Struktury Danych i Złożoność Obliczeniowa
ĆWICZENIA 1

Kryteria analizy algorytmów:
Algorytmy możemy analizować pod względem:

- *Poprawności* - otrzymanie poprawnego wyniku dla każdej konfiguracji danych wejściowych
- *Złożoności* - obliczeniowej, czasowej, pamięciowej
- *Optymalności* - jakości jeżeli chodzi o złożoność obliczeniową
- *Efektywności* - użyteczności w praktycznych sytuacjach

Własność stopu (terminacji):
Własność algorytmu mówiąca, że musi on w pewnym momencie osiągnąć swój logiczny koniec. Zatrzymać się.

Częściowa poprawność:
Własność algorytmu mówiąca, że nigdy nie może on zakończysz swojego działania ze złymi wynikami. Uzyskane w wyniku jego działania wyniki są poprawne, ale niekoniecznie są to wyniki ostateczne (oczekiwane dla wczytanych danych wejściowych). Innymi słowy częściowa poprawność algorytmu mówi, że algorytm poprawnie dokonuje obliczeń pośrednich, lecz niekoniecznie zawsze zwróci nam ostateczną odpowiedź (zwłaszcza dla bardzo złożonych problemów).

Metoda niezmienników:
Metoda pozwalająca na sprawdzenie poprawności algorytmu. W tym celu wykorzystuje się w niej niezmienniki, czyli warunki, które pozostają prawdziwe zarówno przed jak i po wykonaniu danego bloku instrukcji (bez względu na ilość takich wykonań). Minusem metody niezmienników jest to, że znalezienie niezmiennika danego bloku instrukcji zwykle nie jest prostym zadaniem.

Używając niezmiennika pętli udowodnij, że algorytm mnożący jest poprawny.

```
Mnożenie(y,z)
    x ← 0
    while z > 0 do
        if z % 2 = 1 then
            x ← x + y
        y← 2*y
        z← ⌊z/2⌋
return x
```

Proponowany niezmiennik:

$$x + yz = \text{const}$$

$$\updownarrow$$

$$x + yz = x' + y'z'$$

1) Sprawdzamy czy proponowany niezmiennik jest poprawny.

x', y', z' - wartości zmiennych po przebiegu pętli

(1°) Dla parzystego z:

$x' = x$ bo $2 \mid z$

$y' = 2y$

$z' = \lfloor \frac{1}{2} z \rfloor = \frac{1}{2} z$

$x' + y'z' = x + 2y \cdot \frac{1}{2} z =$

$= x + yz$ ✓

(2°) Dla nieparzystego z:

$x' = x + y$ bo $2 \nmid z$

$y' = 2y$

$z' = \lfloor \frac{1}{2} z \rfloor = \frac{1}{2}(z-1)$

$x' + y'z' = x + y + 2y \cdot \frac{1}{2}(z-1) =$

$= x + y + y(z-1) = x + y + yz - y =$

$= x + yz$ ✓

Zatem $(x + yz)$ jest niezmiennikiem tej pętli.

2) Za pomocą niezmiennika wykonujemy dowód:

x_0, y_0, z_0 - wartości wejściowe

$x' \stackrel{?}{=} y_0 z_0$ - do udowodnienia

$x' + y'z' = x_0 + y_0 z_0$ / $x_0 = \text{const} = 0$ w każdym wypadku

$x' + y'z' = y_0 z_0$ / $z' = 0$ bo program wyszedł z pętli aby się zakończyć, a warunek wyjścia to $z = 0$

$x' + y' \cdot 0 = y_0 z_0$

$x' = y_0 z_0$ ∎

Za pomocą niezmiennika udowodniliśmy, że istotnie algorytm zwraca iloczyn $(y \cdot z)$.

