Prima/Kruskala Damian Kamiński 30.05.2021

makeSet()

Metoda tworzy zbiór jednoelementowy. Zatem jej złożoność czasowa wyniesie **O(1).**

findSet()

Metoda szuka zbiór który zawiera dany element. Przegląda więc wszystkie zbiory. Stąd są 2 pętle, 1 do przejścia przez wszystkie zbiory, 2 do przejrzenia wszystkich elementów w przeglądanym zbiorze w celu zidentyfikowania zbioru. Zatem złożoność czasowa wyniesie **O([1]N*n).**

UNION()

Metoda jest oparta o metodę Set::Union, która łączy dwa wybrane zbiory i zwraca ich sumę. Złożoność metody Set::Union wynosi **O([2]N_o(N1 + N2)).** Złożoność pamięciowa O(N1 + N2), ponieważ tworzę nowy zbiór, a dopiero później go zamieniam ze zbiorem 1 a zbiór 2 usuwam.

sortEdges()

Metoda sortuje krawędzie rosnąco. Złożoność pamięciowa wyniesie **O([3]|E|).** Metoda przejdzie przez wszystkie krawędzie, czyli wykona |E| operacji, a następnie wykona sortowanie krawędzi za pomocą algorytmu quicksort, który wykona się w czasie **O(|E|log|E|).** Stąd łączna złożoność wyniesie **O(|E| + |E|log|E|).**

MST_Kruskal()

Na samym początku tworzę |V| zbiorów 1-elementowych oraz sortuje krawędzie niemalejąco. Wykona się to w czasie **O(|V| + |E| + |E|log|E|)**. Następnie wykonuję pętle, która przechodzi po wszystkich krawędziach do momentu, aż nie otrzymamy minimalnego drzewa rozpinającego (czyli pozostanie jeden zbiór ze wszystkimi wierzchołkami). Wewnątrz pętli sprawdzamy czy zbiory zawierające wierzchołki u, v są rozłączne co zajmie

$$O(N*N1) + O(N*N2) + O(N1*N2)$$

czyli,

$$O(N(N1 + N2) + N1*N2)$$

1 N - ilość zbiorów, n - ilość elementów w danym zbiorze
2 N_o – ilość obecnych już elementów, N1/N2 – ilość elementów w zbiorach
3 Liczba krawędzi

Następnie jeśli są rozłączne to dodaje krawędź bezpieczną do tworzonego drzewa - $O([4]|E|_V)$. Na koniec łącze dwa rozłączne zbiory. Zatem całkowita złożoność wyniesie:

$$O(|V| + |E| + |E|\log|E|) + |E| * (O(N(N1 + N2) + N1*N2) + O(|E|_V) + O(N_o(N1 + N2))$$

$$\mathbf{O(|V| + |E| + |E|log|E| + |E| * (N(N1 + N2) + N1*N2 + N_o(N1 + N2)))}$$

MST_Prim()

Na samym początku tworzę zbiór który zawiera wierzchołek startowy w czasie **O(1)**. Następnie do kolejki wrzucam krawędzie wychodzące z tego wierzchołka w czasie $\mathbf{O([5]|E_{aj}|)}$. Następnie rozpoczyna się główna pętla algorytmu który będzie trwał aż do opróżnienia kolejki czyli wykona ona |E| razy (tę liczbę można zmniejszyć poprzez dodanie warunku, że jeśli wielkość zbioru będzie równa ilość wierzchołków należy przerwać pętle). W każdym wywołaniu pętli wykonuje usunięcie z kolejki minimalnego elementu w czasie $\mathbf{O(log^{[6]}n_o)}$. Następnie sprawdzam czy dany wierzchołek z którym należy się połączyć krawędzią o danej wadze nie znajduje się już w tworzonym drzewie co zajmuje $\mathbf{O([7]n).}$ Jeśli nie to dodaje taki wierzchołek do zbioru **O(n)**, krawędź do tworzonego drzewa **O(1)** , i kolejne krawędzie do kolejki które wychodzą z badanego wierzchołka $\mathbf{O(|E_{aj}| * (n + logn_o))}$. Łącznie daje nam to:

$$O(|E_{aj|}) + |E| * (O(logn_o) + O(n) + O(n) + O(|E_{aj}| * (n + logn_o))$$

$$\mathbf{O(|E|*log|E|)}$$

Ponieważ n maksymalnie będzie wynosić |V|, a $n_o = |E|$.

Złożoność pamięciowa będzie wynosić O(|E| + |V|), bo korzystamy z kolejki (krawędzie), zbioru (wierzchołki), i drzewo (|E|).

4 Ilość połączeń z danym wierzchołkiem
5 Ilość krawędzi łączących się z danym wierzchołkiem
6 Ilośc elementów w kolejce
7 Ilośc elementów w zbiorze