Algorytmy i struktury danych projekt nr 1

Wykonał:
Michał Jaskuła

1. Treść zadania.

Dla ciągu (w postaci tablicy) zawierającego wyłącznie wartości 0 lub 1, znajdź podciąg zawierający równą liczbę zer i jedynek którego długość jest największa.

Wejście: 0, 0, 1, 0, 1, 0, 0

Wyjście: Najdłuższym podciągiem są: 0, 1, 0, 1 oraz 1, 0, 1, 0

2. Analiza zadania

Z ciągu złożonego wyłącznie z wartości 0 lub 1 znajdujemy najdłuższy podciąg zawierający równą liczbę zer i jedynek.

Szczegóły implementacji problemu.

Zadanie zostało podzielone na następujące etapy:

1. Utworzenie dynamiczną tablicę *tab, rozmiar tablicy podaje użytkownik.
2. Tablica zostaje wypełniona pseudolosowymi liczbami z zakresu od 0 do 1.
3. Wyszukanie najdłuższego podciągu zawierającego równą liczbę zer i jedynek.
4. Zapis podciągu do pliku tekstowego.

*2.1 Utworzenie dynamiczną tablicę *tab, rozmiar tablicy podaje użytkownik.*

Na samym początku tworzymy dynamiczną tablicę *tab, której zadaniem będzie przechowanie pseudolosowego ciągu złożonego z wartości 0 oraz 1. Rozmiar tablicy podaje z klawiatury użytkownik i jest on przechowywany w zmiennej ile_elem.

2.2 Tablica zostaje wypełniona pseudolosowymi liczbami z zakresu od 0 do 1.

Następnie losujemy podaną przez użytkownika liczbę elementów pseudolosowych ze zbioru liczb naturalnych, z zakresu od 0 do 1. W tym celu posłużymy się funkcją srand().

2.3 Wyszukanie najdłuższego podciągu zawierającego równą liczbę zer i jedynek.

W celu znalezienia najdłuższego podciągu zawierającego równą liczbę zer i jedynek potraktujemy wszystkie 0 jako -1. Następnie będziemy sumować ze sobą kolejne wyrazy ciągu, jeżeli suma wyrazów ciągu od pierwszego elementu (lub ostatniego elementu, jeśli przeszukujemy naszą tablicę od końca) do i-tego elementu jest równa 0 to właśnie jest nasz szukany podciąg.

2.4 Zapis podciągu do pliku tekstowego.

Ostatnim etapem naszego zadania jest zapisanie otrzymanego podciągu do pliku tekstowego.
W pliku tekstowym znajdzie się również informacja odnośnie liczby elementów otrzymanego podciągu, uzyskana za pomocą funkcji size().

3. Znaczenie zmiennych użytych w kodzie:

- *tab – tablica przechowująca wylosowany przez funkcję srand ciąg
- roz – parametr funkcji, przechowujący rozmiar tablicy
- ile_elem – rozmiar tablicy *tab, podawany przez użytkownika
- podciag – wektor przechowujący najdłuższy poszukiwany podciąg w przypadku gdy przeszukujemy tablicę od początku do końca
- podciag2 - wektor przechowujący najdłuższy poszukiwany podciąg w przypadku gdy przeszukujemy tablicę od końca do początku
- indeks, indeks2 – zmienne przechowujące numery indeksów dla których poszukiwane przez nas podciągi będą miały swój koniec
- x, y – zmienne pomocnicze służące do wyszukiwania największego indeksu
- suma, suma2 – zmienne służące do obliczania sumy kolejnych elementów ciągu

4. Działanie programu

Program został napisany w środowisku programistycznym Code::Blocks IDE w języku C++.

Tworzymy dynamiczną tablicę *tab, o rozmiarze podanym przez użytkownika.

```
int ile_elem,  *tab;

cout<<"Podaj rozmiar tablicy"<<endl;
cin>>ile_elem;

tab = new int [ile_elem];
```

Wypełniamy tablicę pseudolosowym ciągiem za pomocą utworzonej przez nas funkcji. Do funkcji przekazujemy parametry: tablicę oraz rozmiar.

```
void generujciag(int *tab, int roz)
{
    srand(time(NULL));
    for(int i = 0; i < roz; i++)
    {
        tab[i] = rand()%2;
    }
}
```

