Projekt #1 – Algorytmy i struktury danych.

Temat projektu: Dla zadanego ciągu zer, jedynek i dwójek, znajdź wszystkie podciągi „symetryczne względem dwójek” występujących w ciągu.
Autor: Stach Kacper
Numer albumu: 173217

1 SPIS TREŚCI

2 SPIS RYSUNKÓW

3 Temat

Dla zadanego ciągu zer, jedynek i dwójek znajdź wszystkie podciągi „symetryczne względem dwójek” występujących w ciągu.

3.1 CECHY PROGRAMU:

A. Program zawiera funkcję która zostaje wywołana wewnątrz programu
B. W głównym programie powinno zostać wykonane kilka testów sprawdzających działanie funkcji
C. Program powinien mieć możliwość odczytywania danych wejściowych i zapisu podciągów jako wyników do plików tekstowych
D. Kod powinien zawierać komentarze dotyczące działania programu

4 PROJEKTOWANIE

4.1 PROBLEMATYKA

W zadaniu należy znaleźć i pokazać wszystkie podciągi symetryczne względem liczby „2” znajdującej się w tablicy. Aby to zrobić należy utworzyć tablicę, którą następnie wypełnimy losowymi liczbami w zakresie od 0 do 2. Następnie funkcja znajduje każdą liczbę 2, po czym sprawdza i porównuje każdy indeks „na lewo i prawo” w tabeli. Gdy odnajdzie liczby równe sobie w równej odległości od liczby 2, wprowadza je do tablicy wyniki.

4.2 METODYKA

4.2.1 Funkcja algorytm()

Rozwiązywanie problemu należy zacząć od stworzenia funkcji „algorytm” będącej wektorem, który przyjmuje wartości w postaci wektora „tablica”.

Następnie tworzymy wektor wyniki, który za zadanie ma przechowywać parę wartości.

Kolejne linie to stworzenie zmiennej przyjmującej wartość równą rozmiarowi tablicy. Będzie ona sprawdzać wartość każdego indeksu tablicy, na której będzie pracował algorytm.

Następnie musimy wprowadzić warunek, który sprawdza maksymalną wielkość tablicy – będzie to potrzebne do mierzenia odległości na którą może poruszyć się algorytm w celu znajdywania danych.

Kolejnym krokiem jest znalezienie wszystkich indeksów z wartością „2”. To właśnie od niej będziemy szukać wyrazów podobnych w jednakowej odległości od liczby.

Następnym krokiem jest pętla, która porównuje wartości w odległości równej zmiennej „dlugosc”. Gdy znajdzie podobny wyraz, inkrementuje zmienną dlugosc, a następnie wprowadza zarówno indeksy jak i same wartości do wektora „wyniki” funkcją push_back.

Kolejny krok to stworzenie pętli, która będzie wykonywać się dopóki i nie osiągnie rozmiaru tablicy wyniki. Tworzymy w niej vector zmienna. Następnie dla każdego j mniejszego od drugiej wartości wektora wyniki, przypisujemy do wektora zmienna pierwszy element tablicy + iterator j. Następnie wprowadzamy te wartości do wektora converted.

Później możemy zauważyć część kodu odpowiedzialną za zapisywanie wyników w formie tablicy do pliku „test.txt”.

Funkcja kończy się zwróceniem wartości wektora converted.

4.2.2 Funkcja Symetria()

Następnym krokiem jest stworzenie funkcji „Symetria”. Funkcja przyjmuje wartość rz, która w domyśle będzie rozmiarem tablicy, który wprowadzi użytkownik po włączeniu programu.

Funkcja rozpoczyna od prostego warunku, sprawdzającego czy rozmiar tablicy jest większy od zera. W przeciwnym wypadku wyświetli stosowny komunikat.

Jeżeli rozmiar tablicy jest prawidłowy, zostaje zaimplementowana funkcja losująca wartości liczb w tabeli. Użyłem tego rozwiązania, aby nie musieć każdorazowo wpisywać ręczne podanych wartości. Następnie tworzymy wektor tablica. Odpowiednią funkcją resize nadajemy mu wielkość odpowiadającą wprowadzonej przez użytkownika wartości.
Kolejna pętla for losuje każdemu indeksowi wartość od 0 do 2.

