UNIWERSYTET RZESZOWSKI

Wydział Matematyczno-Przyrodniczy

Zakład Informatyki Instytutu Matematyki

INFORMATYCZNE ROZWIĄZYWANIE ZADAŃ I PROBLEMÓW

SKRYPT DLA STUDENTÓW

STUDIÓW DZIENNYCH i PODYPLOMOWYCH

Krzysztof Balicki **Jan G. Bazan**

Rzeszów, 4 października 2003

Spis treści

Rozdział 1

Wprowadzenie

1.1 Narodziny heurystyki

Od zarania dziejów człowiek musiał rozwiązywać wiele problemów pojawiających się podczas jego codziennej działalności. Według Georgy Polya (patrz [12]), w realnym świecie *problem* pojawia się wówczas, gdy zachodzi potrzeba świadomego poszukiwania środka, za pomocą którego można osiągnąć dobrze widoczny, lecz chwilowo niedostępny cel. Tak więc *sformułować problem* to znaczy sprecyzować cel do jakiego się dąży. Natomiast *rozwiązać problem* to tyle, co znaleźć środek do osiągnięcia tego celu. Problem z którego rozwiązaniem wiążą się znaczne trudności nazywamy *problemem trudnym*. Natomiast problem jest *łatwy* lub *błahy*, gdy jego rozwiązanie otrzymuje się bez trudu.

Umiejętność rozwiązywania problemów, jako szczególnie cenna własność ludzkiego umysłu, już od starożytności była badana i rozwijana. W starożytnej Grecji sztuką rozwiązywania problemów zajmowali między innymi: Euklides, Apolloniusz z Pergi oraz Aristaeus Starszy. Ich pracę podsumował, usystematyzował i rozwinął Pappus z Aleksandrii, matematyk żyjący na przełomie III i IV wieku (patrz [3]).

W języku starogreckim słowo *heuriskein* oznaczało sztukę znajdowania rozwiązań problemów lub odkrywanie nowych metod rozwiązania. Dlatego dziedzinę zajmującą się metodami rozwiązywania problemów nazywamy *heurystyką*. W literaturze informatycznej termin *heurystyka* często oznacza także metodę rozwiązania problemu. Dlatego w niniejszej pracy tak właśnie będziemy go rozumieć. Natomiast dziedzinę zajmującą się metodami rozwiązywania problemów będziemy po prostu nazywać *sztuką rozwiązywania problemów*.

W czasach nowożytnych sztuką rozwiązywania problemów zajmowali się tacy filozofowie i matematycy jak: Rene Descartes (1596-1650), Gottfried Leibniz (1646-1716) oraz Bernard Bolzano (1781-1848).

Jednak za największy krok w rozwoju metod rozwiązywania problemów jest uważana praca pt. „Jak to rozwiązać?" (patrz [13]), opublikowana w 1945 roku przez Georgy Polya. Praca ta w zasadzie poświęcona jest nauczaniu matematyki, a dokładnie dotyczy nauczania myślenia. Jednak zawiera rozdział pt. „Krótki słownik heurystyczny", który zawiera podstawowe pojęcia dotyczące wykorzystywania istniejących i tworzenia nowych heurystyk. Sztuka rozwiązywania problemów w ujęciu Polya jest rozumiana jako formalizacją doświadczeń zebranych przy rozwiązywaniu zadań i obserwacji rozwiązywania zadań. Polya zwraca także uwagę, na cechy wspólne różnych metod rozwiązania.

1.2 Przykłady rzeczywistych problemów

Oto kilka przykładów problemów pochodzących ze świata rzeczywistego.

1. Dany jest plecak o objętości v oraz n przedmiotów ponumerowanych od 0 do $n-1$. Każdy przedmiot ma określoną wartość W_i i objętość V_i. Należy zapakować plecak spośród przedmiotów ponumerowanych od 0 do $n-1$ w taki sposób, aby wartość przedmiotów w nim zgromadzonych była największa.

2. Fabryka lodów każdego dnia produkuje na tej samej maszynie lody o sześciu różnych smakach. Zmiana produkcji ze smaku i na smak j wymaga przestrojenia maszyny, czyli przygotowania (w tym umycia) do nowej produkcji, które trwa określony czas. Dane są informacje o czasach przestrojenia maszyny. Znaleźć kolejność produkcji, minimalizującą całkowity czas przestrojenia maszyny. Przyjąć, że na koniec dnia maszyna ma być przygotowana do produkcji w następnym dniu.

3. Dane są dwa naczynia, które mogą pomieścić odpowiednio 3 litry i 4 litry wody. Naczynia nie mają miarki. Można je napełniać przy użyciu pompy lub przelewając wodę z jednego w drugie. W jaki sposób można otrzymać dokładnie 2 litry wody w naczyniu czterolitrowym?

4. Kopalnie A i B dostarczają węgiel do miast C, D i E. Kopalnia A dostarcza dziennie 500 ton, natomiast kopalnia B dostarcza dziennie 800 ton węgla. Miasta C, D i E zużywają odpowiednio 500, 400 i 400 ton węgla dziennie. Dany jest także koszt transportu, w dziesiątkach złotych, jednej tony węgla z każdej kopalni do poszczególnych miast. Jak należy zorganizować transport węgla, aby koszt był możliwie najmniejszy?

1.3 Przestrzeń stanów

Każdy problem dotyczy pewnego środowiska, w którym jest definiowany. Zwykle jest ono złożone z pewnej liczby obiektów wzajemnie ze sobą powiązanych. Część z nich ma wpływ na rozwiązanie rozpatrywanego problemu. Wiedza o tych obiektach, tzn. o ich własnościach, aktywnościach i związkach pomiędzy nimi, czyli inaczej o ich *konfiguracji*, określa nam aktualny *stan* środowiska z punktu widzenia rozpatrywanego problemu. Wszystkie pozostałe obiekty, które są nieistotne z punktu widzenia rozpatrywanego problemu nazywamy *tłem*. Na przykład środowiskiem problemu przelewania wody (patrz podrozdział A.27) są: naczynia, pompa wodna oraz człowiek napełniający naczynia. Oprócz tego można się zastanawiać, gdzie te obiekty się znajdują (np. w domu, na ulicy, w parku itd.). Jednakże z punktu widzenia samego rozwiązania problemu, istotnymi elementami środowiska są tylko same naczynia oraz operacje jakie na nich można wykonać. Nie jest na przykład istotne (jest tłem) to, gdzie się znajduje i jak działa pompa, kto nalewa wodę, czy jest to czysta woda, czy też zanieczyszczona itd.

Tak więc w chwili definiowania problemu jego środowisko pozostaje w pewnym stanie, określonym przez wiedzę na temat aktualnej konfiguracji tych obiektów, które są związane z problemem. Stan taki nazywamy *stanem początkowym*. W przykładzie dotyczącym napełnianych naczyń, stanem początkowym może być sytuacja, gdy obydwa naczynia są puste.

Warto także zauważyć, że stanów początkowych także może być wiele. Wszystkie cechują się jednak tym, że bardzo łatwo dają się skonstruować na podstawie opisu samego problemu. Jak już wspominaliśmy, w problemie z napełnianiem naczyń, stan początkowy może być taki, że obydwa naczynia są puste. Jednakże inne możliwe stany początkowe mogą wyglądać w ten sposób, że jedno z naczyń jest napełnione a drugie puste lub też obydwa naczynia są napełnione.

Oprócz stanu początkowego środowisko może mieć potencjalnie wiele innych stanów, które charakteryzują inne konfiguracje obiektów środowiska. Zbiór wszystkich stanów nazywamy *przestrzenią stanów* lub inaczej *przestrzenią przeszukiwania*.

Czynności wykonywane przy rozwiązywaniu problemów można zaklasyfikować jako poszukiwanie stanu środowiska o zadanej konfiguracji. Taki stan nazywamy *stanem końcowym*. Dla danego problemu może być jeden lub więcej stanów końcowych. Może się także zdarzyć, że nie ma stanu końcowego. Wtedy problem nie ma rozwiązania.

1.4 Różne podejścia do znajdowania stanu końcowego

Zatem rozwiązanie problemu może być wyznaczone poprzez przeszukiwanie całej przestrzeni stanów w celu znalezienia stanu końcowego (patrz podrozdział 2.1). Kompletne i mechaniczne przebadanie takiej przestrzeni jest zwykle mało efektywne i możliwe jedynie w przypadkach niewielkich przestrzeni stanów. Od najdawniejszych czasów uważano także, że może być ono bardzo nużące i czasochłonne dla człowieka. Po pojawieniu się komputerów okazało się także, że ze względu na dużą złożoność obliczeniową algorytmów przyglądających mechanicznie całe przestrzenie stanów badanych problemów, często w dalszym ciągu nie można efektywnie przeglądać całych przestrzeni stanów w celu znalezienia stanów końcowych.

Z drugiej strony dla wielu problemów przebadanie całej przestrzeni stanów jest niemożliwe, bo nie można dotrzeć do każdego stanu tej przestrzeni (nieskończona liczba stanów, zbyt duży koszt czasowy lub finansowy sprawdzania pojedynczego stanu itd.).

Dlatego od najdawniejszych czasów starano się konstruować takie metody przeszukiwania przestrzeni stanów, które nie wymagają mechanicznego przeszukiwania całej przestrzeni, ale tylko pewnej jej części lub też dzięki informacji o stanie początkowym oraz obliczeniom matematycznym, pozwalają na wskazanie stanów końcowych bez potrzeby przeglądania jakichkolwiek stanów pośrednich pomiędzy stanem początkowym a końcowym.

W celu konstruowania takich metod od dawna tworzono w naukach przyrodniczych *modele matematyczne* przestrzeni stanów badanego problemu. Obecnie w każdym praktycznie dziale nauk przyrodniczych mamy skonstruowane modele matematyczne. Na przykład w kinematyce są to wzory na prędkość i przyspieszenie, w dynamice są to zasady dynamiki, w grawitacji wzór na siłę przyciągania pomiędzy masami itd. Podstawowe wzory występujące w tych modelach nazywamy prawami fizycznymi, chemicznymi, biologicznymi itd. Modele te były tworzone przez słynnych obecnie naukowców minionego czasu, zwykle na podstawie genialnej matematycznej intuicji podpartej wieloma rzeczywistymi eksperymentami. Eksperymenty te nie były niczym innym, jak badaniem pewnych fragmentów przestrzeni przeszukiwania badanych problemów.

W niniejszym skrypcie zakładamy, że w skład modelu matematycznego przestrzeni stanów badanego problemu wchodzą następujące trzy elementy.

1. *Reprezentacja liczbowa stałych parametrów charakteryzujących ogólnie wszystkie stany* przestrzeni przeszukiwania. Reprezentacja ta jest zwykle realizowana za pomocą szeregu nazwanych stałych liczbowych, być może zgrupowanych w wektory lub macierze. (np. wartości stałych fizycznych lub chemicznych, normy dziennego żywienia itd.).

2. *Reprezentacja liczbowa parametrów charakteryzujących pojedynczy stan* przestrzeni przeszukiwania, przy czym wartości tych parametrów zmieniają się wraz ze zmianą opisywanego stanu. Dlatego realizacja takiej reprezentacji zwykle jest robiona za pomocą szeregu nazwanych zmiennych liczbowych, być może zgrupowanych w wektory lub macierze (np. upływający czas, prędkość ciała, masa ciała, temperatura ciała itd.)

3. *Reprezentacja za pomocą szeregu równań lub nierówności* w których po obu stronach występują funkcje rzeczywiste, *zależności pomiędzy występującymi w nich wymienionymi wyżej parametrami liczbowymi*. Zależności te mogą na przykład określać: do jakiego stanu jest

możliwe bezpośrednie przejście od ustalonego stanu przestrzeni przeszukiwania, czy możliwe jest przejście od jednego ustalonego stanu do innego, a jeśli tak, to w jakiej ilości kroków, dokąd można zajść w n krokach od danego stanu itd.

Jeśli dla danej przestrzeni przeszukiwania uda się utworzyć taki model matematyczny, to wykorzystując zależności pomiędzy liczbowymi parametrami stanów przestrzeni, można rozwiązać wiele problemów dotyczących badanej przestrzeni, tzn. za pomocą obliczeń matematycznych operujących na parametrach i przekształcających równania lub nierówności reprezentujące zależności pomiędzy parametrami, można wyznaczyć szukany stan końcowy związany z rozwiązywanym problemem.

Nie zawsze jednak można utworzyć model matematyczny badanej przestrzeni stanów, który umożliwi rozwiązanie problemu bez przeglądania przestrzeni stanów. Dlatego ważne jest sformułowanie takich metod przeglądania przestrzeni stanów, które będą efektywniejsze od metody polegającej na mechanicznym przeglądaniu całej przestrzeni stanów. Tego typu metody będą przedmiotem następnych rozdziałów skryptu. Teraz wspomnimy tylko jedną, dość często stosowaną metodą zwaną metodą „generowania i testowania". Polega ona na generowaniu kolejnych stanów według określonych zasad i badaniu ich własności, w celu wychwycenia stanów końcowych (patrz podrozdział 2.2). Zasady generowania obiektów nazywane są *operatorami przekształcającymi* lub *regułami produkcyjnymi*. Przykładowym operatorem przekształcającym w problemie przelewania wody (patrz podrozdział A.27) może być operator, z którym wiąże się nalanie wody do jednego z naczyń. Rozwiązanie tak postawionego problemu sprowadza się więc do określenia ciągu operatorów przekształcających stan początkowy w stan końcowy. Dla pojedynczego stanu jest zazwyczaj możliwe (choć nie zawsze) zastosowanie kilku operatorów. Aby jednak móc stosować operatory, trzeba je najpierw zdefiniować. W tym celu trzeba wiedzieć pomiędzy jakimi stanami jest możliwe bezpośrednie przejście oraz pomiędzy którymi stanami nie jest to możliwe. Wiedza ta wynika z opisu problemu i jest nierozerwalnie związana ze środowiskiem problemu.

1.5 Graf stanów

Grafem skierowanym G nazywamy uporządkowaną parę $G = (V, E)$ składającą się z niepustego, skończonego zbioru *wierzchołków* V oraz zbioru *krawędzi* $E \subset V \times V$ (patrz [20]).

Mówimy, że istnieje *droga* z wierzchołka $a \in V$ do wierzchołka $b \in V$, lub że wierzchołek b jest *osiągalny* z wierzchołka a, jeśli istnieje ciąg wierzchołków $v_1, ..., v_k$, taki że: $v_1 = a$, $v_k = b$ i dla każdego $i \in \{1, ..., k-1\}$ mamy: $(v_i, v_{i+1}) \in E$.

Na rysunku 1.1 przedstawiony jest przykładowy graf skierowany, dla którego:

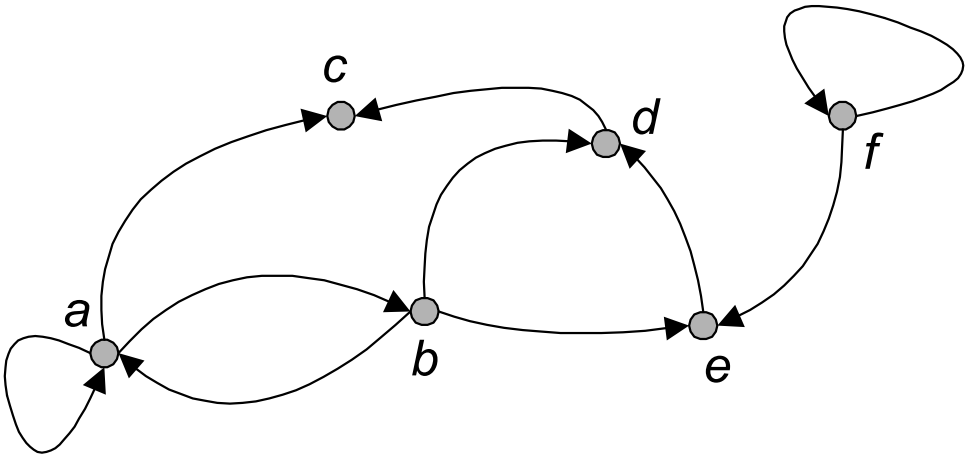
$V = \{a, b, c, d, e, f\}$	jest zbiorem wierzchołków,
$E = \{(a,a), (a,b), (b,a), (a,c), (d,c), (b,e), (e,d), (f,e), (f,f)\}$	jest zbiorem krawędzi.

Zauważmy, że w omawianym grafie wierzchołek c jest osiągalny z każdego innego wierzchołka, natomiast wierzchołek b nie jest osiągalny z wierzchołka f.

Środowisko problemu, określone przez przestrzeń stanów i operacje przejścia między stanami, można zamodelować przy pomocy *grafu skierowanego*.

Graf skierowany, który reprezentuje środowisko problemu będziemy nazywać *grafem stanów* albo *grafem przeszukiwania*.

W grafie przeszukiwania wierzchołki reprezentują stany a krawędzie reprezentują operacje przejścia między stanami.



Rys. 1.1: Przykład grafu skierowanego

1.6 Typy problemów

W nawiązaniu do wspomnianych już prac Polya (patrz [12] i [13]) w niniejszym skrypcie będziemy wyróżniać dwa następujące typy problemów.

