POLITECHNIKA RZESZOWSKA

Wydział Matematyki i Fizyki Stosowanej

Zadanie projektowe 1 - sprawozdanie

Algorytmy i struktury danych

Oliwia Konefał

Nr albumu: 173157

1. Wstęp oraz opis problemu

Algorytm ukazany w programie ma za zadanie rozwiązywać następujący problem: Znajdować malejący podciąg o największej długości, dla ciągu (w postaci tablicy) zawierającego wartości całkowite.

Przykład:

Wejście: A[] = [-10, 5, 8, 1, -4, -4, 10, 3, -1, 1]

Wyjście: Najdłuższe malejące podciągi to 8, 1, -4 oraz 10, 3, 1.

W kodzie programu zostały wykorzystane powyższe, konkretne dane. Algorytm zadziała również dla dowolnych, innych wartości oraz każdej długości tablicy. Do zagadnienia, pomimo krótkiego polecenia, można podejść na wiele, różnych sposobów.

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

2. Algorytm zapisany w pseudokodzie

MAIN()

Start

getLargestSubsets(tablica, długośćTablicy)

Stop

FUNKCJA getLargestSubsets(tablica int, długośćTablicy int) {

string obecnieSprawdzanyPodzbior = String(pierwszyElementTablicy)

string koncowaListaPodzbiorow;

int dlugoscObecnieSprawdzanegoPodzbioru = 1;

int dlugoscNajdłuższegoPodzbioru = 1;

for(int i = 1; i ≤ długoscTablicy - 1; i++) {

if(tablica[i] < tablica[i-1]) {

dlugoscObecnieSprawdzanegoPodzbioru++;

dlugoscNajdluzszegoPodzbioru = długoscNajdluzszegoPodzbioru + “, ” + String(tablica[i])

}

else {

if (długośćNajdluzszegoPodzbioru == dlugoscObecnieSprawdzanegoPodzbioru) { koncowaListaPodzbiorow = koncowaListaPodzbiorow + “ oraz ” + obecnieSprawdzanyPodzbior

}

else if (dlugoscNajdłuższegoPodzbioru < długośćObecnieSprawdzanegoPodzbioru) {

długośćNajdluzszegoPodzbioru = dlugoscObecnieSprawdzanegoPodzbioru

koncowaListaPodzbiorow = obecnieSprawdzanyPodzbior

```
}

dlugoscObecnieSprawdzanegoPodzbioru = 1

obecnieSprawdzanyPodzbior = String(tablica[i])

}

}

if (dlugoscNajdluzszegoPodzbioru == dlugoscObecnieSprawdzanegoPodzbioru)
{ koncowaListaPodzbiorow = koncowaListaPodzbiorow + “ oraz ” +
obecnieSprawdzanyPodzbior

} else if (dlugoscNajdluzszegoPodzbioru <
dlugoscObecnieSprawdzanegoPodzbioru) {

dlugoscNajdluzszegoPodzbioru = dlugoscObecnieSprawdzanegoPodzbioru

koncowaListaPodzbiorow = obecnieSprawdzanyPodzbior

}

„Najdluzsze malejace podciagi to: ” koncowa listaPodzbiorow

}
```

