Programowanie podstawowe

Łukasz Bojarski

Algorytm

Alan Turing

- angielski matematyk i kryptolog
- jeden z twórców informatyki
- twórca pojęcia maszyny Turinga
- twórca urządzenia służącego do łamania kodu Enigmy

Algorytm

Maszyna Turinga

- skończonego **alfabetu** symboli;
- skończonego zbioru **stanów**;
- nieskończonej **taśmy** z zaznaczonymi kwadratami, z których każdy może zawierać pojedynczy symbol;
- ruchomej **głowicy** odczytująco - zapisującej, która może wędrować wzdłuż taśmy przesuwając się na raz o jeden kwadrat
- diagramu przejść między **stanami**, zawierającego instrukcje, które powodują, że zmiany następują przy każdym zatrzymaniu się

Algorytm

jest skończonym, uporządkowanym ciągiem jasno zdefiniowanych czynności, koniecznych do wykonania postawionego zadania

Algorytm

Cechy algorytmów:

- poprawność
- jednoznaczność
- skończoność
- sprawność

Algorytm

Sposoby zapisywania:

- opis słowny
- lista kroków
- schemat blokowy
- pseudokod
- język programowania np. Java

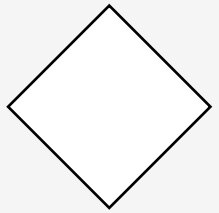
Algorytm

Schemat blokowy

- początek działań schematu

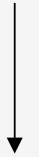
- blok wejścia-wyjścia

- blok obliczeniowy

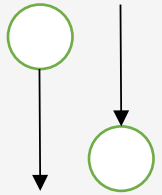


- blok decyzyjny

- zakończenie działania schematu



- łącznik poszczególnych bloków



- łącznik wewnątrzstronicowy

Algorytm

Zadanie

Narysuj schemat blokowy sprawdzający czy suma dwóch liczb jest podzielna przez 2.

Algorytm

Pytanie

Kto wie jak zrobić pętlę *for* w schemacie blokowym?

Algorytm

Pseudokod

:= albo ← przypisanie wartości do zmiennej – Java =

= równe – Java ==

<> nierówne/różne – Java !=

< mniejsze od

> większe od

<= mniejsze lub równe

>= większe lub równe

Algorytm

Pseudokod

set a – pobranie zmiennej a

print a – wypisanie zmiennej a

if a > 0 **then** - instrukcja warunkowa

    <instrukcja>

else

    <instrukcja>

Algorytm

Pseudokod

```
for i := 1 to 10 do                - pętla for
   <instrukcja>

while i < 10 do                    - pętla while
   <instrukcja>

begin                              - blok kodu
  <instrukcja>
end
```

Algorytm

Zadanie

Napisz pseudokod dla poprzedniego zadania.

Struktury danych

sposób przechowywania informacji w komputerze

Struktury danych

Przykłady:

- rekord
- tablica
- stos
- kolejka
- lista
- drzewo
- graf

Struktury danych

Tablica – kontener uporządkowanych danych, w którym poszczególne elementy dostępne są za pomocą kluczy (indeksu).

int[] tab = new int[10] – tablica jednowymiarowa typów int

int[][] tab = new int[10][10] – tablica dwuwymiarowa typów int

Struktury danych

Zadanie

Dana jest tablica liczb całkowitych. Wypisz:

- wszystkie liczby od końca będące na parzystych indeksach,
- sumę tylko tych liczb podzielnych przez 3,
- wynik sumy 5 początkowych liczb i odejmując ostatni element tablicy, zakładając, że tablica jest rozmiaru co najmniej 6.

Struktury danych

Stos - liniowa struktura danych, w której dane dokładane są na wierzch stosu i z wierzchołka stosu są pobierane

(LIFO – Last In, First Out)

Operacje:

- push – dodanie na szczyt stosu nowego elementu
- pop – usunięcie ze szczytu stosu elementu
- isEmpty– zwraca *true*, jeżeli stos nie zawiera żadnego elementu, wpp. *false*
- peek– zwraca element ze szczytu stosu

Struktury danych

Zadanie

Stwórz stos za pomocą tablicy.

Struktury danych

Zadanie

Stwórz stos za pomocą listy.

Struktury danych

Kolejka - liniowa struktura danych, w której nowe dane dopisywane są na końcu kolejki, a z początku kolejki pobierane są dane do dalszego przetwarzania.

(FIFO – First In, First Out)

Operacje:

- add/engueue – dodanie nowego elementu na koniec kolejki
- poll/degueue – usunięcie pierwszego elementu z kolejki
- isEmpty – zwraca *true*, jeżeli kolejka nie zawiera żadnego elementu, wpp. *false*
- peek – zwraca pierwszy element w kolejce

Zadanie

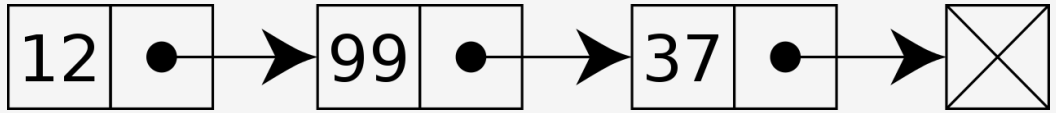
Korzystając z poprzedniego zadania, stwórz kolejkę za pomocą listy.

Struktury danych

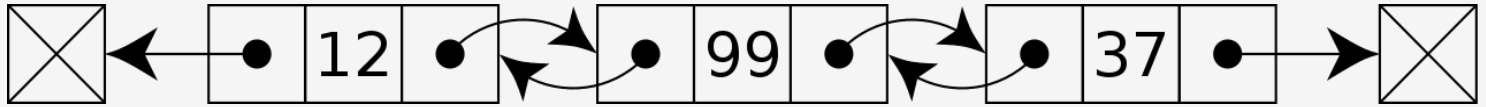
Lista - struktura danych służąca do reprezentacji zbiorów dynamicznych, w której elementy ułożone są w liniowym porządku.

Przykłady:

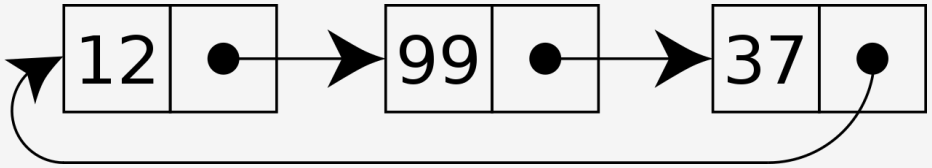
- jednokierunkowa



- dwukierunkowa



- cykliczna



Struktury danych

Zadanie domowe

Na podstawie zdobytej wiedzy, stwórz listę dwukierunkową.