- Problem typu „znajdź stan", którego celem jest wyszukanie pewnego stanu, spełniającego warunki występujące w określeniu danego problemu. Na przykład w decyzyjnym problemie plecakowym (patrz podrozdział A.1), chodzi o znalezienie stanu, w którym plecak zapakowany jest w optymalny sposób.

- Problem typu „znajdź drogę do stanu", którego rozwiązanie polega na przekształcaniu stanu początkowego na stan końcowy, być może z udziałem pewnych stanów pośrednich, przy czym interesuje nas tutaj nie sam stan końcowy, którego postać i własności są od początku znane, ale droga jaka wiedzie do niego poprzez inne stany od stanu początkowego. Wynika stąd, że w przypadku problemów typu „znajdź drogę do stanu" muszą być określone wspomniane wcześniej operatory przekształcające, które umożliwiają zmianę stanów środowiska z jednych na inne, czyli pozwalają na poruszanie się pomiędzy stanami.

Zauważmy, że tak naprawdę, to każdy problem typu „znajdź drogę do stanu" może być sprowadzony do problemu typu „znajdź stan". W tym celu na bazie pierwotnej dla danego problemu przestrzeni przeszukiwania należy tak skonstruować drugą przestrzeń przeszukiwania, aby pojedynczymi stanami tej nowej przestrzeni były drogi pomiędzy stanami w przestrzeni pierwotnej. Wtedy przeszukiwanie stanów drugiej przestrzeni jest jednocześnie przeszukiwaniem dróg pomiędzy stanami w pierwszej przestrzeni. Stąd problem typu „znajdź drogę do stanu" w pierwszej przestrzeni można traktować jako problem typu „znajdź stan" w drugiej przestrzeni. Podejście takie jest jednak dość niewygodne w użyciu. Dlatego wyodrębniliśmy osobny typ problemów do znajdowania dróg pomiędzy stanami i podamy stosowne metody do rozwiązywania tego typu problemów.

1.7 Komputer jako narzędzie do rozwiązywania problemów

W trakcie rozwoju cywilizacji okazało się, że umysł ludzki nie jest w stanie w wystarczająco krótkim czasie rozwiązać wielu problemów. Dlatego na świecie od dawna pojawiali się ludzie pra-

cujący nad skonstruowaniem maszyny, która mogłaby wykonywać pracę intelektualną. Jednak dopiero w latach 1944-1946 na świecie pojawiła się pierwsza generacja kalkulatorów elektronicznych, które można traktować jako pierwsze komputery. Były one ukoronowaniem długiej ewolucji intelektualnej i technicznej. Skonstruowane i wyprodukowane od tamtej pory kolejne generacje komputerów można już było w mniejszym lub większym stopniu wykorzystywać do wspomagania procesu rozwiązywania problemów.

Wszystkie dotychczas skonstruowane komputery łączy jedna cecha. Aby wykonać jakieś obliczenia lub inaczej dokonać analizy dostarczonych im danych, muszą być najpierw szczegółowo poinformowane, co mają robić, zanim zaczną wykonywać jakiekolwiek zadanie. Innymi słowy komputery są jedynie wykonawcami poleceń człowieka. Oznacza to, że w celu przeprowadzenia obliczeń na dostarczonych danych należy oprócz tych danych określić metodę, według której zostaną przeprowadzone obliczenia. Następnie opis tej metody oraz danych należy w wymaganej formie dostarczyć komputerowi. Dlatego aby można było wykorzystać komputer do rozwiązywania różnych problemów związanych z dostarczonymi mu danymi, na początku trzeba dokładnie opisać metodę analizy tych danych. Opis taki nazywamy *algorytmem* i rozumiemy go jako opis metody rozwiązania danego problemu w skończonej liczbie kroków zwanych operacjami elementarnymi. Przez *operacje elementarne* rozumiemy takie operacje, które z naszego punktu widzenia stanowią całość, tzn. nie interesuje nas, jakie akcje trzeba wykonać aby je wykonać. Ważne jest tylko to, aby każda z nich dobiegła końca.

Algorytmiczne reprezentacje metod rozwiązywania problemów noszą nazwę *strategii rozwiązywania problemów*. Z punktu widzenia wykorzystywania informacji o dziedzinie rozwiązywanego problemu, strategie rozwiązywania problemów mogą być:

1. **„ślepe”** - nie wykorzystujące informacji o dziedzinie rozwiązywanego problemu; cechują się one tym, że w poszukiwaniu stanu końcowego mechanicznie przeszukują całą przestrzeń stanów lub wybrany jej fragment,

2. **„inteligentne”** - wykorzystujące informacje o dziedzinie rozwiązywanego problemu (czasem są to informacje niepewne i nieprecyzyjne), które pozwalają na wybór odpowiedniej ścieżki podczas przeszukiwania; dzięki wykorzystaniu tych informacji strategie te zwykle są w stanie (choć nie zawsze) dość szybko znaleźć stan końcowy,

Inne kryterium podziału strategii rozwiązywania problemów wiąże się złożonością obliczeniową stosowanych w nich algorytmów. Dla wielu problemów złożoność obliczeniowa proponowanych strategii dokładnego (optymalnego) ich rozwiązania może być zbyt duża, aby można było użyć tych strategii w praktyce (zbyt długi czas obliczeń lub zbyt dużo użytej pamięci). Dlatego w praktycznych zastosowaniach często nie używa się metod, które dostarczają dokładnych (optymalnych) rozwiązań badanych problemów, ale zamiast nich stosujemy metody pozwalające uzyskać rozwiązania *prawie optymalne*, które na tyle dobrze spełniają stawiane wymagania, że można je z powodzeniem stosować w praktyce. Z tego punktu widzenia, metody (strategie) rozwiązywania problemów można podzielić na:

1. **dokładne** - pozwalają uzyskać dokładne czyli optymalne rozwiązanie badanych problemów,

2. **aproksymacyjne (przybliżone)** - prowadzą do znalezienia rozwiązania *prawie optymalnego*, czyli takiego, które nie jest wprawdzie rozwiązaniem optymalnym, ale jest zadawalające z punktu widzenia wymagań stawianych szukanemu rozwiązaniu w obliczu praktycznych zastosowań tego rozwiązania.

Aby móc później przekazać skonstruowany algorytm do wykonania przez komputer, musimy go zapisać w jakimś języku. Najlepiej, jeśli będzie to tzw. *język sformalizowany*, tzn. język

sztucznie stworzony drogą przyjęcia pewnych aksjomatów i definicji. Dla takiego języka podaje się alfabet i zbiór możliwych słów, ściśle określając zasady składni oraz jednoznacznie definiując reguły wyznaczające sens poszczególnych napisów. Języki naturalne, np. język polski, nie nadają się do dobrego zapisu algorytmów ze względu na dopuszczanie wielu znaczeń poszczególnych słów i całych zdań, dużą złożoność reguł składniowych i znaczeniowych, dopuszczanie wyjątków itd. Sformalizowany język służący do zapisu algorytmów nazywamy *językiem programowania*. Przykładami języków programowania są np: Pascal, C, C++, Java, Logo, Basic itd. Zapis algorytmu w danym języku programowania nazywamy *programem*.

Tak więc, aby rozwiązać dany problem za pomocą komputera najpierw należy wymyślić metodę jego rozwiązania. Następnie na podstawie znajomości tej metody trzeba skonstruować algorytm rozwiązujący problem, oraz zapisać go w postaci programu. Stąd podstawowe techniki konstruowania programów można traktować jako elementarne metody informatycznego rozwiązywania problemów. Jak wiadomo zaliczamy do nich:

- stosowanie zmiennych prostych oraz instrukcji przypisania,
- stosowanie instrukcji warunkowych pozwalających na podejmowanie decyzji podczas działania algorytmu,
- stosowanie instrukcji iteracyjnych (pętli) pozwalających na wielokrotne wykonywanie tych samych instrukcji w algorytmie,
- stosowanie zmiennych indeksowanych (tablic),
- stosowanie procedur (funkcji, metody w klasach) czyli podprogramów realizujących wydzielone zadania.

Na przestrzeni ostatnich kilkudziesięciu lat powstało bardzo wiele rozwiązujących różne problemy programów komputerowych. Zwykle są one rozprowadzane przez liczne firmy odpłatnie (oprogramowanie komercyjne), za niewielką opłatą (tzw. oprogramowanie sharewarowe) lub nieodpłatnie (tzw. oprogramowanie freewarowe). Dlatego obecnie wykorzystując komputer do rozwiązywania problemów często wcale nie musimy pisać programów, ale wykorzystujemy różnego typu gotowe oprogramowanie.

W niniejszym skrypcie zakładamy, że do rozwiązywania problemów za pomocą komputera będziemy używać następujących narzędzi programistycznych:

- arkusza kalkulacyjnego Microsoft Excel,
- systemu do obliczeń symbolicznych i numerycznych Mathematica,
- środowisk pozwalających tworzyć własne programy: JCreator lub Logo Komeniusz.

Przykłady algorytmów będziemy podawać w tradycyjnej, zbliżonej do języka naturalnego notacji, wzbogaconej o pewne elementy wzięte z języka programowania. Natomiast przykłady programów będą podawane w języku Java.

Rozdział 2

Metody dokładne

W tym rozdziale opisujemy takie metody informatycznego rozwiązywania problemów, które pozwalają uzyskać dokładne rozwiązanie badanych problemów. Efektem działania tych metod jest więc wskazanie jednego lub kilku konkretnych stanów pochodzących z przestrzeni stanów i odpowiadających specyfikacji stanu końcowego w tej przestrzeni. W zależności od stosowanych technik informatycznych i matematycznych, w niniejszej pracy, będziemy wyróżniać pięć następujących tego typu metod:

- metody „brutalnej siły",
- metody „generowania i testowania",
- metoda „dziel i zwyciężaj",
- programowanie dynamiczne,
- programowanie liniowe.

Powyższe metody omówimy w kolejnych podrozdziałach.

2.1 Metody „brutalnej siły"

W podrozdziale zaprezentujemy metody, których działanie polega na mechanicznym przeglądaniu wszystkich stanów należących do przestrzeni stanów. Na użytek niniejszego skryptu metody te będziemy nazywać *metodami „brutalnej siły"*, gdyż nie wykorzystują one zwykle żadnych informacji o strukturze przestrzenie stanów, a komputer wykorzystują jedynie jako szybkie narzędzie do „mało inteligentnego" przeszukiwania.

Dobra metoda „brutalnej siły" musi zagwarantować, aby każdy stan był sprawdzany dokładnie jeden raz. Oznacza to, że nie może być pominięty żaden ze stanów, gdyż w ten sposób podczas poszukiwań mógłby być pominięty poszukiwany stan końcowy. Oprócz tego żaden ze stanów nie powinien być sprawdzany wielokrotnie, gdyż tak sytuacja mogłaby znacznie spowolnić przeszukiwanie przestrzeni stanów. Zastosowanie tej metody wymaga więc ustalenia pewnego porządku zgodnie z którym będą przeglądane stany. Porządek przeglądania stanów zwykle jest ustalany zupełnie bez uwzględnienia ewentualnych reguł przechodzenia pomiędzy stanami (była o nich mowa w podrozdziale 1.3).

Reasumując, aby rozwiązać jakiś problem za pomocą metody „brutalnej siły" należy wykonać następujące kroki.

1. Zdefiniować przestrzeń stanów zawierającą wszystkie możliwe konfiguracje tych obiektów środowiska badanego problemu, od których zależy rozwiązanie tego problemu. Chodzi tutaj o taki sposób zdefiniowania przestrzeni stanów, który umożliwi późniejsze przeglądanie wszystkich stanów. Wraz z definicją przestrzeni stanów potrzebne jest wprowadzenie kodu za pomocą którego będą opisywane poszczególne stany.

2. Wyspecyfikować opis stanów końcowych, które mogą być zaakceptowane jako rozwiązanie problemu.

3. Określić porządek przeglądania wszystkich stanów gwarantujący dotarcie do każdego stanu dokładnie jeden raz.

4. Dokonać przeglądu wszystkich stanów wg ustalonego wcześniej porządku, w celu znalezienia stanu lub stanów końcowych, przy czym możliwe są trzy następujące sytuacje:

   - pierwsza, gdy chodzi o znalezienie tylko jednego stanu końcowego o zadanych własnościach (np. może chodzić o znalezienie jednej trójki liczb pitagorejskich - porównaj podrozdział A.3)
   - druga, gdy chodzi o wyszukanie wszystkich stanów końcowych (np. o wyszukanie wszystkich trójek liczb pitagorejskich - patrz podrozdział A.3),
   - trzecia, gdy chodzi o wyszukanie jednego, ale optymalnego pod jakimś względem stanu końcowego (np. w decyzyjnym problemie plecakowym chodzi o optymalne zapakowanie plecaka - patrz podrozdział A.1).

5. Znaleziony stan końcowy zinterpretować jako rozwiązanie problemu.

Ponieważ w metodach „brutalnej siły" zwykle nie uwzględnia się naturalnych operatorów przejścia pomiędzy stanami, to wydaje się, że nie bardzo nadają się one do rozwiązania problemów typu „znajdź drogę do stanu" a jedynie do problemów typu „znajdź stan" (patrz podrozdział 1.6). Jednak z teoretycznego punktu widzenia można stosować metody „brutalnej siły" do problemów typu „znajdź drogę do stanu". W tym celu na bazie pierwotnej dla danego problemu przestrzeni przeszukiwania należy tak skonstruować drugą przestrzeń przeszukiwania, aby pojedynczymi stanami tej nowej przestrzeni były drogi pomiędzy stanami w przestrzeni pierwotnej. Wtedy zastosowanie metody typu „brutalnej siły" do takiej przestrzeni stanów pozwoli na przeglądanie wszystkich możliwych dróg pomiędzy stanami i znalezienie poszukiwanej drogi. Metoda tak jest jednak dość niewygodna w użyciu i czasochłonna w działaniu. Dlatego do rozwiązywania problemów typu „znajdź drogę do stanu" wykorzystuje się zwykle metodę „generowania i testowania", którą opiszemy w następnym podrozdziale.

Z algorytmicznego punktu widzenia, często do przeglądania całej przestrzeni stanów danego problemu wystarczy zastosowanie instrukcji pętli (pojedynczej lub zagnieżdżonej). Dlatego strategie oparte na metodach „brutalnej siły" często realizujemy za pomocą tzw. *przeszukiwania iteracyjnego*. Na przykład dla rozwiązania problemu wyznaczenia wszystkich trójek pitagorejskich (patrz podrozdział A.3) wystarczy wykonać następujący algorytm.

```
Dla x := 1..20 wykonuj:
  Dla y := 1..20 wykonuj:
    Dla z := 1..20 wykonuj:
      Jeśli x*x + y*y = z*z, to wypisz trójkę: x,y,z
```

Oto program, który realizuje powyższy algorytm:

```
class Trojki
{
  public static void main(String[] args)
  {
    for (int x = 1; x <= 20; x++)
      for (int y = 1; y <= 20; y++)
        for (int z = 1; z <= 20; z++)
          if (x*x + y*y == z*z) System.out.println(x + " " + y + " " + z);
  }
}
```

Powyższy program wypisuje następujące trójki liczb:

```
3 4 5
5 12 13
6 8 10
7 24 25
8 15 17
9 12 15
10 24 26
12 16 20
15 20 25
18 24 30
20 21 29
```

Opisy stanów często można utożsamiać z pewnymi obiektami kombinatorycznymi. Chodzi tutaj na przykład o podzbiory zbioru n-elementowego, podzbiory z powtórzeniami zbioru n-elementowego, permutacje zbioru n-elementowego, kombinacje k-elementowe ze zbioru n-elementowego itd. Dlatego metody „brutalnej siły” wiążą się zwykle z zagadnieniem generowania i sprawdzania wszystkich elementów pewnej klasy obiektów kombinatorycznych (patrz np. [9], [15]). Dla przykładu omówimy zagadnienie generowania podzbiorów zbioru n-elementowego.

Każdy zbiór n-elementowy $X = \{x_1, ..., x_n\}$ ma dokładnie 2^n podzbiorów. Aby się o tym przekonać wystarczy zauważyć, że dowolnemu podzbiorowi $Y \subset X$ zbioru X można przyporządkować jednoznacznie ciąg binarny $b_1, b_2, ..., b_n$ określony następująco:

$$b_i = \begin{cases} 1 & \text{jeśli } x_i \in Y \\ 0 & \text{jeśli } x_i \notin Y \end{cases}$$

Jeśli element x_i należy do zbioru Y, to odpowiadający mu wyraz b_i ciągu binarnego ma wartość 1. Jeśli element x_i nie należy do zbioru Y, to odpowiadający mu wyraz b_i ciągu binarnego ma wartość 0.

Z drugiej strony, dowolnemu n-elementowemu ciągowi binarnemu $b_1, b_2, ..., b_n$ można przyporządkować jednoznacznie podzbiór $Y \subset X$ zbioru X określony następująco:

$$x_i \in Y \Leftrightarrow b_i = 1$$

Element x_i należy do zbioru Y wtedy i tylko wtedy, gdy odpowiadający mu wyraz b_i ciągu binarnego ma wartość 1.

Wobec powyższego, pomiędzy wszystkimi podzbiorami n-elementowego zbioru X a wszystkimi n-elementowymi ciągami binarnymi istnieje wzajemnie jednoznaczna odpowiedniość. Skądinąd wiadomo, że wszystkich n-elementowych ciągów binarnych jest 2^n, a zatem wszystkich podzbiorów zbioru n-elementowego jest również 2^n.

