

Donald E. Knuth

Sztuka
programowania

Tom 1
Algorytmy podstawowe

Z angielskiego przełożył
Grzegorz Jakacki

OD REDAKCJI

Udostępniane polskim Czytelnikom wielkie dzieło Donalda E. Knutha *Sztuka programowania* składa się obecnie z trzech tomów. W przedmowie do tomu 1 Autor wspomina o tomach 4–7, które nie zostały jeszcze opublikowane. Czytelnik nie powinien się więc dziwić, gdy w treści znajdzie odwołania do zagadnień zawartych w planowanych tomach.

Autor zamieścił w swoim dziele wiele cytatów literackich. Niektóre z nich są mottami rozdziałów, a inne stanowią podsumowanie treści zawartych w podrozdziałach lub mniejszych partiach tekstu. Część z nich byliśmy zmuszeni pozostawić w wersji oryginalnej. Przekład na język polski nie oddawałby bowiem gry słów wyrażającej myśli Autora. Podobnie postąpiliśmy z cytatami nieangielskimi przytoczonymi przez Autora w oryginalnym brzmieniu. Kilka dzieł, z których pochodzą motta, zostało przełożonych na język polski. Gdy jednak tłumaczenie literackie nie oddawało intencji Autora i nie zawierało słów, na użyciu których Mu zależało, zdecydowaliśmy się przytoczyć nowe tłumaczenie.

Niektóre z oznaczeń matematycznych stosowanych przez Autora różnią się nieco od spotykanych w tradycyjnych książkach matematycznych. Ponieważ zależało nam na tym, by polskie wydanie dzieła było pod każdym względem jak najbardziej zbliżone do oryginału, pozostawiliśmy je bez zmiany.

PRZEDMOWA

Oto książka, o którą prosili nasi czytelnicy w tysiącach dosłownie listów. Przygotowanie jej zajęło nam lata. Wielokrotnie sprawdzaliśmy różne przepisy, by przedstawić w niej tylko to, co najlepsze, najciekawsze i najdoskonalsze. Teraz możemy już powiedzieć i to bez cienia wątpliwości, że jeśli czytelnik będzie postępował zgodnie z podanymi instrukcjami, otrzyma właśnie to, o co chodzi – choćby nie miał żadnego doświadczenia kulinarnego.

— *McCall's Cookbook* (1963)

Proces przygotowywania programu dla komputera cyfrowego jest pociągający nie tylko ze względu na potencjalne korzyści ekonomiczne czy naukowe. Wiążą się z nim również przeżycia estetyczne, podobne do tworzenia poezji lub komponowania muzyki. Niniejsza książka jest pierwszym tomem dzieła mającego na celu przedstawienie Czytelnikowi umiejętności składających się na rzemiosło programistyczne.

Książka *nie* rozpoczyna się od wprowadzenia do programowania. Zakładam, że Czytelnik ma w tej dziedzinie pewne doświadczenie. Wymagania są, prawdę powiedziawszy, niewielkie, choć zrozumienie działania komputera cyfrowego wymaga od początkujących czasu i ćwiczeń. Czytelnik powinien:

- a) Orientować się, jak działa komputer cyfrowy, ale niekoniecznie w sensie elektronicznym, raczej na poziomie wykonywania rozkazów przechowywanych w pamięci.
- b) Umieć wyrażać rozwiązania problemów w sposób na tyle prosty, żeby komputer był w stanie je „zrozumieć”. (Te maszyny nie kierują się zdrowym rozsądkiem; robią dokładnie to, co się im każe. Jest to jeden z najpoważniejszych problemów, z jakimi stykają się początkujący użytkownicy komputerów).
- c) Dysponować pewną wiedzą na temat podstawowych technik obliczeniowych, wykorzystujących pętle (wielokrotnie wykonywane zestawy instrukcji), podprogramy, indeksowanie tablic.
- d) Rozumieć choć trochę żargon komputerowy – czyli wiedzieć, co oznacza „pamięć”, „rejestry”, „bity”, „zmiennopozycyjny”, „przepełnienie”, „oprogramowanie”. Większość słów niezdefiniowanych w tekście jest wyjaśniona w skorowidzu zamieszczonym na końcu każdego tomu.

Te cztery wymogi można zatrzymać w jednym zdaniu: Czytelnik powinien mieć na swoim koncie przynajmniej cztery napisane i przetestowane programy.

Starałem się pisać to kilkutomowe dzieło tak, żeby zaspokoić wiele potrzeb naraz. Przede wszystkim poszczególne tomy można traktować jako prace obejmujące aktualną wiedzę z kilku ważnych dziedzin. Poza tym mogą one służyć jako podręczniki do samokształcenia lub do wykładów uniwersyteckich z informatyki. Z uwagi na ten cel zamieściłem w nich wiele ćwiczeń, w większości z rozwiązaniami. Starałem się także zapełniać strony faktami, a nie ogólnikowymi komentarzami.

Książki te są przeznaczone dla osób, które nie interesują się komputerami jedynie doraźnie. Nie znaczy to bynajmniej, że są adresowane wyłącznie do specjalistów. Tak naprawdę jednym z moich głównych celów było przybliżenie technik programistycznych osobom nie związанныm z informatyką, które nie mogą w pełni wykorzystać wszystkich możliwości komputera, bo nie mają czasu na to, by szukać potrzebnych im informacji w czasopismach specjalistycznych.

Można by powiedzieć, że książki te są poświęcone analizie nienumerycznej. Komputery tradycyjnie wiążą się z rozwiązywaniem problemów numerycznych, takich jak znajdowanie pierwiastków równania, interpolacja, całkowanie itp., ale tym zajmuję się tu tylko pobieżnie. Programowanie numeryczne jest niezwykle ciekawą i szybko rozwijającą się dziedziną, a na jego temat napisano już wiele książek. Od wczesnych lat sześćdziesiątych komputery częściej są jednak używane do rozwiązywania problemów, w których liczby pojawiają się tylko przez przypadek; wykorzystuje się właściwości komputera umożliwiające podejmowanie decyzji, a nie wykonywanie operacji arytmetycznych. W problemach nienumerycznych znajdujemy czasem zastosowanie dla dodawania lub odejmowania, ale już dla mnożenia i dzielenia – rzadko. Z książek niniejszych skorzysta jednak również Czytelnik zainteresowany metodami numerycznymi, ponieważ zaprezentowane techniki nienumeryczne przewijają się także w programach numerycznych.

Wyniki badań w dziedzinie analizy nienumerycznej są rozrzucone po wielu czasopismach specjalistycznych. Podjąłem więc próbę wyodrębnienia z tej obszernej literatury tych najbardziej elementarnych technik, mających zastosowanie w wielu sytuacjach programistycznych. Starałem się także sformować z tych rozwiązań swego rodzaju „teorię”, a również pokazać, jak teorię zastosować do rozmaitych problemów praktycznych.

Oczywiście „analiza nienumeryczna” jest w wypadku tej dziedziny wiedzy nazwą pejoratywną. O wiele lepsze byłoby nie budzące wątpliwości, opisowe określenie tematu. „Przetwarzanie informacji” to termin zbyt obszerny, a „techniki programistyczne” – zbyt wąski. Proponuję więc za nazwę tematu, któremu poświęcone jest to dzieło, przyjąć *analię algorytmów*. Ma ona bowiem oznaczać „teorię własności poszczególnych algorytmów komputerowych”.

Zarys dzieła *Sztuka programowania* przedstawia się następująco:

Tom 1. Algorytmy podstawowe

Rozdział 1. Pojęcia podstawowe

Rozdział 2. Struktury danych

Tom 2. Algorytmy seminumeryczne

- Rozdział 3. Liczby losowe
- Rozdział 4. Arytmetyka

Tom 3. Sortowanie i wyszukiwanie

- Rozdział 5. Sortowanie
- Rozdział 6. Wyszukiwanie

Tom 4. Algorytmy kombinatoryczne

- Rozdział 7. Wyszukiwanie kombinatoryczne
- Rozdział 8. Rekursja

Tom 5. Algorytmy składniowe

- Rozdział 9. Analiza leksykalna
- Rozdział 10. Analiza składniowa

Tom 4 obejmuje tak szeroki zakres materiału, że w istocie składają się nań trzy książki (tom 4A, 4B i 4C). W planach są dwa dodatkowe tomy dotyczące bardziej specjalistycznych zagadnień – tom 6: *Teoria języków* (rozdział 11), i tom 7: *Kompilatory* (rozdział 12).

W 1962 roku zacząłem pisać książkę podzieloną na takie właśnie rozdziały, ale bardzo szybko doszedłem do wniosku, że należy dogłębiście, a nie pobieżnie omówić poszczególne zagadnienia. Objętość powstałego tekstu wskazywała, że materiału zawartego w dowolnym rozdziale wystarcza na jednosemestrальny wykład. Rozsądny więc rozwiązańem było opublikowanie dzieła składającego się z oddzielnich tomów. Wiem, że książka w której jest jeden lub dwa rozdziały, wygląda dziwnie, ale zdecydowałem się zachować oryginalną numerację, co ułatwiło odwoływanie się do fragmentów całego dzieła. W planie mam też krótszą wersję tomów 1–5, mającą służyć za bardziej ogólny przewodnik i/lub podręcznik dla młodszych studentów, w którym nie ma informacji specjalistycznych. Wydanie skrócone będzie miało tę samą numerację rozdziałów, co pełne.

Niniejszy tom może być uznawany za „część wspólną” całego dzieła, z uwagi na fakt, że zawiera podstawowy materiał wykorzystywany w następnych tomach. Tomy 2–5 można czytać niezależnie. Tom 1 nie jest jednak wyłącznie „kluczem” do pozostałych pozycji. Można go używać jako podręcznika *struktur danych* (szczególnie z uwagi na rozdział 2), *matematyki dyskretniej* (podrozdziały 1.1 i 1.2 oraz punkty 1.3.3 i 2.3.4) lub *programowania w języku maszynowym* (podrozdziały 1.3 i 1.4).

Punkt widzenia przyjęty przeze mnie przy pisaniu tych rozdziałów różni się od punktu widzenia innych współczesnych autorów. Ja nie próbuję uczyć Czytelnika, jak ma używać cudzego oprogramowania. Staram się pokazać, jak samemu pisać lepsze programy.

Moim pierwotnym celem było zbliżenie Czytelnika do granic poznania w każdym z podjętych tematów. Bardzo trudno jednak dotrzymać kroku dziedzinie nauki, która przynosi zyski ekonomiczne. Nagły rozwój informatyki uczynił mój cel

nieosiągalnym. Temat stał się bezkresną pstromiastą upstrzoną dziesiątkami tysięcy wyrafinowanych wyników, wypracowanych przez dziesiątki tysięcy zdolnych ludzi z całego świata. Z tego powodu za swój nowy cel przyjąłem skupienie się na technikach „klasycznych”, które najprawdopodobniej pozostaną istotne przez wiele dziesięcioleci, oraz na opisaniu ich najlepiej, jak potrafię. W szczególności próbowałem prześledzić historię każdego tematu i stworzyć stabilne podstawy dalszego postępu. Starałem się stosować zwięzłą terminologię, zgodną z tą aktualnie używaną. Moim celem było opisanie tych wszystkich znanych rozwiązań dotyczących programowania komputerów sekwencyjnych, które odznaczają się jednocześnie pięknem i przejrzystością.

Parę słów muszę powiedzieć o matematycznej zawartości dzieła. Materiał jest ułożony tak, by Czytelnik z przygotowaniem z algebry na poziomie szkoły średniej był w stanie czytać tekst, jeśli pominie fragmenty bardziej najeżone matematyką, a Czytelnik wyrobiony matematycznie poznał wiele ciekawych metod matematycznych związanych z matematyką dyskretną. Przedstawienie materiału na dwóch poziomach okazało się możliwe między innymi dzięki przyporządkowaniu ćwiczeniom ocen trudności; zadania czysto matematyczne są odpowiednio oznaczone. Starałem się również tak opracować rozdziały, by zasadnicze wyniki matematyczne znalazły się przed ich dowodami. Dowody przedstawiłem w formie ćwiczeń (z odpowiedziami na końcu książki) albo zamieściłem na końcu rozdziału.

Czytelnik bardziej zainteresowany programowaniem niż związaną z nim matematyką może przerwać lekturę większości rozdziałów, gdy tylko matematyka zacznie sprawiać mu trudności. Natomiast Czytelnik zainteresowany matematyką najciekawszy materiał znajdzie właśnie tu. W wielu publikacjach matematycznych dotyczących komputerów są poważne nieścisłości. Jednym z zadań, jakie sobie postawiłem przy pisaniu tej książki, było przedstawienie poprawnych sposobów ujęcia matematycznych aspektów programowania. Z zawodu jestem matematykiem. Moim obowiązkiem jest więc dbanie o matematyczną spójność.

Do zrozumienia bez mała całej matematyki zawartej w tej książce powinna wystarczyć podstawowa znajomość analizy matematycznej, gdyż większość faktów dotyczących innych teorii została tu wyprowadzona od podstaw. Czasami muszę jednak posłużyć się pewnymi głębszymi twierdzeniami z zakresu teorii zmiennej zespolonej, rachunku prawdopodobieństwa, teorii liczb i tym podobnych. W takich przypadkach zamieszczam odniesienia do odpowiednich podręczników.

Najtrudniejsza decyzja, jaką musiałem podjąć, przygotowując to dzieło, dotyczyła sposobu prezentacji technik programistycznych. Zalety schematów blokowych (*flow charts*) i nieformalnego opisu kroków algorytmu są dobrze znane; omówienie tego zagadnienia można znaleźć w artykule „Computer-Drawn Flowcharts”, opublikowanym w *ACM Communications*, Vol. 6 (September 1963), s. 555–563. Niemniej do opisu algorytmów potrzebny jest język formalny i precyzyjny. Musiałem zatem decydować, czy posłużyć się językiem algebraicznym, takim jak ALGOL lub FORTRAN, czy językiem maszynowym. Być może wielu współczesnych informatyków uzna wybranie przeze mnie języka maszynowego za

błąd, ale ja przekonałem się, że moja decyzja była ze wszech miar słuszna, a to z następujących powodów:

- a) Język programowania wpływa na sposób pisania programu; istnieje tendencja do stosowania konstrukcji, które są prostsze do wyrażenia w danym języku, zamiast tych, które mają prostszą reprezentację maszynową. Rozumiejąc język maszynowy, programista skłania się ku wykorzystaniu metod o wiele wydajniejszych.
- b) Programy, którymi się zajmujemy, są raczej krótkie (z kilkoma wyjątkami). Zrozumienie ich nie będzie zatem kłopotliwe, jeżeli będziemy dysponować odpowiednim komputerem.
- c) Języki wysokiego poziomu nie nadają się do wykorzystania w badaniu istotnych szczegółów niskopoziomowych, jak współprogramy, generowanie liczb losowych, arytmetyka wysokiej precyzji, a także wielu zagadnień dotyczących efektywnego wykorzystania pamięci.
- d) Osoba poważnie zainteresowana komputerami ma na ogół pewną wiedzę na temat języka maszynowego, gdyż stanowi on zasadniczy element każdego komputera.
- e) Język maszynowy jest i tak potrzebny, ponieważ wiele omawianych programów zapisuje wyniki w postaci języka maszynowego.
- f) Języki algebraiczne wychodzą z mody mniej więcej co pięć lat, a mnie zależy na tym, by uwypuklić te aspekty programowania, które są nieprzemijające.

Przyznaję, że łatwiej pisać programy w językach wysokiego poziomu, a już znacznie łatwiej je uruchamiać. Sam od 1970 roku, tworząc programy, rzadko używałam języków niskiego poziomu. Żyjemy przecież w świecie coraz większych i szybszych komputerów. W wypadku jednak wielu poruszanych przeze mnie problemów najistotniejsza jest sztuka programowania. Weźmy na przykład pewne obliczenia kombinatoryczne, które muszą być powtarzane trylion razy. Na każdej mikrosekundzie, wyciągniętej zewnętrznej pętli, zyskujemy około 11.6 dni obliczeń. Podobnie jest z programem, który ma być wielokrotnie używany codziennie na wielu komputerach. Warto zrobić wszystko, by napisać go dobrze, skoro robi się to tylko raz.

Z decyzją o posłużeniu się językiem maszynowym wiązało się pytanie: którym konkretnie? Mogłem wybrać język maszyny X , ale ci, którzy nią nie dysponują, pomyśleliby, że to książka tylko dla znawców X . Co więcej, maszyna X miałaby prawdopodobnie wiele szczegółów, które, choć całkowicie nieistotne z punktu widzenia tej książki, musiałyby zostać wyjaśnione. A na domiar złego po dwóch latach producent maszyny X wpuściłby na rynek maszynę $X+1$ albo maszynę $10X$, a maszyna X przestałaby kogokolwiek interesować.

Aby uniknąć takich rozterek, spróbowałem zaprojektować „idealny” komputer wyposażony w przejrzysty zestaw operacji (dający się opanować, powiedzmy, w godzinę) i bardzo podobny do istniejących maszyn. Nie ma powodu, dla którego student miałby się ograniczać do poznania szczegółów tylko jednego komputera; nauczenie się jednego języka maszynowego ułatwia przyswojenie

następnego. W pracy zawodowej programista może się zetknąć z wieloma różnymi językami maszynowymi. Jedyną wadą fikcyjnej maszyny jest kłopot z uruchamianiem przeznaczonych dla niej programów. Na szczęście nie jest to istotnym problemem, gdyż wielu ochotników podjęło się zadania napisania symulatora mojego komputera. Takie symulatory świetnie nadają się do celów szkoleniowych, gdyż łatwiej się nimi posługiwać niż prawdziwymi komputerami.

Starałem się przytaczać najlepsze wczesne publikacje z każdej omawianej dziedziny, ale zwracać też uwagę na prace najnowsze. Odwołując się do literatury, używam standardowych skrótów nazw czasopism z wyjątkiem tych najczęściej cytowanych, to znaczy:

CACM = Communications of the Association for Computing Machinery

JACM = Journal of the Association for Computing Machinery

Comp. J. = The Computer Journal (British Computer Society)

Math. Comp. = Mathematics of Computation

AMM = American Mathematical Monthly

SICOMP = SIAM Journal on Computing

FOCS = IEEE Symposium on Foundations of Computer Science

SODA = ACM-SIAM Symposium on Discrete Algorithms

STOC = ACM Symposium on Theory of Computing

Crelle = Journal für die reine und angewandte Mathematik

Przykładowo, zapis „*CACM* 6 (1963), 555–563” oznacza odniesienie zamieszczone kilka akapitów wcześniej. Posługuję się także skrótem „CMath” na oznaczenie książki *Matematyka konkretna*, cytowanej we wstępie do podrozdziału 1.2.

Wiele prezentowanych problemów technicznych zawarłem w ćwiczeniach. Tam, gdzie pomysł niebanalnego ćwiczenia nie pochodzi ode mnie, starałem się uhonorować jego autora. Odpowiednie odwołania do literatury występują zazwyczaj w punkcie zawierającym ćwiczenie lub przy odpowiedzi do ćwiczenia. Niestety, w wielu wypadkach ćwiczenia opierają się na materiałach nieopublikowanych, do których nie sposób się odwołać.

W ciągu wielu lat pracy nad tym dziełem całe rzesze osób udzielały mi wsparcia, za co jestem im wszystkim niezmiernie wdzięczny. Wyrazy uznania należą się przede wszystkim mojej żonie Jill za jej niewyczerpaną cierpliwość, za przygotowanie wielu ilustracji oraz za wszelką okazywaną mi pomoc, a także Robertowi W. Floydowi, który w ciągu lat sześćdziesiątych poświęcił mnóstwo czasu na ulepszanie tego tekstu. Tysiące innych osób wniosło również swój znaczący wkład – lista ich nazwisk zapełniłaby kolejny tom! Wiele z nich zezwoliło mi na wykorzystanie swoich dotychczas niepublikowanych prac. Moje badania w Caltech i Stanford były przez wiele lat hojnie finansowane przez National Science Foundation oraz Office of Naval Research. Wydawnictwo Addison–Wesley udzieliło mi nieocenionego wsparcia i pomocy od 1962 roku, kiedy podjąłem się tego trudnego przedsięwzięcia. Najlepszym znanim mi sposobem podziękowania tym wszystkim ludziom jest wykazanie, że ich wkład pracy nie poszedł na marne, że dzięki nim napisałem takie książki, jakich się po mnie spodziewali.

Przedmowa do wydania trzeciego

Dziesięć lat spędzonych nad systemami **TeX** i **METAFONT** pozwoliło mi spełnić marzenie, które przyświecało mi od początku tej pracy. Chciałem za pomocą **TeX-a** i **METAFONT-a** wykonać skład dzieła *Sztuka programowania*. Doczekałem dnia, w którym cały tekst został zamknięty w moim komputerze, zapisany w postaci elektronicznej zapewniającej łatwe dostosowanie go do przyszłych zmian w technice druku. Nowy układ umożliwił wniesienie dosłownie tysięcy poprawek, na wprowadzenie których czekałem bardzo długo.

W tym nowym wydaniu przejrzałem każde słowo tekstu, starając się zachować młodzieżny styl pierwotnych sformułowań, tu i ówdzie dodając czasem kilka dojrzałych sądów. Dołożyłem masę nowych ćwiczeń; do mnóstwa starych podopisywałem nowe, poprawione odpowiedzi.

 Rozbudowa dzieła *Sztuka programowania* trwa! Z tego powodu niektóre części są oznaczone znakiem „Uwaga, prace budowlane!” ostrzegającym, że tekst nie został zaktualizowany. Moje archiwum puchnie od materiałów, które zamierzam zamieścić w ostatecznym, wspaniałym, czwartym wydaniu tomu 1, być może za 15 lat; najpierw muszę jednak skończyć tomy 4 i 5, a nie chciałbym odwlekać opublikowania ich dłużej niż to konieczne.

Większość czarnej roboty związanej z przygotowaniem nowego wydania wykonali Phyllis Winkler i Silvio Levy, którzy fachowo wpisali i zredagowali tekst wydania drugiego, oraz Jeffrey Oldham, który przekonwertował prawie wszystkie oryginalne ilustracje na format programu **METAPOST**. Poprawiłem wszystkie błędy, które uważni Czytelnicy znaleźli w wydaniu drugim (jak również błędy, których, mimo wszystko, nikt nie znalazł). Starałem się uniknąć błędów w nowych partiach materiału. Spodziewam się jednak, że jakieś usterki pozostały i chciałbym jak najszybciej je poprawić. Z tego powodu bez żalu wypłacie 2.56 USD każdemu pierwszemu znalazcę dowolnego błędu technicznego, typograficznego lub historycznego*. Listę wszystkich zgłaszanych mi na bieżąco poprawek można znaleźć w witrynie WWW, wymienionej na stronie vi niniejszego tomu.

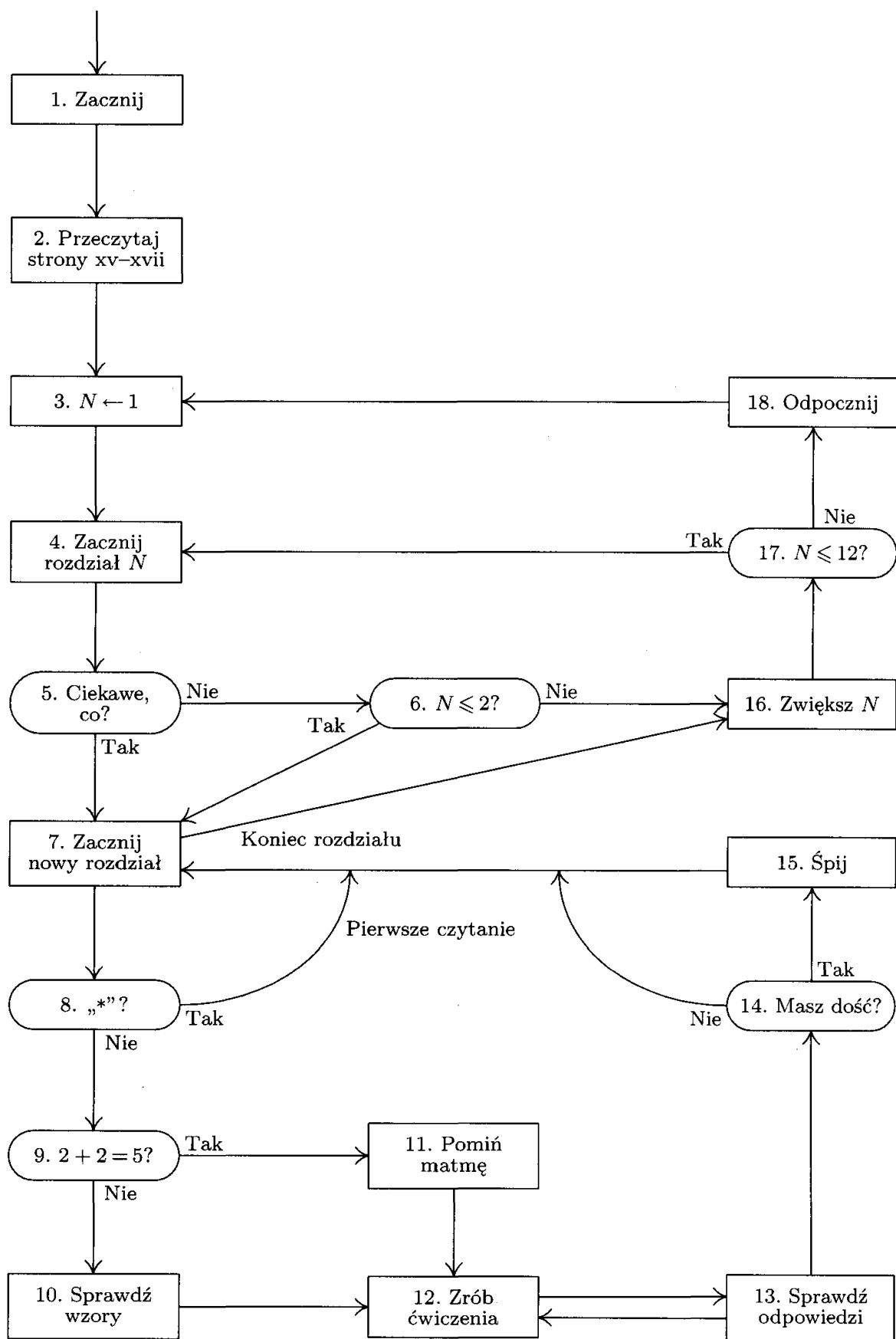
*Stanford, Kalifornia
Kwiecień 1997*

D. E. K.

Wiele się zmieniło przez ostatnich dwadzieścia lat.

— BILL GATES (1995)

* Dotyczy to niestety tylko angielskiego oryginału (przyp. tłum.).



Schemat blokowy procedury czytania dzieła *Sztuka programowania*.

Procedura czytania tego dzieła

1. Zaczniń czytać ten opis, o ile jeszcze tego nie zrobiłeś. *Wykonuj kolejno wszystkie podane tu kroki.* (Ogólna postać tej procedury i schematu blokowego będzie wykorzystywana w całej książce).
2. Przeczytaj uwagi do ćwiczeń zamieszczone na stronach xv – xvii.
3. Przypisz 1 do N .
4. Zaczniń czytać rozdział N . *Nie* czytaj cytatów na początku rozdziału.
5. Czy temat rozdziału Cię zaciekał? Jeśli tak, przejdź do kroku 7; jeśli nie, przejdź do kroku 6.
6. Czy $N \leq 2$? Jeśli nie, przejdź do kroku 16; jeśli tak, mimo wszystko przejrzyj rozdział. (Rozdziały 1 i 2 zawierają istotny materiał wprowadzający oraz przegląd technik programistycznych. Musisz przynajmniej przejrzeć rozdział dotyczące notacji i maszyny MIX).
7. Zaczniń czytać kolejny podrozdział; jeśli w rozdziale nie ma więcej podrozdziałów, to przejdź do kroku 16.
8. Czy podrozdział jest oznaczony „*”? Jeśli tak, to pomiń go przy pierwszym czytaniu (taki podrozdział zawiera materiał specjalistyczny, który jest ciekawy, ale nie niezbędny); wróć do kroku 7.
9. Czy masz zacięcie matematyczne? Jeśli matematyka jest dla Ciebie czarną magią, to przejdź do kroku 11; w przeciwnym razie idź do kroku 10.
10. Sprawdź wyprowadzenia wzorów w podrozdziale (błędy zgłoś Autorowi). Przejdź do kroku 12.
11. Jeżeli bieżący podrozdział jest najeżony obliczeniami matematycznymi, nie czytaj wyprowadzeń. Zapoznaj się mimo wszystko z podstawowymi wynikami przedstawionymi w podrozdziale; są zazwyczaj zamieszczone gdzieś na początku lub wydrukowane *pismem pochyłym* na samym końcu trudnych fragmentów.
12. Wykonaj polecane ćwiczenia zgodnie ze wskazówkami zawartymi w części „Uwagi do ćwiczeń” (które przeczytałeś, wykonując krok 2).
13. Gdy już naprawczysz się nad ćwiczeniami, sprawdź swoje wyniki z odpowiedziami zamieszczonymi na końcu książki (jeśli oczywiście jest tam odpowiedź

do danego ćwiczenia). Przeczytaj także odpowiedzi do ćwiczeń, na które zabrakło Ci czasu. *Uwaga:* W większości wypadków warto przeczytać odpowiedzi do ćwiczenia n przed przystąpieniem do rozwiązywania ćwiczenia $n+1$. Z tego powodu kroki 12–13 są zazwyczaj wykonywane naprzemiennie.

14. Odczuwasz zmęczenie? Jeśli nie, wróć do kroku 7.
15. Idź spać. Jak się obudzisz, wróć do kroku 7.
16. Zwiększ N o jeden. Jeśli $N = 3, 5, 7, 9, 11$ lub 12, rozpoczęj czytanie kolejnego tomu.
17. Jeżeli N jest mniejsze lub równe 12, wróć do kroku 4.
18. Gratulacje. Teraz przekonaj znajomych, by kupili egzemplarz tomu 1 i zaczęli go czytać, a Ty wróć do kroku 3.

Biada temu, kto przeczytał tylko jedną książkę.

— GEORGE HERBERT, *Jacula Prudentum*, 1144 (1640)

*Le défaut unique de tous les ouvrages
c'est d'être trop longs.*

— VAUVENARGUES, *Réflexions*, 628 (1746)

Treść książek to banał. Życie samo w sobie jest wspaniałe.

— THOMAS CARLYLE, *Journal* (1839)

UWAGI DO ĆWICZEŃ

Ćwiczenia zawarte w tym dziele zostały opracowane z myślą o Czytelnikach samokształcących się lub biorących udział w zajęciach grupowych. Jest rzeczą niezmiernie trudną, o ile w ogóle możlią, by nauczyć się czegoś, wyłącznie o tym czytając – nie sprawdzając nabytej wiedzy na konkretnych problemach, a co za tym idzie, nie będąc zmuszonym do myślenia o tym, co się przeczytało. Poza tym najlepiej uczymy się tego, co sami dla siebie odkrywamy. Z tych powodów ćwiczenia stanowią znaczną część książki. Autor dołożył wszelkich starań, by zatrzymać w nich jak najwięcej informacji i wybrać problemy ciekawe, a zarazem kształcące.

W wielu książkach proste ćwiczenia są wymieszane z bardzo trudnymi. Jest to zła praktyka, gdyż Czytelnik chce z góry wiedzieć, ile czasu zajmie mu rozwiązywanie danego problemu, a przy braku tej informacji może zdecydować się na pominięcie wszystkich ćwiczeń. Klasycznym przykładem takiej książki jest *Dynamic Programming* Richarda Bellmana. To ważna, pionierska praca, w której zadania są umieszczone pod wspólnym tytułem „Ćwiczenia i problemy badawcze” na końcu niektórych rozdziałów. Skrajnie proste pytania występują w tej książce miedzy trudnymi, nieroziwiązanymi problemami. Plotka głosi, że dr Bellman zapytany kiedyś, jak odróżnić ćwiczenia od problemów badawczych, odpowiedział: „To, co dasz radę rozwiązać, jest ćwiczeniem, a co nie – problemem badawczym”.

Można znaleźć dobre argumenty za zamieszczeniem w książce zarówno ćwiczeń, jak i problemów badawczych. Aby zatem uwolnić Czytelnika od konieczności rozstrzygania, które są którymi, autor wprowadził *oceny punktowe* wskazujące skalę trudności. Oceny mają następujące znaczenie:

Ocena Interpretacja

- 00 Skrajnie proste ćwiczenie, które można rozwiązać natychmiast, jeżeli zrozumiałe się tekstu. Zazwyczaj takie ćwiczenie można zrobić w pamięci.
- 10 Prosty problem, który zmusza do przemyślenia przeczytanego tekstu. Czytelnik powinien poradzić sobie z rozwiązaniem takiego ćwiczenia w najgorszym razie w ciągu minuty. Przyda się kartka i ołówek.
- 20 Przeciętny problem, umożliwiający sprawdzenie podstawowego opanowania materiału. Rozwiązanie go zajmie od piętnastu do dwudziestu minut.
- 30 Problem o umiarkowanym stopniu trudności i/lub złożoności. Rozwiązanie go może zająć ponad dwie godziny. Przy włączonym telewizorze nawet więcej.

- 40 Dość trudny lub pracochłonny problem, nadający się na kolokwium semestralne. Student powinien sobie z nim poradzić w rozsądny czasie, ale rozwiązanie jest nietrywialne.
- 50 Problem badawczy, którego według informacji autora nikt zadowalająco nie rozwiązał, choć próbowało wielu. Czytelnik, który znajdzie rozwiązanie, powinien je opublikować. Ponadto autor będzie wdzięczny za jak najszybsze poinformowanie go o rozwiązaniu (o ile jest poprawne).

Inne oceny są wyznaczane przez interpolację powyższej „logarytmicznej” skali. Ocena 17 wskazuje na przykład ćwiczenie, które jest nieco prostsze od przeciętnego. Problemy z oceną 50, które zostały ostatnio rozwiązane przez Czytelników, mogą w następnych wydaniach (oraz w erracie publikowanej w Internecie, zobacz strona vi) pojawić się już z oceną 45.

Reszta z dzielenia oceny punktowej przez 5 oznacza ilość szczegółowej pracy do wykonania. Stąd rozwiązywanie ćwiczenia ocenionego na 24 może zająć więcej czasu niż ćwiczenia ocenionego na 25, ale to drugie będzie wymagało więcej pomysłów.

Autor starał się właściwie dobrać oceny, ale osobie, która wymyśla zadanie, trudno stwierdzić, jak pracochłonne będzie znalezienie rozwiązania. Przy tym różnym osobom rozwiązywanie niektórych typów problemów przychodzi łatwiej, a innych trudniej. Pozostaje mieć nadzieję, że oceny punktowe zadowalająco przybliżają poziom trudności. Należy je jednak traktować raczej jako ogólne wskazówki co do stopnia trudności niż dokładne miary trudności.

Książka ta została napisana dla Czytelników o różnym stopniu matematycznej biegłości i wyrafinowania. Skutkiem tego niektóre ćwiczenia są przeznaczone dla osób o zacięciu matematycznym. Ocena punktowa trudności ćwiczenia jest poprzedzona literą *M*, jeżeli dotyczy ono matematyki bardziej zaawansowanej niż potrzebna do pisania programów. Ćwiczenie jest oznaczone literami *HM*, jeżeli jego rozwiązanie wymaga znajomości analizy matematycznej lub matematyki wyższej nie przedstawionej w książce. Oznaczenie *HM* nie musi świadczyć o tym, że ćwiczenie jest trudne.

Niektóre ćwiczenia są poprzedzone strzałką „►”; są one wyjątkowo pouczające i wyjątkowo polecane. Oczywiście po żadnym studencie/czytelniku nie należy się spodziewać, że rozwiąże wszystkie ćwiczenia, dlatego też zostały wyróżnione te najcenniejsze. (Nie oznacza to jednak, że pozostałymi nie warto się zajmować!) Każdy Czytelnik powinien przynajmniej spróbować rozwiązać wszystkie ćwiczenia z oceną 10 lub niższą; strzałki są przy tych z trudniejszych problemów, za które należałyby się zabrać w pierwszej kolejności.

Rozwiązania większości ćwiczeń są zamieszczone w oddzielnej części zatytułowanej „Odpowiedzi do ćwiczeń”. Należy korzystać z nich mądrze, to znaczy nie podglądać odpowiedzi, dopóki naprawdę nie spróbuje się rozwiązać zadania samodzielnie, i nie udawać, że nie ma się czasu. Odpowiedź może się okazać pouczająca i pomocna, ale dopiero po samodzielnym rozwiązaniu ćwiczenia lub po uczciwej próbie zrobienia tego. Podane rozwiązanie jest zazwyczaj krótkie, a jego szczegóły są omówione raczej побieżnie. Autor zakłada bowiem, że Czy-

telnik próbował samodzielnie zrobić ćwiczenie. Czasami rozwiązanie nie stanowi pełnej odpowiedzi na postawione pytanie, a czasami zawiera aż nadto informacji. Całkiem możliwe, że Czytelnik znajdzie lepsze rozwiązanie od tego opublikowanego w książce albo znajdzie w rozwiązaniu błęd. W takim wypadku autor będzie wdzięczny za podanie wszystkich szczegółów. Kolejne wydania książki będą zawierały poprawione rozwiązania opatrzone nazwiskami pogromców błędów.

Przy rozwiązywaniu danego ćwiczenia można posługiwać się rozwiązaniami poprzednich ćwiczeń, chyba że tekst wyraźnie tego zabrania. Ponieważ autor uwzględnił taką sytuację przy doborze ocen punktowych, może się okazać, że ćwiczenie $n + 1$ ma niższą ocenę niż ćwiczenie n , choć rozwiązanie ćwiczenia n wynika z ogólniejszego wyniku uzyskanego w ćwiczeniu $n + 1$.

Zestawienie oznaczeń:

► Polecane

M Dla zainteresowanych matematyką

HM Wymaga znajomości matematyki wyższej

00	Trywialne
10	Proste (minuta)
20	Średnie (kwadrans)
30	Umiarkowanie trudne
40	Na egzamin
50	Problem badawczy

ĆWICZENIA

- 1. [00] Co oznacza ocena „M20”?
- 2. [10] Co mogą dać Czytelnikowi ćwiczenia?
- 3. [14] Udowodnij, że $13^3 = 2197$. Uogólnij odpowiedź. [To jest przykład okropnego rodzaju zadania, którego autor starał się unikać].
- 4. [HM45] Udowodnij, że dla całkowitych n , $n > 2$, równanie $x^n + y^n = z^n$ nie ma rozwiązań w dodatnich liczbach całkowitych x, y, z .

Ale przynajmniej możemy stawić czoło naszej tajemnicy.

Możemy metodycznie i w jakimś porządku posegregować znane nam fakty.

— AGATHA CHRISTIE, *Morderstwo w Orient Expressie*
[Wypowiedź Herkulesa Poirot. Przekład Anna Wiśniewska-Walczyk,
Wydawnictwo Dolnośląskie, Wrocław 1991 (przyp. red.)].

SPIS TREŚCI

Rozdział 1 — Pojęcia podstawowe	1
1.1. Algorytmy	1
1.2. Podstawy matematyczne	11
1.2.1. Indukcja matematyczna	12
1.2.2. Liczby, potęgi i logarytmy	22
1.2.3. Sumy i iloczyny	28
1.2.4. Funkcje całkowitoliczbowe i podstawy teorii liczb	41
1.2.5. Permutacje i silnia	47
1.2.6. Współczynniki dwumianowe	55
1.2.7. Liczby harmoniczne	78
1.2.8. Liczby Fibonacciego	82
1.2.9. Funkcje tworzące	90
1.2.10. Analiza algorytmu	99
*1.2.11. Reprezentacje asymptotyczne	110
*1.2.11.1. Notacja wielkie-O	111
*1.2.11.2. Wzór sumacyjny Eulera	115
*1.2.11.3. Kilka obliczeń asymptotycznych	120
1.3. MIX	128
1.3.1. Opis maszyny MIX	128
1.3.2. Asembler komputera MIX	149
1.3.3. Zastosowania – permutacje	169
1.4. Podstawowe metody programistyczne	192
1.4.1. Podprogramy	192
1.4.2. Współprogramy	200
1.4.3. Interpretery	207
1.4.3.1. Symulator maszyny MIX	209
*1.4.3.2. Śledzenie	218
1.4.4. Wejście i wyjście	221
1.4.5. Historia i bibliografia	236
Rozdział 2 — Struktury danych	240
2.1. Wstęp	240
2.2. Listy liniowe	246
2.2.1. Stosy, kolejki i kolejki dwustronne	246
2.2.2. Tablice	252
2.2.3. Struktury z dowiązaniami	263

2.2.4. Listy cykliczne	283
2.2.5. Listy dwukierunkowe	290
2.2.6. Tablice i listy ortogonalne	309
2.3. Drzewa	320
2.3.1. Przechodzenie drzew binarnych	330
2.3.2. Reprezentacja drzew za pomocą drzew binarnych	347
2.3.3. Inne reprezentacje drzew	361
2.3.4. Podstawowe własności matematyczne drzew	376
2.3.4.1. Drzewa wolne	376
2.3.4.2. Drzewa zorientowane	386
*2.3.4.3. Lemat o drzewach nieskończonych	397
*2.3.4.4. Zliczanie drzew	401
2.3.4.5. Długość ścieżki	415
*2.3.4.6. Historia i bibliografia	422
2.3.5. Listy i odśmiecanie	424
2.4. Struktury z wieloma dowiązaniami	441
2.5. Dynamiczne przydzielanie pamięci	453
2.6. Historia i bibliografia	476
Odpowiedzi do ćwiczeń	485
Dodatek A — Tabele wielkości numerycznych	647
1. Podstawowe stałe (dziesiętnie)	647
2. Podstawowe stałe (ósemkowo)	648
3. Liczby harmoniczne, liczby Bernoulliego, liczby Fibonacciego	649
Dodatek B — Wykaz oznaczeń	651
Skorowidz ze słownikiem	656

ROZDZIAŁ PIERWSZY

POJĘCIA PODSTAWOWE

Wiele osób nie znających nauk matematycznych sądzi, że skoro zadaniem maszyny analitycznej Babbage'a jest podawanie wyników w notacji liczbowej, jej obliczenia muszą być natury arytmetycznej i numerycznej, a nie algebraicznej i analitycznej.

Takie rozumowanie jest błędne.

Maszyna może zestawiać i łączyć wielkości liczbowe tak, jakby były literami lub innymi znakami. Tak naprawdę może przedstawiać wyniki w notacji algebraicznej z zachowaniem wszystkich przyjętych reguł.

— AUGUSTA ADA, Hrabina Lovelace (1844)

*Ćwicz się, na mity Bóg, w małych rzeczach,
a potem przechodź do większych.*

— EPIKTET (*Diatryby IV.i*)

1.1. ALGORYTMY

W dziedzinie programowania komputerów *algorytm* to rzecz podstawowa. Zaczniemy od uważnej analizy tego pojęcia.

Słowo „algorytm” jest interesujące samo w sobie. Na pierwszy rzut oka wydaje się, że ktoś chciał napisać „logarytm”, ale pomieszały mu się cztery pierwsze litery. Słowo nie występowało w słownikach do końca lat pięćdziesiątych. Znajdziemy co najwyżej słowo „algorism” i jego archaiczne znaczenie: wykonywanie działań arytmetycznych przy użyciu liczb arabskich. W średniowieczu istniały dwie kategorie rachmistrzów: *abacist* i *algorist*. *Abacist* to ten, który do obliczeń wykorzystywał *abacus*, natomiast *algorist* liczył, wykorzystując *algorism*. Do czasów Renesansu pierwotne znaczenie tego słowa poszło w zapomnienie. Pierwsi lingwiści próbowali wywieść je z połączeń w stylu: *algiros* [bolesny] + *arithmos* [liczba]; inni twierdzili, że słowo wzięło się od „Króla Algora z Kastylii”. Ostatecznie historycy matematyki odnaleźli prawdziwe pochodzenie słowa „algorism”: pochodzi ono od imienia autora sławnego perskiego podręcznika, które brzmi Abū ‘Abd Allāh Muḥammad ibn Mūsā al-Khwārizmī (około 825 roku) – dosłownie „Ojciec Abdullaha, Mohammed, syn Mojzesza, pochodzący z Khwārizm”. Morze Aralskie w Azji Środkowej było niegdyś znane jako Jezioro Khwārizm, a region Khwārizm znajduje się w dorzeczu Amu-Darii na południe od morza. Al-Khwārizmī napisał znaną książkę *Kitāb al-jabr wa'l-muqābala* („Zasady redukcji i przenoszenia”). Od tytułu tej książki, będącej systematycznym studium na temat rozwiązywania równań liniowych i kwadratowych, pochodzi

inne słowo – „algebra”. [Więcej o życiu i pracach al-Khwārizmiego można zna- leźć w: H. Zemanek, *Lecture Notes in Computer Science* **122** (1981), 1–81].

Stopniowo forma i znaczenie słowa *algorism* zmieniały się. Jak jest to wyjaśnione w *Oxford English Dictionary*, słowo „przeszło wiele pseudo-etymologicznych wynaturzeń, łącznie z ostatnim *algorithm*, które jest mylnie brane” za grecki rdzeń słowa *arytmetyka*. Ta zmiana z *algorism* na *algorithm* nie dziwi, jeśli wziąć pod uwagę, że zapomniano, skąd słowo to pochodzi. Wczesny niemiecki słownik matematyczny, *Vollständiges mathematisches Lexicon* (Leipzig: 1747), podaje następującą definicję słowa *Algorithmus*: „pod tym pojęciem kryją się cztery typy obliczeń arytmetycznych, konkretnie dodawanie, mnożenie, odejmowanie i dzielenie”. Łacińskie określenie *algorithmus infinitesimalis* było w tym czasie używane do nazywania „sposobów obliczania z użyciem nieskończonie małych wartości, jak te wymyślone przez Leibnitza”.

Do 1950 roku słowo „algorytm” było najczęściej kojarzone z algorytmem Euklidesa, tj. procesem znajdowania największego wspólnego dzielnika dwóch liczb, opisanym w *Elementach Euklidesa* (Księga 7, twierdzenia 1 i 2). Pouczające będzie przedstawienie tego algorytmu:

Algorytm E (Algorytm Euklidesa). Dane są dwie dodatnie liczby całkowite m i n , należy znaleźć ich największy wspólny dzielnik, tj. największą dodatnią liczbę całkowitą, która dzieli całkowicie zarówno m , jak i n .

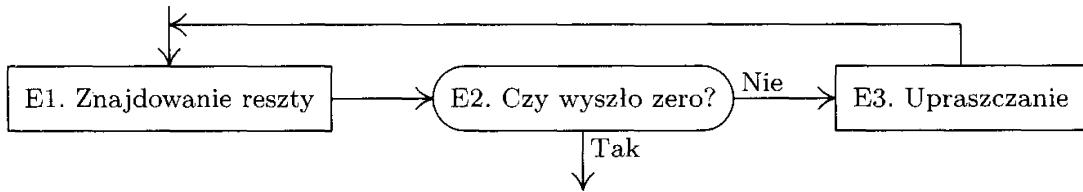
- E1.** [Znajdowanie reszty] Podziel m przez n i niech r oznacza resztę z tego dzielenia. (Mamy $0 \leq r < n$).
- E2.** [Czy wyszło zero?] Jeśli $r = 0$, zakończ algorytm; odpowiedzią jest n .
- E3.** [Uproszczanie] Wykonaj $m \leftarrow n$, $n \leftarrow r$ i wróć do kroku E1. ■

Oczywiście Euklides nie przedstawił swojego algorytmu w taki sposób. Powyższy przykład pokazuje, jak będą prezentowane algorytmy w naszej książce.

Każdemu algorytmowi, który będziemy rozważać, w celu identyfikacji przypisujemy literę (tutaj E), a kroki algorytmu oznaczamy kolejnymi liczbami, poprzedzonymi tą literą (E1, E2, E3). Rozdziały są podzielone na numeryowane podrozdziały. W ramach jednego podrozdziału algorytmy są oznaczane wyłącznie literami, ale gdy mówimy o algorytmach z innych podrozdziałów, do litery oznaczającej algorytm dodajemy numer podrozdziału. Teraz na przykład znajdujemy się w podrozdziale 1.1; w ramach tego podrozdziału algorytm Euklidesa jest nazywany algorytmem E, ale w kolejnych rozdziałach nazywamy go 1.1E.

Opis każdego kroku algorytmu, na przykład kroku E1, zaczyna się krótką frazą w nawiasach kwadratowych, która w zwięzlej formie opisuje krok. Zazwyczaj tekst znajduje się także na towarzyszących opisowi *schematach blokowych*, jak na rysunku 1, dzięki czemu Czytelnikowi łatwiej wyobrazić sobie działanie algorytmu.

Po podsumowaniu pojawia się słowny i symboliczny opis wykonywanych *akcji* oraz podejmowanych decyzji. Czasami występują tu także ujęte w nawiasy *komentarze* (drugie zdanie w opisie kroku E1). Komentarze zawierają



Rys. 1. Schemat blokowy algorytmu E.

wyjaśnienia dotyczące konkretnego kroku, na przykład wskazują na niezmiennicze własności zmiennych lub zarysowują cele wykonywanych akcji. Komentarze nie definiują akcji należących do algorytmu. Mają służyć jako ułatwienie dla Czytelnika.

Strzałka „ \leftarrow ” w kroku E3 jest niezmiernie ważnym symbolem oznaczającym operację *przypisania*, zwaną także operacją *podstawienia* lub *zastąpienia*: „ $m \leftarrow n$ ” oznacza, że aktualna wartość zmiennej n ma zostać przypisana do zmiennej m . W chwili rozpoczęcia algorytmu E wartości m i n są liczbami stanowiącymi dane wejściowe zadania; ale w chwili jego zakończenia te zmienne będą miały być całkiem inne wartości. Strzałki używa się w celu odróżnienia operacji przypisania od relacji równości: Nie powiemy „Niech $m = n$ ”, będziemy natomiast pytać „Czy $m = n?$ ”. Znak „ $=$ ” symbolizuje warunek do sprawdzenia, znak „ \leftarrow ” – akcję do wykonania. Operacja *zwiększenia n o jeden* jest oznaczana przez „ $n \leftarrow n + 1$ ” (czytaj: „do n przypisujemy wartość $n + 1$ ”, albo „ n staje się $n + 1$ ”). W ogólności „zmienna \leftarrow wyrażenie” oznacza, że wyrażenie ma zostać obliczone dla aktualnych wartości zmiennych w nim występujących, a wynik przypisany do zmiennej stojącej po lewej stronie strzałki (stara wartość zmiennej jest gubiona). Osoby nie przyzwyczajone do pracy z komputerem mają czasami skłonność do mówienia „ n staje się $n + 1$ ”, co zapisują „ $n \rightarrow n + 1$ ”, mając na myśli operację zwiększenia n o jeden. Taka symbolika prowadzi wyłącznie do zamieszania i powinno się jej unikać.

Zauważmy, że istotna jest kolejność akcji w kroku E3: „Wykonaj $m \leftarrow n$, $n \leftarrow r$ ” to zupełnie co innego niż „Wykonaj $n \leftarrow r$, $m \leftarrow n$ ”, gdyż w tej drugiej wersji stara wartość n została zniszczona przed przypisaniem jej do m . Druga wersja jest równoważna operacji „Wykonaj $n \leftarrow r$, $m \leftarrow r$ ”. Jeśli wiele zmiennych ma przyjąć tę samą wartość, możemy użyć wielu strzałek, na przykład „ $n \leftarrow r$, $m \leftarrow r$ ” można zapisać jako „ $n \leftarrow m \leftarrow r$ ”. W celu wymiany wartości dwóch zmiennych piszemy „Zamień $m \leftrightarrow n$ ”; tę akcję można także zapisać, korzystając z dodatkowej zmiennej t : „Przypisz $t \leftarrow m$, $m \leftarrow n$, $n \leftarrow t$ ”.

Wykonanie algorytmu rozpoczyna się od kroku o najmniejszym numerze, zazwyczaj 1, i obejmuje kroki o kolejnych numerach, jeżeli nie podano inaczej. W kroku E3 rozkaz „wróć do kroku E1” w oczywisty sposób wyznacza kolejność obliczeń. W kroku E2 akcja jest poprzedzona warunkiem „Jeśli $r = 0$ ”, zatem jeśli $r \neq 0$, to reszta zdania nie ma zastosowania i nie powinna być wykonana żadna akcja. Moglibyśmy dodać nadmiarowe zdanie „Jeśli $r \neq 0$, idź do kroku E3”.

Gruba pionowa kreska „|” pojawiająca się na końcu kroku E3 oznacza koniec algorytmu i dalszy ciąg tekstu.

Omówiliśmy już właściwie wszystkie konwencje używane w niniejszej książce do zapisu algorytmów. Pozostała nam tylko notacja dotycząca indeksowania tablic. Przypuśćmy, że mamy n liczb v_1, v_2, \dots, v_n ; zamiast pisać v_j na oznaczenie j -tej z nich, często używać będziemy notacji $v[j]$. Podobnie $a[i, j]$ będzie preferowaną postacią podwójnego indeksowania a_{ij} . Czasami zmienne będą miały nazwy wieloliterowe, zazwyczaj pisane wielkimi literami. Stąd TEMP może być nazwą zmiennej używanej do tymczasowego przechowywania obliczanej wartości, PRIME[K] może oznaczać K-tą liczbę pierwszą itp.

Wiemy zatem, jak zapisać algorytm. Skupmy się teraz na tym, jak algorytm przeczytać. Na wstępnie trzeba jasno powiedzieć, że algorytmy to nie beletrystyk. Nie należy ich czytać ciurkiem. Z algorytmem jest tak, że jak nie zobaczyś, to nie uwierzysz. Najlepszą metodą poznania algorytmu jest wypróbowanie go. Czytelnik, napotykając w tekście algorytm, powinien sięgnąć po kartkę oraz ołówek i przerobić przykład działania algorytmu. Zazwyczaj w tekście podany jest zarys pracy algorytmu, a jeżeli go nie ma, to znaczy, że Czytelnik z łatwością poradzi sobie sam. Jest to prosty i bezbolesny sposób zrozumienia algorytmu, a wszystkie inne podejścia są generalnie nieskuteczne.

Spróbujmy zatem przerobić przykład działania algorytmu E. Przypuśćmy, że $m = 119$ i $n = 544$; zaczynamy od kroku E1. (Czytelnik powinien teraz śledzić opis algorytmu, jako że będziemy posuwać się krok po kroku). Dzielenie m przez n jest w tym wypadku proste, nawet zbyt proste, gdyż iloraz wynosi zero, a reszta 119. Stąd $r \leftarrow 119$. Przechodzimy do kroku E2, ale ponieważ $r \neq 0$, to nie dzieje się nic. W kroku E3 przypisujemy $m \leftarrow 544, n \leftarrow 119$. Jasne jest, że jeżeli na początku $m < n$, to iloraz w kroku E1 będzie zerem, a algorytm rozpocznie się od wymiany wartości m i n w taki dość skomplikowany sposób. Moglibyśmy dodać nowy krok:

E0. [Zagwarantowanie, że $m \geq n$] Jeżeli $m < n$, to zamień $m \leftrightarrow n$.

Nie wprowadza to istotnej zmiany do algorytmu za wyjątkiem nieznacznego wzrostu długości oraz zmniejszenia czasu wykonania w mniej więcej połowie przypadków.

Znalazłszy się z powrotem w kroku E1, stwierdzimy, że $\frac{544}{119} = 4\frac{68}{119}$, zatem $r \leftarrow 68$. Znów w E2 nie wykonujemy żadnej akcji, a w E3 przypisujemy $m \leftarrow 119, n \leftarrow 68$. W następnym obiegu przypisujemy $r \leftarrow 51$ i $m \leftarrow 68, n \leftarrow 51$. Następnie $r \leftarrow 17$ i $m \leftarrow 51, n \leftarrow 17$. Na koniec, gdy 51 zostanie podzielone przez 17, przypisujemy $r \leftarrow 0$, w związku z czym w kroku E2 wykonanie algorytmu się kończy. Największym wspólnym dzielnikiem 119 i 544 jest 17.

Oto i algorytm. Współczesne znaczenie słowa algorytm jest podobne do znaczenia słów *przepis*, *proces*, *metoda*, *technika*, *procedura*, ale może być też nieco węższe. Słowem algorytm określa się jeszcze skończony zbiór reguł wskazujący kolejność operacji przy rozwiązywaniu problemu pewnego typu. Z tym znaczeniem jest związanych pięć pojęć.

1) Skończoność. Wykonanie algorytmu musi zawsze zatrzymać się po skończonej liczbie kroków. Algorytm E spełnia ten warunek, ponieważ po wykonaniu kroku E1 wartość r jest mniejsza niż n ; stąd jeśli $r \neq 0$, wartość n zmniejsza

się przy każdym wykonaniu kroku E1. Malejący ciąg liczb naturalnych nie może biec w nieskończoność, więc krok E1 jest wykonywany tylko skończoną liczbę razy dla dowolnej początkowej wartości n . Zauważmy jednak, że liczba kroków może być dowolnie duża; przy odpowiednio dobranych liczbach m i n , krok E1 zostanie wykonany ponad milion razy.

(Procedura, która ma wszystkie cechy algorytmu poza, być może, skończonością, nazywana jest *metodą obliczeniową*. Euklides przedstawił nie tylko algorytm znajdowania największego wspólnego dzielnika, ale także bardzo podobną konstrukcję geometryczną na znajdowanie „największej wspólnej miary” długości dwóch odcinków; jest to metoda obliczeniowa, która nie zakończy się, gdy długości są niewspółmierne. Inny przykład nie kończącej się metody obliczeniowej to *proces interakcyjny*, który stale komunikuje się z otoczeniem).

2) Dobre zdefiniowanie. Każdy krok algorytmu musi być opisany precyzyjnie; dla każdego możliwego przypadku akcje muszą być opisane ściśle i jednoznacznie. Autor ma nadzieję, że algorytmy w niniejszej książce są dobrze zdefiniowane. Są one jednak zapisane w języku naturalnym, co stwarza groźbę powstania nieporozumienia. W celu ominięcia podobnych niebezpieczeństw tworzy się specjalne języki programowania, definiowane formalnie i przeznaczone do opisu algorytmów. Każde wyrażenie takiego języka ma precyzyjnie określone znaczenie. Wiele algorytmów w tej książce będzie zapisanych zarówno w języku naturalnym, jak i w języku programowania. Zapis metody obliczeniowej w języku programowania nazywamy *programem*.

Możemy powiedzieć, że algorytm E jest dobrze zdefiniowany, gdy na przykład w kroku E1 Czytelnik dokładnie wie, co to znaczy *podzielić m przez n* i co to jest reszta z dzielenia. Po prawdzie nie ma powszechnej zgody, co to dokładnie oznacza, gdy m i n nie są dodatnimi liczbami całkowitymi; jaka jest reszta z dzielenia -8 przez $-\pi$? Jaka jest reszta z dzielenia $59/13$ przez zero? Z tego powodu kryterium dobrego zdefiniowania wymaga upewnienia się, że w chwili wykonania kroku E1 wartości m i n są zawsze dodatnimi liczbami całkowitymi. Na początku działania algorytmu rzeczywiście tak jest. Po kroku E1 r jest nieujemną liczbą całkowitą, która jednak nie może być zerem, jeżeli dojdziemy do kroku E3. Rzeczywiście, m i n są dodatnimi liczbami całkowitymi.

3) Dane wejściowe. Algorytm ma zero lub więcej danych wejściowych, tj. wartości, które są znane, zanim rozpoczęcie się wykonanie algorytmu, lub dynamicznie podawane w czasie pracy. Dane wejściowe pochodzą z określonych zbiorów. Na przykład algorytm E ma dwie dane wejściowe n i m , pochodzące ze zbioru *dodatnich liczb całkowitych*.

4) Dane wyjściowe. Algorytm generuje jedną lub więcej danych wyjściowych, czyli wartości w określony sposób powiązanych z danymi wejściowymi. Algorytm E w kroku E2 generuje jedną daną wyjściową n , będącą największym wspólnym dzielnikiem danych wejściowych.

(Można łatwo udowodnić, że ta liczba jest naprawdę największym wspólnym dzielnikiem. Po wykonaniu kroku E1 mamy:

$$m = qn + r,$$

dla pewnej liczby całkowitej q . Jeśli $r = 0$, to m jest wielokrotnością n i w takim przypadku widać, że n jest największym wspólnym dzielnikiem m i n . Jeśli $r \neq 0$, to zauważmy, że każda liczba, która dzieli zarówno m , jak i n , musi dzielić także $m - qn = r$, a każda liczba, która dzieli zarówno n , jak i r , musi dzielić $qn + r = m$. Stąd zbiór dzielników liczb $\{m, n\}$ jest taki sam, jak zbiór dzielników liczb $\{n, r\}$. W szczególności *największy wspólny dzielnik* $\{m, n\}$ jest taki sam, jak największy wspólny dzielnik $\{n, r\}$. Krok E3 „nie zmienia” odpowiedzi na pierwotne pytanie).

5) *Efektywne zdefiniowanie.* Algorytm powinien być określony *efektywnie* w tym sensie, że jego operacje powinny być wystarczająco proste, by (przynajmniej teoretycznie) można było je dokładnie i w skończonym czasie wykonać za pomocą ołówka i kartki. Algorytm E korzysta wyłącznie z operacji dzielenia jednej liczby całkowitej przez drugą, sprawdzania, czy liczba całkowita jest zerem, i przypisywania wartości jednych zmiennych do innych. Te operacje są efektywne, ponieważ liczby całkowite mogą być w skończony sposób reprezentowane na papierze i ponieważ istnieje przynajmniej jeden sposób („algorytm dzielenia”) na dzielenie ich przez siebie. Ale te same operacje *nie* byłyby efektywne, jeśli dopuścilibyśmy dowolne wartości rzeczywiste z nieskończonymi rozwinięciami dziesiętnymi, albo na przykład fizyczne długości odcinków prostej (których nie można podać dokładnie). Inny przykład kroku nieefektywnego: „jeśli 4 jest największą liczbą całkowitą n , dla której istnieje rozwiązanie równania $w^n + x^n + y^n = z^n$ w zbiorze dodatnich liczb całkowitych w, x, y, z , to przejdź do kroku E4”. Takie stwierdzenie nie będzie operacją efektywną dopóty, dopóki ktoś nie skonstruuje algorytmu stwierdzającego, czy 4 jest największą liczbą o podanej własności.

Porównajmy algorytm z tradycyjnym przepisem z książki kucharskiej. Przepis ma zazwyczaj własność skończości (choć podobno w garnku się nie zagotuje, jeśli nie odwrócić wzroku), ma dane wejściowe (jajka, mąka itp.), wyjaśnione (na przykład kolacja), ale oczywiście nie jest dobrze zdefiniowany. Często polecenia w przepisie kucharskim są nieprecyzyjne, na przykład „dodać odrobinę soli”. „Odrobina” to „mniej niż 1/8 łyżeczki od herbaty”, sól też jest dobrze określona, ale gdzie tej soli należy dodać? Posypać z wierzchu? Obok? Polecenia takie jak „smażyć na umiarkowanym ogniu na złoty kolor” albo „podgrzać koniak w małym rondlu” są wyjaśnieniami odpowiednimi dla praktykującego kucharza, natomiast algorytm musi być wyspecyfikowany do takiego stopnia, żeby nawet komputer potrafił zastosować się do jego poleceń. Mimo wszystko programista może wiele się nauczyć, studując dobrą książkę kucharską. (Autor poważnie zastanawiał się nad nadaniem niniejszemu tomowi tytułu „Książka kucharska dla programistów”. Być może kiedyś autor przymierzy się do książki „Algorytmy w kuchni”).

Zauważmy, że wymaganie skończości nie jest wystarczająco silne z praktycznego punktu widzenia. Pozytyczny algorytm powinien nie tylko gwarantować zakończenie obliczeń po skończonej liczbie kroków, ale nawet po *ograniczonej*, skończonej, rozsądnej liczbie kroków. Istnieje na przykład algorytm, który stwierdza, czy partia szachów może zawsze zakończyć się wygraną białych, jeżeli

grający nimi nie popełni błędu (zobacz ćwiczenie 2.2.3–28). Ten algorytm służy do rozwiązyania problemu, którym pasjonują się tysiące osób, ale idę o zakład, że nikt z nas do końca życia nie pozna odpowiedzi. Żeby zakończyć działanie, ten algorytm potrzebuje kosmicznie długiego, choć skończonego, czasu. W rozdziale 8 omawiamy pewne skończone liczby, które jednak są tak wielkie, że nie sposób ich sobie wyobrazić.

W praktyce nie zadowalamy się dowolnym algorytmem. Chcemy algorytmów *dobrych*, w sensie pewnej nieściśle określonej estetyki. Jednym z kryteriów jakości algorytmu jest czas potrzebny na jego wykonanie. Można go wyrażać, podając, ile razy został wykonany każdy z kroków algorytmu. Inne kryteria to łatwość dostosowywania algorytmu do różnych komputerów, prostota, elegancja itp.

Często spotykamy wiele różnych algorytmów rozwiązywających ten sam problem, przez co musimy decydować, który z nich jest najlepszy. To prowadzi do bardzo ciekawych zagadnień *analizy algorytmów*: dany jest algorytm, scharakteryzuj jego efektywność.

Rozważmy z tego punktu widzenia algorytm Euklidesa. Przypuśćmy, że postawiono pytanie: „Jeżeli wartość n jest znana, ale m może przybierać dowolną wartość całkowitą dodatnią, to ile wynosi średnia liczba T_n wykonań kroku E1?” Po pierwsze musimy sprawdzić, czy to pytanie ma sens, bo próbujemy liczyć średnią dla nieskończonie wielu wyborów m . Jest oczywiste, że po pierwszym wykonaniu kroku E1 znaczenie ma jedynie reszta z dzielenia m przez n , zatem wystarczy sprawdzić algorytm dla $m = 1, m = 2, \dots, m = n$, policzyć sumaryczną liczbę wykonań kroku E1 i podzielić ją przez n .

Teraz ważnym problemem jest określenie *zachowania* T_n . Czy T_n jest w przybliżeniu równe na przykład $\frac{1}{3}n$, a może \sqrt{n} ? W istocie rzeczy znalezienie odpowiedzi na to pytanie jest niezmiernie trudnym i ciekawym problemem matematycznym, nie do końca jeszcze rozwiązany; zajmiemy się nim szczegółowo w punkcie 4.5.3. Można pokazać, że dla dużych n wartość T_n wynosi w przybliżeniu $(12(\ln 2)/\pi^2) \ln n$, czyli że jest proporcjonalna do *logarytmu naturalnego* liczby n , z dosyć niebanalnym współczynnikiem proporcjonalności. Więcej szczegółów dotyczących algorytmu Euklidesa oraz innych metod obliczania największego wspólnego dzielnika znajduje się w punkcie 4.5.2.

Takie właśnie dociekania upodobał sobie autor nazywać *analizą algorytmów*. W ogólności pomysł polega na rozpatrzeniu konkretnego algorytmu i ilościowym opisaniu jego zachowania. Przy okazji badamy, czy algorytm jest w pewnym sensie optymalny. Odmiennym działem nauki jest *teoria obliczeń*^{*}, która dotyczy badań nad istnieniem algorytmów obliczających konkretne wielkości.

Do tej pory nasze rozważania dotyczące algorytmów były raczej nieprecyzyjne, a Czytelnicy obeznani z matematyką mają podstawy sądzić, że powyższy wstęp jest bardzo chwiejnym fundamentem teorii dotyczących algorytmów. By rozwiązać wątpliwości, zamknijmy niniejszy rozdział krótkim przedstawieniem metody umożliwiającej opisanie algorytmu w języku teorii mnogości. Zdefiniujmy

* Autor posługuje się tu terminem *algorithmics*; w polskiej terminologii przyjmuje się jednak, że *analiza algorytmów* (nauka o wydajności algorytmów) jest działem *algorytmiki* (nauki o algorytmach) (przyp. tłum.).

formalnie *metodę obliczeniową* jako czwórkę (Q, I, Ω, f) , gdzie Q jest zbiorem, I i Ω są jego podzbiorami, a f jest funkcją z Q w Q . Ponadto f powinna być identycznością na Ω , tj. $f(q) = q$ dla dowolnego q ze zbioru Ω . Cztery wartości Q, I, Ω, f mają reprezentować odpowiednio stan obliczeń, dane wejściowe, dane wyjściowe oraz regułę obliczania. Każda dana wejściowa x ze zbioru I w następujący sposób określa *ciąg obliczeń* x_0, x_1, x_2, \dots :

$$x_0 = x \quad \text{i} \quad x_{k+1} = f(x_k) \quad \text{dla } k \geq 0. \quad (1)$$

O ciągu obliczeń mówimy, że *kończy się w k krokach*, jeżeli k jest najmniejszą liczbą całkowitą, dla której x_k należy do Ω . Mówimy wtedy również, że w wyniku obliczenia dla x otrzymaliśmy x_k . (Zauważmy, że jeżeli x_k należy do Ω , to x_{k+1} też, ponieważ w takim przypadku $x_{k+1} = x_k$). Ciągi obliczeń niektórych metod obliczeniowych mogą się nie zakończyć. *Algorytm* jest taką metodą obliczeniową, że dla dowolnego x ze zbioru I jej ciąg obliczeń kończy się w skończonej liczbie kroków.

Na przykład algorytm E może być sformalizowany następująco: niech Q będzie zbiorem wszystkich singletonów (n) , wszystkich par uporządkowanych (m, n) i wszystkich uporządkowanych czwórek $(m, n, r, 1), (m, n, r, 2)$ i $(m, n, p, 3)$, gdzie m, n i p są dodatnimi liczbami całkowitymi, a r jest nieujemną liczbą całkowitą. Niech I będzie podzbiorem Q utworzonym ze wszystkich par (m, n) i niech Ω będzie podzbiorem Q utworzonym ze wszystkich singletonów (n) . Zdefiniujmy f następująco:

$$\begin{aligned} f((n)) &= (n); \\ f((m, n)) &= (m, n, 0, 1); \\ f((m, n, r, 1)) &= (m, n, \text{reszta z dzielenia } m \text{ przez } n, 2); \\ f((m, n, r, 2)) &= (n), \quad \text{gdy } r = 0, \quad (m, n, r, 3) \quad \text{w przeciwnym razie}; \\ f((m, n, p, 3)) &= (n, p, p, 1). \end{aligned} \quad (2)$$

Odpowiedniość między tą notacją a algorytmem E jest oczywista.

Taka formalizacja pojęcia algorytmu nie obejmuje wspomnianego wcześniej wymagania efektywności. Na przykład Q może zawierać ciągi nieskończone, na których nie można wykonać obliczeń przy użyciu ołówka i kartki, funkcja f może też obejmować operacje, których śmiertelnicy nie są w stanie przeprowadzić. Jeżeli chcemy ograniczyć pojęcie algorytmu, by dozwolone były tylko operacje elementarne, możemy nałożyć warunki na Q, I, Ω i f . Niech A będzie skończonym zbiorem liter i niech A^* będzie zbiorem wszystkich napisów nad A (czyli niech A^* będzie zbiorem wszystkich ciągów $x_1 \dots x_n$, gdzie $n \geq 0$ i x_j należy do A dla $1 \leq j \leq n$). Pomysł polega na reprezentowaniu stanów obliczenia za pomocą ciągów elementów zbioru A^* . Niech N będzie nieujemną liczbą całkowitą i niech Q będzie zbiorem wszystkich (σ, j) , gdzie σ należy do A^* , a j jest liczbą całkowitą, $0 \leq j \leq N$. Niech I będzie zbiorem tych par z Q , w których $j = 0$, a Ω tych par, w których $j = N$. Jeżeli θ i σ są napisami z A^* , to mówimy, że θ występuje w σ , jeżeli σ ma postać $\alpha\theta\omega$ dla pewnych napisów α i ω . Musimy jeszcze zdefiniować f : jest ona funkcją określona przez wybór

napisów θ_j, ϕ_j oraz liczb całkowitych a_j, b_j dla $0 \leq j \leq N$ w następujący sposób:

$$\begin{aligned} f(\sigma, j) &= (\sigma, a_j), && \text{jeśli } \theta_j \text{ nie występuje w } \sigma; \\ f(\sigma, j) &= (\alpha\phi_j\omega, b_j), && \text{jeśli } \alpha \text{ jest najkrótszym napisem,} \\ &&& \text{dla którego } \sigma = \alpha\theta_j\omega; \\ f(\sigma, N) &= (\sigma, N). \end{aligned} \quad (3)$$

Tak zdefiniowana metoda obliczeniowa jest efektywna, a doświadczenie pokazuje, że jest także wystarczająco silna, by obliczyć wszystko to, co potrafimy obliczać ręcznie. Istnieje wiele równoważnych sposobów sformułowania pojęcia efektywnej metody obliczeniowej (na przykład przy użyciu maszyn Turinga). Powyższe sformułowanie jest prawie identyczne z podanym przez A. A. Markowa w jego książce *The Theory of Algorithms* [Trudy Mat. Inst. Akad. Nauk **42** (1954), 1–376], poprawionej później i poszerzonej przez N. M. Nagornego (Moskwa: Nauka, 1984; wydanie angielskie, Dordrecht: Kluwer, 1988).

ĆWICZENIA

1. [10] W tekście jest pokazane, w jaki sposób zamienić miejscami wartości zmiennych, przypisując $t \leftarrow m, m \leftarrow n, n \leftarrow t$. Pokaż, w jaki sposób za pomocą ciągu przypisań można przestawić wartości czterech zmiennych (a, b, c, d) , by otrzymać (b, c, d, a) . Innymi słowy, nowa wartość a ma być początkową wartością b itd. Zminimalizuj liczbę przypisań.
 2. [15] Udowodnij, że m jest zawsze większe od n na początku kroku E1, poza być może pierwszym wykonaniem tego kroku.
 3. [20] Zmodyfikuj algorytm E (w celu zwiększenia wydajności), by unikać zbędnych operacji, jak na przykład $m \leftarrow n$. Zapisz ten algorytm w takiej postaci, w jakiej zapisywaliśmy algorytm E, i nazwij go F.
 4. [16] Ile wynosi największy wspólny dzielnik liczb 2166 i 6099?
- ▶ 5. [12] Wykaż, że procedura „Jak czytać «Sztukę programowania»” zamieszczona w przedmowie nie jest prawdziwym algorytmem, gdyż nie spełnia trzech z pięciu podanych kryteriów. Wymień różnice między formami zapisu tej procedury i algorytmu E.
 - ▶ 6. [20] Ile wynosi T_5 , tj. średnia liczba wykonania kroku E1 dla $n = 5$?
 - ▶ 7. [M21] Przypuśćmy, że m jest ustalone, a n przebiega zbiór dodatnich liczb całkowitych; niech U_m będzie średnią liczbą wykonania kroku E1 w algorytmie E. Pokaż, że U_m jest dobrze określone. Czy istnieje jakiś związek między U_m a T_m ?
 - 8. [M25] Skonstruuj „efektywny” formalny algorytm obliczania największego wspólnego dzielnika dwóch dodatnich liczb całkowitych m i n , podając $\theta_j, \phi_j, a_j, b_j$ jak w (3). Niech wejście będzie reprezentowane przez napisy postaci $a^m b^n$, tj. m liter a , po których następuje n liter b . Spróbuj maksymalnie uprościć swoje rozwiązanie. [Wskazówka: Posłuż się algorytmem E, ale dzielenie w kroku E1 zastąp przypisaniem $r \leftarrow |m - n|, n \leftarrow \min(m, n)$].
 - ▶ 9. [M30] Przypuśćmy, że $C_1 = (Q_1, I_1, \Omega_1, f_1)$ i $C_2 = (Q_2, I_2, \Omega_2, f_2)$ są metodami obliczeniowymi. Na przykład C_1 może oznaczać algorytm E, jak w (2), z ograniczeniem na wielkość n i m , a C_2 może oznaczać implementację algorytmu E w postaci programu

komputerowego. (Wówczas Q_2 może być zbiorem wszystkich stanów maszyny, tj. wszystkich możliwych konfiguracji jej pamięci i rejestrów; f_2 może określać pojedynczą akcję maszyny; I_2 może być stanem początkowym, zawierającym zarówno program do wyznaczania największego wspólnego dzielnika, jak i wartości m i n).

Sformułuj teoriomnogościową definicję pojęcia „ C_2 jest reprezentacją C_1 ” albo „ C_2 symuluje C_1 ”. Intuicyjnie ma ono oznaczać, że każdy ciąg obliczenia C_1 może być odwzorowany przez C_2 , z tym że C_2 może wykonać więcej kroków, a w swoim stanie zawierać więcej informacji. (Definiując takie pojęcie, otrzymamy ścisłą interpretację stwierdzenia, że „program X realizuje algorytm Y ”).

1.2. PODSTAWY MATEMATYCZNE

W tym podrozdziale przyjrzymy się notacji matematycznej pojawiającej się w Sztuce Programowania i wyprowadzimy kilka podstawowych, często używanych wzorów. Czytelnicy niezainteresowani skomplikowanymi wyprowadzeniami, powinni przynajmniej zapoznać się ze *znaczeniem* wzorów, by mogli korzystać z przedstawionych wyników.

W tej książce notacji matematycznej używamy z dwóch powodów: za jej pomocą opisujemy algorytmy oraz analizujemy ich wydajność. Jak widać z poprzedniego podrozdziału, notacja używana do opisu algorytmów jest dosyć prosta. Przy analizie wydajności algorytmu musimy jednak korzystać z notacji bardziej wyspecjalizowanej.

Większości omawianych algorytmów towarzyszą obliczenia matematyczne dotyczące spodziewanej szybkości działania. W tych obliczeniach korzystamy z osiągnięć różnych działów matematyki. Systematyczne wyprowadzanie wszystkich używanych metod z pewnością zapełniłoby pokaźny tom. W większości przypadków wystarcza jednak znajomość algebry na poziomie szkoły średniej, a Czytelnik obeznany z podstawami analizy matematycznej będzie w stanie zrozumieć prawie całą zaprezentowaną matematykę. Miejscami musimy uciec się do poważniejszych zagadnień: teorii zmiennej zespolonej, teorii grup, teorii liczb, rachunku prawdopodobieństwa itp. W takich przypadkach będziemy w miarę możliwości starali się wyjaśnić zagadnienie na poziomie elementarnym lub przynajmniej podać odniesienia do innych źródeł informacji.

Techniki matematyczne związane z analizą algorytmów mają swój specyficzny smaczek. Spostrzeżemy na przykład, że często pojawiają się tu skończone sumy liczb rzeczywistych albo związki rekurencyjne. Zagadnienia tego typu są w tradycyjnych kursach matematyki traktowane zazwyczaj bardzo pobieżnie, stąd kolejne rozdziały zostały tak zaplanowane, by nie tylko oswajały Czytelnika z notacją, ale również przybliżały mu użyteczne metody i sposoby wykonywania obliczeń.

Istotna uwaga: chociaż lektura pierwszych rozdziałów stanowi gruntowny trening umiejętności matematycznych, które każdy badacz algorytmów posiadać musi, większość Czytelników z początku nie dostrzeże silnych powiązań zaprezentowanego materiału z programowaniem (z wyjątkiem punktu 1.2.1). Czytelnik może zdecydować się zaufać autorowi i przeczytać uważnie najbliższe rozdziały, wierząc, że zaprezentowane tam zagadnienia są rzeczywiście niezbędne. Wydaje się jednak, że lepiej *przy pierwszym czytaniu jedynie przekartkować tekst*, a przestudiować go dopiero wtedy, gdy w miarę czytania kolejnych rozdziałów przedstawione w nim zagadnienia zaczną się okazywać przydatne. Jeśli Czytelnik poświęci zbyt wiele czasu na studiowanie materiału wprowadzającego, to może nigdy nie dobrnąć do zagadnień związanych z programowaniem! Każdy Czytelnik powinien jednak zaznajomić się z treścią pierwszych rozdziałów i przynajmniej spróbować rozwiązać kilka ćwiczeń, nawet przy pierwszym czytaniu. Zwracamy uwagę na punkt 1.2.10, który jest punktem wyjścia wielu późniejszych wywodów teoretycznych. Czytelnik przechodząc do podrozdziału 1.3, opuści gwałtownie

krainę „czystej matematyki”, by wkroczyć do królestwa „czystego programowania”.

Omawiane zagadnienia są rozwinięte i podane w sposób bardziej przystępny w książce Grahama, Knutha i Patashnika *Matematyka konkretna*, (PWN, Warszawa 1996), którą dalej nazywamy po prostu *CMath*.

1.2.1. Indukcja matematyczna

Niech $P(n)$ będzie pewnym stwierdzeniem mówiącym o liczbie całkowitej n ; na przykład $P(n)$ może oznaczać „ n razy $(n + 3)$ jest liczbą parzystą” albo „jeśli $n \geq 10$, to $2^n > n^3$ ”. Przypuśćmy, że chcemy udowodnić, że $P(n)$ jest prawdziwe dla wszystkich dodatnich liczb całkowitych n . Jest na to następujący sposób:

- a) Przedstawiamy dowód, że $P(1)$ jest prawdziwe.
- b) Przedstawiamy dowód, że „jeśli wszystkie $P(1), P(2), \dots, P(n)$ są prawdziwe, to $P(n + 1)$ też jest prawdziwe”; ten dowód powinien być poprawny dla dowolnej dodatniej liczby całkowitej n .

Jako przykład rozważmy następujący ciąg równań, który od czasów starożytnych odkryło niezależnie wiele osób:

$$\begin{aligned} 1 &= 1^2, \\ 1 + 3 &= 2^2, \\ 1 + 3 + 5 &= 3^2, \\ 1 + 3 + 5 + 7 &= 4^2, \\ 1 + 3 + 5 + 7 + 9 &= 5^2. \end{aligned} \tag{1}$$

Ogólną własność możemy sformułować następująco:

$$1 + 3 + \dots + (2n - 1) = n^2. \tag{2}$$

Teraz na chwilę nazwijmy tę równość $P(n)$. Chcemy udowodnić, że $P(n)$ jest prawdziwe dla wszystkich dodatnich n . Zgodnie z opisanyem powyżej schematem mamy:

- a) „ $P(1)$ jest prawdziwe, bo $1 = 1^2$ ”.
- b) „Jeśli wszystkie $P(1), \dots, P(n)$ są prawdziwe, to w szczególności $P(n)$ jest prawdziwe, zatem zachodzi równość (2). Dodając $2n + 1$ do obu jej stron, otrzymamy

$$1 + 3 + \dots + (2n - 1) + (2n + 1) = n^2 + 2n + 1 = (n + 1)^2,$$

stąd $P(n + 1)$ też jest prawdziwe”.

Możemy patrzeć na tę metodę jak na *algorytm przeprowadzania dowodu*. Poniższy algorytm generuje dowód stwierdzenia $P(n)$ dla dowolnej liczby całkowitej n przy założeniu, że sami wykonamy kroki (a) i (b).

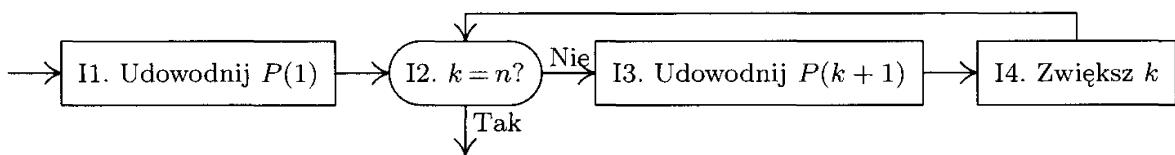
Algorytm I (*Budowanie dowodu*). Dana jest dodatnia liczba całkowita n . Niniejszy algorytm służy do udowodnienia, że stwierdzenie $P(n)$ jest prawdziwe.

I1. [Udowadnianie $P(1)$] Przyjmij $k \leftarrow 1$ i daj w wyniku $P(1)$, wykonany w ramach kroku (a).

I2. [Czy $k = n$?] Jeśli $k = n$, to zakończ wykonanie algorytmu; żądany dowód został już w całości dany w wyniku.

I3. [Udowadnianie $P(k + 1)$] Korzystając z dowodu wykonanego w punkcie (b), daj w wyniku dowód stwierdzenia, że „jeśli wszystkie $P(1), \dots, P(k)$ są prawdziwe, to $P(k + 1)$ też jest prawdziwe”. Daj w wyniku także „Udowodniliśmy $P(1), \dots, P(k)$, stąd $P(k + 1)$ jest prawdziwe”.

I4. [Zwiększenie k] Zwiększ k o 1 i przejdź do kroku I2. ■



Rys. 2. Algorytm I: indukcja matematyczna.

Ponieważ powyższy algorytm w sposób oczywisty prezentuje dowód stwierdzenia $P(n)$ dla dowolnego n , metoda udowadniania polegająca na wykonaniu kroków (a) i (b) jest logicznie uzasadniona. Nazywa się ją „dowodzeniem przez indukcję matematyczną”.

Należy odróżnić pojęcie indukcji matematycznej od tego, co zazwyczaj w nauce nazywa się rozumowaniem indukcyjnym. Naukowiec zbiera obserwacje i posługując się „indukcją”, tworzy ogólną teorię bądź hipotezy, które mają wyjaśniać zaobserwowane fakty; na przykład mogliśmy zauważycię pięć związków (1) i sformułować (2). W tym sensie indukcja nie jest niczym więcej niż trafnym przewidywaniem. Matematycy nazwaliby to wynikiem empirycznym lub koniekturą.

Przyjrzyjmy się kolejnemu przykładowi. Niech $p(n)$ oznacza liczbę podziałów liczby n , tj. liczbę różnych sposobów zapisania n w postaci sumy dodatnich liczb całkowitych, bez uwzględniania kolejności składników. Liczbę 5 można podzielić na siedem sposobów

$$1 + 1 + 1 + 1 + 1 = 2 + 1 + 1 + 1 = 2 + 2 + 1 = 3 + 1 + 1 = 3 + 2 = 4 + 1 = 5,$$

stąd $p(5) = 7$. Nietrudno więc wyznaczyć kilka początkowych wartości

$$p(1) = 1, \quad p(2) = 2, \quad p(3) = 3, \quad p(4) = 5, \quad p(5) = 7.$$

W tym miejscu nieśmiało moglibyśmy sformułować indukcyjnie hipotezę, że ciąg $p(2), p(3), \dots$ przebiega *liczby pierwsze*. W celu sprawdzenia hipotezy posuwamy się dalej, obliczając $p(6)$, i... jest! $p(6) = 11$, co potwierdza naszą koniekturę.

[Niestety, $p(7) = 15$. To niweczy przypuszczenia i musimy zaczynać wszystko od początku. O liczbach $p(n)$ wiadomo, że są dość skomplikowane, chociaż S. Ramanujan osiągnął wiele sukcesów, zgadując, a następnie dowodząc wielu

ich ciekawych własności. Dokładniejsze informacje znajdzie Czytelnik w: G. H. Hardy, *Ramanujan* (London: Cambridge University Press, 1940), rozdziały 6 i 8].

Indukcja matematyczna jest czymś zupełnie innym niż indukcja w powyższym sensie. To nie zgadywanie, lecz kompletny dowód twierdzenia. W istocie jest to dowód nieskończoność wielu twierdzeń, po jednym dla każdego n . Metodę tę nazwano „indukcją”, bo w jakiś sposób trzeba zadecydować, co ma być dowodzone, zanim metoda indukcji matematycznej może zostać zastosowana. Od tego miejsca będziemy używać słowa „indukcja”, mając na myśli dowód przez indukcję matematyczną.

Istnieje geometryczny sposób udowodnienia równości (2). Na rysunku 3 widać (dla $n = 6$) n^2 pól połączonych w grupy złożone z $1 + 3 + \dots + (2n - 1)$ pól. Tym niemniej w ostatecznej analizie rysunek ten może służyć za „dowód” jedynie wtedy, gdy pokażemy, że taka konstrukcja może zostać przeprowadzona dla wszystkich n , a pokazanie tego sprowadza się w gruncie rzeczy do dowodu przez indukcję.

W naszym dowodzie równości (2) korzystaliśmy jedynie ze szczególnego przypadku (b): pokazaliśmy po prostu, że prawdziwość $P(n)$ pociąga za sobą prawdziwość $P(n + 1)$. Z taką sytuacją spotykamy się dość często. Kolejny przykład pozwala lepiej poznać siłę indukcji. Definiujemy ciąg Fibonacciego F_0, F_1, F_2, \dots za pomocą reguły mówiącej, że $F_0 = 0, F_1 = 1$, a każdy następny wyraz jest sumą dwóch poprzednich. Ciąg rozpoczyna się zatem wyrazami $0, 1, 1, 2, 3, 5, 8, 13, \dots$; zajmiemy się nim bardziej szczegółowo w punkcie 1.2.8. Udowodnimy teraz, że jeśli ϕ jest liczbą $(1 + \sqrt{5})/2$, to

$$F_n \leq \phi^{n-1} \quad (3)$$

dla wszystkich dodatnich liczb całkowitych n . Nazwijmy ten wzór $P(n)$.

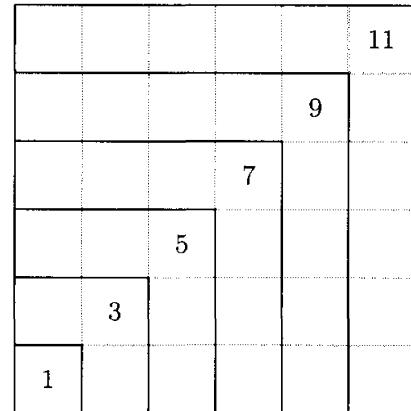
Jeśli $n = 1$, to $F_1 = 1 = \phi^0 = \phi^{n-1}$, zatem udowodniliśmy krok (a). Przed przystąpieniem do kroku (b) zauważmy, że również $P(2)$ jest prawdą, gdyż $F_2 = 1 < 1.6 < \phi^1 = \phi^{2-1}$. Teraz jeśli wszystkie $P(1), P(2), \dots, P(n)$ są prawdziwe i $n > 1$, to wiemy w szczególności, że $P(n - 1)$ i $P(n)$ też są prawdziwe. Stąd $F_{n-1} \leq \phi^{n-2}$ i $F_n \leq \phi^{n-1}$. Dodając te nierówności stronami, otrzymujemy

$$F_{n+1} = F_{n-1} + F_n \leq \phi^{n-2} + \phi^{n-1} = \phi^{n-2}(1 + \phi). \quad (4)$$

Istotną cechą liczby ϕ , a w zasadzie powodem, dla którego umieszczamy ją w tym problemie na honorowym miejscu, jest własność

$$1 + \phi = \phi^2. \quad (5)$$

Podstawiając (5) do (4), otrzymujemy $F_{n+1} \leq \phi^n$, czyli $P(n + 1)$. Zatem krok (b) został wykonany. Zauważmy, że do kroku (b) zabraliśmy się na dwa sposoby:



Rys. 3. Suma liczb nieparzystych jest kwadratem.

udowodniliśmy $P(n+1)$ wprost dla $n = 1$ i użyliśmy metody indukcyjnej dla $n > 1$. Było to konieczne, gdyż dla $n = 1$ odniesienie do $P(n-1) = P(0)$ byłoby nieuprawnione.

Indukcja matematyczna może być także wykorzystana do udowadniania własności algorytmów. Rozważmy następujące uogólnienie algorytmu Euklidesa:

Algorytm E (Zmodyfikowany algorytm Euklidesa). Dane są dwie dodatnie liczby całkowite m i n , obliczamy ich największy wspólny dzielnik d oraz dwie takie liczby całkowite a i b , że $am + bn = d$.

- E1.** [Inicjowanie] Przyjmij $a' \leftarrow b \leftarrow 1$, $a \leftarrow b' \leftarrow 0$, $c \leftarrow m$, $d \leftarrow n$.
- E2.** [Dzielenie] Niech q i r będą odpowiednio ilorazem i resztą z dzielenia c przez d . (Mamy $c = qd + r$ i $0 \leq r < d$).
- E3.** [Czy reszta zero?] Jeśli $r = 0$, to zakończ algorytm; w takim przypadku $am + bn = d$, tak jak miało być.
- E4.** [Ponowienie] Przyjmij $c \leftarrow d$, $d \leftarrow r$, $t \leftarrow a'$, $a' \leftarrow a$, $a \leftarrow t - qa$, $t \leftarrow b'$, $b' \leftarrow b$, $b \leftarrow t - qb$ i powróć od E2. ■

Jeśli z tego algorytmu usuniemy zmienne a , b , a' i b' oraz użyjemy m i n zamiast dodatkowych zmiennych c i d , to otrzymamy nasz stary algorytm 1.1E. Nowa wersja robi trochę więcej – wyznacza współczynniki a i b . Przypuśćmy, że $m = 1769$ i $n = 551$. Mamy kolejno (po kroku E2):

a'	a	b'	b	c	d	q	r
1	0	0	1	1769	551	3	116
0	1	1	-3	551	116	4	87
1	-4	-3	13	116	87	1	29
-4	5	13	-16	87	29	3	0

Odpowiedź jest poprawna: $5 \times 1769 - 16 \times 551 = 8845 - 8816 = 29$, jest to największy wspólny dzielnik liczb 1769 i 551.

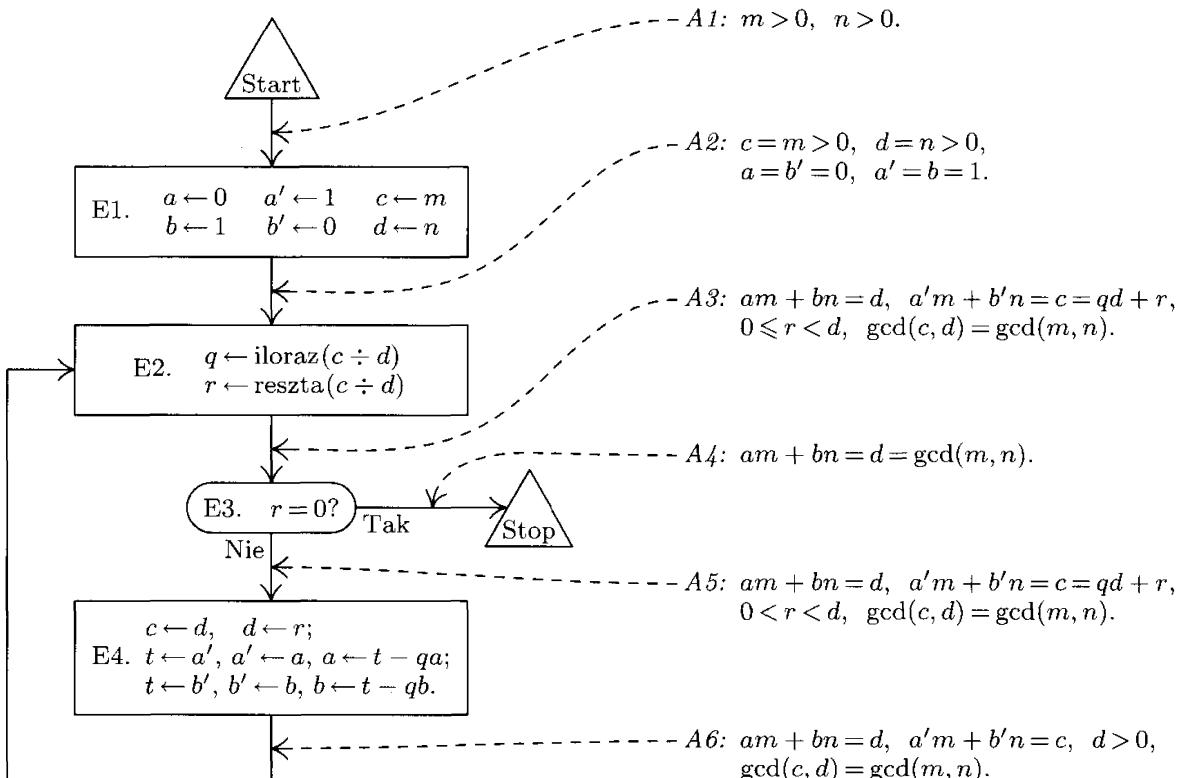
Chcielibyśmy teraz udowodnić, że ten algorytm działa poprawnie dla dowolnych m i n . Możemy spróbować użyć indukcji matematycznej, przyjmując za $P(n)$ stwierdzenie „Algorytm E pracuje poprawnie dla n i wszystkich liczb całkowitych m ”. Tym sposobem jednak w prosty sposób donikąd dojść się nie da, trzeba udowodnić kilka dodatkowych lematów. Po chwili zastanowienia doszczegamy, że należy powiedzieć coś o wartościach a , b , a' i b' . Odpowiednie stwierdzenie mówi, że równości

$$a'm + b'n = c, \quad am + bn = d \tag{6}$$

są prawdziwe po każdym wykonaniu kroku E2. Możemy to udowodnić, zauważając, że równości są prawdziwe, gdy po raz pierwszy dojdziemy do E2, oraz że krok E4 nie zmienia ich prawdziwości. (Zobacz ćwiczenie 6).

Jesteśmy gotowi do pokazania poprawności algorytmu E za pomocą indukcji względem n : jeśli m jest wielokrotnością n , to algorytm oczywiście działa poprawnie, gdyż sprawia jest załatwiona już przy pierwszym wykonaniu kroku E3. Ta

sytuacja występuje zawsze, gdy $n = 1$. Pozostaje tylko przypadek $n > 1$ i m nie będącego wielokrotnością n : algorytm po pierwszym przebiegu przechodzi do wykonania przypisania $c \leftarrow n$, $d \leftarrow r$, a ponieważ $r < n$, to z założenia indukcyjnego otrzymujemy stwierdzenie, że końcowa wartość d jest gcd liczb n i r . Na mocy uzasadnienia przedstawionego w podrozdziale 1.1, pary $\{m, n\}$ i $\{n, r\}$ mają te same wspólne dzielniki, a w szczególności ten sam największy wspólny dzielnik. Stąd d jest gcd liczb m i n , przy tym z (6) otrzymujemy $am + bn = d$.



Rys. 4. Schemat blokowy algorytmu E opisany asercjami, które dowodzą jego poprawności.

Zwroty w powyższym dowodzie oznaczone kursywą to tradycyjne zwroty używane w dowodach indukcyjnych. W części (b) dowodu zamiast mówić: „Teraz przy założeniach $P(1), P(2), \dots, P(n)$ udowodnimy $P(n+1)$ ”, mówimy krótko: „Teraz udowodnimy $P(n)$; z założenia indukcyjnego $P(k)$ jest prawdziwe dla $1 \leq k < n$ ”.

Jeśli uważnie przyjrzymy się dowodowi i nieco zmienimy punkt widzenia, dostrzeżemy ogólną metodę dającą się zastosować do dowodzenia poprawności *dowolnego* algorytmu. Pomysł polega na opisaniu schematu blokowego algorytmu w ten sposób, by przy każdej strzałce znalazła się asercja opisująca stan rzeczy w chwili, gdy wykonywane jest obliczenie dotyczące tej strzałki. Na rysunku 4 asercje zostały oznaczone etykietami $A1, A2, \dots, A6$. (We wszystkich tych asercjach zakładamy niejawnie, że zmienne przyjmują wartości całkowite; założenia te nie zostały wypisane, by nie zaciemniać rysunku). Asercja $A1$ opisuje począ-

kowe założenia po rozpoczęciu algorytmu, natomiast A_4 – to, co chcielibyśmy udowodnić na temat wyjściowych wartości a, b i d .

Ogólna metoda polega na udowodnieniu, że dla każdego bloku zachodzi warunek:

jeśli asercja związana z jakąkolwiek strzałką prowadzącą do bloku jest prawdziwa zanim zostanie wykonana operacja opisywana przez blok, to wszystkie asercje związane ze strzałkami wychodzącymi z bloku są prawdziwe po wykonaniu tej operacji.

(7)

Stąd na przykład musimy udowodnić, że albo A_2 , albo A_6 przed wykonaniem E2 pociąga za sobą A_3 po wykonaniu E2. (W tym wypadku A_2 jest stwierdzeniem silniejszym niż A_6 , tj. A_2 pociąga za sobą A_6 . Wystarczy zatem udowodnić, że A_6 przed E2 pociąga za sobą A_3 po E2. Zauważmy, że warunek $d > 0$ jest potrzebny w A_6 choćby po to, żeby udowodnić, że operacja E2 ma sens). Trzeba również pokazać, że A_3 i $r = 0$ implikuje A_4 ; stąd A_3 i $r \neq 0$ pociąga za sobą A_5 itd. Wszystkie potrzebne dowody są dosyć proste.

Z prawdziwości (7) dla każdego bloku wynika, że wszystkie asercje są prawdziwe podczas każdego wykonania algorytmu. Możemy to wykazać przez indukcję względem liczby kroków obliczenia (w sensie liczby strzałek na schemacie blokowym, po których obliczenie przeszło). Przy przechodzeniu po pierwszej strzałce, tej prowadzącej od bloku „Start”, asercja A_1 jest prawdziwa, bo zawsze zakładamy, że nasze wartości wejściowe spełniają specyfikację. Jeśli asercja związana z n -tą strzałką jest prawdziwa, to na mocy (7) asercja związana z $(n+1)$ strzałką też jest prawdziwa.

Użycie ogólnej metody dowodzenia poprawności algorytmu sprowadza się w zasadzie do umieszczenia na schemacie blokowym odpowiednich asercji. Po sformułowaniu kroku indukcyjnego pozostaje raczej rzemieślnicza praca polegająca na wykazaniu, że każda asercja na strzałce „wchodzącej” pociąga asercje na strzałkach „wychodzących”. Rzemiosłem jest w istocie także wymyślenie większości asercji, gdy kilka najtrudniejszych już się odkryło. W naszym przykładzie bardzo łatwo jest dopisać asercje A_2, A_3 i A_5 , jeśli wymyślimy już A_1, A_4 i A_6 . W tym przykładzie asercja A_6 jest twórczą częścią dowodu; wszystkie pozostałe mogłyby w zasadzie zostać wyprowadzone automatycznie. Z powyższego powodu dla algorytmów prezentowanych w książce nie przedstawiamy tak szczegółowych dowodów, jak na rysunku 4. Wystarczy zaprezentować najistotniejsze asercje indukcyjne. Te asercje wymieniamy albo w tekście omawiającym algorytm, albo umieszczamy pośród komentarzy w opisie algorytmu (w nawiasach okrągłych).

Powyższe podejście do dowodzenia poprawności algorytmów ma jeszcze inny, może nawet ważniejszy aspekt: odzwierciedla sposób rozumienia algorytmu. Przypomnijmy sobie, że w podrozdziale 1.1 ostrzegaliśmy Czytelnika, by algorytmu nie czytał jak beletrystyki. Polecaliśmy wypróbowanie algorytmu na paru przykładach. Zaznaczaliśmy to wyraźnie, bo przyjrzenie się pracy algorytmu pomaga Czytelnikowi podświadomie sformułować różne asercje. Autor jest przekonany, że dopiero wtedy naprawdę rozumiemy algorytm, gdy dochodzimy do punktu, w którym nasz umysł po cichu wypełnił się asercjami, jak na rysunku 4.

Taki pogląd ma konsekwencje psychologiczne dotyczące sposobów przekazywania algorytmów. Wynika z niego, że gdy wyjaśniamy komuś algorytm, kluczowe asercje, które nie dają się w łatwy sposób wywieść automatycznie, powinniśmy wyrazić otwarcie. Opowiadając znajomym algorytm E, koniecznie powiedziecie o asercji A6.

Uważny Czytelnik spostrzeże lukę w naszym ostatnim dowodzie poprawności algorytmu E. Nie pokazaliśmy, że algorytm się zakończy! Udowodniliśmy jedynie, że jeśli się zakończy, to da poprawną odpowiedź.

(Zauważmy, że algorytm E ma sens nawet wtedy, gdy dopuścimy, by zmienne m, n, c, d i r miały postać $u + v\sqrt{2}$, gdzie u i v są liczbami całkowitymi. Zmienne q, a, b, a', b' mogą pozostać całkowite. Jeżeli uruchomimy algorytm dla, powiedzmy, $m = 12 - 6\sqrt{2}$ i $n = 20 - 10\sqrt{2}$, to obliczy on „największy wspólny dzielnik” $d = 4 - 2\sqrt{2}$ oraz $a = +2, b = -1$. Przy tak zmienionych założeniach dowód asercji A1 – A6 pozostaje poprawny; asercje są prawdziwe przez cały czas wykonania algorytmu. Ale jeżeli na wejściu podamy $m = 1$ i $n = \sqrt{2}$, to obliczenie nigdy się nie zakończy (zobacz ćwiczenie 12). Stąd dowód asercji A1 – A6 nie pokazuje, że wykonanie algorytmu się zakończy).

Dowody własności stopu zazwyczaj przeprowadza się oddzielnie. W ćwiczeniu 13 jest pokazane, że jednak w wielu istotnych przypadkach można rozszerzyć powyższą metodę, by dowód własności stopu otrzymywać „przy okazji”.

Poprawność algorytmu E udowodniliśmy już dwa razy. Dla ścisłości wywodu powinniśmy także udowodnić, że pierwszy algorytm w tym podrozdziale, algorytm I, jest poprawny. W końcu użyliśmy tego algorytmu do pokazania poprawności wszystkich dowodów przez indukcję. Jednak gdybyśmy spróbowali dowodzić poprawność algorytmu I, natknęlibyśmy się na problem: nie da się go udowodnić bez indukcji! Powstałoby błędne koło.

W przeprowadzonych analizach w dowodzie każdej własności liczb całkowitych kryje się dowód przez indukcję. Jeśli rozpatrywać zagadnienia na zupełnie podstawowym poziomie, to okazałoby się, że liczby całkowite są *zdefiniowane* indukcyjnie. Stąd możemy przyjąć za aksjomat, że każda liczba całkowita jest albo równa 1, albo możemy ją osiągnąć, biorąc 1 i odpowiednio wiele razy dodając do niej 1. To wystarczy do dowodu poprawności algorytmu I. [Ścisłe studium zagadnień leżących u podstaw liczb całkowitych zawiera artykuł Leona Henkina, „On Mathematical Induction” AMM 67 (1960), 323–338].

Idea indukcji matematycznej jest nierozerwalnie związana z pojęciem liczby. Pierwszym Europejczykiem, który zastosował indukcję w ścisłych dowodach był, włoski naukowiec Francesco Maurolico, a było to w 1575 roku. Pierre de Fermat w XVII wieku poczynił dalsze postępy; nazwał wówczas indukcję „metodą kroków”. Idea indukcji pojawia się również we wczesnych pracach Blaise Pascala (1653). Określenie „indukcja matematyczna” zostało wymyślone przez A. De Morgana na początku dziewiętnastego stulecia. [Zobacz AMM 24 (1917), 199–207; 25 (1918), 197–201; Arch. Hist. Exact Sci. 9 (1972), 1–21]. Bardziej szczegółowe rozważania dotyczące tego zagadnienia można znaleźć w książce G. Pólya *Induction and Analogy in Mathematics* (Princeton, N.J.: Princeton University Press, 1954), rozdział 7.

Sformułowanie metody dowodzenia poprawności algorytmów przez asercje i indukcję pochodzi głównie od R. W. Floyda. Zauważył on, że definicja znaczenia każdej operacji języka programowania może być sformułowana jako reguła logiczna mówiąca, że pewne asercje można udowodnić po operacji przy założeniu, że pewne asercje były prawdziwe przed operacją [zobacz „Assigning Meanings to Programs”, *Proc. Symp. Appl. Math.*, Amer. Math. Soc., **19** (1967), 19–32]. Podobne pomysły były głoszone niezależnie przez Petera Naura, *BIT* **6** (1966), 310–316, który nazywał asercje „ogólnymi migawkami” (*general snapshots*). Uszczegółowienie tych metod w postaci pojęcia „niezmienników” zostało wprowadzone przez C. A. R. Hoare'a; zobacz na przykład *CACM* **14** (1971), 39–45. Później autorzy stwierdzili, że większy pożytek daje odwrócenie kierunku reguł Floyda, tj. rozpoczęwanie od tego, co ma być prawdziwe *po* operacji, by dojść do „najsłabszego warunku wstępnego” (*weakest precondition*), który musi być prawdziwy *przed* wykonaniem operacji. To podejście otwiera drogę do odkrycia nowych algorytmów, jeśli rozpoczniemy od specyfikacji wyjścia i będziemy posuwać się wstecz – sam proces konstrukcji zagwarantuje poprawność algorytmu. [Zobacz E. W. Dijkstra, *CACM* **18** (1975), 453–457; *Umiejętność programowania* (WNT, 1978)].

Pomysł asercji indukcyjnych pojawił się właściwie w postaci zarodkowej już w 1946 roku, w tym samym czasie kiedy H. H. Goldstine i J. von Neumann zaprezentowali schematy blokowe. Ich pierwotne schematy zawierały „bloczki asercji” mające wiele wspólnego z asercjami z rysunku 4. [Zobacz John von Neumann, *Collected Works* **5** (New York: Macmillan, 1963), 91–99. Zobacz też wczesne komentarze A. M. Turinga na temat weryfikacji w *Report of a Conference on High Speed Automatic Calculating Machines* (Cambridge Univ., 1949), 67–68 z rysunkami; porównaj też przedruk z komentarzami F. L. Morrisa i C. B. Jonesa w *Annals of the History of Computing* **6** (1984), 139–143].

Zrozumienie programu może być dużo łatwiejsze, jeśli budując go, doda się w odpowiednim miejscu jedną lub dwie instrukcje dotyczące stanu maszyny. (...)

W metodzie skrajnie teoretycznej jest wymagany niezbity dowód matematyczny asercji.

W metodzie skrajnie empirycznej testuje się program z wieloma różnymi warunkami początkowymi, po czym uznaje go za poprawny, gdy asercje są spełnione w każdym przypadku.

Obie metody mają swoje słabe strony.

— A. M. TURING, Ferranti Mark I Programming Manual (1950)

ĆWICZENIA

1. [05] Wyjaśnij, w jaki sposób zmodyfikować metodę dowodu przez indukcję matematyczną, jeśli chcemy udowodnić pewne twierdzenie $P(n)$ dla wszystkich *niewjemnych* liczb całkowitych, tj. dla $n = 0, 1, 2, \dots$ zamiast dla $n = 1, 2, 3, \dots$
- 2. [15] Z poniższym dowodem jest chyba coś nie tak. Co konkretnie? „**Twierdzenie.** Niech a będzie dowolną liczbą dodatnią. Dla wszystkich dodatnich liczb całkowitych n mamy $a^{n-1} = 1$. Dowód. Jeśli $n = 1$, to $a^{n-1} = a^{1-1} = a^0 = 1$. Z kolei na mocy

indukcji, przy założeniu, że twierdzenie jest prawdziwe dla $1, 2, \dots, n$, mamy

$$a^{(n+1)-1} = a^n = \frac{a^{n-1} \times a^{n-1}}{a^{(n-1)-1}} = \frac{1 \times 1}{1} = 1;$$

stąd stwierdzenie jest prawdziwe również dla $n + 1$.

3. [18] Poniższy dowód przez indukcję wydaje się być poprawny, ale z jakiegoś powodu równość dla $n = 6$ daje $\frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \frac{1}{20} + \frac{1}{30} = \frac{5}{6}$ po lewej stronie, a $\frac{3}{2} - \frac{1}{6} = \frac{4}{3}$ po prawej. Czy potrafisz znaleźć błąd? „Twierdzenie.”

$$\frac{1}{1 \times 2} + \frac{1}{2 \times 3} + \cdots + \frac{1}{(n-1) \times n} = \frac{3}{2} - \frac{1}{n}.$$

Dowód. Korzystamy z indukcji względem n . Dla $n = 1$ oczywiście $3/2 - 1/n = 1/(1 \times 2)$. Założymy, że twierdzenie jest prawdziwe dla n , wówczas

$$\begin{aligned} \frac{1}{1 \times 2} + \cdots + \frac{1}{(n-1) \times n} + \frac{1}{n \times (n+1)} \\ = \frac{3}{2} - \frac{1}{n} + \frac{1}{n(n+1)} = \frac{3}{2} - \frac{1}{n} + \left(\frac{1}{n} - \frac{1}{n+1} \right) = \frac{3}{2} - \frac{1}{n+1}. \end{aligned}$$

4. [20] Udowodnij, że oprócz równości (3) liczby Fibonacciego spełniają nierówność $F_n \geq \phi^{n-2}$.

5. [21] *Liczba pierwsza* to liczba całkowita większa od jedynki, która nie ma całkowitych dzielników poza jedynką i samą sobą. Opierając się na tej definicji, udowodnij za pomocą indukcji matematycznej, że każda liczba całkowita większa od jedynki może być zapisana w postaci iloczynu jednej lub więcej liczb pierwszych.

6. [20] Udowodnij, że jeśli równości (6) zachodzą przed wykonaniem kroku E4, to zachodzą także po jego wykonaniu.

7. [23] Sformułuj i udowodnij przez indukcję wzór na sumę $1^2, 2^2 - 1^2, 3^2 - 2^2 + 1^2, 4^2 - 3^2 + 2^2 - 1^2, 5^2 - 4^2 + 3^2 - 2^2 + 1^2$ itd.

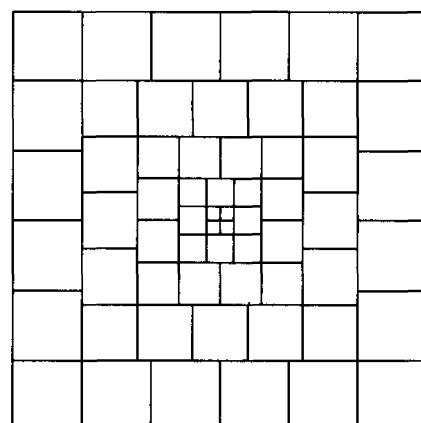
► **8.** [25] (a) Udowodnij przez indukcję twierdzenie Nikomachusa (około 100 roku): $1^3 = 1, 2^3 = 3 + 5, 3^3 = 7 + 9 + 11, 4^3 = 13 + 15 + 17 + 19$ itd. (b) Skorzystaj z tego wyniku, by udowodnić słynny wzór $1^3 + 2^3 + \cdots + n^3 = (1 + 2 + \cdots + n)^2$.

[Uwaga: Interesującą interpretację geometryczną tego wzoru (zobacz rysunek 5) zaproponował autorowi R. W. Floyd. Pomysł ma związek z twierdzeniem Nikomachusa i rysunkiem 3. Inne dowody „wzrokowe” można znaleźć w książkach: Martin Gardner *Knotted Doughnuts* (New York: Freeman, 1986), rozdział 16; J. H. Conway, R. K. Guy, *Księga liczb* (Warszawa: WNT, 1999), rozdział 2].

$$\text{Bok} = 5 + 5 + 5 + 5 + 5 = 5 \cdot (5 + 1)$$

$$\begin{aligned} \text{Bok} &= 5 + 4 + 3 + 2 + 1 + 1 + 2 + 3 + 4 + 5 \\ &= 2(1 + 2 + \cdots + 5) \end{aligned}$$

$$\begin{aligned} \text{Pole} &= 4 \cdot 1^2 + 4 \cdot 2 \cdot 2^2 + 4 \cdot 3 \cdot 3^2 + 4 \cdot 4 \cdot 4^2 + 4 \cdot 5 \cdot 5^2 \\ &= 4(1^3 + 2^3 + \cdots + 5^3) \end{aligned}$$



Rys. 5. Geometryczna wersja ćwiczenia 8(b).

9. [20] Udowodnij przez indukcję, że jeśli $0 < a < 1$, to $(1 - a)^n \geq 1 - na$.
10. [M22] Udowodnij przez indukcję, że jeśli $n \geq 10$, to $2^n \geq n^3$.
11. [M30] Znajdź i udowodnij prosty wzór na sumę

$$\frac{1^3}{1^4 + 4} - \frac{3^3}{3^4 + 4} + \frac{5^3}{5^4 + 4} - \cdots + \frac{(-1)^n(2n+1)^3}{(2n+1)^4 + 4}.$$

12. [M25] Pokaż, że (zgodnie z notką w tekście) algorytm E można tak uogólnić, by jego danymi wejściowymi były liczby postaci $u + v\sqrt{2}$, gdzie u i v są liczbami całkowitymi, a obliczenie będzie można w dalszym ciągu prowadzić metodami elementarnymi (tj. bez posługiwania się nieskończonym rozwinięciem dziesiętnym $\sqrt{2}$). Udowodnij, że obliczenie nie zakończy się dla $m = 1$ i $n = \sqrt{2}$.

- 13. [M23] Rozbuduj algorytm E, dodając nową zmienną T oraz operację „ $T \leftarrow T + 1$ ” na początku każdego kroku. (T zachowuje się jak licznik zliczający liczbę wykonanych kroków). Załóż, że T jest na początku równe zero, stąd asercja A1 na rysunku 4 przyjmuje postać „ $m > 0, n > 0, T = 0$ ”. Dodatkowy warunek „ $T = 1$ ” powinien na podobnej zasadzie zostać dołączony do A2. Pokaż, jak dodać do asercji dodatkowe warunki w taki sposób, żeby z dowolnej z nich wynikało $T \leq 3n$ i w dalszym ciągu dawało się przeprowadzić dowód indukcyjny. (Z takich asercji będzie wynikać, że obliczenie musi się zakończyć w co najwyżej $3n$ krokach).
14. [50] (R. W. Floyd) Napisz program, który jako dane wejściowe wczyta program w pewnym języku programowania z niektórymi asercjami i spróbuje uzupełnić brakujące asercje potrzebne do przeprowadzenia indukcyjnego dowodu jego poprawności. (Na przykład spróbuj napisać program, który mając podane asercje A1, A4 i A6, udowodni poprawność algorytmu E. Aby uzyskać więcej szczegółów, zobacz prace R. W. Floyda i F. C. Kinga w materiałach z Kongresu IFIP, 1971).

- 15. [HM28] (*Uogólniona indukcja*) W tekście jest pokazane, jak dowodzić prawdziwość zdania $P(n)$ zależnego od jednej liczby całkowitej n , ale nie powiedziano nic o prawdziwości zdań $P(n, m)$ zależnych od dwóch liczb. W takich przypadkach dowód przeprowadza się zazwyczaj za pomocą „podwójnej indukcji”, co często jest niejasne. Istnieje zasada, ogólniejsza od prostej indukcji względem zmiennej całkowitej, mająca zastosowanie także w sytuacjach, gdy dowodzimy twierdzeń o zbiorach nieprzeliczalnych, na przykład „ $P(x)$ dla wszystkich x rzeczywistych”.

Niech „ \prec ” będzie dwuargumentową relacją na zbiorze S o następujących własnościach:

- Dla dowolnych x, y i z ze zbioru S , jeśli $x \prec y$ i $y \prec z$, to $x \prec z$.
- Dla dowolnych x i y ze zbioru S zachodzi dokładnie jedna z następujących zależności: $x \prec y$, $x = y$ lub $y \prec x$.
- Jeśli A jest niepustym podzbiorem S , to w zbiorze A istnieje taki element x , że dla dowolnego y z A zachodzi $x \preceq y$ (gdzie $x \preceq y$ oznacza „ $x \prec y$ lub $x = y$ ”).

Relacja \prec nazywana jest *dobrym uporządkowaniem* zbioru S . Na przykład dodatnie liczby całkowite są dobrze uporządkowane przez zwykłą relację mniejszości, $<$.

- Pokaż, że zbiór *wszystkich* liczb całkowitych nie jest dobrze uporządkowany przez relację $<$.
- Zdefiniuj relację dobrego porządku na zbiorze wszystkich liczb całkowitych.
- Czy zbiór wszystkich nieujemnych liczb rzeczywistych jest dobrze uporządkowany przez $<?$

- d) (*Porządek leksykograficzny*) Niech S będzie zbiorem dobrze uporządkowanym przez relację \prec i dla $n > 0$ niech T_n będzie zbiorem wszystkich krotek (x_1, x_2, \dots, x_n) elementów x_j ze zbioru S . Zdefiniujmy: $(x_1, x_2, \dots, x_n) \prec (y_1, y_2, \dots, y_n)$, jeśli istnieje k , $1 \leq k \leq n$, takie że $x_j = y_j$ dla $1 \leq j < k$, ale $x_k \prec y_k$ w zbiorze S . Czy \prec dobrze porządkuje zbiór T_n ?
- e) Ciąg dalszy (d). Niech $T = \bigcup_{n \geq 1} T_n$. Zdefiniujemy: $(x_1, x_2, \dots, x_m) \prec (y_1, y_2, \dots, y_n)$, jeśli $x_j = y_j$ dla $1 \leq j < k$ i $x_k \prec y_k$ dla pewnego $k \leq \min(m, n)$ albo gdy $m < n$ i $x_j = y_j$ dla $1 \leq j \leq m$. Czy \prec dobrze porządkuje zbiór T ?
- f) Pokaż, że \prec jest dobrym uporządkowaniem S wtedy i tylko wtedy, gdy spełnia (i) i (ii) oraz nie istnieje nieskończony ciąg x_1, x_2, x_3, \dots , w którym $x_{j+1} \prec x_j$ dla wszystkich $j \geq 1$.
- g) Niech S będzie dobrze uporządkowany przez \prec i niech $P(x)$ będzie twierdzeniem o elemencie x zbioru S . Pokaż, że jeżeli $P(x)$ można udowodnić przy założeniu, że $P(y)$ jest prawdziwe dla wszystkich $y \prec x$, to $P(x)$ jest prawdziwe dla *wszystkich* x ze zbioru S .

[*Uwagi:* Część (g) jest uogólnieniem prostej indukcji, jak było obiecane na początku ćwiczenia. W przypadku gdy S jest zbiorem liczb całkowitych, jest to po prostu pewna postać prostej indukcji. Musimy wówczas udowodnić, że $P(1)$ jest prawdziwe, jeśli $P(y)$ jest prawdziwe dla wszystkich dodatnich liczb całkowitych $y < 1$. Jest to równoważne stwierdzeniu, że musimy udowodnić $P(1)$, gdyż $P(y)$ jest (trywialnie) prawdziwe dla wszystkich takich liczb y (bo ich po prostu nie ma). W efekcie często okazuje się, że $P(1)$ nie trzeba udowadniać za pomocą osobnego wywodu.

Część (d) w połączeniu z częścią (g) daje silne narzędzie indukcji n -wymiarowej, pomocne przy dowodzeniu twierdzeń $P(m_1, \dots, m_n)$ mówiących o n dodatnich liczbach całkowitych m_1, \dots, m_n .

Część (f) znajduje istotne zastosowanie w algorytmach komputerowych: jeśli potrafimy odwzorować każdy stan obliczenia x w element $f(x)$ dobrze uporządkowanego zbioru S w ten sposób, że każdy krok obliczenia zmienia stan x w taki stan y , że $f(y) \prec f(x)$, to wykonanie algorytmu musi się zakończyć. Ta zasada jest uogólnieniem wywodu o istotnie malejącej wartości n , za pomocą którego dowodziśmy własności stopu algorytmu 1.1E].

1.2.2. Liczby, potęgi i logarytmy

Rozpoczniemy przegląd matematyki numerycznej od przyjrzenia się liczbom. *Liczby całkowite* to:

$$\dots, -3, -2, -1, 0, 1, 2, 3, \dots$$

(ujemne, zero lub dodatnie). *Liczba wymierna* jest określona stosunkiem (ilorazem) dwóch liczb całkowitych, p/q , gdzie p jest dodatnie. *Liczba rzeczywista* jest wartością, która ma *rozwinięcie dziesiętne*

$$x = n + 0.d_1 d_2 d_3 \dots, \quad (1)$$

gdzie n jest liczbą całkowitą, każde d_i jest cyfrą od 0 do 9 oraz ciąg cyfr nie kończy się nieskończonym wieloma dziewiątkami. Reprezentacja (1) oznacza, że

$$n + \frac{d_1}{10} + \frac{d_2}{100} + \dots + \frac{d_k}{10^k} \leq x < n + \frac{d_1}{10} + \frac{d_2}{100} + \dots + \frac{d_k}{10^k} + \frac{1}{10^k}, \quad (2)$$

dla wszystkich dodatnich k . Oto przykłady liczb rzeczywistych, które nie są wymierne:

$$\pi = 3.14159265358979 \dots, \text{ stosunek obwodu do średnicy koła;}$$

$$\phi = 1.61803398874989 \dots, \text{ złota proporcja } (1 + \sqrt{5})/2 \text{ (zob. punkt 1.2.8).}$$

Tabela najważniejszych stałych z dokładnością do czterdziestu miejsc po przecinku znajduje się w dodatku A. Nie będziemy omawiać własności dodawania, odejmowania, mnożenia, dzielenia i porównywania liczb rzeczywistych.

Skomplikowane problemy dotyczące liczb całkowitych rozwiązuje się za pomocą liczb rzeczywistych, a skomplikowane problemy dotyczące liczb rzeczywistych rozwiązuje się często za pomocą szerszej klasy wartości, tj. liczb zespolonych. *Liczeba zespolona* jest wielkością z postaci $z = x + iy$, gdzie x i y są liczbami rzeczywistymi, a i jest specjalną wartością spełniającą równanie $i^2 = -1$. Liczby x i y nazywamy odpowiednio *częścią rzeczywistą* i *częścią urojoną* liczby z , a jej *moduł* definiujemy jako:

$$|z| = \sqrt{x^2 + y^2}. \quad (3)$$

Liczba sprzężona z liczbą z jest $\bar{z} = x - iy$; mamy zatem $z\bar{z} = x^2 + y^2 = |z|^2$. Teoria liczb zespolonych jest w wielu miejscach prostsza i ładniejsza od teorii liczb rzeczywistych, ale powszechnie uważa się ją za skomplikowaną. Z tego powodu w niniejszej książce skupimy się na liczbach rzeczywistych poza przypadkami, kiedy ograniczanie się do nich rodzi niepotrzebne komplikacje.

Dla liczb rzeczywistych u i v spełniających warunek $u \leq v$ przedziałem domkniętym $[u..v]$ nazywamy zbiór liczb rzeczywistych x spełniających nierówności $u \leq x \leq v$. Przedziałem otwartym $(u..v)$ nazywamy zbiór liczb x spełniających nierówności $u < x < v$. Przedziały jednostronne domknięte $[u..v)$ i $(u..v]$ definiujemy analogicznie. Zezwalamy także, by u było równe $-\infty$ i/lub v było równe ∞ w otwartym końcu przedziału, co oznacza, że przedział nie ma dolnego lub górnego ograniczenia. I tak $(-\infty.. \infty)$ oznacza zbiór wszystkich liczb rzeczywistych, a $[0.. \infty)$ wszystkich liczb rzeczywistych nieujemnych.

W tym punkcie litera b będzie oznaczać dodatnią liczbę rzeczywistą. Jeśli n jest całkowite, to b^n definiujemy za pomocą znanych reguł:

$$b^0 = 1, \quad b^n = b^{n-1}b, \quad \text{gdy } n > 0, \quad b^n = b^{n+1}/b, \quad \text{gdy } n < 0. \quad (4)$$

Łatwo jest udowodnić przez indukcję, że dla całkowitych x i y zachodzą prawa działań na potęgach:

$$b^{x+y} = b^x b^y, \quad (b^x)^y = b^{xy}. \quad (5)$$

Jeśli u jest dodatnią liczbą rzeczywistą, a m dodatnią liczbą całkowitą, to zawsze istnieje dokładnie jedna liczba rzeczywista v , taka że $v^m = u$; nazywamy ją *pierwiastkiem m-tego stopnia* z u i oznaczamy $v = \sqrt[m]{u}$.

Teraz zdefiniujemy b^r dla liczb wymiernych $r = p/q$:

$$b^{p/q} = \sqrt[q]{b^p}. \quad (6)$$

Ta definicja, pochodząca od Oresme (około 1360 roku), jest poprawna, ponieważ $b^{ap/aq} = b^{p/q}$, a prawa działań na potęgach obowiązują również dla dowolnych wymiernych x i y (zobacz ćwiczenie 9).

Na koniec zdefiniujemy b^x dla dowolnych rzeczywistych x . Założymy najpierw, że $b > 1$; jeśli x jest dane równaniem (1), to chcielibyśmy, żeby

$$b^{n+d_1/10+\dots+d_k/10^k} \leq b^x < b^{n+d_1/10+\dots+d_k/10^k+1/10^k}. \quad (7)$$

Powyższa nierówność określa dokładnie jedną dodatnią liczbę rzeczywistą b^x , gdyż różnica między wartościami ciągów ograniczających b^x w nierówności (7) wynosi $b^{n+d_1/10+\dots+d_k/10^k}(b^{1/10^k} - 1)$. Ćwiczenie 13 pokazuje, że ta różnica jest mniejsza niż $b^{n+1}(b - 1)/10^k$. Jeśli zatem weźmiemy dostatecznie duże k , to możemy wyznaczyć b^x z dowolną dokładnością.

Pozwala to na przykład stwierdzić, że

$$10^{0.30102999} = 1.9999999739\dots, \quad 10^{0.30103000} = 2.0000000199\dots; \quad (8)$$

skąd dla $b = 10$ i $x = 0.30102999\dots$ znamy wartość 10^x z dokładnością większą od jednej dziesięciomilionowej, (ale wciąż nie wiemy, czy rozwinięcie dziesiętne 10^x to $1.999\dots$, czy $2.000\dots$).

Dla $b < 1$ definiujemy $b^x = (1/b)^{-x}$, jeśli natomiast $b = 1$, to $1^x = 1$. Przy powyższych definicjach można udowodnić poprawność praw działania na potęgach (5) dla dowolnych liczb *rzeczywistych* x i y . Pomyśl zdefiniowania b^x został po raz pierwszy sformułowany przez Johna Wallisa (1655) i Izaaka Newtona (1669).

Dochodzimy do istotnego pytania. Przypuśćmy, że dana jest liczba rzeczywista y ; czy potrafimy znaleźć taką liczbę x , że $y = b^x$? Odpowiedź brzmi „tak” (jeśli $b \neq 1$), bo możemy skorzystać z nierówności (7) do wyznaczenia n i d_1, d_2, \dots , gdy mamy dane $y = b^x$. Otrzymana liczba x jest nazywana *logarymem liczby y przy podstawie b* i oznaczamy ją $x = \log_b y$. Na mocy tej definicji:

$$x = b^{\log_b x} = \log_b(b^x). \quad (9)$$

Na przykład z równości (8) wynika, że:

$$\log_{10} 2 = 0.30102999\dots \quad (10)$$

Z praw działań na potęgach wynika, że:

$$\log_b(xy) = \log_b x + \log_b y, \quad \text{gdy } x > 0, \quad y > 0 \quad (11)$$

oraz

$$\log_b(c^y) = y \log_b c, \quad \text{gdy } c > 0. \quad (12)$$

W równaniu (10) występuje *logarytm dziesiętny*, tzn. logarytm przy podstawie 10. Można się spodziewać, że w zagadnieniach dotyczących komputerów bardziej pożyteczny będzie *logarytm dwójkowy* (przy podstawie 2), gdyż większość komputerów wykorzystuje arytmetykę dwójkową. Rzeczywiście okaże się, że logarytmy dwójkowe są bardzo ważne, ale nie tylko z tego powodu. Powodem zasadniczym jest fakt, że w algorytmach zazwyczaj obliczenia rozgałęziają się

na dwie drogi. Logarytmy dwójkowe pojawiają się tak często, że warto mieć dla nich krótkie oznaczenie, dlatego zgodnie z pomysłem Edwarda M. Reingolda będziemy pisać

$$\lg x = \log_2 x.$$

Powstaje pytanie, czy istnieje związek pomiędzy $\lg x$ i $\log_{10} x$. Na szczęście tak, na mocy równań (9) i (12) możemy napisać:

$$\log_{10} x = \log_{10}(2^{\lg x}) = (\lg x)(\log_{10} 2).$$

Stąd $\lg x = \log_{10} x / \log_{10} 2$, a w ogólności:

$$\log_c x = \frac{\log_b x}{\log_b c}. \quad (14)$$

Równości (11), (12) i (14) są podstawowymi zasadami przekształcania logarytmów.

Okazuje się, że ani 10, ani 2 w wielu przypadkach nie jest najwygodniejszą podstawą logarytmu. Istnieje pewna liczba rzeczywista, oznaczana przez $e = 2.718281828459045\dots$, dla której logarytmy mają najprostsze własności. Logarytm przy podstawie e jest nazywany *logarytmem naturalnym* i oznaczamy go

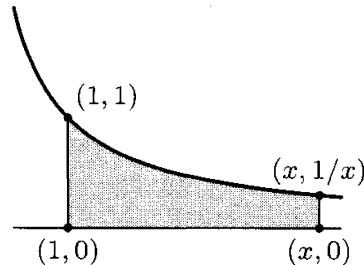
$$\ln x = \log_e x. \quad (15)$$

Ta wzięta znikąd definicja (w zasadzie nie określiliśmy e) raczej nie uderza Czytelnika swoją „naturalnością”; okaże się jednak, że im dłużej popracujemy z $\ln x$, tym bardziej naturalny wyda się ten logarytm. John Napier odkrył logarytm naturalny (z niewielkimi odstępstwami, a także nie zauważyszy związku z potęgami) przed 1590 rokiem, na wiele lat przed odkryciem jakichkolwiek innych logarytmów. Następujące dwa przykłady, udowadniane w każdym podręczniku analizy matematycznej, rzucają nieco światła na sprawę „naturalności” logarytmu Napiera. (a) Na rysunku 6 pole szarej figury wynosi $\ln x$. (b) Jeśli bank dopisuje do lokaty odsetki na poziomie r , naliczane co pół roku, to z każdej złotówki klient otrzyma $(1+r/2)^2$ złotych; jeśli są dopisywane co kwartał, to otrzyma $(1+r/4)^4$ złotych; jeśli co dzień, to $(1+r/365)^{365}$ złotych. Jeśli odsetki byłyby dopisywane *w sposób ciągły*, to klient otrzymałby dokładnie e^r złotych z każdej złotówki (pomijamy błędy zaokrągleń). W wieku komputerów wielu bankierów rzeczywiście osiągnęło tę granicę.

Ciekawa historia logarytmów i potęg jest opowiedziana w serii artykułów F. Cajoriego, *AMM* 20 (1913), 5–14, 35–47, 75–84, 107–117, 148–151, 173–182, 205–210.

Zakończymy ten rozdział rozważaniami na temat *obliczania* logarytmów. Jedna metoda wynika wprost z nierówności (7): Jeśli przyjmiemy $b^x = y$ i podniemy nierówności do 10^k -tej potęgi, otrzymamy

$$b^m \leq y^{10^k} < b^{m+1}, \quad (16)$$



Rys. 6. Logarytm naturalny.

dla pewnej liczby całkowitej m . Aby zatem otrzymać logarytm liczby y , podnosimy y do tej olbrzymiej potęgi i wyznaczamy, między którymi potęgami $(m, m+1)$ liczby b znajduje się wynik; wówczas $m/10^k$ jest odpowiedzią dla k cyfr dziesiętnych.

Niewielka modyfikacja tej niepraktycznej metody prowadzi do prostej i rozsądnej procedury. Pokażemy, jak obliczyć $\log_{10} x$ i przedstawić odpowiedź w systemie dwójkowym jako

$$\log_{10} x = n + b_1/2 + b_2/4 + b_3/8 + \dots \quad (17)$$

Najpierw przesuwamy przecinek dziesiętny w lewo lub w prawo w celu uzyskania $1 \leq x/10^n < 10$; w ten sposób wyznaczamy część całkowitą – n . By znaleźć b_1, b_2, \dots , przyjmujemy $x_0 = x/10^n$ i dla $k \geq 1$ mamy

$$\begin{aligned} b_k &= 0, & x_k &= x_{k-1}^2, & \text{gdy } &x_{k-1}^2 < 10; \\ b_k &= 1, & x_k &= x_{k-1}^2/10, & \text{gdy } &x_{k-1}^2 \geq 10. \end{aligned} \quad (18)$$

Poprawność tej procedury wynika z faktu

$$1 \leq x_k = x^{2^k}/10^{2^k(n+b_1/2+\dots+b_k/2^k)} < 10, \quad (19)$$

dla $k = 0, 1, 2, \dots$, co łatwo pokazać przez indukcję.

Oczywiście w praktyce pracujemy ze skończoną dokładnością, zatem nie możemy dokładnie obliczyć $x_k = x_{k-1}^2$; wartość zostanie *zaokrąglona* (lub *obcięta*) z dokładnością do pewnej liczby miejsc dziesiętnych. Oto przykładowe obliczenie $\log_{10} 2$ przy zaokrąglaniu do czterech miejsc po przecinku:

$$\begin{array}{llllll} x_0 &= 2.000; & & & & & \\ x_1 &= 4.000, & b_1 &= 0; & x_6 &= 1.845, & b_6 &= 1; \\ x_2 &= 1.600, & b_2 &= 1; & x_7 &= 3.404, & b_7 &= 0; \\ x_3 &= 2.560, & b_3 &= 0; & x_8 &= 1.159, & b_8 &= 1; \\ x_4 &= 6.554, & b_4 &= 0; & x_9 &= 1.343, & b_9 &= 0; \\ x_5 &= 4.295, & b_5 &= 1; & x_{10} &= 1.804, & b_{10} &= 0; & \text{itd.} \end{array}$$

Błąd w obliczeniach został rozpropagowany; prawdziwa zaokrąglona wartość wynosi 1.798. Z tego powodu b_{19} zostanie obliczone niedokładnie i otrzymamy liczbę dwójkową 0.01001101000010000011..., co odpowiada liczbie dziesiętnej 0.301031..., a nie wartości podanej w równaniu (10).

Przy rozważaniu podobnych metod trzeba koniecznie zwrócić uwagę na błąd w obliczeniach. Ćwiczenie 27 pokazuje górne ograniczenie błędu; wnioskujemy, że w przypadku czterech miejsc po przecinku, błąd w wartości logarytmu będzie zawsze mniejszy niż 0.00044. Nasza odpowiedź jest bardziej dokładna głównie dlatego, że x_0, x_1, x_2 i x_3 zostały obliczone *dokładnie*.

Ta metoda jest prosta i interesująca, ale zapewne nie jest najlepszym sposobem obliczania logarytmów na komputerze. Inna metoda jest podana w ćwiczeniu 25.

ĆWICZENIA

1. [00] Jaka jest najmniejsza dodatnia liczba wymierna?
 2. [00] Czy $1 + 0.239999999\dots$ jest rozwinięciem dziesiętnym?
 3. [02] Ile wynosi $(-3)^{-3}$?
 - ▶ 4. [05] Ile wynosi $(0.125)^{-2/3}$?
 5. [05] Definiujemy liczby rzeczywiste za pomocą rozwinięcia dziesiętnego. W jaki sposób moglibyśmy zdefiniować je za pomocą rozwinięcia dwójkowego? Sformułuj definicję mogącą zastąpić równanie (2).
 6. [10] Niech $x = m + 0.d_1d_2\dots$ i $y = n + 0.e_1e_2\dots$ będą liczbami rzeczywistymi. Podaj zasadę rozstrzygania, czy $x = y$, $x < y$, $x > y$, korzystającą z rozwinięcia dziesiętnego.
 7. [M23] Udowodnij prawa działań na potęgach dla całkowitych x i y , wychodząc od definicji (4).
 8. [25] Niech m będzie dodatnią liczbą całkowitą. Udowodnij, że każda dodatnia liczba całkowita u ma dokładnie jeden pierwiastek stopnia m , podając metodę znajdowania wartości n i kolejnych cyfr d_1, d_2, \dots w rozwinięciu dziesiętnym pierwiastka.
 9. [M23] Udowodnij prawa działań na potęgach dla wymiernych x i y , zakładając, że zachodzą one dla całkowitych x i y .
 10. [18] Udowodnij, że $\log_{10} 2$ nie jest liczbą wymierną.
 - ▶ 11. [10] Jeśli $b = 10$ i $x \approx \log_{10} 2$, to do ilu miejsc po przecinku musimy znać wartość x , by wyznaczyć cyfry trzech pierwszych miejsc po przecinku rozwinięcia dziesiętnego b^x ? (Uwaga: Możesz posłużyć się wynikiem ćwiczenia 10).
 12. [02] Wyjaśnij, dlaczego równość (10) wynika z równości (8).
 - ▶ 13. [M23] (a) Dla x będącego dodatnią liczbą rzeczywistą i n będącego dodatnią liczbą całkowitą udowodnij nierówność $\sqrt[n]{1+x} - 1 \leq x/n$. (b) Wykorzystaj ten fakt do uzasadnienia uwag występujących w tekście przy nierówności (7).
 14. [15] Udowodnij równość (12).
 15. [10] Udowodnij lub obal
- $$\log_b x/y = \log_b x - \log_b y, \quad \text{gdy } x, y > 0.$$
16. [00] Jak można wyrazić $\log_{10} x$ za pomocą $\ln x$ i $\ln 10$?
 - ▶ 17. [05] Ile wynosi $\lg 32$; $\log_\pi \pi$; $\ln e$; $\log_b 1$? $\log_b (-1)$?
 18. [10] Udowodnij lub obal: $\log_8 x = \frac{1}{2} \lg x$.
 - ▶ 19. [20] Jeśli n jest liczbą całkowitą, której reprezentacja dziesiętna ma 14 cyfr, to czy n zmieści się w słowie maszynowym o długości 47 bitów plus bit znaku?
 20. [10] Czy jest jakiś prosty związek między $\log_{10} 2$ a $\log_2 10$?
 21. [15] (Logarytmy logarytmów) Wyraź $\log_b \log_b x$ za pomocą $\ln \ln x$, $\ln \ln b$ i $\ln b$.
 - ▶ 22. [20] (R. W. Hamming) Udowodnij, że

$$\lg x \approx \ln x + \log_{10} x,$$

z błędem mniejszym niż 1%! (Stąd z tablic logarytmów naturalnych i dziesiętnych można odczytać także przybliżone wartości logarytmów dwójkowych).

23. [M25] Podaj dowód geometryczny równania $\ln xy = \ln x + \ln y$, opierając się na rysunku 6.

24. [15] Wyjaśnij, jak zmodyfikować metodę znajdowania logarytmów dziesiętnych podaną na końcu tego rozdziału, by można było obliczać logarytmy dwójkowe.

25. [22] Przypuśćmy, że mamy komputer liczący dwójkowo i liczbę x , $1 \leq x < 2$. Pokaż, że poniższy algorytm, w którym występują wyłącznie operacje przesunięcia bitowego, dodawania i odejmowania w liczbie proporcjonalnej do liczby wymaganych miejsc po przecinku, można wykorzystać do wyznaczania przybliżenia $y = \log_b x$:

- L1.** [Inicjowanie] Przyjmij $y \leftarrow 0$, $z \leftarrow x$ przesunięte w prawo o 1 bit, $k \leftarrow 1$.
- L2.** [Czy koniec?] Jeśli $x = 1$, to zatrzymaj się.
- L3.** [Porównanie] Jeśli $x - 1 < 1$, to przyjmij $z \leftarrow z$ przesunięte w prawo o 1 bit, $k \leftarrow k + 1$ i powtórz niniejszy krok.
- L4.** [Redukowanie] Przyjmij $x \leftarrow x - z$, $z \leftarrow z$ przesunięte w prawo o k bitów, $y \leftarrow y + \log_b(2^k/(2^k - 1))$ i przejdź do kroku L2. ■

[Uwagi: Ta metoda jest bardzo podobna do metody dzielenia stosowanej w sprzęcie liczącym. Autorem pomysłu był Henry Briggs; użył on tej metody (w wersji dziesiętnej, a nie dwójkowej) do obliczenia tablic logarytmicznych, opublikowanych w 1624 roku. Potrzebujemy dodatkowej tablicy stałych $\log_b 2$, $\log_b(4/3)$, $\log_b(8/7)$ itd., dla tylu wartości, ile cyfr po przecinku ma liczba w reprezentacji maszynowej. W algorytmie występują zamierzone błędy obliczeń, pojawiające się przy przesunięciach bitowych, stąd w końcu x zredukuje się do jedynki i algorytm się zatrzyma. Celem tego ćwiczenia jest pokazanie, dlaczego algorytm się zatrzyma i dlaczego oblicza przybliżenie $\log_b x$].

26. [M27] Wyznacz górne ograniczenie dokładności algorytmu z poprzedniego ćwiczenia, mając daną dokładność operacji arytmetycznych.

► **27.** [M25] Rozważ metodę obliczania $\log_{10} x$ opisaną w tekście. Niech x'_k oznacza obliczone przybliżenie x_k , wyznaczone następująco: $x(1 - \delta) \leq 10^n x'_0 \leq x(1 + \epsilon)$. Niech dodatkowo przy wyznaczaniu x'_k za pomocą równości (18) zamiast $(x'_{k-1})^2$ będzie używana wielkość y_k , gdzie $(x'_{k-1})^2(1 - \delta) \leq y_k \leq (x'_{k-1})^2(1 + \epsilon)$ oraz $1 \leq y_k < 100$. Symbole δ i ϵ oznaczają małe stałe odzwierciedlające dolny i górny błąd powstały w wyniku zaokrąglania lub obcinania wartości. Pokaż, że jeżeli $\log' x$ oznacza wynik obliczeń, to po k krokach mamy

$$\log_{10} x + 2 \log_{10}(1 - \delta) - 1/2^k < \log' x \leq \log_{10} x + 2 \log_{10}(1 + \epsilon).$$

28. [M30] (R. Feynman) Opracuj metodę obliczania b^x dla $0 \leq x < 1$ wyłącznie przy użyciu przesunięć bitowych, dodawania i odejmowania (podobną do algorytmu z ćwiczenia 25) i przeanalizuj jej dokładność.

29. [HM20] Niech x będzie liczbą rzeczywistą większą od 1. (a) Dla jakich liczb rzeczywistych $b > 1$ wartość $b \log_b x$ jest najmniejsza? (b) Dla jakich liczb całkowitych $b > 1$? (c) Dla jakich liczb całkowitych $b > 1$ wartość $(b + 1) \log_b x$ jest najmniejsza?

1.2.3. Sumy i iloczyny

Niech a_1, a_2, \dots będzie dowolnym ciągiem liczbowym. Często interesują nas sumy $a_1 + a_2 + \dots + a_n$, które wygodnie zapisywać w jednej z poniższych (równoważnych) notacji:

$$\sum_{j=1}^n a_j \quad \text{lub} \quad \sum_{1 \leq j \leq n} a_j. \quad (1)$$

Jeśli n jest zerem, to przyjmujemy, że wartość sumy wynosi zero. W ogólności, gdy $R(j)$ jest dowolną relacją zależną od j , symbol

$$\sum_{R(j)} a_j \quad (2)$$

oznacza sumę wszystkich a_j , gdzie j jest liczbą całkowitą spełniającą warunek $R(j)$. Jeśli nie ma takiej liczby, to zapis (2) oznacza zero. Litera j w (1) i (2) jest nazywana *indeksem próbnym* lub *zmienną indeksową*, wprowadzoną wyłącznie na potrzeby notacji. W roli zmiennej indeksowej występują zazwyczaj litery i, j, k, m, n, r, s, t (czasami z ozdobnikami). Wielkie znaki sumy, takie jak we wzorach (1) i (2), można zapisać zwięźle jako $\sum_{j=1}^n a_j$ lub $\sum_{R(j)} a_j$. Wykorzystanie znaku \sum i zmiennych indeksowych do zapisu sumowania o oznaczonych granicach zapoczątkował J. Fourier w 1820 roku.

Ścisłe rzecz biorąc, zapis $\sum_{1 \leq j \leq n} a_j$ jest wieloznaczny, gdyż nie określa, czy sumowanie odbywa się względem zmiennej j czy n . W tym konkretnym przypadku interpretowanie go jako sumy względem $n \geq j$ byłoby trochę niemądre. Można jednak podać sensowne przykłady, w których zmienna indeksowa nie jest jasno określona, na przykład $\sum_{j \leq k} \binom{j+k}{2j-k}$. W takich przypadkach kontekst musi określać, która zmienna jest zmienną indeksową, a która ma znaczenie również poza znakiem sumy. Suma z poprzedniego zdania byłaby zapewne użyta tylko wtedy, gdy albo j , albo k (ale nie jednocześnie) miałyby znaczenie „na zewnątrz”.

W większości przypadków będziemy używać notacji (2) wyłącznie wtedy, gdy suma jest skończona, tj. gdy jedynie skończenie wiele wartości j spełnia $R(j)$ przy $a_j \neq 0$. Do obliczania sum nieskończonych, na przykład:

$$\sum_{j=1}^{\infty} a_j = \sum_{j \geq 1} a_j = a_1 + a_2 + a_3 + \dots$$

z nieskończonym wieloma składnikami niezerowymi, konieczne jest zastosowanie metod analizy matematycznej. Dokładne znaczenie zapisu (2) jest następujące:

$$\sum_{R(j)} a_j = \left(\lim_{n \rightarrow \infty} \sum_{\substack{R(j) \\ -n \leq j < 0}} a_j \right) + \left(\lim_{n \rightarrow \infty} \sum_{\substack{R(j) \\ 0 \leq j < n}} a_j \right), \quad (3)$$

jeżeli obie granice istnieją. Jeśli którakolwiek z granic nie istnieje, nieskończona suma również nie istnieje; mówimy wtedy, że jest *rozbieżna*.

Jeśli pod znakiem \sum zapisane są dwa lub więcej warunki, jak w (3), to znaczy że *wszystkie* mają być spełnione.

Znajomość czterech prostych przekształceń sum umożliwia rozwiązywanie wielu problemów. Zajmiemy się teraz omówieniem tych przekształceń.

a) *Prawo rozdzielności* dla iloczynów sum:

$$\left(\sum_{R(i)} a_i \right) \left(\sum_{S(j)} b_j \right) = \sum_{R(i)} \left(\sum_{S(j)} a_i b_j \right). \quad (4)$$

Aby zrozumieć to prawo, rozważmy przykład:

$$\begin{aligned} \left(\sum_{i=1}^2 a_i \right) \left(\sum_{j=1}^3 b_j \right) &= (a_1 + a_2)(b_1 + b_2 + b_3) \\ &= (a_1 b_1 + a_1 b_2 + a_1 b_3) + (a_2 b_1 + a_2 b_2 + a_2 b_3) \\ &= \sum_{i=1}^2 \left(\sum_{j=1}^3 a_i b_j \right). \end{aligned}$$

Zazwyczaj opuszczamy nawiasy po prawej stronie (4); podwójne sumowanie $\sum_{R(i)} (\sum_{S(j)} a_{ij})$ zapisujemy krótko jako $\sum_{R(i)} \sum_{S(j)} a_{ij}$.

b) *Zamiana zmiennych:*

$$\sum_{R(i)} a_i = \sum_{R(j)} a_j = \sum_{R(p(j))} a_{p(j)}. \quad (5)$$

W tej równości są zawarte dwa rodzaje przekształceń. W pierwszym przypadku zmieniamy jedynie nazwę zmiennej indeksowej z i na j . Drugi przypadek jest bardziej interesujący: $p(j)$ jest funkcją j , będącą *permutacją* zakresu sumowania. Bardziej szczegółowo, dla każdego i spełniającego $R(i)$ musi istnieć dokładnie jedna liczba całkowita j spełniająca $p(j) = i$. Ten warunek jest prawdziwy dla przypadków $p(j) = c + j$ i $p(j) = c - j$, gdzie c jest liczbą całkowitą, która nie zależy od j . A te właśnie przypadki najczęściej pojawiają się w praktyce. Na przykład:

$$\sum_{1 \leq j \leq n} a_j = \sum_{1 \leq j-1 \leq n} a_{j-1} = \sum_{2 \leq j \leq n+1} a_{j-1}. \quad (6)$$

Zachęcamy Czytelnika do uważnego przestudiowania powyższego przykładu.

Nie zawsze można zastąpić j przez $p(j)$ w sumie *nieskończonej*. Przekształcenie jest poprawne, gdy $p(j) = c \pm j$, jak powyżej, ale w innych przypadkach trzeba zachować ostrożność. [Zobacz na przykład: T. M. Apostol, *Mathematical Analysis* (Reading, Mass.: Addison-Wesley, 1957), rozdział 12. Warunkiem wystarczającym do zapewnienia poprawności (5) dla dowolnej permutacji liczb całkowitych $p(j)$ jest istnienie $\sum_{R(j)} |a_j|$].

c) *Zamiana kolejności sumowania:*

$$\sum_{R(i)} \sum_{S(j)} a_{ij} = \sum_{S(j)} \sum_{R(i)} a_{ij}. \quad (7)$$

Rozważmy bardzo prosty przypadek tego równania:

$$\sum_{R(i)} \sum_{j=1}^2 a_{ij} = \sum_{R(i)} (a_{i1} + a_{i2}),$$

$$\sum_{j=1}^2 \sum_{R(i)} a_{ij} = \sum_{R(i)} a_{i1} + \sum_{R(i)} a_{i2}.$$

Na mocy (7) te dwie sumy są równe; nie oznacza to nic więcej niż

$$\sum_{R(i)} (b_i + c_i) = \sum_{R(i)} b_i + \sum_{R(i)} c_i, \quad (8)$$

gdzie $b_i = a_{i1}$ i $c_i = a_{i2}$.

Operacja zmiany kolejności sumowania jest niesamowicie pozytyczna, gdyż często zdarza się, że znamy prostą postać $\sum_{R(i)} a_{ij}$, ale nie $\sum_{S(j)} a_{ij}$. Często musimy także zmieniać kolejność sumowania w bardziej ogólnych przypadkach, gdy relacja S zależy zarówno od i , jak i od j . Wówczas możemy oznaczać ją „ $S(i, j)$ ”. Zamiany kolejności sumowania można teoretycznie dokonać zawsze, w najgorszym wypadku w następujący sposób:

$$\sum_{R(i)} \sum_{S(i,j)} a_{ij} = \sum_{S'(j)} \sum_{R'(i,j)} a_{ij}, \quad (9)$$

gdzie $S'(j)$ jest warunkiem „istnieje liczba całkowita i , że $R(i)$ i $S(i, j)$ są równocześnie prawdziwe”, natomiast $R'(i)$ jest warunkiem: „ $R(i)$ i $S(i, j)$ są jednocześnie prawdziwe”. Na przykład dla sumowania $\sum_{i=1}^n \sum_{j=1}^i a_{ij}$ warunek $S'(j)$ przybiera postać „istnieje taka liczba całkowita i , że $1 \leq i \leq n$ i $1 \leq j \leq i$ ”, czyli $1 \leq j \leq n$. Natomiast $R'(i, j)$ jest warunkiem: „ $1 \leq i \leq n$ i $1 \leq j \leq i$ ”, czyli $j \leq i \leq n$. Stąd

$$\sum_{i=1}^n \sum_{j=1}^i a_{ij} = \sum_{j=1}^n \sum_{i=j}^n a_{ij}. \quad (10)$$

[Uwaga: Jak w przypadku (b), operacja zmiany kolejności sumowania *nie zawsze jest poprawna dla szeregów nieskończonych*. Jeżeli szereg jest *zbieżny bezwzględnie*, tj. jeśli istnieje $\sum_{R(i)} \sum_{S(j)} |a_{ij}|$, to można wykazać, że (7) i (9) są poprawne. Również jeśli *którykolwiek* z warunków $R(i)$ lub $S(j)$ wyznacza sumę *skończoną* w (7), a przy tym wszystkie pojawiające się sumy nieskończone są zbieżne, to zamiana jest dopuszczalna. W szczególności (8) można zastosować zawsze do zbieżnych sum nieskończonych].

d) *Manipulowanie dziedziną*. Jeśli $R(j)$ i $S(j)$ są dowolnymi warunkami, to:

$$\sum_{R(j)} a_j + \sum_{S(j)} a_j = \sum_{R(j) \text{ lub } S(j)} a_j + \sum_{R(j) \text{ i } S(j)} a_j. \quad (11)$$

Na przykład:

$$\sum_{1 \leq j \leq m} a_j + \sum_{m \leq j \leq n} a_j = \left(\sum_{1 \leq j \leq n} a_j \right) + a_m, \quad (12)$$

przy założeniu, że $1 \leq m \leq n$. W tym przypadku „ $R(j)$ i $S(j)$ ” to po prostu „ $j = m$ ”, zredukowaliśmy zatem drugą sumę do „ a_m ”. W wielu zastosowaniach równości (11) albo $R(j)$ i $S(j)$ są spełnione równocześnie tylko dla jednej lub dwóch wartości j , albo $R(j)$ i $S(j)$ nie mogą być jednocześnie prawdziwe dla tego samego j . W tym ostatnim przypadku druga suma po prawej stronie równości (11) po prostu znika.

Skoro poznaliśmy cztery podstawowe sposoby przekształcania sum, przyjrzymy się przykładom ich zastosowania.

Przykład 1.

$$\begin{aligned}
 \sum_{0 \leq j \leq n} a_j &= \sum_{\substack{0 \leq j \leq n \\ j \text{ parzyste}}} a_j + \sum_{\substack{0 \leq j \leq n \\ j \text{ nieparzyste}}} a_j && \text{reguła (d)} \\
 &= \sum_{\substack{0 \leq 2j \leq n \\ 2j \text{ parzyste}}} a_{2j} + \sum_{\substack{0 \leq 2j+1 \leq n \\ 2j+1 \text{ nieparzyste}}} a_{2j+1} && \text{reguła (b)} \\
 &= \sum_{0 \leq j \leq n/2} a_{2j} + \sum_{0 \leq j < n/2} a_{2j+1}.
 \end{aligned}$$

Ostatni krok to jedynie przekształcenia warunków pod znakami \sum .

Przykład 2. Niech

$$\begin{aligned}
 S_1 &= \sum_{i=0}^n \sum_{j=0}^i a_i a_j = \sum_{j=0}^n \sum_{i=j}^n a_i a_j && \text{reguła (c) [zobacz (10)]} \\
 &= \sum_{i=0}^n \sum_{j=i}^n a_i a_j && \text{reguła (b),}
 \end{aligned}$$

Zamieniliśmy miejscami i i j oraz skorzystaliśmy z faktu, że $a_j a_i = a_i a_j$. Oznaczając otrzymaną sumę przez S_2 , dostajemy:

$$\begin{aligned}
 2S_1 &= S_1 + S_2 = \sum_{i=0}^n \left(\sum_{j=0}^i a_i a_j + \sum_{j=i}^n a_i a_j \right) && \text{na mocy (8)} \\
 &= \sum_{i=0}^n \left(\left(\sum_{j=0}^n a_i a_j \right) + a_i a_i \right) && \text{reguła (d) [zobacz (12)]} \\
 &= \sum_{i=0}^n \sum_{j=0}^n a_i a_j + \sum_{i=0}^n a_i a_i && \text{na mocy (8)} \\
 &= \left(\sum_{i=0}^n a_i \right) \left(\sum_{j=0}^n a_j \right) + \left(\sum_{i=0}^n a_i^2 \right) && \text{na mocy (a)} \\
 &= \left(\sum_{i=0}^n a_i \right)^2 + \left(\sum_{i=0}^n a_i^2 \right) && \text{na mocy (b).}
 \end{aligned}$$

Wyprowadziliśmy ważną tożsamość:

$$\sum_{i=0}^n \sum_{j=0}^i a_i a_j = \frac{1}{2} \left(\left(\sum_{i=0}^n a_i \right)^2 + \left(\sum_{i=0}^n a_i^2 \right) \right). \quad (13)$$

Przykład 3 (*Suma ciągu geometrycznego*). Założmy, że $x \neq 1$, $n \geq 0$. Wówczas

$$\begin{aligned}
 a + ax + \cdots + ax^n &= \sum_{0 \leq j \leq n} ax^j && \text{z definicji (2)} \\
 &= a + \sum_{1 \leq j \leq n} ax^j && \text{reguła (d)} \\
 &= a + x \sum_{1 \leq j \leq n} ax^{j-1} && \text{szczególny przypadek (a)} \\
 &= a + x \sum_{0 \leq j \leq n-1} ax^j && \text{reguła (b) [zobacz (6)]} \\
 &= a + x \sum_{0 \leq j \leq n} ax^j - ax^{n+1} && \text{reguła (d).}
 \end{aligned}$$

Przyrównując pierwsze wyrażenie z ostatnim, otrzymujemy

$$(1-x) \sum_{0 \leq j \leq n} ax^j = a - ax^{n+1};$$

skąd dostajemy znany wzór

$$\sum_{0 \leq j \leq n} ax^j = a \left(\frac{1 - x^{n+1}}{1 - x} \right). \quad (14)$$

Przykład 4 (*Suma ciągu arytmetycznego*). Założmy, że $n \geq 0$. Wówczas

$$\begin{aligned}
 a + (a+b) + \cdots + (a+nb) &= \sum_{0 \leq j \leq n} (a+bj) && \text{z definicji (2)} \\
 &= \sum_{0 \leq n-j \leq n} (a+b(n-j)) && \text{reguła (b)} \\
 &= \sum_{0 \leq j \leq n} (a+bn-bj) && \text{upraszczamy} \\
 &= \sum_{0 \leq j \leq n} (2a+bn) - \sum_{0 \leq j \leq n} (a+bj) && \text{na mocy (8)} \\
 &= (n+1)(2a+bn) - \sum_{0 \leq j \leq n} (a+bj),
 \end{aligned}$$

gdzię pierwsza suma to dodawanie $(n+1)$ składników, które nie zależą od j .

Przyrównując pierwsze i ostatnie wyrażenie oraz dzieląc obustronnie przez 2, otrzymujemy

$$\sum_{0 \leq j \leq n} (a + bj) = a(n+1) + \frac{1}{2}bn(n+1). \quad (15)$$

To jest $n+1$ razy $\frac{1}{2}(a + (a + bn))$, czyli liczba wyrazów razy średnia arytmetyczna pierwszego i ostatniego wyrazu.

Zauważmy, że ważne równości (13), (14) i (15) wyprowadziliśmy wyłącznie za pomocą prostych przekształceń sum. Wiele podręczników zamieszcza te wzory wraz z dowodami przez *indukcję*. Indukcja jest oczywiście metodą nieskazitelnie poprawną, nie daje jednak cienia szansy na zrozumienie, jak wyprowadzić wzór bez zgadywania. W analizie algorytmów spotykamy się z setkami sum, które nie przystają do żadnego oczywistego wzorca. Częstokroć przekształcając te sumy, można uzyskać rozwiązanie bez zgadywania.

Wiele przekształceń sum i innych wzorów istotnie się upraszcza, gdy zastosujemy tzw. *notację nawiasową*:

$$[\text{zdanie}] = \begin{cases} 1, & \text{jeżeli zdanie jest prawdziwe;} \\ 0, & \text{jeżeli zdanie jest fałszywe.} \end{cases} \quad (16)$$

Możemy teraz napisać, na przykład:

$$\sum_{R(j)} a_j = \sum_j a_j [R(j)], \quad (17)$$

gdzie sumowanie po prawej stronie równości przebiega względem *wszystkich* liczb całkowitych j . Wyrazy tej nieskończonej sumy są równe zero, gdy $R(j)$ jest fałszywe. (Zakładamy, że a_j jest określone dla wszystkich j).

Notacja nawiasowa umożliwia wyprowadzenie reguły (b) w ciekawy sposób z reguł (a) i (c):

$$\begin{aligned} \sum_{R(p(j))} a_{p(j)} &= \sum_j a_{p(j)} [R(p(j))] \\ &= \sum_j \sum_i a_i [R(i)] [i = p(j)] \\ &= \sum_i a_i [R(i)] \sum_j [i = p(j)]. \end{aligned} \quad (18)$$

Przy założeniu, że p jest permutacją zakresu sumowania (porównaj (5)), suma względem j jest równa 1, jeżeli $R(i)$ jest prawdziwe. Zostaje więc $\sum_i a_i [R(i)]$, czyli $\sum_{R(i)} a_i$. To dowodzi równości (5). Postać (18) jest równoważna wyjściowej sumie nawet wtedy, gdy p nie jest permutacją zakresu sumowania.

Najbardziej znanym szczególnym przypadkiem notacji nawiasowej jest *delta Kroneckera*

$$\delta_{ij} = [i = j] = \begin{cases} 1, & \text{gdy } i = j, \\ 0, & \text{gdy } i \neq j, \end{cases} \quad (19)$$

wprowadzona przez Leopolda Kroneckera w 1868 roku. W 1962 roku K. E. Iverson wprowadził notacje ogólniejsze, jak na przykład (16). Z tego powodu (16) nazywa się często *konwencją Iversona*. [Zobacz D. E. Knuth, *AMM* **99** (1992), 403–422].

Istnieje notacja dla iloczynów, analogiczna do naszej notacji dla sum. Symbol

$$\prod_{R(j)} a_j \quad (20)$$

oznacza iloczyn wszystkich a_j , dla których liczba całkowita j spełnia $R(j)$. Jeżeli takich liczb nie ma, to przyjmuje się, że iloczyn ma wartość 1 (*nie* 0).

Operacje (b), (c) i (d) z niewielkimi modyfikacjami przekładają się z „ \sum ” na „ \prod ”. W ćwiczeniach na końcu tego punktu znajduje się kilka przykładów wykorzystania powyższej notacji.

Zamykamy ten rozdział opisem innej wygodnej notacji dla sum ciągów. Pojedynczy znak \sum może być użyty z jednym lub więcej warunkami zawierającymi wiele zmiennych indeksowych. Oznacza to, że sumujemy względem wszystkich kombinacji wartości zmiennych spełniających warunki. Na przykład

$$\sum_{0 \leq i \leq n} \sum_{0 \leq j \leq n} a_{ij} = \sum_{0 \leq i, j \leq n} a_{ij}; \quad \sum_{0 \leq i \leq n} \sum_{0 \leq j \leq i} a_{ij} = \sum_{0 \leq j \leq i \leq n} a_{ij}.$$

W tej notacji zmienne indeksowe nie są uszeregowane, dzięki czemu można w nowy sposób wyprowadzić (10):

$$\begin{aligned} \sum_{i=1}^n \sum_{j=1}^i a_{ij} &= \sum_{i,j} a_{ij} [1 \leq i \leq n] [1 \leq j \leq i] = \sum_{i,j} a_{ij} [1 \leq j \leq n] [j \leq i \leq n] \\ &= \sum_{j=1}^n \sum_{i=j}^n a_{ij}, \end{aligned}$$

pamiętając, że: $[1 \leq i \leq n][1 \leq j \leq i] = [1 \leq j \leq i \leq n] = [1 \leq j \leq n][j \leq i \leq n]$. Ogólniejsza równość (9) wynika w podobny sposób z tożsamości

$$[R(i)] [S(i, j)] = [R(i) \text{ i } S(i, j)] = [S'(j)] [R'(i, j)]. \quad (21)$$

Kolejnym przykładem ilustrującym użyteczność sumowania względem wielu zmiennych jest

$$\sum_{\substack{j_1 + \dots + j_n = n \\ j_1 \geq \dots \geq j_n \geq 0}} a_{j_1 \dots j_n}, \quad (22)$$

gdzie a jest zmienną indeksowaną n -elementowymi krotkami. Na przykład dla $n = 5$ powyższy zapis oznacza

$$a_{11111} + a_{21110} + a_{22110} + a_{31100} + a_{32000} + a_{41000} + a_{50000}.$$

(Zobacz komentarz na temat podziałów liczby w punkcie 1.2.1).

ĆWICZENIA – zestaw pierwszy

1. [01] Co oznacza zapis $\sum_{1 \leq j \leq n} a_j$, gdy $n = 3.14$?

2. [10] Rozpisz sumy

$$\sum_{0 \leq n \leq 5} \frac{1}{2n+1}$$

oraz

$$\sum_{0 \leq n^2 \leq 5} \frac{1}{2n^2+1}$$

bez używania notacji „ \sum ”.

► **3.** [13] Wyjaśnij, dlaczego pomimo reguły (b) wyniki w poprzednim ćwiczeniu są różne.

4. [10] Zapisz obie strony równości (10) jako sumę sum dla $n = 3$ bez użycia notacji „ \sum ”.

► **5.** [HM20] Udowodnij, że reguła (a) zachodzi dla dowolnych zbieżnych szeregów nieskończonych.

6. [HM20] Udowodnij, że reguła (d) zachodzi dla dowolnych szeregów nieskończonych przy założeniu, że istnieją dowolne trzy z występujących tam czterech sum.

7. [HM23] Pokaż, że dla całkowitych c zachodzi $\sum_{R(j)} a_j = \sum_{R(c-j)} a_{c-j}$ nawet wtedy, gdy obie sumy są nieskończone.

8. [HM25] Znajdź przykład szeregu nieskończonego, dla którego (7) nie zachodzi.

► **9.** [05] Czy wyprowadzenie równania (14) jest poprawne też w przypadku $n = -1$?

10. [05] Czy wyprowadzenie równania (14) jest poprawne też w przypadku $n = -2$?

11. [03] Jak powinna wyglądać prawa strona równania (14) dla $x = 1$?

12. [10] Uprość $1 + \frac{1}{7} + \frac{1}{49} + \frac{1}{343} + \cdots + (\frac{1}{7})^n$.

13. [10] Oblicz $\sum_{j=m}^n j$ za pomocą równania (15) przy założeniu, że $m \leq n$.

14. [11] Korzystając z wyniku poprzedniego ćwiczenia, oblicz $\sum_{j=m}^n \sum_{k=r}^s jk$.

► **15.** [M22] Oblicz sumę $1 \times 2 + 2 \times 2^2 + 3 \times 2^3 + \cdots + n2^n$ dla małych wartości n . Czy dostrzegasz jakiś schemat? Jeśli nie, to odkryj go za pomocą przekształceń podobnych do użytych przy wyprowadzeniu równości (14).

16. [M22] Udowodnij, że

$$\sum_{j=0}^n jx^j = \frac{nx^{n+2} - (n+1)x^{n+1} + x}{(x-1)^2},$$

dla $x \neq 1$. Nie używaj indukcji.

► **17.** [M00] Niech S będzie pewnym zbiorem liczb naturalnych. Ile wynosi suma $\sum_{j \in S} 1$?

18. [M20] Pokaż, w jaki sposób zmienić kolejność sumowania, jak w równaniu (9), gdy $R(i)$ jest warunkiem „ n jest wielokrotnością i ”, a $S(i, j)$ jest warunkiem „ $1 \leq j < i$ ”.

19. [20] Ile wynosi $\sum_{j=m}^n (a_j - a_{j-1})$?

► 20. [25] Dr I. J. Matrix zaobserwował zadziwiający ciąg równości

$$9 \times 1 + 2 = 11, \quad 9 \times 12 + 3 = 111, \quad 9 \times 123 + 4 = 1111, \quad 9 \times 1234 + 5 = 11111.$$

- a) Zapisz odkrycie poczciwego naukowca za pomocą notacji „ \sum ”.
 - b) Twoja odpowiedź na punkt (a) zapewne zawiera liczbę 10 będącą podstawą systemu pozycyjnego. Uogólnij swój wzór, tak by można go było zastosować do dowolnej podstawy b .
 - c) Udowodnij swój wzór z punktu (b) za pomocą równości wyprowadzonych w tekście i/lub ćwiczeniu 16.
- 21. [M25] Wyprowadź regułę (d) z reguł (a) i (c), korzystając z notacji nawiasowej (16).
- 22. [20] Sformułuj odpowiedniki równań (5), (7), (8) i (11) dla *iloczynów* zamiast sum.
23. [10] Wyjaśnij, dlaczego w przypadku, gdy żadna liczba całkowita nie spełnia warunku $R(j)$, definiujemy $\sum_{R(j)} a_j$ i $\prod_{R(j)} a_j$ odpowiednio jako zero i jeden.
24. [20] Założmy, że $R(j)$ jest prawdziwe tylko dla skończenie wielu j . Udowodnij przez indukcję względem liczby argumentów spełniających $R(j)$, że $\log_b \prod_{R(j)} a_j = \sum_{R(j)} (\log_b a_j)$, przy założeniu $a_j > 0$.
- 25. [15] Przyjrzyj się poniższemu wyprowadzeniu. Czy wszystko jest w porządku?

$$\left(\sum_{i=1}^n a_i \right) \left(\sum_{j=1}^n \frac{1}{a_j} \right) = \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq n} \frac{a_i}{a_j} = \sum_{1 \leq i \leq n} \sum_{1 \leq i \leq n} \frac{a_i}{a_i} = \sum_{i=1}^n 1 = n.$$

26. [25] Pokaż, że $\prod_{i=0}^n \prod_{j=0}^i a_i a_j$ można wyrazić za pomocą $\prod_{i=0}^n a_i$, stosując przekształcenia sformułowane w ćwiczeniu 22.

27. [M20] Uogólnij wynik z ćwiczenia 1.2.1–9, dowodząc, że

$$\prod_{j=1}^n (1 - a_j) \geq 1 - \sum_{j=1}^n a_j,$$

przy założeniu $0 < a_j < 1$.

28. [M22] Znajdź prosty wzór na $\prod_{j=2}^n (1 - 1/j^2)$.

- 29. [M30] (a) Zapisz $\sum_{i=0}^n \sum_{j=0}^i \sum_{k=0}^j a_i a_j a_k$ za pomocą notacji „ \sum ” z wieloma zmiennymi jednocześnie, opisanej na końcu tego punktu. (b) Zapisz to samo wyrażenie za pomocą $\sum_{i=0}^n a_i$, $\sum_{i=0}^n a_i^2$ i $\sum_{i=0}^n a_i^3$ [zobacz (13)].

- 30. [M23] (J. Binet, 1812) Bez korzystania z indukcji udowodnij tożsamość

$$\left(\sum_{j=1}^n a_j x_j \right) \left(\sum_{j=1}^n b_j y_j \right) = \left(\sum_{j=1}^n a_j y_j \right) \left(\sum_{j=1}^n b_j x_j \right) + \sum_{1 \leq j < k \leq n} (a_j b_k - a_k b_j)(x_j y_k - x_k y_j).$$

[Istotny przypadek szczególny otrzymujemy, biorąc dowolne liczby zespolone $w_1, \dots, w_n, z_1, \dots, z_n$ i przyjmując, że $a_j = w_j$, $b_j = \bar{z}_j$, $x_j = \bar{w}_j$, $y_j = z_j$:

$$\left(\sum_{j=1}^n |w_j|^2 \right) \left(\sum_{j=1}^n |z_j|^2 \right) = \left| \sum_{j=1}^n w_j z_j \right|^2 + \sum_{1 \leq j < k \leq n} |w_j \bar{z}_k - w_k \bar{z}_j|^2.$$

Wyrazy $|w_j \bar{z}_k - w_k \bar{z}_j|^2$ są nieujemne, tak więc słynna nierówność Cauchy'ego-Schwarza

$$\left(\sum_{j=1}^n |w_j|^2 \right) \left(\sum_{j=1}^n |z_j|^2 \right) \geq \left| \sum_{j=1}^n w_j z_j \right|^2$$

wynika ze wzoru Bineta].

31. [M20] Korzystając ze wzoru Bineta, wyraź sumę $\sum_{1 \leq j < k \leq n} (u_j - u_k)(v_j - v_k)$ za pomocą $\sum_{j=1}^n u_j v_j$, $\sum_{j=1}^n u_j$ i $\sum_{j=1}^n v_j$.

32. [M20] Udowodnij, że:

$$\prod_{j=1}^n \sum_{i=1}^m a_{ij} = \sum_{1 \leq i_1, \dots, i_n \leq m} a_{i_1 1} \dots a_{i_n n}.$$

► **33.** [M30] Pewnego wieczoru dr Matrix odkrył wzory, które mogą okazać się jeszcze bardziej zaskakujące niż te z ćwiczenia 20:

$$\frac{1}{(a-b)(a-c)} + \frac{1}{(b-a)(b-c)} + \frac{1}{(c-a)(c-b)} = 0,$$

$$\frac{a}{(a-b)(a-c)} + \frac{b}{(b-a)(b-c)} + \frac{c}{(c-a)(c-b)} = 0,$$

$$\frac{a^2}{(a-b)(a-c)} + \frac{b^2}{(b-a)(b-c)} + \frac{c^2}{(c-a)(c-b)} = 1,$$

$$\frac{a^3}{(a-b)(a-c)} + \frac{b^3}{(b-a)(b-c)} + \frac{c^3}{(c-a)(c-b)} = a + b + c.$$

Udowodnij, że te wzory są szczególnymi przypadkami ogólnej reguły. Przy założeniu, że liczby x_1, x_2, \dots, x_n są parami różne, pokaż, że

$$\sum_{j=1}^n \left(x_j^r \middle/ \prod_{\substack{1 \leq k \leq n \\ k \neq j}} (x_j - x_k) \right) = \begin{cases} 0, & \text{jeśli } 0 \leq r < n-1; \\ 1, & \text{jeśli } r = n-1; \\ \sum_{j=1}^n x_j, & \text{jeśli } r = n. \end{cases}$$

34. [M25] Udowodnij, że:

$$\sum_{k=1}^n \frac{\prod_{1 \leq r \leq n, r \neq k} (x+k-r)}{\prod_{1 \leq r \leq n, r \neq k} (k-r)} = 1,$$

przy założeniu, że $1 \leq m \leq n$, a x jest dowolne. Na przykład, jeśli $n = 4$ i $m = 2$, to

$$\frac{x(x-2)(x-3)}{(-1)(-2)(-3)} + \frac{(x+1)(x-1)(x-2)}{(1)(-1)(-2)} + \frac{(x+2)x(x-1)}{(2)(1)(-1)} + \frac{(x+3)(x+1)x}{(3)(2)(1)} = 1.$$

35. [HM20] Symbol $\sup_{R(j)} a_j$ oznacza kres góry elementów a_j i jest podobny do notacji „ Σ ” i „ Π ”. (Jeżeli $R(j)$ jest spełnione tylko dla skończonej liczby różnych wartości j , to używa się oznaczenia $\max_{R(j)} a_j$). Pokaż, w jaki sposób reguły (a), (b), (c) i (d) można przystosować do manipulowania tq notacją. W szczególności omów poniższy odpowiednik reguły (a):

$$(\sup_{R(i)} a_i) + (\sup_{S(j)} b_j) = \sup_{R(i)} (\sup_{S(j)} (a_i + b_j)).$$

Podaj rozsądную definicję $\sup_{R(j)} a_j$, gdy $R(j)$ nie jest spełnione dla żadnego j .

ĆWICZENIA – zestaw drugi

Wyznaczniki i macierze. Poniższe problemy przeznaczone są dla Czytelnika, który przynajmniej zetknął się z podstawami teorii wyznaczników i macierzy. Wyznacznik można sprytnie obliczyć jako złożenie operacji: (a) dzielenia wiersza lub kolumny przez stałą, (b) dodawania wiersza (kolumny) pomnożonego przez stałą do innego wiersza (kolumny), (c) rozwijania za pomocą dopełnień algebraicznych. Najprostszą i najczęściej używaną wersją operacji (c) jest po prostu usunięcie całego pierwszego wiersza i całej pierwszej kolumny przy założeniu, że element w lewym górnym rogu jest równy +1, a pozostałe elementy albo w całym pierwszym wierszu, albo w całej pierwszej kolumnie, są zerami. W następnej kolejności obliczamy wyznacznik mniejszej macierzy. W ogólności, dopełnieniem algebraicznym elementu a_{ij} w macierzy $n \times n$ nazywamy iloczyn czynnika $(-1)^{i+j}$ i wartości wyznacznika macierzy $(n-1) \times (n-1)$ otrzymanej przez usunięcie wiersza i kolumny, w której znajdował się element a_{ij} . Wartość wyznacznika jest równa $\sum a_{ij} \cdot \text{dopełnienie algebraiczne}(a_{ij})$ przy ustalonej wartości jednej ze zmiennych i i j , gdy druga zmienna przebiega liczby całkowite od 1 do n .

Jeżeli (b_{ij}) jest macierzą odwrotną do (a_{ij}) , to wartość elementu b_{ij} jest równa dopełnieniu algebraicznemu elementu a_{ji} (*nie* a_{ij}), podzielonemu przez wyznacznik całej macierzy.

Poniższe macierze mają szczególne znaczenie:

macierz Vandermonde'a

$$a_{ij} = x_j^i$$

$$\begin{pmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & & & \vdots \\ x_1^n & x_2^n & \dots & x_n^n \end{pmatrix}$$

macierz kombinatoryczna

$$a_{ij} = y + \delta_{ij}x$$

$$\begin{pmatrix} x+y & y & \dots & y \\ y & x+y & \dots & y \\ \vdots & & & \vdots \\ y & y & \dots & x+y \end{pmatrix}$$

macierz Cauchy'ego

$$a_{ij} = 1/(x_i + y_j)$$

$$\begin{pmatrix} 1/(x_1 + y_1) & 1/(x_1 + y_2) & \dots & 1/(x_1 + y_n) \\ 1/(x_2 + y_1) & 1/(x_2 + y_2) & \dots & 1/(x_2 + y_n) \\ \vdots & & & \vdots \\ 1/(x_n + y_1) & 1/(x_n + y_2) & \dots & 1/(x_n + y_n) \end{pmatrix}$$

36. [M23] Pokaż, że wyznacznik macierzy kombinatorycznej jest równy $x^{n-1}(x+ny)$.

► 37. [M24] Pokaż, że wyznacznik macierzy Vandermonde'a jest równy

$$\prod_{1 \leq j \leq n} x_j \prod_{1 \leq i < j \leq n} (x_j - x_i).$$

► 38. [M25] Pokaż, że wyznacznik macierzy Cauchy'ego jest równy

$$\prod_{1 \leq i < j \leq n} (x_j - x_i)(y_j - y_i) / \prod_{1 \leq i, j \leq n} (x_i + y_j).$$

39. [M23] Pokaż, że macierz odwrotna do macierzy kombinatorycznej jest macierzą kombinatoryczną o elementach $b_{ij} = (-y + \delta_{ij}(x + ny))/x(x + ny)$.

40. [M24] Pokaż, że macierz o elementach:

$$b_{ij} = \left(\sum_{\substack{1 \leq k_1 < \dots < k_{n-j} \leq n \\ k_1, \dots, k_{n-j} \neq i}} (-1)^{j-1} x_{k_1} \dots x_{k_{n-j}} \right) \Big/ x_i \prod_{\substack{1 \leq k \leq n \\ k \neq i}} (x_k - x_i)$$

jest macierzą odwrotną do macierzy Vandermonde'a.

Nie przejmuj się skomplikowaną sumą w liczniku – to po prostu współczynnik przy x^{j-1} w wielomianie $(x_1 - x) \dots (x_n - x)/(x_i - x)$.

41. [M26] Pokaż, że macierz o elementach:

$$b_{ij} = \left(\prod_{1 \leq k \leq n} (x_j + y_k)(x_k + y_i) \right) \Big/ (x_j + y_i) \left(\prod_{\substack{1 \leq k \leq n \\ k \neq j}} (x_j - x_k) \right) \left(\prod_{\substack{1 \leq k \leq n \\ k \neq i}} (y_i - y_k) \right)$$

jest macierzą odwrotną do macierzy Cauchy'ego.

42. [M18] Ile wynosi suma wszystkich n^2 elementów macierzy odwrotnej do macierzy kombinatorycznej?

43. [M24] Ile wynosi suma wszystkich n^2 elementów macierzy odwrotnej do macierzy Vandermonde'a? [Wskazówka: Skorzystaj z ćwiczenia 33].

► **44.** [M26] Ile wynosi suma wszystkich n^2 elementów macierzy odwrotnej do macierzy Cauchy'ego?

► **45.** [M25] Macierz Hilberta, czasami zwana odcinkiem $n \times n$ nieskończonej macierzy Hilberta, jest zdefiniowana wzorem $a_{ij} = 1/(i + j - 1)$. Pokaż, że jest ona szczególnym przypadkiem macierzy Cauchy'ego, znajdź macierz do niej odwrotną i udowodnij, że elementy macierzy odwrotnej są liczbami całkowitymi, a ich suma wynosi n^2 . (Uwaga: Macierze Hilberta wykorzystywano do testowania algorytmów macierzowych, bo są numerycznie niestabilne, a znamy ich macierze odwrotne. Błąd jest jednak porównywanie *znanej* macierzy odwrotnej (rozwiązanie ćwiczenia) z *obliczoną* macierzą odwrotną, gdyż elementy macierzy odwracanej są na wstępnie zaokrąglane. Z powodu niestabilności macierz odwrotna przybliżonej macierzy Hilberta będzie nieco inna, niż w przypadku obliczeń bez zaokrągleń. Ponieważ elementami macierzy odwrotnej są liczby całkowite, a macierz odwrotna jest tak niestabilna jak macierz pierwotna, można dokładnie podać macierz odwrotną i próbować odwrócić ją jeszcze raz. Jest jednak drugi problem: liczby całkowite składające się na macierz odwrotną są dosyć duże). Rozwiązanie tego ćwiczenia wymaga znajomości podstawowych faktów na temat silni oraz współczynników dwumianowych, omawianych w punktach 1.2.5 i 1.2.6).

► **46.** [M30] Niech A będzie macierzą $m \times n$, a B macierzą $n \times m$. Niech symbol $A_{j_1 j_2 \dots j_m}$ dla $1 \leq j_1, j_2, \dots, j_m \leq n$ oznacza macierz $m \times m$ składającą się z kolumn j_1, \dots, j_m macierzy A , a symbol $B_{j_1 j_2 \dots j_m}$ oznacza macierz $m \times m$ składającą się z wierszy j_1, \dots, j_m macierzy B . Udowodnij tożsamość Bineta–Cauchy'ego

$$\det(AB) = \sum_{1 \leq j_1 < j_2 < \dots < j_m \leq n} \det(A_{j_1 j_2 \dots j_m}) \det(B_{j_1 j_2 \dots j_m}).$$

(Przyjrzyj się przypadkom szczególnym: (i) $m = n$, (ii) $m = 1$, (iii) $B = A^T$, (iv) $m > n$, (v) $m = 2$).

47. [M27] (C. Krattenthaler) Udowodnij, że

$$\det \begin{pmatrix} (x+q_2)(x+q_3) & (x+p_1)(x+q_3) & (x+p_1)(x+p_2) \\ (y+q_2)(y+q_3) & (y+p_1)(y+q_3) & (y+p_1)(y+p_2) \\ (z+q_2)(z+q_3) & (z+p_1)(z+q_3) & (z+p_1)(z+p_2) \end{pmatrix} = (x-y)(x-z)(y-z)(p_1-q_2)(p_1-q_3)(p_2-q_3)$$

i uogólnij tę równość do tożsamości dla wyznaczników macierzy $n \times n$ o $3n - 2$ zmiennych $x_1, \dots, x_n, p_1, \dots, p_{n-1}, q_2, \dots, q_n$. Porównaj otrzymany wzór z wynikiem ćwiczenia 38.

1.2.4. Funkcje całkowitoliczbowe i podstawy teorii liczb

Dla dowolnej liczby rzeczywistej x piszemy

- $\lfloor x \rfloor$ = największa liczba całkowita mniejsza lub równa x (*podłoga* x);
- $\lceil x \rceil$ = najmniejsza liczba całkowita większa lub równa x (*sufit* x).

Przed rokiem 1970 pierwszą lub drugą z tych funkcji (zazwyczaj pierwszą) oznaczano symbolem $[x]$. Powyższe oznaczenia, wprowadzone w latach sześćdziesiątych przez K. E. Iversona, są bardziej poręczne, gdyż w praktyce $\lfloor x \rfloor$ i $\lceil x \rceil$ występują zową częstotliwością. Funkcję $\lfloor x \rfloor$ nazywa się czasami funkcją *entier*, od francuskiego słowa oznaczającego liczbę całkowitą.

Łatwo można sprawdzić następujące wzory i przykłady:

$$\lfloor \sqrt{2} \rfloor = 1, \quad \lceil \sqrt{2} \rceil = 2, \quad \left\lfloor +\frac{1}{2} \right\rfloor = 0, \quad \left\lceil -\frac{1}{2} \right\rceil = 0, \quad \left\lfloor -\frac{1}{2} \right\rfloor = -1 \text{ (nie zero!)}$$

$\lfloor x \rfloor = \lfloor x \rfloor$ wtedy i tylko wtedy, gdy x jest całkowite,

$\lceil x \rceil = \lfloor x \rfloor + 1$ wtedy i tylko wtedy, gdy x nie jest całkowite;

$$\lfloor -x \rfloor = -\lceil x \rceil; \quad x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1.$$

Inne interesujące wzory zawierające „podłogi” i „sufity” są zawarte w ćwiczeniach na końcu tego rozdziału.

Dla dowolnych liczb rzeczywistych x i y definiujemy operację dwuargumentową:

$$x \bmod y = x - y \lfloor x/y \rfloor, \quad \text{gdy } y \neq 0; \quad x \bmod 0 = x. \quad (1)$$

Z definicji wynika, że jeśli $y \neq 0$, to

$$0 \leq \frac{x}{y} - \left\lfloor \frac{x}{y} \right\rfloor = \frac{x \bmod y}{y} < 1. \quad (2)$$

Stąd

- jeśli $y > 0$, to $0 \leq x \bmod y < y$;
- jeśli $y < 0$, to $0 \geq x \bmod y > y$;
- liczba $x - (x \bmod y)$ jest całkowitą wielokrotnością y .

Liczbe $x \bmod y$ nazywamy *resztą* z dzielenia x przez y . Podobnie liczbę $\lfloor x/y \rfloor$ nazywamy *ilorazem*.

Operacja „mod” dla całkowitych x i y wydaje się znajoma:

$$5 \bmod 3 = 2, \quad 18 \bmod 3 = 0, \quad -2 \bmod 3 = 1. \quad (3)$$

Równość $x \bmod y = 0$ zachodzi wtedy i tylko wtedy, gdy x jest wielokrotnością y , tj. gdy x jest podzielne przez y . Zapis $y \mid x$ (czytaj: „ y dzieli x ”) oznacza, że y jest dodatnią liczbą całkowitą i $x \bmod y = 0$.

Operację „mod” można zastosować również do x i y przybierających dowolne wartości rzeczywiste. Na przykład dla funkcji trygonometrycznych:

$$\tan x = \tan(x \bmod \pi).$$

Liczba $x \bmod 1$ to *część ułamkowa* liczby x ; na mocy (1) mamy

$$x = \lfloor x \rfloor + (x \bmod 1). \quad (4)$$

Autorzy prac z teorii liczb posługują się często skrótem „mod” w nieco innym sensie. Będziemy korzystać z następującej notacji dla wyrażenia teorioliczbowego pojęcia *przystawania* (*kongruencji*):

$$x \equiv y \pmod{z} \quad (5)$$

oznacza, że $x \bmod z = y \bmod z$. Jest to inny sposób wyrażenia faktu „ $x - y$ jest całkowitą wielokrotnością z ”. Wyrażenie (5) czytamy „ x przystaje do y modulo z ”.

Zajmijmy się teraz podstawowymi własnościami przystawania wykorzystanymi w tej książce w rozumowaniach teorioliczbowych. Zakładamy, że wszystkie zmienne w poniższych wzorach przyjmują wartości ze zbioru liczb całkowitych. O liczbach całkowitych x i y mówimy, że są *względnie pierwsze*, jeżeli nie mają wspólnego czynnika, tj. ich *największy wspólny dzielnik* jest równy 1. Ten związek będziemy oznaczać $x \perp y$. Pojęcie względnej pierwszości liczb jest powszechnie znane, na przykład mówiąc, że ułamka nie da się skrócić, mamy na myśli to, że jego licznik i mianownik są względnie pierwsze.

Prawo A. Jeżeli $a \equiv b$ i $x \equiv y$, to $a \pm x \equiv b \pm y$ i $ax \equiv by$ (modulo m).

Prawo B. Jeżeli $ax \equiv by$ i $a \equiv b$, i jeśli $a \perp m$, to $x \equiv y$ (modulo m).

Prawo C. $a \equiv b$ (modulo m) wtedy i tylko wtedy, gdy $an \equiv bn$ (modulo mn) dla $n \neq 0$.

Prawo D. Jeżeli $r \perp s$, to $a \equiv b$ (modulo rs) wtedy i tylko wtedy, gdy $a \equiv b$ (modulo r) i $a \equiv b$ (modulo s).

Prawo A stwierdza, że możemy dodawać, odejmować i mnożyć modulo m tak jak w zwykłym równaniu. Prawo B dotyczy operacji dzielenia i stwierdza, że gdy dzielnik jest względnie pierwszy z modułem, to możemy podzielić obustronnie. Prawa C i D dotyczą zmian modułu. Prawa pozostawiamy do udowodnienia w ramach ćwiczeń.

Z praw A i B wynika następujące ważne twierdzenie:

Twierdzenie F (*Twierdzenie Fermata, 1640*). *Jeśli p jest liczbą pierwszą, to $a^p \equiv a$ (modulo p) dla dowolnego całkowitego a .*

Dowód. Jeżeli a jest wielokrotnością p , to oczywiście $a^p \equiv 0 \equiv a$ (modulo p). Pozostaje rozpatrzyć przypadek $a \bmod p \neq 0$. Liczba p jest pierwsza, zatem $a \perp p$. Rozważmy p liczb

$$0 \bmod p, \quad a \bmod p, \quad 2a \bmod p, \quad \dots, \quad (p-1)a \bmod p. \quad (6)$$

Te liczby są *parami różne*, bo jeśli $ax \bmod p = ay \bmod p$, to z definicji (5) $ax \equiv ay$ (modulo p), zatem według prawa B $x \equiv y$ (modulo p).

Ze wzoru (6) dostajemy p nieujemnych różnych liczb całkowitych mniejszych od p . Pierwsza z nich to zero, a następnie to $1, 2, \dots, p-1$ w pewnym porządku. Stąd na mocy prawa A

$$(a)(2a) \dots ((p-1)a) \equiv 1 \cdot 2 \dots (p-1) \pmod{p}. \quad (7)$$

Mnożąc obie strony przez a , otrzymujemy

$$a^p(1 \cdot 2 \dots (p-1)) \equiv a(1 \cdot 2 \dots (p-1)) \pmod{p}; \quad (8)$$

co dowodzi twierdzenia, gdyż każdy z czynników $1, 2, \dots, p-1$ jest względnie pierwszy z p , może więc zostać skrócony na mocy prawa B. ■

ĆWICZENIA

1. [00] Ile wynosi $\lfloor 1.1 \rfloor, \lceil -1.1 \rceil, \lceil -1.1 \rceil, \lfloor 0.99999 \rfloor$ i $\lfloor \lg 35 \rfloor$?
 - ▶ 2. [01] Ile wynosi $\lceil \lfloor x \rfloor \rceil$?
 - a) $\lfloor x \rfloor < n$ wtedy i tylko wtedy, gdy $x < n$;
 - b) $n \leq \lfloor x \rfloor$ wtedy i tylko wtedy, gdy $n \leq x$;
 - c) $\lceil x \rceil \leq n$ wtedy i tylko wtedy, gdy $x \leq n$;
 - d) $n < \lceil x \rceil$ wtedy i tylko wtedy, gdy $n < x$;
 - e) $\lceil x \rceil = n$ wtedy i tylko wtedy, gdy $x - 1 < n \leq x$, oraz wtedy i tylko wtedy, gdy $n \leq x < n + 1$;
 - f) $\lceil x \rceil = n$ wtedy i tylko wtedy, gdy $x \leq n < x + 1$, oraz wtedy i tylko wtedy, gdy $n - 1 < x \leq n$.
- [*Powyższe wzory są niezmiernie istotne przy udowadnianiu własności dotyczących funkcji $\lfloor x \rfloor$ i $\lceil x \rceil$.*]
- ▶ 4. [M10] Skorzystaj z poprzedniego ćwiczenia, by udowodnić, że $\lfloor -x \rfloor = -\lceil x \rceil$.
 5. [16] Przy założeniu, że x jest liczbą rzeczywistą, napisz proste wyrażenie oznaczające x zaokrąglone do najbliższej liczby całkowitej. Zaokrąglenie ma dawać $\lfloor x \rfloor$ dla $x \bmod 1 < \frac{1}{2}$ oraz $\lceil x \rceil$ dla $x \bmod 1 \geq \frac{1}{2}$. Odpowiedź powinna zawierać się w jednym wzorze, który obejmuje oba przypadki. Omów reguły zaokrąglania wynikające z Twojego wzoru dla ujemnych x .
 - ▶ 6. [20] Które z poniższych równości są prawdziwe dla wszystkich dodatnich liczb całkowitych x ? (a) $\lfloor \sqrt{\lfloor x \rfloor} \rfloor = \lfloor \sqrt{x} \rfloor$; (b) $\lceil \sqrt{\lceil x \rceil} \rceil = \lceil \sqrt{x} \rceil$; (c) $\lceil \sqrt{\lfloor x \rfloor} \rceil = \lceil \sqrt{x} \rceil$.
 7. [M15] Pokaż, że $\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor$ i że równość zachodzi wtedy i tylko wtedy, gdy $x \bmod 1 + y \bmod 1 < 1$. Czy podobny wzór jest prawdziwy dla sufitów?

8. [00] Ile wynosi $100 \bmod 3$, $100 \bmod 7$, $-100 \bmod 7$, $-100 \bmod 0$?
9. [05] Ile wynosi $5 \bmod -3$, $18 \bmod -3$, $-2 \bmod -3$?
- 10. [10] Ile wynosi $1.1 \bmod 1$, $0.11 \bmod 0.1$, $0.11 \bmod -0.1$?
11. [00] Co w przyjętej notacji oznacza „ $x \equiv y \pmod{0}$ ”?
12. [00] Które liczby całkowite są względnie pierwsze z 1?
13. [M00] Przyjmujemy, że największy wspólny dzielnik zera i n jest równy $|n|$. Które liczby całkowite są względnie pierwsze z 0?
- 14. [12] Jeśli $x \bmod 3 = 2$ i $x \bmod 5 = 3$, to ile wynosi $x \bmod 15$?
15. [10] Udowodnij, że $z(x \bmod y) = (zx) \bmod (zy)$. [Jest to tzw. prawo rozdzielności, z którego wynika prawo C].
16. [M10] Załóżmy, że $y > 0$. Pokaż, że jeśli $(x - z)/y$ jest całkowite i $0 \leq z < y$, to $z = x \bmod y$.
17. [M15] Udowodnij prawo A wprost z definicji przystawania, a także udowodnij jedną z implikacji w prawie D: jeśli $a \equiv b \pmod{rs}$, to $a \equiv b \pmod{r}$ i $a \equiv b \pmod{s}$ (r i s oznaczają dowolne liczby całkowite).
18. [M15] Korzystając z prawa B, udowodnij pozostałą część prawa D: jeśli $a \equiv b \pmod{r}$ i $a \equiv b \pmod{s}$, to $a \equiv b \pmod{rs}$, pod warunkiem że $r \perp s$.
- 19. [M10] (Prawo odwrotności) Jeśli $n \perp m$, to istnieje taka liczba całkowita n' , że $nn' \equiv 1 \pmod{m}$. Udowodnij to prawo za pomocą zmodyfikowanego algorytmu Euklidesa (algorytm 1.2.1E).
20. [M15] Udowodnij prawo B, korzystając z prawa odwrotności i prawa A.
21. [M22] (Podstawowe twierdzenie arytmetyki) Skorzystaj z prawa B i ćwiczenia 1.2.1–5, by udowodnić, że każdą liczbę całkowitą $n > 1$ można w sposób jednoznaczny przedstawić w postaci iloczynu liczb pierwszych (z dokładnością do kolejności czynników). Innymi słowy, pokaż, że prawdziwą równość $n = p_1 p_2 \dots p_k$ można napisać dokładnie na jeden sposób, jeżeli każde p_j jest liczbą pierwszą oraz $p_1 \leq p_2 \leq \dots \leq p_k$.
- 22. [M10] Znajdź przykład na to, że prawo B nie musi zachodzić, gdy liczby a i m nie są względnie pierwsze.
23. [M10] Znajdź przykład na to, że prawo D nie musi zachodzić, gdy liczby r i s nie są względnie pierwsze.
- 24. [M20] Jak dalece można uogólnić prawa A, B, C i D po zastąpieniu liczb całkowitych liczbami rzeczywistymi?
25. [M02] Pokaż, że zgodnie z twierdzeniem F, $a^{p-1} \bmod p = [a \text{ nie jest wielokrotnością } p]$, jeśli p jest liczbą pierwszą.
26. [M15] Niech p będzie nieparzystą liczbą pierwszą i niech a będzie dowolną liczbą całkowitą. Zdefiniujmy $b = a^{(p-1)/2}$. Pokaż, że $b \bmod p$ jest równe 0, 1 lub $p - 1$. [Wskazówka: Zastanów się nad $(b+1)(b-1)$].
27. [M15] Niech dla dodatniej liczby całkowitej n symbol $\varphi(n)$ oznacza liczbę różnych wartości spośród $\{0, 1, \dots, n-1\}$, które są względnie pierwsze z n . Mamy $\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(3) = 2$, $\varphi(4) = 2$ itd. Pokaż, że $\varphi(p) = p - 1$, gdy p jest liczbą pierwszą. Oblicz $\varphi(p^e)$, gdzie e jest dodatnią liczbą całkowitą.
- 28. [M25] Pokaż, że metodę wykorzystaną w dowodzie twierdzenia F można zastosować w dowodzie tzw. twierdzenia Eulera: $a^{\varphi(m)} \equiv 1 \pmod{m}$, dla dowolnej dodatniej liczby całkowitej m , gdzie $a \perp m$. (W szczególności za liczbę n' w ćwiczeniu 19 można przyjąć $n^{\varphi(m)-1} \bmod m$).

29. [M22] O funkcji $f(n)$ określonej na liczbach całkowitych mówimy, że jest *multiplikatywna*, jeżeli $f(rs) = f(r)f(s)$ dla $r \perp s$. Pokaż, że następujące funkcje są multiplikatywne: (a) $f(n) = n^c$, gdzie c jest dowolną stałą; (b) $f(n) = [n \text{ nie dzieli się przez } k^2 \text{ dla dowolnego całkowitego } k > 1]$; (c) $f(n) = c^k$, gdzie k jest liczbą różnych pierwszych dzielników n ; (d) iloczyn dowolnych dwóch funkcji multiplikatywnych.

30. [M30] Udowodnij, że funkcja $\varphi(n)$ z ćwiczenia 27 jest multiplikatywna. Korzystając z tego faktu, oblicz $\varphi(1000000)$ i podaj prostą metodę obliczania $\varphi(n)$, gdy jest znany rozkład n na czynniki pierwsze.

31. [M22] Udowodnij, że jeśli $f(n)$ jest multiplikatywna, to $g(n) = \sum_{d \mid n} f(d)$ też.

32. [M18] Udowodnij tożsamość

$$\sum_{d \mid n} \sum_{c \mid d} f(c, d) = \sum_{c \mid n} \sum_{d \mid (n/c)} f(c, cd),$$

dla dowolnej funkcji $f(x, y)$.

33. [M18] Wiedząc, że m i n są liczbami całkowitymi, oblicz

$$(a) \left\lfloor \frac{1}{2}(n+m) \right\rfloor + \left\lfloor \frac{1}{2}(n-m+1) \right\rfloor; \quad (b) \left\lceil \frac{1}{2}(n+m) \right\rceil + \left\lceil \frac{1}{2}(n-m+1) \right\rceil.$$

(Warto zapamiętać te wzory dla $m = 0$).

► **34.** [M21] Jakie warunki potrzeba i wystarczy nałożyć na liczbę rzeczywistą $b > 1$, by równość $\lfloor \log_b x \rfloor = \lfloor \log_b \lfloor x \rfloor \rfloor$ zachodziła dla wszystkich liczb całkowitych $x \geq 1$?

► **35.** [M20] Udowodnij, że dla całkowitych m i n ($n > 0$) zachodzi równość

$$\lfloor (x+m)/n \rfloor = \lfloor (\lfloor x \rfloor + m)/n \rfloor$$

dla dowolnej rzeczywistej liczby x ($m = 0$ jest ważnym przypadkiem szczególnym). Czy analogiczne twierdzenie zachodzi dla sufitów?

36. [M23] Udowodnij, że $\sum_{k=1}^n \lfloor k/2 \rfloor = \lfloor n^2/4 \rfloor$. Oblicz $\sum_{k=1}^n \lceil k/2 \rceil$.

► **37.** [M30] Pokaż, że dla całkowitych m i n ($n > 0$) zachodzi

$$\sum_{0 \leq k < n} \left\lfloor \frac{mk+x}{n} \right\rfloor = \frac{(m-1)(n-1)}{2} + \frac{d-1}{2} + d \lfloor x/d \rfloor,$$

gdzie d jest największym wspólnym dzielnikiem m i n , a x jest dowolną liczbą rzeczywistą.

38. [M26] (E. Busche, 1909) Udowodnij, że dla dowolnych liczb rzeczywistych x oraz $y > 0$,

$$\sum_{0 \leq k < y} \left\lfloor x + \frac{k}{y} \right\rfloor = \lfloor xy + \lfloor x+1 \rfloor (\lceil y \rceil - y) \rfloor.$$

W szczególności, gdy x jest dodatnie i całkowite, mamy

$$\lfloor x \rfloor + \left\lfloor x + \frac{1}{n} \right\rfloor + \cdots + \left\lfloor x + \frac{n-1}{n} \right\rfloor = \lfloor nx \rfloor.$$

39. [HM35] Funkcję f , taką że dla dowolnej całkowitej liczby n zachodzi $f(x) + f(x + \frac{1}{n}) + \cdots + f(x + \frac{n-1}{n}) = f(nx)$, nazywamy *funkcją powielającą*. W poprzednim ćwiczeniu udowodniłeś, że $\lfloor x \rfloor$ jest powielająca. Pokaż, że następujące funkcje są powielające:

- a) $f(x) = x - \frac{1}{2}$;
- b) $f(x) = [\lfloor x \rfloor]$ (czyli $f(x)$ jest liczbą całkowitą);

- c) $f(x) = [x]$ jest dodatnią liczbą całkowitą;
- d) $f(x) = [\text{istnieje liczba rzeczywista } r \text{ oraz całkowita } m, \text{ że } x = r\pi + m]$;
- e) trzy takie funkcje, jak w punkcie (d), lecz z ograniczeniem, że r i/lub m przyjmują tylko wartości dodatnie;
- f) $f(x) = \log |2 \sin \pi x|$, gdy dopuśćmy wartość $f(x) = -\infty$;
- g) suma dowolnych funkcji powielających;
- h) funkcja powielająca pomnożona przez stałą;
- i) funkcje $g(x) = f(x - [x])$, gdzie $f(x)$ jest powielająca.

40. [HM46] Zbadaj klasę funkcji powielających; wyznacz wszystkie funkcje powielające szczególnych typów. Na przykład, czy funkcja (a) z ćwiczenia 39 jest jedyną ciągłą funkcją powielającą? Można zbadać także nieco szerszą klasę funkcji, dla których

$$f(x) + f\left(x + \frac{1}{n}\right) + \cdots + f\left(x + \frac{n-1}{n}\right) = a_n f(nx) + b_n.$$

Liczby a_n i b_n zależą od n , ale nie od x . Pochodne oraz całki (gdy $b_n = 0$) takich funkcji też należą do tej klasy. Jeśli dodatkowo będziemy wymagać, by $b_n = 0$, to otrzymamy wielomiany Bernoulliego, funkcje trygonometryczne $\cot \pi x$ i $\csc^2 \pi x$, a także uogólnioną funkcję dzeta Hurwitza $\zeta(s, x) = \sum_{k \geq 0} 1/(k+x)^s$ dla ustalonego s . Dla $b_n \neq 0$ wciąż mamy wiele znanych funkcji, jak na przykład funkcja psi.

41. [M23] Niech a_1, a_2, a_3, \dots będzie ciągiem 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, Znajdź wyrażenie opisujące a_n za pomocą n oraz podłóg i/lub sufitów.

42. [M24] (a) Udowodnij, że

$$\sum_{k=1}^n a_k = n a_n - \sum_{k=1}^{n-1} k(a_{k+1} - a_k), \quad \text{gdy } n > 0.$$

(b) Powyższy wzór przydaje się przy obliczaniu pewnych sum zawierających podłogę. Udowodnij, że jeśli $b \geq 2$ jest liczbą całkowitą, to

$$\sum_{k=1}^n \lfloor \log_b k \rfloor = (n+1) \lfloor \log_b n \rfloor - (b^{\lfloor \log_b n \rfloor + 1} - b)/(b-1).$$

43. [M23] Oblicz $\sum_{k=1}^n \lfloor \sqrt{k} \rfloor$.

44. [M24] Pokaż, że $\sum_{k \geq 0} \sum_{1 \leq j < b} \lfloor (n + jb^k)/b^{k+1} \rfloor = n$, jeśli b i n są liczbami całkowitymi, $n \geq 0$ i $b \geq 2$. Jaka jest wartość tej sumy dla $n < 0$?

► **45.** [M28] Odpowiedź do ćwiczenia 37 jest nieco zaskakująca, gdyż wynika z niej, że

$$\sum_{0 \leq k < n} \left\lfloor \frac{mk + x}{n} \right\rfloor = \sum_{0 \leq k < m} \left\lfloor \frac{nk + x}{m} \right\rfloor.$$

Powyższy „związek wzajemności” jest jednym z wielu podobnych wzorów (zobacz punkt 3.3.3). Pokaż, że dla dowolnej funkcji f zachodzi

$$\sum_{0 \leq j < n} f\left(\left\lfloor \frac{mj}{n} \right\rfloor\right) = \sum_{0 \leq r < m} \left\lceil \frac{rn}{m} \right\rceil (f(r-1) - f(r)) + nf(m-1).$$

W szczególności udowodnij, że

$$\sum_{0 \leq j < n} \binom{\lfloor mj/n \rfloor + 1}{k} + \sum_{0 \leq j < m} \left\lceil \frac{jn}{m} \right\rceil \binom{j}{k-1} = n \binom{m}{k}.$$

[Wskazówka: Rozważ podstawienie $r = \lfloor mj/n \rfloor$. Współczynniki dwumianowe $\binom{m}{k}$ są omówione w punkcie 1.2.6].

46. [M29] (Ogólne prawo wzajemności) Uogólnij wzór z ćwiczenia 45, tak by otrzymać wzór na sumę $\sum_{0 \leq j < \alpha n} f(\lfloor mj/n \rfloor)$, gdzie α jest dowolną dodatnią liczbą rzeczywistą.

► 47. [M31] Dla p będącego nieparzystą liczbą pierwszą symbol Legendre'a jest zdefiniowany jako $+1$, 0 lub -1 w zależności od tego, czy $q^{(p-1)/2} \pmod p$ wynosi 1 , 0 czy $p-1$. (Z ćwiczenia 26 wynika, że to jedyne możliwe wartości).

a) Wiedząc, że q nie jest wielokrotnością p , pokaż, że liczby

$$(-1)^{\lfloor 2kq/p \rfloor} (2kq \pmod p), \quad 0 < k < p/2,$$

są w pewnym porządku przystające do liczb $2, 4, \dots, p-1$ (modulo p). Stąd $\left(\frac{q}{p}\right) = (-1)^\sigma$, gdzie $\sigma = \sum_{0 \leq k < p/2} \lfloor 2kq/p \rfloor$.

b) Oblicz $\left(\frac{2}{p}\right)$, korzystając z (a).

c) Wiedząc, że q jest nieparzyste, pokaż, że $\sum_{0 \leq k < p/2} \lfloor 2kq/p \rfloor \equiv \sum_{0 \leq k < p/2} \lfloor kq/p \rfloor$ (modulo 2). [Wskazówka: Zastanów się nad $\lfloor (p-1-2k)q/p \rfloor$].

d) Skorzystaj z ogólnego prawa wzajemności z ćwiczenia 46 i wyprowadź prawo wzajemności kwadratowej, $\left(\frac{q}{p}\right)\left(\frac{p}{q}\right) = (-1)^{(p-1)(q-1)/4}$, dla p i q będących różnymi, nieparzystymi liczbami pierwszymi.

48. [M26] Udowodnij lub obal następujące tożsamości dla całkowitych n i m :

$$(a) \left\lfloor \frac{m+n-1}{n} \right\rfloor = \left\lceil \frac{m}{n} \right\rceil; \quad (b) \left\lfloor \frac{n+2-\lfloor n/25 \rfloor}{3} \right\rfloor = \left\lfloor \frac{8n+24}{25} \right\rfloor.$$

49. [M30] Założmy, że dana funkcja $f(x)$ o wartościach całkowitych spełnia warunki:
(i) $f(x+1) = f(x) + 1$, (ii) $f(x) = f(f(nx)/n)$ dla wszystkich dodatnich liczb całkowitych n . Udowodnij, że albo $f(x) = \lfloor x \rfloor$ dla wszystkich x wymiernych, albo $f(x) = \lceil x \rceil$ dla wszystkich x wymiernych.

1.2.5. Permutacje i silnia

Permutacją zbioru n elementowego nazywamy uporządkowany ciąg n elementów tego zbioru. Istnieje sześć permutacji zbioru 3-elementowego $\{a, b, c\}$:

$$a \ b \ c, \quad a \ c \ b, \quad b \ a \ c, \quad b \ c \ a, \quad c \ a \ b, \quad c \ b \ a. \quad (1)$$

Własności permutacji mają olbrzymie znaczenie dla analizy algorytmów i wiele ciekawych faktów na ich temat zostanie w tej książce wyprowadzonych*. Naszym pierwszym zadaniem będzie po prostu policzenie liczby permutacji: ile jest permutacji n elementów? Pierwszy element można wybrać na n sposobów. Po dokonaniu tego wyboru mamy już tylko $n-1$ możliwości wybrania innego elementu na następne miejsce. Stąd pierwsze dwa miejsca możemy obsadzić na $n(n-1)$ sposobów. Analogicznie trzeci element można wybrać na $n-2$ sposobów, mamy zatem $n(n-1)(n-2)$ możliwości obsadzenia pierwszych trzech miejsc.

* O permutacjach mówi się tak często, że Vaughan Pratt zaproponował, by słowo „permutacja” zastąpić krótkim „perm”. Upowszechnienie konwencji Pratta spowodowałoby zmniejszenie objętości książek informatycznych. A może i ceny by spadły?

W ogólności, jeżeli p_{nk} oznacza liczbę sposobów, na które można wybrać i ustawić k spośród n obiektów, to łatwo zauważyc, że

$$p_{nk} = n(n - 1) \dots (n - k + 1). \quad (2)$$

Stąd całkowita liczba permutacji wynosi $p_{nn} = n(n - 1) \dots (1)$.

Proces *generowania* wszystkich permutacji n obiektów w sposób indukcyjny (tzn. przy założeniu, że wygenerowaliśmy wszystkie permutacje $n - 1$ elementów) jest dla nas bardzo ważny. Przepiszmy (1), używając liczb $\{1, 2, 3\}$ w miejsce liter $\{a, b, c\}$. Nasze permutacje wyglądają teraz następująco:

$$1\ 2\ 3, \quad 1\ 3\ 2, \quad 2\ 1\ 3, \quad 2\ 3\ 1, \quad 3\ 1\ 2, \quad 3\ 2\ 1. \quad (3)$$

Zastanówmy się, jak na podstawie takich informacji zbudować wszystkie permutacje zbioru $\{1, 2, 3, 4\}$. Są dwie zasadnicze metody przejścia od permutacji $n - 1$ elementów do permutacji n elementów.

Metoda 1. Dla każdej permutacji $a_1 a_2 \dots a_{n-1}$ liczb $\{1, 2, \dots, n-1\}$ tworzymy n nowych permutacji poprzez wstawienie liczby n we wszystkie możliwe miejsca, co daje

$$n\ a_1\ a_2\ \dots\ a_{n-1}, \quad a_1\ n\ a_2\ \dots\ a_{n-1}, \quad \dots, \quad a_1\ a_2\ \dots\ n\ a_{n-1}, \quad a_1\ a_2\ \dots\ a_{n-1}\ n.$$

Na przykład dla permutacji $2\ 3\ 1$ w (3) dostaniemy $4\ 2\ 3\ 1, 2\ 4\ 3\ 1, 2\ 3\ 4\ 1, 2\ 3\ 1\ 4$. Jest jasne, że w ten sposób można otrzymać wszystkie permutacje n elementów oraz że żadna permutacja się nie powtórzy.

Metoda 2. Dla każdej permutacji $a_1 a_2 \dots a_{n-1}$ liczb $\{1, 2, \dots, n-1\}$ tworzymy n nowych permutacji w następujący sposób. Generujemy permutacje

$$a_1\ a_2\ \dots\ a_{n-1}\ \frac{1}{2}, \quad a_1\ a_2\ \dots\ a_{n-1}\ \frac{3}{2}, \quad \dots, \quad a_1\ a_2\ \dots\ a_{n-1}\ (n - \frac{1}{2}),$$

a następnie numerujemy elementy każdej z nich liczbami $\{1, 2, \dots, n\}$ z zachowaniem porządku. Na przykład z permutacji $2\ 3\ 1$ w (3) otrzymujemy

$$2\ 3\ 1\ \frac{1}{2}, \quad 2\ 3\ 1\ \frac{3}{2}, \quad 2\ 3\ 1\ \frac{5}{2}, \quad 2\ 3\ 1\ \frac{7}{2},$$

a po przemianowaniu

$$3\ 4\ 2\ 1, \quad 3\ 4\ 1\ 2, \quad 2\ 4\ 1\ 3, \quad 2\ 3\ 1\ 4.$$

Oto inny sposób opisania tego procesu. Bierzemy permutację $a_1 a_2 \dots a_{n-1}$ oraz liczbę k , $1 \leq k \leq n$. Następnie dodajemy jedynkę do wszystkich a_j , których wartość jest $\geq k$, otrzymując permutację $b_1 b_2 \dots b_{n-1}$ elementów $\{1, \dots, k-1, k+1, \dots, n\}$. Wówczas $b_1 b_2 \dots b_{n-1} k$ jest permutacją elementów $\{1, \dots, n\}$.

Znów jasne jest, że każdą permutację n elementów dostaniemy dokładnie raz. Wstawiając k na lewy koniec albo w dowolne inne ustalone miejsce, też oczywiście otrzymamy się poprawny wynik.

Jeżeli p_n jest liczbą permutacji n elementów, to z obu powyższych metod wynika, że $p_n = np_{n-1}$. Stąd dysponujemy dwoma kolejnymi dowodami, że $p_n = n(n - 1) \dots (1)$, jak zauważyliśmy w (2).

Wielkość p_n nazywamy *silnią* i zapisujemy

$$n! = 1 \cdot 2 \cdot \dots \cdot n = \prod_{k=1}^n k. \quad (4)$$

Nasza konwencja dotycząca pustych iloczynów (punkt 1.2.3) każe przyjąć wartość

$$0! = 1 \quad (5)$$

i przy tym założeniu podstawowa tożsamość

$$n! = (n - 1)! \cdot n \quad (6)$$

jest prawdziwa dla wszystkich dodatnich liczb całkowitych n .

Silnia pojawia się w informatyce tak często, że nie jest złym pomysłem zapamiętanie kilku jej wartości:

$$0! = 1, \quad 1! = 1, \quad 2! = 2, \quad 3! = 6, \quad 4! = 24, \quad 5! = 120.$$

Silnia rośnie bardzo szybko. Na przykład $1000!$ w zapisie dziesiętnym ma ponad 2500 cyfr.

Warto też zapamiętać, że $10! = 3\,628\,800$ lub chociaż mieć świadomość, że $10!$ to więcej niż 3.5 miliona. W pewnym sensie ta liczba wyznacza umowną granicę pomiędzy tym, co w praktyce da się policzyć na komputerze, a tym, czego się nie da. Jeżeli algorytm wymaga sprawdzenia ponad $10!$ przypadków, to czas pracy programu może okazać się niedopuszczalnie długi. Z drugiej strony, jeżeli sprawdzenie pojedynczego przypadku zajmuje na przykład milisekundę, to $10!$ przypadków można sprawdzić w czasie rzędu godziny. Te spostrzeżenia są oczywiście bardzo nieprecyzyjne, ale dają wyobrażenie o tym, co jest wykonalne.

Zwykła kolej rzeczy każe zastanowić się nad związkami $n!$ z innymi wielkościami matematycznymi. Czy można w jakiś sposób określić, jak duże jest $1000!$, bez uciekania się do pracowitego mnożenia według wzoru (4)? Odpowiedź na to pytanie znalazł James Stirling w swojej słynnej pracy *Methodus Differentialis* (1730), strona 137; otóż

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e} \right)^n. \quad (7)$$

Symbol „ \approx ” oznacza „równe w przybliżeniu”, zaś „ e ” jest podstawą logarytmu naturalnego, omówionego w punkcie 1.2.2. W podpunkcie 1.2.11.2 pokażemy dowód wzoru przybliżonego Stirlinga (7). Ćwiczenie 24 zawiera prostszy dowód nieco mniej dokładnego wyniku.

Jako przykład wykorzystania wzoru obliczymy

$$40320 = 8! \approx 4\sqrt{\pi} \left(\frac{8}{e} \right)^8 = 2^{26} \sqrt{\pi} e^{-8} \approx 67108864 \cdot 1.77245 \cdot 0.00033546 \approx 39902.$$

W tym przypadku błąd wynosi około 1 procenta. Dalej przekonamy się, że błąd względny wynosi w przybliżeniu $1/(12n)$.

Poza przybliżoną wartością wyznaczoną za pomocą wzoru (7) możemy stosunkowo łatwo poznać dokładną wartość $n!$ w rozkładzie na czynniki pierwsze.

Liczba pierwsza p jest dzielnikiem $n!$ w potędze

$$\mu = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \cdots = \sum_{k>0} \left\lfloor \frac{n}{p^k} \right\rfloor. \quad (8)$$

Na przykład, gdy $n = 1000$ i $p = 3$, to

$$\begin{aligned} \mu &= \left\lfloor \frac{1000}{3} \right\rfloor + \left\lfloor \frac{1000}{9} \right\rfloor + \left\lfloor \frac{1000}{27} \right\rfloor + \left\lfloor \frac{1000}{81} \right\rfloor + \left\lfloor \frac{1000}{243} \right\rfloor + \left\lfloor \frac{1000}{729} \right\rfloor \\ &= 333 + 111 + 37 + 12 + 4 + 1 = 498, \end{aligned}$$

zatem $1000!$ jest podzielne przez 3^{498} , ale nie przez 3^{499} . Choć wzór (8) ma postać nieskończoną sumy, to w istocie ta suma jest zawsze skończona dla dowolnych konkretnych wartości n i p , ponieważ od pewnego momentu składniki równają się zeru. Z ćwiczenia 1.2.4–35 wynika, że $\lfloor n/p^{k+1} \rfloor = \lfloor \lfloor n/p^k \rfloor / p \rfloor$. Znajomość tego faktu ułatwia obliczanie wzoru (8), bo wystarczy po prostu podzielić poprzedni wyraz przez p , ignorując resztę z dzielenia.

Równanie (8) wynika stąd, że $\lfloor n/p^k \rfloor$ jest równe liczbie elementów zbioru $\{1, 2, \dots, n\}$, które są wielokrotnościami p^k . Rozpatrując czynniki iloczynu (4), zauważymy, że każda liczba podzielna przez p^j , ale nie przez p^{j+1} , jest zliczana dokładnie j razy: raz w $\lfloor n/p \rfloor$, raz w $\lfloor n/p^2 \rfloor, \dots$, raz w $\lfloor n/p^j \rfloor$. To odpowiada wszystkim wystąpieniom p w rozkładzie $n!$ na czynniki pierwsze.

Powstaje kolejne pytanie: skoro zdefiniowaliśmy $n!$ dla n nieujemnych, to może silnia ma sens dla dowolnych n wymiernych, a nawet rzeczywistych? Ile wynosi $(\frac{1}{2})!?$ Zilustrujemy ten problem wprowadzając „słabnię”:

$$n? = 1 + 2 + \cdots + n = \sum_{k=1}^n k, \quad (9)$$

która jest analogiczna do silni, z tym że dodajemy, zamiast mnożyć. Umiemy policzyć sumę ciągu arytmetycznego z równości 1.2.3–(15):

$$n? = \frac{1}{2}n(n+1). \quad (10)$$

W tym wzorze kryje się sugestia uogólnienia „słabni” na dowolne n poprzez użycie wzoru (10) w miejsce (9). Stąd $(\frac{1}{2})? = \frac{3}{8}$.

Stirling kilkakrotnie próbował uogólnić $n!$ na niecałkowite wartości n . Rozwinął wzór (7) w sumę nieskończoną, ale niestety ta suma jest rozbieżna dla każdego n – można było uzyskać bardzo dobre przybliżenie, ale nie dokładną wartość. [Omówienie tej niecodziennej sytuacji zamieszczono w: K. Knopp, *Theory and Application of Infinite Series*, wyd. 2 (Glasgow: Blackie, 1951), 518–520, 527, 534].

Stirling próbował dalej, zauważając, że

$$\begin{aligned} n! &= 1 + \left(1 - \frac{1}{1!}\right)n + \left(1 - \frac{1}{1!} + \frac{1}{2!}\right)n(n-1) \\ &\quad + \left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!}\right)n(n-1)(n-2) + \cdots. \end{aligned} \quad (11)$$

(Udowodnimy ten wzór w następnym rozdziale). Nieskończona suma (11) jest w rzeczywistości skończona dla każdej nieujemnej liczby całkowitej n . Nie rozwiązuje to jednak problemu uogólnienia $n!$, bo nieskończona suma istnieje *wyłącznie* dla n nieujemnych i całkowitych (zobacz ćwiczenie 16).

Nie zniechęcony tym Stirling odkrył taki ciąg a_1, a_2, \dots , że

$$\ln n! = a_1 n + a_2 n(n-1) + \dots = \sum_{k \geq 0} a_{k+1} \prod_{0 \leq j \leq k} (n-j). \quad (12)$$

Wciąż nie potrafił udowodnić, że ta suma dobrze definiuje $n!$ dla wszystkich ułamkowych wartości n , ale potrafił wyznaczyć wartość $(\frac{1}{2})! = \sqrt{\pi}/2$.

Mniej więcej w tym samym czasie Leonard Euler rozważał ten sam problem i jako pierwszy znalazł właściwe uogólnienie:

$$n! = \lim_{m \rightarrow \infty} \frac{m^n m!}{(n+1)(n+2)\dots(n+m)}. \quad (13)$$

Euler opisał ten pomysł w liście do Christiana Goldbacha z 13 października 1729 roku. Jego wzór definiuje $n!$ dla dowolnej wartości n poza ujemnymi liczbami całkowitymi. Ćwiczenia 8 i 22 zawierają wyjaśnienie, dlaczego (13) jest właściwą definicją.

Prawie dwa wieki później, w 1900 roku, C. Hermite udowodnił, że wymyślony przez Stirlinga wzór (12) rzeczywiście definiuje $n!$ dla niecałkowitych n oraz że uogólnienia Eulera i Stirlinga są w istocie tym samym.

Dawnymi czasy silnię oznaczano na wiele sposobów. Euler pisał $[n]$, Gauss pisał Πn , a oznaczenia $\lfloor n \rfloor$ i $\lceil n \rceil$ były popularne w Anglii i we Włoszech. Oznaczenie $n!$, powszechnie używane dzisiaj dla całkowitych n , zostało wprowadzone przez względnie mało znanego matematyka Christiana Krampa w pracy dotyczącej algebry [Éléments d'Arithmétique Universelle (Cologne: 1808)].

Oznaczenie $n!$ jest jednak mniej popularne w sytuacjach, gdy n nie musi być liczbą całkowitą. Częstość używa się notacji pochodzącej od A. M. Legendre'a:

$$n! = \Gamma(n+1) = n\Gamma(n). \quad (14)$$

Funkcja $\Gamma(x)$ jest nazywana *funkcją gamma*, a z (13) otrzymujemy definicję

$$\Gamma(x) = \frac{x!}{x} = \lim_{m \rightarrow \infty} \frac{m^x m!}{x(x+1)(x+2)\dots(x+m)}. \quad (15)$$

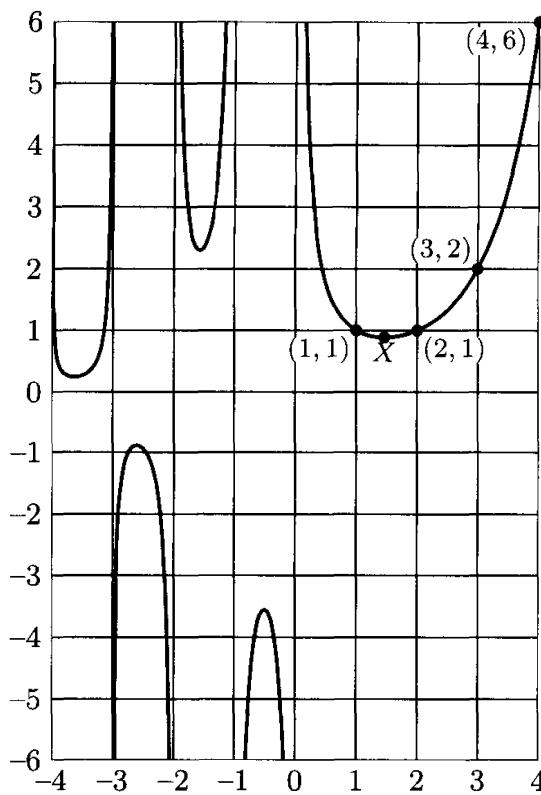
Wykres funkcji $\Gamma(x)$ widać na rysunku 7.

Równania (13) i (15) definiują silnię oraz funkcję gamma również dla argumentów zespolonych. Zasadniczo używamy jednak litery z zamiast n lub x , gdy mamy na myśli zmienną, która ma zarówno część rzeczywistą, jak i urojoną. Silnia i funkcja gamma są związane nie tylko wzorem $z! = \Gamma(z+1)$, ale także

$$(-z)! \Gamma(z) = \frac{\pi}{\sin \pi z}, \quad (16)$$

który zachodzi dla z nie będących liczbami całkowitymi (zobacz ćwiczenie 23).

Chociaż $\Gamma(z)$ ma umowną wartość nieskończoną, gdy z jest zerem lub ujemną liczbą całkowitą, to funkcja $1/\Gamma(z)$ jest dobrze określona dla wszystkich



Rys. 7. Funkcja $\Gamma(x) = (x - 1)!$, minimum lokalne w punkcie X ma współrzędne $(1.46163\ 21449\ 68362\ 34126\ 26595, 0.88560\ 31944\ 10888\ 70027\ 88159)$.

z zespolonych. (Zobacz ćwiczenie 1.2.7–24). W złożonych zastosowaniach funkcji gamma często korzysta się ze wzoru z całką po krzywej zamkniętej, podanego przez Hermanna Hankela:

$$\frac{1}{\Gamma(z)} = \frac{1}{2\pi i} \oint \frac{e^t dt}{t^z}; \quad (17)$$

całkowanie rozpoczyna się w $-\infty$, następnie okrąża się początek układu przeciwnie do ruchu wskazówek zegara i powraca do $-\infty$. [Zeitschrift für Math. und Physik 9 (1864), 1–21].

Wiele wzorów matematyki dyskretnej zawiera silniopodobne iloczyny znane jako *potęgi kroczące (ubywające i przyrastające)*. Liczby x^k i $x^{\bar{k}}$ (czytaj: „ x do k -tej ubywającej” i „ x do k -tej przyrastającej”) definiuje się dla dodatnich liczb całkowitych k :

$$x^k = x(x - 1) \dots (x - k + 1) = \prod_{j=0}^{k-1} (x - j); \quad (18)$$

$$x^{\bar{k}} = x(x + 1) \dots (x + k - 1) = \prod_{j=0}^{k-1} (x + j); \quad (19)$$

Stąd na przykład liczba p_{nk} z (2) to po prostu n^k . Zauważmy, że

$$x^{\bar{k}} = (x + k - 1)^k = (-1)^k (-x)^k. \quad (20)$$

Potęgi ubywające i przyrastające dla innych wartości k definiujemy jako

$$x^k = \frac{x!}{(x-k)!}, \quad x^{\bar{k}} = \frac{\Gamma(x+k)}{\Gamma(x)}. \quad (21)$$

[Oznaczenie $x^{\bar{k}}$ zawdzięczamy A. Capelliemu, *Giornale di Matematiche di Battaglini* **31** (1893), 291–313].

Zajmująca historia silni od czasów Stirlinga do współczesności przedstawiona jest w artykule P. J. Davisa, „Leonhard Euler’s integral: A historical profile of the gamma function”, *AMM* **66** (1959), 849–869. Zobacz także J. Dutka, *Archive for History of Exact Sciences* **31** (1984), 15–34.

ĆWICZENIA

1. [00] Na ile sposobów można potasować dużą talię kart (52 karty)?
 2. [10] Wyjaśnij dlaczego przy oznaczeniach jak w (2) zachodzi $p_{n(n-1)} = p_{nn}$.
 3. [10] Jakie permutacje zbioru $\{1, 2, 3, 4, 5\}$ otrzymamy za pomocą metod 1 i 2 z permutacji 3 1 2 4?
- 4. [13] Wiedząc, że $\log_{10} 1000! = 2567.60464\dots$, wyznacz liczbę cyfr w reprezentacji dziesiętnej $1000!$. Jaka jest *najbardziej znacząca cyfra*? Jaka jest *najmniej znacząca cyfra*?
5. [15] Oszacuj wartość $8!$ za pomocą dokładniejszej wersji wzoru aproksymacyjnego Stirlinga:

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \frac{1}{12n}\right).$$

- 6. [17] Korzystając z (8), zapisz $20!$ jako iloczyn czynników pierwszych.
7. [M10] Pokaż, że „uogólniona słabnia” w równaniu (10) spełnia równość $x? = x + (x - 1)$? dla wszystkich x rzeczywistych.
8. [HM15] Pokaż, że granica we wzorze (13) wynosi $n!$ dla nieujemnych liczb całkowitych n .
9. [M10] Wyznacz wartości $\Gamma(\frac{1}{2})$ i $\Gamma(-\frac{1}{2})$, wiedząc, że $(\frac{1}{2})! = \sqrt{\pi}/2$.
- 10. [HM20] Czy tożsamość $\Gamma(x+1) = x\Gamma(x)$ zachodzi dla wszystkich liczb rzeczywistych x ? (Zobacz ćwiczenie 7).
11. [M15] Niech reprezentacją dwójkową liczby n będzie $n = 2^{e_1} + 2^{e_2} + \dots + 2^{e_r}$, gdzie $e_1 > e_2 > \dots > e_r \geq 0$. Pokaż, że $n!$ jest podzielne przez 2^{n-r} , ale nie przez 2^{n-r+1} .
- 12. [M22] (A. Legendre, 1808) Uogólnijmy poprzednie ćwiczenie. Niech p będzie liczbą pierwszą i niech reprezentacją n w systemie o podstawie p będzie $n = a_k p^k + a_{k-1} p^{k-1} + \dots + a_1 p + a_0$. Wyraź μ z równości (8) w postaci prostego wzoru zawierającego n , p i czynniki a .
13. [M23] (*Twierdzenie Wilsona*; naprawdę pochodzące od Leibniza, 1682) Jeśli p jest liczbą pierwszą, to $(p-1)! \bmod p = p-1$. Udowodnij twierdzenie, ustawiając liczby $\{1, 2, \dots, p-1\}$ w pary, których iloczyn modulo p wynosi 1.
- 14. [M28] (L. Stickelberger, 1890) Przyjmijmy oznaczenia jak w ćwiczeniu 12. Możemy wyznaczyć reprezentację $n! \bmod p$ w systemie pozycyjnym o podstawie p dla dowolnego dodatniego n , otrzymując tym samym uogólnienie twierdzenia Wilsona. Udowodnij, że zachodzi $n!/p^\mu \equiv (-1)^\mu a_0! a_1! \dots a_k! \pmod{p}$.

15. [HM15] Permanent macierzy kwadratowej definiuje się za pomocą takiego samego rozwinięcia jak wyznacznik, z tym że składniki permanentu się dodaje, podczas gdy składniki wyznacznika na przemian dodaje się i odejmuje. Permanent macierzy

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

wynosi $aei + bfg + cdh + gec + hfa + idb$. Oblicz permanent macierzy

$$\begin{pmatrix} 1 \times 1 & 1 \times 2 & \dots & 1 \times n \\ 2 \times 1 & 2 \times 2 & \dots & 2 \times n \\ \vdots & \vdots & \ddots & \vdots \\ n \times 1 & n \times 2 & \dots & n \times n \end{pmatrix}$$

16. [HM15] Pokaż, że suma nieskończona w równaniu (11) nie jest zbieżna, jeżeli n nie jest nieujemną liczbą całkowitą.

17. [HM20] Udowodnij, że nieskończony iloczyn

$$\prod_{n \geq 1} \frac{(n + \alpha_1) \dots (n + \alpha_k)}{(n + \beta_1) \dots (n + \beta_k)}$$

równa się $\Gamma(1 + \beta_1) \dots \Gamma(1 + \beta_k) / \Gamma(1 + \alpha_1) \dots \Gamma(1 + \alpha_k)$, gdy $\alpha_1 + \dots + \alpha_k = \beta_1 + \dots + \beta_k$ oraz gdy żadne β nie jest ujemną liczbą całkowitą.

18. [M20] Załóżmy, że $\pi/2 = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \dots$. (To jest „iloczyn Wallisa”, wyprowadzony przez J. Wallisa w 1655 roku, udowodniony w ćwiczeniu 1.2.6–43). Korzystając z poprzedniego ćwiczenia, udowodnij, że $(\frac{1}{2})! = \sqrt{\pi}/2$.

19. [HM22] Oznaczmy przez $\Gamma_m(x)$ wartość stojącą pod „ $\lim_{m \rightarrow \infty}$ ” w równości (15). Udowodnij, że

$$\Gamma_m(x) = \int_0^m \left(1 - \frac{t}{m}\right)^m t^{x-1} dt = m^x \int_0^1 (1-t)^m t^{x-1} dt, \quad \text{dla } x > 0.$$

20. [HM21] Korzystając z poprzedniego ćwiczenia oraz z faktu $0 \leq e^{-t} - (1-t/m)^m \leq t^2 e^{-t}/m$ dla $0 \leq t \leq m$, pokaż, że $\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt$ dla $x > 0$.

21. [HM25] (L. F. A. Arbogast, 1800) Niech $D_x^k u$ oznacza k -tą pochodną funkcji u ze względu na x . Reguła łańcuchowa mówi, że $D_x^1 w = D_u^1 w D_x^1 u$. Dla drugich pochodnych mamy $D_x^2 w = D_u^2 w (D_x^1 u)^2 + D_u^1 w D_x^2 u$. Pokaż, że zachodzi ogólny wzór

$$D_x^n w = \sum_{j=0}^n \sum_{\substack{k_1+k_2+\dots+k_n=j \\ k_1+2k_2+\dots+nk_n=n \\ k_1, k_2, \dots, k_n \geq 0}} D_u^j w \frac{n!}{k_1! (1!)^{k_1} \dots k_n! (n!)^{k_n}} (D_x^1 u)^{k_1} \dots (D_x^n u)^{k_n}.$$

► **22.** [HM20] Spróbuj postawić się w sytuacji Eulera, szukającego sposobu uogólnienia $n!$ na niecałkowite wartości n . Ponieważ $(n + \frac{1}{2})!/n!$ razy $((n + \frac{1}{2}) + \frac{1}{2})!/(n + \frac{1}{2})!$ równa się $(n + 1)!/n! = n + 1$, to wydaje się naturalne, że $(n + \frac{1}{2})!/n!$ powinno się w przybliżeniu równać \sqrt{n} . Podobnie $(n + \frac{1}{3})!/n!$ powinno wynosić $\approx \sqrt[3]{n}$. Zaproponuj hipotezę dotyczącą wartości $(n + x)!/n!$, gdy n dąży do nieskończoności. Czy Twoja hipoteza jest prawdziwa dla całkowitych x ? Czy mówi cokolwiek o wartościach $x!$, gdy x nie jest liczbą całkowitą?

23. [HM20] Udowodnij wzór (16), wiedząc, że $\pi z \prod_{n=1}^{\infty} (1 - z^2/n^2) = \sin \pi z$.
 ▶ 24. [HM21] Udowodnij pozyteczne nierówności

$$\frac{n^n}{e^{n-1}} \leq n! \leq \frac{n^{n+1}}{e^{n-1}}, \quad \text{całkowite } n \geq 1.$$

[Wskazówka: $1 + x \leq e^x$ dla wszystkich rzeczywistych x ; stąd $(k+1)/k \leq e^{1/k} \leq k/(k-1)$].

25. [M20] Czy potęgi ubywające i przyrostające spełniają prawo podobne do zwykłego prawa działań na potęgach $x^{m+n} = x^m x^n$?

1.2.6. Współczynniki dwumianowe

Kombinację k -elementową zbioru n -elementowego otrzymujemy w wyniku wybrania k różnych elementów spośród n , gdy nie interesuje nas kolejność wybranych elementów. Kombinacje pięciu elementów $\{a, b, c, d, e\}$ branych po trzy naraz są następujące:

$$abc, \quad abd, \quad abe, \quad acd, \quad ace, \quad ade, \quad bcd, \quad bce, \quad bde, \quad cde. \quad (1)$$

Policzenie wszystkich k -elementowych kombinacji n elementów jest proste: z powodu rozdziału z równości (2) wiadomo, że jest $n(n-1)\dots(n-k+1)$ sposobów wybrania k pierwszych obiektów permutacji, przy tym każda kombinacja k -elementowa pojawia się dokładnie $k!$ razy (bo tyle jest permutacji k elementów wchodzących w skład kombinacji k -elementowej). Stąd liczba kombinacji, oznaczana symbolem $\binom{n}{k}$, wynosi

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots(1)}. \quad (2)$$

Na przykład,

$$\binom{5}{3} = \frac{5 \cdot 4 \cdot 3}{3 \cdot 2 \cdot 1} = 10,$$

co odpowiada liczbie kombinacji wyznaczonych w (1).

Wielkość $\binom{n}{k}$ (czytaj „ n po k ”) nazywamy *współczynnikiem dwumianowym*. Współczynniki dwumianowe mają niewiarygodnie wiele zastosowań. To zapewne najważniejsze wielkości w analizie algorytmów, gorąco polecamy więc Czytelnikowi pilne przestudiowanie ich własności.

Równanie (2) może posłużyć za definicję $\binom{n}{k}$ nawet wtedy, gdy n nie jest liczbą całkowitą. Ścisłej, definiujemy $\binom{r}{k}$ dla dowolnej liczby rzeczywistej r i dowolnej liczby całkowitej k w następujący sposób:

$$\binom{r}{k} = \frac{r(r-1)\dots(r-k+1)}{k(k-1)\dots(1)} = \frac{r^k}{k!} = \prod_{j=1}^k \frac{r+1-j}{j}, \quad \text{całkowite } k \geq 0; \quad (3)$$

$$\binom{r}{k} = 0, \quad \text{całkowite } k < 0.$$

Tabela 1

TABELA WSPÓŁCZYNNIKÓW DWUMIANOWYCH (TRÓJKĄT PASCALA)

r	$\binom{r}{0}$	$\binom{r}{1}$	$\binom{r}{2}$	$\binom{r}{3}$	$\binom{r}{4}$	$\binom{r}{5}$	$\binom{r}{6}$	$\binom{r}{7}$	$\binom{r}{8}$	$\binom{r}{9}$
0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
2	1	2	1	0	0	0	0	0	0	0
3	1	3	3	1	0	0	0	0	0	0
4	1	4	6	4	1	0	0	0	0	0
5	1	5	10	10	5	1	0	0	0	0
6	1	6	15	20	15	6	1	0	0	0
7	1	7	21	35	35	21	7	1	0	0
8	1	8	28	56	70	56	28	8	1	0
9	1	9	36	84	126	126	84	36	9	1

W szczególności mamy

$$\binom{r}{0} = 1, \quad \binom{r}{1} = r, \quad \binom{r}{2} = \frac{r(r-1)}{2}. \quad (4)$$

W tabeli 1 są zestawione wartości współczynników dwumianowych dla małych całkowitych wartości r i k . Wartości dla $0 \leq r \leq 4$ najlepiej nauczyć się na pamięć.

Współczynniki dwumianowe mają długą i ciekawą historię. Tabela 1 jest nazywana „trójkątem Pascala”, gdyż pojawiła się w pracy Blaise Pascala *Traité du Triangle Arithmétique* z 1653 roku. Ten traktat był znaczącą pracą z wczesnej teorii prawdopodobieństwa, ale nie Pascal odkrył współczynniki dwumianowe (które wówczas były dobrze w Europie znane). Tabela 1 ukazała się także w traktacie *Szu-yüan Yü-chien* („Nefrytowe zwierciadło czterech pierwiastków”) z 1303 roku, autorstwa chińskiego matematyka Shih-Chieh Chu, który pisze o nich jako o odkryciu nie najnowszym. Najstarsze znane omówienie współczynników dwumianowych znajdziemy w pochodzących z X wieku przypisach komentatora imieniem Halāyudha dotyczących wiekowego klasycznego hinduskiego dzieła *Chandah-sūtra* autorstwa Piṅgali. [Zobacz G. Chakravarti, *Bull. Calcutta Math. Soc.* **24** (1932), 79–88]. Około 1150 roku hinduski matematyk Bhāskara Āchārya bardzo elegancko wyłożył współczynniki dwumianowe w swojej książce *Līlāvatī*, punkt 6, rozdział 4. Dla małych wartości k współczynniki wielomianowe znano dużo wcześniej – pojawiają się one w greckich i rzymskich zapiskach wraz z interpretacją geometryczną (zobacz rysunek 8). Oznaczenie $\binom{r}{k}$ zostało wprowadzone przez Andreasa von Ettingshausena w jego książce *Die combinatorische Analysis* (Vienna: 1826).

Czytelnik z pewnością dostrzegł kilka ciekawych własności liczb w tabeli 1. Współczynniki dwumianowe spełniają dosłownie tysiące nietrywialnych tożsamości, a ich niesamowite zależności były stopniowo odkrywane przez wieki. Na dobrą sprawę znaleziono do tej pory tyle związków, że nowe odkrycie nie wywołuje już sensacji. Współczynniki dwumianowe powinien mieć w małym

palcu każdy, kto na poważnie chce zmierzyć się ze wzorami pojawiającymi się w analizie algorytmów. W tym rozdziale pokażemy, jak sobie z nimi radzić. Mark Twain próbował kiedyś zredukować wszystkie dowcipy do kilkunastu podstawowych kategorii (wieśniak w mieście, zięć i teściowa itp.). My spróbujemy z tysięcy тожsamości uzyskać poręczny zbiór prostych operacji, za pomocą których można zmierzyć się niemal z każdym problemem, w którym występują współczynniki dwumianowe.

W większości zastosowań zarówno r , jak i k w $\binom{r}{k}$ będą liczbami całkowitymi. Niektóre opisane techniki dotyczą wyłącznie takich przypadków. Dla zachowania ostrożności po prawej stronie numerowanych równań będziemy wypisywać założenia dotyczące zmiennych. W równaniu (3), na przykład, pojawia się założenie, że k jest liczbą całkowitą, podczas gdy nie zakładamy nic o r . Im mniej założeń, tym łatwiej stosować тожsamość.

W następnych akapitach zajmiemy się podstawowymi technikami wykonywania działań na współczynnikach dwumianowych.

A. Przedstawienie za pomocą silni. Ze wzoru (3) wynika natychmiast

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}, \quad \text{całkowite } n \geq 0, \text{ całkowite } k \geq 0. \quad (5)$$

Wzór pozwala zapisywać kombinacje silni za pomocą współczynników dwumianowych i na odwrót.

B. Symetria. Ze wzorów (3) i (5) mamy

$$\binom{n}{k} = \binom{n}{n-k}, \quad \text{całkowite } n \geq 0, \text{ całkowite } k. \quad (6)$$

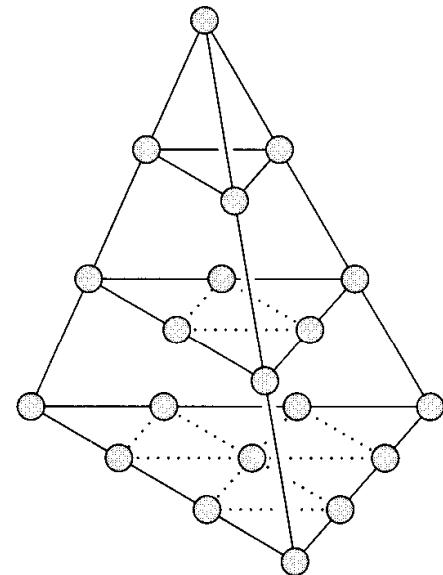
Wzór zachodzi dla wszystkich k całkowitych. Gdy k jest ujemne lub większe niż n , współczynnik dwumianowy równa się zeru (przy założeniu, że n jest nieujemną liczbą całkowitą).

C. Wyłączanie przed nawias. Z definicji (3) mamy

$$\binom{r}{k} = \frac{r}{k} \binom{r-1}{k-1}, \quad \text{całkowite } k \neq 0. \quad (7)$$

Ten wzór jest bardzo przydatny do wiązania współczynników dwumianowych z innymi częściami wyrażenia. W wyniku prostych przekształceń otrzymujemy

$$k \binom{r}{k} = r \binom{r-1}{k-1}, \quad \frac{1}{r} \binom{r}{k} = \frac{1}{k} \binom{r-1}{k-1}.$$



Rys. 8. Interpretacja geometryczna liczby $\binom{n+2}{3}$, $n = 4$.

Pierwszy wzór zachodzi dla całkowitych k , a drugi, gdy nie występuje dzielenie przez zero. Istnieje też podobny związek

$$\binom{r}{k} = \frac{r}{r-k} \binom{r-1}{k}, \quad \text{całkowite } k \neq r. \quad (8)$$

Zilustrujemy powyższe przekształcenia, udowadniając wzór (8). Stosujemy wzory (6), (7) i jeszcze raz (6):

$$\binom{r}{k} = \binom{r}{r-k} = \frac{r}{r-k} \binom{r-1}{r-1-k} = \frac{r}{r-k} \binom{r-1}{k}.$$

[Uwaga: To wyprowadzenie jest poprawne jedynie wtedy, gdy r jest dodatnią liczbą całkowitą $\neq k$, a to z przyczyny założeń w (6) i (7). A jednak sformułowanie (8) nie zawiera tego ograniczenia. Pełną wersję (8) można udowodnić za pomocą prostej, acz niezmiernie istotnej sztuczki. Udowodniliśmy, że

$$r \binom{r-1}{k} = (r-k) \binom{r}{k}$$

dla nieskończonie wielu wartości r . Obie strony równania są wielomianami zmiennej r . Niezerowy wielomian stopnia n może mieć najwyżej n różnych miejsc zerowych. Proste odejmowanie stronami prowadzi do wniosku, że jeżeli dwa wielomiany stopnia $\leq n$ mają równe wartości w $n+1$ różnych punktach, to są równe. Za pomocą powyższej zasady można uogólniać tożsamości na liczby rzeczywiste].

D. Dodawanie. Prosty związek

$$\binom{r}{k} = \binom{r-1}{k} + \binom{r-1}{k-1}, \quad \text{całkowite } k, \quad (9)$$

rzuca się w oczy przy oglądaniu tabeli 1 (każda wartość jest sumą wartości „górnego” i „lewej górnej”), a daje się łatwo uzasadnić za pomocą (3). Można też inaczej, ze wzorów (7) i (8):

$$r \binom{r-1}{k} + r \binom{r-1}{k-1} = (r-k) \binom{r}{k} + k \binom{r}{k} = r \binom{r}{k}.$$

Równość (9) przydaje się w dowodach przez indukcję względem r (gdy r jest całkowite).

E. Sumowanie. Stosując wielokrotnie równanie (9), otrzymujemy

$$\binom{r}{k} = \binom{r-1}{k} + \binom{r-1}{k-1} = \binom{r-1}{k} + \binom{r-2}{k-1} + \binom{r-2}{k-2} = \dots;$$

lub

$$\binom{r}{k} = \binom{r-1}{k-1} + \binom{r-1}{k} = \binom{r-1}{k-1} + \binom{r-2}{k-1} + \binom{r-2}{k} = \dots.$$

To prowadzi do dwóch istotnych wzorów na sumowanie współczynników dwumianowych:

$$\sum_{k=0}^n \binom{r+k}{k} = \binom{r}{0} + \binom{r+1}{1} + \cdots + \binom{r+n}{n} = \binom{r+n+1}{n},$$

całkowite $n \geq 0$. (10)

$$\sum_{k=0}^n \binom{k}{m} = \binom{0}{m} + \binom{1}{m} + \cdots + \binom{n}{m} = \binom{n+1}{m+1},$$

całkowite $m \geq 0$, całkowite $n \geq 0$. (11)

Równość (11) można udowodnić przez indukcję względem n , ale dużo ciekawsze jest wyprowadzenie jej z (10) za pomocą dwukrotnego zastosowania wzoru (6):

$$\begin{aligned} \sum_{0 \leq k \leq n} \binom{k}{m} &= \sum_{0 \leq m+k \leq n} \binom{m+k}{m} = \sum_{-m \leq k < 0} \binom{m+k}{m} + \sum_{0 \leq k \leq n-m} \binom{m+k}{k} \\ &= 0 + \binom{m+(n-m)+1}{n-m} = \binom{n+1}{m+1}, \end{aligned}$$

przy założeniu, że $n \geq m$. Jeśli $n < m$, to (11) jest oczywiste.

Równanie (11) jest wykorzystywane bardzo często. W zasadzie w poprzednich punktach rozpatrywaliśmy jego szczególne przypadki. Na przykład przy $m = 1$ okazuje się być jakże znajomą sumą ciągu arytmetycznego:

$$\binom{0}{1} + \binom{1}{1} + \cdots + \binom{n}{1} = 0 + 1 + \cdots + n = \binom{n+1}{2} = \frac{(n+1)n}{2}.$$

Przypuśćmy, że szukamy prostego wzoru na sumę $1^2 + 2^2 + \cdots + n^2$. Znajdziemy go, zauważając, że $k^2 = 2\binom{k}{2} + \binom{k}{1}$. Stąd

$$\sum_{k=0}^n k^2 = \sum_{k=0}^n \left(2\binom{k}{2} + \binom{k}{1} \right) = 2\binom{n+1}{3} + \binom{n+1}{2}.$$

Powyższa odpowiedź, wyrażona w języku współczynników dwumianowych, może zostać z powrotem przetłumaczona na wielomiany:

$$1^2 + 2^2 + \cdots + n^2 = 2 \frac{(n+1)n(n-1)}{6} + \frac{(n+1)n}{2} = \frac{1}{3}n(n+\frac{1}{2})(n+1). \quad (12)$$

Wzór na sumę $1^3 + 2^3 + \cdots + n^3$ możemy wyznaczyć podobnie. *Dowolny* wielomian $a_0 + a_1 k + a_2 k^2 + \cdots + a_m k^m$ można zapisać jako $b_0 \binom{k}{0} + b_1 \binom{k}{1} + \cdots + b_m \binom{k}{m}$ dla odpowiednich współczynników b_0, \dots, b_m . Wróćmy do tego później.

F. Twierdzenie dwumianowe. Tak, twierdzenie dwumianowe (wzór Newtona) to jedno z naszych podstawowych narzędzi:

$$(x+y)^r = \sum_k \binom{r}{k} x^k y^{r-k}, \quad \text{całkowite } r \geq 0. \quad (13)$$

Na przykład $(x+y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$. (Wreszcie wiadomo, skąd tadziwna nazwa „współczynniki dwumianowe”).

Nie należy przeoczyć faktu, że we wzorze (13) stoi \sum_k , a nie $\sum_{k=0}^r$, jak by się można spodziewać. Gdy nie ustanawiamy ograniczeń na k , sumowanie odbywa się po *wszystkich* wartościach całkowitych, $-\infty < k < +\infty$. Ale w tym wypadku oba zapisy są równoważne, bo wyrazy w (13) są zerami, gdy $k < 0$ lub $k > r$. Lepiej posługiwać się prostszą postacią \sum_k , gdyż im mniej dodatkowych warunków, tym łatwiej manipulować sumami. Ciągle przepisywanie granic sumowania w skomplikowanych przekształceniach to masa zbędnej roboty, której możemy sobie zaoszczędzić, pomijając granice tam, gdzie to możliwe. Taki zapis ma jeszcze jedną zaletę: jeżeli r nie jest nieujemną liczbą całkowitą, to (13) staje się sumą *nieskończoną*, a znana z analizy matematycznej wersja *twierdzenia dwumianowego* orzeka, że (13) zachodzi dla wszystkich r , jeżeli $|x/y| < 1$.

Zauważmy, że wzór (13) daje

$$0^0 = 1. \quad (14)$$

Tej konwencji będziemy się trzymać.

Przypadek szczególny $y = 1$ wzoru (13) jest tak istotny, że wyrazimy go osobno:

$$\sum_k \binom{r}{k} x^k = (1+x)^r, \quad \text{całkowite } r \geq 0 \text{ lub } |x| < 1. \quad (15)$$

O odkryciu twierdzenia dwumianowego Izaak Newton donosił z Oldenburga w listach z 13 czerwca i 24 października 1676 roku. [Zobacz D. Struik, *Source Book in Mathematics* (Harvard Univ. Press, 1969), 284–291]. Ale Newton nie znał poprawnego dowodu. W tamtych czasach nie zdawano sobie do końca sprawy z potrzeby ścisłego dowodu. Pierwszą próbę udowodnienia wzoru podjął L. Euler (w 1774 roku), jednakże nie dowódł on twierdzenia w pełnej ogólności. Dowód z prawdziwego zdarzenia podał wreszcie C. F. Gauss w 1812 roku. Na dobrą sprawę praca Gaussa była pierwszym przypadkiem, kiedy *cokolwiek* na temat sum nieskończonych udowodniono przekonującą.

Na początku XIX wieku N. H. Abel odkrył zaskakujące uogólnienie wzoru dwumianowego (13):

$$(x+y)^n = \sum_k \binom{n}{k} x(x-kz)^{k-1}(y+kz)^{n-k}, \quad \text{całkowite } n \geq 0, \quad x \neq 0. \quad (16)$$

Wzór jest tożsamością o *trzech* zmiennych, x , y i z (zobacz ćwiczenia 50–52). Abel opublikował i udowodnił ten wzór w numerze 1 mającego wkrótce zdobyć popularność czasopisma A. L. Crelle'a *Journal für die reine und angewandte Mathematik* (1826), strony 159–160. Ciekawe, że Abel wniósł do tego pierwszego numeru wiele innych artykułów, m.in. słynny raport o nierozwiązywalności równań algebraicznych stopnia piątego i wyższych przez pierwiastki i o twierdzeniu dwumianowym. Bibliografia dotycząca (16) znajduje się w H. W. Gould, *AMM* **69** (1962), 572.

G. Minus w górnym indeksie. Podstawowa tożsamość

$$\binom{r}{k} = (-1)^k \binom{k-r-1}{k}, \quad \text{całkowite } k, \quad (17)$$

wynika natychmiast z definicji (3), gdy pozamieniamy znaki czynników licznika. Ten wzór jest także często stosowany do przekształcania górnego indeksu.

Prostym wnioskiem z (17) jest wzór:

$$\sum_{k \leq n} \binom{r}{k} (-1)^k = \binom{r}{0} - \binom{r}{1} + \cdots + (-1)^n \binom{r}{n} = (-1)^n \binom{r-1}{n}, \quad \text{całkowite } n. \quad (18)$$

Tę tożsamość można udowodnić przez indukcję za pomocą (9), ale można także użyć wprost wzorów (17) i (10):

$$\sum_{k \leq n} \binom{r}{k} (-1)^k = \sum_{k \leq n} \binom{k-r-1}{k} = \binom{-r+n}{n} = (-1)^n \binom{r-1}{n}.$$

Gdy r jest całkowite, z (17) otrzymujemy kolejną ważną tożsamość:

$$\binom{n}{m} = (-1)^{n-m} \binom{-(m+1)}{n-m}, \quad \text{całkowite } n \geq 0, \text{ całkowite } m. \quad (19)$$

(Podstawiamy $r = n$ i $k = n-m$ w (17), po czym stosujemy (6)). Przemieściliśmy n z góry na dół!

H. Upraszczanie iloczynów. Iloczyny współczynników dwumianowych można zazwyczaj przedstawać na wiele sposobów przez rozwijanie i zwijanie współczynników w kombinacje silni za pomocą równania (5). Na przykład

$$\binom{r}{m} \binom{m}{k} = \binom{r}{k} \binom{r-k}{m-k}, \quad \text{całkowite } m, \quad \text{całkowite } k. \quad (20)$$

Wystarczy udowodnić równanie (20) dla całkowitych $r \geq m$ (zobacz uwagi poczynione przy dowodzie (8)) oraz $0 \leq k \leq m$. Wówczas

$$\binom{r}{m} \binom{m}{k} = \frac{r! m!}{m! (r-m)! k! (m-k)!} = \frac{r! (r-k)!}{k! (r-k)! (m-k)! (r-m)!} = \binom{r}{k} \binom{r-k}{m-k}.$$

Równość (20) przydaje się w sytuacjach, gdy zmienna (konkretnie m) występuje zarówno w górnym, jak i w dolnym indeksie, a my chcielibyśmy, żeby pojawiała się tylko w jednym miejscu. Zauważmy, że równanie (7) jest szczególnym przypadkiem (20), tj. dla $k = 1$.

I. Sumy iloczynów. Aby nasz zbiór operacji na współczynnikach dwumianowych był kompletny, przedstawiamy bardzo ogólne tożsamości, które udowodnimy w ramach ćwiczeń na końcu tego rozdziału. Poniższe wzory pokazują, w jaki sposób upraszczać sumy iloczynów współczynników dwumianowych w zależności od tego, gdzie występuje zmienna indeksowa k :

$$\sum_k \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n}, \quad \text{całkowite } n. \quad (21)$$

$$\sum_k \binom{r}{m+k} \binom{s}{n+k} = \binom{r+s}{r-m+n},$$

całkowite m , całkowite n , całkowite $r \geq 0$. (22)

$$\sum_k \binom{r}{k} \binom{s+k}{n} (-1)^{r-k} = \binom{s}{n-r}, \quad \text{całkowite } n, \text{ całkowite } r \geq 0. \quad (23)$$

$$\sum_{k=0}^r \binom{r-k}{m} \binom{s}{k-t} (-1)^{k-t} = \binom{r-t-s}{r-t-m},$$

całkowite $t \geq 0$, całkowite $r \geq 0$, całkowite $m \geq 0$. (24)

$$\sum_{k=0}^r \binom{r-k}{m} \binom{s+k}{n} = \binom{r+s+1}{m+n+1},$$

całkowite $n \geq$ całkowite $s \geq 0$, całkowite $m \geq 0$, całkowite $r \geq 0$. (25)

$$\sum_{k \geq 0} \binom{r-tk}{k} \binom{s-t(n-k)}{n-k} \frac{r}{r-tk} = \binom{r+s-tn}{n}, \quad \text{całkowite } n. \quad (26)$$

Z tych wszystkich wzorów równość (21) jest najważniejsza – należy zachować ją w pamięci. Prostym sposobem zapamiętania jest interpretowanie prawej strony jako liczby sposobów wybrania n osób spośród r mężczyzn i s kobiet. Każdy wyraz po lewej stronie to liczba sposobów wybrania k mężczyzn i $n - k$ kobiet. Równanie (21) jest popularnie nazywane splotem Vandermonde'a, gdyż A. Vandermonde opublikował je w *Mém. Acad. Roy. Sciences Paris* (1772), 489–498. Niemniej jednak wzór pojawił się już we wspomnianym wcześniej traktacie Shih-Chieh Chu z 1303 roku [zobacz J. Needham, *Science and Civilization in China 3* (Cambridge University Press, 1959), 138–139].

Jeśli w równaniu (26) $r = tk$, to udaje się uniknąć wystąpienia zera w mianowniku przez skrócenie z czynnikiem w liczniku. Stąd (26) jest tożsamością wielomianową zmiennych r , s i t . Oczywiście (21) jest szczególnym przypadkiem (26) dla $t = 0$.

Zastanówmy się jeszcze nad następującym nietrywialnym zastosowaniem równań (23) i (25): często wygodnie jest zastąpić prosty współczynnik dwumianowy po prawej stronie przez bardziej złożony po lewej, zmienić kolejność sumowania i uprościć. Możemy traktować lewe strony jak rozwinięcia

$$\binom{s}{n+a} \quad \text{wyrażone za pomocą} \quad \binom{s+k}{n}.$$

Ze wzoru (23) korzystamy dla a ujemnych, natomiast ze wzoru (25) dla a dodatnich.

Powyższe stwierdzenie zamyka nasze studium zagadnień dotyczących współczynników dwumianowych. Czytelnik powinien nauczyć się przede wszystkim wzorów (5), (6), (7), (9), (13), (17), (20), i (21) – czemuż by nie zakreślić ich fluoresencyjnym flamastrem?

Mając do dyspozycji te wszystkie metody, powinniśmy być w stanie rozwiązać prawie każdy napotkany problem i to przynajmniej na trzy sposoby. Poniższe przykłady ilustrują przedstawione metody.

Przykład 1. Jaka jest wartość $\sum_k \binom{r}{k} \binom{s}{k} k$, gdy r jest dodatnią liczbą całkowitą?

Rozwiązanie. Za pomocą (7) pozbywamy się zewnętrznego k :

$$\sum_k \binom{r}{k} \binom{s}{k} k = \sum_k \binom{r}{k} \binom{s-1}{k-1} s = s \sum_k \binom{r}{k} \binom{s-1}{k-1}.$$

Teraz stosujemy wzór (22) dla $n = -1$. Odpowiedzią jest

$$\sum_k \binom{r}{k} \binom{s}{k} k = \binom{r+s-1}{r-1} s, \quad \text{całkowite } r \geq 0.$$

Przykład 2. Jaka jest wartość $\sum_k \binom{n+k}{2k} \binom{2k}{k} \frac{(-1)^k}{k+1}$, gdy n jest nieujemną liczbą całkowitą?

Rozwiązanie. Ten problem jest trudniejszy. Zmienna indeksowa k występuje w sześciu miejscach! Zaczynamy od zastosowania wzoru (20), w wyniku czego otrzymamy

$$\sum_k \binom{n+k}{k} \binom{n}{k} \frac{(-1)^k}{k+1}.$$

Możemy nieco odetchnąć, bo kilka paskudnych cech wyjściowego wzoru szczęśliwie zniknęło. Kolejny krok powinien być oczywisty: stosujemy wzór (7) podobnie jak w przykładzie 1

$$\sum_k \binom{n+k}{k} \binom{n+1}{k+1} \frac{(-1)^k}{n+1}. \quad (27)$$

Nie jest źle, pozbyliśmy się kolejnego k . W tym miejscu mamy do wyboru dwie równie obiecujące taktyki. Możemy zastąpić $\binom{n+k}{k}$ przez $\binom{n+k}{n}$, zakładając, że $k \geq 0$, i obliczyć sumę za pomocą (23):

$$\begin{aligned} & \sum_{k \geq 0} \binom{n+k}{n} \binom{n+1}{k+1} \frac{(-1)^k}{n+1} \\ &= -\frac{1}{n+1} \sum_{k \geq 1} \binom{n-1+k}{n} \binom{n+1}{k} (-1)^k \\ &= -\frac{1}{n+1} \sum_{k \geq 0} \binom{n-1+k}{n} \binom{n+1}{k} (-1)^k + \frac{1}{n+1} \binom{n-1}{n} \\ &= -\frac{1}{n+1} (-1)^{n+1} \binom{n-1}{-1} + \frac{1}{n+1} \binom{n-1}{n} = \frac{1}{n+1} \binom{n-1}{n}. \end{aligned}$$

Wartość współczynnika dwumianowego $\binom{n-1}{n}$ wynosi zero poza przypadkiem $n = 1$, gdy wynosi jeden. Eleganckim sposobem wyrażenia odpowiedzi jest

skorzystanie z konwencji Iversona [$n = 0$] (równanie 1.2.3–(16)) lub delty Kroneckera δ_{n0} (równanie 1.2.3–(19)).

Inny sposób przekształcania wzoru (27) polega na skorzystaniu z (17), w wyniku czego otrzymamy:

$$\sum_k \binom{-(n+1)}{k} \binom{n+1}{k+1} \frac{1}{n+1}.$$

Możemy teraz zastosować (22) i dostać

$$\binom{n+1-(n+1)}{n+1-1+0} \frac{1}{n+1} = \binom{0}{n} \frac{1}{n+1}.$$

I znów otrzymaliśmy odpowiedź

$$\sum_k \binom{n+k}{2k} \binom{2k}{k} \frac{(-1)^k}{k+1} = \delta_{n0}, \quad \text{całkowite } n \geq 0. \quad (28)$$

Przykład 3. Jaka jest wartość $\sum_k \binom{n+k}{m+2k} \binom{2k}{k} \frac{(-1)^k}{k+1}$ dla dodatnich liczb całkowitych m i n ?

Rozwiązanie. Jeśli m byłoby równe zero, otrzymalibyśmy wzór jak w przykładzie 2. Niestety, obecność m oznacza, że próby zastosowania schematu z poprzedniego przykładu spałą na panewce, gdyż pierwszy krok polegał na wykorzystaniu wzoru (20), a teraz tego nie da się zrobić. W takim przypadku opłaca się nieco skomplikować zadanie przez wymianę niepożądanego $\binom{n+k}{m+2k}$ na sumę wyrazów postaci $\binom{x+k}{2k}$, gdyż w ten sposób nasz problem okaże się sumą problemów, z którymi już umiemy sobie poradzić. Korzystamy ze wzoru (25), przedstawiając

$$r = n + k - 1, \quad m = 2k, \quad s = 0, \quad n = m - 1$$

i otrzymujemy

$$\sum_k \sum_{0 \leq j \leq n+k-1} \binom{n+k-1-j}{2k} \binom{2k}{k} \binom{j}{m-1} \frac{(-1)^k}{k+1}. \quad (29)$$

Chcielibyśmy najpierw sumować po k , ale zamiana kolejności sumowania wymaga sumowania po wartościach k , które są ≥ 0 i $\geq j - n + 1$. Niestety, ten drugi warunek powoduje trudności, bo *nie* znamy wartości szukanej sumy dla $j \geq n$. Kłopotom można zaradzić, zauważając, że wyrazy wzoru (29) są równe zero, gdy $n \leq j \leq n+k-1$. Oznacza to, że $k \geq 1$, zatem $0 \leq n+k-1-j \leq k-1 < 2k$ i pierwszy współczynnik dwumianowy we wzorze (29) znika. Warunek w drugiej sumie możemy teraz zastąpić przez $0 \leq j < n$ i zamiana kolejności sumowania staje się prosta. Sumując po k , na mocy (28) otrzymujemy

$$\sum_{0 \leq j < n} \binom{j}{m-1} \delta_{(n-1-j)0},$$

więc wszystkie wyrazy są zerami poza przypadkiem $j = n - 1$, stąd ostateczną odpowiedzią jest

$$\binom{n-1}{m-1}.$$

Rozwiązywanie powyższego problemu było dość zawikłane, jednakże nie zawierało w sobie nic tajemniczego – dla każdego kroku istniały rozsądne przesłanki. Wyprowadzenie warte jest szczegółowego przestudiowania, ponieważ ilustruje subtelności manipulowania warunkami w równaniach. Po prawdzie istnieje lepsza metoda uporania się z przedstawionym problemem – Czytelnikowi pozostawiamy znalezienie takiego przekształcenia sumy, by można było zastosować wzór (26) (zobacz ćwiczenie 30).

Przykład 4. Udowodnij, że

$$\sum_k A_k(r, t) A_{n-k}(s, t) = A_n(r + s, t), \quad \text{całkowite } n \geq 0, \quad (30)$$

gdzie $A_n(x, t)$ jest wielomianem stopnia n zmiennej x spełniającym warunek

$$A_n(x, t) = \binom{x - nt}{n} \frac{x}{x - nt}, \quad \text{dla } x \neq nt.$$

Rozwiązanie. Możemy założyć, że $r \neq kt \neq s$ dla $0 \leq k \leq n$, gdyż obie strony równości (30) są wielomianami zmiennych r, s, t . Problem sprowadza się do obliczenia

$$\sum_k \binom{r - kt}{k} \binom{s - (n - k)t}{n - k} \frac{r}{r - kt} \frac{s}{s - (n - k)t},$$

co wygląda sto razy gorzej niż problemy straszące dotychczas. Zauważmy jednak silne podobieństwo do wzoru (26), zastanówmy się też nad przypadkiem $t = 0$.

Chciałoby się zastąpić

$$\binom{r - kt}{k} \frac{r}{r - kt} \quad \text{przez} \quad \binom{r - kt - 1}{k - 1} \frac{r}{k},$$

tylko że wtedy tracimy podobieństwo do (26) i zamiana jest niemożliwa dla $k = 0$. Lepszą metodą jest wykorzystanie techniki tzw. *ułamków prostych* polegającej na zamianie ułamka o skomplikowanym mianowniku na sumę ułamków o mianownikach prostszych. Mamy na przykład

$$\frac{1}{r - kt} \frac{1}{s - (n - k)t} = \frac{1}{r + s - nt} \left(\frac{1}{r - kt} + \frac{1}{s - (n - k)t} \right).$$

W wyniku zastosowania tej równości do przekształcenia sumy otrzymujemy

$$\begin{aligned} & \frac{s}{r + s - nt} \sum_k \binom{r - kt}{k} \binom{s - (n - k)t}{n - k} \frac{r}{r - kt} \\ & + \frac{r}{r + s - nt} \sum_k \binom{r - kt}{k} \binom{s - (n - k)t}{n - k} \frac{s}{s - (n - k)t}, \end{aligned}$$

a takie sumy możemy potraktować równością (26), jeżeli w drugiej z nich zastąpimy $n - k$ przez k . Tożsamości (26) i (30) pochodzą z: H. A. Rothe, *Formulæ de Serierum Reversione* (Leipzig: 1793); przypadki szczególne tych wzorów są do dziś dnia często „odkrywane”. Zajmująca historia tych tożsamości oraz pewne

ich uogólnienia są opisane w: H. W. Gould, J. Kaucký, *Journal of Combinatorial Theory* 1 (1966), 233–248.

Przykład 5. Wyznacz wartości a_0, a_1, a_2, \dots , takie że

$$n! = a_0 + a_1 n + a_2 n(n-1) + a_3 n(n-1)(n-2) + \dots \quad (31)$$

dla wszystkich nieujemnych liczb całkowitych n .

Rozwiążanie. Odpowiedź wynika z równości 1.2.5–(11) podanej bez dowodu w poprzednim rozdziale. Będziemy udawać, że jej póki co nie znamy. Problem na pewno ma rozwiązanie, bo podstawiając $n = 0$, możemy wyznaczyć a_0 , następnie podstawiając $n = 1$, wyznaczyć a_1 itd.

Po pierwsze chcielibyśmy zapisać (31) za pomocą współczynników dwumianowych:

$$n! = \sum_k \binom{n}{k} k! a_k. \quad (32)$$

Zagadnienie rozwiązywania równań uwikłanych podobnej postaci nosi nazwę *problemu inwersji*, a metoda, z której skorzystamy, ma zastosowanie również do innych problemów tego typu.

Pomysł polega na rozpatrzeniu szczególnego przypadku $s = 0$ równości (23):

$$\sum_k \binom{r}{k} \binom{k}{n} (-1)^{r-k} = \binom{0}{n-r} = \delta_{nr}, \quad \text{całkowite } n, \text{ całkowite } r \geq 0. \quad (33)$$

Istotne jest w tym wzorze to, że jeśli $n \neq r$, to suma wynosi zero. Dzięki temu można rozwiązać problem, bo wiele wyrazów się wyzeruje, jak w przykładzie 3:

$$\begin{aligned} \sum_n n! \binom{m}{n} (-1)^{m-n} &= \sum_n \sum_k \binom{n}{k} k! a_k \binom{m}{n} (-1)^{m-n} \\ &= \sum_k k! a_k \sum_n \binom{n}{k} \binom{m}{n} (-1)^{m-n} \\ &= \sum_k k! a_k \delta_{km} = m! a_m. \end{aligned}$$

Czytelnik zechce zwrócić uwagę, w jaki sposób w wyniku dodawania odpowiednich wielokrotności (32) dla $n = 0, 1, \dots$ uzyskaliśmy równanie z tylko jednym a_m . Mamy teraz

$$a_m = \sum_{n \geq 0} (-1)^{m-n} \frac{n!}{m!} \binom{m}{n} = \sum_{0 \leq n \leq m} \frac{(-1)^{m-n}}{(m-n)!} = \sum_{0 \leq n \leq m} \frac{(-1)^n}{n!}.$$

To zamyka rozwiązanie przykładu 5. Przyjrzyjmy się konsekwencjom wzoru (33). Gdy r i m są nieujemne i całkowite, mamy

$$\sum_k \binom{r}{k} (-1)^{r-k} \left(c_0 \binom{k}{0} + c_1 \binom{k}{1} + \dots + c_m \binom{k}{m} \right) = c_r,$$

bo inne wyrazy znikają po zsumowaniu. Odpowiednio dobierając współczynniki c_i , możemy przedstawić *każdy* wielomian k zmiennych jako sumę współczynników dwumianowych o górnym indeksie k . Mamy stąd

$$\sum_k \binom{r}{k} (-1)^{r-k} (b_0 + b_1 k + \cdots + b_r k^r) = r! b_r, \quad \text{całkowite } r \geq 0, \quad (34)$$

gdzie $b_0 + \cdots + b_r k^r$ oznacza dowolny wielomian stopnia co najwyżej r . (Ten wzór nie powinien być wielkim zaskoczeniem dla znawców analizy numerycznej, bo $\sum_k \binom{r}{k} (-1)^{r-k} f(x+k)$ to „ r -ta różnica” funkcji $f(x)$).

Za pomocą wzoru (34) możemy natychmiast otrzymać wiele związków, które na pierwszy rzut oka wydają się bardzo skomplikowane i których zazwyczaj bardzo zawile się dowodzi, na przykład

$$\sum_k \binom{r}{k} \binom{s-kt}{r} (-1)^k = t^r, \quad \text{całkowite } r \geq 0. \quad (35)$$

Podręczniki mają to do siebie, że zawierają masę robiących dobre wrażenie przykładów, eleganckich sztuczek itp., a przemilczają niewinnie wyglądające problemy, do których nie da się zastosować podanych metod. Z powyższych przykładów można by wyciągnąć wniosek, że ze współczynnikami dwumianowymi daje się robić wszystko. Należy jednak wspomnieć, że jakkolwiek obiecująco wyglądają równania (10), (11) i (18), to raczej nie ma równie prostego wzoru na podobną sumę jak:

$$\sum_{k=0}^n \binom{m}{k} = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{n}, \quad (36)$$

dla $n < m$. (Dla $n = m$ odpowiedź jest prosta. Jaka? Zobacz ćwiczenie 36).

Jednak ta suma ma postać zamkniętą jako funkcja n , gdy m jest ustaloną liczbą całkowitą, na przykład:

$$\sum_{k=0}^n \binom{-2}{k} = (-1)^n \left\lceil \frac{n+1}{2} \right\rceil. \quad (37)$$

Istnieje także prosty wzór

$$\sum_{k=0}^n \binom{m}{k} \left(k - \frac{m}{2} \right) = -\frac{m}{2} \binom{m-1}{n} \quad (38)$$

na sumę, która mogłaby wydawać się trudniejszą.

W jaki sposób stwierdzić, że nie warto dłużej pracować nad sumą, która nie daje się uprościć? Na szczęście dysponujemy już metodą rozstrzygania tego problemu w wielu istotnych przypadkach. Algorytm autorstwa R. W. Gospera i D. Zeilbergera znajduje postaci zamknięte wyrażone za pomocą współczynników dwumianowych lub dowodzi, że takowe nie istnieją. Opis algorytmu Gospera-Zeilbergera wykracza poza ramy tej książki; szczegóły wyjaśniono w CMath podrozdz. 5.8. Polecamy także książkę $A=B$ napisaną przez Petkovšekiego, Wilfa i Zeilbergera (Wellesley, Mass.: A. K. Peters, 1996).

Najważniejszym narzędziem umożliwiającym systematyczne, mechaniczne podejście do rozwiązywania sum zawierających współczynniki dwumianowe są *funkcje hipergeometryczne*, będące szeregami nieskończonymi zdefiniowanymi za pomocą potęg przyrastających:

$$F\left(\begin{array}{c} a_1, \dots, a_m \\ b_1, \dots, b_n \end{array} \middle| z\right) = \sum_{k \geq 0} \frac{a_1^{\bar{k}} \dots a_m^{\bar{k}}}{b_1^{\bar{k}} \dots b_n^{\bar{k}}} \frac{z^k}{k!}. \quad (39)$$

Wprowadzenie do teorii tych istotnych funkcji znajduje się w podrozdziałach 5.5 i 5.6 książki *CMath*, a odniesienia historyczne zamieszczono w: J. Dutka, *Archive for History of Exact Sciences* **31** (1984), 15–34.

Współczynniki dwumianowe mają kilka ważnych uogólnień, które krótko omówimy. Po pierwsze, możemy zezwolić na dowolne wartości rzeczywiste w dolnym indeksie k w $\binom{r}{k}$ (zobacz ćwiczenia 40–45). Mamy również uogólnienie

$$\binom{r}{k}_q = \frac{(1-q^r)(1-q^{r-1})\dots(1-q^{r-k+1})}{(1-q^k)(1-q^{k-1})\dots(1-q^1)}, \quad (40)$$

które staje się zwykłym współczynnikiem dwumianowym $\binom{r}{k}_1 = \binom{r}{k}$, gdy q zbliża się do wartości granicznej 1. Łatwo to sprawdzić, dzieląc każdy wyraz w liczniku i mianowniku przez $1-q$. Te proste własności „współczynników q -mianowych” rozpatrujemy w ćwiczeniu 58.

Z naszego punktu widzenia najważniejszym uogólnieniem jest *współczynnik wielomianowy*

$$\binom{k_1 + k_2 + \dots + k_m}{k_1, k_2, \dots, k_m} = \frac{(k_1 + k_2 + \dots + k_m)!}{k_1! k_2! \dots k_m!}, \quad \text{całkowite } k_i \geq 0. \quad (41)$$

Najważniejszą własnością współczynnika wielomianowego jest uogólnienie równania (13):

$$(x_1 + x_2 + \dots + x_m)^n = \sum_{k_1 + k_2 + \dots + k_m = n} \binom{n}{k_1, k_2, \dots, k_m} x_1^{k_1} x_2^{k_2} \dots x_m^{k_m}. \quad (42)$$

Należy zauważyc, że każdy współczynnik wielomianowy można wyrazić za pomocą współczynników dwumianowych:

$$\binom{k_1 + k_2 + \dots + k_m}{k_1, k_2, \dots, k_m} = \binom{k_1 + k_2}{k_1} \binom{k_1 + k_2 + k_3}{k_1 + k_2} \dots \binom{k_1 + k_2 + \dots + k_m}{k_1 + \dots + k_{m-1}}, \quad (43)$$

więc do przekształcania współczynników dwumianowych możemy stosować poznane wcześniej metody. Obie strony (20) są współczynnikami trójmianowymi

$$\binom{r}{k, m-k, r-m}.$$

Zakończymy ten rozdział krótką analizą przekształcania wielomianu wyrażonego za pomocą potęg x w wielomian wyrażony za pomocą współczynników dwumianowych. Współczynniki związane z tym przekształceniem nazywane są *liczbami Stirlinga* i pojawiają się w analizie wielu algorytmów.

Istnieją dwie odmiany liczb Stirlinga. Liczby Stirlinga pierwszego rodzaju oznaczamy $[n]_k$, a liczby Stirlinga drugiego rodzaju $\{n\}_k$. Oznaczenia pochodzą od Jovana Karamaty [Mathematica (Cluj) **9** (1935), 164–178] i mają przewagę nad innymi zaproponowanymi notacjami [zobacz D. E. Knuth, AMM **99** (1992), 403–422]. Najłatwiej zapamiętać, że nawiasy w $\{n\}_k$ pochodzą od nawiasów klamrowych oznaczających zbiór, bo $\{n\}_k$ to liczba sposobów podziału n -elementowego zbioru na k rozłącznych podzbiorów (ćwiczenie 64). Liczby Stirlinga $[n]_k$ też mają interpretację kombinatoryczną, którą zajmiemy się w punkcie 1.3.3: $[n]_k$ to liczba n -elementowych permutacji mających k cykli.

W tabeli 2 są pokazane trójkąty Stirlinga, które w pewnym sensie przypominają trójkąt Pascala.

Tabela 2
LICZBY STIRLINGA OBU RODZAJÓW

n	$[n]_0$	$[n]_1$	$[n]_2$	$[n]_3$	$[n]_4$	$[n]_5$	$[n]_6$	$[n]_7$	$[n]_8$
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0
3	0	2	3	1	0	0	0	0	0
4	0	6	11	6	1	0	0	0	0
5	0	24	50	35	10	1	0	0	0
6	0	120	274	225	85	15	1	0	0
7	0	720	1764	1624	735	175	21	1	0
8	0	5040	13068	13132	6769	1960	322	28	1
n	$\{n\}_0$	$\{n\}_1$	$\{n\}_2$	$\{n\}_3$	$\{n\}_4$	$\{n\}_5$	$\{n\}_6$	$\{n\}_7$	$\{n\}_8$
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0
3	0	1	3	1	0	0	0	0	0
4	0	1	7	6	1	0	0	0	0
5	0	1	15	25	10	1	0	0	0
6	0	1	31	90	65	15	1	0	0
7	0	1	63	301	350	140	21	1	0
8	0	1	127	966	1701	1050	266	28	1

Przybliżenia dla dużych n są opisane w: L. Moser, M. Wyman, *J. London Math. Soc.* **33** (1958), 133–146; *Duke Math. J.* **25** (1958), 29–43; D. E. Barton, F. N. David, M. Merrington, *Biometrika* **47** (1960), 439–445; **50** (1963), 169–176; N. M. Temme, *Studies in Applied Math.* **89** (1993), 233–243; H. S. Wilf, *J. Combinatorial Theory A* **64** (1993), 344–349; H.-K. Hwang, *J. Combinatorial Theory A* **71** (1995), 343–351.

Liczby Stirlinga pierwszego rodzaju wykorzystuje się do zamiany potęg ubywających na potęgi zupełnie zwyczajne:

$$\begin{aligned}
x^n &= x(x-1)\dots(x-n+1) \\
&= \begin{bmatrix} n \\ n \end{bmatrix} x^n - \begin{bmatrix} n \\ n-1 \end{bmatrix} x^{n-1} + \dots + (-1)^n \begin{bmatrix} n \\ 0 \end{bmatrix} \\
&= \sum_k (-1)^{n-k} \begin{bmatrix} n \\ k \end{bmatrix} x^k.
\end{aligned} \tag{44}$$

Korzystając z tabeli 2, można na przykład zapisać:

$$\binom{x}{5} = \frac{x^5}{5!} = \frac{1}{120}(x^5 - 10x^4 + 35x^3 - 50x^2 + 24x).$$

Liczby Stirlinga drugiego rodzaju wykorzystuje się do zamiany zwykłych potęg na potęgi ubywające:

$$x^n = \left\{ \begin{array}{l} n \\ n \end{array} \right\} x^n + \dots + \left\{ \begin{array}{l} n \\ 1 \end{array} \right\} x^1 + \left\{ \begin{array}{l} n \\ 0 \end{array} \right\} x^0 = \sum_k \left\{ \begin{array}{l} n \\ k \end{array} \right\} x^k. \tag{45}$$

W istocie ten wzór był pierwotnym powodem, dla którego Stirling zajął się liczbami $\left\{ \begin{array}{l} n \\ k \end{array} \right\}$ w swojej pracy *Methodus Differentialis* (London: 1730). Z tabeli 2 mamy na przykład:

$$\begin{aligned}
x^5 &= x^5 + 10x^4 + 25x^3 + 15x^2 + x^1 \\
&= 120 \binom{x}{5} + 240 \binom{x}{4} + 150 \binom{x}{3} + 30 \binom{x}{2} + \binom{x}{1}.
\end{aligned}$$

Zaprezentujemy teraz najważniejsze tożsamości dotyczące liczb Stirlinga. W poniższych równościach zmienne m i n oznaczają nieujemne liczby całkowite. Dodawanie:

$$\begin{aligned}
\begin{bmatrix} n+1 \\ m \end{bmatrix} &= n \begin{bmatrix} n \\ m \end{bmatrix} + \begin{bmatrix} n \\ m-1 \end{bmatrix}; \\
\left\{ \begin{array}{l} n+1 \\ m \end{array} \right\} &= m \left\{ \begin{array}{l} n \\ m \end{array} \right\} + \left\{ \begin{array}{l} n \\ m-1 \end{array} \right\}.
\end{aligned} \tag{46}$$

Inwersja (porównaj z (33)):

$$\sum_k \begin{bmatrix} n \\ k \end{bmatrix} \left\{ \begin{array}{l} k \\ m \end{array} \right\} (-1)^{n-k} = \delta_{mn}, \quad \sum_k \left\{ \begin{array}{l} n \\ k \end{array} \right\} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{n-k} = \delta_{mn}. \tag{47}$$

Wartości szczególne:

$$\binom{0}{n} = \begin{bmatrix} 0 \\ n \end{bmatrix} = \left\{ \begin{array}{l} 0 \\ n \end{array} \right\} = \delta_{n0}, \quad \binom{n}{n} = \begin{bmatrix} n \\ n \end{bmatrix} = \left\{ \begin{array}{l} n \\ n \end{array} \right\} = 1; \tag{48}$$

$$\begin{bmatrix} n \\ n-1 \end{bmatrix} = \left\{ \begin{array}{l} n \\ n-1 \end{array} \right\} = \binom{n}{2}; \tag{49}$$

$$\begin{bmatrix} n+1 \\ 0 \end{bmatrix} = \left\{ \begin{array}{l} n+1 \\ 0 \end{array} \right\} = 0, \quad \begin{bmatrix} n+1 \\ 1 \end{bmatrix} = n!, \quad \left\{ \begin{array}{l} n+1 \\ 1 \end{array} \right\} = 1, \quad \left\{ \begin{array}{l} n+1 \\ 2 \end{array} \right\} = 2^n - 1. \tag{50}$$

Rozwijanie:

$$\sum_k \begin{bmatrix} n \\ k \end{bmatrix} \binom{k}{m} = \begin{bmatrix} n+1 \\ m+1 \end{bmatrix}, \quad \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{k-m} = \begin{bmatrix} n \\ m \end{bmatrix}; \quad (51)$$

$$\sum_k \left\{ \begin{matrix} k \\ m \end{matrix} \right\} \binom{n}{k} = \left\{ \begin{matrix} n+1 \\ m+1 \end{matrix} \right\}, \quad \sum_k \left\{ \begin{matrix} k+1 \\ m+1 \end{matrix} \right\} \binom{n}{k} (-1)^{n-k} = \left\{ \begin{matrix} n \\ m \end{matrix} \right\}; \quad (52)$$

$$\sum_k \binom{m}{k} (-1)^{m-k} k^n = m! \left\{ \begin{matrix} n \\ m \end{matrix} \right\}; \quad (53)$$

$$\begin{aligned} \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left\{ \begin{matrix} m+k \\ k \end{matrix} \right\} &= \begin{bmatrix} n \\ n-m \end{bmatrix}, \\ \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix} &= \left\{ \begin{matrix} n \\ n-m \end{matrix} \right\}; \end{aligned} \quad (54)$$

$$\sum_k \left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{k-m} = \binom{n}{m}; \quad (55)$$

$$\sum_{k \leq n} \begin{bmatrix} k \\ m \end{bmatrix} \frac{n!}{k!} = \begin{bmatrix} n+1 \\ m+1 \end{bmatrix}, \quad \sum_{k \leq n} \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (m+1)^{n-k} = \left\{ \begin{matrix} n+1 \\ m+1 \end{matrix} \right\}. \quad (56)$$

Kilka innych zasadniczych tożsamości dotyczących liczb Stirlinga pojawia się w ćwiczeniach 1.2.6–61 i 1.2.7–6, a także we wzorach (23), (26), (27) i (28) w punkcie 1.2.9.

Równanie (49) to jeden z przejawów ogólniejszego zjawiska: liczby Stirlinga $\begin{bmatrix} n \\ n-m \end{bmatrix}$ i $\left\{ \begin{matrix} n \\ n-m \end{matrix} \right\}$ są wielomianami zmiennej n stopnia $2m$, gdy m jest nieujemną liczbą całkowitą. Na przykład, wzory dla $m = 2$ i $m = 3$ są następujące:

$$\begin{aligned} \begin{bmatrix} n \\ n-2 \end{bmatrix} &= \binom{n}{4} + 2 \binom{n+1}{4}, & \left\{ \begin{matrix} n \\ n-2 \end{matrix} \right\} &= \binom{n+1}{4} + 2 \binom{n}{4}, \\ \begin{bmatrix} n \\ n-3 \end{bmatrix} &= \binom{n}{6} + 8 \binom{n+1}{6} + 6 \binom{n+2}{6}, & \left\{ \begin{matrix} n \\ n-3 \end{matrix} \right\} &= \binom{n+2}{6} + 8 \binom{n+1}{6} + 6 \binom{n}{6}. \end{aligned} \quad (57)$$

Stąd ma sens zdefiniowanie liczb $\begin{bmatrix} r \\ r-m \end{bmatrix}$ i $\left\{ \begin{matrix} r \\ r-m \end{matrix} \right\}$ dla dowolnych rzeczywistych (lub zespolonych) wartości r . Przy takim uogólnieniu oba rodzaje liczb Stirlinga związane są ciekawym prawem dualności

$$\left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \begin{bmatrix} -m \\ -n \end{bmatrix}, \quad (58)$$

o którym Stirling wspomina w swojej analizie. Co więcej, wciąż prawdziwe jest równanie (45) w tym sensie, że nieskończony szereg

$$z^r = \sum_k \left\{ \begin{matrix} r \\ r-k \end{matrix} \right\} z^{r-k} \quad (59)$$

jest zbieżny, jeśli tylko część rzeczywista z jest dodatnia. Bliźniaczy wzór (44) ma podobne uogólnienie, przyjmuje jednak postać szeregu asymptotycznego (ale nie zbieżnego):

$$z^r = \sum_{k=0}^m \begin{bmatrix} r \\ r-k \end{bmatrix} (-1)^k z^{r-k} + O(z^{r-m-1}). \quad (60)$$

(Zobacz ćwiczenie 65). W książce *CMath* (podrozdz. 6.1, 6.2 i 6.5) znajduje się więcej informacji na temat liczb Stirlinga oraz przekształcania zawierających je wzorów. Ćwiczenie 4.7–21 dotyczy rodziny trójkątów zawierających trójkąt Stirlinga jako przypadek szczególny.

ĆWICZENIA

1. [00] Ile jest kombinacji n przedmiotów branych po $n - 1$ naraz?
2. [00] Ile wynosi $\binom{0}{0}$?
3. [00] Na ile sposobów można rozdać karty w grze w brydża? (13 kart na rękę spośród 52 kart w talii).
4. [10] Zapisz odpowiedź do ćwiczenia 3 jako iloczyn liczb pierwszych.
- ▶ 5. [05] Za pomocą trójkąta Pascala wyjaśnij, dlaczego $11^4 = 14641$.
- ▶ 6. [10] Trójkąt Pascala (tabela 1) można rozszerzać we wszystkie strony za pomocą wzoru (9). Wyznacz trzy wiersze *ponad* tabelą 1 (tj. dla $r = -1, -2$ i -3).
7. [12] Dla jakiego k dwumian Newtona $\binom{n}{k}$ przyjmuje wartość maksymalną, gdy n jest ustaloną dodatnią liczbą całkowitą?
8. [00] Jaka własność trójkąta Pascala jest związana z warunkiem symetrii (6)?
9. [01] Ile wynosi $\binom{n}{n}$? (Rozważ wszystkie całkowite n).
- ▶ 10. [M25] Pokaż, że jeśli p jest liczbą pierwszą, to:
 - a) $\binom{n}{p} \equiv \left\lfloor \frac{n}{p} \right\rfloor \pmod{p}$.
 - b) $\binom{p}{k} \equiv 0 \pmod{p}$, dla $1 \leq k \leq p-1$.
 - c) $\binom{p-1}{k} \equiv (-1)^k \pmod{p}$, dla $0 \leq k \leq p-1$.
 - d) $\binom{p+1}{k} \equiv 0 \pmod{p}$, dla $2 \leq k \leq p-1$.
 - e) (É. Lucas, 1877)

$$\binom{n}{k} \equiv \binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor} \binom{n \bmod p}{k \bmod p} \pmod{p}.$$

f) Jeżeli liczby n i k w pozycyjnym systemie liczbowym o podstawie p mają reprezentację

$$n = a_r p^r + \cdots + a_1 p + a_0, \quad k = b_r p^r + \cdots + b_1 p + b_0, \quad \text{to} \quad \binom{n}{k} \equiv \binom{a_r}{b_r} \cdots \binom{a_1}{b_1} \binom{a_0}{b_0} \pmod{p}.$$

- ▶ 11. [M20] (E. Kummer, 1852) Niech p będzie liczbą pierwszą. Pokaż, że jeśli p^n dzieli

$$\binom{a+b}{a},$$

ale p^{n+1} już nie, to n jest równie liczbie *przeniesień* występujących podczas dodawania a do b w pozycyjnym systemie liczbowym o podstawie p . [Wskazówka: Zobacz ćwiczenie 1.2.5–12].

12. [M22] Czy istnieje dodatnia liczba całkowita n , dla której wszystkie niezerowe pozycje w n -tym wierszu trójkąta Pascala są *nieparzyste*? Jeśli tak, znajdź wszystkie liczby o tej własności.

13. [M13] Udowodnij wzór (10).

14. [M21] Oblicz $\sum_{k=0}^n k^4$.

15. [M15] Udowodnij wzór dwumianowy Newtona (13).

16. [M15] Przy założeniu, że n i k są dodatnie i całkowite, udowodnij symetryczną tożsamość

$$(-1)^n \binom{-n}{k-1} = (-1)^k \binom{-k}{n-1}.$$

► **17.** [M18] Wykaż prawdziwość wzoru Chu-Vandermonde'a (21) za pomocą wzoru (15), korzystając z tego, że $(1+x)^{r+s} = (1+x)^r(1+x)^s$.

18. [M15] Udowodnij (22) za pomocą (21) i (6).

19. [M18] Udowodnij (23) przez indukcję.

20. [M20] Udowodnij (24) za pomocą (21) i (19), a następnie pokaż, że w wyniku kolejnego zastosowania (19) otrzyma się (25).

► **21.** [M05] Obie strony (25) są wielomianami zmiennej s . Dlaczego ta równość nie jest tożsamością ze względu na s ?

22. [M20] Udowodnij szczególny przypadek $s = n - 1 - r + nt$ równości (26).

23. [M13] Udowodnij, że jeżeli równanie (26) jest prawdziwe dla (r, s, t, n) oraz $(r, s-t, t, n-1)$, to jest prawdziwe również dla $(r, s+1, t, n)$.

24. [M15] Wyjaśnij, dlaczego wyniki poprzednich dwóch ćwiczeń składają się na dowód wzoru (26).

25. [HM30] Zdefiniujmy wielomian $A_n(x, t)$ jak we wzorze (30). Niech $z = x^{t+1} - x^t$. Udowodnij, że $\sum_k A_k(r, t) z^k = x^r$ dla dostatecznie małych z . [Uwaga: Powyższy fakt, będący uogólnieniem twierdzenia dwumianowego, przyjmuje dokładnie jego postać dla $t = 0$. W dowodzie można założyć prawdziwość twierdzenia dwumianowego (15)]. Wskazówka: Zacznij od tożsamości

$$\sum_j (-1)^j \binom{k}{j} \binom{r-jt}{k} \frac{r}{r-jt} = \delta_{k0}.$$

26. [HM25] Przy założeniach jak w poprzednim ćwiczeniu udowodnij

$$\sum_k \binom{r-tk}{k} z^k = \frac{x^{r+1}}{(t+1)x - t}.$$

27. [HM21] Rozwiąż przykład 4 z tekstu, posługując się wynikiem ćwiczenia 25. Udowodnij wzór (26), korzystając z poprzednich dwóch ćwiczeń. [Wskazówka: Zobacz ćwiczenie 17].

28. [M25] Udowodnij, że

$$\sum_k \binom{r+tk}{k} \binom{s-tk}{n-k} = \sum_{k \geq 0} \binom{r+s-k}{n-k} t^k,$$

jeśli n jest nieujemną liczbą całkowitą.

- 29.** [M20] Pokaż, że wzór (34) jest szczególnym przypadkiem tożsamości udowodnionej w ćwiczeniu 1.2.3–33.
- **30.** [M24] Pokaż, że jest lepszy sposób rozwiązania przykładu 3 niż opisany w tekście; tak przekształć sumę, by dało się zastosować wzór (26).
- **31.** [M20] Oblicz

$$\sum_k \binom{m-r+s}{k} \binom{n+r-s}{n-k} \binom{r+k}{m+n}$$

względem r, s, m , i n , wiedząc, że m i n są nieujemnymi liczbami całkowitymi. Zaczynij od zastąpienia

$$\binom{r+k}{m+n} \quad \text{przez} \quad \sum_j \binom{r}{m+n-j} \binom{k}{j}.$$

32. [M20] Pokaż, że $\sum_k \binom{n}{k} x^k = x^{\bar{n}}$, gdzie $x^{\bar{n}}$ jest potęgą przyrostającą zdefiniowaną w 1.2.5–(19).

33. [M20] (*Suma Capelliego*) Pokaż, że wzór wielomianowy jest prawdziwy także wtedy, gdy zwykłe potęgi zastąpimy potęgami przyrostającymi. Innymi słowy, udowodnij tożsamość $(x+y)^{\bar{n}} = \sum_k \binom{n}{k} x^{\bar{k}} y^{\bar{n-k}}$.

34. [M23] (*Suma Torelliego*) Podążając tropem poprzedniego ćwiczenia, pokaż, że pochodzące od Abela uogólnienie wzoru dwumianowego, wzór (16), jest prawdziwe także dla potęg przyrostujących:

$$(x+y)^{\bar{n}} = \sum_k \binom{n}{k} x(x-kz+1)^{\bar{k-1}} (y+kz)^{\bar{n-k}}.$$

35. [M23] Wyprowadź wzory (46) bezpośrednio z równań (44) i (45) definiujących liczby Stirlinga.

36. [M10] Ile wynosi $\sum_k \binom{n}{k}$, czyli suma liczb w każdym wierszu trójkąta Pascala? Jaka byłaby suma tych liczb, gdybyśmy co drugiej zmienili znak, tj. $\sum_k \binom{n}{k} (-1)^k$?

37. [M10] Z odpowiedzi do poprzednich ćwiczeń wywnioskuj, jaka będzie wartość sumy parzystych pozycji w wierszu trójkąta Pascala, $\binom{n}{0} + \binom{n}{2} + \binom{n}{4} + \dots$

38. [HM30] (C. Ramus, 1834) Uogólniając wynik poprzedniego ćwiczenia, pokaż, że dla $0 \leq k < m$ prawdziwy jest następujący wzór:

$$\binom{n}{k} + \binom{n}{m+k} + \binom{n}{2m+k} + \dots = \frac{1}{m} \sum_{0 \leq j < m} \left(2 \cos \frac{j\pi}{m} \right)^n \cos \frac{j(n-2k)\pi}{m}.$$

Na przykład

$$\binom{n}{1} + \binom{n}{4} + \binom{n}{7} + \dots = \frac{1}{3} \left(2^n + 2 \cos \frac{(n-2)\pi}{3} \right).$$

[*Wskazówka:* Znajdź odpowiednią kombinację tych współczynników pomnożoną przez m -ty pierwiastek z jedności]. Ta tożsamość jest szczególnie istotna, gdy $m \geq n$.

39. [M10] Ile wynosi suma $\sum_k \binom{n}{k}$ liczb w wierszu trójkąta Stirlinga pierwszego rodzaju? Jaka będzie suma tych liczb, gdy co drugiej zmienimy znak? (Zobacz ćwiczenie 36).

40. [HM17] Funkcja beta $B(x, y)$ jest zdefiniowana dla dodatnich liczb rzeczywistych x, y wzorem $B(x, y) = \int_0^1 t^{x-1} (1-t)^{y-1} dt$.

- a) Pokaż, że $B(x, 1) = B(1, x) = 1/x$.
- b) Pokaż, że $B(x+1, y) + B(x, y+1) = B(x, y)$.
- c) Pokaż, że $B(x, y) = ((x+y)/y) B(x, y+1)$.

41. [HM22] Pokazaliśmy związek między funkcją gamma a funkcją beta z ćwiczenia 1.2.5–19, udowadniając, że $\Gamma_m(x) = m^x B(x, m+1)$, gdy m jest dodatnią liczbą całkowitą.

- a) Udowodnij, że

$$B(x, y) = \frac{\Gamma_m(y)m^x}{\Gamma_m(x+y)} B(x, y+m+1).$$

- b) Pokaż, że

$$B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}.$$

42. [HM10] Wyraź współczynnik dwumianowy $\binom{r}{k}$ za pomocą zdefiniowanej powyżej funkcji beta. (W ten sposób możemy rozciągnąć definicję na wszystkie rzeczywiste wartości k).

43. [HM20] Pokaż, że $B(1/2, 1/2) = \pi$. (Można oprzeć się na wniosku z ćwiczenia 41, że $\Gamma(1/2) = \sqrt{\pi}$).

44. [HM20] Za pomocą uogólnionych współczynników dwumianowych zaprezentowanych w ćwiczeniu 42 pokaż, że

$$\binom{r}{1/2} = 2^{2r+1} / \binom{2r}{r} \pi.$$

45. [HM21] Za pomocą uogólnionych współczynników dwumianowych zaprezentowanych w ćwiczeniu 42 pokaż, że $\lim_{r \rightarrow \infty} \binom{r}{k} / r^k$.

► **46.** [M21] Posługując się przybliżeniem Stirlinga 1.2.5–(7), znajdź przybliżoną wartość $\binom{x+y}{y}$ przy założeniu, że wartości x i y są duże. W szczególności, znajdź przybliżoną wartość $\binom{2n}{n}$ dla dużych n .

47. [M21] Pokaż, że jeżeli k jest liczbą całkowitą, to

$$\binom{r}{k} \binom{r-1/2}{k} = \binom{2r}{k} \binom{2r-k}{k} / 4^k = \binom{2r}{2k} \binom{2k}{k} / 4^k.$$

Podaj prostszy wzór dla przypadku szczególnego $r = -1/2$.

► **48.** [M25] Pokaż, że

$$\sum_{k \geq 0} \binom{n}{k} \frac{(-1)^k}{k+x} = \frac{n!}{x(x+1)\dots(x+n)} = \frac{1}{x^{\binom{n+x}{n}}},$$

jeżeli mianowniki są niezerowe. [Zauważ, że za pomocą tego wzoru można obliczyć odwrotność współczynnika dwumianowego oraz uzyskać rozwinięcie $1/x(x+1)\dots(x+n)$ za pomocą ułamków prostych].

49. [M20] Pokaż, że z tożsamości $(1+x)^r = (1-x^2)^r (1-x)^{-r}$ wynika pewien związek między współczynnikami dwumianowymi.

50. [M20] Udowodnij wzór Abela, czyli równanie (16), dla przypadku szczególnego $x+y=0$.

51. [M21] Udowodnij wzór Abela, czyli równanie (16), zapisując $y = (x + y) - x$, rozwijając prawą stronę jako potęgi $(x + y)$ i stosując wynik poprzedniego ćwiczenia.

52. [HM11] Obliczając prawą stronę wzoru dwumianowego Abela (16) dla $n = x = -1$ i $y = z = 1$, pokaż, że wzór może nie być prawdziwy, gdy n nie jest nieujemną liczbą całkowitą.

53. [M25] (a) Udowodnij poniższą tożsamość przez indukcję względem m , gdzie m i n są liczbami całkowitymi:

$$\sum_{k=0}^m \binom{r}{k} \binom{s}{n-k} (nr - (r+s)k) = (m+1)(n-m) \binom{r}{m+1} \binom{s}{n-m}.$$

(b) Korzystając z istotnych związków wyprowadzonych w ćwiczeniu 47:

$$\binom{-1/2}{n} = \frac{(-1)^n}{2^{2n}} \binom{2n}{n}, \quad \binom{1/2}{n} = \frac{(-1)^{n-1}}{2^{2n}(2n-1)} \binom{2n}{n} = \frac{(-1)^{n-1}}{2^{2n-1}(2n-1)} \binom{2n-1}{n} - \delta_{n0},$$

pokaż, że poniższy wzór można otrzymać jako przypadek szczególny tożsamości z punktu (a):

$$\sum_{k=0}^m \binom{2k-1}{k} \binom{2n-2k}{n-k} \frac{-1}{2k-1} = \frac{n-m}{2n} \binom{2m}{m} \binom{2n-2m}{n-m} + \frac{1}{2} \binom{2n}{n}.$$

(Ten wynik jest znacznie bardziej ogólny niż wzór (26) w przypadku $r = -1$, $s = 0$, $t = -2$).

54. [M21] Potraktuj trójkąt Pascala (tabela 1) jak macierz. Jaka jest jej *macierz odwrotna*?

55. [M21] Potraktuj oba trójkąty Stirlinga (tabela 2) jak macierze i oblicz ich macierze odwrotne.

56. [20] (*Kombinatoryczny system liczbowy*) Dla każdej z całkowitych liczb $n = 0, 1, 2, \dots, 20$ znajdź trzy liczby całkowite a, b, c , dla których $n = \binom{a}{1} + \binom{b}{2} + \binom{c}{3}$ i $0 \leq a < b < c$. Czy dostrzegasz prawidłowość pozwalającą wypisać reprezentacje dla większych n ?

► **57.** [M22] Pokaż, że współczynnik a_m w podejściu Stirlinga do uogólnienia silni, wzór 1.2.5-(12), wynosi

$$\frac{(-1)^m}{m!} \sum_{k \geq 1} (-1)^k \binom{m-1}{k-1} \ln k.$$

58. [M23] Udowodnij „twierdzenie q -mianowe”, przy oznaczeniach jak w (40):

$$(1+x)(1+qx)\dots(1+q^{n-1}x) = \sum_k \binom{n}{k}_q q^{k(k-1)/2} x^k.$$

Znajdź q -mianowe uogólnienie podstawowych tożsamości (17) i (21).

59. [M25] Ciąg liczb A_{nk} , $n \geq 0$, $k \geq 0$, spełnia równości $A_{n0} = 1$, $A_{0k} = \delta_{0k}$, $A_{nk} = A_{(n-1)k} + A_{(n-1)(k-1)} + \binom{n}{k}$ dla $nk > 0$. Znайдź A_{nk} .

► **60.** [M23] Stwierdziliśmy, że $\binom{n}{k}$ jest liczbą kombinacji n przedmiotów branych po k naraz, czyli liczbą sposobów wybrania k różnych elementów z n -elementowego zbioru. Kombinacje z powtórzeniami są podobne do zwykłych kombinacji, jednak możemy wybrać jeden element dowolną liczbę razy. Jeśli zatem rozważalibyśmy kombinacje z powtórzeniami, do listy (1) należałoby dołożyć $aaa, aab, aac, aad, aae, abb$ itd. Ile jest kombinacji k -elementowych ze zbioru n elementów, jeżeli dopuścimy powtórzenia?

61. [M25] Oblicz sumę

$$\sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \left\{ \begin{array}{c} k \\ m \end{array} \right\} (-1)^{k-m},$$

żeby wzór (55) nie czuł się samotny.

► **62.** [M23] W tekście są podane wzory na sumy zawierające iloczyn dwóch współczynników dwumianowych. Z sum zawierających trzy współczynniki dwumianowe najbardziej pożyteczna wydaje się tożsamość z ćwiczenia 31 oraz następujący wzór:

$$\sum_k (-1)^k \binom{l+m}{l+k} \binom{m+n}{m+k} \binom{n+l}{n+k} = \frac{(l+m+n)!}{l! m! n!}, \quad \text{całkowite } l, m, n \geq 0.$$

(Ta suma obejmuje zarówno dodatnie, jak i ujemne wartości k). Udowodnij tę tożsamość. [Wskazówka: Istnieje bardzo krótki dowód, rozpoczynający się od zastosowania wyniku ćwiczenia 31].

63. [M30] Dla całkowitych liczb l, m i n , gdzie $n \geq 0$, udowodnij

$$\sum_{j,k} (-1)^{j+k} \binom{j+k}{k+l} \binom{r}{j} \binom{n}{k} \binom{s+n-j-k}{m-j} = (-1)^l \binom{n+r}{n+l} \binom{s-r}{m-n-l}.$$

► **64.** [M20] Pokaż, że $\left\{ \begin{array}{c} n \\ m \end{array} \right\}$ jest liczbą podziałów zbioru n -elementowego na m niepustych rozłącznych podzbiorów. Na przykład zbiór $\{1, 2, 3, 4\}$ można podzielić na dwa podzbiory na $\left\{ \begin{array}{c} 4 \\ 2 \end{array} \right\} = 7$ sposobów: $\{1, 2, 3\}\{4\}$; $\{1, 2, 4\}\{3\}$; $\{1, 3, 4\}\{2\}$; $\{2, 3, 4\}\{1\}$; $\{1, 2\}\{3, 4\}$; $\{1, 3\}\{2, 4\}$; $\{1, 4\}\{2, 3\}$. Wskazówka: Skorzystaj ze wzoru (46).

65. [HM35] (B. F. Logan) Udowodnij równania (59) i (60).

66. [M29] Niech n będzie dodatnią liczbą całkowitą oraz niech x i y będą liczbami rzeczywistymi spełniającymi nierówności $n \leq y \leq x \leq y+1$. Wówczas zachodzi $\binom{y}{n+1} \leq \binom{x}{n+1} \leq \binom{y+1}{n+1} = \binom{y}{n+1} + \binom{y}{n}$, zatem istnieje dokładnie jedna liczba rzeczywista z , taka że

$$\binom{x}{n+1} = \binom{y}{n+1} + \binom{z}{n}, \quad n-1 \leq z \leq y.$$

Udowodnij, że

$$\binom{x}{n} \leq \binom{y}{n} + \binom{z}{n-1}.$$

[Wskazówka: Zastanów się nad rozwinięciem $\binom{x}{n+1} = \binom{z+1}{n+1} + \sum_{k \geq 0} \binom{z-k}{n-k} \binom{x-z-1+k}{k+1}$].

► **67.** [M20] Często chcielibyśmy mieć pewność, że współczynniki dwumianowe nie mają zbyt dużych wartości. Udowodnij łatwe do zapamiętania ograniczenie górne

$$\binom{n}{k} \leq \left(\frac{ne}{k} \right)^k, \quad \text{gdy } n \geq k \geq 0.$$

68. [M25] (A. de Moivre) Udowodnij, że jeżeli n jest nieujemną liczbą całkowitą, to

$$\sum_k \binom{n}{k} p^k (1-p)^{n-k} |k - np| = 2 \lceil np \rceil \binom{n}{\lceil np \rceil} p^{\lceil np \rceil} (1-p)^{n+1-\lceil np \rceil}.$$

1.2.7. Liczby harmoniczne

W naszych dalszych rozważaniach istotną rolę odgrywa suma:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^n \frac{1}{k}, \quad n \geq 0. \quad (1)$$

Ta suma nie pojawia się zbyt często w klasycznej matematyce i nie ma dla niej standardowej notacji, ale w analizie algorytmów spotykamy ją na każdym kroku i konsekwentnie będziemy nazywać ją H_n . (W literaturze matematycznej poza H_n spotyka się także oznaczenia h_n , S_n oraz $\psi(n+1) + \gamma$. Litera H pochodzi od „harmoniczny”, a H_n czytamy *liczba harmoniczna* ponieważ szereg (1) tradycyjnie nazywa się szeregiem harmonicznym).

Na pierwszy rzut oka może się wydawać, że wartość H_n nie może urosnąć zbyt wysoko dla dużych n , bo dodajemy liczby coraz mniejsze i mniejsze. Jednak nietrudno zauważać, że H_n może uzyskać wartość tak dużą, jak dusza zapragnie, jeśli weźmiemy odpowiednio duże n , ponieważ

$$H_{2^m} \geq 1 + \frac{m}{2}. \quad (2)$$

To dolne ograniczenie można wysnuć z obserwacji, że dla $m \geq 0$ mamy

$$\begin{aligned} H_{2^{m+1}} &= H_{2^m} + \frac{1}{2^m + 1} + \frac{1}{2^m + 2} + \cdots + \frac{1}{2^{m+1}} \\ &\geq H_{2^m} + \frac{1}{2^{m+1}} + \frac{1}{2^{m+1}} + \cdots + \frac{1}{2^{m+1}} = H_{2^m} + \frac{1}{2}. \end{aligned}$$

Jeśli zatem m rośnie o 1, lewa strona (2) rośnie przynajmniej o $\frac{1}{2}$.

Nie poprzestawajmy jednak na nierówności (2). Przybliżona wartość H_n jest wielkością powszechnie znaną (no, przynajmniej w kręgach matematyków):

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \epsilon, \quad 0 < \epsilon < \frac{1}{252n^6}. \quad (3)$$

W tym wzorze $\gamma = 0.5772156649\dots$ oznacza *stała Eulera*, wprowadzoną przez Leonarda Eulera w *Commentarii Acad. Sci. Imp. Pet.* 7 (1734), 150–161. Dokładne wartości H_n dla małych n oraz wartość γ z dokładnością do 40 miejsc po przecinku są podane w tabelach w dodatku A. Równość (3) wyprowadzimy w podpunkcie 1.2.11.2.

Widać więc, że wartość H_n jest bliska wartości logarytmu naturalnego z n . Z ćwiczenia 7(a) w prosty sposób wynika, że H_n zachowuje się mniej więcej logarymicznie.

Wartość H_n dąży do nieskończoności przy rosnącym n . Podobna suma

$$1 + \frac{1}{2^r} + \frac{1}{3^r} + \cdots + \frac{1}{n^r} \quad (4)$$

jest ograniczona dla wszystkich n , gdy r jest dowolnym wykładnikiem rzeczywistym *większym* od 1. (Zobacz ćwiczenie 3). Sumę (4) oznaczamy przez $H_n^{(r)}$.

We wzorze (4) dla wykładników r większych bądź równych 2 wartość $H_n^{(r)}$ jest (poza małymi wartościami n) bardzo bliska $H_\infty^{(r)}$. W matematyce wielkość $H_\infty^{(r)}$ jest szeroko znana pod nazwą *funkcji dzeta Riemanna*:

$$H_\infty^{(r)} = \zeta(r) = \sum_{k \geq 1} \frac{1}{k^r}. \quad (5)$$

Wiadomo, że dla r będącego parzystą liczbą całkowitą wartość $\zeta(r)$ wynosi

$$H_\infty^{(r)} = \frac{1}{2} |B_r| \frac{(2\pi)^r}{r!}, \quad \text{całkowite } r/2 \geq 1, \quad (6)$$

gdzie B_r jest liczbą Bernoulliego (zobacz podpunkt 1.2.11.2 oraz dodatek A). W szczególności

$$H_\infty^{(2)} = \frac{\pi^2}{6}, \quad H_\infty^{(4)} = \frac{\pi^4}{90}, \quad H_\infty^{(6)} = \frac{\pi^6}{945}, \quad H_\infty^{(8)} = \frac{\pi^8}{9450}. \quad (7)$$

Powyższe odkrycia zawdzięczamy Eulerowi. Omówienie i dowód znajdzie Czytelnik w *CMath*, §6.5.

Rozważymy teraz kilka istotnych sum zawierających liczby harmoniczne. Zaczniemy od

$$\sum_{k=1}^n H_k = (n+1)H_n - n. \quad (8)$$

Równość wynika z prostej zamiany kolejności sumowania:

$$\sum_{k=1}^n \sum_{j=1}^k \frac{1}{j} = \sum_{j=1}^n \sum_{k=j}^n \frac{1}{j} = \sum_{j=1}^n \frac{n+1-j}{j}.$$

Wzór (8) jest przypadkiem szczególnym sumy $\sum_{k=1}^n \binom{k}{m} H_k$, którą obliczymy za pomocą ważnej metody nazywanej sumowaniem przez części (zobacz ćwiczenie 10). Sumowanie przez części przydaje się w sytuacjach, gdy trzeba policzyć sumę $\sum a_k b_k$, a $\sum a_k$ i $(b_{k+1} - b_k)$ mają proste formy. W naszym przypadku

$$\binom{k}{m} = \binom{k+1}{m+1} - \binom{k}{m+1},$$

a stąd

$$\binom{k}{m} H_k = \binom{k+1}{m+1} (H_{k+1} - \frac{1}{k+1}) - \binom{k}{m+1} H_k,$$

zatem

$$\begin{aligned} \sum_{k=1}^n \binom{k}{m} H_k &= \left(\binom{2}{m+1} H_2 - \binom{1}{m+1} H_1 \right) + \dots \\ &\quad + \left(\binom{n+1}{m+1} H_{n+1} - \binom{n}{m+1} H_n \right) - \sum_{k=1}^n \binom{k+1}{m+1} \frac{1}{k+1} \\ &= \binom{n+1}{m+1} H_{n+1} - \binom{1}{m+1} H_1 - \frac{1}{m+1} \sum_{k=0}^n \binom{k}{m} + \frac{1}{m+1} \binom{0}{m}. \end{aligned}$$

Stosujemy 1.2.6–(11) i otrzymujemy poszukiwany wzór:

$$\sum_{k=1}^n \binom{k}{m} H_k = \binom{n+1}{m+1} \left(H_{n+1} - \frac{1}{m+1} \right). \quad (9)$$

(Powyższe wyprowadzenie oraz jego ostateczny wynik przypominają obliczanie

$$\int_1^n x^m \ln x \, dx = \frac{n^{m+1}}{m+1} \left(\ln n - \frac{1}{m+1} \right) + \frac{1}{(m+1)^2}$$

za pomocą metody nazywanej w książkach do analizy całkowaniem przez części).

Na zakończenie niniejszego rozdziału rozważymy sumę $\sum_k \binom{n}{k} x^k H_k$, którą na potrzeby poniższych rozważań oznaczać będziemy S_n . Mamy

$$\begin{aligned} S_{n+1} &= \sum_k \left(\binom{n}{k} + \binom{n}{k-1} \right) x^k H_k = S_n + x \sum_{k \geq 1} \binom{n}{k-1} x^{k-1} \left(H_{k-1} + \frac{1}{k} \right) \\ &= S_n + x S_n + \frac{1}{n+1} \sum_{k \geq 1} \binom{n+1}{k} x^k. \end{aligned}$$

Stąd $S_{n+1} = (x+1)S_n + ((x+1)^{n+1} - 1)/(n+1)$, zatem otrzymujemy

$$\frac{S_{n+1}}{(x+1)^{n+1}} = \frac{S_n}{(x+1)^n} + \frac{1}{n+1} - \frac{1}{(n+1)(x+1)^{n+1}}.$$

Z powyższej równości oraz z tego, że $S_1 = x$, wynika

$$\frac{S_n}{(x+1)^n} = H_n - \sum_{k=1}^n \frac{1}{k(x+1)^k}. \quad (10)$$

Suma po prawej stronie równania jest fragmentem rozwinięcia w szereg funkcji $\ln(1/(1-1/(x+1))) = \ln(1+1/x)$, a dla $x > 0$ ten szereg jest zbieżny. Różnica wynosi

$$\sum_{k>n} \frac{1}{k(x+1)^k} < \frac{1}{(n+1)(x+1)^{n+1}} \sum_{k \geq 0} \frac{1}{(x+1)^k} = \frac{1}{(n+1)(x+1)^n x},$$

co stanowi dowód następującego twierdzenia:

Twierdzenie A. Jeżeli $x > 0$, to

$$\sum_{k=1}^n \binom{n}{k} x^k H_k = (x+1)^n \left(H_n - \ln\left(1 + \frac{1}{x}\right)\right) + \epsilon,$$

gdzie $0 < \epsilon < 1/(x(n+1))$. ■

ĆWICZENIA

1. [01] Jakie są wartości H_0 , H_1 i H_2 ?
2. [13] Pokaż, że proste rozumowanie użyte w tekście do pokazania nierówności $H_{2^m} \geq 1 + m/2$, można tak zmodyfikować, by pokazać, że $H_{2^m} \leq 1 + m$.

3. [M21] Uogólnij rozumowanie z poprzedniego ćwiczenia, by pokazać, że dla $r > 1$ suma $H_n^{(r)}$ jest ograniczona względem zmiennej n . Znajdź górne ograniczenie.

► **4.** [10] Rozstrzygnij, które z poniższych stwierdzeń są prawdziwe dla wszystkich dodatnich liczb całkowitych n : (a) $H_n < \ln n$; (b) $H_n > \ln n$; (c) $H_n > \ln n + \gamma$.

5. [15] Podaj wartość H_{10000} z dokładnością do 15 miejsc po przecinku. Skorzystaj z tabel w dodatku A.

6. [M15] Udowodnij, że liczby harmoniczne są ścisłe powiązane z liczbami Stirlinga (opisanymi w poprzednim rozdziale), a konkretne

$$H_n = \left[\frac{n+1}{2} \right] / n!.$$

7. [M21] Niech $T(m, n) = H_m + H_n - H_{mn}$. (a) Pokaż, że gdy m i n rosną, $T(m, n)$ nie rośnie (przy założeniu, że m i n są dodatnie). (b) Oblicz minimalną i maksymalną wartość $T(m, n)$ dla $m, n > 0$.

8. [HM18] Porównaj wzór (8) i $\sum_{k=1}^n \ln k$; oszacuj różnicę jako funkcję n .

► **9.** [M18] Twierdzenie A dotyczy wyłącznie $x > 0$. Jaka jest wartość wymienionej w nim sumy dla $x = -1$.

10. [M20] (*Sumowanie przez części*) W ćwiczeniu 1.2.4–42 oraz wyprowadzeniu wzoru (9) używaliśmy szczególnego przypadku ogólnej metody sumowania przez części. Udowodnij wzór

$$\sum_{1 \leq k < n} (a_{k+1} - a_k) b_k = a_n b_n - a_1 b_1 - \sum_{1 \leq k < n} a_{k+1} (b_{k+1} - b_k).$$

► **11.** [M21] Posługując się sumowaniem przez części, oblicz

$$\sum_{1 < k \leq n} \frac{1}{k(k-1)} H_k.$$

► **12.** [M10] Oblicz $H_\infty^{(1000)}$ przynajmniej do setnego miejsca po przecinku.

13. [M22] Udowodnij tożsamość

$$\sum_{k=1}^n \frac{x^k}{k} = H_n + \sum_{k=1}^n \binom{n}{k} \frac{(x-1)^k}{k}.$$

(Zauważ, że przypadek $x = 0$ to identyczność podobna do tej z ćwiczenia 1.2.6–48).

14. [M22] Pokaż, że $\sum_{k=1}^n H_k/k = \frac{1}{2}(H_n^2 + H_n^{(2)})$ oraz oblicz $\sum_{k=1}^n H_k/(k+1)$.

► **15.** [M23] Zapisz $\sum_{k=1}^n H_k^2$ za pomocą n i H_n .

16. [18] Wyraź sumę $1 + \frac{1}{3} + \dots + \frac{1}{2n-1}$ za pomocą liczb harmonicznych.

17. [M24] (E. Waring, 1782) Niech p będzie nieparzystą liczbą pierwszą. Pokaż, że licznik H_{p-1} jest podzielny przez p .

18. [M33] (J. Selfridge) Jaka jest najwyższa potęga dwójki będąca dzielnikiem licznika ułamka $1 + \frac{1}{3} + \dots + \frac{1}{2n-1}$?

► **19.** [M30] Wypisz wszystkie nieujemne liczby całkowite n , dla których H_n jest liczbą całkowitą. [*Wskazówka:* Gdy H_n ma nieparzysty licznik i parzysty mianownik, to nie może być liczbą całkowitą].

20. [HM22] Istnieje analityczne podejście do sumowania, takie jak rozumowanie prowadzące do twierdzenia A. Wiedząc, że $f(x) = \sum_{k \geq 0} a_k x^k$ i że ten szereg jest zbieżny dla $x = x_0$, udowodnij, że

$$\sum_{k \geq 0} a_k x_0^k H_k = \int_0^1 \frac{f(x_0) - f(xy)}{1-y} dy.$$

21. [M24] Oblicz $\sum_{k=1}^n H_k / (n+1-k)$.

22. [M28] Oblicz $\sum_{k=0}^n H_k H_{n-k}$.

► **23.** [HM20] Zastanów się, w jaki sposób za pomocą $\Gamma'(x)/\Gamma(x)$ można uogólnić H_n na niecałkowite wartości n . Możesz uprzedzić wynik następnego ćwiczenia i skorzystać z faktu, że $\Gamma'(1) = -\gamma$.

24. [HM21] Pokaż, że

$$xe^{\gamma x} \prod_{k \geq 1} \left(\left(1 + \frac{x}{k}\right) e^{-x/k} \right) = \frac{1}{\Gamma(x)}.$$

(Rozważ iloczyny częściowe tego nieskończonego iloczynu).

