Sprawozdanie projektowe z przedmiotu algorytmy i struktury danych

Autor:	Wojciech Świder gr. 8
Nr albumu:	179988
Opiekun przedmiotu:	dr inż. Mariusz Borkowski

Rzeszów, 05.12.2024r.

Spis treści

1. Wstęp

Tematem niniejszego zadania projektowego było znalezienie dla zadanego ciągu liczb całkowitych przechowywanego w tablicy, najdłuższego podciągu o danej sumie. Wczytywanie danych odbywać się będzie poprzez podanie przez użytkownika konsoli wybranej długości ciągu. Następnie użytkownik dostanie możliwość wprowadzenia długości szukanych podciągów. Następnie program wygeneruje tablice o zadanej długości i wypełni je losowymi liczbami z zakresu <-5,5>. Przykładowe dane wejściowe i wyjściowe dla zadania wyglądają następująco:

Wejście:
a[ ] = [0,6,5,1,-5,5,3,5,3,-2,0]
suma = 8

Wyjście
Wszystkie podciągi tablicy a o sumie 8: [-5,5,3,5], [3,5], [5,3]
Najdłuższy podciąg o sumie 8: [-5,5,3,5]

Jak widać na powyższym przykładzie, podciągów zadanej tablicy spełniających warunek zadania może być kilka, zatem pierwszym etapem rozwiązania problemu będzie znalezienie wszystkich podciągów spełniających warunek sumy. Następnie nastąpi wybranie spośród nich podciągu lub podciągów o największej liczbie wyrazów wraz z podaniem jego (bądź ich) długości.

2. Teoretyczny opis zagadnienia

Rozwiązanie zadanego problemu będzie opierało się o iterację po kolejnych wyrazach ciągu wyznaczających początek badanego podciągu (J) oraz pętli zagnieżdżonej iterującej po kolejnych wyrazach podciągu wyznaczających ostatni wyraz badanego podciągu (N). Przy każdej iteracji pętli wewnętrznej program sprawdza, czy suma wyrazów ciągu jest równa zadanej przez użytkownika sumie oraz czy bieżący podciąg jest dłuższy od aktualnie najdłuższego znalezionego dotąd ciągu spełniającego założenia. Długość ta przechowywana jest w zmiennej maxDlugosc. Ze względu na dużą możliwość wystąpienia więcej niż jednego podciągu o zadanej sumie i największej liczbie wyrazów, uwzględniony zostaje jedynie pierwszy wyszukany taki ciąg (przy sprawdzaniu drugiego warunku długość równa nie wystarczy).

Liczba iteracji pętli zewnętrznej J będzie wynosić dokładnie tyle, ile wyrazów ma ciąg. Liczba iteracji wewnętrznej pętli N będzie wynosiła tyle, ile wyrazów jest w ciągu minus liczba wyrazów poprzedzająca pierwszy wyraz aktualnie badanych podciągów. Z każdą następną iteracją pętli zewnętrznej, pętla wewnętrzna będzie zaczynać podciągi od następnego wyrazu ciągu. Z każdą następną iteracją pętli wewnętrznej, bieżący podciąg będzie kończył się na następnym wyrazie. Z powyższego wynika, że algorytm będzie wykonywał się do momentu aż pętle przeiterują po każdym możliwym pociągu tablicy. Niżej zamieszczona została wizualizacja dwóch pętli i jakie podciągi były analizowane.

W pętli zewnętrznej będzie potrzebna zmienna nazwana suma_pomocniczna, która będzie się zerować co iteracje pętli zewnętrznej. Zmienna suma_pomocnicza będzie przechowywać dodawane do siebie wyrazy ciągu w pętli wewnętrznej, przy każdym kolejnym dodaniu wyrazu będzie sprawdzać, czy suma_pomocnicza równa się zadanej sumie.

Przy spełnionym warunku suma_pomocnicza równa sumie zadanej wykona się przypisanie zmiennej długość_podciągu. Następnie program sprawdzi czy długość_podciągu jest największa dotychczas znalezioną długością podciągu, jeśli tak to długość_podciągu przypisuje swoja wartość do max_długość oraz do zapamiętania tego ciągu zapisane zostaną numery początkowego i końcowego wyrazu w tabeli: maxStart oraz maxStop. Następnie program wypisze podciąg przy pomocy iteracji po tabeli od maxStart do maxStop. Po zakończonych iteracjach program wypisze najdłuższy podciąg oraz liczbę jego wyrazów. Następnie program wypisze max_długość i czas trwania operacji.