Złożoność obliczeniowa:
Ilość zasobów komputerowych koniecznych do wykonania programu realizującego algorytm. Złożoność obliczeniową wyrażamy jako funkcji parametru, określającego rozmiar rozwiązywanego zadania (najczęściej ilości użytych w nim danych). W przypadku szacowania złożoności obliczeniowej możemy mówić o złożoności:

- *czasowej* - funkcja rozmiaru danych $f(n)$. Ilość czasu niezbędnego do realizacji algorytmu wrażana w liczbie cykli procesora lub liczbie wszystkich operacji (unikamy pomiaru w jednostkach SI ze względu na różnice w prędkościach maszyn). Często niemożliwym jest uzyskanie dokładnej wartości złożoności czasowej. W takim wypadku stosuje się szacowanie.

- *pamięciowej* - funkcja rozmiaru danych $f(n)$. Ilość pamięci (w bajtach lub zmiennych typów elementarnych) wykorzystywanej w celu realizacji algorytmu.

W wypadku złożoności obliczeniowej możemy mówić o trzech wariantach:

- *optymistycznym* - określającym zużycie zasobów dla najkorzystniejszego zestawu danych. Przykładowo, gdy szukany w liście element znajduje się na pierwszej pozycji.

- *średnim* - określającym zużycie zasobów dla typowych (losowych) danych.

- *pesymistycznym* - określającym zużycie zasobów dla najbardziej niekorzystnego zestawu danych. Przykładowo, gdy szukany element znajduje się na ostatniej pozycji lity bądź nie istnieje.

W większości wypadków mówiąc o złożoności obliczeniowej odnosimy się do wariantu pesymistycznego. Daje nam to gwarancję, że przewidzimy najgorszy możliwy wariant. Do tego często wariant pesymistyczny jest bardzo zbliżony do średniego oraz w rzeczywistości występuje niezwykle często (np. szukanie w bazie danych informacji, której tam nie ma).

Wyznaczenie złożoności czasowej - metoda dokładna:

Wyznacz złożoność czasową algorytmu sumującego liczby od 1 do n, gdzie wartość n jest zczytywana od użytkownika.

1) Tworzymy tabelę zawierającą kolejne instrukcje algorytmu. Pogrupujemy kolejnymi typowi instrukcji czas wykonywania. i ilość wykonań.

KROK	INSTRUKCJA	CZAS	POWTÓRZEŃ
k_1	wczytaj n	t_1	1
k_2	suma := 0	t_2	1
k_3	i := 1	t_2	1
k_4	if (i > n) then	t_3	n+1
k_5	goto 9	t_4	1
k_6	suma := suma + i	t_2	n
k_7	i := i+1	t_2	n
k_8	goto 4	t_4	n
k_9	wyświetl suma	t_5	1
k_{10}	stop	t_6	1

2) Sumujemy czasy i przekształcamy to w funkcję T(n):

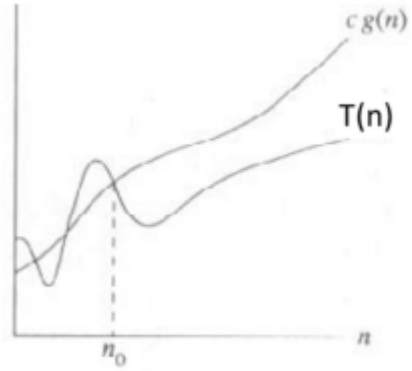
$$T(n) = t_1 + t_2 + t_2 + (n+1)t_3 + t_4 + nt_2 + nt_2 + nt_4 + t_5 + t_6 =$$
$$= (t_1 + 2t_2 + t_5 + t_6 + t_3) + n(t_3 + 2t_2 + t_4) =$$
$$= t_7 n + t_8$$

$$T(n) = t_7 n + t_8, \quad t_8, t_7 = const$$

Złożoność czasowa tego algorytmu jest liniowa.

Notacje asymptotyczne:
Z racji, że metoda dokładna jest bardzo czasochłonna to często szacuje się złożoność algorytmu za pomocą tak zwanej notacji asymptotycznej. Określa złożoność obliczeniową dla bardzo dużych problemów ($n \to \infty$).

