ASD zadanie programistyczne 1
Serhii Kovalenko s17187 25c

I. Opis rozwiązania:

Zadaniem mojego programu jest znaleźenie najdłuższego podciągu stabilnego tablicy wzgłędem liczby maxDiff. Tablicę elementów, jej dlugość oraz liczbę maxDiff odczytujemy z pliku tekstowego (jak jest wymagane w zadaniu). Plik tekstowy o nazwie "Array.txt" musi znajdować się w katalogu glównym. Program jest napisany w języku Java 8.

Dlatego, żeby znaleźcz najdłuższy podciąg stabilny musimy znać długości wszystkich podciągów stabilnych tablicy, porównac ich i znaleźcz najdłuższy. W tym celu stworzyłem dwie metody: **getEnd()** i **getLongestSubarray()**. Podciąg stabilny to jest taki podciąg, że wszystkie jego elementy róznią się między sobą (bez względu na znak) nie więcej niż na liczbę maxDiff. Do znależenia takiego podciągu sluży metoda **getEnd().** Metoda **getEnd()** przyjmuje jako parametry tablicę elementów E, liczbę maxDiff oraz indeksy p i k i sluży do znaleźenia podciągu, który zaczyna się od elementu o indeksie p na przedziale od p do k. Ta metoda działa w sposób nastepujący: w pętli od i = p+1 do k wlącznie porównujemy róznice bezwzględną elemetów o indeksach p oraz i z liczbą maxDiff. Jeżeli ta róznica będzie większa, zapamiętujemy indeks i-1 jako indeks końca k i przerywamy działanie pętli. Następnie porównujemy indeksy p+1 i k. Jeżeli p+1 jest mniejszy od k, wywolujemy rekurencyjnie metodę **getEnd()** dla ciągu od p+1 do k. Jeżeli jest większy bądź równy k, wtedy zwracamy k i mamy koniec tego podciągu. Po

zakończeniu działania tej metody mamy indeksy początku i końcu podciągu stabilnego (p i k odpowiednio).
Teraz musimy znaleźcz indeksy najdłuższego takiego podciągu wykorzystując metodę **getLongestSubarray().** Ona przyjmuje w parametrach tablicę elementów E, liczbę maxDiff i dlugość tablicy E n. Tworzymy zmieną maxLength = 0, która będzie przechowywała długość najdłuższego podciągu stabilnego oraz p i k, które przechowują indeksy pociątku i końca tego podciągu. Teraz musimy znaleźcz wszystkie takie podciągi i porówanać ich długości z długością maxLength i jeżeli ta nowa długość jest większa, zapisać ją w maxLength, a indeksy pociątku i końca w zmienne p i k odpowiednio. W tym celu użyjemy pętli for od 0 do n dla znależenia wszystkich podciągów stabilnych (przy użyciu metody **getEnd()**).
Ponieważ szukamy najdłuższego podciągu, nie ma sensu sprawdzać długości podciągów na odcinkach, które są mniejsze bądź równe maxLength, więc przed wykonaniem kolejnej iteracji sprawdzamy, czy (n-i+1) (czyli długość podtablicy, na której będziemy szukali podciągu stabilnego) jest mniejsza bądź równa maxLength, i jeżeli jest, przerywamy działanie pętli. Po wyjściu z tej pętli będziemy wiedzieli indeksy początku i końca, wiec przypisujemy z tablicy E w tablicę result elementy od p do k i zwracamy tą tablicę.

II. Oszacowanie złożoności algorytmu:

Podstawową operacją w moim algorytmie jest porównanie elementów tablicy, więc oszacowywać złożoność będziemy względem tej operacji.

1) Koszt pesymistyczny:
Najgorsza złozoność czasowa będzie wtedy, kiedy cała tablica E będzie ciągiem róznych od siebie liczb posortowanych rosnąco, a liczba maxDiff = n/2 -1 (n.p. dla E = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], n = 10, maxDiff = 4). Wtedy długość każdego podciągu stabilnego będzie równa n/2 przy każdej iteracji pętli for w metodzie **getLongestSubarray(),** więc złożoność metody **getEnd()** ze względem rekursii będzie równa
W1(n) = (n/2-1) + (n/2-2) +...+1 = ((n/2 – 1)(n/2 – 1 +1))/2 =
= ((n/2 – 1)*n/2)/2 = O(n^2). Ponieważ długosc najdłuższego podciągu stabilnego będzie równa n/2, przerwiemy działanie pętli for w metodzie **getLongestSubarray()** kiedy zostanie nam do dyspozycji n/2 elementów tablicy, stąd liczba wykonań pętli for w tej metodzie będzie równa n/2 razy. Wtedy złożoność pesymistyczna całego algorytmu będzie W(n) =
= n/2 * W1(n) = n/2 * ((n/2 – 1)*n/2)/2 = (n^3/8 – n^2/4)/2=
= n^3/16 – n^2/8 = O(n^3).

2) Koszt średni:
Natomiast koszt średni tego algorytmu policzyć jest bardzo trudno moim względem. W szególności problem polega na wyowlaniu rekuręcyjnym metody **getEnd()** i wpływ wyników działania tej metody na liczbę wykonań pętli for w metodzie **getLongestSubarray().** Naprzykład dla zbyt dużej maxDiff może tak zdarzyć, że cała tablica E będzie podciągiem stabilnym. Wtedy zlożoność czasowa metody **getEnd()** będzie wynosiła T1(n) = (n-1) + (n-2) +… + 1 = O(n^2), ale pętla for w metodzie **getLongestSubarray()** wykona się tylko jeden raz, ponieważ już za pierwszym razem otrzymamy najdłuższy podciąg, więc następne iteracje już nie będą miały sensu, i wtedy cała złożoność algorytmu będzie wynosiła

T(n) = 1*T1(n) = O(n^2). A dla danych postaci E = [1, 3, 1, 3, 1, 3, …] i maxDiff = 1 złożoność czasowa **getEnd()** w każdej iteracji pętli for metody **getLongestSubarray()** będzie równa T1(n) = 1; ta pętla wykona się n-1 razy, więc cała złożoność będzie równa T(n) = 1 *(n-1) = O(n). Chodzi mi o to, że w danym przypadku niezwykle trudno jest oszacować złożoność średnią, ponieważ dla każdej iteracji metoda rekuręcyjna będzie miała rózną złożoność czasową (maksymalna złożoność to O(n^2)), która zależy od danych wejsciowych, i również liczba tych iteracji może zmienić się (maksymalnie może być n-1 iteracji).

3)Koszt pamieciowy:
W każdym przypadku potzebujemy stworzenia 1 tablicy wynikowej, więc złożoność pamięciowa jest równa
S(n) = O(n).