ASD zadanie programistyczne 2
Serhii Kovalenko s17187 25c

I. Opis rozwiązania:

Zadaniem mojego programu jest sprawdzenie, czy podany w postaci tablicy graf jest drzewem, i jeżeli jest, to również obliczenie wagi najcięższego i najłatwiejszego jego poddrzewa. Plik tekstowy o nazwie “Tree.txt” musi znajdować się w katalogu glównym i zawierać treść taką, jaka jest wymagana w zadaniu. Program jest napisany w języku Java 8.

Graf nazywamy drzewem jeżeli on nie zawiera cykli i liczba jego krawędzi jest równa liczbie jego wierzhołków zmniejszonej o 1. W celu sprawdzenia tych warunków stworzona została metoda **dfs(int v)**, która w parametrach przyjmuje wierzhołek, od którego zaczyna przeszukiwanie w głąb. Kiedy odwiedzimy wierzhołek, musimy to zaznaczyć, więc stworzymy tablicę **visited[]** typu boolean. Jeżeli wierzchołek v jeszcze nie był wcześniej przez nas odwiedziony, to zaznaczamy **visited[v]** jako true, a jeżeli był, to dany graf nie jest drzewem, czyli **isTree = false**. Jeśli wszystko jest OK, musimy zwiększyć liczbę krawędzi o jeden. Zadanie wymaga jeszcze policzyć wagi najciezszego i najłatwiejszego poddrzew, więc zwiększamy jescze zmienne **min** i **max** o **(v+1)** (dane odczytane z pliku zostały zapisane w tablicę dwuwymiarową **m[][]**, a numeracja komórek zaczyna się od zera). Potem sprawdzamy długość podtablicy **m[v]**, czyli, czy z krawędzi v wychodzą inne krawędzi, i jeżeli tak jest, wywolujemy metodę **dfs(int v)** dla tych krawedzi. Tak wygląda sprawdzenie, czy podany graf jest drzewem. Ale

teraz mamy wybrać wierzchołek, od którego zacząć to sprawdzenie. W tym celu musimy znaleźć ten najwyższy wierszchołek, z którego wychodzą wszystkie inne. W tym celu stworzyłem dwie metody: **getVertex(int m[][])**, która zwróci nam ten najwyższy wierzchołek i **getPrev(int m[][], int v)**, która zwraca “ojca” wierzcołka v. W metodzie **getVertex(int m[][])** tworzymy zmienną **prev = -1**, a potem w cyklu for szukamy wierzchołka, z którego nie wychodzą żadne krawędzi, a potem tylko dla tego wierzchołka wywolujemy metodę **getPrev(int m[][], int v)** i prypisujemy jej wynik na zmienną **prev**. Znaleźenie takiego wierzchołka jest ważne, ponieważ jeżeli wierchołka bez “dzieci” nie będzie, wtedy możemy odrazu powiedzieć, że ten graf nie jest dzewem i nie tracić potem czasu na wykonanie metody **dfs(int v)**. Po wykonaniu metody **getPrev(int m[][], int v)** po prostu zwrócimy tą zmienną **prev**, która będzie najwyższym wierzchołkiem naszego grafu.
Teraz zajmiemy się metodą **getPrev(int m[][], int v)**. Musimy przejść po całej tabeli **m[][]** dopóki nie znajdziemy taki wierzchołek, z którego wychodzi wierzchołek v (czyli **m[i][j] == v**). Kiedy taki znajdziemy, przypiszemy **prev = i+1** (numeracja komórek tablicy zaczyna się od zera, a numeracja wierzchołków od 1!!!) i przerywamy działanie obu pętli. Ta metoda jest rekuręcyjna, więc istnieje zagrożenie tego, że program może się zapętlić (n.p. dla takiej tablicy m [][] = { {}, {1, 3}, {2} };). Zeby tego nie dopuścić muśimy tak samo, jak w metodzie **dfs(int v)**, zaznaczyć odwiedzone wierzchołki. Dla tego tworzymy tablicę **boolean used[]** i na początku metody **getPrev(int m[][], int v)** zaznaczamy **used[prev - 1] = true**. W końcu, jeżeli **prev != v** (jeżeli jest, znaczy to, że ten wierzchołek nie ma poprzedników i jest to najwyższy wierzchołek grafu, więc zwracamy go), i jeżeli nie był on

