



Repetitorium licencjackie

Egzamin z informatyki na MIMUW

Od autorów

Obszerna treść tej publikacji ma za zadanie dobrze przygotować Cię do egzaminu licencjackiego na MIM-ie. Można ją traktować jako coś na kształt repetytorium maturalnego, bo to właśnie na nich wzorowaliśmy się najbardziej, pisząc i składając ten dokument. W związku z tym wewnątrz nie zabraknie tak przyjemnych i błahych elementów, jak kolorowe ramki i pogrubienia ważnych pojęć. Oraz, co najważniejsze, mnóstwo przykładów! Zależało nam, aby wszystko było jak najbardziej uporządkowane i jasne, żeby przyspieszyć i uatrakcyjnić proces powtórki.

Ponieważ całość jest rozwijana przez dobrowolne kontrybucje, mogą pojawić się tu różne błędy: zarówno literówki, jak i poważniejsze usterki. Wszelkie uwagi (w tym sugestie, co poprawić, gdzie warto napisać dokładniejsze wyjaśnienie, gdzie umieścić rysunek itp.) są mile widziane i z całego serca będziemy za nie wdzięczni! Śmiało zgłaszajcie je w zakładce [Issues w tym repozytorium na GitHubie](#). Niech ta publikacja stanowi nasze wspólne dobro i służy najbliższym pokoleniom!

Przy okazji, jeśli chcecie przyczynić się do (nawet drobnego) rozwoju repetytorium, więcej informacji znajdziecie w pliku *readme* we wcześniej wspomnianym repozytorium.

■ Formuła egzaminu licencjackiego

Egzamin licencjacki jest zwieńczeniem 3-letnich studiów licencjackich, a dodatkowo stanowi jednocześnie egzamin wstępny na studia magisterskie z informatyki. Składa się z **30 zadań**, na rozwiązanie których jest przeznaczone **150 minut**.

Pytania na egzaminie dotyczą wybranych zagadnień z przedmiotów obowiązkowych, do których tutaj odnosić się będziemy jako „podstawa programowa”. Można ją zobaczyć na [stronie wydziału](#), ale wszystkie jej fragmenty znajdują się także w tym dokumencie przy odpowiednich rozdziałach.

W każdym spośród 30 zadań podane są trzy warianty: **A**, **B** i **C**. W kratce przy każdym z wariantów należy odpowiedzieć, czy jest on prawdziwy, wpisując drukowanymi literami **TAK** albo **NIE**. W przypadku omyłkowego wpisu kratkę należy przekreślić i napisać jedno z tych słów po jej lewej stronie.

Oto przykład poprawnie rozwiązanej zadania:

1. Każda liczba całkowita postaci $10^n - 1$, gdzie n jest całkowite i dodatnie

TAK A. dzieli się przez 9

NIE B. jest pierwsza

TAK C. jest nieparzysta

Na teście nie pojawiają się żadne zadania innego typu niż wyżej pokazane.

■ Zasady punktowania

Zdający zdobywa punkty „duże” (od 0 do 30) oraz „małe” (od 0 do 90):

- jeden punkt „duży” jest przyznawany za zadanie, w którym poprawnie wskazana jest prawdziwość albo fałszywość **wszystkich trzech wariantów**;
- jeden punkt „mały” jest przyznawany za każde poprawne wskazanie prawdziwości albo fałszu **pojedynczego wariantu odpowiedzi**.

Oznacza to, że całkowicie poprawnie rozwiązane zadanie zwiększa nasz wynik o trzy „małe” punkty i jeden „duży” punkt.

Ostatecznym wynikiem egzaminu jest liczba $D + 0.01m$, gdzie D oznacza liczbę „dużych”, a m liczbę „małych” punktów. Na przykład, wynik 5.50 oznacza, że kandydat poprawnie wskazał w całym teście prawdziwość albo fałszywość łącznie 50 wariantów odpowiedzi, w tym każdego z trzech wariantów dla pewnych pięciu zadań. Z egzaminu nie można zdobyć więcej niż 30 punktów (wszystkie wyższe wyniki są obcinane w dół do 30).

Zasadniczą rolę w ostatecznym wyniku testu mają więc punkty „duże”. Punkty „małe” służą jedynie rozstrzyganiu sytuacji, w których wielu kandydatów dostało tyle samo „dużych” punktów.

Układ repetytorium

Publikacja podzielona jest na rozdziały względem przedmiotów obowiązkowych, a każdy z nich na podrozdziały o różnej tematyce materiału. Podział materiału jest ściśle skorelowany z podstawą programową egzaminu, która jest zaprezentowana na pierwszej stronie każdego rozdziału, oraz z zadaniami występującymi na poprzednich egzaminach.

Podrozdziały zaczynają się od **teoretycznego wstępu**, przeplatанego wieloma przykładami oraz okazjonalnymi wstawkami **To było na egzaminie**, zawierającymi najbardziej reprezentatywne pytania z archiwalnych egzaminów wraz ze sposobem ich rozwiązania.

Po części teoretycznej każdego podrozdziału zostały umieszczone **zestawy tematycznych zadań** do samodzielnego rozwiązania, wraz z **rozwiązaniami** umieszczonymi na samym końcu rozdziału. **Wskazówka: aby szybko przenieść się do rozwiązania danego zadania (w elektronicznej wersji dokumentu), wystarczy kliknąć w jego numer!**

Na sam koniec, chcielibyśmy podziękować wszystkim poprzednim pokoleniom, których prace archiwizacyjne nieświadomie przyczyniły się do powstania tej publikacji (jako że jedynie niewielka część poprzednich egzaminów jest *oficjalnie* udostępniana). Życzymy miłej pracy z repetytorium i smacznej kawusi!

Autorzy

Spis treści

1. Analiza matematyczna	9
1.1. Granica ciągu	9
1.2. Szeregi o wyrazach dodatnich	12
1.3. Szeregi o wyrazach dowolnych	13
1.4. Ciągłość funkcji	14
1.5. Pochodna	16
1.6. Twierdzenie Lagrange'a o wartości średniej	19
1.7. Wzór Taylora	20
1.8. Zbieżność ciągów liczbowych i funkcyjnych	22
1.9. Ekstrema funkcji	24
1.10. Całki	25
1.11. Elementy teorii miary	28
1.12. Rozwiązania	28
2. Geometria z algabadą liniową	37
2.1. Grupy i ciała	37
2.2. Liczby zespolone	39
2.3. Przestrzenie liniowe	43
2.4. Macierze i przekształcenia liniowe	49
2.5. Wektory i wartości własne	56
2.6. Układy równań liniowych	56
2.7. Przestrzenie z iloczynem skalarnym	57
2.8. Rozwiązania	59
3. Podstawy matematyki	67
3.1. Działania na zbiorach	67
3.2. Funkcje i ich własności	68
3.3. Relacje równoważności i ich własności	70
3.4. Moce zbiorów	72
3.5. Porządkи częściowe i ich własności	74
3.6. Dobre ufundowanie i indukcja	77
3.7. Rachunek zdań	78
3.8. Logika pierwszego rzędu	79
3.9. Rozwiązania	80
4. Matematyka dyskretna	89
4.1. Sumy i współczynniki dwumianowe	89
4.2. Permutacje, liczby Stirlinga	91
4.3. Funkcje tworzące	93
4.4. Metody zliczania	96
4.5. Teoria grafów	98
4.6. Teoria liczb	103
4.7. Asymptotyka	107
4.8. Rozwiązania	108
5. Rachunek prawdopodobieństwa	117

5.1.	Prawdopodobieństwo warunkowe i całkowite	117
5.2.	Dyskretne zmienne losowe	124
5.3.	Parametry rozkładu	127
5.4.	Nierówności probabilistyczne	132
5.5.	Ciągły rozkład prawdopodobieństwa	134
5.6.	Łańcuchy Markowa	140
5.7.	Rozwiązania	141
6. Algorytmy i struktury danych		149
6.1.	Poprawność i efektywność algorytmów	149
6.2.	Kolejki priorytetowe	152
6.3.	Algorytmy sortowania	155
6.4.	Algorytmy grafowe	160
6.5.	Drzewa BST	163
6.6.	Algorytmy tekstowe	165
6.7.	Co to	167
6.8.	Rozwiązania	167
7. Języki, automaty i obliczenia		175
7.1.	Języki regularne	175
7.2.	Języki bezkontekstowe	178
7.3.	Języki obliczalne	182
7.4.	Klasy P, NP oraz NP-zupełność	184
7.5.	Rozwiązania	186
8. Bazy danych		193
8.1.	Relacyjny model danych	193
8.2.	Podstawowe konstrukcje języka SQL	197
8.3.	Zależności funkcyjne	204
8.4.	Redundancja, normalizacja	206
8.5.	Transakcje i współbieżność	210
8.6.	Fizyczna reprezentacja danych	212
8.7.	Rozwiązania	214
9. Programowanie współbieżne		221
9.1.	Podstawy programowania współbieżnego	221
9.2.	Semafora	226
9.3.	Monitory	229
9.4.	Komunikacja synchroniczna i asynchroniczna	231
9.5.	Klasyczne problemy współbieżności	233
9.6.	Systemy rozproszone	238
9.7.	Rozwiązania	239
10 Metody numeryczne		247
10.1.	Algorytmy rozkładu macierzy	247
10.2.	Uwarunkowanie i numeryczna poprawność	251
10.3.	Metody iteracyjne	252
10.4.	Interpolacja wielomianowa	253
10.5.	Aproksymacja jednostajna	255
10.6.	Metody numeryczne całkowania	258
10.7.	Rozwiązania	259
11 Programowanie obiektowe		263
11.1.	Klasy, obiekty i konstruktory	263
11.2.	Kapsułkowanie danych i zakresy widoczności	266
11.3.	Dziedziczenie i hierarchie klas. Polimorfizm	268
11.4.	Klasy abstrakcyjne i interfejsy	279

11.5. Wyjątki	281
11.6. Standardowe kolekcje w Javie	285
11.7. Rozwiązania	286
12JPP i blok JNP	293
12.1. Konstrukcje programistyczne C i C++	293
12.2. Znajomość technik i narzędzi tworzenia oprogramowania	301
12.3. Rozwiązania	302
13Aplikacje WWW	307
13.1. HTML i CSS	307
13.2. JavaScript	312
13.3. Mechanizmy budowania aplikacji internetowych	317
13.4. Django	322
13.5. Rozwiązania	323
14Sieci komputerowe	327
14.1. Warstwy sieci	327
14.2. Protokoły TCP, UDP, IP, ICMP, Ethernet	331
14.3. Adresy internetowe, tablice tras, zasady trasowania, NAT	337
14.4. Systemy nazw domenowych	340
14.5. Sieciowy interfejs gniazd	341
14.6. Rozwiązania	341
15Systemy operacyjne	345
15.1. Programowanie niskopoziomowe	345
15.2. Szeregowanie procesów	348
15.3. Zakleszczenia	355
15.4. Zarządzanie pamięcią	357
15.5. System plików	361
15.6. Rozwiązania	361
16Bezpieczeństwo systemów komputerowych	367
16.1. Klucz symetryczny	367
16.2. Protokół Diffiego-Hellmana, problem logarytmu dyskretnego	370
16.3. Kryptografia klucza publicznego	371
16.4. Zastosowania kryptografii klasycznej	373
16.5. Ataki i obrona	373
16.6. Rozwiązania	375

1

Analiza matematyczna

Materiały teoretyczne z analizy matematycznej zostały opracowane na podstawie skryptu Pawła Strzeleckiego, oryginalnych prac Eulera, Weierstrassa, Darboux, Rolle'a i Lagrange'a oraz tego dokumentu.

Podstawa programowa

1. **Granica ciągu.**
2. **Szeregi** liczbowe, zbieżność bezwzględna i warunkowa.
3. **Ciągłość i jednostajna ciągłość** funkcji i najważniejsze własności funkcji ciągłych.
4. **Pochodna funkcji jednej zmiennej**, interpretacja geometryczna i mechaniczna.
5. **Twierdzenie Lagrange'a o wartości średniej**, jego interpretacja geometryczna i niektóre konsekwencje (monotoniczność, wklęsłość, wypukłość, szacowanie przyrostów).
6. **Wzór Taylora** dla funkcji jednej zmiennej, zastosowania do obliczeń przybliżonych, rozwijanie funkcji w **szeregi potęgowe**.
7. Pojęcie **zbieżności ciągów** liczbowych i funkcyjnych, twierdzenia o przejściu do granicy pod znakiem pochodnej i całki.
8. **Ekstrema funkcji** jednej i kilku zmiennych rzeczywistych: warunki konieczne i dostateczne.
9. **Funkcja pierwotna, całka oznaczona**. Zastosowania geometryczne całki.
10. **Elementy teorii miary.**

1.1.

Granica ciągu

■ Podstawowe informacje

Ciąg $(a_n) \subset \mathbb{R}$ jest **zbieżny do granicy** $a \in \mathbb{R} \iff$

$$\forall \varepsilon > 0 \quad \exists N_0 \in \mathbb{N} \quad \forall n > N_0 : \quad |a_n - a| < \varepsilon$$

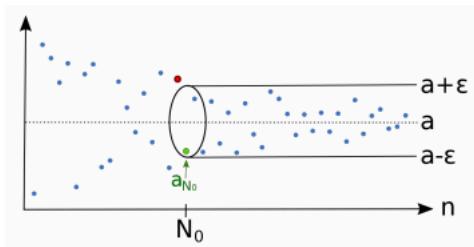
zapisujemy

$$\lim_{n \rightarrow \infty} a_n = a, \quad a_n \rightarrow a \text{ (dla } n \rightarrow \infty\text{)}$$

dla liczb zespolonych analogicznie.

Wnioski

- Każdy ciąg **zbieżny jest ograniczony** (poza ustalonym paskiem jest tylko skończenie wiele a_n).
- Ciąg zbieżny ma **jedną** granicę.



Przykład 1.

$$\lim_{n \rightarrow \infty} \frac{1}{n} = 0, \quad \lim_{n \rightarrow \infty} \sqrt[n]{n} = 1, \quad \text{dla } q \in (-1, 1) \quad \lim_{n \rightarrow \infty} q^n = 0, \quad \lim_{n \rightarrow \infty} n^k q^n = 0$$

Własności arytmetyczne granicy: Jeśli $a_n \rightarrow a$ oraz $b_n \rightarrow b$, to

- $a_n \pm b_n \rightarrow a \pm b$
- $a_n \cdot b_n \rightarrow a \cdot b$
- $\frac{a_n}{b_n} \rightarrow \frac{a}{b}$ o ile $b_n, b \neq 0$

Ważne twierdzenia

Twierdzenie o trzech ciągach/dwóch policjantach: Jeśli $b_n \leq a_n \leq c_n$ dla $n \geq N_1$ oraz $b_n, c_n \rightarrow a$, to także $a_n \rightarrow a$

Przykład 2.

- $\sqrt[3]{69} \rightarrow 1$, bo dla $n > 69$ zachodzi $1 < \sqrt[3]{69} < \sqrt[3]{n}$ oraz $1 \rightarrow 1$, $\sqrt[3]{n} \rightarrow 1$
- $\sqrt[3]{3^n + 5^n + 8^n} \rightarrow 8$, bo $8 = \sqrt[3]{8^n} \leq \sqrt[3]{3^n + 5^n + 8^n} \leq \sqrt[3]{3 * 8^n} = \sqrt[3]{3} * 8 \rightarrow 8$

To było na egzaminie

Ciąg, którego n -ty wyraz zadany jest wzorem $\sqrt{n^2 + (-1)^n} - n$

- A. jest monotoniczny
 B. jest rozbieżny
 C. jest nieograniczony

Zastosujemy twierdzenie o trzech ciągach. Widzimy, że $\sqrt{n^2 - 1} - n \leq \sqrt{n^2 - (-1)^n} - n \leq \sqrt{n^2 + 1} - n$. Pozostaje tylko pokazać granice policjantów. Mamy $\sqrt{n^2 - 1} - n = \frac{(\sqrt{n^2 - 1} - n)(\sqrt{n^2 - 1} + n)}{\sqrt{n^2 - 1} + n} = \frac{-1}{\sqrt{n^2 - 1} + n} \rightarrow 0$. Dla (+1) tak samo. W takim razie B i C to NIE. Czy ciąg jest monotoniczny? $a_1 = -1 < 0$, $a_2 = \sqrt{4 + 1} - 2 > 0$, $a_3 = \sqrt{9 - 1} - 3 = \sqrt{8} - 3 < 0$. Nie jest. Zatem A to też NIE.

Twierdzenie Bolzano-Weierstrassza: Każdy ciąg ograniczony ma podciąg zbieżny.

Twierdzenie: Każdy ciąg monotoniczny i ograniczony ma granicę (skończoną).

Przykład 3.

- $a_1 = 2, \quad a_{n+1} = \sqrt{6 + a_n} \implies a_n \rightarrow 3$
- $b_1 = 2, \quad b_{n+1} = 12 + \frac{b_n}{7} \implies b_n \rightarrow 14$
- $c_1 = 2, \quad c_{n+1} = \frac{c_n^2 + 2}{2c_n} \implies c_n \rightarrow \sqrt{2}$

Jak to wykazać? Analiza dla c_n

- Widać, że wszystkie $c_n > 0$
- Monotoniczność: $c_{n+1} < c_n$, po indukcji: $c_{n+1} \stackrel{?}{<} c_n$ wtedy $c_n - c_{n+1} > 0$, czyli $c_n - \frac{c_n^2 + 2}{2c_n} = \frac{2c_n^2 - c_n^2 - 2}{2c_n} = \frac{c_n^2 - 2}{2c_n} > 0$, co zachodzi jedynie gdy $c_n > \sqrt{2}$
- Czy faktycznie $c_n > \sqrt{2}$? Indukcja: $c_1 = 2 > \sqrt{2}$ – baza ok. Zał. $c_n > \sqrt{2}$, $c_{n+1} - \sqrt{2} = \frac{c_n^2 + 2}{2c_n} - \sqrt{2} = \frac{c_n^2 - 2c_n\sqrt{2} + 2}{2c_n} = \frac{(c_n - \sqrt{2})^2}{2c_n} > 0$

Wszystkie wyrazy ciągu są $> \sqrt{2}$ i jest on malejący, zatem $c_n \rightarrow \sqrt{2}$.

Ciągi Cauchy'ego

$(a_n) \subset \mathbb{R}$ spełnia **warunek Cauchy'ego** (jest ciągiem Cauchy'ego) \iff

$$\forall \varepsilon > 0 \quad \exists N_0 \in \mathbb{N} \quad \forall m, n > N_0 : \quad |a_n - a_m| < \varepsilon$$

Intuicyjnie: dalekie wyrazy różnią się dowolnie mało.

Stwierdzenie: każdy ciąg Cauchy'ego jest ograniczony.

Twierdzenie: Ciąg w \mathbb{R} (w \mathbb{C}) jest zbieżny \iff jest ciągiem Cauchy'ego.

Exp i ln

Definicja:

$$e^x = \exp(x) := \lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n$$

Twierdzenie: \exp jest różniczkowalną surjekcją na $(0, \infty)$.

Definicja:

$$\ln = (\exp)^{-1} : (0, \infty) \rightarrow \mathbb{R}$$

Zestaw zadań

1.1. Jeśli ciąg liczb rzeczywistych dodatnich a_n jest monotoniczny i ograniczony, to

- A. ciąg $\sin(a_n)$ jest monotoniczny
- B. ciąg $\cos(a_n)$ jest zbieżny
- C. ciąg $\log(a_n)$ jest ograniczony

1.2. Ciąg, którego n -ty wyraz zadany jest wzorem $\frac{2010^n - n^{2010}}{2^{n^2} - 2010^n}$,

- A. jest zbieżny
- B. ma granicę równą -1

C. ma granicę równą 0

1.3. Niech $\{a_n\}_{n \geq 0}$ będzie ograniczonym ciągiem dodatnich liczb rzeczywistych. Wtedy

A. jeśli $\{\sin(a_n)\}_{n \geq 0}$ jest zbieżny, to a_n jest monotoniczny

B. $\{\exp(-a_n)\}_{n \geq 0}$ posiada podciąg zbieżny do granicy dodatniej

C. jeśli $\{\exp(-a_n)\}_{n \geq 0}$ jest zbieżny, to $\{\arctan(a_n)\}_{n \geq 0}$ ma granicę dodatnią

1.4. Ciąg określony dla $n \geq 1$ wzorem $(1 + \frac{1}{4^n})^{2^n}$ jest

A. zbieżny do 1

B. rosnący, począwszy od pewnego miejsca

C. ograniczony

1.5. Ciąg $a_n = (0.999 + \frac{1}{4n})^{4n}$

A. jest zbieżny do liczby większej od 0

B. jest zbieżny do liczby niewymiernej

C. jest ograniczony z góry

1.2. Szeregi o wyrazach dodatnich

Definicje

Szereg

$$\sum_{n=1}^{\infty} a_n$$

to **para ciągów**, $(a_n) \subset \mathbb{R}$ (lub \mathbb{C}) oraz

$$s_n = a_1 + \dots + a_n = \sum_{k=1}^n a_k, \quad n \in \mathbb{N},$$

gdzie s_n to **sumy częściowe szeregu** a_n to **wyrazy szeregu**.

Szereg $\sum_{n=1}^{\infty} a_n$ jest **zbieżny** \iff sumy częściowe s_n mają skończoną granicę s. Zapis: $s = \lim_{n \rightarrow \infty} s_n = \sum_{n=1}^{\infty} a_n$

Warunki zbieżności

- warunek konieczny: jeśli szereg $\sum_{n=1}^{\infty} a_n$ jest zbieżny, to $\lim_{n \rightarrow \infty} a_n = 0$
- warunek dostateczny (warunek Cauchy'ego dla szeregów): $\sum_{n=1}^{\infty} a_n$ jest zbieżny \iff zachodzi **warunek Cauchy'ego dla szeregów**

Warunek Cauchy'ego: dla każdego $\varepsilon > 0$ istnieje $n_{\varepsilon} \in \mathbb{N}$ takie, że dla $m > k > n_{\varepsilon}$ jest

$$|a_{k+1} + a_{k+2} + \dots + a_m| < \varepsilon$$

Równoważne warunki:

- szereg $\sum_{n=1}^{\infty} a_n$ jest zbieżny
- Ciąg (s_n) jest ograniczony z góry (**mówimy o szeregach dodatnich**)
- $\sup s_n : n \in \mathbb{N} \in \mathbb{R}$

Przykład 1.

Szereg harmoniczny jest rozbieżny, bo $\frac{1}{n+1} + \frac{1}{n+2} + \dots + \frac{1}{2n} > n \cdot \frac{1}{2n} = \frac{1}{2}$. Każdą paczkę n składników możemy ograniczyć z dołu przez $\frac{1}{2}$. Dokładając kolejne paczki zwiększamy sumy częściowe, które zatem nie są ograniczone z góry.

Kryteria zbieżności szeregów o wyrazach dodatnich

Kryterium zagęszczeniowe

Jeśli $a_1 > a_2 > \dots > 0$, to szeregi

$$\sum_{n=1}^{\infty} a_n \quad \text{i} \quad \sum_{n=1}^{\infty} 2^n a_{2^n}$$

są jednocześnie **oba zbieżne, albooba rozbieżne**.

Przykład 2.

Dla $a_n = \frac{1}{n}$ jest $2^n a_{2^n} = 2^n \cdot \frac{1}{2^n} = 1$. Szereg $\sum 1$ jest rozbieżny, więc $\sum \frac{1}{n}$ także.

Kryterium porównawcze

Niech $a_n, b_n > 0$ oraz $a_n \leq c \cdot b_n$ dla $n \geq n_0$. Wtedy

$$\sum_{n=1}^{\infty} b_n \text{ zbieżny} \implies \sum_{n=1}^{\infty} a_n \text{ zbieżny}, \quad \sum_{n=1}^{\infty} a_n \text{ rozbieżny} \implies \sum_{n=1}^{\infty} b_n \text{ rozbieżny}$$

Kryterium porównawcze asymptotyczne Jeśli $a_n, b_n > 0$ dla $n \geq n_0$ oraz $\lim_{n \rightarrow \infty} \frac{a_n}{b_n} \in (0, +\infty)$, to $\sum a_n$ i $\sum b_n$ są oba zbieżne, **albo** oba rozbieżne.

Przykład 3.

$\sum \frac{1}{n^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \dots + \frac{1}{n^2} < 1 + \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \dots + \frac{1}{n(n-1)} = 1 + (1 - \frac{1}{2}) + (\frac{1}{2} - \frac{1}{3}) + \dots + (\frac{1}{n-1} - \frac{1}{n}) = 2 - \frac{1}{n} < 2$,
czyli sumy częściowe są ograniczone, sztuczka z "teleskopowaniem".

W ogólności $\sum \frac{1}{n^s}$ zbieżny dla $s > 1$.

Zestaw zadań

1.6. $\{S_n\}_{n \geq 1}$ jest ciągiem sum częściowych szeregu rozbieżnego. Wynika z tego, że ciąg $\{S_{n+1} - S_n\}_{n \geq 1}$ jest

- A. rozbieżny
- B. nieograniczony
- C. niezbieżny do 0

1.3.

Szeregi o wyrazach dowolnych

Rodzaje zbieżności

- szereg $\sum_{n=1}^{\infty} a_n$ jest **bezwzględnie** zbieżny $\iff \sum_{n=1}^{\infty} |a_n| < \infty$
- Szereg $\sum_{n=1}^{\infty} a_n$ jest **warunkowo** zbieżny \iff jest zbieżny, ale **nie bezwzględnie**

Naturalnie szereg zbieżny bezwzględnie jest zbieżny.

Kryterium Leibniza

Twierdzenie: jeśli $a_1 \geq a_2 \geq \dots \geq 0$ oraz $a_n \rightarrow 0$, to szereg $\sum_{n=1}^{\infty} (-1)^{n+1} a_n$ jest zbieżny.

Przykład 1.

Szereg anharmoniczny $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$ jest zbieżny (ale nie bezwzględnie). Zbieżność wynika wprost z kryterium Leibniza, a wiemy, że nie jest bezwzględnie zbieżny z działa o szeregach o wyrazach dodatnich.

Kryterium Dirichleta

Twierdzenie Abela:

$$(a_n), (b_n) \subset \mathbb{C}, M > 0.$$

Niech $|A_n| = |a_1 + \dots + a_n| \leq M$ dla $n \in \mathbb{N}$,

$$\sum_{n=1}^{\infty} |b_n - b_{n+1}| < +\infty \text{ oraz } b_n \rightarrow 0 \text{ dla } n \rightarrow \infty$$

Wtedy $\sum a_n b_n$ jest zbieżny. **Wniosek (kryterium Dirichleta)** jeśli $b_1 \geq b_2 \geq \dots \geq 0, b_n \rightarrow 0$ oraz $\sum a_n$ ma sumy częściowe wspólnie ograniczone (jak w tw. Abela), to $\sum a_n b_n$ jest zbieżny.

Przykład 2.

$z \in \mathbb{C}, |z| = 1, z \neq 1$, to $\sum \frac{z^n}{n}$ jest zbieżny. Bierzemy $a_n = z^n$ oraz $b_n = \frac{1}{n}$. Możemy oszacować $|1 + z + z^2 + \dots + z^n| = \frac{1-z^n}{1-z} \leq \frac{2}{1-z}$ ograniczone, b_n oczywiście zbiega do 0.

Zestaw zadań

1.7. Rozważmy szeregi:

$$(A) \sum_{n=1}^{\infty} \frac{7^n + n^9 + \ln(19^{n^2} + 8)}{8^n - 11^{\ln(1+n)} + 2} \quad \text{oraz} \quad (B) \sum_{n=1}^{\infty} \frac{8^n - 11^{\ln(1+n)} + 2}{7^n + n^9 + \ln(19^{n^2} + 8)}$$

- A. ciąg sum częściowych każdego z tych szeregów jest ograniczony
- B. szereg (A) jest zbieżny
- C. szereg (B) jest zbieżny

1.4.

Ciągłość funkcji

Powiemy, że a jest **punktem skupienia** zbioru $A \subset \mathbb{R}$ wtedy i tylko wtedy, gdy istnieje ciąg $(a_n) \subset A - \{a\}$, taki że $a_n \rightarrow a$.

Przykład 1.

- Zbiór \mathbb{Z} nie ma punktów skupienia.
- 0 jest jedynym punktem skupienia zbioru $A = \{\frac{1}{n} : n \in \mathbb{N}\}$.
- Każde $a \in [p, q]$ jest punktem skupienia zbioru (p, q) .

- Każde $a \in \mathbb{Q}$ jest punktem skupienia $\mathbb{R} - \mathbb{Q}$, bo np. $\lim_{n \rightarrow \infty} (a + \frac{\sqrt{2}}{n}) = a$.

Niech a będzie punktem skupienia $A \subset \mathbb{R}$ i weźmy funkcję $f : A \rightarrow \mathbb{R}$. **Granicę funkcji** możemy zdefiniować na dwa sposoby:

Definicja Heinego (ciągowa):

$$\lim_{x \rightarrow a} f(x) = b \iff x_n \rightarrow a \Rightarrow f(x_n) \rightarrow b$$

Definicja Cauchy'ego:

$$\lim_{x \rightarrow a} f(x) = b \iff \forall \varepsilon > 0 \exists \delta > 0 \forall_{x \in A - \{a\}} |x - a| < \delta \Rightarrow |f(x) - b| < \varepsilon$$

Wszystkie arytmetyczne własności granicy ciągu mają zastosowanie w granicach funkcji, włącznie z twierdzeniem o trzech funkcjach.

Przykład 2.

Ważne przykłady granic funkcji:

$$\lim_{x \rightarrow 0} \frac{e^x - 1}{x} = 1 \quad \lim_{x \rightarrow 0} \frac{\sin x}{x} = 1 \quad \lim_{x \rightarrow 0} \frac{\ln(x+1)}{x} = 1 \quad \lim_{x \rightarrow 0} (1+x)^{1/x} = e$$

Niech $d \in D \subset \mathbb{R}$ oraz $f : D \rightarrow \mathbb{R}$. **Definicja Heinego ciągłości funkcji:** funkcja f jest ciągła w punkcie d wtedy i tylko wtedy, gdy dla dowolnego ciągu $(x_n) \subset D$, takiego że $x_n \rightarrow d$, zachodzi $f(x_n) \rightarrow f(d)$.

Zauważmy, że ta definicja bardzo przypomina definicję Heinego granicy. Ale istotne różnice są takie: tu zamiast granicy b jest $f(a)$ oraz tutaj d niekoniecznie musi być punktem skupienia D . Ponadto, może tu zachodzić $x_n = a$ dla pewnych n .

Czasami używa się alternatywnej definicji (Cauchy'ego, która z kolei przypomina definicję Cauchy'ego granicy funkcji). Skoro jednak my zdecydowaliśmy się na definicję ciągłości w wersji Heinego, ta alternatywna definicja będzie dla nas już twierdzeniem.

Twierdzenie (Weierstrassa o przyjmowaniu kresów): Jeśli $f : \mathbb{R} \supset [a, b] \rightarrow \mathbb{R}$ jest ciągła, to istnieją $x_1, x_2 \in [a, b]$ takie, że

$$f(x_1) = \sup_{t \in [a, b]} f(t), \quad f(x_2) = \inf_{t \in [a, b]} f(t)$$

Własność Darboux: Jeśli $f : [a, b] \rightarrow \mathbb{R}$ jest ciągła i dla pewnych liczb $x, y \in [a, b]$, $x < y$, jest

$$f(x) < c < f(y) \quad \text{albo} \quad f(x) > c > f(y)$$

to istnieje $t \in (x, y)$ takie, że $f(t) = c$.

Jednostajna ciągłość

Niech $A \subset \mathbb{R}$. Mówimy, że funkcja $f : A \rightarrow \mathbb{R}$ jest jednostajnie ciągła (na zbiorze A) wtedy i tylko wtedy, gdy dla każdego $\varepsilon > 0$ istnieje taka liczba $\delta > 0$, że dla wszystkich $x, y \in A$ z warunku $|x - y| < \delta$ wynika, że $|f(x) - f(y)| < \varepsilon$.

Warunek Lipschitza: Mówimy, że funkcja $f : A \rightarrow \mathbb{R}$ spełnia warunek Lipschitza (ze stałą L) wtedy i tylko wtedy, gdy dla wszystkich $x, y \in A$ zachodzi nierówność

$$|f(x) - f(y)| \leq L|x - y|$$

Twierdzenie: Jeśli $f : A \rightarrow \mathbb{R}$ spełnia na A warunek Lipschitza to f jest jednostajnie ciągła.

Twierdzenie Cantora o jednostajnej ciągłości: Każda funkcja ciągła $f : [a, b] \rightarrow \mathbb{R}$ jest jednostajnie ciągła na przedziale $[a, b]$.

Przypis redakcji

W części teoretycznej brak informacji o niemal jednostajniej ciągłości, a pojawia się ona w zadaniach.

Zestaw zadań

1.8. Funkcja $f: \mathbb{R} \rightarrow \mathbb{R}$ dana wzorem $f(x) = e^{-x^2} \cdot \sqrt[3]{x} \cdot \sin(e^{x^2})$

- A. jest różniczkowalna na \mathbb{R}
- B. jest jednostajnie ciągła na \mathbb{R}
- C. spełnia warunek Lipschitza na \mathbb{R}

1.9. Dla $n > 0$ określamy

$$a_n = \begin{cases} 1 - \frac{1}{n} & \text{dla } n \text{ nieparzystych} \\ 2 + \frac{1}{n^2} & \text{dla } n \text{ parzystych} \end{cases}$$

Oznaczmy $A = \{a_n \mid n > 0\}$ oraz $-A = \{-a_n \mid n > 0\}$. Wtedy

- A. A zawiera dokładnie jeden ze swoich kresów
- B. A ma dokładnie dwa punkty skupienia
- C. $\sup(-A) = \inf(A)$

1.10. Funkcje $f: (0, 3) \rightarrow \mathbb{R}$ i $g: [0, 3] \rightarrow \mathbb{R}$ są ciągłe oraz $f(1) = g(1) = -7$, $f(2) = g(2) = 7$. Wynika z tego, że

- A. obydwie funkcje są ograniczone
- B. 0 należy do zbiorów wartości obydwu funkcji
- C. obydwie funkcje są jednostajnie ciągłe

1.5.

Pochodna

Definicje

Funkcja $f: \mathbb{R} \supset A \rightarrow \mathbb{R}$ ma pochodną (jest różniczkowalna) w punkcie (skupienia tego zbioru) $a \in A \iff$ istnieje skończona granica

$$\lim_{x \rightarrow a} \frac{f(x) - f(a)}{x - a} =: f'(a)$$

Tak samo definiuje się

- Pochodną funkcji zespolonej $f: \mathbb{C} \supset A \rightarrow \mathbb{C}$, $f'(a) \in \mathbb{C}$
- Pochodną funkcji wektorowej: $f: \mathbb{K} \supset A \rightarrow \mathbb{K}^n$ dla $\mathbb{K} = \mathbb{R}, \mathbb{C}$, $f'(a) \in \mathbb{K}^n$

Interpretacja geometryczna

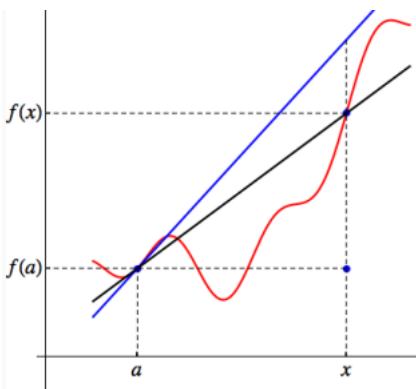
Iloraz różnicowy

$$\frac{f(x) - f(a)}{x - a} = \operatorname{tg} \alpha$$

to tangens kąta α nachylenia siecznej do osi OX.

W granicy, $f'(a)$ to współczynnik kierunkowy stycznej do wykresu f w punkcie $(a, f(a))$.

Prosta o równaniu $y - f(a) = f'(a)(x - a)$ jest styczna do wykresu f .



Naturalna interpretacja wskazuje, że znak pochodnej związany jest z monotonicznością. W szczególności, gdy mamy na wykresie "dolek" - ekstremum lokalne, styczna do funkcji w tym punkcie jest pozioma, tj. pochodna wynosi 0.

Przykład 1.

Pochodne funkcji elementarnych

$$\begin{aligned}
 f(x) = \text{const} &\implies f'(x) = 0 & (x^n)' = nx^{n-1} && (\exp x)' = \exp x && (\ln y)' = \frac{1}{y}, y > 0 \\
 (\sin x)' &= \cos x & (\cos x)' &= -\sin x & (\operatorname{tg} x)' &= 1 + \operatorname{tg}^2 x & (\arctan x)' &= \frac{1}{1+x^2} \\
 (\operatorname{ctgx} x)' &= -1 - \operatorname{ctg}^2 x & (\arcsin y)' &= \frac{1}{\sqrt{1-y^2}}, -1 < y < 1
 \end{aligned}$$

■ Własności pochodnej

Jeśli $f, g : \mathbb{R} \supset A \rightarrow \mathbb{R}$ są różniczkowalne w $a \in A$, to

- $(f + g)'(a) = f'(a) + g'(a)$
- $(fg)'(a) = f'(a)g(a) + f(a)g'(a)$
- $\left(\frac{1}{g}\right)'(a) = -\frac{g'(a)}{g^2(a)}$ o ile $g(a) \neq 0$
- $(f \circ g)'(a) = f'(g(a)) \cdot g'(a)$
- Jeśli $g = f^{-1}$, g ciągła w punkcie $b = f(a)$, $f'(a) \neq 0$, to $g'(b) = \frac{1}{f'(a)}$

Różniczkowalność \implies ciągłość, ale nie na odwrót.

Uwaga: Może być tak, że funkcja jest ciągła na całej swojej dziedzinie, ale nie jest różniczkowalna w żadnym punkcie ze swojej dziedziny. Przykładem takiej funkcji jest **funkcja Weierstrassa**.

Przypis redakcji

W części teoretycznej brak informacji o funkcjach klasy C^n , a pojawia się to w zadaniach i dalszej teorii.

■ Reguła de l'Hospitala

Reguła de l'Hospitala ułatwia obliczanie granic wyrażeń nieoznaczonych typu $\frac{0}{0}$ i $\frac{\infty}{\infty}$.

Twierdzenie: Założymy, że $f, g : \mathbb{R} \supset P \rightarrow \mathbb{R}$ są różniczkowalne w punkcie skupienia $a \in P$ zbioru $P \subset \mathbb{R}$, a ponadto $f(a) = g(a) = 0 \neq g'(a)$. Wówczas granica

$$\lim_{x \rightarrow a} \frac{f(x)}{g(x)}$$

istnieje i jest równa $f'(a)/g'(a)$.

Przykład 2.

Policzymy granicę

$$\lim_{x \rightarrow 0} \frac{1 + 2x + x \sin x - x^2 \cos x - (\exp x)^2}{\sqrt{1+x^2} - \exp(x^2)}$$

Wstawiając 0 w miejsce x widzimy, że powyższa granica jest postaci $\frac{0}{0}$, więc używamy reguły de l'Hospitala:

$$\begin{aligned} \lim_{x \rightarrow 0} \frac{1 + 2x + x \sin x - x^2 \cos x - (\exp x)^2}{\sqrt{1+x^2} - \exp(x^2)} &= \lim_{x \rightarrow 0} \frac{0 + 2 + \sin x + x \cos x - 2x \cos x + x^2 \sin x - 2 \exp(2x)}{2x \frac{1}{2\sqrt{1+x^2}} - 2x \exp(x^2)} = \\ &= \lim_{x \rightarrow 0} \frac{2 + x^2 \sin x + \sin x - x \cos x - 2 \exp(2x)}{\frac{x}{\sqrt{1+x^2}} - 2x \exp(x^2)} = * \end{aligned}$$

Wstawiając 0 za x widzimy, że ponownie uzyskujemy granicę postaci $\frac{0}{0}$, więc znów korzystamy z reguły de l'Hospitala:

$$\begin{aligned} * &= \lim_{x \rightarrow 0} \frac{0 + 2x \sin x + x^2 \cos x + \cos x - \cos x + x \sin x - 2 \cdot 2 \exp(2x)}{\frac{1\sqrt{1+x^2}-x\frac{2x}{2\sqrt{1+x^2}}}{1+x^2} - 2 \exp(x^2) - 2x \cdot 2x \exp(x^2)} = \\ &= \lim_{x \rightarrow 0} \frac{3x \sin x + x^2 \cos x - 4 \exp(2x)}{\frac{1+x^2-x^2}{(1+x^2)\sqrt{1+x^2}} - 4x^2 \exp(x^2) - 2 \exp(x^2)} = \frac{3 \cdot 0 \sin 0 + 0 \cos 0 - 4 \exp(0)}{\frac{1}{(1+0)\sqrt{1+0}} - 4 \cdot 0 \exp(0) - 2 \exp(0)} = \frac{-4}{1-2} = 4 \end{aligned}$$

Reguła de l'Hospitala zawsze doprowadzi nas do wyniku, ale jak widać na przykładzie powyżej, czasami rachunki potrafią być długie i żmudne.

Zestaw zadań

1.11. Dana jest funkcja $f : (a, b) \rightarrow \mathbb{R}$ różniczkowalna na $(a, x_0) \cup (x_0, b)$, gdzie $x_0 \in (a, b)$. Wynika z tego, że

- A. f jest ciągła na (a, b)
- B. jeśli f jest ciągła na (a, b) , to jest też różniczkowalna na (a, b)
- C. f ma ograniczony zbiór wartości na każdym przedziale domkniętym zawartym w (a, b)

1.12. Funkcja $f : \mathbb{R} \rightarrow \mathbb{R}$ jest różniczkowalna oraz $f(0) = f(1) = 0$. Wynika z tego, że skończona jest granica

- A. $\lim_{n \rightarrow \infty} nf\left(\frac{1}{n}\right)$
- B. $\lim_{n \rightarrow \infty} nf\left(\frac{n+1}{n}\right)$
- C. $\lim_{n \rightarrow \infty} nf\left(\frac{n}{n+1}\right)$

1.13. Dla $a, b \in \mathbb{R}$ funkcja $f_{a,b} : \mathbb{R} \rightarrow \mathbb{R}$ jest zdefiniowana wzorem

$$f_{a,b}(x) = \begin{cases} e^x & \text{dla } x < 0 \\ ax + b & \text{dla } x \geq 0 \end{cases}$$

Wtedy

- A. istnieje dokładnie jedna para $a, b \in \mathbb{R}$, taka że $f_{a,b}$ jest funkcją dwukrotnie różniczkowalną

- B. istnieje nieskończenie wiele par $a, b \in \mathbb{R}$, takich że $f_{a,b}$ jest funkcją ciągłą
 C. istnieje dokładnie jedna para $a, b \in \mathbb{R}$, taka że $f_{a,b}$ jest funkcją różniczkowalną

1.14. Funkcja $f : (0; 1) \rightarrow \mathbb{R}$ zadana wzorem $f(x) = \sqrt{|\sin x|}$

- A. jest różniczkowalna
 B. jest ciągła i osiąga swój kres góryny
 C. ma pochodną ograniczoną

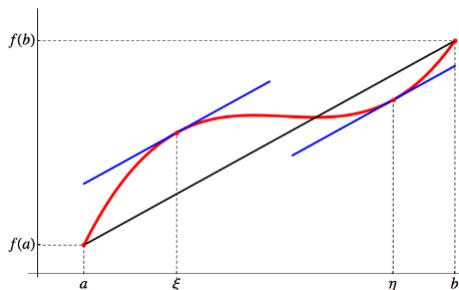
1.6.

Twierdzenie Lagrange'a o wartości średniej

Twierdzenie: Założmy, że $a < b$, zaś funkcja $f : [a, b] \rightarrow \mathbb{R}$ jest ciągła na $[a, b]$ i różniczkowalna w każdym punkcie $x \in (a, b)$. Wówczas istnieje taki punkt $\xi \in (a, b)$, że

$$f'(\xi) = \frac{f(b) - f(a)}{b - a}$$

Twierdzenie Lagrange'a ma bardzo prostą interpretację geometryczną: w przedziale (a, b) istnieje taki punkt, w którym styczna do wykresu f jest równoległa do siecznej, poprowadzonej przez dwa końce łuku wykresu.



Wniosek: Założymy, że $A \subset \mathbb{R}$, $[c, d] \subset A$, $c < d$, a funkcja $f : A \rightarrow \mathbb{R}$ jest ciągła na $[c, d]$ i różniczkowalna w (c, d) . Zachodzą wówczas następujące implikacje:

- Jeśli $f'(x) \geq 0$ dla każdego $x \in (c, d)$, to f jest niemalejąca na $[c, d]$
- Jeśli $f'(x) > 0$ dla każdego $x \in (c, d)$, to f jest rosnąca na $[c, d]$
- Jeśli $f'(x) < 0$ dla każdego $x \in (c, d)$, to f jest malejąca na $[c, d]$
- Jeśli $f'(x) \leq 0$ dla każdego $x \in (c, d)$, to f jest nierosnąca na $[c, d]$

To było na egzaminie

Funkcja $f : [0; 1] \rightarrow \mathbb{R}$ jest różniczkowalna oraz $f(0) = 1$, $f(1) = 0$. Wynika z tego, że istnieje $x \in [0; 1]$, dla którego $f'(x)$ jest

- A. mniejsze od 0
 B. równe -1
 C. większe bądź równe 1

- A. Ponieważ funkcja zmalała między 0 a 1, to na jakimś odcinku na przedziale $[0, 1]$ pochodna była ujemna.
- B. Z twierdzenia Lagrange'a o wartości średniej dostajemy, że istnieje takie ξ , że $f'(\xi) = \frac{f(b)-f(a)}{b-a} = \frac{0-1}{1-0} = -1$, gdzie $a = 0, b = 1$.
- C. Nie możemy nic na ten temat założyć. Funkcja $f(x) = 1 - x$ jest kontrargumentem.

■ Wypukłość

Założymy, że $f : P = (a, b) \rightarrow \mathbb{R}$ jest dwukrotnie różniczkowalna. Wówczas:

- Jeśli $f'' \geq 0$ na P , to f jest wypukła na P
- Jeśli $f'' > 0$ na P , to f jest ściśle wypukła na P
- Jeśli $f'' < 0$ na P , to f jest ściśle wklęsła na P
- Jeśli $f'' \leq 0$ na P , to f jest wklęsła na P

1.7. Wzór Taylora

Oznaczmy $f^{(n)}$ jako n -tą pochodną funkcji f .

Wzór Taylora z resztą Peano: Niech $P \subset \mathbb{R}$ będzie przedziałem otwartym, $x_0 \in P$. Założymy, że $f : P \rightarrow \mathbb{R}$ ma $(k-1)$ pochodnych na P i k -tą pochodną w punkcie x_0 . Wówczas

$$f(x) = \sum_{j=0}^k \frac{f^{(j)}(x_0)}{j!} (x - x_0)^j + r(x), \quad \lim_{x \rightarrow x_0} \frac{r(x)}{(x - x_0)^k} = 0$$

Wzór Taylora z resztą Lagrange'a: Założymy, że funkcja $f : (a, b) \rightarrow \mathbb{R}$ ma w przedziale (a, b) pochodne do rzędu $(k+1)$ włącznie. Wówczas, dla każdego $x_0 \in (a, b)$ i każdego $x \in (a, b)$ istnieje taki punkt c , pośredni między x_0 i x , że

$$f(x) = \sum_{j=0}^k \frac{f^{(j)}(x_0)}{j!} (x - x_0)^j + \frac{f^{(k+1)}(c)}{(k+1)!} (x - x_0)^{k+1}$$

Przykład 1.

Rozwiniecie znanych funkcji w szereg Taylora:

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{1}{1!}x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \dots$$

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

$$\ln(1+x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots$$

Szeregi potęgowe

Definicja \limsup :

$$\limsup_{n \rightarrow \infty} a_n = \inf_{n \in \mathbb{N}} \left(\sup_{k \geq n} a_k \right) = \lim_{n \rightarrow \infty} s_n, \quad \text{gdzie } s_n = \sup_{k \geq n} a_k$$

Równoważne definicje:

- Kres górnny zbioru granic wszystkich podciągów (a_n)
- Największa spośród granic podciągów (a_n)

Szereg potęgowy o współczynnikach $a_n \in \mathbb{C}$ i środku $z_0 \in \mathbb{C}$:

$$S(z) = \sum_{n=0}^{\infty} a_n (z - z_0)^n$$

Twierdzenie: Szereg jw. jest zbieżny (bezwzględnie) w kole $|z - z_0| < R$ i rozbieżny dla $|z - z_0| > R$, gdzie

$$\frac{1}{R} = \limsup_{n \rightarrow \infty} \sqrt[n]{|a_n|}$$

Uwaga: na brzegu koła zbieżności, dla $|z - z_0| = R$, może być różnie ...

Przykład 2.

Pokażemy, że szereg $\sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} x^{2k+1}$ ma promień zbieżności $R = \infty$. Zauważmy, że dla n parzystych mamy $a_n = 0$. Dla n nieparzystych dostajemy podciąg ciągu $(n!)^{-1/n}$, który zbiega do 0:

$$\limsup_{n \rightarrow \infty} |a_n|^{1/n} = \limsup_{n \rightarrow \infty} \left| \frac{1}{(n!)^{1/n}} \right| = \lim_{n \rightarrow \infty} \frac{1}{(n!)^{1/n}} = 0,$$

a więc $\frac{1}{R} = 0 \Rightarrow R = \infty$.

Różniczkowalność sumy szeregu potęgowego

Twierdzenie: Założymy, że $R > 0$ jest promieniem zbieżności szeregu potęgowego $S(z) = \sum_{n=0}^{\infty} a_n z^n$. Wtedy funkcja S ma pochodną w każdym punkcie $z \in \{w \in \mathbb{C} : |w| < R\}$ i zachodzi wzór

$$S'(z) = \sum_{n=1}^{\infty} n a_n z^{n-1}$$

Wniosek:

- Suma szeregu potęgowego jest ciągła wewnątrz koła zbieżności
- Suma szeregu potęgowego jest C^∞ wewnątrz koła zbieżności
- Dla $k \in \mathbb{N}$ jest

$$S^{(k)}(z) = \sum_{n=k}^{\infty} n(n-1)\dots(n-k+1) a_n z^{n-k}, \quad S^{(k)}(0) = k! \cdot a_k$$

(każdy szereg potęgowy sam jest swoim szeregiem Taylora)

1.15. Dany jest szereg $S(x) = \sum_{n=2}^{\infty} a_n x^n$ zbieżny dla $x \in (-1, 1)$ oraz rozbieżny w $x = -1$. Wynika z tego, że

- A. $S(x)$ jest rozbieżny w $x = 1$
- B. $S(x)$ jest rozbieżny w $x = -2$
- C. $S(x)$ jest różniczkowalny na odcinku $(-1, 1)$ oraz $S'(0) = 0$

1.16. Niech $f : \mathbb{R} \rightarrow \mathbb{R}$ będzie dana wzorem $f(x) = |-3x + x^2 - 5|$. Prawdą jest, że

- A. f jest różniczkowalna
- B. f jest dwukrotnie różniczkowalna
- C. jej pierwszy szereg Taylora w otoczeniu $x_0 = 0$ i $x = 1$ wynosi $\sqrt{2}$

1.17. Niech $f : [0, 1] \rightarrow \mathbb{R}$ będzie ciągła. Wtedy

- A. $f'(x)$ istnieje dla każdego $x \in ([0, 1] \setminus A)$ gdzie A jest pewnym skończonym podzbiorem \mathbb{R}
- B. zbiór wartości f jest punktem lub przedziałem domkniętym
- C. f jest sumą pewnego szeregu potęgowego

1.18. Niech $H_n := \sum_{j=1}^n \frac{1}{j}$ będzie n -tą liczbą harmoniczną. Wówczas

- A. promień zbieżności szeregu potęgowego $\sum_{n=1}^{\infty} H_n x^n$ wynosi 1
- B. $\lim_{n \rightarrow \infty} \frac{H_n}{\ln n} = 1$
- C. $\lim_{n \rightarrow \infty} (H_n - \ln n) = 0$

1.8.

Zbieżność ciągów liczbowych i funkcyjnych

Powiemy, że ciąg funkcji (f_n) jest **zbieżny punktowo** do f na zbiorze X wtedy i tylko wtedy, gdy dla każdego punktu $x \in X$ zachodzi równość

$$f(x) = \lim_{n \rightarrow \infty} f_n(x)$$

Piszemy wtedy: $f_n \rightarrow f$ na X .

Powiemy, że ciąg funkcji (f_n) jest **zbieżny jednostajnie** do f na zbiorze X wtedy i tylko wtedy, gdy zachodzi warunek: dla każdego $\varepsilon > 0$ istnieje $n_0 = n_0(\varepsilon) \in \mathbb{N}$ takie, że dla wszystkich $x \in X$ i wszystkich $n > n_0$ jest

$$|f(x) - f_n(x)| < \varepsilon$$

Piszemy wtedy: $f_n \rightrightarrows f$ na X .

Twierdzenie: Założmy, że $P \subset \mathbb{R}$ jest dowolnym przedziałem. Niech $f_n : P \rightarrow \mathbb{R}$ będą funkcjami ciągłymi na P . Jeśli $f_n \rightrightarrows f$ na P , to f jest ciągła na P .

Mówimy, że szereg funkcji $\sum_{k=1}^{\infty} f_k(x)$ jest zbieżny punktowo (jednostajnie) do funkcji $f(x)$ na zbiorze X wtedy i tylko wtedy, gdy ciąg sum częściowych $S_n = \sum_{k=1}^n f_k$ tego szeregu jest zbieżny do f punktowo (jednostajnie) na zbiorze X .

Przykład 1.

Niech $f_n : [0, 1] \rightarrow \mathbb{R}$ będzie dana wzorem $f_n(x) = x^n$. Wtedy

$$\lim_{n \rightarrow \infty} f_n(x) = \lim_{n \rightarrow \infty} x^n = f(x) := \begin{cases} 0, & x \in [0, 1) \\ 1, & x = 1 \end{cases}$$

Innymi słowy, ciąg f_n jest zbieżny **punktowo** na $[0, 1]$ do funkcji f . NIE jest jednak zbieżny **jednostajnie**: dla każdego $n \in \mathbb{N}$ jest

$$|f_n(2^{-1/n}) - f(2^{-1/n})| = \frac{1}{2} - 0 = \frac{1}{2},$$

a zatem warunek z definicji zbieżności jednostajnej nie zachodzi dla żadnej liczby $\varepsilon < \frac{1}{2}$.

Kryteria zbieżności jednostajnej

Twierdzenie: Niech $f_n : X \rightarrow \mathbb{R}$ dla $n = 1, 2, \dots$. Następujące warunki są równoważne:

- Ciąg (f_n) jest zbieżny jednostajnie na X do pewnej funkcji $f : X \rightarrow \mathbb{R}$
- Ciąg (f_n) spełnia **jednostajny warunek Cauchy'ego**: dla każdego $\varepsilon > 0$ istnieje $n_0 \in \mathbb{N}$ takie, że dla wszystkich $m, n > n_0$ i wszystkich $x \in X$ zachodzi nierówność

$$|f_n(x) - f_m(x)| < \varepsilon$$

Twierdzenie (Kryterium Weierstrassa): Niech $f_n : X \rightarrow \mathbb{R}$ dla $n = 1, 2, \dots$. Jeśli $|f_n(x)| \leq a_n$ dla $n \in \mathbb{N}$, a szereg liczbowy $\sum_{n=1}^{\infty} a_n$ jest zbieżny, to wówczas szeregi funkcyjne $\sum_{n=1}^{\infty} f_n(x)$ oraz $\sum_{n=1}^{\infty} |f_n(x)|$ są zbieżne jednostajnie na X . W takiej sytuacji mówimy, że szereg $\sum f_n(x)$ jest zbieżny jednostajnie i bezwzględnie.

Różniczkowanie ciągów funkcyjnych

Twierdzenie: Niech $f_n : \mathbb{R} \supset [a, b] \rightarrow \mathbb{R}$ będą różniczkowalne. Jeśli ciąg $f'_n \rightrightarrows g$ na $[a, b]$, a ponadto istnieje taki punkt $x_0 \in [a, b]$, że ciąg $(f_n(x_0))$ jest zbieżny, to wówczas

- $f_n \rightrightarrows f$ na $[a, b]$, gdzie f ciągła
- Funkcja f jest różniczkowalna na $[a, b]$ i $f' = g$

Zestaw zadań

1.19. Funkcja $f : \mathbb{R} \rightarrow \mathbb{R}$ jest zadana wzorem $f(x) = \sum_{n=1}^{\infty} \frac{\sin(nx)}{n^3}$. Wynika z tego, że

- A. funkcja f jest ograniczona na \mathbb{R}
- B. funkcja f jest różniczkowalna na \mathbb{R}
- C. spełniona jest nierówność $16 > f'(0) > \frac{1}{16}$

1.20. Ciąg funkcji $f_n(x) = x^n$ jest

- A. zbieżny punktowo na $[0, 1]$
- B. zbieżny jednostajnie na $[0, 1]$
- C. ograniczony na $[0, 1 + \varepsilon]$ dla pewnego $\varepsilon > 0$

1.21. Ciąg funkcyjny $\{f_n\}_{n \geq 1}$ składa się z funkcji $f_n : \mathbb{R} \rightarrow \mathbb{R}$ zdefiniowanych wzorami $f_n(x) = x - \frac{1}{n}$ dla $x \in \mathbb{R}$. Ten ciąg funkcyjny jest

- A. zbieżny punktowo i jednostajnie
 B. niemal jednostajnie zbieżny
 C. zbieżny punktowo do funkcji ciągły

1.9.

Ekstrema funkcji

Niech $f : I \rightarrow \mathbb{R}$, gdzie I jest przedziałem w \mathbb{R} . Mówimy, że funkcja f ma w punkcie $a \in I$ maksimum lokalne (odpowiednio: minimum lokalne), jeśli istnieje taka liczba $\delta > 0$, że dla wszystkich $x \in I$, $|x-a| < \delta$, zachodzi nierówność $f(a) \geq f(x)$ (odpowiednio: $f(a) \leq f(x)$).

Słowo **ekstremum** jest wspólną nazwą minimum i maksimum. Należy pamiętać, że wartość funkcji w punkcie ekstremum lokalnego nie musi być najmniejszą ani największą wartością funkcji na danym przedziale. Ekstremów może być wiele, a ponadto jeśli przedział jest otwarty, to funkcja w ogóle nie musi przyjmować swoich kresów.

■ Warunki istnienia ekstremów

Lemat (Fermata, warunek konieczny): Jeśli I jest odcinkiem otwartym, $a \in I$, $f : I \rightarrow \mathbb{R}$ jest różniczkowalna w punkcie a i f ma w a ekstremum lokalne, to $f'(a) = 0$.

Interpretacja geometryczna: styczna do wykresu funkcji różniczkowalnej w punkcie ekstremum lokalnego jest pozioma.

Przykład 1.

Należy pamiętać, że warunek ten nie jest wystarczający, a najlepszym na to przykładem jest funkcja $f(x) = x^3$. Jej pochodna $f'(x) = 3x^2$ zeruje się w $x_0 = 0$, ale oczywiście f nie ma w tym punkcie ekstremum, co widać na wykresie.

Twierdzenie (Rolle'a): Założmy, że $a < b$, zaś funkcja $f : [a, b] \rightarrow \mathbb{R}$ jest ciągła na $[a, b]$ i różniczkowalna w każdym punkcie $x \in (a, b)$. Jeśli $f(a) = f(b)$, to istnieje taki punkt $x_0 \in (a, b)$, że $f'(x_0) = 0$.

Warunek dostateczny istnienia ekstremum: Funkcja ciągła $f : [a, b] \rightarrow \mathbb{R}$, różniczkowalna w (a, b) ma w punkcie $x_0 \in (a, b)$:

- maksimum lokalne wtedy i tylko wtedy, gdy $f'(x_0) = 0$ oraz istnieje $\delta > 0$ takie, że $f'(x) > 0$ dla $x \in (x_0 - \delta, x_0)$ i $f'(x) < 0$ dla $x \in (x_0, x_0 + \delta)$;
- minimum lokalne wtedy i tylko wtedy, gdy $f'(x_0) = 0$ oraz istnieje $\delta > 0$ takie, że $f'(x) < 0$ dla $x \in (x_0 - \delta, x_0)$ i $f'(x) > 0$ dla $x \in (x_0, x_0 + \delta)$.

To było na egzaminie

Liczba rozwiązań równania $e^x = 2x$ jest równa

- A. 0
 B. 1
 C. 2

Oznaczmy $f(x) = e^x - 2x$. Wtedy $f'(x) = e^x - 2$. Policzmy ekstremum funkcji f . Warunek konieczny: $f'(x_0) = 0 \Rightarrow x_0 = \ln 2$. Zauważmy, że dla $x < x_0$ jest $f'(x) < 0$ oraz dla $x > x_0$ jest $f'(x) > 0$. Z warunku dostatecznego otrzymujemy, że x_0 jest minimum lokalnym funkcji f . Skoro $f(x_0) = e^{\ln 2} - 2 \ln 2 = 2 - 2 \ln 2 > 2 - 2 \ln e = 0$,

to znaczy, że f nie ma miejsc zerowych, bo jest dodatnia w całej swojej dziedzinie. Tak więc, liczba rozwiązań równania $e^x = 2x$ wynosi 0.

Zestaw zadań

1.22. Dana jest różniczkowalna funkcja $f : \mathbb{R} \rightarrow \mathbb{R}$, taka że $f'(0) = 0$. Wynika z tego, że

- A. f ma ekstremum lokalne w $x = 0$
- B. jeśli dla każdego $x \neq 0$ zachodzi $f'(x) < 0$, to f jest malejąca
- C. f jest ciągła w $x = 2019$

1.10.

Całki

F jest funkcją pierwotną f na P , gdy $F' = f$ na P .

Całka nieoznaczona

Niech $f : P \rightarrow \mathbb{R}$, gdzie $P \subset \mathbb{R}$ jest dowolnym przedziałem. Całką nieoznaczoną funkcji f nazywamy rodzinę wszystkich funkcji pierwotnych funkcji f . Używa się zwykle zapisu

$$\int f(x) dx = F(x) + \text{const},$$

gdzie F jest dowolnie wybraną funkcją pierwotną f na danym przedziale.

Przykład 1.

Najważniejsze wzory na całki nieoznaczone:

$$\begin{aligned}\int x^n dx &= \frac{1}{n+1} x^{n+1} + C, \quad n \neq -1 & \int \frac{1}{x} dx &= \ln|x| + C & \int a^x dx &= \frac{a^x}{\ln a} + C \\ \int e^x dx &= e^x + C & \int \sin x dx &= -\cos x + C & \int \cos x dx &= \sin x + C \\ \int \operatorname{tg} x dx &= -\ln|\cos x| + C & \int \operatorname{ctg} x dx &= \ln|\sin x| + C & \int \frac{1}{\cos^2 x} dx &= \operatorname{tg} x + C \\ \int \frac{1}{\sin^2 x} dx &= -\operatorname{ctg} x + C & \int \frac{1}{1+x^2} dx &= \operatorname{arctg} x + C & \int \frac{1}{\sqrt{1-x^2}} dx &= \arcsin x + C\end{aligned}$$

Własności całek nieoznaczonych

Liniowość całki: Jeśli $f, g : P \rightarrow \mathbb{R}$ są ciągłe, zaś $a, b \in \mathbb{R}$, to

$$\int (af(x) + bg(x)) dx = a \int f(x) dx + b \int g(x) dx$$

Dowód: przez wzór na pochodną sumy.

Całkowanie przez części: Jeśli $f, g : P \rightarrow \mathbb{R}$ są różniczkowalne, to

$$\int f(x) \cdot g'(x) dx = f(x)g(x) - \int f'(x) \cdot g(x) dx$$

Dowód: przez wzór na pochodną iloczynu.

Przykład 2.

Policzymy całkę $\int \ln x \, dx$. Położmy $f(x) = \ln x$ i $g'(x) = 1$. Wtedy $f'(x) = \frac{1}{x}$ oraz $g(x) = x$. Stosując powyższy wzór dla f i g zdefiniowanych przed chwilą, dostajemy:

$$\int \ln x \, dx = x \ln x - \int \frac{1}{x} \cdot x \, dx = x \ln x - \int dx = x \ln x - x + C$$

Całkowanie przez podstawienie: Niech f, g' będą ciągłe i niech F będzie funkcją pierwotną f . Wtedy

$$\int f(g(x))g'(x) \, dx = F(g(x)) + C$$

Dowód: przez wzór na pochodną złożenia.

Przykład 3.

Policzymy całkę $\int \sin^5 x \cos x \, dx$. Podstawimy $t = \sin x$. Wtedy $dt = \cos x \, dx$. Dostajemy

$$\int \sin^5 x \cos x \, dx = \int t^5 \, dt = \frac{t^6}{6} + C = \frac{\sin^6 x}{6} + C$$

Intuicja: szukamy takiego podstawienia, że mając $t = p(x)$, gdzieś pod całką jest czynnik $p'(x) \cdot dx$, co fajnie działa np. w przypadku iloczynu sinusa z cosinusem, jak widać powyżej.

■ Całka oznaczona

Niech $f : [a, b] \rightarrow \mathbb{R}$ będzie funkcją ciągłą. Całką oznaczoną funkcji f na przedziale $[a, b]$ nazywamy liczbę

$$\int_a^b f(x) \, dx = F(b) - F(a),$$

gdzie F jest dowolną funkcją pierwotną funkcji f .

Wszystkie własności całki nieoznaczonej mają tu zastosowanie. Generalnie, jeśli chcemy policzyć całkę oznaczoną, to liczymy całkę nieoznaczoną, a następnie podstawiamy krańce przedziałów za x i odejmujemy od siebie.

■ Interpretacja geometryczna

Całkę możemy interpretować jako pole pod wykresem funkcji. Suma $\sum_{i=1}^n f(t_i)(x_i - x_{i-1})$ to suma pól prostokątów. Dla dużych n , gdy podstawy wszystkich prostokątów są małe, wartość tej sumy jest dobrym przybliżeniem całki $\int_a^b f(x) \, dx$, tzn. pola zacienionego obszaru.

■ Całka niewłaściwa

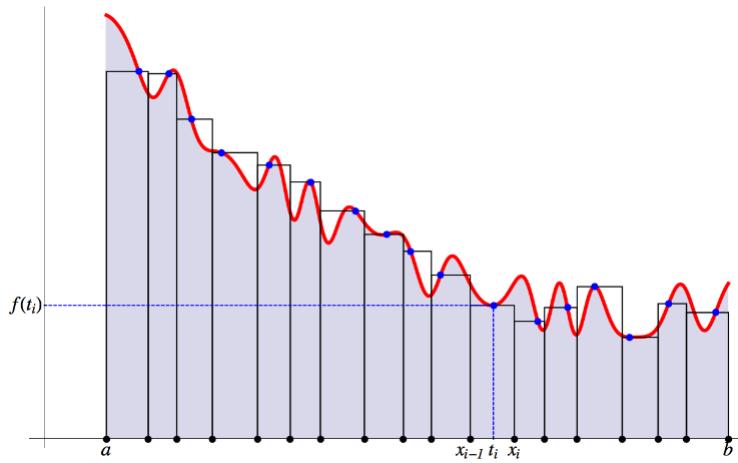
Załóżmy, że funkcja $f : [a, \infty) \rightarrow \mathbb{R}$ jest ciągła. Jeśli istnieje skończona granica

$$\lim_{y \rightarrow \infty} \int_a^y f(x) \, dx,$$

to nazywamy ją całką niewłaściwą funkcji f na przedziale $[a, \infty)$ i oznaczamy

$$\int_a^\infty f(x) \, dx$$

Mówimy wtedy, że całka niewłaściwa f na $[a, \infty)$ jest zbieżna. Jeśli granica nie istnieje, to mówimy, że całka $\int_a^\infty f(x) \, dx$ jest rozbieżna. Analogicznie definiujemy całkę niewłaściwą funkcji $f : (-\infty, a] \rightarrow \mathbb{R}$.



Przykład 4.

Policzymy całkę $\int_0^\infty e^{-x} dx$. Mamy

$$\int_0^\infty e^{-x} dx = \lim_{y \rightarrow \infty} \int_0^y e^{-x} dx = \lim_{y \rightarrow \infty} -e^{-x} \Big|_0^y = \lim_{y \rightarrow \infty} (-e^{-y} + e^0) = \lim_{y \rightarrow \infty} (1 - e^{-y}) = 1,$$

tak więc całka $\int_0^\infty e^{-x} dx$ jest zbieżna do 1.

Zestaw zadań

1.23. Niech $\{f_n\}_{n \geq 0}$ będzie ciągiem ograniczonych funkcji typu $(0, 1) \rightarrow \mathbb{R}$, takich że $\int_0^1 f_n(t) dt = 0$. Wtedy

- A. istnieje prawostronna granica f_0 w 0^+
- B. $\{f_n(x)\}_{n \geq 0}$ jest zbieżny do 0 dla każdego $x \in (0, 1)$
- C. jeśli $\{f_n(x)\}_{n \geq 0}$ jest zbieżny do 0 dla każdego $x \in (0, 1)$, to $\left\{ \int_0^1 |f_n(t)| dt \right\}_{n \geq 0}$ jest zbieżny do 0

1.24. Niech $f : (-1, 1) \rightarrow \mathbb{R}$ będzie zdefiniowana wzorem

$$f(x) = \frac{1}{x^2 - 4x + 3}$$

Wtedy

- A. funkcja f jest ograniczona na $(-1, 1)$
- B. pochodna $f^{(1000)}(0)$ wynosi $1000! \left(\frac{1}{2} - \frac{1}{6 \cdot 3^{1000}} \right)$
- C. zachodzi $\int_{-1}^0 f(t) dt = \frac{\ln 3 - \ln 2}{2}$

1.11.

Elementy teorii miary

Przypis redakcji

Rozdział nie jest jeszcze stworzony – w historii przeanalizowanych na potrzeby tego repetytorium egzaminów pojawiło się tylko jedno zadanie z teorii miary, w dodatku w miarę proste.

Zestaw zadań

- 1.25. W pewnej metryce ρ zachodzi $\rho(x_1, x_2) = 7$. Dane są dwie otwarte kule, K_1 o środku x_1 i promieniu 1 oraz K_2 o środku w x_2 i promieniu 2. Wtedy

- A. $(K_2 \cup K_1)$ jest zbiorem otwartym
- B. istnieje kula otwarta o promieniu 8 zawierająca kulę K_1 i K_2
- C. zbiór $(K_1 \cap K_2)$ jest niepusty

1.12.

Rozwiązańia

Rozwiązańia

- 1.1. Jeśli ciąg liczb rzeczywistych dodatnich a_n jest monotoniczny i ograniczony, to

- NIE A. ciąg $\sin(a_n)$ jest monotoniczny
- TAK B. ciąg $\cos(a_n)$ jest zbieżny
- NIE C. ciąg $\log(a_n)$ jest ograniczony

Ciąg a_n jest zbieżny. **A.** Nie musi tak wcale być, np dla ciągu $a_1 = \frac{\pi}{2}$, $a_2 = -\frac{3\pi}{2}$, $a_n = \frac{1}{n}$ dla $n > 2$. **B.** jest zbieżny, bo $\lim_{n \rightarrow \infty} \cos(a_n) = \cos(\lim_{n \rightarrow \infty} a_n)$. **C.** dla $a_n = \frac{1}{n}$ mamy granicę równą 0 a $\lim_{n \rightarrow 0} \log(n) = -\infty$.

- 1.2. Ciąg, którego n -ty wyraz zadany jest wzorem $\frac{2010^n - n^{2010}}{2^{n^2} - 2010^n}$,

- TAK A. jest zbieżny
- NIE B. ma granicę równą -1
- TAK C. ma granicę równą 0

$$\lim_{n \rightarrow \infty} \frac{2010^n - n^{2010}}{2^{n^2} - 2010^n} = \lim_{n \rightarrow \infty} \frac{1 - \frac{n^{2010}}{2010^n}}{\frac{2^{n^2}}{2010^n} - 1} = \lim_{n \rightarrow \infty} \frac{1 - \frac{n^{2010}}{2010^n}}{\left(\frac{2^n}{2010}\right)^n - 1}$$

Funkcja wykładnicza rośnie szybciej niż wielomianowa, więc składnik $\frac{n^{2010}}{2010^n} \rightarrow 0$, a oczywiście $\frac{2^n}{2010} \rightarrow \infty$, więc $\lim_{n \rightarrow \infty} \frac{2010^n - n^{2010}}{2^{n^2} - 2010^n} = \frac{1-0}{\infty-1} = 0$. Ciąg jest zbieżny do 0, więc odpowiedzi **A** i **C** są prawdziwe.

- 1.3. Niech $\{a_n\}_{n \geq 0}$ będzie ograniczonym ciągiem dodatnich liczb rzeczywistych. Wtedy

- NIE A. jeśli $\{\sin(a_n)\}_{n \geq 0}$ jest zbieżny, to a_n jest monotoniczny
- TAK B. $\{\exp(-a_n)\}_{n \geq 0}$ posiada podciąg zbieżny do granicy dodatniej

NIE C. jeśli $\{\exp(-a_n)\}_{n \geq 0}$ jest zbieżny, to $\{\arctan(a_n)\}_{n \geq 0}$ ma granicę dodatnią

A. podobnie jak w 1.1.A, nie musi tak być. B. a_n jest ograniczony, więc posiada podciąg zbieżny. W takim razie ciąg $1/e^{a_n}$ ma podciąg zbieżny do granicy dodatniej. Wystarczy wziąć ten podciąg zbieżny z a_n i wziąć z niego funkcję \exp . C. Niech $a_n = \frac{1}{n}$. Wtedy $\exp(-a_n)$ jest zbieżny, a_n ograniczony, ale $\arctan(a_n)$ zbiega do $\arctan(0) = 0$.

1.4. Ciąg określony dla $n \geq 1$ wzorem $(1 + \frac{1}{4^n})^{2^n}$ jest

TAK A. zbieżny do 1

NIE B. rosnący, począwszy od pewnego miejsca

TAK C. ograniczony

$$\lim_{n \rightarrow \infty} \left(1 + \frac{1}{4^n}\right)^{2^n} = \lim_{n \rightarrow \infty} \left(\left(1 + \frac{1}{4^n}\right)^{4^n}\right)^{\frac{1}{2^n}} = e^{\lim_{n \rightarrow \infty} \frac{1}{2^n}} = e^0 = 1$$

Ciąg zbiega do 1, więc jest ograniczony. Odpowiedzi A i C są więc prawdziwe. Sprawdźmy, czy ciąg rośnie od pewnego miejsca. W tym celu sprawdzamy, czy iloraz $\frac{a_{n+1}}{a_n} > 1$. Mamy

$$\frac{a_{n+1}}{a_n} = \frac{(1 + \frac{1}{4^{n+1}})^{2^{n+1}}}{(1 + \frac{1}{4^n})^{2^n}} = \left(\frac{(1 + \frac{1}{4^{n+1}})^2}{1 + \frac{1}{4^n}}\right)^{2^n} = \left(\frac{1 + \frac{2}{4^{n+1}} + \frac{1}{4^{2n+2}}}{1 + \frac{1}{4^n}}\right)^{2^n}$$

Musi być $1 + \frac{2}{4^{n+1}} + \frac{1}{4^{2n+2}} > 1 + \frac{1}{4^n}$, czyli $\frac{2}{4} + \frac{1}{4^{2n+2}} > 1$, co jest równoważne $4^{n+2} < 2$. To jest oczywiście nieprawda dla dużych n , więc odpowiedź w B to NIE.

1.5. Ciąg $a_n = (0.999 + \frac{1}{4n})^{4n}$

NIE A. jest zbieżny do liczby większej od 0

NIE B. jest zbieżny do liczby niewymiernej

TAK C. jest ograniczony z góry

Możemy ograniczyć ciąg z dołu przez 0, bo wszystkie jego wyrazy są dodatnie. Zauważmy teraz, że dla bardzo dużych n możemy ograniczyć go z góry $(0.999 + \frac{1}{4n})^{4n} < (0.999 + 0.0001)^{4n} = (0.9991)^{4n} \rightarrow 0$, bo podnosimy liczbę mniejszą od 1 do bardzo dużej potęgi. Z twierdzenia o trzech ciągach otrzymujemy $a_n \rightarrow 0$, więc odpowiedzi A i B są fałszywe. Ograniczenie z góry możemy łatwo pokazać, np. $(0.999 + \frac{1}{4n})^{4n} < (1 + \frac{1}{4n})^{4n} \rightarrow e$, więc odpowiedź C jest prawdziwa.

1.6. $\{S_n\}_{n \geq 1}$ jest ciągiem sum częściowych szeregu rozbieżnego. Wynika z tego, że ciąg $\{S_{n+1} - S_n\}_{n \geq 1}$ jest

NIE A. rozbieżny

NIE B. nieograniczony

NIE C. niezbieżny do 0

Trzeba się dokładnie wczytać co jest ciągiem a co szeregiem. S_n jest ciągiem, tak samo $S_{n+1} - S_n$. Powiedzmy, że szeregiem w zadaniu jest $\sum \frac{1}{n}$. Wtedy $S_n = 1 + \frac{1}{2} + \dots + \frac{1}{n}$, a $(S_{n+1} - S_n) = S_n + \frac{1}{n+1} - S_n = \frac{1}{n+1}$. Ten ciąg jest zbieżny do 0, także ograniczony i nie rozbieżny.

1.7. Rozważmy szereg:

$$(A) \sum_{n=1}^{\infty} \frac{7^n + n^9 + \ln(19^{n^2} + 8)}{8^n - 11^{\ln(1+n)} + 2} \quad \text{oraz} \quad (B) \sum_{n=1}^{\infty} \frac{8^n - 11^{\ln(1+n)} + 2}{7^n + n^9 + \ln(19^{n^2} + 8)}$$

NIE A. ciąg sum częściowych każdego z tych szeregów jest ograniczony

TAK B. szereg (A) jest zbieżny

NIE C. szereg (B) jest zbieżny

Bystry czytelnik zauważa, że wyrazy 7^n i 8^n dominują w obydwu szeregach, więc przy liczeniu warunku koniecznego dostajemy: $\lim_{n \rightarrow \infty} \frac{7^n}{8^n} = 0$ w szeregu (A) oraz $\lim_{n \rightarrow \infty} \frac{8^n}{7^n} = \infty$ w szeregu (B). Wynika z tego, że tylko szereg (A) jest zbieżny, czyli ma ograniczony ciąg sum częściowych.

1.8. Funkcja $f: \mathbb{R} \rightarrow \mathbb{R}$ dana wzorem $f(x) = e^{-x^2} \cdot \sqrt[3]{x} \cdot \sin(e^{x^2})$

NIE A. jest różniczkowalna na \mathbb{R}

TAK B. jest jednostajnie ciągła na \mathbb{R}

NIE C. spełnia warunek Lipschitza na \mathbb{R}

A. Wywala się w 0. Aby to zobaczyć, policzmy pochodną w 0 z definicji:

$$\lim_{x \rightarrow 0} \frac{f(x) - f(0)}{x - 0} = \lim_{x \rightarrow 0} \frac{e^{-x^2} \cdot \sqrt[3]{x} \cdot \sin(e^{x^2})}{x} = \lim_{x \rightarrow 0} \frac{e^{-x^2} \cdot \sin(e^{x^2})}{\sqrt[3]{x^2}} = \frac{e^0 \cdot \sin(e^0)}{0} = \infty,$$

czyli granica nie istnieje, a więc funkcja nie jest różniczkowalna w \mathbb{R} , bo nie jest różniczkowalna w 0.

B. Zbadajmy jednostajną ciągłość $g(x) = e^{-x^2}$. Z twierdzenia Lagrange'a o wartości średniej (o którym później) mamy, że jeśli $x < y$, to istnieje takie $c \in (x, y)$, że $g(y) - g(x) = (y - x)g'(c) = (y - x)(-2ce^{-c^2})$. Znając nierówność $e^x \geq x + 1$ oraz $|x|^2 + 1 \geq 2|x|$ nakładamy moduł na obydwie strony równości dostając:

$$|g(y) - g(x)| = |y - x| \frac{2|c|}{e^{c^2}} \leq |y - x| \frac{2|c|}{c^2 + 1} \leq |y - x| \frac{2|c|}{2|c|} = |y - x|.$$

Dostaliśmy oszacowanie $|g(x) - g(y)| \leq |x - y|$ dla dowolnych x, y . Znaczy to, że g spełnia warunek Lipschitza ze stałą $L = 1$, więc jest jednostajnie ciągła. Zadanie dla Czytelnika: pokazać, że przemnożenie g przez $\sqrt[3]{x} \cdot \sin(e^{-x^2})$ nie popsuje jednostajnej ciągłości.

C. Nie spełnia, bo jeśli położymy $y = 0$ w nierówności z warunku Lipschitza, to dostaniemy $\left| \frac{f(x)}{x} \right| \leq L$. Z podpunktu A. wiemy, że nie da się tego ograniczyć dla małych x .

1.9. Dla $n > 0$ określamy

$$a_n = \begin{cases} 1 - \frac{1}{n} & \text{dla } n \text{ nieparzystych} \\ 2 + \frac{1}{n^2} & \text{dla } n \text{ parzystych} \end{cases}$$

Oznaczmy $A = \{a_n \mid n > 0\}$ oraz $-A = \{-a_n \mid n > 0\}$. Wtedy

NIE A. A zawiera dokładnie jeden ze swoich kresów

TAK B. A ma dokładnie dwa punkty skupienia

TAK C. $\sup(-A) = \inf(A)$

A. Można zobaczyć, że $A = \{0, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \dots\} \cup \{2\frac{1}{4}, 2\frac{1}{9}, \dots\}$ skąd widać, że A zawiera 2 swoje kresy: $\{0, 3\} \subset A$

B. Punktami skupienia A są 1 i 2

C. Można zobaczyć, że $-A = \{0, -\frac{1}{2}, -\frac{2}{3}, -\frac{3}{4}, \dots\} \cup \{-2\frac{1}{4}, -2\frac{1}{9}, \dots\}$, skąd $\sup(-A) = 0$, oraz z kresu dolnego A , mamy $\inf(A) = 0$

1.10. Funkcje $f: (0, 3) \rightarrow \mathbb{R}$ i $g: [0, 3] \rightarrow \mathbb{R}$ są ciągłe oraz $f(1) = g(1) = -7$, $f(2) = g(2) = 7$. Wynika z tego, że

NIE A. obydwie funkcje są ograniczone

TAK B. 0 należy do zbiorów wartości obydwu funkcji

NIE C. obydwie funkcje są jednostajnie ciągłe

A. f nie musi być ograniczona, może mieć asymptotę w 3. **B.** wynika to z tw. Darboux. **C.** twierdzenie Cantora o jednostajnej ciągłości dotyczy przedziałów domkniętych, więc dla f to nie musi być prawda.

- 1.11.** Dana jest funkcja $f : (a, b) \rightarrow \mathbb{R}$ różniczkowalna na $(a, x_0) \cup (x_0, b)$, gdzie $x_0 \in (a, b)$. Wynika z tego, że

NIE **A.** f jest ciągła na (a, b)

NIE **B.** jeśli f jest ciągła na (a, b) , to jest też różniczkowalna na (a, b)

NIE **C.** f ma ograniczony zbiór wartości na każdym przedziale domkniętym zawartym w (a, b)

A. nie wiemy o tym co dzieje się w x_0 , choć na lewo od x_0 i na prawo funkcja jest różniczkowalna, a więc też ciągła. W x_0 może być np. asymptota. **B.** implikacja jest w drugą stronę. **C.** Zakładając, że w x_0 jest asymptota i $f(x) \rightarrow +\infty$ dla $x \rightarrow x_0$, to na przedziale $[a + \epsilon, x_0 + \epsilon]$ f nie ma ograniczonego zbioru wartości.

- 1.12.** Funkcja $f : \mathbb{R} \rightarrow \mathbb{R}$ jest różniczkowalna oraz $f(0) = f(1) = 0$. Wynika z tego, że skończona jest granica

TAK **A.** $\lim_{n \rightarrow \infty} nf\left(\frac{1}{n}\right)$

TAK **B.** $\lim_{n \rightarrow \infty} nf\left(\frac{n+1}{n}\right)$

TAK **C.** $\lim_{n \rightarrow \infty} nf\left(\frac{n}{n+1}\right)$

Każda z granic policzymy regułą de l'Hospitala:

$$\mathbf{A.} \lim_{n \rightarrow \infty} nf\left(\frac{1}{n}\right) = \lim_{n \rightarrow \infty} \frac{f\left(\frac{1}{n}\right)}{\frac{1}{n}} = \lim_{n \rightarrow \infty} \frac{(1/n)'f'\left(\frac{1}{n}\right)}{(1/n)'} = \lim_{n \rightarrow \infty} f'\left(\frac{1}{n}\right) = f'(0)$$

$$\mathbf{B.} \lim_{n \rightarrow \infty} nf\left(\frac{n+1}{n}\right) = \lim_{n \rightarrow \infty} \frac{f\left(\frac{n+1}{n}\right)}{\frac{1}{n}} = \lim_{n \rightarrow \infty} \frac{(1+1/n)'f'\left(1+\frac{1}{n}\right)}{(1/n)'} = \lim_{n \rightarrow \infty} \frac{(1/n)'f'\left(1+\frac{1}{n}\right)}{(1/n)'} = \lim_{n \rightarrow \infty} f'\left(1 + \frac{1}{n}\right) = f'(1)$$

$$\mathbf{C.} \lim_{n \rightarrow \infty} nf\left(\frac{n}{n+1}\right) = \lim_{n \rightarrow \infty} \frac{f\left(\frac{n}{n+1}\right)}{\frac{1}{n}} = \lim_{n \rightarrow \infty} \frac{\left(\frac{n}{n+1}\right)'f'\left(\frac{n}{n+1}\right)}{\left(\frac{1}{n}\right)'} = \lim_{n \rightarrow \infty} \frac{\frac{n+1-n}{(n+1)^2}f'\left(\frac{n}{n+1}\right)}{\frac{-1}{n^2}} = \lim_{n \rightarrow \infty} -\frac{n^2}{(n+1)^2}f'\left(\frac{n}{n+1}\right) = -f'(1)$$

Skoro f jest różniczkowalna, to jej pochodna jest skończona, a więc wszystkie odpowiedzi są prawdziwe.

- 1.13.** Dla $a, b \in \mathbb{R}$ funkcja $f_{a,b} : \mathbb{R} \rightarrow \mathbb{R}$ jest zdefiniowana wzorem

$$f_{a,b}(x) = \begin{cases} e^x & \text{dla } x < 0 \\ ax + b & \text{dla } x \geq 0 \end{cases}$$

Wtedy

NIE **A.** istnieje dokładnie jedna para $a, b \in \mathbb{R}$, taka że $f_{a,b}$ jest funkcją dwukrotnie różniczkowalną

TAK **B.** istnieje nieskończenie wiele par $a, b \in \mathbb{R}$, takich że $f_{a,b}$ jest funkcją ciągłą

TAK **C.** istnieje dokładnie jedna para $a, b \in \mathbb{R}$, taka że $f_{a,b}$ jest funkcją różniczkowalną

Liczmy pierwszą i drugą pochodną $f_{a,b}$:

$$f'_{a,b}(x) = \begin{cases} e^x & \text{dla } x < 0 \\ a & \text{dla } x \geq 0 \end{cases} \quad f''_{a,b}(x) = \begin{cases} e^x & \text{dla } x < 0 \\ 0 & \text{dla } x \geq 0 \end{cases}$$

Zauważmy, że do policzenia pierwszej pochodnej, musimy mieć warunek $f_{a,b}(0) = b = 1$, żeby funkcja była ciągła, więc a w tym przypadku może być dowolne, czyli **B.** to TAK. Do policzenia drugiej pochodnej, musi być znów $f'_{a,b}(0) = a = 1$, aby mieć ciągłość, ale po zastosowaniu drugiej pochodnej funkcja przestaje być ciągła, więc nigdy nie będzie dwukrotnie różniczkowalna, więc **A.** to NIE. Podpunkt **C.** jest prawdziwy, tylko dla $a = b = 1$ funkcja jest różniczkowalna ($f_{a,b}$ ciągła i pierwsza pochodna ciągła).

Przypis redakcji

Tu uwaga na blef: różniczkowalność nie oznacza ciągłości pochodnej, a dwukrotna różniczkowalność – ciągłości drugiej pochodnej.

1.14. Funkcja $f : (0; 1) \rightarrow \mathbb{R}$ zadana wzorem $f(x) = \sqrt{|\sin x|}$

- TAK** A. jest różniczkowalna
NIE B. jest ciągła i osiąga swój kres góryny
NIE C. ma pochodną ograniczoną

A. Jak wynika z własności pochodnej, jeżeli funkcje f i g są różniczkowalne, to ich złożenie również takie jest. Funkcja $f(x) = \sqrt{|\sin x|}$ jest więc różniczkowalna w przedziale $(0, 1)$, ponieważ funkcja $|\sin x|$ oraz pierwiastek kwadratowy są różniczkowalne w tym przedziale.

Pochodna zdefiniowana jest wzorem $f'(x) = \frac{\cos x \sin x}{2|\sin x|^{3/2}}$, a ponieważ na przedziale $(0, 1)$ $\sin(x) \geq 0$ to można porzucić wartość bezwzględną otrzymując $f'(x) = \frac{\cos x}{2\sqrt{\sin x}}$.

B. Funkcja ta jest ciągła ponieważ jest różniczkowalna. Jak łatwo zauważyc, wartość funkcji $f'(x)$ na przedziale $(0, 1]$ jest dodatnia, a zatem funkcja f na tym przedziale jest rosnąca. Kresem górnym jest więc $f(1)$ i nie jest on osiągany ponieważ badany przedział jest otwarty.

C. Zauważamy że dla $x \rightarrow 0$ wartość $\sin x \rightarrow 0$, a co za tym idzie $f'(x) \rightarrow \infty$. Zatem pochodna funkcji f nie jest ograniczona na przedziale $(0, 1)$.

1.15. Dany jest szereg $S(x) = \sum_{n=2}^{\infty} a_n x^n$ zbieżny dla $x \in (-1, 1)$ oraz rozbieżny w $x = -1$. Wynika z tego, że

- NIE** A. $S(x)$ jest rozbieżny w $x = 1$
TAK B. $S(x)$ jest rozbieżny w $x = -2$
TAK C. $S(x)$ jest różniczkowalny na odcinku $(-1, 1)$ oraz $S'(0) = 0$

Z treści możemy wywnioskować że środkiem podanego szeregu jest $z_0 = 0$, a jego promień wynosi 1 i w związku z tym:

- A. Na brzegu koła zbieżności może być różnie, a informacja o sytuacji na jednym z krańców nie daje nam pewności co do drugiego, w związku z tym jest to stwierdzenie fałszywe.
B. Punkt $x = -2$ znajduje się poza kołem zbieżności (oraz jego brzegami), a więc $S(x)$ jest w nim rozbieżny.
C. Odcinek $(-1, 1)$ należy do koła zbieżności szeregu, a więc jak wynika z twierdzenia o różniczkowalności sumy szeregu potęgowego, $S(x)$ jest na nim różniczkowalne oraz $S'(0) = 1! \cdot a_1 = 1 \cdot 0 = 0$.

1.16. Niech $f : \mathbb{R} \rightarrow \mathbb{R}$ będzie dana wzorem $f(x) = |-3x + x^2 - 5|$. Prawdą jest, że

- NIE** A. f jest różniczkowalna
NIE B. f jest dwukrotnie różniczkowalna
NIE C. jej pierwszy szereg Taylora w otoczeniu $x_0 = 0$ i $x = 1$ wynosi $\sqrt{2}$
A. i B. Nie jest różniczkowalna, podobnie jak $|x|$

Przypis redakcji

Powyżej zbyt słabe wytłumaczenie, istnieją przecież funkcje kwadratowe, które obłożone modułem są różniczkowalne.

C. Pierwszy szereg taylora w otoczeniu $x_0 = 0$ można obliczyć z $f(x_0) + f'(x_0)(x - x_0)$. W otoczeniu x_0 funkcja $f(x)$ zachowuje się jak $3x - x^2 + 5$. Mamy wtedy $f'(x) = -2x + 3$ zatem pierwszy szereg taylora w otoczeniu $x_0 = 0$ wynosi $f(x) \approx 8 - 2x$ dla $x = 1$ mamy $6, 6 \neq \sqrt{2}$

1.17. Niech $f : [0, 1] \rightarrow \mathbb{R}$ będzie ciągła. Wtedy

- NIE** A. $f'(x)$ istnieje dla każdego $x \in ([0, 1] \setminus A)$ gdzie A jest pewnym skończonym podzbiorem \mathbb{R}
TAK B. zbiór wartości f jest punktem lub przedziałem domkniętym

NIE C. f jest sumą pewnego szeregu potęgowego

A. Istnieje funkcja, która jest ciągła, a nie jest różniczkowalna w żadnym punkcie – funkcja Weierstrassa. Jeśli f jest jej obcięciem do $[0, 1]$, to **NIE**.

B. Tak – twierdzenie Weierstrassa o przyjmowaniu kresów.

C. Suma szeregu potęgowego jest klasy C^∞ . Nie mamy nic o tym, że funkcja f także jest C^∞ .

1.18. Niech $H_n := \sum_{j=1}^n \frac{1}{j}$ będzie n -tą liczbą harmoniczną. Wówczas

TAK A. promień zbieżności szeregu potęgowego $\sum_{n=1}^{\infty} H_n x^n$ wynosi 1

TAK B. $\lim_{n \rightarrow \infty} \frac{H_n}{\ln n} = 1$

NIE C. $\lim_{n \rightarrow \infty} (H_n - \ln n) = 0$

A. Obliczmy promień zbieżności korzystając ze wzoru $\frac{1}{R} = \lim_{n \rightarrow \infty} \sup \sqrt[n]{|a_n|}$. Możemy skorzystać z tw. o 3 ciągach żeby oszacować ciąg H_n .

$$1 = \lim_{n \rightarrow \infty} \sup \sqrt[n]{1} \leq \lim_{n \rightarrow \infty} \sup \sqrt[n]{H_n} \leq \lim_{n \rightarrow \infty} \sup \sqrt[n]{n} = 1$$

Więc promień jest równy 1.

B. Żeby odpowiedzieć na 2 poniższe pytania potrzebujemy znać fakt, że H_n rośnie tak samo jak $\log n$ oraz, że $\lim_{n \rightarrow \infty} (H_n - \ln n) = \gamma \approx 0.58$. Stąd odpowiedź do **B** to tak a do **C** to nie. **Dowód obu tych faktów jest zbyt skomplikowany na to repetytorium.**

1.19. Funkcja $f : \mathbb{R} \rightarrow \mathbb{R}$ jest zadana wzorem $f(x) = \sum_{n=1}^{\infty} \frac{\sin(nx)}{n^3}$. Wynika z tego, że

TAK A. funkcja f jest ograniczona na \mathbb{R}

TAK B. funkcja f jest różniczkowalna na \mathbb{R}

TAK C. spełniona jest nierówność $16 > f'(0) > \frac{1}{16}$

Szereg $\sum_{n=1}^{\infty} \frac{\sin(nx)}{n^3}$ jest zbieżny, ponieważ $\sum_{n=1}^{\infty} \left| \frac{\sin(nx)}{n^3} \right| \leq \sum_{n=1}^{\infty} \frac{1}{n^3} < \infty$. Wynika z tego, że f jest różniczkowalna (nawet nieskończonie wiele razy), więc odpowiedzi **A.** i **B.** są prawdziwe. Policzymy $f'(x) = \sum_{n=1}^{\infty} \frac{n \cos(nx)}{n^3}$, więc $f'(0) = \sum_{n=1}^{\infty} \frac{n \cos(0)}{n^3} = \sum_{n=1}^{\infty} \frac{1}{n^2}$. Każdy dorosły matematyk wie, że $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6} \approx \frac{3}{6} = \frac{1}{2}$, czyli **C.** to też **TAK**.

1.20. Ciąg funkcji $f_n(x) = x^n$ jest

TAK A. zbieżny punktowo na $[0, 1]$

NIE B. zbieżny jednostajnie na $[0, 1]$

NIE C. ograniczony na $[0, 1 + \varepsilon]$ dla pewnego $\varepsilon > 0$

A. Mamy $\lim_{n \rightarrow \infty} x^n = 0$ dla $x \in [0, 1)$ i $\lim_{n \rightarrow \infty} x^n = 1$ dla $x = 1$, więc ciąg f_n zbiega punktowo do

$$f(x) = \begin{cases} 0 & \text{dla } x \in (0, 1) \\ 1 & \text{dla } x = 1 \end{cases}$$

B. Nie będzie jednostajnej zbieżności, ponieważ dla każdego $n \in \mathbb{N}$ i $x = 2^{-1/n}$ jest: $f_n(x) - f(x) = \frac{1}{2} - 0 = \frac{1}{2} < \varepsilon$, co nie zachodzi dla żadnej liczby $\varepsilon < \frac{1}{2}$.

C. Weźmy $x = 1 + \frac{\varepsilon}{2} \in [0, 1 + \varepsilon]$. Wtedy ciąg x^n jest rozbieżny, więc nieograniczony.

1.21. Ciąg funkcyjny $\{f_n\}_{n \geq 1}$ składa się z funkcji $f_n : \mathbb{R} \rightarrow \mathbb{R}$ zdefiniowanych wzorami $f_n(x) = x - \frac{1}{n}$ dla $x \in \mathbb{R}$. Ten ciąg funkcyjny jest

TAK A. zbieżny punktowo i jednostajnie

TAK B. niemal jednostajnie zbieżny

TAK C. zbieżny punktowo do funkcji ciągły

A. Sprawdźmy zbieżność punktową. Oczywiście $\lim_{n \rightarrow \infty} f_n(x) = x$, więc jest zbieżny punktowo. Sprawdźmy zbieżność jednostajną.

$$|f(x) - f_n(x)| = \left| x - \left(x - \frac{1}{n} \right) \right| = \frac{1}{n} < \varepsilon$$

Więc jest też zbieżna jednostajnie.

Sprawdźmy czy jest jednostajnie zbieżna używając warunku Cauchy'ego

$$|f_n(x) - f_m(x)| = \left| x - \frac{1}{n} - \left(x - \frac{1}{m} \right) \right| = \left| \frac{1}{m} - \frac{1}{n} \right| < \varepsilon$$

B. Ciąg ten jest jednostajnie zbieżny więc jest też niemal jednostajnie zbieżny.

C. Funkcja $f(x) = x$ jest ciągła więc TAK.

1.22. Dana jest różniczkowalna funkcja $f : \mathbb{R} \rightarrow \mathbb{R}$, taka że $f'(0) = 0$. Wynika z tego, że

NIE A. f ma ekstremum lokalne w $x = 0$

TAK B. jeśli dla każdego $x \neq 0$ zachodzi $f'(x) < 0$, to f jest malejąca

TAK C. f jest ciągła w $x = 2019$

A. Kontrprzykładem jest funkcja $f(x) = x^3$ (w ogólności warunki z zadania to warunki konieczne istnienia ekstremum, ale niewystarczające)

B.

C. Z różniczkowalności funkcji f wynika jej ciągłość.

1.23. Niech $\{f_n\}_{n \geq 0}$ będzie ciągiem ograniczonych funkcji typu $(0, 1) \rightarrow \mathbb{R}$, takich że $\int_0^1 f_n(t) dt = 0$. Wtedy

NIE A. istnieje prawostronna granica f_0 w 0^+

NIE B. $\{f_n(x)\}_{n \geq 0}$ jest zbieżny do 0 dla każdego $x \in (0, 1)$

NIE C. jeśli $\{f_n(x)\}_{n \geq 0}$ jest zbieżny do 0 dla każdego $x \in (0, 1)$, to $\left\{ \int_0^1 |f_n(t)| dt \right\}_{n \geq 0}$ jest zbieżny do 0

A. Nie musi tak być. Zauważmy, że istnieje taka funkcja jak $\sin(1/x)$, która nie ma granicy w 0, ale istnieje całka $\int_0^1 \sin(1/x) dx$ i ma skońzoną, dodatnią wartość. Ponadto ta funkcja jest ograniczona, więc wystarczy wziąć f_0 jako taką sklejoną funkcję: od 0 do 1/2 weźmiemy wyżej wspomnianą funkcję, przeskalowaną tak, aby $\int_0^{1/2} f_0(x) dx$ było równe 1. Następnie od 1/2 do 1 weźmiemy po prostu funkcję stałe równą -2.

B. Nie – wystarczy wziąć dla danego x ciąg taki, że każda funkcja w ciągu ma na x wartość 1.

C. Nie jest to prawda. Weźmy ciąg funkcji f_n taki, że $f_n(x) = n$ jeśli $x < \frac{1}{n}$, zaś 0 wpp. Ich całka oczywiście jest równa 1, ponieważ wtedy liczymy pole prostokąta, który ma podstawę $1/n$, zaś wysokość n . Łatwo też widać, że dla każdego $x \in (0, 1)$ zachodzi, że $f_n(x) \rightarrow 0$. Dla każdego takiego x istnieje k takie, że $x > \frac{1}{k}$. Zatem dla $i > k$, wszystkie funkcje f_i będą spełniały $f_i(x) = 0$.

1.24. Niech $f : (-1, 1) \rightarrow \mathbb{R}$ będzie zdefiniowana wzorem

$$f(x) = \frac{1}{x^2 - 4x + 3}$$

Wtedy

NIE A. funkcja f jest ograniczona na $(-1, 1)$

TAK **B.** pochodna $f^{(1000)}(0)$ wynosi $1000! \left(\frac{1}{2} - \frac{1}{6 \cdot 3^{1000}}\right)$

TAK **C.** zachodzi $\int_{-1}^0 f(t) dt = \frac{\ln 3 - \ln 2}{2}$

Zapiszmy f jako

$$f(x) = \frac{1}{(x-1)(x-3)} = \frac{1}{2} \left(\frac{1}{x-3} - \frac{1}{x-1} \right).$$

A. Widzimy, że funkcja f ma w $x=1$ asymptotę pionową, więc nie będzie ograniczona.

B. Weźmy na warsztat $g(x) = \frac{1}{x-3}$. Licząc parę pierwszych pochodnych: $g'(x) = \frac{-1}{(x-3)^2}$, $g''(x) = \frac{2(x-3)}{(x-3)^4} = \frac{2}{(x-3)^3}$, $g'''(x) = \frac{-2 \cdot 3(x-3)^2}{(x-3)^6} = \frac{-2 \cdot 3}{(x-3)^4}$, itd. zauważymy, że wzór na k -tą pochodną funkcji g to $g^{(k)} = \frac{(-1)^k \cdot k!}{(x-3)^{k+1}}$. Oznacza to, że możemy napisać bardzo ładny wzór na k -tą pochodną funkcji f :

$$f^{(k)} = \frac{1}{2} \left(\frac{(-1)^k \cdot k!}{(x-3)^{k+1}} - \frac{(-1)^k \cdot k!}{(x-1)^{k+1}} \right),$$

stąd łatwo policzymy $f^{(1000)}(0) = \frac{1}{2} \left(\frac{1000!}{(-3)^{1001}} - \frac{1000!}{(-1)^{1001}} \right) = \frac{1000!}{2} \left(-\frac{1}{3 \cdot 3^{1000}} + 1 \right) = 1000! \left(\frac{1}{2} - \frac{1}{6 \cdot 3^{1000}} \right)$.

C. Skrupulatnie liczymy całeczkę

$$\begin{aligned} \int_{-1}^0 f(t) dt &= \int_{-1}^0 \frac{1}{(t-1)(t-3)} dt = \int_{-1}^0 \frac{1}{2} \left(\frac{1}{t-3} - \frac{1}{t-1} \right) dt = \frac{1}{2} \left(\int_{-1}^0 \frac{1}{t-3} dt - \int_{-1}^0 \frac{1}{t-1} dt \right) = \\ &= \frac{1}{2} (\ln|t-3| \Big|_{-1}^0 - \ln|t-1| \Big|_{-1}^0) = \frac{1}{2} (\ln(3) - \ln(4) - \ln(1) + \ln(2)) = \frac{\ln(3) - 2\ln(2) - 0 + \ln(2)}{2} = \frac{\ln(3) - \ln(2)}{2}. \end{aligned}$$

1.25. W pewnej metryce ρ zachodzi $\rho(x_1, x_2) = 7$. Dane są dwie otwarte kule, K_1 o środku x_1 i promieniu 1 oraz K_2 o środku w x_2 i promieniu 2. Wtedy

TAK **A.** $(K_2 \cup K_1)$ jest zbiorem otwartym

TAK **B.** istnieje kula otwarta o promieniu 8 zawierająca kulę K_1 i K_2

NIE **C.** zbiór $(K_1 \cap K_2)$ jest niepusty

A. Suma zbiorów otwartych jest otwarta.

B. i C. rysujemy obrazek i widać (wystellarzy na płaszczyźnie dla okręgów).

2

Geometria z algbrą liniową

Materiały teoretyczne z geometrii z algbrą liniową zostały opracowane na podstawie skryptu Pawła Bechlera, skryptu Lechosława Grochowskiego oraz filmów 3Blue1Brown.

Podstawa programowa

1. Definicja **grupy** i grupy przemiennej.
2. Ciała **liczb rzeczywistych i zespolonych**.
3. **Przestrzenie liniowe**, liniowa niezależność wektorów, baza i wymiar przestrzeni liniowej.
4. **Macierze i przekształcenia liniowe**, wyznaczniki.
5. **Wektory i wartości własne** macierzy i przekształceń liniowych.
6. Istnienie i jednoznaczność rozwiązań **układów liniowych**, eliminacja Gaussa.
7. Przestrzenie z **iloczynem skalarnym**.

2.1.

Grupy i ciała

W zbiorze liczb rzeczywistych mamy zdefiniowane dwa działania jednoargumentowe:

- wyznaczenie **liczby przeciwej** do danej: $x \mapsto -x$,
- dla liczby różnej od zera, wyznaczenie jej **odwrotności**: $x \mapsto \frac{1}{x} = x^{-1}$.

Analogicznie, definiujemy także działania dwuargumentowe – **dodawanie** oraz **mnożenie** dwóch liczb: $(x, y) \mapsto x + y$, $(x, y) \mapsto x \cdot y$. Odejmowanie i dzielenie można otrzymać jako złożenie odpowiednich operacji jedno- i dwuargumentowych.

Działania dodawania i mnożenia mają następujące własności:

- Dla dowolnych $a, b \in \mathbb{R}$ zachodzi $a + b = b + a$ oraz $a \cdot b = b \cdot a$. Mówimy, że działania są **przemienne**.
- Dla dowolnych $a, b, c \in \mathbb{R}$ zachodzi $a + (b + c) = (a + b) + c$ oraz $a \cdot (b \cdot c) = (a \cdot b) \cdot c$. Mówimy, że działania są **łączne**.
- Dla dowolnych $a, b, c \in \mathbb{R}$ zachodzi $a \cdot (b + c) = a \cdot b + a \cdot c$. Mówimy, że **mnożenie jest rozdzielne względem dodawania**.

W zbiorze liczb rzeczywistych istnieją również pewne szczególne elementy, które oznaczamy symbolami 0 i 1. Dla dowolnej liczby $a \in \mathbb{R}$:

$$a + 0 = 0 + a = a \quad \text{oraz} \quad a \cdot 1 = 1 \cdot a = a$$

Liczby te nazywamy **elementem neutralnym** odpowiednio dodawania i mnożenia.

Nie zawsze prawdą jest, że wynik działania jedno- lub dwuargumentowego musi należeć do tego samego zbioru, co liczby wejściowe. Jeśli tak jest, to mówimy, że dany zbiór jest **zamknięty** ze względu na rozważane działanie lub że dane działanie jest działańiem **wewnętrzny**m w tym zbiorze.

Przykład 1.

Oto kilka przykładów zamkniętości wybranych podzbiorów \mathbb{R} ze względu na podstawowe działania:

- W liczbach całkowitych \mathbb{Z} wewnętrzne jest działanie wyznaczenia liczby przeciwej, natomiast własności tej nie ma działania wyznaczenia liczby odwrotnej.
- Zbiór liczb rzeczywistych różnych od zera $\mathbb{R}_{\neq 0}$ jest zamknięty na działanie mnożenia, ale nie jest zamknięty na działanie dodawania (bo np. $1 + (-1) = 0 \notin \mathbb{R}_{\neq 0}$).

Grupy

Jeśli zbiór jest jednocześnie zamknięty ze względu na dane działanie dwuargumentowe (dodawanie lub mnożenie) i odpowiadające mu działanie jednoargumentowe (liczba przeciwna lub odwrotna), a dodatkowo element neutralny danego działania też należy do tego zbioru, to mówimy, że taki zbiór jest **grupą**. Formalna definicja jest następująca:

Zbiór G wraz z

- elementem wyróżnionym $e \in G$,
- działaniem dwuargumentowym „ \diamond ” o własnościach
 - zbiór G jest zamknięty ze względu na „ \diamond ”
 - dla dowolnych $x, y, z \in G$ zachodzi $x \diamond (y \diamond z) = (x \diamond y) \diamond z$ (łączność)
 - dla dowolnego $x \in G$ zachodzi $e \diamond x = x \diamond e = x$ (element neutralny)
 - dla każdego $x \in G$ istnieje $y \in G$, taki że $x \diamond y = e$ (element odwrotny)

nazywamy grupą z działaniem „ \diamond ” i elementem neutralnym e . Grupę możemy zapisać jako trójkę uporządkowaną (G, \diamond, e) .

Jeżeli dodatkowo spełniony jest warunek przemienności (dla każdych $x, y \in G$ zachodzi $x \diamond y = y \diamond x$), to taką grupę nazywamy grupą **abelową** lub **przemienną**.

Przykład 2.

Oto kilka przykładów grup (oraz struktur, które grupami nie są):

- Zbiór \mathbb{Z} z działaniem „ $+$ ” i elementem neutralnym 0 jest grupą abelową – w oczywisty sposób spełnia on wszystkie warunki wymienione w definicji grupy.
- Zbiór \mathbb{R} z działaniem „ \cdot ” i elementem neutralnym 1 nie jest grupą, ponieważ liczba 0 nie posiada elementu odwrotnego.
- Niech $\mathbb{Z}_n = \{0, 1, \dots, n - 1\}$ oraz oznaczmy przez „ $+$ ” dodawanie modulo n . Wówczas $(\mathbb{Z}_n, +, 0)$ jest n -elementową grupą abelową (dla dowolnego n).

Ciało

Jak łatwo zauważyc, w zbiorze liczb rzeczywistych \mathbb{R} mamy do czynienia z dwiema podstawowymi grupami, jedną z dodawaniem, a drugą z mnożeniem:

$$(\mathbb{R}, +, 0) \quad \text{oraz} \quad (\mathbb{R}_{\neq 0}, \cdot, 1)$$

Taki opis struktury zbioru liczb rzeczywistych nie uwzględnia jednak tego, że działanie mnożenia jest rozdzielne względem dodawania. W związku z tym wprowadzamy jeszcze jedną strukturę: **ciało**.

Ciałem nazywamy zbiór X wraz z działaniami „ $+$ ” i „ \cdot ”, zwanymi dodawaniem i mnożeniem, oraz wyróżnionymi elementami 0 i 1 , taki że

- $(X, +, 0)$ jest grupą abelową,
- mnożenie jest przemienne i łączne oraz $(X \setminus \{0\}, \cdot, 1)$ jest grupą abelową,
- mnożenie jest rozdzielne względem dodawania.

Ciało możemy zapisać jako piątkę uporządkowaną $(X, +, \cdot, 0, 1)$.

Przykład 3.

Oto przykłady ciał:

- Ciało liczb rzeczywistych: $(\mathbb{R}, +, \cdot, 0, 1)$.
- Ciało liczb wymiernych: $(\mathbb{Q}, +, \cdot, 0, 1)$.
- Dla liczby pierwszej p oraz „ $+$ ” i „ \cdot ” jako dodawania i mnożenia modulo p , $(\mathbb{Z}_p, +, \cdot, 0, 1)$ jest ciałem mającym p elementów (dla koneserów: jest to przykład ciała Galois).

Zestaw zadań

2.1. Niech (G, \diamond, e) będzie grupą, $a, b, c \in G$ i $a \diamond c = e$. Wtedy

- A. $a \diamond b = b \diamond a$
- B. $a \diamond c = c \diamond a$
- C. $(b \diamond a) \diamond (c \diamond b) = b \diamond b$

2.2. Niech $\mathbb{G} = (G, \circ, e)$ będzie grupą. Rozważmy grupę $\mathbb{G}^{op} = (G, \cdot, e)$, gdzie $x \cdot y = y \circ x$ dla $x, y \in G$. Wtedy

- A. dla funkcji $f : \mathbb{G} \rightarrow \mathbb{G}$ określonej wzorem $f(x) = x^{-1}$ oraz dowolnej podgrupy $\mathbb{H} = (H, \circ, e)$ grupy \mathbb{G} zbiór $f(\mathbb{H})$ jest podgrupą grupy \mathbb{G}^{op}
- B. grupy \mathbb{G} i \mathbb{G}^{op} są izomorficzne
- C. $\mathbb{G} = \mathbb{G}^{op}$ wtedy i tylko wtedy, gdy grupa \mathbb{G} jest abelowa

2.2.

Liczby zespolone

Oznaczmy obiekt $\sqrt{-1}$ jako i (**jednostka urojona**). Co jasne, nie jest to liczba rzeczywista, ale taki obiekt okazuje się bardzo przydatny – pozwala nam na spierwiastkowanie dowolnej liczby rzeczywistej ujemnej, np. $(5i)^2 = -25$ albo $(i\sqrt{2})^2 = -2$.

Takie obiekty postaci bi , gdzie $b \in \mathbb{R}$ możemy dodać do znanych już liczb rzeczywistych, tworząc obiekty typu $a + bi$ (dla $a, b \in \mathbb{R}$), czyli **liczby zespolone**. Zespolone z dwóch części: rzeczywistej (tej bez i) oraz urojonej (tej z i). Formalnie, jeśli $z = a + bi$, to

- liczbę rzeczywistą a nazywamy **częścią rzeczywistą** liczby zespolonej z i oznaczamy $\text{Re } z$,
- liczbę rzeczywistą b nazywamy **częścią urojoną** liczby zespolonej z i oznaczamy $\text{Im } z$,

Zbiór wszystkich liczb zespolonych oznaczamy symbolem \mathbb{C} (ang. *complex*). Na zbiorze \mathbb{C} mamy określone działania:

- **dodawania:**

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

z elementem neutralnym $0 = 0 + 0i$,

- **mnożenia:**

$$(a + bi) \cdot (c + di) = (ac - bd) + (ad + bc)i$$

z elementem neutralnym $1 = 1 + 0i$,

- wyznaczenia **liczby przeciwnej** do $z = a + bi$:

$$-z = -a - bi,$$

takiej że $z + (-z) = 0$,

- wyznaczenia **liczby odwrotnej** do $z = a + bi$:

$$z^{-1} = \frac{1}{z} = \frac{1}{a + bi} = \frac{a}{a^2 + b^2} - \frac{b}{a^2 + b^2}i,$$

takiej że $z \cdot z^{-1} = 1$.

Sprzężenie liczby zespolonej $z = a + bi$ definiujemy jako

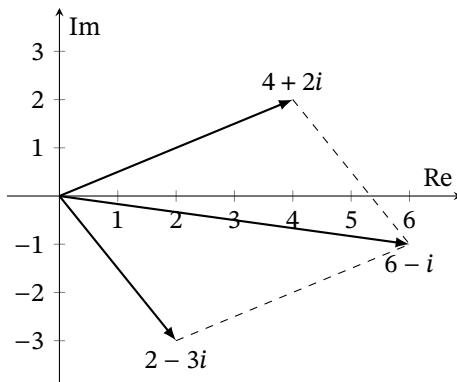
$$\bar{z} = a - bi$$

Jest to o tyle ciekawy obiekt, że pozwala nam na pokazanie ważnej własności wielomianów: jeśli wielomian zmiennej zespolonej o współczynnikach rzeczywistych ma pierwiastek nierzeczywisty, to ma również pierwiastek do niego sprzężony. (**przyp. red.: potrzebny przykład**)

■ Interpretacja wektorowa

Skoro liczby zespolone, podobnie jak punkty na płaszczyźnie, możemy jednoznacznie definiować parami liczb rzeczywistych $(a, b) := a + bi$, to ich naturalną reprezentacją jest **płaszczyzna zespolona**, czyli układ współrzędnych, w którym osi x -ów odpowiada część rzeczywista, a osi y -ów – część zespolona. W ten sposób możemy interpretować liczby zespolone jako wektory.

Zauważmy, że wtedy dodawanie liczb zespolonych jest jak dodawanie wektorów z \mathbb{R}^2 , co obrazuje poniższy rysunek:



Podobnie jak dla liczb rzeczywistych, tak i dla zespolonych można zdefiniować **moduł** jako odległość od zera w układzie współrzędnych: dla $z = a + bi$ mamy

$$|z| = \sqrt{a^2 + b^2}$$

Dla tak określonego modułu zachodzi również **nierówność trójkąta**: dla dowolnych $z, w \in \mathbb{C}$

$$|z + w| \leq |z| + |w|$$

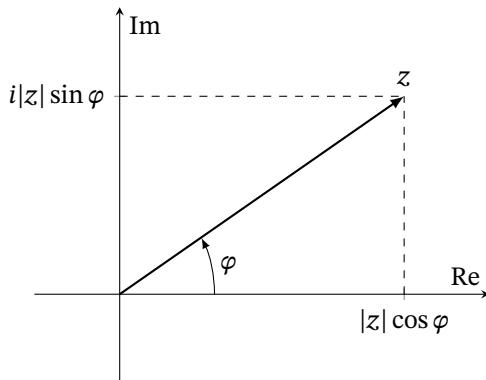
■ Postać trygonometryczna

Do liczb zespolonych można wprowadzić koncepcję współrzędnych biegunowych – zamiast określać wektor jako parę współrzędnych x, y , będziemy ustalać jego długość i kąt skierowania.

Niech $z = x + iy \in \mathbb{C}$ i $z \neq 0$. Oznaczmy literą φ kąt pomiędzy półosią rzeczywistą dodatnią i wektorem odpowiadającym z i ustalmy, że jest to kąt zorientowany przeciwne do ruchu wskazówek zegara. Wówczas $\cos \varphi = \frac{x}{|z|}$, $\sin \varphi = \frac{y}{|z|}$ oraz

$$z = |z|(\cos \varphi + i \sin \varphi)$$

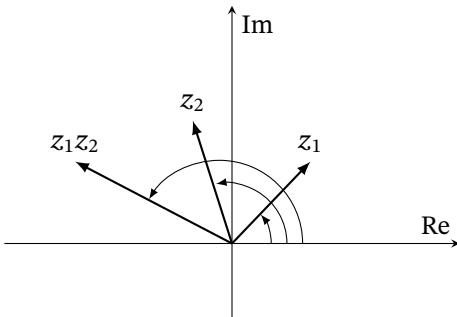
Kąt φ nazywamy **argumentem** liczby zespolonej z i oznaczamy $\varphi = \arg z$.



Przy powyższym zapisie, mając $z = |z|(\cos \varphi + i \sin \varphi)$ i $w = |w|(\cos \theta + i \sin \theta)$, możemy zgrabnie wymnożyć obie liczby:

$$zw = |z| \cdot |w|(\cos(\varphi + \theta) + i \sin(\varphi + \theta)).$$

Widzimy więc, że przy mnożeniu liczb zespolonych ich moduły się mnożą, a argumenty dodają.



Postać trygonometryczną możemy skrócić do **zapisu wykładniczego**:

$$e^{i\varphi} = \cos \varphi + i \sin \varphi.$$

Przy tym zapisie mnożenie liczb zespolonych wygląda prościej:

$$zw = |z| |w| e^{i(\varphi+\theta)}$$

Potęgowanie liczb zespolonych możemy w łatwy sposób wykonać, wykorzystując **wzór de Moivre'a**: jeżeli liczba n jest całkowita, natomiast $z = |z|e^{i\varphi} = |z|(\cos \varphi + i \sin \varphi)$, to

$$z^n = |z|^n e^{in\varphi} = |z|^n (\cos n\varphi + i \sin n\varphi).$$

Przykład 1.

Obliczymy $(1+i)^{21}$. W postaci trygonometrycznej mamy $1+i = \sqrt{2}(\cos \frac{\pi}{4} + i \sin \frac{\pi}{4})$, zatem

$$(1+i)^{21} = (\sqrt{2})^{21} \left(\cos \frac{21\pi}{4} + i \sin \frac{21\pi}{4} \right) = 2^{10}\sqrt{2} \left(\cos \frac{5\pi}{4} + i \sin \frac{5\pi}{4} \right) = 2^{10}(-1-i)$$

Pierwiastki zespolone

Rozpatrzymy rozwiązania równania postaci $x^n = z$, gdzie $z \in \mathbb{C} \setminus \{0\}$. Każde takie rozwiązanie $x \in \mathbb{C}$ będziemy nazywać **pierwiastkiem zespolonym** stopnia n z liczby zespolonej z .

Równanie $x^n = z$ ma zawsze n pierwiastków, które są postaci

$$x_k = \sqrt[n]{|z|} e^{i(\varphi+2k\pi)/n} = \sqrt[n]{|z|} \left(\cos \left(\frac{\varphi+2k\pi}{n} \right) + i \sin \left(\frac{\varphi+2k\pi}{n} \right) \right).$$

Zwróćmy uwagę, że pierwiastek zespolony to nie liczba, a zbiór liczb. Geometrycznie są one wierzchołkami wielokąta foremnego wpisanego w okrąg o środku w punkcie $(0,0)$ i promieniu $\sqrt[n]{|z|}$ (dla $n=2$ otrzymujemy dwa punkty symetryczne względem zera). Wyjaśnienie tego faktu jest intuicyjnie proste: o ile w liczbach rzeczywistych przyjmujemy jako pierwiastki zawsze wartości dodatnie, odrzucając ujemne (np. $\sqrt{9}=3$, mimo że $(-3)^2$ to też 9), o tyle w liczbach zespolonych nie ma powodu, żeby wyróżniać którykolwiek z pierwiastków.

Rozpatrzymy jeszcze jeden szczególny przypadek: gdy $z = 1$, otrzymujemy następujący wzór na n różnych zespolonych **pierwiastków z jedności** stopnia n :

$$x_k = e^{i2k\pi/n} = \cos \left(\frac{2k\pi}{n} \right) + i \sin \left(\frac{2k\pi}{n} \right).$$

Zauważmy, że wtedy $1 = 1 + 0i$ zawsze jest jednym z pierwiastków. Graficznie wystarczy więc wyznaczyć wierzchołki n -kąta foremnego położonego na okręgu o promieniu 1, którego jednym z wierzchołków jest jedynka.

Przypis redakcji

Do sekcji „pierwiastki zespolone” warto byłoby dodać rysunki.

Zestaw zadań

2.3. Liczba zespolona x_0 jest rozwiązaniem równania $x^2 + x + 1 = 0$. Wynika z tego, że

- A. liczba x_0^2 też jest rozwiązaniem tego równania
- B. liczba $1/x_0$ też jest rozwiązaniem tego równania
- C. $x_0^3 = 1$

2.4. Strukturę grupy z działaniem mnożenia liczb zespolonych i 1 jako elementem neutralnym ma zbiór

- A. $\{2^k z \in \mathbb{C} : z^{2019} = 1\}$
- B. $\{z \in \mathbb{C} : \operatorname{Re}(z) \cdot \operatorname{Im}(z) = 0\}$
- C. $\{z \in \mathbb{C} : k \in \mathbb{Z}, |z| = 1\}$

2.3.

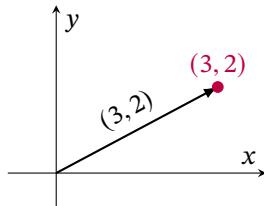
Przestrzenie liniowe

Aby móc odpowiednio wprowadzić intuicję stojącą za przestrzeniami liniowymi, zaczniemy od ujednolicenia pojęcia **wektora**. Rozważając zagadnienia algebry liniowej umawiamy się, że **punktem przyłożenia każdego z rozważanych wektorów jest zero** (o zadanym wymiarze, a więc może to być np. $(0, 0)$, $(0, 0, 0)$ itd.).

Ponadto, w wielu przypadkach będziemy utożsamiać **pojęcie punktu z pojęciem wektora**, tzn. punkt (x, y) będziemy intuicyjnie kojarzyć z wektorem o początku w zerze i końcu w punkcie (x, y) . W związku z tym współrzędne wektora zapisujemy czasem w nawiasach okrągłych (jak w przypadku punktu) i nie zawsze używamy notacji nawiasów kwadratowych.

Przykład 1.

Punkt $(3, 2)$ można utożsamiać z wektorem $(3, 2)$, co pokazuje poniższy rysunek.



Oznaczmy przez \mathbb{K} pewne ciało i przez V zbiór pewnych wektorów, taki że $(V, +, 0)$ jest grupą abelową. Założymy też, że w zbiorze X określone jest działanie „ \cdot ” mnożenia wektora $\vec{v} \in V$ przez skalar $a \in \mathbb{K}$, przy czym spełnia ono następujące własności: dla dowolnych $a, b \in \mathbb{K}$ i $\vec{v}, \vec{w} \in V$

- mnożenie przez skalar jest rozdzielne względem dodawania wektorów:

$$a(\vec{v} + \vec{w}) = a\vec{v} + a\vec{w},$$

- mnożenie przez wektor jest rozdzielne względem dodawania skalarów:

$$(a + b)\vec{v} = a\vec{v} + b\vec{v},$$

- dwukrotne mnożenie wektora przez skalar można zastąpić mnożeniem skalarów:

$$a \cdot (b \cdot \vec{v}) = (a \cdot b) \cdot \vec{v},$$

- elementem neutralnym mnożenia przez skalar jest element neutralny mnożenia skalarów $1 \in \mathbb{K}$:

$$1 \cdot \vec{v} = \vec{v}.$$

Wówczas zbiór V wraz z tak zdefiniowanymi działaniami „ $+$ ” i „ \cdot ” jest **przestrzenią liniową** nad ciałem \mathbb{K} . Elementy przestrzeni liniowej nazywamy wektorami, a ciała \mathbb{K} – skalarami.

Przykład 2.

Oto kilka przykładów przestrzeni liniowych (oraz struktur, które przestrzeniami liniowymi nie są):

- Standardowa przestrzeń rzeczywistych wektorów n -wymiarowych \mathbb{R}^n jest, w oczywisty sposób, przestrzenią liniową nad \mathbb{R} . Nie jest ona jednak przestrzenią liniową nad \mathbb{C} , gdyż np. mamy $i \cdot (1, 1) = (i, i)$ – pomnożenie wektora z \mathbb{R}^n przez skalar z \mathbb{C} sprawia, że opuszczamy przestrzeń wektorów rzeczywistych.
- Ogół funkcji wielomianowych jednej zmiennej rzeczywistej x o współczynnikach rzeczywistych $\mathbb{R}[x]$ jest przestrzenią liniową nad \mathbb{R} . Przykładowo, $x^2 + 1$ oraz $2x^3 + 5x^2 + 3$ są wektorami tej przestrzeni. Możemy je do siebie dodawać i mnożyć przez rzeczywiste skalary, a działania te spełniają wszystkie warunki wymagane w definicji przestrzeni liniowej.

- Wektory z \mathbb{R}^2 postaci $\{(2a, a) \mid a \in \mathbb{R}\}$ rozpinają przestrzeń liniową. Gdybyśmy narysowali zbiór punktów tożsamych z tymi wektorami, uzyskalibyśmy prostą przechodzącą przez $(0, 0)$. Nie jest jednak prawdą, że punkty z dowolnej prostej utworzą przestrzeń wektorową. Na przykład, punkty z prostej o równaniu $y = x + 1$ są postaci $(x, x + 1)$ i nie tworzą przestrzeni wektorowej, gdyż m.in. punkty $(0, 1)$ i $(1, 2)$ do niej należą, a ich suma $(1, 3)$ – już nie.

Jak widać, wektorami mogą być obiekty dość dalekie od definicji ze szkoły średniej (czyli przestrzeni \mathbb{R}^n). Taki ogólne podejście pozwoli przenosić wiele ciekawych zależności ze standardowego \mathbb{R}^n do innych struktur.

Fakt istnienia w przestrzeniach liniowych **wektora i skalara zerowego** definiuje nam kilka dodatkowych własności: jeżeli V jest przestrzenią liniową nad ciałem \mathbb{K} , to dla każdego wektora $\vec{v} \in V$ oraz skalar $a \in \mathbb{K}$ zachodzą zależności:

- $0 \cdot \vec{v} = a \cdot \vec{0} = 0$ (gdzie „ 0 ” po lewej to zero w ciele \mathbb{K} , a „ $\vec{0}$ ” po prawej to element neutralny dodawania w V),
- $a\vec{v} = \vec{0} \Leftrightarrow a = 0 \vee \vec{v} = \vec{0}$,
- $(-a)\vec{v} = a(-\vec{v}) = -(a\vec{v})$, gdzie przez $-\vec{v} \in V$ rozumiemy wektor przeciwny do \vec{v} .

■ Podprzestrzenie liniowe

Założymy, że V jest przestrzenią liniową nad ciałem \mathbb{K} i $P \subset V$ jest jej niepustym podzbiorem, który, wraz z odziedzicznymi z V działaniami dodawania i mnożenia przez skalar, także jest przestrzenią liniową nad \mathbb{K} . Mówimy wówczas, że P jest **podprzestrzenią liniową** przestrzeni V .

Przykład 3.

