

# Pytania – egzamin inżynierski

Lidia J. Opuchlik

Polsko – Japońska Akademia Technk Komputerowych

*s16478@pjwstk.edu.pl*

2 lutego 2021

# Agenda

- 1 Pytanie nr 8
  - Twierdzenie Bayesa.
- 2 Pytanie nr 22
  - Najważniejsze algorytmy wyszukiwania i sortowania.
    - Wyszukiwanie
    - Sortowanie
- 3 Pytanie nr 34
  - Przetwarzanie strumieniowe (środki pakietu `java.util.stream`).

# Twierdzenie Bayesa

# Twierdzenie Bayesa

# Twierdzenie Bayesa

- 1 Thomas Bayes.
- 2 Mówi o **prawdopodobieństwie warunkowym**. Pozwala określić prawdopodobieństwo zajścia jakiegoś zdarzenia, o ile zaszło jakieś inne zdarzenie.

## Treść (dla dwóch zdarzeń)

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}, \quad \text{oraz } P(B) > 0 \text{ i } P(A) \geq 0$$

gdzie:

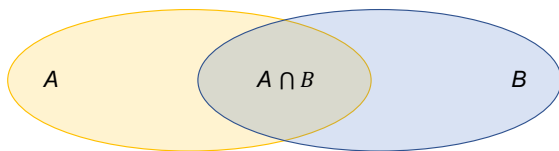
$P(A|B)$  – prawdopodobieństwo zajścia zdarzenia A, gdy zachodzi zdarzenie B (posterior probability),

$P(B|A)$  – prawdopodobieństwo zajścia zdarzenia B, gdy zachodzi zdarzenie A (likelihood),

$P(A)$  – prawdopodobieństwo zajścia zdarzenia A (prior probability),

$P(B)$  – prawdopodobieństwo zajścia zdarzenia B (evidence).

# Dowód (dla dwóch zdarzeń\*)



Rysunek: Iloczyn zbiorów A i B.

Z definicji:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \Leftrightarrow P(A \cap B) = P(A|B) \cdot P(B)$$

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \Leftrightarrow P(A \cap B) = P(B|A) \cdot P(A)$$

## Dowód c. d.

$$P(A \cap B) = P(A|B) \cdot P(B)$$

$$P(A \cap B) = P(B|A) \cdot P(A)$$

$$P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$$



$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

\*dowód dla większej liczby zdarzeń jest dużo bardziej skomplikowany

# W czym przydatne jest twierdzenie Bayesa?

# Trywialny przykład

## Treść zadania:

Obliczyć prawdopodobieństwo, że osoba jest chora na grypę, gdy występuje u niej gorączka, wiedząc, że: zdarzenie A – osoba jest chora na grypę, zdarzenie B – u osoby występuje gorączka. W zadaniu dane są następujące prawdopodobieństwa:  $P(B|A) = 0.7$ ,  $P(A) = 0.1$ ,  $P(B) = 0.2$ .

## Rozwiązanie:

Korzystając ze wzoru Bayesa obliczamy  $P(\text{grypa}|\text{goraczka})$ :

$$P(\text{grypa}|\text{goraczka}) = \frac{P(\text{goraczka}|\text{grypa}) \cdot P(\text{grypa})}{P(\text{goraczka})}$$

$$P(\text{grypa}|\text{goraczka}) = \frac{0.7 \cdot 0.1}{0.2} = \mathbf{0.35}$$



## Wyszukiwanie i sortowanie

# Najważniejsze algorytmy wyszukiwania i sortowania

# Wyszukiwanie i sortowanie

## Wyszukiwanie

- 1 liniowe
- 2 binarne
- 3 skok co k-ty element

## Sortowanie

- 1 przez wybieranie (selection sort)
- 2 bąbelkowe (bubble sort)
- 3 przez scalanie (merge sort)
- 4 szybkie (quick sort)
- 5 przez wstawianie (insertion sort)
- 6 zliczeniowe (counting sort)

# Wyszukiwanie

# Wyszukiwanie

# Wyszukiwanie – na czym polega?

Polega na odszukaniu elementu o danej wartości (klucza) w nieposortowanej lub posortowanej strukturze danych (tablicy lub liście) i zwrócenie indeksu, pod którym znaleziony element się znajduje\*. Jeżeli element nie zostanie znaleziony, wyświetlany jest stosowny komunikat (często wartość -1).

