Politechnika Wrocławska

Struktury danych i złożoność obliczeniowa Wykład 2.

Prof. dr hab. inż. Jan Magott

Złożoność obliczeniowa czasowa

Problemy rozstrzygalne i nierozstrzygalne

Problem rozstrzygalny (rozwiązywalny) to problem, dla którego istnieje algorytm znajdujący rozwiązanie w skończonej liczbie kroków.

Problem nierozstrzygalny (nierozwiązywalny) to problem, dla którego udowodniono, że nie istnieje algorytm znajdujący rozwiązanie w skończonej liczbie kroków.

Złożoność obliczeniową badamy dla problemów rozstrzygalnych.

Złożoność obliczeniowa czasowa

Rozmiar problemu dla danych w postaci zbioru

$$\{x_1, x_2, \dots, x_n\}$$

Przykłady rozmiaru:

- Liczba elementów czyli n,
- Liczba znaków w dziesiętnym kodowaniu n elementów,
- Liczba znaków w dwójkowym kodowaniu n elementów,
- Liczba znaków w jedynkowym kodowaniu n elementów.

Kodowanie

$$(7)_{10} = (111)_2 = (1111111)_1$$

Złożoność obliczeniowa czasowa

Przykłady problemów o więcej niż jednym rozmiarze danych:

Problem mnożenia macierzy nie kwadratowych – 3 rozmiary,

Problemy grafowe – rozmiarami mogą być: liczba wierzchołków, liczba łuków.

Złożoność obliczeniowa czasowa

Wg teorii złożoności obliczeniowej **efektywnym** jest algorytm o złożoności czasowej ograniczonej od góry przez wielomian od rozmiarów problemu.

Algorytmy o wykładniczej złożoności obliczeniowej uważane są za **nieefektywne**.

Złożoność obliczeniowa czasowa

Złożoność wielomianowa a wykładnicza algorytmu

$n = 100$ - rozmiar problemu

n^2 - złożoność algorytmu wielomianowego (liczba kroków)

2^n - złożoność algorytmu wykładniczego

$ck = 10^{-4}s = 100\mu s$ – czas wykonania jednego kroku

$$n^2 \cdot ck = 100^2 \cdot 10^{-4}s = 1s$$

$$2^n = 2^{100} = (2^{10})^{10} \approx (10^3)^{10} = 10^{30}$$

$$10^{30} \cdot ck = 10^{30} \cdot 10^{-4}s = 10^{26}s \approx 3 \cdot 10^{18} lat$$

$$1rok = 3600 \cdot 24 \cdot 365s \approx 3 \cdot 10^7 s$$

$13{,}82 \cdot 10^9 lat$ - czas życia wszechświata

Złożoność obliczeniowa czasowa

Złożoność wielomianowa a wykładnicza algorytmu

$n = 100$ - rozmiar problemu

n^2 - złożoność algorytmu wielomianowego (liczba kroków)

2^n - złożoność algorytmu wykładniczego

$ck = 10^{-4}s = 100\mu s$ – czas wykonania jednego kroku

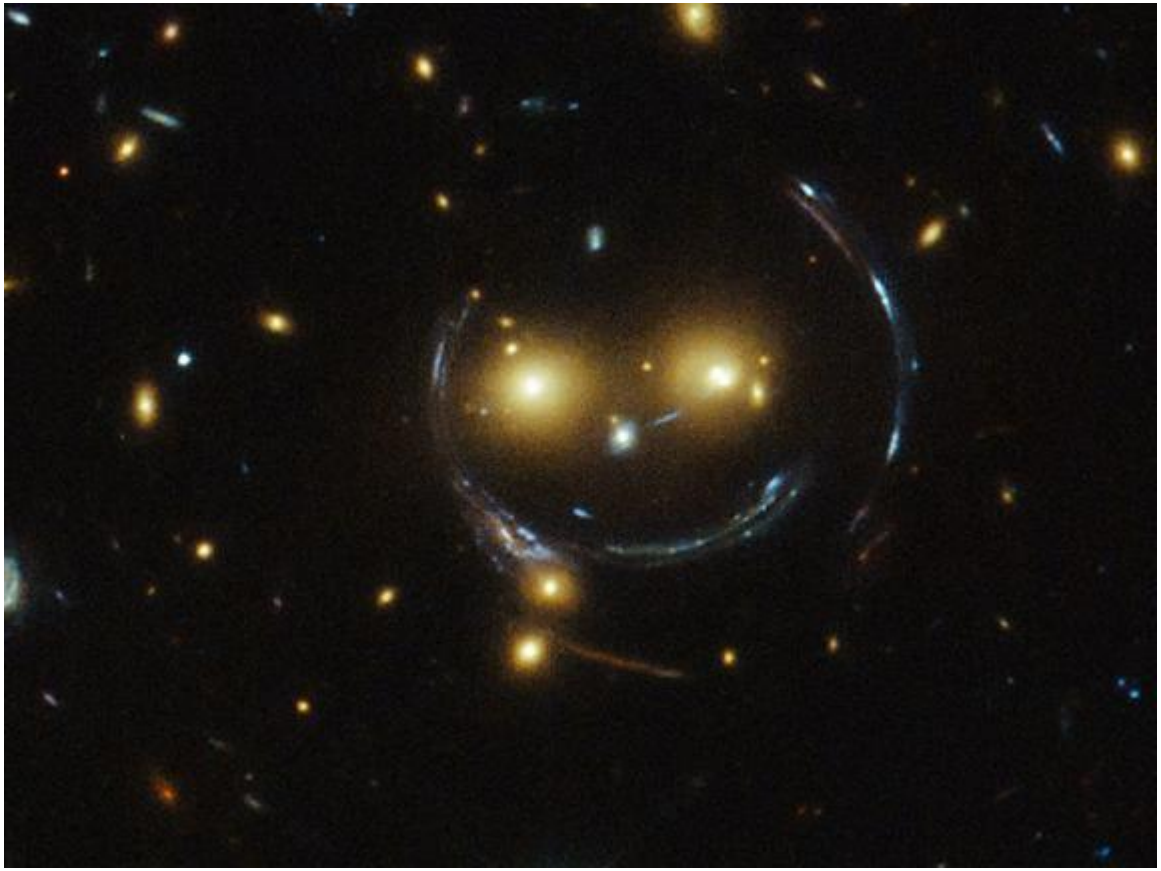
$$n^2 \cdot ck = 100^2 \cdot 10^{-4}s = 1s$$

$$2^n = 2^{100} = (2^{10})^{10} \approx (10^3)^{10} = 10^{30}$$

$$10^{30} \cdot ck = 10^{30} \cdot 10^{-4}s = 10^{26}s \approx 3 \cdot 10^{18} lat$$

$$1rok = 3600 \cdot 24 \cdot 365s \approx 3 \cdot 10^7 s$$

$13{,}82 \cdot 10^9 lat$ - czas życia wszechświata



Dwie galaktyki z efektami soczewkowania grawitacyjnego

[Zdjęcie wykonane teleskopem Hubble’a]