W programie zostanie utworzony za pomocą funkcji generujciag ciąg, ale ponieważ dla dużej tablicy wyświetlanie ciągu trwa dosyć długo, wyświetlanie wylosowanego ciągu jest ujęte w komentarzu. Wyświetlanie ciągu program realizuje za pomocą następujących instrukcji:

```
for(int a = 0; a < ile_elem; a++)
{
    cout<<tab[a]<<", ";
}
```

Następnie tworzymy wektory do których będziemy zapisywać wyszukane podciągi.

```
vector<int> podciag;
vector<int> podciag2;
```

Kolejnym krokiem będzie utworzenie funkcji wykonującej algorytm oraz przekazanie jej niezbędnych parametrów. Nasza funkcja będzie nosić nazwę: wynik.

```
void wynik(vector<int> &podciag, vector<int> &podciag2, int *tab, int roz)
```

W pierwszej kolejności tworzymy zmienne niezbędne do działania, ich znaczenie opisane jest w punkcie trzecim oraz zamieniamy wszystkie 0 na -1.

```
int x, y, indeks = 0, indeks2 = 0, suma = 0, suma2 = 0;
for(int i = 0; i < roz; i++)
{
 if(tab[i]==0)
 {
    tab[i]--;
 }
```

Później obliczamy sumę kolejnych tablicy. Tutaj należy pamiętać, aby zapamiętać numery indeksu elementu dla którego suma elementów jest równa 0. W zadaniu mamy znaleźć najdłuższy ciąg, więc szukamy największego numeru indeksu. W tym celu posłużymy się pomocniczą zmienną x.

```
suma = suma + tab[i];
if(suma == 0)
{
    x = i;
    if(i>indeks)
    {
        indeks = i;
    }
}
```

Po zakończeniu pętli for, zerujemy sumę, aby dla kolejnego ciągu program działał poprawnie (od sumy równej 0 a nie od sumy uzyskanej dla poprzedniego ciągu).

```
suma = 0;
```

Te same czynności powtarzamy przechodząc przez tablicę od jej ostatniego elementu do elementu pierwszego. Należy pamiętać, że jeżeli mamy tablicę zawierającą 100 elementów to jest ona indeksowana od 0 do 99. W tym przypadku od 99 do 0. Iterujemy więc od elementu o jeden mniejszego niż rozmiar do elementu zerowego.

```
for(int j = roz -1 ; j >= 0; j--)
{
    if(tab[j]==0)
    {
        tab[j]--;
    }
    suma2 = suma2 + tab[j];
    if(suma2==0)
    {
        y = j;
        if(j>indeks2)
        {
            indeks2 = y;
        }
    }
}
suma2 = 0;
```

Wpisujemy podciąg od elementu pierwszego do elementu indeks do wektora.

```
for(int k = 0; k <= indeks; k++)
    {
        if(tab[k]==-1)
        {
            tab[k]++;
        }
        podciag.push_back(tab[k]);
    }
```

Tą samą czynność wykonujemy dla drugiego podciągu otrzymanego w wyniku iterowania od końca do początku tablicy.

```
for(int k = roz - 1; k >= indeks2; k--)
    {
        if(tab[k]==-1)
        {
            tab[k]++;
        }
        podciag2.push_back(tab[k]);
    }
```

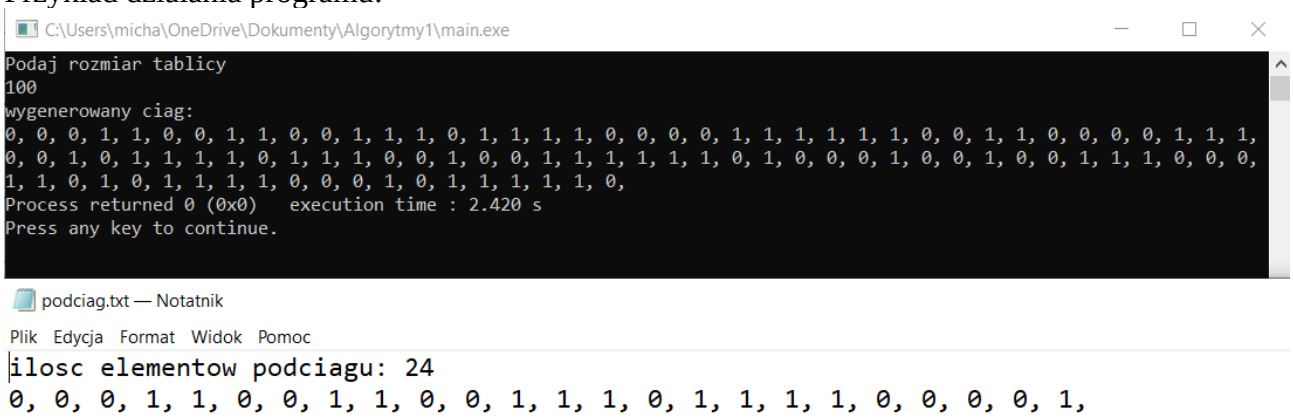
Aby zapisać otrzymany wynik do pliku tekstowego użyjemy utworzonej przez nas funkcji zapis. Parametrami funkcji będą wektory: podciag oraz podciag2.

```
void zapis(vector<int> &podciag, vector<int> &podciag2)
```