1.2.8. Liczby Fibonacciego

Ciąg

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \dots, \quad (1)$$

w którym każda liczba jest sumą dwóch poprzednich, odgrywa istotną rolę w kilkunastu interesujących algorytmach, na pozór nie mających ze sobą nic wspólnego. Wyrazy tego ciągu oznaczamy przez F_n i formalnie definiujemy następująco:

$$F_0 = 0; \quad F_1 = 1; \quad F_{n+2} = F_{n+1} + F_n, \quad n \geq 0. \quad (2)$$

Ten znany ciąg został opublikowany w 1202 roku przez Leonarda Pisano (Leonarda z Pizy), zwanego też Leonardem Fibonaccim (*Filius Bonacci*, syn Bonacciego), którego *Liber Abaci* (Książka liczydeł) zawiera następujące zadanie: „Ile par królików zrodzi się z jednej pary w ciągu roku?”. Wiemy, że każda para ma nową parę potomstwa co miesiąc i że ta nowa para staje się płodna po miesiącu. Co więcej, króliki nie zdychają. Po jednym miesiącu będą dwie pary królików, po dwóch – trzy. Następnego miesiąca pierwsza para oraz para urodzona w pierwszym miesiącu wydadzą po nowej parze, więc par będzie pięć itd.

Fibonacci z pewnością był największym europejskim matematykiem średniodwiecznego. Studiował prace al-Khwārizmīego (od którego imienia pochodzi słowo „algorytm”, zobacz podrozdział 1.1), wniósł też istotny wkład w arytmetykę i geometrię. Prace Fibonacciego przedrukowano w 1857 roku [B. Boncompagni, *Scritti di Leonardo Pisano* (Rome, 1857–1862), 2 tomy; F_n pojawia się w tomie 1, 283–285]. Problem dotyczący królików nie był oczywiście przedstawiony jako praktyczne zastosowanie matematyki do zagadnień biologii i eksplozji demograficznej. Był ćwiczeniem na dodawanie. W istocie to wciąż zupełnie niezłe komputerowe ćwiczenie na dodawanie (zobacz ćwiczenie 3). Fibonacci napisał: „A można tak [dodawać] dla miesięcy nieskończonym wielu”.

Zanim Fibonacci napisał swoją pracę, ciąg $\langle F_n \rangle$ rozważali uczeni hinduscy, którzy od dawna interesowali się wzorcami rytmicznymi złożonymi z pojedynczych i podwójnych uderzeń lub sylab. Liczba takich rytmów mających w sumie n uderzeń to F_{n+1} . Stąd zarówno Gopāla (przed 1135 rokiem), jak i Hemachandra (około 1150 roku) wprost podają liczby 1, 2, 3, 5, 8, 13, 21, . . . [Zobacz P. Singh, *Historia Math.* **12** (1985), 229–244; zobacz także ćwiczenie 4.5.3–32].

Ten sam ciąg pojawia się w wydanej w 1611 roku pracy Johanna Keplera, analizującego dostrzegane wokół siebie liczby [J. Kepler, *The Six-Cornered Snowflake* (Oxford: Clarendon Press, 1966), 21]. Kepler zapewne nie zdawał sobie sprawy, że Fibonacci wspomniał o tym samym ciągu. Liczby Fibonacciego często zauważano w przyrodzie, prawdopodobnie z powodów podobnych do założeń zadania o królikach. [Bardzo przejryste wyjaśnienie znajdzie Czytelnik w: J. Conway, R. Guy, *Księga liczb* (Warszawa: WNT, 1999), 120–122].

Pierwsza wskazówka dotycząca niejawnych związków między F_n i algorytmami pojawiła się w 1837 roku, kiedy to É. Léger wykorzystał ciąg Fibonacciego do analizy efektywności algorytmu Euklidesa. Léger dostrzegł, że jeżeli liczby m i n w algorytmie 1.1E są nie większe niż F_k , to krok E2 wykona się co najwyżej $k + 1$ razy. To było pierwsze praktyczne zastosowanie ciągu Fibonacciego. (Zobacz twierdzenie 4.5.3F). W latach siedemdziesiątych XIX wieku matematyk É. Lucas uzyskał nietrywialne wyniki dotyczące liczb Fibonacciego, a w szczególności użył ich do pokazania, że 39-cyfrowa liczba $2^{127} - 1$ jest pierwsza. Lucas nadał ciągowi $\langle F_n \rangle$ nazwę „liczby Fibonacciego”, której używa się po dziś dzień.

W punkcie 1.2.1 zbadaliśmy побieżnie ciąg Fibonacciego (równość (3) i ćwiczenie 4), odkrywając, że $\phi^{n-2} \leq F_n \leq \phi^{n-1}$, gdzie n jest dodatnią liczbą całkowitą, natomiast

$$\phi = \frac{1}{2}(1 + \sqrt{5}). \quad (3)$$

Niedługo przekonamy się, że wielkość ϕ jest ściśle związana z liczbami Fibonacciego.

Sama liczba ϕ ma bardzo ciekawą historię. Euklides nazywał ją proporcją największej do średniej. Jeśli ϕ jest równe stosunkowi A do B , to stosunek $A + B$ do A też równa się ϕ . Pisarze renesansowi zwali ϕ „boską proporcją”, a w ostatnim stuleciu powszechnie nazywano ją „złotą proporcją”. Wielu artystów i pisarzy twierdziło, że proporcja ϕ do 1 jest najprzyjemniejsza dla ludzkiego oka, a ich opinię można potwierdzić z punktu widzenia programowania komputerów. Historia ϕ jest opisana w wyśmienitym artykule „The Golden Section, Phyllotaxis, and Wythoff's Game”, H. S. M. Coxeter, *Scripta Math.* **19** (1953), 135–143; zobacz także rozdział 8 książki *The 2nd Scientific American Book of Mathematical Puzzles and Diversions* autorstwa Martina Gardnера (New York: Simon and Schuster, 1961). Kilka mitów i plotek na temat ϕ zostało zdemaskowanych przez George'a Markowsky'ego w *College Math. J.* **23** (1992), 2–19. Fakt, że stosunek F_{n+1}/F_n dąży do ϕ , był znany europejskiemu rachmistrzowi Simonowi Jacobowi, który zmarł w 1564 roku [zobacz P. Schreiber, *Historia Math.* **22** (1995), 422–424].

Notacja używana w tym rozdziale jest trochę niewłaściwa. Większość poważnych autorów używa oznaczenia u_n na F_n i τ na ϕ . Nasza notacja występuje prawie wszędzie w matematyce rozrywkowej (a także w różnych podejrzanych publikacjach) i zatacza coraz szersze kręgi. Oznaczenie ϕ pochodzi od imienia greckiego rzeźbiarza Fidiasza, który ponoć często wykorzystywał złotą proporcję. Oznaczenie F_n jest używane w *Fibonacci Quarterly*, gdzie można znaleźć wiele informacji na temat ciągu Fibonacciego. Dobre omówienie F_n zawiera rozdział 17 książki L. E. Dicksona *History of the Theory of Numbers 1* (Carnegie Inst. of Washington, 1919).

Liczby Fibonacciego spełniają wiele ciekawych tożsamości, niektóre z nich znajdują się w ćwiczeniach na końcu rozdziału. Jednym z najczęściej odkrywanych związków jest wymieniona przez Keplera w liście z 1608 roku, ale po raz pierwszy opublikowana przez J. D. Cassiniego w [Histoire Acad. Roy. Paris 1 (1680), 201], równość

$$F_{n+1}F_{n-1} - F_n^2 = (-1)^n, \quad (4)$$

która łatwo udowodnić przez indukcję. Bardziej ezoteryczny sposób udowodnienia tego samego wzoru polega na wykazaniu tożsamości macierzowej

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \quad (5)$$

i obliczeniu wyznacznika obu stron.

Ze wzoru (4) wynika, że F_n i F_{n+1} są względnie pierwsze, bo każdy wspólny dzielnik musiałby dzielić $(-1)^n$.

Z definicji (2) mamy natychmiast

$$F_{n+3} = F_{n+2} + F_{n+1} = 2F_{n+1} + F_n; \quad F_{n+4} = 3F_{n+1} + 2F_n$$

i ogólnie przez indukcję

$$F_{n+m} = F_m F_{n+1} + F_{m-1} F_n \quad (6)$$

dla dowolnej dodatniej liczby całkowitej m .

Jeżeli we wzorze (6) weźmiemy za m wielokrotność n , to odkryjemy, że

F_{nk} jest wielokrotnością F_k .

Stąd co trzecia liczba jest parzysta, co czwarta jest wielokrotnością 3, co piąta wielokrotnością 5 itd.

W istocie zachodzi fakt ogólniejszy. Jeżeli przez $\gcd(m, n)$ oznaczymy największy wspólny dzielnik m i n , na jaw wyjdzie zaskakujące twierdzenie:

Twierdzenie A (É. Lucas, 1876). *Dowolna liczba dzieli zarówno F_m , jak i F_n wtedy i tylko wtedy, gdy jest dzielnikiem F_d , gdzie $d = \gcd(m, n)$; w szczególności*

$$\gcd(F_m, F_n) = F_{\gcd(m, n)}. \quad (7)$$

Dowód. W dowodzie korzystamy z algorytmu Euklidesa. Zauważmy, że na mocy (6) każdy wspólny dzielnik F_m i F_n jest także dzielnikiem F_{n+m} . W drugą stronę, każdy wspólny dzielnik F_{n+m} i F_n jest dzielnikiem $F_m F_{n+1}$. Ponieważ F_{n+1} i F_n są względnie pierwsze, wspólny dzielnik F_{n+m} i F_n dzieli także F_m . Udowodniliśmy zatem, że dla dowolnej liczby d :

$$d \text{ dzieli } F_m \text{ i } F_n \text{ wtedy i tylko wtedy, gdy } d \text{ dzieli } F_{m+n} \text{ i } F_n. \quad (8)$$

Pokażemy teraz, że twierdzenie A zachodzi dla dowolnego ciągu $\langle F_n \rangle$, takiego że $F_0 = 0$ oraz prawdziwy jest warunek (8).

Przede wszystkim widać, że warunek (8) można przez indukcję względem k rozszerzyć do

$$d \text{ dzieli } F_m \text{ i } F_n \text{ wtedy i tylko wtedy, gdy } d \text{ dzieli } F_{m+kn} \text{ i } F_n,$$

gdzie k jest dowolną nieujemną liczbą całkowitą. Ten wynik można wyrazić bardziej elegancko:

$$d \text{ dzieli } F_{m \bmod n} \text{ i } F_n \text{ wtedy i tylko wtedy, gdy } d \text{ dzieli } F_m \text{ i } F_n. \quad (9)$$

Jeśli więc r jest resztą z dzielenia m przez n (innymi słowy, jeśli $r = m \bmod n$), to wspólne dzielniki $\{F_m, F_n\}$ są wspólnymi dzielnikami $\{F_r, F_n\}$. Modyfikacje m i n zachodzące w algorytmie 1.1E nie zmieniają zbioru dzielników $\{F_m, F_n\}$. Wreszcie gdy $r = 0$, wspólne dzielniki to po prostu dzielni $F_0 = 0$ i $F_{\gcd(m,n)}$. ■

Większość ważnych wyników dotyczących liczb Fibonacciego można wyrowadzić, znając przedstawienie F_n za pomocą liczby ϕ . Metoda, którą posłużymy się w poniższym wywodzie, jest niezwykle istotna, a Czytelnicy z zacięciem matematycznym powinni przyjrzeć się jej ze szczególną uwagą. Wnikliwą analizą samej metody zajmiemy się w następnym rozdziale.

Zacznijmy od napisania nieskończonego szeregu

$$\begin{aligned} G(z) &= F_0 + F_1 z + F_2 z^2 + F_3 z^3 + F_4 z^4 + \dots \\ &= z + z^2 + 2z^3 + 3z^4 + \dots \end{aligned} \quad (10)$$

Nie ma żadnego powodu, dla którego *a priori* ta nieskończona suma miałaby istnieć lub dla którego funkcja $G(z)$ miałaby w ogóle być interesująca. Ale bądźmy optymistami i spróbujmy zbadać, co dałoby się powiedzieć o funkcji $G(z)$, gdyby istniała. Nasze podejście ma istotną zaletę polegającą na tym, że $G(z)$ jest pojedynczą wielkością, reprezentującą jednocześnie *cały* ciąg Fibonacciego. Jeśli odkryjemy, że $G(z)$ jest funkcją „znana”, to uda się wyznaczyć współczynniki jej rozwinięcia w szereg. Funkcję $G(z)$ nazywamy *funkcją tworzącą* ciągu $\langle F_n \rangle$.

Przekształćmy funkcję $G(z)$ w następujący sposób:

$$\begin{aligned} zG(z) &= F_0 z + F_1 z^2 + F_2 z^3 + F_3 z^4 + \dots, \\ z^2 G(z) &= \qquad F_0 z^2 + F_1 z^3 + F_2 z^4 + \dots; \end{aligned}$$

a po odjęciu otrzymujemy

$$\begin{aligned} (1 - z - z^2)G(z) &= F_0 + (F_1 - F_0)z + (F_2 - F_1 - F_0)z^2 \\ &\quad + (F_3 - F_2 - F_1)z^3 + (F_4 - F_3 - F_2)z^4 + \dots \end{aligned}$$

Na mocy definicji ciągu F_n wszystkie wyrazy poza ostatnim znikają, zatem całe wyrażenie równa się z . Stąd, jeżeli $G(z)$ istnieje, to

$$G(z) = z/(1 - z - z^2). \quad (11)$$

W istocie ta funkcja *może* zostać rozwinięta w nieskończony szereg względem z (szereg Taylora). Posuwając się wstecz, odkryjemy, że współczynniki takiego rozwinięcia funkcji (11) muszą być liczbami Fibonacciego.

Możemy teraz przekształcić $G(z)$ i dowiedzieć się czegoś nowego o ciągu Fibonacciego. Mianownik $1 - z - z^2$ jest równaniem kwadratowym o dwóch pierwiastkach $\frac{1}{2}(-1 \pm \sqrt{5})$. W wyniku wykonania kilku obliczeń przekonamy się, że $G(z)$ można rozwinać, korzystając z metody ułamków prostych:

$$G(z) = \frac{1}{\sqrt{5}} \left(\frac{1}{1 - \phi z} - \frac{1}{1 - \hat{\phi} z} \right), \quad (12)$$

gdzie

$$\hat{\phi} = 1 - \phi = \frac{1}{2}(1 - \sqrt{5}). \quad (13)$$

Wielkość $1/(1 - \phi z)$ jest sumą nieskończonego szeregu geometrycznego $1 + \phi z + \phi^2 z^2 + \dots$, mamy więc

$$G(z) = \frac{1}{\sqrt{5}} (1 + \phi z + \phi^2 z^2 + \dots - 1 - \hat{\phi} z - \hat{\phi}^2 z^2 - \dots).$$

Współczynnik przy z^n musi być równy F_n , zatem

$$F_n = \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n). \quad (14)$$

Jest to ważna postać zamknięta wzoru na liczby Fibonacciego, odkryta na początku XVIII wieku. (Zobacz D. Bernoulli, *Comment. Acad. Sci. Petrop.* **3** (1728), 85–100, §7, oraz A. de Moivre, *Philos. Trans.* **32** (1722), 162–178, gdzie pokazano ogólną metodę rozwiązywania rekurencyjnych równań liniowych, z której skorzystaliśmy, wyprowadzając wzór (14)).

Mogliśmy oczywiście napisać równość (14) i udowodnić ją przez indukcję. Chcieliśmy jednak pokazać, jak za pomocą funkcji tworzących udało się ją *odkryć*. Metoda funkcji tworzących to niezmiernie wartościowa technika, mająca zastosowanie do całej gamy problemów.

Za pomocą wzoru (14) można udowodnić wiele faktów. Przede wszystkim zauważmy, że $\hat{\phi}$ jest liczbą *ujemną* ($-0.61803\dots$), której wartość bezwzględna jest mniejsza od jedności, zatem gdy n rośnie, $\hat{\phi}^n$ staje się bardzo małe. W istocie wielkość $\hat{\phi}^n/\sqrt{5}$ jest zawsze na tyle mała, że zachodzi

$$F_n = \phi^n/\sqrt{5} \text{ zaokrąglone do najbliższej liczby całkowitej}. \quad (15)$$

Inne wyniki można uzyskać, analizując $G(z)$ bezpośrednio; na przykład,

$$G(z)^2 = \frac{1}{5} \left(\frac{1}{(1 - \phi z)^2} + \frac{1}{(1 - \hat{\phi} z)^2} - \frac{2}{1 - z - z^2} \right), \quad (16)$$

a współczynniki w $G(z)^2$ równają się $\sum_{k=0}^n F_k F_{n-k}$. Wnioskujemy stąd, że

$$\begin{aligned}\sum_{k=0}^n F_k F_{n-k} &= \frac{1}{5} ((n+1)(\phi^n + \bar{\phi}^n) - 2F_{n+1}) \\ &= \frac{1}{5} ((n+1)(F_n + 2F_{n-1}) - 2F_{n+1}) \\ &= \frac{1}{5} (n-1)F_n + \frac{2}{5} n F_{n-1}.\end{aligned}\tag{17}$$

(W drugim kroku wyprowadzenia korzystamy z ćwiczenia 11).

ĆWICZENIA

1. [10] Podaj odpowiedź na pierwotny problem postawiony przez Leonarda Fibonacciego: ile będzie par królików po roku?
 - ▶ 2. [20] Korzystając z (15), oblicz przybliżoną wartość F_{1000} . (Posłuż się logarytmami zamieszczonymi w dodatku A).
 3. [25] Napisz program, który oblicza i wypisuje liczby F_1 do F_{1000} w zapisie dziesiętnym. (Z poprzedniego ćwiczenia wiadomo, jak dużych liczb należy się spodziewać).
 - ▶ 4. [14] Znajdź wszystkie n , dla których $F_n = n$.
 5. [20] Znajdź wszystkie n , dla których $F_n = n^2$.
 6. [HM10] Udowodnij równanie (5).
 - ▶ 7. [15] Jeśli n jest liczbą złożoną, to F_n też jest liczbą złożoną (z jednym wyjątkiem). Udowodnij to twierdzenie i znajdź wyjątek.
 8. [15] W wielu przypadkach wygodnie jest zdefiniować F_n dla n ujemnych, zakładając, że $F_{n+2} = F_{n+1} + F_n$ dla wszystkich n całkowitych. Zbadaj taki wariant definicji. Ile wynosi F_{-1} , a ile F_{-2} ? Czy F_{-n} da się prosto wyrazić za pomocą F_n ?
 9. [M20] Opierając się na konwencji przyjętej w ćwiczeniu 8, sprawdź, czy równości (4), (6), (14) i (15) będą zachodzić, gdy zezwolimy, żeby indeksy przyjmowały dowolne wartości całkowite.
 10. [15] Czy liczba $\phi^n/\sqrt{5}$ jest większa, czy mniejsza niż F_n ?
 11. [M20] Pokaż, że $\phi^n = F_n\phi + F_{n-1}$ i $\bar{\phi}^n = F_n\bar{\phi} + F_{n-1}$, dla wszystkich całkowitych n .
 - ▶ 12. [M26] Ciągi Fibonacciego „drugiego rzędu” definiujemy za pomocą równości
- $$\mathcal{F}_0 = 0, \quad \mathcal{F}_1 = 1, \quad \mathcal{F}_{n+2} = \mathcal{F}_{n+1} + \mathcal{F}_n + F_n.$$
- Wyraź \mathcal{F}_n za pomocą F_n i F_{n+1} . [Wskazówka: Posłuż się funkcjami tworzącymi].
- ▶ 13. [M22] Dla parametrów r , s i c za pomocą liczb Fibonacciego zapisz wzór na n -ty wyraz ciągu:
 - $a_0 = r$, $a_1 = s$; $a_{n+2} = a_{n+1} + a_n$, dla $n \geq 0$.
 - $b_0 = 0$, $b_1 = 1$; $b_{n+2} = b_{n+1} + b_n + c$, dla $n \geq 0$.
 14. [M28] Niech m będzie ustaloną dodatnią liczbą całkowitą. Znajdź a_n , wiedząc, że
- $$a_0 = 0, \quad a_1 = 1; \quad a_{n+2} = a_{n+1} + a_n + \binom{n}{m} \quad \text{dla } n \geq 0.$$
15. [M22] Niech $f(n)$ i $g(n)$ będą dowolnymi funkcjami i niech dla $n \geq 0$
- $$\begin{aligned}a_0 &= 0, \quad a_1 = 1, \quad a_{n+2} = a_{n+1} + a_n + f(n); \\ b_0 &= 0, \quad b_1 = 1, \quad b_{n+2} = b_{n+1} + b_n + g(n); \\ c_0 &= 0, \quad c_1 = 1, \quad c_{n+2} = c_{n+1} + c_n + xf(n) + yg(n).\end{aligned}$$
- Zapisz c_n za pomocą x , y , a_n , b_n i F_n .

- 16. [M20] Liczby Fibonacciego występują niejawnie w trójkącie Pascala, jeżeli przyjrzeć mu się pod odpowiednim kątem. Pokaż, że poniższa suma współczynników dwumianowych jest liczbą Fibonacciego:

$$\sum_{k=0}^n \binom{n-k}{k}.$$

17. [M24] Przyjmując konwencje ustalone w ćwiczeniu 8, udowodnij poniższe uogólnienie (4): $F_{n+k}F_{m-k} - F_nF_m = (-1)^n F_{m-n-k}F_k$.
18. [20] Czy $F_n^2 + F_{n+1}^2$ jest zawsze liczbą Fibonacciego?
- 19. [M27] Ile wynosi $\cos 36^\circ$?
20. [M16] Wyraź $\sum_{k=0}^n F_k$ za pomocą liczb Fibonacciego.
21. [M25] Ile wynosi $\sum_{k=0}^n F_k x^k$?
- 22. [M20] Pokaż, że suma $\sum_k \binom{n}{k} F_{m+k}$ jest liczbą Fibonacciego.
23. [M23] Uogólnij poprzednie ćwiczenie, pokazując, że $\sum_k \binom{n}{k} F_t^k F_{t-1}^{n-k} F_{m+k}$ jest zawsze liczbą Fibonacciego.
24. [HM20] Oblicz wyznacznik poniższej macierzy $n \times n$:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 1 & -1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 1 \end{pmatrix}.$$

25. [M21] Pokaż, że
- $$2^n F_n = 2 \sum_{k \text{ nieparz.}} \binom{n}{k} 5^{(k-1)/2}.$$
- 26. [M20] Skorzystaj z poprzedniego twierdzenia, aby udowodnić, że $F_p \equiv 5^{(p-1)/2}$ (modulo p), jeśli p jest nieparzystą liczbą pierwszą.
27. [M20] Korzystając z poprzedniego ćwiczenia, pokaż, że jeśli p jest liczbą pierwszą różną od 5, to albo liczba F_{p-1} , albo F_{p+1} (ale nie obie) są wielokrotnościami p .
28. [M21] Ile wynosi $F_{n+1} - \phi F_n$?
- 29. [M23] (Współczynniki fibomianowe) Édouard Lucas zdefiniował wielkości

$$\binom{n}{k}_{\mathcal{F}} = \frac{F_n F_{n-1} \dots F_{n-k+1}}{F_k F_{k-1} \dots F_1} = \prod_{j=1}^k \left(\frac{F_{n-k+j}}{F_j} \right)$$

w sposób podobny do współczynników dwumianowych. (a) Sporządź tabelę $\binom{n}{k}_{\mathcal{F}}$ dla $0 \leq k \leq n \leq 6$. (b) Pokaż, że $\binom{n}{k}_{\mathcal{F}}$ jest zawsze całkowite, bo

$$\binom{n}{k}_{\mathcal{F}} = F_{k-1} \binom{n-1}{k}_{\mathcal{F}} + F_{n-k+1} \binom{n-1}{k-1}_{\mathcal{F}}.$$

- 30. [M38] (D. Jarden, T. Motzkin) Ciąg m -tych potęg liczb Fibonacciego spełnia związek rekurencyjny, w którym każdy wyraz zależy od $m+1$ poprzednich wyrazów.

Pokaż, że

$$\sum_k \binom{m}{k}_{\mathcal{F}} (-1)^{\lceil (m-k)/2 \rceil} F_{n+k}^{m-1} = 0, \quad \text{dla } m > 0.$$

Gdy na przykład $m = 3$, otrzymujemy tożsamość $F_n^2 - 2F_{n+1}^2 - 2F_{n+2}^2 + F_{n+3}^2 = 0$.

31. [M20] Pokaż, że $F_{2n}\phi \bmod 1 = 1 - \phi^{-2n}$ i $F_{2n+1}\phi \bmod 1 = \phi^{-2n-1}$.

32. [M24] Reszta z dzielenia liczby Fibonacciego przez liczbę Fibonacciego jest \pm liczbą Fibonacciego. Pokaż, że (modulo F_n)

$$F_{mn+r} \equiv \begin{cases} F_r, & \text{gdy } m \bmod 4 = 0; \\ (-1)^{r+1} F_{n-r}, & \text{gdy } m \bmod 4 = 1; \\ (-1)^n F_r, & \text{gdy } m \bmod 4 = 2; \\ (-1)^{r+1+n} F_{n-r}, & \text{gdy } m \bmod 4 = 3. \end{cases}$$

33. [HM24] Zakładając $z = \pi/2 + i \ln \phi$, pokaż, że $\sin nz / \sin z = i^{1-n} F_n$.

► **34.** [M24] (System liczbowy Fibonacciego) Niech $k \gg m$ oznacza, że $k \geq m+2$. Pokaż, że każda dodatnia liczba całkowita n ma jednoznaczną reprezentację $n = F_{k_1} + F_{k_2} + \dots + F_{k_r}$, gdy $k_1 \gg k_2 \gg \dots \gg k_r \gg 0$.

35. [M24] (System liczbowy ϕ) Rozważmy liczby rzeczywiste zapisywane za pomocą dwóch cyfr: 0 i 1 przy podstawie ϕ , na przykład $(100.1)_\phi = \phi^2 + \phi^{-1}$. Pokaż, że jest nieskończenie wiele możliwości przedstawienia liczby 1, na przykład $1 = (0.11)_\phi = (0.01111\dots)_\phi$. Ale jeżeli będziemy wymagać, żeby w reprezentacji nie występowały dwie jedynki z rzędu oraz żeby ostatnie cyfry reprezentacji nie były nieskończonym ciągiem $01010101\dots$, to każda liczba nieujemna będzie miała reprezentację jednoznaczną. Jak wyglądają reprezentacje liczb całkowitych?

► **36.** [M32] (Słowa Fibonacciego) Niech $S_1 = „a”$, $S_2 = „b”$ i $S_{n+2} = S_{n+1}S_n$, $n > 0$. Inaczej mówiąc, S_{n+2} powstaje przez dopisanie S_n na prawo od S_{n+1} . Mamy $S_3 = „ba”$, $S_4 = „bab”$, $S_5 = „babba”$ itd. Widać, że S_n ma F_n liter. Zbadaj własności S_n . (Gdzie pojawiają się pary jednakowych sąsiadujących liter? Czy umiesz wyznaczyć k -tą literę słowa S_n ? Jaka jest gęstość liter b ?)

► **37.** [M35] (R. E. Gaskell, M. J. Whinihan) Oto gra dla dwóch graczy. Na stole leży n bierek. Pierwszy gracz zdejmuje z planszy dowolną liczbę bierek, ale tak, żeby coś zostało. Od tego momentu gracze zdejmują bierki na przemian, w każdym ruchu usuwając z planszy przynajmniej jedną bierkę, ale *nie więcej niż dwa razy tyle, co przeciwnik w ostatnim ruchu*. Gracz, który zdejmie ostatnią bierkę, wygrywa. (Na przykład: niech $n = 11$, gracz A usuwa trzy bierki. Gracz B może usunąć do sześciu bierek, ale bierze jedną, zostaje siedem. Gracz A może usunąć jedną lub dwie bierki, bierze dwie. Gracz B może usunąć do czterech bierek, ale bierze jedną, zostają cztery. Gracz A bierze jedną bierkę, gracz B musi wziąć przynajmniej jedną bierkę, zatem gracz A wygrywa w następnym ruchu).

Jaki jest optymalny otwierający ruch pierwszego gracza, gdy na planszy znajduje się 1000 bierek?

38. [35] Napisz program, który gra w grę opisaną w poprzednim ćwiczeniu i robi to optymalnie.

39. [M24] Znajdź postać zamkniętą a_n , wiedząc, że $a_0 = 0$, $a_1 = 1$ i $a_{n+2} = a_{n+1} + 6a_n$ dla $n \geq 0$.

40. [M25] Rozwiąż równanie rekurencyjne

$$f(1) = 0; \quad f(n) = \min_{0 < k < n} \max(1 + f(k), 2 + f(n - k)), \quad \text{dla } n > 1.$$

► 41. [M25] (Jurij Matijasiewicz, 1990) Niech $f(x) = \lfloor x + \phi^{-1} \rfloor$. Udowodnij, że jeśli $n = F_{k_1} + \dots + F_{k_r}$ jest reprezentacją n w systemie pozycyjnym Fibonacciego z ćwiczenia 34, to $F_{k_1+1} + \dots + F_{k_r+1} = f(\phi n)$. Znajdź podobny wzór na $F_{k_1-1} + \dots + F_{k_r-1}$.

42. [M26] (D. A. Klarner) Pokaż, że jeżeli m i n są nieujemnymi liczbami całkowitymi, to istnieje dokładnie jeden ciąg indeksów $k_1 \gg k_2 \gg \dots \gg k_r$, taki że

$$m = F_{k_1} + F_{k_2} + \dots + F_{k_r}, \quad n = F_{k_1+1} + F_{k_2+1} + \dots + F_{k_r+1}.$$

(indeksy k mogą być ujemne, a r może równać się zero).

1.2.9. Funkcje tworzące

Chcąc dowiedzieć się czegoś o ciągu $\langle a_n \rangle = a_0, a_1, a_2, \dots$, możemy utworzyć sumę nieskończoną za pomocą „parametru” z ,

$$G(z) = a_0 + a_1 z + a_2 z^2 + \dots = \sum_{n \geq 0} a_n z^n \quad (1)$$

i próbować badać funkcję G . Taka funkcja jest pojedynczym obiektem reprezentującym cały ciąg, a to jest istotna zaleta, gdy ciąg $\langle a_n \rangle$ jest zdefiniowany indukcyjnie (tj. gdy a_n jest zdefiniowane za pomocą a_0, a_1, \dots, a_{n-1}). Co więcej, przy założeniu że suma nieskończona (1) istnieje dla pewnej niezerowej wartości z , za pomocą rachunku różniczkowego możemy wyciągnąć z $G(z)$ konkretne wartości a_0, a_1, \dots .

Funkcję $G(z)$ nazywamy *funkcją tworzącą* ciągu a_0, a_1, a_2, \dots . Zastosowanie funkcji tworzących przybliża nam szeroki wachlarz nowych metod i znacznie zwiększa możliwości rozwiązywania problemów. Jak wspomnieliśmy w poprzednim rozdziale, A. de Moivre wprowadził funkcje tworzące jako ogólną metodę rozwiązywania liniowych równań rekurencyjnych. Teoria de Moivre'a została rozszerzona na trochę bardziej skomplikowane równania rekurencyjne przez Jamesa Stirlinga, który pokazał, w jaki sposób do zestawu operacji arytmetycznych na funkcjach tworzących dodać różniczkowanie i całkowanie. [*Methodus Differentialis* (London: 1730), twierdzenie 15]. Kilka lat później L. Euler zaczął wykorzystywać funkcje tworzące na kilka nowych sposobów, na przykład w swoim artykule o podziałach [*Commentarii Acad. Sci. Pet.* **13** (1741), 64–93; *Novi Comment. Acad. Sci. Pet.* **3** (1750), 125–169]. Metody te zostały rozwinięte przez Pierre'a S. Laplace'a w klasycznej pracy *Théorie Analytique des Probabilités* (Paris: 1812).

Ważne jest pytanie o zbieżność nieskończonej sumy (1). W dowolnej książce dotyczącej teorii szeregów znajdziemy dowody następujących faktów:

- Jeżeli szereg jest zbieżny dla pewnej wartości $z = z_0$, to jest zbieżny także dla wszystkich takich wartości z , że $|z| < |z_0|$.
- Szereg jest zbieżny dla pewnego $z \neq 0$ wtedy i tylko wtedy, gdy ciąg $\langle \sqrt[n]{|a_n|} \rangle$ jest ograniczony. (Jeżeli ten warunek nie jest spełniony, to możemy uzyskać szereg zbieżny dla ciągu $\langle a_n/n! \rangle$ lub dla innych podobnych).

Jednak przekształcając funkcje tworzące, nie zawsze warto martwić się zbieżnością szeregu, ponieważ w istocie badamy jedynie możliwe podejścia do rozwią-

zania problemu. Jeśli w jakikolwiek sposób odkryjemy rozwiązanie, to możemy je później niezależnie potwierdzić. W poprzednim rozdziale skorzystaliśmy na przykład z funkcji tworzących do wymyślenia wzoru (14); mając równanie, można łatwo udowodnić je przez indukcję i nie trzeba nawet wspominać o tym, że odkryliśmy je za pomocą funkcji tworzących. Nie koniec na tym. Możemy pokazać, że większość wykorzystywanych przez nas przekształceń funkcji tworzących (jeśli nie wszystkie) można precyzyjnie uzasadnić bez odwoływanego się do zbieżności szeregow. Zobacz na przykład E. T. Bell, *Trans. Amer. Math. Soc.* **25** (1923), 135–154; Ivan Niven, *AMM* **76** (1969), 871–889; Peter Henrici, *Applied and Computational Complex Analysis* **1** (Wiley, 1974), rozdział 1.

Skupmy się teraz na podstawowych metodach związanych z wykorzystaniem funkcji tworzących.

A. Dodawanie. Jeśli $G(z)$ jest funkcją tworzącą ciągu $\langle a_n \rangle = a_0, a_1, \dots$ oraz $H(z)$ jest funkcją tworzącą ciągu $\langle b_n \rangle = b_0, b_1, \dots$, to $\alpha G(z) + \beta H(z)$ jest funkcją tworzącą ciągu $\langle \alpha a_n + \beta b_n \rangle = \alpha a_0 + \beta b_0, \alpha a_1 + \beta b_1, \dots$:

$$\alpha \sum_{n \geq 0} a_n z^n + \beta \sum_{n \geq 0} b_n z^n = \sum_{n \geq 0} (\alpha a_n + \beta b_n) z^n. \quad (2)$$

B. Przesunięcie. Jeśli $G(z)$ jest funkcją tworzącą ciągu $\langle a_n \rangle = a_0, a_1, \dots$, to $z^m G(z)$ jest funkcją tworzącą ciągu $\langle a_{n-m} \rangle = 0, \dots, 0, a_0, a_1, \dots$:

$$z^m \sum_{n \geq 0} a_n z^n = \sum_{n \geq m} a_{n-m} z^n. \quad (3)$$

Sumowanie w prawej sumie można rozszerzyć na wszystkie $n \geq 0$, jeśli przyjmujemy $a_n = 0$ dla ujemnych n .

Podobnie $(G(z) - a_0 - a_1 z - \dots - a_{m-1} z^{m-1})/z^m$ jest funkcją tworzącą ciągu $\langle a_{n+m} \rangle = a_m, a_{m+1}, \dots$:

$$z^{-m} \sum_{n \geq m} a_n z^n = \sum_{n \geq 0} a_{n+m} z^n. \quad (4)$$

Przy rozwiązywaniu problemu Fibonacciego z poprzedniego rozdziału posłużyliśmy się operacjami A i B: $G(z)$ było funkcją tworzącą ciągu $\langle F_n \rangle$, $zG(z)$ ciągu $\langle F_{n-1} \rangle$, $z^2G(z)$ ciągu $\langle F_{n-2} \rangle$, a $(1 - z - z^2)G(z)$ ciągu $\langle F_n - F_{n-1} - F_{n-2} \rangle$. Stwierdziliśmy zatem, że skoro $F_n - F_{n-1} - F_{n-2}$ jest zerem dla $n \geq 2$, to $(1 - z - z^2)G(z)$ jest wielomianem. Podobnie, dla dowolnego ciągu liniowo-rekurencyjnego, tj. takiego że $a_n = c_1 a_{n-1} + \dots + c_m a_{n-m}$, funkcja tworząca jest wielomianem podzielonym przez $(1 - c_1 z - \dots - c_m z^m)$.

Rozważmy najprostszy przykład: jeśli $G(z)$ jest funkcją tworzącą ciągu stałego 1, 1, 1, ..., to $zG(z)$ jest funkcją tworzącą ciągu 0, 1, 1, ..., a zatem $(1 - z)G(z) = 1$. Otrzymujemy stąd prosty, ale niezmiernie istotny wzór:

$$\frac{1}{1 - z} = 1 + z + z^2 + \dots \quad (5)$$

C. Mnożenie. Jeśli $G(z)$ jest funkcją tworzącą ciągu a_0, a_1, \dots oraz $H(z)$ jest funkcją tworzącą ciągu b_0, b_1, \dots , to

$$\begin{aligned} G(z)H(z) &= (a_0 + a_1z + a_2z^2 + \dots)(b_0 + b_1z + b_2z^2 + \dots) \\ &= (a_0b_0) + (a_0b_1 + a_1b_0)z + (a_0b_2 + a_1b_1 + a_2b_0)z^2 + \dots; \end{aligned}$$

zatem $G(z)H(z)$ jest funkcją tworzącą ciągu c_0, c_1, \dots , gdzie

$$c_n = \sum_{k=0}^n a_k b_{n-k}. \quad (6)$$

Równanie (3) jest tego bardzo szczególnym przypadkiem. Inny ważny przypadek szczególny pojawia się, gdy b_n jest ciągiem stałym równym jeden:

$$\frac{1}{1-z}G(z) = a_0 + (a_0 + a_1)z + (a_0 + a_1 + a_2)z^2 + \dots. \quad (7)$$

Otrzymujemy funkcję tworzącą ciągu sum częściowych.

Wzór na iloczyn trzech funkcji wynika z (6); $F(z)G(z)H(z)$ jest funkcją tworzącą ciągu d_0, d_1, d_2, \dots , gdzie

$$d_n = \sum_{\substack{i,j,k \geq 0 \\ i+j+k=n}} a_i b_j c_k. \quad (8)$$

Ogólna zasada obowiązująca dla iloczynów dowolnej liczby funkcji (jeżeli to oczywiście ma sens) wygląda tak:

$$\prod_{j \geq 0} \sum_{k \geq 0} a_{jk} z^k = \sum_{n \geq 0} z^n \sum_{\substack{k_0, k_1, \dots \geq 0 \\ k_0 + k_1 + \dots = n}} a_{0k_0} a_{1k_1} \dots. \quad (9)$$

Gdy związek rekurencyjny opisujący pewien ciąg zawiera współczynniki dwumianowe, często interesuje nas znalezienie funkcji tworzącej ciągu c_0, c_1, \dots , takiego że

$$c_n = \sum_k \binom{n}{k} a_k b_{n-k}. \quad (10)$$

W takim przypadku lepiej zazwyczaj skorzystać z funkcji tworzących ciągów $\langle a_n/n! \rangle, \langle b_n/n! \rangle, \langle c_n/n! \rangle$, ponieważ

$$\left(\frac{a_0}{0!} + \frac{a_1}{1!}z + \frac{a_2}{2!}z^2 + \dots \right) \left(\frac{b_0}{0!} + \frac{b_1}{1!}z + \frac{b_2}{2!}z^2 + \dots \right) = \left(\frac{c_0}{0!} + \frac{c_1}{1!}z + \frac{c_2}{2!}z^2 + \dots \right), \quad (11)$$

gdzie c_n jest określone jak w (10).

D. Zamiana z. Widać, że $G(cz)$ jest funkcją tworzącą ciągu a_0, ca_1, c^2a_2, \dots . W szczególności funkcja tworząca ciągu $1, c, c^2, c^3, \dots$ jest $1/(1 - cz)$.

Istnieje znana sztuczka pozwalająca wyjąć z ciągu co drugi wyraz:

$$\begin{aligned} \frac{1}{2}(G(z) + G(-z)) &= a_0 + a_2z^2 + a_4z^4 + \dots, \\ \frac{1}{2}(G(z) - G(-z)) &= a_1z + a_3z^3 + a_5z^5 + \dots. \end{aligned} \quad (12)$$

Korzystając z zespolonych pierwiastków z jedności, możemy rozszerzyć powyższy pomysł i wyciągnąć co m -ty wyraz. Niech $\omega = e^{2\pi i/m} = \cos(2\pi/m) + i \sin(2\pi/m)$. Mamy wówczas

$$\sum_{n \bmod m=r} a_n z^n = \frac{1}{m} \sum_{0 \leq k < m} \omega^{-kr} G(\omega^k z), \quad 0 \leq r < m. \quad (13)$$

(Zobacz ćwiczenie 14). Na przykład, jeżeli $m = 3$ i $r = 1$, to $\omega = -\frac{1}{2} + \frac{\sqrt{3}}{2}i$, co jest zespolonym pierwiastkiem sześciennym z jedności; stąd otrzymujemy

$$a_1 z + a_4 z^4 + a_7 z^7 + \dots = \frac{1}{3} (G(z) + \omega^{-1} G(\omega z) + \omega^{-2} G(\omega^2 z)).$$

E. Różniczkowanie i całkowanie. Metody znane z analizy matematycznej to kolejne przydatne przekształcenia funkcji tworzących. Niech $G(z)$ będzie taka, jak we wzorze (1), wówczas jej pochodna to

$$G'(z) = a_1 + 2a_2 z + 3a_3 z^2 + \dots = \sum_{k \geq 0} (k+1) a_{k+1} z^k. \quad (14)$$

Funkcją tworzącą ciągu $\langle na_n \rangle$ jest $zG'(z)$. Manipulując funkcjami tworzącymi, możemy wprowadzić do wyrazów ciągu czynnik będący wielomianem zmiennej n .

Proces można odwrócić – stosując całkowanie, otrzymujemy kolejne pozyteczne przekształcenie:

$$\int_0^z G(t) dt = a_0 z + \frac{1}{2} a_1 z^2 + \frac{1}{3} a_2 z^3 + \dots = \sum_{k \geq 1} \frac{1}{k} a_{k-1} z^k. \quad (15)$$

Szczególnymi przypadkami powyższych reguł są pochodna i całka szeregu (5):

$$\frac{1}{(1-z)^2} = 1 + 2z + 3z^2 + \dots = \sum_{k \geq 0} (k+1) z^k. \quad (16)$$

$$\ln \frac{1}{1-z} = z + \frac{1}{2} z^2 + \frac{1}{3} z^3 + \dots = \sum_{k \geq 1} \frac{1}{k} z^k. \quad (17)$$

Możemy do ostatniego z tych wzorów zastosować równanie (7), by otrzymać funkcję tworzącą ciągu liczb harmonicznych:

$$\frac{1}{1-z} \ln \frac{1}{1-z} = z + \frac{3}{2} z^2 + \frac{11}{6} z^3 + \dots = \sum_{k \geq 0} H_k z^k. \quad (18)$$

F. Znane funkcje tworzące. Wszędzie tam, gdzie umiemy powiedzieć, jak wygląda rozwinięcie funkcji w szereg potęgowy, odkrywamy w istocie funkcję tworzącą pewnego ciągu. Niektóre szczególne funkcje mogą być pożyteczne w połączeniu z opisanymi powyżej przekształceniami. Najważniejsze rozwinięcia w szeregi potęgowe podajemy poniżej.

i) *Twierdzenie dwumianowe.*

$$(1+z)^r = 1 + rz + \frac{r(r-1)}{2}z^2 + \dots = \sum_{k \geq 0} \binom{r}{k} z^k. \quad (19)$$

Gdy r jest ujemną liczbą całkowitą, otrzymujemy przypadek szczególny, opisany wcześniej w (5) i (16):

$$\frac{1}{(1-z)^{n+1}} = \sum_{k \geq 0} \binom{-n-1}{k} (-z)^k = \sum_{k \geq 0} \binom{n+k}{n} z^k. \quad (20)$$

Dysponujemy także uogólnieniem, udowodnionym w ćwiczeniu 1.2.6–25:

$$x^r = 1 + rz + \frac{r(r-2t-1)}{2}z^2 + \dots = \sum_{k \geq 0} \binom{r-kt}{k} \frac{r}{r-kt} z^k, \quad (21)$$

gdzie x jest ciągłą funkcją z będącą rozwiązaniem równania $x^{t+1} = x^t + z$, gdzie $x = 1$ dla $z = 0$.

ii) *Szereg wykładniczy.*

$$\exp z = e^z = 1 + z + \frac{1}{2!}z^2 + \dots = \sum_{k \geq 0} \frac{1}{k!} z^k. \quad (22)$$

W ogólności zachodzi następująca równość zawierająca liczby Stirlinga:

$$(e^z - 1)^n = z^n + \frac{1}{n+1} \left\{ \begin{matrix} n+1 \\ n \end{matrix} \right\} z^{n+1} + \dots = n! \sum_k \left\{ \begin{matrix} k \\ n \end{matrix} \right\} z^k / k!. \quad (23)$$

iii) *Szereg logarytmiczny* (zobacz (17) i (18)).

$$\ln(1+z) = z - \frac{1}{2}z^2 + \frac{1}{3}z^3 - \dots = \sum_{k \geq 1} \frac{(-1)^{k+1}}{k} z^k, \quad (24)$$

$$\frac{1}{(1-z)^{m+1}} \ln\left(\frac{1}{1-z}\right) = \sum_{k \geq 1} (H_{m+k} - H_m) \binom{m+k}{k} z^k. \quad (25)$$

Stosując liczby Stirlinga, podobnie jak w (23), możemy sformułować bardziej ogólny wzór:

$$\left(\ln \frac{1}{1-z} \right)^n = z^n + \frac{1}{n+1} \left[\begin{matrix} n+1 \\ n \end{matrix} \right] z^{n+1} + \dots = n! \sum_k \left[\begin{matrix} k \\ n \end{matrix} \right] z^k / k!. \quad (26)$$

Dalsze uogólnienia, także wiele sum liczb zespolonych, można znaleźć w artykułach: D. A. Zave, *Inf. Proc. Letters* **5** (1976), 75–77; J. Spieß, *Math. Comp.* **55** (1990), 839–863.

iv) *Różności.*

$$z(z+1) \dots (z+n-1) = \sum_k \left[\begin{matrix} n \\ k \end{matrix} \right] z^k, \quad (27)$$

$$\frac{z^n}{(1-z)(1-2z) \dots (1-nz)} = \sum_k \left\{ \begin{matrix} k \\ n \end{matrix} \right\} z^k, \quad (28)$$

$$\frac{z}{e^z - 1} = 1 - \frac{1}{2}z + \frac{1}{12}z^2 + \dots = \sum_{k \geq 0} \frac{B_k z^k}{k!}. \quad (29)$$

Współczynniki B_k pojawiające się w ostatnim wzorze to *liczby Bernoulliego*; przyjrzymy się im bliżej w podpunkcie 1.2.11.2. Liczby Bernoulliego są podane w tabeli 3 w dodatku A.

Kolejna tożsamość, podobna do (21), zostanie udowodniona w ćwiczeniu 2.3.4.4–29:

$$x^r = 1 + rz + \frac{r(r+2t)}{2}z^2 + \dots = \sum_{k \geq 0} \frac{r(r+kt)^{k-1}}{k!} z^k, \quad (30)$$

gdy x jest ciągłą funkcją z będącą rozwiązaniem równania $x = e^{zx^t}$, gdzie $x = 1$ dla $z = 0$. Istotne uogólnienia wzorów (21) i (30) są omówione w ćwiczeniu 4.7–22.

G. Wyciąganie współczynnika. Wygodnie jest używać oznaczenia

$$[z^n] G(z) \quad (31)$$

na współczynnik stojący przy z^n w rozwinięciu $G(z)$. Na przykład, jeśli $G(z)$ jest funkcją tworzącą we wzorze (1), to $[z^n] G(z) = a_n$ i $[z^n] G(z)/(1-z) = \sum_{k=0}^n a_k$. Jednym z podstawowych wyników teorii zmiennej zespolonej jest wzór A. L. Cauchy'ego [*Exercices de Math. 1* (1826), 95–113 = *Oeuvres* (2) 6, 124–145, wzór (11)], dzięki któremu możemy wydobyć dowolny współczynnik za pomocą całki po krzywej zamkniętej:

$$[z^n] G(z) = \frac{1}{2\pi i} \oint_{|z|=r} \frac{G(z) dz}{z^{n+1}}, \quad (32)$$

jeżeli szereg $G(z)$ jest zbieżny dla $z = z_0$ i $0 < r < |z_0|$. Podstawowy pomysł polega na tym, że $\oint_{|z|=r} z^m dz$ jest zerem dla wszystkich liczb całkowitych m poza $m = -1$, gdy wartość całki wynosi

$$\int_{-\pi}^{\pi} (re^{i\theta})^{-1} d(re^{i\theta}) = i \int_{-\pi}^{\pi} d\theta = 2\pi i.$$

Równanie (32) jest istotne szczególnie wtedy, gdy interesują nas przybliżone wartości współczynników.

Zamykamy ten rozdział, powracając do problemu, który nie został do końca rozwiązyany w punkcie 1.2.3. We wzorze 1.2.3–(13) oraz w ćwiczeniu 1.2.3–29 zaobserwowaliśmy, że

$$\begin{aligned} \sum_{1 \leq i \leq j \leq n} x_i x_j &= \frac{1}{2} \left(\sum_{k=1}^n x_k \right)^2 + \frac{1}{2} \left(\sum_{k=1}^n x_k^2 \right); \\ \sum_{1 \leq i \leq j \leq k \leq n} x_i x_j x_k &= \frac{1}{6} \left(\sum_{k=1}^n x_k \right)^3 + \frac{1}{2} \left(\sum_{k=1}^n x_k \right) \left(\sum_{k=1}^n x_k^2 \right) + \frac{1}{3} \left(\sum_{k=1}^n x_k^3 \right). \end{aligned}$$

W ogólności założymy, że mamy n liczb x_1, x_2, \dots, x_n i szukamy

$$h_m = \sum_{1 \leq j_1 \leq \dots \leq j_m \leq n} x_{j_1} \dots x_{j_m}. \quad (33)$$

Jeżeli to możliwe, suma powinna być wyrażona za pomocą S_1, S_2, \dots, S_m , gdzie

$$S_j = \sum_{k=1}^n x_k^j, \quad (34)$$

jest sumą j -tych potęg. W bardziej zwięzlym zapisie powyższe wzory wyglądają tak: $h_2 = \frac{1}{2}S_1^2 + \frac{1}{2}S_2$; $h_3 = \frac{1}{6}S_1^3 + \frac{1}{2}S_1S_2 + \frac{1}{3}S_3$.

Możemy spróbować rozwiązać ten problem, pisząc funkcję tworzącą

$$G(z) = 1 + h_1 z + h_2 z^2 + \dots = \sum_{k \geq 0} h_k z^k. \quad (35)$$

Na mocy reguł mnożenia ciągów stwierdzamy, że

$$\begin{aligned} G(z) &= (1 + x_1 z + x_1^2 z^2 + \dots)(1 + x_2 z + x_2^2 z^2 + \dots) \dots (1 + x_n z + x_n^2 z^2 + \dots) \\ &= \frac{1}{(1 - x_1 z)(1 - x_2 z) \dots (1 - x_n z)}. \end{aligned} \quad (36)$$

Zatem $G(z)$ jest odwrotnością wielomianu. Iloczyn zazwyczaj warto zlogarytmować, ze wzoru (17) otrzymamy wówczas:

$$\begin{aligned} \ln G(z) &= \ln \frac{1}{1 - x_1 z} + \dots + \ln \frac{1}{1 - x_n z} \\ &= \left(\sum_{k \geq 1} \frac{x_1^k z^k}{k} \right) + \dots + \left(\sum_{k \geq 1} \frac{x_n^k z^k}{k} \right) = \sum_{k \geq 1} \frac{S_k z^k}{k}. \end{aligned} \quad (37)$$

Dysponujemy reprezentacją $\ln G(z)$ za pomocą sum S . By otrzymać odpowiedź, wystarczy ponownie obliczyć rozwinięcie $G(z)$ w szereg potęgowy, posługując się wzorami (22) i (9):

$$\begin{aligned} G(z) &= e^{\ln G(z)} = \exp \left(\sum_{k \geq 1} \frac{S_k z^k}{k} \right) = \prod_{k \geq 1} e^{S_k z^k / k} \\ &= \left(1 + S_1 z + \frac{S_1^2 z^2}{2!} + \dots \right) \left(1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2^2 \cdot 2!} + \dots \right) \dots \\ &= \sum_{m \geq 0} \left(\sum_{\substack{k_1, k_2, \dots, k_m \geq 0 \\ k_1 + 2k_2 + \dots + mk_m = m}} \frac{S_1^{k_1}}{1^{k_1} k_1!} \frac{S_2^{k_2}}{2^{k_2} k_2!} \dots \frac{S_m^{k_m}}{m^{k_m} k_m!} \right) z^m. \end{aligned} \quad (38)$$

Wartość w nawiasach to h_m . Ta robiąca wrażenie suma nie jest tak strasznie skomplikowana, jeśli się jej bliżej przyjrzeć. Liczba wyrazów dla konkretnej wartości m równa się $p(m)$, czyli liczbie podziałów liczby m (punkt 1.2.1). Na przykład, jeden podział dwunastki to

$$12 = 5 + 2 + 2 + 2 + 1;$$

co odpowiada rozwiązaniu równania $k_1 + 2k_2 + \dots + 12k_{12} = 12$, gdzie k_j jest liczbą wystąpień j w podziale. W naszym przykładzie $k_1 = 1$, $k_2 = 3$, $k_5 = 1$, a wszystkie pozostałe k są równe zero; dostaliśmy zatem wyraz

$$\frac{S_1}{1^1 1!} \frac{S_2^3}{2^3 3!} \frac{S_5}{5^1 1!} = \frac{1}{240} S_1 S_2^3 S_5$$

jako część wyrażenia opisującego h_{12} . Różniczkując wzór (37), nietrudno dojść do równania rekurencyjnego

$$h_n = \frac{1}{n} (S_1 h_{n-1} + S_2 h_{n-2} + \dots + S_n h_0), \quad n \geq 1. \quad (39)$$

Zajmujące wprowadzenie do zastosowań funkcji tworzących zaprezentował G. Pólya: „On picture writing”, *AMM* **63** (1956), 689–697. Jego podejście podchwycono w *CMath*, rozdział 7. Zobacz także: H. S. Wilf, *generatingfunctionology*, wyd. 2 (Academic Press, 1994).

*Funkcja tworząca to sznur do bielizny,
na którym wieszamy ciąg liczbowy.*
— H. S. WILF (1989)

ĆWICZENIA

1. [M12] Jak wygląda funkcja tworząca ciągu $2, 5, 13, 35, \dots = \langle 2^n + 3^n \rangle$?
- ▶ 2. [M13] Udowodnij (11).
3. [HM21] Zróżniczkuj funkcję tworzącą (18) ciągu $\langle H_n \rangle$ i porównaj z funkcją tworzącą ciągu $\langle \sum_{k=0}^n H_k \rangle$. Czy dostrzegasz jakiś związek?
4. [M01] Wyjaśnij, dlaczego wzór (19) jest szczególnym przypadkiem wzoru (21).
5. [M20] Udowodnij (23) przez indukcję względem n .
- ▶ 6. [HM15] Znajdź funkcję tworzącą ciągu

$$\left\langle \sum_{0 < k < n} \frac{1}{k(n-k)} \right\rangle;$$

zróżniczkuj ją i wyraź współczynniki za pomocą liczb harmonicznych.

7. [M15] Uzasadnij wszystkie kroki prowadzące do równania (38).
8. [M23] Znajdź funkcję tworzącą ciągu $p(n)$, gdzie $p(n)$ jest liczbą podziałów n .
9. [M11] Jak przedstawić wartość h_4 za pomocą S_1, S_2, S_3 i S_4 , przy oznaczeniach jak we wzorach (34) i (35)?
- ▶ 10. [M25] Elementarna funkcja symetryczna jest zdefiniowana następująco

$$a_m = \sum_{1 \leq j_1 < \dots < j_m \leq n} x_{j_1} \dots x_{j_m}.$$

(To jest prawie to samo, co h_m w (33), tyle że nie dopuszczamy jednakowych indeksów). Znajdź funkcję tworzącą ciągu a_m i wyraź a_m za pomocą S_j ze wzoru (34). Zapisz wzory na a_1, a_2, a_3 i a_4 .

► 11. [M25] Korzystając z równości (39), można wyrazić wartości S za pomocą wartości h : $S_1 = h_1$, $S_2 = 2h_2 - h_1^2$, $S_3 = 3h_3 - 3h_1h_2 + h_1^3$ itd. Jakie są współczynniki przy $h_1^{k_1}h_2^{k_2}\dots h_m^{k_m}$ w reprezentacji S_m , gdy $k_1 + 2k_2 + \dots + mk_m = m$?

► 12. [M20] Mamy ciąg a_{mn} z dwoma indeksami $m, n = 0, 1, \dots$. Pokaż, w jaki sposób ten podwójny ciąg może być reprezentowany za pomocą jednej funkcji tworzącej dwóch zmiennych i wyznacz funkcję tworzącą ciągu $a_{mn} = \binom{n}{m}$.

13. [HM22] Transformata Laplace'a funkcji $f(x)$ to

$$\mathbf{L}f(s) = \int_0^\infty e^{-st} f(t) dt.$$

Niech a_0, a_1, a_2, \dots będzie nieskończonym ciągiem o zbieżnej funkcji tworzącej i niech $f(x)$ będzie funkcją schodkową $\sum_k a_k [0 \leq k \leq x]$. Wyraź transformatę Laplace'a funkcji $f(x)$ za pomocą funkcji tworzącej G ciągu $\langle a_k \rangle$.

14. [HM21] Udowodnij równanie (13).

15. [M28] Zbadaj $H(w) = \sum_{n \geq 0} G_n(z)w^n$ i znajdź postać zamkniętą funkcji tworzącej

$$G_n(z) = \sum_{k=0}^n \binom{n-k}{k} z^k = \sum_{k=0}^n \binom{2k-n-1}{k} (-z)^k.$$

16. [M22] Podaj prosty wzór na funkcję tworzącą $G_{nr}(z) = \sum_k a_{nkr} z^k$, gdzie a_{nkr} oznacza liczbę sposobów, na które można wybrać k spośród n elementów przy założeniu, że każdy element wybieramy co najwyżej r razy. (Dla $r = 1$ mamy $\binom{n}{k}$ sposobów, a dla $r \geq k$ mamy liczbę kombinacji z powtórzeniami, jak w ćwiczeniu 1.2.6–60).

17. [M25] Jakie są współczynniki $1/(1-z)^w$, gdy tę funkcję rozwiniemy w podwójny szereg potęgowy ze względu na z i w ?

► 18. [M25] Znajdź prosty wzór na następujące sumy przy założeniu, że n i r są liczbami całkowitymi: (a) $\sum_{1 \leq k_1 < k_2 < \dots < k_r \leq n} k_1 k_2 \dots k_r$; (b) $\sum_{1 \leq k_1 \leq k_2 \leq \dots \leq k_r \leq n} k_1 k_2 \dots k_r$. (Na przykład, gdy $n = 3$ i $r = 2$, wartości sum wynoszą odpowiednio $1 \cdot 2 + 1 \cdot 3 + 2 \cdot 3$ i $1 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 + 2 \cdot 2 + 2 \cdot 3 + 3 \cdot 3$).

19. [HM32] (C. F. Gauss, 1812) Sumy następujących szeregów skończonych są dobrze znane:

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots = \ln 2; \quad 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots = \frac{\pi}{4};$$

$$1 - \frac{1}{4} + \frac{1}{7} - \frac{1}{10} + \dots = \frac{\pi\sqrt{3}}{9} + \frac{1}{3} \ln 2.$$

Korzystając z definicji

$$H_x = \sum_{n \geq 1} \left(\frac{1}{n} - \frac{1}{n+x} \right)$$

zawartej w odpowiedzi do ćwiczenia 1.2.7–24, ciągi te można zapisać jako (odpowiednio):

$$1 - \frac{1}{2} H_{1/2}; \quad \frac{2}{3} - \frac{1}{4} H_{1/4} + \frac{1}{4} H_{3/4}; \quad \frac{3}{4} - \frac{1}{6} H_{1/6} + \frac{1}{6} H_{2/3}.$$

Udowodnij, że w ogólności $H_{p/q}$ ma wartość

$$\frac{q}{p} - \frac{\pi}{2} \cot \frac{p}{q} \pi - \ln 2q + 2 \sum_{0 < k < q/2} \cos \frac{2pk}{q} \pi \cdot \ln \sin \frac{k}{q} \pi,$$

gdzie p i q są liczbami całkowitymi spełniającymi nierówność $0 < p < q$. [Wskazówka: Z twierdzenia granicznego Abela suma równa się

$$\lim_{x \rightarrow 1^-} \sum_{n \geq 1} \left(\frac{1}{n} - \frac{1}{n + p/q} \right) x^{p+nq}.$$

Skorzystaj z (13) i tak zapisz ten szereg potęgowy, by dało się policzyć granicę].

20. [M21] Dla których współczynników c_{mk} zachodzi

$$\sum_{n \geq 0} n^m z^n = \sum_{k=0}^m c_{mk} z^k / (1-z)^{k+1}?$$

21. [HM30] Zapisz funkcję tworzącą ciągu $\langle n! \rangle$ i zbadaj jej własności.

22. [M21] Znajdź funkcję tworzącą $G(z)$, dla której

$$[z^n] G(z) = \sum_{k_0+2k_1+4k_2+8k_3+\dots=n} \binom{r}{k_0} \binom{r}{k_1} \binom{r}{k_2} \binom{r}{k_3} \dots$$

23. [M33] (L. Carlitz) (a) Udowodnij, że dla wszystkich liczb całkowitych $m \geq 1$ istnieją wielomiany $f_m(z_1, \dots, z_m)$ i $g_m(z_1, \dots, z_m)$, takie że wzór

$$\begin{aligned} \sum_{k_1, \dots, k_m \geq 0} \binom{r}{n-k_1} \binom{k_1}{n-k_2} \dots \binom{k_{m-1}}{n-k_m} z_1^{k_1} \dots z_m^{k_m} \\ = f_m(z_1, \dots, z_m)^{n-r} g_m(z_1, \dots, z_m)^r \end{aligned}$$

jest tożsamością dla wszystkich liczb całkowitych $n \geq r \geq 0$.

(b) Uogólnij ćwiczenie 15, znajdując postać zamkniętą sumy

$$S_n(z_1, \dots, z_m) = \sum_{k_1, \dots, k_m \geq 0} \binom{k_1}{n-k_2} \binom{k_2}{n-k_3} \dots \binom{k_m}{n-k_1} z_1^{k_1} \dots z_m^{k_m}$$

wyrażoną za pomocą funkcji f_m i g_m z punktu (a).

(c) Znajdź proste wyrażenie na $S_n(z_1, \dots, z_m)$, dla $z_1 = \dots = z_m = z$.

24. [M22] Udowodnij, że jeśli $G(z)$ jest dowolną funkcją tworzącą, to

$$\sum_k \binom{m}{k} [z^{n-k}] G(z)^k = [z^n] (1 + zG(z))^m.$$

Oblicz obie strony tej tożsamości, gdy $G(z)$ jest funkcją (a) $1/(1-z)$; (b) $(e^z - 1)/z$.

► **25.** [M23] Oblicz sumę $\sum_k \binom{n}{k} \binom{2n-2k}{n-k} (-2)^k$ poprzez uproszczenie równoważnego wzoru $\sum_k [w^k] (1-2w)^n [z^{n-k}] (1+z)^{2n-2k}$.

26. [M40] Zbadaj uogólnienie oznaczenia (31) polegające na tym, że na przykład dla funkcji $G(z)$ zadanej przez (1) możemy napisać $[z^2 - 2z^5] G(z) = a_2 - 2a_5$.

1.2.10. Analiza algorytmu

Spróbujemy teraz zastosować metody poznane w poprzednich punktach. Zbadajmy typowy algorytm.

Algorytm M (*Znajdowanie maksimum*). Danych jest n elementów $X[1], X[2], \dots, X[n]$; szukamy takich m i j , że $m = X[j] = \max_{1 \leq i \leq n} X[i]$ i j jest największym indeksem spełniającym ten warunek.

M1. [Inicjowanie] Niech $j \leftarrow n$, $k \leftarrow n-1$, $m \leftarrow X[n]$. (Podczas działania algorytmu $m = X[j] = \max_{k < i \leq n} X[i]$).

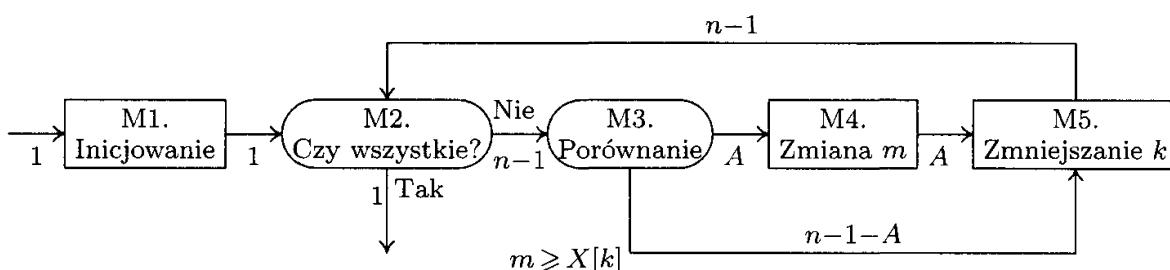
M2. [Czy wszystkie?] Jeśli $k = 0$, to wykonanie algorytmu się kończy.

M3. [Porównanie] Jeśli $X[k] \leq m$, to przejdź do M5.

M4. [Zmiana m] Niech $j \leftarrow k$, $m \leftarrow X[k]$. (Ta wartość m to nowe aktualne maksimum).

M5. [Zmniejszanie k] Zmniejsz k o jeden i wróć do M2. ■

Ten raczej oczywisty algorytm może wydawać się zbyt trywialny, by warto było zajmować się jego szczegółową analizą. Jest to jednak dobra okazja, żeby przyjrzeć się ogólnym sposobom analizowania trudniejszych algorytmów. Analiza algorytmów jest dla programistów bardzo ważna, bo zazwyczaj dla konkretnego zagadnienia istnieje kilka algorytmów i trzeba wybrać najlepszy.



Rys. 9. Algorytm M. Etykiety na strzałkach określają, ile razy sterowanie przechodzi daną krawędzią. Zauważmy, że musi być spełnione „pierwsze prawo Kirchhoffa”: suma tego, co wpływa do bloku, musi się równać sumie tego, co z bloku wypływa.

Zapotrzebowanie algorytmu M na pamięć nie zależy od danych wejściowych, dlatego będziemy analizować jedynie czas jego działania. W tym celu policzymy, ile razy wykonywany jest każdy krok (zobacz rysunek 9):

Numer kroku	Ile razy
M1	1
M2	n
M3	$n - 1$
M4	A
M5	$n - 1$

Wiedząc, ile razy wykonuje się każdy krok, możemy wyznaczyć czas wykonania algorytmu na konkretnym komputerze.

W powyższej tablicy znamy wszystkie wartości oprócz A – nie wiemy, ile razy trzeba zmienić wartość aktualnego maksimum. By doprowadzić studium do końca, musimy zbadać wartość A .

Analiza polega zazwyczaj na znalezieniu *minimalnej* wartości A (dla optymistów), *maksymalnej* wartości A (dla pesymistów), *średniej* wartości A (dla probabilistów) oraz *odchylenia standardowego* A (wartości mówiącej, jak blisko średniej pojawiają się wyniki).

Wartość *minimalna* A to zero. Taki przypadek zachodzi, gdy

$$X[n] = \max_{1 \leq k \leq n} X[k].$$

Wartością maksymalną jest $n - 1$. Tak zdarza się w przypadku

$$X[1] > X[2] > \dots > X[n].$$

Wartość średnia leży zatem między 0 i $n - 1$. Czy jest nią $\frac{1}{2}n$? A może \sqrt{n} ? By odpowiedzieć na te pytania, należy zdefiniować, co rozumiemy przez „wartość średnią”. Musimy przyjąć pewne założenia dotyczące charakteru danych wejściowych $X[1], X[2], \dots, X[n]$. Będziemy zakładać, że wartości $X[k]$ są różne i że każda z $n!$ permutacji tych wartości jest jednakowo prawdopodobna. (To w większości przypadków są rozsądne założenia, ale jak pokażemy w ćwiczeniach na końcu rozdziału, analizę można przeprowadzić, przyjmując odmienne założenia).

Zachowanie algorytmu M nie zależy od konkretnych wartości $X[k]$, istotne jest wyłącznie ich uporządkowanie. Na przykład, jeśli $n = 3$, to zakładamy, że każda z następujących sześciu możliwości jest jednakowo prawdopodobna:

Układ	Wartość A	Układ	Wartość A
$X[1] < X[2] < X[3]$	0	$X[2] < X[3] < X[1]$	1
$X[1] < X[3] < X[2]$	1	$X[3] < X[1] < X[2]$	1
$X[2] < X[1] < X[3]$	0	$X[3] < X[2] < X[1]$	2

Wychodzi średnia wartość A dla $n = 3$ równa $(0 + 1 + 0 + 1 + 1 + 2)/6 = 5/6$.

Jest oczywiste, że bez straty ogólności możemy przyjąć, iż $X[1], X[2], \dots, X[n]$ są liczbami $1, 2, \dots, n$ w pewnym porządku. Przy tym założeniu każda z $n!$ permutacji jest jednakowo prawdopodobna. Prawdopodobieństwo, że A przyjmie wartość k wynosi

$$p_{nk} = (\text{liczba permutacji } n \text{ elementów, dla których } A = k)/n!. \quad (1)$$

Na przykład, z powyższej tabeli $p_{30} = \frac{1}{3}$, $p_{31} = \frac{1}{2}$, $p_{32} = \frac{1}{6}$.

Średnią („oczekiwana”) definiujemy, tradycyjnie, jako

$$A_n = \sum_k kp_{nk}. \quad (2)$$

Wariancję V_n definiujemy jako średnią z $(A - A_n)^2$. Mamy stąd

$$\begin{aligned} V_n &= \sum_k (k - A_n)^2 p_{nk} = \sum_k k^2 p_{nk} - 2 A_n \sum_k kp_{nk} + A_n^2 \sum_k p_{nk} \\ &= \sum_k k^2 p_{nk} - 2 A_n A_n + A_n^2 = \sum_k k^2 p_{nk} - A_n^2. \end{aligned} \quad (3)$$

Wreszcie odchylenie standardowe σ_n definiujemy jako $\sqrt{V_n}$.

Znaczenie σ_n można sobie uświadomić, zauważając, że dla wszystkich $r \geq 1$, prawdopodobieństwo, że A znajdzie się w odległości większej niż $r\sigma_n$ od średniej, wynosi $1/r^2$. Na przykład, $|A - A_n| > 2\sigma_n$ zachodzi z prawdopodobieństwem $< 1/4$. (Dowód: Niech p oznacza powyższe prawdopodobieństwo. Wówczas jeśli $p > 0$, to średnia $(A - A_n)^2$ jest większa niż $p \cdot (r\sigma_n)^2 + (1 - p) \cdot 0$, to jest $V_n > pr^2V_n$). Ten fakt nazywany jest zazwyczaj nierównością Czebyszewa, choć w istocie został odkryty przez J. Bienaymé'a [Comptes Rendus Acad. Sci. Paris 37 (1853), 320–321].

Możemy określić zachowanie A przez wyznaczenie prawdopodobieństw p_{nk} . Nietrudno zrobić to indukcyjnie: we wzorze (1) chcemy wyznaczyć liczbę permutacji n elementów, dla których $A = k$. Oznaczmy tę liczbę przez $P_{nk} = n! p_{nk}$.

Rozważmy permutacje $x_1 x_2 \dots x_n$ zbioru $\{1, 2, \dots, n\}$, jak w punkcie 1.2.5. Gdy $x_1 = n$, wartość A jest o jeden większa niż wartość dla $x_2 \dots x_n$. Gdy $x_1 \neq n$, wartość A jest dokładnie taka sama, jak wartość dla $x_2 \dots x_n$. Stąd $P_{nk} = P_{(n-1)(k-1)} + (n-1)P_{(n-1)k}$ lub równoważnie

$$p_{nk} = \frac{1}{n} p_{(n-1)(k-1)} + \frac{n-1}{n} p_{(n-1)k}. \quad (4)$$

Z tego równania można wyznaczyć p_{nk} , jeśli zadamy warunki początkowe

$$p_{1k} = \delta_{0k}; \quad p_{nk} = 0, \quad \text{jeśli } k < 0. \quad (5)$$

Możemy teraz uzyskać informację o wielkościach p_{nk} za pomocą funkcji tworzących. Niech

$$G_n(z) = p_{n0} + p_{n1}z + \dots = \sum_k p_{nk}z^k. \quad (6)$$

Wiemy, że $A \leq n-1$, zatem $p_{nk} = 0$ dla dużych wartości k . Tym samym $G_n(z)$ jest w istocie wielomianem, chociaż dla wygody napisaliśmy sumę nieskończoną.

Z równań (5) mamy $G_1(z) = 1$, a z (4) mamy

$$G_n(z) = \frac{z}{n} G_{n-1}(z) + \frac{n-1}{n} G_{n-1}(z) = \frac{z+n-1}{n} G_{n-1}(z). \quad (7)$$

(Czytelnik powinien uważnie przyjrzeć się związkowi między wzorami (4) i (7)). Widzimy, że

$$\begin{aligned} G_n(z) &= \frac{z+n-1}{n} G_{n-1}(z) = \frac{z+n-1}{n} \frac{z+n-2}{n-1} G_{n-2}(z) = \dots \\ &= \frac{1}{n!} (z+n-1)(z+n-2) \dots (z+1) \\ &= \frac{1}{z+n} \binom{z+n}{n}. \end{aligned} \quad (8)$$

Zatem $G_n(z)$ jest w zasadzie współczynnikiem dwumianowym!

Ta funkcja pojawiła się w poprzednim punkcie, wzór 1.2.9–(27), gdzie mieliśmy

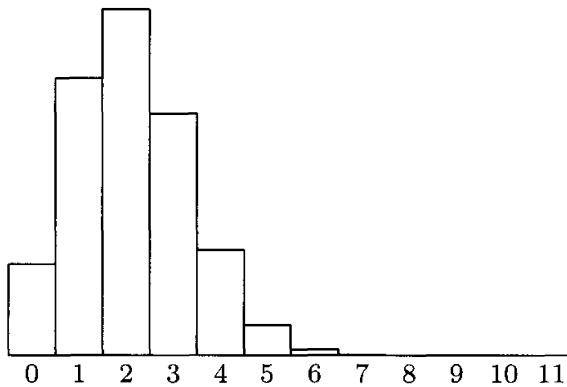
$$G_n(z) = \frac{1}{n!} \sum_k \binom{n}{k} z^{k-1}.$$

Można zatem przedstawić p_{nk} za pomocą liczb Stirlinga:

$$p_{nk} = \binom{n}{k+1} / n!. \quad (9)$$

Rysunek 10 pokazuje względne wartości p_{nk} dla $n = 12$.

Pozostaje wstawić p_{nk} do wzorów (2) i (3), a otrzymamy poszukiwaną średnią. Ale łatwiej to powiedzieć niż zrobić. W istocie bardzo rzadko się zdarza, by dało się jawnie wyznaczyć wartości prawdopodobieństwa p_{nk} . Dla większości



Rys. 10. Rozkład prawdopodobieństwa dla kroku M4 przy $n = 12$. Średnia jest równa $58301/27720$, w przybliżeniu 2.10, a wariancja w przybliżeniu wynosi 1.54.

problemów będziemy znali funkcję tworzącą $G_n(z)$, ale nie będziemy znać prostej postaci samych prawdopodobieństw. Z tego powodu niezmiernie istotne jest to, że można wyznaczyć średnią i wariancję wprost z funkcji tworzącej.

Przypuśćmy, że mamy funkcję tworzącą, której współczynniki reprezentują prawdopodobieństwa:

$$G(z) = p_0 + p_1 z + p_2 z^2 + \dots$$

Wielkość p_k jest tutaj prawdopodobieństwem zdarzenia, że „coś” przyjęło wartość k . Chcemy obliczyć wartość średniej i wariancji:

$$\text{mean}(G) = \sum_k kp_k, \quad \text{var}(G) = \sum_k k^2 p_k - (\text{mean}(G))^2. \quad (10)$$

Nietrudno to zrobić, korzystając z różniczkowania. Zauważmy, że

$$G(1) = 1, \quad (11)$$

bo $G(1) = p_0 + p_1 + p_2 + \dots$ jest sumą wszystkich możliwych prawdopodobieństw. Analogicznie na mocy $G'(z) = \sum_k kp_k z^{k-1}$ mamy

$$\text{mean}(G) = \sum_k kp_k = G'(1). \quad (12)$$

Na koniec różniczkujemy jeszcze raz i otrzymujemy (zobacz ćwiczenie 2)

$$\text{var}(G) = G''(1) + G'(1) - G'(1)^2. \quad (13)$$

Równania (12) i (13) są poszukiwanymi przedstawieniami średniej i wariancji za pomocą funkcji tworzącej.

W naszym przypadku chcemy obliczyć $G'_n(1) = A_n$. Ze wzoru (7) mamy

$$G'_n(z) = \frac{1}{n} G_{n-1}(z) + \frac{z+n-1}{n} G'_{n-1}(z);$$

$$G'_n(1) = \frac{1}{n} + G'_{n-1}(1).$$

Z warunku początkowego $G'_1(1) = 0$ otrzymujemy

$$A_n = G'_n(1) = H_n - 1. \quad (14)$$

To jest poszukiwana wielkość, mówiąca jaka jest średnia liczba wykonień kroku M4. Dla dużych n wynosi ona w przybliżeniu $\ln n$. [Uwaga: r -ty moment $A+1$, tj. wielkość $\sum_k (k+1)^r p_{nk}$, wynosi $[z^n] (1-z)^{-1} \sum_k \binom{r}{k} (\ln \frac{1}{1-z})^k$ i jest w przybliżeniu równy $(\ln n)^r$. Zobacz P. B. M. Roes *CACM* **9** (1966), 342. Rozkładem A po raz pierwszy zajmowali się F. G. Foster i A. Stuart, *J. Roy. Stat. Soc.* **B16** (1954), 1–22].

Możemy podążyć tym tropem, by obliczyć wariancję (tj. V_n). Zatrzymajmy się jednak na chwilę przy ważnym spostrzeżeniu:

Twierdzenie A. Niech G i H będą funkcjami tworzącymi, takimi że $G(1) = H(1) = 1$. Dla $\text{mean}(G)$ i $\text{var}(G)$ zdefiniowanych we wzorach (12) i (13) mamy

$$\text{mean}(GH) = \text{mean}(G) + \text{mean}(H); \quad \text{var}(GH) = \text{var}(G) + \text{var}(H). \quad (15)$$

Udowodnimy to twierdzenie później. Mówi ono, że średnia i wariancja iloczynu funkcji tworzących może być zredukowana do sumy. ■

Biorąc $Q_n(z) = (z+n-1)/n$, otrzymujemy $Q'_n(1) = 1/n$, $Q''_n(1) = 0$; zatem

$$\text{mean}(Q_n) = \frac{1}{n}, \quad \text{var}(Q_n) = \frac{1}{n} - \frac{1}{n^2}.$$

Ponieważ $G_n(z) = \prod_{k=2}^n Q_k(z)$, ostatecznie otrzymujemy

$$\begin{aligned} \text{mean}(G_n) &= \sum_{k=2}^n \text{mean}(Q_k) = \sum_{k=2}^n \frac{1}{k} = H_n - 1 \\ \text{var}(G_n) &= \sum_{k=2}^n \text{var}(Q_k) = \sum_{k=1}^n \left(\frac{1}{k} - \frac{1}{k^2} \right) = H_n - H_n^{(2)}. \end{aligned}$$

Reasumując, określiliśmy statystykę wielkości A : minimum, średnią, maksimum i odchylenie standardowe.

$$A = \left(\min 0, \quad \text{ave } H_n - 1, \quad \max n - 1, \quad \text{dev } \sqrt{H_n - H_n^{(2)}} \right). \quad (16)$$

Z pomocą notacji (16) będziemy w dalszych rozdziałach opisywać charakterystyki zmiennych losowych.

Zakończyliśmy analizę algorytmu M. Pojawiła się w niej pewna nowość – użyliśmy prawdopodobieństwa. W niniejszej książce w większości przypadków korzystamy wyłącznie z elementarnej teorii prawdopodobieństwa. Proste metody zliczania oraz podane definicje średniej, wariancji i odchylenia standardowego pozwalają odpowiedzieć na większość zadawanych przez nas pytań. Analizując bardziej skomplikowane algorytmy, nabędziemy biegłości w posługiwaniu się prawdopodobieństwem.

W celu zebrania doświadczeń zastanówmy się nad pewnym prostym problemem. Prawdopodobieństwo zawsze kojarzy się z rzutem monetą. Rzucamy monetą n razy, prawdopodobieństwo wyrzucenia orła wynosi p . Jaka jest średnia liczba wyrzuconych orłów? Jakie jest odchylenie standardowe?

Będziemy rozważać monetę niesymetryczną, co oznacza, że nie zakładamy $p = \frac{1}{2}$. Dzięki temu problem jest ciekawszy. W istocie każda prawdziwa moneta jest niesymetryczna (bo inaczej nie moglibyśmy odróżnić jednej strony od drugiej).

Postępując tak jak w poprzednim przykładzie, ustalamy, że p_{nk} to prawdopodobieństwo wyrzucenia k orłów oraz że $G_n(z)$ jest funkcją tworzącą tej zmiennej losowej. Mamy

$$p_{nk} = p p_{(n-1)(k-1)} + q p_{(n-1)k}, \quad (17)$$

gdzie $q = 1 - p$ jest prawdopodobieństwem wyrzucenia reszki. Jak poprzednio ze wzoru (17) wyprowadzamy $G_n(z) = (q + pz)G_{n-1}(z)$, a przy oczywistym warunku początkowym $G_1(z) = q + pz$ mamy

$$G_n(z) = (q + pz)^n. \quad (18)$$

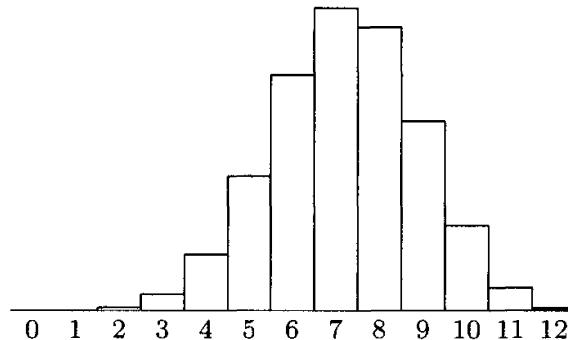
Stąd na mocy twierdzenia A:

$$\begin{aligned} \text{mean}(G_n) &= n \text{ mean}(G_1) = pn; \\ \text{var}(G_n) &= n \text{ var}(G_1) = (p - p^2)n = pqn. \end{aligned}$$

Dla liczby wyrzuconych orłów mamy zatem:

$$(\min 0, \quad \text{ave } pn, \quad \max n, \quad \text{dev } \sqrt{pqn}). \quad (19)$$

Rysunek 11 pokazuje wartości p_{nk} dla $p = \frac{3}{5}$, $n = 12$. Gdy odchylenie standardowe jest proporcjonalne do \sqrt{n} , a różnica między wartością maksymalną a minimalną wynosi n , możemy uważać, że zmienna jest „ustabilizowana” wokół średniej.



Rys. 11. Rozkład prawdopodobieństwa rzutu monetą: 12 niezależnych rzutów, szansa sukcesu w każdym rzucie równa jest $3/5$.

Rozważmy jeszcze jedno proste zagadnienie. Przypuśćmy, że w pewnym doświadczeniu prawdopodobieństwa otrzymania wartości $1, 2, \dots, n$ są *równe*. Funkcja tworząca w takim przypadku to

$$G(z) = \frac{1}{n}z + \frac{1}{n}z^2 + \cdots + \frac{1}{n}z^n = \frac{1}{n} \frac{z^{n+1} - z}{z - 1}. \quad (20)$$

Po długich przekształceniach dochodzimy do wniosku, że

$$G'(z) = \frac{nz^{n+1} - (n+1)z^n + 1}{n(z-1)^2};$$

$$G''(z) = \frac{n(n-1)z^{n+1} - 2(n+1)(n-1)z^n + n(n+1)z^{n-1} - 2}{n(z-1)^3}.$$

Aby teraz obliczyć średnią i wariancję, musimy wyznaczyć $G'(1)$ i $G''(1)$. Ale po podstawieniu $z = 1$ ułamki upraszczają się do postaci 0/0. W związku z tym musimy szukać granicy przy z dążącym do jedynki, a to niekoniecznie jest prostym zadaniem.

Na szczęście istnieje inna, o wiele łatwiejsza droga. Z twierdzenia Taylora otrzymujemy

$$G(1+z) = G(1) + G'(1)z + \frac{G''(1)}{2!}z^2 + \dots; \quad (21)$$

możemy zatem w (20) zamienić z na $z+1$ i po prostu odczytać współczynniki:

$$G(1+z) = \frac{1}{n} \frac{(1+z)^{n+1} - 1 - z}{z} = 1 + \frac{n+1}{2}z + \frac{(n+1)(n-1)}{6}z^2 + \dots$$

Stąd $G'(1) = \frac{1}{2}(n+1)$, $G''(1) = \frac{1}{3}(n+1)(n-1)$, zatem dla rozkładu jednostajnego mamy:

$$\left(\min 1, \quad \text{ave } \frac{n+1}{2}, \quad \max n, \quad \text{dev } \sqrt{\frac{(n+1)(n-1)}{12}} \right). \quad (22)$$

W tym przypadku odchylenie równe w przybliżeniu $0.289n$ oznacza sytuację jawnie *niestabilną*.

Zamknijmy ten rozdział dowodem twierdzenia A i porównaniem wprowadzonych pojęć z klasyczną teorią prawdopodobieństwa. Przypuśćmy, że X jest zmienną losową, która przyjmuje wyłącznie wartości całkowite nieujemne, przy tym $X = k$ z prawdopodobieństwem p_k . Wówczas $G(z) = p_0 + p_1z + p_2z^2 + \dots$ nazywamy *funkcją tworzącą prawdopodobieństwa* zmiennej X , a o wielkości $G(e^{it}) = p_0 + p_1e^{it} + p_2e^{2it} + \dots$ tradycyjnie mówimy jako o *funkcji charakterystycznej* tego rozkładu. Rozkład zadany iloczynem dwóch takich funkcji tworzących jest nazywany *splotem* rozkładów i reprezentuje sumę dwóch niezależnych zmiennych losowych, których rozkłady są reprezentowane przez wymnażane funkcje tworzące.

Wartość średnia (przeciętna) zmiennej losowej X nazywana jest często *wartością oczekiwana* i oznaczana przez $E X$. Wariancja X to zatem $E X^2 - (E X)^2$. W tym zapisie funkcja tworząca prawdopodobieństwa zmiennej X to $G(z) = E z^X$, czyli wartość oczekiwana wielkości z^X , gdy X przybiera wyłącznie nieujemne wartości całkowite. Analogicznie, gdy X jest stwierdzeniem, które może być albo prawdziwe, albo fałszywe, prawdopodobieństwo, że X jest prawdziwe zapisuje się w konwencji Iversona $\Pr(X) = E[X]$ (zobacz 1.2.3-(16)).

Średnia i wariancja to dwa z tzw. *półniezmienników* wprowadzonych przez Thiele'go w 1903 roku. Półniezmienniki $\kappa_1, \kappa_2, \kappa_3, \dots$ definiuje się za pomocą reguły

$$\frac{\kappa_1 t}{1!} + \frac{\kappa_2 t^2}{2!} + \frac{\kappa_3 t^3}{3!} + \dots = \ln G(e^t). \quad (23)$$

Mamy więc

$$\kappa_n = \left. \frac{d^n}{dt^n} \ln G(e^t) \right|_{t=0};$$

w szczególności

$$\kappa_1 = \left. \frac{e^t G'(e^t)}{G(e^t)} \right|_{t=0} = G'(1),$$

bo $G(1) = \sum_k p_k = 1$ oraz

$$\kappa_2 = \left. \frac{e^{2t} G''(e^t)}{G(e^t)} + \frac{e^t G'(e^t)}{G(e^t)} - \frac{e^{2t} G'(e^t)^2}{G(e^t)^2} \right|_{t=0} = G''(1) + G'(1) - G'(1)^2.$$

Z uwagi na fakt, że półniezmienniki są zdefiniowane za pomocą *logarytmów* funkcji tworzących, twierdzenie A jest oczywiste. Co więcej, można je uogólnić na wszystkie półniezmienniki.

Rozkład normalny to taki rozkład, dla którego wszystkie półniezmienniki poza średnią i wariancją są równe zero. Dla rozkładu normalnego możemy istotnie poprawić nierówność Czebyszewa: prawdopodobieństwo, że zmienna losowa o rozkładzie normalnym różni się od średniej o mniej niż odchylenie standardowe wynosi

$$\frac{1}{\sqrt{2\pi}} \int_{-1}^{+1} e^{-t^2/2} dt,$$

to jest około 68.268949213709%. Różnica jest mniejsza niż podwojone odchylenie standardowe z prawdopodobieństwem 95.449973610364%, a mniejsza niż potrojone odchylenie standardowe z prawdopodobieństwem 99.730020393674%. Rozkłady zadane za pomocą wzorów (8) i (18) są w przybliżeniu normalne dla dużych n (zobacz ćwiczenia 13 i 14).

Często chcielibyśmy wiedzieć, że jest mało prawdopodobne, iż zmienna losowa nie przyjmie wartości dużo większych lub dużo mniejszych od średniej. Dwa bardzo proste, ale bardzo ważne wzory, zwane *nierównościami ogonowymi*, pozwalają oszacować prawdopodobieństwa takich zdarzeń. Gdy $G(z)$ jest funkcją tworzącą prawdopodobieństwa zmiennej losowej X , zachodzą nierówności

$$\Pr(X \leq r) \leq x^{-r} G(x) \quad \text{dla } 0 < x \leq 1; \quad (24)$$

$$\Pr(X \geq r) \leq x^{-r} G(x) \quad \text{dla } x \geq 1. \quad (25)$$

Dowody są proste: dla $G(z) = p_0 + p_1 z + p_2 z^2 + \dots$, mamy

$$\Pr(X \leq r) = p_0 + p_1 + \dots + p_{\lfloor r \rfloor} \leq x^{-r} p_0 + x^{1-r} p_1 + \dots + x^{\lfloor r \rfloor - r} p_{\lfloor r \rfloor} \leq x^{-r} G(x)$$

dla $0 < x \leq 1$, natomiast

$$\Pr(X \geq r) = p_{\lceil r \rceil} + p_{\lceil r \rceil + 1} + \dots \leq x^{\lceil r \rceil - r} p_{\lceil r \rceil} + x^{\lceil r \rceil + 1 - r} p_{\lceil r \rceil + 1} + \dots \leq x^{-r} G(x)$$

dla $x \geq 1$. Wybierając wartość x , dla której prawe strony wzorów (24) i (25) przyjmują wartości minimalne lub w przybliżeniu minimalne, uzyskujemy często górne ograniczenia, które okazują się dosyć bliskie faktycznym prawdopodobieństwom ogonowym stojącym po lewej stronie nierówności.

Ćwiczenia 21–23 pokazują kilka istotnych zastosowań nierówności ogonowych. Nierówności ogonowe są szczególnymi przypadkami bardziej ogólnych zasad, wskazanych przez A. N. Kołmogorowa w jego książce *Grundbegriffe der Wahrscheinlichkeitsrechnung* (Springer, 1933): jeśli $f(t) \geq s > 0$ dla wszystkich $t \geq r$, to $\Pr(X \geq r) \leq s^{-1} E f(X)$, jeśli tylko istnieje $E f(X)$. Dla $f(t) = x^t$ i $s = x^r$ dostajemy wzór (25).

ĆWICZENIA

1. [10] Wyznacz wartość p_{n0} ze wzorów (4) i (5), a następnie zinterpretuj wynik w kontekście algorytmu M.

2. [HM16] Wyprowadź równanie (13) ze wzorów (10).

3. [M15] Jaka jest wartość minimalna, maksymalna, średnia oraz odchylenie standardowe liczby wykonań kroku M4, gdy używamy algorytmu M do wyznaczenia największego z 1000 różnych losowo uporządkowanych obiektów? (W odpowiedzi podaj przybliżenia dziesiętne).

4. [M10] Podaj postać zamkniętą wzoru na wartości p_{nk} w doświadczeniu dotyczącym rzutu monetą, równość (17).

5. [M13] Jaka jest średnia i odchylenie standardowe rozkładu z rysunku 11?

6. [HM27] Obliczyliśmy średnią i wariancję rozkładów (8), (18) i (20). Jaki jest trzeci półniezmiennik κ_3 w każdym z tych przypadków?

► **7. [M27]** W analizie algorytmu M założyliśmy, że wszystkie $X[k]$ są różne. Spróbujmy osłabić to założenie, wymagając, by $X[1], X[2], \dots, X[n]$ zawierały dokładnie m różnych wartości. Poza tym wartości są losowe. Jaki jest rozkład prawdopodobieństwa A dla tak postawionego problemu?

► **8. [M20]** Przypuśćmy, że elementy $X[k]$ są wybierane losowo ze zbioru M różnych elementów, tak że każdy z M^n możliwych wyborów $X[1], X[2], \dots, X[n]$ jest jednakowo prawdopodobny. Jakie jest prawdopodobieństwo, że wszystkie $X[k]$ będą różne?

9. [M25] Uogólnij wynik poprzedniego ćwiczenia i znajdź wzór na prawdopodobieństwo zdarzenia, że pośród elementów X trafi się dokładnie m różnych wartości. Wyraź odpowiedź za pomocą liczb Stirlinga.

10. [M20] Zestaw wyniki poprzednich ćwiczeń, by otrzymać wzór na prawdopodobieństwo zdarzenia, że $A = k$ przy założeniu, że elementy tablicy X są wybierane losowo ze zbioru M elementów.

► **11. [M15]** Jak zmienią się półniezmienniki rozkładu, gdy zamienimy $G(z)$ na $F(z) = z^n G(z)$?

12. [HM21] Gdy $G(z) = p_0 + p_1 z + p_2 z^2 + \dots$ reprezentuje rozkład prawdopodobieństwa, wartości $M_n = \sum_k k^n p_k$ i $m_n = \sum_k (k - M_1)^n p_k$ nazywamy odpowiednio „ n -tym momentem” i „ n -tym momentem centralnym”. Pokaż, że $G(e^t) = 1 + M_1 t +$

$M_2 t^2/2! + \dots$, następnie skorzystaj ze wzoru Arbogasta (ćwiczenie 1.2.5–21), by pokazać, że

$$\kappa_n = \sum_{\substack{k_1, k_2, \dots, k_n \geq 0 \\ k_1 + 2k_2 + \dots = n}} \frac{(-1)^{k_1+k_2+\dots+k_n-1} n! (k_1 + k_2 + \dots + k_n - 1)!}{k_1! 1^{k_1} k_2! 2^{k_2} \dots k_n! n^{k_n}} M_1^{k_1} M_2^{k_2} \dots M_n^{k_n}.$$

W szczególności, $\kappa_1 = M_1$, $\kappa_2 = M_2 - M_1^2$ (co już wiedzieliśmy), $\kappa_3 = M_3 - 3M_1 M_2 + 2M_1^3$ i $\kappa_4 = M_4 - 4M_1 M_3 + 12M_1^2 M_2 - 3M_2^2 - 6M_1^4$. Jak w analogiczny sposób zapisuje się κ_n za pomocą momentów centralnych m_2, m_3, \dots , gdy $n \geq 2$?

13. [HM38] Mówimy, że ciąg funkcji tworzących prawdopodobieństwa $G_n(z)$ o średnich μ_n i odchyleniach σ_n przybliża rozkład normalny, jeśli

$$\lim_{n \rightarrow \infty} e^{-it\mu_n/\sigma_n} G_n(e^{it/\sigma_n}) = e^{-t^2/2}$$

dla wszystkich rzeczywistych wartości t . Skorzystaj z takiej funkcji $G_n(z)$, jak we wzorze (8), by pokazać, że $G_n(z)$ przybliża rozkład normalny.

Uwaga: Można pokazać, że według powyższej definicji „przybliżanie rozkładu normalnego” jest równoważne stwierdzeniu

$$\lim_{n \rightarrow \infty} \text{prawdopodobieństwo} \left(\frac{X_n - \mu_n}{\sigma_n} \leqslant x \right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt,$$

gdzie X_n jest wielkością losową, której prawdopodobieństwa są określone przez $G_n(z)$. To jest przypadek szczególny ważnego „twierdzenia o ciągłości” P. Lévy’ego, podstawowego wyniku w matematycznej teorii prawdopodobieństwa. Dowód twierdzenia Lévy’ego odwiódłby nas daleko od tematu tej książki, jakkolwiek nie jest on przesadnie trudny [zobacz na przykład B. V. Gnedenko, A. N. Kolmogorov *Limit Distributions for Sums of Independent Random Variables*, tłum. K. L. Chung (Reading, Mass.: Addison-Wesley, 1954)].

14. [HM30] (A. de Moivre) Stosując oznaczenia z poprzedniego ćwiczenia, pokaż, że rozkład dwumianowy $G_n(z)$ zadany równaniem (18) przybliża rozkład normalny.

15. [HM23] Jeśli prawdopodobieństwo zdarzenia, że pewna wielkość ma wartość k , wynosi $e^{-\mu} (\mu^k / k!)$, to mówimy, że ma ono rozkład Poissona ze średnią μ .

- a) Jak wygląda funkcja tworząca tego prawdopodobieństwa?
- b) Jakie są wartości półniezmienników?
- c) Pokaż, że przy $n \rightarrow \infty$ rozkład Poissona ze średnią np przybliża rozkład normalny w sensie ćwiczenia 13.

16. [M25] Założymy, że X jest zmienną losową, której wartości są *mieszanką* rozkładów opisanych przez funkcje tworzące $g_1(z), g_2(z), \dots, g_r(z)$ w tym sensie, że używamy $g_k(z)$ z prawdopodobieństwem p_k , gdzie $p_1 + p_2 + \dots + p_r = 1$. Jak wygląda funkcja tworząca X ? Wyraź średnią i wariancję X za pomocą średnich i wariancji g_1, g_2, \dots, g_r .

► **17.** [M27] Niech $f(z)$ i $g(z)$ będą funkcjami tworzącymi reprezentującymi rozkłady prawdopodobieństwa.

- a) Pokaż, że $h(z) = g(f(z))$ też jest funkcją tworzącą reprezentującą rozkład prawdopodobieństwa.
- b) Zinterpretuj $h(z)$ za pomocą $f(z)$ i $g(z)$. (Jaki jest znaczenie prawdopodobieństw reprezentowanych przez współczynniki $h(z)$?)
- c) Zapisz wzór na średnią i wariancję h za pomocą średniej i wariancji f oraz g .

- 18.** [M28] Przypuśćmy, że wśród różnych wartości przyjmowanych przez $X[1], X[2], \dots, X[n]$ w algorytmie M jest dokładnie k_1 jedynek, k_2 dwórek, \dots, k_n n -ek, ustawionych w losowym porządku. (Mamy

$$k_1 + k_2 + \dots + k_n = n.$$

W tekście zakładaliśmy, że $k_1 = k_2 = \dots = k_n = 1$). Pokaż, że w tej ogólnej sytuacji funkcja tworząca (8) ma postać

$$\left(\frac{k_{n-1}z + k_n}{k_{n-1} + k_n} \right) \left(\frac{k_{n-2}z + k_{n-1} + k_n}{k_{n-2} + k_{n-1} + k_n} \right) \dots \left(\frac{k_1z + k_2 + \dots + k_n}{k_1 + k_2 + \dots + k_n} \right),$$

gdy przyjmiemy $0/0 = 1$.

- 19.** [M21] Jeśli $a_k > a_j$ dla $1 \leq j < k$, to mówimy, że a_k jest *maksimum lewostronnym* ciągu $a_1 a_2 \dots a_n$. Przypuśćmy, że $a_1 a_2 \dots a_n$ jest permutacją liczb $\{1, 2, \dots, n\}$. Niech $b_1 b_2 \dots b_n$ będzie permutacją odwrotną, tj. $a_k = l$ wtedy i tylko wtedy, gdy $b_l = k$. Pokaż, że a_k jest maksimum lewostronnym $a_1 a_2 \dots a_n$ wtedy i tylko wtedy, gdy k jest minimum prawostronnym ciągu $b_1 b_2 \dots b_n$.

- **20.** [M22] Przypuśćmy, że chcemy obliczyć $\max\{|a_1 - b_1|, |a_2 - b_2|, \dots, |a_n - b_n|\}$ dla $b_1 \leq b_2 \leq \dots \leq b_n$. Pokaż, że wystarczy obliczyć $\max\{m_L, m_R\}$, gdzie

$$m_L = \max\{a_k - b_k \mid a_k \text{ jest maksimum lewostronnym ciągu } a_1, a_2 \dots a_n\},$$

$$m_R = \max\{b_k - a_k \mid a_k \text{ jest minimum prawostronnym ciągu } a_1, a_2 \dots a_n\}.$$

[Jeśli zatem a stoją w losowym porządku, to liczba tych k , dla których trzeba wykonać odejmowanie, wynosi około $2 \ln n$].

- **21.** [HM21] Niech X będzie zmienną losową opisującą liczbę orłów w n rzutach o funkcji tworzącej (18). Skorzystaj ze wzoru (25), by udowodnić, że dla $\epsilon \geq 0$

$$\Pr(X \geq n(p + \epsilon)) \leq e^{-\epsilon^2 n / (2q)}$$

i pokaż podobne oszacowanie na $\Pr(X \leq n(p - \epsilon))$.

- **22.** [HM22] Przypuśćmy, że X ma funkcję tworzącą $(q_1 + p_1 z)(q_2 + p_2 z) \dots (q_n + p_n z)$, gdzie $p_k + q_k = 1$ dla $1 \leq k \leq n$. Niech $\mu = \mathbb{E} X = p_1 + p_2 + \dots + p_n$. (a) Udowodnij, że

$$\Pr(X \leq \mu r) \leq (r^{-r} e^{r-1})^\mu, \text{ gdy } 0 < r \leq 1;$$

$$\Pr(X \geq \mu r) \leq (r^{-r} e^{r-1})^\mu, \text{ gdy } r \geq 1.$$

(b) Zapisz prawe strony tych oszacowań w przejrzystej postaci, gdy $r \approx 1$. (c) Pokaż, że gdy r jest wystarczająco duże, mamy $\Pr(X \geq \mu r) \leq 2^{-\mu r}$.

- 23.** [HM23] Oszacuj prawdopodobieństwa ogonowe zmiennej losowej, która ma *ujemny rozkład dwumianowy* opisany funkcją tworzącą $(q - pz)^{-n}$, gdzie $q = p + 1$.

*1.2.11. Reprezentacje asymptotyczne

Częstość, gdy porównujemy jakieś wielkości, nie interesują nas ich dokładne, a jedynie przybliżone wartości. Na przykład pochodzące od Stirlinga przybliżenie silni jest bardzo poręczne dla dużych n , czyniliśmy także użytek z faktu, że $H_n \approx \ln n + \gamma$. Wyprowadzenia takich wzorów *asymptotycznych* zawierają zazwyczaj elementy matematyki wyższej, niemniej jednak w poniższych rozdziałach nie będziemy wykraczać poza podstawową analizę matematyczną.

***1.2.11.1. Notacja wielkie-*O*.** Paul Bachmann w swojej książce *Analytische Zahlentheorie* (1894) wprowadził bardzo pozytyczną notację do oznaczania przybliżeń. Jest to tzw. notacja wielkie-*O*, pozwalająca zastąpić znak „ \approx ” znakiem „ $=$ ” oraz stopniować dokładność. Na przykład,

$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right). \quad (1)$$

(Czytaj „ha-en równa się logarytm naturalny z en plus stała Ojlera plus o od jedna enta”, czasami też „... plus wielkie o od jedna enta”).

Ogólnie rzecz biorąc, notacji $O(f(n))$ można używać wszędzie tam, gdzie $f(n)$ jest funkcją dodatniej zmiennej całkowitej n . Notacja oznacza *ograniczoną wartość, która nie jest dokładnie znana*. Każde wystąpienie $O(f(n))$ znaczy tyle, że istnieją dodatnie stałe M oraz n_0 , takie że liczba x_n reprezentowana przez $O(f(n))$ spełnia warunek $|x_n| \leq M|f(n)|$ dla wszystkich całkowitych $n \geq n_0$. Nie mówimy, *jakie* konkretnie są M i n_0 . Te stałe mogą być inne dla każdego wystąpienia O .

Wzór (1) oznacza na przykład, że $|H_n - \ln n - \gamma| \leq M/n$ dla $n \geq n_0$. Chociaż stałe M i n_0 nie są podane, możemy mieć pewność, że wielkość $O(1/n)$ będzie odpowiednio mała dla dużych n .

Spójrzmy na kilka innych przykładów. Wiemy, że

$$1^2 + 2^2 + \cdots + n^2 = \frac{1}{3}n(n + \frac{1}{2})(n + 1) = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n;$$

mamy zatem

$$1^2 + 2^2 + \cdots + n^2 = O(n^4), \quad (2)$$

$$1^2 + 2^2 + \cdots + n^2 = O(n^3), \quad (3)$$

$$1^2 + 2^2 + \cdots + n^2 = \frac{1}{3}n^3 + O(n^2). \quad (4)$$

Równość (2) jest oszacowaniem bardzo zgrubnym, ale poprawnym. Równość (3) jest silniejszym stwierdzeniem, a jeszcze silniejsze jest (4). By udowodnić te równości, pokażemy, że jeśli $P(n) = a_0 + a_1n + \cdots + a_m n^m$ jest dowolnym wielomianem stopnia nie większego niż m , to $P(n) = O(n^m)$. Jest tak ponieważ

$$\begin{aligned} |P(n)| &\leq |a_0| + |a_1|n + \cdots + |a_m|n^m = (|a_0|/n^m + |a_1|/n^{m-1} + \cdots + |a_m|)n^m \\ &\leq (|a_0| + |a_1| + \cdots + |a_m|)n^m \end{aligned}$$

dla $n \geq 1$. Możemy zatem wziąć $M = |a_0| + |a_1| + \cdots + |a_m|$ i $n_0 = 1$ albo, na przykład, $M = |a_0|/2^m + |a_1|/2^{m-1} + \cdots + |a_m|$ i $n_0 = 2$.

Notacja wielkie-*O* jest bardzo wygodna do zapisywania wartości przybliżonych, gdyż w prosty sposób ujmuje często wykorzystywane własności. Poza tym pozwala abstrahować od nieistotnych szczegółów. Co więcej, w przekształceniach algebraicznych posługujemy się nią podobnie, jak innymi wielkościami, chociaż trzeba pamiętać o paru szczegółach. Po pierwsze, gdy pracujemy z notacją wielkie-*O*, równości stają się jednokierunkowe: piszemy $\frac{1}{2}n^2 + n = O(n^2)$, ale nigdy $O(n^2) = \frac{1}{2}n^2 + n$. (W przeciwnym razie na mocy $\frac{1}{4}n^2 = O(n^2)$, uzyskalibyśmy takie kwiatki, jak $\frac{1}{4}n^2 = \frac{1}{2}n^2 + n$). Podstawowa zasada mówi,

że prawa strona równości nie może zawierać więcej informacji niż strona lewa; prawa strona równości jest „bardziej zgrubna”.

Powyższą konwencję dotyczącą użycia „ $=$ ” precyzyjnie sformułujemy następująco. Wzory zawierające $O(f(n))$ można traktować jak funkcje n . Symbol $O(f(n))$ oznacza zbiór wszystkich takich funkcji g określonych na zbiorze liczb całkowitych, że istnieją takie stałe M i n_0 , że $|g(n)| \leq M |f(n)|$ dla wszystkich całkowitych $n \geq n_0$. Jeśli S i T są zbiorami funkcji, to $S + T$ oznacza zbiór $\{g + h \mid g \in S \text{ i } h \in T\}$. Podobnie można zdefiniować $S + c$, $S - T$, $S \cdot T$, $\log S$ itd. Jeśli $\alpha(n)$ i $\beta(n)$ są wzorami zawierającymi wielkie- O , to zapis $\alpha(n) = \beta(n)$ mówi, że zbiór funkcji określanych przez $\alpha(n)$ jest *zawarty w* zbiorze funkcji określonych przez $\beta(n)$.

Wiele operacji, do których przywykliśmy dla zwykłych równań, jest również poprawnych dla notacji wielkie- O . Jeśli $\alpha(n) = \beta(n)$ i $\beta(n) = \gamma(n)$, to $\alpha(n) = \gamma(n)$. Także gdy $\alpha(n) = \beta(n)$, a $\delta(n)$ jest wzorem otrzymanym z podstawienia $\beta(n)$ za pewne wystąpienie $\alpha(n)$ we wzorze $\gamma(n)$, zachodzi $\gamma(n) = \delta(n)$. Z tych dwóch faktów wynika na przykład, że jeśli $g(x_1, x_2, \dots, x_m)$ jest dowolną funkcją rzeczywistą i jeśli $\alpha_k(n) = \beta_k(n)$ dla $1 \leq k \leq m$, to $g(\alpha_1(n), \alpha_2(n), \dots, \alpha_m(n)) = g(\beta_1(n), \beta_2(n), \dots, \beta_m(n))$.

Oto proste przekształcenia wyrażeń zawierających wielkie- O :

$$f(n) = O(f(n)), \quad (5)$$

$$c \cdot O(f(n)) = O(f(n)), \quad \text{gdy } c \text{ jest stałą}, \quad (6)$$

$$O(f(n)) + O(f(n)) = O(f(n)), \quad (7)$$

$$O(O(f(n))) = O(f(n)), \quad (8)$$

$$O(f(n))O(g(n)) = O(f(n)g(n)), \quad (9)$$

$$O(f(n)g(n)) = f(n)O(g(n)). \quad (10)$$

Notacja wielkie- O jest wykorzystywana także dla funkcji zmiennej zespolonej z do opisu jej zachowania w otoczeniu punktu $z = 0$. Piszymy $O(f(z))$ na oznaczenie wszystkich takich wielkości $g(z)$, że $|g(z)| \leq M |f(z)|$, dla dowolnego $|z| < r$. (Tak jak przedtem, nie podajemy konkretnych wartości M i r , choć możemy je określić w razie potrzeby). Z kontekstu wystąpienia notacji wielkie- O powinno zawsze wynikać, o jaką dziedzinę funkcji chodzi. Gdy oznaczamy zmieniąną przez n , zakładamy niejawnie, że w wyrażeniu $O(f(n))$ chodzi o zachowanie funkcji dla dużych całkowitych wartości n . Gdy oznaczamy zmienną przez z , zakładamy niejawnie, że w wyrażeniu $O(f(z))$ chodzi o zachowanie funkcji dla małych zespolonych wartości z .

Przypuśćmy, że $g(z)$ jest funkcją zadaną szeregiem potęgowym

$$g(z) = \sum_{k \geq 0} a_k z^k$$

zbieżnym dla $z = z_0$. Wówczas suma wartości bezwzględnych $\sum_{k \geq 0} |a_k z^k|$ także jest zbieżna, jeśli tylko $|z| < |z_0|$. Gdy $z_0 \neq 0$, możemy napisać

$$g(z) = a_0 + a_1 z + \dots + a_m z^m + O(z^{m+1}). \quad (11)$$

Z uwagi na fakt, że $g(z) = a_0 + a_1 z + \cdots + a_m z^m + z^{m+1}(a_{m+1} + a_{m+2} z + \cdots)$, wystarczy pokazać, że wartość w nawiasach jest ograniczona, gdy $|z| \leq r$ dla pewnego dodatniego r . Łatwo dostrzec, że $|a_{m+1}| + |a_{m+2}| r + |a_{m+3}| r^2 + \cdots$ jest ograniczeniem górnym, jeśli tylko $|z| \leq r < |z_0|$.

Z funkcji tworzących zestawionych w punkcie 1.2.9 otrzymujemy wiele ważnych wzorów asymptotycznych dla małych wartości z , m.in.

$$e^z = 1 + z + \frac{1}{2!} z^2 + \cdots + \frac{1}{m!} z^m + O(z^{m+1}), \quad (12)$$

$$\ln(1+z) = z - \frac{1}{2} z^2 + \cdots + \frac{(-1)^{m+1}}{m} z^m + O(z^{m+1}), \quad (13)$$

$$(1+z)^\alpha = 1 + \alpha z + \binom{\alpha}{2} z^2 + \cdots + \binom{\alpha}{m} z^m + O(z^{m+1}), \quad (14)$$

$$\frac{1}{1-z} \ln \frac{1}{1-z} = z + H_2 z^2 + \cdots + H_m z^m + O(z^{m+1}), \quad (15)$$

dla wszystkich nieujemnych liczb całkowitych m . Zauważmy, że stałe M i r ukryte w każdym O są ze sobą powiązane. Na przykład funkcja e^z jest oczywiście $O(1)$ dla $|z| \leq r$ i dowolnego ustalonego r , bo $|e^z| \leq e^{|z|}$, ale nie istnieje taka stała M , że $|e^z| \leq M$ dla wszystkich wartości z . Stąd musimy brać coraz większe ograniczenia M dla coraz większych r .

Czasami szereg asymptotyczny jest poprawny, chociaż nie odpowiada żadnemu zbieżnemu szeregowi nieskończonemu. Na przykład proste wzory na przedstawienie potęg kroczących (przypadających i ubywających) za pomocą zwykłych potęg

$$n^{\bar{r}} = \sum_{k=0}^m \begin{bmatrix} \bar{r} \\ r-k \end{bmatrix} n^{r-k} + O(n^{r-m-1}), \quad (16)$$

$$n^r = \sum_{k=0}^m (-1)^k \begin{bmatrix} r \\ r-k \end{bmatrix} n^{r-k} + O(n^{r-m-1}), \quad (17)$$

są asymptotycznie poprawne dla dowolnego rzeczywistego r i ustalonej liczby całkowitej $m \geq 0$, chociaż suma

$$\sum_{k=0}^{\infty} \begin{bmatrix} 1/2 \\ 1/2-k \end{bmatrix} n^{1/2-k}$$

jest rozbieżna dla wszystkich n (zobacz ćwiczenie 12). Oczywiście, gdy r jest nieujemną liczbą całkowitą, $n^{\bar{r}}$ i n^r są po prostu wielomianami stopnia r , a wzór (17) jest dokładnie taki sam, jak 1.2.6–(44). Gdy r jest ujemną liczbą całkowitą i $|n| > |r|$, suma nieskończona $\sum_{k=0}^{\infty} \begin{bmatrix} r \\ r-k \end{bmatrix} n^{r-k}$ nie zbiega do $n^{\bar{r}} = 1/(n-1)^{-r}$. Korzystając z 1.2.6–(58), można tę sumę zapisać w bardziej naturalnej postaci $\sum_{k=0}^{\infty} \begin{Bmatrix} k-r \\ -r \end{Bmatrix} n^{r-k}$.

Rozważmy prosty przykład ilustrujący wprowadzone pojęcia. Przyjrzyjmy się wielkości $\sqrt[n]{n}$ dla dużych n . Pierwiastkowanie n -tego stopnia powoduje zmniejszenie wartości, ale nie widać na pierwszy rzut oka, czy $\sqrt[n]{n}$ maleje czy rośnie.

Okazuje się, że $\sqrt[n]{n}$ maleje do jedynki. Rozważmy nieco bardziej skomplikowaną wielkość $n(\sqrt[n]{n} - 1)$. Teraz $(\sqrt[n]{n} - 1)$ maleje przy rosnącym n . A jak zachowuje się $n(\sqrt[n]{n} - 1)$?

Problem można rozwiązać za pomocą zaprezentowanych wzorów. Mamy

$$\sqrt[n]{n} = e^{\ln n/n} = 1 + (\ln n/n) + O((\ln n/n)^2), \quad (18)$$

ponieważ $\ln n/n \rightarrow 0$ przy $n \rightarrow \infty$ (zobacz ćwiczenia 8 i 11). Ta równość dowodzi naszego wcześniejszego przekonania, że $\sqrt[n]{n} \rightarrow 1$. Co więcej, mówi nam, że

$$n(\sqrt[n]{n} - 1) = n(\ln n/n + O((\ln n/n)^2)) = \ln n + O((\ln n)^2/n). \quad (19)$$

Innymi słowy, wielkość $n(\sqrt[n]{n} - 1)$ jest w przybliżeniu równa $\ln n$; różnica wynosi $O((\ln n)^2/n)$ i maleje do zera przy n dążącym do nieskończoności.

Wiele osób nadużywa notacji wielkie- O , zakładając mylnie, że ta notacja *dokładnie* określa szybkość wzrostu funkcji, że oznacza zarówno górne, jak i dolne oszacowanie. Ktoś może na przykład powiedzieć, że algorytm sortujący n liczb uważa się za niewydajny, „ponieważ jego czas działania jest równy $O(n^2)$ ”. Ale czas działania $O(n^2)$ nie musi oznaczać, że algorytm nie działa w czasie $O(n)$. Dla ograniczeń dolnych istnieje inna notacja – wielkie- Ω . Stwierdzenie

$$g(n) = \Omega(f(n)) \quad (20)$$

oznacza, że istnieją dodatnie stałe L i n_0 , takie że

$$|g(n)| \geq L|f(n)| \quad \text{dla dowolnego } n \geq n_0.$$

Korzystając z tej notacji, możemy poprawnie wnioskować, że algorytm sortowania o czasie działania $\Omega(n^2)$ nie będzie tak wydajny, jak ten o czasie działania $O(n \log n)$, jeżeli n jest odpowiednio duże. Nie znając jednak stałych ukrytych w O i Ω , nie możemy powiedzieć, jak duże musi być n , by metoda o czasie działania $O(n \log n)$ okazała się lepsza.

Wreszcie, jeśli chcemy opisać dokładną szybkość wzrostu, nie dbając o czynniki stałe, możemy skorzystać z notacji wielkie- Θ .

$$g(n) = \Theta(f(n)) \iff g(n) = O(f(n)) \text{ i } g(n) = \Omega(f(n)). \quad (21)$$

ĆWICZENIA

1. [HM01] Ile wynosi $\lim_{n \rightarrow \infty} O(n^{-1/3})$?
- ▶ 2. [M10] Prof. dr hab. I. Tyzrob uzyskał zaskakujący wynik, korzystając z „oczywistego” wzoru $O(f(n)) - O(f(n)) = 0$. Gdzie popełnił błąd i jaka powinna być prawa strona równości?
 - 3. [M15] Pominóż $(\ln n + \gamma + O(1/n))$ przez $(n + O(\sqrt{n}))$ i przedstaw odpowiedź za pomocą notacji wielkie- O .
 - ▶ 4. [M15] Podaj rozwinięcie asymptotyczne $n(\sqrt[n]{a} - 1)$ dla $a > 0$ z dokładnością do $O(1/n^3)$.
 - 5. [M20] Udowodnij lub obal: $O(f(n) + g(n)) = f(n) + O(g(n))$, gdy $f(n)$ i $g(n)$ są dodatnie dla wszystkich n . (Porównaj ze wzorem (10)).

- 6. [M20] Co jest nie tak z następującym rozumowaniem: „Mamy $n = O(n)$, $2n = O(n)$, …, zatem

$$\sum_{k=1}^n kn = \sum_{k=1}^n O(n) = O(n^2)?$$

7. [HM15] Udowodnij, że jeśli m jest dowolną liczbą całkowitą, to nie istnieje takie M , że $e^x \leq Mx^m$ dla dowolnie dużych x .

8. [HM20] Udowodnij, że $(\ln n)^m/n \rightarrow 0$ przy $n \rightarrow \infty$.

9. [HM20] Pokaż, że $e^{O(z^m)} = 1 + O(z^m)$, dla wszystkich ustalonych $m \geq 0$.

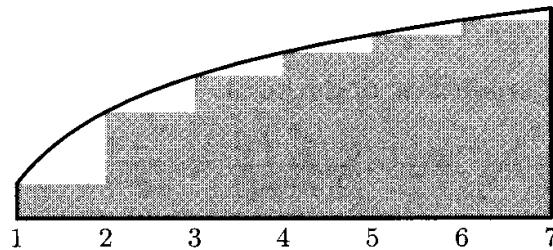
10. [HM22] Sformułuj twierdzenie podobne do twierdzenia z ćwiczenia 9, mówiące o $\ln(1 + O(z^m))$.

- 11. [M11] Wyjaśnij, dlaczego prawdziwe jest równanie (18).

12. [HM25] Udowodnij, że $\left[\frac{1}{1/2-k}\right] n^{-k}$ nie jest zbieżne do zera przy $k \rightarrow \infty$ dla żadnego całkowitego n . Skorzystaj z faktu, że $\left[\frac{1}{1/2-k}\right] = (-\frac{1}{2})^k [z^k] (ze^z/(e^z - 1))^{1/2}$.

- 13. [M10] Udowodnij lub obal, że $g(n) = \Omega(f(n))$ wtedy i tylko wtedy, gdy $f(n) = O(g(n))$.

***1.2.11.2. Wzór sumacyjny Eulera.** Jeden z najwygodniejszych sposobów otrzymywania dobrych przybliżeń wartości sum pochodzi od Leonarda Eulera. Jego metoda polega na przybliżaniu skończonej sumy za pomocą całki i w wielu przypadkach pozwala na uzyskiwanie nie tylko jednego przybliżenia, ale ciągu coraz lepszych przybliżeń. [*Commentarii Academiæ Scientiarum Petropolitanæ* 6 (1732), 68–97].



Rys. 12. Porównanie sumy z całką.

Na rysunku 12 jest pokazane porównanie $\int_1^n f(x) dx$ oraz $\sum_{k=1}^{n-1} f(k)$ dla $n = 7$. Strategia Eulera prowadzi do poręcznego wzoru na różnicę tych dwóch wartości, dającego się zastosować, gdy funkcja $f(x)$ jest różniczkowalna.

Dla wygody będziemy używać oznaczenia

$$\{x\} = x \bmod 1 = x - \lfloor x \rfloor. \quad (1)$$

Wyprowadzenie rozpoczynamy od następującej tożsamości:

$$\begin{aligned} \int_k^{k+1} (\{x\} - \frac{1}{2}) f'(x) dx &= (x - k - \frac{1}{2}) f(x) \Big|_k^{k+1} - \int_k^{k+1} f(x) dx \\ &= \frac{1}{2}(f(k+1) + f(k)) - \int_k^{k+1} f(x) dx. \end{aligned} \quad (2)$$

(To wynika z całkowania przez części). Sumując stronami równania tej postaci dla $1 \leq k < n$, dostajemy

$$\int_1^n (\{x\} - \frac{1}{2}) f'(x) dx = \sum_{1 \leq k < n} f(k) + \frac{1}{2}(f(n) - f(1)) - \int_1^n f(x) dx;$$

to jest

$$\sum_{1 \leq k < n} f(k) = \int_1^n f(x) dx - \frac{1}{2}(f(n) - f(1)) + \int_1^n B_1(\{x\}) f'(x) dx, \quad (3)$$

gdzie $B_1(x)$ jest wielomianem $x - \frac{1}{2}$. To właśnie jest poszukiwany związek między sumą a całką.

Można uzyskać dokładniejsze przybliżenie, całkując przez części. Jednak zanim się za to zabierzemy, powinniśmy przyjrzeć się liczbom Bernoulliego, będącym współczynnikami szeregu

$$\frac{z}{e^z - 1} = B_0 + B_1 z + \frac{B_2 z^2}{2!} + \dots = \sum_{k \geq 0} \frac{B_k z^k}{k!}. \quad (4)$$

Współczynniki te, pojawiające się znienacka w wielu miejscach, zostały wprowadzone przez Jacquesa Bernoulliego w jego *Ars Conjectandi*, opublikowanej pośmiertnie w 1713 roku. Co ciekawe, niemal w tym samym czasie zostały odkryte w Japonii przez Takakazu Seki i opublikowane w 1712 roku, niedługo po jego śmierci. [Zobacz *Takakazu Seki's Collected Works* (Osaka: 1974), 39–42].

Mamy

$$B_0 = 1, \quad B_1 = -\frac{1}{2}, \quad B_2 = \frac{1}{6}, \quad B_3 = 0, \quad B_4 = -\frac{1}{30}; \quad (5)$$

dalsze wartości można znaleźć w dodatku A. Ponieważ

$$\frac{z}{e^z - 1} + \frac{z}{2} = \frac{z}{2} \frac{e^z + 1}{e^z - 1} = -\frac{z}{2} \frac{e^{-z} + 1}{e^{-z} - 1}$$

jest funkcją parzystą, widać, że

$$B_3 = B_5 = B_7 = B_9 = \dots = 0. \quad (6)$$

Gdy w definicji (4) pomnożymy obie strony równości przez $e^z - 1$ i przyrównamy współczynniki szeregów potęgowych zmiennej z , otrzymamy wzór

$$\sum_k \binom{n}{k} B_k = B_n + \delta_{n1}. \quad (7)$$

(Zobacz ćwiczenie 1). Zdefiniujemy teraz *wielomian Bernoulliego*

$$B_m(x) = \sum_k \binom{m}{k} B_k x^{m-k}. \quad (8)$$

Dla $m = 1$ mamy $B_1(x) = B_0 x + B_1 = x - \frac{1}{2}$, co odpowiada wielomianowi użytemu we wzorze (3). Dla $m > 1$ na mocy (7) mamy $B_m(1) = B_m = B_m(0)$. Innymi słowy, $B_m(\{x\})$ nie ma nieciągłości w punktach całkowitych x .

Znaczenie wielomianów i liczb Bernoulliego dla naszego problemu wnet się wyjaśni. Różniczkując wzór (8), otrzymujemy

$$\begin{aligned} B'_m(x) &= \sum_k \binom{m}{k} (m-k) B_k x^{m-k-1} \\ &= m \sum_k \binom{m-1}{k} B_k x^{m-1-k} \\ &= m B_{m-1}(x) \end{aligned} \quad (9)$$

i dla $m \geq 1$ możemy całkować przez części w następujący sposób

$$\begin{aligned} \frac{1}{m!} \int_1^n B_m(\{x\}) f^{(m)}(x) dx &= \frac{1}{(m+1)!} (B_{m+1}(1) f^{(m)}(n) - B_{m+1}(0) f^{(m)}(1)) \\ &\quad - \frac{1}{(m+1)!} \int_1^n B_{m+1}(\{x\}) f^{(m+1)}(x) dx. \end{aligned}$$

Ten wynik pozwala dalej poprawiać przybliżenie (3), dzięki czemu, korzystając z (6), dochodzimy do ogólnego wzoru Eulera:

$$\begin{aligned} \sum_{1 \leq k < n} f(k) &= \int_1^n f(x) dx - \frac{1}{2} (f(n) - f(1)) + \frac{B_2}{2!} (f'(n) - f'(1)) + \dots \\ &\quad + \frac{(-1)^m B_m}{m!} (f^{(m-1)}(n) - f^{(m-1)}(1)) + R_{mn} \\ &= \int_1^n f(x) dx + \sum_{k=1}^m \frac{B_k}{k!} (f^{(k-1)}(n) - f^{(k-1)}(1)) + R_{mn}, \end{aligned} \quad (10)$$

gdzie

$$R_{mn} = \frac{(-1)^{m+1}}{m!} \int_1^n B_m(\{x\}) f^{(m)}(x) dx. \quad (11)$$

Reszta R_{mn} będzie mała dla bardzo małych $B_m(\{x\}) f^{(m)}(x)/m!$. W istocie można pokazać, że

$$\left| \frac{B_m(\{x\})}{m!} \right| \leq \frac{|B_m|}{m!} < \frac{4}{(2\pi)^m} \quad (12)$$

dla parzystych m . [Zobacz CMath, §9.5]. Jednak często okazuje się, że rząd wielkości $f^{(m)}(x)$ staje się duży dla rosnących m , istnieje zatem „najlepsza” wartość m , dla której $|R_{mn}|$ ma najmniejszą wartość przy ustalonym n .

Wiadomo, że dla parzystych m istnieje taka liczba θ , że

$$R_{mn} = \theta \frac{B_{m+2}}{(m+2)!} (f^{(m+1)}(n) - f^{(m+1)}(1)), \quad 0 < \theta < 1, \quad (13)$$

przy założeniu, że $f^{(m+2)}(x) f^{(m+4)}(x) > 0$ dla $1 < x < n$. Zatem w takim przypadku reszta ma ten sam znak, co pierwszy pomijany wyraz, i jest od niego mniejsza. Prostą wersję tego twierdzenia udowadniamy w ćwiczeniu 3.

Zastosujmy teraz wzór Eulera do kilku przykładów. Najpierw zajmijmy się $f(x) = 1/x$. Pochodne mają postać $f^{(m)} = (-1)^m m! / x^{m+1}$, zatem na mocy (10)

$$H_{n-1} = \ln n + \sum_{k=1}^m \frac{B_k}{k} (-1)^{k-1} \left(\frac{1}{n^k} - 1 \right) + R_{mn}. \quad (14)$$

Mamy zatem

$$\gamma = \lim_{n \rightarrow \infty} (H_{n-1} - \ln n) = \sum_{k=1}^m \frac{B_k}{k} (-1)^k + \lim_{n \rightarrow \infty} R_{mn}. \quad (15)$$

Z faktu, że istnieje $\lim_{n \rightarrow \infty} R_{mn} = \pm \int_1^\infty B_m(\{x\}) dx / x^{m+1}$, wnioskujemy, iż rzeczywiście istnieje stała γ . Możemy zatem zestawić ze sobą wzory (14) i (15), otrzymując ogólne przybliżenie dla liczb harmonicznych:

$$\begin{aligned} H_{n-1} &= \ln n + \gamma + \sum_{k=1}^m \frac{(-1)^{k-1} B_k}{kn^k} + \int_n^\infty \frac{B_m(\{x\}) dx}{x^{m+1}} \\ &= \ln n + \gamma + \sum_{k=1}^{m-1} \frac{(-1)^{k-1} B_k}{kn^k} + O\left(\frac{1}{n^m}\right). \end{aligned}$$

Zamieniając m na $m+1$, otrzymujemy

$$H_{n-1} = \ln n + \gamma + \sum_{k=1}^m \frac{(-1)^{k-1} B_k}{kn^k} + O\left(\frac{1}{n^{m+1}}\right). \quad (16)$$

Nie koniec na tym. Ze wzoru (13) widać, że błąd przybliżenia jest mniejszy niż pierwszy pomijany wyraz. W szczególności mamy (po dodaniu $1/n$ do obu stron)

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \epsilon, \quad 0 < \epsilon < \frac{B_6}{6n^6} = \frac{1}{252n^6}.$$

To jest równanie 1.2.7-(3). Liczby Bernoulliego B_k dla dużych k stają się bardzo duże (w przybliżeniu $(-1)^{1+k/2} 2(k!/(2\pi)^k)$ dla parzystych k), zatem dla żadnej wartości n nie można rozszerzyć (16) do zbieżnego szeregu nieskończonego.

Za pomocą tej samej metody możemy wyprowadzić przybliżenie Stirlinga. Tym razem mamy $f(x) = \ln x$, a ze wzoru (10) otrzymujemy

$$\ln(n-1)! = n \ln n - n + 1 - \frac{1}{2} \ln n + \sum_{1 < k \leq m} \frac{B_k (-1)^k}{k(k-1)} \left(\frac{1}{n^{k-1}} - 1 \right) + R_{mn}. \quad (17)$$

Postępując jak poprzednio, stwierdzamy istnienie granicy

$$\lim_{n \rightarrow \infty} (\ln n! - n \ln n + n - \frac{1}{2} \ln n) = 1 + \sum_{1 < k \leq m} \frac{B_k (-1)^{k+1}}{k(k-1)} + \lim_{n \rightarrow \infty} R_{mn}.$$

Nazwijmy ją roboczo σ („stała Stirlinga”). Otrzymujemy wynik uzyskany przez Stirlinga

$$\ln n! = (n + \frac{1}{2}) \ln n - n + \sigma + \sum_{1 < k \leq m} \frac{B_k (-1)^k}{k(k-1)n^{k-1}} + O\left(\frac{1}{n^m}\right). \quad (18)$$

W szczególności dla $m = 5$ mamy

$$\ln n! = \left(n + \frac{1}{2}\right) \ln n - n + \sigma + \frac{1}{12n} - \frac{1}{360n^3} + O\left(\frac{1}{n^5}\right).$$

Teraz przekształcamy obie strony, stosując funkcję wykładniczą:

$$n! = e^\sigma \sqrt{n} \left(\frac{n}{e}\right)^n \exp\left(\frac{1}{12n} - \frac{1}{360n^3} + O\left(\frac{1}{n^5}\right)\right).$$

Korzystając z faktu, że $e^\sigma = \sqrt{2\pi}$ (zobacz ćwiczenie 5), i rozwijając wykładnik, otrzymujemy ostateczny wynik:

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \frac{1}{12n} + \frac{1}{288n^2} - \frac{139}{51840n^3} - \frac{571}{2488320n^4} + O\left(\frac{1}{n^5}\right)\right). \quad (19)$$

ĆWICZENIA

1. [M18] Udowodnij wzór (7).
2. [HM20] Zauważ, że wzór (9) wynika z (8) dla dowolnego ciągu B_n , niekoniecznie dla ciągu zdefiniowanego przez (4). Wyjaśnij, dlaczego akurat taki ciąg jest potrzebny, by zagwarantować poprawność wzoru (10).
3. [HM20] Niech $C_{mn} = ((-1)^m B_m / m!) (f^{(m-1)}(n) - f^{(m-1)}(1))$ będzie m -tą poprawką we wzorze sumacyjnym Eulera. Zakładając, że $f^{(m)}(x)$ ma stały znak dla $1 \leq x \leq n$, udowodnij, że $|R_{mn}| \leq |C_{mn}|$ dla $m = 2k > 0$. Innymi słowy, pokaż, że reszta nie przekracza co do wartości bezwzględnej ostatniego branego pod uwagę wyrazu.
- ▶ 4. [HM20] (Sumy potęg) Dla $f(x) = x^m$ pochodne wysokich rzędów są zerowe, zatem wzór sumacyjny Eulera daje dokładną wartość sumy

$$S_m(n) = \sum_{0 \leq k < n} k^m$$

przedstawioną za pomocą liczb Bernoulliego. (To właśnie badania nad $S_m(n)$ dla $m = 1, 2, 3, \dots$ doprowadziły Bernoulliego i Seki do odkrycia tych liczb). Wyraź $S_m(n)$ za pomocą wielomianów Bernoulliego. Sprawdź swoją odpowiedź dla $m = 0, 1$ i 2 . (Zauważ, że w $S_m(n)$ sumujemy względem $0 \leq k < n$, a nie $1 \leq k < n$; wzór sumacyjny Eulera można zastosować po odpowiednim zastąpieniu zera jedynką).

5. [HM30] Wiedząc, że
- $$n! = \kappa \sqrt{n} \left(\frac{n}{e}\right)^n \left(1 + O\left(\frac{1}{n}\right)\right),$$
- pokaż, że $\kappa = \sqrt{2\pi}$. Skorzystaj z iloczynu Wallisa (ćwiczenie 1.2.5–18). [Wskazówka: Zastanów się nad $\binom{2n}{n}$ dla dużych n].
- ▶ 6. [HM30] Pokaż, że przybliżenie Stirlinga jest prawdziwe również dla niecałkowitych n :
- $$\Gamma(x+1) = \sqrt{2\pi x} \left(\frac{x}{e}\right)^x \left(1 + O\left(\frac{1}{x}\right)\right), \quad x \geq a > 0.$$
- [Wskazówka: Podstaw $f(x) = \ln(x+c)$ we wzorze sumacyjnym Eulera i zastosuj definicję $\Gamma(x)$ podaną w punkcie 1.2.5].

- 7. [HM32] Jaka jest przybliżona wartość $1^1 2^2 3^3 \dots n^n$?
- 8. [M23] Znajdź asymptotyczną wartość wyrażenia $\ln(an^2 + bn)!$ z błędem bezwzględnym $O(n^{-2})$. Użyj jej do wyznaczenia asymptotycznej wartości $\binom{cn^2}{n} / c^n \binom{n^2}{n}$ z błędem względnym $O(n^{-2})$, dla dodatniej stałej c . *Błąd bezwzględny* ϵ oznacza, że $(\text{ideal}) = (\text{przybliżenie}) + \epsilon$; *błąd względny* ϵ oznacza, że $(\text{ideal}) = (\text{przybliżenie})(1 + \epsilon)$.
- 9. [M25] Znajdź asymptotyczną wartość $\binom{2n}{n}$ z błędem względnym $O(n^{-3})$ dwoma sposobami: (a) za pomocą przybliżenia Stirlinga; (b) korzystając z ćwiczenia 1.2.6–47 i wzoru 1.2.11.1–(16).

***1.2.11.3. Kilka obliczeń asymptotycznych.** W tym punkcie zajmiemy się badaniem trzech zaskakujących sum i wyznaczmy ich przybliżone wartości. Oto i sumy:

$$P(n) = 1 + \frac{n-1}{n} + \frac{n-2}{n} \frac{n-2}{n-1} + \dots = \sum_{k=0}^n \frac{(n-k)^k (n-k)!}{n!}, \quad (1)$$

$$Q(n) = 1 + \frac{n-1}{n} + \frac{n-1}{n} \frac{n-2}{n} + \dots = \sum_{k=1}^n \frac{n!}{(n-k)! n^k}, \quad (2)$$

$$R(n) = 1 + \frac{n}{n+1} + \frac{n}{n+1} \frac{n}{n+2} + \dots = \sum_{k \geq 0} \frac{n! n^k}{(n+k)!}. \quad (3)$$

Sumy, choć podobne na pierwszy rzut oka, okazują się istotnie różne. Wszystkie trzy pojawiają się w algorytmach, które przedżej czy później będą przedmiotem naszego zainteresowania. Zarówno $P(n)$, jak i $Q(n)$ są sumami nieskończonymi, podczas gdy $R(n)$ jest sumą skończoną. Wydaje się, że dla dużych n wszystkie trzy sumy będą prawie równe, chociaż nie jest oczywiste, jaka będzie asymptotyczna wartość *którejkolwiek* z nich. Poszukując ich przybliżeń, natkniemy się na wielce pouczające odkrycia. (Czytelnik być może w tym momencie zechce przerwać lekturę i samemu zmierzyć się z sumami).

Po pierwsze, zauważmy istotny związek między $Q(n)$ i $R(n)$:

$$\begin{aligned} Q(n) + R(n) &= \frac{n!}{n^n} \left(\left(1 + n + \dots + \frac{n^{n-1}}{(n-1)!} \right) + \left(\frac{n^n}{n!} + \frac{n^{n+1}}{(n+1)!} + \dots \right) \right) \\ &= \frac{n! e^n}{n^n}. \end{aligned} \quad (4)$$

Ze wzoru Stirlinga wiemy, że $n! e^n / n^n$ wynosi w przybliżeniu $\sqrt{2\pi n}$. Zatem zgadujemy, że $Q(n)$ i $R(n)$ okażą się z grubsza równe $\sqrt{\pi n / 2}$.

By uczynić następny krok, musimy przyjrzeć się sumom częściowym szeregu e^n . Ze wzoru Taylora w postaci z resztą

$$f(x) = f(0) + f'(0)x + \dots + \frac{f^{(n)}(0)x^n}{n!} + \int_0^x \frac{t^n}{n!} f^{(n+1)}(x-t) dt, \quad (5)$$

szybko dochodzimy do ważnej funkcji znanej jako *niepełna funkcja gamma*:

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt. \quad (6)$$

Będziemy zakładać, że $a > 0$. Z ćwiczenia 1.2.5–20 mamy $\gamma(a, \infty) = \Gamma(a)$, skąd właściwie pochodzi nazwa „niepełna funkcja gamma”. Ta funkcja ma dwa istotne rozwinięcia w szereg potęg x (zobacz ćwiczenia 2 i 3):

$$\gamma(a, x) = \frac{x^a}{a} - \frac{x^{a+1}}{a+1} + \frac{x^{a+2}}{2!(a+2)} - \dots = \sum_{k \geq 0} \frac{(-1)^k x^{k+a}}{k! (k+a)}, \quad (7)$$

$$e^x \gamma(a, x) = \frac{x^a}{a} + \frac{x^{a+1}}{a(a+1)} + \frac{x^{a+2}}{a(a+1)(a+2)} + \dots = \sum_{k \geq 0} \frac{x^{k+a}}{a(a+1)\dots(a+k)}. \quad (8)$$

Drugi wzór pozwala dostrzec związek z $R(n)$:

$$R(n) = \frac{n! e^n}{n^n} \left(\frac{\gamma(n, n)}{(n-1)!} \right). \quad (9)$$

Specjalnie napisaliśmy tę równość w nieco bardziej skomplikowanej postaci, gdyż $\gamma(n, n)$ jest częścią $\gamma(n, \infty) = \Gamma(n) = (n-1)!$, a $n! e^n / n^n$ jest wielkością ze wzoru (4).

Problem sprowadza się do wyznaczenia dobrego oszacowania $\gamma(n, n)/(n-1)!$. Wyznaczmy teraz przybliżoną wartość $\gamma(x+1, x+y)/\Gamma(x+1)$ dla ustalonego y i dużych x . Przedstawiana metoda jest ważniejsza niż sam wynik. Czytelnik powinien z uwagą przestudiować poniższe wyprowadzenie.

Z definicji mamy

$$\begin{aligned} \frac{\gamma(x+1, x+y)}{\Gamma(x+1)} &= \frac{1}{\Gamma(x+1)} \int_0^{x+y} e^{-t} t^x dt \\ &= 1 - \frac{1}{\Gamma(x+1)} \int_x^\infty e^{-t} t^x dt + \frac{1}{\Gamma(x+1)} \int_x^{x+y} e^{-t} t^x dt. \end{aligned} \quad (10)$$

Przyjmijmy

$$I_1 = \int_x^\infty e^{-t} t^x dt, \quad I_2 = \int_x^{x+y} e^{-t} t^x dt,$$

i rozważmy każdą z całek oddziennie.

Oszacowanie I_1 : Przekształcamy I_1 w całkę od 0 do nieskończoności, podstawiając $t = x(1+u)$. Podstawiamy dalej $v = u - \ln(1+u)$, $dv = (1 - 1/(1+u)) du$, co jest dozwolone, bo v jest monotoniczną funkcją u :

$$I_1 = e^{-x} x^x \int_0^\infty x e^{-xu} (1+u)^x du = e^{-x} x^x \int_0^\infty x e^{-xv} \left(1 + \frac{1}{u}\right) dv. \quad (11)$$

W ostatniej całce zastępujemy $1 + 1/u$ szeregiem potęgowym zmiennej v . Mamy

$$v = \frac{1}{2}u^2 - \frac{1}{3}u^3 + \frac{1}{4}u^4 - \frac{1}{5}u^5 + \dots = (u^2/2)(1 - \frac{2}{3}u + \frac{1}{2}u^2 - \frac{2}{5}u^3 + \dots).$$

Kładąc $w = \sqrt{2v}$, otrzymujemy

$$w = u(1 - \frac{2}{3}u + \frac{1}{2}u^2 - \frac{2}{5}u^3 + \dots)^{1/2} = u - \frac{1}{3}u^2 + \frac{7}{36}u^3 - \frac{73}{540}u^4 + \frac{1331}{12960}u^5 + O(u^6).$$

(To rozwinięcie można otrzymać, stosując twierdzenie dwumianowe. Wydajne metody stosowania przekształceń szeregów potęgowych, z których poniżej korzystamy, są szczegółowo przedstawione w podrozdziale 4.7). Możemy teraz wyznaczyć rozwiązanie dla niewiadomej u w postaci szeregu potęgowego zmiennej w :

$$\begin{aligned} u &= w + \frac{1}{3}w^2 + \frac{1}{36}w^3 - \frac{1}{270}w^4 + \frac{1}{4320}w^5 + O(w^6); \\ 1 + \frac{1}{u} &= 1 + \frac{1}{w} - \frac{1}{3} + \frac{1}{12}w - \frac{2}{135}w^2 + \frac{1}{864}w^3 + O(w^4) \\ &= \frac{1}{\sqrt{2}}v^{-1/2} + \frac{2}{3} + \frac{\sqrt{2}}{12}v^{1/2} - \frac{4}{135}v + \frac{\sqrt{2}}{432}v^{3/2} + O(v^2). \end{aligned} \quad (12)$$

We wszystkich tych wzorach O dotyczy małych wartości argumentu, czyli $|u| \leq r$, $|v| \leq r$, $|w| \leq r$ dla odpowiednio małych dodatnich r . Czy to wystarczy? Podstawienie szeregu zmiennej v za $1+1/u$ we wzorze (11) powinno być możliwe dla $0 \leq v < \infty$, a nie tylko dla $|v| \leq r$. Na szczęście okazuje się, że wartości całki od 0 do ∞ zależą prawie wyłącznie od wartości funkcji podcałkowej w pobliżu zera. W istocie (zobacz ćwiczenie 4)

$$\int_r^\infty xe^{-xv} \left(1 + \frac{1}{u}\right) dv = O(e^{-rx}) \quad (13)$$

dla dowolnego ustalonego $r > 0$ i dla dużych x . Interesuje nas przybliżenie z dokładnością do wyrazów $O(x^{-m})$, a ponieważ $O((1/e^r)^x)$ jest dużo mniejsze niż $O(x^{-m})$ dla dowolnych dodatnich r i m , całkować musimy jedynie od 0 do r dla dowolnego ustalonego dodatniego r . Bierzemy zatem r odpowiednio małe, tak żeby powyższe przekształcenia szeregów potęgowych były dopuszczalne. (Zobacz 1.2.11.1–(11) i 1.2.11.3–(13)).

Teraz

$$\int_0^\infty xe^{-xv} v^\alpha dv = \frac{1}{x^\alpha} \int_0^\infty e^{-q} q^\alpha dq = \frac{1}{x^\alpha} \Gamma(\alpha + 1) \quad \text{dla } \alpha > -1; \quad (14)$$

zatem podstawiając szereg (12) do (11), otrzymujemy ostatecznie

$$I_1 = e^{-x} x^x \left(\sqrt{\frac{\pi}{2}} x^{1/2} + \frac{2}{3} + \frac{\sqrt{2\pi}}{24} x^{-1/2} - \frac{4}{135} x^{-1} + \frac{\sqrt{2\pi}}{576} x^{-3/2} + O(x^{-2}) \right). \quad (15)$$

Oszacowanie I_2 : W całce I_2 podstawiamy $t = u + x$, otrzymując

$$I_2 = e^{-x} x^x \int_0^y e^{-u} \left(1 + \frac{u}{x}\right)^x du. \quad (16)$$

Teraz

$$\begin{aligned} e^{-u} \left(1 + \frac{u}{x}\right)^x &= \exp\left(-u + x \ln\left(1 + \frac{u}{x}\right)\right) = \exp\left(\frac{-u^2}{2x} + \frac{u^3}{3x^2} + O(x^{-3})\right) \\ &= 1 - \frac{u^2}{2x} + \frac{u^4}{8x^2} + \frac{u^3}{3x^2} + O(x^{-3}) \end{aligned}$$

dla $0 \leq u \leq y$ i dużych x . Dostajemy stąd

$$I_2 = e^{-x} x^x \left(y - \frac{y^3}{6} x^{-1} + \left(\frac{y^4}{12} + \frac{y^5}{40}\right) x^{-2} + O(x^{-3})\right). \quad (17)$$

Na koniec analizujemy współczynniki $e^{-x} x^x / \Gamma(x+1)$ pojawiające się po wymnożeniu wzorów (15) i (17) przez czynnik $1/\Gamma(x+1)$ w (10). Z przybliżenia Stirlinga, które zgodnie z ćwiczeniem 1.2.11.2–6 ma zastosowanie także dla funkcji gamma, otrzymujemy

$$\begin{aligned} \frac{e^{-x} x^x}{\Gamma(x+1)} &= \frac{e^{-1/12x+O(x^{-3})}}{\sqrt{2\pi x}} \\ &= \frac{1}{\sqrt{2\pi}} x^{-1/2} - \frac{1}{12\sqrt{2\pi}} x^{-3/2} + \frac{1}{288\sqrt{2\pi}} x^{-5/2} + O(x^{-7/2}). \quad (18) \end{aligned}$$

A teraz wielkie podsumowanie: z równań (10), (15), (17) i (18) wynika

Twierdzenie A. *Dla dużych wartości x i dla ustalonego y*

$$\begin{aligned} \frac{\gamma(x+1, x+y)}{\Gamma(x+1)} &= \frac{1}{2} + \left(\frac{y-2/3}{\sqrt{2\pi}}\right) x^{-1/2} + \frac{1}{\sqrt{2\pi}} \left(\frac{23}{270} - \frac{y}{12} - \frac{y^3}{6}\right) x^{-3/2} \\ &\quad + O(x^{-5/2}). \quad \blacksquare \quad (19) \end{aligned}$$

Z zaprezentowanej metody widać, w jaki sposób można rozszerzać to przybliżenie na dalsze potęgi x dowolnie daleko.

Twierdzenia A można użyć do wyznaczenia przybliżonych wartości $R(n)$ i $Q(n)$ za pomocą wzorów (4) i (9), ale odłożymy te obliczenia na później. Zabierzmy się teraz za $P(n)$, które wydaje się wymagać nieco innych metod. Mamy

$$P(n) = \sum_{k=0}^n \frac{k^{n-k} k!}{n!} = \frac{\sqrt{2\pi}}{n!} \sum_{k=0}^n k^{n+1/2} e^{-k} \left(1 + \frac{1}{12k} + O(k^{-2})\right). \quad (20)$$

Zatem, by uzyskać wartości $P(n)$, musimy zająć się sumami postaci

$$\sum_{k=0}^n k^{n+1/2} e^{-k}.$$

Niech $f(x) = x^{n+1/2} e^{-x}$ we wzorze sumacyjnym Eulera:

$$\sum_{k=0}^n k^{n+1/2} e^{-k} = \int_0^n x^{n+1/2} e^{-x} dx + \frac{1}{2} n^{n+1/2} e^{-n} + \frac{1}{24} n^{n-1/2} e^{-n} - R. \quad (21)$$

Ze zgrubnej analizy reszty tego szeregu (zobacz ćwiczenie 5) wynika, że $R = O(n^n e^{-n})$, a ponieważ całka jest niepełną funkcją gamma, mamy

$$\sum_{k=0}^n k^{n+1/2} e^{-k} = \gamma\left(n + \frac{3}{2}, n\right) + \frac{1}{2} n^{n+1/2} e^{-n} + O(n^n e^{-n}). \quad (22)$$

W naszym wzorze (20) trzeba jeszcze oszacować sumę

$$\sum_{k=0}^n k^{n-1/2} e^{-k} = \sum_{0 \leq k \leq n-1} k^{(n-1)+1/2} e^{-k} + n^{n-1/2} e^{-n},$$

co także można uzyskać, posługując się (22).

Mamy wreszcie wszystkie potrzebne wzory, by wyznaczyć przybliżone wartości $P(n)$, $Q(n)$ i $R(n)$. Cała reszta to podstawianie, mnożenie itp. Pośród tych przekształceń używamy rozwinięcia

$$(n + \alpha)^{n+\beta} = n^{n+\beta} e^\alpha \left(1 + \alpha \left(\beta - \frac{\alpha}{2} \right) \frac{1}{n} + O(n^{-2}) \right), \quad (23)$$

które udowadniamy w ćwiczeniu 6. Metoda (21) daje tylko dwa pierwsze wyrazy w szeregu asymptotycznym funkcji $P(n)$. Wyrazy kolejne możemy uzyskać, stosując pouczającą metodę opisaną w ćwiczeniu 14.

W wyniku mozołnych obliczeń otrzymujemy poszukiwane wzory asymptotyczne:

$$P(n) = \sqrt{\frac{\pi n}{2}} - \frac{2}{3} + \frac{11}{24} \sqrt{\frac{\pi}{2n}} + \frac{4}{135n} - \frac{71}{1152} \sqrt{\frac{\pi}{2n^3}} + O(n^{-2}), \quad (24)$$

$$Q(n) = \sqrt{\frac{\pi n}{2}} - \frac{1}{3} + \frac{1}{12} \sqrt{\frac{\pi}{2n}} - \frac{4}{135n} + \frac{1}{288} \sqrt{\frac{\pi}{2n^3}} + O(n^{-2}), \quad (25)$$

$$R(n) = \sqrt{\frac{\pi n}{2}} + \frac{1}{3} + \frac{1}{12} \sqrt{\frac{\pi}{2n}} + \frac{4}{135n} + \frac{1}{288} \sqrt{\frac{\pi}{2n^3}} + O(n^{-2}). \quad (26)$$

Analizowane funkcje w dostępnej literaturze potraktowane są po macoszemu. Pierwszy wyraz $\sqrt{\pi n/2}$ w rozwinięciu $P(n)$ został podany przez H. B. Demutha [Praca doktorska (Stanford University, October 1956), 67–68]. W 1963 roku autor, posługując się tym wynikiem, tabelą wartości $P(n)$ dla $n \leq 2000$ oraz dobrym suwakiem logarytmicznym, wyznaczył oszacowanie empiryczne $P(n) \approx \sqrt{\pi n/2} - 0.6667 + 0.575/\sqrt{n}$. Naturalne wydawało się postawienie hipotezy, że 0.6667 jest przybliżeniem $2/3$, a 0.575 prawdopodobnie przybliżeniem $\gamma = 0.57721\dots$ (odrobina optymizmu nie zaszkodzi). Później, kiedy powstawał niższy rozdział, znane było poprawne rozwinięcie $P(n)$, zatem hipoteza $2/3$ została sprawdzona. Dla drugiego współczynnika 0.575 mamy jednak nie γ , lecz $\frac{11}{24} \sqrt{\pi/2} \approx 0.5744$, co zgadza się i z teorią, i z oszacowaniami empirycznymi.

Wzory równoważne asymptotycznym wartośćom $Q(n)$ i $R(n)$ były po raz pierwszy wyznaczone przez genialnego hinduskiego matematyka-samouka S. Ramanujana, który przedstawił problem oszacowania $n! e^n / 2n^n - Q(n)$ w *J. Indian*

Math. Soc. **3** (1911), 128; **4** (1912), 151–152. W swoim rozwiązańiu podał on szereg asymptotyczny $\frac{1}{3} + \frac{4}{135}n^{-1} - \frac{8}{2835}n^{-2} - \frac{16}{8505}n^{-3} + \dots$, który znacznie wykracza poza wzór (25). Jego wyprowadzenie było trochę bardziej eleganckie od naszego. Szacując I_1 , Ramanujan podstawił $t = x + u\sqrt{2x}$ i wyraził funkcję podcałkową jako sumę wyrazów postaci $c_{jk} \int_0^\infty \exp(-u^2) u^j x^{-k/2} du$. Całkę I_2 można w ogóle pominać, gdyż $a\gamma(a, x) = x^a e^{-x} + \gamma(a+1, x)$ dla $a > 0$, zobacz wzór (8). Jeszcze prostsze podejście do asymptotyki $Q(n)$, zapewne najprostsze z możliwych, pojawia się w ćwiczeniu 20. Zaprezentowane wyprowadzanie, nadmiernie skomplikowane, ale pouczające, zawdzięczamy R. Furchowi [*Zeitschrift für Physik* **112** (1939), 92–95], który początkowo interesował się wyznaczeniem takiej wartości y , dla której $\gamma(x+1, x+y) = \Gamma(x+1)/2$. Właściwości asymptotyczne niepełnej funkcji gamma zostały później rozszerzone na argumenty zespolone przez F. G. Tricomiego [*Math. Zeitschrift* **53** (1950), 136–148]. Zobacz także N. M. Temme, *Math. Comp.* **29** (1975), 1109–1114; *SIAM J. Math. Anal.* **10** (1979), 757–766. H. W. Gould podaje bibliografię innych badań nad $Q(n)$ w *AMM* **75** (1968), 1019–1021.

W naszych wyprowadzeniach szeregów asymptotycznych dla $P(n)$, $Q(n)$ i $R(n)$ wykorzystaliśmy wyłącznie elementarną analizę matematyczną. Zauważmy, że dla każdej funkcji posłużyliśmy się innymi metodami! W istocie moglibyśmy rozwiązać wszystkie trzy problemy za pomocą metod z ćwiczenia 14, które są wyjaśnione szerzej w punktach 5.1.4 i 5.2.2. To byłoby bardziej eleganckie, ale mniej pouczające.

Czytelnicy zainteresowani tematem powinni szukać dodatkowych informacji w pięknej książce *Asymptotic Methods in Analysis* autorstwa N. G. de Bruijna (Amsterdam: North-Holland, 1961). Zobacz także nowszy przegląd: A. M. Odlyzko [*Handbook of Combinatorics* **2** (MIT Press, 1995), 1063–1229], zawierający 65 szczegółowych przykładów i dokładną bibliografię.

ĆWICZENIA

1. [HM20] Udowodnij wzór (5) przez indukcję względem n .
2. [HM20] Wyprowadź równanie (7) z (6).
3. [M20] Wyprowadź równanie (8) z (7).
- ▶ 4. [HM10] Udowodnij wzór (13).
5. [HM24] Pokaż, że R we wzorze (21) jest $O(n^n e^{-n})$.
- ▶ 6. [HM20] Udowodnij wzór (23).
- ▶ 7. [HM30] Wyznaczając wartość I_2 , rozważaliśmy $\int_0^y e^{-u} \left(1 + \frac{u}{x}\right)^x du$. Podaj reprezentację asymptotyczną całki

$$\int_0^{yx^{1/4}} e^{-u} \left(1 + \frac{u}{x}\right)^x du$$

do wyrazów rzędu $O(x^{-2})$ dla ustalonego y i dużych x .

8. [HM30] Zakładając, że $f(x) = O(x^r)$ przy $x \rightarrow \infty$ oraz $0 \leq r < 1$, pokaż, że

$$\int_0^{f(x)} e^{-u} \left(1 + \frac{u}{x}\right)^x du = \int_0^{f(x)} \exp\left(\frac{-u^2}{2x} + \frac{u^3}{3x^2} - \dots + \frac{(-1)^{m-1} u^m}{mx^{m-1}}\right) du + O(x^{-s})$$

gdy $m = \lceil (s+2r)/(1-r) \rceil$. [To w szczególności dowodzi wyniku Tricomiego: jeśli $f(x) = O(\sqrt{x})$, to

$$\int_0^{f(x)} e^{-u} \left(1 + \frac{u}{x}\right)^x du = \sqrt{2x} \int_0^{f(x)/\sqrt{2x}} e^{-t^2} dt + O(1).$$

► **9.** [HM36] Jak zachowuje się $\gamma(x+1, px)/\Gamma(x+1)$ dla dużych x ? (Wielkość p jest tutaj stałą rzeczywistą; dla $p < 0$ zakładamy, że x jest całkowite, tak że t^x jest dobrze określone dla ujemnych t). Wyznacz przynajmniej dwa wyrazy rozwinięcia asymptotycznego, zanim uciekniesz się do notacji wielkiej- O .

10. [HM34] Przy założeniach jak w poprzednim ćwiczeniu oraz $p \neq 1$ zapisz rozwinięcie asymptotyczne $\gamma(x+1, px+py/(p-1)) - \gamma(x+1, px)$, dla ustalonego y , do wyrazów tego samego rzędu co w poprzednim ćwiczeniu.

► **11.** [HM35] Uogólnijmy funkcje $Q(n)$ i $R(n)$, wprowadzając parametr x :

$$Q_x(n) = 1 + \frac{n-1}{n}x + \frac{n-1}{n} \frac{n-2}{n}x^2 + \dots,$$

$$R_x(n) = 1 + \frac{n}{n+1}x + \frac{n}{n+1} \frac{n}{n+2}x^2 + \dots.$$

Zbadaj takie uogólnienia i znajdź wzory asymptotyczne dla $x \neq 1$.

12. [HM20] Funkcję $\int_0^x e^{-t^2/2} dt$ pojawiającą się w związku z rozkładem normalnym (zobacz punkt 1.2.10) można przedstawić jako przypadek szczególny niepełnej funkcji gamma. Znajdź takie wartości a , b i y , że $b\gamma(a, y)$ równa się $\int_0^x e^{-t^2/2} dt$.

13. [HM42] (S. Ramanujan) Udowodnij, że $R(n) - Q(n) = \frac{2}{3} + 8/(135(n+\theta(n)))$, gdzie $\frac{2}{21} \leq \theta(n) \leq \frac{8}{45}$. (Pociąga to za sobą o wiele słabszy wynik $R(n+1) - Q(n+1) < R(n) - Q(n)$).

► **14.** [HM39] (N. G. de Bruijn) Celem tego ćwiczenia jest wyznaczenie rozwinięcia asymptotycznego sumy $\sum_{k=0}^n k^{n+\alpha} e^{-k}$ dla ustalonego α , przy $n \rightarrow \infty$.

a) Zastępując k przez $n-k$, pokaż, że suma ta równa się $n^{n+\alpha} e^{-n} \sum_{k=0}^n e^{-k^2/2n} f(k, n)$, gdzie

$$f(k, n) = \left(1 - \frac{k}{n}\right)^\alpha \exp\left(-\frac{k^3}{3n^2} - \frac{k^4}{4n^3} - \dots\right).$$

b) Pokaż, że dla wszystkich $m \geq 0$ i $\epsilon > 0$ wielkość $f(k, n)$ można zapisać w postaci

$$\sum_{0 \leq i \leq j \leq m} c_{ij} k^{2i+j} n^{-i-j} + O(n^{(m+1)(-1/2+3\epsilon)}), \quad \text{jeśli } 0 \leq k \leq n^{1/2+\epsilon}.$$

c) Pokaż, że jako wniosek z (b) mamy

$$\sum_{k=0}^n e^{-k^2/2n} f(k, n) = \sum_{0 \leq i \leq j \leq m} c_{ij} n^{-i-j} \sum_{k \geq 0} k^{2i+j} e^{-k^2/2n} + O(n^{-m/2+\delta}),$$

dla wszystkich $\delta > 0$. [Wskazówka: Sumy względem $n^{1/2+\epsilon} < k < \infty$ są $O(n^{-r})$ dla wszystkich r].

d) Pokaż, że rozwinięcie asymptotyczne $\sum_{k \geq 0} k^t e^{-k^2/2n}$ dla ustalonego $t \geq 0$ można uzyskać ze wzoru sumacyjnego Eulera.

e) Zatem ostatecznie

$$\sum_{k=0}^n k^{n+\alpha} e^{-k} = n^{n+\alpha} e^{-n} \left(\sqrt{\frac{\pi n}{2}} - \frac{1}{6} - \alpha + \left(\frac{1}{12} + \frac{1}{2}\alpha + \frac{1}{2}\alpha^2 \right) \sqrt{\frac{\pi}{2n}} + O(n^{-1}) \right);$$

powyższe obliczenie można ciągnąć do $O(n^{-r})$ dla dowolnie wybranego r .

15. [HM20] Pokaż, że poniższa całka ma związek z $Q(n)$:

$$\int_0^\infty \left(1 + \frac{z}{n}\right)^n e^{-z} dz.$$

16. [M24] Udowodnij tożsamość

$$\sum_k (-1)^k \binom{n}{k} k^{n-1} Q(k) = (-1)^n (n-1)! \quad \text{dla } n > 0.$$

17. [HM29] (K. W. Miller) Dla symetrii powinniśmy rozważyć czwartą sumę, która tak się ma do $P(n)$, jak $R(n)$ do $Q(n)$:

$$S(n) = 1 + \frac{n}{n+1} + \frac{n}{n+2} \frac{n+1}{n+2} + \dots = \sum_{k \geq 0} \frac{(n+k-1)!}{(n-1)!(n+k)^k}.$$

Jakie jest asymptotyczne zachowanie tej funkcji?

18. [M25] Pokaż, że sumy $\sum \binom{n}{k} k^k (n-k)^{n-k}$ i $\sum \binom{n}{k} (k+1)^k (n-k)^{n-k}$ można bardzo prosto zapisać za pomocą funkcji Q .

19. [HM30] (Lemat Watsona) Pokaż, że jeśli całka $C_n = \int_0^\infty e^{-nx} f(x) dx$ istnieje dla wszystkich dużych n i jeśli $f(x) = O(x^\alpha)$ dla $0 \leq x \leq r$, gdzie $r > 0$ i $\alpha > -1$, to $C_n = O(n^{-1-\alpha})$.

► **20.** [HM30] Niech $u = w + \frac{1}{3}w^2 + \frac{1}{36}w^3 - \frac{1}{270}w^4 + \dots = \sum_{k=1}^\infty c_k w^k$ będzie szeregiem potęgowym spełniającym równanie $w = (u^2 - \frac{2}{3}u^3 + \frac{2}{4}u^4 - \frac{2}{5}u^5 + \dots)^{1/2}$, jak w (12). Pokaż, że

$$Q(n) + 1 = \sum_{k=1}^{m-1} k c_k \Gamma(k/2) \left(\frac{n}{2}\right)^{1-k/2} + O(n^{1-m/2})$$

dla wszystkich $m \geq 1$. [Wskazówka: Zastosuj lemat Watsona do tożsamości z ćwiczenia 15].

Czuję, że powinnam osiągnąć coś w matematyce, choć nie rozumiem, dlaczego wydaje mi się to tak ważne.

— HELEN KELLER (1898)

1.3. MIX

Wiele razy w tej książce będziemy odwoływać się do wewnętrznego języka komputera, tzw. języka maszynowego. Korzystać będziemy z mitycznego komputera o nazwie „MIX”. MIX przypomina większość komputerów z lat sześćdziesiątych i siedemdziesiątych, z tą może różnicą, że jest bardziej przyjazny. Język komputera MIX zaprojektowano tak, by był wystarczająco silny do zapisywania większości algorytmów w zwięzłej postaci, a jednocześnie na tyle prosty, by łatwo było się go nauczyć.

Nalegamy, by Czytelnik bardzo uważnie zapoznał się z niniejszym rozdziałem, ponieważ komputer MIX często się w tej książce pojawia. Nie należy obawiać się nauki języka maszynowego. Autorowi zdarzało się nieraz w ciągu jednego tygodnia pisać programy w tylu różnych językach maszynowych, że na palcach jednej ręki by nie zliczył. Każdy, kto poważnie interesuje się komputerami, wcześniej czy później pozna przynajmniej jeden język maszynowy. MIX został tak zaprojektowany, by zachować najbardziej podstawowe aspekty historycznych komputerów, łatwo więc jest oswoić się z jego „charakterem”.

 Przyznajemy, że MIX jest dziś nieco przestarzały. Dlatego w kolejnych wydaniach zostanie on zastąpiony nową maszyną MMIX 2009. Będzie to tzw. komputer o ograniczonej liście rozkazów (RISC), wykonujący obliczenia na słowach 64-bitowych. MMIX będzie maszyną jeszcze przyjemniejszą niż MIX, zbliżoną do maszyn popularnych w latach dziewięćdziesiątych.

Zadanie przerobienia wszystkiego w tej książce z komputera MIX na MMIX zajmie dużo czasu. Autor zachęca ochronników do pomocy w tym dziele i żywi nadzieję, że do czasu zakończenia tej pracy Czytelnicy jeszcze przez parę lat będą zadowalać się staroświecką architekturą MIX, którą tak czy owak warto poznać.

1.3.1. Opis maszyny MIX

MIX jest pierwszym w świecie komputerem wielonienasyconym. Jak większość maszyn, także MIX został oznaczony numerem: 1009. Ten numer pochodzi od nazw 16 prawdziwych komputerów, które są bardzo podobne do komputera MIX i na których łatwo daje się go symulować. Z ich numerów wyciągnięto średnią (z równymi wagami):

$$\lfloor (360 + 650 + 709 + 7070 + U3 + SS80 + 1107 + 1604 + G20 + B220 \\ + S2000 + 920 + 601 + H800 + PDP-4 + II) / 16 \rfloor = 1009. \quad (1)$$

Numer modelu komputera MIX można także otrzymać prościej, posługując się liczbami rzymskimi.

MIX ma tę dziwną cechę, że jest jednocześnie dwójkowy i dziesiętny. Tak naprawdę programiści komputera MIX nie wiedzą, czy programują na maszynie pracującej w arytmetyce przy podstawie 2 czy 10. Stąd algorytmy zapisane w języku komputera MIX mogą być użyte na maszynie dowolnego typu bez konieczności wprowadzania dużych zmian, a także sam komputer MIX łatwo symulować na dowolnej maszynie. Programiści przyzwyczajeni do maszyn dwójkowych mogą wyobrażać sobie, że MIX jest dwójkowy. Programiści przywykli do

systemu dziesiętnego mogą uznawać, że **MIX** to komputer dziesiętny. Programiści z innej planety mogą myśleć, że **MIX** jest trójkowy.

Słowa. Podstawową jednostką informacji jest *bajt*. Każdy bajt zawiera *nieokreślona* ilość informacji, ale musi być w stanie reprezentować przynajmniej 64 różne wartości. Znaczy to, że każda liczba od 0 do 63 włącznie może być przechowywana w jednym bajcie. Co więcej, każdy bajt reprezentuje jedną z *co najwyżej* 100 różnych wartości. Na komputerze dwójkowym bajt musi zatem składać się z sześciu bitów, na komputerze dziesiętnym mamy dwie cyfry na bajt*.

Programy formułowane w języku komputera **MIX** powinny być tak pisane, by nie zakładać, że bajt może przyjmować więcej niż 64 wartości. Chcąc posługiwać się liczbą 80, musimy zarezerwować dwa sąsiednie bajty na jej reprezentację, choć na komputerze dziesiętnym wystarczyłby jeden bajt. *Algorytm przeznaczony do wykonania na komputerze MIX, powinien działać prawidłowo niezależnie od rozmiaru bajtu.* Chociaż moglibyśmy napisać program, którego działanie zależy od rozmiaru bajtu, czyn taki byłby pogwałceniem ducha tej książki. Jedyne dopuszczalne programy to te, które dają poprawny wynik przy dowolnym rozmiarze bajtu. Zazwyczaj trzymanie się tej zasady nie jest dużym problemem. Okazuje się również, że programowanie komputera dziesiętnego nie różni się istotnie od programowania komputera dwójkowego.

Za pomocą:

- dwóch kolejnych bajtów można zapisać liczby od 0 do 4095;
- trzech kolejnych bajtów można zapisać liczby od 0 do 262 143;
- czterech kolejnych bajtów można zapisać liczby od 0 do 16 777 215;
- pięciu kolejnych bajtów można zapisać liczby od 0 do 1 073 741 823.

Słowo maszynowe składa się z pięciu bajtów i znaku. Część przeznaczona na znak może zawierać tylko jedną z dwóch wartości: + lub -.

Rejestry. **MIX** ma dziewięć rejestrów (zobacz rysunek 13):

Rejestr A (akumulator) składa się z pięciu bajtów i znaku.

Rejestr X (*extension*) – rejestr rozszerzenia – również zawiera pięć bajtów i znak.

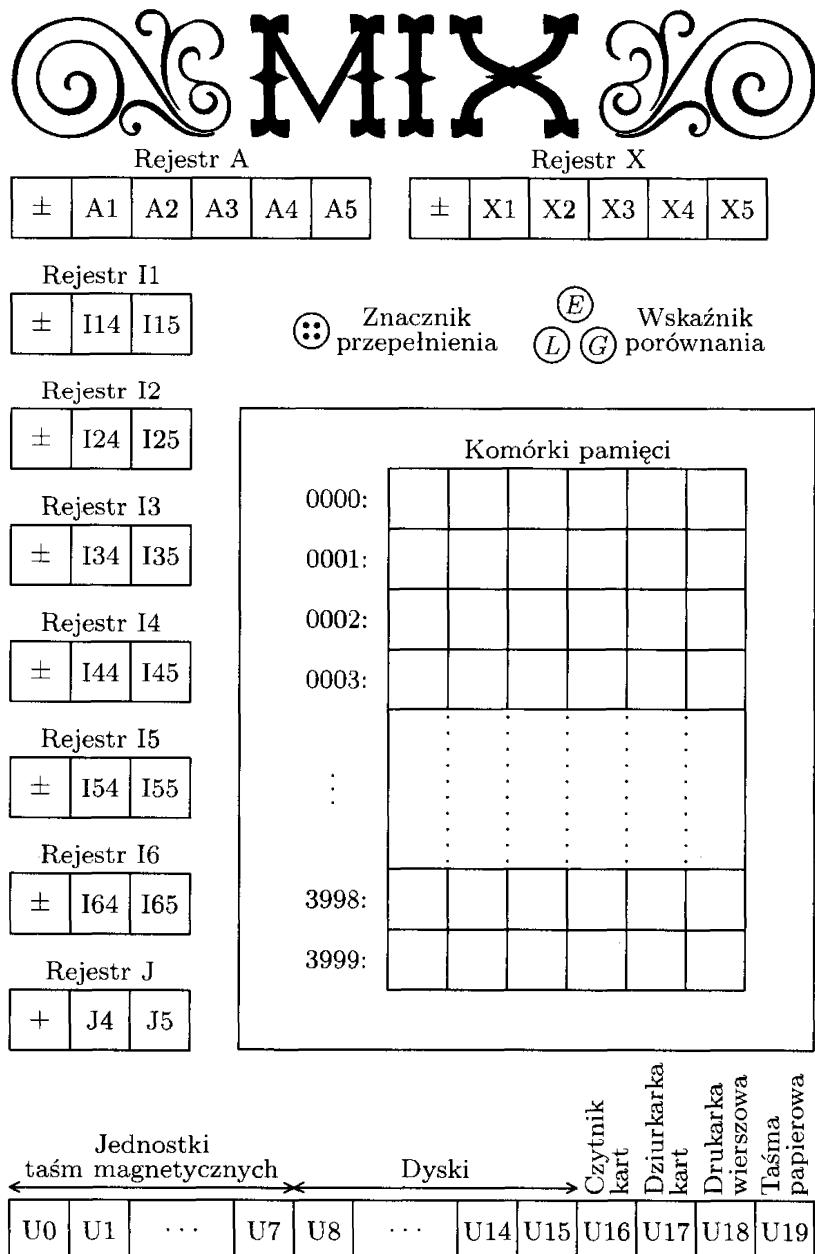
Rejestry I (indeksowe) I₁, I₂, I₃, I₄, I₅ i I₆ mają po dwa bajty ze znakiem.

Rejestr J (*jump*) – rejestr skoku – ma dwa bajty; zawarta w nim liczba zachowuje się tak, jakby zawsze była ze znakiem +.

Odwołując się do rejestrów komputera **MIX**, będziemy nazwę rejestrów poprzedzać małą literą „r”, na przykład „rA” oznacza „rejestr A”.

Rejestr A odgrywa istotną rolę przy operacjach arytmetycznych. Rejestr X służy za rozszerzenie „prawej strony” rA i jest używany w połączeniu z rA do przechowywania 10-bajtowego iloczynu lub dzielnej. Może być w nim także

* Mniej więcej od 1975 roku słowo „bajt” zaczęło oznaczać ciąg dokładnie ośmiu cyfr dwójkowych, mogący reprezentować liczby od 0 do 255. Prawdziwe bajty są zatem większe niż te w hipotetycznej maszynie **MIX**. W istocie staroświeckie bajty maszyny **MIX** są tylko troszeczkę większe niż *nybbles*. Gdy mówimy o bajtach w sensie maszyny **MIX**, mamy na myśli stare znaczenie tego słowa, z czasów gdy bajty nie były jeszcze ustandaryzowane.



Rys. 13. Komputer MIX.

przechowywana informacja, która „wyszła” poza rA przy operacji przesunięcia. Rejestry indeksowe rI1, rI2, rI3, rI4, rI5 i rI6 są wykorzystywane głównie do zliczania oraz wyznaczania adresów w pamięci. W rejestrze J zawsze jest przechowywany adres instrukcji następującej bezpośrednio po ostatnim rozkazie skoku i jest on głównie używany przy wywoływaniu podprogramów.

Poza rejestrami MIX ma

- znacznik przepelnienia* (pojedynczy bit, włączony albo wyłączony);
- wskaźnik porównania* (przybierający trzy wartości: LESS (mniejsze), EQUAL (równie) lub GREATER (większe));
- pamięć* (4000 słów pamięci, a w każdym słowie pięć bajtów i znak)
- oraz *urządzenia wejścia-wyjścia* (karty, taśmy, dyski itp.).

Pola w słowie maszynowym. Bajty i znak w słowie maszynowym są ponumerowane w następujący sposób:

0	1	2	3	4	5
±	bajt	bajt	bajt	bajt	bajt

(2)

Większość rozkazów umożliwia programiście korzystanie z wybranego fragmentu słowa. W takich przypadkach rozkaz musi zawierać „specyfikację pola”. Dopuszcza się korzystanie z wielu przylegających pól składowych słowa maszynowego (wycinka). „Wycinek” słowa oznaczamy przez (L:R), gdzie L jest numerem pola z lewego, a R – z prawego krańca wycinka. Oto kilka przykładów specyfikacji „wycinków” słowa:

- (0:0) – tylko znak.
- (0:2) – znak i dwa pierwsze bajty.
- (0:5) – całe słowo; taką specyfikację spotykamy najczęściej.
- (1:5) – całe słowo poza znakiem.
- (4:4) – tylko czwarty bajt.
- (4:5) – dwa najmniej znaczące bajty.

Wykorzystanie pól zmienia się nieco w zależności od konkretnego rozkazu; wyjaśnimy je szczegółowo dla każdego rozkazu, dla którego ma sens. Każda specyfikacja wycinka (L:R) jest reprezentowana w maszynie za pomocą jednej liczby $8L + R$. Zauważmy, że liczba ta mieści się w jednym bajcie.

Format rozkazu. Słowa maszynowe, będące kodami rozkazu, mają następującą postać:

0	1	2	3	4	5
±	A	A	I	F	C

(3)

Skrajnie prawy bajt C jest *kodem operacji*, mówiącym, którą operację należy wykonać. Na przykład C = 8 oznacza operację LDA: „załaduj wartość do rejestru A”.

Bajt F zawiera *modyfikator* kodu operacji i jest zazwyczaj specyfikacją wycinka (L:R) = $8L + R$. Jeśli na przykład C = 8 i F = 11, to operacją jest „załaduj pola (1:3) do rejestru A”. Czasami F służy do innych celów; na przykład w rozkazach wejścia-wyjścia F wyznacza numer urządzenia wejścia lub wyjścia.

Lewa część rozkazu ±AA to *adres*. (Zauważ, że znak jest częścią adresu). Bajt I, występujący po adresie, to *specyfikacja indeksowania*, która wpływa na wyznaczanie rzeczywistego adresu. Gdy I = 0, adres ±AA jest brany bez zmian. I może także zawierać liczbę i z zakresu od 1 do 6, co oznacza, że przed wykonaniem rozkazu zawartość rejestru indeksowego II ma zostać zsumowana z ±AA, a otrzymaną wartość należy traktować jak adres. Proces indeksowania dotyczy *każdego* rozkazu. Będziemy używać litery M na oznaczenie adresu po indeksowaniu. (Jeśli dodanie zawartości rejestru indeksowego do adresu ±AA daje wynik nie mieszczący się w dwóch bajtach, wartość M jest nieokreślona).

W większości rozkazów M odnosi się do komórki pamięci. Wyrażenia „komórka pamięci” i „lokalna (w pamięci)” są w tej książce używane zamiennie. Zakładamy, że w komputerze MIX jest 4000 komórek pamięci, ponumerowanych od 0 do 3999. Każda komórka może być zatem zaadresowana za pomocą dwóch bajtów. Dla każdego rozkazu, w którym M odnosi się do komórki pamięci, powinien zachodzić warunek $0 \leq M \leq 3999$ i jeśli jest on spełniony, to piszemy CONTENTS(M) na oznaczenie wartości przechowywanej w komórce pamięci o numerze M.

Są rozkazy, dla których „adres” M ma inne znaczenie i może być nawet ujemny. Pewien rozkaz powoduje dodanie M do zawartości rejestru indeksowego; w tej operacji uwzględnia się znak M.

Oznaczenia. By przedstawić rozkazy w sposób czytelny, będziemy posługiwać się notacją

$$\text{OP} \quad \text{ADRES}, I(F) \quad (4)$$

na oznaczenie rozkazu kodowanego według (3). OP oznacza nazwę symboliczną kodu operacji (część C), ADRES to część $\pm AA$, natomiast I i F odpowiadają częściami I i F.

Gdy I jest zerem, pomijamy „,I”. Gdy F jest standardową specyfikacją pól dla danej operacji, nie trzeba pisać „,(F)”. Standardową specyfikacją wycinka dla prawie wszystkich operacji jest (0:5), co oznacza całe słowo. Jeśli dla jakiejś operacji za standardową przyjmuje się inną specyfikację, zaznaczamy ten fakt w opisie operacji.

Na przykład rozkaz powodujący załadowanie liczby do akumulatora ma nazwę symboliczną LDA, a jego kodem operacji jest 8. Mamy więc

Reprezentacja symboliczna	Kod maszynowy	
LDA 2000,2(0:3)	+ 2000 2 3 8	
LDA 2000,2(1:3)	+ 2000 2 11 8	
LDA 2000(1:3)	+ 2000 0 11 8	
LDA 2000	+ 2000 0 5 8	
LDA -2000,4	- 2000 4 5 8	

Rozkaz „LDA 2000,2(0:3)” należy rozumieć jako „Załaduj do A wycinek zero-trzy zawartości komórki nr 2000 indeksowanej rejestrem I2”.

Do reprezentowania zawartości liczbowej słowa zawsze będziemy używać notacji „pułapkowej”, jak powyżej. Zauważmy, że w słowie

+	2000	2	3	8
---	------	---	---	---

liczba +2000 zajmuje dwa przylegające bajty oraz znak. Konkretna zawartość bajtu (1:1) i (2:2) będzie mogła być inna na różnych egzemplarzach komputera MIX, bo różne mogą być rozmiary bajtów. W ramach kolejnego przykładu przyjrzyjmy się obrazkowi:

-	10000	3000
---	-------	------

oznaczającemu słowo o dwóch wycinkach: wycinek „trzy bajty i znak” zawiera liczbę -10000, a wycinek „dwa bajty” zawiera liczbę 3000. Gdy na słowo składa się więcej niż jeden wycinek, mówimy, że słowo jest „upakowane”.

Rozkazy. W akapitach następujących po tabelce (3) są zdefiniowane wielkości M, F i C dla każdego słowa kodującego rozkaz. Opiszemy teraz działanie rozkazu.

Ładowanie wartości do rejestru.

- LDA (załaduj wartość do A). C = 8; F = wycinek.

W wyniku wykonania operacji podany wycinek słowa CONTENTS(M) zastępuje poprzednią wartość przechowywaną w rejestrze A.

Przy wszystkich operacjach wykonywanych na części słowa (na kilku polach) znak jest brany pod uwagę, jeśli jest w tej części słowa. W przeciwnym razie przyjmuje się znak +. Wycinek słowa jest dosuwany w prawo do skraju słowa, zanim zostanie załadowany.

Przykłady: Jeśli F jest standardową specyfikacją wycinka, tj. (0:5), to cała zawartość lokacji M jest kopiowana do rA. Jeśli F jest postaci (1:5), to do rA jest ładowana wartość bezwzględna CONTENTS(M) ze znakiem +. Jeśli M zawiera kod rozkazu, a F jest postaci (0:2), do rA jest ładowana wartość „±AA” jako

±	0	0	0	A	A	.
---	---	---	---	---	---	---

Gdy lokacja 2000 zawiera słowo

-	80	3	5	4	;	(6)
---	----	---	---	---	---	-----

wówczas ładując różne części słowa, otrzymamy następujące wyniki:

Rozkaz	Zawartość rA po wykonaniu rozkazu
--------	-----------------------------------

LDA 2000	<table border="1"> <tr><td>-</td><td>80</td><td>3</td><td>5</td><td>4</td></tr> </table>	-	80	3	5	4	
-	80	3	5	4			
LDA 2000(1:5)	<table border="1"> <tr><td>+</td><td>80</td><td>3</td><td>5</td><td>4</td></tr> </table>	+	80	3	5	4	
+	80	3	5	4			
LDA 2000(3:5)	<table border="1"> <tr><td>+</td><td>0</td><td>0</td><td>3</td><td>5</td><td>4</td></tr> </table>	+	0	0	3	5	4
+	0	0	3	5	4		
LDA 2000(0:3)	<table border="1"> <tr><td>-</td><td>0</td><td>0</td><td>80</td><td>3</td></tr> </table>	-	0	0	80	3	
-	0	0	80	3			
LDA 2000(4:4)	<table border="1"> <tr><td>+</td><td>0</td><td>0</td><td>0</td><td>0</td><td>5</td></tr> </table>	+	0	0	0	0	5
+	0	0	0	0	5		
LDA 2000(0:0)	<table border="1"> <tr><td>-</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	-	0	0	0	0	0
-	0	0	0	0	0		
LDA 2000(1:1)	<table border="1"> <tr><td>+</td><td>0</td><td>0</td><td>0</td><td>0</td><td>?</td></tr> </table>	+	0	0	0	0	?
+	0	0	0	0	?		

(Ostatni wynik jest częściowo nieokreślony, bo rozmiar bajtu nie jest ustalony).

- LDX (załaduj wartość do X). C = 15; F = wycinek.

Operacja analogiczne do LDA, z tym że wartość jest ładowana do rX, a nie do rA.

- LD*i* (załaduj wartość do *i*). C = 8 + *i*; F = wycinek.

Operacja analogiczna do LDA, z tym że wartość jest ładowana do r*i*, a nie do rA. Rejestr indeksowy zawiera tylko dwa bajty (a nie pięć) oraz znak. Zakłada się, że bajty 1, 2 i 3 są równe zeru. Wynik wykonania rozkazu LD*i* byłby nieokreślony, gdyby w wyniku załadowania bajty 1, 2 lub 3 miały otrzymać wartości niezerowe.

W opisie wszystkich rozkazów „*i*” oznacza liczbę całkowitą, $1 \leq i \leq 6$. Stąd LD*i* oznacza sześć różnych rozkazów: LD1, LD2, …, LD6.

- LDAN (załaduj wartość przeciwną do A). C = 16; F = wycinek.
- LDXN (załaduj wartość przeciwną do X). C = 23; F = wycinek.
- LD*i*N (załaduj wartość przeciwną do *i*). C = $16 + i$; F = wycinek.

Powyższych osiem rozkazów działa podobnie, odpowiednio, do LDA, LDX, i LD*i*, z tym wyjątkiem, że ładowanym wartościom zmienia się znak na przeciwny.

Zapisywanie wartości w pamięci.

- STA (zapisz A w pamięci). C = 24; F = wycinek.

W wyniku wykonania operacji wycinek słowa CONTENTS(M) określony przez F zostaje zastąpiony fragmentem rA. Reszta słowa CONTENTS(M) pozostaje bez zmian.

W przypadku operacji *zapisu w pamięci* wycinek F ma znaczenie odwrotne niż w przypadku operacji *ładowania do rejestru*: bajty wycinka są pobierane z prawego końca rejestru i jeśli trzeba przesuwane *w lewo*, by trafiły do odpowiednich pól słowa CONTENTS(M). Znak się nie zmienia, jeśli nie jest częścią wycinka. Zawartość rejestru pozostaje niezmieniona.

Przykłady: założmy, że lokacja 2000 zawiera

-	1	2	3	4	5
---	---	---	---	---	---

a rejestr A zawiera

+	6	7	8	9	0
---	---	---	---	---	---

Wówczas

Rozkaz

Zawartość lokacji 2000 po wykonaniu rozkazu

- | | |
|-----|-----------|
| STA | 2000 |
| STA | 2000(1:5) |
| STA | 2000(5:5) |
| STA | 2000(2:2) |
| STA | 2000(2:3) |
| STA | 2000(0:1) |

+	6	7	8	9	0
-	6	7	8	9	0
-	1	2	3	4	0
-	1	0	3	4	5
-	1	9	0	4	5
+	0	2	3	4	5

- STX (zapisz X do pamięci). C = 31; F = wycinek.

Operacja analogiczna do STA, z tym że zapisywana jest wartość z rX, a nie z rA.

- ST*i* (zapisz *i*). C = $24 + i$; F = wycinek.

Operacja analogiczna do STA, z tym że zapisywana jest wartość z r*i*, a nie z rA.

Przyjmuje się, że bajty 1, 2 i 3 rejestru indeksowego są równe zeru, jeśli zatem rI1 zawiera

\pm	<i>m</i>	<i>n</i>
-------	----------	----------

to rozkaz jest wykonywany tak, jakby rejestr zawierał

\pm	0	0	0	<i>m</i>	<i>n</i>
-------	---	---	---	----------	----------

- STJ (zapisz J w pamięci). C = 32; F = wycinek.

Operacja analogiczna do STi, z tym że zapisywana jest wartość z rJ (ze znakiem +).

Dla operacji STJ standardowa specyfikacja wycinka to (0:2), a nie (0:5). To wydaje się logiczne, gdyż STJ prawie zawsze służy do zapisania wartości do pola adresu kodu rozkazu.

- STZ (zapisz zero w pamięci). C = 33; F = wycinek.

Operacja analogiczna do STA, z tym że zapisywana jest wartość „plus zero”, a nie wartość rA. Innymi słowy: podana część słowa CONTENTS(M) ulega wyzerowaniu.

Operacje arytmetyczne. Dla operacji dodawania, odejmowania, mnożenia i dzielenia zezwalamy na określanie wycinków. Dopuszczalne jest także użycie specyfikacji wycinka „(0:6)” do oznaczenia „operacji zmienopozycyjnej” (zobacz podrozdział 4.2), ale bardzo niewiele programów pisanych przez nas na komputer MIX będzie korzystało z tej możliwości. Jesteśmy zainteresowani głównie algorytmami operującymi na liczbach całkowitych.

Standardowa specyfikacja wycinka dla operacji arytmetycznych to (0:5). Inne wycinki traktuje się jak w operacji LDA. Będziemy używać litery V na oznaczenie części słowa CONTENTS(M) wyznaczanej przez kod rozkazu. V jest zatem wartością, która została załadowana do rejestru A, gdyby kodem operacji było LDA.

- ADD. C = 1; F = wycinek.

W wyniku wykonania operacji wartość V jest dodawana do zawartości rA. Jeśli wynik nie mieści się w rejestrze A, to ustawiany jest znacznik przepelnienia, a w A pozostaje wynik „obcięty”, tak jakby „1” zostało przeniesione do innego rejestru za lewy skraj rA. (W przeciwnym razie znacznik przepelnienia pozostaje niezmieniony). Jeżeli wynikiem jest zero, to znak rA pozostaje niezmieniony.

Przykład: Wykonanie poniższego ciągu rozkazów powoduje obliczenie sumy pięciu bajtów rejestru A.

STA	2000
LDA	2000(5:5)
ADD	2000(4:4)
ADD	2000(3:3)
ADD	2000(2:2)
ADD	2000(1:1)

Jest to czasami nazywane dodawaniem stronami.

W niektórych sytuacjach przepelnienie będzie występować na pewnych wersjach komputera MIX, a na innych nie. Jest to spowodowane różnicami wielkości bajtu. Nie powiedzieliśmy, że przepelnienie będzie zawsze, gdy wartość okaże się większa niż 1073741823. Przepelnienie występuje wówczas, gdy wartość wyniku okazuje się zbyt duża, by pomieścić ją w pięciu bajtach. Te różnice nie oznaczają, że nie można pisać programów, które dają te same końcowe odpowiedzi niezależnie od rozmiaru bajtu.

- SUB (odejmowanie). C = 2; F = wycinek.

W wyniku wykonania operacji wartość V zostaje odjęta od zawartości rA. (Równoważne ADD, gdy zamiast V weźmiemy $-V$).

- MUL (mnożenie). C = 3; F = wycinek.

W wyniku wykonania operacji 10-bajtowy iloczyn V i rA zostaje załadowany do rejestrów A i X. Do znaku rA i do znaku rX zapisywany jest algebraiczny znak iloczynu (tj. +, jeżeli znaki V i rA były takie same, a -, jeżeli były różne).

- DIV (dzielenie). C = 4; F = wycinek.

W wyniku wykonania operacji łączna wartość rA i rX, potraktowana jako jedna liczba 10-bajtowa rAX ze znakiem pochodzącym z rA, zostaje podzielona przez wartość V. Jeśli V = 0 lub jeśli reprezentacja ilorazu wymaga więcej niż pięciu bajtów (co odpowiada warunkowi $|rA| \geq |V|$), do rejestrów A i X jest zapisywana informacja nieokreślona i ustawiany jest znacznik przepełnienia. W przeciwnym razie iloraz $\pm \lfloor |rAX/V| \rfloor$ jest umieszczany w rA, a reszta $\pm (|rAX| \bmod |V|)$ – w rX. Znak rA po operacji jest algebraicznym znakiem ilorazu (tj. +, jeżeli znaki V i rA były takie same, a -, jeżeli były różne). Znak rX po operacji jest znakiem rA przed operacją.

Przykłady rozkazów arytmetycznych: w wielu przypadkach operacje arytmetyczne wykonuje się na całych słowach (liczbach 5-bajtowych) bez wykorzystywania wycinków i mechanizmu upakowania. Można jednak przy zachowaniu pewnej dozy ostrożności wykonywać operacje arytmetyczne na upakowanych słowach. Warto przyjrzeć się poniższym przykładom. (Jak poprzednio, ? oznacza wartość nieokreśloną).

		<table border="1"> <tr><td>+</td><td>1234</td><td>1</td><td>150</td><td></td></tr> <tr><td>+</td><td>100</td><td>5</td><td>50</td><td></td></tr> <tr><td>+</td><td>1334</td><td>6</td><td>200</td><td></td></tr> </table>	+	1234	1	150		+	100	5	50		+	1334	6	200		rA przed operacją									
+	1234	1	150																								
+	100	5	50																								
+	1334	6	200																								
ADD	1000	<table border="1"> <tr><td>-</td><td>1234</td><td>0</td><td>0</td><td>9</td></tr> <tr><td>-</td><td>2000</td><td>150</td><td>0</td><td></td></tr> <tr><td>+</td><td>766</td><td>149</td><td>?</td><td></td></tr> </table>	-	1234	0	0	9	-	2000	150	0		+	766	149	?		komórka 1000									
-	1234	0	0	9																							
-	2000	150	0																								
+	766	149	?																								
SUB	1000	<table border="1"> <tr><td>-</td><td>1234</td><td>0</td><td>0</td><td>9</td></tr> <tr><td>-</td><td>2000</td><td>150</td><td>0</td><td></td></tr> <tr><td>+</td><td>766</td><td>149</td><td>?</td><td></td></tr> </table>	-	1234	0	0	9	-	2000	150	0		+	766	149	?		rA po operacji									
-	1234	0	0	9																							
-	2000	150	0																								
+	766	149	?																								
MUL	1000	<table border="1"> <tr><td>+</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>+</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>+</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr> <tr><td>+</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td></tr> </table>	+	1	1	1	1	1	+	1	1	1	1	1	+	0	1	2	3	4	+	5	4	3	2	1	rA przed operacją komórka 1000 rA po operacji rX po operacji
+	1	1	1	1	1																						
+	1	1	1	1	1																						
+	0	1	2	3	4																						
+	5	4	3	2	1																						
MUL	1000(1:1)	<table border="1"> <tr><td>-</td><td></td><td></td><td></td><td></td><td>112</td></tr> <tr><td>?</td><td>2</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> <tr><td>-</td><td></td><td></td><td></td><td></td><td>0</td></tr> <tr><td>-</td><td></td><td></td><td></td><td></td><td>224</td></tr> </table>	-					112	?	2	?	?	?	?	-					0	-					224	rA przed operacją komórka 1000 rA po operacji rX po operacji
-					112																						
?	2	?	?	?	?																						
-					0																						
-					224																						

MUL 1000

-	50	0	112	4
-	2	0	0	0
+	100	0	0	224
+	8	0	0	0

rA przed operacją
komórka 1000
rA po operacji
rX po operacji

DIV 1000

+				0
?				17
+				3
+				5
+				2

rA przed operacją
rX przed operacją
komórka 1000
rA po operacji
rX po operacji

DIV 1000

-				0
+	1235	0	3	1
-	0	0	0	2
+	0	617	?	?
-	0	0	0	?

rA przed operacją
rX przed operacją
komórka 1000
rA po operacji
rX po operacji

(Powyższe przykłady przygotowano zgodnie z zasadą, że lepiej podać opis pełny i zagmatwany niż zwięzły, lecz niepełny).

Operacje przekazania adresu. W poniższych operacjach „adres” M (ewentualnie indeksowany) jest wykorzystywany po prostu jako liczba ze znakiem, a nie adres komórki pamięci.

- **ENTA** (wpisz wartość do A). C = 48; F = 2.

W wyniku wykonania operacji wartość M jest ładowana do rA. Operacja jest równoważna operacji „LDA” ładowania wartości z komórki pamięci zawierającej wartość M (ze znakiem). Jeśli M = 0, ładowany jest znak komórki zawierającej rozkaz.

Przykłady: W wyniku wykonania „ENTA 0” rejestr rA jest ustawiany na zero ze znakiem +. „ENTA 0,1” powoduje wstawienie do rA bieżącej zawartości rejestru indeksowego 1, z tym że -0 zmieniane jest na +0. Operacja „ENTA -0,1” jest podobna, z tym że +0 zmieniane jest na -0.

- **ENTX** (wpisz wartość do X). C = 55; F = 2.
- **ENT*i*** (wpisz wartość do *i*). C = 48 + *i*; F = 2.

To operacje analogiczne do ENTA, polegające na wpisaniu wartości do odpowiedniego rejestru.

- **ENNA** (wpisz wartość przeciwną do A). C = 48; F = 3.
- **ENN_X** (wpisz wartość przeciwną do X). C = 55; F = 3.
- **ENN*i*** (wpisz wartość przeciwną do *i*). C = 48 + *i*; F = 3.

Operacje takie jak ENTA, ENTX i ENTi, z tym że wpisywana jest wartość z przeciwnym znakiem.

Przykład: „ENN3 0,3” powoduje zastąpienie wartości w rI3 wartością do niej przeciwną, z tym że –0 pozostaje –0.

- INCA (zwiększ A). C = 48; F = 0.

W wyniku wykonania operacji wartość M jest dodawana do rA. Operacja jest równoważna operacji „ADD” dla komórki pamięci zawierającej wartość M. Może wystąpić przepełnienie, które jest traktowane tak, jak przy operacji ADD.

Przykład: „INCA 1” powoduje zwiększenie wartości w rA o jeden.

- INCX (zwiększ X). C = 55; F = 0.

W wyniku wykonania operacji wartość M jest dodawana do wartości w rX. Jeśli wystąpi przepełnienie, operacja przebiega analogicznie do ADD, z tym że w miejscu rA używa się rX. Rozkaz ten nie zmienia zawartości rejestru A.

- INC_i (zwiększ i). C = 48 + i; F = 0.

W wyniku wykonania operacji wartość M jest dodawana do rI_i. Wystąpienie przepełnienia jest błędem. Jeśli M + rI_i nie mieści się w dwóch bajtach, wynik operacji jest nieokreślony.

- DECA (zmniejsz A). C = 48; F = 1.

- DECX (zmniejsz X). C = 55; F = 1.

- DEC_i (zmniejsz i). C = 48 + i; F = 1.

Tych osiem rozkazów jest podobnych odpowiednio do INCA, INCX i INC_i, z tym że wartość M jest odejmowana od wartości w rejestrze (a nie dodawana do niej).

Zauważmy, że kod operacji C jest taki sam dla ENTA, ENNA, INCA i DECA. Rozkazy rozróżniamy na podstawie zawartości pola F.

Operacje porównania. Wszystkie operacje porównania porównują wartość zapisaną w rejestrze z wartością zapisaną w pamięci. *Wskaźnik porównania* jest następnie ustawiany na LESS (mniejsze), EQUAL (równe) lub GREATER (większe) w zależności od tego, czy wartość w rejestrze była mniejsza, równa, czy większa od wartości w komórce pamięci. Minus zero jest *równe plus zeru*.

- CMPA (porównaj A). C = 56; F = wycinek.

W wyniku wykonania operacji wartość podanego wycinka rA jest porównywana z wartością *tego samego* wycinka CONTENTS(M). Jeżeli F nie zawiera znaku, to oba wycinki uznaje się za nieujemne. W przeciwnym razie przy porównaniu znak bierze się pod uwagę. (Porównanie daje w wyniku EQUAL, gdy F jest postaci (0:0), bo minus zero jest równe plus zeru).

- CMPX (porównaj X). C = 63; F = wycinek.

Operacja analogiczna do CMPA.

- CMPI (porównaj i). C = 56 + i; F = wycinek.

Operacja analogiczna do CMPA. Bajty 1, 2 i 3 rejestru indeksowego są przy porównaniu traktowane tak, jakby były zerami. (Stąd jeśli F = (1:2), to wynikiem nie może być GREATER).

Operacje skoku. Rozkazy są wykonywane w porządku sekwencyjnym. Innymi słowy, rozkaz wykonywany po rozkazie z lokacji P to zazwyczaj rozkaz z lokacji P + 1. Istnieje jednak kilka rozkazów skoku umożliwiających zmianę tego porządku. Podczas wykonania rozkazu skoku do rejestru J jest ładowany adres następnego rozkazu w pamięci (tj. adres rozkazu, który byłby wykonany w następnej kolejności, gdyby skok nie nastąpił). Za pomocą rozkazu STJ programista może ustawić pole adresu innego rozkazu, którego późniejsze wykonanie spowoduje powrót w pierwotne miejsce wykonania programu. Zawartość rejestru J jest zmieniana zawsze wtedy, gdy wystąpi skok, poza przypadkiem rozkazu JSJ. Zawartości rejestru J nie zmienia żaden rozkaz nie dotyczący skoku.

- JMP (skok). C = 39; F = 0.

Skok bezwarunkowy: następny rozkaz do wykonania zostanie pobrany z lokacji M.

- JSJ (skok, zachowaj J). C = 39; F = 1.

Tak jak JMP, z tym że zawartość rJ nie jest zmieniana.

- JOV (skok przy przepelnieniu). C = 39; F = 2.

Jeśli jest ustawiony znacznik przepelnienia, to zostaje on wyzerowany, po czym wykonywany jest skok, jak w JMP. W przeciwnym razie nic się nie dzieje.

- JNOV (skok przy braku przepelnienia). C = 39; F = 3.

Jeżeli znacznik przepelnienia jest wyzerowany, wykonywany jest skok, jak w JMP. W przeciwnym razie znacznik jest zerowany.

• JL, JE, JG, JGE, JNE, JLE (skok, gdy mniejsze, większe, większe lub równe, nierówne, mniejsze lub równe) C = 39; F = odpowiednio 4, 5, 6, 7, 8, 9.

Skok, jeśli wskaźnik porównania ma podaną wartość. Na przykład w JNE skok wykona się wtedy, gdy wskaźnik porównania ma wartość LESS lub GREATER. Wskaźnik porównania pozostaje niezmieniony.

• JAN, JAZ, JAP, JANN, JANZ, JANP (skok, gdy A ujemne, zerowe, dodatnie, nieujemne, niezerowe, niedodatnie). C = 40; F = odpowiednio 0, 1, 2, 3, 4, 5.

Gdy zawartość rA spełnia podany warunek, wykonywany jest skok, jak w JMP. W przeciwnym razie nic się nie dzieje. „Dodatni” znaczy *większy niż zero* (nie zero); „niedodatni” znaczy coś dokładnie przeciwnego, tzn. równy zero lub ujemny.

• JXN, JXZ, JXP, JXNN, JXNZ, JXNP (skok, gdy X ujemne, zerowe, dodatnie, nieujemne, niezerowe, niedodatnie). C = 47; F = odpowiednio 0, 1, 2, 3, 4, 5.

• JiN, JiZ, JiP, JiNN, JiNZ, JiNP (skok, gdy i ujemne, zerowe, dodatnie, nieujemne, niezerowe, niedodatnie). C = 40 + i; F = odpowiednio 0, 1, 2, 3, 4, 5. Operacje analogiczne do podobnych operacji dla rA.

Różne operacje.

• SLA, SRA, SLAX, SRAX, SLC, SRC (przesuń A w lewo, przesuń A w prawo, przesuń AX w lewo, przesuń AX w prawo, przesuń cyklicznie AX w lewo, przesuń cyklicznie AX w prawo). C = 6; F = odpowiednio 0, 1, 2, 3, 4, 5.

Powysze operacje to operacje „przesunięć”, w których M określa, o ile bajtów należy przesunąć wartość. M musi być nieujemne. SLA i SRA nie zmieniają

zawartości rX; pozostałe rozkazy przesunięć zmieniają zarówno zawartość rejestru A, jak i X, traktując je tak, jakby były pojedynczym rejestrem 10-bajtowym. W rozkazach SLA, SRA, SLAX i SRAX bajty „wysuwane” z rejestru znikają, natomiast z drugiej strony do rejestru „wsuwane” są zera. W rozkazach przesunięć „cyklicznych”, SLC i SRC, bajty wysuwane z rejestru są „wsuwane” z drugiej strony. W przesunięciu cyklicznym wykorzystywany jest zarówno rA, jak i rX. Rozkazy przesunięć nie zmieniają znaków w rejestrach A i X.

Przykłady:

Początkowa zawartość
 SRAX 1
 SLA 2
 SRC 4
 SRA 2
 SLC 501

+	1	2	3	4	5
+	0	1	2	3	4
+	2	3	4	0	0
+	6	7	8	9	2
+	0	0	6	7	8
+	0	6	7	8	3

Rejestr A

-	6	7	8	9	10
-	5	6	7	8	9
-	5	6	7	8	9
-	3	4	0	0	5
-	3	4	0	0	5
-	4	0	0	5	0

Rejestr X

- MOVE (kopiowanie). C = 7; F = liczba.

Liczba słów określona przez F jest kopiowana do lokacji wyznaczonej przez zawartość rI1 i następnych, spod lokacji M i następnych. Słowa przenosi się po jednym naraz, a na zakończenie operacji zawartość rI1 jest zwiększana o wartość F. Gdy F = 0, nie dzieje się nic.

Należy uważać na przypadki zachodzenia na siebie obszaru źródłowego i docelowego. Niech na przykład F = 3 i M = 1000. Wówczas, jeśli rI1 = 999, to kopujemy CONTENTS(1000) do CONTENTS(999), CONTENTS(1001) do CONTENTS(1000) i CONTENTS(1002) do CONTENTS(1001). Nic nadzwyczajnego się nie zdarzyło. Ale jeśli w rI1 byłaby wartość 1001, to kopowalibyśmy CONTENTS(1000) do CONTENTS(1001), następnie CONTENTS(1001) do CONTENTS(1002), i w końcu CONTENTS(1002) do CONTENTS(1003), więc w efekcie skopiowalibyśmy *to samo* słowo CONTENTS(1000) do trzech lokacji.

- NOP (operacja pusta). C = 0.

Rozkaz nie powoduje wykonania żadnej operacji. F i M są ignorowane.

- HLT (zatrzymanie). C = 5; F = 2.

Maszyna zatrzymuje się. W przypadku ponownego uruchomienia maszyny przez operatora wynik wykonania rozkazu jest taki sam jak rozkazu NOP.

Operacje wejścia-wyjścia. Do komputera MIX można dokupić całą gamę urządzeń wejścia-wyjścia. Każde z urządzeń ma przypisany mu numer:

Numer	Urządzenie	Rozmiar bloku
t	Pamięć taśmowa nr t ($0 \leq t \leq 7$)	100 słów
d	Dysk lub bęben nr d ($8 \leq d \leq 15$)	100 słów
16	Czytnik kart	16 słów
17	Dziurkarka kart	16 słów
18	Drukarka wierszowa	24 słów
19	Dalekopis	14 słów
20	Taśma papierowa	14 słów

Nie do każdego komputera **MIX** podłączone są wszystkie urządzenia. Będziemy czasami zakładali, że pewne urządzenia są podłączone do maszyny wykonującej program. Niektórych urządzeń nie można używać jednocześnie do odczytu i zapisu. Liczba słów wymieniona w tabeli przy każdym typie urządzenia dotyczy pojedynczego egzemplarza.

Przy odczycie lub zapisie taśmy magnetycznej, dysku lub bębna czytane/zapisywane jest całe słowo (pięć bajtów i znak). Inaczej jest z urządzeniami 16–20. Tu odczyt i zapis realizowany jest w *kodzie znakowym*, tzn. każdy bajt reprezentuje jedną literę lub cyfrę, więc na jedno słowo maszyny **MIX** składa się pięć znaków. Kod znakowy jest podany na górze tabeli 1, umieszczonej pod koniec tego rozdziału oraz na wyklejce. Kod 00 odpowiada znakowi „ ”, czyli spacji. Kody 01–29 odpowiadają literom od A do Z oraz kilku literom greckim, a kody 30–39 – cyfrom 0, 1, …, 9. Większe liczby 40, 41, … to kody znaków interpunkcyjnych i innych. (Zbiór znaków komputera **MIX** pochodzi z czasów, gdy komputery nie radziły sobie z małymi literami). Nie możemy używać kodów znakowych do wczytywania lub zapisywania wszystkich możliwych wartości bajtu; niektóre kombinacje są niezdefiniowane. Pewne urządzenia wejścia-wyjścia mogą ponadto nie obsługiwać wszystkich znaków, na przykład znaki Σ oraz Π zapewne nie będą akceptowane przez czytnik kart. Przy wczytywaniu danych w trybie znakowym arytmetyczny znak słów jest ustawiany na +; przy wyprowadzaniu danych w tym trybie znak arytmetyczny jest ignorowany. Jeżeli do wprowadzania danych używa się dalekopisu, naciśnięcie klawisza „powrót karetki” powoduje wypełnienie reszty wiersza spacjami.

Dyski i bębny są urządzeniami pamięci zewnętrznej zawierającymi 100-słowne bloki danych. Rozkazy IN, OUT i IOC dotyczą bloku, którego adres znajduje się w rX. Wartość w rX nie powinna przekraczać pojemności dysku lub bębna, którego dotyczy operacja.

- IN (wejście). C = 36; F = urządzenie.

Rozkaz zapoczątkowuje transmisję danych z urządzenia wejściowego do kolejnych lokacji pamięci, począwszy od M. Liczba transmitowanych bajtów jest równa rozmiarowi bloku dla konkretnego urządzenia (zobacz tabela powyżej). Wykonanie operacji zostanie wstrzymane do czasu zakończenia poprzedniej operacji na tym samym urządzeniu. Transmisja zapoczątkowana przez rozkaz będzie trwała przez pewien czas, zależny od szybkości urządzenia, zatem program nie powinien polegać na wczytanej informacji do momentu zakończenia operacji. Z taśmy magnetycznej nie należy odczytywać bloku leżącego dalej niż ostatni zapisany blok.

- OUT (wyjście). C = 37; F = urządzenie.

Rozkaz zapoczątkowuje transmisję danych z pamięci, począwszy od lokacji M, do podanego urządzenia wyjściowego. Jeżeli urządzenie nie jest gotowe, wykonanie operacji zostanie wstrzymane do momentu uzyskania przez nie gotowości. Transmisja zakończy się po pewnym czasie, zależnym od szybkości urządzenia, zatem program nie powinien przez ten czas zmieniać zawartości zapisywanego fragmentu pamięci.

- IOC (sterowanie wejścia-wyjścia). C = 35; F = urządzenie.

Rozkaz powoduje wstrzymanie wykonywania programu do czasu zakończenia przez urządzenie aktualnej operacji. Następnie wykonywana jest operacja sterująca, której konkretny wynik zależy od typu urządzenia. W książce korzystamy z operacji dotyczących:

Taśmy magnetycznej: M = 0 oznacza rozkaz przewinięcia taśmy. M < 0 oznacza rozkaz cofnięcia taśmy o M bloków (lub do napotkania początku). M > 0 oznacza rozkaz przesunięcia taśmy w przód. Nie należy przesuwać taśmy za ostatni zapisany blok.

Na przykład, ciąg rozkazów „OUT 1000(3); IOC -1(3); IN 2000(3)” powoduje zapisanie stu słów na taśmie 3, a następnie wczytanie ich z powrotem do pamięci. Jeżeli nie uwzględniamy niezawodności taśmy, to ostatnie dwie instrukcje powodują po prostu powolne skopiowanie słów 1000–1099 do lokacji 2000–2099. Ciąg „OUT 1000(3); IOC +1(3)” jest niedopuszczalny.

Dysku lub bębna: M powinno być równe zeru. W wyniku wykonania operacji urządzenie jest ustawione zgodnie z zawartością rX, tak że następne wykonanie na nim operacji IN lub OUT zajmie mniej czasu, o ile będzie dotyczyło bloku o numerze rX.

Drukarki wierszowej: M powinno być równe zeru. „IOC 0(18)” powoduje przesunięcie papieru do początku nowej strony.

Taśmy papierowej: M powinno być równe zeru. „IOC 0(20)” powoduje przewinięcie taśmy.

- JRED (skok, gdy gotowe). C = 38; F = urządzenie.

Rozkaz powoduje wykonanie skoku, jeżeli podane urządzenie jest gotowe, tj. zakończyło operację zapoczątkowaną przez IN, OUT lub IOC.

- JBUS (skok, gdy zajęte). C = 34; F = urządzenie.

Rozkaz podobny do JRED, z tym że skok następuje, gdy urządzenie *nie* jest gotowe.

Przykład: Rozkaz „JBUS 1000(16)” zapisany w komórce pamięci nr 1000 będzie wykonywany w kółko, dopóki urządzenie 16 nie osiągnie stanu gotowości.

Tymi prostymi operacjami kończymy przegląd repertuaru rozkazów wejścia-wyjścia maszyny MIX. Nie mamy na przykład „wskaźnika stanu taśmy” do obsługi sytuacji wyjątkowych dotyczących urządzeń wejścia-wyjścia. Każde takie kłopotliwe zdarzenie (tj. „zakleszczenie” papieru, urządzenie wyłączone, koniec taśmy) jest interpretowane jako pozostawanie urządzenia w stanie zajętości. Jednocześnie dzwonek wzywa operatora, który przybiega i opanowuje sytuację. Pewne bardziej złożone urządzenia wejścia-wyjścia, droższe i bardziej przypominające sprzęt współczesny, są omówione w punktach 5.4.6 i 5.4.9.

Rozkazy konwersji.

- NUM (konwersja na liczbę). C = 5; F = 0.

Operacja powoduje zamianę znaku na jego kod liczbowy. Wartość M jest ignorowana. Rejestry A i X zawierają 10-bajtową liczbę w kodzie znakowym. Wartość tej liczby (w interpretacji dziesiętnej) zostaje w wyniku wykonania rozkazu NUM

zapisana do rA. Wartość rX oraz znak rA pozostają niezmienione. Bajty o wartościach 00, 10, 20, 30, 40, ... są traktowane jak cyfra zero; bajty o wartościach 01, 11, 21, ... jak cyfra jeden itd. Może wystąpić przepełnienie i w takim przypadku w rejestrze pozostaje reszta modulo b^5 , gdzie b jest liczbą różnych wartości, które może przyjmować bajt.

- CHAR (konwersja na znak). C = 5; F = 1.

Operacja jest stosowana do zamiany liczby na postać znakową, dzięki czemu liczbę można wyprowadzić na dziurkarkę kart, taśmę lub drukarkę wierszową. Zawartość rA jest zamieniana na 10-bajtową liczbę będącą reprezentacją początkowej wartości rA w kodzie znakowym. Znaki rA i rX pozostają niezmienione. Wartość M jest ignorowana.

Przykłady:

	Rejestr A						Rejestr X					
Zawartość początkowa	-	00	00	31	32	39	+ 37	57	47	30	30	
NUM 0	-					12977700	+ 37	57	47	30	30	
INCA 1	-					12977699	+ 37	57	47	30	30	
CHAR 0	-	30	30	31	32	39	+ 37	37	36	39	39	

Czasy wykonania rozkazów. Jeśli chcemy ilościowo badać wydajność programów działających na komputerze MIX, musimy każdej operacji przypisać *czas wykonania*, odzwierciedlający czas wykonania podobnych rozkazów na klasycznych komputerach z lat siedemdziesiątych.

ADD, SUB, wszystkie operacje LD?, wszystkie operacje ST? (włącznie z STZ), wszystkie operacje przesunięć i wszystkie operacje porównania potrzebują *dwoch jednostek* czasu. MOVE potrzebuje jednej jednostki plus dwie jednostki na każde kopiowane słowo. Każda z operacji MUL, NUM, CHAR wymaga 10 jednostek, a DIV – 12. Czasy wykonania operacji zmienopozycyjnych są określone w punkcie 4.2.1. Wszystkie pozostałe operacje wymagają jednej jednostki, plus czas wstrzymania przy instrukcjach IN, OUT, IOC i HLT.

Zauważmy, że w szczególności wykonanie rozkazu ENTA zabiera jedną jednostkę czasu, podczas gdy wykonanie LDA – dwie jednostki. Zasady dotyczące czasów wykonania łatwo jest zapamiętać, ponieważ za wyjątkiem operacji przesunięć, konwersji, MUL oraz DIV, liczba jednostek czasu potrzebna na wykonanie rozkazu równa się liczbie odwołań do pamięci (wliczając odwołanie na pobranie kodu instrukcji).

Jednostka czasu na komputerze MIX jest wielkością względną. Będziemy ją po prostu oznaczać przez u . Można przyjąć, że u równa się 10 mikrosekund (na niedrogich modelach) lub 10 nanosekund (na modelach raczej kosztownych).

Przykład: wykonanie LDA 1000; INCA 1; STA 1000 zajmuje dokładnie $5u$.

*And now I see with eye serene
The very pulse of the machine.*

— WILLIAM WORDSWORTH,
She Was a Phantom of Delight (1804)

Tabela 1

Kod znaku:		00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24		
		□ A B C D E F G H I Δ J K L M N O P Q R Σ Π S T U		
00	1	01 2	02 2	03 10
Nie rób nic		rA \leftarrow rA + V	rA \leftarrow rA - V	rAX \leftarrow rA \times V
NOP(0)		ADD(0:5) FADD(6)	SUB(0:5) FSUB(6)	MUL(0:5) FMUL(6)
08	2	09 2	10 2	11 2
rA \leftarrow V		rI1 \leftarrow V	rI2 \leftarrow V	rI3 \leftarrow V
LDA(0:5)		LD1(0:5)	LD2(0:5)	LD3(0:5)
16	2	17 2	18 2	19 2
rA \leftarrow -V		rI1 \leftarrow -V	rI2 \leftarrow -V	rI3 \leftarrow -V
LDAN(0:5)		LD1N(0:5)	LD2N(0:5)	LD3N(0:5)
24	2	25 2	26 2	27 2
M(F) \leftarrow rA		M(F) \leftarrow rI1	M(F) \leftarrow rI2	M(F) \leftarrow rI3
STA(0:5)		ST1(0:5)	ST2(0:5)	ST3(0:5)
32	2	33 2	34 1	35 1 + T
M(F) \leftarrow rJ		M(F) \leftarrow 0	Urządzenie F zajęte?	Sterowanie urządzeniem F
STJ(0:2)		STZ(0:5)	JBUS(0)	IOC(0)
40	1	41 1	42 1	43 1
rA : 0, skok		rI1 : 0, skok	rI2 : 0, skok	rI3 : 0, skok
JA [+]		J1 [+]	J2 [+]	J3 [+]
48	1	49 1	50 1	51 1
rA \leftarrow [rA]? \pm M INCA(0) DECA(1) ENTA(2) ENNA(3)		rI1 \leftarrow [rI1]? \pm M INC1(0) DEC1(1) ENT1(2) ENN1(3)	rI2 \leftarrow [rI2]? \pm M INC2(0) DEC2(1) ENT2(2) ENN2(3)	rI3 \leftarrow [rI3]? \pm M INC3(0) DEC3(1) ENT3(2) ENN3(3)
56	2	57 2	58 2	59 2
CI \leftarrow rA(F) : V CMPA(0:5) FCMP(6)		CI \leftarrow rI1(F) : V CMP1(0:5)	CI \leftarrow rI2(F) : V CMP2(0:5)	CI \leftarrow rI3(F) : V CMP3(0:5)

Legenda:

C	t
Opis	
OP(F)	

C = kod operacji, wycinek (5:5)

F = modyfikator operacji, wycinek (4:4)

M = adres po indeksowaniu

V = M(F) = zawartość wycinka F lokacji M

OP = nazwa symboliczna operacji

(F) = standardowa wartość F

t = czas wykonania; T = czas wstrzymania

25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
V	W	X	Y	Z	0	1	2	3	4	5	6	7	8	9	.	,	()	+	-	*	/	=	\$	<	>	@	:	'	

04	12	05	10	06	2	07	1 + 2F
rA \leftarrow rAX/V rX \leftarrow reszta DIV(0:5) FDIV(6)	Specjalne NUM(0) CHAR(1) HLT(2)	Przesuń o M bajtów SLA(0) SRA(1) SLAX(2) SRAX(3) SLC(4) SRC(5)	Kopiuj F słów z M do rI1 MOVE(1)				
12	2	13	2	14	2	15	2
rI4 \leftarrow V LD4(0:5)	rI5 \leftarrow V LD5(0:5)	rI6 \leftarrow V LD6(0:5)	rX \leftarrow V LDX(0:5)				
20	2	21	2	22	2	23	2
rI4 \leftarrow -V LD4N(0:5)	rI5 \leftarrow -V LD5N(0:5)	rI6 \leftarrow -V LD6N(0:5)	rX \leftarrow -V LDXN(0:5)				
28	2	29	2	30	2	31	2
M(F) \leftarrow rI4 ST4(0:5)	M(F) \leftarrow rI5 ST5(0:5)	M(F) \leftarrow rI6 ST6(0:5)	M(F) \leftarrow rX STX(0:5)				
36	1 + T	37	1 + T	38	1	39	1
Wejście, urządzenie F IN(0)	Wyjście, urządzenie F OUT(0)	Urządzenie F gotowe? JRED(0)	Skoki JMP(0) JSJ(1) JOV(2) JNOV(3) też [*] poniżej				
44	1	45	1	46	1	47	1
rI4:0, skok J4[+]	rI5:0, skok J5[+]	rI6:0, skok J6[+]	rX:0, skok JX[+]				
52	1	53	1	54	1	55	1
rI4 \leftarrow [rI4]? \pm M INC4(0) DEC4(1) ENT4(2) ENN4(3)	rI5 \leftarrow [rI5]? \pm M INC5(0) DEC5(1) ENT5(2) ENN5(3)	rI6 \leftarrow [rI6]? \pm M INC6(0) DEC6(1) ENT6(2) ENN6(3)	rX \leftarrow [rX]? \pm M INCX(0) DECX(1) ENTX(2) ENNX(3)				
60	2	61	2	62	2	63	2
CI \leftarrow rI4(F) : V CMP4(0:5)	CI \leftarrow rI5(F) : V CMP5(0:5)	CI \leftarrow rI6(F) : V CMP6(0:5)	CI \leftarrow rX(F) : V CMPX(0:5)				

[*]: [+]:

rA = rejestr A	JL(4) < N(0)
rX = rejestr X	JE(5) = Z(1)
rAX = połączone rA i rX	JG(6) > P(2)
rI _i = rejestr indeksowy i , $1 \leq i \leq 6$	JGE(7) \geq NN(3)
rJ = rejestr J	JNE(8) \neq NZ(4)
CI = wskaźnik porównania	JLE(9) \leq NP(5)

Podsumowanie. Omówiliśmy wszystkie cechy komputera MIX, poza „przyciskiem START”, który pojawia się w ćwiczeniu 26. Chociaż MIX ma blisko 150 różnych rozkazów, układają się one w proste do zapamiętania wzorce. Tabela 1 zawiera zestawienie operacji dla wszystkich wartości C. W nawiasach za nazwą operacji podano standardową zawartość wycinka F.

Poniższe ćwiczenia posłużą do szybkiego powtórzenia materiału z tego podrozdziału. Ćwiczenia są raczej proste i Czytelnik powinien spróbować rozwiązać prawie wszystkie.

ĆWICZENIA

1. [00] Gdyby MIX był komputerem trójkowym (o podstawie liczenia 3), ile „tritów” przypadałoby na bajt?
2. [02] Pewna wartość może wynosić nawet 99999999. Ilu bajtów należy użyć do reprezentowania jej na komputerze MIX?
3. [02] Podaj specyfikacje wycinków słowa, (L:R), dla (a) pola adresu, (b) pola indeksu, (c) pola F i (d) kodu operacji w kodzie maszynowym rozkazu komputera MIX.
4. [00] Ostatni przykład w (5) to „LDA -2000,4”. Czy to jest poprawne, skoro adresy pamięci nie mogą być ujemne?
5. [10] Jaka notacja symboliczna, analogiczna do (4), odpowiada (6), jeśli (6) zinterpretować jako kod rozkazu komputera MIX?
- 6. [10] Założmy, że lokacja 3000 zawiera

+	5	1	200	15	.
---	---	---	-----	----	---

Jaki jest wynik wykonania następującego rozkazu (podaj, czy i co jest niezdefiniowane lub częściowo niezdefiniowane): (a) LDAN 3000; (b) LD2N 3000(3:4); (c) LDX 3000(1:3); (d) LD6 3000; (e) LDXN 3000(0:0).

7. [M15] Podaj dokładną definicję rezultatu wykonania instrukcji DIV dla wszystkich przypadków, w których nie występuje przepełnienie. Skorzystaj z oznaczeń X mod Y i $[X/Y]$.

8. [15] W ostatnim przykładzie zastosowania instrukcji DIV na stronie 137 wartość „rX przed” równa się

+	1235	0	3	1
---	------	---	---	---

. Jeśli zamiast tej wartości byłoby tam

-	1234	0	3	1
---	------	---	---	---

 (cała reszta przykładu bez zmian), to jakie wartości zawierałyby rejestrze A i X po wykonaniu rozkazu DIV?

- 9. [15] Wypisz wszystkie rozkazy maszyny MIX, które mogą zmienić stan znacznika przepełnienia. (Pomiń operacje zmiennopozycyjne).
- 10. [15] Wypisz wszystkie rozkazy maszyny MIX, które mogą zmienić wartość wskaźnika porównania.
- 11. [15] Wypisz wszystkie rozkazy maszyny MIX, które mogą zmienić wartość rI1.
- 12. [10] Napisz pojedynczy rozkaz, którego wykonanie spowoduje pomnożenie aktualnej zawartości rI3 przez dwa i pozostawienie wyniku w rI3.
- 13. [10] Przypuśćmy, że lokacja 1000 zawiera rozkaz „J0V 1001”. Rozkaz ten powoduje wyłączenie znacznika przepełnienia, jeżeli był włączony. (Następny rozkaz do wykonania zostanie pobrany z lokacji 1001, niezależnie od stanu znacznika). Co by

się zmieniło, gdybyśmy zamiast tego rozkazu wzięli „JNOV 1001? A „JOV 1000”, albo „JNOV 1000”?

14. [20] Zbadaj, czy dla każdego rozkazu maszyny MIX można tak dobrać $\pm AA$, I, i F, by był równoważny rozkazowi NOP (nie licząc czasu wykonania). Załóż, że nic nie wiadomo na temat zawartości rejestrów oraz pamięci. *Przykłady:* Rozkaz INCA jest równoważny NOP, gdy pole adresu i pole indeksu są równe zeru. Rozkaz JMP nie może nigdy być równoważny NOP, bo zawsze zmienia zawartość rJ.

15. [10] Ile znaków alfanumerycznych mieści blok dalekopisu lub taśmy papierowej? Ile blok czytnika/dziurkarki kart? Ile blok drukarki wierszowej?

16. [20] Napisz program, który wypełnia zerami komórki pamięci 0000–0099 i jest (a) najkrótszy z możliwych; (b) najszybszy z możliwych. [*Wskazówka:* Zastanów się nad wykorzystaniem rozkazu MOVE].

17. [26] Wykonaj takie polecenie, jak w poprzednim ćwiczeniu, z tym że mają zostać wyzerowane lokacje od 0000 do N włącznie, gdzie N jest bieżącą zawartością rI2. Twój program powinien działać dla dowolnej wartości $0 \leq N \leq 2999$ i powinien zaczynać się w komórce pamięci o numerze 3000.

- **18.** [22] Jak zmienią się rejestrzy, znaczniki i zawartość pamięci po wykonaniu poniższego „programu jedynkowego”? (Na przykład, jaka jest końcowa wartość w rejestrze rI1? A w rX? Co ze znacznikiem przepelnienia i wskaźnikiem porównania?)

```

STZ 1
ENN 1
STX 1(0:1)
SLAX 1
ENNA 1
INCX 1
ENT1 1
SRC 1
ADD 1
DEC1 -1
STZ 1
CMPA 1
MOVE -1,1(1)
NUM 1
CHAR 1
HLT 1  ■

```

- **19.** [14] Jaki jest czas wykonania programu z poprzedniego ćwiczenia, jeżeli pominie my rozkaz HLT?
- 20.** [20] Napisz program, który do *wszystkich* komórek pamięci wpisuje kod rozkazu „HLT”, a następnie zatrzymuje się.
- **21.** [24] (a) Czy w rejestrze J kiedykolwiek może pojawić się zero? (b) Napisz program, który dla liczby N w rI4 powoduje wpisanie liczby N do rejestru J. Załóż, że $0 < N \leq 3000$. Twój program powinien zaczynać się w lokacji 3000. Zawartość pamięci po wykonaniu programu, powinna być taka, jak przed wykonaniem programu.
- **22.** [28] Lokacja 2000 zawiera liczbę całkowitą X . Napisz dwa programy obliczające X^{13} i zatrzymujące się z wynikiem w rejestrze A. Jeden program powinien używać jak najmniejszej liczby komórek pamięci, drugi powinien być jak najszybszy. Przyjmij, że X^{13} mieści się w jednym słowie.

23. [27] Lokacja 0200 zawiera słowo

+	a	b	c	d	e	;
---	---	---	---	---	---	---

napisz dwa programy, które obliczają słowo „odwrócone”

+	e	d	c	b	a
---	---	---	---	---	---

i zatrzymują się z wynikiem w rejestrze A. Jeden z Twoich programów powinien być napisany bez wykorzystania mechanizmu wycinków. Oba programy powinny zużywać jak najmniej komórek pamięci (wliczając komórki potrzebne do zapamiętania programu oraz wyników pośrednich).

24. [21] Zakładając, że rejesty A i X zawierają odpowiednio

+	0	a	b	c	d	i	+	e	f	g	h	i	,
---	---	---	---	---	---	---	---	---	---	---	---	---	---

napisz dwa warianty programu, który zmienia zawartość tych rejestrów na

+	a	b	c	d	e	i	+	0	f	g	h	i	,
---	---	---	---	---	---	---	---	---	---	---	---	---	---

przy czym (a) zużywa jak najmniej pamięci, (b) potrzebuje jak najmniej czasu.

- **25.** [30] Producent komputera **MIX** chciałby wejść na rynek z nowszą, silniejszą maszyną („Mixmaster”?) i pragnie przekonać aktualnych użytkowników komputera **MIX**, by zainwestowali w szybszy komputer. Producent chce tak zaprojektować nowy model, żeby był on *rozszerzeniem* komputera **MIX**, tzn. by wszystkie programy napisane na komputer **MIX** mogły bez zmian działać na nowym komputerze. Zaproponuj innowacje, które mógłby wprowadzić producent. (Na przykład, czy można jakoś lepiej wykorzystać pole I w kodzie instrukcji?)
- **26.** [32] Celem tego ćwiczenia jest napisanie programu do wczytywania danych z kart perforowanych. Każdy komputer ma swoje dziwaczne problemy związane z umieszczaniem w pamięci pierwszego programu (ang. *bootstrapping*). W przypadku komputera **MIX** zawartość kart może być czytana wyłącznie w kodzie znakowym. Karty zawierające sam program ładujący też podlegają tej zasadzie. Niektóre wartości bajtów nie mogą być zatem wczytane z kart, a wszystkie wczytywane słowa mają dodatni znak.

MIX ma jedną cechę, której nie omówiliśmy w tekście. Maszyna ma przycisk „START”. Naciśnięcie przycisku powoduje uruchomienie komputera „od zera”, tj. wykonanie następujących operacji:

- 1) Do komórek 0000 – 0015 wczytywana jest jedna karta perforowana; jest to równoważne wykonaniu instrukcji „IN 0(16)”.
- 2) Po zakończeniu wczytywania karty, w chwili przejścia czytnika do stanu gotowości, wykonywany jest skok pod adres 0000. Rejestr J jest też ustawiany na zero.
- 3) Komputer rozpoczyna wykonywanie programu wczytanego z karty.

Uwaga: Komputery **MIX**, do których nie podłączono czytnika kart, wczytują program startowy z innego urządzenia. Na potrzeby tego ćwiczenia zakładamy, że do komputera jest podłączony czytnik kart (urządzenie 16).

Należy zatem napisać procedurę ładującą, zgodnie z następującymi wytycznymi:

- i) Plik kart ma zaczynać się procedurą ładującą, po której powinny następować karty informacyjne zawierające liczby do wczytania. Wsad powinien kończyć się „kartą transferową”. Wczytanie tej karty ma powodować zakończenie procedury ładującej

i skok do wczytanego programu. Procedura ładowająca powinna mieścić się na dwóch kartach.

ii) Karty informacyjne mają następujący format:

Kolumny 1–5: ignorowane przez procedurę ładowającą.

Kolumna 6: liczba kolejnych słów do załadowania z bieżącej karty (od 1 do 7 włącznie).

Kolumny 7–10: lokacja pierwszego słowa, zawsze większa od 100 (żeby nie zamazać procedury ładowającej).

Kolumny 11–20: pierwsze słowo.

Kolumny 21–30: drugie słowo (jeśli wartość w kolumnie 6 jest ≥ 2).

...

Kolumny 71–80: siódme słowo (jeśli wartość w kolumnie 6 jest = 7).

Słowa 1, 2, ... są wydziurkowane w postaci liczb dziesiętnych. Jeżeli słowo ma mieć wartość ujemną, nad ostatnią cyfrą znaczącą (tj. w 20 kolumnie) *naddziurkowujemy* minus (dziurki „11”). Zakładamy, że w wyniku tej operacji kod znakowy wczytanej cyfry wynosi 10, 11, 12, ..., 19 zamiast 30, 31, 32, ..., 39. Jeśli na przykład karta w kolumnach 1–40 ma wydziurkowany ciąg

ABCDE310000123456789000000000010000000100

to zostaną wczytane następujące dane:

1000: +0123456789; 1001: +0000000001; 1002: -0000000100.

iii) Karta transferowa w kolumnach 1–10 ma wydziurkowany napis TRANSOnnnn, gdzie nnn jest lokacją, od której należy rozpocząć wykonanie programu.

iv) Program ładowający powinien być tak napisany, by tych samych kart z procedurą ładowającą można było używać na różnych egzemplarzach komputera MIX, niezależnie od rozmiaru bajtu. Żadna karta nie powinna zawierać jakiegokolwiek ze znaków odpowiadających bajtom o wartościach 20, 21, 48, 49, 50, ... (tj. znaków Σ , Π , $=$, $$$, $<$, ...), gdyż pewne czytniki nie potrafią ich odczytać. W szczególności nie można używać instrukcji ENT, INC i CMP.

1.3.2. Asembler komputera MIX

Zastosowanie języka symbolicznego czyni programy na komputer MIX łatwiejszymi do czytania i pisania. Oszczędza także programistę wiele nudnej i żmudnej roboty, sprzyjającej powstawaniu błędów. Język symboliczny MIXAL („MIX Assembly Language”) jest rozszerzeniem notacji, z której korzystaliśmy, zapisując rozkazy w poprzednim rozdziale. Główną cechą tego języka jest możliwość wykorzystania znakowych nazw symbolizujących liczby oraz wiązania nazw z adresami pamięci.

Czytelnikowi najłatwiej będzie przyswoić język MIXAL, analizując prosty przykład. Poniższy kod jest fragmentem większego programu, konkretnie podprogramem wyszukującym największy z n elementów $X[1], \dots, X[n]$, zgodnie z algorytmem 1.2.10M.

Program M (Znajdowanie maksimum). Przydział rejestrów: rA $\equiv m$, rI1 $\equiv n$, rI2 $\equiv j$, rI3 $\equiv k$, $X[i] \equiv \text{CONTENTS}(X + i)$.

	Przetłumaczone instrukcje	Nr wiersza	LOC	OP	ADRES	Czas	Komentarz
		01	X	EQU	1000		
		02		ORIG	3000		
3000:	+ 3009 0 2 32	03	MAXIMUM	STJ	EXIT	1	Ustaw adres powrotu
3001:	+ 0 1 2 51	04	INIT	ENT3	0,1	1	M1. Inicjowanie. $k \leftarrow n$.
3002:	+ 3005 0 0 39	05		JMP	CHANGEM	1	$j \leftarrow n, m \leftarrow X[n], k \leftarrow n - 1$.
3003:	+ 1000 3 5 56	06	LOOP	CMPA	X,3	$n - 1$	M3. Porównywanie.
3004:	+ 3007 0 7 39	07		JGE	*+3	$n - 1$	Do M5, gdy $m \geq X[k]$.
3005:	+ 0 3 2 50	08	CHANGEM	ENT2	0,3	$A + 1$	M4. Zmiana $m. j \leftarrow k$.
3006:	+ 1000 3 5 08	09		LDA	X,3	$A + 1$	$m \leftarrow X[k]$.
3007:	+ 1 0 1 51	10		DEC3	1	n	M5. Zmniejszanie k .
3008:	+ 3003 0 2 43	11		J3P	LOOP	n	M2. Wszystko? Do M3, gdy $k > 0$.
3009:	+ 3009 0 0 39	12	EXIT	JMP	*	1	Powrót do programu głównego. ■

Przyjrzyjmy się paru szczegółom:

- a) Zasadniczo interesują nas kolumny „LOC”, „OP” i „ADRES”. Te pola składają się na wiersz w symbolicznym języku maszynowym **MIXAL**. Szczegóły wyjaśniamy poniżej.
- b) Kolumna „Przetłumaczone instrukcje” zawiera liczby składające się na program maszynowy odpowiadający programowi w języku **MIXAL**. **MIXAL** został tak zaprojektowany, że każdy napisany w nim program daje się łatwo przetłumaczyć na liczbowy kod maszynowy. Tłumaczeniem zajmuje się zazwyczaj inny program, nazywany *programem asemblera* lub *asemblerem*. Programiści mogą zatem wszystkie swoje programy maszynowe pisać w języku **MIXAL**, nie przejmując się koniecznością ręcznego tłumaczenia ich na kod maszynowy. W tej książce prawie wszystkie programy na komputer **MIX** są napisane w języku **MIXAL**.
- c) Kolumna „Nr wiersza” nie stanowi części programu. W przykładach zamieszczamy numery wierszy, by móc się sprawniej odwoływać do konkretnych fragmentów.
- d) Kolumna „Komentarz” zawiera opisową informację dotyczącą programu, wiążącą kod z krokami algorytmu 1.2.10M. Czytelnik powinien porównać program z algorytmem. Zauważmy, że przy kodowaniu algorytmu pozwoliliśmy sobie na małą „licentia programistica” – na przykład krok M2 został umieszczony na końcu. „Przydział rejestrów” na początku programu M wyjaśnia sposób przyporządkowania elementów maszyny **MIX** zmiennym algorytmu.
- e) Kolumna „Czas” jest w wielu przypadkach bardzo pouczająca. Kolumna zawiera *profil*, czyli liczbę wykonania rozkazów (wierszy) podczas wykonania programu. Na przykład rozkaz w wierszu 06 zostanie wykonany $n - 1$ razy. Na podstawie tych informacji możemy określić czas działania podprogramu. Tu wynosi on $(5 + 5n + 3A)u$, gdzie A jest wielkością omawianą w punkcie 1.2.10.

Zajmiemy się teraz właściwym tekstem programu M w języku **MIXAL**. Wiersz 01, X EQU 1000, mówi, że symbol X ma być *równoważny* liczbie 1000. Wynik tej deklaracji jest widoczny w wierszu 06, gdzie liczbową postać rozkazu „CMPA X,3” figuruje jako

+	1000	3	5	56	,
---	------	---	---	----	---

tj. „CMPA 1000,3”.

Wiersz 02 oznacza, że lokacje następujących po nim wierszy powinny być przydzielane po kolej, poczynając od 3000. Z tego powodu symbol **MAXIMUM** w polu LOC wiersza 03 jest równoważny liczbie 3000, **INIT** jest równoważny 3001, **LOOP** jest równoważny 3003 itd.

W wierszach 03–12 pole OP zawiera nazwy symboliczne rozkazów maszyny MIX: **STJ**, **ENT3** itp. Symbole **EQU** i **ORIG**, występujące w wierszach 01 i 02, mają inną naturę. **EQU** i **ORIG** nazywane są *pseudooperacjami*, gdyż należą do języka MIXAL, ale nie do zbioru rozkazów komputera MIX. Pseudooperacje niosą pewne informacje o programie symbolicznym, ale nie są rozkazami programu. Wiersz

```
X EQU 1000
```

zawiera *deklarację* dotyczącą programu M, a w szczególności *nie* powoduje przypisania wartości 1000 podczas wykonania programu. Zauważmy, że w wyniku tłumaczenia wierszy 01 i 02 nie są generowane żadne rozkazy.

Wiersz 03 to rozkaz „zapisz J w pamięci”, który powoduje załadowanie zawartości rejestru J do wycinka (0:2) lokacji EXIT. Innymi słowy, zapisanie zawartości rJ w polu adresu rozkazu z wiersza 12.

Jak zaznaczyliśmy wcześniej, program M ma stanowić część większego programu, w którym na przykład instrukcje

```
ENT1 100
JMP MAXIMUM
STA MAX
```

powodują skok do (pod)programu M z wartością $n = 100$. Program M znajduje największy z elementów $X[1], \dots, X[100]$ i zwraca sterowanie do rozkazu „**STA MAX**”, w rA przekazując wartość, a w rI2=j pozycję największego elementu. (Zobacz ćwiczenie 3).

W wierszu 05 sterowanie jest przenoszone do wiersza 08. Wiersze 04, 05, 06 nie wymagają wyjaśnień. W wierszu 07 wprowadzamy nową notację: gwiazdka (czytaj: „bieżący”) oznacza lokację przydzieloną wierszowi, w którym się ona znajduje. „*+3” („bieżący plus trzy”) oznacza lokację położoną trzy adresy za bieżącym wierszem. Ponieważ wierszowi 07 przydzielono lokację 3004, umieszczony tam napis „*+3” oznacza lokację 3007.

Reszta kodu wyjaśnień nie wymaga. Zauważmy, że gwiazdka pojawia się ponownie w wierszu 12 (zobacz ćwiczenie 2).

Nasz następny przykład demonstruje kilka nowych cech języka asemblerowego. Celem programu jest obliczenie i wydrukowanie tablicy pierwszych 500 liczb pierwszych, w 10 kolumnach po 50 liczb. Na drukarce wierszowej tablica powinna wyglądać następująco:

FIRST FIVE HUNDRED PRIMES

0002	0233	0547	0877	1229	1597	1993	2371	2749	3187
0003	0239	0557	0881	1231	1601	1997	2377	2753	3191
0005	0241	0563	0883	1237	1607	1999	2381	2767	3203
0007	0251	0569	0887	1249	1609	2003	2383	2777	3209
:									:
0229	0541	0863	1223	1583	1987	2357	2741	3181	3571

Skorzystamy z następującej metody:

Algorytm P (*Drukowanie tablicy 500 liczb pierwszych*). Algorytm składa się z dwóch części: w krokach P1–P8 przygotowujemy tablicę 500 liczb pierwszych, a w krokach P9–P11 drukujemy tablicę w powyższej postaci. Druga część korzysta z dwóch „buforów”, w których formowany jest obraz. W czasie gdy zawartość jednego z buforów wysyłana jest na drukarkę, program zajmuje się wypełnianiem drugiego bufora.

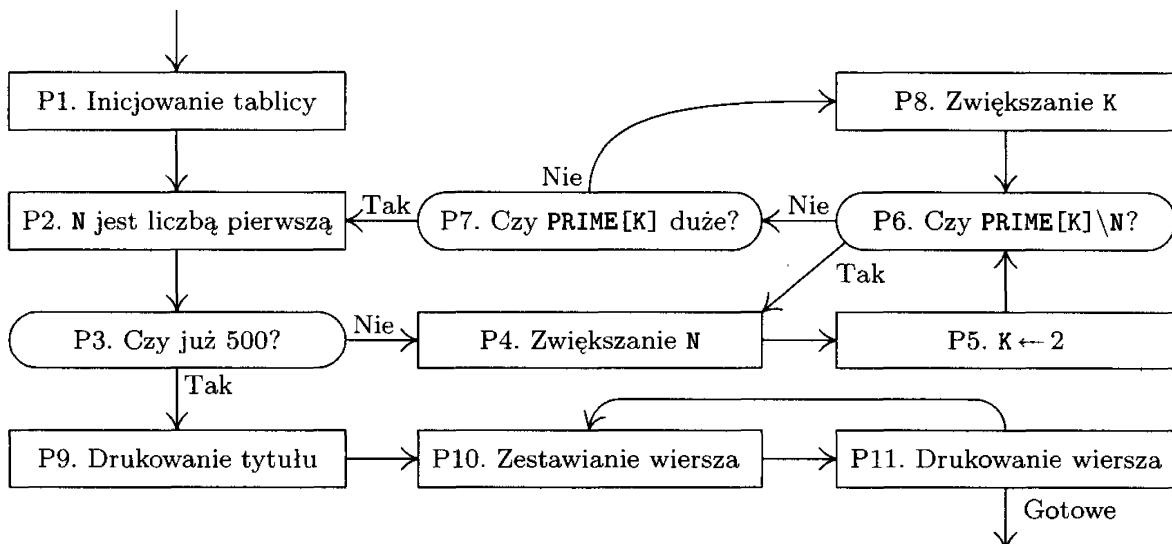
- P1. [Inicjowanie tablicy] Przyjmij $\text{PRIME}[1] \leftarrow 2$, $N \leftarrow 3$, $J \leftarrow 1$. (W programie N przebiega liczby nieparzyste, które są kandydatkami na liczby pierwsze. W J przechowujemy informacje, ile do tej pory znaleziono liczb pierwszych).
- P2. [N jest liczbą pierwszą] Przyjmij $J \leftarrow J + 1$, $\text{PRIME}[J] \leftarrow N$.
- P3. [Czy już 500?] Jeśli $J = 500$, to przejdź do kroku P9.
- P4. [Zwiększanie N] $N \leftarrow N + 2$.
- P5. [$K \leftarrow 2$] Przyjmij $K \leftarrow 2$. ($\text{PRIME}[K]$ będzie przebiegać możliwe dzielniki pierwsze liczby N).
- P6. [Czy $\text{PRIME}[K] \mid N$?] Podziel N przez $\text{PRIME}[K]$; niech Q oznacza iloraz, a R resztę. Jeśli $R = 0$ (zatem N nie jest pierwsza), przejdź do P4.
- P7. [Czy $\text{PRIME}[K]$ duże?] Jeśli $Q \leq \text{PRIME}[K]$, przejdź do P2. (W takim przypadku N musi być liczbą pierwszą. Dowód tego faktu jest ciekawy i nietuzinkowy – zobacz ćwiczenie 6).
- P8. [Zwiększenie K] Zwiększ K o 1, przejdź do P6.
- P9. [Drukowanie tytułu] Jesteśmy gotowi do drukowania tablicy. Przesuń papier do nowej strony. Zapisz wiersz tytułowy do $\text{BUFFER}[0]$ i wydrukuj go. Przyjmij $B \leftarrow 1$, $M \leftarrow 1$.
- P10. [Zestawianie wiersza] Zapisz $\text{PRIME}[M], \text{PRIME}[50+M], \dots, \text{PRIME}[450+M]$ do $\text{BUFFER}[B]$ w odpowiednim formacie.
- P11. [Drukowanie wiersza] Drukuj $\text{BUFFER}[B]$; przyjmij $B \leftarrow 1 - B$ (w ten sposób przełączamy się na drugi bufor), zwiększ M o 1. Jeśli $M \leq 50$, wróć do P10. W przeciwnym razie zakończ algorytm. ■

Program P (*Drukowanie tablicy 500 liczb pierwszych*). Ten program jest napisany może nieco sztucznie, ale za to demonstruje wiele cech języka MIXAL. $rI1 \equiv J - 500$; $rI2 \equiv N$; $rI3 \equiv K$; $rI4$ odpowiada B ; $rI5$ zawiera M plus wielokrotności 50.

```

01 * PRZYKŁADOWY PROGRAM ... TABLICA LICZB PIERWSZYCH
02 *
03 L      EQU  500          Ilu liczb szukamy.
04 PRINTE EQU  18          Numer jednostki odpowiadający drukarce.
05 PRIME  EQU  -1          Obszar pamięci na tablicę liczb pierwszych.
06 BUFO   EQU  2000         Pamięć na  $\text{BUFFER}[0]$ .
07 BUF1   EQU  BUFO+25    Pamięć na  $\text{BUFFER}[1]$ .
08        ORIG 3000         Przesuń do nowej strony.
09 START  IOC  0(PRINTER)

```



Rys. 14. Algorytm P.

10	LD1 =1-L=	<u>P1. Inicjowanie tablicy.</u> J ← 1.
11	LD2 =3=	N ← 3.
12 2H	INC1 1	<u>P2. N jest liczbą pierwszą.</u> J ← J + 1.
13	ST2 PRIME+L, 1	PRIME[J] ← N.
14	J1Z 2F	<u>P3. Czy już 500?</u>
15 4H	INC2 2	<u>P4. Zwiększanie N.</u>
16	ENT3 2	<u>P5. K ← 2.</u>
17 6H	ENTA 0	<u>P6. Czy PRIME[K] \ N?</u>
18	ENTX 0,2	rAX ← N.
19	DIV PRIME, 3	rA ← Q, rX ← R.
20	JXZ 4B	Do P4, jeśli R = 0.
21	CMPA PRIME, 3	<u>P7. Czy PRIME[K] duże?</u>
22	INC3 1	<u>P8. Zwiększanie K.</u>
23	JG 6B	Do P6, jeśli Q > PRIME[K]; w przeciwnym
24	JMP 2B	razie N jest liczbą pierwszą.
25 2H	OUT TITLE(PRINTER)	<u>P9. Drukowanie tytułu.</u>
26	ENT4 BUF1+10	B ← 1.
27	ENT5 -50	M ← 0.
28 2H	INC5 L+1	Zwiększenie M.
29 4H	LDA PRIME, 5	<u>P10. Zestawianie wiersza.</u> (Od prawej do lewej)
30	CHAR	Zamień PRIME[M] na postać dziesiętną.
31	STX 0,4(1:4)	
32	DEC4 1	
33	DEC5 50	(rI5 zmniejsza się o 50, dopóki
34	J5P 4B	nie stanie się niedodatnie)
35	OUT 0,4(PRINTER)	<u>P11. Drukowanie wiersza.</u>
36	LD4 24,4	Przełączanie buforów.
37	J5N 2B	Jeśli rI5 = 0, to gotowe.
38	HLT	
39	* POCZĄTKOWA ZAWARTOŚĆ TABLIC I BUFORÓW	
40	ORIG PRIME+1	
41	CON 2	Pierwszą liczbą pierwszą (!) jest 2.
42	ORIG BUFO-5	

```

43 TITLE ALF FIRST      Dane alfanumeryczne –
44          ALF FIVE       wiersz tytułowy.
45          ALF HUND
46          ALF RED P
47          ALF RIMES
48          ORIG BUF0+24
49          CON  BUF1+10   Bufory są wzajemnie powiązane.
50          ORIG BUF1+24
51          CON  BUF0+10
52          END  START    Koniec programu. ■

```

Zatrzymajmy się przy następujących obserwacjach:

1. Wiersze 01, 02 i 39 rozpoczynają się od gwiazdki. W ten sposób są oznaczane wiersze komentarza, który jest wyłącznie opisem i nie ma żadnego wpływu na tłumaczenie programu.

2. Tak jak w programie M, pseudooperacja **EQU** w wierszu 03 jest deklaracją znaczenia symbolu; w tym przypadku symbol **L** będzie równoważny liczbie 500. (W wierszach 10 – 24 **L** oznacza liczbę poszukiwanych liczb pierwszych). Zauważmy, że w wierszu 05 symbolowi **PRIME** nadajemy wartość ujemną. Symbolom można nadawać wartości 5-bajtowych liczb ze znakiem. W wierszu 07 wartość symbolu **BUF1** wyznaczamy jako **BUF0+25**, tj. 2025. **MIXAL** pozwala wykonywać proste operacje arytmetyczne na liczbach. Kolejny przykład pojawia się w wierszu 13, gdzie wartość **PRIME+L** (w tym przypadku 499) jest obliczana przez assembler.

3. Symbol **PRINTER** w wierszach 25 i 35 stoi w polu **F**. Pole **F**, zapisywane zawsze w nawiasach, może być opisane liczbowo albo symbolicznie, tak jak pozostałe pola kolumny **ADRES**. W wierszu 31 za pomocą dwukropka opisany jest wycinek „(1:4)”.

4. Język **MIXAL** umożliwia zapisanie słowa nie będącego rozkazem na kilka sposobów. W wierszu 41 korzystamy z pseudooperacji **CON** do wygenerowania stałej „2”; wynikiem tłumaczenia wiersza 41 jest słowo

+		2	.
---	--	---	---

W wierszu 49 używamy bardziej skomplikowanej stałej „**BUF1+10**”, na podstawie której wygenerowane zostanie słowo

+		2035	.
---	--	------	---

Stałą ujętą w dwa znaki równości nazywamy *stałą pamięciową* (zobacz wiersze 10 i 11). Dla stałych pamięciowych assembler automatycznie wygeneruje wewnętrzne nazwy i wstawi wiersze „**CON**”. Na przykład wiersze 10 i 11 programu P są tłumaczone tak, jakby miały postać

10	LD1 con1
11	LD2 con2

a na końcu programu, między wiersze 51 i 52, zostaną wstawione wiersze

```
51a con1 CON 1-L
51b con2 CON 3
```

Wiersz 51a zostanie przetłumaczony na słowo

-	499	.
---	-----	---

Korzystanie ze stałych pamięciowych to czysta wygoda, ponieważ programista nie musi ani wymyślać nazw symbolicznych dla prostych stałych, ani troszczyć się o wstawienie stałych na końcu programu. Programista może skupić się na sednie problemu, nie martwiąc się o przyziemne drobiazgi. (Przykłady stałych pamięciowych w programie P są nieco niefortunne, ponieważ otrzymaliśmy trochę lepszy program, zastępując wiersze 10 i 11 bardziej efektywnymi rozkazami „ENT1 1-L” i „ENT2 3”).

5. Dobry język asemblerowy powinien odzwierciedlać *sposób myślenia* programisty o programie maszynowym. Jednym z przykładów tego podejścia jest wykorzystanie stałych pamięciowych, o czym pisaliśmy powyżej. Inny przykład to wykorzystanie symbolu „*”, którego znaczenie wyjaśniliśmy przy okazji omawiania programu M. Trzeci przykład to mechanizm *symboli lokalnych*, takich jak symbol 2H pojawiający się w polu lokacji w wierszach 12, 25 i 28.

Symboli lokalne są szczególnymi symbolami, których wartości mogą wiele razy zostać *przedefiniowane*. Symbol globalny, jak na przykład PRIME, ma ustalone znaczenie w całym programie i jeśli pojawiłby się w polu lokacji więcej niż jednego wiersza, asembler zgłosiłby błąd. Ale symbole lokalne mają inne własności. Piszymy na przykład 2H („2 here”, czyli „2 tutaj”) w polu lokacji i 2F („2 forward”, czyli „2 dalej”) lub 2B („2 backward”, czyli „2 wcześniej”) w polu adresu wiersza programu w języku MIXAL:

2B oznacza najbliższą lokację 2H, *patrząc wstecz*;
 2F oznacza najbliższą lokację 2H, *patrząc w przód*.

Stąd „2F” w wierszu 14 oznacza odwołanie do wiersza 25; „2B” w wierszu 24 oznacza odwołanie wstecz do wiersza 12; „2B” w wierszu 37 oznacza odwołanie do wiersza 28. Adres 2F lub 2B nigdy nie oznacza odwołania do wiersza, w którym występuje. Na przykład trzy wiersze

```
2H EQU 10
2H MOVE 2F(2B)
2H EQU 2B-3
```

są równoważne jednemu wierszowi

```
MOVE *-3(10).
```

Symboli 2F i 2B nie należy używać w polu lokacji. Symbolu 2H nie należy używać w polu adresu. Można posługiwać się dziesięcioma symbolami lokalnymi, powstającymi poprzez wymianę „2” w powyższych przykładach na dowolną cyfrę dziesiętną od 0 do 9.

Mechanizm symboli lokalnych został wprowadzony przez M. E. Conwaya w 1958 roku w związku z programem asemblerowym dla komputera UNIVAC I. Symbole lokalne zwalniają programistę z obowiązku dobierania nazw symbolicznych dla każdego adresu w sytuacjach, gdy wszystkie odwołania do tego adresu znajdują się w programie w zasięgu kilku wierszy. Zazwyczaj trudno wymyślić odpowiednie nazwy na te najbliższe lokacje, zatem programista zaczyna tworzyć enigmatyczne symbole typu X1, X2, X3 itp., narażając się na ryzyko powtórnego użycia któregoś z nich. Z tego powodu symbole lokalne w języku asemblerowym są pozyteczne i naturalne.

6. Adres w wierszach 30 i 38 jest pusty. Oznacza to, że w wyniku tłumaczenia w pole adresu zostanie wpisane zero. Moglibyśmy pozostawić pusty adres także w wierszu 17, ale program byłby wtedy mniej czytelny.

7. W wierszach 43–47 korzystamy z pseudoinstrukcji „ALF”, powodującej utworzenie 5-bajtowej stałej w kodzie alfanumerycznym komputera MIX. Na przykład wiersz 45 powoduje wygenerowanie słowa

+	00	08	24	15	04	,
---	----	----	----	----	----	---

co odpowiada napisowi „LHUND”, który jest fragmentem wiersza tytułowego drukowanego przez program P.

Wartości lokacji, które nie są określone w programie, zostaną ustawione na zero (poza lokacjami wykorzystywanymi przez program ładowający, zazwyczaj 3700–3999). Z tego powodu nie trzeba dbać o wstawienie spacji w słowa składające się na napis tytułowy (tj. za wierszem 47).

8. Argumentem pseudooperacji ORIG może być wyrażenie arytmetyczne, zobacz wiersze 40, 42 i 48.

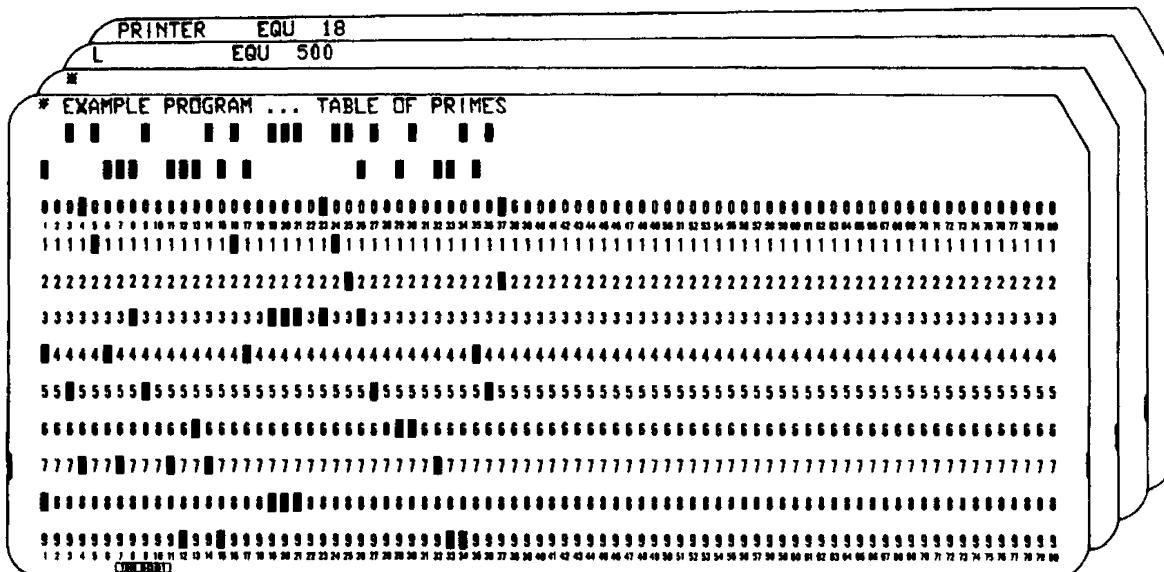
9. Ostatni wiersz programu w języku MIXAL w polu OP zawsze zawiera „END”. Przyjmuje się, że adres podany w tym wierszu to lokacja, od której ma rozpocząć się wykonanie programu po jego załadowaniu do pamięci.

10. Na koniec zauważmy, że program P został tak napisany, że wartości rejestrów indeksowych biegą w kierunku zera i są porównywane z zerem. Na przykład w rI1 trzymamy wartość J-500, a nie J. Warto przyjrzeć się także sprytnie napisanym wierszom 26–34.

Czytelnika może zaciekać odrobina danych statystycznych dotyczących programu P: instrukcja dzielenia w wierszu 19 wykonuje się 9538 razy; czas potrzebny na wykonanie wierszy 10–24 wynosi 182144u.

Programy w języku MIXAL można wydziurkować na kartach lub wpisać za pomocą terminala, jak pokazano na rysunku 15. Dla kart perforowanych używamy następującego formatu:

- | | |
|----------------|-------------------------------------|
| Kolumny 1–10 | pole LOC (lokacja); |
| Kolumny 12–15 | pole OP; |
| Kolumny 17–80 | pole ADRES i ewentualnie komentarz; |
| Kolumny 11, 16 | puste. |



```
* EXAMPLE PROGRAM ... TABLE OF PRIMES
*
L EQU 500
PRINTER EQU 18
PRIME EQU -1
BUFO EQU 2000
BUF1 EQU BUFO+25
ORIG 3000
START IOC 0(PRINTER)
LD1 =1-L=
```

Rys. 15. Pierwsze wiersze programu P (z nazwą w wersji oryginalnej) wydziurkowane na kartach lub wpisane za pomocą terminala.

Jeżeli kolumna 1 zawiera gwiazdkę, to całą kartę traktuje się jak komentarz. Pole ADRES kończy się przed pierwszą pustą kolumną, za kolumną 16. Na prawo od tej pierwszej pustej kolumny można wydziurkować dowolny tekst komentarza, który nie będzie brany pod uwagę przy tłumaczeniu programu. (*Wyjątek*: Jeśli w polu OP stoi ALF, komentarz zawsze zaczyna się w kolumnie 22).

Jeżeli dane są wczytywane z terminala, to używamy mniej sztywnego formatu: pole LOC kończy się przed pierwszą spacją, a pola OP oraz ADRES (jeśli jest) zaczynają się od pierwszego znaku nie będącego spacją i kończą przed następną spacją. Jednak jeśli w polu OP stoi ALF, to po polu OP muszą nastąpić dokładnie dwie spacje i pięć znaków alfanumerycznych lub dokładnie jedna spacja i pięć znaków alfanumerycznych, z których pierwszy nie jest spacją. Reszta wiersza stanowi komentarz.

Asembler komputera **MIX** czyta pliki wejściowe w powyższym formacie i tłumaczy je na program w języku maszynowym, który później można załadować do pamięci. Czytelnikowi być może uda się uzyskać dostęp do asemblera na architekturę **MIX** oraz symulatora komputera **MIX**, co pozwoli dokładnie przerobić zamieszczone w książce ćwiczenia.

Wiemy mniej więcej, jak pisać programy w języku MIXAL. Niniejszy rozdział zakończymy, opisując to bardziej szczegółowo. W szczególności powiemy, jak w języku MIXAL pisać *nie można*. Poniższy (raczej skromny) zestaw reguł definiuje język MIXAL:

1. *Symbol* jest napisem złożonym z jednej do dziesięciu liter i/lub cyfr, zawierającym przynajmniej jedną literę. *Przykłady*: PRIME, TEMP, 20BY20. Zakładamy, że symbole specjalne dH , dF i dB , gdzie d jest pojedynczą cyfrą, są na użytku tego opisu zastąpione innymi unikalnymi symbolami, zgodnie z opisaną wcześniej konwencją „symboli lokalnych”.

2. *Liczba* jest napisem składającym się z jednej do dziesięciu cyfr. *Przykład*: 00052.

3. O każdym wystąpieniu symbolu w programie mówimy, że jest albo „symbolem zdefiniowanym” albo „odwołaniem w przód”. *Symbol zdefiniowany* to symbol, który wystąpił w polu LOC w którymś z poprzednich wierszy programu. *Odwołanie w przód* to symbol, który nie został do tej pory w ten sposób zdefiniowany.

4. *Wyrażenie atomowe* może być

- a) liczbą,
- b) symbolem zdefiniowanym (oznaczającym swój liczbowy równoważnik, zobacz reguła 13),
- c) gwiazdką (oznaczającą wartość \star ; zobacz reguły 10 i 11).

5. *Wyrażenie* może być

- a) wyrażeniem atomowym,
- b) znakiem plus lub minus, za którym stoi wyrażenie atomowe,
- c) wyrażeniem atomowym, za którym stoi operator dwuargumentowy, za którym stoi wyrażenie atomowe.

Dopuszcza się sześć operatorów dwuargumentowych $+$, $-$, $*$, $/$, $//$ i $:$. Operatory definiuje się na słowach maszyny MIX w sposób następujący:

$C = A+B$	LDA AA; ADD BB; STA CC.
$C = A-B$	LDA AA; SUB BB; STA CC.
$C = A*B$	LDA AA; MUL BB; STX CC.
$C = A/B$	LDA AA; SRAX 5; DIV BB; STA CC.
$C = A//B$	LDA AA; ENTX 0; DIV BB; STA CC.
$C = A:B$	LDA AA; MUL =8=; SLAX 5; ADD BB; STA CC.

W powyższym zestawieniu AA, BB i CC oznaczają lokacje zawierające odpowiednio wartości symboli A, B i C. Operacje składające się na wyrażenie są wykonywane od lewej do prawej.

Przykłady:

- 1+5 równa się 4.
- 1+5*20/6 równa się $4*20/6$, równa się $80/6$, równa się 13 (od lewej do prawej).
- 1//3 równa się słowu maszyny MIX, którego wartość wynosi w przybliżeniu $b^5/3$, gdzie b jest rozmiarem bajtu; oznacza to, że słowo odpowiada ułamkowi $\frac{1}{3}$ przy założeniu, że po lewej stronie stoi kropka dziesiętna.
- 1:3 równa się 11 (wykorzystywane zazwyczaj w specyfikacjach wycków).
- *-3 równa się \circledast minus trzy.
- *** równa się \circledast razy \circledast .

6. *Część A* (wykorzystywana do opisu pola adresu rozkazu maszyny MIX) może być

- a) napisem pustym (co oznacza wartość zero),
- b) wyrażeniem,
- c) odwołaniem w przód (co może oznaczać równoważnik symbolu, zobacz reguła 13),
- d) stałą znakową (co oznacza odwołanie do symbolu wygenerowanego przez asembler, zobacz reguła 12).

7. *Część indeksowa* (określająca pole indeksu rozkazu) może być

- a) napisem pustym (co oznacza wartość zero),
- b) przecinkiem, po którym następuje wyrażenie (co oznacza wartość tego wyrażenia).

8. *Część F*, określająca pole F instrukcji, może być

- a) napisem pustym (co oznacza standardowy wycinek dla danej instrukcji, porównaj tabelę 1.3.1–1),
- b) nawiasem otwierającym, po którym następuje wyrażenie, po którym z kolei następuje nawias zamykający (co oznacza wartość wyrażenia).

9. *W-wartość* (używana do zapisania stałej o szerokości słowa maszynowego) może być

- a) wyrażeniem, po którym następuje część F (w tym przypadku pusta część F oznacza (0:5)),
- b) W-wartością, po której następuje przecinek, po którym następuje W-wartość w postaci (a).

W-wartość oznacza wartość liczbową o szerokości słowa maszynowego i jest interpretowana w sposób następujący. Niech W-wartość ma postać „E₁(F₁)”,

$E_2(F_2), \dots, E_n(F_n)$ ", gdzie $n \geq 1$, E są wyrażeniami, a F to pola. Taka W-wartość opisuje słowo, które pojawiłoby się w lokacji WVAL po wykonaniu następującego hipotetycznego programu:

STZ WVAL; LDA C₁; STA WVAL(F₁); ...; LDA C_n; STA WVAL(F_n).

Tutaj C₁, ..., C_n oznaczają lokacje zawierające wartości wyrażeń E₁, ..., E_n. Każda specyfikacja F_i musi być postaci 8L_i+R_i, gdzie 0 ≤ L_i ≤ R_i ≤ 5. Przykłady:

1	opisuje słowo	+ 1 - 1000 1 + 1
1, -1000(0:2)	opisuje słowo	
-1000(0:2), 1	opisuje słowo	

10. W procesie tłumaczenia kodu w języku MIXAL na kod maszynowy wykorzystywana jest wartość oznaczana przez * (tzw. licznik lokacji), która początkowo wynosi zero. Wartość * zawsze jest liczbą nieujemną mieszczącą się w dwóch bajtach. Jeśli pole lokacji jakiegoś wiersza nie jest puste, to musi zawierać symbol, który nie został do tej pory zdefiniowany. Równoważnikiem takiego symbolu staje się aktualna wartość *.

11. Po przetworzeniu zawartości pola LOC, jak opisano w regule 10, dalszy przebieg tłumaczenia zależy od zawartości pola OP. Jest sześć możliwości:

- W polu OP stoi nazwa symboliczna rozkazu maszyny MIX (zobacz tabela 1 pod koniec poprzedniego podrozdziału). Wartości C oraz standardowa wartość F dla rozkazów są zdefiniowane w tabeli. W takim przypadku w polu ADRES powinna znajdować się część A (reguła 6), a potem część indeksowa (reguła 7), po której następuje część F (reguła 8). To wyznacza wartości C, F, A oraz I. Wynikiem tłumaczenia jest wygenerowanie słowa za pomocą ciągu rozkazów „LDA C; STA WORD; LDA F; STA WORD(4:4); LDA I; STA WORD(3:3); LDA A; STA WORD(0:2)” w lokacji wyznaczonej przez * oraz zwiększenie * o 1.
- W polu OP znajduje się „EQU”. W polu ADRES powinna być W-wartość (zobacz reguła 9). Jeżeli pole LOC jest niepuste, równoważnikiem znajdującego się w nim symbolu staje się wartość podana w polu ADRES. Ta reguła przeszła regułę 10. Wartość * pozostaje niezmieniona. (W ramach niebanalnego przykładu rozważmy wiersz

BYTESIZE EQU 1(4:4)

pozwalający programiście dysponować symbolem, którego wartość zależy od rozmiaru bajtu. Jest to sytuacja dopuszczalna pod warunkiem, że zawierający taką definicję program ma sens dla każdego dozwolonego rozmiaru bajtu).

- W polu OP znajduje się „ORIG”. W polu ADRES powinna być W-wartość (zobacz reguła 9) – licznik lokacji (*) jest ustawiany na tę właśnie wartość. (Zauważmy, że ze względu na regułę 10 równoważnik symbolu będący w polu

LOC wiersza zawierającego ORIG (tj. aktualna wartość *) jest wyznaczany, zanim * się zmieni. Na przykład dla

TABLE ORIG *+100

równoważnikiem TABLE będzie *pierwsza ze stu lokacji*.

- d) W polu OP jest „CON”. W polu ADRES powinna znajdować się W-wartość. Wynikiem tłumaczenia jest wygenerowanie słowa o tej właśnie wartości w lokacji wyznaczonej przez * oraz zwiększenie * o 1.
- e) W polu OP jest „ALF”. W wyniku tłumaczenia jest generowane słowo utworzone z kodów znakowych pierwszych pięciu znaków będących w polu adresu (reszta operacji jak dla CON).
- f) W polu OP znajduje się „END”. W polu ADRES powinna być W-wartość, której wycinek (4:5) określa lokację, od której ma się rozpocząć wykonanie programu. Wiersz END oznacza koniec programu w języku MIXAL. Asembler bezpośrednio przed wierszem zawierającym END niejawnie wstawia dodatkowe wiersze, odpowiadające wszystkim niezdefiniowanym symbolom i stałym pamięciowym (zobacz reguły 12 i 13). Kolejność tych wierszy nie jest zdefiniowana. Zgodnie z powyższym symbol w polu LOC wiersza zawierającego END będzie oznaczał pierwszą lokację po wstawionych słowach.

12. Stała pamięciowa: W-wartość krótsza niż 10 znaków może zostać ujęta w znaki „=” i wykorzystana jako odwołanie w przód. Skutkiem natrafienia na taki napis jest niejawnie utworzenie nowego symbolu i wstawienie na koniec programu wiersza zawierającego pseudooperację CON definiującą ten symbol (zobacz uwaga 4 po programie P).

13. Każdy symbol ma dokładnie jedną wartość równoważną, będącą pełnym słowem maszynowym komputera MIX, wyznaczoną przez położenie wiersza, w którym symbol wystąpił w polu LOC, zgodnie z regułami 10 i 11(b). Jeżeli symbol nie występuje nigdzie w polu LOC, to przed wierszem zawierającym END niejawnie wstawia się nowy wiersz, dla którego OP = „CON”, ADRES = „0” oraz LOC = brakujący symbol.

Uwaga: Najpoważniejszą konsekwencją tej reguły jest ograniczenie dotyczące odwołań w przód. Symbol, który nie został do tej pory zdefiniowany w polu LOC, może zostać wykorzystany wyłącznie w polu adresu rozkazu. W szczególności nie można go używać (a) jako argumentu operacji arytmetycznych ani (b) w polu ADRES pseudorozkazu EQU, ORIG i CON. Zarówno LDA 2F+1, jak i CON 3F są niedozwolone. Takie ograniczenie wprowadza się w celu usprawnienia tłumaczenia programu. Jak wynika z doświadczenia nabytego podczas pisania tej książki w praktyce nie jest to istotne ograniczenie.

Na koniec należy wspomnieć, że dla komputera MIX istnieją dwa języki niskiego poziomu: **MIXAL***, język zorientowany maszynowo, dający się łatwo tłumaczyć przez prosty jednoprzebiegowy asembler, oraz **PL/MIX**, język nieco lepiej

* Autor był bardzo zdziwiony, gdy w 1971 roku dowiedział się, że MIXAL to zarazem nazwa jugosławiańskiego proszku do prania, przeznaczonego do użycia w *automate*.

odzwierciedlający struktury danych oraz struktury sterowania, przypominający komentarze zapisywane przez nas w programach w języku MIXAL. Język PL/MIX zostanie opisany w rozdziale 10.

ĆWICZENIA – zestaw pierwszy

1. [00] W tekście jest powiedziane, że „X EQU 1000” nie powoduje wygenerowania rozkazu przypisującego wartość. Przypuśćmy, że piszesz program na komputer MIX i chcesz wstawić wartość 1000 do pewnej komórki pamięci (o nazwie symbolicznej X). Jak to zapisać w języku MIXAL?

- ▶ 2. [10] W wierszu 12 programu M jest instrukcja „JMP *”, gdzie * oznacza lokację przypisaną temu wierszowi. Czemu ten program nie wchodzi w nieskończoną pętlę, polegającą na ustawicznym wykonywaniu tej instrukcji?
- ▶ 3. [23] Jaki jest wynik wykonania poniższego programu, jeżeli połączymy go z programem M?

```

START IN  X+1(0)
        JBUS *(0)
        ENT1 100
1H     JMP  MAXIMUM
        LDX  X,1
        STA  X,1
        STX  X,2
        DEC1 1
        J1P  1B
        OUT  X+1(1)
        HLT
END  START  ■

```

- ▶ 4. [25] Przetłumacz ręcznie program P (to naprawdę potrwa krócej, niż Ci się wydaje). Jak wygląda zawartość pamięci odpowiadająca przetłumaczonemu programowi?
- 5. [11] Czemu program P nie korzysta z rozkazu JBUS do badania gotowości drukarki?
- 6. [HM20] (a) Pokaż, że jeżeli n nie jest liczbą pierwszą, to n ma dzielnik d , taki że $1 < d \leq \sqrt{n}$. (b) Skorzystaj z tego, by pokazać, że na podstawie testu w kroku P7 algorytmu P można wnioskować, że N jest liczbą pierwszą.
- 7. [10] (a) Jakie jest znaczenie „4B” w wierszu 34 programu P? (b) Czy i jaki byłby skutek zmiany lokacji w wierszu 13 na „2H”, a adresu w wierszu 20 na „2B”?
- ▶ 8. [24] Co robi poniższy program? (Nie uruchamiaj go na komputerze!) .

```

* PROGRAM TAJEMNY
BUF ORIG *+3000
1H ENT1 1
        ENT2 0
        LDX  4F
2H ENT3 0,1
3H STZ  BUF,2
        INC2 1
        DEC3 1
        J3P  3B

```

```

STX  BUF,2
INC2 1
INC1 1
CMP1 =75=
JL 2B
ENN2 2400
OUT  BUF+2400,2(18)
INC2 24
J2N *-2
HLT
4H ALF AAAAA
END 1B

```

ĆWICZENIA – zestaw drugi

Poniższe ćwiczenia to krótkie wprawki programistyczne, obejmujące typowe problemy praktyczne oraz metody programistyczne. Zachęcamy Czytelników, by w celu nabrania doświadczenia w posługiwaniu się komputerem **MIX** i pogłębienia umiejętności pisania programów przerobili kilka wybranych ćwiczeń. Ćwiczenia nie są niezbędne do przyswojenia dalszego materiału – można rozwiązywać je równocześnie z dalszą lekturą rozdziału 1.

Oto metody programistyczne ujęte w poniższym zestawie:

Wykorzystanie tablic przełączających do obsługi rozgałęzień wielowyjściowych: ćwiczenia 9, 13 i 23.

Wykorzystanie rejestrów indeksowych do realizacji tablic dwuwymiarowych: ćwiczenia 10, 21 i 23.

Rozpakowywanie znaków: ćwiczenia 13 i 23.

Arytmetyka całkowitoliczbowa i stałopozycyjna: ćwiczenia 14, 16 i 18.

Wykorzystanie podprogramów: ćwiczenia 14 i 20.

Buforowanie wejścia: ćwiczenie 13.

Buforowanie wyjścia: ćwiczenia 21 i 23.

Przetwarzanie list: ćwiczenie 22.

Sterowanie w czasie rzeczywistym: ćwiczenie 20.

Generowanie grafiki: ćwiczenie 23.

Tam, gdzie w treści ćwiczenia jest powiedziane „napisz (pod)program na komputer **MIX**”, wystarczy napisać kod w języku **MIXAL**. Ten kod będzie stanowił fragment (hipotetycznego) programu. W takim fragmencie można nie dbać o wprowadzanie i wyprowadzanie danych. Wystarczy podać zawartości pól LOC, OP i ADRES, wraz z komentarzami. Liczbowy kod maszynowy, numery wierszy oraz kolumna „czas” (zobacz program M) nie są wymagane, jeżeli nie jest to powiedziane wprost. Nie piszemy także wiersza END.

Z drugiej strony, jeżeli w ćwiczeniu powiedziane jest „napisz *kompletny* program na komputer **MIX**”, to znaczy, że należy napisać pełen wykonywalny program w języku **MIXAL**, w szczególności zawierający wiersz END. Dostępne są asemblerы i symulatory komputera **MIX**, na których takie programy można testować.

- 9. [25] Lokacja INST zawiera słowo maszynowe komputera **MIX**, które być może jest rozkazem komputera **MIX**. Napisz program na komputer **MIX**, który wykonuje skok do lokacji GOOD, jeżeli słowo ma poprawne pole C, pole ±AA, pole I oraz pole F, zgodnie z tabelą 1.3.1–1. W przeciwnym przypadku Twój program powinien wykonywać skok do

lokacji **BAD**. Pamiętaj, że poprawność pola F zależy od zawartości pola C; na przykład, jeśli $C = 7$ (**MOVE**), to każda zawartość pola F jest poprawna, ale gdy $C = 8$ (**LDA**), pole F musi być postaci $8L + R$, gdzie $0 \leq L \leq R \leq 5$. Zawartość pola „ $\pm AA$ ” jest niepoprawna, jeżeli C jest kodem rozkazu wymagającego adresu pamięci, $I = 0$ i $\pm AA$ nie jest poprawnym adresem.

Uwaga: Niedoświadczeni programiści próbują rozwiązywać podobne problemy, pisząc długie ciągi testów dotyczących zawartości pola C: „**LDA C; JAZ 1F; DECA 5; JAN 2F; JAZ 3F; DECA 2; JAN 4F; ...**”. To *nie* jest dobra praktyka! Najlepszym sposobem wykonywania takich wielokrotnych rozgałęzień jest przygotowanie pomocniczej *tablicy* wyznaczającej schemat postępowania. Jeżeli mielibyśmy na przykład tablicę 64-elementową, to moglibyśmy napisać „**LD1 C; LD1 TABLE, 1; JMP 0, 1**”, czyli przyspieszyć całą decyzję przez sprowadzenie kodu do jednego skoku. W takiej tablicy można przechowywać także inne przydatne informacje. Podejście tablicowe do przedstawionego problemu powoduje nieznaczne wydłużenie programu (z uwzględnieniem miejsca na tablicę) oraz zasadniczo powoduje zwiększenie szybkości działania i ułatwia modyfikowanie.

► 10. [31] Założymy, że macierz 9×8

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{18} \\ a_{21} & a_{22} & a_{23} & \dots & a_{28} \\ \vdots & & & & \vdots \\ a_{91} & a_{92} & a_{93} & \dots & a_{98} \end{pmatrix}$$

jest tak zapisana w pamięci, że element a_{ij} znajduje się w lokacji $1000 + 8i + j$. Macierz w pamięci wygląda zatem następująco:

$$\begin{pmatrix} (1009) & (1010) & (1011) & \dots & (1016) \\ (1017) & (1018) & (1019) & \dots & (1024) \\ \vdots & & & & \vdots \\ (1073) & (1074) & (1075) & \dots & (1080) \end{pmatrix}.$$

Mówimy, że macierz ma „punkt siodłowy”, jeżeli jakiś element jest najmniejszy w swoim wierszu i największy w swojej kolumnie. Formalnie, a_{ij} jest punktem siodłowym, jeśli

$$a_{ij} = \min_{1 \leq k \leq 8} a_{ik} = \max_{1 \leq k \leq 9} a_{kj}.$$

Napisz program na komputer **MIX**, który oblicza położenie punktu siodłowego (jeśli istnieje) i zatrzymuje się z tą wartością w rI1. Jeżeli punkt siodłowy nie istnieje, program powinien do rI1 wpisywać zero.

11. [M29] Jakie jest prawdopodobieństwo zdarzenia, że macierz z poprzedniego ćwiczenia ma punkt siodłowy przy założeniu, że 72 elementy są parami różne i każdy z 2^{72} układów jest jednakowo prawdopodobny? Jakie jest to prawdopodobieństwo, jeżeli zamiast tego założymy, że elementami macierzy są zera i jedynki, a każdy z 2^{72} układów jest jednakowo prawdopodobny?

12. [HM42] Do ćwiczenia 10 podane są dwa rozwiązania (zobacz s. 534), a zasugerowane jest trzecie. Nie jest jasne, które z tych dwóch jest lepsze. Przeanalizuj algorytmy dla obu założeń z ćwiczenia 11 i oceń, która metoda jest lepsza.

13. [28] Analityk szyfrów chce policzyć częstość występowania liter w pewnym szyfrze. Zaszyfrowany tekst jest wydzielony na taśmie papierowej, a jego koniec oznacono gwiazdką. Napisz kompletny program na komputer **MIX**, któryczyta dane z taśmy, zlicza częstości występowania znaków (do pierwszej gwiazdki) i wypisuje wyniki w postaci

A	0010257
B	0000179
D	0794301

itd. (jeden znak w wierszu). Liczba spacji nas nie interesuje, nie należy także wypisywać znaków, które nie wystąpiły w szyfrogramie (jak C w powyższym przykładzie). Mając na celu wydajność, zastosuj buforowanie: podczas wczytywania bloku do jednego obszaru pamięci, można zliczać znaki w drugim obszarze. Możesz założyć, że za gwiazdką na taśmie znajduje się jeszcze jeden blok.

- **14.** [31] Poniższy algorytm autorstwa neapolitańskiego astronoma Aloysiusa Liliusa i niemieckiego matematyka, jezuita, Christopera Claviusa pochodzi z końca XVI wieku i wykorzystywany jest w większości kościołów zachodnich do wyznaczania daty Wielkanocy dla lat po 1582 roku.

Algorytm E (Data Wielkanocy). Oznaczmy przez Y rok poszukiwanej daty Wielkanocy.

- E1.** [Złota liczba] Przyjmij $G \leftarrow (Y \bmod 19) + 1$. (G to tzw. „złota liczba” roku w 19-letnim cyklu metonicznym).
- E2.** [Stulecie] Przyjmij $C \leftarrow \lfloor Y/100 \rfloor + 1$. (Gdy Y nie jest wielokrotnością 100, wtedy C oznacza numer stulecia; na przykład 1984 to dwudzieste stulecie).
- E3.** [Poprawki] Przyjmij $X \leftarrow \lfloor 3C/4 \rfloor - 12$, $Z \leftarrow \lfloor (8C + 5)/25 \rfloor - 5$. (X oznacza tutaj liczbę lat, takich jak 1900, w których odrzucono rok przestępny w celu synchronizacji z rokiem słonecznym. Z to specjalna poprawka mająca na celu synchronizację Wielkanocy z fazami księżyca).
- E4.** [Znajdowanie niedzieli] Przyjmij $D \leftarrow \lfloor 5Y/4 \rfloor - X - 10$. [Określony liczbą $((-D) \bmod 7)$ dzień marca istotnie będzie niedzielą].
- E5.** [Epakta] Przyjmij $E \leftarrow (11G + 20 + Z - X) \bmod 30$. Jeśli $E = 25$, a złota liczba G jest większa od 11 lub jeśli $E = 24$, to zwiększą E o 1. (Liczba E to tzw. *epakta*, określająca datę pełni księżyca).
- E6.** [Określenie daty pełni księżyca] Przyjmij $N \leftarrow 44 - E$. Jeśli $N < 21$, to przypisz $N \leftarrow N + 30$. (Wielkanoc miała być pierwszą niedzielą po pierwszej pełni księżyca wypadającą po 21 marca. Zmiany orbity księżyca trochę tę regułę psują. Nas interesuje jednak bardziej „księżyk kalendarzowy” niż astronomiczny. W N -ty dzień marca przypada kalendarzowa pełnia).
- E7.** [Przesunięcie do niedzieli] $N \leftarrow N + 7 - ((D + N) \bmod 7)$.
- E8.** [Określenie miesiąca] Jeśli $N > 31$, to poszukiwaną datą jest $(N - 31)$ kwietnia; w przeciwnym razie poszukiwaną datą jest N marca. ■

Napisz podprogram, który obliczy i wypisze datę Wielkanocy w podanym roku przy założeniu, że numer roku jest mniejszy od 100000. Dane wyjściowe powinny mieć postać „*dd mmmmm, rrrr*”, gdzie *dd* to dzień, *mmmm* to miesiąc, a *rrrr* to rok. Napisz kompletny program na komputer **MIX**, który korzystając z tego podprogramu, sporządzi tablicę dat Wielkanocy od roku 1950 do roku 2000.

15. [M30] Częstym błędem popełnianym przy programowaniu algorytmu z poprzedniego ćwiczenia jest przeoczenie faktu, że wartość $(11G + 20 + Z - X)$ w kroku E5 może być ujemna, stąd dodatnia reszta mod 30 może zostać obliczona niepoprawnie. (Zobacz CACM 5 (1962), 556). Na przykład dla roku 14250 mamy $G = 1$, $X = 95$, $Z = 40$. Jeśli zatem wyszłoby $E = -24$ zamiast $E = +6$, to otrzymalibyśmy nonsensowną odpowiedź „42 APRIL”. Napisz kompletny program na komputer MIX znajdujący najwcześniejszy rok, dla którego ten błąd spowodowałby mylne obliczenie daty Wielkanocy.

16. [31] Pokazaliśmy w punkcie 1.2.7, że suma $1 + \frac{1}{2} + \frac{1}{3} + \dots$ dąży do nieskończoności. Ale jeśli oblicza się ją na komputerze ze skończoną dokładnością, to suma istnieje w tym sensie, że od pewnego momentu wyrazy są tak małe, że już nic do sumy nie wnoszą, jeśli są dodawane pojedynczo. Założmy na przykład, że obliczamy sumę przy zaokrągleniu do jednego miejsca po przecinku dziesiętnym. Mamy wówczas $1 + 0.5 + 0.3 + 0.3 + 0.2 + 0.2 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 = 3.9$.

Bardziej szczegółowo: niech $r_n(x)$ będzie liczbą x zaokrągloną do n miejsc po kropce dziesiętnej; definiujemy $r_n(x) = \lfloor 10^n x + \frac{1}{2} \rfloor / 10^n$. Szukamy teraz

$$S_n = r_n(1) + r_n\left(\frac{1}{2}\right) + r_n\left(\frac{1}{3}\right) + \dots;$$

wiemy, że $S_1 = 3.9$. Napisz kompletny program na komputer MIX, który oblicza i drukuje S_n dla $n = 2, 3, 4$ i 5 .

Uwaga: Można to zrobić o wiele szybciej niż przez proste dodawanie $r_n(1/m)$ raz za razem do momentu, gdy $r_n(1/m)$ się wyzeruje. Na przykład $r_5(1/m) = 0.00001$ dla wszystkich wartości m od 66667 do 200000. Nie ma sensu 133334 razy obliczać $1/m$. Należy raczej skorzystać na przykład z takiego algorytmu:

- A. Rozpocznij z $m_h = 1$, $S = 1$.
- B. Przypisz $m_e = m_h + 1$ i oblicz $r_n(1/m_e) = r$.
- C. Znajdź m_h , największe m , dla którego $r_n(1/m) = r$.
- D. Dodaj $(m_h - m_e + 1)r$ do S i wróć do kroku B.

17. [HM30] Przy oznaczeniach z poprzedniego ćwiczenia udowodnij lub obal wzór

$$\lim_{n \rightarrow \infty} (S_{n+1} - S_n) = \ln 10.$$

18. [25] Rosnący ciąg nieskracalnych ułamków o mianownikach $\leq n$ i wartościach pomiędzy zerem a jedynką jest nazywany „ciągiem Fareya rzędu n ”. Na przykład, ciąg Fareya rzędu 7 to

$$\frac{0}{1}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{2}{7}, \frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{1}{2}, \frac{4}{7}, \frac{3}{5}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}, \frac{1}{1}.$$

Jeżeli oznaczymy ten ciąg przez $x_0/y_0, x_1/y_1, x_2/y_2, \dots$, to z ćwiczenia 19 mamy

$$\begin{aligned} x_0 &= 0, \quad y_0 = 1; \quad x_1 = 1, \quad y_1 = n; \\ x_{k+2} &= \lfloor (y_k + n)/y_{k+1} \rfloor x_{k+1} - x_k; \\ y_{k+2} &= \lceil (y_k + n)/y_{k+1} \rceil y_{k+1} - y_k. \end{aligned}$$

Napisz podprogram na komputer MIX, który oblicza ciąg Fareya rzędu n i zapisuje wartości x_k i y_k w lokacjach odpowiednio $\mathbf{X} + k$ i $\mathbf{Y} + k$. (Całkowita liczba wyrazów w tym ciągu w przybliżeniu wynosi $3n^2/\pi^2$, możesz zatem założyć, że n jest niewielkie).

19. [M30] (a) Pokaż, że liczby x_k i y_k zdefiniowane równaniami rekurencyjnymi z poprzedniego ćwiczenia spełniają warunek $x_{k+1}y_k - x_ky_{k+1} = 1$. (b) Korzystając z (a), pokaż, że ułamki x_k/y_k rzeczywiście tworzą ciąg Fareya rzędu n .

- 20. [33] Założmy, że znacznik przepelnienia komputera MIX oraz rejestr X zostały w następujący sposób podłączone do sygnalizatorów świetlnych na rogu Bulwaru Del Mar i Alei Berkeley*:

$$\left. \begin{array}{l} rX(2:2) = \text{światło na Del Mar} \\ rX(3:3) = \text{światło na Berkeley} \end{array} \right\} 0 \text{ wyłączone, } 1 \text{ zielone, } 2 \text{ żółte, } 3 \text{ czerwone;} \\ \left. \begin{array}{l} rX(4:4) = \text{światło dla pieszych na Del Mar} \\ rX(5:5) = \text{światło dla pieszych na Berkeley} \end{array} \right\} 0 \text{ wyłączone, } 1 \text{ „WALK”, } 2 \text{ „DON’T WALK”.}$$

Samochody lub piesi podróżujący Berkeley w poprzek Del Mar muszą włączyć obwód powodujący zapalenie znacznika przepelnienia maszyny MIX. Jeśli to zdarzenie nie nastąpi, Del Mar powinna mieć cały czas światło zielone.

Czasy trwania faz są następujące:

Zielone światło na Del Mar ≥ 30 s, żółte 8 s;

Zielone światło na Berkeley 20 s, żółte 5 s.

Gdy światło dla jednej z ulic jest zielone lub żółte, światło na drugiej powinno być czerwone. Gdy świeci się zielone światło dla samochodów, odpowiadające mu zielone światło dla pieszych też powinno być zapalone, z tym że na dwanaście sekund przed zmianą światła z zielonego na żółte, światło dla pieszych zmienia się na DON’T WALK według schematu:

$$\left. \begin{array}{ll} \text{DON’T WALK} & \frac{1}{2} \text{ s} \\ \text{zgaszone} & \frac{1}{2} \text{ s} \end{array} \right\} 8 \text{ razy;}$$

DON’T WALK 4 s (i cały czas podczas światła żółtego i czerwonego).

Jeżeli układ zapalający znacznik przepelnienia zostaje włączony przy zielonym świetle dla Berkeley, samochód lub pieszy przejeżdża lub przechodzi skrzyżowanie w aktualnym cyklu, ale jeśli obwód zostaje włączony przy świetle żółtym lub czerwonym, potrzeba dodatkowego cyklu, po przepuszczeniu ruchu na Del Mar.

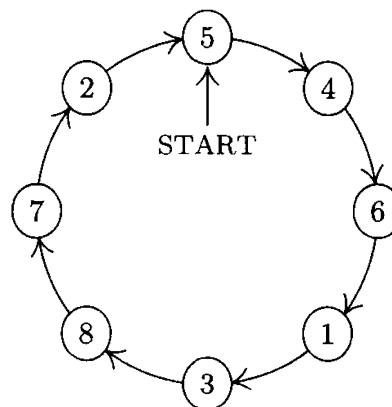
Założ, że jednostka czasu maszyny MIX wynosi $10 \mu\text{s}$. Napisz kompletny program na komputer MIX służący do sterowania światłami za pomocą zmian wartości rX, na podstawie sygnałów otrzymywanych za pośrednictwem znacznika przepelnienia. Podane czasy powinny być przestrzegane dokładnie, chyba że jest to niewykonalne. *Uwaga:* Ustawienie nowej wartości w rX zachodzi dokładnie w chwili zakończenia wykonywania rozkazu LDX lub INCX.

21. [28] *Kwadratem magicznym* rzędu n nazywamy układ liczb od 1 do n^2 w kwadratowej tabeli o tej własności, że suma liczb w każdym wierszu, kolumnie i na obu przekątnych wynosi $n(n^2 + 1)/2$. Na rysunku 16 jest pokazany kwadrat magiczny rzędu 7. Nietrudno zbudować taki kwadrat: rozpoczęynamy od jedynki znajdującej się bezpośrednio pod środkowym polem i posuwamy się po skosie w prawo w dół. Gdy dojdziemy do krawędzi, wyobrażamy sobie, że cała płaszczyzna jest pokryta kopiami kwadratu magicznego i kontynuujemy. Ta metoda działa dla wszystkich n nieparzystych.

Zakładając, że pamięć jest zorganizowana tak jak w ćwiczeniu 10, napisz kompletny program na komputer MIX, który korzystając z powyższej metody, wygeneruje

* Czytelnik zechce wziąć pod uwagę fakt, że w USA sygnalizacja świetlna funkcjonuje inaczej niż w Polsce: sygnały dla samochodów pojawiają się w sekwencji: czerwone, zielone, żółte (wobec polskiego: czerwone, czerwone i żółte, zielone, żółte); sygnały dla pieszych: biały napis WALK, migający pomarańczowy napis DON’T WALK, pomarańczowy napis DON’T WALK (wobec polskiego: zielony ludzik, migający zielony ludzik, czerwony ludzik) (przyp. tłum.).

22	47	16	41	10	35	04
05	23	48	17	42	11	29
30	06	24	49	18	36	12
13	31	07	25	43	19	37
38	14	32	01	26	44	20
21	39	08	33	02	27	45
46	15	40	09	34	03	28



Rys. 16. Kwadrat magiczny. Rys. 17. Problem Józefa Flawiusza, $n = 8$, $m = 4$.

kwadrat magiczny 23×23 i wydrukuje wynik. [Algorytm pochodzi od Ibn al-Haythama, urodzonego w Basrze około roku 965, zmarłego w Kairze około roku 1040. Wiele innych ciekawych kwadratów magicznych, będących w znacznej części znakomitymi ćwiczeniami programistycznymi, opisano w książce: W. W. Rouse Ball, *Mathematical Recreations and Essays*, revised by H. S. M. Coxeter (New York: Macmillan, 1939), rozdział 7].

22. [31] (*Problem Józefa Flawiusza*) W kręgu stoi n osób. Rozpoczynając od ustalonej pozycji, zabijamy co m -tą osobę. Krąg zacieśnia się, gdy kolejne osoby giną. Na przykład kolejność egzekucji dla $n = 8$ i $m = 4$ to 54613872, jak pokazano na rysunku 17. Pierwsza osoba jest piątą w kolejności, druga jest czwartą itd. Napisz kompletny program na komputer MIX, który drukuje kolejność egzekucji dla $n = 24$, $m = 11$. Spróbuj zaprojektować algorytm, który jest szybki dla dużych n i m (to może ocalić Ci życie). *Bibliografia:* W. Ahrens, *Mathematische Unterhaltungen und Spiele 2* (Leipzig: Teubner, 1918), rozdział 15.

23. [37] Dzięki temu ćwiczeniu Czytelnik może zdobyć odrobinę doświadczenia w generowaniu grafiki komputerowej. Naszym celem jest „narysowanie” diagramu krzyżówki.

Dane wejściowe to macierz zer i jedynek. Zero oznacza kratkę białą, jeden oznacza kratkę czarną. Program powinien wyprodukować diagram krzyżówki, z odpowiednią numeracją haseł poziomych i pionowych.

Na przykład dla macierzy

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

1			2	3	
4			5		6
7		8			
		9		10	
11	12			13	
14					

Rys. 18. Diagram odpowiadający macierzy z ćwiczenia 23.

prawidłowy diagram jest pokazany na rysunku 18. Kratka powinna mieć numer, jeżeli jest biała oraz spełniony jest jeden z warunków: (a) kratka pod nią jest biała, a bezpośrednio nad nią nie ma białej kratki, (b) kratka na prawo od niej jest biała, a bezpośrednio na lewo od niej nie ma białej kratki. Nie należy rysować czarnej kratki, jeżeli leży na

brzegu diagramu. Taka sytuacja ma miejsce na rysunku 18, gdzie czarne kratki w rogach diagramu są pominięte. Prostą metodą pozwalającą łatwo zrealizować to wymaganie jest wstawienie sztucznych wierszy i kolumn wypełnionych liczbami -1 ponad, pod oraz po bokach macierzy wejściowej, a następnie zamiana wszystkich $+1$ przylegających do -1 na -1 , aż do momentu, gdy nie ma więcej $+1$ sąsiadujących z -1 .

Otrzymany diagram powinien zostać wydrukowany zgodnie z następującymi zasadami. Każda kratka krzyżówki powinna odpowiadać pięciu kolumnom i trzem wierszom wydruku, zapełnionym następująco:

Biała kratka nienumerowana:	uuuu^+	Biała kratka z numerem nn:	nnuu^+	Czarna kratka:	+++++
	uuuu^+		uuuu^+		+++++
	+++++		+++++		+++++

Kratki „ -1 ” (zależnie od tego czy -1 są na prawo, czy poniżej) powinny wyglądać tak:

uuuu^+	uuuu^+	uuuu	uuuu	uuuu
uuuu^+	uuuu^+	uuuu	uuuu	uuuu
+++++	uuuu^+	+++++	uuuu^+	uuuu

Diagram z rysunku 18 powinien zatem być wydrukowany tak jak na rysunku 19.

W wierszu drukarki szerokości 120 znaków może zmieścić się do 23 kolumn krzyżówki. Dane wejściowe mają postać macierzy zer i jedynek o wymiarach 23×23 , każdy wiersz wydziurkowany w kolumnach 1–23 karty wejściowej. Na przykład karta odpowiadająca powyższej macierzy została wydziurkowana jako: „1000011111111111111111111”. Diagram nie musi być symetryczny i może zawierać długie ścieżki czarnych kratek, połączonych z brzegiem w sposób bardzo zawiły.

Rys. 19. Reprezentacja krzyżówki z rysunku 18 na drukarce wierszowej.

+++++	+++++	+++++	+++++	+++++
$+01$	$+$	$+02$	$+03$	$+$
$+$	$+$	$+$	$+$	$+$
$+04$	$+$	$+++++05$	$+$	$+06$
$+$	$+$	$+++++$	$+$	$+$
$+07$	$+$	$+08$	$+$	$+++++$
$+$	$+$	$+$	$+$	$+++++$
$+09$	$+$	$+10$	$+$	$+$
$+$	$+++++$	$+$	$+$	$+$
$+11$	$+12$	$+$	$+++++13$	$+$
$+$	$+$	$+$	$+++++$	$+$
$+14$	$+$	$+$	$+$	$+$
$+$	$+$	$+$	$+$	$+$
$+++++$	$+++++$	$+++++$	$+++++$	$+++++$

1.3.3. Zastosowania – permutacje

W tym punkcie zamieścimy kolejne programy na komputer MIX. Poczynimy przy okazji kilka ciekawych obserwacji dotyczących permutacji oraz programowania.

Permutacje omówiliśmy w punkcie 1.2.5. Traktowaliśmy permutację $c\,d\,f\,b\,e\,a$ jako *układ* sześciu elementów a, b, c, d, e, f . Istnieje inny punkt widzenia: o permutacji można myśleć jak o *przestawieniu* lub przemianowaniu elementów. Przy tej interpretacji używa się zazwyczaj notacji dwuwierszowej, na przykład:

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix}, \quad (1)$$

oznacza „ a przechodzi na c , b przechodzi na d , c przechodzi na f , d przechodzi na b , e przechodzi na e , f przechodzi na a ”. Gdy mówimy o przestawieniach, powyższy opis oznacza, że element c jest przestawiany na miejsce poprzednio zajmowane przez element a . Gdy mówimy o przemianowaniach, opis oznacza, że element a dostaje nazwę c . Przetworzenie kolumn nie zmienia znaczenia

zapisu w notacji dwuwierszowej. Powyższą permutację można zapisać na przykład jako

$$\begin{pmatrix} c & d & f & b & a & e \\ f & b & a & d & c & e \end{pmatrix}$$

i na 718 innych sposobów.

Innym naturalnym sposobem zapisu permutacji jest tzw. *notacja cyklowa*. Permutację (1) można zapisać jako

$$(a\ c\ f)\ (b\ d), \quad (2)$$

co oznacza „ a przechodzi na c , c przechodzi na f , f przechodzi na a , b przechodzi na d , d przechodzi na b ”. Cykl $(x_1 x_2 \dots x_n)$ oznacza „ x_1 przechodzi na x_2 , \dots , x_{n-1} przechodzi na x_n , x_n przechodzi na x_1 ”. Element e nie pojawia się w notacji cyklowej, ponieważ permutacja *zachowuje* ten element. Innymi słowy, cykle jednoelementowe, takie jak „ (e) ”, pomijamy w zapisie. Jeżeli permutacja zachowuje *wszystkie* elementy (tzn. gdy występują w niej wyłącznie cykle jednoelementowe), to jest nazywana *permutacją identycznościową*. Oznaczamy ją: „ $()$ ”.

Notacja cyklowa nie jest jednoznaczna. Na przykład

$$(b\ d)\ (a\ c\ f), \quad (c\ f\ a)\ (b\ d), \quad (d\ b)\ (f\ a\ c) \quad (3)$$

itp. są równoważne zapisowi (2). Jednakże „ $(a\ f\ c)\ (b\ d)$ ” nie jest tą samą permutacją, ponieważ a przechodzi na f .

Łatwo jest zrozumieć, dlaczego każdą permutację można zapisać w postaci cyklowej. Ustalony element x_1 jest przekształcany przez permutację w jakiś element x_2 , element x_2 jest przekształcany w x_3 itd., aż w końcu (skończona liczba elementów!) natrafimy na element x_{n+1} , który już pojawił się pośród x_1, \dots, x_n . Element x_{n+1} musi być równy x_1 . Dlaczego? Przypuśćmy na przykład, że $x_{n+1} = x_3$. Wiemy, że x_2 przechodzi na x_3 , ale z założenia $x_n \neq x_2$ przechodzi na x_{n+1} . Zatem $x_{n+1} = x_1$ i otrzymujemy cykl $(x_1 x_2 \dots x_n)$ będący częścią permutacji dla pewnego $n \geq 1$. Jeżeli ten cykl nie pokrywa wszystkich elementów, wybieramy jakiś nie pokryty element y_1 i tak samo konstruujemy następny cykl $(y_1 y_2 \dots y_m)$. Żaden z y nie może się równać katemukolwiek x , bo $x_i = y_j$ pociąga $x_{i+1} = y_{j+1}$ itd., więc dostaliśmy $x_k = y_1$ dla pewnego k , co jest niemożliwe, wobec sposobu wyboru y_1 . Powyższa metoda zawsze gwarantuje znalezienie wszystkich cykli.

Jedno z zastosowań tych idei do programowania pojawia się tam, gdzie trzeba zmienić kolejność ustawienia pewnych elementów. Jeżeli chcemy poprzedzić elementy bez przenoszenia ich w inne miejsce, musimy trzymać się układu cykli. Na przykład w celu wykonania przestawienia (1), tj. przypisania

$$(a, b, c, d, e, f) \leftarrow (c, d, f, b, e, a),$$

musimy zgodnie z układem cykli (2) kolejno przypisać

$$t \leftarrow a, \quad a \leftarrow c, \quad c \leftarrow f, \quad f \leftarrow t; \quad t \leftarrow b, \quad b \leftarrow d, \quad d \leftarrow t.$$

Dobrze jest zdawać sobie sprawę, że każde takie przekształcenie można rozbić na rozłączne cykle.

Iloczyny (składanie) permutacji. Możemy wymnożyć dwie permutacje, umawiając się, że mnożenie oznacza wykonanie jednej permutacji po drugiej. Jeśli na przykład po permutacji (1) wykonamy permutację

$$\begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix},$$

to a przejdzie na c i już tym c pozostanie; b przejdzie na d , a następnie na a itd.:

$$\begin{aligned} \begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \times \begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix} \\ = \begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \times \begin{pmatrix} c & d & f & b & e & a \\ c & a & e & d & f & b \end{pmatrix} \\ = \begin{pmatrix} a & b & c & d & e & f \\ c & a & e & d & f & b \end{pmatrix}. \end{aligned} \quad (4)$$

Powinno być jasne, że mnożenie permutacji nie jest przemienne. Innymi słowy, $\pi_1 \times \pi_2$ nie musi być równe $\pi_2 \times \pi_1$, gdzie π_1 i π_2 są permutacjami. Czytelnik może sprawdzić, że mnożenie (4) miałoby inny wynik, gdyby zamienić kolejność czynników (zobacz ćwiczenie 3).

Niektórzy ludzie mnożą permutacje od prawej do lewej, zamiast nieco bardziej naturalnie od lewej do prawej, jak w (4). W istocie zdania matematyków są podzielone, czy wynik wykonania przekształcenia T_1 , a potem T_2 , powinien być oznaczany przez T_1T_2 czy T_2T_1 . My używamy oznaczenia T_1T_2 .

Równość (4) można w notacji cyklowej zapisać następująco:

$$(a\ c\ f)\ (b\ d)\ (a\ b\ d)\ (e\ f) = (a\ c\ e\ f\ b). \quad (5)$$

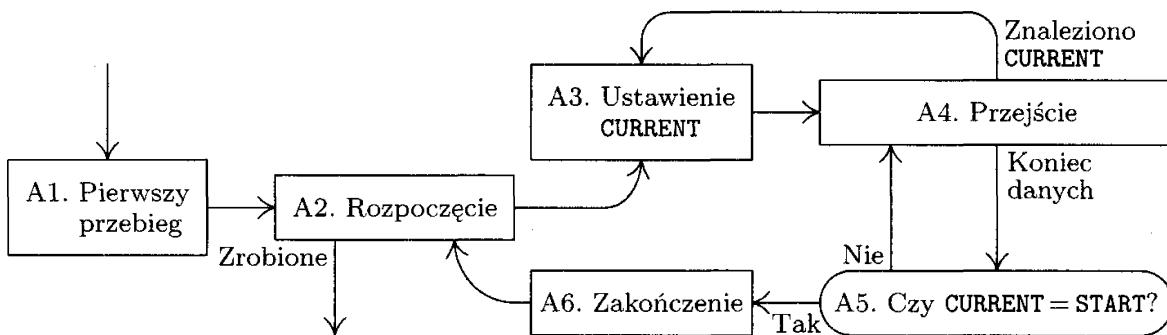
Zauważmy, że zazwyczaj opuszczamy znak mnożenia „ \times ”. Nie powoduje to konfliktu z notacją cyklową, bo łatwo dostrzec, że permutacja $(a\ c\ f)(b\ d)$ jest w istocie iloczynem permutacji $(a\ c\ f)$ i $(b\ d)$.

Mnożenie permutacji można przeprowadzać wprost w notacji cyklowej. Na przykład, obliczając iloczyn kilku permutacji

$$(a\ c\ f\ g)\ (b\ c\ d)\ (a\ e\ d)\ (f\ a\ d\ e)\ (b\ g\ f\ a\ e), \quad (6)$$

idąc od lewej do prawej, wnioskujemy, że „ a przechodzi na c , potem c przechodzi na d , potem d przechodzi na a , potem a przechodzi na d , potem d się nie zmienia”. Wynik zatem jest taki, że w permutacji (6) a przechodzi na d . Piszymy więc „ $(a\ d)$ ” jako część rozwiązania. Teraz analizujemy działanie permutacji na d : „ d przechodzi na b , przechodzi na g ”. Otrzymujemy fragment odpowiedzi: „ $(a\ d\ g)$ ”. Rozważając element g , odkrywamy, że „ g przechodzi na a , na e , na f , na a ”, zatem pierwszy cykl się zamyka: „ $(a\ d\ g)$ ”. Wybieramy teraz nowy element, który nie pojawił się do tej pory, na przykład c . Element c przechodzi na e . Czytelnik zechce sprawdzić poprawność końcowej odpowiedzi, „ $(a\ d\ g)(c\ e\ b)$ ” dla wzoru (6).

Spróbujmy skomputeryzować ten proces. Poniższy algorytm ujmuję metodę opisaną w poprzednim akapicie w sposób zrozumiały dla maszyny.



Rys. 20. Algorytm A, mnożenie permutacji.

Algorytm A (Mnożenie permutacji w postaci cyklowej). Algorytm na podstawie iloczynu cykli, jak na przykład (6), oblicza wynikową permutację w postaci iloczynu rozłącznych cykli. Dla prostoty nie opisujemy usuwania cykli jednoelementowych (dodanie tej funkcji do algorytmu nie nastręcza trudności). Podczas wykonania algorytmu kolejno „oznaczamy” elementy wyrażenia wejściowego. Dokładniej: w pewien sposób oznaczamy te symbole wyrażenia wejściowego, które zostały już przetworzone.

- A1. [Pierwszy przebieg] Oznacz wszystkie lewe nawiasy i zastąp każdy nawias zamykający oznaczoną kopią symbolu wejściowego, który występuje bezpośrednio po ostatnim nawiasie otwierającym (zobacz przykład w tabeli 1).
- A2. [Rozpoczęcie] Przeglądając od lewej do prawej, znajdź pierwszy nie oznaczony element. (Jeśli wszystkie elementy są oznaczone, wykonanie algorytmu się kończy). Zapamiętaj ten element w START. Wyprowadź nawias otwierający. Wyprowadź element. Oznacz element.
- A3. [Ustawienie CURRENT] Przypisz do CURRENT kolejny element wyrażenia.
- A4. [Przejście] Przejdź w prawo do końca wyrażenia lub do elementu równego CURRENT; w tym drugim przypadku oznacz go i przejdź do kroku A3.
- A5. [Czy CURRENT = START?] Jeśli $CURRENT \neq START$, wyprowadź CURRENT i powróć do kroku A4, rozpoczynając znowu od lewej strony (kontynuując tym samym budowanie cyklu w wyrażeniu wyjściowym).
- A6. [Zakończenie] (Cykl znaleziony i wyprowadzony). Wyprowadź nawias zamykający i powróć do kroku A2. ■

W tabeli 1 są pokazane kolejne kroki przetwarzania wyrażenia (6). Pierwszy wiersz tabeli zawiera wyrażenie po zamianie nawiasów zamykających na początek elementy odpowiednich cykli. W następnych wierszach jest pokazany stan po oznaczeniu kolejnych symboli. Na wyjściu pojawia się „(a d g)(c e b)(f)”. Zauważmy, że cykle jednoelementowe będą pojawiły się na wyjściu.

Program na komputer MIX. Kodując ten algorytm w języku maszyny MIX, do oznaczania możemy wykorzystać znak słowa. Założymy, że dane wejściowe są wydziurkowane na kartach następująco: 80-kolumnowa karta jest podzielona na szesnaście 5-znakowych pól. Każde pole ma jeden z poniższych formatów: (a) „ ” oznacza nawias otwierający (rozpoczynający opis cyklu);

Tabela 1
ALGORYTM A URUCHOMIONY DLA (6)

Po kroku START CURRENT (a c f g a (b c d b (a e d a (f a d e f (b g f a e b Wyjście		
A1		(a c f g a (b c d b (a e d a (f a d e f (b g f a e b
A2	a	(a <u>c</u> f g a (b c d b (a e d a (f a d e f (b g f a e b (a
A3	a	c (a <u>c</u> f g a (b c d b (a e d a (f a d e f (b g f a e b
A4	a	c (a c f g a (b <u>c</u> d b (a e d a (f a d e f (b g f a e b
A4	a	d (a c f g a (b c d b (a e d <u>a</u> (f a d e f (b g f a e b
A4	a	a (a c f g a (b c d b (a e d a (f a <u>d</u> e f (b g f a e b
A5	a	d (a c f g a (b c d b (a e d a (f a d e f (b g f a e b <u>b</u>) d
A5	a	g (a c f g a (b c d b (a e d a (f a d e f (b g f a e b <u>b</u>) g
A5	a	a (a c f g a (b c d b (a e d a (f a d e f (b g f a e b <u>b</u>)
A6	a	a (a c f g a (b c d b (a e d a (f a d e f (b g f a e b <u>b</u>))
A2	c	a (a <u>c</u> f g a (b c d b (a e d a (f a d e f (b g f a e b (c
A5	c	e (a c f g a (b c d b (a e d a (f a d e f (b g f a e b <u>b</u>) e
A5	c	b (a c f g a (b c d b (a e d a (f a d e f (b g f a e b <u>b</u>) b
A6	c	c (a c f g a (b c d b (a e d a (f a d e f (b g f a e b <u>b</u>))
A6	f	f (a c f g a (b c d b (a e d a (f a d e f (b g f a e b <u>b</u>) (f)

Znak I oznacza miejsce bezpośrednio za ostatnio rozpatrywanym symbolem; elementy oznaczone są jasnoszare.

(b) „ ” oznacza nawias zamykający (kończący opis cyklu); (c) „ ” (same spacje, tzw. słowo *białe*) – ten napis może zostać wstawiony gdziekolwiek (wypełnienie), (d) jakikolwiek inny napis oznacza element permutacji. Ostatnią kartę rozpoznajemy po tym, że w kolumnach 76–80 ma „ =”. Wzór (6) na przykład można wydziurkować na dwóch kartach jako:

(A C F G)	(B C D)	(A E D)	
(F A D E)	(B G F A E)		=

Dane wyjściowe wyprowadzane przez nasz program będzie stanowiła wierna kopia danych wejściowych, po której zostanie wyprowadzona odpowiedź.

Program A (*Mnożenie permutacji w notacji cyklowej*). Ten program realizuje algorytm A, zawiera obsługę wejścia-wyjścia, a także nie wyprowadza jednoelementowych cykli w permutacji wynikowej.

01 MAXWDS EQU 1200	Maksymalna długość danych wejściowych.
02 PERM ORIG **+MAXWDS	Permutacja wejściowa.
03 ANS ORIG **+MAXWDS	Miejsce na odpowiedź.
04 OUTBUF ORIG **+24	Miejsce do drukowania.
05 CARDS EQU 16	Nr jednostki czytnika kart.
06 PRINTER EQU 18	Nr jednostki drukarki.

07	BEGIN	IN PERM(CARDS)	Wczytaj pierwszą kartę.
08		ENT2 0	
09		LDA EQUALS	
10	1H	JBUS *(CARDS)	Czekaj na zakończenie cyklu.
11		CMPA PERM+15,2	
12		JE *+2	Czy to ostatnia karta?
13		IN PERM+16,2(CARDS)	Nie, wczytaj kolejną.
14		ENT1 OUTBUF	
15		JBUS *(PRINTER)	Drukuj kopię
16		MOVE PERM,2(16)	karty wejściowej.
17		OUT OUTBUF(PRINTER)	
18		JE 1F	
19		INC2 16	
20		CMP2 =MAXWDS-16=	
21		JLE 1B	Powtarzaj, dopóki są dane na wejściu.
22		HLT 666	Zbyt długie dane wejściowe!
23	1H	INC2 15	1 W tym miejscu rI2 słów z wejścia
24		ST2 SIZE	1 znajduje się w PERM, PERM + 1, ...
25		ENT3 0	1 <u>A1. Pierwszy przebieg.</u>
26	2H	LDAN PERM,3	A Pobierz następny element wejścia.
27		CMPA LPREN(1:5)	A Czy „,”?
28		JNE 1F	A
29		STA PERM,3	B Jeśli tak, to oznacz go.
30		INC3 1	B Następny niebiały symbol z wejścia
31		LDXN PERM,3 :	B do rX.
32		JXZ *-2	B
33	1H	CMPA RPREN(1:5)	C
34		JNE *+2	C
35		STX PERM,3	D Zastąp „,)” oznaczonym rX.
36		INC3 1	C
37		CMP3 SIZE	C Czy przetworzono wszystkie elementy?
38		JL 2B	C
39		LDA LPREN	1 Przygotuj część główną.
40		ENT1 ANS	1 rI1 = miejsce na odpowiedź.
41	OPEN	ENT3 0	E <u>A2. Rozpoczęcie.</u>
42	1H	LDXN PERM,3	F Szukaj nie oznaczonego elementu.
43		JXN GO	F
44		INC3 1	G
45		CMP3 SIZE	G
46		JL 1B	G
47	*		Koniec oznaczania. Wyprowadzamy wyniki.
48	DONE	CMP1 =ANS=	Czy otrzymaliśmy permutację
49		JNE *+2	identycznościową?
50		MOVE LPREN(2)	Jeśli tak, zmień na „()”.
51		MOVE =0=	23 białe słowa za odpowiedzią.
52		MOVE -1,1(22)	
53		ENT3 0	
54		OUT ANS,3(PRINTER)	

55	INC3 24	
56	LDX ANS,3	Drukuj odpowiednią liczbę wierszy.
57	JXNZ *-3	
58	HLT	
59 *		
60 LPREN ALF (Stałe używane w programie.
61 RPREN ALF)		
62 EQUALS ALF =		
63 *		
64 GO MOVE LPREN	H	Otwórz cykl.
65 MOVE PERM,3	H	
66 STX START	H	
67 SUCC STX PERM,3	J	Oznacz element.
68 INC3 1	J	Przesuń się o jeden w prawo.
69 LDXN PERM,3(1:5)	J	<u>A3. Przypisanie CURRENT</u> (tj. rX).
70 JXN 1F	J	Pomiń białe słowa.
71 JMP *-3	0	
72 5H STX 0,1	Q	Wyprowadź CURRENT.
73 INC1 1	Q	
74 ENT3 0	Q	Przeszukuj ponownie.
75 4H CMPX PERM,3(1:5)	K	<u>A4. Przechodzenie</u> .
76 JE SUCC	K	Element = CURRENT?
77 1H INC3 1	L	Przesuń się w prawo.
78 CMP3 SIZE	L	Koniec wyrażenia?
79 JL 4B	L	
80 CMPX START(1:5)	P	<u>A5. Czy CURRENT = START?</u>
81 JNE 5B	P	
82 CLOSE MOVE RPREN	R	<u>A6. Zakończenie</u> .
83 CMPA -3,1	R	Uwaga: rA = „.”.
84 JNE OPEN	R	
85 INC1 -3	S	Pomiń cykle jednoelementowe.
86 JMP OPEN	S	
87 END BEGIN		■

W powyższym programie występuje około 75 rozkazów, jest on nieco dłuższy od programów z poprzedniego rozdziału, a nawet od większości programów w tej książce. Jego rozmiar nie powinien jednak przerażać – kod jest podzielony na małe części, które są w znacznym stopniu niezależne. W wierszach 07–22 następuje wczytanie kart wejściowych i wydrukowanie kopii każdej karty. Wiersze 23–38 odpowiadają krokowi A1 algorytmu, czyli wstępнемu przetworzeniu danych wejściowych. W wierszach 39–46 i 64–86 jest wykonywane główne zadanie, a w wierszach 48–57 następuje wyprowadzenie odpowiedzi. Sugerujemy Czytelnikowi, by przestudiował tyle programów zawartych w tej książce, ile tylko może. Jest rzeczą niezmiernie istotną, by nauczyć się czytać programy pisane przez innych. Niestety, wiele kursów programowania przykłada do kształcenia tej umiejętności bardzo małą wagę, co prowadzi do potwornie niewydajnego wykorzystania sprzętu komputerowego.

Czas wykonania. W częściach programu A nie związanych z wejściem-wyjściem obok rozkazów podano liczby mówiące o tym, ile razy rozkaz zostanie wykonany (porównaj program 1.3.2M). Wiersz 30 wykona się na przykład B razy. Dla wygody założono, że w danych wejściowych nie występują ciągi białych słów, poza ewentualnie prawym końcem danych. Przy tym założeniu wiersz 71 nie wykonuje się nigdy, nie wykona się także skok w wierszu 32.

W wyniku dodawania można stwierdzić, że całkowity czas wykonania programu wynosi

$$(7 + 5A + 6B + 7C + 2D + E + 3F + 4G + 8H + 6J + 3K + 4L + 3P + 4Q + 6R + 2S)u \quad (7)$$

plus czas potrzebny na zrealizowanie operacji wejścia-wyjścia. By poznać znaczenie wzoru (7), musimy zbadać piętnaście niewiadomych $A, B, C, D, E, F, G, H, J, K, L, P, Q, R, S$ i powiązać je w jakiś sposób z charakterystyką danych wejściowych. Pokażemy teraz ogólne zasady radzenia sobie z takimi problemami.

Po pierwsze, stosujemy „prawo Kirchhoffa” z teorii obwodów elektrycznych: liczba wykonień rozkazu musi się równać liczbie przejść do tego rozkazu. Ta trywialna zasada zazwyczaj pozwala powiązać różne wartości w sposób zupełnie nietrywialny. Analizując przepływ sterowania w programie A, otrzymujemy następujące równania.

<i>Na podstawie wierszy</i>	<i>wnioskujemy, że</i>
26, 38	$A = 1 + (C - 1)$
33, 28	$C = B + (A - B)$
41, 84, 86	$E = 1 + R$
42, 46	$F = E + (G - 1)$
64, 43	$H = F - G$
67, 70, 76	$J = H + (K - (L - J))$
75, 79	$K = Q + (L - P)$
82, 72	$R = P - Q$

Równania uzyskane z prawa Kirchhoffa nie muszą być liniowo niezależne. Na przykład w naszym przypadku pierwsze i drugie równanie są równoważne. Ponadto ostatnie równanie można wyprowadzić z pozostałych, ponieważ równanie trzecie, czwarte i piąte daje $H = R$, zatem z szóstego mamy $K = L - R$. Co by jednak nie mówić, udało się wyrugować sześć z piętnastu niewiadomych:

$$A = C, \quad E = R + 1, \quad F = R + G, \quad H = R, \quad K = L - R, \quad Q = P - R. \quad (8)$$

Pierwsze prawo Kirchhoffa jest silnym narzędziem, analizujemy je bardziej szczegółowo w podpunkcie 2.3.4.1.

W kolejnym kroku spróbujemy powiązać zmienne z jakimiś cechami danych wejściowych. Analizując wiersze 24, 25, 30 i 36, stwierdzamy, że

$$B + C = \text{liczba słów na wejściu} = 16X - 1, \quad (9)$$

gdzie X jest liczbą kart wejściowych. Z wiersza 28

$$\begin{aligned} B &= \text{liczba znaków „(“ w danych wejściowych} \\ &= \text{liczba cykli w danych wejściowych} \end{aligned} \quad (10)$$

Podobnie z wiersza 34

$$\begin{aligned} D &= \text{liczba znaków „)” w danych wejściowych} \\ &= \text{liczba cykli w danych wejściowych.} \end{aligned} \quad (11)$$

Teraz z (10) i (11) otrzymujemy wniosek, którego nie wysnulibyśmy, używając jedynie prawa Kirchhoffa:

$$B = D. \quad (12)$$

Z wiersza 64

$$\begin{aligned} H &= \text{liczba cykli w danych wejściowych} \\ &\quad (\text{włącznie z cyklami jednoelementowymi}). \end{aligned} \quad (13)$$

Z wiersza 82 można wnioskować, że R jest tą samą wielkością. Fakt, że $H = R$ był w tym przypadku wyprowadzalny za pomocą prawa Kirchhoffa i pojawił się już w (8).

Opierając się na założeniu, że niebiałe słowa są oznaczone, oraz analizując wiersze 29, 35 i 67, dochodzimy do wniosku, że

$$J = Y - 2B, \quad (14)$$

gdzie Y jest liczbą słów niebiałych, występujących w permutacjach wejściowych. Z faktu, że każdy element permutacji wejściowej jest wyprowadzany tylko w jednej kopii w wierszu 65 lub 72, mamy

$$P = H + Q = \text{liczba różnych elementów na wyjściu}. \quad (15)$$

(Zobacz (8)). Po chwili zastanowienia możemy wyprowadzić to również z wiersza 80. Wreszcie, patrząc na wiersz 85, widzimy, że

$$S = \text{liczba cykli jednoelementowych na wyjściu}. \quad (16)$$

Wielkości B, C, H, J, P i S , które właśnie zinterpretowaliśmy, są niezależnymi parametrami, od których zależy czas działania programu A.

W otrzymanym przez nas wyniku do zbadania pozostają dwie niewiadome G i L . To wymaga nieco pomysłów. Przeszukiwanie danych wejściowych, które rozpoczyna się w wierszach 41 i 74, zawsze kończy się w wierszu 47 (ostatni raz) albo 80. Podczas tych $P + 1$ przebiegów rozkaz „INC3 1” jest wykonywany $B + C$ razy. Ma to miejsce w wierszach 44, 68 i 77. Dostajemy zatem nietrywialny związek

$$G + J + L = (B + C)(P + 1) \quad (17)$$

z którego wynika zależność między niewiadomymi G i L . Na szczęście całkowity czas wykonania (7) jest funkcją $G + L$ (wzór (7) zawiera $\dots + 3F + 4G \dots + 3K + 4L + \dots = \dots + 7G + \dots + 7L + \dots$), nie musimy więc dalej analizować G i L .

