

***Sprawozdanie projektowe z
przedmiotu algorytmy i struktury
danych***

*Wojciech Świder gr8
nr albumu 179988*

Wstęp

Treścią zadania projektowego było znalezienie dla zadanego ciągu liczb całkowitych (tablicy) najdłuższy podciąg o zadanej sumie. Zadanie posiadało również następujący przykład:

Wejście

$A[] = [0, 6, 5, 1, -5, 5, 3, 5, 3, -2, 0]$

Suma = 8

Wyjście

Podciągi o sumie 8 to: $[-5, 5, 3, 5]$, $[3, 5]$, $[5, 3]$.

Najdłuższy podciąg to: $[-5, 5, 3, 5]$ o długości 4.

Biorąc pod uwagę powyższy przykład z zadania zamierzam uwzględnić w programie wypisywanie każdego znalezionego podciągu, oraz wypisywanie długości najdłuższego podciągu.

Spis treści

Wstęp	1
Spis treści	2
1. Opis problemu	3
2. Opis podstaw teoretycznych zagadnienia	4
3. Opis szczegółów implementacji problemu	6
4. Schemat blokowy algorytmu	9
5. Algorytm zapisany w pseudokodzie	10
6. Rezultaty testów wykonanych na programie	11
7. Wnioski i podsumowanie	12

1. Opis problemu

Celem zadania będzie napisanie algorytmu który za zadanie dla zadanego ciągu liczb całkowitych (tablicy) ma wypisanie listy wszystkich możliwych podciągów o zadanej sumie. Następnie znalezienie spośród nich najdłuższego podciągu, wypisanie go oraz podanie jego długości.

2. Opis podstaw teoretycznych zagadnienia

Wczytywanie danych będzie się odbywało w formie podania przez użytkownika konsoli długości ciągu którego chce zbadać. Następnie program zapyta użytkownika o sumę jaką chce, aby miały znalezione podciąg. Po czym program wygeneruje tablice losowych liczb (zakres dla większości testów ustawiony na przedział liczb $<-5, 5>$).

Podstawą do rozwiązania zadania będzie zawarcie w algorytmie iteracji podanego wcześniej ciągu aby znaleźć możliwy podciąg o zadanej sumie S . Potrzeba też uwzględnić dużą możliwość występowania więcej niż jednego podciągu, dlatego też funkcja iterowania będzie występować w dwóch pętlach aby zbadać każdą możliwość. Długość wewnętrznej pętli J będzie ściśle zależna od pętli zewnętrznej N programu. Z każdą następną iteracją $[I]$ (zaczynając od $I = 0$) pętli zewnętrznej, pętla wewnętrzna będzie zaczynać swoje działanie od $P = I$ czyli następnego wyrazu ciągu. Z tego wynika, że algorytm będzie się wykonywał do momentu aż N będzie mniejsze od I . Poniżej znajduje się wizualizacja dwóch pętli i jak one na siebie oddziałują. Wyraźnie można zobaczyć, że w wewnętrznej pętli występuje "przesunięcie".

Iteracja w dwóch pętlach, z obrazowaniem									
$i = 0$	Długość iteracji $J = N - i$								→
Długość ciągu N	3	-4	0	5	-5	3	5	0	$i++$
		-4	0	5	-5	3	5	0	$i++$
			0	5	-5	3	5	0	$i++$
				5	-5	3	5	0	$i++$
					-5	3	5	0	$i++$
						3	5	0	$i++$
							5	0	$i++$
								0	$i++$
$y = i$	$y++$	$y++$	$y++$	$y++$	$y++$	$y++$	$y++$	$y++$	$i > N$

W istniejącej wewnętrznej iteracji pętli zostanie zawarta funkcja sprawdzająca czy podciąg jest równy zadanej sumie. Do tego w pętli zewnętrznej N będzie potrzebna zmienna `suma_pomocnicza` która będzie się zerować co iteracje pętli zewnętrznej. Zmienna `suma_pomocnicza` będzie przechowywać dodawane do siebie wyrazy ciągu w pętli wewnętrznej, przy każdym dodaniu wyrazu będzie sprawdzać czy `suma_pomocnicza` równa się zadanej sumie. Podciągi znalezione zostały oznaczone kolorem zielonym.