Elementy pojedynczego obiektu:

- value
- prev
- next

Elementy listy:

- first albo head
- last albo tail

Metody listy:

- addFirst/addLast
- peekFirst/peekLast
- pollFirst/pollLast
- isEmpty
- show/showReverse

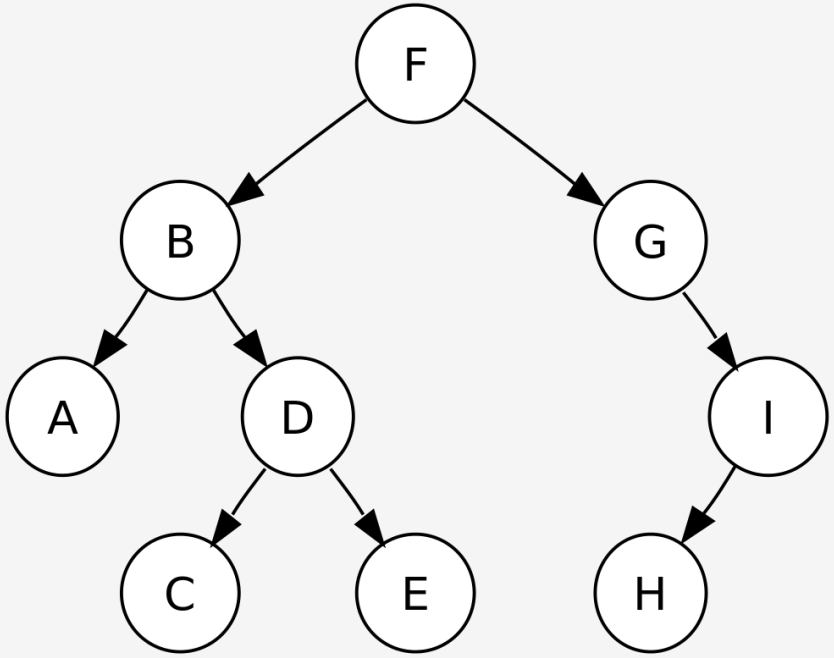
Struktury danych

Drzewo - jest strukturą danych zbudowaną z elementów, które nazywamy **węzłami**. Dane przechowuje się w węzłach drzewa. Węzły są ze sobą powiązane w sposób hierarchiczny za pomocą **krawędzi**, które zwykle przedstawia się za pomocą strzałki określającej hierarchię. Pierwszy węzeł drzewa nazywa się **korzeniem**. Od niego "wyrastają" pozostałe węzły, które nazywamy **dziećmi**. Węzeł nadrzędny w stosunku do dziecka nazywamy **rodzicem.** Jeśli węzeł nie posiada dzieci, to nazywa się **liściem.**

Drzewo, w którym węzły mogą posiadać co najwyżej dwoje dzieci, nazywa się **drzewem binarnym**

Zadanie

Dla przedstawionego obok drzewa wyznacz korzeń, rodziców, dzieci oraz liście. Czy jest to drzewo binarne?



Struktury danych

Drzewo

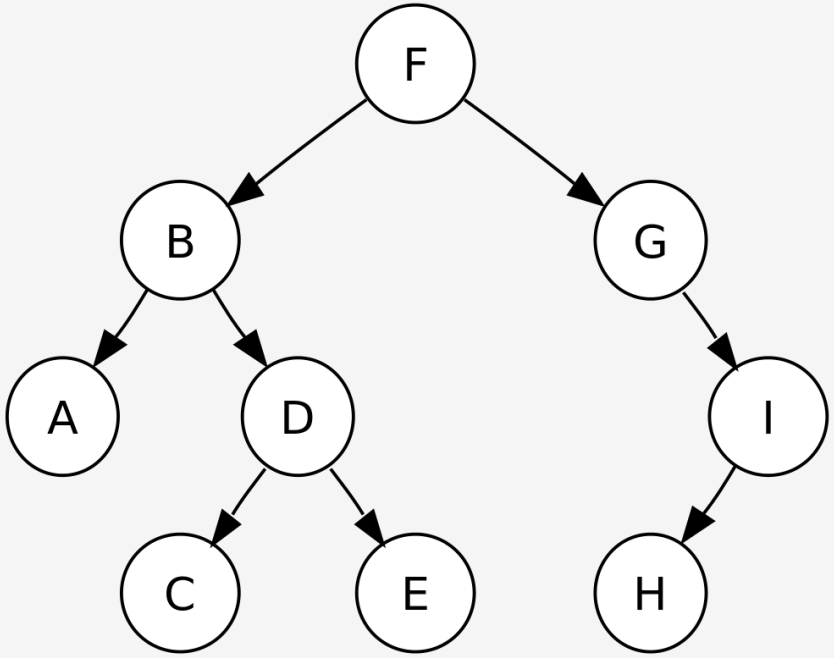
Ciąg krawędzi łączących węzły nazywa się **ścieżką**.

Liczba krawędzi w ścieżce od korzenia do węzła jest nazywana **długością** – liczba ta określa **poziom** węzła.

Wysokością drzewa jest największy poziom istniejący w drzewie.

Zadanie

Dla przedstawionego obok drzewa wyznacz przykładowe ścieżki oraz ich długości. Jaka jest wysokość drzewa?



Struktury danych

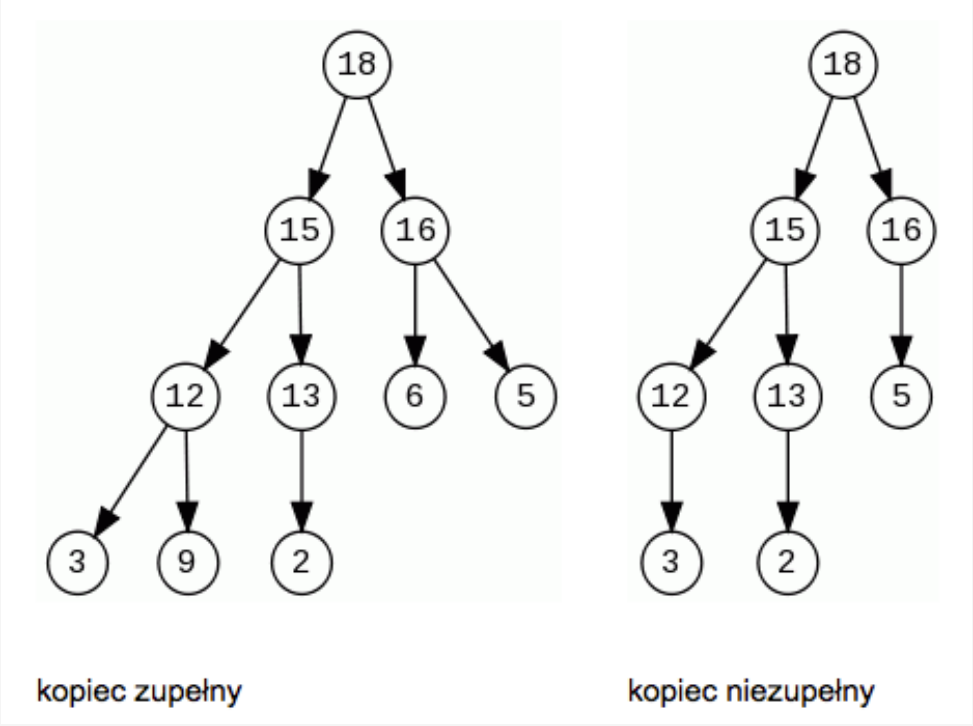
Kopiec (inaczej stóg) – jest drzewem binarnym, spełniającym warunek, że każdy następnik jest niewiększy od swojego poprzednika.

czyli:

- w korzeniu kopca znajduje się największy lub jeden z grupy największych o identycznej wartości
- na ścieżkach (połączeniach między węzłami), od korzenia do liścia, elementy są posortowane nierosnąco

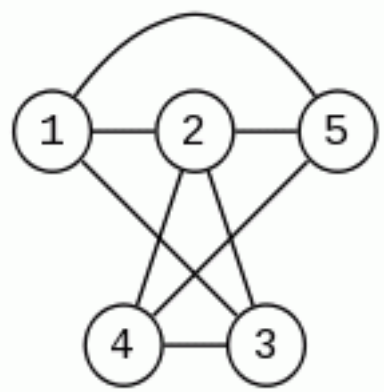
Przykład wykorzystania: sortowanie przez kopcowanie (heapsort)

Kopiec zupełny – to kopiec będący **zupełnym** drzewem binarnym.