Weźmy przestrzeń \mathbb{R}^2 nad \mathbb{R} . Jej przykładową podprzestrzenią jest zbiór $\{(3a, 5a) \mid a \in \mathbb{R}\}$. Z kolei przykładową podprzestrzenią tej przestrzeni jest $\{\vec{0}\}$ – podprzestrzeń jednoelementowa, składająca się wyłącznie z wektora zerowego.

Prawdą jest, że $P \neq \emptyset$ jest podprzestrzenią liniową V wtedy i tylko wtedy, gdy zachodzą wszystkie z poniższych warunków:

- $\vec{0} \in P$
- dla dowolnych $\vec{v}, \vec{w} \in P$ także $\vec{v} + \vec{w} \in P$,
- dla dowolnych $\vec{v} \in P$ i $a \in \mathbb{K}$ także $a\vec{v} \in P$.

■ Liniowa niezależność

Założymy, że V jest przestrzenią liniową nad ciałem \mathbb{K} . **Kombinacją liniową wektorów** $\vec{v}_1, \dots, \vec{v}_n \in V$ o współczynnikach $a_1, \dots, a_n \in \mathbb{K}$ nazywamy każdy z wektorów postaci

$$\vec{v} = a_1\vec{v}_1 + \dots + a_n\vec{v}_n.$$

Przykład 4.

Każdy wektor $\vec{v} \in \mathbb{R}^3$ jest kombinacją liniową wektorów jednostkowych $\vec{e}_1 = (1, 0, 0)$, $\vec{e}_2 = (0, 1, 0)$, $\vec{e}_3 = (0, 0, 1)$:

$$\vec{v} = (a_1, a_2, a_3) = a_1\vec{e}_1 + a_2\vec{e}_2 + a_3\vec{e}_3$$

Niech V będzie przestrzenią liniową, natomiast $G \subset V$ pewnym zbiorem wektorów. Najmniejszą podprzestrzeń liniową zawierającą wszystkie wektory z G nazywamy **podprzestrzenią rozpiętą** przez G i oznaczamy jako $\text{span } G$. Jeżeli $G = \{\vec{g}_1, \dots, \vec{g}_k\}$, to

$$\text{span } G = \{a_1\vec{g}_1 + \dots + a_k\vec{g}_k \mid a_1, \dots, a_k \in \mathbb{K}\}$$

Podprzestrzenie rozpięte przez pewien zbiór wektorów składają się więc ze wszystkich kombinacji liniowych wektorów z tego zbioru.

Przykład 5.

Weźmy przestrzeń funkcji wielomianowych $\mathbb{R}[x]$. Wtedy $\text{span } \{1, x^2\}$ to ogół wielomianów postaci $w(x) = ax^2 + b$, gdzie $a, b \in \mathbb{R}$. Z kolei $\text{span } \{1, x^2 + 1\}$ to ogół wielomianów postaci $w(x) = c(x^2 + 1) + d$, gdzie $c, d \in \mathbb{R}$, czyli tak naprawdę ogół wielomianów postaci $w(x) = ax^2 + b$ – taka zmiana zbioru rozpinającego przestrzeń nie dała w tym wypadku niczego nowego.

A czym jest $\text{span } \{x^2 + 1\}$? To zbiór wszystkich wielomianów postaci $ax^2 + a$ dla $a \in \mathbb{R}$. Jest on mniejszy niż poprzedni (tzn. poprzedni ma wszystkie występujące tu wielomiany i jeszcze jakieś inne, np. $x^2 + 2$).

Przykład 6.

Pozostajemy w przestrzeni $\mathbb{R}[x]$. Tym razem weźmy jednak nieskończony zbiór rozpinający przestrzeń: $G = \{1, x, x^2, x^3, \dots\}$. Widać, że w $\text{span } G$ znajdują się wszystkie wielomiany, a ponieważ wielomiany są podprzestrzenią, w $\text{span } G$ nie będzie nic więcej. Może trochę dziwić fakt, że choć np. $2x^3 + 5x^2 - 7x + 4 \in \text{span } G$, to $1 + x + x^2 + x^3 + \dots \notin \text{span } G$ – weźmy jednak pod uwagę, że w podprzestrzeniach rozpiętych mamy jedynie skończone sumy przeskalowanych wektorów bazowych.

Przykład 7.

Rozważmy przestrzeń \mathbb{R}^3 i weźmy zbiór jednoelementowy $\{(1, 2, 3)\}$. Wtedy

$$\text{span } \{(1, 2, 3)\} = \{a(1, 2, 3) \mid a \in \mathbb{R}\},$$

czyli podprzestrzenią rozpiętą jest zbiór wszystkich przeskalowań wektora $(1, 2, 3)$, a więc wektory

$$(0, 0, 0), (1, 2, 3), (2, 4, 6), (0.5, 1, 1.5), (-1, -2, -3) \text{ itd.}$$

Geometrycznie będzie to prosta w \mathbb{R}^3 przechodząca przez punkt $(0, 0, 0)$.

Dodajmy teraz wektor $(2, 0, 3)$, otrzymując

$$\text{span } \{(1, 2, 3), (2, 0, 3)\} = \{a(1, 2, 3) + b(2, 0, 3) \mid a, b \in \mathbb{R}\}.$$

Ustalając $b = 0$ i wstawiając wszystkie możliwe a , wygenerujemy tę samą prostą, co poprzednio. Dla innych wartości b otrzymamy inne proste, równolegle do siebie wzajemnie. Wszystkie te proste biegną w tym samym kierunku co wektor $(2, 0, 3)$, więc łącząc je wszystkie w całość dochodzimy do wniosku, że nasza przestrzeń $\text{span } \{(1, 2, 3), (2, 0, 3)\}$ to pewna płaszczyzna umieszczona w \mathbb{R}^3 (złożona ze wszystkich powstały prostych).

A co ze $\text{span } \{(1, 2, 3), (2, 0, 3), (-1, 2, -2)\}$? Dodanie wektora $(-1, 2, -2)$ nie zmieni naszej przestrzeni – wystarczy zauważyc, że

$$(-1, 2, -2) = (1, 2, 1) + (-1)(2, 0, 3),$$

więc z pomocą $(-1, 2, -2)$ nie wygenerujemy żadnych nowych kombinacji liniowych.

Rozważając powyższe przykłady, zajmiemy się teraz kwestią zbędnych generatorów, czyli takich wektorów, których dodanie do zbioru rozpinającego przestrzeń nie ma żadnego wpływu na otrzymaną przestrzeń. Nietrudno zauważyc, że jeśli pewien wektor jest kombinacją liniową pozostałych wektorów, to można go odrzucić, gdyż nic nowego nie wniesie.

Zbiór wektorów nazywamy zbiorem wektorów **liniowo zależnych**, jeśli pewien element tego zbioru można przedstawić jako kombinację liniową pewnych pozostałych.

Natomiast układ wektorów niespełniający powyższego warunku jest **liniowo niezależny**. Formalnie, ma on więc następującą własność: dla dowolnych $a_1, \dots, a_n \in \mathbb{K}$ zachodzi

$$a_1\vec{v}_1 + \dots + a_n\vec{v}_n = 0 \iff a_1 = \dots = a_n = 0$$

Przykład 8.

Liniowo niezależny jest zbiór wektorów $\{(1, 0, 0), (1, 1, 0), (1, 1, 1)\}$ – nie da się otrzymać żadnego z wektorów poprzez kombinację liniową pozostałych.

Natomiast zbiór

$$\{(1, 0, 0), (1, 1, 0), (1, 1, 1), (5, 6, 7)\}$$

jest liniowo zależny, bo na przykład

$$(5, 6, 7) = 7 \cdot (1, 1, 1) - (1, 1, 0) - (1, 0, 0)$$

Intuicyjnie, o liniowej niezależności możemy myśleć w następujący sposób: każdy wektor w zbiorze liniowo niezależnym niesie ze sobą *pewną unikalną informację*, której nie da się uzyskać za pomocą pozostałych wektorów w tym zbiorze.

Przytoczymy jeszcze jeden ważny fakt: **dowolny podzbiór zbioru wektorów liniowo niezależnych także jest liniowo niezależny**.

Baza i wymiar przestrzeni liniowej

Jeszcze raz przyjrzymy się kwestii zbędnych generatorów przestrzeni rozpiętych. Wiemy już, że kluczowa jest tu cecha liniowej niezależności zbioru rozpinającego przestrzeń.

Układ wektorów $\vec{v}_1, \dots, \vec{v}_n \in V$ nazywamy **bazą** przestrzeni V , jeżeli jest on liniowo niezależny i rozpina przestrzeń V (tj. $V = \text{span}\{\vec{v}_1, \dots, \vec{v}_n\}$).

Co ważne, każdy wektor z przestrzeni **zapisuje się dokładnie na jeden sposób** jako kombinacja liniowa wektorów z bazy tej przestrzeni. Intuicyjnie więc baza stanowi zbiór wektorów, z których „budujemy” całą przestrzeń bazową (tj. wszystkie jej elementy).

Baza jest zarówno maksymalnym układem wektorów liniowo niezależnych w swojej przestrzeni, jak i minimalnym zbiorem wektorów rozpinających daną przestrzeń. Jak należy to rozumieć?

- **Każdy właściwy podzbiór bazy jest zbiorem liniowo niezależnym, rozpinającym pewną podprzestrzeń bazowej przestrzeni.** Intuicyjnie: skoro baza składa się z wektorów liniowo niezależnych, to usunięcie z niej dowolnego elementu zabiera nam *pewną unikalną informację*, którą ten element wnosił. Co za tym idzie – nie będziemy w stanie wygenerować (za pomocą kombinacji liniowych pozostałych elementów) niektórych elementów z bazowej przestrzeni, do których otrzymania był niezbędny usunięty przez nas element. Taki podzbiór bazy rozepnie więc pewną, mniejszą podprzestrzeń bazowej przestrzeni.
- **Każdy właściwy nadzbiór bazy jest zbiorem liniowo zależnym.** Skoro baza rozpina całą przestrzeń, tj. nie ma w niej elementów, których nie dałoby się uzyskać poprzez kombinację liniową wektorów z bazy, to dodanie jakiegokolwiek elementu nie może przynieść nam już żadnej dodatkowej informacji, która pozwoliłaby nam uzyskać nowe elementy (bo takie nie istnieją). Mamy więc pewność, że każdy dodany do bazy element jest kombinacją liniową wektorów z bazy.

Przykład 9.

Rozważmy przestrzeń \mathbb{R}^3 i jej **bazę kanonczną** (tzn. złożoną z wektorów jednostkowych):

$$\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$$

Nietrudno sprawdzić, że rzeczywiście jest to baza: jej wektory są w oczywisty sposób liniowo niezależne, a

każdy wektor z \mathbb{R}^3 można zapisać jako liniową kombinację wektorów bazowych:

$$(a, b, c) = a(1, 0, 0) + b(0, 1, 0) + c(0, 0, 1)$$

Inną przykładową bazą przestrzeni \mathbb{R}^3 jest

$$\{(1, 0, 0), (7, 7, 0), (42, 42, 42)\}.$$

Z przedstawionych powyżej dwóch faktów możemy wywnioskować, że wszystkie bazy danej przestrzeni składają się z tej samej liczby wektorów. Tę liczbę nazywamy **wymiarom** przestrzeni liniowej V i oznaczamy przez $\dim V$. Jeżeli V nie ma bazy skończonej, przyjmujemy $\dim V = \infty$.

Przykład 10.

Oto kilka przykładów wymiarów baz różnych przestrzeni:

- Każda z przestrzeni \mathbb{R}^n dla $n \in \mathbb{N}_+$ ma wymiar n , gdyż np. jej baza kanoniczna liczy właśnie n elementów (n wektorów bazowych, po jednym na każdą z n współrzędnych wektora).
- Podprzestrzeń zerowa $\{(0, 0, 0)\}$ przestrzeni \mathbb{R}^3 ma wymiar równy 0.
- Przestrzeń funkcji wielomianowych zmiennej rzeczywistej stopnia co najwyżej 2 o współczynnikach rzeczywistych ma wymiar 3, gdyż jej baza kanoniczna $\{1, x, x^2\}$ liczy właśnie 3 elementy.
- Przestrzeń $\mathbb{R}[x]$ ma wymiar równy ∞ – skończona baza nie pozwoliłaby bowiem zapisywać wielomianów odpowiednio wysokich stopni.

Z pojęciem wymiaru związane są bardzo ważne własności.

Załóżmy, że $\dim V < \infty$ i $U \subseteq V$ jest podprzestrzenią liniową V . Wówczas

- $\dim U \leq \dim V$,
- $\dim U = \dim V$ wtedy i tylko wtedy, gdy $U = V$.

Załóżmy, że $\dim X = n < \infty$. Wówczas

- każdy liniowo niezależny układ wektorów z X można uzupełnić do bazy przestrzeni X ,
- z każdego układu wektorów rozpinających przestrzeń X można wybrać bazę przestrzeni X ,
- jeżeli pewne n wektorów $x_1, \dots, x_n \in X$ jest liniowo niezależne, to są one bazą przestrzeni X ,
- jeżeli pewne n wektorów $x_1, \dots, x_n \in X$ rozpina przestrzeń X , to są one jej bazą.

■ Przecięcie i suma podprzestrzeni

Załóżmy, że X jest przestrzenią liniową nad ciałem \mathbb{K} i $Y_j \subset X$ są podprzestrzeniami liniowymi, gdzie $j \in J$. Wówczas zbiór będący **przecięciem** tych przestrzeni,

$$Y = \bigcap_{j \in J} Y_j,$$

też jest podprzestrzenią liniową w X .

Przykład 11.

Przecięciem podprzestrzeni $U = \{x \in \mathbb{R}^3 : x_1 + x_2 = 0\}$, $V = \{x \in \mathbb{R}^3 : x_2 + x_3 = 0\}$ jest

$$U \cap V = \text{span}\{(1, -1, 1)\} \subset \mathbb{R}^3$$

Załóżmy, że X jest przestrzenią liniową nad ciałem \mathbb{K} i $U, V \subset X$ są podprzestrzeniami liniowymi. **Suma** podprzestrzeni U i V to zbiór

$$U + V = \{u + v \in X : u \in U, v \in V\}.$$

Taka suma podprzestrzeni też jest podprzestrzenią liniową w X .

W ogólności nie jest prawdą, że $U + V = U \cup V$. Łatwo widać, że $U \cup V \subset U + V$. Ponadto, zbiór $U \cup V$ nie musi być podprzestrzenią liniową w X .

Następujące warunki są równoważne:

- $U \cap V = \{0\}$,
- dla każdego wektora $x \in U + V$ wektory $u \in U$ i $v \in V$ takie, że $x = u + v$ są wyznaczone jednoznacznie.

Przykład 12.

W przestrzeni wielomianów 3 stopnia o współczynnikach rzeczywistych $\mathbb{R}[x]_3$ zachodzi

$$\text{span}(1, x^2) + \text{span}(1 + x, x^2 + x^3) = \text{span}(1, x, x^2, x^3) = \mathbb{R}[x]_3$$

Załóżmy, że X jest przestrzenią liniową nad ciałem \mathbb{K} i $U, V \subset X$ są podprzestrzeniami liniowymi. Jeśli U i V są rozłączne (tj. $U \cap V = \{0\}$), mówimy, że podprzestrzeń $U + V$ jest **sumą prostą** podprzestrzeni U i V i piszemy $U + V = U \oplus V$.

Przykład 13.

W przestrzeni wielomianów 3 stopnia o współczynnikach rzeczywistych $\mathbb{R}[x]_3$ zachodzi

$$\text{span}(1, x^2) \oplus \text{span}(x, x^3) = \text{span}(1, x, x^2, x^3) = \mathbb{R}[x]_3$$

Jeżeli $Y, Z \subset X$ są podprzestrzeniami skończonego wymiaru, to

- wymiary podprzestrzeni $Y \cap Z$ oraz $Y + Z$ też są skończone,
- $\dim(Y + Z) = \dim Y + \dim Z - \dim(Y \cap Z)$.

Zestaw zadań

2.5. Niech X będzie przestrzenią liniową wymiaru 15. Wynika z tego, że

- A. każdy układ 20 wektorów z X jest liniowo zależny
- B. każdy układ 10 wektorów z X jest liniowo niezależny
- C. każda baza X składa się z 15 wektorów

2.6. Dana jest przestrzeń X wymiaru 10 oraz podprzestrzenie U, V , takie że $\dim(U) = \dim(V)$ oraz $X = U + V$. Wynika z tego, że

- A. $\dim(U) = 5$
- B. $\dim(U) \geq 5$
- C. $\dim(U \cap V) = 0$

2.7. Dane są elementy przestrzeni liniowej nad ciałem liczb rzeczywistych $\vec{a}, \vec{b}, \vec{c}$. Prawdą jest, że

- A. jeżeli $(\vec{a}, \vec{b}, \vec{c})$ jest niezależny liniowo, to $(\vec{a} - \vec{b}, \vec{b} - \vec{c}, \vec{c} - \vec{a})$ jest niezależny liniowo
- B. jeżeli $(\vec{a}, \vec{b}, \vec{c})$ jest niezależny liniowo, to $(\vec{a} + \vec{b}, \vec{b} + \vec{c}, \vec{c} + \vec{a})$ jest niezależny liniowo
- C. jeżeli $(\vec{a} + \vec{b}, \vec{b} + \vec{c}, \vec{c} + \vec{a})$ jest niezależny liniowo, to $(\vec{a}, \vec{b}, \vec{c})$ jest niezależny liniowo

2.4. Macierze i przekształcenia liniowe

Macierzą (nad ciałem \mathbb{K}) wymiaru $m \times n$ nazywamy tablicę prostokątną

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

gdzie $a_{i,j} \in \mathbb{K}$ dla $1 \leq i \leq m$, $1 \leq j \leq n$. Można też zapisać taką macierz jako

$$A = [a_{i,j}]_{i=1,\dots,m, j=1,\dots,n}$$

Macierz $m \times n$ ma m wierszy i n kolumn. Pierwszy indeks odpowiada wierszowi, a drugi kolumnie. Piszemy wtedy, że macierz należy do zbioru macierzy $\mathbb{K}^{m,n}$.

Specyficznym przypadkiem macierzy jest macierz $\mathbb{K}^{n,1}$ lub po prostu macierz \mathbb{K}^n , którą nazywamy **wektorem** długości n :

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Przykłady innych często spotykanych macierzy:

- **macierz zerowa** – jej elementy to same zera,
- **macierz diagonalna** – macierz kwadratowa zawierająca niezerowe wartości jedynie na diagonali (przekątnej), zapisujemy $D = \text{diag}(d_1, \dots, d_n)$,
- **macierz jednostkowa** rozmiaru $n \times n$, taka że $I_n = \text{diag}(1, \dots, 1)$,
- **macierz górnoodolnotrójkątna** – zawierająca niezerowe wartości jedynie na diagonali oraz ponad/poniżej niej,
- **macierz permutacji** – macierz kwadratowa, która ma w każdym wierszu i w każdej kolumnie dokładnie jedną jedynkę oraz zera na pozostałych miejscach.

Działania na macierzach

Niech

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \in \mathbb{K}^{m,n}$$

Transpozycją macierzy $A \in \mathbb{K}^{m,n}$ nazywamy macierz

$$A^T = \begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{m,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \cdots & a_{m,n} \end{bmatrix} \in \mathbb{K}^{n,m}$$

o zamienionych kolumnach z wierszami (dla macierzy niekwadratowych zmieniają się także wymiary macierzy). Macierz A jest **symetryczna** jeśli zachodzi $A = A^T$ oraz **antysymetryczna** jeśli $A = -A^T$.

Hermitowskie sprzężenie macierzy A to jej transpozycja wraz ze sprzężeniem każdego z jej elementów, na przykład

$$\begin{bmatrix} 1+i & 2 & 2-i \\ 2+3i & i & 0 \end{bmatrix}^H = \begin{bmatrix} 1-i & 2-3i \\ 2 & -i \\ 2+i & 0 \end{bmatrix}$$

i, jak widać, $A^H = \overline{A^T} = (\overline{A})^T$. Macierz A jest hermitowska, gdy $A = A^H$.

Dodawanie i odejmowanie macierzy określone jest jedynie dla macierzy o tych samych wymiarach – odpowiadające sobie elementy (o tych samych indeksach) należy dodać lub odjąć i utworzyć w ten sposób macierz wynikową.

Mnożenie macierzy A z macierzą B jest zdefiniowane jedynie, gdy A jest wymiarów $m \times k$ oraz B jest wymiarów $k \times n$, czyli macierz po lewej musi mieć tyle kolumn, ile ta po prawej ma wierszy. Widać to, gdy zapiszemy macierze w sposób wygodny do mnożenia:

$$\begin{bmatrix} 1 & -1 & 1 & 2 \\ 2 & 0 & 1 & 3 \\ 3 & 1 & 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 & 2 \\ 0 & 3 & -2 \end{bmatrix} = \begin{bmatrix} 5 & 1 & 2 & 13 \\ 0 & -2 & 1 & -17 \end{bmatrix}$$

Własności mnożenia macierzy (A, B, C są macierzami takimi, że odpowiednie działania są zdefiniowane, c jest skalarem):

- $(A + B) \cdot C = A \cdot C + B \cdot C$
- $A \cdot (B + C) = A \cdot B + A \cdot C$
- $c(A \cdot B) = cA \cdot B = A \cdot cB$
- $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
- $(A \cdot B)^T = B^T \cdot A^T$
- jeśli A to macierz zerowa, to $A \cdot B$ też jest zerowa
- $A \cdot B$ niekoniecznie wynosi $B \cdot A$, dla macierzy niekwadratowych nie jest w ogóle określone
- jeśli $A \cdot B = [0]_{m,n}$, to niekoniecznie A lub B muszą być macierzami zerowymi

Macierz kwadratowa $A \in \mathbb{K}^{n,n}$ jest **nieosobienna** (odwrotna), gdy istnieje macierz kwadratowa $A^{-1} \in \mathbb{K}^{n,n}$ odwrotna do niej, taka że $A \cdot A^{-1} = A^{-1} \cdot A = I_n$. Tylko dla macierzy kwadratowych istnieją macierze odwrotne.

■ Wyznacznik macierzy

Oznaczmy jako $A_{i,j} \in \mathbb{K}^{n-1,n-1}$ macierz otrzymaną z macierzy $A \in \mathbb{K}^{n,n}$ poprzez usunięcie i -tego wiersza i j -tej kolumny. Niech $n = 1, 2, 3, \dots$ oraz $A = [a_{i,j}]_{i,j=1}^n \in \mathbb{K}^{n,n}$. **Wyznacznik** stopnia n to funkcja $\det = \det_n : \mathbb{K}^{n,n} \rightarrow \mathbb{K}$ zdefiniowana w następujący sposób:

- dla $n = 1$: $\det_1[a_{1,1}] = a_{1,1}$,
- dla $n > 1$: $\det_n A = \sum_{i=1}^n (-1)^{i+1} a_{i,1} \cdot \det_{n-1} A_{i,1}$ – rozwinięcie Laplace'a.

Przykład 1.

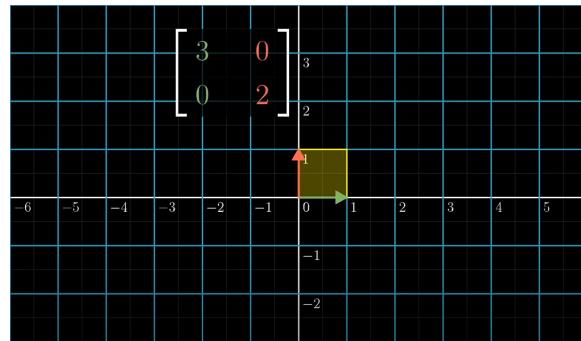
Obliczymy wyznacznik macierzy o wymiarach 2×2 za pomocą rozwinięcia Laplace'a: jeśli $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, to

$$\det A = \det_2 A = a \cdot \det_1 [d] - c \cdot \det_1 [b] = ad - bc$$

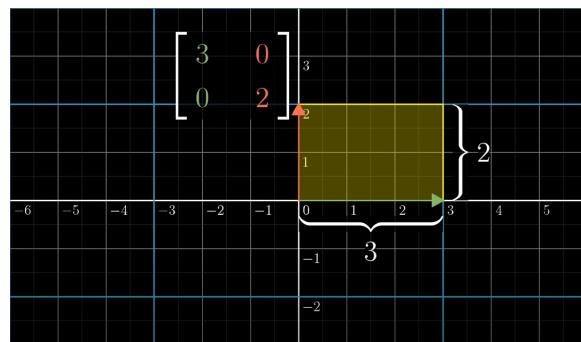
Przy liczeniu wyznaczników przydaje się także **twierdzenie Cauchy'ego**: jeżeli $A, B \in \mathbb{K}^{n,n}$, to

$$\det_n(AB) = \det_n(A) \cdot \det_n(B).$$

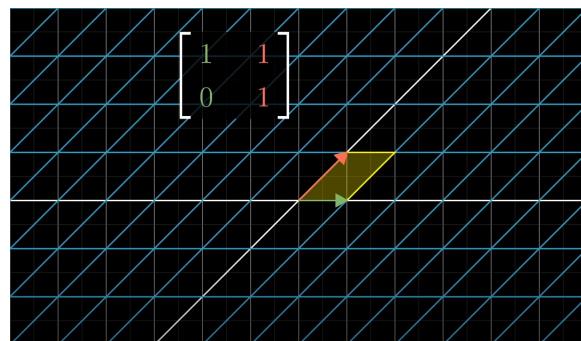
Rozważymy teraz **interpretację geometryczną wyznacznika**: odpowiada ona temu, jak macierz jako przekształcenie liniowe (o tym więcej w następnym dziale) **skaluje pole powierzchni** przypadające na dany jej fragment:



macierz $\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$ przekształca kwadrat o wymiarach 1×1 na prostokąt o wymiarach 3×2 , więc jej wyznacznik jest równy 6 (pole obszaru wzrosło sześciokrotnie),



a macierz $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ ma wyznacznik 1: chociaż zmienia ona ułożenie układu współrzędnych, widać, że początkowy kwadrat zmienił się jedynie w romb o takim samym polu powierzchni.



Z tą wiedzą nietrudno zauważyc, że wyznacznik macierzy odwrotnej musi wynosić $\det(A^{-1}) = \frac{1}{\det(A)}$. Skoro macierz odwrotna cofa zmiany wprowadzone na układzie współrzędnych, w szczególności musi przywrócić pole początkowego kwadratu do pierwotnej powierzchni.

■ Obraz, jądro i rząd macierzy

Obrazem macierzy $A \in \mathbb{K}^{m,n}$ nazywamy zbiór wektorów, które mogą powstać przez przemnożenie ich przez macierz A :

$$\text{im } A = \{A\vec{x} \in \mathbb{K}^m : x \in \mathbb{K}^n\}$$

Obraz macierzy jest podprzestrzenią liniową w \mathbb{K}^m rozpiętą przez jej kolumny.

Przykład 2.

Wyznaczmy obraz macierzy $A = \begin{bmatrix} 1 & 2 & 0 \\ -1 & 1 & -3 \\ 1 & 0 & 2 \end{bmatrix} \in \mathbb{R}^{3,3}$. Weźmy dowolny wektor $\vec{x} = [x_1, x_2, x_3]^T \in \mathbb{R}^3$.

Wtedy

$$\begin{bmatrix} 1 & 2 & 0 \\ -1 & 1 & -3 \\ 1 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} + x_2 \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} + x_3 \begin{bmatrix} 0 \\ -3 \\ 2 \end{bmatrix}.$$

Zatem

$$\text{im } A = \text{span} \left(\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -3 \\ 2 \end{bmatrix} \right).$$

Jądem macierzy $A \in \mathbb{K}^{m,n}$ nazywamy zbiór tych wektorów, które przemnożone przez macierz A stają się wektorem zerowym:

$$\ker A = \{\vec{x} \in \mathbb{K}^n : A\vec{x} = \vec{0}\}.$$

Jądro macierzy jest podprzestrzenią liniową w \mathbb{K}^n .

Rząd macierzy $A \in \mathbb{K}^{m,n}$ to liczba $\text{rank } A = \dim(\text{im } A)$. Zachodzą równości $\text{rank}(A) = \text{rank}(A^T) = \text{rank}(A^H)$. Rząd macierzy opisuje, ile jest w niej liniowo niezależnych kolumn. Jeśli potrafimy zeszadkować macierz i nie otrzymamy żadnej kolumny z samymi zerami, rząd jest maksymalny.

Ważne jest **twierdzenie o wymiarze obrazu i jądra**: jeśli $A \in \mathbb{K}^{m,n}$, to wówczas

$$\dim(\text{im } A) + \dim(\ker A) = n.$$

■ Warunki odwracalności

Istnieje wiele równoważnych warunków sprawdzenia, czy macierz jest nieosobliwa, najważniejsze z nich to:

- wyznacznik jest niezerowy
- jądro macierzy zawiera jedynie wektor zerowy
- rząd macierzy jest maksymalny
- kolumny macierzy są liniowo niezależne
- wiersze macierzy są liniowo niezależne
- macierz A^T jest nieosobliwa

■ Przekształcenia liniowe

Niech X, Y będą przestrzeniami liniowymi nad ciałem \mathbb{K} . Powiemy, że odwzorowanie (funkcja) $f : X \rightarrow Y$ jest **przekształceniem liniowym** z przestrzeni X w przestrzeń Y , jeżeli dla dowolnych wektorów $x, y \in X$ i dowolnych skalarów $\alpha, \beta \in \mathbb{K}$ zachodzi

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y).$$

Zbiór wszystkich przekształceń liniowych z przestrzeni X w przestrzeń Y oznaczamy $L(X, Y)$. Jeżeli $\dim X = n$ i $\dim Y = m$ oraz $m, n < \infty$, to

$$\dim(L(X, Y)) = m \cdot n.$$

Możemy traktować macierze jako przekształcenia liniowe. Rozważmy funkcję $f : \mathbb{R}^3 \rightarrow \mathbb{R}^2$, która wektorowi $[x_1, x_2, x_3]^T$ przyporządkowuje wektor $[x_1 + x_2 + x_3, 2x_1 - 3x_2 + 6x_3]^T$. Wtedy

$$f(\vec{x}) = \begin{bmatrix} 1 & 1 & 1 \\ 2 & -3 & 6 \end{bmatrix} \vec{x}$$

Każda macierz $A \in \mathbb{K}^{m,n}$ jednoznacznie zadaje przekształcenie liniowe $f \in L(\mathbb{K}^n, \mathbb{K}^m)$.

Macierz przekształcenia liniowego $f : X \rightarrow Y$ w bazach A i B , gdzie A jest bazą X oraz B jest bazą Y , zapisujemy jako $M(f)_A^B$. Wyznaczanie najlepiej obrazuje przykład.

Przykład 3.

Funkcję $f : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ określono wzorem

$$f([x_1, x_2, x_3]^T) = [2x_1 + x_2 - x_3, x_1 - x_2 + x_3]$$

Układ

$$A = ([1, 0, 1]^T, [0, 1, 2]^T, [2, 1, 0]^T)$$

jest bazą \mathbb{R}^3 , zaś układ

$$B = ([0, 1]^T, [1, 1]^T)$$

jest bazą \mathbb{R}^2 . Żeby wyznaczyć i -tą kolumnę przekształcenia f , należy zapisać $f(a_i)$ jako kombinację liniową wektorów bazy B i odczytać odpowiednie współczynniki:

$$\begin{aligned} f([1, 0, 1]^T) &= [1, 2]^T = 1 \cdot [0, 1]^T + 1 \cdot [1, 1]^T \\ f([0, 1, 2]^T) &= [-1, 1]^T = 2 \cdot [0, 1]^T - 1 \cdot [1, 1]^T \\ f([2, 1, 0]^T) &= [5, 1]^T = -4 \cdot [0, 1]^T + 5 \cdot [1, 1]^T \end{aligned}$$

Zatem przekształceniu f odpowiada macierz

$$M(f)_A^B = \begin{bmatrix} 1 & 2 & -4 \\ 1 & -1 & 5 \end{bmatrix}.$$

Możemy też mówić o przekształceniach liniowych w kontekście innych przestrzeni liniowych, np. przestrzeni wielomianów.

Przykład 4.

Rozważmy funkcję z przestrzeni $f : \mathbb{R}[x]_2 \rightarrow \mathbb{R}$

$$f(p) = p(0) - 2p(-1).$$

Tak zadana funkcja f jest przekształceniem liniowym z $\mathbb{R}[x]_2$ w \mathbb{R} .

■ Monomorfizmy, epimorfizmy, izomorfizmy

Przekształcenie liniowe $f \in L(X, Y)$ nazwiemy

- **monomorfizmem**, jeżeli f jest różnowartościowe,
- **epimorfizmem**, jeżeli $\text{im } f = Y$,
- **izomorfizmem**, jeżeli jest jednocześnie mono- i epimorfizmem.

Założymy, że $\dim X < \infty$ oraz $f \in L(X, Y)$. Następujące warunki są równoważne:

- f jest monomorfizmem
- $\ker f = \{0\}$
- $\dim(\text{im } f) = \dim X$
- dla pewnej bazy x_1, \dots, x_n przestrzeni X układ $f(x_1), \dots, f(x_n)$ jest bazą przestrzeni $\text{im } f$.

Założymy, że przekształcenie liniowe $f \in L(X, Y)$ ma w pewnych bazach macierz $A \in \mathbb{K}^{m,n}$. Wtedy zachodzi

- f jest monomorfizmem $\Leftrightarrow \ker A = \{0\}$,
- f jest epimorfizmem $\Leftrightarrow \text{rank } A = m$.

■ Przestrzenie dualne

Niech X będzie przestrzenią liniową nad ciałem \mathbb{K} . **Przestrzeń dualna** (sprzężona) do X to przestrzeń $L(X, \mathbb{K})$. Oznaczamy ją symbolem X^* . Elementy przestrzeni X^* (czyli przekształcenia liniowe z X w \mathbb{K}) nazywamy **funkcjonałami liniowymi** na X . Funkcjonały liniowe często oznacza się symbolami x^*, u^* , itp.

Jeżeli X jest przestrzenią liniową i $\dim X = n < +\infty$ to także $\dim X^* = n$.

Niech X będzie przestrzenią liniową nad ciałem \mathbb{K} z bazą (x_1, \dots, x_n) . Jasne jest, że dla każdego $x \in X$ istnieją jednoznacznie wyznaczone współczynniki $\alpha_1, \dots, \alpha_n$, takie że $x = \alpha_1 x_1 + \dots + \alpha_n x_n$. Każde z przekształceń $x_i^* : X \rightarrow \mathbb{K}$ zdefiniowane jako $x_i^*(x) = \alpha_i$ jest funkcjonałem liniowym na X .

Układ takich funkcjonałów (x_1^*, \dots, x_n^*) jest bazą przestrzeni X^* i dla każdego $x \in X$ zachodzi wzór

$$x = \sum_{i=1}^n x_i^*(x) x_i$$

Intuicyjnie, funkcjonały x_i^* wyznaczają współczynniki kombinacji liniowej dowolnego wektora $x \in X$ w danej bazie (x_1, \dots, x_n) .

Bazę (x_1^*, \dots, x_n^*) nazywamy **bazą dualną** (sprzężoną) do bazy (x_1, \dots, x_n) .

Przykład 5.

Rozważmy bazę $(1, x, x^2)$ przestrzeni liniowej wielomianów 2 stopnia o współczynnikach rzeczywistych $\mathbb{R}[x]_2$. Bazą dualną do tej bazy będzie (p_1^*, p_2^*, p_3^*) , gdzie

$$p_1^*(ax^2 + bx + c) = c, \quad p_2^*(ax^2 + bx + c) = b, \quad p_3^*(ax^2 + bx + c) = a$$

Zestaw zadań

2.8. W przestrzeni liniowej $\mathbb{R}[x]_{<3}$ wielomianów rzeczywistych stopnia mniejszego niż 3 dane są wielomia-

ny

$$p_1(t) = 1, \quad p_2(t) = 1 + t^2, \quad p_3(t) = 1 + t + t^2.$$

Wynika z tego, że

- A. układ (p_1, p_2, p_3) stanowi bazę przestrzeni $\mathbb{R}[x]_{<3}$
- B. jeśli $q(t)$ jest wielomianem stopnia pierwszego, to układ (p_1, p_2, q) jest liniowo niezależny
- C. macierz $A = (p_i(j))_{i,j=1}^3$ jest nieosobliwa

2.9. Niech $M = \{(a_{ij})_{i,j=1}^3 \in \mathbb{R}^{3,3} : a_{11} + a_{22} + a_{33} = 0\}$. Prawdą jest, że

- A. zbiór M jest zamknięty na działanie mnożenia macierzy
- B. zbiór M jest podprzestrzenią liniową w $\mathbb{R}^{3,3}$
- C. istnieje niezerowa podprzestrzeń liniowa N przestrzeni $\mathbb{R}^{3,3}$, taka że $M \cap N = \{0\}$, gdzie 0 oznacza macierz zerową

2.10. Macierz rzeczywista A ma n wierszy i k kolumn. Wynika z tego, że

- A. jeśli wymiar jądra macierzy A wynosi n , to $k \geq n$
- B. jeśli rząd macierzy A jest równy n , to $k \geq n$
- C. jeśli rząd macierzy A jest równy k , to macierz $A^T A$ jest nieosobliwa

2.11. Macierz A jest wymiaru 10×10 oraz $\det(A) = 5$. Oznacza to, że

- A. kolumny A są liniowo niezależne
- B. A jest odwracalna
- C. $\det(A^{-1}) = -5$

2.12. W macierzy kwadratowej A o wymiarach $n \times n$ dokładnie n elementów jest jedynkami, a pozostałe elementy są zerami. Wynika z tego, że

- A. $\det(A) \in \{-1, 0, 1\}$
- B. jeśli A jest nieosobliwa, to jest macierzą permutacji
- C. jeśli którykolwiek jedynkę w macierzy A zastąpimy zerem, to wyznacznik powstałej macierzy będzie równy zero

2.13. Niech X, Y będą przestrzeniami liniowymi, a $f : X \rightarrow Y$ przekształceniem liniowym. Prawdą jest, że

- A. jeśli $\ker f = 0$, to $\dim X \leq \dim Y$
- B. jeśli $\ker f \neq 0$, to $\dim X > \dim Y$
- C. jeśli $\dim X > \dim Y$, to $\ker f \neq 0$

2.14. Dane są przestrzenie liniowe X i X' , podprzestrzenie $U, V \subseteq X$ oraz podprzestrzenie $U', V' \subseteq X'$, takie że $X = U \oplus V$ oraz $X' = U' \oplus V'$. Prawdą jest, że

- A. jeżeli $g : U \rightarrow U'$, $h : V \rightarrow V'$ są przekształceniemi liniowymi, to istnieje dokładnie jedno przekształcenie liniowe $f : X \rightarrow X'$, takie że $f|_U = g$ i $f|_V = h$
- B. jeżeli X i X' są izomorficzne, to U, U' są izomorficzne lub U, V' są izomorficzne
- C. jeżeli U jest izomorficzne z U' oraz V izomorficzne z V' , to X jest izomorficzne z X'

2.15. Przekształcenie $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ dane jest wzorem $f \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) = \begin{bmatrix} 2x_1 - x_2 \\ x_1 + x_2 \end{bmatrix}$. Wynika z tego, że

- A. w pewnej bazie przestrzeni \mathbb{R}^2 macierzą przekształcenia f jest $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

- B. f jest przekształceniem różniczkowym i „na”
- C. macierzą przekształcenia f w bazie $([1, 0]^T, [1, 1]^T)$ jest $\begin{bmatrix} 1 & 1 \\ -1 & 2 \end{bmatrix}$
- 2.16.** W przestrzeni $\mathbb{R}[x]_{\leq 1}$ wielomianów rzeczywistych stopnia co najwyżej 1 funkcjonały φ_1 i φ_2 dane są wzorami
- $$\varphi_1(p) = p(0), \quad \varphi_2(p) = p(1),$$
- gdzie $p \in \mathbb{R}[x]_{\leq 1}$. Wynika z tego, że
- A. funkcjonały φ_1 i φ_2 są liniowo niezależne
- B. funkcjonał φ dany wzorem $\varphi(p) = p'(\frac{1}{2})$ jest pewną kombinacją liniową φ_1 i φ_2
- C. wielomiany $p_1(t) = 1 - t$ i $p_2(t) = t$ oraz funkcjonały φ_1 i φ_2 stanowią wzajemnie do siebie sprzężone bazy odpowiednio w $\mathbb{R}[x]_{\leq 1}$ oraz przestrzeni sprzężonej $(\mathbb{R}[x]_{\leq 1})^*$

2.5. Wektory i wartości własne

Wektor własny v i wartość własna λ stanowią parę własną, jeśli

$$Av = \lambda v.$$

Przykład 1.

Obliczanie wartości własnych macierzy. Niech $A = \begin{bmatrix} -5 & -4 \\ 8 & 7 \end{bmatrix}$. Obliczamy wielomian charakterystyczny macierzy A , czyli zapisujemy macierz

$$\begin{bmatrix} \lambda + 5 & -4 \\ 8 & \lambda - 7 \end{bmatrix}$$

i obliczamy miejsca zerowe wyznacznika. Tutaj otrzymujemy $\lambda_1 = 3$, $\lambda_2 = -1$. Wektory własne liczymy z definicji $Av_1 = \lambda_1 v_1$ i mamy $v_1 = [-1, 2]^T$, $v_2 = [1, -1]^T$.

Zestaw zadań

2.17. Niech $A \in \mathbb{C}^{2n,2n}$, gdzie $n \in \mathbb{N}$ oraz $n > 0$, będzie macierzą nieosobliwą. Wynika z tego, że

- A. liczba różnych wartości własnych macierzy A jest parzysta
- B. w zbiorze wartości własnych macierzy A nie ma zera
- C. jeśli λ jest wartością własną macierzy A^{-1} , to λ^{-1} jest wartością własną macierzy A

2.6. Układy równań liniowych

Przypis redakcji

Rozdział nie jest jeszcze stworzony – w historii przeanalizowanych na potrzeby tego repetytorium egzaminów nie pojawiło się żadne zadanie z konkretnie tego materiału.

2.7.

Przestrzenie z iloczynem skalarnym

Niech X będzie przestrzenią liniową nad ciałem \mathbb{K} . **Iloczyn skalarny** na X to funkcja $\varphi : X \times X \rightarrow \mathbb{K}$, którą będziemy zapisywać jako $\langle x, y \rangle = \varphi(x, y)$, o następujących własnościach:

- dla dowolnych $x, y_1, y_2 \in X$ i $\alpha_1, \alpha_2 \in \mathbb{K}$:

$$\langle x, \alpha_1 y_1 + \alpha_2 y_2 \rangle = \alpha_1 \langle x, y_1 \rangle + \alpha_2 \langle x, y_2 \rangle,$$

- dla dowolnych $x, y \in X$

$$\langle x, y \rangle = \overline{\langle y, x \rangle},$$

- dla dowolnego $x \in X \setminus \{0\}$

$$\langle x, x \rangle > 0.$$

Standardowy iloczyn skalarny w \mathbb{R}^n przyjmuje postać

$$\langle \vec{x}, \vec{y} \rangle = \vec{x}^T \vec{y} = \sum_{i=1}^n x_i y_i$$

Założymy, że $\langle \cdot, \cdot \rangle$ jest iloczynem skalarnym na przestrzeni liniowej X nad ciałem \mathbb{K} . Wówczas

- dla dowolnych $x_1, x_2, y \in X, \alpha_1, \alpha_2 \in \mathbb{K}$

$$\langle \alpha_1 x_1 + \alpha_2 x_2, y \rangle = \overline{\alpha_1} \langle x_1, y \rangle + \overline{\alpha_2} \langle x_2, y \rangle,$$

- dla dowolnego $x \in X$

$$\langle x, 0 \rangle = \langle 0, x \rangle = 0,$$

- dla dowolnego $x \in X$

$$\langle x, x \rangle = 0 \text{ wtedy i tylko wtedy, gdy } x = 0,$$

- jeżeli $x \in X$ i dla każdego $y \in X$ zachodzi $\langle x, y \rangle = 0$, to $x = 0$.

Przestrzeń liniową skończenie wymiarową X nad ciałem \mathbb{R} z iloczynem skalarnym $\langle \cdot, \cdot \rangle$ nazywamy **przestrzenią euklidesową**. Przestrzeń liniową X nad ciałem \mathbb{C} z iloczynem skalarnym $\langle \cdot, \cdot \rangle$ nazywamy **przestrzenią unitarną**.

Przestrzeń liniową X z iloczynem skalarnym $\langle \cdot, \cdot \rangle : X \times X \rightarrow \mathbb{K}$ zapisujemy jako parę $(X, \langle \cdot, \cdot \rangle)$.

Normy

Norma na przestrzeni z iloczynem skalarnym $(X, \langle \cdot, \cdot \rangle)$, pochodząca od iloczynu skalarnego, to funkcja $\| \cdot \| : X \rightarrow [0, +\infty)$ zadana wzorem

$$\|x\| = \sqrt{\langle x, x \rangle}, \quad x \in X.$$

Twierdzenie (nierówność Schwarza): W przestrzeni z iloczynem skalarnym $(X, \langle \cdot, \cdot \rangle)$, dla dowolnych $x, y \in X$ zachodzi nierówność

$$|\langle x, y \rangle| \leq \|x\| \cdot \|y\|.$$

Niech $(X, \langle \cdot, \cdot \rangle)$ będzie przestrzenią z iloczynem skalarnym, a $\| \cdot \|$ to norma na X pochodząca od iloczynu skalarnego. Wówczas

- dla każdego $x \in X$, $\|x\| = 0$ wtedy i tylko wtedy, gdy $x = 0$,
- dla dowolnych $x \in X$ i $\alpha \in \mathbb{K}$ $\|\alpha x\| = |\alpha| \cdot \|x\|$,
- dla dowolnych $x, y \in X$ $\|x + y\| \leq \|x\| + \|y\|$.

■ Ortogonalność

Powiemy, że wektory $x, y \in \mathbb{K}$ są **ortogonalne** (prostopadłe), jeżeli $\langle x, y \rangle = 0$. Piszemy wówczas $x \perp y$.

Powiemy, że układ wektorów $x_1, \dots, x_k \in X$ jest **układem ortogonalnym**, jeżeli $x_j \neq 0$ dla każdego j oraz $x_i \perp x_j$ dla $i \neq j$.

Ortogonalny układ wektorów nazywamy **ortonormalnym**, jeżeli dodatkowo $\|x_j\| = 1$ dla każdego j .

Przykład 1.

W \mathbb{R}^4 ze standardowym iloczynem skalarnym wektory

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

są ortogonalne, a jeśli każdy z nich przemnożymy przez $\frac{1}{2}$, to stworzą także układ ortonormalny.

Każdy ortogonalny układ wektorów w przestrzeni z iloczynem skalarnym $(X, \langle \cdot, \cdot \rangle)$ jest liniowo niezależny.

Układ ortogonalny (ortonormalny) w przestrzeni z iloczynem skalarnym $(X, \langle \cdot, \cdot \rangle)$, który jest bazą tej przestrzeni, nazywamy **bazą ortogonalną (ortonormalną)**.

W przestrzeni z iloczynem skalarnym $(X, \langle \cdot, \cdot \rangle)$ dana jest podprzestrzeń liniowa Z . Wówczas, dla każdego $x \in X$ istnieją jednoznacznie wyznaczone wektory $z \in Z$ oraz $w \in Z^\perp$ takie, że $x = z + w$. Wektor z nazywamy **rzutem ortogonalnym** wektora x na podprzestrzeń Z .

Zestaw zadań

2.18. X jest przestrzenią euklidesową i wektory $u, v, w \in X$ tworzą układ ortonormalny. Wynika z tego, że

- A. wektory u, v, w są liniowo niezależne
- B. wektory $u + v + w$ i $u - 2v + w$ są ortogonalne
- C. wektory $u + v + w$ i $u - 2v + w$ są ortonormalne

2.19. Niech \mathbb{R}^3 będzie przestrzenią euklidesową ze zwykłym iloczynem skalarnym $(\vec{x}, \vec{y}) = x_1y_1 + x_2y_2 + x_3y_3$ i niech \mathcal{Y} będzie podprzestrzenią wszystkich $\vec{x} \in \mathbb{R}^3$ spełniających układ równań

$$\begin{cases} x_1 - x_2 = 0 \\ x_2 - x_3 = 0 \end{cases}$$

Wynika z tego, że

- A. zbiór wektorów prostopadłych do \mathcal{Y} jest podprzestrzenią o wymiarze 1
- B. istnieje prostopadły do \mathcal{Y} wektor $\vec{z} \neq \vec{0}$, dla którego $z_1 = z_3$
- C. rzutem prostopadłym wektora $[1, 0, -1]^T$ na podprzestrzeń \mathcal{Y} jest wektor zerowy

2.8.

Rozwiązańia

Rozwiązańia

2.1. Niech (G, \diamond, e) będzie grupą, $a, b, c \in G$ i $a \diamond c = e$. Wtedy

- NIE** A. $a \diamond b = b \diamond a$
TAK B. $a \diamond c = c \diamond a$
TAK C. $(b \diamond a) \diamond (c \diamond b) = b \diamond b$

- A. Ten warunek zachodzi tylko, gdy grupa jest abelowa (a tak być nie musi).
B. Zauważmy, że a i c to elementy odwrotne, w związku z czym zapis $a \diamond c = c \diamond a$ jest prawdziwy.
C. Z łączności wynika, że zapis można przekształcić równoważnie:

$$b \diamond (a \diamond c) \diamond b = b \diamond e \diamond b = b \diamond b$$

2.2. Niech $\mathbb{G} = (G, \circ, e)$ będzie grupą. Rozważmy grupę $\mathbb{G}^{op} = (G, \cdot, e)$, gdzie $x \cdot y = y \circ x$ dla $x, y \in G$. Wtedy

- TAK** A. dla funkcji $f : \mathbb{G} \rightarrow \mathbb{G}$ określonej wzorem $f(x) = x^{-1}$ oraz dowolnej podgrupy $\mathbb{H} = (H, \circ, e)$ grupy \mathbb{G} zbiór $f(\mathbb{H})$ jest podgrupą grupy \mathbb{G}^{op}
TAK B. grupy \mathbb{G} i \mathbb{G}^{op} są izomorficzne
TAK C. $\mathbb{G} = \mathbb{G}^{op}$ wtedy i tylko wtedy, gdy grupa \mathbb{G} jest abelowa

- A. Sprawdźmy, czy rzeczywiście $(f(\mathbb{H}), \cdot, e)$ jest grupą. Weźmy $x, y \in H$, wtedy $x^{-1}, y^{-1} \in f(H)$. Mamy

$$x^{-1} \cdot y^{-1} = y^{-1} \circ x^{-1} = (x \circ y)^{-1}, \quad \text{bo } (x \circ y) \circ (y^{-1} \circ x^{-1}) = e.$$

W związku z tym:

- $f(\mathbb{H})$ jest zamknięty na „ \cdot ”, bo $x^{-1} \cdot y^{-1} = (x \circ y)^{-1}$, a $x \circ y \in H$
- łączność i element neutralny – trywialne, wprost z definicji
- element odwrotny zawsze istnieje: $x^{-1} \in H$, bo \mathbb{H} jest grupą, zatem $(x^{-1})^{-1} \in f(H)$

- B. Nietrudno zauważyc, że f jest izomorfizmem. Musi zachodzić warunek, że dla dowolnych $x, y \in G$ jest $f(x \circ y) = f(x) \cdot f(y)$, a to już udowodniliśmy w podpunkcie A.

- C. Żeby $\mathbb{G} = \mathbb{G}^{op}$, dla dowolnych $x, y \in G$ musi być

$$x \circ y = x \cdot y \Leftrightarrow x \circ y = y \circ x,$$

a to jest definicja grupy abelowej.

2.3. Liczba zespolona x_0 jest rozwiązaniem równania $x^2 + x + 1 = 0$. Wynika z tego, że

- TAK** A. liczba x_0^2 też jest rozwiązaniem tego równania
TAK B. liczba $1/x_0$ też jest rozwiązaniem tego równania
TAK C. $x_0^3 = 1$

W zadaniu zastosujemy algebraiczny trik (który warto zapamiętać!): zauważmy, że

$$x^3 - 1 = (x - 1)(x^2 + x + 1),$$

a stąd mamy

$$x^2 + x + 1 = \frac{x^3 - 1}{x - 1}$$

Widzimy więc, że równanie z zadania możemy zapisać w analogicznej postaci:

$$x^2 + x + 1 = 0 \Leftrightarrow \frac{x^3 - 1}{x - 1} = 0$$

Rozwiązaniami są więc wszystkie pierwiastki zespolone stopnia 3 z jedności ($x^3 - 1$ musi być równe 0) z wykluczeniem jedynki (która „wypadła” z dziedziny: $x - 1 \neq 0$). Ze wzoru na pierwiastki zespolone z jedności (lub geometrycznie, szkicując odpowiedni trójkąt równoboczny na okręgu jednostkowym i odczytując współrzędne wierzchołków), otrzymujemy dwa rozwiązania:

$$x_1 = e^{i \cdot 2\pi/3} = \cos\left(\frac{2\pi}{3}\right) + i \sin\left(\frac{2\pi}{3}\right), \quad x_2 = e^{i \cdot 4\pi/3} = \cos\left(\frac{4\pi}{3}\right) + i \sin\left(\frac{4\pi}{3}\right)$$

A. Rozważmy oba przypadki:

$$x_1^2 = (e^{i \cdot 2\pi/3})^2 = e^{i \cdot 4\pi/3} = x_2, \quad x_2^2 = (e^{i \cdot 4\pi/3})^2 = e^{i \cdot 8\pi/3} \stackrel{(*)}{=} e^{i \cdot 2\pi/3} = x_1$$

Równość $(*)$ wynika z okresowości kosinusa. W obu przypadkach kwadrat jednego z rozwiązań także jest rozwiązaniem wyjściowego równania, odpowiedź jest więc prawdziwa.

B. Zauważmy, że liczby x_1 i x_2 są wzajemnie odwrotne:

$$x_1 \cdot x_2 = e^{i \cdot 2\pi/3} \cdot e^{i \cdot 4\pi/3} = e^{i \cdot 2\pi} = 1$$

W związku z tym mamy pewność, że odwrotność dowolnego rozwiązania wyjściowego równania również jest rozwiązaniem tego równania.

C. Sprawdzamy oba przypadki:

$$x_1^3 = (e^{i \cdot 2\pi/3})^3 = e^{i \cdot 6\pi/3} = 1, \quad x_2^3 = (e^{i \cdot 4\pi/3})^3 = e^{i \cdot 12\pi/3} = 1$$

2.4. Strukturę grupy z działaniem mnożenia liczb zespolonych i 1 jako elementem neutralnym ma zbiór

TAK **A.** $\{z \in \mathbb{C} : z^{2019} = 1\}$

NIE **B.** $\{z \in \mathbb{C} : \operatorname{Re}(z) \cdot \operatorname{Im}(z) = 0\}$

TAK **C.** $\{2^k z \in \mathbb{C} : k \in \mathbb{Z}, |z| = 1\}$

A. Zbiór z tego podpunktu to zbiór pierwiastków stopnia 2019 z jedności. Rozważając ich postać zespoloną $z_k = e^{i \cdot 2k\pi/2019}$ nietrudno zauważyc, że jest to grupa: działanie mnożenia nie wyjdzie poza zbiór (mamy $z_k \cdot z_l = z_m$, gdzie $m = k + l \pmod{2019}$), a elementem odwrotnym dla jakiegoś pierwiastka z_k będzie pierwiastek do niego sprzężony $\overline{z_k} = e^{i \cdot (-2k\pi)/2019}$.

B. Zero (należące do zbioru z tego podpunktu) nie ma elementu odwrotnego, nie jest to więc grupa.

C. Rozważany tu zbiór składa się z liczb zespolonych leżących na okręgach o promieniach różnych potęgiem dwójki. Biorąc pod uwagę graficzną interpretację mnożenia liczb zespolonych (moduły się mnożą, a argumenty dodają), nietrudno dostrzec, że po przemnożeniu dwóch liczb ze wspomnianego zbioru wynik również będzie znajdował się na którymś z okręgów, a dla dowolnego $w = 2^k z$ mamy element odwrotny $w^{-1} = \frac{1}{2^k} z^{-1}$ (który na pewno istnieje, bo z leży na okręgu jednostkowym). Mamy do czynienia z grupą.

2.5. Niech X będzie przestrzenią liniową wymiaru 15. Wynika z tego, że

TAK **A.** każdy układ 20 wektorów z X jest liniowo zależny

NIE **B.** każdy układ 10 wektorów z X jest liniowo niezależny

TAK **C.** każda baza X składa się z 15 wektorów

A. Z definicji wymiaru przestrzeni, wymiar to liczba elementów dowolnej bazy. Układ wektorów jest bazą przestrzeni wtedy i tylko wtedy, gdy ten układ jest maksymalnym układem liniowo niezależnym w przestrzeni. Zatem układ z większą liczbą wektorów niż wymiar przestrzeni musi być liniowo zależny.

B. Jest to nieprawda, możemy wybrać z przestrzeni 10 wektorów, które będą liniowo zależne. Wymiar przestrzeni nie mówi o tym, że układ z mniejszą ilością wektorów z tej przestrzeni niż jej wymiar jest niezależny.

C. Jest to prawda na podstawie przytoczonej definicji w podpunkcie A.

2.6. Dana jest przestrzeń X wymiaru 10 oraz podprzestrzenie U, V , takie że $\dim(U) = \dim(V)$ oraz $X = U + V$. Wynika z tego, że

NIE **A.** $\dim(U) = 5$

TAK **B.** $\dim(U) \geq 5$

NIE **C.** $\dim(U \cap V) = 0$

A. Wiemy, że $\dim(U + V) = \dim(U) + \dim(V) - \dim(U \cap V)$ oraz $\dim(X) = \dim(U + V)$. Jeżeli $\dim(U) = 5$, to $\dim(V) = 5$, co zakładałoby, że $\dim(U \cap V) = 0$, co nie musi być prawdą.

B. Gdyby $\dim(U) < 5$, to $\dim(U) + \dim(V) < 10$ i równanie $\dim(U + V) = \dim(U) + \dim(V) - \dim(U \cap V)$ nie mogłoby być spełnione (oczywiście zakładając, że wymiar nie może być ujemny).

C. Informacje z zadania w żaden sposób nie udowadniają, że $\dim(U \cap V) = 0$.

2.7. Dane są elementy przestrzeni liniowej nad ciałem liczb rzeczywistych $\vec{a}, \vec{b}, \vec{c}$. Prawdą jest, że

NIE **A.** jeżeli $(\vec{a}, \vec{b}, \vec{c})$ jest niezależny liniowo, to $(\vec{a} - \vec{b}, \vec{b} - \vec{c}, \vec{c} - \vec{a})$ jest niezależny liniowo

TAK **B.** jeżeli $(\vec{a}, \vec{b}, \vec{c})$ jest niezależny liniowo, to $(\vec{a} + \vec{b}, \vec{b} + \vec{c}, \vec{c} + \vec{a})$ jest niezależny liniowo

TAK **C.** jeżeli $(\vec{a} + \vec{b}, \vec{b} + \vec{c}, \vec{c} + \vec{a})$ jest niezależny liniowo, to $(\vec{a}, \vec{b}, \vec{c})$ jest niezależny liniowo

A. Zauważmy, że $(\vec{a} - \vec{b}) + (\vec{b} - \vec{c}) + (\vec{c} - \vec{a}) = 0$, więc układ jest liniowo zależny.

B. Weźmy α, β, γ , takie że $\alpha(\vec{a} + \vec{b}) + \beta(\vec{b} + \vec{c}) + \gamma(\vec{c} + \vec{a}) = 0$. Przekształcając równanie, dostajemy $(\gamma + \alpha)\vec{a} + (\alpha + \beta)\vec{b} + (\beta + \gamma)\vec{c} = 0$. Skoro układ $(\vec{a}, \vec{b}, \vec{c})$ jest liniowo niezależny, to $\alpha + \beta = \beta + \gamma = \gamma + \alpha = 0$, co prowadzi do $\alpha = \beta = \gamma = 0$, więc dany układ jest liniowo niezależny.

C. Widzimy, że za pomocą wektorów $\vec{a}, \vec{b}, \vec{c}$ w prosty sposób otrzymamy każdy z trzech wektorów z układu $(\vec{a} + \vec{b}, \vec{b} + \vec{c}, \vec{c} + \vec{a})$, który jest liniowo niezależny, więc układ $\vec{a}, \vec{b}, \vec{c}$ również musi być.

2.8. W przestrzeni liniowej $\mathbb{R}[x]_{<3}$ wielomianów rzeczywistych stopnia mniejszego niż 3 dane są wielomiany

$$p_1(t) = 1, \quad p_2(t) = 1 + t^2, \quad p_3(t) = 1 + t + t^2.$$

Wynika z tego, że

TAK **A.** układ (p_1, p_2, p_3) stanowi bazę przestrzeni $\mathbb{R}[x]_{<3}$

TAK **B.** jeśli $q(t)$ jest wielomianem stopnia pierwszego, to układ (p_1, p_2, q) jest liniowo niezależny

TAK **C.** macierz $A = (p_i(j))_{i,j=1}^3$ jest nieosobliwa

A. Nietrudno zauważyc, że wielomiany te są liniowo niezależne, więc tworzą bazę $\mathbb{R}[x]_{<3}$.

B. Ten układ to $(1, 1 + t^2, at + b)$, więc rzeczywiście jest liniowo niezależny (oczywiście $a \neq 0$).

C. Rozpisujemy macierz i schodkujemy:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 5 & 10 \\ 3 & 7 & 13 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 0 & 3 & 8 \\ 0 & 4 & 10 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 0 & 3 & 8 \\ 0 & 0 & -\frac{2}{3} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 0 & 3 & 8 \\ 0 & 0 & 1 \end{bmatrix}$$

Widzimy, że macierz po przekształceniach jest nieosobliwa.

2.9. Niech $M = \{(a_{ij})_{i,j=1}^3 \in \mathbb{R}^{3,3} : a_{11} + a_{22} + a_{33} = 0\}$. Prawdą jest, że

- NIE** A. zbiór M jest zamknięty na działanie mnożenia macierzy
- TAK** B. zbiór M jest podprzestrzenią liniową w $\mathbb{R}^{3,3}$
- TAK** C. istnieje niezerowa podprzestrzeń liniowa N przestrzeni $\mathbb{R}^{3,3}$, taka że $M \cap N = \{0\}$, gdzie 0 oznacza macierz zerową

A. Nie, np. dla $A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -3 \end{bmatrix}$, gdzie $A \in M$ mamy $A \cdot A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 9 \end{bmatrix}$, czyli $A \cdot A \notin M$, sprzeczność.

B. Sprawdzamy warunki z definicji podprzestrzeni dla przestrzeni liniowej $\mathbb{R}^{3,3}$. W oczywisty sposób dla $A, B \in M$, $\alpha \in \mathbb{K}$ zachodzi $A + B \in M$ oraz $\alpha \cdot A \in M$, zatem zbiór M jest podprzestrzenią liniową w $\mathbb{R}^{3,3}$.

C. Zauważmy, że jeśli $A \in M$, to znając A_{11} oraz A_{22} , wiemy, jaka będzie wartość A_{33} . Nie mamy żadnych informacji o pozostałych elementach A , więc $\dim M = 8$. Skoro $\dim \mathbb{R}^{3,3} = 9$, to istnieje opisane N (i $\dim N = 1$).

2.10. Macierz rzeczywista A ma n wierszy i k kolumn. Wynika z tego, że

- TAK** A. jeśli wymiar jądra macierzy A wynosi n , to $k \geq n$
- TAK** B. jeśli rząd macierzy A jest równy n , to $k \geq n$
- TAK** C. jeśli rząd macierzy A jest równy k , to macierz $A^T A$ jest nieosobliwa
- A. Z równania $\text{rank } A + \dim(\ker A) = k$ wprost wynika, że $k \geq n$.
- B. Analogicznie jak w A.
- C. Skoro A ma rząd równy k , to jest on maksymalny możliwy i jednocześnie $\text{rank } A^T = k$ także. Przypuśćmy, że $\ker(A^T A) \neq \{\vec{0}\}$. Wtedy istnieje wektor niezerowy \vec{v} , dla którego zachodzi $A^T A \vec{v} = \vec{0}$. Możemy pomnożyć lewostronnie przez \vec{v}^T . Mamy $(\vec{v}^T A^T) A \vec{v} = (\vec{v}^T A) \vec{v} = 0$, oznaczmy wektor $y = A \vec{v}$. Otrzymujemy $y^T y = 0$. Ponieważ A ma maksymalny rząd, to w jej jadrze jest tylko wektor zerowy. Dlatego $y = A \vec{v} = \vec{0} \iff \vec{v} = 0$. Sprzeczność.

2.11. Macierz A jest wymiarem 10×10 oraz $\det(A) = 5$. Oznacza to, że

- TAK** A. kolumny A są liniowo niezależne
- TAK** B. A jest odwracalna
- NIE** C. $\det(A^{-1}) = -5$

Ponieważ macierz ma niezerowy wyznacznik, to znaczy, że jest osobliwa (rozwiązuje to podpunkt B.). Z warunków równoważnych osobliwości wnioskujemy także, że podpunkt A. jest prawdziwy.

Fałszywość podpunktu C. łatwo uzyskujemy ze wzoru na wyznacznik macierzy odwrotnej:

$$\det(A^{-1}) = \frac{1}{\det(A)} = \frac{1}{5}$$

2.12. W macierzy kwadratowej A o wymiarach $n \times n$ dokładnie n elementów jest jedynkami, a pozostałe elementy są zerami. Wynika z tego, że

- TAK** A. $\det(A) \in \{-1, 0, 1\}$
- TAK** B. jeśli A jest nieosobliwa, to jest macierzą permutacji
- TAK** C. jeśli którakolwiek jedynkę w macierzy A zastąpimy zerem, to wyznacznik powstałej macierzy będzie równy zero

Zauważmy, że jeśli którakolwiek z kolumn zawiera same zera, to wyznacznik macierzy będzie równy 0 ($\text{rank}(A) < n$), co rozwiązuje podpunkt C. Z tego wynika, że każda nieosobliwa macierz z treści zadania ma dokładnie jedną jedynkę w każdym wierszu i kolumnie, więc jest macierzą permutacji (B.). W podpunkcie A. pozostało policzyć wyznacznik A , korzystając z rozwinięcia Laplace'a względem dowolnej kolumny/wiersza – wynik zawsze będzie ze zbioru $\{-1, 0, 1\}$.

2.13. Niech X, Y będą przestrzeniami liniowymi, a $f : X \rightarrow Y$ przekształceniem liniowym. Prawdą jest, że

- TAK** A. jeśli $\ker f = 0$, to $\dim X \leq \dim Y$
- NIE** B. jeśli $\ker f \neq 0$, to $\dim X > \dim Y$
- TAK** C. jeśli $\dim X > \dim Y$, to $\ker f \neq 0$

- A. Ponieważ f ma jądro równe 0, to jest monomorfizmem, czyli przekształceniem różnowartościowym. W takim razie z pewnością $\dim Y \geq \dim X$.
- B. Macierz $A \in \mathbb{K}^{m,n}$ zadaje przekształcenie liniowe $f \in L(\mathbb{K}^n, \mathbb{K}^m)$. Jeśli jądro jest niezerowe, niekoniecznie musi być tak, że $n > m$.
- C. Jeśli dziedzina jest liczniejsza (tutaj – ma więcej wymiarów) od przeciwdziedziny, przekształcenie nie może być różnowartościowe.

2.14. Dane są przestrzenie liniowe $X \neq X'$, podprzestrzenie $U, V \subseteq X$ oraz podprzestrzenie $U', V' \subseteq X'$, takie że $X = U \oplus V$ oraz $X' = U' \oplus V'$. Prawdą jest, że

- TAK** A. jeżeli $g : U \rightarrow U'$, $h : V \rightarrow V'$ są przekształceniemi liniowymi, to istnieje dokładnie jedno przekształcenie liniowe $f : X \rightarrow X'$, takie że $f|_U = g$ i $f|_V = h$
- NIE** B. jeżeli X i X' są izomorficzne, to U, U' są izomorficzne lub U, V' są izomorficzne
- TAK** C. jeżeli U jest izomorficzne z U' oraz V izomorficzne z V' , to X jest izomorficzne z X'

A. Są trzy podejścia do rozwiązania tego problemu:

1. Intuicja z docsa z odpowiedziami – „będzie jak ****”.
2. Machanie rękami – g mapuje nam U na U' , a h mapuje V na V' . Jakie może być przekształcenie $f : X \rightarrow X'$ o zadanych własnościach? Oczywiście jedno i będzie ono działać tak: wiedząc, że $X = U \oplus V$, to każdy wektor z x należy do U , albo V . W zależności od tego, z której podprzestrzeni bierzemy x , używamy g , albo h do zmapowania go na odpowiednią podprzestrzeń. Innymi słowy, obcinając dziedzinę f do wektorów z podprzestrzeni U działamy funkcją g i analogicznie h dla V , czyli $f|_U = g$ i $f|_V = h$.
3. Dowód formalny – zostawiam jako ćwiczenie dla Czytelnika.

B. Wcale nie musi tak być, założmy, że $\dim X = 4$ i $\dim U = 1$, $\dim V = 3$, $\dim U' = \dim V' = 2$. Wtedy U nie ma jak być izomorficzne zarówno z U' , jak i z V' .

C. Mamy $X = U \oplus V$ i $X' = U' \oplus V'$. Skoro U i U' są izomorficzne, podobnie V i V' , to ich sumy proste również muszą być izomorficzne, więc X i X' są izomorficzne.

2.15. Przekształcenie $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ dane jest wzorem $f \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{bmatrix} 2x_1 - x_2 \\ x_1 + x_2 \end{bmatrix}$. Wynika z tego, że

- NIE** A. w pewnej bazie przestrzeni \mathbb{R}^2 macierzą przekształcenia f jest $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
- TAK** B. f jest przekształceniem różnowartościowym i „na”
- NIE** C. macierzą przekształcenia f w bazie $([1, 0]^T, [1, 1]^T)$ jest $\begin{bmatrix} 1 & 1 \\ -1 & 2 \end{bmatrix}$

A. Gdyby tak było, to przekształcenie to nie zmieniałoby wektorów z bazy (macierz jest identycznościowa), a istotnie je zmienia.

B. Weźmy $\vec{x} = [x_1, x_2]^T$ i $\vec{y} = [y_1, y_2]^T$. Mamy $f(\vec{x}) = [2x_1 - x_2, x_1 + x_2]^T$ i $f(\vec{y}) = [2y_1 - y_2, y_1 + y_2]^T$. Założmy, że $f(\vec{x}) = f(\vec{y})$. Wtedy dostajemy układ równań

$$\begin{cases} 2x_1 - x_2 = 2y_1 - y_2 \\ x_1 + x_2 = y_1 + y_2 \end{cases}$$

z którego wynika, że $x_1 = y_1$ oraz $x_2 = y_2$, więc $\vec{x} = \vec{y}$, czyli przekształcenie jest różnowartościowe. Niech teraz $\vec{y} = [y_1, y_2]$ będzie dowolnym wektorem z przeciwdziedziny. Wtedy nietrudno pokazać, że powstaje on jako wynik $f([\frac{y_1+y_2}{3}, \frac{-y_1+2y_2}{3}]^T)$, więc przekształcenie jest również „na”.

- C. Jest $f([1, 0]^T) = [2, 1]^T = 1 \cdot [1, 0]^T + 1 \cdot [1, 1]^T$ oraz $f([1, 1]^T) = [1, 2]^T = -1 \cdot [1, 0]^T + 2 \cdot [1, 1]^T$, więc macierzą przekształcenia f w tej bazie jest macierz $\begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix}$.

- 2.16.** W przestrzeni $\mathbb{R}[x]_{\leq 1}$ wielomianów rzeczywistych stopnia co najwyżej 1 funkcjonały φ_1 i φ_2 dane są wzorami

$$\varphi_1(p) = p(0), \quad \varphi_2(p) = p(1),$$

gdzie $p \in \mathbb{R}[x]_{\leq 1}$. Wynika z tego, że

- TAK** A. funkcjonały φ_1 i φ_2 są liniowo niezależne
TAK B. funkcjonał φ dany wzorem $\varphi(p) = p'(\frac{1}{2})$ jest pewną kombinacją liniową φ_1 i φ_2
TAK C. wielomiany $p_1(t) = 1 - t$ i $p_2(t) = t$ oraz funkcjonały φ_1 i φ_2 stanowią wzajemnie do siebie sprzężone bazy odpowiednio w $\mathbb{R}[x]_{\leq 1}$ oraz przestrzeni sprzężonej $(\mathbb{R}[x]_{\leq 1})^*$

- A. Jeśli $p(x) = ax+b$, to $\varphi_1(p) = b$, $\varphi_2(p) = a+b$, więc widzimy, że rzeczywiście są liniowo niezależne.
B. Jest $p'(x) = a$, więc $\varphi(p) = p'(\frac{1}{2}) = a = -1 \cdot \varphi_1(p) + 1 \cdot \varphi_2(p)$.
C. Dla ułatwienia zapisu zastosujemy nic nie wnoszące podstawienie $t \mapsto x$. Z definicji bazy dualnej: chcemy sprawdzić, czy funkcjonały φ_1, φ_2 opisują współczynniki kombinacji liniowej dowolnego wielomianu z $\mathbb{R}[x]_2$ (postaci $ax+b$) w bazie $(1-x, x)$:

$$\varphi_1(ax+b) \cdot (1-x) + \varphi_2(ax+b) \cdot x = b(1-x) + (a+b)x = ax+b$$

Widzimy, że w wyniku otrzymaliśmy wejściowy wielomian $ax+b$, więc odpowiedź jest prawdziwa.

- 2.17.** Niech $A \in \mathbb{C}^{2n, 2n}$, gdzie $n \in \mathbb{N}$ oraz $n > 0$, będzie macierzą nieosobliwą. Wynika z tego, że

- ???** A. liczba różnych wartości własne macierzy A jest parzysta
??? B. w zbiorze wartości własne macierzy A nie ma zera
??? C. jeśli λ jest wartością własną macierzy A^{-1} , to λ^{-1} jest wartością własną macierzy A

przyp. red.: TODO

- 2.18.** X jest przestrzenią euklidesową i wektory $u, v, w \in X$ tworzą układ ortonormalny. Wynika z tego, że

- TAK** A. wektory u, v, w są liniowo niezależne
TAK B. wektory $u+v+w$ i $u-2v+w$ są ortogonalne
NIE C. wektory $u+v+w$ i $u-2v+w$ są ortonormalne
- A. Skoro wektory tworzą układ ortonormalny (w szczególności ortogonalny), to są liniowo niezależne.
B. Policzmy $\langle u+v+w, u-2v+w \rangle = \langle u, u \rangle - 2\langle u, v \rangle + \langle u, w \rangle + \langle v, u \rangle - 2\langle v, v \rangle + \langle v, w \rangle + \langle w, u \rangle - 2\langle w, v \rangle + \langle w, w \rangle = *$. Skoro u, v, w tworzą układ ortonormalny, to $\langle u, u \rangle = \langle v, v \rangle = \langle w, w \rangle = 1$ i $\langle x, y \rangle = 0$ wpp. Otrzymujemy $* = 1 - 2 + 1 = 0$, a więc wektory są ortogonalne.
C. Policzmy $\|u+v+w\| = \sqrt{\langle u+v+w, u+v+w \rangle} = \sqrt{\langle u, u \rangle + \langle u, v \rangle + \langle u, w \rangle + \langle v, u \rangle + \langle v, v \rangle + \langle v, w \rangle + \langle w, u \rangle + \langle w, v \rangle + \langle w, w \rangle} = \sqrt{\langle u, u \rangle + \langle v, v \rangle + \langle w, w \rangle} = \sqrt{3} \neq 1$, a więc wektory nie są ortonormalne.

- 2.19.** Niech \mathbb{R}^3 będzie przestrzenią euklidesową ze zwykłym iloczynem skalarnym $(\vec{x}, \vec{y}) = x_1y_1 + x_2y_2 + x_3y_3$ i niech \mathcal{Y} będzie podprzestrzenią wszystkich $\vec{x} \in \mathbb{R}^3$ spełniających układ równań

$$\begin{cases} x_1 - x_2 = 0 \\ x_2 - x_3 = 0 \end{cases}$$

Wynika z tego, że

NIE A. zbiór wektorów prostopadłych do \mathcal{Y} jest podprzestrzenią o wymiarze 1

TAK B. istnieje prostopadły do \mathcal{Y} wektor $\vec{z} \neq \vec{0}$, dla którego $z_1 = z_3$

TAK C. rzutem prostopadłym wektora $[1, 0, -1]^T$ na podprzestrzeń \mathcal{Y} jest wektor zerowy

Zauważmy, że $\mathcal{Y} = \text{span}([1, 1, 1]^T)$.

A. Weźmy wektor $x \in \mathcal{Y} = [\alpha, \alpha, \alpha]$. Wektory prostopadłe do x to takie $y = [y_1, y_2, y_3]^T$, że $\alpha y_1 + \alpha y_2 + \alpha y_3 = \alpha(y_1 + y_2 + y_3) = 0$. Wynika stąd, że $y_3 = -(y_1 + y_2)$, czyli $y = [y_1, y_2, -(y_1 + y_2)]^T$, a więc zbiór takich wektorów y jest podprzestrzenią o wymiarze 2.

B. Oczywiście, że istnieje. Jak wiemy z poprzedniego podpunktu, taki wektor musi być postaci $z = [z_1, -(z_1 + z_3), z_3]^T$, więc np. $z = [1, -2, 1]$ jest prostopadły do \mathcal{Y} i nierówny $\vec{0}$.

C. Tak, wystarczy zauważyć, że wektory $[1, 0, -1]^T$ i $[\alpha, \alpha, \alpha]^T \in \mathcal{Y}$ są do siebie prostopadłe (co jasne, niezależnie od α).

3

Podstawy matematyki

Materiały teoretyczne z tego przedmiotu zostały opracowane na podstawie skryptu Pawła Urzyczyna.

Podstawa programowa

1. Działania na **zbiorach**.
2. **Funkcje** i ich własności.
3. **Relacje równoważności** i ich własności.
4. **Moce zbiorów**.
5. **Porządki częściowe** i ich własności.
6. **Dobre ufundowanie** i indukcja.
7. **Rachunek zdań** – semantyka i naturalna dedukcja.
8. Przykłady opisu własności struktur matematycznych w **logice pierwszego rzędu**.

3.1. Działania na zbiorach

Zbiór złożony dokładnie z tych elementów typu \mathcal{D} , które spełniają warunek $K(x)$, oznaczamy przez

$$\{x : \mathcal{D} \mid K(x)\}$$

Zbiory można też definiować przez *zastępowanie*, czyli:

$$\{f(x) \mid x \in \mathcal{D}\}$$

Dwa zbiory $A, B : \mathcal{P}(\mathcal{D})$ są równe wtedy i tylko wtedy, gdy $A \subseteq B$ oraz $B \subseteq A$.

Niech $A, B : \mathcal{P}(\mathcal{D})$, gdzie $\mathcal{P}(A)$ to zbiór wszystkich podzbiorów A (**zbiór potęgowy**).

- **Sumą** zbiorów A i B nazywamy zbiór

$$A \cup B = \{x : \mathcal{D} \mid x \in A \vee x \in B\}$$

- **Iloczyn** lub **przecięcie** zbiorów A i B to zbiór

$$A \cap B = \{x : \mathcal{D} \mid x \in A \wedge x \in B\}$$

- **Różnicą** zbiorów A i B nazywamy zbiór

$$A - B = \{x : \mathcal{D} \mid x \in A \wedge x \notin B\}$$

- **Dopełnienie** zbioru A (do typu \mathcal{D}) to zbiór

$$-A = \{x : \mathcal{D} \mid x \notin A\}$$

- **Różnica symetryczna** zbiorów A i B to zbiór

$$A \dot{-} B = (A - B) \cup (B - A)$$

Dla odróżnienia od sumy prostej (patrz niżej), wspomnianą „zwykłą” sumę nazywamy czasem sumą teoriom-nogościową.

Działania nieskończone

Pojęcie sumy i iloczynu można uogólnić. Przypuśćmy, że mamy rodzinę zbiorów $\mathcal{R} \subseteq \mathcal{P}(\mathcal{D})$. Wtedy **sumą uogólnioną** rodzinę \mathcal{R} nazywamy zbiór

$$\bigcup \mathcal{R} = \{x : \mathcal{D} \mid \exists A(x \in A \wedge A \in \mathcal{R})\}$$

Jeśli \mathcal{R} jest rodziną niepustą ($\mathcal{R} \neq \emptyset$) to określamy **uogólniony iloczyn** rodzinę \mathcal{R} :

$$\bigcap \mathcal{R} = \{x : \mathcal{D} \mid \forall A(A \in \mathcal{R} \Rightarrow x \in A)\}$$

Produkty, sumy proste

Iloczyn kartezjański zbiorów A i B to zbiór oznaczany przez $A \times B$, który składa się z par uporządkowanych postaci $\langle a, b \rangle$, gdzie $a \in A$ oraz $b \in B$. Para uporządkowana $\langle a, b \rangle$ to abstrakcyjny obiekt zadany przez wybór pierwszej współrzędnej a i drugiej współrzędnej b . Inaczej mówiąc, dwie pary uważały za równe, gdy ich odpowiednie współrzędne są takie same.

Pojęcie produktu można uogólnić na trzy i więcej wymiarów. Można zdefiniować produkt $A \times B \times C$ jako $(A \times B) \times C$, przyjmując, że trójką uporządkowane $\langle a, b, c \rangle$ to para $\langle \langle a, b \rangle, c \rangle$, i tak dalej.

Suma prosta zbiorów A i B , którą oznaczamy przez $A \oplus B$, zwana jest też koproduktem lub sumą rozłączną. Każdy element sumy prostej $A \oplus B$ jest

- albo postaci $\langle a \rangle_1$, gdzie $a \in A$
- albo postaci $\langle b \rangle_2$, gdzie $b \in B$

Innymi słowy, jest to suma zbiorów, która przy okazji oznacza każdy element, z którego zbiorem on pochodził. Oznacza to, że jest to suma pozwalająca na powtórzenia. Przyjmujemy przy tym, że $\langle x \rangle_i = \langle y \rangle_j$ wtedy i tylko wtedy, gdy $x = y$ oraz $i = j$.

Zestaw zadań

- 3.1. Dana jest rodzina zbiorów A oraz zbiory X, Y takie, że $X \in A$. Prawdą jest, że

- A. $(Y \subseteq X) \Rightarrow (Y \subseteq \bigcup A)$
- B. $\bigcap A \subseteq \bigcup A$
- C. $(Y \subseteq X) \Rightarrow (\bigcap A \subseteq Y)$

3.2.

Funkcje i ich własności

O **funkcji** mówimy wtedy, gdy każdemu elementowi $x \in A$ potrafimy jednoznacznie przypisać pewien element $f(x) \in B$. Zapisujemy to jako

$$f : A \rightarrow B$$

Zbiór wszystkich funkcji oznaczamy przez $A \rightarrow B$ albo B^A . **Funkcja częściowa** nie zawsze musi mieć określona wartość. Zbiór $\text{Dom}(f) = \{x \in A \mid f(x) \text{ jest określone}\}$ nazywamy **dziedziną funkcji**.

Dwie funkcje $f, g : A \rightarrow B$ są sobie równe wtedy i tylko wtedy, gdy dla każdego $x \in A$ zachodzi $f(x) = g(x)$.

Wykresem funkcji nazywamy zbiór $\mathcal{W}(f) = \{(x, y) \mid f(x) = y\} \subseteq A \times B$.

Zbiorem wartości funkcji $f : A \rightarrow B$ nazywamy zbiór

$$\text{Rg}(f) = \{y \in B \mid \exists x \in A. f(x) = y\}$$

■ Injekcje, surjekcje, bijekcje

Funkcja $f : A \rightarrow B$ jest **różnowartościowa** wtedy i tylko wtedy, gdy dla każdej pary $x, y \in A$, takiej że $x \neq y$, zachodzi $f(x) \neq f(y)$.

Funkcja $f : A \rightarrow B$ jest „na” wtedy i tylko wtedy, gdy dla każdego $y \in B$ istnieje odpowiadający mu $x \in A$, taki że $f(x) = y$.

Funkcję różnowartościową nazywamy też **injekcją**, funkcję „na” – **surjekcją**, a funkcję, która jest różnowartościowa i „na” – **bijekcją**.

■ Odwracanie i składanie funkcji

Jeśli $f : A \rightarrow B$ jest injekcją, to możemy określić funkcję częściową f^{-1} będącą **odwrotnością funkcji**, taką że dla każdego $x \in A$ jest $f^{-1}(f(x)) = x$.

Niech $f : A \rightarrow B$ oraz $g : B \rightarrow C$. **Złożeniem funkcji** f i g nazywamy funkcję $g \circ f$, określoną równaniem $(g \circ f)(x) = g(f(x))$ dla każdego $x \in A$.

Wybrane własności:

- Jeśli $f : A \rightarrow B, g : B \rightarrow C, h : C \rightarrow D$, to $h \circ (g \circ f) = (h \circ g) \circ f$.
- Jeśli $f : A \rightarrow B$ jest bijekcją, to $f^{-1} \circ f = id_A$ oraz $f \circ f^{-1} = id_B$.
- Jeśli $f : A \rightarrow B$, to $f \circ id_A = f = id_B \circ f$.
- Jeśli $f : A \rightarrow B, g : B \rightarrow C$ są injekcjami, to $g \circ f$ też jest.
- Jeśli $f : A \rightarrow B, g : B \rightarrow C$ są surjekcjami, to $g \circ f$ też jest.

■ Obrazy i przeciwbrazy

Niech $f : A \rightarrow B$. **Obraz** zbioru $C \subseteq A$ przy przekształceniu f to zbiór

$$f(C) = \{f(a) \mid a \in C\}$$

Przeciwbrazem zbioru $D \subseteq B$ przekształceniu f nazwiemy zbiór

$$f^{-1}(D) = \{a \in A \mid a \in \text{Dom}(f) \wedge f(a) \in D\}$$

Przykład 1.

Niech $f : \mathbb{N} \rightarrow \mathbb{N}$ taka, że $f(n) = 2n$. Wtedy obraz zbioru $\{2, 3, 4\}$ przy przekształceniu f to $f(\{2, 3, 4\}) = \{4, 6, 8\}$, zaś przeciwbraz do $f^{-1}(\{2, 3, 4\}) = \{1, 2\}$.

Zestaw zadań

- 3.2.** Dane są trzy funkcje $f : A \rightarrow B, g : B \rightarrow C$ i $h : C \rightarrow D$, których złożenie $h \circ g \circ f : A \rightarrow D$ jest bijekcją. Wynika z tego, że

- A. f jest funkcją różnowartościową (injekcją)
- B. g jest bijekcją
- C. h jest na D (surjekcją)

3.3. Dana jest funkcja $f : \mathbb{N} \rightarrow \mathbb{N}$. Niech f^{2019} będzie 2019-krotnym złożeniem funkcji f . Prawdą jest, że

- A. f jest injekcją $\Leftrightarrow f^{2019}$ jest injekcją
- B. f jest surjekcją $\Leftrightarrow f^{2019}$ jest surjekcją
- C. $f(42) = 42 \Leftrightarrow f^{2019}(42) = 42$

3.4. Jeśli $f : A \rightarrow B$ jest różnowartościowa, to funkcja obrazu $f^\rightarrow : \mathcal{P}(A) \rightarrow \mathcal{P}(B)$

- A. jest różnowartościowa
- B. jest funkcją odwrotną do funkcji przeciwbrazu $f^\leftarrow : \mathcal{P}(B) \rightarrow \mathcal{P}(A)$
- C. spełnia warunek $\bigcup\{f^\rightarrow(Z) : Z \in \mathcal{L}\} = f^\rightarrow(\bigcup \mathcal{L})$ dla dowolnej rodziny $\mathcal{L} \subseteq \mathcal{P}(A)$

3.5. Niech f będzie funkcją ze zbioru A w zbiór B . Wynika z tego, że

- A. obraz zbioru A przy funkcji f to zbiór B
- B. przeciwbraz zbioru B przy funkcji f to zbiór A
- C. jeśli X jest niepustym podzbiorem B , to przeciwbraz zbioru X przy funkcji f jest niepusty

3.3.

Relacje równoważności i ich własności

Relacja dwuargumentowa to zbiór wszystkich uporządkowanych par tych przedmiotów, pomiędzy którymi relacja zachodzi. Jeśli dwa elementy a, b są ze sobą w relacji r , to zapiszemy to jako $a \, r \, b$. Pewne własności relacji dwuargumentowych mają swoje nazwy:

- **zwrotna** – $\forall_{x \in A} x \, r \, x$
- **symetryczna** – $\forall_{x,y \in A} x \, r \, y \Rightarrow y \, r \, x$
- **przechodnia** – $\forall_{x,y,z \in A} x \, r \, y \wedge y \, r \, z \Rightarrow x \, r \, z$
- **antysymetryczna** – $\forall_{x,y \in A} x \, r \, y \wedge y \, r \, x \Rightarrow x = y$
- **spójna** – $\forall_{x,y \in A} x \, r \, y \vee y \, r \, x$ – innymi słowy, każde dwa elementy są porównywalne

Należy pamiętać, że relacje to nadal zwyczajne zbiory, w związku z tym pojęcia takie jak suma relacji, iloczyn relacji itp. to zupełnie zwyczajne sumy i iloczyny.

■ Relacje równoważności

Relacja równoważności to dwuargumentowa relacja r na zbiorze A , która jest zwrotna, symetryczna i przechodnia. Intuicyjnie możemy więc utożsamiać relację równoważności z podziałem zbioru A na klasy, których elementy mają pewną cechę wspólną.

Klasą abstrakcji relacji równoważności r w zbiorze A , wyznaczoną przez element $x \in A$, nazywamy zbiór $[x]_r = \{y \in A \mid x \, r \, y\}$.

Równoważne są warunki:

1. $x \, r \, y$

2. $x \in [y]_r$
3. $y \in [x]_r$
4. $[x]_r = [y]_r$
5. $[x]_r \cap [y]_r \neq \emptyset$

Zbiór ilorazowy relacji r w zbiorze A jest zdefiniowany jako zbiór jej klas abstrakcji i oznaczany jest poprzez A/r .

Przykład 1.

Przykładem relacji równoważności jest relacja przystawania modulo 7 w zbiorze liczb naturalnych. Taka relacja ma siedem klas abstrakcji. Jeśli $x, y \in \mathbb{N}$ należą do różnych klas abstrakcji, to dają one inne reszty z dzielenia przez 7.

Przykład 2.

Szczególnym przykładem relacji równoważności jest relacja identycznościowa $1_A = \{\langle x, x \rangle \mid x \in A\}$. Jej klasami abstrakcji są wszystkie singletony.

Podział zbioru A to rodzina $P \subseteq \mathcal{P}(A)$, która spełnia warunki:

- $\forall_{p \in P} p \neq \emptyset$
- $\forall_{p,q \in P} p = q \vee p \cap q = \emptyset$
- $\bigcup P = A$

Bardziej intuicyjnie, podział zbioru A dzieli go na rozłączne części, a każdy element $x \in A$ należy do pewnej części (czyli nie ma elementów nieprzypadlonych).

Lemat 1. Jeżeli r jest relacją równoważności w A to A/r jest podziałem zbioru A .

Lemat 2. Jeżeli P jest podziałem zbioru A , to istnieje taka relacja równoważności r w A , że $P = A/r$.

Powysze lematy mówią, że relacje równoważności i podziały zbioru to w istocie to samo. Jedno determinuje drugie i na odwrót.

Aksjomat wyboru

Niech r będzie częściową relacją równoważności w typie D . Rozważmy funkcję wyboru

$$\sigma : D/r \rightarrow D \quad \text{taka, że} \quad \forall_{a \in D} \sigma([a]_r) \in [a]_r$$

Zatem funkcja wyboru dla każdego typu ilorazowego powinna zwrócić pewnego reprezentanta tego typu. Oczywiście takich funkcji może być wiele. Założenie, że taka funkcja istnieje nazywamy **aksjomatem (pewnym) wyboru**. To założenie nie jest wcale oczywiste, bo nie zawsze jest możliwe określenie konkretnej funkcji wyboru.

Własność $\sigma(K) \in K$ dla $K \in \text{Dom}(\sigma)$ sugeruje następujące uogólnienie pojęcia funkcji wyboru: jeśli R jest rodziną podzbiorów D (niekoniecznie rozłączną), to funkcję wyboru dla R nazywamy dowolną funkcją $f : R \rightarrow D$ spełniającą dla wszystkich $A \in R$ warunek $f(A) \in A$. Intuicyjnie, podobnie jak powyżej, dla każdego zbioru wybieramy jego reprezentanta.

Zestaw zadań

3.6. Niech A będzie dowolnym zbiorem i niech $s, r \subseteq A \times A$ będą relacjami. Jeśli s i r są

- A. zwrotne, to $s \cup r$ jest relacją zwrotną

- B. symetryczne, to $s \cup r$ jest relacją symetryczną
 C. przechodnie, to $s \cup r$ jest relacją przechodnią
- 3.7.** Niech $f : A \rightarrow B$ będzie funkcją „na” B i niech s_A będzie relacją równoważności na A . Przez $f^{-1}(X)$ oznaczamy przeciobraz X przy f . Następująca relacja r jest relacją równoważności na B :
- A. $b \sim b'$ wtedy i tylko wtedy, gdy $f^{-1}(\{b\}) \cup f^{-1}(\{b'\})$ jest pewną klasą abstrakcji relacji s_A
 B. $b \sim b'$ wtedy i tylko wtedy, gdy istnieją $a, a' \in A$ takie, że $a \in f^{-1}(\{b\})$ i $a' \in f^{-1}(\{b'\})$ oraz $a s_A a'$
 C. $b \sim b'$ wtedy i tylko wtedy, gdy dla każdych $a, a' \in A$ takich, że $a \in f^{-1}(\{b\})$ i $a' \in f^{-1}(\{b'\})$, zachodzi $a s_A a'$
- 3.8.** r jest relacją równoważności na liczbach naturalnych dodatnich określona w następujący sposób: liczby x i y są w relacji r wtedy i tylko wtedy, gdy zbiory dzielników pierwszych liczb x i y są takie same. Wynika z tego, że
- A. wszystkie klasy abstrakcji relacji r są nieskończone
 B. wszystkie klasy abstrakcji relacji r są równoliczne
 C. zbiór ilorazowy relacji r jest skończony
- 3.9.** W zbiorze 5-elementowym
- A. każda relacja przechodnia ma moc co najmniej 3
 B. każda relacja przechodnia ma moc co najwyżej 25
 C. istnieje relacja przechodnia o mocy równej 24
- 3.10.** Niech r będzie relacją równoważności na niepustym zbiorze A . Wynika z tego, że
- A. każda klasa abstrakcji relacji r jest niepusta
 B. dowolne dwie różne klasy abstrakcji relacji r są rozłączne
 C. suma zbioru klas abstrakcji relacji r jest równa A

3.4.

Moce zbiorów

Mówimy, że zbiory A i B są **równoliczne**, albo że są to zbiory **tej samej mocy** (i piszemy $A \sim B$) wtedy i tylko wtedy, gdy istnieje bijekcja $f : A \rightarrow B$.

Pojęcie mocy zbioru, inaczej zwanej jego **liczbą kardynalną**, jest wygodnym skrótem myślowym. Jeśli użyjemy znaku \aleph_0 na oznaczenie mocy jakiegoś zbioru A , to napis $|B| = \aleph_0$ oznacza tyle samo co $B \sim A$.

Relacja równoliczności zbiorów tego samego typu jest relacją równoważności.

Zbiory przeliczalne

Liczba kardynalną (moc) zbioru \mathbb{N} oznaczamy symbolem \aleph_0 (**alef zero**). Mówimy, że zbiór A jest **przeliczalny** wtedy i tylko wtedy, gdy jest skończony lub jest zbiorem mocy \aleph_0 . W przeciwnym razie zbiór A jest **nieprzeliczalny**.

Przykłady zbiorów przeliczalnych: $\mathbb{N} \times \mathbb{N}, \mathbb{Z}, \mathbb{Q}$.

Suma przeliczalnej rodziny zbiorów przeliczalnych jest przeliczalna.

Przykład 1.

Pokażemy, że jeśli alfabet A jest przeliczalny, to zbiór wszystkich słów A^* też jest przeliczalny.

Niech A^n oznacza zbiór wszystkich słów nad A długości n . Nietrudno pokazać przez indukcję, że każdy ze zbiorów A^n jest przeliczalny. Skoro A^* jest sumą wszystkich A^n , dla $n \in \mathbb{N}$, teza wynika ze wspomnianego twierdzenia.

Przykład 2.

Zbiór $\{0, 1\}^{\mathbb{N}}$ wszystkich nieskończonych ciągów zerojedynkowych nie jest równoliczny z \mathbb{N} , a więc nie jest przeliczalny. W przeciwnym razie moglibyśmy wszystkie ciągi zerojedynkowe ustawić w nieskończony ciąg, tworząc dwuwymiarową tablicę bitów. Weźmy ciąg stworzony przez odwracanie kolejnych elementów na przekątnej – nietrudno zauważać, że ten ciąg nie może znajdować się na tablicy, skąd uzyskujemy sprzeczność.

■ Teoria mocy

Moc zbioru \mathbb{R} nazywamy **continuum** i oznaczamy przez \mathfrak{C} .

Przykłady zbiorów o mocy continuum: $\mathcal{P}(\mathbb{N})$, $\{0, 1\}^{\mathbb{N}}$, $\mathbb{N}^{\mathbb{N}}$.

Dla dowolnych niepustych zbiorów A, B następujące warunki są równoważne:

1. $|A| \leq |B|$,
2. istnieje injekcja $f : A \rightarrow B$,
3. istnieje surjekcja $f : B \rightarrow A$,
4. zbiór A jest równoliczny z pewnym podzbiorem zbioru B .

Ponadto warto pamiętać, że dla dowolnych zbiorów A, B , jeśli $A \subseteq B$, to $|A| \leq |B|$.

Bardzo istotne w teorii mocy jest **twierdzenie Cantora-Bernsteina**. Choć może wydawać się oczywiste, za jego pomocą możemy w łatwy sposób wyznaczać nieznane nam moce zbiorów, korzystając ze zbiorów, których moc znamy:

jeśli $|A| \leq |B|$ oraz $|B| \leq |A|$, to $|A| = |B|$

Przypis redakcji

Do twierdzenia Cantora-Bernsteina przydałoby się dorzucić przykład pokazujący jego zastosowanie w praktyce.

■ Arytmetyka liczb kardynalnych

Na liczbach kardynalnych, tak jak na zwykłych liczbach naturalnych, można wykonywać operacje arytmetyczne, jednakże z drobnymi różnicami. Dopóki operujemy na liczbach kardynalnych skończonych (tj. wprawdzie liczbach naturalnych), to arytmetyka jest dokładnie taka, jakiej możemy się spodziewać. W przeciwnym przypadku sprawa wygląda inaczej. Na ogół wystarczy pamiętać, że jeśli κ lub μ jest nieskończone, to:

1. $\kappa + \mu = \kappa \cdot \mu = \max\{\kappa, \mu\}$,
2. $\kappa - \mu = \kappa$ jeśli $\kappa > \mu$, wpp. 0,
3. zakładając, że $\mu \neq 0$, $\kappa/\mu = \kappa$ jeśli $\kappa > \mu$, dla $\kappa = \mu$ wynikiem może być każda liczba kardynalna $\leq \kappa$, zaś w przeciwnym przypadku operacja nie jest zdefiniowana.

Zestaw zadań

3.11. Każdy podzbiór \mathbb{R} o mocy continuum

- A. zawiera przedział otwarty
- B. jest nieograniczony
- C. ma przeliczalne dopełnienie

3.12. Istnieje nieskończenie wiele funkcji z liczb naturalnych w liczby naturalne, dla których

- A. obrazem zbioru $\{1, 2\}$ jest zbiór pusty
- B. obrazem zbioru $\{1, 2\}$ jest zbiór $\{2, 3, 4\}$
- C. przeciwbrazem zbioru $\{1, 2\}$ jest zbiór pusty

3.13. Równoliczne są

- A. zbiór liczb naturalnych i zbiór liczb wymiernych
- B. zbiór liczb rzeczywistych i zbiór potęgowy zbioru liczb naturalnych
- C. zbiór ciągów nieskończonych o wyrazach ze zbioru $\{0, 1\}$ i zbiór ciągów nieskończonych o wyrazach ze zbioru liczb naturalnych

3.14. Zbiorem mocy continuum jest

- A. zbiór liczb naturalnych
- B. zbiór potęgowy zbioru liczb naturalnych
- C. zbiór liczb niewymiernych

3.5.

Porządkи częściowe i ich własności

Relację $r \subseteq A \times A$ nazywamy relacją **częściowego porządku** w A , gdy jest zwrotna, antysymetryczna i przechodnia. Parę $\langle A, r \rangle$, a czasami sam zbiór A , nazywamy zbiorem **częściowo uporządkowanym** lub po prostu częściowym porządkiem. Jeśli dodatkowo relacja r jest spójna, to mówimy, że jest to relacja **liniowego porządku**.

Przykład 1.

Rozważmy kilka relacji:

- Relacja \leq w zbiorze liczb rzeczywistych jest liniowym porządkiem.
- Relacja podzielności $n \mid m$ jest częściowym porządkiem w zbiorze liczb naturalnych.
- Relacja inkluзji \subseteq jest częściowym porządkiem w zbiorze potęgowym liczb naturalnych $\mathcal{P}(\mathbb{N})$ (a także w dowolnej rodzinie zbiorów).
- Porządek leksykograficzny jest relacją liniowego porządku w zbiorze A^* (o ile alfabet jest liniowo uporządkowany, wpp. jest to relacja częściowego porządku).
- Relacja $<$ w zbiorze liczb rzeczywistych nie jest częściowym porządkiem, ponieważ nie jest zwrotna.

Poniżej przedstawione są trzy ważne definicje dla porządku częściowego $\langle A, \leq \rangle$:

1. Elementy $a, b \in A$ są **porównywale**, gdy $a \leq b$ lub $b \leq a$. W przeciwnym razie a, b są **nieporównywale**.

- Jeśli $B \subseteq A$ i każde dwa elementy zbioru B są porównywalne to mówimy, że B jest **łańcuchem** w A .
- Jeśli $B \subseteq A$ i każde dwa różne elementy zbioru B są nieporównywalne, to mówimy, że B jest **antylańcuchem** w A .

Przykład 2.

Oto proste przykłady łańcuchów i antylańcuchów:

- W porządku $\langle \mathbb{N}, | \rangle$ potęgi dwójki tworzą łańcuch, a liczby pierwsze – antylańcuch.
- Zbiory dwuelementowe tworzą antylańcuch w $\langle \mathcal{P}(\mathbb{N}), \subseteq \rangle$.

Elementy wyróżnione

Niech $\langle A, \leq \rangle$ będzie częściowym porządkiem i niech $a \in A$. Mówimy, że element a jest w zbiorze A :

- największy**, gdy $\forall_{x \in A} x \leq a$
- maksymalny**, gdy $\forall_{x \in A} a \leq x \Rightarrow a = x$
- najmniejszy**, gdy $\forall_{x \in A} a \leq x$
- minimalny**, gdy $\forall_{x \in A} x \leq a \Rightarrow a = x$

Z tego wynika, że jeśli a jest elementem największym (najmniejszym) to jest też jedynym elementem maksymalnym (minimalnym).

Przykład 3.

Oto kilka przykładów porządków oraz ich elementy wyróżnione:

- W porządku $\langle \mathbb{N} - \{0, 1\}, | \rangle$ nie ma elementu najmniejszego ani żadnych elementów maksymalnych. Natomiast liczby pierwsze są elementami minimalnymi.
- W zbiorze liczb rzeczywistych \mathbb{R} uporządkowanym przez zwykłą relację \leq , nie ma żadnych elementów minimalnych ani maksymalnych.
- W zbiorze potęgowym liczb naturalnych $\mathcal{P}(\mathbb{N})$ uporządkowanym przez inkluzję elementem najmniejszym jest \emptyset a największym \mathbb{N} .

Niech $\langle A, \leq \rangle$ będzie częściowym porządkiem i niech $B \subseteq A$ oraz $a \in A$. Mówimy, że a jest **ograniczeniem górnym** zbioru B ($a \geq b$), gdy $b \leq a$ dla wszystkich $b \in B$.

Element a jest **kresem górnym** zbioru B ($a = \sup B$), gdy jest najmniejszym ograniczeniem górnym B , czyli:

- $a \geq B$
- jeśli $c \geq B$, to $c \geq a$ dla dowolnego $c \in A$

Analogicznie definiujemy **ograniczenia dolne** ($a \leq B$) i **kresy dolne** ($a = \inf B$).

Przykład 4.

Rozważmy kilka przykładów kresów i ograniczeń:

- W rodzinie wszystkich podzbiorów zbioru A (uporządkowanej przez inkluzję) kresem górnym dowolnej podrodziny $X \subseteq \mathcal{P}(A)$ jest suma $\bigcup X$.
- W zbiorze liczb wymiernych \mathbb{Q} ze zwykłym uporządkowaniem zbiór $\{q \in \mathbb{Q} : q^2 < 2\}$ ma ograniczenia

górnego, ale nie ma kresu górnego.

Izomorfizmy porządków

Często mamy do czynienia z dwoma zbiorami, które są różne, ale „tak samo” uporządkowane. Takie porządkie nazywamy **izomorficznymi**. Mówimy, że zbiory częściowo uporządkowane $\langle A, \leq \rangle$ i $\langle B, \leq \rangle$ są izomorficzne, gdy istnieje bijekcja $f : A \rightarrow B$ spełniająca warunek $a \leq a' \Leftrightarrow f(a) \leq f(a')$ dla dowolnych $a, a' \in A$. Piszymy wtedy $A \approx B$, a funkcję f nazywamy izomorfizmem.

Jeśli dwa zbiory częściowo uporządkowane są izomorficzne i jeden z nich

- ma element najmniejszy, największy, maksymalny, minimalny;
- jest liniowo uporządkowany;
- spełnia dowolny warunek dotyczący tylko relacji porządkującej;

to ten drugi też ma odpowiednią własność.

Przykład 5.

Rozpatrzmy następujące podzbiory \mathbb{R} , uporządkowane jak zwykle:

- $A = \{1 - \frac{1}{n} \mid n \in \mathbb{N} - \{0\}\}$
- $A' = \{1 - \frac{1}{n} \mid n \in \mathbb{N} - \{0\}\} \cup \{1\}$
- $A'' = \{1 - \frac{1}{n} \mid n \in \mathbb{N} - \{0\}\} \cup \{1, 2\}$
- $B = \{m - \frac{1}{n} \mid n, m \in \mathbb{N} - \{0\}\}$

Zbiór wszystkich liczb naturalnych \mathbb{N} jest izomorficzny ze zbiorem A . Żadne dwa spośród zbiorów A, A', A'', B nie są izomorficzne (np. $A \not\approx A'$, ponieważ A nie ma elementu największego w przeciwieństwie do A').

Zestaw zadań

3.15. Dla $x \in \mathbb{N}$ niech $J(x)$ oznacza liczbę jedynek występujących w zapisie binarnym liczby x . Relację częściowego porządku w \mathbb{N} jest relacja

- A. $\{(x, y) \in \mathbb{N} \times \mathbb{N} : J(x) \leq J(y)\}$
- B. $\{(x, y) \in \mathbb{N} \times \mathbb{N} : J(x) < J(y) \text{ lub } x = y\}$
- C. $\{(x, y) \in \mathbb{N} \times \mathbb{N} : J(x) < J(y) \text{ lub } x \leq y\}$

3.16. Istnieje relacja równoważności na $\mathcal{P}(\mathbb{N})$, taka że

- A. jest częściowym porządkiem
- B. ma więcej niż continuum klas abstrakcji
- C. jej dopełnienie też jest relacją równoważności na $\mathcal{P}(\mathbb{N})$

3.17. Zbiór A ma moc \aleph_0 . Wynika z tego, że w częściowym porządku $\langle \mathcal{P}(A), \subseteq \rangle$

- A. każdy podzbiór ma kres górnny
- B. istnieje łańcuch o mocy continuum
- C. istnieje antylańcuch o mocy continuum

3.18. Relacja na zbiorze A : $\{(x, x) \mid x \in A\}$ jest

- A. relacją równoważności
- B. relacją częściowego porządku
- C. przechodnia

3.19. Rozpatrzmy zbiór funkcji różnowartościowych z \mathbb{N} w \mathbb{N} , uporządkowany „po współrzędnych”, tj. $f \leq g$ wtedy i tylko wtedy, gdy dla każdego $n \in \mathbb{N}$ zachodzi $f(n) \leq g(n)$. W tym porządku

- A. każdy niepusty podzbiór ma kres dolny
- B. istnieje antylańcuch mocy continuum
- C. istnieje łańcuch mocy continuum

3.6. Dobre ufundowanie i indukcja

Niech $\langle A, \leq \rangle$ będzie zbiorem częściowo uporządkowanym. Jeśli każdy niepusty podzbiór zbioru A ma element minimalny, to mówimy, że $\langle A, \leq \rangle$ jest częściowym dobrym porządkiem, lub, że A jest **dobrze ufundowany**. Jeśli ponadto porządek $\langle A, \leq \rangle$ jest liniowy, to jest to dobry porządek (wtedy każdy niepusty podzbiór A ma element najmniejszy).

Zbiór $\langle A, \leq \rangle$ jest dobrze ufundowany wtedy i tylko wtedy, gdy nie istnieje w nim nieskończony ciąg malejący.

Przykład 1.

Rozważmy kilka przykładów (częściowo) dobrych porządków:

- Zbiór \mathbb{N} jest dobrze uporządkowany.
- Zbiór \mathbb{N}^k , gdzie $k \in \mathbb{N}$, z porządkiem po współrzędnych jest dobrze uporządkowany.
- Zbiory $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ nie są dobrze uporządkowane.
- Porządek leksykograficzny na A^* , gdzie $A = \{a, b\}$ nie jest dobrze ufundowany.

Indukcja

Zasada indukcji, którą znamy dla liczb naturalnych, uogólnia się łatwo na dowolne zbiory dobrze ufundowane. Tę uogólnioną zasadę indukcji nazywamy czasem indukcją strukturalną.

Podzbiór B zbioru częściowo uporządkowanego A nazywamy odcinkiem początkowym w A , gdy

$$\forall_{x,y \in A} (x \in B \wedge y \leq x) \Rightarrow y \in B$$

Szczególny przypadek odcinka początkowego to odcinek wyznaczony przez element $x \in A$:

$$O_A(x) = \{y \in A \mid y < x\}$$

Przykład 2.

W zbiorze \mathbb{N} ze zwykłym porządkiem zachodzi $O_{\mathbb{N}}(n) = \{0, 1, \dots, n - 1\}$.

Analogicznie uogólniamy schemat definiowania przez indukcję. Jeśli $\langle A, \leq \rangle$ jest dobrze ufundowany, to możemy definiować funkcję $f : A \rightarrow B$, korzystając z dowolnych wartości $f(b)$ dla $b < a$ przy określaniu $f(a)$.

Przykład 3.

Funkcja $f : \mathbb{N} \setminus \{0, 1\} \rightarrow \mathbb{N}$ zdefiniowana w następujący sposób:

$$f(n) = \begin{cases} n, & n \text{ jest pierwsze} \\ f(m) + f(k), & n = mk, m, k \in \mathbb{N}_{>2} \end{cases}$$

jest zdefiniowana przez indukcję ze względu na dobrze ufundowaną relację podzielności.

Zestaw zadań

3.20. Porządek leksykograficzny jest dobrze ufundowany na

- A. \mathbb{N}^k dla każdego $k \in \mathbb{N}$
- B. \mathbb{N}^* – zbiór skończonych ciągów liczb naturalnych
- C. \mathbb{Q}^k dla każdego $k \in \mathbb{N}$

3.21. Każdy porządek częściowy można rozszerzyć do porządku

- A. liniowego
- B. dobrze ufundowanego
- C. dobrego

3.7.

Rachunek zdań

Formułą zdaniową nazwiemy wyrażenie zbudowane rekurencyjnie:

- Symbole zdaniowe (**zmiennne**), zazwyczaj oznaczane jako p, q, r, \dots , są formułami.
- Znaki \top i \perp (odpowiednio „prawda” i „fałsz”) są formułami.
- Jeśli α jest formułą, to $\neg\alpha$ też jest formułą.
- Formuły można łączyć spójnikami: alternatywa (\vee), koniunkcja (\wedge), implikacja (\Rightarrow) i równoważność (\Leftrightarrow).

Formuły zdaniowe mogą przyjmować wartości logiczne „prawda” i „fałsz” zależnie od **wartościowania**. Wartościowaniem dla danej formuły φ nazwiemy ustalone przypisanie każdej zmiennej zdaniowej występującej w φ wartości prawda/fałsz.

Formuła zdaniowa jest **spełnialna**, jeśli istnieje wartościowanie, dla którego ta formuła jest prawdziwa. Formuła jest **tautologią**, jeśli niezależnie od przyjętego wartościowania formuła ta zawsze jest prawdziwa. Warto znać następujące tautologie:

- prawo wyłączonego środka: $p \vee \neg p$,
- prawa De Morgana: $\neg(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$ oraz $\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$,
- alternatywne sformułowanie implikacji: $(p \Rightarrow q) \Leftrightarrow (\neg p \vee q)$,
- dwuwartościowość logiki zdaniowej: $p \Leftrightarrow \neg\neg p$.

Każdą formułę zdaniową można zapisać w **koniunkcyjnej postaci normalnej**, czyli jako koniunkcja alternatyw. Wygląda ona następująco:

$$(p_{i_0^0} \vee p_{i_1^0} \vee p_{i_2^0} \vee \dots) \wedge (p_{i_0^1} \vee p_{i_1^1} \vee p_{i_2^1} \vee \dots) \wedge \dots \wedge (p_{i_0^n} \vee p_{i_1^n} \vee p_{i_2^n} \vee \dots)$$

Naturalna dedukcja

Aby pokazać, że dana formuła jest twierdzeniem (tautologią), można przeprowadzić dowód przy pomocy **naturalnej dedukcji**.

Dowód w naturalnej dedukcji polega na wyprowadzaniu kolejnych wniosków z przyjętych wcześniej założeń. Każdy dowód jest skończonym ciągiem wniosków postaci "ponieważ A , to B ". Rozważać będziemy dowody w stylu Gentzena. System ten rozważa osady postaci $\Gamma \vdash \varphi$, gdzie Γ to **zbiór założeń**, zaś φ to **cel dowodowy**. Dowody w stylu Gentzena będące układać w skończone drzewa przy pomocy **reguł naturalnej dedukcji**:

$$\frac{}{\Gamma \cup \{\varphi\} \vdash \varphi} (\text{Ax})$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} (\text{W } \wedge)$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} (\text{E } \wedge)$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} (\text{E } \wedge)$$

Przypis redakcji

Rozdział nie jest jeszcze dokończony, jednak zawarta tu teoria jest wystarczająca do rozwiązymania zadań z przeanalizowanych na potrzeby tego repetytorium archiwalnych egzaminów.

Zestaw zadań

3.22. W klasycznym rachunku zdań ze spójnikami $\wedge, \vee, \neg, \Rightarrow$

- A. formuła $(p \Rightarrow (q \wedge w)) \Rightarrow (q \vee \neg p)$ jest spełnialna
- B. każda formuła zdaniowa bez negacji jest spełnialna
- C. każda formuła zdaniowa bez negacji jest prawdziwa

3.8.

Logika pierwszego rzędu

Przez **sygnaturę** rozumiemy pewien (zwykle skończony) zbiór symboli relacyjnych i funkcyjnych, każdy z ustaloną liczbą argumentów. **Arnością** symbolu nazwiemy liczbę argumentów, jakie przyjmuje. Szczerą przypadkiem są symbole funkcyjne arności 0, które przyjmują tylko jedną wartość. Takie symbole będące nazywać **stalymi**.

Modelem sygnatury nazwiemy niepusty zbiór (zwany dziedziną lub nośnikiem struktury) wraz z interpretacją symboli z sygnatury jako funkcji i relacji o odpowiedniej liczbie argumentów. Formalniej, dla sygnatury o symbolach funkcyjnych f_0, f_1, \dots, f_n i symbolach relacyjnych r_0, r_1, \dots, r_n , modelem będzie krotka:

$$\mathcal{A} = \langle A, f_0^{\mathcal{A}}, f_1^{\mathcal{A}}, \dots, f_n^{\mathcal{A}}, r_0^{\mathcal{A}}, r_1^{\mathcal{A}}, \dots, r_n^{\mathcal{A}} \rangle,$$

gdzie dla każdego i :

- $f_i^{\mathcal{A}} : A^k \rightarrow A$, gdzie k to arność f_i ,
- $r_i^{\mathcal{A}} \subseteq A^k$, gdzie k to arność r_i .

Przykład 1.

Rozważmy sygnaturę $\Sigma = \{+, \cdot, 0, 1, \leq\}$, gdzie $+, \cdot$ to dwuargumentowe symbole funkcyjne, $0, 1$ to stałe, a \leq to dwuargumentowy symbol relacyjny. Przykładowymi modelami dla tej sygnatury są:

- $\mathcal{R} = \langle \mathbb{R}, +, \cdot, 0, 1, \leq \rangle$ – zbiór liczb rzeczywistych z klasycznymi operacjami dodawania/mnożenia i klasycznym porządkiem,
- $\mathcal{N} = \langle \mathbb{N}, +, \cdot, 0, 1, \leq \rangle$ – zbiór liczb naturalnych z klasycznymi operacjami dodawania/mnożenia i klasycznym porządkiem,
- $\mathcal{P} = \langle \mathcal{P}(\mathbb{N}), \cup, \cap, \emptyset, \mathcal{P}(\mathbb{N}), \subseteq \rangle$ – rodzina podzbiorów \mathbb{N} z operacjami sumy/przecięcia i porządkiem po- przez inkluzję.

Niech V będzie zbiorem **zmiennych**, które są symbolami różnymi od elementów sygnatury. **Termem** nawiemy wyrażenie zbudowane ze zmiennych i symboli funkcyjnych. **Formułę logiki pierwszego rzędu** zdefiniujemy indukcyjnie:

- **Symbol atomowe**, czyli \top, \perp i wyrażenia postaci $r(t_0, \dots, t_n)$, gdzie r to symbol relacyjny, a t_i to terminy, są formułami.
- Formuły połączone klasycznymi spójnikami logicznymi są formułami.
- $\forall x : \varphi$ oraz $\exists x : \varphi$ są formułami, gdzie x to zmienna, φ to formuła, a \forall i \exists nazywamy odpowiednio **kwantyfikatorem ogólnym** i **egzystencjalnym**.

Zbiorem **zmiennych wolnych** formuły jest zbiór zmiennych, które nie są związane przez jakiś kwantyfikator. Dla formuły logiki pierwszego rzędu również istnieje postać normalna, zwana **preneksową postacią normalną**. Formuła jest w takiej postaci, jeśli wygląda następująco:

$$Q_0 x_0 Q_1 x_1 \dots Q_n x_n \varphi,$$

gdzie Q_i to kwantyfikatory, a φ to formuła bez kwantyfikatorów. Każda formuła daje się przedstawić w powyższej postaci.

3.9.

Rozwiązańa

Rozwiązańa

3.1. Dana jest rodzina zbiorów A oraz zbiory X, Y takie, że $X \in A$. Prawdą jest, że

- TAK** A. $(Y \subseteq X) \Rightarrow (Y \subseteq \bigcup A)$
TAK B. $\bigcap A \subseteq \bigcup A$
NIE C. $(Y \subseteq X) \Rightarrow (\bigcap A \subseteq Y)$

- A. $X \subseteq \bigcup A$, więc oczywiście zachodzi $Y \subseteq \bigcup A$.
B. Wprost z definicji iloczynu i sumy uogólnionej.
C. Niech $Y = \{1\}, X = \{1, 2\}$ oraz $A = \{\{1, 2\}, \{2, 3\}\}$. Wówczas $\bigcap A = \{2\}$ i nie zachodzi $\bigcap A \subseteq Y$.

3.2. Dane są trzy funkcje $f : A \rightarrow B$, $g : B \rightarrow C$ i $h : C \rightarrow D$, których złożenie $h \circ g \circ f : A \rightarrow D$ jest bijekcją. Wynika z tego, że

- TAK** A. f jest funkcją różnowartościową (injekcją)
NIE B. g jest bijekcją
TAK C. h jest na D (surjekcją)

A. Dowód nie wprost:

Załóżmy, że funkcja f nie jest różnowartościowa, czyli istnieją takie $x, y \in A$ i $z \in B$ dla których zachodzi $f(x) = z = f(y)$. Niezależnie od definicji g i h istnieje takie $w \in D$ dla którego $h(g(z)) = w$. Otrzymujemy więc, że $h(g(f(x))) = h(g(z)) = w = h(g(f(y)))$, z czego wynika, że złożenie $h \circ g \circ f$ nie jest injekcją, a co za tym idzie, nie jest także bijekcją. Uzyskanie sprzeczności z przyjętym wcześniej założeniem pozwala wywnioskować, że było ono fałszywe, a zatem funkcja f musi być różnowartościowa.

B. Kontrprzykład: zdefiniujemy zbiory: $A = D = \{0\}$, $B = C = \mathbb{N}$ oraz funkcje: $f(x) = x$, $g(x) = 0$, $h(x) = 0$. Jak łatwo zauważyc, złożenie $h \circ g \circ f$ jest bijekcją, a funkcja g nie jest ani injekcją, ani surjekcją, a więc tym bardziej nie jest bijekcją.

Dodatkowo, ten kontrprzykład pokazuje, że funkcja f nie musi być „na”, a funkcja h nie musi być różnowartościowa.

C. Jeżeli złożenie $h \circ g \circ f$ jest bijekcją, to wszystkie elementy ze zbioru D muszą być osiągane. Aby to zachodziło, w szczególności funkcja h (jako ostatnia aplikowana) musi przyjmować wszystkie elementy ze zbioru D , czyli musi być surjekcją.

3.3. Dana jest funkcja $f : \mathbb{N} \rightarrow \mathbb{N}$. Niech f^{2019} będzie 2019-krotnym złożeniem funkcji f . Prawdą jest, że

TAK **A.** f jest injekcją $\Leftrightarrow f^{2019}$ jest injekcją

TAK **B.** f jest surjekcją $\Leftrightarrow f^{2019}$ jest surjekcją

NIE **C.** $f(42) = 42 \Leftrightarrow f^{2019}(42) = 42$

W podpunkcie **A.** jasne jest, że składanie funkcji różnowartościowych daje funkcje różnowartościowe, co daje nam implikację w prawą stronę. Udowodnijmy implikację w lewą stronę. Przypuśćmy przeciwnie, że f nie jest injekcją. Zatem istnieją x, y , że $f(x) = f(y)$. Ale to też znaczy, że $f^{2018}(f(x)) = f^{2018}(f(y))$, czyli $f^{2019}(x) = f^{2019}(y)$ – sprzeczność.

W podpunkcie **B.** jasne jest, że składanie surjekcji daje surjekcje. Pokażemy implikację w lewą stronę. Zauważmy, że $f^{2019}(x) = f(f^{2018}(x))$. Niech $y \in \mathbb{N}$. Ponieważ f^{2019} jest surjekcją, to istnieje takie $x \in \mathbb{N}$, że $f^{2019}(x) = y$. Ale również $f^{2019}(x) = f(f^{2018}(x)) = y$. Zatem f osiąga wartość y . Z dowolności y mamy, że f jest „na”.

W podpunkcie **C.** implikacja w prawą stronę jest trywialna – jeśli $f(42) = 42$, to 42 jest punktem stałym funkcji f , zatem nieważne ile razy nałożymy f na 42, nadal otrzymamy wartość 42. Nie zachodzi jednak implikacja w lewo. Kontrprzykład: $f = \lambda x. x + 1 \text{ mod } 2019$.

3.4. Jeśli $f : A \rightarrow B$ jest różnowartościowa, to funkcja obrazu $f^\rightarrow : \mathcal{P}(A) \rightarrow \mathcal{P}(B)$

TAK **A.** jest różnowartościowa

NIE **B.** jest funkcją odwrotną do funkcji przeciobrazu $f^\leftarrow : \mathcal{P}(B) \rightarrow \mathcal{P}(A)$

TAK **C.** spełnia warunek $\bigcup\{f^\rightarrow(Z) : Z \in \mathcal{L}\} = f^\rightarrow(\bigcup \mathcal{L})$ dla dowolnej rodziny $\mathcal{L} \subseteq \mathcal{P}(A)$

A. Rozważmy dwa różne podzbiory $A_1, A_2 \subseteq A$, b.s.o. $\exists a \in A_2 - A_1$. Wtedy $f(a) \in f^\rightarrow(A_2)$ oraz $f(a) \notin f^\rightarrow(A_1)$, a więc f^\rightarrow jest różnowartościowa.

B. Rozważmy $f : \mathbb{N} \rightarrow \mathbb{N}$ zdefiniowaną wzorem $f(n) = n + 1$ oraz niech $B = \{0, 1\}$. Wtedy $f^\leftarrow(B) = \{0\}$, ale $f^\rightarrow(\{0\}) = \{1\} \neq B$.

C. Aby uniknąć pełnego dowodu, zastanówmy się, co oznacza dana równość. Po jej lewej stronie mamy sumę po obrazach zbiorów z rodziny \mathcal{L} . Zatem tworzymy zbiór wartości możliwych do uzyskania poprzez aplikację f na elementach każdego $Z \in \mathcal{L}$. Oczywiście jest to równoważne zapisowi $f^\rightarrow(\bigcup \mathcal{L})$.

3.5. Niech f będzie funkcją ze zbioru A w zbiór B . Wynika z tego, że

NIE **A.** obraz zbioru A przy funkcji f to zbiór B

TAK **B.** przeciobraz zbioru B przy funkcji f to zbiór A

NIE C. jeśli X jest niepustym podzbiorem B , to przeciobraz zbioru X przy funkcji f jest niepusty

- A. Jeśli funkcja f nie jest surjekcją, to obraz całej dziedziny nie jest całą przeciwdziedziną, np. dla $f : \mathbb{N} \rightarrow \mathbb{N}$, $f(n) = n + 1$ obrazem \mathbb{N} będzie $\mathbb{N} - \{0\}$.
- B. Ponieważ dla każdego elementu z dziedziny A funkcja jest określona (wynika to bezpośrednio z definicji funkcji), więc biorąc jako przeciobraz całe B na pewno uzyskamy całe A .
- C. Możemy rozważyć f z podpunktu A. oraz niepusty zbiór $X = \{0\}$, $X \subseteq \mathbb{N}$. Wtedy przeciobraz X przy f jest pusty.

3.6. Niech A będzie dowolnym zbiorem i niech $s, r \subseteq A \times A$ będą relacjami. Jeśli s i r są

TAK A. zwrotne, to $s \cup r$ jest relacją zwrotną

TAK B. symetryczne, to $s \cup r$ jest relacją symetryczną

NIE C. przechodnie, to $s \cup r$ jest relacją przechodnią

- A. Skoro s oraz r , są zwrotne, to dla każdego a element $\langle a, a \rangle$ należy do obu relacji, a więc też do ich sumy.
- B. Weźmy dowolne $\langle x, y \rangle \in s \cup r$. Ten element musi też należeć do co najmniej jednej relacji s i r . Bez straty ogólności założymy, że $\langle x, y \rangle \in s$. Z symetryczności jest też $\langle y, x \rangle \in s$, więc $\langle y, x \rangle \in s \cup r$.
- C. Kontrprzykład: $A = \mathbb{N}$, $s = \{\langle 1, 2 \rangle\}$, $r = \{\langle 2, 3 \rangle\}$. Z przechodniości $\langle 1, 3 \rangle$ powinno należeć do $s \cup r$, a tak nie jest.

3.7. Niech $f : A \rightarrow B$ będzie funkcją „na” B i niech s_A będzie relacją równoważności na A . Przez $f^{-1}(X)$ oznaczamy przeciobraz X przy f . Następująca relacja r jest relacją równoważności na B :

NIE A. $b \sim b'$ wtedy i tylko wtedy, gdy $f^{-1}(\{b\}) \cup f^{-1}(\{b'\})$ jest pewną klasą abstrakcji relacji s_A

NIE B. $b \sim b'$ wtedy i tylko wtedy, gdy istnieją $a, a' \in A$ takie, że $a \in f^{-1}(\{b\})$ i $a' \in f^{-1}(\{b'\})$ oraz $a s_A a'$

TAK C. $b \sim b'$ wtedy i tylko wtedy, gdy dla każdych $a, a' \in A$ takich, że $a \in f^{-1}(\{b\})$ i $a' \in f^{-1}(\{b'\})$, zachodzi $a s_A a'$

- A. Badana relacja nie jest relacją równoważności, bo w ogólności nie musi być zwrotna – nie musi zachodzić, że dla dowolnego $b \in B$ zbiór $f^{-1}(\{b\})$ jest pewną klasą abstrakcji s_A . Na przykład: $A = B = \mathbb{N}$, $f = \text{id}_{\mathbb{N}}$, $s_A = \mathbb{N} \times \mathbb{N}$.

- B. Rozważmy zbiory $A = \{1, 2, 3, 4\}$ oraz $B = \{1, 2, 3\}$, funkcję f zdefiniowaną następująco:

$$f(1) = 1, \quad f(2) = 2, \quad f(3) = 2, \quad f(4) = 3$$

oraz relację s_A wyznaczoną przez klasy abstrakcji $\{1, 2\}, \{3, 4\}$.

