ZŁOŻONOŚĆ OBLICZENIOWA

Złożoność obliczeniowa to ilość zasobów komputerowych koniecznych do wykonania programu realizującego algorytm. Przedstawiamy ją jako funkcję pewnego parametru, określającego rozmiar rozwiązywanego zadania.

Ponieważ w przypadku szacowania złożoności obliczeniowej mówimy o czasie i pamięci, to wyróżniamy złożoność pamięciową i czasową.

Złożoność pamięciowa (zawsze jako funkcja rozmiaru danych!) to ilość pamięci wykorzystanej w celu realizacji algorytmu, wyrażana w liczbie bajtów lub liczbie zmiennych typów elementarnych.

Złożoność czasowa (zawsze jako funkcja rozmiaru danych!): jest to czas wykonania algorytmu wyrażany w standardowych jednostkach czasu, liczbie cykli procesora lub w liczbie wszystkich operacji. Możemy ją oznaczać jako T(n)

ZŁOŻONOŚĆ OBLICZENIOWA

Gdy zużycie zasobów zależy od rozkładu danych, to można mówić o:

- złożoności optymistycznej - określa zużycie zasobów dla najkorzystniejszego zestawu danych.

- złożoności średniej - określa zużycie zasobów dla typowych (tzw. losowych) danych.

  dokładnie to:

  Tśr = $\sum_{k=1}^{n} p_k(dane\ wejściowe\quad_k) * T_k(n)$, oczywiście $\sum_{k=1}^{n} p_k = 1$

- złożoności pesymistycznej - określa zużycie zasobów dla najbardziej niekorzystnego zestawu danych.

Pd – Sprawdzenie czy w zbiorze liczb istnieje co najmniej jedna liczba ujemna

ZŁOŻONOŚĆ OBLICZENIOWA

Złożoności wielomianowe:

(n) - liniowa np. $T(n) = 230n$

(n^2) - kwadratowa np. $T(n) = 12n^2 + 135n - 23$

(n^3) - sześcienna np. $T(n) = n^3 + 20n^2 - 19n + 1$

Złożoności ograniczone przez wielomian:

(log n) - logarytmiczna np. $T(n) = 3\log (n+1) - 2$

(nlog n) - quasiliniowa np. $T(n) = 3n\log (n+1) – 2$

Złożoności niewielomianowe:

(a^n) - wykładnicza np. $u(n) = e^n + n^{13} + n^2$

Zadania na liczenie T(n) – metoda dokładna

1. **Sumowanie liczb do 1 do n**
2. **Wyszukiwanie liczby ujemnej w tablicy**
3. **Sortowanie bąbelkowe**

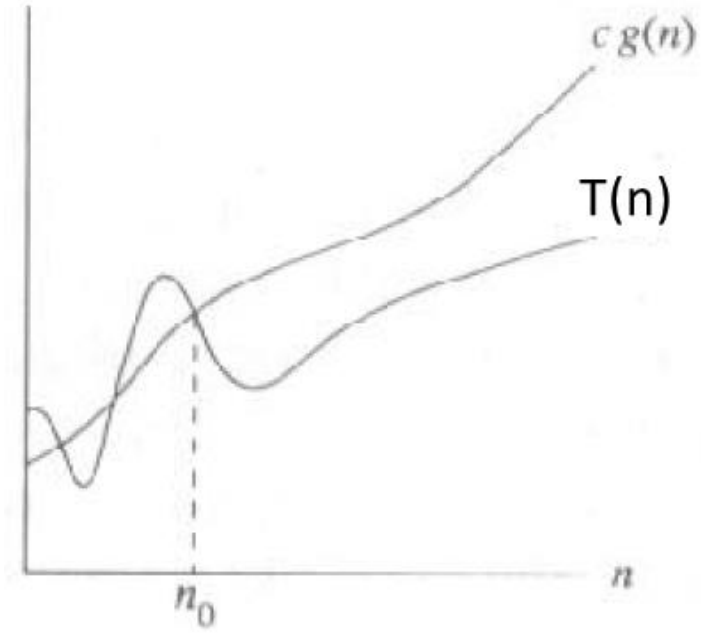
ZŁOŻONOŚĆ OBLICZENIOWA

Klasa złożoności obliczeniowej – określa rząd funkcji T(n) przy n->∞ (tzw. asymptotyczna granica)
Notacje określania złożoności obliczeniowej