Złożoność obliczeniowa czasowa

Problem dodawania macierzy kwadratowych

Dane:

Macierze $A_{n\times n}$, $B_{n\times n}$

Zadanie:

Wyznaczyć macierz

$$C_{n\times n} = A_{n\times n} + B_{n\times n}$$

Złożoność obliczeniowa czasowa

Algorytm dodawania macierzy kwadratowych

Algorytm wyrażony w języku zawierającym elementy języka naturalnego i formalnego

Dla każdej pary $< i, j > \in \{\overline{1,n}\} \times \{\overline{1,n}\}$ wykonaj

$$c_{ij} = a_{ij} + b_{ij}$$

Oznaczenia:

$\{\overline{1,n}\} = \{1,2,\dots,n\}$,

$X \times Y$ - iloczyn kartezjański zbiorów X, Y

Złożoność obliczeniowa algorytmu

$$\approx k \cdot n^2$$

n jest rozmiarem problemu.

Złożoność obliczeniowa czasowa

Sumowanie macierzy w pseudokodzie

$SUM(A, B, n, C)$

for $i \leftarrow 1$ **to** n

for $j \leftarrow 1$ **to** n

do $C[i,j] \leftarrow A[i,j] + B[i,j]$

τ_d - czas dostępu do pamięci,

τ_+ - czas dodawania dwu liczb,

τ_w - czas wyznaczania wartości zmiennej pętli plus czas weryfikacji względem wartości granicznej,

$$\tau \approx n^2 \cdot (3\tau_d + \tau_+) + (n^2 + n) \cdot \tau_w \approx k \cdot n^2$$

Złożoność obliczeniowa czasowa

Problem mnożenia macierzy kwadratowych

Dane:

Macierze $A_{n\times n}$, $B_{n\times n}$

Zadanie:

Wyznaczyć macierz

$$C_{n\times n} = A_{n\times n} \cdot B_{n\times n}$$

Złożoność obliczeniowa czasowa

Algorytm mnożenia macierzy kwadratowych

Algorytm wyrażony w języku zawierającym elementy języka naturalnego i formalnego

Dla każdej pary $< i, j > \in \{\overline{1,n}\} \times \{\overline{1,n}\}$ wykonaj

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Złożoność obliczeniowa algorytmu

$$\approx k \cdot n^3$$

Złożoność obliczeniowa czasowa

Złożoność algorytmów

Algorytmu dodawania macierzy kwadratowych

$$\approx k \cdot n^2$$

Algorytmu mnożenia macierzy kwadratowych

$$\approx k \cdot n^3$$

Powyższe funkcje są wielomianami, a zatem te algorytmy są uważane za efektywne.

Złożoność obliczeniowa czasowa

Problem wyznaczania elementu maksymalnego w zbiorze

$$max = max\{A1, A2, \dots, An\}$$

Pseudokod algorytmu

```
max(A1, A2,…, to An integers);
max := A1;
for i:= 2 to n do
        if max < Ai then max := Ai;
return max {max is the largest element}
```

Złożoność obliczeniowa algorytmu

$$\approx k \cdot n$$

Złożoność obliczeniowa czasowa

Problem sortowania zbioru

$$\{x_1, x_2, \ldots, x_n\}$$

niemalejąco.

Idea algorytmu sortowania bąbelkowego

Algorytm opiera się na zasadzie: każda liczba jest mniejsza lub równa od liczby maksymalnej. Porównując kolejno liczby można wyznaczyć największą z nich. Następnie ciąg częściowo posortowany (mający na końcu posortowane liczby maksymalne), można skrócić o te liczby i ponowić szukanie maksimum, już bez elementów posortowanych i tak długo, aż zostanie nam jeden element. Otrzymane kolejne maksima są coraz mniejsze przez co ciąg jest uporządkowany.

Złożoność obliczeniowa czasowa

Algorytm sortowania bąbelkowego w kodzie C

```
void bubblesort(int table[], int size)
{
        int i, j, temp;
        for (i = 0; i<size; i++)
    {
                for (j=0; j<size-1-i; j++)
                {
                        if (table[j] > table[j+1])
                        {
                                temp = table[j+1];
                                table[j+1] = table[j];
                                table[j] = temp;
                        }
                }
    }
}
```

Złożoność obliczeniowa czasowa

Złożoność algorytmu sortowania bąbelkowego

$$\approx k \cdot n^2$$

Złożoność obliczeniowa czasowa

Problem podziału zbioru

Dane:

- $X = \{x_1, \dots, x_i, \dots, x_k\}$ - zbiór k elementów $x_i \in N_+$, gdzie $N_+ = \{1,2, \dots\}$,
- $B \in N_+$,
- $\sum_{i=1}^{k} x_i = 2B$.

Pytanie:

Czy istnieje podzbiór $X_1 \subset X$ taki, że

$$\sum_{x_i \in X_1} x_i = B \ ?$$

Złożoność obliczeniowa czasowa

Problem:

Czy istnieje algorytm wielomianowy dla problemu podziału zbioru?

Złożoność obliczeniowa czasowa i pamięciowa

Intuicyjne pojmowanie klasy NP

Klasa NP zawiera te wszystkie problemy decyzyjne, dla których znaleziono ponadwielomianowe algorytmy ich rozwiązania (w wielomianowym czasie można odgadnąć rozwiązanie i sprawdzić czy to rozwiązanie daje odpowiedź "tak").

Złożoność obliczeniowa czasowa

Problem:

Czy istnieje algorytm wielomianowy dla problemu podziału zbioru?

