Porównanie algorytmów sortujących

Programowanie dla fizyków

Matejko Marek, Mazur Krzysztof, Paszkot Dawid

Fizyka Techniczna
Politechnika Krakowska

Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Plan Prezentacji

Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Spis treści

Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Cel pracy

Zaimplementowanie wybranych algorytmów sortujących i porównanie ich pod względem:

- czasu wykonania
- łatwości w implementacji

Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Sortowanie bąbelkowe

Sortowanie bąbelkowe jest jednym z najprostszych w implementacji algorytmów porządkujących dane. Nazwa wzięła się stąd, że dane podczas sortowania - tak jak bąbelki w napoju gazowanym - przemieszczają się ku prawej stronie i układają się w odpowiednim szyku.

Wstęp
Implementacja w Pythonie
Implementacja w C++
Zestawienie wyników
Bibliografia

Sortowanie szybkie

Jeden z szybszych algorytmów.

Wybieramy element z sortowanego zbioru, tzw. Pivot.
Następnie elementy nie większe ustawiamy na lewo,
a nie mniejsze na prawo.
Tak powstają dwie części tablicy, gdzie w pierwszej części
znajdują się elementy nie większe od drugiej.
Kontynuujemy sortowanie w taki sam sposób dla
utworzonych tablic.

Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Sortowanie kubełkowe

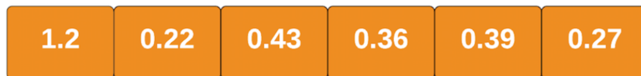
[ang. bucket sort, bin sort] Pomysł pojawił się po raz pierwszy w 1956 roku. Został wymyślony przez E. J. Issac'a i R. C. Singleton'a

Sposób działania sortowania kubełkowego można podzielić na następujące kroki:

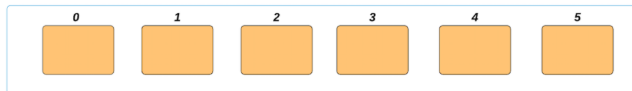
1. należy podzielić znany przedział danych na kubełki
2. umieścić wartość danych do kubełków o odpowiednich indeksach
3. posortować wartości w niepustych kubełkach
4. wyciągnąć posortowane już wartości

Wstęp
Implementacja w Pythonie
Implementacja w C++
Zestawienie wyników
Bibliografia