Reasumując: całkowity czas wykonania programu A, poza operacjami wejścia-wyjścia, sprowadza się do

$$(112NX + 304X - 2M - Y + 11U + 2V - 11)u. \quad (18)$$

W tym wzorze wstawiliśmy nowe nazwy wielkości zależnych od układu danych wejściowych, konkretnie:

$$\begin{aligned} X &= \text{liczba kart wejściowych}, \\ Y &= \text{liczba niebiałych pól na wejściu (poza kończącym „=”)}, \\ M &= \text{liczba cykli na wejściu}, \\ N &= \text{liczba różnych nazw elementów na wejściu}, \\ U &= \text{liczba cykli na wejściu (włączając cykle jednoelementowe)}, \\ V &= \text{liczba cykli jednoelementowych na wyjściu}. \end{aligned} \quad (19)$$

Mamy nadzieję, że przekonaliśmy Czytelnika, iż analiza programu (na przykład programu A) w wielu aspektach przypomina rozwiązywanie łamigłówki.

Poniżej pokażemy, że przy założeniu, iż permutacja wyjściowa jest losowa, wielkości U i V wynoszą średnio H_N i 1.

Inne podejście. Algorytm A mnoży permutacje tak, jak zazwyczaj robią to istoty ludzkie. Dosyć często okazuje się, że problemy rozwiązywane na komputerach są bardzo podobne do problemów, z którymi od wieków stykają się ludzie, zatem sposoby ich rozwiązywania wypraktykowane przez pokolenia śmiertelników, sprawdzają się także w wykonaniu maszyn.

Równie często odkrywamy jednak nowe metody, które okazują się być lepsze dla komputerów, chociaż nie nadają się dla ludzi. Podstawowy powód polega na tym, że komputer „myśli” inaczej. W inny sposób zapamiętuje fakty. Tę różnicę można dostrzec przy mnożeniu permutacji. Za pomocą poniższego algorytmu komputer wykonuje mnożenie w jednym przebiegu, zapamiętując cały aktualny stan permutacji podczas wymnażania jej cykli. „Ludzki” algorytm A przegląda formułę wiele razy, po jednym razie dla każdego elementu permutacji wyjściowej, natomiast nowy algorytm załatwia wszystko za jednym zamachem. Dla *Homo sapiens* bezbłędne wykonanie tego jest niemożliwe.

Jaka jest „komputerowa” metoda mnożenia permutacji? Zasadniczy pomysł jest pokazany w tabeli 2. Gdy wybierzymy jakiś znak z postaci cyklowej zapisanej w górnym wierszu tabeli, wówczas kolumna pod tym znakiem opisuje permutację powstałą z wymnożenia (częściowych) cykli zapisanych *na prawo* od wybranego znaku. Na przykład wymnożone (częściowe) cykle opisane przez fragment „... d e)(b g f a e)” składają się na permutację

$$\begin{pmatrix} a & b & c & d & e & f & g \\ e & g & c & b & ? & a & f \end{pmatrix},$$

która możemy odczytać z kolumny pod ostatnim znakiem d (czyli z dziesiątej kolumny od prawej strony).

Analizując tabelę 2, widzimy, że posuwając się od prawej do lewej, możemy ją zbudować systematycznie. Kolumna pod literą x różni się od swojej prawej sąsiadki (zawierającej poprzedni stan) tylko w wierszu x . Nowa wartość w wierszu x

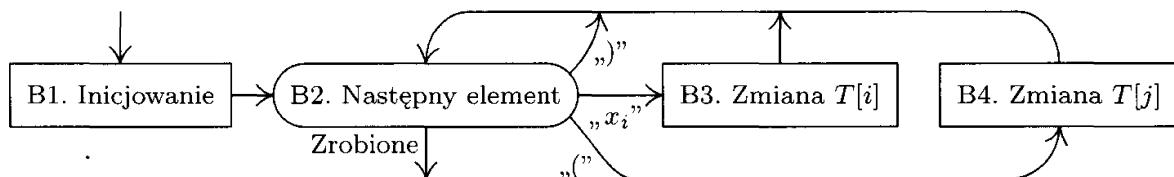
Tabela 2
MNOŻENIE PERMUTACJI W JEDNYM PRZEBIEGU

	$(a c f g) (b c d) (a e d) (f a d e) (b g f a e)$
$a \rightarrow$	$d d a a a a a a a a a a a a d d d d d d d d e e e e e e e e a a$
$b \rightarrow$	$c c c c c c c c g g g g g g g g g g g g g g g g b b b b b$
$c \rightarrow$	$e e e d d d d d d c c c c c c c c c c c c c c c c c c$
$d \rightarrow$	$g g g g g g g g))) d d))) b b b b b d d d d d d d d$
$e \rightarrow$	$b b b b b b b b b b b b a a a))) b b)))) e$
$f \rightarrow$	$f f f f f e e e e e e e e e a a a a a a a a f f f$
$g \rightarrow$	$a))) f f f f f f f f f f f f f f f f f f g g g g$

to dokładnie ta wartość, która zniknęła przy poprzedniej modyfikacji. Szczegóły podajemy w ścisłym opisie algorytmu.

Algorytm B (*Mnożenie permutacji w postaci cyklowej*). Ten algorytm wykonuje to samo zadanie co algorytm A. Założymy, że spermutowane elementy nazywają się x_1, x_2, \dots, x_n . Korzystamy z pomocniczej tablicy $T[1], T[2], \dots, T[n]$. Będziemy zachowywać warunek: we wczytanej permutacji x_i przechodzi na x_j wtedy i tylko wtedy, gdy $T[i] = j$.

- B1.** [Inicjowanie] Przyjmij $T[k] \leftarrow k$ dla $1 \leq k \leq n$. Inicjuj przeszukiwanie wejścia od prawej do lewej.
- B2.** [Następny element] Zbadaj następny element na wejściu (od prawej do lewej). Jeżeli ciąg wejściowy się skończył, zakończ algorytm. Jeżeli elementem wejściowym jest „)””, przypisz $Z \leftarrow 0$ i powtórz krok B2. Jeżeli elementem wejściowym jest „(”, przejdź do kroku B4. W przeciwnym razie element jest postaci x_i dla pewnego i . Przejdz do kroku B3.
- B3.** [Zmiana $T[i]$] Wymień $Z \leftrightarrow T[i]$. Jeżeli w wyniku tych przypisań $T[i] = 0$, to wykonaj $j \leftarrow i$. Powróć do kroku B2.
- B4.** [Zmiana $T[j]$] Przyjmij $T[j] \leftarrow Z$. (W tym miejscu j jest wierszem, w którym stoi „)”” w sensie tabeli 2, przy czym ten zamykający nawias odpowiada właśnie napotkanemu nawiasowi otwierającemu). Powróć do kroku B2.



Rys. 21. Algorytm B na mnożenie permutacji.

Oczywiście po wykonaniu tego algorytmu musimy wyprowadzić zawartość tabeli T w postaci cyklowej. Jak zaraz zobaczymy, daje się to łatwo zrealizować za pomocą metody „oznaczania”.

Napiszemy teraz program na komputer MIX oparty na zaprezentowanym algorytmie. Chcielibyśmy trzymać się tych samych wytycznych co w programie A, zachowując taką samą postać wejścia i wyjścia. Powstaje jednak mały problem: jak mamy zapisać algorytm B, nie wiedząc z góry, jakie będą elementy x_1, x_2, \dots, x_n ? Nie znamy n ; nie wiemy, czy element o nazwie b ma być x_1 czy x_2 itd. Prostą metodą rozwiązania tego problemu jest utrzymywanie tablicy nazw napotkanych elementów i każdorazowe wyszukiwanie bieżącego elementu według nazwy (zobacz wiersze 35–44 w poniższym programie).

Program B (*Robi to co program A*). $rX \equiv Z; rI4 \equiv i; rI1 \equiv j; rI3 = n$, liczba różnych napotkanych nazw.

01	MAXWDS	EQU	1200	Maksymalny rozmiar danych wejściowych.
02	X	ORIG	*+MAXWDS	Tablica nazw.
03	T	ORIG	*+MAXWDS	Pomocnicza tablica stanu.
04	PERM	ORIG	*+MAXWDS	Permutacja wejściowa.
05	ANS	EQU	PERM	Miejsce na odpowiedź.
06	OUTBUF	ORIG	*+24	Miejsce na wydruk.
07	CARDS	EQU	16	
	...			
24		HLT	666	
25	1H	INC2	15	Kopia wierszy 05–22 programu A. W tym miejscu rI2 słów z wejścia
26		ENT3	1	1 znajduje się w PERM, PERM + 1, ... 1 i nie zobaczyliśmy jeszcze żadnej nazwy.
27	RIGHT	ENTX	0	A Przypisanie $Z \leftarrow 0$.
28	SCAN	DEC2	1	B <u>B2. Następny element.</u>
29		LDA	PERM,2	B
30		JAZ	CYCLE	B Pomiń białe słowa.
31		CMPA	RPREN	C
32		JE	RIGHT	C Czy „)” jest następnym elementem?
33		CMPA	LPREN	D
34		JE	LEFT	D Czy jest to „(”?
35		ENT4	1,3	E Przygotuj przeszukiwanie.
36		STA	X	E Zapamiętaj początek tablicy.
37	2H	DEC4	1	F Przeszukuj tablicę nazw.
38		CMPA	X,4	F
39		JNE	2B	F Powtarzaj, dopóki nie znajdziesz.
40		J4P	FOUND	G Czy nazwa już się pojawiła?
41		INC3	1	H Nie, zwiększ rozmiar tablicy.
42		STA	X,3	H Wstaw nową nazwę x_n .
43		ST3	T,3	H Przypisz $T[n] \leftarrow n$,
44		ENT4	0,3	H $i \leftarrow n$.
45	FOUND	LDA	T,4	J <u>B3. Zmiana $T[i]$.</u>
46		STX	T,4	J Zapisz Z .
47		SRC	5	J Ustaw Z .
48		JANZ	SCAN	J
49		ENT1	0,4	K Jeśli Z było zerem, przypisz $j \leftarrow i$.
50		JMP	SCAN	K
51	LEFT	STX	T,1	L <u>B4. Zmiana $T[j]$.</u>
52	CYCLE	J2P	SCAN	P Jeśli nie koniec, to wróć do B2.
53	*			

54	OUTPUT ENT1 ANS	1	Przeszukano wszystkie dane wejściowe.
55	J3Z DONE	1	Tablice x i T zawierają odpowiedź.
56	1H LDAN X,3	Q	Konstruujemy postać cyklową.
57	JAP SKIP	Q	Czy nazwa jest oznaczona?
58	CMP3 T,3	R	Czy cykl jest jednoelementowy?
59	JE SKIP	R	
60	MOVE LPREN	S	Otwórz cykl.
61	2H MOVE X,3	T	
62	STA X,3	T	Oznacz nazwę.
63	LD3 T,3	T	Znajdź następnik elementu.
64	LDAN X,3	T	
65	JAN 2B	T	Oznaczone?
66	MOVE RPREN	W	Tak, zamknij cykl.
67	SKIP DEC3 1	Z	Przejdź do następnej nazwy.
68	J3P 1B	Z	
69	*		
70	DONE CMP1 =ANS=		
...			
84	EQUALS ALF =		
85	END BEGIN		

| } Kopia wierszy 48–62 programu A

Wiersze 54–68, służące do skonstruowania notacji cyklowej z tablicy T oraz tablicy nazw, to piękny, niewielki algorytm, który zasługuje na chwilę uwagi. Wielkości $A, B, \dots, R, S, T, W, Z$ składające się na całkowity czas wykonania programu są, oczywiście, różne od wielkości o tych samych nazwach, występujących w analizie programu A. Czytelnik zechce przeanalizować je w ramach ćwiczenia 10.

Doświadczenie pokazuje, że lvia część czasu wykonania programu B zostanie poświęcona na przeszukiwanie tablicy nazw (wielkość F). Znane są o wiele lepsze algorytmy przeszukiwania i budowania słowników nazw. Nazywa się je *algorytmami związonymi z tablicami symboli* i mają one olbrzymie znaczenie w praktycznych zastosowaniach. Efektywne algorytmy tablicy symboli omawiamy w rozdziale 6.

Permutacje odwrotne. Permutacją odwrotną π^- do permutacji π nazywamy przestawienie, które niweluje efekt przestawienia π . Jeżeli i przechodzi na j w permutacji π , to j przechodzi na i w permutacji π^- . Stąd iloczyn $\pi\pi^-$, a także $\pi^-\pi$, równa się permutacji identycznościowej. Często permutację odwrotną oznacza się pisząc π^{-1} zamiast π^- , ale jedynka jest nadmiarowa (podobnie jak $x^1 = x$).

Każda permutacja ma permutację odwrotną. Na przykład odwrotnością permutacji

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \quad \text{jest} \quad \begin{pmatrix} c & d & f & b & e & a \\ a & b & c & d & e & f \end{pmatrix} = \begin{pmatrix} a & b & c & d & e & f \\ f & d & a & b & e & c \end{pmatrix}.$$

Rozważymy teraz kilka prostych algorytmów wyznaczających permutacje odwrotne.

Na potrzeby tego rozdziału założymy, że pracujemy z permutacjami liczb $\{1, 2, \dots, n\}$. Jeżeli $X[1] X[2] \dots X[n]$ jest taką właśnie permutacją, to istnieje

prosta metoda obliczenia jej permutacji odwrotnej: wykonujemy $Y[X[k]] \leftarrow k$ dla $1 \leq k \leq n$. Wówczas $Y[1] Y[2] \dots Y[n]$ jest poszukiwaną permutacją. W tej metodzie używa się $2n$ komórek pamięci, n na X i n na Y .

Przypuśćmy jednak dla rozrywki, że n jest bardzo duże i chcielibyśmy obliczyć odwrotność $X[1] X[2] \dots X[n]$ bez korzystania z dodatkowej pamięci. Chcemy wykonać obliczenia „w miejscu”, tak by po wykonaniu algorytmu tablica $X[1] X[2] \dots X[n]$ opisywała odwrotność początkowej permutacji. Beztroskie przypisywanie $X[X[k]] \leftarrow k$ dla $1 \leq k \leq n$ nie doprowadzi do niczego dobrego, ale korzystając ze struktury cykli, możemy zaproponować następujący prosty algorytm.

Algorytm I (*Odwracanie permutacji w miejscu*). Zastępuje permutację liczb $\{1, 2, \dots, n\}$ opisaną przez $X[1] X[2] \dots X[n]$ jej permutacją odwrotną. Algorytm pochodzi od Bing-Chao Huanga [*Inf. Proc. Letters* **12** (1981), 237–238].

- I1.** [Inicjowanie] Przyjmij $m \leftarrow n$, $j \leftarrow -1$.
- I2.** [Następny element] Przyjmij $i \leftarrow X[m]$. Jeśli $i < 0$, przejdź do kroku I5 (element został już przetworzony).
- I3.** [Odwracanie] (W tym miejscu $j < 0$ i $i = X[m]$). Jeśli m nie jest największym elementem w swoim cyklu, to w pierwotnej permutacji $X[-j] = m$.
 $X[m] \leftarrow j$, $j \leftarrow -m$, $m \leftarrow i$, $i \leftarrow X[m]$.
- I4.** [Czy koniec cyklu?] Jeśli $i > 0$, to wróć do I3 (cykl się nie skończył). Jeśli nie, to przypisz $i \leftarrow j$. (W tym ostatnim przypadku w pierwotnej permutacji było $X[-j] = m$, a m jest największym elementem swojego cyklu).
- I5.** [Zapamiętanie wartości wynikowej] Przyjmij $X[m] \leftarrow -i$. (Pierwotnie $X[-i]$ równało się m).
- I6.** [Pętla względem m] Zmniejsz m o 1. Jeśli $m > 0$, to wróć do I2. W przeciwnym razie wykonanie algorytmu się kończy. ■

W tabeli 3 jest pokazane przykładowe działanie algorytmu. Metoda opiera się na odwracaniu kolejnych cykli permutacji. Elementy odwrócone są oznaczane za pomocą zmiany znaku na ujemny. Na koniec są przywracane dodatnie znaki.

Tabela 3
OBLICZANIE ODWROTNOŚCI 6 2 1 5 4 3 ZA POMOCĄ ALGORYTMU I

Po kroku:	I2	I3	I3	I3	I5*	I2	I3	I3	I5	I2	I5	I5	I3	I5	I5
$X[1]$	6	6	6	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	3
$X[2]$	2	2	2	2	2	2	2	2	2	2	2	2	-4	2	2
$X[3]$	1	1	-6	-6	-6	-6	-6	-6	-6	-6	6	6	6	6	6
$X[4]$	5	5	5	5	5	5	5	-5	-5	-5	5	5	5	5	5
$X[5]$	4	4	4	4	4	-1	-1	4	4	4	4	4	4	4	4
$X[6]$	3	-1	-1	-1	1	1	1	1	1	1	1	1	1	1	1
m	6	3	1	6	6	5	4	5	5	4	4	3	2	2	1
j	-1	-6	-3	-1	-1	-1	-5	-4	-4	-4	-4	-4	-2	-2	-2
i	3	1	6	-1	-1	4	5	-1	-4	-5	-5	-6	-4	-2	-3

Należy czytać kolumnami od lewej do prawej. W punkcie * cykl (1 6 3) został odwrócony.

Algorytm I przypomina część algorytmu A, a podobieństwo do algorytmu znajdującego cykle w programie B (wiersze 54–68) jest wprost uderzające. Jest to bardzo charakterystyczne dla wielu algorytmów operujących na permutacjach. Kodując algorytm, łatwo zauważać, że w rejestrze wygodniej przechowywać wartość $-i$ zamiast i .

Program I (*Odwracanie permutacji w miejscu*). $\text{rI1} \equiv m; \text{rI2} \equiv -i; \text{rI3} \equiv j; n = N$ (do zdefiniowania w większym programie zawierającym program I).

01	INVERT ENT1 N	1	<u>I1. Inicjowanie.</u> $m \leftarrow n$.
02	ENT3 -1	1	<u>I2. Następny element.</u> $j \leftarrow -1$.
03	2H LD2N X,1	N	<u>I3. Odwracanie.</u> $i \leftarrow X[m]$.
04	J2P 5F	N	Idź do I5, gdy $i < 0$.
05	3H ST3 X,1	N	<u>I3. Odwracanie.</u> $X[m] \leftarrow j$.
06	ENN3 0,1	N	$j \leftarrow -m$.
07	ENN1 0,2	N	$m \leftarrow i$.
08	LD2N X,1	N	$i \leftarrow X[m]$.
09	4H J2N 3B	N	<u>I4. Czy koniec cyklu?</u> Idź do I3, gdy $i > 0$.
10	ENN2 0,3	C	W przeciwnym razie przyjmij $i \leftarrow j$.
11	5H ST2 X,1	N	<u>I5. Zapamiętanie wartości wynikowej.</u> $X[m] \leftarrow -i$.
12	6H DEC1 1	N	<u>I6. Pętla względem m.</u>
13	J1P 2B	N	Idź do I2, gdy $m > 0$. ■

Czas wykonania programu łatwo wyznaczyć za pomocą metod pokazanych wcześniej. Każdemu elementowi $X[m]$ przypisujemy najpierw wartość ujemną (w kroku I3), a potem dodatnią (w kroku I5). Ogólny czas wykonania wynosi $(14N + C + 2)u$, gdzie N jest rozmiarem tablicy, a C jest liczbą cykli. Zachowanie C w losowej permutacji analizujemy poniżej.

Prawie zawsze istnieje kilka różnych algorytmów realizujących to samo zadanie, należy się zatem spodziewać, że permutacje można odwracać jeszcze w inny sposób. Poniższy pomysłowy algorytm zawdzięczamy J. Boothroydowi:

Algorytm J (*Odwracanie permutacji w miejscu*). Algorytm robi to samo co algorytm I, ale inaczej.

- J1. [Zanegowanie wszystkich] Przyjmij $X[k] \leftarrow -X[k]$, dla $1 \leq k \leq n$. Ponadto przyjmij $m \leftarrow n$.
- J2. [Inicjowanie j] Przyjmij $j \leftarrow m$.
- J3. [Znajdowanie ujemnej pozycji] Przyjmij $i \leftarrow X[j]$. Jeśli $i > 0$, to przyjmij $j \leftarrow i$ i powtórz ten krok.
- J4. [Odwracanie] Przyjmij $X[j] \leftarrow X[-i]$, $X[-i] \leftarrow m$.
- J5. [Pętla względem m] Zmniejsz m o 1. Jeśli $m > 0$, wróć do J2. W przeciwnym razie wykonanie algorytmu się kończy. ■

Przykład działania algorytmu Boothroyda widać w tabeli 4. Metoda znów opiera się na strukturze cykli, ale tym razem poprawność algorytmu nie jest taka oczywista! Sprawdzenie pozostawiamy Czytelnikowi (zobacz ćwiczenie 13).

Tabela 4
OBLCZANIE ODWROTNOSCI 6 2 1 5 4 3 ZA POMOCĄ ALGORYTMU J

Po kroku:	J2	J3	J5										
X[1]	-6	-6	-6	-6	-6	-6	-6	-6	3	3	3	3	3
X[2]	-2	-2	-2	-2	-2	-2	-2	-2	-2	2	2	2	2
X[3]	-1	-1	6	6	6	6	6	6	6	6	6	6	6
X[4]	-5	-5	-5	-5	5	5	5	5	5	5	5	5	5
X[5]	-4	-4	-4	-4	-5	-5	4	4	4	4	4	4	4
X[6]	-3	-3	-1	-1	-1	-1	-1	-1	-6	-6	-6	-6	1
m	6	6	5	5	4	4	3	3	2	2	1	1	0
i	-3	-3	-4	-4	-5	-5	-1	-1	-2	-2	-6	-6	-6
j	6	6	6	5	5	5	6	6	2	2	6	6	6

Program J (*Odpowiednik programu I*). rI1 $\equiv m$; rI2 $\equiv j$; rI3 $\equiv -i$.

01	INVERT ENN1 N	1	<u>J1. Zanegowanie wszystkich.</u>
02	ST1 X+N+1,1(0:0)	N	Ustaw znak ujemny.
03	INC1 1	N	
04	J1N *-2	N	Jeszcze?
05	ENT1 N	1	$m \leftarrow n$.
06	2H ENN3 0,1	N	<u>J2. Inicjowanie j. $i \leftarrow m$.</u>
07	ENN2 0,3	A	$j \leftarrow i$.
08	LD3N X,2	A	<u>J3. Znajdowanie ujemnej pozycji.</u>
09	J3N *-2	A	$i > 0?$
10	LDA X,3	N	<u>J4. Odwracanie.</u>
11	STA X,2	N	$X[j] \leftarrow X[-i]$.
12	ST1 X,3	N	$X[-i] \leftarrow m$.
13	DEC1 1	N	<u>J5. Pętla względem m.</u>
14	J1P 2B	N	Idź do J2, gdy $m > 0$. ■

By dowiedzieć się, jak szybko ten program działa, musimy zbadać wielkość A. Jest to zagadnienie tak ciekawe i pouczające, że poświęcamy mu ćwiczenie 14.

Choć algorytm J jest nie mniej pomysłowy niż algorytm I, z analizy widać, że algorytm I jest istotnie lepszy. W istocie średni czas wykonania algorytmu J jest proporcjonalny do $n \ln n$, podczas gdy dla algorytmu I – proporcjonalny do n . Może kiedyś znajdzie się jakieś zastosowanie dla algorytmu J (może po niewielkich modyfikacjach?). Jest taki ładny, że przecież nie można o nim ot po prostu zapomnieć.

Niezwykła odpowiedniość. Zauważliśmy już, że notacja cyklowa jest niejednoznaczna. Sześcioelementową permutację $(1\ 6\ 3)(4\ 5)$ można zapisać jako $(5\ 4)(3\ 1\ 6)$ itd. Dobrze byłoby znać postać kanoniczną notacji cyklowej. Postać kanoniczna jest jedyna. By uzyskać postać kanoniczną, należy:

- a) Zapisać jawnie wszystkie cykle jednoelementowe.
- b) W każdym cyklu liczbę najmniejszą umieścić na początku.
- c) Uporządkować cykle malejąco względem pierwszych elementów.

Na przykład dla $(3\ 1\ 6)(5\ 4)$ otrzymalibyśmy

$$(a): (3\ 1\ 6)(5\ 4)(2); \quad (b): (1\ 6\ 3)(4\ 5)(2); \quad (c): (4\ 5)(2)(1\ 6\ 3). \quad (20)$$

Kanoniczny zapis cyklowy ma tę ważną własność, że można w nim opuścić nawiasy. Nawiasy dają się zawsze jednoznacznie odtworzyć. Istnieje tylko jeden sposób wstawienia nawiasów pomiędzy „4 5 2 1 6 3” dający kanoniczny zapis cyklowy: nawias otwierający trzeba wstawić bezpośrednio przed każdym *minimum lewostronnym* (tj. bezpośrednio przed każdym elementem, którego nie poprzedzają elementy mniejsze).

To wstawianie i usuwanie nawiasów indukuje niezwykłą wzajemnie jednoznaczną odpowiedniość pomiędzy elementami zbioru wszystkich permutacji wyrażonych w postaci cyklowej i elementami zbioru wszystkich permutacji wyrażonych w postaci liniowej. Na przykład, permutacja 6 2 1 5 4 3 w kanonicznej postaci cyklowej to $(4\ 5)(2)(1\ 6\ 3)$. Usuwając nawiasy, otrzymamy 4 5 2 1 6 3, czyli w postaci cyklowej $(2\ 5\ 6\ 3)(1\ 4)$; usunięcie nawiasów daje 2 5 6 3 1 4, czyli w postaci cyklowej $(3\ 6\ 4)(1\ 2\ 5)$ itd.

Ta odpowiedniość ma liczne zastosowania do badania permutacji. Zapytajmy na przykład, „ile cykli ma średnio n -elementowa permutacja?” By odpowiedzieć na to pytanie, rozważamy zbiór wszystkich $n!$ permutacji zapisanych w kanonicznej postaci cyklowej i opuśćmy nawiasy. Otrzymujemy zbiór wszystkich $n!$ permutacji w pewnym porządku. Nasze pytanie można teraz sformułować jako „jaka jest średnia liczba minimów lewostronnych permutacji n -elementowej?” Na to pytanie odpowiedzieliśmy już w punkcie 1.2.10. To była wielkość $(A + 1)$ omawiana przy analizie algorytmu 1.2.10M, dla której wyliczyliśmy

$$\min 1, \quad \text{ave } H_n, \quad \max n, \quad \text{dev } \sqrt{H_n - H_n^{(2)}}. \quad (21)$$

(Po prawdzie omawialiśmy średnią liczbę maksimów prawostronnych, ale widać, że jest to zarazem dokładnie liczba minimów lewostronnych). Udowodniliśmy ponadto, że permutacja n -elementowa ma k minimów lewostronnych z prawdopodobieństwem $\begin{bmatrix} n \\ k \end{bmatrix}/n!$. Zatem permutacja n elementów ma k cykli z prawdopodobieństwem $\begin{bmatrix} n \\ k \end{bmatrix}/n!$.

Mögemy również zapytać o średnią odległość między minimami lewostronnymi, co odpowiada średniej długości cyklu. Na mocy (21), całkowita liczba cykli wśród $n!$ permutacji wynosi $n! H_n$, ponieważ jest równa $n!$ razy średnia liczba cykli. Jeżeli wybierzemy jeden z cykli losowo, to jaka jest średnia długość tego cyklu?

Wyobraźmy sobie wszystkie $n!$ permutacji zbioru $\{1, 2, \dots, n\}$ zapisanych w notacji cyklowej. Ile w tym zapisie występuje cykli długości trzy? By odpowiedzieć na to pytanie zastanówmy się, ile razy występuje konkretny 3-elementowy cykl $(x\ y\ z)$. Jasne jest, że w $(n-3)!$ permutacjach, bo to jest liczba sposobów, na które można ustawić pozostałe $n-3$ elementów. Liczba różnych możliwych cykli 3-elementowych $(x\ y\ z)$ wynosi $n(n-1)(n-2)/3$, ponieważ x można wybrać na n sposobów, y na $(n-1)$, a z na $(n-2)$, a pośród tych $n(n-1)(n-2)$ wyborów każdy cykl 3-elementowy pojawił się w trzech postaciach: $(x\ y\ z)$, $(y\ z\ x)$ i $(z\ x\ y)$. Zatem całkowita liczba cykli 3-elementowych wśród $n!$ permutacji wynosi $n(n-1)(n-2)/3$ razy $(n-3)!$, czyli $n!/3$. Analogicznie całkowita liczba cykli m -elementowych wynosi $n!/m$ dla $1 \leq m \leq n$. (W ten sposób uzyskujemy

innego prosty dowód faktu, że całkowita liczba cykli wynosi $n! H_n$. Stąd średnia liczba cykli w losowej permutacji wynosi H_n , jak już wcześniej zauważyliśmy). W ćwiczeniu 17 pokazujemy, że średnia długość losowo wybranego cyklu wynosi n/H_n , jeśli przyjmiemy, że $n! H_n$ cykli jest jednakowo prawdopodobnych. Ale jeśli w losowej permutacji wybierzemy losowo element, to średnia długość cyklu zawierającego ten element jest nieco większa niż n/H_n .

Brakuje nam jeszcze jednego elementu układanki, by analizę algorytmów A i B uznać za zakończoną. Chcielibyśmy znać średnią liczbę cykli jednoelementowych w losowej permutacji. To jest ciekawy problem. Przypuśćmy, że zapisaliśmy $n!$ permutacji, wymieniając w pierwszej kolejności te, które nie mają cykli jednoelementowych, potem te z jednym cyklem itd. Na przykład dla $n = 4$

brak punktów stałych:	2143 2341 2413 3142 3412 3421 4123 4312 4321
jeden punkt stały:	<u>1</u> 342 <u>1</u> 423 <u>3</u> 241 <u>4</u> 213 <u>2</u> 431 <u>4</u> 1 <u>3</u> 2 <u>2</u> 31 <u>4</u> <u>3</u> 12 <u>4</u>
dwa punkty stałe:	<u>1</u> 243 <u>1</u> 4 <u>3</u> 2 <u>1</u> 32 <u>4</u> <u>4</u> 231 <u>3</u> 21 <u>4</u> <u>2</u> 134
trzy punkty stałe:	
cztery punkty stałe:	<u>1</u> 234

(Na liście są zaznaczone punkty stałe. Cykle jednoelementowe odpowiadają punktom stałym permutacji).

Permutacje nie mające punktów stałych są nazywane *nieporządkami*; liczba nieporządków jest liczbą sposobów, na które można włożyć n listów do n kopert, tak żeby żaden adresat nie otrzymał właściwego listu.

Niech P_{nk} będzie liczbą permutacji n obiektów mających dokładnie k punktów stałych, zatem na przykład

$$P_{40} = 9, \quad P_{41} = 8, \quad P_{42} = 6, \quad P_{43} = 0, \quad P_{44} = 1.$$

Analizując powyższe liczby, zauważymy zasadniczy związek: możemy otrzymać wszystkie permutacje o k punktach stałych, wybierając najpierw k elementów, które mają być tymi punktami stałymi (można to zrobić na $\binom{n}{k}$ sposobów), a potem permutując pozostałe $n - k$ elementów na wszystkie $P_{(n-k)0}$ sposobów, żeby nie wstawić dodatkowych punktów stałych. Stąd

$$P_{nk} = \binom{n}{k} P_{(n-k)0}. \quad (22)$$

Możemy ponadto skorzystać z reguły „całość jest sumą swych części”:

$$n! = P_{nn} + P_{n(n-1)} + P_{n(n-2)} + P_{n(n-3)} + \dots \quad (23)$$

Zestawiając (22) i (23) oraz odrobinę przekształcając, otrzymujemy

$$n! = \frac{P_{00}}{0!} + n \frac{P_{10}}{1!} + n(n-1) \frac{P_{20}}{2!} + n(n-1)(n-2) \frac{P_{30}}{3!} + \dots \quad (24)$$

To równanie musi być prawdziwe dla wszystkich dodatnich liczb całkowitych n . Spotkaliśmy się z nim już wcześniej – pojawiło się w punkcie 1.2.5 w związku z próbą Stirlinga mającą na celu uogólnienie silni. W punkcie 1.2.6 znaleźliśmy

proste wyprowadzenie współczynników tego równania (przykład 5):

$$\frac{P_{m0}}{m!} = 1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^m \frac{1}{m!}. \quad (25)$$

Niech p_{nk} oznacza prawdopodobieństwo zdarzenia, że permutacja n -elementowa ma dokładnie k cykli jednoelementowych. Ponieważ $p_{nk} = P_{nk}/n!$, z (22) i (25) mamy

$$p_{nk} = \frac{1}{k!} \left(1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^{n-k} \frac{1}{(n-k)!} \right). \quad (26)$$

Funkcja tworząca $G_n(z) = p_{n0} + p_{n1}z + p_{n2}z^2 + \cdots$ jest zatem równa

$$G_n(z) = 1 + \frac{1}{1!}(z-1) + \cdots + \frac{1}{n!}(z-1)^n = \sum_{0 \leq j \leq n} \frac{1}{j!}(z-1)^j. \quad (27)$$

Z tego wzoru wynika, że $G'_n(z) = G_{n-1}(z)$. Za pomocą metod z punktu 1.2.10 uzyskujemy następującą charakterystykę liczby cykli jednoelementowych:

$$(\min 0, \text{ ave } 1, \text{ max } n, \text{ dev1}) \quad \text{dla } n \geq 2. \quad (28)$$

Nieco bardziej bezpośrednia metoda wyznaczenia liczby permutacji bez cykli jednoelementowych wynika z *zasady włączania i wyłączania*, będącej ważną metodą przy rozwiązywaniu problemów zliczania. Ogólną zasadę włączania i wyłączania można sformułować następująco: mamy N elementów i M podzbiorów S_1, S_2, \dots, S_M ; naszym celem jest policzenie, ile elementów nie należy do żadnego z podzbiorów. Niech $|S|$ oznacza liczbę elementów zbioru S . Wówczas poszukiwana liczba elementów spoza wszystkich S_j wynosi

$$\begin{aligned} N - \sum_{1 \leq j \leq M} |S_j| + \sum_{1 \leq j < k \leq M} |S_j \cap S_k| - \sum_{1 \leq i < j < k \leq M} |S_i \cap S_j \cap S_k| + \cdots \\ + (-1)^M |S_1 \cap \cdots \cap S_M|. \end{aligned} \quad (29)$$

(Najpierw odejmujemy liczbę elementów w S_1, \dots, S_M od liczby wszystkich elementów N . Ale w ten sposób zabieramy za dużo. Dodajemy z powrotem liczby elementów, które są wspólne dla par podzbiorów $S_j \cap S_k$, dla każdej pary S_j i S_k . Teraz z kolei dodaliśmy za dużo. Odejmujemy zatem elementy wspólne dla trójkę zbiorów itd.) Jest wiele sposobów udowodnienia tego wzoru. Zachęcamy Czytelnika do odkrycia jednego z nich. (Zobacz ćwiczenie 25).

By wyznaczyć liczbę permutacji n -elementowych nie mających cykli jednoelementowych, rozważamy $N = n!$ permutacji i bierzemy za S_j zbiór permutacji, w których element j tworzy cykl jednoelementowy. Jeśli $1 \leq j_1 < j_2 < \cdots < j_k \leq n$, to liczba elementów w $S_{j_1} \cap S_{j_2} \cap \cdots \cap S_{j_k}$ jest liczbą permutacji, w których j_1, \dots, j_k są cyklami jednoelementowymi, czyli $(n-k)!$. Zatem wzór (29) przyjmuje postać

$$n! - \binom{n}{1}(n-1)! + \binom{n}{2}(n-2)! - \binom{n}{3}(n-3)! + \cdots + (-1)^n \binom{n}{n} 0!,$$

zgodnie z (25).

Zasadę włączania i wyłączania zawdzięczamy A. de Moivre'owi [zobacz tegoż autora: *Doctrine of Chances* (Londyn: 1718), 61–63; wyd. 3 (1756, przedruk: Chelsea, 1957), 110–112], ale jej znaczenia nie doceniano, dopóki nie została spopularyzowana i rozwinięta przez W. A. Whitwortha w znanej książce *Choice and Chance* (Cambridge: 1867).

Kombinatoryczne własności permutacji zgłębiamy dalej w podrozdziale 5.1.

ĆWICZENIA

1. [02] Rozpatrzmy przekształcenie zbioru $\{0, 1, 2, 3, 4, 5, 6\}$, które przeprowadza x na $2x \bmod 7$. Pokaż, że to przekształcenie jest permutacją i zapisz je w postaci cyklowej.
 2. [10] W tekście było pokazane, że przypisania $(a, b, c, d, e, f) \leftarrow (c, d, f, b, e, a)$ można dokonać za pomocą ciągu operacji zastępowania ($x \leftarrow y$) i jednej pomocniczej zmiennej t . Pokaż, jak można uzyskać ten sam rezultat za pomocą ciągu operacji *wymiany* ($x \leftrightarrow y$) bez zmiennych pomocniczych.
 3. [03] Oblicz iloczyn $\begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix} \times \begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix}$ i przedstaw odpowiedź w notacji dwuwierszowej (porównaj z (4)).
 4. [10] Przedstaw $(a \ b \ d)(e \ f)(a \ c \ f)(b \ d)$ w postaci iloczynu rozłącznych cykli.
- 5. [M10] Równanie (3) pokazuje kilka równoważnych sposobów przedstawienia tej samej permutacji w postaci cyklowej. Na ile sposobów można zapisać tę permutację, jeżeli będziemy pomijać cykle jednoelementowe?
6. [M23] Jak zmieni się analiza czasu wykonania programu A, gdy pozbędziemy się założenia, że wszystkie białe słowa pojawiają się wyłącznie na prawym końcu?
7. [10] Jakie będą wartości X, Y, M, N, U i V w (19), gdy program A uruchomimy dla danych (6)? Jaki jest całkowity czas działania programu A (bez wejścia-wyjścia)?
- 8. [23] Czy da się tak zmodyfikować program B, żeby przechodził dane od lewej do prawej, zamiast od prawej do lewej?
9. [10] Programy A i B akceptują takie same dane wejściowe i generują dane wyjściowe w takiej samej postaci. Czy rzeczywiście oba programy generują *dokładnie* identyczne dane wyjściowe?
- 10. [M28] Zbadaj charakterystykę czasową programu B, tj. wielkości A, B, \dots, Z . Wyraź całkowity czas działania programu za pomocą wielkości X, Y, M, N, U, V zdefiniowanych w (19) oraz wielkości F . Porównaj całkowity czas działania programu A z całkowitym czasem działania programu B na danych wejściowych (6), korzystając z ćwiczenia 7.
11. [15] Znайдź prostą regułę na zapisywanie π^- w postaci cyklowej, gdy permutacja π jest dana w postaci cyklowej.
12. [M27] (*Transpozycja macierzy prostokątnej*) Przypuśćmy, że w pamięci zapisana jest macierz prostokątna (a_{ij}) wymiaru $m \times n$, $m \neq n$. Macierz jest zapisana tak jak w ćwiczeniu 1.3.2–10, zatem wartość a_{ij} jest przechowywana w lokacji $L + n(i - 1) + (j - 1)$, gdzie L jest lokacją zawierającą a_{11} . Problem polega na znalezieniu metody *przetransponowania* tej macierzy, tj. uzyskania takiej macierzy (b_{ij}) wymiaru $n \times m$, że wartość $b_{ij} = a_{ji}$ jest przechowywana w lokacji $L + m(i - 1) + (j - 1)$. Macierz ma zatem zostać przetransponowana „na samą siebie”. (a) Pokaż, że takie przekształcenie przenosi wartość z lokacji $L + x$ pod adres $L + (mx \bmod N)$ dla wszystkich x z zakresu $0 \leq x < N = mn - 1$. (b) Omów metody realizacji takiej transpozycji na komputerze.

- ▶ 13. [M24] Udowodnij, że algorytm J jest poprawny.
- ▶ 14. [M34] Znajdź średnią wartość wielkości A występującej w analizie czasu działania algorytmu J.
- 15. [M12] Czy istnieje permutacja, która ma dokładnie taki sam zapis w postaci liniowej i w kanonicznym zapisie cyklowym?
- 16. [M15] Zacznię od permutacji 1324 w zapisie liniowym. Przekształć ją do kanonicznego zapisu cyklowego i usuń nawiasy. Powtarzaj ten proces, aż do uzyskania początkowej permutacji. Jakie permutacje pojawiły się po drodze?
- 17. [M24] (a) W tekście powiedziano, że pośród permutacji n elementów występuje w sumie $n! H_n$ cykli. Jeżeli te cykle (włączając cykle jednoelementowe) zapiszemy na oddzielnych kartkach papieru, po jednym cyklu na kartce i wylosujemy jedną z tych kartek, to jaka będzie średnia długość wylosowanego cyklu? (b) Jeżeli zapisalibyśmy $n!$ permutacji na $n!$ kartkach papieru i wybrali losowo liczbę k oraz wylosowali kartkę, to jakie będzie prawdopodobieństwo zdarzenia, że cykl zawierający element k w permutacji na wylosowanej kartce jest cyklem m -elementowym? Jaka jest średnia długość cyklu zawierającego k ?
- ▶ 18. [M27] Ile wynosi p_{nkm} , tj. prawdopodobieństwo zdarzenia, że permutacja n -elementowa ma dokładnie k cykli długości m ? Jak wygląda adekwatna funkcja tworząca $G_{nm}(z)$? Jaka jest średnia liczba cykli m -elementowych, jakie jest odchylenie standarde? (W tekście rozważaliśmy jedynie przypadek $m = 1$).
- 19. [HM21] Pokaż, że przy oznaczeniach we wzorze (25) liczba nieporządków P_{n0} jest równa dokładnie wartości $n!/e$ zaokrąglonej do najbliższej liczby całkowitej dla wszystkich $n \geq 1$.
- 20. [M20] Założmy, że cykle jednoelementowe wypisujemy jawnie. Na ile sposobów można w notacji cyklowej zapisać przy tym założeniu permutację mającą α_1 cykli 1-elementowych, α_2 cykli 2-elementowych, ...? (Zobacz ćwiczenie 5).
- 21. [M22] Ile wynosi prawdopodobieństwo $P(n; \alpha_1, \alpha_2, \dots)$ zdarzenia, że permutacja n -elementowa ma dokładnie α_1 cykli 1-elementowych, α_2 cykli 2-elementowych itd.?
- ▶ 22. [HM34] (Poniższa metoda, autorstwa L. Sheppa i S. P. Lloyda, jest użytecznym i silnym narzędziem mającym zastosowanie do problemów związanych z cyklową strukturą losowych permutacji). Zamiast przyjmować, że liczba elementów n jest ustalona, wybieramy niezależnie wielkości $\alpha_1, \alpha_2, \alpha_3, \dots$ (ćwiczenia 20 i 21) zgodnie z pewnym rozkładem prawdopodobieństwa. Niech w będzie dowolną liczbą całkowitą między 0 a 1.
 - Przypuśćmy, że ustalamy zmienne losowe $\alpha_1, \alpha_2, \alpha_3, \dots$ zgodnie z zasadą „prawdopodobieństwo zdarzenia $\alpha_m = k$ jest równe $f(w, m, k)$ ” dla pewnej funkcji $f(w, m, k)$. Wyznacz tak wartości $f(w, m, k)$, by zachodziły następujące dwa warunki: (i) $\sum_{k \geq 0} f(w, m, k) = 1$, dla $0 < w < 1$ i $m \geq 1$; (ii) prawdopodobieństwo zdarzenia $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \dots = n$ oraz $\alpha_1 = k_1, \alpha_2 = k_2, \alpha_3 = k_3, \dots$ wynosi $(1-w)w^n P(n; k_1, k_2, k_3, \dots)$, gdzie $P(n; k_1, k_2, k_3, \dots)$ jest zdefiniowane w ćwiczeniu 21.
 - Permutacja, której struktura cyklowa jest postaci $\alpha_1, \alpha_2, \alpha_3, \dots$, dotyczy dokładnie $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \dots$ elementów. Pokaż, że jeśli wartości α są wybrane losowo zgodnie z rozkładem prawdopodobieństwa z punktu (a), to prawdopodobieństwo zdarzenia, że $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \dots = n$ wynosi $(1-w)w^n$. Prawdopodobieństwo, że suma $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \dots$ jest nieskończona wynosi zero.

- c) Niech $\phi(\alpha_1, \alpha_2, \dots)$ będzie dowolną funkcją nieskończonie wielu zmiennych $\alpha_1, \alpha_2, \dots$; pokaż, że jeśli liczby α są wybierane zgodnie z rozkładem prawdopodobieństwa z punktu (a), to średnia wartość ϕ wynosi $(1 - w) \sum_{n \geq 0} w^n \phi_n$; przyjmujemy, że ϕ_n oznacza średnią wartość ϕ braną po wszystkich permutacjach n elementów, gdzie zmienna α_j oznacza liczbę cykli j -elementowych permutacji. [Na przykład, gdy $\phi(\alpha_1, \alpha_2, \dots) = \alpha_1$, wartość ϕ_n jest średnią liczbą cykli jednoelementowych w losowej permutacji n -elementowej. Pokazaliśmy w (28), że $\phi_n = 1$, dla dowolnego n].
- d) Skorzystaj z powyższej metody, by znaleźć średnią liczbę cykli długości *parzystej* w losowej permutacji n -elementowej.
- e) Skorzystaj z powyższej metody, by rozwiązać ćwiczenie 18.

23. [HM42] (Golomb, Shepp, Lloyd) Niech l_n oznacza średnią długość *najdłuższego* cyklu w permutacji n -elementowej. Pokaż, że $l_n \approx \lambda n + \frac{1}{2}\lambda$, gdzie stała $\lambda \approx 0.62433$, a mówiąc ściślej udowodnij, że $\lim_{n \rightarrow \infty} (l_n - \lambda n - \frac{1}{2}\lambda) = 0$.

24. [M41] Znajdź wariancję wartości A występującej w analizie czasu działania algorytmu J. (Zobacz ćwiczenie 14).

25. [M22] Udowodnij wzór (29).

► **26.** [M24] Rozszerz zasadę włączania i wyłączania, tak aby uzyskać wzór na liczbę elementów, które należą dokładnie do r podzbiorów S_1, S_2, \dots, S_M . (W tekście rozważamy tylko przypadek $r = 0$).

27. [M20] Skorzystaj z zasady włączania i wyłączania, aby policzyć, ile jest liczb n w przedziale $0 \leq n < am_1m_2 \dots m_t$, które nie są podzielne przez żadną z liczb m_1, m_2, \dots, m_t . Liczby m_1, m_2, \dots, m_t oraz a są dodatnimi liczbami całkowitymi, takimi że $m_j \perp m_k$ gdy $j \neq k$.

28. [M21] (I. Kaplansky) Jeśli „permutacja Józefa Flawiusza” zdefiniowana w ćwiczeniu 1.3.2–22 zostanie przedstawiona w postaci cyklowej, to otrzymamy $(1\ 5\ 3\ 6\ 8\ 2\ 4)(7)$ dla $n = 8$ i $m = 4$. Pokaż, że ta permutacja w ogólnym przypadku jest iloczynem $(n\ n-1\ \dots\ 2\ 1)^{m-1}(n\ n-1\ \dots\ 2)^{m-1}\dots(n\ n-1)^{m-1}$.

29. [M25] Udowodnij, że postać cyklową permutacji Józefa Flawiusza dla $m = 2$ można otrzymać przez zapisanie w postaci cyklowej permutacji „podwajającej” zbioru $\{1, 2, \dots, 2n\}$, która przeprowadza j na $(2j) \bmod (2n+1)$, a następnie biorąc odbicie lustrzane (lewo-prawo) i opuszczając wszystkie liczby większe od n . Na przykład, gdy $n = 11$, permutacją podwajającą jest $(1\ 2\ 4\ 8\ 16\ 9\ 18\ 13\ 3\ 6\ 12)(5\ 10\ 20\ 17\ 11\ 22\ 21\ 19\ 15\ 7\ 14)$ a permutacją Józefa Flawiusza jest $(7\ 11\ 10\ 5)(6\ 3\ 9\ 8\ 4\ 2\ 1)$.

30. [M24] Skorzystaj z ćwiczenia 29, aby pokazać, że liczby całkowite uzyskiwane przez podstawienie dodatniej liczby całkowitej d do wzoru $(2^{d-1} - 1)(2n+1)/(2^d - 1)$, tworzą dokładnie zbiór punktów stałych permutacji Józefa Flawiusza dla $m = 2$.

31. [HM33] Uogólniając ćwiczenia 29 i 30, udowodnij, że k -ta zabijana osoba dla dowolnych m i n stoi na pozycji x , gdzie x obliczamy następująco: przypisujemy $x \leftarrow km$, następnie dopóty powtarzamy przypisanie $x \leftarrow \lfloor (m(x-n)-1)/(m-1) \rfloor$, dopóki nie otrzymamy $x \leq n$. Co za tym idzie, średnia liczba punktów stałych, dla $1 \leq n \leq N$ i ustalonego m przy $N \rightarrow \infty$, dąży do $\sum_{k \geq 1} (m-1)^k / (m^{k+1} - (m-1))^k$. [Ta wartość leży między $(m-1)/m$ i 1, zatem permutacje Józefa Flawiusza mają nieco więcej punktów stałych niż permutacje losowe].

32. [M25] (a) Udowodnij, że dla dowolnej permutacji $\pi = \pi_1 \pi_2 \dots \pi_{2m+1}$ postaci

$$\pi = (2\ 3)^{e_2} (4\ 5)^{e_4} \dots (2m\ 2m+1)^{e_{2m}} (1\ 2)^{e_1} (3\ 4)^{e_3} \dots (2m-1\ 2m)^{e_{2m-1}},$$

gdzie każde e_k jest 0 lub 1, zachodzi $|\pi_k - k| \leq 2$ dla $1 \leq k \leq 2m + 1$.

(b) Mając daną permutację ρ zbioru $\{1, 2, \dots, n\}$, skonstruuj taką permutację π powyższej postaci, że $\rho\pi$ jest pojedynczym cyklem. Wniosek: każda permutacja jest „bliska” cyklowi.

33. [M33] Pokaż, w jaki sposób dla $m = 2^{2^l}$ i $n = 2^{2^l+1}$ skonstruować ciągi permutacji $(\alpha_{j1}, \alpha_{j2}, \dots, \alpha_{jn}; \beta_{j1}, \beta_{j2}, \dots, \beta_{jn})$, $0 \leq j < m$, mające następującą własność „ortogonalności”:

$$\alpha_{i1}\beta_{j1}\alpha_{i2}\beta_{j2} \dots \alpha_{in}\beta_{jn} = \begin{cases} (1\ 2\ 3\ 4\ 5), & \text{jeśli } i = j; \\ (), & \text{jeśli } i \neq j. \end{cases}$$

Wszystkie α_{jk} i β_{jk} powinny być permutacjami zbioru $\{1, 2, 3, 4, 5\}$.

► **34.** [M25] (*Transpozycja bloków danych*) W praktyce programowania bardzo często napotykamy konieczność zamiany $\alpha\beta$ na $\beta\alpha$, gdzie α i β są fragmentami tablicy. Innymi słowy, chcemy zmienić tablicę o zawartości $x_0x_1\dots x_{m+n-1} = \alpha\beta$ w tablicę o zawartości $x_mx_{m+1}\dots x_{m+n-1}x_0x_1\dots x_{m-1} = \beta\alpha$, gdzie $x_0x_1\dots x_{m-1} = \alpha$ i $x_mx_{m+1}\dots x_{m+n-1} = \beta$. To jest permutacja zbioru $\{0, 1, \dots, m+n-1\}$, która przeprowadza k na $(k+m) \bmod (m+n)$. Pokaż, że każde takie „przesunięcie cykliczne” ma prostą strukturę cyklową i zbadaj tę strukturę. Zaproponuj algorytm realizujący opisane przekształcenie.

35. [M30] Pójdźmy za ciosem: niech $x_0x_1\dots x_{l+m+n-1} = \alpha\beta\gamma$, gdzie α , β i γ są napisami o długości odpowiednio l , m i n . Przypuśćmy, że chcemy zamienić $\alpha\beta\gamma$ w $\gamma\beta\alpha$. Pokaż, że odpowiednia permutacja ma prostą strukturę cyklową, co prowadzi do wydajnego algorytmu. [W ćwiczeniu 34 rozważaliśmy przypadek szczególny $m = 0$].
Wskazówka: Zastanów się nad zamianą $(\alpha\beta)(\gamma\beta)$ w $(\gamma\beta)(\alpha\beta)$.

36. [27] Napisz podprogram na komputer MIX realizujący algorytm podany w odpowiedzi do ćwiczenia 35 i przeanalizuj czas wykonania tego podprogramu. Porównaj go z czasem działania programu realizującego prostszy algorytm polegający na przejściu od $\alpha\beta\gamma$ do $(\alpha\beta\gamma)^R = \gamma^R\beta^R\alpha^R$, a następnie do $\gamma\beta\alpha$, gdzie σ^R oznacza odbicie lustrzane (lewo-prawie) napisu σ .

1.4. PODSTAWOWE METODY PROGRAMISTYCZNE

1.4.1. Podprogramy

Gdy pewne zadanie ma być wykonane w wielu różnych miejscach programu, z reguły nie należy kodować go za każdym razem. O wiele lepiej kod zadania (zwany *podprogramem*) umieścić w jednym miejscu, a do programu wstawić kilka instrukcji realizujących skoki do kodu zadania oraz powrót do programu głównego. Przeniesienie sterowania między programem głównym i podprogramem nazywamy *wywołaniem podprogramu*.

Każda maszyna realizuje wywołanie podprogramu na swój specyficzny sposób. Zazwyczaj przeznaczone są do tego specjalne instrukcje. Na maszynie MIX używamy do tego celu rejestru J. Nasze rozważania będą dotyczyć komputera MIX, ale dla innych maszyn sprawa wygląda podobnie.

Dzięki podprogramom oszczędzamy na pamięci. Nie oszczędzamy na czasie, z wyjątkiem oszczędności niejawnych, wynikających ze zmniejszenia rozmiaru programu (krótszy czas ładowania, lepsze wykorzystanie szybkiej pamięci na komputerach z pamięcią wielopoziomową itp.). Dodatkowy czas zużywany na wejście i wyjście z podprogramu jest zazwyczaj zaniedbywalny.

Podprogramy mają jeszcze kilka zalet. Dzięki nim łatwiej uwypuklić strukturę dużego i złożonego programu. Stosując podprogramy, można dokonać logicznego podziału problemu, a dzięki temu zazwyczaj łatwiej jest znaleźć błędy i uruchomić program. Wiele podprogramów jest cennych również z tego powodu, że mogą zostać wykorzystane przez programistów nie będących ich autorami.

Część systemów komputerowych ma wbudowane biblioteki pożytecznych podprogramów. Taka biblioteka znakomicie ułatwia pisanie programów rozwiązywających typowe problemy. Programista powinien jednak pisać podprogramy nie tylko do bibliotek. Nie zawsze podprogramy muszą być przeznaczone do ogólnego użytku. Podprogramy wyspecjalizowane są istotne w równej mierze, choć powstają na użytek jednego tylko programu. Podpunkt 1.4.3.1 zawiera kilka typowych przykładów.

Najprostsze podprogramy mają tylko jedno wejście i jedno wyjście, tak jak rozważany wcześniej podprogram **MAXIMUM** (zobacz punkt 1.3.2, program M). Dla wygody przytaczamy ten program w nieco zmienionej postaci – nowa wersja wyznacza maksimum zawartości stu komórek pamięci:

```
* MAKSUMUM X[1..100]
MAX100 STJ EXIT Ustaw adres powrotu.
          ENT3 100 M1. Inicjowanie.
          JMP 2F
1H      CMPA X,3 M3. Porownywanie.
          JGE *+3
2H      ENT2 0,3 M4. Zmienianie m.
          LDA  X,3 Jest nowe maksimum.
          DEC3 1 M5. Zmniejszanie k.
          J3P  1B M2. Czy wszystkie?
EXIT    JMP  *   Wróć do programu głównego. ■
```

(1)

W większym programie, zawierającym ten kod jako podprogram, pojedynczy rozkaz „JMP MAX100” spowoduje ustawienie rejestru A na wartość będącą maksimum wartości w lokacjach od $X+1$ do $X+100$, a pozycja tego maksimum pojawi się w rI2. Za obsługę wywołania podprogramu odpowiedzialne są w tym przypadku rozkazy „MAX100 STJ EXIT” i (poźniej) „EXIT JMP *”. Zgodnie z zasadami dotyczącymi rejestru J w komputerze MIX rozkaz w lokacji EXIT spowoduje skok do lokacji następnej po rozkazie, w którym wykonano skok pod adres MAX100.

 *Nowsze komputery, takie jak maszyna MMIX, mająca w przyszłości zastąpić maszynę MIX, dysponują lepszymi sposobami zapamiętywania adresu powrotnego. Podstawowa różnica polega na tym, że nie modyfikuje się rozkazów załadowanych do pamięci. Informacje są przechowywane w rejestrach lub specjalnej tablicy, a nie w samym programie. (Zobacz ćwiczenie 7). Kolejne wydanie tej książki będzie uwzględniało te nowe sposoby, ale na razie pozostaniemy przy staromodnej technice samomodyfikującego się kodu.*

Nietrudno uzyskać ilościowe informacje na temat oszczędzania na rozmiarze kodu i strat czasu spowodowanych wywołaniem podprogramów. Przypuśćmy, że fragment kodu zajmuje k lokacji i pojawia się w m miejscach programu. Zapisując go jako podprogram, potrzebujemy dodatkowego rozkazu STJ i rozkazu wyjścia z podprogramu oraz jednego rozkazu JMP w każdym z m miejsc wywołania. To daje w sumie $m + k + 2$ lokacji zamiast mk , zatem zysk na komórkach pamięci wynosi

$$(m - 1)(k - 1) - 3. \quad (2)$$

Jeżeli k lub m jest równe 1, to stosując podprogramy, nie zaoszczędzimy niczego. Jeśli j jest równe 2, to m musi być większe od 4, żeby interes się opłacał itd.

Strata czasu jest spowodowana wstawieniem dodatkowych rozkazów JMP, STJ i JMP i jest zerowa, jeżeli podprogram nie zostanie wywołany. Jeżeli podprogram jest wywoływany t razy, narzut czasowy wynosi $4tu$.

Powyższe oszacowania należy traktować nieco z przymrużeniem oka, ponieważ poczyniliśmy w nich założenia idealistyczne. Wielu podprogramów nie można wywoływać po prostu za pomocą jednego rozkazu JMP. Ponadto, jeśli to samo zadanie kodujemy w wielu miejscach programu, każde z wystąpień można dopasować do kontekstu, tak by wykorzystać szczególne cechy konkretnej części programu. Podprogram jednak musi być napisany w sposób uniwersalny, co zazwyczaj wymaga paru rozkazów więcej.

Działanie podprogramu uniwersalnego zależy od jego *parametrów*. Parametry mogą mieć różne wartości w różnych wywołaniach podprogramu.

Kod programu zewnętrznego, który przekazuje sterowanie do podprogramu, odpowiednio go uruchamiając, jest nazywany *sekwencją wywołania*. Konkretnie wartości parametrów, przekazywane uruchamianemu podprogramowi, są nazywane *argumentami*. Dla naszego podprogramu MAX100 sekwencją wywołania jest po prostu „JMP MAX100”, ale zazwyczaj niezbędne są dłuższe sekwencje, bo podprogramowi trzeba przekazać argumenty. Na przykład program 1.3.2M jest uogólnieniem MAX100 znajdującym największy z n pierwszych elementów tablicy.

Parametr n przekazujemy w rI1, a sekwencja wywołania składa się z dwóch kroków:

LD1 =n=	lub	ENT1 n
JMP MAXIMUM		JMP MAXIMUM

Jeżeli sekwencja wywołania zajmuje c lokacji, wzór (2) na ilość zaoszczędznej pamięci przyjmuje postać

$$(m - 1)(k - c) - \text{stała}, \quad (3)$$

a czas stracony na wywołanie programu nieco się zwiększa.

Ale to jeszcze nie wszystko. Czasami może zachodzić konieczność zapamiętania i odtworzenia wartości niektórych rejestrów. Na przykład dla podprogramu MAX100 musimy pamiętać, że pisząc „JMP MAX100”, nie tylko dostajemy maksimum w rejestrze A i pozycję w rejestrze I2, ale także zero w rejestrze I3. Podprogram może spowodować skasowanie zawartości rejestrów i nie wolno o tym zapominać. Jeżeli chcemy uchronić zawartość rI3 przed zniszczeniem w podprogramie MAX100, nie obejdziemy się bez wstawiania dodatkowych rozkazów. Na maszynie MIX najszybciej i najłatwiej zagwarantować zachowanie zawartości rI3, wstawiając „ST3 3F(0:2)” bezpośrednio po MAX100 oraz „3H ENT3 *” bezpośrednio przed EXIT. Całkowity koszt to dwa dodatkowe wiersze kodu oraz trzy cykle maszynowe na każde wywołanie podprogramu.

Podprogram można traktować jak *rozszerzenie* języka maszynowego. Jeśli podprogram MAX100 znajduje się w pamięci, dysponujemy pojedynczym rozkazem („JMP MAX100”) znajdującym maksimum. Jest ważne, by zdefiniować wynik działania każdego podprogramu tak precyzyjnie, jak robiliśmy to dla rozkazu maszynowego. Programista powinien *zapisać* specyfikację każdego podprogramu, nawet jeśli nikt inny nie będzie z tego podprogramu korzystał. W przypadku podprogramu MAXIMUM z punktu 1.3.2, specyfikacja powinna wyglądać następująco:

Sekwencja wywołania: JMP MAXIMUM.

Warunki wejściowe: $rI1 = n$; zakładamy, że $n \geq 1$.

Warunki wyjściowe: $rA = \max_{1 \leq k \leq n} \text{CONTENTS}(X + k) = \text{CONTENTS}(X + rI2); \quad rI3 = 0$; wartości rJ i CI ulegają zmianie. } (4)

(Zazwyczaj będziemy pomijali informacje, że w wyniku działania podprogramu rejestr J i wskaźnik porównania ulegają zmianie. Zamieściliśmy je tutaj wyłącznie dla pełności obrazu). Zauważmy, że rX i rI1 nie są modyfikowane przez podprogram. Jeśli byłoby inaczej, musielibyśmy wspomnieć o nich na liście warunków wyjściowych. Specyfikacja powinna zawierać także wszystkie lokacje pamięci nie związane bezpośrednio z podprogramem, których zawartość może zostać zmieniona. W powyższym przypadku należy rozumieć, że takie zmiany nie mają miejsca, gdyż w (4) się o nich nie wspomina.

Zastanówmy się teraz nad podprogramem o *wielu wejściach**. Założymy, że do pewnego podprogramułączamy uniwersalny podprogram MAXIMUM, ale

* „Podprogramu o wielu wejściach” (*multiple entrances subroutine*) nie należy mylić z „podprogramem wielowejściowym” (*multientranc subroutine*), czyli takim, który może być wykonywany jednocześnie przez wiele współbieżnych wątków lub procesów (przyp. tłum.).

program w wielu (acz nie wszystkich) przypadkach powinien korzystać ze specjalizowanej wersji MAX100, w której $n = 100$. Te dwa przypadki można połączyć następująco:

```
MAX100 ENT3 100 Pierwsze wejście.  
MAXN STJ EXIT Drugie wejście.  
JMP 2F Ciąg dalszy jak w (1).  
...  
EXIT JMP * Wróć do programu głównego. ■
```

(5)

Podprogram (5) jest taki sam jak (1), z dokładnością do przedstawienia dwóch pierwszych rozkazów. Skorzystaliśmy z faktu, że „ENT3” nie wpływa na zawartość rejestru J. Jeśli chcielibyśmy dodać trzecie wejście MAX50, moglibyśmy na początku umieścić kod

```
MAX50 ENT3 50  
JSJ MAXN
```

(6)

(Przypominamy, że „JSJ” oznacza skok bez modyfikacji zawartości rejestru J).

Gdy liczba parametrów jest niewielka, wygodnie przekazywać je do podprogramu albo w rejestrach (tak jak przekazywaliśmy n w rI3 do MAXN, czy też n w rI1 do MAXIMUM), albo w ustalonych komórkach pamięci.

Inną wygodną metodą przekazywania argumentów jest wypisanie ich *za rozkazem JMP*. Podprogram może odwołać się do parametrów na podstawie wartości rejestru J. Jeśli na przykład chcielibyśmy wykorzystać tę metodę przy wywołaniu MAXN, sekwencja wywołania wyglądałaby tak:

```
JMP MAXN  
CON n
```

(7)

Sam podprogram wyglądałby wówczas:

```
MAXN STJ **+1  
ENT1 * rI1 ← rJ.  
LD3 0,1 rI3 ← n.  
JMP 2F Ciąg dalszy jak w (1).  
...  
J3P 1B  
JMP 1,1 Powrót. ■
```

(8)

Ta konwencja jest szczególnie wygodna na maszynach, w których wywoływanie podprogramów realizuje się przez umieszczenie lokacji wyjścia w rejestrze indeksowym (na przykład System/360). Warto z niej korzystać również wtedy, gdy podprogram ma wiele parametrów albo gdy podprogramy są generowane przez kompilator.

Niestety, zazwyczaj nie daje się tej metody pogodzić z opisywaną wcześniej techniką wielu wejść. Możemy „oszukiwać”, pisząc

```
MAX100 STJ 1F  
JMP MAXN  
CON 100  
1H JMP *
```

ale to już jednak nie to, co (5).

Metoda podobna do wypisywania argumentów po rozkazie skoku jest zazwyczaj używana w podprogramach o *wielu wyjściach*. „Wiele wyjść” oznacza, że po zakończeniu wykonywania podprogramu ma nastąpić skok do jednej z kilku lokacji, zależnie od decyzji podjętych przez podprogram. Mówiąc ściśle, lokacja wyjściowa jest parametrem. Jeśli zatem podprogram ma się zakończyć skokiem w jedno z wielu miejsc, odpowiednie lokacje powinny zostać przekazane jako argumenty. Nasz ostatni przykład podprogramu „maksimum” ma dwa wejścia i dwa wyjścia. Sekwencja wywołania:

Ogólnie dla n	Dla $n = 100$
ENT3 n	
JMP MAXN	JMP MAX100
Wyjście, jeśli $\max \leq 0$ lub $\max \geq rX$.	Wyjście, jeśli $\max \leq 0$ lub $\max \geq rX$.
Wyjście, jeśli $0 < \max < rX$.	Wyjście, jeśli $0 < \max < rX$.

(Innymi słowy: przy wyjściu z podprogramu wykonywany jest skok do lokacji położonej o *dwie* lokacje za rozkazem wywołania podprogramu, jeżeli wartość maksimum jest dodatnia i mniejsza od zawartości rX). Podprogram wygląda następująco:

MAX100 ENT3 100	Wejście dla $n = 100$.
MAXN STJ EXIT	Wejście główne (dowolne n).
JMP 2F	Ciąg dalszy jak w (1).
...	
J3P 1B	
JANP EXIT	Wyjdź normalnie, jeśli $\max \leq 0$. (9)
STX TEMP	
CMPA TEMP	
JGE EXIT	Wyjdź normalnie, jeśli $\max \geq rX$.
ENT3 1	W przeciwnym razie wyjdź drugim wyjściem.
EXIT JMP *,3	Powrót do odpowiedniego miejsca. ■

Podprogramy mogą wywoływać kolejne podprogramy. W skomplikowanych programach zdarzają się nawet wywołania zagnieżdżone pięciokrotnie. Jedyne ograniczenie, wynikające z używanego przez nas sposobu wywoływania podprogramów, polega na tym, że podprogram nie może wywołać podprogramu, który akurat go (pośrednio lub bezpośrednio) wywołuje. Rozważmy na przykład następujący scenariusz:

[Program główny]	[Podprogram A]	[Podprogram B]	[Podprogram C]
	A STJ EXITA	B STJ EXITB	C STJ EXITC
:	:	:	:
JMP A	JMP B	JMP C	JMP A
:	:	:	:
	EXITA JMP *	EXITB JMP *	EXITC JMP *
			(10)

Program główny wywołuje A, A wywołuje B, B wywołuje C, a C znowu woła A. Adres w lokacji EXITA odpowiadający powrotowi do programu głównego zostanie zamazany, powrotu nie da się zrealizować. Podobna uwaga dotyczy komórek pamięci i rejestrów używanych przez podprogramy. Nietrudno zaproponować sposób wywoływanie podprogramów zapewniający poprawną obsługę takich sytuacji. *Rekursję* omawiamy szczegółowo w rozdziale 8.

Podsumujemy ten rozdział krótkimi rozważaniami pod roboczym tytułem „jak pisać program długi i złożony”. Skąd mamy wiedzieć, jakie rodzaje podprogramów będą potrzebne? Jakie powinny być sekwencje wywołania? Jednym ze sprawdzonych sposobów jest posłużenie się poniższą procedurą iteracyjną:

Krok 0 (Pomysły wstępne). Po pierwsze, ustalamy w zarysach plan działania.

Krok 1 (Zarys programu). Zaczynamy od napisania „warstw zewnętrznych” programu (w jakimkolwiek języku). Systematyczną metodę realizacji tego zadania bardzo ładnie opisał E. W. Dijkstra, *Structured Programming* (Academic Press, 1972), rozdział 1, oraz N. Wirth, *CACM* 14 (1971), 221–227. Możemy zacząć od podzielenia całego programu na kilka fragmentów, które na razie można traktować jak podprogramy (wywoływane tylko raz). Te fragmenty dalej dzielimy na coraz mniejsze części, odpowiedzialne za coraz prostsze zadania. Gdy napotkamy zadanie, które już gdzieś się pojawiło (lub prawdopodobnie gdzieś powtórnie się pojawi), definiujemy (prawdziwy) podprogram do jego realizacji. Nie kodujemy go jednak póki co. Piszemy dalej program główny, zakładając, że dysponujemy podprogramem poprawnie wykonującym swoje zadanie. Gdy wreszcie program główny jest jako taki zarysowany, zabieramy się za podprogramy. Rozpoczynamy od tych najbardziej złożonych, by w dalszej kolejności przejść do podpodprogramów itd. W ten sposób otrzymujemy listę podprogramów. Funkcja każdego z podprogramów zapewne zmieniła się po drodze kilka razy, zatem niektóre części programu stały się niepoprawne. Ale nie ma sprawy, przecież to tylko zarys. Mamy już ugruntowane pojęcie o tym, co powinien robić każdy z podprogramów i na ile powinien być uniwersalny. Zazwyczaj opłaca się w tym miejscu specyfikacje podprogramów nieco uogólnić.

Krok 2 (Pierwszy działający program). W tym kroku posuwamy się w przeciwnym kierunku niż w kroku 1. Piszemy teraz w języku programowania, na przykład w języku MIXAL, PL/MIX lub w języku wysokiego poziomu. Zaczynamy od podprogramów najniższego poziomu, program główny zostawiając na koniec. Na ile to tylko możliwe, nie piszemy wywołań podprogramów, które nie zostały jeszcze zakodowane. (W kroku 1 postępowaliśmy odwrotnie: podprogramem zajmowaliśmy się dopiero po napisaniu wszystkich jego wywołań).

Wraz z liczbą zakodowanych podprogramów rośnie nasze zadowolenie i funkcjonalność powstającego programu. Po zakodowaniu każdego podprogramu powinniśmy natychmiast przygotowywać kompletny opis jego działania oraz sekwencji wywołania, jak w (4). Ważne jest także, by komórki pamięci tymczasowej różnych podprogramów nie zachodziły na siebie. Sytuacja, w której każdy podprogram korzysta z lokacji TEMP, miałaby opłakane skutki, chociaż w kroku 1

wygodnie abstrahować od takich szczegółów. Najprostszym sposobem ominięcia takich problemów jest trzymanie się zasad, że każdy podprogram korzysta wyłącznie ze swojej pamięci tymczasowej. Jeżeli jednak jest to zbyt dużą rozrzutnością, można skorzystać z innego schematu: komórkom pamięci tymczasowej nadajemy nazwy TEMP1, TEMP2 itd., a numeracja w podprogramie zaczyna się od TEMP j , gdzie j jest o jeden większe niż największy numer wykorzystany w podpodprogramach tego podprogramu.

Krok 3 (Powtórne sprawdzenie). W drugim kroku otrzymujemy produkt, któremu bardzo blisko do działającego programu. Ale można go zapewne ulepszyć. Znów inaczej patrzymy na strukturę kodu: dla każdego podprogramu badamy *wszystkie* jego wywołania. Może się okazać, że podprogram powinien zostać poszerzony o akcje, które zawsze są wykonywane przed lub po jego wywołaniu. Zapewne kilka podprogramów można połączyć w jeden. Okaże się też być może, że niektóre podprogramy wywoływanie są tylko raz, więc wcale podprogramami być nie powinny. (A może jakieś podprogramy nie są wywoływanie wcale i można je w ogóle pominać?)

Nie jest złym pomysłem, by w tym momencie wyrzucić cały program do kosza i wrócić do kroku 1. Wbrew pozorom to nie jest propozycja ironiczna. Czas, który spędziliśmy do tej pory nad programem, nie poszedł na marne – zdobyliśmy olbrzymią wiedzę o istocie problemu. Dysponując doświadczeniem, odkryliśmy zapewne kilka poprawek, które należały wprowadzić do pierwotnego projektu. Nie należy się obawiać powrotu do pierwszego kroku – tym razem kroki 2 i 3 pójdu o wiele łatwiej! Ponadto na późniejszym wyszukiwaniu błędów zaoszczędzimy zapewne tyle czasu, ile zajmie przepisanie wszystkiego od nowa. Niektóre znakomite programy zawdzięczają swoją jakość temu, że w trakcie ich powstawania cały dorobek nagle przepadał, a autorzy musieli zaczynać pracę od początku.

Z drugiej strony nie można zapewne osiągnąć takiego stanu, by złożonego programu komputerowego nie dawało się jeszcze jakoś poprawić, zatem kroki 1 i 2 należały powtarzać. Gdy widać możliwości wprowadzenia istotnych poprawek, warto poświęcić dodatkowy czas i jeszcze raz zacząć od początku. W końcu dochodzimy jednak do stanu, w którym nie opłaca się inwestować więcej czasu i pracy.

Krok 4 (Uruchamianie programu). Po ostatecznym dopieszczeniu szczegółów, obejmujących zapewne przydział pamięci i tego typu detale, przychodzi czas na obejrzenie programu pod jeszcze innym kątem niż w krokach 1–3. Będziemy przyglądać się kolejności *wykonania* jego składowych. Można to robić ręcznie, ale można oczywiście skorzystać z komputera. Autor posługiwał się na przykład programami systemowymi śledzącymi dwa pierwsze wykonania każdej instrukcji. Istotne jest, by jeszcze raz przemyśleć założenia programu oraz sprawdzić, czy wszystko działa zgodnie z oczekiwaniemi.

Uruchamianie programu jest sztuką wymagającą głębszych studiów. Wiele zależy od możliwości i konfiguracji danego komputera. Zalecanym punktem wyjścia jest przygotowanie odpowiednich testów. Wydaje się, że najbardziej wydajne metody znajdowania błędów w programach opierają się na mechanizmach wbudowanych w język programowania.

dowanych w program – wielu najlepszych współczesnych programistów poświęca prawie połowę programu na zakodowanie mechanizmów wyszukiwania błędów w tej drugiej połowie. Zazwyczaj sprowadza się to do dosyć prostych podprogramów wyświetlających istotne informacje w czytelnej postaci. Te podprogramy w końcu się wyrzuca, ale i tak wzrost produktywności jest zadziwiający.

Inna dobrą praktyką jest zapisywanie wszystkich popełnianych błędów. Może się to wydawać kłopotliwe, tym niemniej warto to robić. Zebrane informacje są bezcenne dla osób szukających przyczyn jakichś skomplikowanych błędów, a także pozwalają nam samym ustrzec się przed błędami w przyszłości.

Uwaga: Większość z powyższych komentarzy powstała w roku 1964, po tym jak autor z powodzeniem zakończył pracę nad kilkoma średnimi projektami programistycznymi, ale zanim wyrobił sobie dojrzały styl programowania. W latach osiemdziesiątych autor zrozumiał, że metoda, zwana *dokumentacją strukturalną* lub programowaniem czytelnym (*literate programming*) jest zapewne jeszcze ważniejsza niż styl. Przegląd aktualnych przekonań autora na temat najlepszych sposobów pisania wszelkich programów zawarto w książce *Literate Programming* (Cambridge Univ. Press, pierwsze wydanie w 1992 roku). Tak się składa, że rozdział 11 tej książki zawiera szczegółową listę wszystkich błędów usuniętych z programu *TeX* w latach 1978–1991.

*Czasami lepiej machnąć ręką na błędy,
niż poświęcić czas na całkowite ich wyeliminowanie
(ile lat by nam to zajęło?).*

— A. M. TURING, Proposals for ACE (1945)

ĆWICZENIA

1. [10] Zapisz specyfikację podprogramu (5) w taki sposób, jak w (4) jest opisana specyfikacja podprogramu 1.3.2M.
2. [10] Zaproponuj kod nie zawierający rozkazu JSJ, którym można zastąpić (6).
3. [M15] Uzupełnij informacje w zestawieniu (4), podając dokładnie, co stanie się z rejestrzem J i wskaźnikiem porównania w wyniku wykonania podprogramu. Podaj również, co się stanie, gdy zawartość rejestru I1 nie jest dodatnia.
- ▶ 4. [21] Napisz podprogram, który jest uogólnieniem MAXN, tj. znajduje największą wartość spośród $X[a], X[a+r], X[a+2r], \dots, X[n]$, gdzie r i n są parametrami, a a jest najmniejszą liczbą dodatnią, dla której $a \equiv n \pmod{r}$, tj. $a = 1 + (n - 1) \bmod r$. Zadbaj o specjalne wejście dla przypadku $r = 1$. Napisz specyfikację Twojego podprogramu, jak w (4).
5. [21] Co by było, gdyby MIX nie miał rejestru J? Wymyśl metodę wywoływania podprogramów, w której nie korzysta się z rejestru J, i zilustruj swój pomysł, zapisując podprogram MAX100 równoważny podprogramowi (1). Podaj specyfikację tego podprogramu w stylu (4). (Pozostań przy konwencji samomodyfikującego się kodu stosowanej w programach na komputer MIX).
- ▶ 6. [26] Przypuśćmy, że MIX nie ma rozkazu MOVE. Napisz podprogram zatytułowany MOVE, który wywołany za pomocą rozkazów „JMP MOVE; NOP A,I(F)” robi to, co rozkaz „MOVE A,I(F)”, jeśli taki rozkaz jest dozwolony. Jedyne różnice powinny sprowadzać się do modyfikacji zawartości rejestru J i nieco dłuższego czasu wykonania.
- ▶ 7. [20] Dlaczego kod samomodyfikujący się popadł w niełaskę?

1.4.2. Współprogramy

Podprogramy można uważać za szczególny przypadek składowych programu zwanych *współprogramami (coroutines)*. W przeciwieństwie do niesymetrycznej relacji między programem głównym a podprogramem, w przypadku współprogramów występuje symetria – współprogramy *wywołują się nawzajem*.

By zrozumieć koncepcję współprogramów, zastanówmy się nad naszym pojmovaniem podprogramów. Poprzedni punkt ukazuje podprogramy jako mechanizm wzbogacania listy rozkazów, pozwalający zmniejszyć rozmiar kodu. Jest to podejście poprawne, można jednak spojrzeć na podprogramy inaczej: po patrzmy na program główny i podprogram jak na *zespoł programów*, którego każdy członek ma określone zadanie do wykonania. Program główny, realizując swoje zadanie, aktywuje podprogram. Podprogram zrealizuje to, co do niego należy, i aktywuje program główny. Możemy wysilić wyobraźnię i przekonać się, że z punktu widzenia podprogramu, wyjście wygląda tak, jak *wywołanie programu głównego*. Program główny wykonuje swój kod, a potem „wraça” do podprogramu. Podprogram za chwilę znowu „wywołuje” program główny itd.

Taka właśnie filozofia leży u podstaw współprogramów, czyli tak napisanych fragmentów kodu, że trudno powiedzieć, który jest podprogramem którego. Przypuszcmy, że kodujemy współprogramy A i B. Kodując A, myślimy o „podprogramie” B, ale kodując B, możemy w ten sam sposób myśleć o A. We współprogramie A do aktywowania współprogramu B używamy rozkazu „JMP B”. We współprogramie B do ponownego aktywowania A używamy rozkazu „JMP A”. Aktywowany współprogram jest wykonywany od tego miejsca, gdzie ostatnio jego działanie zostało zawieszone.

Współprogramy A i B mogą być na przykład programami grającymi w szachy. Możemy połączyć je w ten sposób, żeby grały przeciwko sobie.

Na komputerze MIX takie wzajemne aktywowanie się współprogramów A i B uzyskujemy za pomocą następujących rozkazów:

A STJ BX	B STJ AX	
AX JMP A1	BX JMP B1	

(1)

Jednokrotne przeniesienie sterowania wymaga w tym przypadku czterech cykli maszynowych. Początkowo w AX i BX znajdują się skoki do początków współprogramów, A1 i B1. Przypuśćmy, że chcemy rozpocząć wykonanie programu od współprogramu A, od lokacji A1. Gdy współprogram wykonuje „JMP B” z pewnej lokacji A2, rozkaz z lokacji B zapamiętuje zawartość rJ w lokacji AX, w wyniku czego pojawia się tam rozkaz „JMP A2+1”. Rozkaz BX przerzuca nas do lokacji B1 i współprogram B rozpoczyna działanie, by po jakimś czasie dojść do rozkazu „JMP A” w pewnej lokacji B2. Zawartość rJ zostaje zapamiętana w BX, po czym następuje skok do lokacji A2+1, powodujący wznowienie wykonania współprogramu A do chwili ponownego skoku do B, który zapisze zawartość rJ do AX i wykona skok do B2+1 itd.

Jak widać, podstawowa różnica między wywoływaniem podprogramów a aktywowaniem współprogramów polega na tym, że podprogram jest zawsze akty-

wowany od początku, co zazwyczaj oznacza ustalone miejsce. Program główny albo współprogram jest zawsze aktywowany bezpośrednio za punktem, w którym został ostatnio wstrzymany.

Naturalnym zastosowaniem współprogramów są algorytmy związane z wejściem-wyjściem. Na przykład zadaniem współprogramu A może być wczytywanie kart perforowanych i proste przekształcanie wczytanych danych. Inny współprogram o nazwie B odpowiada za dalsze przetwarzanie i drukuje wyniki. Współprogram B co jakiś czas będzie sięgać po kolejne elementy wejścia wczytane przez współprogram A. Innymi słowy, B będzie wykonywał skok do A, by pobrać kolejny element z wejścia. Współprogram A będzie wykonywał skok do B, gdy wczyta element z wejścia. Czytelnik powie: „Zaraz, zaraz, w takim razie to B jest programem głównym, a A jest po prostu podprogramem odpowiedzialnym za wczytywanie danych”. To jednak przestaje być takie oczywiste, gdy proces A jest bardzo skomplikowany. W istocie możemy sobie wyobrazić, że A jest programem głównym, a B jest podprogramem odpowiedzialnym za wyprowadzanie danych. Idea współprogramów wpasowuje się gdzieś pomiędzy te dwa skrajne punkty widzenia i najłatwiej pogodzić się z nią wtedy, gdy zarówno A, jak i B są skomplikowanymi procesami wywołującymi się wzajemnie w wielu miejscach. Trudno znaleźć krótki i dobry przykład zastosowania współprogramów. Przykłady przekonujące są raczej zbyt obszerne by je cytować.

By przyjrzeć się zastosowaniom współprogramów, skupmy się na następującym nietrywialnym przykładzie. Mamy napisać program, który tłumaczy jeden kod na inny. Kod wejściowy jest ciągiem znaków alfanumerycznych zakończonych kropką, na przykład:

A2B5E3426FG0ZYW3210PQ89R. (2)

Te dane są wydziurkowane na kartach, „białe” kolumny ignorujemy. Dane wejściowe czytamy następująco, od lewej do prawej: jeśli znak jest cyfrą n spośród $0, 1, \dots, 9$, to znaczy, że następny znak należy powtórzyć $(n+1)$ razy niezależnie od tego, czy jest cyfra, czy nie. Znak nie będący cyfrą oznacza po prostu samego siebie. Na wyjściu programu powinien pojawić się ciąg zbudowany zgodnie z powyższymi zasadami. Ciąg wejściowy jest zakończony kropką, kropkę powinien też kończyć się ciąg wyjściowy. Ponadto ciąg wyjściowy ma być podzielony na grupy po trzy znaki (z wyjątkiem być może grupy ostatniej). Na przykład ciąg (2) powinien zostać przetłumaczony na

ABB BEE EEE E44 446 66F GZY W22 220 OPQ 999 999 999 R. (3)

Zauważmy, że 3426F nie oznacza 3427 powtórzeń litery F, lecz „4 czwórki, 3 szóstki, jedno F”. Jeżeli ciąg wejściowy jest postaci „1.”, poprawnym ciągiem wyjściowym jest po prostu „..”, a nie „...”, ponieważ pierwsza napotkana kropka oznacza zakończenie ciągu. Program powinien dziurkować ciąg wyjściowy na kartach, po szesnaście 3-znakowych grup na każdej karcie (być może z wyjątkiem karty ostatniej).

By zaprogramować opisane tłumaczenie, napiszemy dwa współprogramy i podprogram. Podprogram **NEXTCHAR** znajduje pierwszy niebiały znak na wejściu i umieszcza go w rA:

01	*	PODPROGRAM WCZYTUJĄCY ZNAK	
02	READER	EQU 16	Numer jednostki czytnika kart.
03	INPUT	ORIG *+16	Miejsce na karty wejściowe.
04	NEXTCHAR	STJ 9F	Wejście do podprogramu.
05		JXNZ 3F	Początkowo rX = 0.
06	1H	J6N 2F	Początkowo rI6 = 0.
07		IN INPUT(READER)	Wczytaj następną kartę.
08		JBUS *(READER)	Czekaj na zakończenie.
09		ENN6 16	Niech rI6 wskazuje pierwsze słowo.
10	2H	LDX INPUT+16,6	Pobierz następne słowo.
11		INC6 1	Przesuń wskaźnik.
12	3H	ENTA 0	
13		SLAX 1	Następny znak → rA.
14	9H	JANZ *	Pomiń spacje.
15		JMP NEXTCHAR+1	■

Powyższy podprogram ma następującą specyfikację:

Sekwencja wywołania: **JMP NEXTCHAR**.

Warunki wejściowe: rX = znaki do wykorzystania; rI6 wskazuje na następne słowo lub rI6 = 0 oznacza konieczność wczytania następnej karty.

Warunki wyjściowe: rA = następny niebiały znak z wejścia; rX i rI6 są ustawiane dla następnego wywołania **NEXTCHAR**.

Nasz pierwszy współprogram o nazwie **IN** znajduje znak kodu wejściowego z uwzględnieniem powtarzania. Pierwsza aktywacja współprogramu odbywa się w lokacji **IN1**:

16	*	PIERWSZY WSPÓŁPROGRAM	
17	2H	INCA 30	Nie cyfra.
18		JMP OUT	Wyślij do współprogramu OUT.
19	IN1	JMP NEXTCHAR	Pobierz znak.
20		DECA 30	
21		JAN 2B	Czy to litera?
22		CMPA =10=	
23		JGE 2B	Czy to znak specjalny?
24		STA *+1(0:2)	Cyfra n.
25		ENT5 *	rI5 ← n.
26		JMP NEXTCHAR	Pobierz znak.
27		JMP OUT	Wyślij do współprogramu OUT.
28		DEC5 1	Zmniejsz n o 1.
29		J5NN *-2	Powtórz, jeśli trzeba.
30		JMP IN1	Rozpocznij nowy cykl. ■

(Przypominamy, że w kodzie znakowym komputera MIX cyfry 0–9 mają kody 30–39). Powyższy współprogram ma następującą specyfikację:

Sekwencja wywołania: JMP IN.

Warunki wyjściowe

(po skoku do OUT): rA = następny znak z wejścia z uwzględnieniem powtarzania; rI4 niezmieniony w stosunku do wartości wejściowej.

Warunki wejściowe

(przy powrocie): wartości rA, rX, rI5, rI6 takie, jak w chwili ostatniego wyjścia.

Drugi współprogram o nazwie OUT formatuje kod w 3-literowe grupy i dziurkuje karty. Pierwsza aktywacja odbywa się w lokacji OUT1:

31	* DRUGI WSPÓŁPROGRAM		
32	ALF		Stała (do dziurkowania spacji).
33	OUTPUT ORIG *+16		Bufor na odpowiedzi.
34	PUNCH EQU 17		Numer jednostki dziurkarki.
35	OUT1 ENT4 -16		Rozpocznij nową kartę.
36	ENT1 OUTPUT		
37	MOVE -1,1(16)		Zapisz spacje w obszarze wyjściowym.
38	1H JMP IN		Pobierz przetłumaczony znak.
39	STA OUTPUT+16,4(1:1)		Zachowaj w (1:1).
40	CMPA PERIOD		Czy to „.”?
41	JE 9F		
42	JMP IN		Jeśli nie, pobierz następny znak.
43	STA OUTPUT+16,4(2:2)		Zachowaj w (2:2).
44	CMPA PERIOD		Czy to „.”?
45	JE 9F		
46	JMP IN		Jeśli nie, pobierz następny znak.
47	STA OUTPUT+16,4(3:3)		Zachowaj w (3:3).
48	CMPA PERIOD		Czy to „.”?
49	JE 9F		
50	INC4 1		Następne słowo w buforze wyjściowym.
51	J4N 1B		Czy to koniec karty?
52	9H OUT OUTPUT(PUNCH)		Jeśli tak, dziurkuj.
53	JBUS *(PUNCH)		Czekaj na zakończenie.
54	JNE OUT1		Wróć po więcej, jeśli nie było „.”.
55	HLT		
56	PERIOD ALF	■

Powyższy współprogram ma następującą specyfikację:

Sekwencja wywołania: JMP OUT.

Warunki wyjściowe

(po skoku do IN): rA, rX, rI5, rI6 niezmienione w stosunku do wartości wejściowych; być może zmieniona zawartość rI1; po przedni znak wyprowadzony na wyjście.

Warunki wejściowe

(przy powrocie): rA = następny znak z wejścia z uwzględnieniem powtarzania; zawartość rI4 taka jak w momencie ostatniego wyjścia.

Musimy jeszcze połączyć współprogramy (zobacz (1)) i zadbać o poprawne zainicjowanie. Inicjowanie współprogramów ma w sobie coś z żonglerki, ale w istocie nie jest trudne.

```

57 * POŁĄCZENIE I INICJOWANIE
58 START ENT6 0      Inicjowanie rI6 dla NEXTCHAR.
59          ENTX 0      Inicjowanie rX dla NEXTCHAR.
60          JMP OUT1    Uruchom OUT (zobacz ćwiczenie 2).
61 OUT    STJ INX     Aktywowanie współprogramów.
62 OUTX   JMP OUT1
63 IN     STJ OUTX
64 INX   JMP IN1
65 END    START    ■

```

Na tym kończy się kod programu. Czytelnik powinien uważnie go przestudiować, zwracając uwagę na fakt, że każdy współprogram można napisać oddziennie, tak jakby pozostałe współprogramy były podprogramami.

Warunki wejściowe i wyjściowe współprogramów IN i OUT idealnie do siebie pasują. Nie zawsze jednak będziemy mieć tyle szczęścia. Kod obsługujący aktywowanie współprogramów trzeba często rozszerzać o instrukcje zachowywania i odtwarzania zawartości rejestrów. Na przykład, jeśli współprogram OUT niszczyci zawartość rA, to kod aktywujący współprogramy wyglądałby następująco:

```

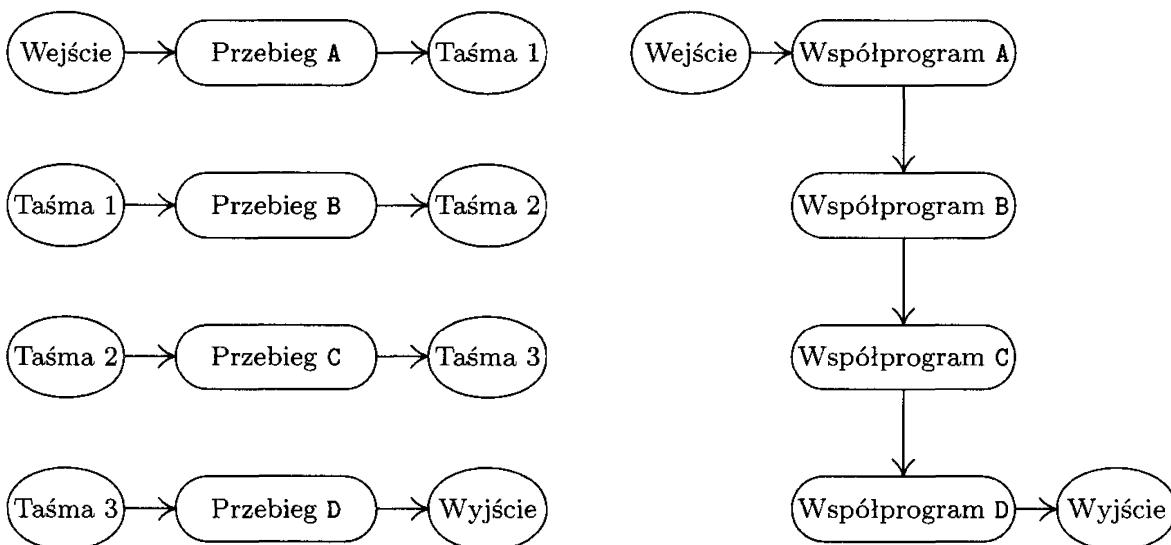
OUT  STJ  INX
     STA  HOLDA Zachowaj A przy wyjściu z IN.
OUTX JMP  OUT1
IN   STJ  OUTX
     LDA  HOLDA Przywróć A przy wyjściu z OUT.
INX  JMP  IN1    ■

```

(4)

Istnieje istotny związek między współprogramami a *algorytmami wieloprzebiegowymi* (*multiple-pass*). Na przykład opisany powyżej proces tłumaczenia można zrealizować w dwóch przebiegach: najpierw dla całego wejścia stosujemy współprogram IN, a wyniki zapisujemy na taśmie magnetycznej. Następnie przewijamy taśmę i uruchamiamy współprogram OUT, który dzieli znaki z taśmy na 3-znakowe grupy. Taki proces nazywamy „dwuprzebiegowym”. (Intuicyjnie „przebieg” oznacza przeczytanie całego wejścia. Ta definicja nie jest precyzyjna i dla wielu algorytmów nie jest jasne, ile przebiegów wykonują; ale tak czy owak intuicyjne pojęcie „przebiegu” jest przydatne).

Na rysunku 22(a) jest pokazany 4-przebiegowy proces. Często okazuje się, że ten sam proces można wykonać w jednym przebiegu, jak widać na rysunku 22(b), gdy przebiegi A, B, C, D zastąpimy współprogramami A, B, C, D. Współprogram A wykona skok do B wtedy, gdy przebieg A zapisałby daną wyjściową na taśmie 1. Współprogram B wykona skok do A, gdy przebieg B przeczytałby daną wejściową z taśmy 1. Współprogram B wykona skok do C, gdy przebieg B zapisałby daną wyjściową na taśmie 2 itd. Użytkownicy systemu UNIX® skojarzą to zapewne z „potokiem” „PrzebiegA | PrzebiegB | PrzebiegC | PrzebiegD”. Programy realizujące przebiegi B, C i D są czasem nazywane „filtrami”.



Rys. 22. Przebiegi: (a) algorytm 4-przebiegowy (b) algorytm 1-przebiegowy.

Odwrotnie, proces realizowany przez n współprogramów można zazwyczaj przekształcić w proces n -przebiegowy. Z uwagi na ten związek warto porównać algorytmy jedno- i wieloprzebiegowe.

a) *Różnica psychologiczna.* Algorytm wieloprzebiegowy zazwyczaj łatwiej wyobrazić i zrozumieć niż algorytm jednoprzebiegowy realizujący to samo zadanie. Łatwiej rozbić proces na szereg małych kroczków wykonywanych w ustalonej kolejności, niż objąć umysłem wiele przekształceń zachodzących równocześnie.

Jeśli ponadto zmagamy się z bardzo dużym problemem i w produkcji programu uczestniczy wiele osób, stosując algorytm wieloprzebiegowy, można w naturalny sposób podzielić pracę między członków zespołu.

Powyzsze zalety algorytmu wieloprzebiegowego przenoszą się na współprogramy, ponieważ każdy współprogram można napisać oddzielnie, a dopiero kod obsługujący aktywację współprogramów decyduje o tym, że wieloprzebiegowy algorytm staje się procesem jednoprzebiegowym.

b) *Różnica w czasie działania.* W algorytmie jednoprzebiegowym nie tracimy czasu na spakowanie, zapisanie, odczytanie i rozpakowanie tymczasowych zbiorów danych, które przekazywane są między przebiegami (na przykład informacja na taśmach na rysunku 22). Z tego powodu algorytm jednoprzebiegowy będzie szybszy.

c) *Różnica w zajętości pamięci.* Algorytm jednoprzebiegowy wymaga, by wszystkie programy przechowywać w pamięci jednocześnie, podczas gdy w algorytmie wieloprzebiegowym musimy w każdym przebiegu przechowywać w pamięci tylko jeden program. To wymaganie może wpływać na szybkość działania nawet bardziej niż powody wymienione w punkcie (b). Na przykład wiele komputerów dysponuje ograniczoną ilością „szybkiej pamięci” i większą ilością pamięci wolniejszej. Jeżeli każdy przebieg zajmuje prawie całą szybką pamięć, to algorytm wieloprzebiegowy będzie znacznie szybszy niż jednoprzebiegowe rozwiązanie korzystające ze współprogramów. (Zastosowanie współprogramów spowoduje

zapewne umieszczenie większej części programu w pamięci wolnej lub częstą wymianę fragmentów programu znajdującego się w pamięci szybkiej).

Czasami trzeba zaprojektować algorytm na kilka różnych komputerów jednocześnie. Może się zdarzyć, że te komputery dysponują pamięcią różnych rozmiarów. W takich przypadkach można napisać program za pomocą współprogramów i odłożyć decyzję o liczbie przebiegów na później. Procedura ładowająca wczyta tyle współprogramów, na ile pozwala rozmiar pamięci, doda kod obsługujący ich wywołania, a jeśli wszystkie współprogramy nie zmieszczą się w pamięci, to w miejsce brakujących elementów doda podprogramy wejścia-wyjścia.

Choć związek między współprogramami i przebiegami jest istotny, nie powinniśmy zapominać, że nie zawsze współprogramy da się przerobić na algorytm wieloprzebiegowy. Jeśli nie tylko współprogram **B** otrzymuje od **A** informacje, ale przepływ danych następuje również w drugą stronę (jak w przytoczonym przykładzie gry w szachy), to ciągu operacji współprogramów nie można zastąpić po prostu wykonaniem przebiegu **A**, a potem przebiegu **B**.

Odwrotnie, jasne jest, że niektórych algorytmów wieloprzebiegowych nie da się zastąpić współprogramami. Pewne algorytmy są z natury wieloprzebiegowe: drugi przebieg może potrzebować informacji zebranych w pierwszym przebiegu (jak na przykład liczba wystąpień konkretnego słowa w danych wejściowych). Jest nawet stary dowcip na ten temat:

Starsza pani w autobusie. „Chłopczyku, nie wiesz czasem, ile przystanków do Grójeckiej?”

Chłopczyk. „Jak pani zobaczy, że wysiadam, to niech pani wysiądzie dwa przystanki wcześniej.”

(Dowcip polega na tym, że chłopczyk podał algorytm dwuprzebiegowy).

To tyle, jeśli chodzi o algorytmy wieloprzebiegowe. Kolejne przykłady współprogramów pojawią się w tej książce jeszcze wielokrotnie, na przykład jako część schematów buforowania w punkcie 1.4.4. Współprogramy grają też istotną rolę w dyskretnych systemach symulacji, zobacz punkt 2.2.5. Pomyśl *współprogramów wielokrotnych* omawiamy w rozdziale 8, a jego ciekawe zastosowanie w rozdziale 10.

ĆWICZENIA

1. [10] Wyjaśnij, dlaczego autorom podręczników trudno jest znaleźć krótkie i proste przykłady zastosowania współprogramów.
- ▶ 2. [20] Program w tekście w pierwszej kolejności uruchamia współprogram OUT. Co by się stało, gdybyśmy zaczęli od IN, czyli zmienili wiersz 60 z „JMP OUT1” na „JMP IN1”?
3. [20] Prawda czy fałsz: trzy instrukcje „CMPA PERIOD” w OUT można pominąć, a program mimo to będzie działał. (Uważaj).
4. [20] Jak wyglądałby kod aktywujący współprogramy analogiczny do (1) na znanych Ci prawdziwych komputerach?

5. [15] Przypuśćmy, że współprogramy IN i OUT wymagają, by między wyjściem a ponownym aktywowaniem współprogramu zawartość rejestru A pozostała niezmieniona. Innymi słowy, tam gdzie we współprogramie OUT pojawia się rozkaz „JMP IN”, zawartość rejestru A w chwili wznowienia wykonania współprogramu od następnego wiersza ma być taka, jak w chwili wykonania skoku. Takie same wymagania mamy w stosunku do „JMP OUT” we współprogramie IN. W jaki sposób należy aktywować te współprogramy? (Porównaj z (4)).

- ▶ 6. [22] Jak powinien wyglądać kod aktywujący współprogramy analogiczny do (1), jeśli współprogramy są trzy? Współprogramy nazywają się A, B i C, a każdy z nich może aktywować dowolny z pozostałych dwóch. (Aktywowany współprogram jest wznowiany od miejsca, w którym został ostatnio wstrzymany).
- ▶ 7. [30] Napisz program na komputer MIX, który odwraca tłumaczenie realizowane przez program omówiony w tekście. Twój program powinien tłumaczyć dane z karty wydziurkowanej jak (3) na dane wydziurkowane jak (2). Napis wyjściowy powinien być jak najkrótszy, więc na przykład zero przed Z w (2) nie powinno zostać wygenerowane.

1.4.3. Interpretery

W niniejszym punkcie omówimy popularny typ programu komputerowego – *interpreter*. Interpreter jest programem wykonującym instrukcje innego programu, napisanego zazwyczaj w jakimś języku niskiego poziomu. Przez język niskiego poziomu rozumiemy taki język, w którym instrukcje zawierają kody operacji, adresy itp. (Ta definicja, jak większość współczesnych definicji dotyczących terminologii komputerowej, nie jest precyzyjna i precyzyjną być nie powinna. Nie można wytyczyć dokładnej granicy oddzielającej interpretery od innych programów).

Z historycznego punktu widzenia pierwsze interpretery budowano dla języków niskiego poziomu zaprojektowanych do tworzenia prostych programów. Takie języki były łatwiejsze w użyciu niż czysty język maszynowy. Powstanie symbolicznych języków programowania po krótkim czasie ugasiło zapotrzebowanie na interpretery tego typu, ale w żadnym razie nie był to dla interpreterów wyrok skazujący. Wprost przeciwnie, różne interpretery zaczęto wykorzystywać coraz intensywniej. Efektywne wykorzystanie interpreterów można uznać za jedną z istotnych cech współczesnego programowania. Nowe zastosowania interpreterów wynikają głównie z następujących powodów:

- a) język niskiego poziomu pozwala wyrazić skomplikowany ciąg decyzji i akcji w zwartej, wydajnej postaci,
- b) za pomocą takiej reprezentacji można bardzo wygodnie przekazywać informacje między przebiegami procesu wieloprzebiegowego.

W takich przypadkach na potrzeby konkretnego programu tworzy się zazwyczaj specjalny język niskiego poziomu, a programy w tym języku często są generowane wyłącznie przez inne programy. (Dzisiaj programiści są również projektantami maszyn, ponieważ tworzą nie tylko interpretery, ale także specyfikacje *maszyn wirtualnych*, których język ma być interpretowany).

Technika interpretacji ma dodatkowe zalety polegające na uniezależnieniu programu od sprzętu. Przeniesienie programu na inny komputer wymaga prze-

niesienia wyłącznie interpretera. Co więcej, w system interpretujący można wbudować mechanizmy znacznie ułatwiające usuwanie błędów i uruchamianie programu.

Przykłady interpreterów rodzaju (a) często pojawiają się w „Sztuce programowania”, na przykład interpreter w rozdziale 8 lub maszyna analizująca składnię w rozdziale 10. Po interpretery sięgamy zazwyczaj wtedy, gdy w problemie pojawia się wiele szczegółowych wariantów, ale nie widać żadnych ogólnych schematów.

Załóżmy, że piszemy kompilator. Kompilator między innymi ma generować ciąg rozkazów maszynowych realizujących dodawanie dwóch argumentów. Argumenty mogą należeć do jednej z dziesięciu klas (stałe, zmienne proste, lokacje tymczasowe, zmienne indeksowane, zawartość akumulatora lub rejestru indeksowego, wartość stałopozycyjna lub zmiennopozycyjna itd.), mamy zatem 100 różnych przypadków. Potrzeba długiego programu, żeby obsłużyć poprawnie każdy z nich. Rozwiążanie interpretacyjne polega na wymyśleniu języka, którego „instrukcje” będą zajmowały po jednym bajcie. W tym języku piszemy 100 „programów”, z których każdy mieści się w jednym słowie. Pomyśl polega na wybieraniu odpowiedniej pozycji z tablicy i wykonywaniu znajdującego się tam programu. Taka metoda jest prosta i wydajna.

Przykład interpretera typu (b) pojawia się w artykule „Computer-Drawn Flowcharts” autorstwa D. E. Knutha, *CACM* 6 (1963), 555–563. W programie wieloprzebiegowym wcześniejsze przebiegi muszą przekazywać informacje do przebiegów późniejszych. Najwygodniej zapisywać je w specjalnym języku niskiego poziomu, jako zestaw rozkazów dla kolejnych przebiegów. Przebiegi „końcowe” nie są zatem niczym innym, jak specjalizowanymi interpreterami, a przebiegi „początkowe” są specjalizowanymi „kompilatorami”. Taką filozofię konstruowania programów wieloprzebiegowych można scharakteryzować następująco: wcześniejszy przebieg zamiast przedstawiać swojemu następcy masę faktów, *instruuje* go co do dalszego przetwarzania.

Inny przykład interpretera typu (b) pojawia się w związku z kompilatorami specjalizowanych języków. Jeżeli język ma wiele cech, które trudno zrealizować na maszynie, nie posługując się podprogramem, wynikowy kod będzie bardzo długim ciągiem wywołań podprogramów. Przykładem takiej sytuacji jest język, który skupia się na operacjach arytmetycznych różnej precyzji. W takim przypadku kod wynikowy byłby istotnie krótszy, gdyby zapisano go w języku interpretowanym; zobacz na przykład *ALGOL 60 Implementation*, B. Randell, L. J. Russell (New York: Academic Press, 1964). W książce tej jest opisany kompilator tłumaczący ALGOL 60 na język interpretowany oraz interpreter tego języka; zobacz także Arthur Evans, Jr., „An ALGOL 60 Compiler”, *Ann. Rev. Auto. Programming* 4 (1964), 87–124. Książka zawiera przykłady interpreterów wykorzystywanych *wewnątrz* kompilatora. Z powstaniem maszyn o rozszerzalnej liście rozkazów oraz specjalizowanych zintegrowanych układów scalonych podejście interpretacyjne stało się jeszcze bardziej atrakcyjne.

Strony książki, którą trzymasz w ręku, wygenerowano za pomocą programu *TeX*. Program *TeX* przetłumaczył plik z tekstem niniejszego rozdziału na

interpretowany język DVI, zaprojektowany przez D. R. Fuchsa w 1979 roku. [Zobacz D. E. Knuth, *TeX: The Program* (Reading, Mass.: Addison–Wesley, 1986), część 31]. Plik DVI wygenerowany przez program *TeX* został później przekształcony przez interpreter *dvips* autorstwa T. G. Rokickiego i przekształcony na plik zawierający instrukcje kolejnego interpretowanego języka o nazwie PostScript[®] [Adobe Systems Inc., *PostScript Language Reference Manual*, 2nd edition (Reading, Mass.: Addison–Wesley, 1990)]. Z tego pliku za pomocą interpretera języka PostScript wyprodukowano matryce drukarskie. Ten trójprzebiegowy proces jest przykładem wykorzystania interpreterów typu (b). *TeX* zawiera niewielki interpreter typu (a), za pomocą którego przetwarza informacje dotyczące tzw. ligatur i kerningu [*TeX: The Program*, §545].

Na program w języku interpretowanym można patrzeć jak na ciąg wywołań podprogramów. Taki program można w zasadzie rozwinąć w długi ciąg wywołań, a długie ciągi wywołań można zamienić na kod interpretowany. Zaletą metod interpretacyjnych jest zwarta reprezentacja, niezależność od maszyny, wygodniejsze uruchamianie i usuwanie błędów. Zazwyczaj można tak napisać interpreter, że czas spędzony na wykonaniu kodu samego interpretera i obsłudze wywoływania odpowiednich podprogramów jest zaniedbywalny.

1.4.3.1. Symulator maszyny MIX. Gdy język wejściowy interpretera jest językiem maszynowym innego komputera, interpreter nazywamy *symulatorem* (albo *emulatorem*).

Autor uważa, że zdecydowanie zbyt dużo pracy programistów poświęcono na pisanie symulatorów i zdecydowanie zbyt dużo czasu komputerów przeznaczono na ich wykonywanie. Powody są proste: instalujemy nowy komputer, a chcemy używać programów ze starej maszyny (zamiast przenosić posiadane programy na nową platformę). To jednak zazwyczaj kosztuje więcej i daje gorsze wyniki niż zlecenie przepisania programów specjalnej grupie programistów. Autor brał kiedyś udział w takim projekcie. Przy przenoszeniu oprogramowania odkryto poważny błąd w programie używanym od lat. Nowy program działał pięć razy szybciej od starego, a „przy okazji” dawał poprawne wyniki! (Nie należy jednak potępiać wszystkich symulatorów. Często producentowi komputerów opłaca się symulować nową maszynę, zanim jeszcze zostanie zbudowana. Dzięki temu oprogramowanie na nową platformę można tworzyć przed premierą produktu. Ale to jest bardzo wyjątkowy przypadek). Skrajnym przypadkiem mało wydajnego wykorzystania symulatorów jest autentyczna historia maszyny *A* symulującej maszynę *B*, na której uruchamiano program symulujący maszynę *C*! W ten prosty sposób można z dużego i drogiego komputera zrobić maszynę kiepską i mało wydajną.

Dlaczego zatem symulator wyszczerza zębiska w tej książce? Są dwa powody:

a) Symulator opisany poniżej jest przykładem typowego interpretera wykorzystującego wiele standardowych metod. Służy również za przykład zastosowania podprogramów w programie średniej wielkości.

b) Opiszemy symulator komputera **MIX**, zapisany w języku komputera **MIX**. To ułatwi pisanie symulatorów komputera **MIX** na większość komputerów. W ko-

dzie programu z premedytacją unikamy wykorzystywania cech specyficznych dla komputera **MIX**. Symulator komputera **MIX** może posłużyć za pomoc naukową przy czytaniu tej książki (a być może innych także).

Symulatorów komputerów opisanych w tym rozdziale nie należy mylić z *symulatorami systemów dyskretnych*. Symulatory systemów dyskretnych są ważnymi programami, omawianymi w punkcie 2.2.5.

Zabierzmy się zatem za napisanie symulatora komputera **MIX**. Na wejściu symulatora podajemy ciąg rozkazów komputera **MIX** oraz dane dla symulowanego programu, zapisane w lokacjach 0000 – 3499. Chcemy naśladować zachowanie komputera **MIX**, udając, że to sam **MIX** wykonuje te rozkazy. Zakodujemy zatem specyfikacje podane w punkcie 1.3.1. Symulator będzie przechowywał m.in. wartość **AREG**, odpowiadającą wartości bezwzględnej w symulowanym rejestrze **A**. Inna zmienna, **SIGNA**, będzie przechowywała znak **rA**. (W dalszej części tego punktu przez „wartość” w symulowanym komputerze **MIX** będziemy rozumieć „wartość bezwzględną”, jeżeli z kontekstu wyraźnie nie wynika inaczej). Zmiennej **CLOCK** użyjemy do śledzenia, ile jednostek symulowanego czasu upłynęło od rozpoczęcia symulacji wykonania programu.

Numeracja rozkazów **LDA**, **LD1**, ..., **LDX** i podobnych sugeruje, by przechowywać zawartość symulowanych rejestrów w kolejnych lokacjach:

AREG, **I1REG**, **I2REG**, **I3REG**, **I4REG**, **I5REG**, **I6REG**, **XREG**, **JREG**, **ZERO**.

ZERO jest „rejestrem”, który cały czas jest wypełniony zerami. Pozycje rejestrów **JREG** i **ZERO** wynikają z kodów rozkazów **STJ** i **STZ**.

Trzymając się filozofii niekorzystania w symulatorze ze specyficznych cech komputera **MIX**, będziemy traktować znak jak niezależną część rejestrów. Na przykład na wielu komputerach nie można reprezentować wartości „minus zero”, podczas gdy na maszynie **MIX** nie ma problemu. Znak będziemy przeto traktować specjalnie. Lokacje **AREG**, **I1REG**, ..., **ZERO** będą zawierać wartości bezwzględne odpowiednich rejestrów. W innych lokacjach, **SIGNA**, **SIGN1**, ..., **SIGNZ**, będziemy przechowywać wartości +1 lub -1, zależnie od tego, czy znak zawartości symulowanego rejestrów jest dodatni czy ujemny.

Zasadniczo interpreter ma centralną część sterującą, która jest wywoływaną między interpretowaniem kolejnych rozkazów. W naszym przypadku po wykonaniu każdego symulowanego rozkazu sterowanie jest przekazywane do lokacji **CYCLE**.

Podprogram sterujący odpowiada za akcje wspólne dla wszystkich rozkazów: rozbija rozkazy na części, umieszcza te części w odpowiednich miejscach do dalszego wykorzystania. Poniższy program wyznacza następujące wartości:

- rI6 = lokacja następnego rozkazu;
- rI5 = M (adres bieżącego rozkazu plus indeksowanie);
- rI4 = kod operacji bieżącego rozkazu;
- rI3 = pole F bieżącego rozkazu;
- INST = bieżący rozkaz.

Program M.

001	*	SYMULATOR KOMPUTERA MIX	
002		ORIG 3500	Symulowana pamięć: od lokacji 0000.
003	BEGIN	STZ TIME(0:2)	
004		STZ OVTOG	OVTOG – symulowany znacznik przepełnienia.
005		STZ COMPI	COMPI, ±1 lub 0, wskaźnik porównania.
006		ENT6 0	Pobierz pierwszy rozkaz z lokacji zero.
007	CYCLE	LDA CLOCK	Początek części sterującej.
008	TIME	INCA 0	Czas wykonania poprzedniego rozkazu (zobacz wiersz 033).
009		STA CLOCK	rA ← symulowany rozkaz.
010		LDA 0,6	
011		STA INST	
012		INC6 1	Zwiększa licznik lokacji.
013		LDX INST(1:2)	Wyznacz wartość bezwzględną adresu.
014		SLAX 5	Dołącz znak do adresu.
015		STA M	
016		LD2 INST(3:3)	Zbadaj pole indeksu.
017		J2Z 1F	Czy zero?
018		DEC2 6	
019		J2P INDEXERROR	Czy niedozwolony indeks?
020		LDA SIGN6,2	Pobierz znak rejestru indeksowego.
021		LDX I6REG,2	Pobierz wartość rejestru indeksowego.
022		SLAX 5	Dołącz znak.
023		ADD M	Dodaj ze znakiem (indeksowanie).
024		CMPA ZERO(1:3)	Czy wynik zbyt duży?
025		JNE ADDRERROR	Jeśli tak, to sygnalizuj błąd.
026		STA M	W przeciwnym razie mamy adres.
027	1H	LD3 INST(4:4)	rI3 ← pole F.
028		LD5 M	rI5 ← M.
029		LD4 INST(5:5)	rI4 ← pole C.
030		DEC4 63	
031		J4P OPERROR	Czy kod operacji ≥ 64 ?
032		LDA OPTABLE,4(4:4)	Pobierz czas wykonania z tablicy.
033		STA TIME(0:2)	
034		LD2 OPTABLE,4(0:2)	Pobierz adres właściwego podprogramu.
035		JNOV 0,2	Skocz do tego podprogramu.
036		JMP 0,2	(Zerowanie znacznika przepełnienia.) ■

Zwracamy uwagę Czytelnika na wiersze 034–036: „tablica przełączająca” zawierająca opis 64 operatorów jest częścią symulatora umożliwiającą szybki skok do podprogramu symulującego odpowiedni rozkaz. Jest to istotna metoda pozwalająca na przyspieszenie wykonania programu (zobacz ćwiczenie 1.3.2–9).

Tablica przełączająca OPTABLE o rozmiarze 64 słów zawiera także czas wykonania operacji. Oto jej zawartość:

037	NOP	CYCLE(1)	Tablica kodów operacji;
038	ADD	ADD(2)	znaczenie pozycji
039	SUB	SUB(2)	„OP podprogram(czas)”
040	MUL	MUL(10)	

```

041      DIV  DIV(12)
042      HLT  SPEC(1)
043      SLA  SHIFT(2)
044      MOVE MOVE(1)
045      LDA  LOAD(2)
046      LD1  LOAD,1(2)
...
051      LD6  LOAD,1(2)
052      LDX  LOAD(2)
053      LDAN LOADN(2)
054      LD1N LOADN,1(2)
...
060      LDXN LOADN(2)
061      STA  STORE(2)
...
069      STJ  STORE(2)
070      STZ  STORE(2)
071      JBUS JBUS(1)
072      IOC  IOC(1)
073      IN   IN(1)
074      OUT  OUT(1)
075      JRED JRED(1)
076      JMP  JUMP(1)
077      JAP  REGJUMP(1)
...
084      JXP  REGJUMP(1)
085      INCA ADDROP(1)
086      INC1 ADDROP,1(1)
...
092      INCX ADDROP(1)
093      CMPA COMPARE(2)
...
100  OPTABLE CMPX COMPARE(2)  ■

```

(Pozycje opisujące operacje LDi , $LDiN$ i $INCi$ mają dodatkowo „,1”, co powoduje wstawienie niezerowej wartości do wycinka (3:3). W ten sposób oznaczamy, że po zasymulowaniu operacji należy sprawdzić, czy w rejestrze indeksowym znajduje się odpowiednia wartość, zobacz wiersze 289–290).

Następna część naszego symulatora to po prostu lista lokacji używanych do przechowywania zawartości symulowanych rejestrów:

```

101  AREG    CON  0  Wartość w rejestrze A.
102  I1REG   CON  0  Wartość w rejestrach indeksowych.
...
107  I6REG   CON  0
108  XREG    CON  0  Wartość w rejestrze X.
109  JREG    CON  0  Wartość w rejestrze J.
110  ZERO    CON  0  Stała zero, dla „STZ”.
111  SIGNA   CON  1  Znak w rejestrze A.

```

```

112 SIGN1 CON 1 Znaki w rejestrach indeksowych.
...
117 SIGN6 CON 1
118 SIGNX CON 1 Znak w rejestrze X.
119 SIGNJ CON 1 Znak w rejestrze J.
120 SIGNZ CON 1 Znak ładowany przez „STZ”.
121 INST CON 0 Symulowany rozkaz.
122 COMPI CON 0 Wskaźnik porównania.
123 OVTOG CON 0 Znacznik przepełnienia.
124 CLOCK CON 0 Symulowany czas wykonania. ■

```

Skupimy się teraz na trzech podprogramach symulatora. Na początek podprogram **MEMORY**:

Sekwencja wywołania: **JMP MEMORY**.

Warunki wejściowe: rI5 = poprawny adres pamięci (w przeciwnym razie podprogram wykona skok do **MEMERROR**).

Warunki wyjściowe: rX = znak słowa w lokacji rI5; rA = wartość słowa w lokacji rI5.

```

125 * PODPROGRAMY
126 MEMORY STJ 9F          Podprogram pobierania wartości z pamięci.
127      J5N MEMERROR
128      CMP5 =BEGIN=       Symulowaną pamięć przechowujemy
129      JGE MEMERROR      w lokacjach od 0000 do BEGIN - 1.
130      LDX 0,5
131      ENTA 1
132      SRAX 5            rX ← znak słowa.
133      LDA 0,5(1:5)      rA ← wartość słowa.
134 9H      JMP *          Wyjście. ■

```

Podprogram **FCHECK** przetwarza specyfikację pól wycinka, upewniając się, że jest ona postaci $8L + R$, gdzie $L \leq R \leq 5$.

Sekwencja wywołania: **JMP FCHECK**.

Warunki wejściowe: rI3 = poprawna specyfikacja wycinka (w przeciwnym razie zostanie wykonany skok do **FERROR**).

Warunki wyjściowe: rA = rI1 = L, rX = R.

```

135 FCHECK STJ 9F          Podprogram sprawdzania wycinka.
136      ENTA 0
137      ENTX 0,3          rAX ← specyfikacja wycinka.
138      DIV =8=           rA ← L, rX ← R.
139      CMPX =5=           Is R > 5?
140      JG FERROR
141      STX R
142      STA L
143      LD1 L              rI1 ← L.
144      CMPA R
145 9H      JLE *          Wyjdź, chyba że L > R.
146      JMP FERROR ■

```

Ostatni podprogram, GETV, wyznacza wartość V (tj. odpowiedni wycinek pól lokacji M) wykorzystywana przez operacje maszyny MIX zgodnie z definicją w punkcie 1.3.1.

Sekwencja wywołania: JMP GETV.

Warunki wejściowe: rI5 = poprawny adres w pamięci; rI3 = poprawna specyfikacja wycinka (w przypadku specyfikacji niepoprawnej zostanie zgłoszony błąd, jak w poprzednim podprogramie).

Warunki wyjściowe: rA = wartość V; rX = znak V; rI1 = L; rI2 = -R.

Drugie wejście: JMP GETAV, wykorzystywane wyłącznie w operacjach porównania do wyznaczenia pól wycinka rejestru.

147	GETAV	STJ	9F	Wejście specjalne, zobacz wiersz 300.
148		JMP	1F	
149	GETV	STJ	9F	Podprogram wyznaczania V.
150		JMP	FCHECK	Sprawdź specyfikację wycinka, rI1 ← L.
151		JMP	MEMORY	rA ← wartość z pamięci, rX ← znak z pamięci.
152	1H	J1Z	2F	Czy wycinek obejmuje znak?
153		ENTX	1	Jeśli nie, to ustaw plus.
154		SLA	-1,1	Wyzeruj wszystkie bajty na lewo
155		SRA	-1,1	od wycinka.
156	2H	LD2N	R	Przesuń w prawo
157		SRA	5,2	na odpowiednią pozycję.
158	9H	JMP	*	Wyjście. ■

Dochodzimy do podprogramów dla poszczególnych operacji. Zamieszczamy je, bo bez nich program byłby niekompletny. Nie ma jednak sensu, żeby Czytelnik studiował wszystkie. Szczególnie polecamy SUB i JUMP. Warto zwrócić uwagę na to, jak podprogramy dla podobnych operacji można ze sobą połączyć, a także w jaki sposób JUMP wykorzystuje dodatkową tablicę przełączającą obsługującą różne rodzaje skoków.

159 * POSZCZEGÓLNE OPERACJE

160	ADD	JMP	GETV	Pobierz wartość z V do rA i rX.
161		ENT1	0	rI1←numer lokacji symulowanego rA.
162		JMP	INC	Skocz do podprogramu dodawania.
163	SUB	JMP	GETV	Pobierz wartość z V do rA i rX.
164		ENT1	0	Niech rI1 wskazuje rejestr A.
165		JMP	DEC	Skocz do programu odejmowania.
166	*			
167	MUL	JMP	GETV	Pobierz wartość z V do rA i rX.
168		CMPX	SIGNA	Czy znaki są takie same?
169		ENTX	1	
170		JE	*+2	Wpisz znak wynikowy do rX.
171		ENN	X 1	
172		STX	SIGNA	Wstaw do obu symulowanych rejestrów.
173		STX	SIGNX	
174		MUL	AREG	Wymnóż argumenty.
175		JMP	STOREAX	Zapisz wartości (bezwzględne).

176 *			
177 DIV	LDA	SIGNA	Ustaw znak reszty.
178	STA	SIGNX	
179	JMP	GETV	Pobierz wartość z V do rA i rX.
180	CMPX	SIGNA	Czy znaki są takie same?
181	ENTX	1	
182	JE	*+2	Wpisz znak wynikowy do rX.
183	ENN X	1	
184	STX	SIGNA	Wstaw do symulowanego rA.
185	STA	TEMP	
186	LDA	AREG	Podziel argumenty.
187	LDX	XREG	
188	DIV	TEMP	
189 STOREAX	STA	AREG	Zapisz wartości (bezwzględne).
190	STX	XREG	
191 OVCHECK	JNOV	CYCLE	Czy wystąpiło przepełnienie?
192	ENTX	1	Jeśli tak, ustaw symulowany znacznik przepełnienia.
193	STX	OVTOG	
194	JMP	CYCLE	Powróć do części sterującej.
195 *			
196 LOADN	JMP	GETV	Pobierz wartość z V do rA i rX.
197	ENT1	47,4	rI1 ← C – 16; wskazuje rejestr.
198 LOADN1	STX	TEMP	Odwróć znak.
199	LDX N	TEMP	
200	JMP	LOAD1	Zmień LOADN na LOAD.
201 LOAD	JMP	GETV	Pobierz wartość V do rA i rX.
202	ENT1	55,4	rI1 ← C – 8, wskazuje rejestr.
203 LOAD1	STA	AREG,1	Zapisz wartość.
204	STX	SIGNA,1	Zapisz znak.
205	JMP	SIZECHK	Sprawdź rozmiar wartości.
206 *			
207 STORE	JMP	FCHECK	rI1 ← L.
208	JMP	MEMORY	Pobierz zawartość lokacji.
209	J1P	1F	Czy wycinek obejmuje znak?
210	ENT1	1	Jeśli tak, to zmień L na 1
211	LDX	SIGNA+39,4	i „zapisz” znak rejestru.
212 1H	LD2N	R	rI2 ← -R.
213	SRAX	5,2	Zapisz obszar na prawo od wycinka.
214	LDA	AREG+39,4	Wstaw zawartość rejestrów do pól.
215	SLAX	5,2	
216	ENN2	0,1	rI2 ← -L.
217	SRAX	6,2	
218	LDA	0,5	Przywróć obszar na lewo od pól.
219	SRA	6,2	
220	SRAX	-1,1	Dołącz znak.
221	STX	0,5	Zapisz w pamięci.
222	JMP	CYCLE	Wróć do części sterującej.
223 *			
224 JUMP	DEC3	9	Rozkazy skoków:

225		J3P	FERROR	Czy F zbyt duże?
226		LDA	COMPI	rA ← wskaźnik porównania.
227		JMP	JTABLE, 3	Skocz do odpowiedniego podprogramu.
228	JMP	ST6	JREG	Ustaw symulowany rejestr J.
229		JMP	JSJ	
230		JMP	JOV	
231		JMP	JNOV	
232		JMP	LS	
233		JMP	EQ	
234		JMP	GR	
235		JMP	GE	
236		JMP	NE	
237	JTABLE	JMP	LE	Koniec tablicy skoków.
238	JOV	LDX	OVTOG	Czy skakać? (Czy wystąpiło przepelnienie?)
239		JMP	*+3	
240	JNOV	LDX	OVTOG	
241		DECX	1	Zaneguj znacznik przepelnienia.
242		STZ	OVTOG	Skasuj znacznik przepelnienia.
243		JXNZ	JMP	Skocz.
244		JMP	CYCLE	Nie skacz.
245	LE	JAZ	JMP	Skocz, gdy rA równe 0 lub ujemne.
246	LS	JAN	JMP	Skocz, gdy rA ujemne.
247		JMP	CYCLE	Nie skacz.
248	NE	JAN	JMP	Skocz, gdy rA ujemne lub dodatnie.
249	GR	JAP	JMP	Skocz, gdy rA dodatnie.
250		JMP	CYCLE	Nie skacz.
251	GE	JAP	JMP	Skocz, gdy rA dodatnie lub równe 0.
252	EQ	JAZ	JMP	Skocz, gdy rA równe 0.
253		JMP	CYCLE	Nie skacz.
254	JSJ	JMP	MEMORY	Sprawdź poprawność adresu pamięci.
255		ENT6	0,5	Symuluj skok.
256		JMP	CYCLE	Wróć do części sterującej.
257	*			
258	REGJUMP	LDA	AREG+23, 4	Skoki zależne od rejestrów:
259		JAZ	*+2	Czy zawartość rejestrów jest równa 0?
260		LDA	SIGNA+23, 4	Jeśli nie, ładuj znak do rA.
261		DEC3	5	
262		J3NP	JTABLE, 3	Zmień na skok warunkowy, chyba że F jest zbyt duże.
263		JMP	FERROR	
264	*			
265	ADDROP	DEC3	3	Operacje przekazania adresu:
266		J3P	FERROR	F zbyt duże?
267		ENTX	0,5	
268		JXNZ	*+2	Wyznacz znak M.
269		LDX	INST	
270		ENTA	1	
271		SRAX	5	rX ← znak M.
272		LDA	M(1:5)	rA ← wartość M.
273		ENT1	15,4	rI1 wskazuje rejestr.

274	JMP	1F , 3	Rozgałżenie na cztery.
275	JMP	INC	Zwiększ.
276	JMP	DEC	Zmniejsz.
277	JMP	LOAD1	Wpisz.
278 1H	JMP	LOADN1	Wpisz ujemne.
279 DEC	STX	TEMP	Odwroć znak.
280	LDXN	TEMP	Sprowadź DEC do INC.
281 INC	CMPX	SIGNA , 1	Podprogram dodawania:
282	JE	1F	Czy znaki są takie same?
283	SUB	AREG , 1	Nie, odejmij wartości.
284	JANP	2F	Czy trzeba zmienić znak?
285	STX	SIGNA , 1	Zmień znak rejestru.
286	JMP	2F	
287 1H	ADD	AREG , 1	Dodaj wartości.
288 2H	STA	AREG , 1(1:5)	Zapisz wartość wyniku.
289 SIZECHK	LD1	OPTABLE , 4(3:3)	Czy załadowaliśmy rejestr indeksowy?
290	J1Z	OVCHECK	
291	CMPA	ZERO(1:3)	Jeśli tak, to sprawdź, czy mieści się w dwóch bajtach.
292	JE	CYCLE	
293	JMP	SIZEERROR	
294 *			
295 COMPARE	JMP	GETV	Pobierz wartość z V do rA i rX.
296	SRAX	5	Dołącz znak.
297	STX	V	
298	LDA	XREG , 4	Pobierz wycinek odpowiedniego rejestru.
299	LDX	SIGNX , 4	
300	JMP	GETAV	
301	SRAX	5	Dołącz znak.
302	CMPX	V	Porównaj (uwaga: -0 = +0).
303	STZ	COMPI	Ustaw wskaźnik porównania na
304	JE	CYCLE	zero, plus jeden
305	ENTA	1	lub minus jeden.
306	JG	**+2	
307	ENNA	1	
308	STA	COMPI	
309	JMP	CYCLE	Wróć do części sterującej.
310 *			
311	END	BEGIN ■	

Powyższy kod uwzględnia subtelną zasadę przedstawioną w punkcie 1.3.1: rozkaz „ENTA -0” powoduje załadowanie zera do rejestru A, tak jak „ENTA -5 , 1”, gdy rI1 zawiera +5. W ogólnosci, gdy M jest równe zero, ENTA ładuje znak rozkazu, a ENNA ładuje znak przeciwny. Przygotowując pierwszą wersję punktu 1.3.1, autor przeoczył konieczność wyspecyfikowania tego drobiazgu. Takie detale wypływają zazwyczaj jedynie wtedy, gdy przystąpi się do kodowania specyfikacji.

Powyższy program, choć obszerny, wciąż jest niekompletny i to z paru powodów:

- a) Nie rozpoznaje operacji zmennopozycyjnych.

- b) Zaprogramowanie operacji o kodach 5, 6 i 7 pozostawiono jako ćwiczenie.
- c) Zaprogramowanie operacji wejścia-wyjścia pozostawiono jako ćwiczenie.
- d) Nie zadbano o ładowanie symulowanych programów (zobacz ćwiczenie 4).
- e) Nie napisano podprogramów

INDEXERROR, ADDRERROR, OPERROR, MEMERROR, FERROR, SIZEERROR

zgłaszających błędy w symulowanym kodzie.

- f) Symulator nie ma mechanizmów ułatwiających wyszukiwanie błędów i uruchamianie programu. (Dobry symulator powinien umożliwiać na przykład drukowanie zawartości symulowanych rejestrów).

ĆWICZENIA

1. [14] Przeanalizuj wszystkie wywołania podprogramu FCHECK w kodzie symulatora. Spróbuj zaproponować lepszą organizację kodu. (Zobacz krok 3 pod koniec punktu 1.4.1).
2. [20] Napisz brakujący podprogram SHIFT (kod operacji: 6).
- ▶ 3. [22] Napisz brakujący podprogram MOVE (kod operacji: 7).
4. [14] Zmodyfikuj podany w tekście program w ten sposób, żeby zaczynał pracę tak, jakby naciśnięto „przycisk START” (zobacz ćwiczenie 1.3.1–26).
- ▶ 5. [24] Wyznacz czas potrzebny na symulację rozkazów LDA oraz ENTA i porównaj go z czasem wykonania tych rozkazów wprost na maszynie MIX.
6. [28] Napisz podprogramy symulujące brakujące rozkazy wejścia-wyjścia JBUS, IOC, IN, OUT i JRED, dopuszczając operacje jedynie na urządzeniach 16 i 18. Załóż, że operacje „czytaj-kartę” i „przesuń-do-nowej-strony” zajmują po $T = 10000u$, a „drukuj-wiersz” zajmuje $T = 7500u$. (Uwaga: Doświadczenie pokazuje, że rozkaz JBUS należy symulować, traktując „JBUS *” jako przypadek szczególny. W przeciwnym razie symulator może pracować bardzo długo).
- ▶ 7. [32] Zmodyfikuj rozwiązanie poprzedniego ćwiczenia w taki sposób, żeby wykonanie rozkazów IN i OUT nie powodowało natychmiastowego wykonania operacji wejścia-wyjścia. Transmisja powinna nastąpić mniej więcej w połowie czasu wymaganego przez symulowane urządzenia. (W ten sposób wykryjemy często popełniane przez studentów błędy, polegające na niepoprawnym użyciu IN i OUT).
8. [20] Prawda czy fałsz: przy każdym wykonaniu wiersza 010 zachodzi warunek $0 \leq rI6 < \text{BEGIN}$.

***1.4.3.2. Śledzenie.** Kiedy maszyna jest symulowana na sobie samej (tak jak komputer MIX był symulowany na komputerze MIX w poprzednim rozdziale) mamy do czynienia ze specjalnym rodzajem symulatora zwanego programem *śledzącym* lub *monitorem*. Takich programów używa się czasami przy uruchamianiu i wyszukiwaniu błędów, ponieważ potrafią wydrukować instrukcje wykonywane przez symulowany program.

Program w poprzednim rozdziale był tak napisany, jakby maszynę MIX miał symulować inny komputer. W programach śledzących stosuje się nieco inne podejście. Wartości symulowanych rejestrów są z reguły przechowywane w odpowiadających im prawdziwych rejestrach, a maszynie pozwala się po prostu

wykonywać większość rozkazów. Zasadniczym wyjątkiem są rozkazy skoku oraz skoku warunkowego, których nie można wykonywać bez specjalnych modyfikacji, ponieważ sterowanie mogłoby się „wymknąć” programowi śledzącemu. Każda maszyna ma sobie tylko właściwe cechy, będące wyzwaniem dla autorów programów śledzących. W przypadku komputera MIX najciekawsze problemy wiążą się z rejestrzem J.

Poniższy program śledzący rozpoczyna działanie, gdy program główny wykonuje skok do lokacji ENTER. W rejestrze J znajduje się adres, od którego śledzenie ma się rozpoczęć, a w rejestrze X adres, na którym ma się zakończyć. Program jest ciekawy i wart uważnego przeanalizowania.

01	*	TRACE ROUTINE	
02	ENTER	STX TEST(0:2)	Ustaw lokację wyjściową.
03		STX LEAVEX(0:2)	
04		STA AREG	Zachowaj zawartość rA.
05		STJ JREG	Zachowaj zawartość rJ.
06		LDA JREG(0:2)	Pobierz lokację startową śledzenia.
07	CYCLE	STA PREG(0:2)	Zapisz lokację następnego rozkazu.
08	TEST	DECA *	Czy to jest lokacja końcowa?
09		JAZ LEAVE	
10	PREG	LDA *	Pobierz następny rozkaz.
11		STA INST	Skopiuj go.
12		SRA 2	
13		STA INST1(0:3)	Zapisz adres i część indeksującą.
14		LDA INST(5:5)	Pobierz C (kod operacji).
15		DECA 38	
16		JANN 1F	Czy C ≥ 38 (JRED)?
17		INCA 6	
18		JANZ 2F	Czy C $\neq 32$ (STJ)?
19		LDA INST(0:4)	
20		STA *+2(0:4)	Zmień STJ na STA.
21	JREG	ENTA *	rA \leftarrow symulowana zawartość rJ.
22		STA *	
23		JMP INCP	
24	2H	DECA 2	
25		JANZ 2F	Czy C $\neq 34$ (JBUS)?
26		JMP 3F	
27	1H	DECA 9	Sprawdź rozkazy skoku.
28		JAP 2F	C > 47 (JXL)?
29	3H	LDA 8F(0:3)	Wykryto rozkaz skoku;
30		STA INST(0:3)	zmień jego adres na „JUMP”.
31	2H	LDA AREG	Odtwórz rejestr A.
32	*		Wszystkie rejesty oprócz J mają teraz wartości takie, jak w programie głównym.
33	*		
34	INST	NOP *	Wykonanie rozkazu.
35		STA AREG	Zapisz ponownie rejestr A.
36	INCP	LDA PREG(0:2)	Przejdź do następnego rozkazu.
37		INCA 1	
38		JMP CYCLE	

39	8H	JSJ	JUMP	Stała dla wierszy 29 i 40.
40	JUMP	LDA	8B(4:5)	Wystąpił skok.
41		SUB	INST(4:5)	Czy to był JSJ?
42		JAZ	*+4	
43		LDA	PREG(0:2)	Jeśli nie, to zmodyfikuj symulowany
44		INCA	1	rejestr J.
45		STA	JREG(0:2)	
46	INST1	ENTA	*	
47		JMP	CYCLE	Przejdź pod adres skoku.
48	LEAVE	LDA	AREG	Odtwórz rejestr A.
49	LEAVEX	JMP	*	Zakończ śledzenie.
50	AREG	CON	0	Symulowana zawartość rA. ■

Zwróćmy uwagę na następujące fakty dotyczące zarówno powyższego programu, jak i programów śledzących w ogóle.

1) Przedstawiliśmy jedynie najciekawszą część programu śledzącego – tę, która zapobiega „ucieczce” sterowania podczas wykonania innego programu. Program śledzący nie na wiele się przyda, jeżeli nie będzie miał podprogramu wyprowadzającego zawartość rejestrów, a tej części nie zamieściliśmy. Jest to część bardzo ważna, jednak zaciemniłaby ogólny obraz. Niezbędne modyfikacje pozostawiamy jako ćwiczenie (zobacz ćwiczenie 2).

2) Pamięć jest zasadniczo cenniejsza niż czas. To znaczy, że program powinien być jak najkrótszy, bo tylko wtedy będzie go można załadować razem z bardzo dużym programem głównym. O czas nie ma się co bić, bo i tak lwią jego część późniejsze wypisywanie wyników.

3) Trzeba bardzo uważać, żeby nie zniszczyć zawartości rejestrów; w istocie nasz program korzysta tylko z rejestrów A maszyny MIX. Ani wskaźnik porównania, ani znacznik przeniesienia nie są modyfikowane przez program śledzący. (Im mniej rejestrów użyjemy, tym mniej rejestrów będziemy musieli odtwarzać).

4) Po wystąpieniu skoku do lokacji JUMP nie musimy wykonywać „STA AREG”, bo zawartość rA nie mogła się zmienić.

5) Po wyjściu z programu śledzącego rejestr J nie jest ustawiony poprawnie. Z ćwiczenia 1 wynika, jak można temu zaradzić.

6) Śledzony program podlega tylko trzem ograniczeniom:

- a) Nie może zapisywać niczego do lokacji używanych przez program śledzący.
- b) Nie może korzystać z urządzenia wejścia-wyjścia, na którym program śledzący zapisuje informacje (na przykład wskazania instrukcji JBUS byłyby niepoprawne).
- c) Podczas śledzenia będzie działał wolniej.

ĆWICZENIA

1. [22] Zmodyfikuj tak przedstawiony program śledzący, żeby odtwarzał rejestr J. (Możesz założyć, że zawartość rejestrów J jest niezerowa).

2. [26] Zmodyfikuj tak przedstawiony program śledzący, żeby przed wykonaniem każdego kroku wypisywał na jednostce taśmowej nr 0 następujące informacje:

Słowo 1, pola (0:2): lokacja.

Słowo 1, pola (4:5): rejestr J (przed wykonaniem).

Słowo 1, pola (3:3): 2, jeśli wskaźnik porównania ustawiony na GREATER; 1, gdy na EQUAL; 0, gdy na LESS; plus 8, jeśli znacznik przepelnienia jest zapalony przed wykonaniem rozkazu.

Słowo 2: rozkaz.

Słowo 3: rejestr A (przed wykonaniem).

Słowa 4–9: rejesty I1–I6 (przed wykonaniem).

Słowo 10: rejestr X (przed wykonaniem).

Słowa 11–100 każdego 100-słowowego bloku taśmy powinny zawierać kolejne dziewięć takich grup po dziesięć słów (w powyższym formacie).

- 3. [10] W poprzednim ćwiczeniu wymaga się, by program śledzący zapisywał informacje na taśmie. Uzasadnij, dlaczego jest to lepsze rozwiązanie niż bezpośrednie wysyłanie informacji na drukarkę.
- 4. [25] Co by się stało, gdybyśmy programem śledzącym śledzili *jego samego*? Co się w szczególności zdarzy, jeżeli instrukcje ENTX LEAVEX; JMP *+1 umieścimy bezpośrednio przed ENTER?
- 5. [28] Podążając tropem poprzedniego ćwiczenia, zastanów się, co by się stało, gdybyśmy umieścili w pamięci dwie kopie programu śledzącego i kazali każdej z nich śledzić tę drugą.
- 6. [40] Napisz program śledzący, który potrafi śledzić samego siebie, w sensie określonym w ćwiczeniu 4: powinien w zwolnionym tempie drukować kroki własnego wykonania, polegającego na śledzeniu samego siebie w jeszcze wolniejszym tempie i tak ad infinitum, aż do wyczerpania pojemności pamięci.
- 7. [25] Omów wydajną realizację *programu śledzącego skoki*. Taki program generuje o wiele mniejszą ilość informacji niż zwykły program śledzący. Zamiast wyświetlać zawartość rejestrów, program informuje o wykonanych skokach. Dane wyjściowe takiego programu to ciąg par $(x_1, y_1), (x_2, y_2), \dots$ oznaczający, że program wykonał skok z lokacji x_1 pod y_1 , następnie (wykonawszy rozkazy w lokacjach $y_1, y_1 + 1, \dots, x_2$) skoczył z x_2 pod y_2 itd. [Z tych informacji za pomocą innego programu można odtworzyć przepływ sterowania w programie i wyznaczyć, ile razy wykonał się każdy rozkaz].

1.4.4. Wejście i wyjście

Prawdopodobnie najbardziej namacalną różnicą między komputerami są dostępne urządzenia wejścia-wyjścia, a co za tym idzie rozkazy sterujące tymi urządzeniami. W jednej książce nie da się poruszyć wszystkich problemów związanych z tą dziedziną, skupimy się zatem na typowych metodach realizacji wejścia-wyjścia. Operacje wejścia-wyjścia komputera MIX stanowią kompromis skrajnie różnych mechanizmów udostępnianych przez rzeczywiste maszyny. By wyrobić sobie właściwe podejście do operacji wejścia-wyjścia, zastanowimy się, jak najlepiej realizować je na komputerze MIX.

 Znów prosimy Czytelnika o wyrozumiałość dla anachronicznego komputera MIX z kartami perforowanymi itp. Chociaż takie urządzenia wydają się dziś bardzo staroświeckie, wciąż z ich pomocą możemy nauczyć się czegoś istotnego. Oczywiście za pomocą komputera MMIX (gdy się wreszcie pojawi), nauczylibyśmy się tego lepiej.

Wielu użytkowników komputerów czuje, że wejście i wyjście to nie jest w zasadzie „prawdziwe” programowanie. Wyrowadzanie i wprowadzanie informacji uważa się za mozolne zadania, które wykonywać trzeba jedynie w celu włożenia lub wyjęcia danych z maszyny. Z tego powodu naukę mechanizmów wejścia-wyjścia pozostawia się zazwyczaj na koniec, a często się zdarza, że w zespole tylko garstka programistów wie cokolwiek o szczegółach wejścia i wyjścia. To podejście jest poniekąd naturalne, ponieważ mechanizmy wejścia-wyjścia na żadnym komputerze nie są specjalnie eleganckie. Nie należy się jednak spodziewać poprawy sytuacji, dopóki więcej osób nie zajmie się tym zagadnieniem na poważnie. W tym i w innych rozdziałach (na przykład w punkcie 5.4.6) zobaczymy, że zagadnienia wejścia-wyjścia dotyczą mimo wszystko ciekawych problemów i eleganckich algorytmów.

Zapewne jest to odpowiednie miejsce na krótką dygresję na temat słownictwa. Chociaż podstawowe słowa „wejście” i „wyjście” są rzeczownikami („Co tam mamy na wyjściu?”), często korzystamy z nich w formie przymiotników odrzecznikowych („Nie wyrzucaj taśmy wejściowej.”). Pojawiają się też analogiczne formy czasownikowe „wprowadzać” i „wyprowadzać” („Program wyprowadza brzydkie słowo na drukarkę.”), ale częściej w ich miejscu używa się równoważnych terminów „wczytywać” i „zapisywać” („Program nie zapisuje danych do pliku.”). Czasami mówi się też o „ładowaniu”, ale termin ten zazwyczaj zarezerwowany jest dla wprowadzania wykonywalnego kodu programów do pamięci maszyny. Zbitka „wejście-wyjście” jest często skracana do „we-wy”. Rzeczy, które się wczytuje lub zapisuje to, ogólnie mówiąc, „dane” (liczba mnoga od „dana”). Tyle, jeśli chodzi o rozważania językowe.

Przypuśćmy, że chcemy wczytać dane z taśmy magnetycznej. Rozkaz **IN**, zdefiniowany w punkcie 1.3.1, powoduje wyłącznie *zainicjowanie* procesu wprowadzania danych, a komputer przechodzi do wykonywania kolejnych rozkazów, równocześnie z trwającą operacją wejścia. Rozkaz „**IN 1000(5)**” rozpoczęte zatem wczytywanie 100 słów z jednostki taśmowej nr 5 do komórek pamięci 1000–1099, ale program, który ją wykonał, nie powinien na razie się do nich odwoływać. Program może mieć pewność, że operacja wejścia się zakończyła jedynie wtedy, gdy (a) zainicjowano inną operację we-wy (tzn. wykonano **IN**, **OUT** lub **IOC**) dotyczącą jednostki 5, albo (b) rozkaz skoku warunkowego **JBUS(5)** lub **JRED(5)** da znać, że jednostka 5 przestała być „zajęta”.

Najprostszym sposobem wczytania bloku z taśmy do lokacji 1000–1099 i zagliwarantowania, że te informacje znajdą się tam, jest wykonanie następujących dwóch rozkazów

IN 1000(5); JBUS *(5). (1)

Korzystaliśmy z tej prostej metody w punkcie 1.4.2 (zobacz wiersze 07–08 i 52–53). Z reguły stosowanie tej metody oznacza jednak poważne marnotrawstwo, gdyż bardzo dużo czasu komputera (rzędu 1000u lub nawet 10000u) marnuje się na wykonywanie w kółko rozkazu „**JBUS**”. Szybkość działania programu można zwiększyć nawet dwukrotnie, jeśli ten dodatkowy czas przeznaczymy na obliczenia. (Zobacz ćwiczenia 4 i 5).

Jedną z metod uniknięcia takiego „aktywnego czekania” jest wykorzystywanie dwóch obszarów pamięci na dane wejściowe: do jednego z nich wczytujemy dane, w tym samym czasie wykonując obliczenia na danych w drugim obszarze. Możemy na przykład rozpoczęć program od rozkazu

IN 2000(5) Rozpocznij czytanie pierwszego bloku. (2)

W dalszej kolejności do wczytania kolejnego bloku możemy posłużyć się następującymi rozkazami

ENT1 1000	Przygotuj operację MOVE.
JBUS *(5)	Czekaj na gotowość jednostki 5.
MOVE 2000(50)	(2000–2049) → (1000–1049).
MOVE 2050(50)	(2050–2099) → (1050–1099).
IN 2000(5)	Rozpocznij wczytywanie następnego bloku.

(3)

Rezultat jest taki sam, jak dla (1), ale taśma pracuje w tym samym czasie, co program wykonujący obliczenia na danych w lokacjach 1000–1099.

Ostatni rozkaz (3) rozpoczyna wczytywanie bloku taśmy do lokacji 2000–2099 przed przetworzeniem poprzedniego bloku. Tę metodę nazywamy *czytaniem z wyprzedzeniem*. Nowy blok jest wczytywany w nadzieję, że w przyszłości będzie potrzebny. W rzeczywistości po przetworzeniu bloku w lokacjach 1000–1099 może się okazać, że nie interesują nas dalsze dane wejściowe. Rozważmy na przykład analogiczną sytuację w przykładzie wykorzystania współprogramów z punktu 1.4.2, gdzie dane wejściowe pochodzą nie z taśmy, lecz z kart perforowanych: pojawienie się „..” na karcie oznaczało, że jest to karta ostatnia. Taka sytuacja uniemożliwia czytanie z wyprzedzeniem, jeżeli nie założymy, że: (a) na końcu pliku kart dokładamy pustą kartę albo specjalną kartę końcową, lub (b) na przykład w 80-tej kolumnie ostatniej karty umieszczamy specjalny znak identyfikacyjny (na przykład „..”). Jeżeli chcemy korzystać z czytania z wyprzedzeniem, to nie obejdziemy się bez jakiegoś sposobu zakończenia wczytywania danych,

Technika nakładania się czasów obliczeń i wejścia-wyjścia jest znana pod nazwą *buforowania*, podczas gdy prosta metoda (1) jest nazywana wejściem *niebuforowanym*. Obszar pamięci 2000–2099 wykorzystywany do przechowywania bloku wczytanego z wyprzedzeniem w (3), jak również obszar 1000–1099, do którego przenosimy dane wejściowe, nazywany jest *buforem*. Słownik Webstera definiuje „bufor” jako „osobę lub rzecz, która służy do zmniejszenia szoku (siły uderzenia)”. (Inżynierowie używają często słowa „bufor” w innym sensie, a mianowicie na oznaczenie elementu urządzenia wejścia-wyjścia, który odpowiada za przechowywanie informacji podczas transmisji. W tej książce „bufor” będzie oznaczał obszar pamięci wykorzystywany w programie do przechowywania wprowadzanych i wyprowadzanych danych).

Kod (3) nie zawsze jest lepszy od (1), wyjątki się zdarzają, jednak niezmiernie rzadko. Porównajmy czasy wykonania. Założymy, że potrzebny jest czas T na wczytanie 100 słów oraz że C jest czasem obliczeń wykonywanych pomiędzy żądaniemi wprowadzenia danych. Metoda (1) wymaga czasu $T + C$ na blok taśmy, podczas gdy metoda (3) zajmuje $\max(C, T) + 202u$. (Wielkość $202u$ to czas potrzebny na wykonanie dwóch rozkazów MOVE). Powyższy czas wykonania

można rozważyć z punktu widzenia tzw. „czasu ścieżki krytycznej”. Tutaj jest to czas, przez który urządzenie wejścia-wyjścia pozostaje bezczynne między kolejnymi uruchomieniami. W metodzie (1) urządzenie pozostaje bezczynne przez C jednostek czasu, a w metodzie (3) – przez 202 jednostki czasu (przy założeniu, że $C < T$).

Powolny rozkaz MOVE z (3) jest niepożądany, w szczególności dlatego, że zabiera czas na ścieżce krytycznej, gdy jednostka taśmowa musi być nieaktywna. Dzięki oczywistemu usprawnieniu tej metody można obyć się bez rozkazu MOVE: wystarczy zmodyfikować program główny w ten sposób, by odwoływał się na zmianę do lokacji 1000 – 1099 i 2000 – 2099. Podczas wczytywania danych do jednego bufora, możemy wykonywać obliczenia w drugim. Następnie możemy rozpoczęć wczytywanie danych do drugiego bufora, wykonując obliczenia w pierwszym. Ta ważna metoda jest znana jako *przełączanie (wymiana) buforów*. Dane o położeniu aktualnego bufora można przechowywać w rejestrze indeksowym (albo w komórce pamięci, jeżeli nie możemy poświęcić rejestru indeksowego). Widzieliśmy już przykład przełączania buforów w algorytmie 1.3.2P (zobacz kroki P9 – P11) i towarzyszącym mu programie.

Rozważając przykład przełączania buforów wejściowych, przypuśćmy, że program użytkowy w bloku taśmy przechowuje sto oddzielnych elementów wielkości jednego słowa. Poniższy podprogram pobiera kolejne słowo z wejścia, rozpoczynając wczytywanie nowego bloku, jeżeli bieżący blok się wyczerpał.

01	WORDIN	STJ	1F	Zapamiętaj lokację powrotu.	
02		INC6	1	Następne słowo.	
03	2H	LDA	0,6	Czy to koniec	
04		CMPA	=SENTINEL=	bufora?	
05	1H	JNE	*	Jeśli nie, wróć.	
06		IN	-100,6(U)	Wypełnij bufor.	
07		LD6	1,6	Przełącz się na inny bufor	(4)
08		JMP	2B	i wróć.	
09	INBUF1	ORIG	*+100	Pierwszy bufor.	
10		CON	SENTINEL	Wartownik na końcu bufora.	
11		CON	*+1	Adres bliźniaczego bufora.	
12	INBUF2	ORIG	*+100	Drugi bufor.	
13		CON	SENTINEL	Wartownik na końcu bufora.	
14		CON	INBUF1	Adres bliźniaczego bufora. ─	

W tym podprogramie rI6 jest używany do adresowania ostatniego słowa z wejścia. Zakładamy, że program wywołujący nie zmienia zawartości tego rejestru. Symbol U oznacza jednostkę pamięci taśmowej, a symbol SENTINEL (wartownik) oznacza znaną wartość, która *nie występuje* wśród zwykłych danych na taśmie.

Zanotujmy następujące obserwacje:

- 1) Stała SENTINEL (wartownik) pojawia się na 101 pozycji każdego z buforów i ułatwia sprawdzanie, czy bufor się wyczerpał. W wielu przypadkach nie można się jednak posłużyć tą metodą, ponieważ na wejściu może pojawić się każda wartość. Taką metodę moglibyśmy jednak zastosować bez problemu przy wczytywaniu danych z kart dziurkowanych (wartownik na 17 słowie bufora), ponieważ

w roli wartownika mogłoby wystąpić dowolne słowo ujemne (na maszynie **MIX** słowo wczytane z czytnika kart jest zawsze nieujemne).

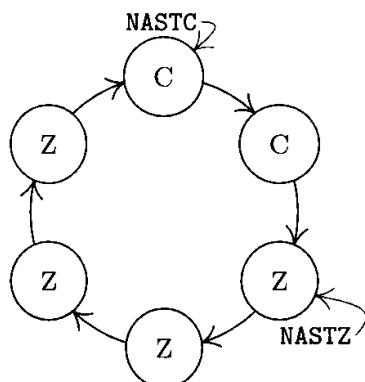
2) Każdy bufor zawiera adres bliźniaczego bufora (zobacz wiersze 07, 11 i 14). Takie „połączenie” ułatwia przełączanie buforów.

3) Nie potrzebowaliśmy rozkazów **JBUS**, ponieważ rozpoczęliśmy wczytywanie nowego bloku przed wykorzystaniem któregokolwiek ze słów z poprzedniego bloku. Jeśli C i T oznaczają czas obliczeń i czas wczytywania z taśmy, jak w poprzednich akapitach, to w tej chwili czas wykonania przypadający na jeden blok taśmy wynosi $\max(C, T)$. Jeśli zatem $C \leq T$, to taśma może cały czas pracować na pełnych obrotach! (*Uwaga:* **MIX** jest komputerem wyidealizowanym, ponieważ program nie musi troszczyć się o błędy we-wy. Na większości maszyn niezbędne byłoby wykonanie rozkazów sprawdzających wynik poprzedniej operacji przed użyciem „**IN**”).

4) Podprogram (4) nie będzie działał poprawnie, jeżeli nie zadbamy o odpowiednie ustawienia na początku pracy. Przeanalizowanie szczegółów pozostawiamy Czytelnikowi (zobacz ćwiczenie 6).

5) Podprogram **WORDIN** sprawia, że reszcie programu wydaje się, iż blok taśmy ma długość 1, a nie 100. Pomyśl polegający na trzymaniu wielu „logicznych” pól w jednym „fizycznym” bloku taśmy nazywamy *grupowaniem rekordów*.

Przedstawione metody z niewielkimi modyfikacjami mają zastosowanie również do operacji wyjścia (zobacz ćwiczenia 2 i 3).



Rys. 23. Pierścień buforów ($N = 6$).

Wiele buforów. Przełączanie buforów jest szczególnym przypadkiem ($N = 2$) ogólnej metody wykorzystującej N buforów. Czasami wygodnie jest mieć więcej niż dwa bufore. Rozważmy algorytm:

Krok 1. Przeczytaj pięć bloków, jeden za drugim.

Krok 2. Wykonaj długie i skomplikowane obliczenia na podstawie wczytanych danych.

Krok 3. Wróć do kroku 1.

W tym przypadku najlepiej byłoby dysponować pięcioma lub sześcioma buforami, tak by podczas kroku 2 można było wczytać kolejną porcję danych. Często operacje wejścia-wyjścia są zgrupowane w czasie, stąd buforowanie wielokrotne daje nam rzeczywiście coś więcej niż przełączanie buforów.

Przypuśćmy, że pewien proces wyjściowy lub wejściowy obsługujący jedno urządzenie wejścia-wyjścia dysponuje N buforami. Wyobraźmy sobie, że te bufory tworzą „pierścień”, jak na rysunku 23. Zakładamy, że z punktu widzenia operacji wejścia-wyjścia program główny ma postać:

```

    :
ASSIGN
    :
RELEASE
    :
ASSIGN
    :
RELEASE
    :

```

innymi słowy, program na zmianę wykonuje „**ASSIGN**” (przydziel) i „**RELEASE**” (zwolnij), pomiędzy którymi następują obliczenia nie mające związku z przydzielaniem buforów.

ASSIGN (przydział) oznacza, że program uzyskuje adres kolejnego bufora; adres jest zapisywany w pewnej zmiennej programu.

RELEASE (zwolnienie) oznacza, że program zakończył pracę na bieżącym buforze.

Pomiędzy **ASSIGN** i **RELEASE** program korzysta z jednego z buforów, nazywanego buforem *bieżącym*. Pomiędzy **RELEASE** i **ASSIGN** program nie korzysta z żadnego bufora.

Być może operacja **ASSIGN** mogłaby następować bezpośrednio po **RELEASE**. Częstokroć analizę buforowania opiera się na tym założeniu. Jeśli jednak operacja **RELEASE** następuje tak szybko, jak tylko jest to możliwe, proces buforujący ma więcej swobody i może dzięki temu być bardziej wydajny. Przekonamy się, że nawet przy rozdzieleniu na oddzielne operacje **ASSIGN** i **RELEASE** metody buforowania nie są niczym trudnym, a zaprezentowane omówienie będzie zawierało istotne spostrzeżenia nawet dla przypadku $N = 1$.

Aby być bardziej dokładnym, rozważmy oddziennie wejście i wyjście. Założymy, że na wejściu korzystamy z czytnika kart. Akcja przydziału **ASSIGN** oznacza, że program potrzebuje informacji z nowej karty. Chcielibyśmy w rejestrze indeksowym dostać adres lokacji, pod którą znajduje się wczytana zawartość karty. Akcja zwolnienia **RELEASE** następuje wtedy, gdy zawartość bieżącej karty nie jest już potrzebna; informacja została „skonsumowana” przez program, być może skopiowana w inne miejsce pamięci itp. Bieżący bufor może zatem posłużyć do wczytania (na zapas) następnej porcji danych.

Przymijmy, że urządzeniem wyjściowym jest drukarka wierszowa. Akcja przydziału **ASSIGN** następuje wtedy, gdy program potrzebuje bufora, w którym będzie budować obraz nowego wiersza. Chcielibyśmy w rejestrze indeksowym

dostać adres lokacji takiego obszaru. Akcja zwolnienia **RELEASE** następuje wtedy, gdy program skompletuje obraz wiersza w buforze, w formie gotowej do wydrukowania.

Przykład: W celu wydrukowania zawartości lokacji 0800–0823 moglibyśmy napisać

```
JMP ASSINP    (rI5←adres bufora).
ENT1 0,5
MOVE 800(24)   Skopiuj 24 słowa do bufora wyjściowego.
JMP RELEASEP
```

(5)

gdzie **ASSINP** i **RELEASEP** są adresami podprogramów realizujących buforowanie drukarki wierszowej.

W sytuacji optymalnej operacja przydziału **ASSIGN** powinna być wykonywana niemal w zerowym czasie. Dla wejścia oznacza to, że obraz każdej karty zostaje wczytany z wyprzedzeniem; dla wyjścia zaś, że zawsze można znaleźć wolne miejsce w pamięci, gdzie program mógłby zbudować obraz wiersza do wydrukowania. W obu przypadkach nie tracimy czasu w oczekiwaniu nagotowość urządzenia wejścia-wyjścia.

Autorowi trudno ukryć fakt, że chciałby opis algorytmu buforowania nieco ubrać. Będziemy mówić, że obszar bufora jest albo zielony, albo żółty, albo czerwony (na rysunku 24 oznaczony Z, Ż i C).

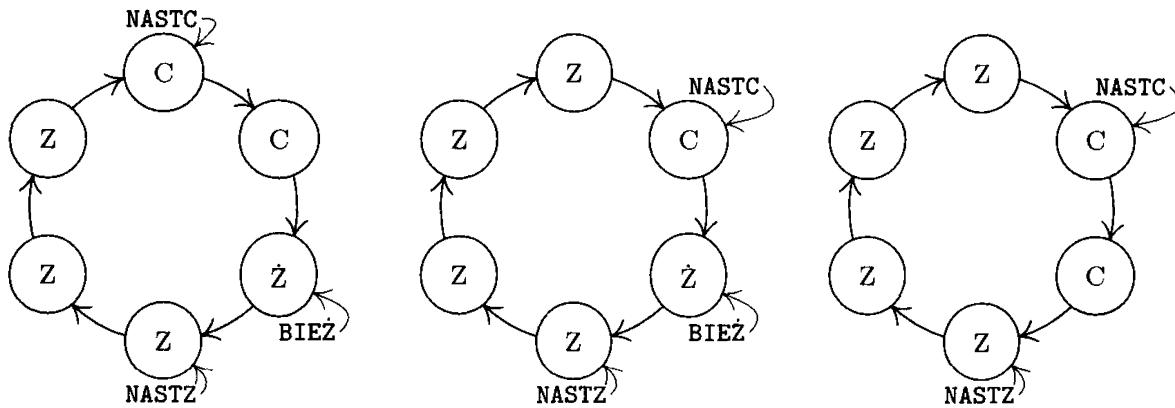
Zielony oznacza, że obszar można przydzielić (operacją **ASSIGN**), czyli że (w przypadku wejścia) do obszaru zostały wczytane dane lub że (w przypadku wyjścia) obszar jest wolny.

Żółty oznacza, że obszar został przydzielony (operacją **ASSIGN**), ale nie zwolniony (operacją **RELEASE**), czyli że obszar jest bieżącym buforem i program korzysta z jego zawartości.

Czerwony oznacza, że obszar został zwolniony (operacją **RELEASE**), czyli że (w przypadku wejścia) obszar jest wolny lub (w przypadku wyjścia) że został zapełniony informacją do wprowadzenia.

Na rysunku 23 są pokazane dwa „wskaźniki” związane z pierścieniem buforów. Należy myśleć o nich jak o rejestrach indeksowych programu. **NASTZ** i **NASTC** wskazują odpowiednio na „następny zielony” i „następny czerwony” bufor. Trzeci wskaźnik, **BIEŻ**, pokazany na rysunku 24, wskazuje na bufor żółty, jeżeli taki w danej chwili istnieje.

Przedstawiony poniżej algorytm nadaje się do buforowania zarówno wejścia, jak i wyjścia, ale skupimy się najpierw na wprowadzaniu danych z czytnika kart. Przypuśćmy, że program znalazł się w stanie pokazanym na rysunku 23. Oznacza to, że proces buforujący wczytał z wyprzedzeniem dane z czterech kart i że znajdują się one w zielonych buforach. W tym momencie przebiegają jednocześnie: (a) obliczenia programu po operacji zwolnienia (**RELEASE**), oraz (b) wczytywanie karty do bufora wskazywanego przez **NASTC**. Taki stan będzie się utrzymywał, dopóki urządzenie nie zakończy odczytu (tzn. dopóki urządzenie nie przejdzie w stan gotowości), albo do chwili wykonania przez program operacji



Rys. 24. Zmiany stanu buforów: (a) po przydiale (**ASSIGN**), (b) po zakończeniu operacji we-wy, (c) po zwolnieniu (**RELEASE**).

przydziału (**ASSIGN**). Przypuśćmy, że to ostatnie zdarzenie zajdzie jako pierwsze. Bufor wskazywany przez **NASTZ** zmieni wówczas kolor na żółty (staje się buforem bieżącym), wskaźnik **NASTZ** przesuwa się zgodnie z ruchem wskazówek zegara i ustawia na pozycję pokazaną na rysunku 24(a). Zakończenie się operacji wyjścia oznacza wczytanie z wyprzedzeniem kolejnego całego bloku, zatem bufor zmienia kolor z czerwonego na zielony, a wskaźnik **NASTC** przemieszcza się zgodnie z rysunkiem 24(b). Jeżeli następną operacją jest operacja zwolnienia (**RELEASE**), to otrzymujemy sytuację jak na rysunku 24(c).

Przykład ilustrujący wyjście zaprezentowano na rysunku 27 na stronie 234. Rysunek ukazuje zmiany „kolorów” buforów w funkcji czasu w programie, który zaczyna się czterema szybkimi operacjami wyjścia, następnie wykonuje cztery operacje rozciągnięte w czasie, by na koniec znów szybko wykonać dwie operacje. W przykładzie korzysta się z trzech buforów.

Wskaźniki **NASTC** i **NASTZ** przesuwają się po pierścieniu zgodnie z ruchem wskazówek zegara, każdy w swoim tempie. Rysunek ilustruje wyścig między programem (który zmienia bufore z zielonych na czerwone) i procesorem buforowania (który zmienia bufore z czerwonych na zielone). Mogą wystąpić dwie sytuacje konfliktowe:

- a) sytuacja, w której wskaźnik **NASTZ** próbuje wyprzedzić **NASTC**, oznacza, że program wyprzedził urządzenie wejścia-wyjścia i musi czekać na gotowość urządzenia;
- b) sytuacja, w której wskaźnik **NASTC** próbuje wyprzedzić **NASTZ**, oznacza, że urządzenie wejścia-wyjścia wyprzedziło program i musimy je wyłączyć do następnej operacji zwolnienia (**RELEASE**).

Obie te sytuacje są pokazane na rysunku 27 (zobacz ćwiczenie 9).

Na szczęście algorytm obsługujący te sytuacje jest prostszy niż jego opis. Przyjmijmy oznaczenia

$$\begin{aligned} N &= \text{całkowita liczba buforów;} \\ n &= \text{aktualna liczba buforów czerwonych.} \end{aligned} \tag{6}$$

Zmiennej n używamy do uniknięcia możliwych konfliktów między wskaźnikami NASTZ i NASTC.

Algorytm A (ASSIGN). Algorytm wykonuje czynności związane z operacją przydziału według opisu przedstawionego powyżej.

- A1. [Czekanie, aż $n < N$] Jeśli $n = N$, wstrzymaj program, dopóki $n < N$. ($n = N$ oznacza, że nie ma wolnych buforów, które mogłyby zostać przydzielone, ale algorytm B, opisany poniżej, działając równolegle z algorytmem A, zapewne w końcu zmieni kolor jakiegoś bufora na zielony).
- A2. [$\text{BIEŻ} \leftarrow \text{NASTZ}$] $\text{BIEŻ} \leftarrow \text{NASTZ}$ (przydzielimy bieżący bufor).
- A3. [Przesuwanie NASTZ] Przesuń NASTZ na następny bufor zgodnie z ruchem wskazówek zegara. ■

Algorytm R (RELEASE). Algorytm wykonuje czynności związane z operacją zwolnienia według opisu przedstawionego powyżej.

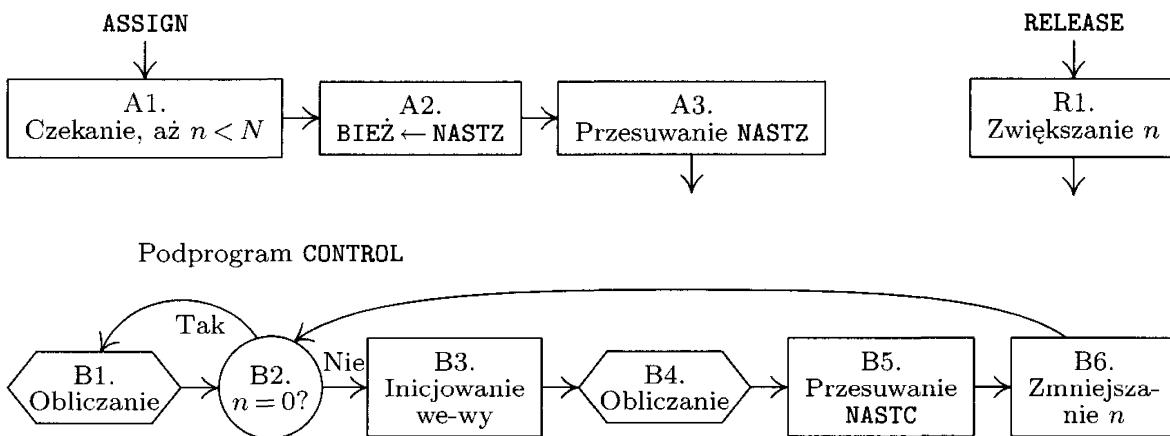
- R1. [Zwiększenie n] Zwiększ n o jeden. ■

Algorytm B (Obsługa bufora). Algorytm inicjuje faktyczną operację wejścia-wyjścia. Operacja jest wykonywana „równocześnie” z programem głównym według opisu przedstawionego powyżej.

- B1. [Obliczanie] Pozwól programowi głównemu przez chwilę wykonywać obliczenia. Krok B2 zostanie wykonany po pewnym czasie, gdy urządzenie wejścia-wyjścia będzie gotowe do wykonania następnej operacji.
- B2. [Czy $n = 0?$] Jeśli $n = 0$, przejdź do kroku B1. (Jeśli brak czerwonych buforów, nie można wykonać operacji wejścia-wyjścia).
- B3. [Inicjowanie we-we] Inicjuj transmisję danych między buforem wskazywanym przez NASTC a urządzeniem wejścia-wyjścia.
- B4. [Obliczanie] Pozwól programowi głównemu działać przez chwilę, następnie przejdź do kroku B5, gdy wykona się operacja wejścia-wyjścia.
- B5. [Przesuwanie NASTC] Przesuń NASTC na następny bufor zgodnie z ruchem wskazówek zegara.
- B6. [Zmniejszanie n] Zmniejsz n o jeden i przejdź do B2. ■

W powyższych algorytmach „równocześnie” pracują dwa procesy: program obsługujący buforowanie oraz program wykonujący obliczenia. Są one w istocie *współprogramami*, które nazywać będziemy CONTROL i COMPUTE. Współprogram CONTROL wykonuje skok do COMPUTE w krokach B1 i B4; współprogram COMPUTE wykonuje skok do CONTROL za pomocą rozkazów „JRED” (skok, gdy urządzenie gotowe) wstawionych co jakiś czas w jego kod.

Zapisanie tego algorytmu w języku maszyny MIX jest niebywale proste. Dla ułatwienia założmy, że bufory są połączone w ten sposób, że adres słowa *poprzedzającego* bufor jest adresem następnego bufora. Na przykład, dla $N = 3$ buforów mamy $\text{CONTENTS(BUF1 - 1)} = \text{BUF2}$, $\text{CONTENTS(BUF2 - 1)} = \text{BUF3}$ i $\text{CONTENTS(BUF3 - 1)} = \text{BUF1}$.



Rys. 25. Algorytmy buforowania wielokrotnego.

Program A (ASSIGN, podprogram wywoływany przez współprogram COMPUTE). $rI4 \equiv BIEŻ$; $rI6 \equiv n$; sekwencja wywołania: JMP ASSIGN; w momencie wyjścia rX zawiera NASTZ.

```

ASSIGN STJ 9F           Ustaw adres powrotu.
1H    JRED CONTROL(U)  A1. Czekanie, aż  $n < N$ .
      CMP6 =N=
      JE   1B
      LD4  NASTZ        A2.  $BIEŻ \leftarrow NASTZ$ .
      LDX  -1,4          A3. Przesunięcie NASTZ.
      STX  NASTZ
9H    JMP   *           Wyjście. ■
  
```

Program R (RELEASE, kod używany we wspólnym programie COMPUTE). $rI6 \equiv n$. Ten krótki kod należy wstawić tam, gdzie program powinien wykonać operację RELEASE.

```

INC6 1                 R1. Zwiększenie  $n$ .
JRED CONTROL(U)       Ewentualnie skocz do współprogramu CONTROL ■
  
```

Program B (Współprogram CONTROL). $rI6 \equiv n$, $rI5 \equiv NASTC$.

```

CONT1 JMP COMPUTE     B1. Obliczanie.
1H    J6Z  *-1         B2. Czy  $n = 0?$ 
      IN   0,5(U)       B3. Inicjowanie we-wy.
      JMP  COMPUTE     B4. Obliczanie.
      LD5  -1,5         B5. Przesuwanie NASTC.
      DEC6 1            B6. Zmniejszanie  $n$ .
      JMP  1B           ■
  
```

Oprócz powyższych fragmentów programu potrzebny jest także zwykły kod aktywujący współprogramy

```

CONTROL STJ COMPUTEX    COMPUTE STJ CONTROLX
CONTROLX JMP CONT1      COMPUTEX JMP COMP1
  
```

Ponadto w kodzie współprogramu COMPUTE powinniśmy mniej więcej co pięćdziesiąt rozkazów umieszczać „JRED CONTROL(U)”.

Jak widać, program wielokrotnego buforowania sprowadza się do siedmiu rozkazów w kodzie **CONTROL**, ośmiu w kodzie **ASSIGN** i dwóch w kodzie **RELEASE**.

Warto zauważyć, że *dokładnie* taki sam algorytm będzie działać dla wejścia i wyjścia. Różnica polega na tym, że program musi skądś wiedzieć, czy wczytywać na zapas (wejście), czy na zapas przydzielać bufory (wyjście). Skąd? Wystarczy odpowiednio zainicjować n : dla wejścia przypisujemy wstępnie $n = N$ (wszystkie bufory czerwone), dla wyjścia $n = 0$ (wszystkie bufory zielone). Odtąd proces będzie zachowywał się poprawnie, zarówno przy buforowaniu wejścia, jak i wyjścia. Musimy pamiętać także o zainicjowaniu wskaźników **NASTC** = **NASTZ**, by wskazywały na jeden z buforów.

Na zakończenie programu należy zatrzymać proces wejścia-wyjścia (jeżeli wczytuje dane) lub poczekać, aż się zakończy (jeżeli zapisuje dane). Przeanalizowanie szczegółów pozostawiamy Czytelnikowi (zobacz ćwiczenia 12 i 13).

Ważne jest postawienie sobie pytania, jaka powinna być wartość N . Oczywiście, gdy zwiększymy N to szybkość działania programu nie zmniejszy się, ale nie będzie również rosnąć w nieskończoność. Przypomnijmy sobie parametry C i T oznaczające czas, który upływa między kolejnymi operacjami wejścia-wyjścia oraz czas przeprowadzenia operacji. Dokładniej, niech C będzie czasem upływającym między kolejnymi operacjami przydziału (**ASSIGN**), a T czasem transmisji bloku. Jeżeli C jest zawsze *większe* od T , to wystarczy przyjąć $N = 2$, ponieważ nietrudno zauważyc, że dysponując dwoma buforami, możemy osiągnąć pełną wydajność komputera przez cały czas wykonania programu. Jeżeli C jest zawsze *mniejsze* od T , to również wartość $N = 2$ jest optymalna, bo wystarcza, by osiągnąć pełną wydajność urządzenia wejścia-wyjścia (chyba, że urządzenie ma specjalną charakterystykę czasową, jak w ćwiczeniu 19). Większe wartości N mają sens głównie wtedy, gdy C oscyluje między wartościami małymi i dużymi. Wydaje się, że średnia liczba kolejnych małych wartości C plus 1 jest odpowiednią kandydatką na N , jeżeli duże wartości C są znacznie większe od T . (Buforowanie jednak niewiele pomoże, jeżeli wszystkie operacje wejścia występują na początku, a wszystkie operacje wyjścia na końcu programu). Jeżeli czas między przydziałem (**ASSIGN**) a zwolnieniem (**RELEASE**) jest zawsze stosunkowo mały, to na mocy przedstawionego wcześniej rozumowania wartość N można zmniejszyć o jeden bez dużego uszczerbku na czasie wykonania.

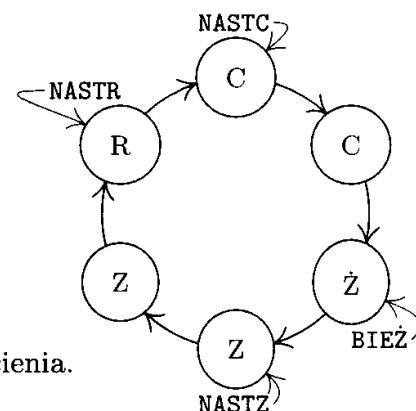
Zaprezentowane podejście do problemu buforowania można na wiele sposobów przystosowywać i rozszerzać. Do tej pory zakładaliśmy, że pracujemy tylko z jednym urządzeniem wejścia-wyjścia. Oczywiście w praktyce pracujemy równocześnie z wieloma urządzeniami.

Problem buforowania wielu urządzeń można rozwiązać na kilka sposobów. Najprostsze podejście to zastosowanie oddzielnego pierścienia buforów dla każdego urządzenia. Z każdym urządzeniem wiążemy właściwe mu n , N , **NASTC**, **NASTZ** i **BIEŻ**, a także oddzielny współprogram **CONTROL**. W ten sposób każde urządzenie wejścia-wyjścia jest buforowane indywidualnie.

Można także „uwspólnić” obszary buforów tego samego rozmiaru w ten sposób, by kilka urządzeń korzystało z buforów ze wspólnej listy. Jedną z metod realizacji tego pomysłu jest wykorzystanie wskaźników, jak w metodzie przedsta-

wionej w rozdziale 2. Należy wówczas utrzymywać listę buforów czerwonych oraz listę buforów zielonych. W takim przypadku rozróżnienie między procesem wyjściowym a wejściowym jest głębsze i wymaga ponownego zakodowania algorytmu bez użycia n i N . Taki algorytm może się zablokować, jeżeli wszystkie bufory zostaną wypełnione danymi wczytanymi z wyprzedzeniem. Należy zatem dbać o to, by zawsze przynajmniej jeden bufor (a jeszcze lepiej jeden bufor dla każdego urządzenia) nie był zielonym buforem wejściowym. Jedynie w przypadku, gdy współprogram COMPUTE jest zablokowany przez krok A1 podczas operacji na pewnym urządzeniu wejściowym, możemy zezwolić na zapełnienie ostatniego bufora z puli przeznaczonej dla tego urządzenia.

W niektórych maszynach na operacje wejścia-wyjścia dotyczące niektórych urządzeń nałożone są dodatkowe warunki, na przykład nie można w tym samym czasie transmitować danych z pewnych par urządzeń (dzieje się tak, gdy wiele urządzeń może być podłączonych do komputera za pomocą jednego „kanału”). Takie uwarunkowania mają wpływ na nasz proces buforujący: jak postępować w sytuacji, gdy musimy dokonać wyboru urządzenia, na którym zainicjować mamy transmisję? Ten problem jest znany pod nazwą problemu „przewidywania”. Najlepsza reguła przewidywania będzie preferowała te urządzenia, dla których wartość n/N jest największa przy założeniu, że rozsądnie dobrano liczby buforów w pierścieniach.



Rys. 26. Wejście i wyjście w ramach jednego pierścienia.

Na zakończenie rozdziału omówimy krótko użyteczną metodę, pozwalającą przy pewnych założeniach wykorzystać *ten sam* pierścień do buforowania zarówno wejścia, jak i wyjścia. Na rysunku 26 jest pokazany nowy rodzaj bufora o kolorze różowym (R). W tym przypadku zielone bufory reprezentują dane wczytane z wyprzedzeniem. W wyniku przydziału (ASSIGN) zielony bufor staje się żółty, aż do wykonania operacji zwolnienia (RELEASE), która powoduje zmianę koloru bufora na czerwony (co reprezentuje fakt, że zawartość bufora ma zostać *wyprowadzona*). Proces wejściowy i wyjściowy niezależnie okrążają pierścień jak poprzednio, z tym że teraz zmieniamy dodatkowo kolor bufora na różowy po zakończeniu wyprowadzania jego zawartości. Ponadto kolor z różowego zmieniamy na zielony przy operacji wejścia. Należy zadbać o to, żeby wskaźniki NASTZ, NASTC i NASTR nie wyprzedzały się nawzajem. W chwili pokazanej na rysunku 26 program wykonuje obliczenia między operacjami przydziału (ASSIGN)

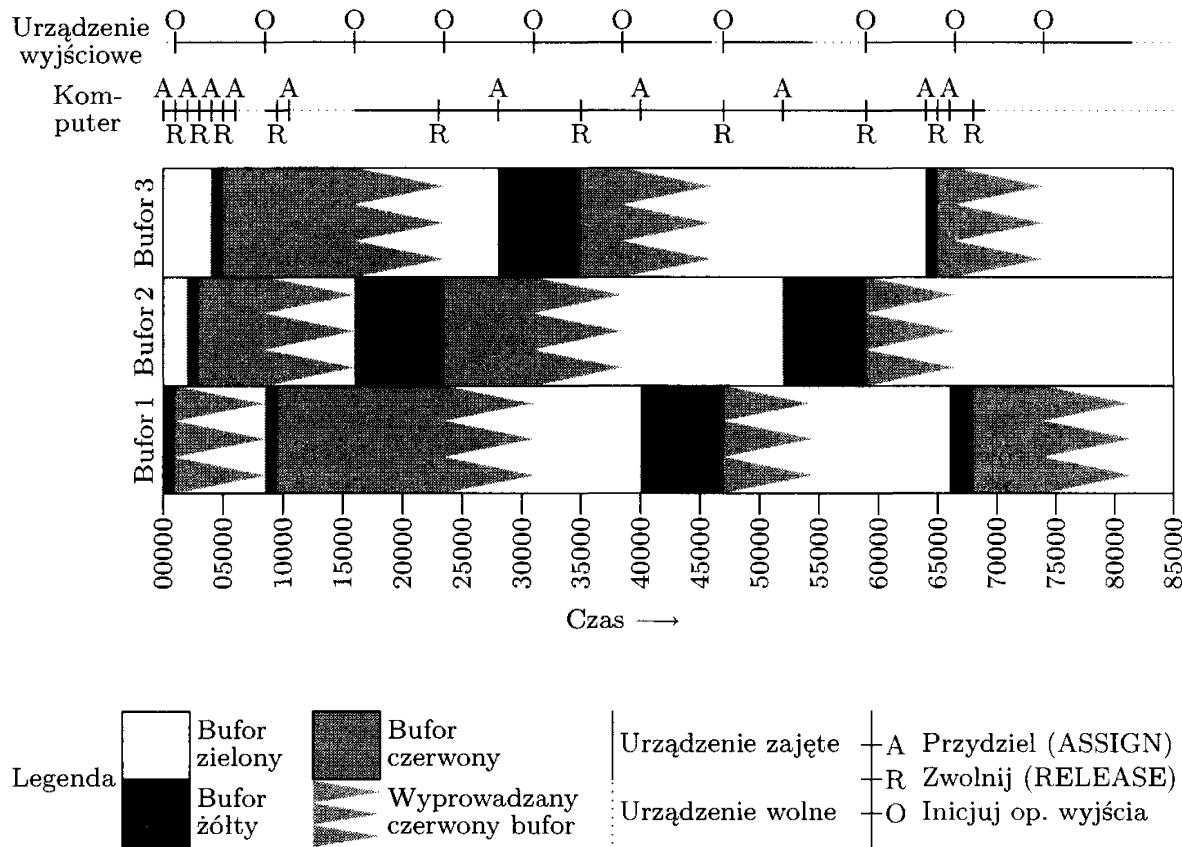
i zwolnienia (**RELEASE**), korzystając z żółtego bufora. Równocześnie odbywa się wczytywanie danych do bufora wskazywanego przez NASTR i wyprowadzanie danych z bufora wskazywanego przez NASTC.

ĆWICZENIA

1. [05] Czy kod (3) będzie poprawny, jeżeli oba rozkazy **MOVE** umieścimy przed **JBUS**? A jeśli umieścimy je za rozkazem **IN**?
2. [10] Za pomocą rozkazów „**OUT 1000(6); JBUS *(6)**” można wyprowadzić blok danych na urządzenie taśmowe bez buforowania, podobnie jak za pomocą rozkazów (1) można wprowadzić dane z wejścia. Podaj metodę analogiczną do (2) i (3) realizującą buforowanie wejścia za pomocą rozkazu **MOVE** i dodatkowego bufora w lokacjach 2000 – 2099.
- ▶ 3. [22] Napisz podprogram przełączający bufore dla operacji wyjścia analogiczny do (4). Podprogram o nazwie **WORDOUT** powinien zapisywać w pamięci zawartość rejestru **rA** zawierającego kolejne słowo wyjścia, a w chwili zapełnienia bufora powinien wyprowadzać 100 słów na jednostkę taśmową **V**. Do przechowywania informacji o pozycji w bieżącym buforze należy wykorzystać rejestr indeksowy **rI5**. Zaprezentuj układ buforów i odpowiedź, czy i które rozkazy na początku i końcu programu gwarantują poprawny zapis pierwszego i ostatniego bloku danych. Jeśli zachodzi potrzeba, do ostatniego bloku wpisz zera.
4. [M20] Pokaż, że jeśli program odwołuje się do pojedynczego urządzenia wejścia-wyjścia, to w sprzyjających okolicznościach zastosowanie buforowania wejścia-wyjścia umożliwia zmniejszenie czasu jego działania o połowę. Pokaż także, że za pomocą tej metody nie można zmniejszyć czasu wykonania więcej niż dwa razy względem czasu wykonania programu nie korzystającego z buforowania.
- ▶ 5. [M21] Uogólnij poprzednie ćwiczenie na n urządzeń wejścia-wyjścia.
6. [12] Jakie rozkazy należy umieścić na początku tworzonego programu, żeby podprogram **WORDIN** z (4) wykonywał się poprawnie? (Trzeba na przykład *jakoś* zainicjować **rI6**).
7. [22] Napisz podprogram **WORDIN**, który robi to samo co (4), tyle że nie korzysta z wartownika.
8. [11] Tekst opisuje hipotetyczny scenariusz dotyczący operacji wejścia, którego kolejne kroki są pokazane na rysunkach 23 oraz 24(a), (b) i (c). Zinterpretuj ten sam scenariusz, zakładając, że mówimy o operacjach wyjścia dotyczących drukarki wierszowej, a nie o operacjach wejścia dotyczących czytnika kart. (Co na przykład dzieje się w sytuacji przedstawionej na rysunku 23?)
- ▶ 9. [21] Program, w wyniku wykonania którego zawartość buforów jest jak na rysunku 27, można scharakteryzować następująco:

*A, 1000, R, 1000, A, 1000, R, 1000, A, 1000, R, 1000, A, 1000, R, 1000,
A, 7000, R, 5000, A, 7000, R, 5000, A, 7000, R, 5000, A, 7000, R, 5000,
A, 1000, R, 1000, A, 2000, R, 1000.*

Ten zapis oznacza „przydziel (A), obliczaj przez 1000u, zwolnij (R), obliczaj przez 1000u, przydziel, ..., obliczaj przez 2000u, zwolnij, obliczaj przez 1000u”. Czasy obliczeń nie obejmują czasów oczekiwania na nie nadążające urządzenie wejścia-wyjścia



Rys. 27. Wyjście realizowane z wykorzystaniem trzech buforów (zobacz ćwiczenie 9).

(zobacz czwarta operacja przydziału na rysunku 27). Urządzenie wyjściowe działa z szybkością $7500u$ na blok.

Poniższe zestawienie zawiera czasy wystąpienia akcji pokazanych na rysunku 27:

Czas	Akcja	Czas	Akcja
0	ASSIGN(BUF1)	38500	OUT BUF3
1000	RELEASE, OUT BUF1	40000	ASSIGN(BUF1)
2000	ASSIGN(BUF2)	46000	Zakończenie operacji wyjścia.
3000	RELEASE	47000	RELEASE, OUT BUF1
4000	ASSIGN(BUF3)	52000	ASSIGN(BUF2)
5000	RELEASE	54500	Zakończenie operacji wyjścia.
6000	ASSIGN (oczekiwanie)	59000	RELEASE, OUT BUF2
8500	BUF1 przydzielony, OUT BUF2	64000	ASSIGN(BUF3)
9500	RELEASE	65000	RELEASE
10500	ASSIGN (oczekiwanie)	66000	ASSIGN(BUF1)
16000	BUF2 przydzielony, OUT BUF3	66500	OUT BUF3
23000	RELEASE	68000	RELEASE
23500	OUT BUF1	69000	Zakończenie obliczeń.
28000	ASSIGN(BUF3)	74000	OUT BUF1
31000	OUT BUF2	81500	Zakończenie operacji wyjścia.
35000	RELEASE		

Całkowity czas wykonania wynosi zatem $81500u$. Komputer był bezczynny w przedziałach czasowych $6000 - 8500$, $10500 - 16000$ i $69000 - 81500$ – razem $20500u$. Urządzenie

wyjściowe było bezczynne w przedziałach czasowych 0–1000, 46000–47000 i 54500–59000 – razem 6500u.

Sporządź analogiczne zestawienie dla tego samego programu, przyjmując, że korzystamy z dwóch buforów.

10. [21] Powtórz ćwiczenie 9 dla *czterech* buforów.
11. [21] Powtórz ćwiczenie 9 dla *jednego* bufora.
12. [24] Przypuśćmy, że z opisanego w tekście algorytmu buforowania wielokrotnego korzystamy do wczytywania danych z kart perforowanych. Wprowadzanie kart ma się zakończyć w momencie przeczytania karty ze znakiem „.” w 80-tej kolumnie. Pokaż jak zmodyfikować współprogram CONTROL (algorytm B i program B), by ciąg operacji wejścia kończyć w ten właśnie sposób.
13. [20] Założymy, że algorytmu buforowania używamy do obsługi wyjścia. Jakie rozkazy należy wstawić na końcu współprogramu COMPUTE, by zagwarantować, że wszystkie informacje z buforów wyjściowych zostaną zapisane?
- 14. [20] Przypuśćmy, że program wykonujący obliczenia nie wykonuje na przemian operacji ASSIGN i RELEASE, lecz ciąg akcji ...ASSIGN...ASSIGN...RELEASE...RELEASE. Jaki ma to wpływ na algorytmy opisane w tekście? Czy może się to do czegoś przydać?
- 15. [22] Napisz kompletny program na maszynę MIX, który kopiuje 100 bloków z jednostki taśmowej 0 na jednostkę taśmową 1, korzystając z trzech buforów. Program powinien być jak najszybszy.
16. [29] Sformułuj algorytm „zielony-żółty-czerwony-różowy” opisany pod koniec rozdziału. Kieruj się sposobem przedstawienia algorytmu buforowania wielokrotnego i zaproponuj trzy współprogramy (sterujący urządzeniem wejściowym, sterujący urządzeniem wyjściowym, wykonujący obliczenia).
17. [40] Zmodyfikuj tak algorytm buforowania wielokrotnego, żeby korzystał ze wspólnych buforów. Wbuduj w algorytm metody zabezpieczające proces przed spowolnieniem spowodowanym faktem wczytania z wyprzedzeniem zbyt dużej ilości danych. Spróbuj sformułować algorytm tak elegancko, jak to tylko możliwe. Zastanów się, jak Twoja metoda radzi sobie z problemami występującymi w praktyce i jak się ma do metod, w których nie korzysta się ze wspólnych buforów.
- 18. [30] Jedno z proponowanych rozszerzeń komputera MIX przewiduje możliwość przerywania obliczeń według wyłożonych poniżej zasad. Twoim zadaniem jest takie zmodyfikowanie algorytmów A, R i B przedstawionych w tekście, by zamiast z rozkazu „JRED” korzystały z tego nowego mechanizmu.

Nową cechą maszyny MIX jest rozszerzenie pamięci o dodatkowe komórki o adresach od -3999 do -0001. Maszyna ma dwa wewnętrzne stany: *stan normalny* i *stan obsługi przerwania*. W stanie normalnym lokacje od -3999 do -0001 są niedostępne, a komputer MIX zachowuje się po staremu. W chwili wystąpienia „przerwania” do lokacji -0009...-0001 wpisywana jest zawartość rejestrów: rA do -0009; rI1...rI6 do -0008...-0003; rX do -0002; rJ, znaczek przepełnienia, wskaźnik porównania i lokacja następnego rozkazu są wpisywane do lokacji -0001 jako

+	nast. inst.	OV, CI	rJ	,
---	----------------	-----------	----	---

a maszyna wchodzi w „stan obsługi przerwania” i rozpoczyna wykonanie programu od lokacji, której adres zależy od rodzaju przerwania.

Lokacja -0010 służy za „zegar”: co $1000u$ liczba w tej lokacji jest zmniejszana o jeden i jeśli w wyniku tego jej zawartość wyniesie zero, to następuje przerwanie połączone z wykonaniem kodu począwszy od lokacji -0011 .

Nowy rozkaz „INT” ($C = 5, F = 9$) jest wykonywany następująco: (a) w stanie normalnym powoduje wystąpienie przerwania połączonego z wykonaniem kodu począwszy od lokacji -0012 . (W ten sposób programista może wymusić przerwanie w celu porozumienia się z podprogramem sterującym; adres w rozkazie INT nie jest przez maszynę MIX brany pod uwagę, jednak program sterujący może na jego podstawie rozróżnić typy przerwań). (b) W stanie obsługi przerwania zawartość rejestrów maszyny MIX jest ładowana spod lokacji $-0009 \dots -0001$, komputer przechodzi w stan normalny i wznowia wykonanie przerwanego programu. W obu przypadkach czas wykonania rozkazu INT wynosi $2u$.

Rozkazy IN, OUT oraz IOC wykonane w stanie obsługi przerwania spowodują wygenerowanie przerwania w chwili zakończenia operacji wejścia-wyjścia. Takie przerwanie jest połączone z wykonaniem kodu począwszy od lokacji $-(0020+ \text{numer jednostki})$.

W stanie obsługi przerwań nie obsługuje się kolejnych przerwań; przerwania są „zachowywane” i zostaną zgłoszone po wykonaniu rozkazu INT oraz jednego rozkazu przerwanego programu po powrocie do stanu normalnego.

- ▶ 19. [M28] Jeśli urządzenie wejściowe lub wyjściowe przechowuje krótkie bloki danych na obrotowym nośniku (na przykład na dysku magnetycznym), to musimy brać ten fakt pod uwagę. Przypuśćmy, że program operuje na $n \geq 2$ kolejnych blokach danych w następujący sposób: blok k jest wczytywany począwszy od chwili t_k , gdzie $t_1 = 0$. Blok jest przydzielany do przetwarzania w chwili $u_k \geq t_k + T$, a bufor jest zwalniany w chwili $v_k = u_k + C$. Dysk wykonuje jeden obrót w ciągu P jednostek czasu, a głowica czytająca przesuwa się nad początkiem nowego bloku co L jednostek. Musimy zatem mieć $t_k \equiv (k-1)L$ (modulo P). Z uwagi na fakt, że przetwarzanie jest sekwencyjne, musi ponadto zachodzić $u_k \geq v_{k-1}$ dla $1 < k \leq n$. Korzystamy z N buforów, zatem $t_k \geq v_{k-N}$ dla $N < k \leq n$.

Jak duże musi być N , by czas zakończenia v_n miał najmniejszą z możliwych wartości, $T+C+(n-1)\max(L,C)$? Podaj ogólną regułę na wyznaczanie najmniejszego N . Zaprezentuj działanie swojej reguły dla $L = 1, P = 100, T = 0.5, n = 100$ i (a) $C = 0.5$; (b) $C = 1.0$; (c) $C = 1.01$; (d) $C = 1.5$; (e) $C = 2.0$; (f) $C = 2.5$; (g) $C = 10.0$; (h) $C = 50.0$; (i) $C = 200.0$.

1.4.5. Historia i bibliografia

Większość podstawowych metod omówionych w podrozdziale 1.4 powstawało w wielu miejscach niezależnie, zatem dokładnej ich historii pewnie nigdy nie poznamy. W tym punkcie spróbujemy odnotować najważniejsze osiągnięcia oraz ich kontekst historyczny.

Podprogramy były pierwszymi udogodnieniami mającymi na celu zaoszczędzenie pracy programistycie. W XIX wieku Charles Babbage snuł wizje biblioteki podprogramów dla swojej Maszyny Analitycznej [zobacz *Charles Babbage and His Calculating Engines*, red. Philip i Emily Morrison (Dover, 1961), 56]. Można powiedzieć, że jego marzenie urzeczywistniło się w 1944 roku, kiedy to Grace M. Hopper napisała podprogram obliczający $\sin x$ na kalkulatorze Harvard Mark [zobacz *Mechanization of Thought Processes* (London: Nat. Phys. Lab., 1959), 164]. Jednakże te „podprogramy otwarte” miały być po prostu wstawia-

ne w odpowiednie miejsca programu, a nie wywoływanie dynamicznie. Projekt przewidywał, że maszyna Babbage'a będzie sterowana kartami perforowanymi jak krosna Jacquarda. Kalkulator Mark I był sterowany taśmami papierowymi. Tak więc różniły się one od dzisiejszych komputerów przechowujących program w pamięci.

Mechanizm wywoływania podprogramów dla komputerów przechowujących program w pamięci, zakładający przekazywanie adresu powrotnego jako parametru, przedstawili Herman H. Goldstine i John von Neumann w swojej niezwykle poczytnej monografii z lat 1946–1947; zobacz: J. von Neumann *Collected Works 5* (New York: Macmillan, 1963), 215–235. W zaproponowanym przez nich rozwiązaniu program główny nie przekazywał parametrów w rejestrach, lecz umieszczał je wprost w ciele podprogramu. W Anglii już w 1945 roku A. M. Turing zaprojektował sprzęt i oprogramowanie umożliwiające wywoływanie podprogramów; zobacz: *Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery* (Cambridge, Mass.: Harvard University, 1949), 87–90; red. B. E. Carpenter, R. W. Doran, A. M. Turing's ACE Report of 1946 and Other Papers (Cambridge, Mass.: MIT Press, 1986), 35–36, 76, 78–79. Zaprogramowanie i wykorzystanie bardzo uniwersalnej biblioteki podprogramów jest głównym tematem pierwszego podręcznika programowania, *The Preparation of Programs for an Electronic Digital Computer*, M. V. Wilkes, D. J. Wheeler, S. Gill, wyd. 1 (Reading, Mass.: Addison–Wesley, 1951).

Słowo „współprogram” (*routine*) zostało wymyślone przez M. E. Conwaya w 1958 roku, po tym jak opracował i jako pierwszy wykorzystał ten pomysł do napisania programu asemblerowego. Mniej więcej w tym samym czasie współprogramami niezależnie od niego zajmowali się J. Erdwinn i J. Merner. Napisali oni artykuł zatytułowany „Bilateral Linkage”, którego nie uznano za wart opublikowania i z tego powodu prawdopodobnie żadna jego kopia nie dotrwała do dziś. Pierwsze publiczne objaśnienie idei współprogramów ukazało się dużo później w artykule Conwaya „Design of a Separable Transition-Diagram Compiler”, *CACM 6* (1963), 396–408. W istocie współprogramy w bardzo prostej postaci pojawiły się jako „sztuczka programistyczna” we wczesnej publikacji UNIVAC-a [*The Programmer 1, 2* (luty 1954), 4]. Notacja dla współprogramów w językach podobnych do języka ALGOL została wprowadzona przez Dahlę i Nygaarda w artykule SIMULA I [*CACM 9* (1966), 671–678], a kilka pięknych przykładów zastosowania współprogramów (również współprogramów wielokrotnych) zawartych jest w książce O.-J. Dahl, E. W. Dijkstra i C. A. R. Hoare'a *Structured Programming*, rozdział 3.

Za pierwszy interpreter można uważać „Uniwersalną Maszynę Turinga”, czyli maszynę Turinga potrafiącą symulować dowolną maszynę Turinga. Maszyny Turinga nie są prawdziwymi komputerami. To konstrukcje teoretyczne wykorzystywane do dowodzenia, że pewne problemy nie dadzą się rozwiązać za pomocą algorytmów. O interpreterach w tradycyjnym sensie na swych wykładach w Moore School w 1946 roku wspominał John Mauchly. Najważniejsze z pierwszych interpreterów miały zapewnić wygodne korzystanie z operacji zmiennopozycyjnych. Były to programy na komputer Whirlwind I (opra-

cowany przez C. W. Adamsa i innych) oraz na komputer Illiac I (autorstwa D. J. Wheelera i innych). Turing też brał udział w takich pracach – systemy interpretujące dla komputera Pilot ACE były pisane pod jego kierownictwem. Bibliografia dotycząca osiągnięć na polu interpreterów w początkach lat pięćdziesiątych znajduje się w artykule „*Interpretative Sub-routines*”, J. M. Bennett, D. G. Prinz, M. L. Woods, *Proc. ACM Nat. Conf.* (1952), 81–87; zobacz też różne artykuły w *Proceedings of the Symposium on Automatic Programming for Digital Computers* (1954), wydanym przez Office of Naval Research, Washington, D.C.

Pośród najintensywniej używanych wczesnych interpreterów ważne miejsce zajmuje system Speedcoding Johna Backusa na „IBM 701” [zobacz *JACM* 1 (1954), 4–6]. Ten interpreter został nieco zmodyfikowany i sprytnie przepisany na IBM 650 przez V. M. Wolontisa i innych pracowników Bell Telephone Laboratories. Ich program pod nazwą „Bell Interpretive System” stał się niezwykle popularny. Interpreter IPL, który od 1956 roku do nieco innych zastosowań (zobacz podrozdział 2.6) projektowali A. Newell, J. C. Shaw i H. A. Simon, był intensywnie wykorzystywany do przetwarzania list. Jak powiedziano we wstępie do punktu 1.4.3, we współczesnej literaturze często mówi się o interpretacji jako o technice zanikającej. Po więcej szczegółów odsyłamy do literatury wymienionej w punkcie 1.4.3.

Pierwszy program śledzący został opracowany przez Stanleya Gilla w 1950 roku, zobacz artykuł tegoż autora w *Proceedings of the Royal Society of London, series A*, **206** (1951), 538–554. Wymieniona praca Wilkesa, Wheelera i Gill'a zawiera kilka programów śledzących. Zapewne najciekawszy z nich jest podprogram C-10 autorstwa D. J. Wheelera, który umożliwia wstrzymanie generowania śladu w momencie wejścia do programu bibliotecznego, wykonanie podprogramu z normalną szybkością i ponowne włącznie śledzenia po powrocie. Publikacje na temat programów śledzących należą do rzadkości w ogólnej literaturze dotyczącej komputerów, jako że stosowane w tej dziedzinie metody są ściśle związane z konkretnym modelem komputera. Jedyną inną pozycję znaną autorowi jest artykuł „*An Experimental Monitoring Routine for the IBM 705*”, H. V. Meek, *Proc. Western Joint Computer Conf.* (1956), 68–70. W pracy jest opisane śledzenie na komputerze, którego konstrukcja wybitnie utrudniała to zadanie. Dziś akcent przesunął się na oprogramowanie umożliwiające symboliczne filtrowanie śladu oraz mierzenie wydajności programu. Jeden z lepszych systemów tego rodzaju został opracowany przez E. Satterthwaite'a i opisany w *Software Practice & Experience* **2** (1972), 197–217.

Początkowo buforowanie realizowane było przez sprzęt, w sposób analogiczny do podprogramu 1.4.4–(3). Wewnętrzny obszar buforowy niedostępny dla programisty grał rolę lokacji 2000–2099, a instrukcje 1.4.4–(3) były wykonywane niejawnie w momencie wykonania operacji wejścia. Pod koniec lat czterdziestych programiści UNIVAC (zobacz podrozdział 5.5) rozwinęli programowe techniki buforowania, szczególnie pożyteczne przy sortowaniu. Niezły przegląd różnych filozofii dotyczących wejścia-wyjścia popularnych w 1952 roku można znaleźć w materiałach Eastern Joint Computer Conference z tegoż roku.

Projekt komputera DYSEAC [Alan L. Leiner, *JACM* 1 (1954), 57–81] wprowadził pomysł polegający na bezpośredniej komunikacji urządzeń wejścia-wyjścia z pamięcią podczas wykonania programu oraz przerywania programu w momencie zakończenia operacji. Taki system wymagał opracowania algorytmów buforowania, ale szczegółów nie opublikowano. Pierwszy opublikowany artykuł na temat metod buforowania w sensie przez nas opisanym prezentuje podejście bardzo wyszukane, zobacz O. Mock, C. J. Swift, „Programmed Input-Output Buffering”, *Proc. ACM Nat. Conf.* (1958), artykuł 19, i *JACM* 6 (1959), 145–151. (Ostrzegamy Czytelnika, że oba artykuły zawierają potężną dawkę lokalnego żargonu, nad zrozumieniem którego trzeba trochę popracować, ale okoliczne artykuły w *JACM* 6 mogą okazać się pomocne). System przerwań umożliwiający buforowanie wejścia i wyjścia został niezależnie opracowany przez E. W. Dijkstrę w 1957 i 1958 roku, w związku z komputerem X-1 B. J. Loopstry i C. S. Scholtena [zobacz *Comp. J.* 2 (1959), 39–43]. W pracy doktorskiej Dijkstry „Communication with an Automatic Computer” (1958) są omówione metody buforowania, które w tym przypadku dotyczyły bardzo długich pierścieni buforów, ponieważ wejście-wyjście było wówczas zorganizowane z wykorzystaniem czytnika taśmy papierowej i dalekopisu. Każdy bufor zawierał pojedynczy znak lub liczbę. Dijkstra rozwinał później swój pomysł w ważną i ogólną technikę *semaforów*, które są podstawowymi pojęciami w zagadnieniach sterowania wykonaniem procesów współbieżnych, niekoniecznie związanych z wejściem-wyjściem [zobacz *Programming Languages*, red. F. Genuys (Academic Press, 1968), 43–112; *BIT* 8 (1968), 174–186; *Acta Informatica* 1 (1971), 115–138]. W artykule „Input-Output Buffering and FORTRAN” autorstwa Davida E. Fergusona, *JACM* 7 (1960), 1–9, są opisane pierścienie buforów oraz proste buforowanie wielu jednostek równocześnie.

Około tysiąca instrukcji to rozsądne górne ograniczenie złożoności znanych dziś problemów.

— HERMAN GOLDSTINE i JOHN VON NEUMANN (1946)

ROZDZIAŁ DRUGI

STRUKTURY DANYCH

*I think that I shall never see
A poem lovely as a tree.*
— JOYCE KILMER (1913)

*Yea, from the table of my memory
I'll wipe away all trivial fond records.*
— Hamlet (akt I, scena 5, wiersz 98)

2.1. WSTĘP

Programy przetwarzają informacje. Informacje nie są bezkształtną masą, zazwyczaj mają strukturę.

Najprostszy przykład zbioru informacji to lista liniowa. Wymieniając właściwości takiej struktury, należy odpowiedzieć na następujące pytania: który element jest pierwszy na liście, który ostatni, który element poprzedza dany element, który element występuje bezpośrednio po nim, ile elementów zawiera ta lista. Nawet w tak prostym przypadku dużo można powiedzieć o strukturze informacji (zobacz podrozdział 2.2).

Przypadek bardziej złożony to dwuwymiarowa tablica (macierz lub siatka mająca kolumny i wiersze), albo nawet tablica o większej liczbie wymiarów. Często spotyka się informacje o strukturze drzewiastej, określającej na przykład związki hierarchiczne. Czasami struktura informacji jest jeszcze bardziej skomplikowana, ma liczne wewnętrzne powiązania przypominające strukturę mózgu.

By poprawnie posługiwać się komputerami musimy zrozumieć te związki strukturalne danych, a także poznać podstawowe metody posługiwania się strukturami danych stanowiącymi reprezentacje informacji w pamięci komputera.

Rozdział 2 stanowi przegląd najważniejszych faktów dotyczących struktur danych: statycznych i dynamicznych właściwości struktur różnego typu; sposobów przydziału pamięci i strukturalnej reprezentacji informacji; efektywnych algorytmów tworzenia, modyfikowania, likwidowania i odzyskiwania informacji. Pokażemy kilka przykładów ilustrujących zastosowanie przedstawionych metod. Przykłady obejmują sortowanie topologiczne, działania na wielomianach, systemy symulacji dyskretnej, przekształcanie macierzy rzadkich, przekształcanie wyrażeń algebraicznych oraz struktury danych wykorzystywane w kompilatorach i systemach operacyjnych. Skupimy się głównie na reprezentacji struktur danych *wewnątrz* komputera. Konwersją z reprezentacji zewnętrznej na reprezentację wewnętrzna zajmujemy się w rozdziałach 9 i 10.

Duża część prezentowanego materiału jest znana pod nazwą „przetwarzania List”, z uwagi na popularność systemów (na przykład LISP) zaprojektowanych w celu ułatwienia pracy z pewną ogólną klasą struktur zwanych *Listami*. (Gdy piszemy słowo „lista” z wielkiej litery, mamy na myśli techniczny sens tego słowa, tj. konkretną strukturę danych omawianą w punkcie 2.3.5). Choć systemy przetwarzania List przydają się w wielu sytuacjach, często narzucają programistom niepotrzebne ograniczenia. Zazwyczaj lepszym rozwiązaniem jest bezpośrednie skorzystanie z metod zaprezentowanych w rozdziale 2 i dopasowanie algorytmu oraz formatu danych do konkretnego problemu. Niestety, wciąż zbyt wielu osobom wydaje się, że przetwarzanie List jest skomplikowane (zatem trzeba koniecznie użyć cudzego, przetestowanego interpretera lub gotowej biblioteki podprogramów) lub że istnieją w tej dziedzinie rozwiązania uniwersalne i jedyne. Przekonamy się, że korzystanie ze złożonych struktur danych nie ma w sobie niczego magicznego, tajemniczego ani trudnego. Takie metody należą do podstawowego repertuaru programisty, a korzystać z nich można, pisząc programy zarówno w języku maszynowym, jak i w FORTRAN-ie, C czy Javie.

Metody realizacji struktur danych pokażemy na przykładzie programów na komputer **MIX**. Czytelnik, którego nie interesują takie szczegóły, powinien przy najmniej zapoznać się ze sposobami reprezentowania struktur danych w pamięci komputera **MIX**.

Doszliśmy do miejsca, w którym koniecznie trzeba zdefiniować nowe pojęcia i oznaczenia, bowiem za chwilę zaczniemy z nich intensywnie korzystać. Struktura danych składa się ze zbioru *węzłów* lub *elementów* (przez niektórych autorów zwanych *rekordami*). Każdy element składa się z jednego lub więcej przylegających słów pamięci, podzielonych na opatrzone nazwami części – *pola*. W najprostszym przypadku element zajmuje pojedyncze słowo maszynowe i składa się z jednego pola obejmującego całe słowo. Ciekawszy przykład: przypuśćmy, że elementy struktury mają reprezentować karty do gry. Możemy posłużyć się elementami o rozmiarze dwóch słów podzielonymi na pięć pól: TAG, SUIT, RANK, NEXT i TITLE, oznaczających odpowiednio znacznik, kolor, wartość, następną kartę i nazwę.

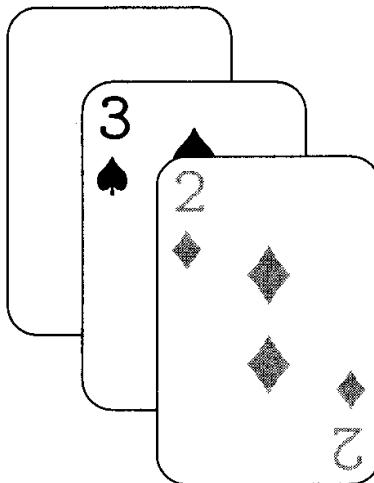
+	TAG	SUIT	RANK	NEXT		(1)
+			TITLE			

(Ten format odpowiada dwóm słowom maszyny **MIX**. Przypominamy, że słowo maszyny **MIX** składa się z pięciu bajtów i znaku; zobacz punkt 1.3.1. W tym przykładzie zakładamy, że znakiem obu słów jest +). *Adres* elementu, zwany także *dowiązaniem*, *wskaźnikiem* lub *odniesieniem* do tego elementu, to lokacja pierwszego słowa. Zazwyczaj adresy podaje się względem jakiegoś ustalonego adresu bazowego, ale w tym rozdziale dla uproszczenia założymy, że mówimy po prostu o bezwzględnych adresach w pamięci.

Zawartością każdego pola może być liczba, ciąg znaków alfabetu, wskaźnik lub co tylko programista sobie wymyśli. W powyższym przykładzie chcemy reprezentować kupkę kart przy stawianiu pasjansa: TAG = 1 oznacza, że karta jest zakryta, TAG = 0, że odkryta. SUIT = 1, 2, 3, 4 oznacza odpowiednio trefl,

karo, kier, pik. $\text{RANK} = 1, 2, \dots, 13$ oznacza asa, dwójkę, ..., króla. NEXT jest wskaźnikiem na następną kartę w kupce. TITLE jest pięcioznakową nazwą karty, pojawiającą się na wydrukach. Przykładowa kupka może wyglądać następująco:

Karty
Reprezentacja kart



100:	+	1	1	10	Λ
101:	+	1	0	T	R E

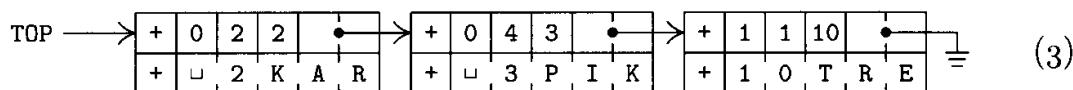
386:	+	0	4	3	100
387:	+	□	3	P	I K

(2)

242:	+	0	2	2	386
243:	+	□	2	K A	R

Adresy, pod którymi przechowywane są reprezentacje kart, to 100, 386 i 242. W tym przykładzie moglibyśmy wybrać jakiekolwiek inne adresy, bo każdy element struktury zawiera dowiązanie do elementu następnego. Zwróćmy uwagę na specjalne dowiązanie „ Λ ” w elemencie pod adresem 100. Będziemy używać greckiej litery lambda na oznaczenie dowiązania pustego, tj. dowiązania, które nie prowadzi do żadnego elementu. Dowiązanie puste Λ pojawia się w elemencie pod adresem 100, ponieważ 10 trefl leży na spodzie kupki. W reprezentacji maszynowej Λ to zazwyczaj jakaś łatwa do rozpoznania wartość, która nie może być adresem elementu. Będziemy zakładać, że żaden element nie może być umieszczony w lokacji zero, zatem Λ będziemy prawie zawsze reprezentować za pomocą wartości 0.

Pojęcie dowiązania do elementów struktury danych jest niezwykle ważne. Dowiązanie to podstawowy sposób reprezentowania złożonych struktur danych. Rysując struktury danych, wygodnie jest oznaczać dowiązania za pomocą strzałek. Przykład (2) możemy przedstawić w takiej postaci:



Zauważmy, że na rysunku nie ma konkretnych adresów 242, 386 i 100 (w końcu i tak są bez znaczenia). Na oznaczenie dowiązania pustego używamy symbolu uziemienia, ściągniętego ze schematów elektrycznych. Zwróćmy ponadto uwagę, że na diagramie (3) na pierwszą kartę wskazuje wartość zapisana w zmiennej TOP. TOP jest tzw. *zmienną wskaźnikową* (lub *dowiązaniową*), tzn. zmienną, której wartościami są dowiązania. Wszystkie odwołania do elementów struktury są realizowane bezpośrednio przez zmienne (lub stałe) dowiązaniowe albo pośrednio przez pola elementów struktury zawierające dowiązania.

Dochodzimy do najważniejszej notacji, umożliwiającej zapisanie odwołania do pól elementów struktury. Będziemy po prostu pisać nazwę pola, po której w nawiasach podamy dowiązanie do elementu struktury. Na przykład w (1), (2) i (3) mamy:

$$\begin{aligned} \text{RANK}(100) &= 10; & \text{SUIT(TOP)} &= 2; \\ \text{TITLE(TOP)} &= „2KAR”; & \text{RANK(NEXT(TOP))} &= 3. \end{aligned} \quad (4)$$

Czytelnik powinien uważnie przyjrzeć się tym przykładom. Z powyższej notacji będziemy korzystać w opisach wielu algorytmów w tym i następnych rozdziałach. Dla przykładu zaprezentujemy prosty algorytm wykonujący przekształcenie na strukturze danych odpowiadającej położeniu na kupce nowo odkrytej karty. Założymy, że NEWCARD jest zmienną wskaźnikową, będącą dowiązaniem do reprezentacji nowej karty.

- A1.** Przymij $\text{NEXT(NEWCARD)} \leftarrow \text{TOP}$. (Ustawiamy dowiązanie w elemencie reprezentującym nową kartę).
- A2.** Przymij $\text{TOP} \leftarrow \text{NEWCARD}$. (Niech TOP wskazuje na nowy wierzch kupki).
- A3.** Przymij $\text{TAG(TOP)} \leftarrow 0$. (Zaznaczamy, że karta jest „odkryta”). ■

Inny przykład: algorytm wyznaczający liczbę kart w kupce:

- B1.** Przymij $\text{N} \leftarrow 0$, $\text{X} \leftarrow \text{TOP}$. (N jest zmienną całkowitoliczbową, X jest zmienną dowiązaniową).
- B2.** Zakończ wykonanie algorytmu, jeśli $\text{X} = \Lambda$; N jest liczbą kart w kupce.
- B3.** Przymij $\text{N} \leftarrow \text{N} + 1$, $\text{X} \leftarrow \text{NEXT(X)}$, wróć do kroku B2. ■

Zauważmy, że w powyższym opisie występują dwa rodzaje nazw symbolicznych: nazwy *zmiennych* (TOP, NEWCARD, N, X) i nazwy *pól* (TAG, NEXT). Nie należy mylić tych dwóch kategorii. Jeśli F jest nazwą pola, a L $\neq \Lambda$ jest dowiązaniem, to F(L) jest zmienną. Ale samo F zmienną nie jest; F nie ma wartości, jeżeli nie występuje w kontekście niepustego dowiązania.

Potrzebujemy jeszcze notacji pozwalającej odwoływać się do wartości przechowywanej pod konkretnym adresem w pamięci:

a) CONTENTS oznacza pole zajmujące cały element o rozmiarze jednego słowa. Zatem CONTENTS(1000) oznacza wartość przechowywaną pod adresem 1000; CONTENTS(1000) traktujemy jako zmienną o tej właśnie wartości. Jeżeli V jest zmienną dowiązaniową, to CONTENTS(V) oznacza wartość, na którą wskazuje V (uwaga, to nie jest wartość V!).

b) Jeśli V jest nazwą pewnej wartości przechowywanej w jakiejś komórce pamięci, to LOC(V) oznacza adres tej komórki. Wynika stąd, że jeżeli V jest zmienną, której wartość jest przechowywana w jednym pełnym słowie pamięci, to CONTENTS(LOC(V)) = V.

Łatwo przepisać tę notację na kod w języku MIXAL, chociaż wersja wykorzystująca ten asembler ma pewne wady. Wartość zmiennej dowiązaniowej

ładujemy do rejestru indeksowego, a odpowiednie pole wyłuskujemy za pomocą mechanizmu wycinków. Na przykład algorytm A można zapisać jako:

NEXT EQU 4:5	Definicja NEXT
TAG EQU 1:1	oraz TAG
LD1 NEWCARD	<u>A1.</u> rI1 \leftarrow NEWCARD.
LDA TOP	rA \leftarrow TOP.
STA 0,1(NEXT)	NEXT(rI1) \leftarrow rA.
ST1 TOP	<u>A2.</u> TOP \leftarrow rI1.
STZ 0,1(TAG)	<u>A3.</u> TAG(rI1) \leftarrow 0. ■

(5)

Łatwość i wydajność, z jaką te operacje dają się realizować na komputerze, jest zasadniczym powodem, dla którego metoda dowiązań jest niesamowicie ważna.

Czasami wygodnie jest dysponować zmienną, w której przechowuje się naraz cały element. Możemy napisać

$$\text{CARD} \leftarrow \text{NODE}(\text{TOP}), \quad (6)$$

gdzie **NODE** jest specyfikacją wycinka, taką jak **CONTENTS**, z tym że obejmującą dokładnie wszystkie pola elementu struktury. W powyższym przykładzie **CARD** jest zmienną, w której przechowuje się całe elementy, jak w (1). Jeżeli na element struktury składa się c słów, to notacja (6) jest skrótem ciągu c przypisów:

$$\text{CONTENTS}(\text{LOC}(\text{CARD}) + j) \leftarrow \text{CONTENTS}(\text{TOP} + j), \quad 0 \leq j < c. \quad (7)$$

Trzeba zdawać sobie sprawę z różnic między językiem asemblera a notacją wykorzystywaną do zapisu algorytmów. Asembler jest bliższy językowi maszynowemu, dlatego symbole w programie w języku MIXAL oznaczają adresy, a nie wartości. Z tego powodu w lewej kolumnie programu (5) symbol **TOP** oznacza naprawdę *adres*, pod którym w pamięci przechowywany jest wskaźnik do karty leżącej na wierzchu. Jednak w (6) i (7), a także w komentarzach w (5), **TOP** oznacza *wartość* będącą adresem karty leżącej na wierzchu. Początkujący programiści często mają problemy z wyczuciem różnicy między assemblerem a językiem wysokiego poziomu, dlatego nalegamy na rozwiążanie ćwiczenia 7. Pozostałe ćwiczenia też są warte uwagi jako powtórzenie wprowadzonych konwencji dotyczących notacji.

ĆWICZENIA

1. [04] Spójrzmy na diagram (3). Jaka jest wartość:
(a) **SUIT(NEXT(TOP))**; (b) **NEXT(NEXT(NEXT(TOP)))**?
2. [10] W tekście było powiedziane, że w wielu przypadkach **CONTENTS(LOC(V)) = V**. Kiedy zachodzi **LOC(CONTENTS(V)) = V**?
3. [11] Podaj algorytm, który niweluje skutki wykonania algorytmu A, tj. zdejmuje wierzchnią kartę z kupki (jeżeli na kupce leży przynajmniej jedna karta) i przypisuje na **NEWCARD** adres zdjętej karty.
4. [18] Podaj algorytm podobny do algorytmu A, który wsunie *zakrytą* kartę na spód kupki. (Kupka może być pusta).

- 5. [21] Podaj algorytm, który niweluje skutki wykonania algorytmu z ćwiczenia 4. Algorytm ten, jeżeli kupka jest niepusta, a na jej spodzie leży zakryta karta, usuwa kartę ze spodu kupki i do zmiennej NEWCARD przypisuje dowiązanie do usuniętej karty. (Ten algorytm czasami nazywa się „oszukiwaniem”).
6. [06] Założmy, że w przykładzie z kartami CARD oznacza zmienną, której wartością jest cały element struktury danych, jak w (6). Operacja $CARD \leftarrow NODE(TOP)$ przypisuje polom zmiennej CARD odpowiednie wartości pól elementu z wierzchu kupki. Który z poniższych napisów po wykonaniu operacji oznacza kolor wierzchniej karty? (a) $SUIT(CARD)$; (b) $SUIT(LOC(CARD))$; (c) $SUIT(CONTENTS(CARD))$; (d) $SUIT(TOP)$?
- 7. [04] W przykładzie (5) zmienna wskaźnikowa TOP jest przechowywana w słowie maszyny MIX, którego asemblerową nazwą jest TOP. Niech struktura ma taki układ pól jak w (1). Który z poniższych ciągów rozkazów wstawia NEXT(TOP) do rejestru A? Wyjaśnij, dlaczego drugi ciąg jest niepoprawny.

a) LDA TOP(NEXT)

b) LD1 TOP
LDA 0,1(NEXT)

- 8. [18] Napisz program na komputer MIX odpowiadający algorytmowi B.
9. [23] Napisz program na komputer MIX, który wypisuje nazwy kart leżących na kupce, zaczynając od karty na wierzchu. Każda nazwa powinna być wypisana w osobnym wierszu. Nazwy zakrytych kart należy umieścić w nawiasach.

2.2. LISTY LINIOWE

2.2.1. Stosy, kolejki i kolejki dwustronne

Zazwyczaj dane mają strukturę bardzo bogatą, więc w komputerze chcemy reprezentować jedynie jej fragment. Na przykład w poprzednim rozdziale każdy element struktury danych reprezentującej karty do gry miał pole **NEXT** oznaczające następną kartę w kupce, ale nie można było w prosty sposób stwierdzić, jaka była *poprzednia* karta w kupce, ani sprawdzić, w której kupce znajduje się dana karta. I oczywiście pominęliśmy całą masę cech, którymi odznaczają się *prawdziwe* karty do gry: wzór na koszulce, związki z innymi obiekty w pomieszczeniu, w którym karty się znajdują, własności poszczególnych molekuł karty itp. Można sobie wyobrazić programy, w których zainteresowani bylibyśmy reprezentowaniem powyższych cech, ale oczywiście nigdy nie chcemy przechowywać *pełnego* opisu reprezentowanych obiektów. W istocie w większości gier karcianych wymienione cechy byłyby zupełnie zbędne, a w wielu przypadkach nawet pole **TAG** okazałoby się niepotrzebne.

Musimy zdecydować, *jaki fragment* struktury chcemy reprezentować w pamięci maszyny i na dostępie do których informacji najbardziej nam zależy. By podjąć tę decyzję, musimy wiedzieć, jakie operacje będą wykonywane na danych. Z tego powodu dla każdego problemu opisanego w tym rozdziale rozważamy *nie tylko strukturę danych, ale także zbiór operacji na niej*. Projekt reprezentacji struktury danych w pamięci maszyny zależy od jej funkcjonalności oraz właściwości samych danych. W istocie równy nacisk na funkcjonalność i formę jest punktem wyjścia wielu problemów związanych z projektowaniem w ogóle.

By zilustrować to stwierdzenie, rozważmy analogiczny problem pojawiający się przy projektowaniu sprzętu komputerowego. Spotykamy różne rodzaje pamięci elektronicznej: „pamięć o dostępie swobodnym” (na przykład pamięć główna komputera **MIX**), „pamięć tylko do odczytu” (zawierająca informacje, których nie trzeba zmieniać), „pamięć masowa” (na przykład jednostki pamięci dyskowej podłączanej do komputera **MIX**, odznaczające się długim czasem dostępu, ale dużą pojemnością), „pamięć asocjacyjna” (pamięć, w której informacje adresuje się za pomocą ich wartości, a nie adresu) itd. Zamierzona funkcjonalność tych różnych typów pamięci jest tak istotna, że wyznacza ich nazwy. Wszystkie te urządzenia są jakiegoś typu „pamięciami”, ale cel, jakiemu mają służyć, ma zasadniczy wpływ na ich konstrukcję i cenę.

Lista liniowa jest ciągiem $n \geq 0$ elementów $X[1], X[2], \dots, X[n]$, których istotne własności strukturalne obejmują jedynie względową pozycję elementu w porządku liniowym. W takich strukturach danych zwracamy uwagę wyłącznie na to, że przy $n > 0$ $X[1]$ jest pierwszym elementem, a $X[n]$ ostatnim. Do tego, jeśli $1 < k < n$, to k -ty element $X[k]$ leży za $X[k - 1]$ i przed $X[k + 1]$.

Przykładowy zbiór operacji na liście obejmuje:

- i) Uzyskiwanie dostępu do k -tego elementu listy w celu odczytania i/lub zmodyfikowania zawartości jego pól.
- ii) Wstawianie nowego elementu bezpośrednio przed lub po k -tym elemencie.

- iii) Usuwanie k -tego elementu.
- iv) Łączenie dwóch lub więcej list liniowych w jedną listę liniową.
- v) Rozbijanie listy liniowej na dwie lub więcej list liniowych.
- vi) Tworzenie kopii listy liniowej.
- vii) Wyznaczanie liczby elementów na liście.
- viii) Sortowanie elementów listy w porządku rosnącym względem wartości ustalonego pola.
- ix) Znajdowanie elementu listy o zadanej wartości ustalonego pola.

W operacjach (i), (ii) i (viii) najważniejsze są przypadki szczególne $k = 1$ i $k = n$, ponieważ w wielu sytuacjach najłatwiej dostać się do pierwszego lub ostatniego elementu listy (łatwiej niż do pozostałych). W tym rozdziale nie będziemy omawiać operacji (viii) i (ix), ponieważ zajmujemy się nimi odpowiednio w rozdziale 5 i 6.

Program komputerowy rzadko kiedy korzysta ze wszystkich dziesięciu wymienionych operacji w całej ich ogólności. Z tego powodu istnieje wiele sposobów reprezentowania list liniowych, z których każdy pozwala wygodnie realizować pewien podzbiór wymienionych operacji. Trudno zaprojektować jedną reprezentację, dla której wszystkie operacje byłyby wydajne. Na przykład uzyskiwanie dostępu do k -tego elementu dłużej listy dla dowolnego k jest względnie trudne do zrealizowania, jeżeli jednocześnie wstawiamy i usuwamy elementy ze środka listy. Z tego powodu wyodrębnia się kilka rodzajów list liniowych, różniących się dopuszczalnymi operacjami, podobnie jak wyodrębniliśmy kilka rodzajów pamięci elektronicznych.

Często spotykamy listy liniowe, dla których wstawianie, usuwanie i dostęp do wartości dotyczy w głównej mierze pierwszego lub ostatniego elementu. Mamy dla nich nawet specjalne nazwy:

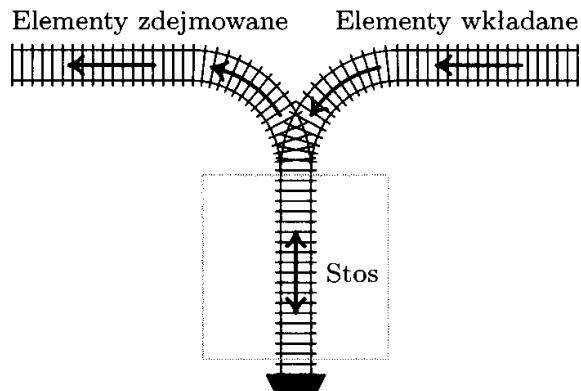
Stos to lista liniowa, dla której operacje wstawiania i usuwania (a zazwyczaj również odczytu) elementu dotyczą tylko jednego końca listy.

Kolejka to lista liniowa, dla której operacje wstawiania dotyczą jednego końca, a operacje usuwania (i zazwyczaj operacje odczytu) drugiego końca.

Kolejka dwustronna to lista liniowa, dla której wszystkie operacje wstawiania oraz usuwania (i zazwyczaj operacje odczytu) dotyczą dowolnego końca listy.

Kolejka dwustronna jest bardziej uniwersalna niż stos lub zwykła kolejka. Kolejka dwustronna ma wiele wspólnego z talią kart. Wyróżniamy ponadto kolejki dwustronne o ograniczonym wyjściu lub o ograniczonym wejściu, w których odpowiednio usuwanie i wstawianie może następować tylko z jednego końca.

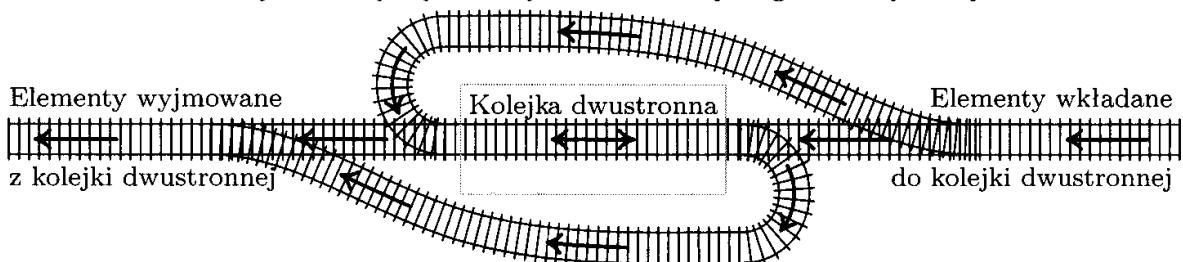
Słowa „kolejka” używa się czasem w szerszym znaczeniu, obejmującym dowolne listy, na których można wykonywać operacje wstawiania i usuwania elementu. My jednak będziemy trzymać się węższego znaczenia, opierając się na analogii z kolejką osób stojących do kas.



Rys. 1. Reprezentacja stosu w postaci bocznicy kolejowej.

Czasami wygodnie jest myśleć o stosie, wyobrażając sobie bocznicę kolejową. Tę analogię zaproponował E. W. Dijkstra (zobacz rysunek 1). Podobne przedstawienie kolejek dwustronnych widać na rysunku 2.

Ten tor jest nieczynny w kolejce dwustronnej o ograniczonym wejściu



Ten tor jest nieczynny w kolejce dwustronnej o ograniczonym wyjściu

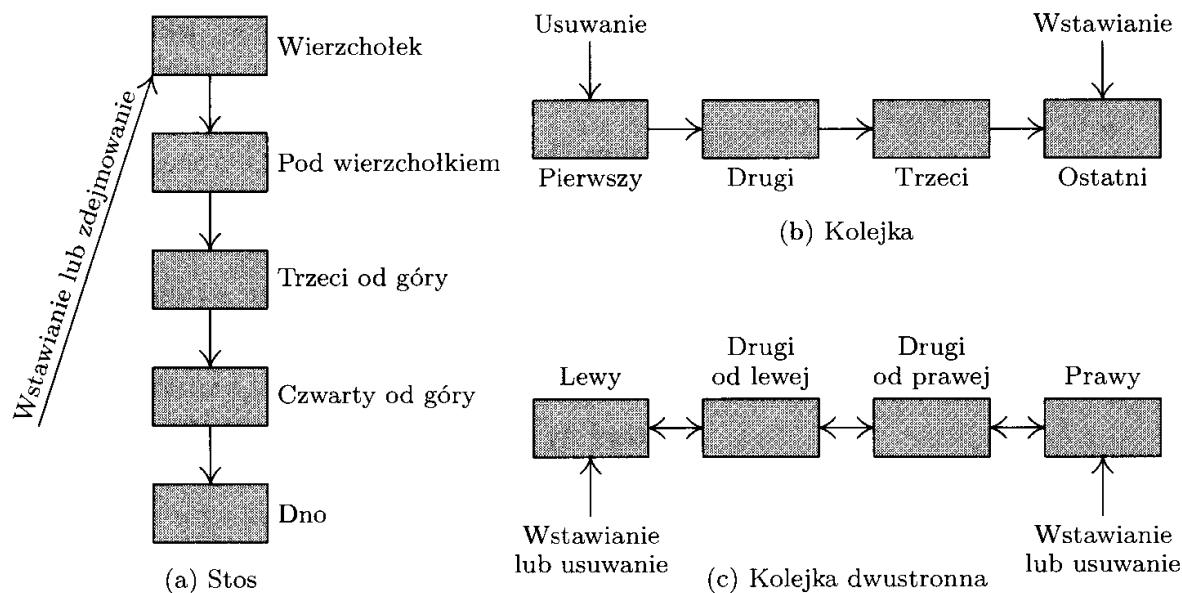
Rys. 2. Kolejka dwustronna jako bocznica kolejowa.

Ze stosu usuwamy zawsze element „najmłodszy”, tzn. ten, który został wstawiony jako ostatni. Z kolejkami jest odwrotnie: usuwamy zawsze element „najstarszy” – elementy opuszczają kolejkę w tym samym porządku, w jakim zostały do niej włożone.

Wiele osób niezależnie odkryło znaczenie stosów i kolejek, nadając im różne nazwy. W literaturze można spotkać następujące określenia stosu*: *push-down lists, reversion storages, cellars, nesting stores, piles, last-in-first-out lists, LIFO* (listy ostatni-wchodzi-pierwszy-wychodzi). Kolejki nazywano *circular stores, first-in-first-out lists, FIFO* (listy pierwszy-wchodzi-pierwszy-wychodzi), a nawet *yo-yo lists*. Przez wiele lat terminy LIFO i FIFO były używane przez księgowych jako nazwy metod inventaryzacji. Kolejki dwustronne o ograniczonym wejściu są nazywane w angielskim *shelf* (regały), a kolejki o ograniczonym wejściu *scrolls* (zwoje) lub *rolls* (szpulki). Ta różnorodność nazw jest ciekawa sama w sobie, ponieważ świadczy o tym, że pomysł stosowania stosów i kolejek jest ważny. Słowa „stos” i „kolejka” powoli stają się standardem. Ze wszystkich wymienionych terminów jedynie *push-down list* pojawia się jeszcze zauważalnie często, zazwyczaj w kontekście teorii automatów.

* Ze względu na brak odpowiedników w języku polskim, zdecydowaliśmy się na pozostawienie ich w oryginalnym brzmieniu (przyp. red.).

Stosy dosyć często spotykamy w praktyce. Możemy na przykład przeglądać zbiór danych i przechowywać listę specjalnych sytuacji lub rzeczy odłożonych na później. Gdy uporamy się z całym zbiorem, możemy dokończyć przetwarzanie, zabierając się za listę i usuwając jej elementy, dopóki całkiem jej nie opróżnimy. (Problem „punktu siodłowego”, ćwiczenie 1.3.2–10, jest przykładem zastosowania tej metody). Taką listę można realizować zarówno jako stos, jak i kolejkę, ale zazwyczaj stos jest wygodniejszy. Rozwiązuje jakiś problem, każdy z nas buduje w pamięci „stos”: jeden problem prowadzi do następnego, a ten do jeszcze następnego. Układamy na stosie problemy i podproblemy, usuwając je, gdy są rozwiązane. Podobny charakter ma proces wchodzenia do i wychodzenia z podprogramów podczas wykonania programu. Stosy przydają się szczególnie przy przetwarzaniu języków o zagnieżdżonej strukturze, jak języki programowania, wyrażenia arytmetyczne oraz niemieckie „Schachtelsätze”. W ogólności stosy pojawiają się w związku z algorytmami jawnie lub niejawnie wykorzystującymi rekursję. Tego typu związki dokładnie omówimy w rozdziale 8.



Rys. 3. Trzy ważne rodzaje list liniowych.

Gdy mówimy o stosach, używamy zazwyczaj specyficznej dla nich terminologii: *wkładamy* element na *wierzch* stosu (element staje się *wierzchołkiem* stosu) lub *zdejmujemy* element ze stosu (zobacz rysunek 3a). *Dnem* stosu nazywamy ten element, którego nie da się zdjąć, dopóki nie zdejmie się wszystkich innych elementów. Mówimy o nim także, że znajduje się na *spodzie* stosu. Po angielsku o wkładaniu elementów na stos mówi się *to push an item down onto a stack*, a o zdejmowaniu ze stosu: *to pop up the stack*. Terminologia ta pochodzi od stosów talerzy umieszczanych na specjalnym sprężynowym mechanizmie popularnym w amerykańskich bufetach. Talerze stoją na okrągłej tacy poruszającej się pionowo na sprężynie w okrągłym szybie zagłębionym w blat. Dostawienie talerza powoduje większe obciążenie tacy, a co za tym idzie, opuszczenie się całego stosu w głąb kontuaru (*push down*). Zdjęcie talerza powoduje nieznaczne

podniesienie się stosu (*pop up*). Natomiast w pamięci komputera nic nie jest przesuwane. Elementy są dodawane na wierzch, jak w stosie pudełek stojących jedno na drugim.

Gdy mówimy o kolejkach, wygodnie posługiwać się terminami *początek* i *koniec* kolejki. Elementy dostawiamy na koniec kolejki, a wyjmujemy te, które znajdują się na początku (zobacz rysunek 3b). W przypadku kolejek dwustronnych wygodnie mówić o *lewym* i *prawym* końcu (zobacz rysunek 3c). Czasami pojęcia wierzchołka, dna, początku i końca stosuje się także do kolejek dwustronnych, ale nie ma utartej konwencji mówiącej, czy wierzchołek, początek itp. znajduje się z lewej, czy prawej strony.

W opisach algorytmów używamy słów kojarzących się z opisaną terminologią: „góra-dół” dla stosów, „czekanie w kolejce” dla kolejek i „lewo-prawo” dla kolejek dwustronnych.

Przyda się jeszcze trochę oznaczeń związanych ze stosami i kolejkami:

$$A \Leftarrow x, \quad (1)$$

Powyższy napis oznacza, że (a) gdy A jest stosem, wartość x jest wkładana na wierzch stosu; (b) gdy A jest kolejką, wartość x jest dostawiana na koniec kolejki. Analogicznie notacja

$$x \Leftarrow A \quad (2)$$

oznacza przypisanie do zmiennej x wartości wierzchołka stosu lub pierwszego (tj. znajdującego się na początku) elementu kolejki i usunięcie go z A . Znaczenie napisu (2) jest nieokreślone, gdy A jest pustą kolejką lub pustym stosem (tj. gdy A nie zawiera żadnych elementów).

Jeśli A jest niepustym stosem, to piszemy

$$\text{top}(A) \quad (3)$$

na oznaczenie jego wierzchołka.

ĆWICZENIA

1. [06] Kolejka dwustronna o ograniczonym wejściu jest listą liniową, do której można dostawiać elementy tylko z jednego końca, ale usuwać z obu. Oczywiście taka kolejka może posłużyć za stos lub zwykłą kolejkę, jeżeli konsekwentnie usuwać elementy wyłącznie z ustalonego końca. Czy kolejka dwustronna o ograniczonym wejściu może posłużyć za stos lub za zwykłą kolejkę?
- ▶ 2. [15] Wyobraź sobie cztery wagony kolejowe o numerach (od lewej) 1, 2, 3, 4, ustawione po stronie „wejściowej” na rysunku 1. Wykonujemy następujący ciąg operacji (zgodnie z kierunkiem strzałek na rysunku oraz przy założeniu, że jeden wagon nie może „przeskoczyć” nad drugim): (i) przetocz wagon 1 na stos; (ii) przetocz wagon 2 na stos; (iii) przetocz wagon 2 na wyjście; (iv) przetocz wagon 3 na stos; (v) przetocz wagon 4 na stos; (vi) przetocz wagon 4 na wyjście; (vii) przetocz wagon 3 na wyjście; (viii) przetocz wagon 1 na wyjście.

W wyniku wykonania tych operacji początkowy układ wagonów 1234, zmienił się na 2431. Celem tego ćwiczenia jest zbadanie, jakie permutacje można w ten sposób uzyskać za pomocą stosów, kolejek oraz kolejek dwustronnych.

Czy sześć wagonów 123456 można ustawić w kolejności 325641? A w układzie 154623? (Jeśli tak, to jak?)

3. [25] Operacje (i)–(viii) z poprzedniego ćwiczenia można zapisać zwięzle jako SSXSSXXX, gdzie S oznacza „przetocz wagon na stos”, a X oznacza „przetocz wagon ze stosu na wyjście”. Niektóre układy symboli S i X nie mają sensu, bo na odpowiednim torze może akurat nie być wagonów. Na przykład ciąg SXXSSXXS nie ma sensu, ponieważ na początku stos jest pusty.

Ciąg złożony z symboli S i X nazywamy *dopuszczalnym*, jeżeli zawiera n symboli S i n symboli X oraz ma sens wg powyższej interpretacji. Sformułuj regułę pozwalającą w łatwy sposób odróżnić ciągi dopuszczalne od niedopuszczalnych. Pokaż ponadto, że nie istnieją dwa różne ciągi dopuszczalne, które dawałyby tę samą permutację wagonów.

4. [M34] Znajdź prosty wzór na liczbę a_n permutacji n elementów, które można uzyskać za pomocą stosu i metody podanej w ćwiczeniu 2.

► **5.** [M28] Pokaż, że permutację $p_1 p_2 \dots p_n$ można uzyskać z $12 \dots n$ za pomocą stosu wtedy i tylko wtedy, gdy nie istnieją indeksy $i < j < k$, takie że $p_j < p_k < p_i$.

6. [00] Rozważ problem z ćwiczenia 2 dla kolejki zamiast stosu. Jakie permutacje możemy uzyskać z $12 \dots n$ za pomocą kolejki?

► **7.** [25] Rozważ problem z ćwiczenia 2 dla kolejki dwustronnej zamiast dla stosu.

(a) Znajdź permutację ciągu 1234, którą można uzyskać za pomocą kolejki dwustronnej o ograniczonym wejściu, ale nie za pomocą kolejki dwustronnej o ograniczonym wyjściu.
 (b) Znajdź permutację ciągu 1234, którą można otrzymać za pomocą kolejki dwustronnej o ograniczonym wyjściu, ale nie za pomocą kolejki dwustronnej o ograniczonym wejściu. [Wniosek z (a) i (b): istnieje istotna różnica między kolejkami dwustronnymi o ograniczonym wejściu i o ograniczonym wyjściu]. (c) Znajdź permutację ciągu 1234, której nie można uzyskać ani za pomocą kolejki dwustronnej o ograniczonym wejściu, ani o ograniczonym wyjściu.

8. [22] Czy istnieją permutacje ciągu $12 \dots n$, których nie można uzyskać za pomocą kolejek dwustronnych (bez ograniczeń wejścia i wyjścia)?

9. [M20] Niech b_n oznacza liczbę permutacji n elementów, które można otrzymać za pomocą kolejki dwustronnej o ograniczonym wejściu. (Zauważ, że zgodnie z ćwiczeniem 7, wartość $b_4 = 22$). Pokaż, że b_n jest także liczbą permutacji n elementów, które można otrzymać za pomocą kolejki dwustronnej o ograniczonym wyjściu.

10. [M25] (Zobacz ćwiczenie 3) Niech S, Q i X oznaczają odpowiednio operacje wstawiania elementu z lewej strony, wstawiania elementu z prawej strony oraz wyjmowania elementu z lewej strony z kolejki dwustronnej o ograniczonym wyjściu. Na przykład ciąg operacji QQXSXSXX zamienia ustawienie elementów 1234 na ustawienie 1342. Ciąg operacji SXQSXSXX daje tę samą permutację.

Wymyśl sposób zdefiniowania pojęcia *dopuszczalności* ciągu złożonego z symboli S, Q i X, tak by spełniony był warunek: *każda permutacja zbioru n -elementowego, którą można uzyskać za pomocą kolejki dwustronnej o ograniczonym wyjściu, odpowiada dokładnie jednemu ciągowi dopuszczalnemu*.

► **11.** [M40] Z ćwiczeń 9 i 10 wnioskujemy, że b_n jest liczbą dopuszczalnych ciągów długości $2n$. Znajdź postać zamkniętą funkcji tworzącej $\sum_{n \geq 0} b_n z^n$.

12. [HM34] Oblicz asymptotyczne wartości a_n i b_n z ćwiczeń 4 i 11.

13. [M48] Ile permutacji n elementów można otrzymać za pomocą nieograniczonej kolejki dwustronnej? [Rosenstiehl i Tarjan w *J. Algorithms* 5 (1984), 389–390, przedstawili algorytm, który w liniowej liczbie kroków stwierdza, czy w ten sposób można uzyskać daną permutację].

► **14.** [26] Przypuśćmy, że jedynymi strukturami danych, których wolno używać, są stosy. W jaki sposób wydajnie zaimplementować kolejkę za pomocą dwóch stosów?

2.2.2. Tablice

Najprostszą i najbardziej naturalną metodą przechowywania listy liniowej w pamięci komputera jest umieszczenie jej elementów jeden za drugim w kolejnych lokacjach pamięci. Zachodzi wówczas warunek

$$\text{LOC}(\mathbf{X}[j+1]) = \text{LOC}(\mathbf{X}[j]) + c,$$

gdzie c jest liczbą słów przypadających na element listy. (Zazwyczaj $c = 1$. Jeśli $c > 1$, to czasem wygodnie jest rozbić listę na c „równoległych” list, tak by k -te słowo elementu $\mathbf{X}[j]$ było przechowywane w stałej odległości od pierwszego słowa elementu $\mathbf{X}[j]$, zależnej od k . Będziemy jednak zakładać, że na pojedynczy element składa się spójna grupa c słów). W ogólności

$$\text{LOC}(\mathbf{X}[j]) = L_0 + cj, \quad (1)$$

gdzie L_0 jest stałą zwaną *adresem bazowym*, lokacją „sztucznego” elementu $\mathbf{X}[0]$.

Jest to tak oczywista i znana technika, że nie ma potrzeby dłużej się nad nią rozwodzić. W dalszej części rozdziału opiszemy jednak o wiele bardziej wyrafinowane metody reprezentowania list, więc rozpoczęcie od zbadania prostego przypadku wydaje się dobrym pomysłem. Należy zdać sobie sprawę z ograniczeń i zalet tablicowych struktur danych.

Wykorzystanie tablicy jest bardzo wygodne w przypadku *stosu*. Utrzymujemy po prostu zmienną T nazywaną *wskaźnikiem stosu*. Gdy stos jest pusty, mamy $T = 0$. Wkładamy element Y na stos, wykonując

$$T \leftarrow T + 1; \quad \mathbf{X}[T] \leftarrow Y. \quad (2)$$

Gdy stos jest niepusty, możemy przypisać wierzchołek stosu do Y i zdjąć go, wykonując sekwencję przeciwną do (2):

$$Y \leftarrow \mathbf{X}[T]; \quad T \leftarrow T - 1. \quad (3)$$

(Z powodu (1) wygodniej zazwyczaj przechowywać wartość cT , a nie T . Widać, gdzie i jakie modyfikacje należałoby wprowadzić, będącym zatem kontynuować opis, zakładając, że $c = 1$).

Reprezentacja *kolejki* lub *kolejki dwustronnej* jest bardziej skomplikowana. W najprostszym rozwiązaniu utrzymuje się dwa wskaźniki F i R^* . $F = R = 0$ oznacza, że kolejka jest pusta. Wkładanie elementu na koniec kolejki realizujemy, wykonując:

$$R \leftarrow R + 1; \quad \mathbf{X}[R] \leftarrow Y. \quad (4)$$

* F (od *front*) – oznacza tu początek kolejki, a R (od *rear*) – koniec kolejki (przyp. tłum.).

Usuwanie elementu z początku kolejki (F wskazuje na komórkę tuż przed pierwszym elementem) realizujemy, wykonując:

$$F \leftarrow F + 1; \quad Y \leftarrow X[F]; \quad \text{jeśli } F = R, \text{ to } F \leftarrow R \leftarrow 0. \quad (5)$$

Zauważmy jednak, co się może stać. Jeśli R zawsze wskazuje dalej niż F (czyli gdy zawsze w kolejce jest przynajmniej jeden element), to nasza reprezentacja wykorzystuje komórki o adresach $X[1], X[2], \dots, X[1000], \dots$, ad infinitum, co jest okropnym marnotrawstwem pamięci. Z prostej metody (4) i (5) należy zatem korzystać jedynie wtedy, gdy wiadomo, że F często „dogania” R , na przykład gdy wszystkie operacje usuwania pojawiają się w seriach powodujących opróżnienie całej kolejki.

By obejść problem kolejki „przetaczającej się” przez pamięć, możemy zarezerwować M elementów $X[1], \dots, X[M]$ niejawnie ustawionych w cykl, w którym po $X[M]$ występuje $X[1]$. Wtedy operacje (4) i (5) przybierają postać

$$\text{jeśli } R = M, \text{ to } R \leftarrow 1, \text{ w przeciwnym razie } R \leftarrow R + 1; \quad X[R] \leftarrow Y. \quad (6)$$

$$\text{jeśli } F = M, \text{ to } F \leftarrow 1, \text{ w przeciwnym razie } F \leftarrow F + 1; \quad Y \leftarrow X[F]. \quad (7)$$

W istocie z podobnym rozwiązaniem spotkaliśmy się w punkcie 1.4.4 przy okazji omawiania buforowania operacji wejścia-wyjścia.

Do tej pory milcząco zakładaliśmy, że nie stanie się nic złego. To bardzo nierealistyczne założenie. Przyjmowaliśmy, że w momencie usuwania elementu z kolejki lub stosu w strukturze danych znajduje się przynajmniej jeden element. Metoda (6) i (7) dopuszcza, by w kolejce znajdowało się co najwyżej M elementów, a metody (2)–(5) nie pozwalają, aby zmienne T i R przyjmowały dowolnie duże wartości. Poniżej jest pokazane, jak należy zmodyfikować operacje, by działały bez tak optymistycznych założeń.

$$X \Leftarrow Y \text{ (włożyć na stos): } \begin{cases} T \leftarrow T + 1; \\ \text{jeśli } T > M, \text{ to PRZEPEŁNIENIE;} \\ X[T] \leftarrow Y. \end{cases} \quad (2a)$$

$$Y \Leftarrow X \text{ (usuń ze stosu): } \begin{cases} \text{jeśli } T = 0, \text{ to NIEDOMIAR;} \\ Y \leftarrow X[T]; \\ T \leftarrow T - 1. \end{cases} \quad (3a)$$

$$X \Leftarrow Y \text{ (wstaw do kolejki): } \begin{cases} \text{jeśli } R = M, \text{ to } R \leftarrow 1, \\ \quad \text{w przeciwnym razie } R \leftarrow R + 1; \\ \text{jeśli } R = F, \text{ to PRZEPEŁNIENIE;} \\ X[R] \leftarrow Y. \end{cases} \quad (6a)$$

$$Y \Leftarrow X \text{ (usuń z kolejki): } \begin{cases} \text{jeśli } F = R, \text{ to NIEDOMIAR;} \\ \text{jeśli } F = M, \text{ to } F \leftarrow 1, \\ \quad \text{w przeciwnym razie } F \leftarrow F + 1; \\ Y \leftarrow X[F]. \end{cases} \quad (7a)$$

Zakładamy, że $X[1], \dots, X[M]$ stanowią całą pamięć przeznaczoną na reprezentację listy. PRZEPEŁNIENIE (OVERFLOW) i NIEDOMIAR (UNDERFLOW) oznaczają nadmiar lub brak elementów. Początkowe wartości $F = R = 0$ dla kolejki nie zdadzą

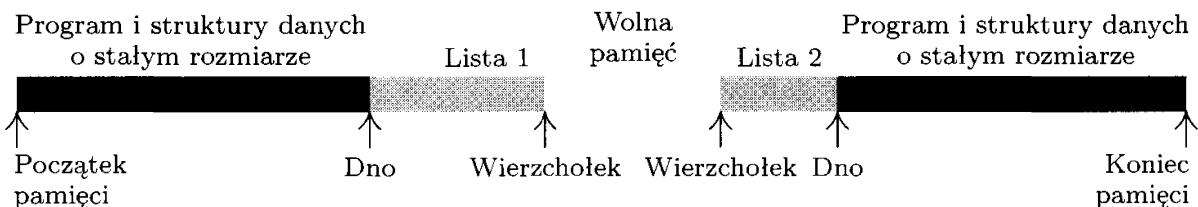
egzaminu, jeśli będziemy korzystać z (6a) i (7a), ponieważ nie zostanie wykryte przepełnienie przy $F = 0$. Powinniśmy zacząć na przykład od $F = R = 1$.

Usilnie namawiamy Czytelnika do zrobienia ćwiczenia 1, które omawia nietrywialne aspekty tego prostego mechanizmu kolejkowania.

Następne pytanie brzmi: „Co należy zrobić, gdy wystąpi PRZEPEŁNIENIE lub NIEDOMIAR?” NIEDOMIAR oznacza, że próbowaliśmy usunąć nieistniejący element i zazwyczaj nie jest to sytuacja błędna, lecz istotna informacja, od której zależy może dalsze wykonanie programu. Możemy chcieć na przykład w pętli usuwać elementy z kolejki aż do wystąpienia NIEDOMIARU. Z drugiej strony PRZEPEŁNIENIE zazwyczaj oznacza błąd – struktura jest już pełna, a my chcielibyśmy włożyć do niej jeszcze jakieś dane. Typowym sposobem postępowania w takiej sytuacji jest zgłoszenie błędu „przekroczenia pojemności dostępnej pamięci” i zakończenie wykonania programu.

Programiści nie lubią poddawać się w przypadku PRZEPEŁNIENIA, szczególnie gdy tylko jedna lista się przepełnia, a inne wciąż dysponują wolnym miejscem. W powyższym opisie mówiliśmy w zasadzie o programie posługującym się jedną listą. W praktyce często spotykamy jednak sytuacje, w których program utrzymuje wiele stosów o dynamicznie zmieniających się rozmiarach. W takich przypadkach nie chcemy narzucać ograniczeń na rozmiar każdego stosu, ponieważ trudno je wyznaczyć. A jeśli już dla każdego stosu ustalimy maksymalny rozmiar, to okaże się, że rzadko kiedy wszystkie stosy zapełniają się równomiernie.

Jeżeli w programie występują dwie listy o zmieniających się rozmiarach, to możemy je tak ulokować w pamięci, by rosły w przeciwnych kierunkach, zajmując kolejne lokacje dzielącego je obszaru pamięci:



Lista 1 rośnie w prawo, a lista 2 w lewo (elementy na liście 2 przechowujemy w odwrotnym porządku). PRZEPEŁNIENIE nie wystąpi, dopóki sumaryczny rozmiar obu list nie przekroczy rozmiaru dostępnej pamięci. Listy mogą rosnąć niezależnie od siebie, zatem maksymalny rozmiar każdej z nich znacznie przewyższa połowę dostępnej pamięci. Taki sposób zagospodarowania pamięci spotyka się bardzo często.

Łatwo się jednak przekonać, że nie da się w analogiczny sposób umieścić w pamięci trzech lub więcej list o zmiennych rozmiarach, tak by (a) PRZEPEŁNIENIE występowało jedynie wtedy, gdy łączny rozmiar list przekroczy rozmiar dostępnej pamięci i (b) dla każdej listy położenie ostatniego elementu było stałe. Gdy mamy do czynienia z trzema lub więcej listami – a taka sytuacja nie jest niczym nadzwyczajnym – problem przydziału pamięci staje się bardzo palący. Jeśli chcemy zagwarantować warunek (a), musimy zrezygnować z warunku (b), tzn. musimy zezwolić, by ostatnie elementy list zmieniały położenie. Oznacza to, że lokacja L_0 występująca w (1) przestaje być stała; nie możemy już odwoływać się

do elementów struktury za pomocą adresów bezwzględnych – wszystkie odwołania musimy realizować względem adresu bazowego L_0 . W przypadku komputera MIX kod realizujący załadowanie I-tego elementu do rejestru A musimy zmienić z

LD1 I na przykład na LDA $L_0, 1$ LD1 I
LDA BASE(0:2)
STA *+1(0:2)
LDA *, 1 (8)

gdzie BASE zawiera

L ₀	0	0	0
----------------	---	---	---

. Za takie adresowanie względne płacimy oczywiście czasem wykonania. Gdyby jednak MIX realizował „adresowanie pośrednie” (zobacz ćwiczenie 3), to narzut byłby niewielki.

Ważny przypadek szczególny pojawia się wtedy, gdy listy o zmiennym rozmiarze są stosami. W dowolnym momencie interesuje nas dostęp wyłącznie do wierzchołka stosu; bazując na tym założeniu, możemy posłużyć się rozwiązaniem prawie tak wydajnym, jak w przypadku dwóch stosów. Przypuśćmy, że stosów jest n . Niech $\text{BASE}[i]$ i $\text{TOP}[i]$ są zmiennymi wskaźnikowymi obsługującymi i -ty stos i niech każdy element stosu zajmuje jedno słowo. Nowe algorytmy wkładania na stos i zdejmowania ze stosu wyglądają tak:

Wkładanie: $\text{TOP}[i] \leftarrow \text{TOP}[i] + 1;$
 jeśli $\text{TOP}[i] > \text{BASE}[i + 1]$, to PRZEPEŁNIENIE;
 w przeciwnym razie $\text{CONTENTS}(\text{TOP}[i]) \leftarrow Y$. (9)

Zdejmowanie: jeśli $\text{TOP}[i] = \text{BASE}[i]$, to NIEDOMIAR; w przeciwnym
razie $Y \leftarrow \text{CONTENTS}(\text{TOP}[i])$, $\text{TOP}[i] \leftarrow \text{TOP}[i] - 1$. (10)

`BASE[i + 1]` jest adresem bazowym ($i + 1$)-go stosu. Warunek `TOP[i] = BASE[i]` oznacza, że stos nr i jest pusty.

W (9) PRZEPEŁNIENIE nie stanowi takiego problemu jak wcześniej. Możemy zreorganizować pamięć, dodając trochę miejsca tej liście, która się przepięniła, i ujmując miejsca tym listom, które do końca się nie zapełniły. Narzuca się kilka sposobów takiej reorganizacji. Niektóre z nich mają szczególne znaczenie w przypadku tablicowych list liniowych. Zaczniemy od metody najprostszej, a potem zastanowimy się nad innymi możliwościami.

Załóżmy, że mamy n stosów i że wartościami $\text{BASE}[i]$ i $\text{TOP}[i]$ posługujemy się tak, jak w (9) i (10). Te stosy mają dzielić wspólną pamięć składającą się ze wszystkich lokacji L , gdzie $L_0 < L \leq L_\infty$. (L_0 i L_∞ są stałymi wyznaczającymi liczbę słów pamięci, które mamy do dyspozycji). Początkowo stosy są puste, zatem

$$\text{BASE}[j] = \text{TOP}[j] = L_0 \quad \text{dla } 1 \leq j \leq n. \quad (11)$$

Przypisujemy także $\text{BASE}[n + 1] = L_\infty$, by algorytmy (9) były poprawne dla $i = n$.

Gdy na stosie *i* pojawi się PRZEPEŁNIEŃIE, możliwe są trzy przypadki:

- a) Znajdujemy najmniejsze k , dla którego $i < k \leq n$ i $\text{TOP}[k] < \text{BASE}[k+1]$, jeśli takie k istnieje. Następnie przesuwamy struktury *w górę* o jedną pozycję:

CONTENTS($L + 1$) \leftarrow **CONTENTS**(L) dla $\text{TOP}[k] \geq L > \text{BASE}[i + 1]$.

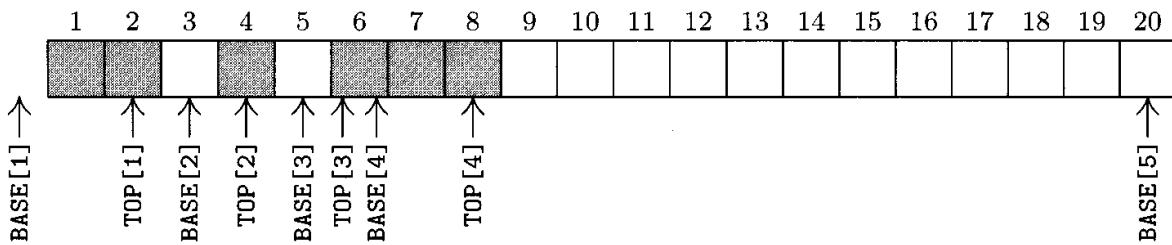
(Wartości zmiennej L muszą maleć, inaczej zamażemy informacje przechowywane w strukturze! Może się zdarzyć, że $\text{TOP}[k] = \text{BASE}[i+1]$; w takim przypadku nie musimy niczego przemieszczać). Na koniec przypisujemy $\text{BASE}[j] \leftarrow \text{BASE}[j] + 1$ i $\text{TOP}[j] \leftarrow \text{TOP}[j] + 1$, dla $i < j \leq k$.

- b) W (a) nie udaje się znaleźć k , ale możemy znaleźć największe k , dla którego $1 \leq k < i$ i $\text{TOP}[k] < \text{BASE}[k+1]$. W tym przypadku przesuwamy struktury *w dół* o jedną pozycję:

$$\text{CONTENTS}(L-1) \leftarrow \text{CONTENTS}(L) \quad \text{dla } \text{BASE}[k+1] < L \leq \text{TOP}[i].$$

(Wartość L musi rosnąć). Następnie przypisujemy $\text{BASE}[j] \leftarrow \text{BASE}[j] - 1$ i $\text{TOP}[j] \leftarrow \text{TOP}[j] - 1$, dla $k < j \leq i$.

- c) Dla wszystkich $k \neq i$ jest $\text{TOP}[k] = \text{BASE}[k+1]$. W takim przypadku oczywiście nie ma miejsca na nowy element i musimy się poddać.



Rys. 4. Przykładowa konfiguracja pamięci po kilku operacjach włożenia i zdania elementu.

Na rysunku 4 jest pokazana konfiguracja pamięci dla przypadku $n = 4$, $L_0 = 0$, $L_\infty = 20$ po poprawnie wykonanych operacjach

$$I_1^* I_1^* I_4^* I_2^* D_1 I_3^* I_1 I_1^* I_2^* I_4 D_2 D_1. \quad (12)$$

(I_j i D_j oznacza włożenie (*insertion*) i zdanie (*deletion*) elementu ze stosu. Gwiazdka oznacza wystąpienie PRZEPEŁNIENIA przy założeniu, że na początku działania programu dla stosów 1, 2 i 3 w ogóle nie zarezerwowano pamięci).

Jasne jest, że wielu przepełnień występujących przy pierwszych operacjach można uniknąć, rozsądnie dobierając początkowe wartości zmiennych określających położenie stosu. Nie jak w (11), gdzie całą pamięć przydzielimy na stos nr n , co jest posunięciem dosyć nierożważnym. Jeśli spodziewamy się, że wszystkie stosy będą tej samej wielkości, możemy na początek przypisać

$$\text{BASE}[j] = \text{TOP}[j] = \left\lfloor \left(\frac{j-1}{n} \right) (L_\infty - L_0) \right\rfloor + L_0 \quad \text{dla } 1 \leq j \leq n. \quad (13)$$

Znajomość charakterystyki konkretnego programu zapewne pomoże w jeszcze lepszym doborze wartości początkowych, jednakże możemy w ten sposób uniknąć co najwyżej stałej liczby przepełnień, a efekt będzie zauważalny jedynie w początkowej fazie działania programu. (Zobacz ćwiczenie 17).

Tę metodę da się poprawiać jeszcze inaczej – po wystąpieniu przepełnienia możemy robić miejsce na więcej niż jeden element. Badaniami nad tą metodą

zajmował się J. Garwick, który zaproponował, by po wystąpieniu przepelnienia zupełnie zreorganizować układ stosów w pamięci, w sposób wyznaczony przez zmianę rozmiaru każdego stosu od ostatniej reorganizacji. Jego algorytm korzysta z pomocniczej tablicy $\text{OLDTOP}[j]$, $1 \leq j \leq n$, w której przechowuje się wartości $\text{TOP}[j]$ sprzed ostatniego przydziału pamięci. Początkowo zmienne ustaviamy tak, jak w naszym dotychczasowym rozwiązaniu oraz $\text{OLDTOP}[j] = \text{TOP}[j]$. Algorytm wygląda następująco:

Algorytm G (*Ponowne przydzielanie pamięci na tablice*). Założmy, że po wykonaniu operacji z (9) na stosie nr i wystąpiło PRZEPEŁNIENIE. W wyniku wykonania algorytmu G albo stwierdzimy faktyczne przekroczenie pojemności pamięci, albo na nowo tak poukładamy struktury w pamięci, że można będzie wykonać $\text{NODE}(\text{TOP}[i]) \leftarrow Y$. (Zauważmy, że wskaźnik stosu $\text{TOP}[i]$ został zwiększyony przed wykonaniem algorytmu G).

- G1.** [Inicjowanie] $\text{SUM} \leftarrow L_\infty - L_0$, $\text{INC} \leftarrow 0$. Wykonaj krok G2 dla $1 \leq j \leq n$. (Wykorzystując zmienną SUM , obliczamy całkowity rozmiar wolnej pamięci, a na zmiennej INC całkowity przyrost rozmiaru struktur od czasu ostatniego przydziału pamięci). Przejdz do kroku G3.
- G2.** [Zbieranie statystyki] $\text{SUM} \leftarrow \text{SUM} - (\text{TOP}[j] - \text{BASE}[j])$. Jeśli $\text{TOP}[j] > \text{OLDTOP}[j]$, to $D[j] \leftarrow \text{TOP}[j] - \text{OLDTOP}[j]$ i $\text{INC} \leftarrow \text{INC} + D[j]$; w przeciwnym razie $D[j] \leftarrow 0$.
- G3.** [Czy pamięć zapełniona?] Jeśli $\text{SUM} < 0$, to nic nie poradzimy.
- G4.** [Obliczanie współczynników przydziału] $\alpha \leftarrow 0.1 \times \text{SUM}/n$, $\beta \leftarrow 0.9 \times \text{SUM}/\text{INC}$. (α i β nie są liczbami całkowitymi, lecz ułamkami, które trzeba obliczać z rozsądnią dokładnością. W kolejnym kroku jest przyznawana wolna pamięć poszczególnym listom: około 10% pamięci zostanie podzielone równo między n list, a pozostałe 90% zostanie podzielone proporcjonalnie do przyrostu rozmiaru struktury od czasu ostatniego przydziału pamięci).
- G5.** [Obliczanie nowych adresów bazowych] Przyjmij $\text{NEWBASE}[1] \leftarrow \text{BASE}[1]$ i $\sigma \leftarrow 0$; następnie dla $j = 2, 3, \dots, n$ przypisz $\tau \leftarrow \sigma + \alpha + D[j-1]\beta$, $\text{NEWBASE}[j] \leftarrow \text{NEWBASE}[j-1] + \text{TOP}[j-1] - \text{BASE}[j-1] + \lfloor \tau \rfloor - \lfloor \sigma \rfloor$ i $\sigma \leftarrow \tau$.
- G6.** [Ponowne upakowanie] $\text{TOP}[i] \leftarrow \text{TOP}[i] - 1$. (To jest faktyczny rozmiar i -tego stosu, do którego nie dołożono jeszcze elementu powodującego przepłnienie. W ten sposób uchronimy się przed przenoszeniem elementu spoza przestrzeni zajmowanej przez i -ty stos). Wykonaj algorytm R (opisany poniżej) i przywróć $\text{TOP}[i] \leftarrow \text{TOP}[i] + 1$. Na koniec $\text{OLDTOP}[j] \leftarrow \text{TOP}[j]$ dla $1 \leq j \leq n$. ■

Zapewne najbardziej interesującą częścią całego algorytmu jest fragment odpowiadający za faktyczną reorganizację. Reorganizacja jest zadaniem niebanalnym, bo niektóre bloki przesuwamy w góre, a niektóre w dół. Trzeba oczywiście dbać, by podczas takich porządków nie zamazać żadnej istotnej informacji.

Algorytm R (*Przemieszczanie tablic*). Dla $1 \leq j \leq n$ informacje znajdujące się pod adresami wyznaczonymi przez $\text{BASE}[j]$ i $\text{TOP}[j]$, zgodnie z opisanymi powyżej konwencjami, są przemieszczane w miejsce wskazywane przez $\text{NEWBASE}[j]$, a wartości $\text{BASE}[j]$ i $\text{TOP}[j]$ są odpowiednio aktualizowane. Ten algorytm opiera się na łatwej do sprawdzenia obserwacji, że bloki przesuwane w dół nie mogą nałożyć się ani na bloki przesuwane w górę, ani na bloki nieruchome.

- R1.** [Inicjowanie] Przyjmij $j \leftarrow 1$.
- R2.** [Znajdowanie początku przesunięcia] (W tym miejscu wszystkie listy spośród list o numerach od 1 do j , które miały zostać przesunięte w dół, zostały przesunięte). Zwiększać j o jeden, aż któryś z następujących warunków będzie prawdziwy:
 - a) $\text{NEWBASE}[j] < \text{BASE}[j]$: w tym przypadku idź do R3;
 - b) $j > n$: w tym przypadku idź do R4.
- R3.** [Przesuwanie w dół] Przyjmij $\delta \leftarrow \text{BASE}[j] - \text{NEWBASE}[j]$. Następnie przyjmij $\text{CONTENTS}(L - \delta) \leftarrow \text{CONTENTS}(L)$, dla $L = \text{BASE}[j] + 1, \text{BASE}[j] + 2, \dots, \text{TOP}[j]$. ($\text{BASE}[j]$ może się równać $\text{TOP}[j]$ i w takim przypadku nic nie musimy robić). Przyjmij $\text{BASE}[j] \leftarrow \text{NEWBASE}[j]$, $\text{TOP}[j] \leftarrow \text{TOP}[j] - \delta$. Wróć do R2.
- R4.** [Znajdowanie początku przesunięcia] (W tym miejscu wszystkie listy spośród list o numerach od j do n , które miały być przesunięte w górę, zostały przesunięte). Zmniejszać j o jeden, aż któryś z następujących warunków będzie prawdziwy:
 - a) $\text{NEWBASE}[j] > \text{BASE}[j]$: w tym przypadku idź do R5;
 - b) $j = 1$: w tym przypadku zakończ algorytm.
- R5.** [Przesuwanie w górę] Przyjmij $\delta \leftarrow \text{NEWBASE}[j] - \text{BASE}[j]$. Następnie wykonaj $\text{CONTENTS}(L + \delta) \leftarrow \text{CONTENTS}(L)$, dla $L = \text{TOP}[j], \text{TOP}[j] - 1, \dots, \text{BASE}[j] + 1$. (Tak jak w kroku R3, może się zdarzyć, że nie musimy tu nic robić). Przyjmij $\text{BASE}[j] \leftarrow \text{NEWBASE}[j]$, $\text{TOP}[j] \leftarrow \text{TOP}[j] + \delta$. Wróć do R4. ■

Zauważmy, że stosu nr 1 nigdy nie trzeba przesuwać. Dlatego na początku powinniśmy umieścić największy stos, jeżeli oczywiście wiemy, który ze stosów będzie największy.

Algorytmy G i R zostały celowo tak napisane, że nie przeskakują fakt, iż

$$\text{OLDTOP}[j] \equiv \text{D}[j] \equiv \text{NEWBASE}[j + 1]$$

dla $1 \leq j \leq n$. Innymi słowy, te trzy zmienne mogą zajmować te same komórki pamięci, ponieważ nigdy nie korzystamy z nich jednocześnie.

Opisaliśmy algorytmy reorganizacji pamięci dla stosów, ale oczywiście można je przystosować do dowolnych struktur wykorzystujących adresowanie względne, których cała reprezentacja mieści się między $\text{BASE}[j]$ a $\text{TOP}[j]$. Listy można wzbogacić o inne wskaźniki (na przykład $\text{FRONT}[j]$ i $\text{REAR}[j]$), co pozwala traktować je jak kolejki lub kolejki dwustronne. Przypadek kolejki rozważamy szczegółowo w ćwiczeniu 8.

Matematyczna analiza algorytmów dynamicznego przydzielenia pamięci jest niezmiernie trudna. Niektóre ciekawe wyniki pojawiają się w ćwiczeniach do tego rozdziału, chociaż w zasadzie muskają jedynie powierzchnię problemu.

By pokazać teorię, którą mimo wszystko *da się* zbudować, rozważmy sytuację, w której na stosie wykonujemy jedynie operacje wstawiania elementów. Operacje usuwania i wstawienia elementu, które znoszą się z punktu widzenia rozmiaru stosu, ignorujemy. Założymy ponadto, że każdy stos wypełnia się danymi z tą samą szybkością. Taką sytuację możemy modelować za pomocą ciągu m operacji włożenia elementu a_1, a_2, \dots, a_m , gdzie a_i jest liczbą całkowitą od 1 do n (oznaczającą operację włożenia na stos elementu a_i). Na przykład ciąg 1, 1, 2, 2, 1 oznacza dwie operacje włożenia na stos 1, następnie dwie operacje włożenia na stos 2 i jedną operację włożenia na stos 1. Możemy przyjąć, że każdy z n^m możliwych ciągów operacji jest jednakowo prawdopodobny. Pytamy, jaka jest średnia liczba przemieszczeń każdego słowa dokonywanych podczas reorganizacji pamięci. Pierwszy algorytm (przydzielający wstępnie całą pamięć na stos nr n) jest analizowany w ćwiczeniu 9. Przekonamy się, że średnia liczba operacji przemieszczenia elementu jest równa

$$\frac{1}{2} \left(1 - \frac{1}{n}\right) \binom{m}{2}. \quad (14)$$

Możemy się zatem spodziewać, że liczba przemieszczeń jest proporcjonalna do *kwadratu* liczby operacji włożenia elementu na stos. Pozostaje to prawdą również wtedy, gdy operacje nie rozkładają się równomiernie na wszystkie stose. (Zobacz ćwiczenie 10).

Morał z tego taki, że potrzeba będzie olbrzymiej liczby przemieszczeń nawet wtedy, gdy liczba elementów struktur będzie zupełnie rozsądna. To jest cena, jaką płacimy za ścisłe upakowanie wielu struktur tablicowych. Nie opracowano żadnej teorii pozwalającej na przeanalizowanie działania algorytmu G i raczej nie wydaje się, by jakikolwiek prosty model był w stanie opisać zachowanie w takim środowisku struktur danych występujących w praktyce. Ćwiczenie 18 gwarantuje jednak, że czas wykonania nie będzie bardzo długi, jeżeli pamięć się za bardzo nie wypełni.

Doświadczenie pokazuje, że gdy pamięć jest wypełniona jedynie w połowie (tj. gdy wolna pamięć stanowi połowę całkowitej pamięci), reorganizacje przeprowadzane przez algorytm G są bardzo skromne. Najważniejszy jest zapewne wniosek, że algorytm zachowuje się dobrze w przypadku „połowicznego zapełnienia” i działa poprawnie w przypadku „prawie całkowitego zapełnienia”.

Przyjrzyjmy się jednak uważniej przypadkowi „prawie całkowitego zapełnienia”. Gdy struktury pokrywają prawie całą dostępną pamięć, algorytm R działa długo. A co gorsze, najwięcej przepełnień występuje tuż przed przekroczeniem dostępnej pojemności pamięci. W większości programów stopień zajętości pamięci zbliży się do dostępnej objętości całkowitej tuż przed jej przekroczeniem. I te programy prawdopodobnie będą marnować koszarne ilości czasu na wykonanie algorytmów G i R, by na koniec zatrzymać się z komunikatem o braku pamięci. By zabezpieczyć się przed takim marnotrawstwem czasu, najlepiej zatrzymać

algorytm G w kroku G3, gdy wartość SUM jest mniejsza od S_{\min} , stałej dobranej przez programistę w celu uniknięcia czasochłonnych reorganizacji. Gdy posługujemy się wieloma tablicowymi strukturami danych o zmiennym rozmiarze, *nie* powinniśmy nastawiać się na wykorzystanie 100% dostępnej pamięci w momencie zgłoszenia błędu braku pamięci.

Wnikliwe badania nad algorytmem G przeprowadzili D. S. Wise i D. C. Watson, *BIT* 16 (1976), 442–450. Zobacz także: A. S. Fraenkel, *Inf. Proc. Letters* 8 (1979), 9–10, od którego pochodzi pomysł dwóch stosów rosnących w przeciwnych kierunkach.

ĆWICZENIA

- ▶ 1. [15] Mamy kolejkę z operacjami (6a) i (7a). Ile elementów można umieścić w kolejce, aby nie wystąpiło PRZEPEŁNIENIE?
- ▶ 2. [22] Uogólnij metodę (6a) i (7a), by można ją było zastosować do dowolnej kolejki dwustronnej o mniej niż M elementach. Innymi słowy, zapisz operacje „usuwania z końca kolejki” i „wstawiania na początek kolejki”.
- ▶ 3. [21] Przypuśćmy, że komputer MIX rozbudowano w następujący sposób. Pole I każdego rozkazu ma postać $8I_1 + I_2$, gdzie $0 \leq I_1 < 8$, $0 \leq I_2 < 8$. W języku asemblera piszemy „OP ADRES, I₁:I₂” lub (jak dotychczas) „OP ADRES, I₂”, gdy $I_1 = 0$. Nowa semantyka nakazuje wykonanie na wartości ADRES „modyfikacji adresu” wyznaczonej przez I₁, następnie „modyfikacji adresu” wyznaczonej przez I₂, a wreszcie wykonanie operacji OP na otrzymanym adresie wynikowym. Modyfikacje adresu są zdefiniowane następująco:

0: $M = A$
 1: $M = A + rI1$
 2: $M = A + rI2$
 ...
 6: $M = A + rI6$
 7: $M = \text{adres wyznaczony przez pola } \text{„ADRES, I}_1:\text{I}_2\text{” z komórką o adresie A. Zabrania się, by pod adresem A było } I_1 = I_2 = 7.$ (To ograniczenie rozważamy w ćwiczeniu 5).

A oznacza adres przed operacją, M oznacza adres po modyfikacjach. We wszystkich przypadkach wynik operacji jest nieokreślony, jeżeli wartość M nie mieści się w dwóch bajtach ze znakiem. Czas wykonania zwiększa się o jedną jednostkę na każde „adresowanie pośrednie” (modyfikacja o kodzie 7).

Przypuśćmy, że lokacja 1000 zawiera „NOP 1000,1:7”, oraz lokacja 1001 zawiera „NOP 1000,2”, a rI1 i rI2 zawierają odpowiednio 1 i 2. Wtedy rozkaz „LDA 1000,7:2” jest równoważny rozkazowi „LDA 1004”, ponieważ

$$1000,7:2 = (1000,1:7),2 = (1001,7),2 = (1000,2),2 = 1002,2 = 1004.$$

a) Korzystając (jeśli to konieczne) z mechanizmu adresowania pośredniego, pokaż, jak uprościć kod po prawej stronie (8), by zaoszczędzić dwa rozkazy na każdym odwołaniu do struktury. O ile szybszy jest nowy kod?

b) Przypuśćmy, że mamy wiele struktur, których adresy bazowe są przechowywane w lokacjach **BASE + 1**, **BASE + 2**, **BASE + 3**, ...; w jaki sposób umieścić I-ty element J-tej

tablicy w rejestrze A za pomocą jednego rozkazu, gdy wartość I mamy w rI1, wartość J w rI2 oraz dysponujemy adresowaniem pośrednim?

c) Jaki skutek ma wykonanie rozkazu „ENT4 X,7”, gdy w polu (3:3) lokacji X jest zero?

4. [25] Przypuśćmy, że komputer MIX rozbudowano jak w ćwiczeniu 3. Pokaż, w jaki sposób za pomocą jednego rozkazu (i pomocniczych stałych):

- Wejść w nieskończoną pętle z powodu nieskończonego adresowania pośredniego.
- Załadować wartość $\text{LINK}(\text{LINK}(x))$ do rejestrów A, gdzie wartość zmiennej wskaźnikowej x znajduje się w polach (0:2) lokacji, której adresem symbolicznym jest X, a wartość $\text{LINK}(x)$ znajduje się w polach (0:2) lokacji x itd. Załóż, że pole (3:3) tych lokacji ma wartość zero.
- Załadować wartość $\text{LINK}(\text{LINK}(\text{LINK}(x)))$ do rejestrów A, przy założeniach jak w (b).
- Załadować zawartość lokacji $rI1 + rI2 + rI3 + rI4 + rI5 + rI6$ do rejestrów A.
- Powiekszyć czterokrotnie wartość rI6.

► 5. [35] Rozszerzenie komputera MIX opisane w ćwiczeniu 3 zawiera nieszczesne ograniczenie zabraniające użycia „7:7” w lokacji adresowanej pośrednio.

- Podaj przykład pokazujący, że bez tego ograniczenia komputer MIX musiałby zawierać duży sprzętowy stos do przechowywania wartości trzybitowych. (To byłoby szalenie kosztowne rozwiązań, nawet dla mitycznego komputera MIX).
- Wyjaśnij, dlaczego przy istniejących ograniczeniach taki stos jest niepotrzebny. Innymi słowy, zaprojektuj algorytm realizujący modyfikację adresów, korzystający z rejestrów sprzętowych o niewielkiej pojemności.
- Zaproponuj łagodniejsze ograniczenie niż to z ćwiczenia 3 dotyczące wykorzystania pola 7:7, które pozwala uniknąć trudności napotkanych w ćwiczeniu 4(c), ale jednocześnie daje się tanio realizować za pomocą sprzętu.

6. [10] Który z poniższych ciągów operacji, przy konfiguracji pamięci jak na rysunku 4, spowoduje przepełnienie lub niedobór:

- I_1 ; (b) I_2 ; (c) I_3 ; (d) $I_4I_4I_4I_4I_4$; (e) $D_2D_2I_2I_2I_2$.

7. [12] Krok G4 algorytmu G zawiera dzielenie przez INC. Czy INC może być w tym miejscu zerem?

► 8. [26] Wyjaśnij, w jaki sposób zmodyfikować (9), (10) oraz algorytm reorganizacji dla przypadku, gdy jedna lub więcej list jest kolejką implementowaną cyklicznie, jak w (6a) i (7a).

► 9. [M27] Za pomocą modelu matematycznego opisanego pod koniec rozdziału udowodnij, że (14) jest spodziewaną liczbą przemieszczeń. (Zauważ, że sekwencja 1, 1, 4, 2, 3, 1, 2, 4, 2, 1 powoduje $0 + 0 + 0 + 1 + 1 + 3 + 2 + 0 + 3 + 6 = 16$ przemieszczeń).

10. [M28] Zmodyfikujmy model matematyczny z ćwiczenia 9 w ten sposób, by niektóre struktury rosły szybciej niż inne: niech p_k będzie prawdopodobieństwem zdarzenia, że $a_j = k$, dla $1 \leq j \leq m$, $1 \leq k \leq n$. Stąd $p_1 + p_2 + \dots + p_n = 1$; w poprzednim ćwiczeniu zajmowaliśmy się przypadkiem szczególnym $p_k = 1/n$ dla wszystkich k . Wyznacz oczekiwany liczbę przemieszczeń dla przypadku ogólnego, analogiczną do (14). Można w ten sposób ustawić względem siebie n list, tak że listy, po których spodziewamy się, że będą duże, ustawiamy na prawo (lub na lewo) od list, które mają być małe. Jakie ustawienie n list minimalizuje oczekiwany liczbę przemieszczeń, przy prawdopodobieństwach p_1, p_2, \dots, p_n ?

11. [M30] Uogólnij rozumowanie z ćwiczenia 9 w ten sposób, że pierwsze t operacji włożenia elementu na każdy ze stosów nie powoduje reorganizacji. Jeśli zatem $t = 2$, to ciąg operacji z ćwiczenia 9 powoduje $0 + 0 + 0 + 0 + 0 + 3 + 0 + 0 + 3 + 6 = 12$ przemieszczeń. Ile wynosi średnia liczba przemieszczeń przy tym założeniu? [To jest przybliżenie zachowania algorytmu w sytuacji, gdy na początku każdemu stosowi przydzielamy t komórek pamięci].

12. [M28] Utrzymywanie dwóch struktur, z których każda rośnie w kierunku adresów zajmowanych przez drugą, ma pewne zalety. Można ilościowo oszacować, jaki zysk przynosi taka organizacja pamięci w porównaniu z przypadkiem, gdy każdą ze struktur przechowujemy w niezależnej ograniczonej przestrzeni. Wykorzystajmy model z ćwiczenia 9 dla $n = 2$. Niech k_1 oznacza liczbę jedynek, a k_2 liczbę dwówek dla każdego z 2^m ciągów operacji a_1, a_2, \dots, a_m . (Liczby k_1 i k_2 są rozmiarami struktur w momencie wypełnienia się pamięci. Algorytm działa w pamięci o rozmiarze $m = k_1 + k_2$, gdy struktury korzystają ze wspólnej przestrzeni, zamiast $2 \max(k_1, k_2)$, gdy struktury są niezależne).

Ile wynosi średnia wartość $\max(k_1, k_2)$?

13. [HM42] Wartość $\max(k_1, k_2)$ badana w ćwiczeniu 12 będzie jeszcze większa, jeżeli będą występować większe wahania rozmiaru struktur spowodowane nie tylko operacjami wstawiania, ale również *usuwania*. Zmodyfikujmy model w ten sposób, że z prawdopodobieństwem p występująca w ciągu operacji wartość a_j jest interpretowana jako operacja zdjęcia elementu ze stosu. Proces kończy się, gdy $k_1 + k_2$ (całkowity rozmiar struktur) osiągnie wartość m . Zdejmowanie elementu z pustego stosu nie powoduje żadnego skutku.

Na przykład, jeśli $m = 4$, to można pokazać, że w momencie zakończenia procesu otrzymujemy następujący rozkład prawdopodobieństwa:

$$(k_1, k_2) = \begin{matrix} (0, 4) & (1, 3) & (2, 2) & (3, 1) & (4, 0) \end{matrix}$$

prawdopodobieństwo $\frac{1}{16 - 12p + 4p^2}, \quad \frac{1}{4}, \quad \frac{6 - 6p + 2p^2}{16 - 12p + 4p^2}, \quad \frac{1}{4}, \quad \frac{1}{16 - 12p + 4p^2}.$

Zatem gdy p rośnie, różnica między k_1 i k_2 również wydaje się rosnąć. Nietrudno pokazać, że w granicy przy p dążącym do jedności rozkład k_1 jest w istocie jednostajny, a graniczna wartość oczekiwana $\max(k_1, k_2)$ jest równa dokładnie $\frac{3}{4}m + \frac{1}{4m}$ [m nieparzyste]. To zachowanie jest całkiem innego od obserwowanego w poprzednim ćwiczeniu (gdzie $p = 0$). Może się okazać, że nie ma to istotnego znaczenia, ponieważ gdy p zbliża się do jedności, czas potrzebny na zakończenie procesu gwałtownie rośnie do nieskończoności. Celem tego ćwiczenia jest zbadanie zależności $\max(k_1, k_2)$ od p i m oraz wyznaczenie wzorów asymptotycznych dla ustalonego p (na przykład $p = \frac{1}{3}$) przy m dążącym do nieskończoności. Szczególnie ciekawy jest przypadek $p = \frac{1}{2}$.

14. [HM43] Uogólnij wynik ćwiczenia 12 na dowolne $n \geq 2$, pokazując, że gdy n jest ustalone, a m rośnie do nieskończoności, wielkość

$$\frac{m!}{n^m} \sum_{\substack{k_1+k_2+\dots+k_n=m \\ k_1, k_2, \dots, k_n \geq 0}} \frac{\max(k_1, k_2, \dots, k_n)}{k_1! k_2! \dots k_n!}$$

asymptotycznie równa się $m/n + c_n \sqrt{m} + O(1)$. Wyznacz stałe c_2, c_3, c_4 i c_5 .

15. [40] Korzystając z metody Monte Carlo, zasymuluj zachowanie algorytmu G przy zmiennych rozkładach operacji wstawiania i usuwania elementu. Co Twoje ekspery-

menty mówią o wydajności algorytmu G? Porównaj jego wydajność z poprzednim algorytmem, który za każdym razem przesuwał struktury o jedną komórkę pamięci.

16. [20] W tekście opisano, w jaki sposób umieścić w pamięci dwa stosy, tak by rozrastały się w przeciwnie strony, zajmując wspólną pamięć. Czy w podobny sposób można zagospodarować pamięć na dwie *kolejki* lub stos i kolejkę, uzyskując równie wydajny algorytm?

17. [30] Niech σ będzie dowolnym ciągiem operacji włożenia i zdjęcia elementu takim jak (12). Niech $s_0(\sigma)$ będzie liczbą przepełnień stosu pojawiających się przy zastosowaniu prostej metody pokazanej na rysunku 4 dla warunków początkowych (11), a $s_1(\sigma)$ niech będzie analogiczną liczbą przepełnień stosu dla warunków początkowych (13). Udowodnij, że $s_0(\sigma) \leq s_1(\sigma) + L_\infty - L_0$.

► **18.** [M30] Pokaż, że całkowity czas wykonania algorytmów G i R dla dowolnego ciągu m operacji włożenia i zdjęcia elementu wynosi $O(m + n \sum_{k=1}^m \alpha_k / (1 - \alpha_k))$, gdzie α_k jest procentem pamięci zajmowanej podczas ostatniej reorganizacji przed operacją nr k , a $\alpha_k = 0$ przed pierwszą reorganizacją. (Jeśli zatem pamięć nie wypełnia się nigdy w stopniu większym niż, powiedzmy, 90%, to każda operacja zabiera co najwyżej $O(n)$ jednostek czasu w sensie zamortyzowanym, niezależnie od całkowitego rozmiaru pamięci). Załóżmy, że $L_\infty - L_0 \geq n^2$.

► **19.** [16] (*Indeksowanie od zera*) Doświadczani programiści wiedzą, że rozsądnie jest numerować elementy listy liniowej od zera, $X[0], X[1], \dots, X[n-1]$, niż bardziej tradycyjnie od jedynki, $X[1], X[2], \dots, X[n]$. Wtedy na przykład adres bazowy L_0 w (1) wskazuje na pierwszy element struktury.

Zastosuj tę konwencję w operacjach wstawiania i usuwania elementu (2a), (3a), (6a) i (7a). Innymi słowy, zmodyfikuj je tak, by elementy listy były przechowywane w tablicy $X[0], X[1], \dots, X[M-1]$, a nie w $X[1], X[2], \dots, X[M]$.

2.2.3. Struktury z dowiązaniami*

Zamiast przechowywać listę liniową w kolejnych lokacjach pamięci, możemy zastosować o wiele bardziej elastyczny schemat, w którym każdy element listy zawiera dowiązanie (wskaźnik) do jej następnego elementu.

Lista tablicowa:

Adres	Zawartość
$L_0 + c:$	Element 1
$L_0 + 2c:$	Element 2
$L_0 + 3c:$	Element 3
$L_0 + 4c:$	Element 4
$L_0 + 5c:$	Element 5

Lista wskaźnikowa:

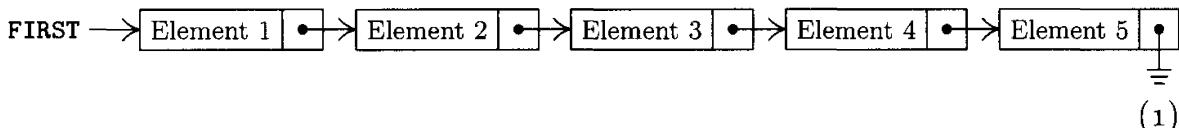
Adres	Zawartość
A:	Element 1
B:	Element 2
C:	Element 3
D:	Element 4
E:	Element 5

A, B, C, D i E są dowolnymi adresami, a Λ jest wskaźnikiem pustym (zobacz podrozdz. 2.1). Program korzystający z listy zapisanej w tablicy powinien mieć dodatkową zmienną lub stałą, której wartość mówiłaby, że struktura zawiera

* Struktury te w polskiej literaturze są również często nazywane strukturami wskaźnikowymi (przyp. tłum.).

pięć elementów. Ta informacja mogłaby być także zakodowana w wartowniku w piątym elemencie lub w następnej lokacji. Program korzystający z listy z dowiązaniami (wskaźnikowej) powinien mieć zmienną zawierającą dowiązanie do A. Do wszystkich elementów listy można uzyskać dostęp za pomocą tego jednego adresu A.

Przypomnijmy sobie konwencję przyjętą w podrozdz. 2.1, zgodnie z którą dowiązania rysowaliśmy w postaci strzałek, ponieważ zazwyczaj nie interesują nas konkretne adresy. Powyższą listę z dowiązaniami możemy zatem narysować tak:



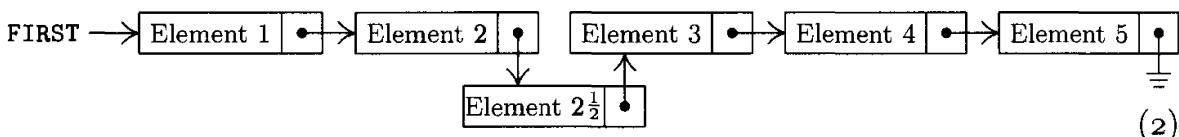
FIRST jest zmienną zawierającą dowiązanie do pierwszego elementu listy (zmiennej wskaźnikowej).

Możemy porównać te dwie metody realizacji struktur danych:

1) Struktury z dowiązaniami wymagają dodatkowych komórek na przechowywanie dowiązań. To czasami może być czynnikiem dominującym. Często okazuje się jednak, że informacja przechowywana w elemencie struktury i tak nie zajmuje całego słowa, zatem bez trudu daje się wygospodarować miejsce na dowiązanie. W wielu sytuacjach jeden element może składać się z wielu pól, zatem jedno dowiązanie przypada na kilka pól elementu struktury (zobacz ćwiczenie 2.5–2). Co ważniejsze, stosując dowiązania, zazwyczaj niejawnie *zyskujemy* na pamięci, ponieważ wspólne części struktur mogą się nakładać. W wielu przypadkach operacje na tablicach nie są tak wydajne, jak na strukturach z dowiązaniemi, chyba że mamy do dyspozycji bardzo dużą pamięć, która w większości pozostaje niewykorzystana. Na końcu poprzedniego rozdziału wyjaśnialiśmy na przykład, dlaczego systemy tam opisane są zawsze nieefektywne, gdy pamięć jest bardzo obciążona.

2) Bardzo łatwo usunąć element listy z dowiązaniemi. By usunąć na przykład element 3, musimy jedynie zmienić dowiązanie związane z elementem 2. Przy sekwencyjnym przydziale pamięci taka operacja zazwyczaj wymaga przemieszczania dużego fragmentu listy w górę do innych lokacji pamięci.

3) Wstawianie elementu w środek listy z dowiązaniemi też jest proste. Na przykład, by wstawić element $2\frac{1}{2}$ do listy (1), musimy jedynie zmienić dwa dowiązania:



Dla porównania, ta operacja byłaby niezmiernie czasochłonna w przypadku dużej tablicy.

4) Odwołania do dowolnych elementów listy są o wiele szybsze w przypadku użycia tablicy. By uzyskać dostęp do k -tego elementu listy zapisanej w tablicy (gdy k jest zmienną) potrzebujemy jedynie stałego czasu, ale w przypadku listy

z dowiązaniami potrzeba k przejść po wskaźnikach, by dotrzeć do właściwego elementu. Użyteczność metody wykorzystującej dowiązania wynika zatem z faktu, że w większości przypadków korzystamy z elementów struktury w pewnej ustalonej kolejności, a nie w sposób przypadkowy. Jeśli już musimy odwołać się na przykład do elementu środkowego lub ostatniego, to powinniśmy utrzymywać dodatkowe wskaźniki do nich.

5) O wiele łatwiej połączyć dwie listy z dowiązaniami, a także rozbić jedną listę na dwie, które później mogą być modyfikowane niezależnie.

6) Listy z dowiązaniami są punktem wyjścia dla dalece bardziej skomplikowanych struktur. Możemy dysponować na przykład zmienną liczbą list o zmiennej długości – każdy element „głównej” listy może być początkiem listy „podzielonej”; elementy struktury mogą mieć wiele dowiązań i być połączone równocześnie w różnych porządkach, odpowiadających różnym listom itd.

7) Proste operacje, jak przeglądanie kolejnych elementów listy, są na wielu komputerach szybsze dla list sekwencyjnych. Na komputerze MIX mamy „INC1 c” kontra „LD1 0,1(LINK)”, co daje różnicę tylko jednego cyklu, ale na wielu maszynach nie można tak po prostu załadować wartości do rejestru indeksowego spod indeksowanego adresu. Jeżeli elementy listy z dowiązaniami należą do różnych segmentów pamięci, to dostęp może zajmować znacznie więcej czasu.

Widzimy, że stosując dowiązania, uwalniamy się od pewnych ograniczeń narzuconych przez tablicowy charakter pamięci i często możemy zwiększyć wydajność, chociaż płacimy za to pewną cenę. Zazwyczaj widać, która metodę należy zastosować w konkretnej sytuacji, a często obie metody są używane do realizacji różnych list w tym samym programie.

W kilku kolejnych przykładach zakładamy, że element listy zajmuje jedno słowo i że składają się nań dwa pola INFO (informacje) i LINK (dowiązanie):



Używanie struktur z dowiązaniami (wskaźnikowymi) zazwyczaj wymaga istnienia jakiegoś mechanizmu wyszukiwania wolnego miejsca na nowe elementy struktury (na przykład gdy wstawiamy element do listy). Standardowa metoda polega na utrzymywaniu dodatkowej listy nazywanej *listą wolnej pamięci*. Będziemy ją nazywali AVAIL (lub stosem AVAIL, gdyż jest ona zazwyczaj uporządkowana jak stos). Zbiór wszystkich wolnych elementów jest przechowywany na jednej liście z dowiązaniami, na początek której wskazuje zmienna AVAIL. Gdy chcemy zatem zarezerwować nowy element i przypisać jego adres do zmiennej X, wykonujemy:

$$X \leftarrow \text{AVAIL}, \quad \text{AVAIL} \leftarrow \text{LINK(AVAIL)}. \quad (4)$$

Te operacje powodują zdjęcie elementu z wierzchołka stosu AVAIL i przypisanie do X wskaźnika do tego elementu. Operacja (4) jest wykorzystywana tak często, że wprowadzimy dla niej specjalne oznaczenie: „ $X \leftarrow \text{AVAIL}$ ” oznacza przypisanie do X wskaźnika do nowego elementu.

Gdy usuwamy element i nie chcemy go więcej używać, wykonujemy operację odwrotną do (4):

$$\text{LINK}(X) \leftarrow \text{AVAIL}, \quad \text{AVAIL} \leftarrow X. \quad (5)$$

Ta operacja zwraca element o adresie X do listy nie używanych elementów. Operację (5) oznaczamy przez „ $\text{AVAIL} \Leftarrow X$ ”.

Nie powiedzieliśmy jeszcze paru istotnych rzeczy o stosie AVAIL ; między innymi, jak należy go zainicjować na początku programu. Robi się to w następujący sposób: (a) Łączymy wszystkie elementy składające się na pamięć, która ma być przydzielana dynamicznie, (b) przypisujemy do AVAIL adres pierwszego z tych elementów, (c) dowlażanie ostatniego elementu ustawiamy na Λ . Zbiór wszystkich elementów przeznaczonych do dynamicznego przydzielania nazywamy *stertą*.

W omówieniu pominęliśmy przypadek przepełnienia. W (4) nie sprawdzamy, czy zostały jeszcze jakieś wolne elementy. Operacja $X \Leftarrow \text{AVAIL}$ powinna być w istocie zdefiniowana następująco:

$$\begin{aligned} &\text{jeśli } \text{AVAIL} = \Lambda, \text{ to PRZEPEŁNIENIE;} \\ &\text{w przeciwnym razie } X \leftarrow \text{AVAIL}, \text{ AVAIL} \leftarrow \text{LINK}(\text{AVAIL}). \end{aligned} \quad (6)$$

Należy zawsze pamiętać o możliwości wystąpienia przepełnienia. W tym przypadku wystąpienie **PRZEPEŁNIENIA** oznacza konieczność zatrzymania programu; możemy ewentualnie uruchomić podprogram „odśmiecania”, który spróbuje znaleźć trochę wolnego miejsca. Odśmiecanie omawiamy w punkcie 2.3.5.

Istnieje inna ważna metoda korzystania ze stosu AVAIL . Często nie wiemy z góry, ile pamięci należy przeznaczyć na stertę. W pamięci może znajdować się na przykład tablica o zmiennym rozmiarze, którą chcemy utrzymywać równocześnie ze strukturami wskaźnikowymi. W takiej sytuacji nie chcemy na stertę przeznaczać całej dostępnej pamięci. Przypuśćmy, że stertę mają tworzyć lokacje o rosnących adresach począwszy od L_0 i że granica tego obszaru nie powinna przekroczyć zmiennej **SEQMIN** (reprezentującej dolną granicę tablicy). Korzystając z nowej zmiennej **POOLMAX**, możemy zastosować następujące rozwiązanie:

- a) Na początek $\text{AVAIL} \leftarrow \Lambda$ i $\text{POOLMAX} \leftarrow L_0$.
- b) Operacja $X \Leftarrow \text{AVAIL}$ przybiera postać:

„Jeśli $\text{AVAIL} \neq \Lambda$, to $X \leftarrow \text{AVAIL}$, $\text{AVAIL} \leftarrow \text{LINK}(\text{AVAIL})$.

W przeciwnym razie $X \leftarrow \text{POOLMAX}$ i $\text{POOLMAX} \leftarrow X + c$,
gdzie c jest rozmiarem elementu;

PRZEPEŁNIENIE występuje, gdy $\text{POOLMAX} > \text{SEQMIN}$ ”.

- c) Gdy w innej części programu próbujemy zmniejszyć wartość **SEQMIN**, powinniśmy zgłosić **PRZEPEŁNIENIE**, jeśli $\text{SEQMIN} < \text{POOLMAX}$.
- d) Operacja $\text{AVAIL} \Leftarrow X$ pozostaje bez zmian w stosunku do (5).

Ten pomysł to trochę więcej niż pomysł poprzedni uzupełniony o metodę pozykiwania pamięci w przypadku **PRZEPEŁNIENIA** opisanego w (6). Dążymy do utrzymywania jak najmniejszej sterty. Wiele osób używa tego rozwiązania nawet wtedy, gdy *wszystkie* listy korzystają z dynamicznego przydzielania pamięci (innymi słowy, gdy **SEQMIN** jest stałą), ponieważ pozwala to na uniknięcie

czasochłonnego łączenia w listę wolnych elementów podczas tworzenia sterty na początku programu, a także ułatwia uruchamianie i wyszukiwanie błędów. Moglibyśmy oczywiście listę sekwencyjną umieścić na dole pamięci, a stertę na górze (i zmienić nazwy POOLMAX i SEQMIN na POOLMIN i SEQMAX).

Widać, że utrzymywanie sterty oraz wydajne poszukiwanie i zwalnianie elementów nie jest rzeczą trudną. Dzięki tej metodzie dysponujemy zasobami surowca do budowy struktur z dowiązaniami. W naszym omówieniu zakładaliśmy, że wszystkie elementy mają stałą wielkość c . Przypadki, w których posługujemy się elementami o różnych rozmiarach, są jednak bardzo istotne, ale odłożymy ich analizę do podrozdz. 2.5. Teraz omówimy kilka najczęściej spotykanych operacji na listach, dotyczących wykorzystania list jako stosów i kolejek.

Najprostszym rodzajem listy z dowiązaniami jest stos. Na rysunku 5 widać typowy stos ze wskaźnikiem T zawierającym dowiązanie do wierzchołka stosu. Gdy stos jest pusty, wskaźnik ma wartość Λ .

Jasne jest, w jaki sposób włożyć informację Y na wierzch takiego stosu za pomocą dodatkowej zmiennej wskaźnikowej P .

$$P \leftarrow \text{AVAIL}, \quad \text{INFO}(P) \leftarrow Y, \quad \text{LINK}(P) \leftarrow T, \quad T \leftarrow P. \quad (8)$$

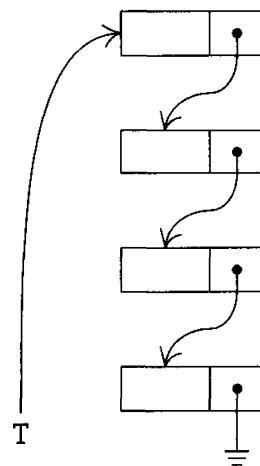
By przypisać do Y informację z wierzchołka stosu i ją ze stosu zdjąć, wykonujemy:

Jeśli $T = \Lambda$, to NIEDOMIAR;
w przeciwnym razie $P \leftarrow T$, $T \leftarrow \text{LINK}(P)$, $Y \leftarrow \text{INFO}(P)$, $\text{AVAIL} \leftarrow P$. (9)

Warto porównać te operacje z analogcznym mechanizmem stosowanym w przypadku stosów przydzielanych sekwencyjnie (2a) i (3a) z punktu 2.2.2. Czytelnik powinien przyjrzeć się niezmiernie ważnym operacjom (8) i (9).

Zanim zajmiemy się kolejkami, zobaczymy, w jaki sposób operacje na stosie można zapisać w postaci programu na komputer MIX. Program realizujący wstawienie elementu ($P \equiv rI1$) może wyglądać tak:

INFO EQU 0:3	Definicja pola INFO.
LINK EQU 4:5	Definicja pola LINK.
LD1 AVAIL	$P \leftarrow \text{AVAIL}$.
J1Z OVERFLOW	Czy $\text{AVAIL} = \Lambda$? $\left. \begin{array}{l} P \leftarrow \text{AVAIL} \\ \text{INFO}(P) \leftarrow Y \\ \text{LINK}(P) \leftarrow T \\ T \leftarrow P \end{array} \right\} \text{P} \leftarrow \text{AVAIL}$
LDA 0,1(LINK)	
STA AVAIL	$\text{AVAIL} \leftarrow \text{LINK}(P)$.
LDA Y	
STA 0,1(INFO)	$\text{INFO}(P) \leftarrow Y$.
LDA T	
STA 0,1(LINK)	$\text{LINK}(P) \leftarrow T$.
ST1 T	$T \leftarrow P$.

(10)


Rys. 5. Przykład stosu z dowiązaniami.

Wykonanie programu zabiera 17 jednostek czasu, podczas gdy analogiczna operacja na strukturze sekwencyjnej wymagała tylko 12 jednostek (choćże obsługa **PRZEPEŁNIENIA** w przypadku sekwencyjnym często trwałaby dużo dłużej). W tym oraz w kolejnych programach w rozdziale 2 przyjmujemy, że sytuacja **PRZEPEŁNIENIA** oznacza albo koniec programu, albo podprogram znajdujący wolne miejsce i powracający do lokacji $rJ - 2$.

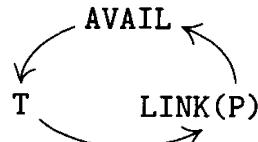
Program realizujący usuwanie elementu jest równie prosty:

LD1	T	$P \leftarrow T.$	
J1Z	UNDERFLOW	Czy $T = \Lambda?$	
LDA	0,1(LINK)		
STA	T	$T \leftarrow \text{LINK}(P).$	
LDA	0,1(INFO)		(11)
STA	Y	$Y \leftarrow \text{INFO}(P).$	
LDA	AVAIL		
STA	0,1(LINK)	$\text{LINK}(P) \leftarrow \text{AVAIL}.$	$\left. \begin{array}{l} \text{AVAIL} \leftarrow P \\ \text{AVAIL} \leftarrow P. \end{array} \right\}$
ST1	AVAIL		

Warto zauważyć, że każda z tych operacji powoduje cykliczną permutację trzech dowiązań. Na przykład w operacji wstawienia: niech P będzie wartością **AVAIL** przed wstawieniem; jeśli $P \neq \Lambda$, to po operacji

wartość **AVAIL** to poprzednia wartość **LINK(P)**,
 wartość **LINK(P)** to poprzednia wartość **T**,
 wartość **T** to poprzednia wartość **AVAIL**.

Zatem wstawianie elementu (poza przypisaniem $\text{INFO}(P) \leftarrow Y$) jest permutacją cykliczną



Podobnie z usuwaniem elementu: gdy P jest wartością T przed operacją oraz $P \neq \Lambda$, mamy $Y \leftarrow \text{INFO}(P)$ i

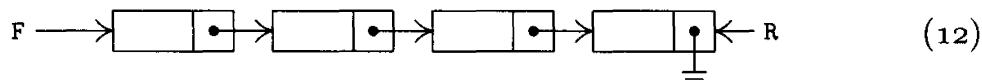


Fakt, że permutacja jest cykliczna, nie jest istotny, bo *każda* permutacja trzech elementów przemieszczająca każdy z nich jest cykliczna. Ważne jest raczej to, że w tych operacjach permutujemy dokładnie trzy dowiązania.

Algorytmy (8) i (9) wkładania i usuwania elementu dotyczą stosów, ale można je stosować do wstawiania i usuwania elementów z *dowolnych* list liniowych. Nowy element jest wstawiany bezpośrednio przed elementem wskazywanym przez zmienną T . Wstawienie elementu $2\frac{1}{2}$ z (2) można zrealizować za pomocą operacji (8), gdy $T = \text{LINK}(\text{LINK}(\text{FIRST}))$.

W równie wygodny sposób można posługiwać się kolejkami. W tym przypadku łatwo zauważać, że dowiązania powinny prowadzić od początku kolejki

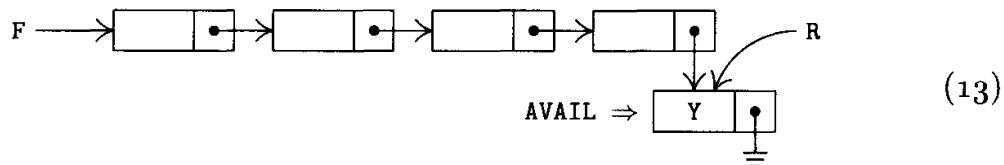
w kierunku jej końca, by po usunięciu elementu z początku od razu wiadomo było, który element będzie pierwszy. Będziemy korzystać ze zmennych dowiązaniowych F i R zawierających dowiązania do początku i końca kolejki:



Z wyjątkiem R ten rysunek jest w zasadzie identyczny z rysunkiem 5 na stronie 267.

Przy projektowaniu układu listy należy uważnie opisać wszystkie przypadki, a szczególnie sytuacje, gdy lista jest pusta. Jeden z najczęściej spotykanych błędów programistycznych mających związek z dowiązaniami polega na błędnej obsłudze pustych list. Inny częsty błąd polega na zapominaniu o konieczności modyfikacji niektórych dowiązań przy modyfikacji struktury. By ustrzec się błędów pierwszego rodzaju, należy zawsze dokładnie zbadać „warunki brzegowe”. Z drugim typem błędów można sobie poradzić, rysując diagramy przedstawiające strukturę „przed” i „po” operacji, a następnie odczytując z nich, które dowiązania należy zmodyfikować i w jaki sposób.

Zilustrujmy powyższe uwagi na przykładzie kolejki. Po pierwsze, zastanówmy się nad operacją wstawiania do kolejki: jeśli przed wstawieniem mamy sytuację taką, jak na rysunku (12), to kolejka po wstawieniu elementu powinna wyglądać tak:



($AVAIL \rightarrow$ oznacza, że z listy wolnych elementów $AVAIL$ pobraliśmy nowy element). Z porównania (12) i (13) wynika, co trzeba zrobić, wstawiając Y na koniec kolejki:

$$P \leftarrow AVAIL, \quad INFO(P) \leftarrow Y, \quad LINK(P) \leftarrow \Lambda, \quad LINK(R) \leftarrow P, \quad R \leftarrow P. \quad (14)$$

Zastanówmy się nad sytuacją „brzegową”, kiedy kolejka jest pusta. Sytuacja „przed” wstawieniem elementu jest na razie niejasna. Póki co wiadomo jednak, że sytuacja „po” powinna wyglądać tak:

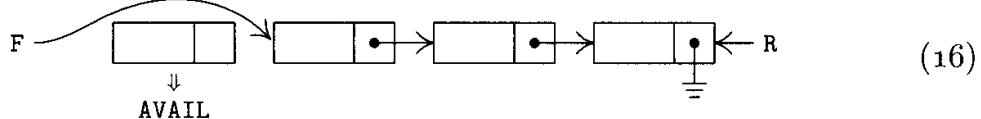


Dobrze by było, gdyby operacja (14) działała poprawnie również w przypadku kolejki pustej, nawet jeśli wstawienie do kolejki oznacza modyfikację *obu* dowiązań F i R , a nie jedynie R . Okazuje się, że (14) działa poprawnie, jeżeli dla pustej kolejki $R = LOC(F)$, przy założeniu $F \equiv LINK(LOC(F))$ – wartość zmiennej F musi być *przechowywana w polu* $LINK$. Wydajne sprawdzanie, czy kolejka jest pusta, umożliwi stosowanie konwencji, według której pustą kolejkę oznacza $F = \Lambda$. W naszym podejściu zakładamy zatem, że

kolejka jest pusta, gdy $F = \Lambda$ oraz $R = LOC(F)$.

Jeśli przy takich założeniach na pustej kolejce wykonamy operację (14), to otrzymamy (15).

Operację usuwania elementu z kolejki konstrujemy w podobny sposób. Jeśli (12) obrazuje sytuację przed usunięciem elementu, to po usunięciu powinniśmy mieć



Warunki brzegowe: musimy upewnić się, że usuwanie zostanie przeprowadzone poprawnie, gdy kolejka jest pusta przed lub po operacji. Te rozważania prowadzą do następującego algorytmu usuwania elementu z kolejki:

Jeśli $F = A$, to NIEDOMIAR;

w przeciwnym razie $P \leftarrow F$, $F \leftarrow \text{LINK}(P)$, $Y \leftarrow \text{INFO}(P)$, $\text{AVAIL} \leftarrow P$, (17)
i jeśli $F = \Lambda$, to $R \leftarrow \text{LOC}(F)$.

Zauważmy, że musimy zmienić R , gdy kolejka się opróżni. To właśnie jeden z takich „warunków brzegowych”, na które trzeba uważać.

Pokazana metoda nie jest jedynym sposobem reprezentowania kolejek za pomocą struktur z dowiązaniami. W ćwiczeniu 30 jest opisany nieco bardziej naturalny sposób, a w dalszej części rozdziału 2 pojawią się jeszcze inne rozwiązania. W istocie żadna z opisanych operacji nie jest jedynie słusznym rozwiązaniem. To raczej ilustracje prostych sposobów posługiwania się listami z dowiązaniami. Czytelnik bez dużego doświadczenia w posługiwaniu się takimi technikami programistycznymi dużo skorzysta na powtórnym przeczytaniu tego punktu, zanim przejdzie do dalszego materiału.

Jak dotąd omówiliśmy pewne sposoby wykonywania operacji na strukturach danych, ale nasze rozważania cały czas były „abstrakcyjne” w tym sensie, że nie pokazaliśmy problemów wziętych z życia. Ludzi zazwyczaj nie ciągnie do studiowania abstrakcyjnego modelu, dopóki nie ujrzą codziennych przejawów modelowanego zjawiska. Operacje omówione do tej pory, tj. wstawianie i usuwanie informacji z list o zmiennym rozmiarze oraz wykorzystanie stosów i kolejek, mają wiele praktycznych zastosowań. Można mieć nadzieję, że Czytelnik do tej pory natknął się na nie wystarczająco często, by interesowały go abstrakcyjne rozważania na ten temat. Teraz jednak porzucimy abstrakcję i zajmiemy się przykładami praktycznego zastosowania poznanych metod.

Nasz pierwszy problem nazywany jest *sortowaniem topologicznym*. Jest to ważny proces pojawiający się przy okazji problemów związanych z sieciami, z tzw. sieciami PERT, a nawet z lingwistyką. W istocie ma on szanse wypływać wszędzie tam, gdzie mamy do czynienia z *porządkiem częściowym*. Porządek częściowy na zbiorze S jest dowolną relacją między elementami zbioru S (oznaczaną za pomocą symbolu „ \preceq ”), która dla dowolnych (niekoniecznie różnych) elementów x , y i z ze zbioru S spełnia następujące warunki:

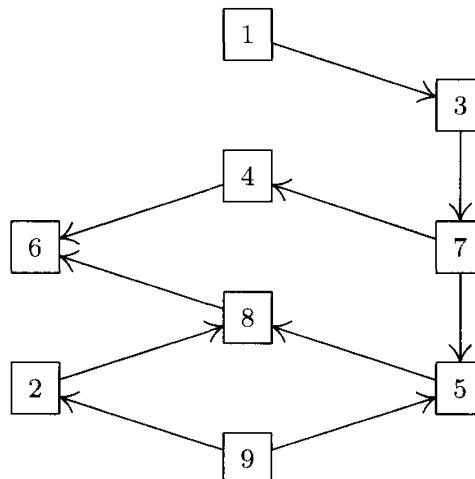
- i) Jeśli $x \preceq y$ i $y \preceq z$, to $x \preceq z$. (Przechodniość)
- ii) Jeśli $x \preceq y$ i $y \preceq x$, to $x = y$. (Antysymetria)
- iii) $x \preceq x$. (Zwrotność)

Zapis $x \preceq y$ można czytać „ x poprzedza lub równa się y ”. Jeśli $x \preceq y$ i $x \neq y$, to piszemy $x \prec y$ i czytamy „ x poprzedza y ”. Widać, że z (i), (ii) i (iii) wynika

- i') Jeśli $x \prec y$ i $y \prec z$, to $x \prec z$. (Przechodniość)
- ii') Jeśli $x \prec y$, to $y \not\prec x$. (Asimetria)*
- iii') $x \not\prec x$. (Przeciwzwrotność)

Relacja $y \not\prec x$ oznacza „ y nie poprzedza x ”. Jeśli mamy relację \prec spełniającą warunki (i'), (ii') i (iii'), to możemy odwrócić powyższy proces i zdefiniować: $x \preceq y$ wtedy i tylko wtedy, gdy $x \prec y$ lub $x = y$. Wówczas zachodzić będą własności (i), (ii) i (iii). Z tego powodu za definicję częściowego porządku możemy uznawać albo warunki (i), (ii), (iii), albo warunki (i'), (ii'), (iii'). Zauważmy, że warunek (ii') wynika z (i') i (iii'), ale (ii) nie wynika z (i) i (iii).

Porządek częściowy spotykamy często i w życiu codziennym, i w matematyce. Za przykład matematyczny mogą posłużyć: relacja $x \leq y$ w zbiorze liczb naturalnych, relacja $x \subseteq y$ między podzbiorami, relacja $x \setminus y$ (x dzieli y) w zbiorze dodatnich liczb całkowitych. W przypadku sieci PERT S jest zbiorem zadań do wykonania, a relacja „ $x \prec y$ ” oznacza „ x trzeba zrobić przed y ”.



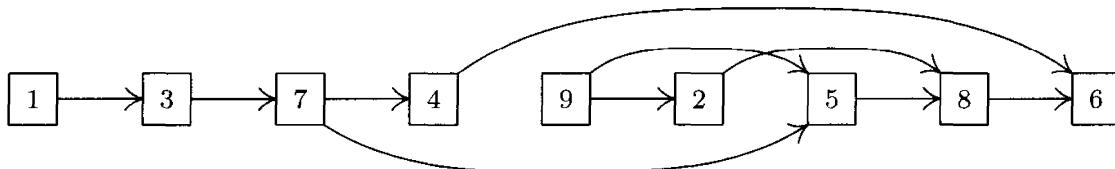
Rys. 6. Porządek częściowy.

Przyjmiemy naturalne założenie, że S jest zbiorem skończonym, ponieważ interesują nas zbiory dające się reprezentować w pamięci komputera. Częściowy porządek na zbiorze skończonym można zawsze przedstawić w postaci diagramu (zobacz rysunek 6). Na rysunku obiekty oznaczono za pomocą kwadracików, a relację za pomocą strzałek; $x \prec y$ oznacza, że na diagramie jest ścieżka zgodna z kierunkiem strzałek, prowadząca od kwadracika x do kwadracika y . Z własności (ii) porządku częściowego wynika, że na takim diagramie *nie może być pętli*. Jeżeli na rysunku 6 dorysowalibyśmy strzałkę od 4 do 1, to przestałby on przedstawiać porządek częściowy.

Problem sortowania topologicznego polega na *zanurzeniu porządku częściowego w porządku liniowym*, tj. na utworzeniu takiego ciągu elementów $a_1 a_2 \dots a_n$,

* W literaturze zamiast nazw „antysymetria” i „asimetria” spotyka się często nazwy „słaba antysymetria” i „silna antysymetria” (przyp. tłum.).

że jeśli $a_j \prec a_k$, to $j < k$. W interpretacji graficznej oznacza to, że kwadraciki trzeba ustawić w szeregu w ten sposób, by wszystkie strzałki były skierowane w prawo (zobacz rysunek 7). Nie jest oczywiste, że zawsze można to zrobić, chociaż na pewno nie dałoby się, gdyby diagram zawierał pętle. Z tego powodu algorytm, który zaraz zaprezentujemy, jest interesujący nie tylko dlatego, że przeprowadza pozytyczną operację, lecz także dlatego, że dowodzi, iż operację da się przeprowadzić dla dowolnego porządku częściowego.



Rys. 7. Relacja z rysunku 6 po sortowaniu topologicznym.

Jako przykład sortowania topologicznego rozważmy olbrzymi leksykon zawierający definicje pojęć technicznych. Piszemy $w_2 \prec w_1$, jeśli definicja słowa w_1 bezpośrednio lub pośrednio opiera się na definicji słowa w_2 . Taka relacja jest porządkiem częściowym, jeśli nie definiuje się nieznanego przez nieznanego. W tym przypadku problem sortowania topologicznego oznacza *znalezienie takiego sposobu ułożenia haseł w leksykonie, by definicje nie zawierały haseł, które jeszcze nie zostały zdefiniowane*. Podobne problemy pojawiają się przy przetwarzaniu deklaracji w pewnych językach programowania, a także przy pisaniu podręczników języków programowania oraz książek o strukturach danych.

Istnieje bardzo prosta metoda sortowania topologicznego: zaczynamy od wybrania elementu, który nie ma poprzednika. Ten element można umieścić na pierwszym miejscu ciągu wynikowego. Następnie usuwamy go ze zbioru S . Zbiór, który zostaje po tej operacji, jest w dalszym ciągu częściowo uporządkowany, więc proces możemy powtarzać, dopóki cały zbiór nie zostanie posortowany. Na rysunku 6 na przykład moglibyśmy zacząć od usunięcia 1 lub 9; po usunięciu 1 moglibyśmy usunąć 3 itd. Algorytm nie zadziałałby jedynie na takim niepustym zbiorze częściowo uporządkowanym, w którym każdy element miałby poprzednik. Dla takiego zbioru algorytm nie znajałby kolejnego elementu do usunięcia. Jednakże jeśli każdy element miałby poprzednik, to moglibyśmy zbudować dowolnie długi ciąg b_1, b_2, b_3, \dots , w którym $b_{j+1} \prec b_j$. Ponieważ S jest zbiorem skończonym, dla pewnych $j < k$ musiałaby zachodzić równość $b_j = b_k$. To pociąga za sobą $b_k \preceq b_{j+1}$, co jest sprzeczne z (ii).

By wydajnie zaprogramować powyższy proces, musimy być przygotowani na wykonywanie opisanych akcji, a konkretnie na znajdowanie elementu, który nie ma poprzedników, oraz usuwanie go ze zbioru. Sposób pisania programu zależy również od charakterystyki wejścia i wyjścia. Program uniwersalny przyjmowałby znakowe nazwy elementów i umożliwiał sortowanie olbrzymich zbiorów, zapewne nie mieszczących się w pamięci komputera. Takie komplikacje zaciemniają jednak istotę problemu. Na danych znakowych można wydajnie wykonywać operacje,

korzystając z metod opisanych w rozdziale 6, a radzenie sobie z dużymi zbiorami pozostawiamy jako ciekawe zadanie dla Czytelnika.

Założymy zatem, że elementy, które mamy posortować, są ponumerowane od 1 do n . Dane wejściowe programu będą zapisane na jednostce pamięci taśmowej nr 1: każdy blok na taśmie zawiera 50 par liczb, gdzie para (j, k) oznacza, że element j poprzedza element k . Pierwsza para ma jednak postać $(0, n)$, gdzie n oznacza liczbę elementów. Para $(0, 0)$ określa koniec danych wejściowych. Założymy, że wartość n plus liczba par będących w relacji jest mniejsza niż pojemność pamięci oraz że program nie musi sprawdzać poprawności danych wejściowych. Program powinien wyprowadzić na jednostkę taśmową nr 2 ciąg numerów elementów w wyznaczonym porządku topologicznym, zakończony zerem.