Oczywiście interesuje nas ten najdłuższy podciąg zatem za pomocą funkcji size porównujemy oba wektory. Ten „większy” (dłuższy podciąg) zostanie wpisany do pliku tekstowego „podciag.txt”.

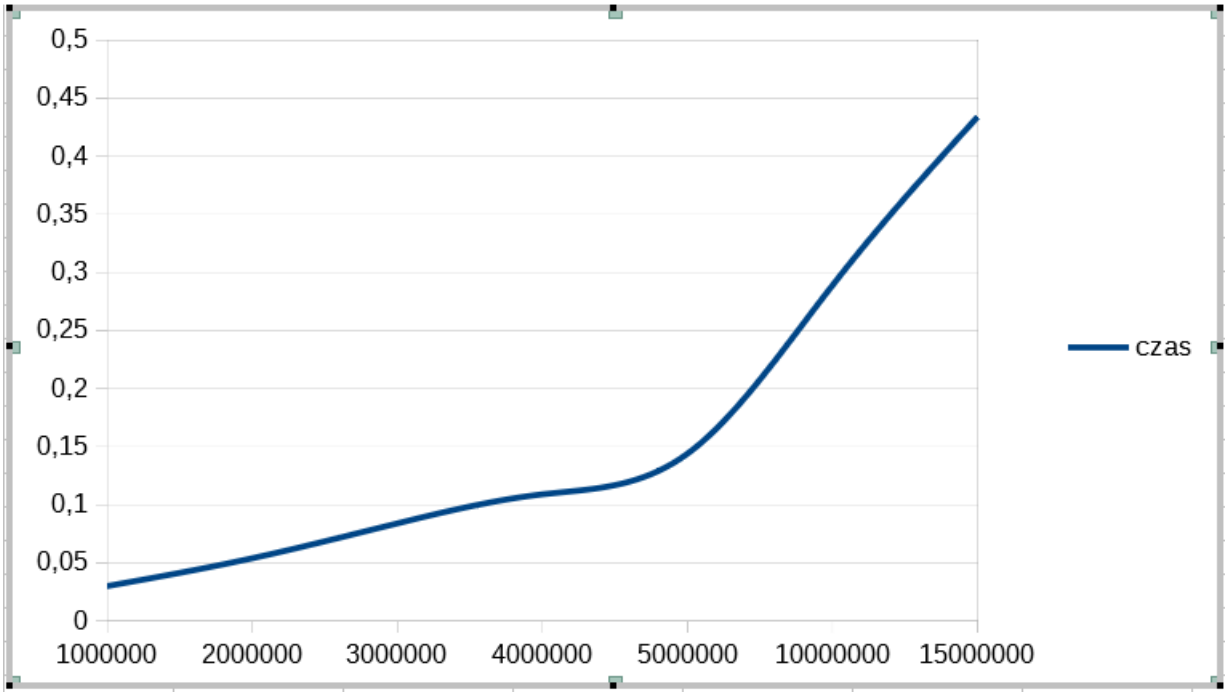
```
void zapis(vector<int> &podciag, vector<int> &podciag2)
{
    ofstream zapis("podciag.txt");
    if(zapis)
    {
        if(podciag.size() > podciag2.size())
        {
            zapis<< "ilosc elementow podciagu: "<<podciag.size()<<endl;
            for(int i = 0; i < podciag.size(); i++)
            {
                zapis << podciag[i] << ", ";
            }
        }
        else if(podciag2.size() > podciag.size())
        {
            zapis<< "ilosc elementow podciagu: "<<podciag2.size()<<endl;
            for(int i = 0; i < podciag2.size(); i++)
            {
                zapis << podciag2[i] << ", ";
            }
        }
    }
}
```