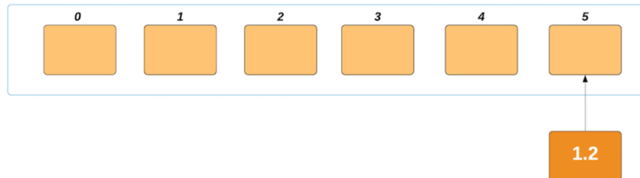
Algorytm kubełkowy



1.2/6=0.2



1.2/0.2=6



Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

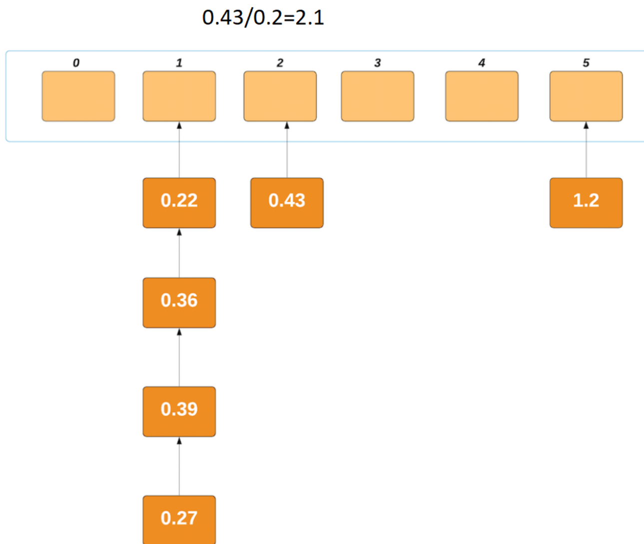
Algorytm kubełkowy

0.22/0.2=1.1

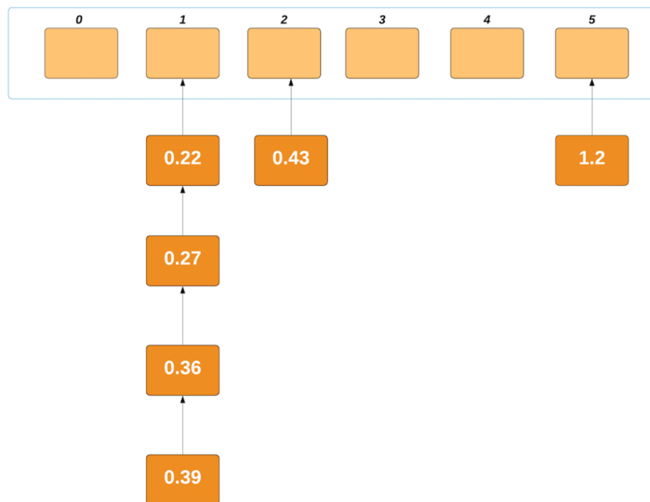
0 1 2 3 4 5

0.22 1.2

Algorytm kubełkowy



Algorytm kubełkowy



Spis treści

Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Część kodu generująca losowe liczby

```
1   import random
2   import time
3
4   list = []
5   for i in range(101):
6       dane = random.randint(100000,999999)
7       list.append(dane)
8   print("Liczba elementów wynosi",i)
9
10  ####################################################################
11  def insertion_sort(bucket):
12      for i in range (1, len (bucket)):
13          var = bucket[i]
14          j = i - 1
15          while (j >= 0 and var < bucket[j]):
16              bucket[j + 1] = bucket[j]
17              j = j - 1
18          bucket[j + 1] = var
```

Funkcja sortująca algorytmem kubełkowym

```
20  def bucket_sort(input_list):
21      start_time = time.time()
22      max_value = max(input_list)
23      size = max_value / len(input_list)
24  
25      buckets_list = []
26      for x in range(len(input_list)):
27          buckets_list.append([])
28  
29      for i in range(len(input_list)):
30          j = int(input_list[i] / size)
31          if j != len(input_list):
32              buckets_list[j].append(input_list[i])
33          else:
34              buckets_list[len(input_list) - 1].append(input_list[i])
35  
36      for z in range(len(input_list)):
37          insertion_sort(buckets_list[z])
38  
39      final_output = []
40      for x in range(len(input_list)):
41          final_output = final_output + buckets_list[x]
42      elapsed_time = time.time() - start_time
43      print("Sortowanie algorytmem kubełkowym", elapsed_time)
44      return final_output
45  ########################################################################
```

Funkcja sortująca algorytmem szybkim

```python

#####################################################################


def partition(array, begin, end):
    pivot = begin
    for i in range(begin+1, end+1):
        if array[i] <= array[begin]:
            pivot += 1
            array[i], array[pivot] = array[pivot], array[i]
    array[pivot], array[begin] = array[begin], array[pivot]
    return pivot


def _quicksort(array, begin, end):
    if begin >= end:
        return
    pivot = partition(array, begin, end)
    _quicksort(array, begin, pivot-1)
    _quicksort(array, pivot+1, end)


def quicksort(array, begin=0, end=None):
    if end is None:
        end = len(array) - 1
    return _quicksort(array, begin, end)

#####################################################################

```


Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Funkcja bąbelkowa i wypisanie wyników

```python
########################################################################


def bubble_sort(list):
    start_time = time.time()
    n = len(list)
    while n > 1:
        switch = False
        for l in range(0, n - 1):
            if list[l] > list[l + 1]:
                list[l], list[l + 1] = list[l + 1], list[l]
                switch = True
        n -= 1
        if switch == False: break
    elapsed_time = time.time() - start_time
    print("Sortowanie algorytmem bąbelkowym",elapsed_time)
    return list
########################################################################


list_1 = bucket_sort(list)

list_2 = bubble_sort(list)


start_time = time.time()
quicksort(list)
elapsed_time = time.time() - start_time
print("Sortowanie algorytmem quicksort",elapsed_time)
```


Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Spis treści

Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

```cpp
#include <iostream>
#include <chrono>
#include <vector>
#include <algorithm>

using namespace std;
```

Rysunek: Bibioteki użyte w programie

- iostream - odpowiada za obiekty kontrolujące odczyt i zapis w standardowym strumieniu,
- chrono - odpowioda za m.in. za klasy mierzenie czasu,
- vector - wektor przechowuje elementy danego typu w rozmieszczeniu liniowym i umożliwia szybki dostęp losowy do dowolnego elementu,
- algorithm - definiuje zbiór funkcji specjalnie zaprojektowanych do użycia na zakresach elementów.

Funkcja sortująca algorytmem bąblekowym

```
// sortowanie bąbelkowe
void sortowanie_babelkowe(int tablica[], int rozmiar)
{
    for (int i=1; i<rozmiar; i++)
        for (int j=rozmiar-1; j>=i; j--)
            if (tablica[j]<tablica[i])
            {
                int tmp=tablica[j-1];
                tablica[j-1]=tablica[j];
                tablica[j] = tmp;
            }
}
// ***********************************
```

Funkcja sortująca algorytmem kubełkowym

```
// sortowanie kubelkowe
void sortowanie_kubelkowe(float tablica[], int rozmiar)
{
    int k = rozmiar/1000;
    vector<float> kub[k];

    for(int i=0; i<rozmiar; i++)
    {
        int nr = k* tablica[i];
        kub[nr].push_back(tablica[i]);
    }

    for (int i=0; i<k; i++)
    {
        sort(kub[i].begin(), kub[i].end());
    }
}
// ***********************************
```

Funkcja sortująca algorytmem szybkim

```
// sortowanie szybkie
void sortowanie_szybkie(int *tab, int lewy, int prawy)
{
    if(prawy <= lewy) return;

    int i = lewy - 1, j = prawy + 1,
    srodek = tab[(lewy+prawy)/2];

    while(true)
    {
        while(srodek>tab[++i]);
        while(srodek<tab[--j]);
        if( i <= j)
            swap(tab[i],tab[j]);
        else
            break;
    }
    if(j > lewy)
    sortowanie_szybkie(tab, lewy, j);
    if(i < prawy)
    sortowanie_szybkie(tab, i, prawy);
}
// ************************************
```

Część kodu generująca losowe liczby

```
// generowanie losowych liczb
int n = 10000; // liczba elementów tablicy
int tablica_losowych_liczb[n];
int tablica_A[n];
int tablica_B[n];
float tablica_C[n];

for (int i=0; i<n; i++)
{
    tablica_losowych_liczb[i] = rand() % 99999 + 10000;
}

for (int j=0; j>n; j++)
{
    tablica_A[j] = tablica_losowych_liczb[j];
    tablica_B[j] = tablica_losowych_liczb[j];
    tablica_C[j] = (float)tablica_losowych_liczb[j];
}
// *************************************
```