Zadana suma ciągu = 5											
Aktualnie analizowany podciąg								wartość i	wartość j	suma = = 5	czy max dług.?
3	-4	0	5	-5	3	5	0	0	0	false	-
3	-4	0	5	-5	3	5	0	0	1	false	-
3	-4	0	5	-5	3	5	0	0	2	false	-
3	-4	0	5	-5	3	5	0	0	3	false	-
3	-4	0	5	-5	3	5	0	0	4	false	-
3	-4	0	5	-5	3	5	0	0	5	false	-
3	-4	0	5	-5	3	5	0	0	6	false	-
3	-4	0	5	-5	3	5	0	0	7	false	-
3	-4	0	5	-5	3	5	0	1	1	false	-
3	-4	0	5	-5	3	5	0	1	2	false	-
3	-4	0	5	-5	3	5	0	1	3	false	-
3	-4	0	5	-5	3	5	0	1	4	false	-
3	-4	0	5	-5	3	5	0	1	5	false	-
3	-4	0	5	-5	3	5	0	1	6	false	-
3	-4	0	5	-5	3	5	0	1	7	false	-
3	-4	0	5	-5	3	5	0	2	2	false	-
3	-4	0	5	-5	3	5	0	2	3	true	true
3	-4	0	5	-5	3	5	0	2	4	false	-
3	-4	0	5	-5	3	5	0	2	5	false	-
3	-4	0	5	-5	3	5	0	2	6	false	-
3	-4	0	5	-5	3	5	0	2	7	false	-
3	-4	0	5	-5	3	5	0	3	3	true	false
3	-4	0	5	-5	3	5	0	3	4	false	-
3	-4	0	5	-5	3	5	0	3	5	false	-
3	-4	0	5	-5	3	5	0	3	6	false	-
3	-4	0	5	-5	3	5	0	3	7	false	-
3	-4	0	5	-5	3	5	0	4	4	false	-
3	-4	0	5	-5	3	5	0	4	5	false	-
3	-4	0	5	-5	3	5	0	4	6	false	-
3	-4	0	5	-5	3	5	0	4	7	false	-
3	-4	0	5	-5	3	5	0	5	5	false	-
3	-4	0	5	-5	3	5	0	5	6	false	-
3	-4	0	5	-5	3	5	0	5	7	false	-
3	-4	0	5	-5	3	5	0	6	6	true	false
3	-4	0	5	-5	3	5	0	6	7	true	false
3	-4	0	5	-5	3	5	0	7		false	-

Tabela 1 - Wizualizacja przeszukania podciągów i znalezienia najdłuższego spełniającego założenia

3. Implementacja rozwiązania

Funkcja main wykonuje komplet funkcji w zależności od preferowanej wersji inputu. Wersja 1 korzysta z generowania tablicy na podstawie pliku tekstowego. Wersja druga wymaga podania od użytkownika długości tablicy, następnie program generuje liczby do tablicy z zakresu <5,-5>.

```
// int *tablica = fileArrayReader();                    ////////GENEROWANIE TABLICY Z PLIKU
// int dlugosc_tablicy = fileLengthSeeker();            ////////GENEROWANIE TABLICY Z PLIKU
// int suma = localSetData2();                          ////////GENEROWANIE TABLICY Z PLIKU
 // algorytm(tablica, suma, dlugosc_tablicy);           ////////GENEROWANIE TABLICY Z PLIKU
 // delete tablica;                                     ////////GENEROWANIE TABLICY Z PLIKU
```

Fragment kodu 1 - Wersja 1.

```
int suma = localSetData2();                         ////////GENEROWANIE PSEUDOLOSOWEJ TABLICY W PROGRAMIE
int dlugosc_tablicy = localSetData();               ////////GENEROWANIE PSEUDOLOSOWEJ TABLICY W PROGRAMIE
int *tablica = generateArray(dlugosc_tablicy);      ////////GENEROWANIE PSEUDOLOSOWEJ TABLICY W PROGRAMIE
algorytm(tablica, suma, dlugosc_tablicy);          ////////GENEROWANIE PSEUDOLOSOWEJ TABLICY W PROGRAMIE
delete tablica;                                     ////////GENEROWANIE PSEUDOLOSOWEJ TABLICY W PROGRAMIE
```

Fragment kodu 2 - Wersja 2.