Na przykład na wejściu może pojawić się opis następujących par będących ze sobą w relacji:

$$9 \prec 2, 3 \prec 7, 7 \prec 5, 5 \prec 8, 8 \prec 6, 4 \prec 6, 1 \prec 3, 7 \prec 4, 9 \prec 5, 2 \prec 8. \quad (18)$$

Nie trzeba podawać tych par, dla których zachodzenie relacji można wywnioskować z podanych par oraz własności porządku częściowego, na przykład $9 \prec 8$ (wynika z $9 \prec 5$ i $5 \prec 8$) można podać, ale można pominąć bez wpływu na opisywaną relację. W ogólności wystarczy podać jedynie pary odpowiadające strzałkom na diagramie.

Poniższy algorytm korzysta z tablicy $X[1], X[2], \dots, X[n]$, a każdy element $X[k]$ jest postaci

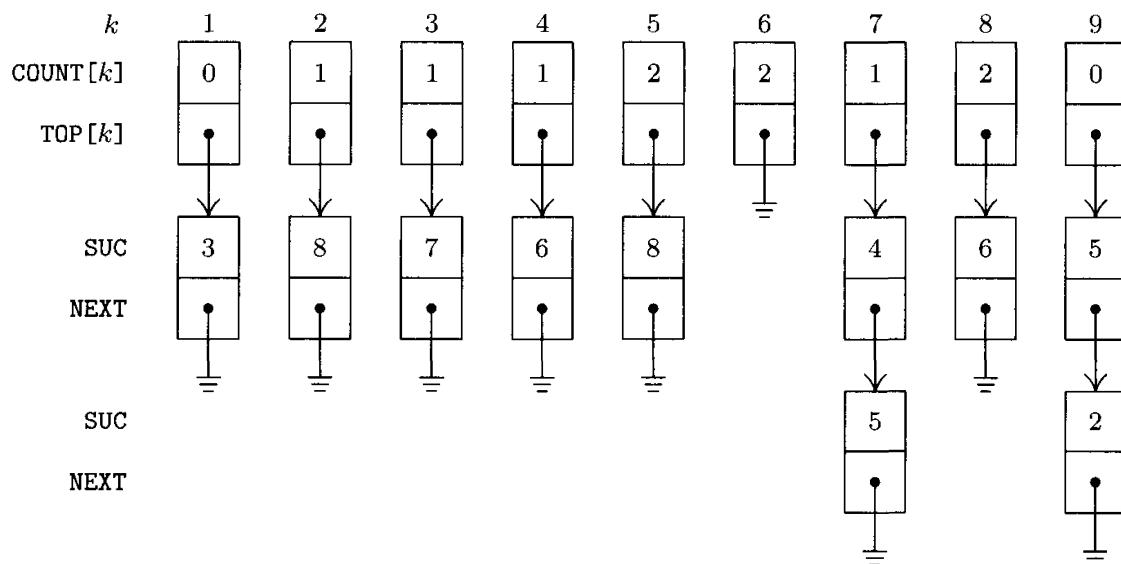
+	0	COUNT [k]	TOP [k]	.
---	---	----------------------	--------------------	---

Wartość **COUNT**[k] oznacza tutaj liczbę poprzedników elementu k (tj. liczbę par $j \prec k$ podanych na wejściu), a **TOP**[k] jest dowiązaniem do początku listy bezpośrednich następców elementu k . Na tę listę składają się elementy postaci

+	0	SUC	NEXT	,
---	---	------------	-------------	---

gdzie **SUC** jest pewnym bezpośredniem następciem k , a **NEXT** wskazuje na następny element listy. Na rysunku 8 pokazano schemat zawartości pamięci odpowiadającej danym wejściowym (18).

Korzystając z takiego projektu struktury danych, nietrudno wymyślić szczegóły algorytmu. Chcemy wypisać te elementy, dla których pole **COUNT** zawiera zero, a następnie zmniejszyć o jeden zawartość pola **COUNT** wszystkich bezpośrednich następców tych elementów. Dowcip polega na tym, by nie „szukać” elementów, dla których pole **COUNT** jest równe zero. Ten efekt można osiągnąć, przechowując wszystkie takie elementy w kolejce. Dowiązania tworzące tę kolejkę utrzymujemy w polach **COUNT**, które już do zliczania poprzedników nie są potrzebne. Dla jasności w poniższym algorytmie używamy notacji **QLINK**[k] na oznaczenie **COUNT**[k], gdy używamy tego pola do przechowywania dowiązań zamiast liczby poprzedników.



Rys. 8. Reprezentacja rysunku 6 odpowiadająca relacji zadanej przez (18).

Algorytm T (Sortowanie topologiczne). Algorytm wczytuje ciąg par $j \prec k$, z których każda oznacza, że element j poprzedza element k w pewnym porządku częściowym, $1 \leq j, k \leq n$. Algorytm wyprowadza ciąg n elementów określający zanurzenie porządku częściowego w porządku liniowym. Wewnętrzne struktury danych: QLINK[0], COUNT[1] = QLINK[1], COUNT[2] = QLINK[2], ..., COUNT[n] = QLINK[n]; TOP[1], TOP[2], ..., TOP[n]; sterta zawierająca po jednym elemencie dla każdej pary na wejściu z polami SUC i NEXT, jak na rysunku powyżej; zmienna wskaźnikowa P do adresowania sterty; zmienne całkowitoliczbowe F i R używane do adresowania początku i końca kolejki zbudowanej z dowiązań w polach QLINK; zmienna N zliczająca liczbę elementów, które należy wyprowadzić na wyjście.

- T1.** [Inicjowanie] Wczytaj wartość n . Przyjmij COUNT[k] $\leftarrow 0$, TOP[k] $\leftarrow \Lambda$ dla $1 \leq k \leq n$. Przyjmij N $\leftarrow n$.
- T2.** [Kolejna para] Pobierz kolejną parę „ $j \prec k$ ” z wejścia; jeżeli nie ma więcej danych na wejściu, to idź do T4.
- T3.** [Zapamiętanie pary] ZwiększM COUNT[k] o jeden. Przyjmij

$$P \Leftarrow \text{AVAIL}, \text{SUC}(P) \leftarrow k, \text{NEXT}(P) \leftarrow \text{TOP}[j], \text{TOP}[j] \leftarrow P.$$

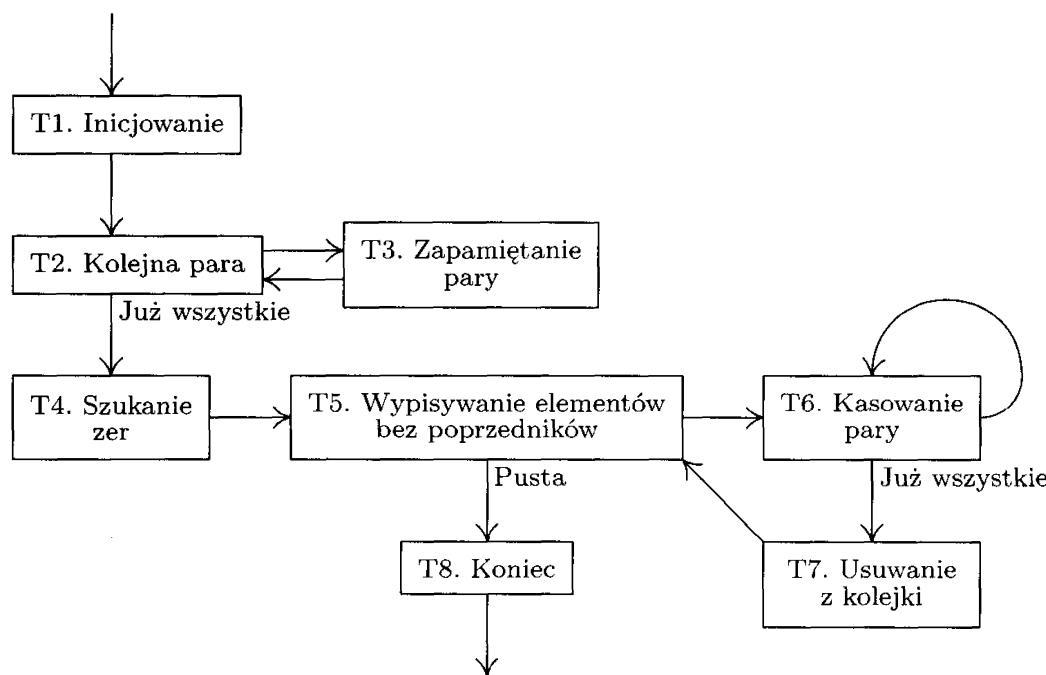
(To jest (8)). Przejdź do T2.

- T4.** [Szukanie zer] (W tym momencie zakończyliśmy fazę wczytywania danych; przykładowe dane wejściowe (18) zostałyby przerobione na reprezentację pokazaną na rysunku 8. Naszym następnym zadaniem jest zainicjowanie kolejki elementów o zerowej wartości COUNT, połączonej za pomocą dowiązań QLINK). Przyjmij R $\leftarrow 0$ i QLINK[0] $\leftarrow 0$. Dla $1 \leq k \leq n$ sprawdź wartość COUNT[k], i jeśli jest zerem, to przyjmij QLINK[R] $\leftarrow k$ i R $\leftarrow k$. Gdy wykonasz to dla wszystkich k , przypisz F $\leftarrow \text{QLINK}[0]$ (to jest pierwsza wartość k , dla której COUNT[k] zawierało zero).

- T5.** [Wypisywanie początku kolejki] Wypisz wartość F . Jeśli $F = 0$, idź do T8; w przeciwnym razie przyjmij $N \leftarrow N - 1$, $P \leftarrow \text{TOP}[F]$. (Ponieważ struktury QLINK i COUNT pokrywają się, mamy $\text{QLINK}[R] = 0$; stąd warunek $F = 0$ oznacza, że kolejka jest pusta).
- T6.** [Kasowanie par] Jeśli $P = \Lambda$, przejdź do T7. W przeciwnym razie zmniejsz $\text{COUNT}[\text{SUC}(P)]$ o jeden. Jeśli wyszło zero, to przyjmij $\text{QLINK}[R] \leftarrow \text{SUC}(P)$ i $R \leftarrow \text{SUC}(P)$. Przypisz $P \leftarrow \text{NEXT}(P)$ i powtórz ten krok. (Usuwamy ze struktury wszystkie pary postaci „ $F \prec k$ ” dla pewnego k , jednocześnie wstawiając do kolejki elementy, których wszystkie poprzedniki zostały usunięte).
- T7.** [Usuwanie z kolejki] Przyjmij $F \leftarrow \text{QLINK}[F]$ i przejdź do T5.
- T8.** [Koniec] Zakończ wykonywanie algorytmu. Jeśli $N = 0$, to wyprowadziliśmy wszystkie elementy w porządku topologicznym, kończąc zerem. W przeciwnym razie N nie wyprowadzonych elementów zawiera pętle, co oznacza, że podana relacja nie była porządkiem częściowym. (W ćwiczeniu 23 podajemy algorytm wypisujący jedną z takich pętli). ■

Polecamy Czytelnikowi ręczne wykonanie algorytmu na danych wejściowych (18). Algorytm T ilustruje jednoczesne zastosowanie technik tablicowych oraz wskaźnikowych. Główna struktura $X[1], \dots, X[n]$ ma charakter tablicowy, ponieważ chcemy szybko móc dostawać się do jej elementów w nieznanym porządku (krok T3). (Jeśli jednak dane wejściowe byłyby nazwami znakowymi, należałoby skorzystać z innej struktury, umożliwiającej szybkie przeszukiwanie, zobacz rozdział 6). Dowiązań używamy w strukturze przechowującej „bezpośrednie następni”, ponieważ kolejność dostępu do tych informacji nie ma znaczenia. Kolejka elementów do wyprowadzenia przechowywana jest wewnątrz struktury tablicowej za pomocą dowiązań łączących elementy w takim porządku, w jakim są wyprowadzane. Te dowiązań realizujemy za pomocą indeksów struktury X zamiast adresów w pamięci. Innymi słowy, gdy początkiem kolejki jest $X[k]$, mamy $F = k$ zamiast $F = \text{LOC}(X[k])$. Operacje na kolejce w krokach T4, T6 i T7 nie są identyczne z tymi w (14) i (17), ponieważ korzystamy ze specyficznych własności tej kolejki w naszym programie – w tej części algorytmu nie musimy tworzyć nowych wierzchołków, ani zwracać do sterty wierzchołków nieużywanych.

Kod algorytmu T w asemblerze komputera MIX ma kilka ciekawych miejsc. Z uwagi na fakt, że nie usuwamy elementów ze struktur danych (bo nie musimy się martwić o zwalnianie miejsca do późniejszego użytku), kod operacji $P \leftarrow \text{AVAIL}$ jest bardzo prosty (zobacz wiersze 19 i 32 poniższego programu) – nie musimy utrzymywać sterty i możemy brać kolejne elementy. Program zawiera pełną obsługę wprowadzania i wyprowadzania danych z/na taśmę magnetyczną, zgodnie z opisanymi wcześniej konwencjami. Dla prostoty pominięto buforowanie. Czytelnik nie powinien mieć problemu z analizą szczegółów kodu, ponieważ bardzo wiernie odzwierciedla on algorytm T. Zwracamy uwagę na wydajne wykorzystanie rejestrów indeksowych, będące istotnym aspektem metod wskaźnikowych.



Rys. 9. Sortowanie topologiczne.

Program T (*Sortowanie topologiczne*). Należy zwrócić uwagę na następujące odwzorowanie: rI6 \equiv N, rI5 \equiv wskaźnik bufora, rI4 \equiv k, rI3 \equiv j i R, rI2 \equiv AVAIL i P, rI1 \equiv F, TOP[j] \equiv X + j(4:5), COUNT[k] \equiv QLINK[k] \equiv X + k(2:3).

01	*	BUFOR I DEFINICJE PÓŁ	
02	COUNT	EQU 2:3	Definicje pól
03	QLINK	EQU 2:3	i nazw symbolicznych.
04	TOP	EQU 4:5	
05	SUC	EQU 2:3	
06	NEXT	EQU 4:5	
07	TAPEIN	EQU 1	Wejście: taśma nr 1.
08	TAPEOUT	EQU 2	Wyjście: taśma nr 2.
09	BUFFER	ORIG *+100	Bufor pamięci taśmowej.
10		CON -1	Wartownik końca bufora.
11	*	WCZYTYWANIE DANYCH	
12	TOPSORT IN	BUFFER(TAPEIN)	1 <u>T1. Inicjowanie.</u> Wczytaj pierwszy blok
13		JBUS *(TAPEIN)	taśmy; czekaj na koniec operacji.
14	1H	LD6 BUFFER+1	1 N \leftarrow n.
15		ENT4 0,6	1
16		STZ X,4	n+1 COUNT[k] \leftarrow 0 i TOP[k] $\leftarrow \Lambda$,
17		DEC4 1	n+1 dla 0 \leqslant k \leqslant n.
18		J4NN *-2	n+1 (Załóż, że QLINK[0] \leftarrow 0 w kroku T4).
19		ENT2 X,6	1 Dostępna pamięć zaczyna się za X[n].
20		ENT5 BUFFER+2	1 Przygotuj się na pierwszą parę (j, k).
21	2H	LD3 0,5	m+b <u>T2. Kolejna para.</u>
22		J3P 3F	m+b Czy j > 0?
23		J3Z 4F	b Koniec wejścia?
24		IN BUFFER(TAPEIN)	b-1 Wartownik; czytaj następny blok
25		JBUS *(TAPEIN)	taśmy, czekaj na koniec operacji.
26		ENT5 BUFFER	b-1 Ustaw wskaźnik bufora.

27	JMP	2B	<i>b</i> - 1
28 3H	LD4	1,5	<i>m</i> <u>T3. Zapamiętanie pary.</u>
29	LDA	X,4(COUNT)	<i>m</i> COUNT[<i>k</i>]
30	INCA	1	<i>m</i> +1
31	STA	X,4(COUNT)	<i>m</i> → COUNT[<i>k</i>].
32	INC2	1	<i>m</i> AVAIL ← AVAIL + 1.
33	LDA	X,3(TOP)	<i>m</i> TOP[<i>j</i>]
34	STA	0,2(NEXT)	<i>m</i> → NEXT(P).
35	ST4	0,2(SUC)	<i>m</i> <i>k</i> → SUC(P).
36	ST2	X,3(TOP)	<i>m</i> P → TOP[<i>j</i>].
37	INC5	2	<i>m</i> Zwiększa wskaźnik bufora.
38	JMP	2B	<i>m</i>
39 4H	IOC	0(TAPEIN)	1 Przewiń taśmę wejściową.
40	ENT4	0,6	1 <u>T4. Szukanie zer.</u> <i>k</i> ← <i>n</i> .
41	ENT5	-100	1 Ustaw wskaźnik bufora do czytania.
42	ENT3	0	1 R ← 0.
43 4H	LDA	X,4(COUNT)	<i>n</i> Sprawdź wartość COUNT[<i>k</i>].
44	JAP	*+3	<i>n</i> Czy niezerowa?
45	ST4	X,3(QLINK)	<i>a</i> QLINK[R] ← <i>k</i> .
46	ENT3	0,4	<i>a</i> R ← <i>k</i> .
47	DEC4	1	<i>n</i>
48	J4P	4B	<i>n</i> <i>n</i> ≥ <i>k</i> ≥ 1.
49 *	FAZA SORTOWANIA		
50	LD1	X(QLINK)	1 F ← QLINK[0].
51 5H	JBUS	*(TAPEOUT)	<u>T5. Wypisywanie początku kolejki.</u>
52	ST1	BUFFER+100,5	<i>n</i> +1 Zapisz F do bufora.
53	J1Z	8F	<i>n</i> +1 Czy F jest zerem?
54	INC5	1	<i>n</i> Zwiększa wskaźnik bufora.
55	J5N	*+3	<i>n</i> Sprawdź, czy bufor pełny.
56	OUT	BUFFER(TAPEOUT)	<i>c</i> -1 Jeśli tak, wyprowadź blok.
57	ENT5	-100	<i>c</i> -1 Ustaw wskaźnik bufora.
58	DEC6	1	<i>n</i> N ← N - 1.
59	LD2	X,1(TOP)	<i>n</i> P ← TOP[F].
60	J2Z	7F	<i>n</i> <u>T6. Kasowanie par.</u>
61 6H	LD4	0,2(SUC)	<i>m</i> rI4 ← SUC(P).
62	LDA	X,4(COUNT)	<i>m</i> COUNT[rI4]
63	DECA	1	<i>m</i> -1
64	STA	X,4(COUNT)	<i>m</i> → COUNT[rI4].
65	JAP	*+3	<i>m</i> Czy doszliśmy do zera?
66	ST4	X,3(QLINK)	<i>n-a</i> Jeśli tak, to QLINK[R] ← rI4.
67	ENT3	0,4	<i>n-a</i> R ← rI4.
68	LD2	0,2(NEXT)	<i>m</i> P ← NEXT(P).
69	J2P	6B	<i>m</i> Powtórz, jeśli P ≠ Λ
70 7H	LD1	X,1(QLINK)	<i>n</i> <u>T7. Usuwanie z kolejki.</u>
71	JMP	5B	<i>n</i> F ← QLINK(F), przejdź do T5.
72 8H	OUT	BUFFER(TAPEOUT)	<i>1</i> <u>T8. Koniec.</u>
73	IOC	0(TAPEOUT)	<i>1</i> Wyprowadź ostatni blok i przewiń.
74	HLT	0,6	<i>1</i> Zatrzymaj, wyświetl N na konsoli.
75 X	END	TOPSORT	Początek pamięci danych. ■

Analizę algorytmu T daje się łatwo przeprowadzić za pomocą prawa Kirchhoffa. Czas wykonania jest postaci $c_1m + c_2n$, gdzie m jest liczbą par na wejściu, n liczbą obiektów, a c_1 i c_2 pewnymi stałymi. Trudno sobie wyobrazić szybszy algorytm rozwiązujący ten problem! Dokładne wartości potrzebne do analizy algorytmu podano w programie T, gdzie $a = \text{liczba elementów nie mających poprzedników}$, $b = \text{liczba bloków na taśmie wejściowej} = \lceil (m+2)/50 \rceil$, $c = \text{liczba bloków na taśmie wyjściowej} = \lceil (n+1)/100 \rceil$. Czas wykonania programu, bez operacji wejścia-wyjścia, wynosi tylko $(32m + 24n + 7b + 2c + 16)u$.

Algorytm sortowania topologicznego podobny do algorytmu T (ale nie zawierający ważnego mechanizmu kolejki) został po raz pierwszy opublikowany przez A. B. Kahna, CACM 5 (1962), 558–562. Fakt, że zawsze można posortować topologicznie porządek częściowy, został po raz pierwszy udowodniony w druku przez E. Szpirajna, Fundamenta Mathematica 16 (1930), 386–389; dowód dotyczy również porządków na zbiorach nieskończonych, a autor zaznacza, że te wyniki były już wcześniej znane wśród jego współpracowników.

Choć algorytm T jest bardzo wydajny, w punkcie 7.4.1 zajmiemy się jeszcze lepszym algorytmem sortowania topologicznego.

ĆWICZENIA

- ▶ 1. [10] Operacja (9) zdejmowania elementu ze stosu przewiduje obsługę sytuacji NIEDOMIARU. Dlaczego operacja (8) wkładania elementu na stos nie przewiduje obsługi PRZEPEŁNIENIA?
- 2. [22] Napisz „uniwersalny” podprogram w asemblerze komputera MIX realizujący operację włożenia elementu na stos (10). Twój podprogram powinien spełniać następujące warunki (porównaj punkt 1.4.1):

Sekwencja wywołania: JMP INSERT Skok do podprogramu.

NOP T Lokacja wskaźnika stosu.

Warunki wejściowe: rA = informacja, którą należy zapisać w polu INFO nowego elementu.

Warunki wyjściowe: Na stosie, którego wskaźnikiem jest zmienna wskaźnikowa T, leży nowy element; rI1 = T; rI2, rI3 są naruszone.

- 3. [22] Napisz „uniwersalny” podprogram w asemblerze komputera MIX realizujący operację zdejmowania elementu ze stosu (11). Twój podprogram powinien spełniać następujące warunki:

Sekwencja wywołania: JMP DELETE Skok do podprogramu.

NOP T Lokacja wskaźnika stosu.

JMP UNDERFLOW Pierwsze wyjście, wystąpił NIEDOMIAR.

Warunki wejściowe: Brak.

Warunki wyjściowe: Jeśli stos, którego wskaźnikiem jest zmienna T, jest pusty, to podprogram korzysta z pierwszego wyjścia. W przeciwnym razie wierzchołek stosu jest usuwany i program wychodzi do trzeciej lokacji po „JMP DELETE”. W tym przypadku rI1 = T, a rA jest zawartością pola INFO usuniętego elementu. W obu przypadkach wartości rI2 i rI3 mogą być zmienione.

- 4. [22] Program (10) korzysta z operacji $P \Leftarrow \text{AVAIL}$ opisanej w (6). Pokaż, w jaki sposób napisać podprogram OVERFLOW, żeby bez żadnych zmian w kodzie (10) operacja

$P \leftarrow \text{AVAIL}$ korzystała z SEQMIN , tak jak w (7). Twój podprogram nie powinien modyfikować żadnego rejestru poza rJ i być może wskaźnikiem porównania. Podprogram powinien wychodzić do lokacji $rJ - 2$ zamiast rJ .

- ▶ 5. [24] Operacje (14) i (17) realizują kolejkę; pokaż, w jaki sposób zdefiniować operację „wstaw na początek”, by dysponować wszystkimi operacjami kolejki dwustronnej o ograniczonym wyjściu. Jak należałoby zdefiniować operację „usuń z końca” (co umożliwiłoby realizację ogólnej kolejki dwustronnej)?
- ▶ 6. [21] W (14) wykonujemy przypisanie $\text{LINK}(P) \leftarrow \Lambda$, podczas gdy kolejne wstawienie na koniec kolejki spowoduje zmianę tej wartości. Pokaż, w jaki sposób można uniknąć przypisania do $\text{LINK}(P)$ w (14), jeśli zmienimy warunek „ $F = \Lambda$ ” w (17).
- ▶ 7. [23] Zaprojektuj algorytm odwracający listę liniową z dowiązaniami, taką jak (1). Algorytm ma odwracać kolejność elementów listy. [W liście (1) po odwróceniu zmienna FIRST powinna wskazywać na element zawierający 5, ten element powinien mieć dowiązanie do elementu zawierającego 4 itd.]. Załóż, że elementy listy mają postać (3).
- 8. [24] Napisz program w asemblerze komputera MIX realizujący algorytm z ćwiczenia 7. Zaprojektuj tak swój program, by działał jak najszybciej.
- 9. [20] Która z następujących relacji jest porządkiem częściowym na zbiorze S ? [Uwaga: Tam, gdzie pojawia się „ $x \prec y$ ”, pytanie należy rozumieć: „czy relacja \preceq zdefiniowana jako $\langle x \preceq y \equiv (x \prec y \text{ lub } x = y) \rangle$ jest porządkiem częściowym”]. (a) $S = \text{wszystkie liczby wymierne}$, $x \prec y$ oznacza $x > y$. (b) $S = \text{wszyscy ludzie}$, $x \prec y$ oznacza, że x jest przedkiem y . (c) $S = \text{wszystkie liczby całkowite}$, $x \preceq y$ oznacza, że x jest wielokrotnością y (tj. $x \text{ mod } y = 0$). (d) $S = \text{wszystkie twierdzenia matematyczne udowodnione w tej książce}$, $x \prec y$ oznacza, że dowód y zależy od prawdziwości x . (e) $S = \text{wszystkie dodatnie liczby całkowite}$, $x \preceq y$ oznacza, że $x + y$ jest liczbą parzystą. (f) $S = \text{zbiór podprogramów}$, $x \prec y$ oznacza „ x wywołuje y ”, tzn. kod y może być wykonywany podczas wykonywania kodu x , nie dopuszczamy rekursji.
- 10. [M21] Wiedząc, że relacja „ \subset ” spełnia warunki (i) i (ii) z definicji porządku częściowego, pokaż, że relacja „ \preceq ” zdefiniowana jako „ $x \preceq y$ wtedy i tylko wtedy, gdy $x = y$ lub $x \subset y$ ”, spełnia wszystkie trzy warunki tej definicji.
- ▶ 11. [24] Wynik sortowania topologicznego nie jest jednoznaczny, ponieważ elementy można czasem ustawić na wiele sposobów spełniających porządek topologiczny. Znajdź wszystkie sposoby posortowania topologicznego elementów z rysunku 6.
- 12. [M20] Zbiór n -elementowy ma 2^n podzbiorów częściowo uporządkowanych przez relację zawierania. Podaj dwa ciekawe sposoby ustawienia takich podzbiorów w porządku topologicznym.
- 13. [M48] Na ile sposobów można ustawić 2^n podzbiorów opisanych w ćwiczeniu 12 w porządku topologicznym? (Podaj odpowiedź jako funkcję n).
- 14. [M21] Porządek liniowy na zbiorze S jest porządkiem częściowym spełniającym dodatkowo warunek „porównywalności”:
 - (iv) Dla dowolnych elementów x, y ze zbioru S albo $x \preceq y$, albo $y \preceq x$.

Udowodnij wprost z definicji, że jeśli relacja \preceq jest porządkiem liniowym, to wynik sortowania topologicznego jest wyznaczony jednoznacznie. (Możesz założyć, że zbiór S jest skończony).

15. [M25] Pokaż, że dla dowolnego porządku częściowego na zbiorze skończonym S istnieje *jedyny* zbiór par postaci „ $x \prec y$ ” wyznaczający ten porządek i nie zawierający par, które wynikają z pozostałych (porównaj (18) i rysunek 6). Czy to twierdzenie zachodzi dla nieskończonych zbiorów S ?

16. [M22] Dla dowolnego porządku częściowego na zbiorze $S = \{x_1, \dots, x_n\}$ możemy zbudować jego *macierz incydencji* (a_{ij}), gdzie $a_{ij} = 1$, jeśli $x_i \preceq x_j$, oraz $a_{ij} = 0$ w przeciwnym razie. Pokaż, że można tak spermutować rzędy i kolumny tej macierzy, że wszystkie elementy poniżej przekątnej będą zerami.

► **17.** [21] Jak wyglądają dane wyjściowe algorytmu T, gdy na wejście podamy (18)?

18. [20] Co, jeśli w ogóle coś, oznaczają wartości $\text{QLINK}[0], \text{QLINK}[1], \dots, \text{QLINK}[n]$ w momencie zakończenia wykonywania algorytmu T?

19. [18] W algorytmie T sprawdzamy element na początku kolejki w kroku T5, ale nie usuwamy go, aż do kroku T7. Co się stanie, jeżeli za zakończenie kroku T5 wykonamy $F \leftarrow \text{QLINK}[F]$, zamiast czekać do T7?

► **20.** [24] W algorytmie T korzystamy z F , R i QLINK , by uzyskać kolejkę zawierającą elementy, dla których wartość COUNT zmalała do zera, ale których następcy nie zostały jeszcze zmodyfikowane. Czy zamiast kolejki moglibyśmy w tym miejscu użyć stosu? Jeśli tak, to porównaj zmodyfikowany algorytm z algorytmem T.

21. [21] Czy algorytm T będzie działał poprawnie, jeżeli któraś z par „ $j \prec k$ ” wystąpi na wejściu kilkakrotnie? Co się stanie, jeżeli dane wejściowe będą zawierały parę postaci „ $j \prec j$ ”?

22. [23] W programie T zakładamy, że na taśmie wejściowej zapisano informacje poprawne, ale w praktyce programy powinny zawsze dokładnie badać dane wejściowe, by wykluczyćomyłki i zapobiec „autodestrukcji”. Na przykład, jeżeli jedna z par zawierałaby liczbę ujemną, to program, zapisując coś do $X[k]$, mógłby „omyłkowo” zmodyfikować jeden ze swoich rozkazów. Zaproponuj sposoby kontroli poprawności danych wejściowych programu T.

► **23.** [27] Gdy algorytm sortowania topologicznego zatrzymuje się z powodu wykrycia pętli (zobacz krok T8), zazwyczaj nie ma sensu informować, że „wykryto pętlę”. O wiele lepiej wypisać jedną z pętli, pokazując tym samym, gdzie był błąd. Rozszerz algorytm T w ten sposób, by w odpowiednich przypadkach wypisywał pętlę. [*Wskazówka:* W tekście znajduje się dowód istnienia pętli, gdy w kroku T8 mamy $N > 0$; wynika z niego algorytm].

24. [24] Wprowadź modyfikacje algorytmu T z ćwiczenia 23 do programu T.

25. [47] Zaprojektuj jak najbardziej efektywny algorytm sortowania topologicznego bardzo dużych zbiorów S (mających znacznie więcej elementów niż da się za jednym razem zmieścić w pamięci komputera). Załącz, że wejście, wyjście oraz pomocnicza pamięć tymczasowa są realizowane za pomocą taśmy magnetycznej. [*Wskazówka:* Stosując zwykłe sortowanie danych wejściowych, można założyć, że wszystkie pary dotyczące konkretnego elementu występują obok siebie. Ale co dalej? W szczególności musimy rozważyć najgorszy przypadek, kiedy dany porządek jest silnie spermutowanym porządkiem liniowym. Ćwiczenie 24 we wstępnie do rozdziału 5 pokazuje, jak sobie z tym poradzić w $O(\log^2 n)$ przebiegach przez dane wejściowe].

26. [29] (*Podprogram przydziału pamięci*) Przypuśćmy, że taśma zawiera bibliotekę podprogramów w formie relokowalnej w stylu komputerów z lat sześćdziesiątych. Program ładowający chce wyznaczyć przesunięcie każdego wykorzystywanego podprogramu,

by w jednym przebiegu taśmy można było wczytać potrzebne podprogramy. Problem polega na tym, że niektóre podprogramy wymagają obecności pewnych innych podprogramów. Programy używane rzadko (umieszczone na końcu taśmy) mogą wywoływać podprogramy używane często (umieszczone na początku taśmy). Chcielibyśmy wieć, które programy należy załadować przed przejrzeniem całej taśmy.

Jeden ze sposobów poradzenia sobie z tym problemem polega na utrzymywaniu w pamięci „katalogu”. Program ładujący korzysta z dwóch struktur. Są to:

- a) Katalog. Ta struktura składa się z elementów o różnej długości postaci

B	SPACE	LINK	B	SPACE	LINK
B	SUB1	SUB2	B	SUB1	SUB2
:				:	
B	SUBn	0	B	SUB(n-1)	SUBn

lub

gdzie SPACE jest liczbą słów pamięci zajmowaną przez podprogram; LINK jest dowiązaniem do pozycji katalogu odpowiadającej następnemu podprogramowi na taśmie; SUB1, SUB2, ..., SUBn ($n \geq 0$) są dowiązaniami do pozycji katalogu odpowiadających podprogramom, z których korzysta bieżący podprogram; B = 0 dla wszystkich słów poza ostatnim, B = -1 dla ostatniego słowa w elemencie. Adres pozycji katalogu odpowiadającej pierwszemu podprogramowi na taśmie znajduje się w zmiennej wskaźnikowej FIRST.

b) Lista podprogramów do załadowania, do których bezpośrednio odwołuje się program główny. Lista jest przechowywana w kolejnych lokacjach X[1], X[2], ..., X[N], gdzie N ≥ 0 jest zmienną znaną programowi ładującemu. Każdy element tej listy jest dowiązaniem do pozycji katalogu odpowiadającej potrzebnemu podprogramowi.

Program ładujący posługuje się także wartością MLOC – przesunięciem dla pierwszego ładowanego podprogramu.

Dla przykładu rozważmy następującą konfigurację:

Katalog			Lista potrzebnych podprogramów	
	B	SPACE	LINK	X[1] = 1003
1000:	0	20	1005	X[2] = 1010
1001:	-1	1002	0	
1002:	-1	30	1010	N = 2
1003:	0	200	1007	FIRST = 1002
1004:	-1	1000	1006	MLOC = 2400
1005:	-1	100	1003	
1006:	-1	60	1000	
1007:	0	200	0	
1008:	0	1005	1002	
1009:	-1	1006	0	
1010:	-1	20	1006	

W tym przypadku z katalogu widać, że podprogramom na taśmie odpowiadają elementy 1002, 1010, 1006, 1000, 1005, 1003 i 1007, w tym porządku. Podprogram 1007 zajmuje 200 lokacji i wymaga załadowania podprogramów 1005, 1002 i 1006; itd. Program główny wymaga załadowania podprogramów 1003 i 1010, które należy umieścić w lokacjach ≥ 2400 . Użycie tych podprogramów wymaga z kolei załadowania podprogramów 1000, 1006 i 1002.

Podprogram przydziału pamięci ma tak zmodyfikować strukturę X, że każdy element X[1], X[2], ... (poza elementem ostatnim, który opisujemy dalej) ma być postaci

+	0	BASE	SUB
---	---	------	-----

gdzie SUB jest podprogramem do załadowania, a BASE jest wielkością relokacji. Te pozycje mają być ustawione w tej kolejności, co podprogramy na taśmie. Jednym z poprawnych wyników dla powyższych danych jest

BASE	SUB	BASE	SUB
X[1]: 2400	1002	X[4]: 2510	1000
X[2]: 2430	1010	X[5]: 2530	1003
X[3]: 2450	1006	X[6]: 2730	0

Ostatnia pozycja powinna zawierać pierwszy wolny adres za załadowanymi podprogramami.

(To oczywiście nie jest jedyna możliwość korzystania z biblioteki podprogramów. Poprawny sposób zaprojektowania biblioteki silnie zależy od komputera i konkretnych zastosowań. Duże współczesne komputery wymagają zupełnie innego podejścia, ale to nie umniejsza zalet tego ćwiczenia, które wymaga jednoczesnego manipulowania dwoma rodzajami struktur danych: sekwencyjnymi oraz z dowiązaniami).

Ćwiczenie polega na zaprojektowaniu algorytmu realizującego opisane zadanie. Twój podprogram przydziału pamięci, przygotowując odpowiedź, może dowolnie modyfikować katalog, ponieważ katalog można ponownie wczytać z taśmy przy kolejnym uruchomieniu podprogramu przydziału pamięci, a w pozostałych częściach programu ładującego nie korzysta się z katalogu.

27. [25] Napisz program w asemblerze komputera MIX realizujący algorytm przydziału pamięci z ćwiczenia 26.

28. [40] Z poniższego opisu wynika, w jaki sposób można „rozwiązywać” gry dla dwóch osób, włączając szachy, nim i wiele prostszych gier. Rozważmy skończony zbiór elementów, z których każdy reprezentuje możliwą sytuację w grze. Dla każdej sytuacji istnieje zero lub więcej ruchów przekształcających ją w pewną inną sytuację. Mówimy, że sytuacja x jest poprzedniczką sytuacji y (oraz, że y jest następczną x), jeżeli istnieje ruch przeprowadzający x w y . Niektóre sytuacje, które nie mają następczyni, są uważane za *wygrane* lub *przegrane*. Gracz, do którego należy ruch zakończony sytuacją x , jest przeciwnikiem gracza, którego ruch zakończy się sytuacją będącą następczną x .

Mając dany układ sytuacji, możemy obliczyć zbiór wszystkich sytuacji wygrywających (tj. takich, że gracz, do którego należy ruch, ma zagwarantowane zwycięstwo) oraz zbiór wszystkich sytuacji przegrywających (tj. takich, że gracz, do którego należy ruch, nie może wygrać, jeżeli przeciwnik nie popełni błędu). Wyznaczenie zbiorów polega na powtarzaniu poniższej operacji, dopóki wprowadza ona jeszcze jakieś zmiany: Oznaczamy jako „przegrane” wszystkie te sytuacje, których następczynie są oznaczone jako „wygrane”; oznaczamy sytuację jako „wygraną”, jeżeli ma przynajmniej jedną następczynię oznaczoną jako „przeganą”.

Od pewnego momentu powtarzanie operacji nic nowego nie wnosi. Mogą jednak pozostać sytuacje nie oznaczone ani jako wygrane, ani jako przegrane. Gracz startujący z takiej sytuacji nie ma ani gwarancji wygranej, ani nie jest skazany na porażkę.

Powyższa procedura znajdowania zbiorów sytuacji wygranych i przegranych może być zapisana w postaci wydajnego algorytmu przypominającego algorytm T. Możemy

dla każdej sytuacji przechowywać liczbę jej poprzedniczek, które zostały oznaczone jako „wygrywające”, oraz listę jej następczyń.

Ćwiczenie polega na przemyśleniu szczegółów zarysowanego algorytmu i zastosowaniu go do ciekawych gier, w których nie występuje zbyt wiele sytuacji [jak np. „Gra Wojenna”: É. Lucas, *Récréations Mathématiques* 3 (Paris: 1893) 105–116; E. R. Berlekamp, J. H. Conway, R. K. Guy, *Winning Ways* 2 (Academic Press, 1982), rozdział 21].

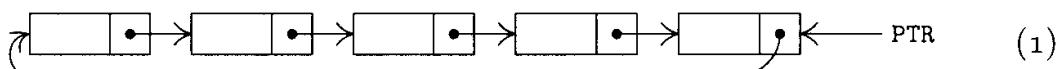
- 29. [21] (a) Podaj algorytm „usuwania” całej listy, takiej jak (1), polegający na włożeniu jej elementów na stos **AVAIL**. Daną wejściową jest wartość **FIRST**. Algorytm powinien być jak najszybszy. (b) Podaj analogiczny algorytm dla takiej listy, jak (12), gdy danymi wejściowymi są **F** i **R**.
- 30. [17] Przypuśćmy, że kolejki są reprezentowane tak, jak (12), ale z tą różnicą, że kolejka pusta jest reprezentowana przez $F = \Lambda$ i R niezdefiniowane. Jakich operacji wstawiania i usuwania elementu należy używać zamiast (14) i (17)?

2.2.4. Listy cykliczne

Niewielka zmiana w schemacie dowiązań prowadzi do nowej struktury danych.

Lista cykliczna z dowiązaniemi (lub krótko: lista cykliczna) ma tę własność, że jej ostatni element zawiera dowiązanie do elementu pierwszego zamiast Λ . Z dowolnego elementu listy można się dzięki temu dostać do wszystkich jej elementów. Uzyskujemy także dodatkową symetrię, jeżeli nie interesują nas pojęcia pierwszego i ostatniego elementu listy.

Oto typowy układ:



Założymy, że elementy mają dwa pola **INFO** i **LINK**, jak w poprzednim rozdziale. Zmienna dowiązaniowa (wskaźnikowa) **PTR** zawiera dowiązanie do skrajnie prawnego elementu, a **LINK(PTR)** jest adresem skrajnie lewego elementu listy. Oto najważniejsze operacje na listach cyklicznych:

- Wstaw **Y** z lewej strony: $P \leftarrow \text{AVAIL}$, $\text{INFO}(P) \leftarrow Y$, $\text{LINK}(P) \leftarrow \text{LINK}(\text{PTR})$, $\text{LINK}(\text{PTR}) \leftarrow P$.
- Wstaw **Y** z prawej strony: wstaw **Y** z lewej strony, następnie $\text{PTR} \leftarrow P$.
- Ustaw **Y** na lewy element i usuń go z listy: $P \leftarrow \text{LINK}(\text{PTR})$, $Y \leftarrow \text{INFO}(P)$, $\text{LINK}(\text{PTR}) \leftarrow \text{LINK}(P)$, $\text{AVAIL} \leftarrow P$.

Operacja (b) jest na pierwszy rzut oka trochę zaskakująca. Wykonanie przypisania $\text{PTR} \leftarrow \text{LINK}(\text{PTR})$ powoduje przesunięcie skrajnie lewego elementu w prawo (przy układzie jak na rysunku (1)). Nietrudno to zrozumieć, gdy uświadomimy sobie, że listę należy postrzegać jako pętlę, a nie sznurek ze związanymi końcami.

Uważny Czytelnik spostrzeże, że w operacjach (a), (b) i (c) popełniliśmy poważny błąd. Jaki? *Odpowiedź:* zapomnieliśmy o przypadku, kiedy lista jest *pusta*. Jeśli na liście wykonamy na przykład operację (c) pięć razy (1), to **PTR** będzie wskazywał na element znajdujący się na liście **AVAIL**, a to może prowadzić do poważnych kłopotów. Lepiej nie myśleć, co by się mogło stać, gdybyśmy operację (c) wykonali jeszcze raz! Jeśli przyjmiemy, że **PTR** ma się równać Λ ,

gdy lista jest pusta, to możemy zaradzić problemom, wstawiając dodatkową sekwencję rozkazów „jeśli $\text{PTR} = \Lambda$, to $\text{PTR} \leftarrow \text{LINK}(P) \leftarrow P$; w przeciwnym razie ...” po „ $\text{INFO}(P) \leftarrow Y$ ” w (a); poprzedzając (c) sprawdzeniem warunku „jeśli $\text{PTR} = \Lambda$, to **NIEDOMIAR**” oraz dopisując na końcu (c) „jeśli $\text{PTR} = P$, to $\text{PTR} \leftarrow \Lambda$ ”.

Zauważmy, że operacje (a), (b) i (c) składają się na zestaw operacji kolejki dwustronnej o ograniczonym wyjściu w sensie określonym w punkcie 2.2.1. To pozwala w szczególności wywnioskować, że listę cykliczną można wykorzystywać jako stos lub kolejkę. Operacje (a) i (c) stanowią obsługę stosu, a operacje (b) i (c) – kolejki. Te operacje są tylko trochę mniej oczywiste niż ich odpowiedniki z poprzedniego podrozdziału, gdzie operacje (a), (b) i (c) były realizowane na listach liniowych z dwoma wskaźnikami F i R .

Okaże się, że listy cykliczne umożliwiają wydajną realizację innych ważnych operacji. Na przykład bardzo wygodnie jest „skasować” listę, tj. włożyć jednym posunięciem wszystkie jej elementy na stos **AVAIL**.

Jeśli $\text{PTR} \neq \Lambda$, to $\text{AVAIL} \leftrightarrow \text{LINK}(\text{PTR})$. (2)

[Przypominamy, że „ \leftrightarrow ” oznacza operację wymiany: $P \leftarrow \text{AVAIL}$, $\text{AVAIL} \leftarrow \text{LINK}(\text{PTR})$, $\text{LINK}(\text{PTR}) \leftarrow P$.] Operacja (2) jest poprawna, gdy PTR wskazuje na *którykolwiek* element listy cyklicznej. Na koniec przypisujemy oczywiście $\text{PTR} \leftarrow \Lambda$.

Używając podobnej metody, gdy PTR_1 i PTR_2 wskazują na rozłączne listy cykliczne L_1 i L_2 , możemy wstawić całą listę L_2 na prawy koniec L_1 :

Jeśli $\text{PTR}_2 \neq \Lambda$, to

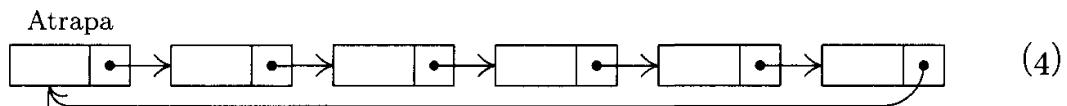
(jeśli $\text{PTR}_1 \neq \Lambda$, to $\text{LINK}(\text{PTR}_1) \leftrightarrow \text{LINK}(\text{PTR}_2)$;(3)
 $\text{PTR}_1 \leftarrow \text{PTR}_2$, $\text{PTR}_2 \leftarrow \Lambda$).

Podział listy cyklicznej na dwie listy to kolejna prosta operacja, którą przy użyciu tych struktur danych można wygodnie realizować na wiele sposobów. Ta operacja odpowiada konkatenacji i rozbijaniu napisów.

Widzimy zatem, że lista cykliczna może służyć nie tylko do reprezentowania struktur, które są ze swej natury cykliczne, ale także do reprezentowania struktur liniowych. Lista cykliczna z jednym wskaźnikiem do ostatniego elementu jest w istocie równoważna zwykłej liście liniowej z dwoma wskaźnikami, do pierwszego i ostatniego elementu. Nasuwające się pytanie to „jak odnaleźć koniec listy, skoro mamy «cykliczną» symetrię?” Końca listy nie oznacza dowiezanie Λ ! Odpowiedź brzmi, że jeżeli wykonujemy operacje na całej liście, przeglądając kolejne elementy, to powinniśmy zatrzymać się w tym miejscu, gdzie zaczeliśmy (jeśli oczywiście „to miejsce” jeszcze się na liście znajduje).

Inne rozwiązanie polega na wstawieniu do listy specjalnego, łatwego do rozpoznania elementu, który wskaże miejsce zatrzymania. Ten specjalny element jest nazywany *atrapą*, a w praktyce okazuje się, że przydaje się utrzymywanie dyscypliny wymagającej, by każda lista cykliczna miała dokładnie jedną atrapę. Jedną z zalet stosowania takiego dodatkowego elementu jest fakt, że przy takim

założeniu listy nigdy nie będą puste. Jeżeli wymagamy, by lista zawierała atrapę, diagram (1) powinien przyjąć postać



Odwołania do listy, takiej jak (4), zazwyczaj realizuje się „poprzez atrapę”, która często znajduje się w ustalonej lokacji. Wadą takiego rozwiązania jest to, że nie mamy wskaźnika do prawego końca, więc musimy odżalaować operację (b).

Diagram (4) można porównać z diagramem 2.2.3–(1) z początku poprzedniego punktu, w którym wskaźnik w elemencie 5 wskazuje teraz na $\text{LOC}(\text{FIRST})$, a nie Λ ; o zmiennej FIRST myślimy teraz jako o dowiązaniu w elemencie, konkretnie dowiązaniu zapisanym w $\text{NODE}(\text{LOC}(\text{FIRST}))$. Zasadnicza różnica między (4) a 2.2.3–(1) polega na tym, że (4) pozwala (choć niekoniecznie efektywnie) poprzez ciąg wskaźników dostać się z dowolnego elementu listy do dowolnego innego jej elementu.

Jako przykład wykorzystania list cyklicznych omówimy *działania na wielomianach* trzech zmiennych (x, y, z) o współczynnikach całkowitych. Jest wiele problemów, w których wygodniej posługiwać się wielomianami niż po prostu liczbami. Chcemy dysponować podprogramami, które mogą wymnożyć

$$(x^4 + 2x^3y + 3x^2y^2 + 4xy^3 + 5y^4) \quad \text{przez} \quad (x^2 - 2xy + y^2),$$

dając wynik:

$$(x^6 - 6xy^5 + 5y^6).$$

Użycie struktur z dowiązaniami jest naturalnym sposobem realizacji operacji tego typu, ponieważ rozmiar wielomianów może się zmieniać, przy tym chcielibyśmy jednocześnie przechowywać w pamięci reprezentacje wielu wielomianów.

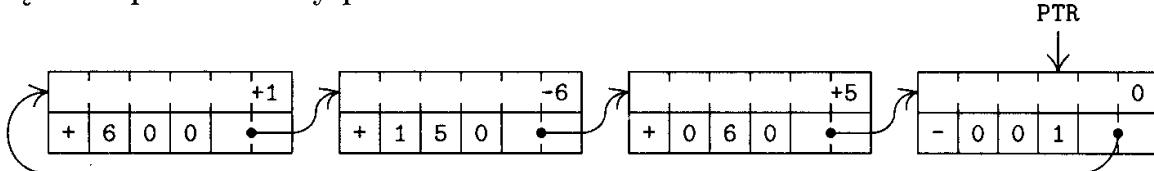
Zajmiemy się dwoma działaniami: dodawaniem i mnożeniem. Przypuśćmy, że wielomian jest reprezentowany za pomocą listy, której każdy element reprezentuje niezerowy wyraz. Element składa się z dwóch słów postaci:

COEF				
\pm	A	B	C	LINK

(5)

COEF jest współczynnikiem stojącym przy $x^A y^B z^C$. Będziemy zakładać, że współczynniki i wykładniki naszych wielomianów należą do przedziałów dających się reprezentować w powyższym formacie i że podczas obliczeń nie trzeba sprawdzać, czy takie założenia spełniają wyniki działań. Zapis ABC będzie oznaczał pola $\pm A B C$ elementu (5) traktowane jako całość. Znak ABC, tj. znak drugiego słowa (5), będzie zawsze dodatni, poza elementem specjalnym stojącym na końcu każdej listy reprezentującej wielomian, dla którego $ABC = -1$ i $\text{COEF} = 0$. Dzięki obecności takiego elementu, analogicznego do niedawno omawianej atrapy, o wiele łatwiej będzie posługiwać się listą. Element specjalny posłuży za wartownika, a także pozwoli na uniknięcie problemów z listą pustą (reprezentującą wielomian zerowy). Elementy listy zawsze są ustawione w porządku malejącym względem

pól ABC, gdy patrzymy na listę zgodnie z kierunkiem dowiązań. Odstępstwem od tej reguły jest element specjalny (dla którego $ABC = -1$) zawierający dowiązanie do elementu o największej wartości ABC. Na przykład wielomian $x^6 - 6xy^5 + 5y^6$ będzie reprezentowany przez:



Algorytm A (*Dodawanie wielomianów*). Algorytm dodaje dwa wielomiany reprezentowane w postaci opisanej powyżej, wskazywane przez zmienne wskaźnikowe P i Q. Wielomiany te będąemy oznaczać odpowiednio $\text{polynomial}(P)$ oraz $\text{polynomial}(Q)$. Lista P pozostanie niezmieniona, lista Q będzie zawierała sumę. Zmienne P i Q po wykonaniu algorytmu mają na powrót wartości początkowe. Wykorzystujemy pomocnicze zmienne wskaźnikowe Q1 i Q2.

- A1. [Inicjowanie] Przyjmij $P \leftarrow \text{LINK}(P)$, $Q1 \leftarrow Q$, $Q \leftarrow \text{LINK}(Q)$. (W tym momencie P i Q wskazują na początkowe elementy reprezentujące wyrazy wielomianów. Przez większą część wykonania algorytmu zmienna Q1 będzie podążała jeden krok za Q w tym sensie, że $Q = \text{LINK}(Q1)$).
- A2. [$\text{ABC}(P) : \text{ABC}(Q)$] Jeśli $\text{ABC}(P) < \text{ABC}(Q)$, to przyjmij $Q1 \leftarrow Q$ i $Q \leftarrow \text{LINK}(Q)$ i powtóż ten krok. Jeśli $\text{ABC}(P) = \text{ABC}(Q)$, to idź do kroku A3. Jeśli $\text{ABC}(P) > \text{ABC}(Q)$, to idź do kroku A5.
- A3. [Dodawanie współczynników] (Znaleźliśmy wyrazy podobne). Jeśli wartość $\text{ABC}(P) < 0$, to algorytm się zatrzymuje. W przeciwnym razie przyjmij $\text{COEF}(Q) \leftarrow \text{COEF}(Q) + \text{COEF}(P)$. Jeśli $\text{COEF}(Q) = 0$, to idź do A4; w przeciwnym razie przyjmij $P \leftarrow \text{LINK}(P)$, $Q1 \leftarrow Q$, $Q \leftarrow \text{LINK}(Q)$ i idź do A2. (Co ciekawe, te ostatnie operacje są takie same, jak w kroku A1).
- A4. [Usuwanie wyrazu zerowego] Przyjmij $Q2 \leftarrow Q$, $\text{LINK}(Q1) \leftarrow Q \leftarrow \text{LINK}(Q)$ i $\text{AVAIL} \leftarrow Q2$. (Wyraz zerowy utworzony w kroku A3 został usunięty z $\text{polynomial}(Q)$). Przyjmij $P \leftarrow \text{LINK}(P)$ i wróć do kroku A2.
- A5. [Wstawianie nowego wyrazu] ($\text{Polynomial}(P)$ zawiera wyraz, którego nie ma w $\text{polynomial}(Q)$, zatem wstawiamy go do $\text{polynomial}(Q)$). Przyjmij $Q2 \leftarrow \text{AVAIL}$, $\text{COEF}(Q2) \leftarrow \text{COEF}(P)$, $\text{ABC}(Q2) \leftarrow \text{ABC}(P)$, $\text{LINK}(Q2) \leftarrow Q$, $\text{LINK}(Q1) \leftarrow Q2$, $Q1 \leftarrow Q2$, $P \leftarrow \text{LINK}(P)$ i wróć do kroku A2. ■

Jedną z wartych odnotowania cech algorytmu A jest sposób, w jaki zmienna wskaźnikowa Q1 podąża za Q po liście cyklicznej. Jest to bardzo typowe dla algorytmów przetwarzania list i zobaczymy jeszcze całą masę podobnych algorytmów. Czy Czytelnik rozumie pomysł wykorzystany w algorytmie A?

Czytelnikom, którzy nie mają dużego doświadczenia w posługiwaniu się listami, proponujemy bardzo uważne prześledzenie algorytmu A. Dobrym pomysłem jest przeanalizowanie zachowania algorytmu na przykładzie sumowania wielomianów $x + y + z$ oraz $x^2 - 2y - z$.

Dysponując algorytmem A, algorytm mnożenia wielomianów możemy zapisać zadziwiająco prosto:

Algorytm M (Mnożenie wielomianów). Algorytm zastępuje $\text{polynomial}(Q)$ wielomianem

$$\text{polynomial}(Q) + \text{polynomial}(M) \times \text{polynomial}(P).$$

M1. [Następny mnożnik] Przyjmij $M \leftarrow \text{LINK}(M)$. Jeśli $\text{ABC}(M) < 0$, to algorytm się zatrzymuje.

M2. [Cykl mnożenia] Wykonaj algorytm A, z tym że każde wystąpienie „ $\text{ABC}(P)$ ” w opisie tego algorytmu zamień na „jeśli $\text{ABC}(P) < 0$ to -1 , w przeciwnym razie $\text{ABC}(P) + \text{ABC}(M)$ ”, a także każde wystąpienie „ $\text{COEF}(P)$ ” zamień na „ $\text{COEF}(P) \times \text{COEF}(M)$ ”. Następnie wróć do kroku M1. ■

Zapisując algorytm A w języku komputera MIX, po raz kolejny zauważymy prostotę obsługi list z dowiązaniami. W poniższym kodzie zakładamy, że **OVERFLOW** jest podprogramem, który albo kończy wykonanie programu (z powodu braku pamięci), albo znajduje wolną pamięć i wychodzi do lokacji $rJ - 2$.

Program A (Dodawanie wielomianów). Ten podprogram jest napisany w taki sposób, by można go było wykorzystać w podprogramie mnożenia (zobacz ćwiczenie 15).

Sekwencja wywołania: **JMP ADD**

Warunki wejściowe: $rI1 = P, rI2 = Q$.

Warunki wyjściowe: $\text{polynomial}(Q)$ zostanie zastąpiony przez $\text{polynomial}(Q) + \text{polynomial}(P)$; $rI1$ i $rI2$ pozostają nie naruszone; pozostałe rejestrów mają nieokreśloną zawartość.

W poniższym kodzie $P \equiv rI1, Q \equiv rI2, Q1 \equiv rI3$ i $Q2 \equiv rI6$ zgodnie z oznaczeniami z opisu algorytmu A.

01	LINK	EQU	4:5	Definicja pola LINK .
02	ABC	EQU	0:3	Definicja pola ABC .
03	ADD	STJ	3F	1 Wejście do podprogramu.
04	1H	ENT3	0,2	1 + m'' <u>A1. Inicjowanie.</u> Przyjmij $Q1 \leftarrow Q$.
05		LD2	1,3(LINK)	1 + m'' $Q \leftarrow \text{LINK}(Q1)$.
06	0H	LD1	1,1(LINK)	1 + p $P \leftarrow \text{LINK}(P)$.
07	SW1	LDA	1,1	1 + p $rA(0:3) \leftarrow \text{ABC}(P)$.
08	2H	CMPA	1,2(ABC)	x <u>A2. ABC(P) : ABC(Q)</u> .
09		JE	3F	x Jeśli równe, to idź do A3.
10		JG	5F	$p' + q'$ Jeśli większe, to idź do A5.
11		ENT3	0,2	q' Jeśli mniejsze, to przyjmij $Q1 \leftarrow Q$.
12		LD2	1,3(LINK)	q' $Q \leftarrow \text{LINK}(Q1)$.
13		JMP	2B	q' Powtórz.
14	3H	JAN	*	$m + 1$ <u>A3. Dodawanie współczynników.</u>
15	SW2	LDA	0,1	m $\text{COEF}(P)$
16		ADD	0,2	m $+ \text{COEF}(Q)$
17		STA	0,2	m $\rightarrow \text{COEF}(Q)$.
18		JANZ	1B	m Skok, gdy nie zero.

19	ENT6 0,2	m'	<u>A4. Usuwanie wyrazu zerowego.</u> $Q2 \leftarrow Q$.
20	LD2 1,2(LINK)	m'	$Q \leftarrow \text{LINK}(Q)$.
21	LDX AVAIL	m'	
22	STX 1,6(LINK)	m'	$\} \quad \text{AVAIL} \Leftarrow Q2$.
23	ST6 AVAIL	m'	
24	ST2 1,3(LINK)	m'	$\text{LINK}(Q1) \leftarrow Q$.
25	JMP OB	m'	Skocz i przesuń P.
26 5H	LD6 AVAIL	p'	<u>A5. Wstawianie nowego wyrazu.</u>
27	J6Z OVERFLOW	p'	
28	LDX 1,6(LINK)	p'	$Q2 \Leftarrow \text{AVAIL}$.
29	STX AVAIL	p'	
30	STA 1,6	p'	$\text{ABC}(Q2) \leftarrow \text{ABC}(P)$.
31 SW3	LDA 0,1	p'	$rA \leftarrow \text{COEF}(P)$.
32	STA 0,6	p'	$\text{COEF}(Q2) \leftarrow rA$.
33	ST2 1,6(LINK)	p'	$\text{LINK}(Q2) \leftarrow Q$.
34	ST6 1,3(LINK)	p'	$\text{LINK}(Q1) \leftarrow Q2$.
35	ENT3 0,6	p'	$Q1 \leftarrow Q2$.
36	JMP OB	p'	Skocz i przesuń P. ■

Zauważmy, że algorytm A przechodzi każdą z dwóch list tylko jednokrotnie. Nie ma potrzeby po kilka razy kręcić się w kółko. Korzystając z prawa Kirchhoffa, odkrywamy, że zliczenie wykonan rozkazów nie nastręcza trudności; czas wykonania zależy od czterech wielkości:

m' = liczba wyrazów podobnych, które się redukują;

m'' = liczba wyrazów podobnych, które się nie redukują;

p' = liczba wyrazów wielomianu $\text{polynomial}(P)$ nie mających odpowiedników;

q' = liczba wyrazów wielomianu $\text{polynomial}(Q)$ nie mających odpowiedników.

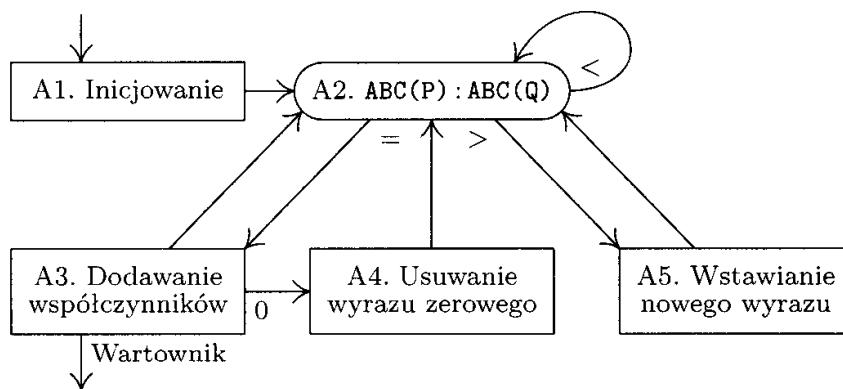
Wśród liczby wykonan rozkazów podanych w tekście programu A skorzystaliśmy z następujących skrótów:

$$m = m' + m'', \quad p = m + p', \quad q = m + q', \quad x = 1 + m + p' + q';$$

Czas wykonania na komputerze MIX wynosi $(27m' + 18m'' + 27p' + 8q' + 13)u$. Całkowita liczba elementów na stercie wykorzystywanych podczas pracy programu to przynajmniej $2 + p + q$ i co najwyżej $2 + p + q + p'$.

ĆWICZENIA

- [21] Tekst na początku tego rozdziału sugeruje, że pustą listę cykliczną można reprezentować za pomocą $\text{PTR} = \Lambda$. Może bardziej zgodne z filozofią listy cyklicznej byłoby oznaczanie listy pustej za pomocą $\text{PTR} = \text{LOC}(\text{PTR})$? Czy taka konwencja umożliwia uproszczenie operacji (a), (b) i (c) opisanych na początku rozdziału?
- [20] Narysuj schematy „przed i po”, ilustrujące skutki operacji konkatenacji (3) przy założeniu, że PTR_1 i PTR_2 są $\neq \Lambda$.
- ▶ 3. [20] Co robi operacja (3), gdy PTR_1 i PTR_2 wskazują na elementy *tej samej* listy cyklicznej?
4. [20] Zapisz operacje wstawiania i usuwania, które umożliwiają korzystanie z listy cyklicznej w reprezentacji (4) jak ze stosu.



Rys. 10. Dodawanie wielomianów.

- ▶ 5. [21] Zaprojektuj algorytm, który w liście cyklicznej w postaci (1) odwraca kierunek strzałek (dowiązań).
- 6. [18] Narysuj schematy list reprezentujących wielomiany (a) $xz - 3$; (b) 0.
- 7. [10] Dlaczego warto zakładać, że pola ABC na liście reprezentującej wielomian są uporządkowane malejąco?
- ▶ 8. [10] Po co w algorytmie A wskaźnik Q1 posuwa się o jeden krok za Q?
- ▶ 9. [23] Czy algorytm A działałby poprawnie, gdyby $P = Q$ (tj. obie zmienne wskazywały ten sam wielomian)? Czy algorytm M działałby poprawnie, gdyby $P = M$, gdyby $P = Q$ lub gdyby $M = Q$?
- ▶ 10. [20] W algorytmach w tym punkcie korzysta się z założenia, że w wielomianach występują trzy zmienne x , y i z , a ich wykładniki nigdy nie przekraczają $b - 1$ (gdzie b jest rozmiarem bajtu komputera MIX). Przypuśćmy, że chcemy dodawać i mnożyć wielomiany tylko jednej zmiennej x , ale chcemy by wykładniki mogły przyjmować wartości do $b^3 - 1$. Jakie zmiany należy wprowadzić w algorytmach A i M?
- 11. [24] (Celem tego i wielu następnych ćwiczeń jest stworzenie pakietu podprogramów do wykonywania działań na wielomianach, w połączeniu z programem A). Ponieważ algorytm A i M zmieniają wartość $\text{polynomial}(Q)$, przydałby się czasem podprogram kopiący wielomian. Napisz podprogram w asemblerze komputera MIX zgodny z następującą specyfikacją:

Sekwencja wywołania: JMP COPY

Warunki wejściowe: rI1 = P

Warunki wyjściowe: rI2 wskazuje na nowo utworzony wielomian równy wielomianowi $\text{polynomial}(P)$; rI1 pozostaje niezmienione; wartości w innych rejestrach nie są zdefiniowane.

- 12. [21] Porównaj czas działania programu z ćwiczenia 11 z czasem działania programu A, gdy $\text{polynomial}(Q) = 0$.

- 13. [20] Napisz podprogram w asemblerze komputera MIX zgodny z następującą specyfikacją:

Sekwencja wywołania: JMP ERASE

Warunki wejściowe: rI1 = P

Warunki wyjściowe: $\text{polynomial}(P)$ został dołączony do listy AVAIL; zawartości wszystkich rejestrów są nie zdefiniowane.

[Uwaga: Tego podprogramu można użyć w połączeniu z podprogramem z ćwiczenia 11, by za pomocą ciągu rozkazów „LD1 Q; JMP ERASE; LD1 P; JMP COPY; ST2 Q” otrzymać „polynomial(Q) \leftarrow polynomial(P)’].

14. [22] Napisz podprogram w asemblerze komputera MIX zgodny z następującą specyfikacją:

Sekwencja wywołania: JMP ZERO

Warunki wejściowe: Brak

Warunki wyjściowe: rI2 wskazuje na nowo utworzony wielomian zerowy; zawartości pozostałych rejestrów nie są zdefiniowane.

15. [24] Napisz podprogram w asemblerze komputera MIX realizujący algorytm M, zgodny z następującą specyfikacją:

Sekwencja wywołania: JMP MULT

Warunki wejściowe: rI1 = P, rI2 = Q, rI4 = M.

Warunki wyjściowe: $\text{polynomial}(Q) \leftarrow \text{polynomial}(Q) + \text{polynomial}(M) \times \text{polynomial}(P)$; rI1, rI2, rI4 pozostają niezmienione; zawartości pozostałych rejestrów nie są zdefiniowane.

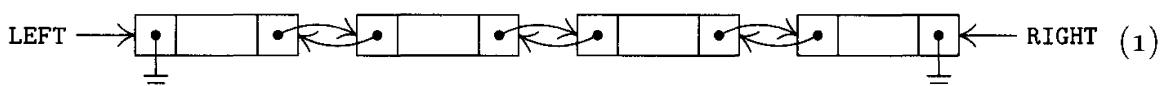
(Uwaga: Skorzystaj z programu A jako podprogramu, zmieniając SW1, SW2 i SW3).

16. [M28] Oszacuj czas wykonania podprogramu z ćwiczenia 15 względem istotnych parametrów.

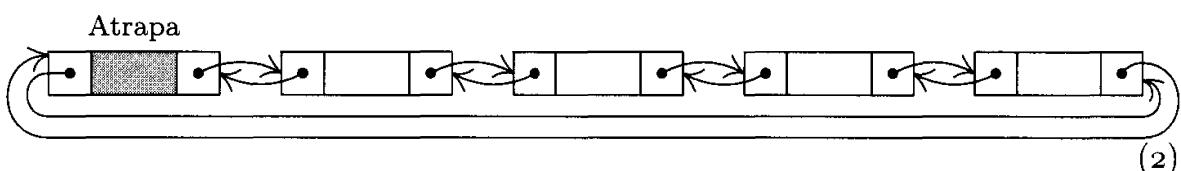
- **17.** [22] Dlaczego reprezentujemy wielomiany za pomocą list cyklicznych, zamiast użyć zwykłych list liniowych zakończonych dowiązaniem Λ (opisanych w poprzednim punkcie)?
- **18.** [25] Zaproponuj taką reprezentację list cyklicznych w pamięci komputera, żeby można je było efektywnie przeglądać w obu kierunkach, ale by na jeden element przypadało tylko jedno dowiązanie. [Wskazówka: Mając wskaźniki do dwóch kolejnych elementów x_{i-1} i x_i , powinniśmy być w stanie zlokalizować zarówno x_{i+1} , jak i x_{i-2}].

2.2.5. Listy dwukierunkowe

Jeszcze większą elastyczność w manipulowaniu listami liniowymi możemy osiągnąć, utrzymując w każdym elemencie po dwa dowiązania, do obu sąsiadów:



LEFT i RIGHT są zmiennymi zawierającymi dowiązania do lewego i prawego końca listy. Taka reprezentacja umożliwia łatwą realizację operacji kolejki dwustronnej, zobacz ćwiczenie 1. Operacje na liście dwukierunkowej prawie zawsze są dużo łatwiejsze, gdy lista zawiera atrapę (porównaj z analogiczną metodą opisaną w poprzednim punkcie dotyczącym list cyklicznych). Typowy schemat listy dwukierunkowej z atrapą wygląda następująco:



Pola RLINK i LLINK atrapy pełnią funkcję LEFT i RIGHT z (1). Istnieje pełna symetria między lewą a prawą stroną; na schemacie (2) atrapa mogłaby równie

dobrze być narysowana po prawej stronie. Gdy lista jest pusta, oba wskaźniki atrapy wskazują na atrapę.

Dla reprezentacji (2) spełniony jest warunek

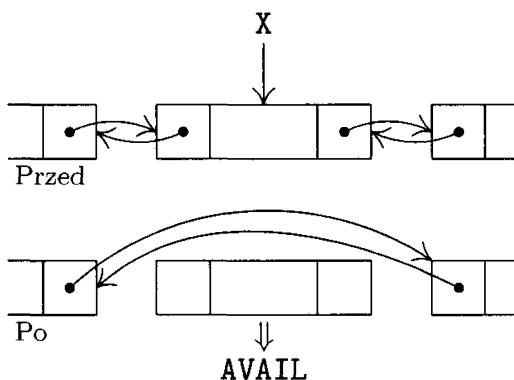
$$\text{RLINK}(\text{LLINK}(X)) = \text{LLINK}(\text{RLINK}(X)) = X, \quad (3)$$

gdy X jest lokacją dowolnego elementu listy (również atrapy). Ten fakt jest podstawowym powodem, dla którego reprezentacja (2) jest wygodniejsza niż (1).

Lista dwukierunkowa zazwyczaj zajmuje więcej pamięci niż lista jednokierunkowa (choć czasami w elemencie, który nie zajmuje całego słowa, jest miejsce na drugie dowiązanie). Ale dodatkowe operacje, które można wydajnie wykonywać na liście dwukierunkowej są zazwyczaj więcej niż skromną rekompensatą za poświęcenie kawałka pamięci. Poza oczywistą korzyścią polegającą na tym, że po liście dwukierunkowej można się poruszać w przód i w tył, jesteśmy w stanie usunąć z listy element $\text{NODE}(X)$, dysponując jedynie wartością X . Tę prostą operację łatwo wysnuć z diagramu „przed i po” (rysunek 11):

$$\begin{aligned} \text{RLINK}(\text{LLINK}(X)) &\leftarrow \text{RLINK}(X), & \text{LLINK}(\text{RLINK}(X)) &\leftarrow \text{LLINK}(X), \\ && \text{AVAIL} &\Leftarrow X. \end{aligned} \quad (4)$$

Z listy, która ma wyłącznie dowiązania jednokierunkowe, nie możemy usunąć elementu $\text{NODE}(X)$, nie wiedząc, który element go poprzedza, ponieważ usunięcie $\text{NODE}(X)$ oznacza konieczność modyfikacji jego poprzednika. We wszystkich algorytmach omawianych w punktach 2.2.3 i 2.2.4 w chwili usuwania elementu dysponowaliśmy tą informacją. W szczególności w algorytmie 2.2.4A wskaźnik Q_1 podąża za wskaźnikiem Q właśnie z tego powodu. Natkniemy się jednak na algorytmy, które wymagają usuwania dowolnych elementów ze środka listy. W takich właśnie sytuacjach korzystamy z list dwukierunkowych. (Należy zauważyć, że dysponując wyłącznie wskaźnikiem X , można usunąć $\text{NODE}(X)$ z jednokierunkowej listy cyklicznej, jeżeli przesuwając się po liście, znajdziemy poprzednik X . Ta operacja jest jednak bardzo mało wydajna, gdy lista jest dłuża, stąd rzadko jesteśmy w stanie zaakceptować takie rozwiązanie zastępcze. Porównaj odpowiedź do ćwiczenia 2.2.4–8).



Rys. 11. Usuwanie elementu z listy dwukierunkowej.

Na podobnej zasadzie można łatwo wstawić nowy element do listy cyklicznej obok danego elementu $\text{NODE}(X)$, zarówno z lewej, jak i z prawej strony. Wsta-

wienie z prawej strony elementu $\text{NODE}(X)$ uzyskamy, wykonując

$$\begin{aligned} P &\leftarrow \text{AVAIL}, & \text{LLINK}(P) &\leftarrow X, & \text{RLINK}(P) &\leftarrow \text{RLINK}(X), \\ && \text{LLINK}(\text{RLINK}(X)) &\leftarrow P, & \text{RLINK}(X) &\leftarrow P. \end{aligned} \quad (5)$$

Zamieniając w powyższych operacjach dowiązanie lewe z prawym, otrzymamy operację wstawiania po lewej stronie. Operacja (5) powoduje zmianę wartości pięciu dowiązań, jest zatem nieco wolniejsza niż operacja wstawiania do listy jednokierunkowej, która wymaga modyfikacji tylko trzech dowiązań.

W ramach przykładu wykorzystania listy dwukierunkowej zajmiemy się napisaniem *symulatora dyskretnego*. „Symulacja dyskretna” oznacza symulację, w której zakładamy, że wszystkie zmiany stanu symulowanego systemu zachodzą w określonych chwilach dyskretnie określonego czasu. Symulowany „system” to zazwyczaj zbiór oddzielnych obiektów, które są w dużym stopniu niezależne, ale w pewien sposób na siebie oddziałują. Przykłady: klienci w sklepie, statki w porcie, pracownicy firmy. W symulacji dyskretniej symulujemy wszystkie zdarzenia, które dzieją się w pewnej chwili symulowanego czasu, po czym przesuwamy zegar skokowo do następnej chwili, w której coś się wydarzy.

Inaczej jest w przypadku „symulacji ciągłej”, polegającej na symulowaniu obiektów podlegających bezustannym ciągłym zmianom, jak ruch na autostradzie, statek kosmiczny lecący na inną planetę itp. Zazwyczaj symulację ciągłą można zadowalająco przybliżyć za pomocą symulacji dyskretnej o bardzo małym odstępie między kolejnymi chwilami symulowanego czasu. W takim jednak przypadku mamy do czynienia z „synchroniczną” symulacją dyskretną, w której wiele elementów systemu w każdym kroku symulacji nieznacznie się zmienia. Wymaga to innej organizacji programu niż symulacja dyskretna, którą my się zajmiemy.

Przedstawiony poniżej program symuluje zachowanie windy w budynku Wydziału Matematyki California Institute of Technology. Wyniki tej symulacji przydadzą się zapewne tylko tym, którzy względnie często odwiedzają Caltech. Lecz nawet im pewnie wygodniej będzie parę razy skorzystać z prawdziwej windy niż z poniższego programu. Jednak, jak to zazwyczaj bywa w badaniach nad symulacją, użyte metody są o wiele istotniejsze niż wyniki symulacji. Zaprezentowana metoda demonstruje typowe sposoby realizacji symulatorów dyskretnych.

Budynek Wydziału Matematyki ma pięć kondygnacji: piwnicę, suterenę, parter, pierwsze piętro i drugie piętro. W budynku jest elektronicznie sterowana winda zatrzymująca się na każdej kondygnacji. Dla wygody kondygnacje ponumerujemy liczbami 0, 1, 2, 3 i 4.

Na każdej kondygnacji przy drzwiach windy znajdują się dwa przyciski: w górę (**UP**) oraz w dół (**DOWN**). (W zasadzie na kondygnacji 0 jest tylko **UP**, a na kondygnacji 4 tylko **DOWN**, ale zignorujemy to odstępstwo, zakładając że dodatkowe przyciski nie są używane). Te przyciski będziemy reprezentować za pomocą dziesięciu zmiennych $\text{CALLUP}[j]$ $\text{CALLDOWN}[j]$, $0 \leq j \leq 4$. Będziemy także korzystać ze zmiennych $\text{CALLCAR}[j]$, $0 \leq j \leq 4$, reprezentujących przyciski w kabinie windy służące do wyboru kondygnacji docelowej. Fakt naciśnięcia przycisku reprezentujemy, przypisując 1 do odpowiedniej zmiennej. Winda ustawia stan przycisku na 0 po realizacji żądania.

Opisaliśmy windę z punktu widzenia pasażera, ale sytuacja wygląda o wiele ciekawiej z punktu widzenia windy. Winda jest zawsze w jednym z trzech stanów: jazda w górę (**GOINGUP**), jazda w dół (**GOINGDOWN**) lub postój (**NEUTRAL**). (Bieżący stan jest sygnalizowany za pomocą podświetlanych strzałek wewnątrz kabiny). Jeżeli winda jest w stanie **NEUTRAL** i nie stoi na kondygnacji 2, to drzwi zostaną zamknięte i (jeśli do momentu zamknięcia się drzwi nie zostanie wydane żadne polecenie) winda przejdzie w stan **GOINGUP** lub **GOINGDOWN**, by dojechać do kondygnacji 2. (Jest to „kondygnacja postojowa”, ponieważ tam wsiada do windy najwięcej pasażerów). Na kondygnacji 2 w stanie **NEUTRAL** drzwi windy otworzą się, a sterownik będzie bezczynnie czekał na polecenie. Pierwsze otrzymane polecenie dotyczące innej kondygnacji sprawia, że winda wchodzi w odpowiedni ze stanów, **GOINGUP** lub **GOINGDOWN**, i pozostaje w nim, jeżeli nie ma innych oczekujących poleceń dotyczących tego samego kierunku jazdy. Następnie winda zmienia kierunek lub zmienia stan na **NEUTRAL** tuż przed otwarciem drzwi, zależnie od innych poleceń oraz zmiennych **CALL**. Winda potrzebuje trochę czasu na otwarcie i zamknięcie drzwi, na ruszenie i zatrzymanie się, a także na przemieszczenie się między kondygnacjami. Wszystkie te wielkości występują w poniższym algorytmie, który jest dużo dokładniejszy niż nieformalny opis słowny. Algorytm być może nie jest dokładnie tym samym zbiorem zasad, które rządzą zachowaniem windy w budynku Wydziału Matematyki, ale wydaje się, że są to najprostsze reguły wyjaśniające wszystkie zjawiska zaobserwowane podczas godzin eksperymentów przeprowadzonych przez autora.

Windę symulujemy za pomocą dwóch współprogramów, jednego dla pasażerów i jednego dla windy. Współprogramy opisują wszystkie zdarzenia, które mogą zajść, a także różne opóźnienia, które musimy uwzględnić w symulacji. Zmienna **TIME** reprezentuje bieżącą wartość zegara mierzącego czas symulacji. Wszystkie wartości dotyczące czasu podajemy w *dziesiątych częściach sekundy*. Poza tym używamy następujących zmiennych:

FLOOR, kondygnacja, na której znajduje się winda;

D1, zmienna cały czas równa zero, poza okresami, gdy pasażerowie wsiadają lub wysiadają z windy;

D2, zmienna, która przyjmuje wartość zero, jeżeli winda bez ruchu stoi na jednej kondygnacji 30 sekund lub dłużej;

D3, zmienna cały czas równa zero, poza okresami, gdy drzwi są otwarte i nikt nie wsiada ani nie wysiada z windy;

STATE, bieżący stan windy: (**GOINGUP**, **GOINGDOWN** lub **NEUTRAL**).

Początkowo **FLOOR** = 2, **D1** = **D2** = **D3** = 0, **STATE** = **NEUTRAL**.

Współprogram U (Pasażerowie). Każdy, kto wchodzi w obręb systemu, wykonuje opisane poniżej działania, zaczynając od kroku **U1**.

U1. [Wejście, przygotowanie następnika] Zakładamy, że poniższe wartości są w pewien sposób wyznaczone:

IN, kondygnacja, na której nowy pasażer wszedł w obręb systemu;

OUT, piętro, na którym pasażer chce wysiąść ($OUT \neq IN$);
GIVEUPTIME, czas oczekiwania na windę, po którym (niedoszły) pasażer zdecyduje się iść schodami;
INTERTIME, czas, po którym w obręb systemu wejdzie następny pasażer.

Po obliczeniu tych wartości symulator tak ustawia swój stan, że następny pasażer wejdzie w obręb systemu o czasie **TIME + INTERTIME**.

- U2.** [Naciśnięcie i oczekивание] (W tym kroku wzywamy windę; pojawiają się przypadki szczególne, jeżeli winda już jest na właściwym piętrze). Jeśli **FLOOR = IN** i jeśli następnym krokiem współprogramu windy jest E6 (tj. jeśli drzwi windy się zamkają), to przesuń natychmiast współprogram windy do kroku E3 i anuluj operację w E6. (To znaczy, że drzwi otworzą się ponownie, zanim winda ruszy). Jeśli **FLOOR = IN** i jeśli $D3 \neq 0$, to przyjmij $D3 \leftarrow 0$, przypisz niezerową wartość do **D1** i ponownie uruchom krok E4 współprogramu windy. (To znaczy, że drzwi windy są otwarte na tej kondycji, ale wszyscy pozostali pasażerowie już wsiedli lub wysiedli. W kroku E4 współprogramu windy pasażerowie mogą wsiąść do windy zgodnie ze zwykłymi zasadami dobrego wychowania. Stąd ponowne uruchomienie kroku E4 daje pasażerowi szansę na wejście do windy przed zamknięciem drzwi). We wszystkich innych przypadkach współprogram pasażera wykonuje **CALLUP[IN] ← 1** lub **CALLDOWN[IN] ← 1** w zależności od tego, czy **OUT > IN**, czy **OUT < IN**. Ponadto jeśli $D2 = 0$ lub winda jest w „spoczynkowym” położeniu E1, wykonywany jest opisany poniżej podprogram **DECISION**. (Podprogram **DECISION** po upływie pewnego czasu powoduje wyjście windy ze stanu **NEUTRAL**).
 - U3.** [Wstawianie do kolejki] Wstaw pasażera na koniec kolejki **QUEUE[IN]** będącej listą liniową reprezentującą pasażerów czekających na piętrze. Teraz pasażer cierpliwie czeka na przyjazd windy przez **GIVEUPTIME** jednostek czasu – dokładniej: czeka dopóki krok E4 współprogramu windy nie „przeniesie” go do U5 i anuluje zdarzenie U4.
 - U4.** [Rezygnacja] Jeżeli **FLOOR ≠ IN** lub **D1 = 0**, to „usuń” pasażera z listy **QUEUE[IN]** i z symulowanego systemu. (Pasażer doszedł do wniosku, że nie ma sensu czekać, a odrobina ruchu jeszcze nikomu nie zaszkodziła). Jeśli **FLOOR = IN** i **D1 ≠ 0**, to pasażer stoi i czeka (wiedząc, że nie będzie czekał długo).
 - U5.** [Wsiadanie] Użytkownik jest usuwany z listy **QUEUE[IN]** i umieszczany na liście stosowej **ELEVATOR**, reprezentującej pasażerów w kabinie windy. Ponadto **CALLCAR[OUT] ← 1**.
- Jeśli teraz **STATE = NEUTRAL**, to odpowiednio przyjmij **STATE ← GOINGUP** lub **GOINGDOWN**; następnie spraw, by krok E5 współprogramu windy został wykonany po 25 jednostkach czasu. (To jest specjalna cecha windy, dzięki temu drzwi mogą zamknąć się szybciej niż zwykle, jeżeli winda jest w stanie **NEUTRAL**, gdy pasażer naciska przycisk z numerem kondycji). W czasie 25-jednostkowego opóźnienia system może w kroku E4 upewnić się, że **D1**

ma odpowiednią wartość w momencie wykonania kroku E5, tj. zamykania drzwi).

Pasażer czeka teraz na przejście z kroku E4 do kroku U6, gdy winda dojedzie na odpowiednie piętro.

U6. [Wysiadanie] Usuwanie pasażera z listy ELEVATOR oraz z symulowanego systemu. ■

Współprogram E (Winda). Współprogram opisuje zachowanie windy; w kroku E4 jest ponadto przejmowane sterowanie wtedy, gdy pasażerowie wsiadają lub wysiadają.

- E1. [Czekanie na wezwanie]** (W tym momencie winda stoi na kondygamacji 2 z zamkniętymi drzwiami i czeka, co się zdarzy). Jeżeli ktoś naciśnie przycisk, podprogram DECISION spowoduje skok do kroku E3 lub E6. Póki co, czekaj.
- E2. [Czy zmiana stanu?]** Jeśli $STATE = GOINGUP$ i $CALLUP[j] = CALLDOWN[j] = CALLCAR[j] = 0$ dla wszystkich $j > FLOOR$, to $STATE \leftarrow NEUTRAL$ lub $STATE \leftarrow GOINGDOWN$ w zależności od tego, czy $CALLCAR[j] = 0$ dla wszystkich $j < FLOOR$, i przypisz zero do wszystkich zmiennych CALL dotyczących bieżącego piętra. Jeśli $STATE = GOINGDOWN$, to wykonaj analogiczne operacje dla odwrotnych kierunków.
- E3. [Otwieranie drzwi]** Przypisz niezerowe wartości do D1 i D2. Spraw, by krok E9 współprogramu windy zaczął się niezależnie wykonywać po 300 jednostkach czasu. (Zlecenie wykonania kroku E9 może zostać anulowane w kroku E6. Jeżeli jedno z poprzednich zleceń wykonania kroku E9 aktualnie „czeka”, to anulujemy jego wykonanie i zlecamy na nowo). Ponadto zleć niezależne wykonanie kroku E5 po 76 jednostkach czasu. Następnie czekaj 20 jednostek (symulujemy otwarcie drzwi) i przejdź do kroku E4.
- E4. [Wpuszczanie/wypuszczanie pasażerów]** Jeżeli dla któregokolwiek pasażera na liście ELEVATOR spełniony jest warunek $OUT = FLOOR$, to „przenies” tego pasażera, który wsiadł jako ostatni, do kroku U6 i czekaj 25 jednostek. Następnie powtórz krok E4. Jeżeli takich pasażerów nie ma, ale lista QUEUE[FLOOR] nie jest pusta, „przenies” osobę z początku tej kolejki do kroku U5 zamiast do U4, następnie oczekaj 25 jednostek i powtórz krok E4. Jeżeli QUEUE[FLOOR] jest pusta, to przyjmij $D1 \leftarrow 0$, przypisz niezerową wartość do D3 i czekaj na zdarzenia. (W kroku E5 nastąpi przejście do E6 lub w kroku U2 zostanie ponownie uruchomiony krok E4).
- E5. [Zamykanie drzwi]** Jeśli $D1 \neq 0$, to czekaj 40 jednostek i powtórz ten krok (drzwi się trochę „szamocą”, ale otwierają się ponownie, ponieważ ktoś wciąż jeszcze chce wsiąść lub wysiąść). W przeciwnym razie przyjmij $D3 \leftarrow 0$ i spraw, by po 20 jednostkach czasu współprogram windy zaczął się wykonywać w kroku E6. (To odpowiada zamknięciu drzwi, gdy ludzie już wsiedli i wysiedli, ale gdy na piętrze pojawiają się nowi pasażerowie podczas zamykania drzwi, drzwi otworzą się ponownie, jak powiedziano w kroku U2).
- E6. [Przygotowanie się do jazdy]** Przypisz zero do $CALLCAR[FLOOR]$; ponadto przypisz zero do $CALLUP[FLOOR]$, jeśli $STATE \neq GOINGDOWN$; przypisz także

zero do CALLDOWN[FLOOR], jeśli STATE \neq GOINGUP. (Uwaga: Jeśli STATE = GOINGUP, to winda nie kasuje CALLDOWN, ponieważ zakładamy, że ludzie, którzy mają jechać w dół, wciąż nie wsiedli; zobacz ćwiczenie 6). Uruchom podprogram DECISION.

Jeśli STATE = NEUTRAL nawet po wykonaniu podprogramu DECISION, to przejdź do E1. W przeciwnym razie jeśli D2 \neq 0, to anuluj zlecenie wykonania kroku E9. Jeśli STATE = GOINGUP, oczekaj 15 jednostek (winda nabiera prędkości) i przejdź do kroku E7; jeśli STATE = GOINGDOWN, oczekaj 15 jednostek i przejdź do E8.

- E7.** [Przejechanie o jedno piętro w góre] Przyjmij FLOOR \leftarrow FLOOR + 1 i oczekaj 51 jednostek czasu. Jeśli teraz CALLCAR[FLOOR] = 1 lub CALLUP[FLOOR] = 1, lub jeśli ((FLOOR = 2 lub CALLDOWN[FLOOR] = 1) i jednocześnie CALLUP[j] = CALLDOWN[j] = CALLCAR[j] = 0 dla wszystkich $j > \text{FLOOR}$), to oczekaj 14 jednostek (hamowanie) i przejdź do kroku E2. W przeciwnym razie powtórz ten krok.
- E8.** [Przejechanie o jedno piętro w dół] Ten krok jest analogiczny do E7, z tym że kierunki są odwrócone, a czasy wynoszą odpowiednio 61 zamiast 51 i 23 zamiast 14. (Winda wolniej jedzie w dół niż w górę).
- E9.** [Ustawianie znacznika bezczynności] Przyjmij D2 \leftarrow 0 i wykonaj podprogram DECISION. (Ta niezależna akcja inicjowana w kroku E3 prawie zawsze jest anulowana w kroku E6. Zobacz ćwiczenie 4). ■

Podprogram D (*Podprogram DECISION*). Podprogram jest wykonywany po upływie ustalonych czasów (zobacz powyższe współprogramy), gdy trzeba podjąć decyzję o kierunku jazdy windy.

- D1.** [Czy trzeba decydować?] Jeśli STATE \neq NEUTRAL, to zakończ wykonanie podprogramu.
- D2.** [Czy trzeba otworzyć drzwi?] Jeżeli winda jest w kroku E1 i jeżeli CALLUP[2], CALLCAR[2] i CALLDOWN[2] nie są równe zero, to spraw, by po 20 jednostkach czasu winda zaczęła wykonywać krok E3. Zakończ działanie podprogramu. (Jeżeli podprogram DECISION został wywołany przez E9, to współprogram windy może wykonywać krok E1).
- D3.** [Czy są jakieś wezwania?] Znajdź najmniejszą wartość $j \neq \text{FLOOR}$, dla której CALLUP[j], CALLCAR[j] lub CALLDOWN[j] ma wartość niezerową i przejdź do kroku D4. Jeżeli takie j nie istnieje, to przyjmij $j \leftarrow 2$, jeżeli podprogram DECISION jest wykonywany w wyniku wywołania w kroku E6; w przeciwnym razie zakończ wykonanie podprogramu.
- D4.** [Ustawianie STATE] Jeśli FLOOR $>$ j, to przyjmij STATE \leftarrow GOINGDOWN; jeśli FLOOR $<$ j, to przyjmij STATE \leftarrow GOINGUP.
- D5.** [Czy winda w położeniu spoczynkowym?] Jeżeli współprogram windy jest w kroku E1 i jeśli $j \neq 2$, to spraw, by współprogram windy wykonał krok E6 po 20 jednostkach czasu. Wyjdź z podprogramu. ■

Tabela 1
PRZYKŁADOWE ZACHOWANIE SYSTEMU WINDY

TIME	STATE	FLOOR	D1	D2	D3	Krok	Akcja	TIME	STATE	FLOOR	D1	D2	D3	Krok	Akcja
0000	N	2	0	0	0	U1	Pasażer 1 pojawia się na kond. 0, jedzie na 2.	1083	D	1	X	X	0	U6	Pasażer 4 wysiada, opuszcza system.
0035	D	2	0	0	0	E8	Winda jedzie w dół.	1108	D	1	X	X	0	U6	Pasażer 3 wysiada, opuszcza system.
0038	D	1	0	0	0	U1	Pasażer 2 pojawia się na kond. 4, jedzie na 1.	1133	D	1	X	X	0	U6	Pasażer 5 wysiada, opuszcza system.
0096	D	1	0	0	0	E8	Winda jedzie w dół.	1139	D	1	X	X	0	E5	Drzwi się „szamocą”.
0136	D	0	0	0	0	U1	Pasażer 3 pojawia się na kond. 2, jedzie na 1.	1158	D	1	X	X	0	U6	Pasażer 2 wysiada, opuszcza system.
0141	D	0	0	0	0	U1	Pasażer 4 pojawia się na kond. 2, jedzie na 1.	1179	D	1	X	X	0	E5	Drzwi się „szamocą”.
0152	D	0	0	0	0	U4	Pasażer 1 rezygnuje, opuszcza system.	1183	D	1	X	X	0	U5	Pasażer 7 wsiada.
0180	D	0	0	0	0	E2	Winda się zatrzymuje.	1208	D	1	X	X	0	U5	Pasażer 8 wsiada.
0180	N	0	0	X	0	E3	Drzwi zaczynają się otwierać.	1219	D	1	X	X	0	E5	Drzwi się „szamocą”.
0200	N	0	X	X	0	E4	Drzwi otwarte, nikogo nie ma.	1233	D	1	X	X	0	U5	Pasażer 9 wsiada.
0256	N	0	0	X	X	E5	Drzwi windy zaczynają się zamkać.	1259	D	1	0	X	X	E5	Drzwi windy zaczynają się zamkać.
0291	U	0	0	X	0	U1	Pasażer 5 pojawia się na kond. 3, jedzie na 1.	1294	D	1	0	X	0	E8	Winda jedzie w dół.
0291	U	0	0	X	0	E7	Winda jedzie do góry.	1378	D	0	0	X	0	E2	Winda się zatrzymuje.
0342	U	1	0	X	0	E7	Winda jedzie do góry.	1378	U	0	0	X	0	E3	Drzwi zaczynają się otwierać.
0364	U	2	0	X	0	U1	Pasażer 6 pojawia się na kond. 2, jedzie na 1.	1398	U	0	X	X	0	U6	Pasażer 8 wysiada, opuszcza system.
0393	U	2	0	X	0	E7	Winda jedzie do góry.	1423	U	0	X	X	0	U5	Pasażer 10 wsiada.
0444	U	3	0	X	0	E7	Winda jedzie do góry.	1454	U	0	0	X	X	E5	Drzwi windy zaczynają się zamkać.
0509	U	4	0	X	0	E2	Winda się zatrzymuje.	1489	U	0	0	X	0	E7	Winda jedzie do góry.
0509	N	4	0	X	0	E3	Drzwi zaczynają się otwierać.	1554	U	1	0	X	0	E2	Winda się zatrzymuje.
0529	N	4	X	X	0	U5	Pasażer 2 wsiada.	1554	U	1	0	X	0	E3	Drzwi zaczynają się otwierać.
0540	D	4	X	X	0	U4	Pasażer 6 rezygnuje, opuszcza system.	1630	U	1	0	X	X	E5	Drzwi windy zaczynają się zamkać.
0554	D	4	0	X	X	E5	Drzwi windy zaczynają się zamkać.	1665	U	1	0	X	0	E7	Winda jedzie do góry.
0589	D	4	0	X	0	E8	Winda jedzie w dół.	...							
0602	D	3	0	X	0	U1	Pasażer 7 pojawia się na kond. 1, jedzie na 2.	4257	N	2	0	X	0	E1	Winda w położeniu postojowym.
0673	D	3	0	X	0	E2	Winda się zatrzymuje.	4384	N	2	0	X	0	U1	Pasażer 17 pojawia się na kond. 2, jedzie na 3.
0673	D	3	0	X	0	E3	Drzwi zaczynają się otwierać.	4404	N	2	0	X	0	E3	Drzwi zaczynają się otwierać.
0693	D	3	X	X	0	U5	Pasażer 5 wsiada.	4424	N	2	X	X	0	U5	Pasażer 17 wsiada.
0749	D	3	0	X	X	E5	Drzwi windy zaczynają się zamkać.	4449	U	2	0	X	X	E5	Drzwi windy zaczynają się zamkać.
0784	D	3	0	X	0	E8	Winda jedzie w dół.	4484	U	2	0	X	0	E7	Winda jedzie do góry.
0827	D	2	0	X	0	U1	Pasażer 8 pojawia się na kond. 1, jedzie na 0.	4549	U	3	0	X	0	E2	Winda się zatrzymuje.
0868	D	2	0	X	0	E2	Winda się zatrzymuje.	4549	N	3	0	X	0	E3	Drzwi zaczynają się otwierać.
0868	D	2	0	X	0	E3	Drzwi zaczynają się otwierać.	4569	N	3	X	X	0	U6	Pasażer 17 wsiada, opuszcza system.
0876	D	2	X	X	0	U1	Pasażer 9 pojawia się na kond. 1, jedzie na 3.	4625	N	3	0	X	X	E5	Drzwi windy zaczynają się zamkać.
0888	D	2	X	X	0	U5	Pasażer 3 wsiada.	4660	D	3	0	X	0	E8	Winda jedzie w dół.
0913	D	2	X	X	0	U5	Pasażer 4 wsiada.	4744	D	2	0	X	0	E2	Winda się zatrzymuje.
0944	D	2	0	X	X	E5	Drzwi windy zaczynają się zamkać.	4744	N	2	0	X	0	E3	Drzwi zaczynają się otwierać.
0979	D	2	0	X	0	E8	Winda jedzie w dół.	4764	N	2	X	X	0	E4	Drzwi otwarte, nikogo nie ma.
1048	D	1	0	X	0	U1	Pasażer 10 pojawia się na kond. 0, jedzie na 4.	4820	N	2	0	X	0	E5	Drzwi windy zaczynają się zamkać.
1063	D	1	0	X	0	E2	Winda się zatrzymuje.	4840	N	2	0	X	0	E1	Winda w położeniu postojowym.
1063	D	1	0	X	0	E3	Drzwi zaczynają się otwierać.	...							

W porównaniu z innymi algorytmami, które są opisane w tej książce, system symulujący windę jest dość skomplikowany. Jednak problem „z życia wzięty” lepiej ilustruje zagadnienia symulacji niż najlepiej nawet spreparowany przykład podręcznikowy.

By lepiej zrozumieć ten system, przyjrzyjmy się tabeli 1, w której jest przedstawiony fragment historii pewnej symulacji. Najlepiej zacząć od czasu 4257: winda z zamkniętymi drzwiami stoi na kondygmacji 2. Pojawia się pasażer (czas 4384), powiedzmy Don. Dwie sekundy później drzwi się otwierają, a po następnych dwóch sekundach Don jest w kabinie. Naciska przycisk „3”, powodując jazdę do góry. Dociera na kondygmację 3, a winda powraca na kondygmację 2.

W pierwszych pozycjach tabeli 1 jest opisany o wiele bardziej skomplikowany scenariusz: pasażer wzywa windę na piętrze 0, ale traci cierpliwość i rezygnuje po 15.2 s. Winda zatrzymuje się na kondygmacji 0, ale tam nikogo nie ma. W następnej kolejności jedzie na piętro 4, bo jest tam kilka wezwań do jazdy w dół itd.

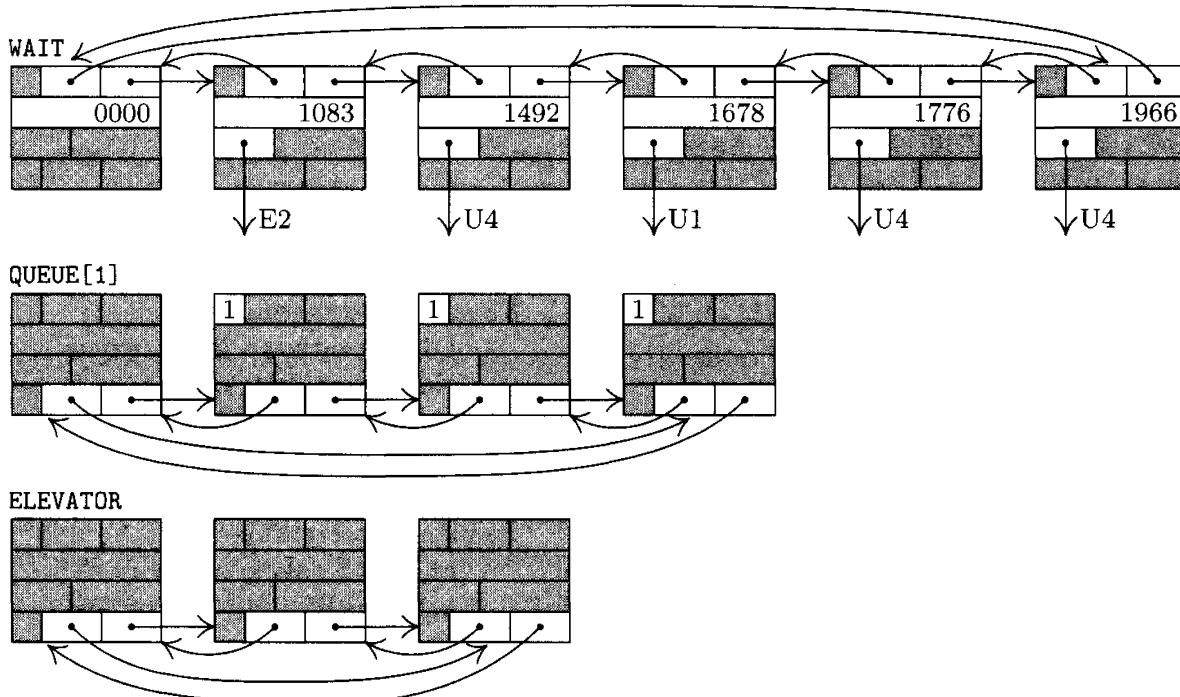
Zagadnienie zaprogramowania systemu na komputerze (MIX w naszym przypadku) zasługuje na dokładne zbadanie. W każdej chwili symulowanego czasu w systemie może być wielu symulowanych pasażerów (w różnych kolejkach, gotowych „zrezygnować” w różnych chwilach), może także zaistnieć konieczność w zasadzie jednoczesnego wykonania kroków E4, E5 i E9, jeżeli wiele osób próbuje wysiąść, gdy winda zamkna drzwi. Upływ symulowanego czasu oraz „jednoczesność” zdarzeń można zaprogramować przy użyciu obiektów reprezentowanych przez elementy zawierające pole **NEXTTIME** (oznaczające czas, kiedy zdarzy się następna akcja dotycząca danego obiektu) oraz pole **NEXTINST** (oznaczające adres w pamięci, od którego ma się zacząć wykonywanie rozkazów dotyczących akcji związanej z obiektem, analogicznie do aktywacji współprogramu). Każdy obiekt czekający na upływ czasu jest umieszczany na dwukierunkowej liście **WAIT**; ta struktura to „lista nadchodzących zdarzeń”, na której utrzymujemy porządek względem pól **NEXTTIME**, tak by akcje były przetwarzane w odpowiedniej kolejności symulowanych czasów ich zajścia. Program korzysta także z list dwukierunkowych **ELEVATOR** i **QUEUE**.

Każdy element listy reprezentujący akcję (pasażera lub windy) ma postać

+	IN	LLINK1	RLINK1		
+	NEXTTIME				
+	NEXTINST	0	0	39	
+	OUT	LLINK2	RLINK2		

(6)

LLINK1 i **RLINK1** są dowiązaniem listy **WAIT**; **LLINK2** i **RLINK2** są wykorzystywane jako dowiązania list **QUEUE** lub **ELEVATOR**. Pola **IN** i **OUT** są nieistotne, gdy element (6) reprezentuje pasażera, ale mają znaczenie w przypadku elementów reprezentujących akcje windy. Trzecie słowo elementu jest w istocie rozkazem „**JMP**” komputera **MIX**.



Rys. 12. Niektóre listy w systemie symulacji windy. (Atrapy list są umieszczone z lewej strony).

Na rysunku 12 są pokazane przykładowe listy WAIT, ELEVATOR i jedna z list QUEUE. Każdy element na liście QUEUE jest jednocześnie elementem listy WAIT, takim że NEXTINST = U4, ale nie pokazano tego na rysunku, ponieważ gąszcz dowiązań zaciemniłby schemat.

Zajmijmy się teraz samym programem. Jest nieco przydługi, chociaż (jak wszystkie długie programy) daje się w nim wyróżnić mniejsze proste części. Kilka pierwszych wierszy kodu definiuje początkową zawartość struktur. Warto zauważyć, że mamy tu atrapy: listy WAIT (wiersze 010–011), list QUEUE (wiersze 026–031) oraz listy ELEVATOR (wiersze 032–033). Każda z nich jest elementem postaci (6), ale bez nieistotnych słów. Atrapa elementu listy WAIT zawiera jedynie pierwsze dwa słowa elementu, a atrapy elementów list QUEUE i ELEVATOR wymagają obecności tylko ostatniego słowa elementu. Mamy także cztery elementy, które zawsze są w systemie obecne (wiersze 012–023): USER1 – element, który w kroku U1 służy do wprowadzania do systemu nowego pasażera; ELEV1 – element odpowiedzialny za główne akcje windy w krokach E1, E2, E3, E4, E6, E7 i E8; ELEV2 i ELEV3 – elementy, odpowiedzialne za akcje E5 i E9, zachodzące niezależnie (w sensie czasu symulacji) od pozostałych akcji windy. Każdy z tych elementów zawiera tylko trzy słowa, ponieważ nigdy nie wstawiamy ich na listy QUEUE lub ELEVATOR. Elementy reprezentujące pasażerów będą pojawiały się na stercie umiejscowionej za programem.

```

001 * SYMULATOR WINDY
002 IN      EQU 1:1
003 LLINK1 EQU 2:3
004 RLINK1 EQU 4:5

```

Definicja pól elementów.

```

005 NEXTINST EQU 0:2
006 OUT      EQU 1:1
007 LLINK2   EQU 2:3
008 RLINK2   EQU 4:5

009 * STRUKTURY O STAŁYM ROZMIARZE ORAZ GŁOWY LIST
010 WAIT      CON *+2(LLINK1),*+2(RLINK1) Głowa listy WAIT
011          CON 0 zawsze NEXTTIME = 0.
012 USER1    CON *-2(LLINK1),*-2(RLINK1) Element reprezentujący akcję
013          CON 0 U1; początkowo jest jedynym
014          JMP U1 elementem listy WAIT.
015 ELEV1    CON 0 Element reprezentujący
016          CON 0 akcje windy, poza
017          JMP E1 E5 i E9.
018 ELEV2    CON 0 Element reprezentujący
019          CON 0 niezależną akcję windy
020          JMP E5 w kroku E5.
021 ELEV3    CON 0 Element reprezentujący
022          CON 0 niezależną akcję windy
023          JMP E9 w kroku E9.
024 AVAIL    CON 0 Dowiązanie do dostępnych elementów.
025 TIME     CON 0 Bieżący czas symulacji.
026 QUEUE    EQU *-3
027          CON *-3(LLINK2),*-3(RLINK2) Głowa listy QUEUE[0].
028          CON *-3(LLINK2),*-3(RLINK2) Głowa listy QUEUE[1].
029          CON *-3(LLINK2),*-3(RLINK2) Początkowo wszystkie listy
030          CON *-3(LLINK2),*-3(RLINK2) są puste.
031          CON *-3(LLINK2),*-3(RLINK2) Głowa listy QUEUE[4].
032 ELEVATOR EQU *-3
033          CON *-3(LLINK2),*-3(RLINK2) Głowa listy ELEVATOR.
034          CON 0
035          CON 0 } „Wyrównywanie” dla
036          CON 0 struktury CALL
037          CON 0 (zobacz wiersze 183–186).
038 CALL     CON 0 CALLUP[0], CALLCAR[0], CALLDOWN[0]
039          CON 0 CALLUP[1], CALLCAR[1], CALLDOWN[1]
040          CON 0 CALLUP[2], CALLCAR[2], CALLDOWN[2]
041          CON 0 CALLUP[3], CALLCAR[3], CALLDOWN[3]
042          CON 0 CALLUP[4], CALLCAR[4], CALLDOWN[4]
043          CON 0 } „Wyrównywanie” dla
044          CON 0 struktury CALL
045          CON 0 (zobacz wiersze 178–181).
046          CON 0
047 D1      CON 0 Otwarte drzwi, aktywność.
048 D2      CON 0 Koniec postoju.
049 D3      CON 0 Otwarte drzwi, brak aktywności. ■

```

Następna część kodu programu zawiera podstawowe podprogramy oraz główny program sterujący symulatora. Podprogramy **INSERT** i **DELETE** wykonują typowe operacje na listach dwukierunkowych; wstawiają lub usuwają bieżący element

z list QUEUE lub ELEVATOR. (W programie „element bieżący” C jest zawsze reprezentowany za pomocą rI6). Mamy tu także podprogramy obsługujące listę WAIT: podprogram SORTIN dodaje bieżący element do listy, wsortowując go zgodnie z zawartością pola NEXTTIME. Podprogram IMMED wstawia bieżący element na początek listy WAIT. Podprogram HOLD wstawia bieżący element do listy WAIT z polem NEXTTIME równym bieżącemu czasowi symulacji plus wartość z rejestrów A. Podprogram DELETEW usuwa bieżący element z listy WAIT.

Podprogram CYCLE jest sercem symulatora: ustala, jaką akcję należy podjąć (tj. pobiera element z listy WAIT, która, jak wiemy, jest niepusta) i wykonuje skok w odpowiednie miejsce kodu. Ten podprogram ma dwa specjalne wejścia: CYCLE1 dodatkowo przed wykonaniem podprogramu ustawia NEXTINST w bieżącym elemencie; HOLDC robi to samo, tyle że wywołuje jeszcze podprogram HOLD. Stąd skutkiem wykonania rozkazu „JMP HOLDC” z wartością t w rejestrze A jest zawieszenie wykonania bieżącego kodu na t jednostek symulowanego czasu i powrót do lokacji następnej po rozkazie skoku.

050 * PODPROGRAMY I PROGRAM STEROWANIA		
051	INSERT STJ 9F	Wstaw NODE(C) na lewo od NODE(rI1):
052	LD2 3,1(LLINK2)	rI2 \leftarrow LLINK2(rI1).
053	ST2 3,6(LLINK2)	LLINK2(C) \leftarrow rI2.
054	ST6 3,1(LLINK2)	LLINK2(rI1) \leftarrow C.
055	ST6 3,2(RLINK2)	RLINK2(rI2) \leftarrow C.
056	ST1 3,6(RLINK2)	RLINK2(C) \leftarrow rI1.
057 9H	JMP *	Wyjście z podprogramu.
058	DELETE STJ 9F	Usuń NODE(C) z listy:
059	LD1 3,6(LLINK2)	P \leftarrow LLINK2(C).
060	LD2 3,6(RLINK2)	Q \leftarrow RLINK2(C).
061	ST1 3,2(LLINK2)	LLINK2(Q) \leftarrow P.
062	ST2 3,1(RLINK2)	RLINK2(P) \leftarrow Q.
063 9H	JMP *	Wyjście z podprogramu.
064	IMMED STJ 9F	Wstaw NODE(C) na początek listy WAIT:
065	LDA TIME	
066	STA 1,6	NEXTTIME(C) \leftarrow TIME.
067	ENT1 WAIT	P \leftarrow LOC(WAIT).
068	JMP 2F	Wstaw NODE(C) na prawo od NODE(P).
069	HOLD ADD TIME	rA \leftarrow TIME + rA.
070	SORTIN STJ 9F	Wsoruj NODE(C) w listę WAIT:
071	STA 1,6	NEXTTIME(C) \leftarrow rA.
072	ENT1 WAIT	P \leftarrow LOC(WAIT).
073	LD1 0,1(LLINK1)	P \leftarrow LLINK1(P).
074	CMPA 1,1	Porównuj pola NEXTTIME, od prawej do lewej.
075	JL *-2	Powtarzaj, dopóki NEXTTIME(C) \geq NEXTTIME(P).
076 2H	LD2 0,1(RLINK1)	Q \leftarrow RLINK1(P).
077	ST2 0,6(RLINK1)	RLINK1(C) \leftarrow Q.
078	ST1 0,6(LLINK1)	LLINK1(C) \leftarrow P.
079	ST6 0,1(RLINK1)	RLINK1(P) \leftarrow C.
080	ST6 0,2(LLINK1)	LLINK1(Q) \leftarrow C.
081 9H	JMP *	Wyjście z podprogramu.

082	DELETEW STJ 9F	Usuń NODE(C) z listy WAIT:
083	LD1 0,6(LLINK1)	(Kod jak w wierszach 058–063,
084	LD2 0,6(RLINK1)	z tym że zamiast LLINK2, RLINK2
085	ST1 0,2(LLINK1)	jest LLINK1, RLINK1).
086	ST2 0,1(RLINK1)	
087	9H JMP *	
088	CYCLE1 STJ 2,6(NEXTINST)	NEXTINST(C) ← rJ.
089	JMP CYCLE	
090	HOLDC STJ 2,6(NEXTINST)	NEXTINST(C) ← rJ.
091	JMP HOLD	Wstaw NODE(C) na listę WAIT, opóźnienie rA.
092	CYCLE LD6 WAIT(RLINK1)	Bieżący element C ← RLINK1(LOC(WAIT)).
093	LDA 1,6	
094	STA TIME	TIME ← NEXTTIME(C).
095	JMP DELETEW	Usuń NODE(C) z listy WAIT.
096	JMP 2,6	Skocz do NEXTINST(C). ■

Teraz zajmiemy się kodem współprogramu U. Na początku kroku U1 bieżącym elementem C jest USER1 (zobacz wiersze 012–014), a w wierszach 099–100 ponownie wstawiamy USER1 na listę WAIT, zatem zlecenie od następnego pasażera zostanie wygenerowane po INTERTIME jednostkach symulowanego czasu. W wierszach 101–114 zajmujemy się przygotowaniem elementu reprezentującego nowego pasażera, m.in. ustawiamy kondycje IN i OUT. Stos AVAIL jest listą jednokierunkową utworzoną za pomocą dowiązań RLINK1. Zauważmy, że w wierszach 101–108 wykonujemy operację „C ← AVAIL”, korzystając z techniki POOLMAX, zobacz 2.2.3–(7). Nie musimy sprawdzać, czy wystąpiło PRZEPEŁNIENIE, ponieważ całkowity rozmiar sterty (liczba pasażerów w systemie w jednej chwili) rzadko przekracza 10 elementów (40 słów). Zwracanie elementów na stertę realizujemy w wierszach 156–158.

W całym programie zawartość rejestrów I4 jest równa wartości zmiennej FLOOR, a rejestr I5 ma wartość dodatnią, ujemną lub zerową, w zależności od tego, czy STATE = GOINGUP, GOINGDOWN czy NEUTRAL. Zmienne CALLUP[j], CALLCAR[j] i CALLDOWN[j] zajmują odpowiednio wycinki (1:1), (3:3) i (5:5) lokacji CALL+j.

097	*	WSPÓŁPROGRAM U	<u>U1. Wejście, przygotowanie następnika.</u>
098	U1	JMP VALUES	Oblicz IN, OUT, GIVEUPTIME, INTERTIME.
099		LDA INTERTIME	INTERTIME jest obliczone przez VALUES.
100		JMP HOLD	Wstaw NODE(C) do WAIT, opóźnienie INTERTIME.
101		LD6 AVAIL	C ← AVAIL.
102		J6P 1F	Skocz, jeśli AVAIL ≠ Λ.
103		LD6 POOLMAX(0:2)	
104		INC6 4	C ← POOLMAX + 4.
105		ST6 POOLMAX(0:2)	POOLMAX ← C.
106		JMP *+3	Załóż, że przepełnienie nie nastąpi.
107	1H	LDA 0,6(RLINK1)	
108		STA AVAIL	AVAIL ← RLINK1(AVAIL).
109		LD1 INFLOOR	rI1 ← INFLOOR (obliczone przez VALUES).
110		ST1 0,6(IN)	IN(C) ← rI1.
111		LD2 OUTFLOOR	rI2 ← OUTFLOOR (obliczone przez VALUES).
112		ST2 3,6(OUT)	OUT(C) ← rI2.

113	ENTA 39	Wstaw stałą (kod rozkazu JMP)
114	STA 2,6	do trzeciego słowa elementu wg (6).
115 U2	ENTA 0,4	<u>U2. Naciśnięcie i czekanie.</u> Przyjmij rA \leftarrow FLOOR.
116	DECA 0,1	FLOOR – IN.
117	ST6 TEMP	Przechowaj wartość C.
118	JANZ 2F	Skocz, jeśli FLOOR \neq IN.
119	ENT6 ELEV1	C \leftarrow LOC(ELEV1).
120	LDA 2,6(NEXTINST)	Czy winda w E6?
121	DECA E6	
122	JANZ 3F	
123	ENTA E3	Jeśli nie, ustaw ją w E3.
124	STA 2,6(NEXTINST)	
125	JMP DELETEW	Usuń z listy WAIT
126	JMP 4F	i ponownie wstaw na początek.
127 3H	LDA D3	Skocz, jeśli D3 = 0.
128	JAZ 2F	W przeciwnym razie wpisz niezero do D1.
129	ST6 D1	D3 \leftarrow 0.
130	STZ D3	Wstaw ELEV1 na początek kolejki WAIT.
131 4H	JMP IMMED	(rI1 i rI2 zostały zmienione).
132	JMP U3	rI2 \leftarrow OUT – IN.
133 2H	DEC2 0,1	
134	ENTA 1	
135	J2P *+3	Skocz, jeśli jedziemy w góre.
136	STA CALL,1(5:5)	CALLDOWN[IN] \leftarrow 1.
137	JMP *+2	
138	STA CALL,1(1:1)	CALLUP[IN] \leftarrow 1.
139	LDA D2	
140	JAZ *+3	Jeśli D2 = 0, to wywołaj DECISION.
141	LDA ELEV1+2(NEXTINST)	
142	DECA E1	Jeśli winda w E1, to wywołaj
143	JAZ DECISION	podprogram DECISION.
144 U3	LD6 TEMP	<u>U3. Wstawianie do kolejki.</u>
145	LD1 0,6(IN)	rI1 \leftarrow LOC(QUEUE[IN]).
146	ENT1 QUEUE,1	Wstaw NODE(C) na prawo od QUEUE[IN].
147	JMP INSERT	
148 U4A	LDA GIVEUPTIME	Czekaj GIVEUPTIME jednostek.
149	JMP HOLDC	<u>U4. Rezygnacja.</u>
150 U4	LDA 0,6(IN)	IN(C) – FLOOR.
151	DECA 0,4	
152	JANZ *+3	FLOOR = IN(C).
153	LDA D1	Zobacz ćwiczenie 7.
154	JANZ U4A	<u>U6. Wysiadanie.</u> Usuń NODE(C)
155 U6	JMP DELETE	z QUEUE lub ELEVATOR.
156	LDA AVAIL	AVAIL \leftarrow C.
157	STA 0,6(RLINK1)	
158	ST6 AVAIL	Kontynuuj symulację.
159	JMP CYCLE	<u>U5. Wsiadanie.</u> Usuń NODE(C)
160 U5	JMP DELETE	z QUEUE.
161	ENT1 ELEVATOR	

162	JMP INSERT	Wstaw na prawo od ELEVATOR.
163	ENTA 1	
164	LD2 3,6(OUT)	
165	STA CALL,2(3:3)	CALLCAR[OUT(C)] ← 1.
166	J5NZ CYCLE	Skocz, jeśli STATE ≠ NEUTRAL.
167	DEC2 0,4	rI2 ← OUT(C) – FLOOR.
168	ENT5 0,2	Przypisz do STATE odpowiedni kierunek.
169	ENT6 ELEV2	C ← LOC(ELEV2).
170	JMP DELETEW	Usuń krok E5 z listy WAIT.
171	ENTA 25	
172	JMP E5A	Wykonaj krok E5 za 25 jednostek czasu. ■

Kod współprogramu E jest dosyć wiernym tłumaczeniem wcześniejszego nieformalnego opisu. Zapewne najciekawszym fragmentem jest przygotowanie niezależnych akcji windy w kroku E3 oraz przeszukiwanie list ELEVATOR i QUEUE w kroku E4.

173	*	WSPÓŁPROGRAM E	
174	E1A	JMP CYCLE1	NEXTINST ← E1, idź do CYCLE.
175	E1	EQU *	<u>E1. Czekanie na wezwanie.</u> (Nic się nie dzieje).
176	E2A	JMP HOLDC	
177	E2	J5N 1F	<u>E2. Czy zmiana stanu?</u>
178		LDA CALL+1,4	Stan GOINGUP.
179		ADD CALL+2,4	
180		ADD CALL+3,4	
181		ADD CALL+4,4	
182		JAP E3	Czy są wezwania z wyższych kondygnacji?
183		LDA CALL-1,4(3:3)	Jeśli nie, to czy pasażerowie w kabinie
184		ADD CALL-2,4(3:3)	chcieli jechać na niższe kondygnacje?
185		ADD CALL-3,4(3:3)	
186		ADD CALL-4,4(3:3)	
187		JMP 2F	
188	1H	LDA CALL-1,4	Stan GOINGDOWN.
189		ADD CALL-2,4	Akcje jak w wierszach 178–186.
:			
196		ADD CALL+4,4(3:3)	
197	2H	ENN5 0,5,	Zmień kierunek w STATE.
198		STZ CALL,4	Przypisz zero do CALL.
199		JANZ E3	Skocz, jeśli wezwanie z przeciwnego kierunku;
200		ENT5 0	w przeciwnym razie STATE ← NEUTRAL.
201	E3	ENT6 ELEV3	<u>E3. Otwieranie drzwi.</u>
202		LDA 0,6	Jeżeli krok E9 jest zlecony, to usuń
203		JANZ DELETEW	zlecenie z listy WAIT.
204		ENTA 300	
205		JMP HOLD	Zleć wykonanie E9 po 300 jednostkach.
206		ENT6 ELEV2	
207		ENTA 76	
208		JMP HOLD	Zleć wykonanie E5 po 76 jednostkach.
209		ST6 D2	Przypisz niezero do D2.

210	ST6 D1	Przypisz niezero do D1.
211	ENTA 20	
212	E4A ENT6 ELEV1	
213	JMP HOLD C	
214	E4 ENTA 0,4	<u>E4. Wpuszczanie/wypuszczanie pasażerów.</u>
215	SLA 4	Wpisz FLOOR do pola OUT rejestru A.
216	ENT6 ELEVATOR	C ← LOC(ELEVATOR).
217	1H LD6 3,6(LLINK2)	C ← LLINK2(C).
218	CMP6=ELEVATOR=	Przeszukuj listę ELEVATOR, od prawej do lewej.
219	JE 1F	Jeśli C = LOC(ELEVATOR), to koniec przeszukiwania.
220	CMPA 3,6(OUT)	Porównaj OUT(C) z FLOOR.
221	JNE 1B	Jeśli różne, kontynuuj przeszukiwanie;
222	ENTA U6	w przeciwnym razie przygotuj się
223	JMP 2F	do „wysłania” pasażera do U6.
224	1H LD6 QUEUE+3,4(RLINK2)	C ← RLINK2(LOC(QUEUE[FLOOR])).
225	CMP6 3,6(RLINK2)	Czy C = RLINK2(C) ?
226	JE 1F	Jeśli tak, to kolejka jest pusta.
227	JMP DELETEW	Jeśli nie, anuluj przejście do U4 tego pasażera.
228	ENTA U5	Przygotuj zastąpienie U4 przez U5.
229	2H STA 2,6(NEXTINST)	Ustaw NEXTINST(C).
230	JMP IMMED	Umieść pasażera na początku listy WAIT.
231	ENTA 25	
232	JMP E4A	Czekaj 25 jednostek i powtóż E4.
233	1H STZ D1	D1 ← 0.
234	ST6 D3	Przypisz niezero do D3.
235	JMP CYCLE	Powróć, by symulować inne zdarzenia.
236	E5A JMP HOLD C	
237	E5 LDA D1	<u>E5. Zamykanie drzwi.</u>
238	JAZ *+3	Czy D1 = 0?
239	ENTA 40	Jeśli nie, pasażerowie wciąż w(y)siadają.
240	JMP E5A	Czekaj 40 jednostek, powtóż E5.
241	STZ D3	Jeśli D1 = 0, to D3 ← 0.
242	ENT6 ELEV1	
243	ENTA 20	
244	JMP HOLD C	Czekaj 20 jednostek, następnie idź do E6.
245	E6 J5N *+2	<u>E6. Przygotowanie do jazdy.</u>
246	STZ CALL,4(1:3)	Jeśli STATE ≠ GOINGDOWN, kasuj CALLUP
247	J5P *+2	i CALLCAR na tym piętrze.
248	STZ CALL,4(3:5)	Jeśli ≠ GOINGUP, to kasuj CALLCAR i CALLDOWN.
249	J5Z DECISION	Wywołaj podprogram DECISION.
250	E6B J5Z E1A	Jeśli STATE = NEUTRAL, idź do E1 i czekaj.
251	LDA D2	
252	JAZ *+4	
253	ENT6 ELEV3	W przeciwnym razie, jeśli D2 ≠ 0,
254	JMP DELETEW	to anuluj zlecenie E9
255	STZ ELEV3	(zobacz wiersz 202).
256	ENT6 ELEV1	
257	ENTA 15	Czekaj 15 jednostek czasu.
258	J5N E8A	Jeśli STATE = GOINGDOWN, to idź do E8.

```

259 E7A JMP HOLDC
260 E7 INC4 1
261 ENTA 51
262 JMP HOLD C
263 LDA CALL,4(1:3)
264 JAP 1F
265 ENT1-2,4
266 J1Z 2F
267 LDA CALL,4(5:5)
268 JAZ E7
269 2H LDA CALL+1,4
270 ADD CALL+2,4
271 ADD CALL+3,4
272 ADD CALL+4,4
273 JANZ E7
274 1H ENTA 14
275 JMP E2A
276 E8A JMP HOLDC
:
:
```

E7. Przejechanie o jedno piętro w góre

Czekaj 51 jednostek.
Czy CALLCAR[FLOOR] lub CALLUP[FLOOR] ≠ 0?

Jeśli nie,
to czy FLOOR = 2?
Jeśli nie, to czy CALLDOWN[FLOOR] ≠ 0?
Jeśli nie, powtórz krok E7.

Czy są wezwania z wyższych pięter?
Czy już zatrzymujemy windę?
Czekaj 14 jednostek i przejdź do E2.

```

292 JMP E2A
293 E9 STZ 0,6
294 STZ D2
295 JMP DECISION
296 JMP CYCLE
:
```

E9. Ustawianie znacznika bezczynności. (Zobacz
D2 ← 0. wiersz 202).
Wywołaj podprogram DECISION.
Wróć do symulacji innych zdarzeń. ■

Nie będziemy zajmować się podprogramem DECISION (zobacz ćwiczenie 9) ani podprogramem VALUES, określającym zapotrzebowanie na jeżdżenie windą. Na samym końcu programu należy jeszcze dołożyć kod

```

BEGIN ENT4 2      Zaczynamy na kondycji FLOOR = 2
        ENT5 0      i w stanie STATE = NEUTRAL.
        JMP CYCLE   Rozpocznij symulację.
POOLMAX NOP POOL
POOL    END BEGIN Stertę umieść za pamięcią tymczasową
        i stałymi pamięciowymi. ■
:
```

Powyższy program, symulując windę, „odwala” kawał niezłą roboty, ale pożytek z niego żaden, bo nie wyprowadza wyników! W istocie autor dodał podprogram PRINT, który – wywoływany w najistotniejszych krokach programu – wyprowadzał informacje, na podstawie których sporządzono tabelę 1. Szczególnie tego rozwiązania umyślnie w kodzie pominięto, gdyż są bardzo proste, a zaciemniają ją program.

Do konstrukcji systemów symulacji dyskretnej zazwyczaj poleca się wykorzystywanie specjalizowanych języków programowania oraz kompilatorów tłumaczących je na kod maszynowy. My w tym punkcie korzystaliśmy z asemblera, ponieważ nacisk kładziemy na podstawowe sposoby manipulowania listami wskaźnikowymi, a także chcieliśmy pokazać, w jaki sposób symulacja dyskretna

może być realizowana na komputerze, który „myśli jednotorowo”. Zaprezentowana metoda, polegająca na wykorzystaniu listy WAIT (listy nadchodzących zdarzeń) do sterowania wykonaniem kolejnych współprogramów, nazywana jest *przetwarzaniem quasi-równoległym*.

Dokładna analiza czasu wykonania tak długiego programu nie jest zadaniem łatwym z uwagi na złożone interakcje. Duże programy najczęściej jednak większość czasu spędzają na wykonaniu względnie krótkich fragmentów kodu odpowiedzialnych za proste zadania. Z tego powodu możemy zazwyczaj polegać na wskazaniach systemu tworzenia szkicu programu (*profiler*), będącego specjalną odmianą programu śledzącego. Taki system śledzi program główny i zapisuje, jak często wykonywane są poszczególne rozkazy. W ten sposób możemy namierzyć „wąskie gardła” – miejsca, na które należy zwrócić szczególną uwagę. [Zobacz ćwiczenie 1.4.3.2–7, a także *Software Practice & Experience* 1 (1971), 105–133, gdzie zamieszczono przykładowe analizy losowo wybranych programów w FORTRAN-ie znalezionych w koszach na śmieci Stanford Computer Center]. Autor przeprowadził eksperyment z symulatorem windy, uruchamiając go na 10000 jednostek symulowanego czasu; system miał obsłużyć 26 pasażerów. Najczęściej (1432 razy) wykonywane były rozkazy w pętli SORTIN, wiersze 073–075. Podprogram SORTIN był wywoływanym 437 razy. Pętla główna CYCLE wykonała się 407 razy. Moglibyśmy trochę zyskać, nie wywołując podprogramu DELETEW w wierszu 095: cztery wiersze tego podprogramu można bezpośrednio wpisać w kod (oszczędzając tym samym 4u na każdym obrocie pętli CYCLE). Analizator dynamiczny ujawnił także, że podprogram DECISION był wywoływany tylko 32 razy, a pętla w kroku E4 (wiersze 216–218) była wykonywana tylko 142 razy.

Autor ma nadzieję, że Czytelnicy, studując przykład, dowiedzą się tyle o symulacji dyskretnej, ile on, pisząc ten rozdział, dowiedział się o windach.

ĆWICZENIA

1. [21] Podaj specyfikacje operacji wstawiania i usuwania elementu z lewego końca listy dwukierunkowej w reprezentacji (1). (Razem z analogicznymi operacjami wstawiania na prawym końcu, które uzyskamy przez symetrię, będziemy dysponować wszystkimi operacjami na nieograniczonej kolejce dwustronnej).
- ▶ 2. [22] Wyjaśnij, dlaczego lista jednokierunkowa nie umożliwia wydajnej realizacji wszystkich operacji na nieograniczonej kolejce dwustronnej. Usuwanie elementów z listy można realizować efektywnie tylko na jednym końcu listy jednokierunkowej.
- ▶ 3. [22] System windy opisany w tekście korzysta z trzech zmiennych CALLUP, CALLCAR i CALLDOWN dla każdej kondygnacji, reprezentujących przyciski naciśnięte przez użytkowników systemu. Można dojść do wniosku, że w zasadzie potrzeba tylko jednej lub dwóch binarnych zmiennych dla każdej kondygnacji. Wyjaśnij, w jaki sposób eksperymentator powinien naciskać przyciski systemu windy, by udowodnić, że dla każdej kondygnacji (poza najniższą i najwyższą) w systemie są trzy niezależne zmienne binarne.
4. [24] Zlecenie wykonania kroku E9 we współprogramie windy jest zazwyczaj anulowane w kroku E6. A nawet wtedy, gdy nie zostało anulowane, bardzo niewiele wnosi. Wyjaśnij w jakich okolicznościach winda zachowywałaby się inaczej, gdyby krok E9

w ogóle wyrzucić z systemu. Czy na przykład w pewnych sytuacjach zmieniłaby się kolejność odwiedzania pięter?

5. [20] W tabeli 1 pasażer 10 przyjechał na kondygnację 0 w chwili 1048. Pokaż, że jeśli pasażer 10 przyjechałby na kondygnację 2 zamiast na 0, to winda pojedzie w górę, a nie w dół, po zabraniu pasażerów na kondygnacji 1, chociaż pasażer 8 chce jechać w dół na kondygnację 0.

6. [23] W przedziale czasu 1183–1233 w tabeli 1 pasażerowie 7, 8 i 9 wsiadają do windy na kondygnacji 1, po czym winda zjeżdża na kondygnację 0 i wysiada tylko pasażer 8. Następnie winda zatrzymuje się ponownie na kondygnacji 1, zapewne by zabrać pasażerów 7 i 9, którzy już są w windzie; w istocie na kondygnacji 1 nikt nie czeka. (Ta sytuacja wcale się nie zdarza w Caltech tak rzadko; jeśli pasażer wsiądzie do windy jadącej w złym kierunku, musi odczekać dodatkowe zatrzymanie windy na piętrze, na którym wsiadł). W wielu systemach windy pasażerowie 7 i 9 nie wsiedliby do kabiny w chwili 1183, ponieważ sygnalizacja przy drzwiach windy poinformowałaby ich, że kabina jedzie w dół. Pasażerowie ci poczekaliby, aż winda zatrzyma się ponownie, jadąc w górę. W opisany systemie nie ma takiej sygnalizacji i nie znajdująąc się w kabinie, nie sposób stwierdzić, czy windy jedzie w dół, czy w górę. Tabela 1 odzwierciedla rzeczywisty stan rzeczy.

Jakie zmiany należały wprowadzić do współprogramów U i E, jeżeli mielibyśmy symulować tę samą windę, ale z dodatkową sygnalizacją kierunku ruchu kabiny, w związku z czym pasażerowie nie wsiadaliby do windy jadącej w złym kierunku?

7. [25] Chociaż błędy w programach są zazwyczaj wstydnym tematem dla programistów, powinniśmy się uczyć na błędach. Z tego powodu należy je zapisywać i opowiadać o nich innym. Następujący błąd (między innymi) pojawił się w pierwszej wersji programu zaprezentowanego w tym rozdziale: w wierszu 154 zamiast „JANZ U4A” było „JANZ CYCLE”. Autor myślał, że jeżeli winda przyjechała na piętro, na którym czeka pasażer, to nie ma potrzeby „rezygnowania” w kroku U4, zatem możemy po prostu skoczyć do CYCLE i kontynuować symulację. Gdzie tkwi błąd?

8. [21] Napisz kod kroku E8, wiersze 277–292, którego brak w zaprezentowanym programie.

9. [23] Napisz kod podprogramu DECISION, którego brak w zaprezentowanym programie.

10. [40] Zapewne warto zaznaczyć, że autor latami korzystał z opisanej windy i wydawało mu się, że ją dobrze zna. Jednak jego pewność legła w gruzach, gdy zebrał się do pisania tego rozdziału i zdał sobie sprawę, że o wielu szczegółach dotyczących wyboru kierunku jazdy windy nie miał pojęcia. Autor sześciokrotnie powracał do eksperymentów z windą, za każdym razem wierząc, że wreszcie zrozumiał jej *modus operandi*. (Doszło do tego, że dziś autor boi się jeździć tą windą w obawie, że odkryje jakiś szczegół sprzeczny z opracowanym algorytmem). Zazwyczaj nie zdajemy sobie sprawy, jak mało wiemy o jakimś zjawisku, zanim nie zechcemy symulować go na komputerze.

Spróbuj wyspecyfikować akcje znanej Ci windy. Zweryfikuj algorytm eksperymentalnie (analizowanie sterownika jest nie fair!). Następnie zaprojektuj symulator dyskretny systemu i uruchom go na komputerze.

► 11. [21] (*Synchroniczne uaktualnianie pamięci*) Poniższy problem często pojawia się w symulatorach *synchronicznych*: system ma n zmiennych $V[1], \dots, V[n]$ i w każdym kroku symulacji nowe wartości niektórych z nich są wyznaczane na podstawie starych

wartości. Zakłada się, że te obliczenia są przeprowadzane „jednocześnie” w tym sensie, że najpierw wyznacza się wszystkie nowe wartości, a dopiero potem je przypisuje. Następujące rozkazy

$$V[1] \leftarrow V[2] \quad \text{ i } \quad V[2] \leftarrow V[1]$$

wykonane w tym samym (symulowanym) czasie powinny zamienić miejscami wartości zmiennych $V[1]$ i $V[2]$. To jest istotnie różne od skutku sekwencyjnego wykonania takich rozkazów.

Pożądany rezultat można oczywiście osiągnąć, przechowując dodatkową tablicę $NEWV[1], \dots, NEWV[n]$. Przed każdym kolejnym krokiem symulacji możemy przypisywać $NEWV[k] \leftarrow V[k]$ dla $1 \leq k \leq n$, następnie umieszczać wszystkie nowe wartości $V[k]$ w $NEWV[k]$, by na koniec po wykonaniu kroku przypisać znowu $V[k] \leftarrow NEWV[k]$, $1 \leq k \leq n$. To „siłowe” rozwiązywanie nie jest do końca satysfakcjonujące z następujących przyczyn: (1) Często n jest bardzo duże, a liczba zmiennych modyfikowanych w jednym kroku mała. (2) Zmienne zazwyczaj nie są przechowywane w eleganckiej tablicy $V[1], \dots, V[n]$, lecz rozrzucone po pamięci w sposób raczej przypadkowy. (3) Ta metoda nie umożliwia wykrycia sytuacji (oznaczającej zazwyczaj błąd w modelu), gdy jedna zmienna otrzymuje w tym samym kroku dwie wartości.

Zakładając, że liczba zmiennych modyfikowanych w jednym kroku jest mała, zaprojektuj wydajny algorytm symulacji, korzystający z dwóch pomocniczych struktur $NEWV[k]$ i $LINK[k]$, $1 \leq k \leq n$. Jeśli to możliwe, Twój algorytm powinien zatrzymywać się z komunikatem o błędzie, jeżeli tej samej zmiennej w jednym kroku nadano dwie wartości.

- 12. [22] Dlaczego użycie list dwukierunkowych zamiast jednokierunkowych w zaprezentowanym programie symulatora jest dobrym pomysłem?

2.2.6. Tablice i listy ortogonalne

Jednym z najprostszych uogólnień list liniowych jest tablica dwu- lub więcej wymiarowa. Przyjrzyjmy się macierzy $m \times n$

$$\begin{pmatrix} A[1,1] & A[1,2] & \dots & A[1,n] \\ A[2,1] & A[2,2] & \dots & A[2,n] \\ \vdots & \vdots & & \vdots \\ A[m,1] & A[m,2] & \dots & A[m,n] \end{pmatrix}. \quad (1)$$

W tej dwuwymiarowej tablicy każdy element $A[j,k]$ należy do dwóch list liniowych: listy „wiersz j ” $A[j,1], A[j,2], \dots, A[j,n]$ oraz listy „kolumna k ” $A[1,k], A[2,k], \dots, A[m,k]$.

Te ortogonalne listy wierszowe i kolumnowe składają się na dwuwymiarową strukturę macierzy. Analogicznie możemy spojrzeć na tablice o większej liczbie wymiarów.

Struktury tablicowe. Jeśli tablica jest przechowywana w kolejnych lokacjach pamięci, to pamięć zazwyczaj jest zorganizowana w ten sposób, że

$$LOC(A[J,K]) = a_0 + a_1 J + a_2 K, \quad (2)$$

gdzie a_0 , a_1 i a_2 są stałymi. Rozważmy bardziej ogólny przypadek: przypuśćmy, że mamy tablicę czterowymiarową o jednosłowowych elementach $Q[I,J,K,L]$ dla

$0 \leq I \leq 2, 0 \leq J \leq 4, 0 \leq K \leq 10, 0 \leq L \leq 2$. Chcielibyśmy tak zorganizować pamięć, by

$$\text{LOC}(Q[I, J, K, L]) = a_0 + a_1 I + a_2 J + a_3 K + a_4 L. \quad (3)$$

Oznacza to, że zmiana któregoś z indeksów I, J, K lub L ma powodować łatwą do obliczenia zmianę lokacji elementu $Q[I, J, K, L]$. Najbardziej naturalną (i najczęściej stosowaną) metodą organizacji pamięci jest ustawienie elementów zgodnie z porządkiem leksykograficznym ich indeksów (ćwiczenie 1.2.1–15(d)), czasami nazywanym „porządkiem wierszowym”:

$$\begin{aligned} Q[0, 0, 0, 0], Q[0, 0, 0, 1], Q[0, 0, 0, 2], Q[0, 0, 1, 0], Q[0, 0, 1, 1], \dots, \\ Q[0, 0, 10, 2], Q[0, 1, 0, 0], \dots, Q[0, 4, 10, 2], Q[1, 0, 0, 0], \dots, \\ Q[2, 4, 10, 2]. \end{aligned}$$

Latwo spostrzec, że ten porządek spełnia wymagania (3); mamy więc:

$$\text{LOC}(Q[I, J, K, L]) = \text{LOC}(Q[0, 0, 0, 0]) + 165I + 33J + 3K + L. \quad (4)$$

W ogólności tablicę k -wymiarową o elementach $A[I_1, I_2, \dots, I_k]$ dla

$$0 \leq I_1 \leq d_1, \quad 0 \leq I_2 \leq d_2, \quad \dots, \quad 0 \leq I_k \leq d_k,$$

po c słów każdy, możemy przechowywać w pamięci jako

$$\begin{aligned} \text{LOC}(A[I_1, I_2, \dots, I_k]) &= \text{LOC}(A[0, 0, \dots, 0]) + c(d_2 + 1) \dots (d_k + 1)I_1 + \dots + c(d_k + 1)I_{k-1} + cI_k \\ &= \text{LOC}(A[0, 0, \dots, 0]) + \sum_{1 \leq r \leq k} a_r I_r, \end{aligned} \quad (5)$$

gdzie

$$a_r = c \prod_{r < s \leq k} (d_s + 1). \quad (6)$$

By przekonać się, że ten wzór rzeczywiście działa, zauważmy, że a_r jest wielkością pamięci potrzebnej do przechowywania podtablicy $A[I_1, \dots, I_r, J_{r+1}, \dots, J_k]$, gdzie I_1, \dots, I_r są stałe, a J_{r+1}, \dots, J_k zmieniają się w przedziałach $0 \leq J_{r+1} \leq d_{r+1}, \dots, 0 \leq J_k \leq d_k$. Zatem z istoty porządku leksykograficznego wynika, że przy zmianie I_r o 1 adres elementu $A[I_1, \dots, I_k]$ należy zmienić dokładnie o a_r .

Wzory (5) i (6) odpowiadają wartości liczb $I_1 I_2 \dots I_k$ w systemie pozycyjnym o mieszanych podstawach. Na przykład jeśli mamy tablicę CZAS[T, D, G, M, S], $0 \leq T < 4, 0 \leq D < 7, 0 \leq G < 24, 0 \leq M < 60, 0 \leq S < 60$, to lokacja elementu CZAS[T, D, G, M, S] będzie lokacją CZAS[0, 0, 0, 0, 0] plus wielkość „ T tygodni + D dni + G godzin + M minut + S sekund” wyrażona w sekundach. Oczywiście trzeba nie lada problemu, by rzeczywiście potrzebna była tablica o 2419200 elementach.

Standardowa metoda przechowywania tablic sprawdza się, gdy tablice mają w pełni prostokątny kształt, czyli gdy w tablicy występują wszystkie elementy $A[I_1, I_2, \dots, I_k]$ dla indeksów z niezależnych zakresów $l_1 \leq I_1 \leq u_1, l_2 \leq I_2 \leq u_2, \dots, l_k \leq I_k \leq u_k$. Ćwiczenie 2 pokazuje, w jaki sposób przystosować

(5) i (6) do przypadku, gdy dolne wartości indeksów (l_1, l_2, \dots, l_k) są różne od $(0, 0, \dots, 0)$.

Zdarza się jednak wiele sytuacji, w których tablica nie jest w pełni prostokątna. Często spotykamy macierze trójkątne – interesuje nas przechowywanie wartości $A[j, k]$ jedynie na przykład dla $0 \leq k \leq j \leq n$:

$$\begin{pmatrix} A[0,0] & & & \\ A[1,0] & A[1,1] & & \\ \vdots & \vdots & \ddots & \\ A[n,0] & A[n,1] & \dots & A[n,n] \end{pmatrix}. \quad (7)$$

Z konkretnego problemu może na przykład wynikać, że wszystkie inne elementy mają wartość zero, albo że $A[j, k] = A[k, j]$, więc wystarczy przechowywać tylko połowę macierzy. Jeśli chcemy przechowywać dolną macierz trójkątną (7) w $\frac{1}{2}(n+1)(n+2)$ kolejnych komórkach pamięci, to musimy zrezygnować z liniowego ustawienia lokacji, jak w przypadku (2). Możemy w zamian posłużyć się schematem postaci

$$\text{LOC}(A[J, K]) = a_0 + f_1(J) + f_2(K) \quad (8)$$

gdzie f_1 i f_2 są funkcjami jednej zmiennej. (Jeśli chcemy, możemy włączyć stałą a_0 do f_1 lub f_2). Adresowanie (8) umożliwia szybkie dostanie się do dowolnego elementu $A[j, k]$, jeżeli przechowujemy dwie (raczej krótkie) dodatkowe tablice wartości funkcji f_1 i f_2 . Dzięki temu nie musimy tych wartości wielokrotnie obliczać.

Okazuje się, że porządek leksykograficzny indeksów tablicy (7) spełnia warunek (8), a dla elementów o rozmiarze jednego słowa mamy prosty wzór

$$\text{LOC}(A[J, K]) = \text{LOC}(A[0,0]) + \frac{J(J+1)}{2} + K. \quad (9)$$

W istocie istnieją jednak o wiele lepsze sposoby przechowywania tablic trójkątnych, jeżeli mamy tyle szczęścia, by w naszym programie występowały dwie takie struktury jednakowego rozmiaru. Przypuśćmy, że w pamięci chcemy przechowywać $A[j, k]$ i $B[j, k]$ dla $0 \leq k \leq j \leq n$. W takim przypadku możemy umieścić dwie macierze trójkątne w jednej macierzy $C[j, k]$ dla $0 \leq j \leq n$, $0 \leq k \leq n+1$, trzymając się konwencji

$$A[j, k] = C[j, k], \quad B[j, k] = C[k, j+1]. \quad (10)$$

Stąd

$$\begin{pmatrix} C[0,0] & C[0,1] & C[0,2] & \dots & C[0,n+1] \\ C[1,0] & C[1,1] & C[1,2] & \dots & C[1,n+1] \\ \vdots & & \vdots & & \\ C[n,0] & C[n,1] & C[n,2] & \dots & C[n,n+1] \end{pmatrix} \equiv \begin{pmatrix} A[0,0] & B[0,0] & B[1,0] & \dots & B[n,0] \\ A[1,0] & A[1,1] & B[1,1] & \dots & B[n,1] \\ \vdots & & & & \vdots \\ A[n,0] & A[n,1] & A[n,2] & \dots & B[n,n] \end{pmatrix}.$$

Dwie macierze trójkątne są spakowane w jedną strukturę rozmiaru $(n+1)(n+2)$ lokacji, ponadto możemy posługiwać się adresowaniem liniowym postaci (2).

Uogólnienie macierzy trójkątnej na więcej wymiarów nazywane jest *tablicą tetrahedryczną*. Ten zajmujący temat jest przedmiotem ćwiczeń 6–8.

Przykład typowych metod programistycznych związanych z wykorzystaniem tablic przechowywanych sekwencyjnie znajdzie Czytelnik w ćwiczeniu 1.3.2–10 i dwóch odpowiedziach do niego. W tych programach należy zwrócić uwagę na metody wydajnego przeglądania tablicy wierszami i kolumnami, a także na wykorzystanie stosów reprezentowanych w tablicy.

Struktury z dowiązaniami. Dowiązania są również stosowane przy realizacji tablic wielowymiarowych. W ogólności elementy mogą zawierać k pól z dowiązaniami, po jednym dla każdej listy, do której należy element. Metoda dowiązań zasadniczo sprawdza się w przypadkach, gdy tablice nie są w pełni prostokątne.

	MALE	FEMALE	A21	A22	A23	A24	A28	BLUE	BROWN	GREEN	HAZEL	BLOND	RED	DARK
PERSON [6]			•	•				•					•	
PERSON [5]	•			•				•					•	
PERSON [4]		•		•					•			•		
PERSON [3]	•				•				•			•		
PERSON [2]			•		•			•		•		•		
PERSON [1]	•					•					•			

Kobieta, lat 21, oczy brązowe, włosy ciemne
Mężczyzna, lat 24, oczy brązowe, włosy ciemne
Kobieta, lat 22, oczy zielone, włosy blond
Mężczyzna, lat 28, oczy piwne, włosy blond
Kobieta, lat 22, oczy niebieskie, włosy rude
Kobieta, lat 21, oczy niebieskie, włosy blond

Rys. 13. Każdy element jest na czterech różnych listach.

Jako przykład rozważmy listę, której elementy reprezentują osoby. Każdy element ma pola określające płeć, wiek, kolor oczu i kolor włosów (SEX, AGE, EYES, HAIR). Za pomocą pola EYES łączymy w listę wszystkie elementy reprezentujące osoby o tym samym kolorze oczu itd. (Zobacz rysunek 13). Można łatwo wyobrazić sobie wydajny algorytm wstawiania nowych elementów do takiego zestawu list. Usuwanie będzie jednak dużo wolniejsze, chyba że posłużymy się listami dwukierunkowymi. Możemy także rozważyć algorytmy o różnym stopniu wydajności; realizujące zadania typu „znajdź wszystkie niebieskookie blondynki w wieku 21–23 lat”; zobacz ćwiczenia 9 i 10. Problemy, w których każdy wierzchołek listy należy jednocześnie do innych list, pojawiają się dosyć często. W symulatorze windy opisany w poprzednim rozdziale występowały elementy należące jednocześnie do list QUEUE i WAIT.

W ramach szczególnego przykładu wykorzystania wskaźnikowych list ortogonalnych przyjrzymy się sposobowi realizacji *macierzy rzadkich* (tj. macierzy dużych rozmiarów, w których większość elementów ma wartość zero). Naszym celem jest posugiwanie się takimi macierzami, tak jak gdyby cała macierz była obecna w pamięci. Ale jednocześnie chcemy zaoszczędzić czas i pamięć, korzystając z faktu, że nie musimy bezpośrednio reprezentować zerowych elementów.

Jednym z dostępnych sposobów, sprawdzającym się w sytuacjach, gdy wymagany jest szybki dostęp do dowolnych elementów macierzy, jest wykorzystanie metod przechowywania i wyszukiwania danych (zob. rozdział 6), umożliwiających odnalezienie $A[j, k]$ na podstawie klucza „ $[j, k]$ ”. Istnieje jednak inny sposób radzenia sobie z macierzami rzadkimi, który często jest nawet bardziej odpowiedni, ponieważ lepiej oddaje strukturę macierzy. Tym sposobem zajmiemy się teraz.

Omawiana reprezentacja składa się z list cyklicznych, po jednej liście na każdy wiersz i kolumnę. Każdy element składa się z trzech słów i pięciu pól:

ROW	UP
COL	LEFT
VAL	

ROW i **COL** to indeksy wiersza i kolumny, w których znajduje się element; **VAL** to wartość elementu macierzy; **LEFT** i **UP** są dowiązaniem odpowiednio do następnego niezerowego elementu stojącego na lewo w tym samym wierszu oraz u góry w tej samej kolumnie. Dla każdego wiersza i kolumny utrzymujemy specjalne atrapy **BASEROW**[*i*] i **BASECOL**[*j*]. Te elementy rozpoznajemy po

$$\text{COL}(\text{LOC}(\text{BASEROW}[i])) < 0 \quad \quad i \quad \quad \text{ROW}(\text{LOC}(\text{BASECOL}[j])) < 0.$$

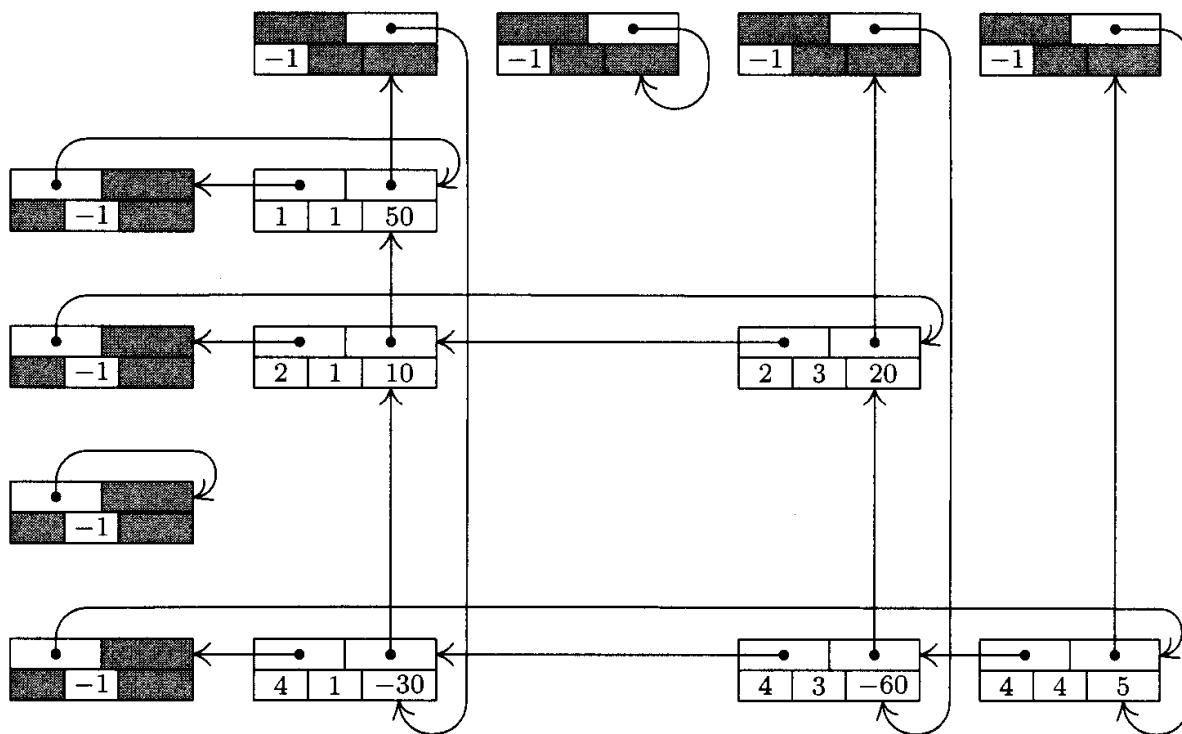
Jak zwykle w liście cyklicznej dowiązanie LEFT w `BASEROW[i]` jest lokacją skrajnie prawej wartości w wierszu, a UP w `BASECOL[j]` wskazuje na najniżej położoną wartość w kolumnie. Jako przykład na rysunku 14 jest pokazana reprezentacja macierzy

$$\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix} \quad (12)$$

W przypadku wykorzystania standardowych struktur tablicowych macierz 200×200 zajęłaby 40000 słów, co przekracza pojemność pamięci wielu komputerów*. Jeżeli jednak taka macierz jest dostatecznie rzadka, to można ją w powyższy sposób reprezentować nawet w 4000-słowowej pamięci komputera MIX. (Zobacz ćwiczenie 11). Czas dostępu do dowolnego elementu $A[j, k]$ też jest całkiem znośny, jeżeli w każdym wierszu i kolumnie jest stosunkowo niewiele elementów. Z uwagi na fakt, że wiele algorytmów macierzowych przegląda macierz wierszami lub kolumnami, reprezentacja wykorzystująca listy z dowiązaniami okazuje się często szybsza od tablicowej.

Jako typowy przykład nietrywialnego algorytmu korzystającego z macierzy rzadkich w powyższej postaci przeanalizujemy operację *osiowania*, będącą istotną częścią algorytmów rozwiązywania równań liniowych, odwracania macierzy

* Pamięć dzisiejszych komputerów osobistych nierzadko sięga setek milionów bajtów (przyp. tłum., rok 2001).



Rys. 14. Reprezentacja macierzy (12), elementy mają postać
Atryby narysowano po stronie lewej i u góry.

LEFT	UP	
ROW	COL	VAL

i rozwiązywania problemów programowania liniowego metodą sympleks. Osiowanianie (*pivot step*) jest następującym przekształceniem macierzy:

Przed osiowaniem	Po osiowaniu
$\begin{array}{ccccc} \text{Kolumna} & & \text{Inna} \\ \text{osiowa} & & \text{kolumna} \\ \hline \dots & \vdots & \vdots & \dots \\ \dots & a & \dots & b & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & c & \dots & d & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$	$\begin{array}{ccccc} \text{Kolumna} & & \text{Inna} \\ \text{osiowa} & & \text{kolumna} \\ \hline \dots & \vdots & \vdots & \dots \\ \dots & 1/a & \dots & b/a & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & -c/a & \dots & d - bc/a & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$

(13)

Zakładamy, że *element osiowy (pivot element)* a jest niezerowy. Na przykład, osiowanie macierzy (12) z elementem 10 w wierszu 2 kolumny 1 w roli elementu osiowego prowadzi do macierzy

$$\begin{pmatrix} -5 & 0 & -100 & 0 \\ 0.1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 5 \end{pmatrix}. \quad (14)$$

Naszym celem jest zaprojektowanie algorytmu wykonującego operację osiowania na macierzach rzadkich w reprezentacji jak na rysunku 14. Jasne jest, że przekształcenie (13) zmienia jedynie te wiersze macierzy, dla których istnieje niezerowy element w kolumnie osiowej oraz że zmienia jedynie te kolumny, dla których istnieje niezerowy element w wierszu osiowym.

Algorytm osiowania polega w znacznej mierze po prostu na zastosowaniu struktur z dowiązaniami, które omawialiśmy wcześniej. W szczególności bardzo przypomina on algorytm 2.2.4A dodający wielomiany. Dwa szczegóły sprawiają jednak, że nasz bieżący problem jest nieco trudniejszy: jeśli w (13) mamy $b \neq 0$ i $c \neq 0$, ale $d = 0$, to w reprezentacji macierzy rzadkiej nie ma elementu d , więc musimy go wstawić. Ponadto, jeśli $b \neq 0$, $c \neq 0$, $d \neq 0$, ale $d - bc/a = 0$, to musimy usunąć element, który się tam wcześniej znajdował. Te operacje wstawiania i usuwania są o wiele ciekawsze w przypadku tablicy dwuwymiarowej niż jednowymiarowej listy. Nasz algorytm sukcesywnie przegląda wiersze macierzy od dołu do góry. Wydajną realizację usuwania i wstawiania osiągamy dzięki wprowadzeniu zbioru zmiennych dowiązaniowych $\text{PTR}[j]$, po jednej na każdą istotną kolumnę. Wartości tych zmiennych przebiegają kolumny z dołu do góry, umożliwiając modyfikację odpowiednich dowiązań w obu wymiarach.

Algorytm S (*Osiowanie w macierzy rzadkiej*). Dla macierzy reprezentowanej jak na rysunku 14 wykonujemy operację osiowania (13). Zakładamy, że PIVOT jest zmienną zawierającą dowiązanie do elementu osiowego. Algorytm korzysta z dodatkowej tablicy zmiennych dowiązaniowych $\text{PTR}[j]$, po jednej dla każdej kolumny macierzy. Zakładamy, że zmienna ALPHA oraz pola VAL elementów zawierają wartości zmiennopozycyjne lub wymierne, podczas gdy wszystkie inne liczby w algorytmie są całkowite.

S1. [Inicjowanie] Przyjmij $\text{ALPHA} \leftarrow 1.0/\text{VAL}(\text{PIVOT})$, $\text{VAL}(\text{PIVOT}) \leftarrow 1.0$ oraz

$$\begin{aligned} \text{IO} &\leftarrow \text{ROW}(\text{PIVOT}), \quad \text{P0} \leftarrow \text{LOC}(\text{BASEROW}[\text{IO}]); \\ \text{J0} &\leftarrow \text{COL}(\text{PIVOT}), \quad \text{Q0} \leftarrow \text{LOC}(\text{BASECOL}[\text{J0}]). \end{aligned}$$

S2. [Obsługa wiersza osiowego] Przyjmij $\text{P0} \leftarrow \text{LEFT}(\text{P0})$, $\text{J} \leftarrow \text{COL}(\text{P0})$. Jeśli $\text{J} < 0$, to przejdź do kroku S3 (przeszliśmy wiersz osiowy). W przeciwnym razie przyjmij $\text{PTR}[\text{J}] \leftarrow \text{LOC}(\text{BASECOL}[\text{J}])$ oraz $\text{VAL}(\text{P0}) \leftarrow \text{ALPHA} \times \text{VAL}(\text{P0})$ i powtóż krok S2.

S3. [Znajdowanie nowego wiersza] Przyjmij $\text{Q0} \leftarrow \text{UP}(\text{Q0})$. (Pozostała część algorytmu polega na kolejnym przetwarzaniu każdego wiersza, od dołu do góry, dla którego istnieje element w kolumnie osiowej). Przyjmij $\text{I} \leftarrow \text{ROW}(\text{Q0})$. Jeśli $\text{I} < 0$, to zakończ wykonanie algorytmu. Jeśli $\text{I} = \text{IO}$, to powtóż krok S3 (wiersz osiowy został już obsłużony). W przeciwnym razie przyjmij $\text{P} \leftarrow \text{LOC}(\text{BASEROW}[\text{I}])$, $\text{P1} \leftarrow \text{LEFT}(\text{P})$. (Wskaźniki P i P1 będą teraz przemieszczać się przez wiersz I od prawej do lewej, a P0 przemieszcza się przez odpowiadające elementy wiersza IO ; porównaj algorytm 2.2.4A. W tym miejscu mamy $\text{P0} = \text{LOC}(\text{BASEROW}[\text{IO}])$).

S4. [Znajdowanie nowej kolumny] Przyjmij $\text{P0} \leftarrow \text{LEFT}(\text{P0})$, $\text{J} \leftarrow \text{COL}(\text{P0})$. Jeśli $\text{J} < 0$, to przyjmij $\text{VAL}(\text{Q0}) \leftarrow -\text{ALPHA} \times \text{VAL}(\text{Q0})$ i wróć do S3. Jeśli $\text{J} = \text{J0}$,

to powtóż krok S4. (Tym samym przetwarzamy elementy kolumny osiowej w wierszu I po przetworzeniu elementów wszystkich innych kolumn, stąd w kroku S7 posługiujemy się wartością $VAL(Q_0)$).

- S5.** [Znajdowanie elementu I, J] Jeśli $COL(P_1) > J$, to przyjmij $P \leftarrow P_1$, $P_1 \leftarrow LEFT(P)$ i powtóż krok S5. Jeśli $COL(P_1) = J$, to idź do kroku S7. W przeciwnym razie idź do kroku S6 (musimy wstawić nowy element w kolumnie J wiersza I).
- S6.** [Wstawianie elementu I, J] Jeśli $ROW(UP(PTR[J])) > I$, to przyjmij wartość $PTR[J] \leftarrow UP(PTR[J])$ i powtóż krok S6. (W przeciwnym razie będziemy mieć $ROW(UP(PTR[J])) < I$; nowy element ma być wstawiony tuż nad $NODE(PTR[J])$ i zaraz na lewo od $NODE(P)$). W przeciwnym razie przyjmij $X \leftarrow AVAIL$, $VAL(X) \leftarrow 0$, $ROW(X) \leftarrow I$, $COL(X) \leftarrow J$, $LEFT(X) \leftarrow P_1$, $UP(X) \leftarrow UP(PTR[J])$, $LEFT(P) \leftarrow X$, $UP(PTR[J]) \leftarrow X$, $P_1 \leftarrow X$.
- S7.** [Osiwanie] Przyjmij $VAL(P_1) \leftarrow VAL(P_1) - VAL(Q_0) \times VAL(P_0)$. Jeśli teraz $VAL(P_1) = 0$, to idź do kroku S8. (Uwaga: Jeśli korzystamy z arytmetyki zmiennopozycyjnej, to warunek „ $VAL(P_1) = 0$ ” należy zastąpić przez „ $|VAL(P_1)| < EPSILON$ ”, a jeszcze lepiej przez warunek „większość cyfr znaczących $VAL(P_1)$ wyzerowała się podczas odejmowania”). W przeciwnym razie przyjmij $PTR[J] \leftarrow P_1$, $P \leftarrow P_1$, $P_1 \leftarrow LEFT(P)$ i idź do kroku S4.
- S8.** [Usuwanie elementu I, J] Jeśli $UP(PTR[J]) \neq P_1$ (lub, co jest dokładnie tym samym, jeśli $ROW(UP(PTR[J])) > I$), to przyjmij $PTR[J] \leftarrow UP(PTR[J])$ i powtóż krok S8; w przeciwnym razie przyjmij $UP(PTR[J]) \leftarrow UP(P_1)$, $LEFT(P) \leftarrow LEFT(P_1)$, $AVAIL \leftarrow P_1$, $P_1 \leftarrow LEFT(P)$. Wróć do kroku S4. ■

Zaprogramowanie powyższego algorytmu pozostawiamy jako bardzo wartościowe ćwiczenie dla Czytelnika (zobacz ćwiczenie 15). Warto tu zwrócić uwagę, że dla elementów $BASEROW[i]$, $BASECOL[j]$ wystarczy zarezerwować po jednym słowie pamięci, gdyż większość ich pól jest nieistotna. (Zobacz zaciemnione pola na rysunku 14 i program w punkcie 2.2.5). Ponadto w celu uzyskania dalszych oszczędności pamięci możemy jako $ROW(LOC(BASECOL[j]))$ przechowywać wartość $-PTR[j]$. Czas pracy algorytmu S jest z grubsza proporcjonalny do liczby elementów macierzy, których dotyczy operacja osiowania.

Reprezentacja macierzy rzadkich za pomocą ortogonalnych list cyklicznych jest pouczająca, ale analiza numeryczna oferuje lepsze metody. Zobacz Fred G. Gustavson, *ACM Trans. on Math. Software* 4 (1978), 250–269; zobacz także: algorytmy grafowe i sieciowe w rozdziale 7.

ĆWICZENIA

- [17] Podaj wzór na $LOC(A[J,K])$, gdy A jest taką macierzą, jak (1), a każdy element tablicy ma dwa słowa długości. Załóż, że elementy są przechowywane w kolejnych komórkach pamięci w leksykograficznym porządku indeksów.
- [21] Wzór (6) wyprowadziliśmy przy założeniu $0 \leq I_r \leq d_r$ dla $1 \leq r \leq k$. Podaj ogólny wzór mający zastosowanie w przypadku $l_r \leq I \leq u_r$, gdzie l_r i u_r są dowolnymi dolnymi i górnymi ograniczeniami wymiarów.

3. [21] W tekście zajmujemy się dolną macierzą trójkątną $A[j, k]$ dla $0 \leq k \leq j \leq n$. Jak należy zmodyfikować wywód dla macierzy o indeksach zaczynających się od 1, a nie od zera, tzn. $1 \leq k \leq j \leq n$?

4. [22] Pokaż, że jeśli w pamięci przechowujemy zawartość *górnej* macierzy trójkątnej $A[j, k]$ dla $0 \leq j \leq k \leq n$ według leksykograficznego porządku indeksów, to spełniony jest warunek (8). Znajdź odpowiedni wzór na $\text{LOC}(A[J, K])$.

5. [20] Pokaż, że nawet wtedy, gdy A jest *macierzą trójkątną* jak (9), można załadować wartość $A[J, K]$ do rejestru A za pomocą jednego rozkazu komputera MIX, jeżeli skorzysta się z adresowania pośredniego (zobacz ćwiczenie 2.2.2–3). (Załącz, że wartości J i K znajdują się w rejestrach indeksowych).

► **6.** [M24] Zajmijmy się „tablicami tetrahedrycznymi” $A[i, j, k]$, $B[i, j, k]$, gdzie $0 \leq k \leq j \leq i \leq n$ dla A i $0 \leq i \leq j \leq k \leq n$ dla B . Przypuśćmy, że obie tablice są zapisane w kolejnych lokacjach pamięci w porządku leksykograficznym indeksów. Pokaż, że $\text{LOC}(A[I, J, K]) = a_0 + f_1(I) + f_2(J) + f_3(K)$ dla pewnych funkcji f_1, f_2, f_3 . Czy $\text{LOC}(B[I, J, K])$ można wyrazić w podobnej postaci?

7. [M23] Znajdź ogólny wzór adresowania k -wymiarowej tablicy tetrahedrycznej postaci $A[i_1, i_2, \dots, i_k]$, gdzie $0 \leq i_k \leq \dots \leq i_2 \leq i_1 \leq n$.

8. [33] (P. Wegner) Przypuśćmy, że chcemy przechowywać w pamięci sześć tablic tetrahedrycznych $A[I, J, K]$, $B[I, J, K]$, $C[I, J, K]$, $D[I, J, K]$, $E[I, J, K]$ i $F[I, J, K]$, gdzie $0 \leq K \leq J \leq I \leq n$. Czy można podać równie elegancką metodę, co metoda (10) dla przypadku dwuwymiarowego?

9. [22] Przypuśćmy, że taka struktura, jak na rysunku 13, ale dużo większa, została utworzona w ten sposób, że dowiązania biegą w takich kierunkach, jak na rysunku (tj. $\text{LINK}(X) < X$ dla wszystkich elementów i dowiązań). Zaprojektuj algorytm, który znajduje adresy wszystkich niebieskookich blondynek w wieku 21–23 lat, przehodząc po dowiązaniach w ten sposób, że w momencie zakończenia algorytmu każdą z list FEMALE, A21, A22, A23, BLOND, BLUE algorytm przeszedł przynajmniej jeden raz.

10. [26] Czy potrafisz wymyślić lepszą strukturę do przechowywania danych osobowych, dzięki której przeszukiwanie opisane w poprzednim ćwiczeniu będzie bardziej wydajne? (Nie wystarczy odpowiedzieć „tak” albo „nie”).

11. [11] Przypuśćmy, że mamy macierz 200×200 , taką że w każdym jej wierszu są co najwyżej cztery niezerowe elementy. Ile pamięci potrzeba na przechowywanie tej macierzy, jeżeli korzystamy z reprezentacji jak na rysunku 14, a każdy element zajmuje trzy słowa, poza atrapami, zajmującymi po jednym słowie?

► **12.** [20] Jakie są wartości $\text{VAL}(Q0)$, $\text{VAL}(P0)$ i $\text{VAL}(P1)$ na początku kroku S7 w sensie notacji a, b, c, d ze schematu (13)?

► **13.** [22] Dlaczego na rysunku 14 są listy cykliczne, a nie zwykłe listy liniowe? Czy można zmienić algorytm S tak, by nie korzystał z list cyklicznych?

14. [22] Algorytm S faktycznie oszczędza czas, wykonując operacje osiowania w macierzy rzadkiej, ponieważ pomija on w rozważaniach te kolumny, w których wartość elementu w wierszu osiowania wynosi zero. Pokaż, że tę oszczędność czasu działania algorytmu można osiągnąć w dużej macierzy rzadkiej zapisanej sekwencyjnie przy użyciu pomocniczej tablicy $\text{LINK}[j]$, $1 \leq j \leq n$.

► **15.** [29] Napisz program w języku MIXAL realizujący algorytm S. Załącz, że pole VAL zawiera liczbę zmiennopozycyjną i można na nim wykonywać operacje za pomocą

rozkazów zmiennopozycyjnych komputera MIX: FADD, FSUB, FMUL i FDIV. Załóż dla uproszczenia, że FADD i FSUB zwracają zero, jeżeli wynik odejmowania lub dodawania jest bardzo mały. Dzięki temu można bezpieczne korzystać z warunku „VAL(P1) = 0” w kroku S7. Operacje zmiennopozycyjne korzystają wyłącznie z rA (nie korzystają z rX).

16. [25] Zaprojektuj algorytm *kopiowania* macierzy rzadkiej. (Innymi słowy, algorytm ma przerabiać umieszczoną w pamięci reprezentację macierzy w postaci z rysunku 14 na dwie oddzielne reprezentacje tej samej macierzy).

17. [26] Zaprojektuj algorytm *mnożenia* macierzy rzadkich. Mając dane macierze A i B, algorytm ma zbudować nową macierz C, gdzie $C[i,j] = \sum_k A[i,k]B[k,j]$. Dwie macierze wejściowe i macierz wyjściowa powinny mieć taką reprezentację, jak na rysunku 14.

18. [22] Poniższy algorytm zastępuje macierz jej macierzą odwrotną przy założeniu, że elementami macierzy są $A[i,j]$, dla $1 \leq i, j \leq n$:

i) Dla $k = 1, 2, \dots, n$ wykonuj co następuje: przejrzyj wszystkie kolumny wiersza k nie wykorzystane do tej pory jako kolumny osiowe w poszukiwaniu elementu o największej wartości bezwzględnej. Przypisz do $C[k]$ numer kolumny, w której znaleziono taki element, i wykonaj operację osiowania, przyjmując ten element za element osiowy. (Jeśli wszystkie takie elementy są zerami, to macierz jest osobliwa i nie można jej odwrócić).

ii) Spermutuj wiersze i kolumny w ten sposób, żeby to, co było wierszem k , stało się wierszem $C[k]$, a to co było kolumną $C[k]$, stało się kolumną k .

Ćwiczenie polega na posłużeniu się opisanym algorytmem do „ręcznego” odwrócenia macierzy

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}.$$

19. [31] Zmodyfikuj algorytm opisany w ćwiczeniu 18 tak, by obliczał odwrotność macierzy rzadkiej w reprezentacji z rysunku 14. Zwróć szczególną uwagę na wydajną realizację permutacji wierszy i kolumn w kroku (ii).

20. [20] Macierz trójdiamondalna ma wszystkie elementy równe zero, poza elementami a_{ij} , dla których $|i - j| \leq 1$, $1 \leq i, j \leq n$. Pokaż, że istnieje funkcja odwzorowania postaci

$$LOC(A[I, J]) = a_0 + a_1 I + a_2 J, \quad |I - J| \leq 1,$$

pozwalającej na reprezentowanie wszystkich istotnych elementów macierzy trójdiamondalnej w $(3n - 2)$ kolejnych komórkach pamięci.

21. [20] Zaproponuj funkcję odwzorowania pamięci dla macierzy $n \times n$, gdzie n jest zmienną. Elementy $A[I, J]$ dla $1 \leq I, J \leq n$ powinny zajmować n^2 kolejnych lokacji, niezależnie od wartości n .

22. [M25] (P. Chowla, 1961) Znajdź wielomian $p(i_1, \dots, i_k)$, który przyjmuje każdą nieujemną wartość całkowitą dokładnie raz, gdy indeksy (i_1, \dots, i_k) przebiegają k -wymiarowe wektory liczb nieujemnych. Dodatkowo spełniona ma być własność „ $i_1 + \dots + i_k < j_1 + \dots + j_k$ pociąga za sobą $p(i_1, \dots, i_k) < p(j_1, \dots, j_k)$ ”.

23. [23] Macierz rozszerzalna ma początkowo wymiary 1×1 , a po osiągnięciu rozmiaru $m \times n$ może zostać powiększona do $(m+1) \times n$ lub do $m \times (n+1)$ przez dodanie nowego wiersza lub kolumny. Pokaż, że dla takiej macierzy istnieje prosta funkcja odwzorowania

pamięci, dla której elementy $A[I, J]$ zajmują mn kolejnych lokacji, dla $0 \leq I < m$ i $0 \leq J < n$; elementy nie mogą zmieniać lokacji przy rozszerzaniu macierzy.

- 24. [25] (*Sztuczka z tablicą rzadką*) Przypuśćmy, że chcemy korzystać z bardzo dużej tablicy o swobodnym dostępie, ale nie będziemy odwoływać się do każdego elementu. Chcemy, by przy pierwszym dostępie $A[k]$ miało wartość zero, ale szkoda nam czasu na wpisywanie zer do wszystkich elementów. Wyjaśnij, w jaki sposób dla dowolnego k poprawnie odczytywać i zapisywać wartość elementu $A[k]$, nie zakładając niczego o faktycznej początkowej zawartości pamięci i wykonując jedynie małą stałą liczbę dodatkowych operacji przy dostępie do tablicy.

2.3. DRZEWA

Zajmiemy się teraz drzewami, najważniejszą strukturą nieliniową pojawiającą się w algorytmach komputerowych. Mówiąc ogólnie, w strukturze drzewiastej związki między elementami tworzą „rozgałęzienia”, podobne do konarów prawdziwych drzew.

Drzewo zdefiniujemy formalnie jako zbiór T jednego lub więcej elementów zwanych *węzłami*, takich że

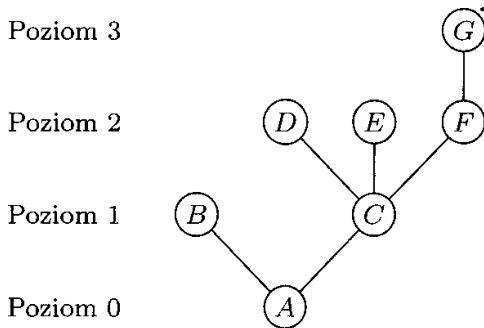
- a) istnieje jeden wyróżniony węzeł nazywany *korzeniem* drzewa, $\text{root}(T)$; oraz
- b) pozostałe węzły (z wyłączeniem korzenia) są podzielone na $m \geq 0$ rozłącznych zbiorów T_1, \dots, T_m , z których każdy jest drzewem. Drzewa T_1, \dots, T_m są nazywane *poddrezewami* korzenia.

Podana definicja jest rekurencyjna: zdefiniowaliśmy drzewo za pomocą innych drzew. Oczywiście nie występuje tu problem definiowania nieznanego przez nieznane, ponieważ drzewa o jednym węzle muszą składać się wyłącznie z korzenia, a drzewa o $n > 1$ węzłach są zdefiniowane za pomocą drzew o liczbie węzłów mniejszej niż n . Wynika stąd, że podana definicja dobrze określa, co to jest drzewo o dwóch, trzech, czy więcej węzłach. Drzewa da się zdefiniować nierekurencyjnie (zobacz na przykład ćwiczenia 10, 12 i 14 oraz punkt 2.3.4), ale definicja rekurencyjna wydaje się najwłaściwsza, gdyż rekurencja jest nieodzowną cechą struktury drzewiastej. Rekurencyjny charakter drzew daje się zaobserwować również w naturze: z pąków na młodych drzewach wyrastają poddrzewa, na których pojawiają się kolejne pąki itd. Ćwiczenie 3 pokazuje, w jaki sposób na podstawie podanej definicji rekurencyjnej można ściśle udowodnić fakty dotyczące drzew za pomocą indukcji względem liczby węzłów drzewa.

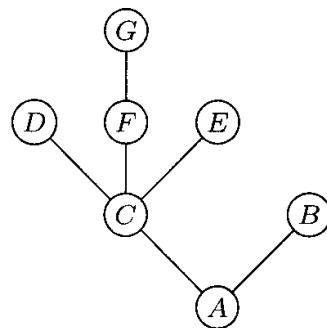
Z naszej definicji wynika, że każdy węzeł drzewa jest korzeniem pewnego poddrzewa zawartego w „większym” drzewie. Liczba poddrzew węzła jest nazywana *stopniem* tego węzła. Węzeł o stopniu zero nazywamy *liściem* lub *węzłem zewnętrznym*. Węzeł nie będący liściem nazywamy *węzłem wewnętrznym*. Poziom węzła w drzewie T jest zdefiniowany rekurencyjnie: poziom korzenia $\text{root}(T)$ równa się zero, a poziom każdego innego węzła jest o jeden większy niż poziom korzenia w najmniejszym zawierającym go poddrzewie.

Poziomy węzłów są zaznaczone na rysunku 15, przedstawiającym przykładowe drzewo o siedmiu węzłach. Korzeniem drzewa jest węzeł A , a drzewo to ma dwa poddrzewa $\{B\}$ i $\{C, D, E, F, G\}$. Korzeniem drzewa $\{C, D, E, F, G\}$ jest węzeł C . Poziom węzła C w całym drzewie równa się 1. Węzeł C ma trzy poddrzewa $\{D\}$, $\{E\}$ i $\{F, G\}$, stąd stopień C równa się 3. Liśćmi na rysunku 15 są B , D , E , i G ; F jest jedynym węzłem o stopniu 1; G jest jedynym węzłem o stopniu 3.

Jeżeli wzajemny porządek poddrzew T_1, \dots, T_m w części (b) definicji jest istotny, to mówimy, że drzewo jest *uporządkowane*. Gdy w drzewie uporządkowanym $m \geq 2$, jest sens mówić o T_2 jako o „drugim poddrzewie” korzenia itd. Drzewa uporządkowane są przez niektórych autorów nazywane „drzewami płaskimi”, ponieważ w przypadku takich drzew istotne jest, w jakiej kolejności rysujemy poddrzewa na płaszczyźnie. Jeśli nie chcemy rozróżniać drzew, które



Rys. 15. Drzewo.

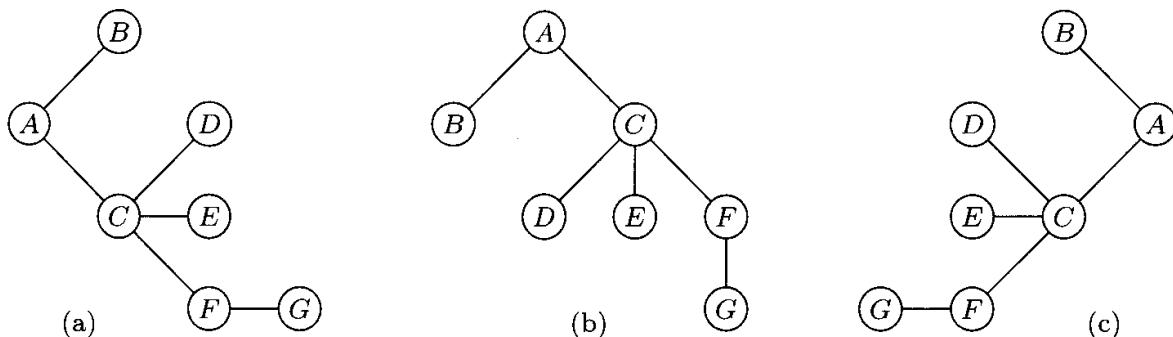


Rys. 16. Inne drzewo.

różnią się jedynie kolejnością poddrzew, mówimy o drzewach *zorientowanych*. Sposób reprezentacji drzew w pamięci komputera narzuca niejawne uporządkowanie poddrzew w każdym drzewie, zatem w większości przypadków interesować nas będą drzewa uporządkowane. Ustalamy zatem, że *wszystkie drzewa, o których mówimy, są uporządkowane, jeżeli wyraźnie nie powiedziano inaczej*. W związku z tym drzewa z rysunków 15 i 16 będą zasadniczo uważać za różne, choć byłyby one takie same jako drzewa zorientowane.

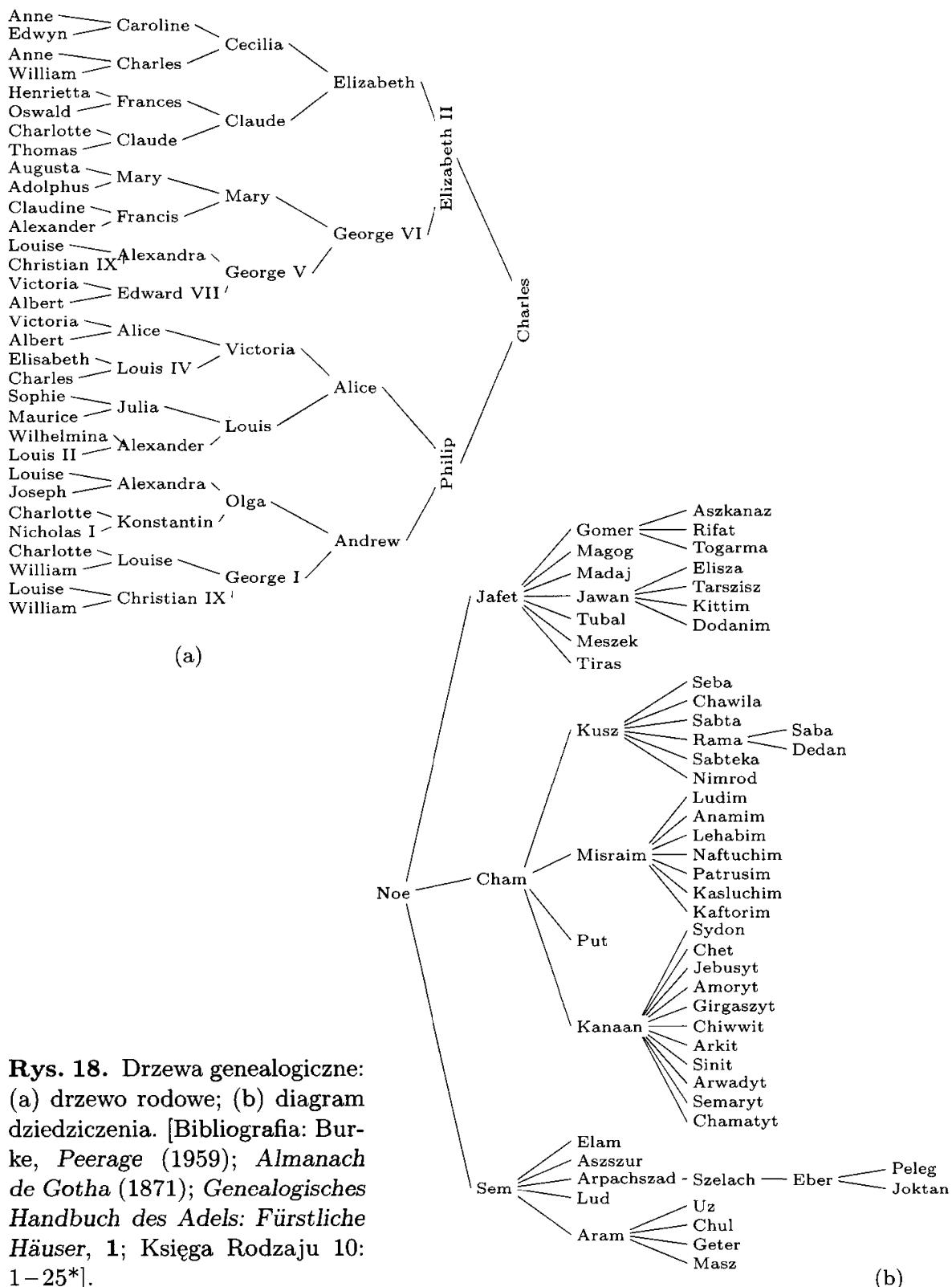
Lasem nazywamy zbiór (zazwyczaj rozważać będziemy w tym miejscu zbiory uporządkowane) zera lub więcej rozłącznych drzew. Innym sposobem wypowiedzenia części (b) definicji drzewa jest stwierdzenie: *węzły drzewa z wyłączeniem korzenia tworzą las*.

Abstrakcyjne drzewa i lasy niewiele się różnią. Jeżeli usuniemy korzeń drzewa, otrzymamy las. W drugą stronę, jeśli wszystkie drzewa w lesie uczynimy poddrzewami jednego dodatkowego węzła, to otrzymamy drzewo. Z tego powodu słowa „drzewa” i „lasy” są przy nieformalnym omawianiu struktur danych używane niemal wymiennie.



Rys. 17. Jak rysować drzewo?

Drzewa można rysować na wiele sposobów. Poza sposobem pokazanym na rysunku 15 istnieją przynajmniej trzy inne metody, pokazane na rysunku 17. Różnią się położeniem korzenia. To nie żart – ustalenie sposobu rysowania schematu drzewa ma istotne konsekwencje, jeśli opisując położenie węzłów, chcemy posługiwać się terminami „nad”, „wyżej niż”, „skrajnie prawy” itp. Niektóre algorytmy drzewowe są znane jako „zstępujące” lub „wstępujące”. Taka terminologia będzie prowadzić do pomyłek, jeżeli nie ustalimy konwencji rysowania schematu drzewa.



Rys. 18. Drzewa genealogiczne:
(a) drzewo rodowe; (b) diagram dziedziczenia. [Bibliografia: Burke, Peerage (1959); Almanach de Gotha (1871); Genealogisches Handbuch des Adels: Fürstliche Häuser, 1; Księga Rodzaju 10: 1–25*].

* W polskim tłumaczeniu *Księgi Rodzaju* [„Biblia Tysiaclecia” wyd. III poprawione, Poznań – Warszawa, 1980] nie występuje syn Mizraima o imieniu Kaftorim, ani synowie Kanaana o imionach Jebusyt, Amoryt, Girgaszyt, Chiwwit, Arkit, Sinit, Arwadyt, Semaryt i Chamatyat. W odróżnieniu od tłumaczenia angielskiego [na przykład *King James Bible*, wyd. 2000] tłumaczenie polskie mówi o Jebusytach, Amorytach, Girgaszytach, Chiwwitach, Arkitach, Sinitach, Arwadytach, Semarytach i Chamatytach będących potomkami Kanaana oraz Kaftorytach będących potomkami Mizraima (przyp. tłum.).

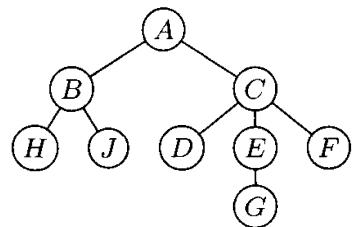
Może się wydawać, że postać drzewa na rysunku 15 jest najprostsza, ponieważ odpowiada temu, jak drzewa rosną w naturze; wobec braku innych powodów, moglibyśmy dostosować się do tej tradycji uświęconej prawami przyrody. W świetle powyższej argumentacji autor, przygotowując niniejszą książkę, trzymał się konwencji „korzeń na dole”. Ale po dwóch latach prób przekonał się o swym błędzie: studiując literaturę komputerową oraz przeprowadzając wiele nieformalnych dyskusji z informatykami, stwierdził, że ludzie rysowali drzewa *korzeniem do góry* w więcej niż 80% zbadanych przypadków. Istnieje silna tendencja do rozszerzania odręcznie rysowanych diagramów w dół (co łatwo wytlumaczyć, biorąc pod uwagę sposób, w jaki piszemy). Nawet termin „poddrezewo”, w odróżnieniu do „naddrzewa”, sugeruje coś leżącego niżej. Z tych obserwacji wynika, że rysunek 15 jest narysowany do góry nogami. Drzewa prawie zawsze będziemy rysowali jak na rysunku 17(b), z korzeniem u góry i liśćmi u dołu.

Potrzebujemy dobrej terminologii do opisu drzew. Zamiast nieco wieloznacznych terminów „pod” i „nad”, będziemy generalnie stosować terminologię *genealogiczną*. Na rysunku 18 są pokazane dwa najczęściej spotykane typy drzew genealogicznych. Te dwa typy są istotnie różne: *drzewo rodowe (pedigree)* przedstawia przodków, a *diagram dziedziczenia (lineal chart)* potomków jednej osoby.

Jeżeli wystąpi małżeństwo krewnych, drzewo rodowe przestaje być drzewem, bo przy naszej definicji gałęzie drzewa nie mogą się połączyć. By zaradzić temu problemowi, na rysunku 18(a) królowa Wiktoria i książę Albert w szóstym pokoleniu występują dwukrotnie, a król Chrystian i królowa Luiza występują i w piątym, i w szóstym pokoleniu. Drzewo rodowe można uważać za prawdziwe drzewo, jeżeli węzły odpowiadają nie osobom, a „osobom w roli matki lub ojca konkretnego dziecka”.

Standardowa terminologia wykorzystywana do opisu drzew pochodzi od drugiej postaci drzewa genealogicznego – diagramu dziedziczenia: o każdym korzeniu mówimy, że jest *rodzicem* korzeni jego poddrzew, a o korzeniach poddrzew mówimy, że są *rodzeństwem*, no i *dziećmi* swojego rodzica. Korzeń całego drzewa nie ma rodzica. Na przykład na rysunku 19 C ma troje dzieci D, E i F; E jest rodzicem G; B i C są rodzeństwem. Można oczywiście rozszerzyć to nazewnictwo; na przykład G jest prawnukiem A; B jest ciotką lub wujem F; H i F są krewnymi pierwszego stopnia. Niektórzy autorzy używają określeń męskich: „ojciec, syn, brat” zamiast „rodzic, dziecko, rodzeństwo”; inni używają terminów „matka, córka, siostra”. W każdym jednak przypadku węzeł ma jednego rodzica. Słów *przodek* i *potomek* używamy na oznaczenie związku rozciągającego się być może na wiele poziomów drzewa: potomkami C na rysunku 19 są D, E, F i G; przodkami G są E, C i A.

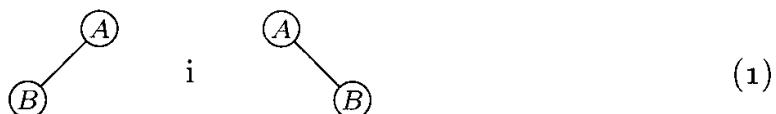
Drzewo genealogiczne na rysunku 18(a) jest przykładem *drzewa binarnego*, będącego bardzo ważnym rodzajem struktury drzewiastej. Czytelnik zapewne spotkał drzewa binarne, kibicując zawodnikom na przykład podczas turniejów tenisowych. W drzewie binarnym każdy węzeł ma co najwyżej dwa poddrzewa,



Rys. 19. Tradycyjny schemat drzewa.

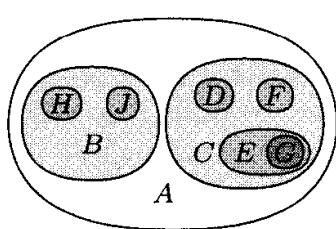
a gdy jest tylko jedno, to czynimy rozróżnienie między poddrzewem lewym i prawym. Bardziej formalnie definiujemy drzewo binarne jako skończony zbiór węzłów, który *albo jest pusty, albo składa się z korzenia i dwóch rozłącznych drzew binarnych nazywanych lewym i prawym poddrzewem korzenia*.

Tej rekurencyjnej definicji drzewa binarnego należy się bacznie przyjrzeć. Zauważmy, że drzewo binarne *nie* jest szczególnym przypadkiem drzewa. Drzewo binarne to inna struktura (chociaż odnotujemy wiele związków między drzewami i drzewami binarnymi). Na przykład drzewa binarne



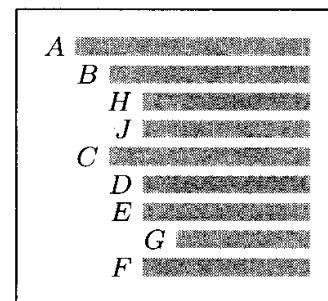
są różne – w jednym przypadku korzeń ma puste prawe poddrzewo, a w drugim prawe poddrzewo jest niepuste. Jeżeli powyższe rysunki potraktowalibyśmy jako schematy zwykłych drzew, to przedstawałyby to samo drzewo. Drzewo binarne może być puste, podczas gdy zwykłe drzewo nie może. Z powyższych powodów będziemy z dużą ostrożnością używać słowa „binarny” do rozróżnienia drzew binarnych i zwykłych drzew. Niektórzy autorzy definiują drzewa binarne nieco odmiennie (zobacz ćwiczenie 20).

Strukturę drzewa można przedstawić graficznie na kilka sposobów, nie nasuwających wcale skojarzeń z prawdziwymi drzewami. Na rysunku 20 są pokazane trzy schematy odpowiadające strukturze drzewa z rysunku 19. Rysunek 20(a) to drzewo *zorientowane*; ten schemat jest szczególnym przypadkiem ogólniejszej koncepcji tzw. *zbiorów zagnieżdzonych*, tj. kolekcji zbiorów, w której każde dwa zbiory albo są rozłączne, albo jeden jest podzbiorem drugiego. (Zobacz ćwiczenie 10). Część (b) rysunku 20 przedstawia wierszową notację zagnieżdzonych, podczas gdy (a) zawiera te same informacje w postaci diagramu. Na część (b) można patrzeć również jak na wyrażenie algebraiczne zawierające zagnieżdżone nawiasy. Część (c) to jeszcze inne przedstawienie struktury drzewiastej, tzw. *wcięcia*. Liczba różnych przedstawień struktury drzewiastej jest przesłanką świadczącą o tym, że jest to ważne pojęcie, tak w programowaniu, jak i w życiu codziennym. Każda klasyfikacja hierarchiczna prowadzi do struktury drzewiastej.



(a)

$$(A(B(H)(J))(C(D)(E(G))(F)))$$



(b)

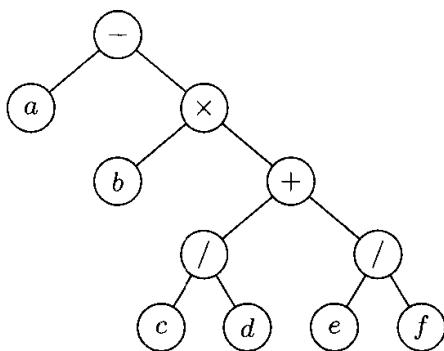
(c)

Rys. 20. Inne metody przedstawiania struktury drzewiastej: (a) zbiory zagnieżdzone; (b) zagnieżdżone nawiasy; (c) wcięcia.

Za pomocą wyrażenia algebraicznego można niejawnie określić strukturę drzewiastą, wyznaczaną nie tylko przez nawiasy. Na przykład na rysunku 21 jest pokazane drzewo odpowiadające wyrażeniu

$$a - b(c/d + e/f). \quad (2)$$

Tradycyjne konwencje matematyczne, zgodnie z którymi mnożenie i dzielenie ma większy priorytet niż dodawanie i odejmowanie, umożliwiają posługiwanie się uproszczoną postacią wzoru (2) zamiast w pełni ponawiasowanym wyrażeniem „ $a - (b \times ((c/d) + (e/f)))$ ”. Ta odpowiedniość wyrażeń i drzew ma bardzo duże znaczenie praktyczne.



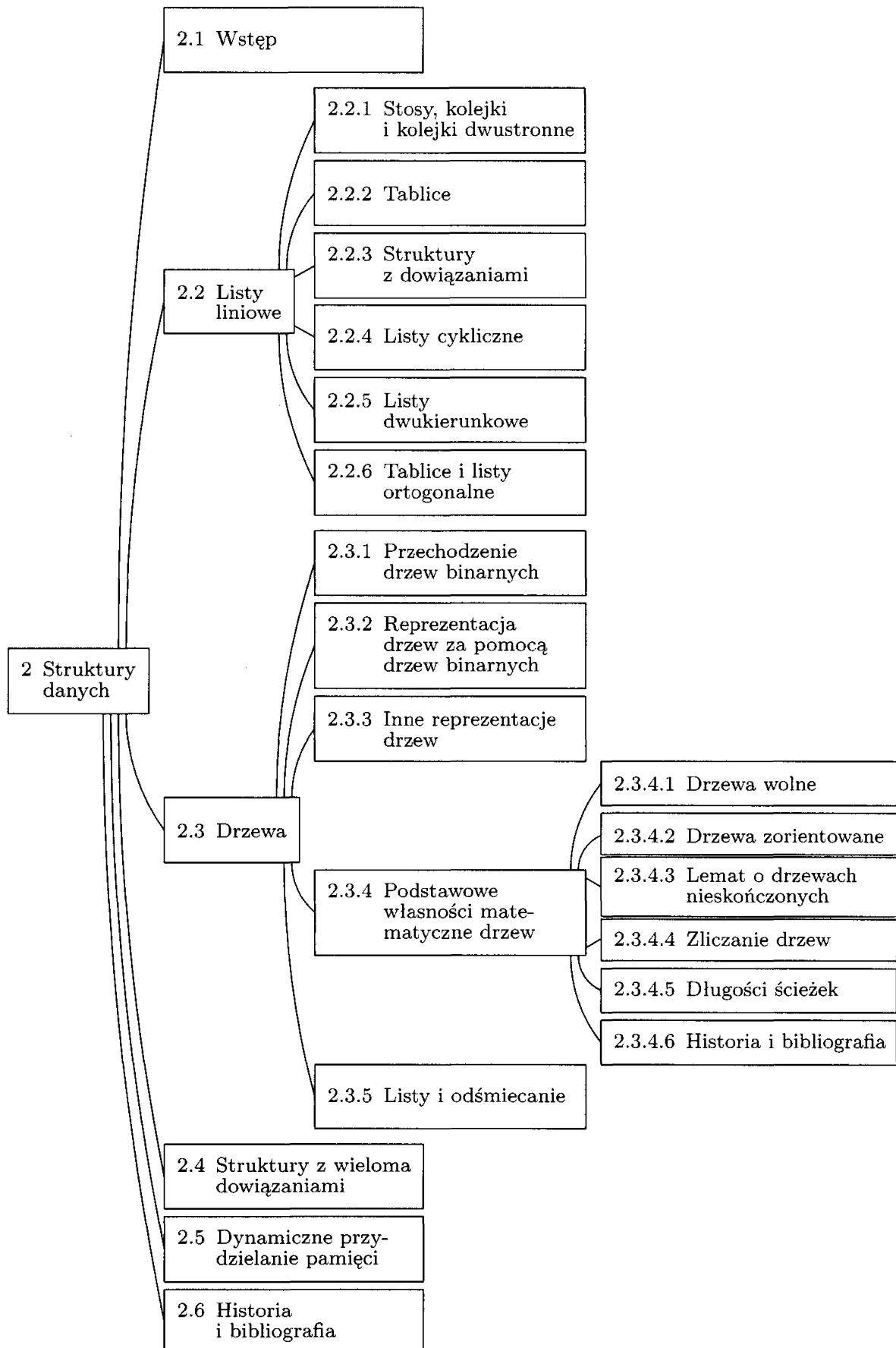
Rys. 21. Drzewowa reprezentacja wyrażenia (2).

Zauważmy, że lista wierszy z wcięciami przedstawiona na rysunku 20(c) wygląda bardzo podobnie do spisu treści. W istocie niniejsza książka też ma strukturę drzewiastą. Strukturę rozdziału 2 widać na rysunku 22. Odnotujmy kolejne istotne spostrzeżenie: *metoda numeracji rozdziałów w niniejszej książce jest kolejnym sposobem wyrażania struktury drzewiastej*. Taką metodę nazywa się często „notacją dziesiętną Dewey'a” dla drzew, przez analogię do systemu klasyfikacji używanego w bibliotekach. Notacja dziesiętna Dewey'a drzewa z rysunku 19 to:

$$\begin{aligned} & 1 A; \quad 1.1 B; \quad 1.1.1 H; \quad 1.1.2 J; \quad 1.2 C; \\ & \quad 1.2.1 D; \quad 1.2.2 E; \quad 1.2.2.1 G; \quad 1.2.3 F. \end{aligned}$$

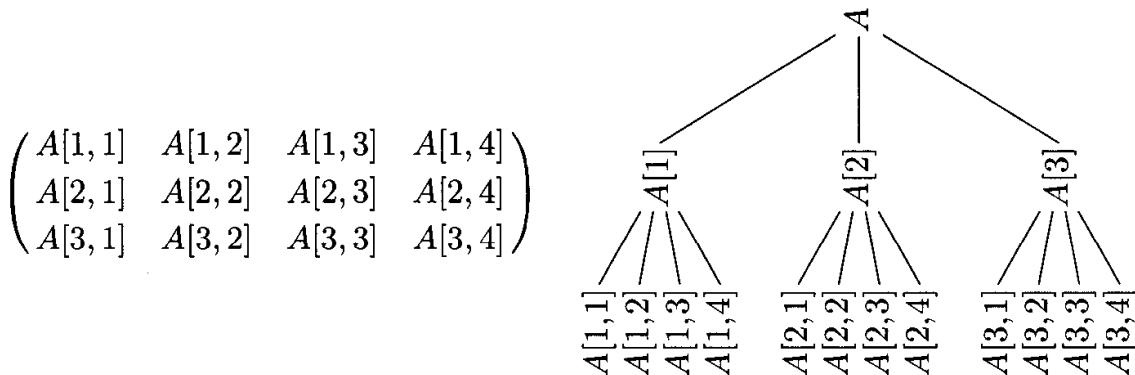
Notację dziesiętną Dewey'a można zastosować do dowolnego lasu: korzeniowi k -tego drzewa nadajemy numer k , a jeśli α jest numerem węzła o stopniu m , to jego dzieciom nadajemy numery $\alpha.1, \alpha.2, \dots, \alpha.m$. Notacja dziesiętna Dewey'a spełnia wiele prostych własności matematycznych i jest pożytecznym narzędziem w analizie drzew. Za jej pomocą można na przykład wyznaczyć naturalny porządek węzłów dowolnego drzewa, analogiczny do porządku rozdziałów w tej książce. Podrozdział 2.3 występuje przed punktem 2.3.1, a po punkcie 2.2.6.

Istnieje subtelny związek między notacją dziesiętną Dewey'a a indeksowaniem zmiennych. Jeśli F jest lasem, to możemy ustalić, że $F[1]$ oznacza zbiór poddrzew pierwszego drzewa, $F[1][2] \equiv F[1, 2]$ oznacza zbiór poddrzew drugiego poddrzewa drzewa $F[1]$, $F[1, 2, 1]$ oznacza pierwszy podlas $F[1, 2]$ itd. Węzeł $a.b.c.d$ w notacji dziesiętnej Dewey'a to rodzin korzeni drzew ($F[a, b, c, d]$). Taka notacja jest rozszerzeniem zwykłej notacji indeksowej, ponieważ dopuszczalny zakres indeksu zależy od wartości indeksów na wcześniejszych pozycjach.



Rys. 22. Struktura rozdziału 2.

W szczególności możemy patrzeć na tablicę prostokątną jako na szczególny przypadek drzewa lub lasu. Oto dwie reprezentacje macierzy 3×4 :



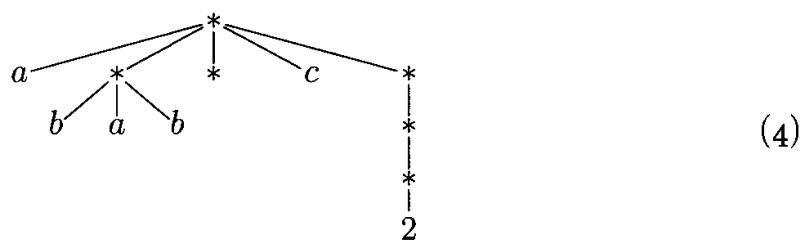
Należy jednak zauważać, że taka struktura drzewiasta nie odzwierciedla dokładnie struktury macierzy – pozwala uchwycić związki między wierszami, ale nie między kolumnami.

Na las można z kolei patrzeć jak na szczególny przypadek tzw. *Listy*. Słowo „lista” występuje tu w bardzo technicznym znaczeniu. By to zaznaczyć będziemy pisać „Lista” wielką literą. Listę definiujemy rekurencyjnie jako skończony ciąg zera lub więcej atomów lub *List*. „Atom” jest tu czymś niezdefiniowanym, oznaczającym element z dowolnego uniwersum obiektów, odróżnialny od Listy. Za pomocą oczywistych konwencji notacyjnych możemy rozróżnić atomy i Listy, a także w wygodny sposób przedstawiać porządek elementów Listy. Rozważmy na przykład napis

$$L = (a, (b, a, b), (), c, (((2)))), \quad (3)$$

opisujący Listę pięcioelementową: najpierw atom a , potem Lista (b, a, b) , potem Lista pusta $()$, potem atom c , na końcu Lista $((2))$). Ostatnia Lista zawiera jeden element – Listę $((2))$, która zawiera Listę (2) , która zawiera atom 2.

Liście L odpowiada następująca struktura drzewiasta:

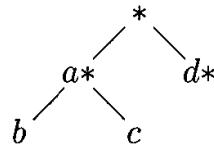


Gwiazdki oznaczają wystąpienie List (elementów nieatomowych). Do List możemy zastosować tę samą notację indeksową, co do lasów; na przykład $L[2] = (b, a, b)$ i $L[2, 2] = a$.

Węzły oznaczające Listy w (4) nie zawierają żadnych informacji (poza informacją, że są Listami). Ale nieatomowym elementom List możemy przypisać etykiety, jak robimy to w przypadku drzew i innych struktur. Stąd

$$A = (a:(b, c), d:())$$

odpowiada drzewu



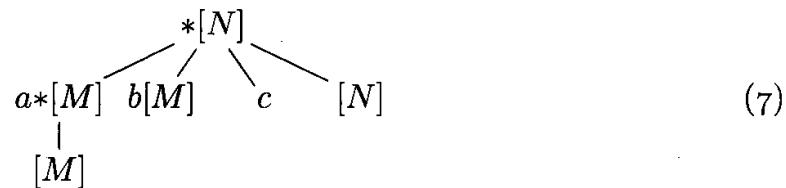
Istotna różnica między Listami a drzewami polega na tym, że Listy mogą się nakładać (tj. pod-Listy nie muszą być rozłączne), a nawet mogą być rekurencyjne (mogą zawierać same siebie). Lista

$$M = (M) \quad (5)$$

nie odpowiada żadnemu drzewu, podobnie jak Lista

$$N = (a:M, b:M, c, N). \quad (6)$$

(W tych przykładach wielkie litery oznaczają Listy, a małe litery etykiety i atomy). Możemy schematycznie przedstawić (5) i (6), oznaczając gwiazdką miejsce, w którym lista jest zdefiniowana:



W istocie Listy nie są tak skomplikowane, jak mogłyby sugerować powyższe przykłady. Listy są prostym uogólnieniem list liniowych, opisanych w podrozdziale 2.2, z tym że element Listy liniowej może być zmienną zawierającą dowiązanie do Listy (być może siebie samej).

Podsumowanie: w wielu miejscach spotykamy cztery ścisłe związane ze sobą struktury: drzewa, lasy, drzewa binarne i Listy. Świadczy to o ich dużym znaczeniu w konstrukcji algorytmów. Zapoznaliśmy się z różnymi sposobami rysowania tych struktur, a także z terminologią i notacją pozwalającą o nich mówić i pisać. W kolejnych rozdziałach uważnie przyjrzymy się powyższym strukturam.

ĆWICZENIA

1. [18] Ile jest różnych drzew o trzech węzłach A , B i C ?
2. [20] Ile jest różnych drzew *zorientowanych* o trzech węzłach A , B i C ?
3. [M20] Udowodnij, korzystając z definicji, że w drzewie do każdego węzła X prowadzi jednoznacznie wyznaczona ścieżka, tj. taki ciąg $k \geq 1$ węzłów X_1, X_2, \dots, X_k , że X_1 jest korzeniem drzewa, $X_k = X$ i X_j jest rodzicem X_{j+1} dla $1 \leq j < k$. (Ten dowód będzie miał postać typową dla prawie wszystkich dowodów własności drzew). *Wskazówka:* Skorzystaj z indukcji względem liczby węzłów drzewa.
4. [01] Prawda czy fałsz: jeśli w tradycyjnym schemacie drzewa (korzeń na górze) węzeł X leży na poziomie o numerze *wyzszym* niż węzeł Y , to węzeł X na rysunku występuje *niżej* niż węzeł Y .
5. [02] Jeśli węzeł A ma troje rodzeństwa, a B jest rodzicem A , to jaki jest stopień B ?

- ▶ 6. [21] Zdefiniuj zdanie „ X jest m -tym kuzynem Y , w n -tym pokoleniu” jako związek między węzłami X i Y drzewa, przez analogię do drzewa rodowego, jeśli $m > 0$ i $n \geq 0$. (Sprawdź w słowniku znaczenie tych terminów w powiązaniu z drzewem rodowym).
- 7. [23] Rozszerz definicję z poprzedniego ćwiczenia na wszystkie $m \geq -1$ i wszystkie liczby całkowite $n \geq -(m+1)$ w taki sposób, że dla dowolnych dwóch węzłów X i Y istnieją jednoznacznie wyznaczone m i n , takie że X jest m -tym kuzynem Y , w n -tym pokoleniu.
- ▶ 8. [03] Które drzewa binarne nie są drzewami?
- 9. [00] Który węzeł (B czy A) jest korzeniem w drzewach binarnych (1)?
- 10. [M20] O zbiorach z dowolnej kolekcji nie zawierającej zbiorów pustych mówimy, że są *zagnieżdżone*, jeżeli dla dowolnej pary zbiorów X, Y z tej kolekcji X i Y są rozłączne lub $X \subseteq Y$, lub $X \supseteq Y$. (Innymi słowy, $X \cap Y$ równa się X, Y lub \emptyset). Rysunek 20(a) sugeruje, że każde drzewo odpowiada kolekcji zbiorów zagnieżdżonych; w drugą stronę: czy każda kolekcja zbiorów zagnieżdżonych odpowiada drzewu?
- ▶ 11. [HM32] Rozszerz definicję drzew na drzewa nieskończoności za pomocą kolekcji zbiorów zagnieżdżonych (porównaj ćwiczenie 10). Czy pojęcia poziomu, stopnia, rodzica i dziecka można zdefiniować dla każdego węzła drzewa nieskończoności? Podaj przykłady zagnieżdżonych zbiorów liczb rzeczywistych odpowiadających drzewu, w którym:
 - każdy węzeł ma stopień nieprzeliczalny, a poziomów jest nieskończonie wiele;
 - istnieją węzły o nieprzeliczalnym poziomie;
 - każdy węzeł ma stopień przynajmniej 2 i jest nieprzeliczalnie wiele poziomów.
- 12. [M23] Przy jakich założeniach zbiór częściowo uporządkowany odpowiada nieuporządkowanemu drzewu lub lasowi? (Zbiór częściowo uporządkowany jest zdefiniowany w punkcie 2.2.3).
- 13. [10] Przypuśćmy, że węzeł X jest oznaczony w systemie dziesiętnym Dewey'a numerem $a_1.a_2.\dots.a_k$; jakie są numery Dewey'a węzłów na ścieżce od X do korzenia (zobacz ćwiczenie 3)?
- 14. [M22] Niech S będzie niepustym zbiorem elementów postaci „ $1.a_1.\dots.a_k$ ”, gdzie $k \geq 0$ i a_1, \dots, a_k są dodatnimi liczbami całkowitymi. Pokaż, że S wyznacza drzewo, gdy jest skończony i spełnia następujący warunek: „Jeśli $\alpha.m$ należy do zbioru, to do zbioru należy także $\alpha.(m-1)$, gdy $m > 1$ lub α gdy $m = 1$ ”. (Ten warunek jest w oczywisty sposób spełniony przez dziesiętną notację Dewey'a; mamy zatem kolejną metodę przedstawienia struktury drzewiastej).
- ▶ 15. [20] Wymyśl notację do opisu węzłów drzew binarnych, analogiczną do notacji dziesiętnej Dewey'a.
- 16. [20] Narysuj drzewa analogiczne do tych na rysunku 21 odpowiadające wyrażeniom arytmetycznym (a) $2(a - b/c)$; (b) $a + b + 5c$.
- 17. [01] Który węzeł jest rodzicem ($F[1, 2, 2]$), gdy F oznacza strukturę z rysunku 19 traktowaną jako las?
- 18. [08] Czym jest $L[5, 1, 1]$ na Liście (3)? Czym jest $L[3, 1]$?
- 19. [15] Narysuj schemat Listy analogiczny do (7) dla Listy $L = (a, (L))$. Czym na tej liście jest $L[2]$? Czym jest $L[2, 1, 1]$?
- ▶ 20. [M21] Zdefiniuj 0-2-drzewo jako drzewo, w którym każdy węzeł ma dokładnie zero lub dwoje dzieci. (Formalnie, 0-2-drzewo składa się z pojedynczego węzła nazywanego

korzeniem oraz 0 lub 2 rozłącznych 0-2-drzew). Pokaż, że każde 0-2-drzewo ma nieparzystą liczbę węzłów. Podaj wzajemnie jednoznaczne odwzorowanie między drzewami binarnymi o n węzłach a (uporządkowanymi) 0-2-drzewami o $2n + 1$ węzłach.

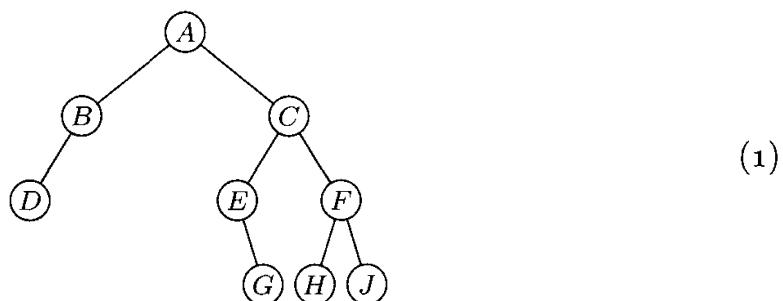
21. [M22] Jeśli drzewo ma n_1 węzłów stopnia 1, n_2 węzłów stopnia 2, … i n_m węzłów stopnia m , to ile ma liści?

► **22.** [21] Standardowe europejskie formaty arkusza papieru A0, A1, A2, …, An, … są prostokątami o stosunku boków jak $\sqrt{2}$ do 1, których powierzchnia równa się 2^{-n} metrów kwadratowych. Jeśli zatem przetniemy w połowie kartkę papieru An, to otrzymamy dwie kartki papieru A($n+1$). Opierając się na tym pomyśle, zaprojektuj graficzną reprezentację drzewa binarnego. Przedstaw swoją propozycję, rysując reprezentację drzewa 2.3.1-(1).

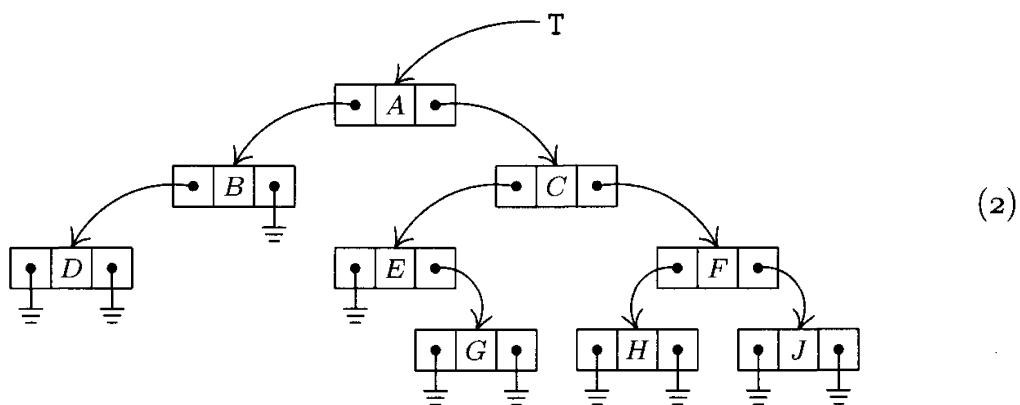
2.3.1. Przechodzenie drzew binarnych

Dobre zrozumienie własności drzew binarnych jest niezmiernie istotne z uwagi na fakt, że w pamięci komputera zwykle drzewa są zazwyczaj reprezentowane za pomocą drzew binarnych.

Zdefiniowaliśmy drzewo binarne jako skończony zbiór węzłów, który jest albo pusty, albo zawiera korzeń oraz dwa drzewa binarne. Ta definicja sugeruje naturalny sposób reprezentacji drzew binarnych w pamięci komputera. W każdym węźle przechowujemy dwa dowiązania LLINK i RLINK, a w programie posługujemy się zmienną dowiązaniową T zawierającą „wskaźnik do drzewa”. Gdy drzewo jest puste, przyjmujemy $T = \Lambda$; w przeciwnym razie T jest adresem węzła będącego korzeniem drzewa, a LLINK(T), RLINK(T) wskazują odpowiednio lewe i prawe poddrzewo korzenia. Te reguły rekurencyjnie definiują reprezentację drzewa binarnego w pamięci. Na przykład:



reprezentujemy jako



To między innymi z powodu tak prostej i naturalnej reprezentacji drzewa binarne są niezmiernie ważnymi strukturami danych. W punkcie 2.3.2 przekonamy się, że zwykłe drzewa można w wygodny sposób reprezentować za pomocą drzew binarnych. Poza tym w wielu praktycznych problemach pojawiają się po prostu drzewa binarne, co świadczy o tym, że są one same w sobie strukturami interesującymi.

Jest wiele algorytmów operujących na strukturach drzewiastych, a operacją pojawiającą się w tych algorytmach niezmiernie często jest *przechodzenie* drzewa. Przechodzenie polega na systematycznym przeglądaniu węzłów w taki sposób, by każdy odwiedzony został dokładnie jeden raz. Konkretne przejście drzewa wyznacza porządek liniowy węzłów, a wiele algorytmów daje się niezmiernie prosto wyrazić, jeśli mówimy o węzłach „poprzednich” lub „następnych” w pewnym liniowym porządku.

Istnieją trzy podstawowe sposoby przechodzenia drzewa binarnego: w porządku *preorder*, *inorder* lub *postorder**. Te trzy porządki definiujemy rekurencyjnie. Gdy drzewo binarne jest puste, nie trzeba nic robić, żeby je „przejść”; w przeciwnym razie obejście drzewa polega na wykonaniu trzech kroków:

Przechodzenie preorder
Odwiedź korzeń
Przejdź lewe poddrzewo
Przejdź prawe poddrzewo

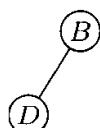
Przechodzenie inorder
Przejdź lewe poddrzewo
Odwiedź korzeń
Przejdź prawe poddrzewo

Przechodzenie postorder
Przejdź lewe poddrzewo
Przejdź prawe poddrzewo
Odwiedź korzeń

Stosując tę definicję do drzewa binarnego (1) i (2), stwierdzamy, że węzły w porządku preorder to

$$A \ B \ D \ C \ E \ G \ F \ H \ J. \quad (3)$$

(Najpierw korzeń *A*, potem lewe poddrzewo w porządku preorder



a potem prawe poddrzewo w porządku preorder). W porządku inorder odwiedzamy korzeń między wizytami w lewym a wizytami w prawym poddrzewie, dokładnie tak jakby węzły zostały „zrzutowane” w dół na linię poziomą. Otrzymujemy ciąg

$$D \ B \ A \ E \ G \ C \ H \ F \ J. \quad (4)$$

* W języku polskim spotyka się także nazwy „porządek przedrostkowy”, „porządek wrostkowy” i „porządek przyrostkowy” (przyp. tłum.).

Wreszcie węzły drzewa w porządku postorder to

$$D \ B \ G \ E \ H \ J \ F \ C \ A. \quad (5)$$

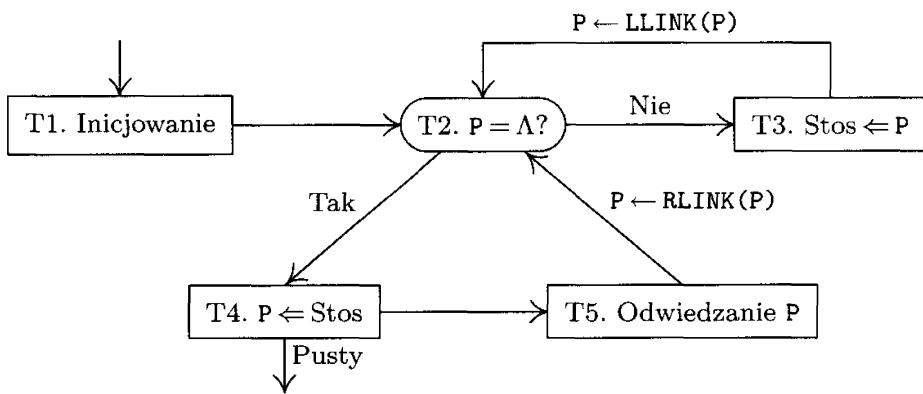
Przekonamy się, że te trzy sposoby ustawienia w ciąg elementów drzewa binarnego są niezmiernie istotne, ponieważ występują w większości algorytmów drzewowych. Nazwy *preorder*, *inorder* i *postorder* pochodzą od względnego położenia korzenia w ciągu odwiedzanych węzłów (pre – przed poddrzewami, in – między poddrzewami, post – po poddrzewach). W wielu przypadkach nie ma znaczenia rozróżnienie między lewym a prawym poddrzewem. Termin *porządek symetryczny* jest wówczas synonimem porządku inorder. Porządek inorder, w którym korzeń odwiedzamy między wizytami w lewym i prawym poddrzewie, jest symetryczny w tym sensie, że jeżeli weźmiemy odbicie symetryczne drzewa binarnego względem pionowej osi, to ciąg węzłów takiego odbicia w porządku inorder będzie dokładnie ciągiem węzłów pierwotnego drzewa w porządku inorder wypisany od tyłu.

Jeśli mamy zamiar korzystać w programach z definicji rekurencyjnej, jak na przykład definicje trzech porządków przechodzenia drzewa, to musimy je do tego przystosować. Ogólne metody radzenia sobie z takimi problemami omawiamy w rozdziale 8; zazwyczaj korzystamy z dodatkowego stosu, jak w następującym algorytmie:

Algorytm T (*Przechodzenie drzewa binarnego w porządku inorder*). Niech T będzie dowiązaniem do drzewa binarnego o reprezentacji takiej jak (2). Algorytm odwiedza wszystkie węzły drzewa binarnego w porządku inorder, korzystając z pomocniczego stosu A.

- T1.** [Inicjowanie] Utwórz pusty stos A i przypisz wartość dowiązania do zmiennej dowiązaniowej: $P \leftarrow T$.
- T2.** [$P = \Lambda?$] Jeśli $P = \Lambda$, to przejdź do kroku T4.
- T3.** [Stos $\Leftarrow P$] (P wskazuje na niepuste drzewo binarne, które mamy przejść). Przyjmij $A \Leftarrow P$, tj. włóż wartość P na stos A. (Zobacz punkt 2.2.1). Następnie przyjmij $P \leftarrow \text{LLINK}(P)$ i wróć do kroku T2.
- T4.** [$P \Leftarrow \text{Stos}$] Jeśli stos A jest pusty, to zakończ wykonanie algorytmu; w przeciwnym razie $P \Leftarrow A$.
- T5.** [Odwiedzenie P] Odwiedź $\text{NODE}(P)$. Następnie przyjmij $P \leftarrow \text{RLINK}(P)$ i wróć do kroku T2. ■

Słowo „odwiedź” w ostatnim kroku algorytmu oznacza wykonanie pewnej operacji na węźle drzewa, które przechodzimy. Algorytm T działa jak współprogram ze względu na tę właśnie operację: program główny aktywuje współprogram przechodzenia drzewa, gdy chce przesunąć P do następnego (w porządku inorder) węzła drzewa. Współprogram wywołuje program główny tylko w jednym miejscu, więc na dobrą sprawę nie różni się szczegółowo od zwykłego podprogramu (zobacz punkt 1.4.2). W algorytmie T zakładamy, że akcje wykonywane poza współprogramem nie usuwają z drzewa węzła $\text{NODE}(P)$ ani jego poprzedników.



Rys. 23. Algorytm T przechodzenia drzewa w porządku inorder.

Czytelnik powinien spróbować w tym miejscu pobawić się algorytmem T, korzystając na przykład ze schematu (2), by zrozumieć mechanizm jego działania. Po dojściu do kroku T3 chcemy przejść drzewo binarne, na którego korzeń wskazuje P. Pomysł polega na odłożeniu P na stos i przejściu w pierwszej kolejności lewego poddrzewa. W ten sposób dostajemy się do kroku T4, gdzie starą wartość P zdejmujemy ze stosu. Po odwiedzeniu korzenia $\text{NODE}(P)$ w kroku T5 pozostaje przejść prawe poddrzewo.

Algorytm T ma strukturę typową dla wielu algorytmów drzewowych, warto więc przyjrzeć się ścisłemu dowodowi stwierdzeń z poprzedniego akapitu. Spróbujmy, korzystając z indukcji względem n , udowodnić, że algorytm T przechodzi drzewo binarne o n węzłach w porządku inorder. Naszym celem jest udowodnienie nieco silniejszego stwierdzenia:

Jeśli w kroku T2 P wskazuje na drzewo binarne o n węzłach, a stos A ma postać $A[1] \dots A[m]$ dla pewnego $m \geq 0$, to w krokach T2–T5 algorytm przejdzie drzewo P i znajdzie się w kroku T4 ze stosem o początkowej zawartości $A[1] \dots A[m]$.

Stwierdzenie jest oczywiście prawdziwe dla $n = 0$, co wynika z postaci kroku T2. Jeśli $n > 0$, to niech P_0 będzie wartością P w chwili rozpoczęcia wykonania kroku T2. Algorytm przejdzie do kroku T3, ponieważ $P_0 \neq \Lambda$. To oznacza, że zawartość stosu zmieni się na $A[1] \dots A[m]P_0$, a do P zostanie przypisana wartość $\text{LLINK}(P_0)$. Lewe poddrzewo ma mniej niż n węzłów, zatem na mocy indukcji algorytm przejdzie lewe poddrzewo w porządku inorder, po czym przejdzie do kroku T4 ze stosem $A[1] \dots A[m]P_0$. W kroku T4 stos wróci do poprzedniego stanu $A[1] \dots A[m]$, ponadto $P \leftarrow P_0$. W kroku T5 algorytm odwiedza $\text{NODE}(P_0)$ i przypisuje $P \leftarrow \text{RLINK}(P_0)$. Prawe drzewo ma mniej niż n węzłów, zatem na mocy indukcji algorytm obejdzie prawe poddrzewo w porządku inorder i nastąpi powrót do kroku T4. Na mocy definicji porządku inorder wnioskujemy, że algorytm przeszedł drzewo w tym właśnie porządku. To kończy dowód.

Niemalże identyczny algorytm przechodzi drzewo binarne w porządku preorder (zobacz ćwiczenie 12). Nieco trudniej przejść drzewo w porządku postorder

(zobacz ćwiczenie 13) i z tej przyczyny porządek postorder nie ma takiego znaczenia, jak pozostałe dwa porządkie przechodzenia drzew.

Przydadzą się nam oznaczenia następców i poprzedników węzłów drzewa w różnych porządkach. Jeśli P wskazuje na węzeł drzewa binarnego, to niech

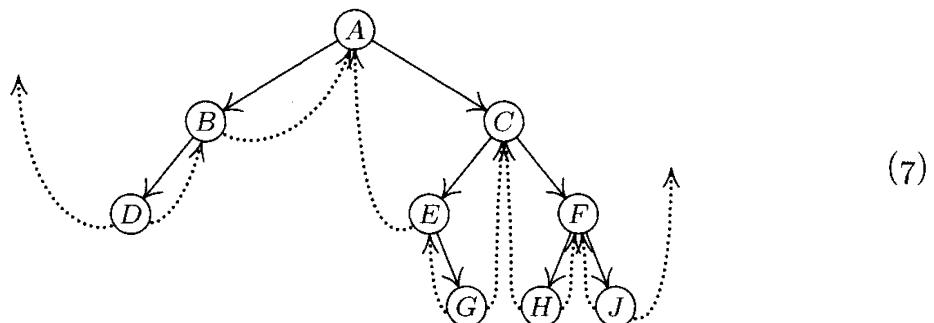
- P^* = adres następcika $\text{NODE}(P)$ w porządku preorder;
 - $P\$$ = adres następcika $\text{NODE}(P)$ w porządku inorder;
 - $P\#$ = adres następcika $\text{NODE}(P)$ w porządku postorder;
 - $*P$ = adres poprzednika $\text{NODE}(P)$ w porządku preorder;
 - $\$P$ = adres poprzednika $\text{NODE}(P)$ w porządku inorder;
 - $\#P$ = adres poprzednika $\text{NODE}(P)$ w porządku postorder.
- (6)

Gdy poprzednik lub następnik $\text{NODE}(P)$ nie istnieje, używamy zazwyczaj wartości $\text{LOC}(T)$, gdzie T jest zewnętrznym dowiązaniem do korzenia drzewa. Mamy $*(P^*) = (*P)* = P$, $\$(P\$) = (\$P)\$ = P$ i $\#(P\#) = (\#P)\# = P$. Rozważmy następujący przykład: niech $\text{INFO}(P)$ będzie literą w węźle $\text{NODE}(P)$ drzewa (2); wtedy jeśli P wskazuje na korzeń, to $\text{INFO}(P) = A$, $\text{INFO}(P^*) = B$, $\text{INFO}(P\$) = E$, $\text{INFO}(\$P) = B$, $\text{INFO}(\#P) = C$ i $P\# = *P = \text{LOC}(T)$.

W tym miejscu Czytelnik może odnieść wrażenie, że intuicyjny opis znaczenia symboli P^* , $P\$$ niesie ze sobą pewne niebezpieczeństwo. W dalszym tekście pojęcia będą stopniowo wyjaśniane, pomocne może okazać się także ćwiczenie 16. Symbol „ $\$$ ” w „ $P\$$ ” ma przypominać literę S jako skrót od „porządek symetryczny”.

Istnieje inna metoda reprezentowania drzewa binarnego, mająca się mniej więcej tak do metody zaprezentowanej powyżej, jak listy cykliczne do zwykłych list jednokierunkowych. Zauważmy, że na schemacie (2) pustych wskaźników jest więcej niż wszystkich innych. W istocie ta sytuacja występuje w każdym drzewie binarnym o takiej reprezentacji. Nie ma sensu marnować tyle pamięci na puste dowiązania. Można na przykład w każdym węźle drzewa przechowywać dwa „znaczniki”, które w dwóch bitach będą zawierały informacje o tym, które dowiązania spośród LLINK i RLINK są puste. Stosując takie rozwiązanie, można zaoszczędzić trochę pamięci.

Pomyśłową metodę wykorzystania pamięci zaoszczędzonej w ten sposób zaproponowali A. J. Perlis i C. Thornton, którzy wpadli na pomysł tzw. *drzewa sfastrygowanego*. W tej metodzie wskaźniki puste są zastąpione przez „fastrygi” wiodące do innych części drzewa i ułatwiające jego obchodzenie. Sfastrygowane drzewo (2) wygląda tak:

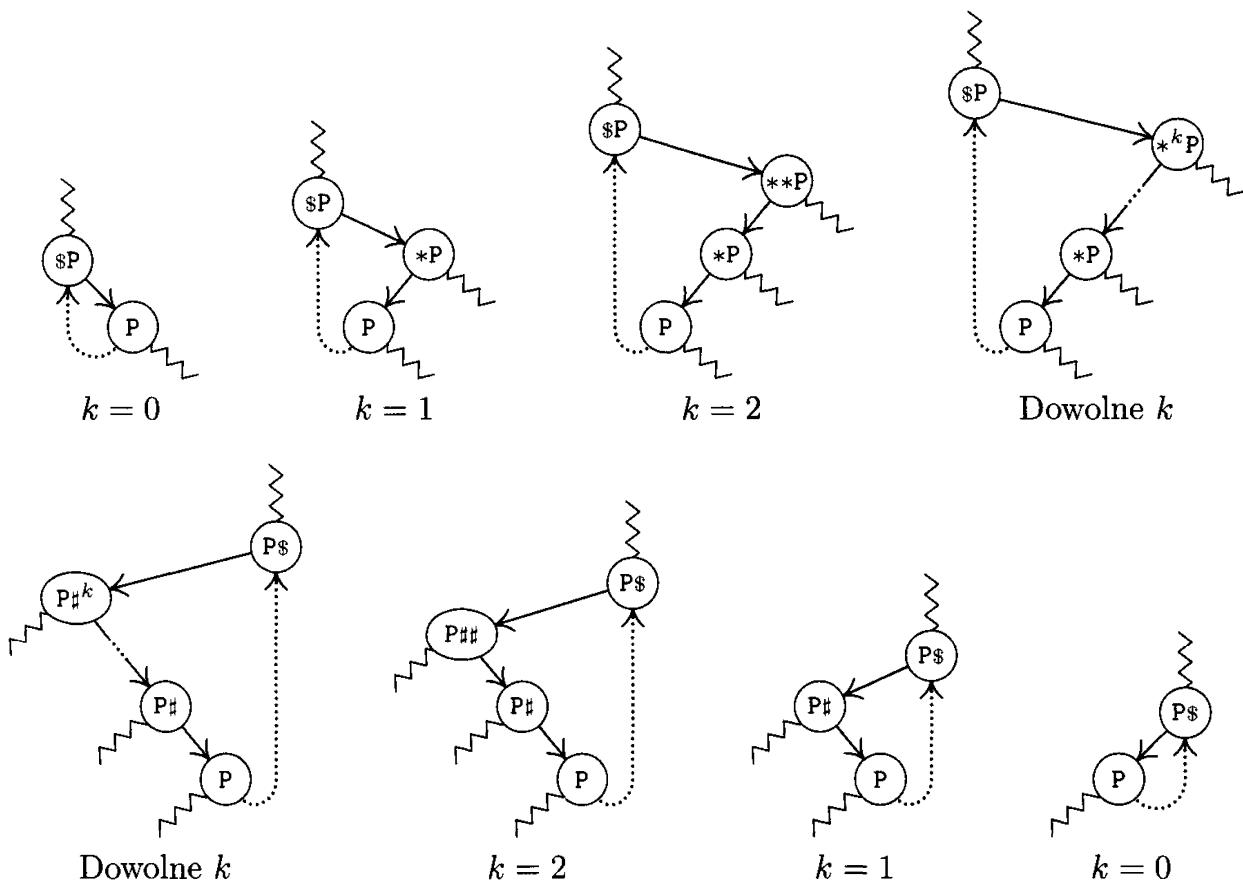


Linie kropkowane reprezentują „fastrygi”, które zawsze prowadzą do wyżej położonych węzłów drzewa. Każdy węzeł zawiera teraz dwa dowiązania: niektóre węzły, jak C , mają dwa zwykłe dowiązania do lewego i prawego poddrzewa. Inne węzły, jak H , mają dwa dowiązania „fastrygowe”, a jeszcze inne – po jednym dowiązaniu każdego rodzaju. O specjalnych dowiązaniach, prowadzących ze skrajnych elementów D i J powiemy później.

W reprezentacji drzewa w pamięci należy dysponować jakimś mechanizmem odróżniania linii ciągłych od kropkowanych; tu też można skorzystać z dwóch jednobitowych pól w każdym węźle (będziemy je nazywać LTAG i RTAG). Reprezentację „sfastrygowaną” możemy w następujący sposób zdefiniować formalnie:

Reprezentacja „niesfastrygowana”	Reprezentacja „sfastrygowana”
$\text{LLINK}(P) = \Lambda$	$\text{LTAG}(P) = 1, \text{LLINK}(P) = \P
$\text{LLINK}(P) = Q \neq \Lambda$	$\text{LTAG}(P) = 0, \text{LLINK}(P) = Q$
$\text{RLINK}(P) = \Lambda$	$\text{RTAG}(P) = 1, \text{RLINK}(P) = P\$$
$\text{RLINK}(P) = Q \neq \Lambda$	$\text{RTAG}(P) = 0, \text{RLINK}(P) = Q$

Zgodnie z powyższą definicją każde dowiązanie „fastrygowe” wskazuje na poprzednik lub następnik danego węzła w porządku symetrycznym (inorder). Na rysunku 24 są pokazane kierunki dowiązań „fastrygowych” w drzewie binarnym.



Rys. 24. Kierunki prawych i lewych dowiązań „fastrygowych” w drzewie binarnym. Linie falowane reprezentują dowiązania (również dowiązania „fastrygowe”) do innych części drzewa.

W niektórych algorytmach możemy zagwarantować, że korzeń każdego poddrzewa będzie zawsze przechowywany w komórce pamięci o adresie mniejszym niż adresy pozostałych węzłów poddrzewa. Wówczas $LTAG(P)$ będzie miało wartość 1 wtedy i tylko wtedy, gdy $LLINK(P) < P$, zatem informacja w $LTAG$ jest nadmiarowa. Na tej samej zasadzie zbędny jest wówczas bit $RTAG$.

Przewaga drzew „sfastrygowanych” nad zwykłymi drzewami binarnymi polega na tym, że algorytm ich przehodzenia jest prostszy. Poniższy algorytm dla danego P oblicza $P\$$:

Algorytm S (*Przejście do następnika w porządku symetrycznym (inorder) w sfastrygowanym drzewie binarnym*). Jeśli P wskazuje na węzeł drzewa binarnego, to wykonanie algorytmu powoduje przypisanie $Q \leftarrow P\$$.

S1. [Czy $RLINK(P)$ jest „fastrygą”?] Przyjmij $Q \leftarrow RLINK(P)$. Jeśli $RTAG(P) = 1$, zakończ wykonanie algorytmu.

S2. [Przeszukiwanie w lewo] Jeśli $LTAG(Q) = 0$, to $Q \leftarrow LLINK(Q)$ i powtórz ten krok. W przeciwnym razie zakończ wykonanie algorytmu. ■

Zauważmy, że w odróżnieniu od realizującego to samo zadanie algorytmu T, nie potrzebujemy stosu. W istocie wykorzystując zwykłą reprezentację (2), nie można w wydajny sposób wyznaczyć $P\$$ wyłącznie na podstawie wskaźnika do pewnego węzła drzewa. W reprezentacji „niesfastrygowanej” nie ma wskaźników wiodących w górę drzewa, nie ma zatem sposobu dostania się do węzła położonego „powyżej”. No, chyba że dysponujemy zapisem historii, na podstawie której możemy odtworzyć drogę dotarcia do węzła. Algorytm T odtwarza drogę, gdy nie ma dowiązań „fastrygowych”.

Twierdzimy, że algorytm S jest „wydajny”, chociaż nie jest to takie oczywiste – krok S2 może się wykonywać wiele razy. Czy z uwagi na pętlę w kroku S2 nie lepiej i szybciej byłoby jednak skorzystać ze stosu, jak w algorytmie T? By odpowiedzieć na to pytanie, zbadamy średnią liczbę wykonania kroku S2, gdy P wskazuje na losowo wybrany węzeł drzewa. Albo, co jest równoważne, zbadamy liczbę wykonania kroku S2, gdy uruchamiany wielokrotnie algorytm S obchodzi całe drzewo.

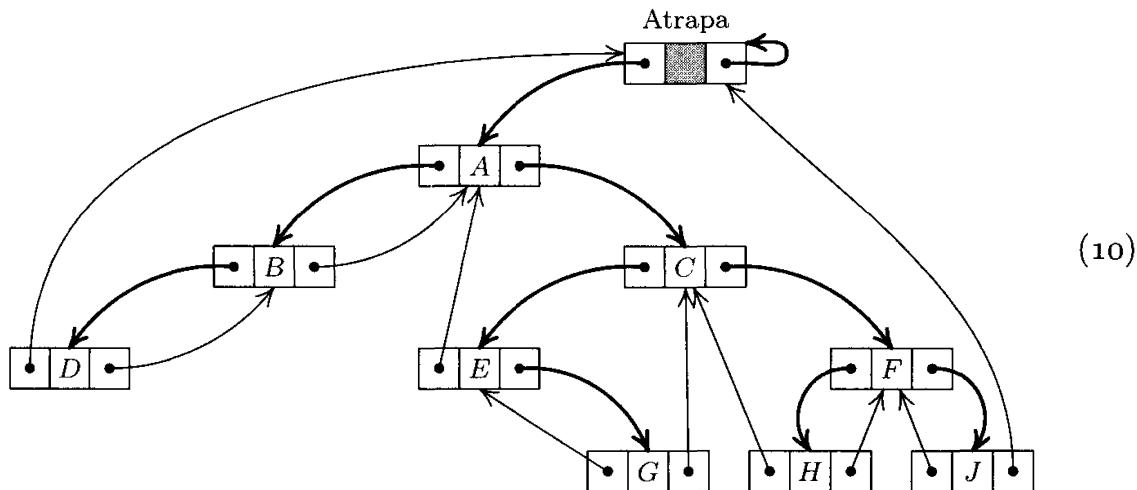
Równolegle z tą analizą zajmiemy się programami realizującymi algorytmy S i T. Jak zwykle powinniśmy zwrócić uwagę na to, czy algorytmy działają poprawnie dla pustych drzew; ponadto gdy T jest wskaźnikiem do drzewa, to chcielibyśmy, by $LOC(T)*$ i $LOC(T)\$$ były *pierwszymi* węzłami odpowiednio w porządku inorder i w porządku symetrycznym. Dla drzew „sfastrygowanych” wszystko się ładnie ułoży, jeżeli $NODE(LOC(T))$ będzie atrapą węzła drzewa, gdzie

$$\begin{aligned} LLINK(HEAD) &= T, & LTG(HEAD) &= 0, \\ RLINK(HEAD) &= HEAD, & RTAG(HEAD) &= 0. \end{aligned} \quad (8)$$

($HEAD$ oznacza $LOC(T)$, tj. adres atrapy). Puste drzewo „sfastrygowane” będzie spełniać warunki

$$LLINK(HEAD) = HEAD, \quad LTG(HEAD) = 1. \quad (9)$$

Nowe elementy wstawiamy do *lewego* poddrzewa atrapy. (Te warunki wynikają głównie z algorytmu obliczającego P^* , omówionego w ćwiczeniu 17). W świetle powyższych konwencji reprezentacją drzewa (1) z fastrygą jest



Uporawszy się z ustaleniami początkowymi, możemy zabrać się za wersje algorytmów S i T na komputer **MIX**. W poniższych programach zakładamy, że węzły drzew binarnych składają się z dwóch słów postaci

LTAG	LLINK	INFO1
RTAG	RLINK	INFO2

W drzewie „niesfastrygowanym” LTAG i RTAG zawsze będą ze znakiem „+”, a dowiązania do pustych poddrzew będą oznaczane przez wartości zerowe. W drzewie sfastrygowanym znak „+” będzie reprezentować znacznik o wartości 0, a „–” znacznik o wartości 1. Skrótkami LLINKT i RLINKT będziemy oznaczać odpowiednio połączone wycinki LTAG-LLINK i RTAG-RLINK.

 Dwa znaczniki zajmują niewykorzystane bity znaku. Na komputerze **MMIX** będziemy mogli korzystać z najmniej znaczących bitów znaczników, ponieważ wartości wskaźnikowe będą zasadniczo parzyste, a również dlatego, że **MMIX** umożliwia w prosty sposób ignorowanie mniej znaczących bitów przy adresowaniu.

Następujące dwa programy przechodzą drzewo binarne w porządku symetrycznym (czyli inorder), wykonując dla każdego węzła skok do lokacji VISIT z dowiązaniem do bieżącego węzła w rejestrze I5.

Program T. W tej realizacji algorytmu T stos jest przechowywany w lokacjach $A + 1, A + 2, \dots, A + MAX$; rI6 jest wskaźnikiem stosu, a rI5 $\equiv P$. Gdy stos zrobi się zbyt duży, wystąpi PRZEPEŁNIENIE. Kolejność kroków algorytmu została nieco zmodyfikowana, tak by nie trzeba było sprawdzać, czy stos jest pusty przy bezpośrednim przejściu od T3 do T2 i dalej do T4.

```
01 LLINK EQU 1:2
02 RLINK EQU 1:2
```

03	T1	LD5 HEAD(LLINK)	1	<u>T1. Inicjowanie.</u> $P \leftarrow T$.
04	T2A	J5Z DONE	1	Zatrzymaj się, jeśli $P = \Lambda$.
05		ENT6 0	1	
06	T3	DEC6 MAX	n	<u>T3. Stos $\leftarrow P$.</u>
07		J6NN OVERFLOW	n	Maksymalny rozmiar stosu?
08		INC6 MAX+1	n	Jeśli nie, zwiększ wskaźnik stosu.
09		ST5 A,6	n	Włóż P na stos.
10		LD5 0,5(LLINK)	n	$P \leftarrow LLINK(P)$.
11	T2B	J5NZ T3	n	Przejdź do T3, jeśli $P \neq \Lambda$.
12	T4	LD5 A,6	n	<u>T4. $P \leftarrow Stos$.</u>
13		DEC6 1	n	Zmniejsz wskaźnik stosu.
14	T5	JMP VISIT	n	<u>T5. Odwiedzenie P.</u>
15		LD5 1,5(RLINK)	n	$P \leftarrow RLINK(P)$.
16	T2C	J5NZ T3	n	<u>T2. Czy $P = \Lambda$?</u>
17		J6NZ T4	a	Sprawdź, czy stos jest pusty.
18	DONE	...		■

Program S. Do algorytmu S dodaliśmy inicjowanie oraz kod powodujący zakończenie wykonania, by program S funkcjonalnie odpowiadał programowi T.

01	LLINKT	EQU 0:2		
02	RLINKT	EQU 0:2		
03	S0	ENT5 HEAD	1	<u>S0. Inicjowanie.</u> $P \leftarrow HEAD$.
04		JMP 2F	1	
05	S3	JMP VISIT	n	<u>S3. Odwiedzenie P.</u>
06	S1	LD5N 1,5(RLINKT)	n	<u>S1. Czy $RLINK(P)$ jest fastryga?</u>
07		J5NN 1F	n	Skocz, jeśli $RTAG(P) = 1$.
08		ENN6 0,5	n - a	W przeciwnym razie $Q \leftarrow RLINK(P)$.
09	S2	ENT5 0,6	n	<u>S2. Przeszukiwanie w lewo.</u> $P \leftarrow Q$.
10	2H	LD6 0,5(LLINKT)	n + 1	$Q \leftarrow LLINKT(P)$.
11		J6P S2	n + 1	Jeśli $LTAG(P) = 0$, powtóż.
12	1H	ENT6 -HEAD,5	n + 1	
13		J6NZ S3	n + 1	Odwiedź, chyba że $P = HEAD$. ■

Wyniki analizy czasu wykonania zamieściliśmy w kodzie. Te wartości łatwo wyprowadzić za pomocą prawa Kirchhoffa oraz następujących obserwacji:

- i) w programie T tyle samo razy wykonujemy operację włożenia elementu na stos, co operację zdjęcia elementu ze stosu;
- ii) w programie T wartości dowiązań LLINK i RLINK każdego węzła odczytujemy dokładnie raz;
- iii) liczba „wizyt” jest liczbą węzłów drzewa.

Z analizy wynika, że program T potrzebuje $15n + a + 4$ jednostek czasu, a program S $11n - a + 7$ jednostek, gdzie n jest liczbą węzłów drzewa, a a jest liczbą pustych prawych dowiązań (dokładniej liczbą węzłów o pustym prawym poddrzewie). Wielkość a może przyjmować wartości tak małe jak 1 (jeśli $n \neq 0$) oraz tak duże jak n . Jeżeli drzewo jest symetryczne, to średnia wartość a wynosi $(n + 1)/2$, co wynika z faktu udowodnionego w ćwiczeniu 14.

Z powyższej analizy płyną następujące wnioski:

- i) Krok S2 jest średnio wykonywany tylko raz przy każdym wykonaniu algorytmu, jeśli P wskazuje na losowy węzeł drzewa.
- ii) Przejście drzewa jest nieco szybsze w przypadku drzew sfastrygowanych, ponieważ nie trzeba obsługiwać stosu.
- iii) Algorytm T potrzebuje więcej pamięci niż algorytm S, ponieważ musimy posługiwać się pomocniczym stosem. W programie T stos przechowujemy w kolejnych lokacjach pamięci, musimy zatem przyjąć jakieś górne ograniczenie na wielkość stosu. Przekroczenie ograniczenia oznacza poważne kłopoty, zatem to ograniczenie powinno być dosyć duże (zobacz ćwiczenie 10). Z tego powodu program T potrzebuje zdecydowanie więcej pamięci niż program S. Nie należy do rzadkości sytuacja, w której złożony program niezależnie i równocześnie przechodzi kilka drzew; korzystając z programu T potrzebujemy w takim przypadku oddzielnego stosu dla każdego drzewa. To sugeruje wykorzystanie stosu z dowiązaniami (zobacz ćwiczenia 20). Czas wykonania zwiększa się wówczas do $30n + a + 4$ jednostek, tj. mniej więcej dwa razy tyle co poprzednio, chociaż nakłady na przechodzenie drzewa mogą nie być bardzo istotne w zestawieniu z czasem wykonania innych fragmentów programu. Jeszcze jedną możliwością jest przechowywanie dowiązań elementów stosu w samym drzewie, za pomocą pewnej sprytnej sztuczki omówionej w ćwiczeniu 21.
- iv) Algorytm S jest oczywiście bardziej ogólny od algorytmu T, ponieważ pozwala przejść od P do $P\$$ nawet wtedy, gdy nie przechodzimy całego drzewa binarnego.

Sfastrygowane drzewo binarne jest zatem wygodniejsze do przechodzenia. Za tę zaletę płacimy nieco większym czasem wstawiania i usuwania węzła. Poza tym, korzystając z drzew niesfastrygowanych, można czasami zaoszczędzić na pamięci, jeżeli zdecydujemy się „nałożyć” na siebie jednakowe poddrzewa. W przypadku drzew sfastrygowanych taką sztuczką posłużyć się nie można. Musimy ściśle trzymać się rozłącznej struktury drzewiastej.

Korzystając z dowiązań fastrygowych, możemy z wydajnością równą wydajności algorytmu S obliczać $P*$, $\$P$ i $\sharp P$. Funkcje $*P$ i $\sharp P$ są nieco trudniejsze do obliczenia, podobnie jak w przypadku reprezentacji niesfastrygowanej. Czytelnika zachęcamy do rozwiązania ćwiczenia 17.

Pozytek płynący z zastosowania fastrygi byłby wątpliwy, gdyby trudno było utrzymywać poprawne dowiązania fastrygowe. W tym jednak cała rzecz, że wstawianie węzłów do drzew sfastrygowanych nie jest dużo trudniejsze niż do zwykłych drzew binarnych. Oto algorytm:

Algorytm I (Wstawianie węzła do sfastrygowanego drzewa binarnego). Algorytm wstawia pojedynczy węzeł $NODE(Q)$ w miejsce prawego poddrzewa $NODE(P)$, jeśli jest ono puste (tj. gdy $RTAG(P) = 1$). W przeciwnym razie algorytm wstawia $NODE(Q)$ między $NODE(P)$ i $NODE(RLINK(P))$, robiąc z $NODE(RLINK(P))$ prawe dziecko węzła $NODE(Q)$. Zakładamy, że drzewo binarne, do którego wsta-

wiamy element, jest sfastrygowane jak (10); inny wariant podajemy w ćwiczeniu 23.

- I1.** [Modyfikowanie znaczników i dowiązań] Przyjmij $\text{RLINK}(Q) \leftarrow \text{RLINK}(P)$, $\text{RTAG}(Q) \leftarrow \text{RTAG}(P)$, $\text{RLINK}(P) \leftarrow Q$, $\text{RTAG}(P) \leftarrow 0$, $\text{LLINK}(Q) \leftarrow P$ oraz $\text{LTAG}(Q) \leftarrow 1$.
- I2.** [Czy $\text{RLINK}(P)$ było fastrygą?] Jeśli $\text{RTAG}(Q) = 0$, to $\text{LLINK}(Q\$) \leftarrow Q$. ($Q\$$ jest wyznaczone przez algorytm S, który będzie działał poprawnie, mimo że $\text{LLINK}(Q\$)$ wskazuje teraz na $\text{NODE}(P)$, a nie $\text{NODE}(Q)$. Ten krok jest potrzebny jedynie przy wstawianiu węzła w środek drzewa, tj. gdy nie wstawiamy liścia). ■

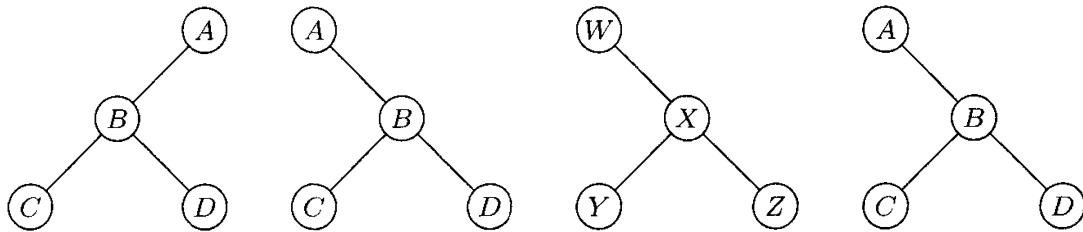
Zamieniając stronę lewą ze stroną prawą (w szczególności zastępując $Q\$$ przez $\$Q$ w kroku I2), uzyskamy algorytm wstawiający nowy węzeł na lewo od węzła $\text{NODE}(P)$.

W naszym dotychczasowym omówieniu sfastrygowanych drzew binarnych korzystaliśmy z fastryg zarówno na wskaźnikach do lewych, jak i prawych poddrzew. Pomiędzy drzewami sfastrygowanymi a niesfastrygowanymi istnieje jednak bardzo użyteczne stadium pośrednie: w *drzewach binarnych z fastrygą prawostronną* korzystamy z fastryg na wskaźnikach RLINK , podczas gdy puste lewe poddrzewa są reprezentowane przez $\text{LLINK} = \Lambda$. (Analogicznie skonstruowane są drzewa binarne z lewostronną fastrygą). Algorytm S nie korzysta w istotny sposób z fastrygi na LLINK . Jeśli zmienimy warunek „ $\text{LTAG} = 0$ ” w kroku S2 na warunek „ $\text{LLINK} \neq \Lambda$ ”, to otrzymamy algorytm przechodzenia drzewa binarnego z fastrygą prawostronną w porządku symetrycznym. Dla fastrygi prawostronnej program S działa bez żadnych zmian. Olbrzymia większość praktycznych zastosowań struktur drzewiastych wymaga przechodzenia drzewa wyłącznie „z lewej do prawej” za pomocą funkcji $\text{P\$}$ i/lub P* . W takich przypadkach nie ma więc potrzeby utrzymywania fastrygi na wskaźnikach LLINK . Opisaliśmy fastrygowanie w obu kierunkach, by uwypuklić symetrię i pokazać wszystkie możliwości, ale w praktyce fastrygowanie jednostronne występuje o wiele częściej.

Zastanowimy się teraz nad pewnymi własnościami drzew binarnych i ich związkami z przechodzeniem drzew. Mówimy, że dwa drzewa binarne T i T' są *podobne*, jeżeli mają ten sam kształt. Formalnie znaczy to, że (a) oba są puste lub (b) oba są niepuste i ich lewe i prawe drzewa są odpowiednio podobne. Inaczej: drzewa binarne są podobne, jeżeli istnieje wzajemnie jednoznaczna odpowiedniość między węzłami drzewa T i drzewa T' , która zachowuje strukturę: jeśli węzły u_1 i u_2 w T odpowiadają węzłom u'_1 i u'_2 w T' , to u_1 należy do lewego poddrzewa u_2 wtedy i tylko wtedy, gdy u'_1 należy do lewego poddrzewa u'_2 oraz ten sam warunek zachodzi dla prawych poddrzew.

O drzewach binarnych T i T' mówimy, że są *równoważne*, jeżeli są podobne, a odpowiadające sobie węzły zawierają te same informacje. Formalnie, niech $\text{info}(u)$ oznacza informację przechowywaną w węźle u . Drzewa są równoważne wtedy i tylko wtedy, gdy (a) oba są puste lub (b) oba są niepuste i $\text{info}(\text{root}(T)) = \text{info}(\text{root}(T'))$ oraz ich lewe i prawe poddrzewa są odpowiednio równoważne.

W ramach przykładu przyjrzyjmy się czterem drzewom binarnym



wśród których dwa pierwsze są niepodobne. Drugie, trzecie i czwarte są podobne, a drugie i czwarte są ponadto równoważne.

W niektórych algorytmach musimy stwierdzić, czy pewne drzewa binarne są podobne lub równoważne. Przydaje się wówczas poniższe twierdzenie:

Twierdzenie A. Niech węzłami drzew binarnych T i T' w porządku preorder będą odpowiednio

$$u_1, u_2, \dots, u_n \quad \text{i} \quad u'_1, u'_2, \dots, u'_{n'}$$

Niech dla każdego węzła u

$$\begin{aligned} l(u) &= 1, \text{ jeśli } u \text{ ma niepuste lewe poddrzewo,} & l(u) &= 0 \text{ w przeciwnym razie;} \\ r(u) &= 1, \text{ jeśli } u \text{ ma niepuste prawe poddrzewo,} & r(u) &= 0 \text{ w przeciwnym razie.} \end{aligned} \tag{11}$$

Przy takich założeniach T i T' są podobne wtedy i tylko wtedy, gdy $n = n'$ oraz

$$l(u_j) = l(u'_j), \quad r(u_j) = r(u'_j) \quad \text{dla } 1 \leq j \leq n. \tag{12}$$

Ponadto T i T' są równoważne wtedy i tylko wtedy, gdy dodatkowo

$$\text{info}(u_j) = \text{info}(u'_j) \quad \text{dla } 1 \leq j \leq n. \tag{13}$$

Zauważmy, że l i r są dopełnieniami bitów LTAG i RTAG w drzewie sfastrygowanym. Powyższe twierdzenie określa charakterystykę kształtu drzewa binarnego za pomocą ciągów zer i jedynek.

Dowód. Warunek równoważności drzew jest oczywisty, jeżeli udowodnimy warunek podobieństwa. Widać, że warunki $n = n'$ i (12) są niezbędne, ponieważ odpowiednie węzły drzew podobnych muszą być tak samo położone w porządku preorder. Wystarczy zatem pokazać, że warunki (12) i $n = n'$ wystarczą, by zagwarantować podobieństwo drzew T i T' . Dowód przebiega przez indukcję względem n i opiera się na następującym lematce:

Lemat P. Niech u_1, u_2, \dots, u_n będą węzłami niepustego drzewa binarnego w porządku preorder i niech $f(u) = l(u) + r(u) - 1$. Wówczas

$$f(u_1) + f(u_2) + \dots + f(u_n) = -1 \quad \text{i} \quad f(u_1) + \dots + f(u_k) \geq 0, \quad 1 \leq k < n. \tag{14}$$

Dowód. Teza jest jasna dla $n = 1$. Jeśli $n > 1$, to drzewo binarne składa się ze swojego korzenia u_1 i innych węzłów. Jeśli $f(u_1) = 0$, to lewe lub prawe poddrzewo jest puste, zatem na mocy indukcji warunek jest spełniony. Jeśli $f(u_1) = 1$, to niech lewe poddrzewo ma n_l węzłów. Na mocy indukcji

$$f(u_1) + \dots + f(u_k) > 0 \quad \text{dla } 1 \leq k \leq n_l, \quad f(u_1) + \dots + f(u_{n_l+1}) = 0, \tag{15}$$

zatem warunek (14) jest spełniony. ■

(Inne twierdzenia analogiczne do lematu P omówimy w rozdziale 10 w związku z notacją polską).

Kończymy dowód twierdzenia A. Twierdzenie jest prawdziwe dla $n = 0$. Jeśli $n > 0$, to z definicji porządku preorder wynika, że u_1 i u'_1 są odpowiednio korzeniami swoich drzew oraz że istnieją takie liczby całkowite n_l i n'_l (rozmiary lewych poddrzew), że

u_2, \dots, u_{n_l+1} i $u'_2, \dots, u'_{n'_l+1}$ są lewymi poddrzewami T i T' ;

u_{n_l+2}, \dots, u_n i $u'_{n'_l+2}, \dots, u'_n$ są prawymi poddrzewami T i T' .

Dowód indukcyjny będzie można uważać za poprawny, jeżeli pokażemy, że $n_l = n'_l$. Są trzy przypadki:

jeśli $l(u_1) = 0$, to $n_l = 0 = n'_l$;

jeśli $l(u_1) = 1$, $r(u_1) = 0$, to $n_l = n - 1 = n'_l$;

jeśli $l(u_1) = r(u_1) = 1$, to na mocy lematu P możemy znaleźć najmniejsze $k > 0$, takie że $f(u_1) + \dots + f(u_k) = 0$; mamy $n_l = k - 1 = n'_l$ (zobacz (15)). ■

Z twierdzenia A wynika, że możemy sprawdzić, czy dwa drzewa są podobne lub równoważne, po prostu przechodząc je w porządku preorder i sprawdzając wartości pól **INFO** oraz **TAG**. Pewne ciekawe warianty twierdzenia A pochodzą od A. J. Blikiego, *Bull. de l'Acad. Polonaise des Sciences, Série des Sciences Math., Astr., Phys.*, **14** (1966), 203–208, który zajmował się nieskończoną klasą możliwych porządków przechodzenia drzew, z których tylko sześć (włączając preorder) zostało nazwanych „bezadresowymi” z uwagi na ich proste własności.

Zamykamy ten rozdział elementarnym algorytmem operującym na drzewach binarnych – algorytmem kopowania drzewa.

Algorytm C (Kopowanie drzewa binarnego). Niech **HEAD** będzie adresem atrapy w drzewie binarnym T ; T jest zatem lewym poddrzewem węzła **HEAD** osiągalnym przez **LLINK(HEAD)**. Niech **NODE(U)** będzie węzłem o pustym lewym poddrzewie. Algorytm tworzy kopię T , która staje się lewym poddrzewem węzła **NODE(U)**. W szczególności jeśli **NODE(U)** jest atrapą w pustym drzewie binarnym, to algorytm zmienia drzewo puste w kopię drzewa T .

C1. [Inicjowanie] Przyjmij $P \leftarrow \text{HEAD}$, $Q \leftarrow U$. Idź do C4.

C2. [Czy jest coś na prawo?] Jeśli **NODE(P)** ma niepuste prawe poddrzewo, to $R \leftarrow \text{AVAIL}$ i doczep **NODE(R)** jako prawe poddrzewo **NODE(Q)**. (Na początku kroku C2 prawe poddrzewo **NODE(Q)** było puste).

C3. [Kopowanie INFO] Przyjmij $\text{INFO}(Q) \leftarrow \text{INFO}(P)$. (Tu **INFO** oznacza wszystkie pola i słowa węzła poza dowiązaniemi).

C4. [Czy jest coś na lewo?] Jeśli **NODE(P)** ma niepuste lewe poddrzewo, to przyjmij $R \leftarrow \text{AVAIL}$ i doczep **NODE(R)** jako lewe poddrzewo **NODE(Q)**. (Na początku kroku C4 lewe poddrzewo **NODE(Q)** było puste).

C5. [Przesuwanie dowiązań] Przyjmij $P \leftarrow P*$, $Q \leftarrow Q*$.

C6. [Sprawdzenie, czy wszystko] Jeśli $P = \text{HEAD}$ (lub równoważnie $Q = \text{RLINK}(U)$, o ile **NODE(U)** ma niepuste prawe poddrzewo), to zatrzymaj algorytm; w przeciwnym razie przejdź do kroku C2. ■

Ten prosty algorytm pokazuje typowe zastosowanie przechodzenia drzewa. Podany opis można zastosować do drzew sfastrygowanych, niesfastrygowanych i częściowo sfastrygowanych. W kroku C5 musimy wyznaczyć następniki w porządku preorder: P^* oraz Q^* . W przypadku drzew niesfastrygowanych z reguły jest do tego potrzebny pomocniczy stos. Dowód poprawności algorytmu C podajemy w ćwiczeniu 29; program na komputer MIX realizujący algorytm dla drzew binarnych z fastrygą prawostronną zamieszczamy w ćwiczeniu 2.3.2–13. Dla drzew sfastrygowanych „podczepianie” węzła w krokach C2 i C4 realizujemy za pomocą algorytmu I.

W podanych poniżej ćwiczeniach jest poruszonych kilka ciekawych zagadnień dotyczących materiału omówionego w tym punkcie.

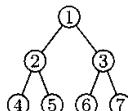
Systemy binarne lub dychotomiczne, choć rządzące się prostą zasadą, są jednymi z najbardziej sztucznych tworów, jakie kiedykolwiek wynaleziono.

— WILLIAM SWAINSON, *A Treatise on the Geography and Classification of Animals* (1835)

ĆWICZENIA

1. [01] Niech $\text{INFO}(P)$ oznacza literę zapisaną w węźle $\text{NODE}(P)$ drzewa binarnego (2). Czym jest $\text{INFO}(\text{LLINK}(\text{RLINK}(\text{RLINK}(T))))$?

2. [11] Wypisz węzły następującego drzewa binarnego w porządku (a) preorder; (b) symetrycznym; (c) postorder.



3. [20] Czy następujące zdanie jest prawdą, czy fałszem. „Liście drzewa binarnego występują w tym samym względnym porządku na liście węzłów drzewa w porządku preorder, inorder i postorder”.

► 4. [20] W tekście zdefiniowano trzy podstawowe porządki obchodzenia drzewa binarnego. A gdyby tak: (a) odwiedzić korzeń, (b) przejść prawe poddrzewo, (c) przejść lewe poddrzewo, używając tej samej rekurencyjnej reguły dla wszystkich niepustych poddrzew? Czy między tym nowym porządkiem a porządkami omówionymi w tekście jest jakiś prosty związek?

5. [22] Węzły drzewa binarnego można utożsamiać z ciągami zer i jedynek w następujący sposób, analogiczny do „notacji dziesiętnej Dewey'a” dla drzew: korzeniowi (jeśli istnieje) odpowiada ciąg „1”. Korzeniom (jeśli istnieją) lewego i prawego poddrzewa węzła, któremu odpowiada ciąg α , odpowiadają odpowiednio ciągi $\alpha 0$ i $\alpha 1$. Na przykład węzeł S w drzewie (1) miałby reprezentację „1110”. (Zobacz ćwiczenie 2.3–15).

Pokaż, że za pomocą tej notacji można wygodnie zdefiniować porządki preorder, inorder i postorder.

6. [M22] Przypuśćmy, że drzewo binarne ma n węzłów, tworzących w porządku preorder ciąg $u_1 u_2 \dots u_n$, a w porządku inorder ciąg $u_{p_1} u_{p_2} \dots u_{p_n}$. Pokaż, że permutację $p_1 p_2 \dots p_n$ można otrzymać z permutacji $1 2 \dots n$ za pomocą stosu w sposób opisany w ćwiczeniu 2.2.1–2. Pokaż też w drugą stronę, że dowolna permutacja $p_1 p_2 \dots p_n$ otrzymywana za pomocą stosu odpowiada w powyższy sposób pewnemu drzewu binarnemu.

7. [22] Pokaż, że mając dane węzły drzewa binarnego w porządku preorder i inorder, można odtworzyć strukturę drzewa. Czy to samo stwierdzenie jest prawdziwe dla pary preorder i postorder? A inorder i postorder?

8. [20] Znajdź wszystkie drzewa binarne, których węzły tworzą ten sam ciąg, gdy wypisze się je w porządkach (a) preorder i inorder; (b) preorder i postorder; (c) inorder i postorder.

9. [M20] Ile razy wykonują się kroki T1, T2, T3, T4 i T5 algorytmu T przy przehodzeniu drzewa binarnego o n węzłach?

► **10.** [20] Ile maksymalnie węzłów drzewa może się znaleźć (w tym samym momencie) na stosie podczas wykonania algorytmu T, jeżeli drzewo binarne ma n węzłów? (To jest bardzo ważne pytanie, jeżeli stos rosnąc zajmuje kolejne komórki pamięci).

11. [HM41] Przeanalizuj średnią względem n wartość maksymalnego rozmiaru stosu występującego podczas wykonania algorytmu T przy założeniu, że wszystkie drzewa o n węzłach pojawiają się z tym samym prawdopodobieństwem.

12. [22] Zaprojektuj algorytm analogiczny do algorytmu T, który przechodzi drzewo binarne w porządku *preorder*, i udowodnij, że Twój algorytm jest poprawny.

► **13.** [24] Zaprojektuj algorytm analogiczny do algorytmu T, który przechodzi drzewo binarne w porządku *postorder*.

14. [22] Pokaż, że jeżeli drzewo binarne o n węzłach ma reprezentację (2), to całkowita liczba dowiązań Λ w tej reprezentacji może być wyrażona jako prosta funkcja n ; ta wielkość nie zależy od kształtu drzewa.

15. [15] W reprezentacji drzewa z fastrygą, takiej jak (10), każdy węzeł poza atrapą ma dokładnie jedno dowiązanie prowadzące do niego z góry, konkretnie dowiązanie od jego rodzica. Niektóre węzły mają poza tym dowiązania prowadzące do nich z dołu; na przykład do węzła C prowadzą dwa dowiązania z dołu, a do węzła E tylko jedno. Czy jest jakiś prosty związek między liczbą dowiązań prowadzących do węzła a jakąś jego inną elementarną własnością? (Musimy wiedzieć, ile dowiązań prowadzi do węzła, gdy zmieniamy kształt drzewa).

► **16.** [22] Na schematach na rysunku 24 jest pokazane, gdzie w drzewie binarnym leży $\text{NODE}(Q\$)$ w odniesieniu do otoczenia węzła $\text{NODE}(Q)$: w przypadku gdy $\text{NODE}(Q)$ ma niepuste prawe poddrzewo, rozważmy $Q = \$P$, $Q\$ = P$ na schematach w górnej części rysunku; $\text{NODE}(Q\$)$ jest „skrajnie lewym” węzłem prawego poddrzewa. W przypadku gdy $\text{NODE}(Q)$ ma puste prawe poddrzewo, rozważmy $Q = P$ w dolnej części rysunku; $\text{NODE}(Q\$)$ znajdujemy, posuwając się w góre drzewa, aż do pierwszego kroku w prawo.

Podaj podobne „intuicyjne” reguły dotyczące położenia $\text{NODE}(Q*)$ w drzewie binarnym w kontekście kształtu drzewa w pobliżu węzła $\text{NODE}(Q)$.

► **17.** [22] Podaj algorytm analogiczny do algorytmu S wyznaczający $P*$ w sfastrygowanym drzewie binarnym. Załącz, że drzewo ma atrapę, jak w (8), (9) i (10).

18. [24] W wielu algorytmach korzystających z drzew musimy odwiedzić każdy węzeł nie raz, a dwa razy w tzw. *porządku podwójnym*, będącym połączeniem porządku preorder i inorder. Przechodzenie drzewa binarnego w porządku podwójnym definiujemy następująco: jeżeli drzewo binarne jest puste, nie rób nic; w przeciwnym razie

- odwiedź korzeń po raz pierwszy;
- przejdź lewe poddrzewo w porządku podwójnym;
- odwiedź korzeń po raz drugi;
- przejdź prawe poddrzewo w porządku podwójnym.

Na przykład obejście drzewa (1) w porządku podwójnym daje ciąg

$$A_1 B_1 D_1 D_2 B_2 A_2 C_1 E_1 E_2 G_1 G_2 C_2 F_1 H_1 H_2 F_2 J_1 J_2,$$

gdzie A_1 oznacza pierwsze odwiedzenie węzła A .

Załóżmy, że P wskazuje na węzeł drzewa, a $d = 1$ lub 2 . Zdefiniujmy $(P, d)^\Delta = (Q, e)$ wtedy i tylko wtedy, gdy e -ta wizyta w węźle $\text{NODE}(Q)$ następuje bezpośrednio po d -tej wizycie w węźle $\text{NODE}(P)$ w porządku podwójnym; jeśli (P, d) jest ostatnim krokiem w porządku podwójnym, to przyjmujemy $(P, d)^\Delta = (\text{HEAD}, 2)$, gdzie HEAD jest adresem atrapy. Definiujemy także $(\text{HEAD}, 1)^\Delta$ jako pierwszy krok w porządku podwójnym.

Zaprojektuj algorytm analogiczny do algorytmu T przechodzący drzewo binarne w porządku podwójnym. Zaprojektuj również algorytm analogiczny do algorytmu S obliczający $(P, d)^\Delta$. Omów związki między tymi algorytmami oraz ćwiczeniami 12 i 17.

- 19. [27] Zaprojektuj algorytm analogiczny do algorytmu S , obliczający $P \#$ w (a) drzewie binarnym z fastrygą prawostronną; (b) drzewie binarnym z pełną fastrygą. Jeśli to możliwe czas działania algorytmu dla dowolnego węzła P powinien być co najwyżej niewielką stałą.
- 20. [23] Zmodyfikuj program T w ten sposób, by elementy stosu przechowywały na liście z dowiązaniem, a nie w kolejnych komórkach pamięci.
- 21. [33] Zaprojektuj algorytm przechodzenia niesfastrygowanego drzewa binarnego w porządku inorder *nie korzystający z pomocniczego stosu*. Podczas przechodzenia drzewa można dowolnie modyfikować pola **LLINK** i **RLINK** pod warunkiem, że po obejściu drzewa (tak jak i przed) ma ono reprezentację (2). Żadnych innych bitów w węzłach drzewa nie wolno przeznaczać na pamięć pomocniczą.
- 22. [25] Napisz program w asemblerze komputera **MIX** realizujący algorytm podany w ćwiczeniu 21 i porównaj czas jego wykonania z czasem wykonania programów S i T .
- 23. [22] Zaprojektuj algorytm analogiczny do algorytmu I , realizujący wstawianie w lewe i prawe poddrzewo w drzewie binarnym z fastrygą prawostronną. Załącz, że węzły mają pola **LLINK**, **RLINK** i **RTAG**.
- 24. [M20] Czy twierdzenie A byłoby prawdziwe również wtedy, gdyby węzły drzew T i T' byłyby podane w porządku symetrycznym zamiast preorder?
- 25. [M24] Niech \mathcal{T} będzie zbiorem drzew binarnych, w których wartość każdego pola należy do danego zbioru S , gdzie S jest liniowo uporządkowany przez relację „ \preceq ” (zobacz ćwiczenie 2.2.3–14). Dla dowolnych drzew T, T' z \mathcal{T} definiujemy: $T \preceq T'$ wtedy i tylko wtedy, gdy
 - i) T jest puste lub
 - ii) T i T' są niepuste oraz $\text{info}(\text{root}(T)) \prec \text{info}(\text{root}(T'))$; lub
 - iii) T i T' są niepuste, $\text{info}(\text{root}(T)) = \text{info}(\text{root}(T'))$, $\text{left}(T) \preceq \text{left}(T')$ i $\text{left}(T)$ nie jest równoważne $\text{left}(T')$; lub
 - iv) T i T' są niepuste, $\text{info}(\text{root}(T)) = \text{info}(\text{root}(T'))$, $\text{left}(T)$ jest równoważne $\text{left}(T')$ i $\text{right}(T) \preceq \text{right}(T')$.

$\text{left}(T)$ i $\text{right}(T)$ oznaczają lewe i prawe poddrzewo T . Udowodnij, że (a) $T \preceq T'$ i $T' \preceq T''$ pociąga $T \preceq T''$; (b) T jest równoważne T' wtedy i tylko wtedy, gdy $T \preceq T'$ i $T' \preceq T$; (c) dla dowolnych T, T' z \mathcal{T} mamy $T \preceq T'$ lub $T' \preceq T$. [Zatem jeśli równoważne drzewa z \mathcal{T} będziemy uznawać za równe, to relacja \preceq wyznaczy liniowe uporządkowanie zbioru \mathcal{T} . Takie uporządkowanie ma wiele zastosowań (na przykład upraszczanie wyrażeń algebraicznych). Gdy S ma tylko jeden element, tak że „info” w każdym węźle jest takie samo, mamy specjalny przypadek, w którym równoważność jest tym samym co podobieństwo].

26. [M24] Rozważmy porządek $T \preceq T'$ zdefiniowany w poprzednim ćwiczeniu. Udosownij twierdzenie analogiczne do twierdzenia A, podając warunek konieczny i wystarczający na to, by $T \preceq T'$, korzystając z porządku podwójnego zdefiniowanego w ćwiczeniu 18.

► **27.** [28] Zaprojektuj algorytm, który dla dwóch drzew T i T' sprawdza, czy $T \prec T'$, czy $T \succ T'$, czy T jest równoważne T' , w sensie relacji zdefiniowanej w ćwiczeniu 25. Zakładamy, że oba drzewa binarne mają fastrygę prawostronną, że każdy węzeł ma pola LLINK, RLINK, RTAG, INFO i że nie wolno korzystać z pomocniczego stosu.

28. [00] Jeśli skopiujemy drzewo za pomocą algorytmu C, to nowe drzewo binarne jest *równoważne* czy *podobne* do oryginału?

29. [M25] Udowodnij ściśle, że algorytm C jest poprawny.

► **30.** [22] Zaprojektuj algorytm fastrygowania drzewa; algorytm powinien na przykład przerobić drzewo (2) na (10). *Uwaga:* Tam, gdzie się da, korzystaj z notacji P^* i $P\$$, zamiast przepisywać kod algorytmu T.

31. [23] Zaprojektuj algorytm, który usuwa drzewo binarne z fastrygą prawostronną. Twój algorytm powinien umieścić wszystkie węzły poza atrapą na liście AVAIL, tak by pozostawiona sama atrapa reprezentowała drzewo puste. Załącz, że każdy węzeł drzewa ma pola LLINK, RLINK, RTAG; nie korzystaj z pomocniczego stosu.

32. [21] Przypuśćmy, że każdy węzeł drzewa binarnego ma cztery dowiązania: LLINK i RLINK wskazujące na lewe i prawe poddrzewo lub równe Λ , jak w drzewie niesfastrygowanym, oraz SUC i PRED, wskazujące następnik i poprzednik węzła w porządku symetrycznym. (Mamy $SUC(P) = P\$$ i $PRED(P) = \$P$. Takie drzewo zawiera więcej informacji niż drzewo sfastrygowane). Zaprojektuj algorytm podobny do algorytmu I realizujący wstawianie węzła do takiego drzewa.

► **33.** [30] Drzewo można sfastrygować na wiele różnych sposobów! Rozważmy następującą reprezentację korzystającą z pól LTAG, LLINK, RLINK węzła:

LTAG(P): zdefiniowane tak, jak w sfastrygowanym drzewie binarnym;

LLINK(P): zawsze równe P^* ;

RLINK(P): zdefiniowane tak, jak w niesfastrygowanym drzewie binarnym.

Omów algorytm wstawiania elementu oraz zapisz szczegółowo algorytm C (kopiowania drzewa) dla drzew o podanej reprezentacji.

34. [22] Niech P wskazuje na węzeł pewnego drzewa binarnego i niech HEAD wskazuje na atrapę w pustym drzewie binarnym. Podaj algorytm, który (i) usuwa NODE(P) i wszystkie jego poddrzewa, a następnie (ii) przyłącza NODE(P) wraz z poddrzewami do NODE(HEAD). Załącz, że wszystkie drzewa binarne, których dotyczy algorytm, mają fastrygę prawostronną oraz pola LLINK, RTAG, RLINK w każdym węzle.

35. [40] Zdefiniuj *drzewo ternarne* (i ogólnie t -arne dla $t \geq 2$) w sposób analogiczny do definicji drzewa binarnego i zbadaj, czy własności omówione w tym rozdziale (włączając ćwiczenia) można sensownie uogólnić na drzewa t -arne.

36. [M29] W ćwiczeniu 1.2.1–15 pokazujemy, że porządek leksykograficzny jest rozszerzeniem dobrego uporządkowania zbioru S na dobre uporządkowanie krotek n -elementowych o składowych ze zbioru S . Ćwiczenie 25 pokazuje, że liniowy porządek informacji w węzłach drzewa można rozszerzyć do liniowego porządku na drzewach za pomocą podobnej definicji. Jeśli relacja \prec dobrze porządkuje zbiór S , to czy rozszerzenie tej relacji opisane w ćwiczeniu 25 dobrze porządkuje zbiór T ?

- 37. [24] (D. Ferguson) Jeśli do zapamiętania dwóch dowiązań oraz pola INFO potrzeba dwóch słów maszynowych, to reprezentacja (2) wymaga $2n$ słów dla drzewa o n węzłach. Zaprojektuj bardziej oszczędny schemat reprezentacji, zakładając, że w jednym słowie mieście się *jedno* dowiązanie oraz pole INFO.

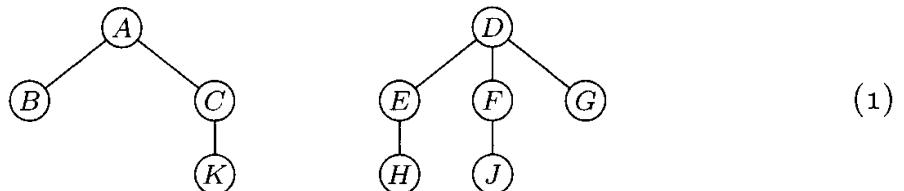
2.3.2. Reprezentacja drzew za pomocą drzew binarnych

Przerzucamy się teraz z drzew binarnych na zwykłe drzewa. Przypomnijmy podstawowe różnice między drzewami a drzewami binarnymi:

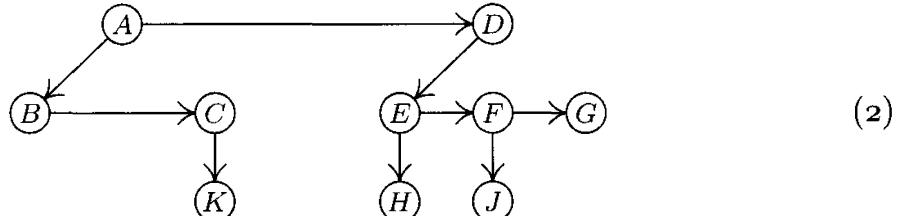
- 1) Drzewo zawsze ma korzeń, zatem nigdy nie jest puste; każdy węzeł drzewa może mieć 0, 1, 2, 3, ... dzieci.
- 2) Drzewo binarne może być puste, a każdy jego węzeł ma 0, 1 lub 2 dzieci; rozróżniamy „lewe” i „prawe” dziecko.

Przypomnijmy także, że las jest uporządkowanym zbiorem zera lub więcej drzew. Zbiór poddrzew położonych bezpośrednio poniżej wybranego węzła w drzewie tworzy las.

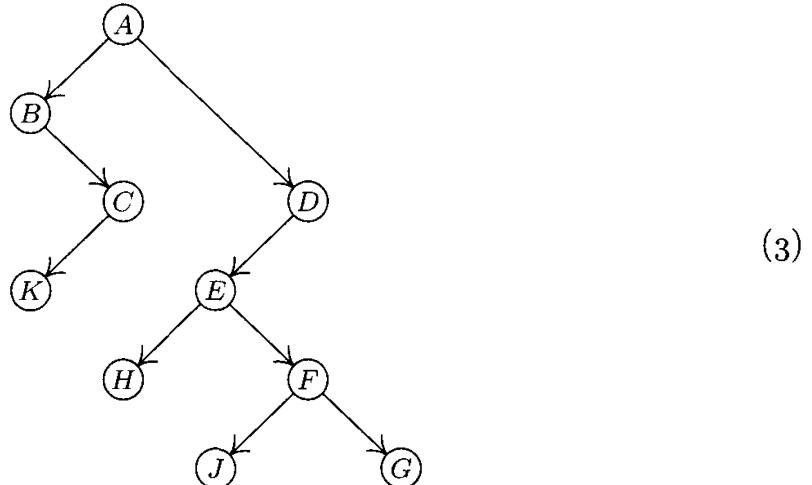
Istnieje naturalny sposób reprezentowania dowolnego lasu za pomocą drzewa binarnego. Przyjrzyjmy się następującemu lasowi zawierającemu dwa drzewa:



Drzewo binarne reprezentujące ten las otrzymujemy, łącząc wszystkie dzieci z jednej rodziny i usuwając dowiązania pionowe, poza dowiązaniem od rodzica do pierwszego dziecka:



Następnie obracamy schemat o 45° , troszeczkę upiększamy i otrzymujemy drzewo binarne:



Łatwo się przekonać, że również każde drzewo binarne odpowiada pewnemu jednoznacznie wyznaczonemu lasowi (wystarczy podane kroki wykonać od tyłu).

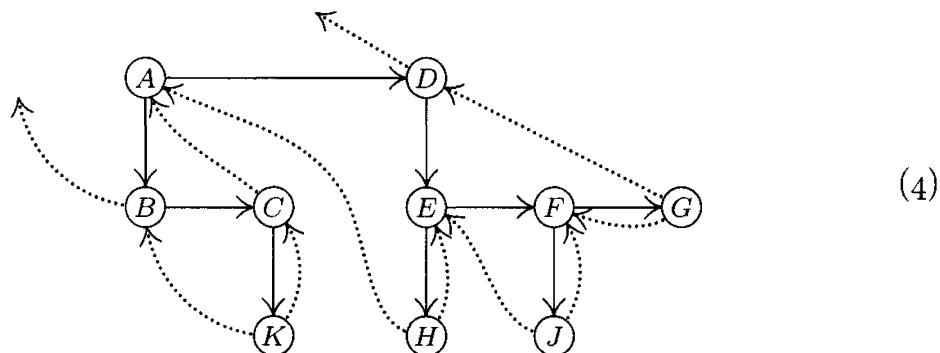
Przekształcenie prowadzące od (1) do (3) jest niezwykle ważne; nazywamy je *naturalną odpowiedniością* między lasami a drzewami binarnymi. W szczególności otrzymujemy odpowiedniość między drzewami a szczególną klasą drzew binarnych, konkretnie tych drzew, które mają korzeń, ale nie mają prawego poddrzewa. (Możemy równie dobrze nieco zmienić punkt widzenia i przyjąć, że korzeń drzewa odpowiada atrapie w drzewie binarnym, uzyskując tym samym wzajemnie jednoznaczną odpowiedniość między drzewami o $n + 1$ węzłach a drzewami binarnymi o n węzłach).

Niech $F = (T_1, T_2, \dots, T_n)$ będzie lasem. Drzewo binarne $B(F)$ reprezentujące F można w sposób ścisły zdefiniować następująco:

- Jeśli $n = 0$, to $B(F)$ jest drzewem pustym.
- Jeśli $n > 0$, to korzeniem $B(F)$ jest $\text{root}(T_1)$; lewym poddrzewem $B(F)$ jest drzewo $B(T_{11}, T_{12}, \dots, T_{1m})$, gdzie $T_{11}, T_{12}, \dots, T_{1m}$ są poddrzewami węzła $\text{root}(T_1)$; prawym poddrzewem $B(F)$ jest $B(T_2, \dots, T_n)$.

Powyzsze reguły precyzyjnie określają przekształcenie prowadzące od (1) do (3).

Od czasu do czasu wygodnie będzie rysować schematy drzew jak (2), bez obracania o 45° . *Sfastrygowane* drzewo binarne odpowiadające (1) to



(porównaj z rysunkiem 24 obróconym o 45°). Zauważmy, że *fastrygi prawostronne* prowadzą od skrajnie prawnego dziecka w rodzinie do rodzica. Dla fastryg lewostronnych nie ma takiej naturalnej interpretacji z powodu braku symetrii lewo-prawo.

Pomysły związane z obchodzeniem drzew binarnych badane w poprzednim rozdziale możemy obejrzeć teraz w kontekście lasów (i zwykłych drzew). Nie ma prostego odpowiednika porządku inorder, ponieważ nie jest oczywiste, w które miejsce pomiędzy potomkami należałoby wstawić korzeń, ale porządkie preorder i postorder jak najbardziej mają sens. Dwie podstawowe metody przechodzenia niepustego lasu można zdefiniować następująco:

Przechodzenie preorder

- Odwiedź korzeń pierwszego drzewa
- Przejdź poddrzewa pierwszego drzewa
- Przejdź pozostałe drzewa

Przechodzenie postorder

- Przejdź poddrzewa pierwszego drzewa
- Odwiedź korzeń pierwszego drzewa
- Przejdź pozostałe drzewa

By zrozumieć znaczenie tych dwóch metod przechodzenia, rozważmy następującą notację opisującą kształt drzew za pomocą zagnieżdżonych nawiasów:

$$(A(B, C(K)), D(E(H), F(J), G)). \quad (5)$$

Powyższy zapis odpowiada lasowi (1): drzewo przedstawiamy, zapisując informację przechowywaną w korzeniu, po której następują reprezentacje poddrzew; niepusty las zapisujemy jako ujętą w nawiasy listę reprezentacji drzew oddzielonych przecinkami.

Jeśli przejdziemy (1) w porządku preorder, to odwiedzimy węzły w kolejności $A B C K D E H F J G$ – a to po prostu (5) bez przecinków i nawiasów. Porządek preorder jest naturalnym sposobem wypisywania węzłów drzewa: zapisujemy najpierw korzeń, a potem jego potomków. Jeżeli strukturę drzewa zapisujemy za pomocą wcięć, jak na rysunku 20(c), to wiersze występują w porządku preorder. Numery rozdziałów w tej książce (zobacz rysunek 22) występują w porządku preorder; stąd na przykład po podrozdziale 2.3 jest punkt 2.3.1, po nim 2.3.2, 2.3.3, 2.3.4, 2.3.4.1, ..., 2.3.4.6, 2.3.5, 2.4 itd.

Warto zauważyć, że porządek preorder jest związany z uświeconą wiekami tradycją i można by go zupełnie sensownie nazywać *porządkiem dynastycznym*. Gdy umierał król, książę lub hrabia, tytuł przechodził na jego pierworodnego syna, w następnej kolejności na potomków pierwszego syna, a jeśli i oni wymarli, to w analogiczny sposób na innych synów w rodzinie. (Tradycja brytyjska każe uwzględniać również córki, z tym że w ramach jednej rodziny pierwszeństwo mają synowie). Teoretycznie moglibyśmy wziąć diagram dziedziczenia całej aristokracji i wypisać węzły w porządku preorder. Pozostawiając wyłącznie ludzi żyjących, otrzymalibyśmy *porządek sukcesji tronu* (poza modyfikacjami wynikającymi z aktów abdykacji).

Porządek postorder węzłów (1) to $B K C A H E J F G D$, analogicznie do preorder. Nieco odmienna jest notacja nawiasowa

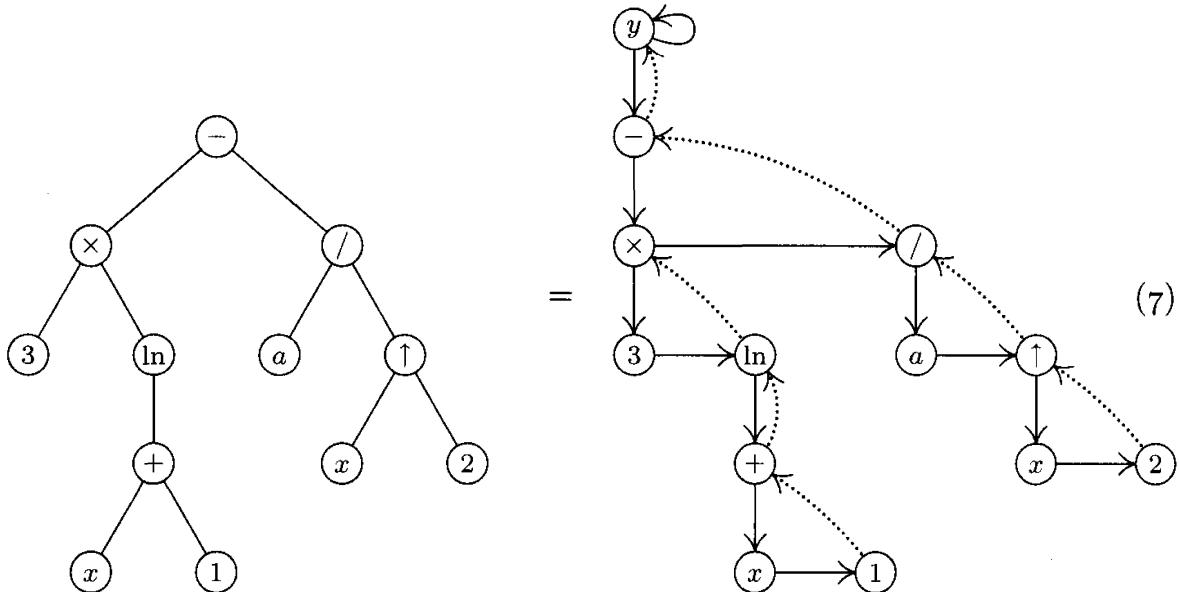
$$((B, (K)C)A, ((H)E, (J)F, G)D), \quad (6)$$

w której węzeł występuje bezpośrednio *po* swoich przodkach, a nie *przed*.

Definicje porządków preorder i postorder bardzo elegancko współgrają z naturalną odpowiedniością między lasami a drzewami binarnymi, ponieważ poddrzewa pierwszego drzewa w lesie odpowiadają lewemu poddrzewu drzewa binarnego, a pozostałe drzewa odpowiadają prawemu poddrzewu drzewa binarnego. Porównując te definicje z odpowiednimi definicjami na stronie 331, przekonamy się, że przechodzenie lasu w porządku preorder jest *dokładnie tym samym*, co przechodzenie odpowiadającego mu drzewa binarnego w porządku preorder. Przechodzenie lasu w porządku postorder jest dokładnie tym samym, co przechodzenie odpowiadającego mu drzewa binarnego w porządku *inorder*. Możemy zatem bez zmian korzystać z algorytmów opracowanych w punkcie 2.3.1. (Zauważmy, że porządek postorder na drzewach binarnych odpowiada porządkowi inorder, a *nie* preorder. To się nawet dobrze składa, bo skądinąd wiemy, że przechodzenie drzew binarnych w porządku postorder jest względnie trudne). Z uwagi na odpowiedniość następnik w porządku *postorder* węzła P w zwykłym

drzewie oznaczamy $\$$, podczas gdy ten sam symbol oznacza następnik w porządku *inorder* w drzewie binarnym.

Jako przykład praktycznego zastosowania powyższych metod zajmiemy się przekształceniemi wyrażeń algebraicznych. Najważniejszym sposobem reprezentowania wyrażeń algebraicznych są właśnie drzewa, a nie jedno- bądź dwuargumentowe układy symboli, a nawet nie drzewa binarne. Na przykład wyrażenie $y = 3 \ln(x + 1) - a/x^2$ można przedstawić za pomocą drzew:



Rysunek po lewej stronie to tradycyjne drzewo (jak na rysunku 21), w którym operatory dwuargumentowe $+$, $-$, \times , $/$ i \uparrow (potęgowanie) mają po dwa poddrzewa odpowiadające ich argumentom; jednoargumentowy operator „ \ln ” ma jedno poddrzewo; zmienne i stałe są liśćmi. Na rysunku po prawej jest pokazane drzewo binarne odpowiadające drzewu z rysunku po lewej, z atrapą y . Atrapa ma postać opisaną w 2.3.1–(8).

Warto zauważyć, że chociaż drzewo po lewej stronie (7) jest podobne do drzewa binarnego, jednak traktujemy je jako *drzewo* i reprezentujemy za pomocą drzewa binarnego o całkiem odmiennym kształcie, pokazanego po prawej stronie (7). Moglibyśmy napisać podprogramy wykonujące przekształcenia algebraiczne wprost na drzewach binarnych – na tzw. reprezentacjach wyrażeń algebraicznych za pomocą „trójkę”. Jednak w praktyce zastosowanie ogólnego sposobu reprezentowania drzew okaże się prostsze, ponieważ zwykłe drzewa łatwiej jest przechodzić w porządku postorder.

Węzły lewego drzewa z (7) w porządkach preorder i postorder to

$$- \quad \times \quad 3 \quad \ln \quad + \quad x \quad 1 \quad / \quad a \quad \uparrow \quad x \quad 2 \quad \text{preorder}; \quad (8)$$

$$3 \quad x \quad 1 \quad + \quad \ln \quad \times \quad a \quad x \quad 2 \quad \uparrow \quad / \quad - \quad \text{postorder}. \quad (9)$$

Postacie (8) i (9) wyrażeń algebraicznych są bardzo ważne i powszechnie nazywa się je „notacją polską”, ponieważ postać (8) została wprowadzona przez polskiego logika Jana Łukasiewicza. Wyrażenie (8) jest tzw. *zapisem przedrostkowym* wyrażenia (7), a wyrażenie (9) *zapisem przyrostkowym*. Do zajmującego tematu

notacji polskiej powrócimy w dalszych rozdziałach; na razie musimy zadowolić się informacją, że notacja polska jest ściśle związana z podstawowymi porządkami przechodzenia drzew.

Będziemy zakładać, że w programach na komputer **MIX** struktury drzewiaste reprezentujące wyrażenia algebraiczne mają następującą postać:

RTAG	RLINK	TYPE	LLINK	
				INFO

(10)

RLINK i LLINK mają dotychczasowe znaczenie, a RTAG ma wartość „–” dla dowiązań będących fastrygami (co odpowiada warunkowi $RTAG = 1$ w opisach algorytmów). Pole TYPE oznacza rodzaj węzła: $TYPE = 0$ – węzeł reprezentuje stałą, a INFO jest wartością tej stałej. $TYPE = 1$ – węzeł reprezentuje zmienną, a INFO jest pięcioznakową, literową nazwą tej zmiennej. $TYPE \geq 2$ – węzeł reprezentuje operator, INFO jest znakową nazwą operatora, a wartość $TYPE = 2, 3, 4, \dots$ wyznacza konkretne działanie $+, -, \times, /$ itd. Nie będziemy się przesadnie koncentrować na budowaniu struktury drzewiastej w pamięci, ponieważ bardzo szczegółowo zajmujemy się tym w rozdziale 10. Założymy po prostu, że drzewo zostało umieszczone w pamięci, odkładając na później pytania o wprowadzanie i wyprowadzanie drzewa.

Omówimy klasyczny przykład przekształcenia algebraicznego – znajdowanie pochodnej względem zmiennej x . Jedne z pierwszych programów wykonujących przekształcenia symboliczne służyły właśnie do różniczkowania – używano ich już w 1952 roku. Różniczkowanie wymaga zastosowania wielu metod przekształcania wyrażeń algebraicznych i ma znaczenie praktyczne w zastosowaniach naukowych.

Czytelnik nie obeznany z analizą matematyczną może potraktować ten problem jako ćwiczenie z przekształcania wyrażeń algebraicznych. Przekształcenie definiujemy za pomocą następujących reguł:

$$D(x) = 1 \quad (11)$$

$$D(a) = 0, \quad \text{jeśli } a \text{ jest stałą lub zmienną } \neq x \quad (12)$$

$$D(\ln u) = D(u)/u, \quad \text{gdzie } u \text{ jest dowolnym wyrażeniem} \quad (13)$$

$$D(-u) = -D(u) \quad (14)$$

$$D(u + v) = D(u) + D(v) \quad (15)$$

$$D(u - v) = D(u) - D(v) \quad (16)$$

$$D(u \times v) = D(u) \times v + u \times D(v) \quad (17)$$

$$D(u / v) = D(u)/v - (u \times D(v))/(v \uparrow 2) \quad (18)$$

$$D(u \uparrow v) = D(u) \times (v \times (u \uparrow (v - 1))) + ((\ln u) \times D(v)) \times (u \uparrow v) \quad (19)$$

Powyższe reguły umożliwiają wyznaczenie pochodnej $D(y)$ dowolnego wyrażenia y zbudowanego za pomocą podanych operatorów. Znak „–” w regule (14) jest operatorem jednoargumentowym i różni się od dwuargumentowego operatora „–” w regule (16); w reprezentacji drzewowej na oznaczenie jednoargumentowego operatora zmiany znaku będziemy używać nazwy „neg”.

Niestety, reguły (11)–(19) to nie wszystko. Jeżeli będziemy je stosować na ślepo, to względnie proste wyrażenie, jak na przykład

$$y = 3 \ln(x + 1) - a/x^2,$$

rozrośnie się do

$$\begin{aligned} D(y) &= 0 \cdot \ln(x + 1) + 3((1 + 0)/(x + 1)) \\ &\quad - (0/x^2 - (a(1(2x^{2-1}) + ((\ln x) \cdot 0)x^2))/(x^2)^2), \end{aligned} \quad (20)$$

co jest poprawne, ale zdecydowanie niezadowalające. By w odpowiedzi nie występuowało tyle zbędnych operacji, musimy obsłużyć specjalne przypadki dodawania zera, mnożenia przez zero, mnożenia przez jeden i podnoszenia do pierwszej potęgi. Te uproszczenia pozwalają zredukować wzór (20) do

$$D(y) = 3(1/(x + 1)) - ((-a(2x))/(x^2)^2), \quad (21)$$

co już jest wynikiem lepszym, ale wciąż niedoskonałym. Pojęcie „prawdziwie satysfakcjonującej odpowiedzi” nie może być dobrze zdefiniowane, ponieważ różni matematycy będą preferować różne postacie wyrażenia. Jasne jest jednak, że wzór (21) można uprościć. By uzyskać znaczący postęp w takich przypadkach jak (21), musimy opracować podprogramy upraszczania wyrażeń algebraicznych (zobacz ćwiczenie 17), które zredukują wyrażenie (21) na przykład do

$$D(y) = 3(x + 1)^{-1} + 2ax^{-3}. \quad (22)$$

Poniżej założymy jednak, że postać (21) jest satysfakcjonującą.

Jak zwykle najbardziej będą nas interesować szczegóły reprezentacji danych w pamięci i przekształceń wykonywanych na nich przez algorytm. W wielu językach programowania wysokiego poziomu dostępne są gotowe podprogramy do przekształcania wyrażeń algebraicznych. Naszym celem jest jednak zdobycie doświadczenia w zakresie wykonywania podstawowych operacji na drzewach.

Pomysł, na którym opiera się algorytm, polega na przejściu drzewa w porządku postorder i obliczeniu pochodnej każdego węzła. W ten sposób obliczymy w końcu pochodną całego drzewa. Korzystamy z porządku postorder, co znaczy, że do operatora (na przykład „+”) docieramy po zróżniczkowaniu jego argumentów. Z reguł (11)–(19) wynika, że każde podwyrażenie różniczkowanego wyrażenia trzeba przedzej czy później zróżniczkować, więc czemu nie różniczkować w porządku postorder.

Korzystając z drzewa binarnego z fastrygą prawostronną, unikamy konieczności użycia stosu. Drzewo sfastrygowane ma jednak tę wadę, że musimy tworzyć kopie poddrzew. Na przykład może się zdarzyć, że realizując regułę dla $D(u \uparrow v)$ będziemy musieli i u , i v kopiować trzykrotnie. Jeżeli zdecydowalibyśmy się zamiast drzewa korzystać z Listy, jak w punkcie 2.3.5, to moglibyśmy uniknąć operacji kopiowania tego rodzaju.

Algorytm D (Różniczkowanie). Jeśli Y jest adresem atrapy w drzewie, która zawiera wskaźnik do wyrażenia reprezentowanego według opisu poniżej i jeśli DY jest adresem atrapy w pustym drzewie, to po zakończeniu działania algorytmu

$\text{NODE}(DY)$ wskazuje na drzewo reprezentujące pochodną analityczną Y względem zmiennej „ X ”.

- D1. [Inicjowanie] Przyjmij $P \leftarrow Y\$$ (tj. przypisz pierwszy węzeł drzewa w porządku postorder, który jest pierwszym węzłem odpowiedniego drzewa binarnego w porządku inorder).
- D2. [Różniczkowanie] Przyjmij $P1 \leftarrow \text{LLINK}(P)$; jeśli $P1 \neq \Lambda$, to $Q1 \leftarrow \text{RLINK}(P1)$. Następnie wykonaj opisany poniżej podprogram $\text{DIFF}[\text{TYPE}(P)]$. (Podprogramy $\text{DIFF}[0]$, $\text{DIFF}[1]$ itd. zbudują pochodną drzewa o korzeniu P i ustawią zmienną wskaźnikową Q na adres korzenia pochodnej. W pierwszej kolejności przypisujemy wartości do zmiennych $P1$ i $Q1$, by uprościć specyfikację podprogramów DIFF).
- D3. [Odtwarzanie wskaźników] Jeśli $\text{TYPE}(P)$ oznacza operator dwuargumentowy, to przyjmij $\text{RLINK}(P1) \leftarrow P2$. (Zobacz wyjaśnienie w następnym kroku).
- D4. [Przejście do $P\$$] Przyjmij $P2 \leftarrow P$, $P \leftarrow P\$$. Jeśli teraz $\text{RTAG}(P2) = 0$ (tj. jeśli $\text{NODE}(P2)$ ma rodzeństwo z prawej), to przyjmij $\text{RLINK}(P2) \leftarrow Q$. (Tu jest trochę magii: tymczasowo zmieniamy kształt drzewa Y ; dowiązanie do pochodnej wyrażenia $P2$ jest zachowane do późniejszego wykorzystania. Brakujące dowiązanie zostanie odtworzone później w kroku D3.Więcej na ten temat można znaleźć w ćwiczeniu 21).
- D5. [Czy gotowe?] Jeśli $P \neq Y$, to wróć do kroku D2. W przeciwnym razie przyjmij $\text{LLINK}(DY) \leftarrow Q$ i $\text{RLINK}(Q) \leftarrow DY$, $\text{RTAG}(Q) \leftarrow 1$. ■

Schemat działania opisany w algorytmie D stanowi w zasadzie tylko tło dla właściwych operacji różniczkowania realizowanych przez podprogramy $\text{DIFF}[0]$, $\text{DIFF}[1]$, ... wywoływanego w kroku D2. Algorytm D na wiele sposobów przypomina część sterującą interpretera lub symulatora (zobacz punkt 1.4.3), tyle że porusza się po drzewie, a nie po ciągu rozkazów.

Aby algorytm D był kompletny, musimy zdefiniować podprogramy realizujące różniczkowanie. W poniższym omówieniu stwierdzenie „ P wskazuje na drzewo” oznacza, że $\text{NODE}(P)$ jest korzeniem drzewa przechowywanego w pamięci jako drzewo binarne z fastrygą prawostronną, ale zarówno $\text{RLINK}(P)$, jak i $\text{RTAG}(P)$ w tym drzewie nie będą miały znaczenia. Będziemy korzystać z funkcji służących do tworzenia nowych drzew przez łączenie mniejszych drzew. Niech x oznacza węzeł pewnego rodzaju (stałą, zmienną lub operator), a U i V niech oznaczają wskaźniki do drzew. Wówczas

$\text{TREE}(x, U, V)$ tworzy nowe drzewo z x w korzeniu oraz poddrzewach U i V :
 $W \Leftarrow \text{AVAIL}$, $\text{INFO}(W) \leftarrow x$, $\text{LLINK}(W) \leftarrow U$, $\text{RLINK}(U) \leftarrow V$, $\text{RTAG}(U) \leftarrow 0$,
 $\text{RLINK}(V) \leftarrow W$, $\text{RTAG}(V) \leftarrow 1$.

$\text{TREE}(x, U)$ w podobny sposób tworzy drzewo o tylko jednym poddrzewie:
 $W \Leftarrow \text{AVAIL}$, $\text{INFO}(W) \leftarrow x$, $\text{LLINK}(W) \leftarrow U$, $\text{RLINK}(U) \leftarrow W$, $\text{RTAG}(U) \leftarrow 1$.

$\text{TREE}(x)$ tworzy nowe drzewo, w którym korzeń x jest jednocześnie liściem:
 $W \Leftarrow \text{AVAIL}$, $\text{INFO}(W) \leftarrow x$, $\text{LLINK}(W) \leftarrow \Lambda$.

Zakładamy ponadto, że $\text{TYPE}(W)$ ma odpowiednią wartość w zależności od x . We wszystkich przypadkach wartością TREE jest W , tzn. wskaźnik do nowo skonstru-

owanego drzewa. Czytelnik powinien uważnie przestudiować te trzy definicje, ponieważ pokazują one zasady reprezentowania drzewa za pomocą drzewa binarnego. Inna funkcja, $\text{COPY}(U)$, tworzy kopię drzewa wskazywanego przez U , a jej wartością jest wskaźnik do kopii. Proste funkcje TREE i COPY umożliwiają budowanie pochodnej w łatwy sposób, krok po kroku.

Operatory bezargumentowe (*stałe i zmienne*). $\text{NODE}(P)$ jest liściem, a wartości P_1, P_2, Q_1 i Q przed operacją nie mają znaczenia.

DIFF[0]: ($\text{NODE}(P)$ jest stałą). Przyjmij $Q \leftarrow \text{TREE}(0)$.

DIFF[1]: ($\text{NODE}(P)$ jest zmienną). Jeśli $\text{INFO}(P) = \text{"X"}$, to przyjmij $Q \leftarrow \text{TREE}(1)$; w przeciwnym razie przyjmij $Q \leftarrow \text{TREE}(0)$.

Operatory jednoargumentowe (*logarytm i zmiana znaku*). $\text{NODE}(P)$ ma jedno dziecko U , na które wskazuje P_1 ; Q wskazuje na $D(U)$. Wartości P_2 i Q_1 przed operacją nie mają znaczenia.

DIFF[2]: ($\text{NODE}(P)$ jest typu „ln”). Jeśli zachodzi $\text{INFO}(Q) \neq 0$, to przyjmij $Q \leftarrow \text{TREE}(\text{"/"}, Q, \text{COPY}(P_1))$.

DIFF[3]: ($\text{NODE}(P)$ jest typu „neg”). Jeśli $\text{INFO}(Q) \neq 0$, to przyjmij $Q \leftarrow \text{TREE}(\text{"neg"}, Q)$.

Operatory dwuargumentowe (*dodawanie, odejmowanie, mnożenie, dzielenie, potęgowanie*). $\text{NODE}(P)$ ma dwoje dzieci U i V , na które wskazują P_1 i P_2 ; Q_1 i Q wskazują odpowiednio na $D(U)$ i $D(V)$.

DIFF[4]: (Operator „+”). Jeśli $\text{INFO}(Q_1) = 0$, to przyjmij $\text{AVAIL} \Leftarrow Q_1$. W przeciwnym razie jeśli $\text{INFO}(Q) = 0$, to przyjmij $\text{AVAIL} \Leftarrow Q$ i $Q \leftarrow Q_1$; w przeciwnym razie przyjmij $Q \leftarrow \text{TREE}(\text{"+"}, Q_1, Q)$.

DIFF[5]: (Operator „-”). Jeśli $\text{INFO}(Q) = 0$, to przyjmij $\text{AVAIL} \Leftarrow Q$ oraz $Q \leftarrow Q_1$. W przeciwnym razie jeśli $\text{INFO}(Q_1) = 0$, to przyjmij $\text{AVAIL} \Leftarrow Q_1$ i $Q \leftarrow \text{TREE}(\text{"neg"}, Q)$; w przeciwnym razie przyjmij $Q \leftarrow \text{TREE}(\text{"-"}, Q_1, Q)$.

DIFF[6]: (Operator „ \times ”). Najpierw jeśli $\text{INFO}(Q_1) \neq 0$, to przyjmij $Q_1 \leftarrow \text{MULT}(Q_1, \text{COPY}(P_2))$. Następnie jeśli $\text{INFO}(Q) \neq 0$, to $Q \leftarrow \text{MULT}(\text{COPY}(P_1), Q)$. Przejdź do DIFF[4].

$\text{MULT}(U, V)$ jest nową funkcją, która służy do konstruowania drzewa reprezentującego $U \times V$, a także do sprawdzania, czy U lub V jest równe 1:

jeśli $\text{INFO}(U) = 1$ i $\text{TYPE}(U) = 0$, to przyjmij $\text{AVAIL} \Leftarrow U$, $\text{MULT}(U, V) \leftarrow V$;
jeśli $\text{INFO}(V) = 1$ i $\text{TYPE}(V) = 0$, to przyjmij $\text{AVAIL} \Leftarrow V$, $\text{MULT}(U, V) \leftarrow U$;
w przeciwnym razie $\text{MULT}(U, V) \leftarrow \text{TREE}(\text{"x"}, U, V)$.

DIFF[7]: (Operacja „/”). Jeśli $\text{INFO}(Q_1) \neq 0$, to przyjmij

$Q_1 \leftarrow \text{TREE}(\text{"/"}, Q_1, \text{COPY}(P_2))$.

Następnie jeśli $\text{INFO}(Q) \neq 0$, to przyjmij

$Q \leftarrow \text{TREE}(\text{"/"}, \text{MULT}(\text{COPY}(P_1), Q), \text{TREE}(\text{"↑"}, \text{COPY}(P_2), \text{TREE}(2)))$.

Przejdź do DIFF[5].

DIFF[8]: (Operacja „ \uparrow ”). Zobacz ćwiczenie 12.

Na zakończenie tego punktu pokażemy, w jaki sposób powyższy opis przełożyć na program komputerowy. Zaczynamy „od zera”, dysponując jedynie językiem maszynowym komputera MIX.

Program D (Różniczkowanie). Poniższy program w języku MIXAL realizuje algorytm D, gdzie $rI1 \equiv P$, $rI3 \equiv P_2$, $rI4 \equiv P_1$, $rI5 \equiv Q$, $rI6 \equiv Q_1$. Kolejność obliczeń została nieznacznie zmieniona z uwagi na wygodę kodowania.

```

001 * RÓZNICZKOWANIE W DRZEWIE Z FASTRYGĄ PRAWOSTRONNĄ
002 LLINK EQU 4:5           Definicje pól, zobacz (10).
003 RLINK EQU 1:2
004 RLINKT EQU 0:2
005 TYPE EQU 3:3
006 * MAIN CONTROL ROUTINE D1. Inicjowanie.
007 D1 STJ 9F               Traktuj program sterujący jak podprogram.
008 LD4 Y(LLINK)             $P_1 \leftarrow LLINK(Y)$ , przygotuj się na poszukiwanie  $Y\$$ .
009 1H ENT2 0,4               $P \leftarrow P_1$ .
010 2H LD4 0,2(LLINK)        $P_1 \leftarrow LLINK(P)$ .
011 J4NZ 1B                 Jeżeli  $P_1 \neq \Lambda$ , powtóż.
012 D2 LD1 0,2(TYPE)        D2. Różniczkowanie.
013 JMP *+1,1                Skocz do DIFF[TYPE(P)].
014 JMP CONSTANT             Przerzuć sterowanie do pozycji dla DIFF[0].
015 JMP VARIABLE              DIFF[1].
016 JMP LN                   DIFF[2].
017 JMP NEG                  DIFF[3].
018 JMP ADD                  DIFF[4].
019 JMP SUB                  DIFF[5].
020 JMP MUL                  DIFF[6].
021 JMP DIV                  DIFF[7].
022 JMP PWR                  DIFF[8].
023 D3 ST3 0,4(RLINK)        D3. Odtworzenie dowiązania.  $RLINK(P_1) \leftarrow P_2$ .
024 D4 ENT3 0,2               D4. Przejście do  $P\$$ .  $P_2 \leftarrow P$ .
025 LD2 0,2(RLINKT)           $P \leftarrow RLINKT(P)$ .
026 J2N 1F                   Skocz, gdy RTAG(P) = 1;
027 ST5 0,3(RLINK)            w przeciwnym razie  $RLINK(P_2) \leftarrow Q$ .
028 JMP 2B                   ( $NODE(P\$)$  będzie liściem).
029 1H ENN2 0,2
030 D5 ENT1 -Y,2              D5. Czy gotowe?
031 LD4 0,2(LLINK)             $P_1 \leftarrow LLINK(P)$ , przygotuj krok D2.
032 LD6 0,4(RLINK)
033 J1NZ D2                 Skocz do D2, jeśli  $P \neq Y$ ;
034 ST5 DY(LLINK)            w przeciwnym razie  $LLINK(DY) \leftarrow Q$ .
035 ENNA DY
036 STA 0,5(RLINKT)           $RLINK(Q) \leftarrow DY$ ,  $RTAG(Q) \leftarrow 1$ .
037 9H JMP *                  Wyjdź z podprogramu różniczkowania. ■

```

Następna część programu zawiera podstawowe podprogramy TREE i COPY. Podprogram TREE ma trzy wejścia TREE0, TREE1 i TREE2, dla różnych liczb poddrzew w konstruowanym drzewie. Niekolejnie od tego, którego wejścia używamy, rA

musi zawierać adres specjalnej stałej mówiącej o tym, jakiego rodzaju węzłem jest korzeń konstruowanego drzewa. Definicje stałych znajdują się w wierszach 105–124.

038 * KONSTRUKTORY DRZEW

039	TREE0	STJ	9F	Funkcja TREE(rA):
040		JMP	2F	
041	TREE1	ST1	3F(0:2)	Funkcja TREE(rA,rI1):
042		JSJ	1F	
043	TREE2	STX	3F(0:2)	Funkcja TREE(rA,rX,rI1):
044	3H	ST1	*(RLINKT)	RLINK(rX) ← rI1, RTAG(rX) ← 0.
045	1H	STJ	9F	
046		LDXN	AVAIL	
047		JXZ	OVERFLOW	
048		STX	0,1(RLINKT)	RLINK(rI1) ← AVAIL, RTAG(rI1) ← 1.
049		LDX	3B(0:2)	
050		STA	*+1(0:2)	
051		STX	*(LLINK)	Ustaw LLINK następnego korzenia.
052	2H	LD1	AVAIL	rI1 ← AVAIL.
053		J1Z	OVERFLOW	
054		LDX	0,1(LLINK)	
055		STX	AVAIL	
056		STA	*+1(0:2)	Kopiuj informacje z korzenia do nowego węzła.
057		MOVE	*(2)	
058		DEC1	2	Ustaw rI1, by wskazywało na nowy korzeń.
059	9H	JMP	*	Wyjdź z TREE, rI1 wskazuje na nowe drzewo.
060	COPYP1	ENT1	0,4	COPY(P1), wejście specjalne do COPY.
061		JSJ	COPY	
062	COPYP2	ENT1	0,3	COPY(P2), wejście specjalne do COPY.
063	COPY	STJ	9F	Funkcja COPY(rI1):
:		:		(zobacz ćwiczenie 13).
104	9H	JMP	*	Wyjdź z COPY, rI1 wskazuje na nowe drzewo.
105	CON0	CON	0	Węzeł reprezentujący stałą „0”.
106		CON	0	
107	CON1	CON	0	Węzeł reprezentujący „1”.
108		CON	1	
109	CON2	CON	0	Węzeł reprezentujący „2”.
110		CON	2	
111	LOG	CON	2(TYPE)	Węzeł reprezentujący „ln”.
112		ALF	LN	
113	NEGOP	CON	3(TYPE)	Węzeł reprezentujący „neg”.
114		ALF	NEG	
115	PLUS	CON	4(TYPE)	Węzeł reprezentujący „+”.
116		ALF	+	
117	MINUS	CON	5(TYPE)	Węzeł reprezentujący „-”.
118		ALF	-	
119	TIMES	CON	6(TYPE)	Węzeł reprezentujący „×”.
120		ALF	*	
121	SLASH	CON	7(TYPE)	Węzeł reprezentujący „/”.
122		ALF	/	

123 UPARROW CON 8(TYPE) Węzeł reprezentujący „↑”.
 124 ALF ** ■

Reszta programu to właściwe podprogramy wykonujące różniczkowanie DIFF[0], DIFF[1], Są one napisane w ten sposób, że po przetworzeniu operatora dwuargumentowego sterowanie wraca do kroku D3, a w przypadku innych operatorów do kroku D4.

125 * PODPROGRAMY RÓŻNICZKOWANIA		
126 VARIABLE LDX 1,2		
127 ENTA CON1		
128 CMPX 2F	Czy INFO(P) = „X”?	
129 JE **2	Jeśli tak, wywołaj TREE(1).	
130 CONSTANT ENTA CONO	Wywołaj TREE(0).	
131 JMP TREE0		
132 1H ENT5 0,1	Q ← lokacja nowego drzewa.	
133 JMP D4	Wróć do części sterującej.	
134 2H ALF X		
135 LN LDA 1,5		
136 JAZ D4		
137 JMP COPYP1	Wróć do części sterującej, jeśli INFO(Q) = 0; w przeciwnym razie rI1 ← COPY(P1).	
138 ENTX 0,5		
139 ENTA SLASH		
140 JMP TREE2	rI1 ← TREE(„/”, Q, rI1).	
141 JMP 1B	Q ← rI1, wróć do części sterującej.	
142 NEG LDA 1,5		
143 JAZ D4	Wróć, jeśli INFO(Q) = 0.	
144 ENTA NEGOP		
145 ENT1 0,5		
146 JMP TREE1	rI1 ← TREE(„neg”, Q).	
147 JMP 1B	Q ← rI1, wróć do części sterującej.	
148 ADD LDA 1,6		
149 JANZ 1F	Skocz, jeśli nie INFO(Q1) = 0.	
150 3H LDA AVAIL	AVAIL ← Q1.	
151 STA 0,6(LLINK)		
152 ST6 AVAIL	Wróć do części sterującej;	
153 JMP D3	operator dwuargumentowy.	
154 1H LDA 1,5		
155 JANZ 1F	Skocz, chyba że INFO(Q) = 0.	
156 2H LDA AVAIL	AVAIL ← Q.	
157 STA 0,5(LLINK)		
158 ST5 AVAIL		
159 ENT5 0,6	Q ← Q1.	
160 JMP D3	Wróć do części sterującej.	
161 1H ENTA PLUS	Przygotuj wywołanie TREE(„+”, Q1, Q).	
162 4H ENTX 0,6		
163 ENT1 0,5		
164 JMP TREE2		
165 ENT5 0,1	Q ← TREE(„±”, Q1, Q).	
166 JMP D3	Wróć do części sterującej.	

167	SUB	LDA 1,5	
168		JAZ 2B	Skocz, jeśli INFO(Q) = 0.
169		LDA 1,6	
170		JANZ 1F	Skocz, chyba że INFO(Q1) = 0.
171		ENTA NEGOP	
172		ENT1 0,5	
173		JMP TREE1	
174		ENT5 0,1	$Q \leftarrow \text{TREE}(\text{,,neg"}, Q)$.
175		JMP 3B	AVAIL $\Leftarrow Q_1$ i wróć.
176	1H	ENTA MINUS	Przygotuj wywołanie TREE(,,-,Q1,Q).
177		JMP 4B	
178	MUL	LDA 1,6	
179		JAZ 1F	Skocz, jeśli INFO(Q1) = 0;
180		JMP COPY2	w przeciwnym razie $rI1 \leftarrow \text{COPY}(P2)$.
181		ENTA 0,6	
182		JMP MULT	$rI1 \leftarrow \text{MULT}(Q_1, \text{COPY}(P2))$.
183		ENT6 0,1	$Q_1 \leftarrow rI1$.
184	1H	LDA 1,5	
185		JAZ ADD	Skocz, jeśli INFO(Q) = 0;
186		JMP COPYP1	w przeciwnym razie $rI1 \leftarrow \text{COPY}(P1)$.
187		ENTA 0,1	
188		ENT1 0,5	
189		JMP MULT	$rI1 \leftarrow \text{MULT}(\text{COPY}(P1), Q)$.
190		ENT5 0,1	$Q \leftarrow rI1$.
191		JMP ADD	
192	MULT	STJ 9F	Podprogram MULT(rA,rI1):
193		STA 1F(0:2)	Ustal rA $\equiv U$, rI1 $\equiv V$.
194		ST2 8F(0:2)	Przechowaj rI2.
195	1H	ENT2 *	$rI2 \leftarrow U$.
196		LDA 1,2	Sprawdź, czy INFO(U) = 1
197		DECA 1	
198		JANZ 1F	
199		LDA 0,2(TYPE)	i czy TYPE(U) = 0.
200		JAZ 2F	
201	1H	LDA 1,1	Jeśli nie, to sprawdź, czy INFO(V) = 1
202		DECA 1	
203		JANZ 1F	
204		LDA 0,1(TYPE)	i czy TYPE(V) = 0.
205		JANZ 1F	
206		ST1 *+2(0:2)	Jeśli tak, zamień $U \leftrightarrow V$.
207		ENT1 0,2	
208		ENT2 *	
209	2H	LDA AVAIL	AVAIL $\Leftarrow U$.
210		STA 0,2(LLINK)	
211		ST2 AVAIL	
212		JMP 8F	Wynikiem jest V.
213	1H	ENTA TIMES	
214		ENTX 0,2	
215		JMP TREE2	Wynikiem jest TREE(,,x",U,V).

216 8H	ENT2 *	Odtwórz rI2.
217 9H	JMP *	Wyjdź z MULT z wynikiem w rI1. ■

Pozostałe dwa podprogramy DIV i PWR są podobne. Ich zakodowanie pozostawiamy jako ćwiczenie (zobacz ćwiczenia 15 i 16).