Kolejnym krokiem jest stworzenie wektora „wyniki”

Następnie, aby zmierzyć czas działania algorytmu, implementujemy stosowną funkcję czasu, po czym przywołujemy funkcje algorytmu. Po skończeniu algorytmu kończy się również mierzenie czasu.

Kolejnym krokiem jest pętla wyświetlająca wyniki w formie przejrzystej tablicy. Nasze podciągi zostaną w ten sposób przedstawione użytkownikowi.

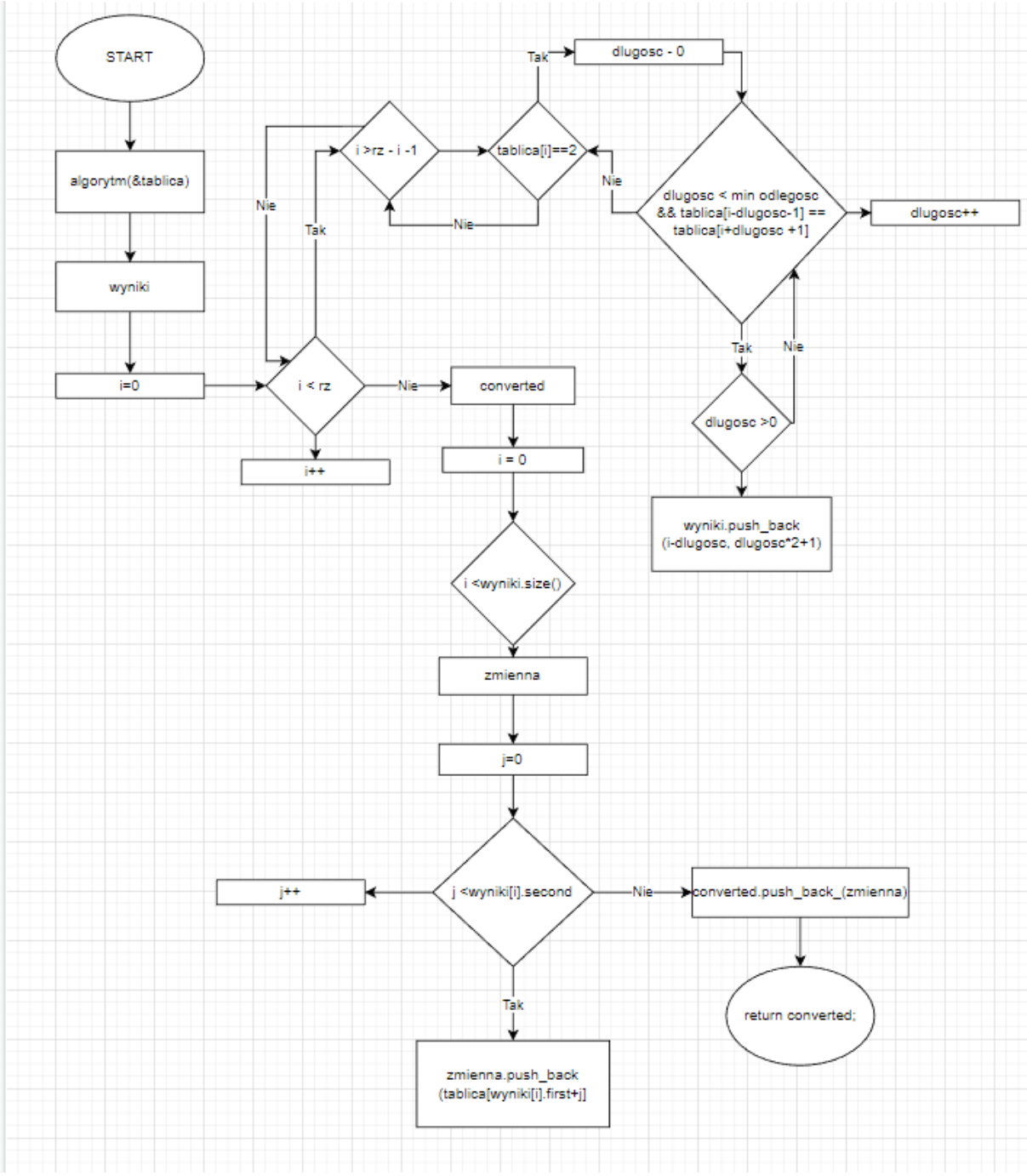
Na koniec funkcja wyświetla czas działania algorytmu, korzystając z wcześniej opisanej funkcji czasu.