Wtedy 1 jest w relacji r z 2 (ponieważ $f^{-1}(\{1\}) = \{1\}$, $f^{-1}(\{2\}) = \{2, 3\}$ i 1 jest w relacji s_A z 2, więc przyjmując $a = 1$, $a' = 2$ spełniony jest warunek z podpunktu) oraz 2 jest w relacji r z 3 (ponieważ $f^{-1}(\{2\}) = \{2, 3\}$, $f^{-1}(\{3\}) = \{4\}$ i 3 jest w relacji s_A z 4, więc dla $a = 3$, $a' = 4$ warunek jest spełniony). Natomiast 1 nie jest w relacji r z 3, ponieważ $f^{-1}(\{1\}) = \{1\}$, $f^{-1}(\{3\}) = \{4\}$, a 1 nie jest w relacji s_A z 3.

Podsumowując powyższe rozważania, zachodzi $1 \sim 2$ oraz $2 \sim 3$, ale nie zachodzi $1 \sim 3$. Relacja r nie jest więc przechodnia, a co za tym idzie – nie jest też relacją równoważności.

- C. Mamy tu do czynienia z relacją równoważności, co pokażemy wprost z definicji:

- *Zwrotność.* Niech $b \in B$. Pokażemy, że $b \sim b$. Musimy pokazać, że dla każdego $a \in f^{-1}(\{b\})$ zachodzi $a s_A a$. To jest oczywiście prawda, bo s_A jest relacją równoważności, więc jest zwrotna.

- *Symetryczność.* Niech $b, b' \in B$ takie, że $b \sim b'$. Pokażemy, że $b' \sim b$. Z założenia mamy, że dla każdych $a, a' \in A$, takich że $a \in f^{-1}(\{b\})$ i $a' \in f^{-1}(\{b'\})$ zachodzi $a s_A a'$. Musimy pokazać, że dla każdych $x, y \in A$, takich że $x \in f^{-1}(\{b'\})$ i $y \in f^{-1}(\{b\})$ zachodzi $x s_A y$. Zbiory $f^{-1}(\{b'\})$ i $f^{-1}(\{b\})$ są niepuste, zatem weźmy dowolne takie x, y i pokażmy, że $x s_A y$. Z wniosku z założenia, przyjmując $a = y$ i $a' = x$, mamy $y s_A x$. Ale s_A jest relacją równoważności, więc jest też symetryczna, czyli $x s_A y$. Z dowolności x, y dostajemy $b' \sim b$.

- *Przechodniość.* Niech $b, b', b'' \in B$ takie, że $b \, r \, b'$ oraz $b' \, r \, b''$. Pokażemy, że $b \, r \, b''$. Z założenia mamy, że:

- $\forall a, a' \in A : a \in f^{-1}(\{b\}) \text{ i } a' \in f^{-1}(\{b'\}) \Rightarrow a \, s_A \, a'$,
- $\forall a', a'' \in A : a' \in f^{-1}(\{b'\}) \text{ i } a'' \in f^{-1}(\{b''\}) \Rightarrow a' \, s_A \, a''$.

Musimy pokazać, że dla każdych $x, y \in A$ takich, że $x \in f^{-1}(\{b\})$ i $y \in f^{-1}(\{b''\})$ zachodzi $x \, s_A \, y$. Ponieważ te przeciwbrazy są niepuste (f jest „na”), weźmy dowolne x, y spełniające pierwsze warunki. Musimy pokazać, że $x \, s_A \, y$. Weźmy dowolne $z \in f^{-1}(\{b'\})$. Z założeniem i warunków (a) i (b) mamy, że $x \, s_A \, z$ oraz $z \, s_A \, y$. Ale s_A jest przechodnia, zatem $x \, s_A \, y$.

- 3.8.** r jest relacją równoważności na liczbach naturalnych dodatnich określona w następujący sposób: liczby x i y są w relacji r wtedy i tylko wtedy, gdy zbiory dzielników pierwszych liczb x i y są takie same. Wynika z tego, że

NIE A. wszystkie klasy abstrakcji relacji r są nieskończone

NIE B. wszystkie klasy abstrakcji relacji r są równoliczne

NIE C. zbiór ilorazowy relacji r jest skończony

- Klasa abstrakcji, do której należy „1”, jest jednoelementowa, bo żadna inna liczba nie posiada tego zbioru dzielników pierwszych.
- Klasa abstrakcji, do której należy „1”, jest jednoelementowa, a wszystkie inne są nieskończone.
- Istnieje nieskończoność wielu klas abstrakcji, ponieważ każda klasa abstrakcji ma swojego reprezentanta w zbiorze potęgowym zbioru liczb pierwszych. Moc takiego zbioru jest oczywiście nieskończona.

- 3.9.** W zbiorze 5-elementowym

NIE A. każda relacja przechodnia ma moc co najmniej 3

TAK B. każda relacja przechodnia ma moc co najwyżej 25

NIE C. istnieje relacja przechodnia o mocy równej 24

- Odwołując się do definicji przechodniości ($\forall_{x,y,z \in A} x \, r \, y \wedge y \, r \, z \Rightarrow x \, r \, z$), kontrprzykładem jest m.in. relacja pusta.
- Każda relacja w zbiorze 5-elementowym ma moc co najwyżej 25 (nie da się utworzyć 26. pary), więc w szczególności każda relacja przechodnia także ma tę własność.
- Skoro maksymalna relacja r_{max} (tj. taka, że każdy element jest w relacji z każdym) w zbiorze 5-elementowym jest mocy 25, to spróbujmy utworzyć relację przechodnią, zabierając jedną parę z maksymalnej relacji. Zauważmy jednak, że jeśli a, b, c to (niekoniecznie różne) elementy pierwszego 5-elementowego zbioru, a my usuniemy z r_{max} parę $\langle a, b \rangle$, to wciąż istnieją w niej pary $\langle a, c \rangle$ oraz $\langle c, b \rangle$, więc nie uzyskujemy przechodniości. Z ogólności rozważań wynika, że nie da się utworzyć relacji przechodniej o mocy 24.

- 3.10.** Niech r będzie relacją równoważności na niepustym zbiorze A . Wynika z tego, że

TAK A. każda klasa abstrakcji relacji r jest niepusta

TAK B. dowolne dwie różne klasy abstrakcji relacji r są rozłączne

TAK C. suma zbioru klas abstrakcji relacji r jest równa A

Ponieważ zbiór ilorazowy (zbiór klas abstrakcji) relacji równoważności można utożsamić z podziałem zbioru, to każdy z punktów można utożsamić z konkretnym jego warunkiem:

A. $\forall_{p \in P} p \neq \emptyset$

B. $\forall_{p,q \in P} p = q \vee p \cap q = \emptyset$

C. $\bigcup P = A$

3.11. Każdy podzbiór \mathbb{R} o mocy continuum

- NIE** A. zawiera przedział otwarty
NIE B. jest nieograniczony
NIE C. ma przeliczalne dopełnienie

Pokażemy kontrprzykłady do każdego podpunktu:

- A. $\mathbb{R} - \mathbb{Q}$
B. $(0, 1)$
C. $(0, 1)$

3.12. Istnieje nieskończenie wiele funkcji z liczb naturalnych w liczby naturalne, dla których

- NIE** A. obrazem zbioru $\{1, 2\}$ jest zbiór pusty
NIE B. obrazem zbioru $\{1, 2\}$ jest zbiór $\{2, 3, 4\}$
TAK C. przeciwbrazem zbioru $\{1, 2\}$ jest zbiór pusty

Mamy do czynienia z funkcjami $f : \mathbb{N} \rightarrow \mathbb{N}$.

- A. Mamy $f(\{1, 2\}) = \{f(1), f(2)\}$. Ponieważ f jest określone dla każdej liczby naturalnej, obraz $\{1, 2\}$ nie może być pusty.
B. Obraz zbioru $\{1, 2\}$ ma maksymalnie dwie wartości. Nie może zatem być równy $\{2, 3, 4\}$.
C. Przeciwbraz to $f^{-1}(\{1, 2\}) = \{a \in \mathbb{N} \mid f(a) \in \{1, 2\}\}$. Oczywiście, przeciwbraz tego zbioru może być pusty. Wystarczy rozpatrzyć funkcje $f(n) = n + k$ dla wszystkich $k > 3$, jest ich nieskończenie wiele.

3.13. Równoliczne są

- TAK** A. zbiór liczb naturalnych i zbiór liczb wymiernych
TAK B. zbiór liczb rzeczywistych i zbiór potęgowy zbioru liczb naturalnych
TAK C. zbiór ciągów nieskończonych o wyrazach ze zbioru $\{0, 1\}$ i zbiór ciągów nieskończonych o wyrazach ze zbioru liczb naturalnych
- A. Obydwa są mocy \aleph_0 .
B. Obydwa są mocy \mathfrak{C} .
C. Obydwa są mocy \mathfrak{C} .

3.14. Zbiorem mocy continuum jest

- NIE** A. zbiór liczb naturalnych
TAK B. zbiór potęgowy zbioru liczb naturalnych
TAK C. zbiór liczb niewymiernych
- A. Zbiór \mathbb{N} jest mocy \aleph_0 .
B. Zbiór $\mathcal{P}(\mathbb{N})$ jest mocy $2^{\mathbb{N}}$, czyli \mathfrak{C} .
C. Zbiór liczb niewymiernych to $\mathbb{R} - \mathbb{Q}$. Skoro \mathbb{R} jest mocy \mathfrak{C} , a \mathbb{Q} mocy \aleph_0 , to zbiór liczb niewymiernych musi być mocy \mathfrak{C} .

3.15. Dla $x \in \mathbb{N}$ niech $J(x)$ oznacza liczbę jedynek występujących w zapisie binarnym liczby x . Relację częścioowego porządku w \mathbb{N} jest relacja

- NIE** A. $\{(x, y) \in \mathbb{N} \times \mathbb{N} : J(x) \leq J(y)\}$
TAK B. $\{(x, y) \in \mathbb{N} \times \mathbb{N} : J(x) < J(y) \text{ lub } x = y\}$

NIE C. $\{\langle x, y \rangle \in \mathbb{N} \times \mathbb{N} : J(x) < J(y) \text{ lub } x \leq y\}$

- A. Nie jest to relacja częściowego porządku, ponieważ nie jest antysymetryczna. Na przykład, liczby 1 i 2 mają w zapisie binarnym tyle samo jedynek, ale nie są to te same liczby.
- B. Ta relacja jest częściowym porządkiem i aby to udowodnić, pokażemy, że jest ona zwrotna, antysymetryczna i przechodnia.
- *Zwrotność.* Bezpośrednio z definicji relacji.
 - *Antysymetryczność.* Rozważamy cztery przypadki. Pierwszy z nich, to że dla pewnego x, y z definicji antysymetryczności xry oraz yrx , gdy $x = y$ oraz $y = x$. Wtedy oczywiście relacja antysymetryczna. Podobnie dla przypadków, gdy xry lub yrx to odpowiednio $x = y$ lub $y = x$. Przypadek $J(x) < J(y) \wedge J(y) < J(x)$ nie może natomiast zachodzić. Zatem relacja jest antysymetryczna.
 - *Przechodniość.* Podobne przypadki jak dla antysymetryczności: dla $J(x) < J(y) \wedge J(y) < J(z)$ (gdzie x, y, z jak w definicji przechodniości) zachodzi oczywiście $J(x) < J(z)$. Podobnie jest dla przypadków $J(x) < J(y) \wedge y = z$ oraz $x = y \wedge J(y) < J(z)$. Dla $x = y \wedge y = z$ mamy natomiast $x = z$. Zatem relacja ta jest przechodnia.
- C. Nie jest to relacja częściowego porządku, ponieważ nie jest antysymetryczna. Przykładem mogą być liczby 4 i 3: 4 jest w relacji z 3, ponieważ w zapisie binarnym 4 ma mniej jedynek niż 3 oraz 3 jest w relacji z 4, ponieważ jest od niej mniejsza. Jednak nie są to te same liczby, zatem nie jest spełniona antysymetryczność.

3.16. Istnieje relacja równoważności na $\mathcal{P}(\mathbb{N})$, taka że

TAK A. jest częściowym porządkiem

NIE B. ma więcej niż continuum klas abstrakcji

NIE C. jej dopełnienie też jest relacją równoważności na $\mathcal{P}(\mathbb{N})$

- A. Możemy wziąć relację taką, że x jest w relacji tylko z samym sobą (relacja $\langle \mathcal{P}(\mathbb{N}), = \rangle$). Jest to oczywiście relacja równoważności oraz taka relacja na $\mathcal{P}(\mathbb{N})$ jest częściowym porządkiem.
- B. Nie jest to możliwe, ponieważ mamy tylko continuum elementów.
- C. Nie jest to możliwe, ponieważ z definicji relacji równoważności, musi ona być zwrotna. Dopełnienie relacji równoważności na $\mathcal{P}(\mathbb{N})$ nie będzie już tego spełniało.

3.17. Zbiór A ma moc \aleph_0 . Wynika z tego, że w częściowym porządku $\langle \mathcal{P}(A), \subseteq \rangle$

TAK A. każdy podzbiór ma kres gorny

TAK B. istnieje łańcuch o mocy continuum

TAK C. istnieje antylańcuch o mocy continuum

- A. Tak, jest to suma elementów maksymalnych tego podzbioru
- B. Niech $A = \mathbb{Q}$. Weźmy funkcję $f : \mathbb{R} \rightarrow \mathcal{P}(\mathbb{Q})$ taką, że $f(r) = \{x \in \mathbb{Q} \mid x < r\}$. Funkcja ta jest izomorfizmem – $a < b$ wtedy $f(a) \subseteq f(b)$. W dodatku jest różnowartościowa, zatem $f(\mathbb{R})$ jest mocy continuum i jest szukanym łańcuchem.
- C. Weźmy nieskończone, ukorzenione, pełne drzewo binarne z wierzchołkami ponumerowanymi liczbami naturalnymi. Niech każda nieskończona ścieżka w oczywisty sposób generuje podzbiór \mathbb{N} . Rodzina wszystkich takich podzbiorów jest szukanym antylańcuchem – żadne dwie ścieżki nie generują porównywalnych podzbiorów, bo to by znaczyło, że jedna ścieżka zawiera się w drugiej, co nie jest możliwe.

Inny przykład – weźmy funkcję $f : \mathbb{R} \rightarrow \mathcal{P}(\mathbb{Q})$ taką, że $f(x) = (x, x + 1)$ (przedział w liczbach wymiernych). Żadne takie dwa przedziały nie są porównywalne, ponadto jest ich continuum wiele, bo \mathbb{R} jest mocy continuum, a f jest różnowartościowa.

3.18. Relacja na zbiorze A : $\{\langle x, x \rangle \mid x \in A\}$ jest

- TAK** A. relacją równoważności
TAK B. relacją częściowego porządku
TAK C. przechodnia

Zadanie jest proste, jeśli uświadomimy sobie, że owa relacja to relacja $\langle A, = \rangle$.

- A. W oczywisty sposób widać, że relacja jest zwrotna, symetryczna i przechodnia, a więc jest relacją równoważności.
B. Wiemy już, że relacja jest zwrotna i przechodnia. Widzimy, że jest również antysymetryczna, więc jest to relacja częściowego porządku.
C. Wprost z definicji przechodniości.

3.19. Rozpatrzmy zbiór funkcji różnowartościowych z \mathbb{N} w \mathbb{N} , uporządkowany „po współrzędnych”, tj. $f \leq g$ wtedy i tylko wtedy, gdy dla każdego $n \in \mathbb{N}$ zachodzi $f(n) \leq g(n)$. W tym porządku

- NIE** A. każdy niepusty podzbiór ma kres dolny
TAK B. istnieje antylańcuch mocy continuum
TAK C. istnieje łańcuch mocy continuum

- A. Niech $f = \mathbb{1}$, a $g(0) = 1, g(1) = 0, g(n) = n$. Wówczas zbiór składający się z tych dwóch funkcji nie ma ograniczenia dolnego, więc w szczególności nie ma kresu dolnego.
B. Dla każdego podzbioru $A \subseteq \mathbb{N}$ konstrujemy funkcję f następująco: f jest „domyślnie” funkcją identycznościową, a jeśli $x \in A$, to zamieniamy ze sobą wartości funkcji f na pozycjach $2x$ i $2x + 1$. Na przykład, zbiór $\{0\}$ wygeneruje funkcję, która przyjmuje kolejno wartości: $1, 0, 2, 3, 4, 5, \dots$. Żadna taka funkcja nie jest porównywalna z żadną inną, bo jeśli dwa ciągi nie zgadzają się na jakiekolwiek pozycji, to od razu ich wygenerowane funkcje są nieporównywalne.
C. Dla każdego podzbioru $A \subseteq \mathbb{N}$ konstrujemy funkcję f następująco: $f(x) = 2x$, jeśli $x \in A$, zaś $2x + 1$ wpp. Innymi słowy, jest to funkcja $\lambda x. 2x$, ale gdy $x \in A$, to na tej samej pozycji wartość f „podskakuje” do góry o 1. Łatwo zauważać, że takie mapowanie dobrze modeluje zbiór $\mathcal{P}(\mathbb{N})$ (bo po „podskokach” łatwo wywnioskować wyjściowy zbiór). Zatem, dwie tak wygenerowane funkcje są porównywalne wtw. ich początkowe podzbiory \mathbb{N} były porównywalne ze względu na zawieranie. Na koniec weźmy dowolny antylańcuch X w zbiorze $\mathcal{P}(\mathbb{N})$ ze względu na relację \subseteq . Wtedy wystarczy każdy element X zmapować na funkcję tak, jak opisano powyżej.

3.20. Porządek leksykograficzny jest dobrze ufundowany na

- TAK** A. \mathbb{N}^k dla każdego $k \in \mathbb{N}$
NIE B. \mathbb{N}^* – zbiór skończonych ciągów liczb naturalnych
NIE C. \mathbb{Q}^k dla każdego $k \in \mathbb{N}$
- A. To po prostu fakt, który trzeba pamiętać.
B. Można utworzyć ciąg nieskończoność zstępujący: $(1), (0, 1), (0, 0, 1), \dots$
C. \mathbb{Q} przy zwykłym porządku nie jest dobrze ufundowany, więc to też nie jest (szczególny przypadek dla $k = 1$).

3.21. Każdy porządek częściowy można rozszerzyć do porządku

- TAK** A. liniowego
NIE B. dobrze ufundowanego
NIE C. dobrego
- A. Wystarczy dodać porównania między brakującymi elementami tak, żeby w interpretacji grafowej (diagramie Hassego) porządku z każdego węzła dało się do dowolnego innego dojść, startując w jednym z nich.

- B.** Kontrprzykład: $\langle \mathbb{Z}, \leq \rangle$ jest częściowym (a nawet liniowym) porządkiem, ale niezależnie od tego, jak go rozszerzymy, zawsze będzie istnieć nieskończony ciąg malejący.
C. Rozumowanie analogiczne jak w **B.**

3.22. W klasycznym rachunku zdań ze spójnikami $\wedge, \vee, \neg, \Rightarrow$

- TAK** **A.** formuła $(p \Rightarrow (q \wedge w)) \Rightarrow (q \vee \neg p)$ jest spełnialna
TAK **B.** każda formuła zdaniowa bez negacji jest spełnialna
NIE **C.** każda formuła zdaniowa bez negacji jest prawdziwa
- A.** Wystarczy, że lewa strona implikacji jest fałszem, czyli że p jest prawdą, a q i w fałszami.
B. Widać to, kiedy zapiszemy formułę w koniunkcyjnej postaci normalnej. Wtedy ustawiając wartości wszystkich zmiennych na „prawdę”, na pewno spełnimy daną formułę zdaniową.
C. Nie, bo np. $p \Rightarrow q$ nie jest prawdziwe dla $p = 1$ i $q = 0$.

4

Matematyka dyskretna

Materiały teoretyczne z matematyki dyskretnej zostały opracowane na podstawie slajdów Adama Malinowskiego oraz notatki Błażeja Wilkoławskego.

Podstawa programowa

1. Metody obliczania **sum skończonych**.
2. **Współczynniki dwumianowe** i inne liczby specjalne występujące w kombinatoryce.
3. Równania rekurencyjne i **funkcje tworzące**.
4. **Metody zliczania**: zasada włączania-wyłączania, enumeratory.
5. **Grafy**: podstawowe pojęcia, cykle Eulera i Hamiltona.
6. **Grafy dwudzielne**: skojarzenia i twierdzenie Halla.
7. **Planarność i kolorowanie** grafów.
8. Elementarna **teoria liczb**: podzielność, liczby pierwsze, rozkład na czynniki, NWD i algorytm Euklidesa.
9. **Arytmetyka modularna**: małe i duże twierdzenie Fermata, twierdzenie Eulera, chińskie twierdzenie o resztach.
10. **Asymptotyka**: notacja asymptotyczna, twierdzenie o rekurencji uniwersalnej, szacowanie sum.

4.1.

Sumy i współczynniki dwumianowe

Wyróżniamy kilka podstawowych własności obliczania skończonych sum:

- $\sum_i kx_i = k \cdot \sum_i x_i$ (**wyłączanie czynnika przed sumą**)
- $\sum_i (x_i + y_i) = \sum_i x_i + \sum_i y_i$ (**rozbicie złożonej sumy**)
- $\sum_{i=1}^n a_j = n \cdot a_j$ (**zamiana sumy niezależnych składników na iloczyn**)

Warto przypomnieć sobie też wzory na **sumę ciągu arytmetycznego i geometrycznego**: niech (a_n) będzie arytmetyczny, a (g_n) – geometryczny o ilorazie q . Wtedy suma n początkowych wyrazów każdego z ciągów jest równa

$$\sum_{i=1}^n a_i = \frac{a_1 + a_n}{2} \cdot n, \quad \sum_{i=1}^n g_i = \frac{1 - q^n}{1 - q} \cdot g_1$$

■ Obliczanie sum skończonych

Ta część zawiera krótki przegląd najczęstszych metod i strategii obliczania skończonych sum.

Pierwszą z nich jest **metoda zaburzania**. Możemy ją zastosować, kiedy interesują nas skończone sumy odcinków początkowych pewnego ciągu (a_n) , czyli sumy postaci $S_n = \sum_{i=0}^n a_i$. Strategia polega na obliczeniu wartości S_{n+1} za pomocą S_n na dwa różne sposoby – wydzielając raz pierwszy, a raz ostatni składnik sumy:

$$S_n + a_{n+1} = a_0 + \sum_{i=0}^n a_{i+1}$$

Jeśli uda nam się ostatnią sumę wyrazić za pomocą S_n , to otrzymamy równanie, którego rozwiążanie jest poszukiwaną sumą.

Przykład 1.

Znajdziemy zwarty wzór sumy $S_n = \sum_{i=0}^n i2^i$ za pomocą metody zaburzania. Rozpiszmy:

$$\begin{aligned} S_n + (n+1)2^{n+1} &= 0 \cdot 2^0 + \sum_{i=0}^n (i+1)2^{i+1} \\ &= 2 \sum_{i=0}^n i2^i + 2 \sum_{i=0}^n 2^i \\ &= 2S_n + 2(2^{n+1} - 1), \end{aligned}$$

gdzie suma $\sum 2^i = 2^{n+1} - 1$ została obliczona ze wzoru na sumę ciągu geometrycznego. Po przekształceniu równania i wyliczeniu z niego S_n , otrzymujemy ostatecznie

$$S_n = (n+1)2^{n+1} - 2(2^{n+1} - 1) = (n-1)2^{n+1} + 2$$

Jeśli mamy do czynienia z **sumą wielokrotną**, czyli taką, w której występuje więcej niż jedna zmienna, to najczęściej opłaca się ją zapisać jako **sumę iterowaną** (sumę wewnętrz sumy):

$$\underbrace{\sum_{\substack{1 \leq i \leq j \leq n}} a_{ij}}_{\text{suma wielokrotna}} = \underbrace{\sum_{i=1}^n \sum_{j=i}^n a_{ij}}_{\text{suma iterowana}}$$

Przy obliczaniu sum iterowanych często korzystamy także z **zamiany kolejności zmiennych** – tak aby po takiej zamianie nowa postać była prosta do przeliczenia.

Przykład 2.

Zdefiniujmy k -tą liczbę harmoniczną jako $H_k = \sum_{i=1}^k \frac{1}{i}$. Znajdziemy zwarty wzór na sumę n początkowych liczb harmonicznych:

$$\sum_{k=1}^n H_k = \sum_{k=1}^n \sum_{i=1}^k \frac{1}{i} \stackrel{(*)}{=} \sum_{i=1}^n \sum_{k=i}^n \frac{1}{i} = \sum_{i=1}^n \left((n-i+1) \cdot \frac{1}{i} \right) = \sum_{i=1}^n \frac{n-i+1}{i} = (n+1)H_n - n$$

W równości (*) użyto zamiany kolejności zmiennych ($k \leftrightarrow i$) w sumie iterowanej.

Czasem suma przyjmuje prostszą postać po zastosowaniu odpowiedniej **zamiany zmiennych**, tj. podstawieniu nowej zmiennej w miejsce jakiegoś wyrażenia i odpowiednim przeindeksowaniu.

Dla przykładu, do obliczenia sumy $\sum_{j=i}^n (j-i)$ moglibyśmy podstawić $k = j-i$ i otrzymać $\sum_{k=0}^{n-i} k$, a to jest prosta do obliczenia suma $(n-i)$ początkowych liczb naturalnych.

■ Współczynniki dwumianowe

Współczynnik dwumianowy $\binom{n}{k}$ to liczba k -elementowych podzbiorów n -elementowego zbioru.

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

Wyrażenie $\binom{n}{k}$ czytamy jako „ n nad k ”, „ n po k ” albo „ k z n ”.

Bezpośrednio z definicji otrzymujemy kilka podstawowych własności:

$$\binom{n}{0} = \binom{n}{n} = 1, \quad \binom{n}{1} = n, \quad \binom{n}{k} = 0 \text{ dla } k > n, \quad \binom{n}{k} = \binom{n}{n-k}, \quad \binom{n}{k} = \frac{n}{k} \cdot \binom{n-1}{k-1}$$

Sama nazwa „współczynniki dwumianowe” wzięła się stąd, że liczby te pojawiają się w rozwinięciu dwumianu $(x+y)^n$. O tym fakcie mówi **twierdzenie o dwumianie**: dla $x, y \in \mathbb{R}$ oraz $n \in \mathbb{N}$ mamy

$$(x+y)^n = \sum_{i=0}^n \binom{n}{i} x^i y^{n-i}$$

Oto dwa ważne zastosowania twierdzenia o dwumianie – warto je zapamiętać:

$$(1+1)^n = \sum_{i=0}^n \binom{n}{i} = 2^n, \quad (-1+1)^n = \sum_{i=0}^n (-1)^i \binom{n}{i} = \begin{cases} 1 & \text{dla } n = 0, \\ 0 & \text{wpp.} \end{cases}$$

4.2.

Permutacje, liczby Stirlinga

Permutacja zbioru X to bijekcja $f : X \rightarrow X$.

Permutacje możemy zapisać na kilka sposobów, m.in. za pomocą

- **listy**, np. $\langle 4, 1, 5, 2, 3 \rangle$ (pierwszy element przechodzi na drugie miejsce, drugi element na czwarte miejsce itd.)
- **zapisu cyklowego** – złożenia rozłącznych cykli, np. $[2, 4, 1][5, 3]$ (taki zapis jest niejednoznaczny, można zamieniać miejscami cykle oraz elementy w obrębie cyklu)
- „**superzapisu**” – ulepszzonego, jednoznacznego zapisu cyklowego: każdy cykl rozpoczyna się najmniejszym swoim elementem oraz cykle są posortowane malejąco, patrząc na ich pierwszy element. W takim zapisie (ze względu na jednoznaczność) możemy pominąć nawiasy kwadratowe. Dla przykładu, permutacja $[2, 4, 1][6][5, 3]$ w superzapisie zostanie przedstawiona jako $6\ 3\ 5\ 1\ 2\ 4$ ($= [6][3, 5][1, 2, 4]$).

O permutacji, która ma λ_i cykli długości i , powiemy, że jest typu $1^{\lambda_1} 2^{\lambda_2} \dots n^{\lambda_n}$.

■ Liczby Stirlinga

Liczba Stirlinga I rodzaju $\left[\begin{matrix} n \\ k \end{matrix} \right]$ to liczba n -permutacji o k cyklach.

Przykład 1.

$\left[\begin{matrix} 4 \\ 2 \end{matrix} \right] = 11$, bo jest $\binom{4}{1} \cdot (3-1)! = 8$ permutacji typu $1^1 3^1$ oraz $\binom{4}{2}/2 = 3$ permutacji typu 2^2 .

Nie ma jawnego wzoru na liczby Stirlinga I rodzaju, ale ich wartość można obliczać, korzystając ze wzoru rekurencyjnego:

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} \text{ dla } k > 0, \quad \begin{bmatrix} n \\ 0 \end{bmatrix} = [n=0]$$

Liczba Stirlinga II rodzaju $\begin{Bmatrix} n \\ k \end{Bmatrix}$ to liczba podziałów n zbioru na k bloków.

Przykład 2.

$\begin{Bmatrix} 4 \\ 2 \end{Bmatrix} = 7$, bo są cztery podziały z singletonem i blokiem 3-elementowym:

$$\{\{1\}, \{2, 3, 4\}\}, \dots, \{\{4\}, \{1, 2, 3\}\}$$

oraz trzy podziały na bloki 2-elementowe:

$$\{\{1, 2\}, \{3, 4\}\}, \{\{1, 3\}, \{2, 4\}\}, \{\{1, 4\}, \{2, 3\}\}$$

Tak samo jak w przypadku rodzaju I, znany jest tylko wzór rekureencyjny

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = k \begin{Bmatrix} n-1 \\ k \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} \text{ dla } k > 0, \quad \begin{Bmatrix} n \\ 0 \end{Bmatrix} = [n=0]$$

Pozostałe (nietrudne) własności, wynikające z definicji liczb Stirlinga:

$$\begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)! \text{ dla } n > 0, \quad \sum_k \begin{bmatrix} n \\ k \end{bmatrix} = n!, \quad \begin{Bmatrix} n \\ 1 \end{Bmatrix} = 1, \quad \begin{Bmatrix} n \\ 2 \end{Bmatrix} = 2^{n-1} - 1,$$

To było na egzaminie

Dla $n \geq 6$ liczba permutacji zbioru $\{1, 2, 3, \dots, n\}$, w których 1 i 2 są w różnych cyklach długości 3, jest równa

- A. $(n-2)!$
- B. $\binom{n-2}{2} \binom{n-4}{2} (n-4)!$
- C. $24 \cdot \sum_{k=2}^{n-4} \binom{n-2}{4} \begin{bmatrix} n-6 \\ k-2 \end{bmatrix}$, gdzie $\begin{bmatrix} a \\ b \end{bmatrix}$ to liczba Stirlinga pierwszego rodzaju

Prześledźmy, na ile sposobów możemy wybierać kolejne liczby w permutacji p . Uwaga co do zapisu: $p(1)$ oznacza „liczbę, na którą przechodzi jedynka” (przypominamy, że permutacja jest funkcją):

- $p(1)$ możemy wybrać na $n-2$ sposobów, bo pasuje wszystko poza 1 i 2,
- $p(p(1))$ możemy wybrać na $n-3$ sposobów, bo pasuje wszystko poza 1, 2 i $p(1)$,
- $p(p(p(1))) = 1$, bo cykl musi mieć długość 3 – jednoznaczne,
- $p(2)$ możemy wybrać na $n-4$ sposobów, bo pasuje wszystko poza 1, 2, $p(1)$, $p(p(1))$,
- $p(p(2))$ możemy wybrać na $n-5$ sposobów, bo pasuje wszystko poza 1, 2, $p(1)$, $p(p(1))$, $p(2)$,
- $p(p(p(2))) = 2$, bo cykl musi mieć długość 3 – jednoznaczne,
- wartości dla pozostałych $n-6$ liczb mogą być dowolne, więc mamy $(n-6)!$ sposobów.

Widzimy zatem, że w podpunkcie A. odpowiedź to TAK. Zarazem nietrudno zauważać, że odpowiedź do podpunktu B. to NIE, bo $\binom{n-2}{2} \binom{n-4}{2} > (n-2)(n-3)$ dla dużych n .

Aby lepiej zrozumieć podpunkt C., ułożymy interpretację kombinatoryczną. Cztery liczby, które będą należeć do cykli z „1” i „2” wybieramy na $\binom{n-2}{4}$ sposobów. Przeprowadzamy rozumowanie analogiczne do poprzedniego, żeby wyliczyć, że można ułożyć je we wspomniane cykle na $4! = 24$ sposobów. Niech k będzie liczbą cykli w permutacji. Resztę wartości wybieramy na $\binom{n-6}{k-2}$ sposobów (z definicji liczb Stirlinga). Sumując się po k , otrzymujemy wszystkie możliwe sposoby. Zatem odpowiedź to TAK.

Zestaw zadań

- 4.1.** Niech f oznacza permutację $\langle 3, 6, 1, 4, 2, 5 \rangle$ i niech f^k oznacza k -krotne złożenie permutacji f . Wynika z tego, że
- A. $f^8 = \langle 1, 2, 3, 4, 5, 6 \rangle$
 B. $f^7 = f$
 C. $f^4 = f^{16}$
- 4.2.** Jeśli ponumerujemy permutacje zbioru $\{1, 2, 3, 4, 5\}$ od 1 do 120 w porządku leksykograficznym, to permutacją o numerze 60 będzie
- A. $\langle 3, 5, 1, 2, 4 \rangle$
 B. $\langle 3, 4, 1, 2, 5 \rangle$
 C. $\langle 3, 2, 5, 4, 1 \rangle$
- 4.3.** Liczba permutacji liczb $\{1, 2, \dots, 2021\}$, takich że „1” jest w cyklu parzystej długości, jest równa
- A. $2021!/2$
 B. $1010 \cdot 2020!$
 C. $1011 \cdot 2020!$

4.3.

Funkcje tworzące

Funkcja tworząca ciągu $\langle a_n \rangle_{n \in \mathbb{N}}$ to obiekt reprezentujący nieskończony ciąg:

$$A(x) = \sum_{n=0}^{\infty} a_n x^n$$

co zapisujemy również jako $A(x) \leftrightarrow \langle a_n \rangle_{n \in \mathbb{N}}$. Zbieżność szeregu formalnego jest nieistotna, dopóki nie podstawimy za x konkretnych wartości.

Własności funkcji tworzących dla $A(x) \leftrightarrow \langle a_n \rangle_{n \in \mathbb{N}}$ i $B(x) \leftrightarrow \langle b_n \rangle_{n \in \mathbb{N}}$:

- liniowość:

$$\alpha A(x) + \beta B(x) \leftrightarrow \langle \alpha a_n + \beta b_n \rangle_{n \in \mathbb{N}}$$

- przesunięcie w prawo:

$$x^m A(x) \leftrightarrow \left\langle \underbrace{0, \dots, 0}_m, a_0, a_1, \dots \right\rangle$$

- przesunięcie w lewo:

$$\frac{A(x) - \sum_{i=0}^{m-1} a_i x^i}{x^m} \leftrightarrow \langle a_m, a_{m+1}, \dots \rangle$$

- splot:

$$A(x) \cdot B(x) = \sum_n \sum_k a_k b_{n-k} x^n \leftrightarrow \left\langle \sum_k a_k b_{n-k} \right\rangle_{n \in \mathbb{N}}$$

- różniczkowanie:

$$A'(x) = a_1 + 2a_2 x + \dots \leftrightarrow \langle (n+1)a_{n+1} \rangle_{n \in \mathbb{N}}$$

- całkowanie:

$$\int_0^x A(t) dt = a_0 x + \frac{1}{2} a_1 x^2 + \dots \leftrightarrow \left\langle \frac{a_{n-1}}{n} \right\rangle_{n \in \mathbb{N}}$$

- podstawianie:

$$A(\alpha x) \leftrightarrow \langle \alpha^n a_n \rangle_{n \in \mathbb{N}}$$

Przykład 1.

Oto przykłady funkcji tworzących dla wybranych ciągów:

- ciąg samych jedynek: $\langle 1, 1, 1, \dots \rangle \leftrightarrow 1 + x + x^2 + \dots = \frac{1}{1-x}$
- splot dowolnego ciągu (a_n) z ciągiem samych jedynek $\langle 1, 1, 1, \dots \rangle$ to **ciąg sum częściowych**

$$\langle a_0 + a_1 + \dots + a_n \rangle_{n \in \mathbb{N}}$$

, więc np. splot $\langle 1, 1, 1, \dots \rangle$ z $\langle 1, 1, 1, \dots \rangle$ to

$$\frac{1}{(1-x)^2} \leftrightarrow \langle n+1 \rangle_{n \in \mathbb{N}}$$

i ogólniej

$$\frac{1}{(1-x)^k} \leftrightarrow \left\langle \binom{n+k-1}{k-1} \right\rangle_{n \in \mathbb{N}}$$

- $(\ln \frac{1}{1-x})' = \frac{1}{1-x} = 1 + x + x^2 + \dots$, skąd na mocy całkowania:

$$\ln \frac{1}{1-x} \leftrightarrow \left\langle 0, 1, \frac{1}{2}, \frac{1}{3}, \dots \right\rangle$$

■ Wykładnicze funkcje tworzące

Oprócz „zwykłych” funkcji tworzących definiujemy także **wykładnicze funkcje tworzące**, tutaj dla ciągu $\langle b_n \rangle_{n \in \mathbb{N}}$:

$$B_e(x) = \sum_{n=0}^{\infty} \frac{b_n x^n}{n!}$$

Własności wykładniczych funkcji tworzących dla $A_e(x) \leftrightarrow \langle a_n \rangle_{n \in \mathbb{N}}$ i $B_e(x) \leftrightarrow \langle b_n \rangle_{n \in \mathbb{N}}$:

- liniowość:

$$\alpha A_e(x) + \beta B_e(x) \leftrightarrow \langle \alpha a_n + \beta b_n \rangle_{n \in \mathbb{N}}$$

- splot dwumianowy:

$$A_e(x) \cdot B_e(x) = \sum_n \left(\sum_k \binom{n}{k} a_k b_{n-k} \right) \frac{x^n}{n!} \leftrightarrow \left\langle \sum_k \binom{n}{k} a_k b_{n-k} \right\rangle_{n \in \mathbb{N}}$$

- różniczkowanie (przesuwanie w lewo):

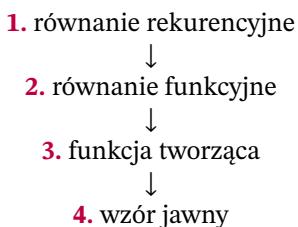
$$B'_e(x) = \sum_n \frac{b_{n+1} x^n}{n!} \leftrightarrow \langle b_1, b_2, \dots \rangle$$

- całkowanie (przesuwanie w prawo):

$$\int_0^x \sum_{n \geq 0} \frac{b_n t^n}{n!} dt = \sum_{n \geq 1} \frac{b_{n-1} x^n}{n!} \leftrightarrow \langle 0, b_0, b_1, \dots \rangle$$

Równania rekurencyjne

Funkcje tworzące pomagają nam rozwiązywać równania rekurencyjne. Ogólny schemat postępowania:



Równanie rekurencyjne przepiszemy na równanie funkcyjne, skąd łatwo dostaniemy funkcję tworzącą dla ciągu. Następnie, korzystając z rozkładu na sumę ułamków prostych lub wielomianu lustrzanego, z funkcji tworzącej dostaniemy jawny wzór na n -ty wyraz ciągu. Schemat postępowania łatwo prześledzimy na przykładzie:

Przykład 2.

Rozwiążemy równanie rekurencyjne

$$1. \begin{cases} a_0 = a_1 = 1 \\ a_n = 2a_{n-1} + a_{n-2} \text{ dla } n \geq 1 \end{cases}$$

Korzystając z notacji Iversona, włączamy warunki brzegowe do równania:

$$a_n = 2a_{n-1} + a_{n-2} + [n=0] - [n=1], \text{ dla } n \geq 0$$

Mnożąc stronami przez x^n i sumując po n dostajemy

$$2. \sum_n a_n x^n = 2 \sum_n a_{n-1} x^n + \sum_n a_{n-2} x^n + 1 - x$$

czyli równanie funkcyjne liniowe na funkcję tworzącą ciągu $\langle a_n \rangle_{n \in \mathbb{N}}$:

$$A(x) = 2x \cdot A(x) + x^2 \cdot A(x) + 1 - x$$

Stąd dostajemy funkcję tworzącą tego ciągu

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

Żeby rozwinać funkcję wymierną w szereg potęgowy, przedstawiamy jej mianownik jako iloczyn czynników postaci $(1-\lambda x)^k$ i rozkładamy na sumę ułamków prostych postaci

$$\frac{C}{(1-\lambda x)^k} = \sum_{n \geq 0} C \binom{n+k-1}{k-1} \lambda^n x^n$$

U nas:

$$A(x) = \frac{1-x}{(1-(1-\sqrt{2})x)(1-(1+\sqrt{2})x)} = \frac{\alpha}{1-(1-\sqrt{2})x} + \frac{\beta}{1-(1+\sqrt{2})x}$$

Zatem

$$4. a_n = \alpha(1-\sqrt{2})^n + \beta(1+\sqrt{2})^n \text{ dla } n \geq 0$$

gdzie α i β możemy wyznaczyć z warunków brzegowych przez rozwiązywanie układu równań liniowych

$$\begin{cases} 1 = a_0 = \alpha + \beta \\ 1 = a_1 = \alpha(1-\sqrt{2}) + \beta(1+\sqrt{2}) \end{cases} \Rightarrow \begin{cases} \alpha = \frac{1}{2} \\ \beta = \frac{1}{2} \end{cases}$$

Zestaw zadań

4.4. Niech $A(x)$ będzie funkcją tworzącą ciągu $\langle a_n \rangle_{n \in \mathbb{N}}$.

- A. $\frac{A(x)}{1-x}$ jest funkcją tworzącą ciągu $\langle a_0 + a_1 + \dots + a_n \rangle_{n \in \mathbb{N}}$
- B. $A^2(x)$ jest funkcją tworzącą ciągu $\langle a_n^2 \rangle_{n \in \mathbb{N}}$
- C. $xA(x)$ jest funkcją tworzącą ciągu $\langle a_{n+1} \rangle_{n \in \mathbb{N}}$

4.5. Funkcją tworzącą $A(x)$ ciągu $\langle (n+1)2^n \rangle_{n \in \mathbb{N}}$ jest

- A. $\frac{1}{(1-2x)(1-x)}$
- B. $\frac{d}{dx} \frac{1}{1-2x}$
- C. $\int_0^x \frac{dt}{1-2t}$

4.6. Niech $A(x)$ będzie funkcją tworzącą (zwykłą) ciągu $\langle a_n \rangle_{n \in \mathbb{N}}$. Wynika z tego, że funkcją tworzącą ciągu

- A. $\langle -a_n \rangle_{n \in \mathbb{N}}$ jest $A(-x)$
- B. $\langle 2^n a_n \rangle_{n \in \mathbb{N}}$ jest $A(2x)$
- C. $\langle \sum_{k=0}^n a_k \rangle_{n \in \mathbb{N}}$ jest $\frac{A(x)}{1-x}$

4.4.

Metody zliczania

W tym rozdziale przytoczymy kilka obiektów i zasad ułatwiających kombinatoryczne zliczanie rozmaitych elementów. Pierwszą z metod będzie **zasada włączania-wyłączania**:

Załóżmy, że elementy uniwersum X mogą mieć n różnych własności nazwanych A_1, A_2, \dots, A_n . Oznaczmy:

$$S_j = \sum_{1 \leq i_1 \leq \dots \leq i_j \leq n} |A_{i_1} \cap \dots \cap A_{i_j}|$$

W szczególności $S_0 = |X|$. Jeśli przez $D(k)$ oznaczymy liczbę elementów mających dokładnie k własności, to mamy $D(0) = \sum_{j \geq 0} (-1)^j S_j$, a ogólniej:

$$D(k) = \sum_{j \geq k} \binom{j}{k} (-1)^{j-k} S_j$$

Przykład 1.

Obliczymy liczbę n -nieporządków, czyli n -permutacji f takich, że dla każdego i zachodzi $f(i) \neq i$.

Uniwersum X to zbiór wszystkich n -permutacji; $A_i = \{n\text{-permutacje } f \text{ takie, że } f(i) = i\}$. Wtedy zachodzi $|A_{i_1} \cap \dots \cap A_{i_j}| = (n-j)!$ (wartości permutacji w j punktach są ustalone, w pozostałych mogą być dowolne), zatem:

$$D(0) = \sum_{j=0}^n (-1)^j \binom{n}{j} (n-j)! = n! \sum_{j=0}^n (-1)^j \frac{1}{j!}$$

■ Enumeratory

Enumerator to funkcja tworząca zliczająca obiekty kombinatoryczne. Warto przypomnieć sobie podstawowe wzory zliczające te obiekty:

- r -kombinacje (r -podzbiory n -zbioru)

$\binom{n}{r}$ – do kombinacji wybieramy r spośród n elementów

- r -kombinacje z powtórzeniami

$\binom{n+r-1}{r}$ – metoda *stars and bars*, gwiazdki są miejscami w kombinacji, a kreski oddzielają krotności

na przykład $* * || * | * * *$ odpowiada kombinacji $\{1, 1, 3, 4, 4, 4\}$

- r -permutacje

$n^r = n(n-1)\dots(n-r+1)$ – najpierw n możliwości, później $(n-1)$ itd.

- r -permutacje z powtórzeniami

n^r – na każde miejsce mamy n możliwości wyboru

Enumeratorem kombinacji r -elementowych ze zbioru n -elementowego jest funkcja

$$\sum_{r=0}^n \binom{n}{r} t^r = (1+t)^n \quad (\text{twierdzenie o dwumianie})$$

Interpretację kombinatoryczną uzyskamy, symulując przemnażanie n nawiasów wyrażenia $(1+t)^n$: wybranie z i -tego nawiasu czynnika t odpowiada wybraniu i -tego elementu do podzbioru, a wybranie jedynki ($= t^0$) odpowiada pominięciu i -tego elementu.

Jeśli i -ty element może wystąpić $\alpha_1, \alpha_2, \dots, \alpha_k$ razy, to zamienimy i -ty nawias na $(t^{\alpha_1} + t^{\alpha_2} + \dots + t^{\alpha_k})$.

Przykład 2.

Enumerator kombinacji z powtórzeniami:

$$(1+t+t^2+\dots)^n = \left(\frac{1}{1-t}\right)^n = (1-t)^{-n} = \sum_{r \geq 0} \binom{-n}{r} (-t)^r = \sum_{r \geq 0} \binom{n+r-1}{r} t^r$$

Przykład 3.

Enumerator kombinacji, w których każdy element musi wystąpić przynajmniej raz:

$$(t+t^2+\dots)^n = \left(\frac{t}{1-t}\right)^n = t^n \sum_{r \geq 0} \binom{n+r-1}{r} t^r \stackrel{(*)}{=} \sum_{r \geq n} \binom{r-1}{r-n} t^r$$

W równości oznaczonej (*) zastosowaliśmy podstawienie $r+n \mapsto r$.

Dla zliczania permutacji wykorzystujemy wykładnicze funkcje tworzące: **enumerator r -permutacji** bez powtórzeń zbioru n -elementowego to funkcja

$$(1+t)^n = \sum_{r \geq 0} n^r \frac{t^r}{r!}$$

Analogicznie, jeśli i -ty element może wystąpić $0, 1, \dots, k$ razy, to zamieniamy i -ty nawias na $\left(1+t+\frac{t^2}{2!}+\dots+\frac{t^k}{k!}\right)$.

Przykład 4.

Enumerator permutacji z powtórzeniami:

$$\left(1 + t + \frac{t^2}{2!} + \dots\right)^n = (\exp(t))^n = \exp(n \cdot t) = \sum_{r \geq 0} n^r \frac{t^r}{r!}$$

Przykład 5.

Wyznaczmy enumerator ciągów liter A , B i C , w których A występuje co najmniej raz, a B – nieparzystą liczbę razy.

Taki ciąg liter długości r traktujemy jako r -permutację z powtórzeniami zbioru $\{A, B, C\}$ spełniającą podane warunki. Jej enumeratorem jest:

$$\left(t + \frac{t^2}{2!} + \frac{t^3}{3!} + \dots\right) \left(t + \frac{t^3}{3!} + \frac{t^5}{5!} + \dots\right) \left(1 + t + \frac{t^2}{2!} + \dots\right) = (\exp(t) - 1) \cdot \frac{\exp(t) - \exp(-t)}{2} \cdot \exp(t)$$

Zestaw zadań

4.7. Liczba sposobów na umieszczenie $n > 0$ nieróżnialnych kul w $k > 0$ rozróżnialnych komorach to współczynnik przy wyrazie

- A. x^n funkcji $(1-x)^{-k}$
- B. x^k funkcji $(1-x)^{-n}$
- C. x^{k-1} funkcji $(1-x)^{-n-1}$

4.8. Dany jest ciąg $a_n = |\{(A, x) : A \subseteq \{1, \dots, n\}, x \in A\}|$. Wtedy

- A. $\lim_{n \rightarrow \infty} \frac{a_{n+1}}{a_n} = \infty$
- B. a_n jest parzyste dla $n \geq 2$
- C. $a_n = n \cdot n!$ dla $n > 0$

4.9. Liczba podzbiorów zbioru 2018-elementowego o mocy co najwyżej 1009 jest równa

- A. liczbie podzbiorów zbioru 2018-elementowego o mocy większej niż 1009
- B. $\binom{2018}{1009}$
- C. 2^{2017}

4.10. Liczba par (A, B) , takich że $A, B \subseteq \{1, 2, \dots, 2023\}$ oraz

- A. $A \subseteq B$ jest równa 3^{2023}
- B. $A \cap B = \emptyset$ jest równa 3^{2023}
- C. $|A \cup B|$ jest liczbą parzystą, jest równa 3^{2023}

4.5.

Teoria grafów

Graf nieskierowany to para $G = \langle V, E \rangle$, gdzie

- V jest zbiorem **wierzchołków**,

- E jest zbiorem **krawędzi**, czyli nieuporządkowanych par postaci $\{a, b\}$.

O ile nie jest powiedziane inaczej, wykluczamy multigrafy (grafy z powtórzeniami w zbiorze krawędzi) oraz pętle (krawędzie typu $\{a, a\}$).

W **grafie skierowanym** krawędź jest parą uporządkowaną $\langle a, b \rangle$, oznaczaną również $a \rightarrow b$.

G' jest **podgrafem** G , jeśli $V_{G'} \subseteq V_G$ oraz $E_{G'} \subseteq E_G$

O krawędzi $\{a, b\}$ powiemy, że jest **incydentna** z wierzchołkami a, b . **Stopień wierzchołka** (ozn. $\deg(v)$) to liczba krawędzi z nim incydentnych. Faktem jest poniższy **lemat o uściskach dłoni**:

$$\sum_{v \in V} \deg(v) = 2|E|$$

Powyższą własność nietrudno zrozumieć intuicyjnie: każda krawędź łączy dwa wierzchołki, a zatem dodając do siebie stopnie sąsiadujących wierzchołków, liczymy każdą z krawędzi dwukrotnie.

W grafach skierowanych rozróżniamy także **stopień wejściowy i wyjściowy** – liczbę krawędzi odpowiednio wchodzących i wychodzących z danego wierzchołka.

Powiemy, że graf jest **k -regularny**, jeśli każdy jego wierzchołek ma stopień k . **Klika** (graf pełny) K_n o n wierzchołkach to graf $(n - 1)$ -regularny.

Cykł w grafie to taki ciąg wierzchołków $\langle v_1, v_2, \dots, v_k \rangle$, że $v_1 = v_k$ oraz każde dwa kolejne wierzchołki są połączone krawędzią, przy czym żadna krawędź nie jest wykorzystana dwukrotnie. **Długość cyklu** to liczba jego wierzchołków (lub, równoważnie, krawędzi). **Cykł prosty** to cykl bez powtórzeń wierzchołków.

Graf jest **spójny** (a w przypadku skierowanym – **silnie spójny**), jeśli między dowolnymi dwoma jego wierzchołkami istnieje ścieżka. Maksymalny spójny podgraf to **spójna składowa**.

Graf jest **dwuspójny**, jeśli usunięcie żadnego pojedynczego wierzchołka (oraz incydentnych z nim krawędzi) go nie rozspojni. **Dwuspójna składowa** to maksymalny podgraf dwuspójny. Prawdziwe są następujące fakty:

- dwuspójna składowa to maksymalny podzbiór krawędzi grafu, taki że każda krawędź jest częścią cyklu prostego w stosunku z każdą inną krawędzią,
- w dwuspójnej składowej pomiędzy każdą parą wierzchołków istnieją dwie rozłączne krawędziowo drogi.

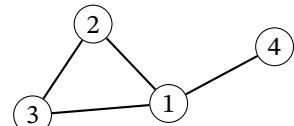
Podzbiór I zbioru wierzchołków grafu G jest **niezależny**, gdy dla dowolnej pary wierzchołków $a, b \in I$ zachodzi $\{a, b\} \notin E_G$.

Podzbiór M zbioru krawędzi grafu G jest **skojarzeniem**, gdy dowolna para krawędzi $e_1, e_2 \in M$ nie ma wspólnych wierzchołków ($e_1 \cap e_2 = \emptyset$). Skojarzenie M jest **doskonałe**, jeśli pokrywa wszystkie wierzchołki, tj. dla każdego wierzchołka $v \in V_G$ istnieje krawędź $e \in M$, taka że $v \in e$.

Przykład 1.

W zbiorze wierzchołków grafu na rysunku obok istnieją dwa dwuelementowe niezależne podzbiory: $\{2, 4\}$ oraz $\{3, 4\}$.

Skojarzeniem E_G jest na przykład $\{\{2, 3\}, \{1, 4\}\}$ i to skojarzenie jest doskonałe.



Drzewo to graf spójny bez cykli. Następujące warunki są równoważne:

1. G jest drzewem.
2. Każde dwa wierzchołki w G są połączone dokładnie jedną drogą.
3. G jest minimalny spójny.
4. G jest maksymalny acykliczny.
5. G jest spójny i $|V| = |E| + 1$.

Cykle Eulera i Hamiltona

Cykl Eulera w grafie to cykl przechodzący przez każdą krawędź dokładnie raz. Powiemy, że graf jest **eulerowski**, gdy zawiera cykl Eulera.

Warunek konieczny i wystarczający: graf spójny G jest eulerowski wtedy i tylko wtedy, gdy **każdy jego wierzchołek jest parzystego stopnia**, a dla grafów skierowanych – każdy wierzchołek ma równy stopień wyjściowy i wejściowy.

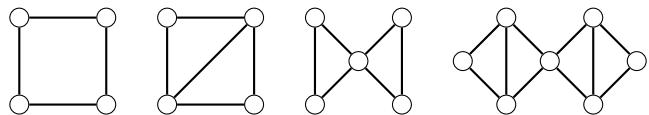
Cykl Hamiltona to cykl przechodzący przez każdy wierzchołek dokładnie raz. Powiemy, że graf jest **hamiltonowski**, gdy zawiera cykl Hamiltona.

W przeciwnieństwie do cyklu Eulera, ogólny problem rozstrzygnięcia, czy dany graf ma cykl Hamiltona, jest zwykle bardzo trudny.

Przykład 2.

Na rysunku znajdują się kolejno przykłady grafu:

- eulerowskiego i hamiltonowskiego
- nieeulerowskiego, ale hamiltonowskiego
- eulerowskiego, ale niehamiltonowskiego
- nieeulerowskiego i niehamiltonowskiego



Grafy dwudzielne

Graf G jest grafem **dwudzielnym**, jeśli $V_G = V_1 \cup V_2$, gdzie V_1 i V_2 są rozłącznymi zbiorami wierzchołków i każda krawędź ma jeden koniec w V_1 , a drugi w V_2 .

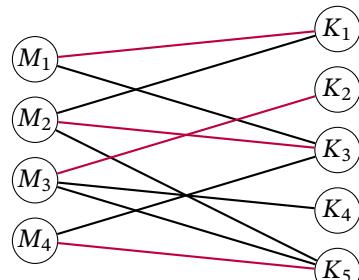
Graf pełny dwudzienny $K_{n,m}$ to taki, w którym $|V_1| = n$, $|V_2| = m$ oraz każdy wierzchołek z V_1 jest połączony z każdym innym z V_2 .

Warunek konieczny i wystarczający: graf jest dwudzielny wtedy i tylko wtedy, gdy **nie zawiera cykli nieparzystej długości**.

Skojarzenia (zbiory niezależnych krawędzi) w grafach dwudzielnych to **Systemy Różnych Reprezentantów** (SRR). Koncept ten jest wykorzystywany przy rozwiązywaniu niektórych problemów z teorii zbiorów, jak na poniższym przykładzie.

Przykład 3.

Chcemy wybrać żony dla mężczyzn M_1, \dots, M_4 wśród kandydatek K_1, \dots, K_5 , tak aby każdy mężczyzna ożenił się z kobietą, którą zna.



Sporządzmy (dwudzienny) graf znajomości: jeśli M_i zna K_j , to odpowiednio wierzchołki połączymy krawędzią.

Rozwiązaniem problemu jest wybór Systemu Różnych Reprezentantów, na przykład odpowiadającego skojarzeniu

$$\{\{M_1, K_1\}, \{M_2, K_3\}, \{M_3, K_2\}, \{M_4, K_5\}\}$$

Naturalnie nasuwającym się pytaniem jest to, kiedy System Różnych Reprezentantów dla danego grafu dwudzielnego w ogóle istnieje. Odpowiedź przynosi nam **twierdzenie Halla**. Okazuje się, że warunkiem koniecznym i wystarczającym na istnienie SRR jest to, by **każda podgrupa k kobiet znała co najmniej k mężczyzn**.

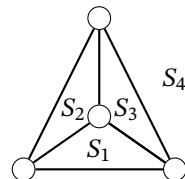
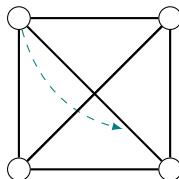
■ Planarność

Płaskie włożenie grafu w płaszczyznę to przedstawienie go graficznie w postaci, w której jego krawędzie nie przecinają się. Graf, który ma takie włożenie, nazywamy **planarnym**.

Ścianą w płaskim włożeniu nazywać będziemy maksymalny spójny obszar płaszczyzny rozłączny z grafem. Jedna ze ścian (zewnętrzna) jest nieograniczona.

Przykład 4.

Poniżej przedstawiono klikę K_4 i jej płaską reprezentację, wraz z wyróżnionymi ścianami. Płaskie włożenie uzyskano, przesuwając wierzchołek z lewego górnego rogu zgodnie ze strzałką.



Liczba ścian płaskiego włożenia grafu spełnia **wzór Eulera**:

$$v - e + f = 2,$$

gdzie v – liczba wierzchołków, e – liczba krawędzi, f – liczba ścian. Ze wzoru Eulera otrzymujemy górne oszacowanie liczby krawędzi w grafie planarnym:

$$\text{dla } v \geq 3 \text{ zachodzi } e \leq 3v - 6,$$

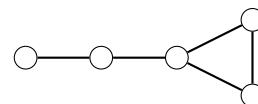
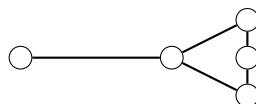
z którego wynika, że na przykład graf K_5 jest nieplanarny.

Dla grafów niezawierających trójkątów (cykli długości 3) prawdziwa jest jeszcze mocniejsza teza:

$$e \leq 2v - 4,$$

z której wiadomo, że graf dwudzielny $K_{3,3}$ także jest nieplanarny. Z powyższych wzorów można też wywnioskować, że **każdy graf planarny zawiera wierzchołek stopnia ≤ 5** , co stanowi warunek konieczny planarności.

Grafy G i H nazwiemy **homeomorficznymi**, jeśli można je uczynić izomorficznymi poprzez dostawianie wierzchołków na ich krawędziach. Dla przykładu, poniższe dwa grafy są homeomorficzne.



Warunek konieczny i wystarczający istnienia płaskiego włożenia grafu opisuje **twierdzenie Kuratowskiego**: graf jest nieplanarny wtedy i tylko wtedy, gdy **zawiera podgraf homeomorficzny z K_5 lub $K_{3,3}$** .

To było na egzaminie

Dany jest graf G o $n \leq 6$ wierzchołkach i m krawędziach. Wówczas

- A. G lub dopełnienie G jest planarne
- B. jeśli $m < 10$, to G jest planarny
- C. jeśli $m > 10$, to G nie jest planarny

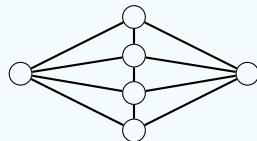
Rozwiążemy podpunkt A. Dla $n \leq 4$ jasne jest, że G jest planarny (twierdzenie Kuratowskiego). Dla $n = 5$

jedynym istniejącym grafem nieplanarnym G jest K_5 , ale wtedy dopełnienie G jest planarne. Rozpatrzmy $n = 6$ i założmy, że G jest nieplanarny. Z twierdzenia Kuratowskiego wynika, że G zawiera podgraf homeomorficzny z K_5 lub $K_{3,3}$.

Podgraf homeomorficzny z K_5 ma 5 wierzchołków stopnia 4. Jeśli podgrafem homeomorficznym w G jest K_5 , to dopełnienie G ma przynajmniej 5 wierzchołków stopnia < 2, więc na podstawie twierdzenia Kuratowskiego dopełnienie G jest planarne.

Z drugiej strony, jeśli G zawierałoby podgraf homeomorficzny z $K_{3,3}$, to skoro G ma 6 wierzchołków, musi być $G = K_{3,3}$ i wtedy dopełnienie G jest planarne. Podpunkt **A.** jest więc prawdziwy.

Do pozostałych podpunktów nietrudno znaleźć kontrprzykłady: dla podpunktu **B.** wystarczy wziąć $G = K_{3,3}$, a dla podpunktu **C.** graf z rysunku poniżej ($n = 6, m = 11$). Oba te stwierdzenia są więc fałszywe.



■ Kolorowanie grafów

Kolorowanie grafu G za pomocą k kolorów to funkcja $f : V_G \rightarrow \{1, \dots, k\}$ przyporządkowująca kolory wierzchołkom w taki sposób, że każde dwa wierzchołki połączone krawędzią są w różnych barwach.

Najmniejsze k , dla którego istnieje k -kolorowanie G , nazywamy **liczbą chromatyczną** $\chi(G)$. Znalezienie jej jest w ogólnym przypadku trudne.

Istnieje szereg twierdzeń pomagających znaleźć liczbę chromatyczną. Jednym z nich jest **twierdzenie o czterech barwach**:

$$\text{jeśli graf jest planarny, to } \chi(G) \leq 4.$$

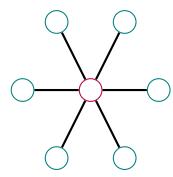
Ponadto, z prostej obserwacji wynika, że

$$\chi(G) = 2 \Leftrightarrow G \text{ jest dwudzielny.}$$

Łącząc powyższe informacje, otrzymujemy wniosek, że graf planarny niebędący grafem dwudzielnym jest zawsze albo 3-, albo 4-kolorowalny. Okazuje się jednak, że sprawdzenie tej informacji dla danego grafu jest problemem NP-trudnym.

Innym narzędziem pozwalającym ograniczyć liczbę chromatyczną z góry jest **twierdzenie Brooksa**. Oznaczmy jako $\Delta(G)$ maksymalny stopień wierzchołka w grafie G . Jeśli spójny graf G nie jest cyklem nieparzystej długości ani kliką, to $\chi(G) \leq \Delta(G)$.

Ale uwaga: to oszacowanie może być bardzo niedokładne, na przykład gwiazda (rysunek obok) ma $\chi(G) = 2$ i dowolnie duże $\Delta(G)$ (wystarczy dokładać wierzchołki na brzegu i łączyć je z środkowym).



Wyróżniamy także **kolorowanie krawędziowe**, czyli funkcję $f : E_G \rightarrow \{1, \dots, k\}$, taką że incydentne krawędzie są różnego koloru. Analogicznie, **indeks chromatyczny** $\chi_e(G)$ to najmniejsze k , dla którego istnieje k -kolorowanie krawędziowe.

Po krótkim rozumowaniu łatwo znaleźć dolne oszacowanie indeksu chromatycznego: $\chi_e(G) \geq \Delta(G)$, jednakże o dodatkowym, nieoczywistym fakcie mówi **twierdzenie Vizinga**:

$$\chi_e(G) \leq \Delta(G) + 1.$$

W związku z tym **indeks chromatyczny** $\chi_e(G)$ jest zawsze równy albo $\Delta(G)$, albo $\Delta(G) + 1$ (dla dowolnego G). Mimo to rozstrzygnięcie, który wariant jest prawdziwy dla danego grafu, jest w ogólnym przypadku trudne. Jeśli jednak graf G jest dwudzielny, to $\chi_e(G) = \Delta(G)$.

Zestaw zadań

4.11. Graf G ma cykl Eulera, ale nie ma cyklu Hamiltona. Wynika z tego, że

- A. dopełnienie grafu G ma cykl Hamiltona
- B. G ma ścieżkę Hamiltona
- C. G ma więcej niż jedną dwuspójną składową

4.12. Każdy graf nieplanarny o n wierzchołkach

- A. zawiera podgraf $K_{3,3}$ lub K_5
- B. ma co najmniej $3n - 5$ krawędzi
- C. ma liczbę chromatyczną nie mniejszą niż 5

4.13. Graf 100-wierzchołkowy G ma liczbę chromatyczną 3. Wynika z tego, że graf G

- A. jest planarny
- B. zawiera cykl
- C. zawiera niezależny zbiór wierzchołków rozmiaru 34

4.14. Dany jest k -regularny graf o n wierzchołkach i m krawędziach. Wtedy

- A. k jest parzyste lub n jest parzyste
- B. k jest dzielnikiem n
- C. k jest dzielnikiem m

4.15. G jest grafem spójnym o $n > 2$ wierzchołkach i takim, że każda jego krawędź należy do pewnego cyklu prostego. Wynika z tego, że

- A. G ma cykl Hamiltona
- B. graf otrzymany przez usunięcie z G jednego wierzchołka jest spójny
- C. każde dwa różne wierzchołki w G są połączone przynajmniej dwiema krawędziowo rozłącznymi ścieżkami

4.16. Graf G ma $n > 2$ wierzchołków i n krawędzi. Wynika z tego, że graf G

- A. jest spójny
- B. zawiera cykl
- C. jest planarny

4.17. G jest n -wierzchołkowym grafem spójnym regularnym stopnia 3. Wynika z tego, że

- A. n jest parzyste
- B. G jest planarny
- C. liczba chromatyczna $\chi(G) \leq 4$

4.6.

Teoria liczb

Na wstępie, przypomnijmy sobie podstawowe definicje i notacje związane z podzielnością liczb.

- Dla $b > 0$ i dowolnego a istnieją jednoznacznie wyznaczone q (**iloraz**) i $0 \leq r < b$ (**reszta**) takie, że $a = b \cdot q + r$.

- **Największy wspólny dzielnik** (NWD) liczb a, b to liczba s taka, że $s \mid a, b$ oraz jeśli $d \mid a, b$, to także $d \mid s$. NWD(a, b) to najmniejszy dodatni element zbioru $\{ax + by : x, y \in \mathbb{Z}\}$.
- Jeśli NWD(a, b) jest równe 1, to a i b są **względnie pierwsze** ($a \perp b$).
- Znane jest **podstawowe twierdzenie arytmetyki**: każda liczba $a > 0$ ma rozkład $a = \prod_{i=1}^n p_i$, gdzie p_i to **czynniki pierwsze**, który jest jednoznaczny z dokładnością do kolejności elementów.

Przykład 1.

Niech $a = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ oraz $b = p_1^{\beta_1} p_2^{\beta_2} \cdots p_k^{\beta_k}$, gdzie p_i to różne czynniki pierwsze. Wtedy

- $b \mid a \iff \forall_i \beta_i \leq \alpha_i$
- $b \perp a \iff \forall_i \min(\alpha_i, \beta_i) = 0$
- $\text{NWD}(a, b) = \prod_{i=1}^k p_i^{\min(\alpha_i, \beta_i)}$
- $\text{NWW}(a, b) = \frac{ab}{\text{NWD}(a, b)} = \prod_{i=1}^k p_i^{\max(\alpha_i, \beta_i)}$

To było na egzaminie

Dla dowolnych dodatnich liczb całkowitych a i b zachodzi implikacja

- A. $a^2 \mid b^2 \Rightarrow a^3 \mid b^3$
 B. $a^2 \mid b^3 \Rightarrow a \mid b$
 C. $a^3 \mid b^2 \Rightarrow a \mid b$

- A. Skoro a^2 dzieli b^2 , to możemy zapisać, że $b^2 = ka^2$ dla pewnego $k \in \mathbb{Z}$. Stąd $k = \frac{b^2}{a^2} = (\frac{b}{a})^2$. Widzimy, że a musi być dzielnikiem b , ponieważ k ma być całkowite. Skoro $a \mid b$, to również $a^3 \mid b^3$, więc odpowiedź to TAK.
- B. Weźmy $a = 2^3$ i $b = 2^2$. Oczywiście $a^2 = 2^6 = b^3$, więc $a^2 \mid b^3$, ale $a \nmid b$, więc odpowiedź to NIE.
- C. Podobnie jak w podpunkcie A., zapiszmy daną relację jako $b^2 = ka^3$ dla pewnego $k \in \mathbb{Z}$. Stąd $ak = (\frac{b}{a})^2$. Skoro ak jest liczbą całkowitą (jako iloczyn liczb całkowitych), to $\frac{b}{a}$ również musi być całkowite, więc $a \mid b$ i odpowiedzią jest TAK.

Algorytm Euklidesa

Algorytm Euklidesa pozwala efektywnie wyznaczyć NWD(a, b) oraz x, y takie, że $\text{NWD}(a, b) = ax + by$:

- jeśli $b = 0$, to $\langle x, y \rangle \leftarrow \langle 1, 0 \rangle$
- wpp. mamy x', y' takie, że $\text{NWD}(a, b) = \text{NWD}(b, a \bmod b) = bx' + (a \bmod b)y'$, ale $a \bmod b = a - b \cdot \lfloor \frac{a}{b} \rfloor$, więc można przyjąć $\langle x, y \rangle \leftarrow \langle y', x' - y' \cdot \lfloor \frac{a}{b} \rfloor \rangle$

Przykład 2.

Znajdziemy x, y , dla których $\text{NWD}(36, 15) = 36x + 15y$:

a	b	x	y
36	15	$\downarrow \uparrow$	-2 5
15	6	$\downarrow \uparrow$	1 -2
6	3	$\downarrow \uparrow$	0 1
3	0	\rightarrow	1 0

Na początku stosujemy wzór $\text{NWD}(a, b) = \text{NWD}(b, a \bmod b)$ tak długo, dopóki nie otrzymamy 0 w drugiej kolumnie. Następnie korzystamy z pierwszego przypadku i przypisujemy wartości $\langle x, y \rangle \leftarrow \langle 1, 0 \rangle$. Idąc do góry, stosujemy wzór $\langle x, y \rangle \leftarrow \langle y', x' - y' \cdot \lfloor \frac{a}{b} \rfloor \rangle$. Ostatecznie otrzymujemy

$$\text{NWD}(36, 15) = 36 \cdot (-2) + 15 \cdot 5 = 3$$

Kongruencje

Dla $n > 0$ mówimy, że a **przystaje do b modulo n** ($a \equiv b \pmod n$), gdy $n \mid a - b$, czyli a i b dają tę samą resztę z dzielenia przez n .

Własności kongruencji:

- $\equiv (\pmod n)$ jest relacją równoważności, jako zbiór reprezentantów jej klas abstrakcji można wziąć $Z_n = \{0, \dots, n-1\}$
- kongruencje można dodawać, odejmować, mnożyć i potęgować stronami (jak równania)
- kongruencję $a \equiv b \pmod n$ można podzielić obustronnie przez d tylko, gdy $d \perp n$
- $ad \equiv bd \pmod{nd} \Leftrightarrow a \equiv b \pmod n$

Jeśli $n_1 \perp n_2$, to układ dwóch kongruencji

$$\begin{cases} a \equiv b \pmod{n_1} \\ a \equiv b \pmod{n_2} \end{cases}$$

jest równoważny pojedynczej kongruencji $a \equiv b \pmod{n_1 n_2}$. Na tym prostym spostrzeżeniu bazuje dużo silniejsze **chińskie twierdzenie o resztach**:

Niech $n = n_1 \cdots n_k$, gdzie n_i parami względnie pierwsze. Wtedy dla dowolnych a_1, \dots, a_k istnieje dokładnie jedno $a \in \{0, \dots, n-1\}$ takie, że

$$a \equiv a_i \pmod{n_i}, \text{ dla } i \in \{1, \dots, k\}$$

Przykład 3.

Niech $\langle n_1, n_2, n_3 \rangle = \langle 7, 11, 13 \rangle$ i wtedy $n = 1001$. Weźmy $\langle a_1, a_2, a_3 \rangle = \langle 5, 3, 11 \rangle$ i znajdźmy a takie, że a daje resztę $a_i \pmod{n_i}$ dla $i = 1, 2, 3$.

Znajdujemy m_i takie, że $n_j \mid m_i$ dla $j \neq i$ oraz $m_i \perp n_i$ (na przykład $m_i = \frac{n}{n_i}$) i przyjmujemy

$$a = \sum_i (a_i m_i (m_i^{-1} \pmod{n_i})) \pmod{n},$$

gdzie $m^{-1} \pmod{n}$ to odwrotność modularna, czyli takie x , że $mx \equiv 1 \pmod{n}$. Odwrotność modularną łatwo znaleźć za pomocą algorytmu Euklidesa jako współczynnik przy m_i w wyrażeniu $m_i x + n_i y = \text{NWD}(m_i, n_i) = 1$. U nas

$$a = (5 \cdot 143 \cdot 5 + 3 \cdot 91 \cdot 4 + 11 \cdot 77 \cdot 12) \pmod{1001} = 817$$

Inne, często wykorzystywane przy kongruencjach twierdzenie to **małe twierdzenie Fermata**: jeśli p jest liczbą pierwszą i $p \nmid a$, to $a^{p-1} \equiv 1 \pmod p$.

Funkcja Eulera

Niech $Z_n^* = \{1 \leq k \leq n : k \perp n\}$, czyli zbiór liczb całkowitych dodatnich niewiększych niż n i względnie pierwszych z n . **Funkcja Eulera** $\Phi : \mathbb{N} \rightarrow \mathbb{N}$ jest określona jako $\Phi(n) = |Z_n^*|$, czyli liczba liczb mniejszych od n względnie pierwszych z n .

Własności:

- jeśli p jest pierwsza, to $\Phi(p^k) = p^k - p^{k-1}$
- multiplikatywność: jeśli $m \perp n$, to $\Phi(mn) = \Phi(m)\Phi(n)$
- wzór iloczynowy: $\Phi(n) = n \prod_{p|n} (1 - \frac{1}{p})$, gdzie p jest pierwsza

Funkcja Eulera pozwala nam uogólnić małe twierdzenie Fermata – w ten sposób powstało **twierdzenie Eulera**: jeśli $a \perp n$, to $a^{\Phi(n)} \equiv 1 \pmod{n}$.

Przykład 4.

Obliczymy $2^{65536} \pmod{2009}$ za pomocą twierdzenia Eulera. Mamy $2009 = 7^2 \cdot 41$, więc korzystając z własności funkcji Eulera:

$$\Phi(2009) = \Phi(7^2)\Phi(41) = 42 \cdot 40 = 1680$$

Stąd dostajemy

$$2^{1680} \equiv 1 \pmod{2009}$$

Znajdziemy teraz iloraz i resztę z dzielenia 65536 przez 1680:

$$65536 = 1680 \cdot q + r = 1680 \cdot 39 + 16$$

To pozwoli nam dojść do szukanego wyrażenia – podnosimy kongruencję stronami do potęgi $q = 39$, a następnie przemnażamy stronami przez $2^r = 2^{16}$:

$$2^{1680 \cdot 39 + 16} \equiv 2^{16} \pmod{2009}$$

stąd

$$2^{65536} \equiv 65536 \equiv 1248 \pmod{2009}$$

Trzeba jednak pamiętać, że twierdzenia Eulera możemy używać jedynie, gdy $a \perp n$. W naszym przypadku $n = 7^2 \cdot 41$, więc oczywiście jest względnie pierwsze z $a = 2$.

Zestaw zadań

4.18. Liczba $p > 2$ jest pierwsza. Wynika z tego, że

- A. $(p-2)^{p-1} - 1$ dzieli się przez p
- B. $(p-2)^{p^2-1} - 1$ dzieli się przez p^2
- C. $(p-2)^{p^3-p^2} - 1$ dzieli się przez p^3

4.19. Jeśli a, b są dodatnimi liczbami całkowitymi, to $\text{NWD}(a, b)$ jest najmniejszym dodatnim elementem zbioru

- A. $\{ax + by \mid x, y \in \mathbb{Z}\}$
- B. $\{a(x+y) + b(x-y) \mid x, y \in \mathbb{Z}\}$
- C. $\{2ax + 3by \mid x, y \in \mathbb{Z}\}$

4.20. Dane są dodatnie liczby całkowite a, b , których największy wspólny dzielnik jest równy d . Wówczas względnie pierwsze są liczby

- A. $\frac{a}{d}$ oraz b

- B. $\frac{ab}{d^2}$ oraz b
 C. $\frac{a^2}{d^2}$ oraz $\frac{b^2}{d^2}$

4.7. Asymptotyka

Notacja asymptotyczna pozwala ukryć zbędne nadmiarowe informacje, uprościć operowanie wyrażeniami i ocenić przybliżone graniczne zachowanie.

Jeśli $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$, to zapisujemy $f = O(g)$, gdy dla pewnej stałej c i dostatecznie dużych n zachodzi

$$f(n) \leq c \cdot g(n)$$

Zapis $f = O(g)$ czytamy jako „ f jest co najwyżej rzędu g ” albo „ f rośnie co najwyżej tak szybko, jak g ”.

Inne stosowane oznaczenia:

- $f = \Omega(g) \Leftrightarrow g = O(f)$ „ f rośnie co najmniej tak szybko, jak g ”
- $f = \Theta(g) \Leftrightarrow f = O(g) \wedge g = O(f)$ „ f rośnie tak samo szybko, jak g ”
- $f = o(g) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ „ f rośnie istotnie wolniej niż g ”
- $f = \omega(g) \Leftrightarrow g = o(f) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ „ f rośnie istotnie szybciej niż g ”
- $f \sim g \Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$ „ f jest asymptotycznie równe g ”

Definicja notacji asymptotycznej jest ściśle powiązana z definicją granicy. Istotnie, jeśli wartość $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ jest określona, możemy z niej wywnioskować relację asymptotyczną między f i g :

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \neq \infty &\Rightarrow f = O(g), & \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \neq 0 &\Rightarrow f = \Omega(g), \\ \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \neq 0, \infty &\Rightarrow f = \Theta(g) \end{aligned}$$

Powszechnie znane są także zależności asymptotyczne między charakterystycznymi funkcjami. Oznaczmy $f < g$, jeśli $f = o(g)$. Wtedy dla dowolnych $a \in (0, 1)$ i $b > 1$ zachodzi:

$$\frac{1}{b^n} < \frac{1}{n^b} < 1 < (\log \log n)^b < \log n < n^a < n < n \log n < n^b < b^n < n!$$

■ Twierdzenie o rekurencji uniwersalnej

Równania rekurencyjne typu „**dziel i zwyciężaj**” są postaci

$$T(n) = a \cdot T\left(\left\lfloor \frac{n}{b} \right\rfloor\right) + f(n),$$

gdzie f jest funkcją nieujemną oraz $a, b \in \mathbb{R}^+$. Tak zdefiniowana funkcja T stanowi pewien schemat działania algorytmów typu „dziel i zwyciężaj” – problem o rozmiarze n dzielony jest na a podproblemów, każdy wielkości $\lfloor \frac{n}{b} \rfloor$, funkcja f przedstawia koszt dzielenia problemu oraz połączenia rozwiązań podproblemów.

W ogólnej postaci powyższego równania można **zignorować podlogi i sufity** (łatwo pokazać, że asymptotycznie nie zmieni to rozwiązania).

Takie równania łatwo rozwiązujemy, korzystając z **twierdzenia o rekurencji uniwersalnej**:

- jeśli $f(n) = O(n^{\log_b a - \varepsilon})$ dla pewnej stałej $\varepsilon > 0$, to $T(n) = \Theta(n^{\log_b a})$
- jeśli $f(n) = \Theta(n^{\log_b a})$, to $T(n) = \Theta(n^{\log_b a} \log n)$
- jeśli $f(n) = \Omega(n^{\log_b a + \varepsilon})$ dla pewnej stałej $\varepsilon > 0$ i jeżeli $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ dla pewnej stałej $c \in (0, 1)$ i dostatecznie dużych n , to $T(n) = \Theta(f(n))$

Intuicyjnie, każdy z trzech przypadków rekurencji uniwersalnej sprowadza się do stwierdzenia, która z funkcji $n^{\log_b a}$ i f jest „większa”. Gdy znana jest odpowiedź na to pytanie, automatycznie znane jest asymptotyczne ograniczenie danej rekursji – jest nią owa „większa” funkcja.

Przykład 1.

Koszt rozwiązania algorytmu wyszukiwania binarnego opisuje równanie

$$T(n) = T\left(\frac{n}{2}\right) + O(1).$$

Korzystając z twierdzenia o rekurencji uniwersalnej, odczytujemy $a = 1, b = 2$ i wtedy $f(n) = \Theta(n^{\log_2 1}) = \Theta(1)$. Stąd otrzymujemy rozwiązanie $T(n) = O(\log n)$.

Zestaw zadań

4.21. Funkcje $f, g : \mathbb{N} \rightarrow \mathbb{R}$ przyjmują wyłącznie wartości dodatnie i są niemalejące. Wynika z tego, że

- A. jeśli $f = O(g)$ i $g = O(f)$, to istnieje skończona granica $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$
- B. jeśli $f = O(g)$, to $f/g = O(1)$
- C. $f = O(g)$ lub $g = O(f)$

4.22. Rozwiązaniem równania rekurencyjnego $T(n) = 2T(\lfloor n/2 \rfloor) + f(n)$ dla $n > 1, T(1) = 0$, jest

- A. $T(n) = \Theta(\log n)$, jeśli $f(n) = O(1)$
- B. $T(n) = \Theta(n)$, jeśli $f(n) = \Theta(\sqrt{n})$
- C. $T(n) = \Theta(n^2)$, jeśli $f(n) = \Theta(n^2)$

4.8.

Rozwiązańia

Rozwiązańia

4.1. Niech f oznacza permutację $\langle 3, 6, 1, 4, 2, 5 \rangle$ i niech f^k oznacza k -krotne złożenie permutacji f . Wynika z tego, że

- NIE** A. $f^8 = \langle 1, 2, 3, 4, 5, 6 \rangle$
- TAK** B. $f^7 = f$
- TAK** C. $f^4 = f^{16}$

Rozważmy zapis cyklowy danej permutacji: $[1, 3][2, 6, 5][4]$. Rozmiary tych cykli to kolejno 2, 3, 1. To oznacza, że $f^6 = id$, gdzie id to permutacja identycznościowa. Z tego powodu dla dowolnego k mamy $f^k = f^{k+6}$ i po odpowiednim podstawieniu łatwo uzyskujemy odpowiedzi do zadania.

4.2. Jeśli ponumerujemy permutacje zbioru $\{1, 2, 3, 4, 5\}$ od 1 do 120 w porządku leksykograficznym, to permutację o numerze 60 będzie

NIE A. $\langle 3, 5, 1, 2, 4 \rangle$

NIE B. $\langle 3, 4, 1, 2, 5 \rangle$

TAK C. $\langle 3, 2, 5, 4, 1 \rangle$

Jak łatwo zauważyc, permutacji z określona liczbą na pierwszym miejscu jest dokładnie $4! = 24$. Może my zatem określić, że permutacje z numerami 1-24 mają jako pierwszą liczbę „1”, numery 25-48 – liczbę „2”, a numery 49-72 – liczbę „3”. Wiemy również, że permutacji z określonymi liczbami na pierwszych dwóch miejscach jest dokładnie $3! = 6$. Zatem możemy określić, że permutacje z numerami 49-54 mają jako początek $\langle 3, 1 \rangle$, a numery 55-60 – początek $\langle 3, 2 \rangle$. Stąd wniosek, że sześćdziesiątą permutacją będzie największa w porządku leksykograficznym permutacja o początku $\langle 3, 2 \rangle$ i jest nią permutacja $\langle 3, 2, 5, 4, 1 \rangle$.

4.3. Liczba permutacji liczb $(1, \dots, 2021)$ takich, że „1” jest w cyklu parzystej długości, jest równa

NIE A. $2021!/2$

TAK B. $1010 \cdot 2020!$

NIE C. $1011 \cdot 2020!$

Szukane permutacje w postaci superzapisu mają liczbę „1” na $(2k + 1)$ -szym miejscu od końca dla $0 \leq k \leq 1009$. Tworząc takie zapisy, najpierw na $2020!$ sposobów układamy liczby $(2, \dots, 2021)$, a następnie na jeden z 1010 sposobów wybieramy, gdzie znajdzie się liczba „1”. Takich permutacji jest więc dokładnie $1010 \cdot 2020!$, stąd tylko odpowiedź **B.** jest poprawna.

4.4. Niech $A(x)$ będzie funkcją tworzącą ciągu $\langle a_n \rangle_{n \in \mathbb{N}}$.

TAK A. $\frac{A(x)}{1-x}$ jest funkcją tworzącą ciągu $\langle a_0 + a_1 + \dots + a_n \rangle_{n \in \mathbb{N}}$

NIE B. $A^2(x)$ jest funkcją tworzącą ciągu $\langle a_n^2 \rangle_{n \in \mathbb{N}}$

NIE C. $xA(x)$ jest funkcją tworzącą ciągu $\langle a_{n+1} \rangle_{n \in \mathbb{N}}$

A. Splot dowolnego ciągu z ciągiem samych jedynek tworzy ciąg sum częściowych (łatwo to pokazać, korzystając wprost z definicji splotu). Odpowiedź jest więc prawdziwa.

B. Tożsamość nie jest prawdziwa, z definicji splotu otrzymujemy

$$A(x)A(x) \leftrightarrow \left\langle \sum_k a_k a_{n-k} \right\rangle_{n \in \mathbb{N}}$$

a to nie musi być $\langle a_n^2 \rangle_{n \in \mathbb{N}}$.

C. Nie, ponieważ $xA(x) \leftrightarrow \langle 0, a_0, a_1, \dots \rangle$ – to przesunięcie w prawo, a nie w lewo.

4.5. Funkcją tworzącą $A(x)$ ciągu $\langle (n+1)2^n \rangle_{n \in \mathbb{N}}$ jest

NIE A. $\frac{1}{(1-2x)(1-x)}$

NIE B. $\frac{d}{dx} \frac{1}{1-2x}$

NIE C. $\int_0^x \frac{dt}{1-2t}$

A. Zauważmy, że wyrażenie można rozpisać jako iloczyn dwóch szeregów geometrycznych:

$$\frac{1}{1-2x} \cdot \frac{1}{1-x} = (1 + 2x + 4x^2 + 8x^3 + \dots)(1 + x + x^2 + x^3 + \dots),$$

a to jest splot funkcji tworzących ciągów $\langle 1, 2, 4, 8, \dots \rangle = \langle 2^n \rangle_{n \in \mathbb{N}}$ oraz $\langle 1, 1, 1, \dots \rangle$. Jak wiemy, splot dowolnego ciągu z ciągiem samych jedynek tworzy ciąg sum częściowych. Stąd

$$\frac{1}{(1-2x)(1-x)} \leftrightarrow \left\langle \sum_{k=0}^n 2^k \right\rangle_{n \in \mathbb{N}}$$

i odpowiedź to **NIE**.

B. Z definicji różniczkowania funkcji tworzącej:

$$\left(\frac{1}{1-2x}\right)' = (1+2x+4x^2+8x^3+\dots)' = 2+8x+24x^2+\dots \leftrightarrow \langle(n+1)2^{n+1}\rangle_{n\in\mathbb{N}}$$

Odpowiedź jest więc fałszywa.

C. Z definicji całkowania funkcji tworzącej:

$$\int_0^x \frac{dt}{1-2t} = \int_0^x (1+2t+4t^2+8t^3+\dots) dt = x + \frac{2}{2}x^2 + \frac{4}{3}x^3 + \frac{8}{4}x^4 + \dots \leftrightarrow \left\langle \frac{2^{n-1}}{n} \right\rangle_{n\in\mathbb{N}}$$

Odpowiedź jest więc fałszywa.

4.6. Niech $A(x)$ będzie funkcją tworzącą (zwykłą) ciągu $\langle a_n \rangle_{n\in\mathbb{N}}$. Wynika z tego, że funkcją tworzącą ciągu

NIE A. $\langle -a_n \rangle_{n\in\mathbb{N}}$ jest $A(-x)$

TAK B. $\langle 2^n a_n \rangle_{n\in\mathbb{N}}$ jest $A(2x)$

TAK C. $\langle \sum_{k=0}^n a_k \rangle_{n\in\mathbb{N}}$ jest $\frac{A(x)}{1-x}$

Odpowiedzi w podpunktach **A.** i **B.** wynikają z własności podstawienia do funkcji tworzących. W podpunkcie **A.** mamy

$$A(-x) \leftrightarrow \langle (-1)^n a_n \rangle_{n\in\mathbb{N}} \neq \langle -a_n \rangle_{n\in\mathbb{N}}$$

Podpunkt **B.** także jest spełniony na podstawie wspomnianej własności.

Prawdziwość podpunktu **C.** wynika z faktu, że splot dowolnego ciągu z ciągiem samych jedynek tworzy串 sum częściowych (a funkcja $\frac{1}{1-x} = 1+x+x^2+\dots$ to funkcja tworząca ciągu samych jedynek).

4.7. Liczba sposobów na umieszczenie $n > 0$ nierozróżnialnych kul w $k > 0$ rozróżnialnych komorach to współczynnik przy wyrazie

TAK A. x^n funkcji $(1-x)^{-k}$

NIE B. x^k funkcji $(1-x)^{-n}$

TAK C. x^{k-1} funkcji $(1-x)^{-n-1}$

Problem z zadania jest analogiczny do problemu wybierania n -kombinacji z powtórzeniami z k -elementowego zbioru. Enumeratorem jest więc

$$(1+x+x^2+\dots)^k = \left(\frac{1}{1-x}\right)^k = (1-x)^{-k},$$

zatem przy podpunkcie **A.** odpowiedzią jest TAK.

Z pomocą metody *stars and bars* możemy również sprawdzić, że szukana w zadaniu liczba sposobów jest równa $\binom{n+k-1}{n}$.

Enumerator z podpunktu **B.** odpowiada umieszczaniu $N = k$ nierozróżnialnych kul w $K = n$ rozróżnialnych komorach, zatem liczba sposobów to $\binom{N+k-1}{N} = \binom{n+k-1}{k} \neq \binom{n+k-1}{n}$, odpowiedź to NIE.

Enumerator z podpunktu **C.** odpowiada umieszczaniu $N = k-1$ nierozróżnialnych kul w $K = n+1$ nierozróżnialnych komorach, zatem liczba sposobów to $\binom{N+k-1}{N} = \binom{n+k-1}{k-1} = \binom{n+k-1}{n}$ i odpowiedź jest prawdziwa.

4.8. Dany jest ciąg $a_n = |\{\langle A, x \rangle : A \subseteq \{1, \dots, n\}, x \in A\}|$. Wtedy

NIE A. $\lim_{n \rightarrow \infty} \frac{a_{n+1}}{a_n} = \infty$

TAK B. a_n jest parzyste dla $n \geq 2$

NIE C. $a_n = n \cdot n!$ dla $n > 0$

Obliczymy kolejne wyrazy ciągu $(a_n)_{n \in \mathbb{N}}$. Chcemy dowiedzieć się, ile jest par składających się ze zbioru i liczby zawartej w tym zbiorze. Liczbę x możemy wybrać na n sposobów. Natomiast liczba zbiorów zawierających x do 2^{n-1} (połowa wszystkich możliwych zbiorów, ponieważ x w 50% przypadków należy do zbioru, a w 50% nie należy).

Mamy więc $a_n = 2^{n-1} \cdot n$, co rozwiązuje podpunkt **C**. W podpunkcie **A**. możemy łatwo zauważać, że granica wynosi 2, a w podpunkcie **B**. że faktycznie dla $n \geq 2$ wyrazy ciągu są parzyste.

4.9. Liczba podzbiorów zbioru 2018-elementowego o mocy co najwyżej 1009 jest równa

NIE A. liczbie podzbiorów zbioru 2018-elementowego o mocy większej niż 1009

NIE B. $\binom{2018}{1009}$

NIE C. 2^{2017}

Obliczymy wymaganą liczbę podzbiorów. Zauważmy symetrię wynikającą z dwumianu Newtona:

$$\binom{2018}{0} = \binom{2018}{2018}, \quad \binom{2018}{1} = \binom{2018}{2017}, \quad \dots$$

Możemy zauważać, że $\binom{2018}{1008} = \binom{2018}{1010}$, natomiast $\binom{2018}{1009}$ jest bez pary, więc **A.** jest fałszywe.

Podpunkt **B.** to liczba podzbiorów o mocy równej 1009, a nie *co najwyżej* 1009 – odpowiedź jest fałszywa.

Przy podpunkcie **C.** możemy zauważać, że 2^{2017} to połowa mocy zbioru potęgowego o 2018 elementach, a – jak pokazaliśmy wyżej przy podpunkcie **A.** – liczba podzbiorów o mocy co najwyżej 1009 jest większa niż połowa, więc odpowiedź jest fałszywa.

4.10. Liczba par (A, B) , takich że $A, B \subseteq \{1, 2, \dots, 2023\}$ oraz

TAK A. $A \subseteq B$ jest równa 3^{2023}

TAK B. $A \cap B = \emptyset$ jest równa 3^{2023}

NIE C. $|A \cup B|$ jest liczbą parzystą, jest równa 3^{2023}

A. Każdy element zbioru $\{1, 2, \dots, 2023\}$ może

- należeć jednocześnie do A i B ,
- należeć wyłącznie do B ,
- nie należeć ani do A , ani do B .

Nietrudno sprawdzić, że takie przyporządkowanie elementów pozwoli nam jednoznacznie zliczyć odpowiednie pary (A, B) . Odpowiedź jest więc prawdziwa – dla każdego z 2023 elementów mamy do podjęcia jeden z trzech wyborów.

B. Rozumowanie analogiczne jak przy podpunkcie **A.**: każdy element zbioru $\{1, 2, \dots, 2023\}$ może

- należeć do A (ale nie do B),
- należeć do B (ale nie do A),
- nie należeć ani do A , ani do B .

Odpowiedź jest prawdziwa.

C. Tym razem każdy element zbioru $\{1, 2, \dots, 2023\}$ może

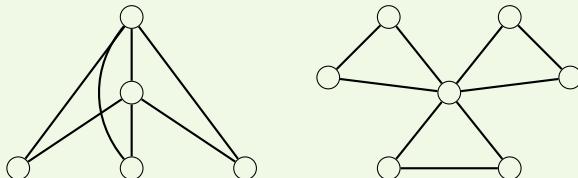
- należeć jednocześnie do A i B ,
- należeć do A (ale nie do B),
- należeć do B (ale nie do A),
- nie należeć ani do A , ani do B .

Interesują nas jednak tylko te przypadki, w których $|A \cup B|$ jest parzyste, lub równoważnie: liczba elementów nienależących ani do A , ani do B jest nieparzysta. Nietrudno sprawdzić, że to dokładnie połowa wszystkich możliwych przypadków (tutaj pominiemy to rozumowanie). Szukaną więc liczbą par (A, B) w podpunkcie **C.** jest $4^{2023}/2 \neq 3^{2023}$.

4.11. Graf G ma cykl Eulera, ale nie ma cyklu Hamiltona. Wynika z tego, że

- NIE** A. dopełnienie grafu G ma cykl Hamiltona
- NIE** B. G ma ścieżkę Hamiltona
- NIE** C. G ma więcej niż jedną dwuspójną składową

Rozważmy następujące grafy spełniające warunki zadania:



- A. W dopełnieniach obydwu grafów ich środkowy wierzchołek będzie odizolowany, więc nie może istnieć tam cykl Hamiltona.
- B. Graf po prawej nie ma ścieżki Hamiltona.
- C. Graf po lewej jest dwuspojny, więc posiada tylko jedną dwuspójną składową – samego siebie.

4.12. Każdy graf nieplanarny o n wierzchołkach

- NIE** A. zawiera podgraf $K_{3,3}$ lub K_5
 - NIE** B. ma co najmniej $3n - 5$ krawędzi
 - NIE** C. ma liczbę chromatyczną nie mniejszą niż 5
- A. Graf nieplanarny jest z K_5 lub $K_{3,3}$ homeomorficzny, ale nie oznacza to, że któryś z nich jest jego podgrafem.
 - B. Kontrprzykładem jest $K_{3,3}$ (9 krawędzi, 6 wierzchołków).
 - C. Wiemy, że jeśli graf jest planarny, to jego liczba chromatyczna jest mniejsza bądź równa 4, ale nie jest to implikacja obustronna. Graf $K_{3,3}$ ma liczbę chromatyczną 2.

4.13. Graf 100-wierzchołkowy G ma liczbę chromatyczną 3. Wynika z tego, że graf G

- NIE** A. jest planarny
- TAK** B. zawiera cykl
- TAK** C. zawiera niezależny zbiór wierzchołków rozmiaru 34

Możemy o takim grafie pomyśleć jak o grafie *trójdzielnym* – dzielimy 100 wierzchołków na 3 zbiory, każdy odpowiadający jednemu kolorowi. Krawędź może wystąpić tylko między wierzchołkami o różnych kolorach (wynika to wprost z definicji kolorowania grafu).

- A. Weźmy oszacowanie na liczbę krawędzi w grafie planarnym: $e \leq 3v - 6$. Wynika z niego, że w grafie G mogą być co najwyżej 294 krawędzie. Weźmy graf, w którym są 33 wierzchołki niebieskie, 33 zielone i 34 czerwone. Połączmy ze sobą wszystkie niebieskie i zielone wierzchołki, otrzymując $33^2 = 1089 > 294$ krawędzi, więc taki graf nie jest planarny.
- B. Gdyby graf G nie zawierał cyklu, jego liczba chromatyczna byłaby równa 2 (moglibyśmy pokolorować wierzchołki dwoma kolorami, na przemian) – sprzeczność z założeniem.
- C. Przy podzieleniu 100 wierzchołków na 3 zbiory, jeden z nich będzie miał moc co najmniej 34 (zasada szuflaapkowa), więc można znaleźć w nim podzbiór 34 niezależnych wierzchołków.

4.14. Dany jest k -regularny graf o n wierzchołkach i m krawędziach. Wtedy

- TAK** A. k jest parzyste lub n jest parzyste

NIE **B.** k jest dzielnikiem n

NIE **C.** k jest dzielnikiem m

- A.** Z definicji k -regularnego grafu wiemy, że z każdego wierzchołka wychodzi k krawędzi. Z lematu o uściskach dloni otrzymujemy równanie $kn = 2m$. Ponieważ m jest liczbą naturalną, k lub n musi być parzyste.
B. Kontrprzykładem jest K_5 .
C. Kontrprzykładem jest K_3 .

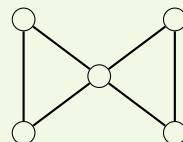
- 4.15.** G jest grafem spójnym o $n > 2$ wierzchołkach i takim, że każda jego krawędź należy do pewnego cyklu prostego. Wynika z tego, że

NIE **A.** G ma cykl Hamiltona

NIE **B.** graf otrzymany przez usunięcie z G jednego wierzchołka jest spójny

TAK **C.** każde dwa różne wierzchołki w G są połączone przynajmniej dwiema krawędziowo rozłącznymi ścieżkami

Rozważmy następujący graf spełniający warunki zadania:



Nietrudno sprawdzić, że nie ma on cyklu Hamiltona, a po usunięciu środkowego wierzchołka graf się rozspójni. Wobec tego odpowiedzi **A.** i **B.** są fałszywe.

Zauważmy, że w grafie G danym w treści zadania nie mogą wystąpić mosty (krawędzie rozspójniające) – gdyby w grafie był most, krawędź ta musiałaby należeć do cyklu prostego, co jest niemożliwe, gdyż po przejściu przez most nie jesteśmy już w stanie „wrócić”, żeby zamknąć cykl. Skoro nie ma mostów, to usunięcie dowolnej krawędzi z grafu go nie rozspójni.

Weźmy więc ścieżkę pomiędzy dowolnymi dwoma wierzchołkami u i v . Usuwając wszystkie krawędzie na ścieżce, graf wciąż jest spójny, więc istnieje druga, rozłączna krawędziowo z pierwszą, ścieżka z u do v . Stąd odpowiedź w **C.** to TAK.

- 4.16.** Graf G ma $n > 2$ wierzchołków i n krawędzi. Wynika z tego, że graf G

NIE **A.** jest spójny

TAK **B.** zawiera cykl

NIE **C.** jest planarny

- A.** Nie musi tak być – weźmy K_4 (4 wierzchołki, 6 krawędzi) i dorzućmy dwa izolowane wierzchołki.
B. Gdyby G nie zawierał żadnego cyklu, to byłby drzewem. Jak wiemy, drzewa to spójne grafy spełniające zależność $|V| = |E| + 1$, więc w naszym przypadku, chcąc dołożyć n -tą krawędź, utworzylibyśmy cykl.
C. Kontrprzykład: bierzemy K_5 i dokładamy tyle izolowanych wierzchołków, żeby ich liczba była równa liczbie krawędzi.

- 4.17.** G jest n -wierzchołkowym grafem spójnym regularnym stopnia 3. Wynika z tego, że

TAK **A.** n jest parzyste

NIE **B.** G jest planarny

TAK **C.** liczba chromatyczna $\chi(G) \leq 4$

- A.** Oznaczmy przez m liczbę krawędzi grafu z zadania. Z lematu o uściskach dloni otrzymujemy zależność $3n = 2m$ i – ponieważ n i m to liczby naturalne – nietrudno zauważyc, że n musi być parzyste.
- B.** Kontrprzykładem jest $K_{3,3}$.
- C.** Ponieważ każdy wierzchołek grafu G jest połączony z dokładnie trzema innymi, mamy pewność, że wystarczą nam 4 barwy do prawidłowego pokolorowania grafu.

4.18. Liczba $p > 2$ jest pierwsza. Wynika z tego, że

- TAK** **A.** $(p - 2)^{p-1} - 1$ dzieli się przez p
- NIE** **B.** $(p - 2)^{p^2-1} - 1$ dzieli się przez p^2
- TAK** **C.** $(p - 2)^{p^3-p^2} - 1$ dzieli się przez p^3

Nietrudno zauważyc, że $(p - 2) \perp p$. Możemy więc skorzystać z małego twierdzenia Fermata, otrzymując $(p - 2)^{p-1} \equiv 1 \pmod{p}$. Zatem podpunkt **A.** jest prawdziwy.

Skoro $(p - 2) \perp p$, to również $(p - 2) \perp p^3$ oraz $\Phi(p^3) = p^3 - p^2$, bo p jest pierwsze. Z twierdzenia Eulera dostajemy $(p - 2)^{p^3-p^2} \equiv 1 \pmod{p^3}$, więc podpunkt **C.** również jest prawdziwy.

Do podpunktu **B.** posłuży nam kontrprzykład: weźmy najmniejszą (sensowną) liczbę pierwszą $p = 5$. Obliczamy:

$$\begin{aligned}(p - 2)^{p^2-1} &\pmod{p^2} \\ (5 - 2)^{25-1} &\pmod{25} \\ 3^{24} &\pmod{25}\end{aligned}$$

Widzimy, że $3^3 \equiv 27 \equiv 2 \pmod{25}$. Podnosimy tę kongruencję stronami do potęgi 8 i dostajemy $3^{24} \equiv 2^8 \equiv 256 \equiv 6 \not\equiv 1 \pmod{25}$. Odpowiedź jest więc fałszywa.

4.19. Jeśli a, b są dodatnimi liczbami całkowitymi, to $\text{NWD}(a, b)$ jest najmniejszym dodatnim elementem zbioru

- TAK** **A.** $\{ax + by \mid x, y \in \mathbb{Z}\}$
- NIE** **B.** $\{a(x + y) + b(x - y) \mid x, y \in \mathbb{Z}\}$
- NIE** **C.** $\{2ax + 3by \mid x, y \in \mathbb{Z}\}$

Prawdziwa odpowiedź do podpunktu **A.** wynika wprost z definicji NWD. W podpunkcie **B.** rozważmy $a = 9, b = 15$ (wtedy $\text{NWD}(a, b) = 3$). Dla pewnych x, y musi zachodzić:

$$9(x + y) + 15(x - y) = 3 \Leftrightarrow 3(x + y) + 5(x - y) = 1 \Leftrightarrow 8x - 2y = 1$$

To prowadzi do sprzeczności, ponieważ po lewej stronie równości znajduje się liczba nieparzysta, a po prawej parzysta.

W podpunkcie **C.** przeprowadzamy taką samą argumentację dla $a = 2, b = 4$ i ponownie uzyskujemy sprzeczność:

$$4x + 12y = 2 \Leftrightarrow 2x + 6y = 1$$

4.20. Dane są dodatnie liczby całkowite a, b , których największy wspólny dzielnik jest równy d . Wówczas względnie pierwsze są liczby

- NIE** **A.** $\frac{a}{d}$ oraz b
- NIE** **B.** $\frac{ab}{d^2}$ oraz b
- TAK** **C.** $\frac{a^2}{d^2}$ oraz $\frac{b^2}{d^2}$

Położmy $a = d \cdot a'$ i $b = d \cdot b'$, gdzie d to NWD(a, b). Oczywiście $a' \perp b'$. Podpunkt **A.** jest fałszywy, bo o ile $a' \perp b'$, to nieprawdą jest, że $a' \perp d$; nie musi tak być chociażby dla $a = 2^2$ i $b = 2 \cdot 3$. Ten przykład pokazuje również, że **B.** nie ma sensu (z tego samego powodu).

Podpunkt **C.** jest jednak prawdziwy, gdyż $\frac{a^2}{d^2} = (a')^2$, $\frac{b^2}{d^2} = (b')^2$, a skoro $a' \perp b'$, to oczywiście $(a')^2 \perp (b')^2$.

- 4.21.** Funkcje $f, g : \mathbb{N} \rightarrow \mathbb{R}$ przyjmują wyłącznie wartości dodatnie i są niemalejące. Wynika z tego, że

NIE **A.** jeśli $f = O(g)$ i $g = O(f)$, to istnieje skończona granica $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$

TAK **B.** jeśli $f = O(g)$, to $f/g = O(1)$

NIE **C.** $f = O(g)$ lub $g = O(f)$

A. Taka granica nie musi istnieć. Kontrprzykładem mogą być funkcje

$$f(n) = n, \quad g(n) = \begin{cases} n & \text{dla } n \text{ nieparzystych}, \\ 2n & \text{dla } n \text{ parzystych.} \end{cases}$$

B. Z definicji notacji asymptotycznej mamy, że dla pewnej stałej c i dostatecznie dużych n zachodzi

$$f(n) \leq c \cdot g(n), \quad \text{a stąd} \quad \frac{f(n)}{g(n)} \leq c.$$

C. Rozważmy funkcje

$$f(n) = \begin{cases} (n-1)^2 & \text{dla } n \text{ nieparzystych}, \\ n^2 & \text{dla } n \text{ parzystych,} \end{cases}$$

$$g(n) = \begin{cases} n^2 & \text{dla } n \text{ nieparzystych}, \\ (n-1)^2 & \text{dla } n \text{ parzystych,} \end{cases}$$

będące ciągami kwadratowymi, przy czym raz rośnie f , a raz g . Wówczas różnica $|f(n) - g(n)|$ wraz ze wzrostem n dąży do nieskończoności oraz $(f(n) - g(n))(f(n+1) - g(n+1)) < 0$.

- 4.22.** Rozwiążaniem równania rekurencyjnego $T(n) = 2T(\lfloor n/2 \rfloor) + f(n)$ dla $n > 1$, $T(1) = 0$, jest

NIE **A.** $T(n) = \Theta(\log n)$, jeśli $f(n) = O(1)$

TAK **B.** $T(n) = \Theta(n)$, jeśli $f(n) = \Theta(\sqrt{n})$

TAK **C.** $T(n) = \Theta(n^2)$, jeśli $f(n) = \Theta(n^2)$

Korzystamy z twierdzenia o rekurencji uniwersalnej. Odczytujemy $a = b = 2$, więc nasze $f(n)$ musimy dopasować do jednego z trzech przypadków:

1. $f(n) = O(n^{\log_b a - \varepsilon}) = O(n^{1-\varepsilon})$,
2. $f(n) = \Theta(n^{\log_b a}) = \Theta(n)$,
3. $f(n) = \Omega(n^{\log_b a + \varepsilon}) = \Omega(n^{1+\varepsilon})$.

A. Skoro $f(n) = O(1)$, to otrzymujemy pierwszy przypadek. Rozwiążaniem równania rekurencyjnego będzie $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$, czyli odpowiedź to **NIE**.

B. Skoro $f(n) = \Theta(\sqrt{n})$, to znów jesteśmy w przypadku pierwszym i rozwiązaniem jest $T(n) = \Theta(n)$, czyli odpowiedź to **TAK**.

C. Jeśli $f(n) = \Theta(n^2)$, to wpadamy do trzeciego przypadku, więc rozwiązaniem jest $T(n) = \Theta(f(n)) = \Theta(n^2)$, czyli **TAK**.

5

Rachunek prawdopodobieństwa

Materiały teoretyczne zostały opracowane na podstawie [notatek Błażeja Wilkoławskiego](#), materiałów na Ważniku oraz [skrypty z Metodyki Nauczania Rachunku Prawdopodobieństwa](#) autorstwa Adama Osękowskiego.

Podstawa programowa

1. **Prawdopodobieństwo warunkowe i całkowite**, wzór Bayesa, niezależność zdarzeń.
2. **Dyskretne zmienne losowe** i ich rozkłady: dwumianowy, geometryczny, Poissona.
3. **Parametry rozkładu**: wartość oczekiwana, wariancja, funkcje tworzące prawdopodobieństwa.
4. **Nierówności probabilistyczne**: Markowa, Czebyszewa, Chernoffa.
5. **Ciągłe zmienne losowe**: definicja, własności, rozkład wykładniczy oraz normalny, centralne twierdzenie graniczne.
6. **Łańcuchy Markowa**: prawdopodobieństwa oraz średnie czasy dotarcia, twierdzenie ergodyczne.
7. **Wnioskowanie statystyczne**: estymatory nieobciążone, estymatory największej wiarygodności.

5.1.

Prawdopodobieństwo warunkowe i całkowite

Doświadczeniem losowym nazwiemy dowolny proces bądź ciąg czynności taki, że:

- jego sposób wykonania i warunki są ściśle określone, a sam proces można dowolnie wiele razy powtarzać,
- zbiór możliwych wyników procesu (**zdarzeń elementarnych**) jest z góry znany,
- wyniku konkretnego doświadczenia nie można z góry przewidzieć.

Zbiór wszystkich zdarzeń elementarnych oznaczamy wielką literą Ω .

W wielu sytuacjach interesuje nas nie tyle konkretny wynik doświadczenia (pojedyncze zdarzenie elementarne), ale to, czy należy on do wcześniej ustalonego podzbioru wszystkich zdarzeń elementarnych. Takie podzbiory nazywamy **zdarzeniami** i zwyczajowo oznaczamy wielkimi literami alfabetu, np. $A, B, X, Y \dots$

Przykład 1.

Dla doświadczenia losowego będącego rzutem kostką, zbiorem zdarzeń elementarnych jest $\Omega = \{1, 2, \dots, 6\}$, a przykładowymi zdarzeniami:

- $A = \{2, 4, 6\}$ – wypadła liczba parzysta
- $B = \{1, 2, 3, 4, 5, 6\}$ – coś wypadło

- $C = \emptyset$ – nic nie wypadło

Uwaga: ponieważ w programie studiów informatycznych nie zajmujemy się innym prawdopodobieństwem niż prawdopodobieństwo klasyczne, pominiemy tutaj formalizmy związane z przestrzeniami probabilistycznymi – nie są one potrzebne przy rozwiązywaniu zadań. Zaznaczymy jedynie, że przestrzenie te służą do definiowania sposobu przypisania poszczególnym zdarzeniom ich prawdopodobieństw.

■ Podstawowe własności prawdopodobieństwa

Niech Ω będzie niepustym zbiorem wszystkich zdarzeń elementarnych doświadczenia losowego, a P – prawdopodobieństwem określonym na podzbiorach Ω . Wtedy:

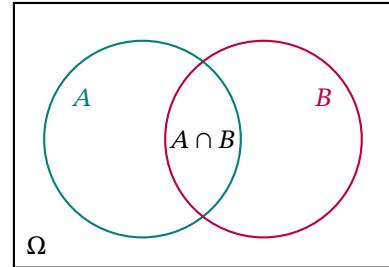
- dla każdego zdarzenia A zachodzi $P(A) \in [0, 1]$, przy czym $P(\emptyset) = 0$ oraz $P(\Omega) = 1$
- dla dowolnych zdarzeń A, B , takich że $A \subseteq B$, jest $P(A) \leq P(B)$
- $P(A') = 1 - P(A)$, gdzie A' to zdarzenie przeciwnie do zdarzenia A
- dla dowolnych zdarzeń A, B mamy

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Przydatnym trikiem jest **wizualizacja geometryczna** przestrzeni probabilistycznej za pomocą **diagramów Venna**, czyli schematów służących ilustrowaniu zależności między zbiorami (w naszym przypadku – między zdarzeniami).

Z rysunku łatwo odczytać na przykład powyższy wzór na prawdopodobieństwo sumy zdarzeń $P(A \cup B)$ oraz wiele innych nieoczywistych relacji, jak np.

$$P(A' \cup B) = 1 - P(A \setminus B)$$



■ Schemat klasyczny

Aby móc wygodnie obliczać prawdopodobieństwa interesujących nas zdarzeń, w większości przypadków będziemy sprowadzać całe rozumowanie do problemu kombinatorycznego za pomocą **schematu klasycznego**.

Załóżmy, że Ω jest zbiorem skończonym i wszystkie zdarzenia elementarne są jednakowo prawdopodobne. Wówczas dla każdego $\omega \in \Omega$ oraz $A \subseteq \Omega$ zachodzi

$$P(\omega) = \frac{1}{|\Omega|} \quad \text{oraz} \quad P(A) = \frac{|A|}{|\Omega|}$$

Przykład 2.

Obliczymy prawdopodobieństwo, że w losowej 5-kartowej ręce pokerowej jest dokładnie jedna para (w szczególności nie ma w niej żadnej trójki ani dwóch par).

W naszym przypadku Ω to zbiór wszystkich 5-elementowych podzbiorów 52 kart, mamy więc $|\Omega| = \binom{52}{5}$. Niech A będzie zbiorem wszystkich układów z dokładnie jedną parą. Ponieważ każdy 5-elementowy podzbiór kart jest równie prawdopodobny, korzystamy ze schematu klasycznego i dostajemy $P(A) = \frac{|A|}{|\Omega|}$.

Sprawdźmy, na ile sposobów można wybrać jedną parę:

1. na 13 sposobów wybieramy rangę kart w parze,
2. na $\binom{4}{2} = 6$ sposobów wybieramy kolory tych kart,
3. na $\binom{12}{3}$ sposobów wybieramy rangi pozostałych kart (muszą być różne),

4. na 4^3 sposobów wybieramy kolory tych kart.

Ostatecznie $|A| = 13 \cdot 6 \cdot \binom{12}{3} \cdot 4^3$ oraz $P(A) \approx 0,423$. Jak widać, użycie schematu klasycznego sprowadziło problem probabilistyczny do problemu kombinatorycznego.

■ Prawdopodobieństwo warunkowe i całkowite

Na początek przytoczymy przykład, który pozwoli nam przybliżyć intuicję idącą za prawdopodobieństwem warunkowym.

Przypuśćmy, że rzucamy dwukrotnie kostką „na ślepo” (nie patrząc na wyniki) i ktoś stojący obok mówi nam, że w sumie wyrzuciliśmy osiem oczek. Jakie jest prawdopodobieństwo, że w pierwszym rzucie kostką uzyskaliśmy dwójkę lub trójkę?

Jasne jest, iż nie dysponując dodatkową informacją o sumie oczek podalibyśmy odpowiedź $\frac{1}{3}$ (dokładnie 12 z 36 możliwych zdarzeń elementarnych spełnia postawiony warunek). Jednak skoro wiemy, że suma oczek wynosi 8, to musiała zajść jedna z pięciu możliwości:

$$(2, 6), (3, 5), (4, 4), (5, 3), (6, 2).$$

Ponieważ w dokładnie dwóch z nich na pierwszej współrzędnej stoi dwójka lub trójka, widzimy, że szukane prawdopodobieństwo wynosi $\frac{2}{5}$.

Ten przykład dobrze pokazuje intuicję stojącą za **prawdopodobieństwem warunkowym** – szukamy prawdopodobieństwa zajścia zdarzenia A pod warunkiem zajścia zdarzenia B , oznaczanego $P(A|B)$.

Skoro zaszło zdarzenie B i interesuje nas zdarzenie A , to w rzeczywistości muszą zajść oba zdarzenia A i B ($A \cap B$). Ponadto, skoro zaszło zdarzenie B , to zbiór B staje się naszą nową, zredukowaną przestrzenią zdarzeń elementarnych (jak w przykładzie). To rozumowanie daje nam naturalną definicję

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Zakładamy przy tym, że $P(B) > 0$.

Warto przytoczyć również **wzór iloczynowy**, pozwalający wyrazić prawdopodobieństwo iloczynu zdarzeń przez iloczyn odpowiednich prawdopodobieństw warunkowych.

Niech $A_1, A_2, \dots, A_n \subseteq \Omega$ oraz $P(A_1 \cap A_2 \cap \dots \cap A_n) > 0$. Wtedy

$$P(A_1 \cap A_2 \cap \dots \cap A_n) = P(A_1) \cdot P(A_2|A_1) \cdot P(A_3|A_1 \cap A_2) \cdot \dots \cdot P(A_n|A_1 \cap \dots \cap A_{n-1})$$

Przykład 3.

Test na obecność koronawirusa ma skuteczność 95%. Wiadomo, że średnio 1 na 1000 osób jest zakażona. Jeśli dla losowo wybranej osoby test dał wynik pozytywny, to jakie jest prawdopodobieństwo, że jest ona faktycznie chora?

Oznaczmy:

- C – wybrana osoba jest chora,
- Z – wybrana osoba jest zdrowa,
- T – test da wynik pozytywny,
- N – test da wynik negatywny.

Naszym celem jest obliczenie $P(C|T)$. Z treści wiemy, że $P(T|C) = 0,95$, $P(N|C) = 0,05$, $P(T|Z) = 0,05$,

$P(N|Z) = 0.95$. Korzystając ze wzoru iloczynowego, możemy znaleźć szukane prawdopodobieństwo:

$$P(C|T) = \frac{P(C \cap T)}{P(T)} = \frac{\underbrace{P(C)P(T|C)}_{P(C \cap T) + P(Z \cap T)}}{P(C)P(T|C) + P(Z)P(T|Z)} = \frac{0.001 \cdot 0.95}{0.001 \cdot 0.95 + 0.999 \cdot 0.05} \approx 0.02.$$

Sposób, w jaki w powyższym przykładzie obliczyliśmy $P(T)$, to szczególny przypadek **twierdzenia o prawdopodobieństwie całkowitym**. Używamy go w szczególności do badania wyników doświadczeń, które składają się z kilku etapów.

Niech $A_1, A_2, \dots, A_n \subset \Omega$ będą takie, że $P(A_i) > 0$ i

- $A_i \cap A_j = \emptyset$ (wszystkie A_i są parami rozłączne),
- $A_1 \cup \dots \cup A_n = \Omega$ (wszystkie A_i pokrywają Ω)

Wówczas dla dowolnego zdarzenia $B \subseteq \Omega$ zachodzi wzór

$$P(B) = P(B|A_1)P(A_1) + \dots + P(B|A_n)P(A_n)$$

Przykład 4.

W urnie znajdują się dwie białe kule i jedna czarna. Losujemy kulę (nie oglądamy jej) i odkładamy na bok. Następnie losujemy drugą kulę. Jakie jest prawdopodobieństwo, że druga kula jest biała?

Opisane doświadczenie składa się z dwóch etapów, będących wylosowaniem odpowiednio pierwszej i drugiej kuli z urny. Są dwa możliwe wyniki pierwszego etapu:

- B_1 – jako pierwszą wylosowano kulę białą,
- C_1 – jako pierwszą wylosowano kulę czarną.

W oczywisty sposób są to rozłączne przypadki oraz pokrywają wszystkie możliwości (nie ma innych kul niż białe i czarne) – spełniają więc założenia twierdzenia o prawdopodobieństwie całkowitym.

Oznaczmy szukane zdarzenie („jako drugą wylosowano kulę białą”) jako B_2 i obliczmy jego prawdopodobieństwo:

$$P(B_2) = P(B_2|B_1) \cdot P(B_1) + P(B_2|C_1) \cdot P(C_1) = \frac{1}{2} \cdot \frac{2}{3} + 1 \cdot \frac{1}{3} = \frac{2}{3}$$

Twierdzenie o prawdopodobieństwie całkowitym pomagało nam przy badaniu wyników doświadczeń wieloetapowych. A co w przypadku, gdy znamy wynik takiego doświadczenia, a pytamy o jego przebieg?

W takich sytuacjach wykorzystujemy **wzór Bayesa**: niech $B, A_1, \dots, A_n \subset \Omega$ będą jak w twierdzeniu o prawdopodobieństwie całkowitym. Dla dowolnego k zachodzi

$$P(A_k|B) = \frac{P(A_k \cap B)}{P(B)} = \frac{P(A_k)P(B|A_k)}{\sum_{i=1}^n P(A_i)P(B|A_i)}$$

Przykład 5.

Kontynuujemy poprzedni przykład z urną i kulami. Założmy, że druga wylosowana kula jest biała. Jakie jest prawdopodobieństwo, że jako pierwszą wyciągnięto kulę czarną?

Przy zachowaniu tych samych oznaczeń zdarzeń, interesuje nas prawdopodobieństwo warunkowe $P(C_1|B_2)$. Korzystając ze wzoru Bayesa, łatwo obliczamy:

$$P(C_1|B_2) = \frac{P(C_1 \cap B_2)}{P(B_2)} = \frac{P(C_1)P(B_2|C_1)}{P(C_1)P(B_2|C_1) + P(B_1)P(B_2|B_1)} = \frac{\frac{1}{3} \cdot 1}{\frac{2}{3}} = \frac{1}{2}$$

Metoda drzewkowa

W sytuacji, gdy badane doświadczenie składa się z większej liczby etapów, warto zilustrować je za pomocą odpowiedniego **drzewa** – w ten sposób, zamiast algebraicznie wyliczać prawdopodobieństwo całego zdarzenia, możemy rozpatrzyć wszystkie możliwe losowania za pomocą rysunku. Ze wzoru iloczynowego wynika, że prawdopodobieństwo znalezienia się w konkretnym wierzchołku jest równe iloczynowi liczb na ścieżce od korzenia do tego wierzchołka.

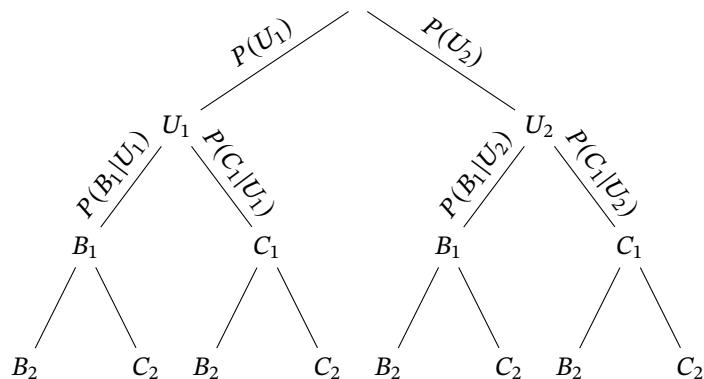
Przykład 6.

Mamy dwie urny z pewną (znaną) liczbą kul białych i czarnych. Wybieramy losowo urnę, a następnie losujemy z niej dwie kule. Jakie jest prawdopodobieństwo, że obie z nich będą białe?

Oznaczmy zdarzenia: U_1, U_2 – wylosowana urna, B_1, C_1 – kolor pierwszej kuli, B_2, C_2 – kolor drugiej kuli. Rozwiążanie algebraiczne wyglądałoby tak:

$$P(B_1 \cap B_2) = P(U_1 \cap B_1 \cap B_2) + P(U_2 \cap B_1 \cap B_2) = P(U_1)P(B_1|U_1)P(B_2|B_1 \cap U_1) + P(U_2)P(B_1|U_2)P(B_2|B_1 \cap U_2)$$

Metoda drzewkowa:



Aby znaleźć $P(B_1 \cap B_2)$ za pomocą powyższego drzewa, sumujemy $P(U_1 \cap B_1 \cap B_2)$ i $P(U_2 \cap B_1 \cap B_2)$, mnożąc liczby na odpowiednich ścieżkach w drzewie.

Schemat Bernoulliego

Schematem Bernoulliego nazywamy ciąg niezależnych powtórzeń tego samego doświadczenia, w którym możliwe są dwa wyniki: jeden z nich nazywamy **sukcesem** (o prawdopodobieństwie p), a drugi **porażką** (o prawdopodobieństwie $1 - p$). Pojedyncze doświadczenie nazywamy **próbą Bernoulliego**.

Taki schemat jest jednoznacznie określony przez podanie liczby prób (oznaczanej n) i prawdopodobieństwa sukcesu p . Można też rozpatrywać schematy Bernoulliego z nieskończoną liczbą prób.

Nietrudno kombinatorycznie pokazać, że prawdopodobieństwo uzyskania dokładnie k sukcesów w schemacie Bernoulliego składającego się z n prób wynosi

$$\binom{n}{k} p^k (1-p)^{n-k}$$

Istotnie, k spośród n prób zakończy się sukcesem (stanie się to z prawdopodobieństwem p^k), pozostałe $n - k$ prób zakończy się porażką (prawdopodobieństwo $(1 - p)^{n-k}$), a na $\binom{n}{k}$ sposobów wybieramy, które k prób będzie pomyślne.

Przykład 7.

Rzucamy 10 razy kostką. Znajdziemy prawdopodobieństwo tego, że szóstka wypadnie raz lub dwa razy.

Oznaczmy poszukiwane zdarzenie jako A . Mamy tu do czynienia ze schematem Bernoulliego składającym się z $n = 10$ prób. Próbą Bernoulliego jest pojedynczy rzut kostką, a sukcesem – wyrzucenie 6 oczek, zatem $p = \frac{1}{6}$. Wobec tego obliczamy

$$P(A) = \underbrace{\binom{10}{1} \left(\frac{1}{6}\right)^1 \left(\frac{5}{6}\right)^9}_{\text{jeden sukces}} + \underbrace{\binom{10}{2} \left(\frac{1}{6}\right)^2 \left(\frac{5}{6}\right)^8}_{\text{dwa sukcesy}}$$

Niezależność zdarzeń

Na ogół, jeśli zdarzenia A, B są takie, że $P(B) > 0$, to prawdopodobieństwo warunkowe $P(A|B)$ różni się od prawdopodobieństwa bezwarunkowego $P(A)$. Innymi słowy, dodatkowa informacja podana za pomocą zdarzenia B w istotny sposób zmienia szansę zajścia zdarzenia A . Czasami jednak to, czy zdarzenie B zaszło czy nie, nie wpływa na prawdopodobieństwo A : mamy wtedy $P(A|B) = P(A)$. W takiej sytuacji mówimy, że A i B są **niezależne**.

Rozpisując wzór na prawdopodobieństwo warunkowe, powyższa równość daje się zapisać w równoważnej postaci stanowiącej definicję niezależności zdarzeń:

$$P(A \cap B) = P(A) \cdot P(B)$$

Jeśli powyższa równość nie zachodzi, to powiemy, że zdarzenia A i B są **zależne**.

O pojęciu niezależności należy więc myśleć w sposób „czy zajście zdarzenia B daje mi dodatkową informację na temat zdarzenia A ?", a nie w terminach związków fizycznych między zdarzeniami.

Przykład 8.

Z talii 52 kart losujemy kartę. Rozważmy zdarzenia:

- A – wylosowano króla,
- B – wylosowano kiera.

Wówczas zdarzenia A i B są niezależne – łatwo sprawdzamy:

$$P(A) = \frac{4}{52}, \quad P(B) = \frac{13}{52} = \frac{1}{4}, \quad P(A \cap B) = \frac{1}{52} = P(A)P(B)$$

To było na egzaminie

Rzucamy symetrycznymi kostkami: sześcienną i ósmiościenną. Niech A będzie zdarzeniem, że na kostce sześcienniej wypadło 6. Zdarzeniami niezależnymi od A są

- A. suma oczek na obu kostkach jest parzysta
- B. suma oczek na obu kostkach jest równa 7
- C. suma oczek na obu kostkach jest większa od 7

Nietrudno zauważyć, że $|\Omega| = 48$ oraz $P(A) = \frac{1}{6}$. W każdym z podpunktów pozostaje nam zatem policzyć $P(A \cap B)$ oraz $P(B)$ (za B oznaczamy zdarzenie z danego podpunktu), a następnie sprawdzić warunek na niezależność zdarzeń.

- A. Oznaczmy B – suma oczek na obu kostkach jest parzysta. Jeśli na pierwszej kostce wypadła parzysta

liczba oczek, to na drugiej również, analogicznie dla nieparzystych. Mamy więc $|B| = 3 \cdot 4 + 3 \cdot 4 = 24$ oraz $P(B) = \frac{24}{48} = \frac{1}{2}$.