3. Opis szczegółowych implementacji problemu

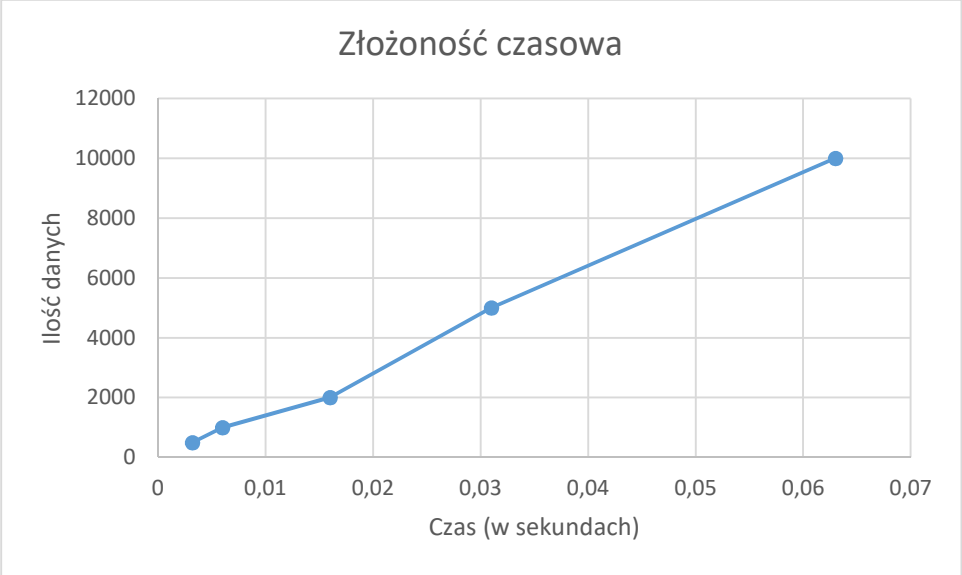
Iterując po tablicy liczb sprawdzamy czy obecny element jest mniejszy od poprzedniego.

- Jeśli tak, to oznacza, że długość obecnie sprawdzanego podciągu wzrasta o jeden. W związku z tym inkrementujemy zmienną currentSubsetLength oraz dodajemy wartość obecnie sprawdzanego elementu tablicy do zmiennej currentSubset, oddzielając elementy podciągu znakami “, ”.
- Jeśli nie, to oznacza, że nasz podciąg został przerwany. Teraz, w zależności od rezultatu porównania obecnie najdłuższego znalezionego podciągu (lengthOfLargestSubset) do tego, który własnie został przerwany (currentSubsetLength) będą wykonywać się inne operacje. Jeśli okaże się, że obecny podzbiór jest tak samo długi jak istniejący najdłuższy to oznacza, że mamy dodatkowy najdłuższy podzbiór. Wtedy, do naszej końcowej zmiennej finalSubset dodajemy obecny podciąg odzielony frazą “ oraz ”. Jeśli długość najdłuższego podciągu jest mniejsza od tego właśnie przerwanego to oznacza, że mamy nowy, jedyny, najdłuższy podciąg. Dlatego wartość lengthOfLargestSubset nadpisujemy długością tego przerwanego ciągu, a wartość finalSubset nadpisujemy tym ciągiem. Na końcu bloku kodu, który jest wywołany, resetujemy tymczasowe wartości przechowujące dane o obecnym ciągu, który został właśnie przerwany.

Ostatni blok kodu, wywołujący się po iteracji po tablicy, ma za zadanie obsługę sytuacji, w której ostatni element tablicy przekazanej do funkcji jest częścią najdłuższego podciągu. Jeśli ostatni element tablicy jest kontynuacją podciągu, to podczas jego iteracji wpadnie on w warunek if(arr[i] < arr[i-1]). Tymczasowe wartości zostaną zwiększone, ale nie zostaną wywołane operacje porównujące obecny podzbiór z najdłuższym podzbiorem. Dlatego musimy ostatni raz porównać te wartości by potencjalnie nadpisać najdłuższy istniejący ciąg.

4. Złożoność czasowa oraz obliczeniowa

W programie zostały wykonane testy mierzące czas wykonywania algorytmu dla podanych ilości danych: 500, 1000, 2000, 5000 i 10000.



Wykres ukazuje zależność – im więcej danych wejściowych, tym więcej czasu wymaganego czasu na obliczenia.

Złożoność obliczeniowa – liniowa – 0(n) – czas działania jest proporcjonalny do rozmiaru danych wejściowych.

Wykonane testy potwierdzają złożoność obliczeniową.

5. Wnioski i podsumowanie

- Główny kod programu został zaimplementowany w oddzielnej funkcji, która jest wywoływana wewnątrz programu
- Wykonano testy sprawdzające złożoność obliczeniową oraz czasową
- Program wypisuje wyniki w konsoli
- Kod programu został opatrzony stosownymi komentarzami, pomagające lepiej zrozumieć przeprowadzane operacje

Spis treści