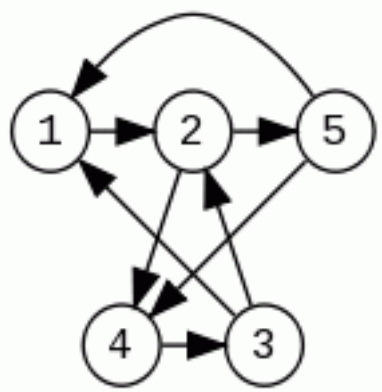


Struktury danych

Graf - struktura danych, która składa się z wierzchołków i krawędzie, przy czym poszczególne wierzchołki (węzły) mogą być połączone krawędziami (skierowanymi lub nieskierowanymi) w taki sposób, iż każda krawędź zaczyna się i kończy w którymś z wierzchołków.



graf nieskierowany



graf skierowany

Algorytmy

klasyfikacje i sposoby implementacji

Algorytmy

Klasyfikacja algorytmów:

- **dziel i zwyciężaj** – dzielimy problem na kilka mniejszych, a te znowu dzielimy, aż ich rozwiązania staną się oczywiste
- **metoda zachłanna** – nie analizujemy podproblemów dokładnie, tylko wybieramy najbardziej obiecującą w danym momencie drogę rozwiązania
- **programowanie dynamiczne** – problem dzielony jest na kilka, ważność każdego z nich jest oceniana i po pewnym wnioskowaniu wyniki analizy niektórych prostszych zagadnień wykorzystuje się do rozwiązania głównego problemu
- **programowanie liniowe** – oceniamy rozwiązanie problemu przez pewną funkcję jakości i szukamy jej minimum
- **wyszukiwanie wyczerpujące (*brute force)*** – przeszukujemy zbiór danych, aż do odnalezienia rozwiązania
- **heurystyka** – człowiek na podstawie swojego doświadczenia tworzy algorytm, który działa w najbardziej prawdopodobnych warunkach

Algorytmy

Programowanie liniowe – przykład:

Mieszanka	Kawa			zysk
	brazylijska [kg]	kolumbijska [kg]	peruwiańska [kg]	[zł/kg]
Northwest Passage	2	4	4	80
Sunrise Blend	4	5	1	60
Harbormaster	3	3	4	40
French Expedition	7	2	1	50
zapas [kg]	800	640	600	

$$\text{zmaksymalizować} \quad f(x_1, x_2, x_3, x_4) = 80x_1 + 60x_2 + 30x_3 + 50x_4$$

Algorytmy

Sposoby implementacji:

- **proceduralność** – podział algorytmu na szereg procedur
- **sekwencyjność** – wykonywanie poszczególnych procedur algorytmu, według kolejności ich wywołań
- **równoległość** – część zadań wykonuje się jednocześnie, wymieniając się danymi
- **rekurencyjność** – funkcja wywołuje sama siebie, aż do uzyskania wyniku
- **obiektowość** – procedury i dane łączy się w pewne klasy reprezentujące najważniejsze elementy algorytmu oraz stan wewnętrzny wykonującego je systemu
- **probabilistyczność** (**randomizowość)** – działa szybko, ale wynik nie jest pewny

Przykładowe algorytmy

Zadanie

Napisz program, który będzie sprawdzał czy w tablicy znajduje się dana liczba.

Przykładowe algorytmy

Binary search (wyszukiwanie binarne) – szukanie danego elementu w tablicy uporządkowanej. Metoda „dziel i zwyciężaj”.

Przykładowe algorytmy

Binary search (wyszukiwanie binarne) – pseudokod:

```
A := [...]       { n-elementowa tablica uporządkowana }
lewo := 1        { indeks początku przedziału }
prawo := n     { indeks końca przedziału - początkowo cała tablica A }

y := poszukiwana wartość
indeks := pusty

while lewo < prawo do
   begin
      środek := (lewo + prawo) div 2; { dzielenie całkowitoliczbowe }

      if A[środek] < y then
         lewo := środek + 1
      else
         prawo := środek;
   end;

if A[lewo] = y then
   indeks := lewo
else
   indeks := brak;
```

Przykładowe algorytmy

Zadanie

Narysuj schemat blokowy, a następnie zaimplementuj algorytmu wyszukiwania binarnego.

Przykładowe algorytmy

Algorytm Euklidesa (GCD) - wyznacza największy wspólny dzielnik (NWD) dwóch liczb.

Przykładowe algorytmy

Algorytm Euklidesa (GCD) - wyznacza największy wspólny dzielnik (NWD) dwóch liczb.

Lista kroków:

1. Weźmy dwie liczby całkowite dodatnie a i b.
2. Jeśli b = 0 to przejdź do kroku 3., wpp. wykonaj:
   2.1 Jeśli a > b to a := a – b
   2.2 Wpp. b := b – a
   2.3 Przejdź do kroku 2.
3. a jest szukanym największym dzielnikiem.
4. Koniec

Przykładowe algorytmy

Zadanie

Narysuj schemat blokowy, napisz pseudokod a następnie zaimplementuj algorytm Euklidesa wykorzystujący odejmowanie.

Przykładowe algorytmy

Algorytm Euklidesa (GCD) – wykorzystujący resztę z dzielenia.

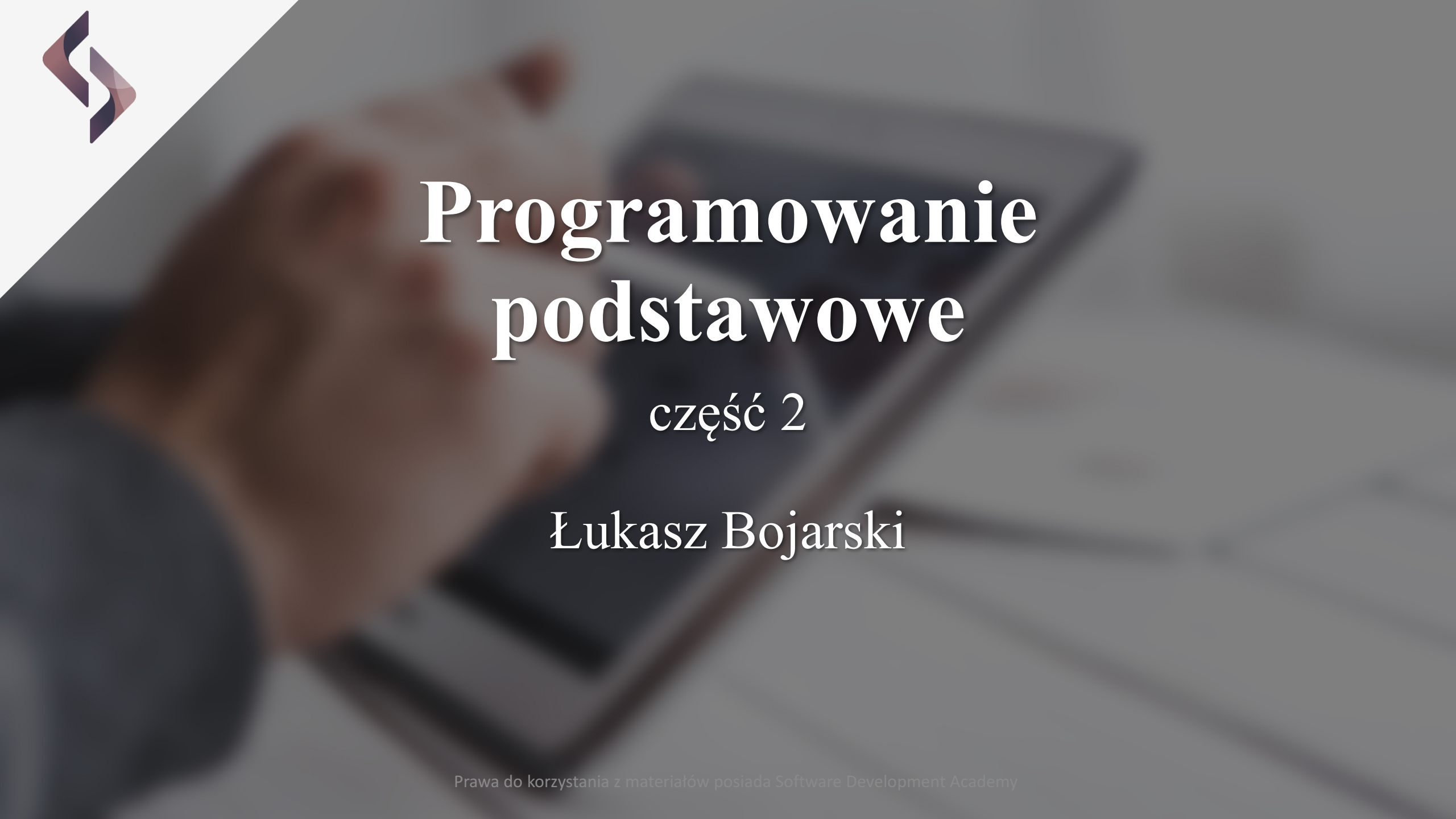
Lista kroków:

1. Weźmy dwie liczby całkowite dodatnie a i b.
2. Jeśli b = 0 to przejdź do kroku 3., wpp. wykonaj:
   2.1 r := a mod b
   2.2 a := b
   2.3 b := r
   2.4 Przejdź do kroku 2.
3. a jest szukanym największym dzielnikiem.
4. Koniec

Przykładowe algorytmy

Zadanie

Zaimplementuj algorytm Euklidesa wykorzystujący resztę z dzielenia.



Programowanie podstawowe

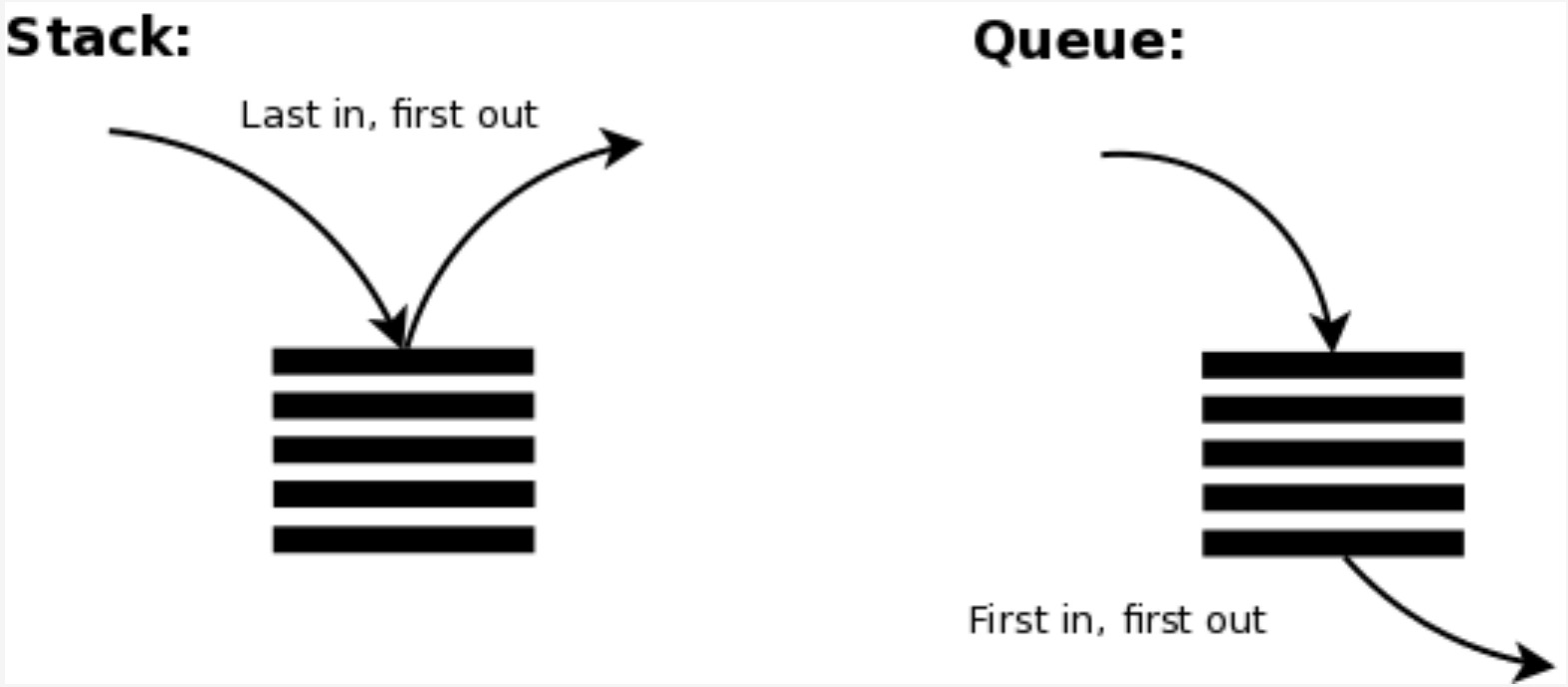
część 2

Łukasz Bojarski

Powtórka

Powtórzenie

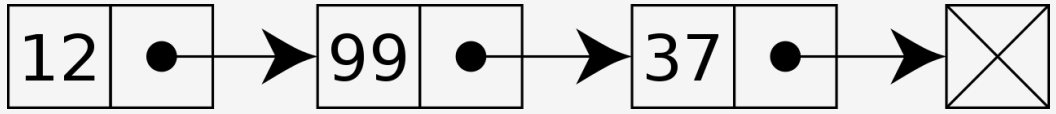
Powtórka



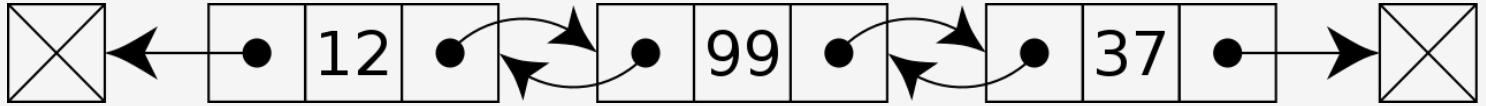
Powtórka

Lista

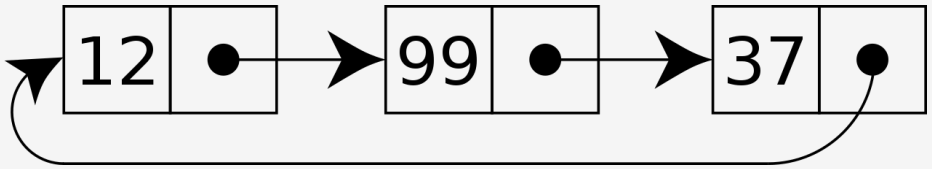
- jednokierunkowa



- dwukierunkowa



- cykliczna



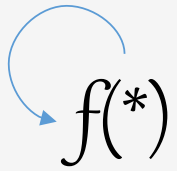
Rekurencja

Rekurencja

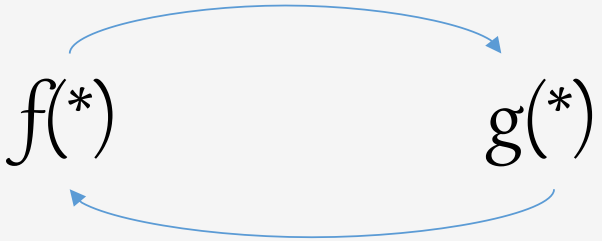
inaczej rekursja, polega na wywołaniu przez funkcję samej siebie.