\*Jeżeli jest kilka takich samych elementów w ciągu, to zwracany jest indeks pierwszego napotkanego.

\*\* klucz i indeksy są liczbami całkowitymi

# Wyszukiwanie liniowe

## Na czym polega??

1. Wartości kolejnych elementów ciągu zaczynając od tego pod indeksem 0 a kończąc na  $n-1$  porównuje się z wartością klucza.
2. Zwraca się indeks znalezionej wartości, bądź  $-1$  w przypadku gdy nie został on znaleziony.

- struktura danych **nie musi** być posortowana
- pesymistyczna złożoność  $O(n)$

# Wyszukiwanie binarne

## Na czym polega??

1. W posortowanym ciągu porównuje się wartość klucza z wartością środkowego elementu. Jeżeli są takie same, zwracany jest indeks.
- 2a. Jeżeli klucz  $<$  środkowy element, to ogranicza się dalsze przeszukiwanie do lewego podciągu.
- 2b. Jeżeli klucz  $>$  środkowy element, to ogranicza się dalsze przeszukiwanie do prawego podciągu.
3. Powtarza się kroki od punktu 1.
4. Zwraca się indeks pasującego elementu lub -1, gdy nie został znaleziony.

- struktura danych **musi** być posortowana
- wykorzystywany jest algorytm "dziel i rządź"
- pesymistyczna złożoność  $O(\log n)$

# Wyszukiwanie poprzez skoki co k-ty element

## Na czym polega??

1. W posortowanym ciągu przeskakuje się co k elementów i porównuje się wartość bieżącego elementu z wartością klucza.
- 2a. Gdy element jest mniejszy niż klucz, przeskakuje się o następne k elementów.
- 2b. Gdy element jest większy niż wartość klucza, to należy sprawdzić już tylko k-1 ostatnio przeskoczonych elementów.
3. Zwraca się indeks pasującego elementu lub -1, gdy nie został znaleziony.

- struktura danych **musi** być posortowana
- algorytm jest k razy szybszy niż algorytm liniowy

# Sortowanie

# Sortowanie



# Sortowanie

Polega na ułożeniu elementów jakiejś nieuporządkowanej struktury danych w określonym porządku, np. rosnąco, malejąco, niemalejąco, ale też np. alfabetycznie.

Algorytmy sortowania w zależności od implementacji charakteryzują się różną wydajnością sortowania.

- ograniczę się do sortowania liczb całkowitych
- przedstawione opisy algorytmów będą dotyczyć sortowania niemalejącego (od najmniejszego do największego elementu)

# Sortowanie – kilka przydatnych pojęć

## Operacja domunująca

Operacja decydująca o złożoności algorytmu, np. porównanie dwóch elementów.

## Złożoność algorytmu

Mówi o tym ile operacji dominujących musi wykonać dany algorytm w najmniej korzystnym (pesymistyczna), przeciętnym (przeciętna) i najbardziej korzystnym (optymistyczna) przypadku wyjściowego ułożenia elementów w strukturze danych.

## Sortowanie – kilka przydatnych pojęć c.d.

### Stabilność

Mówi o tym, czy elementy o jednakowej wartości występują w niezmienionej kolejności w strukturze początkowej i po przeprowadzeniu algorytmu sortowania.

Stabilne: np. bąbelkowe, przez scalanie, posiada  $O(1)$  space complexity.

Niestabilne: np. szybkie.

### Algorytmy "in-place", "out-place"

Mówi o tym, czy do wykonania algorytmu sortowania niezbędne jest wprowadzenie dodatkowych struktur danych (list/tablic).