Notacja O(duże O, omikron)
Mówimy, że T(n) = O(g(n)) (funkcja złożoności obliczeniowej T(n) jest rzędu funkcji g(n)) jeśli istnieje takie $n_0 \in N$ oraz takie $c \in R$, iż dla każdego $n \geq n_0$ prawdziwa jest nierówność:
$T(n) \leq c \cdot g(n)$

Przykład: $2n^2 = O(n^3)$ ($c=1$, $n_0=2$)
$2n^2 = O(n^2)$ ($c=3$, $n_0=1$)
Którą zatem g(n) wybrać ? Jak wyliczyć c i n_0 ?

O(1) – operacja jednostkowa (czas trwania nie zależy od n)



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

Notacja Θ(teta)

Mówimy, że T(n) = Θ (g(n)) jeśli istnieją stałe dodatnie c_1, c_2 i n_0 takie $n_0 \in N$, iż dla każdego $n \geq n_0$ prawdziwa jest nierówność:

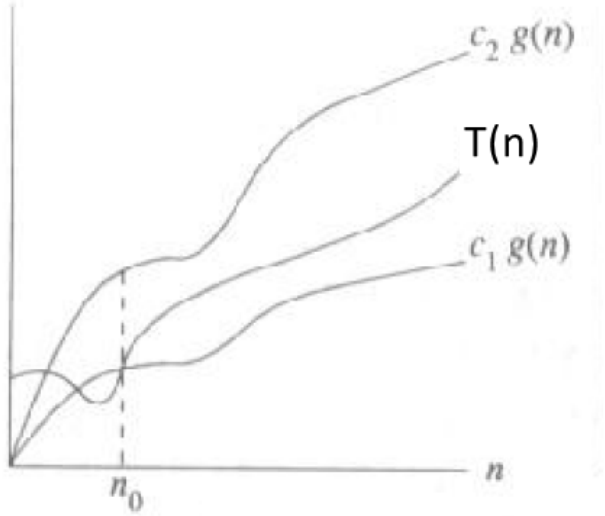
$$c_1 \cdot g(n) \leq T(n) \leq c_2 \cdot g(n)$$

g(n) nazywa się asymptotycznie dokładnym oszacowaniem. Można zauważyć, że

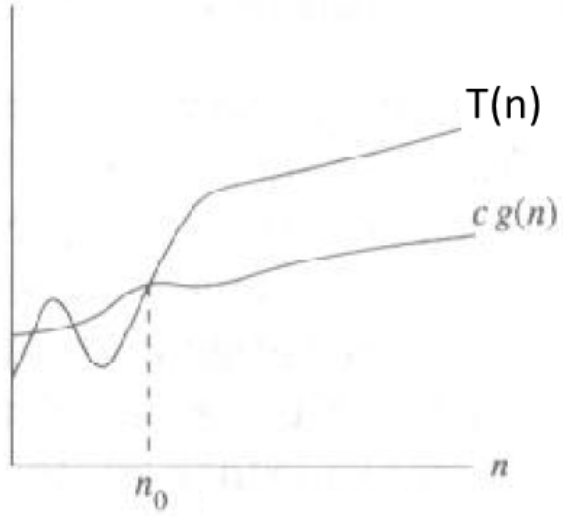
T(n) = Θ (g(n)) implikuje T(n) = O(g(n))

Notacja Ω(omega)

Mówimy, że T(n) = Ω (g(n)) jeśli istnieją stałe dodatnie c i $n_0 \in N$ takie, że dla każdego $n \geq n_0$ prawdziwa jest nierówność:



$c_1 \cdot g(n) \leq T(n) \leq c_2 \cdot g(n)$

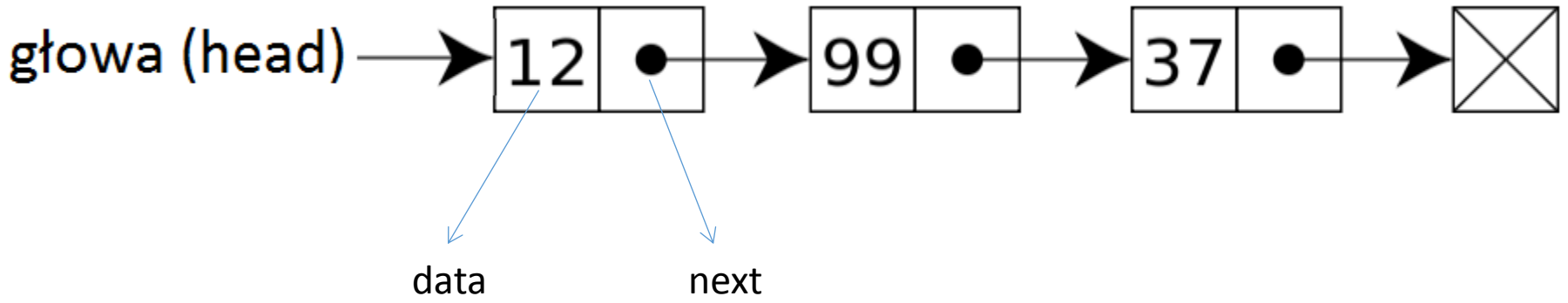


$T(n) \geq c \cdot g(n)$

LISTY

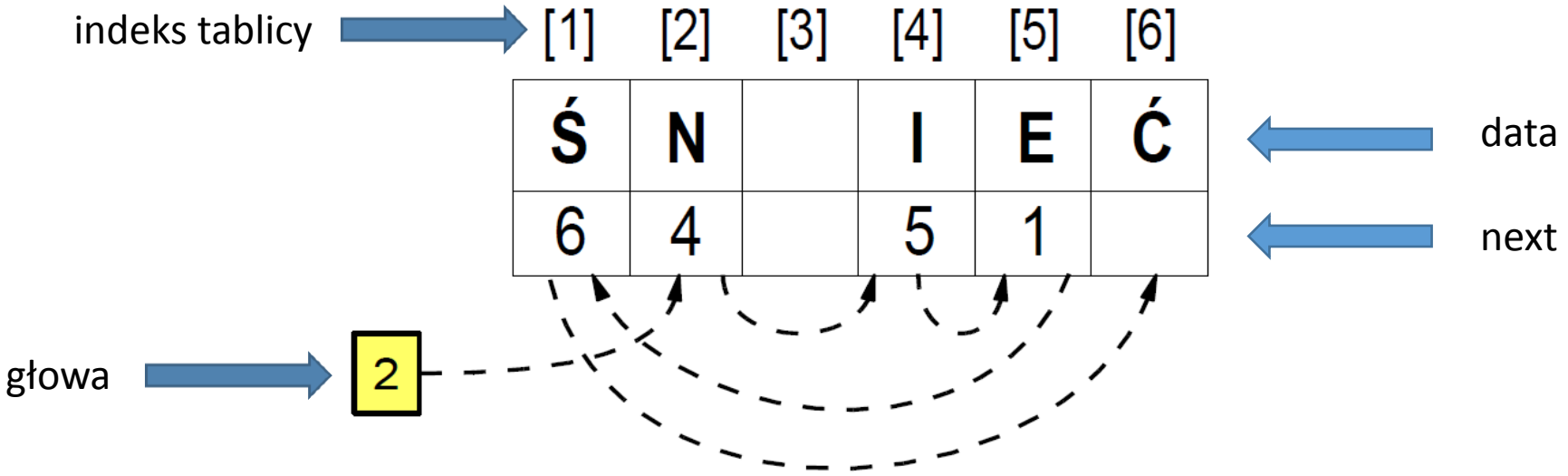
Lista - struktura wykorzystująca wskaźniki, gdzie każda komórka składa się z dwóch pól: pola danych mogącego być rozbudowana strukturą oraz wskaźnika na następny element. Ponadto lista musi mieć (co najmniej) jedna wyszczególniona komórkę zawierającą wyłącznie wskaźnik na pierwszy element , tzw. głowę. W przeciwieństwie do tablicy elementy listy nie muszą tworzyć zwartej pamięci.