Rekurencja

inaczej rekursja, polega na wywołaniu przez funkcję samej siebie.



rekurencja bezpośrednia



rekurencja pośrednia

Rekurencja

silnia

Rekurencja - silnia

Zadanie

Napisz program, który liczy silnię.

Rekurencja – silnia iteracyjnie

```
int silnia(int n) {
        int wynik = 1;
        for ( int i=1; i<=n; i++ )
                wynik *= i;             // wynik = wynik * i;
        return wynik ;
}
```

Rekurencja - silnia

silnia(0) = 1

silnia(1) = 1

silnia(n) = n * silnia(n-1)

Rekurencja - silnia

dla n = 3

silnia(3)
3 *silnia(2)
2 * silnia(1)
return 1
return 2 * 1
return 3 * 2

Wynik: 6

Rekurencja

Co przypomina rekurencja?

Rekurencja

Ciąg Fibonacciego – ciąg liczb naturalnych określony rekurencyjnie w sposób następujący:

Pierwszy wyraz jest równy 0, drugi jest równy 1, każdy następny jest sumą dwóch poprzednich.

Rekurencja

Ciąg Fibonacciego – ciąg liczb naturalnychokreślony rekurencyjnie w sposób następujący:

Pierwszy wyraz jest równy 0, drugi jest równy 1, a każdy następny jest sumą dwóch poprzednich.

$$F_n := \begin{cases} 0 & \text{dla } n = 0; \\ 1 & \text{dla } n = 1; \\ F_{n-1} + F_{n-2} & \text{dla } n > 1. \end{cases}$$

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
0	1	1	2	3	5	8	13	21	34	55

Rekurencja

Ciąg Fibonacciego

```
public int fib(int n) {
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

Rekurencja

NWD i binary search rekurencyjnie

Rekurencja

NWD

```
public int NWD(int a, int b) {
    if (b != 0)
        return NWD(b, a%b);
    else
        return a;
}
```

Rekurencja

Binary search

```
public int binarySearch(int A[], int lewo, int prawo, int szukany) {
        if (lewo > prawo)
                return -1;

        int srodek = (lewo + prawo) / 2;

        if(A[srodek] == szukany)
                return srodek;

        if(A[srodek] < szukany)
                return binarySearch(A, srodek + 1, prawo, szukany);
        else
                return binarySearch(A, lewo, srodek – 1, szukany);
}
```

Złożoność obliczeniowa

ilość zasobów komputera jakiej potrzebuje dany algorytm.

Złożoność obliczeniowa

Oblicz złożoność czasową:

```
public int suma(int n) {
    int wynik = 0;                  // t
    int i = 1;                      // t

    while(i <= n) {                 // t * n
        wynik = wynik + i;          // 2t * n
        i++;                        // t * n
    }

    return wynik;                   // t
}
```

f(n) = t + t + t*n + 2t*n + t*n + t = 4t*n + 3t

O(n)

Złożoność obliczeniowa

Oblicz złożoność czasową:

```
public int suma(int n) {
    int wynik = n * (n + 1) / 2;        // t
    return wynik;                       // t
}
```

f(n) = t + t + 2*t

O(1)

Złożoność obliczeniowa

Oblicz złożoność czasową:

```
public void wypisz(int n) {
    for(int i = 0 ; i < n ; i++) {                    // t * n
        for (int j = 0 ; j < n ; j++) {               // t * n * n
            System.out.println(i + " " + j); // t * n * n
        }
    }
}
```

f(n) = t*n + 2*t*n^2

O(n^2)

Złożoność obliczeniowa

Oblicz złożoność czasową:

```
public void wypisz(int n) {
    for(int i = 0 ; i < n ; i++) {                          // t * n
        for (int j = 0 ; j < n ; j++) {                     // t * n * n
            for (int k = 0 ; k < n ; k++) {                 // t * n * n * n
                System.out.println(i + " " + j + " " + k);  // t * n * n * n
            }
        }
    }
}
```

f(n) = tn + tn^2 + 2tn*3

O(n^3)

Złożoność obliczeniowa

Typy:

- czasowa
- pamięciowa

Rzędy złożoności:

- O(1) – stała
- O(n) – liniowa
- $O(n^2)$ – kwadratowa
- $O(n^x)$ – wielomianowa
- O(log(n)) – logarytmiczna

Złożoność obliczeniowa

Notacja O (dużego O)

$$\forall n \geqslant n_0 : f(n) \leqslant c * g(n)$$

Notacja Θ (theta)

$$\forall n \geqslant n_0 : c_1 * g(n) \geqslant f(n) \geqslant c_2 * g(n)$$

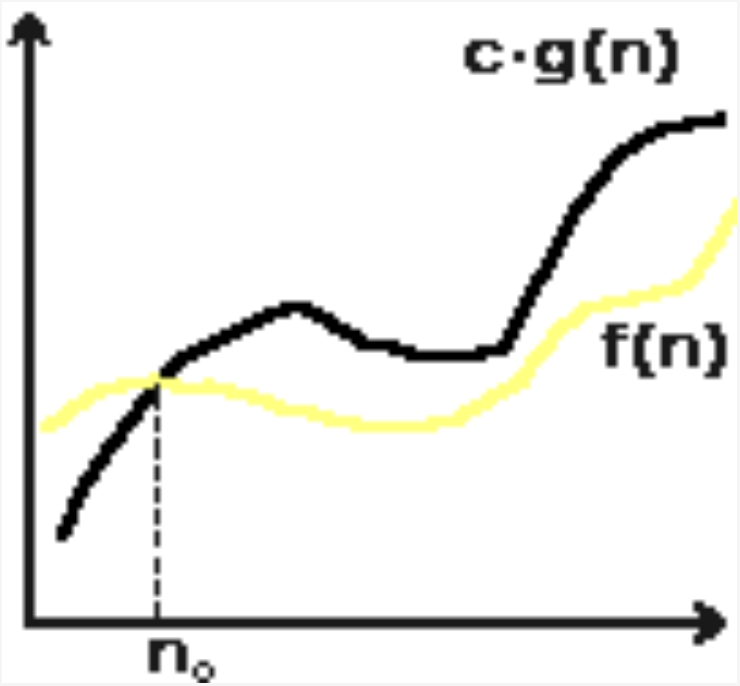
Notacja Ω (omega)

$$\forall n \geqslant n_0 : f(n) \geqslant c * g(n)$$

Złożoność obliczeniowa

Notacja O (dużego O)

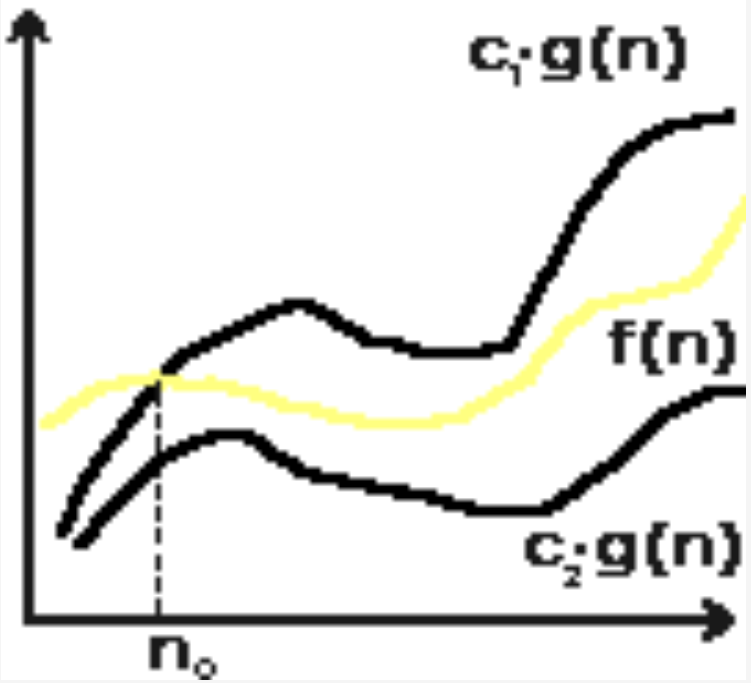
$$\forall n \geqslant n_0 : f(n) \leqslant c * g(n)$$



Złożoność obliczeniowa

Notacja Θ (theta)

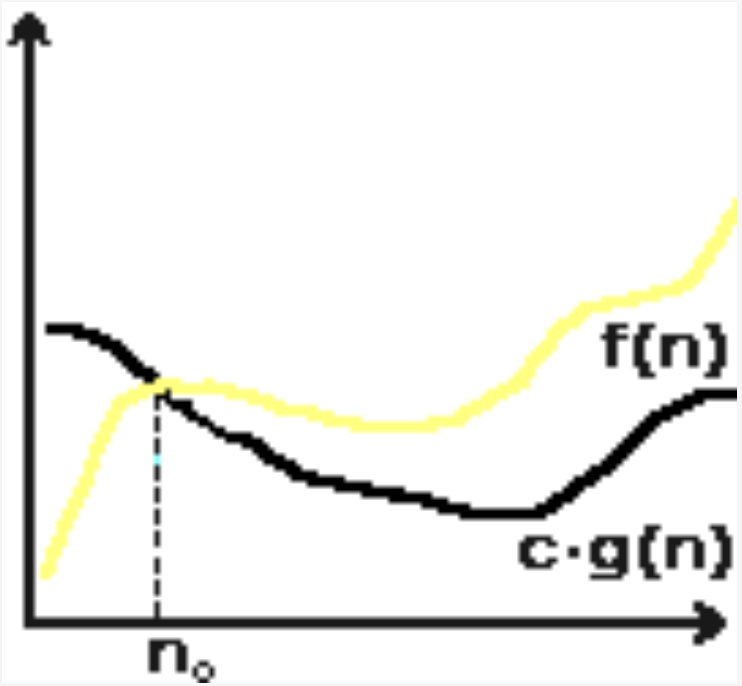
$$\forall n \geqslant n_0 : c_1 * g(n) \geqslant f(n) \geqslant c_2 * g(n)$$



Złożoność obliczeniowa

Notacja Ω (omega)

$\forall n \geqslant n_0 : f(n) \geqslant c * g(n)$



Złożoność obliczeniowa

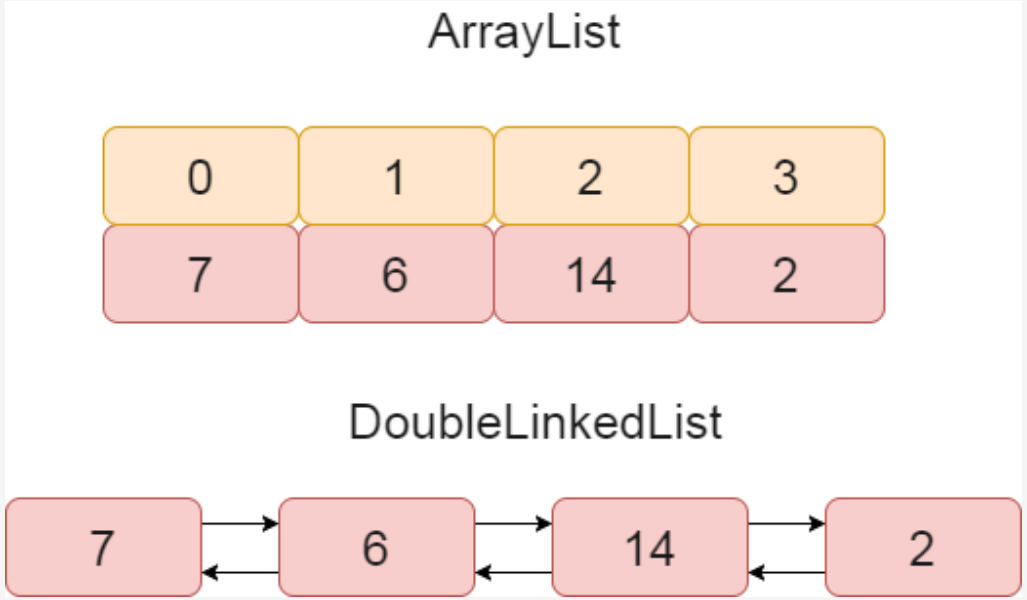
n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
logarytmiczna (log n)	0	0	0	1	1	1	1	1	1	1	1	1	1	1
liniowa (n)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
liniowo logarytmiczna (nlog n)	0	1	1	2	3	5	6	7	9	10	11	13	14	16
kwadratowa (n^2)	1	4	9	16	25	36	49	64	81	100	121	144	169	196
sześcienna (n^3)	1	8	27	64	125	216	343	512	729	1000	1331	1728	2197	2744
wykładnicza (2^n)	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
silnia (n!)	1	2	6	24	120	720	5040	40320	362880	3628800	39916800	4,79E+08	6,23E+09	8,72E+10

Złożoność obliczeniowa

Porównanie złożoności obliczeniowej list w Javie:

- ArrayList
- LinkedList

Złożoność obliczeniowa



	add	**remove**	**get**	**contains**
ArrayList	O(1)	O(n)	O(1)	O(n)
LinkedList	O(1)	O(1)	O(n)	O(n)

Złożoność obliczeniowa

Zadanie

Oblicz złożoność obliczeniową:

- Binary search
- Silni

Sortowanie

Sortowanie

polega na uporządkowaniu zbioru danych względem pewnych cech charakterystycznych dla każdego elementu tego zbioru

Sortowanie

Zadanie

Wczytaj tablicę a następnie posortuj ją niemalejąco

Sortowanie

Sortowanie przez wybieranie (select sort)

Działanie:

1. Szukamy najmniejszego elementu w zbiorze i zamieniamy go z elementem stojącym na pozycji pierwszej.
2. Następnie szukamy znowu elementu najmniejszego w zbiorze pominiętym o pierwszy element i wstawiamy go na pozycję drugą.
3. Czynności powtarzamy do momentu otrzymania jednoelementowego podzbioru

Sortowanie

Start
$for(i := 0\ ;\ i < n - 1; i++)$
  $for(j := i + 1\ ;\ j < n; j++)$
    Jeżeli $a[j] < a[i]$
      zamień $a[j] oraz\ a[i]$
Koniec

Sortowanie przez wybieranie (select sort)

Działanie:

1. Szukamy najmniejszego elementu w zbiorze i zamieniamy go z elementem stojącym na pozycji pierwszej.
2. Następnie szukamy znowu elementu najmniejszego w zbiorze pominiętym o pierwszy element i wstawiamy go na pozycję drugą.
3. Czynności powtarzamy do momentu otrzymania jednoelementowego podzbioru

Sortowanie

Sortowanie bąbelkowe (bubble sort) - polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje tablicę.

Jest jednym z najstarszych algorytmów sortujących.

Wizualizacja:
https://upload.wikimedia.org/wikipedia/commons/c/c8/Bubble-sort-example-300px.gif

Sortowanie

Sortowanie bąbelkowe (bubble sort) - polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje tablicę.

Jest jednym z najstarszych algorytmów sortujących.

Wizualizacja:
https://upload.wikimedia.org/wikipedia/commons/c/c8/Bubble-sort-example-300px.gif

K01: Dla $j = 1, 2, \ldots, n-1$, wykonuj K02

K02: Dla $i = 1, 2, \ldots, n-1$, wykonuj

jeśli $d[i] > d[i+1]$, to

$d[i] \leftrightarrow d[i+1]$

K03: Koniec

Sortowanie

Sortowanie przez zliczanie (counting sort) - metoda sortowania danych, która polega na sprawdzeniu ile wystąpień kluczy mniejszych od danego występuje w sortowanej tablicy.

Sortowanie

Sortowanie przez zliczanie (counting sort) - metoda sortowania danych, która polega na sprawdzeniu ile wystąpień kluczy mniejszych od danego występuje w sortowanej tablicy.