Część kodu licząca czas wykonania

```
// mierzenie czasu sorotwania bobelkowego
auto bubble_czas_roz= chrono::high_resolution_clock::now();
sortowanie_babelkowe (tablica_A, n); // sortowanie bąbelkowe
auto bubble_czas_zak = chrono::high_resolution_clock::now();

auto bubble_czas_wyk = bubble_czas_zak - bubble_czas_roz; // sprawdzic zmienne
cout << "### SORTOWANIE BABELKOWE ###\n";
cout << "Ilosc elementow: " << n << "\n";
cout << "Czas wykonania: " << bubble_czas_wyk/chrono::milliseconds(1)<< " ms\n" << endl;
// ***********************************

// mierzenie czasu sorotwania kubelkowe
auto bucket_czas_roz = chrono::high_resolution_clock::now();
sortowanie_kubelkowe (tablica_C, n); // sortowanie szybkie
auto bucket_czas_zak = chrono::high_resolution_clock::now();

auto bucket_czas_wyk = bucket_czas_zak - bucket_czas_roz; // sprawdzic zmienne
cout << "### SORTOWANIE KUBEŁKOWE ###\n";
cout << "Ilosc elementow: " << n << "\n";
cout << "Czas wykonania: " << bucket_czas_wyk/chrono::milliseconds(1)<< " ms\n" << endl;
// ***********************************

// mierzenie czasu sorotwania szybkiego
auto quick_czas_roz = chrono::high_resolution_clock::now();
sortowanie_szybkie (tablica_B, 0, n); // sortowanie szybkie
auto quick_czas_zak = chrono::high_resolution_clock::now();

auto quick_czas_wyk = quick_czas_zak - quick_czas_roz; // sprawdzic zmienne
cout << "### SORTOWANIE SZYBKIE ###\n";
cout << "Ilosc elementow: " << n << "\n";
cout << "Czas wykonania: " << quick_czas_wyk/chrono::milliseconds(1)<< " ms\n" << endl;
// ***********************************
```

Spis treści

Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Zestawienie wyników

Wyniki dla 1000 elementów odpowiednio w C++ i Pythonie

```
### SORTOWANIE BABELKOWE ###
Ilosc elementow: 1000
Czas wykonania: 1 ms

### SORTOWANIE KUBEŁKOWE ###
Ilosc elementow: 1000
Czas wykonania: 0 ms

### SORTOWANIE SZYBKIE ###
Ilosc elementow: 1000
Czas wykonania: 0 ms
```

```
Liczba elementów wynosi 1000
Sortowanie algorytmem kubełkowym 0.001994609832763672
Sortowanie algorytmem bąbelkowym 0.12627649307250977
Sortowanie algorytmem quicksort 0.03901171684265137


Process finished with exit code 0
```

Zestawienie wyników

Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Wyniki dla 10 000 elementów

```
### SORTOWANIE BABELKOWE ###
Ilosc elementow: 10000
Czas wykonania: 101 ms

### SORTOWANIE KUBEŁKOWE ###
Ilosc elementow: 10000
Czas wykonania: 1 ms

### SORTOWANIE SZYBKIE ###
Ilosc elementow: 10000
Czas wykonania: 1 ms
```

```
Liczba elementów wynosi 10000
Sortowanie algorytmem kubełkowym 0.10270357131958008
Sortowanie algorytmem bąbelkowym 13.17356562614441
Sortowanie algorytmem quicksort 0.03789877891540527

Process finished with exit code 0
```

Zestawienie wyników

Wyniki dla 15 000 elementów

```
### SORTOWANIE BABELKOWE ###
Ilosc elementow: 15000
Czas wykonania: 228 ms

### SORTOWANIE KUBEŁKOWE ###
Ilosc elementow: 15000
Czas wykonania: 3 ms

### SORTOWANIE SZYBKIE ###
Ilosc elementow: 15000
Czas wykonania: 0 ms
```

```
Liczba elementów wynosi 15000
Sortowanie algorytmem kubełkowym 0.25032925605773926
Sortowanie algorytmem bąbelkowym 29.788197994232178
Sortowanie algorytmem quicksort 0.039893150329589844

Process finished with exit code 0
```


Wstęp
Implementacja w Pythonie
Implementacja w C++
Zestawienie wyników
Bibliografia

Spis treści

Wstęp

Implementacja w Pythonie

Implementacja w C++

Zestawienie wyników

Bibliografia

Bibliografia

[1] https://pl.wikipedia.org/wiki/Python

[2] https://pl.wikipedia.org/wiki/Sortowanie kubełkowe

[3] https://analityk.edu.pl/sortowanie-babelkowe/

[4] https://binarnie.pl/sortowanie-kubelkowe/

[5] https://stackabuse.com/bucket-sort-in-python