Notacja duże O, czy też omikron pozwala wskazać rząd funkcji, która ogranicza funkcję złożoności obliczeniowej od góry. Czyli, która dla bardzo dużych problemów będzie co do wartości zawsze większa lub równa $T(n)$. Określa ona zatem pesymistyczne oszacowanie złożoności obliczeniowej (gorzej nie będzie).

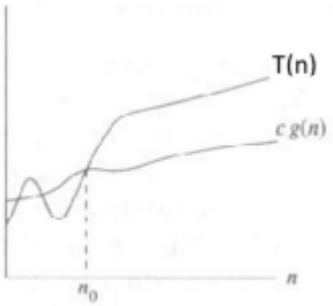


T(n) = O(g(n))

$$T(n) = O(g(n)) \iff T(n) <= c \cdot g(n) \tag{1}$$

$$T(n) = O(g(n)) \iff \lim_{n\to\infty} |\frac{T(n)}{g(n)}| < \infty \tag{2}$$

Notacja omega(Ω) pozwala wskazać rząd funkcji, która ogranicza funkcję złożoności obliczeniowej od dołu. Określa ona najbardziej optymistyczne oszacowanie złożoności obliczeniowej (lepiej nie będzie).

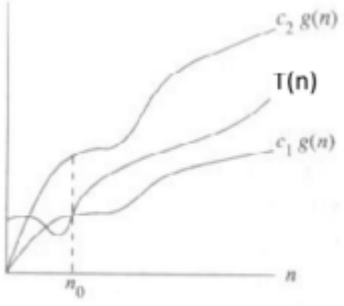


T(n) ≥ c· g(n)

$$T(n) = \Omega(g(n)) \iff T(n) >= c \cdot g(n) \tag{3}$$

$$T(n) = \Omega(g(n)) \iff \lim_{n\to\infty} |\frac{T(n)}{g(n)}| > 0 \tag{4}$$

Notacja teta(Θ) pozwala wskazać dwie funkcję danego rzędu, które ograniczają funkcję złożoności obliczeniowej od góry i od dołu. Pozwala to na określenie "przedziału", w którym zawiera się nasza funkcja złożoności obliczeniowej.



c_1·g(n) ≤ T(n) ≤ c_2·g(n)

$$T(n) = \Theta(g(n)) \iff c_1 \cdot g(n) <= T(n) <= c_2 \cdot g(n) \tag{5}$$

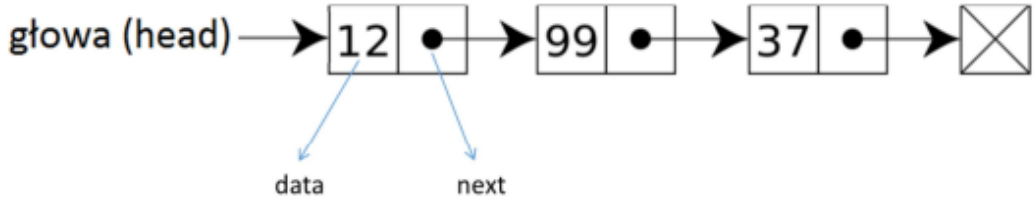
$$T(n) = \Theta(g(n)) \iff 0 < \lim_{n\to\infty} |\frac{T(n)}{g(n)}| < \infty \tag{6}$$

Z zasady do określenie złożoności duże O używamy jak najmniejszej funkcji. Zapewnia ona największą dokładność oszacowania.