```
function countingSort(array, k) is
  count ← new array of k zeros
  for i = 1 to length(array) do
    count[array[i]] ← count[array[i]] + 1
  for i = 2 to k do
    count[i] ← count[i] + count[i - 1]
  for i = length(array) downto 1 do
    output[count[array[i]]] ← array[i]
    count[array[i]] ← count[array[i]] - 1
  return output
```

Sortowanie

Sortowanie przez wstawianie (insert sort)

```
for (int i = 1; i < tab.length; i++) {
    for (int j = i ; j >= 1 && tab[j-1] > tab[j] ; j--) {
        swap( i: j-1, j, tab);
    }
}
```

Działanie:

1. Bierzemy dowolny element ze zbioru nieposortowanego.
2. Porównujemy go z kolejnymi elementami zbioru posortowanego, aż nie napotkamy elementu równego lub większego, bądź nie znajdziemy się na początku/końcu zbioru posortowanego.
3. Wyciągnięty element wstawiamy na miejsce gdzie skończyliśmy porównywać.

Wizualizacja:
https://upload.wikimedia.org/wikipedia/commons/0/0f/Insertion-sort-example-300px.gif

Sortowanie

Sortowanie szybkie (quick sort) – jeden z popularniejszych algorytmów sortowania. Wykorzystuje metodę „dziel i zwyciężaj”.

Działanie:

1. Wybierany jest jeden element w sortowanej tablicy. Może być elementem środkowym, pierwszym, ostatnim, losowym (tzw. pivot).

2. Następnie ustawiamy elementy nie większe na lewo tej wartości, natomiast nie mniejsze na prawo.

3. W ten sposób powstaną nam dwie części tablicy (niekoniecznie równe), gdzie w pierwszej części znajdują się elementy nie większe od drugiej.

4. Następnie każdą z tych podtablic sortujemy osobno według tego samego schematu.

Wizualizacja:
https://ds055uzetaobb.cloudfront.net/image_optimizer/904290ba2b43687554b1d074d091367f370a0c08.gif

Sortowanie

Sortowanie szybkie (quick sort) – jeden z popularniejszych algorytmów sortowania. Wykorzystuje metodę „dziel i zwyciężaj”.