Przykład działania programu:

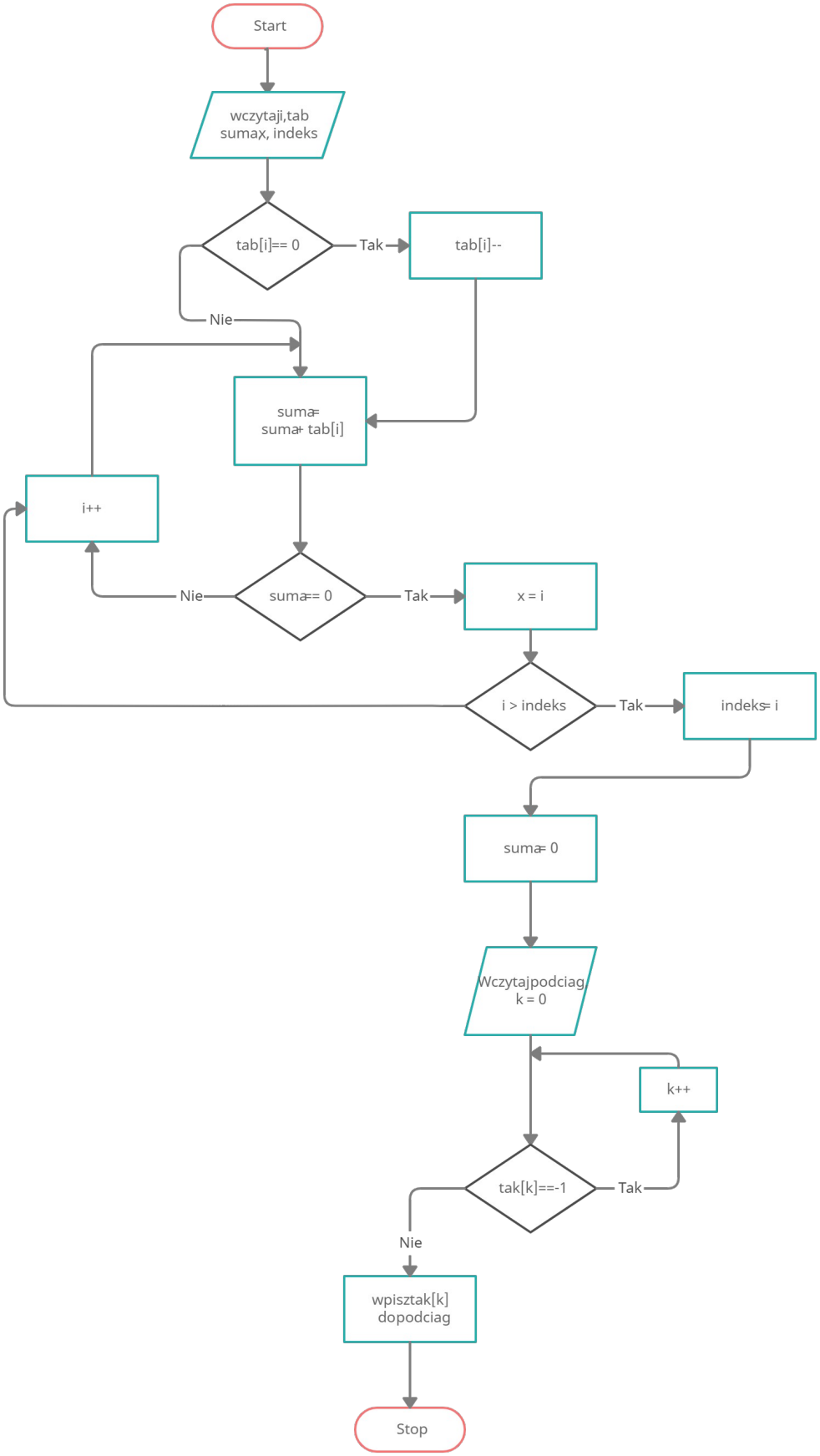


5. *Złożoność czasowa*

W przedstawionym rozwiązaniu złożoność czasowa wynosi: O(4n). Przeprowadzamy testy dla następującej listy elementów: 1mln, 2mln, 3mln, 4mln, 5mln, 10mln, 15mln.



