WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ
POLITECHNIKI RZESZOWSKIEJ

Kacper Kloc

Zadanie projektowe nr 1 - Algorytmy i struktury danych

Rzeszów, 2020

Spis treści

1. Wstęp

Treść zadania: Dla zadanej tablicy liczb całkowitych przesuń wszystkie elementy mniejsze od 0 na jej koniec (należy zachować koleność występowania).

Zadanie wymaga od nas posortowania tablicy w taki sposób, aby przenieść wszystkie liczby ujemne na koniec tej tablicy, zachowując ich kolejność występowania. Moim sposobem na rozwiązanie tego problemu jest podzielenie zadanej tablicy na dwie tablice pomocnicze (jedna z nich przechowa wartości nieujemne, a druga ujemne) i późniejsze połączenie powstałych tablic pomocniczych w jedną tablicę, do której najpierw wpisane zostaną elementy nieujemne.

Działając tak, jak powyżej zostało to opisane, otrzymujemy szybko działający algorytm nawet dla bardzo dużych tablic.

W obrębie programu dla wygody i ułatwienia operacje na tablicach przeprowadzane są na vector'ach - strukturach danych reprezentujących tablicę liczb całkowitych, wczytanych z biblioteki <vector>.

2. Opis działania programu

W programie znajduje się pięć funkcji. Pierwsza z nich służy do załadowania danych z pliku tekstowego (o nazwie „dane.txt”) do vector'a –Drugą z funkcji jest funkcja przesuwająca elementy po tablicy. Zapisuje ona odpowiednio elementy większe od zera lub równe zeru do tablicy pomocniczej tab1 oraz elementy mniejsze od zera do tablicy pomocniczej tab2, zachowując ich pierwotną kolejność. W następnym kroku tablice te są łączone w jedną – najpierw przypisujemy wartości z tab1 do naszej końcowej tablicy, a później na następnych indeksach zapisujemy wartości z tablicy tab2. Na koniec wykonuje się zapisanie posortowanej tablicy do pliku tekstowego(„zapis liczb.txt”) .

Funkcja trzecia również służy do przesuwania elementów w tablicy, ale jest wykorzystywana przy testach algorytmu. Różni się od swojej pierwotnej zapisem danych do innego pliku(„posortowane testy.txt”).

Funkcja czwarta odpowiada za przeprowadzenie testów algorytmu sortującego. Generuje kolejno tablice złożone z 100, 1000, 10000, 100000, 1000000 liczb z zakresu od -100 do 100 i mierzy czas, potrzebny do posortowania takich tablic. Tablice przed posortowaniem zapisują sięw pliku o nazwie testy.txt. Z tej funkcji usunięty został szósty pomiar, którego użyłem w dalszej części programu, ponieważ powodował on znaczne zwiększenie zajmowanego miejsca przez tworzone pliki tekstowe.

Ostatnią z funkcji jest funkcja main, w której wykonuje się cały program.

3. Pseudokod, schemat blokowy, złożoność obliczeniowa, wykresy

3.1. Pseudokod

Start

Otwórz plik tekstowy

Jeżeli udało się załadować plik to

Uzupełnij tablicę arr danymi z pliku
Przypisz do zmiennej arrLength długość tablicy arr
Wyświetl komunikat o poprawnym wczytaniu tablicy oraz jej zawartość
Utwórz zmienne i, j=0, k=0
Utwórz tablice dynamiczne tab1, tab2
dla i=0; i<arrLength;i++

Jeśli arr[i]>=0 to

tab1[j]=arr[i]
j++

W przeciwnym razie

tab2[k]=arr[i]
k++

dla i=0;i<j;i++

arr[i]=tab1[i]

dla i=0;i<k;i++

arr[i+j]=tab2[i]