```
QUICKSORT(A, p, r)
  if p < r
      q = PARTITION(A, p, r)
      QUICKSORT(A, p, q − 1)
      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
  x = A[r]
  i = p − 1
  for j = p to r − 1
      if A[j] ≤ x
          i = i + 1
          exchange A[i] with A[j]
  exchange A[i + 1] with A[r]
  return i + 1
```

Sortowanie

Quick sort vs bubble sort

https://youtu.be/aXXWXz5rF64?t=106

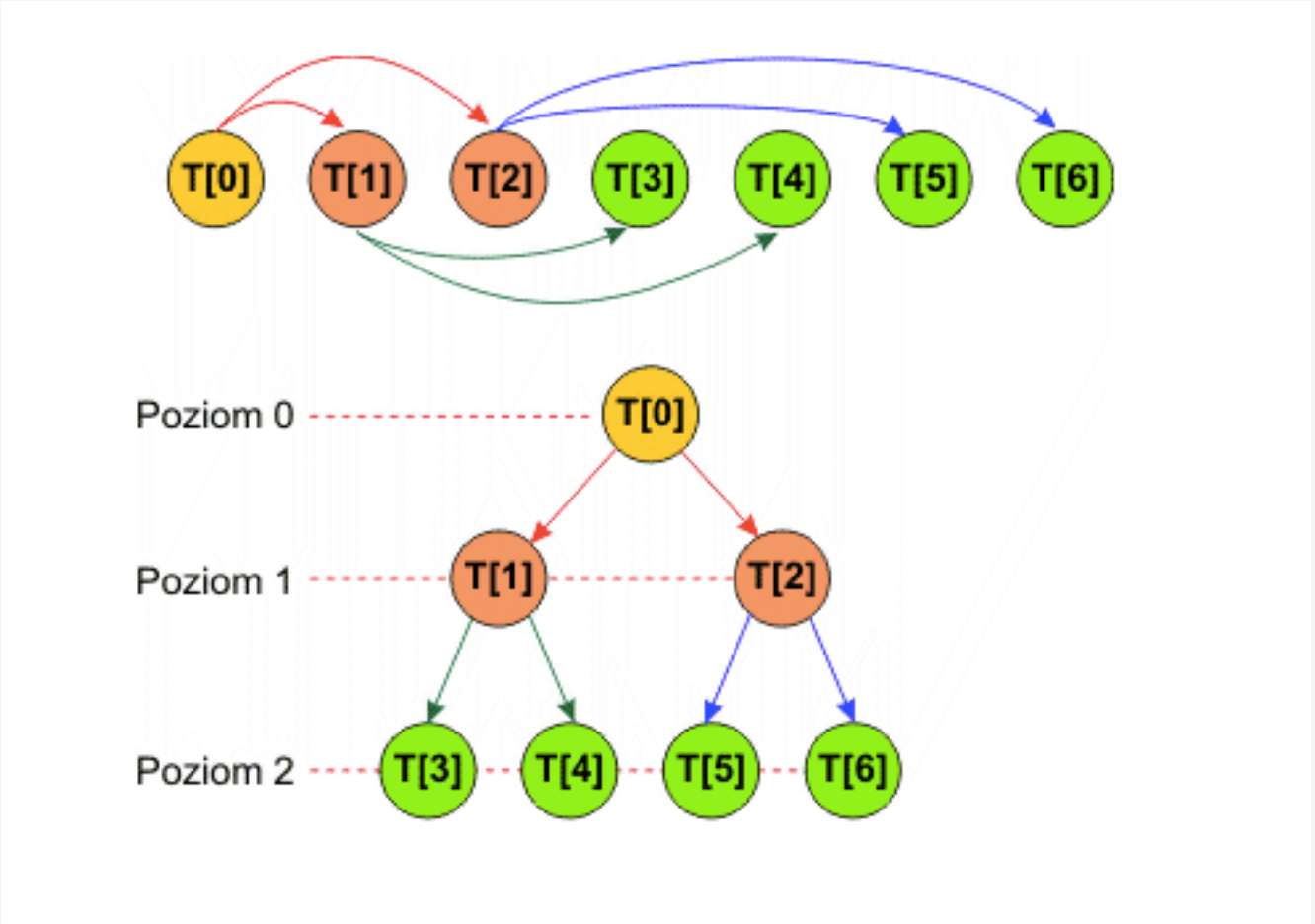
Kopiec

Kopiec (inaczej stóg) – jest drzewem binarnym, spełniającym warunek, że każdy następnik jest niewiększy od swojego poprzednika.

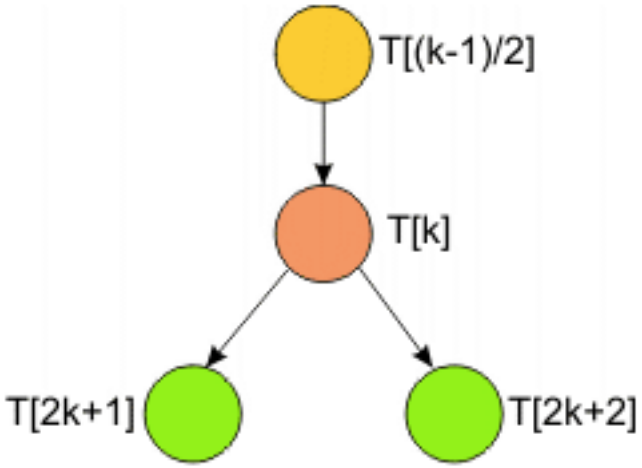
Własności:

- w korzeniu kopca znajduje się największy lub jeden z grupy największych o identycznej wartości
- na ścieżkach (połączeniach między węzłami), od korzenia do liścia, elementy są posortowane nierosnąco

Kopiec



Kopiec

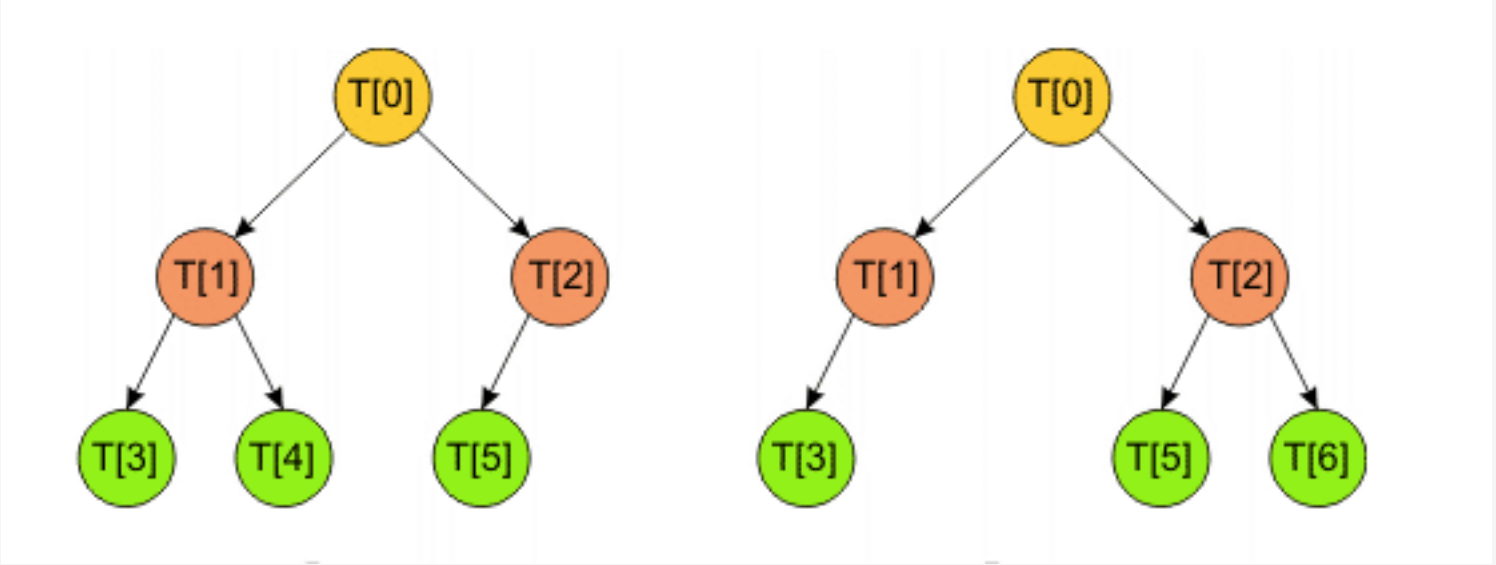


węzeł nadrzędny ma indeks równy (*k* - 1) / 2 (dzielenie całkowitoliczbowe)

k - węzeł bieżący

2*k*+1- lewy potomek 2*k*+2 - prawy potomek

Kopiec



Zupełny/niezupełny?

Kopiec

Tworzenie kopca dla zbioru {3 8 2 6 10 7 9 15 4 18}

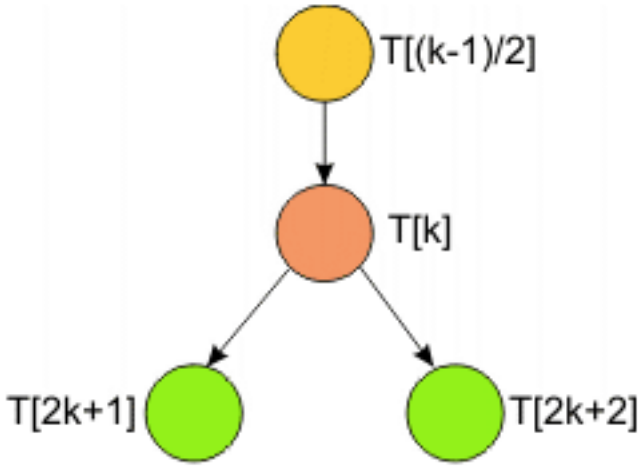
Sortowanie

Sortowanie przez kopcowanie (heap sort)

1. Tworzenie kopca
2. Sortowanie:
    - usunięcie wierzchołka kopca (elemen maksymalny/minimalny)
    - wstawienie w jego miejsce ostatniego elementu kopca
    - odtworzenie kopca

Wizualizacja:
https://ds055uzetaobb.cloudfront.net/image_optimizer/c1fc9fb4a40d810259ea8291ffde0eef182e7d57.gif

Sortowanie przez kopcowanie (heap sort)



węzeł nadrzędny ma indeks równy (*k* - 1) / 2 (dzielenie całkowitoliczbowe)

k - węzeł bieżący

2*k*+1- lewy potomek 2*k*+2 - prawy potomek

Rekurencja

Zadanie domowe

Zaimplementuj metodę iteracyjną, która odwróci napis (String.charAt())

tj. dla **„Ala ma kota!”** otrzymamy **„!atok am alA”**

Rekurencja

Zadanie domowe

Zrób poprzednie zadanie, ale rekurencyjnie