Zdarzenia $A \cap B$ spełniają 4 zdarzenia elementarne: (6, 2), (6, 4), (6, 6), (6, 8). W związku z tym

$$P(A \cap B) = \frac{4}{48} = \frac{1}{12} = P(A) \cdot P(B)$$

Zdarzenia A i B są więc niezależne. Intuicyjnie możemy rozumieć to następująco: nieważne, co wypadnie na pierwszej kostce, na drugiej jest zawsze tyle samo opcji.

B. Oznaczmy B – suma oczek na obu kostkach jest równa 7. Nietrudno obliczamy:

$$P(B) = \frac{6}{48}, \quad P(A \cap B) = \frac{1}{48} = P(A) \cdot P(B)$$

A i B ponownie są niezależne. Intuicyjnie: nieważne, co wypadnie na pierwszej kostce, na drugiej jest tylko jedna opcja.

C. Oznaczmy B – suma oczek na obu kostkach jest większa od 7. Wypisując wszystkie zdarzenia elementarne spełniające ten warunek (pominiemy to w tym przykładzie), otrzymujemy

$$P(B) = \frac{27}{48} = \frac{9}{16}, \quad P(A \cap B) = \frac{7}{48} \neq P(A) \cdot P(B)$$

Tym razem zdarzenia A i B są od siebie zależne. Intuicyjnie: zależnie od tego, co wypadnie na pierwszej kostce, na drugiej będziemy mieli różną liczbę opcji.

Niezależność więcej niż dwóch zdarzeń definiujemy na dwa sposoby:

1. Zdarzenia A_1, A_2, \dots, A_n są **niezależne parami**, jeśli każde dwa spośród nich są niezależne.
2. Zdarzenia A_1, A_2, \dots, A_n są **niezależne** (lub *niezależne łącznie*, bądź *niezależne zespołowo*), jeśli dla dowolnego ciągu indeksów $i_1 < i_2 < \dots < i_k$ zachodzi

$$P(A_{i_1} \cap \dots \cap A_{i_k}) = P(A_{i_1}) \cdot \dots \cdot P(A_{i_k})$$

Przykład 9.

Rozważmy dwukrotny rzut sześcienną kostką i zbadajmy niezależność następujących trzech zdarzeń:

- A – suma oczek wynosi 7,
- B – w pierwszym rzucie uzyskano czwórkę,
- C – w drugim rzucie uzyskano trójkę.

Nietrudno sprawdzić, że $P(A) = P(B) = P(C) = \frac{1}{6}$. Zauważmy, że zdarzenia $A \cap B$, $A \cap C$, $B \cap C$ możemy zdefiniować tak samo: „w pierwszym rzucie wypadła czwórka, w drugim trójka”, stąd

$$P(A \cap B) = P(A \cap C) = P(B \cap C) = \frac{1}{36}$$

Widzimy więc, że zdarzenia A, B, C są niezależne parami (każde dwa spośród nich są niezależne).

Z drugiej strony, $A \cap B \cap C$ również definiujemy jako „w pierwszym rzucie wypadła czwórka, w drugim trójka”, więc

$$P(A \cap B \cap C) = \frac{1}{36} \neq P(A) \cdot P(B) \cdot P(C)$$

Zdarzenia A, B, C nie są więc niezależne.

- 5.1.** Istnieje przestrzeń probabilistyczna Ω i zdarzenia $A, B \subseteq \Omega$ takie, że $P(A) = P(B) = \frac{2}{3}$ oraz
- A. A i B są niezależne
 - B. $P(A|B) = \frac{1}{3}$
 - C. $P(A|B) \neq P(B|A)$
- 5.2.** Chcemy rozpalić ognisko, lecz mamy do dyspozycji tylko trzy zapałki. Prawdopodobieństwo rozpalenia ogniska pojedynczą zapałką wynosi 0.4, dwiema złączonymi zapałkami – 0.6, zaś trzema złączonymi zapałkami – 0.8. Żeby zmaksymalizować prawdopodobieństwo rozpalenia ogniska, należy
- A. użyć od razu trzech zapałek
 - B. użyć najpierw dwóch zapałek, a potem jednej
 - C. używać pojedynczych zapałek
- 5.3.** Rzucamy 10 razy symetryczną monetą. Wynika z tego, że
- A. prawdopodobieństwo, że wyrzucimy dokładnie 2 orły jest równe $\frac{45}{1024}$
 - B. prawdopodobieństwo, że wyrzucimy orła w ostatnim rzucie pod warunkiem, że we wszystkich 10 rzutach wyrzucimy dokładnie 1 orła, jest równe $\frac{1}{2}$
 - C. prawdopodobieństwo, że wyrzucimy orła w ostatnim rzucie pod warunkiem, że w pierwszych 9 rzutach wyrzucimy dokładnie 6 orłów, jest równe $\frac{1}{2}$
- 5.4.** Mamy 9 ponumerowanych kart od 1 do 9. Dwie osoby A oraz B losują bez zwracania po jednej karcie. Wygrywa osoba, która wylosuje kartę z większym numerem. Prawdą jest, że
- A. zdarzenia „wygrała osoba A ” oraz „osoba A wylosowała kartę z numerem 5” są niezależne
 - B. zdarzenie „wygrała osoba A pod warunkiem, że wylosowała kartę z numerem 7” ma prawdopodobieństwo $\frac{3}{4}$
 - C. zdarzenia „wygrała osoba A ” oraz „wygrała osoba B ” są niezależne
- 5.5.** Rzucamy symetryczną kostką dwudziestościenną. Niech X będzie liczbą wyrzuconych oczek. Wtedy niezależne są zdarzenia „wyrzucono parzystą liczbę oczek” oraz
- A. wyrzucono liczbę oczek podzielną przez 3
 - B. wyrzucono liczbę oczek podzielną przez 5
 - C. $X \leq 8$

5.2.

Dyskretnie zmienne losowe

W wielu sytuacjach, podczas przeprowadzania eksperymentu losowego, interesuje nas nie tyle konkretny wynik tego doświadczenia, co raczej jakaś charakterystyka liczbową (ściślej: pewna funkcja od wyniku).

Przykładowo, rozważmy stukrotny rzut kostką i założmy, że interesuje nas zbadanie, czy suma wyrzuconych oczek wyniosła 200. Wówczas nie jest ważne, czy potencjalna dwusetka jest osiągana przez taki czy inny ciąg wyników – liczy się tylko wspomniana suma. Wygodnie byłoby określić pewną funkcję $f : \Omega \rightarrow \mathbb{R}$ zwracającą sumę wyrzuconych oczek i zbadać, z jakim prawdopodobieństwem wynik tej funkcji wynosi 200.

Każdą taką funkcję z Ω w \mathbb{R} nazywamy **zmienną losową**. Zazwyczaj zmienne losowe oznaczamy wielkimi literami X, Y, Z, \dots

Przykład 1.

Rozważmy doświadczenie polegające na rzucie dwiema kostkami i sprawdźmy, w jaki sposób zmienne losowe mogą nam pomóc przy badaniu różnych jego własności.

Przykładowo, zmienna losowa opisująca sumę oczek to funkcja $X((i, j)) = i + j$, podobnie można utworzyć funkcję dla iloczynu oczek, oczek na jednej z kostek itd.

Zmiennej losowej można użyć do definiowania zdarzeń w jej dziedzinie, np.:

- $X = 5$ (suma oczek wynosi 5)
- $X \leq 4$ (suma oczek nie przekracza 4)
- $X \in \{3, 5, 7, 9, 11\}$ (suma oczek jest nieparzysta)

Możemy też definiować zdarzenia za pomocą kilku zmiennych, np. jeśli Y to iloczyn oczek, to zdarzenie $X < Y$ oznacza sumę oczek mniejszą od iloczynu.

Niezależność zmiennych losowych

Zmienne losowe X, Y są **niezależne**, gdy dla każdych $x, y \in \mathbb{R}$ zachodzi

$$P(X = x \wedge Y = y) = P(X = x)P(Y = y)$$

Przykład 2.

Rozważmy dwukrotny rzut kostką oraz zmienne losowe:

- X – wynik pierwszego rzutu,
- Y – wynik drugiego rzutu,
- Z – suma liczb z obu rzutów.

Wówczas zmienne X i Y są niezależne: dla każdych $i, j \in \{1, \dots, 6\}$ mamy

$$P(X = i \wedge Y = j) = P(X = i) \cdot P(Y = j)$$

Natomiast zmienne X i Z nie są niezależne, bo na przykład

$$P(X = 1 \wedge Z = 12) = 0, \quad \text{ale} \quad P(X = 1) \cdot P(Z = 12) = \frac{1}{6} \cdot \frac{1}{36}$$

Widzimy więc, że intuicja stojąca za niezależnością zmiennych losowych jest bardzo podobna jak ta, z którą mamy do czynienia przy niezależności zdarzeń.

Rozkład zmiennej losowej

Jeśli interesują nas własności zmiennej losowej X , to dziedzina oraz to, jak konkretnie X jest zdefiniowana, nie mają dla nas znaczenia. Cała istotna informacja o zmiennej X jest zawarta w jej **rozkładzie**.

Rozkład zmiennej losowej X to zbiór par (x_i, p_i) , gdzie x_i jest wartością zmiennej losowej X , a p_i jest prawdopodobieństwem, że zmienność X przyjmuje wartość x_i .

Jeśli X i Y mają ten sam rozkład, piszemy $X \sim Y$.

Rozkład dyskretny zmiennej X zachodzi, jeśli $\sum_{x \in \mathbb{R}} P(X = x) = 1$, czyli z prawdopodobieństwem 1 zmienność X przyjmuje jedną z przeliczalnie wielu wartości. Mówimy wtedy, że zmienność X jest dyskretna.

Przykład 3.

Rozważmy trzykrotny rzut monetą i niech X oznacza łączną liczbę orłów uzyskanych w tym doświadczeniu. Znajdziemy rozkład zmiennej X .

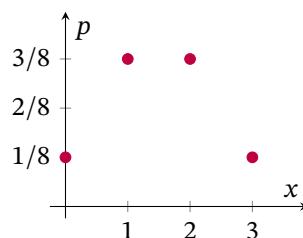
Zmienna X przyjmuje wartości w zbiorze $\{0, 1, 2, 3\}$, mamy:

$$\begin{aligned}X((R, R, R)) &= 0, \\X((R, R, O)) = X((R, O, R)) = X((O, R, R)) &= 1, \\X((R, O, O)) = X((O, R, O)) = X((O, O, R)) &= 2, \\X((O, O, O)) &= 3\end{aligned}$$

Łatwo wyznaczamy prawdopodobieństwa, z jakimi X przyjmuje powyższe wartości:

$$P(X = 0) = 1/8, \quad P(X = 1) = 3/8, \quad P(X = 2) = 3/8, \quad P(X = 3) = 1/8.$$

Równości te determinują rozkład zmiennej X . Łatwo możemy zwizualizować to na wykresie:



■ Podstawowe rozkłady dyskretnie

Opiszymy teraz pokrótko kilka podstawowych rozkładów zmiennych dyskretnych, pojawiających się stosunkowo często w rozmaitych przykładach i zastosowaniach.

Zmienna X ma **rozkład dwumianowy** z parametrami n, p , oznaczany jako $X \sim \text{Binom}(n, p)$, jeśli

$$P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k} \quad \text{dla } k \in \{0, \dots, n\}$$

Zmienna o rozkładzie dwumianowym opisuje **liczbę sukcesów w schemacie n prób Bernoulliego** o prawdopodobieństwie sukcesu p .

Przykład 4.

Rzucamy n razy monetą, na której orzeł wypada z prawdopodobieństwem p . Jeśli zmienna X oznacza liczbę orłów w takim ciągu rzutów, to ma ona rozkład dwumianowy:

$$P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k} \quad \text{dla } k \in \{0, \dots, n\}$$

Istotnie, jeśli w ciągu n rzutów wypadło k orłów, to takich różnych ciągów jest $\binom{n}{k}$ (wybieramy miejsca na orły, a w pozostałych $n - k$ miejscach wrzucamy reszki), przy czym każdy z nich mógł przytrafić się z prawdopodobieństwem $p^k(1 - p)^{n-k}$ (k razy wypadł orzeł i $n - k$ razy wypadła reszka).

Zmienna X ma **rozkład geometryczny** z parametrem p , oznaczany jako $X \sim \text{Geom}(p)$, jeśli

$$P(X = k) = (1 - p)^{k-1} \cdot p \quad \text{dla } k \in \mathbb{N}_+$$

Zmienna o rozkładzie geometrycznym opisuje **numer pierwszej próby zakończonej sukcesem** w nieskończonym schemacie Bernoulliego o prawdopodobieństwie sukcesu p .

Przykład 5.

Rzucamy monetą, na której orzeł wypada z prawdopodobieństwem p , aż do uzyskania pierwszego orła. Jeśli zmienna X oznacza liczbę rzutów w takim ciągu, to ma ona rozkład geometryczny:

$$P(X = k) = (1 - p)^{k-1} \cdot p$$

Istotnie, $X = k$ oznacza, że w $k - 1$ pierwszych rzutach musiała wypaść reszka (z prawdopodobieństwem $1 - p$), a w k -tym rzucie wypadł orzeł (z prawdopodobieństwem p).

Zmienna X ma **rozkład Poissona** z parametrem λ , oznaczany jako $X \sim \text{Pois}(\lambda)$, jeśli

$$P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!} \quad \text{dla } k \in \mathbb{N}$$

Rozkład Poissona stosowany jest w rozmaitych kontekstach. Może być stosowany jako dobre przybliżenie rozkładu Bernoulliego $\text{Binom}(n, p)$ w przypadku, gdy n jest duże, p jest małe, a $n \cdot p$ wynosi w przybliżeniu λ . Dzięki temu rozkład Poissona może być używany do badania zdarzeń „rzadkich” i „całkowicie losowych”, na przykład:

- przewidywanie liczby trzęsień ziemi na podstawie ich średnich wystąpień w przeszłości,
- szacowanie liczby błędów ortograficznych w książce o znanej liczbie znaków,
- zliczanie liczby telefonów odebranych w centrum obsługi klienta w ciągu godziny.

Przykład 6.

Kierownik laboratorium komputerowego otrzymuje średnio λ informacji o awarii komputera na miesiąc. Znajdziemy rozkład zmiennej X zliczającej liczbę otrzymanych takich informacji w miesiącu.

Podzielmy miesiąc na n przedziałów na tyle małych, że otrzymanie dwóch sygnałów w jednym przedziale jest zaniedbywalne (czyli $n \rightarrow \infty$). Ponadto, niech zdarzenia odpowiadające otrzymaniu awarii w dwóch różnych przedziałach będą niezależne. Mamy teraz do czynienia z ciągiem n prób Bernoulliego, obliczymy więc prawdopodobieństwo sukcesu (otrzymania informacji o awarii w podanym przedziale).

Skoro spośród n przedziałów średnio λ z nich zawiera awarię, to prawdopodobieństwo wynosi $p = \frac{\lambda}{n}$ i wtedy $X \sim \text{Binom}(n, \frac{\lambda}{n})$ dla dużych n , czyli

$$P(X = k) = \binom{n}{k} \left(\frac{\lambda}{n}\right)^k \left(1 - \frac{\lambda}{n}\right)^{n-k}$$

Dla $n \rightarrow \infty$ wyrażenie to zbiega do $e^{-\lambda} \lambda^k / k!$, więc (wprost z definicji) X ma rozkład Poissona z parametrem λ .

5.3.

Parametry rozkładu

W tym rozdziale przyjrzymy się kilku istotnym parametrom rozkładów zmiennych losowych, pozwalającym nam na analizowanie ich rozmaitych własności.

■ Wartość oczekiwana

Niech X będzie zmienną losową o rozkładzie dyskretnym. **Wartością oczekiwana** (średnią) X nazywamy wartość sumy

$$\mathbf{E}X = \sum_{x \in \mathbb{R}} (x \cdot P(X = x))$$

Wartość oczekiwana ma bardzo naturalną fizyczną interpretację: jest to ważona średnia wartości, które mogą być przyjęte przez daną zmienną, gdzie każda wartość ma wagę odpowiadającą jej prawdopodobieństwu.

Przykład 1.

Dysponujemy fałszywą kostką, w której na ścianie z trzema oczkami domalowano dwa dodatkowe oczka. Wyznaczmy wartość oczekiwana liczbę wyrzuconych oczek na tej kostce.

Oznaczmy odpowiednią zmienną jako X . Nietrudno znaleźć jej rozkład:

$$P(X = k) = \begin{cases} 1/3 & \text{dla } k = 5 \\ 1/6 & \text{dla } k = 1, 2, 4, 6 \end{cases}$$

Szukana przez nas wartość oczekiwana X wynosi więc

$$\begin{aligned} EX &= 1 \cdot P(X = 1) + 2 \cdot P(X = 2) + 4 \cdot P(X = 4) + 5 \cdot P(X = 5) + 6 \cdot P(X = 6) = \\ &= \frac{1}{6}(1+2+4+6) + \frac{1}{3} \cdot 5 = \frac{23}{6} \end{aligned}$$

Niekiedy zdarza się, że dysponujemy pewną zmienną losową i interesuje nas nie tyle średnia tej zmiennej, co średnia *pewnej funkcji* tej zmiennej; innymi słowy, mamy zadaną pewną funkcję $f : \mathbb{R} \rightarrow \mathbb{R}$ i naszym celem jest wyznaczenie $Ef(X)$. Wówczas możemy skorzystać z następującego faktu:

$$Ef(X) = \sum_{x \in \mathbb{R}} (f(x) \cdot P(X = x))$$

Przykład 2.

Losujemy liczbę ze zbioru $\{1, 2, \dots, 10\}$. Obliczymy średnią odległość tej liczby od 2.

Jeśli przez X oznaczymy wylosowaną liczbę, to naszym celem jest obliczenie $E|X - 2|$. Ponieważ rozkład X to

$$P(X = j) = 1/10 \quad \text{dla } j = 1, 2, \dots, 10,$$

na mocy powyższego twierdzenia zachodzi

$$E|X - 2| = \sum_{j=1}^{10} |j - 2| \cdot P(X = j) = \frac{1}{10} \sum_{j=1}^{10} |j - 2| = \frac{1}{10} + \frac{1}{10} \sum_{j=3}^{10} |j - 2| = \frac{1}{10} + \frac{36}{10} = 3.7$$

Wartość oczekiwana ma własność **liniowości**, tj. dla dowolnych zmiennych X, Y oraz dowolnego $c \in \mathbb{R}$ mamy:

- $E(cX) = c \cdot EX$
- $E(X + Y) = EX + EY$

Jeśli X, Y są niezależnymi zmiennymi losowymi (dyskretnymi), to dodatkowo zachodzi **multiplikatywność**:

$$E(XY) = EX \cdot EY$$

Przykład 3.

Rzucono n razy kostką do gry. Obliczymy wartość oczekiwanałącznej liczby wyrzuconych oczek.

Oznaczmy szukaną sumę jako X . Można próbować obliczyć średnią X bezpośrednio z definicji, ale to bardzo żmudne podejście, gdyż wyznaczenie rozkładu X jest niełatwym zagadnieniem kombinatorycznym.

Aby uniknąć tego problemu, rozbijemy X na sumę prostszych zmiennych: dla $i = 1, 2, \dots, n$ przez X_i oznaczy-

my liczbę oczek wyrzuconych w i -tym rzucie. Wówczas $X = X_1 + X_2 + \dots + X_n$ oraz

$$EX_i = 1 \cdot P(X_i = 1) + 2 \cdot P(X_i = 2) + \dots + 6 \cdot P(X_i = 6) = \frac{7}{2}$$

i możemy skorzystać z liniowości wartości oczekiwanej:

$$EX = EX_1 + EX_2 + \dots + EX_n = \frac{7n}{2}$$

■ Wariancja i odchylenie standardowe

Niech X będzie zmienną losową o skończonej wartości oczekiwanej. **Wariancję** X definiujemy wzorem

$$\text{Var } X = E(X - EX)^2$$

Odchyleniem standardowym zmiennej X nazywamy liczbę $\sigma(X) = \sqrt{\text{Var } X}$.

Przeprowadzając proste przekształcenia, łatwo wyprowadzić alternatywny wzór na wariancję

$$\text{Var } X = E(X^2) - (EX)^2,$$

który w wielu sytuacjach oferuje najprostszy sposób jej obliczenia.

O ile wartość oczekiwana mówi nam o tym, jaka jest średnia wartość zmiennej X , tak wariancja odpowiada za „rozrzut” jej różnych wartości. Aby lepiej zrozumieć intuicję za tym stojącą, rozważmy trzy zmienne losowe:

$$\begin{aligned} X &= 0, \\ Y &= \begin{cases} 1 & \text{z prawdopodobieństwem } 1/2, \\ -1 & \text{z prawdopodobieństwem } 1/2, \end{cases} \\ Z &= \begin{cases} 100 & \text{z prawdopodobieństwem } 1/2, \\ -100 & \text{z prawdopodobieństwem } 1/2. \end{cases} \end{aligned}$$

Z punktu widzenia wartości oczekiwanej, te zmienne nie różnią się między sobą – wszystkie mają średnią równą 0. Tym niemniej jest intuicyjnie jasne, że pierwsza z nich nie ma żadnego rozrzutu, druga z nich ma „umiarkowany” rozrzut, a trzecia z nich wykazuje największe odchylenia od swojej wartości oczekiwanej.

Można by się też zastanowić, dlaczego nie definiujemy wariancji bardziej oczywistym (intuicyjnie) wzorem $E|X - EX|$, mierzącym wprost jej średnie odchylenie od wartości oczekiwanej? Okazuje się, że tak zdefiniowana wariancja nie jest wygodna – nie posiada pewnych specjalnych algebraicznych własności. Stąd znacznie lepszym pomysłem jest mierzenie odchylenia w sensie średniokwadratowym $E(X - EX)^2$.

Przykład 4.

Ze zbioru $\{1, 2, \dots, n\}$ losujemy liczbę X . Obliczymy wariancję X .

Rozkład zmiennej X zadany jest przez równości

$$P(X = k) = 1/n \quad \text{dla } k = 1, 2, \dots, n,$$

więc

$$EX = \sum_{k=1}^n (k \cdot P(X = k)) = \frac{1}{n} \sum_{k=1}^n k = \frac{n+1}{2}$$

i analogicznie

$$E(X^2) = \sum_{k=1}^n (k^2 \cdot P(X = k)) = \frac{1}{n} \sum_{k=1}^n k^2 = \frac{(n+1)(2n+1)}{6}$$

Wobec tego

$$\text{Var } X = \text{E}(X^2) - (\text{E}X)^2 = \frac{(n+1)(2n+1)}{6} - \left(\frac{n+1}{2}\right)^2 = \frac{n^2-1}{12}$$

Wynika stąd, że dla danego n zmienna X średnio „odchyla się” od swojej wartości oczekiwanej o $\sqrt{\frac{n^2-1}{12}}$.

Jeśli zmienne losowe X i Y są niezależne i mają skończoną wariancję, to jest ona **addytywna**:

$$\text{Var}(X + Y) = \text{Var } X + \text{Var } Y$$

Ponadto, bezpośrednio z definicji, wariancja posiada następującą własność: jeśli X jest zmienną losową o skończonej wartości oczekiwanej oraz a, b są liczbami rzeczywistymi, to

$$\text{Var}(aX + b) = a^2 \text{Var } X$$

Przykład 5.

Rzucamy trzy razy monetą i przez X oznaczamy różnicę między liczbą wyrzuconych reszek i orłów. Obliczymy wartość oczekiwana i wariancję X .

Niech R będzie liczbą wyrzuconych reszek, a O – liczbą orłów. Jasne jest, że $R + O = 3$ oraz $X = R - O$, skąd bezpośrednio wynika, że $X = 2R - 3$. Łatwo wyznaczymy też rozkład R :

$$P(R = 0) = \frac{1}{8}, \quad P(R = 1) = \frac{3}{8}, \quad P(R = 2) = \frac{3}{8}, \quad P(R = 3) = \frac{1}{8}$$

Wobec tego, korzystając z liniowości wartości oczekiwanej:

$$\text{E}X = \text{E}(2R - 3) = 2\text{E}R - \text{E}3 = 2\left(0 \cdot \frac{1}{8} + 1 \cdot \frac{3}{8} + 2 \cdot \frac{3}{8} + 3 \cdot \frac{1}{8}\right) - 3 = 0$$

(dygresja: wynik ten jest intuicyjnie oczywisty, średnio liczba reszek i orłów jest ta sama i wynosi 1.5)

Ponadto, korzystając z podanej wcześniej własności wariancji,

$$\text{Var } X = \text{Var}(2R - 3) = 4 \text{Var } R$$

Jak już obliczyliśmy wcześniej, $\text{E}R = 1.5$ oraz

$$\text{E}R^2 = 0^2 \cdot \frac{1}{8} + 1^2 \cdot \frac{3}{8} + 2^2 \cdot \frac{3}{8} + 3^2 \cdot \frac{1}{8} = 3$$

Ostatecznie mamy więc

$$\text{Var } X = 4 \text{Var } R = 4(\text{E}(R^2) - (\text{E}R)^2) = 3$$

■ Wartości parametrów znanych rozkładów dyskretnych

Poniższa tabela przedstawia spis wartości oczekiwanych i wariancji dla znanych nam rozkładów dyskretnych. Wszystkie z nich można w łatwy sposób wyprowadzić wprost z definicji, większość powinna również intuicyjnie wynikać z interpretacji kombinatorycznych poszczególnych rozkładów.

	$\text{E}X$	$\text{Var } X$
$X \sim \text{Binom}(n, p)$	np	$np(1-p)$
$X \sim \text{Geom}(p)$	$1/p$	$(1-p)/p^2$
$X \sim \text{Pois}(\lambda)$	λ	λ

Funkcje tworzące prawdopodobieństwa

Niech X będzie zmienną losową o wartościach naturalnych. **Funkcją tworzącą prawdopodobieństwa** zmiennej X jest

$$g_X(t) = \sum_{k=0}^{\infty} P(X = k) \cdot t^k$$

Za pomocą funkcji tworzących możemy łatwo obliczać wartość oczekiwana i wariancję:

$$\text{EX} = g'_X(1), \quad \text{Var} X = g''_X(1) + g'_X(1) - g'^2_X(1),$$

o ile te wartości istnieją i są skończone. Wzory te wynikają wprost z definicji wartości oczekiwanej i wariancji.

Przykład 6.

Znajdziemy funkcję tworzącą prawdopodobieństwa zmiennej X o rozkładzie dwumianowym $\text{Binom}(n, p)$.

Mamy $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$, więc

$$g_X(t) = \sum_{k=0}^n \binom{n}{k} p^k (1-p)^{n-k} t^k = (pt + (1-p))^n = (pt - p + 1)^n$$

Dzięki takiej funkcji łatwo wyprowadzimy wzory na EX oraz $\text{Var} X$. Zaczniemy od wyznaczenia wzoru na pierwszą i drugą pochodną:

$$g'_X(t) = np(pt - p + 1)^{n-1}, \quad g''_X(t) = n(n-1)p^2(pt - p + 1)^{n-2},$$

skąd otrzymujemy

$$\text{EX} = g'_X(1) = np, \quad \text{Var} X = g''_X(1) + g'_X(1) - g'^2_X(1) = n(n-1)p^2 + np - n^2p^2 = np - np^2 = np(1-p)$$

Jeśli X jest zmienną losową o wartościach naturalnych oraz c jest liczbą naturalną, to łatwo możemy wyznaczyć funkcję zmiennej cX :

$$g_{cX}(t) = g_X(t^c)$$

Jeśli natomiast X i Y są **niezależnymi** zmiennymi o wartościach naturalnych, to funkcja tworząca prawdopodobieństwa sumy tych zmiennych jest równa

$$g_{X+Y}(t) = g_X(t) \cdot g_Y(t)$$

Przykład 7.

Ponownie obliczymy funkcję tworzącą prawdopodobieństwa dla zmiennej X o rozkładzie dwumianowym $\text{Binom}(n, p)$, tym razem rozbijając ją na niezależne zmienne X_i reprezentujące poszczególne próby Bernoulliego w całym schemacie.

Mamy więc $X = X_1 + X_2 + \dots + X_n$, gdzie $P(X_i = 1) = p$ oraz $P(X_i = 0) = 1 - p$. Funkcją tworzącą prawdopodobieństwa dla zmiennych X_i jest

$$g_{X_i}(t) = P(X_i = 0)t^0 + P(X_i = 1)t^1 = (1-p) + pt = pt - p + 1,$$

a skoro X to suma n niezależnych zmiennych o takim rozkładzie, to

$$g_X(t) = (g_{X_i}(t))^n = (pt - p + 1)^n$$

5.6. Wrzucamy losowo dwie kule do dwóch urn. Wartość oczekiwana liczby niepustych urn jest równa

- A. 1
- B. $\frac{3}{2}$
- C. $2(1 - \frac{1}{e})$

5.7. Wrzucamy do worka $n \geq 3$ kul, każdą z nich malujemy niezależnie z prawdopodobieństwem $\frac{1}{2}$ na biało. Wartość oczekiwana liczby białych kul

- A. wynosi $\frac{n(n-1)}{2}$
- B. wynosi $\frac{n}{2}$
- C. jest większa lub równa wariancji liczby białych kul

5.8. Dany jest generator bitów, dla którego każdy kolejny wygenerowany bit jest jedynką z prawdopodobieństwem 0.5. Niech X będzie zmienną losową oznaczającą liczbę wygenerowanych jedynek przez ten generator w 100 próbach. Nie wiadomo, czy kolejne losowania są niezależne. Wówczas

- A. $EX = 50$
- B. $EX = 50$, gdy kolejne bity są generowane niezależnie
- C. $\text{Var}(X) = 25$

5.9. Niech X i Y to zmienne losowe, takie że $EX = 1$, $\text{Var } X = 4$ i $Y = 2X + 3$. Wtedy

- A. $EY = 4$
- B. $\sigma(Y) = 4$
- C. $\text{Var } Y = 4$

5.10. Niech X i Y będą zdarzeniami niezależnymi, wtedy

- A. $E(X - Y) = EX - EY$
- B. $E(XY) = EX \cdot EY$
- C. $E(X/Y) = EX/EY$

5.11. Niech X i Y będą zmiennymi losowymi o wartościach nieujemnych. Wynika z tego, że

- A. $E(XY) = EX \cdot EY$, jeśli X i Y są niezależne
- B. $E(\min(X, Y)) = \min(EX, EY)$
- C. $E(\min(X, Y)) = \min(EX, EY)$, jeśli X i Y są niezależne

5.4.

Nierówności probabilistyczne

W rachunku prawdopodobieństwa czy statystyce często nie jest możliwe dokładne wyliczenie pewnych interesujących nas wartości. Niemniej jednak, w wielu zastosowaniach istotna jest nie tyle dokładna wartość, co jej oszacowanie.

Przykładowo, gracza może interesować, czy prawdopodobieństwo, że przegra, jest mniejsze niż pewna z góry ustalona liczba i na tej podstawie może podjąć decyzję, czy wziąć udział w grze. Podobnie przy badaniach statystycznych ważne jest oszacowanie prawdopodobieństwa, że błąd wyniesie mniej niż interesująca nas dokładność.

Ogólniej: jeśli przez X oznaczymy losowy błąd danej metody pomiarowej, jesteśmy zainteresowani nierów-

nościami postaci $P(X \geq x) \leq c$, gdzie c jest pewnym ustalonym parametrem.

Podstawowym narzędziem matematycznym służącym do uzyskiwania nierówności powyższego typu jest **nierówność Markowa**: dla dowolnej nieujemnej zmiennej losowej X oraz dla każdego $c > 0$ zachodzi

$$P(X \geq c) \leq \frac{EX}{c}$$

Przykład 1.

Oszacujemy prawdopodobieństwo uzyskania w 1000 rzutach monetą przynajmniej 750 orłów.

Niech $X_1, X_2, \dots, X_{1000}$ będą wynikami kolejnych rzutów (1 dla orła, 0 dla reszki). Wtedy $X = \sum_{i=1}^{1000} X_i$ jest liczbą wszystkich orłów. Jasne jest też, że $X \sim \text{Binom}(1000, \frac{1}{2})$.

Ponieważ dla rozkładu dwumianowego $EX = 1000 \cdot \frac{1}{2} = 500$, to z nierówności Markowa otrzymujemy

$$P(X \geq 750) = P\left(X \geq \frac{3}{2}EX\right) \leq \frac{2}{3}$$

Widzimy, że to niezbyt imponujące oszacowanie (intuicyjnie wynik powinien być dużo niższy).

Nierówność Markowa, choć niezwykle prosta, ma bardzo dużo zastosowań. Jej siła wynika między innymi z faktu, że możemy zastosować ją nie tylko do zmiennej losowej X , którą jesteśmy zainteresowani, ale także do zmiennych postaci $f(X)$, uzyskując nowe nierówności.

Na przykład, podstawiając zmienną $(X - EX)^2$ do nierówności Markowa, uzyskamy **nierówność Czebyszewa**: dla dowolnej nieujemnej losowej X oraz dla każdego $c > 0$ zachodzi

$$P(|X - EX| \geq c) \leq \frac{\text{Var } X}{c^2}$$

Przykład 2.

Kontynuujemy poprzedni przykład. Tym razem oszacujemy prawdopodobieństwo uzyskania co najmniej 750 orłów w ciągu 1000 rzutów monetą za pomocą nierówności Czebyszewa.

Dla zmiennej X o rozkładzie dwumianowym mamy $EX = 1000 \cdot \frac{1}{2} = 500$ oraz $\text{Var } X = 1000 \cdot \frac{1}{2} \cdot (1 - \frac{1}{2}) = 250$.

Nierówność Czebyszewa pozwoli nam na oszacowanie prawdopodobieństwa $P(|X - 500| \geq c)$ dla pewnej ustalonej liczby $c \in \mathbb{R}$. Aby móc otrzymać z tego $P(X \geq 750)$, skorzystamy z faktu, że w schemacie Bernouliego (u nas: 1000 rzutów monetą) zachodzi symetria $P(X \geq 750) = P(X \leq 250)$, co pozwala nam uzyskać równość

$$P(|X - 500| \geq 250) = P(X \geq 750) + P(X \leq 250) = 2 \cdot P(X \geq 750),$$

więc mamy

$$P(X \geq 750) = \frac{1}{2} \cdot P(|X - 500| \geq 250) \leq \frac{1}{2} \cdot \frac{250}{250^2} = \frac{1}{500}$$

Widzimy, że to dużo lepsze oszacowanie niż poprzednio.

Przypis redakcji

W rozdziale brakuje informacji na temat nierówności Chernoffa (jest ona wspomniana w podstawie programowej), jednak zawarta tu teoria jest wystarczająca do rozwiązywania zadań z przeanalizowanych na potrzeby tego repetytorium archiwalnych egzaminów.

Zestaw zadań

5.12. Niech X będzie zmienną losową reprezentującą długość trwania programu dla różnych wejść, niech μ to wartość oczekiwana X , zaś σ to odchylenie standardowe. Prawdą jest, że

- A. $P(X > 2\mu) \leq \frac{1}{4}$
- B. $P(X > \mu + 2\sigma) \leq \frac{1}{4}$
- C. $P(X > 100\mu) \leq \frac{1}{10^{10}}$

5.5.

Ciągły rozkład prawdopodobieństwa

Dla dyskretnych zmiennych losowych mieliśmy $\sum_{x \in \mathbb{R}} P(X = x) = 1$, czyli suma prawdopodobieństw przełączalnie wielu wartości uzyskiwanych przez zmienną jest równa 1. Zauważmy jednak, że taka zmienna nie pozwoliłaby nam rozpatrzyć doświadczenia polegającego na wylosowaniu punktu z odcinka $[0, 1]$ – ponieważ zbiór ten jest nieprzeliczalny, a każdy punkt równo prawdopodobny do wylosowania, nie potrafilibyśmy określić prawdopodobieństwa wylosowania pojedynczego punktu, tak żeby wszystkie z nich zsumowały się do 1.

Podobnie jest w każdej sytuacji, w której wynikiem jakiegoś doświadczenia może być dowolna liczba rzeczywista, np. pomiar wzrostu osoby lub prędkości samochodu. Mamy tu do czynienia z **ciągłym rozkładem prawdopodobieństwa** i w tym rozdziale zobaczymy, jak sobie z nim poradzić.

■ Prawdopodobieństwo geometryczne

O ile w problemach dyskretnych korzystamy zazwyczaj ze schematu klasycznego, tak w doświadczeniach losowych o charakterze ciągłym przydatne może okazać się **prawdopodobieństwo geometryczne**. Najprostszym przykładem jest losowanie punktu z pewnego zbioru Ω .

Założmy, że Ω jest pewnym podzbiorem przestrzeni \mathbb{R}^d i ma skończoną miarę (długość, pole powierzchni, objętość... – w zależności od wymiaru d). Wówczas prawdopodobieństwo dowolnego zdarzenia $A \subseteq \Omega$ jest **proporcjonalne do jego miary**:

$$P(A) = \frac{|A|}{|\Omega|}$$

Przykład 1.

Z przedziału $[0, 1]$ wybieramy losowo dwie liczby. Jakie jest prawdopodobieństwo tego, że obie te liczby są mniejsze niż $1/2$?

Zastosujemy prawdopodobieństwo geometryczne. Oznaczmy wybrane liczby przez x, y . Wówczas

$$\Omega = \{(x, y) \mid x, y \in [0, 1]\} = [0, 1]^2$$

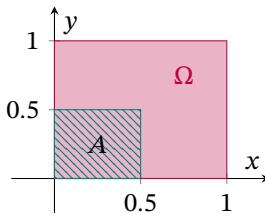
Następnym krokiem jest zinterpretowanie badanego zdarzenia A jako podzbioru Ω . Mamy

$$A = \left\{(x, y) \in \Omega \mid x, y \leq \frac{1}{2}\right\} = \left[0, \frac{1}{2}\right]^2,$$

a zatem obliczając pola powierzchni odpowiednich zbiorów:

$$P(A) = \frac{|A|}{|\Omega|} = \frac{1}{4}$$

Możemy to łatwo zwizualizować w układzie współrzędnych:

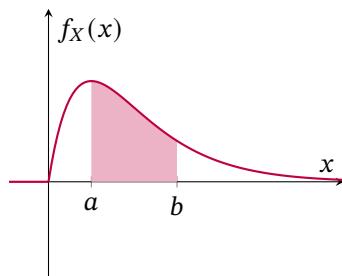


■ Ciągłe zmienne losowe

Zmienna X ma **rozkład ciągły**, jeśli istnieje funkcja $f_X : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$, taka że dla każdego przedziału $[a, b]$ zachodzi

$$P(X \in [a, b]) = \int_a^b f_X(x) dx$$

Funkcja f_X w powyższej definicji to **funkcja gęstości** zmiennej X . Dla zmiennej ciągłej prawdopodobieństwo, że zmienna przyjmie wartość między a i b jest równe polu powierzchni pod krzywą gęstości nad tym odcinkiem:



Oczywiście jest również, że prawdopodobieństwa wszystkich możliwych wartości X muszą sumować się do 1, w związku z czym dla każdej funkcji gęstości mamy

$$\int_{-\infty}^{+\infty} f_X(x) dx = 1$$

Przykład 2.

Rzucamy niesymetryczną monetą, na której prawdopodobieństwo wypadnięcia orła wynosi $1/3$. Jeśli wypadnie orzeł, to losujemy punkt X z odcinka $[-2, 0)$, a jeśli wypadnie reszka, to losujemy X z odcinka $[0, 3]$. Wyznaczmy funkcję gęstości zmiennej X .

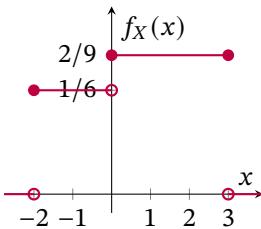
Żeby X przyjęło jedną z wartości z przedziału $[-2, 0)$, na moncie musi wypaść orzeł (z prawdopodobieństwem $1/3$). Ponieważ wylosowanie każdego punktu jest równe prawdopodobne, dla $x \in [-2, 0)$ mamy

$$f_X(x) = \frac{1}{3} \cdot \frac{1}{0 - (-2)} = \frac{1}{6}$$

Analogicznym rozumowaniem dla reszki i przedziału $[0, 3]$ dochodzimy do wniosku, że funkcją gęstości zmiennej X jest

$$f_X(x) = \begin{cases} \frac{1}{6} & \text{dla } x \in [-2, 0), \\ \frac{2}{9} & \text{dla } x \in [0, 3], \\ 0 & \text{wpp.} \end{cases}$$

Oznacza to, że X przyjmie wartość z przedziału $[-2, 0)$ z prawdopodobieństwem $1/6$ oraz wartość z przedziału $[0, 3]$ z prawdopodobieństwem $2/9$. Możemy zobrazować f_X w układzie współrzędnych:



Z rysunku nietrudno przekonać się też, że

$$\int_{-\infty}^{\infty} f_X(x) dx = 2 \cdot \frac{1}{6} + 3 \cdot \frac{2}{9} = 1,$$

co potwierdza dobre zdefiniowanie funkcji gęstości.

Znany jest jeszcze jeden ważny fakt dotyczący rozkładu sum niezależnych zmiennych losowych o rozkładzie ciągłym.

Założmy, że X, Y są **niezależnymi** ciągłymi zmiennymi losowymi z gęstościami f_X, f_Y . Wówczas zmienna $X + Y$ ma rozkład z gęstością będącą splotem funkcji f_X i f_Y :

$$f_{X+Y}(x) = \int_{-\infty}^{+\infty} f_X(y) f_Y(x-y) dy$$

Dystrybuanta

W przypadku doświadczeń losowych o charakterze ciągłym z reguły jesteśmy zainteresowani zdarzeniami typu $P(X \leq c)$ dla pewnej zmiennej losowej X i liczby rzeczywistej c .

Dystrybuanta zmiennej losowej X to funkcja $F_X : \mathbb{R} \rightarrow [0, 1]$ określona jako

$$F_X(x) = P(X \leq x)$$

Nietrudno zauważyć, że dystrybuanta jest ściśle powiązana z funkcją gęstości, przez co **jednoznacznie determinuje rozkład** dowolnej zmiennej losowej.

Co ważne: dystrybuanta nie jest związana wyłącznie z pojęciem rozkładu ciągłego i może być stosowana także do innych zmiennych.

Przykład 3.

Rozważmy (dyskretną) zmienną losową dwupunktową, przyjmującą wartości 1 i -1, każdą z prawdopodobieństwem 1/2. Jej dystrybuantą jest funkcja

$$F(x) = \begin{cases} 0 & \text{dla } x \in (-\infty, -1), \\ \frac{1}{2} & \text{dla } x \in [-1, 1), \\ 1 & \text{dla } x \in [1, \infty). \end{cases}$$

Możemy zauważyć, że nie każda funkcja może być dystrybuantą – ma to związek ze znannymi nam podstawowymi własnościami prawdopodobieństwa. W szczególności wyróżniamy następujące **własności dystrybuanty**:

1. F_X jest niemalejąca,
2. F_X jest prawostronnie ciągła,
3. $\lim_{x \rightarrow -\infty} F_X(x) = 0$ oraz $\lim_{x \rightarrow +\infty} F_X(x) = 1$.

Dodatkowo, jeśli X jest zmienną ciągłą, to F_X jest funkcją ciągłą.

Intuicyjnie patrząc na definicje dystrybuanty i funkcji gęstości można dojść do wniosku, że ich wzajemne powiązanie wygląda podobnie jak relacja funkcji i jej pochodnej. Istotnie, istnieje twierdzenie, które w wielu sytuacjach pozwala obliczyć gęstość zmiennej losowej, gdy znana jest jej dystrybuanta:

Niech X będzie ciągłą zmienną losową. Wtedy jej dystrybuanta F_X jest różniczkowalna wszędzie poza skończonym zbiorem punktów oraz funkcja

$$f_X(x) = \begin{cases} F'_X(x) & \text{jeśli } F'_X(x) \text{ istnieje,} \\ 0 & \text{wpp.} \end{cases}$$

jest gęstością zmiennej X .

Przykład 4.

Rozważmy zmienną losową X o dystrybuancie

$$F_X(x) = \begin{cases} 0 & \text{dla } x \in (-\infty, 0), \\ 2x & \text{dla } x \in (0, \frac{1}{2}), \\ 1 & \text{dla } x \in (\frac{1}{2}, \infty). \end{cases}$$

Funkcja F_X jest różniczkowalna wszędzie poza punktami $x = 0$ i $x = 1/2$. Ponadto, $F'_X(x) = 0$ dla $x \in (-\infty, 0) \cup (1/2, \infty)$ oraz $F'_X(x) = 2$ dla $x \in (0, 1/2)$. Zatem funkcja

$$f_X(x) = \begin{cases} 0 & \text{dla } x \in (-\infty, 0] \cup [\frac{1}{2}, \infty), \\ 2 & \text{dla } x \in (0, \frac{1}{2}). \end{cases}$$

jest gęstością zmiennej X .

Warto również na tym etapie podkreślić, że istnieją rozkłady, które nie są ani ciągłe, ani dyskretnie, co obrazuje poniższy przykład.

Przykład 5.

Rzucamy standardową monetą i jeśli wypadnie orzeł, to zmienna X przyjmuje wartość 3, a jeśli reszka, zmienna X ma wartość wylosowaną z przedziału $(0, 1)$.

Dystrybuanta takiego rozkładu to

$$F_X(x) = \begin{cases} 0 & \text{dla } x \in (-\infty, 0), \\ \frac{x}{2} & \text{dla } x \in [0, 1), \\ \frac{1}{2} & \text{dla } x \in [1, 3), \\ 1 & \text{dla } x \in [3, \infty), \end{cases}$$

Zmienna X nie jest więc ani dyskretna (może przyjąć nieprzeliczalnie wiele wartości), ani ciągła (jej dystrybuanta nie jest ciągła w punkcie $x = 3$).

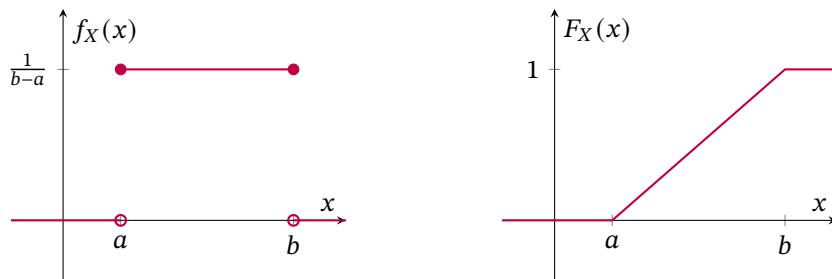
■ Podstawowe rozkłady ciągłe

Opiszemy teraz pokrótko kilka podstawowych rozkładów zmiennych ciągłych, pojawiających się stosunkowo często w rozmaitych przykładach i zastosowaniach.

Zmienna X ma **rozkład jednostajny** z parametrami a, b , oznaczany jako $X \sim \text{Unif}(a, b)$, jeśli X ma gęstość

$$f_X(x) = \frac{1}{b - a} \quad \text{dla } x \in [a, b]$$

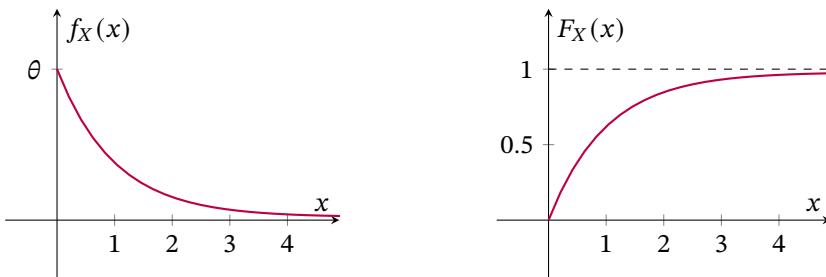
Zmienna o rozkładzie jednostajnym opisuje losowanie liczby z zakresu $[a, b]$ o równo rozłożonym prawdopodobieństwie.



Zmienna X ma **rozkład wykładniczy** z parametrem θ , oznaczany jako $X \sim \text{Exp}(\theta)$, jeśli X ma gęstość

$$f_X(x) = \theta e^{-\theta x} \quad \text{dla } x \geq 0$$

Zmienna o rozkładzie wykładniczym modeluje **czas oczekiwania na zdarzenie**, które ma cały czas taką samą szansę zajścia, na przykład telefon w centrum telefonicznym lub czas do zajścia rozpadu radioaktywnego. Parametr θ określa średnią liczbę wystąpień badanego zdarzenia w ustalonej jednostce czasu. O rozkładzie tym można myśleć jak o ciągłej wersji rozkładu geometrycznego.

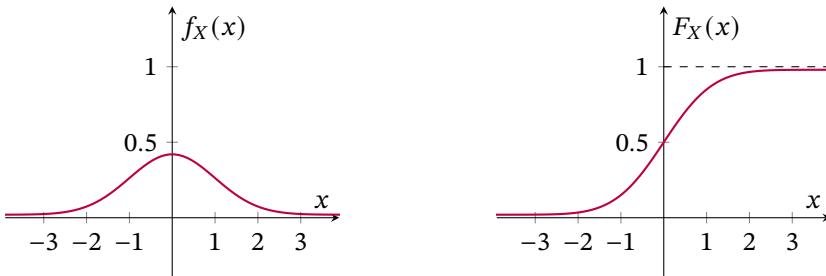


Zmienna X ma **rozkład normalny** (Gaussa) o wartości oczekiwanej μ i odchyleniu standardowym σ , oznaczany jako $X \sim N(\mu, \sigma)$, jeśli X ma gęstość

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Rozkład normalny ma dość skomplikowaną definicję. Suma dużej liczby niezależnych zmiennych, z których żadna nie dominuje pozostałych (tj. nie przyjmuje dużo większych wartości ani nie ma decydującego wpływu wyniku) ma w przybliżeniu rozkład normalny. Przykładem może być wzrost lub masa człowieka o zbliżonych cechach, np. płci i rasie.

Rozkład $N(0, 1)$ nazywamy **standardowym rozkładem normalnym**. Często pojawia się on w definicjach innych rozkładów oraz różnych rozumowaniach.



Jeśli X, Y są **niezależnymi** zmiennymi losowymi o rozkładzie normalnym ze średnimi μ_X, μ_Y oraz odchyleniami standardowymi σ_X, σ_Y , to

- zmienna losowa $X + Y$ ma rozkład normalny $N(\mu_X + \mu_Y, \sqrt{\sigma_X^2 + \sigma_Y^2})$,
- zmienna losowa $X - Y$ ma rozkład normalny $N(\mu_X - \mu_Y, \sqrt{\sigma_X^2 + \sigma_Y^2})$.

■ Parametry rozkładów ciągłych

W ogólności **wszystkie podstawowe własności wartości oczekiwanej i wariancji przenoszą się z przypadku dyskretnego na ciągły**. Dla formalności, poniżej uogólnimy jedynie podstawowe definicje.

Niech X będzie ciągłą zmienną losową o gęstości f_X . **Wartością oczekiwana** X nazywamy wartość

$$EX = \int_{-\infty}^{+\infty} xf_X(x) dx,$$

o ile $|x|f_X(x)$ jest całkowalna. Jeśli wartość oczekiwana istnieje, to **wariancję** obliczamy ze standardowego wzoru:

$$\text{Var } X = E(X - EX)^2 \quad \text{albo} \quad \text{Var } X = E(X^2) - (EX)^2$$

■ Wartości parametrów znanych rozkładów ciągłych

Poniższa tabela przedstawia spis wartości oczekiwanych i wariancji dla znanych nam rozkładów ciągłych.

	EX	$\text{Var } X$
$X \sim \text{Unif}(a, b)$	$(a + b)/2$	$(a - b)^2/12$
$X \sim \text{Exp}(\theta)$	$1/\theta$	$1/\theta^2$
$X \sim N(\mu, \sigma)$	μ	σ

■ Centralne twierdzenie graniczne

Rozkłady normalne należą do najważniejszych rozkładów w rachunku prawdopodobieństwa. Okazuje się, że wiele występujących w przyrodzie wielkości ma rozkład w przybliżeniu normalny, np. rozkład wzrostu, wagi, ilorazu inteligencji czy innych cech populacji. Matematyczne wyjaśnienie tego faktu opisuje **centralne twierdzenie graniczne**.

Niech X_1, X_2, \dots będzie ciągiem niezależnych zmiennych losowych o tym samym rozkładzie, wartości oczekiwanej μ i wariancji $\sigma^2 > 0$. Wówczas dla dowolnego $t \in \mathbb{R}$ mamy

$$\lim_{n \rightarrow \infty} P\left(\frac{X_1 + X_2 + \dots + X_n - n\mu}{\sigma\sqrt{n}} \leq t\right) = \Phi(t),$$

gdzie Φ to dystrybuanta standardowego rozkładu normalnego $N(0, 1)$. Jej konkretne wartości można znaleźć w tablicach matematycznych.

Twierdzenie to mówi, że rozkład sumy wielu niezależnych zmiennych o tym samym rozkładzie jest bliski normalnemu.

Przykład 6.

Rzucono monetą 10000 razy i okazało się, że orzeł wypadł 5200 razy. Sprawdzimy, jak duże są podstawy do przypuszczenia, że moneta jest niesymetryczna.

Rozważmy ciąg zmiennych X_i przyjmujących wartość 1, jeśli w i -tym rzucie wypadł orzeł, lub wartość 0 w przeciwnym przypadku. Jasne jest, że $X_1, X_2, \dots, X_{10000}$ są niezależne o tym samym rozkładzie, możemy więc zastosować centralne twierdzenie graniczne, żeby oszacować prawdopodobieństwo rozważanego zdarzenia (5200 orłów w ciągu 10000 rzutów).

Łatwo obliczamy $EX_i = 1/2$ oraz $\text{Var } X_i = 1/4$, stąd $\mu = 1/2$ i $\sigma = \sqrt{\text{Var } X_i} = 1/2$ i dalej

$$\begin{aligned} P(X_1 + X_2 + \dots + X_{10000} \leq 5200) &= P(X_1 + X_2 + \dots + X_{10000} - 10000\mu \leq 5200 - 5000) = \\ &= P\left(\frac{X_1 + X_2 + \dots + X_{10000} - 10000\mu}{\sigma\sqrt{10000}} \leq \frac{200}{50}\right) \approx \Phi(4) \end{aligned}$$

Sprawdzając w tablicach: $\Phi(4) \approx 0.99997$, więc prawdopodobieństwo, że $X_1 + X_2 + \dots + X_{10000} \geq 5200$ (zdarzenie przeciwnie do obliczonego powyżej – w 10000 rzutach monetą wyrzucimy przynajmniej 5200 orłów) ma prawdopodobieństwo $1 - \Phi(4) \approx 0.00003$. To bardzo mało; są więc podstawy, by sądzić, że moneta nie jest symetryczna.

Zestaw zadań

5.13. Zmienne X i Y mają rozkład jednostajny na przedziale $[0, 2]$. Jeśli $Z = X + Y$, to

- A. $P(Z \geq 1) = \frac{7}{8}$
- B. Z ma rozkład jednostajny na $[0, 4]$ i wartość oczekiwana 2
- C. $P(|EZ - Z| \geq 1) > \frac{2}{3}$

5.14. Niech $F_X(t)$ będzie dystrybuantą pewnej zmiennej losowej X . Założymy, że $F_X(1) = \frac{1}{3}$. Wynika z tego, że

- A. istnieje takie t , że $F_X(t) = 1$
- B. $F_X(2) > \frac{1}{3}$
- C. istnieje takie t , że $F_X(t) < \frac{1}{4}$

5.15. Niech X i Y będą niezależnymi zmiennymi losowymi. Wynika z tego, że

- A. jeśli X i Y mają rozkład Poissona, to $X - Y$ też
- B. jeśli X i Y mają rozkład normalny, to $X - Y$ też
- C. jeśli X i Y mają rozkład geometryczny, to $X - Y$ też

5.16. Niech X i Y będą dwiema niezależnymi zmiennymi losowymi o rozkładzie normalnym o wartości oczekiwanej 0 i wariancji 4. Wynika z tego, że

- A. $X - Y$ ma taki sam rozkład jak $X + Y$
- B. odchylenie standardowe zmiennej losowej $X - Y$ jest równe 4
- C. dla $a, b > 0$ zmienna losowa $aX + bY$ ma rozkład normalny

5.6.

Łańcuchy Markowa

Przypis redakcji

Rozdział nie jest jeszcze stworzony – w historii przeanalizowanych na potrzeby tego repetytorium egzaminów nie pojawiło się żadne zadanie z łańcuchów Markowa.

5.7.

Rozwiązańia

Rozwiązańia

5.1. Istnieje przestrzeń probabilistyczna Ω i zdarzenia $A, B \subseteq \Omega$ takie, że $P(A) = P(B) = \frac{2}{3}$ oraz

TAK A. A i B są niezależne

NIE B. $P(A|B) = \frac{1}{3}$

NIE C. $P(A|B) \neq P(B|A)$

Przy rozwiązyaniu tego typu zadań dobrze skorzystać z diagramu Venna. Łatwo można zauważyć, że zachodzi $\frac{1}{3} \leq P(A \cap B) \leq \frac{2}{3}$, a to pomoże w dalszych rozważaniach:

A. Żeby A i B były niezależne, musi zachodzić

$$P(A \cap B) = P(A) \cdot P(B) = \frac{4}{9}$$

Ponieważ $\frac{1}{3} \leq \frac{4}{9} \leq \frac{2}{3}$, taka sytuacja jest możliwa.

B. Tym razem mamy

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \Leftrightarrow P(A \cap B) = P(A|B) \cdot P(B) = \frac{1}{3} \cdot \frac{2}{3} = \frac{2}{9}$$

Ponieważ $\frac{2}{9} < \frac{1}{3}$, taka sytuacja nie może zajść.

C. Zauważmy, że dla dowolnych zdarzeń A, B spełniających warunki zadania jest

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A \cap B)}{2/3} = \frac{P(A \cap B)}{P(A)} = P(B|A)$$

5.2. Chcemy rozpalić ognisko, lecz mamy do dyspozycji tylko trzy zapałki. Prawdopodobieństwo rozpalenia ogniska pojedynczą zapałką wynosi 0.4, dwiema złączonymi zapałkami – 0.6, zaś trzema złączonymi zapałkami – 0.8. Żeby zmaksymalizować prawdopodobieństwo rozpalenia ogniska, należy

TAK A. użyć od razu trzech zapałek

NIE B. użyć najpierw dwóch zapałek, a potem jednej

NIE C. używać pojedynczych zapałek

Oznaczmy zdarzenia:

- A – rozpalimy ognisko trzema złączonymi zapałkami,
- B – rozpalimy ognisko dwiema zapałkami, a potem jedną,
- C – rozpalimy ognisko trzema pojedynczymi zapałkami.

Obliczamy i porównujemy poszczególne prawdopodobieństwa:

- $P(A) = 0.8$
- $P(B) = 0.6 + (1 - 0.6) \cdot 0.4 = 0.76$ (odpalamy za pierwszym razem dwoma zapałkami lub nie odpalamy za pierwszym razem i odpalamy za drugim)
- $P(C) = 0.4 + (1 - 0.4) \cdot 0.4 + (1 - 0.4)^2 \cdot 0.4 = 0.784$ (analogiczne rozumowanie – rozpalamy za pierwszym, drugim albo trzecim razem)

Najbardziej opłacalne jest więc użycie trzech złączonych zapałek.

5.3. Rzucamy 10 razy symetryczną monetą. Wynika z tego, że

- TAK** A. prawdopodobieństwo, że wyrzucimy dokładnie 2 orły jest równe $\frac{45}{1024}$
- NIE** B. prawdopodobieństwo, że wyrzucimy orła w ostatnim rzucie pod warunkiem, że we wszystkich 10 rzutach wyrzucimy dokładnie 1 orła, jest równe $\frac{1}{2}$
- NIE** C. prawdopodobieństwo, że wyrzucimy orła w ostatnim rzucie pod warunkiem, że w pierwszych 9 rzutach wyrzucimy dokładnie 6 orłów, jest równe $\frac{1}{2}$

A. Oznaczmy zdarzenie „wyrzucono dokładnie 2 orły” jako A . W zadaniu mamy do czynienia ze schematem 10 prób Bernoulliego o prawdopodobieństwie sukcesu $\frac{1}{2}$ (wyrzucono orła). W związku z tym

$$P(A) = \binom{10}{2} \left(\frac{1}{2}\right)^2 \left(1 - \frac{1}{2}\right)^8 = \frac{45}{2^{10}} = \frac{45}{1024}$$

- B. Wiedza o tym, że we wszystkich 10 rzutach wyrzuciliśmy dokładnie 1 orła, ogranicza nam przestrzeń zdarzeń elementarnych do 10 elementów: każdy z nich różni się pozycją orła. Tylko jedno zdarzenie elementarne posiada orła na ostatniej pozycji, więc szukane prawdopodobieństwo warunkowe wynosi $\frac{1}{10}$.
- C. Wiedza o tym, że w pierwszych 9 rzutach wyrzucimy dokładnie 6 orłów, nie dostarcza nam żadnych dodatkowych informacji na temat ostatniego rzutu – jest on wykonywany niezależnie od pierwszych dziewięciu. W związku z tym w tak ograniczonej przestrzeni zdarzeń elementarnych jest tyle samo zdarzeń posiadających na ostatnim miejscu reszkę co zdarzeń na ostatnim miejscu z orłem. Prawdopodobieństwo wynosi więc $\frac{1}{2}$.

Możemy również przekonać się o tym z obliczeń: oznaczmy zdarzenia

- A – wyrzucono orła w ostatnim rzucie,
- B – w pierwszych 9 rzutach wyrzucono dokładnie 6 orłów.

Zdarzenie B to schemat 9 prób Bernoulliego o prawdopodobieństwie sukcesu $\frac{1}{2}$. Obliczamy kolejno:

$$P(B) = \binom{9}{6} \left(\frac{1}{2}\right)^6 \left(1 - \frac{1}{2}\right)^3 = \frac{84}{2^9} = \frac{21}{128}$$

$$P(A \cap B) = \frac{1}{2} \cdot P(B) = \frac{21}{256}$$

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{1}{2}$$

- 5.4. Mamy 9 ponumerowanych kart od 1 do 9. Dwie osoby A oraz B losują bez zwracania po jednej karcie. Wygrywa osoba, która wylosuje kartę z większym numerem. Prawdą jest, że

- TAK** A. zdarzenia „wygrała osoba A ” oraz „osoba A wylosowała kartę z numerem 5” są niezależne
- TAK** B. zdarzenie „wygrała osoba A pod warunkiem, że wylosowała kartę z numerem 7” ma prawdopodobieństwo $\frac{3}{4}$
- NIE** C. zdarzenia „wygrała osoba A ” oraz „wygrała osoba B ” są niezależne

Jasne jest, że $|\Omega| = 9 \cdot 8 = 72$.

- A. Oznaczmy zdarzenia X – wygrała osoba A , Y – osoba A wylosowała kartę z numerem 5. Obliczamy:
- $P(X) = \frac{8+7+\dots+1}{72} = \frac{1}{2}$ (zliczamy liczbę par (a, b) , w których $a > b$)
 - $P(Y) = \frac{8}{72} = \frac{1}{9}$
 - $P(X \cap Y) = \frac{4}{72} = \frac{1}{18}$

Ponieważ $P(X \cap Y) = P(X)P(Y)$, zdarzenia X i Y są niezależne.

- B. Oznaczmy zdarzenia X – wygrała osoba A , Y – osoba A wylosowała kartę z numerem 7. Obliczamy:
- $P(X \cap Y) = \frac{6}{72} = \frac{1}{12}$
 - $P(Y) = \frac{8}{72} = \frac{1}{9}$

- $P(X|Y) = \frac{P(X \cap Y)}{P(Y)} = \frac{3}{4}$

C. Oznaczmy zdarzenia X – wygrała osoba A , Y – wygrała osoba B . Obliczamy:

- $P(X) = P(Y) = \frac{1}{2}$
- $P(X \cap Y) = 0$

Ponieważ $P(X \cap Y) \neq P(X)P(Y)$, zdarzenia X i Y nie są niezależne.

5.5. Rzucamy symetryczną kostką dwudziestościenną. Niech X będzie liczbą wyrzuconych oczek. Wtedy niezależne są zdarzenia „wyrzucono parzystą liczbę oczek” oraz

TAK A. wyrzucono liczbę oczek podzielną przez 3

TAK B. wyrzucono liczbę oczek podzielną przez 5

TAK C. $X \leq 8$

Łatwo rozwiążemy to zadanie za pomocą prostej intuicji idącej za niezależnością zdarzeń: „czy zajście zdarzenia A daje dodatkową информацию na temat zdarzenia B ?”.

Rozważmy, ile mamy liczb podzielnych przez 3 w kostce dwudziestościennej: $\{3, 6, 9, 12, 15, 18\}$, są więc 3 parzyste i 3 nieparzyste. Analogicznie dla liczb podzielnych przez 5: $\{5, 10, 15, 20\}$ – 2 parzyste i 2 nieparzyste. Zdarzenie „wyrzucono parzystą liczbę oczek” w żaden sposób nie dostarcza nam dodatkowych informacji o prawdopodobieństwie wyrzucenia liczby podzielnej przez 3 lub 5, bo jest tyle samo nieparzystych co parzystych wielokrotności tych liczb.

To samo tyczy się prawdopodobieństwa wyrzucenia liczby mniejszej bądź równej 8: jest tyle samo parzystych i nieparzystych liczb w zbiorze $\{1, 2, \dots, 8\}$, więc parzystość nie dostarcza nam żadnych dodatkowych informacji.

Wszystkie zdarzenia są więc niezależne. Można to łatwo potwierdzić za pomocą standardowych obliczeń (które tu pominiemy).

5.6. Wrzucamy losowo dwie kule do dwóch urn. Wartość oczekiwana liczby niepustych urn jest równa

NIE A. 1

TAK B. $\frac{3}{2}$

NIE C. $2(1 - \frac{1}{e})$

Oznaczmy przez X liczbę niepustych urn. Zauważmy, że jedynie możliwe wartości X to 1 i 2. Wartość ta jest jednoznacznie wyznaczona przez wrzucenie drugiej kuli – albo trafi ona do tej samej urny, co pierwsza kula, albo do innej. Oba te zdarzenia są równie prawdopodobne (i niezależne od wrzucenia pierwszej kuli!), więc $P(X = 1) = P(X = 2) = \frac{1}{2}$. Mamy stąd

$$EX = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{2} = \frac{3}{2}$$

5.7. Wrzucamy do worka $n \geq 3$ kul, każdą z nich malujemy niezależnie z prawdopodobieństwem $\frac{1}{2}$ na biało. Wartość oczekiwana liczby białych kul

NIE A. wynosi $\frac{n(n-1)}{2}$

TAK B. wynosi $\frac{n}{2}$

TAK C. jest większa lub równa wariancji liczby białych kul

Oznaczmy przez X liczbę białych kul i podzielmy ją na n niezależnych zmiennych X_i równych 1, gdy i -ta kula jest biała, lub 0, gdy i -ta kula nie jest biała. Stąd $X = X_1 + X_2 + \dots + X_n$.

Mamy teraz

$$EX_i = 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{1}{2}, \quad \text{Var } X_i = E(X^2) - (EX)^2 = \left(0^2 \cdot \frac{1}{2} + 1^2 \cdot \frac{1}{2}\right) - \frac{1}{4} = \frac{1}{4}$$

i możemy skorzystać z liniowości wartości oczekiwanej oraz addytywności wariancji (ponieważ X_i są niezależne):

$$EX = n \cdot EX_i = \frac{n}{2}, \quad \text{Var } X = n \cdot \text{Var } X_i = \frac{n}{4}$$

- 5.8.** Dany jest generator bitów, dla którego każdy kolejny wygenerowany bit jest jedynką z prawdopodobieństwem 0.5. Niech X będzie zmienną losową oznaczającą liczbę wygenerowanych jedynek przez ten generator w 100 próbach. Nie wiadomo, czy kolejne losowania są niezależne. Wówczas

- TAK** A. $EX = 50$
TAK B. $EX = 50$, gdy kolejne bity są generowane niezależnie
NIE C. $\text{Var}(X) = 25$

Niech $X = X_1 + X_2 + \dots + X_{100}$, gdzie X_i jest równe i -temu wygenerowanemu bitowi. Mamy

$$EX_i = 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{1}{2}, \quad \text{Var } X_i = E(X^2) - (EX)^2 = \left(0^2 \cdot \frac{1}{2} + 1^2 \cdot \frac{1}{2}\right) - \frac{1}{4} = \frac{1}{4}$$

Prawdziwość podpunktów **A.** i **B.** wynika wprost z liniowości wartości oczekiwanej – nie ma tu znaczenia, czy losowania kolejnych bitów są niezależne. W obu przypadkach $EX = 100 \cdot EX_i = 50$.

Żeby wariancja X była równa 25, musiałoby być $\text{Var } X = 100 \cdot \text{Var } X_i = 25$, ale to jest prawdziwe wyłącznie, gdy zmienne X_i są ze sobą niezależne (wynika to wprost z addytywności wariancji). W związku z tym podpunkt **C.** jest fałszywy.

- 5.9.** Niech X i Y to zmienne losowe, takie że $EX = 1$, $\text{Var } X = 4$ i $Y = 2X + 3$. Wtedy

- NIE** A. $EY = 4$
TAK B. $\sigma(Y) = 4$
NIE C. $\text{Var } Y = 4$

Z liniowości wartości oczekiwanej mamy, że $EY = 2 \cdot EX + 3 = 2 \cdot 1 + 3 = 5$, co dowodzi fałszywości podpunktu **A.**

Skorzystamy z własności wartości oczekiwanej, że jeśli X jest zmienną losową o skończonej wartości oczekiwanej oraz a, b są liczbami rzeczywistymi, to $\text{Var}(aX + b) = a^2 \text{Var } X$. U nas

$$\text{Var } Y = \text{Var}(2X + 3) = 2^2 \text{Var } X = 16 \quad \text{i tym samym} \quad \sigma(Y) = \sqrt{\text{Var } Y} = 4$$

Stąd podpunkt **B.** jest prawdziwy i **C.** jest fałszywy.

- 5.10.** Niech X i Y będą zdarzeniami niezależnymi, wtedy

- TAK** A. $E(X - Y) = EX - EY$
TAK B. $E(XY) = EX \cdot EY$
NIE C. $E(X/Y) = EX/EY$

Podpunkty **A.** i **B.** są prawdziwe, odpowiednio z własności liniowości i multiplikatywności wartości oczekiwanej.

Nietrudno też zauważyć, że jeśli $EY = 0$, równość w podpunkcie **C.** nie zachodzi.

- 5.11.** Niech X i Y będą zmiennymi losowymi o wartościach nieujemnych. Wynika z tego, że

- TAK** A. $E(XY) = EX \cdot EY$, jeśli X i Y są niezależne
NIE B. $E(\min(X, Y)) = \min(EX, EY)$
NIE C. $E(\min(X, Y)) = \min(EX, EY)$, jeśli X i Y są niezależne

Podpunkt **A.** jest prawdziwy, wprost z definicji multiplikatywności wartości oczekiwanej.

Do pokazania kontrprzykładu do **C.** (a przy okazji do **B.**) rozważmy dwukrotny rzut kostką oraz oznaczmy przez X i Y uzyskane wyniki przy odpowiednio pierwszym i drugim rzucie (zmienne te są w oczywisty sposób niezależne). Wtedy $EX = EY = 3.5$, czyli $\min(EX, EY) = 3.5$.

Obliczymy teraz $E(\min(X, Y))$. Jest to wartość oczekiwana mniejszej z wyrzuconych wartości. Mamy 11 opcji, że będzie to liczba 1:

$$(\{6, 1\}, \{5, 1\}, \dots, \{1, 1\}, \{1, 2\}, \dots, \{1, 6\})$$

Analogicznie sprawdzamy, że jest 9 opcji, że będzie to liczba 2, itd. Zatem

$$E(\min(X, Y)) = \frac{1}{36}(11 \cdot 1 + 9 \cdot 2 + 7 \cdot 3 + 5 \cdot 4 + 3 \cdot 5 + 1 \cdot 6) \approx 2.53 \neq 3.5$$

5.12. Niech X będzie zmienną losową reprezentującą długość trwania programu dla różnych wejść, niech μ to wartość oczekiwana X , zaś σ to odchylenie standardowe. Prawdą jest, że

- NIE** A. $P(X > 2\mu) \leq \frac{1}{4}$
- TAK** B. $P(X > \mu + 2\sigma) \leq \frac{1}{4}$
- NIE** C. $P(X > 100\mu) \leq \frac{1}{10^{10}}$

A. Wstawiając $c = 2\mu$ do nierówności Markowa, otrzymujemy

$$P(X \geq 2\mu) \leq \frac{\mu}{2\mu} = \frac{1}{2},$$

co pozwala nam podejrzewać, że odpowiedź jest fałszywa. Rzeczywiście, nietrudno o kontrprzykład: niech $P(X = 2.01) = P(X = 0.49) = P(X = 0.5) = \frac{1}{3}$. Widzimy, że $EX = 1$, ale

$$P(X > 2\mu) = P(X > 2) = \frac{1}{3} > \frac{1}{4}$$

B. Skorzystamy z nierówności Czebyszewa. Biorąc $c = 2\sigma$ mamy

$$P(|X - \mu| \geq 2\sigma) \leq \frac{\sigma^2}{(2\sigma)^2} = \frac{1}{4}$$

Zauważmy, że

$$P(|X - \mu| \geq 2\sigma) = P(X \geq \mu + 2\sigma) + P(X \leq \mu - 2\sigma) \leq \frac{1}{4},$$

więc w szczególności $P(X \geq \mu + 2\sigma) \leq \frac{1}{4}$ i odpowiedź jest prawdziwa.

C. Nierówność Markowa jest w tym przypadku zbyt słaba (jej ograniczenie z góry jest znacząco za małe), a inne znane nam nierówności nie pozwalają nam uzyskać lepszego wyniku. Taka sytuacja sugeruje fałszywość nierówności, co pokażemy poprzez poniższy kontrprzykład.

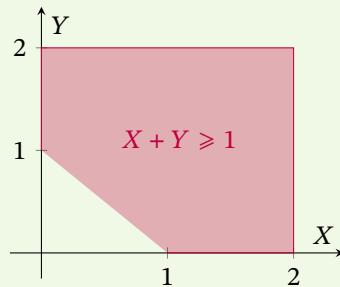
Weźmy $P(X = 101) = \frac{1}{102}$ oraz $P(X = \frac{1}{101}) = \frac{101}{102}$. W oczywisty sposób $EX = 1$, ale

$$P(X > 100\mu) = P(X > 100) = \frac{1}{102} > \frac{1}{10^{10}}$$

5.13. Zmienne X i Y mają rozkład jednostajny na przedziale $[0, 2]$. Jeśli $Z = X + Y$, to

- TAK** A. $P(Z \geq 1) = \frac{7}{8}$
- NIE** B. Z ma rozkład jednostajny na $[0, 4]$ i wartość oczekiwana 2
- NIE** C. $P(|EZ - Z| \geq 1) > \frac{2}{3}$

A. Ponieważ zmienne X, Y mają rozkład jednostajny, wykorzystamy prawdopodobieństwo geometryczne. Zaznaczmy w układzie współrzędnych zdarzenie $X + Y \geq 1$:



Widzimy, że szukane zdarzenie zajmuje dokładnie $\frac{7}{8}$ całego kwadratu 2×2 . Odpowiedź jest więc prawdziwa.

- B.** Z nie ma rozkładu jednostajnego, intuicyjnie widać, że np. częściej wypadnie wynik 2 niż 0. Żeby się o tym przekonać, możemy obliczyć funkcję gęstości Z . Użyjemy znanego nam wzoru na gęstość zmiennej będącej sumą dwóch niezależnych zmiennych losowych:

$$f_Z(x) = \int_{-\infty}^{+\infty} f_X(y) f_Y(x-y) dy$$

Gęstości f_X, f_Y są takie same i, z definicji rozkładu jednostajnego, są dane wzorem

$$f_X(x) = \begin{cases} \frac{1}{2} & \text{dla } x \in [0, 2], \\ 0 & \text{wpp.} \end{cases}$$

Ponieważ f_X, f_Y są niezerowe wyłącznie w przedziale $[0, 2]$, ograniczymy całkowanie do tego przedziału. Obliczamy:

- dla $f_X(y) \neq 0$, czyli $y \in [0, 2]$:

$$f_Z(x) = \int_0^x f_X(y) f_Y(x-y) dy = \int_0^x \frac{1}{2} \cdot \frac{1}{2} dy = \frac{1}{4} \int_0^x dy = \frac{1}{4} \cdot y \Big|_0^x = \frac{x}{4}$$

- dla $f_Y(x-y) \neq 0$, czyli $x \in [2, 4]$:

$$f_Z(x) = \int_{x-2}^2 f_X(y) f_Y(x-y) dy = \int_{x-2}^2 \frac{1}{2} \cdot \frac{1}{2} dy = \frac{1}{4} \int_{x-2}^2 dy = \frac{1}{4} \cdot y \Big|_{x-2}^2 = \frac{1}{4} (2 - (x-2)) = 1 - \frac{x}{4}$$

Ostatecznie funkcją gęstości Z jest więc

$$f_Z(x) = \begin{cases} \frac{x}{4} & \text{dla } x \in [0, 2), \\ 1 - \frac{x}{4} & \text{dla } x \in [2, 4], \\ 0 & \text{wpp.} \end{cases}$$

i pokazuje to, że Z nie ma rozkładu jednostajnego.

- C.** Z nierówności Czebyszewa mamy, że $P(|EZ - Z| \geq 1) \leq \text{Var } Z$. Nietrudno pokazać, że wariancja Z jest równa $\frac{2}{3}$: ze wzoru na wariancję rozkładu jednostajnego Unif(0, 2) mamy

$$\text{Var } X = \text{Var } Y = \frac{(2-0)^2}{12} = \frac{1}{3},$$

a skoro Z jest sumą dwóch niezależnych zmiennych, możemy skorzystać z addytywności wariancji:

$$\text{Var } Z = \text{Var } X + \text{Var } Y = \frac{2}{3}$$

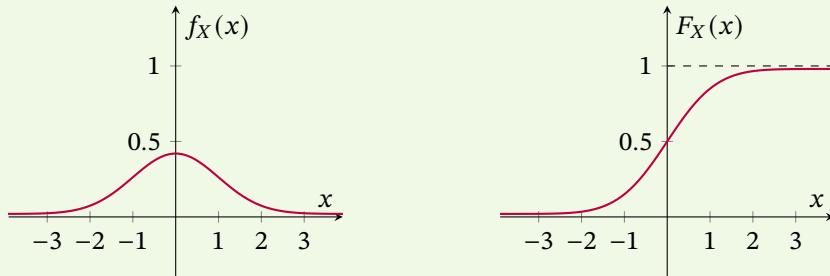
- 5.14.** Niech $F_X(t)$ będzie dystrybuantą pewnej zmiennej losowej X . Założmy, że $F_X(1) = \frac{1}{3}$. Wynika z tego, że

NIE **A.** istnieje takie t , że $F_X(t) = 1$

NIE B. $F_X(2) > \frac{1}{3}$

TAK C. istnieje takie t , że $F_X(t) < \frac{1}{4}$

- A. Zgodnie z własnościami dystrybuanty, mamy pewność, że $\lim_{x \rightarrow \infty} F_X(x) = 1$, ale to niekoniecznie oznacza, że dla pewnego argumentu wartość 1 jest osiągana. Istotnie, nietrudno o przykład, w którym prosta $y = 1$ jest asymptotą poziomą; dzieje się tak choćby dla standardowego rozkładu normalnego:



Wówczas nie istnieje żadne $t \in \mathbb{R}$, dla którego $F_X(t) = 1$.

- B. Dystrybuanta jest funkcją niemalejącą, co oznacza, że $F_X(2) \geq \frac{1}{3}$. Może więc być $F_X(2) = \frac{1}{3}$, co pokazuje fałszywość odpowiedzi.
- C. Ponieważ $\lim_{x \rightarrow -\infty} F_X(x) = 0$, dla bardzo małych argumentów będziemy uzyskiwali bardzo małe wartości, na pewno mniejsze od $\frac{1}{4}$.

- 5.15.** Niech X i Y będą niezależnymi zmiennymi losowymi. Wynika z tego, że

NIE A. jeśli X i Y mają rozkład Poissona, to $X - Y$ też

TAK B. jeśli X i Y mają rozkład normalny, to $X - Y$ też

NIE C. jeśli X i Y mają rozkład geometryczny, to $X - Y$ też

- A. Niech $X \sim \text{Pois}(\lambda_X)$ oraz $Y \sim \text{Pois}(\lambda_Y)$. Takie zmienne modelują liczbę wystąpień pewnego nieprzewidywalnego zdarzenia w ustalonym przedziale czasu, przy czym zdarzenia te mają średnią częstość wystąpienia odpowiednio λ_X, λ_Y .

O ile zmienna $X + Y$ ma rozkład Poissona (badamy łączne wystąpienia obu niezależnych zdarzeń) z parametrem $\lambda_X + \lambda_Y$, o tyle $X - Y$ napotyka na problem: zmienne o rozkładzie Poissona przyjmują wartości całkowite nieujemne, podczas gdy różnica $\lambda_X - \lambda_Y$ może potencjalnie przyjąć wartości mniejsze od 0.

- B. Zgodnie ze znany nam faktem, różnica dwóch niezależnych zmiennych losowych o rozkładach normalnych odpowiednio $N(\mu_X, \sigma_X)$ oraz $N(\mu_Y, \sigma_Y)$ ma również rozkład normalny z wartością oczekiwana $\mu_X - \mu_Y$ i odchyleniem standardowym $\sqrt{\sigma_X^2 + \sigma_Y^2}$.

- C. Kontrargument jest bardzo podobny jak w podpunkcie A. Zmienna o rozkładzie geometrycznym może przyjmować wyłącznie wartości całkowite dodatnie, a różnica $X - Y$ potencjalnie przyjmie wartość ujemną bądź zerową.

- 5.16.** Niech X i Y będą dwiema niezależnymi zmiennymi losowymi o rozkładzie normalnym o wartości oczekiwanej 0 i wariancji 4. Wynika z tego, że

TAK A. $X - Y$ ma taki sam rozkład jak $X + Y$

NIE B. odchylenie standardowe zmiennej losowej $X - Y$ jest równe 4

TAK C. dla $a, b > 0$ zmienna losowa $aX + bY$ ma rozkład normalny

- A.** Korzystając ze znanych nam faktów na temat rozkładu sumy i różnicy dwóch niezależnych zmiennych losowych o rozkładzie normalnym otrzymujemy:

$$X - Y \sim N(\mu_X - \mu_Y, \sqrt{\sigma_X^2 + \sigma_Y^2}) \sim N(0, \sqrt{8})$$

$$X + Y \sim N(\mu_X + \mu_Y, \sqrt{\sigma_X^2 + \sigma_Y^2}) \sim N(0, \sqrt{8})$$

- B.** Jak pokazaliśmy w podpunkcie A., odchylenie standardowe zmiennej $X - Y$ jest równe $\sqrt{8}$.

- C.** Nietrudno intuicyjnie zauważyc, że zmienne aX, bY mają rozkład normalny o przeskalowanych wartościach względem X i Y . Ponieważ suma niezależnych zmiennych o rozkładzie normalnym ma również rozkład normalny, odpowiedź jest prawdziwa.

6

Algorytmy i struktury danych

Materiały teoretyczne zostały opracowane na podstawie slajdów Krzysztofa Diksa. ([przyp. red.: potrzebny link do źródła](#))

Podstawa programowa

1. Kryteria oceny **efektywności algorytmów**.
2. **Koszt zamortyzowany**.
3. Podstawowe **algorytmy sortowania**.
4. **Słowniki** i metody ich realizacji.
5. **Kolejki priorytetowe** i metody ich realizacji.

6.1.

Poprawność i efektywność algorytmów

Powiemy, że algorytm A jest (całkowicie) **poprawny** względem specyfikacji $\langle \alpha, \beta \rangle$, jeśli spełnia następujące warunki:

- **częściowa poprawność**

Dla każdych danych wejściowych spełniających warunek początkowy α , jeśli obliczenie algorytmu A kończy się prawidłowo, wyniki spełniają warunek β .

- **określoność obliczeń**

Dla każdych danych wejściowych spełniających warunek początkowy α obliczenie algorytmu A nie jest przerwane.

- **własność stopu**

Dla każdych danych wejściowych spełniających warunek początkowy α obliczenie algorytmu A nie jest nieskończone.

Złożoność obliczeniowa algorytmu

Interesują nas dwa typy złożoności:

- **pamięciowa**

- a. ilość pamięci niezbędna do wykonania algorytmu;
- b. jednostka miary: słowo pamięci maszyny.

- **czasowa**

- a. czas pracy niezbędny do zrealizowania algorytmu;
- b. jednostka miary: **operacja dominująca**, czyli liczba wszystkich operacji jednostkowych wykonywanych przez algorytm powinna być „proporcjonalna” do liczby wszystkich operacji dominujących. Intuicyjnie myślimy o niej jak o operacji, która najbardziej wpływa na czas działania algorytmu.

Wprowadźmy oznaczenia:

- D_n – zbiór możliwych danych wejściowych rozmiaru n
- $t(d)$ – liczba operacji dominujących dla zestawu danych d
- X_n – zmienna losowa, której wartością jest $t(d)$ dla $d \in D_n$
- $E[X_n]$ - wartość oczekiwana X_n
- $p_{n,k}$ - prawdopodobieństwo, że dla danych rozmiaru n algorytm wykona k operacji dominujących

Przez **pesymistyczną** złożoność czasową algorytmu rozumie się funkcję

$$W(n) = \sup\{t(d) : d \in D_n\}$$

Przez **oczekiwana** złożoność czasową algorytmu rozumie się funkcję

$$A(n) = \sum_{k \geq 0} k \cdot p_{n,k} = E[X_n]$$

Intuicyjnie, pesymistyczna złożoność czasowa opisuje zachowanie algorytmu w najgorszym możliwym przypadku, natomiast oczekiwana złożoność czasowa zachowanie w przeciętnym przypadku.

Algorytm w miejscu poza pamięcią na przechowywanie danych, potrzebuje tylko dodatkowej pamięci stałego rozmiaru niezależnie od rozmiaru danych.

Algorytm stabilny zachowuje względny porządek elementów o tych samych wartościach (np. przy sortowaniu nie zmienia miejscami takich samych elementów).

Przykład 1.

Obliczymy złożoność czasową sortowania przez wstawianie (ang. *insertion sort*). Na wejściu dana jest tablica $a[1...n]$, którą chcemy posortować.

```
a[0] = -INF; // strażnik, dostatecznie mała wartość
for (int i = 2; i <= n; i++) {
    v = a[i];
    j = i - 1;
    while (v < a[j]) { // operacja dominująca
        a[j + 1] = a[j];
        j = j - 1;
    }
    a[j + 1] = v;
}
```

Inwersja w ciągu a to para indeksów (i, j) , $1 \leq i < j \leq n$, taka że $a[i] > a[j]$, czyli że element większy znajduje się przed mniejszym. Liczbę inwersji w tablicy a oznaczamy przez $\text{Inv}(a)$. W oczywisty sposób zachodzi $0 \leq \text{Inv}(a) \leq \frac{1}{2}n(n - 1)$.

Złożoność czasowa algorytmu sortowania przez wstawianie mierzona liczbą wykonania operacji dominującej $v < a[j]$, wynosi $n-1+\text{Inv}(a)$, ponieważ dla każdej inwersji (i, j) wykonamy zamianę miejscami elementów $a[i], a[j]$. Dodatkowo, przed wyjściem z pętli `while` wykonamy jedno porównanie, które nie będzie spełniało warunku pętli.

Z tego wynika, że pesymistyczna złożoność czasowa algorytmu sortowania przez wstawianie wynosi

$$W(n) = n - 1 + \frac{1}{2} \cdot n \cdot (n - 1)$$

Insertion sort jest algorytmem w miejscu oraz stabilnym.

■ Notacja asymptotyczna

Rozważamy funkcje o argumentach będących nieujemnymi liczbami całkowitymi i o nieujemnych, rzeczywistych wartościach.

- Dla danej funkcji $g(n)$ przez $O(g(n))$ oznaczamy zbiór funkcji

$$O(g(n)) = \{f(n) : \text{istnieją dodatnie stałe } c, n_0, \text{ takie że } \forall_{n \geq n_0} 0 \leq f(n) \leq c \cdot g(n)\}$$

Piszemy $f(n) = O(g(n))$, gdy $f(n) \in O(g(n))$. Możemy o tym myśleć jak o **ograniczeniu z góra** tempa wzrostu f przez g , podobnie jak to robimy na analizie matematycznej.

- Dla danej funkcji $g(n)$ przez $\Omega(g(n))$ oznaczamy zbiór funkcji

$$\Omega(g(n)) = \{f(n) : \text{istnieją dodatnie stałe } c, n_0, \text{ takie że } \forall_{n \geq n_0} 0 \leq c \cdot g(n) \leq f(n)\}$$

Piszemy $f(n) = \Omega(g(n))$, gdy $f(n) \in \Omega(g(n))$. Możemy o tym myśleć jak o notacji przeciwej do O , a więc jak o **ograniczeniu z dołu** tempa wzrostu f przez g .

- Dla danej funkcji $g(n)$ przez $\Theta(g(n))$ oznaczamy zbiór funkcji

$$\Theta(g(n)) = \{f(n) : \text{istnieją dodatnie stałe } c_1, c_2, n_0, \text{ takie że } \forall_{n \geq n_0} 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}$$

Piszemy $f(n) = \Theta(g(n))$, gdy $f(n) \in \Theta(g(n))$. Oznacza to, że tempo wzrostu f i g jest takie samo, a więc zachodzi jednocześnie $f(n) = \Omega(g(n))$ oraz $f(n) = O(g(n))$.

Przykład 2.

Kontynuujemy przykład z algorytmem insertion sort. Przypomnijmy obliczoną wcześniej złożoność pesymistyczną:

$$W(n) = n - 1 + \frac{1}{2} \cdot n \cdot (n - 1) = \frac{n^2}{2} + \frac{n}{2} - 1$$

Dzięki notacji asymptotycznej możemy więc powiedzieć, że pesymistyczna złożoność algorytmu insertion sort to $\Theta(n^2)$.

Obliczmy złożoność oczekiwana $A(n)$. Dla każdej pary różnych indeksów i, j utworzą one inwersję z prawdopodobieństwem równym $\frac{1}{2}$. Z niezależności wartości oczekiwanej uzyskujemy

$$A(n) = n - 1 + \frac{1}{2} \cdot \frac{1}{2} \cdot n \cdot (n - 1) = \frac{n^2}{4} + \frac{3n}{4} - 1$$

Zatem oczekiwana złożoność algorytmu insertion sort to również $\Theta(n^2)$.

Przypis redakcji

W części teoretycznej brak informacji o koszcie zamortyzowanym, jednak zawarta tu teoria jest wystarczająca do rozwiązywania zadań z przeanalizowanych na potrzeby tego repetytorium archiwalnych egzaminów.

- 6.1.** Agata i Bartek grają w grę polegającą na tym, że Agata wybiera dwie liczby ze zbioru $\{1, 2, \dots, n\}$, a Bartek zadaje pytania o wybrany podzbior i dowiaduje się, czy co najmniej jedna z wybranych liczb znajduje się w nim. Liczba zapytań wystarczających do określenia przez Bartka obu liczb jest

- A. $O(n)$
- B. $O(\sqrt{n})$
- C. $O(\log n)$

6.2. Kolejki priorytetowe

Kopiec to struktura danych oparta na drzewie. We wszystkich wierzchołkach trzymamy wartości, zwane **kluczami**. Wartość w rodzinie wierzchołka jest zawsze mniejsza (lub zawsze większa) od wartości potomka.

Przypis redakcji

Brakuje tu opisu kopca binarnego, a pojawia się on w dalszej części teorii.

Kopiec zupełny

Kopiec zupełny to kopiec binarny, w którym liście występują tylko na ostatnim i przedostatnim poziomie, spójnie ułożone od lewej do prawej. Jego wysokość zatem jest rzędu $O(\log n)$, gdzie n to liczba wierzchołków. Operacja dodania elementu do takiego kopca polega w uproszczeniu na:

- Jeśli dodawany element jest większy (zakładamy typ MAX) od korzenia, to zamieniamy je. W wyniku tej operacji wciąż mamy jeden element do dołożenia do poddrzewa.
- Wyliczamy, czy nadmiarowy element powinien trafić do lewego czy prawego poddrzewa, żeby utrzymać „zupełność” kopca.
- Rekurencyjnie dodajemy element do wybranego poddrzewa.

Wierzchołki kopca ponumerujemy od 1 do n , tak że ojcem wierzchołka i jest wierzchołek o numerze $\lfloor \frac{i}{2} \rfloor$, a $2i$ oraz $2i + 1$ to odpowiednio jego lewy i prawy syn (o ile istnieją). Wartości w wierzchołkach będziemy trzymać w tablicy a .

Jeśli wierzchołki o numerach $[l + 1, \dots, r]$ spełniają własność kopca, to następująca procedura sprawi, że będzie ją spełniać też wierzchołek l .

```
void DownHeap(l, r) {
    int i = l, j = 2 * i, v = a[i];
    while (j <= r) { // dopóki istnieje syn wierzchołka
        // sprawdzamy, czy prawy syn ma większą wartość
        if (j + 1 <= r && a[j] < a[j + 1])
            j = j + 1;

        // czy nie spełnia wartości kopca?
        if (v < a[j]) {
            // podnosimy wartość z syna
            a[i] = a[j];
            // schodzimy niżej
            i = j;
        }
    }
}
```

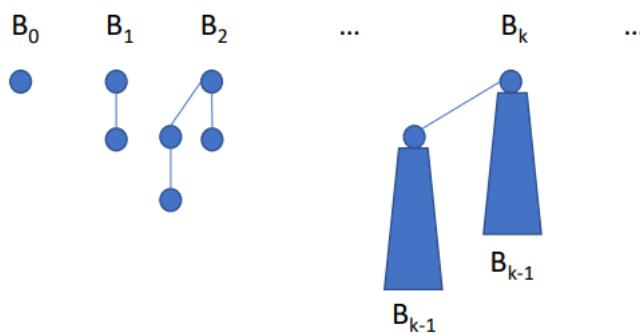
```

        j = 2 * i;
    } else {
        // wszystko jest okej
        break;
    }
}
a[i] = v;
}

```

■ Kopiec dwumianowy (kolejka dwumianowa)

Drzewo dwumianowe B_k jest zdefiniowane rekurencyjnie. Jest to drzewo B_{k-1} z drugim drzewem B_{k-1} przyłączonym krawędzią do korzenia. Drzewo B_0 składa się z pojedynczego wierzchołka. Widać zatem, że $|B_k| = 2^k$.



Rysunek 6.1: Drzewa dwumianowe

Kolejka dwumianowa składa się z lasu drzew dwumianowych o parami różnych rozmiarach, których rozmiary w sumie dają n . W każdym drzewie elementy są rozmieszczone tak, żeby wartości spełniały porządek kopcowy.

Dostęp do najmniejszego elementu obywa się przez wskaźnik na korzeń z najmniejszą wartością.

W takiej kolejce ważne są dwie operacje:

- $\text{Join}(T_1, T_2)$ łączy dwa drzewa tego samego stopnia w jedno drzewo o stopniu o 1 większym z zachowaniem porządku kopcowego. Złożoność: $O(1)$.
- $\text{Union}(Q_1, Q_2)$ łączy dwie kolejki dwumianowe Q_1, Q_2 w jedną kolejkę Q_1 , korzystając z operacji Join . Złożoność: $O(\log n)$.

Powyższe operacje wykorzystuje się, żeby zaimplementować kolejkę priorytetową. Na przykład, wstawienie elementu do kolejki to $\text{Union}()$ z jednoelementową kolejką.

■ Kopiec Fibonacciego

Kopiec Fibonacciego składa się z listy dwukierunkowej korzeni kolejnych drzew, tym razem już może być wiele drzew o tych samych rozmiarach. Utrzymywany jest wskaźnik na korzeń z najmniejszą wartością.

Operacja znajdowania minimum jest zatem trywialna, i działa w czasie stałym.

Wstawianie elementu polega na stworzeniu nowego jednoelementowego drzewa i dodaniu go do listy.

Operacja usunięcia najmniejszego elementu przebiega w 3 fazach.

1. Usuwamy korzeń zawierający minimalny element, a jego dzieci (z poddrzewami) dołączamy do listy drzew kopca.

2. Musimy uaktualnić wskaźnik na minimalny klucz. Na liście korzeni może być n elementów, zatem najpierw będziemy łączyć drzewa o tym samym stopniu (liczba synów korzenia).
3. Uaktualniamy wskaźnik na minimum.

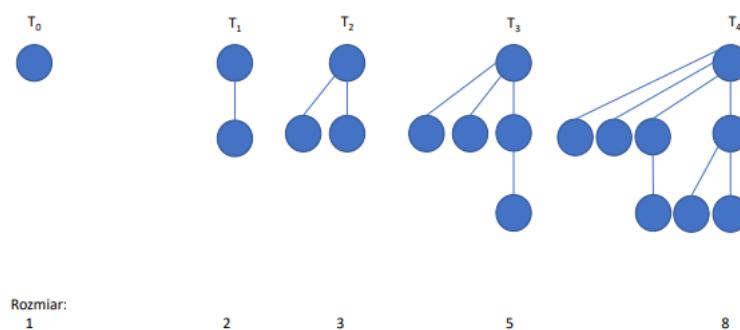
Amortyzowany koszt tej operacji to $O(\log n)$. Jeśli wcześniej wykonywaliśmy same operacje dodawania wierzchołków i usuwania minimum, to otrzymamy w wyniku tej operacji kopiec dwumianowy.

Operacja zmniejszenia wartości (używana w opisany później algorytmie Dijkstry) zaczyna od zmniejszenia wartości zadanego wierzchołka i w razie, gdy nowa wartość zaburza porządek kopcowy (wierzchołek ma wartość mniejszą niż rodzic), wyołujemy operację odcinania rozważanego wierzchołka.

Operacja odcinania wierzchołka x odcina go od ojca y i dołącza do listy korzeni kopca. Dodatkowo, y zostaje oznaczony jako wierzchołek, który utracił jedno dziecko. Przy próbie odcięcia drugiego dziecka, nie odcinamy go, tylko wyołujemy $cut(y)$, przenosząc operację w górę drzewa tak długo, aż dojdziemy do korzenia, lub wierzchołka który nie utracił jeszcze syna.

Dzięki tym operacjom otrzymujemy koszt amortyzowany $O(1)$ operacji zmniejszenia wartości klucza.

Nazwa kopca pochodzi od tego, że minimalny rozmiar drzewa o stopniu d to liczba Fibonacciego o indeksie $(d + 2)$



Rysunek 6.2: Drzewa o najmniejszym możliwym rozmiarze w kopcu Fibonacciego

■ Porównanie

Zestawienie złożoności najważniejszych operacji (złożoności oznaczone gwiazdką są amortyzowane):

operacja	kopiec zupełny	kopiec dwumianowy	kopiec Fibonacciego
inicjalizacja pustego kopca	$O(1)$	$O(1)$	$O(1)$
podanie najmniejszego elementu	$O(1)$	$O(1)$	$O(1)$
usunięcie najmniejszego elementu	$O(\log n)$	$O(\log n)$	$O(\log n)^*$
wstawienie elementu	$O(\log n)$	$O(\log n)$	$O(1)^*$
zmniejszenie jednej wartości	$O(\log n)$	$O(\log n)$	$O(1)^*$

Zestaw zadań

- 6.2.** Do początkowo pustego kopca Fibonacciego typu min wstawiamy kolejno rekordy o priorytetach odpowiednio $1, 2, \dots, 2020$, otrzymując kopiec T , a następnie wykonujemy operację `DeleteMin`, otrzymując kopiec T' . Wynika z tego, że

- A. kopiec T składa się z co najwyżej 8 drzew
- B. kopiec T' składa się z co najwyżej 8 drzew
- C. najwyższe drzewo w kopcu T' ma wysokość (tzn. największą możliwą liczbę krawędzi od korzenia do liścia w drzewie) co najmniej 10

6.3. Algorytmy sortowania

■ Insertion sort

Inaczej sortowanie przez wstawianie, opisane powyżej. Cechy:

- w miejscu
- stabilny
- złożoność zależna od liczby inwersji
- oczekiwana oraz pesymistyczna złożoność kwadratowa

■ Bubble sort

Inaczej sortowanie bąbelkowe.

```
for (int i = n; i >= 2; i--) {  
    for (int j = 1; j <= i - 1; j++) {  
        if (a[j] > a[j + 1])          // #porównania - wszystkie możliwe czyli n * (n - 1) / 2  
            swap(a[j], a[j + 1]);    // #zamiany = Inv(a)  
    }  
}
```

Cechy:

- w miejscu (+);
- stabilny (+);
- liczba zamian zależna od liczby inwersji (+/-);
- kwadratowa liczba porównań, niezależnie do danych (-).

■ Selection sort

Inaczej sortowanie przez wybieranie.

```
for (int i = n; i >= 2; i--) {  
    i_max = 1;  
    for (int j = 2; j <= i; j++) {  
        if (a[j] > a[i_max])    // #porównania = n * (n - 1) / 2  
            i_max := j;  
    }  
    swap(a[i], a[i_max]);    // #zamiany = n - 1  
}
```

Cechy:

- w miejscu (+);
- nie jest stabilny (-);
- mała liczba zamian (+);
- kwadratowa liczba porównań, niezależnie do danych (-).

■ Heap sort

Inaczej sortowanie przez kopcowanie. Dzięki użyciu kopca otrzymujemy szybszy selection sort, ponieważ umiemy znajdować maksimum w logarytmicznym czasie.

```
// Budowa kopca.  
// Zauważmy, że n / 2 wartości w liściach może zostać przepisanych z tablicy a.  
for (int i = n / 2; i >= 1; i--)  
    DownHeap(i, n);  
  
// Niezmiennik: a[1..i] <= a[i+1] <= ... <= a[n] oraz heap(1,i).  
for (int i = n; i >= 2; i--) {  
    swap(a[1], a[i]);  
    DownHeap(1, i - 1);  
}
```

Cechy:

- w miejscu (+);
- nie jest stabilny (-);
- pesymistyczny czas działania $\Theta(n \log n)$ (+).

■ Merge sort

Inaczej sortowanie przez scalanie.

```
// b[1..n] - globalna tablica pomocnicza  
// Niezmiennik: przy wywołaniu funkcji posortowane jest a[1..s] oraz a[s+1..r].  
// Po wywołaniu posortowane będzie a[1...r].  
void merge(int l, int r, int s) {  
    i = l;  
    j = s + 1;  
    k = l - 1;  
    while (i <= s && j <= r) {  
        k = k + 1;  
        if (a[i] <= a[j]) {      // #porównania <= r - l  
            b[k] = a[i];  
            i = i + 1;  
        }  
        else {  
            b[k] = a[j];  
            j = j + 1;  
        }  
    }  
    if (i > s)  
        b[k+1...r] = a[j...r]  
    else  
        b[k+1...r] = a[i...s]  
    a[l...r] = b[l...r]  
}  
  
void merge_sort(int l, int r) {  
    if (l < r) {  
        s = (l + r) / 2;  
        merge_sort(l, s);  
        merge_sort(s + 1, r);  
        merge(l, r, s);  
    }  
}
```

```

    }
}

merge_sort(1, n);

```

Cechy:

- nie jest w miejscu (-) aczkolwiek istnieje trudniejsza wersja w miejscu;
- stabilny (+);
- pesymistyczny czas działania $\Theta(n \log n)$ (+);
- bardzo mało porównań (+);
- sporo przypisań, możliwa redukcja (-/+).

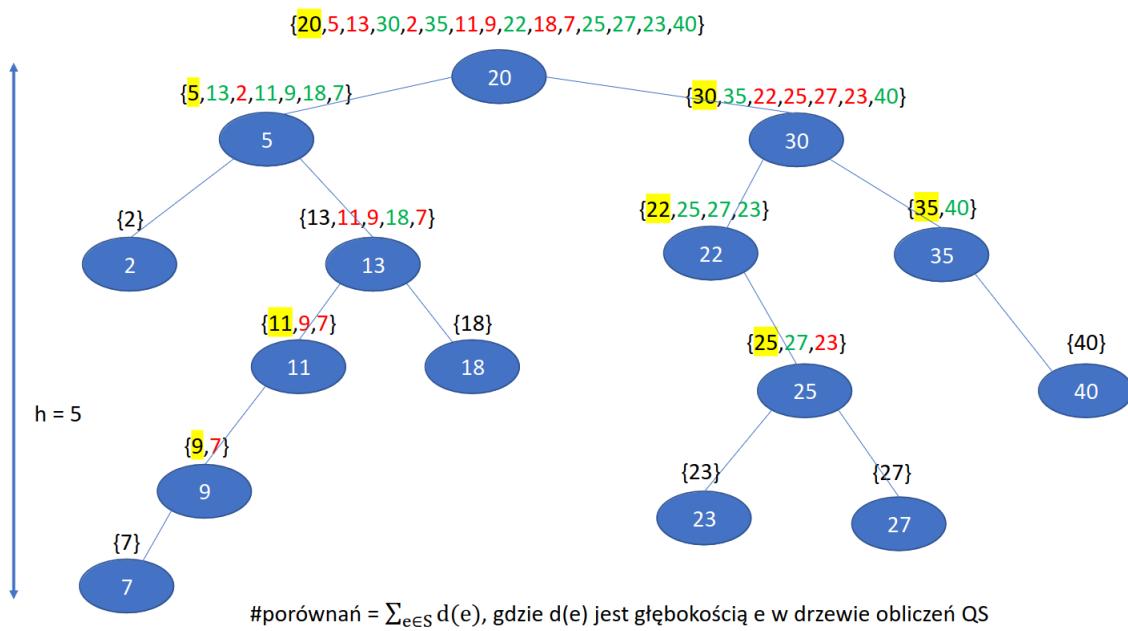
Quick sort

Intuicyjnie jest to algorytm, które po prostu buduje drzewo BST.

```

// Niech S będzie zbiorem, który chcemy posortować.
// elem S zwraca wszystkie elementy zbioru S.
void quick_sort(S) {
    if (|S| <= 1) {
        print(elem S);                                // jedyny element zbioru S
    }
    else {
        x = pivot(S);                                // element dzielący
        S_less = {y: elem S && y < x};             // zbiór elementów mniejszych niż x
        S_greater = {y: elem S && y > x};          // zbiór elementów większych niż x
        quick_sort(S_less);
        print(x);
        quick_sort(S_greater);
    }
}

```



Rysunek 6.3: Ilustracja działania algorytmu quick sort.

W powyższym przykładzie jako pivot wybieramy zawsze pierwszy element sortowanego zbioru. Takie drzewo obliczeń może być niskie (wysokości $O(\log n)$) lub wysokie (np. dla $S = \{1, 2, \dots, n\}$). Przy losowym wyborze pivota algorytm ten działa jednak w oczekiwanej złożoności $\Theta(n \log n)$.

Cechy:

- prawie w miejscu (-/+);
- nie jest stabilny (-);
- pesymistyczna złożoność $\Theta(n^2)$ (-);
- oczekiwana złożoność $\Theta(n \log n)$ (+);
- bardzo dobrze sprawdza się w praktyce.

■ Count sort

Inaczej sortowanie przez zliczanie. Działa dobrze przy założeniu, że elementy tablicy a nie są zbyt duże.

```
// b - tablica zliczająca liczbę wystąpień elementów a
m = max(a) + 1;
b = [0] * m;
for (int i = 1; i <= n; i++)
    b[a[i]]++;

// Dla każdej wartości tablicy a wyznaczamy ostatnią pozycję,
// na której wystąpi ona w posortowanej tablicy.
for (int i = 1; i <= m - 1; i++)
    b[i] += b[i - 1];

// t - tablica pomocnicza
for (int i = n; i >= 1; i--) {
    t[b[a[i]]] = a[i];
    b[a[i]]--;
}

a = t;
```

Cechy:

- nie jest w miejscu (-);
- stabilny (+);
- złożoność $\Theta(n + m)$ (-/+);
- dla $m = O(n)$ algorytm liniowy (+).

■ Bucket sort

Inaczej sortowanie kubełkowe. Idea taka sama jak w count sortie, ale utrzymujemy tablicę listami zamiast zwykłego counter'a.

```
// Inicjacja pustych kubełków.
m = max(a) + 1;
b = {} * m;

// Wypełnienie kubełków.
for (int i = 1; i <= n; i++)
    b[a[i]].push_back(a[i]);
```

```
// Konicowe scalenie
wynik = {};
for (int i = 0; i <= m - 1; i++)
    wynik.append(b[i]);
```

Cechy:

- nie jest w miejscu (-);
- stabilny (+);
- złożoność $\Theta(n + m)$ (-/+);
- dla $m = O(n)$ algorytm liniowy (+).

■ Sortowanie leksykograficzne słów tej samej długości

Dane:

- dodatnie liczby całkowite n, k, m ;
- $s[1..n]$ - tablica słów o długości k nad alfabetem $\{0, 1, \dots, m - 1\}$;
- oznaczenie: $s[i][j]$ – j -ta litera w słowie $s[i]$.

Wynik:

- $w[1..n]$ – permutacja indeksów $1, \dots, n$ wyznaczająca porządek leksykograficzny słów s .

```
w[1..n] = [1, 2, ..., n];
for (int j = k; j >= 1; j--) {
    posortuj w stabilnie w względem znaków z pozycji j w słowach;
}
```

Do posortowania stabilnie możemy wykorzystać bucket sort.

Cechy:

- stabilny (+);
- złożoność $\Theta(k \cdot (n + m))$ (-/+).

■ Sortowanie leksykograficzne

Poprzedni algorytm da się uogólnić na sortowanie leksykograficzne słów różnej długości.

Cechy:

- złożoność $\Theta(k \cdot n + m)$ (-/+).

■ Inne algorytmy sortowania

- Metoda Shella, usprawnienie insertion sorta, h -sortowania.
- Sortowanie introspektywne - quick sort z pesymistyczną złożonością $O(n \log n)$.

■ Ważny fakt

Każdy algorytm sortujący **przez porównania** wykonuje w pesymistycznym przypadku $\Theta(n \log n)$ porównań.

6.3. W modelu losowej permutacji, złożoność $\Omega(n^2)$ ma algorytm

- A. quick sort
- B. insertion sort
- C. heap sort

6.4. Pesymistyczna złożoność algorytmów sortowania w zależności od liczby inwersji inv to

- A. dla insertion sort: $O(\max(n, inv))$
- B. dla quick sort: $O(n \cdot \log(\max(n, inv)))$
- C. dla heap sort: $O(n \cdot \log(\max(n, inv)))$

6.5. Ciąg $\delta = \langle \delta_1, \delta_2, \dots, \delta_n \rangle$ nazywamy k -uporządkowanym rosnąco, gdy każdy jego podciąg złożony z elementów odległych o k jest uporządkowany rosnąco, tzn. $\delta_i < \delta_{i+k}$ dla $i = 1, 2, \dots, n - k$. Ciąg δ można uporządkować za pomocą $O(n)$ porównań, gdy jest on

- A. jednocześnie 2- i 3-uporządkowany
- B. 2012-uporządkowany
- C. $\lfloor \log n \rfloor$ -uporządkowany

6.6. Rozważmy równanie rekurencyjne

$$T(n) = \begin{cases} n & \text{dla } n \leq 1, \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n + 1 & \text{dla } n > 1 \end{cases}$$

Wartość $T(n)$

- A. jest ograniczeniem górnym na liczbę porównań wykonywanych w algorytmie sortowania n różnych elementów przez scalanie
- B. jest ograniczeniem górnym na średnią liczbę porównań w algorytmie sortowania szybkiego dla n różnych elementów, gdy element dzielący jest wybierany losowo z rozkładem jednostajnym
- C. jest rzędu $\Theta(n \log n)$

6.4.

Algorytmy grafowe

Algorytm Dijkstry w spójnym grafie z krawędziami o nieujemnych wagach szuka najkrótszych ścieżek od wyróżnionego wierzchołka do wszystkich pozostałych.

Pseudokod:

```

/*
 * s - wierzchołek startowy
 * L - zbiór wierzchołków, dla których znamy już najkrótsze ścieżki
 * R - dopełnienie L
 * w(u, v) - waga krawędzi u -> v
 * N(v) - sąsiedzi wierzchołka v */
begin
    /* inicjacja */
    d(s) := 0, L := {s}, R := V - {s};
    for v in R do d(v) := +inf;
    for v in N(s) do d(v) := w(s, v);

```

```

/* pętla główna */
while |R| > 0 do
begin
    v := wierzchołek w R o najmniejszej wadze d
    R := R - {v};
    L := L + {v};
    for u in (R and N(v)) do d(i) := min(d(u), d(v) + w(v, u));
end
end

```

Złożoność czasowa algorytmu Dijkstry to $O(m \cdot \text{DecreaseKey} + n \cdot \text{DeleteMin})$, gdzie m to liczba krawędzi w grafie, n to liczba wierzchołków, a koszt DecreaseKey oraz DeleteMin zależy od wykorzystanej struktury danych.

To było na egzaminie

Złożoność algorytmu Dijkstry dla spójnego grafu o n wierzchołkach i m krawędziach w implementacji

- A. z kopcem zupełnym to $O(m \log n)$
- B. z kopcem dwumianowym to $O(m + n\sqrt{\log n})$
- C. z kopcem Fibonacciego to $O(m + n \log n)$

Złożoność algorytmu Dijkstry to $O(m \cdot \text{DecreaseKey} + n \cdot \text{DeleteMin})$. W każdym kopcu operacja DeleteMin ma koszt $O(\log n)$, podobnie jak DecreaseKey, które jedynie w przypadku kopca Fibonacciego amortyzuje się do czasu stałego.

Zauważmy również, że dla spójnego grafu m jest co najmniej rzędu n , a maksymalnie rzędu n^2 .

- A. Czas będzie $O(m \log n + n \log n)$, ale ponieważ graf jest spójny, to ma co najmniej $\Omega(n)$ krawędzi, więc drugi składnik sumy nic nie wnosi do złożoności i odpowiedzią jest TAK.
- B. Złożoność taka sama, jak dla kopca zupełnego ($O(m \log n)$), więc odpowiedź to NIE.
- C. Wstawiając powyższe złożoności kopca Fibonacciego do wzoru otrzymamy dokładnie taką złożoność, odpowiedź to TAK.

Algorytmy przeszukiwania grafu

Przeszukiwanie grafu wszerz (ang. *Breadth First Search – BFS*) pozwala nam znaleźć najkrótsze ścieżki (w grafie bez wag na krawędziach) w czasie $O(n + m)$, gdzie n to liczba wierzchołków a m to liczba krawędzi w grafie.

```

/* s - wierzchołek startowy */
deque<int> S = {s}; /* kolejka FIFO */
vector<int> D(n, -1); /* tablica odległości */
D[s] = 0;
while (!S.empty()) {
    int v = S.front();
    S.pop_front();
    for (int u : adj[v]) {
        if (D[u] == -1) {
            D[u] = D[v] + 1;
            S.push_back(u);
        }
    }
}

```

Przeszukiwanie grafu w głąb (ang. *Depth First Search – DFS*) otrzymujemy, poprzez zastąpienie kolejki FIFO w BFS-ie stosem. Można go efektywnie zapisać rekurencyjnie:

```
vector<bool> vis(n); /* tablica odwiedzonych */
void dfs(int v) {
    vis[v] = true;
    for (int u : adj[v]) {
        if (!vis[u])
            dfs(u);
    }
}
```

Krawędzie, którymi przechodziliśmy przy przeszukiwaniu grafu, tworzą **drzewo rozpinające** tego grafu.

Drzewo BFS grafu jest drzewem najkrótszych ścieżek w tym grafie.

Drzewo DFS grafu ma ważną właściwość. Wszystkie krawędzie niedrzewowe łączą przodka z potomkiem lub w drugą stronę. Nigdy nie idą „w poprzek” drzewa – mogły by być wtedy wykorzystane przez algorytm DFS.

Zestaw zadań

6.7. Dany jest graf G o 10 wierzchołkach z wyróżnionym wierzchołkiem r . Drzewo rozpinające utworzone przejściem DFS od wierzchołka r ma wysokość h (największa liczba krawędzi od korzenia do liścia). Prawdą jest, że

- A. jeśli $h = 8$, graf G ma co najwyżej 44 krawędzie
- B. jeśli $h = 2$, graf G ma co najwyżej 15 krawędzi
- C. rozpinające drzewo powstałe algorytmem BFS ma zawsze mniejszą wysokość niż h

6.8. Niech n będzie liczbą całkowitą większą od 1. Wynika z tego, że

- A. wysokość (czyli największa liczba krawędzi od korzenia do liścia) drzewa BFS w grafie dwudzielnym $K_{n,n}$ wynosi 2
- B. wysokość drzewa DFS w grafie dwudzielnym $K_{n,n}$ wynosi n
- C. jeżeli $n > 10$, to istnieje spójny graf n -wierzchołkowy o n krawędziach, dla którego drzewa BFS i DFS o tym samym korzeniu są takie same

6.9. G jest 1001-wierzchołkowym grafem dwuspójnym wierzchołkowo. Wynika z tego, że

- A. jeżeli wysokość pewnego drzewa DFS grafu G wynosi 1000, to G ma cykl Hamiltona
- B. wysokość każdego drzewa BFS grafu G jest nie większa niż 500
- C. wysokość każdego drzewa DFS grafu G jest różna od 2

6.10. Koszt wykonania algorytmu Dijkstry dla spójnego grafu n -wierzchołkowego o m krawędziach wynosi

- A. $O(m \log n)$ w implementacji z kopcem zupełnym
- B. $O(n \log n + m)$ w implementacji z kopcem Fibonacciego
- C. $O(n^2)$ w implementacji z kolejką dwumianową

6.11. Wysokość drzewa ukorzenionego mierzymy liczbą krawędzi na najdłuższej ścieżce z korzenia do wierzchołka w tym drzewie. Dany jest spójny n -wierzchołkowy graf dwuspójny wierzchołkowo (tzn. bez wierzchołków rozdzielających) o co najmniej 4 wierzchołkach. Prawdą jest, że

- A. wysokość każdego drzewa przeszukiwania wszerz w takim grafie jest mniejsza od $n/2$
- B. wysokość każdego drzewa przeszukiwania w głąb w takim grafie jest większa od $n/2$
- C. wysokość każdego drzewa przeszukiwania w głąb w takim grafie wynosi co najmniej 3

6.12. Wysokość drzewa BST zawierającego 100 wierzchołków

- A. wynosi co najmniej 49
- B. wynosi co najmniej 7
- C. wynosi co najwyżej 99

6.5.

Drzewa BST

Drzewa wyszukiwań binarnych (drzewa BST – Binary Search Trees) – drzewa binarne, w których elementy rozmieszczone są w porządku symetrycznym: dla każdego wierzchołka, wartość w nim (klucz) jest większa od wszystkich wartości w jego lewym poddrzewie oraz mniejsza od wszystkich wartości w jego prawym poddrzewie.

To było na egzaminie

Rozważmy drzewo BST budowane poprzez kolejne wstawianie pewnej permutacji ciągu liczb $1, 2, \dots, 7$. Permutacji takich że powstanie drzewo o wysokości

- A. 6 jest 64
- B. 5 jest 32
- C. 2 jest 80

- A. Aby uzyskać BST o wysokości 6, w każdym kroku budowania drzewa musimy wstawić wierzchołek z najmniejszą lub największą wartością, która nie jest jeszcze wykorzystana (np. w pierwszym kroku wstawimy 1 lub 7). Takich permutacji możemy utworzyć $2^6 = 64$, więc odpowiedź jest prawdziwa.
- B. Rozważmy utworzenie drzewa BST o wysokości 5 w następujący sposób: na początku wstawmy wartość 2, a następnie utwórzmy ścieżkę analogicznie jak w podpunkcie A. z wartości $\{3, 4, 5, 6, 7\}$. Wartość 1 możemy wstawić w dowolne miejsce permutacji, ale później niż wartość 2. To już daje $2^4 \cdot 6 > 32$ opcje, więc odpowiedź to NIE.
- C. Aby otrzymać pełne drzewo binarne, musimy najpierw wstawić wartość 4. Dodatkowo musimy wstawić wartość 2 przed 1 i 3. Analogicznie musimy wstawić wartość 6 przed 5 i 7. Mamy więc $\binom{6}{3} \cdot 2 \cdot 2 = 80$ opcji: wybieramy pozycje dla trójki $\{1, 2, 3\}$, następnie dowolnie permutujemy 1 z 3 oraz 5 z 7. Odpowiedź to TAK.

Drzewo AVL

Drzewo AVL to drzewo wyszukiwań binarnych, w którym dla każdego węzła wysokości jego poddrzew różnią się o co najwyżej 1.

Z tego faktu wynika, że wysokość drzewa jest $O(\log n)$.

Implementacja polega na tym, że przy każdej zmianie w drzewie wykonujemy rotacje, żeby drzewo było znowu zbalansowane.

Przypis redakcji

Trzeba tu dopisać wzory na maksymalną i minimalną liczbę wierzchołków drzewa AVL (pojawia się to w praktycznie każdym zadaniu).

To było na egzaminie

Wysokość drzewa wyszukiwań binarnych mierzymy liczbą krawędzi na najdłuższej ścieżce od korzenia do liścia (węzła z kluczem bez następców). Wysokość drzew z jednym kluczem wynosi 0. Wysokość drzewa AVL z 2022 kluczami

- A. wynosi co najmniej 11
 - B. wynosi co najwyżej 22
 - C. będzie maksymalna, jeśli klucze zostaną wstawione do początkowo pustego drzewa w kolejności rosnącej
-
- A. Dla danej wysokości h maksymalna liczba wierzchołków w AVL drzewie to liczba wierzchołków w pełnym drzewie binarnym, czyli $2^{h+1} - 1$. Mając $2022 < 2047 = 2^{10+1} - 1$ wierzchołki, widzimy, że minimalna wysokość w naszym przypadku będzie równa 10, odpowiedź jest więc fałszywa.
 - B. Najmniejsze liczby wierzchołków tworzą ciąg Fibonacciego dany wzorem $F_n = F_{n-1} + F_{n-2} + 1$: dla $h = 0, 1, 2, 3, 4, \dots$ mamy $\min_n = 1, 2, 4, 7, 12, \dots$, bo mając minimalne drzewa dla $n - 2$ i $n - 1$, chcąc stworzyć minimalne AVL drzewo dla n , rysujemy korzeń i podłączamy te dwa drzewa jako prawego i lewego syna, dostając wspomniany wzór. Licząc na palcach (bo ciąg ten rośnie dość szybko), możemy zauważyc, że z 1596 wierzchołkami dostaniemy maksymalnie wysokość 14, a z 2583 – 15. Wynika stąd, że dla 2022 wierzchołków dostaniemy najwyżej wysokość 14, czyli w szczególności co najwyżej 22 – odpowiedź to TAK.
 - C. Gdy zaczniemy wstawiać klucze w kolejności rosnącej, to szybko zauważymy, że bynajmniej nie maksymalizujemy wysokości drzewa.

Drzewo splay

Wykonanie podstawowych operacji na tym drzewie wiąże się z wykonaniem procedury LocalSplay(x), która powoduje zmianę struktury drzewa, że węzeł x zostaje umieszczony w korzeniu (przy zachowaniu porządku symetrycznego).

Koszt zamortyzowany tej funkcji jest $O(\log n)$ (pesymistycznie $\Theta(n)$), i taki też koszt mają podstawowe operacje.

Drzewo czerwono-czarne

Drzewo czerwono-czarne - drzewo BST, w którym każdy węzeł jest pokolorowany na czerwono lub czarno, zgodnie z następującymi regułami:

- korzeń drzewa jest czarny
- każdy czerwony węzeł ma czarnego rodzica
- każdy węzeł zewnętrzny (NULL) jest czarny
- każda ścieżka (elementarna) z ustalonego węzła do węzła zewnętrznego w jego poddrzewie zawiera tyle samo węzłów czarnych

Z tych warunków wynika, że wysokość drzewa jest $O(\log n)$.

Zestaw zadań

6.13. O drzewach AVL prawdą jest, że

- A. największe drzewo AVL o wysokości 5 ma 64 wierzchołki
- B. najmniejsze drzewo AVL o wysokości 5 ma 20 wierzchołków
- C. każde drzewo czerwono-czarne jest AVL drzewem

6.14. Do początkowo pustego drzewa BST wstawiamy kolejno elementy pewnej permutacji liczb $1, 2, \dots, 16$. Liczba permutacji, dla których

- A. dostaniemy drzewo o wysokości (czyli maksymalnej liczbie krawędzi na ścieżce od korzenia do liścia) równej 15, wynosi co najwyżej 2^{16}
- B. dostaniemy drzewo o wysokości równej 3, wynosi co najmniej 2^{15}
- C. w lewym poddrzewie korzenia będzie tylko węzeł z kluczem 1, wynosi co najmniej jeden miliard

6.15. W zbiorze AVL-drzew o 10 węzłach

- A. istnieje drzewo o wysokości (czyli największej liczbie krawędzi od korzenia do liścia) równej 4
- B. każde drzewo ma wysokość co najmniej 3
- C. maksymalna różnica liczb węzłów poddrzew korzenia wynosi 5

6.6.

Algorytmy tekstowe

Po dokładne opisy algorytmów odsyłam do przedmiotowych slajdów, poniżej znajdują się tylko podstawowe fakty, definicje oraz intuicja.

Haszowanie

Służy do reprezentacji elementów pewnego dużego zbioru (klucze) poprzez elementy mniejszego zbioru. Częstym przypadkiem jest zamiana słów na liczby. Oznaczmy duży zbiór przez U . Słownik S mapujący klucze na wartości implementujemy w tablicy $a[0..m - 1]$, gdzie m jest dużo mniejsze niż rozmiar U . Tworzymy funkcję haszującą $h : U \rightarrow \{0, \dots, m - 1\}$, która odwzorowuje klucze w przestrzeń adresową. S implementujemy w tablicy list $a[0..m - 1]$, gdzie $a[i]$ to lista wszystkich kluczy k , dla których $h(k) = i$. Tablicę, w której implementujemy słownik nazywamy tablicą z haszowaniem (tablicą haszowaną).

Powiemy, że dwa różne klucze e i f są ze sobą w kolizji, gdy $h(e) = h(f)$.

Cechy dobrej funkcji haszującej:

- równomiernie haszuje: losowo wybrany klucz jest z jednakowym prawdopodobieństwem odwzorowywany na każdą z m pozycji, niezależnie od tego, gdzie zostają odwzorowane inne klucze
- łatwo (szybko) obliczalna

Najbardziej znanym haszowaniem jest haszowanie modularne: $h(k) = k \bmod m$, gdzie m jest liczbą pierwszą daleką od potęgi dwójki.

■ Algorytm KMP

Służy do rozwiązywania problemu wyszukiwania wzorca w tekście. Niech Σ – skończony, niepusty alfabet oraz wzorzec $w[1..m]$, tekst $t[1..n]$ – słowa nad alfabetem Σ , $1 \leq m \leq n$. Należy znaleźć wszystkie wystąpienia wzorca w w tekście t , tzn. należy znaleźć wszystkie takie i , $1 \leq i \leq n - m + 1$, że $w[1..m] = t[i..i+m-1]$.

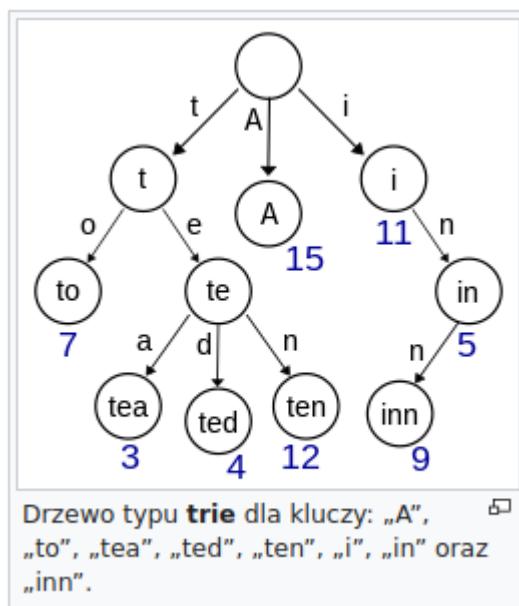
Przykład 1.

Niech $w = [babab]$ oraz $t = [abbabababababab]$. Wzorzec w występuje w tekście t na pozycjach 3, 5 i 10.

Algorytm KMP działa w czasie $O(n + m)$. Zauważmy również, że wzorzec może wystąpić w tekście tylko jeśli jest niedłuższy od tekstu, a więc algorytm działa w czasie $O(n)$.

■ Drzewo trie

Drzewo trie to drzewo poszukiwań przechowujące w węzłach fragmenty kluczy. To pozwala przyspieszyć wyszukiwanie, gdy koszt porównania całego klucza jest duży. Znakomicie sprawdzają się więc do utrzymywania np. zbioru słów.



Rysunek 6.4: Przykładowe drzewo trie

Działania jakie można zrealizować za pomocą drzew trie:

- sprawdzenie, czy słowo długości k jest w drzewie w czasie $O(k)$;
- znalezienie najdłuższego prefiksu słowa występującego w drzewie w czasie $O(m)$, gdzie m jest długością prefiksu;
- wyszukanie wszystkich słów o podanym prefiksie.

