10 Analiza asymptotyczna

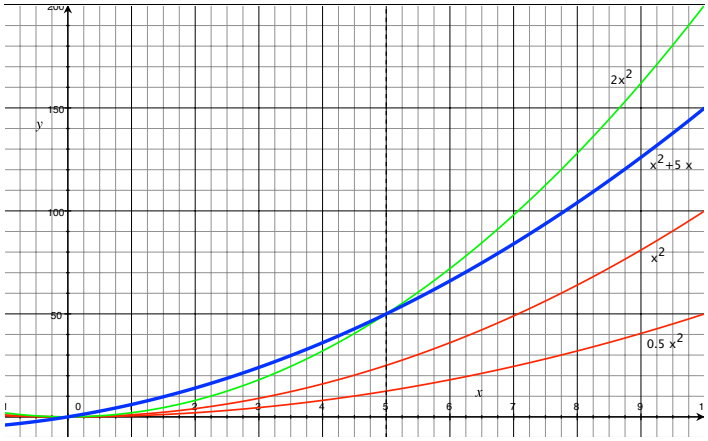
10.1 Stosowane notacje

10.1.1 Notacja $O(\cdot)$

Definicja 1 (Notacja O duże) *Niech $f, g : \mathcal{N} \mapsto \mathcal{R}$. Wtedy*

$$f = O(g) \equiv (\exists_{c>0})(\exists_{k \in \mathcal{N}})(\forall_{n>k}) f(n) < c \cdot g(n).$$

Przykład 1 *Rozpatrzmy funkcję $f(n) = n^2 + 5 \cdot n$ oraz funkcje $g_1(n) = \frac{1}{2} \cdot n^2$, $g_2(n) = n^2$, $g_3(n) = 2 \cdot n^2$. Poniżej przedstawiono wykresy funkcji f, g_1, g_2, g_3 rozszerzając ich dziedziny na liczby rzeczywiste.*



Dla $n > 5$, $f(n) < 2 \cdot n^2$. Zatem $f(n) = O(n^2)$.

10.1.2 Notacja $\Theta(\cdot)$

Definicja 2 (Notacja Θ duże) *Niech $f, g : \mathcal{N} \mapsto \mathcal{R}$. Wtedy*

$$f = \Theta(g) \equiv (f = O(g) \wedge g = O(f)).$$

Twierdzenie 1 *Niech $f, g : \mathcal{N} \mapsto \mathcal{R}$. Załóżmy, że istnieje granica $\lim_{n\to\infty} \frac{f(n)}{g(n)}$. Wówczas*

1. *Jeśli $0 \leq \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$, to $f = O(g)$.*
2. *Jeśli $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$, to $f = \Theta(g)$.*

Dowód 1 *Niech $c = \lim n \to \infty \frac{f(n)}{g(n)}$. Załóżmy, że $0 \leq c < \infty$. Wówczas dla dostatecznie dużych n, $0 \leq \frac{f(n)}{g(n)} < c + 1$. Zatem dla dostatecznie dużych n, $f(n) < (c+1) \cdot g(n)$, więc $f = O(g)$.*

Jeśli ponadto $c > 0$, to $0 \leq \lim_{n\to\infty} \frac{g(n)}{f(n)} = \frac{1}{c} < \infty$, więc z punktu 1. wynika, że $g = O(f)$, czyli $f = \Theta(g)$. □

10.1.3 Notacja $o(\cdot)$

Definicja 3 (Notacja o małe) *Niech $f, g : \mathcal{N} \mapsto \mathcal{R}^+$. Wtedy*

$$f = o(g) \equiv \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

Czasami zamiast pisać $f = o(g)$ będziemy stosować zapis $f \ll g$.
Łatwo sprawdzić następujące zależności:

$$\cdots \ll \sqrt[4]{n} \ll \sqrt[3]{n} \ll \sqrt{n} \ll n \ll n^2 \ll n^3 \ll n^4 \ll \cdots$$

Skorzystamy z poniższego twierdzenia:

Twierdzenie 2 (Reguła de l'Hospitala) *Załóżmy, że f i g są funkcjami różniczkowalnymi tż. $\lim_{x\to\infty} f(x) = \infty$ oraz $\lim_{x\to\infty} g(x) = \infty$. Wówczas*