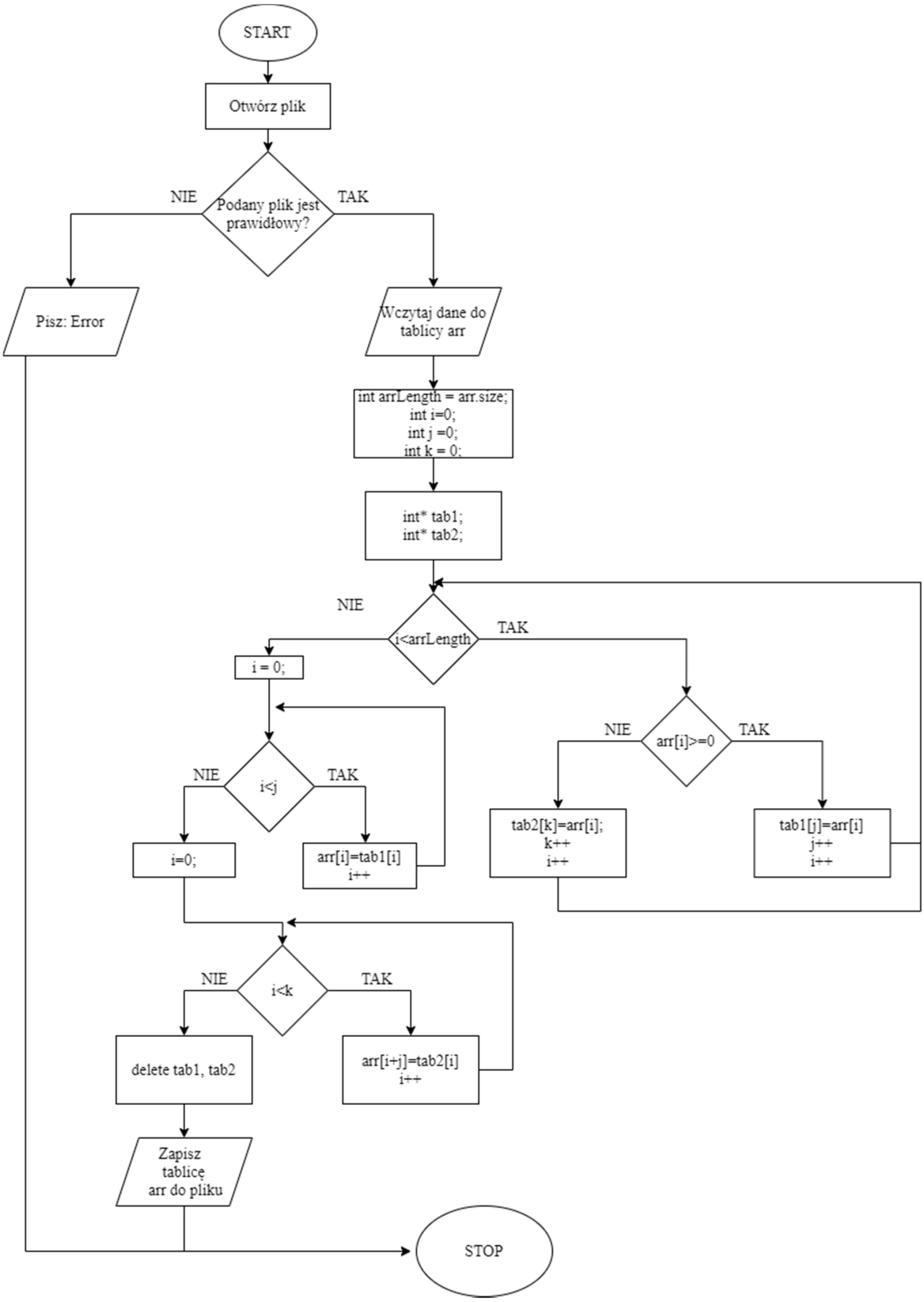
Usuń tab1, tab2
Zapisz tablicę arr do pliku

W przeciwnym razie

Wypisz: „Error”

Koniec

3.2. Schemat blokowy.



Rysunek 1 Schemat blokowy algorytmu

3.3. Złożoność obliczeniowa, wykresy

Weźmiemy pod lupę kolejno pętle z algorytmu przesuwającego elementy ujemne na koniec tablicy.