Ciąg binarny $b_1, b_2, ..., b_n$ może posłużyć do wygenerowania dowolnego podzbioru zbioru n-elementowego. Przy małej liczbie elementów zbioru X, wszystkie jego podzbiory można wygenerować przy pomocy n-krotnie zagnieżdżonej pętli `for`. W przypadku zbioru czteroelementowego otrzymujemy następujący algorytm:

```
Dla b1 := 0..1 wykonuj:
  Dla b2 := 0..1 wykonuj:
    Dla b3 := 0..1 wykonuj:
      Dla b4 := 0..1 wykonuj:
        Generuj podzbiór zakodowany jako {b1,b2,b3,b4}
```

Oto program, który realizuje powyższy algorytm:

```
class Podzbiory1
{
  public static void main(String[] args)
  {
    int b1,b2,b3,b4;
    for (b1 = 0; b1 <= 1; b1++)
      for (b2 = 0; b2 <= 1; b2++)
        for (b3 = 0; b3 <= 1; b3++)
          for (b4 = 0; b4 <= 1; b4++)
          {
            System.out.print("{ ");
            if (b1 == 1) System.out.print("1 ");
            if (b2 == 1) System.out.print("2 ");
            if (b3 == 1) System.out.print("3 ");
            if (b4 == 1) System.out.print("4 ");
            System.out.println("}");
          }
  }
}
```

Program ten wypisuje następujący tekst:

```
{ }
{ 4 }
{ 3 }
{ 3 4 }
{ 2 }
{ 2 4 }
{ 2 3 }
{ 2 3 4 }
{ 1 }
{ 1 4 }
{ 1 3 }
{ 1 3 4 }
{ 1 2 }
{ 1 2 4}
{ 1 2 3 }
{ 1 2 3 4 }
```

Aby wygenerować wszystkie podzbiory zbioru $X = \{1, 2, ..., n\}$, dla dowolnie ustalonego n, musimy wykorzystać inną metodę. Zauważmy, że każdy ciąg binarny $b_1, b_2, ..., b_n$ odpowiada wzajemnie jednoznacznie liczbie dwójkowej $b_1 b_2 ... b_n$ z przedziału od 0 do $2^n - 1$. Wszystkie liczby całkowite z przedziału od 0 do $2^n - 1$ możemy wygenerować zaczynając od 0 i dodając w każdym kolejnym kroku liczbę 1, a ich reprezentacje binarne określą wszystkie podzbiory zbioru n-elementowego. Otrzymujemy następujący algorytm:

```
1. Utwórz n+1 elementową tablicę liczb całkowitych.
2. Przypisz: s := 2^n.
3. Dla l = 0..s-1 wykonuj:
   a) generuj podzbiór określony przez liczbę dwójkową z tablicy,
   b) dodaj 1 do liczby dwójkowej z tablicy.
```

Oto program, który realizuje powyższy algorytm:

```
class Podzbiory2
{
  public static void main(String[] args)
  {
    final int N = 3; // moc zbioru
    int[] tab = new int[N+1];
    int s = (int)Math.pow(2,N);
    for (int l = 0; l < s; l++)
    {
      System.out.print("{ ");
      for (int i = 0; i < N; i++)
        if (tab[i] == 1) System.out.print(i+1 + " ");
      System.out.println("}");
      int i = 0;
      do
      {
        if (tab[i] == 1)
        {
          tab[i] = 0;
          i++;
        }
        else
        {
          tab[i] = 1;
          break;
        }
      } while (true);
    }
  }
}
```

Program ten wypisuje następujący tekst:

```
{ }
{ 1 }
{ 2 }
{ 1 2 }
{ 3 }
{ 1 3 }
{ 2 3 }
{ 1 2 3 }
```

Z problemem generowania podzbiorów spotykamy się w wielu praktycznych zagadnieniach. Na przykład w decyzyjnym problemie plecakowym (patrz podrozdział A.1) potencjalnymi rozwiązaniami problemu są podzbiory zbioru wszystkich przedmiotów pakowanych do plecaka. Podobnie w problemie doboru załogi statku kosmicznego (patrz podrozdział A.4) potencjalnymi rozwiązaniami są podzbiory zbioru kosmonautów.

Ćwiczenia

2.1.1 Użyj metody „brutalnej siły" do rozwiązania decyzyjnego problemu plecakowego (patrz podrozdział A.1).

2.1.2 Użyj metody „brutalnej siły" do rozwiązania ogólnego problemu plecakowego (patrz podrozdział A.2).

2.1.3 Jak za pomocą metody „brutalnej siły" rozwiązać problem wydawania reszty (patrz podrozdział A.28).

2.1.4 Metodą „brutalnej siły" rozwiązać problem doboru załogi statku kosmicznego (patrz podrozdział A.4).

2.1.5 Za pomocą metody „brutalnej siły" rozwiązać problem planowania zawartości zestawu paszowego (patrz podrozdział A.9).

2.2 Metoda „generowania i testowania"

Metoda „generowania i testowania" polega na generowaniu kolejnych stanów według określonych zasad i badanie ich własności, w celu wychwycenia stanów końcowych. Zasady generowania obiektów nazywane są *operatorami przekształcającymi* lub *regułami produkcyjnymi*. Rozwiązanie tak postawionego problemu sprowadza się więc do określenia ciągu operatorów przekształcających stan początkowy w stan końcowy. Dla pojedynczego stanu jest zazwyczaj możliwe (choć nie zawsze) zastosowanie kilku operatorów. Aby jednak móc stosować operatory, trzeba je najpierw zdefiniować. W tym celu trzeba wiedzieć pomiędzy jakimi stanami jest możliwe łagodne przejście oraz pomiędzy którymi stanami jest to niemożliwe. Wiedza ta wynika z opisu problemu i jest nierozerwalnie związana ze środowiskiem problemu.

Reasumując, aby rozwiązać jakiś problem za pomocą metody „generowania i testowania" należy wykonać następujące kroki.

1. Zdefiniować przestrzeń stanów zawierającą wszystkie możliwe konfiguracje tych obiektów środowiska badanego problemu, od których zależy rozwiązanie tego problemu. Być może mogą to być jakieś konfiguracje możliwe teoretycznie, ale praktycznie nigdy nie zdarzające się w środowisku. Jednak w przeciwieństwie do prezentowanych wcześniej metod „brutalnej siły”, chodzi tutaj o taki sposób zdefiniowania przestrzeni stanów, który nie wymaga wymienienia wszystkich stanów, gdyż w praktyce często bywa to bardzo kłopotliwe lub wręcz niemożliwe z powodu dużego rozmiaru tej przestrzeni. Wraz z definicją przestrzeni stanów potrzebne jest także wprowadzenie kodu za pomocą którego będą opisywane poszczególne stany.

2. Wyspecyfikować opis stanów początkowych, traktowanych jako stany wyjściowe do poszukiwania rozwiązania problemu, oraz wskazać jeden lub wiele konkretnych stanów początkowych.

3. Wyspecyfikować opis stanów końcowych, które mogą być zaakceptowane jako rozwiązanie problemu.

4. Wyspecyfikować zbiór operatorów (reguł produkcyjnych), które pozwalają na przemieszczenie się ze stanu do stanu.

5. Wybrać metodę wyboru operatorów podczas eksploracji przestrzeni stanów oraz zastosować wybraną metodę w celu znalezienia ścieżki od stanu początkowego do końcowego.

6. Stan końcowy zinterpretować jako rozwiązanie problemu.

W literaturze opisywane jest wiele metod wyboru operatorów podczas eksploracji przestrzeni stanów. Jedną z nich jest metoda oparta na tzw. strategii przeszukiwania *z powracaniem* (ang. *backtracking*). Polega ona na tym, że dla wybranego stanu generowany jest za pomocą jednego z operatorów tylko jeden tzw. *stan potomny* i jeżeli ten nowy stan można dalej rozszerzać przez stosowanie operatorów i nie jest on końcowy, to jest dalej rozszerzany (tylko ten jeden stan potomny). Gdy po kolejnych rozszerzeniach otrzymywany stan nie można już rozszerzać i nie jest on stanem końcowym, to wtedy w strategii z powracaniem następuje powrót do najbliższego przodka, dla którego możliwe jest wygenerowanie innych stanów potomnych. Takie podejście skłania do następującej implementacji rekurencyjnej, gdyż ocena własności danego stanu zależy ściśle od oceny stanu, który jest jego rodzicem.

```
procedura zPowracaniem(p: stan początkowy) -> sukces lub porażka
  1. Jeśli stan p jest stanem końcowym, to: zwróć(sukces).
  2. Dla wszystkich reguł, które można zastosować do stanu p
     wykonuj:
     a) Zastosuj kolejną regułę i wygeneruj nowy stan t,
     b) Jeśli stan t nie był jeszcze sprawdzany
        wywołaj rekurencyjnie: wynik := zPowracaniem(t),
     c) Jeśli wynik = sukces, to zwróć(sukces).
  3. Zwróć(porażka).
```

Powyższy algorytm dobry jest dla rozwiązywania problemów, które wymagają znalezienia jednego, dowolnego stanu końcowego. Jednak jeśli chodzi o znalezienie optymalnego pod względem pewnej miary stanu końcowego, to powyższy algorytm można przerobić następująco:

```
procedura zPowracaniemOpt(p: stan początkowy,
                          var o: stan optymalny
```

```
                          var m: miara jakości stanu optymalnego) -> sukces lub porażka
  1. Jeśli stan p jest stanem końcowym, to:
        jeśli miara(p) > m, to:
           przypisz o := p oraz m := miara(p).
  2. Dla wszystkich reguł, które można zastosować do stanu p
     wykonuj:
     a) Zastosuj kolejną regułę i wygeneruj nowy stan t,
     b) Jeśli stan t nie był jeszcze sprawdzany
        wywołaj rekurencyjnie: zPowracaniemOpt(t,o,m).
```

Zauważmy, że procedura `zPowracaniemOpt` pozwala na sprawdzenie tych wszystkich stanów, które są osiągalne ze stanu początkowego za pomocą operatorów przekształcających.

W powyższych procedurach łatwo można uzyskać możliwość zarejestrowania drogi w grafie prowadzącej od stanu początkowego do końcowego lub od stanu początkowego do optymalnego końcowego. Na przykład w procedurze `zPowracaniem` wystarczy tylko wykonać kilka modyfikacji, co zostało zrobione w poniższym algorytmie.

```
procedura zPowracaniem2(p: stan początkowy, d: Lista stanów) -> sukces lub porażka
  1. Dodaj p na końcu listy d.
  2. Jeśli stan p jest stanem końcowym, to:
        zwróć(sukces).
  3. Dla wszystkich reguł, które można zastosować do
     stanu p wykonuj:
     a) zastosuj kolejną regułę i wygeneruj nowy stan t,
     b) jeśli stan t nie był jeszcze sprawdzany
        wywołaj rekurencyjnie: wynik := zPowracaniem2(t,d),
     c) jeśli wynik = sukces, to zwróć(sukces).
  4. Usuń p z końca d oraz zwróć(porażka).
```

Metodę przeszukiwania z powracaniem można interpretować także jako budowę rozwiązania problemu w wielu krokach. W każdym kroku procesu wyszukiwania, jeśli rozszerzenie aktualnego rozwiązania częściowego nie jest możliwe, powraca się do krótszego rozwiązania częściowego i ponownie próbuje się je rozszerzać. W ten sposób można także przeglądać wszystkie możliwe rozwiązania problemów.

Dla przykładu, powyższą metodę przeszukiwania z powracaniem zastosujemy do rozwiązania problemu generowania wszystkich permutacji zbioru n-elementowego. Przypomnijmy, że permutacją zbioru n-elementowego X nazywamy dowolną wzajemnie jednoznaczną funkcję $f : X \to X$. Zwykle permutację określamy za pomocą tablicy o dwóch wierszach, z których każdy zawiera wszystkie elementy zbioru X, przy czym element $f(x)$ występuje pod elementem x. Dla przykładu permutację f zbioru $\{a, b, c, d\}$, taką że:

$$f(a) = d,\ f(b) = a,\ f(c) = c \text{ i } f(d) = b$$

zapisujemy następująco:

$$f = \begin{pmatrix} a & b & c & d \\ d & a & c & b \end{pmatrix}.$$

Jeśli ustalimy porządek elementów w górnym wierszu, to każdej permutacji odpowiada jednoznacznie ciąg zawarty w dolnym wierszu, np. dla permutacji f jest to $\langle d, a, c, b\rangle$. Dlatego też permutacją zbioru n-elementowego X będziemy nazywać dowolny n-elementowy ciąg różnowartościowy o wyrazach ze zbioru X.

Z permutacjami spotykamy się w wielu praktycznych problemach. Na przykład w problemie wyprodukowania lodów wszystkich smaków (patrz podrozdział A.5) rozwiązanie tego problemu jest permutacją zbioru smaków.

Stosując strategię z powracaniem, problem wypisywania wszystkich permutacji zbioru $\{1, ..., n\}$ można rozwiązać za pomocą następującej procedury.

```
procedura permutacje(l: lista elementów)
  Jeśli liczba elementów na liście l jest równa n, to:
     wypisz elementy z listy l,
  w przeciwnym razie, dla każdego elementu i ze zbioru {1,...,n},
  którego nie ma jeszcze na liście l wykonuj:
     a) dodaj element i na koniec listy l,
     b) wywołaj rekurencyjnie: permutacje(l),
     c) usuń element i z końca listy.
```

Oto program, który realizuje powyższy algorytm:

```
class Permutacje
{
  final static int N = 3; // permutacje n-elementowe
  private static int[] l = new int[N];
  public static void main(String[] args)
  {
    permutacje(0);
  }
  static void permutacje(int i)
  {
    if (i == N)
    {
      for(int j = 0; j < N; j++)
        System.out.print(l[j] + " ");
      System.out.println();
    }
    else
    {
      for (int j = 1; j <= N; j++)
      {
        int k;
        for (k = 0; k < i; k++)
          if (l[k] == j) break;
        if (k == i)
        {
          l[k] = j;
          permutacje(i+1);
        }
      }
    }
  }
}
```

Powyższy program wypisuje następujący tekst:

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

Ćwiczenia

2.2.1 Użyj strategii przeszukiwania *z powracaniem* do rozwiązania decyzyjnego problemu plecakowego (patrz podrozdział A.1). Zadanie rozwiązać przy pomocy kompilatora Java.

2.2.2 Użyj strategii przeszukiwania *z powracaniem* do rozwiązania ogólnego problemu plecakowego (patrz podrozdział A.2). Zadanie rozwiązać przy pomocy kompilatora Java.

2.2.3 Jak za pomocą strategii przeszukiwania *z powracaniem* rozwiązać problem wydawania reszty (patrz podrozdział A.28). Zadanie rozwiązać przy pomocy kompilatora Java.

2.2.4 Metodą przeszukiwania *z powracaniem* rozwiązać problem doboru załogi statku kosmicznego (patrz podrozdział A.4). Zadanie rozwiązać przy pomocy kompilatora Java.

2.2.5 Metodą przeszukiwania *z powracaniem* rozwiązać problem wyprodukowania lodów wszystkich smaków (patrz podrozdział A.5). Zadanie rozwiązać przy pomocy kompilatora Java.

2.2.6 Za pomocą metody przeszukiwania *z powracaniem* rozwiązać problem planowania zawartości zestawu paszowego (patrz podrozdział A.9). Zadanie rozwiązać przy pomocy kompilatora Java.

2.2.7 Za pomocą metody przeszukiwania *z powracaniem* rozwiązać problem przelewania wody (patrz podrozdział A.27). Zadanie rozwiązać przy pomocy kompilatora Java.

2.2.8 Za pomocą metody przeszukiwania *z powracaniem* rozwiązać problem przeprawy przez rzekę misjonarzy i ludożerców (patrz podrozdział A.29). Zadanie rozwiązać przy pomocy kompilatora Java.

2.2.9 Za pomocą metody przeszukiwania *z powracaniem* rozwiązać problem przeprawy przez rzekę wilka, kozy i kapusty (patrz podrozdział A.30). Zadanie rozwiązać przy pomocy kompilatora Java.

2.2.10 Za pomocą metody przeszukiwania *z powracaniem* rozwiązać problem syntezy związku chemicznego (patrz podrozdział A.31). Zadanie rozwiązać przy pomocy kompilatora Java.

2.3 Metoda „dziel i zwyciężaj”

Przy konstrukcji algorytmów rozwiązujących różne problemy, często bardzo efektywną metodą jest metoda „dziel i zwyciężaj”. Ogólnie, mówiąc stosowanie tej metody polega na tym, że zamiast rozwiązywać wejściowy problem, który może być problemem trudnym, próbuje się wskazać jeden lub więcej innych problemów, które są łatwiejsze do rozwiązania, a ich rozwiązanie pozwala na skonstruowanie rozwiązania problemu wejściowego.