6. Schemat blokowy algorytmu


Start
wczytaji,tab
sumax, indeks
tab[i]== 0
Tak
tab[i]--
Nie
suma=
suma+ tab[i]
i++
Nie
suma== 0
Tak
x = i
i > indeks
Tak
indeks= i
suma= 0
Wczytajpodciag
k = 0
k++
tak[k]==-1
Tak
Nie
wpisztak[k]
dopodciag
Stop

7. Lista kroków.

K01: Start

K02: Wczytaj tab, suma, x, indeks

K03: Jeśli tab[i]==0
tab[i]--
K04: W przeciwnym razie idź do K05

K05: Licz sumę

K06: Jeżeli suma=0
x=i

K07: W przeciwnym razie: i++

K08: Jeżeli i > indeks
indeks = i

K10: W przeciwnym razie: idź do K07

K11: Suma = 0

K12: Wczytaj podciag, k=0

K13: Jeżeli tab[k]==-1
tab[k]++

K14: W przeciwnym razie:
Wpisz tab[k] do podciag

K15: Koniec

8. Kod programu:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <iomanip>
#include <numeric>
#include <chrono>
#include <limits.h>
#include <vector>

using namespace std;
using namespace std::chrono;
```

```
high_resolution_clock::time_point stop;
high_resolution_clock::time_point start;
chrono::duration<double> czas;

void generujciag(int *tab, int roz)
{
   srand(time(NULL));
   for(int i = 0; i < roz; i++)
   {
     tab[i] = rand()%2;
   }
}
void wynik(vector<int> &podciag, vector<int> &podciag2, int *tab, int roz)
{
   int x, y, indeks = 0, indeks2 = 0, suma = 0, suma2 = 0;
   for(int i = 0; i < roz; i++)
   {
    if(tab[i]==0)
    {
      tab[i]--;
    }
    suma = suma + tab[i];
    if(suma == 0)
    {
      x = i;
      if(i>indeks)
      {
         indeks = i;
      }
    }
   }
   suma = 0;

   for(int j = roz -1 ; j >= 0; j--)
   {
      if(tab[j]==0)
      {
         tab[j]--;
      }
      suma2 = suma2 + tab[j];
      if(suma2==0)
      {
         y = j;
         if(j>indeks2)
         {
            indeks2 = y;
         }
      }
   }
   suma2 = 0;
```

```
for(int k = 0; k <= indeks; k++)
        {
           if(tab[k]==-1)
           {
              tab[k]++;
           }
           podciag.push_back(tab[k]);
        }
    for(int k = roz - 1; k >= indeks2; k--)
        {
           if(tab[k]==-1)
           {
              tab[k]++;
           }
           podciag2.push_back(tab[k]);
        }
}

void zapis(vector<int> &podciag, vector<int> &podciag2)
{
    ofstream zapis("podciag.txt");
    if(zapis)
    {
       if(podciag.size() > podciag2.size())
       {
          zapis<< "ilosc elementow podciagu: "<<podciag.size()<<endl;
          for(int i = 0; i < podciag.size(); i++)
          {
             zapis << podciag[i] << ", ";
          }
       }
       else if(podciag2.size() > podciag.size())
       {
          zapis<< "ilosc elementow podciagu: "<<podciag2.size()<<endl;
          for(int i = 0; i < podciag2.size(); i++)
          {
             zapis << podciag2[i] << ", ";
          }
       }
    }
}


int main()
{
    vector<int> podciag;
    vector<int> podciag2;

    int ile_elem,  *tab;
```

```
cout<<"Podaj rozmiar tablicy"<<endl;
    cin>>ile_elem;


    tab = new int [ile_elem];

    //cout<<"wygenerowany ciag: "<<endl;
    generujciag(tab, ile_elem);

    /*for(int a = 0; a < ile_elem; a++)
    {
        cout<<tab[a]<<", ";
    } */
    start = high_resolution_clock::now();
    wynik(podciag, podciag2, tab, ile_elem);
    stop = high_resolution_clock::now();

    czas = stop-start;

    cout << endl << "Czas wykonania algorytmu: "<<setprecision(5)<<czas.count()<<"s\n";

    zapis(podciag, podciag2);

    delete [] tab;

   return 0;
}
```