Zadana suma ciągu = 5							
3	-4	0	5	-5	3	5	0
	-4	0	5	-5	3	5	0
		0	5	-5	3	5	0
			5	-5	3	5	0
				-5	3	5	0
					3	5	0
						5	0
							0

W momencie spełniania warunku `suma_pomocnicza` równa sumie wykonuje się polecenie pętli wywołującej podciąg, nazwę ją pętlą O. Podciąg będzie iterowany liczbą *i*, czyli liczbą która odpowiada za iterowanie pętli zewnętrznej N. Iteracja wywołująca podciąg będzie trwać do osiągnięcia *y* czyli zmiennej iterującej pętle wewnętrzną J.

Na końcu spełnionego warunku `suma_pomocnicza` równa sumie wykona się przypisanie zmiennej `długość_podciągu` definiowana będzie wzorem $długość_podciągu = y + 1 - i$. Następnie program sprawdzi czy `długość_podciągu` jest największa dotychczas znaną długością podciągu, jeśli tak to `długość_podciągu` przypisuje swoją wartość do `max_długości` oraz do zapamiętania tego ciągu zapisujemy koordynaty następującymi zmiennymi: `maxStart = i` oraz `maxStop = y`. Po końcu iteracji pętli zewnętrznej wykona się pętla wywołująca najdłuższy znaleziony podciąg oraz jego długość. Wykona się ona przy pomocy iteracji ciągu od `maxStart` do `maxStop`, poza pętlą system wypisze `max_długość`.

3. Opis szczegółów implementacji problemu

Na początku program wymaga od użytkownika podanie wartości sumy i długości tablicy.

```
cout<<"podaj sume"<<endl;
int suma = 0;
cin>>suma;
int dlugosc_tablicy = 0;
cout << "podaj dlugosc tablicy" << endl;
cin >> dlugosc_tablicy;
```

Następnie z podanymi wcześniej wartościami program inicjuje funkcje algorytm.

```
algorytm(dlugosc_tablicy, suma);
```

W funkcji algorytm inicjujemy następujące zmienne oraz wypełniamy tablice liczbami w pętli. zmienna bool pierwszyZnaleziony służy jedynie do modelowania wyjścia dla lepszej wizualizacji w konsoli.

```
void algorytm(int dlugosc_tablicy, int suma) {
    int maxDlugosc = 0;
    int maxStart = 0;
    int maxStop = 0;
    bool pierwszyZnaleziony = false;
    auto start :time_point<system_clock> = std::chrono::high_resolution_clock::now(); /// kod mierzący czas programu
    int tablica[dlugosc_tablicy];
    srand(Seed:time(0));
    for (int i = 0; i < dlugosc_tablicy; i++) {          /// Generowanie tablicy losowych liczb
        tablica[i] = rand() %11 -5;
    }
}
```

W pętłach oprócz algorytmu opisanego wcześniej występują różne wypisywania do konsoli wykonywane po spełnieniu różnych warunków. Na sam koniec opisu zaprezentuję przykładowe wyjście programu.

```
for (int i = 0; i < dlugosc_tablicy; i++) {
    if (i == 0) cout << "Podciagi o sumie " << suma << " to: ";
    int sumaPomocnicza = 0;
    for (int j = i; j < dlugosc_tablicy; j++) {
        sumaPomocnicza += tablica[j];
        if (sumaPomocnicza == suma) {
            if (!pierwszyZnaleziony) {
                pierwszyZnaleziony = true;
                cout << "[";
            } else {
                cout << ", [";
            }
            for (int x = i; x <= j; x++) {
                if (x == j) cout << tablica[x];
                else cout << tablica[x] << ",";
            }

            cout << "]";
            int dlugoscPodciagu = j + 1 - i;
            if (dlugoscPodciagu > maxDlugosc) {
                maxStart = i;
                maxStop = j;
                maxDlugosc = dlugoscPodciagu;
            }
        }
    }
}
```