Jedno z praktycznych podejść do rozwiązywania problemów metodą „dziel i zwyciężaj” jest takie, że wejściowy problem jest dzielony na kilka mniejszych podproblemów podobnych do początkowego problemu, ale mających mniejsze rozmiary. Następnie podproblemy te są rozwiązywane. Na koniec rozwiązania wszystkich podproblemów są łączone w celu utworzenia rozwiązania problemu wejściowego. Przy zastosowaniu rekurencji, omawiane podejście daje się zrealizować w następujący sposób:

```
procedura dziel_i_zwyciężaj(p : rozwiązywany problem)
  Jeśli rozmiar(p) < rozmiarProgowy to:
     rozwiązanieBezRekurencji(p),
  w przeciwnym razie:
     a) podziel problem p na mniejsze podproblemy: p1,p2,...,p_k.
     b) dla i := 1..k wykonuj:
        r_i := dziel_i_zwyciężaj(p_i),
     c) rozwiązanie_p := połączRozwiązania(r1,...,r_k),
     d) zwróć rozwiązanie_p.
```

Przy pomocy metody „dziel i zwyciężaj” można rozwiązać między innymi klasyczny problem przewidywania liczebności populacji królików, zwany problemem Fibonacciego (patrz podrozdział A.20). Rozwiązanie problemu Fibonacciego sprowadza się do wyznaczenia n-tego wyrazu ciągu Fibonacciego. Ciąg Fibonacciego definiuje się następująco:

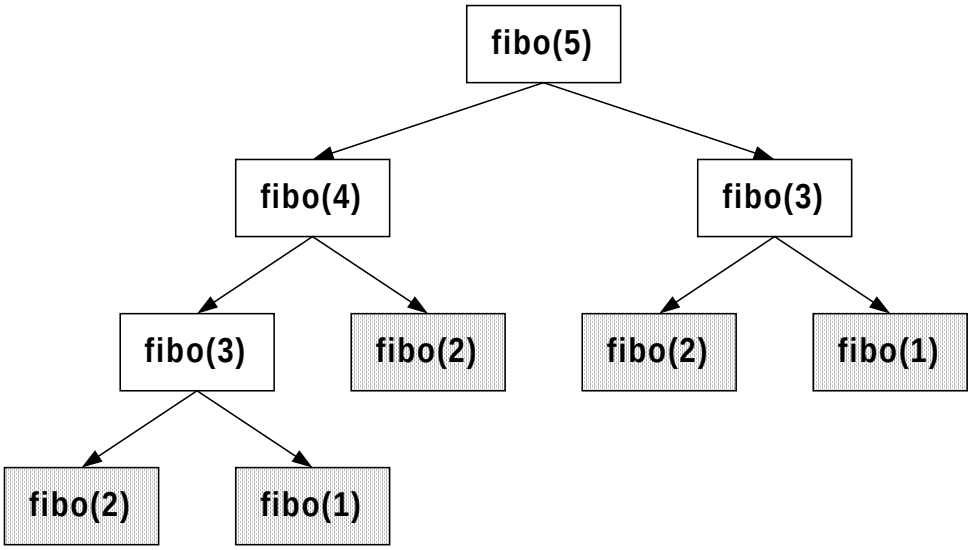
$$fibo(n) = \begin{cases} 1 & \text{dla } n = 1, \\ 1 & \text{dla } n = 2, \\ fibo(n-1) + fibo(n-2) & \text{dla } n \geq 3. \end{cases}$$

Pierwszy i drugi wyraz ciągu Fibonacciego ma wartość 1, a każdy kolejny wyraz ciągu jest sumą dwóch poprzednich wyrazów ciągu. Zwróćmy uwagę, że powyższa definicja odpowiada idei metody „dziel i zwyciężaj”. Aby wyznaczyć n-ty wyraz ciągu Fibonacciego należy najpierw wyznaczyć $n-1$ oraz $n-2$ wyraz ciągu, czyli rozwiązać podproblemy o mniejszym rozmiarze.

Oto program, który wyznacza n-ty wyraz ciągu Fibonacciego:

```
class FiboDZ
{
  public static void main(String[] args)
  {
    final int N = 5; // n-ty wyraz ciągu
    System.out.println(fibo(N));
  }
  static int fibo(int n)
  {
    if (n == 1 || n == 2)
      return 1;
    else
      return fibo(n-1) + fibo(n-2);
  }
}
```

Analiza wywołania metody `fibo(5)` prowadzi do grafu wywołań z rys. 2.1. Każdy liść (zakreskowany wierzchołek) tego grafu reprezentuje problem elementarny. Natomiast każdy inny



Rys. 2.1: Obliczanie `fibo(5)`

wierzchołek grafu reprezentuje problem złożony o rozmiarze $n \geq 3$, który dzielony jest na dwa podproblemy o rozmiarach $n-1$ i $n-2$. Zauważmy, że w powyższym algorytmie pewne obliczenia są powtarzane wielokrotnie (np. obliczenie `fibo(3)`). Aby uniknąć takiej sytuacji, do rozwiązania problemu Fibonacciego zamiast metody „dziel i zwyciężaj" należy zastosować pokrewną metodę *programowania dynamicznego* (patrz podrozdział 2.4).

Niektóre problemy wymagają rozwiązania na danym etapie obliczeń tylko jednego podproblemu o mniejszym rozmiarze. Dla przykładu, aby wyznaczyć wartość $n!$ wystarczy wyznaczyć wartość $(n-1)!$ i pomnożyć ją przez n.

Wyraża to poniższa procedura:

```
procedura dziel_i_zwyciężaj(p : rozwiązywany problem)
  Jeśli rozmiar(p) < rozmiarProgowy, to:
     rozwiązanieBezRekurencji(p),
  w przeciwnym razie:
     a) wyodrębnij z problemu p pewien podproblem pp o mniejszym
        rozmiarze, którego rozwiązanie umożliwi rozwiązanie problemu p,
     b) rozwiązanie_pp := dziel_i_zwyciężaj(pp),
     c) rozwiąż problem p na podstawie rozwiązania rozwiązanie_pp
        podproblemu pp i umieść rozwiązanie w rozwiązanie_p,
     d) Zwróć rozwiązanie_p.
```

Oto program, który oblicza wartość n silnia:

```
class SilniaDZ
{
  public static void main(String[] args)
  {
```

```
    final int N = 5; // liczymy n!
    System.out.println(silnia(N));
  }
  static int silnia(int n)
  {
    if (n == 0 || n == 1)
      return 1;
    else
      return n * silnia(n-1);
  }
}
```

Podamy jeszcze jeden przykład problemu, który można rozwiązać metodą „dziel i zwyciężaj”.

Dany jest plecak o objętości v oraz n przedmiotów ponumerowanych od 0 do $n-1$. Każdy przedmiot ma określoną wartość W_i i objętość V_i. Należy zapakować plecak spośród przedmiotów ponumerowanych od 0 do $n-1$ w taki sposób, aby wartość przedmiotów w nim zgromadzonych była największa (patrz podrozdział A.1).

Decyzyjny problem plecakowy można rozwiązać metodą „dziel i zwyciężaj” przy pomocy wzoru rekurencyjnego:

$$P(i,v) = \begin{cases} 0 & \text{dla } i = 0 \wedge v < V_0 \\ W_0 & \text{dla } i = 0 \wedge v \geq V_0 \\ P(i-1,v) & \text{dla } i > 0 \wedge v < V_i \\ max(P(i-1,v), W_i + P(i-1, v-V_i)) & \text{dla } i > 0 \wedge v \geq V_i \end{cases}$$

$P(i,v)$ – maksymalna wartość plecaka o objętości v, zapakowanego spośród przedmiotów ponumerowanych od 0 do i.

Omówimy teraz powyższy wzór:

1. Jeżeli do zapakowania mamy tylko przedmiot z numerem 0 i nie mieści się on do plecaka, to maksymalna wartość plecaka o objętości v jest równa 0.

2. Jeżeli do zapakowania mamy tylko przedmiot z numerem 0 i mieści się on do plecaka, to maksymalna wartość plecaka o objętości v jest równa wartości tego przedmiotu.

W przypadku, gdy $i > 0$, wzór rekurencyjny sprowadza znajdywanie maksymalnej wartości plecaka zapakowanego spośród przedmiotów ponumerowanych od 0 do i do znajdywania maksymalnej wartości plecaka zapakowanego spośród przedmiotów ponumerowanych od 0 do $i-1$.

3. Jeżeli i-ty przedmiot nie mieści się do plecaka, to maksymalna wartość plecaka o objętości v, zapakowanego spośród przedmiotów ponumerowanych od 0 do i jest równa maksymalnej wartości plecaka o objętości v, zapakowanego spośród przedmiotów ponumerowanych od 0 do $i-1$.

Jeżeli i-ty przedmiot mieści się do plecaka, to w plecaku o maksymalnej wartości przedmiot ten albo się znajdzie albo się w nim nie znajdzie. Zastanówmy się jeszcze, kiedy i-ty przedmiot nie znajdzie się w plecaku? Obrazowo można powiedzieć, że i-ty przedmiot nie znajdzie się w plecaku o maksymalnej wartości, jeśli ma on stosunkowo dużą objętość i stosunkowo małą wartość.

4. Jeżeli i-ty przedmiot mieści się do plecaka, to maksymalna wartość plecaka o objętości v, zapakowanego spośród przedmiotów ponumerowanych od 0 do i jest równa większej z dwu wartości:

a) maksymalnej wartości plecaka o objętości v, zapakowanego spośród przedmiotów ponumerowanych od 0 do $i-1$,

b) maksymalnej wartości plecaka o objętości $v - V_i$, zapakowanego spośród przedmiotów ponumerowanych od 0 do $i-1$ plus wartość i-tego przedmiotu.

Oto program, który formalizuje powyższe rozważania:

```
class PlecakDec
{
  final static int N = 6;                // liczba przedmiotów
  final static int MAX_V = 10;           // objetość plecaka
  final static int[] V = {6,2,3,2,3,1};  // objetości przedmiotów
  final static int[] W = {6,4,5,7,10,2}; // wartości przedmiotów
  static int P(int i, int v)
  {
    int w1; // wartość, gdy nie bierzemy i-tego przedmiotu
    int w2; // wartość, gdy bierzemy i-ty przedmiot
    if (i == 0 && v < V[0])  return 0;
    if (i == 0 && v >= V[0]) return W[0];
    w1 = P(i-1,v);
    if (i > 0 && v < V[i]) return w1;
    w2 = W[i] + P(i-1,v-V[i]);
    if (w2 > w1) // gdy i > 0 && v >= V[i]
      return w2;
    else
      return w1;
  }
  public static void main(String[] args)
  {
    System.out.println("Wartosc przedmiotow: " + P(N-1,MAX_V));
  }
}
```

Podamy teraz analogiczny program, który oprócz wartości plecaka wypisuje również numery przedmiotów zapakowanych do plecaka:

```
class PlecakDecWyp
{
  final static int N = 6;                // liczba wszystkich przedmiotow
  final static int MAX_V = 10;           // objetosc plecaka
  final static int[] V = {6,2,3,2,3,1};  // objetosci przedmiotow
  final static int[] W = {6,4,5,7,10,2}; // wartosci przedmiotow
  static class Plecak
  {
    int wartosc;
    int[] zawartosc = new int[N];
  }
  public static void main(String[] args)
  {
    Plecak p = P(N-1,MAX_V);
```

```
    System.out.println("Wartosc plecaka: " + p.wartosc);
    System.out.print("Przedmioty w plecaku: ");
    for (int i = 0; i < N; i++)
      if (p.zawartosc[i] == 1) System.out.print(i + " ");
    System.out.println();
  }
  static Plecak P(int i, int v)
  {
    Plecak p1 = new Plecak(); // plecak, gdy nie bierzemy i-tego przedmiotu
    Plecak p2 = new Plecak(); // plecak, gdy bierzemy i-ty przedmiot
    if (i == 0 && v < V[0])
    {
      p1.zawartosc[0] = 0;
      p1.wartosc = 0;
      return p1;
    }
    if (i == 0 && v >= V[0])
    {
      p2.zawartosc[0] = 1;
      p2.wartosc = W[0];
      return p2;
    }
    Plecak p = P(i-1,v);
    p1.zawartosc = p.zawartosc;
    p1.zawartosc[i] = 0;
    p1.wartosc = p.wartosc;
    if (i > 0 && v < V[i]) return p1;
    p = P(i-1,v-V[i]);
    p2.zawartosc = p.zawartosc;
    p2.zawartosc[i] = 1;
    p2.wartosc = W[i] + p.wartosc;
    if (p2.wartosc > p1.wartosc) // gdy i > 0 && v >= V[i]
      return p2;
    else
      return p1;
  }
}
```

W powyższym programie plecak reprezentowany jest przez obiekt klasy `Plecak`. Atrybuty obiektów klasy `Plecak` określone są przez zmienną `wartosc`, która przechowuje wartość plecaka oraz przez zmienną `zawartosc`, która przechowuje referencję do tablicy przedmiotów z plecaka. Jeśli `i`-ty przedmiot znajduje się w plecaku, to w tablicy, w komórce z indeksem `i` mamy liczbę `1`. Jeśli w plecaku `i`-ty przedmiot się nie znajduje, to w tablicy, w komórce z indeksem `i` mamy liczbę `0`.

Metoda statyczna `P(int i, int v)` zwraca plecak o objętości `v` i maksymalnej wartości, zapakowany spośród przedmiotów ponumerowanych od `0` do `i`.

Ćwiczenia

2.3.1 Rozwiąż ogólny problem plecakowy przy pomocy metody „dziel i zwyciężaj” (patrz pod-

rozdział A.2).

2.3.2 Jak za pomocą metody „dziel i zwyciężaj” rozwiązać problem odgadywania liczby (patrz podrozdział A.6).

2.4 Programowanie dynamiczne

Rozwiązanie problemu metodą „programowania dynamicznego”, podobnie jak w metodzie „dziel i zwyciężaj”, konstruuje się na podstawie rozwiązań jego podproblemów. Jednakże w metodzie programowania dynamicznego wyniki rozwiązań podproblemów zapamiętywane są w tablicy. Pozwala to wyeliminować problem wielokrotnego rozwiązywania tych samych podproblemów, charakterystyczny dla metody „dziel i zwyciężaj”. W metodzie „programowania dynamicznego” każdy podproblem rozwiązuje się tylko raz.

Konstrukcja programu wykorzystującego zasadę programowania dynamicznego może zostać zrealizowana w następujących etapach:

1. Skonstruowanie rozwiązania problemu metodą dziel i zwyciężaj (wraz z jednoznacznym określeniem przypadków elementarnych).
2. Stworzenie tablicy, w której będzie można zapamiętać rozwiązania przypadków elementarnych i rozwiązania podproblemów, które zostaną obliczone na ich podstawie.
3. Wpisanie do tablicy wartości numerycznych, odpowiadających przypadkom elementarnym.
4. Na podstawie wartości numerycznych wpisanych do tablicy, używając formuły rekurencyjnej, obliczenie rozwiązań problemów wyższych rzędów i wpisanie ich do tablicy aż do problemu o zadanym rozmiarze.

Dla przykładu podamy rozwiązanie problemu Fibonacciego, zrealizowane metodą programowania dynamicznego:

```
class FiboDyn
{
  public static void main(String[] args)
  {
    final int N = 5; // n-ty wyraz ciągu
    System.out.println(fibo(N));
  }
  static int fibo(int n)
  {
    int[] tab = new int[n+1];
    tab[1] = 1;
    tab[2] = 1;
    for (int i = 3; i < tab.length; i++)
      tab[i] = tab[i-1] + tab[i-2];
    return tab[tab.length - 1];
  }
}
```

Ćwiczenia

2.4.1 Użyj metody programowania dynamicznego do rozwiązania decyzyjnego problemu plecakowego (patrz podrozdział A.1).

2.4.2 Użyj metody programowania dynamicznego do rozwiązania ogólnego problemu plecakowego (patrz podrozdział A.2).

2.5 Programowanie liniowe

Programowanie matematyczne jest działem matematyki poświęconym teorii i praktycznym algorytmom wyznaczania ekstremum funkcji wielu zmiennych przy ograniczeniach na obszar ich zmienności. W odróżnieniu od analitycznej teorii badania ekstremów funkcji, programowanie matematyczne zajmuje się tymi zadaniami, w których ograniczenia aktywnie wpływają na poszukiwane ekstrema. W praktycznych zastosowaniach, najczęściej pojawiającą się sytuacją jest przypadek, gdy funkcja ekstremalizowana oraz warunki ograniczające podawane są w postaci liniowej. Wtedy mówimy o odmianie programowania matematycznego, które jest nazywane *programowaniem liniowym*.

Zagadnienie programowania liniowego, w standardowej postaci formułuje się następująco. Znaleźć minimum następującej funkcji celu:

$$z = c_1 x_1 + ... + c_n x_n$$

przy warunkach (ograniczeniach):

$$x_1 \geq 0, ..., x_n \geq 0$$

oraz

$$\begin{aligned} &a_{11} x_1 + ... + a_{1n} x_n = b_1 \\ &\cdot \ \cdot \ \cdot \\ &\cdot \ \cdot \ \cdot \\ &\cdot \ \cdot \ \cdot \\ &a_{m1} x_1 + ... + a_{mn} x_n = b_m \end{aligned}$$

gdzie: c_i, a_{ij}, b_j dla $i = 1, ..., n$ oraz $j = 1, ..., m$ są danymi współczynnikami.

