W tym rozdziale zajmiemy się notacją umożliwiającą szacowanie czasu działania algorytmów.

I. Notacja O

Większość algorytmów rozwiązujących dany problem nie różni się skutecznością, gdy pracujemy na niewielkiej ilości danych. Kiedy jednak ilość ta rośnie, to, choć wszystkie algorytmy potrzebują więcej czasu, staje się widoczne, że jedne są znacząco szybsze, a inne na tyle wolne, że nie ma sensu ich używać. By jednak móc porównać prędkość działania różnych procedur dla dużych liczb, potrzebny jest jakiś opis ich czasu działania.
Przykład Przeszukiwanie worka ziemniaków: a) by znaleźć zepsutego, b) by znaleźć dwa o identycznej wadze.
Przykład Algorytmy sortowania danych.
Przykład Algorytmy rozwiązywania układów równań liniowych.
Niech $t(n)$ oznacza czas, w jakim program poradzi sobie z listą danych długości n. Pomnożenie t przez jakąś liczbę nie powinno nas zbytnio interesować - można to uzyskać używając szybszego lub wolniejszego komputera. Zatem istotna nie jest absolutna wielkość $t(n)$, ale szybkość, z jaką ta wielkość rośnie dla dużych n: czy rośnie tak szybko jak n, n^2, 2^n, a może $\log n$ lub jakaś inna funkcja (przez $\log n$ na tym przedmiocie rozumiemy logarytm o podstawie 2, chyba, że będzie powiedziane inaczej). Żeby dokładnie określić, co mamy na myśli mówiąc „rośnie tak szybko jak...” musimy wprowadzić nowe oznaczenie:

Definicja 1. *Jeśli dane są dwa ciągi: $s(n)$ i $t(n)$ o wartościach rzeczywistych nieujemnych to mówimy, że $s(n) = O(t(n))$ (czytamy: „$s(n)$ jest O od $t(n)$”) jeśli dla dużych n wartości ciągu s są nie większe niż wartości ciągu t, pomnożonego przez pewną stałą. Bardziej formalnie:*

$$\exists_{C\in\mathbb{N}}\exists_{n_0\in\mathbb{N}}\forall_{n>n_0}|s(n)| \leq C|t(n)|.$$

Przykład $6n^4+20n^2+2000$ jest $O(n^4)$. Jest też $O(n^5)$, ale to mniej przydatna informacja. Nie jest $O(n)$ (kolejność kwantyfikatorów!). Uogólnienie na wielomiany.

Definicja 2. *Jeśli $t_A(n)$ i $t_B(n)$ oznaczają ciągi czasów wykonania odpowiednio algorytmów A i B oraz $t_A(n) = O(t_B(n))$, ale nie jest prawdą, że $t_B(n) = O(t_A(n))$, to mówimy, że B ma większą* czasową złożoność obliczeniową *niż algorytm A.*

Zazwyczaj takie określenie oznacza, że algorytm A jest preferowany w stosunku do B, gdyż dla dużych ilości danych działa szybciej. Oczywiście, w konkretnych zastosowaniach, algorytm o większej czasowej złożoności obliczeniowej nie musi być gorszy (bo może nadrabiać innymi zaletami), a nawet nie musi być wolniejszy. Większa czasowa złożoność obliczeniowa mówi tylko, że dla n większych od pewnej wartości czas wykonania A będzie mniejszy niż czas wykonania B, ale ta „pewna wartość” może być tak duża, że w konkretnej, praktycznej implementacji może to nie mieć znaczenia.

Twierdzenie 1 (Przechodniość czasowej złożoności obliczeniowej)**.** *Jeśli $f(n) = O(g(n))$ i $g(n) = O(h(n))$, to $f(n) = O(h(n))$.*
W szczególności, algorytm B ma większą czasową złożoność obliczeniową niż algorytm A, a algorytm C ma większą czasową złożoność obliczeniową) niż algorytm B, to algorytm C ma większą czasową złożoność obliczeniową niż algorytm A.

Twierdzenie 2. *Oto hierarchia najbardziej znanych ciągów uporządkowanych w ten sposób, że każdy z nich jest O od wszystkich ciągów na prawo od niego:*

$$1, \log n, n^{\alpha_1}, n^{\alpha_2}, n, n\log n, n^{\beta_1}, n^{\beta_2}, a^n, b^n, n!, n^n,$$

dla dowolnych rzeczywistych dodatnich $\alpha_1 < \alpha_2 < 1 < \beta_1 < \beta_2$ i $1 < a < b$. 1 oznacza tu ciąg stale równy 1, który w ogóle nie rośnie.

Definicja 3. *Ciąg $t(n)$* rośnie wielomianowo *gdy $t(n) = O(n^m)$ dla pewnego $m \in \mathbb{N}$.*

Z teoretycznego punktu widzenia, tylko algorytmy, których czas rośnie wielomianowo są uważane za praktyczne, jeśli mamy do czynienia z długimi listami przy ich wykonywaniu. W praktyce, najbardziej pożądany (choć nie zawsze osiągalny) jest czas działania rzędu $O(n)$ lub $O(n \log n)$.

Twierdzenie 3 (Działania na symbolach O)**.** *Zachodzą następujące zależności:*
a) Jeśli $f(n) = O(g(n))$ *i* c *jest stałą, to* $cf(n) = O(g(n))$.
b) Jeśli $f(n) = O(g(n))$ *i* $h(n) = O(g(n))$, *to* $f(n) + h(n) = O(g(n))$.
c) Jeśli $f(n) = O(g(n))$ *i* $h(n) = O(k(n))$, *to* $f(n) + h(n) = O(\max\{g(n), k(n)\})$.
d) Jeśli $f(n) = O(a(n))$ *i* $g(n) = O(b(n))$, *to* $f(n) \cdot g(n) = O(a(n) \cdot b(n))$.

Uwaga Jak już wspomniałem wcześniej, notacja O nie podaje oszacowania z dołu na czas działania algorytmu. W szczególności, jeśli jakiś algorytm ma czas działania typu $O(n^2)$, to nic nie wyklucza faktu, że może też być $O(n)$ (co w wielu wypadkach bardzo by nas cieszyło). Wszystkie programy będące $O(n)$ są też $O(n^2)$. Oszacowania z dołu zapisuje się notacją podobną do O - za pomocą symbolu Ω. Ze względu na brak czasu, nie będziemy się tym na wykładzie zajmować.