Pierwszą pętlą for w kodzie jest pętla wykonująca się n razy, w której zawarta jest instrukcja warunkowa if. Instrukcja ta nie wpływa na złożoność tej konkretnej pętli – wykona się ona zawsze n razy.

```
// Funkcja sortujaca
void sortuj(std::vector<int>arr, int arrLength)
{
    // Tworzenie pomocniczych zmiennych oraz dynamicznych tablic
    int j = 0, k = 0;
    int* tabl= new int[arrLength];
    int* tab2= new int[arrLength];
    for(int i=0; i<arrLength; i++)
    {
        // Umieszczenie wszystkich elementów tablicy wiekszych lub równych 0 w pierwszej pomocniczej tablicy
        if(arr[i]>=0)
        {
            tabl[j]=arr[i];
            j++;
        }
        // Umieszczenie wszystkich elementów mniejszych od 0 w drugiej pomocniczej tablicy
        else
        {
            tab2[k]=arr[i];
            k++;
        }
    }
```

Rysunek 2 Pętla for z if'em

Następnie w kodzie znajdują się 2 pętle, które również wykonają się n razy. Na naszą całkowitą złożoność nie ma to wpływu (przecież O(n+n) = O(2n) = O(n))

```
    // Połaczenie obu pomocniczych tablic w jedna, posortowana tablice
    for(int i=0; i<j; i++)
        arr[i]=tabl[i];
    for(int i=0; i<k; i++)
        arr[i+j]=tab2[i];
```

Rysunek 3 Pętle for o złożoności O(n)

Na samym końcu kodu znajduje się jeszcze jedna pętla, zapisująca tablicę do pliku, która również wykona się n razy.

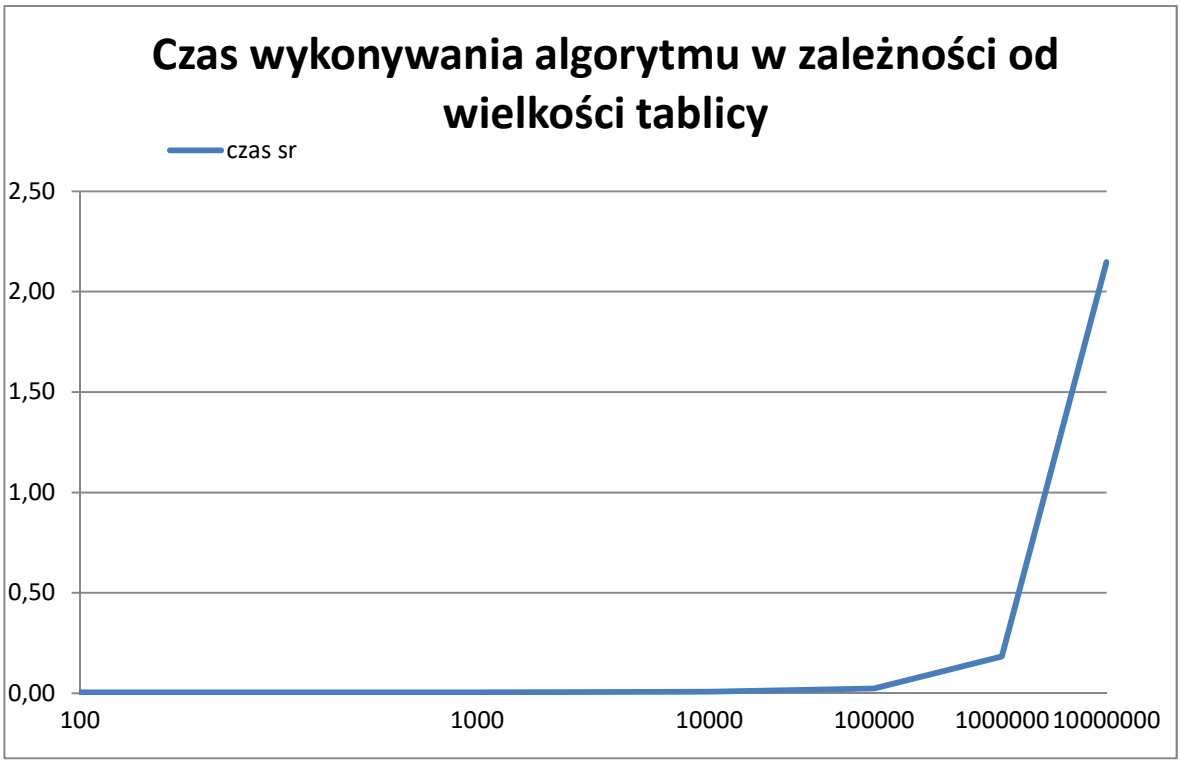
```
    // Zapis wyniku do pliku wynik.txt
    fstream plik;
    plik.open("zapis liczb.txt",ios::out|ios::app);
    for(int i=0; i<arrLength; i++)
    {
        plik<<arr[i]<<" ";

    }
    plik<<endl;
    plik.close();
```