Wiele praktycznych zagadnień w naukach przyrodniczych może być sformułowane jako zastosowania programowania liniowego. Teoria programowania liniowego dobrze się rozwinęła. Dzięki temu powstało wiele oprogramowania komputerowego przeznaczonego do rozwiązywania praktycznych problemów, które dają się sformułować jako problemy programowania liniowego. Chyba najbardziej znanym tego typu programem jest program wykonany przez firmę *Frontline Systems* o nazwie *Solver*, który dostarczany jest jako dodatek do arkuszy kalkulacyjnych Excel, Lotus 1-2-3 oraz Quatro Pro.

Wiele problemów nie pasujących do standardowego sformułowania postaci zagadnienia programowania liniowego może być przetransformowane do postaci standardowej za pomocą kilku następujących prostych operacji.

1. Nierówność:
$$a_{11} x_1 + ... + a_{1n} x_n \leq b_1$$
może być sprowadzona do równości poprzez wprowadzenie zmiennej pomocniczej $x_{n+1} \geq 0$ jak następuje:
$$a_{11} x_1 + ... + a_{1n} x_n + x_{n+1} = b_1.$$
Jeśli nierówność ma przeciwny zwrot, zmienną x_{n+1} należy odjąć.

2. Jeśli problem jest sformułowany w postaci standardowej, ale zmienne, jedna lub więcej, przebiegają również wartości ujemne, to problem można sprowadzić do postaci standardowej, zastępując zmienne x_i zmiennymi y_i i z_i przez podstawienie: $x_i = y_i - z_i$, gdzie zarówno y_i, jak i z_i są nieujemne.

3. Jeśli funkcję celu należy maksymalizować, pomnożenie jej przez -1 zmienia maksymalizację na minimalizację.

4. Jeśli nie interesuje nas maksymalizacja ani minimalizacja funkcji celu ale argument, w którym osiąga ona określoną wartość z_0, to definiujemy nową funkcję celu z' postaci:

$$z' = c_1x_1 + ... + c_nx_n - z_0$$

którą minimalizujemy.

Dlatego przy bardziej ogólnym sformułowaniu zagadnienia programowania liniowego możemy nie tylko minimalizować funkcję celu ale także ją maksymalizować lub szukać argumentów dla których osiąga określoną wartość; wartości zmiennych x_1, ..., x_n wcale nie muszą być nieujemne; zaś ograniczenia liniowe wcale nie muszą być postaci równań, ale mogą to być nierówności. Takie możliwości daje nam właśnie program Solver.

Za pomocą programu Solver można miedzy innymi rozwiązywać następujące problemy.

1. Problemy planowania asortymentu produktów dający maksymalny zysk dla producenta (np. problem planowania produkcji mebli).

2. Problemy transportowe, które dotyczą planowania transportu towarów lub produktów, w ten sposób, aby koszt tego transportu był możliwie najmniejszy (np. problem planowania transportu węgla).

3. Problemy otrzymywania produktów w wyniku mieszania pewnych substratów (produktów naftowych, rud metali, pożywienia dla ludzi i zwierząt itd.), które dotyczą planowania optymalnej zawartości produktu powstałego z mieszania szeregu dostępnych substancji tak, aby spełnione były wszelkie wymagania odnośnie własności wymieszanego produktu, a koszty jego otrzymania były możliwie najmniejsze (np. problemy diety).

4. Problemy dotyczące załadunku towarów na samochody, pociągi, statki itd., które dotyczą optymalnego pakowania towarów (np. problemy plecakowe).

5. Problemy planowania optymalnego przepływu w sieciach wodociągowych, gazowych, drogowych, kolejowych, informatycznych itd. (np. problem najkrótszej drogi).

6. Problemy planowanie cięcia materiałów (bel materiałów tekstylnych, papieru, celofanu, folii metalowej itd.) bez pozostawienia zbędnych resztek (np. problem cięcia bel materiałów tekstylnych).

7. Problemy optymalnej obsady stanowisk osób lub położenia rzeczy w czasie i przestrzeni (np. problem doboru załogi statku kosmicznego).

Natomiast ogólny schemat rozwiązania danego problemu metodą programowania liniowego przy użyciu Solvera jest następujący.

1. Z treści opisu problemu odczytać dane wielkości liczbowe oraz ustalić te parametry, których podanie wartości wiąże się ze znalezieniem rozwiązania problemu.

2. Skonstruować funkcję celu zależną od zmienianych parametrów oraz ustalić czy dla rozwiązania problemu chodzi o znalezienie takich argumentów tej funkcji (czyli wartości zmienianych parametrów), dla których funkcja osiąga minimum, maksimum czy też ustaloną wartość.

3. Ustalić warunki ograniczające na zmieniane parametry.

4. Skonstruować arkusz kalkulacyjny, w którym część komórek przechowuje dane wielkości liczbowe, część przeznaczona jest na wartości zmienianych parametrów oraz jedna komórka jest komórką obliczającą wartości funkcji celu. Oprócz tego pewna liczba komórek ma być związana z obliczaniem warunków ograniczających (patrz następny krok).

5. Opierając się na arkuszu skonstruowanym w poprzednim kroku, wprowadzić do okienka Solvera adres komórki obliczającej funkcję celu (czyli tzw. *komórkę celu*), adresy komórek przeznaczonych na wartości zmienianych parametrów (czyli tzw. *komórki zmieniane*) oraz treści warunków ograniczających będące w istocie zapisami kilku relacji (np. $=$. $>$, $>=$, $<$ itd.), przy czym argumenty tych relacji są adresami komórek z arkusza. Oprócz tego wymagane jest podanie (poprzez zaznaczenie kropki w odpowiednim polu) czy funkcja celu ma być maksymalizowana, minimalizowana czy też chodzi o znalezienie wartości w komórkach zmienianych, dla których osiąga ona ustalona wartość.

6. Znaleźć rozwiązanie problemu poprzez naciśnięcie przycisku *Rozwiąż* w okienku Solvera.

Dla przykładu podamy szkic rozwiązania problemu transportu węgla (patrz podrozdział A.14).

Na początku z treści opisu problemu odczytujemy dane wielkości liczbowe oraz ustalamy te parametry, których podanie wartości wiąże się ze znalezieniem rozwiązania problemu. W omawianym problemie dane są koszty transportu pomiędzy kopalniami i miastami, ilości wydobycia węgla osiągnięte przez obydwie kopalnie oraz potrzeby węgla zgłoszone przez poszczególne miasta. Koszty transportu, w dziesiątkach złotych, przedstawia następująca tabelka.

$_z\backslash^{do}$	C	D	E
A	8	5	5
B	4	6	8

Liczby z powyższej tabelki dla wygody późniejszych zapisów oznaczmy przez K_{IJ}, gdzie I jest jedną z kopalni A lub B, a J jest jednym z miast C, D lub E. Oprócz tego wiadomo, że kopalnia A dostarcza dziennie 500 ton a kopalnia B dostarcza dziennie 800 ton węgla. Natomiast miasta C, D i E zużywają odpowiednio 500, 400 i 400 ton węgla dziennie.

Parametrami problemu, których wartości poszukujemy są ilości węgla jakie trzeba dostarczyć z obydwu kopalni do poszczególnych miast. Oznaczamy je przez X_{IJ}, gdzie I jest jedną z kopalni A lub B, a J jest jednym z miast C, D lub E.

Następnie konstruujemy funkcję celu F zależną od zmienianych parametrów oraz ustalamy, że jest to funkcja obliczająca sumaryczny koszt transportu węgla i rozwiązanie problemu polega na podaniu takich wartości argumentów tej funkcji (czyli wszystkich parametrów: X_{IJ}), aby funkcja ta osiągnęła wartość minimalną. Używając wprowadzonych oznaczeń, funkcję tę można zapisać następująco:

$$\begin{aligned} F(X_{AC}, X_{AD}, X_{AE}, X_{BC}, X_{BD}, X_{BE}) = \\ = K_{AC} \cdot X_{AC} + K_{AD} \cdot X_{AD} + K_{AE} \cdot X_{AE} + \\ + K_{BC} \cdot X_{BC} + K_{BD} \cdot X_{BD} + K_{BE} \cdot X_{BE} \end{aligned}$$

Teraz ustalamy warunki ograniczające na zmieniane parametry.

1. Wszystkie parametry X_{IJ} muszą być liczbami całkowitymi nieujemnymi. Wynika to stąd, że rachunki ilości przewożonego węgla wygodnie wykonuje się na całkowitych ilościach ton, oraz zakładamy, że nie można przewozić ujemnej liczby ton węgla.

2. Wydobycie węgla w kopalniach A i B wynosi odpowiednio 500 ton i 800 ton co oznacza, że konieczne jest wprowadzenie następujących dwóch ograniczeń:

$$X_{AC} + X_{AD} + X_{AE} = 500,$$
$$X_{BC} + X_{BD} + X_{BE} = 800.$$

3. Potrzeby węgla w miastach C, D i E wynoszą odpowiednio 500 ton, 400 ton i 400 ton, stąd trzeba wprowadzić następujące ograniczenia:

$$X_{AC} + X_{BC} = 500,$$
$$X_{AD} + X_{BD} = 400,$$
$$X_{AE} + X_{BE} = 400.$$

Po wprowadzeniu powyższych danych do arkusza kalulacyjnego i okienka Solvera oraz naciśnięciu w okienku Solvera przycisku *Rozwiąż*, otrzymujemy szukane ilości węgla transportowane z obydwu kopalń do poszczególnych miast. Otrzymane w ten sposób liczby podajemy w poniższej tabelce.

$_z\backslash^{do}$	C	D	E
A	0	100	400
B	500	300	0

Ćwiczenia

2.5.1 Używając solvera rozwiązać poniższe problemy:

- problem planowania zawartości zestawu paszowego (patrz podrozdział A.9),
- problem planowania produkcji mebli (patrz podrozdział A.7),
- problem planowania diety dziecka (patrz podrozdział A.8),
- problem czterech hetmanów (patrz podrozdział A.10),
- problem wysyłania pociągów (patrz podrozdział A.12),
- problem przydziału maszyn (patrz podrozdział A.13),
- problem transportu produktów (patrz podrozdział A.15),
- problem produkcji samochodów (patrz podrozdział A.16),
- problem transportu koni (patrz podrozdział A.17).

Rozdział 3

Metody aproksymacyjne

Metody rozwiązywania problemów przedstawione w poprzednim rozdziale, pozwalają uzyskać *dokładne*, czyli *optymalne* rozwiązanie badanych problemów. Efektem działania tych metod jest więc wskazanie jednego lub kilku konkretnych stanów pochodzących z przestrzeni stanów, dokładnie odpowiadających specyfikacji stanu końcowego w tej przestrzeni. Aby osiągnąć ten cel, metody dokładne muszą zwykle w jakiś sposób przeglądać całą przestrzeń stanów, co często powoduje, że nie są one na tyle efektywne, aby mogły być stosowane do rozwiązywania nietrywialnych problemów, czyli takich, które dotyczą przeszukiwania przestrzeni o dużej liczbie stanów. Poprawa skuteczności i efektywności tych metod jest możliwa poprzez dopasowanie kierunków przeszukiwania do potrzeb aktualnie rozwiązywanego problemu. W ten sposób konstruowane są tzw. *strategie heurystyczne*. Działanie każdej strategii heurystycznej polega na tym, że zamiast mechanicznie przeszukiwać całą przestrzeń stanów, bada ona tylko wybraną ścieżkę w grafie przeszukiwania lub bardziej ogólnie – pewien fragment przestrzeni stanów. Jeśli metoda heurystyczna analizuje wybraną ścieżkę w grafie przeszukiwania, to ścieżka ta zaczyna się na stanie początkowym i istnieje przypuszczenie, że prowadzi do optymalnego stanu końcowego. W rzeczywistości tak zawsze być nie musi, tzn. ścieżka badana przez strategie heurystyczne nie musi zaprowadzić do optymalnego stanu końcowego. Dlatego metody heurystyczne często prowadzą do znalezienia rozwiązania *prawie optymalnego*, czyli takiego, które nie jest wprawdzie rozwiązaniem optymalnym, ale jest zadawalające z punktu widzenia wymagań stawianych szukanemu rozwiązaniu w obliczu praktycznych zastosowań tego rozwiązania. Rozwiązania takie często nazywamy *rozwiązaniami przybliżonymi* lub inaczej *rozwiązaniami aproksymacyjnymi*

Przy przeszukiwaniu przestrzeni stanów strategie heurystyczne często wykorzystują wiedzę o dziedzinie danego problemu. Wiedza ta umożliwia charakterystykę analizowanych stanów i może być przydatna przy wyborze najbardziej obiecujących kierunków przeszukiwania. Do reprezentowania wspomnianej wiedzy zwykle wykorzystuje się pewne funkcje, zwane *funkcjami heurystycznymi*. Przyporządkowują one poszczególnym stanom wartości numeryczne, które charakteryzują badane stany z punktu widzenia możliwości osiągnięcia z nich stanu końcowego. Wykorzystanie pojęcia funkcji heurystycznej natrafia jednak na pewną trudność. Wiąże się ona z tym, że funkcje heurystyczne muszą być konstruowane w sposób specyficzny do każdego z rozwiązywanych problemów. Ogólnie można powiedzieć, że numeryczna wartość funkcji heurystycznej `f(s)` wyraża ocenę stanu `s` ze względu na dwa następujące kryteria:

- zbieżność do stanu końcowego, czyli bliskość badanego stanu do stanu końcowego,
- koszt drogi wyznaczonej od stanu początkowego do stanu bieżącego.

Oczywistym wydaje się fakt, że z punktu widzenia funkcji heurystycznej, lepszym stanem jest zawsze taki stan, który jest bliższy stanowi końcowemu i koszt drogi do tego stanu od stanu początkowego jest możliwie najmniejszy.

3.1 Metoda wspinaczkowa

Jedną z najprostszych i bardzo popularnych heurystycznych strategii przeszukiwania jest *niesystematyczna strategia zachłanna*. Działanie jej polega na tym, że na danym etapie przeszukiwania, do dalszej ekspansji wybierany jest ten spośród stanów możliwych do wygenerowania ze stanu bieżącego, który wydaje się najbardziej obiecujący z punktu widzenia dotarcia w czasie przeszukiwania do stanu końcowego. Postępowanie podejmowane w startegii zachłannej jest podobne do akcji turysty atakującej szczyt wzniesienia. Chcąc jak najszybciej osiągnąć sukces, wybiera on aktualnie najlepszy kierunek, chociaż w trakcie dalszej wspinaczki decyzja ta może okazać się błędna i kierunek w danej chwili gorszy, może być w perspektywie całej wspinaczki lepszy. Ze względu na powyższą analogię, omawianą w tym podrozdziale strategię nazywamy *strategią wspinaczkową* lub *strategią wspinaczki górskiej* (ang. *hill-climbing*).

Dla poprawnego działania strategii wspinaczkowej konieczna jest funkcja heurystyczna, które pozwala na skuteczną ocenę jakości poszczególnych stanów, z punktu widzenia naprowadzenia procesu przeszukiwania na stan końcowy.

```
procedura wspinaczka(p: stan początkowy) -> sukces lub porażka
  1. Jeśli stan p jest stanem końcowym, to:
        zwróć(sukces),
     w przeciwnym razie:
        nazwij stan p stanem bieżącym i oznacz go przez s.
  2. Dopóki stan bieżący s nie jest stanem końcowym lub nie
     można ze stanu bieżącego wygenerować innego stanu wykonuj:
     a) spośród stanów, które można wygenerować ze stanu
        bieżącego s wybierz stan dla którego wartość funkcji
        heurystycznej f jest największa i oznacz go przez t,
     b) jeśli stan t jest stanem końcowym, to:
           zwróć(sukces),
        w przeciwnym razie:
           jeśli wartość f(t) jest większa od wartości f(s), to:
              przyjmij stan t jako nowy stan bieżący s.
  3. Zwróć(porażka).
```

Strategia wspinaczkowa ze względu na prostotę obliczeniową jest bardzo popularna. Jednak jej wadą jest brak możliwości powrotu do kierunków przeszukiwania, które na pewnym etapie były lokalnie gorsze. Może to oczywiście prowadzić do badania drogi prowadzącej do nie końcowego stanu, z którego nie można wygenerować żadnego innego stanu.