ĆWICZENIA

- ▶ 1. [20] W tekście podaliśmy formalną definicję drzewa binarnego $B(F)$ odpowiadającego lasowi F . Odwróć tę definicję, tj. podaj formalną definicję lasu $F(B)$ odpowiadającego drzewu binarnemu B .
- ▶ 2. [20] W podrozdziale 2.3 zdefiniowaliśmy notację dziesiętną Dewey'a dla lasów, a w ćwiczeniu 2.3.1–5 dla drzew binarnych. Węzeł „J” w lesie (1) jest reprezentowany przez „2.2.1”, a w odpowiadającym mu drzewie binarnym (3) przez „11010”. Jeśli to możliwe podaj regułę, która bezpośrednio wyraża naturalną odpowiedniość między drzewami i drzewami binarnymi w języku notacji dziesiętnej Dewey'a.
- 3. [22] Jaki jest związek między notacją dziesiętną Dewey'a dla węzłów w lesie a porządkiem preorder i postorder odwiedzania tych węzłów?
- 4. [19] Następne zdanie jest prawdziwe czy fałszywe? „Niezależnie od tego, czy przechodzimy drzewo w porządku preorder czy postorder, porządek odwiedzania liści jest taki sam”.
- 5. [23] Można zdefiniować inną odpowiedniość między lasami a drzewami binarnymi. Niech **RLINK(P)** wskazuje na skrajnie prawe dziecko **NODE(P)**, a **LLINK(P)** na najbliższe rodzeństwo z lewej. Niech F będzie lasem odpowiadającym w ten sposób drzewu binarnemu B . Jaki porządek węzłów B odpowiada porządkowi (a) preorder (b) postorder węzłów F ?
- 6. [25] Niech T będzie niepustym drzewem binarnym, w którym każdy węzeł ma 0 lub 2 dzieci. Jeżeli spojrzymy na T jak na zwykłe drzewo, to będzie ono odpowiadało (w naturalnej odpowiedniości) *innemu* drzewu binarnemu T' . Czy istnieje jakiś prosty związek między porządkami preorder, inorder i postorder węzłów T (w sensie drzewa binarnego) i T' ?
- 7. [M20] Na las można patrzeć jak na porządek częściowy, jeśli przyjmiemy, że każdy węzeł poprzedza swoich potomków w drzewie. Czy węzły są posortowane topologicznie (wg definicji z punktu 2.2.3), gdy wypiszemy je w porządku: (a) preorder; (b) postorder; (c) odwrotnym do preorder; (d) odwrotnym do postorder?
- 8. [M20] W ćwiczeniu 2.3.1–25 pokazaliśmy, w jaki sposób porządek na danych zawartych w węzłach drzewa binarnego można rozszerzyć na porządek liniowy na wszystkich drzewach binarnych. Stosując tę samą konstrukcję oraz naturalną odpowiedniość, otrzymujemy porządek na wszystkich drzewach. Sformułuj definicję z ćwiczenia 2.3.1–25 dla zwykłych drzew.
- 9. [M21] Pokaż, że całkowita liczba węzłów wewnętrznych w lesie jest w prosty sposób związana z całkowitą liczbą prawych dowiązań równych Λ w odpowiadającym temu lasowi niesfastrygowanym drzewie binarnym.
- 10. [M23] Niech F będzie lasem, którego węzły w porządku preorder tworzą ciąg u_1, u_2, \dots, u_n . Niech F' będzie lasem, którego węzły w porządku preorder tworzą ciąg $u'_1, u'_2, \dots, u'_{n'}$. Niech $d(u)$ oznacza stopień (liczbę dzieci) węzła u . Przyjmując te oznaczenia sformułuj i udowodnij twierdzenie analogiczne do twierdzenia 2.3.1A.

- 11.** [15] Narysuj drzewo analogiczne do (7), odpowiadające wyrażeniu $y = e^{-x^2}$.
- 12.** [M21] Podaj brakującą specyfikację podprogramu DIFF[8] (operacja „ \uparrow ”).
- **13.** [26] Zapisz w języku maszynowym komputera MIX podprogram COPY (który należy umieścić w wierszach 063 – 104 programu w tekście rozdziału). [*Wskazówka:* Dostosuj algorytm 2.3.1C do drzew binarnych z fastrygą prawostronną, pamiętając o odpowiednich warunkach początkowych].
- **14.** [M21] Ile czasu potrzebuje program z ćwiczenia 13 na skopiowanie drzewa o n węzłach?
- 15.** [23] Zapisz w języku maszynowym komputera MIX podprogram DIV odpowiadający podprogramowi DIFF[7]. (Podprogram należy dołączyć do programu z tekstu tego punktu, za wierszem 217).
- 16.** [24] Zapisz w języku maszynowym komputera MIX podprogram PWR, odpowiadający podprogramowi DIFF[8] z ćwiczenia 12. (Podprogram należy dołączyć do programu z tekstu tego punktu, zaraz za podprogramem z ćwiczenia 15).
- 17.** [M40] Napisz program realizujący przekształcenia algebraiczne, który by upraszczał na przykład (20) lub (21) do (22). [*Wskazówki:* Do każdego węzła dodaj nowe pole reprezentujące jego współczynnik (dla składników sum) lub jego potęgę (dla czynników iloczynów). Zastosuj tożsamości algebraiczne, jak zastępowanie $\ln(u \uparrow v)$ przez $v \ln u$. Tam gdzie się da, usuń operacje $-$, $/$, \uparrow i neg, korzystając z przedstawień za pomocą dodawania lub mnożenia. Uczyń $+$ i \times operatorami n -argumentowymi. Grupuj wyrazy podobne za pomocą sortowania składników w porządku drzewowym (ćwiczenie 8); niektóre sumy i iloczyny zredukują się do zera lub jedności, otwierając możliwość drogi do dalszych uproszczeń. Narzuca się jeszcze wiele przekształceń, jak na przykład zastępowanie sumy logarytmów logarytmem iloczynu].
- **18.** [25] Drzewo zorientowane reprezentowane za pomocą n dowiązań PARENT[j] dla $1 \leq j \leq n$ wyznacza drzewo uporządkowane, jeżeli węzły w każdej rodzinie są uporządkowane względem swoich lokacji. Zaprojektuj wydajny algorytm, który konstruuje listę dwukierunkową zawierającą wszystkie węzły tego drzewa w porządku preorder. Na przykład dla

$$\begin{aligned} j &= 1 2 3 4 5 6 7 8 \\ \text{PARENT}[j] &= 3 8 4 0 4 8 3 4 \end{aligned}$$

Twój algorytm powinien przypisać

$$\begin{aligned} \text{LLINK}[j] &= 3 8 4 6 7 2 1 5 \\ \text{RLINK}[j] &= 7 6 1 3 8 4 5 2 \end{aligned}$$

oraz poinformować, że korzeniem jest węzeł 4.

- 19.** [M35] Krata wolna jest strukturą algebraiczną, którą (na potrzeby tego ćwiczenia) można zdefiniować po prostu jako zbiór wszystkich formuł utworzonych za pomocą zmiennych oraz dwóch operatorów dwuargumentowych „ \vee ” i „ \wedge ”. Między pewnymi formułami X i Y zgodnie z poniższymi regułami zachodzi relacja „ $X \succeq Y$ ”:

- i) $X \vee Y \succeq W \wedge Z$ wtedy i tylko wtedy, gdy $X \vee Y \succeq W$ lub $X \vee Y \succeq Z$, lub $X \succeq W \wedge Z$, lub $Y \succeq W \wedge Z$;
- ii) $X \wedge Y \succeq Z$ wtedy i tylko wtedy, gdy $X \succeq Z$ i $Y \succeq Z$;
- iii) $X \succeq Y \vee Z$ wtedy i tylko wtedy, gdy $X \succeq Y$ i $X \succeq Z$;
- iv) $x \succeq Y \wedge Z$ wtedy i tylko wtedy, gdy $x \succeq Y$ lub $x \succeq Z$, gdzie x jest zmienną;

- v) $X \vee Y \succeq z$ wtedy i tylko wtedy, gdy $X \succeq z$ lub $Y \succeq z$, gdzie z jest zmienną;
vi) $x \succeq y$ wtedy i tylko wtedy, gdy $x = y$, gdzie x i y są zmiennymi.

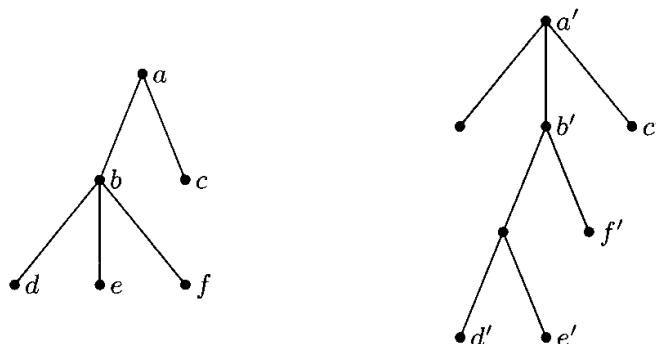
Na przykład $a \wedge (b \vee c) \succeq (a \wedge b) \vee (a \wedge c) \not\succeq a \wedge (b \vee c)$.

Zaprojektuj algorytm, który sprawdza, czy dla danych formuł X i Y w kracie wolnej zachodzi $X \succeq Y$.

- 20. [M22] Udowodnij, że jeżeli u i v są węzłami lasu, to węzeł u jest przodkiem węzła v wtedy i tylko wtedy, gdy u poprzedza v w porządku preorder, i u następuje po v w porządku postorder.

21. [25] Algorytm D przeprowadza różniczkowanie dla operatorów dwuargumentowych, jednoargumentowych i bezargumentowych, czyli drzew, w których węzły mają stopnie 2, 1 i 0. Opis algorytmu nie mówi jednak, w jaki sposób obsługuwać operatory trójargumentowe i węzły o większym stopniu. (Na przykład w ćwiczeniu 17 sugerujemy wprowadzenie operatorów wieloargumentowych). Czy da się w prosty sposób rozszerzyć algorytm D, by obsługiwał operatory stopnia wyższego niż 2?

- 22. [M26] Jeśli T i T' są drzewami, to mówimy, że drzewo T można włożyć w T' , $T \subseteq T'$, jeśli istnieje różnowartościowa funkcja f przeprowadzająca węzły drzewa T na węzły drzewa T' , zachowująca porządek preorder i postorder. (Innymi słowy, u poprzedza v w porządku preorder w drzewie T wtedy i tylko wtedy, gdy $f(u)$ poprzedza $f(v)$ w porządku preorder w drzewie T' i analogicznie dla porządku postorder. Zobacz rysunek 25).



Rys. 25. Jedno drzewo włożone w drugie (zobacz ćwiczenie 22).

Dla drzewa T o więcej niż jednym węźle niech $l(T)$ oznacza skrajnie lewe poddrzewo korzenia drzewa T i niech $r(T)$ oznacza pozostałość drzewa T , tzn. T bez $l(T)$. Udowodnij, że T można włożyć w T' , jeśli (i) T ma tylko jeden węzeł lub (ii) zarówno T , jak i T' mają więcej niż jeden węzeł i albo $T \subseteq l(T')$, albo $T \subseteq r(T')$, albo ($l(T) \subseteq l(T')$ i $r(T) \subseteq r(T')$). Czy twierdzenie odwrotne jest prawdziwe?

2.3.3. Inne reprezentacje drzew

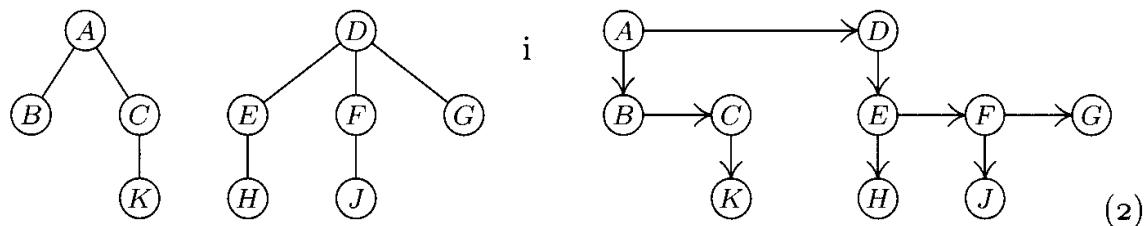
Jest wiele sposobów reprezentowania drzew w pamięci komputera, w poprzednim punkcie omówiliśmy jedną z nich – konkretnie reprezentację za pomocą drzew binarnych zgodnie z konwencją „z lewej dziecko, z prawej rodzeństwo”. Tak jak zwykle, właściwy wybór reprezentacji zależy w głównej mierze od tego, jakie operacje chcemy wykonywać na strukturze danych. W tym punkcie zajmiemy się kilkoma innymi sposobami reprezentowania drzew.

Przede wszystkim drzewa możemy reprezentować w tablicach. Podobnie jak w przypadku list liniowych tablice są odpowiednie wtedy, gdy interesuje nas zwarta reprezentacja drzewa, a jego kształt ani rozmiar podczas wykonania programu nie będzie się radykalnie zmieniał. W wielu przypadkach potrzebujemy po prostu stałych tablic o strukturze drzewiastej, do których można odwoływać się wewnątrz programu. Struktura drzewa w takiej tablicy zależy od sposobu jej wykorzystania.

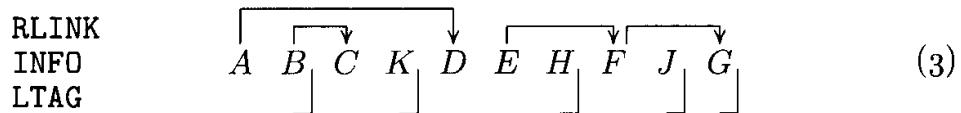
Najczęściej spotykana tablicowa reprezentacja lasów (i drzew) sprowadza się do pominięcia pola LLINK przy założeniu, że korzeń lewego poddrzewa znajduje się w kolejnej lokacji. Przypomnijmy sobie las z poprzedniego punktu

$$(A(B, C(K)), D(E(H), F(J), G)) \quad (1)$$

który rysowaliśmy następująco



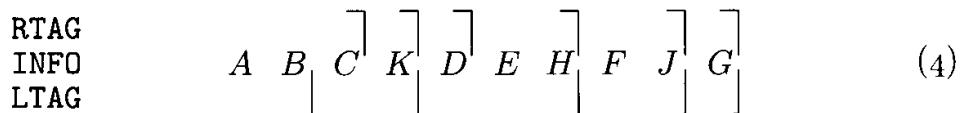
W reprezentacji tablicowej w porządku preorder węzły o polach INFO, RLINK i LTAG są ułożone w porządku preorder:



Niepuste dowiązania RLINK są oznaczone strzałkami, a $\text{LTAG} = 1$ (dla liści) jest oznaczone za pomocą „ \downarrow ”. Nie potrzebujemy LLINK, ponieważ dla każdego węzła to dowiązanie albo byłoby puste, albo wskazywałoby na kolejny węzeł w tablicy. Warto porównać (1) z (3).

Ta reprezentacja ma kilka ciekawych własności. Po pierwsze, wszystkie poddrzewa węzła występują bezpośrednio po nim, zatem wszystkie poddrzewa lasu występują w kolejnych spójnych blokach. [Porównaj z „zagnieżdzonymi nawiasami” w (1) i na rysunku 20(b).] Po drugie, zauważmy, że strzałki RLINK na schemacie (3) nigdy się nie przecinają. Ta własność zachodzi zawsze, ponieważ w drzewie binarnym wszystkie węzły między X i $\text{RLINK}(X)$ w porządku preorder należą do lewego poddrzewa X , zatem z tej części drzewa nie wyjdzie na zewnątrz żadna strzałka. Po trzecie, możemy zauważyc, że pole LTAG informujące, czy węzeł jest liściem, jest zbędne, ponieważ „ \downarrow ” występuje jedynie na końcu lasu oraz bezpośrednio przed każdą strzałką w dół.

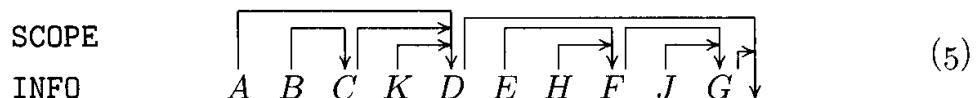
Z powyższych uwag wynika, że większość informacji w polu RLINK jest niepotrzebna. Do reprezentowania struktury wystarczą pola RTAG i LTAG. Stąd rysunek (3) można narysować, dysponując o wiele uboższymi informacjami:



Gdy przechodzimy (4) od lewej do prawej, pozycje, dla których $\text{RTAG} \neq "J"$, odpowiadają niepustym polom RLINK . Za każdym razem, mijając $\text{LTAG} = "J"$, należy dokończyć ostatnio zaczętą strzałkę RLINK . (Lokacje zaczętych strzałek RLINK można zatem trzymać na stosie). W istocie ponownie udowodniliśmy twierdzenie 2.3.1A.

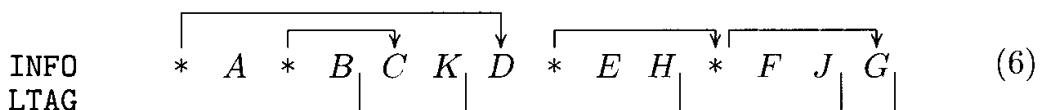
Z faktu, że RLINK lub LTAG są zbędne w schemacie (3), nie wynika dla nas nic pozytecznego. Jeżeli nie zamierzamy przeglądać sekwencyjnie całego lasu, to musielibyśmy wykonywać dodatkowe obliczenia, by wyznaczyć brakujące informacje. Z tego powodu często w pamięci przechowujemy wszystkie informacje ze schematu (3). Marnotrawstwu pamięci związanemu z dowiązaniami równymi Λ (w tym konkretnym lesie to ponad połowa dowiązań RLINK) można zaradzić następująco:

1) W polu RLINK każdego węzła zapisujemy adres pierwszej komórki pamięci następującej bezpośrednio po ostatniej komórce reprezentacji drzewa o korzeniu RLINK . Takie pole często nazywa się „SCOPE” (zakres) zamiast RLINK , ponieważ wskazuje granicę „wpływu” (potomków) każdego węzła. Zamiast (3) mamy w takim przypadku



Strzałki w dalszym ciągu się nie przecinają. Co więcej, warunek $\text{LTAG}(X) = "J"$ jest wyznaczony przez $\text{SCOPE}(X) = X + c$, gdzie c jest liczbą słów przypadającą na węzeł. Przykład wykorzystania takiej reprezentacji znajduje się w ćwiczeniu 2.4–12.

2) Można zmniejszyć rozmiar każdego węzła, usuwając pole RLINK i dodając specjalne węzły „wskaźnikowe” bezpośrednio przed węzłami, które miały niepusty wskaźnik RLINK :



Z pomocą „*” jest tu oznaczony węzeł wskaźnikowy. Zakładamy, że zawartość pola INFO pozwala w jakiś sposób odróżnić taki węzeł od zwykłych węzłów zawierających dane. Jeśli pola INFO i RLINK w schemacie (3) zajmują mniej więcej tyle samo miejsca, to stosując schemat (6), zaoszczędzimy pamięć, ponieważ liczba węzłów oznaczonych „*” jest zawsze mniejsza niż liczba węzłów nią nie oznaczonych. Reprezentacja (6) jest w pewien sposób analogiczna do sekwencji instrukcji maszynowych, gdzie węzły z „*” oznaczają rozkazy skoku warunkowego.

Inną reprezentację sekwencyjną (tablicę) analogiczną do (3) można otrzymać przez pozbycie się dowiązań RLINK . W tym przypadku umieszczaemy węzły drzewa w nowym porządku, który można nazwać *porządkiem rodzinnym*, ponieważ członkowie rodziny występują razem. Porządek rodzinny dla lasu określamy za pomocą następującej definicji rekurencyjnej:

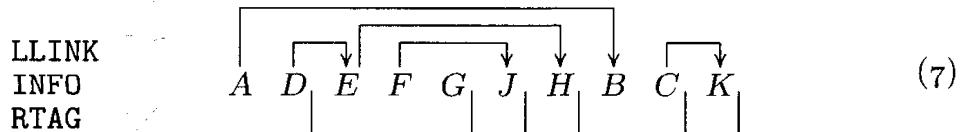
Odwiedź korzeń pierwszego drzewa.

Przejdź pozostałe drzewa (w porządku rodzinnym).

Przejdź poddrzewa korzenia pierwszego drzewa (w porządku rodzinnym).

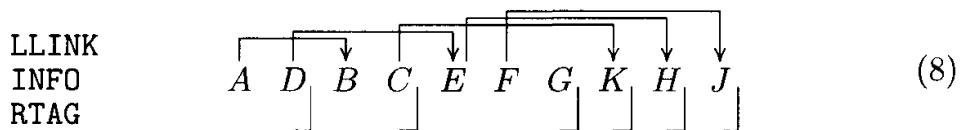
(Czytelnik zechce porównać tę definicję z definicjami porządków preorder i postorder z poprzedniego punktu. Porządek rodzinny jest tożsamy z porządkiem odwrotnym do postorder w odpowiadającym lasowi drzewie binarnym).

Reprezentacja tablicowa w porządku rodzinnym drzewa (2) wygląda następująco:



W tym przypadku pola **RTAG** oddzielają rodziny. W porządku rodzinnym najpierw występują korzenie wszystkich drzew w lesie, a następnie rodziny – kolejna rodzina jest rodziną ostatniego węzła, który się pojawił, a którego rodzina nie wystąpiła do tej pory. Wynika stąd, że strzałki **LLINK** nigdy się nie przetną; z analogicznych przyczyn występują inne własności reprezentacji tablicowej w porządku preorder.

Zamiast korzystać z porządku rodzinnego, moglibyśmy po prostu wypisać węzły poziomami od lewej do prawej. Taki porządek nazywamy „porządkiem poziomym” [zobacz G. Salton, *CACM* 5 (1962), 103–114], a dla lasu (2) *reprezentacja tablicowa w porządku poziomym* wygląda tak:



Jest podobna do (7), z tym że rodziny wypisujemy na zasadzie pierwszy-wchodzi-pierwszy-wychodzi, a nie ostatni-wchodzi-pierwszy-wychodzi. Schematy (7) albo (8) można uważać za naturalny drzewowy odpowiednik tablicowej reprezentacji list liniowych.

Czytelnik z łatwością zrozumie, w jaki sposób zaprojektować algorytm przechodzenia i analizowania drzewa w przedstawionych reprezentacjach tablicowych, ponieważ informacja **LLINK** i **RLINK** jest dostępna tak, jakbyśmy mieli do czynienia ze zwykłymi drzewami z dowiązaniem.

Inna metoda tablicowej reprezentacji drzewa, znana jako reprezentacja w porządku *postorder ze stopniami*, różni się nieco od powyższych metod. Umieszczamy węzły na liście w porządku postorder, ale zamiast dowiązań przechowujemy stopień węzła:

DEGREE	0	0	1	2	0	1	0	1	0	3
INFO	<i>B</i>	<i>K</i>	<i>C</i>	<i>A</i>	<i>H</i>	<i>E</i>	<i>J</i>	<i>F</i>	<i>G</i>	<i>D</i>

(9)

Dowód, że taka reprezentacja wyznacza strukturę drzewa, znajduje się w ćwiczeniu 2.3.2–10. Taki porządek przydaje się, gdy chcemy metodą wstępującą obliczyć wartość funkcji zdefiniowanej na węzłach drzewa. Przyjrzyjmy się następującemu algorytmowi.

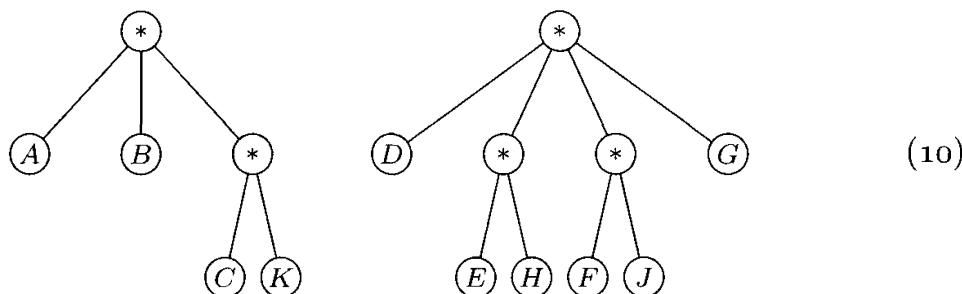
Algorytm F (*Obliczanie funkcji zdefiniowanej lokalnie na węzłach drzewa*). Przypuśćmy, że f jest funkcją zdefiniowaną na węzłach drzewa w taki sposób, że wartość f dla węzła x zależy wyłącznie od x oraz wartości f dla dzieci węzła x . Algorytm oblicza f dla każdego węzła w niepustym lesie, korzystając z pomocniczego stosu.

- F1.** [Inicjowanie] Weź pusty stos. Niech P wskazuje na pierwszy węzeł lasu w porządku postorder.
- F2.** [Obliczanie f] Przyjmij $d \leftarrow \text{DEGREE}(P)$. (Przy pierwszym wykonaniu tego kroku d będzie miało wartość zero. W ogólności zawsze w tym miejscu będzie spełniony warunek mówiący, że najwyższe d elementów na stosie, to $f(x_d), \dots, f(x_1)$, patrząc od wierzchołka stosu, gdzie x_1, \dots, x_d są dziećmi $\text{NODE}(P)$ w porządku od lewej do prawej). Oblicz $f(\text{NODE}(P))$, korzystając z wartości $f(x_d), \dots, f(x_1)$ znajdujących się na stosie.
- F3.** [Modyfikowanie stosu] Usuń d elementów z wierzchu stosu; następnie włóż na stos wartość $f(\text{NODE}(P))$.
- F4.** [Kolejny węzeł] Jeśli P jest ostatnim węzłem drzewa w porządku postorder, to zakończ wykonanie algorytmu. (Stos będzie w tym momencie zawierał $f(\text{root}(T_m)), \dots, f(\text{root}(T_1))$, patrząc od wierzchołka w dół, gdzie T_1, \dots, T_m są drzewami w lesie). W przeciwnym razie przypisz do P jego następnika w porządku postorder (w reprezentacji (9) będzie to po prostu $P \leftarrow P + c$) i wróć do kroku F2. ■

Poprawności algorytmu F dowodzimy przez indukcję względem rozmiaru przetwarzanych drzew (zobacz ćwiczenie 16). Ten algorytm ładząco przypomina metodę różniczkowania z poprzedniego rozdziału (algorytm 2.3.2D), polegającą na obliczaniu funkcji podobnego typu; zobacz ćwiczenie 3. Z tego samego pomysłu korzysta się w interpreterach przy obliczaniu wartości wyrażeń algebraicznych zapisanych w notacji przyrostkowej. Do tego tematu powrócimy w rozdziale 8. Zobacz też ćwiczenie 17, gdzie jest pokazana inna metoda podobna do algorytmu F.

Zapoznaliśmy się z wieloma tablicowymi reprezentacjami drzew i lasów. Zajmiemy się teraz reprezentacjami dowiązaniowymi tych struktur.

Pierwszy pomysł jest związany z przekształceniem schematu (3) w (6): usuwamy pola **INFO** ze wszystkich węzłów wewnętrznych i umieszczamy informacje w nowych liściach poniżej pierwotnego węzła. Na przykład drzewa ze schematu (2) przekształcilibyśmy na



Jak widać z nowych postaci, bez straty ogólności możemy założyć, że wszystkie pola **INFO** w strukturze drzewiastej występują w liściach. Z tego powodu w naturalnej reprezentacji za pomocą drzewa binarnego, opisanej w punkcie 2.3.2, pola **LLINK** i **INFO** wykluczają się wzajemnie i mogą dzielić to samo fizyczne pole węzła. Węzeł może mieć pola

LTAG	LLINK lub INFO	RLINK
-------------	-----------------------	--------------

gdzie znak **LTAG** służy do określenia, czy drugie pole jest dowiązaniem. (Warto porównać tę reprezentację na przykład z dwusłowowym formatem (10) w punkcie 2.3.2). Zmniejszając rozmiar **INFO** z 6 do 3 bajtów, możemy pomieścić każdy węzeł w jednym słowie. Zauważmy jednak, że w miejsce 10 węzłów mamy ich teraz 15. Reprezentacja (10) zajmuje 15 słów pamięci, podczas gdy (2) – 20. W reprezentacji (2) na pola **INFO** jest przeznaczonych 60 bajtów, a w (10) – 30 bajtów. W istocie stosując reprezentację (10), nie oszczędzamy na pamięci, jeżeli wcześniej nie marnowaliśmy pamięci zajmowanej przez pola **INFO**. Za usunięcie w schemacie (10) pól **LLINK** płacimy mniej więcej tą samą liczbą pól **RLINK** w dodanych węzłach. Różnice między powyższymi reprezentacjami omawiamy szczegółowo w ćwiczeniu 4.

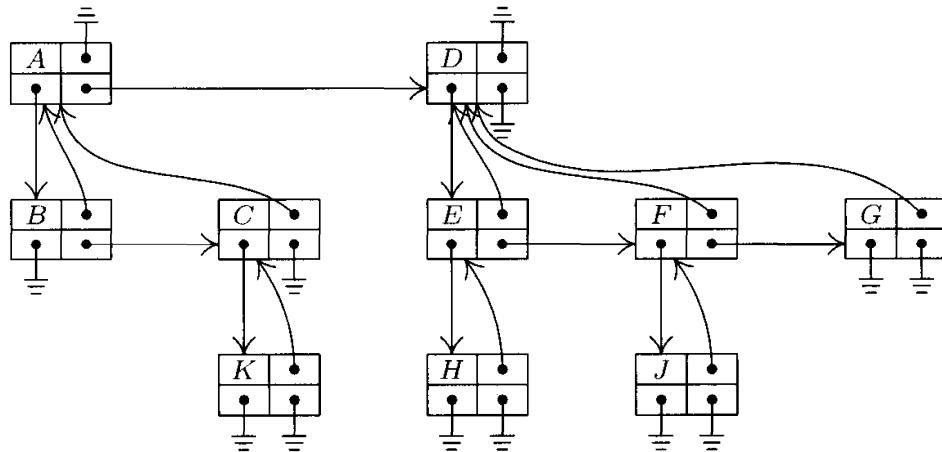
W standardowej reprezentacji drzewa za pomocą drzewa binarnego pole **LLINK** należałoby nazywać **LCHILD** (od ang. *child* – dziecko), ponieważ zawiera dowiązanie prowadzące od rodzica do skrajnie lewego dziecka. Skrajnie lewe dziecko nazywa się często „najmłodszym”, ponieważ łatwiej wstawić węzeł z lewej strony rodziny niż z prawej.

W praktyce w wielu algorytmach związanych ze strukturami drzewiastymi równie często co dowiązań prowadzących w dół drzewa potrzebujemy dowiązań prowadzących w górę. W góre drzewa możemy się poruszać, korzystając z fastygi, nie jest to jednak najbardziej wydajne rozwiązanie. Czasami możemy uzyskać lepsze wyniki, umieszczając w węźle drzewa trzecie dowiązanie **PARENT**, prowadzące do rodzica. W ten sposób otrzymujemy *drzewo z trzema dowiązaniami*, w którym każdy węzeł zawiera dowiązania **LCHILD**, **RLINK** i **PARENT**. Na rysunku 26 jest pokazany las (2) jako drzewo z trzema dowiązaniemi. Przykłady wykorzystania drzew z trzema dowiązaniemi pojawiają się w podrozdziale 2.4.

Jest jasne, że same dowiązania **PARENT** wystarczą do przedstawienia drzewa *zorientowanego* (lub lasu). Możemy przecież narysować schemat drzewa, jeżeli znamy wszystkie dowiązania prowadzące w górę. Każdy węzeł poza korzeniem ma dokładnie jednego rodzica, ale może mieć wiele dzieci; prościej jest zatem podawać dowiązania w górę niż w dół. Dlaczego zatem do tej pory nie zajęliśmy się tym rozwiązaniem? W większości przypadków posługiwanie się wyłącznie dowiązaniemi prowadzącymi w górę nie wystarcza. Na przykład trudno szybko stwierdzić, czy węzeł jest liściem albo znaleźć któreś dziecko węzła itd. Istnieje jednak bardzo ważne zastosowanie drzew o dowiązaniach prowadzących w górę: elegancki algorytm wyznaczania relacji równoważności, autorstwa M. J. Fischera i B. A. Gallera, którym się teraz zajmiemy.

INFO	PARENT
LCHILD	RLINK

Rys. 26. Drzewo z trzema dowiązaniami.



Relacja równoważności „ \equiv ” to relacja między elementami zbioru S , która dla dowolnych (niekoniecznie różnych) elementów x, y i z spełnia następujące warunki:

- i) Jeśli $x \equiv y$ i $y \equiv z$, to $x \equiv z$. (Przechodniość).
- ii) Jeśli $x \equiv y$, to $y \equiv x$. (Symetria).
- iii) $x \equiv x$. (Zwrotność).

(Powyższą definicję warto porównać z definicją częściowego porządku z punktu 2.2.3; relacje równoważności mają zupełnie inną naturę niż relacje częściowego porządku, choć dwa z definiujących je warunków są takie same). Przykładem relacji równoważności jest relacja „ $=$ ”, relacja przystawania (modulo m) na liczbach całkowitych, relacja podobieństwa drzew zdefiniowana w punkcie 2.3.1 itd.

Problem wyznaczania relacji równoważności można sformułować następująco. Dane są pary elementów, o których wiadomo, że są równoważne. Dla podanych dwóch elementów należy następnie stwierdzić, czy da się udowodnić ich równoważność, korzystając wyłącznie z informacji o równoważności elementów w danych parach. Przypuśćmy na przykład, że S jest zbiorem $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ oraz że dane pary to

$$1 \equiv 5, \quad 6 \equiv 8, \quad 7 \equiv 2, \quad 9 \equiv 8, \quad 3 \equiv 7, \quad 4 \equiv 2, \quad 9 \equiv 3. \quad (11)$$

Mogemy udowodnić, że $2 \equiv 6$, ponieważ $2 \equiv 7 \equiv 3 \equiv 9 \equiv 8 \equiv 6$. Nie mogemy jednak pokazać, że $1 \equiv 6$. W istocie pary (11) dzielą zbiór S na dwie klasy

$$\{1, 5\} \quad \text{oraz} \quad \{2, 3, 4, 6, 7, 8, 9\}, \quad (12)$$

takie że dwa elementy są równoważne wtedy i tylko wtedy, gdy należą do tej samej klasy. Nietrudno pokazać, że każda relacja równoważności dzieli zbiór S na rozłączne klasy (zwane klasami równoważności), takie że dwa elementy są równoważne wtedy i tylko wtedy, gdy należą do tej samej klasy.

Z powyższego powodu rozwiążanie problemu wyznaczania relacji równoważności można sprowadzić do znalezienia klas równoważności takich jak (12).

Możemy zacząć od sytuacji, w której każdy element należy do własnej jednoelementowej klasy:

$$\{1\} \{2\} \{3\} \{4\} \{5\} \{6\} \{7\} \{8\} \{9\} \quad (13)$$

Jeśli teraz napotkamy parę $1 \equiv 5$, to łączymy $\{1, 5\}$ w jedną klasę. Po przetworzeniu trzech pierwszych par $1 \equiv 5, 6 \equiv 8, 7 \equiv 2$ podział (13) będzie wyglądał tak

$$\{1, 5\} \{2, 7\} \{3\} \{4\} \{6, 8\} \{9\}. \quad (14)$$

Następnie para $9 \equiv 8$ spowoduje sklejenie $\{6, 8, 9\}$ itd.

Problem polega na znalezieniu takiej reprezentacji sytuacji (12), (13), (14) itp., by można było wydajnie realizować operacje łączenia klas oraz sprawdzania, czy dwa elementy należą do tej samej klasy. Przedstawiony poniżej algorytm korzysta w tym celu z drzew o dowiązaniach prowadzących w górę. Elementy S są węzłami zorientowanego lasu, a dwa elementy uważaemy za równoważne (względem wczytanych do tej pory par) *wtedy i tylko wtedy, gdy należą do tego samego drzewa*. Łatwo to sprawdzić, ponieważ dwa elementy należą do tego samego drzewa wtedy i tylko wtedy, gdy leżą poniżej tego samego korzenia. Ponadto łatwo połączyć dwa drzewa zorientowane, po prostu przyłączając jedno z nich do korzenia drugiego.

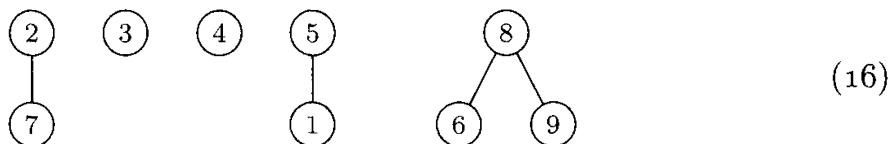
Algorytm E (*Wyznaczanie relacji równoważności*). Niech S będzie zbiorem liczb $\{1, 2, \dots, n\}$ i niech $\text{PARENT}[1], \text{PARENT}[2], \dots, \text{PARENT}[n]$ będą zmiennymi całkowitymi. Algorytm wczytuje zbiór par postaci (11) i w tablicy PARENT buduje zbiór drzew zorientowanych, taki że dwa elementy są w relacji równoważności wynikającej z wczytanych par wtedy i tylko wtedy, gdy należą do tego samego drzewa. (*Uwaga*: W podejściu bardziej ogólnym elementy S byłyby nazwani symbolicznymi, a nie liczbami od 1 do n ; podprogram wyszukujący, jak w rozdziale 6, odnajdowałby węzły odpowiadające elementom S , a PARENT byłoby polem w każdym węźle. Odpowiednie modyfikacje są dosyć oczywiste).

- E1. [Inicjowanie] Przyjmij $\text{PARENT}[k] \leftarrow 0$ dla $1 \leq k \leq n$. (Początkowo każde drzewo składa się wyłącznie z korzenia, porównaj (13)).
- E2. [Wczytanie nowej pary] Wczytaj z wejścia kolejną parę „ $j \equiv k$ ” równoważnych elementów. Zakończ wykonanie algorytmu, jeżeli nie ma więcej par.
- E3. [Znajdowanie korzeni] Jeżeli $\text{PARENT}[j] > 0$, to przyjmij $j \leftarrow \text{PARENT}[j]$ i powtórz ten krok. Jeżeli $\text{PARENT}[k] > 0$, to przyjmij $k \leftarrow \text{PARENT}[k]$ i powtórz ten krok. (Po tej operacji j i k zostaną przesunięte do korzeni dwóch drzew, które mamy połączyć. Para $j \equiv k$ była nadmiarowa, jeżeli w tym momencie mamy $j = k$).
- E4. [Połączenie drzew] Jeżeli $j \neq k$, to przyjmij $\text{PARENT}[j] \leftarrow k$. Wróć do kroku E2. ■

Czytelnik powinien sprawdzić działanie algorytmu dla danych wejściowych (11). Po przetworzeniu par $1 \equiv 5, 6 \equiv 8, 7 \equiv 2$ i $9 \equiv 8$ otrzymamy

$$\begin{array}{llllllllll} \text{PARENT}[k]: & 5 & 0 & 0 & 0 & 0 & 8 & 2 & 0 & 8 \\ k : & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \quad (15)$$

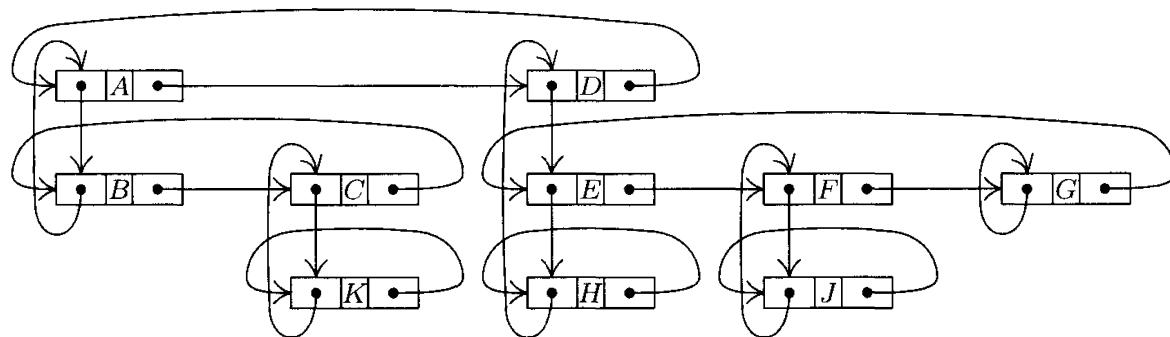
co stanowi reprezentację drzew



Od tego momentu przetwarzanie pozostałych par z (11) jest trochę ciekawsze; zobacz ćwiczenie 9.

Problem wyznaczania relacji równoważności pojawia się w wielu miejscach. Algorytmem E zajmiemy się powtórnie w punkcie 7.4.1 przy okazji omawiania spójności grafów. Bardziej ogólną wersję tego problemu – komplikację „deklaracji równoważności” w językach takich jak FORTRAN – omawiamy w ćwiczeniu 11.

Ale to jeszcze nie wszystkie sposoby reprezentowania drzew. Przypomnijmy sobie trzy podstawowe metody reprezentowania list liniowych, omówione w podrozdziale 2.2: reprezentacja prosta z kończącym dowiązaniem Λ , lista cykliczna i lista dwukierunkowa. Niesfastrygowane drzewo binarne odpowiada prostej reprezentacji listy względem dowiązań LLINK i RLINK. Korzystając z powyższych trzech sposobów reprezentowania list, można uzyskać osiem różnych reprezentacji drzewa binarnego, stosując je niezależnie do dowiązań LLINK i RLINK. Na przykład na rysunku 27 jest pokazana reprezentacja powstała w wyniku zastosowania reprezentacji cyklicznej na obu dowiązaniach. Taka reprezentacja nosi nazwę *struktury pierścieniowej*. Struktury pierścieniowe okazały się bardzo elastyczne w wielu praktycznych zastosowaniach. Wybór właściwej reprezentacji zależy, jak zawsze, od charakterystyk operacji wstawiania, usuwania i przechodzenia, które zamierzamy wykonywać na strukturze. Czytelnik, który śledził prezentowane do tej pory przykłady, nie będzie miał trudności ze zrozumieniem, jak należy posługiwać się każdą z tych reprezentacji.

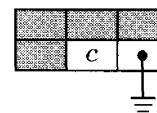
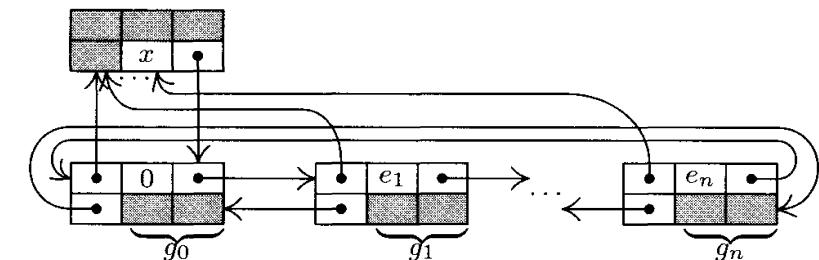
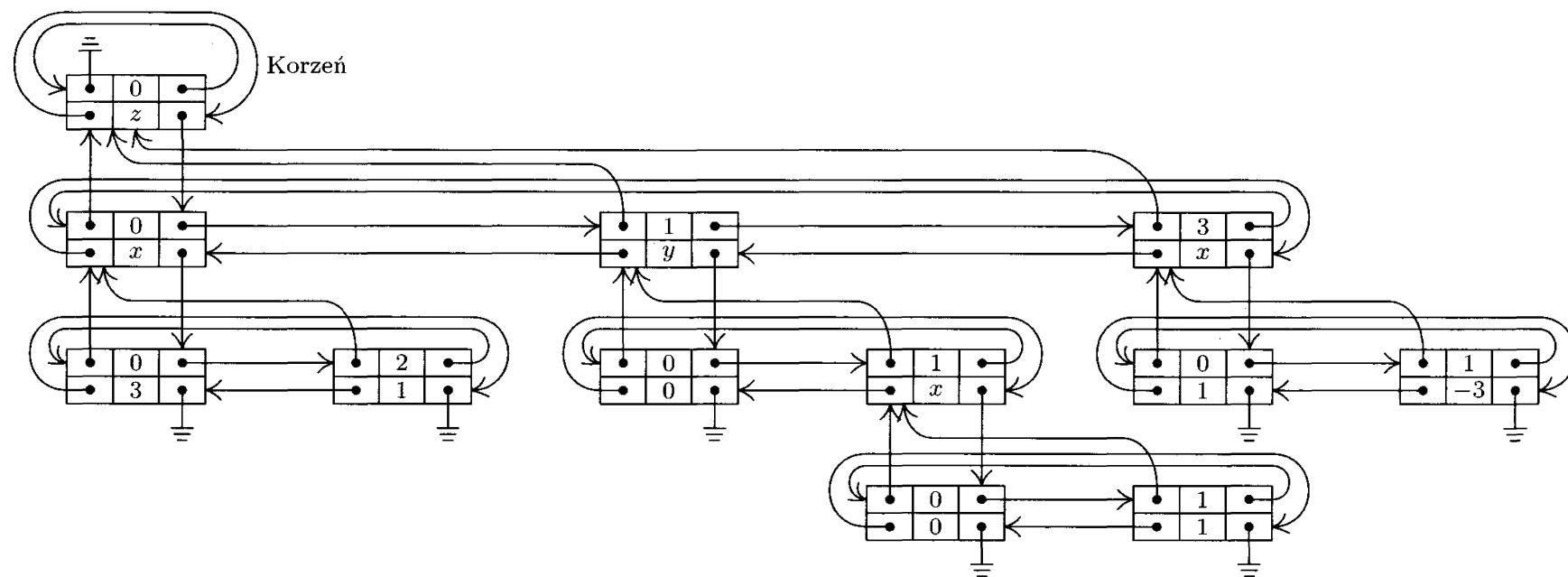


Rys. 27. Struktura pierścieniowa.

Zamknijmy ten punkt przykładem zastosowania nieco zmodyfikowanej dwukierunkowej struktury pierścieniowej do problemu, którym zajmowaliśmy się już wcześniej: do arytmetyki wielomianów. Algorytm 2.2.4A dodaje dwa wielomiany reprezentowane za pomocą list cyklicznych; inne algorytmy z punktu 2.2.4 wykonują inne działania na wielomianach. Założyliśmy tam jednak, że w wielomianach występują co najwyżej trzy zmienne. W przypadku wielu zmiennych zazwyczaj wygodniej posługiwać się drzewem zamiast listą liniową.

(a) Pola

UP	EXP	RIGHT
LEFT	CV	DOWN

(b) Wielomian = c (stała)(c) Wielomian = $g_0 + g_1 x^{e_1} + g_2 x^{e_2} + \dots + g_n x^{e_n}$ (d) Przykład: $3 + x^2 + xyz + z^3 - 3xz^3$ 

Rys. 28. Reprezentacja wielomianów za pomocą dowiązań czterokierunkowych. Zaciemnione pola oznaczają, że w danym kontekście zawarta w nich informacja jest nieistotna.

Wielomian albo jest równy stałej, albo ma postać

$$\sum_{0 \leq j \leq n} g_j x^{e_j},$$

gdzie x jest zmienną, $n > 0$, $0 = e_0 < e_1 < \dots < e_n$, a g_0, \dots, g_n są wielomianami zmiennych, które w alfabetie występują wcześniej niż x ; g_1, \dots, g_n są niezerowe. Ta rekurencyjna definicja wielomianu nadaje się do reprezentowania w postaci drzewiastej pokazanej na rysunku 28. Każdy węzeł ma sześć pól, które na komputerze MIX mieszczą się w trzech słowach:

+	0	LEFT	RIGHT	
+	EXP	UP	DOWN	(17)
		CV		

LEFT, **RIGHT**, **UP** i **DOWN** są dowiązaniem; **EXP** jest liczbą całkowitą reprezentującą wykładowik; **CV** jest albo stałą (współczynnikiem), albo literową nazwą zmiennej. Dla korzenia $UP = \Lambda$, $EXP = 0$, $LEFT = RIGHT = *$ (dowiązanie do samego siebie).

Poniższy algorytm zawiera operacje przechodzenia, wstawiania i usuwania z takiego drzewa z czterema dowiązaniem, warto więc przestudiować go uważnie.

Algorytm A (*Dodawanie wielomianów*). Algorytm dodaje polynomiał (P) do polynomiału (Q) przy założeniu, że P i Q są dowiązaniemami korzeni rozłącznych drzew reprezentujących wielomiany wg schematu pokazanego na rysunku 28. Po wykonaniu algorytmu polynomiał (P) pozostanie niezmieniony, a polynomiał (Q) będzie sumą wielomianów.

- A1.** [Sprawdzanie typu wielomianu] Jeśli $\text{DOWN}(P) = \Lambda$ (tj. jeśli P wskazuje na stałą), to zero lub więcej razy wykonaj $Q \leftarrow \text{DOWN}(Q)$, aż $\text{DOWN}(Q) = \Lambda$ i przejdź do kroku A3. Jeśli $\text{DOWN}(P) \neq \Lambda$ oraz jeśli $\text{DOWN}(Q) = \Lambda$ lub jeśli $\text{CV}(Q) < \text{CV}(P)$, to przejdź do A2. W przeciwnym razie jeśli $\text{CV}(Q) = \text{CV}(P)$, przypisz $P \leftarrow \text{DOWN}(P)$, $Q \leftarrow \text{DOWN}(Q)$ i powtórz ten krok; jeśli $\text{CV}(Q) > \text{CV}(P)$, to przyjmij $Q \leftarrow \text{DOWN}(Q)$ i powtórz ten krok. (W kroku A1 albo znajdujemy dwa wyrazy podobne albo stwierdzamy, że do bieżącego fragmentu wielomianu $\text{polynomial}(Q)$ należy wstawić nową zmienną).

A2. [Wstawianie poniżej] Przyjmij $R \leftarrow \text{AVAIL}$, $S \leftarrow \text{DOWN}(Q)$. Jeśli $S \neq \Lambda$, to $\text{UP}(S) \leftarrow R$, $S \leftarrow \text{RIGHT}(S)$ i jeśli $\text{EXP}(S) \neq 0$, to powtarzaj tę operację, aż $\text{EXP}(S) = 0$. Przyjmij $\text{UP}(R) \leftarrow Q$, $\text{DOWN}(R) \leftarrow \text{DOWN}(Q)$, $\text{LEFT}(R) \leftarrow R$, $\text{RIGHT}(R) \leftarrow R$, $\text{CV}(R) \leftarrow \text{CV}(Q)$ i $\text{EXP}(R) \leftarrow 0$. Na koniec przyjmij $\text{CV}(Q) \leftarrow \text{CV}(P)$ i $\text{DOWN}(Q) \leftarrow R$ oraz powróć do A1. (Wstawiliśmy sztuczny wielomian zerowy bezpośrednio pod $\text{NODE}(Q)$, który odpowiada wielomianowi z drzewa P. Operacje na dowiązaniach przeprowadzane w tym kroku są dosyć proste i można je łatwo zrozumieć, rysując schematy „przed i po”, opisane w punkcie 2.2.3).

A3. [Znalezienie pasujących wielomianów] (W tym miejscu P i Q wskazują na stałe w wyrazach podobnych, możemy wykonać dodawanie). Przyjmij

$CV(Q) \leftarrow CV(Q) + CV(P)$. Jeśli ta suma jest równa zero i jeśli $EXP(Q) \neq 0$, przejdź do kroku A8. Jeśli $EXP(Q) = 0$, to przejdź do kroku A7.

- A4. [Przesunięcie w lewo] (Po dodaniu dwóch wyrazów szukamy następnej pary do dodania). Przyjmij $P \leftarrow LEFT(P)$. Jeśli $EXP(P) = 0$, to przejdź do kroku A6. W przeciwnym razie jeden lub więcej razy wykonaj $Q \leftarrow LEFT(Q)$, aż $EXP(Q) \leq EXP(P)$. Jeśli po tym $EXP(Q) = EXP(P)$, to wróć do kroku A1.
- A5. [Wstawianie z prawej] Przyjmij $R \leftarrow AVAIL$. Następnie przyjmij $UP(R) \leftarrow UP(Q)$, $DOWN(R) \leftarrow \Lambda$, $CV(R) \leftarrow 0$, $LEFT(R) \leftarrow Q$, $RIGHT(R) \leftarrow RIGHT(Q)$, $LEFT(RIGHT(R)) \leftarrow R$, $RIGHT(Q) \leftarrow R$, $EXP(R) \leftarrow EXP(P)$ i $Q \leftarrow R$. Wróć do kroku A1. (Na bieżącym poziomie, tuż na prawo od $NODE(Q)$, musimy wstawić nowy wyraz, odpowiadający wykładownikowi $polynomial(P)$. Podobnie jak w kroku A2 schematy „przed i po” ułatwiają zrozumienie operacji na dowiązaniach).
- A6. [Powrót do góry] (Przeszliśmy cały poziom struktury pierścieniowej stanowiącej reprezentację $polynomial(P)$). Przyjmij $P \leftarrow UP(P)$.
- A7. [Przesunięcie Q w górę na odpowiedni poziom] Jeśli $UP(P) = \Lambda$, to przejdź do kroku A11; w przeciwnym przypadku zero lub więcej razy przypisz $Q \leftarrow UP(Q)$, aż $CV(UP(Q)) = CV(UP(P))$. Powróć do kroku A4.
- A8. [Usuwanie wyrazu zerowego] Przyjmij $R \leftarrow Q$, $Q \leftarrow RIGHT(R)$, $S \leftarrow LEFT(R)$, $LEFT(Q) \leftarrow S$, $RIGHT(S) \leftarrow Q$ i $AVAIL \leftarrow R$. (Wielomian się upraszcza, usuwamy element wiersza struktury reprezentującej $polynomial(Q)$). Jeśli teraz $EXP(LEFT(P)) = 0$ i $Q = S$, to przejdź do kroku A9; w przeciwnym razie wróć do kroku A4.
- A9. [Usuwanie wielomianu stałego] (Uproszczenia spowodowały, że wielomian zredukował się do stałej, zatem usuwamy cały wiersz struktury reprezentującej $polynomial(Q)$). Przyjmij $R \leftarrow Q$, $Q \leftarrow UP(Q)$, $DOWN(Q) \leftarrow DOWN(R)$, $CV(Q) \leftarrow CV(R)$ i $AVAIL \leftarrow R$. Przyjmij $S \leftarrow DOWN(Q)$; jeśli $S \neq \Lambda$, to przyjmij $UP(S) \leftarrow Q$, $S \leftarrow RIGHT(S)$, a jeśli $EXP(S) \neq 0$, to powtarzaj tę operację, aż $EXP(S) = 0$.
- A10. [Czy wykryto zero?] Jeśli $DOWN(Q) = \Lambda$, $CV(Q) = 0$ i $EXP(Q) \neq 0$, to przyjmij $P \leftarrow UP(P)$ i przejdź do kroku A8; w przeciwnym razie przejdź do kroku A6.
- A11. [Zakończenie] Zero lub więcej razy wykonaj $Q \leftarrow UP(Q)$, aż $UP(Q) = \Lambda$ (czyli ustawi Q w korzeniu drzewa). ■

W istocie powyższy algorytm będzie działał zdecydowanie szybciej niż algorytm 2.2.4A, jeśli $polynomial(P)$ będzie miał mało wyrazów, a $polynomial(Q)$ dużo, ponieważ w takim przypadku nie trzeba przechodzić całego wielomianu $polynomial(Q)$. Czytelnik zechce zasymulować ręcznie pracę algorytmu A przy dodawaniu wielomianu $xy - x^2 - xyz - z^3 + 3xz^3$ do wielomianu pokazanego na rysunku 28. (Dla tych danych nie ujawnia się akurat większa wydajność algorytmu, za to pojawiają się wszystkie rodzaje sytuacji, z którymi musi radzić sobie algorytm). Algorytmem A zajmujemy się w ćwiczeniach 12 i 13.

Nie twierdzimy, że reprezentacja pokazana na rysunku 28 jest najlepszą reprezentacją wielomianów o wielu zmiennych. W rozdziale 8 zajmiemy się inną reprezentacją wraz z algorytmami arytmetycznymi korzystającymi z pomocniczego stosu; ta reprezentacja będzie konstrukcyjnie o wiele prostsza. Autor pragnął pokazać Czytelnikowi algorytm A głównie ze względu na występujące w nim typowe operacje na drzewach z wieloma dowiązaniami.

ĆWICZENIA

- ▶ 1. [20] Jeżeli w reprezentacji tablicowej w porządku poziomym, jak (8), mielibyśmy wyłącznie pola LTAG, INFO i RTAG (bez LLINK), to czy dałoby się odtworzyć pola LLINK? (Innymi słowy, czy pola LLINK są nadmiarowe w (8), tak jak pola RLINK w (3)?)
- ▶ 2. [22] (Burks, Warren, Wright, Math. Comp. 8 (1954), 53–57) Drzewa (2) zapisane w porządku *preorder* ze stopniami wyglądająby tak:

DEGREE	2	0	1	0	3	1	0	1	0	0
INFO	A	B	C	K	D	E	H	F	J	G

[porównaj ze schematem (9), gdzie korzystaliśmy z porządku *postorder*]. Zaprojektuj algorytm analogiczny do algorytmu F, obliczający funkcję zdefiniowaną lokalnie na węzłach drzewa, polegający na przejściu powyższej reprezentacji drzewa od prawej do lewej.

- ▶ 3. [24] Zmodyfikuj algorytm 2.3.2D, tak by wykorzystywał pomysły zastosowane w algorytmie F; konkretnie: by pochodne obliczane jako wyniki tymczasowe umieszczały na stosie, zamiast dziwacznych działań w kroku D3. (Zobacz ćwiczenie 2.3.2–21). Stos można utrzymywać za pomocą pól RLINK w korzeniu każdej pochodnej.
- ▶ 4. [18] Drzewa (2) mają 10 węzłów, w tym 5 liści. Tradycyjna reprezentacja tych drzew za pomocą drzew binarnych wymaga 10 pól LLINK i 10 pól RLINK (po jednym w każdym węźle). Reprezentacja tych samych drzew w postaci (1), gdzie pola LLINK i INFO dzielą ten sam fragment pamięci, wymaga 5 pól LLINK i 15 pól RLINK. W każdym przypadku jest 10 pól INFO.

Porównaj liczbę pól LLINK i RLINK w obu reprezentacjach lasu o n węzłach, z których m to liście.

5. [16] Drzewo o trzech dowiązaniach, pokazane na rysunku 26, ma w każdym węźle dowiązania PARENT, LCHILD i RLINK. Tam, gdzie nie istnieje węzeł, do którego powinno prowadzić dowiązanie, lekką ręką wstawiamy Λ . Czy nie byłoby dobrym pomysłem dodanie *fastrygi*, czyli umieszczenie dodatkowo dowiązań fastrygujących w polach LCHILD i RLINK zawierających dowiązania puste, podobnie jak robiliśmy to w punkcie 2.3.1?

- ▶ 6. [24] Przypuśćmy, że węzły lasu *zorientowanego* mają po trzy pola z dowiązaniemi: PARENT, LCHILD i RLINK, ale tylko pola PARENT wyznaczają strukturę drzewa. Pola LCHILD mają wartość Λ , a pola RLINK tworzą listę liniową, łączącą węzły lasu w pewnym porządku. Zmienna dowiązaniowa FIRST wskazuje na pierwszy węzeł, a w ostatnim węźle RLINK = Λ .

Zaprojektuj algorytm, który przechodzi węzły i wypełnia pola LCHILD i RLINK zgodnie z zawartością dowiązań PARENT, budując poprawne drzewo z trzema dowiązaniemi, według schematu z rysunku 26. Algorytm powinien ponadto modyfikować zmienną FIRST, tak by wskazywała na korzeń pierwszego drzewa.

- ▶ 7. [15] Jakie klasy otrzymalibyśmy w (12), jeżeli w (11) nie występowałaby para $9 \equiv 3$?

8. [15] Algorytm E buduje strukturę drzewiastą reprezentującą podane pary elementów równoważnych, ale w tekście nie jest powiedziane, jak należy z niej korzystać. Zaprojektuj algorytm, który odpowiada na pytanie: „Czy $j \equiv k$?” przy założeniu, że $1 \leq j \leq n$, $1 \leq k \leq n$ oraz że za pomocą algorytmu E w tablicy PARENT zbudowano odpowiednią strukturę dla pewnego zbioru par.

9. [20] W postaci analogicznej do tablicy (15) i diagramu (16) zapisz zawartość struktury danych po przetworzeniu przez algorytm E wszystkich par (11) od lewej do prawej.

10. [28] W najgorszym przypadku dla n par algorytm E może wymagać liczby kroków rzędu n^2 . Pokaż, w jaki sposób można zmodyfikować algorytm, by przypadek najgorszy nie był aż tak zły.

- 11. [24] (*Deklaracje równoważności*). Wiele języków kompliwanych, między innymi FORTRAN, pozwala deklarować tablice, których reprezentacje nakładają się w pamięci. Programista podaje zbiór deklaracji postaci „ $X[j] \equiv Y[k]$ ”, co oznacza, że zmiennej $X[j+s]$ należy przydzielić tę samą lokację, co zmiennej $Y[k+s]$, dla dowolnego s . Dla każdej zmiennej tablicowej znany jest także dopuszczalny zakres indeksów: „ $\text{ARRAY } X[l:u]$ ” oznacza, że należy zarezerwować miejsce dla pozycji $X[l]$, $X[l+1]$, \dots , $X[u]$. Dla każdej klasy równoważności zmiennych kompilator rezerwuje jak najmniejszy blok kolejnych lokacji, zawierający wszystkie pozycje tablicy z podanego przedziału dopuszczalnych indeksów.

Przypuśćmy na przykład, że mamy $\text{ARRAY } X[0:10]$, $\text{ARRAY } Y[3:10]$, $\text{ARRAY } A[1:1]$ i $\text{ARRAY } Z[-2:0]$ oraz deklaracje równoważności $X[7] \equiv Y[3]$, $Z[0] \equiv A[0]$ i $Y[1] \equiv A[8]$. Na te zmienne musimy zarezerwować 20 kolejnych lokacji

	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
•	•	•	•	•	•	•	•	•	•	•	•
Z_{-2}	Z_{-1}	Z_0	A_1					Y_3	Y_4	Y_5	Y_6

(Lokacja za $A[1]$ nie jest dostępna z uwagi na zakresy indeksów, ale i tak musimy ją zarezerwować).

Celem tego ćwiczenia jest wprowadzenie modyfikacji do algorytmu E, tak by można go było zastosować do bardziej ogólnego problemu, opisanego powyżej. Przypuśćmy, że piszemy kompilator języka z deklaracjami równoważności. W strukturach danych kompilatora każda tablica komplowanego programu jest reprezentowana za pomocą jednego elementu o polach NAME, PARENT, DELTA, LBD i UBD. Zakładamy, że kompilator przeanalizował deklaracje ARRAY, zatem jeśli w programie występuje „ $\text{ARRAY } X[l:u]$ ” i jeśli P wskazuje na element reprezentujący X, to

$$\begin{aligned} \text{NAME}(P) &= „X”, & \text{PARENT}(P) &= \Lambda, & \text{DELTA}(P) &= 0, \\ \text{LBD}(P) &= l, & \text{UBD}(P) &= u. \end{aligned}$$

Zadanie polega na zaprojektowaniu algorytmu przetwarzającego deklaracje równoważności w ten sposób, że po jego wykonaniu

$\text{PARENT}(P) = \Lambda$ oznacza, że lokacje $X[\text{LBD}(P)], \dots, X[\text{UBD}(P)]$ należy zarezerwować na daną klasę równoważności;

$\text{PARENT}(P) = Q \neq \Lambda$ oznacza, że lokacja $X[k]$ ma być tożsama z $Y[k + \text{DELTA}(P)]$, gdzie $\text{NAME}(Q) = „Y”$.

Jeśli na przykład przed wyznaczeniem klas równoważności struktura danych kompilatora wygląda tak:

P	NAME(P)	PARENT(P)	DELTA(P)	LBD(P)	UBD(P)
α	X	Λ	0	0	10
β	Y	Λ	0	3	10
γ	A	Λ	0	1	1
δ	Z	Λ	0	-2	0

To po ich wyznaczeniu struktura może wyglądać następująco:

α	X	Λ	*	-5	14
β	Y	α	4	*	*
γ	A	δ	0	*	*
δ	Z	α	-3	*	*

(,,*" oznacza informację nieistotną).

Zaprojektuj algorytm realizujący powyższe przekształcenie. Załóż, że dane wejściowe mają postać czwórek (P, j, Q, k) , oznaczających „ $X[j] \equiv Y[k]$ ”, gdzie $NAME(P) = „X”$, a $NAME(Q) = „Y”$. Pamiętaj o sprawdzeniu, czy deklaracje równoważności nie są sprzeczne; na przykład $X[1] \equiv Y[2]$ jest sprzeczna z $X[2] \equiv Y[1]$.

12. [21] Na początku wykonania algorytmu A zmienne P i Q wskazują na korzenie dwóch drzew. Niech P_0 i Q_0 oznaczają wartości P i Q przed wykonaniem algorytmu A. (a) Czy zawsze po zakończeniu wykonania algorytmu Q_0 jest adresem drzewa reprezentującego sumę wielomianów? (b) Czy zawsze po zakończeniu wykonania algorytmu P i Q mają wartości P_0 i Q_0 ?

► **13.** [M29] Podaj nieformalny dowód faktu, że na początku kroku A8 algorytmu A zawsze mamy $EXP(P) = EXP(Q)$ i $CV(UP(P)) = CV(UP(Q))$. (Jest to fakt bardzo istotny dla zrozumienia algorytmu).

14. [40] Podaj formalny dowód (albo kontrprzykład) poprawności algorytmu A.

15. [40] Zaprojektuj algorytm obliczający iloczyn dwóch wielomianów w reprezentacji pokazanej na rysunku 28.

16. [M24] Udowodnij poprawność algorytmu F.

► **17.** [25] Algorytm F oblicza funkcję zdefiniowaną lokalnie na węzłach drzewa za pomocą metody wstępującej, tj. najpierw oblicza się funkcję dla dzieci węzła, a potem dla samego węzła. Funkcja f zdefiniowana zstępująco na węzłach drzewa to funkcja, której wartość dla węzła x zależy jedynie od x oraz wartości f dla rodzica węzła x . Zaprojektuj algorytm analogiczny do algorytmu F, który korzystając z pomocniczego stosu, oblicza zstępującą funkcję f dla każdego węzła drzewa. (Jak w przypadku algorytmu F, Twój algorytm powinien działać efektywnie na drzewach w reprezentacji *postorder* ze stopniami, jak (9)).

► **18.** [28] Zaprojektuj algorytm, który dla dwóch danych tablic $INFO1[j]$ i $RLINK[j]$, $1 \leq j \leq n$, odpowiadających tablicowej reprezentacji preorder, buduje tablice $INFO2[j]$ i $DEGREE[j]$, $1 \leq j \leq n$, odpowiadające reprezentacji postorder ze stopniami. Na przykład, zgodnie z (3) i (9) Twój algorytm powinien przekształcić

j	1	2	3	4	5	6	7	8	9	10
INFO1[j]	A	B	C	K	D	E	H	F	J	G
RLINK[j]	5	3	0	0	0	8	0	10	0	0

w

INFO2[j]	B	K	C	A	H	E	J	F	G	D
DEGREE[j]	0	0	1	2	0	1	0	1	0	3

19. [M27] Zamiast dowiązań SCOPE w schemacie (5) moglibyśmy po prostu wypisać liczbę potomków każdego węzła, w porządku preorder:

DESC	3	0	1	0	5	1	0	1	0	0
INFO	A	B	C	K	D	E	H	F	J	G

Niech $d_1 d_2 \dots d_n$ będą ciągiem liczb potomków węzłów lasu, wyznaczonym w powyższy sposób.

- a) Pokaż, że $k + d_k \leq n$ dla $1 \leq k \leq n$ oraz że $k \leq j \leq k + d_k$ pociąga za sobą $j + d_j \leq k + d_k$.
- b) Udowodnij w drugą stronę, że jeśli $d_1 d_2 \dots d_n$ jest ciągiem nieujemnych liczb całkowitych spełniających warunki (a), to jest on ciągiem liczb będących potomkami węzłów lasu.
- c) Przypuśćmy, że $d_1 d_2 \dots d_n$ i $d'_1 d'_2 \dots d'_n$ są ciągami liczb będących potomkami węzłów dwóch lasów. Udowodnij, że istnieje trzeci las, dla którego ciąg liczb będących potomkami węzłów ma postać

$$\min(d_1, d'_1) \min(d_2, d'_2) \dots \min(d_n, d'_n).$$

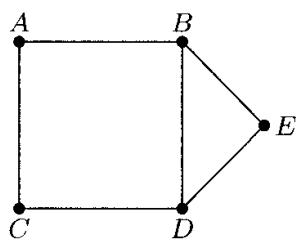
2.3.4. Podstawowe własności matematyczne drzew

Wieloletnie intensywne badania matematyczne zaowocowały odkryciem wielu własności drzew. W tym punkcie zajmiemy się przeglądem matematycznej teorii drzew, która nie tylko umożliwia nam wejście w naturę struktur drzewiastych, ale ma także istotne zastosowania w konstrukcji i analizie algorytmów komputerowych.

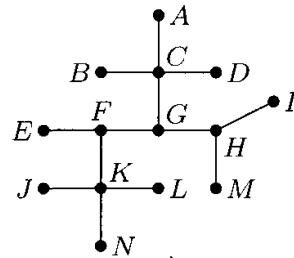
Czytelnikom niezainteresowanym matematyką radzimy pominąć tekst aż do podpunktu 2.3.4.5, w którym omawiamy zagadnienia potrzebne do dalszej lektury.

Prezentowany materiał pochodzi w większości z obszerniejszej dziedziny matematyki – teorii grafów. Niestety, w tej dziedzinie prawdopodobnie nigdy nie ustali się standardowa terminologia, zatem autor postąpił zgodnie z praktyką powszechną we współczesnych książkach dotyczących teorii grafów, tj. posłużył się słowami, które są podobne, acz nie identyczne z terminami występującymi w innych książkach o teorii grafów. W tym rozdziale (a w istocie w całej książce) autor starał się, aby na określenie istotnych pojęć wybrać słowa krótkie, opisowe, które nie kolidują zbyt poważnie z powszechną terminologią. Przyjęta nomenklatura faworyzuje przy tym zastosowania teorii w informatyce. Inżynier elektronik będzie wolałby nazywać „drzewem” to, co my nazywamy „drzewem wolnym”. Jednak nam o wiele wygodniej jest posługiwać się krótkim terminem „drzewo” na określenie pojęcia często pojawiającego się w literaturze informatycznej oraz mającego liczne zastosowania w praktyce programowania. Jeśli chcielibyśmy trzymać się terminologii niektórych autorów piszących o teorii grafów, musielibyśmy mówić „skończone, etykietowane, ukorzenione drzewo uporządkowane” zamiast po prostu „drzewo” i „topologiczne drzewo uporządkowane drugiego stopnia” zamiast „drzewo binarne”!

2.3.4.1. Drzewa wolne. *Graf* definiujemy jako zbiór punktów (zwanych *wierzchołkami*) oraz zbiór odcinków (nazywanych *krawędziami*) łączących pewne pary różnych wierzchołków. Każdą parę wierzchołków łączy co najwyżej jedna



Rys. 29. Graf.



Rys. 30. Drzewo wolne.

krawędź. Wierzchołki nazywamy *sąsiednimi*, jeżeli w grafie występuje łącząca je krawędź. Jeśli V i V' są wierzchołkami oraz jeśli $n \geq 0$, to mówimy, że (V_0, V_1, \dots, V_n) jest ścieżką długości n z V do V' , jeśli $V = V_0$, V_k sąsiaduje z V_{k+1} , dla $0 \leq k < n$, oraz $V_n = V'$. Mówimy, że ścieżka jest *prosta*, jeśli V_0, V_1, \dots, V_{n-1} są parami różne i jeśli V_1, \dots, V_{n-1}, V_n są parami różne. Graf jest *spójny*, jeśli między dowolnymi dwoma wierzchołkami grafu istnieje ścieżka. *Cykł* w grafie jest ścieżką prostą długości co najmniej trzy, prowadzącą od wierzchołka do niego samego.

Powyzsze definicje są zilustrowane na rysunku 29, pokazującym spójny graf o pięciu wierzchołkach i sześciu krawędziach. Wierzchołek C sąsiaduje z A , ale nie z B . Od B do C prowadzą dwie ścieżki o długości dwa, konkretnie (B, A, C) i (B, D, C) . W grafie jest kilka cykli, m.in. (B, D, E, B) .

Drzewo wolne lub „drzewo nieukorzenione” (rysunek 30) definiujemy jako graf spójny bez cykli. Ta definicja jest poprawna zarówno dla grafów skończonych, jak i nieskończonych, chociaż w praktyce informatycznej bardziej interesują nas drzewa skończone. Drzewa wolne można definiować na wiele sposobów; niektóre z nich pojawiają się w znanym twierdzeniu:

Twierdzenie A. Jeśli G jest grafem, to następujące warunki są równoważne:

- G jest drzewem wolnym.
- G jest spójny, ale usunięcie dowolnej krawędzi rozspójni go.
- Jeśli V i V' są różnymi wierzchołkami G , to istnieje dokładnie jedna ścieżka prosta z V do V' .

Ponadto, jeśli G jest skończony i zawiera dokładnie $n > 0$ wierzchołków, poniższe warunki są równoważne dowolnemu z warunków (a), (b) i (c):

- G nie zawiera cykli i ma $n - 1$ krawędzi.
- G jest spójny i ma $n - 1$ krawędzi.

Dowód. Warunek (a) pociąga za sobą (b), bo jeśli usuniemy krawędź $V — V'$, a mimo to G pozostanie spójny, to musi istnieć ścieżka prosta (V, V_1, \dots, V') długości co najmniej dwa – zobacz ćwiczenie 2 – a wtedy w grafie G byłby cykl (V, V_1, \dots, V', V) .

Warunek (b) implikuje (c), ponieważ istnieje przynajmniej jedna ścieżka prosta z V do V' . Jeśli byłyby dwie takie ścieżki (V, V_1, \dots, V') i (V, V'_1, \dots, V') , to moglibyśmy znaleźć najmniejsze k , dla którego $V_k \neq V'_k$. Usunięcie krawędzi $V_{k-1} — V_k$ nie spowodowałoby rozspojnienia grafu, ponieważ wciąż mielibyśmy

ścieżkę $(V_{k-1}, V'_k, \dots, V', \dots, V_k)$ od V_{k-1} do V_k , która nie zawiera usuniętej krawędzi.

Warunek (c) pociąga za sobą znowu (a), ponieważ jeśli G zawiera cykl (V, V_1, \dots, V) , to istnieją dwie ścieżki proste z V do V_1 .

By pokazać, że warunki (d) i (e) są także równoważne warunkom (a), (b) i (c), udowodnimy najpierw twierdzenie pomocnicze: jeśli G jest dowolnym grafem skończonym bez cykli, który ma przynajmniej jedną krawędź, to istnieje przynajmniej jeden wierzchołek sąsiadujący z dokładnie jednym innym wierzchołkiem. Jest tak, ponieważ możemy znaleźć pewien wierzchołek V_1 oraz sąsiadujący z nim wierzchołek V_2 ; a dla $k \geq 2$ albo V_k sąsiaduje z V_{k-1} i żadnym innym, albo dodatkowo sąsiaduje z wierzchołkiem, który nazwiemy $V_{k+1} \neq V_{k-1}$. Ponieważ w grafie nie ma cykli, wierzchołki V_1, V_2, \dots, V_{k+1} muszą być parami różne, zatem nie możemy w nieskończoność napotykać „drugiego sąsiada”.

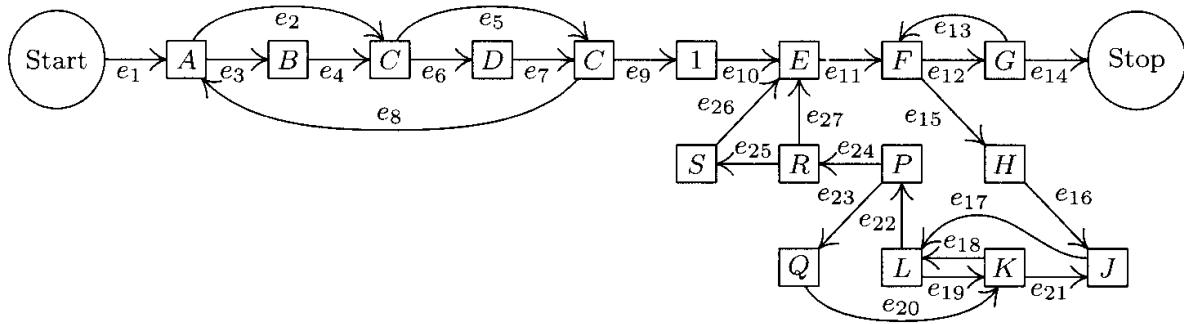
Przypuśćmy teraz, że G jest drzewem o $n > 1$ wierzchołkach. Niech V_n będzie wierzchołkiem sąsiadującym z dokładnie jednym innym wierzchołkiem V_{n-1} . Jeśli usuniemy z grafu V_n oraz krawędź $V_{n-1} — V_n$, to otrzymany graf G' jest drzewem, ponieważ na dowolnej ścieżce prostej w G wierzchołek V_n występował co najwyżej jako pierwszy lub ostatni element. Takie rozumowanie pozwala udowodnić (za pomocą indukcji względem n), że G ma $n - 1$ krawędzi; stąd warunek (a) implikuje (d).

Założymy, że G spełnia warunek (d) i niech V_n, V_{n-1}, G' oznaczają to, co w poprzednim akapicie. Wówczas graf G jest spójny, ponieważ V_n jest połączony z V_{n-1} , który (indukcja względem n) jest połączony ścieżkami prostymi ze wszystkimi innymi wierzchołkami grafu G' . Stąd warunek (d) implikuje (e).

Założymy na koniec, że G spełnia warunek (e). Jeśli G zawiera cykl, to możemy usunąć dowolną krawędź tego cyklu bez rozspojniania G . Możemy zatem kontynuować usuwanie krawędzi, aż otrzymamy spójny graf G' bez cykli o $n - 1 - k$ krawędziach. Ale z uwagi na fakt, że warunek (a) implikuje (d), musi być $k = 0$, tj. $G = G'$. ■

Pojęcie drzewa wolnego ma bezpośrednie zastosowanie w analizie algorytmów komputerowych. W punkcie 1.3.3 omawialiśmy zastosowanie pierwszego prawa Kirchhoffa do wyznaczania liczby wykonan poszczególnych kroków algorytmu. Odkryliśmy, że prawo Kirchhoffa nie umożliwia wyznaczenia do końca liczby wykonanych kroków, ale pozwala zmniejszyć liczbę niewiadomych. Stosując teorię drzew, możemy wyznaczyć liczbę tych niezależnych niewiadomych, a także znaleźć je w sposób systematyczny.

Prezentowaną metodę najłatwiej zrozumieć, analizując przykład. Zajmiemy się zatem przykładem, równolegle budując ogólną teorię. Na rysunku 31 jest pokazany abstrakcyjny schemat blokowy programu 1.3.3A, który analizowaliśmy w punkcie 1.3.3 za pomocą „prawa Kirchhoffa”. Każdy blok na rysunku 31 oznacza fragment obliczenia, a litera lub cyfra wewnątrz bloku – liczbę wykonan tego fragmentu podczas jednego wykonania programu (porównaj oznaczenia z punktu 1.3.3). Strzałki między blokami reprezentują możliwe skoki. Są one potykietowane symbolami e_1, e_2, \dots, e_{27} . Naszym celem jest znalezienie wszystkich



Rys. 31. Abstrakcyjny schemat blokowy programu 1.3.3A.

związków między wielkościami $A, B, C, D, E, F, G, H, J, K, L, P, Q, R$ i S , które wynikają z prawa Kirchhoffa. Jednocześnie mamy nadzieję wejrzenia w ogólną naturę problemów tego typu. (*Uwaga:* Na rysunku 31 już wprowadziliśmy pewne uproszczenia; na przykład blok między C i E ma etykietę „1”, a to jest już wniosek z prawa Kirchhoffa).

Niech E_j oznacza liczbę przejść po strzałce e_j podczas wykonania programu. Prawo Kirchhoffa mówi, że

$$\text{suma wejściowych wartości } E = \text{wartość w bloku} = \text{suma wyjściowych wartości } E; \quad (1)$$

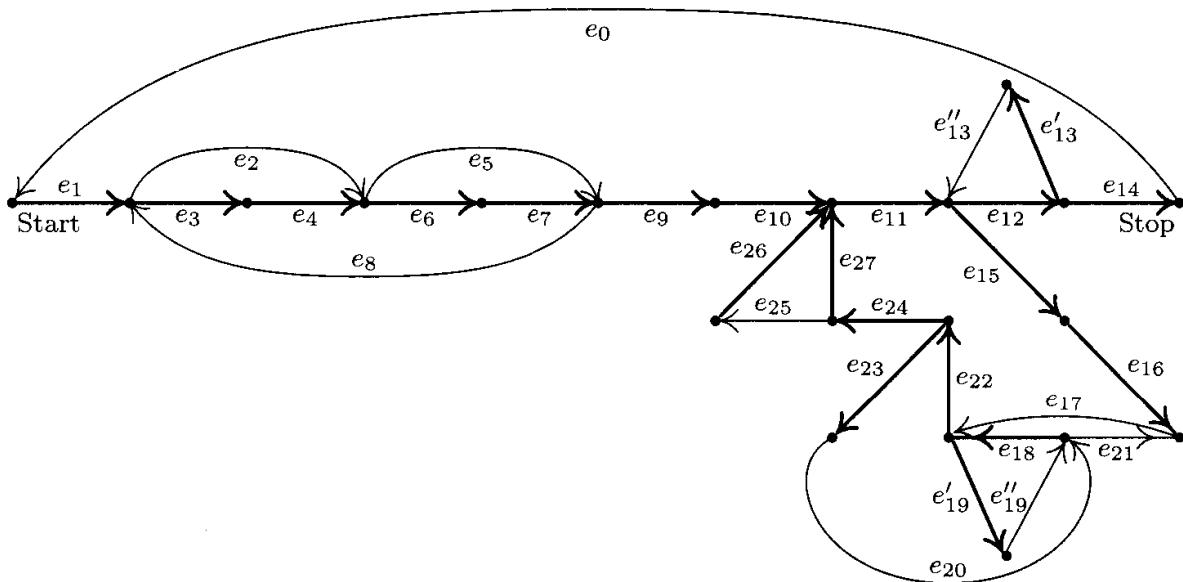
na przykład dla bloku o etykiecie K mamy

$$E_{19} + E_{20} = K = E_{18} + E_{21}. \quad (2)$$

W dalszej części analizy przyjmiemy, że niewiadomymi są E_1, E_2, \dots, E_{27} , a nie A, B, \dots, S .

Na schemat blokowy z rysunku 31 możemy teraz spojrzeć jak na graf. Odpowiadający mu graf G jest pokazany na rysunku 32. Bloki skurczyły się do punktów symbolizujących wierzchołki, a strzałki e_1, e_2, \dots stały się krawędziami grafu. (Ścisłe rzecz biorąc, dla grafu nie zakłada się, że krawędzie mają kierunki. Zwroty strzałek należy zignorować, gdy będziemy mówić o teoriografowych własnościach G . Stosując prawo Kirchhoffa, będziemy jednak zwracali uwagę na kierunek krawędzi). Dla wygody dodaliśmy krawędź e_0 od wierzchołka Stop do wierzchołka Start, dzięki czemu prawo Kirchhoffa jest spełnione dla każdego wierzchołka. Rysunek 32 zawiera także niewielką zmianę w stosunku do rysunku 31: dodaliśmy pomocniczy wierzchołek dzielący krawędź e_{13} na dwie części e'_{13} i e''_{13} , tak by spełniony był warunek z definicji grafu (co najwyżej jedna krawędź między wierzchołkami). W podobny sposób podzieliliśmy krawędź e_{19} . Analogiczne zmiany należałyby wprowadzić w sytuacji, gdy od jakiegoś bloku prowadziłaby strzałka do niego samego.

Niektóre krawędzie na rysunku 32 są narysowane grubszą linią. Te krawędzie składają się na *wolne poddrzewo* grafu, łączące wszystkie jego wierzchołki. Zawsze daje się znaleźć wolne poddrzewo grafów otrzymanych ze schematów blokowych, ponieważ takie grafy są spójne, a na mocy punktu (b) twierdzenia A, jeśli G jest spójny i nie jest drzewem wolnym, to możemy usunąć z niego krawędź, nie rozspojniając grafu. Ten proces można kontynuować, aż do momentu otrzymania drzewa. Inny algorytm wyznaczania wolnego poddrzewa opisujemy



Rys. 32. Graf odpowiadający rysunkowi 31 zawierający wolne poddrzewo.

w ćwiczeniu 6. W istocie zawsze możemy zacząć od usunięcia krawędzi e_0 (prowadzącej od wierzchołka Stop do wierzchołka Start). Będziemy zakładać, że e_0 nie należy do wybranego poddrzewa.

Niech G' będzie wolnym poddrzewem grafu G wyznaczonym w powyższy sposób. Weźmy teraz dowolną krawędź $V - V'$ grafu G , która *nie* należy do G' . Możemy zauważać istotny wniosek płynący z twierdzenia A: G' plus ta krawędź zawiera cykl. W istocie tworzymy w ten sposób *dokładnie jeden* cykl postaci (V, V', \dots, V) , ponieważ w G' istnieje dokładnie jedna ścieżka prosta z V' do V . Jeśli na przykład G' jest wolnym poddrzewem pokazanym na rysunku 23, to gdy dołożymy do niego krawędź e_2 , otrzymamy cykl wiodący przez e_2 , następnie (w kierunku przeciwnym do strzałek) przez e_4 i e_3 . Ten cykl można zapisać symbolicznie jako „ $e_2 - e_4 - e_3$ ”, gdzie znaki „+” i „-” oznaczają zwroty strzałek na kolejnych krawędziach, przez które prowadzi cykl.

Postępując analogicznie dla wszystkich krawędzi grafu nie należących do wolnego poddrzewa, otrzymamy zbiór tzw. *cykli podstawowych*, który w przypadku grafu i drzewa z rysunku 32 wygląda następująco:

$$\begin{aligned}
 C_0: & e_0 + e_1 + e_3 + e_4 + e_6 + e_7 + e_9 + e_{10} + e_{11} + e_{12} + e_{14}, \\
 C_2: & e_2 - e_4 - e_3, \\
 C_5: & e_5 - e_7 - e_6, \\
 C_8: & e_8 + e_3 + e_4 + e_6 + e_7, \\
 C''_{13}: & e''_{13} + e_{12} + e'_{13}, \\
 C_{17}: & e_{17} + e_{22} + e_{24} + e_{27} + e_{11} + e_{15} + e_{16}, \\
 C''_{19}: & e''_{19} + e_{18} + e'_{19}, \\
 C_{20}: & e_{20} + e_{18} + e_{22} + e_{23}, \\
 C_{21}: & e_{21} - e_{16} - e_{15} - e_{11} - e_{27} - e_{24} - e_{22} - e_{18}, \\
 C_{25}: & e_{25} + e_{26} - e_{27},
 \end{aligned} \tag{3}$$

Jest jasne, że krawędź e_j nie należąca do wolnego poddrzewa występuje w dokładnie jednym cyklu podstawowym C_j .

Zbliżamy się powoli do kulminacji. Każdy cykl podstawowy reprezentuje rozwiązanie równań Kirchhoffa. Na przykład rozwiązanie odpowiadające cyklowi C_2 to $E_2 = +1, E_4 = -1, E_3 = -1$ i zera na pozostałych E . Jest jasne, że przepływ wzdłuż cyklu w grafie zawsze spełnia warunek (1) prawa Kirchhoffa. Ponadto równania Kirchhoffa są „jednorodne”, zatem suma lub różnica rozwiązań (1) daje nowe rozwiązanie. Stąd możemy wnioskować, że wartości $E_0, E_2, E_5, \dots, E_{25}$ są *niezależne* w następującym sensie:

Jeśli x_0, x_2, \dots, x_{25} są dowolnymi liczbami rzeczywistymi (po jednym x_j dla każdej krawędzi e_j spoza wolnego poddrzewa G'), to istnieje rozwiązanie równań Kirchhoffa (1), takie że $E_0 = x_0, E_2 = x_2, \dots, E_{25} = x_{25}$. (4)

Takie rozwiązanie polega na obejściu x_0 razy cyklu C_0 , x_2 razy cyklu C_2 itd. Ponadto okazuje się, że wartości pozostałych zmiennych E_1, E_3, E_4, \dots są całkowicie *zależne* od wartości E_0, E_2, \dots, E_{25} :

Rozwiązanie opisane w (4) jest jedyne. (5)

Jest tak dlatego, że mając dwa rozwiązania równań Kirchhoffa, takie że $E_0 = x_0, \dots, E_{25} = x_{25}$, możemy odjąć jedno od drugiego, otrzymując rozwiązanie, w którym $E_0 = E_2 = E_5 = \dots = E_{25} = 0$. Ale wówczas *wszystkie* E_j muszą być równe zero, ponieważ, jak łatwo zauważyc, dla grafu będącego drzewem wolnym nie może istnieć niezerowe rozwiązanie równań Kirchhoffa (zobacz ćwiczenie 4). Z tego powodu oba rozwiązania muszą być jednakowe. Udowodniliśmy zatem, że wszystkie rozwiązania równań Kirchhoffa można otrzymać jako sumy wielokrotności cykli podstawowych.

Gdy zastosujemy powyższe spostrzeżenia do grafu z rysunku 32, otrzymamy następujące ogólne rozwiązanie równań Kirchhoffa względem niezależnych zmiennych E_0, E_2, \dots, E_{25} :

$$\begin{aligned}
 E_1 &= E_0, & E_{14} &= E_0, \\
 E_3 &= E_0 - E_2 + E_8, & E_{15} &= E_{17} - E_{21}, \\
 E_4 &= E_0 - E_2 + E_8, & E_{16} &= E_{17} - E_{21}, \\
 E_6 &= E_0 - E_5 + E_8, & E_{18} &= E''_{19} + E_{20} - E_{21}, \\
 E_7 &= E_0 - E_5 + E_8, & E'_{19} &= E''_{19}, \\
 E_9 &= E_0, & E_{22} &= E_{17} + E_{20} - E_{21}, \\
 E_{10} &= E_0, & E_{23} &= E_{20}, \\
 E_{11} &= E_0 + E_{17} - E_{21}, & E_{24} &= E_{17} - E_{21}, \\
 E_{12} &= E_0 + E''_{13}, & E_{26} &= E_{25}, \\
 E'_{13} &= E''_{13}, & E_{27} &= E_{17} - E_{21} - E_{25}.
 \end{aligned} \tag{6}$$

Równania otrzymujemy dla każdej krawędzi e_j poddrzewa, wypisując z odpowiednim znakiem te E_k , dla których E_j występuje w cyklu C_k . [Stąd macierz

współczynników w (6) jest po prostu transponowaną macierzą współczynników z (3)].

Ściśle rzecz ujmując, cyklu C_0 nie powinniśmy nazywać cyklem podstawowym, ponieważ zawiera sztuczną krawędź e_0 . C_0 bez krawędzi e_0 możemy nazywać ścieżką podstawową z wierzchołka Start do wierzchołka Stop. Warunek brzegowy mówiący, że bloki Start i Stop są wykonywane dokładnie raz, odpowiada związkowi

$$E_0 = 1. \quad (7)$$

Dotychczasowa analiza pokazuje, w jaki sposób otrzymać wszystkie rozwiązania równań Kirchhoffa. Tę samą metodę można zastosować do układów elektrycznych (w istocie tak stosował ją Kirchhoff). Naturalne wydaje się pytanie, czy prawa Kirchhoffa to najsilniejszy możliwy układ warunków, które można sformułować na podstawie schematu blokowego. Albo inaczej: każde wykonanie programu wyznacza pewien układ wartości E_1, E_2, \dots, E_{27} i te wartości spełniają prawo Kirchhoffa. Czy istnieje jednak takie rozwiązanie równań Kirchhoffa, które nie odpowiada wykonaniu żadnego programu? (W tym pytaniu nie zakładamy niczego o programie, znamy jedynie jego uproszczony schemat blokowy). Jeżeli istnieją rozwiązania spełniające prawo Kirchhoffa, ale nie odpowiadające żadnemu wykonaniu programu, to możemy sformułować warunek silniejszy niż prawo Kirchhoffa. Dla układów elektrycznych Kirchhoff podał drugie prawo [*Ann. Physik und Chemie* **64** (1845), 497–514]: suma spadków napięć w cyklu podstawowym musi być równa zeru. To drugie prawo w naszym przypadku nie ma zastosowania.

Istnieje jednak oczywisty warunek, który muszą spełniać zmienne E , jeśli rzeczywiście odpowiadają konkretnemu wykonaniu programu: muszą być całkowite, a nawet całkowite i nieujemne. To nie jest warunek trywialny, bo nie możemy tak po prostu przypisać niezależnym zmiennym E_2, E_5, \dots, E_{25} dowolnych nieujemnych wartości. Jeśli na przykład weźmiemy $E_2 = 2$ i $E_8 = 0$, to okaże się na mocy (6) i (7), że $E_3 = -1$. (Oznacza to, że nie istnieje takie wykonanie programu odpowiadającego schematowi z rysunku 31, które przechodziłoby dwa razy po e_2 , nie przechodząc ani razu po e_8). Ten warunek jeszcze jednak nie wystarcza. Rozważmy na przykład rozwiązanie, w którym $E''_{19} = 1$, $E_2 = E_5 = \dots = E_{17} = E_{20} = E_{21} = E_{25} = 0$. Jedyną możliwością dostania się do e_{18} jest przejście po e_{15} . Następujący warunek jest konieczny i wystarczający: niech E_2, E_5, \dots, E_{25} będą dowolnymi niezależnymi wartościami wyznaczającymi wartości E_1, E_3, \dots, E_{27} (z (6) i (7)). Założymy, że wszystkie E są nieujemnymi liczbami całkowitymi. Założymy ponadto, że graf zbudowany z tych krawędzi e_j , dla których $E_j > 0$, oraz wierzchołków, które leżą na tych krawędziach, jest spójny. Przy takich założeniach zawsze istnieje taka ścieżka od wierzchołka Start do wierzchołka Stop, że po krawędzi e_j obliczenie przechodzi E_j razy. To twierdzenie udowadniamy w następnym podpunkcie (zobacz ćwiczenie 2.3.4.2–24).

Podsumujmy nasze rozważania.

Twierdzenie K. Jeżeli schemat blokowy (taki jak na rysunku 31) zawiera n bloków (włączając Start i Stop) oraz m strzałek, to można znaleźć $m - n + 1$

cykli podstawowych oraz ścieżkę podstawową od wierzchołka Start do wierzchołka Stop, takie że dowolna ścieżka od wierzchołka Start do wierzchołka Stop jest równoważna (w sensie liczby przejść po każdej krawędzi) jednemu przejściu ścieżki podstawowej plus jednoznacznie wyznaczonym liczbom przejść po każdym cyklu podstawowym. (Ścieżka podstawowa i cykle podstawowe mogą zawierać krawędzie, które należy przejść w kierunku *odwrotnym* do kierunku strzałki; przyjmujemy, że oznacza to przejście po strzałce –1 razy).

Odwrotnie, dla dowolnej ścieżki złożonej ze ścieżki podstawowej oraz cykli podstawowych, dla której całkowita liczba przejść po każdej krawędzi jest nieujemna oraz dla której krawędzie o niezerowej liczbie przejść wraz z przylegającymi wierzchołkami tworzą graf spójny, istnieje przynajmniej jedna równoważna ścieżka od wierzchołka Start do wierzchołka Stop. ■

Cykle podstawowe można znaleźć, wyznaczając wolne poddrzewo, jak na rysunku 32. Jeśli wybierzymy inne poddrzewo, to możemy dostać inny zbiór cykli podstawowych. Fakt, że cykli podstawowych jest zawsze $m - n + 1$, wynika z twierdzenia A. Modyfikacje, które wykonaliśmy, przechodząc od rysunku 31 do rysunku 32, po dodaniu e_0 nie mają wpływu na wartość $m - n + 1$, choć mogą spowodować zwiększenie m i n (jednocześnie). Podaną konstrukcję można uogólnić, tak by te trywialne modyfikacje nie były potrzebne (zobacz ćwiczenie 9).

Twierdzenie K jest ciekawe, ponieważ stwierdza, że prawo Kirchhoffa (na które składa się n równań z m niewiadomymi E_1, E_2, \dots, E_m) ma tylko jedno równanie „nadmiarowe”: n równań pozwala wyeliminować $n - 1$ niewiadomych. W naszej analizie niewiadome oznaczały jednak liczby przejść po krawędziach, a nie liczby wejść do bloku. Ćwiczenie 8 pokazuje, w jaki sposób skonstruować graf, którego krawędzie odpowiadają blokom schematu blokowego, dzięki czemu zaprezentowaną tu teorię można wykorzystać do wyznaczenia prawdziwej liczby zmiennych niezależnych.

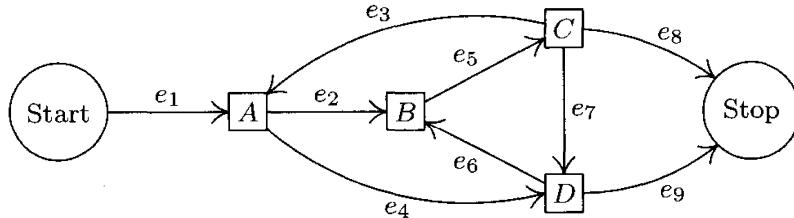
Zastosowania twierdzenia K w oprogramowaniu mierzącym wydajność programów napisanych w językach wysokiego poziomu omawiają Thomas Ball i James R. Larus w *ACM Trans. Prog. Languages and Systems* 16 (1994), 1319–1360.

ĆWICZENIA

1. [14] Wypisz wszystkie cykle grafu z rysunku 29 zawierające B .
2. [M20] Udowodnij, że jeśli w grafie istnieje ścieżka od wierzchołka V do wierzchołka V' , to istnieje ścieżka prosta z V do V' .
3. [15] Która ścieżka prowadząca od wierzchołka Start do wierzchołka Stop odpowiada (w sensie twierdzenia K) jednemu przejściu po ścieżce podstawowej oraz jednemu obejściu cyklu C_2 na rysunku 32?
- ▶ 4. [M20] Niech G' będzie skończonym drzewem wolnym o krawędziach e_1, \dots, e_{n-1} , na których narysowano strzałki. Niech E_1, \dots, E_{n-1} będą liczbami spełniającymi prawo Kirchhoffa (1) w G' . Pokaż, że $E_1 = \dots = E_{n-1} = 0$.
5. [20] Korzystając z (6), za pomocą niezależnych zmiennych E_2, E_5, \dots, E_{25} wyraź wartości A, B, \dots, S pojawiające się w blokach na rysunku 31.

- 6. [M27] Przypuśćmy, że graf ma n wierzchołków V_1, \dots, V_n i m krawędzi e_1, \dots, e_m . Każda krawędź e jest reprezentowana przez parę liczb całkowitych (a, b) , jeśli prowadzi od V_a do V_b . Zaprojektuj algorytm, który wczytuje pary $(a_1, b_1), \dots, (a_m, b_m)$ i wypisuje podzbiór krawędzi składający się na drzewo wolne. Algorytm powinien zgłaszać błąd, jeżeli takiego zbioru wybrać się nie da. Postaraj się wymyślić algorytm wydajny.

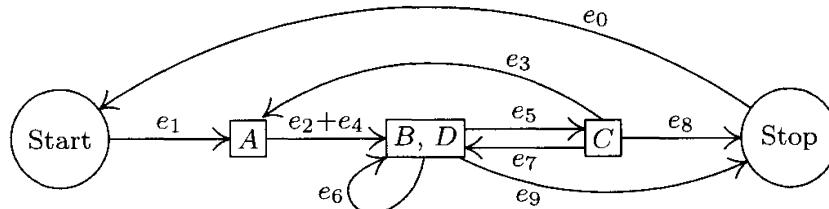
7. [22] Przeprowadź konstrukcję opisaną w tekście dla schematu blokowego



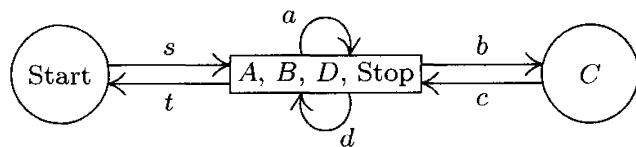
korzystając z wolnego poddrzewa złożonego z krawędzi e_1, e_2, e_3, e_4, e_9 . Jak wyglądają cykle podstawowe? Wyraź E_1, E_2, E_3, E_4, E_9 za pomocą E_5, E_6, E_7 i E_8 .

- 8. [M25] Stosując prawo Kirchhoffa do schematów blokowych programów, jesteśmy raczej zainteresowani *przepływem wierzchołkowym* (liczbą wykonień bloku) niż przepływem krawędziowym, analizowanym w tekście. Na przykład w grafie z ćwiczenia 7 przepływy wierzchołkowe wynoszą $A = E_2 + E_4$, $B = E_5$, $C = E_3 + E_7 + E_8$, $D = E_6 + E_9$.

Gdy zgrupujemy wierzchołki, traktując każdą grupę jak „megawierzchołek”, możemy połączyć przepływy wierzchołkowe odpowiadające pojedynczemu przepływowi wierzchołkowemu. Można na przykład połączyć krawędzie e_2 i e_4 na powyższym diagramie, jeśli połączymy B i D :



(Dodaliśmy ponadto e_0 , jak w tekście). Kontynuując, możemy połączyć $e_3 + e_7$, potem $(e_3 + e_7) + e_8$, następnie $e_6 + e_9$, aż uzyskamy *zredukowany schemat blokowy* o krawędziach $s = e_1$, $a = e_2 + e_4$, $b = e_5$, $c = e_3 + e_7 + e_8$, $d = e_6 + e_9$, $t = e_0$ – dokładnie jedna krawędź na każdy wierzchołek w początkowym schemacie blokowym:



Z własności konstrukcji wynika, że prawo Kirchhoffa obowiązuje również w zredukowanym schemacie blokowym. Nowe przepływy krawędziowe są pierwotnymi przepływami wierzchołkowymi. Wobec tego metodę pokazaną w tekście można zastosować również do zredukowanego schematu blokowego, by wyznaczyć zależności między przepływami wierzchołkowymi.

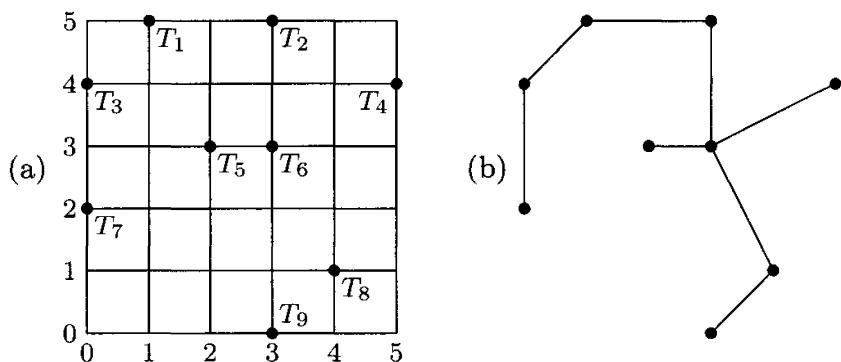
Udowodnij, że powyższy proces redukcji można odwrócić w tym sensie, że każdy zbiór przepływów $\{a, b, \dots\}$ spełniający prawo Kirchhoffa w zredukowanym schemacie blokowym można „podzielić” na zbiór przepływów krawędziowych $\{e_0, e_1, \dots\}$ w pierwotnym schemacie blokowym. Przepływy e_j spełniają prawo Kirchhoffa i składają się

na przepływy $\{a, b, \dots\}$, mogą być jednak ujemne. (Proces redukcji przedstawiono dla jednego konkretnego schematu blokowego, ale Twój dowód powinien być poprawny dla dowolnego schematu blokowego).

9. [M22] Krawędzie e_{13} i e_{19} na rysunku 32 podzielono na dwie części, ponieważ graf nie może zawierać dwóch krawędzi łączących tę samą parę wierzchołków. Gdy jednak spojrzymy na końcowy wynik, okaże się, że dzielenie krawędzi jest dosyć sztuczne, ponieważ w (6) mamy $E'_{13} = E''_{13}$ i $E'_{19} = E''_{19}$, podczas gdy E''_{13} i E''_{19} należą do zmiennych niezależnych. Wyjaśnij, w jaki sposób można uogólnić konstrukcję, by uniknąć takiego sztucznego dzielenia krawędzi.

10. [16] Inżynier elektronik projektujący układy komputerowe stwierdził, że napięcie na n wyprowadzeniach T_1, T_2, \dots, T_n powinno być cały czas takie samo. Inżynier może łączyć przewodami dowolne pary wyprowadzeń. Chodzi o to, aby wykonać tyle połączeń, żeby po przewodach dało się dojść z każdego wyprowadzenia do dowolnego innego. Pokaż, że minimalna liczba przewodów potrzebnych do połączenia wszystkich wyprowadzeń jest równa $n - 1$ i że za pomocą $n - 1$ przewodów można uzyskać żądane połączenie wtedy i tylko wtedy, gdy ich układ jest drzewem wolnym (wyprowadzenia traktujemy jako wierzchołki grafu, przewody jako krawędzie).

11. [M27] (R. C. Prim, *Bell System Tech. J.* **36** (1957), 1389–1401). Rozważmy problem łączenia wyprowadzeń z ćwiczenia 10 z dodatkowym założeniem, że dla $i < j$ określony jest koszt $c(i, j)$ połączenia przewodem wyprowadzeń T_i i T_j . Pokaż, że następujący algorytm wyznacza drzewo połączeń o minimalnym koszcie: „Jeśli $n = 1$, to nie rób nic. W przeciwnym razie przenumeruj wyprowadzenia $\{1, \dots, n - 1\}$ i związane z nimi koszty w ten sposób, żeby $c(n - 1, n) = \min_{1 \leq i < n} c(i, n)$. Połącz wyprowadzenie T_{n-1} z T_n , następnie zmień $c(j, n - 1)$ na $\min(c(j, n - 1), c(j, n))$ dla $1 \leq j < n - 1$ i wykonaj algorytm dla $n - 1$ wyprowadzeń T_1, \dots, T_{n-1} , korzystając z nowych kosztów. (Algorytm należy wykonać ponownie, pamiętając, że wykonanie połączenia między wyprowadzeniami nazywanymi teraz T_j i T_{n-1} oznacza w istocie wykonanie połączenia między T_j i T_n , jeśli jest ono tańsze; stąd T_{n-1} i T_n traktujemy tak, jakby w dalszym ciągu wykonania algorytmu były jednym wyprowadzeniem). Ten algorytm można również wyrazić następująco: „Wybierz wyprowadzenie, od którego chcesz zacząć. Następnie wykonaj najtańsze połączenie od jeszcze nie wybranego wyprowadzenia do któregoś z wybranych, dopóki wszystkie wyprowadzenia nie zostaną wybrane”.



Rys. 33. Drzewo wolne o najmniejszym koszcie (zobacz ćwiczenie 11).

Rozważmy na przykład rysunek 33(a), ukazujący dziewięć wyprowadzeń na kwadratowej siatce. Niech koszt połączenia wyprowadzeń będzie długością przewodu, tj. odlegością między wyprowadzeniami. (Czytelnik powinien spróbować wyznaczyć minimalne drzewo „na oko”, wykorzystując intuicję, a nie podany algorytm). Algorytm połączy najpierw T_8 z T_9 , potem T_6 z T_8 , T_5 z T_6 , T_2 z T_6 , T_1 z T_2 , T_3 z T_1 , T_7 z T_3 ,

a wreszcie T_4 z T_2 lub T_6 . Drzewo o minimalnym koszcie (długość przewodu $7 + 2\sqrt{2} + 2\sqrt{5}$) jest pokazane na rysunku 33(b).

- **12.** [29] Algorytm w ćwiczeniu 11 został przedstawiony w formie niewygodnej z punktu widzenia zapisania go w postaci programu. Przeformułuj opis algorytmu, specyfikując bardziej szczegółowo jego operacje, tak by było oczywiste, w jaki sposób wykonywać je z rozsądnią wydajnością na komputerze.
- 13.** [M24] Rozważmy graf o n wierzchołkach i m krawędziach, przyjmując oznaczenia z ćwiczenia 6. Pokaż, że można zapisać dowolną permutację liczb całkowitych $\{1, 2, \dots, n\}$ jako złożenie transpozycji $(a_{k_1} b_{k_1})(a_{k_2} b_{k_2}) \dots (a_{k_t} b_{k_t})$ wtedy i tylko wtedy, gdy graf jest spójny. (Tak więc istnieją zbiory $n-1$ transpozycji generujące wszystkie permutacje n elementów, a żaden zbiór $n-2$ transpozycji nie wystarcza).

2.3.4.2. Drzewa zorientowane. W poprzednim punkcie przekonaliśmy się, że abstrakcyjny schemat blokowy można traktować jako graf, jeśli zignorujemy kierunki strzałek. Okazało się, że teoriografowe pojęcia takie jak cykl, wolne poddrzewo itp. przydają się do badania schematów blokowych. Można jednak powiedzieć o wiele więcej, gdy później potraktuje się orientację krawędzi. W takim przypadku otrzymujemy twór nazywany „grafem skierowanym” lub „digrafem”.

Graf skierowany definiujemy formalnie jako zbiór wierzchołków oraz zbiór (skierowanych) krawędzi. Każda krawędź prowadzi od pewnego wierzchołka V do pewnego wierzchołka V' . Jeśli e jest krawędzią prowadzącą od V do V' , to mówimy, że V jest *początkowym* wierzchołkiem krawędzi e , a V' jest *końcowym* wierzchołkiem krawędzi e , co zapisujemy odpowiednio $V = \text{init}(e)$, $V' = \text{fin}(e)$. Nie wykluczamy przypadku $\text{init}(e) = \text{fin}(e)$ (choć zabranialiśmy tego w definicji zwykłego grafu), ponadto wiele różnych krawędzi może mieć te same wierzchołki początkowe i końcowe. *Stopniem wyjściowym* wierzchołka V nazywamy liczbę krawędzi, które go opuszczają, tj. liczbę takich krawędzi e , że $\text{init}(e) = V$. Podobnie *stopniem wejściowym* wierzchołka V nazywamy liczbę takich krawędzi, że $\text{fin}(e) = V$.

Pojęcia ścieżek i cykli w grafie skierowanym definiujemy podobnie do ich odpowiedników w zwykłym grafie, musimy jednak zatrudnić się o nowe szczegóły techniczne. Jeśli e_1, e_2, \dots, e_n są (skierowanymi) krawędziami ($n \geq 1$), to mówimy, że (e_1, e_2, \dots, e_n) jest *ścieżką zorientowaną* długości n prowadzącą od V do V' , jeśli $V = \text{init}(e_1)$, $V' = \text{fin}(e_n)$ oraz $\text{fin}(e_k) = \text{init}(e_{k+1})$ dla $1 \leq k < n$. Mówimy, że ścieżka zorientowana (e_1, e_2, \dots, e_n) jest *prosta*, jeśli $\text{init}(e_1), \dots, \text{init}(e_n)$ są parami różne oraz $\text{fin}(e_1), \dots, \text{fin}(e_n)$ są parami różne. *Cykl zorientowany* jest prostą ścieżką zorientowaną, której pierwsza krawędź zaczyna się w tym samym węźle, w którym kończy się ostatnia. (Cykl zorientowany może mieć długość 1 lub 2, podczas gdy takie krótkie cykle były niedopuszczalne w definicji cyklu w zwykłym grafie, podanej w poprzednim punkcie. Czy jasne jest, dlaczego to ma sens?)

W ramach przykładu odwołamy się do rysunku 31 z poprzedniego punktu. Blok oznaczony „ J ” jest wierzchołkiem o stopniu wejściowym 2 (krawędzie e_{16} i e_{21}) i stopniu wyjściowym 1. Ciąg $(e_{17}, e_{19}, e_{18}, e_{22})$ jest ścieżką zorientowaną

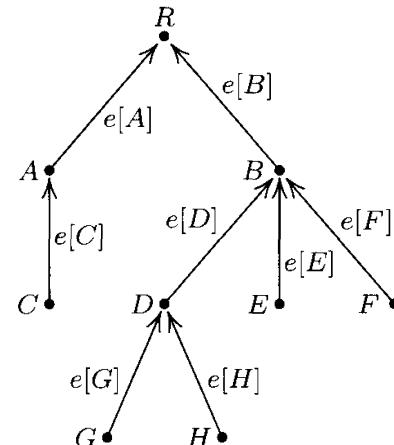
długości 4 prowadzącą od J do P ; ta ścieżka nie jest prosta, bo na przykład $\text{init}(e_{19}) = L = \text{init}(e_{22})$. Schemat nie zawiera cykli zorientowanych długości 1, ale (e_{18}, e_{19}) jest cyklem zorientowanym o długości 2.

Mówimy, że graf skierowany jest *silnie spójny*, jeżeli dla dowolnych wierzchołków $V \neq V'$ istnieje ścieżka zorientowana od V do V' . Mówimy, że graf zorientowany jest *ukorzeniony*, jeżeli istnieje przynajmniej jeden *korzeń*, tj. taki wierzchołek R , że dla dowolnego $V \neq R$ istnieje ścieżka od V do R . „Silna spójność” zawsze pociąga za sobą „ukorzenienie”, ale implikacja w drugą stronę nie musi zachodzić. Schemat blokowy (jak rysunek 31 z poprzedniego rozdziału) jest przykładem grafu ukorzenionego o korzeniu w wierzchołku Stop. Jeżeli dodamy krawędź od wierzchołka Stop do wierzchołka Start (rysunek 32), to graf stanie się silnie spójny.

Każdy skierowany graf G odpowiada w oczywisty sposób zwykłemu (nieskierowanemu) grafowi G_0 – wystarczy zignorować orientację krawędzi oraz usunąć krawędzie wielokrotne i pętle. Mówiąc ściśle, krawędź między V i V' należy do G_0 wtedy i tylko wtedy, gdy $V \neq V'$ i G ma (skierowaną) krawędź od V' do V lub od V do V' . Możemy mówić o (niezorientowanych) ścieżkach i cyklach w G , mając na myśli ścieżki i cykle w G_0 . Możemy powiedzieć, że G jest *spójny*, jeśli odpowiadający mu graf G_0 jest spójny. „Spójność” jest własnością słabszą niż „silna spójność”, słabszą nawet niż „ukorzenienie”.

Drzewo zorientowane (zobacz rysunek 34), czasami przez innych autorów nazywane „drzewem ukorzenionym”, jest grafem skierowanym z wyróżnionym wierzchołkiem R spełniającym poniższe warunki:

- Każdy wierzchołek $V \neq R$ jest wierzchołkiem początkowym dokładnie jednej krawędzi, oznaczanej $e[V]$;
- R nie jest wierzchołkiem początkowym żadnej krawędzi;
- R jest korzeniem w sensie powyższej definicji (tj. dla każdego wierzchołka $V \neq R$ istnieje zorientowana ścieżka od V do R).



Rys. 34. Drzewo zorientowane.

Z powyższej definicji wynika następujący prosty wniosek: dla każdego wierzchołka $V \neq R$ istnieje *dokładnie jedna* ścieżka zorientowana z V do R , zatem w drzewie zorientowanym nie ma cykli zorientowanych.

Łatwo się przekonać, że nasza poprzednia definicja „drzewa zorientowanego” (na początku podrozdziału 2.3) jest zgodna z nową definicją, jeżeli liczba wierzchołków jest skończona – krawędź $e[V]$ odpowiada dowiązaniu prowadzącemu od V do $\text{PARENT}[V]$.

(Nieskierowany) graf odpowiadający drzewu zorientowanemu jest spójny, co wynika z warunku (c). Ponadto nie ma cykli, bo jeśli (V_0, V_1, \dots, V_n) jest nieskierowanym cyklem, takim że $n \geq 3$, oraz jeśli krawędzią między V_0 i V_1 jest $e[V_1]$, to krawędzią między V_1 i V_2 musi być $e[V_2]$; analogicznie krawędzią między

V_{k-1} i V_k musi być $e[V_k]$ dla $1 \leq k \leq n$, co przeczy brakowi cykli skierowanych. Jeśli krawędzią między V_0 i V_1 nie jest $e[V_1]$, to musi to być $e[V_0]$, więc to samo rozumowanie możemy zastosować do cyklu

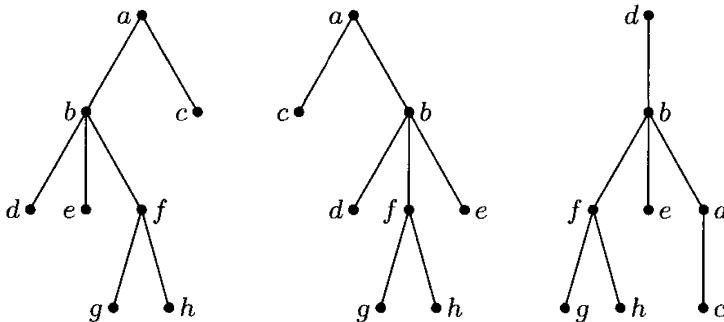
$$(V_1, V_0, V_{n-1}, \dots, V_1),$$

ponieważ $V_n = V_0$. Stąd drzewo zorientowane jest drzewem wolnym, gdy pominiemy orientacje strzałek.

Warto zauważyć, że możemy odwrócić opisany proces. Jeżeli weźmiemy dowolne niepuste drzewo wolne, na przykład takie jak na rysunku 30, to możemy wybrać *dowolny* wierzchołek na korzeń R i tak dobrać orientacje krawędzi, by otrzymać drzewo zorientowane. Intuicyjnie pomysł polega na tym, żeby ująć drzewo za wybrany wierzchołek i podnieść tak, by konary zwiśli w dół, następnie na krawędziach narysować strzałki w góre. Ścisłej powyższą konstrukcję można opisać tak:

Ustal orientację krawędzi $V — V'$ na $V \rightarrow V'$ wtedy i tylko wtedy, gdy istnieje ścieżka prosta od V do R wiodąca przez V' , tj. jeśli istnieje ścieżka prosta postaci (V_0, V_1, \dots, V_n) , gdzie $n > 0$, $V_0 = V$, $V_1 = V'$, $V_n = R$.

By przekonać się, że taka konstrukcja jest poprawna, musimy udowodnić, że każdej krawędzi przypiszemy w ten sposób orientację $V \leftarrow V'$ lub $V \rightarrow V'$. Nietrudno to pokazać, bowiem jeśli (V, V_1, \dots, R) i (V', V'_1, \dots, R) są ścieżkami prostymi, to istnieje cykl, jeśli nie zachodzi warunek $V = V'_1$ lub $V_1 = V'$. Wynika stąd, że kierunki krawędzi w drzewie zorientowanym są wyznaczone jednoznacznie, jeżeli wiemy, który wierzchołek jest korzeniem (zatem jeśli na schemacie zaznaczamy korzeń, to nie musimy rysować kierunków krawędzi).



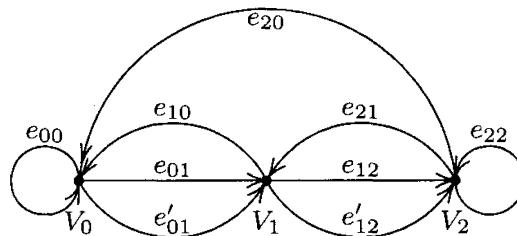
Rys. 35. Trzy drzewa.

Przyjrzyjmy się związkom między trzema typami drzew: drzewami (uporządkowanymi), mającymi największe znaczenie w praktyce programowania, drzewami zorientowanymi (lub nieuporządkowanymi) oraz drzewami wolnymi. Dwa ostatnie typy drzew pojawiają się w analizie algorytmów, jednakże nie tak często jak pierwszy. Podstawową różnicą między tymi typami drzew jest ilość informacji, na które zwracamy uwagę. Na przykład na rysunku 35 pokazano trzy drzewa, które są różne jako drzewa uporządkowane (z korzeniem u góry). Gdy spojrzymy na nie jak na drzewa zorientowane, to drzewo pierwsze i drugie okaże się

identyczne, ponieważ porządek poddrzew w węźle jest nieistotny; wszystkie trzy grafy z rysunku 35 są identyczne jako drzewa wolne, ponieważ w tym ujęciu nie interesuje nas położenie korzenia.

Cykł Eulera w grafie skierowanym jest ścieżką zorientowaną (e_1, e_2, \dots, e_m) , taką że każda krawędź grafu skierowanego występuje na niej dokładnie raz, a ponadto $\text{fin}(e_m) = \text{init}(e_1)$. Cykl Eulera „obchodzi” wszystkie krawędzie grafu. (Nazwa cykli Eulera wzięła się od pochodzącej z 1736 roku znanej analizy autorstwa Leonharda Eulera, który dowodził, że podczas niedzielnego spaceru nie sposób obejść bez powtórzeń wszystkich siedmiu mostów w Królewcu. Euler zajmował się problemem analogicznym do naszego, z tym że dotyczącym grafów nieszczególnionych. Cykli Eulera nie należy mylić z „cyklami Hamiltona”; cykl Hamiltona jest cyklem zorientowanym, w którym każdy *wierzchołek* występuje dokładnie raz; zobacz rozdział 7).

Mówimy, że graf skierowany jest *zrównoważony* (zobacz rysunek 36), jeśli każdy wierzchołek ma ten sam stopień wejściowy i wyjściowy, tzn. jeśli liczba krawędzi o początku w V jest taka sama jak liczba krawędzi o końcu w V . Ten warunek jest ściśle związany z prawem Kirchhoffa (zobacz ćwiczenie 24). Jeżeli graf skierowany ma cykl Eulera, to oczywiście musi być spójny i zrównoważony – chyba że ma *wierzchołki izolowane*, tj. wierzchołki o stopniu wejściowym i wyjściowym równym zeru.



Rys. 36. Zrównoważony graf skierowany.

Jak na razie w tym rozdziale przyglądamy się definicjom (graf skierowany, krawędź skierowana, wierzchołek początkowy, wierzchołek końcowy, stopień wyjściowy, stopień wejściowy, ścieżka zorientowana, prosta ścieżka zorientowana, cykl zorientowany, drzewo zorientowane, cykl Eulera, wierzchołek izolowany, silna spójność, ukorzenienie, zrównoważenie), ale twierdzeń o własnościach definiowanych pojęć jest jak na lekarstwo. Ale to nic, właśnie szukujemy się do czegoś poważniejszego. Pierwszy podstawowy wynik to twierdzenie autorstwa I. J. Gooda [J. London Math. Soc. **21** (1947), 167–169], który pokazał, że cykl Eulera da się poprowadzić zawsze, poza przypadkami, gdy jasno widać, że się nie da:

Twierdzenie G. Skończony graf skierowany bez wierzchołków izolowanych ma cykl Eulera wtedy i tylko wtedy, gdy jest spójny i zrównoważony.

Dowód. Założmy, że G jest zrównoważony i niech

$$P = (e_1, \dots, e_m)$$

będzie najdłuższą ścieżką w G , na której nie powtarza się żadna krawędź. Jeśli $V = \text{fin}(e_m)$ i jeśli k jest stopniem wyjściowym wierzchołka V , to każda krawędź wychodząca z V musi należeć do P . W przeciwnym przypadku moglibyśmy dodać e i otrzymać dłuższą ścieżkę. Ale jeśli $\text{init}(e_j) = V$ i $j > 1$, to $\text{fin}(e_{j-1}) = V$. Graf G jest zrównoważony, zatem musi być

$$\text{init}(e_1) = V = \text{fin}(e_m),$$

bo w przeciwnym razie stopień wejściowy V byłby nie mniejszy niż $k + 1$.

Teraz biorąc cykliczne permutacje P , możemy udowodnić, że jeżeli krawędź nie leży na ścieżce, to ani jej początek, ani koniec nie leży na ścieżce. Zatem jeśli P nie jest cyklem Eulera, to G nie jest spójny ■

Istnieje ważny związek między cyklami Eulera a drzewami zorientowanymi:

Lemat E. Niech (e_1, \dots, e_m) będzie cyklem Eulera w grafie skierowanym G bez wierzchołków izolowanych. Niech $R = \text{fin}(e_m) = \text{init}(e_1)$. Niech dla każdego wierzchołka $V \neq R$ symbol $e[V]$ oznacza tę spośród krawędzi wychodzących z wierzchołka V , która jako ostatnia występuje w cyklu Eulera. Dokładnie:

$$e[V] = e_j, \quad \text{jeśli } \text{init}(e_j) = V \quad \text{oraz} \quad \text{init}(e_k) \neq V \quad \text{dla } j < k \leq m. \quad (1)$$

Wówczas wierzchołki G oraz krawędzie $e[V]$ tworzą drzewo zorientowane o korzeniu R .

Dowód. Warunki (a) i (b) z definicji drzewa zorientowanego są oczywiście spełnione. Na mocy ćwiczenia 7 wystarczy pokazać, że nie ma cykli zorientowanych wśród krawędzi $e[V]$. Jest tak, ponieważ jeśli $\text{fin}(e[V]) = V' = \text{init}(e[V'])$, gdzie $e[V] = e_j$ i $e[V'] = e_{j'}$, to $j < j'$. ■

Najłatwiej zrozumieć powyższy lemat, jeśli odwróciemy kolej rzeczy i rozważymy „pierwsze wejście” do każdego wierzchołka. Pierwsze wejścia tworzą drzewo nieuporządkowane, w którym wszystkie krawędzie prowadzą od wierzchołka R . Istnieje zaskakujące i ważne twierdzenie odwrotne do lematu E, autorstwa T. van Aardenne-Ehrenfest i N. G. de Bruijna [Simon Stevin **28** (1951), 203–217]:

Twierdzenie D. Niech G będzie skończonym, zrównoważonym grafem skierowanym i niech G' będzie zorientowanym drzewem złożonym z wierzchołków oraz niektórych krawędzi G . Niech R będzie korzeniem G' i niech $e[V]$ oznacza krawędź G' o wierzchołku początkowym V . Niech e_1 będzie dowolną krawędzią G , taką że $\text{init}(e_1) = R$. Wówczas $P = (e_1, e_2, \dots, e_m)$ jest cyklem Eulera, jeżeli jest taką ścieżką zorientowaną, że

- i) każda krawędź występuje na niej co najwyżej raz, tj. $e_j \neq e_k$, gdy $j \neq k$.
- ii) $e[V]$ nie występuje w P dopóki nie jest to jedyna krawędź zgodna z regułą (i), tj. jeśli $e_j = e[V]$ i jeśli e jest krawędzią, taką że $\text{init}(e) = V$, to $e = e_k$ dla pewnego $k \leq j$.
- iii) P kończy się tylko wtedy, gdy nie można jej przedłużyć zgodnie z regułą (i), tj. jeśli $\text{init}(e) = \text{fin}(e_m)$, to $e = e_k$ dla pewnego k .

Dowód. Na mocy (iii) i rozumowania z dowodu twierdzenia G musimy mieć $\text{fin}(e_m) = \text{init}(e_1) = R$. Jeśli zatem e jest krawędzią nie wystającą w P , to niech $V = \text{fin}(e)$. Z faktu, że G jest zrównoważony, wynika, że V jest wierzchołkiem początkowym pewnej krawędzi nie należącej do P . A jeśli $V \neq R$, to warunek (ii) mówi, że $e[V]$ nie należy do P . Posługujemy się teraz takim samym rozumowaniem dla $e = e[V]$ i stwierdzamy, że R jest wierzchołkiem początkowym pewnej krawędzi spoza P , co przeczy warunkowi (iii). ■

Istota twierdzenia B polega na tym, że pokazuje ono prosty sposób znalezienia cyklu Eulera w grafie zrównoważonym, jeśli dysponujemy dowolnym zorientowanym poddrzewem tego grafu. (Zobacz przykład w ćwiczeniu 14). W istocie twierdzenie D pozwala znaleźć liczbę cykli Eulera w grafie skierowanym; ten i wiele innych wniosków z materiału przedstawionego w tym rozdziale omawiamy w poniższych ćwiczeniach.

ĆWICZENIA

1. [M20] Udowodnij, że jeśli dla wierzchołków V i V' grafu skierowanego istnieje ścieżka zorientowana od V do V' , to istnieje prosta ścieżka zorientowana od V do V' .
2. [15] Które z „cykli podstawowych” wypisanych w (3) w podpunkcie 2.3.4.1 są cyklami zorientowanymi w grafie skierowanym (rysunek 32) omawianym w tym podpunkcie?
3. [16] Narysuj graf skierowany, który jest spójny, ale nie ukorzeniony.
- ▶ 4. [M20] Sortowanie topologiczne można zdefiniować dla dowolnego skończonego grafu skierowanego G jako liniowe uporządkowanie jego wierzchołków $V_1 V_2 \dots V_n$, takie że dla dowolnej krawędzi e $\text{init}(e)$ poprzedza $\text{fin}(e)$. (Zobacz punkt 2.2.3, rysunki 6 i 7). Nie wszystkie grafy skierowane można posortować topologicznie. Które można? (Formułując odpowiedź, skorzystaj z terminologii podanej w tym rozdziale).
5. [M16] Niech G będzie grafem skierowanym zawierającym ścieżkę zorientowaną (e_1, \dots, e_n) , taką że $\text{fin}(e_n) = \text{init}(e_1)$. Udowodnij, że G nie jest drzewem zorientowanym. Postuż się terminologią przedstawioną w tym rozdziale.
6. [M21] Prawda czy fałsz: ukorzeniony graf skierowany nie zawierający cykli ani cykli zorientowanych jest drzewem zorientowanym.
- ▶ 7. [M22] Prawda czy fałsz: graf skierowany spełniający warunki (a) i (b) z definicji drzewa zorientowanego oraz nie posiadający cykli zorientowanych jest drzewem zorientowanym.
8. [HM40] Przeanalizuj własności grup automorfizmów drzew zorientowanych, to jest grup wszystkich permutacji π wierzchołków i krawędzi, dla których zachodzi warunek $\text{init}(e\pi) = \text{init}(e)\pi$, $\text{fin}(e\pi) = \text{fin}(e)\pi$.
9. [18] Zorientuj krawędzie drzewa wolnego z rysunku 30 ze strony 377, tak by otrzymać drzewo zorientowane o korzeniu G .
10. [22] Drzewo zorientowane o wierzchołkach V_1, \dots, V_n można w następujący sposób reprezentować w pamięci za pomocą tablicy $P[1], \dots, P[n]$: jeśli V_j jest korzeniem, to $P[j] = 0$; w przeciwnym razie $P[j] = k$, jeśli krawędź $e[V_j]$ prowadzi od V_j do V_k . ($P[1], \dots, P[n]$ jest tym samym co tablica „parent” w algorytmie 2.3.3E).

W tekście pokazaliśmy, w jaki sposób drzewo wolne można przekształcić na drzewo zorientowane, wybierając korzeń. Można zacząć do drzewa zorientowanego, przekształcić je na drzewo wolne (wycierając zwroty strzałek), a następnie ustalić nowe orientacje krawędzi, otrzymując drzewo zorientowane o nowym korzeniu. Zaprojektuj algorytm, który wykonuje to przekształcenie. Danymi wejściowymi dla algorytmu są wartości zapisane w tablicy $P[1], \dots, P[n]$, reprezentującej drzewo zorientowane, oraz liczba całkowita j , $1 \leq j \leq n$. Algorytm ma przekształcać tablicę P w taki sposób, by reprezentowała to samo drzewo wolne i drzewo zorientowane o korzeniu V_j .

- ▶ **11.** [28] Przy założeniach takich jak w ćwiczeniu 2.3.4.1–6, z dokładnością do tego, że (a_k, b_k) reprezentuje krawędź skierowaną od V_{a_k} do V_{b_k} , zaprojektuj algorytm który nie tylko wypisuje wolne poddrzewo, ale także cykle podstawowe. [*Wskazówka:* Algorytm podany w rozwiązaniu ćwiczenia 2.3.4.1–6 można połączyć z algorytmem z poprzedniego ćwiczenia].
- 12.** [M10] Spójrzmy na związek między drzewami zorientowanymi zdefiniowanymi na początku podrozdziału 2.3 a zdefiniowanymi w tym rozdziale. Czy *stopień węzła* w tamtych drzewach jest *stopniem wejściowym* czy *stopniem wyjściowym* według nowej definicji?
- ▶ **13.** [M24] Udowodnij, że jeśli R jest korzeniem (być może nieskończonego) grafu skierowanego G , to G zawiera zorientowane poddrzewo o tych samych wierzchołkach, co graf G i korzeniu R . (Wynika stąd, że zawsze można wybrać wolne poddrzewo schematu blokowego w ten sposób, że jest ono poddrzewem *zorientowanym*; dla diagramu z rysunku 32 w podpunkcie 2.3.4.1 moglibyśmy wybrać $e''_{13}, e''_{19}, e_{20}$ i e_{17} zamiast e'_{13}, e'_{19}, e_{23} i e_{15}).
- 14.** [21] Niech G będzie zrównoważonym grafem skierowanym z rysunku 26, a G' niech będzie drzewem zorientowanym o wierzchołkach V_0, V_1, V_2 i krawędziach e_{01}, e_{21} . Znajdź wszystkie ścieżki P spełniające warunki twierdzenia D , które zaczynają się od krawędzi e_{12} .
- 15.** [M20] Prawda czy fałsz: spójny i zrównoważony graf skierowany jest silnie spójny.
- ▶ **16.** [M24] W popularnym „pasjansie zegarowym” 52 karty układają się zakryte w trzy-nastu kupkach po cztery karty. Dwanaście kupek jest ułożonych w okręgu i rozłożonych jak liczby na tarczy zegara, a trzynasta leży po środku. Pasjans polega na tym, że odkrywamy górną kartę ze środkowej kupki i jeżeli wartością karty jest k , to kładziemy ją obok kupki numer k (przyjmujemy, że as ma numer 1, a walet, królowa i król odpowiednio 11, 12 i 13). Następnie odkrywamy kartę z kupki numer k i kładziemy ją obok odpowiadającej jej kupki itd., aż dojdziemy do takiego momentu, że nie można wykonać następnego ruchu, bo położyliśmy kartę obok kupki, na której już nie ma kart. (W tym pasjansie układający nie podejmuje żadnych decyzji). Uznaje się, że pasjans wyszedł, jeśli uda się odkryć wszystkie karty. [Zobacz: E. D. Cheney, *Patience* (Boston: Lee & Shepard, 1870), 62–65; według M. Whitmore Jonesten pasjans był w Anglii nazywany „Travelers’ Patience” (pasjans podrózników), zobacz *Games of Patience* (London: L. Upcott Gill, 1900), rozdział 7].

Pokaż, że pasjans wychodzi wtedy i tylko wtedy, gdy następujący graf skierowany jest drzewem zorientowanym: wierzchołkami grafu są V_1, V_2, \dots, V_{13} ; krawędziami grafu są e_1, e_2, \dots, e_{12} , gdzie e_j prowadzi od V_j do V_k , jeśli k jest kartą na spodzie kupki j po rozłożeniu kart.

(W szczególności, jeśli na spodzie kupki j leży „ j ” dla $j \neq 13$, to widać, że pasjans nie może wyjść, ponieważ ta karta nigdy nie zostanie odkryta. Twierdzenie

będące przedmiotem tego ćwiczenia daje niezwykle szybką metodę układania pasjansa zegarowego!)

17. [M32] Jakie jest prawdopodobieństwo wyjścia pasjansa zegarowego (opisanego w ćwiczeniu 16), jeśli talia jest potasowana losowo? Jakie jest prawdopodobieństwo zdarzenia, że dokładnie k kart jest zakrytych, gdy nie można wykonać następnego ruchu?

18. [M30] Niech G będzie grafem o $n + 1$ wierzchołkach V_0, V_1, \dots, V_n i m krawędziach e_1, \dots, e_m . Zrób z grafu G graf skierowany, nadając orientacje jego krawędziom. Następnie skonstruuj macierz A o wymiarach $m \times (n + 1)$, taką że

$$a_{ij} = \begin{cases} +1, & \text{jeśli } \text{init}(e_i) = V_j; \\ -1, & \text{jeśli } \text{fin}(e_i) = V_j; \\ 0, & \text{w pozostałych przypadkach.} \end{cases}$$

Niech A_0 będzie macierzą $m \times n$ powstałą z macierzy A po usunięciu kolumny 0.

- a) Pokaż dla $m = n$, że wyznacznik macierzy A_0 jest równy 0, jeśli G nie jest drzewem wolnym, oraz równy ± 1 , gdy G jest drzewem wolnym.
- b) Pokaż, że dla dowolnego m wyznacznik macierzy $A_0^T A_0$ jest liczbą wolnych poddrzew grafu G (dokładnie liczbą sposobów, na które można wybrać n spośród m krawędzi, tak by tworzyły drzewo wolne). [Wskazówka: Skorzystaj z (a) oraz wyniku ćwiczenia 1.2.3–46].

19. [M31] (*Twierdzenie macierzowe o drzewach*) Niech G będzie grafem skierowanym o $n + 1$ wierzchołkach V_0, V_1, \dots, V_n . Niech A będzie macierzą $(n + 1) \times (n + 1)$, taką że

$$a_{ij} = \begin{cases} -k, & \text{jeśli } i \neq j \text{ oraz istnieje } k \text{ krawędzi z } V_i \text{ do } V_j; \\ t, & \text{jeśli } i = j \text{ oraz istnieje } t \text{ krawędzi od } V_i \text{ do innych wierzchołków.} \end{cases}$$

(Mamy stąd, że $a_{i0} + a_{i1} + \dots + a_{in} = 0$ dla $0 \leq i \leq n$). Niech A_0 będzie macierzą A po usunięciu zerowego wiersza i kolumny. Na przykład, jeśli G jest grafem skierowanym z rysunku 36, to

$$A = \begin{pmatrix} 2 & -2 & 0 \\ -1 & 3 & -2 \\ -1 & -1 & 2 \end{pmatrix}, \quad A_0 = \begin{pmatrix} 3 & -2 \\ -1 & 2 \end{pmatrix}.$$

- a) Pokaż, że jeśli $a_{00} = 0$ i $a_{jj} = 1$ dla $1 \leq j \leq n$ oraz jeśli G nie zawiera krawędzi prowadzących od wierzchołka do niego samego, to $\det A_0 = [G$ jest drzewem zorientowanym o korzeniu $V_0]$.
- b) Pokaż, że w ogólności $\det A_0$ jest liczbą zorientowanych poddrzew grafu G o korzeniu V_0 (tj. liczbą sposobów, na które można wybrać n krawędzi z G , by tworzyły drzewo zorientowane o korzeniu V_0). [Wskazówka: Posłuż się indukcją względem liczby krawędzi].

20. [M21] G jest nieskierowanym grafem o $n + 1$ wierzchołkach V_0, \dots, V_n . Niech B będzie macierzą $n \times n$ zdefiniowaną następująco (dla $1 \leq i, j \leq n$):

$$b_{ij} = \begin{cases} t, & \text{jeśli } i = j \text{ oraz istnieje } t \text{ krawędzi o końcu w } V_j; \\ -1, & \text{jeśli } i \neq j \text{ oraz } V_i \text{ sąsiaduje z } V_j; \\ 0, & \text{w pozostałych przypadkach.} \end{cases}$$

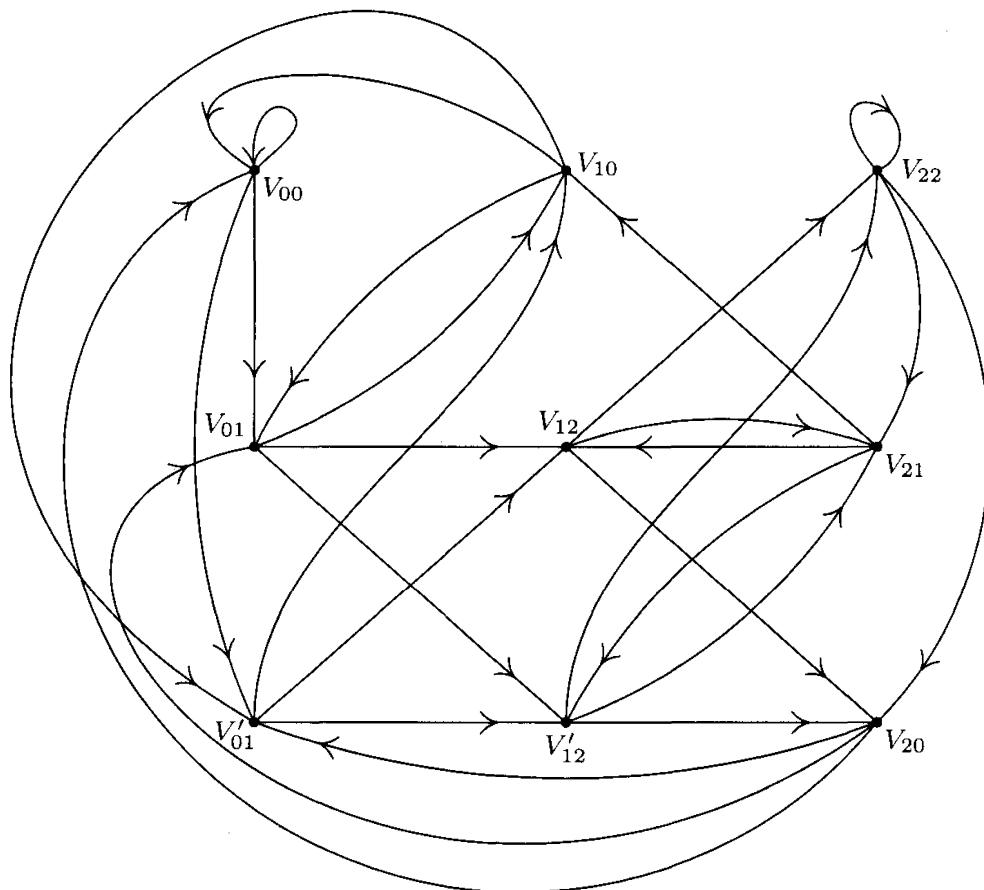
Na przykład, jeśli G jest grafem z rysunku 29 na stronie 377 i $(V_0, V_1, V_2, V_3, V_4) = (A, B, C, D, E)$, to

$$B = \begin{pmatrix} 3 & 0 & -1 & -1 \\ 0 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{pmatrix}.$$

Pokaż, że liczba wolnych poddrzew G jest równa B . [Wskazówka: skorzystaj z ćwiczenia 18 lub 19].

21. [HM38] (T. van Aardenne-Ehrenfest, N. G. de Bruijn) Na rysunku 36 jest pokazany przykładowy graf skierowany, który jest nie tylko zrównoważony, ale również *regularny*, co znaczy, że każdy wierzchołek ma odpowiednio taki sam stopień wejściowy i wyjściowy, jak inne wierzchołki. Niech G będzie regularnym grafem skierowanym o $n + 1$ wierzchołkach V_0, V_1, \dots, V_n , w którym stopień wyjściowy i wejściowy każdego wierzchołka równa się m . (Stąd całkowita liczba krawędzi wynosi $(n + 1)m$). Niech G^* oznacza graf o $(n + 1)m$ wierzchołkach odpowiadających krawędziom grafu G . Oznaczmy przez V_{jk} wierzchołek grafu G^* odpowiadający krawędzi prowadzącej w G od V_j do V_k . W grafie G^* występuje krawędź od V_{jk} do $V_{j'k'}$ wtedy i tylko wtedy, gdy $k = j'$. Graf G^* odpowiadający grafowi G z rysunku 36 jest pokazany na rysunku 37. Cykl Eulera w G odpowiada cyklowi Hamiltona w G^* i odwrotnie.

Udowodnij, że liczba zorientowanych poddrzew grafu G^* wynosi $m^{(n+1)(m-1)}$ razy liczba zorientowanych poddrzew grafu G . [Wskazówka: Skorzystaj z ćwiczenia 19].



Rys. 37. Digraf krawędziowy odpowiadający digrafowi z rysunku 36 (zobacz ćwiczenie 21).

- 22. [M26] Niech G będzie zrównoważonym grafem skierowanym o wierzchołkach V_1, V_2, \dots, V_n , bez wierzchołków izolowanych. Niech σ_j oznacza stopień wyjściowy V_j . Pokaż, że liczba cykli Eulera w G równa się

$$(\sigma_1 + \sigma_2 + \dots + \sigma_n) T \prod_{j=1}^n (\sigma_j - 1)!,$$

gdzie T jest liczbą zorientowanych poddrzew grafu G o korzeniu V_1 . [Uwaga: Czynnik $(\sigma_1 + \dots + \sigma_n)$, który jest liczbą krawędzi w grafie G , należy pominąć, jeżeli cykl Eulera (e_1, \dots, e_m) uznajemy za ten sam cykl co $(e_k, \dots, e_m, e_1, \dots, e_{k-1})$].

- 23. [M33] (N. G. de Bruijn) Niech dla dowolnego ciągu x_1, \dots, x_k nieujemnych liczb całkowitych mniejszych niż m $f(x_1, \dots, x_k)$ będzie nieujemną liczbą całkowitą mniejszą niż m . Definiujemy ciąg nieskończony: $X_1 = X_2 = \dots = X_k = 0$; $X_{n+k+1} = f(X_{n+k}, \dots, X_{n+1})$ dla $n \geq 0$. Dla ilu spośród m^{m^k} funkcji f ten ciąg jest okresowy z maksymalnym okresem długości m^k ? [Wskazówka: Skonstruuj graf skierowany o wierzchołkach (x_1, \dots, x_{k-1}) dla wszystkich $0 \leq x_j < m$ oraz krawędziach od $(x_1, x_2, \dots, x_{k-1})$ do $(x_2, \dots, x_{k-1}, x_k)$; skorzystaj z ćwiczeń 21 i 22].

- 24. [M20] Niech G będzie spójnym grafem skierowanym o krawędziach e_0, e_1, \dots, e_m . Niech E_0, E_1, \dots, E_m będzie zbiorem dodatnich liczb całkowitych spełniających prawo Kirchhoffa dla G , tj. dla każdego wierzchołka V ,

$$\sum_{\text{init}(e_j)=V} E_j = \sum_{\text{fin}(e_j)=V} E_j.$$

Zakładamy ponadto, że $E_0 = 1$. Udowodnij, że istnieje taka zorientowana ścieżka w G prowadząca od $\text{fin}(e_0)$ do $\text{init}(e_0)$, że krawędź e_j występuje na niej dokładnie E_j razy dla $1 \leq j \leq m$, a krawędź e_0 nie występuje wcale. [Wskazówka: Zastosuj twierdzenie G dla odpowiedniego grafu skierowanego].

25. [26] Zaprojektuj sposób reprezentowania w pamięci komputera grafów skierowanych, który byłby uogólnieniem sposobu reprezentowania drzew za pomocą drzew binarnych z fastrygą prawostronną. Posłuż się dwoma polami wskaźnikowymi ALINK, BLINK oraz dwoma jednobitowymi znacznikami ATAG, BTAG. Zaprojektuj reprezentację w taki sposób, by (i) element struktury danych reprezentował krawędź grafu, a nie wierzchołek grafu; (ii) reprezentacja tego grafu skierowanego była taka sama, jak reprezentacja zorientowanego drzewa za pomocą drzewa binarnego z fastrygą prawostronną (z narzuconym pewnym porządkiem dzieci) w tym sensie, że ALINK, BLINK, BTAG odpowiadają LLINK, RLINK, RTAG z punktu 2.3.2, jeśli reprezentowany graf skierowany jest drzewem zorientowanym o korzeniu R i jeśli dodamy krawędź od R do nowego wierzchołka H ; (iii) reprezentacja była symetryczna w tym sensie, by zamiana ALINK, ATAG z BLINK, BTAG odpowiadała zamianie kierunków krawędzi w grafie skierowanym.

- 26. [HM39] (Analiza algorytmu randomizacyjnego) Niech G będzie grafem skierowanym o wierzchołkach V_1, V_2, \dots, V_n . Założmy, że G reprezentuje schemat blokowy algorytmu, V_1 oznacza wierzchołek Start, a V_n wierzchołek Stop. (Stąd V_n jest korzeniem G). Przypuśćmy, że każdej krawędzi e grafu G przypisano prawdopodobieństwo $p(e)$ i że te prawdopodobieństwa spełniają warunek

$$0 < p(e) \leq 1; \quad \sum_{\text{init}(e)=V_j} p(e) = 1 \quad \text{dla } 1 \leq j < n.$$

Rozważmy ścieżkę, która rozpoczyna się w wierzchołku V_1 , a następnie wiedzie kolejnymi krawędziami e z prawdopodobieństwem $p(e)$, aż do osiągnięcia wierzchołka V_n . Wybór krawędzi w każdym wierzchołku jest niezależny od wszystkich wcześniejszych wyborów.

Weźmy na przykład graf z ćwiczenia 2.3.4.1–7 z prawdopodobieństwami $1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 1, \frac{3}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}$ na krawędziach e_1, e_2, \dots, e_9 . Prawdopodobieństwo wybrania ścieżki „Start–A–B–C–A–D–B–C–Stop” jest równe $1 \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{3}{4} \cdot 1 \cdot \frac{1}{4} = \frac{3}{128}$.

Takie losowe ścieżki noszą nazwę *łańcuchów Markowa*, od nazwiska rosyjskiego matematyka Andrieja A. Markowa, który pierwszy badał procesy stochastyczne tego typu. Opisany jest tu model działania pewnych algorytmów, chociaż założenie, że wybory są niezależne, jest bardzo silne. Celem tego ćwiczenia jest zanalizowanie czasu działania takich algorytmów.

Analizę można uprościć, rozważając macierz $n \times n$ $A = (a_{ij})$, $a_{ij} = \sum p(e)$, gdzie sumujemy po wszystkich krawędziach e prowadzących od V_i do V_j . Jeżeli takich krawędzi nie ma, to przyjmujemy $a_{ij} = 0$. Macierz A dla podanego przykładu wygląda tak:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{3}{4} & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Widać, że $(A^k)_{ij}$ jest prawdopodobieństwem zdarzenia, że ścieżka rozpoczynająca się w V_i po k krokach skończy się w V_j .

Udowodnij, że dla dowolnego skierowanego grafu G spełniającego podane założenia zachodzą następujące stwierdzenia:

- Macierz $(I - A)$ jest nieosobliwa. [Wskazówka: Pokaż, że nie ma niezerowych wektorów x , takich że $xA^n = x$].
- Średnia liczba wystąpień wierzchołka V_j na ścieżce równa się

$$(I - A)_{1j}^{-1} = \text{dopełnienie}_{j1}(I - A)/\det(I - A), \quad \text{dla } 1 \leq j \leq n,$$

gdzie dopełnienie to dopełnienie algebraiczne. [Zatem dla podanego przykładu wierzchołki A, B, C, D znajdą się na ścieżce średnio $\frac{13}{6}, \frac{7}{3}, \frac{7}{3}, \frac{5}{3}$ razy].

- Prawdopodobieństwo zdarzenia, że V_j występuje na ścieżce, wynosi

$$a_j = \text{dopełnienie}_{j1}(I - A)/\text{dopełnienie}_{jj}(I - A);$$

ponadto $a_n = 1$, zatem z prawdopodobieństwem równym jeden ścieżka kończy się po skończonej liczbie kroków.

- Prawdopodobieństwo zdarzenia, że losowa ścieżka zaczynająca się w V_j nigdy nie wróci do V_j , wynosi $b_j = \det(I - A)/\text{dopełnienie}_{jj}(I - A)$.
- Prawdopodobieństwo, że V_j występuje dokładnie k razy na ścieżce wynosi $a_j(1 - b_j)^{k-1}b_j$, dla $k \geq 1, 1 \leq j \leq n$.

27. [M30] (Stany stacjonarne) Niech G będzie grafem skierowanym o wierzchołkach V_1, \dots, V_n , którego krawędziom przypisano prawdopodobieństwa $p(e)$, jak w ćwiczeniu 26. Zamiast wyróżniania wierzchołków Start i Stop zakładamy jednak, że graf jest silnie spójny. Każdy wierzchołek grafu jest zatem jego korzeniem. Zakładamy, że prawdopodobieństwa $p(e)$ są dodatnie i spełniają warunek $\sum_{\text{init}(e)=V_j} p(e) = 1$ dla wszystkich j .

Mówimy, że proces losowy opisany w ćwiczeniu 26 ma „stany stacjonarne” (x_1, \dots, x_n), jeśli

$$x_j = \sum_{\text{fin}(e)=V_j} p(e) x_{\text{init}(e)}, \quad 1 \leq j \leq n.$$

Niech t_j będzie sumą składników $\prod_{e \in T_j} p(e)$ po wszystkich zorientowanych poddrzewach T_j grafu G zakorzenionych w V_j . Udowodnij, że (t_1, \dots, t_n) jest stanem stacjonarnym procesu losowego.

- 28. [M35] Rozważmy wyznacznik macierzy $(m+n) \times (m+n)$ pokazany dla $m=2$ i $n=3$:

$$\det \begin{pmatrix} a_{10} + a_{11} + a_{12} + a_{13} & 0 & a_{11} & a_{12} & a_{13} \\ 0 & a_{20} + a_{21} + a_{22} + a_{23} & a_{21} & a_{22} & a_{23} \\ b_{11} & b_{12} & b_{10} + b_{11} + b_{12} & 0 & 0 \\ b_{21} & b_{22} & 0 & b_{20} + b_{21} + b_{22} & 0 \\ b_{31} & b_{32} & 0 & 0 & b_{30} + b_{31} + b_{32} \end{pmatrix}.$$

Udowodnij, że jeśli ten wyznacznik rozwiniemy jako wielomian zmiennych a i b (z indeksami), to każdy niezerowy wyraz będzie miał współczynnik $+1$. Ile jest wyrazów w tym rozwinięciu? Podaj regułę (związaną z drzewami zorientowanymi) mówiącą dokładnie, które wyrazy występują w rozwinięciu.

***2.3.4.3. Lemat o drzewach nieskończonych.** Do tej pory skupialiśmy się głównie na drzewach mających jedynie skończoną liczbę wierzchołków, ale podane definicje drzew wolnych i zorientowanych nie narzucały takiego ograniczenia. Nieskończone drzewa *uporządkowane* można zdefiniować na kilka sposobów. Możemy na przykład rozszerzyć koncepcję „notacji dziesiętnej Dewey'a” na nieskończone zbiory liczb, jak w ćwiczeniu 2.3–14. Nawet przy badaniu algorytmów komputerowych czasami przydaje się wiedza na temat własności drzew nieskończonych – na przykład przy udowadnianiu przez sprowadzenie do sprzeczności, że pewne drzewa *nie* są nieskończone. Jedna z podstawowych własności drzew nieskończonych, po raz pierwszy sformułowana w pełnej ogólności przez D. Königa, mówi, że:

Twierdzenie K. (*Lemat o drzewach nieskończonych*). W nieskończonym drzewie zorientowanym o wierzchołkach mających skończone stopnie istnieje nieskończona ścieżka do korzenia, tj. nieskończony ciąg wierzchołków V_0, V_1, V_2, \dots , w którym V_0 jest korzeniem oraz $\text{fin}(e[V_{j+1}]) = V_j$ dla wszystkich $j \geq 0$.

Dowód. Budujemy ścieżkę, zaczynając od V_0 , czyli korzenia drzewa zorientowanego. Zakładamy, że $j \geq 0$ i że wybrano V_j o nieskończonym wielu potomkach. Stopień V_j jest z założenia skończony, zatem V_j ma skończenie wiele dzieci U_1, \dots, U_n . Przynajmniej jedno z tych dzieci musi mieć nieskończonym wielu potomków, zatem na V_{j+1} wybieramy to właśnie dziecko.

V_0, V_1, V_2, \dots jest nieskończoną ścieżką do korzenia. ■

Znawcom analizy matematycznej powyższe rozumowanie zapewne trochę przypomina dowód klasycznego twierdzenia Bolzana–Weierstrassa, „nieskończony i ograniczony zbiór liczb rzeczywistych ma punkt skupienia”. Jak zauważa-

żył König, jednym z możliwych sposobów wyrażenia twierdzenia K jest powiedzenie, że „jeśli gatunek ludzki nie zginie, to jest taka osoba, której ród nie wymrze”.

Wielu osobom twierdzenie K na pierwszy rzut oka wydaje się oczywiste, ale po chwili zastanowienia stwierdzają, że jest ono głębsze, niż się na początku wydawało. Choć stopień każdego wierzchołka jest skończony, nie zakładamy, że jest *ograniczony* (mniejszy od pewnej liczby N wspólnej dla wszystkich wierzchołków), zatem w drzewie mogą być wierzchołki o coraz większych stopniach. Może się wydawać, że potomkowie każdej osoby w końcu wymrą, chociaż niektóre rodziny będą trwać przez miliony pokoleń, niektóre przez biliony itd. H. W. Watson opublikował kiedyś „dowód” twierdzenia, że z pewnych praw biologicznego prawdopodobieństwa wynika, iż w przyszłości urodzi się nieskończonym wiele ludzi, ale każdy ród wymrze z prawdopodobieństwem jeden. Jego praca [J. Anthropological Inst. Gt. Britain and Ireland 4 (1874), 138–144] zawiera ważne i dalekosiązne twierdzenia, ale również małą lukę, która spowodowała pojawienie się powyższego błędu. Istotne jest to, że autorowi jego wnioski nie wydawały się logicznie niespójne.

Do badania algorytmów komputerowych można twierdzenie K zastosować nie wprost: jeżeli wykonanie algorytmu polega na tym, że co jakiś czas zaczynamy wykonywać skończenie wiele podalgorytmów, to jeśli wykonanie każdego łańcucha kolejno uruchamianych podalgorytmów się zakończy, to wykonanie całego algorytmu też się zakończy.

Stwierdzenie to można wyrazić jeszcze inaczej. Przypuśćmy, że mamy skończony lub nieskończony zbiór S , taki że każdy element S jest ciągiem (x_1, x_2, \dots, x_n) dodatnich liczb całkowitych o długości $n \geq 0$. Przyjmujemy, że zachodzą warunki

- i) jeśli (x_1, \dots, x_n) należy do S , to również (x_1, \dots, x_k) dla $0 \leq k \leq n$;
- ii) jeśli (x_1, \dots, x_n) należy do S , to istnieje tylko skończenie wiele x_{n+1} , dla których $(x_1, \dots, x_n, x_{n+1})$ należy do S ;
- iii) nie istnieje nieskończony ciąg (x_1, x_2, \dots) , którego wszystkie początkowe podciągi (x_1, x_2, \dots, x_n) są elementami S .

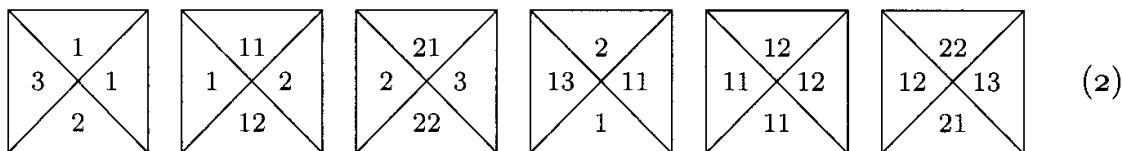
Wówczas S jest niczym innym jak drzewem zorientowanym w notacji dziesiętnej Dewey'a, a z twierdzenia K wiemy, że S jest skończone.

Jeden z najciekawszych przykładów możliwości twierdzenia K pojawia się w związku ze zbiorem interesujących problemów dotyczących pokryć kafelkami, pochodzących od Hao Wanga. *Kafelek* to kwadrat podzielony na cztery części poetykietowane liczbami, na przykład:

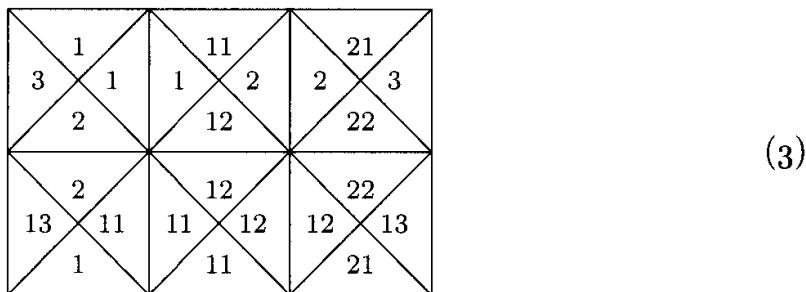


Problem *pokrycia płaszczyzny kafelkami* polega na wzięciu skończonej liczby kafelków, powięleniu każdego nieskończoną liczbę razy i pokryciu nieskończonej płaszczyzny za pomocą takiego zestawu w ten sposób, by kafelki przylegały do

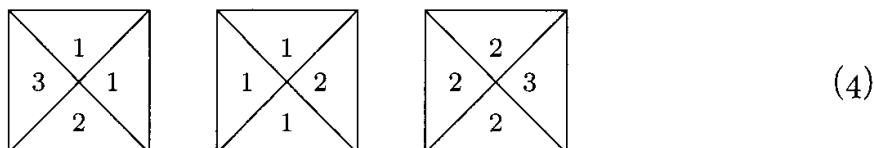
siebie bokami o jednakowych etykietach. Kafelków nie wolno obracać ani wykonywać ich odbić lustrzanych. Można na przykład pokryć płaszczyznę sześcioma kafelkami:



Istnieje tylko jeden schemat takiego pokrycia, polegający na powtarzaniu prostokątnego wzoru



Czytelnik może sprawdzić, że nie da się pokryć płaszczyzny następującymi trzema kafelkami:



Wang zauważył [Scientific American **213**, 5 (November 1965), 98–106], że jeśli da się pokryć pierwszą (prawą górną) ćwiartkę układu współrzędnych, to da się pokryć całą płaszczyznę. Jest to obserwacja nieoczekiwana, ponieważ w metodzie pokrywania ćwiartki płaszczyzny w istotny sposób korzysta się z faktu istnienia „brzegów” wzdłuż osi x i y . Nie widać zatem powodów, dla których za jej pomocą można by wyznaczać pokrycie na przykład drugiej (lewej górnej) ćwiartki (kafelków nie można obracać ani odbijać). Nie pozbędziemy się brzegu przez przesuwanie rozwiązania dla jednej ćwiartki w lewo i w dół, ponieważ przesuwać możemy jedynie o skończoną liczbę kafelków. Ale dowód Wanga przebiega następująco: z istnienia rozwiązania dla pierwszej ćwiartki wynika, że można pokryć kafelkami kwadrat $2n \times 2n$ dla dowolnego n . Zbiór wszystkich pokryć kwadratów o parzystych bokach tworzy drzewo zorientowane, jeśli przyjmiemy, że dziećmi pokrycia $2n \times 2n$ są wszystkie pokrycia $(2n+2) \times (2n+2)$ uzyskane przez otoczenie pokrycia $2n \times 2n$ jednym rzędem kafelków. Korzeniem takiego drzewa zorientowanego jest pokrycie 0×0 ; jego dziećmi są pokrycia 2×2 itd. Każdy węzeł ma tylko skończenie wiele dzieci, ponieważ liczba rodzajów kafelków jest skończona. Na mocy lematu o drzewach nieskończonych istnieje nieskończona ścieżka do korzenia. To oznacza, że istnieje pokrycie całej płaszczyzny (choć nie mamy pojęcia, jak wygląda)!

Nowsze odkrycia w dziedzinie pokryć są przedstawione w pięknej książce B. Grünbauma i G. C. Shepharda *Tilings and Patterns* (Freeman, 1987), rozdział 11.

ĆWICZENIA

1. [M10] W tekście występuje zbiór S zawierający ciągi dodatnich liczb całkowitych. Pojawia się tam stwierdzenie, że S „jest niczym innym, jak drzewem zorientowanym”. Czym są krawędzie i korzeń tego drzewa zorientowanego?
2. [20] Pokaż, że jeśli dopuścimy obracanie kafelków, to zawsze da się pokryć płaszczyznę.
- 3. [M23] Czy jeżeli dopuścimy *nieskończenie* wiele rodzajów kafelków, to z pokrycia pierwszej ćwiartki wynika pokrycie płaszczyzny?
4. [M25] (H. Wang) Sześć kafelków (2) daje pokrycie toroidalne, tj. takie pokrycie, w którym powtarza się pewien wzór, tu (3).

Przyjmij bez dowodu, że jeśli tylko można pokryć płaszczyznę skończonym zestawem kafelków, to istnieje również pokrycie toroidalne kafelkami tych samych rodzajów. Korzystając z tego założenia oraz lematu Königa, zaprojektuj algorytm, który dla specyfikacji dowolnego skończonego zbioru kafelków odpowiada w skończonej liczbie kroków na pytanie, czy istnieje sposób pokrycia płaszczyzny tymi kafelkami.

5. [M40] Pokaż, że za pomocą następującego zestawu 92 kafelków można pokryć płaszczyznę, ale nie w sposób toroidalny (w sensie ćwiczenia 4).

By ułatwić sobie zadanie opisania 92 rodzajów kafelków, wprowadzimy notację pomocniczą: definiujemy „kody podstawowe”:

$$\begin{array}{llll}
 \alpha = (1, 2, 1, 2) & \beta = (3, 4, 2, 1) & \gamma = (2, 1, 3, 4) & \delta = (4, 3, 4, 3) \\
 a = (Q, D, P, R) & b = (\ , \ , L, P) & c = (U, Q, T, S) & d = (\ , \ , S, T) \\
 N = (Y, \ , X, \) & J = (D, U, \ , X) & K = (\ , Y, R, L) & B = (\ , \ , \ , \) \\
 R = (\ , \ , R, R) & L = (\ , \ , L, L) & P = (\ , \ , P, P) & S = (\ , \ , S, S) \\
 & T = (\ , \ , T, T) & X = (\ , \ , X, X) & \\
 Y = (Y, Y, \ , \) & U = (U, U, \ , \) & D = (D, D, \ , \) & Q = (Q, Q, \ , \)
 \end{array}$$

Zestaw 92 kafelków zapisuje się następująco

$$\begin{aligned}
 \alpha\{a, b, c, d\} & & & [4 \text{ kafelki}] \\
 \beta\{Y\{B, U, Q\}\{P, T\}, \{B, U, D, Q\}\{P, S, T\}, K\{B, U, Q\}\} & & [21 \text{ kafelków}] \\
 \gamma\{\{X, B\}\{L, P, S, T\}, R\}\{B, Q\}, J\{L, P, S, T\}\} & & [22 \text{ kafelki}] \\
 \delta\{X\{L, P, S, T\}\{B, Q\}, Y\{B, U, Q\}\{P, T\}, N\{a, b, c, d\}, \\
 & & \\
 & & J\{L, P, S, T\}, K\{B, U, Q\}, \{R, L, P, S, T\}\{B, U, D, Q\}\} & [45 \text{ kafelków}]
 \end{aligned}$$

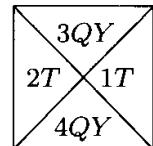
Powyższe skróty oznaczają, że należy „nałożyć na siebie” kody podstawowe i posortować każdy z czterech składników kodu alfabetycznie. Stąd

$$\beta Y\{B, U, Q\}\{P, T\}$$

oznacza sześć kafelków βYBP , βYUP , βYQP , βYBT , βYUT oraz βYQT . Kafelek βYQT to

$$(3, 4, 2, 1)(Y, Y, \ , \)(Q, Q, \ , \)(\ , \ , T, T) = (3QY, 4QY, 2T, 1T)$$

Za pomocą takiego zapisu można przedstawić kafelek pokazany przy prawym marginesie; zamiast etykiet liczbowych używamy kilkuznakowych kombinacji liter i cyfr. Dwa kafelki można położyć obok siebie, jeśli etykiety przy stykających się krawędziach kafelków są dokładnie takie same.



β -kafelek to kafelek, w którego opisie występuje β . Jako punkt wyjściowy rozwiązań przyjmij obserwację, że obaj sąsiedzi β -kafelka po prawej i po lewej muszą być α -kafelkami, a obaj sąsiedzi nad i pod nim muszą być δ -kafelkami. Prawym sąsiadem $\alpha\alpha$ -kafelka musi być βKB lub βKU , lub βKQ , za nim musi stać αb -kafelek itd.

(Powyższa konstrukcja jest uproszczoną wersją konstrukcji autorstwa Roberta Bergera, który udowodnił też, że ogólnego problemu z ćwiczenia 4 bez nieprawdziwego założenia rozwiązać się nie da. Zobacz *Memoirs Amer. Math. Soc.* **66** (1966)).

- 6. [M23] (Otto Schreier) W znanej pracy [Nieuw Archief voor Wiskunde (2) **15** (1927), 212–216] B. L. van der Waerden udowodnił następujące twierdzenie:

Jeśli k i m są dodatnimi liczbami całkowitymi i jeśli mamy k zbiorów S_1, \dots, S_k dodatnich liczb całkowitych, takich że każda dodatnia liczba całkowita należy do przynajmniej jednego z tych zbiorów, to przynajmniej jeden z tych zbiorów S_j zawiera ciąg arytmetyczny długości m .

(Ostatnie stwierdzenie oznacza, że istnieją liczby a oraz $\delta > 0$, takie że $a + \delta, a + 2\delta, \dots, a + m\delta$ należą do S_j). Jeśli to możliwe, to opierając się na tym wyniku oraz na lematie o drzewach nieskończonych, udowodnij następujące, silniejsze twierdzenie:

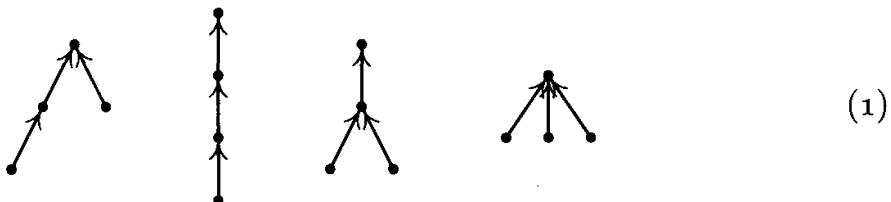
Jeśli k i m są dodatnimi liczbami całkowitymi, to istnieje taka liczba N , że jeśli mamy k zbiorów S_1, \dots, S_k liczb całkowitych, takich że każda liczba całkowita między 1 i N należy do przynajmniej jednego z tych zbiorów, to przynajmniej jeden z tych zbiorów S_j zawiera ciąg arytmetyczny o długości m .

- 7. [M30] Jeśli to możliwe, to korzystając z twierdzenia van der Waerdona z ćwiczenia 6 oraz lematu o drzewach nieskończonych, udowodnij następujące silniejsze, twierdzenie:

Jeśli k jest dodatnią liczbą całkowitą i jeśli mamy k zbiorów S_1, \dots, S_k liczb całkowitych takich, że każda dodatnia liczba całkowita należy do którejś z tych zbiorów, to przynajmniej jeden z tych zbiorów S_j zawiera nieskończony ciąg arytmetyczny.

- 8. [M39] (J. B. Kruskal). Jeśli T i T' są (skończonymi i uporządkowanymi) drzewami, to niech napis $T \subseteq T'$ oznacza, że T można włożyć w T' , jak w ćwiczeniu 2.3.2–22. Udowodnij, że jeśli T_1, T_2, T_3, \dots jest dowolnym nieskończonym ciągiem drzew, to istnieją takie liczby całkowite $j < k$, że $T_j \subseteq T_k$. (Innymi słowy, nie można skonstruować nieskończonego ciągu drzew, w którym żadne drzewo nie zawiera któregoś z poprzednich. Za pomocą tego faktu można udowodnić, że wykonanie pewnych algorytmów musi się zakończyć).

***2.3.4.4. Zliczanie drzew.** Kilka najbardziej pouczających zastosowań matematycznej teorii drzew do analizy algorytmów związanych jest ze wzorami na liczbę różnych drzew konkretnych rodzajów. Wiemy na przykład, że istnieją 4 różne drzewa zorientowane o czterech nieroóżnialnych wierzchołkach:



Zacznijmy od problemu polegającego na wyznaczeniu liczby a_n strukturalnie różnych drzew zorientowanych o n wierzchołkach. Oczywiście $a_1 = 1$. Jeśli

$n > 1$, to drzewo ma korzeń i różne poddrzewa. Przypuśćmy, że jest j_1 poddrzew o jednym wierzchołku, j_2 o dwóch itd. Możemy wówczas wybrać j_k spośród a_k możliwych drzew k -wierzchołkowych na

$$\binom{a_k + j_k - 1}{j_k}$$

sposobów, ponieważ powtórzenia są dozwolone (zobacz ćwiczenie 1.2.6–60), wiemy zatem, że

$$a_n = \sum_{j_1+2j_2+\dots=n-1} \binom{a_1 + j_1 - 1}{j_1} \dots \binom{a_{n-1} + j_{n-1} - 1}{j_{n-1}} \quad \text{dla } n > 1. \quad (2)$$

Jeśli zajmiemy się funkcją tworzącą $A(z) = \sum_n a_n z^n$, gdzie $a_0 = 0$, to stwierdzimy, że tożsamość

$$\frac{1}{(1-z^r)^a} = \sum_j \binom{a+j-1}{j} z^{rj}$$

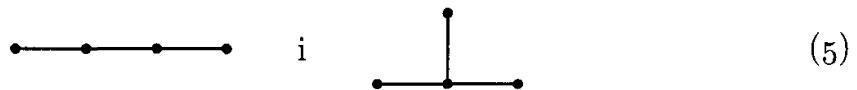
oraz (2) dają

$$A(z) = \frac{z}{(1-z)^{a_1}(1-z^2)^{a_2}(1-z^3)^{a_3}\dots}. \quad (3)$$

To nie jest szczególnie ładna postać $A(z)$, ponieważ zawiera nieskończony iloczyn, a współczynniki a_1, a_2, \dots występują po prawej stronie. Nieco bardziej estetyczny sposób przedstawienia $A(z)$ podajemy w ćwiczeniu 1. Otrzymujemy całkiem efektywny wzór na obliczanie wartości a_n (zobacz ćwiczenie 2); w istocie można go również wykorzystać do wyznaczenia asymptotycznego zachowania a_n dla dużych n (zobacz ćwiczenie 4). Okazuje się, że

$$A(z) = z + z^2 + 2z^3 + 4z^4 + 9z^5 + 20z^6 + 48z^7 + 115z^8 + 286z^9 + 719z^{10} + 1842z^{11} + \dots. \quad (4)$$

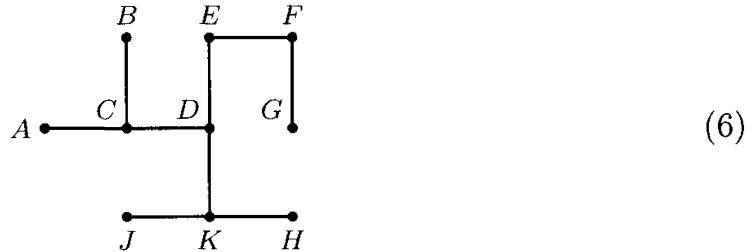
Teraz, gdy w zasadzie wyznaczyliśmy liczbę drzew zorientowanych, można zapytać o liczbę strukturalnie różnych *drzew wolnych* o n wierzchołkach. Istnieją tylko dwa różne drzewa wolne o czterech wierzchołkach, konkretnie



ponieważ pierwsze dwa i ostatnie dwa drzewa zorientowane z (1) stają się identyczne, gdy pominiemy orientację.

Jak wiemy, można wybrać dowolny wierzchołek X drzewa wolnego i dokładnie na jeden sposób ustalić kierunki krawędzi, tak by drzewo stało się drzewem zorientowanym o korzeniu X . Założymy, że po przeprowadzeniu takiej operacji dla wierzchołka X istnieje k poddrzew korzenia X mających odpowiednio

s_1, s_2, \dots, s_k wierzchołków. Jest jasne, że k jest liczbą krawędzi o końcu w X i $s_1 + s_2 + \dots + s_k = n - 1$. Przy takich założeniach mówimy, że *waga* wierzchołka X , oznaczana $\text{weight}(X)$, jest równa $\max(s_1, s_2, \dots, s_k)$. Zatem w drzewie



waga wierzchołka D wynosi 3 (każde z poddrzew D zawiera trzy spośród dziesięciu pozostałych wierzchołków), a waga wierzchołka E wynosi $\max(7, 2) = 7$. Wierzchołek o najmniejszej wadze jest nazywany *centroidem* drzewa wolnego.

Niech X i s_1, s_2, \dots, s_k oznaczają to, co powyżej, i niech Y_1, Y_2, \dots, Y_k będą korzeniami poddrzew wierzchołka X . Waga Y_1 musi wynosić przynajmniej $n - s_1 = 1 + s_2 + \dots + s_k$, ponieważ gdyby Y_1 było korzeniem, to w poddrzewie wierzchołka X byłoby $n - s_1$ wierzchołków. Jeżeli w poddrzewie o korzeniu Y_1 jest centroid Y , to mamy

$$\text{weight}(X) = \max(s_1, s_2, \dots, s_k) \geq \text{weight}(Y) \geq 1 + s_2 + \dots + s_k$$

i jest to możliwe jedynie wtedy, gdy $s_1 > s_2 + \dots + s_k$. Można uzyskać podobny wynik, zastępując w powyższym rozumowaniu Y_1 przez Y_j . Zatem *najwyżej jedno z poddrzew wierzchołka może zawierać centroid*.

To jest silny warunek, ponieważ wynika z niego, że w drzewie wolnym są *co najwyżej dwa centroidy, a jeśli są dwa, to sąsiadują ze sobą*. (Zobacz ćwiczenie 9).

W drugą stronę, jeśli $s_1 > s_2 + \dots + s_k$, to w poddrzewie Y_1 *istnieje* centroid, ponieważ

$$\text{weight}(Y_1) \leq \max(s_1 - 1, 1 + s_2 + \dots + s_k) \leq s_1 = \text{weight}(X),$$

a waga wszystkich wierzchołków w poddrzewach Y_2, \dots, Y_k wynosi przynajmniej $s_1 + 1$. Udowodniliśmy, że *wierzchołek X jest jedynym centroidem drzewa wolnego wtedy i tylko wtedy, gdy*

$$s_j \leq s_1 + \dots + s_k - s_j \quad \text{dla } 1 \leq j \leq k. \quad (7)$$

Stąd liczba drzew wolnych o n wierzchołkach mających tylko jeden centroid jest liczbą drzew zorientowanych o n wierzchołkach minus liczba takich drzew naruszających warunek (7). Drzewa naruszające ten warunek składają się w istocie z drzewa zorientowanego o s_j wierzchołkach oraz innego drzewa zorientowanego o $n - s_j \leq s_j$ wierzchołkach. Liczba drzew z jednym centroidem jest zatem równa

$$a_n - a_1 a_{n-1} - a_2 a_{n-2} - \dots - a_{\lfloor n/2 \rfloor} a_{\lceil n/2 \rceil}. \quad (8)$$

Wolne drzewo z dwoma centroidami ma parzystą liczbę wierzchołków, a waga każdego centroidu wynosi $n/2$ (zobacz ćwiczenie 10). Jeśli zatem $n = 2m$, to

liczba drzew wolnych z dwoma centroidami jest liczbą wyborów z powtórzeniami dwóch przedmiotów spośród a_m , konkretnie

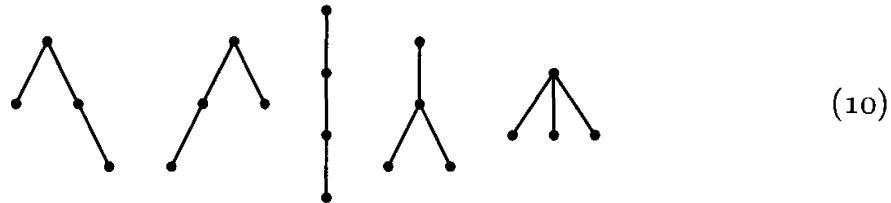
$$\binom{a_m + 1}{2}.$$

By otrzymać całkowitą liczbę drzew wolnych, dodajemy $\frac{1}{2}a_{n/2}(a_{n/2} + 1)$ do (8) dla parzystego n . Postać wzoru (8) sugeruje istnienie prostej funkcji tworzącej. W istocie okazuje się, że funkcją tworzącą liczb strukturalnie różnych drzew wolnych jest

$$\begin{aligned} F(z) &= A(z) - \frac{1}{2}A(z)^2 + \frac{1}{2}A(z^2) \\ &= z + z^2 + z^3 + 2z^4 + 3z^5 + 6z^6 + 11z^7 + 23z^8 \\ &\quad + 47z^9 + 106z^{10} + 235z^{11} + \dots. \end{aligned} \quad (9)$$

Odkrycie tego prostego związku między $F(z)$ i $A(z)$ jest zasługą przede wszystkim C. Jordana, który zajmował się tym problemem w 1869 roku.

Przyjrzyjmy się teraz problemom zliczania *drzew uporządkowanych*, które stanowią centrum naszych zainteresowań ze względu na zastosowania w algorytmach komputerowych. Istnieje pięć strukturalnie różnych drzew uporządkowanych o czterech wierzchołkach



Dwa pierwsze są jednakowe jako drzewa zorientowane, zatem tylko jedno z nich pojawiło się w (1).

Zanim zajmiemy się ogólnym problemem wyznaczenia liczby różnych kształtów drzew uporządkowanych, skupmy się na *drzewach binarnych*, ponieważ są bliższe rzeczywistym strukturam danych i łatwiejsze do analizowania. Niech b_n oznacza liczbę różnych drzew binarnych o n węzłach. Z definicji drzewa binarnego wynika, że $b_0 = 1$, a dla $n > 0$ liczba możliwości jest liczbą sposobów umieszczenia drzewa binarnego o k węzłach na lewo od korzenia i drzewa binarnego o $(n - 1 - k)$ węzłach na prawo od korzenia. Zatem

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \dots + b_{n-1} b_0, \quad n \geq 1. \quad (11)$$

Z tego związku wynika, że funkcja tworząca

$$B(z) = b_0 + b_1 z + b_2 z^2 + \dots$$

spełnia równanie

$$zB(z)^2 = B(z) - 1. \quad (12)$$

Rozwiążując równanie kwadratowe oraz korzystając z faktu, że $B(0) = 1$, otrzymujemy

$$\begin{aligned}
 B(z) &= \frac{1}{2z} (1 - \sqrt{1 - 4z}) = \frac{1}{2z} \left(1 - \sum_{k \geq 0} \binom{\frac{1}{2}}{k} (-4z)^k \right) \\
 &= 2 \sum_{n \geq 0} \binom{\frac{1}{2}}{n+1} (-4z)^n = \sum_{n \geq 0} \binom{-\frac{1}{2}}{n} \frac{(-4z)^n}{n+1} \\
 &= \sum_{n \geq 0} \binom{2n}{n} \frac{z^n}{n+1} \\
 &= 1 + z + 2z^2 + 5z^3 + 14z^4 + 42z^5 + 132z^6 + 429z^7 \\
 &\quad + 1430z^8 + 4862z^9 + 16796z^{10} + \dots \quad (13)
 \end{aligned}$$

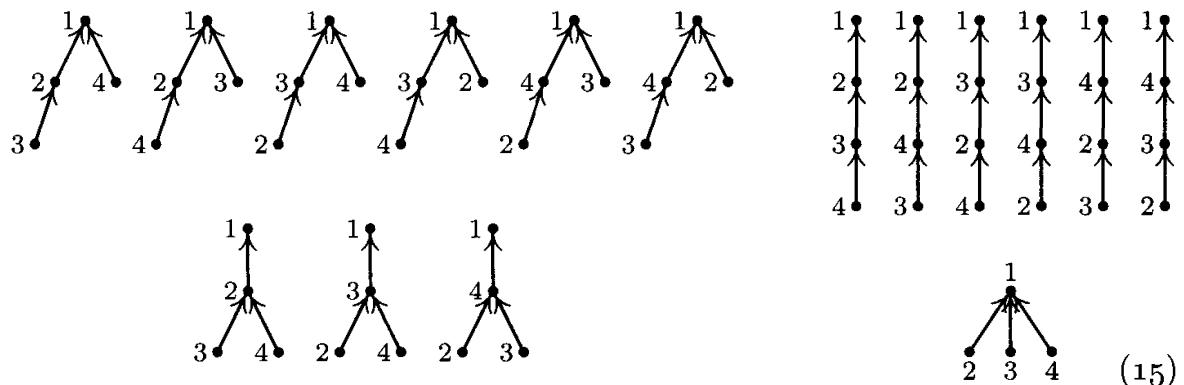
(Zobacz ćwiczenie 1.2.6–47). Poszukiwaną odpowiedzią jest zatem

$$b_n = \frac{1}{n+1} \binom{2n}{n}. \quad (14)$$

Na mocy wzoru Stirlinga jest to w przybliżeniu $4^n/n\sqrt{\pi n} + O(4^n n^{-5/2})$. Niektóre ważne uogólnienia wzoru (14) pojawiają się w ćwiczeniach 11 i 32.

Powróćmy do naszego pytania o drzewa uporządkowane o n węzłach. Łatwo zauważać, że jest to dokładnie pytanie o liczbę drzew binarnych, ponieważ istnieje naturalna odpowiedniość między drzewami binarnymi a lasami, a drzewo bez korzenia jest lasem. Stąd liczba drzew uporządkowanych o n wierzchołkach równa się b_{n-1} , liczba drzew binarnych o $n-1$ wierzchołkach.

W powyższych wyliczeniach zakładamy, że wierzchołki są nieroróżnicjalne. Jeśli w (1) poetykietowalibyśmy wierzchołki liczbami 1, 2, 3, 4 i wymagali, by 1 był korzeniem, to otrzymalibyśmy 16 różnych drzew zorientowanych:



Tak postawione pytanie o liczbę różnych drzew etykietowanych jest zupełnie inne niż pytanie, na które przed momentem odpowiedzieliśmy. W tym przypadku można je sformułować następująco: „Wyobraźmy sobie, że rysujemy po jednej strzałce od wierzchołka 2, 3 i 4 do innego wierzchołka. Strzałkę od każdego wierzchołka możemy narysować na trzy sposoby, zatem w sumie sposobów jest $3^3 = 27$. Ile z tych 27 sposobów daje drzewo zorientowane o korzeniu 1?”. Jak zauważylismy odpowiedź brzmi „16”. Analogiczne sformułowanie tego samego

problemu, tym razem dla przypadku n wierzchołków, brzmi tak: „Niech $f(x)$ będzie taką funkcją o wartościach całkowitych, że $f(1) = 1$ i $1 \leq f(x) \leq n$ dla wszystkich całkowitych $1 \leq x \leq n$. Mówimy, że f jest *odwzorowaniem drzewowym*, jeśli $f^{[n]}(x)$, tj. $f(f(\dots(f(x))\dots))$ złożone n razy, równa się 1 dla dowolnego x . Ile jest odwzorowań drzewowych?”. Taki problem pojawia się na przykład w związku z generowaniem liczb losowych. Okaże się, że średnio jedna na n takich funkcji f jest odwzorowaniem drzewowym.

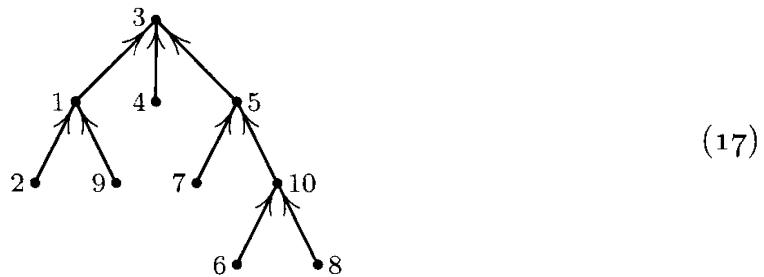
Rozwiążanie powyższego problemu zliczania można uzyskać, stosując ogólne wzory na zliczanie poddrzew grafów wyprowadzone w poprzednim punkcie (zobacz ćwiczenie 12). Ale istnieje o wiele ciekawsze rozwiązanie, pokazujące nowy, zwarte sposób reprezentowania kształtu drzewa zorientowanego.

Przypuśćmy, że mamy drzewo zorientowane o wierzchołkach $\{1, 2, \dots, n\}$ oraz $n - 1$ krawędziach, gdzie krawędź prowadzi od j do $f(j)$ dla wszystkich j poza korzeniem. Istnieje przynajmniej jeden liść. Niech V_1 będzie liściem o najmniejszym numerze. Jeśli $n > 1$, to zapisujemy $f(V_1)$ i usuwamy z drzewa wierzchołek V_1 oraz krawędź $V_1 \rightarrow f(V_1)$. Niech V_2 będzie liściem o najmniejszym numerze w tak otrzymanym drzewie. Jeśli $n > 2$, to zapisujemy $f(V_2)$ i usuwamy z drzewa wierzchołek V_2 oraz krawędź $V_2 \rightarrow f(V_2)$. Kontynuujemy ten proces, aż do usunięcia z drzewa wszystkich krawędzi i wierzchołków poza korzeniem. Otrzymany ciąg $n - 1$ liczb

$$f(V_1), f(V_2), \dots, f(V_{n-1}), \quad 1 \leq f(V_j) \leq n, \quad (16)$$

nazywamy *reprezentacją kanoniczną* drzewa zorientowanego.

Na przykład drzewo zorientowane o 10 wierzchołkach



ma reprezentację kanoniczną $1, 3, 10, 5, 10, 1, 3, 5, 3$.

Istotny jest fakt, że możliwa odwrócić proces i z dowolnego ciągu $n - 1$ liczb określonych wzorem (16) otrzymać z powrotem drzewo zorientowane. Ograniczmy naszą uwagę do liczb z przedziału od 1 do n i weźmy dowolny ciąg x_1, x_2, \dots, x_{n-1} . Niech V_1 oznacza najmniejszą liczbę nie występującą w ciągu x_1, \dots, x_{n-1} ; niech V_2 będzie najmniejszą liczbą $\neq V_1$ nie występującą w ciągu x_2, \dots, x_{n-1} itd. Otrzymawszy permutację $V_1 V_2 \dots V_n$ liczb z przedziału $\{1, 2, \dots, n\}$ dla $1 \leq j < n$, rysujemy krawędź od wierzchołka V_j do wierzchołka x_j . W ten sposób dostajemy graf skierowany bez cykli zorientowanych, a na mocy ćwiczenia 2.3.4.2–7 jest to drzewo zorientowane. Jest jasne, że ciąg x_1, x_2, \dots, x_{n-1} jest ciągiem określonym wzorem (16) dla tego drzewa zorientowanego.

Widzimy zatem, że proces jest odwracalny. Otrzymaliśmy wzajemnie jednoznaczną odpowiedniość między $(n - 1)$ -elementowymi krotkami liczb z przedziału

$\{1, 2, \dots, n\}$ i drzewami zorientowanymi o takich wierzchołkach. Stąd istnieje n^{n-1} różnych drzew zorientowanych o n etykietowanych wierzchołkach. Jeśli przyjmiemy, że pewien wierzchołek ma być korzeniem, to widać, że nie ma różnicy między wierzchołkami, zatem jest n^{n-2} różnych drzew zorientowanych o wierzchołkach $\{1, 2, \dots, n\}$, jeśli ustalimy korzeń. To daje $16 = 4^{4-2}$ drzew w (15). Na podstawie tej informacji łatwo wyznaczyć liczbę drzew wolnych o etykietowanych wierzchołkach (zobacz ćwiczenie 22). Równie łatwo wyznaczyć liczbę drzew uporządkowanych o etykietowanych wierzchołkach, gdy znamy rozwiązanie analogicznego problemu dla drzew bez etykiet (zobacz ćwiczenie 23). Rozwiązaliśmy zatem problem zliczania dla trzech podstawowych typów drzew zarówno z etykietami, jak i bez.

Warto zastanowić się, co się stanie, jeśli wykorzystamy funkcje tworzące do problemu zliczania etykietowanych drzew zorientowanych. W tym celu najwygodniej przyjąć, że $r(n, q)$ oznacza liczbę etykietowanych grafów skierowanych o n wierzchołkach nie zawierających cykli zorientowanych, takich że z każdego spośród q wyróżnionych wierzchołków wychodzi jedna krawędź. Liczba etykietowanych drzew zorientowanych o ustalonym korzeniu równa się $r(n, n - 1)$. Proste rozumowanie pozwala przekonać się, że dla dowolnej ustalonej liczby całkowitej m

$$r(n, q) = \sum_k \binom{q}{k} r(m+k, k) r(n-m-k, q-k), \quad \text{jeśli } 0 \leq m \leq n-q, \quad (18)$$

$$r(n, q) = \sum_k \binom{q}{k} r(n-1, q-k), \quad \text{jeśli } q = n-1. \quad (19)$$

Pierwszy z tych związków otrzymujemy, dzieląc niewyróżnione wierzchołki na dwie grupy A i B , biorąc m wierzchołków do A i $n - q - m$ wierzchołków do B . Następnie q wyróżnionych wierzchołków dzielimy na grupę k wierzchołków rozpoczynających ścieżki wiodące do A oraz $q - k$ wierzchołków rozpoczynających ścieżki wiodące do B . Związek (19) otrzymujemy, rozważając drzewa zorientowane o korzeniu stopnia k .

Postać powyższych zależności wskazuje na to, że efekty przynieść może posłużenie się funkcją tworzącą

$$G_m(z) = r(m, 0) + r(m+1, 1)z + \frac{r(m+2, 2)z^2}{2!} + \dots = \sum_k \frac{r(k+m, k)z^k}{k!}.$$

W tym świetle wzór (18) mówi, że $G_{n-q}(z) = G_m(z)G_{n-q-m}(z)$, a stąd na mocy indukcji względem m mamy $G_m(z) = G_1(z)^m$. Ze wzoru (19) otrzymujemy

$$\begin{aligned} G_1(z) &= \sum_{n \geq 1} \frac{r(n, n-1)z^{n-1}}{(n-1)!} = \sum_{k \geq 0} \sum_{n \geq 1} \frac{r(n-1, n-1-k)z^{n-1}}{k! (n-1-k)!} \\ &= \sum_{k \geq 0} \frac{z^k}{k!} G_k(z) = \sum_{k \geq 0} \frac{(zG_1(z))^k}{k!} = e^{zG_1(z)}. \end{aligned}$$

Innymi słowy, kładąc $G_1(z) = w$, sprowadzamy problem do rozwiązyania równania przestępnego

$$w = e^{zw}. \quad (20)$$

To równanie można rozwiązać za pomocą wzoru inwersyjnego Lagrange'a. Podstawiając $z = \zeta/f(\zeta)$, otrzymamy

$$\zeta = \sum_{n \geq 1} \frac{z^n}{n!} g_n^{(n-1)}(0), \quad (21)$$

gdzie $g_n(\zeta) = f(\zeta)^n$, a f jest analityczna w otoczeniu początku układu współrzędnych oraz $f(0) \neq 0$ (zobacz ćwiczenie 4.7–16). W tym przypadku możemy podstawić $\zeta = zw$, $f(\zeta) = e^\zeta$ i wywnioskować rozwiązanie

$$w = \sum_{n \geq 0} \frac{(n+1)^{n-1}}{n!} z^n, \quad (22)$$

zgodnie z uzyskaną powyżej odpowiedzią.

G. N. Raney pokazał, że możemy tak rozszerzyć tę metodę, by wprost otrzymać szereg potęgowy będący rozwinięciem rozwiązania o wiele bardziej ogólnego równania

$$w = y_1 e^{z_1 w} + y_2 e^{z_2 w} + \cdots + y_s e^{z_s w},$$

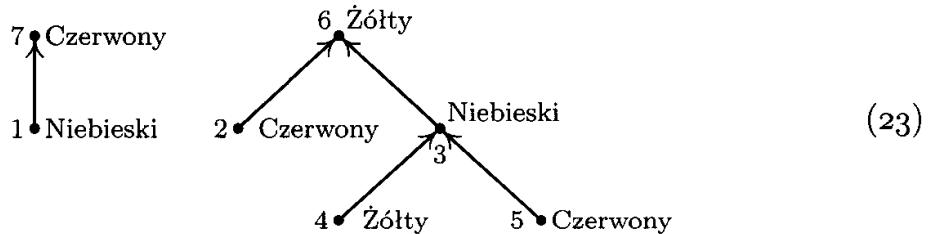
rozwiązuje ją dla w względem szeregiów potęgowych zmiennych y_1, \dots, y_s oraz z_1, \dots, z_s . Na potrzeby tego uogólnienia rozważmy s -wymiarowe wektory liczb całkowitych

$$\mathbf{n} = (n_1, n_2, \dots, n_s)$$

i zapiszmy dla wygody

$$\sum \mathbf{n} = n_1 + n_2 + \cdots + n_s.$$

Przypuśćmy, że mamy s kolorów C_1, C_2, \dots, C_s i rozważmy skierowane grafy, w których każdemu wierzchołkowi przypisano kolor; na przykład,



Niech $r(\mathbf{n}, \mathbf{q})$ będzie liczbą sposobów narysowania krawędzi i przyporządkowania kolorów wierzchołkom $\{1, 2, \dots, n\}$ w ten sposób, że

- i) dla $1 \leq i \leq s$ istnieje dokładnie n_i wierzchołków koloru C_i (stąd $n = \sum \mathbf{n}$);
- ii) jest q krawędzi, po jednej wychodzącej z każdego wierzchołka $\{1, 2, \dots, q\}$;
- iii) dla $1 \leq i \leq s$ istnieje dokładnie q_i krawędzi prowadzących do wierzchołków koloru C_i (stąd $q = \sum \mathbf{q}$);
- iv) nie ma cykli zorientowanych (stąd $q < n$, chyba że $q = n = 0$).

Nazwijmy to (\mathbf{n}, \mathbf{q}) -konstrukcją.

Jeśli na przykład $C_1 = \text{czerwony}$, $C_2 = \text{żółty}$, $C_3 = \text{niebieski}$, to graf (23) jest $((3, 2, 2), (1, 2, 2))$ -konstrukcją. Szczególny przypadek jednego koloru to problem rozwiązyany wcześniej dla drzew zorientowanych. Pomysł Raney'a polegał na uogólnieniu przypadku jednowymiarowego na s wymiarów.

Niech \mathbf{n} i \mathbf{q} będą ustalonymi s -elementowymi wektorami nieujemnych liczb całkowitych i niech $n = \sum \mathbf{n}$, $q = \sum \mathbf{q}$. Dla każdej (\mathbf{n}, \mathbf{q}) -konstrukcji oraz dowolnej liczby k , $1 \leq k \leq n$, definiujemy *reprezentację kanoniczną* złożoną z czterech obiektów:

- liczby t , takie że $q < t \leq n$;
- ciągu n próbek koloru, zawierającego n_i próbek koloru C_i ;
- ciągu q próbek koloru, zawierającego q_i próbek koloru C_i ;
- ciągu q_i elementów zbioru $\{1, 2, \dots, n_i\}$, dla $1 \leq i \leq s$.

Reprezentację kanoniczną definiujemy następująco: zaczynamy od wypisania wierzchołków $\{1, 2, \dots, q\}$ w porządku V_1, V_2, \dots, V_q reprezentacji kanonicznej drzew zorientowanych (zdefiniowanej wcześniej w niniejszym rozdziale). Następnie pod wierzchołkiem V_j piszemy numer $f(V_j)$ wierzchołka, do którego prowadzi krawędź wychodząca z V_j . Niech $t = f(V_q)$ i niech ciąg próbek kolorów (c) składa się odpowiednio z kolorów wierzchołków $f(V_1), \dots, f(V_q)$. Niech ciąg próbek kolorów (b) składa się odpowiednio z kolorów wierzchołków $k, k+1, \dots, n, 1, \dots, k-1$. Wreszcie niech i -ty ciąg w (d) będzie postaci $x_{i1}, x_{i2}, \dots, x_{iq_i}$, gdzie $x_{ij} = m$, jeśli j -ty element o kolorze C_i w ciągu $f(V_1), \dots, f(V_q)$ jest m -tym elementem koloru C_i w ciągu $k, k+1, \dots, n, 1, \dots, k-1$.

Rozważmy na przykład konstrukcję grafu (23) i niech $k = 3$. Zaczynamy od wypisania V_1, \dots, V_5 oraz $f(V_1), \dots, f(V_5)$ (pod spodem):

$$\begin{array}{ccccc} 1 & 2 & 4 & 5 & 3 \\ 7 & 6 & 3 & 3 & 6 \end{array}$$

W tym przypadku $t = 6$, a ciąg (c) zawiera próbki kolorów wierzchołków 7, 6, 3, 3, 6, tj. czerwony, żółty, niebieski, niebieski, żółty. Ciąg (b) zawiera próbki kolorów wierzchołków 3, 4, 5, 6, 7, 1, 2, tj. niebieski, żółty, czerwony, żółty, czerwony, niebieski, czerwony. Na koniec, by wyznaczyć ciągi w (d), postępujemy tak:

	<i>elementy tego koloru</i>	<i>elementy tego koloru</i>	<i>zakoduj kolumnę 3 za pomocą kolumny 2</i>
<i>kolor</i>	<i>w</i> 3, 4, 5, 6, 7, 1, 2	<i>w</i> 7, 6, 3, 3, 6	
czerwony	5, 7, 2	7	2
żółty	4, 6	6, 6	2, 2
niebieski	3, 1	3, 3	1, 1

Otrzymujemy ciągi: 2; 2, 2 i 1, 1.

Z danej reprezentacji kanonicznej możemy odtworzyć zarówno wyjściową (\mathbf{n}, \mathbf{q}) -konstrukcję, jak i liczbę k w następujący sposób: z (a) i (c) znamy kolor wierzchołka t . Ostatni element ciągu w (d) dla tego koloru oraz (b) wyznaczają położenie t w ciągu $k, \dots, n, 1, \dots, k-1$; stąd znamy k oraz kolory wszystkich wierzchołków. Podciągi w (d) oraz (b) i (c) wyznaczają $f(V_1), f(V_2), \dots, f(V_q)$,

a wreszcie odtwarzamy graf, rozmieszczając wierzchołki V_1, \dots, V_q , tak jak robiliśmy to dla drzew zorientowanych.

Dzięki odwrotności reprezentacji kanonicznej możemy wyznaczyć liczbę (\mathbf{n}, \mathbf{q}) -konstrukcji. Jest możliwych $n - q$ wyborów (a); liczba wyborów (b) jest opisana współczynnikiem wielomianowym

$$\binom{n}{n_1, \dots, n_s}$$

podobnie jak liczba wyborów (c)

$$\binom{q}{q_1, \dots, q_s}$$

wreszcie jest $n_1^{q_1} n_2^{q_2} \dots n_s^{q_s}$ wyborów (d). Dzielimy to przez n wyborów k , otrzymując wynik

$$r(\mathbf{n}, \mathbf{q}) = \frac{n - q}{n} \frac{n!}{n_1! \dots n_s!} \frac{q!}{q_1! \dots q_s!} n_1^{q_1} n_2^{q_2} \dots n_s^{q_s}. \quad (24)$$

Co więcej, możemy wyprowadzić odpowiedniki równości (18) i (19):

$$r(\mathbf{n}, \mathbf{q}) = \sum_{\substack{\mathbf{k}, \mathbf{t} \\ \sum(\mathbf{t}-\mathbf{k})=m}} \binom{\sum \mathbf{q}}{\sum \mathbf{k}} r(\mathbf{t}, \mathbf{k}) r(\mathbf{n} - \mathbf{t}, \mathbf{q} - \mathbf{k}), \quad \text{jeżeli } 0 \leq m \leq \sum(\mathbf{n} - \mathbf{q}), \quad (25)$$

przyjmując, że $r(\mathbf{0}, \mathbf{0}) = 1$, a także $r(\mathbf{n}, \mathbf{q}) = 0$, jeśli którykolwiek z n_i bądź q_i jest ujemne lub jeśli $q > n$;

$$r(\mathbf{n}, \mathbf{q}) = \sum_{i=1}^s \sum_k \binom{\sum \mathbf{q}}{k} r(\mathbf{n} - \mathbf{e}_i, \mathbf{q} - k\mathbf{e}_i), \quad \text{jeżeli } \sum \mathbf{n} = 1 + \sum \mathbf{q}, \quad (26)$$

gdzie \mathbf{e}_i jest wektorem mającym 1 na współrzędnej i oraz zera na wszystkich pozostałych współrzędnych. Związek (25) jest oparty na podziale wierzchołków $\{q + 1, \dots, n\}$ na dwie części po m i $n - q - m$ elementów. Drugi związek wyprowadzamy, rozważając strukturę powstającą po usunięciu jedynego korzenia. Otrzymujemy następujący wynik:

Twierdzenie R (George N. Raney, Canadian J. Math. **16** (1964), 755–762).
Niech

$$w = \sum_{\substack{\mathbf{n}, \mathbf{q} \\ \sum(\mathbf{n}-\mathbf{q})=1}} \frac{r(\mathbf{n}, \mathbf{q})}{(\sum \mathbf{q})!} y_1^{n_1} \dots y_s^{n_s} z_1^{q_1} \dots z_s^{q_s}, \quad (27)$$

gdzie $r(\mathbf{n}, \mathbf{q})$ jest zdefiniowane przez (24), a \mathbf{n}, \mathbf{q} są s -wymiarowymi wektorami liczb całkowitych.

Wówczas w spełnia tożsamość

$$w = y_1 e^{z_1 w} + y_2 e^{z_2 w} + \dots + y_s e^{z_s w}. \quad (28)$$

Dowód. Na mocy (25) oraz indukcji względem m

$$w^m = \sum_{\substack{\mathbf{n}, \mathbf{q} \\ \sum(\mathbf{n}-\mathbf{q})=m}} \frac{r(\mathbf{n}, \mathbf{q})}{(\sum \mathbf{q})!} y_1^{n_1} \dots y_s^{n_s} z_1^{q_1} \dots z_s^{q_s}. \quad (29)$$

Następnie na mocy wzoru (26)

$$\begin{aligned} w &= \sum_{i=1}^s \sum_k \sum_{\substack{\mathbf{n}, \mathbf{q} \\ \sum(\mathbf{n}-\mathbf{q})=1}} \frac{r(\mathbf{n} - \mathbf{e}_i, \mathbf{q} - k\mathbf{e}_i)}{k! (\sum \mathbf{q} - k)!} y_1^{n_1} \dots y_s^{n_s} z_1^{q_1} \dots z_s^{q_s} \\ &= \sum_{i=1}^s \sum_k \frac{1}{k!} y_i z_i^k \sum_{\substack{\mathbf{n}, \mathbf{q} \\ \sum(\mathbf{n}-\mathbf{q})=k}} \frac{r(\mathbf{n}, \mathbf{q})}{(\sum \mathbf{q})!} y_1^{n_1} \dots y_s^{n_s} z_1^{q_1} \dots z_s^{q_s} \\ &= \sum_{i=1}^s \sum_k \frac{1}{k!} y_i z_i^k w^k. \quad \blacksquare \end{aligned}$$

Przypadek szczególny $s = 1$ i $z_1 = 1$ we wzorach (27) i (28) ma istotne znaczenie praktyczne i stał się sławny jako „funkcja drzewowa”

$$T(y) = \sum_{n \geq 1} \frac{n^{n-1}}{n!} y^n = y e^{T(y)}. \quad (30)$$

Zobacz Corless, Gonnet, Hare, Jeffrey, Knuth, *Advances in Computational Math.* **5** (1996), 329–359, gdzie omówiono historię tej funkcji i jej ważniejsze własności.

Przegląd wzorów na zliczanie drzew opartych na sprytnych manipulacjach funkcjami tworzącymi jest zamieszczony w pracy: I. J. Gooda [Proc. Cambridge Philos. Soc. **61** (1965), 499–517; **64** (1968), 489]. Matematyczna teoria gatunków rozwinięta ostatnio przez André Joyala [Advances in Math. **42** (1981), 1–82] proponuje abstrakcyjne ujęcie, według którego funkcje tworzące odpowiadają wprost kombinatorycznym własnościom struktur. Książka *Combinatorial Species and Tree-like Structures* F. Bergerona, G. Labelle'a, P. Leroux (Cambridge Univ. Press, 1998) zawiera wiele przykładów z tej pięknej i ciekawej teorii, uogólniających wzory wyrowadzone powyżej.

ĆWICZENIA

1. [M20] (G. Pólya) Pokaż, że

$$A(z) = z \cdot \exp(A(z) + \frac{1}{2} A(z^2) + \frac{1}{3} A(z^3) + \dots).$$

[Wskazówka: Złogarytmuj (3)].

2. [HM24] (R. Otter) Pokaż, że liczba a_n spełnia następujący warunek:

$$na_{n+1} = a_1 s_{n1} + 2a_2 s_{n2} + \dots + na_n s_{nn},$$

gdzie

$$s_{nk} = \sum_{1 \leq j \leq n/k} a_{n+1-jk}.$$

(Te wzory przydają się do obliczania a_n , bo $s_{nk} = s_{(n-k)k} + a_{n+1-k}$).

3. [M40] Napisz program, który wyznacza liczbę (nieetykietowanych) drzew wolnych oraz drzew zorientowanych o n wierzchołkach dla $n \leq 100$. (Skorzystaj z wyniku

ćwiczenia 2). Zbadaj własności arytmetyczne tych liczb. Czy da się coś powiedzieć o ich czynnikach pierwszych albo ich resztach modulo p ?

- 4. [HM39] (G. Pólya, 1937) Posługując się teorią zmiennej zespolonej w następujący sposób, wyznacz asymptotyczną wartość liczby drzew zorientowanych:

- Pokaż, że istnieje liczba rzeczywista α między 0 i 1, dla której $A(z)$ ma promień zbieżności α i $A(z)$ zbiega bezwzględnie dla wszystkich zespolonych z , takich że $|z| \leq \alpha$, przyjmując maksymalną wartość $A(\alpha) = a < \infty$. [Wskazówka: Jeśli szereg potęgowy ma nieujemne współczynniki, to ma jeden dodatni, rzeczywisty punkt osobliwy. Pokaż, że $A(z)/z$ jest ograniczone przy $z \rightarrow \alpha^-$, korzystając z tożsamości z ćwiczenia 1].
- Niech

$$F(z, w) = \exp(zw + \frac{1}{2}A(z^2) + \frac{1}{3}A(z^3) + \dots) - w.$$

Pokaż, że w otoczeniu $(z, w) = (\alpha, a/\alpha)$ funkcja $F(z, w)$ jest analityczna względem każdej zmiennej z osobna.

- Pokaż, że w punkcie $(z, w) = (\alpha, a/\alpha)$, mamy $\partial F / \partial w = 0$; stąd $a = 1$.
- Pokaż, że w punkcie $(z, w) = (\alpha, 1/\alpha)$

$$\frac{\partial F}{\partial z} = \beta = \alpha^{-2} + \sum_{k \geq 2} \alpha^{k-2} A'(\alpha^k) \quad \text{i} \quad \frac{\partial^2 F}{\partial w^2} = \alpha.$$

- Pokaż, że jeśli $|z| = \alpha$ i $z \neq \alpha$, to $\partial F / \partial w \neq 0$; stąd $A(z)$ ma tylko jeden punkt osobliwy dla $|z| = \alpha$.
- Udowodnij, że istnieje obszar większy niż $|z| < \alpha$, w którym

$$\frac{1}{z} A(z) = \frac{1}{\alpha} - \sqrt{2\beta(1 - z/\alpha)} + (1 - z/\alpha)R(z),$$

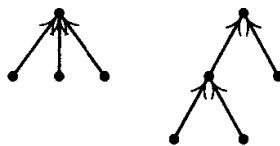
gdzie $R(z)$ jest funkcją funkcją $\sqrt{z - \alpha}$.

- Udowodnij, że

$$a_n = \frac{1}{\alpha^{n-1} n} \sqrt{\beta/2\pi n} + O(n^{-5/2} \alpha^{-n}).$$

(Uwaga: $1/\alpha \approx 2.955765285652$, $\alpha\sqrt{\beta/2\pi} \approx 0.439924012571$).

- 5. [M25] (A. Cayley) Niech c_n będzie liczbą (nieetykietowanych) drzew zorientowanych o n liściach (tj. n wierzchołkach stopnia zero), takich że każdy wierzchołek wewnętrzny ma przynajmniej dwa poddrzewa. Na przykład $c_3 = 2$:



Znайдź wzór analogiczny do (3) na funkcję tworzącą

$$C(z) = \sum_n c_n z^n.$$

6. [M25] Niech „zorientowane drzewo binarne” oznacza drzewo zorientowane, w którym każdy wierzchołek ma stopień wejściowy nie większy niż dwa. Znайдź względnie prosty związek wyznaczający funkcję tworzącą $G(z)$ liczby różnych zorientowanych drzew binarnych o n wierzchołkach i wyznacz kilka początkowych wartości tej liczby.

7. [HM40] Wyprowadź wzór asymptotyczny dla liczb z ćwiczenia 6. (Zobacz ćwiczenie 4).

8. [20] Zgodnie ze wzorem (9) istnieje sześć drzew wolnych o sześciu wierzchołkach. Narysuj je i wskaż ich centroidy.

9. [M20] Z faktu, że co najwyżej jedno poddrzewo wierzchołka drzewa wolnego może zawierać centroid, wyprowadź wniosek, że w drzewie wolnym są co najwyżej dwa centroidy oraz że gdy są dwa, muszą ze sobą sąsiadować.

► **10.** [M22] Udowodnij, że drzewo wolne o n wierzchołkach i dwóch centroidach składa się z dwóch drzew wolnych o $n/2$ wierzchołkach połączonych krawędzią. W drugą stronę, jeśli dwa drzewa wolne o m wierzchołkach połączymy krawędzią, otrzymamy drzewo wolne o $2m$ wierzchołkach i dwóch centroidach.

► **11.** [M28] W tekście był wyprowadzony wzór (14) na liczbę różnych drzew binarnych o n węzłach. Uogólnij ten wynik na drzewa t -arne o n węzłach. (Zobacz ćwiczenie 2.3.1–35; drzewo t -arne jest albo puste, albo składa się z korzenia i t rozłącznych drzew t -arnych). Wskazówka: Skorzystaj ze wzoru (21) z punktu 1.2.9.

12. [M20] Wyznacz liczbę etykietowanych drzew zorientowanych o n wierzchołkach, korzystając z wyznaczników oraz wyniku ćwiczenia 2.3.4.2–19. (Zobacz też ćwiczenie 1.2.3–36).

13. [15] Jakie drzewo zorientowane o wierzchołkach $\{1, 2, \dots, 10\}$ ma reprezentację kanoniczną $3, 1, 4, 1, 5, 9, 2, 6, 5?$

14. [10] Prawda czy fałsz: ostatni element $f(V_{n-1})$ w reprezentacji kanonicznej drzewa zorientowanego jest zawsze korzeniem drzewa.

15. [21] Omów związek (jeśli istnieje) między algorytmem sortowania topologicznego z punktu 2.2.3 a reprezentacją kanoniczną drzew zorientowanych.

16. [25] Zaprojektuj (jak najbardziej wydajny) algorytm przekształcający drzewo zorientowane w reprezentacji kanonicznej na tradycyjną reprezentację korzystającą z dowiązań PARENT.

► **17.** [M26] Niech $f(x)$ będzie funkcją o wartościach całkowitych, taką że $1 \leq f(x) \leq m$ dla wszystkich całkowitych $1 \leq x \leq m$. Definiujemy $x \equiv y$ wtedy i tylko wtedy, gdy $f^{[r]}(x) = f^{[s]}(y)$ dla pewnych $r, s \geq 0$, gdzie $f^{[0]}(x) = x$ i $f^{[r+1]}(x) = f(f^{[r]}(x))$. Korzystając z takich metod zliczania, jak pokazane w tym rozdziale, udowodnij, że liczba funkcji, takich że $x \equiv y$ dla wszystkich x i y , wynosi $m^{m-1}Q(m)$, gdzie $Q(m)$ jest funkcją zdefiniowaną w podpunkcie 1.2.11.3.

18. [24] Pokaż, że poniższa metoda jest innym sposobem zdefiniowania wzajemnie jednoznacznej odpowiedniości między $(n - 1)$ -elementowymi krotkami liczb od 1 do n a drzewami zorientowanymi o n etykietowanych wierzchołkach. Niech V_1, \dots, V_k będzie ciągiem liści drzewa w porządku rosnącym. Niech $(V_1, V_{k+1}, V_{k+2}, \dots, V_q)$ będzie ścieżką od V_1 do korzenia. Wypiszmy wierzchołki $V_q, \dots, V_{k+2}, V_{k+1}$. Następnie niech $(V_2, V_{q+1}, V_{q+2}, \dots, V_r)$ będzie taką najkrótszą ścieżką zorientowaną z V_2 , że V_r już został wypisany. Wypiszmy $V_r, \dots, V_{q+2}, V_{q+1}$. Następnie niech $(V_3, V_{r+1}, \dots, V_s)$ będzie najkrótszą ścieżką zorientowaną z V_3 , taką że V_s już został wypisany. Wypiszmy V_s, \dots, V_{r+1} . I tak dalej. Na przykład drzewo (17) zakodujemy jako $3, 1, 3, 3, 5, 10, 5, 10, 1$. Pokaż, że ten proces jest odwracalny. W szczególności narysuj drzewo zorientowane o wierzchołkach $\{1, 2, \dots, 10\}$, którego reprezentacją jest $3, 1, 4, 1, 5, 9, 2, 6, 5$.

19. [M24] Ile jest różnych takich etykietowanych drzew zorientowanych, że k spośród ich n wierzchołków ma stopień wejściowy równy zero?

20. [M24] (J. Riordan) Ile jest różnych etykietowanych drzew zorientowanych o n wierzchołkach, mających k_0 wierzchołków o stopniu wejściowym 0, k_1 wierzchołków

o stopniu wejściowym 1, k_2 wierzchołków o stopniu wejściowym 2, ...? (Zauważ, że $k_0 + k_1 + k_2 + \dots = n$ i $k_1 + 2k_2 + 3k_3 + \dots = n - 1$).

- 21. [M21] Ile jest etykietowanych drzew zorientowanych, w których każdy wierzchołek ma stopień wejściowy zero lub dwa? (Zobacz ćwiczenie 20 i 2.3–20).
- 22. [M20] Ile jest etykietowanych drzew wolnych o n wierzchołkach? (Innymi słowy, jeśli mamy n wierzchołków, to istnieje $2^{\binom{n}{2}}$ grafów o tymu wierzchołkach, bo graf jest wyznaczony przez wybór krawędzi między wierzchołkami; ile wśród tych grafów jest drzew wolnych?)
- 23. [M21] Ile jest drzew uporządkowanych o n etykietowanych wierzchołkach? (Podaj prosty wzór zawierający silnie).
- 24. [M16] Wszystkie etykietowane drzewa zorientowane o wierzchołkach 1, 2, 3, 4 i korzeniu 1 to grafy (15). Ile jest drzew uporządkowanych o tych samych wierzchołkach i korzeniu?
- 25. [M20] Ile wynosi wartość $r(n, q)$ występująca we wzorach (18) i (19)? (Podaj prosty wzór; w tekście pojawia się jedynie przedstawienie $r(n, n - 1) = n^{n-2}$).
- 26. [20] Przyjmując oznaczenia wprowadzone na końcu bieżącego punktu, narysuj $((3, 2, 4), (1, 4, 2))$ -konstrukcję, analogiczną do (23), i znajdź liczbę k odpowiadającą następującej reprezentacji kanonicznej: (a) $t = 8$; (b) „czerwony, żółty, niebieski, czerwony, żółty, niebieski, czerwony, niebieski, niebieski”; (c) „czerwony, żółty, niebieski, żółty, żółty, niebieski, żółty”; (d) 3; 1, 2, 2, 1; 2, 4.
- 27. [M28] Niech $U_1, U_2, \dots, U_p, \dots, U_q; V_1, V_2, \dots, V_r$ będą wierzchołkami grafu skierowanego, gdzie $1 \leq p \leq q$. Niech f będzie dowolną funkcją ze zbioru $\{p+1, \dots, q\}$ w zbiór $\{1, 2, \dots, r\}$ i niech graf skierowany zawiera dokładnie $q - p$ krawędzi. Prowadzimy krawędź z U_k do $V_{f(k)}$ dla $p < k \leq q$. Pokaż, że liczba sposobów dołożenia r dodatkowych krawędzi, po jednej z każdego z wierzchołków V_1, V_2, \dots, V_r do któregoś z wierzchołków $U_1, U_2, \dots, U_p, \dots, U_q$, tak by otrzymany graf skierowany nie zawierał cykli zorientowanych, wynosi $q^{r-1}p$. Udowodnij to stwierdzenie, uogólniając metodę reprezentacji kanonicznej, tj. zdefiniuj wzajemnie jednoznaczne odwzorowanie między wszystkimi sposobami dodania r krawędzi a zbiorem wszystkich ciągów liczb całkowitych a_1, a_2, \dots, a_r , gdzie $1 \leq a_k \leq q$ dla $1 \leq k < r$ oraz $1 \leq a_r \leq p$.
- 28. [M22] (Drzewa dwupodziałowe) Posłuż się wynikiem ćwiczenia 27 w celu wyznaczenia liczby takich etykietowanych drzew wolnych o wierzchołkach $U_1, \dots, U_m, V_1, \dots, V_n$, że każda krawędź prowadzi od U_j do V_k dla pewnych j i k .
- 29. [HM26] Udowodnij, że jeśli $E_k(r, t) = r(r + kt)^{k-1}/k!$ i jeśli $zx^t = \ln x$, to

$$x^r = \sum_{k \geq 0} E_k(r, t) z^k$$

przy ustalonym t i odpowiednio małych $|z|$ i $|x - 1|$. [Skorzystaj z faktu $G_m(z) = G_1(z)^m$ omówionego w tekście po wzorze (19)]. W tym wzorze r oznacza dowolną liczbę rzeczywistą. [Uwaga: Wnioskiem z tego wzoru jest tożsamość

$$\sum_{k=0}^n E_k(r, t) E_{n-k}(s, t) = E_n(r + s, t);$$

wynika stąd twierdzenie dwumianowe Abela, równanie (16) z punktu 1.2.6; porównaj wzór (30) w tym samym punkcie].

30. [M23] Niech liczby n, x, y, z_1, \dots, z_n będą dodatnie i całkowite. Rozważmy zbiór $x + y + z_1 + \dots + z_n + n$ wierzchołków r_i, s_{jk}, t_j ($1 \leq i \leq x+y$, $1 \leq j \leq n$, $1 \leq k \leq z_j$), pomiędzy którymi narysowano krawędzie od s_{jk} do t_j dla wszystkich j i k . Zgodnie z ćwiczeniem 27 istnieje $(x+y)(x+y+z_1+\dots+z_n)^{n-1}$ sposobów narysowania po jednej krawędzi z każdego z wierzchołków t_1, \dots, t_n do pozostałych wierzchołków, tak by powstały graf nie zawierał cyklu zorientowanego. Korzystając z tego faktu, udowodnij twierdzenie dwumianowe Hurwitzta:

$$\sum x(x+\epsilon_1 z_1 + \dots + \epsilon_n z_n)^{\epsilon_1 + \dots + \epsilon_n - 1} y(y+(1-\epsilon_1)z_1 + \dots + (1-\epsilon_n)z_n)^{n-1-\epsilon_1-\dots-\epsilon_n} = (x+y)(x+y+z_1+\dots+z_n)^{n-1},$$

gdzie sumujemy po wszystkich 2^n wyborach $\epsilon_1, \dots, \epsilon_n$ dla każdego ϵ przybierającego wartość 0 lub 1.

31. [M24] Rozwiąż ćwiczenie 5 dla drzew uporządkowanych, tj. wyprowadź funkcję tworzącą liczby nieetykietowanych drzew uporządkowanych o n liściach bez węzłów stopnia 1.

32. [M37] (A. Erdélyi, I. M. H. Etherington, *Edinburgh Math. Notes* **32** (1940), 7–12) Ile jest (zorientowanych, nieetykietowanych) drzew o n_0 węzłach stopnia 0, n_1 stopnia 1, ..., n_m stopnia m i bez węzłów stopni wyższych niż m ? (Rozwiązanie można przedstawić za pomocą silni, otrzymując uogólnienie wyniku ćwiczenia 11).

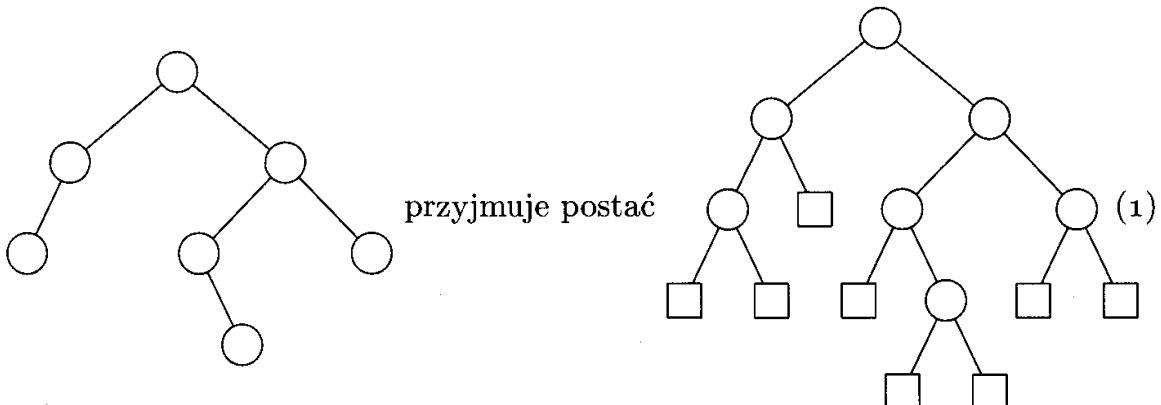
► **33.** [M28] W tekście był podany szereg potęgowy będący rozwiązaniem równania $w = y_1 e^{z_1 w} + \dots + y_r e^{z_r w}$ pojawiającego się we wzorach na zliczanie pewnych laściwów zorientowanych. Pokaż, że wzór na zliczanie z ćwiczenia 32 prowadzi do szeregu potęgowego będącego rozwiązaniem równania

$$w = z_1 w^{e_1} + z_2 w^{e_2} + \dots + z_r w^{e_r}.$$

Posłuż się przedstawieniem wzoru w w postaci szeregu potęgowego zmiennych z_1, \dots, z_r (e_1, \dots, e_r są ustalonimi nieujemnymi liczbami całkowitymi, z których przynajmniej jedna jest zerem).

2.3.4.5. Długość ścieżki. Pojęcie „długości ścieżki” w drzewie ma w analizie algorytmów ogromne znaczenie, ponieważ ta wielkość jest często w prosty sposób związana z czasem wykonania algorytmu. Będziemy zajmować się głównie drzewami binarnymi, jako że mają najwięcej wspólnego z rzeczywistymi reprezentacjami struktur danych w pamięci.

Zaczynamy od rozszerzenia drzew binarnych o specjalne węzły, wstawione w miejsce pustych poddrzew. I tak drzewo



Takie drzewa będziemy nazywać *rozszerzonymi drzewami binarnymi*. Dodanie „kwadratowych” węzłów nieco ułatwia mówienie o takim drzewie; w następnych rozdziałach niejednokrotnie będziemy korzystać z rozszerzonych drzew binarnych. Jasne jest, że każdy węzeł „okrągły” ma dwoje dzieci, a każdy węzeł „kwadratowy” – zero. (Porównaj z ćwiczeniem 2.3–20). Jeśli węzłów okrągłych jest n , a kwadratowych s , to w drzewie jest $n + s - 1$ krawędzi (co widać, gdy potraktujemy drzewo jako drzewo wolne); patrząc na liczbę dzieci stwierdzamy, że jest $2n$ krawędzi. Widać zatem, że

$$s = n + 1; \quad (2)$$

innymi słowy, liczba dodanych do drzewa węzłów zewnętrznych jest o jeden większa niż liczba węzłów wewnętrznych. (Inny dowód pojawił się w ćwiczeniu 2.3.1–14). Wzór (2) jest poprawny nawet dla $n = 0$.

Założymy, że mamy drzewo binarne rozszerzone w opisany sposób. *Długość ścieżki zewnętrznej* E definiujemy jako sumę długości ścieżek od korzenia do każdego węzła braną po wszystkich węzłach zewnętrznych (kwadratowych). *Długość ścieżki wewnętrznej* I to ta sama wielkość sumowana po wszystkich węzłach wewnętrznych (okrągłych). W drzewie (1) długość ścieżki zewnętrznej wynosi

$$E = 3 + 3 + 2 + 3 + 4 + 4 + 3 + 3 = 25,$$

a długość ścieżki wewnętrznej wynosi

$$I = 2 + 1 + 0 + 2 + 3 + 1 + 2 = 11.$$

Te dwie wartości są zawsze powiązane zależnością

$$E = I + 2n, \quad (3)$$

gdzie n jest liczbą węzłów wewnętrznych.

By udowodnić wzór (3), zastanówmy się, co by było, gdybyśmy usunęli węzeł wewnętrzny V leżący w odległości k od korzenia, mający dzieci będące węzłami zewnętrznymi. Wielkość E maleje o $2(k + 1)$, ponieważ usuwamy dzieci V . Następnie rośnie o k , bo V staje się węzłem zewnętrzny. Sumaryczna zmiana E wynosi $-k - 2$. Sumaryczna zmiana I wynosi $-k$. Na mocy indukcji otrzymujemy (3).

Nietrudno dostrzec, że długość ścieżki wewnętrznej (a tym samym także zewnętrznej) jest największa w przypadku zdegenerowanych drzew binarnych o kształcie liniowym. W takim przypadku długość ścieżki wewnętrznej w drzewie równa się

$$(n - 1) + (n - 2) + \dots + 1 + 0 = \frac{n^2 - n}{2}.$$

Można pokazać, że „średnia” długość ścieżki dla wszystkich drzew binarnych jest w istocie proporcjonalna do $n\sqrt{n}$ (zobacz ćwiczenie 5).

Rozważmy problem konstruowania drzew binarnych o n węzłach, dla których długość ścieżki jest *minimalna*. Takie drzewa są ważne, ponieważ umożliwiają zminimalizowanie czasu wykonania wielu algorytmów. Jest jasne, że tylko jeden węzeł (korzeń) może leżeć w odległości零 od korzenia. Co najwyżej dwa węzły mogą leżeć w odległości jeden, co najwyżej cztery w odległości dwa itd. Stąd

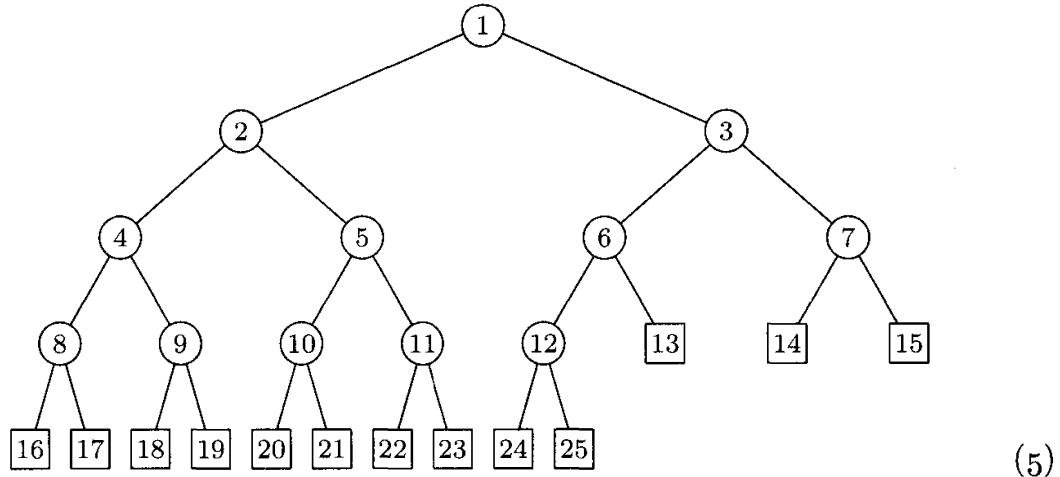
długość ścieżki wewnętrznej jest nie mniejsza niż suma pierwszych n wyrazów ciągu

$$0, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, \dots$$

To jest suma $\sum_{k=1}^n \lfloor \lg k \rfloor$, a z ćwiczenia 1.2.4–42 wiemy, że jest ona równa

$$(n+1)q - 2^{q+1} + 2, \quad q = \lfloor \lg(n+1) \rfloor. \quad (4)$$

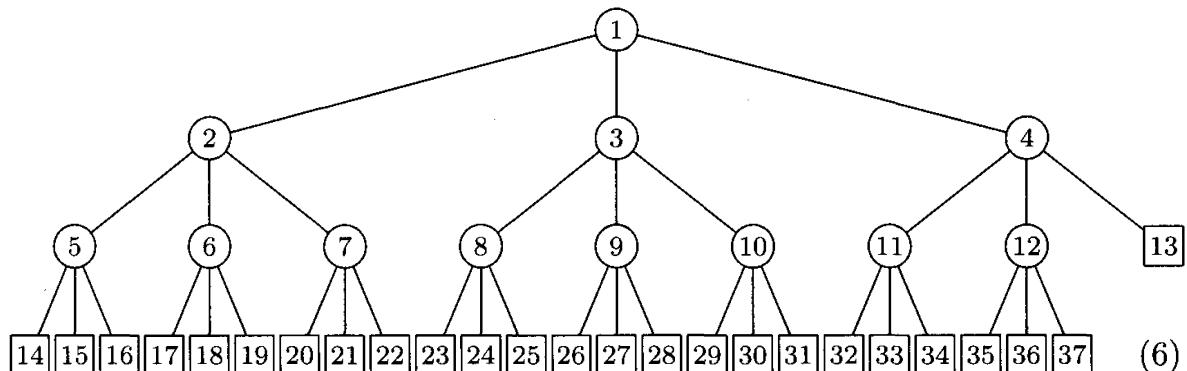
Optymalna wartość (4) wynosi $n \lg n + O(n)$, ponieważ $q = \lg n + O(1)$. Taką wartość osiągamy w drzewach wyglądających jak poniżej (przykład dla $n = 12$):



Drzewo takie jak (5) jest nazywane *zupełnym drzewem binarnym* o n węzłach wewnętrznych. W ogólności możemy ponumerować węzły wewnętrzne liczbami $1, 2, \dots, n$. Ten sposób numerowania ma tę przydatną własność, że rodzic węzła k ma numer $\lfloor k/2 \rfloor$, a dzieci węzła k są węzły $2k$ i $2k + 1$. Węzły zewnętrzne mają numery od $n + 1$ do $2n + 1$ włącznie.

Wynika stąd, że reprezentację zupełnego drzewa binarnego łatwo przechowywać w kolejnych komórkach pamięci w ten sposób, że jego struktura jest niejawnie wyznaczana przez położenie węzłów (nie potrzeba wskaźników). Zupełne drzewa binarne jawnie lub niejawnie pojawiają się w wielu istotnych algorytmach, zatem Czytelnik powinien poświęcić im szczególną uwagę.

Powyższe pojęcia można uogólnić na drzewa ternarne, kwaternarne oraz drzewa wyższych rzędów. Definiujemy *drzewo t -arne* jako zbiór węzłów, który albo jest pusty, albo składa się z korzenia oraz t uporządkowanych, rozłącznych drzew t -arnych. (To jest uogólnienie definicji drzewa binarnego z podrozdziału 2.3). *Zupełne drzewo ternarne* o 12 węzłach wygląda tak:



Łatwo się przekonać, że taką samą konstrukcję można zastosować dla dowolnego $t \geq 2$. W zupełnym drzewie t -arnym o węzłach wewnętrznych $\{1, 2, \dots, n\}$ rodzic węzła k ma numer

$$\lfloor (k+t-2)/t \rfloor = \lceil (k-1)/t \rceil,$$

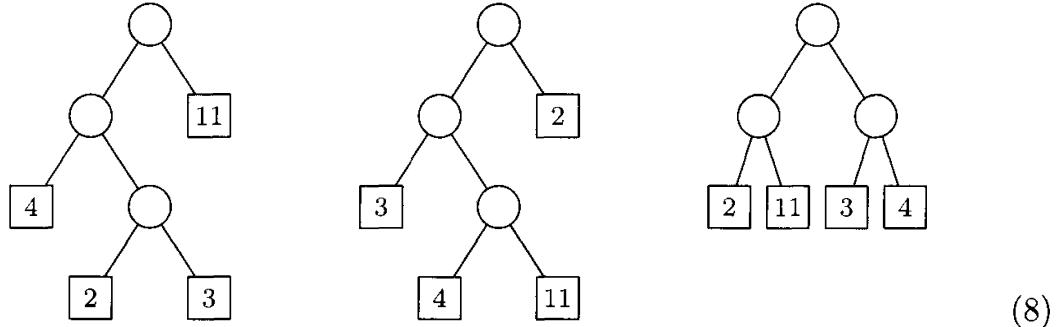
a dziećmi węzła k są węzły

$$t(k-1)+2, \quad t(k-1)+3, \quad \dots, \quad tk+1.$$

Takie drzewo ma minimalną długość ścieżki wewnętrznej pośród wszystkich drzew t -arnych o n węzłach wewnętrznych. W ćwiczeniu 8 dowodzimy, że długość ścieżki wewnętrznej w takim drzewie jest równa

$$\left(n + \frac{1}{t-1} \right) q - \frac{(t^{q+1} - t)}{(t-1)^2}, \quad q = \lfloor \log_t((t-1)n+1) \rfloor. \quad (7)$$

Powyższe wyniki można jeszcze uogólnić w inny ważny sposób, jeśli zmienimy punkt widzenia. Przypuśćmy, że mamy m liczb rzeczywistych w_1, w_2, \dots, w_m . Problem polega na tym, by znaleźć rozszerzone drzewo binarne o m węzłach zewnętrznych i przypisać do nich liczby w_1, \dots, w_m w taki sposób, by zminimalizować sumę $\sum w_j l_j$, gdzie l_j jest długością ścieżki od korzenia, a sumę bierzemy względem wszystkich węzłów zewnętrznych. Na przykład, jeśli dane są liczby 2, 3, 4, 11, to możemy utworzyć rozszerzone drzewa binarne jak poniżej



„Ważone” długości ścieżek $\sum w_j l_j$ wynoszą odpowiednio 34, 53 i 40. (W idealnie zrównoważonym drzewie *nie* otrzymamy minimalnej ważonej długości ścieżki dla wag 2, 3, 4 i 11, chociaż jak stwierdziliśmy, otrzymamy ją w szczególnym przypadku $w_1 = w_2 = \dots = w_m = 1$).

W różnych algorytmach pojawiają się różne interpretacje ważonej długości ścieżki. Możemy ją zastosować na przykład do scalania posortowanych ciągów o długościach w_1, w_2, \dots, w_m (zobacz rozdział 5). Jednym z bardziej bezpośrednich zastosowań tego pojęcia jest spojrzenie na drzewo binarne jak na procedurę poszukiwania. Zaczynamy od korzenia i wykonujemy porównanie. Wynik porównania powoduje wybór jednego z dwóch odgałęzień, gdzie wykonujemy kolejne porównania itd. Jeśli chcemy na przykład zdecydować, który z czterech różnych wariantów jest prawdziwy i jeżeli te warianty są prawdziwe z prawdopodobieństwami odpowiednio $\frac{2}{20}, \frac{3}{20}, \frac{4}{20}$ i $\frac{11}{20}$, to drzewo minimalizujące ważoną długość ścieżki stanowi *optimalną procedurę poszukiwania*. [To są wagi w (8) przemnożone przez współczynnik].

Elegancki algorytm znajdowania drzewa o minimalnej ważonej długości ścieżki został odkryty przez D. Huffmana [Proc. IRE 40 (1952), 1098–1101].

Zaczynamy od znalezienia dwóch w o najmniejszej wartości, powiedzmy w_1 i w_2 . Następnie rozwiążujemy problem dla $m - 1$ wag $w_1 + w_2, w_3, \dots, w_m$ i w znalezionej rozwiązańu zastępujemy węzeł



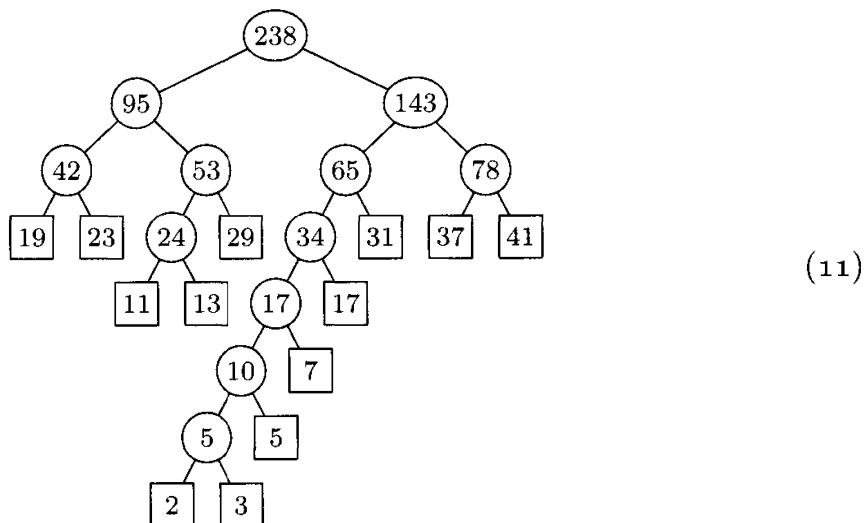
przez



W ramach przykładu wyznaczymy optymalne drzewo dla wag 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41. Zaczynamy od połączenia 2 + 3. Teraz szukamy rozwiązania dla 5, 5, 7, ..., 41. Łączymy 5 + 5 itd. Całe obliczenie można zapisać następująco:

$$\begin{array}{cccccccccccc}
 2 & 3 & 5 & 7 & 11 & 13 & 17 & 19 & 23 & 29 & 31 & 37 & 41 \\
 & \underline{5} & \underline{5} & 7 & 11 & 13 & 17 & 19 & 23 & 29 & 31 & 37 & 41 \\
 & & \underline{10} & \underline{7} & 11 & 13 & 17 & 19 & 23 & 29 & 31 & 37 & 41 \\
 & & & 17 & \underline{11} & \underline{13} & 17 & 19 & 23 & 29 & 31 & 37 & 41 \\
 & & & & \underline{17} & \underline{24} & \underline{17} & 19 & 23 & 29 & 31 & 37 & 41 \\
 & & & & & \underline{24} & \underline{34} & \underline{19} & \underline{23} & 29 & 31 & 37 & 41 \\
 & & & & & & \underline{24} & 34 & & & & & & \\
 & & & & & & & \underline{34} & & & & & & \\
 & & & & & & & & 42 & \underline{29} & 31 & 37 & 41 \\
 & & & & & & & & & \underline{42} & \underline{53} & \underline{31} & 37 & 41 \\
 & & & & & & & & & & \underline{42} & \underline{53} & 65 & \underline{37} & \underline{41} \\
 & & & & & & & & & & & \underline{42} & \underline{53} & 65 & & 78 \\
 & & & & & & & & & & & & 95 & \underline{65} & & 78 \\
 & & & & & & & & & & & & & \underline{95} & & 143 \\
 & & & & & & & & & & & & & & & 238
 \end{array}$$

Następujące drzewo odpowiada konstrukcji Huffmana:



(Liczby wewnętrznych okrągłych węzłów obrazują związek między tym drzewem a powyższym obliczeniem; zobacz także ćwiczenie 9).

Nie trudno udowodnić przez indukcję względem m , że ta metoda rzeczywiście minimalizuje ważoną długość ścieżki. Przypuśćmy, że mamy $w_1 \leq w_2 \leq w_3 \leq \dots \leq w_m$, gdzie $m \geq 2$, i przypuśćmy, że mamy drzewo minimalizujące ważoną długość ścieżki. (Takie drzewo na pewno istnieje, ponieważ jest skończenie wiele drzew binarnych o m liściach). Niech V będzie węzłem wewnętrznym o maksymalnej odległości od korzenia. Jeśli w_1 i w_2 nie są wagami dzieci węzła V , to zamieniamy wagi dzieci na wartości, które tam już są. Taka operacja nie spowoduje wzrostu ważonej długości ścieżki. Istnieje zatem drzewo minimalizujące ważoną długość ścieżki i zawierające poddrzewo (10). Łatwo teraz udowodnić, że ważona długość ścieżki takiego drzewa jest minimalna wtedy i tylko wtedy, gdy drzewo, w którym (10) zamienimy na (9) ma minimalną ważoną długość ścieżki dla wag $w_1 + w_2, w_3, \dots, w_m$. (Zobacz ćwiczenie 9).

Jeśli wagi są nieujemne, to za każdym razem, gdy w powyższej konstrukcji łączymy dwie wagi, są one nie mniejsze niż poprzednio łączone wagi. Oznacza to, że istnieje bardzo elegancki sposób znalezienia drzewa Huffmana, jeśli wagi są posortowane niemalejąco. Utrzymujemy dwie kolejki: w jednej przechowujemy pierwotne wagi, a w drugiej wagi połączone. W każdym kroku najmniejsza niewykorzystana waga znajduje się na początku jednej z kolejek, więc nie musimy jej szukać. Zobacz ćwiczenie 13, pokazujące, że można wykorzystać ten pomysł nawet przy ujemnych wagach.

W ogólności istnieje wiele drzew minimalizujących $\sum w_j l_j$. Jeśli naszkicowany algorytm w przypadku „remisów” przedkłada pierwotne wagi nad wagami otrzymanymi z połączeń, to otrzymane drzewo ma najmniejszą z wartości $\max l_j$ oraz $\sum l_j$ spośród wszystkich drzew minimalizujących $\sum w_j l_j$. Dla dodatnich wag to drzewo minimalizuje $\sum w_j f(l_j)$ dla dowolnej funkcji wypukłej f po wszystkich takich drzewach. [Zobacz E. S. Schwartz, *Information and Control* 7 (1964), 37–44; G. Markowsky, *Acta Informatica* 16 (1981), 363–370].

Metodę Huffmana można uogólnić na drzewa t -arne. (Zobacz ćwiczenie 10). Inne ważne uogólnienie tej metody omawiamy w punkcie 6.2.2. Długości ścieżek omawiamy ponadto w punktach 5.3.1, 5.4.9 i podrozdziale 6.3.

ĆWICZENIA

1. [12] Czy istnieją inne drzewa binarne o 12 węzłach внутренних i minimalnej długości ścieżki poza zupełnym drzewem binarnym (5)?
 2. [17] Narysuj rozszerzone drzewo binarne minimalizujące ważoną długość ścieżki, którego liście mają wagi 1, 4, 9, 16, 25, 36, 49, 64, 81, 100.
- ▶ 3. [M24] Rozszerzone drzewo binarne o m węzłach zewnętrznych wyznacza zbiór długości ścieżek l_1, l_2, \dots, l_m opisujący długości ścieżek od korzenia do węzłów zewnętrznych. W drugą stronę, jeśli mamy zbiór liczb l_1, l_2, \dots, l_m , to czy zawsze da się skonstruować rozszerzone drzewo binarne, dla którego te liczby są długościami ścieżek (w pewnym porządku)? Pokaż, że jest to możliwe wtedy i tylko wtedy, gdy $\sum_{j=1}^m 2^{-l_j} = 1$.
 - ▶ 4. [M25] (E. S. Schwartz i B. Kallick) Założmy, że $w_1 \leq w_2 \leq \dots \leq w_m$. Pokaż, że istnieje rozszerzone drzewo binarne minimalizujące $\sum w_j l_j$ i takie, że jego liście w porządku od lewej do prawej zawierają odpowiednio wartości w_1, w_2, \dots, w_m .

[Na przykład drzewo (11) *nie* spełnia tego warunku, ponieważ wagi występują w porządku 19, 23, 11, 13, 29, 2, 3, 5, 7, 17, 31, 37, 41. Szukamy drzewa, w którym wagi występują w porządku rosnącym, a tak nie musi być w konstrukcji Huffmana].

5. [HM26] Niech

$$B(w, z) = \sum_{n,p \geq 0} b_{np} w^p z^n,$$

gdzie b_{np} jest liczbą drzew binarnych o n węzłach i długości ścieżki wewnętrznej równej p . [Stąd

$$B(w, z) = 1 + z + 2wz^2 + (w^2 + 4w^3)z^3 + (4w^4 + 2w^5 + 8w^6)z^4 + \dots;$$

$B(1, z)$ jest funkcją $B(z)$ z (13) w podpunkcie 2.3.4.4].

- a) Znajdź zależność funkcyjną charakteryzującą $B(w, z)$, będącą uogólnieniem wzoru 2.3.4.4-(12).
- b) Posłuż się wynikiem (a) do wyznaczenia średniej długości ścieżki wewnętrznej drzewa binarnego o n węzłach, przy założeniu że każde z $\frac{1}{n+1} \binom{2n}{n}$ drzew jest jednakowo prawdopodobne.
- c) Znajdź wartość asymptotyczną tej wielkości.

6. [16] Jeśli t -arne drzewo rozszerzymy o węzły kwadratowe, tak jak (1), to jaki jest związek (analogiczny do (2)) między liczbą węzłów kwadratowych i okrągłych?

7. [M21] Jaki jest związek między długością ścieżki zewnętrznej i wewnętrznej w drzewie t -arnym? (Zobacz ćwiczenie 6. Szukamy uogólnienia (3)).

8. [M23] Udowodnij (7).

9. [M21] Liczby w okrągłych węzłach w drzewie (11) są równe sumom wag węzłów zewnętrznych odpowiednich poddrzew. Pokaż, że suma wszystkich wartości w okrągłych węzłach jest równa ważonej długości ścieżki.

► 10. [M26] (D. Huffman) Pokaż, w jaki sposób skonstruować drzewo t -arne o minimalnej ważonej długości ścieżki dla danych wag w_1, w_2, \dots, w_m . Skonstruj optymalne drzewo ternarne dla wag 1, 4, 9, 16, 25, 36, 49, 64, 81, 100.

11. [16] Czy jest jakiś związek między zupełnym drzewem binarnym (5) a „notacją dziesiętną Dewey'a” dla drzew binarnych opisaną w ćwiczeniu 2.3.1–5?

► 12. [M20] Przypuśćmy, że wybraliśmy losowo węzeł w drzewie binarnym (wybór każdego jest równie prawdopodobny). Pokaż, że średni rozmiar poddrzewa o korzeniu w wybranym węźle jest związany z długością ścieżki w drzewie.

13. [22] Zaprojektuj algorytm, który dla danych m wag $w_1 \leq w_2 \leq \dots \leq w_m$ konstruuje rozszerzone drzewo binarne o minimalnej ważonej długości ścieżki. Wyjściowe drzewo ma być reprezentowane w trzech tablicach

$$A[1] \dots A[2m-1], \quad L[1] \dots L[m-1], \quad R[1] \dots R[m-1];$$

$L[i]$ i $R[i]$ wskazują na lewe i prawe dziecko węzła wewnętrznego numer i , korzeń ma numer 1, $A[i]$ jest wagą węzła i . Pierwotne wagi powinny figurować jako wagi węzłów zewnętrznych $A[m], \dots, A[2m-1]$. Twój algorytm powinien wykonywać mniej niż $2m$ porównań wag. *Uwaga:* Niektóre lub wszystkie podane wagi mogą być ujemne!

14. [25] (T. C. Hu, A. C. Tucker) Po k krokach algorytmu Huffmana węzły połączone do tej pory składają się na las zawierający $m - k$ rozszerzonych drzew binarnych. Udowodnij, że ten las ma najmniejszą całkowitą ważoną długość ścieżki pośród wszystkich lasów zawierających $m - k$ rozszerzonych drzew binarnych o danych wagach.

15. [M25] Pokaż, że algorytm podobny do algorytmu Huffmana wyznacza rozszerzone drzewo binarne minimalizujące (a) $\max(w_1 + l_1, \dots, w_m + l_m)$; (b) $w_1 x^{l_1} + \dots + w_m x^{l_m}$, dla danego $x > 1$.

16. [M25] (F. K. Hwang) Niech $w_1 \leq \dots \leq w_m$ i $w'_1 \leq \dots \leq w'_m$ będą dwoma zbiorami wag, takimi że

$$\sum_{j=1}^k w_j \leq \sum_{j=1}^k w'_j \quad \text{dla } 1 \leq k \leq m.$$

Udowodnij, że minimalna ważona długość ścieżki spełnia nierówność $\sum_{j=1}^m w_j l_j \leq \sum_{j=1}^m w'_j l'_j$.

17. [HM30] (C. R. Glassey, R. M. Karp) Niech s_1, \dots, s_{m-1} będą liczbami w węzłach wewnętrznych (okrągłych) rozszerzonego drzewa binarnego utworzonego przez algorytm Huffmana, w kolejności utworzenia. Niech s'_1, \dots, s'_{m-1} będą wagami węzłów wewnętrznych dowolnego rozszerzonego drzewa binarnego na tym samym zbiorze wag $\{w_1, \dots, w_m\}$, wypisanymi w dowolnym porządku, takim że każdy węzeł nie będący korzeniem pojawia się przed swoim rodzicem. (a) Udowodnij, że $\sum_{j=1}^k s_j \leq \sum_{j=1}^k s'_j$ dla $1 \leq k < m$. (b) Wynik (a) jest równoważny

$$\sum_{j=1}^{m-1} f(s_j) \leq \sum_{j=1}^{m-1} f(s'_j)$$

dla dowolnej funkcji f niemalejącej i wklęszej, tj. dowolnej funkcji f , dla której $f'(x) \geq 0$ i $f''(x) \leq 0$. [Zobacz Hardy, Littlewood, Pólya, *Messenger of Math.* **58** (1939), 145–152]. Skorzystaj z tego faktu, by pokazać, że w równaniu rekurencyjnym

$$F(n) = f(n) + \min_{1 \leq k < n} (F(k) + F(n - k)), \quad F(1) = 0$$

minimalna wartość jest osiągana zawsze przy $k = 2^{\lceil \lg(n/3) \rceil}$ dla dowolnej funkcji $f(n)$ spełniającej warunki $\Delta f(n) = f(n+1) - f(n) \geq 0$ i $\Delta^2 f(n) = \Delta f(n+1) - \Delta f(n) \leq 0$.

***2.3.4.6. Historia i bibliografia.** Drzewa istnieją od trzeciego Dnia Stworzenia. Przez wieki posługiwano się różnymi strukturami drzewiastymi (szczególnie drzewami genealogicznymi). Wydaje się, że pomysł formalnego zdefiniowania drzewa jako obiektu matematycznego pojawił się po raz pierwszy w pracach G. Kirchhoffa [*Annalen der Physik und Chemie* **72** (1847), 497–508, tłumaczenie angielskie w *IRE Transactions CT-5* (1958), 4–7]. Kirchhoff posługiwał się drzewami wolnymi do wyznaczenia zbioru cykli podstawowych sieci połączeń elektrycznych, korzystając z praw, które noszą jego imię, tak jak my robiliśmy to w podpunkcie 2.3.4.1. Taki pomysł pojawił się mniej więcej w tym samym czasie w książce *Geometrie der Lage* autorstwa K. G. Chr. von Staudta (strony 20–21). Nazwa „drzewo” oraz wiele wyników związanych głównie ze zliczaniem drzew zaczęła pojawiać się dziesięć lat później w serii artykułów autorstwa Arthura Cayley'a [zobacz *Collected Mathematical Papers of A. Cayley* **3** (1857), 242–246; **4** (1859), 112–115; **9** (1874), 202–204; **9** (1875), 427–460; **10** (1877), 598–600; **11** (1881), 365–367; **13** (1889), 26–28]. Cayley nie znał wcześniejszych prac Kirchhoffa i von Staudta. Jego poszukiwania zaczęły się od badań nad strukturą wzorów algebraicznych i były zainspirowane głównie przez zastosowania do problemów związanych z izomerami w chemii. Struktury drzewiaste były również badane niezależnie przez C. W. Borchardta [*Crelle* **57** (1860), 111–121],

J. B. Listinga [*Göttinger Abhandlungen, Math. Classe*, **10** (1862), 137–139] i C. Jordana [*Crelle* **70** (1869), 185–190].

Lemat o drzewach nieskończonych został sformułowany przez Dénesa Kóniga [*Fundamenta Math.* **8** (1926), 114–134]. To twierdzenie zajmuje honorowe miejsce w klasycznej książce tego autora *Theorie der endlichen und unendlichen Graphen* (Leipzig: 1936), rozdział 6. Podobny wynik zwany twierdzeniem o wachlarzu pojawił się nieco wcześniej w pracy L. E. J. Brouwera [*Verhandelingen Akad. Amsterdam* **12** (1919), 7], ale zawierał o wiele silniejszą tezę. Omówienie pracy Brouwera można znaleźć w: A. Heyting, *Intuitionism* (1956), podrozdział 3.4.

Wzór (3) na zliczanie nieetykietowanych drzew zorientowanych z podpunktu 2.3.4.4 został podany przez Cayley'a w jego pierwszym artykule o drzewach. W drugim artykule Cayley zlicza nieetykietowane drzewa uporządkowane. Równoważny problem geometryczny (zobacz ćwiczenie 1) został postawiony oraz rozwiązany przez J. von Segnera i L. Eulera 100 lat wcześniej [*Novi Commentarii Academiæ Scientiarum Petropolitanæ* **7** (1758–1759), 13–15, 203–210] i był przedmiotem siedmiu artykułów G. Lamé, E. Catalana, O. Rodriguesa i J. Bineta w *Journal de mathématiques* **3**, **4** (1838, 1839); dodatkową bibliografię zamieściliśmy w odpowiedzi do ćwiczenia 2.2.1–4. Związane z tym problemem liczby są dziś powszechnie nazywane „liczbami Catalana”. Mongolsko-chiński matematyk An-T'u Ming, badając szeregi nieskończone, natknął się na liczby Catalana przed 1750 rokiem, ale nie zauważył ich związku z drzewami ani innymi obiektymi kombinatorycznymi [zobacz J. Luo, *Acta Scientiarum Naturalium Universitatis Intramongolicæ* **19** (1988), 239–245; *Combinatorics and Graph Theory* (World Scientific Publishing, 1993), 68–70]. Liczby Catalana pojawiają się w różnych kontekstach. Richard Stanley pokazuje ich ponad 60 w swojej wspaniałej książce *Enumerative Combinatorics* **2** (Cambridge Univ. Press, 1999), rozdział 6. Zapewne najbardziej niezwykłe miejsce, gdzie pojawiają się liczby Catalana to pewne permutacje liczb, które H. S. M. Coxeter nazwał wzorami fryzowymi z uwagi na ich symetrię; zobacz ćwiczenie 4.

Wzór n^{n-2} na liczbę etykietowanych drzew wolnych został odkryty przez J. J. Sylvestera [*Quart. J. Pure and Applied Math.* **1** (1857), 55–56] jako produkt uboczny obliczeń pewnego wyznacznika (ćwiczenie 2.3.4.2–28). Cayley podał niezależne wyprowadzenie tego wzoru w 1889 roku [zobacz bibliografia powyżej]. W jego bardzo niejasno sformułowanym omówieniu można znaleźć pewien związek między etykietowanymi drzewami zorientowanymi i $(n - 1)$ -elementowymi krotkami liczb. Ten związek wprost po raz pierwszy opisał Heinz Prüfer [*Arch. Math. und Phys.* **27** (1918), 142–144] raczej niezależnie od wcześniejszych prac Cayleya. Istnieje bogata literatura na ten temat; klasyczne wyniki są bardzo ładnie przedstawione w książce J. W. Moon'a *Counting Labelled Trees* (Montreal: Canadian Math. Congress, 1970).

Bardzo ważny artykuł na temat zliczania drzew i wielu innych rodzajów struktur kombinatorycznych opublikował G. Pólya w *Acta Math.* **68** (1937), 145–253. Omówienie problemów zliczania grafów i wyśmienita bibliografia wcześniej literatury na ten temat znajduje się w przeglądzie Franka Harary'ego w *Graph Theory and Theoretical Physics* (London: Academic Press, 1967), 1–41.

Zasada minimalizacji ważonej długości ścieżki przez łączenie najmniejszych wag została odkryta przez D. Huffmana [Proc. IRE 40 (1952), 1098–1101] w związku z problemem wyznaczania kodu minimalizującego długość wiadomości. Ten sam pomysł został niezależnie opublikowany przez Setha Zimmermana [AMM 66 (1959), 690–693].

Kilka innych istotnych prac dotyczących teorii drzew cytowaliśmy w podpunktach 2.3.4.1–2.3.4.5 w związku z konkretnymi zagadnieniami.

ĆWICZENIA

- ▶ 1. [21] Znajdź prostą wzajemnie jednoznaczną odpowiedniość między drzewami binarnymi o n węzłach a podziałami wypukłego $(n+2)$ -kąta na n trójkątów, zakładając, że boki wielokąta są rozróżnialne.
- ▶ 2. [M26] T. P. Kirkman w 1857 roku postawił hipotezę, że liczba sposobów narysowania k nieprzecinających się poza wierzchołkami przekątnych w r -kącie równa się $\binom{r+k}{k} \binom{r-3}{k} / (r+k)$.
 - a) Rozszerz związek z ćwiczenia 1, tak by otrzymać równoważny problem związany ze zliczaniem drzew.
 - b) Udowodnij hipotezę Kirkmana, korzystając z metod przedstawionych w ćwiczeniu 2.3.4.4–32.
- ▶ 3. [M30] Rozważmy wszystkie takie sposoby podziału zbioru wierzchołków n -kąta wypukłego na k niepustych części, że żadna przekątna łącząca dwa wierzchołki należące do jednej części nie przecina przekątnej pomiędzy wierzchołkami należącymi do innej części.
 - a) Znajdź wzajemnie jednoznaczną odpowiedniość między takimi podziałami a cięwką klasą drzew.
 - b) Ile jest sposobów dokonania takiego podziału dla danych n i k ?
- ▶ 4. [M38] (Conway, Coxeter) Wzór fryzowy to nieskończona tablica taka jak

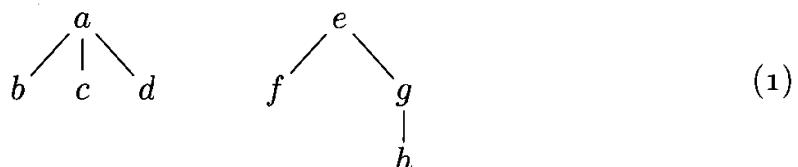
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
3	1	3	1	4	1	2	3	1	3	1	4	1	2	3	1	3	1	4	1	...
5	2	2	2	3	3	1	5	2	2	2	3	3	1	5	2	2	2	3	1	...
3	3	1	5	2	2	2	3	3	1	5	2	2	2	3	3	1	5	2	1	...
1	4	1	2	3	1	3	1	4	1	2	3	1	3	1	4	1	2	3	1	...
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...

w której górnego i dolnego wiersza składa się z samych jedynek, a każdy romb sąsiednich wartości $\begin{matrix} a & b \\ c & d \end{matrix}$ spełnia zależność $ad - bc = 1$. Znajdź wzajemnie jednoznaczną odpowiedniość między n -węzłowymi drzewami binarnymi i $(n+1)$ -wierszowymi wzorami fryzowymi liczb całkowitych.

2.3.5. Listy i odśmiecanie

W podrozdziale 2.3 nieformalnie zdefiniowaliśmy Listę jako „skończony ciąg zera lub więcej atomów lub List”.

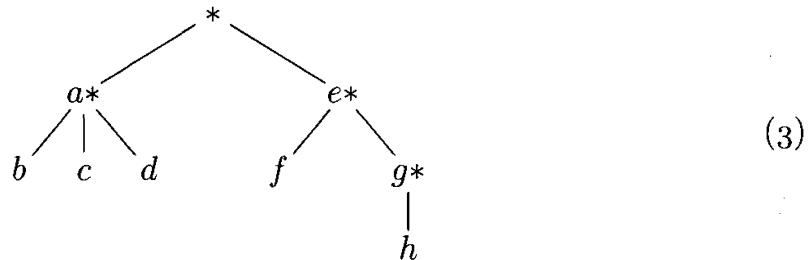
Każdy las jest Listą, na przykład



można uznać za listę

$$(a: (b, c, d), e: (f, g: (h))), \quad (2)$$

której schemat wygląda następująco:

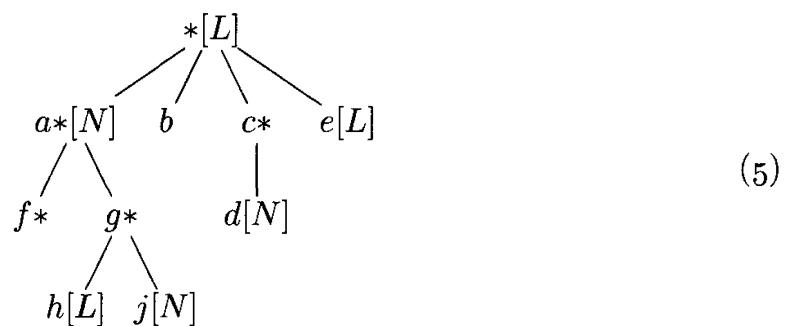


Czytelnik powinien w tym miejscu przypomnieć sobie zamieszczony wcześniej wstęp do zagadnień związanych z Listami, w szczególności (3), (4), (5), (6), (7) na pierwszych stronach podrozdziału 2.3. Przypomnijmy, że w (2) napis „ $a: (b, c, d)$ ” oznacza, że (b, c, d) jest listą złożoną z trzech atomów z etykieta „ a ”. Ta konwencja jest zgodna z naszym ogólnym założeniem, że każdy węzeł drzewa może zawierać dodatkowe informacje poza dowiązaniami wyznaczającymi jego strukturę. Jak jednak powiedzieliśmy w punkcie 2.3.3, często wygodnie zakładać, że wszystkie Listy są nieetykietowane, a co za tym idzie, że informacje przechowujemy wyłącznie w atomach.

Choć każdy las można uważać za Listę, to nie każda Lista jest lasem. Poniższa Lista jest zapewne bardziej typowa niż (2) i (3), ponieważ pokazuje, w jaki sposób Lista może naruszać strukturę drzewiastą:

$$L = (a: N, b, c: (d: N), e: L), \quad N = (f: (), g: (h: L, j: N)) \quad (4)$$

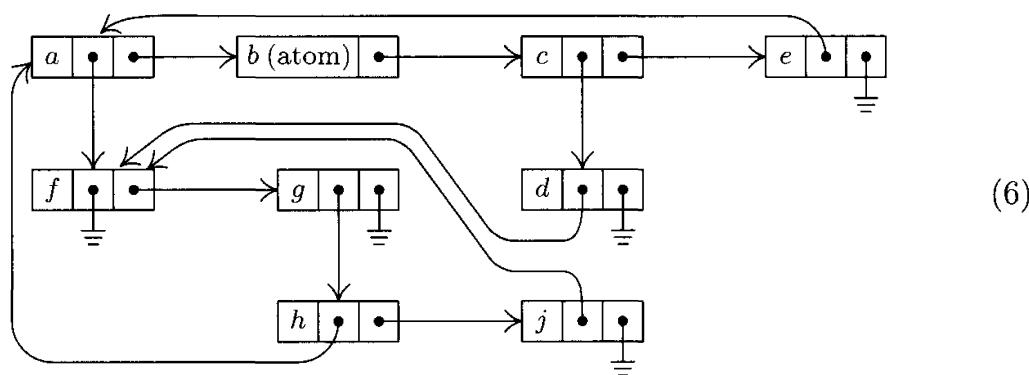
co można narysować tak:



[Porównaj z przykładem w 2.3-(7). Postaci powyższych schematów nie należy traktować zbyt poważnie].

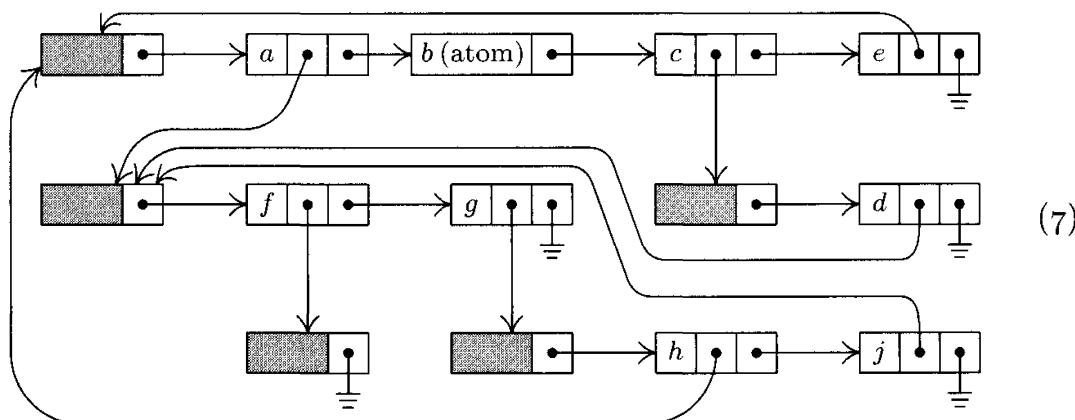
Jak można się spodziewać, jest wiele sposobów reprezentowania List w pamięci komputera. Te metody są zazwyczaj wariacjami na temat tych samych podstawowych technik, z których korzystaliśmy, reprezentując lasy za pomocą drzew binarnych: jedno pole, powiedzmy **RLINK**, zawiera wskaźnik do następnego elementu Listy, a inne pole **DLINK** wykorzystujemy jako wskaźnik do pierwszego elementu pod-Listy. Rozszerzając w naturalny sposób reprezentację opisaną

w punkcie 2.3.2, możemy reprezentować Listę (5) następująco:



Niestety, ten prosty pomysł *nie* jest bardzo praktyczny. Przypuśćmy na przykład, że mamy Listę $L = (A, a, (A, A))$ zawierającą trzy dowiązania do innej listy $A = (b, c, d)$. Jedna z typowych operacji na Listach polega na usunięciu skrajnie lewego elementu A , co powoduje, że lista A przybiera postać (c, d) ; ale to wymaga *trzech* zmian w reprezentacji L , jeśli będziemy trzymać się metody (6), ponieważ wskaźnik do A wskazuje na element b , który właśnie usuwamy. Po chwili zastanowienia można dojść do wniosku, że dużą niewygodą jest zmienianie każdego dowiązania do A tylko dlatego, że usuwamy z A pierwszy element. (W tym przykładzie moglibyśmy posłużyć się sztuczką; jeśli nie ma wskaźników do elementu c , to można skopiować element c w miejsce zajmowane przez b i usunąć stary element c . Ale ta sztuczka się nie sprawdza, gdy usuwamy z A ostatni element i staje się ona listą pustą).

Z powyższego powodu zamiast schematem (6) posługujemy się zazwyczaj podobnym schematem, jednakże na początku każdej Listy trzymamy *atrapę*, jak opisaliśmy w punkcie 2.2.4. Konfiguracja (6) jest zatem reprezentowana jako



Wprowadzenie atrap nie jest w istocie marnowaniem pamięci, ponieważ dla pozornie niewykorzystywanych pól atrap (zaciemnione pola na schemacie) jest wiele zastosowań. Na przykład można tam przechowywać licznik dowiązań albo

wskaźnik do prawego końca Listy, albo nazwę symboliczną, albo pole „robocze” wykorzystywane przez algorytm przechodzenia Listy itp.

Na schemacie (6) element zawierający b jest atomem, podczas gdy element zawierający f oznacza Listę pustą. Te dwa obiekty są strukturalnie identyczne, zatem całkiem usprawiedliwione jest pytanie: dlaczego w ogóle mówimy o „atomach”. Bez straty ogólności moglibyśmy zdefiniować Listę jako „skończony ciąg zera lub więcej List” przy ogólnym założeniu, że każdy element Listy może zawierać dodatkowe informacje poza dowiązaniami strukturalnymi. Ten punkt widzenia nietrudno obronić, co czyni pojęcie „atomu” dosyć sztucznym. Istnieje jednak dobry powód, dla którego wyróżniamy atomy – oszczędna gospodarka pamięcią. Atomy nie podlegają tym samym operacjom co Listy. Reprezentacja (6) ujawnia, że prawdopodobnie w elemencie atomowym b jest więcej miejsca na informację niż w elemencie f . Ponadto gdy korzystamy z atrap, jak w (7), różnica między rozmiarem pamięci potrzebnej do przechowywania elementów b i f jest znaczna. Stąd pojęcie atomów wprowadzamy głównie w celach oszczędności pamięci. Typowe Listy zawierają o wiele więcej atomów niż to mogłoby wynikać z naszych przykładów. W przykładach (4)–(7) są pokazane przypadki złożone, a nie najczęściej spotykane.

Lista to w istocie nic więcej niż lista liniowa, której elementy mogą zawierać wskaźniki do innych List. Najczęściej spotykane operacje na Listach to po prostu standardowe operacje na listach (tworzenie, niszczenie, wstawianie, usuwanie, podział, konkatenacja) oraz kilka operacji, z którymi zetknęliśmy się, omawiając drzewa (kopiowanie, przechodzenie, wprowadzanie i wyprowadzanie informacji o zagnieżdżonej strukturze). Realizację tych operacji umożliwiają trzy podstawowe sposoby reprezentowania list liniowych w pamięci – dowiązania jednokierunkowe, dwukierunkowe i cykliczne. Oczywiście wybór konkretnego wariantu ma wpływ na wydajność algorytmu operującego na strukturze. Dla poszczególnych wariantów reprezentacja w pamięci schematu (7) może wyglądać następująco:

Adres w pamięci	Dowiązania jednokierunkowe			Dowiązania cykliczne			Dowiązania dwukierunkowe			
	INFO	DLINK	RLINK	INFO	DLINK	RLINK	INFO	DLINK	LLINK	RLINK
010:	—	atrapa	020	—	atrapa	020	—	atrapa	050	020
020:	<i>a</i>	060	030	<i>a</i>	060	030	<i>a</i>	060	010	030
030:	<i>b</i>	atom	040	<i>b</i>	atom	040	<i>b</i>	atom	020	040
040:	<i>c</i>	090	050	<i>c</i>	090	050	<i>c</i>	090	030	050
050:	<i>e</i>	010	Λ	<i>e</i>	010	010	<i>e</i>	010	040	010
060:	—	atrapa	070	—	atrapa	070	—	atrapa	080	070
070:	<i>f</i>	110	080	<i>f</i>	110	080	<i>f</i>	110	060	080
080:	<i>g</i>	120	Λ	<i>g</i>	120	060	<i>g</i>	120	070	060
090:	—	atrapa	100	—	atrapa	100	—	atrapa	100	100
100:	<i>d</i>	060	Λ	<i>d</i>	060	090	<i>d</i>	060	090	090
110:	—	atrapa	Λ	—	atrapa	110	—	atrapa	110	110
120:	—	atrapa	130	—	atrapa	130	—	atrapa	140	130
130:	<i>h</i>	010	140	<i>h</i>	010	140	<i>h</i>	010	120	140
140:	<i>j</i>	060	Λ	<i>j</i>	060	120	<i>j</i>	060	130	120

(8)

Pole **LLINK** w reprezentacji listy dwukierunkowej wskazuje na węzeł z lewej. Pola **INFO** i **DLINK** we wszystkich reprezentacjach mają to samo znaczenie.

Nie ma potrzeby powtórnego omawiania operacji na tych trzech reprezentacjach, ponieważ już wiele razy to czyniliśmy. Należy jednak zauważyć następujące własności odróżniające Listy od omówionych wcześniej ich szczególnych przypadków:

1) Niejawnie zakładamy, że dla pokazanych powyżej reprezentacji elementy atomowe są odróżnialne od elementów nieatomowych. Ponadto dla List dwukierunkowych lub cyklicznych dobrze jest w jakiś sposób odróżnić atrapy od właściwych elementów, bo to ułatwi przechodzenie Listy. Zakładamy z tego powodu, że każdy element Listy zawiera pole **TYPE** opisujące jego typ. Możemy korzystać z tego pola w celu odróżniania wielu typów atomów (na przykład atomy reprezentujące wartości alfanumeryczne, całkowite, rzeczywiste).

2) Format elementu Listy na komputerze **MIX** można zaprojektować na dwa sposoby.

a) W formacie jednosłownowym zakładamy, że pola **INFO** występują jedynie w atomach:

S	T	REF	RLINK
---	---	-----	-------

(9)

S (znak): Znacznik wykorzystywany przy odśmiecaniu (zobacz poniżej).

T (typ): $T = 0$ dla atrapy; $T = 1$ dla pod-Listy; $T > 1$ dla atomów.

REF: Gdy $T = 0$, **REF** jest licznikiem dowiązań (zobacz poniżej); gdy $T = 1$, **REF** wskazuje na atrapę pod-Listy; gdy $T > 1$, **REF** wskazuje na węzeł zawierający bit-znacznik i 5 bajtów informacji atomowej.

RLINK: Dowiązanie strukturalne (dla Listy jednokierunkowej lub cyklicznej, porównaj (8)).

b) Format dwusłownowy:

S	T	LLINK	RLINK
INFO			

(10)

S, T: Jak w (9).

LLINK, RLINK: Dowiązania strukturalne dla Listy dwukierunkowej (porównaj (8)).

INFO: Całe słowo informacji związanej z elementem. Dla atrapy to pole może zawierać licznik dowiązań, pomocnicze pole wskaznikowe wykorzystywane w algorytmie przechodzenia Listy, nazwę symboliczną itd. Dla $T = 1$ pole zawiera **DLINK**.

3) Jasne jest, że Listy są strukturami bardzo ogólnymi. Uzasadnione wydaje się stwierdzenie, że dowolną strukturę można reprezentować za pomocą Listy.

Z uwagi na tę uniwersalność opracowano wiele systemów udostępniających operacje na Listach. Takie systemy wykorzystują zazwyczaj uniwersalny format elementu podobny do (9) lub (10) zaprojektowany z myślą o elastyczności systemu. Widać, że takie podejście nie musi zawsze być najlepszym rozwiązaniem w *konkretnych* przypadkach, a czas przetwarzania osiągany przy użyciu uniwersalnych operacji na Listach jest często zauważalnie dłuższy niż czas, który uzyskalibyśmy, ręcznie dopasowując system do konkretnego zadania. Łatwo jest na przykład zauważyć, że w prawie wszystkich problemach, z którymi zetknęliśmy się do tej pory, można skorzystać z uniwersalnych List w reprezentacji (9) lub (10), zamiast różnych struktur proponowanych przez nas dla każdego problemu. Ale algorytm przetwarzający Listę musi często sprawdzać pole T, a tego nie musielibyśmy robić do tej pory w żadnym z programów. Strata wydajności jest w wielu przypadkach rekompensowana względną łatwością programowania i zmniejszeniem czasu potrzebnego na uruchamianie i wyszukiwanie błędów, jeśli korzystamy z uniwersalnego systemu operującego na Listach.

4) Pomiędzy algorytmami prezentowanymi wcześniej a algorytmami wykorzystującymi Listy jest jeszcze jedna olbrzymia różnica. Z uwagi na fakt, że jedna Lista może być elementem wielu innych List, nie jest jasne, kiedy pamięć zajmowaną przez Listę można zwrócić na stertę. Do tej pory we wszystkich algorytmach wykonywaliśmy „**AVAIL** \leftarrow X” wszędzie tam, gdzie pamięć zajmowana przez **NODE(X)** przestawała być potrzebna. Ale Listy mogą rosnąć i ginąć zupełnie nieprzewidywalnie, trudno więc powiedzieć, kiedy konkretny element staje się zbędny. Z tego powodu problem gospodarowania stertą jest dla List o wiele trudniejszy niż w dotychczas napotykanych przypadkach. Końcówkę tego punktu poświęcimy omówieniu problemu odzyskiwania niewykorzystywanej pamięci.

Wyobraźmy sobie, że projektujemy uniwersalny system przetwarzania List, z którego będą korzystać setki programistów. Istnieją dwie zasadnicze metody gospodarowania stertą: *liczniki dowiązań* oraz *odśmiecanie*. W metodzie liczników dowiązań korzysta się z nowego pola w każdym elemencie, które zawiera liczbę „strzałek, które kończą się w tym elemencie”. Dosyć łatwo utrzymywać takie liczniki podczas działania programu. Gdy wartość licznika maleje do zera, konkretny element przestaje być potrzebny. W metodzie odśmiecania wymaga się obecności w każdym elemencie jednobitowego pola zwanego *znacznikiem (mark bit)*. Pomysł polega na pisaniu algorytmów w ten sposób, by nie przejmowały się zwracaniem nieużywanych elementów na stertę. Program działa bez przeskódeł do momentu, w którym brakuje wolnej pamięci. W tym momencie uruchamiany jest specjalny podprogram, który korzystając ze znaczników, „odzyskuje” niepotrzebne elementy i zwraca je na stertę. Po tym „odśmiecaniu” wznowiany jest program główny.

Żadna z tych metod nie jest całkowicie satysfakcjonująca. Podstawową wadą metody liczników dowiązań jest to, że nie zawsze odzyskuje się całą nieużywaną pamięć. Metoda działa nieźle w przypadku nakładających się List (tzn. List zawierających te same pod-Listy), ale pamięć zajmowana przez Listy rekurencyjne, jak *L* i *N* w przykładzie (4), nigdy nie zostanie zwrócona na stertę. Liczniki list rekurencyjnych będą niezerowe (ponieważ te listy zawierają dowiązania do

siebie) nawet wtedy, gdy żadna inna Lista spośród List dostępnych programowi nie zawiera dowiązań do nich. Co więcej, metoda liczników dowiązań wymaga zarezerwowania sporego fragmentu pamięci na licznik dowiązań w każdym elemencie (z drugiej strony czasami ta pamięć i tak pozostaje niewykorzystana z powodu rozmiaru słowa maszynowego).

Wady metody odśmiecania, poza koniecznością odżałowania jednego bitu w każdym elemencie, polegają na tym, że program korzystający z odśmiecania działa bardzo powoli w sytuacjach, gdy prawie cała pamięć jest rzeczywiście zajęta. W takich przypadkach wynik działania podprogramu odśmiecującego polegający na odzyskaniu kilku zaledwie wolnych komórek zupełnie nie wart jest poświęconego czasu. Programy, które przekraczają rozmiar dostępnej pamięci (a wiele nieprzetestowanych programów tak robi!), marnują zazwyczaj wiele czasu na wielokrotne wywoływanie programu odśmiecania, zanim okaże się, że wolna pamięć jednak rzeczywiście się wyczerpała. Częściowym rozwiązaniem tego problemu jest pozwolenie programiście podania liczby k , mówiącej, że nie warto kontynuować wykonania programu, jeśli program odśmiecania odzyskał k lub mniej komórek.

Inny problem polega na tym, że czasem trudno określić, które Listy w danym momencie nie są śmieciami. Jeśli programista używa niestandardowych technik albo przechowuje wartości wskaźnikowe w nietypowych miejscach, to istnieje duże prawdopodobieństwo, że odśmiecanie się nie uda. Największe tajemnice w historii usuwania błędów z programów polegały na tym, że w niespodziewanym momencie działania programu, który setki razy działał poprawnie, wykonywane było niefortunne odśmiecanie. Metoda odśmiecania wymaga również od programisty, by przez cały czas działania programu zmienne wskaźnikowe zawierały sensowne informacje, a przecież często wygodnie jest pozostawiać nieaktualne informacje w zmiennych, których nie zamierzamy używać – na przykład dowiązanie w końcowym elemencie kolejki; zobacz ćwiczenie 2.2.3–6.

Choć odśmiecanie wymaga przeznaczenia jednego bitu (znacznika) na każdy element struktury danych, możemy jednak wszystkie te bity zebrać w tablicę przechowywaną w innym miejscu, ustalając odpowiedniość między lokacją elementu a położeniem jego znacznika. Na niektórych maszynach taka metoda może być bardziej atrakcyjna niż przeznaczanie jednego bitu każdego elementu.

J. Weizenbaum zaproponował ciekawą modyfikację metody liczników dowiązań dla List dwukierunkowych. Modyfikacja polega na umieszczeniu licznika dowiązań wyłącznie w atrapach List. Zmienne wskazujące na elementy Listy nie są zliczane w licznikach poszczególnych elementów. Jeśli znamy zasady utrzymywania liczników dowiązań dla całych List, to wiemy (w teorii), jak nie odwoływać się do List, których licznik odwołań pokazuje zero. Możemy również zupełnie swobodnie ignorować liczniki dowiązań i zwracać wybrane Listy na stertę. Powyższe pomysły wymagają jednak dużej ostrożności; opisane metody są niebezpieczne w rękach niedoświadczonych programistów, utrudniają poza tym znacznie uruchamianie i wyszukiwanie błędów ze względu na konsekwencje odwoływania się do elementu, który został usunięty. Najprzyjemniejsza część podejścia Wizenbauma to traktowanie List, których licznik dowiązań zmalał do zera: takie

Listy dołączamy na *koniec* listy reprezentującej stertę (łatwo to zrealizować na listach dwukierunkowych), a jej elementy uznaje się za wolną pamięć dopiero po zużyciu wszystkich wcześniej dostępnych komórek pamięci. W końcu, gdy poszczególne elementy Listy zostaną uznane za wolną pamięć, liczniki dowiązań List, na które wskazują te elementy, zostaną zmniejszone o jeden. Takie opóźnione usuwanie List jest wydajne w sensie czasu wykonania, ale sprawia, że niepoprawne programy przez pewien czas działają poprawnie! Szczegóły w: *CACM 6* (1963), 524–544.

Algorytmy odśmiecania są interesujące z kilku powodów. Przede wszystkim taki algorytm jest pożyteczny w sytuacjach, gdy chcemy bezpośrednio lub pośrednio oznaczyć wszystkie elementy, do których prowadzą dowiązania z danego elementu. (Możemy na przykład chcieć znaleźć wszystkie podprogramy wywołane bezpośrednio lub pośrednio przez dany podprogram, jak w ćwiczeniu 2.2.3–26).

Ogólnie rzecz biorąc, odśmiecanie wykonujemy w dwóch fazach. Zakładamy, że początkowo wszystkie znaczniki są równe zeru (albo ustawiamy je na zero). W pierwszej fazie oznaczamy wszystkie elementy nie będące śmieciami, zaczynając od tych, na które bezpośrednio wskazują zmienne programu. W drugiej fazie wykonujemy sekwencyjne przejście przez całą pamięć, umieszczając wszystkie nie zaznaczone elementy na liście wolnej pamięci. Faza oznaczania jest bardziej interesująca, zatem skupimy się na niej. Pewne wariacje w fazie drugiej mogą jednakże uczynić ją nietrywialną; zobacz ćwiczenie 9.

Podczas działania algorytmu odśmiecania dysponujemy jedynie bardzo małym fragmentem wolnej pamięci. Ten intrygujący problem za chwilę się wyjaśni. Jest to trudność niedoceniana przez większość ludzi, którzy po raz pierwszy słyszą o odśmiecaniu, a przez wiele lat nie było jasne, jak sobie z nią radzić.

Poniższy algorytm oznaczania jest zapewne najbardziej oczywisty.

Algorytm A (Oznaczanie). Założmy, że pamięć przeznaczona na Listy to komórki NODE(1), NODE(2), ..., NODE(M) oraz że te słowa są albo atomami, albo zawierają dwa pola z dowiązaniemi ALINK i BLINK. Założmy ponadto, że wszystkie elementy są początkowo *nie oznaczone*. Celem działania tego algorytmu jest *oznaczanie* wszystkich elementów, do których można dojść, przechodząc po dowiązaniach ALINK i/lub BLINK w elementach nieatomowych, wychodząc od elementów „dostępnych bezpośrednio”, tj. elementów, na które wskazują wartości w pewnych ustalonych lokacjach programu głównego. Zakładamy, że dostęp do pamięci odbywa się wyłącznie za pomocą tych ustalonych wskaźników.

A1. [Inicjowanie] Zaznacz wszystkie bezpośrednio dostępne elementy. Przyjmij $K \leftarrow 1$.

A2. [Czy NODE(K) pociąga inne?] Przyjmij $K_1 \leftarrow K + 1$. Jeśli NODE(K) jest atomowy lub nieoznaczony, to przejdź do A3. W przeciwnym razie, jeśli NODE(ALINK(K)) jest nieoznaczony, to oznacz go, i jeśli nie jest atomowy, to przypisz $K_1 \leftarrow \min(K_1, ALINK(K))$. Podobnie, jeśli NODE(BLINK(K)) jest nieoznaczony, to go oznacz, a jeśli nie jest atomowy, to przypisz $K_1 \leftarrow \min(K_1, BLINK(K))$.

- A3.** [Czy gotowe?] Przyjmij $K \leftarrow K_1$. Jeśli $K \leq M$, to wróć do kroku A2; w przeciwnym razie zakończ wykonanie algorytmu. ■

W opisie powyższego algorytmu oraz kolejnych algorytmów w tym punkcie zakładamy dla wygody, że nie istniejący element „ $\text{NODE}(\Lambda)$ ” jest oznaczony. (Na przykład $\text{ALINK}(K)$ lub $\text{BLINK}(K)$ może równać się Λ w kroku A2).

Inny wariant algorytmu A różni się tym, że w kroku A1 przypisujemy $K_1 \leftarrow M + 1$, z kroku A2 usuwamy operację „ $K_1 \leftarrow K_1 + 1$ ”, a krok A3 zmieniamy na

- A3'.** [Czy gotowe?] Przyjmij $K \leftarrow K + 1$. Jeśli $K \leq M$, to wróć do kroku A2.

W przeciwnym razie jeśli $K_1 \leq M$, to przypisz $K \leftarrow K_1$, $K_1 \leftarrow M + 1$ i wróć do kroku A2. W pozostałych przypadkach zakończ działanie algorytmu.

Bardzo trudno podać dokładną analizę algorytmu A lub stwierdzić, który z wariantów jest lepszy, ponieważ nie narzuca się żaden sensowny rozkład prawdopodobieństwa danych wejściowych. Możemy powiedzieć, że w najgorszym przypadku wykonanie algorytmu zajmuje czas proporcjonalny do nM , gdzie n jest liczbą oznaczanych komórek. W ogólności możemy mieć pewność, że algorytm jest wolny dla dużych n . Zbyt wolny, by w praktyce wykorzystać go do odśmieciania.

Inny oczywisty algorytm oznaczania przechodzi wszystkie ścieżki, odkładając na stos punkty rozgałęzień:

Algorytm B (Oznaczanie). Ten algorytm wykonuje to samo zadanie co algorytm A, korzystając z pamięci pomocniczej $\text{STACK}[1]$, $\text{STACK}[2]$, ... do zapamiętywania wszystkich ścieżek, którymi jeszcze nie przeszedł.

- B1.** [Inicjowanie] Niech T będzie liczbą elementów dostępnych bezpośrednio. Zaznacz je i umieść wskaźniki do nich w $\text{STACK}[1], \dots, \text{STACK}[T]$.

- B2.** [Czy stos pusty?] Jeśli $T = 0$, to zakończ wykonanie algorytmu.

- B3.** [Usuwanie elementu] Przyjmij $K \leftarrow \text{STACK}[T]$, $T \leftarrow T - 1$.

- B4.** [Badanie dowiązań] Jeśli $\text{NODE}(K)$ jest atomem, to wróć do kroku B2. W przeciwnym razie, jeśli $\text{NODE}(\text{ALINK}(K))$ jest nieoznaczony, oznacz go i wykonaj $T \leftarrow T + 1$, $\text{STACK}[T] \leftarrow \text{ALINK}(K)$; jeśli $\text{NODE}(\text{BLINK}(K))$ jest nieoznaczony, to oznacz go i wykonaj $T \leftarrow T + 1$, $\text{STACK}[T] \leftarrow \text{BLINK}(K)$. Wróć do kroku B2. ■

Czas wykonania algorytmu B jest oczywiście proporcjonalny do liczby oznaczonych komórek i trudno się spodziewać lepszego wyniku. Niestety, ten algorytm też nie nadaje się do praktycznego wykorzystania, ponieważ nie ma miejsca na przechowywanie pomocniczego stosu! Można rozsądnie założyć, że stos w algorytmie B urośnie do, powiedzmy, pięciu procent rozmiaru pamięci. Ale w chwili wywołania podprogramu odśmiecającego cała pamięć jest zajęta, możemy liczyć tylko na mały obszar pamięci o stałym rozmiarze. Wiele wcześniejszych algorytmów odśmieciania bazowało na takim podejściu. Jeśli specjalna przestrzeń stosu się wyczerpała, cały program kończył się z błędem.

Można uzyskać nieco lepsze rozwiązanie, w którym jest wykorzystywany stos o ustalonym rozmiarze, łącząc algorytmy A i B:

Algorytm C (Oznaczanie). Ten algorytm wykonuje to samo zadanie co algorytmy A i B, korzystając z pamięci pomocniczej o rozmiarze H, STACK[0], STACK[1], ..., STACK[H - 1].

W opisie tego algorytmu operacja „włóż X na stos” oznacza: „Wykonaj $T \leftarrow (T + 1) \bmod H$ i $\text{STACK}[T] \leftarrow X$. Jeśli $T = B$, to $B \leftarrow (B + 1) \bmod H$, $K1 \leftarrow \min(K1, \text{STACK}[B])$ ”. (Zwróćmy uwagę, że T wskazuje teraz na aktualny wierzchołek stosu, a B wskazuje na miejsce tuż pod aktualnym dnem stosu; STACK działa w istocie jak kolejka dwustronna o ograniczonym wejściu).

- C1.** [Inicjowanie] Przyjmij $T \leftarrow H - 1$, $B \leftarrow H - 1$, $K1 \leftarrow M + 1$. Oznacz wszystkie elementy dostępne bezpośrednio i kolejno włóż ich lokacje na stos (wg powyższego opisu).
- C2.** [Czy stos pusty?] Jeśli $T = B$, to przejdź do C5.
- C3.** [Usuwanie elementu] Przyjmij $K \leftarrow \text{STACK}[T]$, $T \leftarrow (T - 1) \bmod H$.
- C4.** [Badanie dowiązań] Jeśli NODE(K) jest atomem, to powróć do kroku C2. W przeciwnym razie, jeśli NODE(ALINK(K)) jest nieoznaczony, to oznacz go i wstaw ALINK(K) na stos. Podobnie jeśli NODE(BLINK(K)) jest nieoznaczony, to oznacz go i wstaw BLINK(K) na stos. Wróć do C2.
- C5.** [Zamiatanie] Jeśli $K1 > M$, to zakończ wykonanie algorytmu. (Zmienna K1 reprezentuje najmniejszą lokację, która być może prowadzi do elementu, który powinien być oznaczony). W przeciwnym razie, jeśli NODE(K1) jest atomem lub jest nieoznaczony, zwiększ K1 o 1 i powtórz ten krok. Jeśli NODE(K1) jest oznaczony, to przyjmij $K \leftarrow K1$, zwiększ K1 o 1 i przejdź do C4. ■

Ten algorytm i algorytm B można poprawić, jeżeli nie będziemy wkładać X na stos, gdy NODE(X) jest atomem. Co więcej, w krokach B4 i C4 nie trzeba wkładać elementu na stos, jeżeli wiadomo, że zostanie on zaraz zdjęty. Takie modyfikacje są oczywiste, ale nie wprowadziliśmy ich, by nie komplikować opisu.

Algorytm C jest równoważny algorytmowi A, gdy $H = 1$, i algorytmowi B, gdy $H = M$. Staje się on coraz bardziej efektywny, gdy zwiększamy H. Niestety, algorytm C wymyka się precyzyjnej analizie z tych samych powodów co algorytm A, nie wiemy więc, jak duże powinno być H, by ta metoda była wystarczająco szybka. Wydaje się, że wartość $H = 50$ wystarczy, by algorytm C w większości wypadków nadawał się do odśmiecania.

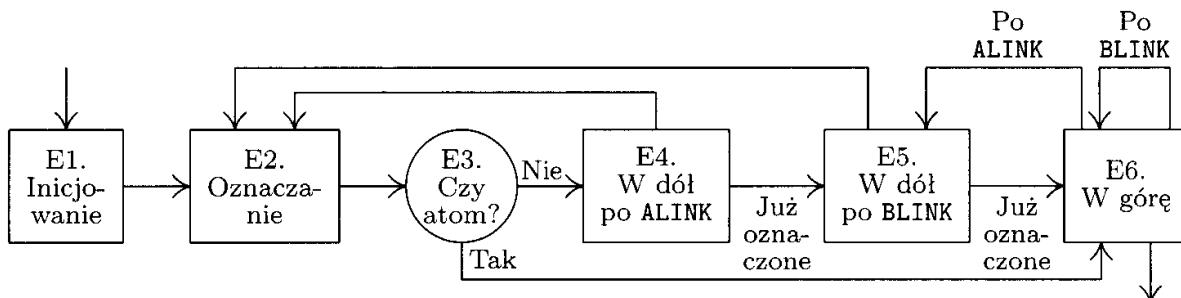
Algorytmy B i C korzystają ze stosu przechowywanego w kolejnych lokacjach pamięci. My przekonaliśmy się jednak, że metody wskaźnikowe nadają się do reprezentowania stosów, których elementy są rozrzucone po pamięci. To sugeruje, że w algorytmie B można posługiwać się stosem w jakiś sposób zanurzonym w *ten sam obszar pamięci, na którym wykonujemy odśmiecanie*. Można by tego łatwo dokonać, gdybyśmy zapewnili procedurze odśmiecania nieco więcej pamięci. Przypuśćmy, że wszystkie Listy są reprezentowane wg (9), z wyjątkiem pól REF w atrapach, które są zarezerwowane dla algorytmu odśmiecania zamiast na liczniki dowiązań. Możemy zmienić algorytm B w ten sposób, by posługiwał się stosem przechowywanym w polach REF atrap:

Algorytm D (Oznaczanie). Ten algorytm wykonuje to samo zadanie co algorytmy A, B i C, ale przy założeniu, że elementy mają pola S, T, REF i RLINK zamiast ALINK i BLINK. Pole S jest wykorzystywane na znacznik; jeśli $S(P) = 1$, to element NODE(P) jest oznaczony.

- D1. [Inicjowanie] Przyjmij $\text{TOP} \leftarrow \Lambda$. Następnie dla każdego dowiązania P do atrapy bezpośrednio dostępnej Listy (zobacz krok A1 algorytmu A), jeśli $S(P) = 0$, wykonaj $S(P) \leftarrow 1$, $\text{REF}(P) \leftarrow \text{TOP}$, $\text{TOP} \leftarrow P$.
- D2. [Czy stos pusty?] Jeśli $\text{TOP} = \Lambda$, to zakończ wykonanie algorytmu.
- D3. [Usuwanie elementu] Przyjmij $P \leftarrow \text{TOP}$, $\text{TOP} \leftarrow \text{REF}(P)$.
- D4. [Przejście Listy] Przyjmij $P \leftarrow \text{RLINK}(P)$; jeśli $P = \Lambda$ lub jeśli $T(P) = 0$, to przejdź do D2. W przeciwnym razie $S(P) \leftarrow 1$. Jeśli $T(P) > 1$, to $S(\text{REF}(P)) \leftarrow 1$ (tym samym oznaczając atom) W przeciwnym razie ($T(P) = 1$) wykonaj $Q \leftarrow \text{REF}(P)$; jeśli $Q \neq \Lambda$ i $S(Q) = 0$, to przyjmij $S(Q) \leftarrow 1$, $\text{REF}(Q) \leftarrow \text{TOP}$, $\text{TOP} \leftarrow Q$. Powtórz krok D4. ■

Algorytm D można przymierzyć do algorytmu B, który jest podobny i ma czas działania proporcjonalny do liczby oznaczonych elementów. *Nie* zalecamy jednak bezkrytycznego stosowania algorytmu D, gdyż jego pozorne łagodne wymagania często okazują się zbyt surowe dla uniwersalnych systemów przetwarzania List. Algorytm wymaga, by zawsze w momencie odśmieciania wszystkie Listy były poprawnie zbudowane, jak (7). Ale algorytmy operujące na Listach chwilowo utrzymują je w stanie niespójnym, co uniemożliwia poprawne wykonanie odśmieciania. Trzeba ponadto uważać w kroku D1, gdy program zawiera wskaźniki do wnętrza Listy.

Powyzsze rozważania prowadzą do algorytmu E, który jest elegancką metodą oznaczania, odkrytą niezależnie przez Petera Deutscha oraz przez Herberta Schorra i W. M. Waite'a w 1965 roku. Założenia tego algorytmu różnią się tylko trochę od założeń algorytmów A–D.



Rys. 38. Algorytm oznaczania E nie korzystający z pomocniczego stosu.

Algorytm E (Oznaczanie). Założymy, że dany jest zbiór elementów o następujących polach:

- MARK (jeden bit),
- ATOM (jeden bit),
- ALINK (wskaźnik),
- BLINK (wskaźnik).

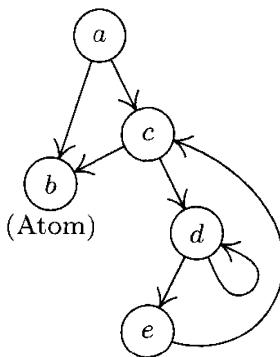
Gdy $\text{ATOM} = 0$, pola **ALINK** i **BLINK** mogą zawierać Λ lub wskaźnik do innych pól o tym samym formacie. Gdy $\text{ATOM} = 1$, zawartość pól **ALINK** i **BLINK** nie ma znaczenia dla algorytmu.

Dla niepustego wskaźnika **P0** algorytm ustawia na 1 pole **MARK** elementu **NODE(P0)** i wszystkich elementów osiągalnych z **NODE(P0)** za pomocą przejścia po wskaźnikach **ALINK** i **BLINK** elementów, dla których $\text{ATOM} = \text{MARK} = 0$. Algorytm korzysta z trzech zmiennych wskaźnikowych **T**, **Q** i **P**. Podczas wykonania algorytmu modyfikowane są dowiązania i bity kontrolne, ale po zakończeniu pracy wszystkie pola **ATOM**, **ALINK** i **BLINK** mają pierwotne wartości.

- E1.** [Inicjowanie] Przyjmij $T \leftarrow \Lambda$, $P \leftarrow P0$. (Od tej chwili podczas wykonania algorytmu zmienna **T** ma podwójne znaczenie: jeśli $T \neq \Lambda$, to **T** wskazuje na wierzchołek czegoś, co w istocie jest stosem z algorytmem D; element, na który wskazuje **T**, zawierał dowiązanie równe **P** w miejscu „sztucznego” strukturalnego dowiązania stosu, aktualnie zajmującego **NODE(T)**).
- E2.** [Oznaczanie] Przypisz $\text{MARK}(P) \leftarrow 1$.
- E3.** [Czy atom?] Jeśli $\text{ATOM}(P) = 1$, to przejdź do E6.
- E4.** [W dół po **ALINK**] Przypisz $Q \leftarrow \text{ALINK}(P)$. Jeśli $Q \neq \Lambda$ i $\text{MARK}(Q) = 0$, to przyjmij $\text{ATOM}(P) \leftarrow 1$, $\text{ALINK}(P) \leftarrow T$, $T \leftarrow P$, $P \leftarrow Q$ i przejdź do E2. (Pola **ATOM** i **ALINK** są tymczasowo zmieniane, tak że kształt Listy w okolicy niektórych oznaczonych elementów diametralnie się zmienia. Pierwotne wartości przywrócimy w kroku E6).
- E5.** [W dół po **BLINK**] Przyjmij $Q \leftarrow \text{BLINK}(P)$. Jeśli $Q \neq \Lambda$ i $\text{MARK}(Q) = 0$, to przyjmij $\text{BLINK}(P) \leftarrow T$, $T \leftarrow P$, $P \leftarrow Q$ i przejdź do E2.
- E6.** [W góre] (W tym kroku odwracamy rezultat przełączenia dowiązań wykonanych w krokach E4 i E5; wartość **ATOM(T)** informuje, czy należy odtworzyć **ALINK(T)** czy **BLINK(T)**). Jeśli $T = \Lambda$, to zakończ wykonanie algorytmu. W przeciwnym razie przyjmij $Q \leftarrow T$. Jeśli $\text{ATOM}(Q) = 1$, to przyjmij $\text{ATOM}(Q) \leftarrow 0$, $T \leftarrow \text{ALINK}(Q)$, $\text{ALINK}(Q) \leftarrow P$, $P \leftarrow Q$ i wróć do E5. Jeśli $\text{ATOM}(Q) = 0$, to przyjmij $T \leftarrow \text{BLINK}(Q)$, $\text{BLINK}(Q) \leftarrow P$, $P \leftarrow Q$ i powtórz krok E6. ■

Przykład działania tego algorytmu widać na rysunku 39, gdzie są pokazane kolejne kroki wykonania go dla prostej Listy. Warto przyjrzeć się temu przykładowi, by zrozumieć, na czym polegają sztuczne zmiany dowiązań w krokach E4 i E5 mające na celu zasymulowanie stosu występującego w algorytmie D. Gdy wracamy do poprzedniego stanu, korzystamy z pola **ATOM**, by dowiedzieć się, które z pól **ALINK**, **BLINK** zawiera sztuczny adres. „Zagnieżdżenia” pokazane na dole rysunku 39 ilustrują, w jaki sposób każdy element nieatomowy jest podczas wykonania algorytmu E odwiedzany trzykrotnie: ta sama konfiguracja (**T,P**) pojawia się na początku kroków E2, E5 i E6.

Dowód poprawności algorytmu E można sformułować za pomocą indukcji względem liczby elementów, które należy oznaczyć. Dowodzimy jednocześnie, że po wykonaniu algorytmu **P** ma na powrót początkową wartość **P0**; szczegóły można znaleźć w ćwiczeniu 3. Algorytm E działałby szybciej, gdybyśmy usunęli



<i>a</i>	ALINK [MARK]	<i>b</i> [0]	.	Λ [1]	.	<i>b</i>	
	BLINK [ATOM]	<i>c</i> [0]	.	[1]	.	[0]	Λ	<i>c</i>	
<i>b</i>	ALINK [MARK]	-[0]	.	.	[1]	
	BLINK [ATOM]	-[1]	
<i>c</i>	ALINK [MARK]	<i>b</i> [0]	[1]	
	BLINK [ATOM]	<i>d</i> [0]	<i>a</i>	<i>d</i>	.	.	.		
<i>d</i>	ALINK [MARK]	<i>e</i> [0]	c [1]	.	.	<i>e</i>	
	BLINK [ATOM]	<i>d</i> [0]	[1]	.	.	[0]	
<i>e</i>	ALINK [MARK]	Λ [0]	[1]	
	BLINK [ATOM]	<i>c</i> [0]	
	T	—	Λ	<i>a</i>	<i>a</i>	Λ	<i>a</i>	<i>a</i>	<i>c</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>c</i>	<i>c</i>	<i>a</i>	Λ						
	P	—	<i>a</i>	<i>b</i>	<i>b</i>	<i>a</i>	<i>c</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>e</i>	<i>e</i>	<i>d</i>	<i>d</i>	<i>c</i>	<i>a</i>						
Kolejny krok	E1	E2	E2	E6	E5	E2	E5	E2	E2	E5	E6	E5	E6	E6	E6	E6	E6	E6	E6	E6	E6	
Zagnieżdżenia																						

Rys. 39. Lista oznaczana przez algorytm E. (W tabeli są pokazane wyłącznie zmiany w stosunku do poprzedniego kroku).

krok E3 i sprawdzali warunek „ATOM(Q) = 1” oraz podejmowali odpowiednie działania w krokach E4 i E5, a także sprawdzali, czy „ATOM(P0) = 1” w kroku E1. Powstrzymaliśmy się jednak od tych optymalizacji, mając na uwadze prostotę opisu. Zaprezentowane modyfikacje omawiamy w ćwiczeniu 4.

Pomysł wykorzystany w Algorytmie E ma zastosowanie nie tylko przy odśmiecaniu. W istocie o jego zastosowaniu do przechodzenia drzew mówiliśmy już w ćwiczeniu 2.3.1–21. Można porównać algorytm E z rozwiązaniem problemu z ćwiczenia 2.2.3–7.

Spośród wszystkich omówionych algorytmów jedynie algorytm D daje się wprost zastosować do List reprezentowanych według (9). Inne algorytmy sprawdzają, czy dany element P jest atomem, a według konwencji (9) nie da się tego łatwo sprawdzić, ponieważ informacja atomowa może wypełniać cały element poza bitem znacznika. Niemniej jednak wszystkie te algorytmy można zmodyfikować w ten sposób, by działały w przypadku, gdy słowo z informacją atomową jest

odróżnialne od słowa zawierającego wskaźniki na podstawie zawartości elementu, który na nie wskazuje. W algorytmach A i C możemy po prostu nie oznaczać słów atomowych, dopóki nie oznaczymy pozostałych słów. Gdy wszystkie słowa nieatomowe są oznaczone, wystarcza jedno przejście po wszystkich danych, by oznaczyć osiągalne słowa atomowe. Modyfikacja algorytmu B jest jeszcze prostsza, ponieważ wystarczy nie wkładać słów atomowych na stos. Przystosowanie algorytmu E jest prawie tak samo proste, chociaż jeśli oba wskaźniki **ALINK** i **BLINK** mogą wskazywać na dane atomowe, to trzeba wprowadzić dodatkowe pole jednobitowe w elementach nieatomowych. To nie jest specjalnie trudne. (Jeśli na przykład na element przypadają dwa słowa, to można wykorzystać najmniej znaczące bity wskaźników).

Chociaż algorytm E wymaga czasu proporcjonalnego do liczby oznaczanych elementów, jego stała proporcjonalności jest większa niż algorytmu B; najszybsze znane metody odśmiecania polegają na połączeniu algorytmu B i E, zobacz ćwiczenie 5.

Spróbujmy teraz ilościowo oszacować wydajność odśmiecania jako alternatywę dla filozofii „**AVAIL** \Leftarrow **X**”, której byliśmy wierni w większości dotychczasowych przykładów. W każdym z omawianych przypadków moglibyśmy pominąć jawne zwracanie elementów na listę wolnej pamięci i w zamian korzystać z programu odśmiecania. (W konkretnych zastosowaniach, w przeciwnieństwie do uniwersalnych podprogramów operujących na Listach, zaprogramowanie i uruchomienie programu odśmiecania jest trudniejsze niż metody wykorzystywane przez nas do tej pory; przy tym oczywiście odśmiecanie wymaga zarezerwowania dodatkowego bitu w każdym elemencie. Ale nas interesuje w tej chwili względna różnica szybkości działania napisanych i przetestowanych programów).

Najlepsze znane podprogramy odśmiecania mają czas wykonania postaci $c_1N + c_2M$, gdzie c_1 i c_2 są stałymi, N jest liczbą oznaczanych elementów, a M jest całkowitą liczbą elementów w pamięci. $M - N$ jest zatem liczbą znalezionych wolnych elementów, a czas potrzebny na odzyskanie każdego z nich równa się $(c_1N + c_2M)/(M - N)$ na każdy element. Niech $N = \rho M$; wówczas wzór na czas przyjmuje postać $(c_1\rho + c_2)/(1 - \rho)$. Jeśli zatem $\rho = \frac{3}{4}$, tj. jeśli pamięć jest zajęta w trzech czwartych, to na odzyskanie elementu zużywamy $3c_1 + 4c_2$ jednostek czasu. Gdy $\rho = \frac{1}{4}$, ten koszt wynosi tylko $\frac{1}{3}c_1 + \frac{4}{3}c_2$. Jeśli nie korzystamy z odśmiecania, to czas odzyskania elementu jest równy stałej c_3 , a wartość c_3/c_1 raczej nie będzie zbyt duża. Widzimy zatem, jak bardzo niewydajne jest odśmiecanie przy dużej zajętości pamięci.

Wiele programów ma tę własność, że stosunek $\rho = N/M$ używanych elementów do rozmiaru pamięci jest mały. Gdy w takich przypadkach sterta się zapełnia, dobrym pomysłem może okazać się skopiowanie wszystkich aktywnych List do bliższej sterty (zobacz ćwiczenie 10), bez dbałości o zachowanie wartości pól wskaźnikowych. Gdy ta druga sterta się zapełni, kopujemy Listy z powrotem do pierwszej sterty itd. Dzięki tej metodzie w szybkiej pamięci można naraz przechowywać więcej danych, ponieważ wskaźniki nie będą zazwyczaj wskazywały odległych lokacji. Nie ma ponadto potrzeby oznaczania elementów, a przydział pamięci jest tablicowy.

Można połączyć odśmiecanie z niektórymi innymi metodami zwalniania pamięci; te pomysły nie wykluczają się, a w niektórych systemach korzysta się jednocześnie z liczników dowiązań, odśmiecania i jawnego zwalniania pamięci. Pomyśl polega na wykorzystaniu odśmiecania na zasadzie „ostatniej deski ratunku”, gdy wszystkie inne metody zwalniania pamięci zawiodły. Rozbudowany system korzystający z tego pomysłu, a także zawierający mechanizm odraczania operacji na licznikach dowiązań w celu uzyskania większej wydajności został opisany przez L. P. Deutscha i D. G. Bobrowa w *CACM 19* (1976), 522–526.

Można również korzystać z tablicowej reprezentacji List, co zwalnia nas z konieczności utrzymywania wielu pól wskaźnikowych kosztem bardziej skomplikowanego zarządzania pamięcią. Zobacz N. E. Wiseman, J. O. Hiles, *Comp. J. 10* (1968), 338–343; W. J. Hansen, *CACM 12* (1969), 499–506 oraz C. J. Cheney, *CACM 13* (1970), 677–678.

Daniel P. Friedman i David S. Wise zauważyli, że w wielu przypadkach można z powodzeniem korzystać z liczników dowiązań, nawet jeśli Listy zawierają wskaźniki do samych siebie, jeżeli utrzymując liczniki, nie będziemy uwzględniać pewnych dowiązań [*Inf. Proc. Letters 8* (1979), 41–45].

Zaproponowano olbrzymią rozmaitość wariantów algorytmów odśmiecania i poprawek do nich. Jacques Cohen w *Computing Surveys 13* (1981), 341–367, przedstawia szczegółowy przegląd literatury związanej z tymi zagadnieniami (do 1981 roku) wraz z komentarzami dotyczącymi dodatkowego kosztu dostępu do pamięci, gdy strony danych są przerzucane między pamięcią szybką i powolną.

Zaprezentowane przez nas metody odśmiecania nie sprawdzają się w systemach „czasu rzeczywistego”, gdzie każda podstawowa operacja na Liście musi trwać krótko; jeśli nawet program do odśmiecania włącza się sporadycznie, to za każdym razem potrzebuje sporo czasu. W ćwiczeniu 12 omawiamy pewne podejścia do problemu odśmiecania w systemach czasu rzeczywistego.

Wielka szkoda, że jest dzisiaj tak mało bezużytecznych informacji.
— OSCAR WILDE (1894)

ĆWICZENIA

- ▶ 1. [M21] W punkcie 2.3.4 pokazaliśmy, że drzewa są szczególnym przypadkiem „klasycznych” obiektów matematycznych – grafów skierowanych. Czy Listy też można opisać w terminologii teoriografowej?
- 2. [20] W punkcie 2.3.1 pokazaliśmy, że dzięki wykorzystaniu fastrygi daje się wygodniej przechodzić drzewa. Czy analogicznie można sfastrygować Listy?
- 3. [M26] Udowodnij poprawność algorytmu E. [*Wskazówka:* Zapoznaj się z dowodem poprawności algorytmu 2.3.1T].
- 4. [28] Napisz program na maszynę MIX realizujący algorytm E przy założeniu, że element jest reprezentowany za pomocą jednego słowa, MARK zajmuje pole (0:0) [,+, = 0, ,-, = 1], ATOM zajmuje pole (1:1), ALINK pole (2:3), BLINK pole (4:5) oraz $\Lambda = 0$. Wyznacz czas wykonania Twojego programu względem istotnych parametrów. (*Uwaga:* Na maszynie MIX problem stwierdzenia, czy komórka pamięci zawiera -0 czy +0 nie jest zupełnie trywialny, a to może mieć znaczenie w Twoim programie).

5. [25] (Schorr, Waite) Podaj algorytm oznaczania, będący połączeniem algorytmów B i E w sposób następujący: zachowujemy założenia algorytmu E dotyczące pól elementów itd., korzystamy jednak z pomocniczego stosu **STACK[1]**, **STACK[2]**, ..., **STACK[N]** jak w algorytmie B, odwołując się do mechanizmu z algorytmu E tylko wtedy, gdy stos się przepędzi.

6. [00] W omówieniu wydajności odśmiecania na końcu rozdziału pada stwierdzenie, że koszt odśmiecania wynosi w przybliżeniu $c_1N + c_2M$ jednostek czasu. Skąd się bierze składnik „ c_2M ”?

7. [24] (R. W. Floyd) Zaprojektuj algorytm oznaczania podobny do algorytmu E, lecz nie wykorzystujący dodatkowego stosu, z tym że (i) wykonujący trudniejsze zadanie, ponieważ każdy element zawiera wyłącznie pola **MARK**, **ALINK** i **BLINK** – nie ma pola **ATOM**; ale jednocześnie (ii) wykonujący zadanie łatwiejsze, ponieważ jesteśmy zainteresowani oznaczaniem węzłów drzewa binarnego, a nie Listy w ogólności. Stąd **ALINK** i **BLINK** pełnią role dowiązań **LLINK** i **RLINK** drzewa binarnego.

- ▶ **8.** [27] (L. P. Deutsch) Zaprojektuj algorytm podobny do algorytmów D i E z uwagi na to, że nie korzysta z pomocniczej pamięci na stos. Zmodyfikuj metodę tak, by działała dla elementów o zmiennym rozmiarze i zmiennej liczbie wskaźników o następującym formacie: pierwsze słowo elementu ma dwa pola **MARK** i **SIZE**. Pole **MARK** należy traktować jak w algorytmie E, pole **SIZE** zawiera liczbę $n \geq 0$. Oznacza ona, że każde z n kolejnych słów występujących w pamięci bezpośrednio po tym (pierwszym) słowie zawiera dwa pola: **MARK** (które ma wartość zero i taka wartość ma w nim pozostać) oraz **LINK** (równe Λ lub zawierające wskaźnik do pierwszego słowa innego elementu). Na przykład na element zawierający trzy wskaźniki składają się cztery kolejne słowa:

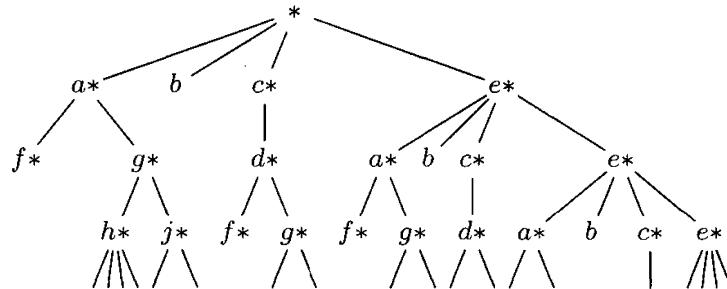
Pierwsze słowo	MARK = 0 (będzie ustawione na 1)	SIZE = 3
Drugie słowo	MARK = 0	LINK = pierwszy wskaźnik
Trzecie słowo	MARK = 0	LINK = drugi wskaźnik
Czwarte słowo	MARK = 0	LINK = trzeci wskaźnik.

Twój algorytm powinien oznaczać elementy osiągalne z danego elementu **P0**.

- ▶ **9.** [28] (D. Edwards) Zaprojektuj algorytm drugiej fazy odśmiecania, który „kompresuje pamięć” w następującym sensie: Niech **NODE(1), ..., NODE(M)** będą jednosłownymi elementami o polach **MARK**, **ATOM**, **ALINK** i **BLINK**, jak było opisane w algorytmie E. Założmy, że **MARK** = 1 dla wszystkich istotnych elementów. Algorytm powinien przenieść (jeśli to konieczne) wszystkie oznaczone elementy, by znajdowały się w kolejnych lokalizacjach **NODE(1), ..., NODE(K)**, jednocześnie powinien zmodyfikować pola **ALINK** i **BLINK** elementów nieatomowych, by zachować pierwotny kształt Listy.
- ▶ **10.** [28] Zaprojektuj algorytm kopiący Listę przy założeniu, że korzystamy z reprezentacji (7). (Jeśli zatem Twój algorytm ma skopiować Listę, której atrapa na schemacie (7) znajduje się w górnym lewym rogu, to powinien stworzyć nowy zbiór List o 14 elementach, zawierających te same informacje i składających się na strukturę tego samego kształtu, co (7)).

Zakładamy, że Lista jest reprezentowana za pomocą pól **S**, **T**, **REF** i **RLINK**, jak na schemacie (9), oraz że **NODE(P0)** jest atrapą Listy, którą należy skopiować. Zakładamy ponadto, że pole **REF** w każdej atrapie jest równe Λ . By zminimalizować zużycie pamięci, Twój algorytm może wykorzystywać pole **REF** (ale powinien przywrócić ich początkowe wartości Λ).

11. [M30] Każdą Listę można „rozwinąć” w drzewo, powielając nakładające się elementy, aż do ich wyczerpania. Dla List rekurencyjnych otrzymujemy drzewa nieskończone. Na przykład Lista (5) rozwija się w nieskończone drzewo, którego pierwsze cztery poziomy wyglądają tak:



Zaprojektuj algorytm sprawdzający równoważność dwóch List, w sensie równoważności ich rozwinięć. Na przykład przy poniższych oznaczeniach listy A i B są równoważne:

$$A = (a: C, b, a: (b: D))$$

$$B = (a: (b: D), b, a: E)$$

$$C = (b: (a: C))$$

$$D = (a: (b: D))$$

$$E = (b: (a: C)).$$

12. [30] (M. Minsky) Pokaż, że można korzystać z odśmiecania w „systemach czasu rzeczywistego”, na przykład gdy komputer steruje jakimś urządzeniem mechanicznym, nawet gdy górne ograniczenie na czas wykonania pojedynczej operacji na Liście jest bardzo surowe. [Wskazówka: Odśmiecanie można tak zorganizować, by było wykonywane równolegle z operacjami na Listach].

2.4. STRUKTURY Z WIELOMA DOWIĄZANAMI

Teraz, gdy szczegółowe omówienie list liniowych oraz struktur drzewiastych mamy już za sobą, podstawowe zasady reprezentowania informacji strukturalnych w pamięci komputera powinny być oczywiste. W tym rozdziale przyjrzymy się innym zastosowaniom poznanych metod, pojawiających się w przypadkach, gdy informacje mają bardziej złożoną strukturę.

„Struktura z wieloma dowiązaniami (*multilinked structure*)” składa się z elementów o wielu dowiązaniach, w odróżnieniu od jednego lub dwóch dowiązań występujących w strukturach omawianych do tej pory. Spotkaliśmy się już z parami przykładami wielokierunkowych dowiązań, na przykład w systemie windy w punkcie 2.2.5 lub w reprezentacji wielomianów wielu zmiennych w punkcie 2.3.3.

Przekonamy się, że występowanie wielu rodzajów dowiązań w jednym elemencie *nie* musi koniecznie oznaczać, że algorytm operujący na takiej strukturze jest istotnie bardziej skomplikowany do zapisania lub zrozumienia. Omówimy także istotne zagadnienie „*Jaką część informacji strukturalnych należy reprezentować w pamięci?*”.

Problem, którym się zajmiemy, pojawia się przy pisaniu kompilatorów dla języka COBOL i jemu podobnych. Programista korzystający z języka COBOL może nadawać wielopoziomowe nazwy symboliczne zmiennym programu. Program może na przykład korzystać z pliku danych dotyczących sprzedaży i zakupów o następującej strukturze:

1 SALES	1 PURCHASES
2 DATE	2 DATE
3 MONTH	3 DAY
3 DAY	3 MONTH
3 YEAR	3 YEAR
2 TRANSACTION	2 TRANSACTION
3 ITEM	3 ITEM
3 QUANTITY	3 QUANTITY
3 PRICE	3 PRICE
3 TAX	3 TAX
3 BUYER	3 SHIPPER
4 NAME	4 NAME
4 ADDRESS	4 ADDRESS

Powyższa konfiguracja oznacza, że każda pozycja **SALES** składa się z dwóch części: **DATE** i **TRANSACTION**; pozycja **DATE** składa się z trzech części, a **TRANSACTION** z pięciu. Podobnie w przypadku **PURCHASES**. Względny porządek tych nazw wyznacza kolejność, w jakiej odpowiadające im wielkości występują w pliku danych (na przykład na taśmie magnetycznej lub na wydruku). Zauważmy, że w tym przykładzie „**DAY**” i „**MONTH**” występują w różnym porządku w różnych plikach. Programista może podać dodatkowe informacje (nie pokazane w przykładzie), mówiące o tym, ile miejsca zajmuje każda dana i w jakim jest formacie. Te szczegóły są dla nas bez znaczenia, więc bedziemy je pomijać.

Programista piszący w języku COBOL najpierw opisuje układ pliku i podaje deklaracje innych zmiennych programu, następnie specyfikuje algorytm operujący na tych wielkościach. By odwołać się do zmiennej z powyższego przykładu, nie wystarczy podać jej nazwę DAY, ponieważ nie wiadomo, czy w takim przypadku chodzi o DAY z pliku SALES, czy z pliku PURCHASES. Z tego powodu w języku COBOL piszemy „DAY OF SALES”, by odwołać się do zmiennej DAY będącej częścią pozycji SALES. Programista może posłużyć się także pełniejszym opisem

„DAY OF DATE OF SALES”,

ale zasada jest taka, że wystarczy podać tyle informacji uściślających, by dało się rozwikłać niejednoznaczności. Stąd

„NAME OF SHIPPER OF TRANSACTION OF PURCHASES”

można skrócić do

„NAME OF SHIPPER”

ponieważ tylko jedna część danych nazywa się SHIPPER.

Powyższe reguły obowiązujące w języku COBOL można zestawić w następujący sposób:

- Przed każdą nazwą podajemy dodatnią liczbę całkowitą nazywaną *numerem poziomu*. Nazwa odnosi się albo do *pozycji elementarnej*, albo do *grupy* jednej lub więcej pozycji o nazwach występujących bezpośrednio po nazwie grupy. Wszystkie pozycje grupy muszą mieć ten sam numer poziomu, który musi być większy niż numer poziomu nazwy grupy. (Na przykład DATE i TRANSACTION mają numer poziomu równy 2, większy od numeru poziomu nazwy SALES równego 1).
- Ogólna postać odwołania do pozycji elementarnej lub grupy pozycji o nazwie A_0 to

$A_0 \text{ OF } A_1 \text{ OF } \dots \text{ OF } A_n,$

gdzie $n \geq 0$ i dla $0 \leq j < n$ nazwa A_j oznacza pewną pozycję zawartą pośrednio lub bezpośrednio w grupie o nazwie A_{j+1} . Musi istnieć dokładnie jedna pozycja A_0 spełniająca ten warunek.

- Jeśli ta sama nazwa A_0 występuje w wielu miejscach, to musi istnieć sposób odwołania się do tej nazwy poprzez uściślanie.

Reguła (c) wyklucza na przykład następującą konfigurację danych

1	AA	(2)
2	BB	
3	CC	
3	DD	
2	CC	

ponieważ nie ma jednoznacznego sposobu odwołania się do drugiego wystąpienia CC (zobacz ćwiczenie 4).

W języku COBOL jest jeszcze inny mechanizm mający wpływ na budowę kompilatora. Istnieje konstrukcja języka pozwalająca odwoływać się do wielu pozycji jednocześnie. W COBOL-u można napisać

MOVE CORRESPONDING α TO β

co powoduje skopiowanie wartości wszystkich pozycji o odpowiednich nazwach z obszaru danych α do obszaru danych β . Na przykład instrukcja

MOVE CORRESPONDING DATE OF SALES TO DATE OF PURCHASES

powoduje skopiowanie wartości pozycji MONTH, DAY i YEAR z pliku SALES do zmiennych DAY, MONTH, YEAR w pliku PURCHASES. (Względny porządek DAY i MONTH zostanie zmieniony).

Problem, którym zajmiemy się w tym rozdziale, polega na zaprojektowaniu trzech algorytmów przeznaczonych do wykorzystania w kompilatorze języka COBOL, realizujących następujące zadania:

Operacja 1. Przeczytać opis nazw i numerów poziomów w postaci (1) oraz utworzyć na tej podstawie struktury danych, z których korzystać będą operacje 2 oraz 3.

Operacja 2. Stwierdzić, czy konkretne uściślone odwołanie (jak w regule (b)) jest poprawne i jeśli tak, znaleźć odpowiadającą mu pozycję danych.

Operacja 3. Znaleźć wszystkie pary odpowiadających sobie pozycji wyznaczonych przez konstrukcję CORRESPONDING.

Założymy, że nasz kompilator ma „podprogram obsługi tablicy symboli” przekształcający nazwę symboliczną na wskaźnik do pozycji odpowiadającej tej nazwie w specjalnej tablicy. (Metody konstruowania algorytmów związanych z tablicą symboli omawiamy szczegółowo w rozdziale 6). Poza Tablicą Symboli kompilator przechowuje większą tablicę zawierającą po jednej pozycji dla każdej pozycji danych komplikowanego programu. Będziemy nazywać tę strukturę *Tablicą Danych*.

Jest jasne, że nie możemy zaprojektować algorytmu wykonującego operację 1, nie wiedząc, jakie informacje należy umieścić w Tablicy Danych. Z kolei postać tej struktury zależy od tego, jakich informacji będziemy potrzebować, by wykonać operacje 2 i 3. Dlatego zaczniemy od przyjrzenia się tym operacjom.