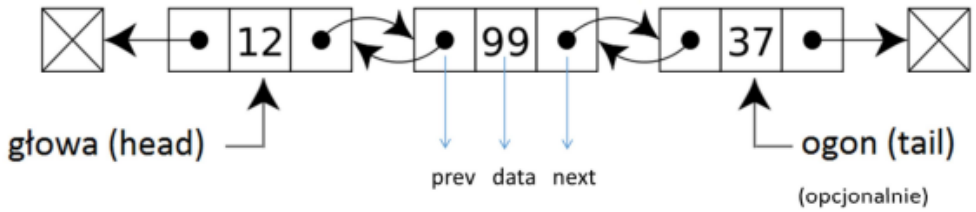
Podział klas złożoności:
Klasy złożoności można podzielić na:

- *złożoności jednostkowe* $O(1)$ - dla algorytmów, których złożoność nie zależy od ilości elementów.
- *złożoności wielomianowe* - dla algorytmów o złożonościach liniowych $O(n)$, kwadratowych $O(n^2)$, sześciennych $O(n^3)$ etc.
- *złożoności ograniczone przez wielomian* - najlepsze po jednostkowych złożoności. Są to złożoności logarytmiczne $O(\log n)$ i quasiliniowe $O(n \log n)$
- *złożoności niewielomianowe* - najgorsze złożoności. Są to złożoności wykładnicze $O(a^n)$ oraz silnia $O(n!)$.

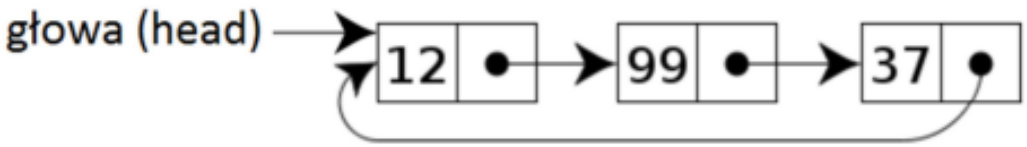
Lista jednokierunkowa:
Struktura danych wykorzystująca wskaźniki, gdzie każda komórka zawiera dane oraz wskazuje na komórkę kolejną. Lista jednokierunkowa posiada do tego dodatkowy wskaźnik wskazujący na pierwszy element (głowę). Elementy listy nie muszą tworzyć zwartego obszaru w pamięci. Ostatni element listy jednokierunkowej posiada wskaźnik na element pusty.



Lista dwukierunkowa:
Struktura danych wykorzystująca wskaźniki, gdzie każda komórka zawiera dane oraz wskazuje na komórkę poprzednią oraz kolejną. Lista jednokierunkowa posiada wskaźnik na swój pierwszy (głowa) oraz ostatni element (ogon). Poprzednikiem pierwszego elementu oraz następcą ostatniego elementu jest element pusty.



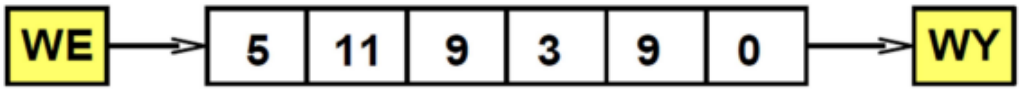
Lista cykliczna:
Lista jedno/dwukierunkowa, której ostatni element jako komórkę następną wskazuje element pierwszy (głowę).



Operacje na listach:
Na listach możemy wykonać następujące operacje:

- wyszukiwanie elementu - $O(n)$
- dodanie elementu na początek kolejki lub odczyt z początku kolejki - $O(1)$
- dodanie elementu na koniec kolejki lub usunięcie z końca kolejki - $O(n)$ bez ogona, $O(1)$ z ogonem
- dodanie elementu w określone miejsce lub usunięcie go stamtąd - $O(n)$

Kolejka (bufor):
Struktura danych, w której odczyt następuje tylko z pierwszej pozycji, a zapis można wykonać wyłącznie za ostatnim elementem. Operacja wczytania i odczytu z kolejki ma zatem złożoność $O(1)$. Operacja wyszukania lub usunięcia konkretnego elementu ma złożoność $O(n)$.



Stos:
Struktura danych, w której odczyt i zapis występuje wyłącznie na pierwszym elemencie (czy też przed nim). Złożoność pobrania lub dodania elementu ma zatem złożoność $O(1)$. Operacje wyszukania i usunięcia konkretnego elementu mają złożoność $O(n)$.