Rysunek 4 Ostatnia pętla for z algorytmu

Sumarycznie daje nam to złożoność algorytmu O(n).

Poniższy wykres przedstawia średni pomiar wykonywania się algorytmu sortującego na losowo wygenerowanych ciągach liczb o długości: 100, 1000, 10000, 100000, 1000000 i 10000000. Czas ten jest średnią arytmetyczną 5 prób pomiarowych.



Rysunek 5 Wykres zależności czasu od N

Z wykresu jasno widać, że program szybko sobie radzi nawet z tablicami zawierającymi 1000000 elementów(średni czas wykonania programu to ~0.25s.). Dłuższe działanie występuje dopiero kiedy wykraczamy poza tę granicę – dla 10000000 elementów czas ten sięga niemalże 2.5s.

4. Podsumowanie i wnioski końcowe

Zadanie, pomimo początkowych trudności okazało się nietrudne. Wymaga jednak pomysłu na wykonanie zadanego przesunięcia. Przez długi czas starałem się je wykonać w całkowicie inny, a zarazem jak się okazało niemożliwy sposób, niż to finalnie zrobiłem.

Do tej pory nie miałem dużo do czynienia z językiem C++. Zrobienie tego projektu nauczyło mnie wielu użytecznych rzeczy – od używania nowych bibliotek, przez używanie różnych źródeł do czerpania wiedzy, jak np. StackOverflow, aż po zmienione myślenie, co do samego programowania. Staram się obecnie nieco inaczej spoglądać na stawiane przede mną problemy.