By określić znaczenie odwołania w języku COBOL postaci

$$A_0 \text{ OF } A_1 \text{ OF } \dots \text{ OF } A_n, \quad n \geq 0, \quad (3)$$

powinniśmy zacząć od wyszukania nazwy A_0 w Tablicy Symboli. Z pozycji Tablicy Symboli powinny prowadzić dowiązania do wszystkich pozycji Tablicy Danych związanych z daną nazwą. Następnie dla każdej pozycji Tablicy Danych będziemy chcieli znaleźć dowiązanie do pozycji związanej z grupą, która ją zawiera. Jeśli teraz w Tablicy Danych napotkamy odniesienie prowadzące z powrotem do Tablicy Symboli, to nie trudno zgadnąć, w jaki sposób powinno być przetwarzane wyrażenie postaci (3). Chcielibyśmy ponadto dysponować jakimś rodzajem dowiązań prowadzących z pozycji Tablicy Danych odpowiadających grupom do pozycji odpowiadających elementom w grupach, by dało się znaleźć pary wyznaczane przez „MOVE CORRESPONDING”.

Zauważliśmy zatem, że jest potrzebnych pięć pól zawierających dowiązania w każdej pozycji P Tablicy Danych:

- PREV (wskaźnik do poprzedniej pozycji o tej samej nazwie co P, jeśli taka istnieje);
- PARENT (wskaźnik do najmniejszej grupy (jeśli istnieje) zawierającej P);
- NAME (wskaźnik do pozycji Tablicy Symboli odpowiadającej P);
- CHILD (wskaźnik do pierwszej podpozycji, jeśli P jest grupą);
- SIB (wskaźnik do następnej pozycji w grupie, do której należy P).

Widać, że struktury danych w języku COBOL, takie jak SALES i PURCHASES, są drzewami; dowiązania PARENT, CHILD i SIB znamy w istocie z poprzednich rozdziałów. (W standardowej reprezentacji drzewa za pomocą drzewa binarnego są używane dowiązania CHILD i SIB; w wyniku dodania dowiązania PARENT otrzymamy strukturę, którą nazywaliśmy „drzewem z trzema dowiązaniami”. Pięć powyższych dowiązań to wymienione trzy oraz dowiązania PREV i NAME, kodujące dodatkowe informacje).

Być może nie wszystkie pięć dowiązań okaże się niezbędne, a może jakieś dowiązanie trzeba będzie dodać. Spróbujmy jednak zaprojektować nasze algorytmy przy założeniu, że pozycje Tablicy Danych zawierają te pięć pól wskaźnikowych (oraz dodatkowe informacje nieistotne dla naszych algorytmów). W ramach przykładu przyjrzyjmy się strukturom

1 A	1 H
. 3 B	5 F
7 C	8 G
7 D	5 B
3 E	5 C
3 F	9 E
4 G	9 D
	9 G

(4)

Reprezentacja tych struktur jest pokazana na schemacie (5) (dowiązania są oznaczone symbolicznie). Pole LINK każdej pozycji Tablicy Symboli zawiera dowiązanie do ostatnio utworzonej pozycji Tablicy Danych związanego z danym symbolem.

Pierwszy z naszych algorytmów buduje Tablicę Danych w powyższej postaci. Zauważmy, że reguły języka COBOL umożliwiają elastyczny wybór numerów poziomów; lewa struktura w (4) jest równoważna strukturze

1 A
2 B
3 C
3 D
2 E
2 F
3 G

ponieważ numery poziomów nie muszą być kolejnymi liczbami.

Tablica Symboli

	LINK
A:	A1
B:	B5
C:	C5
D:	D9
E:	E9
F:	F5
G:	G9
H:	H1

Puste kratki oznaczają nieistotne dla nas informacje dodatkowe.

Tablica Danych

	PREV	PARENT	NAME	CHILD	SIB	
A1:	Λ	Λ	A	B3	H1	
B3:	Λ	A1	B	C7	E3	
C7:	Λ	B3	C	Λ	D7	
D7:	Λ	B3	D	Λ	Λ	
E3:	Λ	A1	E	Λ	F3	
F3:	Λ	A1	F	G4	Λ	
G4:	Λ	F3	G	Λ	Λ	
H1:	Λ	Λ	H	F5	Λ	
F5:	F3	H1	F	G8	B5	
G8:	G4	F5	G	Λ	Λ	
B5:	B3	H1	B	Λ	C5	
C5:	C7	H1	C	E9	Λ	
E9:	E3	C5	E	Λ	D9	
D9:	D7	C5	D	Λ	G9	
G9:	G8	C5	G	Λ	Λ	

(5)

Niektóre ciągi numerów poziomów są niedozwolone. Jeśli na przykład numer poziomu D w (4) zmienimy na „6” (w obu miejscach), to otrzymamy konfigurację danych, która nie spełnia warunku mówiącego, że wszystkie elementy grupy muszą mieć ten sam numer poziomu. Poniższy algorytm sprawdza, czy reguła (a) nie została naruszona.

Algorytm A (Budowania Tablicy Danych). Dane wejściowe algorytmu stanowią ciąg par (L, P) , gdzie L jest dodatnim całkowitym „numerem poziomu”, a P wskazuje pozycję Tablicy Symboli odpowiadającą deklaracji struktury danych w COBOL-u postaci (4). Algorytm buduje Tablicę Danych, taką jak (5) w powyższym przykładzie. Jeśli P wskazuje na element Tablicy Symboli, który wcześniej się nie pojawił, to $LINK(P)$ równa się Λ . Algorytm korzysta z pomocniczego stosu, realizowanego metodami tradycyjnymi (z wykorzystaniem tablicy, jak w punkcie 2.2.2, lub struktur z dowiązaniami, jak w punkcie 2.2.3).

- A1.** [Inicjowanie] Na pusty stos włożyć jeden element $(0, \Lambda)$. (Elementy stosu są parami (L, P) , gdzie L jest liczbą całkowitą, a P dowiązaniem; w czasie wykonania algorytmu stos zawiera numery poziomów i dowiązania do ostatnio napotkanych pozycji danych na wszystkich poziomach wyższych (w drzewie) niż poziom bieżący. Na przykład tuż przed przeczytaniem „3 F” w powyższym przykładzie, stos będzie zawierać

$$(0, \Lambda) \quad (1, A1) \quad (3, E3)$$

patrząc od dna do wierzchołka).

- A2.** [Następna pozycja] Zakończ wykonanie algorytmu, jeśli ciąg danych wejściowych się skończył. W przeciwnym razie niech (L, P) będzie kolejną daną z wejścia. Przyjmij $Q \leftarrow \text{AVAIL}$ (czyli niech Q będzie lokacją nowego elementu, w którym możemy umieścić nową pozycję Tablicy Danych).

- A3.** [Ustawienie dowiązania do nazw] Przyjmij

$$\text{PREV}(Q) \leftarrow \text{LINK}(P), \quad \text{LINK}(P) \leftarrow Q, \quad \text{NAME}(Q) \leftarrow P.$$

(Ustawiliśmy dwa z pięciu dowiązań w $\text{NODE}(Q)$. Jako następne będziemy ustawiać brakujące **PARENT**, **CHILD** i **SIB**).

- A4.** [Porównanie poziomów] Niech (L_1, P_1) będzie wierzchołkiem stosu. Jeśli $L_1 < L$, to przyjmij $\text{CHILD}(P_1) \leftarrow Q$ (lub jeśli $P_1 = \Lambda$, to przyjmij $\text{FIRST} \leftarrow Q$, gdzie **FIRST** jest zmienną wskazującą na pierwszą pozycję Tablicy Danych) i przejdź do A6.

- A5.** [Usuwanie ostatniego poziomu] Jeśli $L_1 > L$, to zdejmij element z wierzchołka stosu. Niech (L_1, P_1) będzie elementem, który właśnie stał się wierzchołkiem stosu; powtóż A5. Jeśli $L_1 < L$, to sygnalizuj błąd (różne numery poziomów na tym samym poziomie). W przeciwnym razie, czyli gdy $L_1 = L$, przypisz $\text{SIB}(P_1) \leftarrow Q$, zdejmij element z wierzchołka stosu i niech (L_1, P_1) będzie elementem, który właśnie stał się wierzchołkiem stosu.

- A6.** [Ustawianie wskaźników „rodzinnych”] Przyjmij wartości $\text{PARENT}(Q) \leftarrow P_1$, $\text{CHILD}(Q) \leftarrow \Lambda$, $\text{SIB}(Q) \leftarrow \Lambda$.

- A7.** [Wkładanie na stos] Włóż (L, Q) na stos i powróć do A2. ■

Dzięki wprowadzeniu pomocniczego stosu (co wyjaśniono w kroku A1) algorytm jest tak przejrzysty, że jakiekolwiek dalsze wyjaśnienia są zbędne.

Następny problem polega na znalezieniu pozycji Tablicy Danych odpowiadającej odwołaniu

$$A_0 \text{ OF } A_1 \text{ OF } \dots \text{ OF } A_n, \quad n \geq 0. \quad (6)$$

Dobry kompilator sprawdzi przy okazji, czy takie odwołanie jest jednoznaczne. W tym przypadku algorytm nasuwa się sam: wystarczy przejrzeć listę pozycji Tablicy Danych dla symbolu A_0 i upewnić się, czy dokładnie jedna z tych pozycji pasuje do uściślenia A_1, \dots, A_n .

Algorytm B (*Sprawdzanie odwołań uściślonych*). Podprogram obsługuje Tablicę Symboli wyznaczyc dowiązania P_0, P_1, \dots, P_n do pozycji Tablicy Symboli odpowiadających nazwom A_0, A_1, \dots, A_n (oznaczenia jak w (6)).

Niniejszy algorytm ma zbadać P_0, P_1, \dots, P_n i stwierdzić, że odwołanie (6) jest błędne, albo przypisać do zmiennej Q adres pozycji Tablicy Danych odpowiadającej pozycji danych programu wyznaczonej przez odwołanie (6).

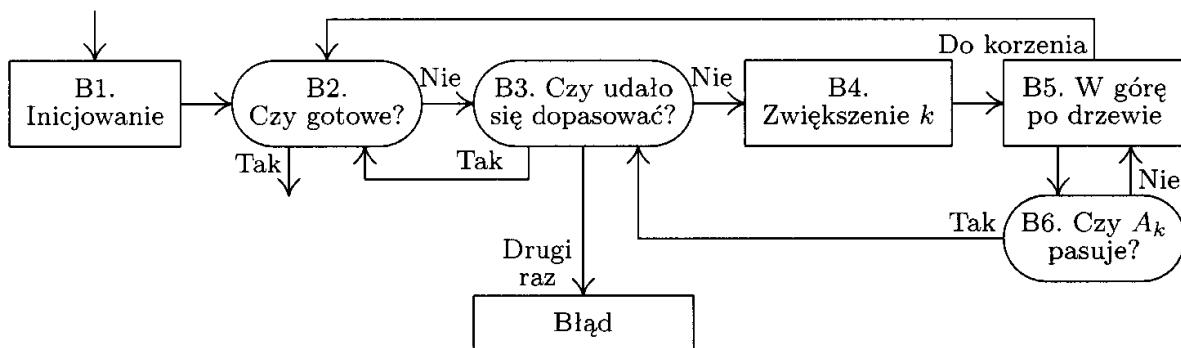
- B1.** [Inicjowanie] Przyjmij $Q \leftarrow \Lambda$, $P \leftarrow \text{LINK}(P_0)$.

- B2.** [Czy gotowe?] Jeśli $P = \Lambda$, to zakończ wykonanie algorytmu. W tym momencie Q jest równe Λ , jeśli (6) nie odpowiada żadnej pozycji Tablicy Danych.

Ale jeśli $P \neq \Lambda$, to przyjmij $S \leftarrow P$ i $k \leftarrow 0$. (S jest zmienną wskaźnikową, która będzie się posuwać w górę drzewa po dowiązaniach **PARENT**, począwszy od elementu P ; k jest zmienną całkowitą przebiegającą liczby od 0 do n . W praktyce dowiązania P_0, \dots, P_n będą zapewne przechowywane na liście z dowiązaniami, a zamiast k użyjemy zmiennej wskaźnikowej przechodzącej po kolejnych pozycjach tej listy; zobacz ćwiczenie 5).

- B3.** [Czy udało się dopasować?] Jeśli $k < n$, to przejdź do B4. W przeciwnym razie znaleźliśmy pasującą pozycję Tablicy Danych. Jeśli $Q \neq \Lambda$, to jest to druga pasująca pozycja, zatem sygnalizuj błąd. Przypisz $Q \leftarrow P$ oraz $P \leftarrow \text{PREV}(P)$ i idź do B2.
- B4.** [Zwiększenie k] Przyjmij $k \leftarrow k + 1$.
- B5.** [W górę po drzewie] Przyjmij $S \leftarrow \text{PARENT}(S)$. Jeśli $S = \Lambda$, to nie udało się znaleźć pasującej pozycji; przyjmij $P \leftarrow \text{PREV}(P)$ i idź do B2.
- B6.** [Czy A_k pasuje?] Jeśli $\text{NAME}(S) = P_k$, to idź do B3, w przeciwnym razie idź do B5. ■

Zauważmy, że algorytm nie korzysta z dowiązań **CHILD** i **SIB**.



Rys. 40. Algorytm sprawdzania odwołań w języku COBOL.

Trzeci i ostatni algorytm jest związany z instrukcją „MOVE CORRESPONDING”. Zanim weźmiemy się za jego projektowanie, musimy dokładnie określić wymagania. Instrukcja

MOVE CORRESPONDING α TO β (7)

gdzie α i β są odwołaniami postaci (6) jest skrótnym zapisem ciągu wszystkich takich instrukcji

MOVE α' TO β'

że istnieje liczba całkowita $n \geq 0$ oraz n takich nazw A_0, A_1, \dots, A_{n-1} , że

$$\begin{aligned} \alpha' &= A_0 \text{ OF } A_1 \text{ OF } \dots \text{ OF } A_{n-1} \text{ OF } \alpha \\ \beta' &= A_0 \text{ OF } A_1 \text{ OF } \dots \text{ OF } A_{n-1} \text{ OF } \beta \end{aligned} \quad (8)$$

i α' lub β' jest pozycją elementarną (nie grupą). Wymagamy ponadto, by poziomy w (8) stanowiły uściślenia pełne, tj. by A_{j+1} było rodzicem A_j dla $0 \leq j < n$; α' i β' muszą leżeć dokładnie n poziomów niżej niż α i β .

Dla (4) instrukcja

MOVE CORRESPONDING A TO H

jest skrótem ciągu dwóch instrukcji

MOVE B OF A TO B OF H
MOVE G OF F OF A TO G OF F OF H

Algorytm wyznaczający wszystkie odpowiadające sobie pary α', β' jest interesujący, choć dosyć prosty. Przechodzimy drzewo α w porządku preorder, jednocześnie szukając pasujących nazw w drzewie β , pomijając drzewa, w których pasujące pozycje nie mogą się pojawić. Nazwy A_0, \dots, A_{n-1} z (8) napotykamy w odwrotnym porządku A_{n-1}, \dots, A_0 .

Algorytm C (*Znajdowania odpowiadających sobie par*). Dla wskaźników P_0 i Q_0 do pozycji α i β w Tablicy Danych algorytm znajduje wszystkie pary (P, Q) wskaźników do pozycji (α', β') spełniających opisane powyżej warunki.

- C1.** [Inicjowanie] Przyjmij $P \leftarrow P_0$, $Q \leftarrow Q_0$. (Zmienne wskaźnikowe P i Q będą przechodziły drzewa o korzeniach α i β).
- C2.** [Czy elementarne?] Jeśli $\text{CHILD}(P) = \Lambda$ lub $\text{CHILD}(Q) = \Lambda$, to wyprowadź (P, Q) jako jedną z poszukiwanych par i przejdź do C5. W przeciwnym razie przyjmij $P \leftarrow \text{CHILD}(P)$, $Q \leftarrow \text{CHILD}(Q)$. (W tym kroku P i Q wskazują na pozycje α' i β' spełniające (8), a my chcemy wykonać **MOVE** $\alpha' \rightarrow \beta'$ wtedy i tylko wtedy, gdy α' i/lub β' jest pozycją elementarną).
- C3.** [Dopasowanie nazwy] (P i Q wskazują na pozycje danych o uściśleniach postaci

$A_0 \text{ OF } A_1 \text{ OF } \dots \text{ OF } A_{n-1} \text{ OF } \alpha$

i

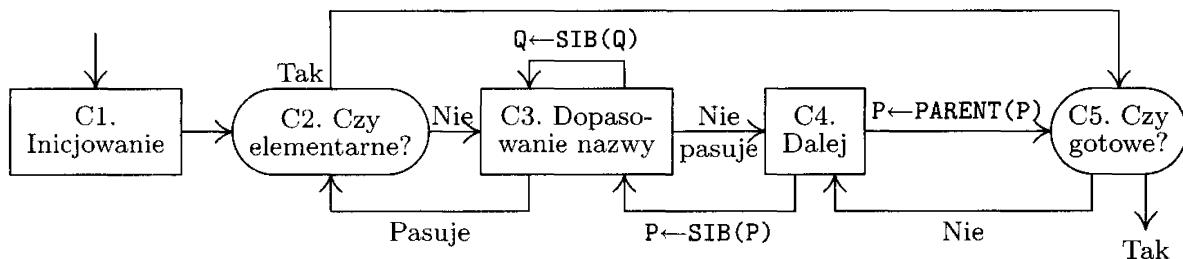
$B_0 \text{ OF } A_1 \text{ OF } \dots \text{ OF } A_{n-1} \text{ OF } \beta.$

Musimy sprawdzić, czy możemy uzyskać $B_0 = A_0$. W tym celu przeglądamy wszystkie nazwy w grupie $A_1 \text{ OF } \dots \text{ OF } A_{n-1} \text{ OF } \beta$. Jeśli $\text{NAME}(P) = \text{NAME}(Q)$, to idź do C2 (znaleźliśmy pasującą nazwę). W przeciwnym razie, jeśli $\text{SIB}(Q) \neq \Lambda$, to przyjmij $Q \leftarrow \text{SIB}(Q)$ i powtórz krok C3. (Jeśli $\text{SIB}(Q) = \Lambda$, to nie istnieje pasująca nazwa, przejdź więc do kroku C4).

- C4.** [Dalej] Jeśli $\text{SIB}(P) \neq \Lambda$, to przyjmij $P \leftarrow \text{SIB}(P)$, $Q \leftarrow \text{CHILD}(\text{PARENT}(Q))$ i przejdź do kroku C3. Jeśli $\text{SIB}(P) = \Lambda$, to $P \leftarrow \text{PARENT}(P)$, $Q \leftarrow \text{PARENT}(Q)$.
- C5.** [Czy gotowe?] Jeśli $P = P_0$, to zakończ wykonanie algorytmu. W przeciwnym razie idź do C4. ■

Schemat blokowy powyższego algorytmu jest przedstawiony na rysunku 41. Dowód poprawności można przeprowadzić przez indukcję względem rozmiaru drzew (zobacz ćwiczenie 9).

Warto w tym miejscu przyjrzeć się, w jaki sposób algorytmy B i C korzystają z pól **PREV**, **PARENT**, **NAME**, **CHILD** i **SIB**. Okazuje się, że te pięć dowiązań tworzy „zbiór pełny” w tym sensie, że nakład pracy związanej z przechodzeniem Tabeli Danych jest w algorytmach B i C minimalny. Jeśli tylko trzeba odwołać się do



Rys. 41. Algorytm realizujący „MOVE CORRESPONDING”.

innej pozycji Tablicy Danych, jej adres jest dostępny natychmiast – pozycji nie trzeba wyszukiwać. Trudno wyobrazić sobie, w jaki sposób można przyspieszyć algorytmy B i C przez dodanie jeszcze jakichś dowiązań (porównaj jednak ćwiczenie 11).

Na każde dowiązanie można patrzeć jak na *podpowiedź* dla programu, dzięki której pewne operacje można wykonać szybciej. (Oczywiście algorytm A budujący struktury danych działa wolniej, bo musi wypełnić więcej pól. Ale struktury danych inicjuje się tylko raz). Jasne jest, że przy naszych założeniach Tablica Danych zawiera wiele informacji nadmiarowych. Zastanówmy się, co by się stało, gdybyśmy *usunęli* jakieś pole z dowiązaniem.

Dowiązanie PREV, choć nie używane w algorytmie C, jest niezmiernie istotne w algorytmie B i wydaje się, że jego odpowiednik musi się znaleźć w strukturach danych każdego kompilatora języka COBOL, chyba że akceptujemy długotrwałe wyszukiwanie. Dowiązania łączące wszystkie pozycje o tej samej nazwie wydają się więc niezbędne z punktu widzenia wydajności. Moglibyśmy nieco zmodyfikować naszą strategię i zastosować dowiązania cykliczne, ale nie widać powodu, dla którego mielibyśmy to robić (chyba że usuniemy jakieś inne pola lub zmienimy ich znaczenie).

Z pola PARENT korzystamy w algorytmach B i C, ale algorytm C mógłby się bez niego obyć, jeżeli użylibyśmy pomocniczego stosu albo jeśli sfastrygowałibyśmy strukturę za pomocą pól SIB (zobacz punkt 2.3.2). Widzimy zatem, że dowiązanie PARENT jest istotne jedynie w algorytmie B. Jeśli poprowadzilibyśmy fastrygę przez pola SIB, tak że elementy, które teraz mają $SIB = \Lambda$, miałyby $SIB = PARENT$, moglibyśmy znaleźć rodzica dowolnego elementu, posuwając się po dowiązaniach SIB. Dowiązania fastrygi moglibyśmy odróżniać albo za pomocą nowego pola TAG w każdym elemencie (mówiącego, czy dowiązanie SIB jest fastrygą) albo sprawdzając warunek „ $SIB(P) < P$ ”, jeśli pozycje Tablicy Danych byłyby przechowywane w rosnących adresach pamięci zgodnie z porządkiem występowania w komplikowanym programie. Taka modyfikacja wymagałaby wprowadzenia wyszukiwania w kroku B5, przez co algorytm działałby wolniej.

Z dowiązania NAME korzystamy wyłącznie w krokach B6 i C3. W obu przypadkach dysponujemy sposobami sprawdzenia warunków „ $NAME(S) = P_k$ ” oraz „ $NAME(P) = NAME(Q)$ ” bez korzystania z pola NAME (zobacz ćwiczenie 10). To jednak spowolniłoby znacznie pętle wewnętrzne zarówno algorytmu B, jak i C. Po raz kolejny możemy zauważać wzajemność między wielkością obszaru pamięci

przeznaczonego na dowiązania a szybkością wykonania algorytmu. (Szybkość wykonania algorytmu C nie ma dużego znaczenia dla szybkości działania kompilatora języka COBOL, gdy weźmiemy pod uwagę typowe zastosowania konstrukcji MOVE CORRESPONDING; ale algorytm B powinien być szybki). Doświadczenie pokazuje, że w kompilatorze języka COBOL pole NAME ma inne ważne zastosowania, szczególnie przy wypisywaniu informacji diagnostycznej.

Algorytm A buduje Tablicę Danych krok po kroku i nigdy nie zwraca elementów na listę wolnej pamięci. Zazwyczaj okazuje się, że Tablica Danych zajmuje kolejne lokacje pamięci w porządku odpowiadającym kolejności występowania pozycji danych w komplikowanym programie. W takich okolicznościach lokacje A1, B3, ... w przykładzie (5) następowaliby bezpośrednio po sobie. Ten sekwenncyjny charakter Tablicy Danych prowadzi do pewnych uproszczeń; na przykład dowiązanie CHILD jest albo równe Λ , albo wskazuje na następny węzeł (w sensie położenia w pamięci). Pole CHILD można zatem zredukować do jednobitowego znacznika. Można również usunąć je zupełnie, zastępując pobranie wartości znacznika sprawdzeniem warunku PARENT(P + c) = P, gdzie c jest rozmiarem elementu Tablicy Danych.

Dochodzimy zatem do wniosku, że nie potrzeba wszystkich pięciu pól wskaźnikowych, niemniej jednak są one pożyteczne z punktu widzenia szybkości wykonania algorytmów B i C. Taka sytuacja jest typowa dla większości struktur z wieloma dowiązaniami.

Warto zauważyć, że przynajmniej sześć osób piszących kompilatory w języku COBOL we wczesnych latach sześćdziesiątych wpadło niezależnie na pomysł utrzymywania pięciu wskaźników w Tablicy Danych (lub czterech, zazwyczaj bez CHILD). Pierwsza publikacja na temat tej techniki pochodzi od: H. W. Lawsona, Jr. [ACM National Conference Digest (Syracuse, N.Y.: 1962)]. Ale w 1965 roku David Dahm opracował pomysłową metodę, za pomocą której można osiągnąć te same rezultaty, korzystając jedynie z dwóch dowiązań i Tablicy Danych, bez wielkiego uszczerobkę na szybkości działania; zobacz ćwiczenia 12–14.

ĆWICZENIA

1. [00] Konfiguracje danych w języku COBOL tworzą drzewa. W jakim porządku węzły tych drzew występują w programie w COBOL-u: preorder, postorder, czy w żadnym z nich?
2. [10] Omów czas wykonania algorytmu A.
3. [22] W języku PL/I można definiować struktury danych jak w języku COBOL, z tym że dopuszcza się dowolny ciąg numerów poziomów. Na przykład ciąg

1 A	1 A
3 B	2 B
5 C jest równoważny	3 C
4 D	3 D
2 E	2 E

W ogólności zmodyfikowana reguła (a) brzmi: „Numery poziomów pozycji występujących w jednej grupie muszą tworzyć ciąg nierosnący oraz muszą być większe niż numer

poziomu grupy". Jak należy zmienić algorytm A, by przystosować go do konwencji języka PL/I?

- 4. [26] Algorytm A nie sprawdza, czy programista piszący w języku COBOL nie złamał reguły (c). Jak należy zmodyfikować algorytm A, żeby przyjmował wyłącznie struktury spełniające regułę (c)?

- 5. [20] W praktyce dowiązania „ P_0, P_1, \dots, P_n ” do pozycji Tablicy Symboli można w algorytmie B przekazywać jako listę liniową. Niech T będzie taką zmienną wskaźnikową, że

$$\text{INFO}(T) \equiv P_0, \text{INFO}(\text{RLINK}(T)) \equiv P_1, \dots, \text{INFO}(\text{RLINK}^{[n]}(T)) \equiv P_n, \text{RLINK}^{[n+1]}(T) = \Lambda.$$

Pokaż, w jaki sposób należy zmodyfikować algorytm B, by korzystał z danych wejściowych tej postaci.

- 6. [23] Struktury w języku PL/I są podobne do struktur w COBOL-u, ale nie ma tam ograniczenia narzuconego przez regułę (c). Przyjmuje się za to, że uściślone odwołanie (3) jest jednoznaczne, jeżeli zawiera „pełne” uścielenie – tj. jeśli A_{j+1} jest rodzicem A_j dla $0 \leq j < n$ i jeśli A_n nie ma rodzica. Reguła (c) jest osłabiona do prostego warunku, że dwie pozycje w tej samej grupie nie mogą mieć tej samej nazwy. Do drugiego „CC” w (2) można się jednoznacznie odwołać za pomocą „CC OF AA”; do pozycji danych

1 A
2 A
3 A

można się odwołać za pomocą „A”, „A OF A”, „A OF A OF A”. (*Uwaga:* W istocie słowo „OF” jest w języku PL/I zastąpione symbolem kropki, a kolejność pozycji jest odwrócona; „CC OF AA” zapisujemy w PL/I jako „AA.CC”, ale w tym ćwiczeniu to nie ma znaczenia). Pokaż, w jaki sposób należy zmodyfikować algorytm B, żeby działał zgodnie z opisaną konwencją języka PL/I.

- 7. [15] Co w języku COBOL oznacza instrukcja „MOVE CORRESPONDING SALES TO PURCHASES” dla struktur danych (1)?

- 8. [10] Kiedy instrukcja „MOVE CORRESPONDING α TO β ” znaczy dokładnie to samo, co „MOVE α TO β ” (wg definicji w tekście rozdziału)?

- 9. [M23] Udowodnij, że algorytm C jest poprawny.

- 10. [23] (a) W jaki sposób można sprawdzić warunek „NAME(S) = P_k ” w kroku B6, jeśli pozbędziemy się pola NAME z elementów Tabeli Danych? (b) W jaki sposób przy tym samym założeniu można sprawdzić warunek „NAME(P) = NAME(Q)” w kroku C3? (Zakładamy, że dysponujemy pozostałymi dowiązaniami i że mają one znaczenie opisane w tekście).

- 11. [23] Czy wprowadzając dodatkowe dowiązania i/lub zmieniając strategię, można przyspieszyć algorytmy B i/lub C?

- 12. [25] (D. M. Dahm) Zastanówmy się nad możliwością reprezentowania Tablicy Danych w kolejnych lokacjach pamięci z dwoma polami zawierającymi dowiązania:

PREV (jak w tekście);

SCOPE (wskaźnik do ostatniej pozycji elementarnej w bieżącej grupie).

Mamy $\text{SCOPE}(P) = P$ wtedy i tylko wtedy, gdy $\text{NODE}(P)$ reprezentuje pozycję elementarną. Na przykład Tablica Danych (5) w tej nowej reprezentacji wyglądałaby tak:

	PREV	SCOPE		PREV	SCOPE		PREV	SCOPE
A1:	Λ	G4	F3:	Λ	G4	B5:	B3	B5
B3:	Λ	D7	G4:	Λ	G4	C5:	C7	G9
C7:	Λ	C7	H1:	Λ	G9	E9:	E3	E9
D7:	Λ	D7	F5:	F3	G8	D9:	D7	D9
E3:	Λ	E3	G8:	G4	G8	G9:	G8	G9

(Porównaj z (5) z punktu 2.3.3). Zauważmy, że $\text{NODE}(P)$ jest częścią poddrzewa $\text{NODE}(Q)$ wtedy i tylko wtedy, gdy $Q < P \leq \text{SCOPE}(Q)$. Zaprojektuj algorytm wykonujący zadanie algorytmu B dla Tablicy Danych w opisanej postaci.

- ▶ 13. [24] Podaj algorytm, którego należy użyć w miejsce algorytmu A, gdy Tablica Danych jest tak zorganizowana, jak w ćwiczeniu 12.
- ▶ 14. [28] Podaj algorytm, którego należy użyć w miejsce algorytmu C, gdy Tablica Danych jest tak zorganizowana, jak opisano w ćwiczeniu 12.
- 15. [25] (David S. Wise) Zmodyfikuj algorytm A w ten sposób, by nie potrzebował dodatkowej pamięci na stos. [*Wskazówka:* Pola SIB wszystkich węzłów, na które wskazują elementy stosu, są równe Λ].

2.5. DYNAMICZNY PRZYDZIAŁ PAMIĘCI

Nauczyliśmy się budować struktury danych z dowiązaniami, których elementy nie są kolejno ułożone w pamięci. Wiele struktur zajmujących wspólny obszar pamięci może niezależnie rosnąć lub maleć. Jednak do tej pory cały czas milcząco zakładaliśmy, że wszystkie elementy mają ten sam rozmiar, że każdy element zajmuje tyle samo komórek pamięci.

W wielu przypadkach można znaleźć rozsądny kompromis i we wszystkich strukturach danych posługiwać się węzłami o jednakowym rozmiarze (zobacz na przykład ćwiczenie 2). Zamiast stosować węzły o największym z potrzebnych rozmiarów, można posłużyć się węzłami mniejszymi oraz zasadą, którą można nazwać klasyczną filozofią wskaźnikową: „jeśli tu nie ma miejsca na jakąś informację, umieśćmy ją gdzie indziej, a tutaj trzymajmy wskaźnik do niej”.

Istnieje jednak nie mniejsza rzesza przypadków, w których stosowanie węzłów o jednakowym rozmiarze nie jest rozsądne. Często chcielibyśmy, by węzły o różnych rozmiarach przechowywane były w tym samym obszarze pamięci. Innymi słowy, chcemy dysponować algorytmami rezerwowania i zwalniania spójnych bloków pamięci o różnych rozmiarach. Tego rodzaju metody nazywamy algorytmami *dynamicznego przydziału pamięci*.

Czasami, na przykład w symulatorach, chcemy przydzielać dynamicznie pamięć na węzły małych rozmiarów (powiedzmy od jednego do dziesięciu słów). W innych przypadkach, na przykład w systemach operacyjnych, mamy do czynienia głównie z dużymi blokami. Te dwa punkty widzenia prowadzą do nieco odmiennych podejść, chociaż obie metody mają wiele wspólnego. By w opisie tych metod posługiwać się jednakową terminologią, będziemy w tym podrozdziale na oznaczenie spójnego ciągu lokacji używać terminów *bloki obszar* zamiast „węzeł”.

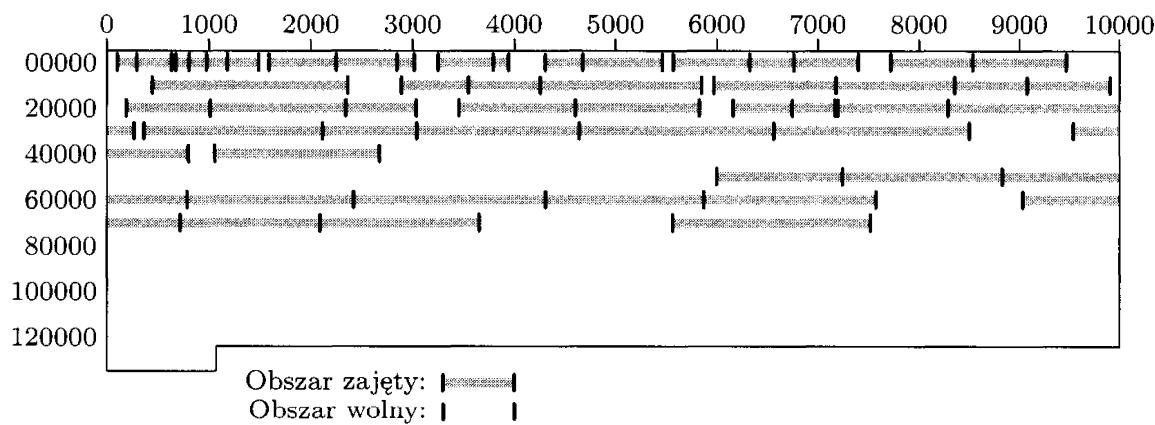
Niektórzy autorzy około 1975 roku zaczęli pulę dynamicznie przydzielanej pamięci nazywać „stertą”*.

A. Rezerwacja. Na rysunku 42 jest pokazana typowa *mapa pamięci* – schemat bieżącego stanu sterty. W tym przypadku pamięć jest podzielona na bloki, z których 53 są „zarezerwowane”, a 21 – „wolnych”. Jeśli program korzysta z dynamicznego przydziału pamięci, to po niedługim czasie pamięć wygląda właśnie w ten sposób. Nasz pierwszy problem polega na znalezieniu odpowiedzi na dwa pytania:

- a) W jaki sposób należy reprezentować stertę?
- b) W jaki sposób powinien działać algorytm, który dysponując reprezentacją sterty, wyszukuje i rezerwuje n kolejnych komórek pamięci?

Odpowiedź na pytanie (a) brzmi: przechowywać gdzieś listę wolnych bloków pamięci; prawie zawsze tę listę przechowuje się właśnie w tych wolnych blokach.

* W języku angielskim termin *heap* oznacza zarówno stertę, jak i kopiec (tj. pewną implementację kolejki priorytetowej, zobacz rozdział 6). Z tego powodu autor stara się unikać terminu „sterta” w kontekście przydziału pamięci. My jednak nie musimy i nie będziemy tego robić (przyp. tłum.).



Rys. 42. Mapa pamięci.

(Wyjątkiem jest przydział miejsca w pamięci dyskowej lub w innego rodzaju pamięci, dla której z uwagi na niejednorodność czasów dostępu lepiej przechowywać oddzielny katalog wolnych bloków).

Możemy powiązać wolne bloki w listę: pierwsze słowo każdego wolnego bloku może zawierać jego rozmiar i adres następnego wolnego bloku. Bloki można powiązać w porządku malejących lub rosnących rozmiarów, albo w porządku ich adresów, albo w zupełnie dowolnym porządku.

Spójrzmy na rysunek 42 ukazujący pamięć o rozmiarze 131072 słów adresowanych liczbami od 0 do 131071. Przypuśćmy, że chcemy powiązać wolne bloki w porządku ich adresów. Potrzebujemy do tego celu jednej zmiennej **AVAIL** wskazującej na pierwszy wolny blok (w tym przypadku **AVAIL** równa się 0). Pozostałe bloki reprezentujemy następująco:

<i>lokacja</i>	<i>SIZE</i>	<i>LINK</i>
0	101	632
632	42	1488
:	:	: [17 podobnych pozycji]
73654	1909	77519
77519	53553	Λ [specjalny znacznik ostatniego dowlądzania]

Lokacje 0–100 składają się na pierwszy wolny blok. Dalej, po dwóch zajętych obszarach 101–290 i 291–631 pokazanych na rysunku 42, mamy wolne miejsce w lokacjach 632–673 itd.

Zajmijmy się pytaniem (b): jeśli potrzebujemy n kolejnych słów pamięci, to musimy znaleźć pewien wolny blok o rozmiarze $m \geq n$ i zmniejszyć jego rozmiar do $m - n$. (Jeśli ponadto $m = n$, to musimy także usunąć ten blok z listy). Na stercie może być wiele bloków o rozmiarze nie mniejszym niż n . Powstaje więc pytanie: *który z nich wybrać?*

Narzucają się dwie podstawowe odpowiedzi: możemy skorzystać z metody *najlepszego dopasowania* albo *pierwszego dopasowania*. W pierwszym przypadku wybieramy blok o najmniejszym rozmiarze, nie mniejszym niż n . To może wymagać przeszukania całej listy wolnych bloków. Metoda pierwszego dopasowania polega po prostu na wyborze pierwszego znalezionego bloku, który jest dostatecznie duży.

Z historycznego punktu widzenia metoda najlepszego dopasowania była szeroko stosowana przez wiele lat. Wydaje się, że jest to dobra strategia, ponieważ zachowuje większe bloki pamięci do późniejszego użycia. Można jednak wysunąć przeciwko niej wiele zarzutów; jest powolna, ponieważ wymaga czasochłonnego przeszukiwania. Jeśli metoda najlepszego dopasowania nie jest z jakichś powodów istotnie lepsza niż metoda pierwszego dopasowania, to ta strata czasu zupełnie się nie opłaca. Co ważniejsze, metoda najlepszego dopasowania ma tendencję do tworzenia wielu małych bloków, a namnażanie małych bloków jest zazwyczaj niepożądane. Są sytuacje, gdy metoda pierwszego dopasowania jest lepsza od metody najlepszego dopasowania. Przypuśćmy na przykład, że mamy dwa bloki wolnej pamięci o rozmiarach 1300 i 1200, a program wysuwa kolejno żądania przydziału bloków o rozmiarach 1000, 1100 i 250:

<i>żądanie</i>	<i>wolne bloki, pierwsze dopasowanie</i>	<i>wolne bloki, najlepsze dopasowanie</i>	
—	1300, 1200	1300, 1200	(1)
1000	300, 1200	1300, 200	
1100	300, 100	200, 200	
250	50, 100	odmowa	

(Odwrotny przykład pokazujemy w ćwiczeniu 7). Problem polega na tym, że żadna metoda nie jest istotnie lepsza od drugiej, stąd poleca się metodę pierwszego dopasowania jako prostszą.

Algorytm A (*Metoda pierwszego dopasowania*). Niech AVAIL wskazuje na pierwszy wolny blok sterty. Przypuśćmy, że każdy wolny blok o adresie P ma dwa pola: $\text{SIZE}(P)$, zawierające liczbę słów w bloku, oraz $\text{LINK}(P)$, zawierające wskaźnik do następnego wolnego bloku. Ostatni wskaźnik równa się Λ . Algorytm znajduje i rezerwuje blok o rozmiarze N słów albo zgłasza błąd.

- A1.** [Inicjowanie] Przyjmij $Q \leftarrow \text{LOC}(\text{AVAIL})$. (Korzystamy z dwóch wskaźników Q i P , dla których zasadniczo zachodzi warunek $P = \text{LINK}(Q)$. Zakładamy, że $\text{LINK}(\text{LOC}(\text{AVAIL})) = \text{AVAIL}$).
- A2.** [Czy koniec listy?] Przyjmij $P \leftarrow \text{LINK}(Q)$. Jeśli $P = \Lambda$, to zakończ działania algorytmu, zgłaszając niepowodzenie; nie ma miejsca na blok o rozmiarze N słów.
- A3.** [Czy wystarczająco duży?] Jeśli $\text{SIZE}(P) \geq N$, to przejdź do A4; w przeciwnym razie przyjmij $Q \leftarrow P$ i wróć do kroku A2.
- A4.** [Zarezerwowanie N] Przyjmij $K \leftarrow \text{SIZE}(P) - N$. Jeśli $K = 0$, to przyjmij $\text{LINK}(Q) \leftarrow \text{LINK}(P)$ (usuwając tym samym pusty blok z listy); w przeciwnym razie $\text{SIZE}(P) \leftarrow K$. Zakończ (pomyślnie) wykonanie algorytmu, zarezerwowałwszy blok długości N zaczynający się w lokacji $P + K$. ■

Powyższy algorytm jest oczywisty. Można jednak istotnie poprawić czas jego działania, bardzo nieznacznie zmieniając strategię. Ta poprawka jest dosyć ważna, a jej samodzielne odkrycie sprawi Czytelnikowi dużą przyjemność (zobacz ćwiczenie 6).

Z algorytmu A można korzystać dla małych i dużych wartości N. Założymy jednak na chwilę, że interesują nas duże wartości N. Zauważmy, co się dzieje, gdy $\text{SIZE}(P)$ równa się $N + 1$: w kroku A4 zmniejszamy $\text{SIZE}(P)$ do 1. Innymi słowy, tworzymy wolny blok o rozmiarze 1. Z uwagi na mały rozmiar blok ten jest w zasadzie bezużyteczny i tylko zawadza. Lepiej już zarezerwować cały blok o rozmiarze $N + 1$, zamiast „zaoszczędzić” jedno słowo. Zazwyczaj lepiej odjąłować parę słów pamięci, by nie troszczyć się o niepotrzebne szczegóły. Podobne uwagi mają sens w przypadku bloków o rozmiarze $N + K$ słów, gdy K jest bardzo małe.

Jeśli dopuścimy możliwość rezerwowania obszaru nieco większego niż N słów, to będziemy musieli w jakiś sposób zapamiętywać, ile słów zarezerwowano, by później zwolnić wszystkie $N+K$ słów. Ta odrobina księgowości służy tylko temu, by oznaczyć że zamrażamy trochę miejsca w *każdym* bloku, żeby zapewnić większą wydajność systemu tylko w niektórych sytuacjach (tj. gdy znajdziemy ciasne dopasowanie). Taka strategia nie wydaje się szczególnie atrakcyjna. Okazuje się jednak, że z innych powodów wygodnie jest przeznaczyć pierwsze słowo każdego bloku na tzw. *słowo kontrolne*, można więc śmiało zakładać, że w pierwszym słowie bloku (zarówno zajętego, jak i wolnego) znajduje się pole **SIZE** określające jego rozmiar.

Zgodnie z powyższymi konwencjami zmodyfikujemy krok A4:

- A4'.** [Zarezerwowanie $\geq N$] Przyjmij $K \leftarrow \text{SIZE}(P) - N$. Jeśli $K < c$ (gdzie c jest małą stałą całkowitą mówiącą, ile pamięci jesteśmy gotowi poświęcić, by zaoszczędzić na czasie), to przyjmij $\text{LINK}(Q) \leftarrow \text{LINK}(P)$, $L \leftarrow P$. W przeciwnym razie przyjmij $\text{SIZE}(P) \leftarrow K$, $L \leftarrow P + K$, $\text{SIZE}(L) \leftarrow N$. Zakończ (pomyślnie) wykonanie algorytmu, zarezerwowawszy blok długości przynajmniej N, rozpoczynający się w lokacji L.

Zazwyczaj poleca się wartość c wynoszącą około 8 lub 10, ale przemawia za tym bardzo mało dowodów zarówno teoretycznych, jak i empirycznych. Dla metody najlepszego dopasowania sprawdzenie warunku $K < c$ jest nawet *ważniejsze* niż dla metody pierwszego dopasowania, ponieważ są większe szanse, że wystąpią ciasne dopasowania (mniejsze wartości K), a z uwagi na charakterystykę algorytmu przydziału pamięci duża liczba wolnych bloków jest bardzo niepożądana.

B. Zwalnianie. Zastanówmy się teraz nad problemem przeciwnym: W jaki sposób zwracać bloki na listę wolnej pamięci, gdy program już ich nie potrzebuje?

Mozna pominąć ten problem, uciekając się do odśmiecania (punkt 2.3.5). Rozwiążanie takie polegałoby na tym, by nie robić nic, dopóki jest wolne miejsce, a gdy go zabraknie wyznaczyć wszystkie używane obszary i utworzyć nową listę **AVAIL**.

Nie poleca się jednak bezkrytycznego stosowania odśmiecania we wszystkich przypadkach. Po pierwsze, odśmiecanie wymaga dyscypliny przy posługiwaniu się wskaźnikami, jeśli chcemy bezpiecznie wyznaczyć wszystkie wykorzystywane obszary pamięci. Po drugie, jak zdążyliśmy się przekonać, odśmiecanie jest powolne przy dużej zajętości pamięci.

Istnieje jeszcze ważniejszy powód, dla którego odśmiecanie nie spełnia naszych oczekiwaniań. Ze zjawiskiem tym nie spotkaliśmy się w dotychczasowych rozważaniach. Przypuśćmy, że mamy dwa przylegające obszary pamięci. Oba są wolne, ale ponieważ przyjęliśmy filozofię odśmiecania, jeden z nich (na rysunku szary) nie figuruje na liście **AVAIL**.



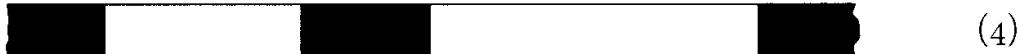
(2)

Na rysunku czarny kolor oznacza, że obszar jest zajęty. Możemy zarezerwować fragment obszaru, o którym wiemy, że jest wolny:



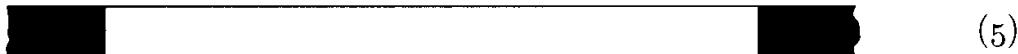
(3)

Jeśli w tym momencie nastąpi odśmiecanie, otrzymamy dwa oddzielne obszary wolne



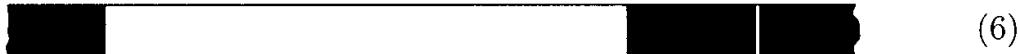
(4)

Granice między obszarami wolnymi i zajętymi mają tendencję do utrwalania się, a z upływem czasu sytuacja staje się coraz gorsza. Ale jeśli trzymalibyśmy się zasad nakazującej zwracać bloki na listę **AVAIL**, skoro tylko program je zwolni i jednocześnie scalać przylegające wolne bloki, to zamiast (2) mielibyśmy



(5)

Po przydziale bloku pamięć wyglądałaby tak



(6)

co jest sytuacją dużo lepszą niż (4). Odśmiecanie sprawia, że pamięć jest bardziej „posiekana”, niż to konieczne.

By pozbyć się tego problemu, możemy korzystać z odśmiecania połączonego z procesem *kompresji pamięci*, tj. z przemieszczaniem wszystkich zajętych bloków w ten sposób, by zajmowały kolejne lokacje, dzięki czemu po odśmiecieniu wolne bloki tworzą jeden spójny obszar. Algorytm przydziału pamięci staje się przy tych założeniach trywialny, ponieważ cały czas w systemie jest tylko jeden wolny blok. Chociaż w tej metodzie do skopiowania wszystkich zajętych lokacji i zmiany wartości wskaźników jest wymagany pewien czas, można z niej korzystać, osiągając rozsądную wydajność, pod warunkiem że utrzymujemy dyscyplinę w posługiwaniu się wskaźnikami i że w każdym bloku jest wolne pole, którego może użyć algorytm odśmiecania (zobacz ćwiczenie 33).

Ponieważ wiele programów nie spełnia wymagań nakładanych przez odśmiecanie, zajmiemy się teraz metodami jawnego zwracania bloków na listę wolnej pamięci. Jedyna trudność tej metody wiąże się ze scalaniem bloków: dwa przylegające wolne bloki powinny zostać scalone w jeden. Tak na prawdę, gdy wolny staje się obszar ograniczony przez dwa wolne bloki, wtedy wszystkie trzy bloki należy połączyć w jeden. Dzięki temu jest możliwe dobre wykorzystanie pamięci, nawet jeśli bloki pamięci są wielokrotnie przydzielane i zwalniane. (W celu udowodnienia tego faktu zapoznaj się z „regułą pięćdziesięciu procent” przedstawioną poniżej).

Problem polega na stwierdzeniu, czy obszary po obu stronach zwalnianego bloku są wolne. Jeśli tak, to powinniśmy odpowiednio zmodyfikować listę **AVAIL**, a to nie jest takie proste.

Pierwsze rozwiążanie polega na przechowywaniu bloków na liście **AVAIL** w porządku rosnących adresów.

Algorytm B (*Zwalnianie z użyciem listy posortowanej*). Przy założeniach takich jak w algorytmie A oraz dodatkowo, że lista **AVAIL** jest posortowana względem adresów bloków (tj., jeśli P wskazuje na wolny blok i $\text{LINK}(P) \neq \Lambda$, to $\text{LINK}(P) > P$), algorytm dodaje do listy **AVAIL** blok N kolejnych komórek pamięci, zaczynający się w lokacji P_0 . Zakładamy, że żadna z tych N komórek nie jest aktualnie oznaczona jako wolna.

- B1.** [Inicjowanie] Przyjmij $Q \leftarrow \text{LOC}(\text{AVAIL})$. (Zobacz uwagi przy kroku A1).
- B2.** [Zwiększenie P] Przyjmij $P \leftarrow \text{LINK}(Q)$. Jeśli $P = \Lambda$ lub jeśli $P > P_0$, to przejdź do B3; w przeciwnym razie przyjmij $Q \leftarrow P$ i powtórz krok B2.
- B3.** [Sprawdzenie górnej granicy] Jeśli $P_0 + N = P$ i $P \neq \Lambda$, to przyjmij wartość $N \leftarrow N + \text{SIZE}(P)$, $\text{LINK}(P_0) \leftarrow \text{LINK}(P)$. W przeciwnym razie przyjmij $\text{LINK}(P_0) \leftarrow P$.
- B4.** [Sprawdzenie dolnej granicy] Jeśli $Q + \text{SIZE}(Q) = P_0$ (zakładamy, że

$$\text{SIZE}(\text{LOC}(\text{AVAIL})) = 0,$$

zatem powyższy warunek jest fałszywy, gdy $Q = \text{LOC}(\text{AVAIL})$), to przyjmij $\text{SIZE}(Q) \leftarrow \text{SIZE}(Q) + N$, $\text{LINK}(Q) \leftarrow \text{LINK}(P_0)$. W przeciwnym razie $\text{LINK}(Q) \leftarrow P_0$, $\text{SIZE}(P_0) \leftarrow N$. ■

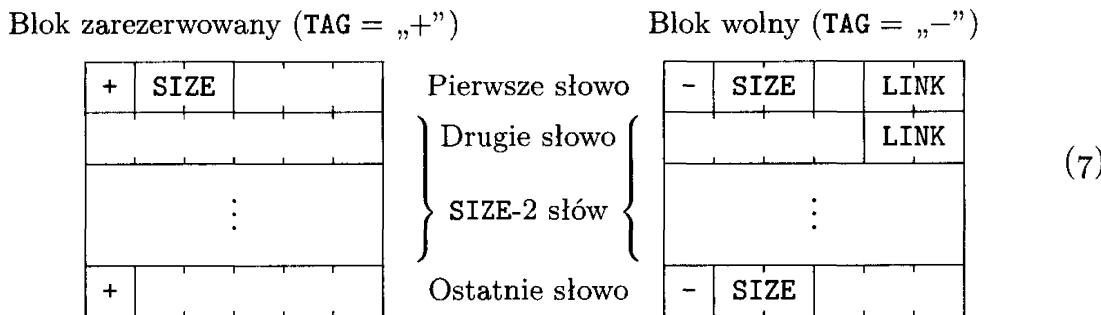
W krokach B3 i B4 jest wykonywane właściwe scalanie przy uwzględnieniu założenia, że wartości wskaźników $Q < P_0 < P$ są początkowymi lokacjami trzech kolejnych wolnych bloków.

Jeżeli nie utrzymujemy bloków na liście **AVAIL** w porządku ich adresów, to łatwo zauważyc, że „siłowe” podejście do problemu scalania bloków będzie wymagało przeszukania całej listy **AVAIL**. Algorytm B musi przejrzeć średnio *połowę* węzłów listy **AVAIL** (w kroku B2). W ćwiczeniu 11 pokazujemy, w jaki sposób zmodyfikować algorytm B, by średnio musiał przeglądać jedynie około jednej trzeciej węzłów listy **AVAIL**. Oczywiście, gdy lista **AVAIL** jest dłuża, żadna z tych metod nie spełnia naszych oczekiwani. Czy nie ma jakiegoś sposobu rezerwowania i zwalniania pamięci, który nie wymagałby pracochłonnego przeszukiwania listy **AVAIL**?

Zajmiemy się teraz metodą, która przy zwalnianiu pamięci pozwala obejść się bez przeszukiwania, a można ją zmodyfikować w ten sposób (ćwiczenie 6), by prawie wcale nie wymagała przeszukiwania przy rezerwowaniu pamięci. W metodzie tej korzysta się z pola **TAG** na obu końcach bloku oraz pola **SIZE** w pierwszym słowie każdego bloku. Taki narzut jest bardzo znikomy, gdy posługujemy się rozsądnie dużymi blokami, choć zapewne jest ceną zbyt wysoką, gdy średni rozmiar bloku jest bardzo mały. Inna metoda, opisana w ćwiczeniu 19, wymaga

tylko jednego bitu w pierwszym słowie każdego bloku, kosztem nieco większego czasu wykonania i skomplikowania programu.

Załóżmy jednak, że nie szkoda nam poświęcić odrobiny miejsca na informację kontrolną, jeżeli mamy w zamian uzyskać znaczne przyspieszenie algorytmu względem algorytmu B dla przypadków, gdy lista **AVAIL** jest długa. W opisywanej metodzie zakładamy, że każdy blok ma następującą postać:

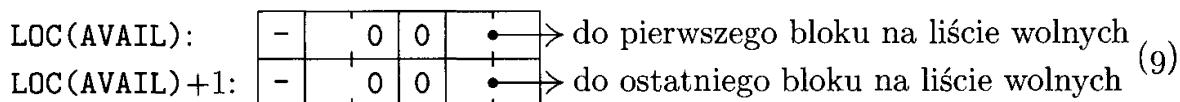


Pomysł polega na utrzymywaniu listy dwukierunkowej **AVAIL**, tak by można było łatwo dodawać i usuwać elementy z dowolnego miejsca listy. Pole **TAG** na końcach bloku można wykorzystać do sterowania procesem scalania, ponieważ dzięki niemu szybko jesteśmy w stanie stwierdzić, czy dwa przyległe bloki są wolne.

Dowiązania dwukierunkowe realizujemy podobnie: **LINK** w pierwszym słowie wskazuje na następny wolny blok na liście, a **LINK** w drugim słowie wskazuje na poprzedni blok. Jeśli zatem **P** jest adresem wolnego bloku, to zawsze zachodzi warunek

$$\text{LINK}(\text{LINK}(P) + 1) = P = \text{LINK}(\text{LINK}(P + 1)). \quad (8)$$

By warunek był spełniony w „przypadkach brzegowych”, korzystamy z atrapy postaci:



Algorytm „pierwszego dopasowania”, w którym stosuje się tę metodę, można zaprojektować bardzo podobnie do algorytmu A, więc nie będziemy się tym tutaj zajmować (zobacz ćwiczenie 12). Główna zaleta naszej metody polega na tym, że można zwolnić blok w czasie stałym.

Algorytm C (Zwalnianie ze znacznikami brzegowymi). Zakładamy, że bloki są postaci (7) oraz że **AVAIL** jest listą dwukierunkową. Algorytm wstawia na listę **AVAIL** blok zaczynający się od adresu **P0**. Jeśli sterta obejmuje adresy od m_0 do m_1 włącznie, to zakładamy dla wygody, że

$$\text{TAG}(m_0 - 1) = \text{TAG}(m_1 + 1) = „+”.$$

- C1.** [Sprawdzenie dolnej granicy] Jeśli $\text{TAG}(P0 - 1) = „+”$, to idź do C3.
- C2.** [Usuwanie dolnego bloku] Przyjmij $P \leftarrow P0 - \text{SIZE}(P0 - 1)$, $P1 \leftarrow \text{LINK}(P)$, $P2 \leftarrow \text{LINK}(P + 1)$, $\text{LINK}(P1 + 1) \leftarrow P2$, $\text{LINK}(P2) \leftarrow P1$, $\text{SIZE}(P) \leftarrow \text{SIZE}(P) + \text{SIZE}(P0)$, $P0 \leftarrow P$.

- C3.** [Sprawdzenie górnej granicy] Przyjmij $P \leftarrow P_0 + \text{SIZE}(P_0)$. Jeśli wartość $\text{TAG}(P) = „+”$, to idź do C5.
- C4.** [Usuwanie górnego bloku] Przyjmij $P_1 \leftarrow \text{LINK}(P)$, $P_2 \leftarrow \text{LINK}(P+1)$, $\text{LINK}(P_1+1) \leftarrow P_2$, $\text{LINK}(P_2) \leftarrow P_1$, $\text{SIZE}(P_0) \leftarrow \text{SIZE}(P_0) + \text{SIZE}(P)$, $P \leftarrow P + \text{SIZE}(P)$.
- C5.** [Wstawianie na listę AVAIL] Przyjmij $\text{SIZE}(P-1) \leftarrow \text{SIZE}(P_0)$, $\text{LINK}(P_0) \leftarrow \text{AVAIL}$, $\text{LINK}(P_0+1) \leftarrow \text{LOC}(\text{AVAIL})$, $\text{LINK}(\text{AVAIL}+1) \leftarrow P_0$, $\text{AVAIL} \leftarrow P_0$, $\text{TAG}(P_0) \leftarrow \text{TAG}(P-1) \leftarrow „-”$. ■

Kroki algorytmu C wynikają w prosty sposób z układu pamięci (7). Nieco dłuższy, ale szybszy algorytm podajemy w ćwiczeniu 15. W kroku C5 AVAIL oznacza $\text{LINK}(\text{LOC}(\text{AVAIL}))$, jak pokazano w (9).

C. System bloków bliźniaczych. Zajmiemy się teraz innym podejściem do dynamicznego przydziału pamięci, sprawdzającego się w komputerach binarnych. W tej metodzie korzysta się z jednego dodatkowego bitu w każdym bloku oraz wymaga, by bloki były długości 1, 2, 4, 8, 16 itd. Rozmiary przydzielanych bloków zaokrąglają się w górę do najbliższej potęgi dwójki.

Pomysł polega na utrzymywaniu oddzielnej listy wolnych bloków dla każdego rozmiaru 2^k , $0 \leq k \leq m$. Cała sterta składa się z 2^m słów i możemy założyć, że składają się na nią lokacje od 0 do $2^m - 1$. Początkowo wolny jest jeden blok o rozmiarze 2^m . Gdy pojawia się żądanie przydziału bloku o rozmiarze 2^k , a nie ma akurat żadnego wolnego bloku o tym rozmiarze, dzielimy większy wolny blok na dwie równe części. Po (być może wielu) podziałach otrzymamy wolny blok rozmiaru 2^k . Bloki powstałe z podziału jednego bloku na pół nazywamy *blokami bliźniaczymi*. Gdy dwa wolne bloki bliźniacze w przyszłości się spotkają, zostaną scalone w jeden większy blok. W ten sposób przydział i zwalnianie można kontynuować w nieskończoność, chyba że w pewnym momencie zabraknie wolnej pamięci.

Kluczowym faktem leżącym u podstaw praktycznej przydatności tej metody jest to, że znając adres bloku (tj. lokację jego pierwszego słowa) oraz rozmiar bloku, znamy także adres jego bloku bliźniaczego. Na przykład bliźniak bloku o rozmiarze 16 zaczynającego się w lokacji 101110010110000 jest blokiem zaczynającym się w lokacji 101110010100000. By przekonać się, dlaczego zawsze tak musi być, zauważmy, że *adres bloku o rozmiarze 2^k jest wielokrotnością 2^k* . Innymi słowy, w zapisie dwójkowym taki adres ma przynajmniej k zer z prawej strony. Tę obserwację łatwo uzasadnić przez indukcję: jeśli jest prawdziwa dla wszystkich bloków rozmiaru 2^{k+1} , to na pewno jest prawdziwa dla bloków o rozmiarze o połowę mniejszym.

Blok o rozmiarze, powiedzmy, 32 ma zatem adres postaci $xx\dots x00000$ (gdzie x reprezentuje 0 lub 1). Jeśli go podzielimy, to otrzymane bloki bliźniacze będą miały adresy $xx\dots x00000$ i $xx\dots x10000$. W ogólności, niech $\text{buddy}_k(x) = \text{adres bliźniaka bloku o rozmiarze } 2^k \text{ i adresie } x$; okazuje się, że

$$\text{buddy}_k(x) = \begin{cases} x + 2^k, & \text{gdy } x \bmod 2^{k+1} = 0; \\ x - 2^k, & \text{gdy } x \bmod 2^{k+1} = 2^k. \end{cases} \quad (10)$$

Tę funkcję łatwo obliczyć za pomocą operacji „bitowej alternatywy wykluczającej” (nazywanej czasami „dodawaniem bez przeniesienia” lub XOR) występującej zazwyczaj w repertuarze operacji komputerów dwójkowych; (zobacz ćwiczenie 28).

W systemie bloków bliźniaczych korzysta się z jednobitowego znacznika TAG w każdym bloku:

$$\begin{aligned} \text{TAG}(P) = 0, & \quad \text{jeśli blok o adresie } P \text{ jest zarezerwowany;} \\ \text{TAG}(P) = 1, & \quad \text{jeśli blok o adresie } P \text{ jest wolny.} \end{aligned} \quad (11)$$

Poza znacznikiem TAG, występującym we wszystkich blokach, bloki *wolne* mają dwa dodatkowe pola wskaźnikowe LINKF i LINKB, które są zwyczajnymi dowiązaniami w przód i w tył na liście dwukierunkowej. Bloki wolne mają ponadto pole KVAL zawierające wartość k dla bloku o rozmiarze 2^k . W poniższym algorytmie korzysta się z tablicy lokacji AVAIL[0], AVAIL[1], ..., AVAIL[m], w której są przechowywane atrapy listy wolnych bloków o rozmiarach 1, 2, 4, ..., 2^m . Listy są dwukierunkowe, zatem atrapy zawierają po dwa wskaźniki (zobacz punkt 2.2.5):

$$\begin{aligned} \text{AVAILF}[k] = \text{LINKF}(\text{LOC}(\text{AVAIL}[k])) &= \text{wskaźnik na koniec listy } \text{AVAIL}[k]; \\ \text{AVAILB}[k] = \text{LINKB}(\text{LOC}(\text{AVAIL}[k])) &= \text{wskaźnik na początek listy } \text{AVAIL}[k]. \end{aligned} \quad (12)$$

Początkowo, gdy nie przydzielono żadnego bloku, mamy

$$\begin{aligned} \text{AVAILF}[m] &= \text{AVAILB}[m] = 0, \\ \text{LINKF}(0) &= \text{LINKB}(0) = \text{LOC}(\text{AVAIL}[m]), \\ \text{TAG}(0) &= 1, \quad \text{KVAL}(0) = m \end{aligned} \quad (13)$$

(jeden blok rozmiaru 2^m zaczynający się od adresu 0) oraz

$$\text{AVAILF}[k] = \text{AVAILB}[k] = \text{LOC}(\text{AVAIL}[k]) \quad \text{dla } 0 \leq k < m \quad (14)$$

(pusta lista dla rozmiaru 2^k dla $k < m$).

Na podstawie powyższego opisu można spróbować samemu zaprojektować algorytmy przydziału i zwalniania pamięci, zanim się je przeczyta. Zauważmy, jak łatwo dzielić bloki w algorytmie przydziału.

Algorytm R (*Przydzielanie w systemie bloków bliźniaczych*). Algorytm znajduje i przydziela w systemie bloków bliźniaczych blok o rozmiarze 2^k lub zgłasza błąd.

R1. [Znajdowanie bloku] Niech j będzie najmniejszą liczbą całkowitą z przedziału $k \leq j \leq m$, dla której $\text{AVAILF}[j] \neq \text{LOC}(\text{AVAIL}[j])$, tj. dla której lista wolnych bloków o rozmiarze 2^j nie jest pusta. Jeżeli takie j nie istnieje, to zakończ działanie algorytmu, sygnalizując błąd – nie ma wolnego bloku pamięci, który zaspokoiłby żądanie.

R2. [Usuwanie z listy] Przyjmij $L \leftarrow \text{AVAILF}[j]$, $P \leftarrow \text{LINKF}(L)$, $\text{AVAILF}[j] \leftarrow P$, $\text{LINKB}(P) \leftarrow \text{LOC}(\text{AVAIL}[j])$, $\text{TAG}(L) \leftarrow 0$.

R3. [Czy trzeba podzielić?] Jeśli $j = k$, to zakończ wykonanie algorytmu (zauważono i przydzielono blok rozpoczynający się w lokacji L).

R4. [Dzielenie] Zmniejsz j o 1. Następnie wykonaj $P \leftarrow L + 2^j$, $\text{TAG}(P) \leftarrow 1$, $\text{KVAL}(P) \leftarrow j$, $\text{LINKF}(P) \leftarrow \text{LINKB}(P) \leftarrow \text{LOC}(\text{AVAIL}[j])$, $\text{AVAILF}[j] \leftarrow \text{AVAILB}[j] \leftarrow P$. (Dzielimy blok i wstawiamy nie wykorzystaną połówkę na pustą listę $\text{AVAIL}[j]$). Wróć do kroku R3. ■

Algorytm S (*Zwalnianie w systemie bloków bliźniaczych*). Algorytm zwalnia w systemie bloków bliźniaczych blok o rozmiarze 2^k zaczynający się w lokacji L.

S1. [Czy blok bliźniaczy jest wolny?] Przyjmij wartość $P \leftarrow \text{buddy}_k(L)$. (Zobacz wzór (10)). Jeśli $k = m$ lub $\text{TAG}(P) = 0$ lub jeśli $\text{TAG}(P) = 1$ i $\text{KVAL}(P) \neq k$, to idź do S3.

S2. [Scal z blokiem bliźniaczym] Przyjmij

$$\text{LINKF}(\text{LINKB}(P)) \leftarrow \text{LINKF}(P), \quad \text{LINKB}(\text{LINKF}(P)) \leftarrow \text{LINKB}(P).$$

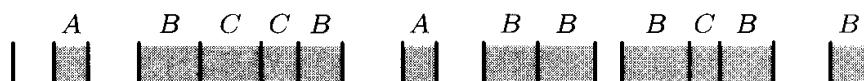
(Usuwamy blok P z listy $\text{AVAIL}[k]$). Następnie przyjmij $k \leftarrow k + 1$, jeśli $P < L$, to przyjmij $L \leftarrow P$. Wróć do S1.

S3. [Wstawianie na listę] Przyjmij $\text{TAG}(L) \leftarrow 1$, $P \leftarrow \text{AVAILF}[k]$, $\text{LINKF}(L) \leftarrow P$, $\text{LINKB}(P) \leftarrow L$, $\text{KVAL}(L) \leftarrow k$, $\text{LINKB}(L) \leftarrow \text{LOC}(\text{AVAIL}[k])$, $\text{AVAILF}[k] \leftarrow L$. (Wstawiamy blok L na listę $\text{AVAIL}[k]$). ■

D. Porównanie metod. Matematyczna analiza dynamicznego przydziału pamięci okazuje się być bardzo skomplikowana, ale istnieje jedno ciekawe zjawisko, które daje się się przeanalizować – „reguła pięćdziesięciu procent”:

Jeśli program działa w ten sposób, że kolejno uruchamiając algorytmy A i B, dąży się do stanu równowagi, w którym średnio przydzielonych jest N bloków i prawdopodobieństwo zwolnienia jest dla każdego z nich takie samo, a wielkość K w algorytmie A przybiera wartości niezerowe (lub bardziej ogólnie, wartości $\geq c$ jak w kroku A4') z prawdopodobieństwem p , to średnia liczba wolnych bloków dąży w przybliżeniu do $\frac{1}{2}pN$.

Ta reguła mówi, jaka będzie przybliżona długość listy AVAIL. Gdy wielkość p jest bliska 1 – co ma miejsce, gdy c jest bardzo małe i gdy rozmiary bloków są zróżnicowane – mamy mniej więcej dwa razy tyle bloków zajętych co wolnych. Stąd nazwa „reguła pięćdziesięciu procent”. Nietrudno wyprowadzić tę regułę. Przyjrzyjmy się następującej mapie pamięci:



Podzieliliśmy zajęte bloki na trzy kategorie:

A: po zwolnieniu liczba wolnych bloków zmniejszy się o jeden;

B: po zwolnieniu liczba wolnych bloków się nie zmieni;

C: po zwolnieniu liczba wolnych bloków zwiększy się o jeden.

Niech N będzie liczbą zajętych bloków, a M liczbą bloków wolnych. Niech A , B i C będą liczbami bloków typu odpowiednio A , B i C . Mamy

$$\begin{aligned} N &= A + B + C \\ M &= \frac{1}{2}(2A + B + \epsilon) \end{aligned} \quad (15)$$

gdzie $\epsilon = 0, 1$ lub 2 , w zależności od sytuacji na górnej i dolnej granicy.

Założymy, że N jest stałe, ale A , B , C i ϵ są wartościami losowymi, które osiągnęły rozkład stacjonarny po zwolnieniu bloku i (nieco inny) rozkład stacjonarny po przydzieleniu bloku. Średnia zmiana wartości M , gdy blok jest zwalniany, jest średnią wartością $(C - A)/N$; średnia wartość M , gdy blok jest przydzielany, równa się $1 - p$. Założenie o równowadze mówi, że średnia wartość $C - A - N + pN$ równa się zero. A wtedy średnia wartość $2M$ wynosi pN plus średnia wartość ϵ , ponieważ $2M = N + A - C + \epsilon$ z (15). Stąd wynika reguła pięćdziesięciu procent.

Nasze złożenie, że zwalniany jest losowo wybrany spośród zajętych bloków, będzie prawdziwe, o ile czas życia bloku będzie zmienną losową o rozkładzie wykładniczym. Z drugiej strony, jeśli wszystkie bloki mają mniej więcej ten sam czas życia, to założenie nie jest prawdziwe. John E. Shore zauważał, że bloki typu A są zazwyczaj „starsze” niż bloki typu C, gdy operacje przydzielania i zwalniania mają charakterystykę typu „pierwszy-wchodzi-pierwszy-wychodzi”, ponieważ bloki w spójnych ciągach mają tendencję do występowania w kolejności od najmłodszego do najstarszego, a także dlatego, że bardzo rzadko najpóźniej przydzielony blok jest typu A. Wszystko to wpływa na zmniejszenie liczby wolnych bloków, co skutkuje większą wydajnością, niż wynikłoby to z reguły pięćdziesięciu procent. [Zobacz CACM 20 (1977), 812–820].

Więcej informacji na temat reguły pięćdziesięciu procent można znaleźć w: D. J. M. Davies, BIT 20 (1980), 279–288; C. M. Reeves, Comp. J. 26 (1983), 25–35; G. Ch. Pflug, Comp. J. 27 (1984), 328–333.

Poza tą ciekawą regułą cała nasza wiedza na temat dynamicznego przydziału pamięci opiera się w całości na doświadczeniach przeprowadzanych metodą Monte Carlo. Wybierając algorytm przydziału pamięci dla konkretnego programu lub klasy programów, zechce Czytelnik we własnym zakresie przeprowadzić małą symulację. Autor wykonał kilka takich eksperymentów przed napisaniem tego podrozdziału (rzeczywiście dawało się zaobserwować regułę pięćdziesięciu procent, zanim jeszcze udało się znaleźć jej dowód). Przyjrzyjmy się pokrótkę metodom i wynikom tych eksperymentów.

Prosty symulator działa następująco (początkowo TIME ma wartość 0, a cała pamięć jest wolna):

- P1.** Zwiększ TIME o 1.
- P2.** Zwolnij wszystkie bloki, które miały być zwolnione przy bieżącej wartości zmiennej TIME.
- P3.** Wyznacz wartości S (losowy rozmiar) i T (losowy czas życia), opierając się na wybranych rozkładach prawdopodobieństwa i posługując się metodami z rozdziału 3.

P4. Przydziel nowy blok o długości S , który ma zostać zwolniony, gdy **TIME** osiągnie wartość (**TIME** + T). Wróć do P1. ■

Za każdym razem, gdy zmienna **TIME** przyjmowała wartość podzielną przez 200, była drukowana szczegółowa statystyka dotycząca wydajności algorytmów przydziału i zwalniania pamięci. Dla każdej pary testowanych algorytmów posługiwano się tym samym ciągiem wartości S i T . Mniej więcej wtedy, gdy zmienna **TIME** przekraczała wartość 2000, system zazwyczaj osiągał stan stabilny i wszysko wskaazywało na to, że mniej więcej taki stan utrzymywać się będzie w nieskończoność. Dla pewnych rozmiarów sterty oraz pewnych rozkładów S i T (krok P3), algorytmowi przydziału nie udawało się czasem znaleźć wystarczająco dużego bloku, zatem eksperyment był przerywany.

Niech C będzie całkowitą liczbą wolnych lokacji i niech \bar{S} i \bar{T} oznaczają średnie wartości S i T w kroku P3. Łatwo zauważać, że spodziewana liczba zajętych lokacji w dowolnym momencie to $\bar{S}\bar{T}$, jeśli wartość **TIME** jest odpowiednio duża. Gdy $\bar{S}\bar{T}$ przekraczało około $\frac{2}{3}C$, zazwyczaj pojawiał się błąd przekroczenia pamięci, częstokroć zanim rzeczywiście potrzebnych było C lokacji. Dla bloków o małym rozmiarze pamięć potrafiła zapełnić się w 90 procentach, ale gdy dopuściło się występowanie bloków o rozmiarach przekraczających $\frac{1}{3}C$ (nie zabraniając przy tym pojawiania się bloków dużo mniejszych) program stwierdzał, że pamięć jest „pełna” w sytuacjach, gdy faktycznie wykorzystanych było mniej niż $\frac{1}{2}C$ lokacji. Dowody empiryczne wskazują na to, że jeśli zależy nam na wydajności, to *nie należy korzystać z dynamicznego przydziału pamięci, gdy rozmiary bloków przekraczają $\frac{1}{10}C$.*

Przyczyny takiego stanu rzeczy można wytlumaczyć za pomocą reguły pięćdziesięciu procent: jeśli system osiąga stan równowagi, w którym rozmiar f średniego wolnego bloku jest mniejszy niż rozmiar r średniego zajętego bloku, to możemy się spodziewać żądania, które nie da się zrealizować, jeśli nie będziemy dysponować większym wolnym blokiem. W naszym systemie, który nie ma tendencji do przepełniania się, mamy $f \geq r$, stąd $C = fM + rN \geq rM + rN \approx (\frac{1}{2}p + 1)rN$. Całkowity rozmiar wykorzystywanej pamięci wynosi zatem $rN \leq C/(\frac{1}{2}p + 1)$. Gdy $p \approx 1$, nie możemy korzystać z pamięci większej niż około $\frac{2}{3}$ wszystkich lokacji.

Eksperymenty przeprowadzono dla trzech rozkładów rozmiaru bloku (S):

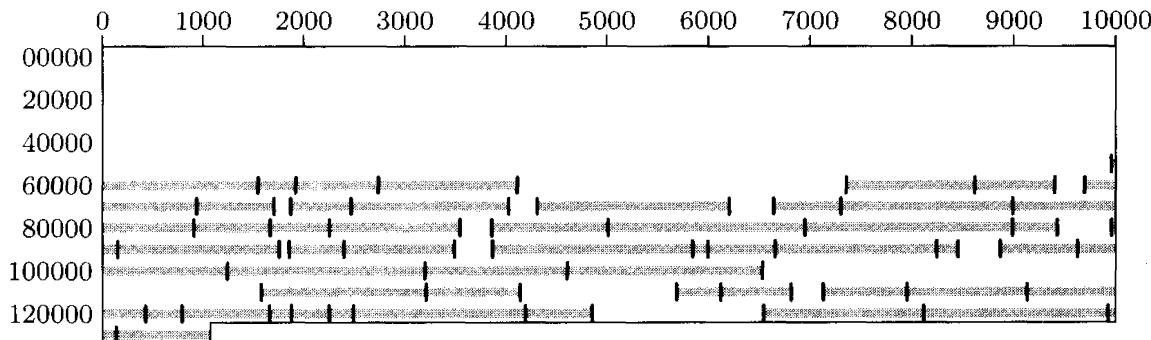
(*S1*) liczba całkowita wybierana z równym prawdopodobieństwem z przedziału między 100 a 2000;

(*S2*) rozmiary (1, 2, 4, 8, 16, 32) wybierane odpowiednio z prawdopodobieństwem $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32})$;

(*S3*) rozmiary (10, 12, 14, 16, 18, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 500, 1000, 2000, 3000, 4000) wybierane z równym prawdopodobieństwem.

Czas życia (T) był zazwyczaj liczbą wybieraną z równym prawdopodobieństwem z zakresu od 1 do t , dla ustalonego $t = 10, 100$ lub 1000.

Przeprowadzono także eksperymenty, w których wartość T była wybierana w kroku P3 z równym prawdopodobieństwem z przedziału liczb od 1 do



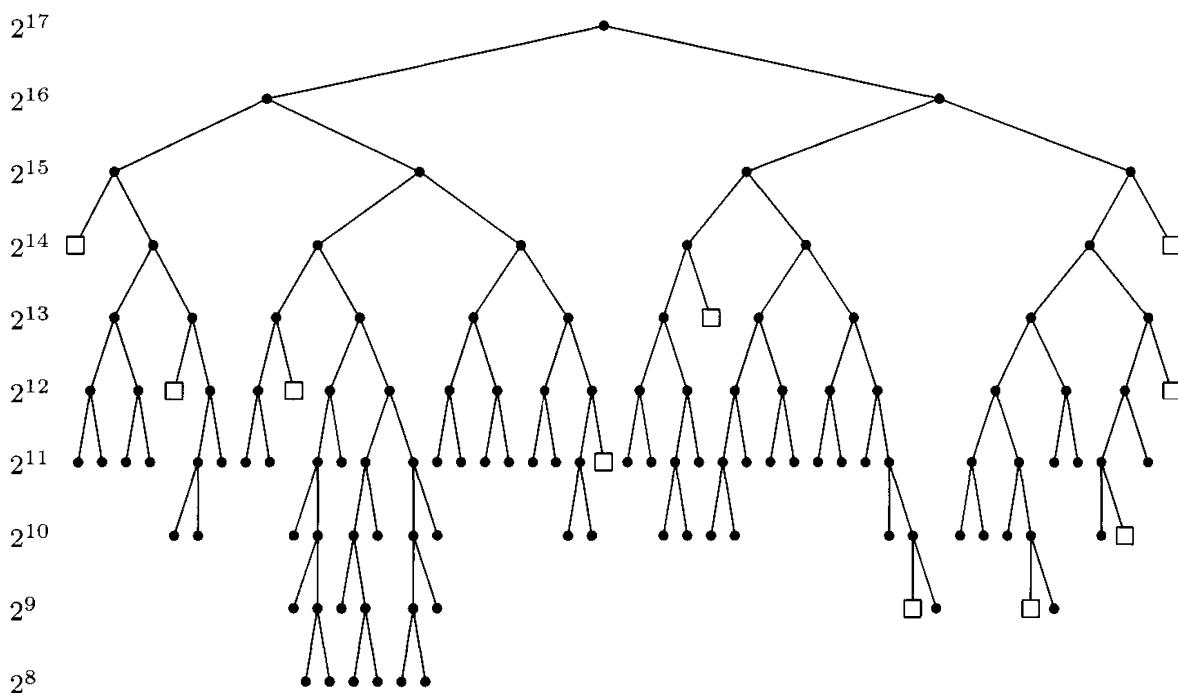
Rys. 43. Mapa pamięci otrzymana przy wykorzystaniu metody najlepszego dopasowania. (Mapy pamięci dla tych samych ciągów żądań, ale innych algorytmów przydziału, pokazano na rysunkach 42 (metoda pierwszego dopasowania) oraz 44 (system bloków bliźniaczych)).

$\min(\lfloor \frac{5}{4}U \rfloor, 12500)$, gdzie U jest liczbą jednostek czasu do najbliższej zleconej operacji zwolnienia jakiegoś zajętego bloku. Ten rozkład ma symulować zachowanie „prawie-pierwszy-wchodzi-pierwszy-wychodzi” – jeśli zawsze wybieramy $T \leq U$, to system przydziału pamięci redukuje się do prostych operacji na stosie. (Zobacz ćwiczenie 1). Dla podanego rozkładu T jest większe od U w około 20 procentach przypadków, zatem mamy do czynienia prawie wyłącznie (ale *prawie*) z operacjami na stosie. Przy takim rozkładzie algorytmy A, B i C zachowywały się o wiele lepiej niż zwykle. Rzadko kiedy na liście AVAIL znajdowały się więcej niż dwa bloki, podczas gdy bloków zajętych było około czternastu. Z drugiej strony algorytmy R i S systemu bloków bliźniaczych były wolniejsze, ponieważ przy takich stosopodobnych operacjach podziały i scalania bloków występowały o wiele częściej. Okazuje się, że nie tak łatwo wyznaczyć teoretycznie własności rozkładu T (zobacz ćwiczenie 32).

Na rysunku 42 z początku tego rozdziału była pokazana konfiguracja pamięci w chwili $\text{TIME} = 5000$ dla rozmiaru bloku wybieranego wg rozkładu ($S1$) oraz czasu życia wybieranego spośród liczb od 1 do 100 z równym prawdopodobieństwem, dla metody pierwszego dopasowania (algorytmy A i B). W tym eksperymencie prawdopodobieństwo p z „reguły pięćdziesięciu procent” było w zasadzie równe 1, moglibyśmy się zatem spodziewać mniej więcej dwa razy tylu bloków zajętych co wolnych. W istocie na rysunku 42 jest 21 bloków wolnych i 53 bloki zajęte. Nie powoduje to obalenia reguły pięćdziesięciu procent: na przykład w chwili $\text{TIME} = 4600$ bloków wolnych było 25, a zajętych 49. Konfiguracja pokazana na rysunku 42 jest po prostu ilustracją tego, że reguła pięćdziesięciu procent podlega statystycznym fluktuacjom. Liczba wolnych bloków wahała się w przedziale 20–30, a liczba bloków zajętych w przedziale 45–55.

Na rysunku 43 jest pokazana konfiguracja pamięci otrzymana dla *tych samych danych*, ale przy wykorzystaniu metody najlepszego dopasowania. Przyjęto stałą c z kroku A4' równą 16, by uniknąć małych bloków, skutkiem czego prawdopodobieństwo p zmalało do około 0.7 i zmniejszyła się liczba wolnych bloków.

Gdy rozkład czasu życia zmieniono w ten sposób, że wartości były wybierane spośród liczb od 1 do 1000, otrzymano sytuacje analogiczne do tych z rysun-



Rys. 44. Mapa pamięci otrzymana przy wykorzystaniu systemu bloków bliźniaczych. (Drzewo pokazuje podziały większych bloków na pary bloków bliźniaczych. Kwadraciki oznaczają bloki wolne).

ków 42 i 43 – odpowiednie wartości były mniej więcej dziesięć razy większe. Na przykład bloków zajętych było 515; odpowiednik sytuacji z rysunku 42 zawierał 240 wolnych bloków, a odpowiednik sytuacji z rysunku 43 zawierał 176 wolnych bloków.

We wszystkich eksperymentach porównujących metodę najlepszego dopasowania i pierwszego dopasowania ta druga zawsze okazywała się lepsza. Dla przypadków, w których pamięć się przepełniała, metoda pierwszego dopasowania zazwyczaj „poddawała się” później.

Na tym samym ciągu danych przetestowano system bloków bliźniaczych. Na rysunku 44 widać wynik. Wszystkie bloki o rozmiarach od 257 do 512 były traktowane tak, jakby miały rozmiar 512, bloki o rozmiarach od 513 do 1024, tak jak bloki o rozmiarze 1024 itd. Oznacza to, że średnio przydzielano mniej więcej cztery trzecie rzeczywiście potrzebnej pamięci (zobacz ćwiczenie 21); oczywiście system bloków bliźniaczych działa lepiej dla rozkładu rozmiaru bloków, takiego jak (*S2*). Zauważmy, że na rysunku 44 są wolne bloki o rozmiarach 2^9 , 2^{10} , 2^{11} , 2^{12} , 2^{13} i 2^{14} .

Z symulacji systemu bloków bliźniaczych wynika, że sprawuje się on lepiej, niż można by się spodziewać. Jest jasne, że w systemie bloków bliźniaczych może zdażyć się sytuacja, gdy dwa przylegające wolne bloki nie są scalane (bo nie są bliźniakami). Na rysunku 44 nie ma jednak takiego przypadku, a w praktyce zdarza się on rzadko. W przypadkach przepełnienia pamięci zajętość wynosiła około 95 procent, a to jest zadziwiająco dobry wynik. Ponadto bardzo rzadko zachodziła konieczność podziału bloków w algorytmie R albo ich scalania w algorytmie S. Drzewo bloków bliźniaczych przez cały czas symulacji wyglądało

mniej więcej tak, jak na rysunku 44, z wolnymi blokami o najczęściej używanych rozmiarach. Pewne wyniki matematyczne dotyczące takiego właśnie zachowania na najniższych poziomach opisanego drzewa można znaleźć w: P. W. Purdom, Jr., S. M. Stigler, *JACM* 17 (1970), 683–697.

Kolejną niespodzianką było świetne zachowanie algorytmu A po modyfikacji opisanej w ćwiczeniu 6. Średnio potrzebnych było tylko 2.8 sprawdzeń rozmiarów wolnych bloków (przy rozkładzie rozmiaru (S_1) i czasach życia wybieranych z jednakowym prawdopodobieństwem spośród liczb z zakresu od 1 do 1000), a w ponad połowie przypadków wystarczała wartość minimalna (jedna iteracja). Było to prawdą wbrew temu, że wolnych bloków było około 250. Ten sam eksperyment z niezmodyfikowanym algorytmem A pokazał, że średnio potrzebnych było około 125 iteracji (zatem za każdym razem przeglądano około połowę węzłów listy **AVAIL**); w 20 procentach przypadków potrzebnych było ponad 200 iteracji.

Takie zachowanie niezmodyfikowanego algorytmu A można w istocie przewidzieć jako konsekwencję reguły pięćdziesięciu procent. W stanie równowagi obszar pamięci zawierający starszą połowę zajętych bloków będzie także zawierał młodszą połowę zwolnionych bloków. Ten obszar będzie brał udział w połowie wszystkich zwolnień, więc w stanie równowagi również w połowie wszystkich przydziałów. Rozumowanie jest poprawne również wtedy, gdy „pół” zastąpimy dowolnym ułamkiem. (Te obserwacje pochodzą od J. M. Robsona).

Ćwiczenia na końcu tego punktu zawierają programy na maszynę **MIX** realizujące dwie podstawowe zalecane przez nas metody: (i) system znaczników brzegowych, zgodnie z modyfikacjami z ćwiczeń 12 i 16, oraz (ii) system bloków bliźniaczych. Oto przybliżone wyniki

	Czas przydziału	Czas zwalniania
System znaczników brzegowych:	$33 + 7A$	18, 29, 31 lub 34
System bloków bliźniaczych:	$19 + 25R$	$27 + 26S$

Tu $A \geq 1$ jest liczbą iteracji przy wyszukiwaniu odpowiednio dużego wolnego bloku; $R \geq 0$ jest liczbą podziałów bloku na pół (początkowa różnica $j - k$ w algorytmie R); $S \geq 0$ jest liczbą par bloków bliźniaczych łączonych przez algorytm S. Eksperyment symulacyjny pokazał, że przy opisanych założeniach oraz rozkładzie rozmiaru bloku (S_1) i czasie życia bloku wybieranym spośród liczb od 1 do 1000 mamy średnio $A = 2.8$, $R = S = 0.04$. (Przy opisany wcześniejszej rozkładzie „prawie-pierwszy-wchodzi-pierwszy-wychodzi” uzyskano wartości $A = 1.3$, $R = S = 0.9$). Wynika z tego, że obie metody są względnie szybkie, a system bloków bliźniaczych na komputerze **MIX** ma niewielką przewagę. Pamiętajmy, że system bloków bliźniaczych wymaga około 44 procent dodatkowego miejsca, jeśli rozmiary bloków nie są potęgami dwójki.

Analogiczne oszacowanie czasu dla metody odśmiecania i kompresji (zobacz ćwiczenie 33) wynosi około 104 jednostki czasu na znalezienie wolnego bloku przy założeniu, że odśmiecanie jest wykonywane, gdy pamięć zapełni się w połowie, a elementy struktur danych mają średni rozmiar 5 i 2 dowiezania na element. Zalety i wady odśmiecania są omówione w punkcie 2.3.5. Jeśli pamięć nie jest zajeta w dużym stopniu i kiedy przestrzega się pewnych ograniczeń, metoda

odśmiecania i kompresji jest bardzo wydajna. Na przykład na komputerze MIX metoda odśmiecania jest szybsza od opisanych powyżej dwóch metod jawnego zwalniania pamięci, jeśli osiągalne elementy są względnie małe i zajmują w sumie nie więcej niż około jedną trzecią pamięci.

Jeśli program spełnia wymagania umożliwiające zastosowanie odśmiecania, to najlepszą strategią może okazać się podzielenie pamięci na połowy i przydzielanie kolejnych bloków pamięci w jednej z nich. Bloków w żaden sposób nie zwalniamy. Czekamy po prostu, aż połowa pamięci się przepłni. Wtedy kopujemy wszystkie osiągalne bloki do drugiej połowy (porównaj ćwiczenie 33), układając je po kolej (w wyniku czego usuwamy dziury po zwolnionych blokach). Przy operacji przenoszenia bloków można dodatkowo pokusić się o modyfikacje rozmiarów sterty (stert).

Za pomocą opisanych metod symulacyjnych przetestowano także inne algorytmy przydziału pamięci. Inne metody okazały się tak mało wydajne, że jedynie krótko je opiszemy.

a) Utrzymywano oddzielną listę AVAIL dla każdego rozmiaru bloku. Jeśli była taka potrzeba pojedynczy blok dzielono na dwa mniejsze, ale nie próbowało ich później w żaden sposób scalać. Sterta rozpadała się na coraz mniejsze bloki. Ten prosty schemat jest równoważny niezależnemu przydzielaniu pamięci w rozłącznych obszarach. Jeden rozmiar bloku odpowiada jednemu obszarowi.

b) Przydział dwupoziomowy: pamięć była podzielona na 32 duże sektory. Duże bloki pamięci, zajmujące 1, 2 lub 3 (raczej nie więcej) przylegające sektory, przydzielano za pomocą metody „siłowej”. Każdy taki duży blok dzielono na mniejsze, odpowiadające żądaniom przydziału pamięci. Gdy skończyło się miejsce w dużym bloku, przydzielano następny duży blok i w nim dokonywano kolejnych „małych” przydziałów. Duży blok zwalniano wtedy, gdy *wszystkie* jego małe bloki zostały zwolnione. Ta metoda w większości przypadków bardzo szybko powoduje przekroczenie pojemności pamięci.

Chociaż ta konkretna metoda dwupoziomowego przydziału pamięci słabo sprawdziła się dla danych testowych w eksperymentach autora, istnieją okoliczności (całkiem często występujące w praktyce), w których wielopoziomowe przydzielanie pamięci może okazać się wydajne. Jeśli na przykład duży program działa w kilku fazach, to możemy wiedzieć, że węzły pewnego typu są potrzebne jedynie w jakimś konkretnym podprogramie. Czasami pozyteczne jest także stosowanie różnych strategii przydziału pamięci dla struktur danych różnego typu. Pomysł przydziału całych stref pamięci, gdzie w każdej strefie stosujemy być może inną strategię, a także mamy możliwość zwolnienia całej strefy, jest omówiony w pracy Dougla T. Rossa, *CACM* **10** (1967), 481–492.

Więcej wyników empirycznych dotyczących dynamicznego przydziału pamięci można znaleźć w: B. Randell, *CACM* **12** (1969), 365–369, 372; P. W. Purdom, S. M. Stigler, T. O. Cheam, *BIT* **11** (1971), 187–195; B. H. Margolin, R. P. Parmelee, M. Schatzoff, *IBM Systems J.* **10** (1971), 283–304; J. A. Campbell, *Comp. J.* **14** (1971), 7–9; John E. Shore, *CACM* **18** (1975), 433–440; Norman R. Nielsen, *CACM* **20** (1977), 864–873.

***E. Dopasowanie według rozkładu.** Jeśli rozkład rozmiarów bloków jest z góry znany oraz jeśli prawdopodobieństwo zwolnienia zajętych bloków jest takie samo i nie zależy od tego, kiedy je przydzielono, to możemy się posłużyć metodą odznaczającą się istotnie lepszym wykorzystaniem pamięci niż uniwersalne techniki opisane do tej pory (pomysł pochodzi od E. G. Coffmana, Jr. i F. T. Leightona [J. Computer and System Sci. 38, (1989), 2–35]). Metoda „dopasowania wg rozkładu” polega na podziale pamięci na mniej więcej $N + \sqrt{N} \lg N$ przegródek, gdzie N jest maksymalną liczbą bloków, które chcemy utrzymywać na stercie w stanie równowagi. Każda przegródka ma stały rozmiar, chociaż różne przegródki mogą mieć różne rozmiary. Każda przegródka ma ustalone granice i może być albo pusta, albo zawierać dokładnie jeden przydzielony blok.

Pierwszych N przegródek w metodzie Coffmana i Leightona ma taki rozkład rozmiarów, jak przewidywany rozkład rozmiarów bloków, a ostatnich $\sqrt{N} \lg N$ przegródek ma rozmiar maksymalny. Jeśli na przykład założymy, że rozmiary bloków będą rozłożone równomiernie w przedziale od 1 do 256 i jeśli chcemy obsługiwać jednocześnie $N = 2^{14}$ takich bloków, to podzielimy pamięć na $N/256 = 2^6$ przegródek o rozmiarach 1, 2, …, 256, po których umieszczać „obszar awaryjny” zawierający $\sqrt{N} \lg N = 2^7 \cdot 14 = 1792$ bloków rozmiaru 256. Gdy system pracuje przy pełnym obciążeniu, spodziewamy się N bloków o średnim rozmiarze $\frac{257}{2}$, zajmujących $\frac{257}{2}N = 2^{21} + 2^{13} = 2105344$ lokacji; jest to przestrzeń zajmowana przez pierwszych N przegródek. Odłożyliśmy także dodatkowe $1792 \cdot 256 = 458,752$ lokacji, by poradzić sobie z losowymi fluktuacjami. Ten narzut wynosi $O(N^{-1/2} \log N)$ rozmiaru całej pamięci (w przypadku systemu bloków bliźniaczych mieliśmy narzut proporcjonalny do N), zatem przy $N \rightarrow \infty$ jest zaniedbywalny. W naszym przykładzie stanowi mimo wszystko 18% rozmiaru pamięci.

Przegródki należy uporządkować w ten sposób, żeby mniejsze występuły wcześniej. Przy takim uporządkowaniu możemy korzystać z metody pierwszego dopasowania lub najlepszego dopasowania. (W tym przypadku, ze względu na uporządkowanie przegródek, te dwie metody są równoważne). Po nadjęciu żądania przydziału wyszukiwanie wolnego bloku rozpoczynamy od losowej pośród pierwszych N przegródek.

Jeśli początkowa przegródka przy każdym wyszukiwaniu jest rzeczywiście wybrana losowo spośród przegródek $1 - N$, to nie będziemy wchodzili zbyt często na obszar nadmiarowy. Jeśli przydzielimy N bloków, każdorazowo rozpoczynając poszukiwanie w losowym miejscu, to przepełnienie pierwszych N przegródek wystąpi średnio jedynie w $O(\sqrt{N})$ przypadkach. Łatwo się o tym przekonać, porównując powyższy algorytm z algorytmem haszowania z próbkowaniem liniowym (algorytm 6.4L), zachowującym się bardzo podobnie. Różnica polega na tym, że zamiast wchodzić na obszar nadmiarowy, kontynuujemy poszukiwanie od początku tablicy. Analiza algorytmu 6.4L w twierdzeniu 6.4K pokazuje, że po wstawieniu N elementów średnie przemieszczenie każdego elementu względem jego adresu haszowania wynosi $\frac{1}{2}(Q(N) - 1) \sim \sqrt{\pi N/8}$; z „symetrii cyklicznej” wartość ta odpowiada średniej liczbie kroków wyszukiwania bloku. Przepełnienie początkowego obszaru w metodzie dopasowania wg rozkładu odpowiada wyszu-

kiwaniu w tablicy z haszowaniem, w którym przechodzimy od przegródki N do przegródki 1. Z tym że jesteśmy w lepszej sytuacji, bo unikamy pewnych zatorów, dysponując dodatkowo obszarem awaryjnym. Stąd średnio wystąpi mniej niż $\sqrt{\pi N}/8$ przepełnień. W tej analizie nie uwzględniamy operacji zwalniania bloku, która zachowywałaby założenia algorytmu 6.4L, jeśli przesuwalibyśmy bloki w tył, przy usuwaniu bloku, który znajdował się między początkową a ostatecznie przydzieloną przegródką (zobacz algorytm 6.4R). Przesuwanie bloków zwiększyłoby jednak tylko szanse wystąpienia przepełnienia. W naszej analizie nie uwzględniamy także efektu pojawienia się więcej niż N bloków jednocześnie. Tak może się stać, gdy założymy, że czas pojawiania się kolejnych bloków jest równy około $1/N$ czasu życia bloku. Dla przypadku więcej niż N bloków musielibyśmy rozszerzyć analizę algorytmu 6.4L, ale Coffman i Leighton udowodnili, że obszar awaryjny prawie nigdy nie wymaga ponad $\sqrt{N} \lg N$ przegródek. Prawdopodobieństwo wyczerpania całej pamięci łącznie z obszarem awaryjnym jest mniejsze niż $O(N^{-M})$ dla dowolnego M .

W naszym przykładzie numer przegródki, od której rozpoczynamy wyszukiwanie, nie ma rozkładu jednostajnego na przedziale 1, 2, ..., N . Ma za to rozkład jednostajny na zbiorze 1, 65, 129, ..., $N - 63$, ponieważ są $N/256 = 64$ przegródki każdego rozmiaru. Ale to odstępstwo od modelu losowego sprawia, że prawdopodobieństwo przepełnienia jest jeszcze mniejsze. Wszystko oczywiście bierze w leb, jeśli nie są spełnione założenia dotyczące rozkładu rozmiarów bloków i czasu życia bloku.

F. Przepełnienie. Co robić, gdy nie ma wolnego miejsca? Przypuśćmy, że zgłoszono żądanie przydziału bloku o rozmiarze n , a wszystkie aktualnie wolne bloki są zbyt małe. Zazwyczaj w momencie pierwszego wystąpienia takiej sytuacji wolnych jest więcej niż n lokacji, ale nie tworzą one odpowiednio dużego, spójnego bloku. Kompresja pamięci (tj. przesunięcie niektórych zajętych lokacji w celu zgrupowania wolnych bloków) umożliwi zrealizowanie żądania. Ale kompresja jest wolna i wymaga dyscypliny w korzystaniu ze wskaźników. Ponadto w olbrzymiej większości przypadków wystąpienie przepełnienia przy korzystaniu z metody pierwszego dopasowania oznacza, że wcześniej czy później definitelywnie zabraknie pamięci. Z tego powodu nie warto zazwyczaj pisać programu kompresującego pamięć, chyba że mamy do czynienia ze szczególnymi okolicznościami związanymi z odśmiecaniem, jak w ćwiczeniu 33. Jeśli spodziewamy się przepełniania, to można skorzystać z metod przenoszenia bloków z pamięci wewnętrznej do pamięci zewnętrznej pod warunkiem, że będziemy wiedzieli, jak i kiedy sprowadzić blok, gdy będzie potrzebny. To oznacza, że programy operujące na tak zorganizowanej pamięci dynamicznej muszą podlegać surowym ograniczeniom dotyczącym odwołań do innych bloków. W ogólności wydajne stosowanie tego typu metod wymaga obecności specjalnych mechanizmów sprzętowych (na przykład przerwań spowodowanych brakiem danych, automatycznego stronicowania itp.).

Trzeba w jakiś sposób decydować, które bloki są najlepszymi kandydatami do usunięcia. Jeden z pomysłów polega na utrzymywaniu listy dwukierunkowej zarezerwowanych bloków, do których dawno nie było odwołania, i przesuwaniu

jej początek każdego bloku, do którego wystąpiło odwołanie. Bloki są dzięki temu posortowane skutecznie w porządku ostatniego odwołania – blok na końcu listy jest pierwszym kandydatem do usunięcia. Podobny skutek można osiągnąć metodą nieco prostszą. Umieszczamy wszystkie bloki na liście cyklicznej, a w każdym z nich utrzymujemy znacznik „ostatnio użyty”. Znacznik ustawiamy na 1 przy każdym dostępie do bloku. Gdy trzeba usunąć blok, przeglądamy listę cykliczną, ustawiając znaczniki na zero, aż do napotkania bloku, który ma znacznik równy zero.

J. M. Robson pokazał [JACM 18 (1971), 416–423], że strategie dynamicznego przydziału pamięci, które nie przemieszczają przydzielonych bloków, nie mogą efektywnie wykorzystywać pamięci. Zawsze można wskazać sytuacje patologiczne, w których taka metoda się nie sprawdza. Na przykład, jeśli bloki są wyłącznie rozmiaru 1 i 2, to przepełnienie może wystąpić przy pamięci zapełnionej w dwóch trzecich niezależnie od stosowanego algorytmu! Wyniki Robsona omawiamy w ćwiczeniach 36–40, a w ćwiczeniach 42–43 wskazujemy naprawdę beznadziejny przypadek zachowania się metody najlepszego dopasowania w porównaniu z metodą pierwszego dopasowania.

G. Bibliografia. Szczegółowy przegląd krytyczny technik dynamicznego przydziału pamięci przedstawiono w: Paul R. Wilson, Mark S. Johnstone, Michael Neely, David Boles, *Lecture Notes in Computer Science* 986 (1995), 1–116.

ĆWICZENIA

1. [20] Jakie uproszczenia można wprowadzić do algorytmu przydziału i zwalniania pamięci, jeśli żądania pojawiają się zawsze w kolejności „ostatni-wchodzi-pierwszy-wychodzi”, tj. jeśli nie zwalniamy bloku, zanim nie zwolnimy wszystkich bloków zarezerwowanych po nim?
2. [HM23] (E. Wolman). Przypuśćmy, że chcemy pracować z elementami o jednakowym rozmiarze, ale przechowywać informacje różnej wielkości. Przypuśćmy ponadto, że jeżeli informacja zajmuje l słów, a element ma rozmiar k słów, to do jej przechowywania potrzeba $\lceil l/(k - b) \rceil$ elementów. (Stała b oznacza, że w każdym elemencie występuje b słów kontrolnych, takich jak na przykład wskaźnik do następnego elementu). Jaka wartość k minimalizuje rozmiar potrzebnej pamięci, gdy średnia wartość l wynosi L ? (Załóż, że wartość średnia $(l/(k - b)) \bmod 1$ przy sumowaniu względem l i dla ustalonego k wynosi $1/2$).
3. [40] Posługując się symulacją komputerową, porównaj następujące metody przydziału pamięci: metodę najlepszego dopasowania, metodę pierwszego dopasowania i metodę *najgorszego dopasowania*. Ostatnia z wymienionych metod polega na wybieraniu zawsze największego wolnego bloku. Czy jest jakaś istotna różnica w ilości użytej pamięci?
4. [22] Napisz program na komputer MIX realizujący algorytm A. Zwróć szczególną uwagę na to, by pętla wewnętrzna była szybka. Załóż, że pole SIZE zajmuje wycinek (4:5), pole LINK wycinek (0:2) oraz $\Lambda < 0$.
- ▶ 5. [18] Przypuśćmy, że wiadomo, iż N w algorytmie A nigdy nie jest mniejsze niż 100. Czy przyjęcie $c = 100$ w zmodyfikowanym kroku A4' to dobry pomysł?

- ▶ **6.** [23] (*Metoda kolejnego dopasowania*) Po wielu wykonaniach algorytmu A daje się zauważać tendencja do grupowania się małych bloków w początkowej części listy **AVAIL**, zatem daleko trzeba szukać, by znaleźć większy blok. Zwróćmy uwagę, jak zwiększa się rozmiar bloków na rysunku 42 wraz ze wzrostem adresów. (Zgodnie z wymaganiami algorytmu B lista **AVAIL**, na podstawie której sporządzono rysunek, była posortowana względem adresów bloków). Czy umiesz zaproponować zmiany w algorytmie A, dzięki którym (a) krótkie bloki nie miałyby tendencji do grupowania się w jednym miejscu; (b) listę **AVAIL** można by w przechowywać posortowaną względem adresów bloków, jak tego wymaga algorytm B?
- 7.** [10] Przykład (1) pokazuje, że metoda pierwszego dopasowania może w pewnych sytuacjach być zdecydowanie lepsza od metody najlepszego dopasowania. Podaj podobny przykład obrazujący przypadek, w którym metoda najlepszego dopasowania jest lepsza od metody pierwszego dopasowania.
- 8.** [21] Pokaż, jak w prosty sposób zmodyfikować algorytm A, by zamiast metody pierwszego dopasowania otrzymać metodę najlepszego dopasowania.
- ▶ **9.** [26] Jak można zaprojektować algorytm przydziału pamięci metodą najlepszego dopasowania, tak by nie musiał przeszukiwać całej listy **AVAIL**? (Pomyśl o sposobach zoptymalizowania operacji wyszukiwania).
- 10.** [22] Pokaż, w jaki sposób zmodyfikować algorytm B, by można było zwolnić blok N kolejnych lokacji, poczynając od lokacji P0, bez dbania o to, czy rzeczywiście każda z tych N lokacji jest aktualnie zajęta. W istocie należy przyjąć, że zwalniany obszar może zachodzić na obszary aktualnie wolne.
- 11.** [M25] Pokaż, że modyfikacja algorytmu A proponowana w odpowiedzi do ćwiczenia 5 sugeruje również niewielką zmianę algorytmu B, powodującą zmniejszenie średniej liczby przeszukiwanych bloków z połowy do jednej trzeciej długości listy **AVAIL**. (Załącz, że zwalniany blok będzie wstawiany w losowe miejsce listy **AVAIL**).
- ▶ **12.** [20] Zmodyfikuj algorytm A, by korzystał z konwencji wskaźników brzegowych (porównaj (7) – (9)), zawierał modyfikacje zawarte w kroku A4' oraz ulepszenia opisane w ćwiczeniu 6.
- 13.** [21] Napisz program na komputer **MIX** realizujący algorytm opisany w ćwiczeniu 12.
- 14.** [21] Jakie znaczenie dla algorytmu C i algorytmu z ćwiczenia 12 ma obecność pola **SIZE** w: (a) ostatnim słowie wolnego bloku, (b) pierwszym słowie zajętego bloku?
- ▶ **15.** [24] Pokaż, w jaki sposób można nieco przyspieszyć algorytm C (kosztem dłuższego kodu). *Wskazówka:* W każdym z czterech przypadków wyznaczonych przez znaki **TAG(P0 - 1)**, **TAG(P0 + SIZE(P0))** można ograniczyć się do zmieniania tylko tych wskaźników, które rzeczywiście trzeba zmienić.
- 16.** [24] Napisz program na komputer **MIX** realizujący algorytm C i uwzględniający pomysł z ćwiczenia 15.
- 17.** [10] Jaka powinna być wartość **LOC(AVAIL)** i **LOC(AVAIL) + 1** w (9), gdy nie ma wolnych bloków?
- ▶ **18.** [20] Rysunki 42 i 43 uzyskano, stosując te same ciągi danych i te same algorytmy (algorytmy A i B). Przygotowując rysunek 43, tak zmodyfikowano algorytm A, by korzystał z metody najlepszego dopasowania, a nie pierwszego dopasowania. Dlaczego spowodowało to, iż na rysunku 42 więcej wolnych lokacji jest w górnym adresach, a na rysunku 43 w dolnych?

- 19. [24] Przypuśćmy, że blok pamięci ma postać (7), ale nie ma pól TAG i SIZE w ostatnim słowie. Przypuśćmy ponadto, że zwalniamy bloki za pomocą poniższego algorytmu: $Q \leftarrow \text{AVAIL}$, $\text{LINK}(P_0) \leftarrow Q$, $\text{LINK}(P_0 + 1) \leftarrow \text{LOC}(\text{AVAIL})$, $\text{LINK}(Q + 1) \leftarrow P_0$, $\text{AVAIL} \leftarrow P_0$, $\text{TAG}(P_0) \leftarrow \text{--}$. (Ten algorytm nie powoduje scalania przylegających obszarów).

Zaprojektuj algorytm przydziału pamięci podobny do algorytmu A, który realizuje scalanie przyległych bloków podczas przeszukiwania listy AVAIL, tym samym zapobiegając niepotrzebnej fragmentacji pamięci, jak w przykładach (2)–(4).

20. [00] Dlaczego warto, by w systemie bloków bliźniaczych listy AVAIL[k] były dwukierunkowe?

21. [HM25] Zbadaj iloraz a_n/b_n przy $n \rightarrow \infty$, gdzie a_n jest sumą pierwszych n wyrazów spośród $1 + 2 + 4 + 4 + 8 + 8 + 8 + 16 + 16 + \dots$, a b_n jest sumą pierwszych n wyrazów spośród $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + \dots$.

- 22. [21] Tekst stwierdza, że w systemie bloków bliźniaczych można korzystać tylko z bloków o rozmiarze 2^k , a ćwiczenie 21 pokazuje, że może to prowadzić do istotnego wzrostu zapotrzebowania na rozmiar pamięci. Ale jeśli potrzebny jest blok o rozmiarze 11 słów, to czy nie można znaleźć bloku o rozmiarze 16 i podzielić go na blok o rozmiarze 11 oraz dwa bloki o rozmiarach 4 i 1?

23. [05] Jaki jest adres dwójkowy bloku bliźniaczego dla bloku o rozmiarze 4, którego adresem jest 011011110000? Co by było, gdyby blok miał rozmiar 16, a nie 4?

24. [20] Zgodnie z algorytmem podanym w tekście największy blok (o rozmiarze 2^m) nie ma bloku bliźniaczego, ponieważ reprezentuje całą pamięć. Czy można zdefiniować $\text{buddy}_m(0) = 0$ (czyli uczynić ten blok własnym blokiem bliźniaczym) i pozbyć się sprawdzania warunku $k = m$ w kroku S1?

- 25. [22] Zaopiniuj następujący pogląd: „Dynamiczny przydział pamięci korzystający z systemu bloków bliźniaczych w praktyce nigdy nie przydzieli bloku o rozmiarze 2^m (ponieważ zająłby całą pamięć), a w ogólności jest taki rozmiar bloku 2^n , że większe bloki nie zostaną nigdy przydzielone. Z tego powodu nie warto rozpoczynać pracy z takimi dużymi blokami ani scalać ich w algorytmie S, jeśli rozmiar wynikowego bloku jest większy niż 2^n ”.

- 26. [21] Wyjaśnij, w jaki sposób można korzystać z dynamicznego przydziału komórek pamięci o adresach od 0 do M–1 za pomocą systemu bloków bliźniaczych w sytuacji, gdy M nie jest postaci 2^m .

27. [24] Napisz program na komputer MIX realizujący algorytm R i wyznacz czas jego działania.

28. [25] Założmy, że MIX jest komputerem binarnym i że dysponuje nową instrukcją XOR zdefiniowaną następująco (korzystamy z oznaczeń z punktu 1.3.1): „C = 5, F = 5. Dla każdego bitu w lokacji M, który ma wartość 1, wartość odpowiadającego mu bitu w rejestrze A jest zmieniana na przeciwną; znak rA się nie zmienia. Czas wykonania wynosi 2u”.

Napisz program na komputer MIX realizujący algorytm S i wyznacz czas jego działania.

29. [20] Czy system bloków bliźniaczych może działać bez znaczników bitowych w każdym zarezerwowanym bloku?

30. [M48] Przeanalizuj zachowanie algorytmów R i S dla rozsądnych rozkładów ciągów żądań.

31. [M40] Czy można zaprojektować system analogiczny do systemu bloków bliźniaczych oparty na liczbach Fibonacciego, a nie na potęgach dwójki? (Moglibyśmy zaczynać od pamięci o rozmiarze F_m i dzielić wolny blok rozmiaru F_k na dwa bloki „bliźniacze” o rozmiarach F_{k-1} i F_{k-2}).

32. [HM47] Wyznacz granicę $\lim_{n \rightarrow \infty} \alpha_n$ (jeśli istnieje), gdzie α_n jest średnią wartością t_n w następującym ciągu: Przypuśćmy, że mamy wartości t_k dla $1 \leq k < n$; t_n wybieramy z rozkładem jednostajnym z pośród $\{1, 2, \dots, g_n\}$, gdzie

$$g_n = \lfloor \frac{5}{4} \min(10000, f(t_{n-1} - 1), f(t_{n-2} - 2), \dots, f(t_1 - (n-1))) \rfloor,$$

i $f(x) = x$ dla $x > 0$, $f(x) = \infty$ dla $x \leq 0$. (Uwaga: Pewne badania empiryczne wskazują, że α_n może w przybliżeniu równać się 14, ale zapewne nie jest to bardzo dokładna wartość).

- **33.** [28] (*Odśmietanie i kompresja*) Założymy, że lokacje 1, 2, ..., AVAIL-1 są wykorzystywane jako sterta do przechowywania węzłów struktur danych o różnych rozmiarach i następującej postaci: pierwsze słowo węzła NODE(P) zawiera pole

$\text{SIZE}(P)$ = liczba słów w NODE(P);

$T(P)$ = liczba wskaźników w NODE(P); $T(P) < \text{SIZE}(P)$;

$\text{LINK}(P)$ = specjalne pole z dowiązaniem na potrzeby odśmietania.

W pamięci bezpośrednio za węzłem NODE(P) znajduje się węzeł NODE($P + \text{SIZE}(P)$). Zakładamy, że jedynymi polami węzła NODE(P) zawierającymi dowiązania do innych węzłów są $\text{LINK}(P+1)$, $\text{LINK}(P+2)$, ..., $\text{LINK}(P+T(P))$ i że każde z tych pól zawiera albo Λ , albo adres pierwszego słowa innego węzła. Zakładamy również, że istnieje ponadto tylko jedna zmienna dowiązaniowa USE wskazująca na jeden z węzłów.

Zaprojektuj algorytm, który (i) wyznacza wszystkie węzły osiągalne za pomocą zmiennej USE, (ii) przesuwa te wszystkie węzły do lokacji od 1 do K-1, dla pewnego K, modyfikując odpowiednio wszystkie dowiązania, (iii) przypisuje $\text{AVAIL} \leftarrow K$.

Rozważmy na przykład następującą zawartość pamięci, gdzie $\text{INFO}(L)$ oznacza zawartość lokacji L bez wskaźnika $\text{LINK}(L)$:

1: $\text{SIZE} = 2$, $T = 1$	6: $\text{SIZE} = 2$, $T = 0$	$\text{AVAIL} = 11$,
2: $\text{LINK} = 6$, $\text{INFO} = A$	7: $\text{CONTENTS} = D$	$\text{USE} = 3$.
3: $\text{SIZE} = 3$, $T = 1$	8: $\text{SIZE} = 3$, $T = 2$	
4: $\text{LINK} = 8$, $\text{INFO} = B$	9: $\text{LINK} = 8$, $\text{INFO} = E$	
5: $\text{CONTENTS} = C$	10: $\text{LINK} = 3$, $\text{INFO} = F$	

Po wykonaniu Twojego algorytmu pamięć powinna wyglądać tak:

1: $\text{SIZE} = 3$, $T = 1$	4: $\text{SIZE} = 3$, $T = 2$	$\text{AVAIL} = 7$,
2: $\text{LINK} = 4$, $\text{INFO} = B$	5: $\text{LINK} = 4$, $\text{INFO} = E$	$\text{USE} = 1$.
3: $\text{CONTENTS} = C$	6: $\text{LINK} = 1$, $\text{INFO} = F$	

34. [29] Napisz program na komputer MIX realizujący algorytm z ćwiczenia 33 i wyznacz czas jego działania.

35. [22] Zestaw metody dynamicznego przydziału pamięci opisane w tym podrozdziale z metodami realizacji list tablicowych o zmiennych rozmiarach, opisanymi w punkcie 2.2.2.

- **36.** [20] W pewnym lokalu w Hollywood przy barze mogą siedzieć jednocześnie 23 osoby. Goście wchodzą w grupach jedno lub dwuosobowych, a kelnerka wskazuje im

miejsca. Udowodnij, że zawsze da się usadzić gości bez rozdzielania par, jeśli pojedynczych klientów nie sadzamy na miejscach 2, 5, 8, ..., 20 przy założeniu, że liczba gości nigdy nie przekracza 16. (Obie osoby z pary wychodzą jednocześnie).

- 37. [26] Ciąg dalszy ćwiczenia 36. Udowodnij, że kelnerka nie ma szans, jeśli przy barze będą tylko 22 miejsca: niezależnie od strategii usadzania gości może się zdarzyć, że choć siedzi tylko 14 osób, to nie można usadzić pary.

38. [M21] (J. M. Robson) Problem barowy z ćwiczeń 36 i 37 można uogólnić w celu znalezienia najgorszego zachowania dowolnego algorytmu dynamicznego przydziału pamięci, który nie przemieszcza zajętych bloków. Niech $N(n, m)$ będzie najmniejszym rozmiarem pamięci, takim że dowolny ciąg żądań przydziałów i zwolnień można obsłużyć bez przepełnienia przy założeniu, że rozmiary wszystkich bloków są $\leq m$, a całkowity rozmiar żądanej pamięci nie przekracza n . Ćwiczenia 36 i 37 pokazują, że $N(16, 2) = 23$; wyznacz dokładną wartość $N(n, 2)$ dla dowolnego n .

39. [HM23] (J. M. Robson) Przy oznaczeniach takich jak w ćwiczeniu 38 pokaż, że $N(n_1 + n_2, m) \leq N(n_1, m) + N(n_2, m) + N(2m - 2, m)$; stąd dla ustalonego m istnieje granica $N(m) = \lim_{n \rightarrow \infty} N(n, m)/n$.

40. [HM50] Ciąg dalszy ćwiczenia 39. Wyznacz $N(3)$, $N(4)$ i $\lim_{m \rightarrow \infty} N(m)/\lg m$, jeśli istnieje.

41. [M27] Celem tego ćwiczenia jest analiza najgorszego przypadku wykorzystania pamięci w systemie bloków bliźniaczych. Pewien niekorzystny przypadek pojawia się na przykład, gdy weźmiemy pustą pamięć i postępujemy tak: rezerwujemy $n = 2^{r+1}$ bloków o długości 1, które zostają przydzielone w lokacjach od 0 do $n-1$; następnie dla $k = 1, 2, \dots, r$ zwalniamy wszystkie bloki zaczynające się w lokacjach niepodzielnych przez 2^k i rezerwujemy $2^{-k-1}n$ bloków długości 2^k , które zostają przydzielone w lokacjach od $\frac{1}{2}(1+k)n$ do $\frac{1}{2}(2+k)n - 1$. Przydzielone jest $1 + \frac{1}{2}r$ raza tyle pamięci, ile chcieliśmy zarezerwować.

Udowodnij, że najgorszy przypadek nie może być istotnie gorszy od powyższego: jeśli wszystkie żądania przydziału bloków o rozmiarach 1, 2, ..., 2^r , a także sumaryczne żądanie przydziałów pamięci w dowolnej chwili nie przekracza n , gdzie n jest wielokrotnością 2^r , to w systemie bloków bliźniaczych nie nastąpi przepełnienie pamięci rozmiaru $(r+1)n$.

42. [M40] (J. M. Robson, 1975) Niech $N_{BF}(n, m)$ będzie rozmiarem pamięci potrzebnym, by zagwarantować brak przepełnienia, gdy do przydziału w ćwiczeniu 38 stosujemy metodę najlepszego dopasowania. Znajdź strategię postępowania, pozwalającą pokazać, że $N_{BF}(n, m) \geq mn - O(n + m^2)$.

43. [HM35] Ciąg dalszy ćwiczenia 42. Niech $N_{FF}(n, m)$ będzie pamięcią potrzebną przy algorytmie pierwszego dopasowania. Znajdź strategię obrony, pozwalającą pokazać, że $N_{FF}(n, m) \leq H_m n / \ln 2$. (Stąd najgorszy przypadek dla metody pierwszego dopasowania jest bliski najlepszemu najgorszemu przypadkowi).

44. [M21] Przypuśćmy, że rozkład $F(x) = (\text{prawdopodobieństwo, że blok ma rozmiar } \leq x)$ jest ciągły. Na przykład $F(x)$ jest określony wzorem $(x-a)/(b-a)$ dla $a \leq x \leq b$, jeśli rozmiar bloku ma rozkład jednostajny na przedziale $a \leq x \leq b$. Podaj wzór wyrażający rozmiar pierwszych N przegródek w metodzie dopasowania według rozkładu.

2.6. HISTORIA I BIBLIOGRAFIA

Listy liniowe oraz tablice prostokątne przechowywane w kolejnych komórkach pamięci były używane od zarania komputerów, a pierwsze traktaty o programowaniu podawały podstawowe algorytmy obchodzenia tych struktur. [Zobacz na przykład J. von Neumann, *Collected Works 5*, 113–116 (powstałe w 1946 roku); M. V. Wilkes, D. J. Wheeler, S. Gill, *The Preparation of Programs for an Electronic Digital Computer* (Reading, Mass.: Addison-Wesley, 1951), podprogram V-1; szczególnie polecamy: Konrad Zuse, *Berichte der Gesellschaft für Mathematik und Datenverarbeitung 63* (Bonn: 1972), napisane w 1945 roku. Zuse jako pierwszy opracował nietrywialny algorytm pracujący na listach o dynamicznie zmieniającej się długości]. Zanim pojawiły się rejestrystre indeksowe, operacje na listach sekwencyjnych były realizowane za pomocą operacji arytmetycznych z wykorzystaniem rozkazów języka maszynowego, a konieczność wykonywania takich operacji była jedną z pierwszych przesłanek przemawiających za zaprojektowaniem komputerów, w których programy znajdują się w tej samej pamięci co ich dane.

Metody umożliwiające sekwencyjne przechowywanie w jednej tablicy wielu list o zmiennej długości, które w razie potrzeby są przesuwane w górę lub w dół, jak te opisane w punkcie 2.2.2, opracowano zdecydowanie później. J. Dunlap z Digitek Corporation rozwinał takie techniki przed 1963 rokiem w związku z projektem serii kompilatorów. Mniej więcej w tym samym czasie analogiczny pomysł pojawił się niezależnie w projekcie kompilatora języka COBOL w IBM Corporation, a biblioteka związanych z tą techniką podprogramów, o nazwie CITRUS, była używana na różnych komputerach. Metody te pozostały nieopublikowane aż do dnia, kiedy niezależnie odkrył je Jan Garwick z Norwegii; zobacz *BIT 4* (1964), 137–140.

Wydaje się, że pomysł *niesekwencyjnych* list liniowych powstał w związku z konstrukcją komputerów z pamięcią bębnową. Po wykonaniu rozkazu z lokacji n taki komputer zazwyczaj nie mógł wykonać rozkazu z lokacji $n + 1$, ponieważ bęben zdążył się już obrócić dalej. W zależności od rozkazu, następną pozycję mogło być na przykład $n + 7$ lub $n + 18$, a komputer mógł działać sześć do siedmiu razy szybciej, jeśli instrukcje były rozłożone optymalnie, a nie w kolejnych lokacjach. [Omówienie ciekawych problemów związanych z rozmieszczeniem rozkazów można znaleźć w artykule autora w *JACM 8* (1961), 119–150]. Z powyższych powodów w kodzie każdego rozkazu maszynowego było specjalne pole, w którym było przechowywane dowiązanie do następnego rozkazu. Ta koncepcja, nazywana „adresowaniem jeden-plus-jeden”, została omówiona przez Johna Mauchly’ego w 1946 roku [*Theory and Techniques for the Design of Electronic Computers 4* (U. of Pennsylvania, 1946), wykład 37]. Przedstawała ona listy z dowiązaniemi w postaci zarodkowej, chociaż operacje dynamicznego wstawiania i usuwania były wciąż nieznane. Pomyśl dowiązań pojawił się również w memorandum H. P. Luhna z 1953 roku, w którym było wspomniane o wykorzystaniu „łańcuchów” przy wyszukiwaniu zewnętrznym; zobacz podrozdział 6.4.

Tak naprawdę narodziny technik wskaźnikowych miały jednak miejsce wtedy, gdy A. Newell, J. C. Shaw i H. A. Simon rozpoczęli prace nad heurystycznym rozwiązywaniem problemów za pomocą maszyn. Niejako przy okazji, wiosną 1965 roku opracowali oni pierwszy język zawierający operacje na Listach – IPL-II, gdyż potrzebowali takiego narzędzia w swoich programach poszukujących dowodów wyrażonych w logice matematycznej. (IPL jest skrótem od *Information Processing Language*). Ich system korzystał ze wskaźników, a także posługiwał się listą wolnej pamięci, ale pojęcie stosów nie było wtedy jeszcze dobrze poznane. IPL-III, zaprojektowany rok później, zawierał już operacje wkładania na stos i zdejmowania ze stosu. [Więcej informacji dotyczących IPL-II można znaleźć w: *IRE Transactions IT-2* (September 1956), 61–70; *Proc. Western Joint Comp. Conf.* 9 (1957), 218–240. Materiały na temat IPL-III po raz pierwszy ukazały się jako notatki do wykładów na University of Michigan latem 1957 roku].

Prace Newella, Shawa i Simona zainspirowały wielu programistów do korzystania ze wskaźników, które wówczas często nazywano „pamięcią NSS”, ale głównie korzystano z nich w problemach związanych z symulowaniem ludzkiego procesu myślowego. Stopniowo jednak zaczęto sobie zdawać sprawę z użyteczności tych metod jako podstawowych narzędzi programistycznych. Pierwszy artykuł opisujący korzyści płynące z zastosowania wskaźników do problemów „przyziemnych” został opublikowany przez J. W. Carra, III, w *CACM* 2, 2 (February 1959), 4–6. Carr zauważał, że listami wskaźnikowymi można w prosty sposób posługiwać się za pomocą zwykłych języków programowania, bez uciekania się do wymyślnych podprogramów lub interpreterów. Zobacz także G. A. Blaauw, „Indexing and control-word techniques”, *IBM J. Res. and Dev.* 3 (1959), 288–301.

Na początku posługiwano się strukturami danych z dowiązaniami, w których na jeden element przypadało jedno słowo maszynowe, ale około 1959 roku różne grupy osób stopniowo zaczęły dostrzegać korzyści płynące z wykorzystania elementów o większym rozmiarze oraz struktur z wieloma dowiązaniami. Pierwszy artykuł poświęcony temu zagadnieniu został opublikowany przez D. T. Rossa, *CACM* 4 (1961), 147–150.

Odniesienia do pól w ramach elementu zasadniczo można zapisywać na dwa sposoby: nazwa identyfikatora pola może występować przed lub po wyrażeniu opisującym dowiązanie. My pisaliśmy „INFO(P)”, ale inni autorzy wolą pisać, na przykład, „P.INFO”. Kiedy powstawał ten rozdział, obie notacje wydawały się być jednakowo rozpowszechnione. Przyjęta notacja ma tę zaletę, że bezpośrednio tłumaczy się na FORTRAN, COBOL lub podobne języki, jeżeli zdefiniujemy INFO i LINK jako tablice i traktujemy P jak indeks. Wydaje się ponadto, że wygodnie jest korzystać z matematycznej notacji funkcyjnej do oznaczania atrybutów elementu. Alternatywna notacja P.INFO jest mniej naturalna, ponieważ jest w niej położony nacisk na P. Zapis INFO(P) wydaje się lepszy, ponieważ P jest zmienią, a w chwili stosowania notacji INFO ma ustalone znaczenie. Przez analogię moglibyśmy traktować wektor $A = (A[1], A[2], \dots, A[100])$ jak element mający sto składowych o nazwach 1, 2, …, 100. W naszym zapisie do drugiej składowej

odwoływalibyśmy się za pomocą „ $2(P)$ ”, gdzie P wskazuje na wektor A . Ale gdy odwołujemy się do j -tego elementu wektora, to bardziej naturalny wydaje nam się zapis $A[j]$, w którym zmienna „ j ” występuje jako druga. Podobnie wydaje się bardziej naturalne zapisywanie zmiennej „ P ” na drugim miejscu w notacji $\text{INFO}(P)$.

Prawdopodobnie odkrywcami pojęcia „stosu” (ostatni-wchodzi-pierwszy-wychodzi) i „kolejki” (pierwszy-wchodzi-pierwszy-wychodzi) byli księgowi; omówienie metod odpowiednio „LIFO” i „FIFO” w kontekście inwentaryzacji można znaleźć w dowolnym podręczniku księgowości, na przykład C. F. i W. J. Schlatte, *Cost Accounting* (New York: Wiley, 1957), rozdział 7. W połowie lat czterdziestych A. M. Turing opracował mechanizm stosu (nazywany *reversion storage*) w celu obsługi wywołań podprogramów, zmiennych lokalnych i parametrów. (Zobacz punkt 1.4.5). Bez wątpienia prosty tablicowy stos występował w programach od zawsze. Stosy wskaźnikowe po raz pierwszy pojawiły się w IPL; nazwa „stos” pochodzi z terminologii IPL, ale była też niezależnie wprowadzana przez E. W. Dijkstrę [*Numer. Math.* **2** (1960), 312–318].

Pochodzenie list cyklicznych i dwukierunkowych jest niejasne; zapewne te pomysły równocześnie wydały się naturalne wielu osobom. Ważnym czynnikiem popularyzującym tego typu metody było istnienie wykorzystujących je uniwersalnych systemów przetwarzania List [głównie Knotted List Structures, *CACM* **5** (1962), 161–165, a także Symmetric List Processor, *CACM* **6** (1963), 524–544, autorstwa J. Weizenbauma]. Ivan Sutherland niezależnie wprowadził listy dwukierunkowe o dużych elementach w swoim systemie Sketchpad (praca doktorska, Mass. Inst. of Technology, 1963).

Różne metody adresowania i przechodzenia tablic wielowymiarowych były niezależnie rozwijane przez różnych zdolnych programistów od najwcześniejszych dni istnienia komputerów. W ten sposób powstała część nieopublikowanego informatycznego folkloru. Po raz pierwszy drukowany przegląd tych zagadnień autorstwa H. Hellermana pojawił się w *CACM* **5** (1962), 205–207. Zobacz także J. C. Gower, *Comp. J.* **4** (1962), 280–286.

Drzewa występujące jawnie w pamięci komputera początkowo wykorzystywano do przekształcania wyrażeń algebraicznych. Język maszynowy kilku pierwszych komputerów umożliwiał reprezentowanie obliczenia wyrażenia arytmetycznego za pomocą kodu trójadresowego. Ten kod jest odpowiednikiem pól **INFO**, **LLINK** i **RLINK** w reprezentacji drzewa binarnego. W 1952 roku H. G. Kahrmanian opracował algorytmy różniczkowania wyrażeń algebraicznych reprezentowanych za pomocą rozszerzonego kodu trójadresowego; zobacz *Symposium on Automatic Programming* (Washington, D.C.: Office of Naval Research, May 1954), 6–14.

Od tamtego czasu różne odmiany drzew w związku z ich licznymi zastosowaniami były przedmiotem studiów wielu badaczy, ale podstawowe metody posługiwania się drzewami (ale konkretnie drzewami, a nie ogólnie Listami) rzadko pojawiały się w publikacjach, z wyjątkiem szczegółowych opisów konkretnych algorytmów. Pierwszy artykuł przeglądowy na ten temat powstał w związku z bardziej ogólnymi badaniami nad strukturami danych prowadzonymi przez

K. E. Iversona i L. R. Johnsona [IBM Corp. research reports RC-390, RC-603, 1961; zobacz też Iverson, *A Programming Language* (New York: Wiley, 1962), rozdział 3 oraz G. Salton, *CACM* 5 (1962), 103–114].

Pomysł drzew *sfastrygowanych* pochodzi od A. J. Perlisa i C. Thorntona, *CACM* 3 (1960), 195–204. W tym artykule wprowadzono też ważne pojęcie przechodzenia drzewa w różnych porządkach oraz podano liczne przykłady algorytmów wykonujących przekształcenia algebraiczne. Niestety, ten ważny artykuł był przygotowywany w pośpiechu, przez co zawiera wiele błędów. Listy sfastrygowane Perlisa i Thorntoną były w naszej terminologii tylko „drzewami z fastrygą prawostronną”. Drzewa binarne sfastrygowane w *obu* kierunkach były niezależnie odkryte przez A. W. Holta, *A Mathematical and Applied Investigation of Tree Structures* (Thesis, U. of Pennsylvania, 1963). Porządki postorder i preorder były przez Z. Pawlaka nazywane odpowiednio porządkiem zwykłym wzdłużnym (*normal along order*) i dualnym wzdłużnym (*dual along order*) [*Colloquium on the Foundation of Mathematics*, Tihany, 1962 (Budapest: Akadémiai Kiadó, 1965), 227–238]. W cytowanej wyżej pracy Iverson i Johnson porządek preorder nazywali „porządkiem poddrzew”. Graficzne przedstawienia kształtów drzew i odpowiadająca im notacja liniowa zostały opisane przez A. G. Oettingera, *Proc. Harvard Symp. on Digital Computers and their Applications* (April 1961), 203–224. Reprezentacja drzew w porządku preorder ze stopniami wraz z algorytmem pokazującym jej związek z notacją dziesiętną Dewey'a i inne własności drzew zostały opisane przez S. Gorna, *Proc. Symp. Math. Theory of Automata* (Brooklyn: Poly. Inst., 1962), 223–240.

Historia drzew jako obiektów matematycznych wraz z bibliografią na ten temat jest opisana w podpunkcie 2.3.4.6.

Kiedy w 1966 roku powstawał ten rozdział, największy wkład w popularyzację wiedzy o strukturach danych miały systemy przetwarzania list, które w historii informatyki odegrały bardzo ważną rolę. Pierwszym z tych systemów stosowanym na szeroką skalę był IPL-V (potomek IPL-III, opracowany w 1959 roku). IPL-V był interpreterem języka niskiego poziomu operującego na Listach. Mniej więcej w tym samym czasie H. Gelerter z zespołem opracował FLPL (zbiór podprogramów w FORTRAN-ie przeznaczonych do posługiwania się Listami, inspirowany systemem IPL, ale wykorzystujący wywołania podprogramów zamiast rozkazów interpretera). Trzeci system, LISP, został zaprojektowany przez J. McCarthy'ego, też w 1959 roku. LISP różnił się nieco od swoich poprzedników: programy w LISP-ie były (i wciąż są) wyrażane za pomocą matematycznej notacji funkcyjnej połączonej z „wyrażeniami warunkowymi”, a dopiero później tłumaczone na Listy. Wiele systemów przetwarzania List powstało w latach sześćdziesiątych. Najważniejszym z historycznego punktu widzenia okazał się SLIP opracowany przez J. Weizenbauma, będący zbiorem podprogramów w FORTRAN-ie do obsługi list dwukierunkowych.

Artykuł Bobrowa i Raphaela, *CACM* 7 (1964), 231–240, może służyć za krótkie wprowadzenie do języków IPL-V, LISP i SLIP. Wyśmienite wprowadzenie do języka LISP zostało przedstawione w: P. M. Woodward, D. P. Jenkins, *Comp. J.* 4 (1961), 47–53. Zobacz także rozważania autorów na temat ich

własnego systemu zawarte w: A. Newell, F. M. Tonge, „An introduction to IPL-V”, *CACM* **3** (1960), 205–211; H. Gelernter, J. R. Hansen, C. L. Gerberich, „A FORTRAN-compiled List Processing Language”, *JACM* **7** (1960), 87–101; John McCarthy, „Recursive functions of symbolic expressions and their computation by machine, I”, *CACM* **3** (1960), 184–195; J. Weizenbaum, „Symmetric List Processor”, *CACM* **6** (1963), 524–544. Artykuł Weizenbauma zawiera pełny opis wszystkich algorytmów wykorzystanych w systemie SLIP. Ze wszystkich tych systemów tylko LISP przetrwał kolejne dziesięciolecia rozwoju. McCarthy opisał początki języka LISP w *History of Programming Languages* (Academic Press, 1981), 173–197.

W latach sześćdziesiątych pojawiło się też kilka systemów przetwarzania napisów (*string manipulation*) przeznaczonych głównie do wykonywania operacji na ciągach informacji tekstowej o zmiennej długości, jak wyszukiwanie podciągów, zastępowanie jednych podciągów innymi itp. Z historycznego punktu widzenia najważniejszymi z nich były COMIT [V. H. Yngve, *CACM* **6** (1963), 83–84] i SNOBOL [D. J. Farber, R. E. Griswold, I. P. Polonsky, *JACM* **11** (1964), 21–30]. Systemy przetwarzania napisów były używane powszechnie. Składały się na nie głównie algorytmy podobne do zaprezentowanych w tym rozdziale, ale w historii struktur danych ich rola była znikoma. Użytkownicy takich systemów byli odizolowani od szczegółów konstrukcji systemu. Przegląd pierwszych metod przetwarzania napisów można znaleźć w: S. E. Madnick, *CACM* **10** (1967), 420–424.

Systemy IPL-V i FLPL nie korzystały ani z odśmiecania, ani z techniki licznika dowiązań. Gdy wiele List miało wspólną pod-Listę, pod-Lista ta była „własnością” jednej z nich, a inne ją „pożyczalały”. Pod-Lista była usuwana razem z listą, której była własnością. Na programiście spoczywała zatem obowiązek pilnowania, by nie usunąć listy, którą „pożyczałały” inne istniejące listy. Metoda licznika dowiązań dla List została wprowadzona przez G. E. Collinsa, *CACM* **3** (1960), 655–657, i wyjaśniona szerzej w *CACM* **9** (1966), 578–588. Odśmiecanie zostało po raz pierwszy opisane w artykule McCarthy’ego z 1960 roku; zobacz także uwagi Weizenbauma w *CACM* **7** (1964), 38, oraz Cohen, Trilling, *BIT* **7** (1967), 22–30.

Rosnąca popularność technik wskaźnikowych spowodowała wprowadzenie związanych z nimi mechanizmów do algebraicznych języków programowania zaprojektowanych po 1965 roku. Nowe języki umożliwiały programiście wybranie odpowiedniej reprezentacji struktury danych bez uciekania się do języka maszynowego, a także bez start związanych z wykorzystaniem zbyt ogólnych metod związanych z Listami. Do podstawowych osiągnięć w tej dziedzinie zaliczyć należy prace takich badaczy, jak: N. Wirth i H. Weber [*CACM* **9** (1966), 13–23, 25, 89–99]; H. W. Lawson [*CACM* **10** (1967), 358–367]; C. A. R. Hoare [*Symbol Manipulation Languages and Techniques*, red. D. G. Bobrow (Amsterdam: North-Holland, 1968), 262–284]; O.-J. Dahl, K. Nygaard [*CACM* **9** (1966), 671–678]; A. van Wijngaarden, B. J. Mailloux, J. E. L. Peck, C. H. A. Koster [*Numerische Math.* **14** (1969), 79–218]; Dennis M. Ritchie [*History of Programming Languages—II* (ACM Press, 1996), 671–698].

Algorytmy dynamicznego przydziału pamięci były używane na kilka lat przed ukazaniem się pierwszej publikacji na ten temat. Bardzo czytelne omówienie przygotował i opublikował w 1961 roku W. T. Comfort [CACM 7 (1964), 357–362]. Metoda znaczników brzegowych, opisana w podrozdziale 2.5, została zaprojektowana przez autora w 1962 roku na użytek systemu operacyjnego komputera Burroughs B5000. System bloków bliźniaczych po raz pierwszy zastosował H. Markowitz w systemie SIMSCRIPT w 1963 roku, a niezależnie odkrył go i opisał K. Knowlton, CACM 8 (1965), 623–625; zobacz także CACM 9 (1966), 616–625. Omówienia metod dynamicznego przydziału pamięci można znaleźć w: Iliffe, Jodeit, Comp. J. 5 (1962), 200–209; Bailey, Barnett, Burleson, CACM 7 (1964), 339–346; A. T. Berztiss, CACM 8 (1965), 512–513; D. T. Ross, CACM 10 (1967), 481–492.

Ogólne omówienie struktur danych i ich związków z programowaniem sporządziła Mary d'Imperio, „Data Structures and their Representation in Storage”, Annual Review in Automatic Programming 5 (Oxford: Pergamon Press, 1969). Jej artykuł zawiera szczegółową analizę struktur wykorzystanych w dwunastu systemach przetwarzania List i napisów. Więcej szczegółów historycznych można znaleźć w: CACM 3 (1960), 183–234 i CACM 9 (1966), 567–643. (Podane odniesienia to materiały z sympozjów. Niektóre z zawartych w nich prac były cytowane wcześniej).

Wyśmienita, skomentowana bibliografia wczesnych prac na temat obliczeń symbolicznych i przekształceń wzorów algebraicznych została zestawiona przez Jeana E. Sammeta; zobacz Computing Reviews 7 (July–August 1966), B1–B31.

W rozdziale 2 zapoznaliśmy się szczegółowo z pewnym rodzajem struktur danych. Nie jest złym pomysłem krótkie podsumowanie przedstawionych zagadnień i przyjrzenie się im w szerszej perspektywie (by ujrzeć cały las, a nie tylko pojedyncze drzewa). Wyszliśmy od pojęcia *elementu* jako podstawowej cegiełki struktury danych. Przyjrzaliśmy się wielu przykładom, pokazującym jak można reprezentować związki strukturalne między elementami danych w sposób niejawny (za pomocą względnego porządku ułożenia elementów w pamięci) lub jawny (za pomocą dowiązań do innych elementów). Ilość informacji strukturalnych, które trzeba przechowywać w pamięci, zależy od operacji, jakie chcemy wykonywać na strukturze danych.

Ze względów pedagogicznych skupiliśmy się na związkach między strukturami danych a ich reprezentacjami maszynowymi, zamiast omawiać zagadnienia te oddzielnie. Jednak aby lepiej zrozumieć naturę struktur danych, można przyjrzeć się im z bardziej abstrakcyjnego punktu widzenia. Opracowano kilka metod opartych na tym założeniu; warto sięgnąć po zmuszające do myślenia artykuły: G. Mealy, „Another look at data”, Proc. AFIPS Fall Joint Computer Conf. 31 (1967), 525–534; J. Earley, „Toward an understanding of data structures”, CACM 14 (1971), 617–627; C. A. R. Hoare, „Notes on data structuring” w O.-J. Dahl, E. W. Dijkstra, C. A. R. Hoare, Structured Programming (Academic Press, 1972), 83–174; Robert W. Engles, „A tutorial on data-base organization”, Annual Review in Automatic Programming 7 (1972), 3–63.

Nasze omówienie nie obejmuje całokształtu zagadnień związanych ze strukturami danych; nie podjęliśmy przynajmniej trzech ważnych tematów:

a) Często zachodzi konieczność wyszukania w strukturze danych elementu (elementów) mającego określoną wartość; potrzeba dysponowania taką operacją ma zazwyczaj istotny wpływ na budowę struktury danych. Zajmujemy się tym problemem w rozdziale 6.

b) Zajmowaliśmy się głównie wewnętrzna reprezentacją struktur danych w pamięci komputera. Ale to oczywiście tylko jedna strona medalu, ponieważ trzeba dysponować metodami zewnętrznej reprezentacji struktur, by można było je wprowadzać do i wyprowadzać z pamięci. W prostych przypadkach reprezentacja zewnętrzna może być taka sama jak reprezentacja wewnętrzna, ale zagadnienia związane z zamianą ciągów znaków na złożone struktury są również istotne. Analizujemy je w rozdziałach 9 i 10.

c) Omawialiśmy zasadniczo reprezentację struktur danych w szybkiej pamięci o dostępie swobodnym. Gdy korzystamy z wolniejszych urządzeń, takich jak dyski lub taśmy, okazuje się, że wszystkie problemy dotyczące struktur danych stają się jeszcze ważniejsze – dysponowanie wydajnymi algorytmami i dobrymi reprezentacjami jest sprawą podstawową. W takich przypadkach dowiązania powinny wskazywać na blisko położone elementy. Zazwyczaj tego rodzaju problemy silnie zależą od cech konkretnej maszyny, zatem omówienie trudno uogólniać. Proste przykłady zaprezentowane w rozdziale 2 powinny przygotować Czytelnika do rozwiązywania bardziej skomplikowanych problemów związanych z obsługą mniej wyidealizowanych pamięci. Niektóre z takich problemów są szczegółowo omówione w rozdziałach 5 i 6.

Jakie wnioski płyną z rozdziału 2? Zapewne najważniejszy z nich mówi, iż koncepcje, którymi się zajmowaliśmy, nie dotyczą wyłącznie programowania komputerów, ale pojawiają się także w życiu codziennym. Zbiór elementów zawierających dowiązania do innych elementów to bardzo dobry abstrakcyjny model różnego rodzaju związków strukturalnych. Ten model pokazuje, w jaki sposób można zbudować złożone struktury ze struktur prostszych. Przekonaliśmy się także, że algorytmy przekształcające takie struktury można zaprojektować w sposób naturalny.

Wydaje się zatem, że warto rozwinać teorię dotyczącą zbiorów elementów połączonych dowiązaniami. Prawdopodobnie najbardziej oczywistym sposobem zbudowania takiej teorii jest zdefiniowanie maszyny abstrakcyjnej lub „automatu” działającego na strukturach z dowiązaniami. Można podać na przykład następującą nieformalną definicję: mamy liczby k, l, r, s ; automat przetwarza elementy zawierające k pól z dowiązaniami i r pól informacyjnych; automat ma l rejestrów do przechowywania dowiązań (wskaźnikowych) i s rejestrów informacyjnych umożliwiających sterowanie jego pracą. Pola i rejesty informacyjne mogą zawierać dowolne symbole z ustalonego zbioru symboli informacyjnych; każde pole lub rejestr wskaźnikowy zawiera albo Λ , albo wskaźnik do elementu. Maszyna może: (i) tworzyć nowe elementy (wstawiać dowiązanie do nowego elementu do rejestru), (ii) przeprowadzać test na równość symboli

informacyjnych lub wartości dowiązań, (iii) kopiować symbole informacyjne lub dowiązania między rejestrami i elementami. Odpowiednie ograniczenia dotyczące zachowania takiej maszyny dadzą automaty należące do różnych znanych klas.

Podobny model obliczeniowy zaproponował A. N. Kołmogorow już w 1952 roku. Jego maszyna działa na grafie G o wyróżnionym wierzchołku początkowym v_0 . Akcja maszyny w każdym kroku zależy wyłącznie od podgrafa G' składającego się z tych wierzchołków G , których odległość od v_0 jest $\leq n$, i polega na zastąpieniu G' przez graf $G'' = f(G')$, gdzie G'' zawiera v_0 oraz wierzchołki oddalone od v_0 dokładnie o n (a poza tym być może inne nowe wierzchołki). Reszta grafu G pozostaje nie zmieniona. Wartość n jest ustalona z góry dla każdego algorytmu, ale może być dowolnie duża. Z każdym wierzchołkiem związany jest symbol ze skończonego alfabetu i wymaga się, by żaden wierzchołek nie miał dwóch różnych sąsiadów o tym samym symbolu. (Zobacz A. N. Kołmogorow, *Uspiechi Mat. Nauk* 8, 4 (1953), 175–176; Kołmogorow, Uspienski, *Uspiechi Mat. Nauk* 13, 4 (1958), 3–28; Amer. Math. Soc. Translations, series 2, 29 (1963), 217–245).

Automat z dowiązaniami może symulować maszynę grafową, wykonując co najwyżej ograniczoną liczbę kroków dla każdego kroku maszyny grafowej. Wydaje się jednak, że maszyna grafowa działająca na grafach o ograniczonych stopniach elementów nie może symulować automatu z dowiązaniami bez istotnego wzrostu czasu (tzn. większego niż o czynnik stały), jeżeli w definicji nie weźmiemy grafów skierowanych. Model z dowiązaniami jest oczywiście bliski operacjom dostępnym na rzeczywistych maszynach, natomiast model grafowy nie.

Jeden z najciekawszych, nieroziwiązanych problemów dotyczących tych modeli polega na zbadaniu, jak szybko lub przy jakiej liczbie wierzchołków można za ich pomocą rozwiązać pewne problemy (na przykład tłumaczenie pewnych języków formalnych). Kiedy powstawał ten rozdział, kilka ciekawych problemów tego rodzaju zostało rozwiązanych (znaczący wkład wnieśli J. Hartmanis i R. E. Stearns), ale tylko dla szczególnych klas maszyn Turinga o wielu taśmach i głowicach zapisującco-odczytujących. Maszyna Turinga jest modelem względnie mało realistycznym, stąd powyższe wyniki mają raczej mało wspólnego z problemami praktycznymi.

Z uwagi na fakt, że liczba n elementów utworzonych przez automat może dążyć od nieskończoności, nie wiemy, w jaki sposób zbudować urządzenie fizyczne zachowujące się jak automat z dowiązaniami, ponieważ chcielibyśmy, żeby operacje automatu wykonywane były w takim samym czasie niezależnie od wartości n . Jeżeli dowiązania są adresami w pamięci komputera, to należy ograniczyć liczbę wierzchołków, ponieważ pola z dowiązaniami mają ustalony rozmiar. Wielotaśmowa maszyna Turinga jest zatem modelem bardziej realistycznym dla dużych n . Wydaje się mimo wszystko, że automat z dowiązaniami stanowi lepszą podstawę teorii złożoności algorytmów niż maszyna Turinga, nawet dla wzorów asymptotycznych dla dużych n , ponieważ wyniki teorii będą ważniejsze dla realnych wartości n . Jeśli n jest większe niż 10^{30} , to nawet maszyna Turinga przestaje być realistyczna – takiego urządzenia nie da się zbudować.

Minęło wiele lat od kiedy autor napisał większość powyższych uwag. Należy się cieszyć z istotnego postępu, jaki dokonał się od tamtej pory w dziedzinie automatów z dowiązaniami. Jednak wciąż dużo pozostaje do zrobienia.

*Opracowano ogólne zasady programowania.
Z większości z nich przez długi czas korzystano
na dworcu przeładunkowym w Kansas City.*
— DERRICK LEHMER (1949)

*Jestem pewny, że się ze mną zgodzisz ... iż jeśli
strona 534 znajduje się dopiero w drugim rozdziale,
to pierwszy musi być nieznośnie długi.*
— SHERLOCK HOLMES, w *The Valley of Fear* (1888)

ODPOWIEDZI DO ĆWICZEŃ

I am not bound to please thee with my answers.

— Shylock, w *Kupiec Wenecki* (akt IV, scena 1, wiersz 65)

UWAGI DO ĆWICZEŃ

1. Średni problem dla Czytelnika zainteresowanego matematyką.
4. Zobacz W. J. LeVeque, *Topics in Number Theory 2* (Reading, Mass.: Addison – Wesley, 1956), rozdział 3; P. Ribenboim, *13 Lectures on Fermat's Last Theorem* (New York: Springer-Verlag, 1979); A. Wiles, *Annals of Mathematics* **141** (1995), 443–551.

1.1

1. $t \leftarrow a, a \leftarrow b, b \leftarrow c, c \leftarrow d, d \leftarrow t.$
2. Po pierwszym wykonaniu wartości zmiennych m i n są odpowiednio poprzednimi wartościami n i r , przy tym $n > r$.
3. **Algorytm F** (*Algorytm Euklidesa*). Dane są dwie dodatnie liczby całkowite m i n , znaleźć ich największy wspólny dzielnik.

F1. [Reszta z dzielenia m/n] Podziel m przez n i niech m będzie resztą z tego dzielenia.

F2. [Czy równe zero?] Jeśli $m = 0$, to zakończ wykonanie algorytmu z odpowiedzią n .

F3. [Reszta z dzielenia n/m] Podziel n przez m i niech n będzie resztą z tego dzielenia.

F4. [Czy równe zero?] Jeśli $n = 0$, to zakończ algorytm z odpowiedzią m ; w przeciwnym przypadku wróć do kroku F1.

4. Według algorytmu E, $n = 6099, 2166, 1767, 399, 171, 57$. Odpowiedź: 57
5. Procedura nie jest ani skończona, ani dobrze zdefiniowana, ani efektywna, być może brak wyjścia. Jeśli chodzi o format: numery kroków nie są poprzedzone literą, nie ma podsumowań w nawiasach kwadratowych, brak znaku „|”.
6. Uruchamiając algorytm E dla $n = 5$ i $m = 1, 2, 3, 4, 5$, odkrywamy, że krok E1 jest wykonywany odpowiednio 2, 3, 4, 3, 1 raz(y). Stąd średnia wynosi $2.6 = T_5$.
7. W prawie wszystkich przypadkach (we wszystkich poza skończoną liczbą) $n > m$. A skoro $n > m$, pierwsza iteracja algorytmu E po prostu zamienia te wartości miejscami. Zatem $U_m = T_m + 1$.

8. Niech $A = \{a, b, c\}$, $N = 5$. Wykonanie algorytmu zostanie wtedy zakończone z wynikiem $a^{\gcd(m,n)}$.

j	θ_j	ϕ_j	b_j	a_j	
0	ab	(pusty)	1	2	Usuń jedno a i jedno b , albo przejdź do 2.
1	(pusty)	c	0	0	Dopisz c na skrajnie lewej pozycji, wróć do 0.
2	a	b	2	3	Zamień wszystkie a na b .
3	c	a	3	4	Zamień wszystkie c na a .
4	b	b	0	5	Jeśli pozostały b , powtóż.

9. Możemy powiedzieć, że C_2 jest reprezentacją C_1 , jeżeli istnieją: funkcja g z I_1 w I_2 , funkcja h z Q_2 w Q_1 przeprowadzająca Ω_2 w Ω_1 i funkcja j z Q_2 o wartościach całkowitych dodatnich, spełniające następujące warunki:

- a) Jeśli s należy do I_1 , to wynikiem działania C_1 na y jest x wtedy i tylko wtedy, gdy istnieje y' z Ω_2 , dla którego wynikiem działania C_2 na $g(x)$ jest y' i $h(y') = y$.
- b) Jeśli q należy do Q_2 , to $f_1(h(q)) = h(f_2^{[j(q)]}(q))$, gdzie $f_2^{[j(q)]}$ oznacza funkcję f_2 iterowaną (złożoną z samą sobą) $j(q)$ -krotnie.

Na przykład, niech metoda C_1 będzie taka, jak w definicji (2), i niech w metodzie C_2 będzie $I_2 = \{(m, n)\}$, $\Omega_2 = \{(m, n, d)\}$, $Q_2 = I_2 \cup \Omega_2 \cup \{(m, n, a, b, 1)\} \cup \{(m, n, a, b, r, 2)\} \cup \{(m, n, a, b, r, 3)\} \cup \{(m, n, a, b, r, 4)\} \cup \{(m, n, a, b, 5)\}$. Niech $f_2((m, n)) = (m, n, m, n, 1)$; $f_2((m, n, d)) = (m, n, d)$; $f_2((m, n, a, b, 1)) = (m, n, a, b, a \bmod b, 2)$; $f_2((m, n, a, b, r, 2)) = (m, n, b)$ dla $r = 0$, w pozostałych przypadkach $(m, n, a, b, r, 3)$; $f_2((m, n, a, b, r, 3)) = (m, n, b, b, r, 4)$; $f_2((m, n, a, b, r, 4)) = (m, n, a, r, 5)$; $f_2((m, n, a, b, 5)) = f_2((m, n, a, b, 1))$.

Niech teraz $h((m, n)) = g((m, n)) = (m, n)$; $h((m, n, d)) = (d)$; $h((m, n, a, b, 1)) = (a, b, 0, 1)$; $h((m, n, a, b, r, 2)) = (a, b, r, 2)$; $h((m, n, a, b, r, 3)) = (a, b, r, 3)$; $h((m, n, a, b, r, 4)) = h(f_2((m, n, a, b, r, 4)))$; $h((m, n, a, b, 5)) = (a, b, b, 1)$; $j((m, n, a, b, r, 3)) = j((m, n, a, b, r, 4)) = 2$, w przeciwnym przypadku $j(q) = 1$. Wówczas C_2 reprezentuje C_1 .

Uwagi: Trudno oprzeć się pokusie napisania prostszych definicji – ot choćby tak: ustalamy, że g odwzorowuje Q_1 w Q_2 , oraz wymagamy, by dla ciągu obliczeń x_0, x_1, \dots metody C_1 ciąg $g(x_0), g(x_1), \dots$ był podciągiem ciągu obliczenia metody C_2 rozpoczęjącego się od $g(x_0)$. To podejście jest jednak nieodpowiednie. W powyższym przykładzie C_1 gubi początkowe wartości m i n , a C_2 nie.

Jeśli C_2 reprezentuje C_1 przez funkcje g, h, j a C_3 reprezentuje C_2 przez funkcje g', h', j' , to C_3 reprezentuje C_1 przez funkcje g'', h'', j'' , gdzie:

$$g''(x) = g'(g(x)), \quad h''(x) = h(h'(x)) \quad \text{oraz} \quad j''(q) = \sum_{0 \leq k < j(h'(q))} j'(q_k),$$

gdzie $q_0 = q$ i $q_{k+1} = f_3^{[j'(q_k)]}(q_k)$. Z tego powodu zdefiniowana powyżej relacja jest przechodnia. Mówimy, że metoda C_2 bezpośrednio reprezentuje C_1 , jeśli funkcja j jest ograniczona; ta relacja też jest przechodnia. Relacja „ C_2 reprezentuje C_1 ” generuje relację równoważności, w której dwie metody obliczeniowe są równoważne wtedy i tylko wtedy, gdy obliczają izomorficzne funkcje swoich wejść; relacja „ C_2 dokładnie reprezentuje C_1 ” generuje bardziej interesującą relację równoważności, która lepiej pasuje do intuicyjnego pojęcia „bycia w zasadzie tym samym algorytmem”.

Alternatywne podejście do symulacji zaprezentowano w: R. W. Floyd, R. Beigel, *The Language of Machines* (Computer Science Press, 1994), podrozdział 3.3.

1.2.1

1. (a) Udowodnij $P(0)$. (b) Udowodnij, że $P(0), \dots, P(n)$ pociąga za sobą $P(n+1)$ dla wszystkich $n \geq 0$.

2. Twierdzenie nie zostało udowodnione dla $n = 2$; w drugiej części dowodu przyjmijmy $n = 1$; zakładamy, że $a^{-1} = 1$. Jeśli ten warunek jest prawdziwy (a więc $a = 1$), twierdzenie rzeczywiście zachodzi.

3. Poprawną odpowiedzią jest $1 - 1/n$. Błąd pojawia się w dowodzie dla $n = 1$, gdzie lewą stronę równości można uznać za źle określona lub zakładać, że wynosi zero (bo stoi tam $n - 1$ składników).

5. Jeśli n jest liczbą pierwszą, to oczywiście jest iloczynem jednej lub więcej liczb pierwszych. W pozostałych przypadkach n ma właściwy dzielnik, więc $n = km$ dla pewnych k i m , gdzie $1 < k < n$, $1 < m < n$. Ponieważ zarówno k , jak i m są mniejsze od n , na mocy założenia indukcyjnego mogą być zapisane jako iloczyny liczb pierwszych; stąd n jest iloczynem liczb pierwszych występujących w reprezentacjach k i m .

6. Przyjmując oznaczenia z rysunku 4, dowodzimy, że $A5$ implikuje $A6$. To jest jasne, bo $A5$ pociąga za sobą $(a' - qa)m + (b' - qb)n = (a'm + b'n) - a(am + bn) = c - qd = r$.

$$\mathbf{7. } n^2 - (n-1)^2 + \cdots - (-1)^n 1^2 = 1 + 2 + \cdots + n = n(n+1)/2.$$

8. (a) Pokażemy, że suma $(n^2 - n + 1) + (n^2 - n + 3) + \cdots + (n^2 + n - 1)$ wynosi n^3 . Ta suma zapisuje się jako $(1 + 3 + \cdots + (n^2 + n - 1)) - (1 + 3 + \cdots + (n^2 - n - 1)) = ((n^2 + n)/2)^2 - ((n^2 - n)/2)^2 = n^3$. Skorzystaliśmy z równania (2). Proszono nas jednak o dowód indukcyjny, trzeba zatem inaczej zabrać się do rzeczy! Dla $n = 1$ wynik jest oczywisty. Założymy $n \geq 1$; $(n+1)^2 - (n+1) = n^2 - n + 2n$, zatem każdy z początkowych składników sumy dla $n+1$ jest o $2n$ większy. Stąd suma dla $n+1$ wynosi tyle co suma dla n plus

$$\underbrace{2n + \cdots + 2n}_{n \text{ razy}} + (n+1)^2 + (n+1) - 1;$$

a to równa się $n^3 + 2n^2 + n^2 + 3n + 1 = (n+1)^3$. (b) Pokazaliśmy, że pierwszy składnik sumy dla $(n+1)^3$ jest o dwa większe od ostatniego składnika dla n^3 . Stąd na mocy równania (2), $1^3 + 2^3 + \cdots + n^3 = \text{suma kolejnych liczb nieparzystych poczynając od jedynki} = (\text{liczba składników})^2 = (1 + 2 + \cdots + n)^2$.

10. Oczywiste dla $n = 10$. Dla $n \geq 10$ mamy $2^{n+1} = 2 \cdot 2^n > (1 + 1/n)^3 2^n$, a przez indukcję to jest większe niż $(1 + 1/n)^3 n^3 = (n+1)^3$.

$$\mathbf{11. } (-1)^n(n+1)/(4(n+1)^2 + 1).$$

12. Jednym punktem, gdzie należy nietrywialnie zmodyfikować algorytm jest obliczenie liczby q w E2. Można to zrobić metodą sukcesywnego odejmowania i sprawdzania, czy $u + v\sqrt{2}$ jest dodatnie, ujemne czy równe zero.

Łatwo pokazać, że $u + v\sqrt{2} = u' + v'\sqrt{2}$ wtedy i tylko wtedy, gdy $u = u'$ i $v = v'$, bo $\sqrt{2}$ jest liczbą niewymierną. Wynika z tego jasno, że 1 i $\sqrt{2}$ nie mają wspólnego dzielnika, jeśli zdefiniujemy dzielnik tak, że $u + v\sqrt{2}$ dzieli $a(u + v\sqrt{2})$ wtedy i tylko wtedy, gdy a jest całkowite. Algorytm zmodyfikowany w powyższy sposób oblicza po prostu właściwy ułamek łańcuchowy reprezentujący stosunek liczb podanych na wejściu; zobacz punkt 4.5.3.

(Uwaga: Jeśli jednak rozszerzymy pojęcie dzielnika, mówiąc, że $u + v\sqrt{2}$ dzieli $a(u + v\sqrt{2})$ wtedy i tylko wtedy, kiedy a ma postać $u' + v'\sqrt{2}$ dla całkowitych u' i v' , to istnieje sposób takiego zmodyfikowania algorytmu E, żeby jego wykonanie zawsze się zakończyło. Jeśli w kroku E2 mamy $c = u + v\sqrt{2}$ i $d = u' + v'\sqrt{2}$, obliczamy

$c/d = c(u' - v'\sqrt{2})/(y'^2 - 2v'^2) = x + y\sqrt{2}$, gdzie x i y są wymierne. Niech teraz $q = u'' + v''\sqrt{2}$, gdzie u'' i v'' są liczbami całkowitymi najbliższymi x i y . Niech $r = c - qd$. Jeśli $r = u''' + v'''\sqrt{2}$, to $|u'''^2 - 2v''''^2| < |u'^2 - 2v'^2|$, zatem obliczenie się zakończy. Więcej szczegółów znajdziesz Czytelnik pod hasłem „kwadratowe dziedziny euklidesowe” w podręcznikach teorii liczb).

13. Dołącz „ $T \leq 3(n-d) + k$ ” do asercji A_3, A_4, A_5, A_6 dla k odpowiednio równego 2, 3, 3, 1. Dołącz także „ $d > 0$ ” do A_4 .

15. (a) Niech $A = S$ z punktu (iii); każdy niepusty zbiór dobrze uporządkowany ma element najmniejszy.

(b) Niech $x \prec y$, gdy $|x| < |y|$ lub gdy $|x| = |y|$ i $x < 0 < y$.

(c) Nie, podzbiór złożony ze wszystkich dodatnich liczb rzeczywistych nie spełnia warunku (iii). [Uwaga: Używając tzw. pewnika wyboru, można w pewien skomplikowany sposób udowodnić, że każdy zbiór daje się w jakiś sposób dobrze uporządkować; do tej pory nikomu nie udało się jednak pokazać, jak konkretnie wygląda relacja dobrze porządkująca zbiór liczb rzeczywistych].

(d) Udowodnij (iii) dla T_n , korzystając z indukcji względem n . Niech A będzie niepustym podzbiorem T_n . Rozważmy A_1 – zbiór elementów występujących jako pierwsza składowa krotek ze zbioru A . A_1 jest niepustym podzbiorem S , a zbiór S jest dobrze uporządkowany, zatem A_1 ma element najmniejszy x . Rozważmy teraz A_x – zbiór tych krotek z A , które na pierwszej pozycji mają x . A_x można utożsamiać z pewnym podzbiorem T_{n-1} , gdy pominiemy pierwszą składową krotek. Stąd na mocy indukcji A_x ma element najmniejszy (x, x_2, \dots, x_n) , który jest także najmniejszym elementem A .

(e) Nie, chociaż warunki (i) i (ii) zachodzą. Jeżeli S ma przynajmniej dwa różne elementy, $a \prec b$, to zbiór $(b), (a, b), (a, a, b), (a, a, a, b), \dots$ nie ma elementu najmniejszego. Z drugiej strony T można dobrze uporządkować, jeśli tak zdefinujemy relację \prec , by $(x_1, \dots, x_m) \prec (y_1, \dots, y_n)$ zachodziło wtedy i tylko wtedy, gdy $m < n$ lub $m = n$ i $(x_1, \dots, x_n) \prec (y_1, \dots, y_n)$ w T_n .

(f) Niech S będzie zbiorem dobrze uporządkowanym przez relację \prec . Jeśli taki nieskończony ciąg istnieje, to zbiór A składający się z elementów tego ciągu nie spełnia warunku (iii), bo nie może mieć elementu najmniejszego. W drugą stronę: jeśli \prec jest relacją spełniającą (i) i (ii), ale nie (iii), to weźmy A będący niepustym podzbiorem S , który nie ma elementu najmniejszego. Ponieważ A jest niepusty, to możemy w nim znaleźć x_1 ; x_1 nie jest najmniejszym elementem A , zatem możemy znaleźć takie x_2 , że $x_2 \prec x_1$; x_2 też nie jest najmniejszym elementem, możemy więc znaleźć takie x_3 , że $x_3 \prec x_2$ itd.

(g) Niech A będzie zbiorem wszystkich x , dla których $P(x)$ jest fałszywe. Jeśli A jest niepusty, to zawiera element najmniejszy x_0 . Stąd $P(y)$ jest prawdziwe dla wszystkich $y \prec x_0$. Ale wówczas z założenia $P(x_0)$ jest prawdziwe, zatem x_0 nie należy do A (sprzeczność). Stąd A musi być pusty: $P(x)$ jest zawsze prawdziwe.

1.2.2

1. Nie ma takiej liczby. Jeśli r jest dodatnią liczbą wymierną, to liczba $r/2$ jest mniejsza.

2. Nie, jeśli wielokropek oznacza nieskończenie wiele dziewiątek. W takim przypadku zgodnie z równością (2) rozwinięciem dziesiętnym liczby jest $1 + 0.24000000\dots$

3. $-1/27$, ale to nie wynika z podanej definicji.

4. 4.

6. Każda liczba rzeczywista ma jednoznaczne rozwinięcie dziesiętne, zatem $x = y$ wtedy i tylko wtedy, gdy $m = n$ i $d_i = e_i$ dla wszystkich $i \geq 1$. Jeśli $x \neq y$, to można porównywać m z n , d_1 z e_1 , d_2 z e_2 itd; pojawienie się pierwszej nierówności informuje, że strona o większej wartości pochodzi z większej liczby.

7. W skrócie: można użyć indukcji względem x , dowodząc najpierw dla dodatnich, a potem dla ujemnych x .

8. Wykonując algorytm dla $n = 0, 1, 2, \dots$, znajdujemy wartość n , dla której $n^m \leq u < (n+1)^m$. Przy założeniu indukcyjnym, że n, d_1, \dots, d_{k-1} zostały wyznaczone, d_k jest taką cyfrą, że:

$$\left(n + \frac{d_1}{10} + \cdots + \frac{d_k}{10^k} \right)^m \leq u < \left(n + \frac{d_1}{10} + \cdots + \frac{d_k}{10^k} + \frac{1}{10^k} \right)^m.$$

9. $((b^{p/q})^{u/v})^{qv} = (((b^{p/q})^{u/v})^v)^q = ((b^{p/q})^u)^q = ((b^{p/q})^q)^u = b^{pu}$, stąd $(b^{p/q})^{u/v} = b^{pu/qv}$. Mamy zatem udowodnione drugie prawo. Pierwsze udowodnimy za pomocą drugiego: $b^{p/q}b^{u/v} = (b^{1/qv})^{pv}(b^{1/qv})^{qu} = (b^{1/qv})^{pv+qu} = b^{p/q+u/v}$.

10. Jeśli $\log_{10} 2 = p/q$, gdzie p i q są dodatnie, to $2^q = 10^p$, co jest niedorzeczną, bo prawa strona dzieli się przez 5, a lewa nie.

11. Nieskończenie wielu! Nieważne, ile znamy cyfr dziesiętnych x , nie będziemy mieli pewności, czy $10^x = 1.99999\dots$, czy $10^x = 2.00000\dots$, jeśli cyfry liczby x zgadzają się z cyframi $\log_{10} 2$. Nie ma w tym nic tajemniczego ani paradoksalnego. Podobna sytuacja występuje przy dodawaniu, kiedy dodajemy $0.444444\dots$ do $0.555555\dots$

12. Są to jedyne wartości d_1, \dots, d_8 , które spełniają nierówność (7).

13. (a) Udowadniamy przez indukcję, że jeśli $y > 0$, to $1 + ny < (1 + y)^n$. Potem podstawiamy $y = x/n$ i wyciągamy pierwiastek stopnia n z obu stron równości. (b) $x = b - 1$, $n = 10^k$.

14. Podstawiamy $x = \log_b c$ w drugiej równości (5), potem obustronnie logarytmujemy.

15. Udowadniamy, przenosząc $\log_b y$ na drugą stronę równania i używając (11).

16. $\ln x / \ln 10$.

17. 5; 1; 1; 0; nieokreślone.

18. Fałsz. $\log_8 x = \lg x / \lg 8 = \frac{1}{3} \lg x$

19. Tak, bo $\lg n < (\log_{10} n) / 0.301 < 14 / 0.301 < 47$.

20. Są swoimi odwrotnościami.

21. $(\ln \ln x - \ln \ln b) / \ln b$.

22. Zgodnie z tabelą w dodatku A $\lg x \approx 1.442695 \ln x$; $\log_{10} x \approx 0.4342945 \ln x$. Błąd względny wynosi $\approx (1.442695 - 1.4342945) / 1.442695 \approx 0.582\%$.

23. Weźmy figurę jak na rysunku 6, ale o polu $\ln y$. Podzielimy jej wysokość przez x , a szerokość pomnożmy przez x – powierzchnia figury się nie zmieni, za to stanie się ona przystająca do fragmentu, który zostaje po zabraniu kawałka o polu $\ln x$ z figury o polu $\ln xy$, bo wysokość w punkcie $x + xt$ górnego brzegu figury o polu $\ln xy$ wynosi $1/(x+xt) = (1/(1+t))/x$.

24. Wstawić 2 wszędzie tam, gdzie występuje 10.

25. Zauważmy, że $z = 2^{-p} \lfloor 2^{p-k} x \rfloor > 0$, gdzie p jest dokładnością (liczbą cyfr dwójkowych po kropce). Wielkość $y + \log_b x$ pozostaje w przybliżeniu stała.

27. Udowadniamy przez indukcję względem k , że

$$x^{2^k} (1 - \delta)^{2^{k+1}-1} \leq 10^{2^k(n+b_1/2+\dots+b_k/2^k)} x'_k \leq x^{2^k} (1 + \epsilon)^{2^{k+1}-1}$$

i logarytmujemy obustronnie.

28. W następującym rozwiążaniu korzysta się z tej samej tablicy pomocniczej, co w poprzednim algorytmie.

E1. [Inicjowanie] Jeśli $1 - \epsilon$ jest największą możliwą wartością x , to przyjmij $y \leftarrow (\text{najlepsze przybliżenie } b^{\epsilon-1})$, $x \leftarrow 1 - x$, $k \leftarrow 1$. (Wielkość $y b^{-x}$ będzie w kolejnych krokach w przybliżeniu stała).

E2. [Czy koniec?] Jeśli $x = 0$, to zatrzymaj się.

E3. [Porównywanie] Jeśli $x < \log_b(2^k/(2^k-1))$, to zwiększ k o 1 i powtórz niniejszy krok.

E4. [Redukowanie] Przyjmij $x \leftarrow x - \log_b(2^k/(2^k-1))$, $y \leftarrow y - (y \text{ przesunięte w prawo o } k \text{ bitów})$ i przejdź do kroku E2. ■

Jeśli do y przypiszemy $b^{1-\epsilon}(1+\epsilon_0)$ w kroku E1, to kolejne błędy obliczeń powstają, gdy $x \leftarrow x + \log_n(1 - 2^{-k}) + \delta_j$ oraz $y \leftarrow y(1 - 2^{-k})(1 + \epsilon_j)$ podczas j -tego wykonania kroku E4, dla pewnych małych błędów δ_j i ϵ_j . Kiedy algorytm się kończy, mamy obliczone $y = b^{x - \sum \delta_j} \prod_j (1 + \epsilon_j)$. W bardziej szczegółowej analizie należy brać pod uwagę b oraz rozmiar słowa maszynowego. Zauważmy, że zarówno tu, jak i w ćwiczeniu 26 można nieco doszlifować oszacowania błędu, jeśli podstawą jest e , z uwagi na fakt, że dla wielu wartości k pozycję tablicy $\ln(2^k/(2^k-1))$ można podać z dużą dokładnością wynoszącą $2^{-k} + \frac{1}{2}2^{-2k} + \frac{1}{3}2^{-3k} + \dots$.

Uwaga: Podobny algorytm można podać dla funkcji trygonometrycznych. Zobacz J. E. Meggitt, *IBM J. Res. and Dev.* **6** (1962), 210–226; **7** (1963), 237–245. Zobacz także T. C. Chen, *IBM J. Res. and Dev.* **16** (1972), 380–388; V. S. Linsky, *Vychisl. Mat.* **2** (1957), 90–119; D. E. Knuth, *METAFONT: The Program* (Reading, Mass.: Addison-Wesley, 1986), §120–§147.

29. e ; 3; 4

1.2.3

1. $a_1 + a_2 + a_3$.

2. $\frac{1}{1} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \frac{1}{9} + \frac{1}{11}; \frac{1}{9} + \frac{1}{3} + \frac{1}{1} + \frac{1}{3} + \frac{1}{9}$.

3. Reguła dotycząca $p(j)$ nie jest spełniona. Po pierwsze, $n^2 = 3$ nie zachodzi dla żadnego n ; po drugie, $n^2 = 4$ zachodzi dla dwóch różnych n . [Zobacz (18)].

4. $(a_{11}) + (a_{21} + a_{22}) + (a_{31} + a_{32} + a_{33}) = (a_{11} + a_{21} + a_{31}) + (a_{22} + a_{32}) + (a_{33})$.

5. Wystarczy skorzystać z równości $a \sum_{R(i)} x_i = \sum_{R(i)} (ax_i)$:

$$\left(\sum_{R(i)} a_i \right) \left(\sum_{S(j)} b_j \right) = \sum_{R(i)} a_i \left(\sum_{S(j)} b_j \right) = \sum_{R(i)} \left(\sum_{S(j)} a_i b_j \right).$$

7. Użyj równości (3), zamień granice miejscami i przerzuć wyrazy pomiędzy a_0 i a_c spod jednej granicy pod drugą.

8. Niech $a_{(i+1)i} = +1$ i $a_{i(i+1)} = -1$ dla wszystkich $i \geq 0$, a wszystkie inne $a_{ij} = 0$. Niech $R(i) = S(i) = „i \geq 0”$. Lewa strona jest równa -1 , prawa strona jest równa $+1$.

9, 10. Nie; regułę (d) można wykorzystać jedynie pod warunkiem, że $n \geq 0$. (Wynik dla $n = -1$ jest poprawny, ale wyprowadzenie nie).

11. $(n+1)a$.

12. $\frac{7}{6}(1 - 1/7^{n+1})$.

13. $m(n-m+1) + \frac{1}{2}(n-m)(n-m+1)$, czyli $\frac{1}{2}(n(n+1) - m(m-1))$.

14. $\frac{1}{4}(n(n+1) - m(m-1))(s(s+1) - r(r-1))$, jeśli $m \leq n$ i $r \leq s$.

15, 16. Najważniejsze przekształcenia to:

$$\begin{aligned} \sum_{0 \leq j \leq n} jx^j &= x \sum_{1 \leq j \leq n} jx^{j-1} = x \sum_{0 \leq j \leq n-1} (j+1)x^j \\ &= x \sum_{0 \leq j \leq n} jx^j - nx^{n+1} + x \sum_{0 \leq j \leq n-1} x^j. \end{aligned}$$

17. Jest równa liczbie elementów zbioru S .

18. $S'(j) = „1 \leq j < n”$. $R'(i, j) = „n”$ jest wielokrotnością i oraz $i > j”$.

19. $a_n - a_{m-1}$.

20. $(b-1) \sum_{k=0}^n (n-k)b^k + n + 1 = \sum_{k=0}^n b^k$; ta równość wynika z (14) oraz odpowiedzi do ćwiczenia 16.

21. $\sum_{R(j)} a_j + \sum_{S(j)} a_j = \sum_j a_j [R(j)] + \sum_j a_j [S(j)] = \sum_j a_j ([R(j)] + [S(j)])$; teraz należy skorzystać z tego, że $[R(j)] + [S(j)] = [R(j) \text{ lub } S(j)] + [R(j) \text{ i } S(j)]$. W ogólności wykorzystując notację nawiasową, można wykonać operacje na wyrażeniach „pod symbolem sumy” zamiast „w indeksie symbolu sumy”.

22. Dla wzorów (5) i (7) wystarczy zmienić \sum na \prod . Mamy ponadto $\prod_{R(i)} b_i c_i = (\prod_{R(i)} b_i)(\prod_{R(i)} c_i)$ oraz

$$\left(\prod_{R(j)} a_j \right) \left(\prod_{S(j)} a_j \right) = \left(\prod_{R(j) \text{ lub } S(j)} a_j \right) \left(\prod_{R(j) \text{ i } S(j)} a_j \right).$$

23. $0 + x = x$ i $1 \cdot x = x$. Dzięki przyjętej konwencji uproszczeniu ulega wiele równań i przekształceń – porównaj regułę (d) z jej odpowiednikiem z poprzedniego ćwiczenia.

25. Pierwsze i ostatnie przekształcenie jest poprawne. W drugim przekształceniu wykorzystuje się ten sam symbol i do oznaczenia dwóch różnych rzeczy. W trzecim kroku powinniśmy otrzymać $\sum_{i=1}^n n$.

26. Gdy zapiszemy wyrażenie jak w przykładzie 2, kluczowymi przekształceniami są:

$$\begin{aligned} \prod_{i=0}^n \left(\prod_{j=0}^n a_i a_j \right) &= \prod_{i=0}^n \left(a_i^{n+1} \prod_{j=0}^n a_j \right) \\ &= \left(\prod_{i=0}^n a_i^{n+1} \right) \left(\prod_{i=0}^n \left(\prod_{j=0}^n a_j \right) \right) = \left(\prod_{i=0}^n a_i \right)^{2n+2}. \end{aligned}$$

Odpowiedź: $(\prod_{i=0}^n a_i)^{n+2}$.

28. $(n+1)/2n$.

29. (a) $\sum_{0 \leq k \leq j \leq i \leq n} a_i a_j a_k$. (b) Niech $S_r = \sum_{i=0}^n a_i^r$. Rozwiązanie: $\frac{1}{3}S_3 + \frac{1}{2}S_1 S_2 + \frac{1}{6}S_1^3$. Ogólne rozwiązanie tego problemu dla dużej liczby indeksów znajduje się w punkcie 1.2.9, równanie (38).

30. Zapisz lewą stronę jako sumę $\sum_{1 \leq j, k \leq n} a_j b_k x_j y_k$ i wykonaj podobne przekształcenie po prawej stronie. (Ta tożsamość jest szczególnym przypadkiem ćwiczenia 46 dla $m = 2$).

31. Niech $a_j = u_j$, $b_j = 1$, $x_j = v_j$ i $y_j = 1$, wówczas otrzymujemy $n \sum_{j=1}^n u_j v_j - (\sum_{j=1}^n u_j)(\sum_{j=1}^n v_j)$.

33. Można to udowodnić przez indukcję względem n , jeśli zapiszemy wzór jako

$$\frac{1}{x_n - x_{n-1}} \left(\sum_{j=1}^n \frac{x_j^r (x_j - x_{n-1})}{\prod_{1 \leq k \leq n, k \neq j} (x_j - x_k)} - \sum_{j=1}^n \frac{x_j^r (x_j - x_n)}{\prod_{1 \leq k \leq n, k \neq j} (x_j - x_k)} \right).$$

Mamy zatem sumy takiej postaci, jak suma wyjściowa, poza $n - 1$ elementami. Dla $0 \leq r \leq n - 1$ dowodzimy przez indukcję. Dla $r = n$ bierzemy tożsamość

$$0 = \sum_{j=1}^n \frac{\prod_{k=1}^n (x_j - x_k)}{\prod_{1 \leq k \leq n, k \neq j} (x_j - x_k)} = \sum_{j=1}^n \frac{x_j^n - (x_1 + \dots + x_n)x_j^{n-1} + P(x_j)}{\prod_{1 \leq k \leq n, k \neq j} (x_j - x_k)}$$

gdzie $P(x_j)$ jest wielomianem stopnia $n - 2$. Korzystamy z rozwiązania dla $r = 0, 1, \dots, n - 1$ i gotowe.

Uwagi: Dr. Matrixa uprzedził L. Euler, który w liście do Christiana Goldbacha pisał o powyższym odkryciu 9 listopada 1762 roku, zobacz *Institutionum Calculi Integralis* **2** (1769), §1169; oraz E. Waring, *Phil. Trans.* **69** (1779), 64–67. Istnieje bardziej elegancka, acz mniej elementarna metoda dowodu powyższego twierdzenia, wykorzystująca teorię zmiennej zespolonej. Z twierdzenia o residiach wartość sumy wynosi:

$$\frac{1}{2\pi i} \int_{|z|=R} \frac{z^r dz}{(z - x_1) \dots (z - x_n)}$$

gdzie $R > |x_1|, \dots, |x_n|$. Rozwinięcie Laurenta funkcji podcałkowej zbiega jednostajnie dla $|z| = R$,

$$\begin{aligned} z^{r-n} \left(\frac{1}{1 - x_1/z} \right) \dots \left(\frac{1}{1 - x_n/z} \right) \\ = z^{r-n} + (x_1 + \dots + x_n)z^{r-n-1} + (x_1^2 + x_1x_2 + \dots)z^{r-n-2} + \dots \end{aligned}$$

Gdy całkujemy po wyrazach, znika wszystko poza współczynnikiem z^{-1} . Za pomocą tej metody otrzymujemy *uniwersalny* wzór dla dowolnego całkowitego $r \geq 0$:

$$\sum_{\substack{j_1 + \dots + j_n = r-n+1 \\ j_1, \dots, j_n \geq 0}} x_1^{j_1} \dots x_n^{j_n}.$$

[J. J. Sylvester, *Quart. J. Math.* **1** (1857), 141–152].

34. Jeśli Czytelnik próbował uczciwie rozwiązać to zadanie bez zaglądania do odpowiedzi, zapewne osiągnął cel. Pokusa potraktowania liczników jako wielomianów x , a nie k jest nie do odparcia. Niewątpliwie byłoby łatwiej udowodnić znacznie bardziej ogólny wzór:

$$\sum_{k=1}^n \frac{\prod_{1 \leq r \leq n-1} (y_k - z_r)}{\prod_{1 \leq r \leq n, r \neq k} (y_k - y_r)} = 1,$$

który jest tożsamością o $2n - 1$ zmiennych!

35. Jeżeli $R(j)$ nie zachodzi, wartością wyrażenia powinno być $-\infty$. Przedstawiony odpowiednik reguły (a) jest oparty na tożsamości $a + \max(b, c) = \max(a + b, a + c)$. Podobnie, gdy wszystkie a_i i b_j są *niewielkie*, mamy

$$\sup_{R(i)} a_i \sup_{S(j)} b_j = \sup_{R(i)} \sup_{S(j)} a_i b_j.$$

Reguły (b), (c) nie ulegają zmianie; reguła (d) upraszcza się do

$$\sup(\sup_{R(j)} a_j, \sup_{S(j)} a_j) = \sup_{R(j) \text{ or } S(j)} a_j.$$

36. Odejmij pierwszą kolumnę od kolumn $2, \dots, n$. Do pierwszego wiersza dodaj wiersze $2, \dots, n$. Wystarczy obliczyć wyznacznik macierzy trójkątnej.

37. Odejmij kolumnę pierwszą od kolumn $2, \dots, n$, następnie odejmij wiersz $k - 1$ pomnożony przez x_1 od wiersza k dla $k = n, n - 1, \dots, 2$ (w tej kolejności). Teraz podziel pierwszą kolumnę przez x_1 , a kolumnę k przez $x_k - x_1$, dla $k = 2, \dots, n$. Otrzymasz $x_1(x_2 - x_1) \dots (x_n - x_1)$ razy wyznacznik macierzy Vandermonde'a stopnia $n - 1$. Dalej przez indukcję.

Oto alternatywny dowód (uwaga, matematyka wyższa). Wyznacznik jest wielomianem zmiennych x_1, \dots, x_n stopnia $1 + 2 + \dots + n$, który zeruje się, gdy $x_j = 0$ lub $x_i = x_j$ ($i < j$), a współczynnik przy $x_1^1 x_2^2 \dots x_n^n$ jest równy $+1$. Te obserwacje umożliwiają wyznaczenie wielomianu. W ogólności, jeżeli dwa wiersze macierzy są równe dla $x_i = x_j$, to ich różnica jest zazwyczaj podzielna przez $x_i - x_j$. W wielu przypadkach ta operacja pozwala przypieszyć obliczanie wyznacznika.

38. Odejmij kolumnę pierwszą od kolumn $2, \dots, n$, następnie podziel wszystkie wiersze i kolumny przez

$$(x_1 + y_1)^{-1} \dots (x_n + y_1)^{-1} (y_1 - y_2) \dots (y_1 - y_n).$$

Następnie, odejmij wiersz pierwszy od wierszy $2, \dots, n$ i podziel przez $(x_1 - x_2) \dots (x_1 - x_n) (x_1 + y_2)^{-1} \dots (x_1 + y_n)^{-1}$. Otrzymasz macierz Cauchy'ego stopnia $n - 1$.

39. Niech I będzie macierzą jednostkową (δ_{ij}), a J macierzą złożoną z samych jedynek. Mamy $J^2 = nJ$, zatem $(xI + yJ)((x + ny)I - yJ) = x(x + ny)I$.

$$\mathbf{40.} \sum_{t=1}^n b_{it} x_j^t = x_j \prod_{\substack{1 \leq k \leq n \\ k \neq i}} (x_k - x_j) \Bigg/ x_i \prod_{\substack{1 \leq k \leq n \\ k \neq i}} (x_k - x_i) = \delta_{ij}.$$

41. Wynika to wprost ze związku między elementami macierzy odwrotnej i ich dopełnieniami algebraicznymi. Interesujący jest także dowód bezpośredni. Mamy

$$\sum_{t=1}^n \frac{1}{x_i + y_t} b_{tj} = \sum_{t=1}^n \frac{\prod_{k \neq t} (x_j + y_k - x) \prod_{k \neq i} (x_k + y_t)}{\prod_{k \neq j} (x_j - x_k) \prod_{k \neq t} (y_t - y_k)}$$

gdy $x = 0$. To jest wielomian zmiennej x stopnia co najwyżej $n - 1$. Po podstawieniu $x = x_j + y_s$, $1 \leq s \leq n$, zerują się wszystkie wyrazy poza przypadkiem $s = t$, zatem wartość wielomianu wynosi

$$\prod_{k \neq i} (-x_k - y_s) \Bigg/ \prod_{k \neq j} (x_j - x_k) = \prod_{k \neq i} (x_j - x_k - x) \Bigg/ \prod_{k \neq j} (x_j - x_k).$$

To są wielomiany stopnia co najwyżej $n - 1$, równe dla n różnych wartości x , zatem dla $x = 0$ także. Stąd

$$\sum_{t=1}^n \frac{1}{x_i + y_t} b_{tj} = \prod_{k \neq i} (x_j - x_k) \Bigg/ \prod_{k \neq j} (x_j - x_k) = \delta_{ij}.$$

42. $n/(x + ny)$.

43. $1 - \prod_{k=1}^n (1 - 1/x_k)$. Łatwo to sprawdzić, jeśli któryś $x_i = 1$, bo suma elementów macierzy odwrotnej do dowolnej macierzy zawierającej wiersz lub kolumnę jedynek wynosi jeden. Jeżeli żadne x_i nie równa się jeden, zsumuj elementy wiersza i jak w ćwiczeniu 44, otrzymując $\prod_{k \neq i} (x_k - 1)/x_i \prod_{k \neq i} (x_k - x_i)$. Teraz sumujemy względem i , wykorzystując ćwiczenie 33 dla $r = 0$ (pomnóż licznik i mianownik przez $(x_i - 1)$).

44. Stosując wynik ćwiczenia 33, otrzymujemy

$$c_j = \sum_{i=1}^n b_{ij} = \prod_{k=1}^n (x_j + y_k) \Bigg/ \prod_{\substack{1 \leq k \leq n \\ k \neq j}} (x_j - x_k).$$

Dalej

$$\begin{aligned} \sum_{j=1}^n c_j &= \sum_{j=1}^n \frac{(x_j^n + (y_1 + \dots + y_n)x_j^{n-1} + \dots)}{\prod_{1 \leq k \leq n, k \neq j} (x_j - x_k)} \\ &= (x_1 + x_2 + \dots + x_n) + (y_1 + y_2 + \dots + y_n). \end{aligned}$$

45. Niech $x_i = i$, $y_j = j - 1$. Na mocy ćwiczenia 44 suma elementów macierzy odwrotnej wynosi $(1 + 2 + \dots + n) + ((n-1) + (n-2) + \dots + 0) = n^2$. Z ćwiczenia 38 wiemy, że elementy macierzy odwrotnej są postaci

$$b_{ij} = \frac{(-1)^{i+j}(i+n-1)!(j+n-1)!}{(i+j-1)(i-1)!^2(j-1)!^2(n-i)!(n-j)!}.$$

Tę wartość można przedstawić na kilka sposobów za pomocą współczynników dwumianowych, na przykład

$$\frac{(-1)^{i+j}ij}{i+j-1} \binom{-i}{n} \binom{n}{i} \binom{-j}{n} \binom{n}{j} = (-1)^{i+j} j \binom{i+j-2}{i-1} \binom{i+n-1}{i-1} \binom{j+n-1}{n-i} \binom{n}{j}.$$

Z prawego wzoru wynika, że nie dość, iż b_{ij} jest liczbą całkowitą, to jeszcze dzieli się przez i , j , n , $i+j-1$, $i+n-1$, $j+n-1$, $n-i+1$, i $n-j+1$. Zapewne najładniejszą postacią b_{ij} jest

$$(i+j-1) \binom{i+j-2}{i-1}^2 \binom{-(i+j)}{n-i} \binom{-(i+j)}{n-j}.$$

Bez obserwacji, że macierz Hilberta jest szczególnym przypadkiem macierzy Cauchy'ego, rozwiązań zadania byłoby bardzo trudne. Łatwiej rozwiązać problem bardziej ogólny niż przypadek szczególny! Często pomocne jest uogólnienie problemu do jego „domknięcia indukcyjnego”, tj. do najskromniejszego uogólnienia, dla którego wszystkie podproblemy pojawiające się podczas prób skonstruowania dowodu przez indukcję należą do tej samej klasy, co udowadniane twierdzenie. Tutaj dopełnienia algebraiczne macierzy Cauchy'ego są macierzami Cauchy'ego, ale dopełnienia algebraiczne macierzy Hilberta nie są macierzami Hilberta. [Więcej informacji znajdzie Czytelnik w: J. Todd, *J. Res. Nat. Bur. Stand.* **65** (1961), 19–22].

46. Dla dowolnych liczb całkowitych k_1, k_2, \dots, k_m zdefiniujmy funkcję $\epsilon(k_1, \dots, k_m) = \text{sign}(\prod_{1 \leq i < j \leq m} (k_j - k_i))$, gdzie $\text{sign } x = [x > 0] - [x < 0]$. Jeśli (l_1, \dots, l_m) jest równe (k_1, \dots, k_m) z dokładnością do zamiany k_i z k_j , to $\epsilon(l_1, \dots, l_m) = -\epsilon(k_1, \dots, k_m)$. Stąd otrzymujemy $\det(B_{k_1 \dots k_m}) = \epsilon(k_1, \dots, k_m) \det(B_{j_1 \dots j_m})$, gdy $j_1 \leq \dots \leq j_m$ są liczbami k_1, \dots, k_m ustawnionymi w porządku niemalejącym. Teraz z definicji wyznacznika

$$\begin{aligned}\det(AB) &= \sum_{1 \leq l_1, \dots, l_m \leq m} \epsilon(l_1, \dots, l_m) \left(\sum_{k=1}^n a_{1k} b_{kl_1} \right) \dots \left(\sum_{k=1}^n a_{mk} b_{kl_m} \right) \\ &= \sum_{1 \leq k_1, \dots, k_m \leq n} a_{1k_1} \dots a_{mk_m} \sum_{1 \leq l_1, \dots, l_m \leq m} \epsilon(l_1, \dots, l_m) b_{k_1 l_1} \dots b_{k_m l_m} \\ &= \sum_{1 \leq k_1, \dots, k_m \leq n} a_{1k_1} \dots a_{mk_m} \det(B_{k_1 \dots k_m}) \\ &= \sum_{1 \leq k_1, \dots, k_m \leq m} \epsilon(k_1, \dots, k_m) a_{1k_1} \dots a_{mk_m} \det(B_{j_1 \dots j_m}) \\ &= \sum_{1 \leq j_1 \leq \dots \leq j_m \leq n} \det(A_{j_1 \dots j_m}) \det(B_{j_1 \dots j_m}).\end{aligned}$$

Ponadto jeżeli dwa „ j ” są równe, to $\det(A_{j_1 \dots j_m}) = 0$. [*J. de l'École Polytechnique* **9** (1813), 280–354; **10** (1815), 29–112. Binet i Cauchy przedstawili swoje wyniki tego samego dnia 1812 roku].

47. Niech $a_{ij} = (\prod_{k=1}^{j-1} (x_i + p_k)) (\prod_{k=j+1}^n (x_i + q_k))$. Odejmujemy kolumnę $k-1$ od kolumny k i dzielimy przez $p_{k-j} - q_k$, dla $k = n, n-1, \dots, j+1$ (w tej kolejności), dla $j = 1, 2, \dots, n-1$ (w tej kolejności). Stąd zostaje $\prod_{1 \leq i < j \leq n} (p_i - q_j)$ razy $\det(b_{ij})$, gdzie $b_{ij} = \prod_{k=j+1}^n (x_i + q_k)$. Teraz odejmujemy q_{k+j} razy kolumnę $k+1$ od kolumny k dla $k = 1, \dots, n-j$ i dla $j = 1, \dots, n-1$. Otrzymujemy stąd $\det(c_{ij})$, gdzie $c_{ij} = x_i^{n-j}$, czyli w istocie macierz Vandermonde'a. Można ją przekształcać dalej jak w ćwiczeniu 37, jednakże biorąc wiersze, a nie kolumny. Otrzymujemy

$$\det(a_{ij}) = \prod_{1 \leq i < j \leq n} (x_i - x_j)(p_i - q_j).$$

Jeśli $p_j = q_j = y_j$ dla $1 \leq j \leq n$, to macierz dana w ćwiczeniu jest macierzą Cauchy'ego, w której wiersz i pomnożono przez $\prod_{j=1}^n (x_i + y_j)$. Z tego punktu widzenia powyższy wynik powoduje uogólnienie ćwiczenia 38 przez dodanie $n-1$ niezależnych parametrów. [*Manuscripta Math.* **69** (1990), 177–178].

1.2.4

1. 1, -2, -1, 0, 5.
2. $\lfloor x \rfloor$.
3. Na mocy definicji $\lfloor x \rfloor$ jest największą liczbą całkowitą mniejszą lub równą x ; stąd $\lfloor x \rfloor$ jest liczbą całkowitą, $\lfloor x \rfloor \leq x$ i $\lfloor x \rfloor + 1 > x$. Powyższe własności oraz fakt, że dla całkowitych m i n relacja $m < n$ zachodzi dokładnie wtedy, gdy $m \leq n-1$, prowadzą do prostych dowodów faktów (a) i (b). Podobnie dowodzi się (c) i (d). Wreszcie (e) i (f) to po prostu koniunkcje poprzednich podpunktów.
4. $x - 1 < \lfloor x \rfloor \leq x$, zatem $-x + 1 > -\lfloor x \rfloor \geq -x$; dalej łatwe.
5. $\lfloor x + \frac{1}{2} \rfloor$. Wartość $(-x \text{ zaokrąglone})$ będzie taka sama, jak $-(x \text{ zaokrąglone})$, poza $x \bmod 1 = \frac{1}{2}$, kiedy to wartości będą zaokrąglane w kierunku zera.

6. (a) Prawda: $\lfloor \sqrt{x} \rfloor = n \iff n^2 \leq x < (n+1)^2 \iff n^2 \leq \lfloor x \rfloor < (n+1)^2 \iff \lfloor \sqrt{\lfloor x \rfloor} \rfloor = n$. Podobnie (b). Natomiast (c) nie jest spełnione na przykład dla $x = 1.1$.

7. $\lfloor x+y \rfloor = \lfloor \lfloor x \rfloor + x \bmod 1 + \lfloor y \rfloor + y \bmod 1 \rfloor = \lfloor x \rfloor + \lfloor y \rfloor + \lfloor x \bmod 1 + y \bmod 1 \rfloor$. Dla sufitów prawdziwa jest nierówność \geq , a równość zachodzi wtedy i tylko wtedy, gdy x lub y jest całkowite lub gdy $x \bmod 1 + y \bmod 1 > 1$.

8. 1, 2, 5, -100.

9. -1, 0, -2.

10. 0.1, 0.01, -0.09.

11. $x = y$.

12. Wszystkie.

13. +1, -1.

14. 8.

15. Pomnóż obie strony (1) przez z . Przypadek $y = 0$ jest jeszcze łatwiejszy.

17. Rozważmy na przykład fragment prawa A dotyczący mnożenia. Mamy $a = b + qm$ oraz $x = y + rm$ dla pewnych całkowitych q i r , zatem $ax = by + (br + yq + qrm)m$.

18. Dla pewnej liczby całkowitej k zachodzi $a - b = kr$, a także $kr \equiv 0$ (modulo s). Stąd na mocy prawa B $k \equiv 0$ (modulo s), zatem $a - b = qsr$ dla pewnej liczby całkowitej q .

20. Pomnóż obie strony przez a' .

21. Na mocy dowodów z wcześniejszych ćwiczeń istnieje przynajmniej jeden taki sposób przedstawienia. Przypuśćmy, że istnieją dwa różne sposoby przedstawienia $n = p_1 \dots p_k = q_1 \dots q_m$, wówczas $q_1 \dots q_m \equiv 0$ (modulo p_1). Skoro żadna z liczb $q_1 \dots q_m \equiv 0$ nie jest równa p_1 , to możemy je wszystkie poskracać, korzystając z prawa B, skąd otrzymujemy $1 \equiv 0$ (modulo p_1). Ale to jest niemożliwe, bo p_1 jest różne od 1. Zatem któryś q_j równa się p_1 i $n/p_1 = p_2 \dots p_k = q_1 \dots q_{j-1} q_{j+1} \dots q_m$. Albo n jest liczbą pierwszą i wynik mamy natychmiast, albo nie. Wówczas przez indukcję udowadniamy, że oba rozkłady n/p_1 są takie same.

22. Niech $m = ax$, gdzie $a > 1$ i $x > 0$. Wówczas $ax \equiv 0$, ale $x \not\equiv 0$ (modulo m).

24. Prawo A zachodzi zawsze dla dodawania i odejmowania, a prawo C zachodzi zawsze.

26. Jeśli liczba b nie jest wielokrotnością p , to liczba $b^2 - 1$ jest, zatem któryś z jej czynników także.

27. Liczba jest względnie pierwsza z p^e wtedy i tylko wtedy, gdy nie jest wielokrotnością p . Zliczamy zatem te liczby, które nie są wielokrotnościami p i dostajemy $\varphi(p^e) = p^e - p^{e-1}$.

28. Jeśli a i b są względnie pierwsze z m , to z $ab \bmod m$ także, gdyż każda liczba pierwsza, która dzieliłaby zarówno m , jak i $ab \bmod m$, musiałaby dzielić także a lub b . Niech $x_1, \dots, x_{\varphi(m)}$ będą liczbami względnie pierwszymi z m . Zauważmy, że $ax_1 \bmod m, \dots, ax_{\varphi(m)} \bmod m$ są tymi samymi liczbami w pewnym porządku itd.

29. Udowodnimy (b): jeśli $r \perp s$ i jeśli k^2 dzieli rs , to p^2 dzieli rs dla pewnej liczby pierwszej p , zatem p dzieli r i nie może dzielić s ; stąd p^2 dzieli r . Widać, że $f(rs) = 0$ wtedy i tylko wtedy, gdy $f(r) = 0$ lub $f(s) = 0$.

30. Przypuśćmy, że $r \perp s$. Pierwszy pomysł polega na pokazaniu, że $\varphi(rs)$ liczb względnie pierwszych z rs to dokładnie $\varphi(r)\varphi(s)$ różnych liczb $(sx_i + ry_j) \bmod (rs)$, gdzie $x_1, \dots, x_{\varphi(r)}$ i $y_1, \dots, y_{\varphi(s)}$ są odpowiednio wartościami dla r i s .

Ponieważ funkcja φ jest multiplikatywna, więc $\varphi(10^6) = \varphi(2^6)\varphi(5^6) = (2^6 - 2^5)(5^6 - 5^5) = 400000$. W ogólności, dla $n = p_1^{e_1} \dots p_r^{e_r}$, zachodzi $\varphi(n) = (p_1^{e_1} - p_1^{e_1-1}) \dots (p_r^{e_r} - p_r^{e_r-1}) = n \prod_{p \nmid n, p \text{ pierwsze}} (1 - 1/p)$. (Inny dowód jest podany w ćwiczeniu 1.3.3–27).

31. Skorzystaj z faktu, że dzielnik rs może być jednoznacznie zapisany w postaci cd , gdzie c dzieli r , a d dzieli s . Podobnie, jeżeli $f(n) \geq 0$, to można pokazać, że funkcja $\max_{d \mid n} f(d)$ jest multiplikatywna (zobacz ćwiczenie 1.2.3–35).

33. Albo $n+m$, albo $n-m+1$ jest parzyste, zatem jedna z liczb w nawiasach $(\lfloor \cdot \rfloor$ i $\lceil \cdot \rceil$) jest całkowita. Stąd we wzorze z ćwiczenia 7 mamy równość, zatem: (a) n , (b) $n+1$.

34. Liczba b musi być liczbą całkowitą ≥ 2 (bo można wziąć $x = b$). Dostateczności dowodzimy podobnie jak w ćwiczeniu 6. Ten sam warunek jest konieczny i wystarczający dla $\lceil \log_b x \rceil = \lceil \log_b \lceil x \rceil \rceil$.

Uwaga: R. J. McEliece wskazał następujące uogólnienie. Niech f będzie funkcją ciągłą i ściśle rosnącą określona na przedziale A i niech zarówno $\lfloor x \rfloor$, jak i $\lceil x \rceil$ należą do A , gdy x należy do A . Wówczas warunek $\lfloor f(x) \rfloor = \lfloor f(\lfloor x \rfloor) \rfloor$ zachodzi dla wszystkich x z przedziału A wtedy i tylko wtedy, gdy warunek $\lceil f(x) \rceil = \lceil f(\lceil x \rceil) \rceil$ zachodzi dla wszystkich x z przedziału A , wtedy i tylko wtedy gdy spełniony jest warunek następujący: „jeśli wartość $f(x)$ jest całkowita, to wartość x jest całkowita”. Warunek jest oczywiście konieczny, bo gdy wartość $f(x)$ jest całkowita i równa $\lfloor f(\lfloor x \rfloor) \rfloor$ lub $\lceil f(\lceil x \rceil) \rceil$, wówczas x musi się równać $\lfloor x \rfloor$ lub $\lceil x \rceil$. Jednak jeśli na przykład $\lfloor f(\lfloor x \rfloor) \rfloor < \lfloor f(x) \rfloor$, to z ciągłości musi istnieć y spełniające $\lfloor x \rfloor < y \leq x$, dla którego $f(y)$ jest liczbą całkowitą, ale takie y nie może być całkowite.

$$35. \frac{x+m}{n} - 1 = \frac{x+m}{n} - \frac{1}{n} - \frac{n-1}{n} < \frac{\lfloor x \rfloor + m}{n} - \frac{n-1}{n} \leq \left\lfloor \frac{\lfloor x \rfloor + m}{n} \right\rfloor \leq \frac{x+m}{n}; \text{ sko-}$$

rzystaj z ćwiczenia 3. Posługując się ćwiczeniem 4, można uzyskać podobny wynik dla sufitów. Obie tożsamości można wyprowadzić jako przypadek szczególny twierdzenia McEliece'a z ćwiczenia 34.

36. Założymy najpierw, że $n = 2t$. Wówczas

$$\sum_{k=1}^n \left\lfloor \frac{k}{2} \right\rfloor = \sum_{k=1}^n \left\lfloor \frac{n+1-k}{2} \right\rfloor;$$

zatem z ćwiczenia 33

$$\sum_{k=1}^n \left\lfloor \frac{k}{2} \right\rfloor = \frac{1}{2} \sum_{k=1}^n \left(\left\lfloor \frac{k}{2} \right\rfloor + \left\lfloor \frac{n+1-k}{2} \right\rfloor \right) = \frac{1}{2} \sum_{k=1}^n \left\lfloor \frac{2t+1}{2} \right\rfloor = t^2 = \frac{n^2}{4}.$$

Jeśli $n = 2t+1$, to $t^2 + \lfloor n/2 \rfloor = t^2 + t = n^2/4 - 1/4$. Dla drugiej sumy analogicznie dostajemy $\lceil n(n+2)/4 \rceil$.

37. $\sum_{0 \leq k < n} \frac{mk+x}{n} = \frac{m(n-1)}{2} + x$. Niech $\{y\}$ oznacza $y \bmod 1$. Musimy odjąć

$$S = \sum_{0 \leq k < n} \left\{ \frac{mk+x}{n} \right\}.$$

S składa się z d kopii tej samej sumy, bo dla $t = n/d$ mamy

$$\left\{ \frac{mk+x}{n} \right\} = \left\{ \frac{m(k+t)+x}{n} \right\}.$$

Niech $u = m/d$, wówczas

$$\sum_{0 \leq k < t} \left\{ \frac{mk+x}{n} \right\} = \sum_{0 \leq k < t} \left\{ \frac{x}{n} + \frac{uk}{t} \right\},$$

a ponieważ $t \perp u$, można poprzestawiać składniki uzyskując

$$\left\{ \frac{x \bmod d}{n} \right\} + \left\{ \frac{x \bmod d}{n} + \frac{1}{t} \right\} + \cdots + \left\{ \frac{x \bmod d}{n} + \frac{t-1}{t} \right\}.$$

Na koniec usuwamy nawiasy klamrowe, bo $(x \bmod d)/n < 1/t$. Otrzymujemy

$$S = d \left(\frac{t(x \bmod d)}{n} + \frac{t-1}{2} \right).$$

Zastosowanie ćwiczenia 4 pozwala wyprowadzić podobną tożsamość:

$$\sum_{0 \leq k < n} \left\lceil \frac{mk+x}{n} \right\rceil = \frac{(m+1)(n-1)}{2} - \frac{d-1}{2} + d[x/d].$$

Wzór byłby symetryczny względem m i n , gdybyśmy rozszerzyli sumowanie na zakres $0 \leq k \leq n$. (Symetrię można wyjaśnić, rysując wykres wyrażenia pod sumą jako funkcję zmiennej k , a następnie odbijając symetrycznie względem prostej $y = x$).

38. Obie strony zwiększą się o $[y]$, gdy x zwiększa się o 1. Możemy więc ograniczyć się do przypadku $0 \leq x < 1$. Obie strony są równe zero, gdy $x = 0$ i obie strony zwiększą się o 1, gdy x przekracza wartość $1 - k/y$ dla $y > k \geq 0$. [Crelle 136 (1909), 42; przypadek $y = n$ pochodzi od C. Hermite'a, Acta Math. 5 (1884), 315].

39. Dowód punktu (f). Rozważmy bardziej ogólną tożsamość $\prod_{0 \leq k < n} 2 \sin \pi(x+k/n) = 2 \sin \pi nx$, którą można uzasadnić następująco: ponieważ $2 \sin \theta = (e^{i\theta} - e^{-i\theta})/i = (1 - e^{-2i\theta})e^{i\theta-i\pi/2}$, zatem tożsamość wynika ze wzorów:

$$\prod_{0 \leq k < n} (1 - e^{-2\pi(x+ik/n)}) = 1 - e^{-2\pi nx} \quad \text{oraz} \quad \prod_{0 \leq k < n} e^{\pi(x-(1/2)+(k/n))} = e^{\pi(nx-1/2)}.$$

Drugi z nich jest prawdziwy, gdyż funkcja $x - \frac{1}{2}$ jest powielająca; pierwszy wzór zachodzi dlatego, że możemy przyjąć $z = 1$ w rozkładzie wielomianu $z^n - \alpha^n = (z - \alpha)(z - \omega\alpha) \dots (z - \omega^{n-1}\alpha)$, gdzie $\omega = e^{-2\pi i/n}$.

40. (Spostrzeżenie N. G. de Bruijna) Jeżeli f jest powielająca, to $f(nx+1) - f(nx) = f(x+1) - f(x)$ dla dowolnego $n > 0$. Stąd jeżeli f jest ciągła, to $f(x+1) - f(x) = c$ dla dowolnego x . Przy tym $g(x) = f(x) - c[x]$ jest powielająca i okresowa. Z rozwinięcia w szereg Fouriera

$$\int_0^1 e^{2\pi i n x} g(x) dx = \frac{1}{n} \int_0^1 e^{2\pi y} g(y) dy;$$

wynika, że dla $0 < x < 1$ zachodzi $g(x) = (x - \frac{1}{2})a$. Stąd $f(x) = (x - \frac{1}{2})a$. W ogólności z powyższego rozumowania mamy, że każda funkcja powielająca lokalnie całkowalna w sensie Riemanna jest prawie wszędzie postaci $(x - \frac{1}{2})a + b \max(\lfloor x \rfloor, 0) + c \min(\lfloor x \rfloor, 0)$. Więcej informacji można znaleźć w: L. J. Mordell, J. London Math. Soc. 33 (1958), 371–375; M. F. Yoder, *Æquationes Mathematicæ* 13 (1975), 251–261.

41. Chcemy $a_n = k$, gdy $\frac{1}{2}k(k-1) < n \leq \frac{1}{2}k(k+1)$. Ponieważ n jest całkowite, odpowiada to warunkowi

$$\frac{k(k-1)}{2} + \frac{1}{8} < n < \frac{k(k+1)}{2} + \frac{1}{8},$$

tj. $k - \frac{1}{2} < \sqrt{2n} < k + \frac{1}{2}$. Stąd $a_n = \lfloor \sqrt{2n} + \frac{1}{2} \rfloor$, co jest liczbą całkowitą najbliższą $\sqrt{2n}$. Inne poprawne odpowiedzi to $\lceil \sqrt{2n} - \frac{1}{2} \rceil$, $\lceil (\sqrt{8n+1}-1)/2 \rceil$, $\lfloor (\sqrt{8n-7}+1)/2 \rfloor$ itd.

42. (a) Zobacz ćwiczenie 1.2.7–10. (b) Podana suma to $n\lfloor \log_b n \rfloor - S$, gdzie

$$S = \sum_{\substack{1 \leq k \leq n \\ k+1 \text{ jest potegą } b}} k = \sum_{1 \leq t \leq \log_b n} (b^t - 1) = (b^{\lfloor \log_b n \rfloor + 1} - b)/(b - 1) - \lfloor \log_b n \rfloor.$$

43. $\lfloor \sqrt{n} \rfloor (n - \frac{1}{6}(2\lfloor \sqrt{n} \rfloor + 5)(\lfloor \sqrt{n} \rfloor - 1))$.

44. Dla ujemnych n wartość sumy wynosi $n + 1$.

45. $\lfloor mj/n \rfloor = r$ wtedy i tylko wtedy, gdy $\left\lceil \frac{rn}{m} \right\rceil \leq j < \left\lceil \frac{(r+1)n}{m} \right\rceil$, stąd dana suma jest równa

$$\sum_{0 \leq r < m} f(r) \left(\left\lceil \frac{(r+1)n}{m} \right\rceil - \left\lceil \frac{rn}{m} \right\rceil \right).$$

Dowodzoną równość otrzymujemy, zmieniając kolejność wyrazów w powyższej sumie, tak by zgrupować wyrazy o jednakowych wartościach $\lceil rn/m \rceil$. Drugi wzór dostajemy natychmiast po podstawieniu

$$f(x) = \binom{x+1}{k}.$$

46. $\sum_{0 \leq j < \alpha n} f(\lfloor mj/n \rfloor) = \sum_{0 \leq r < \alpha m} \lceil rn/m \rceil (f(r-1) - f(r)) + \lceil \alpha n \rceil f(\lceil \alpha m \rceil - 1)$.

47. (a) Liczby $2, 4, \dots, p-1$ są parzystymi resztami z dzielenia przez p ; ponieważ $2kq = p\lfloor 2kq/p \rfloor + (2kq) \bmod p$, to liczba $(-1)^{\lfloor 2kq/p \rfloor} ((2kq) \bmod p)$ jest parzystą resztą albo parzystą resztą minus p , a każda parzysta reszta pojawia się tylko raz. Stąd $(-1)^\sigma q^{(p-1)/2} 2 \cdot 4 \dots (p-1) \equiv 2 \cdot 4 \dots (p-1)$. (b) Niech $q = 2$. Jeśli $p = 4n+1$, to $\sigma = n$; jeśli $p = 4n+3$, to $\sigma = n+1$. Stąd $\binom{2}{p} = (1, -1, -1, 1)$ odpowiednio dla $p \bmod 8 = (1, 3, 5, 7)$. (c) Dla $k < p/4$ mamy

$$\lfloor (p-1-2k)q/p \rfloor = q - \lceil (2k+1)q/p \rceil = q - 1 - \lfloor (2k+1)q/p \rfloor \equiv \lfloor (2k+1)q/p \rfloor \pmod{2}.$$

Możemy zatem zastąpić końcowe wyrazy $\lfloor (p-1)q/p \rfloor, \lfloor (p-3)q/p \rfloor, \dots$ wyrażeniami $\lfloor q/p \rfloor, \lfloor 3q/p \rfloor$ itd. (d) $\sum_{0 \leq k < p/2} \lfloor kq/p \rfloor + \sum_{0 \leq r < q/2} \lceil rp/q \rceil = \lceil p/2 \rceil (\lceil q/2 \rceil - 1) = (p+1)(q-1)/4$. Podobnie $\sum_{0 \leq r < q/2} \lceil rp/q \rceil = \sum_{0 \leq r < q/2} \lfloor rp/q \rfloor + (q-1)/2$. Pomysł tego dowodu pochodzi z: G. Eisenstein, *Crelle* **28** (1844), 246–248; w tym samym numerze Eisenstein podał kilka dowodów tego oraz innych praw wzajemności.

48. (a) Oczywiście nie zawsze zachodzi dla $n < 0$. Łatwo sprawdzić dla $n > 0$. (b) $\lfloor (n+2-\lfloor n/25 \rfloor)/3 \rfloor = \lceil (n-\lfloor n/25 \rfloor)/3 \rceil = \lceil (n+\lceil -n/25 \rceil)/3 \rceil = \lceil \lceil 24n/25 \rceil / 3 \rceil = \lceil 8n/25 \rceil = \lfloor (8n+24)/25 \rfloor$. Przedostatnia równość jest wytłumaczona w ćwiczeniu 35.

49. Ponieważ $f(0) = f(f(0)) = f(f(0) + 0) = f(0) + f(0)$, zatem $f(n) = n$ dla wszystkich całkowitych n . Jeśli $f(\frac{1}{2}) = k \leq 0$, to zachodzi $k = f(\frac{1}{1-2k} f(\frac{1}{2} - k)) = f(\frac{1}{1-2k} (f(\frac{1}{2}) - k)) = f(0) = 0$. Ponadto jeśli $f(\frac{1}{n-1}) = 0$, to $f(\frac{1}{n}) = f(\frac{1}{n} f(1 + \frac{1}{n-1})) = f(\frac{1}{n-1}) = 0$. Dalej, dla $1 \leq m < n$ mamy $f(\frac{m}{n}) = f(\frac{1}{a} f(\frac{am}{n})) = f(\frac{1}{a}) = 0$ przy $a = \lceil n/m \rceil$ (indukcja po m). Stąd jeśli $f(\frac{1}{2}) \leq 0$ to $f(x) = \lfloor x \rfloor$ dla wszystkich wymiernych x . Natomiast jeśli $f(\frac{1}{2}) > 0$, to funkcja $g(x) = -f(-x)$ spełnia (i) i (ii), a także $g(\frac{1}{2}) = 1 - f(\frac{1}{2}) \leq 0$, zatem $f(x) = -g(-x) = -\lfloor -x \rfloor = \lceil x \rceil$ dla wszystkich wymiernych x . [P. Eisele i K. P. Hadeler, *AMM* **97** (1990), 475–477].

Zauważmy jednak, że warunek $f(x) = \lfloor x \rfloor$ ani $f(x) = \lceil x \rceil$ nie musi zachodzić dla wszystkich x rzeczywistych. Jeśli na przykład $h(x)$ jest dowolną funkcją, taką że

$h(1) = 1$ i $h(x + y) = h(x) + h(y)$ dla wszystkich rzeczywistych wartości x i y , to funkcja $f(x) = \lfloor h(x) \rfloor$ spełnia (i) i (ii); ale $h(x)$ może być nieograniczona i w dużej mierze dowolna dla $0 < x < 1$ [G. Hamel, *Math. Annalen* **60** (1905), 459–462].

1.2.5

1. 52!. Ta liczba to 806 58175 17094 38785 71660 63685 64037 66975 28950 54408 83277 82400 00000 00000. (!)

2. $p_{nk} = p_{n(k-1)}(n - k + 1)$. Po ustawnieniu $n - 1$ pierwszych obiektów jest tylko jeden sposób na ustawnienie ostatniego.

3. 53124, 35124, 31524, 31254, 31245; 42351, 41352, 41253, 31254, 31245.

4. Cyfr jest 2568. Cyfrą najbardziej znaczącą jest 4 (bo $\log_{10} 4 = 2 \log_{10} 2 \approx 0.602$). Cyfrą najmniej znaczącą jest 0, bo na mocy (8) wszystkie 249 najmniej znaczących cyfr to zera. Dokładna wartość $1000!$ została obliczona przez H. S. Uhlera, który dysponował wyłącznie zwykłym kalkulatorem i olbrzymią cierpliwością. Wartość opublikowano w *Scripta Mathematica* **21** (1955), 266–267. Początek reprezentacji dziesiętnej to 402 38726 00770 (Ostatni krok obliczeń polegający na wymnożeniu $750!$ przez $\prod_{k=751}^{1000} k$ został wykonany za pomocą maszyny UNIVAC I przez Johna W. Wrencha, Jr., „w rewelacyjnym czasie dwóch i pół minut”. Dziś komputer osobisty bez trudu oblicza $1000!$ w ułamku sekundy, dzięki czemu możemy się przekonać, że wartość otrzymana przez Uhlera była całkowicie poprawna).

5. $(39902)(97/96) \approx 416 + 39902 = 40318$.

6. $2^{18} \cdot 3^8 \cdot 5^4 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$.

8. Granica wynosi $\lim_{m \rightarrow \infty} m^n m! / ((n+m)!/n!) = n! \lim_{m \rightarrow \infty} m^n / ((m+1) \dots (m+n)) = n!$, gdyż $m/(m+k) \rightarrow 1$.

9. $\sqrt{\pi}$ i $-2\sqrt{\pi}$ (zostało wykorzystane ćwiczenie 10).

10. Tak, poza przypadkiem, gdy x jest zerem lub ujemną liczbą całkowitą, bo

$$\Gamma(x+1) = x \lim_{m \rightarrow \infty} \frac{m^x m!}{x(x+1) \dots (x+m)} \left(\frac{m}{x+m+1} \right).$$

11, 12. $\mu = (a_k p^{k-1} + \dots + a_1) + (a_k p^{k-2} + \dots + a_2) + \dots + a_k$
 $= a_k(p^{k-1} + \dots + p + 1) + \dots + a_1 = (a_k(p^k - 1) + \dots + a_0(p^0 - 1))/(p - 1)$
 $= (n - a_k - \dots - a_1 - a_0)/(p - 1)$.

13. Dla dowolnego n wyznacz n' jak w ćwiczeniu 1.2.4–19. Na mocy prawa 1.2.4D takie n' jest wyznaczone jednoznacznie, przy tym $(n')' = n$. Stąd liczby $\{1, 2, \dots, p-1\}$ możemy ustawić w pary, jeżeli $n' \neq n$. Jeżeli $n' = n$, to $n^2 \equiv 1$ (modulo p), zatem (jak w ćwiczeniu 1.2.4–26) $n = 1$ lub $n = p - 1$. Stąd $(p-1)! \equiv 1 \cdot 1 \dots 1 \cdot (-1)$, bo 1 i $p-1$ są jedynymi niesparowanymi elementami.

14. Wśród liczb $\{1, 2, \dots, n\}$, które *nie* są wielokrotnościami p , jest $\lfloor n/p \rfloor$ pełnych zbiorów $p-1$ kolejnych elementów, każdy o iloczynie przystającym do -1 (modulo p) (z twierdzenia Wilsona). Poza tym jest jeszcze a_0 elementów, które przystają do $a_0!$ (modulo p). Stąd wkład czynników, które nie są wielokrotnościami liczby p , wynosi $(-1)^{\lfloor n/p \rfloor} a_0!$. Wkład czynników, które *są* wielokrotnościami p , jest taki sam, jak wkład w $\lfloor n/p \rfloor!$. Rozumowanie można więc powtórzyć, co prowadzi do poszukiwanego wzoru.

15. $(n!)^3$. Mamy $n!$ składników. Każdy składnik ma po jednym czynniku z każdego wiersza i kolumny, zatem jego wartość wynosi $(n!)^2$.

16. Składniki nie są zbieżne do zera, bo współczynniki są zbieżne do $1/e$.

17. Zapisz funkcję gamma w postaci granicy za pomocą równania (15).

$$\text{18. } \prod_{n \geq 1} \frac{n}{n - \frac{1}{2}} \frac{n}{n + \frac{1}{2}} = \frac{\Gamma(\frac{1}{2})\Gamma(\frac{3}{2})}{\Gamma(1)\Gamma(1)} = 2\Gamma(\frac{3}{2})^2.$$

[Pierwotny heurystyczny „dowód” Wallisa można znaleźć w książce D. J. Struika *Source Book in Mathematics* (Harvard University Press, 1969), 244–253].

19. Zamiana zmiennych $t = mt$, całkowanie przez części, indukcja.

20. [Dla porządku pokażemy najpierw, jak wyprowadzić podaną nierówność. Zaczynamy od prostej nierówności $1 + x \leq e^x$; podstawiamy $x = \pm t/n$ i podnosimy do n -tej potęgi, otrzymując $(1 \pm t/n)^n \leq e^{\pm t}$. Stąd $e^{-t} \geq (1 - t/n)^n = e^{-t}(1 - t/n)^n e^t \geq e^{-t}(1 - t/n)^n(1 + t/n)^n = e^{-t}(1 - t^2/n^2)^n \geq e^{-t}(1 - t^2/n)$ na mocy ćwiczenia 1.2.1–9].

Jeśli od podanej całki odejmiemy $\Gamma_m(x)$, to otrzymamy

$$\int_m^\infty e^{-t} t^{x-1} dt + \int_0^m \left(e^{-t} - \left(1 - \frac{t}{m}\right)^m \right) t^{x-1} dt.$$

Przy $m \rightarrow \infty$ pierwsza z tych całek dąży do zera, gdyż $t^{x-1} < e^{t/2}$ dla dużych t . Druga całka jest mniejsza niż

$$\frac{1}{m} \int_0^m t^{x+1} e^{-t} dt < \frac{1}{m} \int_0^\infty t^{x+1} e^{-t} dt \rightarrow 0.$$

21. Niech $c(n, j, k_1, k_2, \dots)$ oznacza odpowiedni współczynnik. W wyniku różniczkowania otrzymamy

$$\begin{aligned} c(n+1, j, k_1, \dots) &= c(n, j-1, k_1-1, k_2, \dots) + (k_1+1)c(n, j, k_1+1, k_2-1, k_3, \dots) \\ &\quad + (k_2+1)c(n, j, k_1, k_2+1, k_3-1, k_4, \dots) + \dots \end{aligned}$$

Równości $k_1 + k_2 + \dots = j$ i $k_1 + 2k_2 + \dots = n$ są zachowywane przez powyższy związek indukcyjny. Możemy wyłączyć czynnik $n!/(k_1!(1!)^{k_1} k_2!(2!)^{k_2} \dots)$ z każdego składnika po prawej stronie równości dla $c(n+1, j, k_1, \dots)$, pozostałe wówczas $k_1 + 2k_2 + 3k_3 + \dots = n+1$. (W dowodzie wygodnie założyć, że jest nieskończenie wiele k , choć widać, że $k_{n+1} = k_{n+2} = \dots = 0$).

W powyższym rozwiążaniu są wykorzystane standardowe metody, ale nie jest wyjaśnione, dlaczego wzór ma taką postać ani w jaki sposób go odkryto. Spróbujmy wyjaśnić to za pomocą rozumowania kombinatorycznego zaproponowanego przez H. S. Walla [Bull. Amer. Math. Soc. 44 (1938), 395–398]. Dla wygody oznaczmy $w_j = D_u^j w$, $u_k = D_x^k u$. Wówczas $D_x(w_j) = w_{j+1}u_1$, a $D_x(u_k) = u_{k+1}$. Te dwie reguły oraz wzór na pochodną iloczynu dają

$$D_x^1 w = w_1 u_1$$

$$D_x^2 w = (w_2 u_1 u_1 + w_1 u_2)$$

$$D_x^3 w = ((w_3 u_1 u_1 u_1 + w_2 u_2 u_1 + w_2 u_1 u_2) + (w_2 u_1 u_2 + w_1 u_3)) \text{ itd.}$$

Analogicznie możemy utworzyć tableau podziałów zbiorów:

$$\mathcal{D}^1 = \{1\}$$

$$\mathcal{D}^2 = (\{2\}\{1\} + \{2, 1\})$$

$$\mathcal{D}^3 = ((\{3\}\{2\}\{1\} + \{3, 2\}\{1\} + \{2\}\{3, 1\}) + (\{3\}\{2, 1\} + \{3, 2, 1\})) \text{ itd.}$$

Formalnie: dla zbioru $\{1, 2, \dots, n-1\}$ i jego podziału $a_1 a_2 \dots a_j$ definiujemy

$$\begin{aligned} \mathcal{D} a_1 a_2 \dots a_j &= \{n\} a_1 a_2 \dots a_j + (a_1 \cup \{n\}) a_2 \dots a_j \\ &\quad + a_1 (a_2 \cup \{n\}) \dots a_j + \dots + a_1 a_2 \dots (a_j \cup \{n\}). \end{aligned}$$

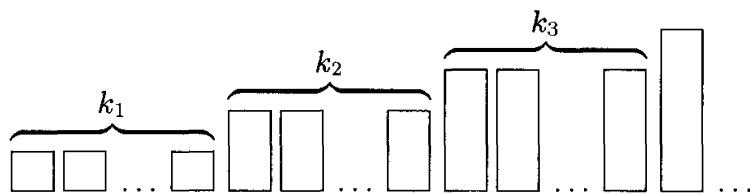
Ta reguła dokładnie odpowiada

$$\begin{aligned} D_x(w_j u_{r_1} u_{r_2} \dots u_{r_j}) &= w_{j+1} u_1 u_{r_1} u_{r_2} \dots u_{r_j} + w_j u_{r_1+1} u_{r_2} \dots u_{r_j} \\ &\quad + w_j u_{r_1} u_{r_2+1} \dots u_{r_j} + \dots + w_j u_{r_1} u_{r_2} \dots u_{r_j+1}, \end{aligned}$$

jeśli utożsamimy wyraz $w_j u_{r_1} u_{r_2} \dots u_{r_j}$ z podziałem $a_1 a_2 \dots a_j$, w którym zbiór a_t ma r_t elementów, $1 \leq t \leq j$. Istnieje zatem naturalne przekształcenie z \mathcal{D}^n na $D_x^n w$, a ponadto łatwo się przekonać, że \mathcal{D}^n zawiera każdy podział zbioru $\{1, 2, \dots, n\}$ dokładnie raz. (Zobacz ćwiczenie 1.2.6–64).

Z powyższych obserwacji wynika, że jeśli zgrupujemy wyrazy tak jak w $D_x^n w$, to otrzymamy sumę wyrazów $c(k_1, k_2, \dots) w_1^{k_1} u_1^{k_1} u_2^{k_2} \dots$, taką że $j = k_1 + k_2 + \dots$ i $n = k_1 + 2k_2 + \dots$, gdzie $c(k_1, k_2, \dots)$ jest liczbą podziałów zbioru $\{1, 2, \dots, n\}$ na j podzbiorów, w których k_t podzbiorów ma t elementów.

Pozostaje policzyć podziały. Rozważmy zestaw k_t pudełek o pojemności t :



Liczba sposobów ułożenia n różnych elementów w tych pudełkach jest współczynnikiem wielomianowym

$$\binom{n}{1, 1, \dots, 1, 2, 2, \dots, 2, 3, 3, \dots, 3, 4, \dots} = \frac{n!}{1!^{k_1} 2!^{k_2} 3!^{k_3} \dots}.$$

Żeby dostać $c(k_1, k_2, k_3, \dots)$, powinniśmy podzielić tę liczbę przez $k_1! k_2! k_3! \dots$, gdyż pudełka w każdej k_t -elementowej grupie są nieroróżnicjalne – można je spermutować na $k_t!$ sposobów i nie ma to wpływu na podział.

Pierwotny dowód Arbogasta [Du Calcul des Dérivations (Strasbourg: 1800), §52] został oparty na spostrzeżeniu, że $D_x^k u / k!$ jest współczynnikiem stojącym przy z^k w $u(x+z)$, a $D_u^j w / j!$ jest współczynnikiem stojącym przy y^j w $w(u+y)$, stąd współczynnik stojący przy z^n w $w(u(x+z))$ równa się

$$\frac{D_x^n w}{n!} = \sum_{j=0}^n \frac{D_u^j w}{j!} \sum_{\substack{k_1+k_2+\dots+k_n=j \\ k_1+2k_2+\dots+nk_n=n \\ k_1, k_2, \dots, k_n \geq 0}} \frac{j!}{k_1! k_2! \dots k_n!} \left(\frac{D_x^1 u}{1!} \right)^{k_1} \left(\frac{D_x^2 u}{2!} \right)^{k_2} \dots \left(\frac{D_x^n u}{n!} \right)^{k_n}.$$

Wzór Arbogasta został zapomniany na wiele lat i niezależnie odkryty przez F. Faà di Bruno [Quarterly J. Math. 1 (1857), 359–360], który zauważył, że wynik można również przedstawić jako wyznacznik

$$D_x^n = \det \begin{pmatrix} \binom{n-1}{0} u_1 & \binom{n-1}{1} u_2 & \binom{n-1}{2} u_3 & \dots & \binom{n-1}{n-2} u_{n-1} & \binom{n-1}{n-1} u_n \\ -1 & \binom{n-2}{0} u_1 & \binom{n-2}{1} u_2 & \dots & \binom{n-2}{n-3} u_{n-2} & \binom{n-2}{n-2} u_{n-1} \\ 0 & -1 & \binom{n-3}{0} u_1 & \dots & \binom{n-3}{n-4} u_{n-3} & \binom{n-3}{n-3} u_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -1 & \binom{0}{0} u_1 \end{pmatrix}$$

gdzie $u_j = (D_x^j u) D_u$ (obie strony tej równości są operatorami różniczkowymi, które należy zastosować do w). Uogólnienie wzoru Arbogasta na funkcje wielu zmiennych oraz bibliografię można znaleźć w artykule I. J. Gooda, *Annals of Mathematical Statistics* 32 (1961), 540–541.

22. Hipoteza $\lim_{n \rightarrow \infty} (n+x)!/(n! n^x) = 1$ jest prawdziwa dla całkowitych x . Jeśli liczba x jest na przykład dodatnia, rozpatrywana wielkość to $(1+1/n)(1+2/n)\dots(1+x/n)$, co rzeczywiście jest zbieżne do jedynki. Jeśli ponadto założymy, że $x! = x(x-1)!$, to hipoteza prowadzi nas do natychmiastowego wniosku

$$1 = \lim_{n \rightarrow \infty} \frac{(n+x)!}{n! n^x} = x! \lim_{n \rightarrow \infty} \frac{(x+1)\dots(x+n)}{n! n^x},$$

co jest równoważne definicji przedstawionej w tekście.

- 23.** $z(-z)!\Gamma(z) = \lim_{m \rightarrow \infty} \prod_{n=1}^m (1-z/n)^{-1}(1+z/n)^{-1}$ ze wzorów (13) i (15).
- 24.** $n^n/n! = \prod_{k=1}^{n-1} (k+1)^k/k^k \leq \prod_{k=1}^{n-1} e$; $n!/n^{n+1} = \prod_{k=1}^{n-1} k^{k+1}/(k+1)^{k+1} \leq \prod_{k=1}^{n-1} e^{-1}$.
- 25.** $x^{\underline{m+n}} = x^{\underline{m}}(x+m)^{\bar{n}}$; $x^{\overline{m+n}} = x^{\bar{m}}(x-m)^{\bar{n}}$. Te prawa obowiązują również wtedy, gdy liczby m i n nie są całkowite, wystarczy zastosować wzór (21).

1.2.6

1. n , bo każda kombinacja pomija dokładnie jeden przedmiot.
2. 1. Istnieje dokładnie jeden sposób wybrania niczego ze zbioru pustego.
3. $\binom{52}{13} = 635013559600$.
4. $2^4 \cdot 5^2 \cdot 7^2 \cdot 17 \cdot 23 \cdot 41 \cdot 43 \cdot 47$.
5. $(10+1)^4 = 10000 + 4(1000) + 6(100) + 4(10) + 1$.
6. $r = -3: \quad 1 \ -3 \ \ 6 \ -10 \ \ 15 \ -21 \ \ 28 \ -36 \ \dots$
 $r = -2: \quad 1 \ -2 \ \ 3 \ -4 \ \ 5 \ -6 \ \ 7 \ -8 \ \dots$
 $r = -1: \quad 1 \ -1 \ \ 1 \ -1 \ \ 1 \ -1 \ \ 1 \ -1 \ \dots$
7. $\lfloor n/2 \rfloor$ albo (równoważnie) $\lceil n/2 \rceil$. Z (3) wynika, że dla wartości mniejszych wspólnik dwumianowy jest ściśle rosnący, a dla wartości większych maleje do zera.
8. Niezerowe pozycje w każdym wierszu są takie same z lewej do prawej, jak z prawej do lewej.
9. Jeden, gdy n jest dodatnie lub równe zero; zero, gdy n jest ujemne.
10. (a), (b) i (f) wynikają wprost z (e); (c) i (d) wynikają z (a), (b) i wzoru (9). Wystarczy udowodnić (e). Zapiszmy $\binom{n}{k}$ w postaci ułamka, jak w (3) z czynnikami w liczniku i mianowniku. Pierwsze $k \bmod p$ czynników w mianowniku nie zawiera

czynnika p , a $k \bmod p$ pierwszych wyrazów w liczniku i mianowniku przystaje do odpowiadających im wyrazów w

$$\binom{n \bmod p}{k \bmod p},$$

od których różnią się o p razy. (Mając do czynienia z liczbami nie będącymi wielokrotnościami p , możemy w liczniku i mianowniku wykonywać działania modulo p , bo jeśli $a \equiv c$ i $b \equiv d$ i $a/b, c/d$ są całkowite, to $a/b \equiv c/d$). Zostaje $k - k \bmod p$ czynników, które dzielimy na $\lfloor k/p \rfloor$ grup po p kolejnych wartości w każdej. Każda grupa zawiera dokładnie jedną wielokrotność p . Pozostałe $p - 1$ czynników grupy przystaje (modulo p) do $(p-1)!$, zatem upraszczają się w liczniku i mianowniku. Pozostaje nam poradzić sobie z $\lfloor k/p \rfloor$ wielokrotnościami p w liczniku i mianowniku. Dzielimy każdą z nich przez p , otrzymując współczynnik dwumianowy

$$\binom{\lfloor (n-k \bmod p)/p \rfloor}{\lfloor k/p \rfloor}.$$

Dla $k \bmod p \leq n \bmod p$ jego wartość wynosi

$$\binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor},$$

tak jak chcieliśmy. Jeśli $k \bmod p > n \bmod p$, to drugi czynnik $\binom{n \bmod p}{k \bmod p}$ równa się zero, zatem twierdzenie też jest prawdziwe. [American J. Math. 1 (1878), 229–230; zobacz też N. J. Fine, AMM 54 (1947), 589–592].

11. Jeśli $a = a_r p^r + \dots + a_0$, $b = b_r p^r + \dots + b_0$ i $a + b = c_r p^r + \dots + c_0$, to wartość n (zgodnie z ćwiczeniem 1.2.5–12 i wzorem (5)) wynosi

$$(a_0 + \dots + a_r + b_0 + \dots + b_r - c_0 - \dots - c_r)/(p-1).$$

W wyniku przeniesienia c_j zmniejsza się o p , a c_{j+1} zwiększa się o 1, powodując w powyższym wzorze zmianę wartości o 1. [Podobne własności zachodzą dla współczynników q -mianowych i fibomianowych; zobacz Knuth, Wilf, Crelle 396 (1989), 212–219].

12. Na mocy dowolnego z dwóch ostatnich ćwiczeń n musi być o 1 mniejsze od potęgi dwójką. Ogólnie, $\binom{n}{k}$ nie dzieli się przez liczbę pierwszą p ($0 \leq k \leq n$) wtedy i tylko wtedy, gdy $n = ap^m - 1$, $1 \leq a < p$, $m \geq 0$.

$$\begin{aligned} 14. \quad 24 \binom{n+1}{5} + 36 \binom{n+1}{4} + 14 \binom{n+1}{3} + \binom{n+1}{2} \\ = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30} = \frac{n(n+1)(n+\frac{1}{2})(3n^2+3n-1)}{15}. \end{aligned}$$

15. Indukcja i wzór (9).

17. Możemy założyć, że r i s są dodatnimi liczbami całkowitymi. Dla dowolnego x

$$\begin{aligned} \sum_n \binom{r+s}{n} x^n &= (1+x)^{r+s} = \sum_k \binom{r}{k} x^k \sum_m \binom{s}{m} x^m \\ &= \sum_k \binom{r}{k} x^k \sum_n \binom{s}{n-k} x^{n-k} = \sum_n \left(\sum_k \binom{r}{k} \binom{s}{n-k} \right) x^n. \end{aligned}$$

Zatem współczynniki przy x^n muszą być identyczne.

21. Lewa strona jest wielomianem stopnia n , prawa jest wielomianem stopnia $m+n+1$. Wielomiany są równe w $n+1$ punktach, ale to nie wystarcza, żeby udowodnić ich równość (aczkolwiek dla $m=0$ dowodzi to, że obie strony są wielokrotnościami pewnego wielomianu; rzeczywiście, w przypadku $m=0$ okazuje się, że równanie jest tożsamością ze względu na s , bo to po prostu jest to równanie (11)).

22. Założmy, że $n > 0$. Wówczas k -ty wyraz równa się

$$\begin{aligned} \frac{1}{n!} \binom{n}{k} \prod_{0 < j < k} (r - tk - j) \prod_{0 \leq j < n-k} (n-1-r+tk-j) \\ = \frac{(-1)^{k-1}}{n!} \binom{n}{k} \prod_{0 < j < k} (-r+tk+j) \prod_{k \leq j < n} (-r+tk+j) \end{aligned}$$

a dwa iloczyny dają wielomian zmiennej k stopnia $n-1$. Stąd na mocy (34) suma po k wynosi zero.

24. Dowód przez indukcję względem n . Jeżeli $n \leq 0$, to teza jest oczywista. Dla $n > 0$ dowodzimy przez indukcję po całkowitych $m \geq 0$, korzystając z dwóch poprzednich ćwiczeń oraz tezy dla $n-1$, że twierdzenie zachodzi dla $(r, n-r+nt+m, t, n)$. W ten sposób uzasadniamy prawdziwość tożsamości (r, s, t, n) dla nieskończoność wielu s , co kończy dowód, bo obie strony są wielomianami zmiennej s .

25. Za pomocą kryterium ilorazowego i prostych oszacowań dla dużych k udowadniamy zbieżność. (Można także posłużyć się teorią zmiennej zespolonej, by udowodnić analityczność funkcji w sąsiedztwie $x=1$). Mamy

$$\begin{aligned} 1 &= \sum_{k,j} (-1)^j \binom{k}{j} \binom{r-jt}{k} \frac{r}{r-jt} w^k = \sum_j (-1)^j \frac{r}{r-jt} \sum_k \binom{k}{j} \binom{r-jt}{k} w^k \\ &= \sum_j \frac{(-1)^j r}{r-jt} \sum_k \binom{r-jt}{j} \binom{r-jt-j}{k-j} w^k = \sum_j (-1)^j A_j(r, t) (1+w)^{r-jt-j} w^j. \end{aligned}$$

Niech teraz $x = 1/(1+w)$, $z = -w/(1+w)^{1+t}$. Powyższy dowód zawdzięczamy H. W. Gouldowi [AMM 63 (1956), 84–91]. Bardziej ogólne wzory omawiamy w ćwiczeniach 2.3.4.4–33 i 4.7–22.

26. Możemy zacząć od tożsamości (35) w postaci

$$\sum_j (-1)^j \binom{k}{j} \binom{r-jt}{k} = t^k$$

i kontynuować jak w ćwiczeniu 25. Inny sposób to wyznaczenie pochodnej wzoru z tego ćwiczenia względem x . Dostajemy

$$\sum_k k A_k(r, t) z^k = z \frac{d(x^r)}{dz} = \frac{(x^{t+1} - x^t) r x^r}{(t+1)x^{t+1} - tx^t},$$

skąd otrzymujemy wartość

$$\sum_k \left(1 - \frac{t}{r} k\right) A_k(r, t) z^k.$$

27. Dowód wzoru (26): należy pomnożyć szereg będący rozwinięciem $x^{r+1}/((t+1)x-t)$ przez szereg będący rozwinięciem x^s , otrzymując w wyniku szereg będący rozwinięciem $x^{r+s+1}/((t+1)x-t)$, w którym współczynniki stojące przy z można przymierzyć do współczynników szeregu będącego rozwinięciem $x^{(r+s)+1}/((t+1)x-t)$.

28. Oznaczmy lewą stronę równości przez $f(r, s, t, n)$. Wówczas

$$\binom{r+s}{n} + tf(r-t-1, s+t, t, n-1) = f(r, s, t, n)$$

z tożsamości

$$\sum_k \binom{r+tk}{k} \binom{s-tk}{n-k} \frac{r}{r+tk} + \sum_k \binom{r+tk}{k} \binom{s-tk}{n-k} \frac{tk}{r+tk} = f(r, s, t, n).$$

29. $(-1)^k \binom{n}{k} / n! = (-1)^k / (k!(n-k)!) = (-1)^n / \prod_{\substack{0 \leq j \leq n \\ j \neq k}} (k-j).$

30. Stosujemy (7), (6) i (19), otrzymując

$$\sum_{k \geq 0} \binom{-m-2k-1}{n-m-k} \binom{2k+1}{k} \frac{(-1)^{n-m}}{2k+1}.$$

Teraz możemy skorzystać z (26) przy $(r, s, t, n) = (1, m-2n-1, -2, n-m)$, co daje

$$(-1)^{n-m} \binom{-m}{n-m} = \binom{n-1}{n-m}.$$

Ten wzór pokrywa się z poprzednim wynikiem dla dodatnich n , ale dla $n = 0$ nasza nowa odpowiedź jest poprawna, a odpowiedź $\binom{n-1}{m-1}$ nie. Zyskaliśmy nawet więcej, bo odpowiedź $\binom{n-1}{n-m}$ jest poprawna dla $n \geq 0$ i *wszystkich* całkowitych m .

31. [Jako pierwszy postać zamkniętą tej sumy otrzymał J. F. Pfaff, *Nova Acta Acad. Scient. Petr.* **11** (1797), 38–57] Mamy

$$\begin{aligned} & \sum_k \sum_j \binom{m-r+s}{k} \binom{n+r-s}{n-k} \binom{r}{m+n-j} \binom{k}{j} \\ &= \sum_j \sum_k \binom{m-r+s}{j} \binom{n+r-s}{n-k} \binom{r}{m+n-j} \binom{m-r+s-j}{k-j} \\ &= \sum_j \binom{m-r+s}{j} \binom{r}{m+n-j} \binom{m+n-j}{n-j}. \end{aligned}$$

Zamieniając $\binom{m+n-j}{n-j}$ na $\binom{m+n-j}{m}$ i ponownie stosując (20), otrzymujemy

$$\sum_j \binom{m-r+s}{j} \binom{r}{m} \binom{r-m}{n-j} = \binom{r}{m} \binom{s}{n}.$$

32. Zamień x na $-x$ we wzorze (44).

33, 34. [Giornale di Mat. Battaglini **31** (1893), 291–313; **33** (1895), 179–182] Mamy $x^{\bar{n}} = n! \binom{x+n-1}{n}$. Stąd równanie można doprowadzić do postaci

$$\binom{x+y+n-1}{n} = \sum_k \binom{x+(1-z)k}{k} \binom{y-1+nz+(n-k)(1-z)}{n-k} \frac{x}{x+(1-z)k},$$

co jest szczególnym przypadkiem wzoru (26). Analogiczny wzór $(x+y)^n = \sum_k \binom{n}{k} x(x-kz-1)^{k-1} (y+kz)^{n-k}$ zawdzięczamy Rothe'owi [*Formulæ de Serierum Reversione* (Leipzig: 1793), 18].

35. Dla przykładu pokażemy pierwszy wzór:

$$\sum_k (-1)^{n+1-k} \left(n \begin{bmatrix} n \\ k \end{bmatrix} + \begin{bmatrix} n \\ k-1 \end{bmatrix} \right) x^k = -nx^n + xx^n = x^{n+1}.$$

36. Ze wzoru (13), przy założeniu, że n jest nieujemną liczbą całkowitą, mamy odpowiednio 2^n i δ_{n0} .

37. 2^{n-1} , dla $n > 0$. (Suma wyrazów parzystych i suma wyrazów nieparzystych są równe, więc każda z nich równa się połowie sumy wszystkich wyrazów).

38. Niech $\omega = e^{2\pi i/m}$. Wówczas

$$\sum_{0 \leq j < m} (1 + \omega^j)^n \omega^{-jk} = \sum_t \sum_{0 \leq j < m} \binom{n}{t} \omega^{j(t-k)}.$$

Teraz

$$\sum_{0 \leq j < m} \omega^{rj} = m [r \equiv 0 \pmod{m}]$$

(suma ciągu geometrycznego), zatem prawa suma to $m \sum_{t \bmod m=k} \binom{n}{t}$. Pierwotną sumę po lewej stronie równości można zapisać następująco

$$\sum_{0 \leq j < m} (\omega^{-j/2} + \omega^{j/2})^n \omega^{j(n/2-k)} = \sum_{0 \leq j < m} \left(2 \cos \frac{j\pi}{m} \right)^n \omega^{j(n/2-k)}.$$

Ponieważ wiemy, że wartość jest rzeczywista, możemy ograniczyć się do części rzeczywistych, otrzymując dowodzony wzór.

Przypadki $m = 3$ i $m = 5$ mają szczególne własności, omówione w CMath, ćwiczenia 5.75 i 6.57.

39. $n!$ i $\delta_{n0} - \delta_{n1}$. (Suma wiersza w trójkącie drugiego rodzaju nie jest taka prosta do policzenia; w ćwiczeniu 64 okaże się, że $\sum_k \binom{n}{k}$ to liczba sposobów podziału n -elementowego zbioru na rozłączne podzbiory, co jest jednocześnie liczbą relacji równoważności na zbiorze $\{1, 2, \dots, n\}$).

40. Dowód (c): Całkujemy przez części

$$B(x+1, y) = -\frac{t^x (1-t)^y}{y} \Big|_0^1 + \frac{x}{y} \int_0^1 t^{x-1} (1-t)^y dt.$$

Teraz wystarczy skorzystać z (b).

41. $m^x B(x, m+1) \rightarrow \Gamma(x)$, gdy $m \rightarrow \infty$, niezależnie od tego, czy m przebiega liczby całkowite, czy nie (z monotoniczności). Stąd $(m+y)^x B(x, m+y+1) \rightarrow \Gamma(x)$ i $(m/(m+y))^x \rightarrow 1$.

42. $1/((r+1)B(k+1, r-k+1))$, jeśli przyjmiemy definicję jak w ćwiczeniu 41(b). W ogólności, gdy z i w są dowolnymi liczbami zespolonymi, definiujemy

$$\binom{z}{w} = \lim_{\zeta \rightarrow z} \lim_{\omega \rightarrow w} \frac{\zeta!}{\omega! (\zeta - \omega)!}, \quad \text{gdzie } \zeta! = \Gamma(\zeta + 1);$$

wartość ta jest nieskończona, gdy z jest ujemną liczbą całkowitą, a w nie jest liczbą całkowitą.

Przy takiej definicji warunek symetrii (6) zachodzi dla wszystkich zespolonych n i k oprócz przypadku, gdy n jest ujemną liczbą całkowitą, a k jest liczbą całkowitą.

Równości (7), (9) i (20) nigdy nie są fałszywe, aczkolwiek mogą być nieokreślone, jak w przypadkach: $0 \cdot \infty$ lub $\infty + \infty$. Równanie (17) przyjmuje postać:

$$\binom{z}{w} = \frac{\sin \pi(w-z-1)}{\sin \pi z} \binom{w-z-1}{w}.$$

Możemy nawet rozszerzyć twierdzenie dwumianowe (13) i splot Vandermonde'a (21), otrzymując $\sum_k \binom{r}{\alpha+k} z^{\alpha+k} = (1+z)^r$ i $\sum_k \binom{r}{\alpha+k} \binom{s}{\beta-k} = \binom{r+s}{\alpha+\beta}$. Te równości zachodzą dla wszystkich zespolonych r, s, z, α, β , jeśli tylko szeregi są zbieżne oraz przy założeniu, że odpowiednio zdefiniujemy potęgi zespolone. [Zobacz L. Ramshaw, *Inf. Proc. Letters* 6 (1977), 223–226].

43. $\int_0^1 dt/(t^{1/2}(1-t)^{1/2}) = 2 \int_0^1 du/(1-u^2)^{1/2} = 2 \arcsin u|_0^1 = \pi.$

45. Dla dużych r , $\frac{1}{k\Gamma(k)} \sqrt{\frac{r}{r-k}} \frac{1}{e^k} \frac{(1-k/r)^k}{(1-k/r)^r} \rightarrow \frac{1}{\Gamma(k+1)}$.

46. $\sqrt{\frac{1}{2\pi} \left(\frac{1}{x} + \frac{1}{y}\right)} \left(1 + \frac{y}{x}\right)^x \left(1 + \frac{x}{y}\right)^y$ oraz $\binom{2n}{n} \approx 4^n/\sqrt{\pi n}$.

47. Wszystkie trzy wielkości są równe δ_{k0} dla $k \leq 0$, a zwiększenie k o 1 pociąga za sobą wzrost każdej z nich $(r-k)(r-\frac{1}{2}-k)/(k+1)^2$ razy. Stąd dla $r = -\frac{1}{2}$ mamy $\binom{-1/2}{k} = (-1/4)^k \binom{2k}{k}$.

48. Twierdzenie można udowodnić przez indukcję, korzystając z faktu, że dla $n > 0$

$$0 = \sum_k \binom{n}{k} (-1)^k = \sum_k \binom{n}{k} \frac{(-1)^k k}{k+x} + \sum_k \binom{n}{k} \frac{(-1)^k x}{k+x}.$$

Można też inaczej:

$$B(x, n+1) = \int_0^1 t^{x-1} (1-t)^n dt = \sum_k \binom{n}{k} (-1)^k \int_0^1 t^{x+k-1} dt.$$

(W istocie suma w tezie twierdzenia równa się $B(x, n+1)$ również dla niecałkowitych n , jeżeli szeregi są zbieżne).

49. $\binom{r}{m} = \sum_k \binom{r}{k} \binom{-r}{m-2k} (-1)^{m+k}$, całkowite m . (Zobacz ćwiczenie 17).

50. k -ty składnik ma wartość $\binom{n}{k} (-1)^{n-k} (x-kz)^{n-1} x$. Zastosuj wzór (34).

51. Po prawej stronie mamy

$$\begin{aligned} & \sum_k \binom{n}{n-k} x (x-kz)^{k-1} \sum_j \binom{n-k}{j} (x+y)^j (-x+kz)^{n-k-j} \\ &= \sum_j \binom{n}{j} (x+y)^j \sum_k \binom{n-j}{n-j-k} x (x-kz)^{k-1} (-x+kz)^{n-k-j} \\ &= \sum_{j \leq n} \binom{n}{j} (x+y)^j 0^{n-j} = (x+y)^n. \end{aligned}$$

W ten sam sposób można udowodnić sumę Torelliego (ćwiczenie 34).

Inny elegancki dowód wzoru Abela opiera się na obserwacji, że można go łatwo przekształcić w bardziej ogólną symetryczną tożsamość wyprowadzoną w ćwiczeniu 2.3.4.4–29:

$$\sum_k \binom{n}{k} x (x+kz)^{k-1} y (y+(n-k)z)^{n-k-1} = (x+y)(x+y+nz)^{n-1}.$$

Twierdzenie Abela ma jeszcze szersze uogólnienie autorstwa A. Hurwitza [Acta Mathematica **26** (1902), 199–203]:

$$\sum x(x + \epsilon_1 z_1 + \cdots + \epsilon_n z_n)^{\epsilon_1 + \cdots + \epsilon_n - 1} (y - \epsilon_1 z_1 - \cdots - \epsilon_n z_n)^{n - \epsilon_1 - \cdots - \epsilon_n} = (x + y)^n$$

gdzie sumujemy niezależnie po wszystkich 2^n wyborach $\epsilon_1, \dots, \epsilon_n = 0$ lub 1. Ten wzór jest tożsamością ze względu na x, y, z_1, \dots, z_n i stanowi przypadek szczególny wzoru Abela dla $z_1 = z_2 = \cdots = z_n$. Wzór Hurwitza wynika z ćwiczenia 2.3.4.4–30.

52. $\sum_{k \geq 0} (k+1)^{-2} = \pi^2/6$. [M. L. J. Hautus zauważył, że ta suma jest zbieżna bezwzględnie dla wszystkich zespolonych x, y, z, n , jeśli tylko $z \neq 0$, bo wyrazy dla dużych k są zawsze rzędu $1/k^2$. Ta zbieżność jest jednostajna w przedziałach ograniczonych, zatem możemy różniczkować szereg wyraz po wyrazie. Jeżeli $f(x, y, n)$ jest wartością sumy dla $z = 1$, to $(\partial/\partial y)f(x, y, n) = nf(x, y, n-1)$ oraz $(\partial/\partial x)f(x, y, n) = nf(x-1, y+1, n-1)$. Powyższe wzory pasują do $f(x, y, n) = (x+y)^n$, ale w istocie ta równość zachodzi rzadko (jeśli w ogóle), jeżeli suma nie jest skończona. Ponadto pochodna ze względu na z jest prawie zawsze niezerowa].

53. Punkt (b): podstawiamy $r = \frac{1}{2}$ i $s = -\frac{1}{2}$ do wyniku punktu (a).

54. Wstaw do macierzy minusy według poniższego wzoru:

$$\begin{pmatrix} 1 & -0 & 0 & -0 \\ -1 & 1 & -0 & 0 \\ 1 & -2 & 1 & -0 \\ -1 & 3 & -3 & 1 \end{pmatrix}$$

To odpowiada mnożeniu a_{ij} przez $(-1)^{i+j}$. Na mocy wzoru (33) wynik jest szukaną macierzą odwrotną.

55. Powstawiaj minusy, jak w poprzednim ćwiczeniu. Macierz otrzymana z liczb pierwszego rodzaju będzie macierzą odwrotną macierzy liczb drugiego rodzaju i vice versa (wzór (47)).

56. 012 013 023 123 014 024 124 034 134 234 015 025 125 035 135 235 045 145 245 345 016. Dla ustalonego c liczby a i b przebiegają kombinacje c obiektów branych jednocześnie po dwa. Dla ustalonych c i b liczba a przebiega kombinacje b obiektów branych po jednym naraz.

W podobny sposób możemy przedstawić wszystkie liczby w postaci $n = \binom{a}{1} + \binom{b}{2} + \binom{c}{3} + \binom{d}{4}$, gdzie $0 \leq a < b < c < d$. Ciąg reprezentacji zaczyna się od 0123 0124 0134 0234 1234 0125 0135 0235 ... Reprezentację kombinatoryczną możemy znaleźć za pomocą metody „zachłannej” – najpierw wybieramy największe możliwe d , potem największe możliwe c dla $n - \binom{d}{4}$ itd. [W punkcie 7.2.1 omawiamy dalsze własności takiej reprezentacji].

58. Przez indukcję, mamy

$$\binom{n}{k}_q = \binom{n-1}{k}_q + \binom{n-1}{k-1}_q q^{n-k} = \binom{n-1}{k}_q q^k + \binom{n-1}{k-1}_q.$$

Wnioskujemy, że q -uogólnieniem wzoru (21) jest

$$\sum_k \binom{r}{k}_q \binom{s}{n-k}_q q^{(r-k)(n-k)} = \sum_k \binom{r}{k}_q \binom{s}{n-k}_q q^{(s-n+k)k} = \binom{r+s}{n}_q.$$

Dzięki tożsamości $1 - q^t = -q^t(1 - q^{-t})$ łatwo uogólnić wzór (17) do postaci:

$$\binom{r}{k}_q = (-1)^k \binom{k-r-1}{k}_q q^{kr-k(k-1)/2}.$$

Współczynniki q -mianowe pojawiają się w wielu niespodziewanych miejscach; zobacz na przykład punkt 5.1.2, a także uwagi autora w *J. Combinatorial Theory A*10 (1971), 178–180.

Pozyteczne fakty: Gdy n jest nieujemną liczbą całkowitą, $\binom{n}{k}_q$ jest wielomianem stopnia $k(n-k)$ zmiennej q o współczynnikach całkowitych i nieujemnych. Spełnia ponadto prawa

$$\binom{n}{k}_q = \binom{n}{n-k}_q = q^{k(n-k)} \binom{n}{k}_{q^{-1}}.$$

Jeżeli lewą stronę zastąpimy przez $\prod_{k \geq 0} ((1+q^k x)/(1+q^{n+k} x))$, to twierdzenie q -mianowe zachodzi dla dowolnej liczby rzeczywistej n , gdy $|q| < 1$ i $|x| < 1$. Dzięki własnościom szeregów potęgowych wystarczy sprawdzić równość jedynie dla całkowitych dodatnich n , ponieważ możemy podstawić $q^n = y$ i stwierdzić, że równość zachodzi dla nieskończonie wielu wartości y . Zmieniając znak górnego indeksu w twierdzeniu q -mianowym, otrzymujemy wzór

$$\prod_{k \geq 0} \frac{(1-q^{k+r+1}x)}{(1-q^kx)} = \sum_k \binom{-r-1}{k}_q q^{k(k-1)/2} (-q^{r+1}x)^k = \sum_k \binom{k+r}{k}_q x^k,$$

autorstwa Cauchy'ego [*Comptes Rendus Acad. Sci. Paris* 17 (1843), 523] i Heinego [*Crelle* 34 (1847), 285–328]. Więcej informacji można znaleźć w: G. Gasper, M. Rahman, *Basic Hypergeometric Series* (Cambridge Univ. Press, 1990).

59. $(n+1)\binom{n}{k} - \binom{n}{k+1}$.

60. $\binom{n+k-1}{k}$. Łatwo zapamiętać ten wzór, bo jest to

$$\frac{n(n+1)\dots(n+k-1)}{k(k-1)\dots 1},$$

co przypomina wzór (2), z tym że wartości w liczniku rosną, a nie maleją. Ciekawa metoda dowodu polega na zauważeniu, że w istocie chcemy policzyć, ile jest rozwiązań całkowitoliczbowych (a_1, \dots, a_k) nierówności $1 \leq a_1 \leq a_2 \leq \dots \leq a_k \leq n$. Równoważnie możemy przyjąć nierówność $0 < a_1 < a_2 + 1 < \dots < a_k + k - 1 < n + k$. Liczba rozwiązań nierówności

$$0 < b_1 < b_2 < \dots < b_k < n + k$$

jest równa liczbie wyborów k różnych elementów ze zbioru $\{1, 2, \dots, n+k-1\}$. (To sztuczka autorstwa H. F. Scherka, *Crelle* 3 (1828), 97; co ciekawe, została podana również przez W. A. Förstemanna w tym samym czasopiśmie, 13 (1835), 237, który stwierdził „Można nieopatrznie dać wiarę, że ten pomysł jest od dawna znany, jednakowoż nie znalazłem go nigdzie, choć przestudiowałem wiele pism w tej materii”).

61. Jeżeli a_{mn} jest szukaną wartością, to na mocy (46) i (47) zachodzi $a_{mn} = na_{m(n-1)} + \delta_{mn}$. Stąd odpowiedią jest $[n \geq m] n!/m!$. Ten sam wzór można łatwo uzyskać przez inwersję wzoru (56).

62. Skorzystaj z tożsamości z ćwiczenia 31 dla $(m, n, r, s, k) \leftarrow (m+k, l-k, m+n, n+l, j)$ i pożongluj silniami:

$$\begin{aligned} & \sum_k (-1)^k \binom{l+m}{l+k} \binom{m+n}{m+k} \binom{n+l}{n+k} \\ &= \sum_{j,k} (-1)^k \binom{l+m}{l+k} \binom{l+k}{j} \binom{m-k}{l-k-j} \binom{m+n+j}{m+l} \\ &= \sum_{j,k} (-1)^k \binom{2l-2j}{l-j+k} \frac{(m+n+j)!}{(2l-2j)! j! (m-l+j)! (n+j-l)!}. \end{aligned}$$

Suma po k znika poza przypadkiem $j = l$.

Przypadek $l = m = n$ został opublikowany przez A. C. Dixona [*Messenger of Math.* **20** (1891), 79–80], który 12 lat później podał postać ogólną [*Proc. London Math. Soc.* **35** (1903), 285–289]. W tym czasie jednak L. J. Rogers opublikował bardziej ogólny wzór [*Proc. London Math. Soc.* **26** (1895), 15–32, §8]. Zobacz także prace P. A. MacMahona, *Quarterly Journal of Pure and Applied Math.* **33** (1902), 274–288, oraz Johna Dougalla, *Proc. Edinburgh Math. Society* **25** (1907), 114–132. Odpowiednie tożsamości q -mianowe mają postać

$$\begin{aligned} & \sum_k \binom{m-r+s}{k}_q \binom{n+r-s}{n-k}_q \binom{r+k}{m+n}_q q^{(m-r+s-k)(n-k)} = \binom{r}{m}_q \binom{s}{n}_q, \\ & \sum_k (-1)^k \binom{l+m}{l+k}_q \binom{m+n}{m+k}_q \binom{n+l}{n+k}_q q^{(3k^2-k)/2} = \frac{(l+m+n)!_q}{l!_q m!_q n!_q}, \end{aligned}$$

gdzie $n!_q = \prod_{k=1}^n (1+q+\cdots+q^{k-1})$.

63. Zobacz CMath, ćwiczenia 5.83 i 5.106.

64. Niech $f(n, m)$ będzie liczbą podziałów zbioru $\{1, 2, \dots, n\}$ na m części. Widać, że $f(1, m) = \delta_{1m}$. Dla $n > 1$ mamy dwa typy podziałów: (a) Jeden z podzbiorów jest jednoelementowy i zawiera wyłącznie n ; takie podziały można skonstruować na $f(n-1, m-1)$ sposobów. (b) Element n należy do jakiegoś większego podzbioru. Jest m sposobów dołożenia m do każdego m -podziału zbioru $\{1, 2, \dots, n-1\}$, stąd istnieje $mf(n-1, m)$ sposobów skonstruowania podziałów drugiego typu. Otrzymujemy wniosek $f(n, m) = f(n-1, m-1) + mf(n-1, m)$, skąd przez indukcję $f(n, m) = \{ \begin{smallmatrix} n \\ m \end{smallmatrix} \}$.

65. Zobacz AMM **99** (1992), 410–422.

66. Zauważmy najpierw, że $\binom{z}{n+1} \leq \binom{y}{n+1}$. To jest jasne, gdy $z \geq n$, bo $z \leq y$. Założymy zatem, że $n-1 \leq z \leq n$, wówczas $\binom{z}{n+1} \leq 0 \leq \binom{y}{n+1}$. Stąd $\binom{z+1}{n+1} = \binom{z}{n+1} + \binom{z}{n} \leq \binom{y}{n+1} + \binom{z}{n} = \binom{x}{n+1}$ i mamy $x \geq z+1$.

Teraz możemy dowieść, że każdy wyraz sumy

$$\binom{x}{n+1} - \binom{y}{n+1} = \sum_{k \geq 0} \binom{z-k}{n-k} t_k, \quad t_k = \binom{x-z-1+k}{k+1} - \binom{y-z-1+k}{k+1},$$

jest nieujemny. Współczynnik $\binom{z-k}{n-k}$ jest nieujemny, bo $z \geq n-1$; podobnie $\binom{x-z-1+k}{k+1}$, bo $x \geq z+1$. Stąd $z \leq y \leq x$ pociąga za sobą $\binom{y-z-1+k}{k+1} \leq \binom{x-z-1+k}{k+1}$.

Teza jest oczywista, gdy $x = y$ i $z = n-1$. W pozostałych przypadkach

$$\binom{x}{n} - \binom{y}{n} - \binom{z}{n-1} = \sum_{k \geq 0} \binom{z-k}{n-1-k} (t_k - \delta_{k0}) = \sum_{k \geq 0} \frac{n-k}{z-n+1} \binom{z-k}{n-k} (t_k - \delta_{k0})$$

jest mniejsze od zera lub równe

$$\sum_{k \geq 0} \frac{n-1}{z-n+1} \binom{z-k}{n-k} (t_k - \delta_{k0}) = \frac{n-1}{z-n+1} \left(\binom{x}{n+1} - \binom{y}{n+1} - \binom{z}{n} \right) = 0,$$

jako że $t_0 - 1 = x - y - 1 \leq 0$. [L. Lovász, *Combinatorial Problems and Exercises*, Problem 13.31(a)].

67. Dla $k > 0$ z ćwiczenia 1.2.5–24 mamy nieco ostrzejsze (ale trudniejsze do zapamiętania) ograniczenie $\binom{n}{k} = n^k/k! \leq n^k/k! \leq \frac{1}{e} \left(\frac{ne}{k}\right)^k \leq \left(\frac{ne}{k+1}\right)^k$. Analogicznym ograniczeniem dolnym jest $\binom{n}{k} \geq \left(\frac{(n-k)e}{k}\right)^k \frac{1}{e^k}$.

68. Niech $t_k = k \binom{n}{k} p^k (1-p)^{n+1-k}$. Wówczas $t_k - t_{k+1} = \binom{n}{k} p^k (1-p)^{n-k} (k - np)$, zatem suma jest równa

$$\sum_{k < \lceil np \rceil} (t_{k+1} - t_k) + \sum_{k \geq \lceil np \rceil} (t_k - t_{k+1}) = 2t_{\lceil np \rceil}.$$

[De Moivre opublikował tę tożsamość w *Miscellanea Analytica* (1730), 101, dla przypadków, gdy np jest całkowite. H. Poincaré udowodnił wzór ogólny w swoim *Calcul des Probabilités* (1896), 56–60. Ciekawa historia tej tożsamości, wraz z mnóstwem podobnych wzorów, została opisana przez P. Diaconisa i S. Zabellę w *Statistical Science* 6 (1991), 284–302].

1.2.7

1. 0, 1 i $3/2$.
2. Zamień każdy z wyrazów $1/(2^m + k)$ na górne ograniczenie $1/2^m$.
3. $H_{2^m-1}^{(r)} \leq \sum_{0 \leq k < m} 2^k / 2^{kr}$; $2^{r-1}/(2^{r-1} - 1)$ jest górnym ograniczeniem.
4. (b) i (c).
5. 9.78760 60360 44382 ...
6. Indukcja oraz wzór 1.2.6–(46).
7. $T(m+1, n) - T(m, n) = 1/(m+1) - 1/(mn+1) - \dots - 1/(mn+n) \leq 1/(m+1) - (1/(mn+n) + \dots + 1/(mn+n)) = 1/(m+1) - n/(mn+n) = 0$. Wyrażenie przyjmuje wartość maksymalną dla $m = n = 1$, a do wartości minimalnej zbliża się dla bardzo dużych m i n . Na mocy wzoru (3) kresem dolnym jest γ i wartość ta nigdy nie zostaje osiągnięta. Uogólnienie tego wyniku można znaleźć w *AMM* 70 (1963), 575–577.
8. Ze wzoru Stirlinga $\ln n!$ to w przybliżeniu $(n + \frac{1}{2}) \ln n - n + \ln \sqrt{2\pi}$, natomiast $\sum_{k=1}^n H_k$ to w przybliżeniu $(n+1) \ln n - n(1-\gamma) + (\gamma + \frac{1}{2})$; różnica w przybliżeniu wynosi $\gamma n + \frac{1}{2} \ln n + 0.158$.
9. $-1/n$.
10. Rozpisz lewą stronę na dwie sumy, zamień k na $k+1$ w drugiej sumie.
11. $2 - H_n/n - 1/n$, dla $n > 0$.
12. 1.000... jest wartością dokładną do ponad trzechsetnego miejsca po przecinku.
13. Zastosuj indukcję, tak jak w dowodzie twierdzenia A, albo analizę: zróżniczkuj ze względu na x oraz oblicz dla $x = 1$.
14. Zobacz punkt 1.2.3, przykład 2. Druga suma to $\frac{1}{2}(H_{n+1}^2 - H_{n+1}^{(2)})$.
15. $\sum_{j=1}^n (1/j) \sum_{k=j}^n H_k$ daje się policzyć za pomocą wzorów przedstawionych w tekście; wychodzi $(n+1)H_n^2 - (2n+1)H_n + 2n$.

16. $H_{2n-1} - \frac{1}{2}H_{n-1}$.

17. *Rozwiązań pierwsze* (elementarne). Przyjmując, że mianownik jest równy $(p-1)!$, co jest wielokrotnością prawdziwego mianownika, ale nie p , musimy pokazać jedynie, że odpowiadający mu licznik $(p-1)!/1 + (p-1)!/2 + \dots + (p-1)!/(p-1)$, jest wielokrotnością p . Modulo p , $(p-1)!/k \equiv (p-1)! k'$, gdzie k' można wyznaczyć ze związku $kk' \bmod p = 1$. Zbiór $\{1', 2', \dots, (p-1)'\}$ to po prostu zbiór $\{1, 2, \dots, p-1\}$, zatem licznik przystaje do $(p-1)!(1+2+\dots+p-1) \equiv 0$.

Rozwiązań drugie (zaawansowane). Na mocy ćwiczenia 4.6.2–6 mamy $x^{\bar{p}} \equiv x^p - x$ (modulo p); stąd $\begin{bmatrix} p \\ k \end{bmatrix} \equiv \delta_{kp} - \delta_{k1}$ na mocy ćwiczenia 1.2.6–32. Teraz korzystamy z ćwiczenia 6.

Składniad wiadomo, że licznik H_{p-1} jest wielokrotnością p^2 dla $p > 3$; zobacz Hardy, Wright, *An Introduction to the Theory of Numbers*, rozdział 7.8.

18. Dla $n = 2^k m$ przy m nieparzystym suma równa się $2^{2k} m_1/m_2$, gdzie oba m_1 i m_2 są nieparzyste. [AMM 67 (1960), 924–925].

19. Tylko $n = 0$ i $n = 1$. Rozważmy $n \geq 2$. Niech $k = \lfloor \lg n \rfloor$. Istnieje dokładnie jeden wyraz, którego mianownik jest równy 2^k , zatem $2^{k-1} H_n - \frac{1}{2}$ jest sumą wyrazów mających w mianowniku wyłącznie liczby pierwsze. Gdyby H_n było liczbą całkowitą, $2^{k-1} H_n - \frac{1}{2}$ miałyby mianownik równy 2.

20. Rozpisz wyrazy w funkcji podcałkowej. Zobacz też AMM 69 (1962), 239, a także artykuł H. W. Goulda, *Mathematics Magazine* 34 (1961), 317–321.

21. $H_{n+1}^2 - H_{n+1}^{(2)}$.

22. $(n+1)(H_n^2 - H_n^{(2)}) - 2n(H_n - 1)$.

23. $\Gamma'(n+1)/\Gamma(n+1) = 1/n + \Gamma'(n)/\Gamma(n)$, gdyż $\Gamma(x+1) = x\Gamma(x)$. Stąd $H_n = \gamma + \Gamma'(n+1)/\Gamma(n+1)$. Funkcję $\psi(x) = \Gamma'(x)/\Gamma(x) = H_{x-1} - \gamma$ nazywamy *funkcją psi* lub *funkcją digamma*. Niektóre jej wartości dla wymiernych r są zamieszczone w dodatku A.

24. Mamy

$$x \lim_{n \rightarrow \infty} e^{(H_n - \ln n)x} \prod_{k=1}^n \left(\left(1 + \frac{x}{k}\right) e^{-x/k} \right) = \lim_{n \rightarrow \infty} \frac{x(x+1)\dots(x+n)}{n^x n!}.$$

Uwaga: Wnioskujemy stąd, że uogólnienie H_n rozważane w poprzednim ćwiczeniu jest równe $H_x^{(r)} = \sum_{k \geq 0} (1/(k+1)^r - 1/(k+1+x)^r)$, gdy $r = 1$. Ten sam pomysł można zastosować do większych wartości r . Nieskończony iloczyn jest zbieżny dla wszystkich x zespolonych.

1.2.8

1. Po k miesiącach jest F_{k+2} par. Odpowiedź: $F_{14} = 377$ par.

2. $\ln(\phi^{1000}/\sqrt{5}) = 1000 \ln \phi - \frac{1}{2} \ln 5 = 480.40711$; $\log_{10} F_{1000}$ to $1/(\ln 10)$ razy tyle, czyli 208.64. Stąd F_{1000} jest liczbą 209-cyfrową, której pierwszą cyfrą jest 4.

4. 0, 1, 5; później F_n rośnie zbyt szybko.

5. 0, 1, 12.

6. Indukcja. (Równanie zachodzi również dla n ujemnych; zobacz ćwiczenie 8).

7. Jeśli d jest właściwym dzielnikiem n , to F_d dzieli F_n . F_d jest większe od jednego i mniejsze od F_n przy założeniu, że d jest większe od 2. Jedyną liczbą złożoną, która nie ma właściwego czynnika większego od 2, jest $n = 4$, zatem $F_4 = 3$ jest jedynym wyjątkiem.

- 8.** $F_{-1} = 1; F_{-2} = -1; F_{-n} = (-1)^{n+1}F_n$ przez indukcję względem n .
- 9.** (15) Nie. Pozostałe tak, na mocy rozumowania indukcyjnego polegającego na dowodzie prawdziwości twierdzenia dla $n - 1$ przy założeniu, że jest ono prawdziwe dla liczb n i większych.
- 10.** Dla parzystych n jest większa, dla n nieparzystych mniejsza. (Zobacz (14)).
- 11.** Indukcja; zobacz ćwiczenie 9. To jest szczególny przypadek ćwiczenia 13(a).
- 12.** Skoro $\mathcal{G}(z) = \sum \mathcal{F}_n z^n$, to $(1 - z - z^2)\mathcal{G}(z) = z + F_0 z^2 + F_1 z^3 + \dots = z + z^2 G(z)$. Stąd $\mathcal{G}(z) = G(z) + zG(z)^2$; z (17) mamy $\mathcal{G}_n = ((3n+3)/5)F_n - (n/5)F_{n+1}$.
- 13.** (a) $a_n = rF_{n-1} + sF_n$. (b) Mamy $(b_{n+2} + c) = (b_{n+1} + c) + (b_n + c)$, możemy zatem zbudować nowy ciąg $b'_n = b_n + c$. Korzystamy z punktu (a), by policzyć b'_n i otrzymujemy $cF_{n-1} + (c+1)F_n - c$.
- 14.** $a_n = F_{m+1}F_{n-1} + (F_{m+2} + 1)F_n - \binom{n}{m} - \binom{n+1}{m-1} - \dots - \binom{n+m}{0}$.
- 15.** $c_n = xa_n + yb_n + (1-x-y)F_n$.
- 16.** F_{n+1} . Indukcja oraz $\binom{n+1-k}{k} = \binom{n-k}{k} + \binom{(n-1)-(k-1)}{k-1}$.
- 17.** W ogólności wyrażenie $(x^{n+k} - y^{n+k})(x^{m-k} - y^{m-k}) - (x^n - y^n)(x^m - y^m)$ równa się $(xy)^n(x^{m-n-k} - y^{m-n-k})(x^k - y^k)$. Podstaw $x = \phi$, $y = \bar{\phi}$ i podziel przez $(\sqrt{5})^2$.
- 18.** Tak, F_{2n+1} .
- 19.** Niech $u = \cos 72^\circ$, $v = \cos 36^\circ$. Mamy $u = 2v^2 - 1$; $v = 1 - 2\sin^2 18^\circ = 1 - 2u^2$. Stąd $u + v = 2(v^2 - u^2)$, tj. $1 = 2(v - u) = 2v - 4v^2 + 2$. Zatem $v = \frac{1}{2}\phi$. (A także $u = \frac{1}{2}\phi^{-1}$, $\sin 36^\circ = \frac{1}{2}5^{1/4}\phi^{-1/2}$, $\sin 72^\circ = \frac{1}{2}5^{1/4}\phi^{1/2}$).
- 20.** $F_{n+2} - 1$.
- 21.** Pomnóż przez $x^2 + x - 1$; rozwiązaniem jest $(x^{n+1}F_{n+1} + x^{n+2}F_n - x)/(x^2 + x - 1)$. Mianownik jest zerem dla x równego $1/\phi$ lub $1/\bar{\phi}$; w tych przypadkach rozwiązaniem jest
- $$((n+1)x^n F_{n+1} + (n+2)x^{n+1} F_n - 1)/(2x + 1).$$
- 22.** F_{m+2n} ; podstaw $t = 2$ w kolejnym ćwiczeniu.
- 23.**
$$\begin{aligned} \frac{1}{\sqrt{5}} \sum_k \binom{n}{k} (\phi^k F_t^k F_{t-1}^{n-k} \phi^m - \bar{\phi}^k F_t^k F_{t-1}^{n-k} \bar{\phi}^m) \\ = \frac{1}{\sqrt{5}} (\phi^m (\phi F_t + F_{t-1})^n - \bar{\phi}^m (\bar{\phi} F_t + F_{t-1})^n) = F_{m+tn}. \end{aligned}$$
- 24.** F_{n+1} (rozwiń dopełnienia algebraiczne w pierwszym wierszu).
- 25.** $2^n \sqrt{5} F_n = (1 + \sqrt{5})^n - (1 - \sqrt{5})^n$.
- 26.** Z twierdzenia Fermata $2^{p-1} \equiv 1$; teraz wystarczy zastosować poprzednie twierdzenie oraz ćwiczenie 1.2.6–10(b).
- 27.** Twierdzenie jest prawdziwe dla $p = 2$. W pozostałych przypadkach zachodzi $F_{p-1}F_{p+1} - F_p^2 = -1$, zatem z poprzedniego ćwiczenia oraz twierdzenia Fermata $F_{p-1}F_{p+1} \equiv 0$ (modulo p). Tylko jeden z tych czynników może być wielokrotnością p , bo $F_{p+1} = F_p + F_{p-1}$.
- 28.** $\bar{\phi}^n$. Uwaga: Rozwiązaniem równania rekurencyjnego $a_{n+1} = Aa_n + B^n$, $a_0 = 0$ jest
- $$a_n = (A^n - B^n)/(A - B) \text{ gdy } A \neq B, \quad a_n = nA^{n-1} \text{ gdy } A = B.$$

29. (a)	$\binom{n}{0}_{\mathcal{F}}$	$\binom{n}{1}_{\mathcal{F}}$	$\binom{n}{2}_{\mathcal{F}}$	$\binom{n}{3}_{\mathcal{F}}$	$\binom{n}{4}_{\mathcal{F}}$	$\binom{n}{5}_{\mathcal{F}}$	$\binom{n}{6}_{\mathcal{F}}$
	1	0	0	0	0	0	0
	1	1	0	0	0	0	0
	1	1	1	0	0	0	0
	1	2	2	1	0	0	0
	1	3	6	3	1	0	0
	1	5	15	15	5	1	0
	1	8	40	60	40	8	1

(b) Wynika z (6). [É. Lucas, *Amer. J. Math.* **1** (1878), 201–204].

30. Dowodzimy przez indukcję względem m . Twierdzenie jest oczywiste dla $m = 1$. Dalej mamy:

$$(a) \sum_k \binom{m}{k}_{\mathcal{F}} (-1)^{\lceil(m-k)/2\rceil} F_{n+k}^{m-2} F_k = F_m \sum_k \binom{m-1}{k-1}_{\mathcal{F}} (-1)^{\lceil(m-k)/2\rceil} F_{n+k}^{m-2} = 0.$$

$$(b) \sum_k \binom{m}{k}_{\mathcal{F}} (-1)^{\lceil(m-k)/2\rceil} F_{n+k}^{m-2} (-1)^k F_{m-k} \\ = (-1)^m F_m \sum_k \binom{m-1}{k}_{\mathcal{F}} (-1)^{\lceil(m-1-k)/2\rceil} F_{n+k}^{m-2} = 0.$$

(c) Ponieważ $(-1)^k F_{m-k} = F_{k-1} F_m - F_k F_{m-1}$ i $F_m \neq 0$, to z (a) i (b) wynika, że $\sum_k \binom{m}{k}_{\mathcal{F}} (-1)^{\lceil(m-k)/2\rceil} F_{n+k}^{m-2} F_{k-1} = 0$.

(d) Ponieważ $F_{n+k} = F_{k-1} F_n + F_k F_{n+1}$, wynik otrzymujemy z (a) i (c). Można go także udowodnić w nieco bardziej ogólnej postaci za pomocą twierdzenia q -mianowego z ćwiczenia 1.2.6–58. *Bibliografia:* Dov Jarden, *Recurring Sequences*, wyd. 2 (Jerusalem, 1966), 30–33; J. Riordan, *Duke Math. J.* **29** (1962), 5–12.

31. Skorzystaj z ćwiczeń 8 i 11.

32. Ciąg Fibonacciego modulo F_n wygląda następująco: $0, 1, \dots, F_{n-1}, 0, F_{n-1}, -F_{n-2}, F_{n-3}, \dots$

33. Zauważ, że $\cos z = \frac{1}{2}(e^{iz} + e^{-iz}) = -i/2$ dla tego konkretnego z . Następnie skorzystaj z faktu, że $\sin(n+1)z + \sin(n-1)z = 2 \sin nz \cos z$, dla dowolnego z .

34. Udowodnij, że jedyną możliwą wartością F_{k_1} jest największa liczba Fibonacciego mniejsza lub równa n . Stąd $n - F_{k_1}$ jest mniejsze niż F_{k_1-1} , zatem na mocy indukcji istnieje jednoznaczna reprezentacja $n - F_{k_1}$. Szkic tego dowodu jest podobny do rozumowania na temat jednoznaczności rozkładu na czynniki pierwsze. System liczbowy Fibonacciego pochodzi od E. Zeckendorfa [zobacz *Simon Stevin* **29** (1952), 190–195; *Bull. Soc. Royale des Sciences de Liège* **41** (1972), 179–182]; uogólnienia są omówione w ćwiczeniu 5.4.2–10.

35. Zobacz G. M. Bergman, *Mathematics Magazine* **31** (1957), 98–110. Reprezentację $x > 0$ znajdujemy, dobierając największe k , takie że $\phi^k \leq x$, i zapisując x jako ϕ^k plus reprezentacja $x - \phi^k$.

Reprezentacje nieujemnych liczb całkowitych można uzyskać z poniższych całkowitoliczbowych reguł rekurencyjnych, zaczynając od trywialnej reprezentacji 0 i 1. Niech $L_n = \phi^n + \bar{\phi}^n = F_{n+1} + F_{n-1}$. Reprezentacją $L_{2n} + m$ dla $0 \leq m \leq L_{2n-1}$ i $n \geq 1$ jest $\phi^{2n} + \phi^{-2n}$ plus reprezentacja m . Reprezentacją $L_{2n+1} + m$ dla $0 < m < L_{2n}$ i $n \geq 0$ jest $\phi^{2n+1} + \phi^{-2n-2}$ plus reprezentacja $m - \phi^{-2n}$, gdzie to ostatnie wyrażenie pochodzi z zastosowania reguły $\phi^k - \phi^{k-2j} = \phi^{k-1} + \phi^{k-3} + \dots + \phi^{k-2j+1}$. Okazuje się, że wszystkie słowa α złożone z zer i jedynek, takie że α rozpoczyna się jedynką i nie zawiera dwóch jedynek z rzędu, pojawiają się na lewo od przecinka w reprezentacji

dokładnie jednej liczby całkowitej, poza słowami zakończonymi na $10^{2k}1$ (takie napisy w ogóle nie występują w reprezentacjach).

36. Możemy rozważyć nieskończony ciąg S_∞ , gdyż S_n dla $n > 1$ składa się z pierwszych F_n liter S_∞ . Nie ma podwójnych a , ani potrójnych b . Słowo S_n zawiera F_{n-2} liter a i F_{n-1} liter b . Jeśli zapiszemy $m - 1$ w systemie pozycyjnym Fibonacciego, jak w ćwiczeniu 34, to m -tą literą S_∞ jest a wtedy i tylko wtedy, gdy $k_r = 2$. A k -tą literą S_∞ jest b wtedy i tylko wtedy, gdy $\lfloor (k+1)\phi^{-1} \rfloor - \lfloor k\phi^{-1} \rfloor = 1$; liczba liter b wśród pierwszych k liter wynosi zatem $\lfloor (k+1)\phi^{-1} \rfloor$. Ponadto b jest k -tą literą wtedy i tylko wtedy, gdy $k = \lfloor m\phi \rfloor$ dla pewnej dodatniej liczby całkowitej m . Ten ciąg był badany przez Jeana Bernoulliego III w XVIII wieku, przez A. A. Markowa w XIX wieku i przez wielu innych matematyków; zobacz K. B. Stolarsky, *Canadian Math. Bull.* **19** (1976), 473–482.

37. [Fibonacci Quart. **1** (December 1963), 9–12] Rozważmy system liczbowy Fibonacciego z ćwiczenia 34. Niech w tym systemie $n = F_{k_1} + \dots + F_{k_r} > 0$ oraz $\mu(n) = F_{k_r}$. Niech $\mu(0) = \infty$. Mamy: (A) Jeśli $n > 0$, to $\mu(n - \mu(n)) > 2\mu(n)$. Dowód: $\mu(n - \mu(n)) = F_{k_{r-1}} \geq F_{k_r+2} > 2F_{k_r}$, bo $k_r \geq 2$. (B) Jeśli $0 < m < F_k$, to $\mu(m) \leq 2(F_k - m)$. Dowód: Niech $\mu(m) = F_j$; $m \leq F_{k-1} + F_{k-3} + \dots + F_{j+(k-1-j) \bmod 2} = -F_{j-1+(k-1-j) \bmod 2} + F_k \leq -\frac{1}{2}F_j + F_k$. (C) Jeśli $0 < m < \mu(n)$, to $\mu(n - \mu(n) + m) \leq 2(\mu(n) - m)$. Dowód: Wynika z (B). (D) Gdy $0 < m < \mu(n)$, to $\mu(n - m) \leq 2m$. Dowód: Podstawiamy $m = \mu(n) - m$ w (C).

Teraz udowodnimy, że jeżeli jest n bierek i jeżeli można wziąć co najwyżej q w następnym ruchu, to istnieje ruch wygrywający wtedy i tylko wtedy, gdy $\mu(n) \leq q$. Dowód: (a) Jeśli $\mu(n) > q$, to wszystkie ruchy pozostawiają konfigurację n' , q' , gdzie $\mu(n') \leq q'$. [To wynika z (D)]. (b) Jeśli $\mu(n) \leq q$, to możemy albo wygrać w tym ruchu (jeśli $q \geq n$), albo wykonać ruch, który pozostawi konfigurację n' , q' , $\mu(n') > q'$. [To wynika z (A): ruch polega na usunięciu $\mu(n)$ bierek]. Można zauważyć, że wszystkie ruchy wygrywające (dla $n = F_{k_1} + \dots + F_{k_r}$) polegają na usunięciu $F_{k_j} + \dots + F_{k_r}$ bierek dla pewnego j , takiego że $1 \leq j \leq r$ oraz zachodzi $j = 1$ lub $F_{k_{j-1}} > 2(F_{k_j} + \dots + F_{k_r})$.

Reprezentacja Fibonacciego liczby 1000 to $987 + 13$; jedynym ruchem zapewniającym zwycięstwo jest usunięcie 13 bierek. Pierwszy gracz może zawsze wygrać, jeżeli n nie jest liczbą Fibonacciego.

Rozwiązań dotyczących dużo bardziej ogólnych gier tego typu uzyskał A. Schwenk [Fibonacci Quarterly **8** (1970), 225–234].

39. $(3^n - (-2)^n)/5$.

40. Dowodzimy przez indukcję względem m , że $f(n) = m$ dla $F_m < n \leq F_{m+1}$: po pierwsze, $f(n) \leq \max(1 + f(F_m), 2 + f(n - F_m)) = m$. Po drugie, jeżeli $f(n) < m$, to istnieje takie $k < n$, że $1 + f(k) < m$ (zatem $k \leq F_{m-1}$) i $2 + f(n - k) < m$ (stąd $n - k \leq F_{m-2}$), ale wtedy $n \leq F_{m-1} + F_{m-2}$. [Zatem drzewa Fibonacciego zdefiniowane w rozdziale 6.2.1 minimalizują maksymalny koszt ścieżki od korzenia do liścia, gdy prawa gałąź kosztuje dwa razy tyle co lewa gałąź].

41. $F_{k_1+1} + \dots + F_{k_r+1} = \phi n + (\hat{\phi}^{k_1} + \dots + \hat{\phi}^{k_r})$ jest liczbą całkowitą, a wartość w nawiasach leży między $\hat{\phi}^3 + \hat{\phi}^5 + \dots = \phi^{-1} - 1$ a $\hat{\phi}^2 + \hat{\phi}^4 + \dots = \phi^{-1}$. Podobnie $F_{k_1-1} + \dots + F_{k_r-1} = \phi^{-1}n + (\hat{\phi}^{k_1} + \dots + \hat{\phi}^{k_r}) = f(\phi^{-1}n)$. [Takie przesuwanie reprezentacji w systemie liczbowym Fibonacciego jest wygodnym sposobem konwersji między odległościami w milach i kilometrach, zobacz CMath, §6.6].

42. [Fibonacci Quarterly **6** (1968), 235–244] Jeśli taka reprezentacja istnieje, to

$$mF_{N-1} + nF_N = F_{k_1+N} + F_{k_2+N} + \dots + F_{k_r+N} \quad (*)$$

dla wszystkich liczb całkowitych N . Stąd istnienie dwóch różnych reprezentacji byłoby w sprzeczności z ćwiczeniem 34.

Odwrotnie, możemy przez indukcję udowodnić istnienie takich wspólnych reprezentacji dla wszystkich nieujemnych m i n . Ale ciekawiej jest skorzystać z poprzedniego ćwiczenia i udowodnić, że takie wspólne reprezentacje istnieją dla (być może ujemnych) liczb całkowitych m i n wtedy i tylko wtedy, gdy $m + \phi n \geq 0$. Niech N będzie odpowiednio duże, tak że $|m\hat{\phi}^{N-1} + n\hat{\phi}^N| < \phi^{-2}$ i niech reprezentuje $mF_{N-1} + nF_N$ jak w (*). Wówczas $mF_N + nF_{N+1} = \phi(mF_{N-1} + nF_N) + (m\hat{\phi}^{N-1} + n\hat{\phi}^N) = f(\phi(mF_{N-1} + nF_N)) = F_{k_1+N+1} + \dots + F_{k_r+N+1}$, zatem (*) jest prawdziwe dla wszystkich N . Teraz podstawiamy $N = 0$ i $N = 1$.

1.2.9

1. $1/(1 - 2z) + 1/(1 - 3z)$.
2. Wynika ze wzoru (6), gdyż $\binom{n}{k} = n!/k!(n-k)!$.
3. $G'(z) = \ln(1/(1-z))/(1-z)^2 + 1/(1-z)^2$. Z tego oraz wiedząc, że $G(z)/(1-z)$, otrzymujemy $\sum_{k=1}^{n-1} H_k = nH_n - n$; to się zgadza z 1.2.7-(8).
4. Podstaw $t = 0$.
5. Na mocy (11) i (22) współczynnik przy z^k jest równy

$$(n-1)! \sum_{0 \leq j < k} \left\{ \begin{matrix} j \\ n-1 \end{matrix} \right\} \binom{k}{j}.$$

Korzystamy z 1.2.6-(46) oraz 1.2.6-(52). (Można także zróżniczkować i skorzystać z 1.2.6-(46)).

6. $(\ln(1/(1-z)))^2$; pochodna to podwojona funkcja tworząca ciągu liczb harmonicznych. Suma wynosi zatem $2H_{n-1}/n$.

8. $1/((1-z)(1-z^2)(1-z^3)\dots)$. [Jest to jedno z pierwszych w historii zastosowań funkcji tworzących. Omówienie osiemnastowiecznych badań L. Eulera nad funkcjami tworzącymi można znaleźć w: G. Pólya, *Induction and Analogy in Mathematics* (Princeton: Princeton University Press, 1954), rozdział 6].

9. $\frac{1}{24}S_1^4 + \frac{1}{4}S_1^2S_2 + \frac{1}{8}S_2^2 + \frac{1}{3}S_1S_3 + \frac{1}{4}S_4$.

10. $G(z) = (1+x_1z)\dots(1+x_nz)$. Logarytmując, jak w wyprowadzeniu (38), otrzymujemy takie same wzory, tyle że zamiast (17) jest (24), a odpowiedź jest prawie identyczna, bo zamiast $-S_2, -S_4, -S_6, \dots$ stoją S_2, S_4, S_6, \dots . Mamy $a_1 = S_1, a_2 = \frac{1}{2}S_1^2 - \frac{1}{2}S_2, a_3 = \frac{1}{6}S_1^3 - \frac{1}{2}S_1S_2 + \frac{1}{3}S_3, a_4 = \frac{1}{24}S_1^4 - \frac{1}{4}S_1^2S_2 + \frac{1}{8}S_2^2 + \frac{1}{3}S_1S_3 - \frac{1}{4}S_4$ (zobacz ćwiczenie 9). Zachodzi związek rekurencyjny analogiczny do (39): $na_n = S_1a_{n-1} - S_2a_{n-2} + \dots$. Uwaga: Te równania rekurencyjne nazywane są *tożsamościami Newtona*, ponieważ Izaak Newton opublikował je w *Arithmetica Universalis* (1707); zobacz D. J. Struik *Source Book in Mathematics* (Harvard University Press, 1969), 94–95.

11. Ponieważ $\sum_{m \geq 1} S_m z^m/m = \ln G(z) = \sum_{k \geq 1} (-1)^{k-1} (h_1 z + h_2 z^2 + \dots)^k/k$, poszukiwany współczynnik to $(-1)^{k_1+k_2+\dots+k_m-1} m(k_1 + k_2 + \dots + k_m - 1)!/k_1!k_2!\dots k_m!$. [Wystarczy pomnożyć przez $(-1)^{m-1}$, by otrzymać współczynnik przy $a_1^{k_1}a_2^{k_2}\dots a_m^{k_m}$ w przedstawieniu S_m za pomocą a z ćwiczenia 10. Albert Girard przedstawił wzory na S_1, S_2, S_3 i S_4 wyrażone za pomocą a_1, a_2, a_3 i a_4 pod koniec pracy *Invention Nouvelle en Algébre* (Amsterdam: 1629); to były narodziny teorii funkcji symetrycznych].

12. $\sum_{m,n \geq 0} a_{mn} w^m z^n = \sum_{m,n \geq 0} \binom{n}{m} w^m z^n = \sum_{n \geq 0} (1+w)^n z^n = 1/(1-z-wz).$

13. $\int_n^{n+1} e^{-st} f(t) dt = (a_0 + \dots + a_n)(e^{-sn} - e^{-s(n+1)})/s.$ Sumując te wyrażenia dla wszystkich $n,$ otrzymujemy $\mathbf{L}f(s) = G(e^{-s})/s.$

14. Zobacz ćwiczenie 1.2.6–38.

15. $G_n(z) = G_{n-1}(z) + zG_{n-2}(z) + \delta_{n0},$ zatem $H(w) = 1/(1-w-zw^2).$ Stąd mamy

$$G_n(z) = \left(\left(\frac{1+\sqrt{1+4z}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{1+4z}}{2} \right)^{n+1} \right) / \sqrt{1+4z} \quad \text{gdy } z \neq -\frac{1}{4};$$

$$G_n(-\frac{1}{4}) = (n+1)/2^n \text{ dla } n \geq 0.$$

16. $G_{nr}(z) = (1+z+\dots+z^r)^n = \left(\frac{1-z^{r+1}}{1-z} \right)^n.$ [Przyjrzyj się przypadkowi $r = \infty$].

17. $\sum_k \binom{-w}{k} (-z)^k = \sum_k \frac{w(w+1)\dots(w+k-1)}{k(k-1)\dots 1} z^k = \sum_{n,k} \binom{k}{n} z^k w^n / k!.$

(Można inaczej: zapisz jako $e^{w \ln(1/(1-z))}$ i w pierwszej kolejności rozwiń w potęgi w).

18. (a) Dla ustalonego n i indeksu r funkcja tworząca ma postać:

$$\begin{aligned} G_n(z) &= (1+z)(1+2z)\dots(1+nz) = z^{n+1} \left(\frac{1}{z} \right) \left(\frac{1}{z} + 1 \right) \left(\frac{1}{z} + 2 \right) \dots \left(\frac{1}{z} + n \right) \\ &= \sum_k \binom{n+1}{k} z^{n+1-k} \end{aligned}$$

na mocy (27). Stąd odpowiedź $\binom{n+1}{n+1-r}.$ (b) Podobnie funkcja tworząca to

$$\frac{1}{1-z} \cdot \frac{1}{1-2z} \cdot \dots \cdot \frac{1}{1-nz} = \sum_k \binom{k}{n} z^{k-n}$$

na mocy (28), zatem odpowiedzią jest $\binom{n+r}{n}.$

19. $\sum_{n \geq 1} (1/n - 1/(n+p/q)) x^{p+nq} = \sum_{k=0}^{q-1} \omega^{-kp} \ln(1 - \omega^k x) - x^p \ln(1 - x^q) + \frac{q}{p} x^p = f(x) + g(x),$ gdzie $\omega = e^{2\pi i/q}$ i

$$f(x) = \sum_{k=1}^{q-1} \omega^{-kp} \ln(1 - \omega^k x), \quad g(x) = (1 - x^p) \ln(1 - x) + \frac{q}{p} x^p - x^p \ln \frac{1 - x^q}{1 - x}.$$

Teraz mamy $\lim_{x \rightarrow 1^-} g(x) = q/p - \ln q.$ Korzystając z tożsamości

$$\ln(1 - e^{i\theta}) = \ln \left(2e^{i(\theta-\pi)/2} \frac{e^{i\theta/2} - e^{-i\theta/2}}{2i} \right) = \ln 2 + \frac{1}{2}i(\theta - \pi) + \ln \sin \frac{\theta}{2},$$

możemy zapisać $f(x) = A + B,$ gdzie

$$A = \sum_{k=1}^{q-1} \omega^{-kp} \left(\ln 2 - \frac{i\pi}{2} + \frac{ik\pi}{q} \right) = -\ln 2 + \frac{i\pi}{2} + \frac{i\pi}{(\omega^{-p} - 1)};$$

$$\begin{aligned} B &= \sum_{k=1}^{q-1} \omega^{-kp} \ln \sin \frac{k}{q} \pi = \sum_{0 < k < q/2} (\omega^{-kp} + \omega^{-(q-k)p}) \ln \sin \frac{k}{q} \pi \\ &= 2 \sum_{0 < k < q/2} \cos \frac{2pk}{q} \pi \cdot \ln \sin \frac{k}{q} \pi. \end{aligned}$$

Ostatecznie

$$\frac{i}{2} + \frac{i}{(\omega^{-p} - 1)} = \frac{i}{2} \left(\frac{1 + \omega^p}{1 - \omega^p} \right) = \frac{i}{2} \left(\frac{\omega^{p/2} + \omega^{-p/2}}{\omega^{p/2} - \omega^{-p/2}} \right) = \frac{1}{2} \cot \frac{p}{q} \pi.$$

[Gauss wyprowadził powyższe wyniki w §33 swojej monografii na temat szeregów hipergeometrycznych, równanie [75], ale nie zamieścił pełnego dowodu; pełne uzasadnienie zostało podane przez Abela w *Crelle 1* (1826), 314–315].

20. $c_{mk} = k! \{ \begin{matrix} m \\ k \end{matrix} \}$, na mocy 1.2.6–(45).

21. Mamy $z^2 G'(z) + zG(z) = G(z) - 1$. Rozwiązańem tego równania różniczkowego jest $G(z) = (-1/z) e^{-1/z} (E_1(-1/z) + C)$, gdzie $E_1(x) = \int_z^\infty e^{-t} dt/t$, a C jest stałą. Ta funkcja bardzo źle zachowuje się w okolicach $z = 0$, a $G(z)$ nie ma rozwinięcia w szereg potęgowy. Ponieważ $\sqrt[n]{n!} \approx n/e$ jest nieograniczona, funkcja tworząca nie jest w tym przypadku zbieżna, jednak dla $z < 0$ jest asymptotycznym rozwinięciem powyższej funkcji. [Zobacz K. Knopp, *Infinite Sequences and Series* (Dover, 1956), rozdział 66].

22. $G(z) = (1+z)^r (1+z^2)^r (1+z^4)^r (1+z^8)^r \dots = (1-z)^{-r}$. Wynika stąd, że podana suma to $\binom{r+n-1}{n}$.

23. Dla $m = 1$ jest to twierdzenie dwumianowe, gdzie $f_1(z) = z$ i $g_1(z) = 1 + z$. Dla $m \geq 1$ możemy zwiększyć m o 1, jeżeli zamienimy z_m na $z_m(1 + z_{m+1}^{-1})$ i przyjmiemy $f_{m+1}(z_1, \dots, z_{m+1}) = z_{m+1} f_m(z_1, \dots, z_{m-1}, z_m(1 + z_{m+1}^{-1}))$, $g_{m+1}(z_1, \dots, z_{m+1}) = z_{m+1} g_m(z_1, \dots, z_{m-1}, z_m(1 + z_{m+1}^{-1}))$. Stąd $g_2(z_1, z_2) = z_1 + z_2 + z_1 z_2$ i

$$\frac{g_m(z_1, \dots, z_m)}{f_m(z_1, \dots, z_m)} = 1 + \cfrac{z_1^{-1}}{1 + \cfrac{z_2^{-1}}{1 + \cfrac{\ddots}{1 + z_m^{-1}}}}.$$

Dla obu wielomianów f_m i g_m zachodzi ta sama zależność rekurencyjna $f_m = z_m f_{m-1} + z_{m-1} f_{m-2}$, $g_m = z_m g_{m-1} + z_{m-1} g_{m-2}$, z warunkami początkowymi $f_{-1} = 0$, $f_0 = g_{-1} = g_0 = z_0 = 1$. Wynika stąd, że g_m jest sumą wszystkich wyrazów, które można otrzymać, zaczynając od $z_1 \dots z_m$ i wykreślając zero lub więcej niesąsiadujących ze sobą czynników. Można to zrobić na F_{m+2} sposobów. Podobnie interpretować można f_m , tyle że z_1 musi pozostać. W punkcie (b) odkrywamy, że wielomian $h_m = z_m g_{m-1} + z_{m-1} f_{m-2}$; to jest suma wszystkich wyrazów otrzymanych z $z_1 \dots z_m$ przez wykreślenie czynników, które nie sąsiadują cyklicznie. Na przykład $h_3 = z_1 z_2 z_3 + z_1 z_2 + z_1 z_3 + z_2 z_3$.

(b) Na mocy (a), $S_n(z_1, \dots, z_{m-1}, z) = [z_m^n] \sum_{r=0}^n z^r z_m^{n-r} f_m^{n-r} g_m^r$; stąd

$$S_n(z_1, \dots, z_m) = \sum_{0 \leq s \leq r \leq n} \binom{r}{s} \binom{n-r}{s} a^{r-s} b^s c^s d^{n-r-s},$$

gdzie $a = z_m g_{m-1}$, $b = z_{m-1} g_{m-2}$, $c = z_m f_{m-1}$, $d = z_{m-1} f_{m-2}$. Pomnożenie tego równania przez z^n i sumowanie najpierw względem n , potem względem r , a następnie względem s daje postać zamkniętą

$$S_n(z_1, \dots, z_m) = [z^n] \frac{1}{(1-az)(1-dz)-bcz^2} = \frac{\rho^{n+1} - \sigma^{n+1}}{\rho - \sigma},$$

gdzie $1 - (a+d)z + (ad - bc)z^2 = (1 - \rho z)(1 - \sigma z)$. Zatem $a + d = h_m$, a $ad - bc$ upraszcza się do $(-1)^m z_1 \dots z_m$. [Przy okazji odkryliśmy związek rekurencyjny $S_n =$

$h_m S_{n-1} - (-1)^m z_1 \dots z_m S_{n-2}$, który niełatwo wyprowadzić bez korzystania z funkcji tworzących].

(c) Niech $\rho_1 = (z + \sqrt{z^2 + 4z})/2$ i $\sigma_1 = (z - \sqrt{z^2 + 4z})/2$ będą pierwiastkami dla $m = 1$; wówczas $\rho_m = \rho_1^m$ i $\sigma_m = \sigma_1^m$.

Carlitz posłużył się tym wynikiem w dowodzie zaskakującego faktu: wielomianem charakterystycznym $\det(xI - A)$ macierzy $n \times n$

$$A = \begin{pmatrix} 0 & 0 & \dots & 0 & \binom{0}{0} \\ 0 & 0 & \dots & \binom{1}{0} & \binom{1}{1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \binom{n-1}{0} & \binom{n-1}{1} & \dots & \binom{n-1}{n-2} & \binom{n-1}{n-1} \end{pmatrix}$$

„dosuniętych do prawej współczynników dwumianowych” jest $\sum_k \binom{n}{k}_{\mathcal{F}} (-1)^{\lceil (n-k)/2 \rceil} x^k$, z współczynnikami fibomianowymi (zobacz ćwiczenie 1.2.8–30). Za pomocą podobnych metod pokazał także, że

$$\begin{aligned} \sum_{k_1, \dots, k_m \geq 0} \binom{k_1 + k_2}{k_1} \binom{k_2 + k_3}{k_2} \dots \binom{k_m + k_1}{k_m} z_1^{k_1} \dots z_m^{k_m} \\ = \frac{1}{\sqrt{z_1^2 \dots z_m^2 h_m(-z_1^{-1}, \dots, -z_m^{-1})^2 - 4z_1 \dots z_m}}. \end{aligned}$$

[Collectanea Math. 27 (1965), 281–296].

24. Obie strony są równe $\sum_k \binom{m}{k} [z^n] (zG(z))^k$. Gdy $G(z) = 1/(1-z)$, tożsamość przyjmuje postać $\sum_k \binom{m}{k} \binom{n-1}{n-k} = \binom{m+n-1}{n}$, co jest przypadkiem 1.2.6–(21). Gdy $G(z) = (e^z - 1)/z$, staje się $\sum_k m^k \binom{n}{k} = m^n$, czyli 1.2.6–(45).

25. $\sum_k [w^k] (1-2w)^n [z^n] z^k (1+z)^{2n-2k} = [z^n] (1+z)^{2n} \sum_k [w^k] (1-2w)^n (z/(1+z)^2)^k$, co jest równe $[z^n] (1+z)^{2n} (1-2z/(1+z)^2)^n = [z^n] (1+z^2)^n = \binom{n}{n/2}$ [n parzyste].

Podobnie $\sum_k \binom{n}{k} \binom{2n-2k}{n-k} (-4)^k = (-1)^n \binom{2n}{n}$. Wiele przykładów tej metody sumowania można znaleźć w książce G. P. Jegoryczewa *Integral Representation and the Computation of Combinatorial Sums* (Amer. Math. Soc., 1984), będącej tłumaczeniem z wydania rosyjskiego z 1977 roku.

26. $[F(z)] G(z)$ oznacza wolny wyraz $F(z^{-1})G(z)$. Porównaj omówienie autorstwa D. E. Knutha w *A Classical Mind* (Prentice-Hall, 1994), 247–258.

1.2.10

1. $G_n(0) = 1/n$; jest to prawdopodobieństwo zdarzenia, że element $X[n]$ jest największy.

2. $G''(1) = \sum_k k(k-1)p_k$, $G'(1) = \sum_k kp_k$.

3. (min 0, ave 6.49, max 999, dev 2.42). Zauważ, że $H_n^{(2)}$ to w przybliżeniu $\pi^2/6$; zobacz 1.2.7–(7).

4. $\binom{n}{k} p^k q^{n-k}$.

5. Średnia: $36/5 = 7.2$; odchylenie standardowe: $6\sqrt{2}/5 \approx 1.697$.

6. Dla (18) ze wzoru

$$\ln(q + pe^t) = \ln\left(1 + pt + \frac{pt^2}{2} + \frac{pt^3}{6} + \dots\right) = pt + p(1-p)\frac{t^2}{2} + p(1-p)(1-2p)\frac{t^3}{6} + \dots$$

wnioskujemy, że $\kappa_3/n = p(1-p)(1-2p) = pq(q-p)$. (Ten elegancki schemat nie sprawdza się niestety już dla współczynnika przy t^4). Dla rozkładu (8) podstawiając $p = k^{-1}$, otrzymamy $\kappa_3 = \sum_{k=2}^n k^{-1}(1-k^{-1})(1-2k^{-1}) = H_n - 3H_n^{(2)} + 2H_n^{(3)}$. Dla (20) mamy $\ln G(e^t) = t + H(nt) - H(t)$, gdzie $H(t) = \ln((e^t - 1)/t)$. Ponieważ $H'(z) = e^t/(e^t - 1) - 1/t$, mamy $\kappa_r = (n^r - 1)B_r/r$ dla wszystkich $r \geq 2$, w szczególności $\kappa_3 = 0$.

7. Prawdopodobieństwo, że $A = k$ wynosi p_{mk} . Możemy założyć, że algorytm operuje na wartościach $1, 2, \dots, m$. Dla dowolnego podziału n pozycji na m rozłącznych podzbiorów istnieje $m!$ sposobów przyporządkowania liczb $1, \dots, m$ tym podzbiorom. Algorytm M zachowuje się tak, jakby dla każdego z tych podzbiorów w tablicy znajdował tylko jeden jego element, położony najbardziej na prawo. Zatem p_{mk} jest średnią dla każdego ustalonego podziału. Na przykład, gdy $n = 5, m = 3$ jednym z podziałów jest

$$\{X[1], X[4]\} \quad \{X[2], X[5]\} \quad \{X[3]\};$$

możliwe układy to 12312, 13213, 21321, 23123, 31231, 32132. Dla każdego podziału procent układów dla których $A = k$ jest taki sam.

Jednak rozkład prawdopodobieństwa zmieniłby się, gdybyśmy uwzględniali więcej informacji. W poprzednim rozdziale wyznaczyliśmy na przykład sześć możliwości dla $n = 3$ i $m = 2$: 122, 212, 221, 211, 121, 112. Gdybyśmy wiedzieli, że „2” występuje dwa razy, a „1” raz, to ta informacja kazałaby rozważać dalej jedynie trzy pierwsze układy. Ale tego nie przewiduje treść zadania.

8. M^n/M^n . Im większe M , tym prawdopodobieństwo bliższe jedynki.

9. Niech q_{nm} będzie prawdopodobieństwem zdarzenia, że wystąpiło dokładnie m różnych wartości. Wówczas z zależności

$$q_{nm} = \frac{M-m+1}{M} q_{(n-1)(m-1)} + \frac{m}{M} q_{(n-1)m}$$

wyprowadzamy

$$q_{nm} = M! \left\{ \begin{matrix} n \\ m \end{matrix} \right\} / (M-m)! M^n.$$

Zobacz też ćwiczenie 1.2.6–64.

10. Szukane prawdopodobieństwo to $q_{nm}p_{mk}$ zsumowane po wszystkich m , to jest $M^{-n} \sum_m \binom{M}{m} \left\{ \begin{matrix} n \\ m \end{matrix} \right\} \left[\begin{matrix} m \\ k+1 \end{matrix} \right]$. Wydaje się, że nie ma prostego wzoru na średnią, która jest o jeden mniejsza od

$$H_M - \sum_{m=1}^M \left(1 - \frac{m}{M}\right)^n m^{-1} = H_n + \sum_{k=1}^n \left(\binom{n}{k} - 1 \right) B_k M^{-k} k^{-1}.$$

11. Ponieważ jest to iloczyn, więc sumujemy półniezmienniki czynników. Gdy $H(z) = z^n$, $H(e^t) = e^{nt}$, zatem $\kappa_1 = n$, a dalej zera. Mamy $\text{mean}(F) = n + \text{mean}(G)$, a inne półniezmienniki się nie zmieniają. (Stąd nazwa „półniezmienniki”).

12. Pierwsza tożsamość jest oczywista, rozwijamy szereg potęgowy e^{kt} . Dla drugiej podstawiamy $u = 1 + M_1 t + M_2 t^2/2! + \dots$. Gdy $t = 0$, mamy $u = 1$ i $D_t^k u = M_k$. Również $D_u^j(\ln u) = (-1)^{j-1}(j-1)!/u^j$. Z ćwiczenia 11 ten sam wzór jest prawdziwy dla momentów centralnych, z wyjątkiem polegającym na opuszczeniu wszystkich wyrazów, dla których $k_1 > 0$. Stąd $\kappa_2 = m_2$, $\kappa_3 = m_3$, $\kappa_4 = m_4 - 3m_2^2$.

13. $G_n(z) = \frac{\Gamma(n+z)}{\Gamma(z+1)n!} = \frac{e^{-z}(n+z)^{z-1}}{\Gamma(z+1)} 8 \left(1 + \frac{z}{n}\right)^n (1 + O(n^{-1})) = \frac{n^{z-1}}{\Gamma(z+1)} (1 + O(n^{-1})).$

Niech $z_n = e^{it/\sigma_n}$. Gdy $n \rightarrow \infty$, a t jest ustalone, mamy $z_n \rightarrow 1$; stąd $\Gamma(z_n + 1) \rightarrow 1$ oraz

$$\begin{aligned} \lim_{n \rightarrow \infty} z_n^{-\mu_n} G_n(z_n) &= \lim_{n \rightarrow \infty} \exp \left(\frac{-it\mu_n}{\sigma_n} + (e^{it/\sigma_n} - 1) \ln n \right) \\ &= \lim_{n \rightarrow \infty} \exp \left(\frac{-t^2 \ln n}{2\sigma_n^2} + O\left(\frac{1}{\sqrt{\log n}}\right) \right) = e^{-t^2/2}. \end{aligned}$$

Uwagi: To twierdzenie pochodzi od Gonczarowa [Izw. Akad. Nauk SSSR Ser. Math. 8 (1944), 3–48]. P. Flajolet i M. Soria [Disc. Math. 114 (1993), 159–180] rozszerzyli analizę, by pokazać, że $G_n(z)$ i duża rodzina podobnych rozkładów nie tylko są w przybliżeniu normalne w okolicy swoich średnich, ale mają jednorodnie wykładnicze reszty w takim sensie, że

$$\text{prawdopodobieństwo} \left(\left| \frac{X_n - \mu_n}{\sigma_n} \right| > x \right) < e^{-ax}$$

dla pewnej dodatniej stałej a oraz dla wszystkich n i x .

14. $e^{-itpn/\sqrt{pqn}} (q + pe^{it/\sqrt{pqn}})^n = (qe^{-itp/\sqrt{pqn}} + pe^{itq/\sqrt{pqn}})^n$. Rozwiń wykładniki w szeregi wykładnicze, by otrzymać $(1 - t^2/2n + O(n^{-3/2}))^n = \exp(n \ln(1 - t^2/2n + O(n^{-3/2}))) = \exp(-t^2/2 + O(n^{-1/2})) \rightarrow \exp(-t^2/2)$.

15. (a) $\sum_{k \geq 0} e^{-\mu} (\mu z)^k / k! = e^{\mu(z-1)}$. (b) $\ln e^{\mu(e^t-1)} = \mu(e^t - 1)$, zatem wszystkie półnieziemianki są równe μ . (c) $\exp(-itnp/\sqrt{np}) \exp(np(it/\sqrt{np} - t^2/2np + O(n^{-3/2}))) = \exp(-t^2/2 + O(n^{-1/2}))$.

16. Funkcja tworząca: $g(z) = \sum_k p_k g_k(z)$; średnia: $\text{mean}(g) = \sum_k p_k \text{mean}(g_k)$; wariancja: $\text{var}(g) = \sum_k p_k \text{var}(g_k) + \sum_{j < k} p_j p_k (\text{mean}(g_j) - \text{mean}(g_k))^2$.

17. (a) Współczynniki $f(z)$ i $g(z)$ są nieujemne, zatem $f(1) = g(1) = 1$. Oczywiście tę cechę ma również $h(z)$, bo $h(1) = g(f(1))$, a współczynniki h są wielomianami względem współczynników wielomianów f i g o nieujemnych współczynnikach. (b) Niech $f(z) = \sum p_k z^k$, gdzie p_k jest prawdopodobieństwem zdarzenia, że w pewnym doświadczeniu otrzymujemy „wynik” k . Niech $g(z) = \sum q_k z^k$, gdzie q_k jest prawdopodobieństwem k -krotnego zajścia zdarzenia opisywanego przez f (zajście każdego zdarzenia jest niezależne od pozostałych). Wówczas $h(z) = \sum r_k z^k$, gdzie r_k jest prawdopodobieństwem zdarzenia, że suma wyników jest równa k . (Łatwo to dostrzec, gdy zauważymy, że $f(z)^k = \sum s_t z^t$, gdzie s_t jest prawdopodobieństwem zdarzenia, że ogólna punktacja t jest uzyskana w k niezależnych zdarzeniach). Przykład: jeśli f określa prawdopodobieństwo zdarzenia, że osoba ma k potomków płci męskiej, a g – prawdopodobieństwo zdarzenia, że w n -tym pokoleniu jest k mężczyzn, to h określa prawdopodobieństwo zdarzenia, że w $(n+1)$ -szym pokoleniu jest k mężczyzn, przy założeniu niezależności. (c) $\text{mean}(h) = \text{mean}(g) \text{mean}(f)$; $\text{var}(h) = \text{var}(g) \text{mean}^2(f) + \text{mean}(g) \text{var}(f)$.

18. Spójrzmy na wybór $X[1], \dots, X[n]$ jak na proces, w którym najpierw rozmieszcza my n -ki, potem $(n-1)$ -ki między tymi n -kami, ..., na końcu wstawiając jedynki. Gdy rozmieszcza my r -ki między ustawnionymi $r+1, \dots, n$, liczba lokalnych maksimów przy przesuwaniu się z prawej do lewej zwiększa się o jeden wtedy i tylko wtedy, gdy postawimy r na skrajnie prawej pozycji. To zdarza się z prawdopodobieństwem $k_r / (k_r + k_{r+1} + \dots + k_n)$.

19. Niech $a_k = l$. Wówczas a_k jest maksimum lewostronnym ciągu $a_1 \dots a_n \iff j < k$ pociąga $a_j < l \iff a_j > l$ pociąga $j > k \iff j > l$ pociąga $b_j > k \iff k$ jest minimum prawostronnym ciągu $b_1 \dots b_n$.

20. Mamy $m_L = \max\{a_1 - b_1, \dots, a_n - b_n\}$. Dowód: przypuśćmy, że to nieprawda. Niech k będzie najmniejszym indeksem, takim że $a_k - b_k > m_L$. Wówczas a_k nie jest maksimum lewostronnym, zatem istnieje $j < k$, takie że $a_j \geq a_k$. Ale wtedy $a_j - b_j \geq a_k - b_k > m_L$, zatem k nie mogło być minimalne. Podobnie $m_R = \max\{b_1 - a_1, \dots, b_n - a_n\}$.

21. Twierdzenie jest oczywiste, gdy $\epsilon \geq q$, zatem możemy założyć $\epsilon < q$. Przyjmujemy $x = \frac{p+\epsilon}{p} \frac{q}{q-\epsilon}$ we wzorze (25), skąd otrzymujemy $\Pr(X \geq n(p+\epsilon)) \leq ((\frac{p}{p+\epsilon})^{p+\epsilon} (\frac{q}{q-\epsilon})^{q-\epsilon})^n$. Teraz $(\frac{p}{p+\epsilon})^{p+\epsilon} \leq e^{-\epsilon}$, bo dla wszystkich t rzeczywistych mamy $t \leq e^{t-1}$. Tak więc $(q-\epsilon) \ln \frac{q}{q-\epsilon} = \epsilon - \frac{1}{2 \cdot 1} \epsilon^2 q^{-1} - \frac{1}{3 \cdot 2} \epsilon^3 q^{-2} - \dots \leq \epsilon - \frac{1}{2q} \epsilon^2$. (Analizując dokładniej, otrzymamy nieco silniejsze oszacowanie $\exp(-\epsilon^2 n / (2pq))$ dla $p \geq \frac{1}{2}$; kolejny wysiłek zostanie nagrodzony górnym ograniczeniem $\exp(-2\epsilon^2 n)$ dla dowolnego p).

Zamieniając role orła i reszki, otrzymujemy

$$\Pr(X \leq n(p-\epsilon)) = \Pr(n-X \geq n(q+\epsilon)) \leq e^{-\epsilon^2 n / (2p)}.$$

22. (a) Niech $x = r$ we wzorach (24) i (25). Zauważmy, że $q_k + p_k r = 1 + (r-1)p_k \leq e^{(r-1)p_k}$. [Zobacz H. Chernoff, *Annals of Math. Stat.* **23** (1952), 493–507].

(b) Niech $r = 1 + \delta$, gdzie $|\delta| \leq 1$. Wówczas $r^{-r} e^{r-1} = \exp(-\frac{1}{2 \cdot 1} \delta^2 + \frac{1}{3 \cdot 2} \delta^3 - \dots)$, co jest $\leq e^{\delta^2/2}$ dla $\delta \leq 0$ i $\leq e^{-\delta^2/3}$ dla $\delta \geq 0$.

(c) Funkcja $r^{-r} e^{r-1}$ maleje od 1 do 0, gdy r rośnie od 1 do ∞ . Dla $r \geq 2$ jest $\leq \frac{1}{2} e^{1/2} < 0.825$; dla $r \geq 4.32$ jest $< \frac{1}{2}$.

Tak się akurat składa, że nierówności ogonowe przy $x = r$ dają dokładnie to samo oszacowanie $(r^{-r} e^{r-1})^\mu$, gdy X ma rozkład Poissona z ćwiczenia 15.

23. Przyjmując $x = \frac{p-\epsilon}{p} \frac{q}{q-\epsilon}$ w (24), otrzymujemy nierówność $\Pr(X \leq n(p-\epsilon)) \leq ((\frac{p}{p-\epsilon})^{p-\epsilon} (\frac{q}{q})^{q-\epsilon})^n \leq e^{-\epsilon^2 n / (2pq)}$. Podobnie $x = \frac{p+\epsilon}{p} \frac{q}{q+\epsilon}$ daje $\Pr(X \geq n(p+\epsilon)) \leq ((\frac{p}{p+\epsilon})^{p+\epsilon} (\frac{q}{q})^{q+\epsilon})^n$. Niech $f(\epsilon) = (q+\epsilon) \ln(1 + \frac{\epsilon}{q}) - (p+\epsilon) \ln(1 + \frac{\epsilon}{p})$. Zauważmy, że $f'(\epsilon) = \ln(1 + \frac{\epsilon}{q}) - \ln(1 + \frac{\epsilon}{p})$. Dostajemy zatem $f(\epsilon) \leq -\epsilon^2 / (6pq)$.

1.2.11.1

1. Zero.

2. Każde wystąpienie O oznacza inną wartość; po lewej stronie możemy mieć nawet $f(n) - (-f(n)) = 2f(n)$, zatem nie możemy powiedzieć nic więcej niż $O(f(n)) - O(f(n)) = O(f(n))$, co wynika z (6) i (7). By udowodnić wzór (7), zauważmy, że jeśli $|x_n| \leq M|f(n)|$ dla $n \geq n_0$ i $|x'_n| \leq M'|f(n)|$ dla $n \geq n'_0$, to $|x_n \pm x'_n| \leq |x_n| + |x'_n| \leq (M + M')|f(n)|$ dla $n \geq \max(n_0, n'_0)$. (Podpisano J. H. Quick, student).

3. $n(\ln n) + \gamma n + O(\sqrt{n} \ln n)$.

4. $\ln a + (\ln a)^2/2n + (\ln a)^3/6n^2 + O(n^{-3})$.

5. Jeśli $f(n) = n^2$ i $g(n) = 1$, to n należy do zbioru $O(f(n) + g(n))$, ale nie do zbioru $f(n) + O(g(n))$, zatem hipoteza jest fałszywa.

6. Nieznana liczba n wystąpień symboli O została zastąpiona pojedynczym O przy błędnych założeniu, że pojedyncza wartość M „obsłuży” wszystkie wyrazy $|kn| \leq Mn$. Podana suma, jak wiemy, jest $\Theta(n^3)$. Ostatnia równość $\sum_{k=1}^n O(n) = O(n^2)$ jest prawdziwa.

7. Jeśli x jest dodatnie, z szeregu potęgowego 1.2.9–(22) odczytujemy nierówność $e^x > x^{m+1}/(m+1)!$. Zatem iloraz e^x/x^m nie jest ograniczony przez jakiekolwiek M .