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

Funkcja $\log_2 n$ rośnie wolniej od pierwiastka dowolnego stopnia:

Twierdzenie 3

$$(\forall_{\alpha>0}) \log_2 n = o(n^\alpha).$$

Dowód 2 *Niech $\alpha > 0$. Funkcje $f(x) = \log_2 x$ oraz $g(x) = x^\alpha$ spełniają założenia twierdzenia 2. Zatem*

$$\lim_{x\to\infty} \frac{\log_2 x}{x^\alpha} = \lim_{x\to\infty} \frac{\frac{1}{x\cdot\ln 2}}{\alpha \cdot x^{\alpha-1}} = \lim_{x\to\infty} \frac{1}{\alpha \cdot \ln 2 \cdot x^\alpha} = 0.$$

□

Możemy zapisać, że

$$\log_2 n \ll \cdots \ll \sqrt[4]{n} \ll \sqrt[3]{n} \ll \sqrt{n} \ll n.$$

10.2 Złożoność obliczeniowa

Ten sam problem można rozwiązać wieloma różnymi algorytmami. Ważne jest aby znać kryteria jakimi można algorytmy te porównywać.

Najważniejsze z nich to:

- Jak szybko działa algorytm?
- Jak dużo wymaga pamięci?

Są możliwe jeszcze inne kryteria, jak np. jak odległe od najlepszego rozwiązania jest znalezione przez algorytm rozwiązanie (w przypadku problemów optymalizujących), ale skupimy się na tych dwóch.

Analiza złożoności algorytmu bada nie tyle ile dokładnie milisekund działa algorytm albo ile dokładnie bajtów pamięci wymaga ale jak szybko czas i rozmiar pamięci rośnie wraz ze wzrostem rozmiaru problemu.

10.2.1 Rozmiar problemu

Rozmiar problemu mówi nam jak duże są dane na których operuje program. Najczęściej jest to po prostu liczba liczb składających się na dane.

Na przykład, jeśli program ma uporządkować n liczb od najmniejszej do największej, to za rozmiar danych dla problemu sortowania przyjmiemy n.

Zdarza się jednak, że algorytm operuje na stałej liczbie danych.

Na przykład algorytm, który rozkłada liczbę całkowitą na czynniki pierwsze ma tylko jedną liczbę w swoich danych.

W takim przypadku za rozmiar danych przyjmuje się długość binarnej reprezentacji danych. Zatem rozmiarem danych w przypadku algorytmu operującego na jednej liczbie $x \in \mathcal{N}$ jest $n = \lceil \log_2 x \rceil$.

10.2.2 Asymptotyka

Badając funkcję $f : \mathcal{N} \mapsto \mathcal{R}$, będziemy skupiać się nie tyle na jej dokładnym przebiegu ale na tym, jak szybko rośnie ona gdy n wzrasta do nieskończoności.

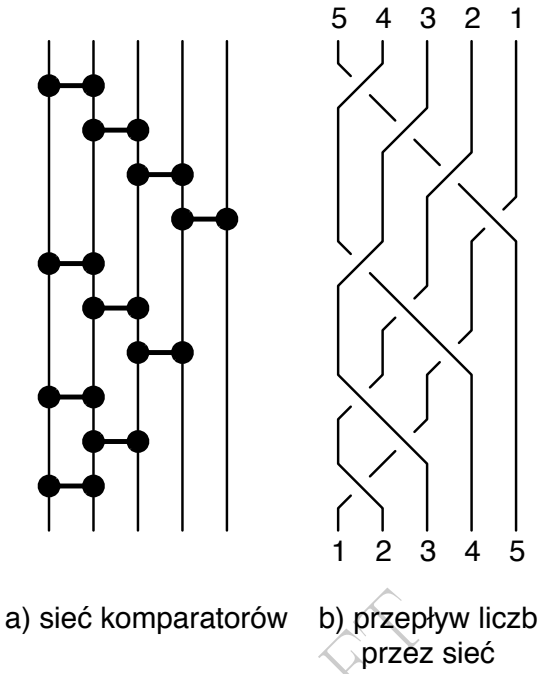
Z punktu widzenia takiej analizy wszystkie trzy funkcje $f_1(n) = n^2 + 10 \cdot n$, $f_2(n) = 1000 \cdot n^2 - 5 \cdot n$ i $f_3(n) = \frac{1}{1\,000\,000} \cdot n^2$ rosną jednakowo szybko bo tak jak wielomian drugiego stopnia:

$$\begin{aligned} \lim_{n\to\infty} \frac{f_1(n)}{n^2} &= 1 \in (0, +\infty) \\ \lim_{n\to\infty} \frac{f_2(n)}{n^2} &= 1000 \in (0, +\infty) \\ \lim_{n\to\infty} \frac{f_3(n)}{n^2} &= \frac{1}{1\,000\,000} \in (0, +\infty) \end{aligned}$$