Udowodniono:

Problem podziału zbioru jest tzw. Problemem NP-zupełnym, tzn. z jego rozwiązania można wyznaczyć rozwiązanie każdego problemu klasy NP.

Dla problemów NP-zupełnych wydaje się prawie pewne, że algorytmu wielomianowego nie uda się zbudować.

Złożoność obliczeniowa czasowa

Decyzyjny problem szeregowania zadań niezależnych na dwu maszynach (procesorach)

Dane:

- $T = \{t_1, \ldots, t_i, \ldots, t_k\}$ - zbiór czasów wykonania k zadań niezależnych (bez ograniczeń kolejności wykonywania) gdzie $t_i \in N_+$,
- $B \in N_+$,
- $\sum_{i=1}^{k} t_i = 2B$.

Pytanie:

Czy można te zadania tak rozłożyć na dwie maszyny, że sumaryczny czas wykonania będzie równy B ?

Złożoność obliczeniowa czasowa

Czy rozwiązanie jednego z dwu problemów:

Decyzyjnego problemu szeregowania zadań niezależnych na dwu maszynach

i

Problemu podziału zbioru

można wyznaczyć na postawie rozwiązania drugiego?

Powyższe pytanie dotyczy redukcji (transformacji) jednego problemu do drugiego.

Złożoność obliczeniowa czasowa

Problem podziału zbioru

Dane:

- $X = \{x_1, \dots, x_i, \dots, x_k\}$ - zbiór k elementów $x_i \in N_+$, gdzie $N_+ = \{1,2, \dots\}$,
- $B \in N_+, \quad \sum_{i=1}^{k} x_i = 2B.$

Pytanie:

Czy istnieje podzbiór $X_1 \subset X$ taki, że $\sum_{x_i \in X_1} x_i = B$?

Decyzyjny problem szeregowania zadań niezależnych na dwu maszynach (procesorach)

Dane:

- $T = \{t_1, \dots, t_i, \dots, t_k\}$ - zbiór czasów wykonania k zadań niezależnych (bez ograniczeń kolejności wykonywania) gdzie $t_i \in N_+$,
- $B \in N_+, \ \sum_{i=1}^{k} t_i = 2B.$

Pytanie:

Czy można te zadania tak rozłożyć na dwie maszyny, że sumaryczny czas wykonania będzie równy B ?

Złożoność obliczeniowa czasowa

Czy rozwiązanie jednego z dwu problemów:

Decyzyjnego problemu szeregowania zadań niezależnych na dwu maszynach

i

Problemu podziału zbioru

można wyznaczyć na postawie rozwiązania drugiego?

Powyższe pytanie dotyczy redukcji (transformacji) jednego problemu do drugiego.

Jakie ograniczenie narzucilibyśmy na złożoność obliczeniową transformacji?

Złożoność obliczeniowa czasowa

Problem charakteryzowany jest poprzez

Dane (dane wejściowe) i *Zadanie* do wykonania (np. Prob. opt.)

lub

Dane (dane wejściowe) i *Pytanie* (Prob. dec.).

Złożoność obliczeniowa czasowa

Problem dodawania macierzy kwadratowych

Dane:

Macierze $A_{n\times n}$, $B_{n\times n}$

Zadanie:

Wyznaczyć macierz

$$C_{n\times n} = A_{n\times n} + B_{n\times n}$$

Problem podziału zbioru

Dane:

- $X = \{x_1, \dots, x_i, \dots, x_k\}$ - zbiór k elementów $x_i \in N_+$, gdzie $N_+ = \{1,2, \dots\}$,
- $B \in N_+$, $\sum_{i=1}^{k} x_i = 2B$.

Pytanie:

Czy istnieje podzbiór $X_1 \subset X$ taki, że $\sum_{x_i \in X_1} x_i = B$?

Złożoność obliczeniowa czasowa i pamięciowa

Złożoność obliczeniowa algorytmu
a złożoność problemu

Algorytm rozwiązujący problem może mieć czasową lub pamięciową złożoność obliczeniową logarytmiczną, liniową, wielomianową, wykładniczą lub wiele innych.

Złożoność obliczeniowa problemu określa wymagania czasowe lub pamięciowe na algorytm rozwiązujący ten problem.

Jeśli problem jest NP-zupełny, to prawie na pewno nie zostanie znaleziony wielomianowy algorytm rozwiązujący.

Lepiej nie mówić: Dla problemu NP-zupełnego najprawdopodobniej nie zostanie znaleziony wielomianowy algorytm rozwiązujący.

Złożoność obliczeniowa czasowa i pamięciowa

P – klasa problemów, dla których rozwiązania wymagany jest czas ograniczony od góry przez wielomian od rozmiarów problemu,

PSPACE – klasa problemów, dla których rozwiązania wymagana jest pamięć ograniczona przez wielomian od rozmiarów problemu,

EXPTIME – klasa problemów, dla których rozwiązania wymagany jest czas ograniczony przez funkcję wykładniczą od rozmiarów problemu,

EXPSPACE – klasa problemów, dla których rozwiązania wymagana jest pamięć ograniczona przez funkcję wykładniczą od rozmiarów problemu,

Złożoność obliczeniowa czasowa i pamięciowa

Intuicyjne pojmowanie klasy NP

Klasa NP zawiera te wszystkie problemy decyzyjne, dla których znaleziono ponadwielomianowe algorytmy ich rozwiązania (w wielomianowym czasie można odgadnąć rozwiązanie i sprawdzić czy to rozwiązanie daje odpowiedź "tak").

