Algorytmy sortowania

Bubble sort, merge sort, selection sort, bucket sort

18 czerwca 2021

Adrianna Saribekyan, Ewelina Kowal

The Faculty of Materials Engineering and Physics
Cracow University of Technology

Algorytmy sortowania

Sortowanie danych polega na uporządkowaniu danych, w taki sposób, aby zniwelować problem ich wyszukiwania, tak aby były one czytelnie przedstawione odbiorcy. Zadaniem algorytmów sortujących jest przedstawienie ciągu liczb n w sposób rosnący.

Przykłady sortowania

Przykłady omawiane na zajęciach:

- Sortowanie bąbelkowe
- Sortowanie przez wstawianie
- Sortowanie przez scalanie
- Szybkie sortowanie
- Sortowanie przez kopcowanie

Dodatkowe przykłady:

- Sortowanie kubełkowe
- Sortowanie przez wybór
- Sortowanie przez zliczanie
- Sortowanie Shella

Przykłady sortowania

Bubble sort

Sortowanie bąbelkowe polega na porównaniu dwóch kolejnych wyrazów danego ciągu i ułożenie ich w odpowiedniej kolejności.

```python
def sort_babelkowe(tab, n):  # tab=liczby
    j = 1
    while n > 1:
        for x in range(0, n-1):
            if tab[x] > tab[x+1]:
                tab[x], tab[x+1] = tab[x+1], tab[x]
                print("\nKrok ", j, " : Zamiana miejscami liczby ", tab[x], " z liczbą ", tab[x+1])
                j += 1
        n -= 1
```

https://commons.wikimedia.org/wiki/File:Bubble-sort.gif

Bubble sort

Zalety	Wady
krótki kod	czas akcji rośnie ze wzrostem liczby elementów
łatwy w zrozumieniu	ilość wykonywanych operacji jest niezależna od permutacji elementów
prosty do napisania	najwolniejszy typ sortowania

Przykłady sortowania

Merge sort

Algorytm sortowania przez scalanie polega na rekuryncyjnym dzieleniu tablicy wejściowej na podtablice, aż do momentu uzyskania jednoelementowych tablic. W ostatniej fazie sortowania, wartości elementów znajdujących się w podtablicy są porównywane, a następnie łączone według ustalonej zasady.

```
def sort_scal(tab, l, r):
    if r > l:
        m = int((l + r) / 2)
        sort_scal(tab, l, m)
        sort_scal(tab, m + 1, r)
        scal(tab, l, m, r)
```

https://pl.wikipedia.org/wiki/Plik:Merge-sort-example-300px.gif

Merge sort

Zalety

- Podtablice można sortować niezależnie od siebie
- Stabilny algorytm
- Prosty w implementacji

Wady

- Potrzebny jest dodatkowy obszar pamięci przechowujący kopie podtablic

Przykłady sortowania

Selection sort

Algorytm selection sort zaczyna pracę od wyszukania elementu minimalnego zbioru. Następnie element minimalny zostaje umiejscowiony na początku tablicy. W dalszym etapie działania, algorytm w taki sam sposób sprawdza pozostałe minimalne elementy i ustawia je na kolejnych miejscach tablicy wyjściowej.

```
def sort_wyb(tab, n):  # n=ilość elementów, tab=lista/tablica
    for x in range(n-1):  # x=aktualny element
        minimum = x
        for j in range(x+1, n):
            if tab[j] < tab[minimum]:
                minimum = j
        if x != minimum:
            print("\nKrok ", x+1, ": wstawianie liczby ", tab[minimum], " na pozycje ", x)
            pom = tab[x]
            tab[x] = tab[minimum]
            tab[minimum] = pom
```

https://algorithms.tutorialhorizon.com/

Selection sort

Zalety

- Dobra wydajnośc dla wstępnie posortowanych danych.
- Prosty w implementacji
- Niskie zapotrzebowanie pamięciowe.

Wady

- Niska efetktyność w przypadku tablic z dużą ilością danych.
- Niestabilny podczas porównywania równych elementów

Przykłady sortowania

Bucket sort

Sortowanie kubełkowe polega na utworzeniu kubełków w których zliczane są poszczególne elementy tablicy wejściowej. Ilość kubełków jest uzależniona od róznicy wartości największego i najmniejszego elementu zbioru. W końcowej fazie nowa tablica jest tworzona z elementów znajdujących się w kubełkach po posortowaniu.