Lista jednokierunkowa



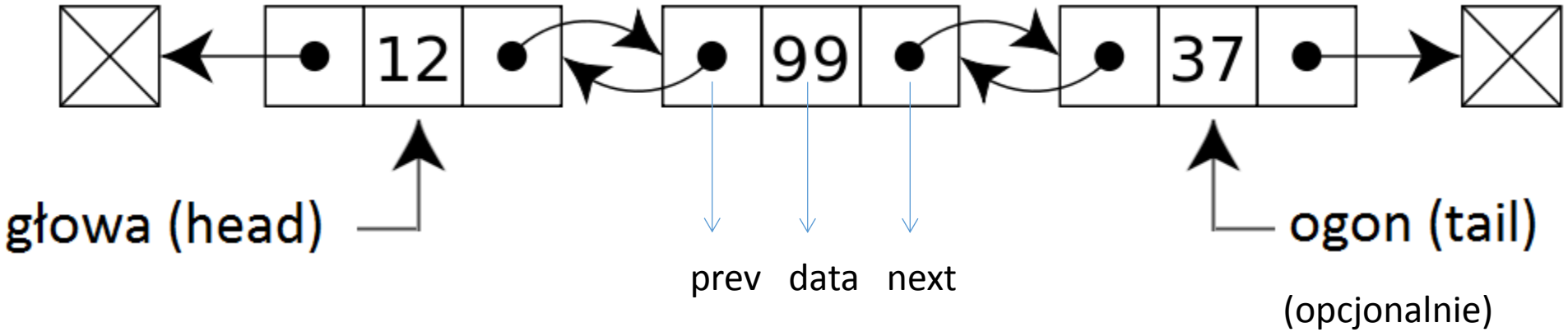

```
struct ElemList1
{
    int data; <- dowolny typ danych
    ElemList1 *next;  <- wskaźnik na następny element
}
```

Implementacja lista jednokierunkowej jako tablicy



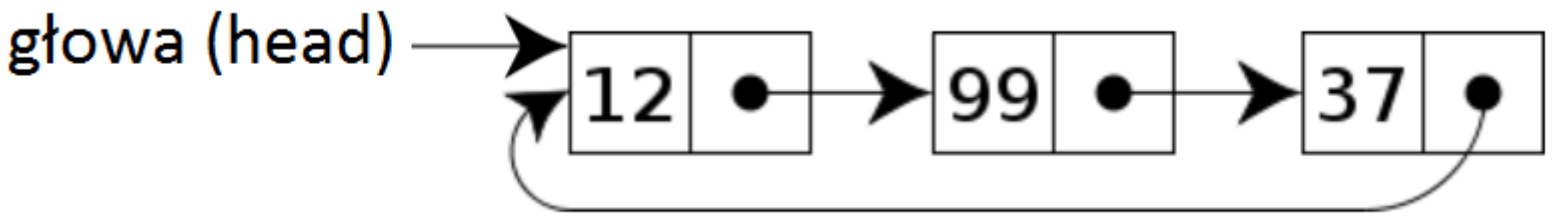
element pusty – np. w pole next wpisać 0

Lista dwukierunkowa



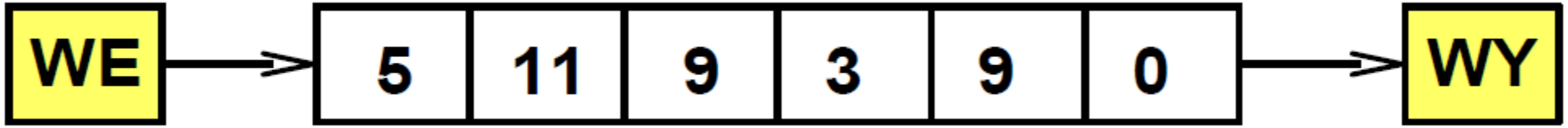

```
struct ElemList2
{
    int data; <- dowolny typ danych
    ElemList2 *next, *prev;  <- wskaźnik na następny element i poprzedni element
}
```

Lista cykliczna (może być oparta na liście jedno lub dwukierunkowej)