Złożoność obliczeniowa czasowa i pamięciowa

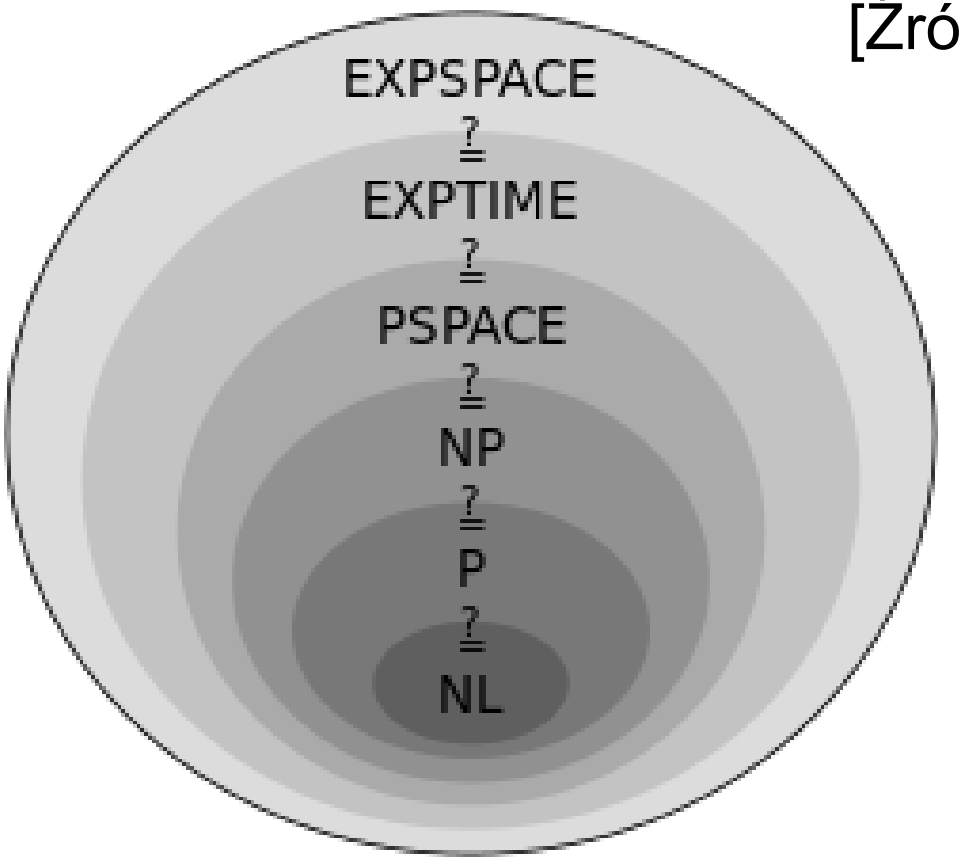
Klasa NL zawiera te wszystkie problemy decyzyjne, dla których znaleziono niedeterministyczne i wymagające logarytmicznej przestrzeni algorytmy ich rozwiązania (w logarytmicznej przestrzeni można odgadnąć rozwiązanie (niedeterministycznie) i sprawdzić czy to rozwiązanie daje odpowiedź "tak").

Uwaga:

W przypadku klasy o złożoności pamięciowej mniejszej niż liniowa nie jest uwzględniana przestrzeń potrzebna do reprezentacji problemu.

Złożoność obliczeniowa czasowa i pamięciowa

[Źródło Wikipedia]



$P \subsetneq EXPTIME$, $PSPACE \subsetneq EXPSPACE$

Czasowa złożoność obliczeniowa

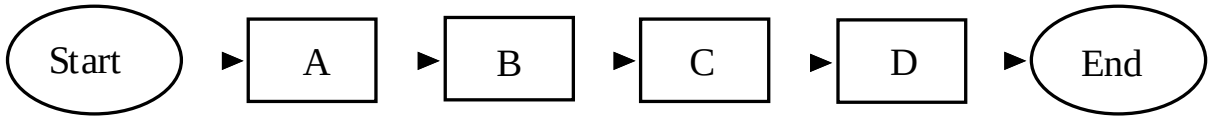
Rodzaje czasowej złożoności obliczeniowej:

- Najlepszego przypadku,
  - Złożoność rozwiązania problemu dla najlepszych danych wejściowych rozmiaru n,
- Najgorszego przypadku,
  - Złożoność rozwiązania problemu dla najgorszych danych wejściowych rozmiaru n,
- Średniego przypadku,
  - Średnia złożoność rozwiązania problemu o rozmiarze n.

Czasowa złożoność obliczeniowa

- Assumption

  Execution times of operations, actions or statements are given by a real number.



Linear program Pr where *A*, *B*, *C*, *D* are operations, actions or statements

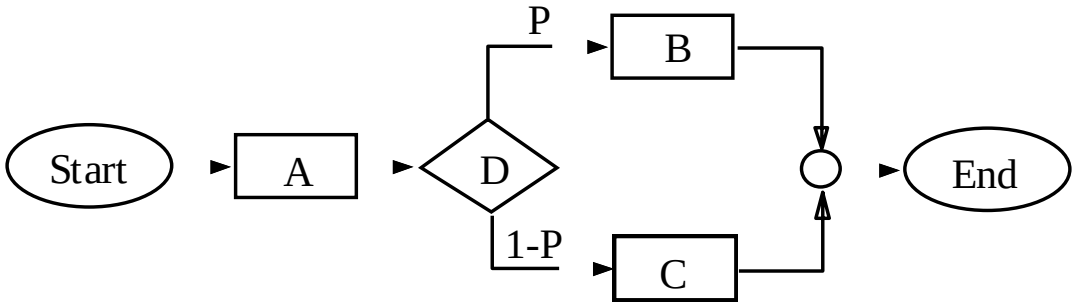
$\tau(A) \in R_+$ - execution time of *A*

- Execution time of the program Pr

$$\tau(\mathrm{Pr}) = \tau(A) + \tau(B) + \tau(C) + \tau(D)$$

[J. Magott, Information Systems Analysis]

Czasowa złożoność obliczeniowa



A program with decision

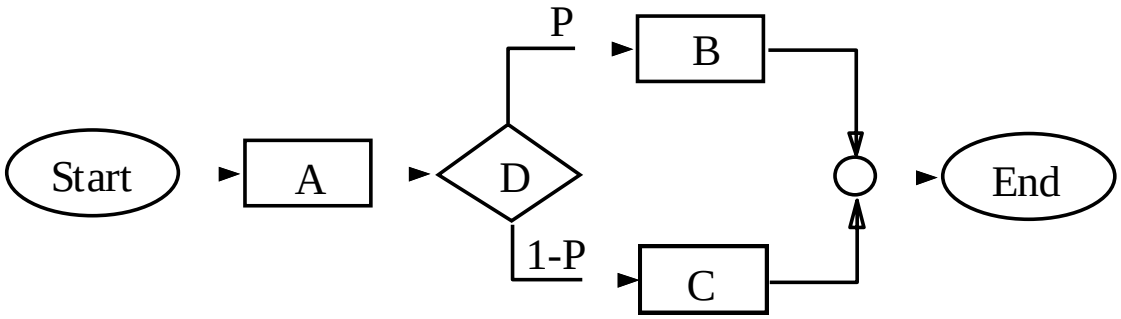
$\tau(D)$ – execution time of decision D

P – probability of an event that after the decision D, the B action is executed

If $\tau(B) \neq \tau(C)$ then execution time of the program depends on the decision.