3.2 Algorytmy zachłanne

Idee leżące u podstaw opisanej w poprzednim podrozdziale metody wspinaczkowej często w literaturze formułuje się bardziej ogólnie w postaci tzw. *algorytmów zachłannych* (ang. *greedy algorithm*). Algorytmy zachłanne cechują się tym, że zawsze wykonują takie działania, które w danej chwili wydają się najkorzystniejsze. Okazuje się, że podczas swej codziennej działalności, ludzie często postępują podobnie. Objawia się to tym, że rozwiązując jakieś zadanie zadawalamy się jego szybkim i w miarę poprawnym rozwiązaniem, choć niekoniecznie optymalnym. Algorytmy zachłanne nie zawsze prowadzą bowiem do optymalnych rozwiązań, choć dla wielu problemów rozwiązania jakie dostarczają algorytmy zachłanne są zupełnie wystarczające. Tak więc, algorytmy zachłanne stosujemy wtedy, gdy na podstawie pewnych danych wejściowych, należy szybko skonstruować rozwiązanie danego problemu. Bowiem algorytm zachłanny zawsze

stara się jak najszybciej skonstruować rozwiązanie problemu, używając tych fragmentów danych wejściowych, które na danym etapie konstrukcji rozwiązania wydają się najbardziej użyteczne, tzn. w danym momencie najbardziej przybliżają do skonstruowania ostatecznego rozwiązania. Schemat działania algorytmu zachłannego przedstawia następująca procedura.

```
procedura algorytmZachłanny(W: zbiór danych wejściowych)
  1. R := zbiór pusty;
  2. Dopóki czyZnaleziono(rozwiązanie) = fałsz i W jest niepuste wykonuj:
     a) x := wybór(W),
     b) R := suma(R,{x}),
     c) W := różnica(W,{x}),
  3. Jeśli czyRozwiązanie(R) = prawda, to:
        zwróć R,
     w przeciwnym razie:
        wypisz "Brak rozwiazan".
```

Przy czym:

`R` – wybrany podczas działania algorytmu podzbiór zbioru danych wejściowych, na podstawie którego ma być konstruowane rozwiązanie,

`x` – pojedynczy element zbioru `W`,

`wybór(W)` – procedura dokonująca optymalnego na danym etapie obliczeń, wyboru elementu ze zbioru `W` do konstrukcji rozwiązania problemu,

`suma(R,{x})` – procedura dołączająca element `x` do zbioru `R`,

`różnica(W,{x})` – procedura usuwająca element `x` ze zbioru `W`,

`czyRozwiązanie(R)` – procedura sprawdzająca, czy na podstawie zbioru `R` można już skonstruować kompletne rozwiązanie problemu.

Klasycznym przypadkiem problemu, który można rozwiązać za pomocą powyższego podejścia jest problem wydawania reszty (patrz podrozdział A.28). Aby rozwiązać ten problem wystarczy zauważyć, że aby szybko wydać ustaloną kwotę pieniędzy (tzn. minimalną liczbą monet), trzeba wydawać możliwie największe nominały, aż do wydania całej reszty. Tak sformułowany pomysł, można bardziej formalnie zapisać w postaci następującej procedury:

```
procedura wydajReszte(r: reszta do wydania)
  1. Wybierz monetę M o największym nominale.
  2. Dopóki r > 0 wykonuj:
     Jeśli r >= M, to:
        a) wydaj monetę M,
        b) odejmij od reszty wartość monety M,
     w przeciwnym razie:
        wybierz monetę M o mniejszym nominale.
```

Oto program, który realizuje powyższy algorytm:

```
import java.io.*;
class ResztaZachlanny
{
  public static void main(String[] args) throws IOException
  {
    InputStreamReader str = new InputStreamReader(System.in);
    BufferedReader wejscie = new BufferedReader(str);
    String tekst;
    final int[] M = {500,200,100,50,20,10,5,2,1};
    int zl, gr, r, i = 0;
    System.out.println("Podaj reszte..");
    System.out.print("zlotych: ");
    tekst = wejscie.readLine();
    zl = Integer.parseInt(tekst);
    System.out.print("groszy: ");
    tekst = wejscie.readLine();
    gr = Integer.parseInt(tekst);
    System.out.print("Reszta: ");
    r = zl*100 + gr;
    while (r > 0)
    {
      if (r >= M[i])
      {
        System.out.print(M[i]/100.0 + " ");
        r = r - M[i];
      }
      else
        i++;
    }
    System.out.println();
  }
}
```

Ćwiczenia

3.2.1 Skonstruuj algorytm zachłanny rozwiązujący decyzyjny problem plecakowy (patrz podrozdział A.1).

3.2.2 Skonstruuj algorytm zachłanny rozwiązujący ogólny problem plecakowy (patrz podrozdział A.2).

3.2.3 Skonstruuj algorytm zachłanny rozwiązujący problem doboru załogi statku kosmicznego (patrz podrozdział A.4).

3.2.4 Skonstruuj algorytm zachłanny rozwiązujący problem wyprodukowania lodów wszystkich smaków (patrz podrozdział A.5).

3.3 Metody stochastyczne

Oprócz strategii zachłannych, istnieją jeszcze inne metody aproksymacyjne. Są to np. *metody stochastyczne*. Ogólnie mówiąc metody te polegają na tym, że zamiast systematycznie przeszukiwać całą przestrzeń stanów w celu znalezienia stanu końcowego, przegląda się tylko pewną liczbę stanów wybraną w większym lub mniejszym stopniu losowo, mając nadzieje, że pośród nich znajdzie się stan końcowy, będący rozwiązaniem optymalnym badanego problemu lub chociaż stan końcowy, będący zadawalającym nas rozwiązaniem prawie optymalnym. Tak więc, każde rozwiązanie stochastyczne ma w sobie pewien element losowy, wyrażający się w niedeterministycznym wyborze kolejnych stanów, które są badane w czasie przeszukiwania przestrzeni stanów. Ze względu na sposób wybierania badanych stanów wyróżnia się dwa typy metod stochastycznych. Są to tzw. *metody globalne* oraz *metody lokalne*. Obydwa wspomniane typy omówimy w kolejnych dwóch podrozdziałach.

3.3.1 Metody globalne

Działanie *stochastycznych metod globalnych* polega na tym, że losowo przeszukują one całą przestrzeń stanów traktując podobnie każdy z analizowanych stanów. Innymi słowy, metody globalne nie wyróżniają podczas przeszukiwania żadnych konkretnych stanów. Jedną z najprostszych tego typu metod jest tzw. *metoda Monte Carlo*. Działa ona w ten sposób, że najpierw losowana jest pewna liczba stanów, po czym spośród nich wybierany jest taki stan, który najbardziej jest zbliżony do wymagań stawianych optymalnemu stanowi końcowemu. Ten właśnie stan zwracany jest jako rozwiązanie analizowanego problemu. Aby z odpowiednio wysokim prawdopodobieństwem uzyskać zadawalające rozwiązanie analizowanego problemu, należy przebadać odpowiednio wybraną próbkę stanów należących do badanej przestrzeni stanów. Próbka ta musi być reprezentatywna oraz mieć odpowiednią wielkość (stosowne oszacowania statystyczne można znaleźć np. [21]).

Dla przykładu, zastosujemy metodę Monte Carlo do rozwiązania problemu wydawania reszty (patrz A.28). Najprościej można to zrobić w ten sposób, że losuje się pewną liczbę konfiguracji monet, które pozwalają na wydanie ustalonej reszty i jako wynik zwracana jest ta konfiguracja, która ma najmniejszą liczbę monet. Poniższy program przedstawia rozwiązanie problemu wydawania reszty metodą Monte Carlo zapisane w języku Java.

```
import java.io.*;
import java.util.*;
class ResztaMonteCarlo
{
  public static void main(String[] args) throws IOException
  {
    InputStreamReader str = new InputStreamReader(System.in);
    BufferedReader wejscie = new BufferedReader(str);
    String tekst;
    Random losuj = new Random();
    final int[] MONETY = {500,200,100,50,20,10,5,2,1};
    final int LIMIT_WYDANYCH_MONET = 1000; // limit wydanych monet + 1
    final int LICZBA_LOSOWAN = 100;
    int minLiczbaWydanychMonet = LIMIT_WYDANYCH_MONET;
    int[] wydaneMonety = new int[LIMIT_WYDANYCH_MONET];
    int[] minWydaneMonety = new int[LIMIT_WYDANYCH_MONET];
    int moneta, zl, gr, r;
```

```
    System.out.println("Podaj reszte..");
    System.out.print("zlotych: ");
    tekst = wejscie.readLine();
    zl = Integer.parseInt(tekst);
    System.out.print("groszy: ");
    tekst = wejscie.readLine();
    gr = Integer.parseInt(tekst);
    for (int i = 0; i < LICZBA_LOSOWAN; i++)
    {
      r = zl*100 + gr;
      int liczbaWydanychMonet = 0;
      while (r > 0 && liczbaWydanychMonet < LIMIT_WYDANYCH_MONET)
      {
        moneta = MONETY[losuj.nextInt(MONETY.length)];
        if (r >= moneta)
        {
          wydaneMonety[liczbaWydanychMonet] = moneta;
          r = r - moneta;
          liczbaWydanychMonet++;
        }
      }
      if (minLiczbaWydanychMonet > liczbaWydanychMonet)
      {
        minLiczbaWydanychMonet = liczbaWydanychMonet;
        for (int j = 0; j < liczbaWydanychMonet; j++)
          minWydaneMonety[j] = wydaneMonety[j];
      }
    }
    if (minLiczbaWydanychMonet < LIMIT_WYDANYCH_MONET)
    {
      System.out.print("Reszta: ");
      for (int i = 0; i < minLiczbaWydanychMonet; i++)
        System.out.print(minWydaneMonety[i]/100.0 + " ");
    }
    else
      System.out.print("Nie znaleziono rozwiazania.");
    System.out.println();
  }
}
```

Ćwiczenia

3.3.1 Użyj metody Monte Carlo do rozwiązania decyzyjnego problemu plecakowego (patrz podrozdział A.1).

3.3.2 Użyj metody Monte Carlo do rozwiązania ogólnego problemu plecakowego (patrz podrozdział A.2).

3.3.3 Użyj metody Monte Carlo do rozwiązania problemu doboru załogi statku kosmicznego (patrz podrozdział A.4).

3.3.4 Użyj metody Monte Carlo do rozwiązania problemu wyprodukowania lodów wszystkich smaków (patrz podrozdział A.5).

3.3.5 Użyj metody Monte Carlo do rozwiązania problemu odgadywania liczby (patrz podrozdział A.6).

3.3.2 Metody lokalne

Stochastyczne metody lokalne charakteryzują się tym, że przeszukiwanie przestrzeni stanów rozpoczynają od jednego lub czasem kilku losowo wybranych stanów. Następnie sprawdzany jest stan (wybrany w większym lub mniejszym stopniu losowo), który w jakimś sensie jest stanem sąsiednim do analizowanego wcześnie stanu (np. jest bardzo bliski w sensie definiowanej wcześniej odległości pomiędzy stanami). Klasycznym przykładem tego typu metody jest strategia o nazwie *błądzenie losowe*, która opiera się właśnie na mocno intuicyjnym przypuszczeniu, że poruszając się od stanu do stanu, bez żadnej konkretnej strategii (czyli na sposób „losowy"), natrafi się kiedyś na stan końcowy. Strategia ta działa w ten sposób, że kolejny stan wybierany jest losowo spośród tych stanów, które możliwe są do osiągnięcia ze stanu bieżącego. Zatem strategia ta wykorzystuje operatory przejścia od stanu do stanu, przy czym przy przejściu od stanu do stanu, wybierany jest losowo jeden operator spośród tych, które są możliwe do zastosowania w danym stanie.

```
procedura błądzenieLosowe(p: stan początkowy) -> sukces lub porażka
  1. Jeśli stan p jest stanem końcowym, to:
        zwróć(sukces),
     w przeciwnym razie:
        nazwij stan p stanem bieżącym.
  2. Dopóki stan bieżący nie jest stanem końcowym lub nie można.
     ze stanu bieżącego wygenerować innego stanu wykonuj:
     a) spośród stanów, które można wygenerować ze stanu
        bieżącego wybierz losowo jeden stan i oznacz go przez t,
     b) jeśli stan t jest stanem końcowym, to:
           zwróć(sukces),
        w przeciwnym razie:
           podstaw stan t jako nowy stan bieżący.
  3. Zwróć(porażka).
```

Strategia błądzenia losowego jest intuicyjnie zrozumiała i łatwa do zaimplementowania. Niestety posiada ona też pewne wady. Jeśli w trakcie przeszukiwania przestrzeni stanów natrafimy na stan, który nie jest stanem końcowym i z którego nie można wygenerować innego stanu, to przeszukanie pozostałych stanów nie będzie możliwe. Co więcej, jeśli przeszukaliśmy już pewną liczbę stanów, to i tak nie jesteśmy w stanie określić, jaki czas pozostał do zakończenia procesu przeszukiwania.

Jak już pisaliśmy, stosowanie strategii błądzenia losowego wymaga podania operatorów przejścia od stanu do stanu. W ogólnym przypadku podanie tych operatorów nie zawsze może być łatwe lub może ich być bardzo dużo. Dlatego metodę błądzenia losowego wygodnie jest stosować do problemów, dla których operatory przejścia od stanu do stanu są z góry określone lub są łatwe do zdefiniowania. Jest tak np. w problemie przeprawy przez rzekę wilka, kozy i kapusty (patrz podrozdział A.30), gdzie możemy wyróżnić trzy następujące operatory:

- przewiezienie wilka na drugi brzeg,
- przewiezienie kozy na drugi brzeg,
- przewiezienie kapusty na drugi brzeg.

Oto program, który rozwiązuje problem przeprawy przez rzekę wilka, kozy i kapusty:

```
import java.util.*;
class WilkKozaKapusta
{
  public static void main(String[] args) throws CloneNotSupportedException
  {
    Stan[] tab = new Stan[100];
    tab[0] = new Stan(); // stan początkowy
    int i;
    for (i = 1; i < 100; i++)
    {
      tab[i] = tab[i-1].nastepnyStan();
      if (tab[i].stanKoncowy()) break;
    }
    if (i == 100)
      System.out.println("Nie znaleziono rozwiazania.");
    else
    {
      System.out.println("Droga do stanu koncowego:");
      for (int j = 0; j <= i; j++)
      {
        tab[j].wypiszStan();
      }
      System.out.println();
    }
  }
}
class Stan implements Cloneable
{
  private boolean wilk, koza, kapusta;
  private static Random losuj = new Random();
  boolean stanKoncowy() // czy stan koncowy ?
  {
    return wilk && koza && kapusta;
  }
  void wypiszStan()
  {
    System.out.print(wilk ? "(1," : "(0,");
    System.out.print(koza ? "1," : "0,");
    System.out.print(kapusta ? "1) " : "0) ");
  }
  Stan nastepnyStan() throws CloneNotSupportedException
  {
```

```
    Stan nastepny = (Stan)clone();
    do
    {
      int i = losuj.nextInt(3);
      if (i == 0 && (koza ^ kapusta)) // przewiezienie wilka
      {
        nastepny.wilk = !wilk;
        break;
      }
      if (i == 1) // przewiezienie kozy
      {
        nastepny.koza = !koza;
        break;
      }
      if (i == 2 && (wilk ^ koza)) // przewiezienie kapusty
      {
        nastepny.kapusta = !kapusta;
        break;
      }
    } while(true);
    return nastepny;
  }
}
```

W problemie przeprawy przez rzekę wilka, kozy i kapusty, stan z przestrzeni przeszukiwania odzwierciedla rozmieszczenie wilka, kozy i kapusty na obu brzegach rzeki. W programie stan reprezentowany jest przez obiekt klasy `Stan`. Własności obiektów klasy `Stan` określone są przez zmienne `wilk`, `koza`, `kapusta` typu `boolean`. Wartość `false` dla zmiennej oznacza, że odpowiadający jej obiekt ze środowiska problemu znajduje się na lewym brzegu rzeki. Wartość `true` dla zmiennej oznacza, że odpowiadający jej obiekt ze środowiska problemu znajduje się na prawym brzegu rzeki. Przy stanie początkowym wszystkie zmienne mają wartość `false` – wilk, koza i kapusta znajdują się na lewy brzegu rzeki. Przejście od danego stanu do innego stanu następuje przez wywołanie metody obiektowej `nastepnyStan`. Metoda obiektowa `nastepnyStan` tworzy kopie aktualnego obiektu klasy `Stan`, losuje jeden z trzech dostępnych operatorów i sprawdza, czy może on zostać zastosowany dla danego stanu. Operator „przewiezienie wilka na drugi brzeg" może zostać zastosowany tylko wtedy, gdy koza i kapusta znajdują na różnych brzegach rzeki. Zmienne obiektowe `koza` i `kapusta` muszą mieć wtedy różne wartości. Aby to sprawdzić korzystamy z operatora logicznego `^` (dysjunkcja lub różnica symetryczna). Operator dysjunkcji zwraca wartość `true` wtedy i tylko wtedy, gdy jego argumenty mają różne wartości logiczne. Operator „przewiezienie kozy na drugi brzeg" może zostać zastosowany dla dowolnego stanu. Operator „przewiezienie kapusty na drugi brzeg" może zostać zastosowany tylko wtedy, gdy wilk i koza znajdują się na różnych brzegach rzeki. Jeżeli wylosowany operator nie może zostać zastosowany dla danego stanu, to losowany jest kolejny operator. Po znalezieniu operatora właściwego dla danego stanu metoda obiektowa `nastepnyStan` zwraca obiekt klasy `Stan`, który reprezentuje stan następniczy w stosunku do aktualnego stanu. Przewiezienie obiektu ze środowiska problemu na drugi brzeg rzeki realizowane jest przez zanegowanie wartości odpowiedniej zmiennej obiektowej. Wszystkie stany, począwszy od stanu początkowego zapamiętywane są w tablicy. Kolejne stany generowane są do momentu znalezienia stanu końcowego. Jeśli stan końcowy nie zostanie znaleziony po wygenerowaniu 100 stanów, to zostanie wypisany napis `Nie znaleziono rozwiazania.` Wszystkie stany,

począwszy od stanu początkowego do stanu końcowego wypisywane są na ekranie w postaci trójek liczb o wartościach `0` lub `1`. Liczby te określają rozmieszczenie wilka, kozy i kapusty na obu brzegach rzeki. Liczba `0` odpowiada wartości `false` a liczba `1` wartości `true`. Na podstawie kolejności wygenerowanych stanów możemy odczytać operatory przejścia między stanami.