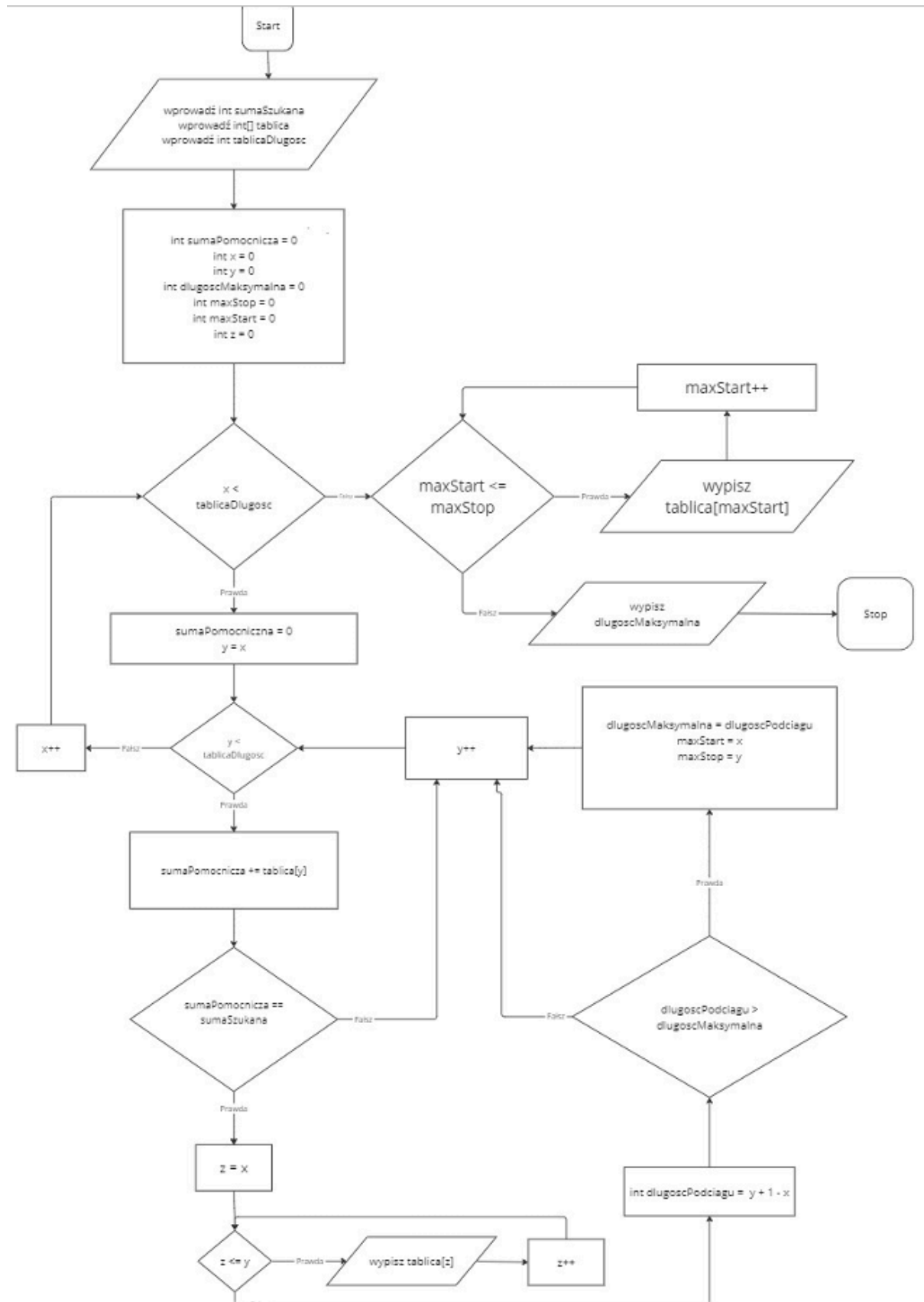

Na koniec funkcji algorytm znajduje się kod odpowiadający za wypisywanie najdłuższego ciągu (w kodzie znajdują się linijki odpowiadające za mierzenie czasu programu, są one z komentarzem).

```
if (maxDlugosc != 0) {
    cout << "." << endl << "Najdluzszy podciag to: " << "[";
    for (int x = maxStart; x <= maxStop; x++) {
        if (x == maxStop) cout << tablica[x];
        else cout << tablica[x] << ",";
    }
    cout << "]";
    cout << " o dlugosci " << maxDlugosc << ".";
} else cout << "brak podciagow.";
cout << endl;
auto end_time_point<system_clock> = std::chrono::high_resolution_clock::now(); /// kod mierzący czas programu
chrono::duration<double, milli> elapsed = end - start; /// kod mierzący czas programu
cout << elapsed.count() << "ms"; // czas programu w milisekundach
```

Przykładowe wejście i wyjście programu w konsoli.

```
podaj sume
5
podaj dlugosc tablicy
10
Podciagi o sumie 5 to: [-2,2,5], [-2,2,5,0], [5], [5,0], [5].