Czasowa złożoność obliczeniowa

Execution time estimation methods



1. *The worst case method*

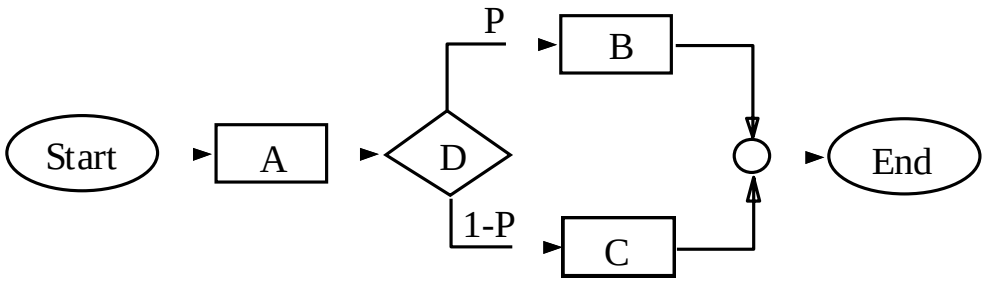
$$\tau^{w}(\mathrm{Pr}) = \tau(A) + \tau(D) + \max\{\tau(B), \tau(C)\}$$

$\tau^{w}(\mathrm{Pr})$ - execution time estimation of the worst case

Application: real-time systems.

Czasowa złożoność obliczeniowa

2. *The most probable path method*

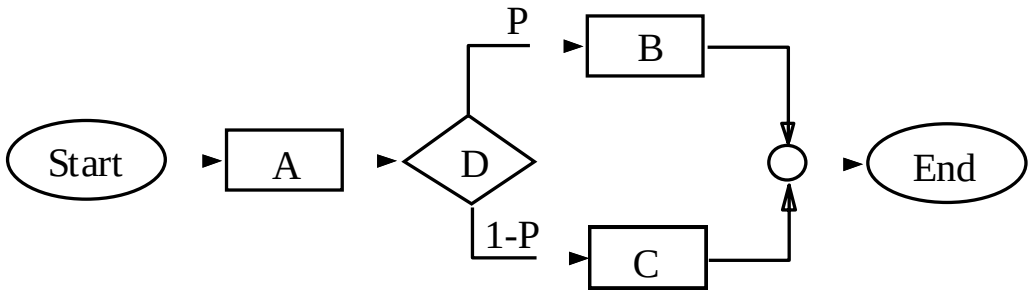


If P>1-P then $\tau^g(\mathrm{Pr}) = \tau(A) + \tau(D) + \tau(B)$

If P<1-P then $\tau^g(\mathrm{Pr}) = \tau(A) + \tau(D) + \tau(C)$

Czasowa złożoność obliczeniowa

3. *The arithmetic mean method*



$$\tau^{a}(\mathrm{Pr}) = \tau(A) + \tau(D) + \frac{\tau(B) + \tau(C)}{2}$$

4. *The random variable mean method*

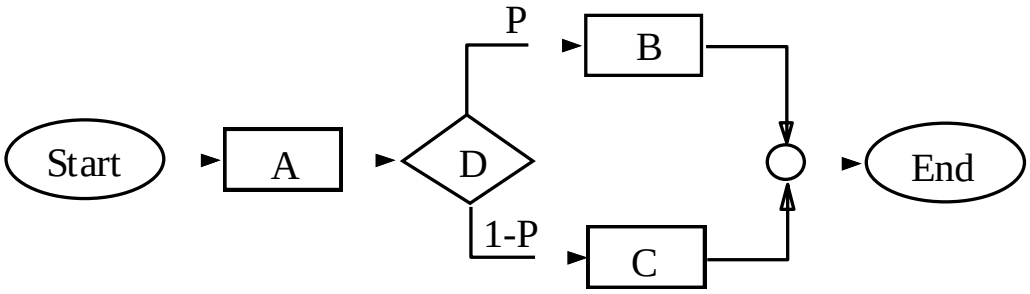
$$\tau^{m}(\mathrm{Pr}) = \tau(A) + \tau(D) + P \cdot \tau(B) + (1 - P) \cdot \tau(C)$$

$\tau(A)$ – random variable of execution time of A

Application: general purpose systems.

Czasowa złożoność obliczeniowa

Czasowa złożoność obliczeniowa średniego przypadku



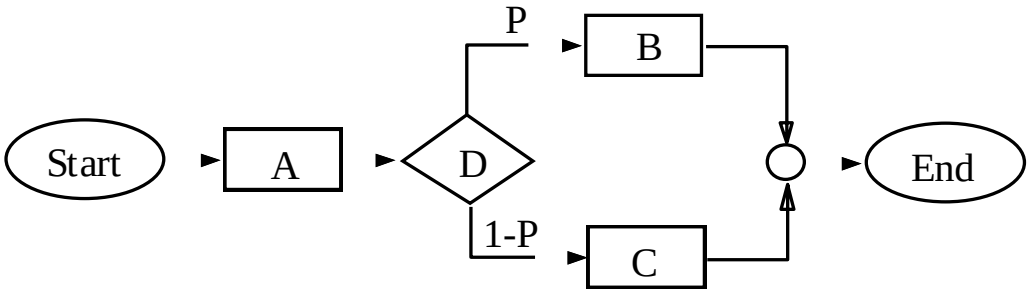
$$\tau^m(\mathrm{Pr}) = \tau(A) + \tau(D) + P \cdot \tau(B) + (1-P) \cdot \tau(C)$$

Struktury sterowania:

- Kompozycja sekwencyjna,
- Alternatywa,
- Instrukcje pętli: for, while, do while,
- Instrukcja przełączania switch.