4.2.3 Funkcja main()

Deklarujemy zmienną rz. Pozwalamy użytkownikowi na określenie jej wielkości (w domyśle jest to dowolna liczba rzeczywista większa od 0)

Pozostaje nam już tylko przywołać funkcję Symetria przyjmującą wartość odpowiadającą zmiennej o wartości wpisanej przez użytkownika

4.3 SCHEMAT BLOKOWY ALGORYTMU



Rysunek 1 – SCHEMAT BLOKOWY ALGORYTMU

4.4 PSEUDOKOD

```
rz = tablica.size()
dla i < rz
    jesli i>(rz - i - 1) to min odlegosc = rz - i -1
    jesli tablica[i]==2
        to dlugosc =0
        dopoki dlugosc < min_odleglosc
        && (tablica[i-dlugosc-1]
        == tablica[i + dlugosc +1
            to dlugosc zwieksz o 1
            jesli dlugosc jest > 0
            wstaw i-dlugosc, dlugosc*2+1
            do wyniki

    stworz wektor converted
    dla i < wyniki.size()
        stworz wektor zmienna
        dla j < wyniki[i].second
            wprowadz tablica[wyniki[i].first +j]
            do wektora zmienna

    wprowadz zmienna do wektora converted

zwroc converted
```

Rysunek 2 – PSEUDOKOD ALGORYTMU

5 KOD ALGORYTMU

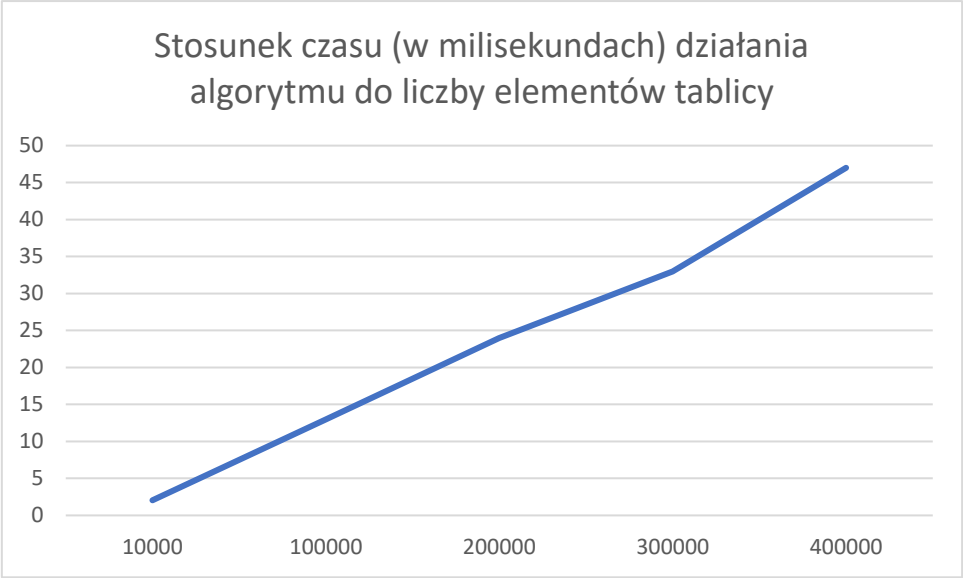
C: > Users > renga > OneDrive > Dokumenty > cpp.cpp > algorytm(std::vector<int>&)

```cpp
std::vector <std::vector<int>> algorytm(std::vector<int>&tablica)
{
    std::vector<std::pair<int,int>>wyniki;
    int rz = tablica.size();
    for(int i = 0; i <rz; i++) // Petla sprawdzajaca wartoœæ kazdego indeksu tablicy
    {

    int min_odleglosc = i;
        if (i > (rz - i - 1)) min_odleglosc = rz - i - 1; //Warunek sprawdzajacy maksymalna ilosc krokow "w lewo i prawo w tablicy"
        if (tablica[i]==2)  // Warunek sprawdzajacy wartosc pod indeksem - musi byc ona równa 2
        {
            int dlugosc = 0;
            while (dlugosc < min_odleglosc && (tablica[i-dlugosc-1] == tablica[i + dlugosc +1])) // Pêtla porównuj¹ca wartoœci w równych odleglosciach od i
            {
                dlugosc++;

            if (dlugosc > 0) wyniki.push_back(std::pair<int,int>(i-dlugosc, dlugosc*2+1 )); // Dodawanie wartosci do wektora wyniki

            }

        }
    }
    std::vector<std::vector<int>> converted;


    for(int i = 0; i < wyniki.size();i++)
    {
        std::vector<int> zmienna;
        for (int j = 0; j < wyniki[i].second; j++) {
            zmienna.push_back(tablica[wyniki[i].first + j]); //wpisanie do tablicy elementu tablicy wejsciowej +
        }
        converted.push_back(zmienna);
    }
    return converted;
}
```