Najdluzszy podciag to: [-2,2,5,0] o dlugosci 4.
3.3505ms
Process finished with exit code 0
```

4. Schemat blokowy algorytmu



5. Algorytm zapisany w pseudokodzie

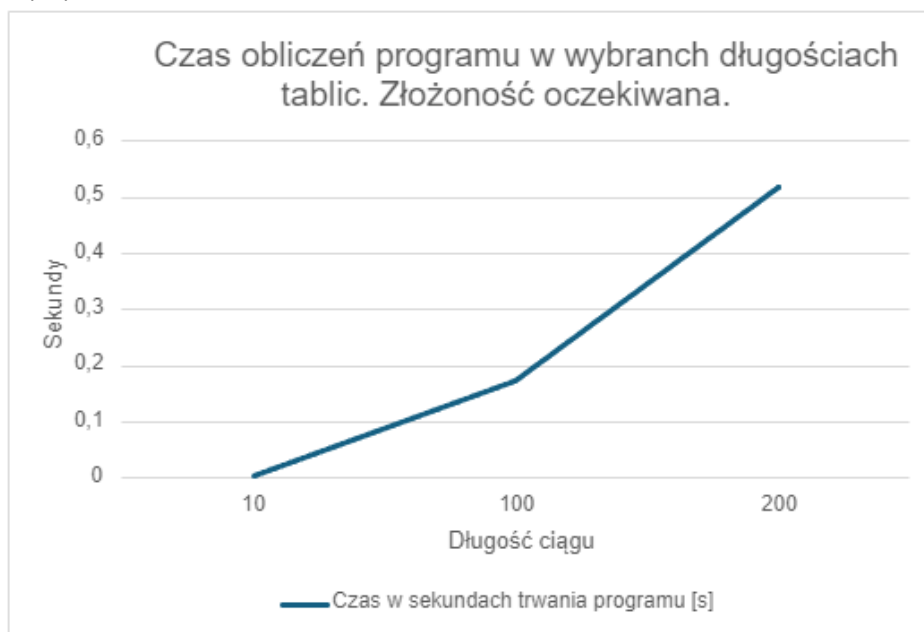
```
wczytaj(sumaSzukana)
wczytaj(tablica[])
wczytaj(tablicaDlugosc)
inicjuj sumaPomocniczna <-- 0
inicjuj dlugoscMaksymalna <-- 0
inicjuj maxStop <-- 0
inicjuj maxStart <-- 0
inicjuj z <-- 0

dla i <-- 0 do dlugoscTablicy - 1 powtarzaj
    sumaPomocniczna <-- 0
    dla j <-- i do dlugoscTablicy - 1 powtarzaj
        sumaPomocnicza <-- sumaPomocnicza + tablica[j]
        jezeli sumaPomocnicza = tablica[j] to
            dla z <-- i do j powtarzaj
                wypisz(tablica[z])
            inicjuj dlugoscPodciagu <-- j + 1 - i
            jezeli dlugoscPodciagu > dlugoscMaksymalna to
                dlugoscMaksymalna <-- dlugoscPodciagu
            maxStart <-- i
            maxStop <-- j
dla k <-- maxStart do maxStop powtarzaj
    wypisz(tablica[k])
wypisz(dlugoscMaksymalna)
```