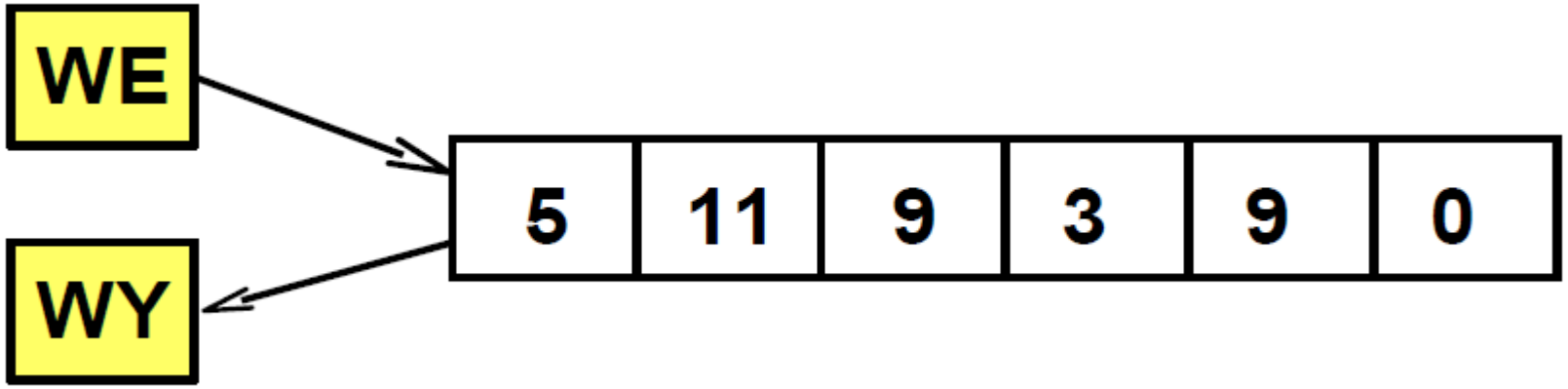
Kolejka (bufor)

Jest strukturą typu FIFO (ang. First In First Out), w której odczyt następuje tylko z pierwszej pozycji, a wstawienie nowego elementu może nastąpić tylko na koniec struktury (za ostatni element). Zatem dostęp (pobranie) oraz wstawienie nowego elementu zajmuje O(1) czasu, natomiast wyszukanie i usunięcie konkretnego elementu O(n).



Stos

Jest strukturą typu LIFO (ang. Last In First Out), czyli taką, w której dostęp (bezpośredni) jest tylko do pierwszego elementu, a dodanie nowego następuje przed pierwszym (na pierwszej pozycji). Zatem tak jak w przypadku kolejki złożoność operacji dostępu (pobrania) oraz dodania elementu wynosi O(1), natomiast dla wyszukania i usunięcia konkretnego elementu to O(n).



Operacje na liście:

- wyszukiwanie – O(n)
- dodawanie na początek – O(1)
- dodawanie na koniec – O(n) lub O(1) jeśli używamy ogona
- dodawanie w inne określone miejsce - O(n)
- usuwanie - analogicznie

Kontenery STL (wybrane):

queue: **template <class T, class Container = deque<T> > class queue;**

queue<int> intQueue – przykład definicji kolejki przechowywującego l. całk.

push – wstawienie elementu do kolejki

pop – usunięcie najstarszego elementu z kolejki

front – zwraca referencje do najstarszego elementu w kolejce

back – zwraca referencje do najmłodszego elementu w kolejce

stack: **template <class T, class Container = deque<T> > class stack;**

stack<int> intStack – przykład definicji zmiennej typu stos przechowywującego l. całk.

push – wstawienie elementu

pop – usunięcie elemetnu ze stosu

list: (implementacja listy dwukierunkowej)

list<int> intList – przykład definicji zmiennej typu lista przechowywującej l. całk.

push_front (push_back) – wstawienie elementu na początek(koniec) listy

pop_front, pop_back) – usunięcie elementu z początku (końca) listy

LISTY

Kontenery STL (wybrane):

deque: (double ended queue) – domyślny kontener dla stack i queue
  push_front (push_back) – wstawienie elementu na początek(koniec) listy
  pop_front (pop_back) – usunięcie elementu z początku (końca) listy
  front (back) – referencja do pierwszego (ostatniego) elementu listy
  [] – dostęp do elementu listy poprzez indeks

forward_list: lista jednokierunkowa (C11)
  push_front (push_back) – wstawienie elementu na początek listy
  pop_front (pop_back) – usunięcie elementu z końca listy
  insert_after (erase_after) – wstawienie (usunięcie) elementu po podanym elemencie (element podajemy jak wskaźnik – iterator)

LISTY

Zadania:
- wyświetlanie listy
- dodawanie na początek
- usuwanie z początku

Zadanie domowe:
-dodawanie na koniec listy jednokierunkowej
-usuwanie ostatniego elementu listy jednokierunkowej