■ Drzewo i tablica sufiksowa

Są to struktury reprezentujące oraz porządkujące wszystkie sufiksy danego słowa. Dla słowa długości n klasyczne drzewo sufiksowe działa w czasie liniowym $O(n)$, natomiast klasyczna tablica sufiksowa kosztuje nas dodatkowy czynnik logarytmiczny. Możliwe jest jednak zbudowanie tablicy sufiksowej w czasie $O(n)$.

Przykłady problemów, które można szybko rozwiązać przy pomocy tych struktur zbudowanych na słowie t :

- znajdź wszystkie wystąpienia danego słowa s w t ;
- znajdź najdłuższe słowo, które pojawia się w t co najmniej 2 razy;
- znajdź najdłuższe wspólne podsłowo dla t oraz danego słowa s ;
- policz ilość różnych podsłów t .

Zestaw zadań

6.16. Niech $\Sigma = \{a, b\}$ będzie dwuznakowym alfabetem, a s i t słowami nad Σ o długościach odpowiednio m i n , gdzie $0 < m \leq n$. W algorytmie KMP wyszukiwania wzorca s w tekście t

- A. wszystkie wystąpienia słowa s w słowie t zostaną znalezione w czasie $O(n)$
- B. preprocessing zajmuje czas $\Omega(n)$
- C. każdy symbol ze słowa t jest porównywany z co najwyżej jednym symbolem ze słowa s

6.7.

Co to

Zestaw zadań

6.17. Dana jest funkcja

```
int coto(int n) {
    if (n == 0)
        return 1;
    else if (n > 0)
        return coto(n - 1) + coto(-n);
    else
        return coto(n + 1) - 1;
}
```

Przypisanie $y = coto(x)$ spowoduje, że będzie zachodzić zależność

- A. $|y| \geq |x|$
- B. $y \leq 1$
- C. jeśli $x = -2012$, to $y = -2011$

6.8.

Rozwiązańia

Rozwiązańia

6.1. Agata i Bartek grają w grę polegającą na tym, że Agata wybiera dwie liczby ze zbioru $\{1, 2, \dots, n\}$, a Bartek zadaje pytania o wybrany podzbiór i dowiaduje się, czy co najmniej jedna z wybranych liczb znajduje się w nim. Liczba zapytań wystarczających do określenia przez Bartka obu liczb jest

- A. $O(n)$
- B. $O(\sqrt{n})$
- C. $O(\log n)$

Oczywistym jest, że najbardziej optymalnym algorytmem w tym przypadku jest wyszukiwanie binarne o złożoności $O(\log n)$. Ponieważ zachodzi $\log n = 2 \log \sqrt{n} \leq 2\sqrt{n}$ oraz $\sqrt{n} \leq n$, to prawdziwe jest także $O(\log n) \subseteq O(\sqrt{n}) \subseteq O(n)$. Zatem liczba zapytań rzędu $O(\log n)$ spełnia jednocześnie $O(\sqrt{n})$ oraz $O(n)$.

- 6.2.** Do początkowo pustego kopca Fibonacciego typu min wstawiamy kolejno rekordy o priorytetach odpowiednio 1, 2, ..., 2020, otrzymując kopiec T , a następnie wykonujemy operację `DeleteMin`, otrzymując kopiec T' . Wynika z tego, że

- NIE** A. kopiec T składa się z co najwyżej 8 drzew
TAK B. kopiec T' składa się z co najwyżej 8 drzew
TAK C. najwyższe drzewo w kopcu T' ma wysokość (tzn. największą możliwą liczbę krawędzi od korzenia do liścia w drzewie) co najmniej 10
- A. Kopiec Fibonacciego jest reprezentowany przez listę dwukierunkową, przy wstawianiu elementu dodajemy do listy jednoelementowe drzewo. Zatem kopiec T składa się z 2020 drzew.
B. Po usunięciu minimum trzeba znaleźć kolejne, ta operacja jednak łączy drzewa tych samych rozmiarów (póki istnieją). Uzyskujemy wtedy tyle drzew ile występuje w zapisie binarnym liczby $2019 = 11111100011_2$. Są to więc odpowiednio drzewa o 1024, 512, 256, 128, 64, 32, 2 i 1 elementach, łącznie jest ich osiem.
C. Z poprzedniego podpunktu wynika, że najwyższe drzewo ma 1024 elementy, ponieważ jest binarne, to jego wysokość to $\log 1024 = 10$.

- 6.3.** W modelu losowej permutacji, złożoność $\Omega(n^2)$ ma algorytm

- NIE** A. quick sort
TAK B. insertion sort
NIE C. heap sort

Zapis $\Omega(n^2)$ oznacza ograniczenie dolne tempa wzrostu funkcji przez n^2 . Wiemy, że zarówno quick sort, jak i heap sort zadziałają w czasie $n \log n$, więc niekoniecznie w n^2 . Jednak insertion sort w modelu losowej permutacji zawsze wykona n^2 kroków.

- 6.4.** Pesymistyczna złożoność algorytmów sortowania w zależności od liczby inwersji inv to

- TAK** A. dla insertion sort: $O(\max(n, inv))$
NIE B. dla quick sort: $O(n \cdot \log(\max(n, inv)))$
TAK C. dla heap sort: $O(n \cdot \log(\max(n, inv)))$
- A. Złożoność algorytmu insertion sort zależy od występującej liczby inwersji. Jeżeli jest ona mniejsza niż n , to algorytm działa liniowo, gdyż przechodzi po całej tablicy i dokonuje $< n$ zamian.
B. Najbardziej pesymistyczny przypadek liczby inwersji to $\frac{n(n-1)}{2}$. Gdyby rzeczywiście złożoność była zależna od liczby inwersji w taki sposób, jak podano w podpunkcie B., algorytm quick sort miałby pesymistyczną złożoność $n \log n^2$ – a w rzeczywistości ma $n \log n$.
C. Analogicznie jak w podpunkcie B.

- 6.5.** Ciąg $\delta = \langle \delta_1, \delta_2, \dots, \delta_n \rangle$ nazywamy k -uporządkowanym rosnąco, gdy każdy jego podciąg złożony z elementów odległych o k jest uporządkowany rosnąco, tzn. $\delta_i < \delta_{i+k}$ dla $i = 1, 2, \dots, n - k$. Ciąg δ można uporządkować za pomocą $O(n)$ porównań, gdy jest on

- TAK** A. jednocześnie 2- i 3-uporządkowany
TAK B. 2012-uporządkowany
NIE C. $\lfloor \log n \rfloor$ -uporządkowany
- (a) Skoro ciąg jest 2-uporządkowany, to możemy podzielić go na dwie tablice – o parzystych indeksach i nieparzystych – i połączyć jak w merge sorcie w czasie $O(n)$.

- (b) Podobnie jak w A., tylko mamy 2012 tablic, ale wciąż jest to czas liniowy.
(c) W tym przypadku to nie zadziała, ponieważ mamy $\log n$ tablic i w każdym z n kroków musimy wybrać minimalny element z $\log n$ elementów, więc czas będzie $O(n \log n)$.

6.6. Rozważmy równanie rekurencyjne

$$T(n) = \begin{cases} n & \text{dla } n \leq 1, \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n + 1 & \text{dla } n > 1 \end{cases}$$

Wartość $T(n)$

- TAK** A. jest ograniczeniem górnym na liczbę porównań wykonywanych w algorytmie sortowania n różnych elementów przez scalanie
NIE B. jest ograniczeniem górnym na średnią liczbę porównań w algorytmie sortowania szybkiego dla n różnych elementów, gdy element dzielący jest wybierany losowo z rozkładem jednostajnym
TAK C. jest rzędem $\Theta(n \log n)$

- A. Merge sort dzieli tablicę na pół, wykonuje się rekurencyjnie na pierwszej oraz drugiej połowie, a następnie scalą obie połówki, porównując elementy z pierwszej i drugiej. Założymy, że $T(n)$ to koszt merge sorta dla tablicy n -elementowej. W takim razie $T(\lfloor \frac{n}{2} \rfloor)$ to koszt merge sorta dla połowy tablicy. Zatem równanie podane w zadaniu dokładnie odpowiada temu, co robi merge sort ($\pm(n+1)$ porównań na końcu, ale w treści mamy, że jest to ograniczenie górne).
B. Taka złożoność dla quick sorta byłaby prawidłowa, gdybyśmy zawsze wybierali medianę jako pivot, co nie jest prawdą przy rozkładzie jednostajnym.
C. Skoro w podpunkcie A. stwierdziliśmy, że jest to złożoność merge sorta, to faktycznie jest rzędem $\Theta(n \log n)$, bo merge sort ma dokładnie taką złożoność.

6.7. Dany jest graf G o 10 wierzchołkach z wyróżnionym wierzchołkiem r . Drzewo rozpinające utworzone przejęciem DFS od wierzchołka r ma wysokość h (największa liczba krawędzi od korzenia do liścia). Prawdą jest, że

- TAK** A. jeśli $h = 8$, graf G ma co najwyżej 44 krawędzie
NIE B. jeśli $h = 2$, graf G ma co najwyżej 15 krawędzi
NIE C. rozpinające drzewo powstałe algorytmem BFS ma zawsze mniejszą wysokość niż h
- A. Graf pełny o 10 wierzchołkach posiada 45 krawędzi, a drzewo rozpinające (DFS) w tym przypadku ma zawsze wysokość 9. Niemożliwe jest więc, aby drzewo o wysokości 8 rozpiniało graf pełny 10-wierzchołkowy, w związku z czym ograniczenie górne wynosi 44.
B. Kontrprzykład: graf, którym wierzchołek r jest połączony krawędzią z s , oraz każdy z nich posiada krawędzie do każdego z pozostałych 8 wierzchołków, co daje $1 + 2 \cdot 8 = 17$ krawędzi (rysunek w zadaniu 9). Istnieje drzewo rozpinające (DFS) ten graf o wysokości 2 (korzeń s ma tylko jedno dziecko - r , które ma 8 dzieci), co spełnia warunki zadania.
C. Jeżeli graf G jest drzewem, to oba drzewa rozpinające są tej samej wysokości.

6.8. Niech n będzie liczbą całkowitą większą od 1. Wynika z tego, że

- TAK** A. wysokość (czyli największa liczba krawędzi od korzenia do liścia) drzewa BFS w grafie dwudzielnym $K_{n,n}$ wynosi 2
NIE B. wysokość drzewa DFS w grafie dwudzielnym $K_{n,n}$ wynosi n
NIE C. jeżeli $n > 10$, to istnieje spójny graf n -wierzchołkowy o n krawędziach, dla którego drzewa BFS i DFS o tym samym korzeniu są takie same

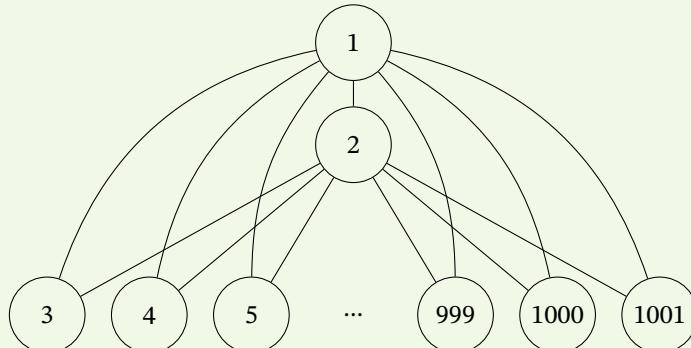
Graf $K_{n,n}$ to dwudzielna klika, czyli wszystkie wierzchołki z jednej składowej są połączone z wszystkimi wierzchołkami z drugiej składowej.

- A. Bez straty ogólności możemy założyć, że zaczynamy BFS-a z wierzchołka pierwszej składowej i w pierwszej iteracji pokrywamy wszystkie wierzchołki drugiej składowej. W kolejnej iteracji BFS-a pokrywamy wszystkie pozostałe wierzchołki z pierwszej składowej. Wykonaliśmy dwie iteracje, więc wysokość takiego drzewa to 2.
- B. DFS będzie przechodzić między wierzchołkami pierwszej i drugiej składowej, a przez to, że graf jest kliką, drzewo DFS będzie listą o długości $2n - 1$ krawędzi – bo tyle potrzeba do połączenia $2n$ wierzchołków.
- C. Spójny graf o n wierzchołkach i n krawędziach musi zawierać w sobie cykl (wiemy to z własności drzew). Gdy istnieje cykl, to drzewo DFS jest zawsze głębsze od drzewa BFS.

6.9. G jest 1001-wierzchołkowym grafem dwuspójnym wierzchołkowo. Wynika z tego, że

- NIE** A. jeżeli wysokość pewnego drzewa DFS grafu G wynosi 1000, to G ma cykl Hamiltona
- TAK** B. wysokość każdego drzewa BFS grafu G jest nie większa niż 500
- NIE** C. wysokość każdego drzewa DFS grafu G jest różna od 2

- A. Kontrprzykładem jest graf będący ścieżką wierzchołków ponumerowanych od 1 do 1001, posiadający dodatkowo krawędzie $\{1, 1000\}$ i $\{2, 1001\}$. Wtedy da się przejść DFS po całym grafie, nie cofając się (uzyskując wysokość 1000), ale graf nie ma cyklu Hamiltona.
- B. W grafie dwuspójnym wierzchołkowo każde dwa wierzchołki leżą na jakimś cyklu. Maksymalizując wysokość drzewa BFS rozpatrzmy graf będący cyklem 1001 wierzchołkowym. Startując z wierzchołka 1, przeszukując drzewo wszerz, dostaniemy wysokość 500 i jest to maksymalna wysokość jaką możemy uzyskać.
- C. Rozważmy graf jak z rysunku poniżej. Drzewo DFS startujące z wierzchołka o numerze 1 i wchodzące najpierw do wierzchołka numer 2 będzie miało wysokość równą 2.



6.10. Koszt wykonania algorytmu Dijkstry dla spójnego grafu n -wierzchołkowego o m krawędziach wynosi

- TAK** A. $O(m \log n)$ w implementacji z kopcem zupełnym
- TAK** B. $O(n \log n + m)$ w implementacji z kopcem Fibonacciego
- NIE** C. $O(n^2)$ w implementacji z kolejką dwumianową

Złożoność algorytmu Dijkstry to $O(m \cdot \text{DecreaseKey} + n \cdot \text{DeleteMin})$. W każdym kopcu operacja `DeleteMin` ma koszt $O(\log n)$, podobnie jak `DecreaseKey`, które jedynie w przypadku kopca Fibonacciego amortyzuje się do czasu stałego.

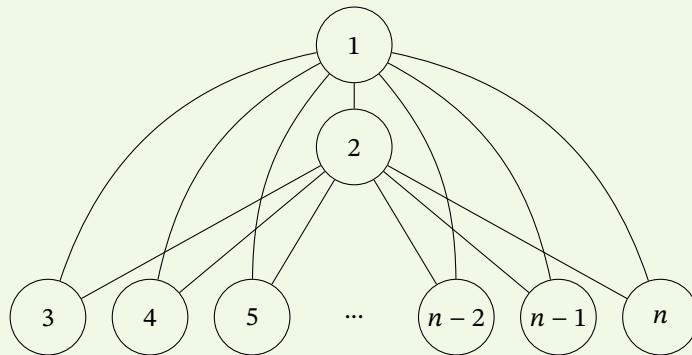
- A. Czas będzie $O(m \log n + n \log n)$, ale ponieważ graf jest spójny, to ma co najmniej $\Omega(n)$ krawędzi, więc drugi składnik sumy nic nie wnosi do złożoności i odpowiedzią jest TAK.
- B. Wstawiając powyższe złożoności kopca Fibonacciego do wzoru otrzymamy dokładnie taką złożoność, odpowiedź to TAK.
- C. Złożoność jest taka sama jak w A. ($O(m \log n)$), a m może być rzędu n^2 , więc odpowiedź to NIE.

- 6.11.** Wysokość drzewa ukorzenionego mierzmy liczbą krawędzi na najdłuższej ścieżce z korzenia do wierzchołka w tym drzewie. Dany jest spójny n -wierzchołkowy graf dwuspojny wierzchołkowo (tzn. bez wierzchołków rozdzielających) o co najmniej 4 wierzchołkach. Prawdą jest, że

- NIE** A. wysokość każdego drzewa przeszukiwania wszerz w takim grafie jest mniejsza od $n/2$
- NIE** B. wysokość każdego drzewa przeszukiwania w głęb w takim grafie jest większa od $n/2$
- NIE** C. wysokość każdego drzewa przeszukiwania w głęb w takim grafie wynosi co najmniej 3

Kontrprzykładem w **A.** może być cykl 4-wierzchołkowy. W takim cyklu głębokość drzewa BFS jest równa $2 = \frac{n}{2}$ (niezależnie od wierzchołka startowego).

Kontrprzykład do **B.** oraz **C.** narysowany jest poniżej (wierzchołek o numerze 1 jest korzeniem w drzewie, a DFS wchodzi najpierw do wierzchołka numer 2):



- 6.12.** Wysokość drzewa BST zawierającego 100 wierzchołków

- NIE** A. wynosi co najmniej 49
- NIE** B. wynosi co najmniej 7
- TAK** C. wynosi co najwyższej 99

Drzewo BST o największej wysokości to prosta z naniesionymi kolejnymi wierzchołkami, zatem wysokość takiego drzewa jest równa liczbie wierzchołków pomniejszonej o jeden (w naszym przypadku 99).

Drzewo BST z najmniejszą liczbą wierzchołków, to drzewo, w którym wierzchołki na każdym poziomie (oprócz ostatniego) mają po dwóch synów. Zatem wysokość takiego drzewa będzie wynosiła $\lfloor \log_2(n) \rfloor$, gdzie n to liczba wierzchołków (w naszym przypadku 6).

- 6.13.** O drzewach AVL prawdą jest, że

- NIE** A. największe drzewo AVL o wysokości 5 ma 64 wierzchołki
 - TAK** B. najmniejsze drzewo AVL o wysokości 5 ma 20 wierzchołków
 - NIE** C. każde drzewo czerwono-czarne jest AVL drzewem
- A. Największe drzewo AVL to po prostu największe drzewo BST, czyli drzewo posiadające dwóch synów na każdym poziomie. Zatem największe drzewo AVL o wysokości h będzie posiadać $2^{h+1} - 1$ wierzchołków (czyli w naszym przypadku 63).
 - B. Rozwiązanie tego podpunktu dość łatwo narysować na kartce, dokładając kolejno potrzebne wierzchołki. Można też skorzystać ze wzoru $M(h) = 1 + M(h-1) + M(h-2)$, gdzie $M(h)$ to minimalna liczba wierzchołków w drzewie AVL o wysokości h .
 - C. Nie, ponieważ drzewo czerwono-czarne nie musi być zbalansowane, tak jak drzewo AVL.

- 6.14.** Do początkowo pustego drzewa BST wstawiamy kolejno elementy pewnej permutacji liczb $1, 2, \dots, 16$. Liczba permutacji, dla których

TAK A. dostaniemy drzewo o wysokości (czyli maksymalnej liczbie krawędzi na ścieżce od korzenia do liścia) równej 15, wynosi co najwyżej 2^{16}

NIE B. dostaniemy drzewo o wysokości równej 3, wynosi co najmniej 2^{15}

TAK C. w lewym poddrzewie korzenia będzie tylko węzeł z kluczem 1, wynosi co najmniej jeden miliard

A. Żeby zbudować drzewo o takiej (maksymalnej) wysokości, w permutacji muszą występować kolejno największa lub najmniejsza aktualnie możliwa liczba, np. 16, 1, 2, 3, 15, 4, 14, 13, 12, 5, 6, 11, Na każdym kroku mamy wybór pomiędzy dwoma liczbami, więc takich permutacji będzie $2^{15} \leq 2^{16}$.

B. Drzewo o wysokości 3 może mieć maksymalnie 15 elementów, więc jest 0 takich permutacji.

C. Aby w lewym poddrzewie była tylko jedynka, pierwszym elementem permutacji musi być „2”, reszta może być losowo. Zatem takich permutacji jest $15!$. Wiemy że $7! > 1000 = 10^3$ oraz $15 \cdot 14 \cdot 13 \cdot 12 \cdot 11 \cdot 10 \cdot > 10^6$ więc $15! > 10^9$.

6.15. W zbiorze AVL-drzew o 10 węzłach

NIE A. istnieje drzewo o wysokości (czyli największej liczbie krawędzi od korzenia do liścia) równej 4

TAK B. każde drzewo ma wysokość co najmniej 3

TAK C. maksymalna różnica liczb węzłów poddrzew korzenia wynosi 5

A. Minimalna liczba wierzchołków w AVL drzewie o wysokości 4 wynosi 12, więc mając 10 wierzchołków możemy uzyskać co najwyżej wysokość 3.

B. Pełne drzewo binarne o wysokości 2 ma 7 wierzchołków, więc mając ich 10 możemy zbudować drzewo co najmniej o wysokości równej 3.

C. Maksymalną różnicę dostaniemy, jeśli jednym poddrzewem korzenia będzie wierzchołek mający jednego syna, który jest liściem, a drugie poddrzewo korzenia będzie pełnym drzewem binarnym o wysokości równej 2. Wtedy lewe poddrzewo ma 2 wierzchołki, a prawe – 7.

6.16. Niech $\Sigma = \{a, b\}$ będzie dwuznakowym alfabetem, a s i t słowami nad Σ o długościach odpowiednio m i n , gdzie $0 < m \leq n$. W algorytmie KMP wyszukiwania wzorca s w tekście t

TAK A. wszystkie wystąpienia słowa s w słowie t zostaną znalezione w czasie $O(n)$

TAK B. preprocessing zajmuje czas $\Omega(n)$

NIE C. każdy symbol ze słowa t jest porównywany z co najwyżej jednym symbolem ze słowa s

A. Algorytm KMP działa w czasie $O(n)$.

B. Trzeba policzyć tablicę prefikso-sufiksów, co zajmuje czas liniowy.

C. Nie musi tak być, np. mając $s = abab$ i $t = ababab$, prefikso-sufiks słowa s wynosi 2, więc po każdym wystąpieniu wzorca w tekście cofniemy się o 2, więc niektóre symbole z t porównamy dwa razy.

6.17. Dana jest funkcja

```
int coto(int n) {
    if (n == 0)
        return 1;
    else if (n > 0)
        return coto(n - 1) + coto(-n);
    else
        return coto(n + 1) - 1;
}
```

Przypisanie $y = coto(x)$ spowoduje, że będzie zachodzić zależność

NIE A. $|y| \geq |x|$

TAK B. $y \leq 1$

TAK C. jeśli $x = -2012$, to $y = -2011$

Prześledźmy, co robi funkcja `coto(x)`. Jeśli $x = 0$, to **wyznacza** wartość 1. Jeśli $x < 0$, to funkcja **oblicza** wartość $|x| \cdot (-1) + 1 = -|x| + 1 = x + 1$. Jeśli $x > 0$, to funkcja **przekazuje** wartość rekurencji $F_n = F_{n-1} - n + 1$. Ostatecznie,

$$\text{coto}(x) = \begin{cases} 1 & \text{gdy } x = 0 \\ \text{coto}(x-1) - x + 1 & \text{gdy } x > 0 \\ x + 1 & \text{gdy } x < 0 \end{cases} = \begin{cases} \text{coto}(x-1) - x + 1 & \text{gdy } x > 0 \\ x + 1 & \text{wpp.} \end{cases}$$

- A. Widzimy, że wstawiając za x ujemną liczbę (np. -5) dostaniemy w wyniku liczbę większą o 1 (np. -4), czyli w module będzie to liczba mniejsza o 1, a więc fałsz.
- B. Dla liczb niedodatnich widać od razu. Dla liczb dodatnich można by policzyć tą rekurencję, ale można też rozpisać kilka pierwszych wyrazów ciągu: $(1, 0, -2, -5, -9, \dots)$. Już dla $x > 2$ dostajemy liczby ujemne, a dla $x = 0$ i $x = 1$ coś nie większego od 1, czyli odpowiedź to TAK
- C. Widać od razu – dla liczby ujemnej dostajemy liczbę większą o 1.

PS. Istnieje w języku polskim niezbyt szczerliwe tłumaczenie słowa „*return*”, jako „*zwracać*”. Jest to niezbyt zręczna kalka językowa bez specjalnego uzasadnienia. Zamiast mówić „funkcja zwraca wartość” proponuję „funkcja {wyznacza, oblicza, przekazuje} wartość”. Autor rozwiązania trzymał się konwencji poznanej na Wstępie do Programowania Imperatywnego autorstwa docenta Piotra Chrząstowskiego-Wachtla i zachęca, aby Czytelnik również się do niej stosował.

7

Języki, automaty i obliczenia

Materiały teoretyczne z tego przedmiotu zostały opracowane na podstawie skryptu Szymona Toruńczyka.

Podstawa programowa

1. **Języki regularne**, wyrażenia regularne, automaty skończone.
2. **Języki bezkontekstowe**, gramatyki bezkontekstowe, automaty ze stosem.
3. **Lematy o pompowaniu** dla języków regularnych i bezkontekstowych.
4. **Języki obliczalne** i częściowo obliczalne. Problem stopu oraz metoda przekątniowa.
5. **Klasy P, NP** oraz NP-zupełność.

7.1.

Języki regularne

Automat niedeterministyczny (NFA) to krotka $\langle A, Q, I, F, \delta \rangle$, gdzie:

- A to alfabet, czyli skończony zbiór symboli zwanych dalej *literami*,
- Q to skończony zbiór stanów,
- $I \subseteq Q$ to zbiór stanów początkowych,
- $F \subseteq Q$ to zbiór stanów akceptujących,
- $\delta \subseteq Q \times A \times Q$ to relacja przejścia.

Automat deterministyczny (DFA) to prawie taka sama krotka $\langle A, Q, I, F, \delta \rangle$ jak powyżej, ale z tą różnicą, że δ to funkcja typu $Q \times A \rightarrow Q$, a nie tylko relacja.

Automat z ε -przejściami to automat dopuszczający przejścia postaci (p, ε, q) , gdzie ε to słowo puste.

Terminologia:

- *tranzycja (przejście)* – trójka $(p, a, q) \in \delta$, również oznaczana jako $p \xrightarrow{a} q$,
- *bieg* – ciąg tranzycji $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} q_n$, gdzie $q_0 \in I$ i który można interpretować jako ścieżkę w grafie automatu,
- *bieg akceptujący* – bieg, który kończy w stanie akceptującym,
- *bieg po słowie w* – bieg $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} q_n$ taki, że $a_1 a_2 a_3 \dots a_n = w$.
- automat \mathcal{A} akceptuje słowo $w \in A^*$, jeżeli istnieje bieg akceptujący po tym słowie,

- automat \mathcal{A} rozpoznaje język L , jeśli $w \in L \Leftrightarrow \mathcal{A}$ akceptuje w i oznaczamy to jako $L = \mathcal{L}(A)$.

Wyrażenie regularne to wyrażenie zbudowane rekurencyjnie, które reprezentuje pewien język. Tak samo jak dla automatów, dla wyrażenia regularnego A reprezentującego język L piszemy, że $L = \mathcal{L}(A)$. Można je budować przy pomocy następujących reguł:

- \emptyset jest wyrażeniem regularnym reprezentującym język pusty,
- a jest wyrażeniem regularnym reprezentującym język $A = \{a\}$,
- dla wyrażeń A, B : $A + B$ reprezentuje język $\mathcal{L}(A) \cup \mathcal{L}(B)$,
- dla wyrażeń A, B : $A \cdot B$ (lub po prostu AB) reprezentuje język $K \cdot L = \{v \cdot w \mid v \in K, w \in L\}$, gdzie $K = \mathcal{L}(A), L = \mathcal{L}(B)$,
- dla wyrażenia A : A^* reprezentuje język $\{w_1 \cdot w_2 \cdot \dots \cdot w_n \mid w_1, w_2, \dots, w_n \in \mathcal{L}(A), n \in \mathbb{N}\}$.

Prawdziwy jest następujący **lemat o równoważności automatów i wyrażeń regularnych**: niech $L \subseteq A^*$. Następujące warunki są równoważne:

- język L jest reprezentowany przez pewne wyrażenie regularne,
- język L jest rozpoznawany przez pewien automat niedeterministyczny,
- język L jest rozpoznawany przez pewien automat deterministyczny,
- język L jest rozpoznawany przez pewien automat (nie)deterministyczny z ϵ -przejściami.

Języki regularne są zamknięte na:

- sumę** – dla języków regularnych K i L , język $K \cup L$ jest regularny.
- konkatenację** – dla języków regularnych K i L , język $K \cdot L = \{w_K \cdot w_L : w_K \in K, w_L \in L\}$ jest regularny.
- gwiazdkę Kleene'ego** – dla języka regularnego L , język $L^* = \{w_1 \cdot w_2 \cdot \dots \cdot w_n : w_1, w_2, \dots, w_n \in L, n \in \mathbb{N}\}$ jest regularny.
- przecięcie** – dla języków regularnych K i L , język $K \cap L$ jest regularny.
- dopełnienie** – dla języków regularnych K i L , język $K - L$ jest regularny.
- odwrócenie języka** – Dla języka regularnego L , język $L^R = \{w^R \mid w \in L\}$ jest regularny.

■ Minimalizacja automatów

Ponieważ wiele automatów może rozpoznawać ten sam język regularny, możemy rozważyć **problem minimalizacji** – jaka jest najmniejsza liczba stanów, jaką musi posiadać automat, żeby rozpoznawał dany język? Rozważany automat będzie automatem *deterministycznym* i będzie nazywany *automatem syntaktycznym*. Do znalezienia takiego automatu przydatna będzie relacja Myhill-Nerodego:

$$u \sim_L v \Leftrightarrow \forall w \in A^* : (uw \in L \Leftrightarrow vw \in L)$$

Możemy interpretować tę relację następująco: słowa u, v są w relacji jeśli mają tę samą „przyszłość” – zaczynając od wczytania u , wczytywanie dalszych liter doprowadza nas do tego samego rezultatu, jak gdybyśmy zaczęli od v . Relację tę z problemem minimalizacji łączy twierdzenie:

Jeśli język L ma skończenie wiele klas abstrakcji w sensie relacji Myhill-Nerodego, to jest językiem regularnym. Ponadto, automat syntaktyczny dla tego języka posiada dokładnie tyle stanów, ile było klas abstrakcji.

Przykład 1.

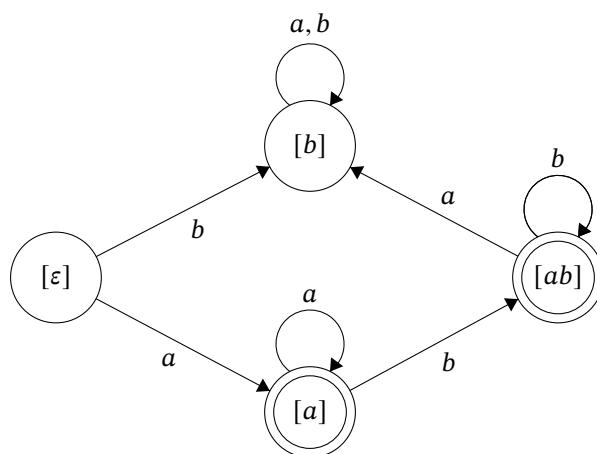
Znajdziemy liczbę stanów automatu syntaktycznego rozpoznającego język $L = aa^*b^*$. W tym celu znajdziemy wszystkie klasy abstrakcji relacji Myhillla-Nerodego nad tym językiem. Pierwszą klasą abstrakcji będzie $[\epsilon]$. Metodą prób i błędów znajdziemy następne klasy abstrakcji.

Rozpatrzmy $[a]$ i sprawdźmy, czy jest to klasa abstrakcji różna od $[\epsilon]$. Musimy znaleźć słowo w , które jest różnymi „przyszłościami” dla a i ϵ – czyli takie, że tylko jedno ze słów aw i $\epsilon w = w$ należy do L . Łatwo zauważyc, że działa $w = \epsilon$. Zatem $[a] \neq [\epsilon]$.

Spróbujemy znaleźć następną klasę. Rozpatrzmy $[b]$. Zauważmy, że jeśli na początek wczytamy b , to „nie mamy przyszłości” – nieważne co wczytamy dalej, nie osiągniemy akceptacji. Łatwo widać, że jest to klasa różna od poprzednich.

Ostatnią klasą abstrakcji będzie $[ab]$ – po wczytaniu słowa ab , aby dostać akceptację możemy wczytywać już tylko same litery b , co nie było prawdą dla poprzednich klas.

Mogliśmy próbować szukać następnych klas, ale okazywało się będzie, że są równe znalezionym już przez nas klasom. Aby się upewnić, można narysować automat języka L , kładąc klasy abstrakcji jako stany:



Rzeczywiście, jest to automat deterministyczny rozpoznający powyższy język. Zatem minimalny automat ma 4 stany.

■ Lemat o pompowaniu dla języków regularnych

Przypuśćmy, że język $L \subseteq A^*$ jest regularny. Wówczas istnieje taka stała $N \in \mathbb{N}$, że dla każdego słowa $w \in L$ długości co najmniej N istnieje dekompozycja $w = w_1 \cdot w_2 \cdot w_3$ o następujących własnościach:

- słowo w_2 jest niepuste,
- $|w_1 \cdot w_2| \leq N$,
- dla dowolnej liczby naturalnej $k \geq 0$ słowo $w_1 \cdot w_2^k \cdot w_3$ należy do języka L .

Przykład 2.

Pokażemy, że język $L = \{a^n b^n \mid n \geq 0\}$ nie jest regularny. Przypuśćmy przeciwnie. Wówczas istnieje stała $N \in \mathbb{N}$, o której mowa w lematie o pompowaniu. Rozważmy słowo $w = a^N b^N$. To słowo należy do języka L , więc istnieje dekompozycja $w = w_1 \cdot w_2 \cdot w_3$, taka, że $w_2 \neq \epsilon$, $w_1 \cdot w_2^k \cdot w_3 \in L$ dla $k \geq 0$, oraz $|w_1 \cdot w_2| \leq N$. Wynika stąd, że słowo w_2 zawiera wyłącznie litery a oraz jest niepuste. Słowo $w_1 \cdot w_2 \cdot w_3$ zawiera więcej liter a niż liter b , więc nie należy do języka L , co jest sprzecznością. Otrzymana sprzeczność pokazuje, że język L nie jest regularny.

- 7.1.** Rozważmy wyrażenie regularne $K = A^*bbA^* + A^*aAA + A^*aA$ nad alfabetem $\{a, b\}$ (zapis A w wyrażeniu to skrót na $(a+b)$). Niech L to język tego wyrażenia. Wtedy
- A. automat minimalny dla L ma co najwyżej 4 stanów
 - B. automat minimalny dla L ma przynajmniej 6 stanów
 - C. istnieje automat niedeterministyczny dla L o 5 stanach
- 7.2.** Dany jest język regularny L dany wyrażeniem regularnym $(a+b)^*a(a+b)(a+b)$. Wówczas
- A. każdy automat deterministyczny rozpoznający L ma co najmniej 9 stanów
 - B. minimalny automat deterministyczny rozpoznający L ma 7 stanów
 - C. każdy automat niedeterministyczny rozpoznający L ma co najmniej 6 stanów
- 7.3.** Regularny jest język nad alfabetem $\{a, b\}$ złożony ze wszystkich słów, w których każde podsłowo
- A. długości 3 występuje parzystą liczbę razy
 - B. długości większej niż 3 występuje parzystą liczbę razy
 - C. występuje również jako prefiks
- 7.4.** Regularny jest język złożony ze wszystkich słów nad alfabetem $\{a, b, c\}$, które
- A. zaczynają się od $baca$
 - B. nie zawierają $baca$ jako podsłowa
 - C. zawierają $baca$ jako podsłowo parzystą liczbę razy
- 7.5.** L jest regularny. Czy regularny jest:
- A. $\{w : w \in L \wedge w \in L^R\}$
 - B. $\{ww : w \in L\}$
 - C. prefiksy słów z L parzystej długości
- 7.6.** Językiem regularnym jest
- A. język $L \subseteq \{1\}^*$ składający się z unarnych reprezentacji liczb pierwszych
 - B. język $L \subseteq \{0, 1\}^*$ składający się ze słów, w których liczba wystąpień cyfry 0 przystaje modulo 2 do liczby wystąpień cyfry 1
 - C. język $L \subseteq \{0, 1\}^*$ składający się z binarnych reprezentacji liczb podzielnych przez 7

7.2.

Języki bezkontekstowe

Gramatyka bezkontekstowa \mathcal{G} ma następujące składniki:

- Zbiór *terminali* A ,
- Zbiór *nieterminali* N , rozwłączny z A ,
- Zbiór *produkcji* postaci $q \rightarrow w$, gdzie $q \in N$ oraz $w \in (A \cup N)^*$ jest słowem składającym się z terminali i/lub nieterminali.
- Symbolu *startowego* S

Drzewem parsowania takiej gramatyki jest drzewo etykietowane, w którym:

- etykieta korzenia to symbol startowy S
- liście etykietowane są terminalami ze zbioru $A \cup \{\varepsilon\}$,
- wierzchołki wewnętrzne etykietowane są nieterminalami ze zbioru N ,
- jeżeli wierzchołek wewnętrzny ma etykietę q , to ma k dzieci o etykietach $s_1, \dots, s_k \in (N \cup A)$ (czytając od lewej do prawej) oraz $q \rightarrow s_1 s_2 \dots s_k$ jest produkcją gramatyki (wyjątek: zamiast $k = 0$ robimy dziecko o etykiecie ε)

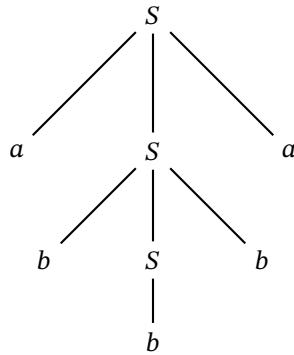
Plon drzewa to słowo utworzone z etykiet liści, czytanych od lewej do prawej.

Przykład 1.

Rozważmy gramatykę \mathcal{G} o terminalach a oraz b , nieterminalu S oraz produkcjach

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$$

Drzewo parsowania dla słowa $abbba$:



Przez indukcję po rozmiarze drzewa parsowania widać, że plonem każdego drzewa parsowania jest palindrom, tj. takie słwo w , że $w = w^R$. Odwrotna inkluza również zachodzi, co możemy udowodnić (ponownie przez indukcję) konstruując drzewo parsowania dla każdego palindromu. Tak więc, $L(\mathcal{G})$ to język palindromów nad alfabetem $\{a, b\}$.

Lemat 1. *Każdy język regularny jest bezkontekstowy.*

Szkic dowodu: dla każdego automatu, da się skonstruować gramatykę, która mu odpowiada.

■ Lemat o pompowaniu dla języków bezkontekstowych

Lemat 2. *Przypuśćmy, że język $L \subseteq A^*$ jest bezkontekstowy. Wtedy istnieje takie $N \in \mathbb{N}$, że każde słwo $w \in L$ długości co najmniej N posiada faktoryzację*

$$w = \text{prefix} \cdot \text{left} \cdot \text{infix} \cdot \text{right} \cdot \text{suffix}$$

o następujących własnościach:

- słowo $\text{left} \cdot \text{right}$ jest niepuste
- słowo $\text{left} \cdot \text{infix} \cdot \text{right}$ ma długość co najwyżej N
- dla każdego $k \geq 0$ słowo $w_k = \text{prefix} \cdot \text{left}^k \cdot \text{infix} \cdot \text{right}^k \cdot \text{suffix}$ należy do języka L .

Przykład 2.

Pokażemy, że język $L = \{a^n b^n c^n \mid n \geq 0\} \subseteq \{a, b, c\}^n$ nie jest bezkontekstowy. Przypuśćmy przeciwnie. Wtedy istnieje stała $N \in \mathbb{N}$ o której mowa w lemacie o pompowaniu. Rozważmy słowo $w = a^N b^N c^N$ i jego faktoryzację. Gdyby *left* zawierał dwie różne litery (tj. a i b , lub a i c , lub b i c), to słowo w_2 nie mogłoby należeć do języka L . Zatem, *left* używa tylko jednej litery i analogicznie *right*. Zatem jedna z liter a, b, c , nazwijmy ją x , nie pojawia się ani w słowie *left*, ani w słowie *right*. Ponieważ przynajmniej jedno ze słów *left*, *right* jest niepuste, to jakaś litera y różna od x pojawia się w *left* lub *right*. Słowo w_2 ma więc więcej wystąpień litery y niż litery x , więc $w_2 \notin L$. To przeczy tezie lematu o pompowaniu. Tak więc, język L nie jest bezkontekstowy.

Zauważmy, że język L jest przecięciem dwóch języków bezkontekstowych: $\{a^n b^n c^m : n, m \geq 0\}$ oraz $\{a^n b^m c^m : n, m \geq 0\}$. To przykład na to, że języki bezkontekstowe nie są zamknięte na przecięcia, a w konsekwencji nie są też zamknięte na dopełnienie.

Języki bezkontekstowe są zamknięte na:

- sumę, konkatenację, gwiazdkę Kleene'go, odwrócenie,
- przecięcia z językami regularnymi – jeśli L jest językiem bezkontekstowym, a K jest językiem regularnym, to $L \cap K$ jest językiem bezkontekstowym.

■ Automaty ze stosem

Języki bezkontekstowe są rozpoznawane przez pewien rodzaj automatów, nazywanych *automatami ze stosem*. Są one uogólnieniem NFA z ϵ -przejściami, ale z dodatkowym *stosem*. Automat taki ma następujące składniki:

- Alfabet wejściowy A ,
- Alfabet stosowy Γ zawierający symbol dna stosu,
- Zbiór stanów Q ,
- Zbiór stanów początkowych $I \subseteq Q$,
- Zbiór stanów akceptujących $F \subseteq Q$,
- Zbiór przejść δ , gdzie każde jest postaci:

$$p \xrightarrow{\text{pop}(Z), a, \text{push}(\gamma)} q$$

dla pewnych $p, q \in Q$, $a \in A \cup \{\epsilon\}$, $Z \in \Gamma$, $\gamma \in \Gamma^*$.

Oczywiście **pop**(Z) usuwa literę Z z góry stosu, zaś **push**(γ) wrzuca na stos słowo γ (zakładamy, że wrzucenie całego słowa na stos działa tak samo jak wrzucanie po kolej liter od pierwszej do ostatniej – innymi słowy, stos rośnie w prawo i **push** dokleja słowo do stosu).

Ważne fakty o automatach ze stosem:

- Automaty ze stosem mają tę samą siłę wyrazu co języki bezkontekstowe,
- Zamiast rozważać stany akceptujące można rozważać model, w którym akceptacja zachodzi wtedy, gdy stos jest pusty – te dwa modele są równoważne,
- Każdy automat ze stosem można przerobić na równoważny, mający tylko jeden stan i akceptujący poprzez pusty stos.

7.7. Założymy, że L to język bezkontekstowy, zaś K to język regularny,oba nad alfabetem A . Przez X^R oznaczamy język powstały z X przez odwrócenie wszystkich słów z X . Językiem bezkontekstowym jest język

- A. $L^R \cap K$
- B. $(A^* - L) \cup K^*$
- C. $L^* \cup K^R$

7.8. Dane są język bezkontekstowy L i język regularny K . Prawdą jest, że

- A. $K \setminus L$ jest bezkontekstowy
- B. $K \cap L$ jest regularny
- C. $L \setminus K$ jest bezkontekstowy

7.9. Niech L będzie nieskończonym językiem bezkontekstowym nad skończonym alfabetem. Wówczas

- A. L jest regularny
- B. L zawiera nieskończony język regularny
- C. L zawiera słowo postaci $uv^{2020}xy^{2020}z$, gdzie $vy \neq \varepsilon$

7.10. Niech L, K to języki bezkontekstowe. Bezkontekstowy wtedy też jest język

- A. $L \cup K$
- B. $L \cap K$
- C. $L \cap a^*b^*$

7.11. Dla języka L , niech $\Psi(L)$ oznacza zbiór długości słów z języka L . Istnieje taki język bezkontekstowy L , dla którego $\Psi(L)$ jest zbiorem

- A. liczb naturalnych podzielnych przez 11
- B. kwadratów liczb naturalnych
- C. sześcianów liczb naturalnych

7.12. L_1 i L_2 są językami bezkontekstowymi nad alfabetem Σ . Wynika z tego, że bezkontekstowy jest język

- A. $L_1 \cap L_2$
- B. $\Sigma^* - L_1$
- C. $L_1 \cup L_2$

7.13. L jest językiem bezkontekstowym. Wynika z tego, że bezkontekstowy jest język

- A. $\{v : \text{istnieje słowo z } L, \text{ które można otrzymać przez usunięcie parzystej liczby (niekoniecznie kolejnych) liter z } v\}$
- B. $\{ww : w \in L\}$
- C. $\{a_1 \cdots a_n : a_n \cdots a_1 \in L\}$ (tzn. słowa z L pisane do tyłu)

7.14. Następujący język jest bezkontekstowy:

- A. $\{a^n b^m c^k d^l : n = k \text{ i } m = l\}$
- B. $\{a^n b^m c^k d^l : n = l \text{ i } m = k\}$
- C. $\{a^n b^m c^k d^l : n = m \text{ i } k = l\}$

7.15. Jeśli L_1 jest językiem bezkontekstowym, a L_2 jest językiem regularnym, to bezkontekstowy jest język

- A. $L_1 \cap L_2$
 B. $\{vw : v \in L_1, w \in L_2\}$
 C. $\{v^R : v \in L_1\}$, gdzie v^R oznacza odwrócenie słowa v

7.16. Zbiór dziesiętnych zapisów liczb podzielnych przez 7 jest

- A. językiem regularnym
 B. językiem bezkontekstowym
 C. językiem kontekstowym

7.3. Języki obliczalne

Maszyna Turinga to krotka $M = \langle A, B, Q, q_I, q_F, \delta \rangle$, gdzie:

- A - alfabet wejściowy
- B - alfabet roboczy (taki, którym posługuje się maszyna)
- Q - skończony zbiór stanów
- q_I - stan początkowy
- q_F - stan końcowy
- δ - funkcja przejścia, gdzie $\delta : Q \times B \rightarrow B \times \{\leftarrow, \rightarrow\} \times Q$

Charakterystyka podstawowej maszyny Turinga:

- działa na jednostronnie nieskończonym ciągu komórek nazywanym **taśmą**
- posiada głowicę poruszającą się wzdłuż taśmy
- głowica wskazuje na aktualnie rozważaną komórkę taśmy
- może wykonać tranzycję $t = (p, a, b, d, q)$ w następujący sposób:
 - maszyna znajduje się w stanie p
 - w aktualnie rozważanej komórce znajduje się litera a
 - wykonując tranzycję w miejsce litery a wpisana zostaje litera b
 - głowica zostaje przesunięta w kierunku $d \in \{\leftarrow, \rightarrow\}$
 - maszyna przechodzi do stanu q

Intuicyjnie maszyna Turinga M jest w stanie sprawdzać swój aktualny stan i na jego podstawie decydować, co zamierza zrobić dalej, być może zmieniając zawartość taśmy. Po więcej szczegółowych definicji odsyłam do [skryptu](#).

Równoważne warianty maszyn Turinga:

- zwykła maszyna Turinga opisana powyżej;
- maszyna Turinga z binarnym alfabetem roboczym ($B = \{0, 1, _\}$);
- maszyna Turinga z taśmą nieskończoną w obu kierunkach;
- maszyna Turinga z wieloma taśmami;

- niedeterministyczna maszyna Turinga.

Funkcje obliczalne: oznaczmy przez $L(M)$ zbiór słów $w \in A^*$, na których maszyna M terminuje. Oznaczmy przez $f_M(w) : A^* \rightarrow B^*$ funkcję częściową, która słowi $w \in L(M)$ przyporządkowuje wynik obliczenia M na w , a dla pozostałych słów jest nieokreślona. Mówimy, że M oblicza funkcję f_M . Funkcja częściowa $f : A^* \rightarrow B^*$ jest **obliczalna** jeśli istnieje maszyna Turinga, która ją oblicza.

Przykład 1.

Rozważmy słowa $w \in \{0, 1\}^*$ reprezentujące grafy. Funkcja f ma za zadanie zwrócić jeden wtedy i tylko wtedy, gdy w jest opisem grafu spójnego. Taka funkcja f jest obliczalna, ponieważ możemy zasymulować na taśmie maszyny Turinga algorytm BFS (zapisując m. in. zbiór odwiedzonych wierzchołków i wykonując odpowiednie przejścia).

Język $L \subseteq A^*$ jest **obliczalny**, jeżeli jego funkcja charakterystyczna jest obliczalna. Jeżeli język L jest obliczalny, to odpowiadający mu problem decyzyjny jest **rozstrzygalny**. Innymi słowami problem decyzyjny jest rozstrzygalny jeśli istnieje algorytm, który dla każdego słowa wejściowego w terminuje oraz odpowiada "TAK" lub "NIE" w zależności od tego czy $w \in L$. Przykładem języków obliczalnych są języki regularne oraz bezkontekstowe. Języki obliczalne są zamknięte na:

- sumy;
- przecięcia;
- dopełnienia.

Język $L \subseteq A^*$ jest **częściowo obliczalny**, jeżeli istnieje maszyna Turinga, która terminuje dla wszystkich słów $w \in L$, a dla słów $w' \in A^* - L$ nie terminuje. Jeżeli język L jest częściowo obliczalny, to odpowiadający mu problem decyzyjny jest **półrozstrzygalny**. Innymi słowami problem decyzyjny jest półrozstrzygalny jeśli istnieje algorytm, który dla każdego słowa wejściowego w odpowiada "TAK" wtedy i tylko wtedy, gdy $w \in L$, a w przeciwnym wypadku się wiesza. Każdy język obliczalny jest też częściowo obliczalny. Języki częściowo obliczalne są zamknięte na:

- sumy;
- przecięcia.

Lemat 1. Język $L \subseteq A^*$ jest obliczalny wtw. gdy L oraz $A^* - L$ są częściowo obliczalne.

Lemat 2. Języki częściowo obliczalne to dokładnie klasa języków rozpoznawanych przez maszyny Turinga.

Problem stopu – inaczej nazywany problemem **HALT** – dla danej maszyny Turinga M oraz słowa w stwierdź, czy M terminuje na w . Do języka HALT należą więc wszystkie pary $\langle M, w \rangle$ takie, że M terminuje na w . Język HALT jest częściowo obliczalny (ale nie jest obliczalny). Dopełnienie języka HALT nie jest częściowo obliczalne.

Przydatnym wariantem powyższego problemu jest problem HALT_ε , czyli język maszyn Turinga terminujących na słowie pustym. Nie jest on obliczalny. Aby to udowodnić, wykorzystamy metodę **redukcji**.

Przykład 2.

Załóżmy, że dana jest maszyna M oraz słowo wejściowe w . Niech nowa maszyna M_w działa w następujący sposób: M_w wymazuje swoje wejście zastępując je słowem w , a następnie symuluje działanie M na w . Maszyna M_w zatrzyma się na słowie pustym wtedy, gdy M zatrzyma się na słowie w . Zatem

- dla danego kodu k maszyny M oraz słowa $w \in A^*$ istnieje kod k_w , że $k \in \text{HALT} \Leftrightarrow k_w \in \text{HALT}_\varepsilon$
- funkcja $(k, w) \rightarrow k_w$ jest obliczalna

Z tego wynika, że HALT_ε nie jest obliczalny.

Formalnie, **redukcja** problemu $L \subseteq A^*$ do problemu $K \subseteq B^*$ to funkcja $f : A^* \rightarrow B^*$ taka, że $\forall w \in A^*$ zachodzi $w \in L \Leftrightarrow f(w) \in K$. Redukcja f jest obliczalna jeżeli f jest obliczalna. Oznacza to w szczególności, że jeżeli istnieje redukcja obliczalna języka L do K oraz K jest obliczalny, to L również jest obliczalny.

Znane problemy nierostrzygalne:

- problem stopu
- czy język bezkontekstowy L rozpoznaje każde słowo $w \in A^*$?
- czy język bezkontekstowy L zawiera się w języku bezkontekstowym K ?

7.4. Klasy P, NP oraz NP-zupełność

Język L jest w klasie P jeżeli istnieje **deterministyczna** maszyna Turinga M oraz wielomian $f : \mathbb{N} \rightarrow \mathbb{N}$ takie, że:

- $L(M) = L$;
- dla każdego słowa $w \in A^*$ maszyna M wykonuje co najwyżej $f(|w|)$ kroków na słowie w .

Mniej formalnie: $L \in P$ jeżeli istnieje algorytm rozwiązujący problem L w czasie wielomianowym.

Język L jest w klasie NP jeżeli istnieje **niedeterministyczna** maszyna Turinga M oraz wielomian $f : \mathbb{N} \rightarrow \mathbb{N}$ takie, że:

- $L(M) = L$;
- dla każdego słowa $w \in A^*$, każdy bieg maszyny M po słowie w wykonuje co najwyżej $f(|w|)$ kroków.

Oczywiście $P \subseteq NP$, ale czy $NP \subseteq P$? Jest to bardzo ważny problem otwarty. Wiemy natomiast, że równoważne są warunki:

- $P = NP$
- któryś z problemów z poniższej listy należy do klasy P
- każdy z problemów z poniższej listy należy do klasy P

Lista niektórych problemów NP-zupełnych:

- CNF-SAT
To problem spełnialności formuł w postaci CNF, a więc koniunkcji wielu klauzul, gdzie klauzula jest alternatywą literałów. Literał to zmienna lub negacja zmiennej.
- 3CNF-SAT
Instancja problemu CNF-SAT, w której każda klauzula składa się z maksymalnie trzech literałów.
- Ścieżka Hamiltona (analogicznie cykl Hamiltona)
Czy w zadanym grafie istnieje ścieżka, która odwiedza każdy z wierzchołków grafu dokładnie raz?
- Problem kliki
Czy dany graf zawiera klikę rozmiaru k ?
- Zbiór niezależny
Czy istnieje zbiór mocy k złożony z wierzchołków zadanego grafu G taki, że żadne dwa wierzchołki nie są połączone krawędzią w G ?
- 3-kolorowanie grafu
Czy da się pokolorować wierzchołki grafu na 3 kolory w taki sposób, że żadna krawędź nie łączy dwóch wierzchołków o tym samym kolorze?

- akceptacja pustego słowa

Dana niedeterministyczna maszyna Turinga M oraz liczba n , czy M akceptuje słowo puste w co najwyżej n krokach?

Aby dowieść, że problem P jest NP-zupełny przeprowadzamy **redukcję wielomianową** z innego, znanego problemu NP-trudnego:

1. Uzasadnij, że P jest w klasie NP.
2. Zredukuj przy pomocy redukcji wielomianowej znany problem NP-trudny do problemu P .
3. P jest w klasie NP i jest NP-trudny $\implies P$ jest NP-zupełny.

Zauważmy, że problem może być NP-trudny (to znaczy co najmniej tak trudny jak inne problemy NP-trudne) nie będąc NP-zupełnym. Jest tak na przykład z problemem stopu.

Przykład 1.

Założymy, że wiemy, że problem 3-kolorowania grafu (3-KOL) jest NP-trudny. Pokażemy, że problem 3-CNF-SAT jest NP zupełny.

1. 3-CNF-SAT jest w klasie NP, ponieważ możemy niedeterministycznie zgadnąć wartościowanie zmiennych, a następnie w czasie wielomianowym sprawdzić, czy formuła logiczna jest spełniona.
2. Niech $G = (V, E)$ będzie wejściem dla problemu 3-KOL. Dla każdego $v \in V$ tworzymy zmienną x_v^i , $i \in \{1, 2, 3\}$ oznaczającą, że wierzchołek v ma kolor i . Oczywiście każdy wierzchołek musi mieć przyporządkowany dokładnie jeden kolor, więc tworzymy klauzule:

$$\forall_{v \in V} (x_v^1 \vee x_v^2 \vee x_v^3) \wedge (\neg x_v^1 \vee \neg x_v^2) \wedge (\neg x_v^1 \vee \neg x_v^3) \wedge (\neg x_v^2 \vee \neg x_v^3)$$

Dodatkowo dla każdej krawędzi chcemy, by kolory incydentnych do niej wierzchołków nie były równe:

$$\forall_{e=(v,u) \in E} \forall_{i \in \{1, 2, 3\}} (\neg x_v^i \vee \neg x_u^i)$$

W ten sposób otrzymujemy formułę φ w postaci 3-CNF. Gdybyśmy potrafili rozwiązać problem 3-CNF-SAT dla φ , to potrafilibyśmy rozwiązać problem 3-KOL dla $G = (V, E)$ przepisując wartościowanie zmiennych na odpowiednie kolory. Taka redukcja jest **wielomianowa** - dla każdego wierzchołka i krawędzi oryginalnego grafu wykonujemy wielomianową (stałą) liczbę operacji.

3. 3-CNF-SAT jest w klasie NP oraz jest NP-trudny, a zatem jest NP-zupełny.

Zestaw zadań

7.17. Język $L = \{a^n b^n \mid n \in \mathbb{N}\}$ jest

- A. regularny
 B. bezkontekstowy
 C. obliczalny w czasie wielomianowym

7.18. Zbiór dziesiętnych zapisów potęg dwójką jest

- A. językiem bezkontekstowym
 B. językiem należącym do klasy P
 C. językiem należącym do klasy NP

7.19. Zakładamy, że język regularny jest dany jako automat niedeterministyczny, a język bezkontekstowy jako gramatyka bezkontekstowa. Problemem rozstrzygalnym jest stwierdzenie

- A. czy dany język bezkontekstowy jest zawarty w danym języku bezkontekstowym

- B. czy dany język bezkontekstowy jest zawarty w danym języku regularnym
 C. czy dany język regularny jest zawarty w danym języku bezkontekstowym

7.20. Dla dowolnego języka $L \subseteq A^*$ nad alfabetem A oznaczmy przez L^R zbiór „lustrzanych odbić” słów z L . Formalnie:

$$\varepsilon^R = \varepsilon, \quad (wa)^R = a(w^R) \text{ dla } a \in A, w \in A^*, \quad L^R = \{w^R : w \in L\}.$$

Niech $L' = LL^R$ oznacza konkatenację języków L i L^R . Wynika z tego, że

- A. jeżeli L jest regularny, to L' też
 B. jeżeli L jest bezkontekstowy, to L' też
 C. jeżeli L jest rozstrzygalny, to L' też

7.21. Problemem rozstrzygalnym jest stwierdzenie

- A. czy gramatyka bezkontekstowa generuje język regularny
 B. czy możemy rozpoznać słowo puste na maszynie Turinga \mathcal{M} w mniej niż n kroków
 C. czy deterministyczna maszyna Turinga w $f(n)$ krokach rozpozna wszystkie słowa długości n dla każdego n

7.22. Dany jest alfabet $A = \{a, b\}$ oraz

- język bezkontekstowy K nad A dany jako gramatyka,
- język regularny R nad A dany jako niedeterministyczny automat.

Problemem rozstrzygalnym jest

- A. określenie niepustości podzbioru R takich słów, dla których liczba liter a jest równa liczbie liter b
 B. sprawdzenie, że $K \subset R$
 C. sprawdzenie, że $R \subset K$

7.23. Problem stopu dla maszyn Turinga jest

- A. w klasie NP
 B. częściowo rozstrzygalny
 C. rozstrzygalny

7.5.

Rozwiązańia

Rozwiązańia

7.1. Rozważmy wyrażenie regularne $K = A^*bbA^* + A^*aAA + A^*aA$ nad alfabetem $\{a, b\}$ (zapis A w wyrażeniu to skrót na $(a+b)$). Niech L to język tego wyrażenia. Wtedy

- TAK** A. automat minimalny dla L ma co najwyżej 4 stanów
NIE B. automat minimalny dla L ma przynajmniej 6 stanów
TAK C. istnieje automat niedeterministyczny dla L o 5 stanach

Znajdziemy klasy abstrakcji przy relacji Myhill-Nerodego.

Uwaga. Dalej w rozwiązyaniu będziemy skrótnie mówić o słowie w , na przykład, że słowo w „rozróżnia” dane klasy abstrakcji – chodzić będzie zawsze o w z definicji relacji Myhill-Nerodego.

Pierwszą oczywiście będzie $[\varepsilon]$. Jako następną sprawdzimy $[a]$. Są to różne klasy abstrakcji, bo dla $w = a$ mamy różne przyszłości – εw nie akceptuje, zaś aw tak. Sprawdźmy $[b]$. Jest to klasa różna od $[\varepsilon]$ – rozróżnia ją na przykład $w = b$. Klasy $[b]$ i $[a]$ rozróżniają $w = a$. Zatem jest to nowa klasa różna od poprzednich. Następnie sprawdźmy klasę $[aa]$. Klasę tę rozróżnia od każdej poprzedniej przyszłość $w = \varepsilon$.

Zanim kontynuujemy poszukiwania zauważmy istotną obserwację: język L zawiera wszystkie słowa długości co najmniej 3 (warto sprawdzić samemu dlaczego – wystarczy przyjrzeć się trzem ostatnim literom przykładowego słowa) i prawie wszystkie długości 2 – nie zawiera tylko słowa ba . Stąd wynika, że wszystkie klasy abstrakcji postaci $[u]$, gdzie $|u| \geq 3$, są sobie równe. Ponadto klasa $[aaa]$ jest równa klasie $[aa]$ (ponieważ $w = \varepsilon$ daje ten sam rezultat, a $w \neq \varepsilon$ w obu przypadkach prowadzi do słowa długości ≥ 3). To oznacza, że jedyne klasy jakie pozostały sprawdzić, to klasy o dwuliterowych reprezentatach.

Łatwo zauważyc (podobnie do argumentu, że $[aa] = [aaa]$), że $[ab]$ i $[bb]$ również są tą samą klasą co $[aa]$. Z kolei $[ba]$ jest tą samą klasą co $[a]$. Zatem mamy wszystkie klasy: $[\varepsilon]$, $[a]$, $[b]$ i $[aa]$. Stąd minimalny automat (deterministyczny) ma 4 stany. W punkcie C warto pamiętać, że automaty deterministyczne są również w szczególności automatami niedeterministycznymi, a znaleźliśmy 4-stanowy.

- 7.2. Dany jest język regularny L dany wyrażeniem regularnym $(a + b)^*a(a + b)(a + b)$. Wówczas

- NIE** A. każdy automat deterministyczny rozpoznający L ma co najmniej 9 stanów
NIE B. minimalny automat deterministyczny rozpoznający L ma 7 stanów
NIE C. każdy automat niedeterministyczny rozpoznający L ma co najmniej 6 stanów

Zauważmy, że to wyrażenie regularne, rozpoznaje dokładnie wszystkie słowa, których trzecia literka od końca to a .

Zatem, każdą klasę abstrakcję w relacji Myhill-Nerodego można utożsamić z ze stanem: na jakich z ostatnich 3 ostatnich pozycji znajdują się literki a ? Nietrudno zauważyc, że jeśli dla dwóch słów, odpowiedź na to pytanie jest różna, można znaleźć taki człon, że po doklejeniu go do słów, tylko jedno z nich jest akceptujące.

Mamy więc $2^3 = 8$ klas abstrakcji, czyli co najmniej 8 stanów w DFA, zatem odpowiedzi w A. oraz B. to NIE.

C. można skonstruować 5-stanowe NFA. Wyrażenie regularne ma 4 człony $((a + b)^*, a, (a + b), (a + b))$, i -ty ($i \in \{0, 1, 2, 3, 4\}$) stan oznacza, że już rozpoznaliśmy i członów wyrażenia.

- 7.3. Regularny jest język nad alfabetem $\{a, b\}$ złożony ze wszystkich słów, w których każde podsłowo

- TAK** A. długości 3 występuje parzystą liczbę razy
TAK B. długości większej niż 3 występuje parzystą liczbę razy
TAK C. występuje również jako prefiks

W A. tworzymy automat, który dla każdego z ośmiu możliwych słów długości 3 pamięta dla każdego z nich parzystość jego wystąpień jako podsłowa. Akceptacja ma miejsce w stanie odpowiadającym samym parzystym wystąpieniom. W B. korzystamy z następującej obserwacji: jednym z podłów słowa w jest całe słowo w . Oczywiście pojawia się ono jako podsłowo dokładnie raz, a więc nieparzystą liczbę razy. Język ten składa się więc ze słów o mniej niż 3 literach, więc jest regularny. W C. zauważamy, że do języka należą tylko słowa złożone z samych liter a lub samych liter b . Jest on oczywiście regularny ($L = a^* + b^*$).

- 7.4. Regularny jest język złożony ze wszystkich słów nad alfabetem $\{a, b, c\}$, które

- TAK** A. zaczynają się od $baca$
TAK B. nie zawierają $baca$ jako podsłowa

TAK C. zawierają *baca* jako podsłowo parzystą liczbę razy

A. Ten język jest rozpoznawany przez wyrażenie regularne $baca(a+b+c)^*$.

B. Stworzymy DFA, który rozpoznaje ten język. Niech stanem będą 4 ostatnie literki słowa. Przejścia konstruujemy w naturalny sposób. Usuwamy przejścia ze stanu *baca*, żeby nie akceptować go jako podsłowa.

C. Znowu będziemy tworzyć DFA. Niech stan będzie postaci:

$\langle 4 \text{ ostatnie literki słowa, parzystość liczby wystąpień } baca \rangle$

Ponownie, konstruujemy przejścia z definicji, i oznaczamy stany z parzystą liczbą wystąpień jako akceptujące.

7.5. L jest regularny. Czy regularny jest:

TAK A. $\{w : w \in L \wedge w \in L^R\}$

NIE B. $\{ww : w \in L\}$

TAK C. prefiksów słów z L parzystej długości

A. rozważmy przecięcie L z L^R . Języki regularne są zamknięte na przecięcia oraz odwracanie, więc odpowiedź jest twierdząca.

B. rozważmy $L = A^*$. Wtedy język z treścią to język zduplikowanych słów nad A - nazwijmy go L' . Jeśli L' byłby regularny, to jego przecięcie z językiem regularnym $L'' = a^*b^*a^*b^*$ byłoby równe $L' \cap L'' = \{a^n b^m a^n b^m\}$. Ten język oczywiście nie jest regularny (proste pomponowanie), a więc ze względu na zamkniętość języków regularnych na przecięcia L' nie mógł być regularny.

C. rozważmy automat A rozpoznający L i stworzymy na jego podstawie automat A' , który dla każdego stanu s automatu A utworzy dwie wersje s : s_0 oraz s_1 . Do s_0 będziemy mogli się dostać tylko w parzystej liczbie ruchów, a do s_1 tylko w nieparzystej (wymaga to zwykłego skopiowania A i dostawienia analogicznych krawędzi łączących stany parzyste z nieparzystymi). Akceptujemy w parzystych wersjach stanów akceptujących A .

7.6. Językiem regularnym jest

NIE A. język $L \subseteq \{1\}^*$ składający się z unarnych reprezentacji liczb pierwszych

TAK B. język $L \subseteq \{0, 1\}^*$ składający się ze słów, w których liczba wystąpień cyfry 0 przystaje modulo 2 do liczby wystąpień cyfry 1

TAK C. język $L \subseteq \{0, 1\}^*$ składający się z binarnych reprezentacji liczb podzielnych przez 7

A. Ze względu na nieskończoność i nieregularność liczb pierwszych automat generujący ten język musiałby być nieskończony.

B. Automat generujący ten język składa się z 4 stanów poetykietowanych ($\#_0 \bmod 2, \#_1 \bmod 2$). Łatwo zauważyc, że przejścia są postaci $(x, y) \xrightarrow{0} (1-x, y)$ i $(x, y) \xrightarrow{1} (x, 1-y)$, stan początkowy to $(0, 0)$, a stany akceptujące to (x, x) .

C. Można zauważyć, że kolejne potęgi 2 przy dzieleniu przez 7 dają kolejno reszty 1, 2, 4. A zatem automat generujący ten język składa się z 21 stanów poetykietowanych $(x, y) : x \in \{0, 1, 2, 3, 4, 5, 6\}, y \in \{1, 2, 4\}$. Łatwo zauważyc, że przejścia są postaci $(x, y) \xrightarrow{z} ((x + zy) \bmod 7, y^2 \bmod 7)$, stan początkowy to $(0, 1)$, a stany akceptujące to $(0, x)$.

7.7. Założymy, że L to język bezkontekstowy, zaś K to język regularny, oba nad alfabetem A . Przez X^R oznaczamy język powstały z X przez odwrócenie wszystkich słów z X . Językiem bezkontekstowym jest język

TAK A. $L^R \cap K$

NIE B. $(A^* - L) \cup K^*$

TAK C. $L^* \cup K^R$

A) L^R jest bezkontekstowy (odwrócenie automatu albo gramatyki), wiemy że przecięcie języka bezkontekstowego z regularnym daje bezkontekstowy.

B) Języki bezkontekstowe nie są zamknięte na dopełnienie.

C) Języki bezkontekstowe są zamknięte na gwiazdkę Kleene'go, a języki regularne na odwrócenia, więc ich suma jest językiem bezkontekstowym.

7.8. Dane są język bezkontekstowy L i język regularny K . Prawdą jest, że

NIE A. $K \setminus L$ jest bezkontekstowy

NIE B. $K \cap L$ jest regularny

TAK C. $L \setminus K$ jest bezkontekstowy

A) Założmy, że K to język zawierające wszystkie słowa nad alfabetem $\{a, b\}$. Wówczas $K - L$ jest dopełnieniem, a przecież języki bezkontekstowe nie są zamknięte na dopełnienie.

B) Takie przecięcie jest bezkontekstowe ale nie regularne.

C) $L - K = L \cap (A^* - K)$. Dopełnienie języka regularnego jest językiem regularnym więc w przecięciu z językiem bezkontekstowym daje język bezkontekstowy.

7.9. Niech L będzie nieskończonym językiem bezkontekstowym nad skończonym alfabetem. Wówczas

NIE A. L jest regularny

NIE B. L zawiera nieskończony język regularny

TAK C. L zawiera słowo postaci $uv^{2020}xy^{2020}z$, gdzie $vy \neq \epsilon$

A. Każdy język regularny jest językiem bezkontekstowym, ale nie na odwrotnie. Np. $a^n b^n$

B. Żaden nieskończony podzbiór języka bezkontekstowego $a^n b^n$ nie będzie językiem regularnym

C. Jeśli L jest bezkontekstowy, to zachodzi dla niego lemat o pompowaniu dla j. bezkontekstowych. Biorąc dowolne słowo postaci $uvxyz \in L$, wiemy że dla $uv \neq \epsilon$ zachodzi $uv^k xy^k z \in L$, a w szczególności $uv^{2020}xy^{2020}z \in L$.

7.10. Niech L, K to języki bezkontekstowe. Bezkontekstowy wtedy też jest język

TAK A. $L \cup K$

NIE B. $L \cap K$

TAK C. $L \cap a^*b^*$

A. Języki bezkontekstowe są zamknięte na sumę.

B. Języki bezkontekstowe nie są zamknięte na przecięcia.

C. Języki bezkontekstowe są zamknięte na przecięcia językiem regularnym.

7.11. Dla języka L , niech $\Psi(L)$ oznacza zbiór długości słów z języka L . Istnieje taki język bezkontekstowy L , dla którego $\Psi(L)$ jest zbiorem

TAK A. liczb naturalnych podzielnych przez 11

NIE B. kwadratów liczb naturalnych

NIE C. sześcielanów liczb naturalnych

A. Tak, wystarczy wziąć język $(a^{11})^*$. To język regularny, a języki regularne są w szczególności bezkontekstowe.

B. Taki język łamałby lemat o pompowaniu. Przypuśćmy, że jest taki język i istnieje dla niego stała N z lematu o pompowaniu. Wtedy weźmy dowolne słowo w o długości $\geq N$. Zauważmy, że pompując to słowo, jego długość rośnie jak ciąg arytmetyczny. Ale odległości pomiędzy kwadratami liczb naturalnych rosną, im większe są te kwadraty, w związku z tym nie jest możliwe, aby ww. pompowanie generowało słowa, których długości zawsze są kwadratami liczb naturalnych.

C. Analogicznie do B.

7.12. L_1 i L_2 są językami bezkontekstowymi nad alfabetem Σ . Wynika z tego, że bezkontekstowy jest język

NIE A. $L_1 \cap L_2$

NIE B. $\Sigma^* - L_1$

TAK C. $L_1 \cup L_2$

A. Języki bezkontekstowe nie są zamknięte na przecięcia.

B. Języki bezkontekstowe nie są zamknięte na dopełnienia.

C. Języki bezkontekstowe są zamknięte na sumę.

7.13. L jest językiem bezkontekstowym. Wynika z tego, że bezkontekstowy jest język

TAK A. $\{v : \text{istnieje słowo z } L, \text{ które można otrzymać przez usunięcie parzystej liczby (niekoniecznie kolejnych) liter z } v\}$

NIE B. $\{ww : w \in L\}$

TAK C. $\{a_1 \cdots a_n : a_n \cdots a_1 \in L\}$ (tzn. słowa z L pisane do tyłu)

A **B** Nie, byłoby tak dla języka $\{ww^R : w \in L\}$, ale tutaj nie możemy się nagle dowiedzieć, że jesteśmy na początku słowa w , bo informacja o tym jest na dole stosu. **C** języki bezkontekstowe są zamknięte na odwrócenia.

7.14. Następujący język jest bezkontekstowy:

NIE A. $\{a^n b^m c^k d^l : n = k \text{ i } m = l\}$

TAK B. $\{a^n b^m c^k d^l : n = l \text{ i } m = k\}$

TAK C. $\{a^n b^m c^k d^l : n = m \text{ i } k = l\}$

A. mamy język $a^n b^k c^n d^k$. Weźmy N z lematu o pompowaniu i słowo $a^{2N} b^{3N} c^{2N} d^{3N}$. Niech $left$, $right$ będą wewnątrz jednej literki. Wtedy psuje się symetria między a i c lub b i d . Jeśli $left$ jest w a , $right$ zaś w b , to także psuje się symetria. Jeśli $left$ lub $right$ są same w sobie na skraju literek, psuje się całkiem układ słowa. Stąd nie jest to język bezkontekstowy. **B.** język $a^n b^k c^k d^n$ wyraża gramatyka $S \rightarrow aRd$, $R \rightarrow aRd \mid L$, $L \rightarrow bLc \mid \epsilon$. **C.** język $a^n b^n c^k d^k$ wyraża gramatyka $S \rightarrow LR$, $L \rightarrow aLb \mid \epsilon$, $R \rightarrow cRb \mid \epsilon$.

7.15. Jeśli L_1 jest językiem bezkontekstowym, a L_2 jest językiem regularnym, to bezkontekstowy jest język

TAK A. $L_1 \cap L_2$

TAK B. $\{vw : v \in L_1, w \in L_2\}$

TAK C. $\{v^R : v \in L_1\}$, gdzie v^R oznacza odwrócenie słowa v

A. Języki bezkontekstowe są zamknięte na przecięcia z językami regularnymi.

B. Każdy język regularny jest bezkontekstowy, zatem L_2 bezkontekstowy. Języki bezkontekstowe są zamknięte na konkatenację.

C. Języki bezkontekstowe są zamknięte na odwrócenie.

7.16. Zbiór dziesiętnych zapisów liczb podzielnych przez 7 jest

TAK A. językiem regularnym

TAK B. językiem bezkontekstowym

TAK C. językiem kontekstowym

Pokażemy, że jest to język regularny poprzez opis automatu rozpoznającego ten język. Automat ten będzie miał 8 stanów: 7 stanów q_i dla $i \in \{0, 1, \dots, 6\}$ oznaczających resztę z dzielenia dotychczas wpisanej liczby (stan q_0 jest stanem akceptującym), oraz ósmy stan q_I , który będzie stanem początkowym – jest on potrzebny, bo zanim wczytamy pierwszą literę, wczytaliśmy tylko puste słowo, a ono nie reprezentuje żadnej liczby. Przejścia w takim automacie są następujące:

- $q_I \xrightarrow{n} q_{n \bmod 7}$ dla $n \in \{0, 1, \dots, 9\}$,
- $q_i \xrightarrow{n} q_j$ wtw. $j \equiv 10i + n \bmod 7$.

Przejścia z pierwszego punktu to po prostu zapamiętanie reszty z dzielenia przez 7, gdy wczytujemy pierwszą cyfrę. Przejścia z drugiego punktu to update reszty z dzielenia po dopisaniu kolejnej cyfry – zauważmy, że dopisanie cyfry k do liczby odpowiada pomnożeniu tej liczby przez 10 i dodaniu k .

Języki regularne są również bezkontekstowe i kontekstowe.

Uwaga: języki kontekstowe nie są już w programie – ale dla ciekawskich, języki kontekstowe to języki rozpoznawane przez gramatyki, które po lewych stronach reguł mogą mieć więcej niż jeden symbol (stąd są „kontekstowe”, bo liczy się kontekst).

7.17. Język $L = \{a^n b^n \mid n \in \mathbb{N}\}$ jest

- NIE** A. regularny
TAK B. bezkontekstowy
TAK C. obliczalny w czasie wielomianowym

A. klasyczny przykład języka nieregularnego (lemat o pompowaniu). B. jest bezkontekstowy, generuje go gramatyka $S \rightarrow aSb \mid \epsilon$. C. możemy go obliczyć w czasie liniowym.

7.18. Zbiór dziesiętnych zapisów potęg dwójki jest

- NIE** A. językiem bezkontekstowym
TAK B. językiem należącym do klasy P
TAK C. językiem należącym do klasy NP

A.

B. Sprawdzenie, czy liczba jest potągią dwójki zajmuje czas wielomianowy, więc jest w klasie P.
C. Ponieważ $P \subseteq NP$, a w B. jest TAK, to tutaj też musi być.

7.19. Zakładamy, że język regularny jest dany jako automat niedeterministyczny, a język bezkontekstowy jako gramatyka bezkontekstowa. Problemem rozstrzygalnym jest stwierdzenie

- NIE** A. czy dany język bezkontekstowy jest zawarty w danym języku bezkontekstowym
TAK B. czy dany język bezkontekstowy jest zawarty w danym języku regularnym
NIE C. czy dany język regularny jest zawarty w danym języku bezkontekstowym

A. Rozważmy języki bezkontekstowe $L_1 = A^*$ oraz L_2 dowolne. Wtedy zadanie sprowadza się do pytania, czy gramatyka generuje wszystkie możliwe słowa, co jest problemem nierostrzygalnym.

B. L_1 - język bezkontekstowy, L_2 - język regularny. $L_1 \subseteq L_2 \Leftrightarrow L_1 \cap (A^* - L_2) = \emptyset$. Umiemy znajdować dopełnienia j. regularnych oraz przecinać gramatyki z j. regularnymi. Do tego rozstrzygalne jest stwierdzenie, czy gramatyka rozpoznaje słowo puste, więc odpowiedź to TAK.

C. Analogicznie jak w A. niech $L_1 = A^*$ oraz L_2 - dowolna gramatyka. Pytanie o zawieranie to teraz pytanie o pełność gramatyki.

7.20. Dla dowolnego języka $L \subseteq A^*$ nad alfabetem A oznaczmy przez L^R zbiór „lustrzanych odbić” słów z L . Formalnie:

$$\varepsilon^R = \varepsilon, \quad (wa)^R = a(w^R) \text{ dla } a \in A, w \in A^*, \quad L^R = \{w^R : w \in L\}.$$

Niech $L' = LL^R$ oznacza konkatenację języków L i L^R . Wynika z tego, że

- TAK** A. jeżeli L jest regularny, to L' też
TAK B. jeżeli L jest bezkontekstowy, to L' też
TAK C. jeżeli L jest rozstrzygalny, to L' też

- A.** Języki regularne są zamknięte na odwrócenie i konkatenację.
B. Języki bezkonstekstowe są zamknięte na odwrócenie i konkatenację.
C. Możemy skonstruować maszynę Turinga, która obliczy L' . Dla każdego prefiksu słowa, maszyna sprawdzi, czy należy on do L , a jeśli tak, to sprawdzi, czy reszta słowa należy do L^R .

7.21. Problemem rozstrzygalnym jest stwierdzenie

- NIE** **A.** czy gramatyka bezkontekstowa generuje język regularny
TAK **B.** czy możemy rozpoznać słowo puste na maszynie Turinga \mathcal{M} w mniej niż n kroków
NIE **C.** czy deterministyczna maszyna Turinga w $f(n)$ krokach rozpozna wszystkie słowa długości n dla każdego n

- A.** Jeśli by tak było, moglibyśmy się spytać, czy gramatyka generuje język $\{a, b\}^*$, czyli odpowiedzieć na pytanie o pełność gramatyki, co nie jest rozstrzygalne.
B. Odpalamy maszynę \mathcal{M} i po $n - 1$ krokach patrzymy, czy rozpoznała słowo puste.
C. Dla danego z góry n jest to rozstrzygalne, ale nie będzie rozstrzygalne dla każdego n .

7.22. Dany jest alfabet $A = \{a, b\}$ oraz

- język bezkontekstowy K nad A dany jako gramatyka,
- język regularny R nad A dany jako niedeterministyczny automat.

Problemem rozstrzygalnym jest

- TAK** **A.** określenie niepustości podzbioru R takich słów, dla których liczba liter a jest równa liczbie liter b
TAK **B.** sprawdzenie, że $K \subset R$
NIE **C.** sprawdzenie, że $R \subset K$

A jest to problem rozstrzygalny. Weźmy język bezkontekstowy taki, w którym liczba wystąpień a jest równa liczbie wystąpień b . Przecięcie regularnego języka R z tym bezkontekstowym jest bezkontekstowe, a sprawdzenie pustoci gramatyki jest rozstrzygalne. **B** jest rozstrzygalne. Bierzemy dopełnienie języka R i przecinamy je z K . Przecięcie ich jest bezkontekstowe. Jeśli przecięcie jest niepuste, to K nie zawiera się w R . **C** jeśli R jest językiem A^* , to jest to efektywnie problem sprawdzenia pełności gramatyki języka bezkontekstowego, co nie jest rozstrzygalne.

7.23. Problem stopu dla maszyn Turinga jest

- NIE** **A.** w klasie NP
TAK **B.** częściowo rozstrzygalny
NIE **C.** rozstrzygalny

A problem stopu nie jest obliczalny w żadnym czasie. **B** problem stopu jest klasycznym przykładem problemu częściowo rozstrzygalnego. **C** nie jest rozstrzygalny, gdyż jego dopełnienie nie jest częściowo rozstrzygalne.

8

Bazy danych

Materiały teoretyczne z baz danych zostały opracowane na podstawie skryptu Zbigniewa Jurkiewicza, materiałów Filipa Murlaka oraz notatek z wykładu Jana Kwiatkowskiego i Błażeja Wilkoławskego.

Podstawa programowa

1. **Relacyjny model danych.**
2. Podstawowe konstrukcje języka **SQL** i sposoby ich realizacji.
3. **Rodzaje metadanych** i ich rola.
4. **Redundancja** a postacie normalne.
5. Przejście od **modelu pojęciowego** do **modelu logicznego**.
6. **Fizyczna reprezentacja danych.**

8.1.

Relacyjny model danych

Relacyjny model danych pomaga przedstawić dane w sposób sformalizowany z użyciem języka matematyki. Opiera się on na **relacjach**, czyli tabelach z danymi, na przykład relacja **Narty** może wyglądać następująco:

model	producent
Cool Minx	Atomic
Jewel Cristal	Salomon

Strukturę danych i powiązania między nimi opisuje się w schemacie bazy danych. **Schemat relacji** podaje nazwę relacji i listę jej atrybutów, można też dodatkowo określić typy atrybutów, na przykład dla powyższej tabeli narty:

- **Narty**(model, producent)
- **Narty**(model: string, producent: string)

Schemat bazy danych obejmuje schematy wszystkich relacji zawartych w tej bazie. W ten sposób bazę danych w modelu relacyjnym traktujemy po prostu jako kolekcję relacji.

Przedstawianie danych w postaci relacji ma wiele zalet, między innymi

- prosty model, często intuicyjnie pasujący do danych
- znane własności matematyczne (*algebra relacji*)
- podstawa języka SQL

Każda relacja powinna mieć swój **klucz**, czyli atrybut lub atrybuty takie, że w żadnych dwóch wierszach tabeli nie mogą one mieć takiej samej wartości. Klucze to pewny przypadek ograniczenia (wiązów) nakładanego na bazę danych, aby zagwarantować pewne warunki poprawności.

Przykład 1.

Oto przykład pewnego schematu bazy danych. Podkreślone atrybuty są kluczami.

- **Narty**(model, producent)
- **Wypożyczalnie**(nazwa, adres, telefon)
- **Narciarze**(PESEL, imię, nazwisko, adres, telefon)
- **Preferencje**(narciarz, narty)
- **Wypożyczenia**(wypożyczalnia, narty, cena)
- **Rejestracje**(narciarz, wypożyczalnia)

To było na egzaminie

W tabeli R są tylko kolumny A i B oraz nie występują wartości NULL. Ponadto, A jest kluczem głównym, liczba różnych wartości w kolumnie A to k , a w kolumnie B to l . Oznacza to, że zachodzi

- A. $k \leq l$
- B. $k = l$
- C. $k \geq l$

Kolumna A , jako kolumna kluczowa, nie może zawierać powtórzeń wartości, natomiast kolumna B – może. Przypadek $k < l$ jest więc niemożliwy (każda wartość w kolumnie B jest powiązana z dokładnie jedną, unikalną wartością w kolumnie A). Przypadek $k = l$ jest dość oczywisty – tworzymy tabelę, w której każdemu kluczowi z kolumny A odpowiada unikalna wartość w kolumnie B . Gdyby wartości w kolumnie B nie były jednak unikalne, czyli istniałyby dwa klucze k_1, k_2 o przypisanej wspólnej wartości l_1 , to zachodziłoby $k > l$.

W związku z tym tylko odpowiedź C. jest prawdziwa.

Algebra relacji

Algebra relacji to model teoretyczny do opisywania semantyki relacyjnych baz danych. Jej operacje zostały dobrane tak, aby odpowiadały typowym operacjom występującym w zapytaniach podczas wyszukiwania informacji z tabel w bazie danych.

Relacje w algebrze relacji reprezentujemy ich nazwami. Z nazwą każdej nowej relacji związany jest jej schemat – ciąg nazw atrybutów (odpowiadających kolumnom modelowanej tabeli), np.

- $R(A, B, C)$
- **Student**(numer indeksu, imię, nazwisko)

Nazwy atrybutów w schemacie relacji muszą być różne. Elementy relacji często nazywa się i reprezentuje jako **krotki**, które odpowiadają wierszom tabel z bazy danych. Ponieważ relacje (w algebraicznym sensie) są zbiorami, wszystkie ich krotki są różne – powtórzenia są eliminowane i scalane w jedno.

Zestaw operacji obejmuje typowe operacje teoriomnogościowe: **sumę**, **iloczyn** i **różnicę** zbiorów. Wymaga się wtedy, aby oba argumenty miały ten sam schemat atrybutów.

$$R \cup S = \{(a_1, \dots, a_n) \mid (a_1, \dots, a_n) \in R \vee (a_1, \dots, a_n) \in S\}$$

$$R \cap S = \{(a_1, \dots, a_n) \mid (a_1, \dots, a_n) \in R \wedge (a_1, \dots, a_n) \in S\}$$

$$R \setminus S = \{(a_1, \dots, a_n) \mid (a_1, \dots, a_n) \in R \wedge (a_1, \dots, a_n) \notin S\}$$

Iloczyn kartezjański $R \times S$ także jest zdefiniowany klasycznie. Ponieważ jednak argumenty mogą mieć atrybuty o takich samych nazwach, nazwy kolumn w schemacie wynikowym trzeba czasem poprzedzać nazwami relacji, z których pochodzą, na przykład dla relacji $R(A, B)$ i $S(B, C)$ schematem ich iloczynu kartezjańskiego będzie $R \times S(A, R.B, S.B, C)$.

$$R \times S = \{(a_1, \dots, a_n, b_1, \dots, b_m) \mid (a_1, \dots, a_n) \in R, (b_1, \dots, b_m) \in S\}$$

Przykład 2.

Oto przykład dwóch prostych relacji i ich iloczynu kartezjańskiego.

R_1	R_2	$R_1 \times R_2$			
A	B	A	$R_1.B$	$R_2.B$	C
1 3	2 4	10 30	20 40	2 30 10 30	20 40 20 40

Oprócz operacji teoriomnogościowych w algebrze relacji określono także kilka innych, specyficznych dla niej. Pierwsza z nich to **selekcja** (wybór), oznaczana jako $\sigma_{\text{warunek}}(R)$. Zgodnie z nazwą, wybiera ona z relacji tylko te krotki, dla których jest spełniony podany warunek.

$$\begin{aligned}\sigma_{i=j}(R) &= \{(a_1, \dots, a_n) \mid (a_1, \dots, a_n) \in R, a_i = a_j\} \\ \sigma_{i=C}(R) &= \{(a_1, \dots, a_n) \mid (a_1, \dots, a_n) \in R, a_i = C\}\end{aligned}$$

Przykład 3.

Oto przykład tabeli **Zwierzaki** i operacji selekcji na niej.

Zwierzaki	gatunek	imię	$\sigma_{\text{gatunek}='Papuga'}(\text{Zwierzaki})$	gatunek	imię
	Papuga	Kropka		Papuga	Kropka
	Papuga	Lulu		Papuga	Lulu
	Papuga	Hipek		Papuga	Hipek
	Lis	Fufu			
	Krokodyl	Czako			

Podobna do selekcji jest operacja **rzutowania** (projekcji), oznaczana przez $\pi_{kolumna_1, \dots, kolumna_n}(R)$: z relacji wybieramy tylko podane kolumny. Zwróćmy uwagę, że mogłyby to doprowadzić do utworzenia relacji, w której niektóre wiersze są takie same. Ponieważ jednak relacje są zbiorami, więc takie duplikaty są automatycznie eliminowane.

$$\pi_{i_1, \dots, i_k}(R) = \{(a_{i_1}, \dots, a_{i_k}) \mid (a_1, \dots, a_n) \in R\}$$

Przykład 4.

Oto przykład tabeli **Zwierzaki** i operacji rzutowania na niej.

Zwierzaki	gatunek	imię	$\pi_{\text{gatunek}}(\text{Zwierzaki})$	gatunek
	Papuga	Kropka		Papuga
	Papuga	Lulu		Lis
	Papuga	Hipek		Krokodyl
	Lis	Fufu		
	Krokodyl	Czako		

Złączenie naturalne $R \bowtie S$ jest podobne do iloczynu kartezjańskiego, ale łączy ze sobą tylko niektóre pary wierszy:

- R i S muszą mieć przynajmniej jedną wspólną kolumnę o tej samej nazwie

- warunkiem złączenia jest równość dla wszystkich par atrybutów o tych samych nazwach
- w wyniku zostaje tylko jedna kolumna z pary kolumn o tych samych nazwach

Przy złączeniach wprowadza się pojęcie **porzuconej krotki** – jest to wiersz z jednej relacji, do którego nie pasuje żaden wiersz z drugiej relacji.

W wyniku złączenia naturalnego nie rozpatruje się porzuconych krotek z żadnej relacji. Wyróżniamy także **złączenia lewo- i prawostronne** \bowtie_L, \bowtie_R , w których brane pod uwagę są tylko porzucone krotki z, odpowiednio, pierwszego i drugiego argumentu. Jako wypełniacz brakujących wartości w dołączonych kolumnach używa się NULL-i.

Przykład 5.

Oto przykład tabel **Zwierzaki** i **Gatunki** oraz ich złączenia naturalnego.

	gatunek	imię		gatunek	kontynent
Zwierzaki =	Papuga	Kropka	Gatunki =	Papuga	Ameryka
	Papuga	Lulu		Lis	Europa
	Papuga	Hipek		Krokodyl	Afryka
	Lis	Fufu		Krowa	Europa
	Krokodyl	Czako			
	Świnia	Bubu			

	gatunek	imię	kontynent
Zwierzaki \bowtie Gatunki =	Papuga	Kropka	Ameryka
	Papuga	Lulu	Ameryka
	Papuga	Hipek	Ameryka
	Lis	Fufu	Europa
	Krokodyl	Czako	Afryka
	Świnia	Bubu	

Porzucona krotka z tabeli **Zwierzaki** to {Świnia, Bubu}, a z tabeli **Gatunki** – {Krowa, Europa}. Złączenie lewostronne wyglądałoby następująco:

	gatunek	imię	kontynent
Zwierzaki \bowtie_L Gatunki =	Papuga	Kropka	Ameryka
	Papuga	Lulu	Ameryka
	Papuga	Hipek	Ameryka
	Lis	Fufu	Europa
	Krokodyl	Czako	Afryka
	Świnia	Bubu	NULL

Zestaw zadań

- 8.1. Przypuśćmy, że w tabeli R o kolumnach A, B, C para kolumn A, B jest kluczem. Założymy też, że w tabeli R nie występują wartości NULL, w kolumnie A pojawia się dokładnie k różnych wartości, w kolumnie B dokładnie l różnych wartości, a w kolumnie C dokładnie m różnych wartości. Wynika z tego, że

- A. $\min(k, l) \leq m$
 B. $k + l \geq m$
 C. $k \cdot l \geq m$

- 8.2. Dane są relacje R i Q , każda zawierająca dokładnie n krotek. Wynika z tego, że relacja $R \bowtie Q$ (złączenie naturalne R i Q) ma

- A. co najmniej n krotek
 B. co najwyżej n^2 krotek
 C. dokładnie $2n$ krotek

8.2.

Podstawowe konstrukcje języka SQL

Podstawowym językiem komunikacji z relacyjnymi bazami danych jest **SQL**. Zawiera on zarówno konstrukcje do definiowania schematu danych, jak i do operowania na zawartości bazy. Na tych drugich skupimy się w tym rozdziale.

Zapytania i filtrowanie

Do zadawania zapytań służy polecenie **SELECT**. Najprostsza jego postać ma format

```
SELECT kolumna1, kolumna2, ...
FROM tabela1, tabela2, ...
WHERE warunek1, warunek2, ...;
```

Realizacja takiego zapytania składa się z następujących etapów:

1. weź tabele podane w części **FROM**,
2. wybierz wiersze, używając warunku z frazy **WHERE** (operacja selekcji),
3. wybierz tylko kolumny wskazane frazą **SELECT** (operacja rzutowania).

SELECT musi być pierwszą frazą zapytania. Oprócz nazw kolumn i wyrażeń nad nimi można w niej używać dodatkowo specjalnego symbolu gwiazdki (*) oznaczającego „wszystkie atrybuty relacji”.

Jeśli we frazie **FROM** podamy więcej niż jedną tabelę, zostanie utworzony ich iloczyn kartezjański (zgodnie z definicją z algebra relacji), a następnie zapytanie będzie przetwarzane na tak powstały iloczynie.

Fraza **WHERE** jest opcjonalna – jej pominięcie oznacza, że wynik będzie zawierał odpowiedniki wszystkich wierszy źródłowych tabeli. W warunkach umieszczanych po **WHERE** można umieszczać dowolne wyrażenia logiczne, najważniejsze elementy ich budowy to:

- operatory arytmetyczne +, -, *, /
- operatory porównywania =, <>, <, >, <=, >=, można nimi również porównywać napisy i daty
- spójniki logiczne AND, OR, NOT
- wyrażenie „wartość **IN** zbiór” bada przynależność wartości do zbioru; zbiór może być podany jawnie przez wyliczenie elementów (e1, e2, ..., en) lub jako podzapytanie (**SELECT**)
- wyrażenie „**EXISTS** podzapytanie” sprawdza, czy wynik podzapytania zawiera przynajmniej jeden rekord
- wyrażenie „wartość **BETWEEN** a **AND** b” sprawdza przynależność wartości do przedziału domkniętego [a, b]
- wyrażenie „napis **LIKE** wzorzec” sprawdza dopasowanie napisu do wzorca; wzorzec jest napisem-szablonem, w którym % oznacza dowolny ciąg znaków, zaś _ dowolny pojedynczy znak

Przykład 1.

W bazie istnieją dwie tabele: **Dzieci**(imię, wiek) i **Zwierzaki**(imię, wiek, gatunek). Oto kilka przykładowych zapytań:

```
-- Wybieramy wyłącznie kolumnę imię z tabeli Dzieci.
SELECT imię FROM Dzieci;

-- Wypisujemy wszystkie możliwe krotki (imię dziecka, imię zwierzęcia).
SELECT Dzieci.imię, Zwierzaki.imię
```

```

FROM Dzieci, Zwierzaki;

-- Wybieramy zwierzęta starsze niż 4 lata.
SELECT *
FROM Zwierzaki
WHERE wiek > 4;

-- Listujemy imiona wszystkich lwów starszych niż 4 lata.
SELECT imię
FROM Zwierzaki
WHERE gatunek = 'lew' AND wiek > 4;

-- Listujemy imiona wspólne dla pewnego dziecka i pewnego zwierzaka.
SELECT imię
FROM Dzieci
WHERE imię IN (SELECT imię FROM Zwierzaki);

```

Przy tworzeniu zapytań przydatne mogą być **aliasy** tworzone za pomocą słowa kluczowego **AS**. Używamy ich do nadania kolumnie, tabeli lub (pod)zapytaniu własnej, tymczasowej nazwy.

W przypadku tworzenia aliasów tabel podanych po frazie **FROM** można pominąć **AS** i użyć skróconej składni, pisząc po prostu „**FROM** tabela1 alias1, tabela2 alias2...”.

Przykład 2.

W bazie danych istnieją tabele **Konsumenti**(id, imię, nazwisko) oraz **Zamówienia**(id, id_konsumenta, data_zamówienia). Oto kilka przykładów prawidłowego użycia aliasów w zapytaniach SQL:

```

-- Alias użyty jako nadanie nowej nazwy kolumnie.
SELECT imię + ' ' + nazwisko AS pełne_imię
FROM Konsumenti;

-- Aliasy nadające skróconą nazwę tabelom - tu można pominąć słowo AS.
SELECT z.id, z.data_zamówienia
FROM Konsumenti k, Zamówienia z
WHERE k.imię = 'Kamil' AND k.id = z.id_konsumenta;

```

Przykład 3.

W bazie danych istnieje tabela **Zwierzaki**(imię, gatunek), w której kluczem głównym jest imię. Wypiszemy wszystkie możliwe pary zwierzaków (ich imiona) tego samego gatunku.

Będziemy chcieli unikać identycznych par typu (x, x) oraz, dla ujednolicenia wyniku, w obrębie pary zachowamy porządek alfabetyczny, tzn. (x, y) , a nie (y, x) .

Aby złączyć tabelę z nią samą, musimy użyć aliasów – tak abyśmy mogli odnosić się do odpowiednich kolumn bez niejednoznaczności.

```

SELECT z1.imię, z2.imię
FROM Zwierzaki z1, Zwierzaki z2
WHERE z1.gatunek = z2.gatunek AND z1.imię < z2.imię;

```

To było na egzaminie

Tabele Emp i Dept mają, odpowiednio, E i D wierszy. Zapytanie

```
SELECT * FROM Emp e, Dept d WHERE e.deptno = d.deptno
```

- A. zawsze ma niepusty wynik, jeżeli $D \cdot E \neq 0$
- B. ma pesymistyczną złożoność czasową $O(D \cdot E)$
- C. może mieć wynik rozmiaru $D \cdot E$

- A. Jeżeli żadna z wartości w kolumnie e.deptno nie wystąpi w d.deptno, to zapytanie zwróci pusty wynik, więc odpowiedź to NIE.
- B. Jeżeli w kolumnach e.deptno i d.deptno występuje (łącznie) tylko jedna, ta sama wartość, to silnik bazodanowy przetwarzający zapytanie wypisze na wyjście wszystkie możliwe pary, których jest $D \cdot E$, więc odpowiedź to TAK.
- C. Przykład analogiczny do B.: jeśli wszystkie wartości w obu kolumnach deptno będą takie same, to w wyniku dostaniemy $D \cdot E$ krotek.

W bazach danych często przechowujemy niekompletną informację. Na takie okazje SQL posiada wyróżnioną wartość pustą – NULL.

Należy bardzo uważać na wartości NULL w warunkach. Logika dla warunków w SQL jest **trójwartościowa**: *true, false, unknown*. Jakiekolwiek normalne porównanie z wartością NULL daje wynik *unknown*, podobnie dla operacji arytmetycznych. Dlatego do sprawdzania wartości pustych należy używać specjalnych operatorów porównania **IS NULL** i **IS NOT NULL**.

Przydatne może być również wyrażenie „**COALESCE(v1, v2)**”. Jego wartością jest v_1 , o ile nie jest NULL-em, w przeciwnym razie v_2 .

■ SQL a algebra relacji

SQL **nie jest algebrą relacji**, dlatego **powtórzenia nie są automatycznie eliminowane z tabel**. Do usuwania powtórzeń z wyników zapytań służy modyfikator **DISTINCT** we frazie **SELECT**.

Przykład 4.

W bazie istnieje tabela **Gatunki**(gatunek, kontynent). Wypiszemy zakres kontynentów, z których pochodzą wszystkie zawarte w tabeli gatunki zwierząt:

```
SELECT DISTINCT kontynent  
FROM Gatunki;
```

Przy braku **DISTINCT** każdy kontynent zostałby wypisane wielokrotnie (tyle razy, ile razy występuje w tabeli **Gatunki**).

Operacje teoriomnogościowe UNION (\cup), INTERSECT (\cap) i EXCEPT (\setminus) automatycznie eliminują powtórzenia, o ile nie zastosowano modyfikatora **ALL**.

Przykład 5.

W bazie istnieje tabela **Zwierzaki**(imię, gatunek, wiek, waga). Rozważmy następujące zapytanie:

```
(SELECT DISTINCT gatunek  
FROM Zwierzaki  
WHERE waga > 100)  
UNION ALL  
(SELECT DISTINCT gatunek  
FROM Zwierzaki  
WHERE wiek > 10);
```

Wynikiem będą gatunki zwierząt, których przynajmniej jeden osobnik waży powyżej 100 kg lub ma więcej niż 10 lat. Ponieważ zastosowano modyfikator **ALL**, potencjalnie pojawią się duplikaty gatunków spełniających oba powyższe warunki jednocześnie.

Funkcje agregujące

Funkcje agregujące są przeznaczone do obliczania wartości parametrów statystycznych, takich jak średnia czy suma, dotyczących całej tabeli (lub wybranych grup wierszy).

Standardowe funkcje agregujące to AVG, COUNT, MAX, MIN i SUM. Z wyjątkiem wyrażenia „**COUNT(*)**” wartości puste są pomijane.

Funkcji **COUNT** warto przyjrzeć się dokładniej:

- **COUNT(*)** zlicza wiersze uzyskane w wyniku zapytania – także takie, które zawierają wartości puste (NULL) w niektórych kolumnach.
- **COUNT(nazwa_kolumny)** zlicza wartości z podanej kolumny. Powtórzenia są wliczane, ale pomijane są wiersze zawierające w tej kolumnie wartość pustą.
- **COUNT(DISTINCT nazwa_kolumny)** zlicza liczbę różnych wartości w podanej kolumnie.

Przykład 6.

W bazie istnieje tabela **Zwierzaki**(imię, gatunek, wiek, waga). Obliczymy kilka statystycznych parametrów na temat niedźwiedzi:

```
-- Średnia waga niedźwiedzi.  
SELECT AVG(waga)  
FROM Zwierzaki  
WHERE gatunek = 'niedźwiedź';  
  
-- Liczba wszystkich niedźwiedzi.  
SELECT COUNT(*)  
FROM Zwierzaki  
WHERE gatunek = 'niedźwiedź';  
  
-- Liczba wszystkich nazwanych niedźwiedzi.  
SELECT COUNT(imię)  
FROM Zwierzaki  
WHERE gatunek = 'niedźwiedź';  
  
-- Liczba różnych imion nadanych niedźwiedziom.  
SELECT COUNT(DISTINCT imię)  
FROM Zwierzaki  
WHERE gatunek = 'niedźwiedź';
```

Grupowanie, czyli dzielenie wierszy na grupy frazą **GROUP BY**, ułatwia równoczesne obliczanie parametrów statystycznych dla wybranych podzbiorów wierszy. Moglibyśmy np. chcieć obliczyć średnią wagę dla każdego gatunku zwierzęcia. Służłyby do tego zapytanie

```
SELECT gatunek, AVG(waga)
FROM Zwierzaki
GROUP BY gatunek;
```

Warunkiem frazy **WHERE** można ograniczyć grupowanie tylko do wybranych wierszy. Na przykład, dopisanie do powyższego zapytania „**WHERE wiek > 10**” obliczyłoby dla każdego gatunku średnią wagę zwierząt mających ponad 10 lat.

Czasem chcemy również filtrować wynik zapytania po warunkach dla całych grup, a nie tylko pojedynczych wierszy. Służy do tego słowo kluczowe **HAVING**. Na przykład zapytanie

```
SELECT gatunek, AVG(waga)
FROM Zwierzaki
GROUP BY gatunek
HAVING COUNT(*) > 2;
```

przy obliczaniu wyniku uwzględnia tylko gatunki posiadające przynajmniej 3 zwierzaki.

Przykład 7.

Znajdziemy najwyższą średnią wagę zwierząt dla pewnego gatunku. Uwzględnimy przy tym wyłącznie gatunki mające przynajmniej 5 zwierząt, a przy liczeniu średnich weźmiemy pod uwagę tylko osobniki o wieku powyżej 4 lat.

W tym celu napiszemy proste zapytanie zagnieżdżone: najpierw obliczymy średnie, a potem wybierzymy największą z nich.

```
SELECT MAX(srednia_waga)
FROM (SELECT gatunek, AVG(waga) as srednia_waga
      FROM Zwierzaki
      WHERE wiek > 4
      GROUP BY gatunek
      HAVING COUNT(*) > 5) Srednie;
```

Złączenia

Często dane w bazie są trzymane w wielu tabelach, a przy tworzeniu zapytań używamy **złączeń**. SQL ma zdefiniowane kilka standardowych operatorów złączeń do używania we frazie **FROM**:

- **CROSS JOIN** – standardowy iloczyn kartezjański (taki sam efekt, jak wypisanie kilku tabeli po przecinku)
- **NATURAL JOIN** – złączenie naturalne (równościowe po kolumnach o tych samych nazwach)
- **(INNER) JOIN, INTERSECT** – standardowe złączenie **bez porzuconych krotek**
- **LEFT/RIGHT (OUTER) JOIN** – złączenie uwzględniające porzucone krotki wyłącznie z lewego/prawego argumentu
- **FULL (OUTER) JOIN** – złączenie uwzględniające porzucone krotki z obu argumentów

Słowo kluczowe **ON** określa warunek, na podstawie którego łączone są tabele. Kiedy nie ma warunku **ON**, kolumny są dopasowywane po nazwach – jak w złączeniu naturalnym.

Przykład 8.

Oto przykład tabel **Sportowcy** i **Punkty**:

	nazwisko	imię
Sportowcy =	Abacka	Alicja
	Babacki	Błażej
	Cabacki	Czesław

	nazwisko	punkty
Punkty =	Babacki	1
	Cabacki	2
	Dabacki	3

Rozważmy następujące zapytanie łączące obie tabele:

```
SELECT s.imię, p.punkty  
FROM Sportowcy s JOIN Punkty p  
ON s.nazwisko = p.nazwisko;
```

przy czym słowo kluczowe **JOIN** będziemy kolejno zamieniać na różne rodzaje złączeń. W wyniku otrzymamy:

INNER JOIN		LEFT JOIN		RIGHT JOIN		FULL JOIN	
imię	punkty	imię	punkty	imię	punkty	imię	punkty
Błażej	1	Alicja	NULL	Błażej	1	Alicja	NULL
Czesław	2	Błażej	1	Czesław	2	Błażej	1

Modyfikacja wierszy w tabeli

Nowe wiersze do tabeli wstawiamy poleceniem **INSERT**:

```
INSERT INTO tabela  
VALUES (wartosc1, wartosc2, ...);
```

Zmian w już istniejących wierszach dokonujemy poleceniem **UPDATE**:

```
UPDATE tabela  
SET kolumna1 = wartosc1, kolumna2 = wartosc2, ...  
WHERE warunek;
```

Zmiana dotyczy wszystkich wierszy, dla których jest spełniony warunek. Jeśli warunek został pominięty, zmiana dotyczy wszystkich wierszy w tabeli.

Do usuwania wierszy służy polecenie **DELETE**:

```
DELETE FROM tabela  
WHERE warunek;
```

Podobnie jak poprzednio, usuwane są wszystkie wiersze, dla których spełniony jest podany warunek, a jeśli został on pominięty, usuwane są wszystkie wiersze w tabeli.

Zestaw zadań

8.3. Dana jest tabela Sprawdzian:

student	kolokwium	egzamin
A	45	NULL
B	NULL	90
C	100	80

Wynik zapytania w SQL

```
SELECT student  
FROM Sprawdzian  
WHERE (kolokwium > egzamin AND egzamin > 75) OR kolokwium < 50
```

będzie zawierał identyfikator studenta

- A. A
- B. B
- C. C

8.4. W tabelach R i S kluczem głównym jest kolumna A , która jest jednocześnie jedyną kolumną. Wszystkie krotki z R zawarte w S zwróci zapytanie

- A. `SELECT * FROM R WHERE EXISTS (SELECT * FROM S WHERE R.A = S.A)`
- B. `SELECT R.A FROM R LEFT JOIN S ON R.A = S.A`
- C. `(SELECT * FROM R) INTERSECT (SELECT * FROM S)`

8.5. Mamy dwie tablice A, B z kolumnami kolejno a, b . Istnieją takie ich zawartości, że zapytanie

```
SELECT * FROM A, B WHERE A.a = B.b
```

- A. zwróci pusty wynik
- B. zwróci dokładnie $|A| \cdot |B|$ krotek
- C. zapętli się

8.6. Mamy relację R z kolumnami a oraz b i wykonujemy następujące zapytanie SQL:

```
SELECT R1.a, R2.b FROM R AS R1, R AS R2
```

W wyniku

- A. znajdą się tylko krotki z R
- B. znajdą się wszystkie krotki z R
- C. nie znajdzie się żadna krotka z R

8.7. Zapytanie `SELECT R.A FROM R` zwraca r wierszy, zapytanie `SELECT S.A FROM S` zwraca s wierszy, a zapytanie `SELECT R.A FROM R WHERE R.A NOT IN (SELECT S.A FROM S)` zwraca k wierszy. Wynika stąd, że

- A. $k = r - s$
- B. $r - s \leq k \leq r$
- C. $0 \leq k \leq r$

8.8. Dana jest tabela $R(a, b, c, d)$ oraz szkielet zapytania

```
SELECT [...]  
FROM R  
GROUP BY a, b;
```

Poprawne wyrażenie SQL otrzymamy, wstawiając w miejsce [...]

- A. `MIN(c + d)`
- B. `a, b`

C. b, c

8.9. Niech R będzie tabelą o dwóch kolumnach X, Y oraz 0 wierszach. Rozważmy dwie instrukcje:

<pre>INSERT INTO R VALUES ('a', 'b');</pre>	-- Instrukcja A
<pre>DELETE FROM R WHERE X = 'a';</pre>	-- Instrukcja B

W związku z tym

- A. dwukrotne wykonanie instrukcji A daje taki sam wynik jak jej jednokrotne wykonanie
- B. jeśli kolumna X jest kluczem głównym, to dwukrotne wykonanie instrukcji A skutkuje błędem
- C. wykonanie instrukcji B skutkuje błędem

8.3. Zależności funkcyjne

Czasem tabela jest tak zaprojektowana, że po ustaleniu wartości pewnych jej atrybutów wartości jakichś innych atrybutów są jednoznacznie wyznaczone. Mamy wtedy do czynienia z **zależnością funkcyjną**.

Zapis $R : A_1A_2\dots A_k \rightarrow B_1B_2\dots B_n$ oznacza, że w tabeli R wartości w kolumnach A_1, \dots, A_k determinują wartości w kolumnach B_1, \dots, B_n . Inaczej, jeśli dwie krotki w tabeli R mają te same wartości w kolumnach A_1, \dots, A_k , to mają też te same wartości w kolumnach B_1, \dots, B_n .

Przykład 1.

Jeśli w ZOO każdy zwierzak nazywa się inaczej, to dla relacji **Zwierzaki**(imię, gatunek, waga, wiek) zachodzi na przykład zależność

$$\text{imię} \rightarrow \text{gatunek}$$

Zależność funkcyjną, której prawa strona zawiera kilka atrybutów

$$X \rightarrow A_1A_2 \dots A_n$$

można zastąpić zbiorem zależności o pojedynczych prawych stronach

$$X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$$

Operacja ta jest oczywiście odwracalna. Uwaga: nie wolno rozbijać lewych stron zależności!

Zależność $X \rightarrow Y$ nazywamy **trywialną**, jeśli spełnione jest $Y \subseteq X$, np. $AB \rightarrow A$. Zależności trywialne zachodzą zawsze i nie dostarczają żadnej informacji.

W posługiwaniu się zależnościami funkcyjnymi mogą pomóc **reguły wnioskowania Armstronga**:

- **zwrotność**: jeśli $Y \subseteq X$, to zachodzi $X \rightarrow Y$
- **rozszerzanie**: jeśli zachodzi $X \rightarrow Y$, to również $XZ \rightarrow YZ$ dla dowolnego Z
- **przechodniość**: jeśli zachodzą $X \rightarrow Y$ i $Y \rightarrow Z$, to również $X \rightarrow Z$

Klucze

Nadklucz to podzbiór N zbioru atrybutów relacji $R(A_1, A_2, \dots, A_n)$, jeśli zachodzi zależność funkcyjna

$$N \rightarrow A_1A_2\dots A_n$$

W szczególności zbiór wszystkich atrybutów relacji jest nadkluczem.

Klucz to nadklucz składający się z minimalnej liczby kolumn, czyli nadklucz, którego żaden podzbiór (oprócz jego samego) nie jest nadkluczem.

Przykład 2.

Niech w relacji **Zwierzaki**(imię, gatunek, waga, wiek) zachodzi zależność

$$\text{imię} \rightarrow \text{gatunek waga wiek}$$

Wówczas prawdziwe są następujące stwierdzenia:

- (imię, gatunek) jest nadkluczem w tej relacji,
- (imię, gatunek) nie jest kluczem, bo (imię) też jest nadkluczem,
- (imię) jest kluczem i nie ma innych kluczów.

Zestaw zadań

8.10. Dana jest relacja dwuargumentowa $R = (A, B)$ z zależnością funkcyjną: $A \rightarrow B$, przy czym w żadnej kolumnie nie ma wartości NULL. Niech a oznacza liczbę różnych wartości w kolumnie A , b oznacza liczbę różnych wartości w kolumnie B , oznacza to, że

- A. $a \leq b$
- B. $a \geq b$
- C. jeśli $a > 0$, to $b > 0$

8.11. Mamy daną relację $R(A, B, C, D)$, której kluczem jest AB . Wynika z tego zależność funkcyjna

- A. $ABC \rightarrow D$
- B. $B \rightarrow A$
- C. $CD \rightarrow AB$

8.12. Dla relacji $R(A, B, C, D)$ nie określono żadnych zależności funkcyjnych. Wynika z tego, że relacja R

- A. ma co najmniej jeden klucz
- B. ma co najwyżej jeden klucz
- C. ma 4 klucze

8.13. Dana jest tabela $R(A, B, C, D, E)$ z (jednymi) zależnościami funkcyjnymi

$$A \rightarrow B, \quad AB \rightarrow CD, \quad D \rightarrow ABCE.$$

Wynika z tego, że kluczem R jest

- A. A
- B. AB
- C. CD

8.4.

Redundancja, normalizacja

Jeden z celów, który staramy się osiągnąć podczas projektowania schematu bazy danych to unikanie **redundancji** (nadmiarowości). Redundancja oznacza, że niektóre informacje są zapisane w bazie danych wielokrotnie. Prowadzi to do różnych anomalii podczas modyfikacji i usuwania danych z bazy.

Przykład 1.

Oto przykład błędego projektu bazy danych. Przypuśćmy, że mamy relację **Zwierzaki**(imię, gatunek, waga, kontynent), w której kluczem jest imię, z następującymi zależnościami:

$$\text{imię} \rightarrow \text{gatunek waga kontynent}, \quad \text{gatunek} \rightarrow \text{kontynent}$$

Występuje redundancja: nie ma potrzeby przechowywania informacji na temat kontynentu pochodzenia danego zwierzęcia w każdym wierszu, bo możemy go łatwo wyznaczyć na podstawie gatunku. Tabela zawiera w ten sposób nadmiarowe, niepotrzebne dane.

Normalizacja to proces służący do przekształcania schematu w taki sposób, aby wyeliminować redundancję. Zależnie od potrzeb określa się różne poziomy normalizacji – zakresy usuwania redundancji, nazywane **postaciami normalnymi**.

Dotychczas formalnie zdefiniowano pięć (poziomów) postaci normalnych, choć my nie będziemy zajmować się ostatnią z nich. Tylko trzy pierwsze są powszechnie używane podczas projektowania baz danych.

Postacie normalne są ponumerowane kolejno, ale istnieje postać pośrednia BCNF między 3 i 4 poziomem. Postacie o wyższych numerach automatycznie (z definicji) spełniają warunki dla niższych postaci, dlatego na przykład relacja w drugiej postaci normalnej jest automatycznie w pierwszej postaci normalnej. Odwrotne wynikanie oczywiście nie zachodzi; drugą postać normalną otrzymujemy z pierwszej po nałożeniu dodatkowego warunku.

Proces normalizacji polega na **dekompozycji** tabel (rozkładu danej relacji na kilka innych) aż do otrzymania pożąданej postaci.

Pierwsza postać normalna (1NF) to najbardziej podstawowa postać normalna. W takiej relacji:

- każdy atrybut przyjmuje wartości niepodzielne (jest wartością skalarną, a nie np. listą),
- w danej kolumnie występuje tylko jeden typ danych,
- nie używa się kolejności wierszy do przekazania jakiekolwiek informacji.

Przykład 2.

Zdefiniujmy relację wypisującą dla każdego studenta listę przedmiotów, na które jest zapisany:

student	przedmioty
Grześ	BD, GAL
Patryk	BD, MD

Taka relacja nie jest w pierwszej postaci normalnej, ponieważ w atrybucie „przedmioty” występuje lista. Znormalizowana relacja wyglądałaby w następujący sposób:

student	przedmiot
Grześ	BD
Grześ	GAL
Patryk	BD
Patryk	MD

Relacja jest w **drugiej postaci normalnej (2NF)**, jeśli:

- jest w 1NF,
- jej każdy niekluczowy atrybut zależy funkcyjnie od całego klucza, a nie tylko jego części.

Przykład 3.

Rozszerzymy relację z poprzedniego przykładu: **Zajęcia**(student, kierunek, przedmiot, ćwiczeniowiec). Założamy przy tym, że każdy student studiuje jeden kierunek oraz że każdy przedmiot ma kilku ćwiczeniowców do wyboru. Zależności występujące w tej tabeli to

$$\text{student przedmiot} \rightarrow \text{ćwiczeniowiec}, \quad \text{student} \rightarrow \text{kierunek},$$

więc kluczem jest para (student, przedmiot).

Oto przykładowa zawartość tej tabeli:

student	kierunek	przedmiot	ćwiczeniowiec
Grześ	MAT	BD	Zbyszek
Grześ	MAT	GAL	Męcel
Patryk	INF	BD	Murlak
Patryk	INF	MD	Amal

Kierunek studiów zależy tylko od części klucza, czyli atrybutu „student”. Relacja nie jest więc w 2NF. Należy ją zdekomponować na dwie relacje: **Zajęcia**(student, przedmiot, ćwiczeniowiec) oraz **Kierunki**(student, kierunek).

Relacja jest w **trzeciej postaci normalnej** (3NF), jeśli:

- jest w 2NF,
- w każdej zależności lewa strona jest nadkluczem lub prawa strona zawiera jedynie atrybuty z kluczy (klucz może być kilka).

Oznacza to, że atrybuty niekluczowe powinny zależeć funkcyjnie wyłącznie od klucza i niczego więcej. Wykluczamy w ten sposób zależności przechodnie.

Przykład 4.

Rozważmy relację **Studia**(student, rok, stopień). Relacja ta posiada dwie zależności funkcyjne:

$$\text{student} \rightarrow \text{rok stopień}, \quad \text{rok} \rightarrow \text{stopień}$$

a jedynym kluczem jest atrybut „student”. Przykładowo:

student	rok	stopień
Grześ	3	licencjacki
Patryk	2	licencjacki
Paweł	4	magisterski

W przypadku drugiej zależności funkcyjnej ani lewa strona nie jest nadkluczem, ani prawa strona nie zawiera jedynie atrybutów z kluczy, więc relacja nie jest w 3NF.

Z praktycznego punktu widzenia możemy też dostrzec, że aktualizując w pewnym wierszu pole „rok” narządzamy się na potencjalne błędy danych, ponieważ moglibyśmy zapomnieć o jednoczesnej aktualizacji stopnia. Taką relację należy zdekomponować na **Lata**(student, rok) oraz **Stopnie**(rok, stopień).

Postać normalna Boyce'a-Codda (BCNF) jest bardzo podobna do 3NF. Relacja jest w BCNF, jeśli:

- jest w 3NF,

- w każdej zależności funkcyjnej lewa strona jest nadkluczem.

Przykład 5.

Rozważmy relację **Zajęcia**(student, przedmiot, ćwiczeniowiec). Zakładamy, że każdy student może chodzić na wiele przedmiotów, każdy przedmiot może być nauczany przez wielu ćwiczeniowców, ale każdy ćwiczeniowiec może uczyć tylko jednego przedmiotu. Występujące zależności funkcyjne to:

$$\begin{aligned} \text{student przedmiot} &\rightarrow \text{ćwiczeniowiec}, & \text{ćwiczeniowiec} &\rightarrow \text{przedmiot}, \\ \text{student ćwiczeniowiec} &\rightarrow \text{przedmiot}, \end{aligned}$$

więc klucze tej relacji to (student, przedmiot) oraz (student, ćwiczeniowiec). Przykładowo:

student	przedmiot	ćwiczeniowiec
Grześ	BD	Zbyszek
Grześ	GAL	Męcel
Patryk	BD	Murlak
Patryk	MD	Amal

Relacja ta jest w 3NF, ale nie jest w BCNF, ponieważ lewa strona drugiej zależności nie jest nadkluczem. Należy zdekomponować ją na **Ćwiczeniowcy**(ćwiczeniowiec, przedmiot) oraz **Zajęcia**(student, ćwiczeniowiec).

Znany jest ogólny **algorytm dekompozycji do BCNF**:

- dopóki dla jakiejś relacji R z zależnościami F istnieje zależność $X \rightarrow Y$ z rozłącznymi X, Y naruszającą warunek BCNF:
 - zdekomponuj R na $R'(XY)$ oraz $R''(\text{sort}(R) - Y)$, gdzie $\text{sort}(R)$ to zbiór wszystkich kolumn R

Przykład 6.

Dana jest relacja $R(ABCDEF)$ z zależnościami

$$AB \rightarrow C, \quad C \rightarrow E, \quad D \rightarrow E, \quad C \rightarrow D.$$

- Zaczynamy z $R_0(ABCDEF)$. Weźmy $X_0 = AB, Y_0 = C$. Zauważmy, że z X_0 wynika $ABCDE$, ale nie wynika F – nie jest to nadklucz, więc narusza reguły BCNF.
- Zgodnie z algorytmem rozbijamy R_0 na $R_1(ABC)$ i $R_2(ABDEF)$.
- R_1 jest w BCNF (w jej jedynej zależności funkcyjnej, $AB \rightarrow C$, lewa strona jest nadkluczem). W R_2 zachodzi $AB \rightarrow DE$, bierzemy więc $X_2 = AB, Y_2 = DE$. Analogicznie jak wcześniej, rozbijamy R_2 na $R_3(ABDE)$ oraz $R_4(ABF)$.
- R_4 jest w BCNF, z R_3 bierzymy $X_3 = D, Y_3 = E$.
- Rozbijamy na $R_5(ABD), R_6(DE)$. Cała baza jest teraz w BCNF.

Czasem zależności funkcyjne powodują kłopoty przy przejściu do BCNF, ponieważ po takim przekształceniu niektóre zależności zostają utracone. Widzieliśmy to już w Przykładzie 5, gdzie utraciliśmy zależność „student przedmiot \rightarrow ćwiczeniowiec”. Zobaczmy jeszcze poniższy przykład.

Przykład 7.

Rozważmy relację **Lokalizacje**(adres, miasto, kod pocztowy), która ma 2 klucze: (adres, miasto) oraz (adres, kod pocztowy) i 2 zależności funkcyjne:

$$\text{adres miasto} \rightarrow \text{kod pocztowy}, \quad \text{kod pocztowy} \rightarrow \text{miasto}$$

Zależność „kod pocztowy → miasto” narusza postać BCNF, więc musimy dokonać dekompozycji tabeli **Lokalizacje** na dwie relacje: **Adresy**(adres, kod pocztowy) oraz **Kody**(kod pocztowy, miasto). Zauważmy, że utraciliśmy w ten sposób zależność

adres miasto → kod pocztowy,

ponieważ atrybuty te nie występują w żadnej relacji.

To było na egzaminie

Dana jest relacja $R(A, B, C, D, E)$ oraz zależności funkcyjne $A \rightarrow BC, AC \rightarrow D, D \rightarrow E$. Wówczas

- A. relacja R ma dokładnie jeden klucz
 - B. R jest w 3NF
 - C. R można przekształcić w BCNF bez utraty zależności funkcyjnych
-
- A. Na początku ustalmy klucze relacji R . Widzimy, że z atrybutu A wynikają BC , więc korzystając z przechodniości zależności funkcyjnych, na podstawie A możemy wywnioskować też atrybuty D i E . Zatem kluczem jest atrybut A i nietrudno sprawdzić, że nie ma innych kluczy, co powoduje że A. jest prawdziwe.
 - B. Musimy sprawdzić warunki 3NF, tj. czy lewa strona każdej zależności jest nadkluczem lub czy prawa zawiera atrybuty tylko z kluczy. Lewe strony pierwszej i drugiej zależności są nadkluczami, więc spełniają warunek 3NF. Ostatnia zależność nie spełnia żadnego z warunków postaci 3NF, więc R nie jest w 3NF.
 - C. Przekształcimy relację R do postaci BCNF i sprawdzimy czy jakieś zależności zostały utracone. Jedyną zależnością naruszającą BCNF jest $D \rightarrow E$. Zgodnie z algorytmem dekompozycji do BCNF, dekomponujemy R do $R_1(DE)$ oraz $R_2(ABCD)$. Widzimy teraz, że obie relacje są w BCNF, a my nie utraciliśmy żadnych zależności.

Zestaw zadań

8.14. Z tego, że relacja R jest w drugiej postaci normalnej (2NF), wynika że

- A. każda kolumna zależy funkcyjnie od całego klucza głównego
- B. R jest w pierwszej postaci normalnej
- C. wszystkie pola nie będące polami klucza głównego są od niego zależne bezpośrednio

8.15. Dana jest relacja $R(A, B, C, D)$ w pierwszej postaci normalnej i zależności funkcyjne: $A \rightarrow B, B \rightarrow C, C \rightarrow A$. Wynika stąd, że relacja jest też w

- A. 2NF
- B. 3NF
- C. BCNF

8.16. Relacja R ma kolumny A, B, C, D, E i zależności funkcyjne $A \rightarrow BC, CA \rightarrow D, B \rightarrow E$. Wynika z tego, że

- A. relacja R ma dokładnie trzy klucze

- B. relacja R jest w trzeciej postaci normalnej
- C. schemat relacji R daje się sprowadzić do postaci Boyce'a-Codda z zachowaniem zależności funkcyjnych i informacji

8.17. Każda tabela w pierwszej postaci normalnej (1NF) mająca dokładnie dwie kolumny jest

- A. w drugiej postaci normalnej (2NF)
- B. w trzeciej postaci normalnej (3NF)
- C. w postaci normalnej Boyce'a-Codda (BCNF)

8.18. Każda relacja

- A. w postaci 3NF jest także w BCNF
- B. w postaci BCNF jest także w 3NF
- C. dwuargumentowa jest w postaci BCNF

8.5.

Transakcje i współbieżność

Transakcje to jedno z podstawowych pojęć dotyczących sposobów realizacji zapytań języka SQL we wspólnoczesnych systemach baz danych. Umożliwiają one współbieżny dostęp do zawartości bazy, dostarczając niezbędnych mechanizmów synchronizacji.

Istotą transakcji jest integrowanie kilku operacji w niepodzielną całość. Jako **transakcja** rozumieć będziemy ciąg operacji na bazie danych (**SELECT**, **DELETE**, **UPDATE**, **INSERT**) zakończony potwierdzeniem zmian (**COMMIT**) albo ich cofnięciem (poprzez błąd systemu lub „na życzenie” poprzez **ROLLBACK**).

W trakcie wykonywania transakcja może być wycofana w dowolnym momencie. Wszelkie wprowadzone przez nią zmiany danych zostaną wtedy zignorowane. Realizacja tego mechanizmu wymaga „tymczasowego” środowiska pracy – zmiany danych są tylko obliczane i zapisywane w specjalnym dzienniku transakcji. Po pomyślnym zakończeniu wykonywania transakcji następuje jej zatwierdzenie, w wyniku czego zmiany z dziennika są utrwalane w bazie danych.

Poziomy izolacji

Aby zapobiec pewnym anomaliom danych związanych ze współbieżnym wykonywaniem transakcji, wprowadza się pojęcie **poziomu izolacji** transakcji. Poziom izolacji opisuje, jak dana transakcja chce widzieć bazę danych (i nie ma on wpływu na widzenie bazy danych przez pozostałe transakcje).