W obu wersjach wykona się ta sama funkcja nazwana algorytm.

```
int algorytm(int tablica[], int suma, int dlugosc_tablicy) {
    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

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

Fragment kodu 3 - Funkcja algorytm

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

Fragment kodu 4 – Druga część funkcji algorytm

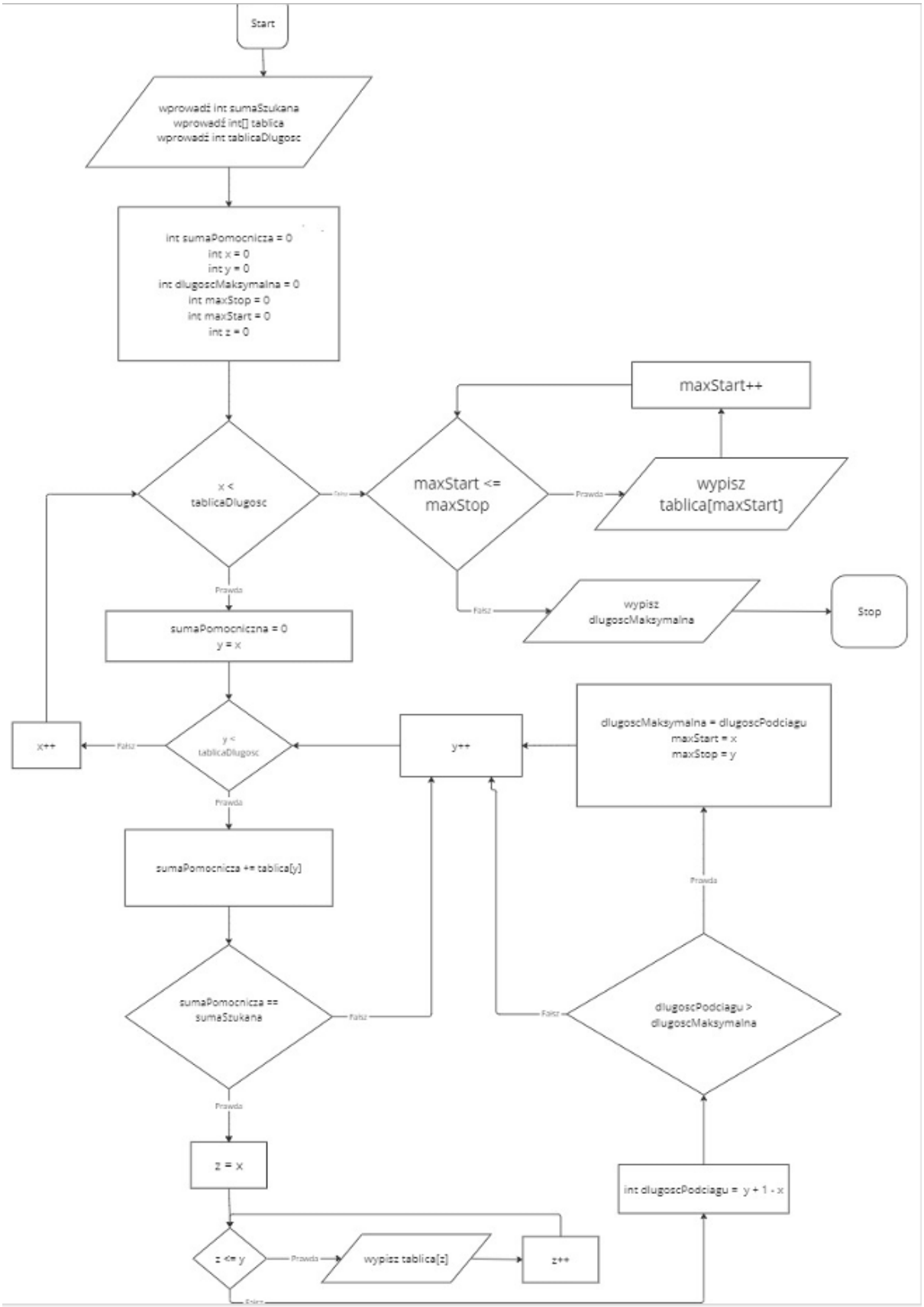
W pierwszej części funkcji algorytm, program wypisuje wszystkie znalezione podciągi odpowiadające zadanej sumie. Algorytm również zachowuje w zmiennych maxStart, maxStop wartości do wyznaczenia w dalszej części najdłuższego znalezionego podciągu.