wcześniej używany (**!used[prev-1]**), to wywolujemy metodę **getPrev(int m[][], int v)**, i jej wynik przypisujemy na zmienną **prev**, a jeżeli ten wierzchołek już był używany, to robimy wniosek, że graf ma cykl, więc nie jest drzewem, i wtedy zwracamy -1.
W końcy w metodzie **main()** musimy sprawdzić, czy jest ten graf drzewem i policzyć wagi poddrzew. Na początku, oczywiście, musimy wywołać metodę **getVertex(int m[][])** i przypisać wynik na zmienną **int vertex**. Potem, jeżeli **isTree** nadal jest **true**, musimy wywołać metodę **dfs(int v)**. Ponieważ trzeba policzyć wagi poddrew, musimy wywołać tą metodę dla każdego “syna” najwyższego wierzchołka i dla każdego tego “syna” odrębnie policzyć wagi najcięższego i najłatwiejszego poddrzewa, a potem porównać ich i znaleźć **min** i **max** na całym drzewie. Po zakończeniu tego musimy porównać liczbę krawędzi z liczbą wierzchołków zmiejszoną o jeden, i jeżeli te liczby nie są sobie równe, robimy wniosek, że ten graf nie jest drzewem, a jeżeli są, wtedy wszystko jest OK.

II. Oszacowanie złożoności algorytmu:

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

1) Koszt pesymistyczny:

Jeżeli podany graf jest drzewem, metoda **dfs(int v)** będzie wywołana dla każdego jego wierzchołku, czyli n razy. Ponieważ w tej metodzie sprawdzamy tylko to, czy ten wierzchołek juz był przez nas odwiedziony (**visited[v] = true**), jej złożoność będzie równa Θ(1). W najgorszym przypadku (graf jest drzewem) ta metoda wykona się dokładnie n razy,

wiec cała złożoność metody **dfs(int v)** w najgorszym przypadku będzie równa W1(n) = Θ(1) * n = Θ(n).
Jeszcze mamy dwie metody **getVertex(int m[][])** i **getPrev(int m[][], int v)** kiedy szukamy najwyższy wierzchołek tego grafu. W metodzie **getVertex(int m[][])** poszukujemy w petli for pierwszego wierszchołku, z którego nie wychodzą inne krawędzi. Po znaleźeniu takiego wierzchołku przerywamy tą pętle i wywołujemy metodę **getPrev(int m[][], int v)** dla tego wierzchołku. W najgorszym przypadku, taki wierzchołek będzie jeden w całym grafie i będzie on znajdował na samym końcu tej naszej tablicy (czyli ta tablica będzie miała postać mniej więcej taką: m[][] = { {2}, {3}, {4}, {5}, …, {n}, {} }), więc ta pętla w najgorszym przypadku wykona się n razy, czyli złożoność metody **getVertex(int m[][])** w najgorszym przypadku będzie równa się W2(n) = O(n).
W metodzie **getPrev(int m[][], int v)** idziemy po każdej podtablice tablicy **m[][]** i szukamy ideksu takiej, w której znajduje się ten wierzchołek v. I tutaj mamy problem, dlatego że nie możemy wiedzieć, jakie będą długości tych podtablic, ponieważ one są rózne (jeżeli napotykamy się na zero – przerywamy działanie pętli wewnęsznej). Jeżeli graf jest drewem, to suma wszystkich długości podtablic będzie równa dokładnie n-1, gdzie n będzie długościa tablicy **m[][]**. Jeżeli założmy, że mamy tablicę taką, jak w najgorszym przypadku metody **getVertex(int m[][])** (tak naprawdę, jeżeli graf jest drzewem, to długości tych podtablic zawsze będą stałymi liczbami, więc będzie mniej więcej taka sama złożoność tej metody), wtedy złożoność metody **getPrev(int m[][], int v)** z uwzględnieniem rekursji będzie wynosiła W3(n) = (n-1) +
+ (n-2) + … + 1 = ((n-1)* (n-1+1))/2 = O(n^2 /2) = O(n^2).
Więc, cała złożoność mojego programu w najgorszym przypadku będzie wynosiła W(n) = W1(n) + W2(n) + W3(n) =

= Θ(n) + O(n) + O(n^2) = O(n^2).

2)Koszt pamięciowy:
W trakcie działania programu potrzebujemy stworzenia tablic pomocniczych **visited[]** i **used[]**, które mają długości n, więc cała złożoność pamięciowa będzie równa S(n) = O(2n) = O(n).