8. Zastąp n przez e^n i skorzystaj z poprzedniego ćwiczenia.

9. Jeśli $|f(x)| \leq M|z|^m$ dla $|z| \leq r$, to $e^{f(z)} \leq e^{M|z|^m} = 1 + |z|^m(M + M^2|z|^m/2! + M^3|z|^{2m}/3! + \dots) \leq 1 + |z|^m(M + M^2r^m/2! + M^3r^{2m}/3! + \dots)$.

10. $\ln(1+O(z^m)) = O(z^m)$, gdy m jest dodatnią liczbą całkowitą. *Dowód:* Jeśli $f(z) = O(z^m)$, to istnieją liczby dodatnie $r < 1$, $r' < 1$ oraz stała M , takie że $|f(z)| \leq M|z|^m \leq r'$, gdy $|z| \leq r$. Wówczas $|\ln(1+f(z))| \leq |f(z)| + \frac{1}{2}|f(z)|^2 + \dots \leq |z|^m M(1 + \frac{1}{2}r' + \dots)$.

11. Korzystamy ze wzoru (12) przy $m = 1$ i $z = \ln n/n$. Możemy tak zrobić, ponieważ $\ln n/n \leq r$ dla dowolnego $r > 0$, gdy n jest wystarczająco duże.

12. Niech $f(z) = (ze^z/(e^z - 1))^{1/2}$. Jeżeliby $[_{1/2-k}]$ było $O(n^k)$, to z podanej tożsamości wynikałoby, że $[z^k] f(z) = O(n^k/(k-1)!)$, zatem $f(z)$ byłaby zbieżna dla $z = 2\pi i$. Ale $f(2\pi i) = \infty$.

13. *Dowód:* Możemy podstawić $L = 1/M$ w definicji O i Ω .

1.2.11.2

1. $(B_0 + B_1 z + B_2 z^2/2! + \dots) e^z = (B_0 + B_1 z + B_2 z^2/2! + \dots) + z$; następnie zastosuj wzór 1.2.9–(11).

2. Funkcja $B_{m+1}(\{x\})$ musi być ciągła, by można było całkować przez części.

3. $|R_{mn}| \leq |B_m/(m!)| \int_1^n |f^{(m)}(x)| dx$. [Uwagi: Mamy $B_m(x) = (-1)^m B_m(1-x)$, a $B_m(x)$ to $m!$ razy współczynnik przy z^m w $ze^{xz}/(e^z - 1)$. W szczególności z faktu, że $e^{z/2}/(e^z - 1) = 1/(e^{z/2} - 1) - 1/(e^z - 1)$, wynika $B_m(\frac{1}{2}) = (2^{1-m} - 1)B_m$. Nietrudno udowodnić, że maksimum $|B_m - B_m(x)|$ dla $0 \leq x \leq 1$ wypada w $x = \frac{1}{2}$ dla dodatnich m . Dla $m = 2k \geq 4$ możemy zapisać wielkości R_{mn} i C_{mn} w prostej postaci R_m i C_m . Mamy $R_{m-2} = C_m + R_m = \int_1^n (B_m - B_m(\{x\})) f^{(m)}(x) dx / m!$, przy tym $B_m - B_m(\{x\})$ znajduje się między 0 i $(2 - 2^{1-m})B_m$. Stąd R_{m-2} leży między 0 i $(2 - 2^{1-m})C_m$. Otrzymujemy stąd nieco silniejszy wynik: R_m leży między $-C_m$ i $(1 - 2^{1-m})C_m$. Widać, że gdy $f^{(m+2)}(x) f^{(m+4)}(x) > 0$ dla $1 < x < n$, to wielkości C_{m+2} i C_{m+4} mają przeciwnie znaki, podczas gdy R_m ma znak C_{m+2} , a R_{m+2} ma znak C_{m+4} oraz $|R_{m+2}| \leq |C_{m+2}|$. To dowodzi (13). Zobacz J. F. Steffensen, *Interpolation* (Baltimore: 1937), §14].

$$\text{4. } \sum_{0 \leq k < n} k^m = \frac{n^{m+1}}{1+m} + \sum_{k=1}^m \frac{B_k}{k!} \frac{m!}{(m-k+1)!} n^{m-k+1} = \frac{1}{m+1} B_{m+1}(n) - \frac{1}{m+1} B_{m+1}.$$

5. Mamy

$$\kappa = \sqrt{2} \lim_{n \rightarrow \infty} \frac{2^{2n} (n!)^2}{\sqrt{n} (2n)!};$$

$$\kappa^2 = \lim_{n \rightarrow \infty} \frac{2}{n} \frac{n^2(n-1)^2 \dots (1)^2}{(n-\frac{1}{2})^2(n-\frac{3}{2})^2 \dots (\frac{1}{2})^2} = 4 \frac{2 \cdot 2 \cdot 4 \cdot 4 \dots}{1 \cdot 3 \cdot 3 \cdot 5 \dots} = 2\pi.$$

6. Założmy, że $c > 0$ i rozważmy $\sum_{0 \leq k < n} \ln(k+c)$. Wówczas mamy

$$\begin{aligned} \ln(c(c+1) \dots (c+n-1)) &= (n+c) \ln(n+c) - c \ln c - n - \frac{1}{2} \ln(n+c) + \frac{1}{2} \ln c \\ &\quad + \sum_{1 < k \leq m} \frac{B_k(-1)^k}{k(k-1)} \left(\frac{1}{(n+c)^{k-1}} - \frac{1}{c^{k-1}} \right) + R_{mn}. \end{aligned}$$

Mamy także

$$\ln(n-1)! = (n - \frac{1}{2}) \ln n - n + \sigma + \sum_{1 < k \leq m} \frac{B_k(-1)^k}{k(k-1)} \left(\frac{1}{n^{k-1}} \right) - \frac{1}{m} \int_n^\infty \frac{B_m(\{x\}) dx}{x^m}.$$

Teraz $\ln \Gamma_{n-1}(c) = c \ln(n-1) + \ln(n-1)! - \ln(c \dots (c+n-1))$. Podstawiając i przechodząc z $n \rightarrow \infty$, otrzymujemy

$$\ln \Gamma(c) = -c + (c - \frac{1}{2}) \ln c + \sigma + \sum_{1 < k \leq m} \frac{B_k(-1)^k}{k(k-1)c^{k-1}} - \frac{1}{m} \int_0^\infty \frac{B_m(\{x\}) dx}{(x+c)^m}.$$

Widać stąd, że $\Gamma(c+1) = ce^{\ln \Gamma(c)}$ ma takie samo rozwinięcie jak $c!$.

7. A $n^{n^2/2+n/2+1/12}e^{-n^2/4}$, gdzie A jest stałą. Ten wynik otrzymujemy, stosując wzór sumacyjny Eulera do $\sum_{k=1}^{n-1} k \ln k$. Dokładniejszy wzór uzyskamy, mnożąc powyższą odpowiedź przez

$$\exp(-B_4/(2 \cdot 3 \cdot 4n^2) - \dots - B_{2t}/((2t-2)(2t-1)(2t)n^{2t-2}) + O(1/n^{2t})).$$

Liczba A jest „stałą Glaishera” 1.2824271... [Messenger of Math. 7 (1877), 43–47]. Można pokazać, że jest ona równa $e^{1/12-\zeta'(-1)} = (2\pi e^{\gamma-\zeta'(2)/\zeta(2)})^{1/12}$ [de Bruijn, Asymptotic Methods in Analysis, §3.7].

8. Mamy na przykład $\ln(an^2 + bn) = 2 \ln n + \ln a + \ln(1 + b/(an))$. Stąd odpowiedzią na pierwsze pytanie jest $2an^2 \ln n + a(\ln a - 1)n^2 + 2bn \ln n + bn \ln a + \ln n + b^2/(2a) + \sigma + (3a - b^2)b/(6a^2n) + O(n^{-2})$. Rozliczne możliwości uproszczeń ułatwiają obliczenie $\ln(cn^2)! - \ln(cn^2 - n)! - n \ln c - \ln n^2! + \ln(n^2 - n)! = (c-1)/(2c) - (c-1)(2c-1)/(6c^2n) + O(n^{-2})$. Odpowiedzią jest

$$e^{(c-1)/(2c)} \left(1 - \frac{(c-1)(2c-1)}{6c^2n} \right) (1 + O(n^{-2})).$$

Tak się akurat składa, że $\binom{cn^2}{n} / c^n \binom{n^2}{n}$ można zapisać jako $\prod_{j=1}^{n-1} (1 + \alpha j/(n^2 - j))$, gdzie $\alpha = 1 - 1/c$.

9. (a) Mamy $\ln(2n)! = (2n + \frac{1}{2}) \ln 2n - 2n + \sigma + \frac{1}{24n} + O(n^{-3})$ oraz $\ln(n!)^2 = (2n + 1) \ln n - 2n + 2\sigma + \frac{1}{6n} + O(n^{-3})$, stąd $\binom{2n}{n} = \exp(2n \ln 2 - \frac{1}{2} \ln \pi n - \frac{1}{8n} + O(n^{-3})) = 2^{2n} (\pi n)^{-1/2} (1 - \frac{1}{8} n^{-1} + \frac{1}{128} n^{-2} + O(n^{-3}))$. (b) Ponieważ $\binom{2n}{n} = 2^{2n} \binom{n-1/2}{n}$ i $\binom{n-1/2}{n} = \Gamma(n+1/2)/(n\Gamma(n)\Gamma(1/2)) = n^{-1} n^{1/2} / \sqrt{\pi}$, ze wzoru 1.2.11.1-(16) uzyskujemy taki sam wynik, gdyż

$$\begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} = 1, \quad \begin{bmatrix} 1/2 \\ -1/2 \end{bmatrix} = \binom{1/2}{2} = -\frac{1}{8}, \quad \begin{bmatrix} 1/2 \\ -3/2 \end{bmatrix} = \binom{1/2}{4} + 2 \binom{3/2}{4} = \frac{1}{128}.$$

Z metody (b) wynika, dlaczego wszystkie mianowniki w

$$\binom{2n}{n} = \frac{2^{2n}}{\sqrt{\pi n}} \left(1 - \frac{n^{-1}}{8} + \frac{n^{-2}}{128} + \frac{5n^{-3}}{1024} - \frac{21n^{-4}}{32768} - \frac{399n^{-5}}{262144} + \frac{869n^{-6}}{4194304} + O(n^{-7}) \right)$$

są potęgami dwójką [Knuth, Vardi, AMM 97 (1990), 629–630].

1.2.11.3

1. Całkowanie przez części.
2. W całce podstaw rozwinięcie e^{-t} .

3. Zobacz równanie 1.2.9–(11) oraz ćwiczenie 1.2.6–48.

4. $1 + 1/u$ jest ograniczone jako funkcja v , gdyż zbiega do zera przy v zmieniającym się od r do nieskończoności. Zastąp to wyrażenie przez M , a wynikowa całka przybierze postać Me^{-rx} .

5. $f''(x) = f(x)((n+1/2)(n-1/2)/x^2 - (2n+1)/x + 1)$ zmienia znak w punkcie $r = n + 1/2 - \sqrt{n+1/2}$, zatem $|R| = O(\int_0^n |f''(x)| dx) = O(\int_0^r f''(x) dx - \int_r^n f''(x) dx) = O(f'(n) - 2f'(r) + f'(0)) = O(f(n)/\sqrt{n})$.

6. Mamy $n^{n+\beta} \exp((n+\beta)(\alpha/n - \alpha^2/2n^2 + O(n^{-3})))$ itd.

7. W funkcji podcałkowej potraktowanej jako szereg potęgowy zmiennej x^{-1} jest $x^{-n} = O(u^{2n})$. Po całkowaniu wyrazy w x^{-3} to $Cu^7/x^3 = O(x^{-5/4})$ itd. W odpowiedzi uzyskujemy $O(x^{-2})$, pomijając wyrazy u^n/x^m przy $4m-n \geq 9$. Stąd rozwinięcie iloczynu $\exp(-u^2/2x) \exp(u^3/3x^2) \dots$ prowadzi wprost do odpowiedzi

$$yx^{1/4} - \frac{y^3}{6}x^{-1/4} + \frac{y^5}{40}x^{-3/4} + \frac{y^4}{12}x^{-1} - \frac{y^7}{336}x^{-5/4} - \frac{y^6}{36}x^{-3/2} + \left(\frac{y^9}{3456} - \frac{y^5}{20} \right)x^{-7/4} + O(x^{-2}).$$

8. (Autor rozwiązania: Miklós Simonovits) Mamy $|f(x)| < x$, gdy x jest odpowiednio duże. Niech $R(x) = \int_0^{|f(x)|} (e^{-g(u,x)} - e^{-h(u,x)}) du$ będzie różnicą pomiędzy danymi całkami, gdzie $g(u,x) = u - x \ln(1+u/x)$ i $h(u,x) = u^2/2x - u^3/3x^2 + \dots + (-1)^m u^m/mx^{m-1}$. Zauważmy, że $g(u,x) \geq 0$ i $h(u,x) \geq 0$ dla $|u| < x$; również $g(u,x) = h(u,x) + O(u^{m+1}/x^m)$.

Z twierdzenia o wartości średniej $e^a - e^b = (a - b)e^c$ dla pewnego c między a i b . Stąd $|e^a - e^b| \leq |a - b|$, gdy $a, b \leq 0$. Mamy zatem

$$\begin{aligned} |R(x)| &\leq \int_{-|f(x)|}^{|f(x)|} |g(u,x) - h(u,x)| du = O\left(\int_{-Mx^r}^{Mx^r} \frac{u^{m+1} du}{x^m}\right) \\ &= O(x^{(m+2)r-m}) = O(x^{-s}). \end{aligned}$$

9. Możemy założyć, że $p \neq 1$, ponieważ przypadek $p = 1$ jest pokazany w twierdzeniu A. Zakładamy też, że $p \neq 0$, ponieważ przypadek $p = 0$ jest trywialny.

Przypadek 1: $p < 1$. Podstawiamy $t = px(1-u)$ i $v = -\ln(1-u) - pu$. Mamy $dv = ((1-p+pu)/(1-u)) du$, zatem przekształcenie jest monotoniczne dla $0 \leq u \leq 1$, dostajemy więc całkę postaci

$$\int_0^\infty x e^{-xv} dv \left(\frac{1-u}{1-p+pu} \right).$$

Z uwagi na fakt, że wielkość w nawiasach to $(1-p)^{-1}(1-v(1-p)^{-2} + \dots)$, odpowiedzią jest

$$\frac{p}{1-p} (pe^{1-p})^x \frac{e^{-x} x^z}{\Gamma(x+1)} \left(1 - \frac{1}{(p-1)^2 x} + O(x^{-2}) \right).$$

Przypadek 2: $p > 1$. To jest $1 - \int_{px}^\infty (\dots)$. W całce podstaw $t = px(1+u)$ i $v = pu - \ln(1+u)$, dalej jak w przypadku 1. Odpowiedzią okazuje się ten sam wzór, co w przypadku 1 plus jeden. Zauważ, że $pe^{1-p} < 1$, zatem $(pe^{1-p})^x$ jest bardzo małe.

W odpowiedzi do ćwiczenia 11 jest pokazany inny sposób rozwiązania tego zadania.

10. $\frac{p}{p-1} (pe^{1-p})^x e^{-x} x^x \left(1 - e^{-y} - \frac{e^{-y}(e^y - 1 - y - y^2/2)}{x(p-1)^2} + O(x^{-2}) \right).$

11. Po pierwsze, $xQ_x(n) + R_{1/x}(n) = n!(x/n)^n e^{n/x}$ jest uogólnieniem (4). Po wtóre, mamy $R_x(n) = n!(e^x/nx)^n \gamma(n, nx)/(n-1)!$, co jest uogólnieniem (9). Ponieważ $a\gamma(a, x) = \gamma(a+1, x) + e^{-x}x^a$, możemy napisać $R_x(n) = 1 + (e^x/nx)^n \gamma(n+1, nx)$, naginając problem do ćwiczenia 9. Możemy także podejść do $Q_x(n)$ i $R_x(n)$ wprost, korzystając ze wzoru 1.2.9–(27) i (28) w celu wyprowadzenia rozwinięcia w szereg zawierający liczby Stirlinga:

$$1 + xQ_x(n) = \sum_{k \geq 0} x^k n^k / n^k = \sum_{k,m} \frac{(-1)^m}{n^m} \begin{Bmatrix} k \\ k-m \end{Bmatrix} x^k;$$

$$R_x(n) = \sum_{k \geq 0} x^k n^k / (n+1)^k = \sum_{k,m} \frac{(-1)^m}{n^m} \left\{ \begin{matrix} k+m \\ k \end{matrix} \right\} x^k.$$

Sumy względem k są zbieżne dla ustalonego m , gdy $|x| < 1$, a przy $|x| > 1$ możemy skorzystać ze związku między $Q_x(n)$ i $R_{1/x}(n)$. To prowadzi do wzorów

$$Q_x(n) = \frac{1}{1-x} - \frac{x}{(1-x)^3 n} + \cdots + \frac{(-1)^m q_m(x)}{(1-x)^{2m+1} n^m} + O(n^{-1-m}),$$

$$R_x(n) = \frac{1}{1-x} - \frac{x}{(1-x)^3 n} + \cdots + \frac{(-1)^m r_m(x)}{(1-x)^{2m+1} n^m} + O(n^{-1-m}), \quad \text{gdy } x < 1;$$

$$Q_x(n) = \frac{n! x^{n-1} e^{n/x}}{n^n} + \frac{1}{1-x} - \frac{x}{(1-x)^3 n} + \cdots + \frac{(-1)^m q_m(x)}{(1-x)^{2m+1} n^m} + O(n^{-1-m}),$$

$$R_x(n) = \frac{n! e^{nx}}{n^n x^n} + \frac{1}{1-x} - \frac{x}{(1-x)^3 n} + \cdots + \frac{(-1)^m r_m(x)}{(1-x)^{2m+1} n^m} + O(n^{-1-m}), \quad \text{gdy } x > 1,$$

gdzie

$$q_m(x) = \left\langle \begin{matrix} m \\ 0 \end{matrix} \right\rangle x^{2m-1} + \left\langle \begin{matrix} m \\ 1 \end{matrix} \right\rangle x^{2m-2} + \cdots$$

i

$$r_m(x) = \left\langle \begin{matrix} m \\ 0 \end{matrix} \right\rangle x + \left\langle \begin{matrix} m \\ 1 \end{matrix} \right\rangle x^2 + \cdots$$

są wielomianami, których współczynniki są „liczbami Eulera drugiego rzędu” [CMath §6.2; zobacz L. Carlitz, Proc. Amer. Math. Soc. **16** (1965), 248–252]. Przypadek $x = -1$ jest delikatny, ale można sobie poradzić, korzystając z ciągłości, ponieważ ograniczenie w $O(n^{-1-m})$ jest niezależne od x , gdy $x < 0$. Ciekawe, że $R_{-1}(n) - Q_{-1}(n) = (-1)^n n! / e^n n^n \approx (-1)^n \sqrt{2\pi n} / e^{2n}$ jest niesamowicie małe.

12. $\gamma(\frac{1}{2}, \frac{1}{2}x^2)/\sqrt{2}$.

13. Zobacz P. Flajolet, P. Grabner, P. Kirschenhofer, H. Prodinger, J. Computational and Applied Math. **58** (1995), 103–116.

15. Rozwijając funkcję podcałkową, z twierdzenia dwumianowego otrzymujemy wartość $1 + Q(n)$.

16. Zapisz $Q(k)$ w postaci sumy i zmień kolejność sumowania, korzystając z 1.2.6–(53).

17. $S(n) = \sqrt{\pi n/2} + \frac{2}{3} - \frac{1}{24}\sqrt{\pi/2n} - \frac{4}{135}n^{-1} + \frac{49}{1152}\sqrt{\pi/2n^3} + O(n^{-2})$. [Zauważ, że $S(n+1) + P(n) = \sum_{k \geq 0} k^{n-k} k! / n!$, podczas gdy $Q(n) + R(n) = \sum_{k \geq 0} n! / k! n^{n-k}$].

18. Niech $S_n(x, y) = \sum_k \binom{n}{k} (x+k)^k (y+n-k)^{n-k}$. Wówczas dla $n > 0$ mamy $S_n(x, y) = x \sum_k \binom{n}{k} (x+k)^{k-1} (y+n-k)^{n-k} + n \sum_k \binom{n-1}{k} (x+1+k)^k (y+n-1-k)^{n-1-k} = (x+y+n)^n + n S_{n-1}(x+1, y)$ ze wzoru Abela 1.2.6–(16), skąd otrzymujemy wzór

$S_n(x, y) = \sum_k \binom{n}{k} k! (x + y + n)^{n-k}$. [Ten wzór pochodzi od Cauchy'ego, który udowodnił go za pomocą rachunku residuów; zobacz tegoż autora *Euvres* (2) 6, 62–73]. Rozpatrywane sumy to odpowiednio $n^n(1+Q(n))$ i $(n+1)^nQ(n+1)$.

19. Przypuśćmy, że C_n istnieje dla wszystkich $n \geq N$ i $|f(x)| \leq Mx^\alpha$ dla $0 \leq x \leq r$. Niech $F(x) = \int_r^x e^{-Nt} f(t) dt$. Wówczas dla $n > N$ mamy

$$\begin{aligned} |C_n| &\leq \int_0^r e^{-nx} |f(x)| dx + \left| \int_r^\infty e^{-(n-N)x} e^{-Nx} f(x) dx \right| \\ &\leq M \int_0^r e^{-nx} x^\alpha dx + (n-N) \left| \int_r^\infty e^{-(n-N)x} F(x) dx \right| \\ &\leq M \int_0^\infty e^{-nx} x^\alpha dx + (n-N) \sup_{x \geq r} |F(x)| \int_r^\infty e^{-(n-N)x} dx \\ &= M\Gamma(\alpha + 1)n^{-1-\alpha} + \sup_{x \geq r} |F(x)| e^{-(n-N)r} = O(n^{-1-\alpha}). \end{aligned}$$

[E. W. Barnes, *Phil. Trans. A* **206** (1906), 249–297; G. N. Watson, *Proc. London Math. Soc.* **17** (1918), 116–148].

20. [C. C. Rousseau, *Applied Math. Letters* **2** (1989), 159–161] Mamy $Q(n) + 1 = n \int_0^\infty e^{-nx} (1+x)^n dx = n \int_0^\infty e^{-n(x-\ln(1+x))} dx = n \int_0^\infty e^{-nu} g(u) du$ po podstawieniu $u = x - \ln(1+x)$ i oznaczeniu $g(u) = dx/du$. Zauważmy, że $x = \sum_{k=1}^\infty c_k (2u)^{k/2}$, gdy u jest wystarczająco małe. Stąd $g(u) = \sum_{k=1}^{m-1} c_k (2u)^{k/2-1} + O(u^{m/2-1})$. Możemy zatem zastosować lemat Watsona do $Q(n) + 1 - n \int_0^\infty e^{-nu} \sum_{k=1}^{m-1} k c_k (2u)^{k/2-1} du$.

1.3.1

1. Cztery. Każdy bajt mógłby reprezentować $3^4 = 81$ różnych wartości.
2. Pięć, bo pięć bajtów zawsze wystarczy. Cztery bajty mogą nie wystarczyć.
3. (0:2); (3:3); (4:4); (5:5).
4. Zapewne rI4 zawiera wartość większą lub równą 2000, zatem po indeksowaniu otrzymujemy adres poprawny.

5. „DIV -80,3(0:5)” lub prościej „DIV -80,3”.

6. (a) rA \leftarrow

-	5	1	200	15
---	---	---	-----	----

. (b) rI2 \leftarrow -200. (c) rX \leftarrow

+	0	0	5	1	?
---	---	---	---	---	---

.

(d) Wynik niezdefiniowany. Nie można ładować tak dużej wartości do rejestru indeksowego. (e) rX \leftarrow

-	0	0	0	0	0
---	---	---	---	---	---

.

7. Niech $n = |\text{rAX}|$ będzie wartością w rejestrach A i X przed operacją, niech $d = |V|$ będzie wartością dzielnika. Po wykonaniu rozkazu wartość w rA wynosi $\lfloor n/d \rfloor$, a wartość rX to $n \bmod d$. Znak w rX po operacji jest znakiem rA przed operacją. Znak w rA po operacji to +, jeżeli poprzednie znaki rA i V były jednakowe, lub - w przeciwnym przypadku.

Innymi słowy, jeżeli znaki rA i V są takie same, $\text{rA} \leftarrow [\text{rAX}/V]$ i $\text{rX} \leftarrow \text{rAX} \bmod V$. W przeciwnym razie $\text{rA} \leftarrow [\text{rAX}/V]$ i $\text{rX} \leftarrow \text{rAX} \bmod -V$.

8. rA \leftarrow

+	0	617	0	1
---	---	-----	---	---

; rX \leftarrow

-	0	0	0	1	1
---	---	---	---	---	---

.

9. ADD, SUB, DIV, NUM, JOV, JNOV, INCA, DECA, INCX, DECX.

10. CMPA, CMP1, CMP2, CMP3, CMP4, CMP5, CMP6, CMPX. (I jeszcze operacja zmiennopozycyjna FCMP).

11. MOVE, LD1, LD1N, INC1, DEC1, ENT1, ENN1.

12. INC3 0,3.

13. Dla „J0V 1000” nie ma żadnej różnicy poza czasem wykonania. „JNOV 1001” w większości wypadków inaczej ustawia wartość rJ. Dla „JNOV 1000” różnica jest zasadnicza, gdyż rozkaz może spowodować wejście w nieskończoną pętlę, czyli zawieszenie komputera.

14. NOP z dowolną zawartością pól; ADD, SUB przy F = (0:0) lub gdy pole adresu równa się * (tj. adresowi lokacji zawierającej rozkaz) i F = (3:3); HLT (przy odrobinie dobrej woli); dowolna operacja przesunięcia, gdy pola adresu i indeksu są równe zeru; SLC lub SRC, gdy indeks jest równy零, a adres jest wielokrotnością 10; MOVE, gdy F = 0; JSJ *+1; INC lub DEC, gdy adres i indeks są równe zeru. „ENT1 0,1” nie zawsze nic nie robi, bo może zmienić rI1 z -0 na +0.

15. 70; 80; 120. (Rozmiar bloku razy 5).

16. (a) STZ 0; ENT1 1; MOVE 0(49); MOVE 0(50). Jeśli wiedzielibyśmy, że rozmiar bajtu równa się 100, wystarczyłby tylko jeden rozkaz MOVE, ale nie wolno nam robić żadnych założeń na temat rozmiaru bajtu. (b) 100 rozkazów STZ.

17. (a) STZ 0,2; DEC2 1; J2NN 3000.

(b)

STZ	0
ENT1	1
JMP	3004
(3003)	MOVE 0(63)
(3004)	DEC2 63
	J2P 3003
	INC2 63
	ST2 3008(4:4)
(3008)	MOVE 0

(Istnieje rozwiązanie nieco szybsze, acz szalone, składające się z 993 rozkazów STZ: JMP 3995; STZ 1,2; STZ 2,2; ...; STZ 993,2; J2N 3999; DEC2 993; J2NN 3001; ENN1 0,2; JMP 3000,1).

18. (Jeśli Czytelnik poprawnie śledził wykonanie rozkazów, powinien odkryć przepełnienie przy ADD, po którym w rA zostaje minus zero).

Odpowiedź: Znacznik przepełnienia jest włączony, wskaźnik porównania ma wartość EQUAL, rA ma wartość

-	30	30	30	30	30
---	----	----	----	----	----

, rX ma wartość

-	31	30	30	30	30
---	----	----	----	----	----

, rI1 ma wartość +3, a w komórkach pamięci o numerach 0001 i 0002 znajdują się wartości +0 (chyba, że jest tam umieszczony program).

19. $42u = (2 + 1 + 2 + 2 + 1 + 1 + 1 + 2 + 2 + 1 + 2 + 2 + 3 + 10 + 10)u$.

20. (Autor rozwiązania: H. Fukuoka)

(3991)	ENT1 0
	MOVE 3995 (standardowe F dla MOVE wynosi 1)
(3993)	MOVE 0(43) (3999 = 93 razy 43)
	JMP 3993
(3995)	HLT 0

21. (a) Nie, jeżeli nie można go w jakiś sposób ustawić „zewnętrznie” (zobacz „przykłady START”, ćwiczenie 26), ponieważ program może spowodować $rJ \leftarrow N$ wyłącznie w wyniku skoku spod adresu $N - 1$.

(b)

LDA	-1,4
LDX	3004
STX	-1,4
JMP	-1,4
(3004) JMP	3005
(3005) STA	-1,4

22. *Minimalny czas:* Jeśli b jest rozmiarem bajtu, to z założenia $|X^{13}| < b^5$ można wywnioskować, że $X^2 < b$, zatem X^2 mieści się w jednym bajcie. Ten fakt został wykorzystany w pomysłowym rozwiążaniu zaproponowanym przez Y. N. Patta. Znak rA jest znakiem X .

(3000) LDA 2000
 MUL 2000(1:5)
 STX 3500(1:1)
 SRC 1
 MUL 3500
 STA 3501
 ADD 2000
 MUL 3501(1:5)
 STX 3501
 MUL 3501(1:5)
 SLAX 1
 HLT 0
 (3500) NOP 0
 (3501) NOP 0

	rA		rX		
	X^2	0	0	0	0
	X^4	0	0	0	0
	X^4	0	0	0	0
	X^4	0	0	X	0
	X^8		0	X^5	0
	X^8		0	X^5	0
	0	X^{13}		0	0
	X^{13}		0	0	0

pamięć = 14; czas = 54u, nie licząc HLT.

Zgodnie z teorią przedstawioną w punkcie 4.6.3, pięć operacji mnożenia jest „niezbędnych”, a tu mamy tylko cztery! W istocie jest jeszcze lepsze rozwiązanie:

Minimalna pamięć: (3000) ENT4 12 DEC4 1
 LDA 2000 J4P 3002
 (3002) MUL 2000 HLT 0
 SLAX 5 pamięć = 7; czas = 171u.

Naprawdę minimalny czas: Jak zauważył R. W. Floyd, z warunków wynika, że $|X| \leq 5$, zatem minimalny czas wykonania osiągniemy, tablicując wartości:

(3000) LD1 2000
 LDA 3500,1
 HLT 0
 (3495) $(-5)^{13}$
 (3496) $(-4)^{13}$
 :
 (3505) $(+5)^{13}$

pamięć = 14; czas = 4u.

23. Wydaje się, że poniższe rozwiązanie spełnia wszystkie warunki. (Autor rozwiązania: R. D. Dixon).

(3000) ENT1 4	DEC1 1
(3001) LDA 200	J1NN 3001
SRA 0,1	SLAX 5
SRAX 1	HLT 0 ■

24. (a) DIV 3500, gdzie 3500 =

	+	1	0	0	0	0
--	---	---	---	---	---	---

.

(b) SRC 4; SRA 1; SLC 5.

25. Niektóre pomysły: (a) Rzeczy oczywiste: szybsza pamięć, więcej urządzeń wejścia-wyjścia. (b) Pole I można wykorzystać do indeksowania rejestru J i/lub indeksowania wielokrotnego (tj. do indeksowania dwoma różnymi rejestrami indeksowymi) i/lub adresowania pośredniego (ćwiczenia 2.2.2–3, 4, 5). (c) Rejestry indeksowe i rejestr J można rozszerzyć do pięciu bajtów; lokacje o wyższych adresach byłyby dostępne jedynie przez indeksowanie, ale można by to zaakceptować wobec obecności indeksowania wielokrotnego opisanego w (b). (d) Można wprowadzić mechanizm przerwań wykorzystujący adresy ujemne, jak w ćwiczeniu 1.4.4–18. (e) Przydałby się „zegar czasu rzeczywistego” w ujemnych adresach. (f) W dwójkowych modelach komputera MIX można wprowadzić operacje bitowe, skoki warunkowe zależne od parzystości rejestrów i przesunięcia bitowe (zobacz na przykład ćwiczenie 2.5–28, 5.2.2–12 i 6.3–9; także program 4.5.2B, 6.4–(24) oraz podrozdział 7.1). (g) Rozkaz „wykonaj”, oznaczający wykonanie rozkazu znajdującego się pod adresem M, mógłby stanowić kolejny wariant C = 5. (h) Inne warianty C = 48, . . . , 55 mogą oznaczać operacje CI ← rejestr : M.

26. Aż ręce świerzbią, by kolumny 7–10 odczytywać, manipulując wycinkiem (2:5), ale tego się zrobić nie da, bo $2 \cdot 8 + 5 = 21$. W celu uczynienia programu bardziej czytelnym przedstawiamy go w postaci symbolicznej, uprzedzając punkt 1.3.2.

			<i>Wydziurkowane znaki:</i>
BUFF EQU	29	Bufor w obszarze 0029–0044.	
ORIG	0		
00LOC IN	16(16)	Wczytaj drugą kartę.	„0„06
01 READ IN	BUFF(16)	Wczytaj następną kartę.	„Z„06
02 LD1	0(0:0)	rI1 ← 0.	„uuuuI
03 JBUS	*(16)	Poczekaj na zakończenie odczytu.	„C„04
04 LDA	BUFF+1	rA ← kolumny 6–10.	„0„EH
05=1= SLA	1		„A„uF
06 SRAX	6	rAX ← kolumny 7–10.	„F„CF
07=30= NUM	30		„0„uE
08 STA	LOC	LOC ← lokacja początkowa.	„uuuEU
09 LDA	BUFF+1(1:1)		„0„IH
10 SUB	=30=(0:2)		„G„BB
11 LOOP LD3	LOC	rI3 ← LOC.	„uuuEJ
12 JAZ	0,3	Skocz, jeśli karta transferowa.	„uCA.
13 STA	BUFF	BUFF ← licznik.	„Z„EU
14 LDA	LOC		„uuuEH
15 ADD	=1=(0:2)		„E„BA
16 STA	LOC	LOC ← LOC + 1.	„uuuEU
17 LDA	BUFF+3,1(5:5)		„2A-H
18 SUB	=25=(0:2)		„S„BB

19	STA	0,3(0:0)	Zapamiętaj znak.	uu <u>C</u> <u>U</u>
20	LDA	BUFF+2,1		u <u>1AEH</u>
21	LDX	BUFF+3,1		u <u>2AEN</u>
22=25=	NUM	25		u <u>V</u> <u>U</u> <u>E</u>
23	STA	0,3(1:5)	Zapamiętaj wartość.	uu <u>CLU</u>
24	MOVE	0,1(2)	rI1 \leftarrow rI1 + 2. (!)	uu <u>ABG</u>
25	LDA	BUFF		u <u>Z</u> <u>U</u> <u>EH</u>
26	SUB	=1=(0:2)	Zmniejsz licznik.	u <u>E</u> <u>BB</u>
27	JAP	LOOP	Powtarzaj, dopóki licznik nie jest zerem.	u <u>J</u> <u>U</u> <u>B</u> .
28	JMP	READ	Wczytaj nową kartę.	u <u>A</u> <u>U</u> <u>9</u>

1.3.2

1. ENTX 1000; STX X.

2. Rozkaz STJ w wierszu 03 zmienia ten adres. (Przyjęto konwencję, że w takich rozkazach adres oznaczamy przez „*”, bo to jest czytelne, a także pozwala zauważać sytuacje niepoprawnego wejścia do procedury (spowodowanego jakimś przeoczeniem). Niektórzy programiści wolą pisać „*-*”).

3. Wczytaj 100 słów z urządzenia taśmowego nr zero; zamień największe z nich z ostatnim ze słów; zamień największe z pozostałych 99 słów z ostatnim z nich itd. W końcu 100 słów zostanie posortowanych w porządku niemalejącym. Wynik jest zapisywany na urządzeniu taśmowym numer jeden. (Porównaj z algorytmem 5.2.3S).

4. Lokacje o zawartości różnej od zera:

3000:	+	0000	00	18	35
3001:	+	2051	00	05	09
3002:	+	2050	00	05	10
3003:	+	0001	00	00	49
3004:	+	0499	01	05	26
3005:	+	3016	00	01	41
3006:	+	0002	00	00	50
3007:	+	0002	00	02	51
3008:	+	0000	00	02	48
3009:	+	0000	02	02	55
3010:	-	0001	03	05	04
3011:	+	3006	00	01	47
3012:	-	0001	03	05	56
3013:	+	0001	00	00	51
3014:	+	3008	00	06	39
3015:	+	3003	00	00	39
3016:	+	1995	00	18	37
3017:	+	2035	00	02	52
3018:	-	0050	00	02	53
3019:	+	0501	00	00	53
3020:	-	0001	05	05	08

3021:	+	0000	00	01	05
3022:	+	0000	04	12	31
3023:	+	0001	00	01	52
3024:	+	0050	00	01	53
3025:	+	3020	00	02	45
3026:	+	0000	04	18	37
3027:	+	0024	04	05	12
3028:	+	3019	00	00	45
3029:	+	0000	00	02	05
0000:	+				2
1995:	+	06	09	19	22
1996:	+	00	06	09	25
1997:	+	00	08	24	15
1998:	+	19	05	04	00
1999:	+	19	09	14	05
2024:	+				2035
2049:	+				2010
2050:	+				3
2051:	-				499

(ostatnie dwie lokacje mogą być zamienione miejscami, co musi zostać odzwierciedlone w 3001 i 3002)

5. Każdy rozkaz OUT jest wstrzymywany do czasu zakończenia poprzedniej operacji na urządzeniu (tu: drukowania danych z drugiego bufora).

6. (a) Jeśli n nie jest liczbą pierwszą, to z definicji n ma dzielnik d , taki że $1 < d < n$. Jeśli $d > \sqrt{n}$, to n/d jest dzielnikiem, takim że $1 < n/d < \sqrt{n}$. (b) Jeśli N nie jest liczbą pierwszą, to N ma dzielnik pierwoszy d , taki że $1 < d \leq \sqrt{N}$. Algorytm sprawdził, że N nie ma dzielników pierwszych $\leq p = \text{PRIME}[K]$; ponadto $N = pQ + R < pQ + p \leq p^2 + p < (p + 1)^2$. Dowolny dzielnik pierwoszy N jest zatem większy od $p + 1 > \sqrt{N}$.

Musimy jeszcze udowodnić, że znajdzie się odpowiednio duża liczba pierwsza mniejsza niż N , gdy N jest liczbą pierwszą, tzn. że $(k+1)$ -sza liczba pierwsza p_{k+1} jest mniejsza niż $p_k^2 + p_k$. W przeciwnym razie K przekroczyłoby J , a $\text{PRIME}[K]$ byłoby zerem, gdy powinno być duże. Odpowiedni dowód wynika z „postulatu Bertranda”: jeśli p jest liczbą pierwszą, to istnieje większa liczba pierwsza mniejsza niż $2p$.

7. (a) Symbol odpowiada lokacji wiersza 29. (b) Program byłby błędny. W wierszu 14 odwoływalibyśmy się do wiersza 15 zamiast 25, a w wierszu 24 odwoływalibyśmy się do wiersza 15 zamiast 12.

8. Program powoduje wydrukowanie 100 wierszy. Jeśli wydrukować te 12000-znakowe wiersze bez łamania, każdy z nich składałby się z pięciu spacji, po których następuje pięć A, po których następuje 10 spacji, po których następuje pięć A, po których następuje piętnaście spacji ... po których następuje $5k$ spacji, po których następuje pięć A, po których następuje $5(k + 1)$ spacji ... aż uzbiera się 12000 znaków. Trzeci wiersz od końca urywa się na AAAAA, po którym stoi 35 spacji. Dwa ostatnie wiersze są puste. Ogólnie rzecz ujmując, programu drukuje jedno z dzieł OP-artu.

9. W wycinku (4:4) pól poniższej tabeli przechowywana jest największa dopuszczalna wartość F. Wycinek (1:2) odpowiada lokacji odpowiedniego podprogramu sprawdzającego poprawność.

```

B      EQU  1(4:4)
BMAX  EQU  B-1
UMAX  EQU  20
TABLE NOP GOOD(BMAX)
       ADD FLOAT(5:5)
       SUB FLOAT(5:5)
       MUL FLOAT(5:5)
       DIV FLOAT(5:5)
       HLT GOOD
       SRC GOOD
       MOVE MEMORY(BMAX)
       LDA FIELD(5:5)
...
       STZ FIELD(5:5)
       JBUS MEMORY(UMAX)
       IOC GOOD(UMAX)
       IN  MEMORY(UMAX)
       OUT MEMORY(UMAX)
       JRED MEMORY(UMAX)
       JLE MEMORY
       JANP MEMORY

```

```

...
JXNP MEMORY
ENNA GOOD

...
ENN X GOOD
CMPA FLOAT(5:5)
CMP1 FIELD(5:5)

...
CMPX FIELD(5:5)

BEGIN LDA INST
      CMPA VALID(3:3)
      JG BAD          Pole I > 6?
      LD1 INST(5:5)
      DEC1 64
      J1NN BAD         Pole C ≥ 64?
      CMPA TABLE+64,1(4:4)
      JG BAD          Pole F > największe F?
      LD1 TABLE+64,1(1:2) Skocz do odpowiedniego
      JMP 0,1          podprogramu.
FLOAT CMPA VALID(4:4)   F = 6 dozwolone dla
      JE GOOD         operacji arytmetycznych.
FIELD ENTA 0
      LDX INST(4:4)   Sprytnie sprawdzamy poprawność
      DIV =9=          specyfikacji wycinka.
      STX *+1(0:2)
      INCA 0
      CMPA =5=
      JG BAD
MEMORY LDX INST(3:3)
      JXNZ GOOD        Jeśli I = 0, to sprawdź,
      LDX INST(0:2)    czy adres rzeczywiście
      JXN BAD          określa miejsce w pamięci.
      CMPX =3999=
      JLE GOOD
      JMP BAD
VALID CMPX 3999,6(6)  ■

```

10. Dowcip polega na tym, że w każdym wierszu i kolumnie może być kilka maksimów lub minimów, z których każde jest potencjalnym punktem siodłowym.

Rozwiążanie 1: Przeglądamy wszystkie wiersze, tworząc listy kolumn, w których występuje minimum danego rzędu. Dla każdej kolumny na liście sprawdzamy, czy minimum rzędu jest maksimum kolumny. $rX \equiv$ aktualne minimum; $rI1$ przebiega tablicę, zmieniając się od 72 do zera, do momentu znalezienia punktu siodłowego; $rI2 \equiv$ numer kolumny odpowiadającej $rI1$; $rI3 \equiv$ rozmiar listy minimów. Zauważmy, że wszystkie warunki pętli są postaci: rejestr indeksowy ≤ 0 .

* ROZWIĄZANIE 1

A10	EQU	1008	Lokacja a_{10} .
LIST	EQU	1000	

START	ENT1 9*8	Zaczni od dolnego prawego rogu.
ROWMIN	ENT2 8	rI1 wskazuje na element w ósmej kolumnie.
2H	LDX A10,1	Kandydat na minimum w wierszu.
	ENT3 0	Lista jest pusta.
4H	INC3 1	
	ST2 LIST,3	Wstaw numer kolumny do listy.
1H	DEC1 1	Przesuń się o jeden w lewo.
	DEC2 1	
	J2Z COLMAX	Wiersz sprawdzony?
3H	CMPX A10,1	
	JL 1B	Czy rX jest wciąż minimum?
	JG 2B	Nowe minimum?
	JMP 4B	Zapamiętaj inne minimum.
COLMAX	LD2 LIST,3	Pobierz kolumnę z listy.
	INC2 9*8-8	
1H	CMPX A10,2	
	JL NO	Minimum wiersza < element kolumny?
	DEC2 8	
	J2P 1B	Kolumna sprawdzona?
YES	INC1 A10+8,2	Tak; rI1 \leftarrow adres siodła.
	HLT	
NO	DEC3 1	Czy lista pusta?
	J3P COLMAX	Nie; próbujemy dalej.
	J1P ROWMIN	Wszystkie wiersze?
	HLT	Tak; rI1 = 0, brak siodła. ■

Rozwiążanie 2: Wykorzystując odrobinę matematyki, można wyprowadzić inny algorytm.

Twierdzenie. Niech $R(i) = \min_j a_{ij}$, $C(j) = \max_i a_{ij}$. Element $a_{i_0 j_0}$ jest punktem siodłowym wtedy i tylko wtedy, gdy $R(i_0) = \max_i R(i) = C(j_0) = \min_j C(j)$.

Dowód. Jeśli $a_{i_0 j_0}$ jest punktem siodłowym, to dla dowolnego ustalonego i mamy $R(i_0) = C(j_0) \geq a_{ij_0} \geq R(i)$. Stąd $R(i_0) = \max_i R(i)$. Podobnie $C(j_0) = \min_j C(j)$. W drugą stronę mamy $R(i) \leq a_{ij} \leq C(j)$ dla dowolnych i oraz j . Zatem z $R(i_0) = C(j_0)$ wynika, że $a_{i_0 j_0}$ jest punktem siodłowym. ■

(Z dowodu wynika, że zawsze $\max_i R(i) \leq \min_j C(j)$. Zatem punkt siodłowy nie istnieje wtedy i tylko wtedy, gdy wszystkie R są mniejsze niż wszystkie C).

Zgodnie z twierdzeniem, wystarczy znaleźć najmniejsze maksimum kolumny, a potem poszukać równego minimum wiersza. W programie podczas fazy 1 mamy: rI1 \equiv numer kolumny; rI2 przebiega macierz. Podczas fazy 2 mamy: rI1 \equiv możliwa odpowiedź; rI2 przebiega macierz; rI3 \equiv numer z wiersza razy 8; rI4 \equiv numer kolumny.

* ROZWIĄZANIE 2		
CMAX	EQU 1000	
A10	EQU CMAX+8	
FAZA1	ENT1 8	Zaczynamy od kolumny nr 8.
3H	ENT2 9*8-8,1	
	JMP 2F	
1H	CMPX A10,2	Czy rX wciąż jest maksimum?
	JGE **2	

2H	LDX A10,2	Nowe maksimum w kolumnie.
	DEC2 8	
	J2P 1B	
	STX CMAX+8,2	Zachowaj maksimum kolumny.
	J2Z 1F	Po raz pierwszy?
	CMPA CMAX+8,2	Czy rA jest wciąż min max?
	JLE **2	
1H	LDA CMAX+8,2	
	DEC1 1	Przesuń się w lewo.
	J1P 3B	
	FAZA2 ENT3 9*8-8	W tym miejscu rA = $\min_j C(j)$.
3H	ENT2 8,3	Przygotuj się do przeszukania wiersza.
	ENT4 8	
1H	CMPA A10,2	Czy $\min_j C(j) > a[i, j]$?
	JG NO	W tym wierszu nie ma siodła.
	JL 2F	
	CMPA CMAX,4	Czy $a[i, j] = C(j)$?
	JNE 2F	
	ENT1 A10,2	Zapamiętaj potencjalny punkt siodłowy.
2H	DEC4 1	Przesuń się w lewo.
	DEC2 1	
	J4P 1B	
	HLT	Znaleziono punkt siodłowy.
NO	DEC3 8	
	J3P 3B	
	ENT1 0	Sprawdź następny wiersz.
	HLT	rI1 = 0; nie ma siodła. ■

Pozostawiamy Czytelnikowi znalezienie jeszcze lepszego rozwiązania, które w fazie 1 zapamiętuje te wiersze, które są kandydatami do przeszukiwania w fazie 2. Nie ma potrzeby przeszukiwania wszystkich wierszy, wystarczy przeszukać tylko każdy wiersz i_0 , taki że $C(j_0) = \min_j C(j)$ pociąga $a_{i_0, j_0} = C(j_0)$. Zazwyczaj taki wiersz jest jeden.

Dla kilku testowych zestawów danych, o elementach wybranych losowo spośród $\{0, 1, 2, 3, 4\}$, rozwiązanie 1 wymagało czasu około $730u$, a rozwiązanie 2 około $530u$. Dla macierzy zerowej rozwiązanie 1 znajduje punkt siodłowy w czasie $137u$, rozwiązanie 2 – w czasie $524u$.

Jeżeli macierz $m \times n$ ma elementy parami różne, można rozwiązać problem, zaglądając jedynie do $O(m + n)$ elementów i wykonując $O(m \log n)$ operacji dodatkowych. Zobacz Bienstock, Chung, Fredman, Schäffer, Shor, Suri, *AMM* **98** (1991), 418–419.

11. Weźmy macierz $m \times n$. (a) Na mocy twierdzenia podanego w rozwiążaniu ćwiczenia 10 wszystkie punkty siodłowe macierzy mają tę samą wartość, zatem (przy założeniu, że elementy są parami różne) istnieje co najwyżej jeden punkt siodłowy. Na mocy symetrii poszukiwane prawdopodobieństwo jest równe mn razy prawdopodobieństwo zdarzenia, że a_{11} jest punktem siodłowym, a to z kolei wynosi $1/(mn)!$ razy liczba permutacji, dla których $a_{12} > a_{11}, \dots, a_{1n} > a_{11}, a_{11} > a_{21}, \dots, a_{11} > a_{m1}$. To jest $1/(m+n-1)!$ razy liczba permutacji $m+n-1$ elementów, dla których pierwszy jest większy od $(m-1)$ kolejnych i mniejszy od pozostałych $(n-1)$, czyli $(m-1)!(n-1)!$. Odpowiedzią jest zatem

$$mn(m-1)!(n-1)!/(m+n-1)! = (m+n)/\binom{m+n}{n}.$$

W naszym przypadku jest to $17/{8^{17}}$, czyli zaledwie jedna szansa na 1430. (b) Przy drugim założeniu musimy skorzystać z zupełnie innej metody, bo punktów siodłowych może być wiele. W istocie albo cały wiersz, albo cała kolumna musi się składać z punktów siodłowych. Szukane prawdopodobieństwo wynosi tyle, co prawdopodobieństwo zdarzenia, że istnieje punkt siodłowy o wartości zero plus prawdopodobieństwo zdarzenia, że istnieje punkt siodłowy o wartości jeden. Pierwsze z nich jest równe prawdopodobieństwu zdarzenia, że przynajmniej jedna kolumna zawiera same zera; drugie – prawdopodobieństwu zdarzenia, że przynajmniej jeden wiersz zawiera same jedynki. Odpowiedzią jest $(1 - (1 - 2^{-m})^n) + (1 - (1 - 2^{-n})^m)$, w naszym przypadku $924744796234036231/18446744073709551616$, tj. około 1 na 19.9. Odpowiedź przybliżona: $n2^{-m} + m2^{-n}$.

12. M. Hofri i P. Jacquet [Algorithmica **22** (1998), 516–528] przeanalizowali przypadek, gdy pola macierzy $m \times n$ są różne i losowe. Czas wykonania obu programów na komputerze MIX wynosi wtedy odpowiednio $(6mn + 5mH_n + 8m + 6 + 5(m+1)/(n-1))u + O((m+n)^2/(m+n))$ oraz $(5mn + 2nH_m + 8m + 7n + 9H_n - 3)u + O(1/n) + O((\log n)^2/m)$ przy $m \rightarrow \infty$ i $n \rightarrow \infty$, jeżeli $(\log n)/m \rightarrow 0$.

13. * PROBLEM ANALIZY SZYFRU (TAJNE SPECJALNEGO ZNACZENIA)

TAPE	EQU 20	Numer jednostki wejściowej.
TYPE	EQU 19	Numer jednostki wyjściowej.
SIZE	EQU 14	Rozmiar bloku wejściowego.
OSIZE	EQU 14	Rozmiar bloku wyjściowego.
TABLE	EQU 1000	Tablica liczników
	ORIG TABLE	(początkowo zawiera zera
	CON -1	poza pozycjami odpowiadającymi
	ORIG TABLE+46	spacji i gwiazdce).
	CON -1	
	ORIG 2000	
BUF1	ORIG *+SIZE	Pierwszy bufor.
	CON -1	„Wartownik” na końcu bufora.
	CON *+1	Adres drugiego bufora.
BUF2	ORIG *+SIZE	Drugi bufor.
	CON -1	„Wartownik”.
	CON BUF1	Adres pierwszego bufora.
BEGIN	IN BUF1(TAPE)	Wczytaj pierwszy blok.
	ENT6 BUF2	
1H	IN 0,6(TAPE)	Wczytaj kolejny blok.
	LD6 SIZE+1,6	Podczas wczytywania przygotuj
	ENT5 0,6	przetwarzanie poprzedniego bloku.
	JMP 4F	
2H	INCA 1	
	STA TABLE,1	Zmodyfikuj pozycję tablicy.
3H	SLAX 1	
	STA *+1(2:2)	rII1 \leftarrow następny znak.
	ENT1 0	
	LDA TABLE,1	
	JANN 2B	Zwykły znak?
	J1NZ 3F	Gwiazdka?
	JXP 3B	Pomiń spację.
	INC5 1	

pętla
główna,
powinna
działać jak
najszybciej

4H	LDX 0,5	rX ← pięć znaków.
	JXNN 3B	Skok, jeśli to nie wartownik.
	JMP 1B	Blok załutowiony.
3H	ENT1 1	Powoli kończymy: rI1 ← „A”.
2H	LDA TABLE,1	
	JANP 1F	Pomiń zerowe odpowiedzi.
	CHAR	Zamień na dziesiętne.
	JBUS *(TYPE)	Czekaj na gotowość.
	ST1 CHAR(1:1)	
	STA CHAR(4:5)	
	STX FREQ	
	OUT ANS(TYPE)	Drukuj odpowiedź.
1H	CMP1 =63=	
	INC1 1	Zliczone kody
	JL 2B	aż do 63.
	HLT	
ANS	ALF	Bufor wyjściowy.
	ALF	
CHAR	ALF C NN	
FREQ	ALF NNNNN	
	ORIG ANS+OSIZE	Reszta bufora jest pusta.
	END BEGIN	Tu zostanie wygenerowana stała znakowa =63=. ■

Dla tego problemu buforowanie *wyjścia* nie jest zalecane, gdyż można na nim zaoszczędzić najwyżej 7u na wierszu. Więcej informacji na temat częstotliwości występowania liter w języku angielskim można znaleźć w Charles P. Bourne, Donald F. Ford, „A study of the statistics of letters in English words”, *Information and Control* 4 (1961), 48–67.

14. W poniższym rozwiązaniu, które w części zawdzięczamy J. Petolino, jest wykorzystanych wiele sztuczek zmniejszających czas wykonania. A może Czytelnik wycisnie jeszcze parę mikrosekund?

* DATA WIELKANOCY		
EASTER	STJ EASTX	
	STX Y	
	ENTA 0	<u>E1.</u>
	DIV =19=	
	STX GMINUS1(0:2)	
	LDA Y	<u>E2.</u>
	MUL =1//100+1=	(zobacz
	INCA 61	poniżej)
	STA CPLUS60(1:2)	
	MUL =3//4+1=	
	STA XPLUS57(1:2)	
CPLUS60	ENTA *	
	MUL =8//25+1=	rA ← Z + 24.
GMINUS1	ENT2 *	<u>E5.</u>
	ENT1 1,2	rI1 ← G.
	INC2 1,1	
	INC2 0,2	

```

INC2 0,1
INC2 0,2
INC2 773,1      rI2 ← 11G + 773.
XPLUS57 INCA -*,2      rA ← 11G + Z - X + 20 + 24 · 30 ( $\geq 0$ ).
SRAX 5
DIV =30=        rX ← E.
DECX 24
JXN 4F
DECX 1
JXP 2F
JXN 3F
DEC1 11
J1NP 2F
3H INCX 1
2H DECX 29      E6.
4H STX 20MINUSN(0:2)
LDA Y           E4.
MUL =1//4+1=
ADD Y
SUB XPLUS57(1:2) rA ← D - 47.
20MINUSN ENN1 *
INCA 67,1       E7.
SRAX 5         rX ← D + N.
DIV =7=
SLAX 5
DECA -4,1      rA ← 31 - N.
JAN 1F          E8.
DECA 31
CHAR
LDA MARCH
JMP 2F
1H CHAR
LDA APRIL
2H JBUS *(18)
STA MONTH
STX DAY(1:2)
LDA Y
CHAR
STX YEAR
OUT ANS(18)     Drukuj.
EASTX JMP *
MARCH ALF MARCH
APRIL ALF APRIL
ANS ALF
DAY ALF DD
MONTH ALF MMMMM
ALF ,
YEAR ALF YYYY
ORIG *+20

```

```

BEGIN    ENTX 1950      Program
        ENT6 1950-2000   główny,
        JMP  EASTER       korzysta
        INC6 1            z powyższego
        ENTX 2000,6       podprogramu.
        J6NP EASTER+1
        HLT
END    BEGIN

```

|

Ścisłe uzasadnienie, że w kilku miejscach dzielenie można zamienić na mnożenie, opiera się na fakcie, że liczba w rA nie jest zbyt duża. Program działa dla wszystkich rozmiarów bajtu.

[By obliczyć datę Wielkanocy dla lat przed 1582 rokiem, sięgnij do CACM 5 (1962), 209–210. Pierwszym systematycznym algorytmem obliczania daty Wielkanocy był *canon paschalis* autorstwa Wiktora z Akwitanii (457 rok). Wiele wskazuje na to, że obliczanie daty Wielkanocy było jedynym nietrywialnym zastosowaniem arytmetyki w średniowiecznej Europie. Zobacz komentarz w *Puzzles and Paradoxes*, T. H. O’Beirne (London: Oxford University Press, 1965), rozdział 10. Różne algorytmy związane z datami omówiono w *Calendrical Calculations*, N. Dershowitz, E. M. Reingold (Cambridge Univ. Press, 1997)].

15. Pierwszym takim rokiem jest 10317, chociaż o mały włos udaje się uniknąć błędu dla lat $10108 + 19k$ dla $0 \leq k \leq 10$.

T. H. O’Beirne zauważył, że data Wielkanocy powtarza się w cyklu o długości 5,700,000 lat. Obliczenia przeprowadzone przez Roberta Hilla dowodzą, że najczęściej występującą datą Wielkanocy jest 19 kwietnia (220400 razy na okres), podczas gdy najwcześniejszą i najrzadziej pojawiającą się datą jest 22 marca (27550 razy); datą najpóźniejszą i drugą co do rzadkości występowania jest 25 kwietnia (42000 razy). Hill znalazł eleganckie wyjaśnienie ciekawego faktu, że liczba wystąpień dowolnej daty w cyklu jest wielokrotnością 25.

16. Pracujemy w arytmetyce stałopozycyjnej, $R_n = 10^n r_n$. $R_n(1/m) = R$ wtedy i tylko wtedy, gdy $10^n/(R + \frac{1}{2}) < m \leq 10^n/(R - \frac{1}{2})$. Zatem $m_h = \lfloor 2 \cdot 10^n / (2R - 1) \rfloor$.

```

* SUMA SZEREGU HARMONICZNEGO
BUF    ORIG **+24
START  ENT2 0
        ENT1 3      5 - n
        ENTA 20
OUTER MUL   =10=
        STX   CONST   2 · 10n
        DIV   =2=
        ENTX 2
        JMP   1F
INNER STA   R
        ADD   R
        DECA 1
        STA   TEMP   2R - 1
        LDX   CONST
        ENTA 0
        DIV   TEMP
        INCA 1

```

	STA TEMP	$m_h + 1$
	SUB M	
	MUL R	
	SLAX 5	
	ADD S	
	LDX TEMP	
1H	STA S	Suma częściowa
	STX M	$m = m_e$
	LDA M	
	ADD M	
	STA TEMP	
	LDA CONST	
	ADD M	Oblicz $R = R_n(1/m) =$
	SRAK 5	$\lfloor (2 \cdot 10^n + m)/(2m) \rfloor.$
	DIV TEMP	
	JAP INNER	$R > 0?$
	LDA S	$10^n S_n$
	CHAR	
	SLAX 0,1	Staranne formatowanie.
	SLA 1	
	INCA 41	Kropka dziesiętna.
	STA BUF,2	
	STX BUF+1,2	
	INC2 3	
	DEC1 1	
	LDA CONST	
	J1NN OUTER	
	OUT BUF(18)	
	HLT	
	END START	■

Dane wyjściowe mają postać

0006.16	0008.449	0010.7509	0013.05363
---------	----------	-----------	------------

i dostajemy je w czasie $65595u$ plus czas wyprowadzenia danych. (Byłoby szybciej, gdybyśmy obliczali $R_n(1/m)$ wprost dla $m < 10^{n/2}\sqrt{2}$, a potem stosowali pokazaną procedurę).

17. Niech $N = \lfloor 2 \cdot 10^n / (2m + 1) \rfloor$. Wówczas suma wynosi $S_n = H_N + O(N/10^n) + \sum_{k=1}^m (\lfloor 2 \cdot 10^n / (2k - 1) \rfloor - \lfloor 2 \cdot 10^n / (2k + 1) \rfloor)k/10^n = H_N + O(m^{-1}) + O(m/10^n) - 1 + 2H_{2m} - H_m = n \ln 10 + 2\gamma - 1 + 2 \ln 2 + O(10^{-n/2})$ przy sumowaniu przez części i podstawieniu $m \approx 10^{n/2}$.

Tak się składa, że następnymi wartościami są $S_6 = 15.356262$, $S_7 = 17.6588276$, $S_8 = 19.96140690$, $S_9 = 22.263991779$ i $S_{10} = 24.5665766353$; nasze przybliżenie S_{10} to 24.566576621, czyli bliżej niż przewidywaliśmy.

18. FAREY STJ 9F Zakładamy, że rII zawiera n , gdzie $n > 1$.

STZ X	$x_0 \leftarrow 0$
ENTX 1	
STX Y	$y_0 \leftarrow 1$
STX X+1	$x_1 \leftarrow 1$

```

ST1 Y+1       $y_1 \leftarrow n.$ 
ENT2 0         $k \leftarrow 0.$ 
1H   LDX Y,2
      INCX 0,1
      ENTA 0
      DIV Y+1,2
      STA TEMP   $\lfloor (y_k + n)/y_{k+1} \rfloor$ 
      MUL Y+1,2
      SLAX 5
      SUB Y,2
      STA Y+2,2   $y_{k+2}$ 
      LDA TEMP
      MUL X+1,2
      SLAX 5
      SUB X,2
      STA X+2,2   $x_{k+2}$ 
      CMPA Y+2,2  Sprawdź, czy  $x_{k+2} < y_{k+2}$ .
      INC2 1         $k \leftarrow k + 1.$ 
      JL  1B        Jeśli tak, kontynuuj.
9H   JMP *       Wyjdź z podprogramu. ■

```

19. (a) Indukcja. (b) Niech $k \geq 0$ i $X = ax_{k+1} - x_k$, $Y = ay_{k+1} - y_k$, gdzie $a = \lfloor (y_k + n)/y_{k+1} \rfloor$. Na mocy (a) oraz z faktu, że $0 < Y \leq n$, mamy $X \perp Y$ i $X/Y > x_{k+1}/y_{k+1}$. Jeśli zatem $X/Y \neq x_{k+2}/y_{k+2}$, to z definicji mamy $X/Y > x_{k+2}/y_{k+2}$. Stąd wynika

$$\begin{aligned}
\frac{1}{Yy_{k+1}} &= \frac{Xy_{k+1} - Yx_{k+1}}{Yy_{k+1}} = \frac{X}{Y} - \frac{x_{k+1}}{y_{k+1}} \\
&= \left(\frac{X}{Y} - \frac{x_{k+2}}{y_{k+2}} \right) + \left(\frac{x_{k+2}}{y_{k+2}} - \frac{x_{k+1}}{y_{k+1}} \right) \\
&\geq \frac{1}{Yy_{k+2}} + \frac{1}{y_{k+1}y_{k+2}} = \frac{y_{k+1} + Y}{Yy_{k+1}y_{k+2}} > \frac{n}{Yy_{k+1}y_{k+2}} \geq \frac{1}{Yy_{k+1}}.
\end{aligned}$$

Uwaga historyczna: C. Haros podał (bardziej skomplikowaną) regułę otrzymywania takich ciągów w *J. de l'École Polytechnique* 4, 11 (1802), 364–368. Jego metoda była poprawna, ale dowód nie. Kilka lat później geolog John Farey niezależnie postawił hipotezę, że x_k/y_k jest zawsze równe $(x_{k-1} + x_{k+1})/(y_{k-1} + y_{k+1})$ [*Philos. Magazine and Journal* 47 (1816), 385–386]. Niedługo potem dowód tego faktu został zaprezentowany przez A. Cauchy'ego [*Bull. Société Philomathique de Paris* (3) 3 (1816), 133–135], który do tego ciągu przyleił etykietkę z nazwiskiem Fareya. Więcej ciekawych informacji można znaleźć w: G. H. Hardy, E. M. Wright, *An Introduction to the Theory of Numbers*, rozdział 3.

20. * SYGNALIZACJA ŚWIETLNA

BSIZE EQU 1(4:4)	Rozmiar bajtu.
2BSIZE EQU 2(4:4)	Podwojony rozmiar bajtu.
DELAY STJ 1F	Jeśli rA zawiera n ,
DECA 6	to ten podprogram
DECA 2	czeka dokładnie $\max(n, 7)u$,
JAP *-1	nie licząc skoku
JAN *+2	do podprogramu.
NOP	

1H	JMP *		
FLASH	STJ 2F	4	Podprogram „migocze” odpowiednim
	ENT2 8	5	sygnałem DON'T WALK.
1H	LDA =49991=	7	
	JMP DELAY	8	
	DECX 0,1	9	Wyłącz światło.
	LDA =49996=	2	
	JMP DELAY	3	
	INCX 0,1	4	„DON'T WALK”.
	DEC2 1	1	
	J2Z 1F	2	Powtórz osiem razy.
	LDA =49993=	4	
	JMP 1B	5	Zsynchronizuj.
1H	LDA =399992=	4	Po wyjściu żółte światło przez 2u.
	JMP DELAY	5	
2H	JMP *	6	
WAIT	JNOV *	5	Zielone dla Del Mar.
TRIP	INCX BSIZE	6	DON'T WALK na Del Mar.
	ENT1 2BSIZE	1	
	JMP FLASH	2	Migaj na Del Mar.
	LDX BAMBER	8	Żółte na bulwarze.
	LDA =799995=	2	
	JMP DELAY	3	Czekaj 8 sekund.
	LDX AGREEN	5	Zielone na alei.
	LDA =799996=	2	
	JMP DELAY	3	Czekaj 8 sekund.
	INCX 1	4	DON'T WALK na Berkeley.
	ENT1 2	1	
	JMP FLASH	2	Migaj na Berkeley.
	LDX AAMBER	8	Żółte na alei.
	JOV *+1	1	Anuluj zbędny cykl.
	LDA =499994=	3	
	JMP DELAY	4	Czekaj 5 sekund.
BEGIN	LDX BGREEN	6	Zielone na bulwarze.
	LDA =1799994=	2	
	JMP DELAY	3	Czekaj przynajmniej
	JMP WAIT	4	18 sekund.
AGREEN	ALF CABA		Zielone dla alei.
AAMBER	ALF CBBB		Żółte dla alei.
BGREEN	ALF ACAB		Zielone dla bulwaru.
BAMBER	ALF BCBB		Żółte dla bulwaru.
END	BEGIN		■

22. * PROBLEM JÓZefa FLAWIUSZA

N	EQU 24		
M	EQU 11		
X	ORIG *+N		
OH	ENT1 N-1	1	Numer następnej
	STZ X+N-1	1	osoby w ciągu.
	ST1 X-1,1	<i>N</i> - 1	

	DEC1 1	$N - 1$	
	J1P *-2	$N - 1$	
	ENTA 1	1	(Mamy rI1 = 0)
1H	ENT2 M-2	$N - 1$	(Zakładamy $M > 2$)
	LD1 X,1	$(M - 2)(N - 1)$	Licz po okręgu.
	DEC2 1	$(M - 2)(N - 1)$	
	J2P *-2	$(M - 2)(N - 1)$	
	LD2 X,1	$N - 1$	rI1 ≡ szczęściaż
	LD3 X,2	$N - 1$	rI2 ≡ pechowiec
	CHAR	$N - 1$	rI3 ≡ następny
	STX X,2(4:5)	$N - 1$	Przechowaj numer egzekucji.
	NUM	$N - 1$	
	INCA 1	$N - 1$	
	ST3 X,1	$N - 1$	Wybierz osobę.
	ENT1 0,3	$N - 1$	
	CMPA =N=	$N - 1$	
	JL 1B	$N - 1$	
	CHAR	1	Została jedna osoba;
	STX X,1(4:5)	1	jego też nie ominie.
	OUT X(18)	1	Drukuj odpowiedź.
	HLT	1	
	END OB		

■

Ostatnia osoba stoi na pozycji 15. Całkowity czas pracy przed wyprowadzeniem danych wynosi $(4(N - 1)(M + 7.5) + 16)u$. Można wprowadzić wiele usprawnień, na przykład (D. Ingalls) operować fragmentami kodu „DEC2 1; J2P NEXT; JMP OUT”, gdzie OUT modyfikuje pole NEXT, w wyniku czego „kasujemy” ten fragment. Asymptotycznie szybszą metodę pokazujemy w ćwiczeniu 5.1.1–5.

1.3.3

1. $(1\ 2\ 4)(3\ 6\ 5)$.
2. $a \leftrightarrow c, c \leftrightarrow f; b \leftrightarrow d$. Widać, że tę metodę można uogólnić na dowolną permutację.
3. $\begin{pmatrix} a & b & c & d & e & f \\ d & b & f & c & a & e \end{pmatrix}$.
4. $(adcf\ e)$.
5. 12. (Zobacz ćwiczenie 20).
6. Całkowity czas wykonania zwiększa się o $4u$ na każde białe słowo, dla którego poprzedzającym słowem niebiałym jest „(”, plus $5u$ na każde białe słowo, dla którego poprzedzającym słowem niebiałym jest nazwa elementu permutacji. Białe słowa na początku danych oraz między cyklami nie zmieniają wyniku przeprowadzonej analizy.
7. $X = 2, Y = 29, M = 5, N = 7, U = 3, V = 1$. Czas całkowity, zgodnie z (18), wynosi $2161u$.
8. Tak. Należy przechowywać odwrotność permutacji, by spełniony był warunek „ x_i przechodzi na x_j wtedy i tylko wtedy, gdy $T[j] = i$ ”. (Odpowiedź należy wówczas budować od prawej do lewej, korzystając z tablicy T).
9. Nie. Na przykład, jeśli na wejściu podamy (6), wtedy program A wygeneruje „(ADG)(CEB)”, a program B wygeneruje „(CEB)(DGA)”. Odpowiedzi są równoważne, ale

nie identyczne, co jest spowodowane niejednoznacznością notacji cyklowej. Program A za pierwszy element w cyklu bierze pierwszy z lewej, a program B – element o nazwie, którą napotka jako ostatnią przy przechodzeniu w lewo.

10. (1) Z prawa Kirchhoffa mamy $A = 1 + C - D$; $B = A + J + P - 1$; $C = B - (P - L)$; $E = D - L$; $G = E$; $Q = Z$; $W = S$. (2) Interpretacje: B = liczba słów na wejściu $= 16X - 1$; C = liczba słów niebiałych $= Y$; $D = C - M$; $E = D - M$; F = liczba porównań przy przeszukiwaniu tablicy nazw; $H = N$; $K = M$; $Q = N$; $R = U$; $S = R - V$; $T = N - V$, ponieważ wszystkie pozostałe nazwy są oznaczane. (3) Reasumując: $(4F + 16Y + 80X + 21N - 19M + 9U - 16V)u$, czyli nieco lepiej niż dla programu A, ponieważ F jest mniejsze niż $16NX$. W podanym przypadku czas wynosi $983u$, ponieważ $F = 74$.

11. Wystarczy wyznaczyć „odbicie lustrzane”. Na przykład permutacją odwrotną do $(a\ c\ f)(b\ d)$ jest $(d\ b)(f\ c\ a)$.

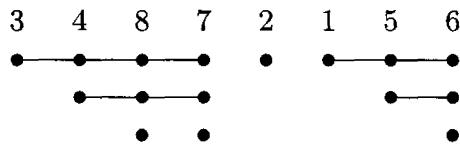
12. (a) Wartość z komórki $L + mn - 1$ nie jest przemieszczana, więc pomijamy ją w dalszych rozważaniach. W pozostałych przypadkach dla $x = n(i - 1) + (j - 1) < mn - 1$, wartość $L + x$ wędruje do komórki $L + mx \bmod N = L + (mn(i - 1) + m(j - 1)) \bmod N = L + m(j - 1) + (i - 1)$, ponieważ $mn \equiv 1 \pmod{N}$ oraz $0 \leq m(j - 1) + (i - 1) < N$. (b) Jeżeli wolno nam wykorzystać jeden bit z każdej komórki pamięci (na przykład znak), to możemy „oznaczać” przenoszone elementy za pomocą algorytmu podobnego do algorytmu I. [Zobacz M. F. Berman, *JACM* **5** (1958), 383–384]. Jeżeli nie ma miejsca na bit-znacznik, trzeba przechowywać znaczniki w pomocniczej tablicy. Można także skorzystać z reprezentacji listowej cykli niejednoelementowych: dla każdego dzielnika d liczby N możemy oddziennie transponować elementy, które są wielokrotnościami d , ponieważ m jest pierwsze względem N . Długość cyklu zawierającego x , gdy $\gcd(x, N) = d$, jest najmniejszą liczbą całkowitą $r > 0$, taką że $m^r \equiv 1 \pmod{N/d}$. Dla każdego d chcemy znaleźć $\varphi(N/d)/r$ reprezentantów, po jednym z każdego z tych cykli. Możemy zastanawiać się nad zastosowaniem do tego pewnych metod teorioliczbowych, ale są one zbyt skomplikowane, by zastosować je w praktyce. Efektywny, ale skomplikowany algorytm można otrzymać, korzystając z odrobiny teorii liczb oraz małej pomocniczej tablicy bitów-znaczników. [Zobacz N. Brenner, *CACM* **16** (1973), 692–694]. Istnieje w końcu metoda analogiczna do algorytmu J; jest wolniejsza, ale nie wymaga dodatkowej pamięci i wykonuje dowolną permutację *in situ**. [Zobacz P. F. Windley, *Comp. J.* **2** (1959), 47–48; D. E. Knuth, *Proc. IFIP Congress* (1971), **1**, 19–27; E. G. Cate, D. W. Twigg, *ACM Trans. Math. Software* **3** (1977), 104–110; F. E. Fich, J. I. Munro, P. V. Poblete, *SICOMP* **24** (1995), 266–278].

13. Należy pokazać przez indukcję, że na początku kroku J2 zachodzi $X[i] = +j$ wtedy i tylko wtedy, gdy $j > m$ i π przeprowadza j na i . $X[i] = -j$ wtedy i tylko wtedy, gdy π^{k+1} przeprowadza i na j , gdzie k jest najmniejszą nieujemną liczbą całkowitą, taką że π^k przeprowadza i na liczbę $\leq m$.

14. Zapiszmy *odwrotność* danej permutacji w kanonicznym zapisie cyklowym i opuśćmy nawiasy. Dla każdego elementu wyznaczamy podciąg elementów permutacji, posuwając się w prawo do napotkania elementu mniejszego lub do końca napisu. Liczba A jest sumą długości takich podciągów (braną po wszystkich elementach permutacji). Weźmy na przykład permutację $(1\ 6\ 5)(3\ 7\ 8\ 4)$. Jej permutacją odwrotną w kanonicz-

* *in situ* – łac. *w miejscu* (przyp. tłum.).

nym zapisie cyklowym jest $(3\ 4\ 8\ 7)(2)(1\ 5\ 6)$; z pomocniczego rysunku



odczytujemy wartość A jako liczbę „kropek”: 16. Liczba kropek poniżej k -tego elementu to liczba minimów prawostronnych pośród pierwszych k elementów (w powyższym przykładzie pod 7 są 3 kropki, ponieważ w ciągu 3487 są trzy minima prawostronne). Średnia wartość wynosi zatem $H_1 + H_2 + \dots + H_n = (n+1)H_n - n$.

15. Jeżeli pierwszym znakiem reprezentacji liniowej jest 1, to ostatnim znakiem reprezentacji kanonicznej jest 1. Jeżeli pierwszym znakiem reprezentacji liniowej jest $m > 1$, to w reprezentacji kanonicznej występuje „ $\dots 1m \dots$ ”. Jedynymi rozwiązaniami są permutacje jednego tylko elementu. (No cóż, istnieje nawet permutacja *zero* elementów).

16. 1324, 4231, 3214, 4213, 2143, 3412, 2413, 1243, 3421, 1324, ...

17. (a) Prawdopodobieństwo zdarzenia, że cykl jest cyklem m -elementowym wynosi $n!/m$ dzielone przez $n! H_n$, czyli $p_m = 1/(mH_n)$. Średnia długość wynosi $p_1 + 2p_2 + 3p_3 + \dots = \sum_{m=1}^n (m/mH_n) = n/H_n$. (b) Z uwagi na fakt, że całkowita liczba cykli m -elementowych wynosi $n!/m$, całkowita liczba wystąpień elementów w cyklach m -elementowych wynosi $n!$. Każdy element występuje równie często co pozostałe, zatem k występuje w cyklach m -elementowych $n!/n$ razy. W tym przypadku mamy zatem $p_m = 1/n$ dla wszystkich k i m ; średnia wynosi $\sum_{m=1}^n m/n = (n+1)/2$.

18. Zobacz ćwiczenie 22(e).

19. $|P_{n0} - n!/e| = n!/(n+1)! - n!/(n+2)! + \dots$, szereg naprzemienny o malejących wartościach, mniejszy od $n!/(n+1)! \leq \frac{1}{2}$.

20. Mamy $\alpha_1 + \alpha_2 + \dots$ cykli, które można dowolnie permutować, a dodatkowo każdy cykl można zapisać na m sposobów. Odpowiedź:

$$(\alpha_1 + \alpha_2 + \dots)! 1^{\alpha_1} 2^{\alpha_2} 3^{\alpha_3} \dots$$

21. $1/(\alpha_1! 1^{\alpha_1} \alpha_2! 2^{\alpha_2} \dots)$ gdy $n = \alpha_1 + 2\alpha_2 + \dots$; zero w przeciwnym przypadku.

Dowód. Zapiszmy obok siebie α_1 cykli 1-elementowych, α_2 cykli 2-elementowych itd., nie wpisując konkretnych wartości. Na przykład dla $\alpha_1 = 1, \alpha_2 = 2, \alpha_3 = \alpha_4 = \dots = 0$, piszemy „(-) (--) (--)”. Wypełniamy teraz puste pozycje na wszystkie $n!$ możliwych sposobów. Każdą permutację zadanej postaci otrzymujemy dokładnie $\alpha_1! 1^{\alpha_1} \alpha_2! 2^{\alpha_2} \dots$ razy.

22. (a) Jeżeli $k_1 + 2k_2 + \dots = n$, to prawdopodobieństwo w (ii) wynosi $\prod_{j \geq 0} f(w, j, k_j)$, co z założenia równa się $(1-w)w^n/k_1! 1^{k_1} k_2! 2^{k_2} \dots$. Zatem

$$\frac{f(w, m, k_m + 1)}{f(w, m, k_m)} = \left(\prod_{j \geq 0} f(w, j, k_j) \right)^{-1} \prod_{j \geq 0} f(w, j, k_j + \delta_{jm}) = \frac{w^m}{m(k_m + 1)}.$$

Stąd przez indukcję mamy

$$f(w, m, k) = \frac{1}{k!} \left(\frac{w^m}{m} \right)^k f(w, m, 0).$$

Warunek (i) pociąga za sobą

$$f(w, m, k) = \frac{1}{k!} \left(\frac{w^m}{m} \right)^k e^{-w^m/m}.$$

[Innymi słowy: wybieramy wartości α_m zgodnie z rozkładem Poissona; zobacz ćwiczenie 1.2.10–15].

$$(b) \sum_{\substack{k_1+2k_2+\dots=n \\ k_1, k_2, \dots \geq 0}} \left(\prod_{j \geq 0} f(w, j, k_j) \right) = (1-w)w^n \sum_{\substack{k_1+2k_2+\dots=n \\ k_1, k_2, \dots \geq 0}} P(n; k_1, k_2, \dots) \\ = (1-w)w^n.$$

Prawdopodobieństwo, że $\alpha_1 + 2\alpha_2 + \dots \leq n$, wynosi zatem $(1-w)(1+w+\dots+w^n) = 1-w^{n+1}$.

(c) Średnia wartość ϕ równa się

$$\sum_{n \geq 0} \left(\sum_{k_1+2k_2+\dots=n} \phi(k_1, k_2, \dots) \Pr(\alpha_1 = k_1, \alpha_2 = k_2, \dots) \right) \\ = (1-w) \sum_{n \geq 0} w^n \left(\sum_{k_1+2k_2+\dots=n} \phi(k_1, k_2, \dots) / k_1! 1^{k_1} k_2! 2^{k_2} \dots \right).$$

(d) Niech $\phi(\alpha_1, \alpha_2, \dots) = \alpha_2 + \alpha_4 + \alpha_6 + \dots$. Średnia wartość kombinacji liniowej ϕ jest sumą średnich wartości $\alpha_2, \alpha_4, \alpha_6, \dots$. Średnia wartość α_m wynosi

$$\sum_{k \geq 0} kf(w, m, k) = \sum_{k \geq 1} \frac{1}{(k-1)!} \left(\frac{w^m}{m} \right)^k e^{-w^m/m} = \frac{w^m}{m}.$$

Stąd średnia wartość ϕ równa się

$$\frac{w^2}{2} + \frac{w^4}{4} + \frac{w^6}{6} + \dots = \frac{1-w}{2} (H_1 w^2 + H_1 w^3 + H_2 w^4 + H_2 w^5 + H_3 w^6 + \dots).$$

Poszukiwaną odpowiedzią jest $\frac{1}{2} H_{\lfloor n/2 \rfloor}$.

(e) Podstawmy $\phi(\alpha_1, \alpha_2, \dots) = z^{\alpha_m}$ i zauważmy, że średnia wartość ϕ jest równa

$$\sum_{k \geq 0} f(w, m, k) z^k = \sum_{k \geq 0} \frac{1}{k!} \left(\frac{w^m z}{m} \right)^k e^{-w^m/m} = e^{w^m(z-1)/m} = \sum_{j \geq 0} \frac{w^{mj}}{j!} \left(\frac{z-1}{m} \right)^j \\ = (1-w) \sum_{n \geq 0} w^n \left(\sum_{0 \leq j \leq n/m} \frac{1}{j!} \left(\frac{z-1}{m} \right)^j \right) \\ = (1-w) \sum_{n \geq 0} w^n G_{nm}(z).$$

Zatem

$$G_{nm}(z) = \sum_{0 \leq j \leq n/m} \frac{1}{j!} \left(\frac{z-1}{m} \right)^j; \quad p_{nkm} = \frac{1}{m^k k!} \sum_{0 \leq j \leq n/m-k} \frac{(-1/m)^j}{j!};$$

otrzymujemy ($\min 0$, ave $1/m$, $\max \lfloor n/m \rfloor$, dev $\sqrt{1/m}$), dla $n \geq 2m$.

23. Wartość stałej λ wynosi $\int_0^\infty \exp(-t - E_1(t)) dt$, gdzie $E_1(x) = \int_x^\infty e^{-t} dt/t$. Zobacz Trans. Amer. Math. Soc. **121** (1966), 340–357, gdzie znajdują się dowody wielu innych wyników; w szczególności dowód faktu, że długość *najkrótszego* cyklu równa się w przybliżeniu $e^{-\gamma} \ln n$. Dalsze wyrazy rozwinięcia asymptotycznego l_n można znaleźć w pracy Xaviera Gourdon [w druku]; szereg rozpoczyna się od

$$\lambda n + \frac{1}{2}\lambda - \frac{1}{24}e^\gamma n^{-1} + (\frac{1}{48}e^\gamma - \frac{1}{8}(-1)^n)n^{-2} + (\frac{17}{3840}e^\gamma + \frac{1}{8}(-1)^n + \frac{1}{6}\omega^{1-n} + \frac{1}{6}\omega^{n-1})n^{-3},$$

gdzie $\omega = e^{2\pi i/3}$. William C. Mitchell obliczył λ z dużą dokładnością, otrzymując $\lambda = 0.62432 99885 43550 87099 29363 83100 83724 41796+$ [Math. Comp. **22** (1968), 411–415]; nie są znane żadne związki między λ i klasycznymi stałymi matematycznymi. Ta sama stała została jednak obliczona w innym kontekście przez Karla Dickmana w *Arkiv för Mat., Astron. och Fys.* **22A**, 10 (1930), 1–14; fakt ten pozostawał niezauważony przez wiele lat [Theor. Comp. Sci. **3** (1976), 373].

24. Zobacz D. E. Knuth, *Proc. IFIP Congress (1971)*, **1**, 19–27.

25. Jeden dowód, przez indukcję względem N , jest oparty na fakcie, że kiedy N -ty element należy do s zbiorów, to wnosi do sumy dokładnie

$$\binom{s}{0} - \binom{s}{1} + \binom{s}{2} - \cdots = (1 - 1)^s = \delta_{s0}.$$

Inny dowód, przez indukcję względem M , jest oparty na fakcie, że liczba elementów, które należą do S_M , ale nie do $S_1 \cup \dots \cup S_{M-1}$ wynosi

$$|S_M| - \sum_{1 \leq j < M} |S_j \cap S_M| + \sum_{1 \leq j < k < M} |S_j \cap S_k \cap S_M| - \cdots.$$

26. Niech $N_0 = N$ i niech

$$N_k = \sum_{1 \leq j_1 < \dots < j_k \leq M} |S_{j_1} \cap \dots \cap S_{j_k}|.$$

Wówczas poszukiwany wzór ma postać

$$N_r - \binom{r+1}{r} N_{r+1} + \binom{r+2}{r} N_{r+2} - \cdots.$$

Wzór można udowodnić z samej zasady włączania i wyłączania albo ze wzoru

$$\binom{r}{r} \binom{s}{r} - \binom{r+1}{r} \binom{s}{r+1} + \cdots = \binom{s}{r} \binom{s-r}{0} - \binom{s}{r} \binom{s-r}{1} + \cdots = \delta_{sr},$$

jak w ćwiczeniu 25.

27. Niech S_j będzie zbiorem wielokrotności m_j w podanym przedziale i niech $N = am_1 \dots m_t$. Wówczas $|S_j \cap S_k| = N/m_j m_k$ itd., zatem odpowiedzią jest

$$N - N \sum_{1 \leq j \leq t} \frac{1}{m_j} + N \sum_{1 \leq j < k \leq t} \frac{1}{m_j m_k} - \cdots = N \left(1 - \frac{1}{m_1}\right) \cdots \left(1 - \frac{1}{m_t}\right).$$

W ten sposób otrzymujemy również rozwiązanie ćwiczenia 1.2.4–30, gdy za m_1, \dots, m_t podstawimy liczby pierwsze dzielące N .

29. Pozostawiając osobę przy życiu, nadajemy jej nowy numer (rozpoczynając od $n+1$). Wtedy k -ta zabita osoba ma numer $2k$, a poprzedni numer osoby j (dla $j > n$) to $(2j) \bmod (2n+1)$.

31. Zobacz CMath, rozdział 3.3.

32. (a) W istocie dla parzystych k zachodzi $k-1 \leq \pi_k \leq k+2$, a dla nieparzystych k zachodzi $k-2 \leq \pi_k \leq k+1$. (b) Dobieramy wykładniki od lewej do prawej, biorąc $e_k = 1$ wtedy i tylko wtedy, gdy k i $k+1$ w permutacji otrzymanej do tej pory leżą w różnych cyklach. [Steven Alpern, *J. Combinatorial Theory* **B25** (1978), 62–73].

33. Dla $l=0$ bierzemy $(\alpha_{01}, \alpha_{02}; \beta_{01}, \beta_{02}) = (\pi, \rho; \epsilon, \epsilon)$ i $(\alpha_{11}, \alpha_{12}; \beta_{11}, \beta_{12}) = (\epsilon, \epsilon; \pi, \rho)$, gdzie $\pi = (1\ 4)(2\ 3)$, $\rho = (1\ 5)(2\ 4)$ i $\epsilon = ()$.

Przypuśćmy, że przeprowadziliśmy konstrukcję dla pewnego $l \geq 0$, gdzie $\alpha_{jk}^2 = \beta_{jk}^2 = ()$ dla $0 \leq j < m$ i $1 \leq k \leq n$. Wówczas permutacje

$$(A_{(jm+j')1}, \dots, A_{(jm+j')(4n)}; B_{(jm+j')1}, \dots, B_{(jm+j')(4n)}) = \\ (\sigma^- \alpha_{j1} \sigma, \dots, \sigma^- \alpha_{jn} \sigma, \tau^- \alpha_{j'1} \tau, \dots, \tau^- \alpha_{j'n} \tau, \\ \sigma^- \beta_{j1} \sigma, \dots, \sigma^- \beta_{jn} \sigma, \tau^- \beta_{j'1} \tau, \dots, \tau^- \beta_{j'n} \tau; \\ \sigma^- \beta_{j1} \sigma, \dots, \sigma^- \beta_{jn} \sigma, \tau^- \beta_{j'1} \tau, \dots, \tau^- \beta_{j'n} \tau, \\ \sigma^- \alpha_{j1} \sigma, \dots, \sigma^- \alpha_{jn} \sigma, \tau^- \alpha_{j'1} \tau, \dots, \tau^- \alpha_{j'n} \tau)$$

mają następującą własność:

$$A_{(im+i')1} B_{(jm+j')1} \dots A_{(im+i')(4n)} B_{(jm+j')(4n)} = \\ \sigma^-(12345) \sigma \tau^-(12345) \tau \sigma^-(54321) \sigma \tau^-(54321) \tau$$

gdy $i = j$ i $i' = j'$, a w przeciwnym przypadku iloczyn jest permutacją identyczną. Przyjmując $\sigma = (23)(45)$ i $\tau = (345)$, otrzymamy wymagany iloczyn (12345) , gdy $im + i' = jm + j'$.

Konstrukcja przypadku $l+1$ na podstawie przypadku l pochodzi od Davida A. Barringtona [J. Comp. Syst. Sci. 38 (1989), 150–164], który udowodnił ogólne twierdzenie, na mocy którego każda funkcja boolowska może zostać przedstawiona jako iloczyn permutacji zbioru $\{1, 2, 3, 4, 5\}$. Za pomocą podobnej konstrukcji możemy znaleźć ciągi permutacji $(\alpha_{j1}, \dots, \alpha_{jn}; \beta_{j1}, \dots, \beta_{jn})$, takie że na przykład

$$\alpha_{i1} \beta_{j1} \alpha_{i2} \beta_{j2} \dots \alpha_{in} \beta_{jn} = \begin{cases} (12345), & \text{gdy } i < j; \\ (), & \text{gdy } i \geq j; \end{cases}$$

dla $0 \leq i, j < m = 2^{2^l}$, gdy $n = 6^{l+1} - 4^{l+1}$.

34. Niech $N = m+n$. Jeśli $m \perp n$, to permutacja ma tylko jeden cykl, ponieważ każdy element można zapisać w postaci $am \bmod N$ dla pewnego całkowitego a . W przypadku ogólnym, jeśli $d = \gcd(m, n)$, to mamy dokładnie d cykli C_0, C_1, \dots, C_{d-1} , gdzie C_j zawiera elementy $\{j, j+d, \dots, j+N-d\}$ w pewnym porządku. By zrealizować permutację, dla każdego $0 \leq j < d$ wykonujemy (równolegle, jeśli wola) co następuje: Przypisujemy $t \leftarrow x_j$ i $k \leftarrow j$; następnie dopóki $(k+m) \bmod N \neq j$, dopóty przypisujemy $x_k \leftarrow x_{(k+m) \bmod N}$ i $k \leftarrow (k+m) \bmod N$; na koniec przypisujemy $x_k \leftarrow t$. W tym algorytmie warunek $(k+m) \bmod N \neq j$ będzie zachodził wtedy i tylko wtedy, gdy $(k+m) \bmod N \geq d$, zatem możemy sprawdzać ten z warunków, który daje się bardziej wydajnie zaprogramować. [W. Fletcher, R. Silver, CACM 9 (1966), 326].

35. Niech $M = l+m+n$ i $N = l+2m+n$. Cykle szukanej permutacji otrzymujemy z cykli permutacji zbioru $\{0, 1, \dots, N-1\}$, która przeprowadza k na $(k+l+m) \bmod N$, przez wykreślenie wszystkich elementów każdego cyklu, które są $\geq M$. (Porównaj ćwiczenie 29). *Dowód:* Rozważmy zamianę podaną we wskazówce. Przypisujemy $x_k \leftarrow x_{k'} \text{ i } x_{k'} \leftarrow x_{k''}$ dla pewnego k , gdzie $k' = (k+l+m) \bmod N$, $k'' = (k'+l+m) \bmod N$ i $k' \geq M$. Wiemy, że $x_{k'} = x_{k''}$, zatem przestawienie $\alpha\beta\gamma \rightarrow \gamma\beta\alpha$ powoduje zastąpienie x_k przez $x_{k''}$.

Wynika stąd, że permutacja ma dokładnie $d = \gcd(l+m, m+n)$ cykli. Możemy użyć algorytmu podobnego do tego z poprzedniego ćwiczenia.

Warto wspomnieć, że istnieje nieco prostsza metoda zredukowania tego problemu do szczególnego przypadku ćwiczenia 34, jakkolwiek wymaga ona większej liczby odwołań do pamięci. Przypuśćmy, że $\gamma = \gamma'\gamma''$ gdzie $|\gamma''| = |\alpha|$. Wówczas możemy zmienić

$\alpha\beta\gamma'\gamma''$ na $\gamma''\beta\gamma'\alpha$ i wymienić γ'' z $\beta\gamma'$. Podobne podejście można zastosować również wtedy, gdy $|\alpha| > |\gamma|$. [Zobacz J. L. Mohammed, C. S. Subi, *J. Algorithms* 8 (1987), 113–121].

1.4.1

1. Sekwencja wywołania: JMP MAXN; JMP MAX100, gdy $n = 100$.

Warunki wejściowe: dla wejścia MAXN, rI3 = n ; zakładając $n \geq 1$.

Warunki wyjściowe: jak w (4).

2. MAX50 STJ EXIT

ENT3 50
JMP 2F

3. Warunki wejściowe: $n = rI1$, gdy $rI1 > 0$; w przeciwnym razie $n = 1$.

Warunki wyjściowe: rA i rI2 jak w (4); rI1 niezmieniony; rI3 = $\min(0, rI1)$; rJ = EXIT + 1; wskaźnik porównania niezmieniony, jeśli $n = 1$, w przeciwnym razie wskaźnik porównania ma wartość GREATER, EQUAL, LESS, gdy odpowiednio maksimum jest większe od $X[1]$, maksimum jest równe $X[1]$ oraz $rI2 > 1$, maksimum jest równe $X[1]$ przy $rI2 = 1$.

(Analogiczne ćwiczenie dla (9) byłoby oczywiście nieco bardziej skomplikowane).

4. SMAX1 ENT1 1 $r = 1$.

SMAX STJ EXIT Dowolne r .

JMP 2F Ciąg dalszy, jak w poprzedniej wersji.

...

DEC3 0,1 Zmniejsz o r .

J3P 1B

EXIT JMP * Wyjście.

Sekwencja wywołania: JMP SMAX; JMP SMAX1, dla $r = 1$.

Warunki wejściowe: rI3 = n , zakładając $n > 0$; dla wejścia SMAX rI1 = r , zakładając $r > 0$.

Warunki wyjściowe: rA = $\max_{0 \leq k < n/r} \text{CONTENTS}(X + n - kr) = \text{CONTENTS}(X + rI2)$; rI3 = $(n - 1) \bmod r + 1 - r = -((-n) \bmod r)$.

5. Można korzystać z dowolnego innego rejestru. Na przykład,

Sekwencja wywołania: ENTA *+2

JMP MAX100

Warunki wejściowe: brak.

Warunki wyjściowe: jak w (4).

Kod podprogramu – jak w (1), z tym, że zamiast pierwszej instrukcji wstawiamy „MAX100 STA EXIT(0:2)”.

6. (Autorzy rozwiązania: Joel Goldberg i Roger M. Aarons)

MOVE STJ 3F

STA 4F Zachowaj rA i rI2.

ST2 5F(0:2)

LD2 3F(0:2) rI2 ← adres „NOP A,I(F)”.

LDA 0,2(0:3) rA ← „A,I”.

STA *+2(0:3)

LD2 5F(0:2) Odtwórz rI2, bo I może być 2.

ENTA * rA ← adres indeksowany.

```

LD2 3F(0:2)
LD2N 0,2(4:4) rI2 ← -F.
J2Z 1F
DECA 0,2
STA 2F(0:2)
DEC1 0,2      rI1 ← rI1 + F.
ST1 6F(0:2)
2H LDA *,2
6H STA *,2
INC2 1      Zwiększa rI2 do uzyskania zera.
J2N 2B
1H LDA 4F      Odtwórz rA i rI2.
5H ENT2 *
3H JMP *      Wyjdź (skok do rozkazu NOP).
4H CON 0 ■

```

7. (1) System operacyjny może bardziej wydajnie przydzielać szybką pamięć, gdy wiadomo, że bloki są przeznaczone „tylko do odczytu”. (2) Pamięć podręczna przechowująca rozkazy może być szybsza i tańsza, gdy wiadomo, że rozkazy się nie zmieniają. (3) Technologia „potokowa” – zmodyfikowanie rozkazu po załadowaniu go do kolejki wymaga opróżnienia kolejki; podukłady sprawdzające warunki tego typu są złożone i powolne. (4) Samomodyfikujący się kod nie może być używany przez więcej niż jeden proces naraz. (5) Samomodyfikujący się kod może zmylić program śledzący skoki (ćwiczenie 1.4.3.2–7), który jest ważnym narzędziem przy „profilowaniu” (tj. przy obliczaniu, ile razy wykonał się każdy rozkaz).

1.4.2

1. Jeżeli współprogram jest wywoływany tylko raz, to jest po prostu podprogramem. Potrzebujemy zatem przykładu, w którym każdy współprogram wywołuje inny współprogram przynajmniej w dwóch miejscach. Ale nawet wówczas łatwo zazwyczaj skorzystać z własności danych lub pomocniczej zmiennej i na tej podstawie wykonywać skok w odpowiednie miejsce współprogramu. I znów wychodzi zwykły podprogram. Zastosowanie współprogramów ma sens, gdy liczba wzajemnych odwołań jest duża.

2. Zgubilibyśmy pierwszy znak wyliczony przez IN. [Zaczynamy od OUT, ponieważ musimy dokonać zainicjowania w wierszach 58–59 przed wywołaniem IN. Gdybyśmy zaczynali od IN, musielibyśmy zainicjować OUT przez wykonanie rozkazu „ENT4 -16” i wyczyszczenie bufora wyjściowego, jeżeli nie ma pewności, że zawiera spacje. W takim przypadku moglibyśmy w wierszu 62 wykonać skok do wiersza 39].

3. *Prawie* prawdziwe, bo „CMPA =10=” w IN zostaje wówczas jedynym rozkazem porównania w programie, a kodem „.” jest 40. (!) Ale jeśli kończąca kropka byłaby poprzedzona cyfrą (oznaczającą powtórzenie), to nie zauważilibyśmy problemu. (*Uwaga:* W najwydajniejszym rozwiążaniu najpewniej usunięto by wiersze 40, 44 i 48, wstawiając w zamian „CMPA PERIOD” między wiersze 26 i 27. Jeśli jednak stan wskaźnika porównania ma być przekazywany między współprogramami, trzeba zaznaczyć ten fakt w specyfikacji).

4. Oto trzy przykłady z trzech istotnie różnych klasycznych maszyn: (i) Na IBM 650, w asemblerze SOAP, sekwencje wywołania zapisalibyśmy „LDD A” oraz „LDD B”, a obsługę aktywowania „A STD BX AX” oraz „B STD AX BX”. (ii) Na IBM 709, w popular-

nym asemblerze, sekwencje wywołania wyglądałyby „**TSX A,4**” i „**TSX B,4**”, a obsługa aktywowania:

A	SXA	BX,4
AX	AXT	1-A1,4
	TRA	1,4

B	SXA	AX,4
BX	AXT	1-B1,4
	TRA	1,4

(iii) Na CDC 1604 sekwencję wywołania byłby „skok z powrotem” (**SLJ 4**) do **A** lub **B**, a obsługa aktywowania wyglądałaby na przykład tak:

A:	SLJ	B1;	ALS	0
B:	SLJ	A1;	SLJ	A

(dwa kolejne słowa 48-bitowe).

5. Należy wstawić rozkazy „**STA HOLDAIN; LDA HOLDAOUT**” między **OUT** i **OUTX** oraz „**STA HOLDAOUT; LDA HOLDAIN**” między **IN** i **INX**.

6. W A piszemy „**JMP AB**”, by aktywować B, „**JMP AC**”, by aktywować C. Lokacji **BA**, **BC**, **CA** i **CB** analogicznie użyjemy w B i C. Kod aktywujący wygląda następująco:

AB	STJ	AX	BC	STJ	BX	CA	STJ	CX
BX	JMP	B1	CX	JMP	C1	AX	JMP	A1
CB	STJ	CX	AC	STJ	AX	BA	STJ	BX
	JMP	BX		JMP	CX		JMP	AX

(*Uwaga:* Powyższa metoda wymaga $2(n - 1)n$ komórek pamięci przy n współprogramach. Dla dużego n można użyć „scentralizowanego” programu, który będzie obsługiwał aktywowanie. Łatwo wymyślić metodę wymagającą $3n + 2$ komórek. W praktyce jednak zaprezentowana metoda jest szybsza i wymaga tylko $2m$ komórek, gdzie m jest liczbą takich par (i, j) , że współprogram i aktywuje współprogram j . W wypadkach, kiedy wiele współprogramów niezależnie aktywuje się nawzajem, sterowanie podlega zazwyczaj zewnętrznym wpływom (porównaj punkt 2.2.5)).

1.4.3.1

1. Podprogram **FCHECK** jest wywoływany tylko dwa razy, a za każdym razem jego wywołanie jest poprzedzone wywołaniem podprogramu **MEMORY**. Zyskalibyśmy trochę na wydajności, przerabiając **FCHECK** na specjalne wejście do podprogramu **MEMORY**, a także umieszczając w nim kod powodujący $rI2 \leftarrow -R$.

2.	SHIFT	J5N	ADDRERROR
	DEC3	5	
	J3P	FERROR	
	LDA	AREG	
	LDX	XREG	
	LD1	1F,3(4:5)	
	ST1	2F(4:5)	
	J5Z	CYCLE	
2H	SLA	1	
	DEC5	1	
	J5P	2B	
	JMP	STOREAX	
	SLA	1	
	SRA	1	

```

SLAX 1
SRAX 1
SLC 1
1H SRC 1 ■

3. MOVE J3Z CYCLE
JMP MEMORY
SRAX 5
LD1 I1REG
LDA SIGN1
JAP *+3
J1NZ MEMERROR
STZ SIGN1(0:0)
CMP1 =BEGIN=
JGE MEMERROR
STX 0,1
LDA CLOCK
INCA 2
STA CLOCK
INC1 1
ST1 I1REG
INC5 1
DEC3 1
JMP MOVE ■

```

4. Wystarczy wstawić „IN 0(16)” i „JBUS *(16)” między wiersze 003 i 004. (Oczywiście na innym komputerze wyglądałoby to zupełnie inaczej, ponieważ zawartość karty należałoby przetłumaczyć na kod znakowy komputera MIX).

5. Czas pracy części sterującej wynosi $34u$ plus $15u$, jeśli występuje indeksowanie. Podprogram GETV zabiera $52u$ plus $5u$, jeśli $L \neq 0$. Dodatkowy czas potrzebny na faktyczne załadowanie wartości to $11u$ dla LDA i LDX, $13u$ dla LD i , $21u$ dla ENTA i ENT X , $23u$ dla ENT i (plus $2u$ dla ostatnich dwóch czasów, jeśli $M = 0$). Reasumując, całkowity czas symulacji to $97u$ dla LDA i $55u$ dla ENTA, plus $15u$ na indeksowanie oraz w pewnych przypadkach jeszcze $5u$ lub $2u$. Wygląda na to, że w tym przypadku stosunek szybkości symulatora do szybkości wykonania rozkazów na maszynie MIX wynosi mniej więcej 50:1. (Test zajmujący $178u$ czasu symulowanego wykonał się w czasie $8422u$, co daje trochę lepszą wartość 47:1).

7. Wykonanie IN lub OUT przypisuje zmiennej związanej z urządzeniem czas, w którym ma zostać wykonana transmisja. Część sterująca („CYCLE”) powinna w każdym cyklu sprawdzać, czy wartość CLOCK przekroczyła wartość którejś ze zmiennych związanych z urządzeniami. Jeśli tak, to należy przeprowadzić transmisję i ustawić wartość zmiennej na ∞ . (Jeżeli chcielibyśmy w ten sposób obsługiwać więcej niż dwa urządzenia wejścia-wyjścia, to lepiej byłoby przechowywać odpowiadające im zmienne na posortowanej liście zrealizowanej za pomocą dowiązań, zobacz punkt 2.2.5). Należy zadbać o zakończenie operacji wejścia-wyjścia przy symulacji instrukcji HLT.

8. Fałsz; rI6 może mieć wartość BEGIN, jeśli „przyjdzie” z lokacji BEGIN – 1. Ale w takim przypadku zostanie zasygnalizowany błąd MEMERROR przy próbie zasymulowania rozkazu załadowania wartości (STZ) do lokacji TIME! Z drugiej jednak strony wiersz 254 gwarantuje, że zawsze $0 \leq rI6 \leq BEGIN$.

1.4.3.2

1. Należy zmienić wiersze 48 i 49 na:

XREG ORIG **2	JMP -1,1
LEAVE STX XREG	1H JMP *+1
ST1 XREG+1	STA -1,1
LD1 JREG(0:2)	LD1 XREG+1
LDA -1,1	LDX XREG
LDX 1F	LDA AREG
STX -1,1	LEAVEX JSJ *

Najważniejszy jest oczywiście rozkaz „JSJ”.

2. * TRACE ROUTINE

ORIG *+99	STA BUF+1,1(4:5)
BUF CON 0	ENTA 8
..... wiersze 02–04	JNOV 1F
ST1 I1REG	1H ADD BIG
..... wiersze 05–07	1H JL 1F
PTR ENT1 -100	INCA 1
JBUS *(0)	JE 1F
STA BUF+1,1(0:2)	INCA 1
..... wiersze 08–11	1H STA BUF+1,1(3:3)
STA BUF+2,1	INC1 10
..... wiersze 12–13	J1N 1F
LDA AREG	OUT BUF-99(0)
STA BUF+3,1	ENT1 -100
LDA I1REG	1H ST1 PTR(0:2)
STA BUF+4,1 wiersze 14–31
ST2 BUF+5,1	LD1 I1REG
ST3 BUF+6,1 wiersze 32–35
ST4 BUF+7,1	ST1 I1REG
ST5 BUF+8,1 wiersze 36–48
ST6 BUF+9,1	LD1 I1REG
STX BUF+10,1 wiersze 49–50
LDA JREG(0:2)	B4 EQU 1(1:1)
	BIG CON B4-8,B4-1(1:1) ■

Po zakończeniu śledzenia należy wywołać dodatkowy podprogram, który zapisze ostatni bufor i przewinie taśmę nr 0.

3. Taśma jest szybsza. Ponadto redagowanie tej informacji (formowanie znaków) wymagałoby zbyt dużo pamięci, a oprócz tego zawartości taśmy nie trzeba drukować w całości.

4. Nie uzyskamy takiego śladu, jaki zażyczyliśmy sobie w ćwiczeniu 6, ponieważ naruszone jest ograniczenie (a). Pierwsza próba śledzenia CYCLE spowoduje powrót do śledzenia ENTER+1, ponieważ nadpisana zostanie zawartość komórki PREG.

6. Sugestia: należy przechowywać tablicę z zawartościami lokacji pamięci w śledzonym obszarze, które zostały zmienione przez program główny.

7. Program powinien przeszukiwać śledzony kod do napotkania pierwszego rozkazu skoku (ewentualnie warunkowego). Po zmodyfikowaniu tego i następnego rozkazu program powinien odtworzyć zawartości rejestrów i pozwolić wykonać się wszystkim

rozkazom aż do znalezionej miejsca. [Ta metoda może zawieść, jeżeli program modyfikuje własne rozkazy skoku lub wstawia rozkazy skoku w swój kod. Na dobrą sprawę możemy zakazać takich praktyk, poza STJ, który to przypadek należy i tak potraktować osobno].

1.4.4

1. (a) Nie, ponieważ nie mamy pewności, że operacja wejścia się zakończyła. (b) Nie, ponieważ istnieje ryzyko, że wykonanie operacji wejścia „przegoni” wykonanie operacji MOVE.

2. ENT1 2000

```
JBUS *(6)
MOVE 1000(50)
MOVE 1050(50)
OUT 2000(6) ■
```

3. WORDOUT STJ 1F

```
STA 0,5
INC5 1
2H CMP5 BUFMAX
1H JNE *
OUT -100,5(V)
LD5 0,5
ST5 BUFMAX
```

DEC5 100

```
JMP 2B
BUFMAX CON ENDBUF1
* BUFORY
OUTBUF1 ORIG *+100
ENDBUF1 CON ENDBUF2
OUTBUF2 ORIG *+100
ENDBUF2 CON ENDBUF1 ■
```

Na początek programu należy wstawić rozkaz „ENT5 OUTBUF1”. Na koniec programu należy wstawić

```
LDA BUFMAX
DECA 100,5
JAZ *+6
STZ 0,5
```

```
INC5 1
CMP5 BUFMAX
JNE *-3
OUT -100,5(V)
```

4. Jeżeli czas obliczeń równa się dokładnie czasowi wykonania operacji wejścia-wyjścia (sprzyjające okoliczności), to program wykonujący na zmianę obliczenia i operacje wejścia-wyjścia będzie pracował dwa razy dłużej niż program wykonujący wejście-wyjście i obliczenia równocześnie. Formalnie rzecz ujmując, niech C oznacza czas wykonania wszystkich obliczeń programu i niech T będzie całkowitym czasem zużywanym na operacje wejścia-wyjścia. Wówczas najlepszy czas, jaki możemy uzyskać przy buforowaniu, to $\max(C, T)$, podczas gdy czas wykonania bez buforowania wynosi $C + T$. Oczywiście $\frac{1}{2}(C + T) \leq \max(C, T) \leq C + T$.

Niektóre urządzenia mają niezerowy „czas rozruchu”, co powoduje dodatkową stratę czasu, gdy zbyt długo na urządzeniu nie wykonywano żadnej operacji. W takim przypadku można uzyskać ponaddwukrotne przyspieszenie. (Zobacz na przykład ćwiczenie 19).

5. Program można przyspieszyć co najwyżej $n + 1$ razy.

6. $\left\{ \begin{array}{l} \text{IN INBUF1(U)} \\ \text{ENT6 INBUF2+99} \end{array} \right\}$ lub $\left\{ \begin{array}{l} \text{IN INBUF2(U)} \\ \text{ENT6 INBUF1+99} \end{array} \right\}$

(a zapewne wcześniej jeszcze IOC 0(U), by w razie potrzeby przewinąć taśmę na początek).

7. Można użyć współprogramów:

INBUF1	ORIG *+100	INC6 1
INBUF2	ORIG *+100	J6N 2B
1H	LDA INBUF2+100,6	IN INBUF1(U)
	JMP MAIN	ENN6 100
	INC6 1	JMP 1B
	J6N 1B	WORDIN STJ MAINX
WORDIN1	IN INBUF2(U)	WORDINX JMP WORDIN1
	ENN6 100	MAIN STJ WORDINX
2H	LDA INBUF1+100,6	MAINX JMP *
	JMP MAIN	

Dodając kilka rozkazów obsługujących szczególne przypadki, możemy nawet uczynić ten program szybszym niż (4).

8. W sytuacji przedstawionej na rysunku 23 dwa czerwone bufory zostały wypełnione obrazami gotowych do wydrukowania wierszy, a jeden z nich (wskazywany przez NASTC) jest akurat drukowany. W tym samym czasie program wykonuje obliczenia między wykonaniem operacji zwolnienia i przydziału. Gdy program dokona przydziału, zielony bufor wskazywany przez NASTZ stanie się żółty, wskaźnik NASTZ przemieści się zgodnie z ruchem wskazówek zegara, a program zacznie zapełniać żółty bufor. Po zakończeniu operacji wyjścia NASTC przemieści się zgodnie z ruchem wskazówek zegara, wydrukowany bufor stanie się zielony i rozpoczęcie się drukowanie zawartości pozostałoego czerwonego bufora. Na koniec program zwolni żółty bufor, który stanie się kolejnym buforem gotowym do wydrukowania zawartości.

9, 10, 11.

Czas	Akcja ($N = 1$)	Akcja ($N = 2$)	Akcja ($N = 4$)
0	ASSIGN(BUF1)	ASSIGN(BUF1)	ASSIGN(BUF1)
1000	RELEASE, OUT BUF1	RELEASE, OUT BUF1	RELEASE, OUT BUF1
2000	ASSIGN (oczekiwanie)	ASSIGN(BUF2)	ASSIGN(BUF2)
3000		RELEASE	RELEASE
4000		ASSIGN (oczekiwanie)	ASSIGN(BUF3)
5000			RELEASE
6000			ASSIGN(BUF4)
7000			RELEASE
8000			ASSIGN (oczekiwanie)
8500	BUF1 przydzielony, zakończenie operacji wyjścia	BUF1 przydzielony, OUT BUF2	BUF1 przydzielony, OUT BUF2
9500	RELEASE, OUT BUF1	RELEASE	
10500	ASSIGN (oczekiwanie)	ASSIGN (oczekiwanie)	
15500			RELEASE

i tak dalej. Całkowity czas dla $N = 1$ wynosi $110000u$, dla $N = 2$ wynosi $89000u$, dla $N = 3$ wynosi $81500u$, dla $N \geq 4$ wynosi $76000u$.

12. Ostatnie trzy wiersze programu B należy zastąpić rozkazami

```
STA 2F
LDA 3F
CMPA 15,5(5:5)
LDA 2F
```

```

LD5 -1,5
DEC6 1
JNE 1B
JMP COMPUTE
JMP *-1 (lub JMP COMPUTEX)
2H CON 0
3H ALF 0000. ■

```

13. JRED CONTROL(U)
J6NZ *-1 ■

14. Przy $N = 1$ algorytm zadziała błędnie (prawdopodobnie odwołując się do bufora podczas dotyczącej go operacji wejścia-wyjścia). W pozostałych przypadkach pojawią się dwa żółte bufory. To może mieć sens, jeżeli program musi pracować równocześnie na dwóch buforach. Jednakże powoduje to zmniejszenie przestrzeni przeznaczonej na bufory. W ogólności w każdej chwili różnica między liczbą wykonanych operacji ASSIGN a liczbą wykonanych operacji RELEASE powinna być dodatnia i nie większa niż N .

15. U EQU 0	IN BUF3(U)
V EQU 1	OUT BUF2(V)
BUF1 ORIG **+100	IN BUF1(U)
BUF2 ORIG **+100	OUT BUF3(V)
BUF3 ORIG **+100	DEC1 3
TAPECOPY IN BUF1(U)	J1P 1B
ENT1 99	OUT BUF1(V)
1H IN BUF2(U)	HLT
OUT BUF1(V)	END TAPECOPY ■

Jest to szczególny przypadek algorytmu pokazanego na rysunku 26.

18. Rozwiążanie częściowe: W poniższym algorytmie t jest zmienną, której wartość wynosi 0, gdy urządzenie wejścia-wyjścia jest wolne, oraz 1, gdy jest zajęte.

Algorytm A (ASSIGN, wykonywany w stanie normalnym).

Bez zmian w stosunku do algorytmu 1.4.4A.

Algorytm R (RELEASE, wykonywany w stanie normalnym).

R1. Zwiększ n o jeden.

R2. Jeśli $t = 0$, wymuś przerwanie (operacja INT), powodując przejście do kroku B3. ■

Algorytm B (Podprogram obsługi bufora, obsługujący przerwania).

B1. Wznów program główny.

B2. Jeśli $n = 0$, przypisz $t \leftarrow 0$ i przejdź do B1.

B3. $t \leftarrow 1$, inicjuj operację wejścia-wyjścia dla bufora wskazywanego przez NASTC.

B4. Wznów program główny. W momencie zakończenia operacji wejścia-wyjścia przerwanie spowoduje skok do kroku B5.

B5. Przesuń NASTC na następny bufor zgodnie z ruchem wskazówek zegara.

B6. Zmniejsz n o jeden i przejdź do kroku B2. ■

19. Jeśli $C \leq L$, to możemy mieć $t_k = (k-1)L$, $u_k = t_k + T$ i $v_k = u_k + C$ wtedy i tylko wtedy, gdy $NL \geq T+C$. Jeśli $C > L$, to sytuacja jest bardziej złożona. Możemy mieć $u_k = (k-1)C + T$ i $v_k = kC + T$ wtedy i tylko wtedy, gdy istnieją takie liczby całkowite $a_1 \leq a_2 \leq \dots \leq a_n$, że $t_k = (k-1)L + a_k P$ spełnia $u_k - T \geq t_k \geq v_{k-N}$ dla $N < k \leq n$. Równoważny warunek mówi, że $NC \geq b_k$ dla $N < k \leq n$, gdzie $b_k = C + T + ((k-1)(C-L)) \bmod P$. Niech $c_l = \max\{b_{l+1}, \dots, b_n, 0\}$; wówczas c_l maleje ze wzrostem l , a najmniejsza wartość N zapewniająca płynne przetwarzanie jest najmniejszym l , takim że $c_l/l \leq C$. Ponieważ $c_l < C + T + P$ i $c_l \leq L + T + n(C-L)$, ta wartość N nigdy nie przekracza $\lceil \min\{C + T + P, L + T + n(C-L)\}/C \rceil$. [Zobacz A. Itai, Y. Raz, *CACM* **31** (1988), 1339–1342].

Dla podanych przykładów mamy zatem (a) $N = 1$; (b) $N = 2$; (c) $N = 3$, $c_N = 2.5$; (d) $N = 35$, $c_N = 51.5$; (e) $N = 51$, $c_N = 101.5$; (f) $N = 41$, $c_N = 102$; (g) $N = 11$, $c_N = 109.5$; (h) $N = 3$, $c_N = 149.5$; (i) $N = 2$, $c_N = 298.5$.

2.1

1. (a) $\text{SUIT}(\text{NEXT}(\text{TOP})) = \text{SUIT}(\text{NEXT}(242)) = \text{SUIT}(386) = 4$. (b) Λ .

2. Jeśli tylko V jest zmienną dowiązaniową (w przeciwnym razie napis $\text{CONTENTS}(V)$ nie ma sensu), której wartość jest różna od Λ . Najlepiej jednak *nie używać LOC* w takim kontekście.

3. Przyjmij $\text{NEWCARD} \leftarrow \text{TOP}$ i jeśli $\text{TOP} \neq \Lambda$, to $\text{TOP} \leftarrow \text{NEXT}(\text{TOP})$.

4. **C1.** Przyjmij $X \leftarrow \text{LOC}(\text{TOP})$. (Poczytajmy dla wygody sensowne założenie, że $\text{TOP} \equiv \text{NEXT}(\text{LOC}(\text{TOP}))$, tzn. że wartość TOP znajduje się w wycinku NEXT zawierającej ją alokacji. To założenie jest zgodne z programem (5), ponadto dzięki niemu nie musimy oddzielnie obsługiwać przypadku pustej kupki).

C2. Jeśli $\text{NEXT}(X) \neq \Lambda$, to przyjmij $X \leftarrow \text{NEXT}(X)$ i powtórz ten krok.

C3. Przyjmij $\text{NEXT}(X) \leftarrow \text{NEWCARD}$, $\text{NEXT}(\text{NEWCARD}) \leftarrow \Lambda$, $\text{TAG}(\text{NEWCARD}) \leftarrow 1$. ■

5. **D1.** Przyjmij $X \leftarrow \text{LOC}(\text{TOP})$, $Y \leftarrow \text{TOP}$. (Zobacz krok C1. Z założenia $Y \neq \Lambda$. Podczas pracy algorytmu X podąża o jeden krok za Y , tzn. $Y = \text{NEXT}(X)$).

D2. Jeśli $\text{NEXT}(Y) \neq \Lambda$, to przyjmij $X \leftarrow Y$, $Y \leftarrow \text{NEXT}(Y)$ i powtórz ten krok.

D3. (Mamy $\text{NEXT}(Y) = \Lambda$, zatem Y wskazuje na kartę na spodzie, a X na kartę przedostatnią). $\text{NEXT}(X) \leftarrow \Lambda$, $\text{NEWCARD} \leftarrow Y$. ■

6. (b) i (d). (a) *Nie!* CARD jest elementem, a nie dowiązaniem!

7. Ciąg (a) daje $\text{NEXT}(\text{LOC}(\text{TOP}))$, co w tym przypadku równa się wartości TOP . Ciąg (b) jest poprawny. To wcale nie jest zagmatwane. Przypomnijmy sobie, w jaki sposób wstawiamy wartość zmiennej liczbowej X do rejestru A; piszemy $\text{LDA } X$, a nie $\text{ENTA } X$, bo ten drugi rozkaz spowodowałby załadowanie $\text{LOC}(X)$ do rejestru.

8. Niech $rA \equiv N$, $rI1 \equiv X$.

$\text{ENTA } 0$ $\text{LD1 } \text{TOP}$ $\text{J1Z } *+4$	<u>B1.</u> $N \leftarrow 0$. $X \leftarrow \text{TOP}$. <u>B2.</u> Czy $X = \Lambda$?	<u>INCA 1</u> $\text{LD1 } 0,1(\text{NEXT})$ $\text{J1NZ } *-2$	<u>B3.</u> $N \leftarrow N + 1$. $X \leftarrow \text{NEXT}(X)$.
---	--	---	--

9. Niech $rI2 \equiv X$.

$\text{PRINTER EQU } 18$ $\text{TAG EQU } 1:1$ $\text{NEXT EQU } 4:5$ $\text{NAME EQU } 0:5$	Numer urządzenia drukarki wierszowej. Definicja pól.
---	---

PBUF	ALF	KUPKA	Komunikat: pusta kupka.
	ALF	PUSTA	
	ORIG	PBUF+24	
BEGIN	LD2	WIERZCH	Przyjmij $X \leftarrow \text{TOP}$.
	J2Z	2F	Czy kupka pusta?
1H	LDA	0,2(TAG)	$rA \leftarrow \text{TAG}(X)$.
	ENT1	PBUF	Przygotuj rozkaz MOVE.
	JBUS	*(PRINTER)	Czekaj na gotowość drukarki.
	JAZ	**+3	Czy $\text{TAG} = 0$ (karta odkryta)?
	MOVE	PAREN(3)	Nie: kopiuj nawiasy.
	JMP	*+2	
	MOVE	BLANKS(3)	Tak: kopiuj spacje.
	LDA	1,2(NAME)	$rA \leftarrow \text{NAME}(X)$.
	STA	PBUF+1	
	LD2	0,2(NEXT)	$X \leftarrow \text{NEXT}(X)$.
2H	OUT	PBUF(PRINTER)	Drukuj wiersz.
	J2NZ	1B	Jeśli $X \neq \Lambda$, to powtórz drukowanie.
DONE	HLT		
PAREN	ALF	(
BLANKS	ALF		
	ALF)	
	ALF		

■

2.2.1

1. Tak. (Trzeba konsekwentnie wstawiać elementy wyłącznie na jednym końcu).
2. Układ 325641 otrzymujemy, wykonując operacje SSSXXSSXSXXX (według oznaczeń z następnego ćwiczenia). Układu 154623 nie da się uzyskać, ponieważ wagon 2 może stać przed 3 jedynie wtedy, gdy usuniemy go ze stosu przed wstawieniem wagonu 3.

3. Ciągi dopuszczalne to dokładnie te ciągi, w których liczba napotkanych symboli X nie przekracza liczby napotkanych symboli S przy przeglądaniu ciągu z lewej do prawej.

Dwa różne ciągi dopuszczalne muszą dawać różne permutacje; ciągi są jednakowe do pewnego miejsca, gdzie w jednym jest S, a w drugim X. Ten drugi powoduje wstawienie na wyjście wagonu, który przy przetwarzaniu pierwszego ciągu nie zostanie wyprowadzony na wyjście, dopóki nie wyprowadzi się wagonu wstawionego właśnie na stos operacją S.

4. Ten problem jest równoważny wielu innym ciekawym problemom, jak zliczanie drzew binarnych, wyznaczanie liczby sposobów wstawienia nawiasów do wyrażenia lub wyznaczanie liczby sposobów podziału wielokąta na trójkąty, i pojawił się już w 1759 roku w notatkach Eulera i von Segnera (zobacz podpunkt 2.3.4.6).

W poniższym eleganckim rozwiązaniu korzysta się z „zasady symetrii” pochodzącej od D. André (1878): istnieje $\binom{2n}{n}$ ciągów symboli S i X zawierających po n wystąpień każdego z nich. Pozostaje wyznaczyć liczbę ciągów *niedopuszczalnych* (tych, które zawierają poprawną liczbę wystąpień symboli X i S, ale naruszają drugi warunek). Dla każdego niedopuszczalnego ciągu znajdź pierwszy X, dla którego liczba symboli X przewyższa liczbę symboli S. Następnie w początkowym fragmencie ciągu do tego symbolu X włącznie zamień każdy symbol X na S, a każdy symbol S na X. Otrzymasz ciąg zawierający $(n+1)$ symboli S i $(n-1)$ symboli X. W drugą stronę, dla każdego

ciągu tego typu możemy znaleźć ciąg niedopuszczalny, z którego możemy go uzyskać w opisany sposób. Na przykład ciąg XXSXSSSSXXXSS musiał być otrzymany z ciągu SSXSXXXXXXXSSS. Z tej odpowiedniości wynika, że liczba ciągów niedopuszczalnych jest równa $\binom{2n}{n-1}$, stąd $a_n = \binom{2n}{n} - \binom{2n}{n-1}$.

Za pomocą tego samego pomysłu możemy rozwiązać problem bardziej ogólny, znany w teorii prawdopodobieństwa pod nazwą „problemu głosowania”, polegający na policzeniu dla danych liczb wystąpień symboli S i X wszystkich częściowych ciągów dopuszcjalnych. Ten problem został rozwiązany już w 1708 roku przez Abrahama de Moivre'a, który pokazał, że liczba ciągów zawierających l symboli A i m symboli B oraz zawierających przynajmniej jeden początkowy spójny podciąg, w którym liczba wystąpień A jest o n większa od liczby wystąpień B, wynosi $f(l, m, n) = \binom{l+m}{\min(m, l-n)}$. W szczególności $a_n = \binom{2n}{n} - f(n, n, 1)$, jak wyżej. (De Moivre podał ten wynik bez dowodu [Philos. Trans. 27 (1711), 262–263], ale z innych fragmentów jego pracy wynika, że wiedział jak go udowodnić, ponieważ wzór jest prawdziwy dla $l \geq m + n$, a zaprezentowane podejście do podobnych problemów wykorzystujące funkcje tworzące pozwala po prostych przekształceniach algebraicznych otrzymać wniosek $f(l, m, n) = f(m + n, l - n, n)$). Najnowszą historię problemu głosowania i pewnych jego uogólnień można znaleźć w: D. E. Barton, C. L. Mallows, *Annals of Math. Statistics* 36 (1965), 236–260; zobacz także ćwiczenie 2.3.4.4–32 i punkt 5.1.4.

Przedstawiamy tu nową metodę rozwiązania problemu głosowania, w której korzysta się z podwójnych funkcji tworzących. Metoda została zapożyczona z rozwiązań problemów nieco trudniejszych, takich jak na przykład pytanie postawione w ćwiczeniu 11.

Niech g_{nm} będzie liczbą ciągów o długości n złożonych symboli S i X, w których liczba symboli X nigdy nie przekracza liczby symboli S przy przeglądaniu ciągu od lewej do prawej oraz w których występuje o m więcej symboli S niż symboli X (w całym ciągu). Wówczas $a_n = g_{(2n)0}$. Oczywiście g_{nm} jest równe zero, jeżeli $m + n$ ma wartość parzystą. Przekonamy się, że te liczby można zdefiniować za pomocą równań rekurencyjnych

$$g_{(n+1)m} = g_{n(m-1)} + g_{n(m+1)}, \quad m \geq 0, \quad n \geq 0; \quad g_{0m} = \delta_{0m}.$$

Rozważmy funkcję tworzącą dwóch zmiennych $G(x, z) = \sum_{n,m} g_{nm} x^m z^n$ i niech $g(z) = G(0, z)$. Powyższe związki rekurencyjne są równoważne równaniu

$$\left(x + \frac{1}{x}\right) G(x, z) = \frac{1}{x} g(z) + \frac{1}{z} (G(x, z) - 1), \quad \text{tj.} \quad G(x, z) = \frac{zg(z) - x}{z(x^2 + 1) - x}.$$

Niestety to równanie nic nam nie powie, gdy podstawimy $x = 0$, ale możemy posunąć się dalej, rozkładając mianownik na $z(1 - r_1(z)x)(1 - r_2(z)x)$, gdzie

$$r_1(z) = \frac{1}{2z}(1 + \sqrt{1 - 4z^2}), \quad r_2(z) = \frac{1}{2z}(1 - \sqrt{1 - 4z^2}).$$

(Zauważmy, że $r_1 + r_2 = 1/z$, $r_1 r_2 = 1$). Spróbujmy teraz skorzystać z metody heurystycznej. Problem sprowadza się do znalezienia takiej wartości $g(z)$, że $G(x, z)$ jest dana powyższym wzorem i ma nieskończone rozwinięcie w szereg potęgowy zmiennych x i z . Funkcja $r_2(z)$ ma rozwinięcie w szereg i $r_2(0) = 0$. Ponadto dla ustalonego z wartość $x = r_2(z)$ powoduje zniknięcie mianownika $G(x, z)$. To sugeruje, że powinniśmy tak dobrąć $g(z)$, by licznik też zniknął dla $x = r_2(z)$. Innymi słowy, powinniśmy zapewne

wziąć $zg(z) = r_2(z)$. Taki wybór powoduje, że wzór na $G(x, z)$ upraszcza się do postaci

$$G(x, z) = \frac{r_2(z)}{z(1 - r_2(z)x)} = \sum_{n \geq 0} (r_2(z))^{n+1} x^n z^{-1}.$$

To jest rozwinięcie w szereg spełniające równanie wyjściowe, zatem znaleźliśmy właściwą funkcję $g(z)$.

Współczynniki funkcji $g(z)$ są rozwiązaniami naszego problemu. W zasadzie możemy pójść dalej i otrzymać prostą postać wszystkich współczynników $G(x, z)$: z twierdzenia dwumianowego,

$$r_2(z) = \sum_{k \geq 0} z^{2k+1} \binom{2k+1}{k} \frac{1}{2k+1}.$$

Niech $w = z^2$ i $r_2(z) = zf(w)$. Wówczas przy oznaczeniach z ćwiczenia 1.2.6–25 mamy $f(w) = \sum_{k \geq 0} A_k(1, -2)w^k$. Stąd

$$f(w)^r = \sum_{k \geq 0} A_k(r, -2)w^k.$$

Zatem

$$G(x, z) = \sum_{n, m} A_m(n, -2) x^n z^{2m+n},$$

więc ogólnym rozwiązaniem jest

$$\begin{aligned} g_{(2n)(2m)} &= \binom{2n+1}{n-m} \frac{2m+1}{2n+1} = \binom{2n}{n-m} - \binom{2n}{n-m-1}; \\ g_{(2n+1)(2m+1)} &= \binom{2n+2}{n-m} \frac{2m+2}{2n+2} = \binom{2n+1}{n-m} - \binom{2n+1}{n-m-1}. \end{aligned}$$

5. Jeśli $j < k$ i $p_j < p_k$, to musimy zdjąć p_j ze stosu, zanim włożymy p_k . Jeśli $p_j > p_k$, to musimy pozostawić na stosie p_k , dopóki p_j się na nim nie znajdzie. Gdy połączymy te reguły, okaże się, że nie mogą jednocześnie zachodzić warunki $i < j < k$ i $p_j < p_k < p_i$, ponieważ oznaczałoby to, że musimy zdjąć ze stosu p_j przed p_k i po p_i , ale p_i występuje po p_k .

W drugą stronę. Permutację można otrzymać za pomocą następującego algorytmu: „Dla $j = 1, 2, \dots, n$ włożyć na stos tyle elementów, ile potrzeba, by pojawiło się na nim p_j . Wtedy zdejmij p_j ”. Ten algorytm nie zadziała jedynie wtedy, gdy dojdziemy do takiego j , dla którego p_j nie leży na szczycie stosu, ale jest przykryte pewnym elementem p_k , gdzie $k > j$. Wartości na stosie tworzą zawsze ciąg rosnący, zatem $p_j < p_k$, więc element musiał się tam dostać, ponieważ był mniejszy niż p_i dla pewnego $i < j$.

P. V. Ramanan [SICOMP 13 (1984), 167–169] podał charakteryzację permutacji, którą można uzyskać, gdy oprócz stosu dysponujemy m komórkami dodatkowej pamięci. (To uogólnienie jest zaskakująco trudne).

6. Wyłącznie trywialne $12\dots n$, co wynika z definicji kolejki.

7. Jeśli elementem stojącym na pierwszym miejscu permutacji wyjściowej ma być n , to przed pierwszą operacją wyjęcia elementu z kolejki musi się w niej znajdować (i) ciąg $1, 2, \dots, n$ w przypadku kolejki dwustronnej o ograniczonym wejściu; (ii) ciąg $p_1 p_2 \dots p_n$ w przypadku kolejki dwustronnej o ograniczonym wyjściu. Otrzymujemy stąd (jedyne) odpowiedzi (a) 4132, (b) 4213, (c) 4231.

8. Jeśli $n \leq 4$, to nie. Dla $n = 5$ istnieją cztery takie permutacje (zobacz ćwiczenie 13).

9. Niech $(p_1, \dots, p_n)^R = (p_n, \dots, p_1)$, a π^- oznacza permutację niwelującą efekt permutacji π . Wówczas dla dowolnej permutacji π uzyskanej za pomocą kolejki dwustronnej o ograniczonym wejściu możemy uzyskać permutację $((\pi^R)^-)^R$ za pomocą kolejki dwustronnej o ograniczonym wyjściu, wykonując operacje w odwrotnej kolejności. Podobnie w drugą stronę. Istnieje zatem wzajemnie jednoznaczna odpowiedniość między tymi dwoma zbiorami permutacji.

10. (i) Ciąg powinien zawierać n symboli X oraz w sumie n symboli S i Q. (ii) Liczba symboli X nigdy nie może przekroczyć łącznej liczby symboli S i Q napotkanych przy przeglądaniu ciągu od lewej do prawej. (iii) Zawsze, gdy liczba symboli X równa się sumarycznej liczbie symboli S i Q (przy przeglądaniu od lewej do prawej), następnym symbolem musi być Q. (iv) W ciągu nie może występować spójny podciąg XQ.

Reguły (i) i (ii) są oczywiste. Dodatkowe reguły (iii) i (iv) są potrzebne w celu uniknięcia niejednoznaczności, ponieważ operacja S ma ten sam skutek co Q, gdy kolejka jest pusta, a operacje XQ można zawsze zastąpić przez QX. Każdą permutację (która można otrzymać za pomocą kolejki dwustronnej o ograniczonym wyjściu) można zatem otrzymać za pomocą przynajmniej jednego takiego ciągu operacji.

By pokazać, że dwa różne dopuszczalne ciągi operacji dają różne permutacje, rozważmy ciągi, które do pewnego miejsca są takie same i różnią się w nim w ten sposób, że w jednym ciągu stoi S, a w drugim X lub Q. Ponieważ na mocy (iii) kolejka dwustronna nie jest pusta, permutacje uzyskane za pomocą tych ciągów są różne. Pozostaje przypadek, w którym ciągi A i B są jednakowe do pewnego miejsca, gdzie w A stoi Q, a w B stoi X. W ciągu B mogą występować kolejne symbole X, których ciąg na mocy (iv) musi się kończyć wystąpieniem symbolu S, więc również w tym przypadku permutacje są różne.

11. Postępujemy podobnie, jak w ćwiczeniu 4. Niech g_{nm} będzie liczbą częściowo dopuszczalnych ciągów operacji o długości n , powodujących pozostawienie m elementów w kolejce dwustronnej i *nie zakończonych* symbolem X. Analogiczną definicję przyjmujemy dla h_n , z tym że wymagamy, by ciągi *kończyły się* symbolem X. Mamy $g_{(n+1)m} = 2g_{n(m-1)} + h_{n(m-1)}$ [$m > 1$] oraz $h_{(n+1)m} = g_{n(m+1)} + h_{n(m+1)}$. Definiujemy $G(x, z)$ i $H(x, z)$ analogicznie, jak w ćwiczeniu 4. Mamy

$$\begin{aligned} G(x, z) &= xz + 2x^2z^2 + 4x^3z^3 + (8x^4 + 2x^2)z^4 + (16x^5 + 8x^3)z^5 + \dots; \\ H(x, z) &= z^2 + 2xz^3 + (4x^2 + 2)z^4 + (8x^3 + 6x)z^5 + \dots. \end{aligned}$$

Przyjmując $h(z) = H(0, z)$, stwierdzamy, że $z^{-1}G(x, z) = 2xG(x, z) + x(H(x, z) - h(z)) + x$ i $z^{-1}H(x, z) = x^{-1}G(x, z) + x^{-1}(H(x, z) - h(z))$. Zatem

$$G(x, z) = \frac{xz(x - z - xh(z))}{x - z - 2x^2z + xz^2}.$$

Jak w ćwiczeniu 4 próbujemy tak dobrać $h(z)$, by licznik upraszczał się z czynnikiem mianownika. Stwierdzamy, że $G(x, z) = xz/(1 - 2xr_2(z))$, gdzie

$$r_2(z) = \frac{1}{4z}(z^2 + 1 - \sqrt{(z^2 + 1)^2 - 8z^2}).$$

Jeśli umówimy się, że $b_0 = 1$, to szukaną funkcję tworzącą można zapisać jako:

$$\frac{1}{2}(3 - z - \sqrt{1 - 6z + z^2}) = 1 + z + 2z^2 + 6z^3 + 22z^4 + 90z^5 + \dots.$$

Różniczkując znajdujemy wygodny związek rekurencyjny: $nb_n = 3(2n - 3)b_{n-1} - (n - 3)b_{n-2}$, $n \geq 2$.

W innej metodzie rozwiązania tego problemu, zaproponowanej przez V. Pratta, wykorzystuje się gramatyki bezkontekstowe do opisu zbioru słów (zobacz rozdział 10). Nieskończona gramatyka o produkcjach $S \rightarrow q^n(Bx)^n$, $B \rightarrow sq^n(Bx)^{n+1}B$, dla wszystkich $n \geq 0$ i $B \rightarrow \epsilon$, jest jednoznaczna i pozwala wyznaczyć liczbę słów zawierających n symboli x , jak w ćwiczeniu 2.3.4.4–31.

12. Na mocy wzoru Stirlinga $a_n = 4^n / \sqrt{\pi n^3} + O(4^n n^{-5/2})$. Zanim zabierzemy się za b_n , zastanówmy się nad ogólnym problemem oszacowania współczynników stojących przy w^n w szeregu potęgowym będącym rozwinięciem $\sqrt{1-w}\sqrt{1-\alpha w}$ dla $|\alpha| < 1$. Mamy

$$\sqrt{1-w}\sqrt{1-\alpha w} = \sqrt{1-w}\sqrt{1-\alpha+\alpha(1-w)} = \sqrt{1-\alpha}\sum_k \binom{1/2}{k} \beta^k (1-w)^{k+1/2},$$

gdzie $\beta = \alpha/(1-\alpha)$; stąd poszukiwane współczynniki są postaci

$$(-1)^n \sqrt{1-\alpha} \sum_k \binom{1/2}{k} \beta^k \binom{k+1/2}{n}.$$

Teraz

$$(-1)^n \binom{k+1/2}{n} = \binom{n-k-3/2}{n} = \frac{\Gamma(n-k-1/2)}{\Gamma(n+1)\Gamma(-k-1/2)} = -\frac{(k+1/2)^{k+1}}{\sqrt{\pi n}} n^{-k-1/2},$$

oraz $n^{-k-1/2} = \sum_{j=0}^m \binom{-k-1/2}{-k-1/2-j} n^{-k-1/2-j} + O(n^{-k-3/2-m})$ na mocy 1.2.11.1–(16).

Otrzymujemy zatem szereg asymptotyczny $[w^n] \sqrt{1-w}\sqrt{1-\alpha w} = c_0 n^{-3/2} + c_1 n^{-5/2} + \dots + c_m n^{-m-3/2} + O(n^{-m-5/2})$, gdzie

$$c_j = -\sqrt{\frac{1-\alpha}{\pi}} \sum_{k=0}^j \binom{1/2}{k} (k+\frac{1}{2})^{k+1} \binom{j+1/2}{k+1/2} \frac{\alpha^k}{(1-\alpha)^k}.$$

Dla b_n piszemy $1-6z+z^2 = (1-(3+\sqrt{8})z)(1-(3-\sqrt{8})z)$ i podstawiamy $w = (3+\sqrt{8})z$, $\alpha = (3-\sqrt{8})/(3+\sqrt{8})$, otrzymując wzór asymptotyczny

$$b_n = \frac{(\sqrt{2}-1)(3+\sqrt{8})^n}{2^{3/4}\pi^{1/2}n^{3/2}} (1+O(n^{-1})) = \frac{(\sqrt{2}+1)^{2n-1}}{2^{3/4}\pi^{1/2}n^{3/2}} (1+O(n^{-1})).$$

13. V. Pratt odkrył, że permutacji nie da się otrzymać wtedy i tylko wtedy, gdy zawiera podciąg, który z dokładnością do stałej addytywnej jest postaci

$$5, 2, 7, 4, \dots, 4k+1, 4k-2, 3, 4k, 1 \quad \text{lub} \quad 5, 2, 7, 4, \dots, 4k+3, 4k, 1, 4k+2, 3$$

dla pewnego $k \geq 1$, lub takiej samej postaci z dokładnością do kolejności dwóch ostatnich elementów i/lub elementów 1 i 2. Stąd dla $k = 1$ zabronionymi podciągami są 52341, 52314, 51342, 51324, 5274163, 5274136, 5174263, 5174236. [STOC 5 (1973), 268–277].

14. (Autor rozwiązania: R. Melville, 1980). Niech R i S będą takimi stosami, że kolejkę tworzą elementy stosu R od góry do dołu i dalej elementy stosu S od dołu do góry. Gdy stos R się opróżni, należy przełożyć (po jednym) wszystkie elementy ze stosu S na stos R . By usunąć element z kolejki, zdejmij element ze stosu R , który nie będzie pusty, dopóki nie opróżni się symulowana kolejka. By wstawić element na koniec kolejki, włoż go na stos S (chyba, że stos R jest pusty). Każdy element jest co najwyżej dwa razy wkładany na stos i co najwyżej dwa razy zdejmowany ze stosu, zanim opuści kolejkę.

2.2.2

1. M – 1, nie M. Jeżeli dopuścilibyśmy M elementów, jak w (6) i (7), to nie mielibyśmy możliwości odróżnienia kolejki pełnej od pustej na podstawie samych tylko wartości R i F, ponieważ rozróżnić możemy tylko M przypadków. Lepiej już poświęcić jedną komórkę pamięci, niż przesadnie komplikować program.

2. Usuwanie z końca: jeśli R = F, to NIEDOMIAR; Y \leftarrow X[R]; jeśli R = 1, to R \leftarrow M, w przeciwnym razie R \leftarrow R – 1. Wstawianie na początek: X[F] \leftarrow Y; jeśli F = 1, to F \leftarrow M, w przeciwnym razie F \leftarrow F – 1; jeśli F = R, to PRZEPEŁNIENIE.

3. (a) LD1 I; LDA BASE,7:1. Potrzeba 5 cykli zamiast 4 lub 8 jak w (8).

(b) Rozwiążanie 1: LDA BASE,2:7, gdy w lokacjach przechowujących adresy bazowe I₁ = 0, I₂ = 1. Rozwiążanie 2: Jeśli wolimy w lokacjach przechowujących adresy bazowe mieć I₁ = I₂ = 0, to możemy napisać LDA X2,7:1, gdzie lokacja X2 zawiera NOP BASE,2:7. Rozwiążanie drugie zabiera jeden cykl więcej, ale pozwala korzystać z dowolnego rejestru indeksowego.

(c) Rozkaz jest równoważny „LD4 X(0:2)” i jego wykonanie zabiera tyle samo czasu, z tym że do rI4 zostanie zapisana wartość +0, gdy X(0:2) zawiera –0.

4. (a) NOP *,7. (b) LDA X,7:7(0:2). (c) Nie da się: rozkaz LDA Y,7:7, gdzie lokacja Y zawiera NOP X,7:7, łamie ograniczenie dotyczące 7:7. (Zobacz ćwiczenie 5). (d) LDA X,7:1, gdy dysponujemy pomocniczymi stałymi:

```
X  NOP  *+1,7:2
      NOP  *+1,7:3
      NOP  *+1,7:4
      NOP  0,5:6
```

Czas wykonania wynosi 6 jednostek. (e) INC6 X,7:6, gdy X zawiera NOP 0,6:6.

5. (a) Rozważ rozkaz ENTA 1000,7:7 przy zawartości pamięci

lokacja	ADDRESS	I ₁	I ₂
1000:	1001	7	7
1001:	1004	7	1
1002:	1002	2	2
1003:	1001	1	1
1004:	1005	1	7
1005:	1006	1	7
1006:	1008	7	7
1007:	1002	7	1
1008:	1003	7	2

gdy rI1 = 1, rI2 = 2. Okazuje się, że 1000,7,7 = 1001,7,7,7 = 1004,7,1,7,7 = 1005,1,7,1,7,7 = 1006,7,1,7,7 = 1008,7,7,1,7,7 = 1003,7,2,7,1,7,7 = 1001,1,1,2,7,1,7,7 = 1002,1,2,7,1,7,7 = 1003,2,7,1,7,7 = 1005,7,1,7,7 = 1006,1,7,1,7,7 = 1007,7,1,7,7 = 1002,7,1,1,7,7 = 1002,2,2,1,1,7,7 = 1004,2,1,1,7,7 = 1006,1,1,7,7 = 1007,1,7,7 = 1008,7,7 = 1003,7,2,7 = 1001,1,1,2,7 = 1002,1,2,7 = 1003,2,7 = 1005,7 = 1006,1,7 = 1007,7 = 1002,7,1 = 1002,2,2,1 = 1004,2,1 = 1006,1 = 1007. (Szybszą metodą obliczenia tego wyniku byłoby wyliczenie kolejno adresów wyznaczanych przez lokacje 1002, 1003, 1007, 1008, 1005, 1006, 1004, 1001, 1000, w tym porządku. Ale komputer musi wykonać to obliczenie dokładnie tak jak pokazano). Autor wypróbował kilka wymyslnych schematów polegających na zmienianiu zawartości pamięci podczas wyznaczania adresu i odtwarzaniu jej po tej operacji.

Podobne algorytmy występują w punkcie 2.3.5. Te próby okazały się jednak bezwocne. Wydaje się, że po prostu nie ma miejsca na przechowywanie dodatkowych informacji.

(b,c) Niech H i C będą pomocniczymi rejestrami i niech N będzie licznikiem. Efektywny adres M dla instrukcji w lokacji L wyznaczamy następująco:

- A1.** [Inicjowanie] Przyjmij $H \leftarrow 0$, $C \leftarrow L$, $N \leftarrow 0$. (W tym algorytmie C jest lokacją „bieżącej”, H służy do sumowania zawartości różnych rejestrów indeksowych, a N służy do mierzenia głębokości adresowania pośredniego).
- A2.** [Badanie adresu] Przyjmij $M \leftarrow \text{ADDRESS}(C)$. Jeśli $I_1(C) = j$, $1 \leq j \leq 6$, to $M \leftarrow M + rIj$. Jeśli $I_2(C) = j$, $1 \leq j \leq 6$, to $H \leftarrow H + rIj$. Jeśli $I_1(C) = I_2(C) = 7$, to $N \leftarrow N + 1$, $H \leftarrow 0$.
- A3.** [Czy adresowanie pośrednie?] Jeśli albo $I_1(C)$, albo $I_2(C)$ równa się 7, to przyjmij $C \leftarrow M$ i przejdź do A2. W przeciwnym razie przyjmij $M \leftarrow M + H$, $H \leftarrow 0$.
- A4.** [Zmniejszanie głębokości] Jeśli $N > 0$, to przyjmij $C \leftarrow M$, $N \leftarrow N - 1$ i przejdź do A2. W przeciwnym razie M jest poszukiwanym adresem. ■

Powyższy algorytm obsłuży poprawnie każdą sytuację poza tą, w której $I_1 = 7$ i $1 \leq I_2 \leq 6$ oraz wyznaczanie adresu zawiera przypadek $I_1 = I_2 = 7$. Skutek jest taki, jakby I_2 było zerem. W celu zrozumienia algorytmu A przypomnijmy sobie notację z punktu (a); stan „L,7,1,2,5,2,7,7,7,7” jest reprezentowany przez C lub $M = L$, $N = 4$ (liczba kończących siódemek) i $H = rI1 + rI2 + rI5 + rI2$ (indeksowanie na końcu). W rozwiązaniu punktu (b) licznik N będzie zawsze równy 0 albo 1.

6. (c) Powoduje PRZEPEŁNIENIE. (e) Powoduje NIEDOMIAR, a jeśli program będzie działał dalej, spowoduje PRZEPEŁNIENIE przy ostatniej operacji I_2 .

7. Nie, ponieważ $\text{TOP}[i]$ musi być większe niż $\text{OLDTOP}[i]$.

8. W przypadku stosu istotne informacje występują z jednej strony, a wolna przestrzeń z drugiej:



gdzie $A = \text{BASE}[j]$, $B = \text{TOP}[j]$, $C = \text{BASE}[j + 1]$. W przypadku kolejki lub kolejki dwustronnej istotne informacje występują na końcach, a wolna przestrzeń po środku:



lub wolna przestrzeń na końcach, a istotne informacje po środku:



gdzie $A = \text{BASE}[j]$, $B = \text{REAR}[j]$, $C = \text{FRONT}[j]$, $D = \text{BASE}[j + 1]$. Te dwa przypadki można łatwo odróżnić: jeśli kolejka jest niepusta, to w pierwszym zachodzi $B \leq C$, w drugim $B > C$. Jeśli wiadomo, że kolejka się nie przepiąła, warunki te można zastąpić odpowiednio przez $B < C$ i $B \geq C$. Algorytm należy zatem tak zmodyfikować, by powiększał lub zmniejszał przerwy między blokami istotnych informacji. (Zatem w przypadku przepelnienia, gdy $B = C$, tworzymy wolną przestrzeń między B i C , przesuwając jeden z bloków zawierających informacje). Przy obliczaniu SUM i $D[j]$

w kroku G2, każdą kolejkę należy traktować tak, jakby miała o jeden element więcej (zobacz ćwiczenie 1).

9. Dla dowolnego ciągu a_1, a_2, \dots, a_m potrzeba jednej operacji przemieszczenia dla każdej pary (j, k) , takiej że $j < k$ i $a_j > a_k$. (Taka para jest nazywana „inwersją”; zobacz punkt 5.1.1). Liczba takich par jest zatem liczbą potrzebnych przemieszczeń. Wyobraźmy sobie wszystkie n^m sekwencji. Dla każdej pary (j, k) spośród $\binom{m}{2}$ par, dla których $j < k$, musimy policzyć, ile jest takich ciągów, że $a_j > a_k$. To jest oczywiście liczba wyborów a_j i a_k , czyli $\binom{n}{2}$, razy liczba sposobów wypełnienia pozostałych miejsc, czyli n^{m-2} . Całkowita liczba przemieszczeń dla wszystkich ciągów wynosi zatem $\binom{m}{2} \binom{n}{2} n^{m-2}$. By otrzymać średnią (14) dzielimy tę wartość przez n^m .

10. Tak jak w ćwiczeniu 9 okazuje się, że wartość oczekiwana wynosi

$$\begin{aligned} \binom{m}{2} \sum_{1 \leq j \leq k \leq n} p_j p_k &= \frac{1}{2} \binom{m}{2} ((p_1 + \dots + p_n)^2 - (p_1^2 + \dots + p_n^2)) \\ &= \frac{1}{2} \binom{m}{2} (1 - (p_1^2 + \dots + p_n^2)). \end{aligned}$$

Dla tego modelu *nie ma zupełnie znaczenia*, jakie będzie względne ustawienie list! (Po chwili zastanowienia widać dlaczego: jeżeli rozważamy wszystkie możliwe permutacje ciągu a_1, \dots, a_m , to całkowita liczba przemieszczeń sumowana po wszystkich permutacjach zależy wyłącznie od liczby par różnych elementów $a_j \neq a_k$).

11. Zliczając tak jak w poprzednich ćwiczeniach, otrzymujemy, że wartość oczekiwana wynosi

$$\frac{1}{n^m} \binom{n}{2} \sum_{0 \leq s < m} \sum_{r \geq t} \binom{s}{r} (n-1)^{s-r} n^{m-s-2} (m-s-1).$$

s oznacza $j-1$ w terminologii użytej w poprzedniej odpowiedzi, a r jest liczbą elementów w a_1, a_2, \dots, a_s równych a_j . Ten wzór można trochę uprościć, zapisując odpowiadającą mu funkcję tworzącą. Dochodzimy w ten sposób do

$$\frac{1}{2n^t} \sum_{k=0}^{m-t} \binom{t-1+k}{k} \binom{m-t-k}{2} \left(1 - \frac{1}{n}\right)^{k+1} \quad \text{dla } t \geq 0.$$

Czy da się tę odpowiedź zapisać w prostszej postaci? Nie, ponieważ dla ustalonych n i t funkcja tworząca wygląda następująco

$$\sum_m E_{mnt} z^m = \frac{n-1}{2n} \frac{z^2}{(1-z)^3} \left(\frac{z}{n-(n-1)z} \right)^t.$$

12. Jeśli $m = 2k$, to średnia wynosi 2^{-2k} razy

$$\binom{2k}{0} 2k + \binom{2k}{1} (2k-1) + \dots + \binom{2k}{k} k + \binom{2k}{k+1} (k+1) + \dots + \binom{2k}{2k} 2k.$$

Ta suma to

$$\binom{2k}{k} k + 2 \left(\binom{2k-1}{k} 2k + \dots + \binom{2k-1}{2k-1} 2k \right) = \binom{2k}{k} k + 4k \cdot \frac{1}{2} \cdot 2^{2k-1}.$$

Podobne rozumowanie można przeprowadzić w przypadku $m = 2k+1$. Odpowiedią jest

$$\frac{m}{2} + \frac{m}{2^m} \binom{m-1}{\lfloor m/2 \rfloor}.$$

13. A. C. Yao udowodnił, że dla dużych m , gdy $p < \frac{1}{2}$, mamy

$$\mathbb{E} \max(k_1, k_2) = \frac{1}{2}m + (2\pi(1-2p))^{-1/2} \sqrt{m} + O(m^{-1/2}(\log m)^2)$$

[SICOMP 10 (1981), 398–403]. P. Flajolet rozszerzył tę analizę, pokazując w szczególności, że asymptotyczna wartość oczekiwana dla $p = \frac{1}{2}$ równa się αm , gdzie

$$\alpha = \frac{1}{2} + 8 \sum_{n \geq 1} \frac{\sin(n\pi/2) \cosh(n\pi/2)}{n^2 \pi^2 \sinh n\pi} \approx 0.6753144833.$$

Ponadto, gdy $p > \frac{1}{2}$, końcowa wartość k_1 zbliża się do rozkładu jednostajnego przy $m \rightarrow \infty$. Zatem $\mathbb{E} \max(k_1, k_2) \approx \frac{3}{4}m$. [Zobacz Lecture Notes in Comp. Sci. 233 (1986), 325–340].

14. Niech $k_j = n/m + \sqrt{m}x_j$. (Ten pomysł pochodzi od N. G. de Bruijna). Z rozwińcia Stirlinga

$$\begin{aligned} n^{-m} \frac{m!}{k_1! \dots k_n!} \max(k_1, \dots, k_n) \\ = (\sqrt{2\pi})^{1-n} n^{n/2} \left(\frac{m}{n} + \sqrt{m} \max(x_1, \dots, x_n) \right) \\ \times \exp \left(-\frac{n}{2}(x_1^2 + \dots + x_n^2) \right) (\sqrt{m})^{1-m} \left(1 + O\left(\frac{1}{\sqrt{m}}\right) \right), \end{aligned}$$

gdy $k_1 + \dots + k_n = m$ i gdy zmienne x są jednostajnie ograniczone. Suma tej wielkości względem wszystkich nieujemnych k_1, \dots, k_n spełniających ten warunek jest przybliżeniem pewnej całki Riemanna. Możemy wnioskować, że asymptotycznie ta suma zachowuje się jak $a_n(m/n) + c_n \sqrt{m} + O(1)$, gdzie

$$\begin{aligned} a_n &= (\sqrt{2\pi})^{1-n} n^{n/2} \int_{x_1+\dots+x_n=0} \exp \left(-\frac{n}{2}(x_1^2 + \dots + x_n^2) \right) dx_2 \dots dx_n, \\ c_n &= (\sqrt{2\pi})^{1-n} n^{n/2} \int_{x_1+\dots+x_n=0} \max(x_1, \dots, x_n) \exp \left(-\frac{n}{2}(x_1^2 + \dots + x_n^2) \right) dx_2 \dots dx_n, \end{aligned}$$

gdzie można pokazać, że odpowiednie sumy zbliżają się na ϵ do a_n i c_n dla dowolnego ϵ .

Wiemy, że $a_n = 1$, stąd sumę można obliczyć. Całka pojawiająca się w wyrażeniu opisującym c_n równa się nI_1 , gdzie

$$I_1 = \int_{\substack{x_1+\dots+x_n=0 \\ x_1 \geq x_2, \dots, x_n}} x_1 \exp \left(-\frac{n}{2}(x_1^2 + \dots + x_n^2) \right) dx_2 \dots dx_n.$$

Podstawiamy

$$x_1 = \frac{1}{n}(y_2 + \dots + y_n), \quad x_2 = x_1 - y_2, \quad x_3 = x_1 - y_3, \quad \dots, \quad x_n = x_1 - y_n;$$

i otrzymujemy $I_1 = I_2/n^2$, gdzie

$$I_2 = \int_{y_2, \dots, y_n \geq 0} (y_2 + \dots + y_n) \exp \left(-\frac{Q}{2} \right) dy_2 \dots dy_n,$$

a $Q = n(y_2^2 + \dots + y_n^2) - (y_2 + \dots + y_n)^2$. Teraz przez symetrię I_2 równa się $(n-1)$ razy ta sama całka z $(y_2 + \dots + y_n)$ zamienionym na y_2 . Stąd $I_2 = (n-1)I_3$, gdzie

$$\begin{aligned} I_3 &= \int_{y_2, \dots, y_n \geq 0} (ny_2 - (y_2 + \dots + y_n)) \exp \left(-\frac{Q}{2} \right) dy_2 \dots dy_n \\ &= \int_{y_3, \dots, y_n \geq 0} \exp \left(-\frac{Q_0}{2} \right) dy_3 \dots dy_n; \end{aligned}$$

tu Q_0 to Q , w którym y_2 jest zastąpione przez zero. [Dla $n = 2$, niech $I_3 = 1$]. Teraz weźmy $z_j = \sqrt{n}y_j - (y_3 + \dots + y_n)/(\sqrt{2} + \sqrt{n})$, $3 \leq j \leq n$. Wówczas $Q_0 = z_3^2 + \dots + z_n^2$ i wnioskujemy, że $I_3 = I_4/n^{(n-3)/2}\sqrt{2}$, gdzie

$$\begin{aligned} I_4 &= \int_{y_3, \dots, y_n \geq 0} \exp\left(-\frac{z_3^2 + \dots + z_n^2}{2}\right) dz_3 \dots dz_n \\ &= \alpha_n \int \exp\left(-\frac{z_3^2 + \dots + z_n^2}{2}\right) dz_3 \dots dz_n = \alpha_n (\sqrt{2\pi})^{n-2}, \end{aligned}$$

gdzie α_n jest „kątem bryłowym” rozpinanym w przestrzeni $(n-2)$ -wymiarowej przez wektory $(n + \sqrt{2n}, 0, \dots, 0) - (1, 1, \dots, 1), \dots, (0, 0, \dots, n + \sqrt{2n}) - (1, 1, \dots, 1)$, dzielonym przez całkowity kąt brylowy całej przestrzeni. Stąd

$$c_n = \frac{(n-1)\sqrt{n}}{2\sqrt{\pi}} \alpha_n.$$

Mamy $\alpha_2 = 1$, $\alpha_3 = \frac{1}{2}$, $\alpha_4 = \pi^{-1} \arctan \sqrt{2} \approx .304$ i

$$\alpha_5 = \frac{1}{8} + \frac{3}{4\pi} \arctan \frac{1}{\sqrt{8}} \approx .206.$$

[Wartość c_3 została wyznaczona przez Roberta M. Kozelkę, *Annals of Math. Stat.* **27** (1956), 507–512, ale nikt nie opublikował rozwiązań tego problemu dla większych wartości n].

16. Jeżeli kolejki nie spełniają ograniczeń narzucających proste sposoby realizacji, jak (4) i (5), to niestety nie.

17. Po pierwsze należy pokazać, że zawsze $\text{BASE}[j]_0 \leq \text{BASE}[j]_1$. Następnie zauważmy, że każde przepelenie stosu i w $s_0(\sigma)$, które nie jest jednocześnie przepeleniem w $s_1(\sigma)$, pojawia się w chwili, gdy stos i przekroczył swój dotychczasowy największy rozmiar, a jego nowy rozmiar nie jest mniejszy niż rozmiar pamięci pierwotnie przydzielonej na stos i w $s_1(\sigma)$.

18. Założymy, że koszt włożenia elementu równa się a plus $bN + cn$, jeżeli trzeba przeprowadzić reorganizację, gdzie N jest liczbą zajętych komórek pamięci. Niech zdjęcie elementu kosztuje d . Przyjmijmy, że po reorganizacji, w wyniku której pozostaje N komórek zajętych i $S = M - N$ wolnych, do następnej reorganizacji każde włożenie elementu kosztuje $a + b + 10c + 10(b + c)nN/S = O(1 + n\alpha/(1 - \alpha))$, gdzie $\alpha = N/M$. Jeśli przed reorganizacją pojawi się p operacji włożenia i q operacji zdjęcia elementu, to zgodnie z tym, co przyjęliśmy, łączny koszt wynosi $p(a + b + 10c + 10(b + c)nN/S) + qd$, podczas gdy rzeczywisty łączny koszt wynosi $pa + bN' + cn + qd \leq pa + pb + bN + cn + qd$. To jest mniej niż koszt, który przyjęliśmy, ponieważ $p > 0.1S/n$. Założenie $M \geq n^2$ pociąga za sobą $cS/n + (b + c)N \geq bN + cn$.

19. Można po prostu zmniejszyć wszystkie indeksy o jeden, ale poniższe rozwiązanie jest trochę bardziej eleganckie. Początkowo $T = F = R = 0$.

Włożenie Y na stos X: Jeżeli $T = M$, to PRZEPEŁNIENIE; $X[T] \leftarrow Y$; $T \leftarrow T + 1$.

Zdjęcie Y ze stosu X: jeśli $T = 0$, to NIEDOMIAR; $T \leftarrow T - 1$; $Y \leftarrow X[T]$.

Wstawienie Y do kolejki X: $X[R] \leftarrow Y$; $R \leftarrow (R + 1) \bmod M$; jeśli $R = F$, to PRZEPEŁNIENIE.

Usunięcie Y z kolejki X: jeśli $F = R$, to NIEDOMIAR; $Y \leftarrow X[F]$; $F \leftarrow (F + 1) \bmod M$.

Tak jak przedtem, T jest liczbą elementów na stosie, a $(R - F) \bmod M$ liczbą elementów w kolejce. Wierzchołek stosu znajduje się teraz jednak w $X[T - 1]$, a nie w $X[T]$.

Chociaż informatykom prawie zawsze wygodniej jest liczyć od zera, reszta świata zapewne nigdy nie przerzuci się na taki sposób numerowania. Nawet Edsger Dijkstra liczy „1–2–3–4 | 1–2–3–4”, grając na fortepianie!

2.2.3

1. PRZEPEŁNIENIE niejawnie występuje w operacji $P \leftarrow \text{AVAIL}$.

2. **INSERT** STJ 1F Zapisz lokację „NOP T”.
 STJ 9F Ustaw adres powrotu.
 LD1 AVAIL $rI1 \leftarrow \text{AVAIL}$.
 J1Z OVERFLOW
 LD3 0,1(LINK)
 ST3 AVAIL
 STA 0,1(INFO) $\text{INFO}(rI1) \leftarrow Y$.
 1H LD3 *(0:2) $rI3 \leftarrow \text{LOC}(T)$.
 LD2 0,3 $rI2 \leftarrow T$.
 ST2 0,1(LINK) $\text{LINK}(rI1) \leftarrow T$.
 ST1 0,3 $T \leftarrow rI1$.
 9H JMP * ■
3. **DELETE** STJ 1F Zapisz lokację „NOP T”.
 STJ 9F Ustaw adres powrotu.
 1H LD2 *(0:2) $rI2 \leftarrow \text{LOC}(T)$.
 LD3 0,2 $rI3 \leftarrow T$.
 J3Z 9F Is $T = \Lambda$?
 LD1 0,3(LINK) $rI1 \leftarrow \text{LINK}(T)$.
 ST1 0,2 $T \leftarrow rI1$.
 LDA 0,3(INFO) $rA \leftarrow \text{INFO}(rI1)$.
 LD2 AVAIL $\text{AVAIL} \Leftarrow rI3$.
 ST2 0,3(LINK)
 ST3 AVAIL
 ENT3 2 Wyjdź drugim wyjściem.
 9H JMP *,3 ■
4. **OVERFLOW** STJ 9F Zapisz rJ.
 ST1 8F(0:2) Przechowaj rI1.
 LD1 POOLMAX
 ST1 AVAIL Nowa lokacja do AVAIL.
 INC1 c
 ST1 POOLMAX Zwiększa POOLMAX.
 CMP1 SEQMIN
 JG TOOBAD Czy przepelniliśmy pamięć?
 STZ -c,1(LINK) $\text{LINK}(\text{AVAIL}) \leftarrow \Lambda$.
 9H ENT1 * Weź początkową wartość rJ.
 DEC1 2 Odejmij 2.
 ST1 **+2(0:2) Ustaw adres powrotu.
 8H ENT1 * Odtwórz rI1.
 JMP * Powrót. ■

5. Wstawianie na początek kolejki prawie nie różni się od prostej operacji wstawienia (8) poszerzonej o sprawdzenie, czy kolejka jest pusta: $P \leftarrow \text{AVAIL}$, $\text{INFO}(P) \leftarrow Y$, $\text{LINK}(P) \leftarrow F$; jeśli $F = \Lambda$, to $R \leftarrow P$; $F \leftarrow P$.

By usunąć element z końca kolejki, musimy znaleźć element zawierający dowiązanie do $\text{NODE}(R)$, a tego nie da się zrobić wydajnie, ponieważ musimy przeszukać całą kolejkę począwszy od F . Można zrobić to na przykład tak:

- a) Jeśli $F = \Lambda$, to NIEDOMIAR, w przeciwnym razie przyjmij $P \leftarrow \text{LOC}(F)$.

- b) Jeśli $\text{LINK}(P) \neq R$, to przyjmij $P \leftarrow \text{LINK}(P)$ i powtarzaj ten krok, aż $\text{LINK}(P) = R$.
c) Przyjmij $Y \leftarrow \text{INFO}(R)$, $\text{AVAIL} \leftarrow R$, $R \leftarrow P$, $\text{LINK}(P) \leftarrow \Lambda$.

6. Moglibyśmy usunąć operację $\text{LINK}(P) \leftarrow \Lambda$ z (14), jeśli usunęlibyśmy z (17) „ $F \leftarrow \text{LINK}(P)$ ” oraz „jeśli $F = \Lambda$, to $R \leftarrow \text{LOC}(F)$ ”. Ten drugi fragment należy zastąpić przez „jeśli $F = R$, to $F \leftarrow \Lambda$ i $R \leftarrow \text{LOC}(F)$ ”, w przeciwnym razie $F \leftarrow \text{LINK}(P)$ ”.

Skutkiem tych zmian pole LINK będzie zawierało mylną informację, która jednak nigdy nie zostanie odczytana przez program. Sztuczki tego rodzaju pozwalają zaoszczędzić na czasie wykonania, jednakże stanowią pogwałcenie jednego z podstawowych założeń odśmiecania (zobacz punkt 2.3.5), zatem w systemach z odśmiecaniem stosować ich nie można.

7. (Upewnij się, czy Twoje rozwiązańe radzi sobie z listą pustą).

I1. Przyjmij $P \leftarrow \text{FIRST}$, $Q \leftarrow \Lambda$.

I2. Jeśli $P \neq \Lambda$, to przyjmij $R \leftarrow Q$, $Q \leftarrow P$, $P \leftarrow \text{LINK}(Q)$, $\text{LINK}(Q) \leftarrow R$ i powtórz ten krok.

I3. Przyjmij $\text{FIRST} \leftarrow Q$. ■

W istocie zdejmujemy elementy z jednego stosu i wkładamy je na drugi.

8.	LD1 FIRST	1 <u>I1.</u> $P \equiv rI1 \leftarrow \text{FIRST}$.
	ENT2 0	1 $Q \equiv rI2 \leftarrow \Lambda$.
	J1Z 2F	1 <u>I2.</u> Jeśli pierwsza lista jest pusta, wykonaj skok.
1H	ENTA 0,2	$n R \equiv rA \leftarrow Q$.
	ENT2 0,1	$n Q \leftarrow P$.
	LD1 0,2(LINK)	$n P \leftarrow \text{LINK}(Q)$.
	STA 0,2(LINK)	$n \text{LINK}(Q) \leftarrow R$.
	J1NZ 1B	n Czy $P \neq \Lambda$?
2H	ST2 FIRST	1 <u>I3.</u> $\text{FIRST} \leftarrow Q$. ■

Czas $(7n + 6)u$. Można osiągnąć $(5n + \text{stała})u$, zobacz ćwiczenie 1.1–3.

9. (a) Tak. (b) Tak, jeśli rozważamy bycie rodzicem w sensie biologicznym; nie, jeśli rozważamy bycie rodzicem w sensie prawnym (na przykład kobieta może poślubić teścia swojego ojca, jak w piosenке „W ten sposób zostałem swoim własnym dziadkiem”). (c) Nie ($-1 \prec 1$ i $1 \prec -1$). (d) Miejmy nadzieję. W przeciwnym razie gdzieś jest błędne koło. (e) $1 \prec 3$ i $3 \prec 1$. (f) Stwierdzenie jest wieloznaczne. Jeżeli przyjmiemy, że zbiór podprogramów wywoływanych przez podprogram y zależy od tego, jaki podprogram wywołuje y , to stwierdzimy, że relacja może nie mieć własności przechodniości. (Na przykład uniwersalny podprogram obsługi wejścia-wyjścia wywołuje różne podprogramy do obsługi różnych urządzeń, ale nie wszystkie te podprogramy są wywoływanie podczas pracy konkretnego programu głównego. Tego typu problemy są zmorą systemów automatycznego generowania kodu).

10. Dla (i) mamy trzy przypadki: $x = y$; $x \subset y$ i $y = z$; $x \subset y$ i $y \subset z$. Dla (ii) dwa: $x = y$; $x \neq y$, oba są proste, (iii) z resztą też.

11. 52 możliwe rozwiązania można obejrzeć po „wymnożeniu” wyrażenia: $13749(25 + 52)86 + (1379 + 1397 + 1937 + 9137)(4258 + 4528 + 2458 + 5428 + 2548 + 5248 + 2584 + 5284)6 + (1392 + 1932 + 1923 + 9123 + 9132 + 9213)7(458 + 548 + 584)6$.

12. Na przykład: (a) Wypisz wszystkie zbiory zawierające k elementów (w dowolnym porządku) przed wypisaniem jakiegokolwiek zbioru o $k + 1$ elementach, $0 \leq k < n$.

(b) Każdemu podzbiorowi można przypisać ciąg zer i jedynek wskazujący, które elementy w podzbiorze są, a których nie ma. To wyznacza odpowiedniość między podzbiorami a liczbami całkowitymi od 0 do $2^n - 1$ (bo ciągi można traktować jak liczby w systemie dwójkowym). Naturalny porządek na liczbach wyznacza porządek topologiczny na podzbiorach.

13. Sha i Kleitman, *Discrete Math.* **63** (1987), 271–278, udowodnili, że nie na więcej niż $\prod_{k=0}^n \binom{n}{k}$. To $2^{2^n + O(n)}$ razy więcej niż oczywiste ograniczenie dolne $\prod_{k=0}^n \binom{n}{k}! = 2^{2^n(n+O(\log n))}$; autorzy przypuszczają, że ograniczenie dolne jest bliższe prawdy.

14. Jeśli $a_1 a_2 \dots a_n$ i $b_1 b_2 \dots b_n$ są dwoma różnymi wynikami sortowania topologicznego, to niech j będzie najmniejsze, takie że $a_j \neq b_j$. Wówczas $a_k = b_j$ i $a_j = b_m$ dla pewnych $k, m > j$. Stąd $b_j \not\leq a_j$, ponieważ $k > j$, a $a_j \not\leq b_j$, ponieważ $m > j$, zatem (iv) nie jest spełnione. W drugą stronę, jeżeli istnieje tylko jeden możliwy wynik sortowania topologicznego $a_1 a_2 \dots a_n$, to musi być $a_j \preceq a_{j+1}$ dla $1 \leq j < n$, ponieważ w przeciwnym razie moglibyśmy zamienić miejscami a_j i a_{j+1} . Stąd i z przechodniości wynika (iv).

Uwaga: Poniższy dowód jest poprawny także dla zbiorów nieskończonych. (a) Każdy porządek częściowy można zanurzyć w porządku liniowym. Jeśli weźmiemy dwa elementy, takie że $x_0 \not\leq y_0$ i $y_0 \not\leq x_0$, to możemy utworzyć inny porządek częściowy. Jest nim na przykład relacja „ $x \preceq y$ lub $x \preceq x_0$ i $y_0 \preceq y$ ”. Ten nowy porządek „zawiera” porządek wyjściowy, a ponadto zachodzi dla niego $x_0 \preceq y_0$. Teraz wystarczy zastosować lemat Zorna* lub indukcję pozaskończoną. (b) Oczywiście porządku liniowego nie da się zanurzyć w innym porządku liniowym. (c) Porządek częściowy mający elementy nieporównywalne x_0 i y_0 jak w (a) można rozszerzyć do dwóch porządków liniowych, takich że w jednym $x_0 \preceq y_0$, a w drugim $y_0 \preceq x_0$, zatem istnieją przynajmniej dwa różne porządki liniowe.

15. Jeżeli zbiór S jest skończony, to możemy wypisać wszystkie pary $a \prec b$ prawdziwe w danej relacji porządku częściowego. Następnie możemy usuwać para po parze te z nich, które wynikają z pozostałych. Otrzymamy w ten sposób zbiór, w którym nie ma „nadmiarowych” par. Problem polega na pokazaniu, że jest tylko jeden taki zbiór i że zawsze do niego dojdziemy niezależnie od tego, w jakiej kolejności będziemy usuwać pary. Jeżeli istniałyby dwa nienadmiarowe zbiory U i V , takie że para „ $a \prec b$ ” występuje w U , ale nie w V , to w V musi być $k+1$ par $a \prec c_1 \prec \dots \prec c_k \prec b$ dla pewnego $k \geq 1$. Ale z U można wywnioskować $a \prec c_1$ i $c_1 \prec b$, nie korzystając z $a \prec b$ (ponieważ $b \not\leq c_1$ i $c_1 \not\leq a$), zatem para $a \prec b$ jest nadmiarowa w U .

Twierdzenie nie zachodzi dla nieskończonych zbiorów S , jeżeli istnieje co najwyżej jeden nienadmiarowy zbiór par. Na przykład gdy S jest zbiorem liczb całkowitych z elementem ∞ i zdefiniujemy $n \prec n+1$ i $n \prec \infty$ dla wszystkich n , to nie istnieje nienadmiarowy zbiór par wyznaczających ten porządek częściowy.

16. Niech $x_{p_1} x_{p_2} \dots x_{p_n}$ będzie topologicznym porządkiem na zbiorze S ; wystarczy zastosować permutację $p_1 p_2 \dots p_n$ do wierszy i do kolumn.

17. Jeżeli w kroku T4 k rośnie od 1 do n , to dostaniemy 1932745860. Jeżeli k maleje od n do 1, jak w programie T, to dostaniemy 9123745860.

18. Łączą elementy w porządku topologicznym: najpierw `QLINK[0]`, następnie element `QLINK[QLINK[0]]` itd.; `QLINK[ostatni] = 0`.

* W literaturze polskiej spotyka się nazwę lemat Kuratowskiego-Zorna (przyp. tłum.).

19. W pewnych przypadkach spowoduje to błąd. Jeśli w kroku T5 kolejka zawiera tylko jeden element, to zmodyfikowany algorytm przypisze $F = 0$ (tym samym opróżniając kolejkę), ale w kroku T6 do kolejki mogą zostać włożone inne elementy. Proponowana modyfikacja wymaga wprowadzenia dodatkowego sprawdzenia, czy $F = 0$ w kroku T6.

20. Rzeczywiście, można użyć stosu w następujący sposób. (Znika krok T7).

T4. Przyjmij $T \leftarrow 0$. Dla $1 \leq k \leq n$, jeśli $COUNT[k]$ jest zerem, wykonaj operację $SLINK[k] \leftarrow T$, $T \leftarrow k$. ($SLINK[k] \equiv QLINK[k]$).

T5. Wyprowadź wartość T . Jeśli $T = 0$, to idź do T8; w przeciwnym razie przyjmij $N \leftarrow N - 1$, $P \leftarrow TOP[T]$, $T \leftarrow SLINK[T]$.

T6. Jak poprzednio, tyle że przechodzimy do T5, a nie do T7. Ponadto gdy $COUNT[SUC(P)]$ zmala je do zera, przypisujemy $SLINK[SUC(P)] \leftarrow T$ oraz $T \leftarrow SUC(P)$.

21. Powtórzone pary nieco spowalniają wykonanie algorytmu i powodują większe zużycie pamięci. Relacja „ $j \prec j'$ zostanie potraktowana jak pętla (strzałka od kwadracika j do niego samego), co narusza warunek częściowego porządku.

22. By uczynić program „odpornym na błędy” powinniśmy (a) sprawdzać, czy $0 < n <$ pewne maksimum; (b) dla każdej pary $j \prec k$ sprawdzać warunek $0 < j, k \leq n$; (c) upewnić się, że liczba par nie spowoduje przekroczenia pojemności sterty.

23. Na końcu kroku T5 należy dodać „ $TOP[F] \leftarrow \Lambda$ ”. (Wtedy zawsze $TOP[1], \dots, TOP[n]$ wskazują na nieusunięte pary). W kroku T8, jeśli $N > 0$, wypisujemy „LOOP DETECTED IN INPUT” i przypisujemy $QLINK[k] \leftarrow 0$ dla $1 \leq k \leq n$. Trzeba jeszcze dodać następujące kroki:

T9. Dla $1 \leq k \leq n$, przyjmij $P \leftarrow TOP[k]$, $TOP[k] \leftarrow 0$ i wykonaj krok T10. (W ten sposób $QLINK[j]$ wskazuje na jeden z poprzedników obiektu j , dla wszystkich nie wypisanych j). Przejdz do kroku T11.

T10. Jeśli $P \neq \Lambda$, to przyjmij $QLINK[SUC(P)] \leftarrow k$, $P \leftarrow NEXT(P)$ i powtórz ten krok.

T11. Znajdź takie k , że $QLINK[k] \neq 0$.

T12. Przyjmij $TOP[k] \leftarrow 1$ i $k \leftarrow QLINK[k]$. Jeśli $TOP[k] = 0$, to powtórz ten krok.

T13. (Znaleźliśmy początek pętli). Wypisz wartość k , przypisz $TOP[k] \leftarrow 0$ oraz $k \leftarrow QLINK[k]$ i jeśli $TOP[k] = 1$, to powtórz ten krok.

T14. Wypisz wartość k (początek i koniec pętli) i zatrzymaj algorytm. (Uwaga: Pętla została wypisana „wstecz”; jeżeli chcemy wypisywać pętlę w innej kolejności, powinniśmy między krokami T12 i T13 zastosować na przykład taki algorytm jak w ćwiczeniu 7). ■

24. Wstaw trzy wiersze:

```
08a PRINTER EQU 18
14a           ST6 NO
59a           STZ X,1(TOP)      TOP[F] ← Λ.
```

Zastąp wiersze 74–75 wierszami:

```
74           J6Z DONE
75           OUT LINE1(PRINTER)  Wypisz nagłówek.
76           LD6 NO
```

```

77      STZ X,6(QLINK)    QLINK[k] ← 0.
78      DEC6 1
79      J6P *-2           n ≥ k ≥ 1.
80      LD6 NO
81 T9     LD2 X,6(TOP)   P ← TOP[k].
82      STZ X,6(TOP)   TOP[k] ← 0.
83      J2Z T9A          Czy P = Λ?
84 T10    LD1 0,2(SUC)   rI1 ← SUC(P).
85      ST6 X,1(QLINK)  QLINK[rI1] ← k.
86      LD2 0,2(NEXT)   P ← NEXT(P).
87      J2P T10          Czy P ≠ Λ?
88 T9A    DEC6 1
89      J6P T9           n ≥ k ≥ 1.
90 T11    INC6 1
91      LDA X,6(QLINK)
92      JAZ *-2          Znajdź takie k, że QLINK[k] ≠ 0.
93 T12    ST6 X,6(TOP)  TOP[k] ← k.
94      LD6 X,6(QLINK)  k ← QLINK[k].
95      LD1 X,6(TOP)
96      J1Z T12          Czy TOP[k] = 0?
97 T13    ENTA 0,6
98      CHAR             Zamień k na znak.
99      JBUS *(PRINTER)
100     STX VALUE        Drukuj.
101     OUT LINE2(PRINTER)
102     J1Z DONE          Skończ, gdy TOP[k] = 0.
103     STZ X,6(TOP)
104     LD6 X,6(QLINK)  TOP[k] ← 0.
105     LD1 X,6(TOP)    k ← QLINK[k].
106     JMP T13
107 LINE1  ALF LOOP     Nagłówek.
108     ALF DETEC
109     ALF TED I
110     ALF N INP
111     ALF UT:
112 LINE2  ALF           Kolejne wiersze.
113 VALUE  EQU LINE2+3
114     ORIG LINE2+24
115 DONE   HLT           Stop.
116 X      END TOPSORT  ■

```

Uwaga: Jeśli do (18) dodamy 9 ← 1 i 6 ← 9, to powyższy program wpisze cykl „9, 6, 8, 5, 9”.

26. Jedno rozwiązanie polega na wykonaniu dwóch faz:

Faza 1. (Korzystamy ze struktury X jak ze stosu (sekwencyjnego), zaznaczając za pomocą B = 1 lub 2 programy do załadowania).

A0. Dla $1 \leq J \leq N$ przypisz $B(X[J]) \leftarrow B(X[J]) + 2$, if $B(X[J]) \leq 0$.

A1. Jeśli $N = 0$, przejdź do fazy 2; w przeciwnym razie przyjmij $P \leftarrow X[N]$ i zmniejsz N o 1.

- A2.** Jeśli $|B(P)| = 1$, to idź do A1, w przeciwnym razie przyjmij $P \leftarrow P + 1$.
- A3.** Jeśli $B(SUB1(P)) \leq 0$, to przyjmij $N \leftarrow N + 1$, $B(SUB1(P)) \leftarrow B(SUB1(P)) + 2$, $X[N] \leftarrow SUB1(P)$. Jeśli $SUB2(P) \neq 0$ i $B(SUB2(P)) \leq 0$, to postępuj podobnie z $SUB2(P)$. Idź do A2. ■

Faza 2. (Przeglądamy strukturę i przydzielamy pamięć).

B1. Przyjmij $P \leftarrow FIRST$.

B2. Jeśli $P = \Lambda$, to przyjmij $N \leftarrow N + 1$, $BASE(LOC(X[N])) \leftarrow MLOC$, $SUB(LOC(X[N])) \leftarrow 0$ i zakończ algorytm.

B3. Jeśli $B(P) > 0$, to przyjmij $N \leftarrow N + 1$, $BASE(LOC(X[N])) \leftarrow MLOC$, $SUB(LOC(X[N])) \leftarrow P$, $MLOC \leftarrow MLOC + SPACE(P)$.

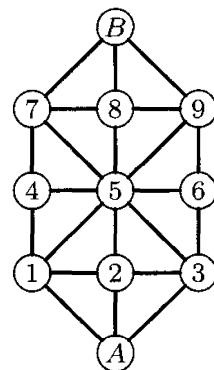
B4. $P \leftarrow LINK(P)$ i wróć od B2. ■

27. Skomentowanie poniższego kodu pozostawiamy Czytelnikowi.

B	EQU	0:1	A1	J1Z	B1		INC1	1
SPACE	EQU	2:3		LD2	X,1		INCA	2
LINK	EQU	4:5		DEC1	1		STA	0,3(B)
SUB1	EQU	2:3	A2	LDA	0,2(1:1)		ST3	X,1
SUB2	EQU	4:5		DECA	1		JMP	A2
BASE	EQU	0:3		JAZ	A1	B1	ENT2	FIRST
SUB	EQU	4:5		INC2	1		LDA	MLOC
A0	LD2	N	A3	LD3	0,2(SUB1)		JMP	1F
	J2Z	B1		LDA	0,3(B)	B3	LDX	0,2(B)
1H	LD3	X,2		JAP	9F		JXNP	B4
	LDA	0,3(B)		INC1	1		INC1	1
	JAP	*+3		INCA	2		ST2	X,1(SUB)
	INCA	2		STA	0,3(B)		ADD	0,2(SPACE)
	STA	0,3(B)		ST3	X,1	1H	STA	X+1,1(BASE)
	DEC2	1	9H	LD3	0,2(SUB2)	B4	LD2	0,2(LINK)
	J2P	1B		J3Z	A2	B2	J2NZ	B3
	LD1	N		LDA	0,3(B)		STZ	X+1,1(SUB)
				JAP	A2			■

28. Podajemy tu jedynie garść komentarzy dotyczących gry wojennej. Niech A oznacza gracza poruszającego się trzema pionkami, ustawionymi początkowo na polach A13; niech B oznacza drugiego gracza. Gra polega na tym, że A musi „zablokować” B , a B wygrywa, jeśli doprowadzi do powtórnego wystąpienia jakiegoś ustawienia pionków na planszy. By uniknąć traktowania całej historii rozgrywki jako własności sytuacji (co pociągałoby za sobą konieczność rozpatrywania o wiele większej liczby sytuacji), musimy nieco zmodyfikować algorytm. Zaczynamy od oznaczenia ustawień 157–4, 789–B, 359–6 przy ruchu gracza B jako „przegranych” i stosujemy opisany algorytm. Gracz A powinien wykonywać ruchy prowadzące do sytuacji przegranych dla B . Ale A musi uważać także na to, by nie powtórzyć któregoś z poprzednich ustawień. Dobry program powinien korzystać z generatora liczb losowych do wybrania jednego z wygrywających posunięć, stąd oczywistą techniką dla programu wcielającą się w gracza A wydaje się losowy wybór spomiędzy ruchów prowadzących do sytuacji przegranych dla B . Można jednak pokazać partię, w której ta procedura zawiedzie. Rozważmy na przykład ustawienie 258–7, gdy ruch ma wykonać gracz A . Jest to

sytuacja wygrana (dla B). A może próbować ruszyć się na 158–7 (która według obliczeń algorytmu jest sytuacją przegrana dla B), ale B rusza się wówczas na 158–B, a to zmusza A do zagrania 258–B, po czym B rusza się z powrotem na 258–7. B wygrał, bo powtórzyło się jedno z poprzednich ustawięń. Ten przykład pokazuje, że algorytm trzeba każdorazowo zastosować po wykonaniu ruchu, poczynając od każdej sytuacji, która poprzednio została oznaczona jako „przegrana” (jeżeli ma się ruszyć gracz A) lub „wygrana” (jeżeli ma się ruszyć gracz B). Gra wojenna jest bardzo wdzięcznym tematem dla programu demonstracyjnego.



Plansza do „gry wojennej”.

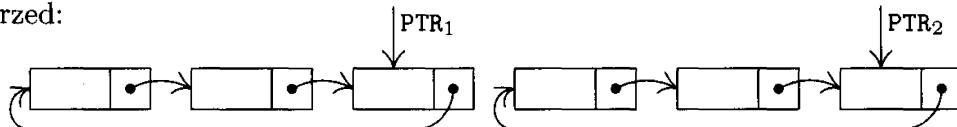
29. (a) Jeśli $\text{FIRST} = \Lambda$, to nie rób nic; w przeciwnym razie przyjmij $P \leftarrow \text{FIRST}$, a następnie zero lub więcej razy powtarzaj $P \leftarrow \text{LINK}(P)$, aż $\text{LINK}(P) = \Lambda$. Na koniec przyjmij $\text{LINK}(P) \leftarrow \text{AVAIL}$ i $\text{AVAIL} \leftarrow \text{FIRST}$ (i zapewne jeszcze $\text{FIRST} \leftarrow \Lambda$). (b) Jeśli $F = \Lambda$, to nie rób nic; w przeciwnym razie przyjmij $\text{LINK}(R) \leftarrow \text{AVAIL}$ i $\text{AVAIL} \leftarrow F$ (i zapewne jeszcze $F \leftarrow \Lambda$, $R \leftarrow \text{LOC}(F)$).

30. Wstawianie: $P \leftarrow \text{AVAIL}$, $\text{INFO}(P) \leftarrow Y$, $\text{LINK}(P) \leftarrow \Lambda$, jeśli $F = \Lambda$, to $F \leftarrow P$, w przeciwnym razie przyjmij $\text{LINK}(R) \leftarrow P$ i $R \leftarrow P$. Usuwanie: wykonaj operację (9), ale zamiast T weź F . (Choć wygodnie jest pozostawiać R niezdefiniowane przy pustej kolejce, taki brak dyscypliny może wprowadzić w błąd algorytm odśmiecania, jak w ćwiczeniu 6).

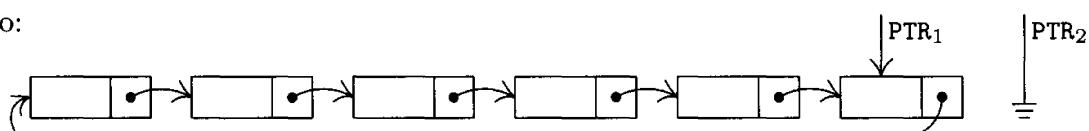
2.2.4

1. Nie upraszcza, a wprost utrudnia. (Podana konwencja *nie* jest spójna z filozofią listy cyklicznej, chyba że umieścimy $\text{NODE}(\text{LOC}(\text{PTR}))$ na liście jako jej początek).

2. Przed:



Po:

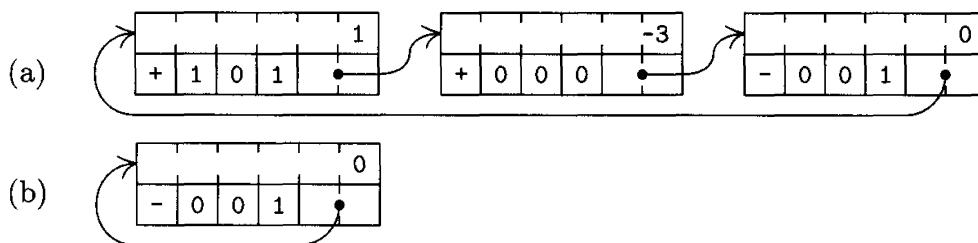


3. Jeśli $\text{PTR}_1 = \text{PTR}_2$, to jedynym skutkiem jest $\text{PTR}_2 \leftarrow \Lambda$. Jeśli $\text{PTR}_1 \neq \text{PTR}_2$, to zamiana dowiązań powoduje rozbicie listy na dwie części (można ją porównać do obręczy przeciętej w dwóch miejscach – rozpada się ona na dwie części). Druga część operacji sprawia, że PTR_1 wskazuje na listę cykliczną złożoną z tych elementów, które odwiedziliśmy, przechodząc pierwotną listę od elementu wskazywanego przez PTR_1 do elementu wskazywanego przez PTR_2 .

4. Niech HEAD będzie adresem głowy listy. Włożenie Y na stos: $P \leftarrow \text{AVAIL}$, $\text{INFO}(P) \leftarrow Y$, $\text{LINK}(P) \leftarrow \text{LINK}(\text{HEAD})$, $\text{LINK}(\text{HEAD}) \leftarrow P$. Zdjęcie elementu ze stosu do Y : jeśli dowiązanie $\text{LINK}(\text{HEAD}) = \text{HEAD}$, to **NIEDOMIAR**; w przeciwnym razie $P \leftarrow \text{LINK}(\text{HEAD})$, $\text{LINK}(\text{HEAD}) \leftarrow \text{LINK}(P)$, $Y \leftarrow \text{INFO}(P)$, $\text{AVAIL} \leftarrow P$.

5. (Porównaj z ćwiczeniem 2.2.3–7). Przyjmij $Q \leftarrow \Lambda$, $P \leftarrow \text{PTR}$, a następnie dopóki $P \neq \Lambda$ przypisuj $R \leftarrow Q$, $Q \leftarrow P$, $P \leftarrow \text{LINK}(Q)$, $\text{LINK}(Q) \leftarrow R$. (Po zakończeniu $Q = \text{PTR}$).

6.



7. Dzięki temu wyrazy podobne można znaleźć w jednym przebiegu listy, zatem unika się wielokrotnego przeszukiwania listy. Jednakże porządek *rosnący* nie pasowałby do wartości wartownika „-1”.

8. Musimy wiedzieć, który element zawiera dowiązanie do aktualnie przetwarzanego elementu, na wypadek gdybyśmy chcieli go skasować lub wstawić przed nim nowy element. Można jednak inaczej. Moglibyśmy kasować **NODE(Q)**, przypisując $Q_2 \leftarrow \text{LINK}(Q)$, a następnie $\text{NODE}(Q) \leftarrow \text{NODE}(Q_2)$, $\text{AVAIL} \leftarrow Q_2$. Wstawianie **NODE(Q2)** przed **NODE(Q)** można zrealizować, wymieniając najpierw $\text{NODE}(Q_2) \leftrightarrow \text{NODE}(Q)$, a następnie przypisując $\text{LINK}(Q) \leftarrow Q_2$, $Q \leftarrow Q_2$. Te chytre sztuczki pozwalają na usuwanie i wstawianie elementów w sytuacji, gdy *nie* wiemy, który element zawiera dowiązanie do **NODE(Q)**. Takie metody były wykorzystywane we wcześniejszych wersjach IPL. Jest jednak problem: w wyniku ich stosowania wartownik z końca listy może się przemieścić, a w programie mogą istnieć zmienne wskazujące na jego pierwotną lokację.

9. Dla $P = Q$ algorytm A po prostu mnoży wielomian $\text{polynomial}(Q)$ przez dwa, tak jak powinien, poza przypadkiem gdy dla pewnego wyrazu o $ABC \geq 0$ jest $\text{COEF} = 0$, kiedy fatalnie zawodzi. Algorytm M wykonany dla $P = M$ też daje spodziewany wynik. Algorytm M dla $P = Q$ sprawia, że $\text{polynomial}(P) \leftarrow \text{polynomial}(P)$ razy $(1 + t_1)(1 + t_2) \dots (1 + t_k)$, jeśli $M = t_1 + t_2 + \dots + t_k$ (choćż to nie jest bardzo oczywiste). Gdy $M = Q$, algorytm M niespodziewanie daje oczekiwany wynik, $\text{polynomial}(Q) \leftarrow \text{polynomial}(Q) + \text{polynomial}(Q) \times \text{polynomial}(P)$, z tym że pojawiają się kłopoty, gdy wolnym wyrazem wielomianu $\text{polynomial}(P)$ jest -1 .

10. Żadne. (Jedyna ewentualna zmiana może dotyczyć kroku M2 i polegać na usunięciu testów, czy poszczególne wartości A, B lub C nie przekroczyły dopuszczalnego zakresu; tych testów nie zapisywaliśmy, ponieważ założyliśmy, że nie trzeba ich przeprowadzać). Innymi słowy, algorytmy z tego rozdziału można traktować jako operacje na wielomianach $f(x^{b^2}, x^b, x)$ zamiast $f(x, y, z)$.

11.	COPY STJ 9F	(Uzupełnienie	ST6 1,3(LINK)
	ENT3 9F	komentarzy	ENT3 0,6
	LDA 1,1	pozostawiamy	LD1 1,1(LINK)
1H	LD6 AVAIL	Czytelnikowi)	LDA 1,1
	J6Z OVERFLOW		JANN 1B
	LDX 1,6(LINK)		LD2 8F(LINK)
	STX AVAIL		ST2 1,3(LINK)
	STA 1,6		9H JMP *
	LDA 0,1		8H CON 0 ■
	STA 0,6		

12. Niech kopowany wielomian ma p wyrazów. Program A potrzebuje $(29p + 13)u$, a żeby porównanie było sprawiedliwe dodajmy jeszcze czas potrzebny na utworzenie wielomianu zerowego – powiedzmy $18u$ – zgodnie z ćwiczeniem 14. Program z ćwiczenia 11 potrzebuje $(21p + 31)u$, tj. około $\frac{3}{4}$ raza tyle.

- 13. ERASE STJ 9F**
- ```

LDX AVAIL
LDA 1,1(LINK)
STA AVAIL
STX 1,1(LINK)
9H JMP * ■

```
- 14. ZERO STJ 9F**
- ```

LD1 AVAIL
J1Z OVERFLOW
LDX 1,1(LINK)
STX AVAIL
ENT2 0,1

```
- | | |
|------------|---------------|
| MOVE 1F(2) | ST2 1,2(LINK) |
| 9H JMP * | 1H CON 0 |
| | CON -1(ABC) ■ |
- 15. MULT STJ 9F**
- | | |
|---|------------------|
| Wejście do podprogramu. | LDA 5F |
| Zmień ustawienia znaczników SW. | STA SW1 |
| | LDA 6F |
| | STA SW2 |
| | STA SW3 |
| | JMP **+2 |
| M2. Cykl mnożenia. | 2H JMP ADD |
| <u>M1. Następny mnożnik.</u> M ← LINK(M). | 1H LD4 1,4(LINK) |
| Idź do M2, jeśli ABC(M) ≥ 0. | LDA 1,4 |
| Przywróć ustawienia znaczników SW. | JANN 2B |
| | 8H LDA 7F |
| | STA SW1 |
| | LDA 8F |
| | STA SW2 |
| | STA SW3 |
| Powrót. | 9H JMP * |
| Nowe ustawienie SW1. | 5H JMP **+1 |
| rX ← COEF(P) × COEF(M). | LDA 0,1 |
| ABC(P). | MUL 0,4 |
| ABC(M), if ABC(P) ≥ 0. | LDA 1,1(ABC) |
| Przesuń w pole 0:3 rA. | JAN **+2 |
| Zachowaj rX do wykorzystania w SW2 u SW3. | ADD 1,4(ABC) |
| | SLA 2 |
| | STX OF |
| | JMP SW1+1 |
| Nowe ustawienia SW2 i SW3. | 6H LDA OF |
| Normalne ustawienie SW1. | 7H LDA 1,1 |
| Normalne ustawienia SW2 i SW3. | 8H LDA 0,1 |
| Pamięć tymczasowa. ■ | OH CON 0 |

16. Niech r będzie liczbą wyrazów w wielomianie $\text{polynomial}(M)$. Podprogram potrzebuje $21pr + 38r + 29 + 27 \sum m' + 18 \sum m'' + 27 \sum p' + 8 \sum q'$ jednostek czasu, gdzie sumowanie dotyczy odpowiednich wartości podczas r wywołań programu A. Liczba wyrazów w $\text{polynomial}(Q)$ wzrasta o $p' - m'$ przy każdym wywołaniu programu A. Jeżeli poczynimy uzasadnione założenie, że $m' = 0$ i $p' = \alpha p$, gdzie $0 < \alpha < 1$, to odpowiednie sumy są równe 0, $(1 - \alpha)pr$, αpr i $rq'_0 + \alpha p(r(r - 1)/2)$, gdzie q'_0 jest wartością q'

w pierwszej iteracji. Wartość całkowita wynosi $4\alpha pr^2 + 40pr + 4\alpha pr + 8q'_0r + 38r + 29$. Z powyższej analizy wynika, że mnożnik powinien mieć mniej wyrazów niż mnożna, ponieważ częściej musimy pomijać wyrazy w $\text{polynomial}(Q)$. (Szybszy algorytm rozważamy w ćwiczeniu 5.2.3–29).

17. Jest tylko jeden drobny powód. Podprogramy dodawania i mnożenia byłyby dla list liniowych w zasadzie identyczne. Lepsza wydajność podprogramu **ERASE** przy reprezentacji cyklicznej (ćwiczenie 13) jest jedynym istotnym powodem.

18. Niech w elemencie x_i pole z dowiązaniem zawiera $\text{LOC}(x_{i+1}) \oplus \text{LOC}(x_{i-1})$, gdzie „ \oplus ” oznacza „bitową różnicę symetryczną”. Można zastosować także inne odwracalne operacje, jak dodawanie lub odejmowanie modulo zakres wartości wskaźnikowych. Na takiej liście warto przechowywać dwie przylegające do siebie głowy, by uniknąć problemów z inicjowaniem. (Pochodzenie tej pomysłowej metody jest nieznane).

2.2.5

1. Wstaw Y na lewy koniec: $P \Leftarrow \text{AVAIL}$; $\text{INFO}(P) \leftarrow Y$; $\text{LLINK}(P) \leftarrow \Lambda$; $\text{RLINK}(P) \leftarrow \text{LEFT}$; jeśli $\text{LEFT} \neq \Lambda$, to $\text{LLINK}(\text{LEFT}) \leftarrow P$, w przeciwnym razie $\text{RIGHT} \leftarrow P$; $\text{LEFT} \leftarrow P$. Przypisz do Y element najdalszy z lewej i usuń: jeśli $\text{LEFT} = \Lambda$, to **NIEDOMIAR**; $P \leftarrow \text{LEFT}$; $\text{LEFT} \leftarrow \text{RLINK}(P)$; jeśli $\text{LEFT} = \Lambda$, to $\text{RIGHT} \leftarrow \Lambda$, w przeciwnym razie $\text{LLINK}(\text{LEFT}) \leftarrow \Lambda$; $Y \leftarrow \text{INFO}(P)$; $\text{AVAIL} \Leftarrow P$.

2. Rozważmy przypadek kilku kolejnych operacji usuwania elementu z tego samego końca. Po każdym usunięciu musimy wiedzieć, który element usunąć w następnej kolejności, zatem dowiązania w liście muszą prowadzić w kierunku drugiego końca. Usuwanie z obu stron wymaga zatem, by dowiązania prowadziły w obu kierunkach. Z drugiej strony ćwiczenie 2.2.4–18 pokazuje, w jaki sposób reprezentować dwa dowiązania w jednym polu; takie rozwiązanie umożliwia zaimplementowanie operacji na nieograniczonej kolejce dwustronnej.

3. By pokazać niezależność **CALLUP** i **CALLDOWN**, zauważmy że w przykładzie pokazanym w tabeli 1 winda nie zatrzymuje się na kondygmacjach 2 i 3 w czasie 0393–0444, chociaż czekają tam pasażerowie. Ci pasażerowie nacisnęli **CALLDOWN**, bo gdyby nacisnęli **CALLUP**, to winda by się zatrzymała.

By pokazać niezależność **CALLCAR** od pozostałych przycisków, zauważmy, że z tabeli 1 wynika, że gdy w chwili 1378 drzwi zaczynają się otwierać, winda już postanowiła jechać w górę (**GOINGUP**). Stanem windy byłoby w tym momencie **NEUTRAL**, jeżeli $\text{CALLCAR}[1] = \text{CALLCAR}[2] = \text{CALLCAR}[3] = \text{CALLCAR}[4] = 0$ zgodnie z krokiem E2, ale pasażerowie 7 i 9 nacisnęli **CALLCAR[2]** i **CALLCAR[3]**. (Jeśli wyobrażymy sobie tę samą sytuację z numerami pięter zwiększonimi o 1, to fakt **STATE = NEUTRAL** lub **STATE = GOINGUP** w chwili, gdy drzwi są otwarte, miałby wpływ na to, czy winda ewentualnie będzie dalej zjeżdżać w dół, czy bezwarunkowo pojedzie na górę).

4. Jeśli dwunastu lub więcej pasażerów wysiadałoby na tym samym piętrze, to winda mogłaby przez cały ten czas być w stanie **STATE=NEUTRAL**. Gdy w kroku E9 wywołano by więc **DECISION**, podprogram mógłby wprowadzić windę w nowy stan, zanim ktokolwiek wsiadłby na tym piętrze. To rzeczywiście rzadko się zdarza (i bez wątpienia było najbardziej zagadkowym zjawiskiem zaobserwowanym przez autora podczas eksperymentów z windą).

5. Stan od momentu rozpoczęcia otwierania drzwi w chwili 1063 do momentu zabraania pasażera 7 w chwili 1183 byłby stanem **NEUTRAL**, ponieważ nie byłoby wezwań na kondygamację 0 i nikogo nie byłoby w windzie. Wtedy pasażer 7 spowodowałby $\text{CALLCAR}[2] \leftarrow 1$ i stan odpowiednio zmieniłby się na **GOINGUP**.

6. Należy dodać warunek „jeśli OUT < IN, to STATE \neq GOINGUP; jeśli OUT > IN, to STATE \neq GOINGDOWN” do warunku „FLOOR = IN” w krokach U2 i U4. W kroku E4 należy wpuszczać wyłącznie tych pasażerów z QUEUE[FLOOR], którzy chcą jechać w kierunku zgodnym z aktualnym kierunkiem ruchu kabiny, chyba że STATE = NEUTRAL (kiedy to wpuszczamy wszystkich chętnych).

[Wydział Matematyki Stanford University ma taką windę, ale pasażerowie praktycznie nie zwracają uwagi na dodatkową sygnalizację. Ludzie starają się dostać do kabiny jak najszybciej, niezależnie od kierunku jazdy. Dlaczego projektanci wind nie zdają sobie z tego sprawy i nie projektują sterowników wind tak, by zerowały jednocześnie CALLUP i CALLDOWN? Proces byłby szybszy, bo winda nie zatrzymywałaby się tak często].

7. W wierszu 227 zakładamy, że pasażer jest na liście WAIT. Skok do U4A gwarantuje prawdziwość tego założenia. Zakładamy, że GIVEUPTIME jest wartością dodatnią; w praktyce jest to zapewne przynajmniej 100.

8. Uzupełnienie komentarzy pozostawiamy Czytelnikowi.

277	E8	DEC4 1	
278		ENTA 61	
279		JMP HOLD	
280		LDA CALL,4(3:5)	
281		JAP 1F	
282		ENT1 -2,4	
283		J1Z 2F	
284		LDA CALL,4(1:1)	
285		JAZ E8	
286	2H	LDA CALL-1,4	
287		ADD CALL-2,4	
288		ADD CALL-3,4	
289		ADD CALL-4,4	
290		JANZ E8	
291	1H	ENTA 23	
292		JMP E2A	
9. 01	DECISION	STJ 9F	Ustaw adres powrotu.
02		J5NZ 9F	<u>D1. Czy trzeba decydować?</u>
03		LDX ELEV1+2(NEXTINST)	
04		DECX E1	<u>D2. Czy trzeba otworzyć drzwi?</u>
05		JXNZ 1F	Skok, jeśli winda nie jest w E1.
06		LDA CALL+2	
07		ENT3 E3	Przygotuj zlecenie E3, jeśli jest wezwanie na kondygnację 2.
08		JANZ 8F	
09	1H	ENT1 -4	<u>D3. Czy są jakieś wezwania?</u>
10		LDA CALL+4,1	Szukaj niezerowej zmiennej CALL.
11		JANZ 2F	
12	1H	INC1 1	rI1 \equiv j - 4.
13		J1NP *-3	
14		LDA 9F(0:2)	Wszystkie CALL[j], j \neq FLOOR, są zerami.
15		DECA E6B	Czy lokacja wyjściowa = wiersz 250?
16		JANZ 9F	
17		ENT1 -2	j \leftarrow 2.
18	2H	ENT5 4,1	<u>D4. Ustawienie STATE.</u>

19	DEC5 0,4	STATE $\leftarrow j - \text{FLOOR}.$
20	J5NZ *+2	
21	JANZ 1B	$j = \text{FLOOR}$ nie może wystąpić.
22	JXNZ 9F	<u>D5. Czy winda w spoczynku?</u>
23	J5Z 9F	Skocz, jeśli nie krok E1 lub jeśli $j = 2$.
24	ENT3 E6	W przeciwnym razie zleć E6.
25 8H	ENTA 20	Czekaj 20 jednostek czasu.
26	ST6 8F(0:2)	Przechowaj rI6.
27	ENT6 ELEV1	
28	ST3 2,6(NEXTINST)	Ustaw NEXTINST na E3 lub E6.
29	JMP HOLD	Zleć wykonanie.
30 8H	ENT6 *	Odtwórz rI6.
31 9H	JMP *	Wyjście z podprogramu.

11. Początkowo niech $\text{LINK}[k] = 0$, $1 \leq k \leq n$, i $\text{HEAD} = -1$. Podczas kroku symulacji powodującego modyfikację $V[k]$ sygnalizujemy błąd, jeśli $\text{LINK}[k] \neq 0$; w przeciwnym razie $\text{LINK}[k] \leftarrow \text{HEAD}$, $\text{HEAD} \leftarrow k$ oraz przypisujemy do $\text{NEWV}[k]$ nową wartość $V[k]$. Po każdym kroku symulacji przyjmujemy $k \leftarrow \text{HEAD}$, $\text{HEAD} \leftarrow -1$ i zero lub więcej razy powtarzamy następujące operacje, dopóki nie osiągniemy $k < 0$: $V[k] \leftarrow \text{NEWV}[k]$, $t \leftarrow \text{LINK}[k]$, $\text{LINK}[k] \leftarrow 0$, $k \leftarrow t$.

W oczywisty sposób tę metodę da się zastosować w przypadku zmiennych rozrzuconych po pamięci, jeżeli do każdego elementu związanego ze zmienną V dołożymy pola NEWV i LINK .

12. Elementy z listy WAIT usuwamy od lewej do prawej, ale wstawiane elementy są wsortowywane od prawej do lewej (ponieważ przy takim sposobie wstawiania przeszukiwanie ma szansę trwać krócej). Zdarza się także usuwać elementy ze wszystkich trzech list wtedy, gdy nie znamy poprzednika ani następnika usuwanego elementu. Jedynie lista ELEVATOR mogłaby zostać zrealizowana jako lista jednokierunkowa bez dużych strat na wydajności.

Uwaga: Listę WAIT w symulatorze dyskretnym można realizować za pomocą listy nieliniowej, by zmniejszyć czas wsortowywania nowych elementów. W punkcie 5.2.3 są omówione ogólne problemy utrzymywania kolejek priorytetowych. Znanych jest kilka sposobów realizacji operacji usuwania i wstawiania elementu do struktury wymagających tylko $O(\log n)$ operacji, gdzie n jest liczbą elementów w strukturze. Jednak nie ma sensu stosować takich wyszukanych metod, gdy wiadomo, że n jest małe.

2.2.6

1. (Indeksujemy od 1 do n , nie od 0 do n jak w (6)). $\text{LOC}(\mathbf{A}[J,K]) = \text{LOC}(\mathbf{A}[0,0]) + 2nJ + 2K$, gdzie $\mathbf{A}[0,0]$ jest hipotetycznym elementem, który w rzeczywistości nie istnieje. Przyjmując $J = K = 1$, otrzymamy $\text{LOC}(\mathbf{A}[1,1]) = \text{LOC}(\mathbf{A}[0,0]) + 2n + 2$, zatem odpowiedź można wyrazić na wiele sposobów. Fakt, że $\text{LOC}(\mathbf{A}[0,0])$ może mieć wartość ujemną, doprowadził do niejednego błędu w kompilatorach i programach ładujących.

2. $\text{LOC}(\mathbf{A}[I_1, \dots, I_k]) = \text{LOC}(\mathbf{A}[0, \dots, 0]) + \sum_{1 \leq r \leq k} a_r I_r = \text{LOC}(\mathbf{A}[l_1, \dots, l_k]) + \sum_{1 \leq r \leq k} a_r I_r - \sum_{1 \leq r \leq k} a_r l_r$, gdzie $a_r = c \prod_{r < s \leq k} (u_s - l_s + 1)$.

Uwaga: Informacje na temat uogólnienia struktur występujących w językach programowania jak C oraz prosty algorytm obliczania potrzebnych stałych znajdzie Czytelnik w: P. Deuel, *CACM* 9 (1966), 344–347.

3. $1 \leq k \leq j \leq n$ wtedy i tylko wtedy, gdy $0 \leq k-1 \leq j-1 \leq n-1$; we wszystkich wzorach wyprowadzonych dla zera jako dolnego ograniczenia zastępujemy k, j, n odpowiednio przez $k-1, j-1, n-1$.

4. $\text{LOC}(\mathbf{A}[J, K]) = \text{LOC}(\mathbf{A}[0, 0]) + nJ - J(J-1)/2 + K.$

5. Niech $A_0 = \text{LOC}(\mathbf{A}[0, 0])$. Są przynajmniej dwa rozwiązania, jeśli założymy, że J jest w $rI1$, a K jest w $rI2$. (i) „**LDA TA2,1:7**”, gdy w lokacji $TA2+j$ jest „**NOP j+1*j/2+A0,2**”; (ii) „**LDA C1,7:2**”, gdy w lokacji $C1$ jest rozkaz „**NOP TA,1:7**”, a w lokacji $TA+j$ jest „**NOP j+1*j/2+A0**”. Drugie rozwiązanie wymaga jednego cyklu więcej, ale nie wiąże tablicy z rejestrem indeksowym 2.

6. (a) $\text{LOC}(\mathbf{A}[I, J, K]) = \text{LOC}(\mathbf{A}[0, 0, 0]) + \binom{I+2}{3} + \binom{J+1}{2} + \binom{K}{1}.$

(b) $\text{LOC}(\mathbf{B}[I, J, K]) = \text{LOC}(\mathbf{B}[0, 0, 0])$

$$+ \binom{n+3}{3} - \binom{n+3-I}{3} + \binom{n+2-I}{2} - \binom{n+2-J}{2} + K - J,$$

stąd B też można wyrazić w podanej postaci.

7. Adres komórki $\text{LOC}(\mathbf{A}[I_1, \dots, I_k]) = \text{LOC}(\mathbf{A}[0, \dots, 0]) + \sum_{1 \leq r \leq k} \binom{I_r + k - r}{1 + k - r}$. Zobacz ćwiczenie 1.2.6–56.

8. (Autor rozwiązania: P. Nash) Niech $X[I, J, K]$ będzie określone dla $0 \leq I \leq n, 0 \leq J \leq n+1, 0 \leq K \leq n+2$. Możemy przyjąć $\mathbf{A}[I, J, K] = X[I, J, K]; \mathbf{B}[I, J, K] = X[J, I+1, K]; \mathbf{C}[I, J, K] = X[I, K, J+1]; \mathbf{D}[I, J, K] = X[J, K, I+2]; \mathbf{E}[I, J, K] = X[K, I+1, J+1]; \mathbf{F}[I, J, K] = X[K, J+1, I+2]$. Jest to najlepszy możliwy schemat, ponieważ umożliwia upakowanie $(n+1)(n+2)(n+3)$ elementów sześciu tablic tetrahedrycznych w kolejnych lokacjach bez nakładania się macierzy. *Dowód:* A i B pokrywają wszystkie komórki $X[i, j, k]$ dla $k = \min(i, j, k)$; C i D pokrywają wszystkie komórki dla $j = \min(i, j, k) \neq k$; E i F pokrywają wszystkie komórki dla $i = \min(i, j, k) \neq j, k$.

(Konstrukcję można uogólnić na m wymiarów, jeśli ktoś miałby ochotę upakować elementy $m!$ uogólnionych tablic tetrahedrycznych w $(n+1)(n+2)\dots(n+m)$ kolejnych lokacjach. Z każdą tablicą związymy permutację $a_1 a_2 \dots a_m$. Element tablicy przechowujemy w lokacji $X[I_{a_1} + B_1, I_{a_2} + B_2, \dots, I_{a_m} + B_m]$, gdzie $B_1 B_2 \dots B_m$ jest wektorem inwersji $a_1 a_2 \dots a_m$, zdefiniowanym w ćwiczeniu 5.1.1–7).

9. G1. Ustaw zmienne wskaźnikowe P1, P2, P3, P4, P5, P6 na lokacje pierwszych elementów list **FEMALE**, **A21**, **A22**, **A23**, **BLOND**, **BLUE**. W dalszej części zakładamy, że koniec każdej listy jest oznaczony przez Λ i że Λ ma wartość mniejszą niż wszystkie inne dowiązania. Jeśli $P6 = \Lambda$, to zatrzymaj algorytm (lista jest niestety pusta).

G2. (Poniższe akcje można wykonywać w różnej kolejności; my przyjęliśmy kolejność najpierw **EYES**, potem **HAIR**, potem **AGE**, potem **SEX**). Zero lub więcej razy przypisz $P5 \leftarrow \text{HAIR}(P5)$, aż $P5 \leq P6$. Jeśli $P5 < P6$, to przejdź do kroku G5.

G3. Wielokrotnie (jeśli trzeba) przypisz $P4 \leftarrow \text{AGE}(P4)$, aż $P4 \leq P6$. Wykonaj to samo dla P3 i P2, aż uzyskasz $P3 \leq P6$ i $P2 \leq P6$. Jeśli P4, P3, P2 są mniejsze niż P6, to idź do G5.

G4. Przypisuj $P1 \leftarrow \text{SEX}(P1)$, aż $P1 \leq P6$. Jeśli $P1 = P6$, to znaleźliśmy jedną z poszukiwanych pań, zatem należy wypisać jej adres P6. (Wiek można wyznaczyć z wartości P2, P3 i P4).

G5. Przyjmij $P6 \leftarrow \text{EYES}(P6)$. Skończ, jeżeli $P6 = \Lambda$; w przeciwnym razie powróć do G2. ■

Ten algorytm jest ciekawy, ale istnieją lepsze sposoby organizacji list ułatwiające takie wyszukiwanie.

10. Zobacz podrozdział 6.5.

11. Co najwyżej $200 + 200 + 3 \cdot 4 \cdot 200 = 2800$ słów.

12. $\text{VAL}(Q_0) = c, \text{VAL}(P_0) = b/a, \text{VAL}(P_1) = d$.

13. Wygodnie na końcu każdej listy mieć wartownika, który ma najmniejszą wartość w polu, względem którego uporządkowana jest lista. Można by użyć zwykłych list jednokierunkowych. Wystarczy pozostawić jedynie pole LEFT w $\text{BASEROW}[i]$ i pole UP w $\text{BASECOL}[j]$ oraz następująco zmodyfikować algorytm S: W S2 sprawdzamy, czy $P_0 = \Lambda$, zanim przypiszemy $J \leftarrow \text{COL}(P_0)$; jeśli tak, to przypisujemy $P_0 \leftarrow \text{LOC}(\text{BASEROW}[IO])$ i przechodzimy do S3. W kroku S3 sprawdzamy, czy $Q_0 = \Lambda$; jeśli tak, kończymy algorytm. Krok S4 powinien zostać zmieniony analogicznie do S2. W kroku S5 sprawdzamy, czy $P_1 = \Lambda$; jeśli tak, to postępujemy jak gdyby $\text{COL}(P_1) < 0$. W kroku S6 sprawdzamy, czy $UP(\text{PTR}[J]) = \Lambda$; jeśli tak, postępujemy jak gdyby w polu ROW była wartość ujemna.

Powyższe modyfikacje komplikują algorytm i nie dają żadnych oszczędności pamięci poza polami ROW i COL w atrapach (co w przypadku komputera MIX nie jest żadną oszczędnością).

14. Można najpierw powiązać te kolumny, które mają niezerowe elementy w wierszu osiowym, tak że wszystkie pozostałe kolumny mogą być pominięte, kiedy jest wykonywane osiowanie w każdym wierszu. Wiersze, w których elementy w kolumnach osiowych są równe zeru, są pomijane natychmiast.

15. Niech $rI1 \equiv \text{PIVOT}, J, rI2 \equiv P_0, rI3 \equiv Q_0, rI4 \equiv P, rI5 \equiv P_1, X; \text{LOC}(\text{BASEROW}[i]) \equiv \text{BROW} + i; \text{LOC}(\text{BASECOL}[j]) \equiv \text{BCOL} + j; \text{PTR}[j] \equiv \text{BCOL} + j(1:3)$.

01	ROW	EQU	0:3	
02	UP	EQU	4:5	
03	COL	EQU	0:3	
04	LEFT	EQU	4:5	
05	PTR	EQU	1:3	
06	PIVOTSTEP	STJ	9F	Wejście do podprogramu, $rI1 = \text{PIVOT}$.
07	S1	LD2	0,1(ROW)	<u>S1. Inicjowanie.</u>
08		ST2	I0	$I0 \leftarrow \text{ROW}(\text{PIVOT})$.
09		LD3	1,1(COL)	
10		ST3	J0	$J0 \leftarrow \text{COL}(\text{PIVOT})$.
11		LDA	=1.0=	Stała zmennopozycyjna 1.
12		FDIV	2,1	
13		STA	ALPHA	$\text{ALPHA} \leftarrow 1/\text{VAL}(\text{PIVOT})$.
14		LDA	=1.0=	
15		STA	2,1	$\text{VAL}(\text{PIVOT}) \leftarrow 1$.
16		ENT2	BROW,2	$P_0 \leftarrow \text{LOC}(\text{BASEROW}[IO])$.
17		ENT3	BCOL,3	$Q_0 \leftarrow \text{LOC}(\text{BASECOL}[J0])$.
18		JMP	S2	
19	2H	ENTA	BCOL,1	
20		STA	BCOL,1(PTR)	$\text{PTR}[J] \leftarrow \text{LOC}(\text{BASECOL}[J])$.
21		LDA	2,2	
22		FMUL	ALPHA	
23		STA	2,2	$\text{VAL}(P_0) \leftarrow \text{ALPHA} \times \text{VAL}(P_0)$.

24	S2	LD2 1,2(LEFT)	<u>S2. Obsługa wiersza osiowego.</u> $P_0 \leftarrow \text{LEFT}(P_0)$.
25		LD1 1,2(COL)	$J \leftarrow \text{COL}(P_0)$.
26		J1NN 2B	Jeśli $J \geq 0$, to obsłuż J.
27	S3	LD3 0,3(UP)	<u>S3. Znajdowanie nowego wiersza.</u> $Q_0 \leftarrow \text{UP}(Q_0)$.
28		LD4 0,3(ROW)	$rI4 \leftarrow \text{ROW}(Q_0)$.
29	9H	J4N *	Wyjdź, jeśli $rI4 < 0$.
30		CMP4 I0	
31		JE S3	Powtórz, jeśli $rI4 = I_0$.
32		ST4 I(ROW)	$I \leftarrow rI4$.
33		ENT4 BROW,4	$P \leftarrow \text{LOC}(\text{BASEROW}[I])$.
34	S4A	LD5 1,4(LEFT)	$P_1 \leftarrow \text{LEFT}(P)$.
35	S4	LD2 1,2(LEFT)	<u>S4. Znajdowanie nowej kolumny.</u> $P_0 \leftarrow \text{LEFT}(P_0)$.
36		LD1 1,2(COL)	$J \leftarrow \text{COL}(P_0)$.
37		CMP1 J0	
38		JE S4	Powtórz, jeśli $J = J_0$.
39		ENTA 0,1	
40		SLA 2	$rA(0:3) \leftarrow J$.
41		J1NN S5	
42		LDAN 2,3	Jeśli $J < 0$, to
43		FMUL ALPHA	$\text{VAL}(Q_0) \leftarrow -\text{ALPHA} \times \text{VAL}(Q_0)$.
44		STA 2,3	
45		JMP S3	
46	1H	ENT4 0,5	$P \leftarrow P_1$.
47		LD5 1,4(LEFT)	$P_1 \leftarrow \text{LEFT}(P)$.
48	S5	CMPA 1,5(COL)	<u>S5. Znajdowanie elementu I, J.</u>
49		JL 1B	Powtarzaj, aż $\text{COL}(P_1) \leq J$.
50		JE S7	Jeśli $=$, idź wprost do S7.
51	S6	LD5 BCOL,1(PTR)	<u>S6. Wstawianie elementu I, J.</u> $rI5 \leftarrow \text{PTR}[J]$.
52		LDA I	$rA(0:3) \leftarrow I$.
53	2H	ENT6 0,5	$rI6 \leftarrow rI5$.
54		LD5 0,6(UP)	$rI5 \leftarrow \text{UP}(rI6)$.
55		CMPA 0,5(ROW)	
56		JL 2B	Skocz, jeśli $\text{ROW}(rI5) > I$.
57		LD5 AVAIL	$X \Leftarrow \text{AVAIL}$.
58		J5Z OVERFLOW	
59		LDA 0,5(UP)	
60		STA AVAIL	
61		LDA 0,6(UP)	
62		STA 0,5(UP)	$\text{UP}(X) \leftarrow \text{UP}(\text{PTR}[J])$.
63		LDA 1,4(LEFT)	
64		STA 1,5(LEFT)	$\text{LEFT}(X) \leftarrow \text{LEFT}(P)$.
65		ST1 1,5(COL)	$\text{COL}(X) \leftarrow J$.
66		LDA I(ROW)	
67		STA 0,5(ROW)	$\text{ROW}(X) \leftarrow I$.
68		STZ 2,5	$\text{VAL}(X) \leftarrow 0$.
69		ST5 1,4(LEFT)	$\text{LEFT}(P) \leftarrow X$.
70		ST5 0,6(UP)	$\text{UP}(\text{PTR}[J]) \leftarrow X$.
71	S7	LDAN 2,3	<u>S7. Osiowanie.</u> $-\text{VAL}(Q_0)$
72		FMUL 2,2	$\times \text{VAL}(P_0)$

73	FADD 2,5	+ VAL(P1).
74	JAZ S8	Jeśli różnica jest bardzo mała, idź do S8.
75	STA 2,5	W przeciwnym razie wpisz do VAL(P1).
76	ST5 BCOL,1(PTR)	PTR[J] ← P1.
77	ENT4 0,5	P ← P1.
78	JMP S4A	P1 ← LEFT(P), to S4.
79 S8	LD6 BCOL,1(PTR)	<u>S8. Usuwanie elementu I, J. rI6 ← PTR[J].</u>
80	JMP *+2	
81	LD6 0,6(UP)	rI6 ← UP(rI6).
82	LDA 0,6(UP)	
83	DECA 0,5	Czy UP(rI6) = P1?
84	JANZ *-3	Powtarzaj, dopóki nie równe.
85	LDA 0,5(UP)	
86	STA 0,6(UP)	UP(rI6) ← UP(P1).
87	LDA 1,5(LEFT)	
88	STA 1,4(LEFT)	LEFT(P) ← LEFT(P1).
89	LDA AVAIL	AVAIL ⇌ P1.
90	STA 0,5(UP)	
91	ST5 AVAIL	
92	JMP S4A	P1 ← LEFT(P), to S4. ■

Uwaga: Korzystając z konwencji z rozdziału 4, wiersze 71–74 zapisalibyśmy

LDA 2,3; FMUL 2,2; FCMP 2,5; JE S8; STA TEMP; LDA 2,5; FSUB TEMP;

umieszczając odpowiedni parametr EPSILON w lokacji zero.

17. Dla każdego wiersza i i elementu $A[i,k] \neq 0$ dodajemy $A[i,k]$ razy wiersz k macierzy B do wiersza i macierzy C. Można to zrobić, operując wyłącznie wskaźnikami COL macierzy C. Dowiązania ROW łatwo potem uzupełnić. [A. Schoor, *Inf. Proc. Letters* **15** (1982), 87–89].

18. Trzy operacje osiowania, odpowiednio w kolumnach 3, 1, 2, dają

$$\begin{pmatrix} \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ -\frac{2}{3} & -\frac{1}{3} & -\frac{2}{3} \\ -\frac{1}{3} & -\frac{2}{3} & -\frac{1}{3} \end{pmatrix}, \quad \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{3}{2} & \frac{1}{2} & 1 \\ -\frac{1}{2} & -\frac{1}{2} & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 0 \\ -2 & 1 & 1 \\ 1 & -2 & 0 \end{pmatrix};$$

po ostatniej permutacji dostajemy odpowiedź

$$\begin{pmatrix} 1 & -2 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}.$$

20. $a_0 = \text{LOC}(A[1,1]) - 3$, $a_1 = 1$ lub 2 , $a_2 = 3 - a_1$.

21. Na przykład $M \leftarrow \max(I, J)$, $\text{LOC}(A[I,J]) = \text{LOC}(A[1,1]) + M(M-1) + I - J$. (Takie wzory podało niezależnie wiele osób. A. L. Rosenberg i H. R. Strong zaproponowali następujące uogólnienie k -wymiarowe: $\text{LOC}(A[I_1, \dots, I_k]) = L_k$, gdzie $L_1 = \text{LOC}(A[1, \dots, 1]) + I_1 - 1$, $L_r = L_{r-1} + (M_r - I_r)(M_r^{r-1} - (M_r - 1)^{r-1})$, a $M_r = \max(I_1, \dots, I_r)$ [IBM Tech. Disclosure Bull. **14** (1972), 3026–3028]. Więcej podobnych wyników zawarto w *Current Trends in Programming Methodology* **4** (Prentice-Hall, 1978), 263–311).

22. Zgodnie z kombinatorycznym systemem liczbowym (ćwiczenie 1.2.6–56), możemy przyjąć

$$p(i_1, \dots, i_k) = \binom{i_1}{1} + \binom{i_1 + i_2 + 1}{2} + \dots + \binom{i_1 + i_2 + \dots + i_k + k - 1}{k}.$$

[Det Kongelige Norske Videnskabers Selskabs Forhandlinger **34** (1961), 8–9].

23. Niech $c[J] = \text{LOC}(A[0,J]) = \text{LOC}(A[0,0]) + mJ$, jeżeli przy rozszerzeniu liczby kolumn od J do $J+1$ było m wierszy. Podobnie, niech $r[I] = \text{LOC}(A[I,0]) = \text{LOC}(A[0,0]) + nI$, jeżeli w chwili tworzenia wiersza I było n kolumn. Możemy skorzystać z funkcji przydzielenia pamięci

$$\text{LOC}(A[I,J]) = \begin{cases} I + c[J], & \text{jeśli } c[J] \geq r[I]; \\ J + r[I], & \text{w przeciwnym razie.} \end{cases}$$

Nietrudno udowodnić, że $c[J] \geq r[I]$ pociąga za sobą $c[J] \geq r[I] + J$, a $c[J] \leq r[I]$ pociąga $c[J] + I \leq r[I]$; stąd również zachodzi

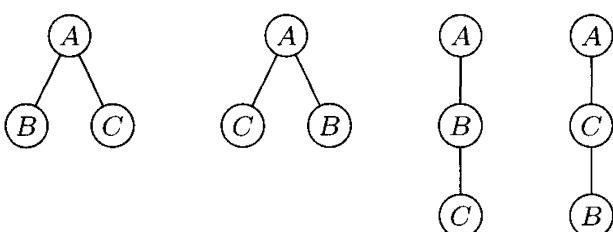
$$\text{LOC}(A[I,J]) = \max(I + \text{LOC}(A[0,J]), J + \text{LOC}(A[I,0])).$$

Nie musimy ograniczać przydziału pamięci do mn kolejnych lokacji; jedyne ograniczenie polega na przydzieleniu m lub n kolejnych nowych komórek pamięci w lokacjach większych niż przydzielone poprzednio. Ta konstrukcja pochodzi od E. J. Otoo i T. H. Mertetta [Computing **31** (1983), 1–9], którzy także uogólnili ją na k wymiarów.

24. [Aho, Hopcroft, Ullman, *Projektowanie i analiza algorytmów komputerowych* (PWN, Warszawa 1983), ćwiczenie 2.12] Oprócz tablicy A utrzymujemy tej samej wielkości tablicę „weryfikacyjną” V oraz listę L wykorzystanych lokacji. Niech n będzie liczbą elementów na liście L ; początkowo $n = 0$, a zawartości L , A , i V są dowolne. Przy każdym dostępie do $A[k]$ dla wartości k , która do tej pory mogła nie zostać użyta, sprawdzamy najpierw, czy $0 \leq V[k] < n$ i $L[V[k]] = k$. Jeśli nie, to przyjmujemy $V[k] \leftarrow n$, $L[n] \leftarrow k$, $A[k] \leftarrow 0$ i $n \leftarrow n + 1$. W przeciwnym razie można być pewnym, że $A[k]$ zawiera dopuszczalne dane. (Nieznacznie rozszerzając tę metodę, można przechowywać i odtwarzać zawartość wszystkich elementów A i V , które zostały zmienione podczas obliczenia).

2.3

1. Korzeń można wybrać na trzy sposoby. Gdy wybierzemy korzeń, powiedzmy A , możemy na trzy sposoby podzielić pozostałe węzły na poddrzewa: $\{B\}, \{C\}; \{C\}, \{B\}; \{B, C\}$. W ostatnim przypadku istnieją dwa sposoby utworzenia drzewa z $\{B, C\}$ (wybór korzenia). Mamy zatem cztery drzewa, w których A jest korzeniem, a ogólnie 12. Rozwiążanie tego problemu dla dowolnej liczby węzłów znajduje się w odpowiedzi do ćwiczenia 2.3.4.4–23.



2. Pierwsze dwa drzewa z odpowiedzi do ćwiczenia 1 traktowane jako drzewa zorientowane są tym samym drzewem, mamy więc tylko 9 różnych możliwości. Rozwiązanie ogólne jest podane w podpunkcie 2.3.4.4, gdzie znajduje się dowód wzoru n^{n-1} .

3. Część 1: Pokażemy, że istnieje *przynajmniej jeden* taki ciąg. Niech n oznacza liczbę węzłów drzewa. Gdy $n = 1$ sprawą jest prosta, bo X musi być korzeniem. Gdy $n > 1$, na mocy definicji istnieje korzeń X_1 i poddrzewa T_1, T_2, \dots, T_m ; albo $X = X_1$, albo X jest elementem dokładnie jednego poddrzewa T_j . W tym drugim przypadku na mocy indukcji istnieje ścieżka X_2, \dots, X , gdzie X_2 jest korzeniem T_j , a ponieważ X_1 jest rodzicem X_2 , mamy ścieżkę X_1, X_2, \dots, X .

Część 2: Pokażemy, że istnieje *co najwyżej jeden* taki ciąg. Udowodnimy przez indukcję, że jeśli X nie jest korzeniem drzewa, to X ma dokładnie jednego rodzica (w ten sposób X_k wyznacza X_{k-1} , X_{k-1} wyznacza X_{k-2} itd). Jeśli drzewo ma jeden węzeł, to nie ma czego dowodzić; w przeciwnym razie X należy do dokładnie jednego drzewa T_j . Albo X jest korzeniem T_j , a wtedy z definicji ma dokładnie jednego rodzica, albo X nie jest korzeniem T_j , a wtedy na mocy indukcji X ma dokładnie jednego rodzica w drzewie T_j , a żaden węzeł spoza T_j nie może być rodzicem węzła X .

4. Prawda (niestety).

5. 4.

6. Niech $\text{parent}^{[0]}(X)$ oznacza X i niech $\text{parent}^{[k+1]}(X) = \text{parent}(\text{parent}^{[k]}(X))$, tak że $\text{parent}^{[1]}(X)$ jest rodzicem X -a, a $\text{parent}^{[2]}(X)$ jest dziadkiem X -a; gdy $k \geq 2$, $\text{parent}^{[k]}(X)$ jest „(pra-) $^{k-2}$ dziadkiem” X -a. Poszukiwana relacja pokrewieństwa to „ $\text{parent}^{[m+1]}(X) = \text{parent}^{[m+n+1]}(Y)$, ale $\text{parent}^{[m]}(X) \neq \text{parent}^{[m+n]}(Y)$ ”. Jeśli $n > 0$, to ta relacja nie jest symetryczna względem X i Y , choć na codzień ludzie tak ją traktują.

7. Skorzystaj z (niesymetrycznego) warunku zdefiniowanego w ćwiczeniu 6, zakładając, że $\text{parent}^{[j]}(X) \neq \text{parent}^{[k]}(Y)$, jeśli j lub k (lub oba) są równe -1 . By pokazać, że ta relacja jest zawsze spełniona dla dokładnie jednej pary m i n , rozważmy notację dziesiętną Dewey'a dla X i Y , tj. $1.a_1.\dots.a_p.b_1.\dots.b_q$ i $1.a_1.\dots.a_p.c_1.\dots.c_r$, gdzie $p \geq 0$, $q \geq 0$, $r \geq 0$ i (jeśli $qr \neq 0$) $b_1 \neq c_1$. Liczby Dewey'a dla każdej pary węzłów można zapisać w tej postaci, a widać, że musimy wziąć $m = q - 1$ i $m + n = r - 1$.

8. Żadne drzewo binarne nie jest drzewem; te pojęcia są zupełnie rozłączne, choć diagram niepustego drzewa binarnego może przypominać diagram zwykłego drzewa.

9. Korzeniem jest A , bo umówiliśmy się, że korzeń będziemy rysować u góry.

10. Każda skończona kolekcja zbiorów zagnieżdżonych odpowiada lasowi w sensie definicji podanej w tekście: niech A_1, \dots, A_n oznacza te zbiory z kolekcji, które nie są właściwymi podzbiorami żadnych zbiorów z kolekcji. Dla ustalonego j podkolekcja wszystkich zbiorów z kolekcji będących podzbiorami A_j tworzy kolekcję zbiorów zagnieżdżonych. Możemy zatem założyć, że ta podkolekcja odpowiada (nieuporządkowanemu) drzewu o korzeniu A_j .

11. Niech w kolekcji \mathcal{C} zbiorów zagnieżdżonych relacja $X \equiv Y$ będzie prawdziwa wtedy i tylko wtedy, gdy istnieje takie $Z \in \mathcal{C}$, że $X \cup Y \subseteq Z$. Ta relacja w oczywisty sposób jest zwrotna i symetryczna, a w istocie jest relacją równoważności, ponieważ z $W \equiv X$ i $X \equiv Y$ wynika, że istnieją Z_1 i Z_2 z kolekcji \mathcal{C} , takie że $W \subseteq Z_1$, $X \subseteq Z_1 \cap Z_2$ i $Y \subseteq Z_2$. Ponieważ $Z_1 \cap Z_2 \neq \emptyset$, to $Z_1 \subseteq Z_2$ lub $Z_2 \subseteq Z_1$; stąd $W \cup Y \subseteq Z_1 \cup Z_2 \in \mathcal{C}$. Jeśli \mathcal{C} jest kolekcją zbiorów zagnieżdżonych, to możemy zdefiniować zorientowany las odpowiadający \mathcal{C} za pomocą reguły „ X jest przodkiem Y i Y jest potomkiem X wtedy i tylko wtedy, gdy $X \supset Y$ ”. Każda klasa równoważności na \mathcal{C} odpowiada drzewu zorientowanemu, będącemu lasem zorientowanym, w którym $X \equiv Y$ dla wszystkich X, Y . (Uogólniliśmy w ten sposób definicje lasu i drzewa na zbiory nieskończone). Przy takich ustalenach możemy zdefiniować poziom węzła X jako liczbę kardynalną zbioru

przodków X . Podobnie stopień węzła X możemy zdefiniować jako liczbę kardynalną zbioru klas równoważności na kolekcji wszystkich potomków X (która oczywiście jest kolekcją zbiorów zagnieżdżonych). Mówimy, że węzeł X jest *rodzicem* węzła Y , a węzeł Y jest *dzieckiem* węzła X , jeżeli węzeł X jest przodkiem węzła Y , ale nie istnieje takie Z , że $X \supset Z \supset Y$. (W przypadku zbiorów nieskończonych X może mieć potomków, ale nie mieć dzieci, a także mieć przodków, ale nie mieć rodziców). Jeśli interesują nas drzewa *uporządkowane*, to wystarczy uporządkować klasy równoważności w jakimś dowolnie wybranym porządku, na przykład zanurzając \subseteq w porządku liniowym, jak w ćwiczeniu 2.2.3–14.

Przykład (a): Niech $S_{\alpha k} = \{x \mid x = 0.d_1 d_2 d_3 \dots \text{ w zapisie dziesiętnym, gdzie } \alpha = 0.e_1 e_2 e_3 \dots \text{ w notacji dziesiętnej, a } d_j = e_j, \text{ gdy } j \bmod 2^k \neq 0\}$. Kolekcja $\mathcal{C} = \{S_{\alpha k} \mid k \geq 0, 0 < \alpha < 1\}$ jest kolekcją zbiorów zagnieżdżonych odpowiadającą drzewu o nieskończonym wielu poziomach i nieprzeliczalnym stopniu każdego węzła.

Przykłady (b), (c): O wiele wygodniej zdefiniować ten zbiór na płaszczyźnie, a to wystarczy, bo między punktami płaszczyzny a wszystkimi liczbami rzeczywistymi istnieje wzajemnie jednoznaczna odpowiedniość. Niech $S_{\alpha mn} = \{(\alpha, y) \mid m/2^n \leq y < (m+1)/2^n\}$ i niech $T_\alpha = \{(x, y) \mid x \leq \alpha\}$. Kolekcja $\mathcal{C} = \{S_{\alpha mn} \mid 0 < \alpha < 1, n \geq 0, 0 \leq m < 2^n\} \cup \{T_\alpha \mid 0 < \alpha < 1\}$ jest kolekcją zbiorów zagnieżdżonych, co nietrudno zauważyc. Dziećmi $S_{\alpha mn}$ są $S_{\alpha(2m)(n+1)}$ i $S_{\alpha(2m+1)(n+1)}$, a T_α ma dzieci $S_{\alpha 00}$ plus drzewo $\{S_{\beta mn} \mid \beta < \alpha\} \cup \{T_\beta \mid \beta < \alpha\}$. Zatem każdy węzeł ma stopień 2 i nieprzeliczalnie wielu przodków postaci T_α . Ta konstrukcja pochodzi od R. Bigelowa.

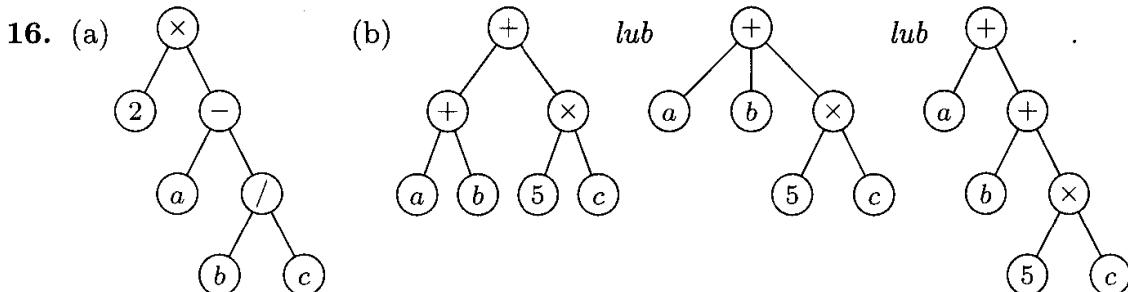
Uwaga: Jeśli weźmiemy odpowiedni dobry porządek na liczbach rzeczywistych i zdefiniujemy $T_\alpha = \{(x, y) \mid x > \alpha\}$, to możemy powyższą konstrukcję nieco poprawić, otrzymując kolekcję zbiorów zagnieżdżonych, w której każdy węzeł leży na poziomie o numerze nieprzeliczalnym, ma stopień 2 oraz dwoje dzieci.

12. Nakładamy na porządek częściowy dodatkowy warunek (analogiczny do warunku na „zbiory zagnieżdżone”), by wymusić odpowiedniość z pewnym lasem: jeśli $x \preceq y$ i $x \preceq z$, to $y \preceq z$ lub $z \preceq y$. Innymi słowy, dla dowolnego elementu wszystkie elementy od niego większe są uporządkowane liniowo. By dostać drzewo, musimy założyć dodatkowo, że istnieje element największy r , taki że $x \preceq r$ dla wszystkich x . Dowód, że przy takich warunkach dostajemy drzewo nieuporządkowane wg definicji z tekstu, przebiega analogicznie do dowodu w ćwiczeniu 10.

14. S jest niepusty, zatem zawiera element $1.a_1 \dots a_k$, gdzie k jest jak najmniejsze. Jeśli $k > 0$, to możemy także wziąć najmniejsze z możliwych a_k . Szybko okaże się, że k musi być równe 0. Innymi słowy, S musi zawierać element 1. Niech 1 będzie korzeniem. Dla wszystkich innych elementów $k > 0$, zatem pozostałe elementy S można podzielić na zbiory $S_j = \{1.j.a_2 \dots a_k\}$, $1 \leq j \leq m$, dla pewnego $m \geq 0$. Jeśli $m \neq 0$ i S_m jest niepusty, to za pomocą podobnego rozumowania dochodzimy do wniosku, że $1.j$ należy do S_j dla każdego S_j ; stąd każdy zbiór S_j jest niepusty. Łatwo się przekonać, że zbiory $S'_j = \{1.a_2 \dots a_k \mid 1.j.a_2 \dots a_k \text{ należy do } S_j\}$ spełniają ten sam warunek co S . Na mocy indukcji każdy ze zbiorów S_j tworzy drzewo.

15. Niech 1 będzie korzeniem i niech korzeniami lewego i prawego poddrzewa węzła α będą odpowiednio $\alpha.0$ i $\alpha.1$ (jeżeli te korzenie istnieją). Na przykład król Chrystian IX pojawia się w dwóch miejscach na rysunku 18(a), konkretnie 1.0.0.0.0 i 1.1.0.0.1.0. Możemy skrócić zapis, pomijając kropki, i pisać po prostu 10000 i 110010. *Uwaga:* Notację tę zauważamy Francisowi Galtonowi; zobacz *Natural Inheritance* (Macmillan, 1889), 249. Przy drzewach rodowych lepiej używać skrótów O i M zamiast liczb 0 i 1, a także pomijać początkową jedynkę. W takiej notacji Chrystian IX jest

MOOMO Karola, tj. jego matki ojca ojca matki ojcem. Konwencja zerojedynkowa jest ciekawa z innego powodu: wyznacza ważny związek między węzłami drzewa binarnego a dodatnimi liczbami całkowitymi zapisanymi w systemie dwójkowym (adresy pamięci komputera).



17. $\text{parent}(F[1]) = A$; $\text{parent}(F[1, 2]) = C$; $\text{parent}(F[1, 2, 2]) = E$.

18. $L[5, 1, 1] = (2)$. $L[3, 1]$ nie ma sensu, ponieważ $L[3]$ jest listą pustą.

19.

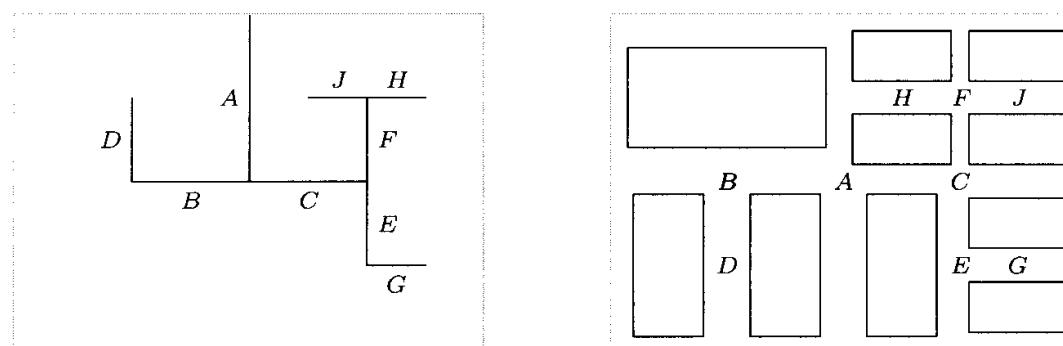
$$\begin{array}{c} * [L] \\ \diagup \quad \diagdown \\ a \quad * \\ \quad \quad | \\ \quad \quad [L] \end{array} \quad L[2] = (L); \quad L[2, 1, 1] = a.$$

20. (Intuicyjnie, odpowiedniość między 0-2-drzewami a drzewami binarnymi uzyskujemy, usuwając wszystkie liście 0-2-drzewa; zobacz: ważna konstrukcja w podpunkcie 2.3.4.5). Niech 0-2-drzewo o jednym węźle odpowiada pustemu drzewu binarnemu. Niech 0-2-drzewo o więcej niż jednym węźle, składające się z korzenia r i 0-2-drzew T_1 i T_2 , odpowiada drzewu binarnemu o korzeniu r , lewym poddrzewie T'_1 i prawym poddrzewie T'_2 , gdzie T_1 i T_2 odpowiadają drzewom T'_1 i T'_2 .

21. $1 + 0 \cdot n_1 + 1 \cdot n_2 + \dots + (m-1) \cdot n_m$. Dowód: Liczba węzłów w drzewie wynosi $n_0 + n_1 + n_2 + \dots + n_m$, a to równa się także $1 + (\text{liczba dzieci w drzewie}) = 1 + 0 \cdot n_0 + 1 \cdot n_1 + 2 \cdot n_2 + \dots + m \cdot n_m$.

22. Pomysł polega na rekurencyjnym rysowaniu schematu. Reprezentacja niepstego drzewa binarnego to reprezentacja jego korzenia plus zmniejszone dwukrotnie i przekręcone o 90° reprezentacje jego poddrzew. W ten sposób, dysponując odpowiednio silnym szkłem powiększającym, na jednej kartce papieru można narysować dowolnie duże drzewo binarne.

Mozna wymyślać wiele wariacji na ten temat. Na przykład: reprezentujemy korzeń za pomocą pionowej linii od środka poziomo położonej kartki do jej górnej krawędzi; reprezentację lewego poddrzewa obracamy o 90° zgodnie z ruchem wskazówek zegara i ustaviamy na lewej połówce, a reprezentacje prawego poddrzewa obracamy o 90° przeciwnie do ruchu wskazówek zegara i ustaviamy na prawej połówce. Każdy węzeł jest reprezentowany za pomocą linii. (Gdy zastosujemy tę metodę do pełnych drzew



binarnych o $2^k - 1$ węzłach na k poziomach, dostajemy tzw. „H-drzewa”, które są najbardziej wydajnym ułożeniem takich drzew binarnych na płytach układów VLSI; zobacz R. P. Brent, H. T. Kung, *Inf. Proc. Letters* 11 (1980), 46–48). Inny pomysł polega na reprezentowaniu pustych drzew binarnych za pomocą prostokątów i obracaniu reprezentacji niepustych poddrzew w taki sposób, by reprezentacje lewych poddrzew były na przemian na lewo lub poniżej reprezentacji odpowiednich prawych poddrzew w zależności od tego, czy poziom rekursji jest parzysty, czy nie. Prostokąty odpowiadają węzłom zewnętrznym w rozszerzonych drzewach binarnych (zobacz podpunkt 2.3.4.5). Ta reprezentacja, mająca związek z drzewami dwuwymiarowymi oraz drzewami czterokowymi omawianymi w podrozdziale 6.5, jest szczególnie przydatna wtedy, gdy węzły zewnętrzne zawierają informacje, a wewnętrzne nie.

2.3.1

1. $\text{INFO}(T) = A$, $\text{INFO}(\text{RLINK}(T)) = C$ itd.; odpowiedź: H .
2. Preorder: 1245367; porządek symetryczny: 4251637; postorder: 4526731.
3. Zdanie jest prawdziwe. Zauważmy na przykład, że węzły 4, 5, 6, 7 w ćwiczeniu 2 zawsze występują w tym porządku. Dowód przez indukcję względem rozmiaru drzewa binarnego jest natychmiastowy.
4. Jest to porządek odwrotny do porządku postorder. (Łatwy dowód przez indukcję).
5. Na przykład w drzewie z ćwiczenia 2 porządek preorder daje 1, 10, 100, 101, 11, 110, 111 w zapisie dwójkowym (który w tym akurat przypadku jest równoważny zapisowi w systemie Dewey'a). Ciągi zerojedynkowe zostały posortowane tak, jak słowa w słowniku.

W ogólności węzły stoją w porządku preorder, jeśli są posortowane leksykograficznie od lewej do prawej przy założeniu, że „spacja” $< 0 < 1$. Węzły stoją w porządku postorder, jeśli są posortowane leksykograficznie przy założeniu $0 < 1 <$ „spacja”. Dla porządku inorder – analogicznie, $0 < \text{„spacja”} < 1$.

(Co więcej, jeśli wyobrażymy sobie spacje po lewej stronie i potraktujemy etykiety Dewey'a jako zwykłe liczby dziesiętne, otrzymamy *porządek poziomy*; zobacz 2.3.3–(8)).

6. Fakt, że permutacja $p_1 p_2 \dots p_n$ jest osiągalna za pomocą stosu, można łatwo udowodnić przez indukcję względem n . Możemy także zauważyć, że operacje stosowe algorytmu T robią dokładnie to, co potrzeba. (Odpowiedni ciąg S-ów i X-ów, jak w ćwiczeniu 2.2.1–3, jest taki sam jak ciąg indeksów będących jedynkami i dwójkami w porządku podwójnym, zobacz ćwiczenie 18).

W drugą stronę, jeśli permutację $p_1 p_2 \dots p_n$ można otrzymać za pomocą stosu i jeśli $p_k = 1$, to $p_1 \dots p_{k-1}$ jest permutacją zbioru $\{2, \dots, k\}$, a $p_{k+1} \dots p_n$ jest permutacją zbioru $\{k+1, \dots, n\}$. Te permutacje odpowiadają lewemu i prawemu poddrzewu, przy tym obie można uzyskać za pomocą stosu. Reszta na mocy indukcji.

7. Z porządku preorder wiemy, który węzeł jest korzeniem; z porządku inorder możemy więc wyznaczyć lewe i prawe poddrzewo, a w istocie znamy porządek preorder i inorder węzłów tych poddrzew. Wynika stąd, że drzewo daje się w łatwy sposób skonstruować (jest coś niezwykłego w prościutkim algorytmie budującym drzewo z węzłów połączonych w listę w porządku preorder za pomocą pól LLINK i jednocześnie w listę w porządku inorder za pomocą pól RLINK). Podobnie strukturę wyznacza się za pomocą porządku postorder i inorder, ale nie preorder i postorder. Są dwa drzewa binarne, których węzły w porządku preorder to AB , a w postorder BA . Jeżeli wszystkie węzły

wewnętrzne drzewa mają niepuste oba poddrzewa, to struktura drzewa jest wyznaczona jednoznacznie przez parę preorder-postorder.

8. (a) Drzewa binarne, w których wszystkie LLINK są dowiązaniem pustym. (b) Drzewa binarne o liczbie węzłów nie przekraczającej jeden. (c) Drzewa binarne, w których wszystkie RLINK są dowiązaniem pustym.

9. T1 raz, T2 $2n+1$ razy, T3 n razy, T4 $n+1$ razy, T5 n razy. Te wartości można wyznaczyć za pomocą indukcji, prawa Kirchhoffa i analizy programu T.

10. Przy przechodzeniu drzewa binarnego, w którym wszystkie prawe poddrzewa są puste, na stos zostanie włożonych wszystkich n adresów węzłów drzewa, zanim którykolwiek zostanie zdjęty.

11. Niech a_{nk} będzie liczbą drzew binarnych o n węzłach, dla których stos w algorytmie T nigdy nie zawiera więcej niż k elementów. Jeżeli $g_k(z) = \sum_n a_{nk} z^n$, to $g_1(z) = 1/(1-z)$, $g_2(z) = 1/(1-z/(1-z)) = (1-z)/(1-2z)$, ..., $g_k(z) = 1/(1-zg_{k-1}(z)) = q_{k-1}(z)/q_k(z)$, gdzie $q_{-1}(z) = q_0(z) = 1$, $q_{k+1}(z) = q_k(z) - zq_{k-1}(z)$; stąd $g_k(z) = (f_1(z)^{k+1} - f_2(z)^{k+1})/(f_1(z)^{k+2} - f_2(z)^{k+2})$, gdzie $f_j(z) = \frac{1}{2}(1 \pm \sqrt{1-4z})$. Można teraz pokazać, że $a_{nk} = [u^n](1-u)(1+u)^{2n}(1-u^{k+1})/(1-u^{k+2})$; stąd $s_n = \sum_{k \geq 1} k(a_{nk} - a_{n(k-1)})$ jest równe $[u^{n+1}](1-u)^2(1+u)^{2n} \sum_{j \geq 1} u^j/(1-u^j)$ minus a_{nn} . Zastosowanie metody z ćwiczenia 5.2.2–52 umożliwia otrzymanie szeregu asymptotycznego

$$s_n/a_{nn} = \sqrt{\pi n} - \frac{3}{2} - \frac{13}{24} \sqrt{\frac{\pi}{n}} + \frac{1}{2n} + O(n^{-3/2}).$$

[N. G. de Bruijn, D. E. Knuth, S. O. Rice, *Graph Theory and Computing*, ed. R. C. Read (New York: Academic Press, 1972), 15–22].

Gdy drzewo binarne reprezentuje las, jak było opisane w punkcie 2.3.2, wtedy analizowana wielkość jest *wysokością* tego lasu (największą odlegością między węzłem i korzeniem plus jeden). Uogólnienia na wiele innych rodzajów drzew podali Flajolet i Odlyzko [*J. Computer and System Sci.* **25** (1982), 171–213]. Rozkład asymptotyczny wysokości, nie tylko w pobliżu średniej, był następnie analizowany przez Flajoleta, Gao, Odlyzkę i Richmonda [*Combinatorics, Probability, and Computing* **2** (1993), 145–156].

12. Odwiedź NODE(P) między krokami T2 i T3, zamiast w kroku T5. Dowód sprowadza się do wykazania stwierdzenia „Jeśli w kroku T2 ... o zawartości pierwotnej $A[1] \dots A[m]$ ”, jak w tekście.

13. (Autor rozwiązania: S. Araújo, 1976) Kroki T1–T4 pozostawiamy bez zmian, z tym że w kroku T1 na nową zmienną Q przypisujemy Λ ; Q będzie wskazywać na ostatni odwiedzony węzeł (jeśli taki istnieje). Krok T5 rozbijamy na dwa kroki:

T5. [Czy prawa gałąź obsłużona?] Jeśli $RLINK(P) = \Lambda$ lub $RLINK(P) = Q$, to przejdź do T6; w przeciwnym razie $A \leftarrow P$, $P \leftarrow RLINK(P)$ i wróć do T2.

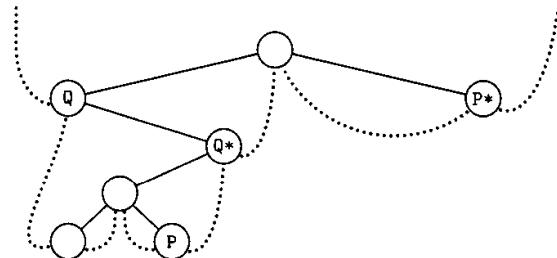
T6. [Odwiedź P] „Odwiedź” NODE(P), $Q \leftarrow P$, wróć do T4.

Dowód wygląda podobnie. (Można tak zoptymalizować kroki T4 i T5, by elementy nie były zdejmowane ze stosu i za chwilę wkładane z powrotem).

14. Przez indukcję dowodzimy, że zawsze jest dokładnie $n+1$ dowiązań Λ (jeżeli liczymy T, gdy drzewo jest puste). Dowiązań niepustych jest n (wliczając T), zatem prawdziwe jest pojawiające się w tekście stwierdzenie, że dowiązań pustych jest więcej niż niepustych.

15. Fastryga na LLINK lub RLINK prowadzi do węzła wtedy i tylko wtedy, gdy ma on odpowiednio niepuste lewe lub prawe poddrzewo. (Zobacz rysunek 24).

16. Jeśli $\text{LTAG}(Q) = 0$, to Q^* jest $\text{LLINK}(Q)$; stąd Q^* leży jeden krok w dół i na lewo. W przeciwnym razie do Q^* dostajemy się, idąc (jeśli trzeba) w górę po drzewie tak długo, aż można zejść w dół w prawo, nie cofając się. Przykłady: trasa od P do P^* i od Q do Q^* w następującym drzewie:



17. Jeśli $\text{LTAG}(P) = 0$, to $Q \leftarrow \text{LLINK}(P)$ i zakończ algorytm. W przeciwnym razie $Q \leftarrow P$, następnie zero lub więcej razy $Q \leftarrow \text{RLINK}(Q)$, aż do natrafienia na $\text{RTAG}(Q) = 0$; na koniec jeszcze raz $Q \leftarrow \text{RLINK}(Q)$.

18. Zmodyfikuj algorytm T, wstawiając krok T2.5, „Odwiedź $\text{NODE}(P)$ po raz pierwszy”; w kroku T5 odwiedzamy $\text{NODE}(P)$ po raz drugi.

Obchodzenie drzewa sfastrygowanego jest niezwykle proste:

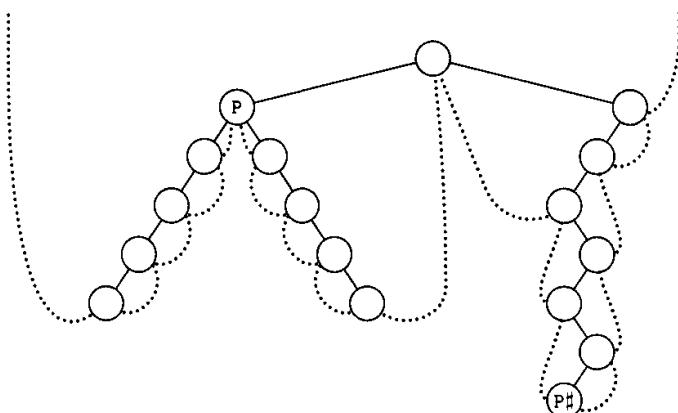
$$(P, 1)^\Delta = (\text{LLINK}(P), 1) \text{ jeśli } \text{LTAG}(P) = 0, \text{ w przeciwnym razie } (P, 2);$$

$$(P, 2)^\Delta = (\text{RLINK}(P), 1) \text{ jeśli } \text{RTAG}(P) = 0, \text{ w przeciwnym razie } (\text{RLINK}(P), 2).$$

W każdym przypadku przemieszczamy się co najwyżej o jeden krok; w praktyce porządek podwójny jest zakodowany w strukturze programu, a wartości d i e nie pojawiają się w programie w sposób jawny.

Pomijając wszystkie pierwsze wizyty, otrzymujemy dokładnie algorytmy T i S; pomijając wszystkie drugie wizyty, otrzymujemy rozwiązania ćwiczeń 12 i 17.

19. Pomysł polega na rozpoczęciu od wyznaczenia rodzica Q węzła P . Następnie jeśli $P \neq \text{LLINK}(Q)$, to mamy $P\# = Q$; w przeciwnym razie możemy wyznaczyć $P\#$, zero lub więcej razy wykonując $Q \leftarrow Q\$$, aż $\text{RTAG}(Q) = 1$. (Porównaj P i $P\#$ w pokazanym drzewie).



Nie istnieje wydajny algorytm wyznaczania rodzica węzła P w drzewie binarnym z fastrygą prawostronną, ponieważ drzewo zdegenerowane, w którym wszystkie lewe dowiązania są puste, jest niczym więcej jak listą cykliczną z dowiązaniami w złą stronę. Nie możemy zatem obchodzić drzewa binarnego z fastrygą prawostronną w porządku postorder z taką samą wydajnością, jak w korzystającej ze stosu metodzie z ćwiczenia 13, jeżeli nie przechowujemy „historii” wyznaczającej drogę, którą przyszliśmy do węzła P .

Ale jeśli drzewo jest sfastrygowane w obu kierunkach, to *możemy* w wydajny sposób znaleźć rodzica węzła P:

F1. $Q \leftarrow P$ i $R \leftarrow P$.

F2. Jeśli $LTAG(Q) = RTAG(R) = 0$, to $Q \leftarrow LLINK(Q)$ i $R \leftarrow RLINK(R)$; powtórz ten krok. W przeciwnym razie idź do F4, jeśli $RTAG(R) = 1$.

F3. Przyjmij $Q \leftarrow LLINK(Q)$ i zakończ algorytm, jeśli $P = RLINK(Q)$. W przeciwnym razie zero lub więcej razy wykonaj $R \leftarrow RLINK(R)$, aż $RTAG(R) = 1$; następnie przyjmij $Q \leftarrow RLINK(R)$ i zakończ algorytm.

F4. Przyjmij $R \leftarrow RLINK(R)$ i jeżeli $P = LLINK(R)$, to przyjmij $Q \leftarrow R$ i zakończ algorytm. W przeciwnym razie zero lub więcej razy wykonaj $Q \leftarrow LLINK(Q)$, aż $LTAG(Q) = 1$, następnie przyjmij $Q \leftarrow LLINK(Q)$ i zakończ algorytm. ■

Średni czas wykonania algorytmu F wynosi $O(1)$, gdy P jest dowolnym węzłem drzewa; jeśli policzymy tylko kroki $Q \leftarrow LLINK(Q)$ dla P będącego prawym dzieckiem lub tylko kroki $R \leftarrow RLINK(R)$ dla P będącego lewym dzieckiem, to po każdym dowiązaniu przechodzimy dokładnie dla jednego węzła P.

20. Należy zamienić wiersze 06–09 na: Należy zamienić wiersze 12–13 na:

T3	ENT4	0,6
LD6	AVAIL	
J6Z	OVERFLOW	
LDX	0,6(LINK)	
STX	AVAIL	
ST5	0,6(INFO)	
ST4	0,6(LINK)	

LD4	0,6(LINK)
LD5	0,6(INFO)
LDX	AVAIL
STX	0,6(LINK)
ST6	AVAIL
ENT6	0,4

Jeżeli w wierszu 06 wstawimy dodatkowo:

T3 LD3 0,5(LLINK)
J3Z T5

Przejdź do T5, jeśli $LLINK(P) = \Lambda$.

nie zapominając o odpowiednich modyfikacjach w wierszach 10 i 11, to czas wykonania zmniejszy się z $(30n + a + 4)u$ do $(27a + 6n - 22)u$. (Podobne usprawnienie umożliwia zmniejszenie czasu wykonania programu T do $(12a + 6n - 7)u$, co daje pewien zysk, gdy $a = (n + 1)/2$).

21. Poniższe rozwiązańe, autorstwa Josepha M. Morrisa [Inf. Proc. Letters 9 (1979), 197–200], pozwala przejść drzewo również w porządku preorder (zobacz ćwiczenie 18).

U1. [Inicjowanie] $P \leftarrow T$ i $R \leftarrow \Lambda$.

U2. [Czy gotowe?] Jeśli $P = \Lambda$, to algorytm się zatrzymuje.

U3. [Przejście w lewo] $Q \leftarrow LLINK(P)$. Jeśli $Q = \Lambda$, to odwiedź $NODE(P)$ w porządku preorder i przejdź do U6.

U4. [Szukanie fastrygi] Zero lub więcej razy wykonaj $Q \leftarrow RLINK(Q)$, aż $Q = R$ lub $RLINK(Q) = \Lambda$.

U5. [Wstawianie lub usuwanie fastrygi] Jeśli $Q \neq R$, to $RLINK(Q) \leftarrow P$ i przejdź do U8. W przeciwnym razie $RLINK(Q) \leftarrow \Lambda$ (chwilowo było zmienione na P, ale właśnie przeszliśmy lewe poddrzewo węzła P).

U6. [Odwiedzanie w porządku inorder] Odwiedź $NODE(P)$ w porządku inorder.

U7. [Przejście w prawo lub w góre] Przyjmij $R \leftarrow P$, $P \leftarrow \text{RLINK}(P)$ i wróć do U2.

U8. [Odwiedzanie w porządku preorder] Odwiedź $\text{NODE}(P)$ w porządku preorder.

U9. [Przejście w lewo] $P \leftarrow \text{LLINK}(P)$ i wróć do kroku U3. ■

Morris zaproponował także nieco bardziej skomplikowany sposób przechodzenia drzewa w porządku postorder.

Zupełnie inne rozwiązanie pochodzi od J. M. Robsona [*Inf. Proc. Letters* 2 (1973), 12–14]. Powiemy, że węzeł jest „pełny”, jeśli ani LLINK, ani RLINK nie są dowiązaniem pustymi. Powiemy, że węzeł jest „pusty”, jeśli jednocześnie jego LLINK i RLINK są dowiązaniem pustym. Robson znalazł sposób, który umożliwia za pomocą pól LLINK i RLINK węzłów pustych utrzymywać stos wskaźników do węzłów pełnych, których prawe poddrzewa są właśnie odwiedzane!

Jeszcze inny sposób poradzenia sobie bez pomocniczego stosu odkryli niezależnie G. Lindstrom i B. Dwyer, *Inf. Proc. Letters* 2 (1973), 47–51, 143–145. Ich algorytm przechodzi drzewa w porządku potrójnym – każdy węzeł jest odwiedzany dokładnie trzy razy, raz w porządku preorder, raz inorder i raz postorder – ale przy odwiedzaniu węzła nie wiadomo, która to wizyta.

W1. [Inicjowanie] Przyjmij $P \leftarrow T$ i $Q \leftarrow S$, gdzie S jest wartownikiem – dowolną liczbą różną od dowolnego dowiązania w drzewie (na przykład –1).

W2. [Pominięcie pustych] Jeśli $P = \Lambda$, to przyjmij $P \leftarrow Q$ i $Q \leftarrow \Lambda$.

W3. [Czy gotowe?] Jeśli $P = S$, to zakończ algorytm. (W chwili zakończenia będzie $Q = T$).

W4. [Odwiedzanie] Odwiedź $\text{NODE}(P)$.

W5. [Wykonanie rotacji] Przyjmij $R \leftarrow \text{LLINK}(P)$, $\text{LLINK}(P) \leftarrow \text{RLINK}(P)$, $\text{RLINK}(P) \leftarrow Q$, $Q \leftarrow P$, $P \leftarrow R$ i wróć do W2. ■

Poprawność algorytmu wynika z faktu, że startując w kroku W2 ze wskaźnikiem P wskazującym na korzeń drzewa binarnego T oraz wskaźnikiem Q wskazującym na X , gdzie X nie jest dowiązaniem do tego drzewa, trzykrotnie przejdziemy drzewo i znajdziemy się w kroku W3 z wartościami $P = X$ i $Q = T$.

Jeśli $\alpha(T) = x_1x_2\dots x_{3n}$ jest wynikowym ciągiem węzłów w porządku potrójnym, to mamy $\alpha(T) = T \alpha(\text{LLINK}(T)) T \alpha(\text{RLINK}(T)) T$. Stąd, jak zauważył Lindstrom, każdy z trzech podciągów $x_1x_4\dots x_{3n-2}$, $x_2x_5\dots x_{3n-1}$, $x_3x_6\dots x_{3n}$ zawiera każdy węzeł drzewa dokładnie raz. (Z uwagi na fakt, że x_{j+1} jest albo rodzicem, albo dzieckiem x_j , kolejność węzłów w podciagu jest taka, że każdy węzeł znajduje się w odległości co najwyżej trzech dowiązań od swojego poprzednika. W punkcie 7.2.1 opisujemy ogólny schemat zwany porządkiem *prepostorder*, który ma tę własność nie tylko dla drzew binarnych, ale dla drzew w ogóle).

22. W tym programie przyjmujemy takie konwencje, jak w programach T i S ; Q znajduje się w rI6 i/lub rI4. Staroświecki MIX nie sprawdza się przy porównywaniach rejestrów indeksowych, zatem pomineliśmy zmienną R , a warunek „ $Q = R$ ” zamieniliśmy na „ $\text{RLINK}(Q) = P$ ”.

01	U1	LD5	HEAD(LLINK)	1	<i>U1. Inicjowanie.</i> $P \leftarrow T$.
02	U2A	J5Z	DONE	1	Zatrzymaj się, jeśli $P = \Lambda$.
03	U3	LD6	0,5(LLINK)	$n+a-1$	<i>U3. Przejście w lewo.</i> $Q \leftarrow \text{LLINK}(P)$.
04		J6Z	U6	$n+a-1$	Idź do U6, jeśli $Q = \Lambda$.

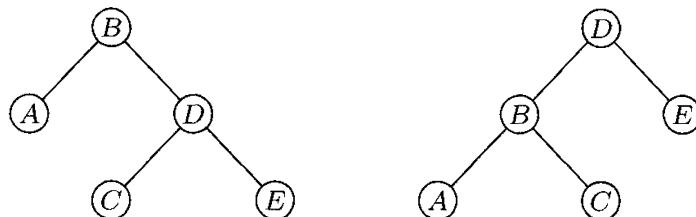
05	U4	CMP5 1,6(RLINK)	$2n - 2b$	<u>U4. Szukanie fastrygi.</u>
06		JE 5F	$2n - 2b$	Skocz, jeśli RLINK(Q) = P.
07		ENT4 0,6	$2n - 2b - a + 1$	$rI4 \leftarrow Q$.
08		LD6 1,6(RLINK)	$2n - 2b - a + 1$	
09		J6NZ U4	$2n - 2b - a + 1$	Idź do U4.
10	U5	ST5 1,4(RLINK)	$a - 1$	<u>U5a. Wstawienie fastrygi.</u> RLINK(Q) $\leftarrow P$.
11	U9	LD5 0,5(LLINK)	$a - 1$	<u>U9. Przejście w lewo.</u> P $\leftarrow LLINK(P)$.
12		JMP U3	$a - 1$	Idź do U3.
13	5H	STZ 1,6(RLINK)	$a - 1$	<u>U5b. Usunięcie fastrygi.</u> RLINK(Q) $\leftarrow \Lambda$.
14	U6	JMP VISIT	n	<u>U6. Odwiedzanie w porządku inorder.</u>
15	U7	LD5 1,5(RLINK)	n	<u>U7. W prawo lub w góre.</u> P $\leftarrow RLINK(P)$.
16	U2	J5NZ U3	n	<u>U2. Czy gotowe?</u> Idź do U3, jeśli P $\neq \Lambda$.
17	DONE	...		

Całkowity czas wykonania wynosi $21n + 6a - 3 - 14b$, gdzie n jest liczbą węzłów, a jest liczbą pustych dowiązań RLINK (stąd $a - 1$ jest liczbą niepustych dowiązań LLINK), a b jest liczbą węzłów w skrajnie prawej ścieżce T, RLINK(T), RLINK(RLINK(T)) itd.

23. Wstawianie w prawe poddrzewo: przyjmij RLINKT(Q) $\leftarrow RLINKT(P)$, RLINK(P) $\leftarrow Q$, RTAG(P) $\leftarrow 0$, LLINK(Q) $\leftarrow \Lambda$. Wstawianie w lewe poddrzewo zakładając LLINK(P) = Λ : LLINK(P) $\leftarrow Q$, LLINK(Q) $\leftarrow \Lambda$, RLINK(Q) $\leftarrow P$, RTAG(Q) $\leftarrow 1$. Wstawianie w lewe poddrzewo między P i LLINK(P) $\neq \Lambda$: R $\leftarrow LLINK(P)$, LLINK(Q) $\leftarrow R$, następnie zero lub więcej razy wykonaj R $\leftarrow RLINK(R)$, aż RTAG(R) = 1; na koniec RLINK(R) $\leftarrow Q$, LLINK(P) $\leftarrow Q$, RLINK(Q) $\leftarrow P$, RTAG(Q) $\leftarrow 1$.

(Dla ostatniego przypadku można podać wydajniejszy algorytm, jeżeli dysponujemy takim węzłem F, że P = LLINK(F) lub P = RLINK(F); na przykład przy tym ostatnim założeniu możemy wykonać INFO(P) \leftrightarrow INFO(Q), RLINK(F) $\leftarrow Q$, LLINK(Q) $\leftarrow P$, RLINKT(P) $\leftarrow RLINKT(P)$, RLINK(P) $\leftarrow Q$, RTAG(P) $\leftarrow 1$; to zajmuje stały czas, ale generalnie nie poleca się takiego rozwiązania, gdyż powoduje ono zmianę położenia węzłów w pamięci).

24. Nie:



25. Po pierwsze dowodzimy (b) przez indukcję względem liczby węzłów T , podobnie (c). Teraz (a) rozpada się na kilka przypadków: piszemy $T \preceq_1 T'$, jeśli jest spełnione (i), $T \preceq_2 T'$, jeśli (ii) itd. Z $T \preceq_1 T'$ i $T' \preceq T''$ wynika $T \preceq_1 T''$; z $T \preceq_2 T'$ i $T' \preceq T''$ wynika $T \preceq_2 T''$; pozostałe dwa przypadki załatwiamy, dowodząc (a) przez indukcję względem liczby węzłów T .

26. Bierzymy ciąg węzłów drzewa T w porządku podwójnym $(u_1, d_1), (u_2, d_2), \dots, (u_{2n}, d_{2n})$, gdzie u oznaczają węzły, a d numery wizyt 1 i 2. Z tego ciągu konstruujemy „ślad” drzewa $(v_1, s_1), (v_2, s_2), \dots, (v_{2n}, s_{2n})$, gdzie $v_j = \text{info}(u_j)$ i $s_j = l(u_j)$ lub $r(u_j)$ w zależności od tego, czy $d_j = 1$ czy 2. $T \preceq T'$ wtedy i tylko wtedy, gdy ślad T (wg powyższej definicji) leksykograficznie poprzedza lub jest równy śladowi T' . Formalnie oznacza to, że mamy $n \leq n'$ i $(v_j, s_j) = (v'_j, s'_j)$ dla $1 \leq j \leq n$, albo istnieje takie k , że $(v_j, s_j) = (v'_j, s'_j)$ dla $1 \leq j < k$ i albo $v_k \prec v'_k$, albo $v_k = v'_k$ i $s_k < s'_k$.

- 27.** **R1.** [Inicjowanie] Przyjmij $P \leftarrow \text{HEAD}$, $P' \leftarrow \text{HEAD}'$; to są atrapy danych drzew binarnych z fastrygą prawostronną. Przejdź do R3.
- R2.** [Sprawdzanie INFO] Jeśli $\text{INFO}(P) \prec \text{INFO}(P')$, to zakończ algorytm ($T \prec T'$); jeśli $\text{INFO}(P) \succ \text{INFO}(P')$, to zakończ algorytm ($T \succ T'$).
- R3.** [Przejście w lewo] Jeśli $\text{LLINK}(P) = \Lambda = \text{LLINK}(P')$, to przejdź do R4; jeśli $\text{LLINK}(P) = \Lambda \neq \text{LLINK}(P')$, to zakończ algorytm ($T \prec T'$); jeśli $\text{LLINK}(P) \neq \Lambda = \text{LLINK}(P')$, to zakończ algorytm ($T \succ T'$); w przeciwnym razie $P \leftarrow \text{LLINK}(P)$, $P' \leftarrow \text{LLINK}(P')$ i przejdź do R2.
- R4.** [Czy koniec drzewa?] Jeśli $P = \text{HEAD}$ (lub, co równoważne, jeśli $P' = \text{HEAD}'$), to zakończ algorytm (T i T' są równoważne).
- R5.** [Przejście w prawo] Jeśli $\text{RTAG}(P) = 1 = \text{RTAG}(P')$, to $P \leftarrow \text{RLINK}(P)$, $P' \leftarrow \text{RLINK}(P')$, przejdź do kroku R4. Jeśli $\text{RTAG}(P) = 1 \neq \text{RTAG}(P')$, to zakończ algorytm ($T \prec T'$). Jeśli $\text{RTAG}(P) \neq 1 = \text{RTAG}(P')$, to zakończ algorytm ($T \succ T'$). W przeciwnym razie $P \leftarrow \text{RLINK}(P)$, $P' \leftarrow \text{RLINK}(P')$ i przejdź do kroku R2. ■

By udowodnić poprawność tego algorytmu (i przez to zrozumieć, jak działa) można pokazać przez indukcję ze względu na rozmiar drzewa T_0 , że prawdziwe jest następujące stwierdzenie: *Algorytm, startując w R2 z P i P' wskazującymi na korzeń dwóch niepustych drzew binarnych T_0 i T'_0 z fastrygą prawostronną, zakończy się, jeśli T_0 i T'_0 nie są równoważne, informując, czy $T_0 \prec T'_0$, czy $T_0 \succ T'_0$. Algorytm dojdzie do kroku R4, jeśli T_0 i T'_0 są równoważne, a P i P' będą wskazywać odpowiednio na następcy T_0 i T'_0 w porządku symetrycznym.*

28. Równoważne oraz podobne.

29. Udowodnij przez indukcję względem rozmiaru T , że następujące stwierdzenie jest prawdziwe: *Algorytm, startując w kroku C2 z P wskazującym na korzeń niepustego drzewa binarnego T i z Q wskazującym na węzeł o pustych obu poddrzewach, dojdzie do kroku C6 po przypisaniu $\text{INFO}(Q) \leftarrow \text{INFO}(P)$ i dołączeniu kopii lewego i prawego poddrzewa węzła $\text{NODE}(P)$ do $\text{NODE}(Q)$, z P i Q wskazującymi odpowiednio na następcy korzenia T i $\text{NODE}(Q)$ w porządku preorder.*

30. Założmy, że wskaźnik T w (2) jest równy $\text{LLINK}(\text{HEAD})$ w (10).

L1. [Inicjowanie] Przyjmij $Q \leftarrow \text{HEAD}$, $\text{RLINK}(Q) \leftarrow Q$.

L2. [Przesuwanie] Przyjmij $P \leftarrow Q$. (Zobacz poniżej).

L3. [Fastrygowanie] Jeśli $\text{RLINK}(Q) = \Lambda$, to przyjmij $\text{RLINK}(Q) \leftarrow P$, $\text{RTAG}(Q) \leftarrow 1$; w przeciwnym razie $\text{RTAG}(Q) \leftarrow 0$. Jeśli $\text{LLINK}(P) = \Lambda$, to $\text{LLINK}(P) \leftarrow Q$, $\text{LTAG}(P) \leftarrow 1$; w przeciwnym razie $\text{LTAG}(P) \leftarrow 0$.

L4. [Czy gotowe?] Jeśli $P \neq \text{HEAD}$, to przyjmij $Q \leftarrow P$ i powróć do kroku L2. ■

W kroku L2 aktywujemy współprogram przechodzenia drzewa, na przykład algorytm T z dodatkowym założeniem, że po przejściu drzewa odwiedzamy HEAD. Ta notacja jest wygodnym uproszczeniem w opisie algorytmów drzewowych, ponieważ nie musimy za każdym razem wytaczać maszynerii związanej ze stosem, ukrytej w algorytmie T. Oczywiście algorytmu S w kroku L2 nie można użyć, bo drzewo nie zostało jeszcze sfastrygowane. Ale algorytm z ćwiczenia 21 można wykorzystać, w wyniku czego otrzymamy doskonały sposób sfastrygowania drzewa bez korzystania z pomocniczego stosu.

X1. Przyjmij $P \leftarrow \text{HEAD}$.

X2. Przyjmij $Q \leftarrow P\$$ (korzystając na przykład z algorytmu S, przystosowanego do drzew z fastrygą prawostronną).

X3. Jeśli $P \neq \text{HEAD}$, to $\text{AVAIL} \leftarrow P$.

X4. Jeśli $Q \neq \text{HEAD}$, to przyjmij $P \leftarrow Q$ i wróć do X2.

X5. Przyjmij $\text{LLINK}(\text{HEAD}) \leftarrow \Lambda$.

Łatwo wskazać rozwiązania z mniejszą długością pętli wewnętrznej, ale kolejność podstawowych kroków algorytmu jest istotna. Podany algorytm działa dzięki temu, że nie zwracamy węzła na stertę, dopóki algorytm S nie przejdzie po obu jego dowiązaniach, LLINK i RLINK . Jak zauważono w tekście, podczas przechodzenia drzewa przez każde z tych dowiązań przechodzi się dokładnie raz.

32. $\text{RLINK}(Q) \leftarrow \text{RLINK}(P)$, $\text{SUC}(Q) \leftarrow \text{SUC}(P)$, $\text{SUC}(P) \leftarrow \text{RLINK}(P) \leftarrow Q$, $\text{PRED}(Q) \leftarrow P$, $\text{PRED}(\text{SUC}(Q)) \leftarrow Q$.

33. Wstawianie nowego węzła $\text{NODE}(Q)$ na lewo i w dół od węzła $\text{NODE}(P)$ jest proste: $\text{LLINKT}(Q) \leftarrow \text{LLINKT}(P)$, $\text{LLINK}(P) \leftarrow Q$, $\text{LTAG}(P) \leftarrow 0$, $\text{RLINK}(Q) \leftarrow \Lambda$. Wstawianie z prawej strony jest istotnie trudniejsze, ponieważ wymaga znalezienia $*Q$, które znaleźć równie trudno, jak $Q\$$ (zobacz ćwiczenie 19). Zapewne można skorzystać z metody przymieszczania węzłów omówionej w ćwiczeniu 23. Operacje wstawiania są trudniejsze przy takiej reprezentacji, ale algorytm C korzysta ze szczególnego typu operacji wstawiania. Okazuje się, że kopiowanie jest nawet nieco szybsze przy tym sposobie fastrygowania:

C1. Przyjmij $P \leftarrow \text{HEAD}$, $Q \leftarrow U$, idź do C4. (Opieramy się na założeniach i koncepcji algorytmu C z tekstu).

C2. Jeśli $\text{RLINK}(P) \neq \Lambda$, to przyjmij $R \leftarrow \text{AVAIL}$, $\text{LLINK}(R) \leftarrow \text{LLINK}(Q)$, $\text{LTAG}(R) \leftarrow 1$, $\text{RLINK}(R) \leftarrow \Lambda$, $\text{RLINK}(Q) \leftarrow \text{LLINK}(Q) \leftarrow R$.

C3. Przyjmij $\text{INFO}(Q) \leftarrow \text{INFO}(P)$.

C4. Jeśli $\text{LTAG}(P) = 0$, to przyjmij $R \leftarrow \text{AVAIL}$, $\text{LLINK}(R) \leftarrow \text{LLINK}(Q)$, $\text{LTAG}(R) \leftarrow 1$, $\text{RLINK}(R) \leftarrow \Lambda$, $\text{LLINK}(Q) \leftarrow R$, $\text{LTAG}(Q) \leftarrow 0$.

C5. Przyjmij $P \leftarrow \text{LLINK}(P)$, $Q \leftarrow \text{LLINK}(Q)$.

C6. Jeśli $P \neq \text{HEAD}$, to idź do C2. ■

Algorytm wygląda zbyt prosto, by mógł być poprawny! A jednak.

Algorytm C dla drzew binarnych z fastrygą lub fastrygą prawostronną potrzebuje dodatkowego czasu na obliczenie $P*$ i $Q*$ w kroku C5.

Można by zrealizować fastrygę tradycyjnie, za pomocą dowiązań RLINK albo w połączaniu z powyższą metodą kopiowania wstawić $\$P$ do $\text{RLINK}(P)$, odpowiednio ustawiając wartości $\text{RLINK}(R)$ i $\text{RLINK}(Q)$ w krokach C2 i C4.

34. **A1.** Przyjmij $Q \leftarrow P$, następnie wykonuj $Q \leftarrow \text{RLINK}(Q)$ zero lub więcej razy, aż $\text{RTAG}(Q) = 1$.

A2. Przyjmij $R \leftarrow \text{RLINK}(Q)$. Jeśli $\text{LLINK}(R) = P$, to $\text{LLINK}(R) \leftarrow \Lambda$. W przeciwnym razie $R \leftarrow \text{LLINK}(R)$, następnie wykonuj $R \leftarrow \text{RLINK}(R)$ zero lub więcej razy, aż $\text{RLINK}(R) = P$; na koniec przyjmij $\text{RLINKT}(R) \leftarrow \text{RLINKT}(Q)$. (W tym kroku usuwa się z drzewa węzeł $\text{NODE}(P)$ z poddrzewami).

A3. Przyjmij $\text{RLINK}(Q) \leftarrow \text{HEAD}$, $\text{LLINK}(\text{HEAD}) \leftarrow P$. ■

(Kluczem do wymyślenia i/lub zrozumienia tego algorytmu jest rzetelne przygotowanie schematów „przed i po”).

36. Nie; zobacz odpowiedź do ćwiczenia 1.2.1–15(e).

37. Jeśli w reprezentacji (2) $\text{LLINK}(P) = \text{RLINK}(P) = \Lambda$, to ustalamy $\text{LINK}(P) = \Lambda$; w przeciwnym razie $\text{LINK}(P) = Q$, gdzie węzeł $\text{NODE}(Q)$ odpowiada $\text{NODE}(\text{LLINK}(P))$, a $\text{NODE}(Q+1)$ odpowiada $\text{NODE}(\text{RLINK}(P))$. Warunek $\text{LLINK}(P)$ lub $\text{RLINK}(P) = \Lambda$ reprezentujemy za pomocą wartownika odpowiednio w $\text{NODE}(Q)$ lub $\text{NODE}(Q+1)$. Ta reprezentacja wymaga od n do $2n - 1$ lokacji. Przy podanym założeniu (2) wymagałoby 18 słów, a podany schemat 11 słów. Operacje wstawiania i usuwania są mniej więcej tak samo wydajne dla obu reprezentacji, lecz ta reprezentacja nie jest równie elastyczna w połączeniu z innymi strukturami.

2.3.2

1. Jeśli B jest puste, to $F(B)$ jest lasem pustym. W przeciwnym razie $F(B)$ składa się z drzewa T oraz lasu $F(\text{right}(B))$, gdzie $\text{root}(T) = \text{root}(B)$ oraz $\text{subtrees}(T) = F(\text{left}(B))$.

2. Liczba zer w notacji binarnej jest liczbą kropek w notacji dziesiętnej; dokładny wzór wyrażający odpowiedniość to

$$a_1.a_2.\dots.a_k \leftrightarrow 1^{a_1}01^{a_2-1}0\dots01^{a_k-1},$$

gdzie 1^a oznacza ciąg a jedynek.

3. Sortujemy leksykograficznie zapisy w notacji dziesiętnej Dewey'a (od lewej do prawej, jak w słowniku), umieszczając ciągi krótsze $a_1.\dots.a_k$ (i) przed ich rozszerzeniami $a_1.\dots.a_k.\dots.a_r$ dla porządku preorder, (ii) po ich rozszerzeniach dla porządku postorder. Jeśli zatem sortowaliśmy słowa, a nie ciągi liczb, to by otrzymać porządek preorder słowo *kat* umieścilibyśmy przed słowem *katarakta*, jak w słowniku; by otrzymać porządek postorder odwróilibyśmy kolejność sortowania przedrostków i najpierw umieścilibyśmy słowo *katarakta*, a potem *kat*. Te zasady łatwo udowodnić za pomocą indukcji względem rozmiaru drzewa.

4. Prawda; dowód przez indukcję względem liczby węzłów.

5. (a) Inorder. (b) Postorder. Sformułowanie dowodu indukcyjnego odpowiedniości porządków jest ciekawym zadaniem.

6. Mamy $\text{preorder}(T) = \text{preorder}(T')$ i $\text{postorder}(T) = \text{inorder}(T')$, nawet jeśli T ma węzły z tylko jednym dzieckiem. Pomiędzy pozostałymi dwoma porządkami nie zachodzi żaden prosty związek; na przykład korzeń T występuje w jednym przypadku na końcu, a w drugim przypadku mniej więcej po środku.

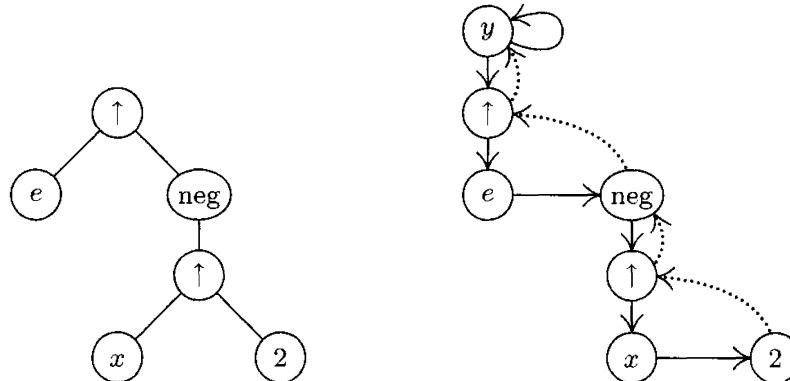
7. (a) Tak; (b) nie; (c) nie; (d) tak. Zauważ, że porządek odwrotny do preorder w przypadku lasu równa się porządkowi postorder swojego odbicia lustrzanego względem osi pionowej.

8. $T \preceq T'$ znaczy, że albo $\text{info}(\text{root}(T)) \prec \text{info}(\text{root}(T'))$, albo „info” są równe i zachodzi następujący warunek: przypuśćmy, że poddrzewami węzła $\text{root}(T)$ są T_1, \dots, T_n , a poddrzewami węzła $\text{root}(T')$ są $T'_1, \dots, T'_{n'}$ i niech $k \geq 0$ będzie największe, takie że T_j jest równoważne T'_j dla $1 \leq j \leq k$. Wówczas albo $k = n$, albo $k < n$ i $T_{k+1} \preceq T'_{k+1}$.

9. W niepustym lesie liczba węzłów wewnętrznych jest o jeden mniejsza niż liczba prawych dowiązań równych Λ , ponieważ puste prawe dowiązanie odpowiada skrajnie prawemu dziecku każdego węzła wewnętrznego oraz korzeniowi skrajnie prawego drzewa w lesie. (Ten fakt stanowi kolejny dowód tezy postawionej w ćwiczeniu 2.3.1–14, ponieważ liczba pustych wskaźników jest oczywiście równa liczbie liści).

10. Lasy są podobne wtedy i tylko wtedy, gdy $n = n'$ i $d(u_j) = d(u'_j)$, dla $1 \leq j \leq n$. Lasy są równoważne wtedy i tylko wtedy, gdy dodatkowo $\text{info}(u_j) = \text{info}(u'_j)$, $1 \leq j \leq n$. Dowód jest podobny; należy uogólnić lemat 2.3.1P. Weź $f(u) = d(u) - 1$.

11.



12. Jeżeli $\text{INFO}(Q1) \neq 0$: przyjmij $R \leftarrow \text{COPY}(P1)$; następnie jeżeli $\text{TYPE}(P2) = 0$ oraz $\text{INFO}(P2) \neq 2$, to przyjmij $R \leftarrow \text{TREE}(\text{"↑"}, \text{TREE}(\text{INFO}(P2) - 1))$; jeżeli $\text{TYPE}(P2) \neq 0$, to przyjmij $R \leftarrow \text{TREE}(\text{"↑"}, R, \text{TREE}(\text{"-"}, \text{COPY}(P2), \text{TREE}(1)))$; następnie przyjmij $Q1 \leftarrow \text{MULT}(Q1, \text{MULT}(\text{COPY}(P2), R))$.

Jeżeli $\text{INFO}(Q) \neq 0$: przyjmij $Q \leftarrow \text{TREE}(\text{"x"}, \text{MULT}(\text{TREE}(\text{"ln"}, \text{COPY}(P1)), Q), \text{TREE}(\text{"↑"}, \text{COPY}(P1), \text{COPY}(P2)))$.

Na koniec przejdź do DIFF[4].

13. Poniższy program realizuje algorytm 2.3.1C przy założeniach $rI1 \equiv P$, $rI2 \equiv Q$, $rI3 \equiv R$ z niezbędnymi modyfikacjami dotyczącymi warunków początkowych oraz zakończenia:

064	ST3 6F(0:2)	Przechowaj zawartość rI3, rI2.
065	ST2 7F(0:2)	<u>C1. Inicjowanie.</u>
066	ENT2 8F	Zacznij od utworzenia NODE(U)
067	JMP 1F	z RLINK(U) = Λ .
068	8H CON 0	Stała zero do inicjowania.
069	4H LD1 0,1(LLINK)	P \leftarrow LLINK(P) = P*.
070	1H LD3 AVAIL	R \leftarrow AVAIL.
071	J3Z OVERFLOW	
072	LDA 0,3(LLINK)	
073	STA AVAIL	
074	ST3 0,2(LLINK)	LLINK(Q) \leftarrow R.
075	ENNA 0,2	
076	STA 0,3(RLINKT)	RLINK(R) \leftarrow Q, RTAG(R) \leftarrow 1.
077	INCA 8B	rA \leftarrow LOC(nowy węzeł) - Q.
078	ENT2 0,3	Q \leftarrow R = Q*.
079	JAZ C3	Pierwszy raz, idź do C3.
080	C2 LDA 0,1	<u>C2. Coś na prawo?</u>
081	JAN C3	Skocz, jeśli RTAG(P) = 1.
082	LD3 AVAIL	R \leftarrow AVAIL.
083	J3Z OVERFLOW	
084	LDA 0,3(LLINK)	
085	STA AVAIL	
086	LDA 0,2(RLINKT)	
087	STA 0,3(RLINKT)	RLINKT(R) \leftarrow RLINKT(Q).
088	ST3 0,2(RLINKT)	RLINK(Q) \leftarrow R, RTAG(Q) \leftarrow 0.

089	C3	LDA	1, 1	<u>C3. Kopiowanie INFO.</u>
090		STA	1, 2	Pole INFO skopiowane.
091		LDA	0, 1(TYPE)	
092		STA	0, 2(TYPE)	Pole TYPE skopiowane.
093	C4	LDA	0, 1(LLINK)	<u>C4. Czy jest coś na lewo?</u>
094		JANZ	4B	Skocz, jeśli LLINK(P) $\neq \Lambda$.
095		STZ	0, 2(LLINK)	LLINK(Q) $\leftarrow \Lambda$.
096	C5	LD2N	0, 2(RLINKT)	<u>C5. Przesuwanie wskaźników.</u> Q $\leftarrow -RLINKT(Q)$.
097		LD1	0, 1(RLINK)	P $\leftarrow RLINK(P)$.
098		J2P	C5	Skocz, jeśli RTAG(Q) miało wartość 1.
099		ENN2	0, 2	Q $\leftarrow -Q$.
100	C6	J2NZ	C2	<u>C6. Sprawdzenie, czy to wszystko.</u>
101		LD1	8B(LLINK)	rI1 \leftarrow lokacja pierwszego utworzonego węzła.
102	6H	ENT3	*	Odtwórz rejesty indeksowe.
103	7H	ENT2	*	■

14. Niech a będzie liczbą kopiowanych węzłów wewnętrznych (operatorów). Liczby wykonań poszczególnych wierszy programu wyglądają następująco: 064–067, 1; 069, a ; 070–079, $a+1$; 080–081, $n-1$; 082–088, $n-1-a$; 089–094, n ; 095, $n-a$; 096–098, $n+1$; 099–100, $n-a$; 101–103, 1. Czas całkowity wynosi $(36n + 22)u$; około 20% czasu zajmuje pozyskiwanie nowych węzłów ze sterty, 40% przechodzenie drzewa i 40% kopiowanie informacji z pól INFO i LINK.

15. Uzupełnienie komentarzy pozostawiamy Czytelnikowi.

218	DIV	LDA	1, 6	227	JMP	COPYP2	236	ENTA	0, 1
219		JAZ	1F	228	ST1	1F(0:2)	237	ENT1	0, 5
220		JMP	COPYP2	229	ENTA	CON2	238	JMP	MULT
221		ENTA	SLASH	230	JMP	TREE0	239	ENTX	0, 1
222		ENTX	0, 6	231	ENTA	UPARROW	240	1H	ENT1 *
223		JMP	TREE2	232	1H	ENTX *	241	ENTA	SLASH
224		ENT6	0, 1	233	JMP	TREE2	242	JMP	TREE2
225	1H	LDA	1, 5	234	ST1	1F(0:2)	243	ENT5	0, 1
226		JAZ	SUB	235	JMP	COPYP1	244	JMP	SUB

16. Uzupełnienie komentarzy pozostawiamy Czytelnikowi.

245	PWR	LDA	1, 6	263	JMP	TREE0	281	ENTA	LOG
246		JAZ	4F	264	1H	ENTX *	282	JMP	TREE1
247		JMP	COPYP1	265	ENTA	MINUS	283	ENTA	0, 1
248		ST1	R(0:2)	266	JMP	TREE2	284	ENT1	0, 5
249		LDA	0, 3(TYPE)	267	5H	LDX	285	JMP	MULT
250		JANZ	2F	268	ENTA	UPARROW	286	ST1	1F(0:2)
251		LDA	1, 3	269	JMP	TREE2	287	JMP	COPYP1
252		DECA	2	270	ST1	R(0:2)	288	ST1	2F(0:2)
253		JAZ	3F	271	3H	JMP	289	JMP	COPYP2
254		INCA	1	272	ENTA	0, 1	290	2H	ENTX *
255		STA	CON0+1	273	R	ENT1 *	291	ENTA	UPARROW
256		ENTA	CON0	274	JMP	MULT	292	JMP	TREE2
257		JMP	TREE0	275	ENTA	0, 6	293	1H	ENTX *
258		STZ	CON0+1	276	JMP	MULT	294	ENTA	TIMES
259		JMP	5F	277	ENT6	0, 1	295	JMP	TREE2
260	2H	JMP	COPYP2	278	4H	LDA	1, 5	ENT5	0, 1
261		ST1	1F(0:2)	279	JAZ	ADD	296	JMP	ADD
262		ENTA	CON1	280	JMP	COPYP1	297		■

17. Bibliografia dotycząca pierwszych prac na ten temat znajduje się w przekrojowym artykule: J. Sammet, *CACM* 9 (1966), 555–569.

18. W pierwszej kolejności wykonujemy $\text{LLINK}[j] \leftarrow \text{RLINK}[j] \leftarrow j$ dla wszystkich j ; w ten sposób każdy węzeł staje się listą cykliczną o długości 1. Następnie dla $j = n, n-1, \dots, 1$ (w tej kolejności), jeśli $\text{PARENT}[j] = 0$, to przyjmujemy $r \leftarrow j$. W przeciwnym razie wstawiamy listę cykliczną zaczynającą się od j w listę cykliczną zaczynającą się od $\text{PARENT}[j]$ w następujący sposób: $k \leftarrow \text{PARENT}[j]$, $l \leftarrow \text{RLINK}[k]$, $i \leftarrow \text{LLINK}[j]$, $\text{LLINK}[j] \leftarrow k$, $\text{RLINK}[k] \leftarrow j$, $\text{LLINK}[l] \leftarrow i$, $\text{RLINK}[i] \leftarrow l$. To działa, ponieważ (a) każdy węzeł nie będący korzeniem zawsze jest poprzedzany przez swojego rodzica lub potomka swojego rodzica; (b) węzły każdej rodziny występują na liście swojego rodzica w porządku odpowiadającym ich lokacjom; (c) preorder jest jedynym porządkiem spełniającym (a) i (b).

20. Jeśli u jest przodkiem v , to na mocy indukcji mamy natychmiast, że u występuje przed v w porządku preorder i po v w porządku postorder. W drugą stronę, przypuśćmy, że u poprzedza v w porządku preorder i następuje po v w porządku postorder. Musimy pokazać, że u jest przodkiem v . Nie ma problemu, gdy u jest korzeniem pierwszego drzewa. Jeśli u jest innym węzłem pierwszego drzewa, to v też musi być w tym drzewie, ponieważ u występuje po v w porządku postorder, indukcja. Podobnie jeśli u nie jest węzłem pierwszego drzewa, to v też nie może być, ponieważ u poprzedza v w porządku preorder. (To twierdzenie łatwo wyprowadzić także z ćwiczenia 3. Dzięki temu można szybko sprawdzić relację „bycia przodkiem”, jeżeli dysponujemy pozycjami węzłów w porządkach preorder i postorder).

21. Jeśli $\text{NODE}(P)$ jest operatorem dwuargumentowym, to wskaźnikami do jego argumentów są $P_1 = \text{LLINK}(P)$ i $P_2 = \text{RLINK}(P_1) = \P . Algorytm D korzysta z faktu, że $P_2\$ = P$, zatem do wskaźnika $\text{RLINK}(P_1)$ można przypisać Q_1 , tj. wskaźnik do pochodnej wyrażenia $\text{NODE}(P_1)$. Wskaźnik $\text{RLINK}(P_1)$ jest ustawiany ponownie w kroku D3. Dla operatorów trójargumentowych mamy na przykład $P_1 = \text{LLINK}(P)$, $P_2 = \text{RLINK}(P_1)$, $P_3 = \text{RLINK}(P_2) = \P , zatem trudno uogólnić sztuczkę odpowiednią dla operatorów dwuargumentowych. Obliczywszy Q_1 , moglibyśmy póki co przypisać $\text{RLINK}(P_1) \leftarrow Q_1$, a po obliczeniu następnej pochodnej Q_2 przypisać $\text{RLINK}(Q_2) \leftarrow Q_1$ i $\text{RLINK}(P_2) \leftarrow Q_2$ oraz ponownie ustawić $\text{RLINK}(P_1) \leftarrow P_2$. To oczywiście jest mało eleganckie, a dla operatorów o większej arności będzie eleganckie jeszcze mniej. Z tego powodu tymczasowa zmiana $\text{RLINK}(P_1)$ w algorytmie D jest *sztuczką*, a nie *metodą*. Bardziej estetyczną metodę obsługi procesu różniczkowania, którą łatwo uogólnić na operatory o wyższej arności i w której nie korzysta się ze specyficznych sztuczek, można oprzeć na algorytmie 2.3.3F; zobacz ćwiczenie 2.3.3–3.

22. Z definicji wynika natychmiast, że relacja jest przechodnia, tj. jeśli $T \subseteq T'$ i $T' \subseteq T''$, to $T \subseteq T''$. (W istocie łatwo spostrzec, że relacja jest porządkiem częściowym). Niech f będzie funkcją przeprowadzającą węzły na nie same; wówczas $l(T) \subseteq T$ i $r(T) \subseteq T$. Stąd jeśli $T \subseteq l(T')$ lub $T \subseteq r(T')$, to musi być $T \subseteq T'$.

Przypuśćmy, że f_l i f_r są funkcjami wyznaczającymi relacje odpowiednio $l(T) \subseteq l(T')$ i $r(T) \subseteq r(T')$. Niech $f(u) = f_l(u)$, jeśli u leży w $l(T)$, $f(u) = \text{root}(T')$, jeśli u jest korzeniem T , w przeciwnym razie $f(u) = f_r(u)$. Widać, że f wyznacza $T \subseteq T'$. Na przykład, jeśli $r'(T)$ oznacza $r(T) \setminus \text{root}(T)$, to mamy $\text{preorder}(T) = \text{root}(T)$ $\text{preorder}(l(T))$ $\text{preorder}(r'(T))$; $\text{preorder}(T') = f(\text{root}(T))$ $\text{preorder}(l(T'))$ $\text{preorder}(r'(T'))$.

Twierdzenie odwrotne nie jest prawdziwe: wystarczy wziąć poddrzewa o korzeniach b i b' z rysunku 25.

2.3.3

1. Tak, moglibyśmy je odtworzyć w podobny sposób co (3) z (4), zamieniając miejscami LTAG i RTAG oraz LLINK i RLINK, a ponadto używając kolejki zamiast stosu.

2. W algorytmie F należy wprowadzić następujące zmiany: W kroku F1, „ostatni węzeł lasu w porządku preorder”. W kroku F2 w dwóch miejscach zmień „ $f(x_d), \dots, f(x_1)$ ” na „ $f(x_1), \dots, f(x_d)$ ”. W kroku F4: „Jeżeli P jest pierwszym węzłem w porządku preorder, to zakończ wykonanie algorytmu (stos będzie w tym momencie zawierał $f(\text{root}(T_1)), \dots, f(\text{root}(T_m))$, patrząc od wierzchołka w dół, gdzie T_1, \dots, T_m są drzewami w lesie, od lewej do prawej). W przeciwnym razie przypisz do P dowiązanie do poprzednika w porządku preorder (w naszej reprezentacji $P \leftarrow P - c$) i wróć do kroku F2”.

3. W kroku D1 należy dodatkowo wykonać $S \leftarrow \Lambda$. (S jest zmienną dowiązaniową zawierającą dowiązanie do wierzchołka stosu). Krok D2 przybierze na przykład postać: „Jeśli NODE(P) oznacza operator jednoargumentowy, to przyjmij $Q \leftarrow S$, $S \leftarrow \text{RLINK}(Q)$, $P_1 \leftarrow \text{LLINK}(P)$; jeśli oznacza operator dwuargumentowy, to przyjmij $Q \leftarrow S$, $Q_1 \leftarrow \text{RLINK}(Q)$, $S \leftarrow \text{RLINK}(Q_1)$, $P_1 \leftarrow \text{LLINK}(P)$, $P_2 \leftarrow \text{RLINK}(P_1)$. Następnie wykonaj $\text{DIFF}[\text{TYPE}(P)]$ ”. Krok D3 przybierze postać „ $\text{RLINK}(Q) \leftarrow S$, $S \leftarrow Q$ ”. Krok D4 przybierze postać „ $P \leftarrow P_s$ ”. Operacji $\text{LLINK}(D_Y) \leftarrow Q$ w kroku D5 można uniknąć, jeśli założymy, że $S \equiv \text{LLINK}(D_Y)$. Tę metodę łatwo uogólnić na operacje o trzech lub większej liczbie argumentów.

4. W reprezentacji (10) potrzeba $n - m$ pól LLINK i $n + (n - m)$ pól RLINK. Różnica w całkowitej liczbie dowiązań to $n - 2m$. Schemat (10) jest lepszy, gdy pola LLINK i INFO zajmują mniej więcej tyle samo miejsca i gdy m jest duże, tj. gdy węzły wewnętrzne mają duże stopnie.

5. Nie ma sensu umieszczać fastrygi w polu RLINK skoro wskazywałaby na ten sam węzeł co PARENT. Fastryga po LLINK, jak w 2.3.2-(4), mogłaby okazać się pozyteczna, jeżeli chcielibyśmy przesuwać się po drzewie w lewo, na przykład jeżeli chcielibyśmy przechodzić drzewo w porządku odwrotnym do postorder albo w porządku rodzinnym; ale te operacje nie są znaczaco trudniejsze bez fastrygi lewostronnej, chyba że węzły mają bardzo duże stopnie.

6. **L1.** Przyjmij $P \leftarrow \text{FIRST}$, $\text{FIRST} \leftarrow \Lambda$.

L2. Jeśli $P = \Lambda$, to zakończ wykonanie algorytmu. W przeciwnym razie $Q \leftarrow \text{RLINK}(P)$.

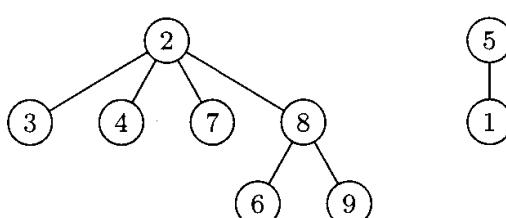
L3. Jeśli $\text{PARENT}(P) = \Lambda$, to $\text{RLINK}(P) \leftarrow \text{FIRST}$, $\text{FIRST} \leftarrow P$; w przeciwnym razie $R \leftarrow \text{PARENT}(P)$, $\text{RLINK}(P) \leftarrow \text{LCHILD}(R)$, $\text{LCHILD}(R) \leftarrow P$.

L4. Przyjmij $P \leftarrow Q$ i wróć do kroku L2. ■

7. $\{1, 5\}\{2, 3, 4, 7\}\{6, 8, 9\}$.

8. Wykonaj krok E3 algorytmu E, następnie sprawdź, czy $j = k$.

9. **PARENT**[k]: 5 0 2 2 0 8 2 2 8
 $k :$ 1 2 3 4 5 6 7 8 9



10. Jeden pomysł polega na tym, by w tablicy PARENT dla każdego korzenia przechowywać z ujemnym znakiem liczbę węzłów w drzewie (tę wartość daje się w łatwy sposób uaktualniać). Jeśli teraz w kroku E4 $|PARENT[j]| > |PARENT[k]|$, to należy zamienić rolami j i k . Ta metoda (autorstwa M. D. McIlroy'a) gwarantuje, że każda operacja zajmie co najwyżej $O(\log n)$ kroków.

Jeśli wciąż nam mało, możemy skorzystać z sugestii Alana Trittera: W kroku E4 wykonujemy $PARENT[x] \leftarrow k$ dla wszystkich wartości $x \neq k$ napotkanych w kroku E3. To wymaga dodatkowego przejścia w górę drzewa, ale powoduje jego spłaszczenie, zatem kolejne operacje wyszukiwania zajmą mniej czasu. (Zobacz punkt 7.4.1).

11. Wystarczy zdefiniować przekształcenie, które należy wykonać dla każdej wejściowej czwórki (P, j, Q, k) :

- T1.** Jeśli $PARENT(P) \neq \Lambda$, to przyjmij $j \leftarrow j + \text{DELTA}(P)$, $P \leftarrow PARENT(P)$ i powtórz ten krok.
- T2.** Jeśli $PARENT(Q) \neq \Lambda$, to przyjmij $k \leftarrow k + \text{DELTA}(Q)$, $Q \leftarrow PARENT(Q)$ i powtórz ten krok.
- T3.** Jeśli $P = Q$, to sprawdź, czy $j = k$ (w przeciwnym razie dane wejściowe zawierają sprzeczne deklaracje). Jeśli $P \neq Q$, to $\text{DELTA}(Q) \leftarrow j - k$, $PARENT(Q) \leftarrow P$, $LBD(P) \leftarrow \min(LBD(P), LBD(Q) + \text{DELTA}(Q))$ oraz $UBD(P) \leftarrow \max(UBD(P), UBD(Q) + \text{DELTA}(Q))$. ■

Uwaga: Można zezwolić, by deklaracje „ARRAY X[l:u]” były przeplatane deklaracjami równoważności lub zezwolić na przypisania adresów zmiennym, zanim inne zmienne uczynią się z nimi równoważnymi itp. Nietrudno wywnioskować warunki, które należy zachować, by dopuścić powyższe rozszerzenia. Więcej informacji na temat rozszerzeń podanego algorytmu znajdzie Czytelnik w CACM 7 (1964), 301–303, 506.

12. (a) Tak. (Jeśli nie wymagalibyśmy tego warunku, to moglibyśmy pominąć pętle przesuwające S w krokach A2 i A9). (b) Tak.

13. Kluczowa obserwacja polega na zauważeniu, że łańcuch dowiązań UP prowadzący w górę od P zawsze dotyczy tych samych wykładników odpowiednich zmiennych co analogiczny łańcuch prowadzący w górę od Q, z tym że łańcuch prowadzący od Q może zawierać dodatkowe elementy reprezentujące zmienne o wykładniku zero. (Ten warunek jest prawdziwy w większości kroków; wyjątkami są A9 i A10). Do kroku A8 dochodzimy albo z A3, albo z A10 i w obu przypadkach wiemy, że $\text{EXP}(Q) \neq 0$. Stąd $\text{EXP}(P) \neq 0$, a zatem w szczególności $P \neq \Lambda$, $Q \neq \Lambda$, $UP(P) \neq A$, $UP(Q) \neq \Lambda$; stąd wynika postawiona teza. Dowód sprowadza się zatem do pokazania, że warunek dotyczący łańcuchów dowiązań UP jest zachowywany przez operacje wykonywane przez algorytm na strukturze danych.

16. Dowodzimy (przez indukcję względem liczby węzłów w pojedynczym drzewie T), że jeśli P jest dowiązaniem do T i jeśli stos jest początkowo pusty, to kroki F2–F4 zakończą się pozostawieniem na stosie pojedynczej wartości $f(\text{root}(T))$. To jest prawda dla $n = 1$. Jeśli $n > 1$, to mamy $0 < d = \text{DEGREE}(\text{root}(T))$ poddrzew T_1, \dots, T_d ; z własności stosu oraz faktu, że porządek postorder to najpierw drzewa T_1, \dots, T_d , a potem $\text{root}(T)$, na mocy indukcji otrzymujemy wniosek, że algorytm oblicza $f(T_1), \dots, f(T_d)$, a następnie $f(\text{root}(T))$. Wynika stąd również poprawność algorytmu F dla lasów.

17. G1. Zainicjuj pusty stos. Niech P wskazuje na korzeń drzewa (ostatni węzeł w porządku postorder). Oblicz $f(\text{NODE}(P))$.

G2. Włóż na stos $\text{DEGREE}(P)$ kopii $f(\text{NODE}(P))$.

G3. Jeśli P jest pierwszym węzłem w porządku postorder, to zakończ wykonanie algorytmu. W przeciwnym razie przypisz do P dowiązanie do poprzednika w porządku postorder (w przypadku (9) to jest po prostu $P \leftarrow P - c$).

G4. Oblicz $f(\text{NODE}(P))$, korzystając z wartości znajdującej się na szczytce stosu, równej $f(\text{NODE}(\text{PARENT}(P)))$. Zdejmij tę wartość ze stosu i wróć do G2. ■

Uwaga: Można skonstruować analogiczny algorytm korzystający z porządku preorder zamiast postorder, jak w ćwiczeniu 2. W istocie można zastosować także porządek rodzinny lub porządek poziomy. W przypadku porządku poziomego zamiast stosu powinniśmy użyć kolejki.

18. Tablice INFO1 i RLINK , zgodnie z sugestią z tekstu dotyczącą obliczania znacznika LTAG , można traktować jak drzewo binarne w tradycyjnej reprezentacji. Pomysł polega na przejściu tego drzewa w porządku postorder i zliczaniu stopni:

P1. Niech R , D i I będą początkowo pustymi stosami. Przypisujemy $R \leftarrow n + 1$, $D \leftarrow 0$, $j \leftarrow 0$, $k \leftarrow 0$.

P2. Jeśli $\text{top}(R) > j + 1$, to przejdź do P5. (Jeśli dysponujemy polem LTAG , to wystarczy sprawdzić $\text{LTAG}[j] = 0$ zamiast $\text{top}(R) > j + 1$).

P3. Jeśli I jest pusty, to zakończ wykonanie algorytmu; w przeciwnym razie przyjmij $i \leftarrow I$, $k \leftarrow k + 1$, $\text{INFO2}[k] \leftarrow \text{INFO}[i]$, $\text{DEGREE}[k] \leftarrow D$.

P4. Jeśli $\text{RLINK}[i] = 0$, to przejdź do kroku P3; w przeciwnym razie usuń wierzchołek stosu R (równy $\text{RLINK}[i]$).

P5. Przypisz $\text{top}(D) \leftarrow \text{top}(D) + 1$, $j \leftarrow j + 1$, $I \leftarrow j$, $D \leftarrow 0$ i jeśli $\text{RLINK}[j] \neq 0$, to $R \leftarrow \text{RLINK}[j]$. Przejdź do kroku P2. ■

19. (a) Równoważnie możemy powiedzieć, że dowiązania SCOPE się nie „krzyżują”.
(b) Pierwsze drzewo w lesie zawiera $d_1 + 1$ elementów, dalej możemy korzystać z indukcji. (c) Biorąc wartości najmniejsze, zachowujemy warunki (a).

Uwagi: Na mocy ćwiczenia 2.3.2–20 ciąg $d_1 d_2 \dots d_n$ można także zinterpretować za pomocą inwersji: jeśli k -ty węzeł w porządku postorder jest p_k -tym węzłem w porządku preorder, to d_k jest liczbą elementów $> k$ występujących na lewo od k w $p_1 p_2 \dots p_n$.

Podobny schemat, w którym wypisujemy liczby potomków każdego węzła w lesie w porządku *postorder*, daje ciąg liczb $c_1 c_2 \dots c_n$, charakteryzowany przez następujące warunki: (i) $0 \leq c_k < k$, (ii) $k \geq j \geq k - c_k$ pociąga za sobą $j - c_j \geq k - c_k$. Algorytmy oparte na takich ciągach badał J. M. Pallo, *Comp. J.* **29** (1986), 171–175. Zauważmy, że c_k jest rozmiarem lewego poddrzewa k -tego węzła w porządku symetrycznym drzewa binarnego reprezentującego drzewo w naturalny sposób. Możemy także interpretować d_k jako rozmiar *prawego* poddrzewa k -tego węzła odpowiedniego drzewa binarnego w porządku symetrycznym, tj. drzewa binarnego odpowiadającego danemu lasowi według dualnej metody przedstawionej w ćwiczeniu 2.3.2–5.

Zależność $d_k \leq d'_k$ dla $1 \leq k \leq n$ wyznacza interesujący porządek kratowy dla lasów i drzew binarnych, wprowadzony po raz pierwszy w nieco innej postaci przez D. Tamari'ego [Thèse (Paris: 1951)]; zobacz ćwiczenie 6.2.3–32.

2.3.4.1

1. (B, A, C, D, B) , (B, A, C, D, E, B) , (B, D, C, A, B) , (B, D, E, B) , (B, E, D, B) , (B, E, D, C, A, B) .

2. Niech (V_0, V_1, \dots, V_n) będzie najkrótszą ścieżką z V do V' . Jeśli dla pewnego $j < k$ byłoby $V_j = V_k$, to ścieżka $(V_0, \dots, V_j, V_{k+1}, \dots, V_n)$ byłaby krótsza.

3. (Ścieżka podstawowa przechodzi po e_3 i e_4 , ale w ramach obchodzenia cyklu C_2 przechodzimy po każdej z tych krawędzi -1 razy, co w sumie daje zero). Odpowiedź: $e_1, e_2, e_6, e_7, e_9, e_{10}, e_{11}, e_{12}, e_{14}$.

4. Przypuśćmy, że nie. Niech G'' będzie podgrafem G' otrzymanym przez usunięcie z G' tych krawędzi, dla których $E_j = 0$. G'' jest skończonym grafem bez cykli o przynajmniej jednej krawędzi, zatem z rozumowania przeprowadzonego przy okazji dowodu twierdzenia A wynika, że istnieje przynajmniej jeden wierzchołek V sąsiadujący z innym wierzchołkiem V' . Niech e_j będzie krawędzią prowadzącą z V do V' . Wówczas z równania Kirchhoffa (1) dla wierzchołka V wynika, że $E_j = 0$, co jest sprzeczne z konstrukcją G'' .

5. $A = 1 + E_8, B = 1 + E_8 - E_2, C = 1 + E_8, D = 1 + E_8 - E_5, E = 1 + E_{17} - E_{21}, F = 1 + E''_{13} + E_{17} - E_{21}, G = 1 + E''_{13}, H = E_{17} - E_{21}, J = E_{17}, K = E''_{19} + E_{20}, L = E_{17} + E''_{19} + E_{20} - E_{21}, P = E_{17} + E_{20} - E_{21}, Q = E_{20}, R = E_{17} - E_{21}, S = E_{25}$. Uwaga: W tym przypadku można również rozwiązać układ względem niewiadomych E_2, E_5, \dots, E_{25} i parametrów A, B, \dots, S . Zatem niezależnych rozwiązań jest dziewięć, skąd widać, dlaczego wyeliminowaliśmy sześć zmiennych w 1.3.3-(8).

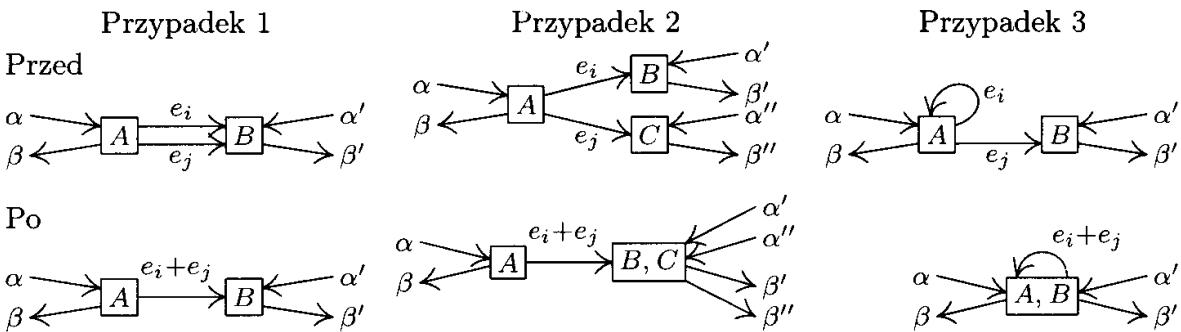
6. (Poniższe rozwiązanie opiera się na pomyśle, że możemy wypisać każdą krawędź, która nie tworzy cyklu z wypisanymi krawędziami). Korzystamy z algorytmu 2.3.3E, każdą parę (a_i, b_i) traktując tak, jak $a_i \equiv b_i$ (porównaj notację stosowaną do zapisu algorytmu 2.3.3E). Jedyna zmiana polega na tym, że wypisujemy (a_i, b_i) , jeśli w kroku E4 $j \neq k$.

By pokazać, że algorytm jest poprawny, musimy udowodnić, że (a) algorytm nie wypisuje krawędzi, które tworzą cykl, oraz (b) jeśli G zawiera przynajmniej jedno wolne poddrzewo, to algorytm wypisuje $n-1$ krawędzi. Niech $j \equiv k$ oznacza, że istnieje ścieżka od V_j do V_k lub $j = k$. To jest oczywiście relacja równoważności, a ponadto $j \equiv k$ wtedy i tylko wtedy, gdy ten związek da się wyprowadzić ze związków $a_1 \equiv b_1, \dots, a_m \equiv b_m$. Warunek (a) zachodzi, ponieważ algorytm nie wypisuje krawędzi tworzących cykl z poprzednio wypisanymi krawędziami. Warunek (b) zachodzi, ponieważ jeśli wszystkie wierzchołki są równoważne, to $\text{PARENT}[k] = 0$ dla dokładnie jednego k .

Bardziej wydajny algorytm można oprzeć na przeszukiwaniu w głąb; zobacz algorytm 2.3.5A oraz punkt 7.4.1.

7. Cykle podstawowe: $C_0 = e_0 + e_1 + e_4 + e_9$ (ścieżka podstawowa: $e_1 + e_4 + e_9$); $C_5 = e_5 + e_3 + e_2$; $C_6 = e_6 - e_2 + e_4$; $C_7 = e_7 - e_4 - e_3$; $C_8 = e_8 - e_9 - e_4 - e_3$. Stąd $E_1 = 1, E_2 = E_5 - E_6, E_3 = E_5 - E_7 - E_8, E_4 = 1 + E_6 - E_7 - E_8, E_9 = 1 - E_8$.

8. W każdym kroku redukując łączymy dwie strzałki e_i i e_j wychodzące z tego samego bloku. Wystarczy udowodnić, że taki krok można odwrócić. Mając daną wartość $e_i + e_j$ po połączeniu strzałek, musimy wyznaczyć prawidłowe wartości e_i i e_j przed połączeniem. Są trzy istotnie różne przypadki:



A, B i C oznaczają wierzchołki lub megawierzchołki, a α i β oznaczają okoliczne przepływy. Na każdy z tych przepływów może składać się kilka krawędzi, ale bez

straty ogólności możemy założyć, że krawędź jest jedna. W przypadku 1 (e_i i e_j prowadzą do tego samego bloku), możemy wybrać e_i dowolnie, a następnie przypisać $e_j \leftarrow (e_i + e_j) - e_i$. W przypadku 2 (e_i i e_j prowadzą do różnych bloków) musimy wziąć $e_i \leftarrow \beta' - \alpha'$, $e_j \leftarrow \beta'' - \alpha''$. W przypadku 3 (e_i tworzy pętlę, ale e_j nie) musimy wziąć $e_j \leftarrow \beta' - \alpha'$, $e_i \leftarrow (e_i + e_j) - e_j$. W każdym przypadku daje się odwrócić krok redukcji.

Wynik tego ćwiczenia dowodzi, że liczba cykli podstawowych w zredukowanym schemacie blokowym jest minimalną liczbą przepływów wierzchołkowych, które trzeba określić, by wyznaczyć pozostałe. W podanym przykładzie z analizy zredukowanego schematu blokowego wynika, że wystarczy określić przepływy w trzech wierzchołkach (na przykład a, c, d), podczas gdy w pierwotnym schemacie z ćwiczenia 7 były cztery niezależne przepływy krawędziowe. Każde wystąpienie przypadku 1 podczas redukcji oznacza zaoszczędzenie jednego pomiaru.

Podobną metodę redukcji można oprzeć na łączeniu strzałek *wchodzących* do bloku. Można pokazać, że uzyskamy w ten sposób taki sam zredukowany schemat blokowy, różniący się jedynie nazwami megawierzchołków.