```
if (maxDlugosc != 0) {
    cout << "." << endl << "Najdluzszy podciag to: " << "[";
    wynikFile << "." << endl << "Najdluzszy podciag to: " << "[";
    for (int x = maxStart; x <= maxStop; x++) {
        if (x == maxStop) {
            cout << tablica[x];
            wynikFile << tablica[x];
        }
        else {
            cout << tablica[x] << ",";
            wynikFile << tablica[x] << ",";
        }
    }
    cout << "]";
    wynikFile <<"]";
    cout << " o dlugosci " << maxDlugosc << ".";
    wynikFile<<" o dlugosci "<<maxDlugosc<< ".";
} else {
    cout << "brak podciagow.";
    wynikFile<<"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" << endl; // czas programu w milisekundach
return maxDlugosc;
```

Fragment kodu 5 - Końcowe wypisania

Dodatkowo program zapisuje znalezione podciągi w pliku tekstowym wynikZadania.txt niezależnie od wybranej wersji programu w funkcji main.

4. Schemat blokowy algorytmu



Rysunek 1 - Schemat działania algorytmu

5. Zapis algorytmu w pseudokodzie

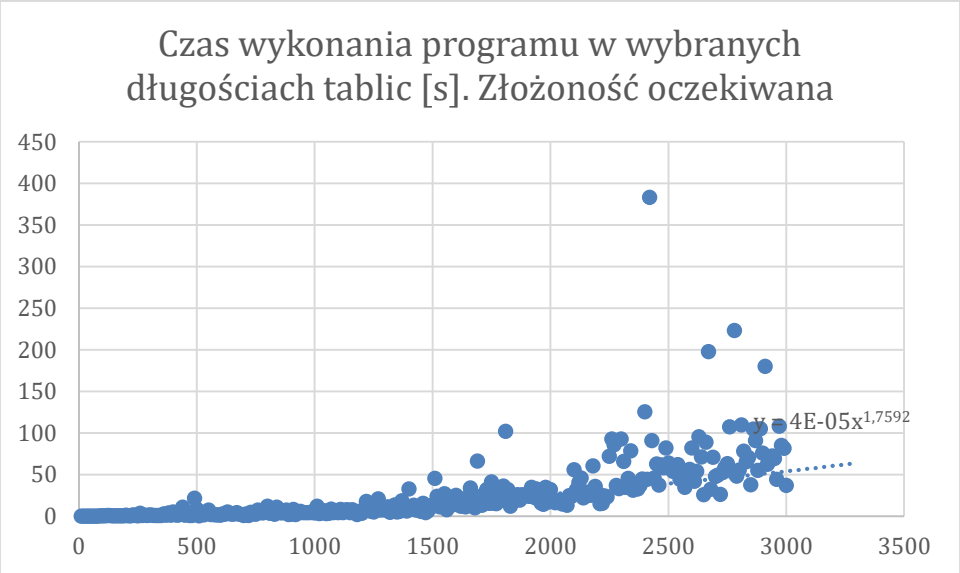
```
wczytaj(sumaSzukana)
wczytaj(tablica[])
wczytaj(tablicaDlugosc)
inicjuj sumaPomocniczna ← 0
inicjuj dlugoscMaksymalna ← 0
inicjuj maxStop ← 0
inicjuj maxStart ← 0
inicjuj z ← 0