Czasowa złożoność obliczeniowa

Czasowa złożoność obliczeniowa średniego przypadku



$$\tau^m(\mathrm{Pr}) = \tau(A) + \tau(D) + P \cdot \tau(B) + (1 - P) \cdot \tau(C)$$

Metody wyznaczania:

- Analityczna – na podstawie prawdopodobieństw rozgałęzień i rozkładów czasów wykonania instrukcji,
- Symulacyjna – metoda Monte Carlo na podstawie analogicznych danych.

Notacja asymptotyczna

Notacja O

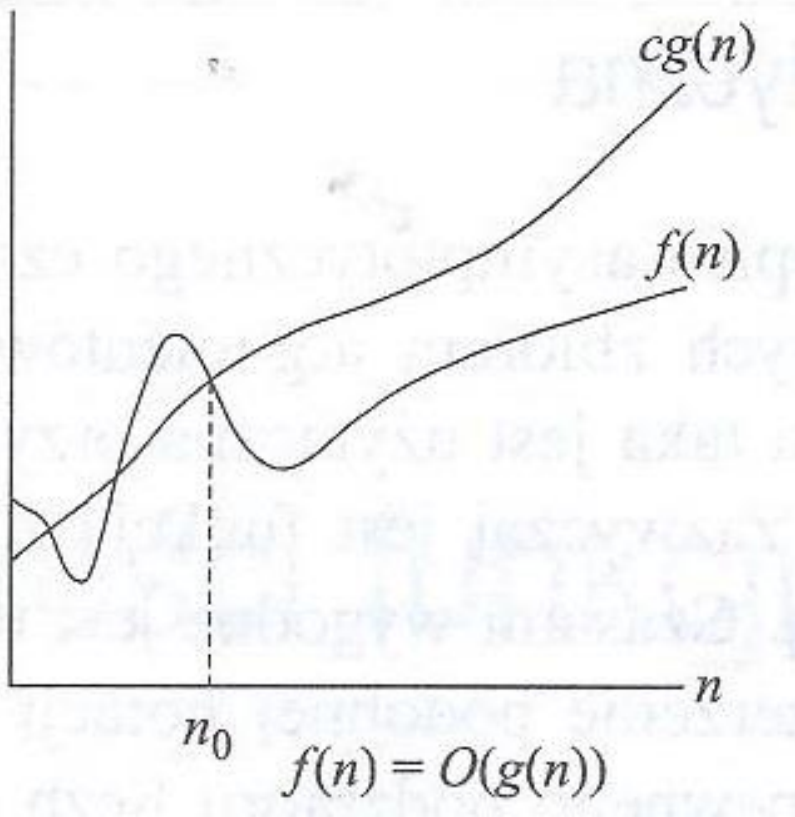
Funkcja $f(n)$ jest rzędu funkcji (jest asymptotycznie ograniczona od góry przez) $g(n)$, co zapisujemy $f(n) = O(g(n))$, jeśli istnieje stała $0 < c$ taka, że istnieje $0 < n_0$ takie, że $n_0 < n$ (czyli prawie dla wszystkich wartości n)

$$0 \leq f(n) \leq c \cdot g(n).$$

Lepsze a gorsze ograniczenie górne.

Ponadto istnieją c' i $g(n)$ takie, że

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c' .$$



Źródło:

[CLRS, Wprowadzenie do algorytmów]

Notacja asymptotyczna

Przykład

$$2n^3 - 5n^2 + 7n - 11 = O(n^3)$$

ponieważ

$$\lim_{n\to\infty} \frac{2n^3 - 5n^2 + 7n - 11}{n^3} = 2$$

Notacja asymptotyczna

Własności notacji *O*

Jeżeli $f(n) = O(a(n))$ i $g(n) = O(b(n))$,
to $f(n) + g(n) = O\big(a(n) + b(n)\big) = O(\max\{a(n), b(n)\})$.

Jeżeli $f(n) = O(a(n))$ i $g(n) = O(b(n))$,
to $f(n) \cdot g(n) = O\big(a(n)b(n)\big)$.

Jeżeli $p(n)$ jest wielomianem stopnia k, $p(n) = O(n^k)$.

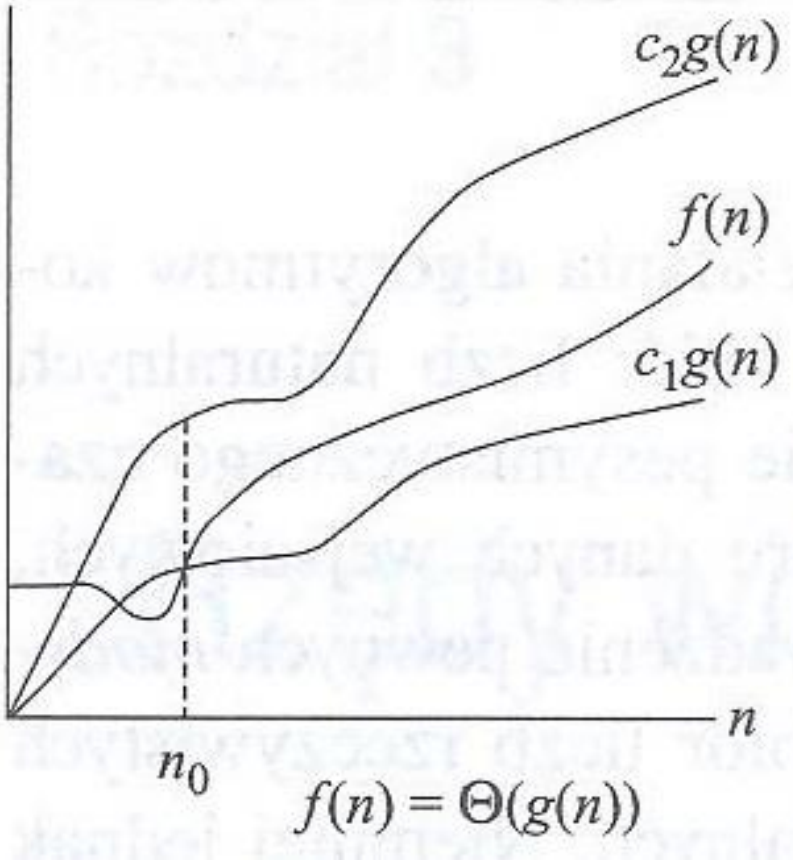
Notacja asymptotyczna

Notacja Θ

Asymptotycznie dokładnym oszacowaniem funkcji $f(n)$ jest funkcja $g(n)$, co zapisujemy $f(n) = \Theta(g(n))$, jeśli istnieją stałe $0 < c_1, c_2$ takie, że istnieje $0 < n_0$ takie, że dla $n_0 < n$