"In-place": np. szybkie, przez wstawianie.

"Out-place": np. przez scalanie.

# Sortowanie przez wybieranie

## Algorytm

- ❶ Ustawia się zmienną pomocniczą – np.  $i$  – na indeks 0.
  - ❷ Szuka się najmniejszego elementu listy.
  - ❸ Porównuje się wartość znalezionej minimum z wartością elementu znajdującego się pod bieżącym indeksem –  $i$ .
  - ❹ Jeżeli znaleziona wartość minimum jest mniejsza od bieżącego elementu to robi się zamianę wartości pod wskazanymi indeksami i inkrementuje się indeks o 1.
  - ❺ Jeżeli znaleziona wartość minimum jest większa od bieżącego elementu to inkrementuje się indeks o 1.
  - ❻ Powtarza się od 2-giego kroku aż cała lista będzie posortowana.
- algorytm ten dzieli wejściową strukturę danych na podstrukturę posortowaną (lewa) i do posortowania (prawa)

# Sortowanie przez wybieranie

## Sortowanie przez wybieranie

Złożoność czasowa pesymistyczna	$n^2$
Złożoność czasowa przeciętna	$n^2$
Złożoność czasowa optymistyczna	$n$
Stabilność	—
Złożoność pamięciowa	1

nr iteracji (wartość i)	tablica
0	[9,1,6,8,4,3,2,0]
1	[0,1,6,8,4,3,2,9]
2	[0,1,6,8,4,3,2,9]
3	[0,1,2,8,4,3,6,9]
4	[0,1,2,3,4,8,6,9]
5	[0,1,2,3,4,8,6,9]
6	[0,1,2,3,4,6,8,9]

**Rysunek:** Sortowanie przez wybieranie.

Legenda: czerwony – minimum, czarny pogrubiony – elementy do posortowania, niebieski – elementy na właściwym miejscu.

# Sortowanie bąbelkowe

## Algorytm

- ❶ Zaczynając od początku listy porównuje się wartości dwóch kolejnych elementów listy.
  - ❷ Jeżeli ich kolejność jest prawidłowa, to inkrementuje się indeks o 1.
  - ❸ Jeżeli ich kolejność jest nieprawidłowa, to zamienia się elementy miejscami i inkrementuje się indeks o 1.
  - ❹ Gdy dojdzie się do końca listy, to kroki powtarza się od początku do momentu aż cała lista będzie posortowana.
- listę przechodzi się tyle razy aż wszystkie elementy będą na swoich miejscach
  - algorytm ten jest zbyt wolny i niepraktyczny w użyciu

# Sortowanie bąbelkowe

## Sortowanie bąbelkowe

Złożoność czasowa pesymistyczna	$n^2$
Złożoność czasowa przeciętna	$n^2$
Złożoność czasowa optymistyczna	$n$
Stabilność	+
Złożoność pamięciowa	1

$[4, 2, 5, 1, 7] \rightarrow [2, 4, 5, 1, 7] \rightarrow [2, 4, 5, 1, 7] \rightarrow [2, 4, 1, 5, 7]$   
 $[2, 4, 1, 5, 7] \rightarrow [2, 4, 1, 5, 7] \rightarrow [2, 1, 4, 5, 7]$   
 $[2, 1, 4, 5, 7] \rightarrow [1, 2, 4, 5, 7]$   
 $[1, 2, 4, 5, 7]$

Rysunek: Sortowanie bąbelkowe.

# Sortowanie przez scalanie

Na czym polega??

Kroki

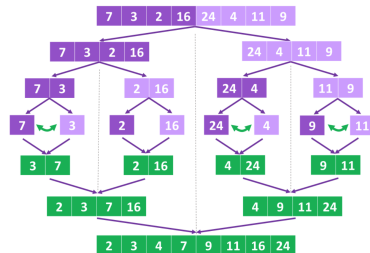
- Uwagi



# Sortowanie przez scalanie

## Sortowanie przez scalanie

Złożoność czasowa pesymistyczna	$n \cdot \log n$
Złożoność czasowa przeciętna	$n \cdot \log n$
Złożoność czasowa optymistyczna	$n \cdot \log n$
Stabilność	+
Złożoność pamięciowa	$n$



Rysunek: Sortowanie przez scalanie.