Standard SQL definiuje 4 poziomy izolacji, wypisane tu od najbezpieczniejszego do najmniej restrykcyjnego:

- **serializable** – gwarantuje wykonywanie transakcji w taki sposób, jakby wykonywały się w całości (nie-podzielnie) jedna po drugiej
- **repeatable read** – kolejne odczyty (**SELECT**) w ramach jednej transakcji zwracają zawsze te same wiersze, ale mogą pojawić się dodatkowe wiersze (jeśli zostały w międzyczasie wstawione przez inne, zatwierdzone transakcje). Żadne już odczytane wiersze nie mogą jednak zniknąć lub zmienić swojej wartości
- **read committed** – analogiczny do *repeatable read*, ale przy kolejnych odczytach w ramach jednej transakcji, oprócz pojawienia się nowych wierszy, dotychczas odczytane wiersze mogą również zniknąć lub zmienić swoją wartość (jeśli zostały w międzyczasie usunięte lub zmodyfikowane przez inne, zatwierdzone transakcje)
- **read uncommitted** – odczyty w ramach takiej transakcji widzą zmiany dokonane przez inne jeszcze niezatwierdzone transakcje

Można więc wyszczególnić trzy zjawiska anomalii danych związanych ze współbieżnym dostępem do nich:

- **dirty read** – transakcja widzi lokalną zmianę wprowadzoną przez inną transakcję zanim ta druga za-twierdzi zmiany,
- **non-repeatable read** – przy powtórny wykonyaniu odczytu (w ramach jednej transakcji) ginie jakiś wiersz lub ma inne wartości,
- **phantom read** – przy powtórny wykonyaniu odczytu (w ramach jednej transakcji) pojawiają się nowe wiersze.

W poniżej tabeli zaznaczono, które z nich mogą wystąpić w każdym z czterech dostępnych poziomów izolacji:

	dirty read	non-repeatable read	phantom read
serializable	–	–	–
repeatable read	–	–	✓
read committed	–	✓	✓
read uncommitted	✓	✓	✓

Przykład 1.

W bazie istnieje tabela **Produkt**(nazwa, cena) o kluczu głównym „nazwa” i początkowych wartościach:

nazwa	cena
ołówek	20
długopis	30

Rozważmy poniższe dwie transakcje T_1, T_2 :

```
/* T1 */  
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
INSERT INTO Produkt VALUES (pióro, 40); -- operacja S1  
UPDATE Produkt SET cena = cena + 30 WHERE nazwa = 'ołówek'; -- operacja S2  
COMMIT;  
  
/* T2 */  
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
SELECT AVG(cena) AS a1 FROM Produkt; -- operacja S3  
SELECT AVG(cena) AS a2 FROM Produkt; -- operacja S4  
COMMIT;
```

Znajdziemy wszystkie możliwe wartości (a_1, a_2) odczytane przez transakcję T_2 .

Ponieważ obie transakcje są na poziomie izolacji *serializable*, jedynie możliwe przeploty to T_1T_2 albo T_2T_1 . Stąd wyjątkowymi odczytami (a_1, a_2), jakie możemy uzyskać, są kolejno (40, 40) i (25, 25).

W celu zapobiegnięcia konfliktom mogącym doprowadzić do zabronionych (na danym poziomie izolacji) anomali, używa się wewnętrznego **blokowania** dostępu do elementów używanych przez daną transakcję.

Przykład 2.

Jak zmieniłyby się odczyty (a_1, a_2), gdyby T_2 z poprzedniego przykładu była na poziomie izolacji *repeatable read*, a pozostałe dane pozostały bez zmian?

Poziom izolacji *repeatable read* zezwala na zajście *phantom read*: gdyby między wywołaniem S_3 i S_4 pojawiły się nowe wiersze w tabeli, to S_4 uwzględniałby je przy obliczaniu średniej.

Jedynym możliwym przeplotem realizującym tę sytuację jest $S_3T_1S_4$ (T_1 musi wykonać się w całości, ponieważ zmiany przez nią naniesione widzimy dopiero po ich zatwierdzeniu). Zachodzi tu jednak jeszcze jedna anomalia: wykonanie S_2 spowoduje *non-repeatable read* (aktualizację już odczytanego wiersza przez T_2), co nie jest dopuszczone na poziomie izolacji *repeatable read*, więc transakcja T_1 po dojściu do operacji S_2 zatrzyma się (bez wykonania **COMMIT**) i zaczeka na wykonanie T_2 . Jest to związane z blokadą założoną przez wykonującą się już transakcję T_2 .

Tym samym przeplot $S_3T_1S_4$ nie jest możliwy i jedynymi możliwymi parami (a_1, a_2) są nadal (40, 40) i (25, 25).

Przykład 3.

A co w przypadku, gdy T_2 będzie na poziomie izolacji *read committed*?

Read committed dopuszcza zajście *non-repeatable read* z sytuacji z poprzedniego przykładu. To umożliwia zajście przeplotu $S_3T_1S_4$ i, oprócz (40, 40) i (25, 25), będziemy w stanie uzyskać też wynik (25, 40).

Przykład 4.

Ostatnim rozważanym przypadkiem będzie T_2 na poziomie izolacji *read uncommitted*.

Ten poziom izolacji zezwala na wszystkie wyszczególnione przez nas anomalie, w szczególności *dirty reads*, czyli odczyty wprowadzonych zmian jeszcze przed ich zatwierdzeniem. Dodatkowymi możliwymi wynikami (a_1, a_2) są zatem:

- (25, 30) przy przeplocie $S_3S_1S_4S_2$,
- (30, 30) przy przeplocie $S_1S_3S_4S_2$,
- (30, 40) przy przeplocie $S_1S_3S_2S_4$.

8.6.

Fizyczna reprezentacja danych

Znajomość fizycznej reprezentacji danych pomaga nam optymalizować realizację zapytań bazodanowych. Ponieważ dane przechowywane są na dysku, algorytmy projektujemy uwzględniając użycie **modelu I/O**.

Dysk trzeba traktować inaczej niż pamięć, bo pojedynczy dostęp do dysku trwa miliony cykli procesora – to przynajmniej 30 razy wolniej w porównaniu do pamięci operacyjnej (RAM). Dlatego będziemy skupiali się głównie na ograniczeniu odczytów i zapisów z dysku, które są najbardziej kosztownymi operacjami.

W modelu I/O:

- dane są zorganizowane w tabele składające się z rekordów (wierszy)
- tabele czytamy i zapisujemy w **blokach** obejmujących pewną liczbę **kolejnych** wierszy
- dane na dysku mają pewną, ustaloną liczbę bloków (np. m), w pamięci RAM mieści się pewna inna, ustalona liczba bloków (np. n , przy czym zwykle $m \ll n$)

Koszt algorytmu to łączna liczba odczytów z dysku i zapisów na dysk. Projektując algorytmy w modelu I/O, będziemy więc zwracać uwagę na to, aby wczytany blok zawierał jak największą przydatnych danych i aby dla raz wczytanych danych zrobić jak największej, zanim się je zapisze.

Kluczowe komponenty algorytmiczne w bazach danych

Wyróżniamy trzy kluczowe komponenty algorytmów realizacji zapytań do baz danych. Rozważymy tu każdy z nich wraz z krótkim opisem na temat jego optymalnej postaci.

Pierwszym zagadnieniem jest **skanowanie**, czyli na przykład filtrowanie krotek tabeli względem zadanego warunku. Warto pamiętać, że:

- zamiast przetwarzać po jednej krotce, przetwarzamy po całym bloku
- wczytujemy blok po bloku do pamięci, pasujące krotki zostawiamy, resztę wyrzucamy
- kiedy nazbieramy cały blok pasujących krotek, zapisujemy go na dysk

Drugim aspektem jest **sortowanie**. Najbardziej optymalnym podejściem przy bazach danych jest zazwyczaj *merge sort*.

Przyjmijmy, że dane na dysku mają n bloków, a w pamięci mieści się jednocześnie m bloków. Schemat wykonania merge sorta jest następujący:

- podziel dane na porcje rozmiaru m bloków, posortuj każdą porcję w pamięci (dowolnym, szybkim algorytmem)
- po takim sortowaniu mamy n/m porcji
- łącz po m porcji aż zostanie tylko jedna (wynikowa):
 - wczytaj po jednym bloku każdej z m porcji
 - łącz jak w zwykłym algorytmie *merge sort*, przeglądając krotka po krotce:
 - * spośród pierwszych krotek w każdym z m bloków wybierz najmniejszą
 - * przenieś ją na bok do wypisania
 - za każdym razem, gdy uzbierasz cały blok wyjściowy, zapisz go na dysku jako kolejny
 - za każdym razem, gdy zużyjesz cały blok wejściowy, wczytaj kolejny z tej samej porcji

W ogólności można przyjąć, że taki algorytm *merge sort* w modelu I/O jest **liniowy**. Zarówno koszt zapisu, jak i koszt odczytu wynoszą $O(mn)$.

Ostatnim rozważanym zagadnieniem będzie **wyszukiwanie**. Jeśli chcemy uniknąć przeglądzania całych danych, potrzebujemy pomocniczej struktury danych. W algorytmie taka struktura nazywa się „słownik”, a w bazach danych – **indeks**. Indeksy są zapisane w pamięci i służą do szybszego obliczania zapytań.

Do dużych zapytań często potrzebne są indeksy złożone, których klucze składają się z kilku kolumn. Sztuka korzystania z indeksów złożonych polega na odpowiednim wyborze kolejności kolumn w indeksie. Zamiast korzystać z intuicyjnie „naturalnej” kolejności kolumn, należy zacząć od kolumny dającej największą redukcję. Budowa optymalnych indeksów może o rząd wielkości zmienić czas wykonania zapytania.

Przykład 1.

Rozważmy indeks obejmujący kolumny (kod firmy, numer konta, rodzaj transakcji) w pewnej tabeli bazy danych dla księgowości finansowej. Jeśli istnieją tylko dwie firmy, to kolumna numeru konta redukuje liczbę wierszy wynikowych znacznie bardziej niż kolumna kodu firmy.

Podobnie, typ transakcji redukuje dalej wiersze wynikowe bardziej niż kod firmy. Tak więc optymalna kolejność kolumn w indeksie to (numer konta, rodzaj transakcji, kod firmy).

Zestaw zadań

- 8.19. Założymy, że tabela T o kolumnach a i b zajmuje 1000 bloków dyskowych, a w pamięci operacyjnej jest miejsce na 11 bloków dyskowych. Wynika z tego, że ewaluacja zapytania

```
SELECT a FROM T WHERE b = (SELECT min(b) FROM T)
```

wymaga

- A. użycia indeksu
- B. posortowania tabeli T
- C. odczytania pewnego bloku tabeli T co najmniej 3 razy

- 8.20. Rozważmy tabele $R(A, B), S(B, C), T(C, A)$, przy czym w tabeli R kluczem jest para kolumn A, B , w tabeli S para kolumn B, C , a w tabeli T kolumna C . Optymalną kolejnością wyliczenia złączenia naturalnego tych trzech tabel jest

- A. $(R \bowtie S) \bowtie T$
- B. $(S \bowtie T) \bowtie R$
- C. $(T \bowtie R) \bowtie S$

8.7.

Rozwiązań

Rozwiązań

- 8.1. Przypuśćmy, że w tabeli R o kolumnach A, B, C para kolumn A, B jest kluczem. Założymy też, że w tabeli R nie występują wartości NULL, w kolumnie A pojawia się dokładnie k różnych wartości, w kolumnie B dokładnie l różnych wartości, a w kolumnie C dokładnie m różnych wartości. Wynika z tego, że

- NIE A. $\min(k, l) \leq m$
- NIE B. $k + l \geq m$
- TAK C. $k \cdot l \geq m$

- A. Prosty kontrprzykład: wystarczy, że mamy po dwie różne wartości w kolumnach A i B , natomiast kolumna C składa się wyłącznie z powtórzeń.
- B. Jeśli na przykład kolumny A i B mają po 3 różne wartości, to możemy ułożyć z nich 9 różnych kluczy, czyli może też istnieć 9 różnych wartości w kolumnie C .
- C. Skoro para kolumn A, B jest kluczem, możemy maksymalnie utworzyć $k \cdot l$ różnych wartości klucza – a wartości w kolumnie C nie może być więcej niż kluczy.

- 8.2. Dane są relacje R i Q , każda zawierająca dokładnie n krotek. Wynika z tego, że relacja $R \bowtie Q$ (złączenie naturalne R i Q) ma

- NIE A. co najmniej n krotek
- TAK B. co najwyżej n^2 krotek
- NIE C. dokładnie $2n$ krotek

- A. W najgorszym przypadku, gdy żadna z krotek nie spełnia warunków złączenia naturalnego, wynikowa relacja będzie rozmiaru 0, a zatem ograniczenie z dołu przez n jest niepoprawne.

- B. Wykonując złączenie naturalne dwóch relacji A i B (o liczbie krotek odpowiednio k i l), możemy ograniczyć liczbę wynikowych krotek od góry przez $k \cdot l$. Maksymalny przypadek osiągamy, gdy każda krotka w relacji A może być połączona z każdą krotką w relacji B , a zatem ograniczenie z góry przez n^2 jest poprawne.
- C. Liczba krotek w wynikowej relacji zależy od danych i spełniania warunków złączenia naturalnego, przez co posiadając tylko informacje z treści zadania nie jesteśmy w stanie jednoznacznie stwierdzić, że będzie ich dokładnie $2n$. Kontrprzykładem może być relacja z pusta z podpunktu A.

8.3. Dana jest tabela Sprawdzian:

student	kolokwium	egzamin
A	45	NULL
B	NULL	90
C	100	80

Wynik zapytania w SQL

```
SELECT student
FROM Sprawdzian
WHERE (kolokwium > egzamin AND egzamin > 75) OR kolokwium < 50
```

będzie zawierał identyfikator studenta

- TAK A. A
 NIE B. B
 TAK C. C

W celu ustalenia odpowiedzi rozważymy, które wiersze z tabeli spełniają warunki zawarte w klauzuli WHERE:

- A. Warunek $45 > \text{NULL}$ będzie miał wartość *undefined*, podobnie jak $\text{NULL} > 75$, stąd koniunkcja AND nie może zostać uznana za spełnioną. Jednakże warunek $\text{kolokwium} < 50$ jest prawdziwy ($45 < 50$), więc alternatywa OR jest spełniona i student A pojawi się w wyniku zapytania.
- B. Warunek $\text{NULL} > 90$ jest *undefined*, co wyklucza prawdziwość koniunkcji AND. Sprawdzając wynik $\text{kolokwium} \text{NULL} < 50$, również otrzymujemy *undefined*, przez co cała alternatywa OR nie jest spełniona i student B jest pominięty w wyniku zapytania.
- C. Koniunkcja AND jest w oczywisty sposób spełniona ($100 > 80$ oraz $80 > 75$), więc cała alternatywa OR również i student C pojawi się w wyniku zapytania.

8.4. W tabelach R i S kluczem głównym jest kolumna A , która jest jednocześnie jedyną kolumną. Wszystkie krotki z R zawarte w S zwróci zapytanie

- TAK A. `SELECT * FROM R WHERE EXISTS (SELECT * FROM S WHERE R.A = S.A)`
 NIE B. `SELECT R.A FROM R LEFT JOIN S ON R.A = S.A`
 TAK C. `(SELECT * FROM R) INTERSECT (SELECT * FROM S)`

- A. Zapytanie wybiera wszystkie elementy R , dla których istnieje bliźniacza krotka z S .
- B. Użycie `LEFT JOIN` sprawi, że wynik zapytania uwzględnii także krotki z R niezawarte w S (zgodnie z definicją, będą to porzucone krotki).
- C. Standardowe przecięcie tabel przy użyciu `INTERSECT` nie uwzględnia porzuconych krotek i wypisuje część wspólną obu tabel – a to oczywiście, w naszym przypadku, poprawny wynik.

8.5. Mamy dwie tablice A, B z kolumnami kolejno a, b . Istnieją takie ich zawartości, że zapytanie

```
SELECT * FROM A, B WHERE A.a = B.b
```

- TAK** A. zwróci pusty wynik
TAK B. zwróci dokładnie $|A| \cdot |B|$ krotek
NIE C. zapętli się

- A. Wystarczy, żeby kolumny $A.a$ oraz $B.b$ nie miały żadnego wspólnego elementu.
B. Aby osiągnąć ten wynik, musimy zwrócić pełny iloczyn kartezjański tabel A, B . Taką sytuację uzyskamy, gdy kolumny $A.a$ oraz $B.b$ zawierają (łącznie) tylko jedną, tę samą wartość, np.:

$$A = \begin{array}{c|c} & \begin{array}{c|c} a & b \\ \hline 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{array} \end{array} \quad B = \begin{array}{c|c} & \begin{array}{c|c} a & b \\ \hline 5 & 1 \\ 6 & 1 \end{array} \end{array}$$

- C. Zapytanie wykona się poprawnie, niezależnie od wartości tabel A, B .

- 8.6.** Mamy relację R z kolumnami a oraz b i wykonujemy następujące zapytanie SQL:

```
SELECT R1.a, R2.b FROM R AS R1, R AS R2
```

W wyniku

- NIE** A. znajdą się tylko krotki z R
TAK B. znajdą się wszystkie krotki z R
NIE C. nie znajdzie się żadna krotka z R

Zapytanie jest iloczynem kartezjańskim tabeli R ze sobą, więc w jego wyniku uzyskamy wszystkie możliwe pary postaci $(R.a, R.b)$.

- A. W wynikowej relacji mogą pojawić się krotki spoza R , na przykład biorąc

$$R = \begin{array}{c|c} & \begin{array}{c|c} a & b \\ \hline 1 & 2 \\ 3 & 4 \end{array} \end{array}$$

w wyniku dostaniemy m.in. krotkę $(1, 4)$, która nie należy do oryginalnej relacji.

- B. W iloczynie kartezjańskim uzyskamy wszystkie możliwe pary, w szczególności takie, które pierwotnie należały do R .
C. Oczywista konsekwencja B.

- 8.7.** Zapytanie `SELECT R.A FROM R` zwraca r wierszy, zapytanie `SELECT S.A FROM S` zwraca s wierszy, a zapytanie `SELECT R.A FROM R WHERE R.A NOT IN (SELECT S.A FROM S)` zwraca k wierszy. Wykonając, stąd, że

- NIE** A. $k = r - s$
TAK B. $r - s \leq k \leq r$
TAK C. $0 \leq k \leq r$

Analizując zapytania, dochodzimy do wniosku, że k jest liczbą tych wierszy z tabeli R , które nie występują w tabeli S .

- A. To oczywiście nie musi być prawdą: wystarczy, że tabela R zawiera przynajmniej jeden wiersz nie występujący w S .
B. W najgorszym przypadku, gdy $S \subseteq R$ mamy $k = r - s$. Szacując z drugiej strony, gdy $R \cap S = \emptyset$, jest $k = r$. Nierówności są więc prawdziwe.
C. Zapytanie zwróci pewien podzbiór tabeli R – jasne jest więc, że będzie to przynajmniej 0 oraz maksymalnie r wierszy.

- 8.8.** Dana jest tabela $R(a, b, c, d)$ oraz szkielet zapytania

```
SELECT [...]
FROM R
GROUP BY a, b;
```

Poprawne wyrażenie SQL otrzymamy, wstawiając w miejsce [...]

- TAK** A. $\text{MIN}(c + d)$
- TAK** B. a, b
- NIE** C. b, c

W zapytaniach z grupowaniem każda kolumna wymieniona w klauzuli SELECT musi albo być częścią frazy GROUP BY, albo być wykorzystana w funkcji agregującej.

- A. Kolumny c i d nie są częścią klauzuli GROUP BY, ale są wykorzystane w funkcji agregującej, a zatem jest to poprawne wyrażenie.
- B. Kolumny a i b są częścią klauzuli GROUP BY, a zatem jest to poprawne wyrażenie.
- C. Kolumna c nie jest częścią klauzuli GROUP BY ani nie jest wykorzystana w funkcji agregującej, a zatem jest to niepoprawne wyrażenie.

- 8.9.** Niech R będzie tabelą o dwóch kolumnach X, Y oraz 0 wierszach. Rozważmy dwie instrukcje:

```
INSERT INTO R VALUES ('a', 'b');      -- Instrukcja A
DELETE FROM R WHERE X = 'a';          -- Instrukcja B
```

W związku z tym

- NIE** A. dwukrotne wykonanie instrukcji A daje taki sam wynik jak jej jednokrotne wykonanie
 - TAK** B. jeśli kolumna X jest kluczem głównym, to dwukrotne wykonanie instrukcji A skutkuje błędem
 - NIE** C. wykonanie instrukcji B skutkuje błędem
- A. Jako że SQL nie jest algebrą relacji, powtórzenia w tabelach nie są automatycznie usuwane i są dozwolone. W związku z tym po dwukrotnym wykonaniu instrukcji wstawiania w tabeli znajdą się dwa (identyczne) wiersze, a po jednokrotnym – jeden wiersz.
 - B. Jeśli kolumna X jest kluczem głównym, to nie może zawierać zduplikowanych wartości. Próba wstawienia duplikatu wartości 'a' w kolumnie X spowoduje naruszenie ograniczenia klucza głównego, co w SQL poskutkuje błędem.
 - C. Wykonanie instrukcji usuwania na pustej tabeli nie wprowadzi żadnych zmian (żaden wiersz nie spełni warunku podanego w WHERE). Usunięcie nieistniejącego wiersza nie powoduje błędu SQL.

- 8.10.** Dana jest relacja dwuargumentowa $R = (A, B)$ z zależnością funkcyjną: $A \rightarrow B$, przy czym w żadnej kolumnie nie ma wartości NULL. Niech a oznacza liczbę różnych wartości w kolumnie A , b oznacza liczbę różnych wartości w kolumnie B , oznacza to, że

- NIE** A. $a \leq b$
- TAK** B. $a \geq b$
- TAK** C. jeśli $a > 0$, to $b > 0$

Zależność $A \rightarrow B$ wyklucza sytuację $b > a$ – zauważmy, że wtedy pewne dwie różne wartości b_1, b_2 byłyby przypisane do tej samej wartości a_1 , ale z zależnością $A \rightarrow B$ wynikałoby, że $b_1 = b_2$, co prowadzi do sprzeczności. Sytuacje $a = b$ oraz $a > b$ są nietrudne do uzyskania (w pierwszym przypadku każdej wartości w kolumnie A jest przypisana unikalna wartość w kolumnie B , w drugim przypadku pewnym dwóm wartościom z kolumny A jest przypisana ta sama wartość w kolumnie B). W związku z tym podpunkt **A**. jest fałszywy i podpunkt **B**. jest prawdziwy.

Podpunkt **C**. jest także w oczywisty sposób prawdziwy – jeśli $a > 0$, to istnieje pewna wartość a_1 , a zależność $A \rightarrow B$ (oraz brak NULL-i) implikuje istnienie przypisanej do a_1 wartości b_1 , stąd także $b > 0$.

8.11. Mamy daną relację $R(A, B, C, D)$, której kluczem jest AB . Wynika z tego zależność funkcyjna

TAK A. $ABC \rightarrow D$

NIE B. $B \rightarrow A$

NIE C. $CD \rightarrow AB$

- A. Z definicji klucza wiemy, że zachodzi $AB \rightarrow D$, więc w szczególności także $ABC \rightarrow D$.
- B. Gdyby zachodziło $B \rightarrow A$, to AB nie byłoby kluczem (który z definicji jest minimalny).
- C. Z definicji klucza nie wynika, że zachodzi $CD \rightarrow AB$ oraz nie mamy informacji o żadnych innych zależnościach funkcyjnych.

8.12. Dla relacji $R(A, B, C, D)$ nie określono żadnych zależności funkcyjnych. Wynika z tego, że relacja R

TAK A. ma co najmniej jeden klucz

TAK B. ma co najwyżej jeden klucz

NIE C. ma 4 klucze

Skoro relacja nie ma żadnych zależności funkcyjnych, to kluczem są wszystkie jej kolumny: $ABCD$. Istnieje zatem dokładnie jeden klucz.

8.13. Dana jest tabela $R(A, B, C, D, E)$ z (jedynymi) zależnościami funkcyjnymi

$$A \rightarrow B, \quad AB \rightarrow CD, \quad D \rightarrow ABCE.$$

Wynika z tego, że kluczem R jest

TAK A. A

NIE B. AB

NIE C. CD

Ponieważ z A wynika B , to również z A wynika CD . Dodatkowo, ponieważ z D wynika E , to z A również wynika E . Zachodzi więc zależność $A \rightarrow ABCDE$, czyli A jest nadkluczem, a skoro nie ma mniejszych nadkluczów, to A jest także kluczem.

AB i CD nie mogą być kluczami – nie byłyby minimalne możliwe.

8.14. Z tego, że relacja R jest w drugiej postaci normalnej (2NF), wynika, że

NIE A. każda kolumna zależy funkcyjnie od całego klucza głównego

TAK B. R jest w pierwszej postaci normalnej

TAK C. wszystkie pola nie będące polami klucza głównego są od niego zależne bezpośrednio

- A. Warunek 2NF brzmi „każdy niekluczowy atrybut zależy funkcyjnie od całego klucza, a nie tylko jego części”. Mowa tu więc o atrybutach niekluczowych, a nie o wszystkich kolumnach. Kontrprzykładem może być relacja $R(A, B, C)$ z zależnościami $A \rightarrow B, B \rightarrow A$, w której kluczami są AC i BC , a kolumna B zależy wyłącznie od A .
- B. Zgodnie z definicją 2NF (i ogólną ideą postaci normalnych), każda relacja będąca w 2NF jest również w 1NF.
- C. Ponieważ zależności funkcyjne są przechodnie i rozszerzalne (reguły wnioskowania Armstronga), słowo „bezpośrednio” nie ma tu większego znaczenia. Odpowiedź jest oczywiście prawdziwa: w 2NF każda kolumna niebędąca kluczową zależy bezpośrednio od całego klucza, a nie od jego części.

8.15. Dana jest relacja $R(A, B, C, D)$ w pierwszej postaci normalnej i zależności funkcyjne: $A \rightarrow B, B \rightarrow C, C \rightarrow A$. Wynika stąd, że relacja jest też w

TAK A. 2NF

TAK B. 3NF

NIE C. BCNF

- A. Wiemy, że relacja jest w 1NF. Kluczami są AD , BD oraz CD , więc nie ma niekluczowych atrybutów i w szczególności nie są one zależne tylko od części klucza, stąd relacja jest również w 2NF.
- B. Relacja jest w 2NF. Ponadto, z lewej strony każdej zależności funkcyjnej musi być nadklucz albo z prawej strony jedynie elementy z klucza. Tutaj zawsze zachodzi warunek dla prawej strony (łatwo to sprawdzić), więc jest to 3NF.
- C. Aby relacja była w BCNF, z lewej strony każdej zależności musi znajdować się nadklucz, a w tym przypadku ani A , ani B , ani C nie są nadkluczami.

8.16. Relacja R ma kolumny A, B, C, D, E i zależności funkcyjne $A \rightarrow BC$, $CA \rightarrow D$, $B \rightarrow E$. Wynika z tego, że

NIE A. relacja R ma dokładnie trzy klucze

NIE B. relacja R jest w trzeciej postaci normalnej

NIE C. schemat relacji R daje się sprowadzić do postaci Boyce'a-Codda z zachowaniem zależności funkcyjnych i informacji

- A. Klucz to najmniejszy nadklucz i w tym przypadku jest nim A . Żadna inna kolumna sama w sobie nie jest nadkluczem.
- B. Żeby relacja była w 3NF, to po lewej stronie każdej zależności musi znajdować się nadklucz lub prawa strona musi zawierać jedynie atrybuty z kluczy. W naszym przypadku A jest kluczem, a B w oczywisty sposób nie jest nadkluczem, więc zależność $B \rightarrow E$ psuje 3NF.
- C. Zależność $B \rightarrow E$ zaburza BCNF, więc zgodnie z algorytmem dekompozycji do BCNF możemy rozbić R na dwie relacje $R_1(BE)$ i $R_2(ABCD)$. Zachowane zostaną wszystkie zależności funkcyjne, a obydwie relacje są w BCNF. Problem w tym, że nie mamy założenia, że relacja R jest w 1NF. Jeśli nie jest, to nie da się tego zrobić nie tracąc informacji.

8.17. Każda tabela w pierwszej postaci normalnej (1NF) mająca dokładnie dwie kolumny jest

TAK A. w drugiej postaci normalnej (2NF)

TAK B. w trzeciej postaci normalnej (3NF)

TAK C. w postaci normalnej Boyce'a-Codda (BCNF)

W przypadku A.: przy dwóch kolumnach nie może wystąpić sytuacja, w której niekluczowy atrybut zależy funkcyjnie od części klucza – klucz musiałby się składać co najmniej z dwóch kolumn oraz jeszcze jedna kolumna musiałaby stanowić niekluczowy atrybut. W związku z tym relacja jest w 2NF.

Dla B. i C.: przy dwóch kolumnach lewa strona każdej zależności funkcyjnej musi być nadkluczem, bo determinuje wszystkie wartości. Relacja jest więc w 3NF i BCNF.

8.18. Każda relacja

NIE A. w postaci 3NF jest także w BCNF

TAK B. w postaci BCNF jest także w 3NF

NIE C. dwuargumentowa jest w postaci BCNF

- A. Niekoniecznie – każda relacja w BCNF jest w 3NF (wprost z definicji), ale już nie na odwrót.
- B. Tak, wprost z definicji BCNF.
- C. Nie jest, nie musi być nawet w 1NF. Wystarczy na przykład, żeby jedna z kolumn trzymała wartości typu listowego, a nie będzie spełniony warunek z 1NF o niepodzielności danych.

- 8.19.** Założymy, że tabela T o kolumnach a i b zajmuje 1000 bloków dyskowych, a w pamięci operacyjnej jest miejsce na 11 bloków dyskowych. Wynika z tego, że ewaluacja zapytania

```
SELECT a FROM T WHERE b = (SELECT min(b) FROM T)
```

wymaga

- NIE** A. użycia indeksu
- NIE** B. posortowania tabeli T
- NIE** C. odczytania pewnego bloku tabeli T co najmniej 3 razy

Rozważmy najbardziej brutalny algorytm: będziemy przechodzić przez całą tabelę, wczytując ją blok po bloku (wystarczy nam miejsce na 1 blok w pamięci). Przy pierwszym przejściu znajdziemy minimalną wartość b , a przy drugim będziemy szukać wierszy, w których wartość b jest równa tej minimalnej.

Nie używamy tutaj ani indeksów, ani sortowania, więc A. i B. są fałszywe. Dodatkowo, każdy blok tabeli zostanie odczytany 2 razy, co pokazuje fałszywość podpunktu C.

Dygresja: opisane wyżej podejście można by zoptymalizować do jednego odczytu, tworząc indeks, który pozwala nam „od razu” wyznaczyć $\min(b)$.

- 8.20.** Rozważmy tabele $R(A, B), S(B, C), T(C, A)$, przy czym w tabeli R kluczem jest para kolumn A, B , w tabeli S para kolumn B, C , a w tabeli T kolumna C . Optymalną kolejnością wyliczenia złączenia naturalnego tych trzech tabel jest

- NIE** A. $(R \bowtie S) \bowtie T$
- TAK** B. $(S \bowtie T) \bowtie R$
- NIE** C. $(T \bowtie R) \bowtie S$

Złączenie naturalne wymaga zgodności w kolumnach o tej samej nazwie w łączonych tabelach. Optymalizacja zapytania z zadania dąży do zminimalizowania rozmiaru pośrednich tabel wynikowych.

Zaczynając od złączenia S i T ,łączymy je po kolumnie C i ponieważ jest ona kluczem tabeli T , wynik będzie miał tyle samo wierszy, co tabela S (lub mniej, jeśli w $C.S$ są wartości spoza $C.T$). Następnie łączymy $(S \bowtie T)$ z R po kolumnach A, B .

Nietrudno zauważyć, że pozostałe warianty są mniej optymalne, ponieważ pierwsze wykonane złączenie może stworzyć dużą tabelę pośrednią, co zmniejszy wydajność całego zapytania.

9

Programowanie współbieżne

Materiały teoretyczne z programowania współbieżnego zostały opracowane na podstawie slajdów Marcina Engela oraz tego dokumentu.

Podstawa programowa

1. **Poprawność** programów współbieżnych.
2. **Mechanizmy synchronizacji** programów współbieżnych w systemach scentralizowanych i rozproszonych.
3. **Klasyczne problemy współbieżności:** problem wzajemnego wykluczania, producenta-konsumenta, czytelników i pisarzy, pięciu filozofów.
4. **Algorytmy rozproszone:** wzajemne wykluczanie, synchronizacja zegarów logicznych, uzgadnianie.
5. Wsparcie dla współbieżności w językach programowania **Java**, **C++** oraz w systemie operacyjnym **Unix**.

9.1.

Podstawy programowania współbieżnego

Program współbieżny składa się ze skończonej liczby procesów sekwencyjnych. Każdy proces sekwencyjny wykonuje ciąg **operacji atomowych** (niepodzielnych, wykonujących się w całości albo wcale).

Jako **równoległość** będziemy rozumieć faktyczne jednocześnie wykonywanie wielu czynności, natomiast **współbieżność** to tylko abstrakcja (złudzenie) równoległości.

Programy współbieżne wykonują się w **przeplocie**: polega on na cyklicznym przełączaniu wykonania między różnymi wątkami lub procesami w celu wykonywania ich operacji atomowych. Jednocześnie tylko jeden wątek/proces może być aktywny, pozostałe wtedy oczekują na przydzielenie czasu wykonania (są *zawieszone*). Przeplot jest więc mechanizmem, który umożliwia wykonanie fragmentów kodu różnych wątków/procesów w szybki i nieregularny sposób, uzyskując wspomniane złudzenie równoległości.

To było na egzaminie

Zmienna *x* jest zmienną globalną o wartości początkowej 0. W systemie wykonują się współbieżnie dwa procesy o następującej treści:

```
process P() {
    for (int i = 1; i <= 5; i++)
        x = x + 1;
}
```

Po zakończeniu wykonania obu procesów wartość zmiennej *x* jest

- A. nie mniejsza niż 5
- B. równa 10
- C. mniejsza niż 10

Wbrew pozorom, linia kodu $x = x + 1$; jest złożona z trzech operacji atomowych:

1. odczytanie wartości zmiennej x
2. obliczenie wartości $x + 1$
3. przypisanie $x \mapsto x + 1$

Pokażemy kontrprzykład do każdego z podpunktów:

A. Oznaczmy poszczególne procesy przez P_1 i P_2 oraz rozważmy następujący przeplot:

- P_1 odczytuje $x = 0$ i zawiesza się.
- P_2 robi 4 obroty pętli. Wtedy $x = 4$ i zanim P_2 rozpocznie piąty obrót pętli, zawiesza się.
- P_1 zapisuje $x = 1$ i zawiesza się.
- P_2 odczytuje wartość $x = 1$ i zawiesza się.
- P_1 wykonuje wszystkie pozostałe obroty pętli i zapisując $x = 5$, kończy przebieg.
- P_2 kończy ostatni obrót pętli, zapisując $x = 2$.

Można zauważyć, że $x = 2$ jest najmniejszą możliwą wartością do uzyskania w tym zadaniu.

B. Kontrprzykład jak do podpunktu **A**.

C. Wartość $x = 10$ możemy uzyskać, wykonując procesy w całości jeden po drugim.

Stąd wszystkie odpowiedzi są fałszywe.

■ Właściwości programów współbieżnych

Programy współbieżne są narażone na zupełnie nowe rodzaje problemów, które nie występują w programowaniu sekwencyjnym. Wyróżniamy dwie właściwości programów, które powinny być zapewnione, aby program współbieżny działał prawidłowo.

Pierwszą z nich jest **bezpieczeństwo**. Jest to zapewnienie, że nigdy nie dojdzie do sytuacji niepożąданej, którą definiuje specyfikacja problemu synchronizacyjnego. Bezpieczeństwo jest odpowiednikiem częściowej poprawności programu sekwencyjnego.

Drugą właściwością jest **życiowość**. Zapewnia ona, że każdy proces, który chce wykonać pewną akcję, w skończonym czasie będzie mógł to zrobić. Życiowość jest bezpośrednio związana z wykonaniem programu współbieżnego i jest odpowiednikiem całkowitej poprawności programu sekwencyjnego (razem z właściwością bezpieczeństwa).

Przykład 1.

Załóżmy, że w systemie wykonują się procesy, które mają dostęp do wspólnego pliku. Jednocześnie z pliku może korzystać tylko jeden proces. W tak postawionym problemie synchronizacyjnym:

- właściwość **bezpieczeństwa** oznacza zapewnienie, że w danej chwili z pliku współdzielonego będzie korzystał maksymalnie jeden proces,

- własność **żywotności** oznacza zapewnienie, że każdy proces, który chce uzyskać dostęp do pliku wspólnego, będzie mógł to zrobić w skończonym czasie.

■ Dostęp do wspólnych zasobów

W programowaniu współbieżnym projektujemy rozwiązania dostępu kilku wątków/procesów do wspólnych zasobów (np. pamięci współdzielonej, plików czy urządzeń).

Sekcja lokalna to część kodu wątku/procesu, która nie korzysta z żadnych współdzielonych zasobów. W sekcji lokalnej wątek/proces może wykonywać operacje na swoich prywatnych zmiennych i lokalnych danych. Sekcja lokalna jest **bezpieczna do równoczesnego wykonywania** przez wiele wątków/procesów.

Sekcja krytyczna to część kodu, w której wątek/proces korzysta z współdzielonych zasobów. Dostęp do sekcji krytycznej musi być odpowiednio synchronizowany, aby zapobiec sytuacjom, które mogą prowadzić do nieprzewidywalnych wyników i błędów w programie. **Tylko jeden wątek lub proces może znajdować się w sekcji krytycznej** w danym czasie.

Niepoprawna synchronizacja dostępu do wspólnych zasobów może powodować problemy.

Zakleszczenie (ang. *deadlock*, globalny brak żywotności) to sytuacja, w której co najmniej dwie różne akcje czekają na siebie nawzajem, więc żadna nie może się zakończyć. Występuje, gdy wiele zadań w tym samym czasie konkuuruje o wyłączny dostęp do zasobów.

Przykład 2.

Załóżmy, że w systemie wykonują się dwa wątki *A, B*, które mają dostęp do dwóch zasobów *a, b*. Dany zasób może być jednocześnie używany przez jeden wątek.

Oto kod wątku *A*:

```
acquire(a); // Wykonywanie operacji na zasobie a  
acquire(b); // Wykonywanie operacji na zasobie b  
release(a);  
release(b);
```

oraz kod wątku *B*:

```
acquire(b); // Wykonywanie operacji na zasobie b  
acquire(a); // Wykonywanie operacji na zasobie a  
release(a);  
release(b);
```

Jeśli przeplot zostanie ustalony tak, że wątek *A* zablokuje zasób *a*, a wątek *B* – zasób *b*, to oba wątki utkną w zakleszczeniu: żaden z nich nie będzie mógł wykonać kolejnej operacji, jaką jest uzyskanie dostępu do zasobu, który aktualnie blokuje inny wątek.

Zagłodzenie (ang. *starvation*, lokalny brak żywotności) to sytuacja, w której proces czeka w nieskończoność, gdyż zdarzenie, na które czeka, zawsze powoduje wznowienie innego procesu.

Przykład 3.

Załóżmy, że mamy program, który korzysta z wątków do wykonania pewnego zadania. Wątki starają się uzyskać dostęp do wspólnej drukarki, aby wydrukować pewien dokument.

Oto kod każdego z wątków:

```
| while (true) {
```

```

acquire(printer);
if isMyTurn() {
    printer.print();
    release(printer);
    break;
} else {
    release(printer);
    continue;
}
}

```

W tym kodzie wątek wykonuje pętlę w nieskończoność, próbując uzyskać blokadę na drukarce. Jeśli jest jego kolej, wątek drukuje dokument, zwalnia blokadę i kończy działanie. W przeciwnym razie wątek zwalnia blokadę i ponownie próbuje uzyskać dostęp.

Zakładając, że istnieje więcej niż jeden wątek wykonujący ten kod, zagłodzenie może wystąpić, jeśli pewien wątek ciągle nie ma możliwości uzyskania dostępu do drukarki, bo jest blokowany przez pozostałe wątki. Na przykład, jeśli inne wątki stale zgłaszały swoje żądania przed nim lub są one traktowane bardziej priorytetowo, wątek może pozostać zablokowany w pętli **while**, nie mając szansy na wydrukowanie dokumentu. W efekcie wątek ten doświadcza zagłodzenia, ponieważ nie jest w stanie wykonać swojego zadania pomimo ciągłych prób.

Zagłodzenie może wystąpić, gdy mechanizmy zarządzania kolejnością dostępu do współdzielonych zasobów nie są odpowiednio zaimplementowane lub uwzględnione. W programowaniu współbieżnym zakładamy, że system operacyjny – pełniący funkcję zarządcy dostępu do czasu wykonania – jest **uczciwy**, czyli zapewnia, że każdy proces gotowy do wykonania stanie się w końcu aktywny.

Aktywne oczekiwanie

Oczekiwanie na wspólne zasoby może zostać zrealizowane w postaci

```

while (blokadaAktywna()) {
    // Nic nie rób
}

```

W ten sposób wątek/proces wejdzie do sekcji krytycznej dopiero wtedy, gdy zasoby, z których chce skorzystać, będą dla niego dostępne (tj. nie będą zajmowane przez inne wątki/procesy). Taki sposób realizacji, w którym wątek/proces na bieżąco sprawdza, czy dany warunek jest spełniony, nazywamy **wirującą blokadą** (ang. **spinlock**). Działa ona na zasadzie **aktywnego oczekiwania** – zużywa czas procesora, wykonując cały czas pustą pętlę.

Spinlock:

- ma możliwość zagłodzenia,
- jest stosowany w praktyce do blokowania na krótki czas, zwłaszcza na poziomie jądra systemu,
- pozwala na uniknięcie kosztownego czasowo wstrzymania procesu przez system i przełączenia kontekstu na kod systemowy,
- w systemach z pamięcią podręczną wykorzystuje aktywne lokalne oczekiwanie, tj. sprawdza wartość zapisaną w pamięci podręcznej (ma to związek z szybkością dostępu do pamięci podręcznej w porównaniu do pamięci głównej).

Zazwyczaj jednak ten rodzaj synchronizacji jest szczególnie nieefektywny. Dużo lepszą metodą jest wykorzystanie jednego z dostępnych mechanizmów synchronizacji działających na zasadzie usypiania wątku, które zostaną opisane w kolejnych rozdziałach.

9.1. Rozważmy następujący program wykonywany przez trzy procesy:

```
int x = 0;

process P(int id) { // id = 0, 1, 2
    int y;
    for (i = 0; i < 5; i++) {
        y = x;
        y = y + 1;
        x = y;
    }
}
```

Istnieje taki przeplot, że wartość zmiennej x po zakończeniu wszystkich trzech procesów jest równa

- A. 15
- B. 3
- C. 2

9.2. Własność żywotności rozwiązania problemu wzajemnego wykluczania oznacza, że

- A. każdy proces kiedyś wejdzie do sekcji krytycznej
- B. w sekcji krytycznej zawsze znajdzie się co najwyżej jeden proces
- C. każdy proces wchodzi do sekcji krytycznej bez czekania

9.3. Dany jest następujący program wzajemnego wykluczania dla dwóch procesów:

```
int jest[2] = (0, 0);

process P(int id) { /* id = 0 lub id = 1 */
    while (true) {
        /* sekcja lokalna */
        while (jest[1 - id]);           // (*)
        jest[id] = 1;                  // (*)
        /* sekcja krytyczna */
        jest[id] = 0;
    }
}
```

Taki program

- A. ma własność bezpieczeństwa
- B. ma własność żywotności
- C. będzie miał własność żywotności, kiedy zamienimy kolejność wierszy oznaczonych gwiazdką

9.4. Dany jest kod dla dwóch procesów:

```
boolean czeka1, czeka2;

process P1() {
    czeka1 = false;
    while (true) {
        czeka1 = true;
        while (czeka2) {}
        // Sekcja krytyczna.
```

```

        czeka1 = false;
    }
}

process P2() {
    czeka2 = false;
    while (true) {
        czeka2 = true;
        while (czeka1) {}
        // Sekcja krytyczna.
        czeka2 = false;
    }
}

```

Taki program

- A. wykorzystuje aktywne czekanie
- B. zapewnia żywotność
- C. zapewnia bezpieczeństwo

9.5. Synchronizacja za pomocą blokad wirujących (ang. *spinlock*)

- A. pozwala uniknąć przełączania kontekstu niezbędnego w przypadku konieczności wstrzynania procesu
- B. zapewnia żywotność
- C. w systemach z pamięcią podręczną wykorzystuje lokalne aktywne oczekiwanie (sprawdzana jest wartość zapisana w pamięci podrycznej)

9.2.

Semafora

Semafora to najbardziej podstawowe mechanizmy synchronizacji nie używające aktywnego oczekiwania. Rozróżniamy kilka rodzajów semaforów, które omówimy w tym rozdziale.

Semafor ogólny to abstrakcyjny typ danych z operacjami:

- **inicjacji** (od razu przy deklaracji, poza procesami),
- **opuszczenia P()** (próby przejścia przez semafor),
- **podniesienia V()**.

Operacje P() i V() są atomowe oraz wykluczają się nawzajem. Nie są dopuszczalne żadne inne operacje na semaforze.

Semafor ogólny interpretować możemy jako zmienną całkowitoliczbową przyjmującą wartości nieujemne. Gdy semafor ma wartość 0, to mówimy, że jest **zamknięty**, a jeśli większą od 0, to jest **otwarty**.

Jeśli semafor jest otwarty, to operacja P() zmniejsza jego wartość o 1. Jeśli zaś jest zamknięty, to proces wykonujący operację P() jest wstrzymywany.

Operacja V() powoduje zwiększenie wartości semafora o 1, jeśli żaden proces na tym semaforze nie czeka. W przeciwnym przypadku któryś z czekających procesów jest wznowiany.

Rodzaje semaforów

Semafor Dijkstry to taki, w którym **nie zakładamy nic o implementacji czekania ani o kolejności budzenia procesów**. Jego implementacja wygląda następująco:

```
P(S):  
    czekaj aż S > 0  
    S := S - 1
```

```
V(S):  
    S := S + 1
```

Semafor słaby działa następująco:

```
P(S):  
    if S > 0:  
        S := S - 1  
    else:  
        wstrzymaj wykonujący proces na semaforze S
```

```
V(S):  
    if jakiś proces czeka na S:  
        wznów dowolnie wybrany proces spośród oczekujących na S  
    else:  
        S := S + 1
```

Semafor silny (uczciwość słaba) działa następująco:

```
P(S):  
    if S > 0:  
        S := S - 1  
    else:  
        wstrzymaj wykonujący proces w kolejce procesów związanych z semaforem S
```



```
V(S):  
    if jakiś proces czeka na S:  
        wznów pierwszy proces z kolejki oczekujących na S  
    else:  
        S := S + 1
```

Semafor silnie uczciwy (uczciwość mocna) działa następująco:

```
P(S):  
    if S > 0:  
        S := S - 1  
    else:  
        wstrzymaj wykonujący proces na semaforze S
```



```
V(S):  
    if jakiś proces czeka na S:  
        wznów dowolny proces spośród oczekujących na S  
    else:  
        S := S + 1
```

Przy czym jeśli operacja V() będzie wykonana nieskończenie wiele razy, to w końcu każdy proces oczekujący na S zostanie wznowiony (silna uczciwość). W dalszych rozważaniach przyjmujemy tę właśnie definicję.

Semafor binarny (ang. *mutex*) to semafor, który może przyjmować jedynie wartości 0 lub 1. Podniesienie takiego semafora, gdy już jest otwarty, powoduje błąd.

Semafor uogólniony działa następująco:

```
P(S, n):  
    czekaj, aż S >= n  
    S := S - n
```

```
V(S, n):  
    S := S + n
```

Semafor dwustronnie ograniczony przyjmuje wartości z przedziału $[0, M]$, gdzie M jest wartością podawaną przy inicjalizacji semafora:

```
P(S):  
    czekaj, aż S >= 0  
    S := S - 1
```

```
V(S):  
    czekaj, aż S < M  
    S := S + 1
```

Dziedziczenie sekcji krytycznej

Jedną z technik rozwiązywania problemów synchronizacyjnych jest **dziedziczenie sekcji krytycznej**. Proces, który faktycznie kogoś budzi, nie podnosi semafora, obudzony proces zastaje zamknięty semafor (dziedziczy sekcję krytyczną) i musi podnieść go w imieniu procesu, który go obudził. Dzięki temu między obudzeniem a faktycznym wznowieniem działania przez proces nic nie może się zdarzyć.

Przykład 1.

Poniższy przykład pokazuje prawidłową implementację techniki dziedziczenia sekcji krytycznej. Proces kończący pracę sprawdza, czy może kogoś obudzić, i jeśli tak, to nie zwalnia muteksa – ten obowiązek zostaje zrzucony na proces, który został wznowiony.

```
binarySemaphore mutex = 1;  
binarySemaphore delay = 0;  
int ilu_czeka = 0;  
  
// Protokół wstępny  
P(mutex);  
if (trzeba poczekać) {  
    ilu_czeka++;  
    V(mutex);  
    P(delay); // Dziedziczenie sekcji krytycznej  
    ilu_czeka--;  
}  
...  
V(mutex);  
  
// Protokół końcowy  
P(mutex);  
...  
if (można kogoś wznowić && ilu_czeka > 0)  
    V(delay); // Nie zwalniamy muteksa  
else  
    V(mutex);
```

Zestaw zadań

- 9.6. W sali wystawowej może jednocześnie przebywać co najwyżej K osób. Wystawę zwiedzają grupy reprezentowane przez procesy Grupa. Parametrem procesu Grupa jest liczba osób w grupie, będąca liczbą dodatnią mniejszą bądź równą K . Jeśli grupa w całości nie mieści się w sali, musi poczekać. Rozważmy

następujące rozwiązanie tego zadania korzystające z semafora silnego S o wartości początkowej K .

```
process Grupa (int liczność) {
    while (true) {
        for (int i = 0; i < liczność; ++i) P(S); // Grupa czeka na miejsce.
        // Grupa zwiedza wystawę.
        for (int i = 0; i < liczność; ++i) V(S); // Grupa opuszcza wystawę.
        // Grupa odpoczywa.
    }
}
```

Prawdą jest, że

- A. rozwiązanie ma własność bezpieczeństwa
- B. rozwiązanie ma własność żywotności
- C. istnieje taka liczba grup i takie wykonanie programu, w którym pewna grupa zwiedza wystawę nieskończenie wiele razy

9.7. Niech S będzie semaforem binarnym. Oto rozwiązanie problemu wzajemnego wykluczania dla dowolnej liczby procesów:

```
// Sekcja lokalna.
P(S);
// Sekcja krytyczna.
V(S);
```

Jeśli S

- A. ma wartość początkową 0, to powyższe rozwiązanie ma własność bezpieczeństwa
- B. ma wartość początkową 0, to żaden z procesów nie zostanie zagłodzony
- C. ma wartość początkową 1, to powyższe rozwiązanie ma własność bezpieczeństwa

9.8. Rozważmy rozwiązanie problemu wzajemnego wykluczania $N \geq 3$ procesów przy użyciu semafora S o wartości początkowej równej 1.

```
process P(int id) {
    while (true) {
        /* sekcja lokalna */
        P(S);
        /* sekcja krytyczna */
        V(S);
    }
}
```

Prawdą jest, że

- A. jeśli semafor S jest semaforem słabym, to rozwiązanie jest żywotne
- B. jeśli semafor S jest semaforem Dijkstry i $N \leq 2$, to rozwiązanie jest żywotne
- C. rodzaj semafora S nie ma wpływu na bezpieczeństwo rozwiązania

9.3.

Monitory

Innym mechanizmem synchronizacji od semaforów są **monitory**, czyli moduły programistyczne udostępniające na zewnątrz pewne procedury i funkcje, a ukrywające wszystkie zmienne, stałe i typy.

Monitor działa jak sekcja krytyczna, tj. w danej chwili tylko jeden proces wykonuje funkcje monitora („jest w monitorze”). Jeśli pewien proces wywoła funkcję monitora w czasie, gdy inny proces jest już w monitorze, to zostanie automatycznie wstrzymany. Co ważne, wzajemne wykluczanie odbywa się na poziomie całego monitora, a nie poszczególnych jego funkcji.

Monitor gwarantuje ochronę zmiennych globalnych, które są w nim umieszczone (bo dostęp do nich mają tylko funkcje monitora, a może z nich korzystać jeden proces naraz).

Monitory oferują dodatkowy abstrakcyjny typ danych – **zmienne warunkowe** (oznaczane typem `condition`). Z każdą zmienną warunkową związany jest zbiór procesów oczekujących na niej. Można je traktować jako osobne kolejki procesów czekających na dostęp do monitora, istniejące oprócz tej „ogólnej” (dla procesów, które do monitora chcą wejść po raz pierwszy z zewnątrz). Dostępne są trzy operacje na zmiennych warunkowych:

- `wait(c)` – bezwarunkowe wstrzymanie procesu wykonującego tę operację. Proces opuszcza monitor i oczekuje na zmiennej `c`.
- `signal(c)` – obudzenie jednego procesu oczekującego na zmiennej `c`. Wznowiony proces kontynuuje wykonanie od kolejnej instrukcji po `wait(c)`.
- `empty(c)` – zwraca `true`, gdy żaden proces nie czeka na zmiennej.

■ Semantyka `wait()`

Rozróżniamy dwie różne semantyki operacji `wait()`.

Pierwszą z nich jest **semantyka Hoare'a**: ze zmiennymi warunkowymi związane są **kolejki proste (FIFO)**. Procesy oczekujące na wejście do monitora także czekają na kolejce prostej.

Drugą jest **semantyką języka Mesa**, w której **nie zakłada się budzenia procesów w kolejności ich zawieszania** – wznowiany proces jest wybierany przez moduł szeregujący zgodnie z zaimplementowaną strategią, można jednak założyć **silną uczciwość**.

■ Semantyka `signal()`

W związku z tym, że w monitorze może być co najwyżej jeden proces naraz, pojawia się problem, gdy `signal(c)` nie jest ostatnią operacją wykonywaną przez proces w monitorze i na zmiennej warunkowej `c` ktoś czeka – nie można po prostu obudzić procesu, bo wtedy w monitorze znalazłyby się dwa procesy. Rozważymy dwa najpopularniejsze sposoby rozwiązania tego problemu, związane z poprzednimi dwoma semantykami.

Semantyka Hoare'a: proces sygnalizujący wychodzi z monitora i **ustawia się na początku kolejki** procesów oczekujących na wejście do monitora, a **jego miejsce w monitorze zajmuje obudzony proces**. Procesy budzone mają priorytet przed oczekującymi na wejście do monitora (jest to **warunek natychmiastowego wznowienia**). Dodatkowo:

- jeśli `signal(c)` był ostatnią instrukcją wykonaną w monitorze przez proces lub nikt na zmiennej warunkowej `c` nie czekał, to proces sygnalizujący **wychodzi z monitora bez czekania**;
- gdy obudzony proces wyjdzie z monitora, nie budząc nikogo, to tuż po nim **do monitora wejdzie proces, który go obudził** – procesy wstrzymane na skutek wykonania `signal()` mają pierwszeństwo przed procesami oczekującymi na wejście do monitora.

Semantyka języka Mesa: proces sygnalizujący **wykonuje się dalej** i dopiero po jego wyjściu z monitora wznowiany jest dowolny proces: albo oczekujący na wejście, albo obudzony z `wait()`. Niestety, proces wznowiający działanie po `wait()` nie ma gwarancji, że warunek, na który czekał, jest nadal spełniony.

■ Semantyka klasycznych monitorów

O ile nie jest wprost powiedziane, że jest inaczej, klasyczne monitory przyjmują **semantykę Hoare'a**:

- Procesy oczekują na wejście do monitora w kolejce prostej.

- Z każdą zmienną warunkową jest związana osobna kolejka prosta.
- `signal()` na pustej kolejce nie robi nic.
- `signal()`, który faktycznie budzi jakiś proces, powoduje wstrzymanie procesu sygnalizującego i umieszczenie go na **początku** kolejki procesów oczekujących na wejście do monitora; w jego miejsce do monitora wraca obudzony proces, który wznowia wykonanie od kolejnej instrukcji po `wait()`.
- `signal()` wykonywany jako ostatnia instrukcja w monitorze nie wstrzymuje procesu.

Zestaw zadań

9.9. Zgodnie z semantyką klasycznych monitorów (Hoare'a), proces może zostać wstrzymany

- A. przed rozpoczęciem wykonania funkcji eksportowanej przez monitor
- B. wskutek wykonania operacji `wait()` na zmiennej warunkowej wewnętrz monitora
- C. wskutek wykonania operacji `signal()` na zmiennej warunkowej wewnętrz monitora

9.10. Operacja `signal()` wykonana przez proces *Q* na zmiennej warunkowej *c*, na której czeka dokładnie jeden proces *P*

- A. w semantyce Hoare'a powoduje wyjście procesu *Q* z monitora i wstawienie go do kolejki związanej ze zmienną warunkową *c*
- B. zarówno w semantyce Hoare'a, jak i w semantyce Mesy gwarantuje, że proces *P* zostanie kiedyś obudzony z warunku *c*
- C. w semantyce Mesy powoduje wyjście aktualnie wykonującego się procesu z monitora i ustawnie go na początku kolejki na wejściu do monitora

9.11. Semafor silny

- A. gwarantuje silną uczciwość przy budzeniu uśpionych na nim procesów
- B. jest szczególnym przypadkiem semafora słabego (tzn. każdy semafor silny jest semaforem słabym)
- C. może być zaimplementowany za pomocą monitorów z semantyką Hoare'a

9.4.

Komunikacja synchroniczna i asynchroniczna

Do tej pory zajmowaliśmy się scentralizowanym modelem współbieżności, czyli takim, w którym procesy wykonują się w środowisku ze wspólną pamięcią ze współdzielonymi globalnymi zmiennymi. Istnieje także **model rozproszony**, gdzie taka wspólna pamięć nie istnieje. Synchronizacja odbywa się wtedy przez wymianę komunikatów. Rozróżniamy różne typy jej realizacji:

- I.
 - **Synchroniczna** – mamy z nią do czynienia wtedy, gdy chcąc się ze sobą skomunikować procesy są wstrzymywane do chwili, gdy komunikacja będzie się mogła odbyć. Przykładem komunikacji synchronicznej jest rozmowa telefoniczna. Nadawca przekazuje swój komunikat dopiero wówczas, gdy odbiorca chce go wysłuchać.
 - **Asynchroniczna** – nie wymaga współistnienia komunikujących się procesów w tym samym czasie. Polega na tym, że nadawca wysyła komunikat, nie czekając na nic. Komunikaty są buforowane w jakimś miejscu (odpowiada za to system operacyjny lub mechanizmy obsługi sieci) i stamtąd pobierane przez odbiorcę. Przykładem komunikacji asynchronicznej jest poczta.
- II.
 - **Symetryczna** – procesy znajdują nawzajem swoje identyfikatory.
 - **Asymetryczna** – tylko jeden proces zna identyfikator drugiego.

- III.
- **Bezpośrednia** – bezpośrednie odwołanie się jednego wątku lub procesu do innego, wymaga znamioności identyfikatorów docelowych. Wykorzystuje operacje synchronizacyjne.
 - **Pośrednia** – wymiana informacji za pośrednictwem współdzielonych struktur danych lub mechanizmów komunikacyjnych, takich jak kolejki, bufory, kanały. Nie wymaga bezpośredniej znamioności identyfikatorów docelowych.

■ Realizacja komunikacji synchronicznej w Rendezvous

W systemie **Rendezvous** procesy porozumiewają się ze sobą za pomocą komunikatów, które mogą mieć nazwy (ale nie muszą) i mają zero lub więcej argumentów, np.

- Chcę(5)
- Możesz()
- (x + y, 5)
- 12

Wysyłając komunikat, wskazujemy proces, do którego komunikat jest skierowany. Nadawca jest wstrzymywany do chwili, gdy odbiorca będzie gotowy do odbioru wiadomości. Dla przykładu, „send Bufor .5” wysyła komunikat „5” do procesu o nazwie „Bufor”.

Odbierając komunikat, można (ale nie trzeba) wskazać nadawcę. Odbiorca jest wstrzymywany do chwili, gdy nadawca będzie gotowy do wysłania wiadomości. Parametrami komunikatu są lokacje (zmienné), w których zostaną umieszczone ewentualne argumenty odebranego komunikatu. Przykłady:

- receive Możesz() – odbiera bezargumentowy komunikat „Możesz()” (od dowolnego nadawcy)
- receive x – odbiera komunikat będący pojedynczą daną (np. liczbą, jeśli zmienna x jest typu liczbowego) i zapisuje tę daną w zmiennej x
- receive Q.Dodaj(x, y) – odbiera dwuargumentowy komunikat „Doda j()” od procesu o nazwie „Q” i zapisuje oba argumenty w zmiennych x, y

Instrukcje wysyłania i odbierania komunikatu są do siebie dopasowywane na podstawie adresu nadawcy i odbiorcy, nazwy komunikatu oraz liczby i typów argumentów. Po zakończeniu komunikacji oba procesy kontynuują swoje działanie. Zakładamy uczciwość komunikacji.

■ Realizacja komunikacji asynchronicznej w przestrzeni krotek

Przestrzeń krotek to coś w rodzaju nieskończonego bufora, za pomocą którego porozumiewają się ze sobą procesy. Jest jedna wielka, globalna przestrzeń krotek.

Procesy porozumiewają się ze sobą, umieszczając w przestrzeni krotek komunikaty i pobierając je z niej. Procesy nie muszą wiedzieć nic o sobie nawzajem; co więcej, mogą nie istnieć w tym samym czasie. Nadawca może odebrać komunikat od procesu, który już dawno zakończył swoje działanie.

Implementacja przestrzeni krotek jest realizowana np. za pomocą **notacji Linda**, której najważniejsze operacje to:

- **void tsPut(char const* format, args...)** umieszcza w przestrzeni krotkę opisaną formatem z uwzględnieniem ewentualnych argumentów, analogicznie jak przy C++-owym printf(). Operacja jest nieblokująca – proces umieszcza krotkę w przestrzeni i wykonuje się dalej.

Przykład użycia: `tsPut("KROTKA %d %f %c", 5, 3.14, 'c');`

Operacja tsPut() może umieścić w przestrzeni krotkę o nieokreślonej wartości pewnych składowych z użyciem pytajnika, np. `tsPut("%d ?c", 2);`. Taka krotka ma jednoznacznie określoną postać i dziury na pewnych składowych. W operacjach selektywnego wyboru dziury pasują do każdej wartości, ale w przypadku zapisania dziury na zmienną ma ona nieokreoloną wartość.

- **void tsFetch(char const* format, ...)** usuwa z przestrzeni krotkę opisaną formatem, a wartości jej składowych przypisuje kolejnym argumentom (jak w funkcji scanf()). Jeśli krotki o podanym formacie nie ma w przestrzeni krotek, to operacja tsFetch() wstrzymuje proces. Format konstrujemy jak w operacji wyjścia, ale zamiast znaku % używamy pytajnika.

Przykład użycia:

```
| int x; double y; char z;
| tsFetch("%d %f %c", &x, &y, &z);
```

Nie jest określona kolejność budzenia procesów, ale zakłada się silną uczciwość, to samo z wyjmowaniem pasujących krotek.

Elementem formatu w operacji tsFetch() może być też pole rozpoczynające się od znaku %. W takiej sytuacji wyjmowana z przestrzeni krotka musi mieć w danym miejscu wskazaną wartość:

```
tsFetch("%d %f", 25, &y);
```

- **void tsRead(char const* format, ...)** działa jak tsFetch(), ale pozostawia odczytaną krotkę w przestrzeni krotek.

Przykład wykorzystania notacji Linda poznamy w następnym rozdziale (przy problemie producentów i konsumentów).

Zestaw zadań

- 9.12.** Proces może wysłać wiadomość do samego siebie w trybie

- A. bezpośredniej komunikacji asynchronicznej
- B. bezpośredniej komunikacji synchronicznej
- C. komunikacji z użyciem przestrzeni krotek

9.5.

Klasyczne problemy współbieżności

W tym rozdziale omówimy kilka klasycznych problemów synchronizacyjnych, których schematy rozwiązania mają często zastosowanie przy bardziej skomplikowanych algorytmach.

■ Problem wzajemnego wykluczania

Pierwszym klasycznym problemem synchronizacyjnym, najczęściej pojawiającym się w praktyce, jest **problem wzajemnego wykluczania**. Przypuśćmy, że mamy procesy, z których każdy wykonuje następujący program:

```
process P() {
    while (true) {
        własneSprawy();
        protokolWstepny();
        sekcjaKrytyczna();
        protokolKoncowy();
    }
}
```

Procedura `własneSprawy()` to fragment kodu, który nie wymaga żadnych działań synchronizacyjnych. Proces może wykonywać własne sprawy **dowolnie długo** (nawet nieskończenie długo), może też ulec tam awarii. `sekcjaKrytyczna()` to fragment programu, który może być jednocześnie wykonywany przez co najwyżej jeden proces. Zakładamy, że **każdy proces wchodzący do sekcji krytycznej w skończonym czasie z niej wyjdzie**. Oznacza to w szczególności, że podczas pobytu w sekcji krytycznej nie wystąpi błąd ani proces się nie zapętli.

Zadanie polega na napisaniu protokołu wstępnego i protokołu końcowego, tak aby:

- W sekcji krytycznej przebywał co najwyżej jeden proces jednocześnie (**bezpieczeństwo**).
- Każdy proces, który chce wykonać sekcję krytyczną, w skończonym czasie do niej wszedł (**żywotność**).

Protokoły wstępny i końcowy konstruuje się, korzystając z dostępnych mechanizmów synchronizacyjnych, np. **algorytm Petersona**. Poniżej przedstawiono rozwiązanie problemu wzajemnego wykluczania z zastosowaniem właśnie tego algorytmu:

```
boolean chce[2] = {false, false};  
int ktoCzeka = 0;  
  
process P0() {  
    while (true) {  
        // Własne sprawy  
        chce[0] = true;  
        ktoCzeka = 0;  
        while (ktoCzeka == 0 && chce[1]) {}  
        // Sekcja krytyczna  
        chce[0] = false;  
    }  
}  
  
process P1() {  
    while (true) {  
        // Własne sprawy  
        chce[1] = true;  
        ktoCzeka = 1;  
        while (ktoCzeka == 1 && chce[0]) {}  
        // Sekcja krytyczna  
        chce[1] = false;  
    }  
}
```

■ Producenci i konsumenci

W systemie działa $P > 0$ procesów, które produkują pewne dane oraz $K > 0$ procesów, które odbierają dane od producentów. Między producentami a konsumentami może znajdować się bufor o pojemności B , którego zadaniem jest równoważenie chwilowych różnic w czasie działania procesów. Procesy produkujące dane będziemy nazywać **producentami**, a procesy odbierające dane – **konsumentami**. Zadanie polega na synchronizacji pracy producentów i konsumentów, tak aby:

- Konsument oczekiwany na pobranie danych w sytuacji, gdy bufor jest pusty (**bezpieczeństwo**).
- Producent, umieszczając dane w buforze, nie nadpisywał danych już zapisanych, a jeszcze nie odebranych przez żadnego konsumenta. Wymaga to wstrzymania producenta w sytuacji, gdy w buforze nie ma już wolnych miejsc.
- Jeśli wielu konsumentów oczekuje, aż w buforze pojawią się jakieś dane oraz ciągle są produkowane nowe dane, to każdy oczekujący konsument w końcu coś z bufora pobierze. Nie zdarzy się tak, że pewien konsument czeka w nieskończoność na pobranie danych, jeśli tylko ciągle napływają one do bufora (**żywotność**).
- Jeśli wielu producentów oczekuje, aż w buforze będzie wolne miejsce, a konsumenti ciągle coś z bufora pobierają, to każdy oczekujący producent będzie mógł coś do bufora włożyć. Nie zdarzy się tak, że pewien producent czeka w nieskończoność, jeśli tylko ciągle z bufora coś jest pobierane (**żywotność**).

Problem producentów i konsumentów jest abstrakcją wielu sytuacji występujących w systemach komputerowych, na przykład zapis danych do bufora klawiatury przez sterownik klawiatury i ich odczyt przez system operacyjny.

Przykładowe rozwiążanie problemu z użyciem przestrzeni krotek i notacji Linda. Początkowo w przestrzeni: lprod 1, lkons 1 oraz M krotek miejsce.

```
// i = 1, ..., P
process Producent(int i) {
    int p, ile;
    while (true) {
        produkuj(&p);
        tsFetch("miejsce");
        tsFetch("lprod ?d", &ile);
        tsPut("%d %d", ile, p);
        tsPut("lprod %d", ile + 1);
    }
}

// j = 1, ..., K
process Konsument(int j) {
    int p, ile;
    while (true) {
        tsFetch("lkons ?d", &ile);
        tsFetch("%d ?d", ile, &p);
        tsPut("miejsce");
        tsPut("lkons ?d", ile + 1);
        konsumuj(p);
    }
}
```

■ Czytelnicy i pisarze

W systemie działa $C > 0$ procesów, które odczytują pewne dane oraz $P > 0$ procesów, które zapisują te dane. Procesy zapisujące będziemy nazywać **pisarzami**, a odczytujące – **czytelnikami**. Moment, w którym procesy mają dostęp do danych, będziemy nazywać *pobytom w czytelni*.

Zauważmy, że jednocześnie wiele procesów może odczytywać dane. Jeśli jednak ktoś chce te dane zmodyfikować, to rozsądnie jest zablokować dostęp do tych danych dla wszystkich innych procesów na czas zapisu. Zapobiegnie to odczytaniu niespójnych informacji (na przykład danych tylko częściowo zmodyfikowanych).

Należy tak napisać protokoły wstępne i końcowe poszczególnych procesów, aby:

- Wielu czytelników miało jednocześnie dostęp do czytelni.
- Jeśli w czytelni przebywa pisarz, to nikt inny w tym czasie nie pisze ani nie czyta (**bezpieczeństwo**).
- Każdy czytelnik, który chce odczytać dane, w końcu je odczyta (**żywotność**).
- Każdy pisarz, który chce zmodyfikować dane, w końcu je zapisze (**żywotność**).

Poniżej przedstawiono przykładowe rozwiążanie problemu (z wykorzystaniem monitorów):

```
// i = 0, ..., C
process Czytelnik(int i) {
    while (true) {
        // Własne sprawy
        Czytelnia.chceCzytac();
        // Czytanie
        Czytelnia.koniecCzytania();
    }
}
```

```

}

// j = 0, ..., P
process Pisarz(int j) {
    while (true) {
        // Własne sprawy
        Czytelnia.chcePisac();
        // Pisanie
        Czytelnia.koniecPisania();
    }
}

monitor Czytelnia {
    int czytelnicy = 0;
    boolean pisanie = false;
    condition czekamNaPisanie, czekamNaCzytanie;

    chceCzytac() {
        if (pisanie || !empty(czekamNaPisanie))
            wait(czekamNaCzytanie);
        czytelnicy++;
        signal(czekamNaCzytanie);
    }

    chcePisac() {
        if (czytelnicy != 0 || pisanie)
            wait(czekamNaPisanie);
        pisanie = true;
    }

    koniecCzytania() {
        czytelnicy--;
        if (czytelnicy == 0)
            signal(czekamNaPisanie);
    }

    koniecPisania() {
        pisanie = false;
        if (!empty(czekamNaCzytanie))
            signal(czekamNaCzytanie);
        else
            signal(czekamNaPisanie);
    }
}

```

■ Pięciu filozofów

Pięciu filozofów siedzi przy okrągłym stole, przed każdym stoi talerz i między talerzami leżą widelce. Każdy filozof myśli, a gdy zgłodnieje, sięga po widelce znajdujące się po jego prawej i lewej stronie, po czym rozpoznaje posiłek. Gdy już się naje, odkłada widelce i ponownie oddaje się myśleniu. Schemat działania filozofa jest więc następujący:

```

// i = 0, ..., 4
process Filozof(int i) {
    while (true) {
        mysli();
        protokolWstepny();
        je();
    }
}

```

```

        protokolKoncowy();
    }
}

```

Należy tak napisać protokoły wstępny i końcowy, aby:

- Jednocześnie tym samym widelcem jadł co najwyżej jeden filozof (**bezpieczeństwo**).
- Każdy filozof jadł zawsze dwoma (i zawsze tymi, które leżą przy jego talerzu) widelcami (**bezpieczeństwo**).
- Żaden filozof nie umarł z głodu (**żywotność**).

Chcemy ponadto, aby każdy filozof działał w ten sam sposób.

Pierwszy pomysł rozwiązania polega na tym, aby każdy filozof, gdy zgłodnieje, sprawdzał, czy widelec po jego lewej stronie jest wolny. Jeśli tak, to filozof powinien podnieść ten widelec i oczekwać na dostępność drugiego. Nietrudno zauważyc, że w ten sposób może dojść do zakleszczenia – dzieje się tak, gdy filozofowie jednocześnie zgłodnieją i każdy z nich w tej samej chwili podniesie lewy widelec. Każdy oczekuje teraz na prawy widelec, ale nigdy się na niego nie doczeka.

A co stałoby się, jeśli każdy filozof podnosiłby naraz oba widelece, jeśli oba są wolne? Wtedy do zakleszczenia nie dochodzi. Tym razem jednak doszłoby do zagłodzenia filozofa: jeśli dwaj sąsiedzi pewnego filozofa „zmówią się” przeciw niemu, to mogą nie dopuścić do sytuacji, w której obydwa widelece będą dostępne, na zmianę blokując mu dostęp raz do prawego, raz do lewego wideleca.

Poprawne rozwiązanie tego problemu przedstawiono poniżej (z wykorzystaniem semaforów):

```

semaphore w[5] = {1, 1, 1, 1, 1}; // Widelece
semaphore lokaj = 4;

// i = 0, ..., 4
process Filozof(int i) {
    while (true) {
        // Myśli
        P(lokaj);
        P(w[i]);
        P(w[(i + 1) % 5]);
        // Je
        V(w[(i + 1) % 5]);
        V(w[i]);
        V(lokaj);
    }
}

```

Zestaw zadań

9.13. Rozważmy następujące (być może niepoprawne) rozwiązanie problemu ucztujących filozofów:

```

int N = 5; // liczba filozofów
binary_semaphore S[N] = {1, 1, 1, 1, 1}; // ochrona wideleci
process filozof(int i) {
    while (true) {
        // myśli
        P(S[i]); // bierze lewy widelec
        P(S[(i + 1) % N]); // bierze prawy widelec
        // je
        V(S[i]); // odkłada lewy widelec
        V(S[(i + 1) % N]); // odkłada prawy widelec
    }
}

```

Prawdą jest, że

- A. program ma własność bezpieczeństwa
- B. program ma własność żywotności
- C. istnieje przebieg programu, w którym każdy z filozofów je nieskończenie wiele razy

9.14. Procesy P_1 i P_2 są synchronizowane algorytmem Petersona. Wynika z tego, że

- A. P_1 i P_2 wchodzą do sekcji krytycznej na zmianę
- B. P_1 wejdzie kiedyś do sekcji krytycznej
- C. w dowolnym momencie w sekcji krytycznej przebywa co najwyżej jeden proces

9.6. Systemy rozproszone

Przypis redakcji

Ten rozdział wymaga gruntownego przeredagowania – wypadałoby napisać coś sensownego zamiast zdawkowych informacji na temat algorytmów.

W szczególności brak tu szczegółowych informacji na temat algorytmu synchronizacji zegarów logicznych Lamporta – a jak widać po zestawie zadań, pytania na ten temat się pojawiają i są nietrywialne.

Algorytmy rozproszone to techniki i procedury, które są stosowane do zarządzania równoczesnym działaniem procesów lub wątków w rozproszonym środowisku.

■ Wzajemne wykluczanie

Algorytmy wzajemnego wykluczania służą do zapewnienia, że dwa lub więcej procesów nie wykonują jednocześnie krytycznej sekcji kodu. Istnieje wiele algorytmów wzajemnego wykluczania, takich jak [algorytm Ricarta-Agrawali](#) korzystający z komunikacji asynchronicznej, czy algorytm Nielsena-Misuno opierający się o metodę żetonową. Te algorytmy umożliwiają procesom uzyskanie dostępu do wspólnego zasobu w sposób chroniony przed równoczesnym dostępem przez inne procesy.

■ Synchronizacja zegarów logicznych

W rozproszonym środowisku, gdzie procesy działają na różnych węzłach, trudno jest utrzymać dokładną synchronizację zegarów. Algorytmy synchronizacji zegarów logicznych, takie jak [algorytm Lamporta](#), umożliwiają procesom osiągnięcie względnej synchronizacji czasu. Dzięki temu można ustalić porządek wykonywania zdarzeń na różnych węzłach, niezależnie od różnic w czasie systemowym.

Algorytm synchronizacji zegarów logicznych Lamporta

Działanie:

Algorytm opiera się na przesyłaniu wiadomości między procesami oraz wykorzystaniu wartości zegarów logicznych. Każdy proces utrzymuje lokalny zegar logiczny, który jest zwiększany w momencie wystąpienia lokalnych zdarzeń. Po otrzymaniu wiadomości, proces aktualizuje wartość swojego zegara, biorąc pod uwagę zarówno swoje lokalne zdarzenia, jak i zdarzenia związane z otrzymanymi wiadomościami. Proces zwiększa wartość zegara do maksimum spośród swojej wartości zegara i wartości otrzymanej z wiadomości, a następnie dodaje 1.

Wykorzytanie:

- Algorytmy wzajemnego wykluczania w systemach rozproszonych (np. w algorytmie Ricarta-Agrawali)
- Rozproszone systemy plików

Uzgadnianie

Algorytmy uzgadniania są stosowane w celu osiągnięcia konsensusu między wieloma procesami w rozproszonym środowisku, w którym niektóre węzły mogą być wadliwe. Często wymagane jest podjęcie decyzji wspólnie przez wszystkie procesy, na przykład w przypadku ustalania globalnego stanu systemu lub wyboru lidera. Algorytmy takie jak algorytm króla czy algorytm Paxos umożliwiają procesom osiągnięcie jednomyślnego porozumienia.

Zestaw zadań

- 9.15.** W systemie rozproszonym działa N procesów, które mogą komunikować się parami, każdy z każdym. Jeden z procesów chce rozgłosić pewną informację wszystkim pozostałym procesom. Informacja jest na tyle krótka, że mieści się w jednym komunikacie. Przyjmijmy, że przesłanie pojedynczego komunikatu pomiędzy dwoma procesami zajmuje jedną jednostkę czasu. Istnieje schemat komunikacji między N procesami, w którym ogłoszenie informacji do wszystkich procesów
- A. trwa $O(N)$ jednostek czasu
 B. trwa $O(\log N)$ jednostek czasu
 C. wymaga rozesłania łącznie $O(\log N)$ komunikatów
- 9.16.** Algorytm synchronizacji zegarów logicznych Lamporta
- A. służy do synchronizacji zegara systemowego z siecią przy starcie systemu operacyjnego
 B. poprawia zegar logiczny procesu na podstawie otrzymanych komunikatów
 C. może cofnąć zegar logiczny
- 9.17.** Zegary logiczne Lamporta
- A. służą do walidacji pamięci podręcznych w systemach rozproszonych
 B. są wykorzystywane w algorytmie Ricarta-Agrawali wzajemnego wykluczania w systemach rozproszonych
 C. służą do synchronizacji czasu w sieciowym systemie plików NFS

9.7.

Rozwiązania

Rozwiązania

- 9.1.** Rozważmy następujący program wykonywany przez trzy procesy:

```
int x = 0;  
  
process P(int id) { // id = 0, 1, 2  
    int y;  
    for (i = 0; i < 5; i++) {  
        y = x;  
        y = y + 1;
```

```

        x = y;
    }
}

```

Istnieje taki przeplot, że wartość zmiennej x po zakończeniu wszystkich trzech procesów jest równa

- TAK A. 15
- TAK B. 3
- TAK C. 2

Zauważmy, że wartość $x = 15$ będzie uzyskana, gdy wszystkie procesy wykonają się w całości jeden po drugim. W związku z tym odpowiedź **A.** jest prawdziwa.

Wartość $x = 2$ będzie uzyskana przy następującym przeplocie:

- P_0 zapisuje $y = 0$ i zawiesza się.
- P_1 wykonuje się w całości, zostawiając $x = 5$.
- P_2 wykonuje 4 obroty pętli. Wtedy $x = 9$ i zanim P_2 rozpocznie piąty obrót pętli, zawiesza się.
- P_0 dokańcza pierwszy obrót pętli, zapisując $x = 1$ i zawieszając się.
- P_2 zapisuje $y = 1$ i zawiesza się.
- P_0 wykonuje wszystkie pozostałe obroty pętli i zapisując $x = 5$, kończy przebieg.
- P_2 kończy ostatni obrót pętli, zapisując $x = 2$.

Widzimy więc, że odpowiedź **C.** również jest prawdziwa.

Nietrudno zmodyfikować powyższy przeplot, aby uzyskać wartość $x = 3$: wystarczy na przykład, aby P_2 wykonał (w odpowiednim momencie) 3 zamiast 4 obrotów pętli, dzięki czemu w ostatnim kroku będzie miał do zrobienia 2 zamiast 1 obrotu pętli i zapisze $x = 3$. Odpowiedź **B.** jest więc prawdziwa.

9.2. Dany jest następujący program wzajemnego wykluczania dla dwóch procesów:

```

int jest[2] = {0, 0};

process P(int id) { /* id = 0 lub id = 1 */
    while (true) {
        /* sekcja lokalna */
        while (jest[1 - id]); // (*)
        jest[id] = 1; // (*)
        /* sekcja krytyczna */
        jest[id] = 0;
    }
}

```

Taki program

- NIE A. ma własność bezpieczeństwa
- NIE B. ma własność żywotności
- NIE C. będzie miał własność żywotności, kiedy zamienimy kolejność wierszy oznaczonych gwiazdką

9.3. Własność żywotności rozwiązania problemu wzajemnego wykluczania oznacza, że

- TAK A. każdy proces kiedyś wejdzie do sekcji krytycznej
- NIE B. w sekcji krytycznej zawsze znajdzie się co najwyżej jeden proces
- NIE C. każdy proces wchodzi do sekcji krytycznej bez czekania

- A. Wynika to bezpośrednio z definicji żywotności.

- B. Jest to własność bezpieczeństwa, a nie własność żywotności.
- C. Definicja żywotności nie mówi o niczym takim. Dodatkowo, gdyby każdy proces wchodził do sekcji krytycznej bez czekania, to nie zostałyby zachowana własność bezpieczeństwa.
- A. Początkowo tablica `jest[]` ma oba pola wyzerowane. Może wystąpić przeplot, w którym pierwszy proces przejdzie przez pętlę `while (jest[1 - id])`, po czym drugi proces również przejdzie przez pętlę (zanim pierwszy proces wykona `jest[id] = 1`), przez co oba procesy trafią jednocześnie do sekcji krytycznej.
- B. Jeden z procesów może w łatwy sposób zagłodzić drugiego: wystarczy, że ten drugi przez cały czas trwania będzie otrzymywał czas procesora wyłącznie w przypadku, w którym nie będzie mógł przejść przez pętlę `while (jest[1 - id])`. Tym sposobem nie wejdzie on nigdy do sekcji krytycznej.
- C. Po zamianie wierszy oznaczonych gwiazdką można doprowadzić do zakleszczenia. Wystarczy, że oba procesy najpierw wykonają instrukcję `jest[id] = 1`, a dopiero potem razem wejdą do pętli `while (jest[1 - id])`, w której utkną w nieskończoność.

9.4. Dany jest kod dla dwóch procesów:

```
boolean czeka1, czeka2;

process P1() {
    czeka1 = false;
    while (true) {
        czeka1 = true;
        while (czeka2) {}
        // Sekcja krytyczna.
        czeka1 = false;
    }
}

process P2() {
    czeka2 = false;
    while (true) {
        czeka2 = true;
        while (czeka1) {}
        // Sekcja krytyczna.
        czeka2 = false;
    }
}
```

Taki program

TAK A. wykorzystuje aktywne czekanie

NIE B. zapewnia żywotność

TAK C. zapewnia bezpieczeństwo

- A. Tak, wprost z definicji aktywnego oczekiwania.
- B. Rozwiążanie można zakleszczyć w bardzo prosty sposób: oba procesy najpierw ustawiają swoje zmienne czeka na `true`, a następnie oba procesy wchodzą do pętli `while (czeka)`, w której utkną w nieskończoność, tworząc deadlock.
- C. Gdyby każdy z procesów znalazła się w jednym momencie w sekcji krytycznej, to obie zmienne czeka musiałyby mieć wartość `false` (tak aby żaden z procesów nie zatrzymał się w pętli przed sekcją krytyczną). Nietrudno zauważyć, że taka sytuacja jest niemożliwa.

9.5. Synchronizacja za pomocą blokad wirujących (ang. *spinlock*)

TAK A. pozwala uniknąć przełączania kontekstu niezbędnego w przypadku konieczności wstrzymywania procesu

NIE B. zapewnia żywotność

TAK C. w systemach z pamięcią podręczną wykorzystuje lokalne aktywne oczekiwanie (sprawdzana jest wartość zapisana w pamięci podręcznej)

A. Tak, jest to jedna z nielicznych zalet używania spinlocka.

B. Nie, spinlock może doprowadzić do zagłodzenia.

C. Tak, jest to cecha szczególna dla systemów z pamięcią podręczną.

9.6. W sali wystawowej może jednocześnie przebywać co najwyżej K osób. Wystawę zwiedzają grupy reprezentowane przez procesy Grupa. Parametrem procesu Grupa jest liczba osób w grupie, będąca liczbą dodatnią mniejszą bądź równą K . Jeśli grupa w całości nie mieści się w sali, musi poczekać. Rozważmy następujące rozwiązanie tego zadania korzystające z semafora silnego S o wartości początkowej K .

```
process Grupa (int liczność) {
    while (true) {
        for (int i = 0; i < liczność; ++i) P(S); // Grupa czeka na miejsce.
        // Grupa zwiedza wystawę.
        for (int i = 0; i < liczność; ++i) V(S); // Grupa opuszcza wystawę.
        // Grupa odpoczywa.
    }
}
```

Prawdą jest, że

TAK A. rozwiązanie ma własność bezpieczeństwa

NIE B. rozwiązanie ma własność żywotności

TAK C. istnieje taka liczba grup i takie wykonanie programu, w którym pewna grupa zwiedza wystawę nieskończenie wiele razy

A. Ponieważ sekcja krytyczna (zwiedzanie wystawy) jest chroniona semaforem, rozwiązanie ma własność bezpieczeństwa.

B. W łatwy sposób może dojść do zakleszczenia: przypuśćmy, że mamy 2 grupy po 2 osoby, a wystawę mogą jednocześnie zwiedzać dwie osoby. Wystarczy, że na semaforze zawiesi się po jednej osobie z każdej z obu grup – wtedy nikt nie zwiedza wystawy, a semafor jest zablokowany i nikt go nie zwolni.

C. Tak: wystarczy przyjąć, że istnieje tylko jedna grupa.

9.7. Niech S będzie semaforem binarnym. Oto rozwiązanie problemu wzajemnego wykluczania dla dowolnej liczby procesów:

```
// Sekcja lokalna.
P(S);
// Sekcja krytyczna.
V(S);
```

Jeśli S

TAK A. ma wartość początkową 0, to powyższe rozwiązanie ma własność bezpieczeństwa

NIE B. ma wartość początkową 0, to żaden z procesów nie zostanie zagłodzony

TAK C. ma wartość początkową 1, to powyższe rozwiązanie ma własność bezpieczeństwa

A. Wtedy po prostu każdy zawiesi się na semaforze i nikt nigdy nie wejdzie do sekcji krytycznej – rozwiązanie jest więc bezpieczne.

- B. Każdy zostanie zagłodzony, bo każdy zawiesi się na semaforze.
- C. Sekcja krytyczna jest chroniona semaforem binarnym, więc nie ma możliwości, żeby dwa procesy naraz weszły do sekcji krytycznej – rozwiązanie jest bezpieczne.

9.8. Rozważmy rozwiązanie problemu wzajemnego wykluczania $N \geq 3$ procesów przy użyciu semafora S o wartości początkowej równej 1.

```
process P(int id) {
    while (true) {
        /* sekcja lokalna */
        P(S);
        /* sekcja krytyczna */
        V(S);
    }
}
```

Prawdą jest, że

- NIE** A. jeśli semafor S jest semaforem słabym, to rozwiązanie jest żywotne
 - TAK** B. jeśli semafor S jest semaforem Dijkstry i $N \leq 2$, to rozwiązanie jest żywotne
 - TAK** C. rodzaj semafora S nie ma wpływu na bezpieczeństwo rozwiązania
- A. Rozważmy przypadek, gdzie mamy $N = 3$ procesy oraz przeplot, w którym pierwszy z nich wchodzi do sekcji krytycznej, a pozostałe czekają na semaforze. Następnie, po wyjściu z sekcji krytycznej proces 1 obudzi proces 2, po czym zawiesi się na S . Proces 2 wychodząc z sekcji obudzi proces 1 i się zawiesi – i tak w kółko. Proces 3 zostaje w ten sposób zagłodzony, więc rozwiązanie nie jest żywotne.
 - B. Dla $N \leq 2$ rozwiązanie jest żywotne (nie ma kto być zagłodzony, ponieważ maksymalnie jeden proces będzie czekał na semaforze).
 - C. Niezależnie od tego, jaki semafor to był, nie ma opcji, żeby dwa procesy przeszły przez semafor o wartości początkowej 1.

9.9. Zgodnie z semantyką klasycznych monitorów (Hoare'a), proces może zostać wstrzymany

- TAK** A. przed rozpoczęciem wykonania funkcji eksportowanej przez monitor
 - TAK** B. wskutek wykonania operacji `wait()` na zmiennej warunkowej wewnętrz monitora
 - TAK** C. wskutek wykonania operacji `signal()` na zmiennej warunkowej wewnętrz monitora
- A. Jeżeli jakiś proces jest w monitorze, to nowo przybyły zawiesi się na kolejce do wejścia do monitora.
 - B. Operacja `wait()` zawsze wstrzymuje proces.
 - C. Jeśli proces obudzi inny proces operacją `signal()`, to sam zostaje wstrzymany (idzie na początek kolejki wejścia do monitora).

9.10. Operacja `signal()` wykonana przez proces Q na zmiennej warunkowej c , na której czeka dokładnie jeden proces P

- NIE** A. w semantyce Hoare'a powoduje wyjście procesu Q z monitora i wstawienie go do kolejki związanej ze zmienną warunkową c
 - TAK** B. zarówno w semantyce Hoare'a, jak i w semantyce Mesy gwarantuje, że proces P zostanie kiedyś obudzony z warunku c
 - NIE** C. w semantyce Mesy powoduje wyjście aktualnie wykonującego się procesu z monitora i ustawnie go na początku kolejki na wejściu do monitora
- A. Zgodnie z semantyką Hoare'a, proces sygnalizujący, który faktycznie budzi inny proces ze zmiennej warunkowej, zostaje umieszczony na początku kolejki procesów oczekujących na wejście do monitora (a nie: związanej ze zmienną warunkową).

- B. W semantyce Hoare'a proces P zostanie obudzony natychmiast, a semantyka Mesy gwarantuje silną uczciwość kolejek.
- C. Jest to charakterystyczne zachowanie dla semantyki Hoare'a, a nie Mesy.

9.11. Semafor silny

- NIE** A. gwarantuje silną uczciwość przy budzeniu uśpionych na nim procesów
- TAK** B. jest szczególnym przypadkiem semafora słabego (tzn. każdy semafor silny jest semaforem słabym)
- TAK** C. może być zaimplementowany za pomocą monitorów z semantyką Hoare'a
- A. Zauważmy, że kolejka w silnym semaforze może być priorytetowa i wtedy procesy o wyższej randze mogą zagłodzić te o niższej.
- B. Tak, kolejność wynikająca z kolejkowania jest szczególnym przypadkiem losowej kolejności semafora słabego.
- C. Tak: po wejściu do monitora sprawdzana jest „wartość semafora” i jeżeli pozwala na wykonanie akcji, to jest ona wykonywana, a jeżeli nie, to proces zawiesza się na zmiennej warunkowej. Budzenie odbywa się podczas operacji opuszczania semafora przez inny proces.

9.12. Proces może wysłać wiadomość do samego siebie w trybie

- TAK** A. bezpośredniej komunikacji asynchronicznej
- NIE** B. bezpośredniej komunikacji synchronicznej
- TAK** C. komunikacji z użyciem przestrzeni krotek
- A. Tak – proces nie musi czekać na odbiór wiadomości i zna swój identyfikator.
- B. Nie – proces zawiesi się na procesie wysyłania i nigdy nie odbierze wiadomości.
- C. Tak – proces nie musi czekać na odbiór wiadomości.

9.13. Rozważmy następujące (być może niepoprawne) rozwiązanie problemu ucztujących filozofów:

```
int N = 5; // liczba filozofów
binary_semaphore S[N] = {1, 1, 1, 1, 1}; // ochrona widelców
process filozof(int i) {
    while (true) {
        // myśli
        P(S[i]); // bierze lewy widelec
        P(S[(i + 1) % N]); // bierze prawy widelec
        // je
        V(S[i]); // odkłada lewy widelec
        V(S[(i + 1) % N]); // odkłada prawy widelec
    }
}
```

Prawdą jest, że

- TAK** A. program ma własność bezpieczeństwa
- NIE** B. program ma własność żywotności
- TAK** C. istnieje przebieg programu, w którym każdy z filozofów je nieskończonie wiele razy
- A. Nie jest trudno zauważyć, że ani nie uzyskamy sytuacji, w której ktoś je niesiądującym widelcem, ani sytuacji, w której ktoś je bez użycia dwóch widelców.
- B. Łatwo o zakleszczenie: każdy z filozofów jednocześnie pobiera pierwszy widelec, a potem w nieskończoność oczekuje na drugi.

C. Tak, wystarczy nie dopuścić do zagłodzenia, czyli sytuacji opisanej w podpunkcie B.

9.14. Procesy P_1 i P_2 są synchronizowane algorytmem Petersona. Wynika z tego, że

NIE A. P_1 i P_2 wchodzą do sekcji krytycznej na zmianę

NIE B. P_1 wejdzie kiedyś do sekcji krytycznej

TAK C. w dowolnym momencie w sekcji krytycznej przebywa co najwyżej jeden proces

A. Może się zdarzyć, że proces P_1 będzie wykonywał dłucho sekcję lokalną, przez co P_2 zdąży w tym czasie wejść do sekcji krytycznej kilka razy.

B. P_1 może się zapętlić albo ulec awarii podczas wykonywania sekcji lokalnej i nigdy nie wejść do sekcji krytycznej.

C. Tak, ponieważ algorytm Petersona zapewnia własność bezpieczeństwa.

9.15. W systemie rozproszonym działa N procesów, które mogą komunikować się parami, każdy z każdym. Jeden z procesów chce rozgłosić pewną informację wszystkim pozostałym procesom. Informacja jest na tyle krótka, że mieści się w jednym komunikacie. Przyjmijmy, że przesłanie pojedynczego komunikatu pomiędzy dwoma procesami zajmuje jedną jednostkę czasu. Istnieje schemat komunikacji między N procesami, w którym ogłoszenie informacji do wszystkich procesów

TAK A. trwa $O(N)$ jednostek czasu

TAK B. trwa $O(\log N)$ jednostek czasu

NIE C. wymaga rozesłania łącznie $O(\log N)$ komunikatów

A. Taką złożoność ma najprostsze rozwiązanie, tj. każdy proces przekazuje komunikat następnikowi, aż wszyscy otrzymają wiadomość. Zajmie to $O(N)$ jednostek czasu.

B. Aby uzyskać czas logarytmiczny, możemy ułożyć procesy w strukturę drzewa binarnego. Rozsyłanie rozpoczęcie proces będący w wierzchołku. Każdy proces po otrzymaniu komunikatu rozesła go do swoich synów. Przekazanie wiadomości w ten sposób zajmie $O(\log N)$ jednostek czasu (procesy znajdujące się na tym samym poziomie głębokości przesyłają wiadomość jednocześnie, co pozwala zaoszczędzić czas).

C. Ponieważ odbiorcą każdego komunikatu może być tylko jeden proces, nie ma możliwości, aby rozesłanie informacji do wszystkich wymagało rozesłania mniej niż $O(N)$ komunikatów.

9.16. Algorytm synchronizacji zegarów logicznych Lamporta

NIE A. służy do synchronizacji zegara systemowego z siecią przy starcie systemu operacyjnego

TAK B. poprawia zegar logiczny procesu na podstawie otrzymanych komunikatów

NIE C. może cofnąć zegar logiczny

A. Nie, bo jest to tylko zegar logiczny.

B. Tak, w dokładne taki sposób jest zrealizowany algorytm.

C. Nie, zegar logiczny za każdym razem może być tylko zwiększały.

przyp. red.: upewnić się, że rozwiązanie jest poprawne

9.17. Zegary logiczne Lamporta

NIE? A. służą do walidacji pamięci podręcznych w systemach rozproszonych

TAK B. są wykorzystywane w algorytmie Ricarta-Agrawali wzajemnego wykluczania w systemach rozproszonych

TAK? C. służą do synchronizacji czasu w sieciowym systemie plików NFS

przyp. red.: TODO

10

Metody numeryczne

Materiały teoretyczne z metod numerycznych zostały opracowane na podstawie slajdów Piotra Krzyżanowskiego.

Podstawa programowa

1. Algorytmy **rozkładu macierzy** i ich zastosowania.
2. **Uwarunkowanie i numeryczna poprawność.**
3. **Metody iteracyjne** rozwiązywania układów równań liniowych i nieliniowych.
4. **Interpolacja wielomianowa.**
5. **Aproksymacja jednostajna** oraz w przestrzeniach unitarnych.
6. Metody numeryczne **całkowania**.

10.1. Algorytmy rozkładu macierzy

■ Normy

Warto jest orientować się w normach wektorowych

- $\|x\|_1 = \sum_{i=1}^N |x_i|$, $\|x\|_2 = \sqrt{\sum_{i=1}^N |x_i|^2}$, $\|x\|_p = \left(\sum_{i=1}^N |x_i|^p\right)^{1/p}$
- $\|x\|_\infty = \max_i |x_i|$, $\|x\|_\infty \leq \|x\|_1 \leq N \|x\|_\infty$
- $\|x\|_2 \leq \|x\|_1 \leq \sqrt{N} \|x\|_2$, $\|x\|_\infty \leq \|x\|_2 \leq \sqrt{N} \|x\|_\infty$

oraz normach macierzowych

- $\|Ax\| \leq \|A\| \cdot \|x\|$, $\|AB\| \leq \|A\| \cdot \|B\|$, $\|I\| = 1$
- $\|A\|_1 = \max_j \sum_i |a_{ij}|$ (kolumnowa)
- $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ (wierszowa)
- $\|A\|_2 = \max\{\sqrt{\mu} : \mu \text{ to w.wł } A^T A\} = \sqrt{\rho(A^T A)}$ (spektralna)

■ Ortogonalność

Macierz $Q \in \mathbb{R}^{n,n}$ nazywamy **ortogonalną**, jeśli

$$Q^T Q = I,$$

czyli kolumny Q są wzajemnie prostopadłe, a ich norma euklidesowa jest równa 1. **UWAGA!**: na numerkach zazwyczaj ortogonalność to znana z GALu ortonormalność (stąd uwaga o normie euklidesowej).

■ Przydatne typy macierzy

Macierz górnodolnotrójkątna to macierz kwadratowa, która ma elementy niezerowe jedynie na diagonali oraz ponad (poniżej) niej.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ 0 & 0 & a_{3,3} & \dots & a_{3,n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{n,n} \end{bmatrix} \quad \text{jest macierzą górnodolnotrójkątną.}$$

Wyznacznik macierzy trójkątnej równy jest iloczynowi elementów na jej diagonali.

Macierz k-diagonalna to macierz kwadratowa, która ma elementy niezerowe jedynie na diagonali oraz wstęźce wokół niej, która jest szerokości k .

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \dots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & 0 \\ 0 & a_{3,2} & a_{3,3} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & \dots & a_{n,n-1} & a_{n,n} \end{bmatrix} \quad \text{jest macierzą trójdiagonalną.}$$

Macierz Householdera wyznaczana jest przez jeden wektor(!) v . Ma ona wzór

$$H = I - 2 \frac{vv^T}{v^Tv},$$

albo jak ktoś woli

$$H = I - \gamma vv^T, \quad \text{gdzie } \gamma = 2/\|v\|_2^2.$$

Macierz Householdera ma wszystkie własności. Przede wszystkim $H = H^T$, $H^T H = I$ – ortogonalność oraz $H = H^{-1}$. Jest też bardzo efektywna pamięciowo, bo zamiast $O(N^2)$ komórek, możemy pamiętać jedynie wektor rozmiaru $O(N)$.

Macierz Givensa to macierz

$$\begin{bmatrix} 1 & 0 & \dots & 0 & \dots & 0 & 0 \\ \vdots & \ddots & \dots & \dots & \dots & \dots & \vdots \\ 0 & \dots & c & \dots & s & \dots & 0 \\ \vdots & \ddots & \dots & \dots & \dots & \dots & \vdots \\ 0 & \dots & -s & \dots & c & \dots & 0 \\ \vdots & \ddots & \dots & \dots & \dots & \dots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix}$$

gdzie $c = \cos \varphi$ oraz $s = \sin \varphi$. Macierz Givensa wykorzystujemy do zerowania konkretnego elementu w wektorze (w szczególności w kolumnie większej macierzy) kosztem innego. Przemnożenie wektora przez macierz Givensa to czas stały, zatem przemnożenie całej macierzy lewostronnie przez macierz Givensa to czas liniowy – zerujemy 1 element w $O(N)$.

Przykład 1.

Mając wektor $[x_1, \dots, x_n]^T$ i chcąc wyzerować n -ty element, naruszając przy tym $(n-1)$ -szy element, dobrabibyśmy wartości

$$\cos \varphi = \frac{x_{n-1}}{\sqrt{x_{n-1}^2 + x_n^2}}, \quad \sin \varphi = \frac{x_n}{\sqrt{x_{n-1}^2 + x_n^2}},$$

a wartości umieścili na indeksach $\{n-1, n-1\}, \{n-1, n\}, \{n, n-1\}, \{n, n\}$. Tzn. nasz Givens to

$$G = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & \cos \varphi & \sin \varphi \\ & & -\sin \varphi & \cos \varphi \end{bmatrix}.$$

Po przemnożeniu Gx otrzymamy wektor $[x_1, x_2, \dots, x'_{n-1}, 0]^T$. Konkretnie $x'_{n-1} = \sqrt{x_{n-1}^2 + x_n^2}$.

■ Układy równań liniowych i ich łatwe wersje

Numeryk zajmuje się regularnie rozwiązywaniem układów równań liniowych, do czego preferuje korzystanie z macierzy. W ogólności, nie da się znaleźć x takiego, że dla danych A oraz y zachodzi $Ax = y$ w czasie lepszym niż $O(N^3)$. Student politechniki mógłby chcieć po prostu odwrócić macierz A , przemnożyć z lewej i otrzymać $A^{-1}Ax = x = A^{-1}y$. Złamałby przy tym pierwsze przykazanie numeryka: **nie będziesz odwracał macierzy swojej nadaremno**. Czasem specyficzne macierze umożliwiają szybsze znalezienie wektora x .

1. Macierz trójkątna w czasie $O(N^2)$.
2. Macierz k-diagonalna rzadka w czasie $O(kN)$.
3. Macierz ortogonalna w czasie $O(N^2)$, bo $Q^T = Q^{-1}$, zatem rozwiązaniem układu $Qx = b$ jest $x = Q^T b$.
4. Macierz Householdera w czasie $O(4N)$, przy czym ważna jest kolejność mnożenia $Hx = (I - 2\frac{vv^T}{v^Tv})x = x - \frac{2}{\|v\|_2^2}(v^T x)v$.

■ Rozkład LU

Rozkładem LU macierzy A nazywamy takie macierze L – dolnotrójkątna z 1 na diagonali oraz U – górnortrójkątna, że $A = LU$. Klasyczny algorytm rozkładu LU działa na macierzach nieosobliwych (dlaczego? zadanie dla Czytelnika). Wykonujemy go, aby móc efektywniej rozwiązywać układy równań. Wykonując rozkład LU, układ równań liniowych $Ax = y$, gdzie x to wektor niewiadomych, wykonujemy następująco:

1. kosztem $O(N^3)$ znajdujemy macierze L, U takie, że $A = LU$,
2. rozwiązujemy $Lz = y$ w $O(N^2)$,
3. rozwiązujemy $Ux = z$ w $O(N^2)$,

co łącznie kosztuje nas też $O(N^3)$, ale łatwiej się liczy. Można też, mając taki rozkład, błyskawicznie obliczyć wyznacznik. Podzielmy macierze na bloki

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

gdzie bloki $(1, 1)$ są rozmiaru $k \times k$ oraz U_{ii} to macierze trójkątne górne, a L_{ii} macierze trójkątne dolne z jedynkami na diagonali (one dają nam jednoznaczność rozkładu). Algorytm działa następująco:

1. Wyznacz (rekurencyjnie) rozkład $A_{11} = L_{11}U_{11}$
2. $U_{12} = L_{11}^{-1}A_{12}$ (tzn. rozwiąż układ z macierzą trójkątną)
3. $L_{21} = A_{21}U_{11}^{-1}$
4. Aktualizuj $A'_{22} = A_{22} - L_{21}U_{12}$
5. Wyznacz (rekurencyjnie) rozkład $A'_{22} = L_{22}U_{22}$

Można też go zapisać w pseudokodzie, ale nikt tego nie zapamięta.

Wybór elementu głównego (partial pivoting) pozwala na uzyskanie rozkładu LU także w przypadku macierzy osobliwych. Wtedy nie zmienia się złożoność czasowa algorytmu, a rozkład jest postaci $PA = LU$, gdzie P to macierz permutacji uzyskana poprzez kolejne wybory elementu głównego.

Rozkład QR

Zakładamy, że $A \in \mathbb{R}^{m,n}$, $m \geq n$. Istnieją dwa rodzaje rozkładu QR

- **Rozkład wąski**

$$A = \hat{Q}\hat{R}$$

gdzie $\hat{Q} \in \mathbb{R}^{m,n}$ ortogonalna oraz $\hat{R} \in \mathbb{R}^{n,n}$ górnopróbkątna.

- **Rozkład pełny**

$$A = [\hat{Q} \quad \tilde{Q}] \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} = QR$$

gdzie $Q \in \mathbb{R}^{m,m}$ – ortogonalna, natomiast $R \in \mathbb{R}^{m,n}$ górnopróbkątna.

Warto zauważyć, że wąski rozkład jest bardziej efektywny pamięciowo, bo używamy $O(mn)$ pamięci, a w pełnym aż $O(m^2)$.

Koszt czasowy rozkładu QR z wykorzystaniem metody Householdera to około $2mn^2 - \frac{2}{3}n^3$.

LZNK

Problem: niech $A \in \mathbb{R}^{m,n}$, $m \geq n$ oraz $\text{rank}(A) = n$ i niech $b \in \mathbb{R}^m$. Znaleźć $x \in \mathbb{R}^n$ taki, że

$$\|b - Ax\|_2 \rightarrow \min!$$

tzn.

$$\|b - Ax\|_2 \leq \|b - Ay\|_2 \quad \forall y.$$

Do rozwiązywania Liniowego Zadania Najmniejszych Kwadratów wykorzystuje się rozkład QR.

Algorytm: jeśli znamy rozkład QR macierzy A pełnego rzędu

$$A = QR = [\hat{Q} \quad \tilde{Q}] \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}$$

zadanie staje się horrendalnie trywialne:

$$\begin{aligned} \|b - Ax\|_2^2 &= \|b - QRx\|_2^2 = \|Q(Q^T b - Rx)\|_2^2 = \|Q^T b - Rx\|_2^2 = \left\| \begin{bmatrix} \hat{Q}^T \\ Q^T \end{bmatrix} b - \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} x \right\|_2^2 \\ &= \left\| \begin{bmatrix} \hat{Q}^T - \hat{R}x \\ \tilde{Q}^T b \end{bmatrix} \right\|_2^2 = \|\hat{Q}^T b - \hat{R}x\|_2^2 + \|\tilde{Q}^T b\|_2^2 \end{aligned}$$

liczy się złożoność takiego algorytmu. W skrócie wygląda on tak:

- Wyznacz rozkład QR macierzy A w czasie $O(mn^2)$
- Oblicz $\hat{b} = \hat{Q}^T b$ w czasie $O(mn)$
- Rozwiąż $\hat{R}x = \hat{b}$ w czasie $O(n^2)$

Zestaw zadań

10.1. A jest macierzą nieosobliwą $N \times N$. Wynika z tego, że

- A. istnieją macierze L dolnotrójkątna i U górnoprójkątna $N \times N$, takie że $A = LU$
- B. istnieją macierz permutacji Q oraz macierze L dolnotrójkątna i U górnoprójkątna $N \times N$, takie że $AQ = LU$
- C. istnieją macierz permutacji P oraz macierze L dolnotrójkątna i U górnoprójkątna $N \times N$, takie że $PA = LU$

10.2. A jest macierzą nieosobliwą o wymiarach $N \times N$. Wynika z tego, że koszt wyznaczenia macierzy A^{-1}

- A. za pomocą algorytmu LU wynosi $O(N^2)$ operacji arytmetycznych
- B. wynosi $O(N^2)$ operacji arytmetycznych, jeśli A jest górnoprójkątna
- C. wynosi $O(1)$ operacji arytmetycznych, jeśli A jest macierzą Householdera

10.2.

Uwarunkowanie i numeryczna poprawność

Błędy

Mówimy o błędzie bezwzględnym

$$\|\tilde{x} - x\|$$

oraz o względnym (dla $x \neq 0$)

$$\frac{\|\tilde{x} - x\|}{\|x\|}$$

gdzie \tilde{x} to przybliżone rozwiązanie, a x to dokładne rozwiązanie.

Uwarunkowanie układu równań liniowych

Oznaczmy

$$cond(A) = \|A\| \cdot \|A^{-1}\|$$

jeśli $\epsilon cond(A) \leq \frac{1}{2}$, to

$$\frac{\|\tilde{y} - y\|}{\|y\|} \leq 4cond(A) \cdot \epsilon.$$

ZŁA WIADOMOŚĆ: macierze mogą być **PATOLOGICZNIE ŹLE UWARUNKOWANE**, tj. $cond(A) \gg 1$.

Algorytm numerycznie poprawny

Dla każdego $x \in X$ wynik algorytmu A zrealizowanego w fl $fl(A(fl(x)))$ jest dokładnym rozwiązaniem zadania dla danych x zaburzonych na poziomie błędu reprezentacji.

Wniosek algorytm NP (gdzie NP to oczywiście Numeryczna Poprawność) daje wynik, którego błąd można szacować na podstawie własności zadania obliczeniowego:

$$\frac{\|\tilde{y} - y\|}{\|y\|} = \frac{\|P(\tilde{x}) - P(x)\|}{\|P(x)\|} \lesssim cond_{rel}(P, x) \frac{\|\tilde{x} - x\|}{\|x\|} \leq K \cdot cond_{rel}(P, x) \cdot v.$$

Przykład 1.

Zadanie obliczeniowe: $P(x_1, x_2) = x_1 + x_2$, gdzie $x_1, x_2 \in \mathbb{R}$.

Algorytm: $A(x_1, x_2) = x_1 + x_2$.

Czy jest NP?

$$fl(A(fl(x))) = fl(x_1(1+\epsilon_1) + x_2(1+\epsilon_2)) = x_1(1+\epsilon_1)(1+\delta) + x_2(1+\epsilon_2)(1+\delta) = x_1(1+E_1) + x_2(1+E_2) = \tilde{x}_1 + \tilde{x}_2$$

przy czym

$$\|E_i\| \leq \|\epsilon_i\| + \|\delta\| \leq 2 \cdot v.$$

10.3. Metody iteracyjne

Metody iteracyjne to metody rozwiązywania układów równań polegające na tym, że każda kolejna iteracja zbliża nas do faktycznego wyniku.

■ Metody oparte na rozszczepieniu macierzy

Niech $A = M - Z$ przy czym M – nieosobliwa

$$Ax^* = b \iff (M - Z)x^* = b \iff Mx^* = b + Zx^* \iff x^* = M^{-1}(b + Zx^*)$$

Rozwiązanie x^* spełnia więc

$$x^* = M^{-1}Zx^* + M^{-1}b = Bx^* + c$$

Twierdzenie o zbieżności metody rozszczepienia

Ciąg

$$x_{k+1} = Bx_k + c$$

jest zbieżny do x^* dla każdego $x_0 \iff \rho(B) < 1$,
gdzie $\rho(B) = \max\{|\lambda| : \lambda \text{ jest wartością własną } B\}$ to promień spektralny macierzy B .

Właściwości promienia spektralnego

- $\rho(B) \leq \|B^k\|^{\frac{1}{k}}$ w szczególności $\rho(B) \leq \|B\|$
- $\rho(AB) = \rho(BA)$
- $\rho(A^2) = \rho(A)^2$
- Jeśli A, B symetryczne to $\rho(AB) \leq \rho(A)\rho(B)$
- Jeżeli A symetryczna to $\rho(A) = \|A\|_2$

Przykłady metod iteracyjnych: Niech $A = L + D + U, D = \text{diag}(A)$

$$x_{k+1} = x_k + M^{-1}(b - Ax_k)$$

- Metoda Jacobiego: $M = D$
- Metoda Gaussa-Seidela: $M = D + L$
- Metoda SOR: $M = \frac{1}{\omega}D + L$

■ Metoda potęgowa

PROBLEM: Znaleźć (potencjalnie największą) wartości własne/wektory własne macierzy A . Czyli

$$Av = \lambda v, \quad \|v\| = 1$$

W kolejnych iteracjach stosujemy algorytm

$$x_{k+1} = Ax_k, \quad x_{k+1} = \frac{x_{k+1}}{\|x_{k+1}\|}$$

Twierdzenie

Jeśli $x_0^T v_1 \neq 0$ to x_k dąży do v_1 dla $k \rightarrow \infty$

Zestaw zadań

10.3. Mamy daną macierz $A = \begin{bmatrix} 7 & -13 & 16 \\ -13 & 10 & -13 \\ 16 & -13 & 7 \end{bmatrix}$ oraz wektor $x_0 = \begin{bmatrix} 2019 \\ 2020 \\ 2021 \end{bmatrix}$. Niech $y_n = Ax_n$ oraz $x_{n+1} = \frac{y_n}{\|y_n\|}$.

Wtedy

- A. $\lim_{k \rightarrow \infty} \|x_k\| = 1$
- B. $\lim_{k \rightarrow \infty} \frac{x_k^T Ax_k}{x_k^T x_k} = 1$
- C. ciąg $\{x_n\}_{n \geq 0}$ jest zbieżny

10.4.

Interpolacja wielomianowa

Bazy wielomianowe

Wielomian stopnia co najwyżej N : $p(x) = \sum_{i=0}^N a_i x^i$. Wszystkie wielomiany stopnia $\leq N$ tworzą przestrzeń liniową wymiaru $N + 1$ oznaczaną P_N . Ogólnie, jeśli $\varphi_0, \dots, \varphi_N$ są bazą przestrzeni P_N , to każdy $p \in P_N$ daje się zapisać

$$p(x) = \sum_{i=0}^N \alpha_i \varphi_i(x).$$

Najważniejsze bazy wielomianowe:

- „naturalna”: $\varphi_i(x) = x^i$,

- Newtona:

$$\varphi_0(x) = 1, \quad \varphi_i(x) = \prod_{j < i} (x - x_j),$$

gdzie x_0, \dots, x_{N-1} – zadane punkty,

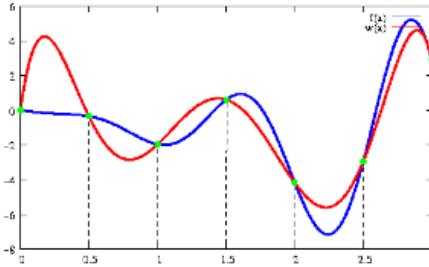
- Lagrange'a:

$$I_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j},$$

- i wiele więcej...

Interpolacja wielomianowa

Problem (interpolacja Lagrange'a): Dla zadanych (parami różnych) węzłów interpolacji x_0, \dots, x_N i wartości $f(x_0), \dots, f(x_N)$ znaleźć wielomian $p \in P_N$ taki, że $p(x_i) = f(x_i)$ dla $i = 0, \dots, N$.



Twierdzenie: Zadanie interpolacji Lagrange'a ma jednoznaczne rozwiązanie.

Wyznaczanie WIL:

- w bazie naturalnej, tzn. $w(x) = \sum_i a_i x^i$ mamy

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^N \\ 1 & x_1 & x_1^2 & \cdots & x_1^N \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^N \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix}$$

Macierz po lewej, zwana macierzą Vandermonde'a, jest gęsta, niesymetryczna, koszt GEPP to $O(N^3)$, a co najgorsze, jest **patologicznie** źle uwarunkowana! Lepiej wziąć inną bazę.

- w bazie Newtona, tzn. $p(x) = \sum_i b_i \cdot (x - x_0) \cdots (x - x_{i-1})$ mamy $b_i = f(x_0, x_1, \dots, x_i)$, $0 \leq i \leq N$, gdzie

$$f(t_0, t_1, \dots, t_s) = \frac{f(t_1, t_2, \dots, t_s) - f(t_0, t_1, \dots, t_{s-1})}{t_s - t_0}$$

to **różnice dzielone**. Algorytm różnic dzielonych wyznacza WIL *in situ* kosztem $O(N^2)$ flopów.

$$\begin{array}{ccccccccc} x_0 & f(x_0) & & & & & & & \\ x_1 & f(x_1) & f(x_0, x_1) & & & & & & \\ x_2 & f(x_2) & f(x_1, x_2) & f(x_0, x_1, x_2) & & & & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & & & & \\ x_N & f(x_N) & f(x_{N-1}, x_N) & f(x_{N-2}, x_{N-1}, x_N) & \cdots & f(x_0, x_1, \dots, x_N) & & & \end{array}$$

Powyższą tabelkę wypełniamy kolejnymi kolumnami.

- w bazie Lagrange'a, tzn.

$$p(x) = \sum_{i=0}^N a_i \underbrace{\prod_{j \neq i} \frac{x - x_j}{x_i - x_j}}_{=I_i(x)}$$

mamy $p(x) = \sum_{i=0}^N f(x_i) \cdot I_i(x)$, a więc koszt „zerowy”! (ale pamiętajmy o koszcie obliczenia wartości p – istnieje algorytm precomputingu o koszcie $O(N^2)$).

Twierdzenie (błąd interpolacji): Niech p będzie WIL dla f w punktach x_0, \dots, x_N . Wtedy

- dla dowolnego $\bar{x} \in \mathbb{R}$

$$f(\bar{x}) - p(\bar{x}) = f(x_0, x_1, \dots, x_N, \bar{x}) \cdot \varphi_{N+1}(\bar{x}),$$

- ponadto, jeśli $f \in C^{(N+1)}[a, b]$, gdzie $[a, b]$ zawiera x_0, \dots, x_N, \bar{x} , to istnieje $\xi = \xi(\bar{x}) \in (a, b)$ takie, że

$$f(\bar{x}) - p(\bar{x}) = \frac{f^{(N+1)}(\xi)}{(N+1)!} \cdot \varphi_{N+1}(\bar{x}).$$

Pamiętamy, że $\varphi_{N+1}(x) = (x - x_0) \cdots (x - x_N)$.

10.5.

Aproksymacja jednostajna

Aproksymacja

PRO8L3M: Niech F będzie przestrzenią liniową i niech $V \subset F$ – podprzestrzeń skończonego wymiaru. Dla danego $f \in F$ znaleźć $v^* \in V$ taki, że

$$\|f - v^*\| \leq \|f - v\| \quad \forall v \in V$$

Mówimy, że v^* jest ENA dla f .

W przypadku aproksymacji jednostajnej chcemy przybliżyć ciągłą funkcję wielomianami:

Niech:

- $F = C[a, b]$ – przestrzeń funkcji ciągłych
- $V = P_N$ – przestrzeń wielomianów stopnia co najwyżej N
- $\|f\|_\infty = \sup_{x \in [a, b]} |f(x)|$

Wówczas dla zadanej funkcji $f \in C[a, b]$ chcemy znaleźć wielomian $p^* \in P_N$ taki, że

$$\|f - p^*\|_\infty \rightarrow \min!$$

Alternans

Zadajmy zbiór $N + 2$ punktów takich, że $a \leq \xi_0 < \xi_1 \dots < \xi_{N+1} \leq b$. Wówczas alternans to funkcja $e = f - p^*$ taka, że

- $|e(\xi_i)| = \|e\|_\infty \quad \text{dla } i = 0, \dots, N + 1$
- $e(\xi_i) = -e(\xi_{i+1}) \quad \text{dla } i = 0, \dots, N$

Twierdzenie o alternansie:

$p^* \in P_N$ jest ENA dla $f \iff$ istnieje alternans dla f i p^*

Węzły Czebyszewa: Weźmy $N + 1$ węzłów interpolacji $a \leq x_0 < x_1 \dots < x_N \leq b$. Wówczas z twierdzenie na postać błędu interpolacji wiemy, że:

$$\|f - p\|_\infty \leq \frac{\|f^{(N+1)}\|_\infty}{(N+1)!} \cdot \|\varphi_{N+1}\|_\infty$$

gdzie $\varphi_{N+1}(x) = (x - x_0) \cdots (x - x_N)$.

PRO8L3M: Jak dobrać węzły interpolacji tak żeby było jak najbardziej optymalnie?

Jeśli $[a, b] = [-1, 1]$ to z pomocą przychodzi wybitny ukraiński matematyk Pafnutij Czebyszew. Okazuje się, że najbardziej optymalne węzły to miejsca zerowe wielomianu Czebyszewa T_{N+1} postaci:

$$x_i = \cos\left(\frac{2i+1}{2N+2}\pi\right)$$

Co więcej, jeśli p jest WIL opartym na węzłach Czebyszewa to istnieje ograniczenie

$$\|f - p\|_\infty \leq \frac{\|f^{(N+1)}\|_\infty}{2^N(N+1)!}$$

■ Metoda bisekcji

Żeby opisać metodę bisekcji posłużę się słynną interakcją z naszego wydziału:

STUDENT: Żaden informatyk nie potrafi bezbłędnie napisać binsearcha od ręki.

PCHW: Ja potrafię. [jebać pchw - przyp. red.]

[Pisze binsearcha z błędem]

Weźmy sobie $f \in C[a, b]$ taką, że $f(a) \cdot f(b) < 0$. Z twierdzenia Darboux wiemy, że f ma miejsce zerowe w (a, b) .

PRO8L3M: Znaleźć miejsce zerowe.

Robimy binsearcha.

Otrzymujemy $x_n = \frac{1}{2^n} |b - a|$

Przy n iteracjach dostajemy ograniczenie

$$|x_n - x^*| \leq \frac{1}{2^n} |b - a|$$

■ Szybkość zbieżności metody iteracyjnej

Ciąg $x_n \rightarrow x^* \in \mathbb{R}^N$ jest:

- zbieżny z rzędem (co najmniej) $p > 1$ jeśli

$$\exists C \geq 0 \quad \|x_{n+1} - x^*\|_\infty \leq C \|x_n - x^*\|_\infty^p$$

Dla $p = 2$ zbieżność kwadratowa, dla $p = 3$ kubiczna

- zbieżny (co najmniej) liniowo jeśli

$$\exists \gamma \in [0, 1) \geq 0 \quad \|x_{n+1} - x^*\|_\infty \leq \gamma \|x_n - x^*\|_\infty$$

- r -zbieżny z rzędem p (odp. liniowo) jeśli

$$\|x_{n+1} - x^*\|_\infty \leq r_n$$

dla pewnego ciągu r_n zbieżnego z rzędem p do 0.

Np. metoda bisekcji jest zbieżna r -liniowo z ilorazem $\frac{1}{2}$.

■ Punkt stały

PRO8L3M: Niech $\Phi : D \rightarrow \mathbb{R}^N$, gdzie $D \subset \mathbb{R}^N$. Znajdź $x^* \in D$ taki, że

$$x^* = \Phi(x^*)$$

czyli znajdź punkt stały Φ .

Jest to oczywiście równoważny pro8l3m do wyznaczania miejsca zerowego funkcji np.

$$f(x) = x - \Phi(x)$$

Crème de la crème polskiej matematyki, czyli Twierdzenie Banacha o kontrakcji:

Niech $\Phi : D \rightarrow D$ będzie kontrakcją

$$\exists L \in [0, 1) \quad \|\Phi(x) - \Phi(y)\| \leq L \|x - y\| \quad \forall x, y \in D$$

Wówczas

- istnieje dokładnie jeden punkt stały Φ

- iteracja Banacha

$$x_{n+1} = \Phi(x_n)$$

jest zbieżna liniowo do x^* z dowolnego $x_0 \in D$

Twierdzenie o maszynce rzędu Założmy, że iteracja $x_{n+1} = \Phi(x_n)$ jest zbieżna do punktu stałego x^* , przy czym $\Phi \in C^p(D)$, $p \geq 1$ oraz

$$\Phi'(x^*) = \dots = \Phi^{p-1}(x^*) = 0$$

Jeśli x_0 jest dostatecznie blisko x^* to

$$\exists C \|x_{n+1} - x^*\| \leq C \|x_n - x^*\|^p$$

Metoda Newtona

Niech $f : D \rightarrow \mathbb{R}^n$ oraz $D \subset \mathbb{R}^N$ otwarty i niepusty. Niech x^* będzie miejscem zerowym f . Jeśli

- f jest różniczkowalna oraz

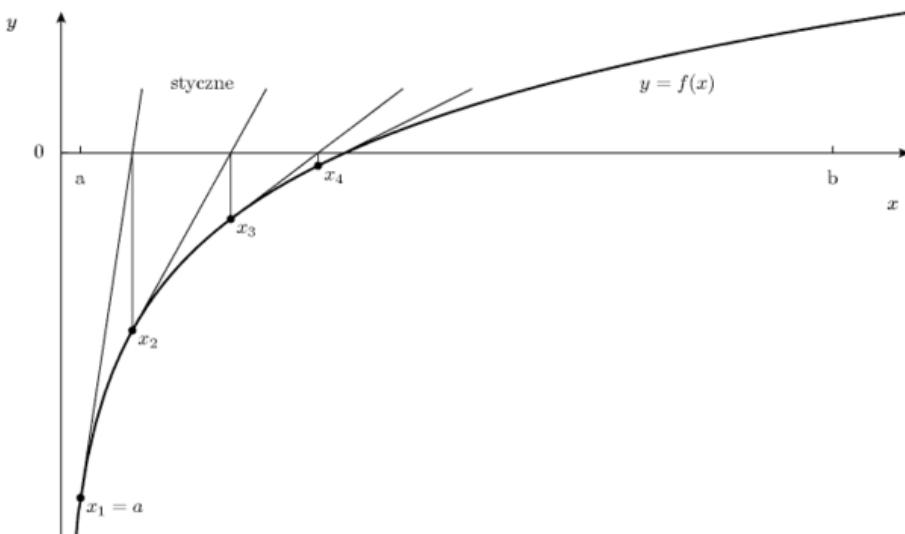
$$\exists L \quad \|f'(x) - f'(y)\| \leq L \|x - y\| \quad \forall x, y \in D$$

- $f'(x^*)$ jest nieosobliwa

Iteracja wygląda następująco:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

to dla x_0 dostatecznie blisko (cokolwiek to znaczy) x^* metoda Newtona jest zbieżna kwadratowo. Jeśli dołożymy warunek, że $f''(x^*) = 0$ to mamy nawet zbieżność sześcienną



Rysunek 10.1: Metoda Newtona, pierwsze 4 iteracje

Zestaw zadań

10.4. Iteracyjna metoda rozwiązywania równania $f(x) = x - \frac{1}{2(1+x^2)}$ gdzie $x_{n+1} = \frac{1}{2(1+x_n^2)}$, jest

A. zbieżna kwadratowo

B. zbieżna dla $x_0 = 0$

C. zbieżna dla $x_0 = 10$

10.5. Szukamy zera funkcji $f(x) = x^2 - a$ na przedziale $[0, a]$, używając metody bisekcji. Prawdą jest, że

A. dla $a = 99$ i dziesięciu iteracji błąd znalezionego rozwiązania szacuje się z góry przez 10^{-4}

B. 10 iteracji wymaga policzenia co najmniej 20 wartości funkcji

C. wzór iteracji dla tej metody to $x_{n+1} = x_n^2 + \frac{a}{2x_n}$

10.6. Dla danego $a \in \mathbb{R}$ szukamy rzeczywistych pierwiastków równania nieliniowego $x^3 - x = a$ metodą Newtona. Wówczas

A. wzór na kolejną iterację metody to $x_{k+1} = \frac{2x_k^3 - a}{3x_k^2 + 1}$

B. dla $a = 1$ i dowolnego $x_0 \geq 1$ metoda Newtona jest zbieżna do jakiegoś pierwiastka równania

C. dla $a = 0$ i $x_0 = 10^{-2}$ metoda Newtona jest zbieżna kwadratowo do zera

10.6. Metody numeryczne całkowania

Kwadratury

Będziemy zajmować się następującym **problemem**: dla $f : [a, b] \rightarrow \mathbb{R}$ przybliżyć wartość

$$I(f) = \int_a^b f(x)\rho(x) dx$$

za pomocą kwadratury $Q(f) = \sum_{i=0}^n A_i f(x_i)$. Chodzi nam o wyznaczenie **współczynników kwadratury** A_i oraz **węzłów kwadratury** $x_i \in [a, b]$ takich, by $Q(f) \approx I(f)$ możliwe dobrze w pewnej klasie funkcji $f \in F$.