Podczas jednego z uruchomień, program wypisał następujące rozwiązanie:

```
Droga do stanu koncowego:
(0,0,0) (0,1,0) (1,1,0) (0,1,0) (1,1,0) (0,1,0) (0,1,1) (0,0,1) (1,0,1) (1,1,1)
```

Powyższe rozwiązanie nie jest optymalne. Znaleziona droga od stanu początkowego do stanu końcowego nie jest najkrótsza. Aby się o tym przekonać, wystarczy zauważyć, że pewne stany powtarzają się w niej – np. stan `(0,1,0)`. Przy stosowaniu metody błądzenia losowego bardzo często otrzymujemy rozwiązania nieoptymalne.

Ćwiczenia

3.3.6 Użyj metody błądzenia losowego do rozwiązania decyzyjnego problemu plecakowego (patrz podrozdział A.1).

3.3.7 Użyj metody błądzenia losowego do rozwiązania problemu doboru załogi statku kosmicznego (patrz podrozdział A.4).

3.3.8 Użyj metody błądzenia losowego do rozwiązania problemu przeprawy przez rzekę misjonarzy i ludożerców (patrz podrozdział A.29).

Dodatek A

Wybrane problemy

A.1 Decyzyjny problem plecakowy

Dany jest plecak o objętości $v = 10$ oraz 6 przedmiotów ponumerowanych od 0 do 5. Każdy przedmiot ma określoną wartość W_i i objętość V_i. Należy zapakować plecak spośród przedmiotów ponumerowanych od 0 do 5 w taki sposób, aby wartość przedmiotów w nim zgromadzonych była największa. Wartości i objętości przedmiotów określone są w poniższej tabeli:

i	0	1	2	3	4	5
V_i	6	2	3	2	3	1
W_i	6	4	5	7	10	2

Odp: W plecaku o maksymalnej wartości znajdą się przedmioty `1,2,3,4` o wartości `26`.

A.2 Ogólny problem plecakowy

Dany jest plecak o objętości $v = 23$ oraz nieograniczona liczba egzemplarzy 6 różnych przedmiotów ponumerowanych od 0 do 5. Każdy przedmiot ma określoną wartość W_i i objętość V_i. Należy zapakować plecak spośród przedmiotów ponumerowanych od 0 do 5 w taki sposób, aby wartość przedmiotów w nim zgromadzonych była największa. Wartości i objętości przedmiotów określone są w poniższej tabeli:

i	0	1	2	3	4	5
V_i	6	2	3	2	3	1
W_i	6	4	5	7	10	2

Odp: W plecaku o maksymalnej wartości znajdzie się dziesięć przedmiotów z numerem `3` i jeden przedmiot z numerem `4`. Łączna wartość przedmiotów w plecaku wyniesie `80`.

A.3 Problem liczb pitagorejskich

Liczby pitagorejskie - to trzy liczby naturalne x, y, z, które spełniają warunek $x^2 + y^2 = z^2$. Znaleźć wszystkie takie trójki liczb, przy założeniu, że każda z tych liczb ma należy do zbioru $\{1, ..., 30\}$.

A.4 Problem doboru załogi statku kosmicznego

Organizowana jest załogowa ekspedycja kosmiczna na Marsa. W ramach kompletowania załogi statku kosmicznego główny organizator ekspedycji ma rozwiązać następujący problem. Do jego dyspozycji jest pięciu kosmonautów (k1, k2, k3, k4, k5), którzy przeszli pozytywnie kwalifikację wstępną do udziału w ekspedycji. Na pokładzie specjalnego statku kosmicznego spośród nich będzie potrzebny przynajmniej jeden specjalista w każdej z dziedzin: A, B, C i D. Wiadomym jest, że w dziedzinie A specjalizują się kosmonauci k1 i k4, w dziedzinie B specjalizują się kosmonauci k2, k3 i k4, w dziedzinie C specjalizują się kosmonauci k3 i k5 oraz w dziedzinie D specjalizują się kosmonauci k1, k2 i k5. Ze względu na szczupłość miejsca na pokładzie statku kosmicznego, z powyższej piątki trzeba wybrać jak najmniejszą grupkę kosmonautów, tak aby znajdował się w niej przynajmniej jeden specjalista w każdej z dziedzin A, B, C, D.

Odp: Na Marsa mogą polecieć np. kosmonauci k4 i k5.

A.5 Problem wyprodukowania lodów wszystkich smaków

Fabryka lodów każdego dnia produkuje na tej samej maszynie lody o sześciu różnych smakach. Zmiana produkcji ze smaku i na smak j wymaga przestrojenia maszyny, czyli przygotowania (w tym umycia) do nowej produkcji, które trwa określony czas. Podana niżej tablica zawiera informację o czasach przestrojenia maszyny.

$_i\backslash^j$	1	2	3	4	5	6
1	0	7	20	21	12	23
2	27	0	13	16	46	5
3	53	15	0	25	27	6
4	16	2	35	0	47	10
5	31	29	5	18	0	4
6	28	24	1	17	5	0

Znaleźć kolejność produkcji, przy której całkowity czas przestrojenia maszyny jest minimalny. Przyjąć, że na koniec dnia maszyna ma być przygotowana do produkcji w następnym dniu.

Odp: Minimalny całkowity czas przestrojenia maszyny wynosi `63`. Oto przykładowa kolejność produkcji lodów, dla której całkowity czas przestrojenia maszyny jest minimalny: `1,2,6,5,3,4,1`.

A.6 Problem odgadywania liczby

Jak odgadnąć liczbę pomyślaną przez rozmówcę z zakresu od 0 do 1000, zadając jak najmniejszą liczbę pytań rozmówcy? Przy czym, pytania zadawane rozmówcy muszą być jedynie typu: „Czy to jest liczba 546?”, na które rozmówca może odpowiadać na jeden z następujących sposobów: „TAK”, „ZA MAŁA” lub „ZA DUŻA”.

A.7 Problem planowania produkcji mebli

Wytwórca mebli produkuje stoły, krzesła, biurka i szafy biblioteczne. Do produkcji wykorzystuje dwa typy desek. W magazynie wytwórca posiada 1500 m pierwszego typu desek i 1000 m

drugiego. Dysponuje kapitałem 860 godzin roboczych na wykonanie całej pracy. Przewidywane zapotrzebowanie plus potwierdzone zamówienia wymagają wykonania co najmniej 40 stołów, 130 krzeseł, 30 biurek i nie więcej niż 10 szaf bibliotecznych. Każdy stół, krzesło, biurko i szafa biblioteczna wymaga odpowiednio 5, 1, 9 i 12 m desek pierwszego typu i 2, 3, 4 i 1 m desek drugiego typu. Na wykonanie stołu potrzebne są 3 godziny pracy, krzesła 2 godziny, biurka 5 godzin i szafy bibliotecznej 10 godzin. Przy sprzedaży jednego stołu, krzesła, biurka i szafy bibliotecznej wytwórca osiąga zysk odpowiednio 48 złotych, 20 złotych, 60 złotych i 40 złotych. Zaplanować produkcję mebli w ten sposób, aby wytwórca wywiązał się z przyjętych zamówień i osiągnął maksymalny zysk.

Odp: Wytwórca otrzyma maksymalny zysk `11600` złotych, jeżeli wyprodukuje `150` stołów, `130` krzeseł i `30` biurek.

A.8 Problem planowania diety dziecka

Matka chce, aby jej dzieci otrzymały pewne ilości czynników odżywczych w porannej owsiance. Dzieci mogą wybrać płatki owsiane firmy A lub firmy B albo mieszaninę tych dwóch. Ich śniadanie powinno zawierać co najmniej 1 mg witaminy B_1, 5 mg witaminy PP i 400 kalorii. Wiadomo, że 30 g płatków firmy A zawiera 0.1 mg witaminy B_1, 1 mg witaminy PP i 110 kalorii. Natomiast 30 g płatków firmy B zawiera 0.25 mg witaminy B_1, 0.25 mg witaminy PP i 120 kalorii. Cena 30 g płatków firmy A i płatków firmy B wynosi odpowiednio 16 i 18 groszy. W jaki sposób należy przyrządzić poranna owsiankę, aby spełnione były warunki zdrowotne, a cena produktów była możliwie najniższa.

Odp: Minimalny koszt `116` groszy uzyskamy, jeśli przyrządzimy śniadanie z `5` porcji płatków firmy `A` i `2` porcji płatków firmy `B`.

A.9 Problem planowania zawartości zestawu paszowego

W zestaw paszowy bydła w okresie zimowym wchodzą trzy składniki: siano, kiszonka i pasza treściwa, które zawierają trzy rodzaje podstawowych składników odżywczych, takich jak: białko, wapno i witaminy. Ich zawartość w kilogramie paszy, ceny (w złotówkach za kilogram) oraz minimalne normy dzienne (w kilogramach) niezbędne do prawidłowego odżywiania bydła podaje poniższa tabelka.

Rodzaje paszy	Składniki odżywcze			Ceny
	białko	wapno	witaminy	
Siano	50	6	2	3
Kiszonka	20	4	1	2
Pasza treściwa	180	3	1	5
Mineralne normy dzienne	2 000	120	40	-

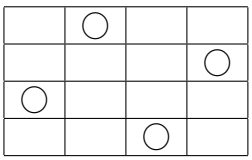
Ustalić dzienny zestaw paszowy o minimalnych kosztach przy dodatkowym założeniu, że w normie dziennej ilość siana nie przekroczy 10 kilogramów.

Odp: Minimalny koszt `91` złotych uzyskamy dla zestawu paszowego: `10` kg siana, `13` kg kiszonki, `7` kg paszy treściwej.

A.10 Problem czterech hetmanów

Jak umieścić 4 hetmanów na szachownicy 4x4 tak, aby nie biły się wzajemnie, tzn. żadnych dwóch hetmanów nie było umieszczonych w tym samym wierszu, kolumnie lub na przekątnej szachownicy.

Odp: Oto przykładowy rozkład hetmanów na szachownicy:



A.11 Problem planowania liczebności klas

Zaplanować liczbę uczniów w klasach Ia, Ib, Ic, Id przy założeniu, że do szkoły przyjęto 68 dziewcząt i 54 chłopców. Maksymalna liczba uczniów w poszczególnych klasach może wynosić:

Ia	32
Ib	28
Ic	34
Id	36

Dodatkowo wymaga się, aby w poszczególnych klasach liczba dziewcząt była większa lub równa liczbie chłopców.

Odp: W poniższej tabeli podany jest przykładowy rozkład uczniów w klasach:

	ch	dz
Ia	5	19
Ib	14	14
Ic	17	17
Id	18	18

A.12 Problem wysyłania pociągów

Z miejscowości wypoczynkowej można w końcu sezonu wysłać najwyżej 12 pociągów dziennie. Są to pociągi spalinowe (tzn. z lokomotywami o napędzie spalinowym) albo elektryczne (tzn. z lokomotywami o napędzie elektrycznym). Wszystkie miejsca w tych pociągach są numerowane (czyli są to tzw. „miejscówki"). w skład każdego pociągu spalinowego wchodzi 2 wagony 80-osobowe i 4 wagony 48-osobowe. Natomiast w skład każdego pociągu elektrycznego wchodzi 5 wagonów 80-osobowych i 2 wagony 48-osobowe. Stacja może wysłać dziennie najwyżej 42 wagony 80-osobowe i najwyżej 40 wagonów 48 osobowych. Ile i jakich pociągów należy wysłać dziennie, aby liczba przewiezionych pasażerów była możliwie największa? Przyjąć, że liczba lokomotyw każdego rodzaju nie jest ograniczona.

Odp: Liczba przewiezionych pasażerów będzie największa i wyniesie `5088` osób, jeżeli będziemy codziennie wysyłać 6 pociągów spalinowych i 6 pociągów elektrycznych.

A.13 Problem przydziału maszyn

Fabryka produkuje dwa modele: A i B. Każdy model musi kolejno przejść przez dwie maszyny I i II. Dla wykonania każdego egzemplarza A maszyna I musi pracować jedna godzinę, a maszyna II - 2.5 godziny. Dla wykonania modelu B maszyny I i II muszą pracować odpowiednio 4 godziny i 2 godziny. Maszyna I może być w ruchu najwyżej 8 godzin dziennie, a maszyna II - 12 godzin. Każdy model A przynosi 120 złotych zysku, natomiast każdy model B przynosi 160 złotych zysku. Ilu sztuk każdego modelu powinna produkować dziennie fabryka, aby zysk ze sprzedaży był możliwie największy?

Odp: Zysk ze sprzedaży będzie największy i wyniesie `640` złotych, jeżeli fabryka wyprodukuje dziennie `4` sztuki modelu A i `1` sztukę modelu B.

A.14 Problem transportu węgla

Kopalnie A i B dostarczają węgiel do miast C, D i E. Kopalnia A dostarcza dziennie 500 ton, natomiast kopalnia B dostarcza dziennie 800 ton węgla. Miasta C, D i E zużywają odpowiednio 500, 400 i 400 ton węgla dziennie. Koszt transportu, w dziesiątkach złotych, jednej tony węgla do poszczególnych miast podaje następująca tabelka:

$_z\backslash^{do}$	C	D	E
A	8	5	5
B	4	6	8

Jak należy zorganizować transport węgla, aby koszt był możliwie najmniejszy?

Odp: Aby uzyskać minimalny koszt transportu węgla wynoszący `6300` dziesiątek złotych, należy z kopali A i B przetransportować do miast C, D i E węgiel w następujących proporcjach:

$_z\backslash^{do}$	C	D	E
A	0	100	400
B	500	300	0

A.15 Problem transportu produktów

Pewne przedsiębiorstwo ma centrale zbytu swojego produktu w miastach Warszawa, Gdańsk i Wrocław. Centrale te posiadają odpowiednio 40, 20 i 40 jednostek produktu. Punkty sprzedaży zamówiły następujące ilości jednostek: Szczecin 25, Poznań 10, Olsztyn 20, Katowice 30, Rzeszów 15. Koszt transportu (w dziesiątkach złotych) między każdą centralą i punktem sprzedaży podaje następująca tablica:

$_z\backslash^{do}$	Szczecin	Poznań	Olsztyn	Katowice	Rzeszów
Warszawa	52	30	23	31	33
Wrocław	37	17	45	18	42
Gdańsk	38	31	18	53	65

Jak rozesłać zamówione produkty z central do punktów sprzedaży, aby koszty transportu były możliwie najmniejsze?

Odp: Aby uzyskać minimalny koszt transportu produktów wynoszący 2675 dziesiątek złotych, należy z central Warszawa, Wrocław i Gdańsk rozesłać do punktów sprzedaży Szczecin, Poznań, Olsztyn, Katowice i Rzeszów produkty w następujących proporcjach:

$_z\backslash^{do}$	Szczecin	Poznań	Olsztyn	Katowice	Rzeszów
Warszawa	0	0	20	5	15
Wrocław	5	10	0	25	0
Gdańsk	20	0	0	0	0

A.16 Problem produkcji samochodów

Fabryka samochodów produkuje samochody osobowe i ciężarowe. Fabryka składa się z czterech oddziałów: 1-budowa nadwozi, 2-budowa silników, 3-montaż samochodów osobowych, 4-montaż samochodów ciężarowych. Miesięcznie zdolności produkcyjne tych oddziałów są następujące. Pierwszy oddział może wyprodukować maksymalnie 1000 nadwozi do samochodów osobowych oraz 2000 nadwozi do samochodów ciężarowych. Drugi oddział może wyprodukować maksymalnie 1200 silników do samochodów osobowych oraz 1440 silników do samochodów ciężarowych. Trzeci oddział może zmontować maksymalnie 1100 samochodów osobowych. Natomiast czwarty oddział może zmontować maksymalnie 1200 samochodów ciężarowych. Zysk przy produkcji samochodu osobowego wynosi 3200 złotych, a przy produkcji samochodu ciężarowego 2200 złotych. Ile fabryka powinna produkować miesięcznie samochodów osobowych a ile ciężarowych, aby zysk z produkcji był największy?

Odp: Maksymalny zysk 5840000 złotych uzyskamy, jeżeli fabryka wyprodukuje co miesiąc 1000 samochodów osobowych i 1200 samochodów ciężarowych.

A.17 Problem transportu koni

Pewna firma jest właścicielem pięciu stadnin koni: A, B, C, D i E. Po inwentaryzacji stwierdzono, że w stadninie A jest o 8, a w stadninie B o 6 koni za dużo. Konie te postanowiono rozdzielić między pozostałe gospodarstwa następująco: 5 koni posłać do gospodarstwa C, 5 do D oraz 4 do E. Koszty transportu koni pomiędzy stadninami w dziesiątkach złotych podaje następująca tabela.

$_z\backslash^{do}$	C	D	E
A	16	10	15
B	10	12	10

Jak rozesłać konie z gospodarstw A i B, aby koszt transportu był najmniejszy?