(czyli prawie dla wszystkich wartości n)

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n).$$



Źródło:

[CLRS, Wprowadzenie do algorytmów]

Notacja asymptotyczna

Notacja Ω

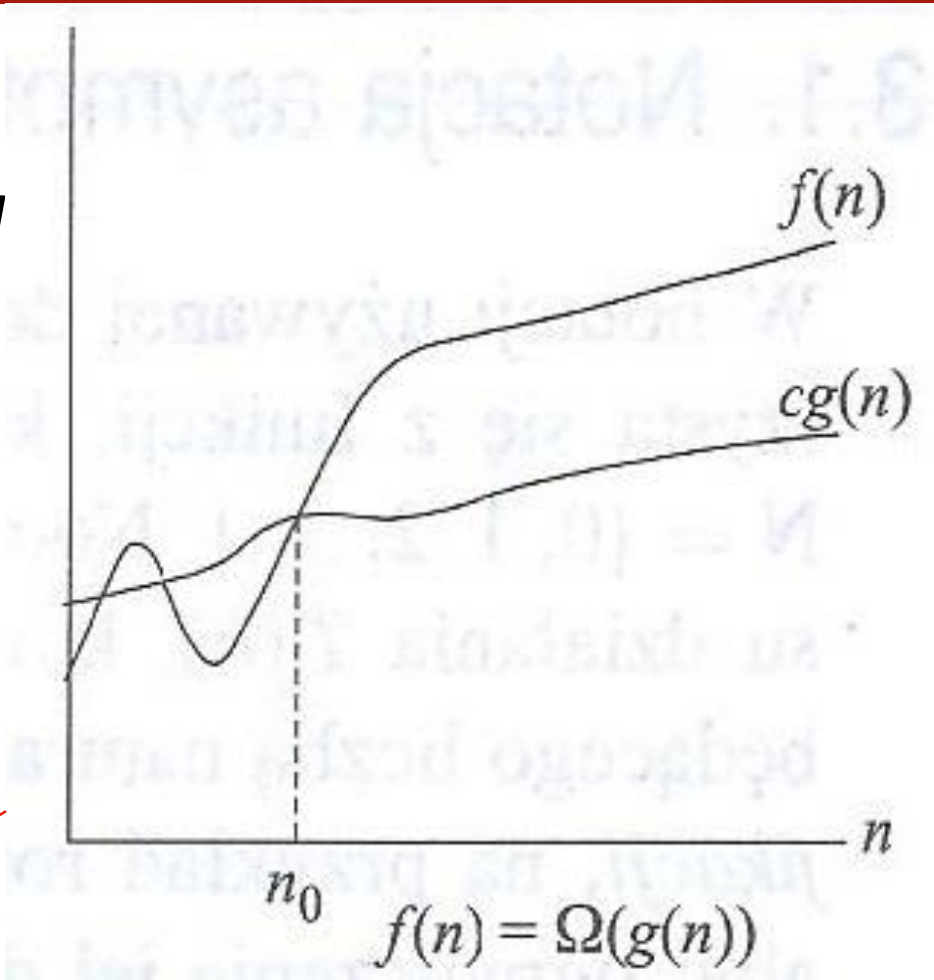
Funkcja $f(n)$ jest asymptotycznie ograniczona od dołu przez $g(n)$, co zapisujemy $f(n) = \Omega\big(g(n)\big)$, jeśli istnieje stała $0 < c$ taka, że istnieje $0 < n_0$ takie, że $n_0 < n$

(czyli prawie dla wszystkich wartości n)

$$0 \leq c \cdot g(n) \leq f(n)$$

Źródło:

[CLRS, Wprowadzenie do alg.]



Koszt zamortyzowany

Operacja 1

Operacja 2

…

…

…

Operacja n

$C_{Operacja\ i}$ - koszt operacji i-tej,

$C = \max_{i \in \{\overline{1,n}\}} \{C_{Operacja\ 1}, \ldots, C_{Operacja\ i}, \ldots, C_{Operacja\ n}\}$

$n \cdot C$ może być zbyt pesymistycznym oszacowaniem kosztu wykonania n operacji

Koszt zamortyzowany

Cel:

Znalezienie **jak najmniejszego pesymistycznego kosztu** wykonania algorytmu.

Sposób osiągnięcia:

Wyznaczenie kosztu zamortyzowanego algorytmu –

co nie wymaga wyznaczania wielkości probabilistycznych.

Koszt zamortyzowany

Metody analizy kosztu zamortyzowanego:

- **Kosztu sumarycznego**,
- **Księgowania**,
- Potencjału.

Koszt zamortyzowany
(Metoda kosztu sumarycznego)

$T(n)$ - pesymistyczny czas wykonania ciągu n operacji,

$\frac{T(n)}{n}$ - koszt zamortyzowany (średni) jednej operacji, nawet jeśli operacje są różne,

Cel:

Wyznaczenie jak najmniejszego kosztu zamortyzowanego.

Koszt zamortyzowany
(Metoda kosztu sumarycznego)

Przykład - operacje na stosie

PUSH(*S*,*x*) – wkłada element *x* na stos *S*,

POP(*S*) – zdejmuje element ze stosu *S*.

Czas wykonania operacji PUSH, POP jest $O(1)$;
przyjmujemy, że ich koszt jest równy 1.

MULTIPOP(*S*,*k*) – usuwa kolejnych k elementów ze szczytu stosu lub opróżnia go, jeśli elementów było mniej niż k.

Koszt zamortyzowany
(Metoda kosztu sumarycznego)

MULTIPOP(*S*,*k*) – usuwa kolejnych k elementów ze szczytu stosu *S* lub opróżnia go, jeśli elementów było mniej niż k.