Rysunek 3 – KOD ALGORYTMU

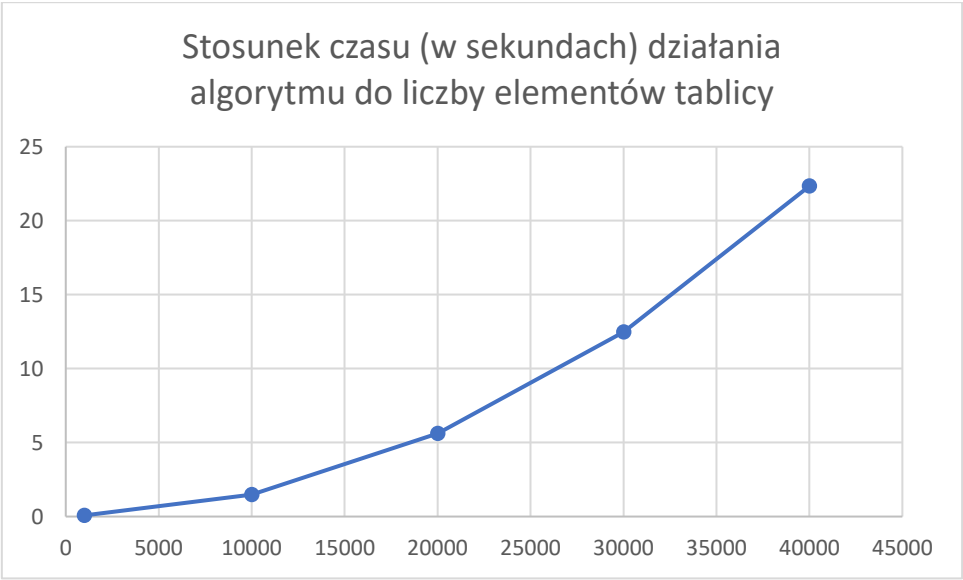
6 WYKRESY

6.1 WYKRES ZŁOŻONOŚCI CZASOWEJ W STOSUNKU DO ILOŚCI ELEMENTÓW TABELI (BEZ ZAPISYWANIA ICH DO PLIKU TXT)



Rysunek 4 -WYKRES

6.2 WYKRES ZŁOŻONOŚCI CZASOWEJ W STOSUNKU DO ILOŚCI ELEMENTÓW TABELI (Z ZAPISYWANIEM DO PLIKU TXT)



Rysunek 5 – WYKRES #2