Odp: Aby uzyskać minimalny koszt transportu koni wynoszący 155 dziesiątek złotych, należy ze stadnin A i B przetransportować do stadnin C, D i E konie w następujących ilościach:

$_z\backslash^{do}$	C	D	E
A	0	5	3
B	5	0	1

A.18 Problem cięcia bel materiałów tekstylnych

Producent materiałów tekstylnych dostarcza swoje wyroby klientom cięte w belach, przy czym posiada on dwie maszyny tnące A i B. Maszyna A może ciąć standardową belę o szerokości 250 cm, a maszyna B może ciąć standardową belę o szerokości 200 cm. Bele są sprzedawane w kilku szerokościach proponowanych przez klientów. Zamówienia określają więc żądane szerokości oraz wymieniają ilość bel jaka ma być dostarczona z każdej szerokości. Przy dopasowaniu listy zamówień do dostępnych bel standardowych i maszyn tnących, zwykle zdarza się, że na skutek obcinania zbywających kawałków końcowych straty są nieuniknione. Problem polega więc na dopasowaniu zamówień do bel i maszyn w taki sposób, aby obniżyć straty do absolutnego minimum. Zgodnie z ostatnimi zamówieniami, należy pociąć standardowe bele w taki sposób, aby otrzymać: 862 bele o szerokości 112 cm, 341 bel o szerokości 90 cm, 87 bel o szerokości 77 cm i 216 bel o szerokości 35 cm. Zakładamy, że jest dostępne tyle standardowych bel, ile jest konieczne i że wszystkie cięcia wykonywane są celem uzyskania tylko bel o zamówionych szerokościach.

A.19 Problem najkrótszej drogi

Wyznaczyć najkrótszą drogę z Warszawy do Sofii, korzystając z sieci połączeń przedstawionej w poniższej tabeli, gdzie w przypadku istniejącego połączenia pomiędzy miastami, podane są odległości (w kilometrach). Natomiast jeśli pomiędzy dwoma miastami nie ma bezpośredniego połączenia - zamiast odległości umieszczono słowo: *brak*.

Miasta	Warszawa	Katowice	Zakopane	Lwów	Wiedeń	Budapeszt	Bukareszt	Zagrzeb	Sofia
Warszawa	0	300	402	356	brak	brak	brak	brak	brak
Katowice	300	0	brak	brak	440	474	brak	brak	brak
Zakopane	402	brak	0	brak	brak	330	brak	brak	brak
Lwów	356	brak	brak	0	brak	brak	823	brak	brak
Wiedeń	brak	440	brak	brak	0	brak	brak	430	brak
Budapeszt	brak	474	330	brak	brak	0	813	365	774
Bukareszt	brak	brak	brak	823	brak	813	0	brak	403
Zagrzeb	brak	brak	brak	brak	430	365	brak	0	768
Sofia	brak	brak	brak	brak	brak	774	403	768	0

Odp: Najkrótsza droga z Warszawy do Sofii ma długość 1506 km i prowadzi przez Zakopane oraz Budapeszt.

A.20 Problem przewidywania liczebności populacji królików

W roku 1202 Leonardo Fibonacci sformułował następujący, obecnie bardzo popularny w informatyce problem, dotyczący rozmnażania się królików. Na początku mamy parę nowonarodzonych królików i o każdej parze królików zakładamy, że:

- nowa para staje się płodna po miesiącu życia,
- każda płodna para rodzi jedną parę nowych królików w miesiącu,
- króliki nigdy nie umierają.

W oparciu o powyższe warunki, Fibonacci sformułował następujące pytanie: ile będzie par królików po upływie roku, które można uogólnić pytając o to: ile będzie par królików po upływie n miesięcy? Liczbę tę zwykle oznacza się przez F_n, jest ona nazywana liczbą Fibonacciego. Przeprowadzić symulację mająca na celu stwierdzenie: po ilu miesiącach populacja królików osiągnie: 100, 1000, 10000, 100000 oraz 1 milion par.

A.21 Problem przewidywania wzrostu PKB

W roku 1998 produkt krajowy brutto (w skrócie PKB) Polski wyniósł 6000 dolarów, a przyrost PKB utrzymywał się na poziomie 6 procent w skali roku. Załóżmy, że przyrost PKB Polski w następnych latach utrzyma się na takim samym poziomie. Ile lat musi upłynąć, aby PKB Polski podwoił się? W roku 1998 PKB Niemiec wyniósł 12000 dolarów, a przyrost PKB utrzymywał się na poziomie 1.2 procenta w skali roku. Załóżmy, że przyrost PKB Niemiec w następnych latach utrzyma się na takim samym poziomie. Czy jest szansa, aby w ciągu następnych 20 lat PKB Polski dorównał PKB Niemiec?

Odp:

PKB Polski podwoi się po `12` latach.
PKB Polski dorówna PKB Niemiec po `15` latach.

A.22 Problem przewidywania oprocentowania od lokaty

Dysponujesz kwotą `x` złotych. Zaoferowano Ci umieszczenie tej kwoty na lokacie terminowej oprocentowanej `p` procent w skali roku z kapitalizacją odsetek co `6` miesięcy. Jakiej kwoty możesz się spodziewać na swoim koncie po trzech miesiącach, dziewięciu miesiącach, po roku, po dwóch latach oraz po trzech latach?

Wskazówka:

Jeśli `x` oznacza wkład początkowy a `p` oprocentowanie w skali roku na lokacie terminowej z kapitalizacją odsetek co `6` miesięcy, to po pół roku na koncie będzie kwota `x1 = x + 0.5*p*x` a po roku kwota `x2 = x1 + 0.5*p*x1`.

A.23 Problem przewidywania przebiegu epidemii

W pewnej zamkniętej społeczności liczącej 100000 osób pojawiło się 10 osób chorych na katar, co spowodowało „epidemię kataru”. Wiedząc, że spośród 10 osób chorych na katar każda zaraża codziennie jeszcze dwie osoby, podać przewidywany przebieg epidemii. W szczególności odpowiedzieć na pytania: kiedy będzie najwięcej chorych i kiedy epidemia wygaśnie z powodu braku osób podatnych na zachorowanie? Założyć przy tym, że katar trwa 7 dni od dnia zarażenia i przez ten okres chorzy mogą zarażać inne osoby. Oprócz tego osoby, które wyzdrowiały nie mogą już zachorować. Jak zmieni się przebieg epidemii jeśli dopuścimy, że po 2 tygodniach po wyzdrowieniu, znowu można zachorować na katar?

A.24 Problem rozmnażania bakterii

Rozważmy proces rozmnażania bakterii. Zakładamy, że bakterie znajdują się w środowisku o stałych parametrach, w związku z czym szybkość i sposób ich rozmnażania jest stały. Wiadomo, że bakterie rozmnażają się przez podział, w którego wyniku z jednej bakterii powstaje 2 nowe. Podział ten następuje co 2 minuty. Przyjąć, że wszystkie bakterie dzielą się w tych samych chwilach. Po jakim czasie liczba bakterii zwiększy się 1000 razy, 10000 razy, 100000 razy oraz 1 mln razy.

A.25 Problem łososi i rekinów

Pewna rodzina łososi rozwija się zgodnie z prawem Malthusa, co można wyrazić w ten sposób, że jeśli wielkość populacji w chwili t wynosi $p(t)$, to w wyniku naturalnego rozrodu populacji w chwili $t+1$ jej wielkość wynosi:

$$p(t+1) = p(t) \cdot e^{0.003}$$

gdzie t jest czasem mierzonym w minutach. Rekiny, które zagnieździły się na wodach zasiedlonych przez łososie, zjadają 2 promile populacji łososi na minutę. Poza tym, ze względu na niekorzystne warunki 2 promile łososi z całej populacji na minutę odpływa z tych wód. Wiedząc, że w chwili początkowej było milion łososi, odpowiedzieć na pytanie jak będzie zmieniać się liczebność populacji łososi tzn. jeśli liczebność będzie się zmniejszać, to kiedy osiągnie jakieś istotne progi (np. 1/2, 1/3, 1/10, 1/100 początkowej liczebności) oraz czy i kiedy populacja łososi na omawianym obszarze wymrze. Za moment wymarcia populacji łososi uważamy sytuację, kiedy liczebność łososi spadnie poniżej 100 sztuk.

A.26 Problem utylizacji odpadów

Zakład utylizacji odpadów posiada pewną liczbę jednolitrowych butelek, które oryginalnie zawierały roztwór toksycznego związku chemicznego X o stężeniu 0.5 (50 procent związku X + 50 procent wody). Butelki są zgodnie z normą puste, ale w każdej z pozostaje 1 mililitr roztworu. Każda butelka przechodzi następujący proces płukania: dodaj V mililitrów wody, wymieszaj i wylej roztwór do cysterny. Przyjmujemy, że po wylaniu roztworu w butelce pozostaje go 1 mililitr. Proces jest powtarzany tak długo, aż stężenie X w pozostającym roztworze będzie niższe od 0.00000001. Z każdym krokiem płukania wiąże się koszt wykonanej pracy równy 0.25 grosza, podobnie pewien koszt jest związany z wylaniem roztworu do cysterny (1.5 grosza na litr wylanego roztworu). Zadanie polega na znalezieniu optymalnej zawartości V ze względu na koszty przy uwzględnieniu stężenia początkowego i wymaganego.

A.27 Problem przelewania wody

Dane są dwa naczynia, które mogą pomieścić odpowiednio 3 litry i 4 litry wody. Naczynia nie mają miarki. Można je napełniać przy użyciu pompy lub przelewając wodę z jednego w drugie. W jaki sposób można otrzymać dokładnie 2 litry wody w naczyniu czterolitrowym?

Odp: Niech para `(a,b)`, gdzie: `a` – ilość wody w naczyniu 3 litrowym, `b` – ilość wody w naczyniu 4 litrowym, reprezentuje stan ze środowiska problemu. Oto przykładowa i najkrótsza droga od stanu początkowego do stanu końcowego: `(0,0) (3,0) (0,3) (3,3) (2,4) (2,0) (0,2)`.

A.28 Problem wydawania reszty

Jak wydawać resztę za pomocą możliwie najmniejszej liczby monet o nominałach 1gr, 2gr, 5gr, 10gr, 20gr, 50gr, 1zł, 2zł i 5zł. Skonstruuj algorytm, do którego na wejście podajemy pewną sumę pieniędzy, czyli resztę, jaką trzeba wydawać; natomiast na wyjściu wypisywane są monety za pomocą których należy tę sumę wydać.

A.29 Problem przeprawy przez rzekę misjonarzy i ludożerców

Pewnego dnia, trzech misjonarzy i trzech ludożerców znalazło się na jednym brzegu rzeki z zamiarem przeprawy na jej drugą stronę. Misjonarze nie byli jednak pewni czy przypadkiem ludożercy oprócz samej przeprawy przez rzekę nie mają jeszcze innych planów. Wiedzieli bowiem, że ludożercy są bardzo głodni. Dlatego misjonarze wymyślili, że przeprawa przez rzekę musi tak się odbywać, aby w dowolnym jej momencie liczba misjonarzy po obydwu stronach rzeki była zawsze nie mniejsza od liczby ludożerców po tej samej stronie rzeki. Wtedy bowiem misjonarze mieli czuć się bezpieczniej uważając, że w takiej sytuacji zjedzenie kogoś z nich przez ludożerców jest mało prawdopodobne. Jednak łódka służąca do przeprawy mogła pomieścić tylko dwie osoby. Jak więc powinna odbyć się przeprawa misjonarzy i ludożerców, aby ryzyko zjedzenia misjonarzy przez ludożerców było jak najmniejsze?

Odp: Oto przykładowa i najkrótsza droga od stanu początkowego do stanu końcowego:

m	l	łódka		m	l
3	3	→		0	0
3	1		←	0	2
3	2	→		0	1
3	0		←	0	3
3	1	→		0	2
1	1		←	2	2
2	2	→		1	1
0	2		←	3	1
0	3	→		3	0
0	1		←	3	2
1	1	→		2	2
0	0		←	3	3

A.30 Problem przeprawy przez rzekę wilka, kozy i kapusty

W jaki sposób można przewieźć przez rzekę wilka, kozę i kapustę bez żadnego dla nich uszczerbku łódką, która ma tylko dwa miejsca, tzn. jedno miejsce dla przewoźnika i drugie miejsce dla jednej z przewożonych rzeczy? Wiadomo jednak, że jeśli na jednym brzegu rzeki pozostanie bez opieki przewoźnika wilk i koza, to wilk zje kozę; natomiast jeśli na jednym brzegu rzeki pozostanie bez opieki przewoźnika koza i kapusta, to koza zje kapustę.

Odp: Niech trójka `(a,b,c)` – gdzie: jeśli `a` ma wartość `0`, to wilk znajduje się na lewym brzegu rzeki, jeśli `b` ma wartość `0`, to koza znajduje się na lewym brzegu rzeki, jeśli `c` ma wartość `0`, to kapusta znajduje się na lewym brzegu rzeki, jeśli zmienne `a,b,c` mają wartość `1`, to odpowiadające im obiekty ze środowiska problemu znajdują się na prawym brzegu rzeki – reprezentuje stan ze środowiska problemu. Oto przykładowa i najkrótsza droga od stanu początkowego do stanu końcowego: `(0,0,0) (0,1,0) (0,1,1) (0,0,1) (1,0,1) (1,1,1)`.

A.31 Problem syntezy związku chemicznego

Podczas procesu technologicznego prowadzona jest synteza pewnego związku chemicznego T z substratu S. Jednak związek T nie może być otrzymywany bezpośrednio ze związku S, gdyż potrzebne są pewne pośrednie reakcje chemiczne podczas których udział biorą także inne związki

chemiczne: A, B, C, D, E i F. Wszystkie możliwe reakcje chemiczne dotyczące opisywanego procesu technologicznego wraz z ich oczekiwanymi wydajnościami zestawione są w następującej tabeli.

Reakcja chemiczna	Współczynnik wydajności reakcji
S ⇒ A	0.40 (40%)
S ⇒ D	0.70 (70%)
A ⇒ B	0.65 (65%)
A ⇒ D	0.80 (80%)
C ⇒ B	0.70 (70%)
D ⇒ E	0.45 (45%)
B ⇒ E	0.85 (85%)
E ⇒ F	0.66 (66%)
E ⇒ T	0.66 (66%)

Za pomocą jakiego ciągu reakcji chemicznych należy otrzymywać związek T ze związku S, aby otrzymać maksymalny współczynnik wydajności przeprowadzonych reakcji chemicznych? Przyjąć, że współczynnik wydajności dwóch następujących po sobie reakcji chemicznych jest równy iloczynowi współczynników wydajności tych reakcji.

Literatura

[1] Banachowski, L., Kreczmar, A.: *Elementy analizy algorytmów*, Warszawa: WNT (1989).

[2] Banachowski, L., Diks, K., Rytter, W.: *Algorytmy i struktury danych*, Warszawa: WNT (1996).

[3] Bolc, L., Cytowski, J.: *Metody przeszukiwania heurystycznego*, Warszawa: PWN (1989), tom I.

[4] Bolc, L., Cytowski, J.: *Metody przeszukiwania heurystycznego*, Warszawa: PWN (1991), tom II.

[5] Cormen, T., H., Leiserson, C., E., Rivest, R., L.: *Wprowadzenie do algorytmów*, Warszawa: WNT (1998).

[6] Harel, D.: *Rzecz o istocie informatyki - Algorytmika*, Warszawa: WNT (1992).

[7] Hippe, Z.: *Zastosowanie sztucznej inteligencji w chemii*, Warszawa: PWN (1993).

[8] Liengme, B., V.: *Excel w nauce i technice*, Warszawa: Wydawnictwo RM (2002).

[9] Lipski, W.: *Kombinatoryka dla programistów*, Warszawa: WNT (1983).

[10] Michalewicz, Z., Fogel, D., B.: *How to Solve It: Modern Heuristics*, Berlin: Springer-Verlag (1998).

[11] Palczewski, A.: *Równania różniczkowe zwyczajne - teoria i metody numeryczne z wykorzystaniem komputerowego systemu obliczeń symbolicznych*, Warszawa: WNT (1999).

[12] Polya, G.: *Odkrycia matematyczne*, Warszawa: WNT (1975).

[13] Polya, G.: *Jak to rozwiązać?*, Warszawa: PWN (1993), wydanie drugie.

[14] Rich, E., Knight, K.: *Artificial Inteligence*, New York: McGraw-Hill, Inc. (1991), second edition.

[15] Reingold, E., M., Nievergelt, J., Deo, N.: *Algorytmy kombinatoryczne*, Warszawa: PWN (1985).

[16] Suraj, Z., Rumak, T.: *Algorytmiczne rozwiązywanie zadań i problemów*, Rzeszów: FOSZE (1995).

[17] Sysło, M.: *Algorytmy*, Warszawa: WSiP (1997).

[18] Sysło, M.: *Piramidy, szyszki i inne konstrukcje algorytmiczne*, Warszawa: WSiP (1998).

[19] *Decyzje menadżerskie z Excelem*, praca zbiorowa pod readakcją Tomasza Szapiro, Warszawa: Polskie Wydawnictwo Ekonomiczne (2000).

[20] Wilson, R.: *Wprowadzenie do teorii grafów*, Warszawa: PWN (1998).

[21] Zieliński, R., Neumann, P.: *Stochastyczne metody poszukiwania minimum funkcji*, Warszawa: WNT (1986).