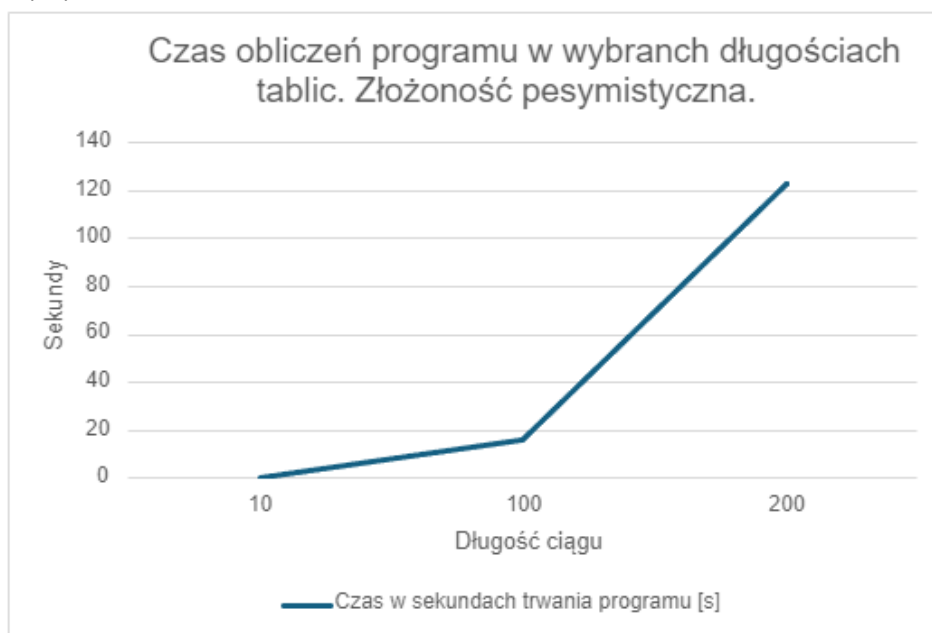
6. Rezultaty testów wykonanych na programie

Badając złożoność obliczeniową zauważyłem, że może ona się zmieniać w zależności od danych jakie program otrzyma. W złożoności oczekiwanej, czyli "typowych danych" program można opisać $O(n^2)$. Natomiast w momencie nietypowych danych przyjmijmy tablica = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] oraz suma = 0, złożoność obliczeniowa sięga $O(n^3)$ jest to przypadek pesymistyczny. Niżej przedstawiam wykresy dwóch przypadków.

$O(n^2)$:



$O(n^3)$



7. Wnioski i podsumowanie

Złożoność obliczeniowa programu wygląda satysfakcjonującą oprócz sytuacji kiedy każdy element ciągu tworzy podciąg, w takim momencie złożoność sięga $O(n^3)$. Program nie jest tak wydajny jak w oczekiwanych złożonościach. Jednak myślę, że zamysł do szukania podciągów o zadanej sumie w ciągu liczb nie powinien analizować takich ciągów w których przykładowo ciąg składa się z samych wyrazów równych 0 a suma ma wynosić 0, ponieważ nie ma to praktycznej wartości.