# Sortowanie szybkie

Na czym polega??

Kroki

- Uwagi

# Sortowanie szybkie

## Sortowanie szybkie

Złożoność czasowa pesymistyczna	$n^2$
Złożoność czasowa przeciętna	$n \cdot \log n$
Złożoność czasowa optymistyczna	$n \cdot \log n$
Stabilność	—
Złożoność pamięciowa	1

Tu nie będzie obrazka -  
tu będzie raczej opis jego  
zastosowań.

Animacja obrazująca  
quicksort:  
[Click](#)

# Sortowanie przez wstawianie

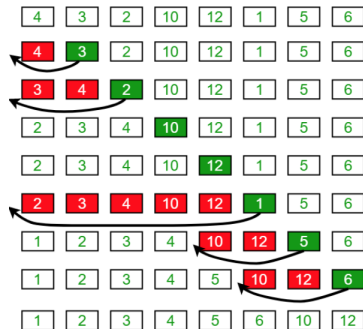
## Algorytm

- element od razu ma wylądować na swojej docelowej pozycji

# Sortowanie przez wstawianie

## Sortowanie przez wstawianie

Złożoność czasowa pesymistyczna	$n^2$
Złożoność czasowa przeciętna	$n^2$
Złożoność czasowa optymistyczna	$n$
Stabilność	+
Złożoność pamięciowa	1



Rysunek: Sortowanie przez wstawianie.

# Sortowanie zliczeniowe

Na czym polega??



# Sortowanie zliczeniowe

## Sortowanie zliczeniowe

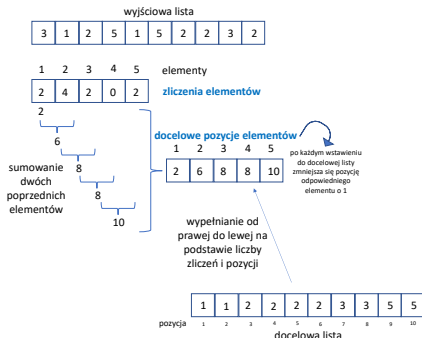
Złożoność czasowa pesymistyczna	$n + k$
---------------------------------	---------

Złożoność czasowa przeciętna	$n + k$
------------------------------	---------

Złożoność czasowa optymistyczna	$n + k$
---------------------------------	---------

Stabilność	+
------------	---

Złożoność pamięciowa	$n + k$
----------------------	---------



Rysunek: Sortowanie zliczeniowe.

# Porównanie wybranych algorytmów sortowania

**Tabela:** Porównanie parametrów wybranych algorytmów.

Rodzaj sort.	Złoż. czas. pes.	Złoż. czas. przec.	Złoż. czas. opt.	Stabilność	Złoż. pamięć.
p. wybieranie	$n^2$	$n^2$	$n$	—	1
bąbelkowe	$n^2$	$n^2$	$n$	+	1
p. scalanie	$n \cdot \log n$	$n \cdot \log n$	$n \cdot \log n$	+	$n$
szybkie	$n^2$	$n \cdot \log n$	$n \cdot \log n$	—	1
p. wstawianie	$n^2$	$n^2$	$n$	+	1
zliczeniowe	$n + k$	$n + k$	$n + k$	+	$n + k$

$n$  – liczba elementów struktury danych,  $k$  – liczba możliwych wartości

Użyta notacja to  $O()$  – służy do oznaczania rzędu wielkości (górnego ograniczenia tempa wzrostu) funkcji ▶



## Pakiet java.util.stream

# Pakiet java.util.stream

# Pakiet java.util.stream