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

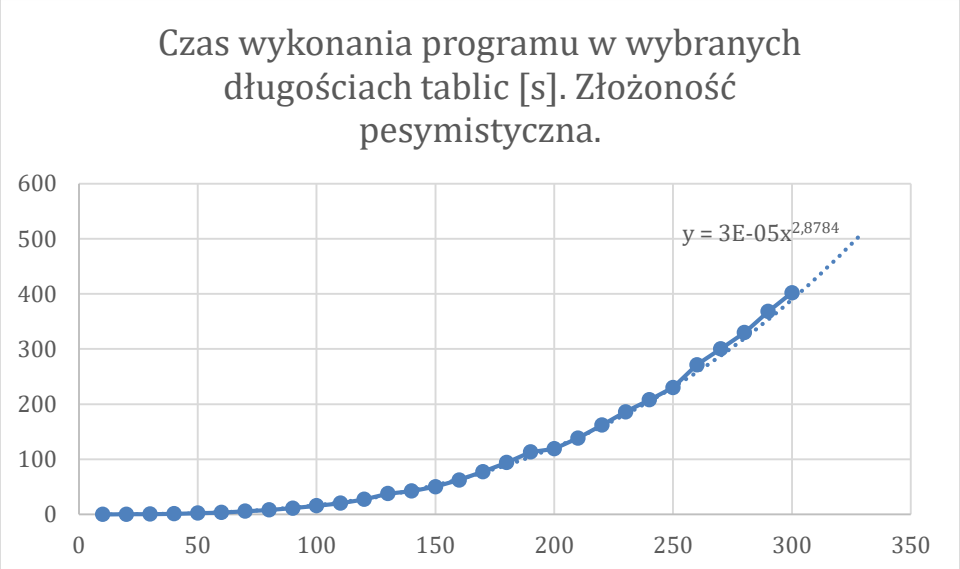
6. Wyniki przeprowadzonych testów

Podczas przeprowadzania testów analizowano przypadki długości tabeli zaczynając od tabeli o długości 10, za każdym razem inkrementując wynik o 10. W złożoności oczekiwanej, czyli dla losowo dobranych danych, program przyjmuje złożoność obliczeniową O(n^2). Rozpatrzono przypadki do

Natomiast w momencie nietypowych danych, dobranych specjalnie aby maksymalnie wydłużyć działanie algorytmu niezależnie od długości zadanej tabeli, złożoność obliczeniowa sięga O(n^3). Przypadek pesymistyczny zakłada, że przy każdej długości zadanej tabeli jest ona wypełniona wartościami 0, a zadana suma wynosi 0. Wobec czego każdy z możliwych podciągów zadanej tabeli spełnia założenie o sumie i będzie przeanalizowany. Linia trendu



Rysunek 2 – Czas wykonania programu. Złożoność oczekiwana.



Rysunek 3 – Czas wykonania programu. Złożoność pesymistyczna.

7. Podsumowanie

Złożoność obliczeniowa programu wygląda zgodnie z oczekiwaniami, wyłączając sytuację, gdy każdy element ciągu jest równy 0, a zadana suma również wynosi 0, zatem każdy podciąg tabeli spełnia założenie o sumie. W takiej sytuacji złożoność osiąga $O(n^3)$. Przeprowadzona analiza wykazała, że algorytm skutecznie identyfikuje podciągi spełniające zdefiniowany warunek, zapewniając poprawność wyników w testowanych przypadkach. Algorytm działa sprawnie dla krótkich i średniej długości ciągów. W przypadku dużych zbiorów danych obserwuje się jednak znaczny wzrost czasu przetwarzania, co sugeruje możliwość optymalizacji. Obecna implementacja charakteryzuje się ograniczoną skalowalnością, co może być problematyczne przy pracy z dużymi zbiorami danych w praktycznych zastosowaniach.

Propozycje dalszej pracy nad projektem obejmować mogą zastosowanie bardziej zaawansowanych metod, aby poprawić efektywność obliczeniową. Należy do nich również zbadanie możliwości równoległego przetwarzania danych dla przyspieszenia analizy w dużych zbiorach. Można też rozszerzyć zestaw testów, szczególnie o przypadki brzegowe i scenariusze nietypowe, a także przeprowadzić testy wydajnościowe na rzeczywistych danych, aby ocenić praktyczne ograniczenia algorytmu.