P O L I T E C H N I K A R Z E S Z O W S K A
im. Ignacego Łukasiewicza
WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

JAKUB BARAN

Projekt C++

Zadanie 2
kierunek studiów: Inżynieria i Analiza Danych

Opiekun pracy:

Mariusz Borkowski

Rzeszów 2022

Opis Problemu

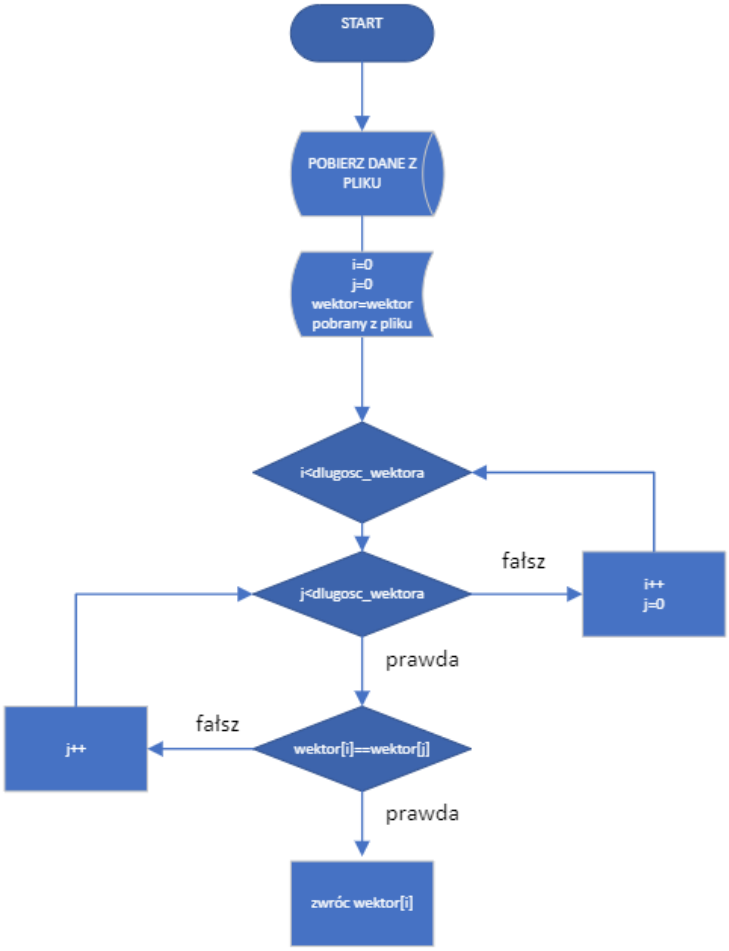
Należy napisać program szukający powtarzającego się elementu dla zadanej tablicy liczb całkowitych o rozmiarze n zawierającej wartości z przedziału [1, n-1]

Przykład.
Wejście: 1, 2, 3, 4, 4
Wyjście powtarzający się element to 4
Wejście: 1, 2, 3, 4, 2
Wyjście powtarzający się element to 2

Opis szczegółów implementacji problemu

Program powinien pobierać dane z pliku dla tego lepiej zastosować wektory bo są dynamiczne – nie trzeba deklarować ilości zmiennych (nie wiemy ile użytkownik będzie chciał wpisać danych). Program Główny powinien zawierać pętle szukające powtarzającego się elementu wykonujące się tyle razy ile wynosi długość wektora z danymi. Po znalezieniu właściwego elementu program powinien zwrócić ten element do głównej funkcji abyśmy mogli go zapisać do pliku. Warto by było zmierzyc czas algorytmu. Należy również zrobić dwie funkcję - pierwszą do wczytywania danych z pliku do wektora a drugą zapisującą wyniki z programu do pliku

Schemat Blokowy Algorytmu



Pseudokod

(cały program)

```
funkcja powtarzajacy_element(wektor) {
        dla (i = 1 do  rozmiar wektora) wykonuj
        dopóki (j <= rozmiar wektora) wykonuj
        jeżeli (i==j)  - przerwij
        jeżeli (wektor[i]==wektor[j])
        zwróć wektor[i]
}
funkcja pobranie_danych (nazwa){
        dane <- z pliku (nazwa)
        jeżeli (dane nie sa otwarte)
                wypisz  blad otwarcia pliku
        dopóki(dane sa wczytywane ) wykonuj
                dane >> tmp
                wektor.dodaj (tmp)
        dane.zamknij()
```

```
        zwróc dane
        }
funkcja zapisanie_danych(nazwa, powtarzajacy_element, czas){
        Wypisz do pliku  “Powtarzajacy element to: “, powtarzajacy_element
        Wypisz do pliku  “Czas wykonywania programu:  “,, czas “sekundy”
        }
funkcja główna
{
start = start_mierzenia_czasu()
dane = pobranie_danych(“nazwa_pliku_z_danymi”)
        powtarzajacy_element(dane)
        koniec =  koniec_mierzenia_czasu()
        czas = koniec-dane
        zapisanie_danych(“nazwa_pliku_do_zapisu”, powtarzajacy_element(dane), czas)
        zwróc 0
 )
```