Kod MULTIPOP(*S*,*k*)

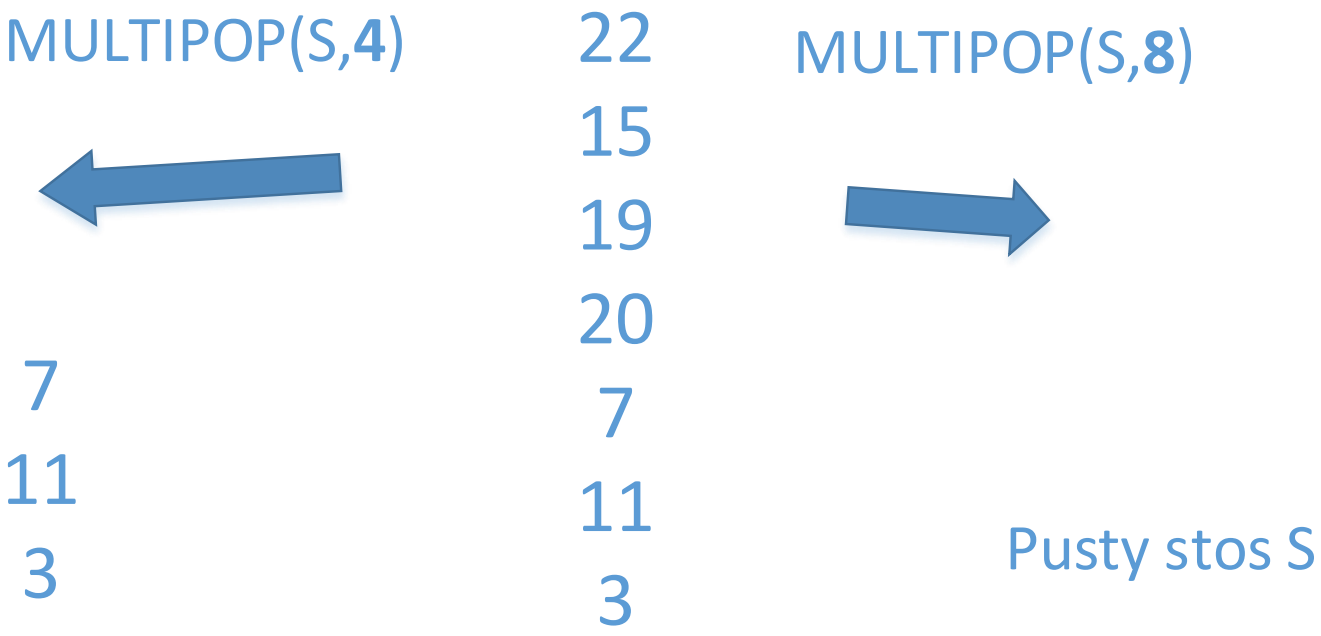
while not STACK-EMPTY(S) **and** $k \neq 0$

do POP(*S*)

$k \leftarrow k - 1$

W ramach operacji MULTIPOP(*S*,*k*) operacja POP jest wykonywana $\min(k, s)$, gdzie s – liczba elementów na stosie *S*.

Koszt zamortyzowany
(Metoda kosztu sumarycznego)



W ramach operacji MULTIPOP(*S*,*k)* operacja POP jest wykonywana $\min(k, s)$, gdzie s - liczba elementów na stosie *S*.

Koszt zamortyzowany (Metoda kosztu sumarycznego)

Dowolna sekwencja n operacji spośród PUSH, POP, MULTIPOP.

Założenie: Pusty stos przed wykonaniem n operacji.

Cel:

Wyznaczenie jak najmniejszego kosztu zamortyzowanego $\frac{T(n)}{n}$.

1. n operacji MULTIPOP; każda o czasie wykonania $O(n)$

$$\frac{T(n)}{n}=\frac{n\cdot O(n)}{n}=\frac{O(n^2)}{n}=O(n)$$

2. Uwzględniając operacje POP w ramach MULTIPOP, n operacji PUSH, POP, MULTIPOP stanowi n operacji PUSH, POP

$$\frac{T(n)}{n}=\frac{n\cdot O(1)}{n}=\frac{O(n)}{n}=O(1)$$

Koszt zamortyzowany (Metoda księgowania)

W metodzie kosztu sumarycznego koszt zamortyzowany jest średnim kosztem ciągu n operacji. **W metodzie księgowania, operacjom mogą być przypisane różne koszty zamortyzowane.**

c_i - faktyczny koszt i —tej operacji,

$\hat{c}_i$ - zamortyzowany koszt i —tej operacji,

Jeśli chcemy minimalizować koszt ciągu n operacji, to wymagamy:

$$\sum_{i=1}^{1=n} \hat{c}_i \geq \sum_{i=1}^{i=n} c_i$$

Koszt zamortyzowany (Metoda księgowania)

W metodzie księgowania zawyżamy koszt wcześniej wykonywanej operacji na elemencie struktury danych, aby wystarczył na pokrycie kosztu również późniejszej operacji na tym elemencie.

Przykład przypisania kosztów zamortyzowanych

Założenie: Pusty stos przed wykonaniem n operacji.

PUSH $1 + \alpha$ gdzie $1 \leq \alpha$

α – koszt poniesiony przy wykonywaniu operacji POP lub jednego kroku operacji MULTIPOP,

POP 0

MULTIPOP 0

Koszt zamortyzowany (Metoda księgowania)

MULTIPOP(*S*,*k*) – usuwa kolejnych k elementów ze szczytu stosu *S* lub opróżnia go, jeśli elementów było mniej niż k.

Kod MULTIPOP(*S*,*k*)

while not STACK-EMPTY(S) **and** $k \neq 0$

do POP(*S*)

$k \leftarrow k - 1$

W ramach operacji MULTIPOP(*S*,*k*) operacja POP jest wykonywana $\min(k, s)$, gdzie s – liczba elementów na stosie *S*.

Koszt zamortyzowany (Metoda księgowania)

Przykład przypisania kosztów zamortyzowanych

Założenie: Pusty stos przed wykonaniem n operacji.

PUSH $1 + \alpha$ gdzie $1 \leq \alpha$

α – koszt poniesiony przy wykonywaniu operacji POP lub jednego kroku operacji MULTIPOP,

POP 0

MULTIPOP 0

Nie można wykonać więcej operacji POP w ramach operacji POP i MULTIPOP niż liczba wykonań operacji PUSH.

$T(n) = n \cdot (1 + \alpha) = O(n)$ - ograniczenie górne całkowitego kosztu zamortyzowanego n operacji POP, PUSH, MULTIPOP.