Kwadratury interpolacyjne: Idea: funkcję f przybliżyć wielomianem interpolacyjnym p , a całkę z f – całką z p . Niech x_0, \dots, x_n węzły interpolacji Lagrange'a oraz $p(x) = \sum_{i=0}^n f(x_i) \cdot \underbrace{I_i(x)}_{\text{baza Lagrange'a}}$. Wtedy

$$I(f) \approx Q(f) := \int_a^b p(x) dx = \int_a^b \sum_{i=0}^n f(x_i) I_i(x) dx = \sum_{i=0}^n \left(\underbrace{\int_a^b I_i(x) dx}_{=: A_i} \right) \cdot f(x_i)$$

A_i – wyznaczamy raz na zawsze (nie zależą od f) w precomputingu: umiemy obliczać całki z wielomianów. Nie wszystkie kwadratury są interpolacyjne.

Błąd kwadratur interpolacyjnych: Jeśli $f \in C^{n+1}[a, b]$, a Q jest kwadraturą interpolacyjną opartą na węzłach $x_0, \dots, x_n \in [a, b]$, to

$$|I(f) - Q(f)| \leq \frac{\|f^{(n+1)}\|}{(n+1)!} |b-a|^{n+2}.$$

Istnieje parę prostych kwadratur wymienionych poniżej, wraz z ich błędami:

- kwadratura prostokątów:

$$Q^P(f) = (b-a) \cdot f\left(\frac{a+b}{2}\right), \quad I(f) - Q^P(f) = \frac{(b-a)^3}{24} f''(\xi), \text{ dla pewnego } \xi \in [a, b],$$

- kwadratura trapezów:

$$Q^T(f) = \frac{b-a}{2} \cdot (f(a) + f(b)), \quad I(f) - Q^T(f) = -\frac{(b-a)^3}{12} f''(\xi), \text{ dla pewnego } \xi \in [a, b],$$

- kwadratura parabol (Simpsona):

$$Q^S(f) = \frac{b-a}{6} \cdot \left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right), \quad I(f) - Q^S(f) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi), \text{ dla pewnego } \xi \in [a, b].$$

Kwadratura Q dla całki $I(f) = \int_a^b f(x)\rho(x) dx$ jest rzędu (co najmniej) r , jeśli

$$\forall_{p \in P_{r-1}} I(p) = Q(p).$$

Jest **rzędem** r , gdy jest co najmniej rzędem r , ale już nie $r+1$.

Wniosek: Każda kwadratura interpolacyjna oparta na n węzłach jest co najmniej rzędu n .

Twierdzenie: Rząd kwadratury interpolacyjnej opartej na n węzłach nie przekracza $2n$. Realizuje go kwadratura Gaussa, oparta na zerach n -tego wielomianu ortogonalnego w $L_\rho^2(a, b)$.

Zestaw zadań

10.7. Całkę $\int_{-1}^1 f(x) dx$ przybliżamy kwadraturą interpolacyjną $Q(f) = A_1 f(x_1) + A_2 f(x_2)$. Wtedy

- A. dla $A_1 = A_2 = 1$, $x_1 = -1$, $x_2 = 1$ kwadratura Q jest dokładna dla wszystkich wielomianów stopnia 1
- B. można dobrać x_1 i x_2 , tak aby kwadratura Q była rzędu 2 oraz $A_1 = A_2 = 2$
- C. maksymalny rząd kwadratury Q jest równy 4

10.8. Stosując metodę złożonej kwadratury trapezów na odcinku $[a, b]$ dla funkcji f , można obliczyć dokładnie (pomijając błędy zaokrągleń) całkę $\int_a^b f(x) dx$ dla

- A. $f(x) = 3 - x$
- B. $f(x) = 3 - x + x^2$
- C. $f(x) = \sin(x)$

10.7.

Rozwiązańia

Rozwiązańia

10.1. A jest macierzą nieosobliwą $N \times N$. Wynika z tego, że

- A. istnieją macierze L dolnotrójkątna i U górnopróbkątna $N \times N$, takie że $A = LU$
- B. istnieją macierz permutacji Q oraz macierze L dolnotrójkątna i U górnopróbkątna $N \times N$, takie że $AQ = LU$
- C. istnieją macierz permutacji P oraz macierze L dolnotrójkątna i U górnopróbkątna $N \times N$, takie że $PA = LU$

Na macierzy nieosobliwej działa klasyczny algorytm podziału LU bez wyboru elementu głównego. Dlatego A. to prawda. W szczególności macierz permutacji Q w drugim podpunkcie może być macierz identycznościowa. Wtedy faktycznie $AQ = AI = A = LU$, więc B. TAK. Ten podpunkt był jednak podchwytliwy, gdyż przy algorytmie z wyborem elementu głównego otrzymujemy $PA = LU$, a nie $AP = LU$ (a przynajmniej zawsze tak pisał Przykry). Ponieważ działa algorytm bez GEPP, to zadziała $P = I$, ale przy wyborze elementu głównego może się faktycznie coś pozmieniać i wtedy P będzie jakieś inne, ale też w porządku. Stąd C. jest git.

10.2. A jest macierzą nieosobliwą o wymiarach $N \times N$. Wynika z tego, że koszt wyznaczenia macierzy A^{-1}

- NIE** A. za pomocą algorytmu LU wynosi $O(N^2)$ operacji arytmetycznych
NIE B. wynosi $O(N^2)$ operacji arytmetycznych, jeśli A jest górnopróbkątna
TAK C. wynosi $O(1)$ operacji arytmetycznych, jeśli A jest macierzą Householdera

A. Uzyskanie rozkładu LU już zajmuje czas $O(N^3)$.

B. Zajmuje czas $O(N^3)$, chociaż układ $Rx = y$ możemy rozwiązać w $O(N^2)$. Znając x i mając $R^{-1}Rx = Ix = x = R^{-1}y$ widać, że tak łatwo tego nie obliczymy – może być $O(N^3)$ elementów. A tak po prostu odwracać macierz to radzę nawet nie próbować.

C. Macierz Householdera jest swoją własną odwrotnością.

- 10.3.** Mamy daną macierz $A = \begin{bmatrix} 7 & -13 & 16 \\ -13 & 10 & -13 \\ 16 & -13 & 7 \end{bmatrix}$ oraz wektor $x_0 = \begin{bmatrix} 2019 \\ 2020 \\ 2021 \end{bmatrix}$. Niech $y_n = Ax_n$ oraz $x_{n+1} = \frac{y_n}{\|y_n\|}$.

Wtedy

TAK A. $\lim_{k \rightarrow \infty} \|x_k\| = 1$

NIE B. $\lim_{k \rightarrow \infty} \frac{x_k^T Ax_k}{x_k^T x_k} = 1$

TAK C. ciąg $\{x_n\}_{n \geq 0}$ jest zbieżny

A. Zauważmy, że $x_{n+1} = \frac{y_n}{\|y_n\|} = \frac{Ax_n}{\|Ax_n\|}$, czyli z każdą iteracją normalizujemy wektor, tj. skracamy/wydłużamy go, aby miał długość 1, czyli ta granica już po pierwszej iteracji jest prawdziwa.

B. Wiedząc, że $x_k \rightarrow v$, gdzie v jest wektorem własnym macierzy A mamy

$$\lim_{k \rightarrow \infty} \frac{x_k^T Ax_k}{x_k^T x_k} = \lim_{k \rightarrow \infty} \frac{v^T Av}{v^T v} = \lambda \lim_{k \rightarrow \infty} \frac{v^T v}{v^T v} = \lambda = 1.$$

Pytanie, czy 1 jest wartością własną macierzy A ? Aby to sprawdzić, rozwiązujeśmy układ równań $Av = v$, czyli dla $v = [x_1, x_2, x_3]^T$:

$$\begin{cases} 7x_1 - 13x_2 + 16x_3 = x_1 \\ -13x_1 + 10x_2 - 13x_3 = x_2 \\ 16x_1 - 13x_2 + 7x_3 = x_3 \end{cases}$$

co zostawiam jako zadanie dla Czytelnika.

C. Patrząc na wzór z podpunktu A. widzimy, że jest to metoda potęgowa, która jest zbieżna do wektora/wartości własnej macierzy A .

- 10.4.** Iteracyjna metoda rozwiązywania równania $f(x) = x - \frac{1}{2(1+x^2)}$ gdzie $x_{n+1} = \frac{1}{2(1+x_n^2)}$, jest

NIE A. zbieżna kwadratowo

TAK B. zbieżna dla $x_0 = 0$

TAK C. zbieżna dla $x_0 = 10$

Trzeba zauważyć, że funkcja $\Phi(x) = \frac{1}{2(1+x^2)}$ to kontrakcja (jak to zauważyć? trudno). Wówczas z twierdzenia Banacha o kontrakcji wiemy, że B. i C. jest prawdą. Użyjmy maszynki do rzędu żeby ocenić szybkość zbieżności tej metody. Pochodna $\Phi'(x) = \frac{-x}{(1+x^2)^2}$ ma miejsce zerowe dla $x_0 = 0$, a to na pewno nie jest punkt stały, więc metoda jest zbieżna liniowo.

- 10.5.** Szukamy zera funkcji $f(x) = x^2 - a$ na przedziale $[0, a]$, używając metody bisekcji. Prawdą jest, że

NIE A. dla $a = 99$ i dziesięciu iteracji błąd znalezionego rozwiązania szacuje się z góry przez 10^{-4}

NIE B. 10 iteracji wymaga policzenia co najmniej 20 wartości funkcji

NIE C. wzór iteracji dla tej metody to $x_{n+1} = x_n^2 + \frac{a}{2x_n}$

A. Błąd ten szacuje się z góry przez $\frac{1}{2^{10}} \cdot 99 \approx 0.1$.

B. Wymaga tylko 10.

C. Taki ciąg nie oddaje idei metody bisekcji. Dla $x > 0$ ten ciąg jest rosnący, co nie ma sensu z punktu widzenia bisekcji.

10.6. Dla danego $a \in \mathbb{R}$ szukamy rzeczywistych pierwiastków równania nieliniowego $x^3 - x = a$ metodą Newtona. Wówczas

NIE **A.** wzór na kolejną iterację metody to $x_{k+1} = \frac{2x_k^3 - a}{3x_k^2 + 1}$

TAK **B.** dla $a = 1$ i dowolnego $x_0 \geq 1$ metoda Newtona jest zbieżna do jakiegoś pierwiastka równania

TAK **C.** dla $a = 0$ i $x_0 = 10^{-2}$ metoda Newtona jest zbieżna kwadratowo do zera

A. Szukamy miejsca zerowego funkcji $f(x) = x^3 - x - a$, która jest różniczkowalna, a jej pochodna to $f'(x) = 3x^2 - 1$. Wówczas, wzór na kolejną iterację tej metody to

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{x_k^3 - x_k - a}{3x_k^2 - 1} = \frac{3x_k^3 - x_k - x_k^3 + x_k + a}{3x_k^2 - 1} = \frac{2x_k^3 + a}{3x_k^2 - 1},$$

co wygląda podobnie, ale to nie jest $\frac{2x_k^3 - a}{3x_k^2 + 1}$.

B. Mamy $f(x) = x^3 - x - 1$ i $f'(x) = 3x^2 - 1$. Szybko zauważymy, że w punkcie $x = \frac{1}{\sqrt{3}}$ funkcja f osiąga minimum lokalne, a rozwiązanie $x^* \in (1, 2)$ z własności Darboux. Ponadto, dla $x > \frac{1}{\sqrt{3}}$ funkcja już cały czas rośnie do nieskończoności, więc startując z dowolnego punktu $x_0 > \frac{1}{\sqrt{3}}$ na pewno metoda zbiegnie kwadratowo do miejsca zerowego x^* .

C. Szukamy miejsca zerowego $f(x) = x^3 - x = x(x-1)(x+1)$. Startując z $x_0 = 10^{-2}$ jesteśmy bardzo blisko 0 i najprawdopodobniej metoda Newtona zbiegnie kwadratowo właśnie do 0, ale nawet jak coś się wydarzy niespodziewanego, zbiegnie tak czy siak do -1 lub 1.

10.7. Całkę $\int_{-1}^1 f(x) dx$ przybliżamy kwadraturą interpolacyjną $Q(f) = A_1 f(x_1) + A_2 f(x_2)$. Wtedy

TAK **A.** dla $A_1 = A_2 = 1$, $x_1 = -1$, $x_2 = 1$ kwadratura Q jest dokładna dla wszystkich wielomianów stopnia 1

NIE **B.** można dobrać x_1 i x_2 , tak aby kwadratura Q była rzędu 2 oraz $A_1 = A_2 = 2$

TAK **C.** maksymalny rząd kwadratury Q jest równy 4

A. Niech $p(x) = ax + b$. Wtedy $\int_{-1}^1 p(x) dx = (\frac{a}{2}x^2 + bx)|_{-1}^1 = (\frac{a}{2} + b) - (\frac{a}{2} - b) = 2b$, a kwadratura wynosi $Q(f) = A_1 f(x_1) + A_2 f(x_2) = (a+b) + (-a+b) = 2b$, więc jest dokładna.

B. NIE, bo jak wiemy z poprzedniego podpunktu, $\int_{-1}^1 p(x) dx = 2b$, a taka kwadratura Q wynosi $Q(f) = 2(ax_1 + b) + 2(ax_2 + b) = 2a(x_1 + x_2) + 4b$, więc nawet jakbyśmy wzięli $x_1 = -x_2$, to nie pozbędziemy się nadmiarowych $2b$.

C. Wynika to wprost z twierdzenia o rzędzie.

10.8. Stosując metodę złożonej kwadratury trapezów na odcinku $[a, b]$ dla funkcji f , można obliczyć dokładnie (pomijając błędy zaokrągleń) całkę $\int_a^b f(x) dx$ dla

TAK **A.** $f(x) = 3 - x$

NIE **B.** $f(x) = 3 - x + x^2$

NIE **C.** $f(x) = \sin(x)$

A. $\int_a^b (3-x) dx = (3x - \frac{x^2}{2}) \Big|_a^b = (3b - \frac{b^2}{2}) - (3a - \frac{a^2}{2}) = \frac{a^2 - b^2}{2} + 3(b-a) = \frac{b-a}{2}(6-a-b)$,
 $Q^T(f) = \frac{b-a}{2}(3-a+3-b) = \frac{b-a}{2}(6-a-b)$, więc odpowiedź to TAK.

B. $\int_a^b (3-x+x^2) dx = (3x - \frac{x^2}{2} + \frac{x^3}{3}) \Big|_a^b = (3b - \frac{b^2}{2} + \frac{b^3}{3}) - (3a - \frac{a^2}{2} + \frac{a^3}{3}) = \frac{b^3 - a^3}{3} + \frac{a^2 - b^2}{2} + 3(b-a) =$
 $\frac{b-a}{2}(6-a-b + \frac{2(a^b+ab+b^2)}{3})$,
 $O^T(f) = \frac{b-a}{2}(3-a+a^2+3-b+b^2) = \frac{b-a}{2}(6-a-b+a^b+b^2)$, więc odpowiedź to NIE.

C. Aż żal to liczyć, widać, że $I(f) \neq Q^T(f)$.

11

Programowanie obiektowe

Materiały teoretyczne zostały opracowane na podstawie materiałów Janusza Jabłonowskiego (dostęp do [kursu na Moodle](#)) oraz [dokumentacji Javy](#).

Podstawa programowa

1. Pojęcia **klasy i obiektu**.
2. **Konstruktory w Javie** i ich zastosowanie.
3. **Kapsułkowanie danych** i zakresy widoczności w Javie.
4. **Dziedziczenie** i hierarchie klas. Klasy abstrakcyjne i interfejsy.
5. Podmienianie metod jako realizacja **polimorfizmu**.
6. Obsługa **wyjątków**. Hierarchie wyjątków.
7. Standardowe **kolekcje** w Javie.

11.1.

Klasy, obiekty i konstruktory

Klasa jest szablonem lub wzorcem, który definiuje strukturę i zachowanie obiektów. Można ją traktować jako „fabrykę” obiektów. Definiuje ona właściwości (**atrybuty**) i zachowania (**metody**).

Obiekt jest instancją klasy. Możemy tworzyć wiele obiektów opartych na jednej klasie, a każdy z tych obiektów będzie miał swoje własne wartości pól (atrybutów).

Konstruktory

Obiekty są tworzone (tylko) za pomocą **konstruktorów**, czyli specjalnej metody zdefiniowanej w klasie. Zadaniem konstruktora jest zainicjalizować obiekt, czyli ustawić jego początkowe wartości pól. Najważniejsze cechy konstruktora:

- Nazwa konstruktora jest taka sama jak nazwa klasy, w której się znajduje.
- Konstruktor nie ma żadnego typu zwracanego.
- Może istnieć wiele konstrktorów w jednej klasie, o ile różnią się one pod względem liczby argumentów lub ich typów – jest to tzw. **przeciążanie konstruktorów**.
- Jeśli nie zdefiniujemy żadnego konstruktora w klasie, kompilator dostarczy **konstruktor domyślny**, który nie przyjmuje żadnych argumentów i nie wykonuje żadnych dodatkowych operacji.

■ Obiekty w Javie

W języku Java istnieje słowo kluczowe (metazmienna) **this**. Jego zastosowanie to:

- Odwoływanie się do pól obiektu – pozwala to odróżnić lokalne zmienne od pól o tych samych nazwach.
- Wywoływanie innego konstruktora w tej samej klasie, co pozwala na uniknięcie duplikacji kodu. Takie wywołanie musi znajdować się w pierwszej linii konstruktora.
- Przekazywanie obiektu do innych metod.

Przykład 1.

Przykładowa definicja klasy w języku Java:

```
class Dog {  
  
    // Atrybuty  
    String name;  
    int age;  
  
    // Konstruktory  
    Dog(String name) {  
        this(name, 0);  
    }  
  
    Dog(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    // Metody  
    void hello() {  
        System.out.println("Hi! My name is " + name + ". Woof, woof!");  
    }  
  
    int getAge() {  
        return age;  
    }  
  
    void setName(String name) {  
        this.name = name;  
    }  
}
```

Wykonując następujący program

```
class Main {  
    public static void main(String args[]) {  
        Dog myDog = new Dog("Rocky", 4);  
        myDog.hello();  
        System.out.println("My dog is " + myDog.getAge() + " years old.");  
  
        Dog puppy = new Dog("<in progress>");  
        puppy.hello();  
        puppy.setName("Max");  
        puppy.hello();  
    }  
}
```

w konsoli zostanie wypisane:

```
Hi! My name is Rocky. Woof Woof!
My dog is 4 years old.
Hi! My name is <in progress>. Woof Woof!
Hi! My name is Max. Woof Woof!
```

■ Kolejność wykonywania instrukcji konstruktorów obiektów

Kolejność wykonywania instrukcji w konstruktorach obiektów z atrybutami zależy od tego, czy pierwszą instrukcją jest wywołanie konstruktora przy użyciu konstrukcji **this**.

Dla konstruktora korzystającego z konstruktora **this** kolejność ta jest następująca:

1. Konstruktor parametrów konstruktora
2. Konstruktor **this** (i wszystko z nim związane)
3. Reszta ciała konstruktora

Dla konstruktora nie korzystającego z konstruktora **this**:

1. Konstruktor parametrów konstruktora
2. Konstruktor zdefiniowanych (w definicji klasy) atrybutów
3. Reszta ciała konstruktora

Działanie to obrazuje poniższy przykład:

Przykład 2.

Poniżej zdefiniowano dwie klasy w Javie

```
class I {
    int val;

    I() {
        this.val = 0;
        System.out.print("I() ");
    }

    I(int val) {
        this.val = val;
        System.out.print("I(" + val + ") ");
    }
}

class A {
    I i;

    A() {
        System.out.print("A() ");
    }

    A(int i) {
        this.i = new I(i);
```

```

        System.out.print("A(" + i + ") ");
    }

}

class B {
    A a = new A(42);

    B() {
        this(new A(7));
        System.out.print("B() ");
    }
    B(A a) {
        System.out.print("B(A {i = " + a.i + "}) ");
    }
}

```

Wykonując następujący program

```

class Main {
    public static void main(String[] args) {
        System.out.print("a1: ");
        A a1 = new A();
        System.out.print("\na2: ");
        A a2 = new A(2);
        System.out.print("\nb1: ");
        B b1 = new B();
        System.out.print("\nb2: ");
        B b2 = new B(a1);
        System.out.print("\nb3: ");
        B b3 = new B(a2);
    }
}

```

w konsoli zostanie wypisane:

```

a1: A()
a2: I(2) A(2)
b1: I(7) A(7) I(42) A(42) B(A {i = I@85ede7b}) B()
b2: I(42) A(42) B(A {i = null})
b3: I(42) A(42) B(A {i = I@5b2133b1})

```

11.2.

Kapsułkowanie danych i zakresy widoczności

Kapsułkowanie danych to mechanizm programowania obiektowego, który polega na ukrywaniu wewnętrznej implementacji danych obiektu i udostępnianiu kontrolowanego dostępu do nich za pomocą **metod dostępowych** (gettery i settery). Dzięki temu kontrolujemy, jak dane są odczytywane i modyfikowane, co zapewnia większe bezpieczeństwo i elastyczność kodu.

Zakresy widoczności w Javie

W języku Java rozróżniamy 4 zakresy widoczności:

- **prywatna (private)**: najbardziej restrykcyjny poziom widoczności. Pola i metody są dostępne tylko

wewnątrz tej samej klasy (oraz ewentualnie z klasy zadeklarowanej na najwyższym poziomie struktury programu, w której jest zawarta taka deklaracja. W ramach widoczności prywatnej atrybuty prywatne są też dostępne z innych obiektów tej samej klasy.).

- **domyślna** (brak modyfikatora / widoczność pakietowa): jeśli nie określmy żadnego modyfikatora widoczności, pola i metody są dostępne **wewnątrz pakietu** (package), w którym się znajdują.
- **chroniona (protected)**: chronione pola i metody są dostępne dla klas **w tym samym pakiecie oraz dla klas dziedziczących** (podklas, definiowanych słowem kluczowym **extends**).
- **publiczna (public)**: najbardziej otwarty poziom widoczności. Pola i metody publiczne są dostępne **dla wszystkich klas**, zarówno wewnątrz jak i na zewnątrz pakietu.

Przykład 1.

W poniższym przykładzie kod funkcji *Local2::someFunction()* jest poprawny, ponieważ atrybut *someData* jest dostępny w klasie *Outer* zgodnie z właściwościami widoczności prywatnej, a co za tym idzie, jest również dostępny w klasie *Local2*.

```
class Outer {  
  
    class Local1 {  
        private int someData;  
    }  
  
    class Local2 {  
        int someFunction() {  
            return new Local1().someData;  
        }  
    }  
}
```

Zestaw zadań

11.1. Mówimy, że widoczność *A* jest szersza niż widoczność *B*, gdy zawsze kiedy widoczne jest *B*, widoczne jest też *A*. Widoczność pakietowa jest szersza niż widoczność

- A. publiczna
- B. chroniona
- C. prywatna

11.2. Dany jest kod w języku Java:

```
class A {  
    private int i1;  
    protected int i2;  
}  
  
class B extends A {  
    private int j1;  
  
    public void m(B b) {  
        // b.i1 = 0;  
        // b.i2 = 0;  
        // b.j1 = 0;  
    }  
}
```

Kod skompiluje się bez błędów po odkomentowaniu linii

- A. b.i1 = 0;
- B. b.i2 = 0;
- C. b.j1 = 0;

11.3.

Dziedziczenie i hierarchie klas. Polimorfizm

Dziedziczenie to jeden z fundamentalnych konceptów programowania obiektowego. W Javie klasy tworzą hierarchię w formie lasu, a w innych językach (np. C++) może być to las DAG-ów. Klasy dziedziczą po sobie atrybuty oraz metody – inaczej nazywane składowymi.

Java umożliwia dziedziczenie klas przy użyciu słowa kluczowego **extends**.

Przykład 1.

Zdefiniowana poniżej klasa *Nektarynka* dziedziczy po klasie *Owoc* i nadpisuje jej metodę *czyMaPestke()* (co dodatkowo zaznaczamy za pomocą adnotacji **@Override**):

```
class Owoc {  
  
    public boolean czyMaPestke() {  
        return false;  
    }  
}  
  
class Nektarynka extends Owoc {  
  
    @Override  
    public boolean czyMaPestke() {  
        return true;  
    }  
}
```

W Javie **nie istnieje wielodziedziczenie** po klasach (w C++ już tak – jest to popularny *diamond inheritance problem*). Oznacza to, że dana klasa Java nie może dziedziczyć po więcej niż jednej klasie lub interfejsie.

Zdefiniowaną w powyższym przykładzie klasę *Owoc* nazywamy **nadklassą** lub klasą bazową, a klasę *Nektarynka* – **podklassą** lub klasą pochodną. Intuicyjnie, dziedziczenie klasowe opisuje, czym jest dana podklasa.

Podczas tworzenia się obiektu podklasy **zawsze na początku wywoływany jest bezargumentowy konstruktor nadklasty**, chyba że w pierwszej linijce konstruktora jest użycie konkretnego konstruktora nadklasty, lub innego konstruktora danej klasy. W szczególności konstruktor nadklasty wołany jest przed konstruktorami atrybutów

Działanie to obrazuje poniższy przykład:

Przykład 2.

Zdefiniowane klasy z Javie

```
class A {  
    A() {  
        System.out.print("A() ");  
    }  
}
```

```

A(int i) {
    System.out.print("A(" + i + ") ");
}

A(A a) {
    System.out.print("A(" + a + ") ");
}
}

class B extends A {
    A x = new A(0);
    B() {
        this(new A(4));
        System.out.print("B() ");
    }

    B(int i) {
        System.out.print("B(" + i + ") ");
    }

    B(A a) {
        super(a);
        System.out.print("B(" + a + ") ");
    }
}

```

Wykonując następujący program

```

class Main {
    public static void main(String[] args) {
        System.out.print("a: ");
        A a = new A();
        System.out.print("\nb1: ");
        B b1 = new B();
        System.out.print("\nb2: ");
        B b2 = new B(2);
        System.out.print("\nb3: ");
        B b3 = new B(a);
    }
}

```

w konsoli zostanie wypisane:

```

a: A()
b1: A(4) A(A@7adf9f5f) A(0) B(A@7adf9f5f) B()
b2: A() A(0) B(2)
b3: A(A@77459877) A(0) B(A@77459877)

```

Metoda wirtualna to metoda, którą możemy nadpisać (ang. *override*). **Metody prywatne nie są wirtualne.**

Przykład 3.

Poniżej przykład próby nadpisania prywatnej metody `Owoc::czyMaPestke()` w klasie `Nektarynka` (dziedziczącej po klasie `Owoc`):

```
class Owoc {  
  
    private boolean czyMaPestke() {  
        return false;  
    }  
}  
  
class Nektarynka extends Owoc {  
  
    // To się skompiluje, ale nie nadpisze metody, tylko stworzy nową  
    private boolean czyMaPestke() {  
        return true;  
    }  
  
    // To się nie skompiluje, dlatego używanie @Override jest ważne!  
    @Override  
    private boolean czyMaPestke() {  
        return true;  
    }  
}
```

■ Polimorfizm

Polimorfizm to mechanizm polegający na tym, że klasa bazowa może zostać uzyskana przez stworzenie jej klas pochodnych.

Przykład 4.

Bazując na klasach `Owoc` i `Nektarynka` z poprzednich przykładów, dzięki polimorfizmowi możemy zapisać

```
Owoc o = new Nektarynka();
```

W Javie zastosowanie ma także **zasada podstawialności**: zawsze można podstawić obiekt podklasy zamiast obiektu nadklasy.

Przykład 5.

W poniższym przykładzie zdefiniowano trzy klasy dziedziczące po sobie w kolejności `SuperE`, `E`, `SubE` oraz pokazano zastosowanie zasady podstawialności:

```
class SuperE {}  
  
class E extends SuperE {}  
  
class SubE extends E {}  
  
class A {  
    public E foo(E e) {  
        return new E();  
    }  
}
```

```

}

class B extends A {

    // To się skompiluje
    @Override
    public SubE foo(E e) {
        return new SubE();
    }

    // A to nie
    @Override
    public SuperE foo(E e) {
        return new SuperE();
    }
}

```

Przykład 6.

Możemy również zwiększyć zakres widoczności dziedziczonych metod:

```

class Owoc {

    protected boolean czyMaPestke() {
        return false;
    }
}

class Nektarynka extends Owoc {

    @Override
    public boolean czyMaPestke() {
        return true;
    }
}

```

Przypis redakcji

A co w przypadku zmniejszenia zakresu widoczności dziedziczonych metod? Wypadałoby dorzucić jeszcze taki przykład (może być w tej samej ramce powyżej), bo taka sytuacja jest np. w zadaniu niżej.

To było na egzaminie

Rozważamy programy napisane w języku Java. W pewnym pakiecie zdefiniowano klasy *A* i *B*. Jeżeli klasa *A* jest nadklassą *B*, to w treści klasy *B* można zdefiniować

- A. metody o takich samych nagłówkach jak metody zdefiniowane w klasie *A*
- B. metodę o takim samym nagłówku, jak metoda zdefiniowana w klasie *A*, ale z większą widocznością niż była podana w metodzie z klasy *A*
- C. metodę o takim samym nagłówku, jak metoda zdefiniowana w klasie *A*, ale z szerszą widocznością niż była podana w metodzie z klasy *A*

W przypadku podpunktu **A.** metody zostaną wtedy nadpisane (jeśli nie są to metody prywatne) lub zdefiniowane jako nowe (jeśli są prywatne) – odpowiedź to TAK.

Faktem jest, że żeby nadpisać funkcję, można to zrobić jedynie z takim samym nagłówkiem i taką samą lub szerszą widocznością. Odpowiedź w **B.** jest więc fałszywa, a w **C.** – prawdziwa.

■ Składowe klasowe oraz **final**

Składowe klasowe (**static**) to składowe mające jednego reprezentanta dla wszystkich obiektów danej klasy.

```
class LosowaKlasa {  
    static int licznik; // jest wspólny dla wszystkich instancji LosowejKlasy  
  
    static void hello() {  
        System.out.println("Hello");  
    }  
}
```

Metody klasowe (statyczne) nie są wirtualne (tj. nie można ich nadpisać).

Przypis redakcji

Za mało teorii o metodach statycznych: brakuje informacji m.in. o możliwości ich wywołania bez wywoływanego konstruktora obiektu klasy, w której są zdefiniowane.

Składowe **final**:

- Klasa oznaczona słowem kluczowym **final** nie może być klasą bazową innej klasy.
- Metoda oznaczona słowem kluczowym **final** nie jest wirtualna.
- Zmienna oznaczona słowem kluczowym **final** nie może zostać zmieniona po tym jak raz została zainicjalizowana.

■ Przeciążanie

Oprócz nadpisania, istnieje jeszcze mechanizm **przeciążania**, który polega na stworzeniu kilku definicji danej metody różniących się parametrami (ale: typ metody musi się zgadzać).

Przykład 7.

W poniższym przykładzie w klasie *B* metoda *foo()* została przeciążona na trzy różne sposoby:

```
class SuperE {}  
  
class E extends SuperE {}  
  
class SubE extends E {}  
  
class A {  
    public E foo(E e) {  
        return new E();  
    }  
}
```

```

class B extends A {

    public E foo(SuperE e) {
        return new E();
    }

    public E foo(SubE e) {
        return new E();
    }

    public E foo(String s, int i) {
        return new E();
    }
}

```

■ Przykłady ogólne

Rozważmy jeszcze parę ważnych przykładów obrazujących działanie dziedziczenia.

Przykład 8.

Dane są następujące definicje klas *A*, *B*, *C*:

```

class A {
    int iA = 1;
    void infoA() {
        System.out.println("Jestem infoA(), wywołano mnie w obiekcie klasy" +
                           this.getClass().getSimpleName() + ", iA = " + iA);
    }
}

class B extends A {
    int iB = 2;
    void infoB() {
        infoA();
        System.out.println("Jestem infoB(), wywołano mnie w obiekcie klasy" +
                           this.getClass().getSimpleName() + ", iA = " + iA + ", iB = " + iB);
    }
}

class C extends B {
    int iC = 3;
    void infoC() {
        infoA();
        infoB();
        System.out.println("Jestem infoC(), wywołano mnie w obiekcie klasy" +
                           this.getClass().getSimpleName() + ", iA = " + iA + ", iB = " + iB +
                           ", iC = " + iC);
    }
}

```

Po wywołaniu funkcji

```

public static void main(String[] args) {
    C c = new C();
}

```

```
    c.infoC();  
}
```

na wyjściu wypisane zostanie

```
Jestem infoA(), wywołano mnie w obiekcie klasy C, iA = 1  
Jestem infoA(), wywołano mnie w obiekcie klasy C, iA = 1  
Jestem infoB(), wywołano mnie w obiekcie klasy C, iA = 1, iB = 2  
Jestem infoC(), wywołano mnie w obiekcie klasy C, iA = 1, iB = 2, iC = 3
```

Natomiast po wywołaniu funkcji

```
public static void main(String[] args) {  
    A a = new C();  
    C c = new C();  
    System.out.println("a.iA = " + a.iA + ", c.iA = " + c.iA);  
}
```

zgodnie z zasadą podstawialności wypisane zostanie `a.iA = 1, c.iA = 1.`

Przykład 9.

Dokonajmy niewielkiej zmiany w atrybutach klas *A*, *B*, *C* z poprzedniego przykładu:

```
class A {  
    int i = 1;  
    void infoA() {  
        System.out.println("Jestem infoA(), wywołano mnie w obiekcie klasy" +  
                           this.getClass().getSimpleName() + ", i z A = " + i);  
    }  
}  
  
class B extends A {  
    int i = 2;  
    void infoB() {  
        infoA();  
        System.out.println("Jestem infoB(), wywołano mnie w obiekcie klasy" +  
                           this.getClass().getSimpleName() + ", i z A =" + ((A) this).i +  
                           ", i z B = " + i);  
        // Zamiast ((A) this).i mogłoby być super.i  
    }  
}  
  
class C extends B {  
    int i = 3;  
    void infoC() {  
        infoA();  
        infoB();  
        System.out.println("Jestem infoC(), wywołano mnie w obiekcie klasy" +  
                           this.getClass().getSimpleName() + ", i z A =" + ((A) this).i +  
                           ", i z B = " + ((B) this).i + ", i z C = " + i);  
        // Nie mogłoby być super.i, bo super sięga tylko 1 poziom wyżej  
    }  
}
```

Teraz po wywołaniu funkcji

```
public static void main(String[] args) {
    C c = new C();
    c.infoC();
}
```

na wyjściu wypisane zostanie

```
Jestem infoA(), wywołano mnie w obiekcie klasy C, i z A = 1
Jestem infoA(), wywołano mnie w obiekcie klasy C, i z A = 1
Jestem infoB(), wywołano mnie w obiekcie klasy C, i z A = 1, i z B = 2
Jestem infoC(), wywołano mnie w obiekcie klasy C, i z A = 1, i z B = 2, i z C = 3
```

Natomiast funkcja

```
public static void main(String[] args) {
    A a = new C();
    C c = new C();
    System.out.println("a.i = " + a.i + ", c.i = " + c.i);
}
```

wypisze a.i = 1, c.i = 3.

Uwaga: o wartości atrybutu decyduje typ statyczny obiektu – czyli *a.i* w powyższym przykładzie przyjmuje wartość 1, bo typ *a* to *A*. Odwrotnie jest w przypadku metod, co obrazuje poniższy przykład.

Przykład 10.

Zmodyfikujemy deklaracje klas *A*, *B*, *C* z poprzedniego przykładu:

```
class A {
    int i = 1;
    void info() {
        System.out.println("Jestem infoA(), wywołano mnie w obiekcie klasy" +
                           this.getClass().getSimpleName() + ", i z A = " + i);
    }
}

class B extends A {
    int i = 2;
    void info() {
        super.info();
        System.out.println("Jestem infoB(), wywołano mnie w obiekcie klasy" +
                           this.getClass().getSimpleName() + ", i z A =" + ((A) this).i +
                           ", i z B = " + i);
    }
}

class C extends B {
    int i = 3;
    void info() {
        super.info();
        System.out.println("Jestem infoC(), wywołano mnie w obiekcie klasy" +
                           this.getClass().getSimpleName() + ", i z A =" + ((A) this).i +
```

```
        ", i z B = " + ((B) this).i + ", i z C = " + i);  
    }  
}
```

Teraz po wywołaniu metody

```
public static void main(String[] args) {  
    C c = new C();  
    c.info();  
}
```

zostanie wypisane

```
Jestem infoA(), wywołano mnie w obiekcie klasy C, i z A = 1  
Jestem infoA(), wywołano mnie w obiekcie klasy C, i z A = 1  
Jestem infoB(), wywołano mnie w obiekcie klasy C, i z A = 1, i z B = 2  
Jestem infoC(), wywołano mnie w obiekcie klasy C, i z A = 1, i z B = 2, i z C = 3
```

Okazuje się, że metoda

```
public static void main(String[] args) {  
    A a = new C();  
    a.info();  
}
```

wypisze dokładnie taki sam komunikat, jak poprzednia!

To było na egzaminie

Dany jest kod w języku Java:

```
class A {  
    public void m1() { System.out.println("A"); }  
    public void m2() { m1(); }  
}  
  
class B extends A {  
    public void m1() { System.out.println("B"); }  
}  
  
class Main {  
    public static void main(String[] args) {  
        A a = new A();  
        B b = new B();  
        // a.m1();  
        // b.m1();  
        // b.m2();  
    }  
}
```

Po uruchomieniu program wypisze literę „A” po odkomentowaniu linii

A. a.m1();

B. b.m1();

C. `b.m2();`

- A. Obiekt `a` z klasy `A` wywoła zdefiniowaną dla jego klasy metodę `m1()`, odpowiedź to TAK.
- B. Obiekt `b` z klasy `B` wywoła zdefiniowaną dla jego klasy metodę `m1()`, która nadpisuje metodę `m1()` z poprzedniego podpunktu – odpowiedź to NIE.
- C. Obiekt `b` z klasy `B` wywoła zdefiniowaną dla jego nadklasy `A` metodę `m2()`. Wewnątrz `m2()` wywołane zostanie `m1()`, które jest nadpisywane przez `B`, więc wypisane zostanie „B” i odpowiedź jest fałszywa.

Zestaw zadań

- 11.3. Dana jest definicja klasy w języku Java:

```
final class A {  
    static A a;  
    A() { a = this; }  
    static boolean f() { return a.equals(a); }  
    boolean g() { return a == this; }  
    boolean h() { return equals(this); }  
}
```

Dla takiej klasy

- A. każde wywołanie funkcji `f` daje wynik `true`
- B. każde wywołanie funkcji `g` daje wynik `true`
- C. każde wywołanie funkcji `h` daje wynik `true`

(przyp. red.: w części teoretycznej trzeba dopisać sekcję na temat tych wszystkich rodzajów równości, bo nie ma jak zrobić tego zadania)

- 11.4. Dany jest program w języku Java:

```
class A {  
    public void m1() { System.out.println("A"); }  
    public void m2(A a) { a.m1(); }  
}  
  
class B extends A {  
    public void m1() { System.out.println("B"); }  
}  
  
class C extends B {  
    public void m1() { System.out.println("C"); }  
    public void m2(A a) { a.m2(this); }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        A a1 = new A();  
        A a2 = new B();  
        A a3 = new C();  
        a3.m1(); // (a)  
        a2.m2(a3); // (b)  
    }  
}
```

```

        a3.m2(a1); // (c)
    }
}

```

Podczas wykonania

- A. wiersza a3.m1(); tego programu zostanie wypisane A
- B. wiersza a2.m2(a3); tego programu zostanie wypisane B
- C. wiersza a3.m2(a1); tego programu zostanie wypisane C

11.5. Rozważamy programy napisane w języku Java. Jeżeli klasa *A* jest nadklasą *B*, to w treści klasy *B*

- A. trzeba zdefiniować wszystkie metody zdefiniowane w klasie *A*
- B. można zdefiniować metody o innych nagłówkach niż metody zdefiniowane w klasie *A*
- C. można zdefiniować metodę o takim samym nagłówku jak metoda zdefiniowana w klasie *A*, ale z większą widocznością niż była podana w metodzie z klasy *A*

11.6. Dany jest program w języku Java:

```

class A {
    public A() { System.out.print("1"); m1(); }
    public A(int i) { System.out.print("2"); }
    public A(A a) { System.out.print("3"); }
    public void m1() { System.out.print("4"); }
}

class B extends A {
    A a;
    public B(A a) { System.out.print("5"); }
    public B() { this(new A()); System.out.print("6"); }
    @Override
    public void m1() { System.out.print("7"); }
}

public class Main {
    public static void main(String[] args) {
        A a = new B();
    }
}

```

Po uruchomieniu tego programu na pewno (oprócz, być może, innych znaków) zostanie wypisana cyfra

- A. 1
- B. 4
- C. 3

11.7. Dany jest program w języku Java:

```

class A {
    void m1() { System.out.println("1"); }
    static void m2() { System.out.println("2"); }
    void m3() { m1(); }
    void m4() { m2(); }
}

class B extends A {
}

```

```

@Override
void m1() { System.out.println("3"); }
static void m2() { System.out.println("4"); }
}

public class Main {
    public static void main(String[] args) {
        A a = new B();
        a.m1();
        a.m3();
        a.m4();
    }
}

```

W takim programie

- A. wywołanie a.m1() spowoduje wypisanie jedynki
- B. wywołanie a.m3() spowoduje wypisanie jedynki
- C. wywołanie a.m4() spowoduje wypisanie dwójki

11.4.

Klasy abstrakcyjne i interfejsy

Ważnym konceptem, który zapewnia wyabstrahowanie wspólnych cech wielu klas, są **klasy abstrakcyjne**. Deklarujemy je przy użyciu słowa kluczowego **abstract**. Należy pamiętać o kilku zasadach:

- nie da się stworzyć instancji klasy abstrakcyjnej,
- klasa abstrakcyjna może mieć metody abstrakcyjne zadeklarowane ze słowem kluczowym **abstract**, które nie mogą mieć treści,
- metoda abstrakcyjna musi być zdefiniowana w podklasach nieabstrakcyjnych,
- można wywoływać metody abstrakcyjne i nie ma w tym nic gorszącego,
- możliwe jest (choć rzadkie) podmienienie konkretnej metody na abstrakcyjną,
- klasa która posiada lub odziedziczy metodę abstrakcyjną musi być zadeklarowana jako abstrakcyjna.

Przykład 1.

W poniższym przykładzie zaimplementowano klasę abstrakcyjną *Student* oraz dziedziczącą po niej klasę (konkretną) *StudentMIMu*:

```

abstract class Student {
    public String nrIndeksu;

    // Wymusza implementację w podkласach
    public abstract void obliczCalke(String calka);

    // Nie wymusza implementacji w podkласach
    public void dajGlos() {
        System.out.println("Debil");
    }
}

```

```

// Konkretna metoda może wywołać abstrakcyjną
public void yooWtf(){
    obliczCalke("calka");
}
}

public class StudentMIMu extends Student {

    @Override
    public void obliczCalke(String calka) {
        System.out.println("Już");
    }

    @Override
    public void dajGlos() {
        System.out.println("Nadal debil, ale z MIM-u");
    }
}

```

■ Interfejsy

Kolejnym konceptem zapewniającym abstrakcję są **interfejsy**, których używamy przy użyciu słowa kluczowego **implements**. Intuicyjnie, interfejsy mówią o tym, co dana klasa potrafi zrobić, a nie czym jest. Ten sam interfejs mogą implementować dwie kompletne niezwiązane ze sobą klasy. Interfejsy zapewniają polimorfizm oraz wielodziedziczenie.

Przykład 2.

Poniżej przedstawiono interfejs *Comparable* oraz klasy *Student* i *Makita40VMaxXGT* implementujące go:

```

public interface Comparable {
    boolean compare();
}

// Klasa może implementować kilka interfejsów
public abstract class Student implements Comparable, Debilable {

    // Klasa musi zaimplementować metodę z interfejsu
    public boolean compare() { ... }
}

public class Makita40VMaxXGT implements Comparable {
    public boolean compare() { ... }
}

```

Interfejs może zawierać:

- stałe – są z definicji **public static final** i nie można zmienić ich widoczności,
- abstrakcyjne metody,
- statyczne metody,
- metody z domyślną implementacją, oznaczone słowem kluczowym **default**,
- zagnieżdżone interfejsy,

- zagnieżdżone klasy.

Należy pamiętać, że:

- nie da się stworzyć instancji interfejsu,
- interfejs może być całkowicie pusty,
- metody są z założenia publiczne i abstrakcyjne,
- nie można używać słowa kluczowego **final**,
- metody nie mogą być **protected**.

Przykład 3.

Poniżej zaimplementowano interfejs *Printable*:

```
public interface Printable {
    // Stała
    String msg = "Hi!";

    // Metoda abstrakcyjna
    void printMe();

    // Metoda statyczna
    void printMeButStatic() {
        System.out.println("Static method example");
    }

    // Metoda z domyślną implementacją
    default void printMeDefault() {
        System.out.println("NPC method");
    }

    // Zagnieżdżony interfejs
    interface NestedPrintable {
        void wypiszWymaluj();
    }

    // Zagnieżdżona klasa (dowolnie bogata)
    class NPC { ... }
}
```

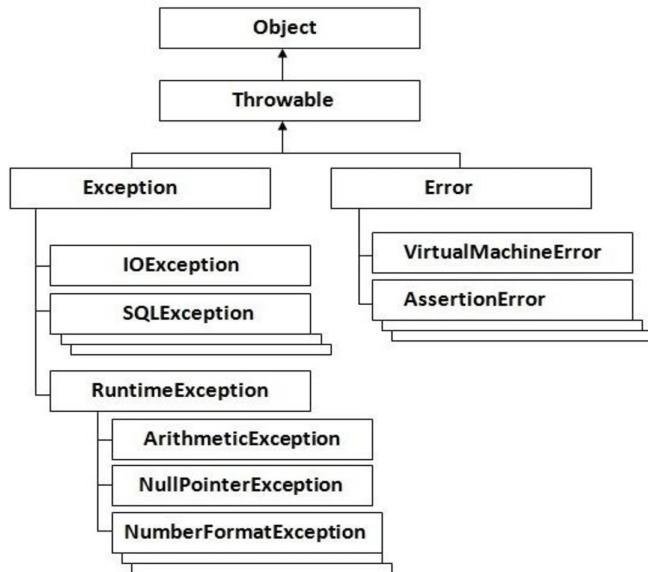
Interfejsy mogą się rozszerzać również przez użycie **extends**.

```
public interface Printable extends Comparable { ... }
```

11.5.

Wyjątki

W Javie **wyjątkami** są podklasy klasy *Throwable*, w praktyce jednak są to podklasy klasy *Exception*.



Klasa `Throwable` zawiera takie metody, jak na przykład `getMessage()`, `printStackTrace()` czy `getCause()`, które przekazują więcej informacji o wyjątku.

Klasa `Error` dotyczy poważnych problemów, które nie są przechwytywane.

Klasa `Exception` zawiera natomiast wyjątki (oprócz `RuntimeException`), które powinny być obsługiwane przez program i są to tak zwane **wyjątki nadzorowane** (ang. *checked exceptions*). **Wyjątki nienadzorowane** (ang. *unchecked exceptions*) są to natomiast podklasy `RuntimeException` i ich nie obsługuje my.

■ Zgłaszanie wyjątków

Zgłaszanie wyjątków odbywa się za pomocą słowa kluczowego `throw`, który przerywa normalne wykonanie programu. Następnie na stosie wykonania programu odbywa się poszukiwanie obsługi wyjątku: bloki i metody niezawierające obsługi są zdejmowane ze stosu, a brak obsługi kończy się przerwaniem działania programu.

Lista zgłaszanych przez metodę wyjątków jest deklarowana w jej nagłówku za pomocą `throws`. Lista ta zawiera tylko wyjątki nadzorowane oraz **musi zawierać wszystkie zgłaszane przez metodę wyjątki** (choć może mieć też ich więcej). Podczas komplikacji kompilator weryfikuje poprawność tej listy.

Przy nadpisywaniu metody zakres tej listy **może być węższy** niż metody nadzędnej (ale **nie może być szerszy**). Jest to ważne, by uniknąć sytuacji, w której nie możemy użyć obiektu podklasy jako obiektu nadklasy (ang. *Liskov substitution*). Z tego powodu nie możemy zadeklarować więcej wyjątków niż było ich w nadklassie. Możemy zadeklarować mniej, ponieważ nadpisywana metoda, różniąca się od oryginalnej, mogłaby po prostu w kodzie wcale tych wyjątków nie rzucać, wtedy nie ma potrzeby deklarować ich w nagłówku. W przeciwnym wypadku byłibyśmy zmuszeni obsługiwać nierzucane wyjątki, używając obiektu danej klasy wprost (nie jako obiekt nadklasy), co byłoby bardzo niepraktyczne.

■ Obsługa wyjątków

Obsługa wyjątków odbywa się za pomocą instrukcji `try ... catch ...`. Po `try` wpisujemy normalne działanie algorytmu, w którym może wystąpić wyjątek, a po `catch` obsługę konkretnego wyjątku (instrukcji `catch` może być wiele i ich kolejność ma znaczenie: jeśli kilka klauzul pasuje, to zostanie wybrana pierwsza z nich). W `catch` możemy też zgłosić wyjątek, ale nie będzie on już obsłużony w tej samej instrukcji `try ... catch`.

Do instrukcji `try ... catch` możemy jeszcze dodać dodatkową instrukcję `finally ...`, która zostanie wykonana zawsze przy wychodzeniu z bloku `try ... catch` (można też pominąć wszystkie `catch` i mieć tylko blok `try ... finally ...`). Instrukcje zawarte po `finally` wykonane zostaną zawsze, gdy złapany zostanie wyjątek (nawet taki niepasujący do żadnego `catch`).

Przykład 1.

W poniższym kodzie zaprezentowano obsługę wyjątków:

```
public class UczymySieWyjatkow {
    void rzucamyWyjatkiem(String s) throws Exception1, Exception2 {
        System.out.println("Przed rzuceniem wyjątku");
        if (s == null)
            throw new Exception1();
        else if (s == "Rzucam studia")
            throw new Exception2();
        System.out.println("Po rzuceniu wyjątku");
    }

    void obsługujemyWyjatek(String s) {
        try {
            System.out.println("Przed wywołaniem niebezpiecznej funkcji");
            rzucamyWyjatkiem(s);
            System.out.println("Po wywołaniu niebezpiecznej funkcji");

            // W jednej klauzuli catch można przechwycić kilka wyjątków
        } catch (Exception1 || Exception2 e) {
            System.out.println("Obsługujemy wyjątek");
            e.printStackTrace(System.out);
        } finally {
            System.out.println("To wykonuje się zawsze");
        }
    }

    public static void main(String[] args) {
        UczymySieWyjatkow u = new UczymySieWyjatkow();
        System.out.println("Oby się udało");
        u.obiętak(null);
        System.out.println("Udało się!");
    }
}
```

Wykonanie takiego programu wypisze na wyjście:

```
Oby się udało
Przed wywołaniem niebezpiecznej funkcji
Przed rzuceniem wyjątku
Obsługujemy wyjątek
java.lang.Exception
    at test.UczymySieWyjatkow.rzucamyWyjatkiem(UczymySieWyjatkow.java:5)
    at test.UczymySieWyjatkow.obiętak(UczymySieWyjatkow.java:14)
    at test.UczymySieWyjatkow.main(UczymySieWyjatkow.java:28)
To wykonuje się zawsze
Udało się!
```

■ Własne klasy wyjątków

Możemy też definiować własne klasy wyjątków, są to podklasy *Exception* lub *RuntimeException*. Mogą zawierać dodatkowe pola i **koniecznie muszą mieć konstruktor**. Standardowe klasy wyjątków mają przeważnie konstruktor przyjmujący napis (który można potem uzyskać na przykład metodą *getMessage()*). We własnych wyjątkach taki napis ustawiamy w konstruktorze za pomocą słowa kluczowego **super**.

Przykład 2.

Poniżej przedstawiony jest przykład własnej klasy wyjątku *StudentRzuciłStudio* oraz jego użycie:

```
class StudentRzuciłStudio extends Exception {  
    StudentRzuciłStudio(String s) {  
        super(s);  
    }  
  
    public class Student {  
        public static void main(String[] args) {  
            try {  
                throw new StudentRzuciłStudio("Student debil");  
            } catch (StudentRzuciłStudio e) {  
                System.out.println(e.getMessage());  
            }  
        }  
    }  
}
```

Zestaw zadań

11.8. Dany jest program w języku Java:

```
final class A {  
  
    public boolean f(Object x) {  
        return equals(x);  
    }  
  
    public boolean g(Object x) {  
        return x.equals(this);  
    }  
}
```

W takim programie

- A. wywołanie funkcji *f* może zwrócić wyjątek klasy dziedziczącej po klasie *Exception*
- B. wywołanie funkcji *f* zawsze zwraca **false**
- C. wywołanie funkcji *g* może zwrócić wyjątek klasy dziedziczącej po klasie *Exception*

11.9. Rozważamy programy napisane w języku Java. Jeżeli klasa *A* jest nadklassą *B*, to

- A. w treści klasy *B* można zdefiniować metodę o takim samym nagłówku jak metoda zdefiniowana w klasie *A*
- B. jeśli obie klasy *A* i *B* mają po jednym konstruktorze, to w nagłówku konstruktora z klasy *B* należy podać listę wyjątków obejmującą wszystkie wyjątki nadzorowane wymienione w konstruktorze z klasą *A*
- C. w treści klasy *B* można zdefiniować metodę o takim samym nagłówku jak metoda zdefiniowana w klasie *A*, ale z większą widocznością niż była podana w metodzie z klasą *A*

11.10. W języku Java

- A. nie trzeba obsługiwać wszystkich wyjątków kontrolowanych (ang. *checked exceptions*)
- B. w podklasie można nadpisać (ang. *override*) metodę i zadeklarować ją z większą listą wyjątków



- C. implementując interfejs, w którym pewna metoda rzuca listą wyjątków, trzeba zadeklarować tę metodę z tymi wszystkimi wyjątkami (tj. lista rzucanych wyjątków (`throws`) jest dokładnie taka sama)

11.6.

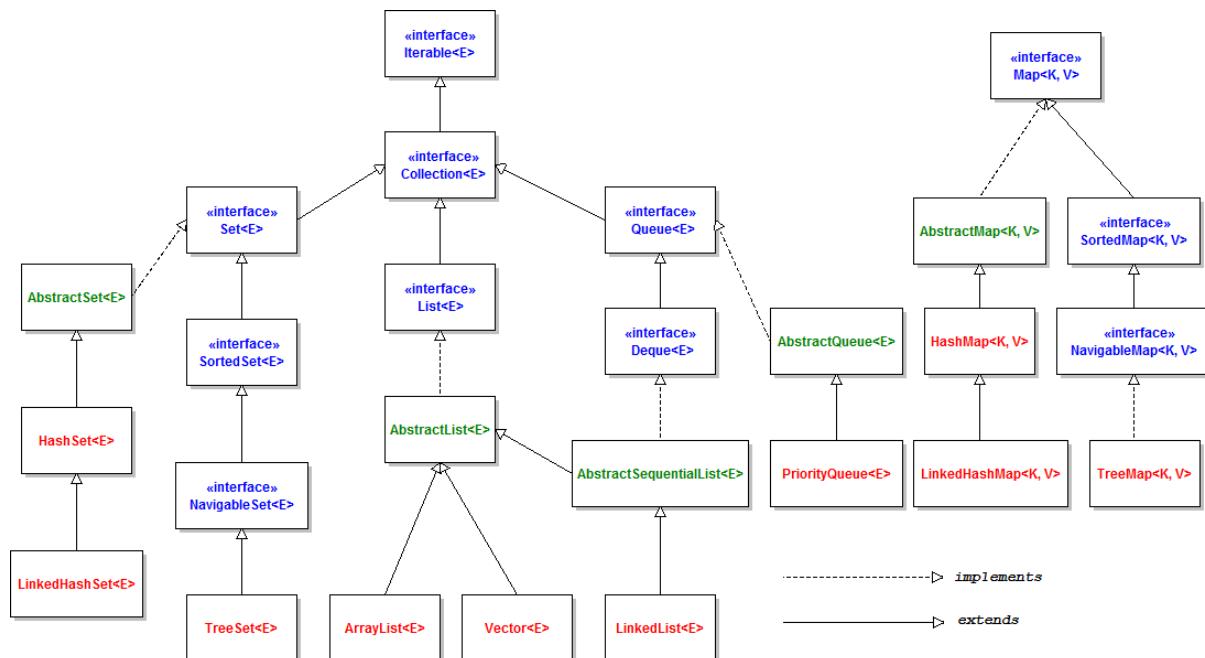
Standardowe kolekcje w Javie

Kolekcja to obiekt:

- służący do przechowywania (innych) obiektów,
- udostępniający mechanizmy pozwalające na wstawianie, przeglądanie i pobieranie przechowywanych obiektów,
- nie mający zadanego specyficznego dla siebie rozmiaru.

Hierarchia dziedziczenia

Kolekcje w standardowej bibliotece Javy implementują różne interfejsy. Poniższy diagram pokazuje hierarchię dziedziczenia interfejsów, klasy abstrakcyjne oraz klasy kolekcji dostępnych w Javie.



Warto zwrócić uwagę: klasy i interfejsy powyżej są generyczne (uogólnione), tzn. kolekcja może być dowolnego typu (<E>).

Najważniejsze metody dostępne w interfejsie kolekcji

```
// Dodaje element do kolekcji (operacja opcjonalna)
boolean add(E e);

// Dodaje wszystkie elementy z danej kolekcji (operacja opcjonalna)
boolean addAll(Collection<? extends E> c);

// Usuwa wszystkie elementy z tej kolekcji (operacja opcjonalna)
void clear();
```

```

// Zwraca true, jeśli ta kolekcja zawiera określony element
boolean contains(Object o);

// Zwraca true, jeśli ta kolekcja zawiera wszystkie elementy z określonej kolekcji
boolean containsAll(Collection<?> c);

// Porównuje określony obiekt z tą kolekcją pod kątem równości
boolean equals(Object o);

// Zwraca true, jeśli ta kolekcja nie zawiera żadnych elementów
boolean isEmpty();

// Zwraca iterator po elementach w tej kolekcji
Iterator<E> iterator();

// Usuwa pojedynczy egzemplarz danego elementu, jeśli istnieje (operacja opcjonalna)
boolean remove(Object o);

// Usuwa wszystkie elementy, które są zawarte w danej kolekcji (operacja opcjonalna)
boolean removeAll(Collection<?> c);

// Zwraca liczbę elementów w kolekcji
int size();

```

11.7. Rozwiązańia

Rozwiązańia

- 11.1.** Mówimy, że widoczność *A* jest szersza niż widoczność *B*, gdy zawsze kiedy widoczne jest *B*, widoczne jest też *A*. Widoczność pakietowa jest szersza niż widoczność

NIE A. publiczna

NIE B. chroniona

TAK C. prywatna

Oznaczmy $A > B$, jeżeli widoczność *A* jest szersza niż widoczność *B*. Z definicji zakresów widoczności wiemy, że zachodzi: publiczna $>$ chroniona $>$ pakietowa $>$ prywatna. Z tego natomiast bezpośrednio wynikają odpowiedzi.

- 11.2.** Dany jest kod w języku Java:

```

class A {
    private int i1;
    protected int i2;
}

class B extends A {
    private int j1;

    public void m(B b) {
        // b.i1 = 0;
        // b.i2 = 0;
        // b.j1 = 0;
    }
}

```

Kod skompiluje się bez błędów po odkomentowaniu linii

- NIE A. b.i1 = 0;
- TAK B. b.i2 = 0;
- TAK C. b.j1 = 0;

- A. Ponieważ pole i1 jest **prywatnym** polem klasy A, nie jest ono dostępne w klasie B.
- B. pole i2 ma widoczność **protected**, czyli jest dostępne również w klasach dziedziczących.
- C. pole j1 jest polem **prywatnym**, ale w klasie B, czyli tej w której aktualnie się znajdujemy.

11.3. Dana jest definicja klasy w języku Java:

```
final class A {  
    static A a;  
    A() { a = this; }  
    static boolean f() { return a.equals(a); }  
    boolean g() { return a == this; }  
    boolean h() { return equals(this); }  
}
```

Dla takiej klasy

- NIE A. każde wywołanie funkcji *f* daje wynik **true**
- NIE B. każde wywołanie funkcji *g* daje wynik **true**
- TAK C. każde wywołanie funkcji *h* daje wynik **true**

- A. Funkcja *f* jest statyczna, zatem można ją wywołać bez wywoływania konstruktora *A*. W takim przypadku *a* będzie **NULL**-em i wywołanie funkcji zakończy się błędem.
- B. Operator == porównuje adresy, które tutaj nie muszą być równe.
- C. Domyślana implementacja *equals()* porównuje adresy, a jako że porównujemy obiekt z samym sobą to zawsze otrzymamy **true**.

11.4. Dany jest program w języku Java:

```
class A {  
    public void m1() { System.out.println("A"); }  
    public void m2(A a) { a.m1(); }  
}  
  
class B extends A {  
    public void m1() { System.out.println("B"); }  
}  
  
class C extends B {  
    public void m1() { System.out.println("C"); }  
    public void m2(A a) { a.m2(this); }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        A a1 = new A();  
        A a2 = new B();  
        A a3 = new C();  
        a3.m1(); // (a)  
        a2.m2(a3); // (b)  
    }  
}
```

```

        a3.m2(a1); // (c)
    }
}

```

Podczas wykonania

- NIE** A. wiersza `a3.m1()`; tego programu zostanie wypisane A
NIE B. wiersza `a2.m2(a3)`; tego programu zostanie wypisane B
TAK C. wiersza `a3.m2(a1)`; tego programu zostanie wypisane C

- A. Zgodnie z typem faktycznym (a nie zadeklarowanym) obiektu `a3`, wywoła się metoda `m1()` zdefiniowana w klasie `C`, więc wypisane zostanie C.
B. Wywoła się metoda `m2()` zdefiniowana w klasie `A` z argumentem klasy `C`, więc wywołanie `a.m1()` spowoduje wypisanie się C.
C. Wywoła się metoda `m2()` zdefiniowana w klasie `C` z argumentem klasy `A`. Następnie `a.m2()` spowoduje wywołanie się metody z klasy `A` z argumentem klasy `C`. To spowoduje wywołanie się metody `m1()` z klasy `C`, więc zostanie wypisane C.

11.5. Rozważamy programy napisane w języku Java. Jeżeli klasa `A` jest nadklassą `B`, to w treści klasy `B`

- NIE** A. trzeba zdefiniować wszystkie metody zdefiniowane w klasie `A`
TAK B. można zdefiniować metody o innych nagłówkach niż metody zdefiniowane w klasie `A`
NIE/TAK C. można zdefiniować metodę o takim samym nagłówku jak metoda zdefiniowana w klasie `A`, ale z większą widocznością niż była podana w metodzie z klasy `A`
- A. Nie trzeba, część metod może być zaimplementowana w klasie `A` lub klasa `B` może być klasą abstrakcyjną.
B. Jak najbardziej można utworzyć własne metody, niezdefiniowane w `A`.
C. Jeżeli autor miał na myśli nadpisanie funkcji, to nie, bo takie można dać jedynie z takim samym nagłówkiem i szerszą widocznością.
W ogólności klasy `A` i `B` mogą należeć do innych pakietów i wtedy jeżeli funkcja w `A` ma widoczność pakietową, to odpowiednia funkcja zdefiniowana w `B` może być prywatna, więc odpowiedź to tak.

11.6. Dany jest program w języku Java:

```

class A {
    public A() { System.out.print("1"); m1(); }
    public A(int i) { System.out.print("2"); }
    public A(A a) { System.out.print("3"); }
    public void m1() { System.out.print("4"); }
}

class B extends A {
    A a;
    public B(A a) { System.out.print("5"); }
    public B() { this(new A()); System.out.print("6"); }
    @Override
    public void m1() { System.out.print("7"); }
}

public class Main {
    public static void main(String[] args) {
        A a = new B();
    }
}

```

Po uruchomieniu tego programu na pewno (oprócz, być może, innych znaków) zostanie wypisana cyfra

TAK A. 1

NIE B. 4

NIE C. 3

Prześledźmy wykonanie kodu krok po kroku:

- Tworzony jest obiekt klasy *B* poprzez wywołanie jego bezargumentowego konstruktora.
- Wywołanie `this(new A())` powoduje utworzenie obiektu *A* z użyciem konstruktora *A(int i)* – wypisana zostanie „2”.
- Następnie wywołujemy konstruktor *B(A a)*. Ponieważ klasa *B* dziedziczy po *A*, a pierwszą linią konstruktora nie jest wywołanie konstruktora nadklasy, to automatycznie wywołany zostanie konstruktor bezargumentowy klasy *A* – wypisana zostanie „1”
- Konstruktor bezargumentowy klasy *A* wywoła metodę *m1* obiektu *B*, który tę metodę nadpisał – wypisana zostanie „7”.
- Dalej, konstruktor *B(A a)* wypisze „5”.
- Ostatecznie, konstruktor bezargumentowy klasy *B* wypisze „6”, co zakończy działanie programu.

Warto zwrócić uwagę na to, że gdyby w funkcji *main()* zamienić deklarację typu obiektu z klasy *A* na *B* (tj. `B b = new B();`), nie zmieniłoby to przebiegu wykonania programu.

11.7. Dany jest program w języku Java:

```
class A {  
    void m1() { System.out.println("1"); }  
    static void m2() { System.out.println("2"); }  
    void m3() { m1(); }  
    void m4() { m2(); }  
}  
  
class B extends A {  
    @Override  
    void m1() { System.out.println("3"); }  
    static void m2() { System.out.println("4"); }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        A a = new B();  
        a.m1();  
        a.m3();  
        a.m4();  
    }  
}
```

W takim programie

NIE A. wywołanie `a.m1()` spowoduje wypisanie jedynki

NIE B. wywołanie `a.m3()` spowoduje wypisanie jedynki

TAK C. wywołanie `a.m4()` spowoduje wypisanie dwójki

Obiekt *a* jest typu *A*, ale do jego stworzenia użyto konstruktora klasy *B*, więc metody zostały odziedziczone.

A. Metoda *m1()* została nadpisana przez podkласę *B*, więc zostanie wypisane 3.

B. Metoda `m3()` wywołuje metodę `m1()`, która została nadpisana przez podkласę `B`, więc zostanie wypisane 3.

C Metoda `m2()` jest statyczna, czyli została ona statycznie powiązana z typem obiektu a podczas komplikacji. Znaczy to, że wypisane zostanie 2.

11.8. Dany jest program w języku Java:

```
final class A {  
  
    public boolean f(Object x) {  
        return equals(x);  
    }  
  
    public boolean g(Object x) {  
        return x.equals(this);  
    }  
}
```

W takim programie

NIE **A.** wywołanie funkcji `f` może zwrócić wyjątek klasy dziedziczącej po klasie `Exception`

NIE **B.** wywołanie funkcji `f` zawsze zwraca `false`

TAK **C.** wywołanie funkcji `g` może zwrócić wyjątek klasy dziedziczącej po klasie `Exception`

A. Nie, ponieważ wykona się funkcja `equals()` na obiekcie klasy `A`. Funkcja ta jest domyślnie zdefiniowana i nie rzuca wyjątkiem podklasy `Exception`.

B. Może zwrócić `true`, na przykład gdy obiektem `x` będzie `this`. Wtedy zostanie wykonane porównanie `this.equals(this)`, które zwraca `true`.

C. Jeżeli obiekt `x` będzie równy `NULL`, to dostaniemy `NullPointerException`, który jest podklassą klasy `RuntimeException`, zatem też i `Exception`.

11.9. Rozważamy programy napisane w języku Java. Jeżeli klasa `A` jest nadklassą `B`, to

TAK **A.** w treści klasy `B` można zdefiniować metodę o takim samym nagłówku jak metoda zdefiniowana w klasie `A`

TAK **B.** jeśli obie klasy `A` i `B` mają po jednym konstruktorze, to w nagłówku konstruktora z klasy `B` należy podać listę wyjątków obejmującą wszystkie wyjątki nadzorowane wymienione w konstruktorze z klasy `A`

NIE/TAK **C.** w treści klasy `B` można zdefiniować metodę o takim samym nagłówku jak metoda zdefiniowana w klasie `A`, ale z większą widocznością niż była podana w metodzie z klasy `A`

A. Można, zostanie ona wtedy nadpisana (jeśli nie jest to metoda prywatna) lub zdefiniowana jako nowa (jeśli jest prywatna).

B. Ponieważ konstruktor podklasy zawiera wywołanie konstruktora nadklasy (jeśli nie jest ono jawnie, kompilator sam „doda” jego wywołanie), konstruktor podklasy musi deklarować te same wyjątki, co konstruktor nadklasy.

C. Patrz zadanie 5.

11.10. W języku Java

NIE **A.** nie trzeba obsługiwać wszystkich wyjątków kontrolowanych (ang. *checked exceptions*)

TAK **B.** w podklasie można nadpisać (ang. *override*) metodę i zadeklarować ją z większą listą wyjątków

NIE **C.** implementując interfejs, w którym pewna metoda rzuca listą wyjątków, trzeba zadeklarować tę metodę z tymi wszystkimi wyjątkami (tj. lista rzucanych wyjątków (`throws`) jest dokładnie taka sama)

- A.** Deklarowanie wyjątku jako wyjątek kontrolowany zobowiązuje nas do obsłużenia go w jakiś sposób – po to go w ogóle tak piszemy.
- B.** Możemy tak zrobić – jedyne co jest ważne to to, żeby ta lista wyjątków była kompatybilna z listą metody z nadklasy, żeby nie było sytuacji, w której nie możemy użyć obiektu podklasy jako obiektu nadklasy (ang. *Liskov substitution*). Z tego powodu nie możemy zadeklarować więcej wyjątków niż było w nadklasie. Możemy zadeklarować mniej, ponieważ nadpisywana metoda, różniąca się od oryginalnej, mógłaby po prostu w kodzie wcale tych wyjątków nie rzucać, wtedy nie ma potrzeby deklarować ich w nagłówku. W przeciwnym wypadku byłibyśmy zmuszeni obsługiwać nierzucone wyjątki, używając obiektu danej klasy wprost (nie jako obiekt nadklasy), co byłoby bardzo niepraktyczne.
- C.** Nie, z tego samego powodu co powyżej. Z praktycznego/logicznego punktu widzenia – nie miałoby sensu wymuszanie tej samej listy wyjątków, bo programista piszący metodę z implementowanego interfejsu wcale nie musiałby napisać kodu, który rzuca każdy wymieniony wyjątek.

12

IPP i blok JNP

Materiały teoretyczne zostały opracowane na podstawie materiałów znalezionych w zakamarkach Internetu oraz czytanek z Moodle'a.

Podstawa programowa

1. Znajomość konstrukcji programistycznych **C** i **C++**.
2. Znajomość metod **zarządzania konfiguracjami i wersjami oprogramowania**.
3. Znajomość technik i narzędzi tworzenia oprogramowania (**linkowanie, debugowanie, profilowanie** itd.)

12.1.

Konstrukcje programistyczne C i C++

Klasy w C++ definiujemy za pomocą słowa kluczowego **class**. Każda klasa może posiadać elementy, takie jak zmienne czy deklaracje funkcji. Dodatkowo, elementy te posiadają **specyfikatory dostępu**:

- **prywatny** (**private**) – elementy klasy są dostępne **z innych elementów tej samej klasy**,
- **chroniony** (**protected**) – elementy są dostępne **z innych elementów tej samej klasy** oraz **z elementów klas dziedziczących**,
- **publiczny** (**public**) – elementy są dostępne **wszędzie**, gdzie obiekt klasy jest dostępny.

Domyślnie zmienne oraz funkcje w ciele klasy są oznaczone jako prywatne.

Konstruktory i destruktory

Każda klasa musi posiadać **konstruktor**, czyli specjalną funkcję automatycznie wołaną podczas tworzenia nowego obiektu tej klasy, pozwalającą na inicjalizację zmiennych. Funkcja ta jest deklarowana jak zwykła funkcja, jednak jej nazwa musi być taka sama jak nazwa klasy oraz **nie może zwracać żadnego typu**.

Przy deklaracji obiektu danej klasy, wołany jest jej **konstruktor domyślny** (ang. *default constructor*), który nie przyjmuje żadnych argumentów. Jeśli konstruktor klasy nie będzie zdefiniowany jawnie, kompilator automatycznie utworzy jej konstruktor domyślny.

Każda klasa ma domyślnie zaimplementowane trzy konstruktory: wspomniany wcześniej *default constructor*, **copy constructor** i **move constructor**. Inaczej niż konstruktor domyślny, *copy constructor* i *move constructor* przyjmują po jednym argumencie, będącym obiektem danej klasy.

Copy constructor kopiuje zawartość obiektu przekazywanego w argumencie, tworząc tym samym nowy obiekt. Natomiast *move constructor* tworzy nowy wskaźnik na obiekt przekazany w argumencie, dzięki czemu nic nie jest kopiowane i nowa pamięć nie jest alokowana.

Podczas tworzenia nowego obiektu możemy również użyć słowa kluczowego **new**. Tworzy on nowy obiekt (wywołując odpowiedni konstruktor) oraz zwraca adres do jego pamięci.

Destruktory pełnią odwrotną rolę do konstruktorów, czyli odpowiadają za wyczyszczenie zasobów potrzebnych przez klasę. Podobnie jak konstruktor, destruktor musi mieć taką samą nazwę jak nazwa klasy, jednak poprzedzamy ją tyldą: ~. Dodatkowo, **nie przyjmuje on żadnych argumentów i nie zwraca żadnego typu**.

Podobnie jak przy konstruktorach, jeśli destruktor nie zostanie jawnie zdefiniowany, kompilator automatycznie utworzy **destruktor domyślny**.

Przykład 1.

Poniżej przedstawiono implementację klasy *C* z domyślnym konstruktorem i destruktorem (wygenerowanymi przez kompilator, więc niewidocznymi w kodzie):

```
class C {
public:
    int x;
};

int main() {
    // Wywołanie konstruktora domyślnego (x nie jest inicjalizowany i zawiera śmieci)
    C c;
    // Brak wywołania konstruktora domyślnego. Korzysta z value initialization
    // x zainicjalizowane na wartość domyślną dla int
    C c1 = C();
    // Korzysta z value initialization jak wyżej, x zainicjalizowane
    C *pc = new C();
    // Korzysta z uniform initialization, x zainicjalizowane
    C c2 = {};
    // To samo co wyżej, wykorzystywane w C++11
    C c3{};

    return 0;
}
```

Przypis redakcji

Czym w powyższym przykładzie są *value initialization* i *uniform initialization* oraz czym się różnią? Wypadałoby to dopisać.

Przykład 2.

Poniżej przedstawiono implementację klasy *Circle* z jawnie zdefiniowanym konstruktorem oraz domyślnym destruktorem (wygenerowanym przez kompilator, więc niewidocznym w kodzie):

```
class Circle {
    double radius;

public:
    Circle(double r) { radius = r; }

int main() {
    Circle foo(10.0);    // Zwykłe wywołanie konstruktora
    Circle bar = 20.0;   // Wywołanie konstruktora (możliwe tylko z jednym argumentem)
```

```
Circle baz{30.0}; // Uniform initialization  
  
return 0;  
}
```

Dziedziczenie

Klasy mogą **dziedziczyć** po innych klasach. Klasa dziedzicząca dziedziczy wtedy po klasie bazowej jej elementy, do których może dodać swoje własne. Dodatkowo, określana jest relacja dziedziczenia za pomocą słów kluczowych:

- **private** – wszystkie elementy są dziedziczone jako prywatne, (**przyp. red.: elementy private nie są dziedziczone w ogóle, więc to wprowadza w błąd**)
- **protected** – wszystkie elementy z dostępem publicznym zostaną dziedziczone z dostępem chronionym (reszta jak w klasie bazowej),
- **public** – wszystkie elementy zostaną dziedziczone z takim samym poziomem dostępnosci, jak w klasie bazowej.

Relacja dziedziczenia dotyczy tylko elementów dziedziczonych (czyli nadal można deklarować publiczne elementy w klasie dziedziczącej z relacją **private**). Przy braku specyfikacji relacji jest ona ustawiana na **private**. Publicznie dziedziczona klasa dziedziczy również dostęp do konstruktorów/destruktorów klasy bazowej. Inaczej niż w Javie, **klasa może dziedziczyć po kilku innych klasach**.

Oczywiście, klasy dziedziczące mogą **nadpisywać** (ang. *override*) elementy z klasy bazowej.

Przykład 3.

W poniższym przykładzie przedstawiono implementację klasy *Employee* oraz klasy po niej dziedziczącej *Programmer* z publiczną relacją dziedziczenia:

```
// Klasa bazowa  
class Employee {  
protected:  
    int salary;  
};  
  
// Klasa dziedzicząca po Employee  
class Programmer: public Employee {  
  
private:  
    int code_lines = 0;  
  
// Wiele elementów o tym samym specyfikatorze dostępu możemy deklarować grupowo  
public:  
    int bonus;  
    void set_salary(int s) {  
        salary = s;  
    }  
    int get_salary() {  
        return salary;  
    }  
    int get_code_lines() {  
        return code_lines;  
    }  
}
```

```
| };
```

■ Wskaźniki

Podobnie jak z innymi typami, możemy odwoływać się do obiektów przez **wskaźniki**. Do elementów klasy odwołujemy się wtedy nie przez kropkę, a przez operator `->`. Przypomnienie operatorów:

- `*x` – wskaźnik na `x`,
- `&x` – adres `x`,
- `x.y` – `y` to element obiektu `x`,
- `x->y` – `y` to element obiektu wskazującego na `x`,
- `(*x).y` – `y` to element obiektu wskazującego na `x`.

Przykład 4.

Przykład pokazuje implementację klasy `Rectangle` oraz różne metody tworzenia jej instancji za pomocą wskaźników:

```
class Rectangle {  
    int width, height;  
public:  
    Rectangle(int x, int y) : width(x), height(y) {}  
    int area(void) { return width * height; }  
};  
  
int main() {  
    Rectangle obj (3, 4);  
    Rectangle *foo, *bar;  
    foo = &obj;  
    bar = new Rectangle (5, 6);  
    cout << "obj's area: " << obj.area() << '\n';  
    cout << "*foo's area: " << foo->area() << '\n';  
    cout << "*bar's area: " << bar->area() << '\n';  
    return 0;  
}  
// obj's area: 12  
// *foo's area: 12  
// *bar's area: 30
```

■ Metody wirtualne

W klasie możemy zdefiniować **wirtualne metody** za pomocą słowa kluczowego **virtual**. Metody te są zdefiniowane w klasie bazowej i mogą zostać przeddefiniowane w klasie dziedziczącej. Mogą one wtedy opcjonalnie używać słowa kluczowego **override**(uniemożliwia ono skompilowanie kodu, w którym metoda klasy pochodnej nie posiadała analogicznej metody wirtualnej w klasie bazowej).

Zasady metod wirtualnych:

- nie mogą być statyczne,
- są związywane podczas wykonania programu, a nie przy jego komplikacji,

- dostęp do nich powinien się odbywać przez wskaźnik lub referencję do klasy bazowej,
- metody wirtualne zapewniają poprawność funkcji wołanej przez obiekt, niezależnie od typu wskaźnika/referencji użytego w trakcie wywołania,
- nagłówek funkcji w klasie bazowej i dziedziczącej musi być ten sam,
- klasa dziedzicząca nie musi nadpisywać metod wirtualnych z klasą bazową (wtedy są one takie jak w klasie bazowej),
- klasa może mieć wirtualny destruktor, ale nie może mieć wirtualnego konstruktora,
- klasa powinna mieć wirtualny destruktor, jeśli będziemy potencjalnie chcieli usunąć instancję klasy dziedziczącej przez wskaźnik do klasy bazowej (na przykład gdy klasa bazowa ma metody wirtualne).

Przykład 5.

Przykład poniżej pokazuje różnicę pomiędzy metodami wirtualnymi (*print()*) a metodami zwykłymi (*show()*):

```
class Base {
public:
    virtual void print() { cout << "print base class\n"; }

    void show() { cout << "show base class\n"; }
};

class Derived : public Base {
public:
    void print() { cout << "print derived class\n"; }

    void show() { cout << "show derived class\n"; }
};

int main() {
    Base* base;
    Derived d;
    base = &d;

    base->print(); // Wirtualna funkcja, związana podczas runtime
    base->show(); // Niewirtualna funkcja, związana podczas kompilacji

    return 0;
}
// print derived class
// show base class
```

To było na egzaminie

Dany jest następujący kod w C++:

```
class A {
public:
    void m1() { std::cout << "A\n"; }
    virtual void m2() { std::cout << "A\n"; }
    void m3() { std::cout << "A\n"; }
};

class B: public A {
public:
```

```

void m1() { std::cout << "B\n"; }
virtual void m2() { std::cout << "B\n"; }
virtual void m3() { std::cout << "B\n"; }
};

int main() {
    A* p = new B();
    // p->m1();
    // p->m2();
    // p->m3();
}

```

Po odkomentowaniu następującej linijki na standardowe wyjście zostanie wypisana litera „B”:

- A. p->m1();
- B. p->m2();
- C. p->m3();

- A. Ponieważ metoda m1() nie jest metodą wirtualną w klasie A, to jest ona wiązana w trakcie komplikacji według typu zadeklarowanego. Zmienna p jest typu A*, zatem zostanie wywołana metoda z klasy A, wypisującą literę „A”.
- B. Z zasad metod wirtualnych, metoda m2() zostanie związana w trakcie wykonywania programu, a zatem zostanie ona wywołana z klasy B, wypisując literę „B”.
- C. Podobnie jak w podpunkcie A., metoda m3() w klasie A nie jest wirtualna, zatem zostanie ona wywołana z klasy A (jako związana w czasie komplikacji), wypisując „A”.

Klasy, które deklarują lub dziedziczą metody wirtualne, są nazywane **klasami polimorficznymi**.

Oprócz tego mamy jeszcze **abstrakcyjne klasy bazowe**, które mogą być używane tylko jako klasy bazowe i mogą zawierać niezdefiniowane metody wirtualne. Nie można stworzyć obiektu takich klas, ale można tworzyć wskaźniki na tę klasę. Klasy dziedziczące muszą implementować niezdefiniowane metody wirtualne z klasą bazową.

Przykład 6.

Przykład pokazuje abstrakcyjną klasę bazową *Rectangle* z niezdefiniowaną metodą wirtualną *area()*:

```

class Rectangle {
protected:
    int width, height;
public:
    void set_values (int a, int b)
    { width = a; height = b; }
    virtual int area () = 0; // metoda wirtualna bez definicji
};

```

Zestaw zadań

12.1. Dany jest kod w C++:

```
| #include <iostream>
```

```

using namespace std;

class A {
public:
    int i = 0;
    virtual int m() { return i; }
};

class B : public A {
public:
    int i = 1;
    int m() override { return i; }
};

int main() {
    A a;
    B b;
    A &c = b;
    // cout << a.i;
    // cout << a.m();
    // cout << c.m();
}

```

Na wyjście zostanie wypisane 0 po uprzednim odkomentowaniu linii

- A. cout << a.i;
- B. cout << a.m();
- C. cout << c.m();

12.2. Dany jest program w C++:

```

#include <iostream>
using namespace std;
class A {
public:
    void m1() { cout << 'A'; }
    virtual void m2() { cout << 'B'; }
    virtual void m3() { cout << 'C'; }
};

class B: public A {
public:
    void m1() { cout << 'D'; }
    void m2() { cout << 'E'; }
};

class C: public B {
public:
    void m3() { cout << 'F'; }
};

int main() {
    A* a = new B();
    a->m1();
    a->m2();
    a->m3();
}

```

Wówczas

- A. wywołanie metody `a->m1()` spowoduje wypisanie litery „A”
- B. wywołanie metody `a->m2()` spowoduje wypisanie litery „B”
- C. wywołanie metody `a->m3()` spowoduje wypisanie litery „C”

12.3. Dany jest fragment programu w C++:

```
class A {  
private:  
    int i;  
public:  
    A() { i = 13; }  
};  
  
class B: public A {  
private:  
    int j;  
};
```

W podanym fragmencie programu

- A. klasa B ma konstruktor bezparametryowy
- B. klasa B ma konstruktor bezparametryowy, odziedziczony po klasie A
- C. po utworzeniu obiektu klasy B jego składowa `i` będzie miała wartość 13

12.4. W języku C++

- A. zaleca się, by destruktor w klasach mających metody wirtualne był wirtualny
- B. wszystkie metody domyślnie są wirtualne
- C. przeddefiniowanie (tzn. zdefiniowane z dokładnie takim samym nagłówkiem) w podklasie metody zdefiniowanej w nadklasie jako niewirtualna jest zabronione

12.5. Dany jest kod w języku C++ w wersji 11:

```
#include <iostream>  
  
class A {  
public:  
    virtual void m() { std::cout << "A\n"; }  
};  
  
class B : public A {  
public:  
    void m() override { std::cout << "B\n"; }  
};  
  
int main() {  
    // tutaj wstawiamy kod  
}
```

Na wyjście zostanie wypisana pojedyncza litera „A” po wstawieniu do funkcji `main()` kodu

- A. `A a = new B(); a.m();`
- B. `A* a { new B(); }; a->m();`
- C. `A a { B(); }; a.m();`

12.6. Dany jest program w C++:

```
#include <iostream>

class A {
public:
    virtual void g() {
        std::cout << "A";
    }
};

class C : public A {
public:
    void g() {
        std::cout << "C";
    }
};

int main() {
    A* p = new C();
    p->g();
}
```

W podanym programie

- A. inicializacja zmiennej p zostanie odrzucona przez kompilator
- B. wywołanie metody p->g() spowoduje wypisanie „C”
- C. wywołanie metody p->g() spowoduje wypisanie „AC”

12.2.

Znajomość technik i narzędzi tworzenia oprogramowania

Istnieje wiele narzędzi używanych przy pracy z C/C++. Są to oczywiście na przykład narzędzia służące do debugowania i profilowania kodu, jednakże ze względu na niskopoziomową i komplikowaną naturę tych języków w ten skład wchodzą również kompilatory/linkery, narzędzia sprawdzające poprawność zarządzania pamięcią przez program i tym podobne.

■ Linkowanie

Zadaniem **linkera** jest łączenie wielu plików obiektowych (z rozszerzeniem .o), aby utworzyć plik wykonywalny. Jest to konieczne, ponieważ kompilatory C/C++ działają tak, że kompilują każdy plik .c jako odzielną jednostkę, tworząc takie niedopieczone pliki .o, którymi dopiero później zajmie się linker. Linker jest potrzebny, ponieważ pliki .o *by design* są nieświadome informacji, które nie zostały wspomniane w ich kodzie źródłowym. Do tych informacji należą na przykład odwołania do funkcji zdefiniowanych w innej jednostce komplikacji. Rozważmy na przykład plik źródłowy:

```
extern void f();

int main() {
    f();
    return 0;
}
```

Wygenerowany z tego pliku plik obiektowy będzie wiedział jedynie, że ma oczekiwana definicję funkcji f()

w jakiejś innej jednostce kompilacji. Linker, mając parę tych plików, będzie mógł poinformować ten pierwszy, gdzie jest adres treści funkcji `f()` i uzupełnić wywołanie funkcji w kodzie asemblerowym.

Najpopularniejszym linkerem używanym na przykład przez `gcc` jest `ld`, który jest używany niejawnie przy wywołaniu `gcc`, żeby skompilować dane źródła kodu. Linkerów (w szczególności `ld`) raczej nie używa się bezpośrednio i jest to z reguły wszczepione w proces kompilacji, tak jak opisano powyżej w przypadku `gcc`.

■ Debugowanie

Narzędzia do **debugowania** pozwalają łatwo wykrywać błędy w kodzie, na przykład poprzez możliwość ustalania breakpointów, wykonywania programu po jednej instrukcji, oglądania zawartości pamięci w danym momencie wykonania programu itp. Tego typu funkcjonalności udostępnia oprogramowanie `gdb`, które jest najczęściej używanym debuggerem programów C/C++.

Jedną z funkcji, które udostępnia `gdb`, jest możliwość uruchomienia programu i sprawdzenia, gdzie się wywalił. Przykład użycia:

```
\$ gdb ./example
GNU gdb (GDB) Fedora (7.3.50.20110722-13.fc16)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /path/example...done.
(gdb) run
Starting program: /path/example

Program received signal SIGSEGV, Segmentation fault.
0x0000000000400527 in foo_len (s=0x0) at example.c:7
7          return strlen (s);
```

■ Inne narzędzia

Inne warte wspomnienia narzędzia, które pomagają w tworzeniu bezpiecznego i optymalnego kodu, to na przykład Valgrind, który służy do zapewniania, że program poprawnie zarządza pamięcią i nie powoduje wycieków pamięci. Inną grupą narzędzi są te służące do **profilowania**, tj. badania, jakie kawałki kodu zabierają najwięcej czasu podczas wykonania. Przykładami takich programów są np. Callgrind (program bliźniaczy do Valgrind), a także perf, często domyślnie dostępny na platformach Linux.

12.3.

Rozwiązańia

Rozwiązańia

12.1. Dany jest kod w C++:

```
#include <iostream>
using namespace std;

class A {
public:
    int i = 0;
```

```

    virtual int m() { return i; }

};

class B : public A {
public:
    int i = 1;
    int m() override { return i; }
};

int main() {
    A a;
    B b;
    A &c = b;
    // cout << a.i;
    // cout << a.m();
    // cout << c.m();
}

```

Na wyjście zostanie wypisane 0 po uprzednim odkomentowaniu linii

- TAK A. cout << a.i;
- TAK B. cout << a.m();
- NIE C. cout << c.m();

W pierwszych dwóch podpunktach obiekt klasy A zachowuje się zgodnie z oczekiwaniami. Jeśli chodzi o c.m(), to należy pamiętać, że metody wirtualne wiązane są podczas wykonania, więc brany pod uwagę jest prawdziwy typ obiektu b (klasa B), a nie deklarowany (klasa A). Wypisane zostanie więc 1.

12.2. Dany jest program w C++:

```

#include <iostream>
using namespace std;
class A {
public:
    void m1() { cout << 'A'; }
    virtual void m2() { cout << 'B'; }
    virtual void m3() { cout << 'C'; }
};

class B: public A {
public:
    void m1() { cout << 'D'; }
    void m2() { cout << 'E'; }
};

class C: public B {
public:
    void m3() { cout << 'F'; }
};

int main() {
    A* a = new B();
    a->m1();
    a->m2();
    a->m3();
}

```

Wówczas

- TAK A. wywołanie metody `a->m1()` spowoduje wypisanie litery „A”
- NIE B. wywołanie metody `a->m2()` spowoduje wypisanie litery „B”
- TAK C. wywołanie metody `a->m3()` spowoduje wypisanie litery „C”

Zmienna `a` jest wskaźnikiem na typ `A`, ale został użyty konstruktor podklasy `B`, więc odziedziczone zostaną metody wirtualne. Oznacza to, że wywołanie `a->m1()` wypisze „A”, ponieważ jest to metoda klasy `A`. Natomiast metoda `m2()` jest wirtualna, więc jej definicja została nadpisana przez podkласę `B` i wypisane zostanie „E”. Klasa `B` nie nadpisuje jednak definicji metody `m3()`, więc podczas jej wywołania zostanie wypisane „C”.

12.3. Dany jest fragment programu w C++:

```
class A {  
private:  
    int i;  
public:  
    A() { i = 13; }  
};  
  
class B: public A {  
private:  
    int j;  
};
```

W podanym fragmencie programu

- TAK A. klasa `B` ma konstruktor bezparametryowy
- TAK B. klasa `B` ma konstruktor bezparametryowy, odziedziczony po klasie `A`
- TAK C. po utworzeniu obiektu klasy `B` jego składowa `i` będzie miała wartość 13

A. Konstruktor bezparametryowy jest zawsze domyślnie generowany przez klasę, chyba że jawnie go usuniemy.

B. Podklasa dziedzicząca publicznie po nadklasie dziedziczy wszystkie jej konstruktory.

C. Tak, bo klasa `B` dziedziczy po klasie `A` zmienną `i` oraz konstruktor ustawiający jej wartość na 13.

12.4. W języku C++

- TAK A. zaleca się, by destruktor w klasach mających metody wirtualne był wirtualny
- NIE B. wszystkie metody domyślnie są wirtualne
- NIE C. przedefiniowanie (tzn. zdefiniowane z dokładnie takim samym nagłówkiem) w podklasie metody zdefiniowanej w nadklasie jako niewirtualna jest zabronione

A. Tak, jest to zalecane.

B. Metody niewirtualne są niewirtualne.

C. Nie jest, wystarczy rozejrzeć się po pozostałych zadaniach i w większości tak jest robione. Różnica polega na tym, że tak przedefiniowana metoda nie jest dziedziczona, tylko po prostu nadpisywana. Zauważmy też, że nagłówek w klasie bazowej i klasie dziedziczącej musi być ten sam w przypadku metody wirtualnej (w zadaniu mowa o metodzie niewirtualnej).

12.5. Dany jest kod w języku C++ w wersji 11:

```

#include <iostream>

class A {
public:
    virtual void m() { std::cout << "A\n"; }
};

class B : public A {
public:
    void m() override { std::cout << "B\n"; }
};

int main() {
    // tutaj wstawiamy kod
}

```

Na wyjście zostanie wypisana pojedyncza litera „A” po wstawieniu do funkcji `main()` kodu

- NIE** A. A a = `new B()`; a.m();
NIE B. A* a { `new B()` }; a->m();
TAK C. A a { B() }; a.m();

- A. Ponieważ operator `new` zwraca adres, nie możemy go przypisać na zmienną typu `A`, zatem kod ten w ogóle się nie skompiluje.
- B. Zmienna `a` będzie wskazywać na nowy obiekt typu `B`. Z zasad metod wirtualnych, metoda `m()` zostanie wywołana z klasy `B`.
- C. Ze względu na to, że zmienna `a` nie jest ani wskaźnikiem, ani referencją oraz ze względu na zasady metod wirtualnych, metoda `m()` zostanie wywołana z klasy `A`.

12.6. Dany jest program w C++:

```

#include <iostream>

class A {
public :
    virtual void g() {
        std::cout << "A";
    }
};

class C : public A {
public :
    void g() {
        std::cout << "C";
    }
};

int main() {
    A* p = new C();
    p->g();
}

```

W podanym programie

- NIE** A.inicjalizacja zmiennej `p` zostanie odrzucona przez kompilator
TAK B. wywołanie metody `p->g()` spowoduje wypisanie „C”

NIE C. wywołanie metody `p->g()` spowoduje wypisanie „AC”

- (a) Operator `new` zwraca adres nowo zaalokowanej pamięci dla danego obiektu. Przypisanie wskaźnikowi pewnego adresu jest w pełni poprawne, zatem kompilator nie wypisze żadnego błędu.
- (b) Ze względu na zasady metod wirtualnych, funkcja `g()` zostanie wywołana z klasy C, wypisując „C”.
- (c) Patrz: podpunkt B.

13

Aplikacje WWW

Materiały teoretyczne zostały opracowane na podstawie materiałów Łukasza Sznuka – [kurs na Moodle](#).

Podstawa programowa

1. Rodzina **protokołów HTTP**.
2. Mechanizmy tworzenia stron internetowych: **HTML, CSS**.
3. Język **JavaScript** oraz jego unikalne cechy. Języki kompilowane na JavaScript.
4. **Mechanizmy budowania aplikacji internetowych**: ciasteczka, żądania, routing, widoki, mapowanie obiektowo-relacyjne.
5. **Bezpieczeństwo aplikacji webowych**.

13.1. HTML i CSS

HTML to język opisu strony, a **CSS** to język opisu jej wyglądu.

■ HTML

Dokument HTML składa się ze **znaczników** (inaczej nazywanych tagami lub elementami HTML) i można go podzielić na dwie główne części: **nagłówek** (*head*), który zawiera metadane dokumentu (można w nim używać ograniczonego zestawu znaczników) oraz **treść strony** (*body*).

Przykład 1.

Poniżej przedstawiono podstawowy szkielet prostej strony internetowej:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Treść nagłówka -->
    <title>Tytuł</title>
  </head>
  <body>
    <!-- Znacznik otwierający - div -->
    <!-- Atrybuty - lang, class, id -->
    <!-- Wartości atrybutów - en, c1 c2, adiv -->
    <div lang="en" class='c1 c2' id=adiv>
      Treść.
    <!-- Znacznik zamykający - div -->
```

```
</div>

<!-- Znacznik otwierający bez znacznika zamykającego -->

</body>
</html>
```

Nagłówek może zawierać między innymi:

```
<!-- Tytuł -->
<title>Tytuł</title>

<!-- System kodowania -->
<meta charset="utf8">

<!-- Bazowy URL -->
<base href="https://mimuw.edu.pl/" target="_blank">

<!-- Zewnętrzne linki -->
<link rel="stylesheet" href="style.css">

<!-- Link do skryptu lub jego kod -->
<script src="kod.js"></script>
<script> console.log("!"); </script>

<!-- Kod CSS dotyczący stylu -->
<style> a {color: green;} </style>
```

Treść strony zawiera między innymi znaczniki, które na stronie wyświetlają się zwykle jako prostokątne pudelka zawierające coś wewnątrz siebie. Tymi znacznikami są:

- <body> – cała treść strony
- <header> – przeważnie jakaś wstępna treść lub linki, znajduje się na górze strony
- <footer> – stopka znajdująca się na dole strony
- <main> – główna treść
- <nav> – linki nawigacyjne
- <article> – niezależna treść
- <aside> – treść znajdująca się przeważnie po boku strony, oprócz głównej treści
- <section> – sekcja dokumentu
- <div> – używany, gdy żaden z powyższych nie pasuje do znajdującej się w znaczniku treści

Oprócz tego, w treści strony używane są takie znaczniki jak:

- nagłówki: <h1>, <h2>, ..., <h6>
- akapity: <p>
- listy: (nieuporządkowana), (uporządkowana), (elementy listy)
- wyróżnienie tekstu: , <i>, , <u>
- pusty wiersz:

- fragment: <**span**>
- hiperłącza: <**a href="link"**>
- obrazki: <**img src="link"**>

CSS

CSS można dodać do dokumentu HTML na kilka sposobów: w oddzielnym pliku (dodajemy link do pliku stylu w nagłówku), w treści strony (umieszczamy kod stylu w nagłówku) lub w docelowym znaczniku.

Selektory służą do wyboru węzłów w drzewie DOM, do których stosuje się deklaracje stylu (wypisane wewnątrz nawiasów klamrowych). Przykładowe selektory i ich zasięg:

- * { } – selektor uniwersalny, pasuje do wszystkiego
- **div** { } – selektor znacznika
- .**class_name** { } – selektor klasy
- [**atribute**] { } – selektor atrybutu
- #**id** { } – selektor id
- **Łączenie selektorów** (wszystkie połączone selektory muszą występować w jednym znaczniku) – piszemy selektory po sobie, bez spacji (np. połączenie selektora klasy i id to .**class_name#id** { })
- **div p** { } – selektor potomka (styl obejmuje wszystkie **p** będące potomkiem **div**)
- **div > p** { } – selektor bezpośredniego potomka (dziecka)
- **h1 + p** { } – selektor następnego sąsiada (**h1** i **p** są na tym samym poziomie i **p** występuje bezpośrednio po **h1**)
- **h1 ~ p** { } – selektor dalszych sąsiadów (wszystkie **p** będące na tym samym poziomie co **h1** i występujące po **h1**)

Wartości niektórych właściwości stylu są **dzierdziczone** od rodzica w drzewie DOM. Na przykład, dziedziczone są: **color**, **font**, **text-align**, a nie są **width** czy **display**.

Przykład 2.

Poniższy przykład przedstawia zastosowanie stylu CSS w kodzie HTML:

```
<style>

/* Pseudoelementy ::first-letter i ::after */
p::first-letter {
    color: orange;
}

div::after {
    display: block;
    background-color: green;
    border: 4px solid black;
}

/* Pseudoklasa :nth-child */
ul > li:nth-child(2n) {
    background-color: yellow;
}


```

```


- ul > li:nth-child(2n+1) {
    background-color: lightblue;
}



#id_name { color: green; }



.class_name #id_name { color: black; }


```

</style>

<p>

!-- Pierwsza litera tekstu będzie pomarańczowa -->

Jakiś tam tekst.

</p>

<div>

!-- Po tekście pojawi się prostokąt z czarną obwódką i zielonym tłem -->

Kolejny tekst.

</div>

!-- Nieparzyste elementy będą zakolorowane na niebiesko, a parzyste na żółto -->

Element 1

Element 2

Element 3

Element 4

!-- Ze względu na specyficzność selektorów, tekst zostanie wyświetlony na czarno -->

<div class="class_name">

<p id="id_name"> Tekst </p>

</div>

Specyficzność selektorów (priorytetowość) wylicza się jako $A.B.C$, gdzie:

- A = liczba selektorów id w selektorze,
- B = liczba selektorów klas, atrybutów i pseudoklas,
- C = liczba selektorów znaczników i pseudoelementów.

Następnie specyficzności (postaci $A.B.C$) są porównywane leksykograficznie. Im większa specyficzność selektora, tym ma on większy priorytet. Jest to szczególnie ważne w przypadku, gdy do danego znacznika pasują dwa selektory - wtedy ten z większym priorytetem jest stosowany. Można też założyć, że im bardziej szczegółowy jest specyfikator, tym ma on większą specyficzność, jednak przy bardziej skomplikowanych przypadkach lepiej jest policzyć specyficzności ze wzoru i je porównać.

Ponadto:

- jeżeli styl jest określony bezpośrednio w znaczniku, to przyjmuje on najwyższy priorytet, niezależnie od innych określonych stylów,
- selektory, które są dopasowane bezpośrednio do danego znacznika mają wyższy priorytet nad tymi, które są odziedziczone po przodku.

To było na egzaminie

Dany jest kod HTML:

```

<style>
    div > p { color: yellow; }
    div * p { color: red; }

```

```

body p.div { color: green; }
p#foo { color: green; }
.x#foo { color: black; }
</style>

<body>
  <div class="x">
    <p id="foo"> XXX </p>
    <p> YYY </p>
  </div>
</body>

```

Po jego wyrenderowaniu

- A. „XXX” będzie zielone
- B. „YYY” będzie czerwone
- C. „YYY” będzie żółte

Policzmy najpierw specyficzność dla każdego selektora:

- div > p - $a = 0, b = 0, c = 2$
- div * p - $a = 0, b = 0, c = 2$
- body p.div - $a = 0, b = 1, c = 2$
- p#foo - $a = 1, b = 0, c = 1$
- .x#foo - $a = 1, b = 1, c = 0$

Teraz zastanówmy się jakie selektory mogłyby być zaaplikowane do „XXX”. Będzie to div > p (ponieważ znajduje się w znaczniku p będącym bezpośrednim potomkiem znacznika div) oraz p#foo (ponieważ znajduje się w znaczniku p posiadającym id="foo"). Patrzmy teraz na specyficzność: pierwszy selektor ma 0.0.2, a drugi 1.0.1, zatem większą specyficzność ma drugi selektor i to on będzie zastosowany. Zatem „XXX” będzie zielone.

Następnie zbadajmy możliwe selektory dla „YYY”. Jest to tylko div > p (ponieważ znajduje się w znaczniku p będącym bezpośrednim potomkiem znacznika div), zatem to on zostanie wybrany i „YYY” będzie żółte.

Zestaw zadań

13.1. Dla danego stylu (nie są stosowane żadne inne style):

```

div {color: yellow;}
div p {color: red;}
div.p {color: green;}
p#id {color: green;}
.x#id {color: black;}

```

i fragmentu HTML

```

<body>
  <div class="x">
    <p id="id"> XXX </p>
    <p> YYY </p>

```

```
    zzz  
  </div>  
</body>
```

- A. napis „XXX” ma kolor zielony
- B. napis „YYY” ma kolor czerwony
- C. napis „ZZZ” ma kolor żółty

13.2. Dla fragmentu HTML:

```
<!DOCTYPE html>  
<html>  
<head><title>t</title>  
<style>  
  .foo {color: red;}  
  #foo {color: blue;}  
  #main p {color: green;}  
</style>  
</head>  
<body>  
  <div id="main">  
    <p id="foo" class="bar">  
      C  
      <span class="foo"> A </span>  
      B  
    </p>  
  </div>  
</body>  
</html>
```

- A. litera „A” jest czerwona
- B. litera „B” jest niebieska
- C. litera „C” jest zielona

13.2.

JavaScript

JavaScript jest językiem obiektowym. Oprócz wartości prymitywnych, **wszystko jest obiektem**. Do **wartości prymitywnych** zaliczamy:

- liczby,
- napisy,
- wartości boolowskie,
- wartość specjalna **null**,
- wartość specjalna **undefined**,
- tzw. symbole.

Przypis redakcji

Dobrze byłoby dopisać przykłady do konkretnych wartości prymitywnych z listy powyżej.

Czym są „tzw. symbole” z listy powyżej? Nie jest to oczywiste.

Na **obiekty** można patrzeć w pewnym sensie jak na mapy – są to po prostu zbiory par (klucz, wartość), do których swobodnie można dopisywać kolejne pary:

```
let obj = { a: 123 };
console.log(obj.a); // 123
obj.b = "abc";
console.log(obj.b); // "abc"
```

Na boku: powyższy sposób przypisywania wartości pod danym kluczem do obiektów to coś, co odróżnia zwykłe obiekty od prymitywów.

```
let str = "abc";
str.a = 1;
console.log(str.a); // undefined (nie zaszło przypisanie)
```

Składnia JavaScript

Ważne elementy składni:

- Deklaracja zmiennych:

```
let a = 1; // brak podanego typu!
a = "abc"; // można zmienić na wartość innego typu
const stała = 123;
stała = true; // błąd
var b = 2; // podobne do let, ale już raczej nieużywane (patrz: domknięcia)
```

Uwaga na temat var. Zmienne deklarowane za pomocą var mają albo scope globalny, albo funkcyjny. To znaczy, że jeśli taka zmienna jest zadeklarowana **gdziekolwiek poza ciałem jakiejś funkcji**, to jest też widoczna wszędzie dalej. W dodatku nie jest widoczna w ciele żadnej funkcji, nawet zadeklarowanej później w kodzie, która gdzieś w swoim ciele deklaruje zmienną o tej samej nazwie. Może to być bardzo dziwne, ale dzieje się tak przez tzw. *hoisting* – deklaracje var są przetwarzane przed wykonaniem jakiegokolwiek kodu i traktowane są tak, jakby były na samej górze kodu (lub ciała funkcji). Obrazują to przykłady:

```
a = 2;
console.log(a); // 2
var a;
console.log(a); // 2

var a = 1;

function f() {
    console.log(a);
    var a = 2;
    console.log(a);
    a = 3;
    console.log(a);
}

const g = () => {
    console.log(a);
    if (true) {
```

```

        var a = 3;
    }
    console.log(a);
}

let h = () => {
    console.log(a);
    a = 5;
    console.log(a);
}

// wypisane zostanie:
console.log(a); // 1
f(); // undefined, 2, 3
console.log(a); // 1
g(); // undefined, 3
console.log(a); // 1
h(); // 1, 5
console.log(a); // 5

```

Przykład z funkcją f może być zaskakujący, bo hoisting na pierwszy rzut oka powinien wyciągnąć deklarację var a = 2; na początek ciała funkcji. Jednak należy pamiętać, że taka linijka kodu jest jedynie **lukrem syntaktycznym** na deklarację i operację przypisania: var a; a = 2;. W związku z tym linijka var a = 2; zostaje przekrojona na pół i w miejscu zostaje samo przypisanie, a deklaracja jest wyniesiona na początek.

- **Obiekty:**

```

let obj = { a: 1, b: "abc" };
obj.a; // 1
obj[b]; // "abc"
obj.c; // undefined

```

- **Listy:**

```
| let list = [1, "abc", true, { a: 1 }]; // można mieszać typy
```

- **Koercja typów:**

```

let a = 1 + "2"; // "12", dodanie różnych typów konwertuje argumenty na stringa
let b = "22" - 2; // 20, operator "-" konwertuje oba argumenty na liczby
let c = +"1"; // 1, unary plus konwertuje na liczbę
let d = 1 == "1"; // true, zwykłe == konwertuje na wspólny typ (nieprzewidywalne)
let e = 1 === "1" // false, brak konwersji z ===

let obj1 = {a: 1};
let obj2 = {a: 1};
obj1 == obj2; // false, porównywanie obiektów jak w Javie, po referencjach
obj1 === obj2; // false, jak wyżej

```

- **Funkcje:**

```

function f(a, b, c) {
    let d = a + b + c;
    return d;
}

// lambda
let g = (a, b) => a + b;

// lambda z ciałem

```

```
const h = a => {
    a++;
    console.log(a);
}
```

- **Pętla for-each:**

```
const list = [1, 2, 17, 42];
for (const v of list) {
    console.log(v); // 1 2 17 42 (wartości)
}

for (const k in list) {
    console.log(k); // 0 1 2 3 (indeksy)
}

const obj = {foo: 1, bar: "abc"};
for (const k in obj) {
    console.log(k); // foo bar
}
```

Uwaga 1: `for ... in` iteruje się po wszystkich kluczach w obiekcie. Można w ten sposób zaobserwować, że listy to tak naprawdę po prostu obiekty ze specyficznymi atrybutami: obiekt `list` to prawie to samo co `{ 0: 1, 1: 2, 2: 17, 3: 42 }`.

Uwaga 2: `for ... of` nie chodzi po prostu po wszystkich wartościach w obiekcie. Jest to specjalny typ iteracji wymagający, aby obiekt był „iterowalny” (ang. *iterable*), co spełniają np. listy. Próba przeiterowania się w ten sposób po `obj` wyrzuci błąd.

■ Prototypy w JavaScript

Obiekty, oprócz wspomnianych własności mapy, posiadają również tzw. **prototyp**, czyli w pewnym sensie obiekt-ojca, po którym dziedziczy. System prototypów ustawia wszystkie obiekty w JavaScriptie w jedno wielkie drzewo, w którym na ścieżce od danego wierzchołka aż do korzenia (w którym przebywa specjalny obiekt `Object.prototype`), leżą kolejne prototypy danego obiektu.

```
let str = new String("123");
let strProto = str.__proto__;
console.log(strProto); // specjalny obiekt String.prototype
let strProto2 = strProto.__proto__;
console.log(strProto2); // specjalny obiekt Object.prototype
let strProto3 = strProto2.__proto__;
console.log(strProto3); // null
```

Pytając o wartość w obiekcie `obj` pod jakimś kluczem najpierw sprawdzane jest, czy sam `obj` zawiera wartość pod tym kluczem. Jeśli jej tam nie ma, to wdrudzamy w górę łańcucha prototypów odpytując po kolei kolejne obiekty na tej ścieżce. Jeśli w żadnym z tych obiektów nie istnieje wartość pod tym kluczem, zwracane jest `undefined`.

```
let str = new String("123");
let strProto = str.__proto__;
strProto.a = 2;
console.log(str.a); // 2
str.a = 1;
console.log(str.a); // 1
```

■ Domknięcia w JavaScript

Definiowanie funkcji w JavaScript tworzy tzw. **domknięcie**, czyli zbiór referencji do zmiennych zadeklarowanych w momencie jej utworzenia. Odwołania do zmiennych w ciele tworzonej funkcji to odwołania do

zmiennych z domknięcia, a nie kopie ich wartości z momentu utworzenia funkcji, co może powodować niespodziewane zachowania:

```
let l = [];
let a;

for(a = 0; a < 3; a++) {
    l.push(() => a);
}

l[0](); // ???
```

Trzeba zauważyć, że nie są tutaj tworzone funkcje `() => 0`, `() => 1` i `() => 2`, tylko trzy takie same funkcje zwracające wartość zmiennej `a`, pobieranej z domknięcia. Można o tym myśleć tak, że przy tworzeniu funkcji jej ciało zapamiętywane jest tak, jak jest napisane w kodzie, a nie przetwarzane w jakiś sposób, jak np. w C/C++. Ponieważ na koniec pętli wartość zmiennej `a` wynosi 3, to wszystkie trzy funkcje w liście zwrócią 3, ponieważ taka jest wartość zmiennej `a` w momencie wywołania funkcji `l[0]`.

Żeby „naprawić” powyższy kod, wystarczy dorzucić deklarację `let` do nagłówka pętli `for`. Co ciekawe, poniższy kod również wypisze 3:

```
let l = [];

for(var a = 0; a < 3; a++) {
    l.push(() => a);
}

l[0](); // ???
```

Jest to związane z tym, że zmienne `var` deklarują zmienną widoczną globalnie, lub w zakresie funkcji. Ponieważ tutaj jest ona zadeklarowana w miejscu, którego nie otacza ciało żadnej funkcji, to tak naprawdę jest zmienną globalną, a nie lokalną ze względu na pętlę `for`. Ten problem rozwiązuje deklaracja `let`, która ma zakres blokowy, ponadto w przypadku pętli `for` zmienna jest osobna dla każdego obrotu pętli.

■ Ważne funkcje

```
let list = [1, 2, 3, 4, 5];
let sum = list.reduce((x, a) => x + a, 0); // 15
let list2 = list.map(x => x * 2); // [2, 4, 6, 8, 10]
let list3 = list.filter(x => x < 3); // [1, 2]
let list4 = list.slice(1, 4); // [2, 3, 4]
list.push(6); // [1, 2, 3, 4, 5, 6]
list.pop(); // [1, 2, 3, 4, 5], usunięto szóstkę
```

Zestaw zadań

13.3. Dany jest kod w języku JavaScript:

```
let tab = [1, 2, 3, 4];
tab.s = function() { return this.reduce((s, a) => s + a, 0) }
let t = "";
```

Wartość `t` będzie równa "1234" (bez cudzysłowu) po wykonaniu fragmentu kodu

- A. `for (let i in tab) t = t + tab[i];`
- B. `for (let e in tab) t = t + e;`
- C. `for (let e of tab) t = t + e;`

13.4. Domknięcia w języku JavaScript:

- A. pozwalają emulować zmienne prywatne
- B. służą do realizacji mechanizmu wyjątków
- C. można zaimplementować w CSS

13.5. Dany jest kod w JavaScriptie:

```
var a = 1;
function f() {
    console.log(a);
    var a = 2;
    console.log(a);
}
f();
console.log(a);
```

Po jego wykonaniu

- A. w pierwszej linii wyjścia pojawi się wartość undefined
- B. w drugiej linii wyjścia pojawi się wartość 2
- C. w trzeciej linii wyjścia pojawi się wartość 2

13.3.

Mechanizmy budowania aplikacji internetowych

■ Node.js i NPM

Definicje

- **Node.js** - asynchroniczne, sterowane zdarzeniami środowisko uruchomieniowe JavaScript zaprojektowane do tworzenia skalowalnych aplikacji sieciowych
- **NPM (Node Package Manager)** - narzędzie wykorzystywane w ekosystemie Node.js do zarządzania zależnościami oraz udostępniania pakietów oprogramowania. NPM pozwala programistom na łatwe pobieranie, instalowanie, aktualizowanie i usuwanie pakietów, które są potrzebne do budowy aplikacji w oparciu o Node.js.
- **Eslint** - narzędzie do statycznej analizy kodu JavaScript, które pomaga w wykrywaniu potencjalnych błędów, zgodności ze standardami kodowania i ogólnych problemów w kodzie. (Instalowane przy użyciu **NPM**)

Najważniejsze komendy

```
//uruchomienie aplikacji Node.js
node plik.js
//inicjalizacja nowego projektu Node.js
npm init
//instalacja pakietu i zapisanie go do pliku konfiguracyjnego
npm install nazwa-pakietu --save-dev
//uruchomienie lintera: Eslint
npx eslint plik.js
```

Plik konfiguracyjny npm

W pliku **package.json** zawiera się informacje o projekcie oraz definiuje jego zależności, skrypty do uruchamiania i inne metadane (również zasady, według których działa Eslint). Skrypty można zdefiniować:

```
{  
  // ...  
  "scripts": {  
    "hello": "echo 'Hello, World!'"  
  }  
  // ...  
}
```

a następnie uruchomić tak zdefiniowany skrypt polecienniem:

```
| npm run hello
```

Przydaje się to w momencie uruchamiania: budowania pakietu, testów, czy serwera.

■ HTTP

Definicja

HTTP (Hypertext Transfer Protocol) - protokół komunikacyjny używany w sieciach komputerowych do przesyłania danych pomiędzy klientami a serwerami. Jest to protokół warstwy aplikacji, który opiera się na modelu żądanie-odpowiedź. ([Szczegóły na SIK-u](#))

Przykład 1.

Request:

```
GET / HTTP/1.1  
Host: www.mimuw.edu.pl  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/*,  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate, br  
DNT: 1  
Connection: keep-alive  
Cookie: SSESSb8b5556cce9d79f51b870d9e111=FQ8qYP4giEMjUH1nFpmUckpZUeUuSONhas_js=1
```

- **Metoda:** GET
Określa, że żądanie ma na celu pobranie zasobu.
- **Ścieżka:** /
Określa ścieżkę lub adres URL zasobu, który jest żądany. W tym przypadku jest to główna strona.
- **Wersja protokołu:** HTTP/1.1
Wersja protokołu HTTP używana w żądaniu.
- **Host:** www.mimuw.edu.pl
Określa nazwę hosta, do którego jest kierowane żądanie. W tym przypadku jest to "www.mimuw.edu.pl".
- **User-Agent:** Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0
Informuje serwer o typie i wersji przeglądarki lub aplikacji klienckiej używanej do wysłania żądania.
- **Accept:** text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/*,...
Informuje serwer o typach mediów, jakie klient akceptuje w odpowiedzi. W tym przypadku wskazane są preferowane typy takie jak "text/html", "application/xhtml+xml", "application/xml" itp.

- **Accept-Language:** en-US,en;q=0.5

Informuje serwer o preferowanym języku odpowiedzi. W tym przypadku preferowany jest język angielski (en-US) z drugorzędnym priorytetem dla ogólnego języka angielskiego (en).

- **Accept-Encoding:** gzip, deflate, br

Informuje serwer o preferowanych algorytmach kompresji, które klient akceptuje w odpowiedzi. W tym przypadku preferowane są algorytmy kompresji Gzip, Deflate i Brotli.

- **DNT:** 1

Do Not Track (DNT) to nagłówek, który informuje serwer, czy klient chce być śledzony w celach reklamowych lub statystycznych. Wartość "1" oznacza, że klient nie chce być śledzony.

- **Connection:** keep-alive

Informuje serwer, że klient chce utrzymać połączenie aktywne, aby możliwe było ponowne użycie go w przyszłości.

- **Cookie:** SSESSb8b5556cce9d79ff1fef51be111=FQ8qYP4giEMjUH1nFpmUckpZUeUuSONhas_js=1

Zawiera ciasteczka (cookies), które klient wysyła w żądaniu. W tym przypadku przekazywane są konkretne ciasteczka, które zostały wcześniej zapisane przez serwer.

Przykład 2.

Odpowiedź:

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 30 Mar 2022 10:05:52 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Expires: Sun, 19 Nov 1978 05:00:00 GMT
...
```

Najważniejszy jest **status odpowiedzi** w pierwszej linijce. Dostępne statusy odpowiedzi przy HTTP to:

- **1xx** – Informacyjne
- **2xx** – Sukces
- **3xx** – Przekierowanie
- **4xx** – Błąd klienta
- **5xx** – Błąd serwera

Metody dostępne w HTTP wykorzystywane chociażby przy pisaniu API:

- **GET:** Metoda GET służy do pobierania lub pobrania reprezentacji zasobu.
- **POST:** Metoda POST służy do przesyłania danych do przetworzenia przez zasób.
- **PUT:** Metoda PUT służy do aktualizacji lub zamiany zasobu danymi przekazanymi w żądaniu.
- **PATCH:** Metoda PATCH służy do częściowej aktualizacji zasobu.
- **DELETE:** Metoda DELETE służy do usuwania określonego zasobu.
- **HEAD:** Metoda HEAD jest podobna do metody GET, ale pobiera tylko nagłówki odpowiedzi, bez ciała odpowiedzi.
- **OPTIONS:** Metoda OPTIONS służy do żądania obsługiwanych metod i możliwości serwera dla danego zasobu.

■ Schemat adresów: URI

URI = scheme ":" hier-part ["?" query] ["#" fragment]

- **scheme:** Oznacza protokół komunikacyjny używany do dostępu do zasobu. Np. "http", "https", "ftp"
- **hier-part:** Jest to hierarchiczna część URI, która może zawierać informacje dotyczące lokalizacji zasobu. Zazwyczaj obejmuje nazwę hosta, ścieżkę do zasobu i inne komponenty zależne od schematu.
- **query:** Jest to opcjonalna część URI, która zawiera dane przekazywane do zasobu w formie parametrów. Zazwyczaj jest to zestaw klucz-wartość, gdzie klucz i wartość są oddzielone znakiem równości "=" i parametry są oddzielone znakiem "&". Część query rozpoczyna się po znaku "?".
- **fragment:** Jest to opcjonalna część URI, która wskazuje na określoną część zasobu. Zazwyczaj jest to odniesienie do konkretnej sekcji lub fragmentu w dokumencie.

Przykład 3.

http://example.com:8042/over/there?name=ferret#nose
| \ / | \ _____ / \ _____ / \ _____ / \ _____ / \ _____ /
scheme authority path query fragment

■ Express

Przykład 4.

Najprostszy serwer w Node.js

```
import * as http from 'http';
const requestListener = function (req, res) {
  res.writeHead(200);
  res.end('Hello, World!');
}
const server = http.createServer(requestListener);
server.listen(8080)
```

Jednak obsłużenie wielu ścieżek jest problematyczne, ale z pomocą przychodzi nam **Express**.

Definicja

Express - minimalistyczny, lekki framework dla Node.js, który ułatwia tworzenie aplikacji sieciowych i API. Oparty na Node.js i zapewniający prostą ale potężną warstwę abstrakcji dla:

- obsługi żądań HTTP,
- zarządzania trasami (routingu),
- renderowania widoków.

Prosta aplikacja Express

```
import express from 'express';

let app = express();
app.get('/', (req, res) => {
  res.status(200).send('<!doctype html><title>Strona.</title>')
});
app.listen(8080);
```

W tak zdefiniowanej aplikacji, serwer otrzymujący żądanie, które nie jest obsługiwane, zwraca odpowiedź ze statusem 404 - Not Found.

Routing

Routing to mechanizm, który umożliwia mapowanie żądań HTTP na odpowiednie funkcje obsługujące te żądania. Pozwala to na definiowanie różnych ścieżek (URL) i określanie, jak serwer powinien odpowiedzieć na żądania wysłane do tych ścieżek.

Definicja trasy (route) w Express: używamy metody odpowiadającej dla danego typu żądania HTTP, np. app.get() dla żądań GET, app.post() dla żądań POST itp. Następnie podajemy ścieżkę (URL) jako pierwszy argument, a jako drugi argument podajemy funkcję obsługującą żądanie dla tej ścieżki. Ta funkcja przyjmuje obiekty req (request) i res (response), które reprezentują żądanie klienta i odpowiedź serwera.

Dopasowywanie funkcji: wywoływana jest pierwsza funkcja, której parametr pasuje do URL z żądania. Zwykle piszemy po jednej funkcji dla danej ścieżki, ale można użyć dodatkowego parametru **next**, żeby wywołać kolejną pasującą funkcję:

```
app.get('/', (req, res, next) => {
  res.status(200).send('<title>Strona.</title>tak.');
  next();
};

app.get('/', (req, res, next) => {
  console.log('Kolejna pasująca funkcja');
});
```

Zaawansowane dopasowywanie ścieżek - **Parametry w ścieżkach**, czyli dynamiczne dopasowywanie ścieżek:

- ":" pozwala na użycie dynamicznych identyfikatorów w ścieżce. Dla '/users/:id' w ciele funkcji możemy się odwoać do podanego id poprzez **const** userId = req.params.id; Czyli dla żądania "/users/dynamiczneId" pole userId będzie równe "dynamiczneId".
- Podobnie można zrobić bardziej wyszukane dopasowanie, przy użyciu wyrażeń regularnych. Przykładowo "/^\\users\\/(\\[a-z\\d\\]+)\\\$/i", które dopasowuje ścieżkę /users/username tylko dla użytkowników o nazwach składających się z małych liter i cyfr. W ciele funkcji można się następnie odwoać do takiego id przy użyciu: **const** username = req.params[0];

Middleware

Może się zdarzyć sytuacja, w której chcielibyśmy zalogować informacje o obsłudze każdego z żądań. Można pisać kod w każdej funkcji obsługującej żądanie, ale to niewygodne - z pomocą przychodzi nam Middleware. Definiuje się je za pomocą **app.use()**, wtedy middleware aplikowany jest do każdego żądania.

Przykład 5.

```
app.use((req, res, next) => {
  console.log('Obsługuję ' + req.path);
  next();
});
```

Jednak można też przy użyciu `app.get()`. Przykładowo jeśli chcemy dopasować wszystkie żądania, które zawierają "users".

Przykład 6.

```
app.get('/users', (req, res, next) => {
  console.log('Middleware dla ścieżki "/users"');
  // Przekazanie żądania do kolejnego middleware lub docelowej funkcji obsługującej
  next();
});
```

Ponadto można je wykorzystywać do:

- analizy ciasteczek,
- sprawdzania praw dostępu,
- tworzenia sesji,
- i wiele innych...

Zestaw zadań

13.6. Routing w aplikacjach webowych polega między innymi na

- A. uruchamianiu różnych funkcji w zależności od adresu URL
- B. przesyłaniu pakietów przez odpowiednie routery
- C. kierowaniu zapytań do różnych serwerów bazodanowych (np. w celu rozłożenia obciążenia)

13.4.

Django

Przypis redakcji

Rozdział nie jest stworzony – w roku akademickim, w którym powstało repetytorium, Django nie wchodziło w zakres materiału Aplikacji WWW. W archiwalnych egzaminach licencjackich pojawiło się jednak zadanie na jego temat, stąd decyzja o pozostawieniu tej sekcji do ewentualnego jej napisania.

Zestaw zadań

13.7. Django

- A. udostępnia swoje usługi za pomocą protokołu HTTP
- B. jest rozszerzeniem JavaScriptu
- C. zawiera warstwę odwzorowania relacyjno-obiektowego

13.5.

Rozwiązańia

Rozwiązańia

13.1. Dla danego stylu (nie są stosowane żadne inne style):

```
div {color: yellow;}  
div p {color: red;}  
div.p {color: green;}  
p#id {color: green;}  
.x#id {color: black;}
```

i fragmentu HTML

```
<body>  
  <div class="x">  
    <p id="id"> XXX </p>  
    <p> YYY </p>  
    ZZZ  
  </div>  
</body>
```

- A. napis „XXX” ma kolor zielony
- B. napis „YYY” ma kolor czerwony
- C. napis „ZZZ” ma kolor żółty

Na początek zauważmy, że selektor **div.p** nie ma zastosowania w naszym kodzie, ponieważ nie mamy żadnego znacznika **<div>**, który posiadałby klasę o nazwie p. Podobnie jest z selektorem **.x#id**, ponieważ nie mamy żadnego znacznika z klasą o nazwie x oraz id o nazwie id.

- A. Do napisu „XXX” pasują kolory żółty, czerwony oraz zielony (z selektora **p#id**). Zostanie wybrany selektor **p#id**, ponieważ ma największą specyficzność (czyli jest „najdokładniejszy”).
- B. Do napisu „YYY” pasują kolory żółty i czerwony. Wybrany zostanie kolor czerwony, ponieważ jego selektor ma większą specyficzność.
- C. Do napisu „ZZZ” pasuje tylko kolor żółty, zatem to on zostanie wybrany.

13.2. Dla fragmentu HTML:

```
<!DOCTYPE html>  
<html>  
  <head><title>t</title>  
  <style>  
    .foo {color: red;}  
    #foo {color: blue;}  
    #main p {color: green;}  
  </style>  
</head>
```

```

<body>
  <div id="main">
    <p id="foo" class="bar">
      C
      <span class="foo"> A </span>
      B
    </p>
  </div>
</body>
</html>

```

TAK A. litera „A” jest czerwona

NIE B. litera „B” jest niebieska

TAK C. litera „C” jest zielona

Litery „B” i „C” będą zielone, ponieważ są wewnątrz znacznika `<div>`, którego id to `main`, i w akapicie `<p>`. Zgodnie z zasadami specyficzności selektorów, jest to bardziej dokładne niż bycie tylko w akapicie o id `foo`. Litera „A” jest bezpośrednio wewnątrz znacznika `` z klasą `foo`, więc będzie miała czerwony kolor (selektor `.foo` dotyczy bezpośrednio tego elementu, więc ma wyższy priorytet niż `#main p`, który jest odziedziczony po rodzicu).

13.3. Dany jest kod w języku JavaScript:

```

let tab = [1, 2, 3, 4];
tab.s = function() { return this.reduce((s, a) => s + a, 0) }
let t = "";

```

Wartość `t` będzie równa "1234" (bez cudzysłowu) po wykonaniu fragmentu kodu

NIE A. `for (let i in tab) t = t + tab[i];`

NIE B. `for (let e in tab) t = t + e;`

TAK C. `for (let e of tab) t = t + e;`

Kod w podpunkcie **A.** przeiteruje się po każdym kluczu w obiekcie `tab` i doklei do napisu `t` wartość pod tym kluczem. Wynikiem nie będzie tu napis "1234", ponieważ istnieje też klucz "s". Po tym fragmencie kodu w "`t`" będzie napis "1234function() { return this.reduce((s, a) => s + a, 0) }".

Kod w podpunkcie **B.** z pewnością nie da w wyniku "1234", ponieważ dokleja do `t` nazwy kluczy, czyli da w wyniku "0123s".