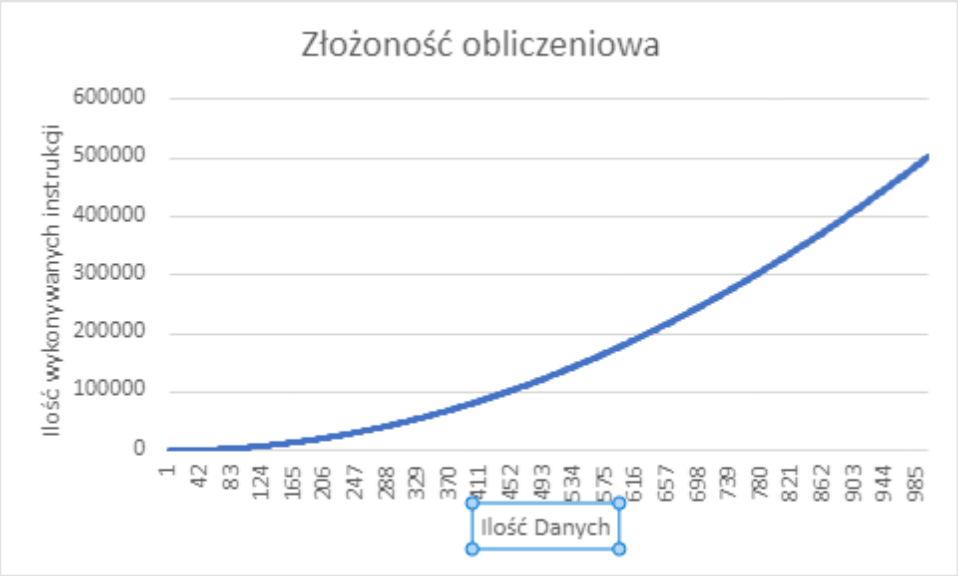
(główny algorytm)

```
funkcja powtarzajacy_element(wektor) {
        dla (i = 1 do  rozmiar wektora) wykonuj
        dopóki (j <= rozmiar wektora) wykonuj
        jeżeli (i==j)  - przerwij
        jeżeli (wektor[i]==wektor[j])
        zwróć wektor[i]
}
```

Złożoność Obliczeniowa

Po analizy złożoności funkcji wyszła złożoność kwadratowa, ponieważ algorytm zawiera funkcję for w funkcji – oznacza to, że instrukcje wykonują się kwadratowo co do ilości danych

Wykres złożoności:



Wnioski

Główny algorytm w złożoności kwadratowej działa poprawnie. Niestety nie dałem rady mniejszej złożonośći próbowałem z sortowaniem bąbelkowym w ten sposób ale to też jest złożoność kwadratowa i oprócz funkcji sortujących potrzebna jest dodatkowa funkcja z znalezieniem powtarzającego się elementu. Kod tego rozwiązania:

```
vector<int> wektor = { 1, 2, 3, 3, 4 };
for (int i = 0; i < wektor.size() - 1; i++)
	{
	for (int j = 0; j < wektor.size() - 1; j++)
		{
			if (wektor[j] > wektor[j + 1])
				{
				swap(wektor, wektor[j + 1]);
				}
		}
}

for(int i=0; i<wektor.dlugosc+1; i++)
if wektor[i] == wektor [i+1]{ return wektor[i] }
```

Program również liczy czas trwania algorytmu i podaje go w sekundach
Zrobiłem też funkcję wczytującą dane z pliku do wektora i drugą zapisujące dane do pliku z wynikiem głównej funkcji - powtarzającym się elemencie i z czasem trwania algorytmu. Program jest kompletny nie mam pomysłu jak go można było zrobić lepiej

Program

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <chrono>
using namespace std;
//tworzenie funkcji szukającej powtarzającego się elementu
int pow(vector<int> tablica) {
for (int i = 0; i <= tablica.size(); i++) { //funkcje szukające powtarzającego się elemetu
for (int j = 0; j <= tablica.size(); j++) {
if (i == j) { break; }
if (tablica[i] == tablica[j]) { //jesli znalezlismy powtarzajacy element to go zwracamy
return tablica[i];

}
}
}
}
vector<int> pobranie_danych(string nazwa) {// funkcja pobierajaca dane z pliku do wektora
ifstream indata(nazwa); //otwarcie pliku
if (!indata.is_open()) {
cout << "Blad otwarcia pliku " << nazwa <<endl; // w razie bledu otwarcia pliki
return {};
}
int tmp;
vector<int> data;
while (!indata.fail() && !indata.eof()) // do konca danych
{
indata >> tmp;
data.push_back(tmp);
}
indata.close(); //zamkniecie pliku
return data;
}
void zapisanie_danych(string nazwa, int pow, double czas) { // funkcja zapisujaca dane z wektora do
pliku
ofstream do_pliku(nazwa);
do_pliku<< "Powtarzajacy element to " << pow << endl; // zapisanie elementow do pliku
do_pliku <<"Czas wykonywania programu: " << czas << " sekundy";
}

int main()
{
auto start = chrono::high_resolution_clock::now(); // start pomiaru czasu

vector<int> dane = pobranie_danych("dane.txt");
```

```
cout << "Powtarzajacy element to " << pow(dane) << endl;
auto koniec = chrono::high_resolution_clock::now(); //koniec pomiaru czasu
auto miara_czasu = chrono::duration_cast<chrono::nanoseconds>(koniec - start);
double czas = miara_czasu.count() * 1e-9;
zapisanie_danych("zapis.txt", pow(dane), czas);

return 0;
}
```