`https://binarnie.pl/wp-content/uploads/2019/11/jubelkowe_calkowite.gif`

Bucket sort

```python
def sort_kub(tab, n):
    minmax = minmmax(tab)
    ymin = minmax[0]  # wyznaczamy min i max z tablicy
    ymax = minmax[1]
    buckets = [0] * (ymax - ymin + 1)  # nastepnie tworzymy tablice przechowujaca kubelki
    print("Stworzono kubelki:")
    for x in range(ymax - ymin + 1):
        print("B", (x + ymin), end=" | ")
    print()
    for x in range(n):
        buckets[tab[x] - ymin] += 1  # zliczamy nastepnie ilosc wystapien dla kazdego z kubelkow
    print("Zliczono elementy:")
    for x in range(ymax - ymin + 1):
        print("B", (x + ymin), "=", buckets[x], end=" | ")
    print()
    lastindex = 0
    for x in range(ymax - ymin + 1):  # na koniec sortujemy wedlug kubelkow tablice
        y = lastindex
        while y < buckets[x] + lastindex:
            tab[y] = x + ymin
            y += 1
        lastindex = y
```

Rysunek: Fragment kodu bucket sort.

Bucket sort

Zalety	Wady
sprawny i szybki czas akcji	dane powinny być rownmiernie rozłożone
sortowanie odbywa się w miejscu	konieczność wskazania elementu minimalnego i maxymalnego

n	Insertion Sort t[s]	Selestion Sort t[s]	Bubble Sort t[s]	Heap Sort t[s]	Quick Sort* t[s]	Merge Sort t[s]	Counting Sort t[s]
20000	0,46	0,78	2,29	0,01	0,00	0,01	0,00
40000	1,87	3,14	9,13	0,01	0,01	0,01	0,00
60000	4,20	7,04	20,52	0,02	0,02	0,01	0,00
70000	5,73	9,57	27,86	0,02	0,02	0,02	0,00
80000	7,57	12,51	36,46	0,03	0,02	0,02	0,00
90000	9,45	15,83	46,17	0,04	0,02	0,03	0,01
100000	11,69	19,54	56,93	0,04	0,02	0,03	0,01
120000	16,78	28,16	82,01	0,05	0,02	0,04	0,01
150000	26,17	44,15	128,35	0,06	0,03	0,04	0,01
200000	47,03	78,68	226,07	0,07	0,04	0,06	0,01
300000	104,60	179,53	536,91	0,14	0,08	0,09	0,02
400000	195,40	328,67	955,88	0,19	0,10	0,12	0,04
500000	304,88	514,14	1492,83	0,25	0,13	0,16	0,05
750000	691,20	1163,54	3356,55	0,38	0,21	0,26	0,09
1000000	1246,62	2062,72	5829,55	0,53	0,28	0,33	0,09

*- metoda Quick Sort z podziałem wg środkowego elementu

Zakres liczb [1, n].

Rysunek: Porównanie niektórych algorytmów sortowania w zależności od czasu operacji.

Bibliografia

Na co należy zwóricć uwagę przy wybieraniu algorytmu :

- zastosowanie
- złożoność czasowa
- złożonośc pamięciowa
- stabilność
- inne powody (m.in.: łatwość implementacji, wydajność w zależności od rozmiarów tablic)

Pórównanie algorytmów

Algorytm sortowania	Bubble	Merge	Select	Bucket
czasowa	$O(n^2)$	O(nlogn)	$O(n^2)$	$O(n^2)$
pamięciowa	O(1)	O(n)	O(1)	O(1)

Największą wagę podczas porównywania metod sortowania ma dla nas zależność czasowa oraz pamięciowa algorytmów.

Bibliografia

- https://binarnie.pl/
- https://eduinf.waw.pl/inf/alg/003_sort/0020.php
- https://prezi.com/
- https://en.wikipedia.org/
- https://www.geeksforgeeks.org/
- http://www.cs.put.poznan.pl/arybarczyk/TeoriaAiZO1.pdf
- https://algorithms.tutorialhorizon.com/selection-sort-java-implementation/selection-sort-gif/

Dziękujemy za uwagę