Konstrukcja podana w tym ćwiczeniu jest oparta na pomysłach Armena Nahapetiana i F. Stevenson. Więcej informacji można znaleźć w: A. Nahapetian, *Acta Informatica* **3** (1973), 37–41; D. E. Knuth, F. Stevenson, *BIT* **13** (1973), 313–322.

9. Każdą krawędź prowadzącą od wierzchołka do niego samego traktujemy jako samostny „cykl podstawowy”. Jeśli pomiędzy wierzchołkami V i V' jest $k+1$ krawędzi $e, e', \dots, e^{(k)}$, to tworzymy k cykli podstawowych $e' \pm e, \dots, e^{(k)} \pm e$ (wybierając + lub – w zależności od tego, czy krawędzie prowadzą w tych samych, czy przeciwnych kierunkach) i dalej postępujemy tak, jakby między wierzchołkami była tylko krawędź e .

W zasadzie sytuacja byłaby znacznie bardziej przejrzysta, gdybyśmy definiując graf, dopuścili możliwość występowania krawędzi wielokrotnych oraz pętli (krawędzi o obu końcach w tym samym wierzchołku). Ścieżki i cykle należałyby wówczas zdefiniować względem krawędzi, a nie wierzchołków. Taką definicję w istocie proponujemy dla grafów skierowanych w podpunkcie 2.3.4.2.

10. Skoro wszystkie wyprowadzenia muszą zostać połączone, to graf połączeń musi być spójny. Najbardziej oszczędny układ przewodów nie będzie zawierał cykli, zatem musi być drzewem wolnym. Na mocy twierdzenia A drzewo wolne zawiera $n-1$ przewodów, a graf o n wierzchołkach i $n-1$ krawędziach jest drzewem wolnym wtedy i tylko wtedy, gdy jest spójny.

11. Wystarczy udowodnić, że jeśli $n > 1$ i $c(n-1, n)$ jest najmniejsze z $c(i, n)$, to istnieje przynajmniej jedno drzewo o minimalnym koszcie, w którym T_{n-1} jest połączony przewodem z T_n . (Bo dowolne drzewo o minimalnym koszcie mające $n > 1$ wierzchołków, w którym T_{n-1} jest połączony przewodem z T_n , musi być także drzewem o minimalnym koszcie mającym $n-1$ wierzchołków, gdy T_{n-1} i T_n uznamy za „sklejone” zgodnie z konwencją przyjętą w opisie algorytmu).

By udowodnić powyższe twierdzenie, przypuśćmy, że mamy drzewo o minimalnym koszcie, w którym T_{n-1} nie jest połączone przewodem z T_n . Jeśli dołożymy przewód $T_{n-1} — T_n$, to otrzymamy cykl, więc będziemy mogli usunąć dowolny przewód z tego cyklu. W wyniku usunięcia innego przewodu przyłączonego do T_n otrzymamy inne drzewo, którego całkowity koszt jest nie większy niż koszt pierwotnego drzewa, w którym ponadto jest krawędź $T_{n-1} — T_n$.

12. Należy przechowywać dwie pomocnicze tablice $a(i)$ i $b(i)$, $1 \leq i < n$, reprezentujące fakt, że jednym z uprzednio wybranych wyprowadzeń, które daje najmniejszy koszt przy połączeniu z wyprowadzeniem T_i , jest wyprowadzenie $T_{b(i)}$, a koszt połączenia

wynosi $a(i)$. Początkowo $a(i) = c(i, n)$ i $b(i) = n$. Następnie $n - 1$ razy powtarzamy następujące operacje: znajdź takie i , że $a(i) = \min_{1 \leq j < n} a(j)$; połącz T_i z $T_{b(i)}$; dla $1 \leq j < n$, jeśli $c(i, j) < a(j)$, to $a(j) \leftarrow c(i, j)$ i $b(j) \leftarrow i$; przypisz $a(i) \leftarrow \infty$. Tutaj $c(i, j)$ oznacza $c(j, i)$, gdy $j < i$.

(Nieco bardziej wydajna metoda polega na pozbyciu się ∞ i utrzymywaniu w zamian listy liniowej tych j , które nie zostały jeszcze wybrane. Jednakże zarówno z tym usprawnieniem, jak i bez niego, algorytm wykonuje $O(n^2)$ operacji). Zobacz także: E. W. Dijkstra, Proc. Nederl. Akad. Wetensch. A63 (1960), 196–199; D. E. Knuth, The Stanford GraphBase (New York: ACM Press, 1994), 460–497. Istotnie lepsze algorytmy znajdowania drzewa rozpinającego o minimalnym koszcie omawiamy w punkcie 7.5.4.

13. Jeżeli dla pewnych $i \neq j$ nie istnieje ścieżka z V_i do V_j , to żadne złożenie transpozycji nie przeprowadzi i na j . Jeśli zatem generowane są wszystkie permutacje, to graf musi być spójny. W drugą stronę, jeśli graf jest spójny, to jeśli trzeba, usuńmy tyle krawędzi, by otrzymać drzewo. Przenumerujmy następnie wierzchołki w taki sposób, że V_n sąsiaduje z dokładnie jednym innym wierzchołkiem V_{n-1} . (Zobacz dowód twierdzenia A). Transpozycje różne od $(n-1\ n)$ tworzą drzewo o $n - 1$ wierzchołkach, zatem na mocy indukcji, jeśli π jest dowolną permutacją zbioru $\{1, 2, \dots, n\}$ o punkcie stałym n , to π można zapisać jako złożenie tych transpozycji. Jeśli π przeprowadza n na j , to $\pi(n-1\ n)(n-1\ n) = \rho$ ma punkt stały n ; stąd $\pi = \rho(n-1\ n)(j\ n-1)$ można zapisać jako złożenie danych transpozycji.

2.3.4.2

1. Niech (e_1, \dots, e_n) będzie najkrótszą ścieżką zorientowaną z V do V' . Jeśli $\text{init}(e_j) = \text{init}(e_k)$ dla $j < k$, to ścieżka $(e_1, \dots, e_{j-1}, e_k, \dots, e_n)$ byłaby krótsza. Z podobnego rozumowania korzystamy dla $\text{fin}(e_j) = \text{fin}(e_k)$, gdzie $j < k$. Stąd ścieżka (e_1, \dots, e_n) jest prosta.

2. Te, w których wszystkie znaki są jednakowe: $C_0, C_8, C''_{13}, C_{17}, C''_{19}, C_{20}$.

3. Narysuj na przykład trzy wierzchołki A, B, C oraz krawędzie od A do B i od A do C .

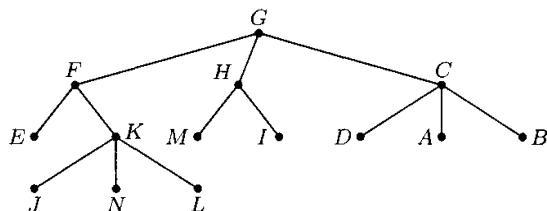
4. Jeśli nie ma cykli zorientowanych, to algorytm 2.2.3T sortuje topologicznie G . Jeżeli jest cykl zorientowany, to posortowanie topologiczne jest niemożliwe. (Dodatkowo przy jednej z możliwych interpretacji polecenia należy pominąć cykle długości 1).

5. Niech k będzie najmniejszą liczbą całkowitą, taką że $\text{fin}(e_k) = \text{init}(e_j)$ dla pewnego $j \leq k$. Wówczas (e_j, \dots, e_k) jest cyklem zorientowanym.

6. Fałsz (z powodu drobiazgu technicznego), bo może być wiele krawędzi między dwoma wierzchołkami.

7. Prawda dla skończonych grafów skierowanych. Jeśli weźmiemy dowolny wierzchołek V i będziemy posuwać się wzduż jedynej ścieżki zorientowanej, to nigdy nie napotkamy powtórnie żadnego wierzchołka, zatem musimy w końcu natknąć się na R (jedyny wierzchołek bez poprzednika). Dla nieskończonych grafów skierowanych stwierdzenie jest oczywiście fałszem, bo możemy mieć wierzchołki R, V_1, V_2, V_3, \dots oraz krawędzie od V_j do V_{j+1} dla $j \geq 1$.

9. Wszystkie strzałki wskazują w górę.



10. **G1.** Przyjmij $k \leftarrow P[j]$, $P[j] \leftarrow 0$.

G2. Jeśli $k = 0$, to się zatrzymaj; w przeciwnym razie $m \leftarrow P[k]$, $P[k] \leftarrow j$, $j \leftarrow k$, $k \leftarrow m$ i powtórz krok G2. ■

11. Poniższy algorytm jest połączeniem algorytmu 2.3.3E oraz metody podanej w poprzednim ćwiczeniu; wszystkie krawędzie drzew zorientowanych odpowiadają krawędziom w grafie skierowanym; $S[j]$ jest tablicą pomocniczą, w której przechowujemy informacje, czy krawędź prowadzi z j do $P[j]$ ($S[j] = +1$), czy z $P[j]$ do j ($S[j] = -1$). Początkowo $P[1] = \dots = P[n] = 0$. W poniższych krokach następuje przetwarzanie jednej krawędzi (a, b) :

C1. Przyjmij $j \leftarrow a$, $k \leftarrow P[j]$, $P[j] \leftarrow 0$, $s \leftarrow S[j]$.

C2. Jeśli $k = 0$, to idź do C3; w przeciwnym razie przyjmij $m \leftarrow P[k]$, $t \leftarrow S[k]$, $P[k] \leftarrow j$, $S[k] \leftarrow -s$, $s \leftarrow t$, $j \leftarrow k$, $k \leftarrow m$ i powtórz krok C2.

C3. (a jest korzeniem swojego drzewa). Przyjmij $j \leftarrow b$, następnie jeśli $P[j] \neq 0$, to przypisz $j \leftarrow P[j]$, aż $P[j] = 0$.

C4. Jeśli $j = a$, to przejdź do C5; w przeciwnym razie przyjmij $P[a] \leftarrow b$, $S[a] \leftarrow +1$, wypisz (a, b) jako krawędź należącą do wolnego poddrzewa i zakończ wykonanie algorytmu.

C5. Wypisz „CYCLE” a następnie „ (a, b) ”.

C6. Jeśli $P[b] = 0$, to zakończ wykonanie algorytmu. W przeciwnym razie, jeśli $S[b] = +1$, to wypisz „ $+(b, P[b])$ ”; w przeciwnym razie wypisz „ $-(P[b], b)$ ”; przypisz $b \leftarrow P[b]$ i powtórz krok C6. ■

Uwaga: Jeśli skorzystamy z sugestii McIlroya podanej w odpowiedzi do ćwiczenia 2.3.3–10, to wykonanie powyższego algorytmu zajmie $O(m \log n)$. Istnieje jednak o wiele lepsze rozwiążanie wymagające tylko $O(m)$ kroków: za pomocą przeszukiwania w głąb skonstruuj „drzewo palmowe” z jednym cyklem podstawowym dla każdego liścia „palmowego” [R. E. Tarjan, *SICOMP* 1 (1972), 146–150].

12. Stopniem wejściowym; stopień wyjściowy każdego wierzchołka jest równy albo zero, albo jeden.

13. Zdefiniujmy ciąg poddrzew zorientowanych grafu G : G_0 jest samym korzeniem. G_{k+1} to G_k plus dowolny wierzchołek V grafu G , który nie należy do G_k , ale dla którego istnieje krawędź od V do V' , taka że V' należy do G_k , plus jedna krawędź $e[V]$ dla każdego takiego wierzchołka. Na mocy indukcji otrzymujemy natychmiast, że G_k jest drzewem zorientowanym dla wszystkich $k \geq 0$ oraz że jeśli istnieje ścieżka zorientowana długości k prowadząca od V do R w G , to V należy do G_k . Stąd G_∞ , zbiór wszystkich V i $e[V]$ we wszystkich grafach G_k jest poszukiwanym zorientowanym poddrzewem grafu G .

14. $(e_{12}, e_{20}, e_{00}, e'_{01}, e_{10}, e_{01}, e'_{12}, e_{22}, e_{21})$, $(e_{12}, e_{20}, e_{00}, e'_{01}, e'_{12}, e_{22}, e_{21}, e_{10}, e_{01})$,
 $(e_{12}, e_{20}, e'_{01}, e_{10}, e_{00}, e_{01}, e'_{12}, e_{22}, e_{21})$, $(e_{12}, e_{20}, e'_{01}, e'_{12}, e_{22}, e_{21}, e_{10}, e_{00}, e_{01})$,
 $(e_{12}, e_{22}, e_{20}, e_{00}, e'_{01}, e_{10}, e_{01}, e'_{12}, e_{21})$, $(e_{12}, e_{22}, e_{20}, e_{00}, e'_{01}, e'_{12}, e_{21}, e_{10}, e_{01})$,
 $(e_{12}, e_{22}, e_{20}, e'_{01}, e_{10}, e_{00}, e_{01}, e'_{12}, e_{21})$, $(e_{12}, e_{22}, e_{20}, e'_{01}, e'_{12}, e_{21}, e_{10}, e_{00}, e_{01})$,

(w porządku leksykograficznym). Osiem możliwości bierze się stąd, że niezależnie możemy wybierać czy e_{00} poprzedza e'_{01} , czy e_{10} poprzedza e'_{12} oraz czy e_{20} poprzedza e_{22} .

15. Prawda: jeśli jest spójny i zrównoważony i ma więcej niż jeden wierzchołek, to ma cykl Eulera, który obchodzi wszystkie wierzchołki.

16. Rozważmy taki graf skierowany G o wierzchołkach V_1, \dots, V_{13} , który ma krawędzie od V_j do V_k dla każdej karty k w kupce j . Zakończenie pasjansa sukcesem jest równoważne przejściu cyklu Eulera w tym grafie (bo jeśli pasjans „wyszedł”, to ostatnia odkryta karta musiała pochodzić ze środkowej kupki; ten graf jest zrównoważony). Jeśli zatem pasjans wyszedł, to na mocy lematu E graf podany w sformułowaniu ćwiczenia jest zorientowanym poddrzewem grafu G . W drugą stronę, jeśli podany graf skierowany jest drzewem zorientowanym, to pasjans wychodzi na mocy twierdzenia D.

17. $\frac{1}{13}$. Tę odpowiedź można otrzymać (tak jak za pierwszym razem otrzymał ją autor) przez żmudne zliczanie pewnych drzew zorientowanych za pomocą funkcji tworzących itp., korzystając z metod przedstawionych w podpunkcie 2.3.4.4. Można jednak o wiele prościej. Definiujemy porządek odkrywania *wszystkich* kart w talii w sposób następujący: dopóki się da, postępujemy zgodnie z regułami pasjansa, a gdy się nie da – „oszukujemy”, odkrywając pierwszą kartę, którą się da (znajdujemy pierwszą niepustą kupkę, idąc od kupki numer 1 zgodnie z ruchem wskazówek zegara) i kontynuujemy, aż do odwrócenia wszystkich kart. Karty w porządku odkrywania są ułożone losowo (ponieważ wartość karty nie musi być znana aż do chwili jej odkrycia). Problem sprowadza się zatem do obliczenia prawdopodobieństwa zdarzenia, że w losowo potasowanej talii ostatnią kartą jest król. Bardziej ogólnie, prawdopodobieństwo zdarzenia, że k kart jest zakrytych, gdy nie można wykonać ruchu, jest prawdopodobieństwem zdarzenia, że za ostatnim królem w losowo potasowanej talii jest jeszcze k kart, co jest równe $4! \binom{51-k}{3} \frac{48!}{52!}$. Stąd osoba układająca pasjansa bez oszukiwania będzie odkrywać średnio 42.4 karty na rozłożenie. *Uwaga:* Na podobnej zasadzie można pokazać, iż prawdopodobieństwo zdarzenia, że układający będzie musiał „oszukać” k razy jest zadane dokładnie liczbą Stirlinga $\left[\begin{smallmatrix} 13 \\ k+1 \end{smallmatrix} \right] / 13!$. (Zobacz wzór 1.2.10–(9) oraz ćwiczenie 1.2.10–7; przypadek uogólnionej talii pojawia się w ćwiczeniu 1.2.10–18).

18. (a) Jeśli istnieje cykl (V_0, V_1, \dots, V_k) , w którym koniecznie $3 \leq k \leq n$, to suma k wierszy A odpowiadających k krawędziom tego cyklu (z odpowiednimi znakami) jest wierszem zerowym. Jeśli zatem G nie jest drzewem wolnym, to wyznacznik A_0 jest równy零.

Ale jeśli G jest drzewem wolnym, to możemy spojrzeć nań jak na drzewo uporządkowane o korzeniu V_0 i możemy przestawić wiersze i kolumny A_0 w taki sposób, że kolumny są w porządku preorder, a k -ty wiersz odpowiada krawędzi od k -tego wierzchołka (kolumny) do jego rodzica. Macierz staje się wówczas macierzą trójkątną, a na jej przekątnej występuje wyłącznie ± 1 , zatem wyznacznik jest równy ± 1 .

(b) Na mocy wzoru Bineta–Cauchy’ego (ćwiczenie 1.2.3–46) mamy

$$\det A_0^T A_0 = \sum_{1 \leq i_1 < \dots < i_n \leq m} (\det A_{i_1 \dots i_n})^2$$

gdzie $A_{i_1 \dots i_n}$ oznacza macierz składającą się z wierszy i_1, \dots, i_n macierzy A_0 (odpowiadającą zatem wyborowi n krawędzi grafu G). Stosując (a), otrzymujemy poszukiwany wynik.

[Zobacz S. Okada, R. Onodera, *Bull. Yamagata Univ.* **2** (1952), 89–117].

19. (a) Warunki $a_{00} = 0$ i $a_{jj} = 1$ to po prostu warunki (a) i (b) z definicji drzewa zorientowanego. Jeśli G nie jest drzewem zorientowanym, to (na mocy ćwiczenia 7) istnieje cykl zorientowany, a wiersze A_0 odpowiadające wierzchołkom tego cyklu sumują się, w wyniku dając wiersz zerowy, stąd $\det A_0 = 0$. Jeśli G jest drzewem zorientowanym, to ustalmy dowolny porządek dzieci każdego wierzchołka i spójrzmy na G jak na drzewo uporządkowane. Permutujemy teraz wiersze i kolumny A_0 , tak

by odpowiadały porządkowi preorder wierzchołków. Ponieważ tę samą permutację stosujemy do wierszy i kolumn, wartość wyznacznika się nie zmienia. Otrzymujemy macierz trójkątną z wartościami +1 na przekątnej.

(b) Możemy założyć, że dla wszystkich j jest $a_{0j} = 0$, ponieważ żadna krawędź wychodząca z V_0 nie może należeć do poddrzewa zorientowanego. Możemy także założyć, że dla wszystkich $j \geq 1$ jest $a_{jj} > 0$, ponieważ w przeciwnym przypadku cały j -ty wiersz byłby zerowy i oczywiście nie byłoby zorientowanych poddrzew. Korzystamy teraz z indukcji względem liczby krawędzi: jeśli $a_{jj} > 1$, to niech e będzie pewną krawędzią wychodzącą z V_j ; niech B_0 będzie macierzą taką jak A_0 , ale z usuniętą krawędzią e , a C_0 niech będzie macierzą taką jak A_0 , ale z usuniętymi wszystkimi krawędziami wychodzącymi z V_j oprócz e . Przykład: Jeśli $A_0 = \begin{pmatrix} 3 & -2 \\ -1 & 2 \end{pmatrix}$, $j = 1$, a e jest krawędzią prowadzącą od V_1 do V_0 , to $B_0 = \begin{pmatrix} 2 & -2 \\ -1 & 2 \end{pmatrix}$, $C_0 = \begin{pmatrix} 1 & 0 \\ -1 & 2 \end{pmatrix}$. W ogólności mamy $\det A_0 = \det B_0 + \det C_0$, ponieważ macierze są jednakowe we wszystkich wierszach poza j -tym, a A_0 w tym wierszu jest sumą B_0 i C_0 . Ponadto liczba zorientowanych poddrzew G jest liczbą poddrzew, które *nie zawierają* e (na mocy indukcji równej $\det B_0$) plus liczba tych, które *zawierają* e (na mocy indukcji równa $\det C_0$).

Uwagi: Macierz A jest często nazywana laplasjanem grafu przez analogię do podobnego pojęcia w teorii równań różniczkowych cząstkowych. Jeżeli usuniemy dowolny zbiór S wierszy macierzy A oraz ten sam zbiór kolumn, to wyznacznikiem powstałej w ten sposób macierzy będzie liczba lasów zorientowanych, których korzeniami są wierzchołki $\{V_k \mid k \in S\}$ i których krawędzie należą do danego grafu. Twierdzenie macierzowe o drzewach dla drzew zorientowanych zostało podane bez dowodu przez J. J. Sylvestera w 1857 roku (zobacz ćwiczenie 28), następnie zapomniane na wiele lat, aż ponownie odkrył je W. T. Tutte [Proc. Cambridge Phil. Soc. 44 (1948), 463–482, §3]. Pierwszy opublikowany dowód dotyczy szczególnego przypadku grafów *nieskierowanych*, dla których macierz A jest symetryczna; został on podany przez C. W. Borchardta [Crelle 57 (1860), 111–121]. Kilku autorów przypisuje to twierdzenie Kirchhoffowi, ale Kirchhoffowi zawdzięczamy inny (choć podobny) wynik.

20. Z ćwiczenia 18 mamy $B = A_0^T A_0$. Natomiast z ćwiczenia 19: B jest macierzą A_0 dla grafu skierowanego G' , powstałego z G przez zastąpienie każdej (nieskierowanej) krawędzi dwiema skierowanymi krawędziami o przeciwnych zwrotach. Każde wolne poddrzewo G odpowiada jednoznacznie zorientowanemu poddrzewu G' o korzeniu w V_0 , ponieważ kierunki krawędzi są wyznaczone w wyniku wyboru korzenia.

21. Budujemy macierze A i A^* jak w ćwiczeniu 19. Dla przykładowych grafów G i G^* z rysunków 36 i 37 mamy:

$$A = \begin{pmatrix} 2 & -2 & 0 \\ -1 & 3 & -2 \\ -1 & -1 & 2 \end{pmatrix}, \quad A^* = \begin{pmatrix} [00] & [10] & [20] & [01] & [01] & [21] & [12] & [12] & [22] \\ [00] & 2 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ [10] & -1 & 3 & 0 & -1 & -1 & 0 & 0 & 0 \\ [20] & -1 & 0 & 3 & -1 & -1 & 0 & 0 & 0 \\ \hline [01] & 0 & -1 & 0 & 3 & 0 & 0 & -1 & -1 & 0 \\ [01] & 0 & -1 & 0 & 0 & 3 & 0 & -1 & -1 & 0 \\ [21] & 0 & -1 & 0 & 0 & 0 & 3 & -1 & -1 & 0 \\ \hline [12] & 0 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & -1 \\ [12] & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 3 & -1 \\ [22] & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 2 \end{pmatrix}.$$

Dodajemy element nieoznaczony λ do lewego górnego elementu A i A^* (w przykładzie to daje $2 + \lambda$ w miejscu 2). Jeśli $t(G)$ i $t(G^*)$ oznaczają liczby zorientowanych poddrzew

grafów G i G^* , to mamy $t(G) = \lambda^{-1}(n+1) \det A$, $t(G^*) = \lambda^{-1}m(n+1) \det A^*$. (Liczba zorientowanych drzew w grafie zrównoważonym jest taka sama dla każdego korzenia, co wynika z ćwiczenia 22).

Jeśli pogrupujemy wierzchołki V_{jk} względem równych k , to macierz A^* można podzielić wg schematu pokazanego powyżej. Niech $B_{kk'}$ będzie podmacierzą A^* złożoną z wierszy odpowiadających V_{jk} i kolumn odpowiadających $V_{j'k'}$, dla wszystkich j i j' , takich że V_{jk} i $V_{j'k'}$ należą do G^* . Dodając drugą, ..., m -tą kolumnę każdej podmacierzy do pierwszej kolumny, a następnie odejmując pierwszy wiersz każdej podmacierzy od drugiego, ..., m -tego wiersza, przekształcamy macierz A^* w ten sposób, że

$$B_{kk'} = \begin{pmatrix} a_{kk'} & * & \dots & * \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \quad \text{dla } k \neq k', \quad B_{kk} = \begin{pmatrix} a_{kk} + \lambda \delta_{k0} & * & \dots & * \\ -\lambda \delta_{k0} & m & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\lambda \delta_{k0} & 0 & \dots & m \end{pmatrix}.$$

Wynika stąd, że $\det A^*$ to $\lambda^{-1}m^{m(n-1)}$ razy wyznacznik macierzy

$$\begin{pmatrix} \lambda + a_{00} & * & * & \dots & * & a_{01} & \dots & a_{0n} \\ -\lambda & m & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & & & & & & \\ -\lambda & 0 & 0 & \dots & m & 0 & \dots & 0 \\ a_{10} & * & * & \dots & * & a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots & & & & \\ a_{n0} & * & * & \dots & * & a_{n1} & \dots & a_{nn} \end{pmatrix}.$$

W powyższym zapisie „*” oznacza wartość nieistotną dla dalszego wyprowadzenia; w istocie wszystkie gwiazdki są zerami poza dokładnie jedną -1 w każdej kolumnie. Dodajemy teraz ostatnie n wierszy do górnego wiersza i rozwijamy wyznacznik względem pierwszego wiersza, otrzymując $m^{n(m-1)+m-1} \det A - (m-1)m^{n(m-1)+m-2} \det A$.

To wyprowadzenie można uogólnić, co daje sposób wyznaczenia liczby zorientowanych poddrzew grafu G^* dla dowolnego skierowanego grafu G ; zobacz R. Dawson, I. J. Good, Ann. Math. Stat. **28** (1957), 946–956; D. E. Knuth, Journal of Combinatorial Theory **3** (1967), 309–314. Alternatywny, czysto kombinatoryczny dowód został podany przez J. B. Orlina, Journal of Combinatorial Theory **B25** (1978), 187–198.

22. Całkowita liczba cykli Eulera to $(\sigma_1 + \dots + \sigma_n)$ razy liczba cykli Eulera zaczynających się od krawędzi e_1 , gdzie $\text{init}(e_1) = V_1$. Na mocy lematu E każdy taki cykl wyznacza drzewo zorientowane o korzeniu w V_1 , a dla każdego spośród T zorientowanych poddrzew istnieje $\prod_{j=1}^n (\sigma_j - 1)!$ ścieżek spełniających warunki twierdzenia D, odpowiadających różnym porządkom występowania krawędzi $\{e \mid \text{init}(e) = V_j, e \neq e[V_j], e \neq e_1\}$ na ścieżce P . (W ćwiczeniu 14 jest prostszy przykład).

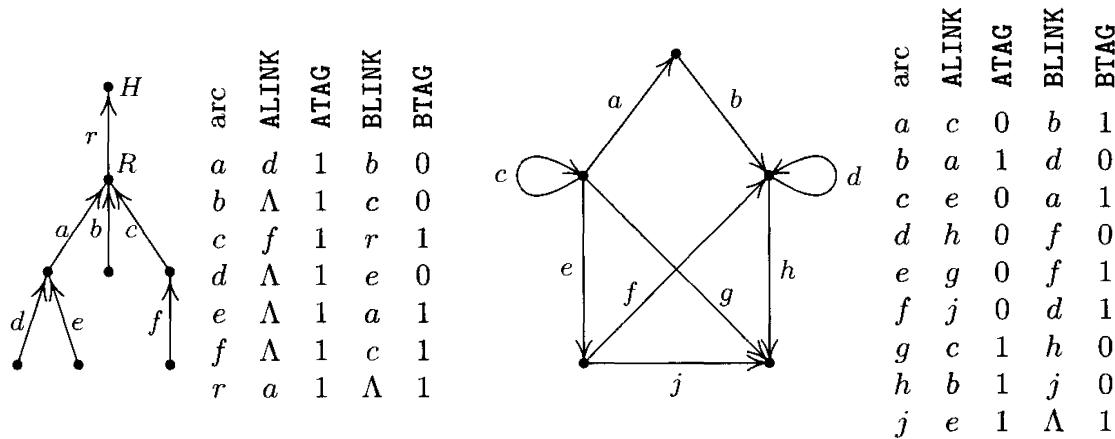
23. Konstruujemy graf skierowany G_k o m^{k-1} wierzchołkach, zgodnie ze wskazówką. Niech $[x_1, \dots, x_k]$ oznacza krawędzie opisane we wskazówce. Możemy określić wzajemnie jednoznaczna odpowiedniość między funkcjami o maksymalnym okresie a cyklami Eulera, biorąc $f(x_1, \dots, x_k) = x_{k+1}$, jeśli w cyklu bezpośrednio po krawędzi $[x_1, \dots, x_k]$ występuje krawędź $[x_2, \dots, x_{k+1}]$. (Przyjmujemy, że cykle będące swoimi cyklicznymi permutacjami są równe). Teraz $G_k = G_{k-1}^*$ w sensie ćwiczenia 21, zatem G_k ma $m^{m^{k-1}-m^{k-2}}$ razy tyle drzew zorientowanych, co G_{k-1} . Na mocy indukcji G_k ma $m^{m^{k-1}-1}$ poddrzew zorientowanych, a $m^{m^{k-1}-k}$ dla ustalonego korzenia. Stąd na mocy ćwiczenia 22 liczba funkcji o maksymalnym okresie, tj. liczba cykli Eulera w grafie G_k

zaczynających się od danej krawędzi, wynosi $m^{-k}(m!)^{m^{k-1}}$. [Szczególny przypadek tego twierdzenia dla $m = 2$ pochodzi z: C. Flye Sainte-Marie, *L'Intermédiaire des Mathématiciens* 1 (1894), 107–110].

24. Definiujemy nowy graf skierowany, w którym jest po E_j kopii krawędzi e_j dla $0 \leq j \leq m$. Ten graf jest zrównoważony, ponieważ (na mocy twierdzenia G) zawiera cykl Eulera (e_0, \dots) . Poszukiwaną ścieżkę otrzymujemy, usuwając e_0 z cyklu Eulera.

25. Ustalamy dowolny porządek na wszystkich krawędziach w rodzinach zbiorów $I_j = \{e \mid \text{init}(e) = V_j\}$ i $F_j = \{e \mid \text{fin}(e) = V_j\}$. Dla każdej krawędzi e z I_j ustalamy $\text{ATAG}(e) = 0$ i $\text{ALINK}(e) = e'$, jeśli e' poprzedza e w porządku na I_j . Ponadto ustalamy $\text{ATAG}(e) = 1$ i $\text{ALINK}(e) = e'$, jeśli e jest ostatnim elementem I_j i e' jest pierwszym elementem w F_j . Jeśli F_j jest pusty, to przyjmujemy $\text{ALINK}(e) = \Lambda$. Analogicznie ustalamy wartości BLINK i BTAG , zamieniając rolami „init” i „fin”.

Przykłady (korzystamy z porządku alfabetycznego w każdym ze zbiorów krawędzi):



Uwaga: Jeśli w reprezentacji drzewa zorientowanego dodamy krawędź od H do niego samego, to otrzymamy ciekawą sytuację: albo dostajemy strukturę w standarodowej konwencji 2.3.1–(8) z *zamienionymi LLINK, LTAG, RLINK, RTAG* w atrapie, albo (jeśli nowa krawędź jest ostatnią wg przyjętego porządku) strukturę w standardowej konwencji z tym wyjątkiem, że $\text{RTAG} = 0$ w wierzchołku związanym z korzeniem drzewa.

To ćwiczenie jest oparte na pomyśle przekazanym autorowi przez W. C. Lynch. Czy za pomocą takiej reprezentacji algorytm obchodzenia drzewa, taki jak algorytm 2.3.1S, można uogólnić na klasę grafów skierowanych, które nie są drzewami zorientowanymi?

27. Niech a_{ij} będzie sumą $p(e)$ względem wszystkich krawędzi e prowadzących od V_i do V_j . Mamy udowodnić, że $t_j = \sum_i a_{ij} t_i$ dla wszystkich j . Z uwagi na fakt, że $\sum_i a_{ji} = 1$, musimy udowodnić, że $\sum_i a_{ji} t_j = \sum_i a_{ij} t_i$. Ale to nie jest trudne, ponieważ obie strony równania reprezentują sumy wszystkich iloczynów $p(e_1) \dots p(e_n)$ brane po podgrafach $\{e_1, \dots, e_n\}$ grafu G , takich że $\text{init}(e_i) = V_i$, oraz takich, że istnieje dokładnie jeden cykl zorientowany zawarty w $\{e_1, \dots, e_n\}$ i zawierający V_j . Usuwając dowolną krawędź z cyklu, otrzymamy drzewo zorientowane. Lewą stronę równania uzyskujemy, grupując wyrazy względem krawędzi wychodzących z V_j , a prawą stronę – względem krawędzi wchodzących do V_j .

W istocie to ćwiczenie jest połączeniem ćwiczenia 19 i 26.

28. Każdy wyraz rozwinięcia jest postaci $a_{1p_1} \dots a_{mp_m} b_{1q_1} \dots b_{nq_n}$, gdzie $0 \leq p_i \leq n$ dla $1 \leq i \leq m$ i $0 \leq q_j \leq m$ dla $1 \leq j \leq n$, razy pewna stała całkowita. Wyraźmy ten iloczyn jako graf skierowany o wierzchołkach $\{0, u_1, \dots, u_m, v_1, \dots, v_n\}$ i o krawędziach z u_i do v_{p_i} oraz z v_j do u_{q_j} , gdzie $u_0 = v_0 = 0$.

Jeśli ten graf skierowany zawiera cykl, to współczynnik jest równy zeru. Dzieje się tak dlatego, że każdy cykl odpowiada współczynnikowi postaci

$$a_{i_0 j_0} b_{j_0 i_1} a_{i_1 j_1} \dots a_{i_{k-1} j_{k-1}} b_{j_{k-1} i_0} \quad (*)$$

gdzie indeksy $(i_0, i_1, \dots, i_{k-1})$ są parami różne oraz indeksy $(j_0, j_1, \dots, j_{k-1})$ są parami różne. Sumę wszystkich wyrazów zawierających czynnik $(*)$ w wyznaczniku uzyskujemy, podstawiając $a_{i_l j} \leftarrow [j = j_l]$ dla $0 \leq l \leq k-1$ i $b_{j_l i} \leftarrow [i = i_{(l+1) \bmod k}]$ dla $0 \leq l < k$, zmienne w pozostałych $m+n-2k$ wierszach pozostawiając bez zmian. Ten wyznacznik jest tożsamościowo równy zero, ponieważ suma wierszy i_0, i_1, \dots, i_{k-1} w górnej części równa się sumie wierszy j_0, j_1, \dots, j_{k-1} w dolnej części.

Jednak jeśli graf skierowany nie zawiera cykli, to współczynnik jest równy +1. Jest tak, ponieważ każdy czynnik $a_{i_p i} b_{j_q j}$ musi pochodzić z przekątnej macierzy: jeśli dowolny pochodzący spoza przekątnej element $a_{i_0 j_0}$ zostanie wybrany w wierszu i_0 w górnej części, to musimy wybrać pewien element $a_{i_1 j_1}$ spoza przekątnej z wiersza i_1 górnej części itd., co daje cykl.

Zatem współczynnik ma wartość +1 wtedy i tylko wtedy, gdy odpowiadający wyrazowi graf skierowany jest drzewem zorientowanym o korzeniu 0. Liczbę takich wyrazów (i liczbę takich zorientowanych drzew) uzyskujemy, biorąc a_{ij} i b_{ji} równe 1; na przykład

$$\begin{aligned} \det \begin{pmatrix} 4 & 0 & 1 & 1 & 1 \\ 0 & 4 & 1 & 1 & 1 \\ 1 & 1 & 3 & 0 & 0 \\ 1 & 1 & 0 & 3 & 0 \\ 1 & 1 & 0 & 0 & 3 \end{pmatrix} &= \det \begin{pmatrix} 4 & 0 & 1 & 1 & 1 \\ -4 & 4 & 0 & 0 & 0 \\ 1 & 1 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 & 0 \\ 0 & 0 & -3 & 0 & 3 \end{pmatrix} = \det \begin{pmatrix} 4 & 0 & 3 & 1 & 1 \\ 0 & 4 & 0 & 0 & 0 \\ 2 & 1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{pmatrix} \\ &= \det \begin{pmatrix} 4 & 3 \\ 2 & 3 \end{pmatrix} \cdot 4 \cdot 3 \cdot 3. \end{aligned}$$

W ogólności otrzymujemy $\det \begin{pmatrix} n+1 & n \\ m & m+1 \end{pmatrix} \cdot (n+1)^{m-1} \cdot (m+1)^{n-1}$.

Uwagi: J. J. Sylvester rozważał przypadek szczególny $m = n$ i $a_{10} = a_{20} = \dots = a_{m0} = 0$ w *Quarterly J. of Pure and Applied Math.* **1** (1857), 42–56, gdzie (poprawnie) przypuszczał, że całkowita liczba wyrazów wynosi $n^n(n+1)^{n-1}$. Podał również bez dowodu fakt, że $(n+1)^{n-1}$ niezerowych wyrazów występujących, gdy $a_{ij} = \delta_{ij}$, odpowiada wszystkim spójnym bezcyklowym grafom o wierzchołkach $\{0, 1, \dots, n\}$. Dla szczególnego przypadku zredukował macierz pod wyznacznikiem do postaci z twierdzenia macierzowego o drzewach z ćwiczenia 19, tj.:

$$\det \begin{pmatrix} b_{10} + b_{12} + b_{13} & -b_{12} & -b_{13} \\ -b_{21} & b_{20} + b_{21} + b_{23} & -b_{23} \\ -b_{31} & -b_{32} & b_{30} + b_{31} + b_{32} \end{pmatrix}.$$

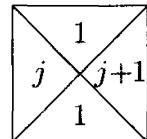
Cayley cytował ten wynik w *Crelle* **52** (1856), 279, przypisując go Sylvesterowi; jak na ironię twierdzenie o liczbie takich grafów często przypisuje się Cayleyowi.

Odwracając znak pierwszych m wierszy macierzy pod wyznacznikiem, a następnie odwracając znak pierwszych m kolumn, możemy sprowadzić to ćwiczenie do twierdzenia macierzowego o drzewach.

[Macierze tej postaci, co macierz w tym ćwiczeniu, mają istotne znaczenie przy iteracyjnym rozwiązywaniu równań różniczkowych cząstkowych; mówi się, że mają „własność A”. Zobacz na przykład: Louis A. Hageman, David M. Young, *Applied Iterative Methods* (Academic Press, 1981), rozdział 9].

2.3.4.3

1. Korzeń jest ciągiem pustym. Krawędź prowadzi od (x_1, \dots, x_n) do (x_1, \dots, x_{n-1}) .
2. Bierzemy jeden kafelek i obracamy go o 180° . Te dwa rodzaje kafelków wystarczą, by (bez obracania) pokryć całą płaszczyznę, powtarzając wzór 2×2 .



3. Rozważmy następujący zestaw kafelków dla wszystkich dodatnich liczb całkowitych j . Pierwszą ćwiartkę można pokryć na nieskończoność wiele sposobów, ale położenie kafelka w środku układu współrzędnych ogranicza odległość, na jaką możemy pokryć płaszczyznę w lewo.

4. Należy systematycznie przeglądać wszystkie możliwe pokrycia kwadratów $n \times n$ dla $n = 1, 2, \dots$, poszukując w nich pokryć toroidalnych. Jeżeli nie ma sposobu pokrycia płaszczyzny, to z lematu Kóniga istnieje n , dla którego nie istnieje pokrycie $n \times n$. Jeżeli *istnieje* sposób pokrycia płaszczyzny, to na mocy założenia istnieje n , dla którego pokrycie kwadratu $n \times n$ zawiera prostokąt dający rozwiązań toroidalne. Stąd w każdym przypadku wykonanie algorytmu się zakończy. (Podane założenie jest jednak fałszywe, co pokazujemy w kolejnym ćwiczeniu. W istocie nie istnieje algorytm, który umożliwiałby stwierdzenie, czy płaszczyznę można pokryć za pomocą danego zestawu kafelków).

5. Zaczynamy od zauważenia, że w każdym rozwiązyaniu klasy kafelków muszą się powtarzać w grupach 2×2 postaci $\begin{smallmatrix} \alpha & \beta \\ \gamma & \delta \end{smallmatrix}$. Następnie, krok 1: zajmujemy się tylko kafelkami α i pokazujemy, że wzór $\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$ musi się powtarzać w grupach 2×2 kafelków α . Krok $n > 1$: wyznaczamy schemat, który musi się pojawić w „krzyżu” o szerokości i wysokości $2^n - 1$. Środki krzyży układają się we wzór $\begin{smallmatrix} Na & Nb \\ Nc & Nd \end{smallmatrix}$ powtarzający się na całej płaszczyźnie.

Na przykład po kroku 3 będziemy znać zawartość powtarzających się obszarów 7×7 , porozdzielanych paskami o szerokości jednego kafelka (co osiem kafelków). Obszar 7×7 z kafelkiem typu Na w środku krzyża jest postaci

αa	βKQ	αb	βQP	αa	βBK	αb
γPJ	δNa	γRB	δQK	γLJ	δNb	γPB
αc	βDS	αd	βQTY	αc	βBS	αd
γPQ	δPJ	γPXB	δNa	γRQ	δRB	γRB
αa	βUK	αb	βDP	αa	βBK	αb
γTJ	δNc	γSB	δDS	γST	δNd	γTB
αc	βQS	αd	βDT	αc	βBS	αd

Środkowa kolumna i środkowy wiersz to „krzyż” ułożony w kroku 3; pozostałe cztery kwadraty 3×3 zostały ułożone w kroku 2. Kwadraty leżące na prawo i poniżej naszego kwadratu 7×7 są fragmentami krzyża o rozmiarach 15×15 , który zostanie ułożony w kroku 4.

Podobna konstrukcja prowadzi do zestawu 35 kafelków dających wyłącznie pokrycia nietoroidalne; zobacz R. M. Robinson, *Inventiones Math.* **12** (1971), 177–209. Robinson pokazuje także jeszcze inny zbiór sześciu figur dających wyłącznie nietoroidalne pokrycia płaszczyzny, nawet gdy dopuścimy obroty i odbicia lustrzane. W 1974 roku Roger Penrose, zarzucając siatkę kwadratową na rzecz siatki opartej na złotej proporcji,

odkrył zbiór zawierający jedynie dwa wielokąty, który daje wyłącznie acykliczne pokrycia płaszczyzny. Dzięki temu udało się pokazać zestaw jedynie 16 kafelków dający wyłącznie pokrycia nietoroidalne [zobacz B. Grünbaum i G. C. Shephard, *Tilings and Patterns* (Freeman, 1987), rozdziały 10–11; Martin Gardner, *Penrose Tiles to Trapdoor Ciphers* (Freeman, 1989), rozdziały 1–2].

6. Niech k i m będą ustalone. Rozważmy drzewo zorientowane, którego każdy wierzchołek dla pewnego n reprezentuje podział zbioru $\{1, \dots, n\}$ na k części, nie zawierających ciągu arytmetycznego o długości m . Wierzchołek reprezentujący podział $\{1, \dots, n+1\}$ jest dzieckiem wierzchołka reprezentującego podział $\{1, \dots, n\}$, jeśli te dwa podziały są zgodne na $\{1, \dots, n\}$. Jeśli istniałaby nieskończona ścieżka do korzenia, to istniałby podział zbioru *wszystkich* liczb całkowitych na k zbiorów nie zawierających ciągu arytmetycznego o długości m . Stąd na mocy lematu o drzewach nieskończonych i twierdzenia van der Waerdena to drzewo jest skończone. (Dla $k = 2$ i $m = 3$ drzewo można łatwo wyznaczyć ręcznie. Okazuje się, że najmniejsza wartość N wynosi 9. Zobacz *Studies in Pure Mathematics*, ed. L. Mirsky (Academic Press, 1971), 251–260, gdzie opisano zmagania van der Waerdena z dowodem jego twierdzenia).

7. Dodatnie liczby całkowite można podzielić na takie dwa zbiory S_0 i S_1 , że żaden z nich nie zawiera nieskończonego *obliczalnego* ciągu (zobacz ćwiczenie 3.5–32). Zatem w szczególności żaden z tych zbiorów nie może zawierać ciągu arytmetycznego. Twierdzenie K nie ma zastosowania, ponieważ nie da się utworzyć drzewa rozwiązań częściowych o skończonych stopniach wierzchołków.

8. Niech „ciąg-kontrprzykład” będzie nieskończonym ciągiem drzew naruszającym twierdzenie Kruskala. Założymy, że twierdzenie jest fałszywe. Wówczas niech T_1 będzie drzewem o najmniejszej liczbie wierzchołków, które może wystąpić na pierwszej pozycji ciągu-kontrprzykładu. Jeśli mamy ciąg T_1, \dots, T_j , to niech T_{j+1} będzie drzewem o najmniejszej możliwej liczbie wierzchołków, takim że T_1, \dots, T_j, T_{j+1} jest początkiem pewnego ciągu-kontrprzykładu. Powyższe reguły określają ciąg-kontrprzykład $\langle T_n \rangle$. Żadne z występujących w nim drzew T nie składa się wyłącznie z korzenia. Przyjrzymy się teraz uważnie temu ciągowi:

(a) Przypuśćmy, że istnieje podciąg T_{n_1}, T_{n_2}, \dots , dla którego $l(T_{n_1}), l(T_{n_2}), \dots$ jest ciągiem-kontrprzykładem. Tak jednak być nie może, bo $T_1, \dots, T_{n_1-1}, l(T_{n_1}), l(T_{n_2}), \dots$ byłby ciągiem-kontrprzykładem, co stoi w sprzeczności z definicją T_{n_1} .

(b) Z uwagi na (a), istnieje tylko skończenie wiele j , dla których $l(T_j)$ nie można włożyć w $l(T_k)$ dla dowolnego $k > j$. Stąd biorąc n_1 większe niż wszystkie takie j , znajdujemy podciąg, dla którego $l(T_{n_1}) \subseteq l(T_{n_2}) \subseteq l(T_{n_3}) \subseteq \dots$

(c) Teraz na mocy wyniku ćwiczenia 2.3.2–22, $r(T_{n_j})$ nie można włożyć w $r(T_{n_k})$ dla żadnego $k > j$, bo mielibyśmy $T_{n_j} \subseteq T_{n_k}$. Stąd $T_1, \dots, T_{n_1-1}, r(T_{n_1}), r(T_{n_2}), \dots$ jest ciągiem-kontrprzykładem. Ale to stoi w sprzeczności z definicją T_{n_1} .

Uwagi: Kruskal, w *Trans. Amer. Math. Soc.* **95** (1960), 210–225, udowodnił w istocie wynik silniejszy, korzystając ze słabszego pojęcia włożenia. Jego twierdzenie nie wynika wprost z lematu o drzewach nieskończonych, choć wynik jest podobny. W istocie sam König udowodnił przypadek szczególny twierdzenia Kruskala, pokazując, że nie istnieje nieskończony ciąg n -elementowych krotek nieujemnych liczb całkowitych nieporównywalnych po współrzędnych, gdzie porównywalność oznacza, że każda składowa krotki jest mniejsza lub równa odpowiadającej jej składowej drugiej krotki. [*Matematikai és Fizikai Lapok* **39** (1932), 27–29]. Nowsze wyniki można znaleźć w: *J. Combinatorial Theory A* **13** (1972), 297–305; zobacz też: N. Dershowitz, *Inf. Proc. Letters* **9** (1979), 212–215, gdzie pokazano zastosowania w dowodzeniu własności stopu algorytmów.

2.3.4.4

$$1. \ln A(z) = \ln z + \sum_{k \geq 1} a_k \ln \left(\frac{1}{1-z^k} \right) = \ln z + \sum_{k,t \geq 1} \frac{a_k z^{kt}}{t} = \ln z + \sum_{t \geq 1} \frac{A(z^t)}{t}.$$

2. Różniczkując i przyrównując współczynniki przy z^n otrzymujemy tożsamość

$$na_{n+1} = \sum_{k \geq 1} \sum_{d \mid k} da_d a_{n+1-k}.$$

Teraz zmieniamy kolejność sumowania.

4. (a) $A(z)$ jest zbieżna przynajmniej dla $|z| < \frac{1}{4}$, ponieważ a_n jest mniejsze niż liczba uporządkowanych drzew b_{n-1} . Ponieważ $A(1)$ jest nieskończony i wszystkie jego współczynniki są dodatnie, zatem istnieje liczba dodatnia $\alpha \leq 1$, taka że $A(z)$ jest zbieżne dla $|z| < \alpha$ i występuje osobliwość dla $z = \alpha$. Niech $\psi(z) = A(z)/z$; mamy $\psi(z) > e^{z\psi(z)}$, zatem $\psi(z) = m$ pociąga $z < \ln m/m$, więc $\psi(z)$ jest ograniczone i istnieje $\lim_{z \rightarrow \alpha^-} \psi(z)$. Stąd $\alpha < 1$, więc z twierdzenia granicznego Abela $a = \alpha \cdot \exp(a + \frac{1}{2}A(\alpha^2) + \frac{1}{3}A(\alpha^3) + \dots)$.

(b) $A(z^2), A(z^3), \dots$ są analityczne dla $|z| < \sqrt{\alpha}$, zaś $\frac{1}{2}A(z^2) + \frac{1}{3}A(z^3) + \dots$ zbiega jednostajnie w nieco mniejszej kuli.

(c) Jeśli $\partial F/\partial w = a - 1 \neq 0$, to z twierdzenia o funkcji uwikłanej wynika, że istnieje funkcja analityczna $f(z)$ w otoczeniu $(\alpha, a/\alpha)$, taka że $F(z, f(z)) = 0$. Ale stąd $f(z) = A(z)/z$, co jest sprzeczne z faktem, że $A(z)$ jest osobliwa w punkcie α .

(d) Oczywiste.

(e) $\partial F/\partial w = A(z) - 1$ i $|A(z)| < A(\alpha) = 1$, ponieważ wszystkie współczynniki $A(z)$ są dodatnie. Stąd, tak jak w (c), $A(z)$ jest regularny we wszystkich takich punktach.

(f) W pobliżu $(\alpha, 1/\alpha)$ mamy tożsamość $0 = \beta(z - \alpha) + (\alpha/2)(w - 1/\alpha)^2 +$ wyrazy wyższych rzędów, gdzie $w = A(z)/z$; zatem w jest funkcją analityczną zmiennej $\sqrt{z - \alpha}$, na mocy twierdzenia o funkcji uwikłanej. Istnieje zatem obszar $|z| < \alpha_1$ minus wycinek $[\alpha, \alpha_1]$, w którym $A(z)$ jest podanej postaci. (Wybraliśmy znak minus, ponieważ plus sprawiłby, że współczynniki byłyby ujemne).

(g) Dowolna funkcja podanej postaci ma współczynniki asymptotycznie równe $\frac{\sqrt{2\beta}}{\alpha^n} \binom{1/2}{n}$. Zauważmy, że

$$\binom{3/2}{n} = O\left(\frac{1}{n} \binom{1/2}{n}\right).$$

Więcej informacji na temat asymptotycznych wartości liczby drzew wolnych znajduje się w: R. Otter, *Ann. Math.* (2) **49** (1948), 583–599.

$$5. c_n = \sum_{j_1+2j_2+\dots=n} \binom{c_1+j_1-1}{j_1} \dots \binom{c_n+j_n-1}{j_n} - c_n, \quad n > 1.$$

Stąd

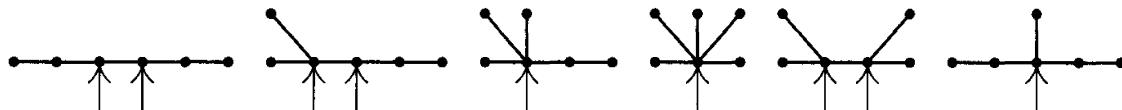
$$2C(z) + 1 - z = (1 - z)^{-c_1} (1 - z^2)^{-c_2} (1 - z^3)^{-c_3} \dots = \exp(C(z) + \frac{1}{2}C(z^2) + \dots).$$

$C(z) = z + z^2 + 2z^3 + 5z^4 + 12z^5 + 33z^6 + 90z^7 + 261z^8 + 766z^9 + \dots$. Dla $n > 1$ liczba sieci szeregowo-równoległych o n krawędziach równa się $2c_n$ [zobacz P. A. MacMahon, *Proc. London Math. Soc.* **22** (1891), 330–339].

6. $zG(z)^2 = 2G(z) - 2 - zG(z^2)$; $G(z) = 1 + z + z^2 + 2z^3 + 3z^4 + 6z^5 + 11z^6 + 23z^7 + 46z^8 + 98z^9 + \dots$. Funkcja $F(z) = 1 - zG(z)$ spełnia prostą zależność $F(z^2) = 2z + F(z)^2$. [J. H. M. Wedderburn, *Annals of Math.* **24** (1922), 121–140].

$$7. g_n = ca^n n^{-3/2} (1 + O(1/n)), \text{ gdzie } c \approx 0.7916031835775, a \approx 2.483253536173.$$

8.



9. Jeśli są dwa centroidy, to rozważając ścieżkę od jednego do drugiego, dochodzimy do wniosku, że nie może być na niej wierzchołków pośrednich. Zatem dowolne dwa centroidy muszą ze sobą sąsiadować. Nie może być trzech wzajemnie sąsiadujących ze sobą wierzchołków, więc centroidy są co najwyżej dwa.

10. Jeśli X i Y są sąsiadami, to niech $s(X, Y)$ będzie liczbą wierzchołków w poddrzewie Y drzewa X . Wówczas $s(X, Y) + s(Y, X) = n$. Z rozumowania zaprezentowanego w tekście wnioskujemy, że jeśli Y jest centroidem, to $\text{weight}(X) = s(X, Y)$. Stąd jeśli zarówno X , jak i Y są centroidami, to $\text{weight}(X) = \text{weight}(Y) = n/2$.

Przy wprowadzonych oznaczeniach z rozumowania podanego w tekście wnioskujemy, że jeśli $s(X, Y) \geq s(Y, X)$, to istnieje centroid w poddrzewie Y drzewa X . Zatem jeśli dwa drzewa wolne o m wierzchołkach połączymy krawędzią między X i Y , to otrzymamy drzewo wolne, w którym $s(X, Y) = m = s(Y, X)$. Muszą więc być dwa centroidy (X i Y).

[Obliczenie $s(X, Y)$ dla wszystkich sąsiadujących wierzchołków X i Y w czasie $O(n)$ jest dobrym ćwiczeniem programistycznym; na podstawie tej informacji możemy szybko znaleźć centroidy. Pierwszy wydajny algorytm wyznaczania położenia centroidu został podany przez A. J. Goldmana, *Transportation Sci.* 5 (1971), 212–221].

11. $zT(z)^t = T(z) - 1$; stąd $z + T(z)^{-t} = T(z)^{1-t}$. Stosując wzór 1.2.9–(21), $T(z) = \sum_n A_n(1, -t) z^n$, zatem liczba drzew t -arnych równa się

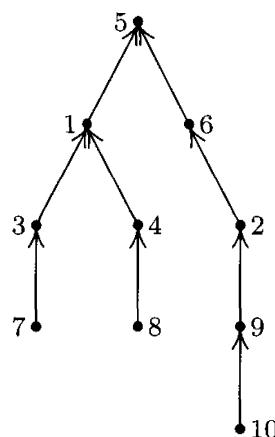
$$\binom{1+tn}{n} \frac{1}{1+tn} = \binom{tn}{n} \frac{1}{(t-1)n+1}.$$

12. Rozważmy graf skierowany, w którym dla $i \neq j$ istnieje jedna krawędź od V_i do V_j . Macierz A_0 z ćwiczenia 2.3.4.2–19 jest macierzą kombinatoryczną $(n-1) \times (n-1)$ mającą $n-1$ na przekątnej i -1 poza przekątną. Stąd jej wyznacznik ma postać

$$(n + (n-1)(-1))n^{n-2} = n^{n-2},$$

co jest liczbą drzew zorientowanych o ustalonym korzeniu. (Moglibyśmy skorzystać także z ćwiczenia 2.3.4.2–20).

13.



14. Prawda, ponieważ korzeń nigdy nie stanie się liściem, dopóki nie usuniemy wszystkich konarów.

15. W reprezentacji kanonicznej $V_1, V_2, \dots, V_{n-1}, f(V_{n-1})$ jest porządkiem topologicznym drzewa zorientowanego potraktowanego jak graf skierowany, ale w ogólności algorytm 2.2.3T wyznacza inne uporządkowania. Algorytm 2.2.3T można zmienić w ten sposób, by wyznaczał wartości V_1, V_2, \dots, V_{n-1} , jeśli operacja „wstaw do kolejki” w kroku T6 zostanie zastąpiona przez operację, która modyfikuje dowiązania w taki sposób, że elementy listy są uporządkowane rosnąco w kierunku od początku do końca kolejki. Taka kolejka staje się kolejką priorytetową.

(Nie jest jednak potrzebna ogólna kolejka priorytetowa do wyznaczenia reprezentacji kanonicznej. Musimy jedynie przeglądać wierzchołki od 1 do n w poszukiwaniu liści, usuwając ścieżki od nowo powstałych liści o numerach mniejszych niż aktualnie przeglądany liść; zobacz następujące ćwiczenie).

16. D1. Przyjmij $C[1] \leftarrow \dots \leftarrow C[n] \leftarrow 0$, następnie $C[f(V_j)] \leftarrow C[f(V_j)] + 1$ dla $1 \leq j \leq n$. (Wierzchołek k jest liściem wtedy i tylko wtedy, gdy $C[k] = 0$).
Przyjmij $k \leftarrow 0$ i $j \leftarrow 1$.

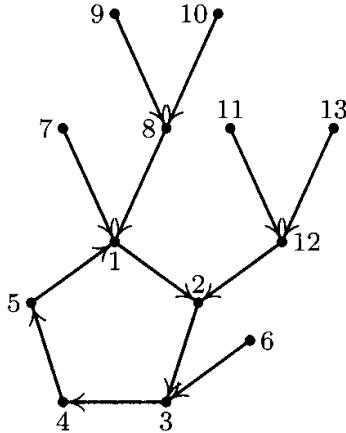
D2. Jeden lub więcej razy zwiększ k , aż $C[k] = 0$, następnie przyjmij $l \leftarrow k$.

D3. Przyjmij $PARENT[l] \leftarrow f(V_j)$, $l \leftarrow f(V_j)$, $C[l] \leftarrow C[l] - 1$, $j \leftarrow j + 1$.

D4. Jeśli $j = n$, to przyjmij $PARENT[l] \leftarrow 0$ i zakończ wykonanie algorytmu.

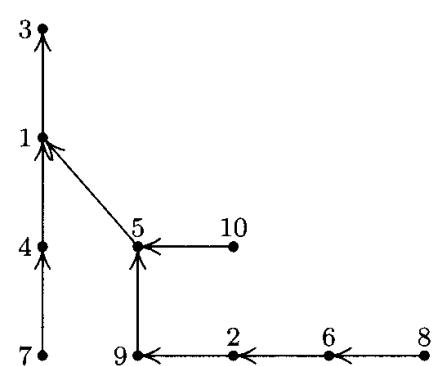
D5. Jeśli $C[l] = 0$ i $l < k$, to przejdź do D3; w przeciwnym razie wróć do D2. ■

17. Musi istnieć dokładnie jeden cykl x_1, x_2, \dots, x_k , gdzie $f(x_j) = x_{j+1}$ i $f(x_k) = x_1$. Będziemy zliczać wszystkie takie funkcje f mające cykl o długości k , że iteracje na dowolnym x prowadzą do tego cyklu. Definiujemy reprezentację kanoniczną $f(V_1), f(V_2), \dots, f(V_{m-k})$ tak jak w tekście; teraz $f(V_{m-k})$ jest w cyklu, zatem możemy kontynuować tworzenie „reprezentacji kanonicznej”, wypisując resztę cyklu $f(f(V_{m-k})), f(f(f(V_{m-k})))$ itd. Na przykład dla funkcji o grafie pokazanym obok ($m = 13$) otrzymujemy reprezentację 3, 1, 8, 8, 1, 12, 12, 2, 3, 4, 5, 1. Dostajemy ciąg $m-1$ liczb, w którym ostatnich k liczb jest różnych. Odwrotnie, dla dowolnego takiego ciągu możemy odwrócić konstrukcję (przy założeniu, że znamy k); stąd jest dokładnie $m^k m^{m-k-1}$ takich funkcji mających cykl długości k . (Podobne wyniki pojawiają się w ćwiczeniu 3.1-14. Wzór $m^{m-1}Q(m)$ jako pierwszy otrzymał L. Katz, *Annals of Math. Statistics* **26** (1955), 512–517).



18. Rekonstrukcję drzewa z ciągu s_1, s_2, \dots, s_{n-1} zaczynamy, biorąc s_1 za korzeń i kolejno dodając krawędzie prowadzące do s_1, s_2, \dots . Jeśli wierzchołek s_k wystąpił wcześniej, to wierzchołek początkowy krawędzi prowadzącej do s_{k-1} pozostawiamy nienazwany. W przeciwnym razie nadajemy mu nazwę s_k . Po narysowaniu wszystkich $n-1$ krawędzi nadajemy nazwy wszystkim jeszcze nie nazwanym wierzchołkom, posługując się niewykorzystanymi numerami, przydzieliając coraz większe numery później utworzonym wierzchołkom.

Na przykład z 3, 1, 4, 1, 5, 9, 2, 6, 5 możemy odtworzyć drzewo pokazane po prawej stronie. Nie ma prostego związku między tą metodą a metodą przedstawioną w tekście. Można wymyślić kilka



innych reprezentacji; zobacz: E. H. Neville, *Proc. Cambridge Phil. Soc.* **49** (1953), 381–385.

19. W reprezentacji kanonicznej będziemy mieć dokładnie $n - k$ różnych wartości, zatem zliczamy $(n-1)$ -elementowe ciągi liczbowe o tej własności. Odpowiedź: $\frac{(n-1)!}{(0!)^{k_0}(1!)^{k_1} \dots (n!)^{k_n}} \times \frac{n!}{k_0! k_1! \dots k_n!}$.

20. Weźmy reprezentację kanoniczną takich drzew. Pytamy, ile jest wyrazów rozwoju $(x_1 + \dots + x_n)^{n-1}$ mających k_0 wykładników równych zero, k_1 wykładników równych jeden itd. To jest po prostu współczynnik takiego wyrazu razy liczba takich wyrazów, konkretnie

$$\frac{(n-1)!}{(0!)^{k_0}(1!)^{k_1} \dots (n!)^{k_n}} \times \frac{n!}{k_0! k_1! \dots k_n!}.$$

21. Nie istnieją takie drzewa o $2m$ wierzchołkach. Jeśli $n = 2m + 1$, to odpowiedź uzyskujemy, podstawiając $k_0 = m+1$, $k_2 = m$ w ćwiczeniu 20, co daje $\binom{2m+1}{m} (2m)! / 2^m$.

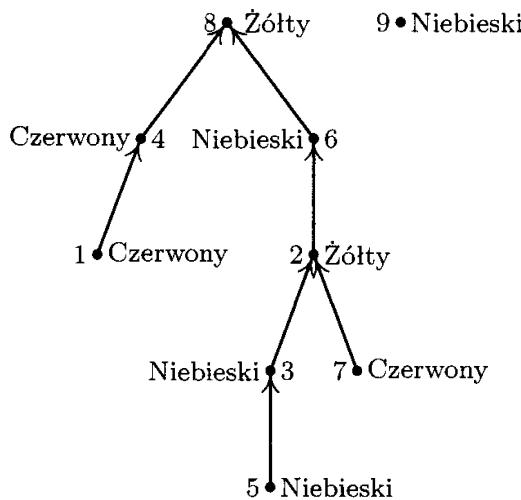
22. Dokładnie n^{n-2} , bo jeśli X jest konkretnym wierzchołkiem, to istnieje wzajemnie jednoznaczna odpowiedniość między drzewami wolnymi a drzewami zorientowanymi o korzeniu X .

23. Niepoetykietowane drzewo uporządkowane można poetykietować na $n!$ sposobów i każde takie etykietowanie daje inne drzewo. Łączna liczba wynosi zatem $n! b_{n-1} = (2n-2)!/(n-1)!$.

24. Drzewo jest tyle samo, niezależnie od wyboru korzenia, zatem odpowiedź w ogólności brzmi: n razy mniej niż w ćwiczeniu 23. W tym konkretnym przypadku 30.

25. Dla $0 \leq q < n$, $r(n, q) = (n-q)n^{q-1}$. (Przypadek szczególny $s = 1$ w równaniu (24)).

26. ($k = 7$)



27. Dla takiej funkcji g z $\{1, 2, \dots, r\}$ w $\{1, 2, \dots, q\}$, że dodanie krawędzi od V_k do $U_{g(k)}$ powoduje powstawanie cyklu zorientowanego, konstruujemy ciąg a_1, \dots, a_r . Wierzchołek V_k nazywamy „wolnym”, jeżeli nie istnieje ścieżka zorientowana od V_j do V_k dla każdego $j \neq k$. Ponieważ nie ma cykli zorientowanych, musi więc być przynajmniej jeden wierzchołek wolny. Niech b_1 będzie najmniejszą liczbą całkowitą, taką że V_{b_1} jest wierzchołkiem wolnym. Założymy, że wybrano b_1, \dots, b_t . Niech b_{t+1} będzie najmniejszą liczbą całkowitą różną od b_1, \dots, b_t , dla której wierzchołek $V_{b_{t+1}}$ jest wolny w grafie otrzymanym przez usunięcie krawędzi prowadzących od V_{b_k} do $U_{g(b_k)}$ dla $1 \leq k \leq t$. Ta reguła wyznacza permutację $b_1 b_2 \dots b_r$ liczb $\{1, 2, \dots, r\}$.

Niech $a_k = g(b_k)$ dla $1 \leq k \leq r$. W ten sposób otrzymamy taki ciąg, że $1 \leq a_k \leq q$ dla $1 \leq k < r$ oraz $1 \leq a_r \leq p$.

W drugą stronę, jeśli mamy ciąg a_1, \dots, a_r , to wierzchołek V_k nazywamy „wolnym”, jeśli nie istnieje takie j , że $a_j > p$ i $f(a_j) = k$. Z uwagi na fakt $a_r \leq p$ istnieje co najwyżej $r - 1$ wierzchołków nie będących wierzchołkami wolnymi. Niech b_1 będzie najmniejszą liczbą całkowitą, taką że wierzchołek V_{b_1} jest wolny. Założymy, że wybrano b_1, \dots, b_t . Niech b_{t+1} będzie najmniejszą liczbą różną od b_1, \dots, b_t , dla której $V_{b_{t+1}}$ jest wolny względem ciągu a_{t+1}, \dots, a_r . Ta reguła umożliwia wyznaczenie permutacji $b_1 b_2 \dots b_r$ liczb $\{1, 2, \dots, r\}$. Niech $g(b_k) = a_k$ dla $1 \leq k \leq r$. W ten sposób otrzymamy taką funkcję, że dodanie krawędzi prowadzącej od V_k do $U_{g(k)}$ spowoduje powstanie cyklu zorientowanego.

28. Niech f będzie dowolną spośród n^{m-1} funkcji z $\{2, \dots, m\}$ w $\{1, 2, \dots, n\}$. Rozważmy graf skierowany o wierzchołkach $U_1, \dots, U_m, V_1, \dots, V_n$ i krawędziach prowadzących od U_k do $V_{f(k)}$ dla $1 < k \leq m$. Korzystamy z ćwiczenia 27, biorąc $p = 1$, $q = m$, $r = n$, by pokazać, że istnieje m^{n-1} sposobów dodania nowych krawędzi od wierzchołków V do wierzchołków U , dających drzewo zorientowane o korzeniu U_1 . Ponieważ istnieje wzajemnie jednoznaczna odpowiedniość między interesującym nas zbiorem drzew wolnych a zbiorem drzew zorientowanych o korzeniu U_1 , odpowiedzią jest $n^{m-1}m^{n-1}$. [Tę konstrukcję można dalece uogólnić; zobacz D. E. Knuth, *Canadian J. Math.* **20** (1968), 1077–1086].

29. Jeśli $y = x^t$, to $(tz)y = \ln y$ i widzimy, że wystarczy wykazać tożsamość dla $t = 1$. Jeśli $zx = \ln x$, to z ćwiczenia 25 wiemy, że $x^m = \sum_k E_k(m, 1)z^k$ dla nieujemnych liczb całkowitych m . Stąd

$$\begin{aligned} x^r = e^{zxr} &= \sum_k \frac{(zxr)^k}{k!} = \sum_{j,k} \frac{r^k z^{k+j} E_j(k, 1)}{k!} = \sum_k \frac{z^k}{k!} \sum_j \binom{k}{j} j! E_j(k-j, 1) r^{k-j} \\ &= \sum_k \frac{z^k}{k!} \binom{k-1}{j} k^j r^{k-j} = \sum_k z^k E_k(r, 1). \end{aligned}$$

[W ćwiczeniu 4.7–22 wprowadzamy wniosek znacznie bardziej ogólny].

30. Każdy graf o podanych własnościach wyznacza zbiór $C_x \subseteq \{1, \dots, n\}$, taki że j należy do C_x wtedy i tylko wtedy, gdy istnieje ścieżka od t_j do r_i dla pewnego $i \leq x$. Dla danego zbioru C_x każdy graf opisanej postaci składa się z dwóch niezależnych części: jedna część to $x(x + \epsilon_1 z_1 + \dots + \epsilon_n z_n)^{\epsilon_1 + \dots + \epsilon_n - 1}$ grafów rozpiętych na wierzchołkach r_i, s_{jk}, t_j dla $i \leq x$ i $j \in C_x$, gdzie $\epsilon_j = |j \in C_x|$; druga część to $y(y + (1 - \epsilon_1)z_1 + \dots + (1 - \epsilon_n)z_n)^{(1 - \epsilon_1) + \dots + (1 - \epsilon_n) - 1}$ grafów rozpiętych na pozostałych wierzchołkach.

31. $G(z) = z + G(z)^2 + G(z)^3 + G(z)^4 + \dots = z + G(z)^2/(1 - G(z))$. Stąd $G(z) = \frac{1}{4}(1+z-\sqrt{1-6z+z^2}) = z+z^2+3z^3+11z^4+45z^5+\dots$ [Uwagi: Równoważny problem został postawiony i rozwiązany przez E. Schrödera, *Zeitschrift für Mathematik und Physik* **15** (1870), 361–376. Wyznaczył on liczbę sposobów poprowadzenia przekątnych nieprzecinających się poza wierzchołkami w wypukłym $(n+1)$ -kącie. Dla $n > 1$ te liczby są dwa razy mniejsze od wartości otrzymanych w ćwiczeniu 2.2.1–11, ponieważ gramatyka Pratta dopuszcza, by korzeń drzewa wprowadzenia był stopnia jeden. Wartość asymptotyczną obliczyliśmy w ćwiczeniu 2.2.1–12. Co ciekawe, wydaje się, że wartość $[z^{10}] G(z) = 103049$ obliczył już Hipparch w II wieku pne., jako „liczbę złożonych zdań twierdzących, które można zbudować, mając tylko dziesięć zdań prostych”; zobacz R. P. Stanley, *AMM* **104** (1997), 344–350].

32. Zero, jeśli $n_0 \neq 1 + n_2 + 2n_3 + 3n_4 + \dots$ (zobacz ćwiczenie 2.3–21), w przeciwnym razie

$$(n_0 + n_1 + \dots + n_m - 1)! / n_0! n_1! \dots n_m!.$$

By udowodnić ten wzór, przypomnijmy sobie, że nieetykietowane drzewo o $n = n_0 + n_1 + \dots + n_m$ węzłach jest wyznaczone przez ciąg stopni węzłów w porządku postorder $d_1 d_2 \dots d_n$ (zobacz punkt 2.3.3). Co więcej, taki ciąg stopni odpowiada drzewu wtedy i tylko wtedy, gdy $\sum_{j=1}^k (1 - d_j) > 0$ dla $0 < k \leq n$. (Tę ważną własność polskiej notacji przyrostkowej łatwo udowodnić przez indukcję; zobacz algorytm 2.3.3F, gdzie f jest funkcją konstruującą drzewo, jak TREE w punkcie 2.3.2). W szczególności d_1 musi być równe 0. Odpowiedź na postawione pytanie sprowadza się zatem do wyznaczenia liczby takich ciągów $d_2 \dots d_n$, w których j występuje n_j razy dla $j > 0$, a to jest współczynnik wielomianowy

$$\binom{n-1}{n_0-1, n_1, \dots, n_m},$$

minus liczba takich ciągów $d_2 \dots d_n$, że $\sum_{j=2}^k (1 - d_j) < 0$ dla pewnego $k \geq 2$.

Te odejmowane ciągi możemy zliczyć następująco: niech t będzie najmniejszą liczbą, taką że $\sum_{j=2}^t (1 - d_j) < 0$; wówczas $\sum_{j=2}^t (1 - d_j) = -s$, gdzie $1 \leq s < d_t$, i możemy utworzyć podciąg $d'_2 \dots d'_n = d_{t-1} \dots d_{t-1} 0 d_{t+1} \dots d_n$, w którym j występuje n_j razy dla $j \neq d_t$ lub $n_j - 1$ razy dla $j = d_t$. Suma $\sum_{j=2}^k (1 - d'_j)$ jest równa d_t , gdy $k = n$, i równa $d_t - s$, gdy $k = t$; dla $k < t$ jej wartość wynosi

$$\sum_{2 \leq j < t} (1 - d_j) - \sum_{2 \leq j \leq t-k} (1 - d_j) \leq \sum_{2 \leq j < t} (1 - d_j) = d_t - s - 1.$$

Wynika stąd, że mając s i dowolny ciąg $d'_2 \dots d'_n$, konstrukcję możemy odwrócić. Stąd liczba ciągów $d_2 \dots d_n$ o zadanych wartościach d_t i s jest współczynnikiem wielomianowym

$$\binom{n-1}{n_0, \dots, n_{d_t}-1, \dots, n_m}.$$

Liczبę ciągów $d_2 \dots d_n$ odpowiadających drzewom otrzymujemy zatem, sumując względem wszystkich możliwych wartości d_t i s :

$$\sum_{j=0}^m (1-j) \binom{n-1}{n_0, \dots, n_j-1, \dots, n_m} = \frac{(n-1)!}{n_0! n_1! \dots n_m!} \sum_{j=0}^m (1-j) n_j$$

Ostatnia suma ma wartość 1.

Jeszcze prostszy dowód podał G. N. Raney (*Transactions of the American Mathematical Society* 94 (1960), 441–451). Jeśli $d_1 d_2 \dots d_n$ jest dowolnym ciągiem, w którym j występuje n_j razy, to istnieje dokładnie jedna cykliczna permutacja $d_k \dots d_n d_1 \dots d_{k-1}$ odpowiadająca drzewu, konkretnie taka permutacja, że k jest największą liczbą, dla której wartość sumy $\sum_{j=1}^k (1 - d_j)$ jest minimalna. [Takie rozumowanie dla drzew binarnych zostało po raz pierwszy przeprowadzone przez C. S. Peirce'a w nieopublikowanym manuskrypcie; zobacz też tegoż autora *New Elements of Mathematics* 4 (The Hague: Mouton, 1976), 303–304. Odkrywcą przypadku dla drzew t -arnych są Dvoretzky i Motzkin, *Duke Math. J.* 14 (1947), 305–313].

Jeszcze jeden dowód, pochodzący od G. Bergmana, polega na indukcyjnym zaśnięciu $d_k d_{k+1}$ przez $(d_k + d_{k+1} - 1)$, jeśli $d_k > 0$ [*Algebra Universalis* 8 (1978), 129–130].

Zaprezentowane metody można uogólnić, by pokazać, że liczba (uporządkowanych, nieetykietowanych) lasów o f drzewach i n_j węzłach stopnia j równa się $(n-1)!f/n_0!n_1!\dots n_m!$, przy założeniu, że spełniony jest warunek $n_0 = f + n_2 + 2n_3 + \dots$.

33. Rozważmy liczbę drzew o n_1 węzłach o etykiecie 1, n_2 węzłach o etykiecie 2, ..., takich że każdy węzeł o etykiecie j jest stopnia e_j . Oznaczmy tę liczbę przez $c(n_1, n_2, \dots)$, przyjmując, że stopnie e_1, e_2, \dots są już ustalone. Funkcja tworząca $G(z_1, z_2, \dots) = \sum c(n_1, n_2, \dots)z_1^{n_1}z_2^{n_2}\dots$ spełnia tożsamość $G = z_1G^{e_1} + \dots + z_rG^{e_r}$, ponieważ $z_jG^{e_j}$ jest liczbą drzew, których korzeń ma etykietę j . Na mocy wyniku poprzedniego ćwiczenia

$$c(n_1, n_2, \dots) = \begin{cases} \frac{(n_1 + n_2 + \dots - 1)!}{n_1! n_2! \dots}, & \text{jeśli } (1 - e_1)n_1 + (1 - e_2)n_2 + \dots = 1; \\ 0, & \text{w przeciwnym razie.} \end{cases}$$

Bardziej ogólnie: z faktu, że G^f zlicza uporządkowane lasy o takich etykietach wynika, że dla całkowitych liczb $f > 0$

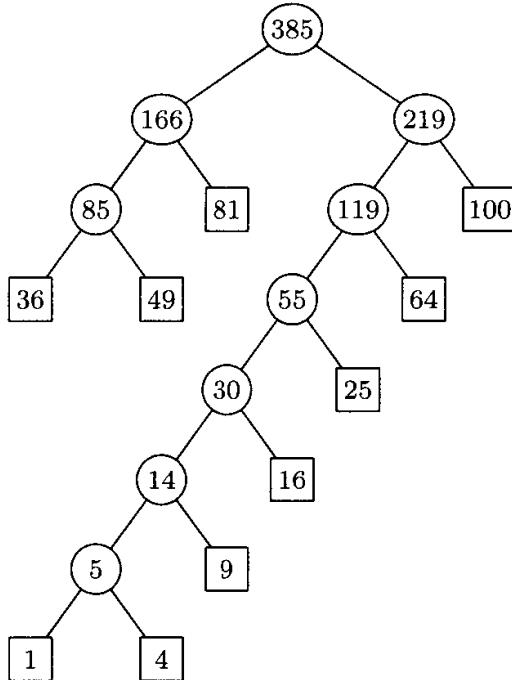
$$w^f = \sum_{f=(1-e_1)n_1+(1-e_2)n_2+\dots} \frac{(n_1 + n_2 + \dots - 1)! f}{n_1! n_2! \dots} z_1^{n_1} z_2^{n_2} \dots$$

Te wzory mają sens również dla $r = \infty$ i w istocie są równoważne wzorowi inwersyjnemu Lagrange'a.

2.3.4.5

1. Tak, $\binom{8}{5}$. Węzły o numerach 8, 9, 10, 11, 12 można uczynić dowolnymi dziećmi węzłów 4, 5, 6 i 7.

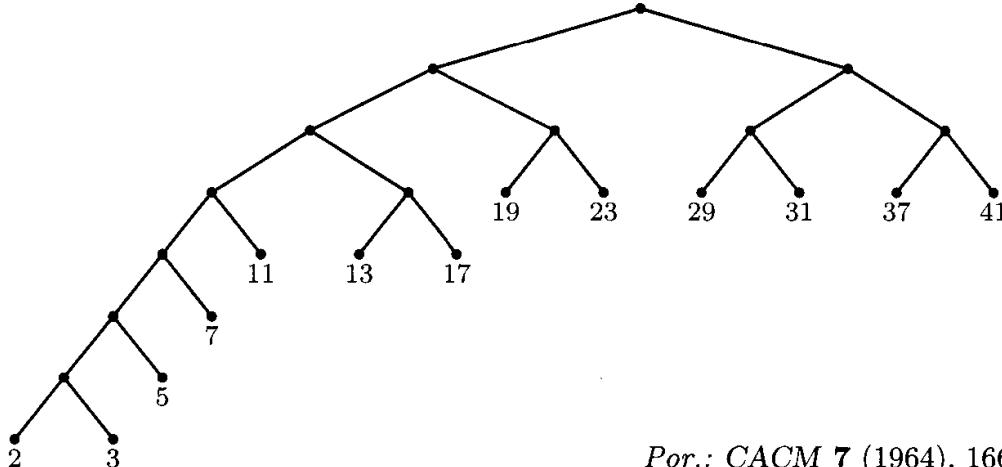
2.



3. Na mocy indukcji względem m warunek jest konieczny. W drugą stronę, jeśli $\sum_{j=1}^m 2^{-l_j} = 1$, to chcemy skonstruować rozszerzone drzewo binarne o długościach ścieżek l_1, \dots, l_m . Gdy $m = 1$, wtedy $l_1 = 0$ i konstrukcja jest trywialna. W przeciwnym razie możemy założyć, że długości l są uporządkowane w ten sposób, że $l_1 = l_2 = \dots = l_q > l_{q+1} \geq l_{q+2} \geq \dots \geq l_m > 0$ dla pewnego q , $1 \leq q \leq m$. Mamy

$2^{l_1-1} = \sum_{j=1}^m 2^{l_1-l_j-1} = \frac{1}{2}q +$ liczba całkowita, ponieważ q jest parzyste. Na mocy indukcji względem m istnieje drzewo o długościach ścieżek $l_1-1, l_3, l_4, \dots, l_m$. Bierzymy takie drzewo i zamieniamy jeden z węzłów zewnętrznych na poziomie l_1-1 na węzeł wewnętrzny, którego dzieci znajdują się na poziomie $l_1 = l_2$.

4. Zaczynamy od znalezienia drzewa metodą Huffmana. Jeśli $w_j < w_{j+1}$, to $l_j \geq l_{j+1}$, ponieważ drzewo jest optymalne. Stosując konstrukcję z odpowiedzi do ćwiczenia 3, otrzymamy inne drzewo o tych samych długościach ścieżek i o wagach w odpowiedniej kolejności. Na przykład drzewo (11) przyjmuje postać



Por.: CACM 7 (1964), 166–169.

5. (a) $b_{np} = \sum_{\substack{k+l=n-1 \\ r+s+n-1=p}} b_{kr} b_{ls}$. Stąd $zB(w, wz)^2 = B(w, z) - 1$.

(b) Wyznaczamy pochodną cząstkową względem w :

$$2zB(w, wz)(B_w(w, wz) + zB_z(w, wz)) = B_w(w, z).$$

Stąd, jeśli $H(z) = B_w(1, z) = \sum_n h_n z^n$, to $H(z) = 2zB(z)(H(z) + zB'(z))$; ze znanego wzoru na $B(z)$ otrzymujemy

$$H(z) = \frac{1}{1-4z} - \frac{1}{z} \left(\frac{1-z}{\sqrt{1-4z}} - 1 \right), \quad \text{więc} \quad h_n = 4^n - \frac{3n+1}{n+1} \binom{2n}{n}.$$

Wartość średnia wynosi h_n/b_n . (c) Asymptotycznie sprowadza się to do $n\sqrt{\pi n} - 3n + O(\sqrt{n})$.

Rozwiążanie podobnych problemów można znaleźć w: John Riordan, IBM J. Res. and Devel. 4 (1960), 473–478; A. Rényi, G. Szekeres, J. Australian Math. Soc. 7 (1967), 497–507; John Riordan, N. J. A. Sloane, J. Australian Math. Soc. 10 (1969), 278–282; ćwiczenie 2.3.1–11.

6. $n + s - 1 = tn$.

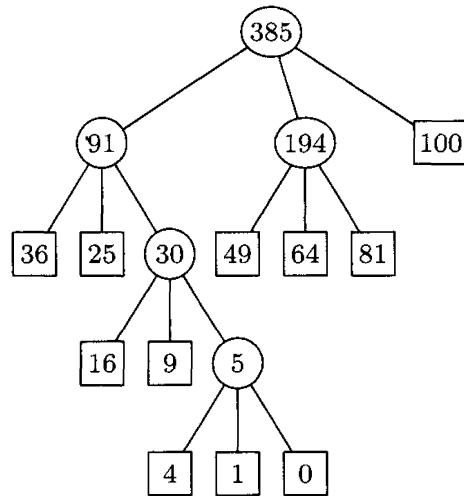
7. $E = (t-1)I + tn$.

8. Sumując przez części, otrzymamy $\sum_{k=1}^n \lfloor \log_t((t-1)k) \rfloor = nq - \sum k$, gdzie sumowanie po prawej stronie przebiega po wartościach k , takich że $0 \leq k \leq n$ i $(t-1)k + 1 = t^j$ dla pewnego j . Tę sumę można zapisać jako $\sum_{j=1}^q (t^j - 1)/(t-1)$.

9. Indukcja względem rozmiaru drzewa.

10. Dodając (jeśli potrzeba) wagi zerowe, możemy założyć, że $m \bmod (t - 1) = 1$. Drzewo t -arne o minimalnej ważonej długości ścieżek otrzymujemy, zastępując w każdym kroku t najmniejszych wartości przez ich sumę. Dowód jest w zasadzie taki sam, jak w przypadku drzew binarnych. Poszukiwane drzewo ternarne jest pokazane na rysunku obok.

F. K. Hwang zauważył [SIAM J. Appl. Math. **37** (1979), 124–127], że podobna procedura działa dla minimalnej ważonej długości ścieżki dla drzew o zadanym multizbiorze stopni: w każdym kroku łączymy najmniejsze t wag, gdzie t jest jak najmniejsze.



11. „Notacja Dewey'a” jest dwójkową reprezentacją numeru węzła.

12. Z ćwiczenia 9 wynika, że jest to długość ścieżki wewnętrznej dzielona przez n , plus 1. (Ten fakt zachodzi także dla zwykłych drzew).

13. [Zobacz J. van Leeuwen, *Proc. 3rd International Colloq. Automata, Languages and Programming* (Edinburgh University Press, 1976), 382–410].

H1. [Inicjowanie] Przyjmij $A[m - 1 + i] \leftarrow w_i$ dla $1 \leq i \leq m$. Następnie przyjmij $A[2m] \leftarrow \infty$, $x \leftarrow m$, $i \leftarrow m + 1$, $j \leftarrow m - 1$, $k \leftarrow m$. (Podczas wykonania algorytmu $A[i] \leq \dots \leq A[2m - 1]$ jest kolejką niewykorzystanych wag zewnętrznych; $A[k] \geq \dots \geq A[j]$ jest kolejką niewykorzystanych wag wewnętrznych, pustą, jeśli $j < k$; x i y są bieżącymi wskaźnikami, lewym i prawym).

H2. [Znajdowanie prawego wskaźnika] Jeśli $j < k$ lub $A[i] \leq A[j]$, to przyjmij $y \leftarrow i$, $i \leftarrow i + 1$; w przeciwnym razie przyjmij $y \leftarrow j$, $j \leftarrow j - 1$.

H3. [Utworzenie węzła wewnętrznego] Przyjmij $k \leftarrow k - 1$, $L[k] \leftarrow x$, $R[k] \leftarrow y$, $A[k] \leftarrow A[x] + A[y]$.

H4. [Czy gotowe?] Zakończ wykonanie algorytmu, jeśli $k = 1$.

H5. [Znajdowanie lewego wskaźnika] (W tym miejscu $j \geq k$, a kolejki zawierają w sumie k niewykorzystanych wag. Jeśli $A[y] < 0$, to $j = k$, $i = y + 1$ i $A[i] > A[j]$). Jeśli $A[i] \leq A[j]$, to przyjmij $x \leftarrow i$, $i \leftarrow i + 1$; w przeciwnym razie przyjmij $x \leftarrow j$, $j \leftarrow j - 1$. Wróć do kroku H2. ■

14. Można zastosować dowód dla $k = m - 1$ z małymi zmianami. [Zobacz SIAM J. Appl. Math. **21** (1971), 518].

15. W (9) zamiast funkcji $w_1 + w_2$ do łączenia wag bierzemy (a) $1 + \max(w_1, w_2)$, (b) $xw_1 + xw_2$. [Część (a) pochodzi od M. C. Golumbica, IEEE Trans. **C-25** (1976), 1164–1167; część (b) od T. C. Hu, D. Kleitmana i J. K. Tamaki, SIAM J. Appl. Math. **37** (1979), 246–256. Problem Huffmana jest przypadkiem granicznym (b) przy $x \rightarrow 1$, ponieważ $\sum(1 + \epsilon)^{l_j} w_j = \sum w_j + \epsilon \sum w_j l_j + O(\epsilon^2)$].

D. Stott Parker, Jr. zauważył, że algorytm podobny do algorytmu Huffmana wyznaczy również minimum z $w_1 x^{l_1} + \dots + w_m x^{l_m}$, gdy $0 < x < 1$, jeśli jak w części (b) w każdym kroku połączymy dwie maksymalne wagi. W szczególności najmniejsza wartość z $w_1 2^{-l_1} + \dots + w_m 2^{-l_m}$, dla $w_1 \leq \dots \leq w_m$, wynosi $w_1/2 + \dots + w_{m-1}/2^{m-1} + w_m/2^m$. Zobacz D. E. Knuth, J. Comb. Theory **A32** (1982), 216–224.

16. Niech $l_{m+1} = l'_{m+1} = 0$. Wówczas

$$\sum_{j=1}^m w_j l_j \leq \sum_{j=1}^m w_j l'_j = \sum_{k=1}^m (l'_k - l'_{k+1}) \sum_{j=1}^k w_j \leq \sum_{k=1}^m (l'_k - l'_{k+1}) \sum_{j=1}^k w'_j = \sum_{j=1}^m w'_j l'_j,$$

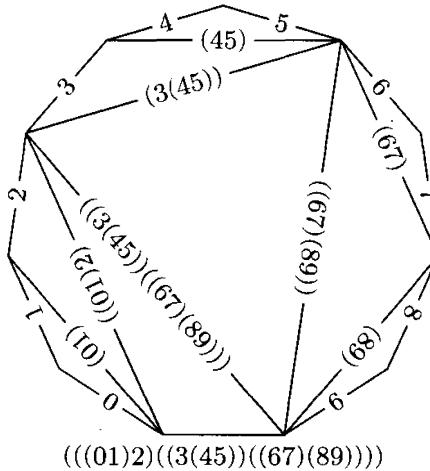
ponieważ $l'_j \geq l'_{j+1}$, jak w ćwiczeniu 4. Ten sam dowód pozostaje w mocy dla wielu innych drzew optymalnych, włączając te z ćwiczenia 10.

17. (a) Ćwiczenie 14. (b) Możemy rozszerzyć $f(n)$ do funkcji wklęszej $f(x)$, tak by zachodziła podana nierówność. Teraz niech $F(m)$ będzie najmniejszą z $\sum_{j=1}^{m-1} f(s_j)$, gdzie s_j są wagami węzłów wewnętrznych rozszerzonego drzewa binarnego o wagach 1, 1, ..., 1. Algorytm Huffmana, konstruujący w tym przypadku pełne drzewo binarne o $m-1$ węzłach wewnętrznych, daje drzewo optymalne. Wybór $k = 2^{\lceil \lg(n/3) \rceil}$ wyznacza drzewo binarne o tych samych wagach wewnętrznych, zatem daje minimum w równaniu rekurencyjnym dla dowolnego n . [SIAM J. Appl. Math. 31 (1976), 368–378]. Możemy wyznaczyć $F(n)$ w $O(\log n)$ krokach; zobacz ćwiczenie 5.2.3–20 i 5.2.3–21. Jeśli $f(n)$ jest wypukła zamiast wklęsła, tak że $\Delta^2 f(n) \geq 0$, rozwiązanie równania rekurencyjnego otrzymujemy dla $k = \lfloor n/2 \rfloor$.

2.3.4.6

1. Wybieramy jedną krawędź wielokąta i nazywamy ją bazą. Dla danej triangulacji trójkątów oparty na bazie odpowiada korzeniowi drzewa binarnego, a pozostałe jego boki są bazami dwóch podwielokątów, odpowiadających lewemu i prawemu poddrzewu. Postępujemy analogicznie, aż osiągniemy wielokąty „jednokrawędziowe”, odpowiadające pustym drzewom binarnym.

Mogą ten związek wyrazić inaczej. Etykietujemy krawędzie niebazowe strzegowane (podzielonego na trójkąty) wielokąta liczbami całkowitymi $0, \dots, n$. Jeśli dwa przyległe boki trójkąta mają etykiety α i β (zgodnie z ruchem wskazówek zegara), to trzeciemu bokowi nadajemy etykietę $(\alpha\beta)$. Etykietą bazy wyznacza drzewo binarne oraz triangulację. Na przykład



odpowiada drzewu binarnemu 2.3.1–(1).

2. (a) Tak jak w ćwiczeniu 1 przyjmujemy pewną krawędź za bazową. Ta krawędź jest krawędzią $(d+1)$ -kąta w podzielonym r -kącie – odpowiada jej węzeł mający d dzieci, odpowiadających pozostałym d krawędziom będącym bazami podwielokątów r -kąta. W ten sposób jest wyznaczany związek między problemem Kirkmana a problemem zliczania drzew o $r-1$ liściach i $k+1$ węzłach wewnętrznych, nie mających węzłów stopnia 1. (Dla $k=r-3$ jest to sytuacja z ćwiczenia 1).

(b) Istnieje $\binom{r+k}{k} \binom{r-3}{k}$ ciągów $d_1 d_2 \dots d_{r+k}$ nieujemnych liczb całkowitych, takich że $r-1$ spośród d są zerami, żadne nie jest jedynką, a ich suma wynosi $r+k-1$. Dokładnie jedna z cyklicznych permutacji $d_1 d_2 \dots d_{r+k}$, $d_2 \dots d_{r+k} d_1, \dots, d_{r+k} d_1 \dots d_{r+k-1}$ spełnia ponadto warunek $\sum_{j=1}^q (1 - d_j) > 0$ dla $1 \leq q \leq r+k$.

[Kirkman podał argumenty przemawiające za prawdziwością jego hipotezy w *Philos. Trans.* **147** (1857), 217–272, §22. Cayley udowodnił ją w *Proc. London Math. Soc.* **22** (1891), 237–262, nie dostrzegając związku z drzewami].

3. (a) Przyjmijmy, że ponumerowano wierzchołki liczbami $\{1, 2, \dots, n\}$. Rysujemy dowiązanie RLINK od i do j , jeśli i i j są kolejnymi elementami z tej samej części $i < j$. Rysujemy dowiązanie LLINK z j do $j+1$, jeśli $j+1$ jest najmniejsze w swojej części. Mamy $k-1$ niepustych dowiązań LLINK, $n-k$ niepustych dowiązań RLINK oraz drzewo binarne, którego węzłami w porządku preorder są $12\dots n$. Przez naturalną odpowiedniość (punkt 2.3.2) ta reguła wyznacza odpowiedniość wzajemnie jednoznaczna między „podziałami wierzchołków n -kąta na k nieprzecinających się części” i „łasami o n wierzchołkach i $n-k+1$ liściach”. Zamieniając LLINK i RLINK, otrzymamy „łasy o n wierzchołkach i k liściach”.

(b) Las o n wierzchołkach i k liściach odpowiada także ciągowi zagnieżdżonych nawiasów zawierającemu n nawiasów lewych, n nawiasów prawych i k wystąpienie „()”. Możemy zliczyć takie ciągi w następujący sposób:

Mówimy, że ciąg zer i jedynek jest (m, n, k) -ciągiem, jeśli zawiera m zer, n jedynek i k wystąpienie „01”. Na przykład 0010101001110 jest $(7, 6, 4)$ -ciągiem. Liczba (m, n, k) -ciągów to $\binom{m}{k} \binom{n}{k}$, ponieważ dowolnie wybieramy, które spośród zer i jedynek mają utworzyć pary 01.

Niech $S(\alpha)$ będzie liczbą zer w α minus liczba jedynek. Mówimy że ciąg σ jest *dobry*, jeśli $S(\alpha) \geq 0$, o ile α jest prefiksem σ (innymi słowy, jeśli z faktu $\sigma = \alpha\beta$ wynika, że $S(\alpha) \geq 0$); w przeciwnym razie σ jest ciągiem *zły*. Oto sposób alternatywny wobec „zasady odwracania” z ćwiczenia 2.2.1–4 wyznaczania wzajemnie jednoznacznej odpowiedniości między złymi (n, n, k) -ciągami a wszystkimi $(n-1, n+1, k)$ -ciągami:

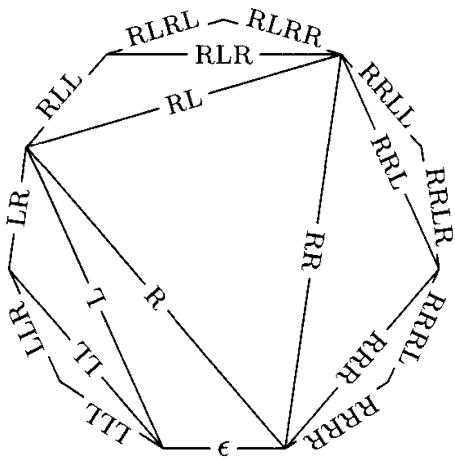
Dowolny zły (n, n, k) -ciąg σ można jednoznacznie zapisać w postaci $\sigma = \alpha 0 \beta$, gdzie $\bar{\alpha}^R$ i β są dobre. ($\bar{\alpha}^R$ oznacza ciąg powstały z α przez zapisanie w odwrotnej kolejności i odwrócenie bitów). Wówczas $\sigma' = \alpha 1 \beta$ jest $(n-1, n+1, k)$ -ciągiem. W drugą stronę, każdy $(n-1, n+1, k)$ -ciąg można zapisać jednoznacznie w postaci $\alpha 1 \beta$, gdzie $\bar{\alpha}^R$ i β są dobre, zatem $\alpha 0 \beta$ jest zły (n, n, k) -ciągiem.

Wynika stąd, że liczba łasów o n wierzchołkach i k liściach równa się $\binom{n}{k} \binom{n}{k} - \binom{n-1}{k} \binom{n+1}{k} = n! (n-1)! / (n-k+1)! (n-k)! k! (k-1)!$

Uwagi: G. Kreweras, *Discrete Math.* **1** (1972), 333–350, podał kilka różnych sposobów zliczania nieprzecinających się podziałów. Częściowe uporządkowanie podziałów przez zagęszczanie prowadzi do ciekawego częściowego porządku na łasach, różnego od kraty Tamariego omówionej w ćwiczeniu 2.3.3–19; zobacz Y. Poupard, *Cahiers du Bureau Univ. de Recherche Opérationnelle* **16** (1971), rozdział 8; *Discrete Math.* **2** (1972), 279–288; P. Edelman, *Discrete Math.* **31** (1980), 171–180, **40** (1982), 171–179.

Trzeci sposób zdefiniowania naturalnego kratowego porządku na łasach został pokazany przez R. Stanleya w *Fibonacci Quarterly* **13** (1975), 215–232. Przypuśćmy, że reprezentujemy las za pomocą ciągu σ złożonego z zer i jedynek odpowiadających lewym i prawym nawiasom, jak w ćwiczeniu powyżej. Wówczas $\sigma \leq \sigma'$ wtedy i tylko wtedy, gdy $S(\sigma_k) \leq S(\sigma'_k)$ dla wszystkich k , gdzie σ_k oznacza pierwsze k bitów ciągu σ . Krata Stanley'a jest *dystrybutywna* w odróżnieniu od dwóch pozostałych.

4. Niech $m = n + 2$; na mocy ćwiczenia 1 poszukujemy odpowiedniości między triangulowanymi m -kątami i $(m - 1)$ -wierszowymi fryzami. Zacznijmy od przyjrzenia się poprzedniej odpowiedniości, definiując etykietowanie krawędzi w sposób „zstępujący”, a nie „wstępujący”. Przypisujemy bazie etykię pustą ϵ , następnie rekurencyjnie nadajemy etykiety αL i αR krawędziom trójkątów, których jeden bok dostał etykię α . W tej konwencji poprzedni rysunek na przykład przyjmie postać



Jeśli krawędź bazowa w tym przykładzie ma numer 10, a poprzednie krawędzie numery 0–9 (tak jak przyjeliśmy), to możemy napisać $0 = 10LLL$, $1 = 10LLR$, $2 = 10LR$, $3 = 10RLL$ itd. Każdą inną krawędź można uczynić bazą. Jeśli weźmiemy 0, otrzymamy $1 = 0L$, $2 = 0RL$, $3 = 0RRLLL$ itd. Nietrudno sprawdzić, że jeśli $u = v\alpha$, to mamy $v = u\alpha^T$, gdzie α^T powstaje z α przez zapisanie symboli od prawej do lewej oraz zamianę L i R . Na przykład $10 = 0RRR = 1LRR = 2LR = 3RRL$ itd. Jeśli u , v i w są krawędziami wielokąta, takimi że $w = u\alpha L\gamma$ i $w = v\beta R\gamma$, to $u = v\beta L\alpha^T$ i $v = u\alpha R\beta^T$.

Dla danej triangulacji wielokąta, której krawędzie są ponumerowane liczbami $0, 1, \dots, m-1$, definiujemy (u, v) dla dowolnej pary różnych krawędzi u i v . Niech $u = v\alpha$, gdzie α jest macierzą 2×2 , którą otrzymamy biorąc $L = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ i $R = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$. (u, v) definiujemy jako element w górnym lewym rogu macierzy α . Zauważmy, że α^T jest transpozycją macierzy α , ponieważ $R = L^T$. Stąd mamy $(v, u) = (u, v)$. Zauważmy także, że $(u, v) = 1$ wtedy i tylko wtedy, gdy u_- i v_- są połączone krawędzią triangulacji, gdzie u_- oznacza wezel między krawedziami u i $u-1$.

Niech $(u, u) = 0$ dla wszystkich krawędzi u wielokąta. Możemy udowodnić, że $v = u\alpha$ implikuje

$$\alpha = \begin{pmatrix} (u, v) & (u, v+1) \\ (u+1, v) & (u+1, v+1) \end{pmatrix} \quad \text{dla wszystkich } u \neq v, \quad (*)$$

gdzie $u + 1$ i $v + 1$ są następnikami (zgodnie z ruchem wskazówek zegara) węzłów u i v . Dowodzimy przez indukcję względem m : $(*)$ jest trywialne dla $m = 2$, ponieważ dwie równoległe krawędzie u i v są w związku $u = v\epsilon$, a $\alpha = \epsilon$ jest macierzą jednostkową. Jeśli w dowolnej triangulacji do pewnej krawędzi v dostawimy trójkąt $v v' v''$, to przy $v = u\alpha$ mamy $v' = u\alpha L$ i $v'' = u\alpha R$. Stąd (u, v') i (u, v'') w rozszerzonym wielokącie są odpowiednio równe (u, v) i $(u, v) + (u, v + 1)$ w wielokącie pierwotnym. Wynika stąd, że

$$\alpha L = \begin{pmatrix} (u, v') & (u, v'') \\ (u+1, v') & (u+1, v'') \end{pmatrix} \quad \text{oraz} \quad \alpha R = \begin{pmatrix} (u, v'') & (u, v'' + 1) \\ (u+1, v'') & (u+1, v'' + 1) \end{pmatrix},$$

a (*) pozostaje prawdziwe w rozszerzonym wielokącie.

Wzór fryzowy odpowiadający danej triangulacji definiujemy jako ciąg okresowy

$$\begin{array}{ccccccccc}
 (0, 1) & (1, 2) & (2, 3) & \dots & (m-1, 0) & (0, 1) & (1, 2) & \dots \\
 & (0, 2) & (1, 3) & (2, 4) & \dots & (m-1, 1) & (0, 2) & (1, 3) & \dots \\
 (m-1, 2) & (0, 3) & (1, 4) & \dots & (m-2, 1) & (m-1, 2) & (0, 3) & \dots \\
 (m-1, 3) & (0, 4) & (1, 5) & \dots & (m-2, 2) & (m-1, 3) & (0, 4) & \dots
 \end{array}$$

i tak dalej aż do $m - 1$ pierwszego wiersza. Ostatni wiersz zaczyna się od $(\lceil m/2 \rceil + 1, \lfloor m/2 \rfloor)$ dla $m > 3$. Warunek (*) dowodzi, że ten wzór jest wzorem fryzowym, tj. że

$$(u, v)(u+1, v+1) - (u, v+1)(u+1, v) = 1, \quad (**)$$

ponieważ $\det L = \det R = 1$ implikuje $\det \alpha = 1$. Dla naszej przykładowej triangulacji otrzymamy

$$\begin{array}{cccccccccccccccccccc}
 1 & \dots \\
 1 & 2 & 4 & 2 & 1 & 5 & 1 & 3 & 1 & 4 & 3 & 1 & 2 & 4 & 2 & 1 & 5 & 1 & 3 & 1 & 4 & \dots \\
 2 & 1 & 7 & 7 & 1 & 4 & 4 & 2 & 2 & 3 & 11 & 2 & 1 & 7 & 7 & 1 & 4 & 4 & 2 & 2 & 3 & \dots \\
 1 & 3 & 12 & 3 & 3 & 3 & 7 & 1 & 5 & 8 & 7 & 1 & 3 & 12 & 3 & 3 & 3 & 7 & 1 & 5 & 8 & \dots \\
 3 & 2 & 5 & 5 & 8 & 2 & 5 & 3 & 2 & 13 & 5 & 3 & 2 & 5 & 5 & 8 & 2 & 5 & 3 & 2 & 13 & \dots \\
 5 & 3 & 2 & 13 & 5 & 3 & 2 & 5 & 5 & 8 & 2 & 5 & 3 & 2 & 13 & 5 & 3 & 2 & 5 & 5 & 8 & \dots \\
 3 & 7 & 1 & 5 & 8 & 7 & 1 & 3 & 12 & 3 & 3 & 3 & 7 & 1 & 5 & 8 & 7 & 1 & 3 & 12 & 3 & \dots \\
 4 & 2 & 2 & 3 & 11 & 2 & 1 & 7 & 7 & 1 & 4 & 4 & 2 & 2 & 3 & 11 & 2 & 1 & 7 & 7 & 1 & \dots \\
 5 & 1 & 3 & 1 & 4 & 3 & 1 & 2 & 4 & 2 & 1 & 5 & 1 & 3 & 1 & 4 & 3 & 1 & 2 & 4 & 2 & \dots \\
 1 & \dots
 \end{array}$$

Warunek $(u, v) = 1$ wyznacza krawędzie triangulacji, stąd różne triangulacje dają różne wzory fryzowe. By zakończyć dowód wzajemnej jednoznaczności odpowiedniości, musimy pokazać, że każdy $(m-1)$ -wierszowy wzór fryzowy dodatnich liczb całkowitych można w powyższy sposób otrzymać z pewnej triangulacji.

Bierzemy dowolny fryz o $m - 1$ wierszach i rozszerzamy go, dodając nowy wiersz 0 na górze i nowy wiersz m na dole, złożone z samych zer. Niech teraz elementy w wierszu 0 nazywają się $(0, 0), (1, 1), (2, 2)$ itd. i niech dla wszystkich nieujemnych liczb całkowitych $u < v \leq u + m$ symbol (u, v) oznacza element na południowo-wschodniej przekątnej zawierającej (u, u) i południowo-zachodniej przekątnej zawierającej (v, v) . Z założenia warunek (*) zachodzi dla wszystkich $u < v < u + m$. Możemy rozszerzyć (*) na znacznie ogólniejszy związek

$$(t, u)(v, w) + (t, w)(u, v) = (t, v)(u, w) \quad \text{dla } t \leq u \leq v \leq w \leq t + m. \quad (***)$$

Założmy, że (***) jest fałszywe. Niech (t, u, v, w) będzie kontrprzykładem, dla którego wartość $(w-t)m + u - t + w - v$ jest najmniejsza. Przypadek 1: $t + 1 < u$. Wówczas (***) zachodzi dla $(t, t+1, v, w), (t, t+1, u, v)$ i $(t+1, u, v, w)$, zatem $((t, u)(v, w) + (t, w)(v, u))(t+1, v) = (t, v)(u, w)(t+1, v)$; to implikuje $(t+1, v) = 0$, sprzeczność. Przypadek 2: $v + 1 < w$. Wówczas (***) zachodzi dla $(t, u, w-1, w), (u, v, w-1, w)$ i $(t, u, v, w-1)$; otrzymujemy podobną sprzeczność $(u, w-1) = 0$. Przypadek 3: $u = t+1$ i $w = v + 1$. W tym przypadku (***) redukuje się do (*).

Bierzemy teraz $u = t+1$ i $w = t+m$ w (***), skąd dostajemy $(t, v) = (v, t+m)$ dla $t \leq v \leq t+m$, ponieważ $(t+1, t+m) = 1$ i $(t, t+m) = 0$. Wnioskujemy, że elementy dowolnego $(m-1)$ -wierszowego wzoru fryzowego tworzą cykl: $(u, v) = (v, u+m) = (u+m, v+m) = (v+m, u+2m) = \dots$.

Każdy wzór fryzowy dodatnich liczb całkowitych zawiera 1 w wierszu 2. Jest tak, ponieważ jeśli weźmiemy $t = 0, v = u + 1$ i $w = u + 2$ w (***), to otrzymamy $(0, u+1)(u, u+2) = (0, u) + (0, u+2)$, stąd $(0, u+2) - (0, u+1) \geq (0, u+1) - (0, u)$

wtedy i tylko wtedy, gdy $(u, u+2) \geq 2$. To nie może być prawdą dla wszystkich u z przedziału $0 \leq u \leq m-2$, ponieważ $(0,1)-(0,0)=1$ i $(0,m)-(0,m-1)=-1$.

Wreszcie, jeśli $m > 3$, nie możemy mieć dwóch kolejnych jedynek w wierszu 2, ponieważ $(u, u+2) = (u+1, u+3) = 1$ pociąga $(u, u+3) = 0$. Stąd możemy zredukować dany wzór fryzowy do innego fryzu o liczbie wierszy m zmniejszonej o jeden, jak w poniższym przykładzie:

$$\begin{array}{ccccccccccccc}
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\
 a & b & c & d+1 & 1 & e+1 & y & z & \dots & a & b & c & d & e & y & z & \dots \\
 p & q & c+r & d & e & u+y & v & w & \dots & p & q & r & s & u & v & w & \dots \\
 u & q+v & r & s & u & q+v & r & s & \dots & u & v & w & p & q & r & s & \dots \\
 u+y & v & w & p & q & c+r & d & e & \dots & y & z & a & b & c & d & e & \dots \\
 y & z & a & b & c & d+1 & 1 & e+1 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots
 \end{array}$$

Zredukowany fryz odpowiada triangulacji, a niezredukowany fryz odpowiada dołączeniu nowego trójkąta. [Math. Gazette 57 (1974), 87–94, 175–183; Conway, Guy, Księga liczb (Warszawa: WNT, 1996), 85–87, 93–95, 106–107].

Uwagi: Powyższy dowód pokazuje, że funkcja (u, v) , którą zdefiniowaliśmy za pomocą macierzy 2×2 dla dowolnej triangulacji, spełnia (**), jeśli (t, u, v, w) są krawędziami wielokąta w porządku zgodnym z ruchem wskazówek zegara. Możemy wyrazić (u, v) jako funkcję wielomianową liczb $a_j = (j-1, j+1)$; takie wielomiany są w istocie tym samym, co „kontynuanty” omówione w punkcie 4.5.3, z wyjątkiem znaku poszczególnych wyrazów. W istocie $(j, k) = i^{1-k+j} K_{k-j-1}(ia_{j+1}, ia_{j+2}, \dots, ia_{k-1})$. Stąd (**) jest równoważne tożsamości Eulera dla kontynuant z odpowiedzi do ćwiczenia 4.5.3–32. Macierze L i R mają tę ciekawą własność, że dowolną macierz 2×2 o elementach nieujemnych całkowitych i wyznaczniku 1 można jednoznacznie przedstawić jako iloczyn (wielu) macierzy L i R .

To nie koniec ciekawych związków; na przykład liczby w drugim wierszu fryzu całkowitoliczbowego wzoru fryzowego są liczbami trójkątów dotykających każdego wierzchołka w odpowiadającym fryzowi striangulowanemu wielokątowi. Całkowita liczba wystąpień $(u, v) = 1$ w obszarze $0 \leq u < v-1 < m-1$ i $(u, v) \neq (0, m-1)$ jest liczbą przekątnych triangulacji, tj. $m-3=n-1$. Całkowita liczba dwójek też wynosi $n-1$, ponieważ $(u, v) = 2$ wtedy i tylko wtedy, gdy u_- i v_- są przeciwnymi wierzchołkami dwóch trójkątów przylegających do przekątnej.

Inną interpretację (u, v) odkryli D. Broline, D. W. Crowe i I. M. Isaacs [Geometriæ Dedicata 3 (1974), 171–176]: jest to liczba sposobów ustawienia $v-u-1$ wierzchołków między krawędziami u i $v-1$, tak by trójkąty dotykające tych węzłów były różne.

2.3.5

1. Lista jest grafem skierowanym, w którym krawędzie wychodzące z każdego wierzchołka są uporządkowane i w którym niektóre wierzchołki o stopniu wyjściowym równym zero są wyróżnione jako „atomy”. Jest w nim ponadto taki wierzchołek S , że istnieje ścieżka zorientowana od S do V dla dowolnego wierzchołka $V \neq S$. (Gdyby odwrócić kierunki krawędzi, wierzchołek S byłby „korzeniem”).

2. Nie w ten sam sposób, ponieważ fastryga w standardowej reprezentacji prowadzi do rodzica, który w przypadku pod-List nie jest jedyny. Można zapewne skorzystać z reprezentacji omówionej w ćwiczeniu 2.3.4.2–25 lub z podobnych metod (ten pomysł nie był jednak jeszcze zbadany, gdy rozdział ten powstawał).

3. Jak powiedziano w tekście, dowodzimy dodatkowo, że $P = P_0$ w chwili zakończenia wykonania algorytmu. Jeśli jedynym zaznaczanym elementem jest P_0 , to algorytm oczywiście działa poprawnie. Jeśli trzeba oznaczyć $n > 1$ elementów, to musi być $\text{ATOM}(P_0) = 0$. W kroku E4 przypisujemy $\text{ALINK}(P_0) \leftarrow \Lambda$ i wykonujemy algorytm z $\text{ALINK}(P_0)$ podstawionym w miejsce P_0 oraz P_0 podstawionym w miejsce T . Na mocy indukcji (zauważmy, że $\text{MARK}(P_0) = 1$, zatem wszystkie wskaźniki do P_0 są równe Λ , co wynika z kroków E4 i E5) oznaczamy wszystkie elementy na ścieżkach rozpoczynających się w $\text{ALINK}(P_0)$ i nie prowadzących przez P_0 . Następnie przechodzimy do E6, $T = P_0$ i $P = \text{ALINK}(P_0)$. Teraz mamy $\text{ATOM}(T) = 1$, zatem w kroku E6 odtwarzamy $\text{ALINK}(P_0)$ i $\text{ATOM}(P_0)$ i przechodzimy do kroku E5. W kroku E5 przypisujemy $\text{BLINK}(P_0) \leftarrow \Lambda$ itd. Z podobnego rozumowania wynika, że oznaczamy wszystkie elementy na ścieżkach zaczynających się w $\text{BLINK}(P_0)$ i nie prowadzących przez P_0 lub elementy osiągalne z $\text{ALINK}(P_0)$. Następnie przechodzimy do E6, $T = P_0$, $P = \text{BLINK}(P_0)$, by po raz kolejny dostać się do E6 z wartościami zmiennych $T = \Lambda$, $P = P_0$.

4. Poniższy program zawiera sugerowane poprawki dotyczące przetwarzania atomów (podane w tekście po opisie algorytmu E).

W krokach E4 i E5 sprawdzamy, czy $\text{MARK}(Q) = 0$. Jeśli $\text{NODE}(Q) = +0$, to mamy do czynienia ze specjalnym przypadkiem, z którym można sobie poradzić, zmieniając wartość na -0 i traktując ją tak, jakby pierwotnie była równa -0 , ponieważ oba wskaźniki ALINK i BLINK są równe Λ . Tego uproszczenia nie odzwierciedlają obliczenia czasu działania programu.

$rI1 \equiv P$, $rI2 \equiv T$, $rI3 \equiv Q$, i $rX \equiv -1$ (obsługa pól MARK).

01	MARK	EQU	0:0		
02	ATOM	EQU	1:1		
03	ALINK	EQU	2:3		
04	BLINK	EQU	4:5		
05	E1	LD1	P0	1	<u>E1. Inicjowanie.</u> $P \leftarrow P_0$.
06		ENT2	0	1	$T \leftarrow \Lambda$.
07		ENTX	-1	1	$rX \leftarrow -1$.
08	E2	STX	0,1(MARK)	1	<u>E2. Oznaczanie.</u> $\text{MARK}(P) \leftarrow 1$.
09	E3	LDA	0,1(ATOM)	1	<u>E3. Atom?</u>
10		JAZ	E4	1	Skocz, jeśli $\text{ATOM}(P) = 0$.
11	E6	J2Z	DONE	n	<u>E6. W góre.</u>
12		ENT3	0,2	n - 1	$Q \leftarrow T$.
13		LDA	0,3(ATOM)	n - 1	
14		JANZ	1F	n - 1	Skocz, jeśli $\text{ATOM}(T) = 1$.
15		LD2	0,3(BLINK)	t ₂	$T \leftarrow \text{BLINK}(Q)$.
16		ST1	0,3(BLINK)	t ₂	$\text{BLINK}(Q) \leftarrow P$.
17		ENT1	0,3	t ₂	$P \leftarrow Q$.
18		JMP	E6	t ₂	
19	1H	STZ	0,2(ATOM)	t ₁	$\text{ATOM}(T) \leftarrow 0$.
20		LD2	0,3(ALINK)	t ₁	$T \leftarrow \text{ALINK}(Q)$.
21		ST1	0,3(ALINK)	t ₁	$\text{ALINK}(Q) \leftarrow P$.
22		ENT1	0,3	t ₁	$P \leftarrow Q$.
23	E5	LD3	0,1(BLINK)	n	<u>E5. W dół po BLINK.</u> $Q \leftarrow \text{BLINK}(P)$.
24		J3Z	E6	n	Skocz, jeśli $Q = \Lambda$.
25		LDA	0,3	n - b ₂	
26		STX	0,3(MARK)	n - b ₂	$\text{MARK}(Q) \leftarrow 1$.

27	JANP E6	$n - b_2$	Skocz, jeśli NODE(Q) już oznaczony.
28	LDA 0,3(ATOM)	$t_2 + a_2$	
29	JANZ E6	$t_2 + a_2$	Skocz, jeśli ATOM(Q) = 1.
30	ST2 0,1(BLINK)	t_2	BLINK(P) $\leftarrow T$.
31	E4A ENT2 0,1	$n - 1$	$T \leftarrow P$.
32	ENT1 0,3	$n - 1$	$P \leftarrow Q$.
33	E4 LD3 0,1(ALINK)	n	<u>E4. W dół po ALINK. $Q \leftarrow ALINK(P)$.</u>
34	J3Z E5	n	Skocz, jeśli $Q = \Lambda$.
35	LDA 0,3	$n - b_1$	
36	STX 0,3(MARK)	$n - b_1$	MARK(Q) $\leftarrow 1$.
37	JANP E5	$n - b_1$	Skocz, jeśli NODE(Q) już oznaczony.
38	LDA 0,3(ATOM)	$t_1 + a_1$	
39	JANZ E5	$t_1 + a_1$	Skocz, jeśli ATOM(Q) = 1.
40	STX 0,1(ATOM)	t_1	ATOM(P) $\leftarrow 1$.
41	ST2 0,1(ALINK)	t_1	ALINK(P) $\leftarrow T$.
42	JMP E4A	t_1	$T \leftarrow P$, $P \leftarrow Q$, idź do E4. ■

Na mocy prawa Kirchhoffa $t_1 + t_2 + 1 = n$. Całkowity czas jest równy $(34n + 4t_1 + 3a - 5b - 8)u$, gdzie n jest liczbą oznaczonych elementów nieatomowych, a jest liczbą oznaczonych elementów atomowych, b jest liczbą dowiązań Λ napotkanych przy oznaczaniu elementów nieatomowych, a t_1 jest liczbą przejść w przód po dowiązaniu ALINK ($0 \leq t_1 < n$).

5. (Poniższy algorytm jest najszybszym ze znanych algorytmów oznaczania dla pamięci jednopoziomowej).

- S1. Przyjmij $MARK(P_0) \leftarrow 1$. Jeśli $ATOM(P_0) = 1$, to zakończ działanie algorytmu; w przeciwnym razie przyjmij $S \leftarrow 0$, $R \leftarrow P_0$, $T \leftarrow \Lambda$.
- S2. Przyjmij $P \leftarrow BLINK(R)$. Jeśli $P = \Lambda$ lub $MARK(P) = 1$, to przejdź do S3. W przeciwnym razie przyjmij $MARK(P) \leftarrow 1$. Jeśli teraz $ATOM(P) = 1$, to przejdź do S3; w przeciwnym razie, jeśli $S < N$, to przyjmij $S \leftarrow S + 1$, $STACK[S] \leftarrow P$ i przejdź do S3; w przeciwnym razie idź do S5.
- S3. Przyjmij $P \leftarrow ALINK(R)$. Jeśli $P = \Lambda$ lub $MARK(P) = 1$, to idź do S4. W przeciwnym razie przyjmij $MARK(P) \leftarrow 1$. Jeśli $ATOM(P) = 1$, idź do S4; w przeciwnym razie przyjmij $R \leftarrow P$ i wróć do S2.
- S4. Jeśli $S = 0$, to zakończ działanie algorytmu; w przeciwnym razie przyjmij $R \leftarrow STACK[S]$, $S \leftarrow S - 1$ i przejdź do S2.
- S5. Przyjmij $Q \leftarrow ALINK(P)$. Jeśli $Q = \Lambda$ lub $MARK(Q) = 1$, to idź do S6. W przeciwnym razie przyjmij $MARK(Q) \leftarrow 1$. Jeśli $ATOM(Q) = 1$, to przejdź do S6; w przeciwnym razie przyjmij $ATOM(P) \leftarrow 1$, $ALINK(P) \leftarrow T$, $T \leftarrow P$, $P \leftarrow Q$ i idź do S5.
- S6. Przyjmij $Q \leftarrow BLINK(P)$. Jeśli $Q = \Lambda$ lub $MARK(Q) = 1$, to przejdź do S7; w przeciwnym razie przyjmij $MARK(Q) \leftarrow 1$. Jeśli $ATOM(Q) = 1$, to idź do S7; w przeciwnym razie przyjmij $BLINK(P) \leftarrow T$, $T \leftarrow P$, $P \leftarrow Q$ i idź do S5.
- S7. Jeśli $T = \Lambda$, to idź do S3. W przeciwnym razie przyjmij $Q \leftarrow T$. Jeśli $ATOM(Q) = 1$, to $ATOM(Q) \leftarrow 0$, $T \leftarrow ALINK(Q)$, $ALINK(Q) \leftarrow P$, $P \leftarrow Q$ i wróć do S6. Jeśli $ATOM(Q) = 0$, to przyjmij $T \leftarrow BLINK(Q)$, $BLINK(Q) \leftarrow P$, $P \leftarrow Q$, wróć do S7. ■

Źródło: CACM 10 (1967), 501–506.

6. Z drugiej fazy odśmiecania (a w niektórych systemach także stąd, że wszystkie znaczniki musimy ustawić na zero).

7. Usuwamy kroki E2 i E3, usuwamy „ $\text{ATOM}(P) \leftarrow 1$ ” w E4. Przypisujemy $\text{MARK}(P) \leftarrow 1$ w kroku E5, a w kroku E6 wstawiamy „ $\text{MARK}(Q) = 0$ ”, „ $\text{MARK}(Q) = 1$ ” odpowiednio w miejsce „ $\text{ATOM}(Q) = 1$ ”, „ $\text{ATOM}(Q) = 0$ ”. Pomysł polega na ustawianiu bitu MARK jedynie po oznaczeniu lewego poddrzewa. Algorytm działa nawet wtedy, gdy drzewo zawiera nakładające się poddrzewa, ale nie działa na Listach rekurencyjnych, jak na przykład Lista, w której $\text{NODE}(\text{ALINK}(Q))$ jest przodkiem $\text{NODE}(Q)$. (Zauważmy, że pole ALINK oznaczanego elementu nie jest zmieniane).

8. Rozwiążanie 1: Analogiczne do algorytmu E, ale prostsze.

F1. Przyjmij $T \leftarrow \Lambda$, $P \leftarrow P_0$.

F2. Przyjmij $\text{MARK}(P) \leftarrow 1$, $P \leftarrow P + \text{SIZE}(P)$.

F3. Jeśli $\text{MARK}(P) = 1$, to idź do F5.

F4. Przyjmij $Q \leftarrow \text{LINK}(P)$. Jeśli $Q \neq \Lambda$ i $\text{MARK}(Q) = 0$, to przyjmij $\text{LINK}(P) \leftarrow T$, $T \leftarrow P$, $P \leftarrow Q$ i idź do F2. W przeciwnym razie przyjmij $P \leftarrow P - 1$ i wróć do F3.

F5. Jeśli $T = \Lambda$, to zakończ wykonanie algorytmu. W przeciwnym razie przyjmij $Q \leftarrow T$, $T \leftarrow \text{LINK}(Q)$, $\text{LINK}(Q) \leftarrow P$, $P \leftarrow Q - 1$ i powróć do F3. ■

Podobny algorytm, który czasami ma mniejszy narzut pamięciowy i ma zabezpieczenie przed dowiązaniami prowadzącymi w środek elementu, zaproponowano w: Lars-Erik Thorelli, *BIT* 12 (1972), 555–568.

Rozwiążanie 2: Analogiczne do algorytmu D. W tym rozwiążaniu zakładamy, że pole SIZE jest wystarczająco duże, by pomieścić wskaźnik. Takie założenie zapewne nie jest usprawiedliwione przy podanym sformułowaniu problemu, ale tam gdzie daje się zastosować, pozwala skorzystać z nieco szybszej metody niż rozwiązanie pierwsze.

G1. Przyjmij $T \leftarrow \Lambda$, $\text{MARK}(P_0) \leftarrow 1$, $P \leftarrow P_0 + \text{SIZE}(P_0)$.

G2. Jeśli $\text{MARK}(P) = 1$, to idź do G5.

G3. Przyjmij $Q \leftarrow \text{LINK}(P)$, $P \leftarrow P - 1$.

G4. Jeśli $Q \neq \Lambda$ i $\text{MARK}(Q) = 0$, to przyjmij $\text{MARK}(Q) \leftarrow 1$, $S \leftarrow \text{SIZE}(Q)$, $\text{SIZE}(Q) \leftarrow T$, $T \leftarrow Q + S$. Idź do G2.

G5. Jeśli $T = \Lambda$, to zakończ wykonanie algorytmu. W przeciwnym razie przyjmij $P \leftarrow T$ i znajdź pierwszą wartość $Q = P, P-1, P-2, \dots$ dla której $\text{MARK}(Q) = 1$; przypisz $T \leftarrow \text{SIZE}(Q)$ i $\text{SIZE}(Q) \leftarrow P - Q$. Idź do G2. ■

9. H1. Przyjmij $L \leftarrow 0$, $K \leftarrow M + 1$, $\text{MARK}(0) \leftarrow 1$, $\text{MARK}(M + 1) \leftarrow 0$.

H2. Zwięksź L o jeden i jeśli $\text{MARK}(L) = 1$, powtórz ten krok.

H3. Zmniejsz K o jeden i jeśli $\text{MARK}(K) = 0$, powtórz ten krok.

H4. Jeśli $L > K$, to idź do H5; w przeciwnym razie przyjmij $\text{NODE}(L) \leftarrow \text{NODE}(K)$, $\text{ALINK}(K) \leftarrow L$, $\text{MARK}(K) \leftarrow 0$ i wróć do H2.

H5. Dla $L = 1, 2, \dots, K$ wykonaj co następuje: przypisz $\text{MARK}(L) \leftarrow 0$; jeśli $\text{ATOM}(L) = 0$ i $\text{ALINK}(L) > K$, to $\text{ALINK}(L) \leftarrow \text{ALINK}(\text{ALINK}(L))$; jeśli $\text{ATOM}(L) = 0$ i $\text{BLINK}(L) > K$, to $\text{BLINK}(L) \leftarrow \text{ALINK}(\text{BLINK}(L))$. ■

Zobacz też ćwiczenie 2.5–33.

- 10.** **Z1.** [Inicjowanie] Przyjmij $F \leftarrow P_0$, $R \leftarrow \text{AVAIL}$, $\text{NODE}(R) \leftarrow \text{NODE}(F)$, $\text{REF}(F) \leftarrow R$. (F i R są wskaźnikami kolejki pól REF zawierającej wszystkie napotkane atrapy).
- Z2.** [Rozpoczęcie nowej Listy] Przyjmij $P \leftarrow F$, $Q \leftarrow \text{REF}(P)$.
- Z3.** [Przesuwanie się w prawo] Przyjmij $P \leftarrow \text{RLINK}(P)$. Jeśli $P = \Lambda$, to przejdź do Z6.
- Z4.** [Kopiowanie jednego elementu] Przyjmij $Q_1 \leftarrow \text{AVAIL}$, $\text{RLINK}(Q) \leftarrow Q_1$, $Q \leftarrow Q_1$, $\text{NODE}(Q) \leftarrow \text{NODE}(P)$.
- Z5.** [Wskaźnik do pod-Listy.] Jeśli $T(P) = 1$, to przyjmij $P_1 \leftarrow \text{REF}(P)$ i jeśli $\text{REF}(P_1) = \Lambda$, to przyjmij $\text{REF}(R) \leftarrow P_1$, $R \leftarrow \text{AVAIL}$, $\text{REF}(P_1) \leftarrow R$, $\text{NODE}(R) \leftarrow \text{NODE}(P_1)$, $\text{REF}(Q) \leftarrow R$. Jeśli $T(P) = 1$ i $\text{REF}(P_1) \neq \Lambda$, to przyjmij $\text{REF}(Q) \leftarrow \text{REF}(P_1)$. Idź do Z3.
- Z6.** [Następna Lista] Przyjmij $\text{RLINK}(Q) \leftarrow \Lambda$. Jeśli $\text{REF}(F) \neq R$, to przyjmij $F \leftarrow \text{REF}(\text{REF}(F))$ i wróć do Z2. W przeciwnym razie przyjmij $\text{REF}(R) \leftarrow \Lambda$, $P \leftarrow P_0$.
- Z7.** [Sprzątanie] Przyjmij $Q \leftarrow \text{REF}(P)$. Jeśli $Q \neq \Lambda$, to przyjmij $\text{REF}(P) \leftarrow \Lambda$ i $P \leftarrow Q$; powtórz krok Z7. ■

Takie wykorzystanie pola REF uniemożliwia oczywiście posługiwanie się algorytmem odśmiecania opartym na algorytmie D. Algorytmu D nie można użyć również dlatego, że Listy podczas kopiowania nie są poprawnie zbudowane.

Zaproponowano kilka eleganckich algorytmów kopiowania i przenoszenia List, opierających się na istotnie słabszych założeniach na temat postaci List. Zobacz D. W. Clark, *CACM 19* (1976), 352–354; J. M. Robson, *CACM 20* (1977), 431–433.

11. Oto metoda typu „ołówek i kartka”, której bardziej formalne sformułowanie stanowi rozwiążanie postawionego zadania. Po pierwsze, każdej Liście z zadanego zbioru nadajemy niepowtarzalną nazwę (na przykład wielką literę). Na przykład $A = (a: C, b, a: F)$, $F = (b: D)$, $B = (a: F, b, a: E)$, $C = (b: G)$, $G = (a: C)$, $D = (a: F)$, $E = (b: G)$. Teraz sporządzamy zestawienie par List, których równość należy wykazać. Do tego zbioru dodajemy kolejne pary do chwili znalezienia sprzeczności spowodowanej znalezieniem pary List, które nie pasują do siebie na pierwszym poziomie (to znaczy, że „duże” Listy nie są równe), albo do takiej chwili, kiedy nasza procedura nie powoduje dopisywania żadnych nowych par (to znaczy, że „duże” Listy są równe). W tym przykładzie zestawienie na początku składa się z jednej pary AB . Następnie dostawiamy pary CF , EF (wynikające z zestawienia A i B), DG (z CF). Z otrzymanego zbioru nie wynikają już żadne nowe pary.

By udowodnić poprawność tej metody, zauważmy, że (i) jeśli dostajemy odpowiedź „nierówne”, to dane Listy nie są równe; (ii) jeśli dane Listy nie są równe, to dostajemy odpowiedź „nierówne”; (iii) dopisywania par nie można kontynuować w nieskończoność.

12. Gdy lista **AVAIL** zawiera N , gdzie N jest pewną stałą dobraną według poniższych uwag, aktywujemy współprogram, który dzieli czas procesora z programem głównym i wykonuje następujący algorytm: (a) Oznacza wszystkie N elementów listy **AVAIL**; (b) oznacza wszystkie pozostałe elementy, osiągalne poprzez zmienne programu; (c) łączy wszystkie nieoznaczone elementy w nową listę **AVAIL**, która zostanie użyta w momencie wyczerpania bieżącej listy; (d) usuwa znaczniki wszystkich elementów. Należy tak dobrąć N oraz proporcje podziału czasu między współprogram a program główny, by można było zagwarantować, że operacje (a), (b), (c) i (d) zdążą się wykonać przed wyczerpaniem bieżącej listy **AVAIL**, a jednocześnie by nie spowolnić zbytnio programu głównego. Trzeba wykazać się ostrożnością przy wyznaczaniu elementów „osiagalnych”, bo operujemy na działającym programie; w niniejszym opisie pomijamy szczegóły. Jeśli

lista utworzona w (c) ma mniej niż N elementów, to może się okazać, że będzie trzeba zatrzymać program z powodu braku pamięci. [Więcej informacji można znaleźć w: Guy L. Steele Jr., *CACM* **18** (1975), 495–508; P. Wadler, *CACM* **19** (1976), 491–500; E. W. Dijkstra, L. Lamport, A. J. Martin, C. S. Scholten, E. F. M. Steffens, *CACM* **21** (1978), 966–975; H. G. Baker, Jr., *CACM* **21** (1978), 280–294].

2.4

1. Preorder.
2. Proporcjonalny do liczby pozycji utworzonych w Tablicy Danych.
3. Zmieniamy krok A5 na:

A5'. [Usuwanie ostatniego poziomu] Zdejmij wierzchołek stosu; jeśli numer poziomu, który pokazał się w nowym wierzchołku stosu jest $\geq L$, to niech (L_1, P_1) będzie nowym wierzchołkiem stosu; powtórz ten krok. W przeciwnym razie przyjmij $SIB(P_1) \leftarrow Q$ i niech (L_1, P_1) będzie elementem, który stał się nowym wierzchołkiem stosu.

4. (Autor rozwiązania: David S. Wise) Reguła (c) jest naruszona wtedy i tylko wtedy, gdy istnieje pozycja danych, dla której pełne uściślenie $A_0 \text{ OF } \dots \text{ OF } A_n$ jest jednocześnie odniesieniem do innej pozycji danych. Z uwagi na fakt, że rodzic $A_1 \text{ OF } \dots \text{ OF } A_n$ też musi spełniać regułę (c), możemy założyć, że ta inna pozycja danych jest potomkiem tego samego rodzica. Rozszerzamy zatem algorytm A w ten sposób, by przy dodawaniu nowej pozycji danych do Tablicy Danych sprawdzał, czy jej rodzic jest przodkiem innej pozycji danych o tej samej nazwie oraz czy rodzic jakiejś innej pozycji danych o tej samej nazwie znajduje się na stosie. (Zakładamy, że rodzic Λ jest przodkiem wszystkich pozycji danych i zawsze znajduje się na stosie).

Z drugiej strony, jeśli pozostawimy algorytm A bez zmian, to kompilator zasygnalizuje błąd, gdy algorytm B natknie się na wystąpienie błędного odwołania. Jedynie konstrukcja MOVE CORRESPONDING zostanie skompilowana bez zauważenia błędu.

5. Należy wprowadzić następujące zmiany:

Krok	jest	winno być
B1.	$P \leftarrow \text{LINK}(P_0)$	$P \leftarrow \text{LINK}(\text{INFO}(T))$
B2.	$k \leftarrow 0$	$K \leftarrow T$
B3.	$k < n$	$\text{RLINK}(K) \neq \Lambda$
B4.	$k \leftarrow k + 1$	$K \leftarrow \text{RLINK}(K)$
B6.	$\text{NAME}(S) = P_k$	$\text{NAME}(S) = \text{INFO}(K)$

6. Na skutek prostej modyfikacji algorytm B znajduje jedynie pełne odwołania (jeśli $k = n$ i $\text{PARENT}(S) \neq \Lambda$ w kroku B3 i jeśli $\text{NAME}(S) \neq P_k$ w kroku B6, to przypisujemy $P \leftarrow \text{PREV}(P)$ i przechodzimy do B2). Pomysł polega na tym, by wykonać najpierw zmodyfikowany algorytm B i jeśli Q wciąż równa się Λ , wykonać jego starą wersję.

7. MOVE MONTH OF DATE OF SALES TO MONTH OF DATE OF PURCHASES. MOVE DAY OF DATE OF SALES TO DAY OF DATE OF PURCHASES. MOVE YEAR OF DATE OF SALES TO YEAR OF DATE OF PURCHASES. MOVE ITEM OF TRANSACTION OF SALES TO ITEM OF TRANSACTION OF PURCHASES. MOVE QUANTITY OF TRANSACTION OF SALES TO QUANTITY OF TRANSACTION OF PURCHASES. MOVE PRICE OF TRANSACTION OF SALES TO PRICE OF TRANSACTION OF PURCHASES. MOVE TAX OF TRANSACTION OF SALES TO TAX OF TRANSACTION OF PURCHASES.

8. Wtedy i tylko wtedy, gdy α lub β są pozycjami elementarnymi. (Czytelnika może zainteresować fakt, że pierwsza wersja algorytmu C zawierała błąd polegający na niepoprawnej obsłudze tego przypadku, który na dodatek komplikował opis algorytmu).

9. Instrukcja „MOVE CORRESPONDING α TO β ”, gdy ani α , ani β nie jest pozycją elementarną, jest równoważna zbiorowi instrukcji „MOVE CORRESPONDING A OF α TO A OF β ” branych po wszystkich nazwach A wspólnych dla grup α i β . (To jest bardziej elegancki sposób przedstawienia definicji „MOVE CORRESPONDING” niż tradycyjne sformułowanie podane w tekście). Możemy przekonać się, że algorytm C spełnia tę definicję, posługując się dowodem indukcyjnym faktu, że wykonanie kroków C2–C5 zakończy się z wartościami $P = P_0$ i $Q = Q_0$. Szczegóły dowodu wyglądają niemal identycznie, jak w wielu przeprowadzonych przez nas dowodach indukcyjnych dla drzew (zobacz na przykład dowód poprawności algorytmu 2.3.1T).

10. (a) Przyjmij $S_1 \leftarrow \text{LINK}(P_k)$, po czym zero lub więcej razy wykonuj $S_1 \leftarrow \text{PREV}(S_1)$, aż $S_1 = \Lambda$ ($\text{NAME}(S) \neq P_k$) lub $S_1 = S$ ($\text{NAME}(S) = P_k$). (b) Przyjmij $P_1 \leftarrow P$, po czym zero lub więcej razy wykonaj $P_1 \leftarrow \text{PREV}(P_1)$, aż $\text{PREV}(P_1) = \Lambda$; wykonaj analogiczną operację na zmiennych Q_1 i Q ; następnie sprawdź, czy $P_1 = Q_1$. Jeśli pozycje Tablicy Danych są uporządkowane, tak że $\text{PREV}(P) < P$ dla wszystkich P , to można posłużyć się szybszą metodą polegającą na sprawdzeniu warunku $P > Q$ i posuwaniu się po dowiązaniach PREV prowadzących od większego spośród elementów (P, Q) do najmniejszego spośród (P, Q).

11. Można osiągnąć nieznaczne przyspieszenie kroku C4 przez wprowadzenie nowego dowiązania $\text{SIB}_1(P) \equiv \text{CHILD}(\text{PARENT}(P))$. Większe znaczenie może mieć modyfikacja dowiązań CHILD i SIB, bo $\text{NAME}(\text{SIB}(P)) > \text{NAME}(P)$; to przyspieszyłoby wyszukiwanie w kroku C3, ponieważ do znalezienia pasujących numerów wystarczyłoby jedno przejście po każdej rodzinie. W ten sposób usunęlibyśmy jedyne „wyszukiwanie” w algorytmie C. Powyższe usprawnienie łatwo wprowadzić do algorytmów A i C – pozostawiamy to jako interesujące ćwiczenie dla Czytelnika. (Jeśli jednak weźmiemy pod uwagę fakt, że konstrukcja MOVE CORRESPONDING występuje w programach rzadko, a rozmiary rodzin grup nie są zwykle bardzo duże, to może się okazać, że przyspieszenie procesu komplikacji nie będzie bardzo znaczące).

12. Pozostawiamy kroki B1, B2, B3 bez zmian; pozostałe kroki zmieniamy następująco:

B4. Przyjmij $k \leftarrow k + 1$, $R \leftarrow \text{LINK}(P_k)$.

B5. Jeśli $R = \Lambda$, to nie da się dopasować; przyjmij $P \leftarrow \text{PREV}(P)$ i przejdź do B2. Jeśli $R < S \leq \text{SCOPE}(R)$, to przyjmij $S \leftarrow R$ i przejdź do B3. W przeciwnym razie $R \leftarrow \text{PREV}(R)$ i powtórz krok B5. ■

Tego algorytmu *nie* da się przystosować do konwencji języka PL/I z ćwiczenia 6.

13. Można skorzystać z tego samego algorytmu. Usuwamy operacje dotyczące NAME, PARENT, CHILD i SIB. Zdejmując wierzchołek stosu w kroku A5, wykonujemy równocześnie $\text{SCOPE}(P_1) \leftarrow Q - 1$. Gdy w kroku A2 skończą się dane wejściowe, wykonujemy $L \leftarrow 0$ i kontynuujemy, kończąc wykonanie algorytmu, jeśli w kroku A7 mamy $L = 0$.

14. W poniższym algorytmie korzysta się z pomocniczego stosu. Numeracja kroków odzwierciedla związek z akcjami starego algorytmu C.

C1. Przyjmij $P \leftarrow P_0$, $Q \leftarrow Q_0$ i zainicjuj pusty stos.

C2. Jeśli $\text{SCOPE}(P) = P$ lub $\text{SCOPE}(Q) = Q$, to wyprowadź (P, Q) jako jedną z poszukiwanych par i przejdź do C5. W przeciwnym razie włoż (P, Q) na stos i wykonaj $P \leftarrow P + 1$, $Q \leftarrow Q + 1$.

C3. Sprawdź, czy P i Q wskazują na pozycje o tych samych nazwach (zobacz ćwiczenie 10(b)). Jeśli tak, to przejdź do C2. Jeśli nie, to niech (P_1, Q_1)

będzie elementem ze szczytu stosu; jeśli $\text{SCOPE}(Q) < \text{SCOPE}(Q_1)$, to przyjmij $Q \leftarrow \text{SCOPE}(Q) + 1$ i powtóż krok C3.

C4. Niech (P_1, Q_1) będzie elementem ze szczytu stosu. Jeśli $\text{SCOPE}(P) < \text{SCOPE}(P_1)$, to przyjmij $P \leftarrow \text{SCOPE}(P) + 1$, $Q \leftarrow Q_1 + 1$ i wróć do C3. Jeśli $\text{SCOPE}(P) = \text{SCOPE}(P_1)$, to $P \leftarrow P_1$, $Q \leftarrow Q_1$ i zdejmij wierzchołek stosu.

C5. Jeśli stos jest pusty, to zakończ wykonanie algorytmu. W przeciwnym razie idź do C4. ■

2.5

1. W tak sprzyjających okolicznościach możemy potraktować stertę jak stos: niech na stertę składają się lokacje od 0 do $M - 1$ i niech **AVAIL** wskazuje pierwszą wolną lokację. Przydział: zgłaszamy błąd, jeśli $\text{AVAIL} + N \geq M$; w przeciwnym razie $\text{AVAIL} \leftarrow \text{AVAIL} + N$. Zwalnianie (tego samego bloku o rozmiarze N) to po prostu wykonanie $\text{AVAIL} \leftarrow \text{AVAIL} - N$.

Jeśli żądania pojawiają się zgodnie ze schematem pierwszy-wchodzi-pierwszy-wychodzi, to analogicznie stertę możemy potraktować jak kolejkę.

2. Rozmiar pamięci potrzebnej do przechowania informacji zajmującej l słów równa się $k \lceil l/(k-b) \rceil$, a wartość średnia tej wielkości to $kL/(k-b) + (1-\alpha)k$, gdzie zakładamy, że α równa się $1/2$ niezależnie od k . To wyrażenie opisuje minimum (dla rzeczywistych wartości k), gdy $k = b + \sqrt{2bL}$. Bierzemy zatem całkowite k nieco większe lub mniejsze od powyższej wartości, wybierając to, które daje mniejszą wartość $kL/(k-b) + \frac{1}{2}k$. Jeśli na przykład $b = 1$ i $L = 10$, to możemy wziąć $k \approx 1 + \sqrt{20} = 5$ lub 6; obie możliwości są jednakowo dobre. Więcej szczegółów na ten temat można znaleźć w **JACM 12** (1965), 53–70.

4. $rI1 \equiv Q$, $rI2 \equiv P$.

A1	LDA N	$rA \leftarrow N$.
	ENT2 AVAIL	$P \leftarrow \text{LOC(AVAIL)}$.
A2A	ENT1 0,2	$Q \leftarrow P$.
A2	LD2 0,1(LINK)	$P \leftarrow \text{LINK}(Q)$.
	J2N OVERFLOW	Jeśli $P = \Lambda$, to nie ma miejsca.
A3	CMPA 0,2(SIZE)	
	JG A2A	Skocz, jeśli $N > \text{SIZE}(P)$.
A4	SUB 0,2(SIZE)	$rA \leftarrow N - \text{SIZE}(P) \equiv K$.
	JANZ *+3	Skocz, jeśli $K \neq 0$.
	LDX 0,2(LINK)	
	STX 0,1(LINK)	$\text{LINK}(Q) \leftarrow \text{LINK}(P)$.
	STA 0,2(SIZE)	$\text{SIZE}(P) \leftarrow K$.
	LD1 0,2(SIZE)	Na zakończenie możemy
	INC1 0,2	przypisać $rI1 \leftarrow P + K$. ■

5. Zapewne nie. Zajęty blok bezpośrednio poprzedzający lokację P w końcu zostanie zwolniony, a jego długość zostanie zwiększoną o K ; przyrost o 99 jest nie do pogardzenia.

6. Pomysł polega na tym, by za każdym razem rozpoczynać wyszukiwanie bloku w innym miejscu listy **AVAIL**. Możemy posłużyć się „wskaźnikiem wodzącym”. Zmieniąc wskaźnikową **ROVER** będziemy traktowali następująco: w kroku A1 wykonujemy $Q \leftarrow \text{ROVER}$. Po kroku A4 wykonujemy $\text{ROVER} \leftarrow \text{LINK}(Q)$, gdy $\text{LINK}(Q) \neq \Lambda$, a w przeciwnym razie przypisanie $\text{ROVER} \leftarrow \text{LOC(AVAIL)}$. Jeśli w kroku A2 $P = \Lambda$ po raz pierwszy podczas danego wykonania algorytmu A, to wykonujemy $Q \leftarrow \text{LOC(AVAIL)}$

i powtarzamy krok A2. Jeśli $P = \Lambda$ po raz drugi, to (pomyślnie) kończymy wykonanie algorytmu. Dzięki powyższym zabiegom zmienna ROVER wskazuje w mniej więcej przypadkowe miejsce listy AVAIL, zatem rozmiary bloków będą rozłożone bardziej równomiernie. Na początku wykonania programu przypisujemy $\text{ROVER} \leftarrow \text{LOC}(\text{AVAIL})$; musimy ponadto przypisać do ROVER wartość $\text{LOC}(\text{AVAIL})$ we wszystkich tych sytuacjach, gdy blok o adresie równym bieżącej zawartości zmiennej ROVER jest usuwany z listy AVAIL. (Czasami warto jednak mieć małe bloki na początku, tak jak w ortodoksyjnej metodzie pierwszego dopasowania; na przykład jeśli chcielibyśmy przechowywać stos tablicowy w pamięci o wysokich adresach. W takich przypadkach możemy zmniejszyć czas wyszukiwania, stosując rozwiązań wykorzystujące drzewa, sugerowane w ćwiczeniu 6.2.3–30).

7. 2000, 1000; żądania 800, 1300.

[Przykład, w którym metoda najgorszego dopasowania działa, a metoda najlepszego dopasowania nie, podał R. J. Weiland].

8. W kroku A1 wykonujemy dodatkowo $M \leftarrow \infty$, $R \leftarrow \Lambda$. W kroku A2, jeśli $P = \Lambda$, to przechodzimy do A6. W kroku A3 zmieniamy skok do A4 na skok do A5. Dodajemy następujące kroki:

A5. [Czy lepsze dopasowanie?] Jeśli $M > \text{SIZE}(P)$, to przyjmij $R \leftarrow Q$, $M \leftarrow \text{SIZE}(P)$. Następnie $Q \leftarrow P$ i powróć do A2.

A6. [Czy znaleziono coś?] Jeśli $R = \Lambda$, to zakończ wykonanie algorytmu, sygnalizując błąd. W przeciwnym razie przyjmij $Q \leftarrow R$, $P \leftarrow \text{LINK}(Q)$ i przejdź do A4. ■

9. Jeśli mamy szczęście i znajdziemy blok, w którym $\text{SIZE}(P) = N$, to lepszego dopasowania znaleźć się nie da, nie trzeba więc dalej szukać. (Jeśli różnych rozmiarów bloków jest mało, to zdarza się to dosyć często). Jeśli korzystamy z metody znaczników brzegowych, jak w algorytmie C, to bloki na liście AVAIL można utrzymywać w porządku ich rozmiarów. W ten sposób średni czas przeszukiwania listy możemy zmniejszyć o połowę. Jeśli jednak spodziewamy się wielu bloków na liście AVAIL, to lepiej przechowywać je za pomocą drzewa zrównoważonego (zobacz punkt 6.2.3).

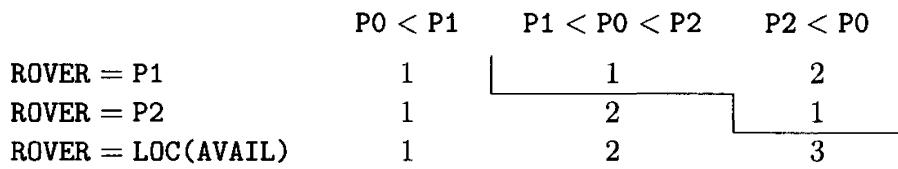
10. Należy wprowadzić następujące zmiany:

Krok B2: zamiast „ $P > P_0$ ” przyjąć „ $P \geq P_0$ ”.

Krok B3: wstawić „Jeśli $P_0 + N > P$ i $P \neq \Lambda$, to przyjmij $P \leftarrow \text{LINK}(P)$ i powtórz krok B3”.

Krok B4: zamiast „ $Q + \text{SIZE}(Q) = P_0$ ” przyjąć „ $Q + \text{SIZE}(Q) \geq P_0$ ”; zamiast „ $\text{SIZE}(Q) \leftarrow \text{SIZE}(Q) + N$ ” przyjąć „ $\text{SIZE}(Q) \leftarrow P_0 + N - Q$ ”.

11. Jeśli wartość P_0 jest większa niż wartość ROVER, to w kroku B1 możemy wykonać $Q \leftarrow \text{ROVER}$ zamiast $Q \leftarrow \text{LOC}(\text{AVAIL})$. Jeśli długość listy AVAIL wynosi n , to średnia liczba iteracji w kroku B2 równa się $(2n+3)(n+2)/6(n+1) = \frac{1}{3}n + \frac{5}{6} + O\left(\frac{1}{n}\right)$. Jeśli na przykład $n = 2$, to otrzymujemy 9 jednakowo prawdopodobnych sytuacji, gdzie P_1 i P_2 wskazują na dwa wolne bloki:



Dla każdego przypadku diagram pokazuje liczbę potrzebnych iteracji. Średnia wynosi

$$\frac{1}{9} \left(\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2} \right) = \frac{1}{9} \left(\binom{5}{3} + \binom{4}{3} \right) = \frac{14}{9}.$$

12. A1. Przyjmij $P \leftarrow ROVER$, $F \leftarrow 0$.

A2. Jeśli $P = LOC(AVAIL)$ i $F = 0$, to przyjmij $P \leftarrow AVAIL$, $F \leftarrow 1$ i powtórz kroki A2. Jeśli $P = LOC(AVAIL)$ i $F \neq 0$, to zakończ wykonanie algorytmu, sygnalizując błąd.

A3. Jeśli $SIZE(P) \geq N$, to przejdź do A4; w przeciwnym razie przyjmij $P \leftarrow LINK(P)$ i wróć do A2.

A4. Przyjmij $ROVER \leftarrow LINK(P)$, $K \leftarrow SIZE(P) - N$. Jeśli $K < c$ (gdzie c jest stałą ≥ 2), to przyjmij $LINK(LINK(P+1)) \leftarrow ROVER$, $LINK(ROVER+1) \leftarrow LINK(P+1)$, $L \leftarrow P$. W przeciwnym razie przyjmij $L \leftarrow P + K$, $SIZE(P) \leftarrow SIZE(L-1) \leftarrow K$, $TAG(L-1) \leftarrow "-"$, $SIZE(L) \leftarrow N$. Na koniec przypisz $TAG(L) \leftarrow TAG(L + SIZE(L) - 1) \leftarrow "+"$. ■

13. $rI1 \equiv P$, $rX \equiv F$, $rI2 \equiv L$.

LINK	EQU	4:5	
SIZE	EQU	1:2	
TSIZE	EQU	0:2	
TAG	EQU	0:0	
A1	LDA	N	$rA \leftarrow N$.
	SLA	3	Przesuń do pola SIZE.
	ENTX	0	$F \leftarrow 0$.
	LD1	ROVER	$P \leftarrow ROVER$.
	JMP	A2	
A3	CMPA	0,1(SIZE)	
	JLE	A4	Skocz, jeśli $N \leq SIZE(P)$.
	LD1	0,1(LINK)	$P \leftarrow LINK(P)$.
A2	ENT2	-AVAIL,1	$rI2 \leftarrow P - LOC(AVAIL)$.
	J2NZ	A3	
	JXNZ	OVERFLOW	Czy $F \neq 0$?
	ENTX	1	$F \leftarrow 1$.
	LD1	AVAIL(LINK)	$P \leftarrow AVAIL$.
	JMP	A2	
A4	LD2	0,1(LINK)	
	ST2	ROVER	$ROVER \leftarrow LINK(P)$.
	LDA	0,1(SIZE)	$rA \equiv K \leftarrow SIZE(P) - N$.
	SUB	N	
	CMPA	=c=	
	JGE	1F	Skocz, jeśli $K \geq c$.
	LD3	1,1(LINK)	$rI3 \leftarrow LINK(P+1)$.
	ST2	0,3(LINK)	$LINK(rI3) \leftarrow ROVER$.
	ST3	1,2(LINK)	$LINK(ROVER+1) \leftarrow rI3$.
	ENT2	0,1	$L \leftarrow P$.
	LD3	0,1(SIZE)	$rI3 \leftarrow SIZE(P)$.
	JMP	2F	
1H	STA	0,1(SIZE)	$SIZE(P) \leftarrow K$.
	LD2	0,1(SIZE)	
	INC2	0,1	$L \leftarrow P + K$.

```

LDAN 0,1(SIZE)    rA ← -K.
STA  -1,2(TSIZE)   SIZE(L - 1) ← K, TAG(L - 1) ← „-”.
LD3   N             rI3 ← N.
2H    ST3  0,2(TSIZE)  TAG(L) ← „+”, a także SIZE(L) ← rI3.
INC3 0,2
STZ  -1,3(TAG)     TAG(L + SIZE(L) - 1) ← „+”. ■

```

- 14.** (a) To pole jest potrzebne do wyznaczenia początku bloku w kroku C2. Można je zastąpić (zapewne z korzyścią) przez dowiązanie do pierwszego słowa bloku. Zobacz też ćwiczenie 19. (b) To pole jest potrzebne, ponieważ czasami rezerwujemy więcej niż N słów (na przykład gdy K = 1), musimy więc znać liczbę zarezerwowanych słów przy zwalnianiu bloku.

- 15, 16.** rI1 ≡ P0, rI2 ≡ P1, rI3 ≡ F, rI4 ≡ B, rI6 ≡ -N.

```

D1 LD1  P0          D1.
LD2  0,1(SIZE)
ENN6 0,2           N ← SIZE(P0).
INC2 0,1           P1 ← P0 + N.
LD5  0,2(TSIZE)
J5N  D4           Skocz do D4, jeśli TAG(P1) = „-”.
D2 LD5  -1,1(TSIZE) D2.
J5N  D7           Skocz do D7, jeśli TAG(P0 - 1) = „-”.
D3 LD3  AVAIL(LINK) D3. F ← AVAIL.
ENT4 AVAIL        B ← LOC(AVAIL).
JMP  D5           Skocz do D5.
D4 INC6 0,5        D4. N ← N + SIZE(P1).
LD3  0,2(LINK)    F ← LINK(P1).
LD4  1,2(LINK)    B ← LINK(P1 + 1).
CMP2 ROVER        (Nowy kod, realizujący pomysł
                  z ćwiczenia 12:
                  Jeśli P1 = ROVER,
                  to ROVER ← LOC(AVAIL)).
ENTX AVAIL
STX  ROVER
DEC2 0,5           P1 ← P1 + SIZE(P1).
LD5  -1,1(TSIZE)
J5N  D6           Skocz do D6, jeśli TAG(P0 - 1) = „-”.
D5 ST3  0,1(LINK) D5. LINK(P0) ← F.
ST4  1,1(LINK)    LINK(P0 + 1) ← B.
ST1  1,3(LINK)    LINK(F + 1) ← P0.
ST1  0,4(LINK)    LINK(B) ← P0.
JMP  D8           Skocz do D8.
D6 ST3  0,4(LINK) D6. LINK(B) ← F.
ST4  1,3(LINK)    LINK(F + 1) ← B.
D7 INC6 0,5        D7. N ← N + SIZE(P0 - 1).
INC1 0,5           P0 ← P0 - SIZE(P0 - 1).
D8 ST6  0,1(TSIZE) D8. SIZE(P0) ← N, TAG(P0) ← „-”.
ST6  -1,2(TSIZE)   SIZE(P1 - 1) ← N, TAG(P1 - 1) ← „-”. ■

```

- 17.** Oba pola LINK powinny być równe LOC(AVAIL).

- 18.** Algorytm A rezerwuje górny fragment większego bloku. Kiedy cała pamięć jest wolna, metoda pierwszego dopasowania zaczyna od rezerwowania górnego adresów, ale gdy się zwolnią, nie są powtórnie rezerwowane, ponieważ zazwyczaj wyszukiwanie

wolnego bloku kończy się wcześniej w niższych adresach. Stąd dla metody pierwszego dopasowania duże bloki na początku pamięci szybko znikają. Duży blok rzadko kiedy jest jednak najlepszym dopasowaniem, zatem w metodzie najlepszego dopasowania duże bloki na początku pamięci są omijane.

19. Należy skorzystać z algorytmu z ćwiczenia 12, po usunięciu z niego odwołań do $\text{SIZE}(L - 1)$, $\text{TAG}(L - 1)$ i $\text{TAG}(L + \text{SIZE}(L) - 1)$ w kroku A4. Ponadto trzeba wstawić nowy krok pomiędzy kroki A2 i A3:

A2 $\frac{1}{2}$. Przyjmij $P1 \leftarrow P + \text{SIZE}(P)$. Jeśli $\text{TAG}(P1) = „+”$, to przejdź do kroku A3.

W przeciwnym razie przyjmij $P2 \leftarrow \text{LINK}(P1)$, $\text{LINK}(P2+1) \leftarrow \text{LINK}(P1+1)$, $\text{LINK}(\text{LINK}(P1+1)) \leftarrow P2$, $\text{SIZE}(P) \leftarrow \text{SIZE}(P) + \text{SIZE}(P1)$. Jeśli $\text{ROVER} = P1$, to przyjmij $\text{ROVER} \leftarrow P2$. Powtórz krok A2 $\frac{1}{2}$.

Jest jasne, że sytuacja (2)–(4) się tu nie pojawi. Jedyny istotny efekt polega na tym, że wyszukiwanie będzie nieco dłuższe niż w ćwiczeniu 12, a czasami K będzie mniejsze niż c, chociaż tak naprawdę przed bieżącym blokiem jest blok wolny, tylko o nim nie wiemy.

(Inne rozwiązanie polega na wyjęciu scalania poza pętlę w kroku A3 i wykonywaniu scalania tylko w kroku A4 przed ostatecznym przydziałem lub w pętli wewnętrznej na zasadzie ostatniej deski ratunku, gdyby algorytm miał zakończyć się niepomyślnie. Te pomysły wymagają zbadania symulacyjnego, bo tak na prawdę nie wiadomo, czy poprawią, czy popsują algorytm).

[Powyższa metoda z paroma poprawkami okazała się całkiem zadowalająca w realizacjach systemów *TeX* i *METAFONT*. Zobacz *TeX: The Program* (Addison-Wesley, 1986), §125].

20. Gdy w pętli scalającej okazuje się, że blok bliźniaczy jest wolny, chcemy usunąć go z listy $\text{AVAIL}[k]$, ale nie wiemy, które dowiązania należy zmodyfikować; dowiedzieć możemy się (i) przeszukując listę (co może długo trwać) albo (ii) korzystając z dowiązań dwukierunkowych.

21. Dla $n = 2^k\alpha$, gdzie $1 \leq \alpha \leq 2$, a_n jest postaci $2^{2k+1}(\alpha - \frac{2}{3}) + \frac{1}{3}$, a b_n jest postaci $2^{2k-1}\alpha^2 + 2^{k-1}\alpha$. Iloraz a_n/b_n dla dużych n to $4(\alpha - \frac{2}{3})/\alpha^2$. To wyrażenie przyjmuje wartość minimalną $\frac{4}{3}$ dla $\alpha = 1$ i 2 , a wartość maksymalną $\frac{3}{2}$ dla $\alpha = 1\frac{1}{3}$. Zatem a_n/b_n nie ma granicy – oscyluje między tymi dwoma ekstremami. Jednakże metody uśredniania zaprezentowane w rozdziale 4.2.4 umożliwiają wyznaczenie wartości średniej ilorazu: $4(\ln 2)^{-1} \int_1^2 (\alpha - \frac{2}{3}) d\alpha / \alpha^3 = (\ln 2)^{-1} \approx 1.44$.

22. Ten pomysł wymaga obecności znacznika **TAG** nie tylko w pierwszym słowie bloku o rozmiarze 11. Pomysł nie jest zły, jeżeli mamy gdzie przechować te dodatkowe bity i na pewno nadaje się do wykorzystania w projektowaniu układów cyfrowych.

23. 011011110100; 011011100000.

24. Spowodowałoby to wprowadzenie błędu do programu. W kroku S1 możemy się znaleźć, gdy $\text{TAG}(0) = 1$, ponieważ do S1 skaczemy z S2. By program działał, należy dodać „ $\text{TAG}(L) \leftarrow 0$ ” po „ $L \leftarrow P$ ” w kroku S2. (Łatwiej przyjąć, że $\text{TAG}(2^m) = 0$).

25. Pogląd jest całkowicie słuszny. Można zrezygnować z atrap $\text{AVAIL}[k]$ dla $n < k \leq m$. Algorytmy podane w tekście będą działać poprawnie, gdy zamienimy „ m ” na „ n ” w krokach R1 i S1. Warunki początkowe (13) i (14) należy tak zmienić, by reprezentowały sytuację, w której jest 2^{m-n} bloków rozmiaru 2^n , a nie jeden blok rozmiaru 2^m .

26. Posługując się zapisem dwójkowym M, możemy wyznaczyć warunki początkowe (13) i (14), dzieląc lokacje na bloki o rozmiarach będących potęgami dwójkii, tak by

bloki coraz mniejsze miały coraz wyższe adresy. W algorytmie S należy przyjąć, że znacznik $\text{TAG}(P)$ jest równy 0 dla $P \geq M - 2^k$.

27. $rI1 \equiv k$, $rI2 \equiv j$, $rI3 \equiv j - k$, $rI4 \equiv L$, $\text{LOC}(\text{AVAIL}[j]) = \text{AVAIL} + j$; zakładamy, że istnieje pomocnicza tablica $\text{TWO}[j] = 2^j$ przechowywana w lokacjach $\text{TWO} + j$, dla $0 \leq j \leq m$. Zakładamy ponadto, że „+” i „-” reprezentują znaczniki równe 0 i 1 oraz $\text{TAG}(\text{LOC}(\text{AVAIL}[j])) = \text{“-”}$; ale $\text{TAG}(\text{LOC}(\text{AVAIL}[m + 1])) = \text{“+”}$ jest wartownikiem.

00	KVAL	EQU	5:5		
01	TAG	EQU	0:0		
02	LINKF	EQU	1:2		
03	LINKB	EQU	3:4		
04	TLNKF	EQU	0:2		
05	R1	LD1	K	1	<u>R1. Znajdź blok.</u>
06		ENT2	0,1	1	$j \leftarrow k$.
07		ENT3	0	1	
08		LD4	AVAIL,2(LINKF)	1	
09	1H	ENT5	AVAIL,2	1 + R	
10		DEC5	0,4	1 + R	
11		J5NZ	R2	1 + R	Skocz, jeśli $\text{AVAILF}[j] \neq \text{LOC}(\text{AVAIL}[j])$.
12		INC2	1	R	Zwiększ j .
13		INC3	1	R	
14		LD4N	AVAIL,2(TLNKF)	R	
15		J4NN	1B	R	Czy $j \leq m$?
16		JMP	OVERFLOW		
17	R2	LD5	0,4(LINKF)	1	<u>R2. Usuń z listy.</u>
18		ST5	AVAIL,2(LINKF)	1	$\text{AVAILF}[j] \leftarrow \text{LINKF}(L)$.
19		ENTA	AVAIL,2	1	
20		STA	0,5(LINKB)	1	$\text{LINKB}(L) \leftarrow \text{LOC}(\text{AVAIL}[j])$.
21		STZ	0,4(TAG)	1	$\text{TAG}(L) \leftarrow 0$.
22	R3	J3Z	DONE	1	<u>R3. Czy trzeba podzielić?</u>
23	R4	DEC3	1	R	<u>R4. Podziel.</u>
24		DEC2	1	R	Zmniejsz j .
25		LD5	TWO,2	R	$rI5 \equiv P$.
26		INC5	0,4	R	$P \leftarrow L + 2^j$.
27		ENNA	AVAIL,2	R	
28		STA	0,5(TLNKF)	R	$\text{TAG}(P) \leftarrow 1$, $\text{LINKF}(P) \leftarrow \text{LOC}(\text{AVAIL}[j])$.
29		STA	0,5(LINKB)	R	$\text{LINKB}(P) \leftarrow \text{LOC}(\text{AVAIL}[j])$.
30		ST5	AVAIL,2(LINKF)	R	$\text{AVAILF}[j] \leftarrow P$.
31		ST5	AVAIL,2(LINKB)	R	$\text{AVAILB}[j] \leftarrow P$.
32		ST2	0,5(KVAL)	R	$\text{KVAL}(P) \leftarrow j$.
33		J3P	R4	R	Przejdz do R3.
34	DONE	...			■

28. $rI1 \equiv k$, $rI5 \equiv P$, $rI4 \equiv L$; zakładamy, że $\text{TAG}(2^m) = \text{“+”}$.

01	S1	LD4	L	1	<u>S1. Czy blok bliźniaczy jest wolny?</u>
02		LD1	K	1	
03	1H	ENTA	0,4	1 + S	
04		XOR	TWO,1	1 + S	$rA \leftarrow \text{buddy}_k(L)$.
05		STA	TEMP	1 + S	
06		LD5	TEMP	1 + S	$P \leftarrow rA$.
07		LDA	0,5	1 + S	

08	JANN S3	$1 + S$	Skocz, jeśli $\text{TAG}(P) = 0$.
09	CMP1 0,5(KVAL)	$B + S$	
10	JNE S3	$B + S$	Skocz, jeśli $\text{KVAL}(P) \neq k$.
11	S2 LD2 0,5(LINKF)	S	<u>S2. Łączanie z blokiem bliźniaczym.</u>
12	LD3 0,5(LINKB)	S	
13	ST3 0,2(LINKF)	S	$\text{LINKF}(\text{LINKB}(P)) \leftarrow \text{LINKF}(P)$.
14	ST2 0,3(LINKB)	S	$\text{LINKB}(\text{LINKF}(P)) \leftarrow \text{LINKB}(P)$.
15	INC1 1	S	Zwiększ k .
16	CMP4 TEMP	S	
17	JL 1B	S	
18	ENT4 0,5	A	Jeśli $L > P$, to $L \leftarrow P$.
19	JMP 1B	A	
20	S3 LD2 AVAIL,1(LINKF)	1	<u>S3. Wstawianie na listę.</u>
21	ENNA AVAIL,1	1	
22	STA 0,4(0:4)	1	$\text{TAG}(L) \leftarrow 1$, $\text{LINKB}(L) \leftarrow \text{LOC}(\text{AVAIL}[k])$.
23	ST2 0,4(LINKF)	1	$\text{LINKF}(L) \leftarrow \text{AVAILF}[k]$.
24	ST1 0,4(KVAL)	1	$\text{KVAL}(L) \leftarrow k$.
25	ST4 0,2(LINKB)	1	$\text{LINKB}(\text{AVAILF}[k]) \leftarrow L$.
26	ST4 AVAIL,1(LINKF)	1	$\text{AVAIL}[k] \leftarrow L$. ■

29. Tak, ale musimy wówczas przeprowadzać wyszukiwania lub (lepiej) utrzymywać pomocniczą tablicę, w której upakowane są znaczniki TAG. (Kusząca wydaje się propozycja, by nie scalać bloków bliźniaczych w algorytmie S, tylko w algorytmie R, gdy nie ma wystarczająco dużego bloku, by zrealizować żądanie; to zapewne prowadzi jednak do dużej fragmentacji pamięci).

31. Zobacz David L. Russell, *SICOMP* 6 (1977), 607–621.

33. G1. [Inicjowanie pól LINK] Wykonaj $P \leftarrow 1$ i powtarzaj operację $\text{LINK}(P) \leftarrow \Lambda$, $P \leftarrow P + \text{SIZE}(P)$, aż $P = \text{AVAIL}$. (Po prostu przypisujemy Λ do pól LINK w pierwszych słowach wszystkich węzłów; można założyć, że ten krok jest niepotrzebny, bo pole LINK będzie równe Λ albo w wyniku przypisania w kroku G9, albo w wyniku zainicjowania przez algorytm przydzielający pamięć).

G2. [Inicjowanie fazy oznaczania] Przyjmij $\text{TOP} \leftarrow \text{USE}$, $\text{LINK}(\text{TOP}) \leftarrow \text{AVAIL}$, $\text{LINK}(\text{AVAIL}) \leftarrow \Lambda$. (TOP wskazuje na wierzchołek stosu, jak w algorytmie 2.3.5D).

G3. [Zdejmowanie ze stosu] Przyjmij $P \leftarrow \text{TOP}$, $\text{TOP} \leftarrow \text{LINK}(\text{TOP})$. Jeśli $\text{TOP} = \Lambda$, to przejdź do G5.

G4. [Włożenie nowych wskaźników na stos] Dla $1 \leq k \leq T(P)$ wykonaj następujące operacje: $Q \leftarrow \text{LINK}(P + k)$; potem jeśli $Q \neq \Lambda$ i $\text{LINK}(Q) = \Lambda$, to przyjmij $\text{LINK}(Q) \leftarrow \text{TOP}$, $\text{TOP} \leftarrow Q$. Następnie przejdź do G3.

G5. [Inicjowanie następnej fazy] (W tym momencie $P = \text{AVAIL}$, a faza oznaczania się zakończyła, zatem w pierwszym słowie każdego osiągalnego węzła pole LINK jest równe od Λ . Naszym kolejnym zadaniem jest połączenie przylegających bloków nieosiągalnych, dzięki czemu przyspieszymy wykonanie kolejnych kroków, a także przydzielenie nowych adresów węzłom osiągalnym). Przyjmij $Q \leftarrow 1$, $\text{LINK}(\text{AVAIL}) \leftarrow Q$, $\text{SIZE}(\text{AVAIL}) \leftarrow 0$, $P \leftarrow 1$. (Lokację AVAIL wykorzystujemy w roli wartownika oznaczającego koniec pętli w kolejnych fazach).

G6. [Przypisanie nowego adresu] Jeśli $\text{LINK}(P) = \Lambda$, to przejdź do G7. W przeciwnym razie, jeśli $\text{SIZE}(P) = 0$, to przejdź do G8. W przeciwnym razie przyjmij $\text{LINK}(P) \leftarrow Q$, $Q \leftarrow Q + \text{SIZE}(P)$, $P \leftarrow P + \text{SIZE}(P)$ i powtórz ten krok.

G7. [Scalanie wolnych bloków] Jeśli $\text{LINK}(P + \text{SIZE}(P)) = \Lambda$, to zwiększa SIZE(P) o $\text{SIZE}(P + \text{SIZE}(P))$ i powtórzy ten krok. W przeciwnym razie przyjmij $P \leftarrow P + \text{SIZE}(P)$ i wróć do kroku G6.

G8. [Modyfikowanie wskaźników] (Pole LINK w pierwszym słowie każdego osiągalnego węzła zawiera adres, pod który dany węzeł zostanie przemieszczony). Przyjmij $\text{USE} \leftarrow \text{LINK}(\text{USE})$, $\text{AVAIL} \leftarrow Q$. Następnie ustaw $P \leftarrow 1$ i powtarzaj następującą operację, aż $\text{SIZE}(P) = 0$: jeśli $\text{LINK}(P) \neq \Lambda$, to $\text{LINK}(Q) \leftarrow \text{LINK}(\text{LINK}(Q))$ dla wszystkich takich Q, że $P < Q \leq P + \text{T}(P)$ i $\text{LINK}(Q) \neq \Lambda$; następnie niezależnie od wartości $\text{LINK}(P)$, przypisz $P \leftarrow P + \text{SIZE}(P)$.

G9. [Przemieszczanie] Przyjmij $P \leftarrow 1$ i powtarzaj następującą operację, aż $\text{SIZE}(P) = 0$: przyjmij $Q \leftarrow \text{LINK}(P)$ i jeśli $Q \neq \Lambda$, to $\text{LINK}(P) \leftarrow \Lambda$ i $\text{NODE}(Q) \leftarrow \text{NODE}(P)$; następnie niezależnie od tego, czy $Q = \Lambda$, przypisz $P \leftarrow P + \text{SIZE}(P)$. (Operacja $\text{NODE}(Q) \leftarrow \text{NODE}(P)$ oznacza skopiowanie $\text{SIZE}(P)$ słów; zawsze $Q \leq P$, zatem bezpieczeństwo gwarantuje przenoszenie w kolejności od mniejszych lokacji do większych). ■

[Ta metoda jest nazywana „odśmiecaczem LISP 2”. Ciekawy jej wariant, w którym pole LINK nie jest wymagane, można oprzeć na pomyśle polegającym na powiązaniu wszystkich wskaźników wskazujących na jeden węzeł – zobacz Lars-Erik Thorelli, *BIT* **16** (1976), 426–441; R. B. K. Dewar, A. P. McCann, *Software Practice & Exp.* **7** (1977), 95–113; F. Lockwood Morris, *CACM* **21** (1978), 662–665, **22** (1979), 571; H. B. M. Jonkers, *Inf. Proc. Letters* **9** (1979), 26–30; J. J. Martin, *CACM* **25** (1982), 571–581; F. Lockwood Morris, *Inf. Proc. Letters* **15** (1982), 139–142, **16** (1983), 215. Inne metody opublikowali: B. K. Haddon, W. M. Waite, *Comp. J.* **10** (1967), 162–165; B. Wegbreit, *Comp. J.* **15** (1972), 204–208; D. A. Zave, *Inf. Proc. Letters* **3** (1975), 167–169. Cztery z tych rozwiązań przeanalizowali Cohen i Nicolau w: *ACM Trans. Prog. Languages and Systems* **5** (1983), 532–553].

34. Niech $\text{TOP} \equiv rI1$, $Q \equiv rI2$, $P \equiv rI3$, $k \equiv rI4$, $\text{SIZE}(P) \equiv rI5$. Zakładamy, że $\Lambda = 0$ i $\text{LINK}(0) \neq 0$, co uprości krok G4. Pomijamy krok G1.

```

01 LINK EQU 4:5
02 INFO EQU 0:3
03 SIZE EQU 1:2
04 T   EQU 3:3
05 G2 LD1 USE      1  G2. Inicjowanie fazy oznaczania. TOP ← USE.
06 LD2 AVAIL    1
07 ST2 0,1(LINK) 1  LINK(TOP) ← AVAIL.
08 STZ 0,2(LINK) 1  LINK(AVAIL) ← Λ.
09 G3 ENT3 0,1    a+1 G3. Zdejmowanie ze stosu. P ← TOP.
10 LD1 0,1(LINK) a+1 TOP ← LINK(TOP).
11 J1Z G5      a+1 Skocz do G5, jeśli  $\text{TOP} = \Lambda$ .
12 G4 LD4 0,3(T)  a  G4. Włożenie nowych wskaźników na stos. k ← T(P).
13 1H J4Z G3    a+b  $k = 0?$ 
14 INC3 1      b   $P \leftarrow P + 1.$ 
15 DEC4 1      b   $k \leftarrow k - 1.$ 
16 LD2 0,3(LINK) b   $Q \leftarrow \text{LINK}(P).$ 
17 LDA 0,2(LINK) b
18 JANZ 1B     b  Skocz, jeśli  $\text{LINK}(Q) \neq \Lambda$ .
19 ST1 0,2(LINK) a-1 W przeciwnym razie  $\text{LINK}(Q) \leftarrow \text{TOP}$ ,
20 ENT1 0,2     a-1  $\text{TOP} \leftarrow Q.$ 
21 JMP 1B     a-1

```

22 G5	ENT2 1	1	<u>G5. Inicjowanie następnej fazy.</u> $Q \leftarrow 1$.
23	ST2 0,3	1	LINK(AVAIL) $\leftarrow 1$, SIZE(AVAIL) $\leftarrow 0$.
24	ENT3 1	1	$P \leftarrow 1$.
25	JMP G6	1	
26 1H	ST2 0,3(LINK)	a	LINK(P) $\leftarrow Q$.
27	INC2 0,5	a	$Q \leftarrow Q + \text{SIZE}(P)$.
28	INC3 0,5	a	$P \leftarrow P + \text{SIZE}(P)$.
29 G6	LDA 0,3(LINK)	a+1	<u>G6. Przypisanie nowego adresu.</u>
30 G6A	LD5 0,3(SIZE)	a+c+1	
31	JAZ G7	a+c+1	Skocz, jeśli LINK(P) = Λ .
32	J5NZ 1B	a+1	Skocz, jeśli SIZE(P) $\neq 0$.
33 G8	LD1 USE	1	<u>G8. Modyfikowanie wskaźników.</u>
34	LDA 0,1(LINK)	1	
35	STA USE	1	USE $\leftarrow \text{LINK}(USE)$.
36	ST2 AVAIL	1	AVAIL $\leftarrow Q$.
37	ENT3 1	1	$P \leftarrow 1$.
38	JMP G8P	1	
39 1H	LD6 0,6(SIZE)	d	
40	INC5 0,6	d	$rI5 \leftarrow rI5 + \text{SIZE}(P + \text{SIZE}(P))$.
41 G7	ENT6 0,3	c+d	<u>G7. Scalanie wolnych bloków.</u>
42	INC6 0,5	c+d	$rI6 \leftarrow P + \text{SIZE}(P)$.
43	LDA 0,6(LINK)	c+d	
44	JAZ 1B	c+d	Skocz, jeśli LINK(rI6) $\equiv \Lambda$.
45	ST5 0,3(SIZE)	c	SIZE(P) $\leftarrow rI5$.
46	INC3 0,5	c	$P \leftarrow P + \text{SIZE}(P)$.
47	JMP G6A	c	
48 2H	DEC4 1	b	$k \leftarrow k - 1$.
49	INC2 1	b	$Q \leftarrow Q + 1$.
50	LD6 0,2(LINK)	b	
51	LDA 0,6(LINK)	b	
52	STA 0,2(LINK)	b	LINK(Q) $\leftarrow \text{LINK}(\text{LINK}(Q))$.
53 1H	J4NZ 2B	a+b	Skocz, jeśli $k \neq 0$.
54 3H	INC3 0,5	a+c	$P \leftarrow P + \text{SIZE}(P)$.
55 G8P	LDA 0,3(LINK)	1+a+c	
56	LD5 0,3(SIZE)	1+a+c	
57	JAZ 3B	1+a+c	Czy LINK(P) = Λ ?
58	LD4 0,3(T)	1+a	$k \leftarrow T(P)$.
59	ENT2 0,3	1+a	$Q \leftarrow P$.
60	J5NZ 1B	1+a	Skocz, chyba że SIZE(P) = 0.
61 G9	ENT3 1	1	<u>G9. Przemieszczanie.</u> $P \leftarrow 1$.
62	ENT1 1	1	Ustaw rI1 dla instrukcji MOVE.
63	JMP G9P	1	
64 1H	STZ 0,3(LINK)	a	LINK(P) $\leftarrow \Lambda$.
65	ST5 *+1(4:4)	a	
66	MOVE 0,3(*)	a	NODE(rI1) $\leftarrow \text{NODE}(P)$, rI1 $\leftarrow rI1 + \text{SIZE}(P)$.
67 3H	INC3 0,5	a+c	$P \leftarrow P + \text{SIZE}(P)$.
68 G9P	LDA 0,3(LINK)	1+a+c	
69	LD5 0,3(SIZE)	1+a+c	
70	JAZ 3B	1+a+c	Skocz, jeśli LINK(P) = Λ .
71	J5NZ 1B	1+a	Skocz, chyba że SIZE(P) = 0. ■

W wierszu 66 zakładamy, że rozmiar każdego węzła jest na tyle mały, że można go skopiować za pomocą pojedynczego rozkazu MOVE. Dla większości przypadków, w których można zastosować tego rodzaju odśmiecanie, powyższe założenie wydaje się rozsądne.

Całkowity czas wykonania tego algorytmu równa się $(44a + 17b + 2w + 25c + 8d + 47)u$, gdzie a jest liczbą węzłów osiągalnych, b jest liczbą zawartych w nich dowiązań, c jest liczbą węzłów nieosiągalnych, których *nie poprzedza* żaden węzeł nieosiągalny, d jest liczbą węzłów nieosiągalnych, które *poprzedza* pewien węzeł nieosiągalny, w jest całkowitą liczbą słów w węzłach osiągalnych. Jeśli pamiętać zawiera n węzłów, z których ρn jest nieosiągalnych, to możemy podać oszacowania $a = (1 - \rho)n$, $c = (1 - \rho)\rho n$, $d = \rho^2 n$. Przykład: węzły o średnim rozmiarze pięciu słów, średnio dwa wskaźniki w węźle, 1000 węzłów w pamięci. Jeśli $\rho = \frac{1}{5}$, to odzyskanie wolnego węzła zabiera $374u$; jeśli $\rho = \frac{1}{2}$, to $104u$; jeśli $\rho = \frac{4}{5}$, to tylko $33u$.

36. Samotnego klienta będzie można posadzić na jednym z szesnastu miejsc 1, 3, 4, 6, ..., 23. Gdy przyjdzie para, na pewno znajdzie się miejsce. Brak miejsca oznaczałby, że przynajmniej dwie osoby siedzą na miejscach (1, 2, 3), przynajmniej dwie na (4, 5, 6), ..., przynajmniej dwie na (19, 20, 21) i przynajmniej jedna na 22 lub 23, zatem siedzi już przynajmniej piętnaście osób.

37. Kelnerka usadza szesnastu samotników. Pomiędzy zajętymi miejscami jest w sumie 17 „dziur”, wliczając dziury na brzegach i uwzględniając dziury rozmiaru zero (pomiędzy dwoma kolejnymi zajętymi miejscami). Całkowita liczba wolnych miejsc, tj. suma rozmiarów wszystkich siedemnastu dziur, wynosi 6. Przypuśćmy, że x dziur ma rozmiar nieparzysty. Mamy wówczas $6 - x$ miejsc do usadzania par. (Zauważmy, że $6 - x$ jest parzyste i ≥ 0). Teraz każdy gość spośród 1, 3, 5, 7, 9, 11, 13, 15, od lewej do prawej, który ma parzyste dziury po obu stronach, wychodzi. Każda z nieparzystych dziur zatrzymuje co najwyżej jednego z tych ośmiu gości, stąd wychodzi przynajmniej $8 - x$ osób. Wciąż jest tylko $6 - x$ miejsc do usadzania par. A tu nagle przychodzi $(8 - x)/2$ par!

38. Pokazane rozumowanie łatwo daje się uogólnić; $N(n, 2) = \lfloor (3n - 1)/2 \rfloor$ dla $n \geq 1$. [Jeśli kelnerka używałaby nie strategii optymalnej, a strategii pierwszego dopasowania, to jak pokazał Robson, konieczna i wystarczająca liczba miejsc wynosi $\lfloor (5n - 2)/3 \rfloor$].

39. Dzielimy pamięć na trzy niezależne regiony o rozmiarach $N(n_1, m)$, $N(n_2, m)$ i $N(2m - 2, m)$. Żądanie przydziału bloku realizujemy w tym regionie, którego pojemność nie została jeszcze przekroczena, korzystając w nim z optymalnej strategii (dla tego regionu). Przydział nie może się nie powieść, bo jeśli nie moglibyśmy obsłużyć żądania przydziału x lokacji, to przynajmniej $(n_1 - x + 1) + (n_2 - x + 1) + (2m - x - 1) > n_1 + n_2 - x$ lokacji musiałoby być zajętych.

Jeśli teraz $f(n) = N(n, m) + N(2m - 2, m)$, to zachodzi prawo subaddytywności $f(n_1 + n_2) \leq f(n_1) + f(n_2)$. Stąd istnieje $\lim f(n)/n$. (Dowód: $f(a + bc) \leq f(a) + bf(c)$; stąd $\limsup_{n \rightarrow \infty} f(n)/n = \max_{0 \leq a < c} \limsup_{b \rightarrow \infty} f(a + bc)/(a + bc) \leq f(c)/c$ dla dowolnego c ; stąd $\limsup_{n \rightarrow \infty} f(n)/n \leq \liminf_{n \rightarrow \infty} f(n)/n$). Wobec tego istnieje $\lim N(n, m)/n$.

[Z ćwiczenia 38 wiemy, że $N(2) = \frac{3}{2}$. Wartość $N(m)$ nie jest znana dla żadnego $m > 2$. Nietrudno pokazać, że mnożnik dla bloków o dwóch rozmiarach 1 i b równa się $2 - 1/b$; stąd $N(3) \geq 1\frac{2}{3}$. Z metody Robsona wynika, że $N(3) \leq 1\frac{11}{12}$ i $2 \leq N(4) \leq 2\frac{1}{6}$].

40. Robson, korzystając z poniższej strategii, udowodnił, że $N(2^r) \leq 1 + r$: przy każdym żądaniu przydziału bloku rozmiaru k , gdzie $2^m \leq k < 2^{m+1}$, przydzielamy pierwszy wolny blok k lokacji zaczynający się pod adresem będącym wielokrotnością 2^m .

Niech $N(\{b_1, b_2, \dots, b_n\})$ oznacza mnożnik dla przypadku, gdy wszystkie rozmiary bloków należą do zbioru $\{b_1, b_2, \dots, b_n\}$, zatem $N(n) = N(\{1, 2, \dots, n\})$. Robson i S. Krogdahl odkryli, że $N(\{b_1, b_2, \dots, b_n\}) = n - (b_1/b_2 + \dots + b_{n-1}/b_n)$, jeśli b_i jest wielokrotnością b_{i-1} dla $1 < i \leq n$; w istocie Robson podał dokładny wzór $N(2^r m, \{1, 2, 4, \dots, 2^r\}) = 2^r m(1 + \frac{1}{2}r) - 2^r + 1$. Stąd w szczególności $N(n) \geq 1 + \frac{1}{2}\lfloor \lg n \rfloor$. Robson wyprowadził także ograniczenie górne $N(n) \leq 1.1825 \ln n + O(1)$ i wysunął hipotezę, że $N(n) = H_n$. Hipoteza wynikałaby z faktu, że w ogólności $N(\{b_1, b_2, \dots, b_n\})$ równa się $n - (b_1/b_2 + \dots + b_{n-1}/b_n)$, ale niestety nie jest to prawda, ponieważ Robson udowodnił, że $N(\{3, 4\}) \geq 1\frac{4}{15}$. (Zobacz Inf. Proc. Letters 2 (1973), 96–97; JACM 21 (1974), 491–499).

41. Skupmy się na blokach rozmiaru 2^k : żądania przydziału bloków o rozmiarach 1, 2, 4, ..., 2^{k-1} będą co jakiś czas powodować konieczność podziału bloku rozmiaru 2^k lub blok takiego rozmiaru będzie zwracany. Możemy udowodnić przez indukcję względem k , że całkowity rozmiar pamięci zajmowanej przez takie podzielone bloki nigdy nie przekroczy kn , bo po każdym podziale bloku rozmiaru 2^{k+1} używamy co najwyżej kn lokacji w podzielonych 2^k -blokach i co najwyżej n lokacji w niepodzielonych.

To rozumowanie można wzmocnić, by pokazać, że wystarcza $a_r n$ komórek, gdzie $a_0 = 1$ oraz $a_k = 1 + a_{k-1}(1 - 2^{-k})$. Mamy

$$\begin{array}{ccccccc} k & = & 0 & 1 & 2 & 3 & 4 & 5 \\ a_k & = & 1 & 1\frac{1}{2} & 2\frac{1}{8} & 2\frac{55}{64} & 3\frac{697}{1024} & 4\frac{18535}{32768} \end{array}$$

W drugą stronę, dla $r \leq 5$, można pokazać, że system bloków bliźniaczych czasami wymaga $a_r n$ komórek, jeśli mechanizm w krokach R1 i R2 tak zmodyfikujemy, by do podziału wybierał najgorszy (a nie pierwszy) z wolnych 2^j -bloków.

Dowód Robsona, że $N(2^r) \leq 1 + r$ (zobacz ćwiczenie 40) można zmodyfikować, by pokazać, że strategia „skrajnie lewy” nigdy nie będzie wymagać więcej niż $(1 + \frac{1}{2}r)n$ komórek, by przydzielić pamięć dla bloków rozmiaru 1, 2, 4, ..., 2^r , ponieważ bloki rozmiaru 2^k nigdy nie zostaną umieszczone w lokacjach $\geq (1 + \frac{1}{2}k)n$. Choć algorytm Robsona bardzo przypomina system bloków bliźniaczych, okazuje się, że żaden system bloków bliźniaczych nie będzie spisywał się tak dobrze, nawet jeśli zmodyfikowalibyśmy kroki R1 i R2 w ten sposób, by do podziału wybierały najlepszy wolny 2^j -blok. Rozważmy na przykład następujący ciąg „migawek” pamięci ($n = 16$ i $r = 3$):

11111111	11111111	00000000	00000000
10101010	10101010	2-2-2-2-	00000000
11110000	11110000	2-110000	00000000
11111111	11110000	11110000	00000000
10101010	10102-2-	10102-2-	00000000
10001000	10002-00	10002-00	4--4---
10000000	10000000	10000000	4--0000

0 oznacza wolną lokację, a k – początek k -bloku. Na podobnej zasadzie dla n będącego wielokrotnością 16 można skonstruować ciąg operacji, który powoduje, że $\frac{3}{16}n$ bloków rozmiaru 8 jest wypełnionych w $\frac{1}{8}$, a inne $\frac{1}{16}n$ bloków jest wypełnionych w $\frac{1}{2}$. Jeśli n jest wielokrotnością 128, to kolejne żądania $\frac{9}{128}n$ bloków rozmiaru 8 będą wymagały ponad $2.5n$ komórek pamięci. (W systemie bloków bliźniaczych niechciane „jedynki” mogą się rozplenić w $\frac{3}{16}n$ spośród 8-bloków, ponieważ w krytycznych momentach brakuje „dwójkę”, które można by podzielić; algorytm „skrajnie lewy” ogranicza pleniące się „jedynki”).

42. Możemy założyć, że $m \geq 6$. Pomyśl polega na uzyskaniu schematu zajętości $R_{m-2}(F_{m-3}R_1)^k$ w początkowej części pamięci, dla $k = 0, 1, \dots$, gdzie R_j i F_j oznaczają zajęty i wolny blok rozmiaru j . Przejście od k do $k+1$ rozpoczynamy od

$$\begin{aligned} R_{m-2}(F_{m-3}R_1)^k &\rightarrow R_{m-2}(F_{m-3}R_1)^k R_{m-2}R_{m-2} \\ &\rightarrow R_{m-2}(F_{m-3}R_1)^{k-1} F_{2m-4}R_{m-2} \\ &\rightarrow R_{m-2}(F_{m-3}R_1)^{k-1} R_m R_{m-5} R_1 R_{m-2} \\ &\rightarrow R_{m-2}(F_{m-3}R_1)^{k-1} F_m R_{m-5} R_1; \end{aligned}$$

następnie wykonujemy k -krotnie sekwencję przestawień

$$\begin{aligned} F_{m-3}R_1F_mR_{m-5}R_1 &\rightarrow F_{m-3}R_1R_{m-2}R_2R_{m-5}R_1 \\ &\rightarrow F_{2m-4}R_2R_{m-5}R_1 \\ &\rightarrow R_mR_{m-5}R_1R_2R_{m-5}R_1 \\ &\rightarrow F_mR_{m-5}R_1F_{m-3}R_1, \end{aligned}$$

aż otrzymamy $F_mR_{m-5}R_1(F_{m-3}R_1)^k \rightarrow F_{2m-5}R_1(F_{m-3}R_1)^k \rightarrow R_{m-2}(F_{m-3}R_1)^{k+1}$. Gdy k jest dostatecznie duże, kończymy w sposób, który powoduje przepełnienia, jeżeli rozmiar pamięci nie wynosi przynajmniej $(n - 4m + 11)(m - 2)$; szczegółowo w *Comp. J.* **20** (1977), 242–244. [Zauważmy, że najgorszy przypadek, jaki można sobie wyobrazić, rozpoczynający się od schematu $F_{m-1}R_1F_{m-1}R_1F_{m-1}R_1\dots$, jest tylko nieznacznie gorszy od powyższego. Stosując strategię kolejnego dopasowania z ćwiczenia 6, można wygenerować taki fatalny schemat].

43. Pokażemy, że jeśli D_1, D_2, \dots jest dowolnym ciągiem liczb, takich że $D_1/m + D_2/(m+1) + \dots + D_m/(2m-1) \geq 1$ dla dowolnego $m \geq 1$, oraz jeśli $C_m = D_1/1 + D_2/2 + \dots + D_m/m$, to $N_{\text{FF}}(n, m) \leq nC_m$. W szczególności z uwagi na fakt

$$\frac{1}{m} + \frac{1}{m+1} + \dots + \frac{1}{2m-1} = 1 - \frac{1}{2} + \dots + \frac{1}{2m-3} - \frac{1}{2m-2} + \frac{1}{2m-1} > \ln 2,$$

ciąg stały $D_m = 1/\ln 2$ spełnia potrzebne warunki. Dowód przebiega przez indukcję względem m . Niech $N_j = nC_j$ dla $j \geq 1$ i przypuśćmy, że pewne żądania przydziału bloków rozmiaru m nie mogą być zrealizowane w skrajnie lewych N_m komórkach pamięci. Wówczas $m > 1$. Dla $0 \leq j < m$ definiujemy N'_j jako skrajnie prawą pozycję przydzieloną blokom rozmiarów $\leq j$ lub jako 0, jeśli wszystkie zajęte bloki są większe niż j . Z indukcji mamy $N'_j \leq N_j$. Ponadto definiujemy N'_m jako skrajnie prawą zajętą pozycję $\leq N_m$, zatem $N'_m \geq N_m - m + 1$. Wówczas przedział $(N'_{j-1} \dots N'_j]$ zawiera przynajmniej $\lceil j(N'_j - N'_{j-1})/(m+j-1) \rceil$ zajętych komórek, ponieważ znajdującej się w nim wolne bloki są rozmiaru $< m$, a znajdujące się w nim zajęte bloki są rozmiaru $\geq j$. Mamy stąd, że $n - m \geq$ liczba zajętych komórek $\geq \sum_{j=1}^m j(N'_j - N'_{j-1})/(m+j-1) = mN'_m/(2m-1) - (m-1) \sum_{j=1}^{m-1} N'_j/(m+j)(m+j-1) > mN_m/(2m-1) - m - (m-1) \sum_{j=1}^{m-1} N_j(1/(m+j-1) - 1/(m+j)) = \sum_{j=1}^m nD_j/(m+j-1) - m \geq n - m$, sprzeczność.

[Ten dowód to trochę więcej, niż chcieliśmy. Jeżeli zdefiniujemy D za pomocą $D_1/m + \dots + D_m/(2m-1) = 1$, to ciąg C_1, C_2, \dots jest równy $1, \frac{7}{4}, \frac{161}{72}, \frac{7483}{2880}, \dots$; wynik można dalej poprawić jak w ćwiczeniu 38, nawet w przypadku $m = 2$].

44. $\lceil F^{-1}(1/N) \rceil, \lceil F^{-1}(2/N) \rceil, \dots, \lceil F^{-1}(N/N) \rceil$.

DODATEK A

TABELE WIELKOŚCI NUMERYCZNYCH

Tabela 1

WIELKOŚCI CZĘSTO WYKORZYSTYWANE W PROCEDURACH STANDARDOWYCH
I W ANALIZIE PROGRAMÓW KOMPUTEROWYCH (40 MIEJSC DZIESIĘTNYCH)

$\sqrt{2} = 1.41421\ 35623\ 73095\ 04880\ 16887\ 24209\ 69807\ 85697-$
$\sqrt{3} = 1.73205\ 08075\ 68877\ 29352\ 74463\ 41505\ 87236\ 69428+$
$\sqrt{5} = 2.23606\ 79774\ 99789\ 69640\ 91736\ 68731\ 27623\ 54406+$
$\sqrt{10} = 3.16227\ 76601\ 68379\ 33199\ 88935\ 44432\ 71853\ 37196-$
$\sqrt[3]{2} = 1.25992\ 10498\ 94873\ 16476\ 72106\ 07278\ 22835\ 05703-$
$\sqrt[3]{3} = 1.44224\ 95703\ 07408\ 38232\ 16383\ 10780\ 10958\ 83919-$
$\sqrt[4]{2} = 1.18920\ 71150\ 02721\ 06671\ 74999\ 70560\ 47591\ 52930-$
$\ln 2 = 0.69314\ 71805\ 59945\ 30941\ 72321\ 21458\ 17656\ 80755+$
$\ln 3 = 1.09861\ 22886\ 68109\ 69139\ 52452\ 36922\ 52570\ 46475-$
$\ln 10 = 2.30258\ 50929\ 94045\ 68401\ 79914\ 54684\ 36420\ 76011+$
$1/\ln 2 = 1.44269\ 50408\ 88963\ 40735\ 99246\ 81001\ 89213\ 74266+$
$1/\ln 10 = 0.43429\ 44819\ 03251\ 82765\ 11289\ 18916\ 60508\ 22944-$
$\pi = 3.14159\ 26535\ 89793\ 23846\ 26433\ 83279\ 50288\ 41972-$
$1^\circ = \pi/180 = 0.01745\ 32925\ 19943\ 29576\ 92369\ 07684\ 88612\ 71344+$
$1/\pi = 0.31830\ 98861\ 83790\ 67153\ 77675\ 26745\ 02872\ 40689+$
$\pi^2 = 9.86960\ 44010\ 89358\ 61883\ 44909\ 99876\ 15113\ 53137-$
$\sqrt{\pi} = \Gamma(1/2) = 1.77245\ 38509\ 05516\ 02729\ 81674\ 83341\ 14518\ 27975+$
$\Gamma(1/3) = 2.67893\ 85347\ 07747\ 63365\ 56929\ 40974\ 67764\ 41287-$
$\Gamma(2/3) = 1.35411\ 79394\ 26400\ 41694\ 52880\ 28154\ 51378\ 55193+$
$e = 2.71828\ 18284\ 59045\ 23536\ 02874\ 71352\ 66249\ 77572+$
$1/e = 0.36787\ 94411\ 71442\ 32159\ 55237\ 70161\ 46086\ 74458+$
$e^2 = 7.38905\ 60989\ 30650\ 22723\ 04274\ 60575\ 00781\ 31803+$
$\gamma = 0.57721\ 56649\ 01532\ 86060\ 65120\ 90082\ 40243\ 10422-$
$\ln \pi = 1.14472\ 98858\ 49400\ 17414\ 34273\ 51353\ 05871\ 16473-$
$\phi = 1.61803\ 39887\ 49894\ 84820\ 45868\ 34365\ 63811\ 77203+$
$e^\gamma = 1.78107\ 24179\ 90197\ 98523\ 65041\ 03107\ 17954\ 91696+$
$e^{\pi/4} = 2.19328\ 00507\ 38015\ 45655\ 97696\ 59278\ 73822\ 34616+$
$\sin 1 = 0.84147\ 09848\ 07896\ 50665\ 25023\ 21630\ 29899\ 96226-$
$\cos 1 = 0.54030\ 23058\ 68139\ 71740\ 09366\ 07442\ 97660\ 37323+$
$-\zeta'(2) = 0.93754\ 82543\ 15843\ 75370\ 25740\ 94567\ 86497\ 78979-$
$\zeta(3) = 1.20205\ 69031\ 59594\ 28539\ 97381\ 61511\ 44999\ 07650-$
$\ln \phi = 0.48121\ 18250\ 59603\ 44749\ 77589\ 13424\ 36842\ 31352-$
$1/\ln \phi = 2.07808\ 69212\ 35027\ 53760\ 13226\ 06117\ 79576\ 77422-$
$-\ln \ln 2 = 0.36651\ 29205\ 81664\ 32701\ 24391\ 58232\ 66946\ 94543-$

Tabela 2

WIELKOŚCI CZĘSTO WYKORZYSTYWANE W PROCEDURACH STANDARDOWYCH
I W ANALIZIE PROGRAMÓW KOMPUTEROWYCH (45 MIEJSC ÓSEMKOWYCH)

Wielkości po lewej stronie znaku „=” są podane w notacji dziesiętnej.

0.1 =	0.06314 63146 31463 14631 46314 63146 31463 14631 46315–
0.01 =	0.00507 53412 17270 24365 60507 53412 17270 24365 60510–
0.001 =	0.00040 61115 64570 65176 76355 44264 16254 02030 44672+
0.0001 =	0.00003 21556 13530 70414 54512 75170 33021 15002 35223–
0.00001 =	0.00000 24761 32610 70664 36041 06077 17401 56063 34417–
0.000001 =	0.00000 02061 57364 05536 66151 55323 07746 44470 26033+
0.0000001 =	0.00000 00153 27745 15274 53644 12741 72312 20354 02151+
0.00000001 =	0.00000 00012 57143 56106 04303 47374 77341 01512 63327+
0.000000001 =	0.00000 00001 04560 27640 46655 12262 71426 40124 21742+
0.0000000001 =	0.00000 00000 06676 33766 35367 55653 37265 34642 01627–
$\sqrt{2}$ =	1.32404 74631 77167 46220 42627 66115 46725 12575 17435+
$\sqrt{3}$ =	1.56663 65641 30231 25163 54453 50265 60361 34073 42223–
$\sqrt{5}$ =	2.17067 36334 57722 47602 57471 63003 00563 55620 32021–
$\sqrt{10}$ =	3.12305 40726 64555 22444 02242 57101 41466 33775 22532+
$\sqrt[3]{2}$ =	1.20505 05746 15345 05342 10756 65334 25574 22415 03024+
$\sqrt[3]{3}$ =	1.34233 50444 22175 73134 67363 76133 05334 31147 60121–
$\sqrt[4]{2}$ =	1.14067 74050 61556 12455 72152 64430 60271 02755 73136+
$\ln 2$ =	0.54271 02775 75071 73632 57117 07316 30007 71366 53640+
$\ln 3$ =	1.06237 24752 55006 05227 32440 63065 25012 35574 55337+
$\ln 10$ =	2.23273 06735 52524 25405 56512 66542 56026 46050 50705+
$1/\ln 2$ =	1.34252 16624 53405 77027 35750 37766 40644 35175 04353+
$1/\ln 10$ =	0.33626 75425 11562 41614 52325 33525 27655 14756 06220–
π =	3.11037 55242 10264 30215 14230 63050 56006 70163 21122+
$1^\circ = \pi/180$ =	0.01073 72152 11224 72344 25603 54276 63351 22056 11544+
$1/\pi$ =	0.24276 30155 62344 20251 23760 47257 50765 15156 70067–
π^2 =	11.67517 14467 62135 71322 25561 15466 30021 40654 34103–
$\sqrt{\pi} = \Gamma(1/2)$ =	1.61337 61106 64736 65247 47035 40510 15273 34470 17762–
$\Gamma(1/3)$ =	2.53347 35234 51013 61316 73106 47644 54653 00106 66046–
$\Gamma(2/3)$ =	1.26523 57112 14154 74312 54572 37655 60126 23231 02452+
e =	2.55760 52130 50535 51246 52773 42542 00471 72363 61661+
$1/e$ =	0.27426 53066 13167 46761 52726 75436 02440 52371 03355+
e^2 =	7.30714 45615 23355 33460 63507 35040 32664 25356 50217+
γ =	0.44742 14770 67666 06172 23215 74376 01002 51313 25521–
$\ln \pi$ =	1.11206 40443 47503 36413 65374 52661 52410 37511 46057+
ϕ =	1.47433 57156 27751 23701 27634 71401 40271 66710 15010+
e^γ =	1.61772 13452 61152 65761 22477 36553 53327 17554 21260+
$e^{\pi/4}$ =	2.14275 31512 16162 52370 35530 11342 53525 44307 02171–
$\sin 1$ =	0.65665 24436 04414 73402 03067 23644 11612 07474 14505–
$\cos 1$ =	0.42450 50037 32406 42711 07022 14666 27320 70675 12321+
$-\zeta'(2)$ =	0.74001 45144 53253 42362 42107 23350 50074 46100 27706+
$\zeta(3)$ =	1.14735 00023 60014 20470 15613 42561 31715 10177 06614+
$\ln \phi$ =	0.36630 26256 61213 01145 13700 41004 52264 30700 40646+
$1/\ln \phi$ =	2.04776 60111 17144 41512 11436 16575 00355 43630 40651+
$-\ln \ln 2$ =	0.27351 71233 67265 63650 17401 56637 26334 31455 57005–

Wiele spośród 40-cyfrowych wartości z Tabeli 1 obliczył na biurkowym kalkulatorze John W. Wrench, Jr. na potrzeby pierwszego anglojęzycznego wydania tej książki. Kiedy w latach siedemdziesiątych dostępne stało się oprogramowanie komputerowe przeznaczone do takich obliczeń, wszystkie wyniki okazały się poprawne.

Wartości innych podstawowych stałych można znaleźć w odpowiedzi do ćwiczenia 1.3.3–23.

Tabela 3

WARTOŚCI LICZB HARMONICZNYCH, LICZB BERNOULLIEGO,
I LICZB FIBONACCIEGO DLA MAŁYCH WARTOŚCI n

n	H_n	B_n	F_n	n
0	0	1	0	0
1	1	-1/2	1	1
2	3/2	1/6	1	2
3	11/6	0	2	3
4	25/12	-1/30	3	4
5	137/60	0	5	5
6	49/20	1/42	8	6
7	363/140	0	13	7
8	761/280	-1/30	21	8
9	7129/2520	0	34	9
10	7381/2520	5/66	55	10
11	83711/27720	0	89	11
12	86021/27720	-691/2730	144	12
13	1145993/360360	0	233	13
14	1171733/360360	7/6	377	14
15	1195757/360360	0	610	15
16	2436559/720720	-3617/510	987	16
17	42142223/12252240	0	1597	17
18	14274301/4084080	43867/798	2584	18
19	275295799/77597520	0	4181	19
20	55835135/15519504	-174611/330	6765	20
21	18858053/5173168	0	10946	21
22	19093197/5173168	854513/138	17711	22
23	444316699/118982864	0	28657	23
24	1347822955/356948592	-236364091/2730	46368	24
25	34052522467/8923714800	0	75025	25
26	34395742267/8923714800	8553103/6	121393	26
27	312536252003/80313433200	0	196418	27
28	315404588903/80313433200	-23749461029/870	317811	28
29	9227046511387/2329089562800	0	514229	29
30	9304682830147/2329089562800	8615841276005/14322	832040	30

Dla dowolnego x niech $H_x = \sum_{n \geq 1} \left(\frac{1}{n} - \frac{1}{n+x} \right)$. Wówczas

$$H_{1/2} = 2 - 2 \ln 2,$$

$$H_{1/3} = 3 - \frac{1}{2}\pi/\sqrt{3} - \frac{3}{2}\ln 3,$$

$$H_{2/3} = \frac{3}{2} + \frac{1}{2}\pi/\sqrt{3} - \frac{3}{2}\ln 3,$$

$$H_{1/4} = 4 - \frac{1}{2}\pi - 3\ln 2,$$

$$H_{3/4} = \frac{4}{3} + \frac{1}{2}\pi - 3\ln 2,$$

$$H_{1/5} = 5 - \frac{1}{2}\pi\phi^{3/2}5^{-1/4} - \frac{5}{4}\ln 5 - \frac{1}{2}\sqrt{5}\ln\phi,$$

$$H_{2/5} = \frac{5}{2} - \frac{1}{2}\pi\phi^{-3/2}5^{-1/4} - \frac{5}{4}\ln 5 + \frac{1}{2}\sqrt{5}\ln\phi,$$

$$H_{3/5} = \frac{5}{3} + \frac{1}{2}\pi\phi^{-3/2}5^{-1/4} - \frac{5}{4}\ln 5 + \frac{1}{2}\sqrt{5}\ln\phi,$$

$$H_{4/5} = \frac{5}{4} + \frac{1}{2}\pi\phi^{3/2}5^{-1/4} - \frac{5}{4}\ln 5 - \frac{1}{2}\sqrt{5}\ln\phi,$$

$$H_{1/6} = 6 - \frac{1}{2}\pi\sqrt{3} - 2\ln 2 - \frac{3}{2}\ln 3,$$

$$H_{5/6} = \frac{6}{5} + \frac{1}{2}\pi\sqrt{3} - 2\ln 2 - \frac{3}{2}\ln 3,$$

i ogólnie dla $0 < p < q$ (zobacz ćwiczenie 1.2.9–19)

$$H_{p/q} = \frac{q}{p} - \frac{\pi}{2} \cot \frac{p}{q}\pi - \ln 2q + 2 \sum_{1 \leq n < q/2} \cos \frac{2pn}{q}\pi \cdot \ln \sin \frac{n}{q}\pi.$$

DODATEK B

WYKAZ OZNACZEŃ

Przyjmuje się poniższe znaczenia zmiennych, o ile opis nie stanowi inaczej:

j, k	wyrażenie arytmetyczne o wartości całkowitej
m, n	wyrażenie arytmetyczne o wartości nieujemnej
x, y	wyrażenie arytmetyczne o wartości rzeczywistej
f	funkcja o wartościach rzeczywistych lub zespolonych
P	wyrażenie o wartości wskaźnikowej (Λ lub adres w pamięci komputera)
S, T	zbiór lub multizbiór
α	napis (ciąg symboli)

Oznaczenie symboliczne	Znaczenie	Gdzie definicja
$V \leftarrow E$	nadanie zmiennej V wartości wyrażenia E	1.1
$U \leftrightarrow V$	zamiana miejscami wartości zmiennych U i V	1.1
A_n lub $A[n]$	n -ty element tablicy liniowej A	1.1
A_{mn} lub $A[m, n]$	element w wierszu m i kolumnie n tablicy prostokątnej A	1.1
$\text{NODE}(P)$	element (zestaw zmiennych rozróżnialnych po nazwach odpowiadających im pól) o adresie P ; zakładamy, że $P \neq \Lambda$	2.1
$F(P)$	zmienna w $\text{NODE}(P)$ o nazwie pola F	2.1
$\text{CONTENTS}(P)$	zawartość słowa maszynowego o adresie P	2.1
$\text{LOC}(V)$	adres zmiennej V w pamięci komputera	2.1
$P \Leftarrow \text{AVAIL}$	przypisanie zmiennej wskaźnikowej P adresu nowego elementu	2.2.3
$\text{AVAIL} \Leftarrow P$	zwracanie $\text{NODE}(P)$ na stertę	2.2.3
$\text{top}(S)$	element na szczytce niepustego stosu S	2.2.1
$X \Leftarrow S$	przypisanie wierzchołka S do X : $X \leftarrow \text{top}(S)$; następnie usunięcie $\text{top}(S)$ z niepustego stosu S	2.2.1
$S \Leftarrow X$	włożenie X na stos S : wstawienie wartości X jako nowego elementu na szczytce stosu S	2.2.1

Oznaczenie symboliczne	Znaczenie	Gdzie definicja
$(B \Rightarrow E; E')$	wyrażenie warunkowe: oznacza E , jeśli B jest prawdziwe, zaś E' , jeśli B jest fałszywe	
$[B]$	funkcja charakterystyczna warunku B : $(B \Rightarrow 1; 0)$	1.2.3
δ_{kj}	delta Kroneckera: $[j = k]$	1.2.6
$[z^n] g(z)$	współczynnik przy z^n w szeregu potęgowym $g(z)$	1.2.9
$\sum_{R(k)} f(k)$	suma wszystkich $f(k)$, takich że zmienna k jest całkowita i spełniona jest zależność $R(k)$	1.2.3
$\prod_{R(k)} f(k)$	iloczyn wszystkich $f(k)$, takich że zmienna k jest całkowita i spełniona jest zależność $R(k)$	1.2.3
$\min_{R(k)} f(k)$	najmniejsza wartość spośród wszystkich $f(k)$, takich że zmienna k jest całkowita i spełniona jest zależność $R(k)$	1.2.3
$\max_{R(k)} f(k)$	największa wartość spośród wszystkich $f(k)$, takich że zmienna k jest całkowita i spełniona jest zależność $R(k)$	1.2.3
$j \setminus k$	j dzieli k : $k \bmod j = 0$ i $j > 0$	1.2.4
$S \setminus T$	różnica zbiorów: $\{a \mid a \text{ należy do } S \text{ i } a \text{ nie należy do } T\}$	
$gcd(j, k)$	największy wspólny dzielnik j i k : $(j = k = 0 \Rightarrow 0; \max_{d \setminus j, d \setminus k} d)$	1.1
$j \perp k$	j jest względnie pierwsze z k : $gcd(j, k) = 1$	1.2.4
A^T	transpozycja tablicy prostokątnej A : $A^T[j, k] = A[k, j]$	1.2.3
α^R	odbicie lustrzane α prawo-lewo	
x^y	x do potęgi y (dla dodatnich x)	1.2.2
x^k	x do potęgi k : $\left(k \geq 0 \Rightarrow \prod_{0 \leq j < k} x; \quad 1/x^{-k} \right)$	1.2.2
$x^{\bar{k}}$	x do potęgi k przyrostającej: $\Gamma(x+k)/\Gamma(x) =$ $\left(k \geq 0 \Rightarrow \prod_{0 \leq j < k} (x+j); \quad 1/(x+k)^{-k} \right)$	1.2.5
x^k	x do potęgi k ubywającej: $x!/(x-k)! =$ $\left(k \geq 0 \Rightarrow \prod_{0 \leq j < k} (x-j); \quad 1/(x-k)^{-k} \right)$	1.2.5

Oznaczenie symboliczne	Znaczenie	Gdzie definicja
$n!$	n silnia: $\Gamma(n+1) = n^n$	1.2.5
$\binom{x}{k}$	współczynnik dwumianowy: ($k < 0 \Rightarrow 0$; $x^k/k!$)	1.2.6
$\binom{n}{n_1, n_2, \dots, n_m}$	współczynnik wielomianowy (określony jedynie wtedy, gdy $n = n_1 + n_2 + \dots + n_m$)	1.2.6
$\left[\begin{smallmatrix} n \\ m \end{smallmatrix} \right]$	liczba Stirlinga pierwszego rodzaju: $\sum_{0 < k_1 < k_2 < \dots < k_{n-m} < n} k_1 k_2 \dots k_{n-m}$	1.2.6
$\left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\}$	liczba Stirlinga drugiego rodzaju: $\sum_{1 \leq k_1 \leq k_2 \leq \dots \leq k_{n-m} \leq m} k_1 k_2 \dots k_{n-m}$	1.2.6
$\{a \mid R(a)\}$	zbiór wszystkich a , takich że jest spełnione $R(a)$	
$\{a_1, \dots, a_n\}$	zbiór lub multizbiór $\{a_k \mid 1 \leq k \leq n\}$	
$\{x\}$	część ułamkowa (oznaczenie używane w kontekście, z którego wynika, że chodzi o wartość rzeczywistą, a nie o zbiór): $x - \lfloor x \rfloor$	1.2.11.2
$[a .. b]$	przedział domknięty: $\{x \mid a \leq x \leq b\}$	1.2.2
$(a .. b)$	przedział otwarty: $\{x \mid a < x < b\}$	1.2.2
$[a .. b)$	przedział lewostronnie domknięty: $\{x \mid a \leq x < b\}$	1.2.2
$(a .. b]$	przedział prawostronnie domknięty: $\{x \mid a < x \leq b\}$	1.2.2
$ S $	moc zbioru, liczba kardynalna: liczba elementów w zbiorze S	
$ x $	wartość bezwzględna x : ($x \geq 0 \Rightarrow x; -x$)	
$ \alpha $	długość α	
$\lfloor x \rfloor$	podłoga z x , największa liczba całkowita nie większa niż x : $\max_{k \leq x} k$	1.2.4
$\lceil x \rceil$	sufit z x , najmniejsza liczba całkowita nie mniejsza niż x : $\min_{k \geq x} k$	1.2.4
$x \bmod y$	funkcja modulo: ($y = 0 \Rightarrow x; x - y \lfloor x/y \rfloor$)	1.2.4
$x \equiv x' \pmod{y}$	relacja kongruencji: $x \bmod y = x' \bmod y$	1.2.4
$O(f(n))$	wielkie o od $f(n)$, przy $n \rightarrow \infty$	1.2.11.1
$O(f(z))$	wielkie o od $f(z)$, przy $z \rightarrow 0$	1.2.11.1
$\Omega(f(n))$	wielkie omega od $f(n)$, przy $n \rightarrow \infty$	1.2.11.1
$\Theta(f(n))$	wielkie theta od $f(n)$, przy $n \rightarrow \infty$	1.2.11.1

Oznaczenie symboliczne	Znaczenie	Gdzie definicja
$\log_b x$	logarytm x przy podstawie b (dla $x > 0$, $b > 0$, $b \neq 1$): y jest takie, że $x = b^y$	1.2.2
$\ln x$	logarytm naturalny: $\log_e x$	1.2.2
$\lg x$	logarytm dwójkowy: $\log_2 x$	1.2.2
$\exp x$	funkcja wykładnicza zmiennej x : e^x	1.2.2
$\langle X_n \rangle$	uneskończony ciąg X_0, X_1, X_2, \dots (litera n stanowi część oznaczenia)	1.2.9
$f'(x)$	pochodna f w punkcie x	1.2.9
$f''(x)$	druga pochodna f w punkcie x	1.2.10
$f^{(n)}(x)$	n -ta pochodna: ($n = 0 \Rightarrow f(x); g'(x)$), gdzie $g(x) = f^{(n-1)}(x)$	1.2.11.2
$H_n^{(x)}$	liczba harmoniczna rzędu x : $\sum_{1 \leq k \leq n} 1/k^x$	1.2.7
H_n	liczba harmoniczna: $H_n^{(1)}$	1.2.7
F_n	liczba Fibonacciego: $(n \leq 1 \Rightarrow n; F_{n-1} + F_{n-2})$	1.2.8
B_n	liczba Bernoulliego: $n! [z^n] z/(e^z - 1)$	1.2.11.2
$\det(A)$	wyznacznik macierzy kwadratowej A	1.2.3
$\text{sign}(x)$	znak x : $[x > 0] - [x < 0]$	
$\zeta(x)$	funkcja dzeta: $\lim_{n \rightarrow \infty} H_n^{(x)}$ (dla $x > 1$)	1.2.7
$\Gamma(x)$	funkcja gamma: $(x-1)! = \gamma(x, \infty)$	1.2.5
$\gamma(x, y)$	niepełna funkcja gamma: $\int_0^y e^{-t} t^{x-1} dt$	1.2.11.3
γ	stała Eulera: $\lim_{n \rightarrow \infty} (H_n - \ln n)$	1.2.7
e	podstawa logarytmu naturalnego: $\sum_{n \geq 0} 1/n!$	1.2.2
π	stosunek obwodu do średnicy okręgu: $4 \sum_{n \geq 0} (-1)^n / (2n+1)$	
∞	uneskończoność: wartość większa od dowolnej liczby	
Λ	wskaźnik pusty (nie wskazuje na żaden adres)	2.1
ϵ	napis pusty (o długości zero)	
\emptyset	zbiór pusty (bez elementów)	
ϕ	złota proporcja: $\frac{1}{2}(1 + \sqrt{5})$	1.2.8
$\varphi(n)$	funkcja Eulera: $\sum_{0 \leq k < n} [k \perp n]$	1.2.4
$x \approx y$	x jest w przybliżeniu równe y	1.2.5, 4.2.2

Oznaczenie symboliczne	Znaczenie	Gdzie definicja
$\Pr(S(X))$	prawdopodobieństwo, że stwierdzenie $S(X)$ jest prawdziwe dla losowej wartości X	1.2.10
$E X$	wartość oczekiwana zmiennej X : $\sum_x x \Pr(X = x)$	1.2.10
$\text{mean}(g)$	wartość średnia rozkładu prawdopodobieństwa reprezentowanego przez funkcję tworzącą g : $g'(1)$	1.2.10
$\text{var}(g)$	wariancja rozkładu prawdopodobieństwa reprezentowanego przez funkcję tworzącą g : $g''(1) + g'(1) - g'(1)^2$	1.2.10
$(\min x_1, \text{ave } x_2, \max x_3, \text{dev } x_4)$	zmienna losowa o wartości mimmimalnej x_1 , średniej (oczekiwanej) x_2 , maksymalnej x_3 i odchyleniu standardowym x_4	1.2.10
P^*	adres następnika NODE(P) w porządku preorder w drzewie lub drzewie binarnym	2.3.1, 2.3.2
$P\$$	adres następnika NODE(P) w porządku postorder w drzewie lub następnika w porządku inorder w drzewie binarnym	2.3.1, 2.3.2
$P\#$	adres następnika NODE(P) w porządku postorder w drzewie binarnym	2.3.1
$*P$	adres poprzednika NODE(P) w porządku preorder w drzewie lub drzewie binarnym	2.3.1, 2.3.2
$\$P$	adres poprzednika NODE(P) w porządku postorder w drzewie lub poprzednika w porządku inorder w drzewie binarnym	2.3.1, 2.3.2
$\sharp P$	adres poprzednika NODE(P) w porządku postorder w drzewie binarnym	2.3.1
\blacksquare	koniec algorytmu, programu lub dowodu	1.1
\sqcup	jeden znak odstępu	1.3.1
rA	rejestr A (akumulator) maszyny MIX	1.3.1
rX	rejestr X (rozszzerzenie) maszyny MIX	1.3.1
$rI1, \dots, rI6$	rejestry I1, ..., I6 (indeksowe) maszyny MIX	1.3.1
rJ	rejestr J (skoku) maszyny MIX	1.3.1
$(L:R)$	wycinek słowa maszyny MIX, $0 \leq L \leq R \leq 5$	1.3.1
$\text{OP ADDRESS}, I(F)$	oznaczenie rozkazu maszyny MIX	1.3.1, 1.3.2
u	jednostka czasu na maszynie MIX	1.3.1
$*$	odniesienie do bieżącego adresu w języku MIXAL	1.3.2
$0F, 1F, 2F, \dots, 9F$	odniesienia w przód do symboli lokalnych w języku MIXAL	1.3.2
$0B, 1B, 2B, \dots, 9B$	odniesienia wstecz do symboli lokalnych w języku MIXAL	1.3.2
$0H, 1H, 2H, \dots, 9H$	deklaracja symbolu lokalnego w języku MIXAL	1.3.2

SKOROWIDZ ZE SŁOWNIKIEM

Są tacy, co udają, że zrozumieli całą książkę,
a przeczytali tylko skorowidz.
To tak jakby opowiadać o pałacu,
gdy widziało się tylko wchodek.

— JONATHAN SWIFT, *Mechanical Operation of the Spirit* (1704)

Gdy pozycja skorowidza dotyczy ćwiczenia, warto zatrzymać uwagę również do rozwiązań. Hasła zawarte w rozwiązańach występują w skorowidzu tylko wtedy, gdy nie pojawiają się w sformułowaniu ćwiczenia.

- (-) 170, zob. także permutacja
identycznościowa
0-2-drzewo 329
zorientowane 414
 Δ 242–243
 π 23, 647
 Π , iloczyn Wallisa 54, 119
 ϕ , zob. złota proporcja 84, 86–90
- Aardenne-Ehrenfest, Tatyana van 390, 394
Aarons, Roger M. 550
Abel, Niels Henrik 60, 519, 527
ACE, komputer 199
Adams, Charles William 238
ADD 135–136, 214
Adobe Systems 209
adres: liczba określająca położenie
w pamięci 241
bazowy 252
w języku MIXAL 150, 156–157
adresowanie pośrednie 255, 260–261
agenda, zob. lista nadchodzących zdarzeń
Aho, Alfred Vaino 585
Ahrens, Wilhelm Ernst Martin Georg 168
aktywne czekanie 222
aktywowanie współprogramów 200,
206, 230, 302
al-Khwārizmī, Abū ‘Abd Allāh Muḥammad
ibn Mūsā محمد بن موسى الخوارزمي 1, 82
ALF (dane alfanumeryczne) 156–157, 161
ALGOL 208, 237
algorytm 1–10
analiza 7, 9, 99–110, 176–178, 185,
259, 261, 288–290, 336–338, 344,
395–397, 462–463, 469
definicja teoriomnogościowa 8–10
dowód poprawności 5–6, 15–18, 333,
375, 438, 451
Euklidesa 2–10, 83
zmodyfikowany 15–16, 44
Huffmana 418–422
jak czytać 4, 18
algorytm oznaczania: algorytm, który
„oznacza” wszystkie elementy osiągalne
z danego elementu 281–282, 431–440
- algorytm
postać 2–4
przekazywanie innym 18
słowo 1–2
wieloprzebiegowy 204–206
własności 4–6, 9
wydajny 6, 8–9
wyznaczania relacji równoważności
374–375, 604, 607
znajdowania
liczb pierwszych 151–154
maksimum 99–104, 149
algorytm/program optymalizacyjny 192
algorytmy
komplikacji 208, 374, 441–452, 580
niskopoziomowe 28, 639
równoważne 486
sprzętowe 28, 261, 639
związane z tablicą symboli 181, 443
Alpern, Steven Robert 548
alternatywa
rozłączna 473
wykluczająca 461
AMM: *American Mathematical Monthly*
– czasopismo wydawane przez
Mathematical Association of America
od 1894 r.
analiza algorytmów 7, 99–110, 176–178,
185, 259, 261, 288–290, 336–338,
344, 395–397, 462–463, 469
analizator dynamiczny 307
André, Antoine Désiré 559
Apostol, Tom Mike 30
Araújo, Saulo 590
Arbogast, Louis François Antoine 54, 109
argumenty podprogramów 193, 195
arytmetyczny: dotyczący dodawania,
odejmowania, mnożenia i dzielenia
arytmetyka
całkowitoliczbowa 163
stałopozycyjna 163
zmiennopozycyjna 135, 318
assembler: (1) język programowania
pozwalający na korzystanie z nazw
symbolicznych i mnemonicznych

- do zapisu programów w języku maszynowym; (2) program tłumaczący kod w asemblerze na język maszynowy 150, 157
- asembler**
komputera MIX 149–162
kontra język wysokiego poziomu 244, 558
- asercje indukcyjne** 12–19
- asympotyka** 110–127, 251, 412, 547, 590
- atom** (na liście) 327, 424–429, 435
- atomy, po co?** 427
- atrapa**
(głowa) listy 284–285, 290–291,
298–299, 313–314
w drzewie sfastrygowanym 336, 350
w Liście 426, 430, 433
w liście dwukierunkowej 290–291,
298–299, 459, 461
w sfastrygowanym drzewie binarnym
324, 346
- automat:** formalnie zdefiniowana maszyna abstrakcyjna; w zamierzeniu często model pewnych aspektów rzeczywistego komputera
z dowiązaniami 482–484
- AVAIL, zob.** lista AVAIL
- Babbage, Charles 1, 236
- Bachmann, Paul Gustav Heinrich 111
- Backus, John Warner 238
- Bailey, Michael John 481
- bajt:** podstawowa jednostka danych, zazwyczaj równoważna jednemu znakowi alfanumerycznemu 129
maszyny MIX 128–129, 146
- Baker, Henry Givens, Jr. 633
- Ball, Thomas Jaudon 383
- Ball, Walter William Rouse 168
- Barnes, Ernest William 528
- Barnett, Michael Peter 481
- Barrington, David Arno 549
- Barry, David McAlister, dyskretny układ w kierunku xv, 283
- Barton, David Elliott 68, 560
- Beigel, Richard 486
- Bell, Eric Temple 91
- Bell Interpretive System 238
- Bellman, Richard Ernest xix
- Bendix G20 128
- Bennett, John Makepeace 238
- Berger, Robert 401
- Bergeron, François 411
- Bergman, George Mark 515, 620
- Berlekamp, Elwyn Ralph 283
- Berman, Martin Fredric 545
- Bernoulli, Jacques (= Jakob = James) 116, 119
- Bernoulli, Jean (= Johann = John), III 516
- Bertrand, Joseph Louis François 533
- Berztiss, Alfs Teodors 481
- bęben** 140–142
- Bhāskara Āchārya II (भास्कराचार्य) 56
- Bienaymé, Irénée Jules 101
- Bienstock, Daniel 536
- Bigelow, Richard Henry 587
- Binet, Jacques Philippe Marie 37, 423, 495, 608
- bit:** cyfra dwójkowa (ang. *Binary digit*); zero lub jedynka
- bit – znacznik „ostatnio używany”** 471
- BIT:** *Nordisk Tidskrift for Informations-Be-handling* – międzynarodowe czasopismo wydawane w Skandynawii od 1961 r.
- Blaauw, Gerrit Anne 477
- Blikle, Andrzej Jacek 342
- blok**
danych zewnętrznych 141–142
pamięci 453
- błąd** 580
bezwzględny 120
obliczeniowy 26–28
względny 120
- błędne rozumowanie** 20, 115, 484
- błędy**
obliczeń 316
unikanie 269–270
- Bobrow, Daniel Gureasko 438, 479–480
- bocznica kolejowa** 248
- Boles, David Alan 471
- Bolzano, Bernhard 397
- Boncompagni, Prince Baldassarre 82
- Boothroyd, John 183
- bootstrapping 148
- Borchardt, Carl Wilhelm 422, 609
- Bourne, Charles Percy 538
- brat, w drzewie** 323
- Brenner, Norman Mitchell 545
- Brent, Richard Peirce 589 .
- Briggs, Henry 28
- Broline, Duane Marvin 628
- Brouwer, Luitzen Egbertus Jan 423
- Bruijn, Nicolaas Govert de 125, 126, 390, 394, 395, 498, 525, 567, 590
- buforowanie wejścia-wyjścia** 163, 223–236
historia 238
wymiana (przełączanie buforów) 152, 165, 224–225, 233
- bufory**
przełączanie 152, 165, 224–225, 233
wspólne 231, 235
zwalnianie 226–230, 233–235
- bug:** (pluskwa) błąd w programie,
zob. uruchamianie
- Burke, John 323
- Burks, Arthur Walter 373
- Burleson, Peter Barrus 481
- Burroughs B220 128
- Burroughs B5000 481

- Busche, Conrad Heinrich Edmund
Friedrich 45
bycie własnym dziadkiem 570
- C 237, 241, 580
CACM: Communications of the ACM – czasopismo wydawane przez Association for Computing Machinery od 1958 r.
- Cajori, Florian 25
California Institute of Technology xiv, 292
całka funkcji wykładniczej 519
całkowanie 93
a sumowanie 115–120
przez części 80, 116–117
- Campbell, John Arthur 468
Capelli, Alfredo 53, 74
Carlitz, Leonard 99, 520, 527
Carlyle, Thomas xviii
Carpenter, Brian Edward 237
Carr, John Weber, III 477
Case Institute of Technology i
Cassini, Jean Dominique 84
Catalan, Eugène Charles 423
Cate, Esko George 545
Cauchy, Augustin Louis 38, 95, 495,
510, 528, 542, 608
macierz 39–40, 495
nierówność 38
- Cayley, Arthur 412, 422, 423, 612, 625
CDC 1604, komputer 128, 552
centroid
położenie 616
w drzewie 403–404, 413
- Chakravarti, Gurugovinda (গুরুগোবিন্দ চক্রবর্তী) 56
CHAR (konwersja na znak) 143
Cheam, Tat Ong 468
Chen, Tien Chi (陳天機) 490
Cheney, Christopher John 438
Cheney, Ednah Dow Littlehale 392
Chernoff, Herman 523
Chowla, Paromita (পারমিতা চৌলা) 318
Christie Mallowan, Agatha Mary Clarissa (Miller) xxi
Chu Shih-Chieh (= Zhū Shíjié Hànqīng, Zhū Sōngtíng; 朱世傑漢卿, 朱松庭) 56, 62, 73
Chung, Fan Rong King (鍾金芳蓉) 536
Chung, Kai Lai (鍾開萊) 109
CI: symboliczna nazwa wskaźnika porównania w komputerze MIX,
zob. wskaźnik porównania
- ciąg
arytmetyczny 12, 33–34, 59
Fareya 166
Fibonacciego 14, 20, 82–90, 649
geometryczny 33, 91
liniowo-rekurencyjny 91
ciągi dwójkowe 625–626
- CITRUS 476
Clark, Douglas Wells 632
Clavius, Christopher 165
CMath: Matematyka konkretna – książka autorstwa R. L. Grahama, D. E. Knutha i O. Patashnika 12
- CMP1 (porównaj rI1) 138, 217
CMPA (porównaj rA) 138, 217
CMPX (porównaj rX) 138, 217
COBOL 441–452, 476–477
Coffman, Edward Grady, Jr. 469–470
Cohen, Jacques 438, 480, 642
Collins, George Edwin 480
Comfort, Webb T. 481
COMIT 480
Comp. J.: The Computer Journal – czasopismo wydawane przez British Computer Society od roku 1958 r.
- CON (stała) 154–155, 161
CONTENTS 132, 243–244
Conway, John Horton 20, 83, 283, 424, 628
Conway, Melvin Edward 156, 237
Corless, Robert Malcolm 411
Coxeter, Harold Scott Macdonald 83, 168, 423, 424
córka, w drzewie 323
- Crelle: Journal für die reine und angewandte Mathematik* – międzynarodowe czasopismo założone przez A. L. Crellego w 1826 r.
- Crelle, August Leopold 60, 658
Crowe, Donald Warren 628
cyfra: jeden z symboli wykorzystywanych do zapisu pozycyjnego liczb
cyfra dziesiętna: jeden z symboli 0, 1, ..., 9
cykl: ścieżka prowadząca od wierzchołka do niego samego
elementarny 392
Eulera 389–391, 394, 611
zliczanie 394–395
Hamiltona 389, 394
jednoelementowy 170
podstawowy 380–383
singleton, w permutacji 170, 177, 186–187
w grafie 376
skierowany 376
wykrywanie 280
zorientowany 386
- cykle w permutacjach 170–172, 182–183, 188–190
w permutacji losowej 185–190
wykrywanie 280
- cykliczna definicja, zob. definicja, cykliczna
czas blokowania: czas, w którym następuje wstrzymanie działania jednej części systemu, podczas gdy inna część kończy wykonanie pewnego zadania

- czas
 symulacji 293, 298
 ścieżki krytycznej 224
 wykonania
 metody wyznaczania 99, 176–178
 rozkazu maszyny MIX 143–144
 zamortyzowany 263
 Czebyszew, Pafnutij Lwowicz (Чебышев, Панфутий Львович), nierówność 101, 107
 czekanie aktywne 222
 częstość występowania liter 165
 częśc
 adresowa instrukcji komputera MIX 131
 rzeczywista liczby zespolonej 23
 ułamkowa 42
 urojona liczby zespolonej 23
 czterokierunkowe drzewo binarne 346
 czytanie
 z dysku, buforowane 236
 z wyprzedzeniem: *zob.* buforowanie 223
 Ćwiczenia, uwagi do xix–xxi, 294
- d'Imperio, Mary Evelyn 481
 Dahl, Ole-Johan 237, 480, 481
 Dahm, David Michael 450, 451
 dalekopis 140–142, 239
 dane: precyzyjna, formalna reprezentacja
 pewnych faktów lub pojęć, często wartości numerycznych lub znakowych, umożliwiająca wykonywanie na nich obliczeń 222
 upakowane: dane zapisane w małej przestrzeni, np. dwa elementy umieszczone w jednym słowie pamięci 163
 wejściowe 5
 wyjściowe 5
 David, Florence Nightingale 68
 Davies, David Julian Meredith 463
 Davis, Philip Jacob 53
 Dawson, Reed 610
 de Bruijn, Nicolaas Govert 125, 126, 390, 394, 395, 498, 525, 567, 590
 de Moivre, Abraham 77, 86, 90, 109, 188, 560
 De Morgan, Augustus 18
 DEC1 (zmniejsz rI1) 138, 216
 DECA (zmniejsz rA) 138, 216
 DECX (zmniejsz rX) 138, 216
 definicja
 cykliczna, *zob.* cykliczna definicja
 efektywna 6, 8–9
 nieznanego przez nieznane (*ignotum per ignotum*) 272, 320
 rekurencyjna 348, 359
 drzewa 320
 drzewa binarnego 324, 330
 listy 327
- definicja
 rekurencyjna
 porządku przechodzenia drzewa 331
 wielomianu 371
 definicje liczb 22
 deklaracje równoważności 374
 delta Kroneckera 35, 64, 652
 Demuth, Howard B. 124
 Dershowitz, Nachum 540, 614
 Deuel, Phillip DeVere, Jr. 580
 Deutsch, Laurence Peter 434, 438–439
 Dewar, Robert Berriedale Keith 642
 Dewey, Melvil, notacja 325, 329, 343, 359, 397–398, 421, 479
 Diaconis, Persi Warren 512
 diagram dziedziczenia 323
 Dickman, Karl Daniel 548
 Dickson, Leonard Eugene 84
 digraf 386, *zob.* graf skierowany
 Dijkstra, Edsger Wijbe 19, 237, 239, 248, 478, 481, 568, 606, 633
 DIV (dzielenie) 136–137, 146, 215
 Dixon, Alfred Cardew 511
 Dixon, Robert Dan 531
 DLINK w Listach 425, 428
 długość
 przewodów, minimalna 385
 ścieżki 415–422
 minimalna 416
 średnia 421
 ważona 418–421
 wewnętrznej 418, 421
 średnia 421
 zewnętrznej 416, 421
 dno stosu 249
 dobre uporządkowanie 21, 346
 dodatni: większy od zera (ale nie zerowy)
 dodawanie
 do listy: *zob.* wstawianie
 stronami 500
 wielomianów 285–290, 369–372
 domknięcie indukcyjne 494
 dopasowanie wg rozkładu 469–470, 475
 dopełnienie algebraiczne elementu
 w macierzy prostokątnej: wyznacznik macierzy otrzymanej po zamianie tego wybranego elementu na jedynkę i wyzerowaniu wszystkich innych elementów w jego wierszu i kolumnie 39, 396
 Doran, Robert William 237
 Dougall, John 511
 dowiązania
 cykliczne 283–290, 313, 369, 427, 478
 dwustronne 290–292
 fastrygowe 334, 438
 historia 477–480
 manipulowanie 269–270
 po co 449–450, 481

dowiązanie 241–245
 do dziecka 444–450
 do rodzeństwa 444–450
 do rodzica 360, 366–369, 373–375,
 387, 391, 444–450
LLINK 290–292, 298
PARENT 360
 puste (Λ) 242–243, 330
 sposób reprezentowania 242
 w drzewie 330
 w drzewie binarnym 335, 344
RLINK, *zob.* **RLINK**
SCOPE, *zob.* **SCOPE**
 dowód poprawności algorytmu 5–6, 15–18,
 333, 375, 438, 451
 dowód własności stopu 18, 21–22, 401
 Doyle, Arthur Conan 484
 drukarka wierszowa 140–142
 drzewa 240, 320–440
 naturalna odpowiedniość 347–348, 359
 podobne 340–342, 359
 rozłączne, *zob.* las
 równoważne 340–342
 własności matematyczne 376–424
 zliczanie 393–394, 401–415, 422–424
 drzewo 240, 320–440
 binarne 323–324, 329–350, 376, 478
 atrapa 336, 346, 350
 czterokierunkowe 346
 definicja 324
 definicja rekurencyjna 324, 330
 długość ścieżki 415–422
 dobre uporządkowanie 346
 dowiązanie **LLINK** 330, 335, 340,
 346, 478
 dowiązanie puste 335, 344
 koplowanie 342–343, 346, 360
 lewe poddrzewo 324, 330
 o nałożonych poddrzewach 339, 631
 odwiedzanie węzła 332
 porządek
 inorder 331–332, 336, 343–345, 359
 podwójny 344, 346, 589
 postorder 331, 334, 343–345, 359
 preorder 331, 334, 343–345, 359
 symetryczny 332, 343–345
 prawe poddrzewo 324, 330
 przechodzenie 331–332, 336, 344–345,
 348, 479
 reprezentacja 330, 334, 340, 346–347
 rozszerzone 415–422
 równoważność 340–342
 schemat 324, 330, 588
 sfastrygowane 334, 344
 sfastrygowane a niesfastrygowane 339
 tablicowe 417
 uporządkowanie liniowe 345
 w notacji Dewey'a 325, 329, 343, 359,
 397–398, 421, 479

drzewo
 binarne
 z czterema dowiązaniami 346
 z dowiązaniami 330
 z fastrygą prawostronną 340, 345–346
 usuwanie 346
 wstawianie węzła 339, 345–346
 zliczanie 404–405, 421
 zorientowane 412
 zupełne 417, 421
 czwórkowe 589
 definicja 320, 329, 377, 387
 rekurencyjna 320, 324, 330
 dowiązanie **LLINK** 351, 361, 373, 395
 dwumianowe xxiv
 dwupodziąłowe 414
 dwuwymiarowe 589
 etykietowane, zliczanie 405, 423
 fastrygowanie 346
 Fibonacciego 516
 genealogiczne 323, 329, 422
 historia 422–424, 478–479
 konstruowanie 353–354, 356, 445–446
 koplowanie 342–343, 346, 360
 korzeń 320–321, 329
 nieskończone 329, 397–401
 nieukorzenione 377, *zob.* drzewo wolne
 notacja dziesiętna Dewey'a 325, 329, 343,
 359, 397–398, 421, 479
 palmowe 607
 płaskie 320
 porównanie typów 320–321, 388
 porządek
 liniowy 345, 359
 postorder 348–353, 359, 361, 479
 poziomy 364, 589, 603
 preorder 348–350, 359, 361, 479
 poziom węzła 320, 328–329
 przechodzenie 331–332, 336, 344–345,
 348, 479
 reprezentacja 361–369, 373–376, 479
 tablicowa 361–365, 373–376, 450
 w porządku poziomym 364, 373
 w porządku preorder 362, 375
 w porządku rodzinnym 364
 rodowe 323, 329
 rozmiar węzła 453, 471
 rozpinające, minimalne 385
 schemat 321–327, 350, 360, 362, 479
 sfastrygowane 348, 479
 a niesfastrygowane 339
 t-arne 346, 413, 417, 421, 620
 tablicowe 417
 zliczanie 413, 620
 zupełne 417–418
 tablicowe 361–365, 373–376, 450
 terminologia 323
 ternarne 346, 417

- drzewo
 uporządkowane 320–321, 388, 404–405,
 414, 423
 zliczanie 404–405, 414, 423
 usuwanie węzłów 372
 włożenie drzewa w drzewo 361, 401
 wolne 376–386
 definicja 377
 zliczanie 402–404, 414, 423
 wstawianie węzła 339, 345–346
 z czterema dowiązaniami 369
 z dowiązaniami (wskaźnikowe) 347,
 365–369
 z fastrygą prawostronną 351, 395
 z trzema dowiązaniami 366, 373, 444–450
 zorientowane 321, 324, 386–397
 definicja 387
 dowiązanie PARENT 360
 konwersja na drzewo uporządkowane
 360
 reprezentacja kanoniczna 406–410,
 413–414, 617–618
 reprezentacja w komputerze 360,
 366, 392
 zliczanie 401–402, 405–411, 423
 zmiana korzenia 392
 zupełne 417, 421, 588
- Dunlap, James Robert 476
- Dutka, Jacques 53, 68
- Dvoretsky, Aryeh 620
- dwa stosy 254, 260, 262–263
- Dwyer, Barry 593
- dynamiczne przydzielanie pamięci 254–263,
 265–268, 429–431, 453–475
 czas wykonania 467
 historia 476–477, 481
- DYSEAC komputer 239
- dysk 140–142, 454, 482
- dziadek, w drzewie 570
- działania na potęgach, prawa 23, 27, 55
- dziecko, w drzewie 323, 329, 347–348, 366
- dziedziny euklidesowe 488
- dzielenie zamienione na mnożenie 538–540
- dzielnik: x jest dzielnikiem y , jeśli
 $y \bmod x = 0$ i $x > 0$; jest on
 dzielnikiem właściwym y , jeśli jest
 takim dzielnikiem, że $1 < x < y$
- Earley, Jackson Clark 481
- Edelman, Paul Henry 625
- Edwards, Daniel James 439
- efektywność 8–9
- Egorychev, Georgii Petrovich (Егорычев,
 Георгий Петрович; Jegoryczew
 Gieorgij Pietrowicz) 520
- Eisele, Peter 499
- Eisenstein, Ferdinand Gotthold Max 499
- element struktury danych 241, 481–484
 adres 241
- element struktury danych
 dowiązanie 241
 pola 241–245
 reprezentacja graficzna 242
 rozmiar 266, 310, 453, 471
- elementarne funkcje symetryczne 40, 97, 517
- emulator 209
- END 156, 161, 306
- Engles, Robert William 481
- ENN1 (wpisz wartość przeciwną do rII)
 137, 216
- ENNA (wpisz wartość przeciwną do rA)
 137, 216
- ENNX (wpisz wartość przeciwną do rX)
 137, 216
- ENT1 (wpisz wartość do rII) 137, 216
- ENTA (wpisz wartość do rA) 137, 216
- ENTX (wpisz wartość do rX) 137, 216
- Epiktet (Epictetus of Hierapolis; Ἐπίκτητος
 ὁ Ἱεραπόλεως) 1
- EQU (równoważne z) 151, 154, 160
- Erdélyi, Arthur 415
- Erdwinn, Joel Dyne 237
- Etherington, Ivor Malcolm Haddon 415
- Ettingshausen, Andreas von 56
- etymologia 1–2
- Euler, Leonhard (Эйлер, Леонард) 51,
 53–54, 60, 78–79, 90, 115, 389,
 423, 492, 517, 559, 628
 cykl 389–391, 394, 611
 liczby drugiego rzędu 527
 funkcja tocjent $\varphi(n)$ 44, 190
- Evans, Arthur, Jr. 208
- Faà di Bruno, Francesco 503
- FADD (dodawanie zmiennopozycyjne) 318
- faktoryzacja, zob. rozkład na czynniki
 pierwsze
- Farber, David Jack 480
- Farey, John 542
- fastryga 334, 438
- fastrygowanie drzewa 346
- FCMP (porównanie zmiennopozycyjne)
 529, 584
- FDIV (dzielenie zmiennopozycyjne) 318
- Ferguson, David Elton 239, 347
- Fermat, Pierre de 18, 485
- Ferranti Mark I, komputer 19
- Feynman, Richard Phillips 28
- Fibonacci, Leonardo, of Pisa 82–83, 87
 ciąg 14, 20, 82–90, 649
 drzewo 516
 liczby 14, 20, 82–90, 649
 system bloków bliźniaczych 474
 system liczbowy 89, 516
- Fich, Faith Ellen 545
- Fidiasz, syn Charmidesa (Phidias; Φειδίας
 Χαρμίδου) 84
- FIFO 248, 364, 478, 635, zob. kolejka
- filotaksja 83

- filozofia wskaźnikowa 453
 filtry 204
 Fine, Nathan Jacob 504
 Fischer, Michael John 366
 Flajolet, Philippe Patrick Michel 522,
 527, 567, 590
 Flawiusz, Józef, syn Mateusza (יוסף בן מתתיהו
 = Φλάβιος Ἰωσηπος Ματθίου) 168, 190
 Floyd, Robert W. 19–21, 439, 486, 530
 FLPL 479–480
 Flye Sainte-Marie, Camille 611
FMUL (mnożenie zmiennopozycyjne) 318
FOCS: Proceedings of the IEEE Symposia
 on Foundations of Computer Science
 (1975–), wcześniej zwane *Symposia*
 on Switching Circuit Theory and
 Logic Design (1960–1965), *Symposia*
 on Switching and Automata Theory
 (1966–1974)
 Ford, Donald Floyd 538
 Förstemann, Wilhelm 510
 FORTRAN 239, 241, 307, 374, 477, 479
 Foster, Frederic Gordon 104
 Fourier, Jean Baptiste Joseph 29
 Fraenkel, Aviezri S. 260
 fragmentacja 457, 468, 474
 Fredman, Michael Lawrence 536
 Friedman, Daniel Paul 438
FSUB (odejmowanie zmiennopozycyjne) 318
 Fuchs, David Raymond 209
 Fukuoka, Hiromumi (福岡博文) 529
 funkcja
 beta $B(x, y)$ 75
 charakterystyczna rozkładu
 prawdopodobieństwa 106
 digamma $\psi(z)$ 46, 79, 513
 drzewowa $T(z)$ 375, 411
 dzeta, Riemana $\zeta(s)$ 46, 79
 gamma $\Gamma(z)$ 51–55, 75, 82, 119, 121–123
 niepełna 121–126
 multiplikatywna 45
 obliczanie 364
 powielająca 45–46
 psi $\psi(z)$ 46, 78, 513
 punkt osobliwy 412
 sign x 495
 symetryczna 95–98, 517
 tocjent Eulera $\varphi(n)$ 44, 190
 tworząca 85–87, 90–99, 251, 402–405,
 407–408, 410–415, 562
 ciągu Fibonacciego 85–86
 dyskretnego rozkładu prawdopodo-
 bieństwa 101–110, 187
 podwójna 98, 412, 421, 560, 562
 prawdopodobieństwa 102–110
 wykładnicza ciągu $\langle a_n \rangle$: $\sum a_n z^n/n!$ 92
 wkłesła 422, 624
 wypukła 420, 624
 funkcje
 hipergeometryczne 68
 podstawowe 510
 indeksujące 309–312, 316–319
 symetryczne 40, 95–98, 517
 elementarne 40, 97, 517
 trygonometryczne 46, 236, 490
 Furch, Robert 125
 Galler, Bernard Aaron 366
 Galton, Francis 587, 659
 Gao, Zhicheng (高志成) 590
 Gardner, Martin 20, 83, 614
 Garwick, Jan Vaumund 257, 476
 Gaskell, Robert Eugene 89
 Gasper, George, Jr. 510
 Gates, William Henry, III xv
 Gauß (= Gauss), Johann Friedrich Carl
 (= Carl Friedrich) 51, 60, 98
 gcd: zob. największy wspólny dzielnik
 Gelernter, Herbert Leo 479–480
 Genuys, François 239
 Gerberich, Carl Luther 480
 Gill, Stanley 237, 238, 476
 Girard, Albert 517
 Glaisher, James Whitbread Lee 525
 Glassey, Charles Roger 422
 główna listy 290, zob. także atrapa
 Gnedenko, Boris Vladimirovich (Гнеденко,
 Борис Владимирович; Gniedienko,
 Borys Władimirowicz) 109
 Goldbach, Christian 51, 492
 Goldberg, Joel 550
 Goldman, Alan Joseph 616
 Goldstine, Herman Heine 19, 237
 Golomb, Solomon Wolf 190
 Columbic, Martin Charles
 מונחים צבויים באנדרה וולומברק) 623
 Goncharov, Vasilii Leonidowicz (Гончаров,
 Василий Леонидович; Gonczarow,
 Wasilij Leonidowicz) 522
 Gonnet Haas, Gaston Henry 411
 Good, Irving John 389, 411, 503, 610
 Gopāla (गोपाल) 83
 Gorn, Saul 479
 Gosper, Ralph William, Jr. 67
 Gould, Henry Wadsworth 60, 66, 125,
 505, 513
 Gourdon, Xavier Richard 547
 Gower, John Clifford 478
 gra wojenna 283, 574
 Grabner, Peter Johannes 527
 graf 376–387, 483
 cykl podstawowy 380–381, 383, 387
 krawędziowy, skierowany 394
 skierowany 386–389, 438
 cykl Eulera 389–391, 394, 611
 cykl Hamiltona 389, 394

- graf
 skierowany
 cykl zorientowany 386
 jako schemat blokowy 378–379, 387
 krawędziowy 394
 regularny 394
 silnie spójny 387, 392
 ścieżka 386
 zrównoważony 389
 spójny 376
 zorientowany, zob. skierowany
 graficzna reprezentacja struktur danych 242
 grafika 168
 Graham, Ronald Lewis 12, 658
 gramatyki bezkontekstowe 563
 Griswold, Ralph Edward 480
 Grójecka, ulica 206
 Grünbaum, Branko 399, 614
 grupowanie rekordów 225, 233
 gry 282–283
 Guy, Richard Kenneth 20, 83, 283, 628
 gwiazdka (*) w asemblerze 151, 154, 158
- H-drzewa 589
 Haddon, Bruce Kenneth 642
 Hadeler, Karl-Peter Fritz 499
 Hageman, Louis Alfred 612
 Halāyudha (हलायुध) 56
 Hamel, Georg 500
 Hamilton, William Rowan
 cykl 389, 394
 Hamming, Richard Wesley 27
 Hankel, Hermann 52
 Hansen, James Rone 480
 Hansen, Wilfred James 438
 Haralambous, Yannis (Χαραλάμπους, Ἰωάννης) 679
 Harary, Frank 423
 Hardy, Godfrey Harold 14, 422, 513, 542
 Hare, David Edwin George 411
 Haros, C. 542
 Hartmanis, Juris 483
 Harvard Mark I, kalkulator 236
 Hautus, Matheus Lodewijk Johannes 509
 Heine, Heinrich Eduard 510
 Hellerman, Herbert 478
 Hemachandra, Āchārya (आचार्य हेमचन्द्र) 83
 Henkin, Léon Albert 18
 Henrici, Peter Karl Eugen 91
 Herbert, George xviii
 Hermite, Charles 51, 498
 Heyting, Arend 423
 hierarchia pamięci 205, 438, 454, 482
 Hiles, John Owen 438
 Hilbert, David
 macierz 40
 Hill, Robert 540
 Hipparch (Hipparchus of Nicæa; Ἡππαρχος Νικαιας) 619
- HLT (zatrzymanie) 140, 147
 Hoare, Charles Antony Richard 19,
 237, 480–481
 Hobbes, Thomas 679
 Hobby, John Douglas 679
 Holmes, Thomas Sherlock Scott 484
 Holt Hopfenberg, Anatol Wolf 479
 Honeywell H800, komputer 128
 Hopcroft, John Edward 585
 Hopper, Grace Brewster Murray 236
 Hu, Te Chiang (胡德強) 421, 623
 Huang Bing-Chao (黃秉超) 182
 Huffman, David Albert 418, 424
 algorytm 418–422
 Hurwitz, Adolf 46, 509
 twierdzenie dwumianowe 46, 415, 509
 Hwang, Frank Kwangming (黃光明)
 68–69, 422, 623
- I/O: zob. wejście-wyjście 222
 I1, rejestr komputera MIX 129, 146
 IBM 650, komputer i, 128, 238, 551
 IBM 701, komputer 238
 IBM 705, komputer 238
 IBM 7070, komputer 128
 IBM 709, komputer 128, 551
 Ibn al-Haytham 168
 Iliffe, John Kenneth 481
 Illiac I, komputer 238
 iloczyn 35
 iloraz 41
 d'Imperio, Mary Evelyn 481
 IN (wejście) 141, 222–223
 INC1 (zwiększa II) 138, 216
 INCA (zwiększa RA) 138, 216
 INCX (zwiększa RX) 138, 216
 indeks: liczba wskazująca pozycję w tablicy
 4, 309, 325, 327
 próbny 29
 indeksowanie 4
 od zera 263, 292, 310–312, 316–317
 indukcja matematyczna 12–22, 34, 328, 494
 ogólniona 21
 strukturalna 333, 590, 595
 infinum: zob. kres dolny
 informacja: znaczenie przypisywanie
 danym – fakty reprezentowane za
 pomocą danych; termin ten występuje
 też w węższym znaczeniu, jako
 synonim słowa „dane” lub w szerszym
 znaczeniu, obejmującym wszystkie
 własności danych
 informacje, struktura, zob. struktury danych
 Ingalls, Daniel Henry Holmes 544
 inicjowanie tablicy, sztuczka 319
 inorder, zob. porządek inorder
 instrukcja, zob. rozkaz
 INT (przerwanie) 236
 interpreter 207–209, 353
 inwersja 66, 566, 603

- IOC (sterowanie wejścia-wyjścia) 142
 IPL 238, 477–480, 576
 Isaacs, Irving Martin 628
 Itai, Alon (אילן איטי) 558
 Iverson, Kenneth Eugene 35, 41, 479
 konwencja 34–35, 64, 106
- J1N (skok, gdy rI1 ujemne) 139, 216
 J1NN (skok, gdy rI1 nieujemne) 139, 216
 J1NP (skok, gdy rI1 niedodatnie) 139, 216
 J1NZ (skok, gdy rI1 niezerowe) 139, 216
 J1P (skok, gdy rI1 dodatnie) 139, 216
 J1Z (skok, gdy rI1 równe zero) 139, 216
JACM: Journal of the AC – publikacja Association for Computing Machinery wydawana od 1954 r.
 Jacob, Simon 83
 Jacquard, Joseph Marie 237
 Jakacki, Grzegorz Paweł (瑞佳仕) iii
 JAN (skok, gdy rA ujemne) 139, 216
 JANN (skok, gdy rA nieujemne) 139, 216
 JANP (skok, gdy rA niedodatnie) 139, 216
 JANZ (skok, gdy rA niezerowe) 139, 216
 JAP (skok, gdy rA dodatnie) 139, 216
 Jarden, Dov (דב ירדן) 88, 515
 Java 241
 JAZ (skok, gdy rA zero) 139, 216
 JBUS (skok, gdy zajęte) 142, 162, 218, 222
 JE (skok, gdy równe) 139, 215
 Jeffrey, David John 411
 Jenkins, D. P. 479
 język: zbiór ciągów symboli (składnia),
 zazwyczaj w połączeniu z zasadami określającymi ich „znaczenie”
 (semanticzka) xii–xiii
 C 237, 241, 580
 Java 241
 język komputerowy, zob. język programowania, C, ALGOL,
 FORTRAN, język maszynowy, asembler
 język maszynowy: język interpretowany
 bezpośrednio przez sprzęt komputerowy,
 zob. asembler xii–xiv, 128
 język programowania: język formalny
 służący do zapisu programów 249, 480
 JG (skok, gdy większe) 139, 215
 JGE (skok, gdy większe lub równe) 139, 215
 JL (skok, gdy mniejsze) 139, 215
 JLE (skok, gdy mniejsze lub równe) 139, 215
 JMP (skok) 139, 193, 215, 298
 JNE (skok, gdy nierówne) 139, 215
 JNOV (skok przy braku przepelnienia)
 139, 146, 215
 Jodeit, Jane Griffin 481
 Johnson, Lyle Robert 479
 Johnstone, Mark Stuart 471
 Jones, Clifford Bryn 19
 Jones, Mary Whitmore 392
 Jonkers, Henricus (= Hans) Bernardus
 Maria 642
- Jordan, Marie Ennemond Camille 404, 423
 JOV (skok, gdy przepelnienie) 139, 146, 215
 Joyal, André 411
 JRED (skok, gdy gotowe) 142, 229–230
 JSJ (skok, zachowaj rJ) 139, 195, 216, 554
 JXN (skok, gdy rX ujemne) 139, 216
 JXNN (skok, gdy rX nieujemne) 139, 216
 JXNP (skok, gdy rX niedodatnie) 139, 216
 JXNZ (skok, gdy rX niezerowe) 139, 216
 JXP (skok, gdy rX dodatnie) 139, 216
 JXZ (skok, gdy rX zero) 139, 216
- kafelki 398–401
 Kahn, Arthur B. 278
 Kahrimanian, Harry George 478
 kalendarz 165
 Kallick, Bruce 420
 Kaplansky, Irving 190
 Karamata, Jovan 69
 Karp, Richard Manning 422
 karta perforowana 140–142, 156, 237
 karty do gry 53, 72, 241–246, 392–393
 kasowanie listy liniowej 289
 Katz, Leo 617
 Kaucký, Josef 66
 Keller, Helen Adams 127
 Kepler, Johann 83–84
 Kilmer, Alfred Joyce 240
 King, James Cornelius 21
 Kirchhoff, Gustav Robert 422, 609
 pierwsze prawo 100, 176, 278, 288,
 378–385
 Kirkman, Thomas Penyngton 424
 Kirschenhofer, Peter 527
 Klärner, David Anthony 90
 Kleitman, Daniel J (Isaiah Solomon)
 571, 623
 Knopp, Konrad Hermann Theodor 50, 519
 Knowlton, Kenneth Charles 481
 Knuth, Donald Ervin (高德纳) ii, vi, xv, 12,
 35, 69, 124, 199, 208, 209, 307, 308, 411,
 476, 481, 490, 504, 520, 525, 545, 548,
 590, 605, 606, 610, 619, 623, 679
 Knuth, Nancy Jill Carter (高精蘭)
 xiv, xxiv
 kod
 operacji komputera MIX 131
 w języku MIXAL 150
 samomodyfikujący się 193, 199
 znakowy, MIX 141
 kodowanie: (tu:) mało elegancki synonim
 terminu „programowanie”
 kody
 alfanumeryczne (tabela 1) 144
 MIX 142
 trudności ćwiczeń xxi
 kolejka 247–252, 273–275, 478, 603, 635
 z dowiązaniami 268–270, 279,
 283–284, 299

- kolejka
 dwustronna 247–252, 279
 o ograniczonym wejściu 247–252, 433
 o ograniczonym wyjściu 247–252,
 279, 284
 tablicowa 260
 z dowiązaniami 290, 307
 usuwanie elementu 260, 307
 wstawianie elementu 260, 307
 początek 250
 priorytetowa 580, 617
 tablicowa 252–261, 263
 usuwanie elementu 250, 253, 260, 263,
 270, 275, 283–284, 307
 wstawianie elementu 252–253, 260, 263,
 269, 275, 283–284, 307
 Kolmogorov, Andrei Nikolaevich
 (Колмогоров, Андрей Николаевич)
 Kolmogorow Andriej Nikołajewicz
 108–109, 483
 kombinacje 55, 72
 z ograniczonymi powtórzeniami 98
 z powtórzeniami 76, 402, 404
 kombinatoryczny system liczbowy 76, 585
 komentarze, w asemblerze 2–3, 150, 154
 komórka pamięci: jedno słowo pamięci
 komputera 132
 kompilator: program tłumaczący język(i)
 programowania
 kompresja
 ścieżki 602
 wiadomości 424
 kompresowanie pamięci 439, 457, 467,
 470, 474
 komputer: maszyna do przetwarzania danych
 komputer dwójkowy: komputer
 przeprowadzający większość obliczeń
 na liczbach w reprezentacji dwójkowej,
 por. komputer dziesiętny
 komputer dziesiętny: komputer
 przeprowadzający większość obliczeń
 na liczbach w reprezentacji dziesiętnej,
 por. komputer dwójkowy
 komputer MIX, zob. MIX
 komputer RISC (Reduced Instruction Set
 Computer) komputer o zredukowanej
 liście rozkazów 128
 kongruencja 42
 koniec
 kolejki 250
 pliku 223, 235
 König, Dénes 397, 423, 614
 konkatenacja 284
 konsolidacja 281–282
 konstruowanie drzew 353–354, 356,
 445–446
 kontynuanty 628
 konwencja Iversona 34–35, 64, 106
 kopianie
 drzewa binarnego 342–343, 346, 360
 dwuwymiarowej listy wskaźnikowej 318
 i kompresja 437
 Listy 439
 listy liniowej 289
 kopiowanie struktury danych: wykonanie
 kopii obiektu reprezentującego
 dane poprzez utworzenie innego
 obiektu reprezentującego te same
 wartości i powiązania
 korzeń drzewa 320–321, 329
 zorientowanego, zmiana 392
 grafu skierowanego 387
 Koster, Cornelis (= Kees) Hermanus
 Antonius 480
 Kozelka, Robert Marvin 568
 Kramp, Christian 51
 krata: struktura algebraiczna z operacjami
 będącymi uogólnieniem \cup i \cap
 krata
 Tamari'ego 603, 625
 wolna 360
 kratowy porządek na lasach 603, 625
 Krattenthaler, Christian 41
 krawędź w grafie 376
 kres dolny: największe ograniczenie dolne
 kres górny: najmniejsze ograniczenie
 górne 38
 Kremeras, Germain 625
 Krogdahl, Stein 645
 Kronecker, Leopold 35, 652
 delta 35, 64, 652
 krotka 22, 35
 królik i psy, zob. gra wojenna 283
 Kruskal, Joseph Bernard 401, 614
 krzyżówka 168
 Kummer, Ernst Eduard 72
 Kung, Hsiang Tsung (孔祥重) 589
 kupka kart 241
 Kuratowski, Kazimierz 571
 kuzyn, w drzewie 329
 kwadrat magiczny 167
 Labelle, Gilbert 411
 Lagrange (= de la Grange), Joseph
 Louis, Comte
 wzór inwersyjny 408, 621
 Lamé, Gabriel 423
 Lamport, Leslie B. 633
 Lapko, Olga Georgievna (Лапко,
 Ольга Георгиевна; Łapko, Olga
 Georgijewna) 679
 Laplace (= de la Place), Pierre Simon,
 Marquis de 90
 laplasjan grafu 609
 Larus, James Richard 383
 las: zero lub więcej drzew, zob. drzewa
 321, 424
 a drzewo binarne 347–348, 359

- las
 reprezentacja 347
 uporządkowanie kratowe 603, 625
 zliczanie 405, 621
 zorientowany 366–369
- lasy
 notacja indeksowa 325, 327, 329
 podobieństwo 359
 równoważność 359
- Lawson, Harold Wilbur, Jr. 450, 480
- LCHILD 366, 373
- LD1 (załaduj wartość do rI1) 133, 215
- LD1N (załaduj wartość przeciwną do rI1) 134, 215
- LDA (załaduj wartość do rA) 133, 215
- LDAN (załaduj wartość przeciwną do rA) 134, 215
- LDX (załaduj wartość do rX) 133, 215
- LDXN (załaduj wartość przeciwną do rX) 134, 215
- Leeuwen, Jan van 623
- Legendre (= Le Gendre), Adrien Marie 51, 53
- Léger, Émile 83
- Lehmer, Derrick Henry 484
- Leibniz, Gottfried Wilhelm, Freiherr von 2, 53
- Leighton, Frank Thomson 469, 470
- Leiner, Alan Lewine 239
- lemat
 Kuratowskiego-Zorna, zob. lemat Zorna o drzewach nieskończonych 397–401
 Watsona 127
 Zorna 571
- Leonardo Pisano (Leonardo z Pizy) 82
- Leroux, Pierre 411
- LeVeque, William Judson 485
- Lévy, Paul 109
- Levy, Silvio Vieira Ferreira xv
- lewe poddrzewo drzewa binarnego 324, 330
- liczba sprzężona 23
- liczby
 Bernoulliego 79, 95, 116–119
 tabela 649
 całkowite 22
 podziały 13, 35, 96
- Catalana 423
 definicje 22
- Eulera drugiego rzędu 527
- Fibonacciego F_n (elementy ciągu Fibonacciego) 14, 20, 82–90
 tabela 649
- harmoniczne H_n 78–82, 118
 funkcja tworząca 93
 tabela 649–650
- Lucasa L_n 515
 pierwsze 20, 43, 47, 50, 53, 72–73, 87–88
 algorytm obliczający 151–154
 faktoryzacja 44
- liczby
 rzeczywiste 23
 Stirlinga 68–72, 74–77, 81, 102–104, 527, 608
 asymptotyka 68
 funkcja tworząca 94
 interpretacja kombinatoryczna 69, 77, 185
 mod p 513
 prawo dualności 71
 tabela 68
 wymierne 22, 27, 166
- względnie pierwsze 42
- zespolone
 część rzeczywista 23
 część urojona 23
- liczenie od zera 292, 310–312, 316–317
- licznik
 dowiązań 429–431, 438, 480
 lokacji 160
- LIFO 248, 471, 478, zob. ostatni-wchodzi-pierwszy-wychodzi, zob. także stos
- Lilius, Aloysius 165
- Lindstrom, Gary Edward 593
- Linsky, Vladislav Sergeevich (Линский, Владислав Сергеевич; Linski Władisław Siergiejewicz) 490
- LISP 241, 479–480
- lista: uporządkowany ciąg zera lub więcej elementów
 AVAIL 265–270, 275, 278, 288, 302, 429–431, 453–462
 historia 477
 cykliczna 283–290, 313, 369, 427, 478
 dowiązanie LLINK 428
 dwukierunkowa 290–292, 298–301, 307–309, 369, 427, 459, 461, 470, 478
 porównanie z listą jednokierunkową 291, 309
 jednokierunkowa 263, 267–269, 427
 liniowa 242–319
 kasowanie 289
 kopiowanie 289
 odwracanie 279, 289
 sekwencyjna 337
 tablicowa 273–275, 476
 z dowiązaniem 242–245, 273–275, 279, 476–478
- nadchodzących zdarzeń 298, 302, 307, 580
- pole DLINK 425, 428
- rekurencyjna 328
- wolnej pamięci, zob. lista AVAIL
- wskaźnikowa dwuwymiarowa, kopiowanie 318
- Lista (wielką literą) 327–328, 424–440, 479–481
- a lista 241, 427
- kopiowanie 439
- notacja 327–328, 425

- Lista (wielką literą)
 reprezentacja 425–429, 438
 schemat 327–329, 425
 tablicowa 438
 Listing, Johann Benedict 423
 listy ortogonalne 309–319
 Listy, równoważność 440
 liść 320, 330, 366, 413, 624
 Littlewood, John Edensor 422
 LLINK (dowiązanie do lewego) 290–292, 298
 w drzewie 351, 361, 373, 395
 w drzewie binarnym 330, 335, 340,
 346, 478
 w liście 428
 LLINKT 337
 Lloyd, Stuart Phinney 189–190
 LOC 243–244
 Logan, Benjamin Franklin (= Tex), Jr. 77
 logarytm 24–28
 dwójkowy 24, 27
 dziesiętny 24
 naturalny 27
 logarytmiczny szereg potęgowy 94
 Loopstra, Bram Jan 239
 Lovász, László 512
 Lovelace, Augusta Ada Byron King,
 Countess of 1
 LTAG 335, 346, 362–363, 366, 373, 395
 Lucas, François Édouard Anatole 72,
 83–84, 88, 283, 515
 Luhn, Hans Peter 476
 Luo, Jianjin (罗见今) 423
 Lynch, William Charles 611
- ładowanie pierwszego programu 148
 łańcuch: (1) zbiór uporządkowany liniowo;
 (2) ciąg znaków, słowo; (3) lista liniowa
 łańcuch Markowa 395–397
 Lukasiewicz, Jan 350
- macierz: tablica dwuwymiarowa 309, 327
 Cauchy'ego 39–40, 495
 Hilberta 40
 incydencji 280
 kombinatoryczna 39–40, 616
 odwrotna 39–40, 76, 318
 osiowanie 313–316, 318
 osobliwa 318
 permanent 54
 reprezentacja 163–164, 309–319
 rozszerszalna 318
 rzadka 312–318
 mnożenie 318
 tablicowa 316–319
 tetrahedryczna 317
 transpozycja 188
 trójdiamondalna 318
 trójkątna 311, 317
 unimodularna 628
- macierz
 Vandermonde'a 39–40, 495
 wielomian charakterystyczny 520
 wyznacznik 39–41, 84, 393–394, 397
 MacMahon, Percy Alexander 511, 615
 Madnick, Stuart Elliot 480
 Mailloux, Barry James 480
 makro: nazwa reprezentująca ciąg instrukcji
 maksimum/minimum lewo-, prawostronne
 100–104, 108–110, 185
 malloc: zob. dynamiczne przydzielanie
 pamięci 453
 Mallows, Colin Lingwood 560
 mapa pamięci 453–454
 Margolin, Barry Herbert 468
 Mark I (kalkulator; Harvard) 236
 Mark I (komputer; Ferranti) 19
 Markov, Andrei Andreevich (Марков,
 Андрей Андреевич; Markow Andriej
 Andriejewicz), młodszy 9
 Markov, Andrei Andreevich (Марков,
 Андрей Андреевич; Markow Andriej
 Andriejewicz), starszy 516
 łańcuch 395–397
 Markowitz, Harry Max 481
 Markowsky, George 83, 420
 Martin, Alain Jean 633
 Martin, Johannes Jakob 642
 maszyna
 analityczna 1, 236
 Turinga 9, 237, 483
 wirtualna 207
- Math.Comp.: Mathematics of Computation*
 (1960–) – publikacja American
 Mathematical Society wydawana
 od 1965 r.; założona przez National
 Research Council of the National
 Academy of Sciences pod pierwotnym
 tytułem *Mathematical Tables
 and Other Aids to Computation*
 (1943–1959)
- Matiyasevich, Yuri Vladimirovich
 (Матијасевич, Јуриј Владимирович;
 Matijasiewicz Jurij Władimirowicz) 90
- matka, w drzewie 323
 Matrix (Bush), Irving Joshua 37–38
 Mauchly, John William 237
 Maurolico, Francesco 18
 McCall's ix
 McCann, Anthony Paul 642
 McCarthy, John 479–480
 McEliece, Robert James 497
 McIlroy, Malcolm Douglas 602, 607
 Mealy, George 481
 Meek, H. V. 238
 Meggitt, John Edward 490
 Melville, Robert Christian 563
 Merner, Jack Newton Forsythe 237
 Merrett, Timothy Howard 585

- Merrington, Maxine 68
METAFONT vi, xv, 639, 679
METAPOST xv, 679
 metoda
 bloków bliźniaczych, *zob.* system
 bloków bliźniaczych
 kolejnego dopasowania 467, 472, 646
 Monte Carlo 262
 najdawniej używany 470
 najgorszego dopasowania 471, 636
 najlepszego dopasowania 454–455,
 465, 471–475
 obliczeniowa 8–9
 pierwszego dopasowania 454–456,
 471–475
 znaczników brzegowych 472, 481
 zstępująca 375
 mieszanka rozkładów prawdopodobieństwa
 109
 Miller, Kenneth William 127
 Ming, An-T'u (明安圖) 423
 Minsky, Marvin Lee 440
 Mirsky, Leon 614
 Mitchell, William Charles 548
MIX xii–xiv, 128–149
 asembler 149–162
 kod znakowy 141
 kody alfanumeryczne 142
 rozkazy 131–149
 arytmetyczne 135–137, 214
 konwersji 142
 ładowania wartości do
 pamięci 215
 rejestru 133, 146, 215
 porównania 138, 217
 przekazania adresu 216
 przesunięć 139, 218
 skoku 139, 215
 symboliczne 132, 149
 wejścia 140–142, 222–223
 wyjścia 140–142
 zapisywania wartości w pamięci 134
 zmiennopozycyjne 318
 rozszerzenie 148, 235, 260–261, 473
 symulator 210–218
 zestawienie rozkazów 144
MIXAL: asembler maszyny **MIX** 149–162,
 243–244
 pole wierszu języka 150, 157
 pole adresu w wierszu języka 156–157
 pole OP wiersza języka 160
MMIX (komputer) 128, 193, 221, 337
 mnożenie
 macierzy rzadkich 318
 permutacji 171–172, 178–179, 386
 wielomianów 287, 290, 375
 Mock, Owen Russell 239
 mod 41–42
 modulo 42
 moduł
 liczby zespolonej 23
 w przystawaniu 42
 Mohammed, John Llewelyn 550
 moment rozkładu prawdopodobieństwa 108
 Moon, John Wesley 423
 Mordell, Louis Joel 498
 Morris, Francis Lockwood 19, 642
 Morris, Joseph Martin 592
 Morrison, Emily Kramer 236
 Morrison, Philip 236
 Moser, Leo 68
 Motzkin, Theodor Samuel
 תִּיאוֹדָר שְׁמוּאֵל מוֹצְקִין (1883–1950) 88, 620
MOVE 140, 147, 199, 218
MOVE CORRESPONDING 443, 447–448, 451
MUL (mnożenie) 136, 214
 multizbiór: struktura podobna do zbioru z tą
 różnicą, że elementy mogą się powtarzać
 Munro, James Ian 545
 Nagorny, Nikolai Makarovich (Нагорный,
 Николай Макарович; Nagorný,
 Nikolaј Makarowicz) 9
 Nahapetian, Armen 605
 największy wspólny dzielnik (gcd) 2–10,
 15–16, 84–85
 Napier, John, Laird of Merchiston 25
 napis: skończony ciąg zera lub więcej
 symboli z danego alfabetu 8, 89,
 191, 284
 konkatenacja 284
 przetwarzanie 480
 Nash, Paul 581
 National Science Foundation xiv
 naturalna odpowiedniość między drzewami
 binarnymi i lasami 347–348, 359
 Naur, Peter 19
 nawias Iversona 34–35, 64, 106
 nawiasy zagnieżdzone 324–325, 362, 625
 nazwa uściślona 441–452
 Needham, Joseph 62
 Neely, Michael 471
 Neumann, John von (= Margittai Neumann
 János) 19, 237, 476
 Neville, Eric Harold 618
 Newell, Allen 238, 477, 480
 Newton, Isaac 24, 60, 517
 Nicolau, Alexandru 642
NIEDOMIAR 253, 255, 268, 278, 284
 Nielsen, Norman Russell 468
 niepełna funkcja gamma $\gamma(a, x)$ 121–126
 nieporządkи 186, 189
 nierówności ogonowe 107, 110
 nierówność
 Cauchy'ego-Schwarza $(\sum a_k b_k)^2 \leq (\sum a_k^2)(\sum b_k^2)$ 38
 Czebyszewa 101, 107
 nieujemny: dodatni lub równy zeru
 niezmienniki 19

- niezwykła odpowiedniość 184–185
 Nikomachus (Nicomachus of Gerasa; Νικόμαχος ὁ ἐξ Γεράσαων) 20
 Niven, Ivan Morton 91
NODE 244
 Noe syn Lamecha (Noah ben Lamech, נָחָר בֶּן לָמֶךְ) 323
NOP (operacja pusta) 140
 norma maksimum 110
 notacja
 Π dla iloczynów 35
 cyklowa 170–172, 178–181, 188
 dziesiętna Dewey'a 325, 329, 343, 359, 397–398, 421, 479
 indeksowa dla lasów 325, 327, 329
 nawiasowa dla współczynników 95
 dla zdań 34–35, zob. nawiasy Iversona
 polska 350, 620, zob. notacja
 przedrostkowa, notacja przyrostkowa
 przedrostkowa (zapis przedrostkowy) 350
 przyrostkowa (zapis przyrostkowy)
 350, 365, 620
 wielkie- Ω 114–115
 wielkie- O 111–115, 122
 wielkie- Θ 114
 NUM (konwersja na liczbę) 142
 NWD, zob. największy wspólny dzielnik
 Nygaard, Kirsten 237, 480
- O'Beirne, Thomas Hay 540
 obliczać: przetwarzanie dane
 obliczanie funkcji 364
 na drzewie 375
 potęg 530
 obliczenia symboliczne: obliczenia
 nienumeryczne, np. przetwarzanie słów
 lub wzorów algebraicznych
 obszar pamięci 453
 odchylenie standardowe rozkładu
 prawdopodobieństwa: pierwiastek
 kwadratowy z wariancji, miara tego,
 jak bardzo wartość losowa odbiega od
 wartości oczekiwanej 101
 Odlyzko, Andrew Michael 125, 590
 odśmiecacz LISP 2 642
 odśmiecanie 266, 429–440, 456–457, 467, 474, 480, 575
 wydajność 437
 w systemach czasu rzeczywistego 440
 odwiedzanie węzła 332
 odwołanie w przód 158, 161
 ograniczenia 161
 odwracanie listy 279, 289
 odwrotność
 modulo m 44
 permutacji 110, 181–184, 188, 581
 odwzorowanie drzewowe 406
 Oettinger, Anthony Gervin 479
 Office of Naval Research xiv, 238
- ograniczenie dolne: wartość mniejsza od dowolnego elementu w zbiorze
 ograniczenie górne: wartość większa od dowolnego elementu w zbiorze
 ojciec, w drzewie 323
 Okada, Satio (岡田幸雄, późniejszej 岡田幸千生) 608
 Oldenburg, Henry 60
 Oldham, Jeffrey David xv
 Onodera, Rikio (小野寺力夫) 608
 operacja
 przypisania (\leftarrow) 3
 wymiany (\leftrightarrow) 3, 188, 284
 zastąpienia (\leftarrow) 3
 zmiennopozycyjna 135
 operacje komputera MIX, zob. MIX, rozkazy
 operator logiczny, bitowy 473
 opóźnienie 236, 476
 oprogramowanie, przenoszenie 209
 optymalna procedura poszukiwania 418
 Oresme, Nicole 24
 ORIG (wartość początkowa) 151, 156, 160
 Orlin, James Berger 610
 ortogonalne ciągi permutacji 191
 osiowanie macierzy 313–316, 318
 ostatni-wchodzi-pierwszy-wychodzi 248, 471, 478
 prawie 465, 467, 474
 oszukiwanie 245, 608
 Otoo, Ekow Joseph 585
 Otter, Richard Robert 411, 615
 OUT (wyjście) 141, 233
 OVERFLOW, zob. PRZEPEŁNIENIE
 oznaczenia 651
 ósemkowe 648
- Pallo, Jean Marcel 603
 pamięć: część systemu komputerowego
 przeznaczona na przechowywanie danych 130
 bębnowa 140–142, 476
 dyskowa 140–142, 236, 454, 482
 hierarchia 205, 438, 454, 482
 komórka 132
 mapa 453–454
 rodzaje 246
 tymczasowa 197
 uaktualnianie synchroniczne 308
 wielopoziomowa 205
 zwalnianie 265, 268, 302, 429–431, 437, 456–460, 462, 471–475
 parametry podprogramów 193, 195, 237
 Parker, Douglass Stott, Jr. 623
 Parmelee, Richard Paine 468
 Pascal, Blaise 18, 56
 pasjans 392–393
 Patashnik, Oren 12, 658
 Patt, Yale Nance 530
 Pawlak, Zdzisław 479
 PDP-4, komputer 128

- Peck, John Edward L. 480
 Peirce, Charles Santiago Sanders 620
 Penrose, Roger 613
 Perlis, Alan Jay 334, 479
 permanent 54
 permutacja 47–48, 53, 101, 169–191, 250–252
 ciągi ortogonalne 191
 cykl jednoelementowy 170, 177, 186–187
 cykle 170–172, 182–183, 188–190
 identycznościowa 170, 181
 notacja cyklowa 170–172, 178–181, 188
 notacja dwuwierszowa 169, 188
 odwrotna 110, 181–184, 188, 581
 punkt stały 170, 186
 stosowa 250–252, 343
 w miejscu 9, 170, 191, 545
 permutacje
 generowanie 48
 mnożenie 171–172, 178–179, 386
 składanie, *zob.* permutacje, mnożenie
 Petkovšek, Marko 67
 Petolino, Joseph Anthony, Jr. 538
 pętla w grafie skierowanym: krawędź
 prowadziła od wierzchołka do
 niego samego 387
 pętla w programie: kod powodujący
 wielokrotne wykonanie tych samych
 instrukcji
 Pfaff, Johann Friedrich 506
 Pflug, Georg Christian 463
 Philco S2000, komputer 128
 pierścień buforów 225–235, 239
 pierwiastek 113
 pierwiastek liczby 23, 27
 pierwiastki z jedności 93
 pierwszy-wchodzi-pierwszy-wychodzi 248, 364, 478, 635, *zob.* także kolejka
 prawie 465, 467
 Pilot ACE, komputer 238
 Pingala, Āchārya (आचार्य पिङ्गल) 56
 PL/I, język 450–452
 PL/MIX język 162
 Poblete Olivares, Patricio Vicente 545
 pochodna 93, 351
 początek kolejki 250
 poddrzewo 320
 lewe 324, 330
 prawe 324, 330
 średni rozmiar 421
 wolne 379
 o najmniejszym koszcie 385
 zliczanie 393–394
 zliczanie 393–394
 zorientowane 379
 zliczanie 393
 podłoga $\lfloor x \rfloor$ 41–43
 podobieństwo
 drzew binarnych 340–342
 lasów 359
 podprogram 163, 192–204, 213–214, 218, 278, 289–290, 300–302, 356
 argumenty 193, 195
 historia 236–238
 o wielu wejściach 194
 o wielu wyjściach 196
 otwarty 236
 sekwencka wywołania 193–196, 199, 202–203
 wywołanie rekursywne 197
 podstawienie 3
 podstawowe twierdzenie arytmetyki 44
 podwójna funkcja tworząca: funkcja
 tworząca będąca funkcją dwóch
 zmiennych 98, 412, 421, 560, 562
 podział wielokąta 424
 podziały 13, 35, 90, 96
 funkcja tworząca 97
 liczby całkowitej 13, 35, 96
 zbioru 77, 502
 Poincaré, Jules Henri 512
 Poirot, Hercule (Poirot, Herkules) xxi
 Poisson, Siméon Denis 109, 547
 pokrycie
 kafelkami 398–401
 tetraedryczne 398–401
 toroidalne 400
 pola
 elementu struktury danych 241–245
 notacja 243–245, 477
 słowa komputera MIX 131–132, 146, 148, 213
 w wierszu języka MIXAL 150, 157
 pole: wydzielony fragment struktury
 danych, zazwyczaj złożony z ciągu
 przyległych symboli, *zob.* także wycinek
 131–132, 146, 148, 213
 adresu w wierszu języka MIXAL 156–157
 OP wiersza języka MIXAL 160
 Polonsky, Ivan Paul 480
 Pólya, György (= George) 18, 97, 411–412, 422–423, 517
 porównywalność 279
 porządek: relacja dwuargumentowa
 na elementach zbioru mająca
 pewne specjalne własności (m.in.
 przechodniość)
 częściowy 270–271, 279–280, 359,
 587, 600
 dynastyczny 349, *zob.* porządek preorder
 inorder 331–332; 336, 343–345, 359
 leksykograficzny 22, 310–311, 317
 liniowy 21, 271, 279
 na drzewach 345, 359
 na drzewach binarnych 345
 podwójny 344, 346, 589

- porządek
 postorder
 w drzewie 348–353, 359, 361, 479
 binarnym 331, 334, 343–345,
 348–353, 359, 361, 479
 ze stopniami 364, 375
 potrójny 593
 poziomy 364, 589, 603
 reprezentacja sekwencyjna 373
 preorder
 w drzewie 348–350, 359, 361, 479
 binarnym 331, 334, 343–345, 359
 ze stopniami 373, 479
 prepostorder 593
 rodzinny 363, 603
 sukcesji 349
 symetryczny 332, 334, 343–345
 wierszowy 164, 188, 310
 postorder, *zob.* porządek postorder
 PostScript 209
 postulat Bertranda 533
 potęga liczby 23
 obliczanie 530
 potęgi
 kroczące 52, 113, *zob.* potęgi
 przyrastające, potęgi ubywające
 obliczanie 530
 przyrastające 52, 55, 69, 71, 74, 113, 652
 ubywające 52, 55, 113, 652
 potok 204
 potomek, w drzewie 323
 Poupard, Yves 625
 poziom węzła w drzewie 320, 328–329, 392
 półniezmienniki rozkładu prawdopodobieństwa 107–109
 Pratt, Vaughan Ronald 47, 563, 619
 prawa działań na potęgach 23, 27, 55
 prawe poddrzewo 324, 330
 prawo
 dualności 71
 Kirchhoffa, pierwsze 100, 176, 278,
 288, 378–385
 przemienności 171
 rozdzielności 29, 38, 44
 subaddytywności 644
 preorder, *zob.* porządek preorder
 prepostorder, *zob.* porządek prepostorder
 Prim, Robert Clay 385
 Prinz, Dietrich G. 238
 problem
 głosowania 560–561
 inwersji 66, 408, 566, 603, 621
 Józefa Flawiusza 168
 rezerwacji miejsc 474
 procedura
 czytania tej książki xvi–xviii, 9
 ładująca 148
 poszukiwania, optymalna 418
 procent składany 25
 proces
 interakcyjny 5
 wstępujący 321, 364, 375, 626
 zchodzący 321, 375, 626
 Prodinger, Helmut 527
 profil programu 150, 176, 221, 307, 551
 program: precyzyjny opis metody
 obliczeniowej w pewnym języku
 formalnym
 ładujący 148, 281–282
 monitorujący, *zob.* program śledzący
 obliczania liczb pierwszych 156
 obsługi windy 292–309
 odporny na błędy 280
 projektowanie 197–199
 przebieg 204–206
 śledzący 198, 218–221, 307
 śledzący skoki 221, 551
 wieloprzebiegowy 208
 programowanie czytelne 199
 projektowanie programu 197–199
 Prüfer, Ernst Paul Heinz 423
 przechodniość 270, 367
 przechodzenie drzewa 348, 479
 binarnego 331, 336
 w porządku
 inorder 332
 postorder 344–345, 348–353,
 359, 361, 479
 preorder 344
 przechowywanie tymczasowe, *zob.* pamięć
 tymczasowa
 przedział, notacja 23
 przekątne wielokątów 424
 przekształcanie wyrażeń algebraicznych
 478–481
 PRZEPEŁNIENIE 253–260, 265–268, 278
 przepis 6
 przerwanie 235
 przesunięcie cykliczne 140, 191
 przeszukiwanie
 w głąb 604, 607
 wszerz 364
 przetwarzanie
 napisów 480
 quasi-równoległe 307
 przewidywanie 232
 przodek, w drzewie 323, 361
 przybliżenie Stirlinga 53, 118–120
 przycisk
 GO 218
 START 130, 146, 148
 przydział bufora (ASSIGN) 226–230,
 233–235
 przydzielanie pamięci: wybieranie komórek
 pamięci do przechowywania danych,
 zob. lista wolnej pamięci, dynamiczne

- przydzielanie pamięci, sekwencyjne
- przydzielanie pamięci dynamiczne 254–263, 265–268, 429–431, 453–475
- metoda
 - bloków bliźniaczych 460–462, 466–467, 473–475, 481
 - kolejnego dopasowania 467, 472, 646
 - najdawniej używany 470
 - najgorszego dopasowania 471, 636
 - najlepszego dopasowania 454–455, 465, 471–475
 - pierwszego dopasowania 454–456, 471–475
 - znaczników brzegowych 472, 481
- powtórne 439
- sekwencyjne 252
- przypisanie (\leftarrow) 3
- przystawanie 42–44
- pseudooperator: konstrukcja w języku programowania służąca do sterowania komplikacją 151
- punkt
 - osobliwy funkcji 412
 - siodłowy 164
 - stały permutacji 170, 186
- Purdom, Paul Walton, Jr. 467–468
- Rahman, Mizanur (মিজানুর রহমান) 510
- Ramanan, Prakash Viriyur (பிரகாஷ் விரியூர் ரமணன்) 561
- Ramanujan Iyengar, Srinivasa (ஸ்ரீநிவாஸ் ராமானுஜன் ஜயங்காரி) 14, 124, 126
- Ramshaw, Lyle Harold 508
- Ramus, Christian 74
- Randell, Brian 208, 468
- Raney, George Neal 408–410, 620
- Raphael, Bertram 479
- Raz, Yoav (רֻאָבֶן רָז) 558
- RCA 601, komputer 128
- Read, Ronald Cedric 590
- realokacja: powtórne przydzielanie pamięci, *zob. także* kompresowanie pamięci
- realokacja struktur tablicowych 255–259, 476
- Recomp II 128
- Reeves, Colin Morrison 463
- referencja 241, *zob. dowiązanie*
- regularny graf skierowany 394
- reguła
 - łańcuchowa 54
 - pięćdziesięciu procent 462–463, 465, 467
- Reingold, Edward Martin 25, 540
- rejestr komputera MIX
 - A 129
 - I 129, 146
 - indeksowy 129, 131
 - J 129, 147, 192, 195, 218–221
 - X 129
- rejestry: elementy sprzętu komputerowego służące do przechowywania danych, zapewniające bardzo szybki dostęp do zapamiętywanych danych indeksowe 163, 275
- zapamiętywanie i odtwarzanie zawartości 194, 204, 220, 235
- rekord: spójny fragment danych w pamięci, *zob. także* element struktury danych 241
- grupowanie 225, 233
- relacja: własność zachodząca dla pewnych zestawów (zazwyczaj par) elementów. Na przykład „ $<$ ” jest relacją określoną dla uporządkowanych par liczb całkowitych (x, y) ; własność „ $x < y$ ” zachodzi wtedy i tylko wtedy, gdy x jest mniejsze od y
- antysymetryczna 270
- asymetryczna 271
- przechodnia: *zob. porządek* 112, 270, 367
- przeciwwrotna 271
- rekurencyjna: reguła definiująca wyraz ciągu za pomocą wyrazów występujących wcześniej
- równoważności 366–369, 507
- symetryczna 367
- zwrotna 270, 367
- Rényi, Alfréd 622
- reprezentacja
 - drzewa binarnego 330, 334, 340, 346–347
 - kanoniczna drzwea zorientowanego 406–410, 413–414, 617–618
 - kolejki 252–261, 263, 268–270, 279, 283–284, 299
 - kolejki dwustronnej 260, 290, 307
 - lasów 347
 - macierzy 163–164, 309–319
 - metody wyboru 246, 441
 - stosu 252–263, 267, 337, 345, 433
 - tablicowa drzew 361–365, 373–376, 450
 - w porządku
 - poziomym 364, 373
 - preorder 362, 375
 - rodzinnym 363–364, 603
 - Listy 425–429, 438
 - wielomianów 285–286, 290, 369
 - wyrażeń algebraicznych 350
 - reszta 41
 - rezerwacja wolnej pamięci 265–268, 302, 454–456, 471–475
 - Ribenboim, Paulo 485
 - Rice, Stephan Oswald 590
 - Richmond, Lawrence Bruce 590
 - Riemann, Georg Friedrich Bernhard 79
 - funkcja dzeta $\zeta(s)$ 46, 79
 - Riordan, John 413, 515, 622
 - Ritchie, Dennis MacAlistair 480
 - RLINK (dowiązanie do prawego)
 - 290–292, 298
 - w drzewie 351, 361–366, 373, 395

- RLINK**
 w drzewie binarnym 330, 335, 340,
 346, 478
 w Listach 425, 428
- RLINKT** 337
- Robinson, Raphael Mitchel 613
- Robson, John Michael 467, 471, 475,
 593, 632, 644
- Rodrigues, Benjamin Olinde 423
- rodzeństwo, w drzewie 323
- rodzic, w drzewie 323, 329, 347–348
 sfastrygowany 592
- Roes, Piet Bernard Marie 104
- Rogers, Leonard James 511
- Rokicki, Tomas Gerhard 209
- Rosenberg, Arnold Leonard 584
- Rosenstiehl, Pierre 252
- Ross, Douglas Taylor 468, 477, 481
- Rothe, Heinrich August 65, 506
- Rousseau, Cecil Clyde 528
- rozkaz języka maszynowego: kod
 interpretowany przez sprzęt
 komputerowy jako rozkaz wykonania
 pewnej akcji
- rozkazy komputera MIX 131–149
 arytmetyczne 135–137, 214
 konwersji 142
 ładowania wartości do
 pamięci 215
 rejestru 133, 146, 215
 porównania 138, 217
 przekazania adresu 216
 przesunięć 139, 218
 skoku 139, 215
 wejścia 140–142
 wyjścia 140–142
 zapisywania wartości w pamięci 134
 zestawienie 144
 zmiennopozycyjne 135, 318
- rozkład na czynniki pierwsze 44
- rozkład prawdopodobieństwa: funkcja
 określona na zdarzeniach losowych
 opisująca prawdopodobieństwo
 otrzymania poszczególnych wartości
 zmiennej losowej 101–110
 dwumianowy 104–105
 nierówności ogonowe 110
 ujemny 110
 jednostajny 105, 262, 464
 normalny 107, 126
 odchylenie standardowe 101
 przybliżony 109
 Poissona 109, 547
 nierówności ogonowe 523
 wariancja 101, 103–107, 190
 wartość średnia (oczekiwana) 100–101
- rozmiar bajta na maszynie MIX: liczba
 różnych wartości, które można
 przechowywać w bajcie
- rozmiar elementu struktury danych
 266, 310, 453, 471
- rozmiar słowa na maszynie MIX: liczba
 różnych wartości, które można
 przechowywać w pięciu bajtach
- rozmiar węzła 453, 471
- rozszerzenie komputera MIX 148, 235,
 260–261, 473
- rozszerzone drzewo binarne 415–422
- rozoważania językowe 222
- równania rekurencyjne 90–92
 liniowe 86, 91
- równoległość 307
- równości jednokierunkowe 111
- równoważnik symbolu języka MIXAL 161
- równoważność
 drzew binarnych 340–342
 lasów 359
 List 440
 różniczka symetryczna 578
 różniczkowanie 93, 351–360, 478
 RTAG 335, 345–346, 351, 362–364, 373, 395
 Russell, David Lewis 641
 Russell, Lawford John 208
 rytmika 56, 83
 rzut monetą 104–105
 nierówności ogonowe 110
- Salton, Gerard Anton 364, 479
- Sammet, Jean Elaine 481, 600
- Satterthwaite, Edwin Hallowell, Jr. 238
- Schäffer, Alejandro Alberto 536
- Schatzoff, Martin 468
- schemat
 blokowy xvi–xviii, 2–3, 9, 16–19,
 378–379
 drzewa 321–327, 350, 360, 362, 479
 drzewa binarnego 324, 330, 588
 Listy 327–329, 425
- schematy 289
- Scherk, Heinrich Ferdinand 510
- Schlatter, Charles Fordemwalt 478
- Schlatter, William Joseph 478
- Scholten, Carel Steven 239, 633
- Schoor, Amir (אמיר שור) 584
- Schorr, Herbert 434, 439
- Schreiber, Peter 83
- Schreier, Otto 401
- Schröder, Friedrich Wilhelm Karl Ernst 619
- Schwartz, Eugene Sidney 420
- Schwartz, Karl Hermann Amandus 38
- Schwenk, Allen John 516
- SCOPE 363, 376, 451
- Segner, Johann Andreas von 423, 559
- Seki, Takakazu (関孝和) 116, 119
- Selfridge, John Lewis 81
- semafony 239
- sfastrygowane drzewo binarne 334,
 344, 348, 479
- Sha, Jichang (沙基昌) 571

- Shakespeare (= Shakspere), William
240, 485
- Shaw, John Clifford 238, 477
- Shephard, Geoffrey Colin 399, 614
- Shepp, Lawrence Alan 189–190
- Shor, Peter Williston 536
- Shore, John Edward 463, 468
- Shylock 485
- SICOMP: SIAM Journal on Computing*
– czasopismo wydawane przez
Society for Industrial and Applied
Mathematics od 1972 r.
- sieci PERT 270–271
- sieć: graf opisany dodatkowymi danymi,
takimi jak wagi krawędzi lub
wierzchołków
szeregowo-równoległa 615
- silna spójność 387, 392
- silnia 48–55
związek z funkcją gamma 51
- Simon, Herbert Alexander 238, 477
- Simonovits, Miklós 526
- SIMSCRIPT 481
- SIMULA I 237
- Singh, Parmanand (परमानंद सिंह) 83
- singleton, zob. cykl jednoelementowy
- siodło, zob. punkt siodłowy
- siostra, w drzewie 323
- skierowany graf krawędziowy 394
- SLA (przesuń w lewo rA) 139, 552
- SLAX (przesuń w lewo rAX) 139, 552
- SLC (przesuń cyklicznie w lewo rAX)
139, 552
- SLIP 479–480
- Sloane, Neil James Alexander 622
- słabnia 50, 53
- słowa Fibonacciego 89
- słownik 1–2
- Słownik Webstera 223
- słowo: fragment pamięci komputera
dający się zaadresować
białe 173
komputera MIX 129–131, 515
- SNOBOL 480
- SODA: Proceedings of the ACM-SIAM
Symposia on Discrete Algorithms* –
materiały wydawane od 1990 r.
- Soria, Michèle 522
- sortowanie topologiczne 270–280,
359, 391, 413
- Speedcoding 238
- Spieß, Jürgen 94
- splot rozkładów prawdopodobieństwa:
rozkład zmiennej losowej będącej
sumą dwóch niezależnych zmiennych
losowych 106
- spójność 376
silna 387, 392
- SRA (przesuń w prawo rA) 139, 552
- SRA (przesuń w prawo rAX) 139, 552
- SRC (przesuń cyklicznie w prawo rAX)
139, 552
- ST1 (zapisz rI1 do pamięci) 134, 215
- STA (zapisz rA do pamięci) 134, 215
- stała
Eulera γ 78, 118, 647
pamięciowa 154, 161
Stirlinga σ 118
- stałe w programie asemblerowym 154, 161
- stan
obsługi przerwania 235
stacjonarny 396–397
- Stanford University ii, xiv, 307, 579
- Stanley, Richard Peter 423, 619, 625
- Staudt, Karl Georg Christian von 422
- Stearns, Richard Edwin 483
- Steele, Guy Lewis, Jr. 633
- Steffens, Elisabeth Francisca Maria 633
- Steffensen, Johan Frederik 524
- sterta 266, 453, zob. także lista AVAIL
- Stevenson, Francis Robert 605
- Stickelberger, Ludwig 53
- Stigler, Stephen Mack 467, 468
- Stirling, James 49–51, 71, 76, 90, 118, 186
liczby 68–72, 74–77, 81, 102–104,
527, 608
prawo dualności 71
- stała σ 118
wzór aproksymacyjny 53, 118–120
- STJ (zapisz rJ w pamięci) 135, 193, 215
- STOC: Proceedings of the ACM Symposia
on Theory of Computing* materiały
wydawane od 1969 r.
- Stolarsky, Kenneth Barry 516
- stopień węzła w drzewie 320, 329, 392
wejściowy (wyjściowy) 386, 392
- stos 247–252, 332–339, 365, 375, 432,
439, 445–446, 477–478
- dno 249
wierzchołek 249–250
- wkładanie elementu na 250, 252–253, 255,
263, 267, 278, 283–284, 288, 477
- z dowiązaniami 267, 278, 280, 283–284,
345, 433
- zdejmowanie elementu 250, 252–255,
262–263, 268, 278, 283–284, 288, 477
- tablicowy 252–253, 255–263, 337
- Strong, Hovey Raymond, Jr. 584
- stronicowanie 470
- Struik, Dirk Jan 60, 501, 517
- struktura danych: zbiór elementów
powiązanych zależnościami
- struktura danych dla zbiorów rozłącznych
368, 374
- struktury
danych 240–484
drzewa 240, 320–440
Lista 327–328, 424–440, 479–481

- struktury
 - danych
 - lista liniowa 242–319
 - listy ortogonalne 309–319
 - piersienniowe 369
 - reprezentacja
 - graficzna 242
 - w pamięci 242, 246, 441–450, 481
 - wskaźnikowe, *zob.* struktury
 - z dowiązaniami 263
- z dowiązaniami 263–265
 - drzewa 347, 365–369
 - filozofia 264
 - historia 477–480
 - lista liniowa 273–275, 279, 476–478
 - tablice 242, 312–318
 - z wieloma dowiązaniami 298–299, 369, 441–452, 477
- tablicowe a struktury z dowiązaniami 263–265, 309, 450
- strzałka (używana do oznaczenia dowiązań) 242
- Stuart, Alan 104
- STX (zapiss rX do pamięci) 134, 215
- STZ (zapiss zero do pamięci) 135, 215
- SUB (odejmowanie) 135–136, 214
- subaddytywność 644
- Subi, Carlos Samuel 550
- sufit $\lceil x \rceil$ 41, 43
- suma
 - ciągu arytmetycznego 33, 50, 59
 - geometrycznego 33, 91
- nieskończona 60
- potęg 119
- z wieloma zmiennymi 35–38
- sumowanie 28–41
 - a całkowanie 115–120
 - przez części 46, 79–81
 - wielokrotne 35–38
 - wzór Eulera 115–120, 123, 126
 - zmiana kolejności 30, 35–36, 45
- supremum, *zob.* kres górny 38
- Suri, Subhash (सुभाष सूरी) 536
- Sutherland, Ivan Edward 478
- Swainson, William 343
- Swift, Charles James 239
- Swift, Jonathan 656
- sygnalizacja świetlna 167
- Sylvester, James Joseph 423, 492, 609, 612
- symbol
 - definiowany 158
 - Legendre'a 47
 - lokalny 155–156, 162
- symboliczny język maszynowy, *zob.* asembler
 - symetryczna 367
- symulacja: naśladowanie pewnego procesu 463
- ciągła 292
- symulacja
 - dyskretna 210, 292–309
 - synchroniczna 292
 - komputerów 209–210
 - samego siebie 218–221
- symulator komputera MIX 210–218
- syn, w drzewie 323
- synchroniczna symulacja dyskretna 292
- synchroniczne uaktualnianie pamięci 308
- System/360, komputery 128, 195
- system: zbiór obiektów lub procesów połączonych lub współpracujących ze sobą
- system bloków bliźniaczych 460–462, 466–467, 473–475, 481
 - dla liczb Fibonacciego 474
- system dziesiętny Dewey'a 325, 329, 343, 359, 397–398, 421, 479
- system liczbowy: język do reprezentowania liczb
 - dwójkowy (binarny) 26–28, 128
 - dziesiętny 22–23, 128, 647
 - Fibonacciego 89, 516
 - kombinatoryczny 76, 585
 - ósemkowy 648
 - o mieszanych podstawach 310
 - o niewymiernej podstawie 90
 - ϕ 89, 516
- szachy 6, 200, 282
- Szekeres, George 622
- szereg
 - harmoniczny 166
 - hipergeometryczny, podstawy 510
- szereg nieskończony: suma nieskończonie wielu składników
- szereg potęgowy: suma postaci $\sum_{k \geq 0} a_k z^k$, *zob.* funkcja tworząca
 - logarytmiczny 94
 - przekształcenia 122
 - zbieżność 90, 412
 - zbieżność bezwzględna 31
- szeregi
 - nieskończone 29–31, 60
 - rozbieżne 78
- szkic programu, *zob.* profil
- Szpilrajn, Edward 278
- szyfr 165
- ścieżka
 - długość 415–422
 - średnia 421
 - drzewa binarnego 594
 - krytyczna 224
 - losowa 396, *zob.* łańcuch Markowa
 - podstawowa 382
 - prosta 377, 383
 - w grafie 377
 - wewnętrzna, długość 418, 421
 - zewnętrzna, długość 416, 421

- ścieżka
 zorientowana 386
 prosta 386
 ślad wykonania 220, 307, 551
 śledzenie skoków 307
- tabele
 wartości liczbowych 56, 68
 wielkości numerycznych 647
- tablica: prostokątna struktura danych,
 czasem wielowymiarowa 4, 163–164,
 309–319
 dwuwymiarowa: *zob.* macierz
 implementacja
 kolejki 252–261, 263
 listy liniowej 252–263
 stosu 252–253, 255–263, 337
 jako drzewo 327
 jednowymiarowa: *zob.* lista liniowa
 niezainicjowana 319
 przełączająca 211–212, 215–216, 552
 rzadka, sztuczka 319
 sekwencyjne 163–164, 309–312
 z dowiązaniami 242, 312–318
- Tamaki, Jeanne Keiko (玉置惠子) 623
- Tamari, Dov 603
 krata 603, 625
- Tarjan, Robert Endre 252, 607
- taśma
 magnetyczna 140–142, 482
 papierowa 140–142, 237, 239
- technologia potokowa 551
- Temme, Nicolaas Maria 68, 69, 125
- teoria
 automatów 237, 248, 482–484
 liczb 42–47
 terminologia 47, 248, 323, 376, 453
- tetraedryczne pokrycie kafelkami 398–401
- TeX vi, xv, 209, 639, 679
- Thiele, Thorvald Nicolai 107
- Thorelli, Lars-Erik 631, 642
- Thornton, Charles 334, 479
- tocjent $\varphi(n)$ 44, 190
- Todd, John 494
- Tonge, Frederic McLanahan, Jr. 480
- Torelli, Gabriele 74, 508
- toroidalne pokrycie kafelkami 400
- tożsamości Newtona 517
- transformata Laplace'a 98
- transpozycja: (1) permutacja polegająca
 na zamianie miejscami dokładnie
 dwóch elementów; (2) operacja
 zamiany wierszy macierzy w kolumny
 i na odwóz
- bloków danych 191
- macierzy prostokątnej 188
- permutacji 188, 386
- triangulacja wielokąta 424, 626–628
- Tricomi, Francesco Giacomo Filippo
 125–126
- Trilling, Laurent 480
- trit 146
- Tritter, Alan Levi 602
- trójkąt Pascala 56, 72, 74, 76, 88, 520,
zob. także współczynniki dwumianowe
- Tucker, Alan Curtiss 421
- Turing, Alan Mathison 19, 199,
 237–238, 478
- Tutte, William Thomas 609
- Twain, Mark (= Clemens, Samuel
 Langhorne) 57
- twierdzenie
 dwumianowe 59–60, 94
 Abela 74, 414
 Hurwitza 46, 415, 509
 uogólnione 73, 94
- Eulera 44
- Fermata 43
- graniczne Abela 99
- macierzowe o drzewach 393, 612
- q -mianowe 76, 515
- wielomianowe 68
- Wilsona 53
- Twigg, David William 545
- Tyżrób, Idzi 114
- uaktualnianie pamięci, synchroniczne 308
- Uhler, Horace Scudder 500
- ujemny: mniejszy od zera
- ujemny rozkład dwumianowy 110
- układ VLSI 589
- Ullman, Jeffrey David 585
- ułamki
 łańcuchowe 519
 proste 65, 75, 86
- UNDERFLOW, *zob.* NIEDOMIAR
- UNIVAC 1107, komputer 128
- UNIVAC I, komputer 156, 237, 500
- UNIVAC III, komputer 128
- UNIVAC SS80, komputer 128
- UNIX 204
- upakowanie 132
- uruchamianie programu 198–199,
 208, 267, 308
- urządzenia wejścia-wyjścia 140
- Uspensky, Vladimir Andreevich (Успенский,
 Владимир Андреевич; Uspinski
 Władimir Andriejewicz) 483
- usuwanie
 błędów z programu 198–199, 208, 308
 drzewa binarnego z fastrygą prawostronną
 346
- elementu
 z drzewa 372
 z dwukierunkowej struktury
 pierścieniowej 372

- usuwanie
 elementu
 z kolejki 250, 253, 263, 270, 275,
 283–284
 dwustronnej 260, 307
 z listy 244
 dwukierunkowej 291, 301, 307
 dwuwymiarowej 316
 liniowej 247
 z dowiązaniem 264, 286, 316
 ze stosu, *zob.* zdejmowanie elementu
 ze stosu
 listy liniowej 283–284
 Listy 429
- van Aardenne-Ehrenfest, Tatyana 390, 394
 van der Waerden, Bartel Leendert 401, 614
 van Leeuwen, Jan 623
 van Wijngaarden, Adriaan 480
 Vandermonde, Alexandre Théophile 62, 73
 macierz 39–40, 495
 Vardi, Ilan 525
 Vauvenargues, Luc de Clapiers, Marquis
 de xviii
 Velthuis, Frans Jozef 679
 von Ettingshausen, Andreas 56
 von Neumann, John (= Margittai Neumann
 János) 19, 237, 476
 von Segner, Johann Andreas 423, 559
 von Staudt, Karl Georg Christian 422
- W-wartość 159–160
 Wadler, Philip Lee 633
 Waerden, Bartel Leendert van der 401, 614
 Waite, William McCastline 434, 439, 642
 Wall, Hubert Stanley 501
 Wallis, John 24, 54
 iloczyn 35
 Wang, Hao (王浩) 398–400
 wariancja
 funkcji tworzącej 103–106
 rozkładu prawdopodobieństwa 101,
 103–107, 190
 Waring, Edward 81, 492
 Warren, Don W. 373
 wartość asymptotyczna: funkcja opisująca
 zachowanie wielkości liczbowych przy
 przejściu do granicy
 pochodna 110–127, 251, 412, 547, 590
 wartość oczekiwana, *zob.* wartość średnia
 wartość średnia (oczekiwana) 100–101
 z funkcji tworzącej 103–106
 wartownik: specjalna wartość umieszczana
 w strukturze danych, łatwa do
 rozpoznania przez program wykonujący
 operacje na strukturze 224, 285, 593
 warunek wstępny, najsłabszy 19
 Watanabe, Masatoshi (渡邊雅俊) 679
 Watson, Dan Caldwell 260
- Watson, George Neville 528
 lemat 127
 Watson, Henry William 398
 wcięcia w tekście 324
 wczytywanie danych 222
 Weber, Helmut 480
 Wedderburn, Joseph Henry Maclagan 615
 Wegbreit, Eliot Ben 642
 Wegner, Peter 317
 Weierstrass, Karl Theodor Wilhelm 397
 Weiland, Richard Joel 636
 Weizenbaum, Joseph 430, 478–480
 wejścia do podprogramów 192–197
 wejście 221–236
 wejście-wyjście (we-wy) 222
 wektor, *zob.* lista liniowa
 wektory ortogonalne permutacji 191
 węzeł 241, 376, 386
 odwiedzanie 332
 wewnętrzny 320, 416–422
 zewnętrzny 320, 416–421
 węzły o różnych rozmiarach 453–475
 Wheeler, David John 237–238, 476
 Whinihan, Michael James 89
 Whirlwind I, komputer 237
 Whitworth, William Allen 188
 wiele wyjść 278
 Wielkanoc, wyznaczanie daty 165
 wielkie- O 111–115, 122
 wielkie- Ω 114–115
 wielkie- Θ 114
 wielokrotność: x jest wielokrotnością y , jeśli
 $x = ky$ dla pewnego całkowitego k
 wielomian
 Bernoulliego 46, 116–119
 charakterystyczny macierzy 520
 wielomiany 58–59, 67, 69, 71, 73, 111
 dodawanie 285–290, 369–372
 mnożenie 287, 290, 375
 reprezentacja 285–286, 290, 369
 różnica 67
 wierzchołek 376, 386
 izolowany 389
 końcowy 386
 początkowy 386
 stosu 249–250
 wierzchołki sąsiednie 376
 Wijngaarden, Adriaan van 480
 Wiktor z Akwitani 540
 Wilde, Oscar Fingal O'Flahertie Wills 438
 Wiles, Andrew John 485
 Wilf, Herbert Saul 67–69, 97, 504
 Wilkes, Maurice Vincent 237–238, 476
 Wilson, John
 twierdzenie 53
 Wilson, Paul Robinson 471
 Windley, Peter F. 545
 Windsor, House of 323
 Winkler, Phyllis Astrid Benson xv

- Wirth, Niklaus Emil 480
 Wise, David Stephen 260, 438, 452, 633
 Wiseman, Neil Ernest 438
 Wiśniewska-Walczyk, Anna xxi
 wkładanie elementu na stos 250, 252–253,
 255, 263, 267, 278, 283–284, 288, 477
 własność
 A 612
 stopu 18, 21–22, 401
 włożenie drzewa 361, 401
 włożenie porządku częściowego w porządek
 liniowy: zob. sortowanie topologiczne
 Wolman, Eric 471
 wolne poddrzewo 379
 Wolontis, Vidar Michael 238
 Woods, M. L. 238
 Woodward, Philip Mayne 479
 Wordsworth, William 143
 Wrench, John William, Jr. 500, 649
 Wright, Edward Maitland 513, 542
 Wright, Jesse Bowdle 373
 wskaźnik 241, zob. dowiązanie
 porównania 130, 146, 217, 220, 235
 stosu 252, 267
 wodzący 635
 wskaźnikowe drzewo binarne, zob. drzewo
 binarne z dowiązaniem
 wspólne bufory 231, 235
 współczynniki
 dwumianowe 55–77, 92
 asymptotyka 75
 definicja 55
 funkcja tworząca 94, 98
 interpretacja kombinatoryczna 76
 ograniczenia 77
 suma 88, 99
 tabela 56
 uogólnione 68, 75
 w sumach 58–77, 79–81, 88
 fibomianowe 88, 520
 q -mianowe 68, 76, 511
 wielomianowe 68, 410
 uogólnione 88
 współprogramy 200–207, 229–230,
 293–307, 332
 aktywowanie 200, 206, 230, 302
 historia 237
 wstawianie elementu do
 czterokierunkowego drzewa binarnego
 346
 dwukierunkowej struktury pierścieniowej
 371
 kolejki 250, 252–253, 263, 269,
 275, 283–284
 dwustronnej 260, 307
 listy
 dwukierunkowej 291, 301, 307
 dwuwymiarowej 316
 liniowej 247
 wstawianie
 elementu do
 listy z dowiązaniemi 243, 264, 286, 316
 sfastrygowanego drzewa binarnego
 339, 345
 węzła do drzewa 339, 345–346
 wyciąganie współczynnika 95
 wycinek (fragment słowa) 146, 148, 213
 wydajność odśmiecania 437
 wyjście: miejsce, w którym sterowanie
 opuszcza kod podprogramu 221–236
 wyjście z podprogramu 192–197
 Wyman, Max 68
 wyrażenia algebraiczne 325
 przekształcanie 478–481
 reprezentacja 350
 różniczkowanie 93, 351–360, 478
 wyrażenie arytmetyczne: zob. wyrażenie
 algebraiczne
 wyrażenie warunkowe 479, 652
 wysokość drzewa lub lasu 590
 wyszukiwanie błędów 267
 Wythoff (= Wijthoff), Willem Abraham 83
 wywołanie: aktywowanie (uruchomienie)
 podprogramu
 wywołanie podprogramu 192–196, 199,
 202–203, 237
 wyznacznik macierzy kwadratowej 39–41,
 84, 393–394, 397
 wzajemność 46–47
 kwadratowa 47
 wzór
 aproksymacyjny Stirlinga 53, 118–120
 dwumianowy Abela 75–76
 fryzowy 423–424
 inwersyjny Lagrange'a 408, 621
 Newtona 59
 sumacyjny Eulera 115–120, 123, 126
 Taylora z resztą 120
 X-1 239
 XDS 920, komputer 128
 XOR (exclusive or) 473
 Yao, Andrew Chi-Chih (姚期智) 567
 Yngve, Victor Huse 480
 Yoder, Michael Franz 498
 Young, David Monaghan, Jr. 612
 Zabell, Sandy Lew 512
 zamiana zmiennych sumowania 30, 34
 zamortyzowany czas wykonania 263
 zanurzenie zbioru częściowo uporządkowanego 271
 zaokrąglanie 43, 86, 166, 189
 zapis
 cyklowy, kanoniczny 184–185
 przedrostkowy, zob. notacja przedrostkowa
 zapisywanie danych 222

- zasada
odwracania 620, 625
symetrii 559
włączania i wyłączania 187
zachowania przepływu 176
Zave, Derek Alan 94, 642
zbieżność: ciąg nieskończony jest zbieżny,
jeśli wartości jego wyrazów zbliżają
się do granicy przy n rosnącym
do nieskończoności; mówimy, że
suma nieskończonego szeregu lub
produkту „jest zbieżna” (lub po prostu
„istnieje”), jeśli wg reguł analizy
matematycznej ma dobrze określoną
wartość; zob. równanie 1.2.3–(3)
bezwzględna 31
szeregu potęgowego 90, 412
zbiory zagnieździone 324, 329
zbiór częściowo uporządkowany,
zanurzenie 271
zdejmowanie elementu ze stosu 250,
252–255, 262–263, 268, 278,
283–284, 288, 477
zegar
czasu rzeczywistego 236
symulowany 293, 298
Zeilberger, Doron (זילברגר דורון) 67
Zemanek (= Zemánek), Heinz 2
· Zhang, Linbo (张林波) 679
- Zimmerman, Seth 424
zliczanie drzew 393–394, 401–415, 422–424
historia 422–424
poddzew 393–394
zliczanie poddrzew zorientowanych 393
złota proporcja 14, 20, 23, 83, 647
złoty podział, zob. złota proporcja
zmienna: obiekt mogący zmieniać
swoją wartość podczas wykonania
programu 3–4, 243
dowiązaniowa 242–244
indeksowa 29
wskaźnikowa, zob. dowiązaniowa 242
znacznik, zob. także wartownik
brzegowy 459–460, 472, 481
przepełnienia 130, 139, 146, 215, 235
znaczniki 429–431
znak alfanumeryczny: litera, cyfra lub
znak specjalny
zorientowane drzewo binarne 412
Zorn, Max 571
lemat 571
zrównoważony graf skierowany 389
zupełne drzewo binarne 417, 421
Zuse, Konrad 476
zwalnianie bufora (RELEASE) 226–230,
233–235
pamięci 265, 268, 302, 429–431, 437,
456–460, 462, 471–475

*Nie należy... sądzić, że obliczanie,
czyli rozumowanie,
dotyczy jedynie liczb.*

— THOMAS HOBBES, *Elementary Philosophy* (1656)

Oryginał tej książki został złożony na Sun SPARCstation z użyciem krojów pisma rodzinny Computer Modern. Wykorzystano do tych celów systemy TeX i METAFONT, opisane w tomach A–E dzieła Donald'a E. Knutha pt. *Computers & Typesetting* (Reading, Mass.: Addison-Wesley 1986). Do wykonania ilustracji użyto systemu METAPOST Johna Hobby'ego. Niektóre nazwiska w skorowidzu zostały złożone w oryginalnej pisowni, czcionkami opracowanymi przez Yannisa Haralambousa (greckie, hebrajskie, arabskie), Olgę G. Lapko (cyrylicą), Fransa J. Velthuisa (w dewanagari), Masatoshiego Watanabe (japońskie) i Limbo Zhanga (chińskie).

Kod znaku:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
□	A	B	C	D	E	F	G	H	I	Δ	J	K	L	M	N	O	P	Q	R	Σ	Π	S	T	U

00	1	01	2	02	2	03	10
Nie rób nic NOP(0)		rA \leftarrow rA + V ADD(0:5) FADD(6)		rA \leftarrow rA - V SUB(0:5) FSUB(6)		rAX \leftarrow rA \times V MUL(0:5) FMUL(6)	
08	2	09	2	10	2	11	2
rA \leftarrow V LDA(0:5)		rI1 \leftarrow V LD1(0:5)		rI2 \leftarrow V LD2(0:5)		rI3 \leftarrow V LD3(0:5)	
16	2	17	2	18	2	19	2
rA \leftarrow -V LDAN(0:5)		rI1 \leftarrow -V LD1N(0:5)		rI2 \leftarrow -V LD2N(0:5)		rI3 \leftarrow -V LD3N(0:5)	
24	2	25	2	26	2	27	2
M(F) \leftarrow rA STA(0:5)		M(F) \leftarrow rI1 ST1(0:5)		M(F) \leftarrow rI2 ST2(0:5)		M(F) \leftarrow rI3 ST3(0:5)	
32	2	33	2	34	1	35	1 + T
M(F) \leftarrow rJ STJ(0:2)		M(F) \leftarrow 0 STZ(0:5)		Urządzenie F zajęte? JBUS(0)		Sterowanie urządzeniem F IOC(0)	
40	1	41	1	42	1	43	1
rA : 0, skok JA[+]		rI1 : 0, skok J1[+]		rI2 : 0, skok J2[+]		rI3 : 0, skok J3[+]	
48	1	49	1	50	1	51	1
rA \leftarrow [rA]? \pm M INCA(0) DECA(1) ENTA(2) ENNA(3)		rI1 \leftarrow [rI1]? \pm M INC1(0) DEC1(1) ENT1(2) ENN1(3)		rI2 \leftarrow [rI2]? \pm M INC2(0) DEC2(1) ENT2(2) ENN2(3)		rI3 \leftarrow [rI3]? \pm M INC3(0) DEC3(1) ENT3(2) ENN3(3)	
56	2	57	2	58	2	59	2
CI \leftarrow rA(F) : V CMPA(0:5) FCMP(6)		CI \leftarrow rI1(F) : V CMP1(0:5)		CI \leftarrow rI2(F) : V CMP2(0:5)		CI \leftarrow rI3(F) : V CMP3(0:5)	

Legenda:

C = kod operacji, wycinek (5 : 5)

F = modyfikator operacji, wycinek (4 : 4)

M = adres po indeksowaniu

V = M(F) = zawartość wycinka F lokacji M

OP = nazwa symboliczna operacji

(F) = standardowa wartość F

t = czas wykonania; T = czas wstrzymania

C	t
Opis	
OP(F)	

25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
V	W	X	Y	Z	0	1	2	3	4	5	6	7	8	9	.	,	()	+	-	*	/	=	\$	<	>	@	;	:	'

04	12	05	10	06	2	07	1 + 2F
rA \leftarrow rAX/V rX \leftarrow reszta DIV(0:5) FDIV(6)		Specjalne NUM(0) CHAR(1) HLT(2)		Przesuń o M bajtów SLA(0) SRA(1) SLAX(2) SRAX(3) SLC(4) SRC(5)		Kopiuj F słów z M do rI1 MOVE(1)	
12	2	13	2	14	2	15	2
rI4 \leftarrow V LD4(0:5)		rI5 \leftarrow V LD5(0:5)		rI6 \leftarrow V LD6(0:5)		rX \leftarrow V LDX(0:5)	
20	2	21	2	22	2	23	2
rI4 \leftarrow -V LD4N(0:5)		rI5 \leftarrow -V LD5N(0:5)		rI6 \leftarrow -V LD6N(0:5)		rX \leftarrow -V LDXN(0:5)	
28	2	29	2	30	2	31	2
M(F) \leftarrow rI4 ST4(0:5)		M(F) \leftarrow rI5 ST5(0:5)		M(F) \leftarrow rI6 ST6(0:5)		M(F) \leftarrow rX STX(0:5)	
36	1 + T	37	1 + T	38	1	39	1
Wejście, urządzenie F IN(0)		Wyjście, urządzenie F OUT(0)		Urządzenie F gotowe? JRED(0)		Skoki JMP(0) JSJ(1) JOV(2) JNOV(3) też [*] poniżej	
44	1	45	1	46	1	47	1
rI4 : 0, skok J4[+]		rI5 : 0, skok J5[+]		rI6 : 0, skok J6[+]		rX : 0, skok JX[+]	
52	1	53	1	54	1	55	1
rI4 \leftarrow [rI4]? \pm M INC4(0) DEC4(1) ENT4(2) ENN4(3)		rI5 \leftarrow [rI5]? \pm M INC5(0) DEC5(1) ENT5(2) ENN5(3)		rI6 \leftarrow [rI6]? \pm M INC6(0) DEC6(1) ENT6(2) ENN6(3)		rX \leftarrow [rX]? \pm M INCX(0) DECX(1) ENTX(2) ENNX(3)	
60	2	61	2	62	2	63	2
CI \leftarrow rI4(F) : V CMP4(0:5)		CI \leftarrow rI5(F) : V CMP5(0:5)		CI \leftarrow rI6(F) : V CMP6(0:5)		CI \leftarrow rX(F) : V CMPX(0:5)	

[*] : [+]:

rA = rejestr A
rX = rejestr X
rAX = połączone rA i rX
rIi = rejestr indeksowy i , $1 \leq i \leq 6$
rJ = rejestr J
CI = wskaźnik porównania

JL(4) < N(0)
JE(5) = Z(1)
JG(6) > P(2)
JGE(7) \geq NN(3)
JNE(8) \neq NZ(4)
JLE(9) \leq NP(5)