Kod z podpunktu **C.** spowoduje ustawnienie `t` na napis "1234". Należy pamiętać, że pętla `for ... of` nie iteruje się po prostu po wartościach kluczy, a wykonuje specjalną iterację zdefiniowaną przez dany obiekt. Dla list pętla ta iteruje się „klasycznie” – po prostu po elementach listy. Ponieważ atrybut pod kluczem "s" nie jest elementem listy, to nie będzie uwzględniony podczas iteracji.

13.4. Domknięcia w języku JavaScript:

TAK A. pozwalają emulować zmienne prywatne

NIE B. służą do realizacji mechanizmu wyjątków

NIE C. można zaimplementować w CSS

W podpunkcie **A.** emulację obrazować może przykładowy kod:

```

var myBankAccount = (function(){
  var balance = 0;

  return {

```

```
    getBalance: () => balance
  }
})()
```

Rozgryźmy powyższy kod. Na początek widzimy funkcję, w której deklarowana jest zmienna `balance`, po czym zwracany jest obiekt, w którym jedynym atrybutem jest funkcja `getBalance()` zwracająca wartość zmiennej `balance`. Funkcja ta jest anonimowa i natychmiastowo wywoływana. W wyniku tego, do zmiennej `myBankAccount` przypisany zostaje wyżej wspomniany obiekt z funkcją zwracającą wartość `balance` z domknięcia. Ponadto, nie ma żadnego sposobu na dostanie się do tej zmiennej oprócz otrzymania jej wartości poprzez ww. funkcję.

W podpunkcie **B.** odpowiedzią jest **NIE**, bo wyjątki są zrealizowane natywnie i bynajmniej nie korzystają z domknięć.

W podpunkcie **C.** oczywiście odpowiedzią jest **NIE**, ponieważ domknąć nie ma w CSS.

13.5. Dany jest kod w JavaScriptie:

```
var a = 1;
function f() {
  console.log(a);
  var a = 2;
  console.log(a);
}
f();
console.log(a);
```

Po jego wykonaniu

- TAK** **A.** w pierwszej linii wyjścia pojawi się wartość `undefined`
- TAK** **B.** w drugiej linii wyjścia pojawi się wartość 2
- NIE** **C.** w trzeciej linii wyjścia pojawi się wartość 2

W pierwszej linii wyjścia pojawi się `undefined`, ponieważ zmienna `a` została zadeklarowana w ciele funkcji `f()`, ale zainicjowana dopiero po pierwszym `console.log(a)`.

W drugiej linii wyjścia pojawi się 2, ponieważ zmienna `a` została już zainicjowana.

W trzeciej linii wyjścia pojawi się 1, a nie 2 – pomimo że przypisanie wewnątrz funkcji `f()` zachodziło względem zmiennej o nazwie `a`, to była to wersja lokalna dla ciała tej funkcji, która jedyne co miała wspólnego z wersją zadeklarowaną globalnie to nazwa. W związku z tym, ponieważ jesteśmy poza ciałem funkcji `f`, zmienna `a` nadal ma wartość 1.

13.6. Routing w aplikacjach webowych polega między innymi na

- TAK** **A.** uruchamianiu różnych funkcji w zależności od adresu URL
- NIE** **B.** przesyłaniu pakietów przez odpowiednie routery
- NIE** **C.** kierowaniu zapytań do różnych serwerów bazodanowych (np. w celu rozłożenia obciążenia)

Z definicji routingu w aplikacjach webowych – jest to mechanizm, który umożliwia mapowanie żądań HTTP na odpowiednie funkcje obsługujące te żądania. Wynika stąd, że ani podpunkt **B.**, ani podpunkt **C.** nie są prawdziwe. Zachodzi jedynie **A.**

13.7. Django

- TAK** **A.** udostępnia swoje usługi za pomocą protokołu HTTP
- NIE** **B.** jest rozszerzeniem JavaScriptu
- TAK** **C.** zawiera warstwę odwzorowania relacyjno-objektowego

14

Sieci komputerowe

Materiały teoretyczne zostały opracowane na podstawie slajdów Agaty Janowskiej.

Podstawa programowa

1. **Warstwy sieci.**
2. **Protokoły** TCP, UDP, IP, ICMP, Ethernet.
3. **Adresy internetowe**, tablice tras, zasady trasowania, NAT.
4. Systemy **nazw domenowych**.
5. Sieciowy **interfejs gniazd**.

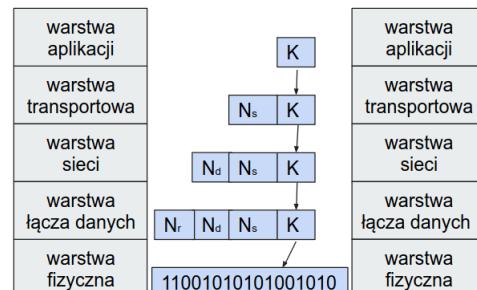
14.1.

Warstwy sieci

Sieć możemy podzielić na warstwy, z których każda operuje na coraz niższym poziomie abstrakcji. Zalety takiego podziału to m.in.:

- łatwiejsze definiowanie składników dużego systemu
- wyższa warstwa korzysta z usług oferowanych przez niższą warstwę
- modularność ułatwia zmiany
- w skład warstwy wchodzą protokoły wraz z urządzeniami i oprogramowaniem
- stos protokołów - kombinacja protokołów różnych warstw

Kapsułkowanie



K - komunikat, N_s - nagłówek segmentu
N_d - nagłówek datagramu, N_r - nagłówek ramki

Rysunek 14.1: Wizualizacja przekazywania komunikatów do kolejnych warstw

■ Warstwa aplikacji

Aplikacje sieciowe to programy uruchamiane na różnych systemach końcowych, komunikujące się ze sobą za pośrednictwem sieci np. przeglądarka internetowa z serwerem WWW. Wyróżniamy dwa główne podejścia:

1. Architektura klient-serwer

- serwer obsługuje żądania klientów
- klienci nie komunikują się bezpośrednio ze sobą
- najczęściej serwer posiada ogólnie znany adres IP i jest cały czas dostępny
- przykłady: poczta elektroniczna, przeglądarki, zdalne logowanie
- zamiast pojedynczego serwera mogą być grupy serwerów, centra danych np. Amazon

2. Architektura P2P (ang. Peer-to-Peer)

- sieć równorzędna, bez pośrednictwa serwera
- aplikacje wykorzystują bezpośrednie połączenia między parami komunikujących się hostów nazywanych węzłami
- węzłami są systemy końcowe użytkowników
- sieci dystrybucji plików (np. BitTorrent), systemy wideokonferencji (np. Skype)
- samoskalowalne, tanie
- niski poziom bezpieczeństwa, niezawodności, wydajności

Protokół warstwy aplikacji określa w jaki sposób procesy wymieniają się komunikatami, czyli definiuje:

- typy komunikatów
- składnię różnego typu komunikatów
- semantykę, czyli znaczenie poszczególnych pól komunikatu
- zasady mówiące kiedy i w jaki sposób proces wysyła komunikaty i na nie odpowiada

Główny protokół: **HTTP** stanowiący podstawowy składnik technologii WWW. W warstwie transportowej korzysta z protokołu TCP.

■ Warstwa transportowa

Odpowiada za przekazanie komunikatów między procesami tworzącymi aplikację w sposób **połączeniowy** (TCP) lub **bezpołączeniowy** (UDP). Warstwa transportowa po stronie nadawcy dzieli komunikaty otrzymane od aplikacji na mniejsze fragmenty i dodaje do nich nagłówki warstwy transportowej tworząc **segmenty**. Następnie

- segment przekazywany jest warstwie sieci;
- warstwa sieci kapsułkuje segment we własnym pakiecie (datagramie) i stara się go dostarczyć do odbiorcy;
- po stronie odbiorcy warstwa sieci wyodrębnia z datagramu segment i przekazuje go warstwie transportowej;
- warstwa transportowa przetwarza segment i przekazuje dane aplikacji odbiorczej.

Zadania warstwy transportowej:

- multipleksowanie w warstwie transportowej – tworzenie segmentów i przekazanie ich warstwie sieci
- demultipleksowanie w warstwie transportowej – dostarczenie danych zawartych w segmentach do odpowiedniego procesu

- kontrola błędów (choć w UDP tylko trochę)
- zapewnienie niezawodnej usługi transferu danych (tylko TCP)
- kontrola przepływu (tylko TCP)
- kontrola przeciążenia (tylko TCP)

Warstwa sieci

Główne cechy:

- wykorzystanie protokołu IP (Internet Protocol)
- cel: niegwarantowany (ang. best-effort), bezpołączeniowy transfer danych
 - adresacja (IPv4, IPv6)
 - trasowanie
- protokoły DHCP, ICMP
- NAT
- przekazywanie **datagramów** między dowolnymi hostami

Warstwa łącza danych

Główne cechy:

- przekazywanie danych pojedynczym łączem między **sąsiednimi** węzłami
 - **ramkowanie**
 - dostęp do łącza
 - detekcja i usuwanie błędów (np. suma kontrolna, CRC)
- implementacja może być
 - sprzętowa na karcie sieciowej
 - programowa

Typy kanałów:

- kanały rozgłaszenia
 - wiele hostów podłączonych do tego samego kanału komunikacyjnego
 - np. bezprzewodowe sieci lokalne, satelitarne, hybrydowe używające światłowodu i kabla koncentrycznego
 - konieczny protokół wielodostępu do nośnika
- łącza punkt-punkt
 - na przykład między dwoma routerami powiązanymi łączem czy komputerem i przełącznikiem (ang. switch)

Adresy MAC:

- każdy interfejs posiada adres MAC (ang. Media Access Control) (uwaga: przełączniki nie mają adresów MAC)
- 48-bitowy, wyrażany w notacji szesnastkowej z dwukropkami lub myślnikami

- adresy MAC zaprojektowano jako stałe, ale obecnie można je zmienić programowo
- unikalny, organizacja IEEE zarządza przestrzenią adresów, przydziela producentom kart sieciowych bloki liczące 2^{24} adresów
- płaska struktura, nie zmienia się wraz ze zmianą lokalizacji

Przesyłanie ramek:

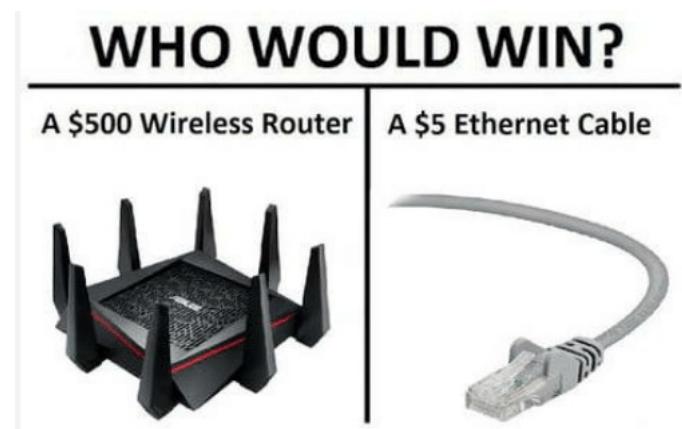
- nadawca umieszcza w ramce adres odbiorcy i przesyła ją do łącza
- kiedy karta sieciowa innego hosta otrzymuje ramkę, która nie jest do niej adresowana, odrzuca ją
- jeśli docelowy adres MAC pasuje do adresu MAC karty sieciowej, z ramki wyodrębniany jest datagram, który jest przekazywany w górę stosu protokołów
- karta sieciowa wpływa na pracę hosta, tylko jeżeli otrzyma adresowaną do niego ramkę
- adres rozgłoszeniowy FF:FF:FF:FF:FF:FF "pasuje" do każdego adresu MAC
- nadawca poznaje adres MAC odbiorcy poprzez **protokół ARP**

■ Warstwa fizyczna

Fizyczne nośniki:

- skrótka nieekranowa
 - najtańszy i najbardziej popularny nośnik
 - dwa izolowane przewody miedziane, skrócone, aby zlikwidować zakłócenia elektryczne
- światłowód
 - cienki, elastyczny nośnik przenoszący impulsy świetlne
 - stosowany tam, gdzie nie można zastosować skrętki (np. większe odległości)
 - odporny na zakłócenie elektromagnetyczne, znakome tłumienie sygnału

Ethernet - najpopularniejsza technologia sieci "przewodowej" opisana w szczegółach niżej.



Rysunek 14.2: Graficzne przedstawienie genetycznej dominacji potężnego kabla Ethernet

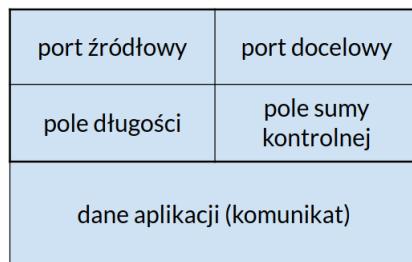
14.2.

Protokoły TCP, UDP, IP, ICMP, Ethernet

■ UDP

Protokołowi temu przyświeca motto: "być może nie wszystkie dane przeżyją, ale jest to poświęcenie, na które jestem gotów". Główne cechy:

- oferuje minimalną funkcjonalność jaką musi zapewnić warstwa transportowa: multipleksowanie i demultipleksowanie
- nie zapewnia niezawodności poza podstawową kontrolą błędów
- jest bezpołączeniowy
 - odbiera komunikat od procesu aplikacji
 - dołącza numery portów źródłowego i docelowego
 - dodaje dwa niewielkie pola i przekazuje warstwie sieci
 - po stronie odbiorcy wykorzystuje numer portu, aby odnaleźć odpowiedni proces aplikacji



Rysunek 14.3: Protokół UDP, struktura segmentu

■ TCP

Zadaniem TCP jest zapewnienie **niezawodnego** transferu danych. W tym celu wykorzystuje on poniższe triki:

- suma kontrolna: wykrywanie uszkodzeń pakietu
- mechanizm potwierdzeń: pozytywne (wszystko doszło, jest okej) i negatywne (coś poszło źle)
- jeśli dane uszkodzone – retransmisja
- jeśli potwierdzenie uszkodzone – retransmisja
- właściwa kolejność danych – numery sekwencyjne
- wykrywanie **utraty** pakietu – potrzebne zegary, oczekiwanie na potwierdzenie przez określony czas (czyli większy niż RTT: ang. Round-Trip Time)
- wykrycie utraty pakietu oczywiście powoduje retransmisję
- aby nadawca nie musiał czekać po wysłaniu każdego pakietu – **potokowanie**, czyli wysyłanie wielu pakietów bez oczekiwania na potwierdzenie, konieczne buforowanie

Zauważmy, że retransmisja eliminuje efekty przeciążenia, ale nie jego przyczynę.

Główne cechy:

- multipleksowanie i demultipleksowanie

- zorientowany na połączenie – zanim zostaną przesłane jakiekolwiek dane, musi zostać przeprowadzony proces negocjacji (SYN, ACK)
- umożliwia komunikację w obydwie strony (ang. full duplex)
- niezawodny transfer danych (dzięki elementom przedstawionym powyżej)
- kontrola przepływu – dostosowanie przepustowości transmisji do możliwości odbiorcy oraz sieci (więc np. dba, by nie zapchał się bufor odbiorcy)
- kontrola przeciążenia – przeciążenie sieci pojawia się kiedy zbyt wiele nadawców wysyła dane z nadmierną szybkością i np. bufory routera się zapychają, dlatego należy ograniczyć szybkość nadawcy w przypadku wykrycia przeciążenia sieci
- może wykorzystywać algorytm Nagle'a – łączenie kilku małych komunikatów i wysyłanie ich w jednym segmencie (minimalizacja liczby wysyłanych segmentów)



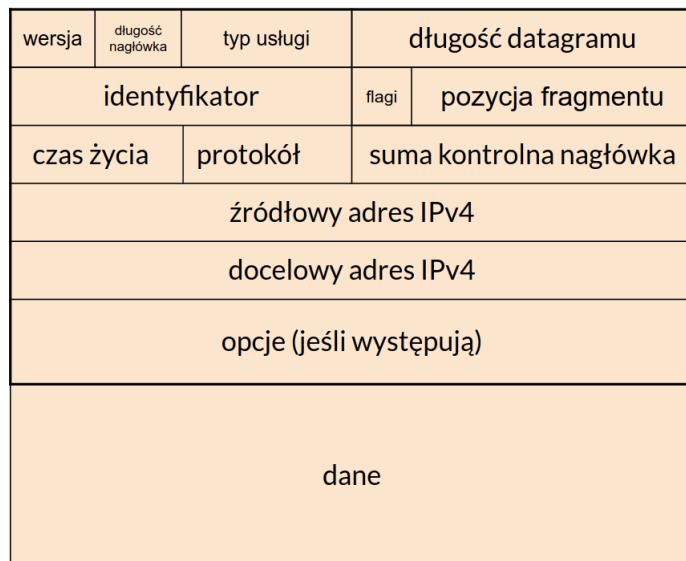
Rysunek 14.4: Protokół TCP, struktura segmentu

Zastosowanie	Przykładowy protokół warstwy aplikacji	Protokół transportowy
Poczta elektroniczna	SMTP	TCP
Zdalny dostęp	SSH	TCP
Technologia WWW	HTTP	TCP
Strumieniowa transmisja danych audio i wideo	zwykle zastrzeżony	UDP lub TCP
Wideokonferencje	zwykle zastrzeżony	UDP lub TCP
Translacja nazw	DNS	UDP

20

Rysunek 14.5: Porównanie zastosowań TCP oraz UDP

■ IP



Rysunek 14.6: Datagram IPv4

- wersja protokołu IP – 4
- długość nagłówka, IHL (Internet Header Length) – liczona w 32-bitowych słowach, nagłówek może być maksymalnie 60 bajtowy
- typ usługi, ToS (Type of Service) – nieaktualne
- całkowita długość datagramu (16 bitów), ograniczenie rozmiaru do 2^{16} bajtów
- czas życia, TTL (Time To Live) (8 bitów) zmniejszany o 1 przez każdy router
- protokół warstwy wyższej (8 bitów), którego pakiet jest przenoszony np. 6 (TCP), 17 (UDP), 41 (IPv6)
- źródłowy adres IPv4 (32 bity)
- docelowy adres IPv4 (32 bity)
- standardowa internetowa suma kontrolna - obliczana tylko dla nagłówka, przetwarzana przez każdy router, zmienna ze względu na zmniejszanie TTL i możliwą zmianę opcji

Fragmentacja datagramu

Niekiedy konieczny jest podział datagramu na mniejsze pakiety.

- nadawca nadaje identyfikator każdemu segmentowi
- fragmenty są składane zanim trafią do warstwy transportowej odbiorcy (nie są w routeraх)
- warstwa sieci odbiorcy posługuje się identyfikatorem, pozycją fragmentu i flagą MF (more fragments) przy defragmentacji datagramu



Rysunek 14.7: Datagram IPv6

- stały rozmiar nagłówka: 40B
- brak sumy kontrolnej
- brak informacji dotyczących fragmentacji ze względu na to, że nie jest ona możliwa na routeraх, a jedynie na systemach końcowych, zbyt duży pakiet jest odrzucany przez router
- etykieta przepływu – routery mają traktować tak samo datagramy należące do tego samego przepływu

DHCP

DHCP - Dynamic Host Resolution Protocol

- w każdej sieci, bez względu na rozmiar
- przydział ma postać odnawialnej **dzierżawy**
- korzysta z UDP na portach 67 i 68
- pierwszy źródłowy adres klienta to 0.0.0.0
- docelowy adres serwera DHCP 255.255.255.255, nasłuchuje port 68, wysyła przez 67
- klient i serwer identyfikują się za pomocą **ID transakcji**

DHCPv6 ma dwa tryby działania: stanowy (podobny do DHCPv4) i bezstanowy (wykorzystuje autokonfigurację adresów).

Przekaźnik DHCP (agent przekazywania) - rozszerza dostępność DHCP poza sieć lokalną. Zwykle jest to router.

ICMP

Internet Control Message Protocol

- uzupełnienie funkcjonalności protokołu IP,
- służy do przesyłania informacji o błędach lub informacji kontrolnych
- komunikaty o **błędach** kierowane są do procesów użytkownika lub do warstwy wyższej (zawierają kolejne części datagramu, który był przyczyną błędu)

- na ogólnie komunikatami informacyjnymi zarządza system operacyjny

Kiedy komunikaty nie są generowane?

- inne komunikaty ICMP o błędach
- datagramy rozgłoszeniowe lub rozsyłania grupowego
- datagramy, których adres źródłowy nie określa konkretnego hosta, czyli np. z adresem zerowym, pętli zwrotnej, niepoprawnym
- kolejne fragmenty podzielonego datagramu

Przykłady komunikatów:

- **Destination Unreachable**,
- **Packet Too Big**,
- **Redirect** (jeśli router, zgodnie ze swoją tablicą tras, ma skierować pakiet do tej samej sieci, z której został wysłany, oznacza to błąd trasowania),
- **Time Exceeded** - TTL spadł do 0
- **RS** - zapytanie o router, **RA** - ogłoszenie routera (przydatne w DHCP)

■ Protokoły zapewniające bezpieczeństwo

SSL

- zapewnia bezpieczeństwo protokołu TCP
- najczęściej używa się go do ochrony transakcji realizowanych przez protokół HTTP (HTTPS), ale można go zastosować do dowolnej aplikacji korzystającej z TCP
- udostępnia oparty na gniazdach interfejs

IPSec

- zapewnia bezpieczeństwo w warstwie sieci
- chroni datagramy IP pomiędzy dwoma węzłami nawiązującymi logiczne połączenie
- służy do tworzenia wirtualnych sieci prywatnych, VPN (ang. Virtual Private Network), alternatywy dla sieci prywatnych, gdzie wewnętrzny ruch jest przesyłany w bezpieczny sposób przez sieć publiczną

■ Ethernet

Zestaw zadań

14.1. Przechwycono następujące zapytanie HTTP programem Wireshark:

```
GET /files/docs1.html HTTP/1.1\r\n
Host: example.com\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: pl;q=0.8, en-gb;q=0.2\r\n
Connection: keep-alive\r\n
\r\n
```

Wówczas

- A. adres URL żądanego zasobu to <http://example.com/files/docs1.html>
- B. ustanawiane jest połączenie nietrwałe
- C. zadeklarowano obsługę algorytmu kompresji gzip

14.2. IPsec sprawdzi się lepiej niż TCP z SSL do

- A. wideo telekonferencji
- B. przesyłania pliku
- C. rozmowy telefonicznej

14.3. W nagłówku TCP znajduje się

- A. numer portu nadawcy
- B. adres IP odbiorcy
- C. 32-bitowy numer kolejny pakietu

14.4. Protokół DHCP

- A. służy do odwzorowania adresów sieciowych IP na adresy sprzętowe MAC
- B. korzysta z protokołu UDP
- C. umożliwia wynajęcie adresu IP na określony czas

14.5. Konflikt w sieci Ethernet

- A. występuje w wyniku błędnej konfiguracji
- B. jest naturalnym zjawiskiem rozwiązywanym przez mechanizm CSMA/CD
- C. występuje gdy występują dwa urządzenia o tym samym adresie MAC

14.6. W sieciach TCP/IP zbudowanych w oparciu o technologię Ethernet parametr MTU oznacza

- A. maksymalny rozmiar pola danych ramki Ethernetowej
- B. maksymalny rozmiar segmentu danych TCP
- C. maksymalny rozmiar datagramu IP, który nie wymaga jeszcze stosowania fragmentacji

14.7. Nagłówek datagramu IP zawiera między innymi

- A. numer wersji tego protokołu
- B. adres IP nadawcy
- C. numer portu nadawcy

14.8. Protokół ARP służy do

- A. wiązania adresu fizycznego komputera z adresem IP
- B. wiązania nazwy komputera z adresem IP
- C. wiązania nazwy komputera z adresem fizycznym

14.9. Host *A* przesyła przez TCP plik o rozmiarze 1GiB do hosta *B*. *B* w warstwie aplikacji nie wysyła nic do *A*. Wynika stąd, że

- A. host *A* wyśle więcej niż 2020 segmentów z flagą ACK do *B*
- B. host *A* czeka na potwierdzenie każdego segmentu przed wysłaniem kolejnego
- C. liczba niepotwierdzonych bajtów wysłanych przez *A* nigdy nie jest większa niż rozmiar okna roboczego *B*

14.3.

Adresy internetowe, tablice tras, zasady trasowania, NAT

■ Adresy IP

Adresy IPv4 32-bitowe:

Reprezentacja dziesiętna	Reprezentacja binarna
10.1.0.122	00001010 00000001 00000000 01111010
193.0.96.129	11000001 00000000 01100000 10000001

Adresy IPv6 128-bitowe:

Reprezentacja szesnastkowa	Reprezentacja uproszczona
2001:06a0:5001:0002:34ed:69ae:55aa:7f61	2001:6a0:5001:2:34ed:69ae:55aa:7f61
2001:db8:0:0:0:0:2	2001:db8::2

Zasady upraszczania reprezentacji szesnastkowej:

- cyfry małymi literami, usuwanie wiodących zer
- redukcja najdłuższego spójnego ciągu zer

IPv6 bywa zapisywane w kwadratowych nawiasach np. [http://\[2001:db8:85a3:8d3:1319:8a2e:370:7348\]](http://[2001:db8:85a3:8d3:1319:8a2e:370:7348])

Historyczne klasy adresów IPv4

A	0	Numer sieci (7 bitów)	Numer hosta (24 bity)	
B	1 0	Numer sieci (14 bitów)	Numer hosta (16 bity)	
C	1 1 0	Numer sieci (21 bitów)	Numer hosta (8 bity)	
D	1 1 1 0	Adres multicast (28 bitów)		
E	1 1 1 1	Adresy zarezerwowane (28 bitów)		

CIDR - Classless Inter-Domain Routing

- zapewnia adresowanie bezklasowe
- wymaga wprowadzenia maski CIDR (prefiksu)

Reprezentacja bitowa	Zapis dziesiętny	Zapis prefiksowy
10000000 00000000 00000000 00000000	128.0.0.0	/1
11111111 11000000 00000000 00000000	255.192.0.0	/10

Nakładanie maski - koniunkcja bitowa.

Adres	00001010 00000010 00000110 10110011	10.2.6.179
Maska podsieci	11111111 11111111 11111111 00000000	255.255.255.0
Adres sieci	00001010 00000010 00000110 00000000	10.2.6.0/24

Obliczanie liczby urządzeń w sieci: $2^{32-\# \text{ bitów maski}} - 2$.

Adresy specjalne

Adres rozgłoszeniowy - alternatywa bitowa z zanegowaną maską. Jest to największy adres w sieci.

Adres	00001010 00000010 00000110 10110011	10.2.6.179
Zanegowana maska	00000000 00000000 00000000 11111111	0.0.0.255
Wynik	00001010 00000010 00000110 11111111	10.2.6.255

- 255.255.255.255/32 - ostatni adres z przestrzeni adresowej IPv4 - ograniczone rozgłoszanie lokalne,
- 127.0.0.0/8 - wszystkie adresy z tej sieci są adresami loopback, czyli wskazuje na bieżący węzeł; taki adres może być tylko **docelowym** adresem paczki,
- 224.0.0.0/4 - rozgłoszanie grupowe - od 224.0.0.0 do 239.255.255.255,
- 0.0.0.0/8 - wyłącznie adres źródłowy (np. kiedy nie jest jeszcze znany)

Adresy prywatne - mogą być używane tylko w sieci lokalnej - nie są rutownalne w sieci publicznej. **(przyp. red.: jakie to są adresy prywatne? dobrze byłoby to wyjaśnić)**

Prefiks	Pierwszy adres	Ostatni adres	Rozmiar
10.0.0.0/8	10.0.0.1	10.255.255.254	$2^{24} - 2$
172.16.0.0/12	172.16.0.1	172.31.255.254	$2^{20} - 2$
192.168.0.0/16	192.168.0.1	192.168.255.254	$2^{16} - 2$

Przykład 1.

Rozważmy sieć o masce /15. W tej sieci znajduje się komputer o adresie **111.96.99.74**.

Podaj liczbę dostępnych adresów dla urządzeń w tej sieci. – $2^{17} - 2$.

Czy adres 111.97.187.73 należy do tej sieci? – Tak, bo obydwa adresy należą do sieci 111.96.0.0

■ Przydzielanie adresów IP

Rodzaje:

- ręczne
- dynamiczne, za pomocą usług sieciowych
- automatyczne

■ Trasowanie

Algorytmy trasowania:

- algorytm stanu łączna
- algorytm wektora odległości

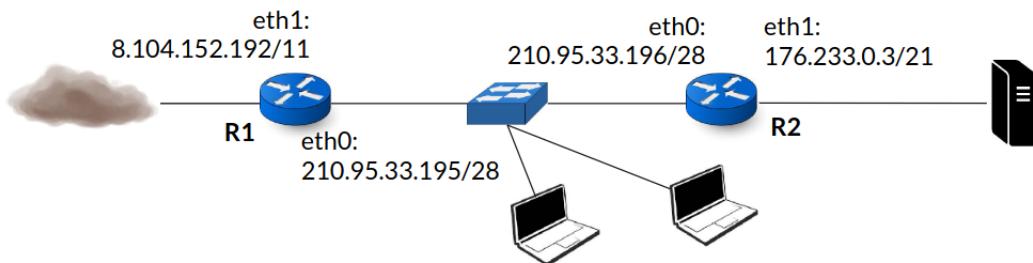
```
Destination   Gateway      Genmask        Flags Metric Ref Use Iface
 0.0.0.0       192.168.2.1  0.0.0.0        UG     100    0    0  eth0
 192.168.2.0   0.0.0.0     255.255.255.0  U      100    0    0  eth0
```

Elementy tablicy tras:

- **przeznaczenie** – razem z polem maski stanowią kryteria dopasowania adresu docelowego, wartość zerowa odpowiada trasie domyślnej
- **brama** - adres następnego węzła, wartość zerowa oznacza sieć lokalną
- **maska** - zostaje nałożona na adres docelowy, po czym następuje próba dopasowania wyniku do któregoś z pól przeznaczenia
- flagi, koszt, liczniki odwołań
- interfejs, który powinien dostarczyć datagram

Przykład 2.

Tablica tras dla routera R2.



Cel	Maska	Brama	Interfejs
210.95.33.192	255.255.255.240	0.0.0.0	eth0
176.233.0.0	255.255.248.0	0.0.0.0	eth1
0.0.0.0	0.0.0.0	210.95.33.195	eth0

Zasady dopasowania w trasowaniu:

- adres docelowy datagramu pasuje do trasy, jeśli jego koniunkcja bitowa z maską dla tej trasy daje w wyniku wartość pola przeznaczenie
- w przypadku wielu pasujących tras wybierany jest trasa z najdłuższym pasującym prefiksem (czyli z najdłuższą maską)
- jeśli nie zostanie odnaleziona pasująca trasa, to aplikacja dostanie komunikat "Network unreachable" lub do nadawcy zostanie wysłany odpowiedni komunikat ICMP

Przykład 3.

Destination	Gateway	Genmask	Iface
0.0.0.0	193.0.96.31	0.0.0.0	eth0
10.0.0.0	10.1.0.1	255.0.0.0	eth1
10.1.0.0	0.0.0.0	255.255.240.0	eth1
193.0.96.0	0.0.0.0	255.255.255.0	eth0

Nałożenie maski 255.255.255.0 na adres **193.0.96.42** daje w wyniku 193.0.96.0, więc adres pasuje do trasy czwartej.

Adres **10.1.0.42** pasuje do pozycji pierwszej, drugiej i trzeciej, wybrana zostanie trzecia.

NAT

Network Address Translation

- adres źródłowy, który jest adresem prywatnym zostaje zamieniony na adres globalny jakim dysponuje router NAT
- router z funkcją NAT oddziela prywatną sieć od internetu
- translacja odpowiedzi odbywa się w przeciwną stronę – globalny adres zostaje zamieniony na oryginalny adres prywatny

Zestaw zadań

14.10. Jeśli ruter musi przesyłać datagram IP w wersji 4 o całkowitej długości 4500 oktetów w sieci o MTU 1500, to

- A. podzieli go na 3 fragmenty
- B. zawsze w nagłówku drugiego fragmentu ustawia znacznik, że są jeszcze dalsze fragmenty
- C. zawsze w nagłówku ostatniego fragmentu ustawia znacznik zabraniający dalszej fragmentacji

14.11. Ruter o co najmniej dwóch interfejsach sieciowych ma między innymi adresy fizyczne 202.107.123.15/26, 150.192.0.20/19. Brama domyślna w jego tablicy tras ma najmniejszy możliwy adres w swojej sieci. Możliwe adresy bramy domyślnej to:

- A. 202.107.123.1
- B. 150.192.0.1
- C. inny niż 202.107.123.1 i 150.192.0.1

14.12. Mamy do dyspozycji sieć IPv4 klasy C. Chcemy podzielić ją na co najmniej 4 podsieci po co najmniej 16 urządzeń w każdej. Pozwoli nam na to maska

- A. 255.255.255.192
- B. 255.255.255.224
- C. 255.255.255.240

14.13. W sieci fizycznej, w której znajduje się maszyna o adresie 77.145.123.18/20,

- A. adresem IP urządzenia może być 77.145.112.255
- B. adresem ukierunkowanego rozgłaszenia jest 77.145.123.255
- C. można zaadresować $2^{20} - 2 = 1048574$ urządzenia

14.14. Adres planowanej sieci należy do dawnej klasy C. Chcemy podzielić sieć na co najmniej 5 podsieci, tak aby w każdej znalazło się co najmniej 16 urządzeń. Realizację tego wymagania zapewnia maska

- A. 255.255.255.240
- B. 255.255.255.224
- C. 255.255.255.192

14.15. W sieci publicznej nierutowalnymi adresami są:

- A. 127.0.0.1
- B. 10.1.1.2
- C. 174.49.2.87

14.4.

Systemy nazw domenowych

DNS - Domain Name System - zapewnia odwzorowanie nazwy na adres IP. Korzysta z protokołu DNS (port 53).

Serwery w hierarchii DNS:

- serwery główne
- TLD - serwery domen najwyższego poziomu (com, edu, net)

- serwery autorytatywne - każda organizacja dysponująca serwerami publicznymi musi udostępniać odwzorowanie ich nazw na adresy IP

Serwery lokalne - pierwszy krok w rozwiązywaniu nazwy. Obsługują zapytania klientów. Nie należą do hierarchii DNS.

Pliki strefy zawierają rekordy, czyli (nazwa, klasa, typ, wartość). Do typów należą m.in. (A - IPv4 hosta, AAAA- IPv6 hosta, MX - nazwa kanoniczna serwera pocztowego).

Zestaw zadań

14.16. W usłudze DNS

- A. każde zapytanie kierowane jest do głównego serwera IANA
- B. możliwe jest poznanie nazwy komputera z jego adresu IP
- C. wymagane jest użycie klienta Ethernetu jako drugiej warstwy sieciowej

14.5. Sieciowy interfejs gniazd

Gniazda BSD (BSD sockets): interfejs pomiędzy programami użytkowników a implementacją stosu TCP/IP. Procesy komunikują się za pomocą modelu klienta i serwera.

Tryby komunikacji gniazd: bezpołączeniowy (UDP) i połączeniowy (TCP).

Klient musi mieć informacje o adresie, porcie i protokole na którym serwer nasłuchuje. Serwer nie musi mieć informacji o kliencie – dostaje informacje po nawiązaniu komunikacji.

14.6. Rozwiązania

Rozwiązania

14.1. Przechwycono następujące zapytanie HTTP programem Wireshark:

```
GET /files/docs1.html HTTP/1.1\r\n
Host: example.com\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: pl;q=0.8, en-gb;q=0.2\r\n
Connection: keep-alive\r\n
\r\n
```

Wówczas

- TAK** A. adres URL żdanego zasobu to `http://example.com/files/docs1.html`
 - NIE** B. ustanawiane jest połączenie nietrwałe
 - TAK** C. zadeklarowano obsługę algorytmu kompresji gzip
- A. Łączymy informacje o protokole (`http`), hoście (`example.com`) oraz ścieżki zapytania (`/files/docs1.html`) zgodnie z zasadami tworzenia zapytań HTTP otrzymując adres z pytania.
- B. Zdefiniowane w zapytaniu `Connection: keep-alive` oznacza, że połączenie jest trwałe.
- C. Zdefiniowane w zapytaniu `Accept-Encoding: gzip` oznacza umiejętność obsługi kompresji gzip.

14.2. IPsec sprawdzi się lepiej niż TCP z SSL do

- A.** wideo telekonferencji
- B.** przesyłania pliku
- C.** rozmowy telefonicznej

IPsec korzysta z UDP, które nie nadaje się do przesyłania plików, ponieważ raczej chcemy, żeby plik dotarł do odbiorcy w całości. Jeśli chodzi o wideo telekonferencje i rozmowy telefoniczne, to najważniejsze, by rozmowa była płynna, w czym lepiej poradzi sobie protokół UDP.

14.3. W nagłówku TCP znajduje się

- A.** numer portu nadawcy
- B.** adres IP odbiorcy
- C.** 32-bitowy numer kolejny pakietu
 - A.** Wprost z grafiki ze strukturą.
 - B.** To dopiero jest w warstwie sieci.
 - C.** Wprost z grafiki ze strukturą.

14.4. Protokół DHCP

- A.** służy do odwzorowania adresów sieciowych IP na adresy sprzętowe MAC
- B.** korzysta z protokołu UDP
- C.** umożliwia wynajęcie adresu IP na określony czas
 - A.** To nie jest ARP, tylko DHCP.
 - B.** Owszem, na portach 67 i 68.
 - C.** No tak.

14.5. Konflikt w sieci Ethernet

- A.** występuje w wyniku błędnej konfiguracji
- B.** jest naturalnym zjawiskiem rozwiązywanym przez mechanizm CSMA/CD
- C.** występuje gdy występują dwa urządzenia o tym samym adresie MAC

(sprawdzić, czy da się uzupełnić to zadanie, tak żeby miało dydaktyczny sens)

14.6. W sieciach TCP/IP zbudowanych w oparciu o technologię Ethernet parametr MTU oznacza

- A.** maksymalny rozmiar pola danych ramki Ethernetowej
- B.** maksymalny rozmiar segmentu danych TCP
- C.** maksymalny rozmiar datagramu IP, który nie wymaga jeszcze stosowania fragmentacji

Wystarczy przeczytać, czym jest MTU i odpowiedzi stają się jasne.

14.7. Nagłówek datagramu IP zawiera między innymi

- A.** numer wersji tego protokołu
- B.** adres IP nadawcy
- C.** numer portu nadawcy
 - A.** Wprost z grafiki o datagramie IP.
 - B.** Źródłowy adres IP jest elementem nagłówka – wprost z grafiki o datagramie IP.
 - C.** To jest w nagłówku warstwy transportowej.

14.8. Protokół ARP służy do

- A.** wiązania adresu fizycznego komputera z adresem IP
- B.** wiązania nazwy komputera z adresem IP
- C.** wiązania nazwy komputera z adresem fizycznym

Protokół ARP służy do tłumaczenia adresu sieciowego IP na adres sprzętowy MAC, czyli dokładnie podpunkt **A**.

14.9. Host *A* przesyła przez TCP plik o rozmiarze 1GiB do hosta *B*. *B* w warstwie aplikacji nie wysyła nic do *A*. Wynika stąd, że

- A.** host *A* wyśle więcej niż 2020 segmentów z flagą ACK do *B*
- B.** host *A* czeka na potwierdzenie każdego segmentu przed wysłaniem kolejnego
- C.** liczba niepotwierdzonych bajtów wysłanych przez *A* nigdy nie jest większa niż rozmiar okna roboczego *B*

A. Host *A* nie musi wcale wysłać tylu segmentów z flagą ACK, bo *B* nie wysyła nic do *A*. Na początku połączenia podadzą sobie rękę, ale to nie będzie 2020 segmentów.

B. Nie jest tak, wykorzystywane jest buforowanie.

C. Tak działa okno robocze *B*, nie możemy umieścić w buforze więcej bajtów, niż jest w nim w ogóle dostępne.

14.10. Jeśli ruter musi przesyłać datagram IP w wersji 4 o całkowitej długości 4500 oktetów w sieci o MTU 1500, to

- A.** podzieli go na 3 fragmenty
- B.** zawsze w nagłówku drugiego fragmentu ustawi znacznik, że są jeszcze dalsze fragmenty
- C.** zawsze w nagłówku ostatniego fragmentu ustawi znacznik zabraniający dalszej fragmentacji

Datagram IP zawiera nagłówek oraz resztę danych. Dzieląc go na fragmenty za każdym razem powielamy nagłówek, a dane jakoś dzielimy. Stąd w **A.** podzielimy go na więcej niż 3 fragmenty. **B.** tak jest po prostu, logiczne. **C.** skoro jest to co w **B.**, to to by już było useless, nie ma tego.

14.11. Ruter o co najmniej dwóch interfejsach sieciowych ma między innymi adresy fizyczne 202.107.123.15/26, 150.192.0.20/19. Brama domyślana w jego tablicy tras ma najmniejszy możliwy adres w swojej sieci. Możliwe adresy bramy domyślnej to:

- A.** 202.107.123.1
- B.** 150.192.0.1
- C.** inny niż 202.107.123.1 i 150.192.0.1

Adres 202.107.123.15 ma maskę 26, czyli zmienić możemy jedynie ostatnie 6 bitów w adresie. W adresie 150.192.0.20 z maską 19 możemy zmienić 5 młodszych bitów w 3-cim od lewej bajcie i wszystkie 8 w najbardziej prawym. **A.** adres 202.107.123.1 jest adresem urządzenia w pierwszej sieci, więc ok. **B.** tak samo, ustawiamy wszystkie bity oprócz ostatniego na 0 i jest ok. **C.** wiemy, że ruter ma co najmniej dwa interfejsy i podane są dwa przykłady adresów, ale może być ich więcej – nie wiemy tego.

14.12. Mamy do dyspozycji sieć IPv4 klasy C. Chcemy podzielić ją na co najmniej 4 podsieci po co najmniej 16 urządzeń w każdej. Pozwoli nam na to maska

- A.** 255.255.255.192
- B.** 255.255.255.224
- C.** 255.255.255.240

(A. dla bardzo starych sieci (sprzed roku 1995) to może nie działać. Patrz: <https://www.cisco.com/c/en/us/support/docs/ip/dynamic-address-allocation-resolution/13711-40.html#problemswithsubnetzero>. W oryginalnej wersji zadania było wspomniane, że sieć jest stara (i dlatego mamy do czynienia z klasami), ale nie było powiedziane, jak bardzo stara, więc obie odpowiedzi się bronią.)

Przyp. red.: dopisać rozwiązanie do tego zadania (obecnie brak wyjaśnienia, dlaczego odpowiedzi są takie, a nie inne) oraz przerzucić całe to zadanie do ramki „To było na egzaminie” w sensownym miejscu w części teoretycznej – istnieje już bardzo analogiczne zadanie, więc nie ma sensu, żeby się dublowało w tym zestawie zadań.

14.13. W sieci fizycznej, w której znajduje się maszyna o adresie 77.145.123.18/20,

- TAK A. adresem IP urządzenia może być 77.145.112.255
 NIE B. adresem ukierunkowanego rozgłoszania jest 77.145.123.255
 NIE C. można zaadresować $2^{20} - 2 = 1048574$ urządzenia

Mamy maskę 20, więc 20 najbardziej lewych bitów jest ustawione. Adres sieci to zatem 77.145.112.0, a adres rozgłoszeniowy to 77.145.127.255. A. 77.145.112.255 jest pomiędzy adresem sieci a rozgłoszeniowym – git. B. Nie. C. Z maską równą k można w IPv4 adresować $2^{32-k} - 2$ urządzeń, czyli tutaj $2^{12} - 2 = 4094$.

14.14. Adres planowanej sieci należy do dawnej klasy C. Chcemy podzielić sieć na co najmniej 5 podsieci, tak aby w każdej znalazło się co najmniej 16 urządzeń. Realizację tego wymagania zapewnia maska

- NIE A. 255.255.255.240
 TAK B. 255.255.255.224
 NIE C. 255.255.255.192

14.15. W sieci publicznej nierutowalnymi adresami są:

- TAK A. 127.0.0.1
 TAK B. 10.1.1.2
 NIE C. 174.49.2.87

14.16. W usłudze DNS

- NIE? A. każde zapytanie kierowane jest do głównego serwera IANA
 TAK? B. możliwe jest poznanie nazwy komputera z jego adresu IP
 NIE? C. wymagane jest użycie klienta Ethernetu jako drugiej warstwy sieciowej

(być może w B. chodzi o reverse DNS)

15

Systemy operacyjne

Materiały teoretyczne zostały opracowane na podstawie slajdów Marcina Engela i Marcina Peczarskiego.
(przyp. red.: potrzebny link do źródła)

Podstawa programowa

1. **Mechanizmy sprzętowe** potrzebne do realizacji wielodostępnych, wieloprocesorowych systemów operacyjnych.
2. Podstawy **programowania niskopoziomowego**, asembler.
3. Algorytmy **szeregowania procesów**.
4. **Pamięć wirtualna**. Cechy charakterystyczne różnych technik realizacji pamięci wirtualnej.
5. Funkcje systemowe do **obsługi plików** z poziomu użytkownika (czynności wykonywane przez system operacyjny, struktury danych).

15.1.

Programowanie niskopoziomowe

Przypomnienie assemblera: w składni NASM operandy są w kolejności najpierw **dest** (miejsce docelowe) później **src** (źródło).

Przykład 1.

Instrukcja `mov ax, 9` zapisuje wartość 9 do rejestru ax.

■ Reprezentacja liczb całkowitych

Wartość n -bitową $b_{n-1}b_{n-2}\dots b_1b_0$ w **naturalnym kodzie binarnym** (NKB) interpretujemy jako $\sum_{i=0}^{n-1} b_i 2^i$, a w **kodzie uzupełniającym do dwójki** (U2) jako $-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$.

Te same układy arytmetyczne mogą być użyte do dodawania, odejmowania i mnożenia w NKB i U2, niezależnie od interpretacji.

Little-endian (cienkokońcowość) to forma zapisu danych, w której najmniej znaczący bajt umieszczany jest jako pierwszy. W **big-endian** (grubokońcowość) najbardziej znaczący bajt umieszczony jest jako pierwszy.

Procesory x86 używają little-endian.

Przypis redakcji

Brakuje interpretacji moduł-znak, warto o niej tu coś napisać, bo pojawia się w dalszej teorii i zadaniach.

To było na egzaminie

W ośmiobitowym rejestrze procesora zapisana jest wartość $(11000000)_2$, która interpretowana

- A. w naturalnym kodzie binarnym reprezentuje liczbę 192
- B. w kodzie uzupełnieniowym do dwójki reprezentuje liczbę -64
- C. w kodzie moduł-znak reprezentuje liczbę -64

- A. Liczby w naturalnym kodzie binarnym są interpretowane jako $\sum_{i=0}^{n-1} b_i 2^i$, zatem wartość $(11000000)_2$ reprezentuje liczbę $2^7 + 2^6 = 192$.
- B. Liczby w kodzie uzupełnieniowym do dwójki są interpretowane jako $-b_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$, zatem wartość $(11000000)_2$ reprezentuje liczbę $-2^7 + 2^6 = -64$.
- C. Liczby w kodzie moduł-znak są interpretowane jako $(-1)^{b_{n-1}} \cdot \sum_{i=0}^{n-2} b_i 2^i$, zatem wartość $(11000000)_2$ reprezentuje liczbę $(-1) \cdot (2^6) = -64$.

■ Rejestr stanu

W x86 obecny jest rejestr stanu. Zawiera on pewne bity, aktualizowane po każdej operacji arytmetyczno-logicznej. Ważniejsze bity:

- bit OF – *overflow*, wynik spoza zakresu przy interpretacji U2,
- bit SF – *sign*, najstarszy bit wyniku operacji, przy interpretacji w U2 znak wyniku,
- bit CF – *carry*, przeniesienie lub pożyczka z najstarszego bitu,
- bit ZF – *zero*, wynik operacji jest zerem.

Przykład 2.

odejmowanie	interpretacja w NKB	interpretacja w U2
10000010	130	-126
00000011	3	3
01111111	127	127

Bity stanu: CF = 0 OF = 1 SF = 0

odejmowanie	interpretacja w NKB	interpretacja w U2
10000001	129	-127
10000010	130	-126
11111111	255	-1

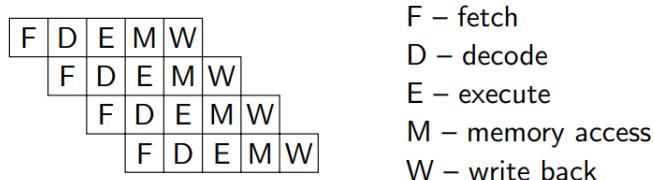
Bity stanu: CF = 1 OF = 0 SF = 1

■ Architektury

CISC (Complex Instruction Set Computer)	RISC (Reduced Instruction Set Computer)
wiele rozkazów, potencjalnie skomplikowanych	ograniczony, proste rozkazy
mała liczba rejestrów lub rejestrów specjalizowane	duża liczba uniwersalnych rejestrów
rozkazy arytmetyczno-logiczne pobierają argumenty z pamięci lub umieszczą w niej wynik	rozkazy arytmetyczno-logiczne wykonywane są tylko na rejestrach
duża liczba trybów adresowania	mała liczba trybów adresowania
dozwolone adresowanie niewyrówniane	dozwolone tylko adresowanie wyrówniane
kody rozkazów zmiennej długości, od jednego do kilkunastu bajtów	kody rozkazów o stałej długości, typowo 4 bajty
prefiksowanie rozkazów, utrudniające dekodowanie	stałe rozmieszczenie pól w kodach, ułatwiające dekodowanie
przykładem jest x86, często używane w PC	przykładem jest ARM, często używane w smartfonach i Mac'ach

■ Potokowanie

Różne części hardware'u są odpowiedzialne za kolejne fazy przetwarzania instrukcji. Można zatem równolegle przetwarzać kolejne instrukcje, potencjalnie znacznie przyspieszając działanie.



F – fetch
D – decode
E – execute
M – memory access
W – write back

Rysunek 15.1: Fazy przetwarzania instrukcji w czasie.

Przy skokach warunkowych, nie wiadomo jakie instrukcje zaczynać przetwarzać do przodu. Współczesne procesory dokonują predykcji skoków.

Rozkazy x86 źle się potokują, bo kod rozkazu nie ma stałego rozmiaru, jest bardziej skomplikowany, oraz czasy wykonywania poszczególnych rozkazów mogą się bardzo różnić.

Rozwiążanie: pośrednie tłumaczenie rozkazów na RISC.

Zestaw zadań

15.1. W procesorze x86 wykonano następujące instrukcje:

```
mov al, 3
mov bl, 130
sub al, bl
```

Wtedy

- A. w rejestrze `al` jest zapisana wartość -127 przy interpretacji w kodzie uzupełnień do dwóch
- B. w rejestrze `al` jest zapisana wartość 127 przy interpretacji w naturalnym kodzie binarnym
- C. ustawiione zostaną flagi CF, OF, SF

15.2. Cechą architektury RISC jest

- A. zapisywanie wyników instrukcji arytmetyczno-logicznych tylko w rejestrach
- B. kodowanie wszystkich instrukcji maszynowych za pomocą tej samej liczby bajtów

- C. dopuszczenie adresowania niewyrównanego

15.3. Cechą architektury CISC jest

- A. zapisywanie wyników instrukcji arytmetyczno-logicznych tylko w rejestrach
 B. kodowanie wszystkich instrukcji maszynowych za pomocą tej samej liczby bajtów
 C. dopuszczenie adresowania niewyrównanego

15.4. W czterech kolejnych bajtach pamięci, począwszy od adresu X , znajdują się odpowiednio wartości 1, 3, 5 i 7. Procesor cienkokońcowkowy (ang. *little-endian*) wczytał 32-bitową liczbę spod adresu X , odjął od niej $(16)_{10}$ i zapisał wynik pod adresem X . Po tych operacjach bajt o adresie

- A. X zawiera wartość $(F1)_{16}$
 B. $X + 1$ zawiera wartość $(03)_{16}$
 C. $X + 2$ zawiera wartość $(05)_{16}$

15.5. W ośmiorzeczu procesora zapisana jest wartość $(11000000)_2$, która interpretowana

- A. w naturalnym kodzie binarnym reprezentuje liczbę 192
 B. w kodzie uzupełnieniowym do dwójkii reprezentuje liczbę -64
 C. w kodzie moduł-znak reprezentuje liczbę -64

15.6. Przetwarzanie potokowe

- A. zawsze przyspiesza lub nie spowalnia wykonywania rozkazów
 B. niewykorzystywane jest dla rozkazu skoku
 C. zostało zastosowane po raz pierwszy w procesorach Pentium

(przyp. red.: odpowiedzi C. nie da się wywnioskować na podstawie części teoretycznej)

15.7. Każdy procesor 32-bitowy ma

- A. 32-bitową zewnętrzną szynę danych
 B. 32-bitową zewnętrzną szynę adresową
 C. 32-bitowe rejestrory danych

(przyp. red.: tego zadania nie da się wywnioskować na podstawie części teoretycznej)

15.2. Szeregowanie procesów

Program to zestaw instrukcji w formie kodu, tekstu.

Proces to wykonanie programu. Aktywny obiekt w odróżnieniu od programu. Wiele procesów może wykonywać ten sam program ale proces wykonuje tylko jeden na raz. Proces może w trakcie działania zmieniać wykonywany program.

Na proces składają się:

- przydzielona mu pamięć, czyli **przestrzeń adresowa** (zawiera między innymi Kod programu, dane programu, stos procesu)
- zbiór rejestrów (licznik rozkazów, wskaźnik stosu, rejestr stanu, rejestrory specjalne)
- informacje administracyjne (stan, kod zakończenia, ...)

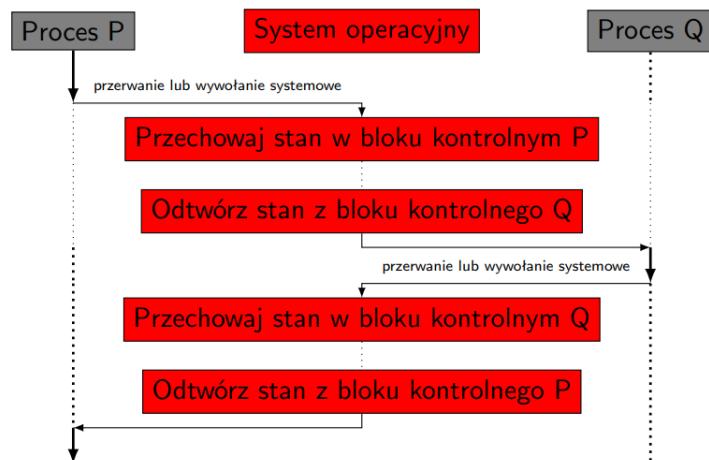
- inne zasoby potrzebne do wykonywania programu:
 - otwarte pliki
 - identyfikator procesu (pid)
 - przestrzeń wejścia-wyjścia
 - stosy alternatywne (wykorzystywane w trakcie obsługi przerwań lub wywołań systemowych)

Informacje o procesach w systemie przechowywane są w tablicy/liście procesów w formie PCB (Process Control Block). Na PCB składają kluczowe dane do działania procesów.

Podział czasu:

- każdy proces otrzymuje kwant czasu
- po upływie kwantu procesor "odkłada" wykonanie procesu i rozpoczyna wykonanie innego - **przełączanie kontekstu**
- proces nie musi wykorzystywać całego kwantu czasu (np. wykonanie operacji wejścia-wyjścia)

Przełączanie kontekstu

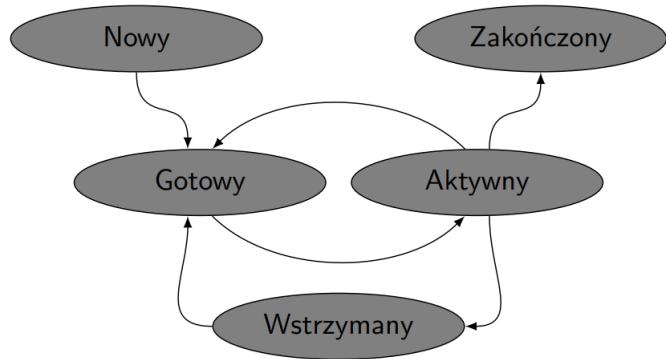


Szeregowanie to zadanie wyboru kolejnego procesu:

- do wykonania (szeregowanie krótkoterminowe)
 - Outdated, dotyczyło raczej systemów z przetwarzaniem wsadowym.
 - Chodziło o dobranie procesów tak, aby procesor był równomiernie wykorzystywany. Stworzenie mieszanki procesów ograniczanych przez wejście/wyjście i obliczeniowych.
- do wprowadzenia do systemu (szeregowanie długoterminowe).

■ Szeregowanie krótkoterminowe

Uproszczony schemat stanów procesów



Szeregowanie krótkoterminowe można podzielić na 2 typy:

- **wywłaszczające**. Decyzja o przydiale procesora, gdy:
 - aktywny proces zmienia stan na wstrzymany
(dobrowolne oddanie procesora: operacja wejścia/wyjścia, wstrzymanie na semaforze)
 - aktywny proces zmienia stan na gotowy (w systemie z podziałem czasu kończy się kwant czasu)
 - wstrzymany lub nowy proces zmienia stan na gotowy
(wstrzymany → gotowy, np. system otrzymuje przerwanie o zakończeniu operacji wejścia/wyjścia)
 - aktywny proces kończy się
- **niewywłaszczające** Decyzja o przydiale procesora, gdy:
 - aktywny proces zmienia stan na wstrzymany
 - aktywny proces kończy się

Szeregowanie niewywłaszczające

W ogólności mało przydatne, jednak jedyna metoda w przypadku niektórych sprzętów (np. tych bez wsparcia systemowego w postaci czasomierza). Występuje mniej problemów synchronizacyjnych, jednak procesy dłużej oczekują na możliwość wykonania się.

Szeregowanie wywłaszczające

Synchronizacja dostępu do danych dzielonych jest wymagana (semafony, monitory). Ponownie można tutaj wprowadzić podział:

- jądro wywłaszcjalne - przełączamy kontekst nawet w trakcie wykonywania procedur jądra systemowego
- jądro niewywłaszcjalne - wszystko można przełączyć oprócz procedur jądra systemu (czekamy, aż ona się skończy)

■ Algorytmy szeregowania

FCFS

First Come First-Served - szeregowanie niewywłaszczające na zasadzie kolejki prostej. Przychodzące procesy na koniec kolejki, następne do wykonania bierzemy z jej początku.

Procesy

P_1	20
P_2	2
P_3	2
P_4	4

Jeśli kolejność w kolejce to P_1, P_2, P_3, P_4



Czasy oczekiwania:

Proces	Czas oczekiwania
P_1	0
P_2	20
P_3	22
P_4	24
Średnio	16,5

Procesy

P_1	20
P_2	2
P_3	2
P_4	4

Jeśli kolejność w kolejce to P_4, P_3, P_2, P_1



Czasy oczekiwania:

Proces	Czas oczekiwania
P_1	8
P_2	6
P_3	4
P_4	0
Średnio	4,5

Przykład 1.

Przykład 2.

Wnioski:

- Łatwy do zrozumienia i zaimplementowania
- Średni czas oczekiwania może być długi
- Duża wariancja czasu oczekiwania
- Efekt konwoju - proces P_1 wykonujący 1 sekundowe fazy aktywności. Procesy $P_2 \dots P_n$ wykonujące 1200 operacji wyjścia i wyjścia. Skutek: $P_2 \dots P_n$ szybko wykorzystają swoją fazę i będą musiały czekać na kolejną 1200 razy, co daje czas oczekiwania 1200s = 20m

SJF

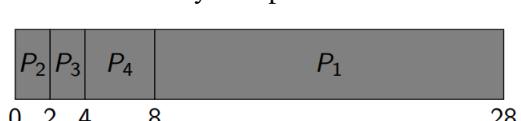
Shortest Job First - szeregowanie niewywłaszczające. Algorytm zachłanny, optymalizujący średni czas oczekiwania. Czas wykonania jednak w ogólności nie jest znany, dlatego stosuje się heurystyki. Najpierw bierzemy procesy o najkrótszej fazie.

Przykład:

Procesy

P_1	20
P_2	2
P_3	2
P_4	4

Przydział procesora



Czasy oczekiwania:

Proces	Czas oczekiwania
P_1	8
P_2	0
P_3	2
P_4	4
Średnio	3,5

Wnioski:

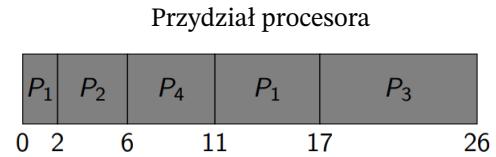
- Optymalizuje średni czas oczekiwania
- Nie daje się zastosować przy planowaniu przydziału procesora, bo trudno oszacować długość następnej fazy procesora

SRTF

SRTF - Shortest Remaining Time First. Wywłaszczająca wersja SJF - gdy pojawia się nowy proces o krótszej fazie procesora niż długość fazy pozostałojej wykonywanemu, to dochodzi do wywłaszczenia.

Przykład:

Proces	Czas przybycia	Czas następnej fazy
P_1	0	8
P_2	2	4
P_3	3	9
P_4	4	5



Wnioski:

- Może wystąpić zagłodzenie

Szeregowanie priorytetowe

Szeregowanie priorytetowe to uogólniona wersja SJF. Priorytety mogą być:

- wewnętrzne - obliczane przez SO na podstawie pewnych właściwości procesu
- zewnętrzne - przydzielane poza SO

Wnioski:

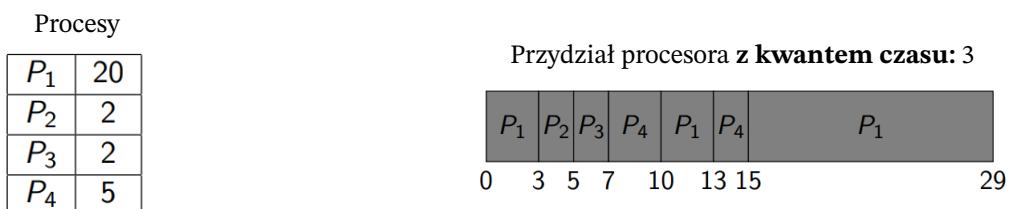
- Może wystąpić zagłodzenie
- Może być wywłaszczające lub niewywłaszczające

Można sobie jednak radzić z zagłodzeniem co jakiś czas zwiększając priorytet procesom niewybranym do wykonania - **postarzanie**.

Szeregowanie rotacyjne

Round-robin (RR) - wywłaszczająca wersja FCFS. Procesy gotowe przechowywane są w kolejce. Wykonywany proces jest wywłaszczany co ustalony kwant czasu i wraca na koniec kolejki. Do wykonania wybierany jest następny proces z kolejki.

Przykład:



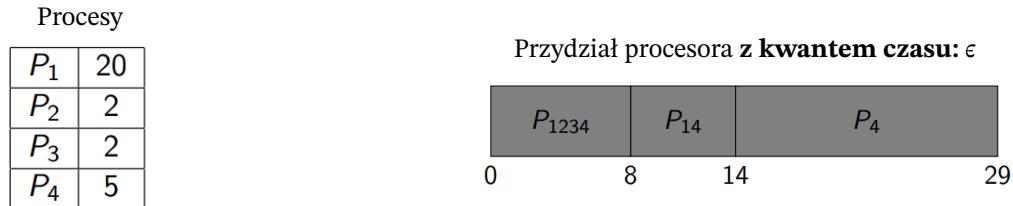
Wnioski:

- Do realizacji konieczne wsparcie sprzętowe – czasomierz generujący przerwania
- Trzeba dobrze dobrać kwant czasu:
 - Duży kwant czasu **RR=FCFS**

- Kwant czasu mniejszy od przełączania kontekstu → niewykorzystywany procesor.

Szczególnym przypadkiem szeregowania rotacyjnego jest Podział Czasu - uzyskujemy je w momencie gdy kwant czasu wynosi ϵ

Przykład:



Szeregowanie wielopoziomowe

Implementowane we współczesnych systemach, między innymi: Minix i Linux

- Osobne kolejki dla poszczególnych grup procesów, np.
 - procesy czasu rzeczywistego
 - procesy systemowe
 - pierwszoplanowe procesy interakcyjne
 - procesy drugoplanowe
- Każda kolejka z własnym algorytmem szeregowania, np.
 - pierwszoplanowe procesy interakcyjne – RR
 - procesy drugoplanowe – FCFS
- Zasady wyboru kolejki, np.
 - procesy systemowe mają zawsze priorytet przed interakcyjnymi
 - kolejka procesów systemowych otrzymuje 80% czasu procesora

■ Systemy czasu rzeczywistego i ich szeregowanie

System reaktywny to system działający w sposób ciągły. Otrzymuje dane wejściowe od układów sprzętowych i przetwarza i przekazuje innym.

System czasu rzeczywistego to system reaktywny, który musi spełniać ścisłe wymagania na czas reakcji.

System wbudowany to system, w którym komputer jest tylko jednym z wielu elementów, np. systemy sterujące lotem.

Szeregując systemy czasu rzeczywistego chcemy:

- Zapewnić dotrzymania terminów uruchomienia poszczególnych procesów (żeby piec nie wybuchł sobie, jak za późno świeczkę zgasimy).
- Zapewnić przewidywalność i powtarzalność

Szeregowanie systemów rzeczywistych można podzielić na:

- Szeregowanie synchroniczne.
- Szeregowanie asynchroniczne.

Szeregowanie synchroniczne

Czas procesora dzielimy na przedziały o ustalonej długości – **ramy** (frames). Program dzielimy na zadania. Zadania muszą kończyć się w ciągu jednej ramy. Opisywane za pomocą **tablicy szeregującej**, która zawiera przypisanie zadań do ram. Jest ona konstruowana *a priori*.

Przykład (wg Ben-Ariego):

Wymagania czasowe			Tablica szeregująca				
Zadanie	Długość	Częstość	0	1	2	3	4
Próbkowanie	1	co 5 ram	Próbk.	Oblicz.	Ster.	Telem. 1	Samospr.
Obliczenia	1	co 5 ram	5	6	7	8	9
Sterowanie	1	co 5 ram	Próbk.	Oblicz.	Ster.	Telem. 2	Telem. 1
Telemetria	2	co 15 ram	10	11	12	13	14
Samosprawdzenie	1	co 10 ram	Próbk.	Oblicz.	Ster.	Telem. 2	Samospr.

Istotne informacje:

- W sytuacji, w której zadanie nie mieści się w jednej ramie, trzeba podzielić je na części.
- Jeżeli zmieni się rozmiar ramy, to trzeba przeliczyć tablicę szeregującą i ponownie podzielić zadania na ramy.
- Jeśli zmieni się kod zadania, trzeba upewnić się że dalej mieście się w ramach.
- Jeśli zadanie nie wypełnia ramy - czas procesora się marnuje.
- Zastosowanie: systemy sterujące ze sztywnymi więzami czasowymi.

Wnioski:

- Zalety: powtarzalne, proste
- Wady: marnowanie czasu procesora, nieskalowalne, trudne w pielęgnacji

Szeregowanie asynchroniczne

Klasyczne szeregowanie asynchroniczne, czyli takie w którym nie wiemy kiedy dokładnie wystąpi przełączenie kontekstu. Może być realizowane tak jak w systemach ogólnego przeznaczenia **z wywłaszczeniem** albo **bez wywłaszczenia**.

Bez wywłaszczenia:

- brak powtarzalności, zachowaniem terminów ostatecznych. Nie jesteśmy w stanie powiedzieć kiedy ostatecznie zakończy się zadanie, gdyż nie wiemy ile poprzednie będą się wykonywać.

Dlatego w praktyce stosuje się szeregowanie z wywłaszczeniem i priorytetami zadań.

Zestaw zadań

15.8. Proces jest w stanie

- A. aktywnym, gdy jest obecnie wykonywany przez procesor
- B. wstrzymanym, gdy czeka na zakończenie operacji wejścia-wyjścia, lub czeka na jakieś zdarzenie
- C. gotowym, gdy zakończył się i czeka na odczytanie końcowego stanu przez rodzica

15.9. W chwili 0 w kolejce znajdują się procesy z następującą długością najbliższej fazy procesora:

$$P = 10, \quad Q = 4, \quad R = 2, \quad S = 6$$

- A. w strategii szeregowania z podziałem czasu (ang. *time sharing*) proces *R* kończy swój ostatni przydział procesora w chwili 7
- B. w strategii szeregowania rotacyjnego (ang. *round-robin*) z kwantem czasu 3 proces *Q* kończy swój ostatni przydział procesora w chwili 15
- C. w strategii szeregowania SJF (ang. *shortest job first*) bez wywłaszczenia proces *P* dostaje procesor jako ostatni

15.3. Zakleszczenia

Zbiór procesów *P* jest w stanie zakleszczenia, gdy każdy proces z *P* czeka na zdarzenie, które może być spowodowane tylko przez inny proces z *P*.

■ Warunki konieczne

- **niepodzielność** co najmniej jednego zasobu
- **przetrzymywanie i oczekiwanie** – pewien proces otrzymał co najmniej jeden zasób i oczekuje na inny przetrzymywany przez inny proces
- **brak wywłaszczeń** – zasób może być zwolniony jedynie dobrowolnie przez przetrzymujący go proces
- **czekanie cykliczne** – istnieje zbiór procesów $\{P_0, \dots, P_n\}$, taki że P_i czeka na zasób przetrzymywany przez P_{i+1} dla $i \in \{0, \dots, n - 1\}$, a P_n czeka na zasób przetrzymywany przez P_0

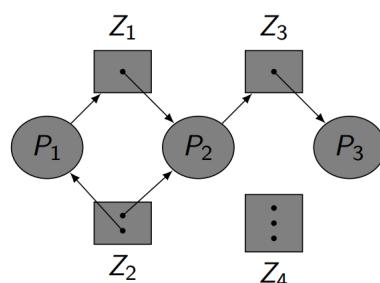
■ Graf przydziału zasobów

To skierowany graf dwudzielny, którego wierzchołkami są:

- procesy P_1, P_2, \dots - przedstawiane jako kółka
- zasoby systemu Z_1, Z_2, \dots - przedstawiane jako kwadraty

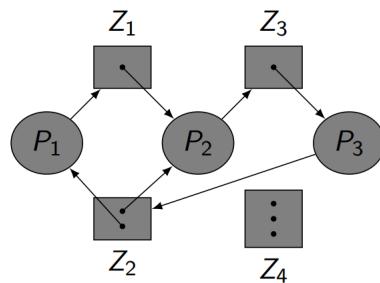
Ponadto:

- Egzemplarze zasobów oznaczane są jako kropki
- Krawędź $P_i \rightarrow Z_j$ oznacza, że P_i zamówił Z_j i czeka na niego
- Krawędź $Z_j \rightarrow P_i$ oznacza, że P_i ma przydzielony zasób Z_j

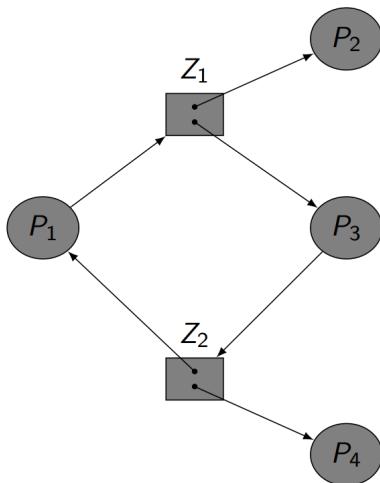


■ Warunek konieczny zakleszczenia

Jeśli system jest w stanie zakleszczenia, to w grafie zasobów istnieje cykl.



Jednak odwrotne twierdzenie nie jest prawdziwe - kontrprzykład:



■ Radzenie sobie z zakleszczeniami

Można zapobiegać, unikać, wykrywać i usuwać lub nie robić nic. Nic nie robienie to najczęściej stosowane rozwiązanie we współczesnych systemach.

Jednak gdy zdecydujemy się co robić, to jak?

- **Zapobieganie** - upewnić się, że któryś z warunków koniecznych nie jest spełniony
- **Unikanie** - wprowadzamy pojęcia:
 - Każdy proces deklaruje **maksymalną** liczbę potrzebnych zasobów każdego typu.
 - **Stan bezpieczny** - stan, w którym system może przydzielić zasoby każdemu procesowi (w stopniu maksymalnym), unikając zakleszczenia.
 - **Stan zagrożenia** - stan, który nie jest bezpieczny. Stan zagrożenia może prowadzić do zakleszczenia.
 - **Stan zakleszczenia** - szczególny przypadek stanu zagrożenia.

Następnie wykorzystujemy je przy realizacji algorytmu **Bankiera** - analogii bankiera, który nigdy nie zainwestuje gotówki w sposób uniemożliwiający zaspokojenie roszczeń wszystkich klientów

- **Wykrywanie i usuwanie:**
 - Wykrywanie - symulacja przydziału i zwrotu zasobów na podstawie aktualnego stanu
 - Usuwanie - kończenie procesów i wywłaszczenie zasobów

Zestaw zadań

15.10. W grafie przydziału zasobów istnieje cykl. Wynika z tego, że

- A. system jest w stanie blokady
- B. system jest w stanie bezpiecznym
- C. istnieje proces oczekujący na zasoby

15.11. Warunkiem wystarczającym powstania zakleszczenia jest

- A. niepodzielność co najmniej jednego zasobu
- B. brak wywłaszczeń
- C. przetrzymywanie i oczekiwanie

15.4.

Zarządzanie pamięcią

Brak zarządzania pamięcią – użytkownik musi sam wszystko oprogramować.

Pojedynczy obszar pamięci – pamięć użytkownika stanowi jeden duży obszar, system operacyjny udostępnia tylko podstawowe operacje (np. ładowanie programów).

Strefy statyczne – pamięć jest podzielona na pewną liczbę obszarów, każdy obszar ma ustalony rozmiar. W każdym obszarze może wykonywać się jeden proces. Gdy zgłasza się proces, szuka się dla niego wystarczająco dużego obszaru. Zachodzi **fragmentacja wewnętrzna** – część obszaru przydzielonego pozostaje niewykorzystana.

Przydział ciągły — procesy otrzymują ciągłe bloki pamięci. Obszary wolnej pamięci tworzą tzw. dziury. Gdy pojawia się proces, system operacyjny szuka dla niego wystarczająco obszernej dziury. Zachodzi **fragmentacja zewnętrzna** – suma rozmiarów wolnych obszarów by starczyła do spełnienia zamówienia, ale nie jest spójnym obszarem.

Przydział ciągły może być obsługiwany przez **algorytm bliźniaków**:

- Pamięć jest podzielona na 2^n jednostek, początkowo stanowi jeden spójny blok.
- Procesy zawsze proszą o 2^k jednostek (fragmentacja wewnętrzna). Jeśli nowy proces chce pamięć rozmiaru jakiegoś ciągłego bloku, to go dostaje.
- W przeciwnym przypadku szukamy większego bloku i dzielimy go na pół, aż dojdziemy do oczekiwanej rozmiaru.
- Przy oddawaniu pamięci, system operacyjny sprawdza, czy bliźniak bloku jest wolny. Jeśli tak, to sklejamy dopóki się da.

Stronicowanie

Pamięć fizyczna jest podzielona na ramki o takim samym rozmiarze będącym potęgą dwójki, pamięć logiczna jest podzielona na strony o takim samym rozmiarze jak ramka.

Każdy proces ma tablicę stron, która zawiera odwzorowanie numerów stron na numery ramek

Wymaga wsparcia sprzętowego.

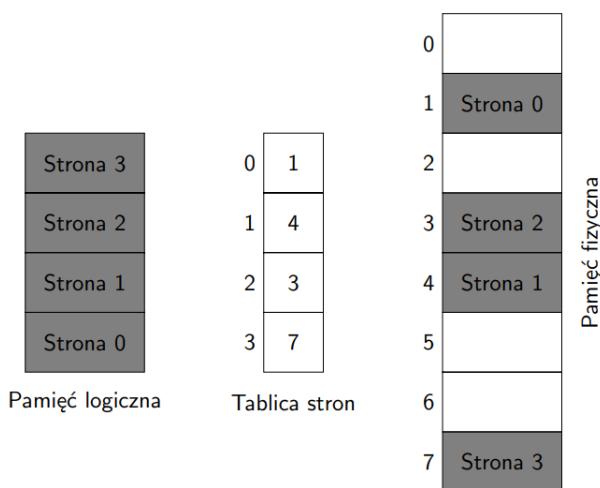
Adresy logiczne składają się z numeru strony i przesunięcia w obrębie strony.

Przykład 1.

W architekturze 32-bitowej przy rozmiarze strony 4 KiB:

- bity 0-11 oznaczają przesunięcie

- bity 12-31 oznaczają numer strony



Rysunek 15.2: Przykład tablicy stron procesu

Translacja adresu jest wykonywana sprzętowo, polega na zamianie numeru strony na numer ramki, korzystając z tablicy stron.

Zalety: mała segmentacja wewnętrzna, brak segmentacji zewnętrznej.

Problemy:

- Zwiększa się czas potrzebny na dostęp do komórki pamięci.
Remedium: bufor translacji adresów strony (TLB). Mała, szybka pamięć asocjacyjna. Zawiera kilka pozycji z tablicy stron (parę numer strony, numer ramki).
- Tablica stron może być bardzo duża.
Remedium: wielopoziomowe tablice stron.

Przykład 2.

Przy dwupoziomowych tablicach stron, mamy katalog tablic stron który trzyma adresy odpowiednich tablic stron. Część komórek może być pusta, jeśli nie mamy odwołań do odpowiadającym ich adresem.

Przy 32-bitowych adresach i rozmiarze strony 4KiB, przykładowy podział adresu:

- bity 0-11 oznaczają przesunięcie
- bity 12-21 oznaczają indeks w tablicy stron
- bity 22-31 oznaczają indeks w katalogu tablic stron

Pamięć wirtualna

Przy **pamięci wirtualnej**, nie wszystkie strony procesu muszą być w pamięci fizycznej.

W tablicy stron przechowywany jest bit, który informuje, czy dana strona znajduje się w pamięci operacyjnej. Gdy następuje odwołanie do niezaładowanej strony, sprzęt stronicujący wykrywa nieustawiony bit i generuje błąd braku strony.

System operacyjny obsługuje błąd braku strony. Sprawdzane jest, czy odwołanie jest dozwolone, po czym znajdowana jest wolna ramka i ewentualne następuje sprowadzenie strony z dysku.

To było na egzaminie

W systemie z wirtualną pamięcią i stronicowaniem strona ma rozmiar 2MiB, adres wirtualny – 48 bitów, adres fizyczny – 40 bitów. Wtedy w systemie mogą istnieć

- A. proces z adresem wirtualnym $0x00c00f04$ zmapowanym na adres $0x00d00004$
- B. proces z adresem wirtualnym $0x0a000004$ zmapowanym na adres $0x00800004$
- C. proces z adresem wirtualnym $0x000c0000$ oraz proces z adresem wirtualnym $0x001a0000$ zmapowanymi na ten sam adres fizyczny $0x00140000$

Strona ma rozmiar 2MiB, zatem offset będzie miał 21 bitów (1MiB to 2^{20} bajtów).

- A. Adres wirtualny $0x00c00f04$ ma offset $0x000f04$, adres fizyczny ma $0x00d00004$ ma offset $0x100004$, zatem takie mapowanie jest niepoprawne
- B. Adres wirtualny $0x0a000004$ ma offset $0x000004$, adres fizyczny $0x00800004$ ma offset $0x000004$, zatem takie mapowanie mogło by się wydarzyć.
- C. Nie, bo offsety się nie zgadzają. (Gdyby się zgadzały, to taka sytuacja jest możliwa, na przykład po wywołaniu `fork()`)

Zestaw zadań

15.12. Fragmentacja wewnętrzna pamięci

- A. może wystąpić przy zarządzaniu pamięcią metodą stronicowania
- B. polega na tym, że część obszaru przydzielonego procesowi pozostaje niewykorzystana
- C. może wystąpić przy zarządzaniu pamięcią metodą stref statycznych

15.13. Fragmentacja wewnętrzna to

- A. niemożność przydziału procesowi spójnego kawałka pamięci pomimo posiadania wystarczającej ilości pamięci
- B. występowanie bezużytecznych obszarów pamięci w pamięci procesu
- C. niemożność scalenia wielu małych bloków pamięci w jeden duży

15.14. Tablica stron

- A. zawiera informacje o rozmieszczeniu stron procesu w pamięci fizycznej
- B. jest tworzona podczas komplikacji programu
- C. jest wykorzystywana podczas translacji adresów

15.15. Bufory translacji adresów (TLB)

- A. przechowują dane z tych komórek pamięci, do których procesor odwoływał się niedawno
- B. redukują liczbę odwołań do tablic stron
- C. mogą być realizowane w postaci szybkiej pamięci asocjacyjnej

15.16. W pewnym systemie operacyjnym z wirtualizacją pamięci strona ma rozmiar 8KiB, adres wirtualny jest 56-bitowy, a adres fizyczny jest 44-bitowy. Istnieje taka konfiguracja mapowania adresów dwóch różnych procesów A i B, że

- A. adres wirtualny $(123456)_{16}$ procesu A jest odwzorowywany na adres fizyczny $(312456)_{16}$
- B. adres wirtualny $(123456)_{16}$ procesu B jest odwzorowywany na adres fizyczny $(321456)_{16}$

- C. zarówno adres wirtualny $(abcdef)_{16}$ procesu A, jak i adres wirtualny $(abcdef)_{16}$ procesu B są odwzorowywane na ten sam adres fizyczny $(abcdef)_{16}$
- 15.17.** Mamy proces z 52-bitową pamięcią wirtualną i 48-bitową pamięcią fizyczną. Zastosowane jest stronicowanie dwupoziomowe, a wielkość wpisu w tablicy stron i katalogu tablic stron wynosi 16B. Wówczas
- A. strona ma 16 KiB
- B. numer strony ma 16 bitów
- C. istnieje konfiguracja, która adresowi fizycznemu 0x16161616 przypisuje adres wirtualny 0x61611616
- 15.18.** Pewien procesor używa jednopoziomowego mechanizmu stronicowania. Rozmiar tablicy stron jest równy rozmiarowi strony. Jeden wpis w tablicy stron zajmuje 4 bajty. Adres wirtualny ma 32 bity. Wynika z tego, że w tym procesorze
- A. strona ma rozmiar 4 KiB
- B. górne ograniczenie na rozmiar pamięci fizycznej wynosi 4 GiB
- C. procesowi można przydzielić co najwyżej 32768 stron
- 15.19.** W pewnym systemie operacyjnym, w którym zaimplementowano stronicowanie, rozmiar strony wynosi 1 kilobajt. Tablica stron każdego procesu mieści się na jednej stronie, a każdy wpis w tablicy stron zajmuje 2 bajty. Wynika z tego, że:
- A. pamięć operacyjna jest nie większa niż 512 kilobajtów
- B. przestrzeń adresowa procesu jest nie większa niż 512 kilobajtów
- C. przestrzeń adresowa procesu jest nie większa niż 512 bajtów
- 15.20.** Pamięć ze spójnego obszaru o wielkości 1 MiB jest przydzielana metodą bliźniaków. Cztery procesy kolejno poprosiły o przydział pamięci o wielkości: 32 KiB, 256 KiB, 512 KiB i 64 KiB. W wyniku powstania tej sekwencji żądań
- A. każdy proces otrzymał spójny obszar pamięci
- B. wszystkie żądania zostały spełnione
- C. największe żądanie, które może teraz zostać spełnione, to prośba o przydział pamięci wielkości 128 KiB
- 15.21.** W pewnym systemie komputerowym, w którym zastosowano dwupoziomowe stronicowanie, adresy logiczne są 48-bitowe, rozmiar strony, katalogu tablic stron i tablicy stron mają 256 KiB, a wpis w katalogu i tablicy zajmuje 8B. Wówczas
- A. offset ma 17 bitów
- B. numer strony ma 16 bitów
- C. numer tablicy stron ma 15 bitów
- 15.22.** W pewnym systemie komputerowym, w którym zastosowano stronicowanie, odwołania do pamięci generowane przez procesy są 16-bitowe. Rozmiar strony wynosi 1 KiB (1024 bajty), a jeden wpis w tablicy stron zajmuje 2 bajty. Wynika z tego, że
- A. tablica stron mieści się na jednej stronie
- B. proces może zaadresować przestrzeń wielkości 512 KiB
- C. tablica stron musi być co najmniej dwupoziomowa

15.5.

System plików

Plik – logiczny pojemnik na dane. System operacyjny udostępnia pewne operacje do zarządzania plikami, ale nie interpretuje ich zawartości, traktując go jako ciąg bajtów.

Dowiązanie – odniesienie pliku do katalogu. Do jednego pliku może istnieć wiele dowiązań.

Informacje o pliku, takie jak liczba dowiązań i położenie bloków z danymi, są przechowywane na dysku w **i-węźle**.

Przed użyciem plik należy otworzyć. Funkcja systemowa `open` pobiera jako argument nazwę ścieżkową pliku, a przekazuje w wyniku **deskryptory**. Deskryptory pliku można traktować jako sesję dostępu do pliku.

Deskryptory mogą być duplikowane, na przykład przez wywołanie `fork()`. Wielokrotne otwieranie tego samego pliku tworzy nowe deskryptory.

W strukturze danych procesów znajdują się tablica z informacjami o otwartych przez proces plikach, której komórki odpowiadają kolejnym deskryptorom. Znajdują tam się odniesienia do systemowej tablicy otwartych plików.

Wywołanie ‘`open()`’ tworzy nową pozycję w systemowej tablicy otwartych plików, ale ‘`fork()`’ tylko zwiększa liczbę odniesień do jednej pozycji w tablicy.

ext2

ext2 to jedna z implementacji systemu plików.

Przy inicjowaniu systemu administrator wybiera optymalny rozmiar **bloku** (zazwyczaj 1-4 KiB) oraz liczbę i-węzłów dla partycji określonego rozmiaru.

Pierwszy blok partycji jest zarezerwowany dla sektora startowego. Reszta partycji jest podzielona na **grupy bloków**. Mapa bitowa opisująca stan zajętości bloków w grupie musi się mieścić w jednym bloku.

Przykład 1.

Jeśli plik ma rozmiar 1KiB, to grupa może mieć rozmiar 8MiB, ponieważ bitmapa bloków posiada $8 \cdot 1024$ bitów, więc w grupie zmieści się $8 \cdot 1024$ bloków po 1KiB każdy.

Maksymalny rozmiar pliku to:

$$\min \left(\left(\frac{b}{4} \right)^3 + \left(\frac{b}{4} \right)^2 + \left(\frac{b}{4} \right) + 12, 2^{41} \right)$$

gdzie b to chyba rozmiar pliku. Dowód pozostawiamy jako ćwiczenie dla czytelnika.

Maksymalny rozmiar partycji zależy od 32-bitowych numerów bloków na dysku, czyli jest to $2^{32} \cdot \text{rozmiar_bloku}$.

15.6.

Rozwiązania

Rozwiązania

15.1. W procesorze x86 wykonano następujące instrukcje:

```
mov al, 3
mov bl, 130
sub al, bl
```

Wtedy

TAK A. w rejestrze `al` jest zapisana wartość -127 przy interpretacji w kodzie uzupełnień do dwóch

NIE B. w rejestrze **a1** jest zapisana wartość 127 przy interpretacji w naturalnym kodzie binarnym

TAK C. ustawione zostaną flagi CF, OF, SF

A. W U2 pod **a1** zapisane jest 3, pod **b1** -126, więc po odjęciu dostaniemy 129, czyli -127 w U2.

B. w NKB pod **a1** zapisane jest 3, pod **b1** 130, więc po odjęciu dostajemy -127, co przeskoczy na 129 w NKB.

C. Flaga CF zostanie ustawiona, ponieważ wynik operacji -127 nie mieści się w zakresie NKB. Flaga OF również zostanie ustawiona, bo przy interpretacji U2 rejestr **b1** przechowuje wartość -126, więc wynikiem operacji będzie 129, co jest spoza zakresu U2 i zostanie przekonwertowane na -127. Flaga SF również zostanie ustawiona, ponieważ jak widzimy, wynikiem operacji jest liczba ujemna, więc pierwszy bit wyniku w interpretacji U2 ustawiony jest na 1.

15.2. Cechą architektury RISC jest

TAK A. zapisywanie wyników instrukcji arytmetyczno-logicznych tylko w rejestrach

TAK B. kodowanie wszystkich instrukcji maszynowych za pomocą tej samej liczby bajtów

NIE C. dopuszczenie adresowania niewyrównanego

Odpowiedzi z tabelki wyżej

15.3. Cechą architektury CISC jest

NIE A. zapisywanie wyników instrukcji arytmetyczno-logicznych tylko w rejestrach

NIE B. kodowanie wszystkich instrukcji maszynowych za pomocą tej samej liczby bajtów

TAK C. dopuszczenie adresowania niewyrównanego

Odpowiedzi z tabelki wyżej

15.4. W czterech kolejnych bajtach pamięci, począwszy od adresu X , znajdują się odpowiednio wartości 1, 3, 5 i 7. Procesor cienkokońcowkowy (ang. *little-endian*) wczytał 32-bitową liczbę spod adresu X , odjął od niej $(16)_{10}$ i zapisał wynik pod adresem X . Po tych operacjach bajt o adresie

TAK A. X zawiera wartość $(F1)_{16}$

NIE B. $X + 1$ zawiera wartość $(03)_{16}$

TAK C. $X + 2$ zawiera wartość $(05)_{16}$

Mamy do czynienia z procesorem cienkokońcowkowym, czyli najmniej znaczące bity są zapisane na początku. Liczba pod X to $1 \cdot 2^0 + 3 \cdot 2^8 + 5 \cdot 2^{16} + 7 \cdot 2^{24}$ i odejmujemy 16. Wystarczy spojrzeć na adresy X i $X + 1$. Pod nimi otrzymamy $1 + 3 \cdot 256 - 16 = 769 - 16 = 753$. $\lfloor \frac{753}{256} \rfloor = 2$. Pod $X + 1$ mamy liczbę 2, a pod X mamy $753 - 2 \cdot 256 = 241 = (F1)_{16}$. Stąd **A.** to prawda, a **B.** to nieprawda. **C.** adres $X + 2$ nie zostanie tknięty, nadal jest ta sama wartość co przed odejmowaniem.

15.5. W ośmiobitowym rejestrze procesora zapisana jest wartość $(10101100)_2$, która interpretowana

NIE A. w naturalnym kodzie binarnym reprezentuje liczbę 162

TAK B. w kodzie uzupełnieniowym do dwójkii reprezentuje liczbę -84

NIE C. w kodzie moduł-znak reprezentuje liczbę -43

A. Liczby w naturalnym kodzie binarnym są interpretowane jako: $\sum_{i=0}^{n-1} b_i 2^i$, zatem wartość z polecenia to będzie $2^7 + 2^5 + 2^3 + 2^2 = 172$

B. Liczby w kodzie uzupełnieniowym do dwójkii są interpretowane jako: $-b_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$, zatem wartość z polecenia to będzie $-2^7 + 2^5 + 2^3 + 2^2 = -84$

C. Liczby w kodzie moduł-znak są interpretowane jako: $(-1)^{b_{n-1}} \cdot \sum_{i=0}^{n-2} b_i 2^i$, zatem wartość z polecenia to będzie $(-1) \cdot (2^5 + 2^3 + 2^2) = -44$

15.6. Przetwarzanie potokowe

- TAK** A. zawsze przyspiesza lub nie spowalnia wykonywania rozkazów
NIE B. niewykorzystywane jest dla rozkazu skoku
NIE C. zostało zastosowane po raz pierwszy w procesorach Pentium

A. IMO kontrowersyjne, bo jednak stosowane jest tłumaczenie rozkazów z CISC na RISC, co zajmuje jakiś czas i może być pechowy case samych jumpów itd., ale niech będzie, że ogólnie przyspiesza.
B. Wykorzystuje się, choć są problematyczne. Dziś korzysta się też z tzw. predykcji skoków.
C. Bez komentarza.

15.7. Każdy procesor 32-bitowy ma

- NIE** A. 32-bitową zewnętrzną szynę danych
NIE B. 32-bitową zewnętrzną szynę adresową
TAK C. 32-bitowe rejestrory danych

15.8. Proces jest w stanie

- TAK** A. aktywnym, gdy jest obecnie wykonywany przez procesor
TAK B. wstrzymanym, gdy czeka na zakończenie operacji wejścia-wyjścia, lub czeka na jakieś zdarenie
NIE C. gotowym, gdy zakończył się i czeka na odczytanie końcowego stanu przez rodzica
A. i B. z definicji. C. opisuje proces w stanie „zakończony”.

15.9. W chwili 0 w kolejce znajdują się procesy z następującą długością najbliższej fazy procesora:

$$P = 10, \quad Q = 4, \quad R = 2, \quad S = 6$$

- NIE** A. w strategii szeregowania z podziałem czasu (ang. *time sharing*) proces R kończy swój ostatni przydział procesora w chwili 7
TAK B. w strategii szeregowania rotacyjnego (ang. *round-robin*) z kwantem czasu 3 proces Q kończy swój ostatni przydział procesora w chwili 15
TAK C. w strategii szeregowania SJF (ang. *shortest job first*) bez wywłaszczenia proces P dostaje procesor jako ostatni

A. Time sharing jest tak naprawdę round-robinem z epsilonowym kwantem czasu. Z tego powodu proces R kończy swój ostatni przydział procesora w chwili $4 \cdot 2 = 8$.

B. Procesy używają procesora w następującej kolejności:

- P używa procesora przez 3 kwanty czasu;
- Q używa procesora przez 3 kwanty czasu;
- R używa procesora przez 2 kwanty czasu (i kończy się);
- S używa procesora przez 3 kwanty czasu;
- P używa procesora przez 3 kwanty czasu;
- Q używa procesora przez 1 kwant czasu (i kończy się).

C. Z definicji SJF P jako proces zajmujący procesor na najdłużej wykona się jako ostatni.

15.10. W grafie przydziału zasobów istnieje cykl. Wynika z tego, że

- NIE** A. system jest w stanie blokady
NIE B. system jest w stanie bezpiecznym
TAK C. istnieje proces oczekujący na zasoby

- A. To, że istnieje cykl, nie znaczy, że mamy zakleszczenie (patrz: rysunek przy warunkach koniecznych zakleszczenia).
- B. To, że mamy cykl, **może** znaczyć, że jest zakleszczenie, a to z pewnością nie jest stan bezpieczny.
- C. Gdyby nikt nie czekał na zasoby, to cykl nie jest możliwy.

15.11. Warunkiem wystarczającym powstania zakleszczenia jest

- NIE A. niepodzielność co najmniej jednego zasobu
- NIE B. brak wywłaszczeń
- NIE C. przetrzymywanie i oczekiwanie

Wszystkie te warunki są warunkami koniecznymi, ale żaden z nich nie jest wystarczający.

15.12. Fragmentacja wewnętrzna pamięci

- TAK A. może wystąpić przy zarządzaniu pamięcią metodą stronicowania
- TAK B. polega na tym, że część obszaru przydzielonego procesowi pozostaje niewykorzystana
- TAK C. może wystąpić przy zarządzaniu pamięcią metodą stref statycznych
- A. Niewielka, ale występuje.
- B. Dokładnie tak.
- C. Występuje, i to w większym stopniu niż przy stronicowaniu.

15.13. Fragmentacja wewnętrzna to

- NIE A. niemożność przydziału procesowi spójnego kawałka pamięci pomimo posiadania wystarczającej ilości pamięci
- TAK B. występowanie bezużytecznych obszarów pamięci w pamięci procesu
- NIE C. niemożność scalenia wielu małych bloków pamięci w jeden duży
- A. Jest to definicja fragmentacji zewnętrznej.
- B. Taka jest właściwie definicja.
- C. Nie jest to fragmentacja wewnętrzna, tylko przykład patologii w algorytmie bliźniaków.

15.14. Tablica stron

- TAK A. zawiera informacje o rozmieszczeniu stron procesu w pamięci fizycznej
- NIE B. jest tworzona podczas komplikacji programu
- TAK C. jest wykorzystywana podczas translacji adresów
- A. Z definicji tym się zajmuje, tak samo C. Skoro każdy proces ma swoją tablicę stron, to ciężko, żeby była ona tworzona w czasie komplikacji programu, kiedy nie wiadomo ile procesów w trakcie powstanie, zatem B. to nieprawda.

15.15. Bufory translacji adresów (TLB)

- NIE A. przechowują dane z tych komórek pamięci, do których procesor odwoływał się niedawno
- TAK B. redukują liczbę odwołań do tablic stron
- TAK C. mogą być realizowane w postaci szybkiej pamięci asocjacyjnej
- A. Nie, nie zawiera danych z tych komórek pamięci (to by zajmowało bardzo dużo pamięci, a TLB są bardzo małe). Zawiera jedynie informacje gdzie te dane są, redukując potrzebę odwoływania się do tablicy stron. Stąd B. to prawda. C. tak po prostu jest.

- 15.16.** W pewnym systemie operacyjnym z wirtualizacją pamięci strona ma rozmiar 8KiB, adres wirtualny jest 56-bitowy, a adres fizyczny jest 44-bitowy. Istnieje taka konfiguracja mapowania adresów dwóch różnych procesów *A* i *B*, że

- NIE** A. adres wirtualny $(123456)_{16}$ procesu *A* jest odwzorowywany na adres fizyczny $(312456)_{16}$
- TAK** B. adres wirtualny $(123456)_{16}$ procesu *B* jest odwzorowywany na adres fizyczny $(321456)_{16}$
- TAK** C. zarówno adres wirtualny $(abcdef)_{16}$ procesu *A*, jak i adres wirtualny $(abcdef)_{16}$ procesu *B* są odwzorowywane na ten sam adres fizyczny $(abcdef)_{16}$

Strona ma rozmiar 8KiB, czyli $2^{13}B$. Oznacza to, że offset ma 13 bitów. Do zweryfikowania odpowiedzi będziemy liczyć offsety adresu wirtualnego i fizycznego, czyli najmłodsze 13 bitów adresów.

- A. Offset adresu wirtualnego to $(1456)_{16}$, a fizycznego $(0456)_{16}$, więc takie mapowanie jest niemożliwe.
- B. Offset adresu wirtualnego to $(1456)_{16}$, a fizycznego $(1456)_{16}$. Są równe, więc takie mapowanie jest możliwe.
- C. Każdy proces ma swoją osobną tablicę stron, a offsety wszystkich wymienionych tu adresów są sobie równe. Jednakże, dwa różne procesy nie mogą wskazywać na ten sam adres fizyczny, ale taka sytuacja może się wydarzyć po wywołaniu `fork()`.

- 15.17.** Mamy proces z 52-bitową pamięcią wirtualną i 48-bitową pamięcią fizyczną. Zastosowane jest stronicowanie dwupoziomowe, a wielkość wpisu w tablicy stron i katalogu tablic stron wynosi 16B. Wówczas

- NIE** A. strona ma 16 KiB
- TAK** B. numer strony ma 16 bitów
- NIE** C. istnieje konfiguracja, która adresowi fizycznemu $0x16161616$ przypisuje adres wirtualny $0x61611616$

A. Założymy że strona ma rozmiar $S = 2^k B$. Wobec tego wpisów w tablicy jest $\frac{2^k}{2^4}$, tak samo w katalogu tablic. Różnych stron może być tyle ile jest wpisów więc jest to $\frac{S^2}{2^8}$. Żeby je zakodować potrzebujemy $\log(\frac{S^2}{2^8}) = 2k - 8$. Dodatkowo skoro strona mieści $2^k B$ to potrzebujemy k bitów (offset) żeby mieć dostęp do konkretnego bajtu w stronie. Otrzymujemy więc równość $2k - 8 + k = 52 \implies k = 20$. Zatem strona ma rozmiar $2^{20}B$ czyli 1MiB.

- B. Skoro strona ma $2^{20} B$ a wpis zajmuje $2^4 B$ to strona jest 2^{16} więc numer strony zawiera 16 bitów.
- C. 20 bitów na offset to 5 ostatnich znaków. Widzimy że 5 bit od końca różni się w obu tych adresach więc NIE.

- 15.18.** Pewien procesor używa jednopoziomowego mechanizmu stronicowania. Rozmiar tablicy stron jest równy rozmiarowi strony. Jeden wpis w tablicy stron zajmuje 4 bajty. Adres wirtualny ma 32 bity. Wynika z tego, że w tym procesorze

- NIE** A. strona ma rozmiar 4 KiB
- NIE** B. górne ograniczenie na rozmiar pamięci fizycznej wynosi 4 GiB
- TAK** C. procesowi można przydzielić co najwyżej 32768 stron

A. Założymy że strona ma rozmiar $S = 2^k B$. Wobec tego wpisów w tablicy jest $\frac{2^k}{4}$. Różnych stron może być tyle ile jest wpisów, a żeby je zakodować potrzebujemy $\log(\frac{2^k}{4}) = k - 2$ bitów. Dodatkowo, skoro strona mieści $2^k B$ to potrzebujemy k bitów (offset), żeby mieć dostęp do konkretnego bajtu w stronie. Otrzymujemy więc równość $k - 2 + k = 32 \implies k = 17$. Zatem strona ma rozmiar $2^{17}B$ czyli 128 KiB.

B. Pamięć fizyczna może przekroczyć 4 GiB, patrz: [link](#).

C. Wpisów jest $2^{15} = 32768$, więc tyle można przydzielić stron procesowi.

- 15.19.** W pewnym systemie operacyjnym, w którym zaimplementowano stronicowanie, rozmiar strony wynosi 1 kilobajt. Tablica stron każdego procesu mieści się na jednej stronie, a każdy wpis w tablicy stron zajmuje 2 bajty. Wynika z tego, że:

- NIE A. pamięć operacyjna jest nie większa niż 512 kilobajtów
 TAK B. przestrzeń adresowa procesu jest nie większa niż 512 kilobajtów
 NIE C. przestrzeń adresowa procesu jest nie większa niż 512 bajtów

15.20. Pamięć ze spójnego obszaru o wielkości 1 MiB jest przydzielana metodą bliźniaków. Cztery procesy kolejno poprosiły o przydział pamięci o wielkości: 32 KiB, 256 KiB, 512 KiB i 64 KiB. W wyniku tworzenia tej sekwencji żądań

- TAK A. każdy proces otrzymał spójny obszar pamięci
 TAK B. wszystkie żądania zostały spełnione
 TAK C. największe żądanie, które może teraz zostać spełniona, to prośba o przydział pamięci wielkości 128 KiB

Z definicji metody **bliźniaków** wiemy, że po prośbie pierwszego procesu struktura pamięci będzie wyglądała tak:

32 KiB	32 KiB	64 KiB	128 KiB	256 KiB	512 KiB
--------	--------	--------	---------	---------	---------

Zatem przychodzące procesy otrzymają kolejno pierwszy, piąty, szósty oraz trzeci fragment pamięci. Czwarty fragment pozostał niewykorzystany. Zatem wszystkie odpowiedzi to TAK.

15.21. W pewnym systemie komputerowym, w którym zastosowano dwupoziomowe stronicowanie, adresy logiczne są 48-bitowe, rozmiar strony, katalogu tablic stron i tablicy stron mają 256 KiB, a wpis w katalogu i tablicy zajmuje 8B. Wówczas

- NIE A. offset ma 17 bitów
 NIE B. numer strony ma 16 bitów
 TAK C. numer tablicy stron ma 15 bitów

- A. Rozmiar strony to 256 KiB, więc 2^18 . Znaczy to, że offset ma 18 bitów.
B. Skoro offset ma 18 bitów, a adres logiczny 48, to numer strony ma połowę pozostałych bitów, czyli $(48 - 18)/2 = 15$.
C. Jest to 15 bitów, tyle samo co numer strony.

15.22. W pewnym systemie komputerowym, w którym zastosowano stronicowanie, odwołania do pamięci generowane przez procesy są 16-bitowe. Rozmiar strony wynosi 1 KiB (1024 bajty), a jeden wpis w tablicy stron zajmuje 2 bajty. Wynika z tego, że

- TAK A. tablica stron mieści się na jednej stronie
 NIE B. proces może zaadresować przestrzeń wielkości 512 KiB
 NIE C. tablica stron musi być co najmniej dwupoziomowa

- A. Mamy do dyspozycji 16 bitów. Tablica ma $2^{10}B = 1024B$ więc offset zajmuje 10 bitów. Zatem zostaje 6 bitów na tablicę stron, jeden wpis zajmuje 2B więc będzie $2^5 = 32$ wpisów o wielkości 2B, więc spokojnie się zmieści na jednej stronie.

- Przyp. red.: powyższe rozumowanie jest potencjalnie błędne: 6 bitów na tablicę stron oznacza 6 bitów na indeks w tablicy stron, tj. stron może być 64 (rozmiar wpisu nie ma tu nic do rzeczy)**
B. Skoro do dyspozycji jest 16 bitów, to może zaadresować max $2^{16} = 64$ KiB.
C. Skoro tablica stron mieści się na jednej stronie to nie musi być dwupoziomowa.

16

Bezpieczeństwo systemów komputerowych

Materiały teoretyczne zostały opracowane na podstawie slajdów Tomasza Kazany.

Kod do Moodle: BSK2022.8b52-g8

■ Podstawa programowa

1. Podstawy **kryptografii**.
2. Infrastruktura **klucza publicznego**.
3. **Modele i klasy bezpieczeństwa** systemów informatycznych.
4. **Modele uwierzytelniania**, strategie kontroli dostępu.
5. **Bezpieczeństwo protokołów komunikacyjnych** i aplikacji.
6. Praktyczna **ochrona systemów operacyjnych** i usług aplikacyjnych z wykorzystaniem izolacji, ścian ogniowych, VPN, TLS, PGP.
7. Problematyka **bezpiecznego programowania**.
8. Zagrożenia związane z **przestępcością elektroniczną**.

16.1.

Klucz symetryczny

■ Historyczne szyfrowanie z kluczem symetrycznym

Szyfr Cezara:

np. A -> E, B-> F, ... - przesuwamy alfabet np. o 4

Ogólniej **szyfr zastępowy (permutacyjny)**

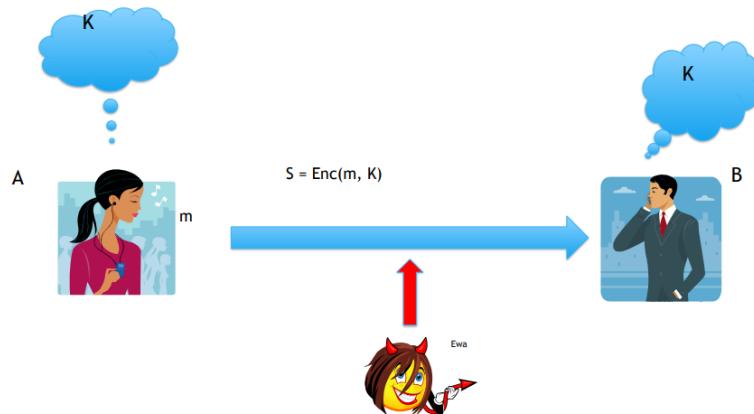
np. E -> W, S -> E, A -> D, ...

ESSA -> WEED

Wszystkie te szyfry działają o wcześniej potajemnie ustalony **tajny klucz (klucz prywatny)**.

Enigma - maszyna szyfrująca używana przez Niemców podczas WWII też tak działała.

Tradycyjne szyfrowanie symetryczne



Formalnie:

m - wiadomość

K - klucz symetryczny

S - szyfrogram

Szyfrowanie: $S = \text{Enc}(m, K)$

Deszyfrowanie: $\text{Dec}(S, K)$

Poprawność: $\text{Dec}(\text{Enc}(m, K), K) = m$

Nawet jak Ewa zna szyfrogram, funkcje Dec i Enc to bez K nie jest w stanie rozszyfrować wiadomości.

Poprawna definicja bezpieczeństwa to, że na podstawie S przeciwnik nie pozna żadnej dodatkowej informacji o m - formalnie: rozkład m i $\text{Enc}(m, K)$ są niezależne.

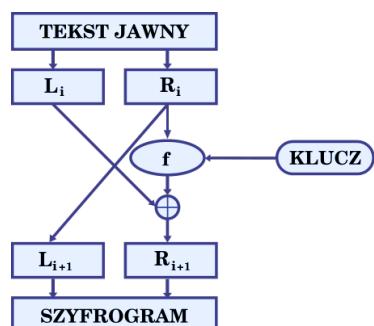
Istnieje idealnie bezpieczny szyfr - OTP (One Time Pad) polegający na XORowaniu wiadomości i klucza prywatnego ale jest useless bo klucza można użyć tylko raz.

Współczesne szyfry symetryczne

DES - szyfr blokowy oparty o sieć Feistela zatwierdzony przez NSA w latach 70-tych. Polega na szyfrowaniu bloków tej samej długości

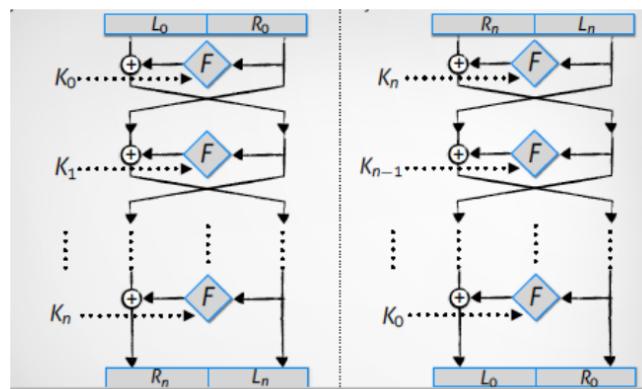
Zasada konfuzji - zależność między bitami szyfrogramu a klucza powinna być możliwie skomplikowana, w szczególności NIE liniowa

Zasada dyfuzji - zmiana jednego bitu tekstu jawnego powinna zawsze spowodować zmianę około połowy bitów szyfrogramu

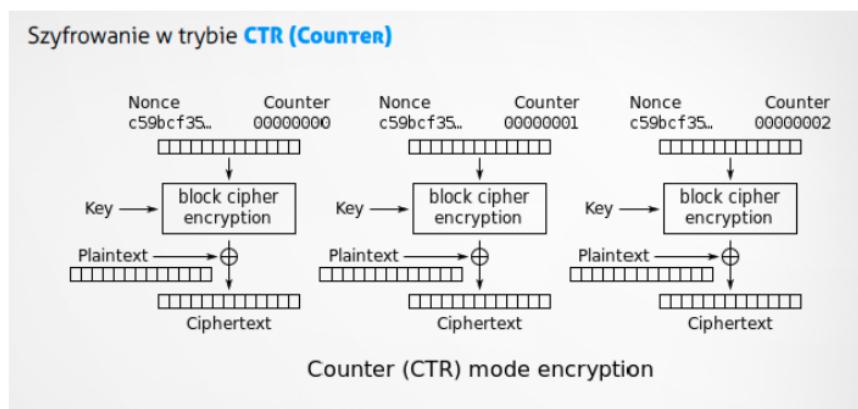


Sieć Feistela - deszyfrowanie

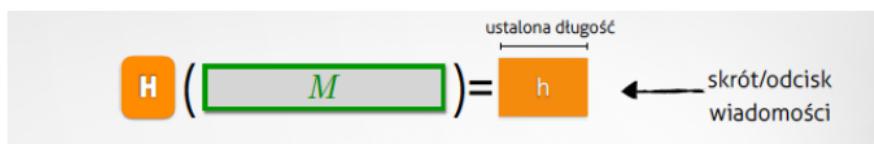
nie wymaga odwrócenia funkcji F
z lewej szyfrowanie, z prawej deszyfrowanie



DES jest przestarzały, obecnie używa się **3DES** (szyfrowanie 3 razy z różnymi kluczami) albo **AES**
Szyfrowanie tych samych bloków jeden po drugim może dać dodatkową informację więc stosuje się sztuczki
np. **CTR (counter)**



■ Haszowanie, MAC



Funkcja haszująca (**funkcja skrótu**) jest:

- deterministyczna
- łatwo obliczalna
- trudno odwracalna
- odporna na kolizje

Przykłady funkcji hashujących:

- SHA-1 - 160 bitów

- SHA-2 - 224, 256, 384, 512 bitów
- SHA-3 - 224, 256, 384, 512 bitów

Z hashowania korzysta się w generowaniu **MACa** czyli krótkiej informacji umożliwiającej weryfikację autentyczności

$$t = \text{MAC}(m, K)$$

MAC pełni rolę podpisu cyfrowego - nie służy do utajniania wiadomości. Naiwne MACi są podatne na ataki więc w protokołach TLS/SSL i IPSec stosuje się **HMAC**.

HMAC - standard stosowany w protokołach TLS/SSL oraz IPSec

- $\text{HMAC}(K, m) = H(K \oplus opad || H((K \oplus ipad) || m))$
- gdzie
 - ipad = 0x363636...3636
 - opad = 0x5c5c5c...5c5c

Zestaw zadań

16.1. Od kryptograficznej funkcji skrótu f wymagamy, aby dla danego x trudne obliczeniowo było znalezienie takiego $y \neq x$, że

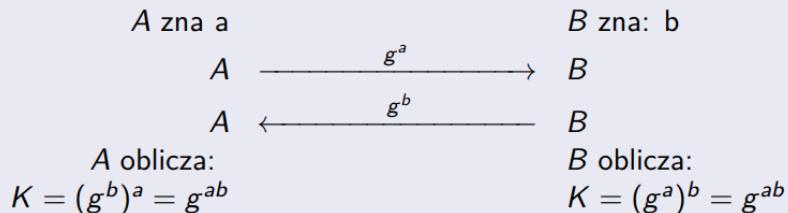
- A. $f(f(y)) = x$
 B. $f(x) = f(y)$
 C. $f(y) = x$

16.2.

Protokół Diffiego-Hellmana, problem logarytmu dyskretnego

Jak dotychczas zakładaliśmy, że obie strony mają utajniony klucz. Ale jak tego dokonać, żeby było bezpieczne? Z pomocą przychodzi protokół Diffiego-Hellmana.

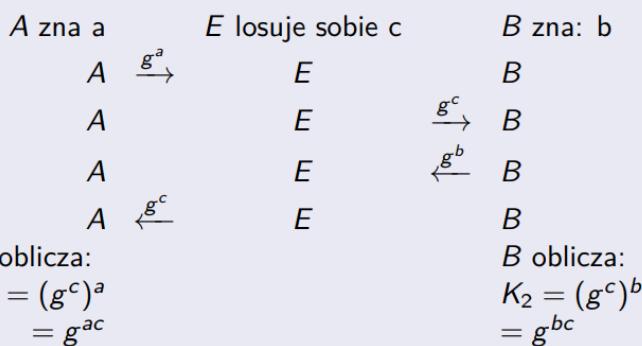
Wybieramy grupę cykliczną G z generatorem g . Powszechnie stosuje się grupy mnożenia modulo liczba pierwsza i grupy dodawania punktów na krzywej eliptycznej.



Podsłuchujący zna g, g^a, g^b ale nie pozwala mu to obliczyć K - przy założeniu że wybieramy grupę dla której problem logarytmu dyskretnego (dla danego g, g^x obliczyć x) jest obliczeniowo trudny.

Generalnie DH jest fajny ale mamy w nim dosyć mocne założenie że podsłuchujący nie może ingerować w przesyłane wiadomości. Nie zawsze jest to prawda i przykładem jest atak "Man in the middle"

Protokół ataku:



Ewa jest w stanie obliczyć zarówno K_1 jak i K_2 .

W 2015 odkryto że agencje rządowe są w stanie obliczyć logarytm dyskretny liczby 1024-bitowej więc trzeba stosować większe.

Ostatecznie jednak protokół Diffiego-Hellmana nie jest idealny ale nadal się go stosuje w połączeniu z kryptografią klucza publicznego.

Zestaw zadań

16.2. Po wykonaniu algorytmu Diffiego-Hellmana

- A. obie strony znają wspólny sekret
- B. obie strony poznają dokładnie jeden bit informacji
- C. obie strony otrzymały komunikat wysłany przez drugą stronę, którego wcześniej nie znały, a który jest chroniony przed odczytem przez trzecią stronę

16.3.

Kryptografia klucza publicznego

W takiej kryptografii mamy podział na pary **PK (public key)** oraz **SK (secret key)**. Każdy może zaszyfrować swoją wiadomość używając PK ale tylko właściciel SK może ją odszyfrować, dlatego że odwrócenie funkcji szyfrującej opiera się na problemie faktoryzacji.

RSA

RSA to przykład szyfru asymetrycznego

p, q - 2 duże liczby pierwsze

$$N = p \cdot q$$

$$\varphi(N) = (p-1)(q-1)$$

e - dowolna liczba pierwsza względnie pierwsza z $\varphi(N)$

$$d = e^{-1} \pmod{\varphi(N)}$$

$$PK = (N, e)$$

$$SK = (N, d)$$

Szyfrogram: C

Szyfrowanie: $C = M^e \pmod{N}$

Deszyfrowanie: $M' = C^d \pmod{N}$

Jak działa to $M' = M$

■ Podpis cyfrowy

Podpis cyfrowy zapewnia:

- autentyczność - możliwość weryfikacji autora
- niezaprzeczalność - autor nie może się wyprzeć podpisu
- integralność - nie da się dokonać nieautoryzowanej manipulacji podpisanej wiadomości

Podpis: $\sigma = \text{Sig}(\text{SK}, M)$

$\text{Ver}(\text{PK}, \sigma, M) \in 0,1$

Bardzo trudno stworzyć poprawną parę (σ, M) bez znania SK $\text{Ver}(\text{PK}, \text{Sig}(\text{SK}, M), M) = 1$

Zwykłe RSA jest za słabe dla podpisu cyfrowego:

- kowalność - znając podpisy σ_1, σ_2 dla M_1, M_2 można wygenerować podpis dla $M_1 \cdot M_2$
- można generować podpisy dla losowych wiadomości
- podatność na ataki powtórzeniowe - raz podpisana wiadomość (np. o treści TAK/WYRAŻAM ZGODĘ etc) można wielokrotnie wykorzystać

W praktyce:

- dodaje się hasz - zapobiega kowalności i generowaniu losowych wiadomości.
- wiadomości się ulasawia i dodaje timestampy - zapobiega atakom powtórzeniowym

Obecnie najczęściej używa się RSA-PSS.

Istnieje również podpis El-Gamala.

■ Certyfikat kluczy publicznych

Skąd wiadomo że jakiś klucz publiczny jest legit? Istnieją 2 modele certyfikacji

- infrastruktura klucza publicznego PKI - certyfikaty wydawane przez zaufane urzędy
- sieć zaufania np. w PGP - użytkownicy wzajemnie poświadczają

Zestaw zadań

16.3. W kryptografii z kluczem publicznym klucz

- A. publiczny służy do szyfrowania i podpisywania cyfrowego
- B. prywatny służy do deszyfrowania i weryfikowania podpisu cyfrowego
- C. prywatny służy do szyfrowania, a publiczny do deszyfrowania

16.4. Współczesne rekomendacje dla kluczy publicznych używanych w szyfrowaniu RSA określają, że dla zapewnienia dostatecznie bezpiecznej poufnej komunikacji wystarczająca jest długość klucza równa

- A. 2048 bitów
- B. 1024 bitów

C. 256 bitów

16.5. Do zabezpieczenia komunikacji od ataku powtórzeniowego stosuje się technikę

- A. dodawania numeru kolejnego pakietu do każdego komunikatu
- B. dodawania timestampów do każdego komunikatu
- C. stosowania kryptograficznej funkcji haszującej 2 razy

16.4.

Zastosowania kryptografii klasycznej

Jednym z zastosowań kryptografii klasycznej są wykorzystywane aktualnie sposoby na bezpieczną komunikację w Internecie.

Kerberos - protokół uwierzytelniania z wykorzystaniem KDC (centrum dystrybucji kluczy). Szyfrowanie odbywa się przy użyciu kryptografii symetrycznej. Protokół polega na tym, że każdy użytkownik ma tajny klucz, znany KDC. Gdy użytkownik A chce się skomunikować z B, to zgłasza się do KDC z informacją o planowanej komunikacji. Następnie KDC wysyła do A nowo wylosowany klucz K_{AB} oraz komunikat $C = Enc_{K_B}\{K_{AB}, A\}$ (K_B to tajny klucz B oraz wszystkie wiadomości są zaszyfrowane tajnym kluczem użytkownika A). A wysyła do B komunikat C, dzięki czemu A i B mogą się bezpiecznie komunikować za pomocą klucza K_{AB} .

Wzbogacenie standardowego Internetu o szyfrowanie odbywa się poprzez zmianę protokołu komunikacji w różnych warstwach sieciowych. Szyfrowanie nie jest ustalone z góry, przed komunikacją następuje faza negocjacji, w której wybierana jest metoda szyfrowania (tzw. IKE - Internet Key Exchange). Wyróżnia się dwa typy rozszerzonego protokołu:

- **IPsec** - na poziomie sieci, obowiązkowy w IPv6, dodaje szyfrowanie do wysyłanych pakietów (nagłówek IP oraz nagłówek AH/ESP z danymi wykorzystywanymi do uwierzytelniania są jawne, pozostałe dane są zaszyfrowane)
- **TLS** (Transport Layer Security) - podobnie jak IPsec, ale na poziomie sesji. Może zapobiegać atakom man-in-the-middle.

VPN (Wirtualna Sieć Prywatna) - utworzenie bezpiecznej sieci wewnętrznej niezabezpieczonego Internetu, technicznie realizowane za pomocą IPsec lub TLS.

TOR (The Onion Router) - celem tego systemu jest stworzenie rozwiązania, które utajnia nie tylko treść, ale również sam fakt komunikacji między stronami.

16.5.

Ataki i obrona

Ataki na aplikacje WWW

Ataki na aplikacje WWW:

- **atak XSS** (skróty atak skryptowy) - dotyczy serwisów, które umożliwiają jednemu klientowi uruchamianie skryptów stworzonych przez drugiego klienta,
- **atak XSSRF, CSRF** (skróty fałszowanie żądania) - użytkownik zalogowany do pewnego serwisu uruchamia spłaszcowanego odnośnik (może być uruchomiony gdzieś indziej, np. w innej zakładce), który powoduje wykonanie niechcianej akcji w tym serwisie,
- **wstrzykiwanie kodu SQL**

Przykład 1.

Rozważamy serwer, który wykonuje zapytanie SQL podstawiając pod zmienne \$login i \$password dane wprowadzone przez użytkownika i akceptując użytkownika, jeśli zapytanie zwróci niezerową liczbę:

```
SELECT count(*) FROM client WHERE name='$login' and pwd='$password';
```

Możliwy atak: jako login oraz password wpisujemy: d' or '1'='1. Wtedy nasze zapytanie będzie miało postać:

```
SELECT count(*) FROM client WHERE name='d' or '1'='1' and pwd='d' or '1'='1';
```

Zapytanie to (dla niepustej bazy) nigdy nie zwróci 0, zatem użytkownik zostanie zawsze wpuszczony.

Inne ataki

Wirus/robak to program, który się samoreplikuje. Robak komunikuje się i przyjmuje polecenia od centrali przez sieć, natomiast wirus raz wypuszczony działa samodzielnie.

Atak DDoS (Distributed Denial of Service) - świadome zmasowane zwiększenie ruchu sieciowego (w celu spowolnienia lub czasowego wyłączenia usługi). Jedną z popularnych technik ataku jest wysyłanie dużej ilości małych pakietów do botnetu (duża grupa zainfekowanych komputerów), który z kolei wysyła większe pakiety do atakowanych serwerów. Zaatakowane serwery przestają odpowiadać lub ich łączna internetowa osiągają limit przepustowości.

Reverse engineering - próba analizy kodu binarnego w celu znalezienia w nim luk i zasad kodowanych danych.

Atak buffer overflow - idea polega na tym, że wprowadzamy jakoś do pamięci atakowanego programu złośliwy kod, tzw. shell code (np. poprzez podanie go jako argument wejściowy). Następnie zmieniamy poprawne działanie atakowanego programu zmieniając mechanizmy sterowania wykonaniem programu (np. poprzez nadpisanie w pamięci adresu powrotu z wykonywanej funkcji). To właśnie da się osiągnąć dzięki między innymi przepełnieniu buforu (buffer overflow), który występuje wtedy, gdy zapisujemy do wyznaczonego buforu większej ilości danych, niż była zarezerwowana. Dzięki temu możemy stworzyć na tyle dużo obiektów w ramach sterty, aby zahaczyć o stos i odpowiednio nadpisać adres powrotu do wykonywanej funkcji.

Techniki obrony przed atakiem buffer overflow:

- **kanarek** - unikalna wartość umieszczana na stosie między zmiennymi a adresem powrotu, sprawdzana przed wykonaniem powrotu,
- DEP (data execution prevention) - OS nie pozwala na wykonanie kodu umieszczonego dynamicznie na stercie (choć musi zrobić wyjątek dla bibliotek ładowanych dynamicznie).

ROP (return oriented programming) - atak polegający na nie korzystaniu z shell code, tylko skakaniu do kodu atakowanego programu w odpowiednio dobrane miejsca, z których atakujący skleja złośliwą funkcjonalność.

Buffer overread - atakujący, manipulując wskaźnikami, dostaje możliwość odczytu pamięci, której nie powinien widzieć.

Zestaw zadań

16.6. Atak DDoS polegający na wysyłaniu wielu nieukończonych fragmentów pakietów IP działa, ponieważ

- A. na komputerze docelowym wyczerpana zostaje limit zasobów na obsługę pakietów
- B. na oprogramowaniu pewnego routera pośredniczącego wyczerpana zostaje limit zasobów na obsługę pakietów

C. przekroczony zostaje MTU

16.7. Aby utrudnić dokonywanie ataków poprzez przepełnienie stosu (ang. *stack overflow*), można

- A. przekazywać argumenty w rejestrach zamiast na stosie
- B. alokować wszystkie bufore na stercie zamiast na stosie
- C. użyć architektury RISC, w której występuje specjalny rejestr LR (ang. *link register*), który zawiera adres powrotu do funkcji wołającej

16.6.

Rozwiązańia

16.1. Od kryptograficznej funkcji skrótu f wymagamy, aby dla danego x trudne obliczeniowo było znalezienie takiego $y \neq x$, że

A. $f(f(y)) = x$

B. $f(x) = f(y)$

C. $f(y) = x$

A. Chcemy, żeby obliczeniowo trudne było znalezienie jakiejkolwiek własności f , której nie powiniśmy znać. W tym przypadku potrafilibyśmy odszyfrowywać wiadomości zaszyfrowane dwukrotnie, czego raczej nie chcemy.

B. W tej sytuacji, łatwo by było znaleźć inną wiadomość, która jest taka sama jak oryginalna po zaszyfrowaniu. Można by wtedy przechwycić oryginalną wiadomość, a do odbiorcy wysłać jakiś śmiec.

C. Tutaj to już w ogóle tragedia, bo moglibyśmy czytać zaszyfrowane wiadomości.

16.2. Po wykonaniu algorytmu Diffiego-Hellmana

A. obie strony znają wspólny sekret

B. obie strony poznają dokładnie jeden bit informacji

C. obie strony otrzymały komunikat wysłany przez drugą stronę, którego wcześniej nie znały, a który jest chroniony przed odczytem przez trzecią stronę

A. Tak, obydwa znają g^{ab} , dzięki czemu mogą się komunikować.

B. Nie bardzo.

C. Zdanie jest obiecujące, ale ostatnia część „który jest chroniony przed odczytem przez trzecią stronę” jest nieprawdą.

16.3. W kriptografii z kluczem publicznym klucz

A. publiczny służy do szyfrowania i podpisywania cyfrowego

B. prywatny służy do deszyfrowania i weryfikowania podpisu cyfrowego

C. prywatny służy do szyfrowania, a publiczny do deszyfrowania

A. Do podpisywania cyfrowego służy klucz prywatny.

B. Do weryfikowania podpisu służy klucz publiczny.

C. Na odwrót.

16.4. Współczesne rekomendacje dla kluczy publicznych używanych w szyfrowaniu RSA określają, że dla zapewnienia dostatecznie bezpiecznej poufnej komunikacji wystarczająca jest długość klucza równa

- TAK A. 2048 bitów
- NIE B. 1024 bitów
- NIE C. 256 bitów

Jak wiemy, agencje rządowe potrafią obliczyć logarytm dyskretny liczby 1024-bitowej w skończonym czasie, ale 2048 bitów jest już bezpieczne.

16.5. Do zabezpieczenia komunikacji od ataku powtórzeniowego stosuje się technikę

- TAK A. dodawania numeru kolejnego pakietu do każdego komunikatu
- TAK B. dodawania timestampów do każdego komunikatu
- NIE C. stosowania kryptograficznej funkcji haszującej 2 razy

A. Numery kolejnych pakietów mogą działać jak swego rodzaju timestampy, o których mowa w podpunkcie poniżej

B. Dodanie timestampów pozwala na uniknięcie ataku powtórzeniowego, ze względu na to, że powtórnie użyty komunikat będzie miał zapisany inny timestamp niż czas ponownego użycia

C. Podwójne zastosowanie funkcji haszującej nie pozwoli na uniknięcie ponownego wykorzystania komunikatu

16.6. Atak DDoS polegający na wysyłaniu wielu nieukończonych fragmentów pakietów IP działa, ponieważ

- NIE A. na komputerze docelowym wyczerpany zostaje limit zasobów na obsługę pakietów
- TAK B. na oprogramowaniu pewnego routera pośredniczącego wyczerpany zostaje limit zasobów na obsługę pakietów
- NIE C. przekroczony zostaje MTU

16.7. Aby utrudnić dokonywanie ataków poprzez przepełnienie stosu (ang. *stack overflow*), można

- TAK? A. przekazywać argumenty w rejestrach zamiast na stosie
- TAK? B. alokować wszystkie bufory na stercie zamiast na stosie
- NIE? C. użyć architektury RISC, w której występuje specjalny rejestr LR (ang. *link register*), który zawiera adres powrotu do funkcji wołającej