Mówimy zatem, że wszystkie trzy funkcje są tego samego rzędu $\Theta(n^2)$.

Czwarta funkcja $f_4(n) = 1\,000\,000 \cdot n \cdot \log_2 n$ rośnie wolniej od powyższych trzech gdyż $f_4(n) = o(n^2)$:

$$\lim_{n\to\infty} \frac{f_4(n)}{n^2} = 0.$$



Rysunek 10.1: Idea sortowania bąbelkowego.

10.2.3 Pesymistyczna złożoność czasowa

Analizując złożoność czasową algorytmu na danych rozmiaru n można się zastanawiać na tym średnio jak długo działa (analiza średniego przypadku) i jak najdłużej działa (analiza pesymistyczna).

Analiza średniego przypadku jest bardziej złożona i będzie omawiana na kursie **Algorytmy i struktury danych**. Teraz zajmować się będziemy tylko analizą najgorszego przypadku.

10.2.4 Złożoność pamięciowa

UZUPEŁNIĆ

10.3 Przykłady analizy

10.3.1 Sortowanie bąbelkowe

Na rysunku 10.1a przedstawiono ideę sortowania bąbelkowego. Możemy sobie wyobrazić, że pięć sortowanych liczb porusza się po pięciu drutach z góry na dół. Kiedy dwie sąsiednie liczby dojdą do łączącego ich druty komparatora, zostają porównane i jeśli są w złej kolejności (większa przed mniejszą), to zamienią się miejscami.

Jak widać działanie algorytmu składa się z kolejnych faz. W pierwszej zostają porównane wszystkie sąsiednie liczby. Po takim porównaniu największa z nich znajdzie się na ostatnim prawym drucie.

W następnej fazie należy powtórzyć porównywanie ale już bez ostatniego drutu.

Porównywania takie powtarza się dla coraz mniejszej liczby drutów aż na końcu zostaną porównane tylko dwie skrajnie lewe liczby.

Na rysunku 10.1b przedstawiono przebieg sortowania na przykładzie ciągu liczb $(5, 4, 3, 2, 1)$.

Kod algorytmu sortowania bąbelkowego zapisany w języku C przedstawiono poniżej:

```
for(int i = n-1; i > 0; i--)
  for(int j = 0; j < i; j++)
    if(t[j] > t[j+1])
    {
      int tmp = t[j];
      t[j] = t[j+1];
      t[j+1] = tmp;
    }
```

Oznaczmy przez $C(n)$ liczbę porównań wykonywanych w instrukcji **if** podczas sortowania n liczb na pozycjach $t[0], t[1], \ldots, t[n-1]$.

Twierdzenie 4 *Algorytm sortowania wykonuje kwadratową liczbę porównań:*

$$C(n) = \Theta(n^2).$$

Dowód 3 *Podczas wykonywania wewnętrznej pętli wykonuje się i porównań. Zatem łączna liczba porównań jest równa:*

$$C(n) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{1}{2} \cdot n^2 - \frac{1}{2} \cdot n.$$

Na mocy wcześniejszego twierdzenia 1:

$$\lim_{n\to\infty} \frac{\frac{1}{2} \cdot n^2 - \frac{1}{2} \cdot n}{n^2} = \lim_{n\to\infty} (\frac{1}{2} - \frac{1}{2 \cdot n}) = \frac{1}{2} \in (0, +\infty)$$

wnioskujemy, że $C(n) = \Theta(n^2)$. □

W rozdziale 13 zostanie przestawiony algorytm o mniejszej pesymistycznej złożoności czasowej bo $O(n \log n)$.

10.3.2 Binarne wyszukiwanie

UZUPEŁNIĆ

$$B(n) = O(\log n)$$