5. Kod programu

```
#include <iostream>
#include <fstream>
#include <iterator>
#include <iomanip>
#include <limits.h>
#include <cstdlib>
#include <time.h>
#include <vector>
#include <chrono>
using namespace std::chrono;
using namespace std;

// Funkcja wczytująca dane z pliku
std::vector<int> wczytanie()
{
   std::vector<int> arr;
   ifstream plik("dane.txt");       // Wczytywany plik

   // Sprawdzenie czy plik istnieje i otworzenie go
   if (plik.good())
   {
      // Wczytanie danych do tablicy
      int x = 0;
      while (plik >> x)
      {
         arr.push_back(x);
      }
      // Zamknięcie pliku.
      plik.close();
   }
   else
   {
      cout << "Error!";
      _exit(0);
   }
   return arr;
}
// Funkcja sortująca
void sortuj(std::vector<int>arr, int arrLength)
{
   // Tworzenie pomocniczych zmiennych oraz dynamicznych tablic
   int j = 0, k = 0;
   int* tab1= new int[arrLength];
   int* tab2= new int[arrLength];
   for(int i=0; i<arrLength; i++)
   {
      // Umieszczenie wszystkich elementów tablicy większych lub równych 0 w pierwszej
pomocniczej tablicy
      if(arr[i]>=0)
      {
         tab1[j]=arr[i];
```

```
            j++;
        }
        // Umieszczenie wszystkich elementów mniejszych od 0 w drugiej pomocniczej tablicy
        else
        {
            tab2[k]=arr[i];
            k++;
        }
    }
    // Połączenie obu pomocniczych tablic w jedną, posortowaną tablicę
    for(int i=0; i<j; i++)
        arr[i]=tab1[i];
    for(int i=0; i<k; i++)
        arr[i+j]=tab2[i];
    // Usunięcie tablic pomocniczych
    delete[] tab1;
    delete[] tab2;
    // Zapis wyniku do pliku wynik.txt
    fstream plik;
    plik.open("zapis liczb.txt",ios::out|ios::app);
    for(int i=0; i<arrLength; i++)
    {
        plik<<arr[i]<<" ";

    }
    plik<<endl;
    plik.close();
}
void sortuj_test(std::vector<int>tab, int arrLength)
{
    // Tworzenie pomocniczych zmiennych oraz dynamicznych tablic
    int j = 0, k = 0;
    int* tab1= new int[arrLength];
    int* tab2= new int[arrLength];
    for(int i=0; i<arrLength; i++)
    {
        // Umieszczenie wszystkich elementów tablicy większych lub równych 0 w pierwszej
pomocniczej tablicy
        if(tab[i]>=0)
        {
            tab1[j]=tab[i];
            j++;
        }
        // Umieszczenie wszystkich elementów mniejszych od 0 w drugiej pomocniczej tablicy
        else
        {
            tab2[k]=tab[i];
            k++;
        }
    }
    // Połączenie obu pomocniczych tablic w jedną, posortowaną tablicę
    for(int i=0; i<j; i++)
        tab[i]=tab1[i];
    for(int i=0; i<k; i++)
```

```
        tab[i+j]=tab2[i];
    // Usunięcie tablic pomocniczych
    delete[] tab1;
    delete[] tab2;
    // Zapis wyniku do pliku posortowane testy.txt
    fstream plik;
    plik.open("posortowane testy.txt",ios::out|ios::app);
    for (int i = 0; i < arrLength; i++)
    {
        plik<<tab[i]<<" ";
    }
    plik<<endl;
    plik.close();

}
void testy()
{
    // Tworzenie zmiennych
    int N = 100;
    std::vector<int> tab;
    int liczba_testow = 5;

    // Pętla generujaca liczba_testów tablic o wielkosci N*10
    for (int i=0; i<liczba_testow; i++)
    {
        srand((unsigned)time(NULL));

        // Zapisanie wygenerowanej tablicy do pliku
        fstream plik;
        plik.open("testy.txt",ios::out|ios::app);
        for (int i =0; i < N; i++)
        {
            int b = rand() % 200 - 100;
            tab.push_back(b);

            plik<<tab[i]<<" ";
        }
        plik<<endl;
        plik.close();

        /* testy algorytmu */
        auto start = high_resolution_clock::now();
        sortuj_test(tab, N);

        auto stop = high_resolution_clock::now();
        std::chrono::duration<double> czas = stop-start;
        std::cout << std::setw(9) << czas.count() << " s."<< endl;
        // zwiekszenie rozmiarow tablicy
        N = N * 10 ;
    }
}


int main()
```

```
{
   std::vector<int>arr;
   arr=wczytanie();
   // Przypisanie długosci vector'a arr do zmiennej
   int arrLength = arr.size();
   cout<<"Wczytano "<<arrLength<<" liczb do tablicy: ";
   for(int i=0; i<arrLength; i++)
      cout<<arr[i]<<" ";



   /* czas wykonania obliczen algorytmem */
   auto start = high_resolution_clock::now();

   sortuj(arr, arrLength);

   auto stop = high_resolution_clock::now();
   std::chrono::duration<double> czas = stop-start;
   std::cout << "Czas obliczen: " <<
         std::setw(9) << czas.count() << " s."<< endl;
   testy();
}
```