Zadanie 4 lista 3

Jakub Kuciński, prowadzący Szymon Dudycz

26 czerwca 2020

Spis treści

1 Treść zadania

(P) Dane jest nieukorzenione drzewo z naturalnymi wagami na krawędziach oraz liczba naturalna C. Ułóż algorytm obliczający, ile jest par wierzchołków odległych od siebie o C.

2 Idea rozwiązania

Aby obliczyć liczbę par wierzchołków odległych od siebie o C można dla każdego wierzchołka obliczyć liczbę wierzchołków odległych od niego o C, następnie zsumować otrzymane wyniki i podzielić przez dwa – każda para zostanie policzona dwa raz, raz dla każdego wierzchołka z pary. Skoro graf jest drzewem, to istnieje dokładnie jedna droga między dowolnymi dwoma wierzchołkami. Wiemy zatem, że jeśli będziemy przechodzili drzewo za pomocą np. DFS'a i zapamiętywali odległość w jakiej jesteśmy od początkowego wierzchołka, to odwiedzając dany wierzchołek wiemy w jakiej jest odległości od początkowego (bo istnieje tylko jedna droga). Jeśli uznajemy 0 za liczbę naturalną to osobno musimy rozpatrzyć przypadek $C = 0$, bo wówczas każdy wierzchołek jest w odległości 0 do samego siebie i każda para (v, v) jest rozpatrywana dokładnie raz.

3 Algorytm

Algorithm 1 $DFS(v, d)$

```
1: k ← 0
2: if d = C then
3:     k ← k + 1
4: for u ∈ N(v) do
5:     if u nieodwiedzony then
6:         oznacz u jako odwiedzony
7:         if d + d(v, u) ⩽ C then
8:             k ← k + DFS(u, d + d(v, u))
   return k
```

Algorithm 2 *Algorytm*

```
1: p ← 0
2: for v ∈ V(G) do
3:     oznacz wszystkie wierzchołki jako nieodwiedzone
4:     oznacz v jako odwiedzony
5:     p ← p + DFS(v, 0)
6: if C = 0 then
7:     p ← p + n(G)
   return p/2
```

4 Złożoność obliczeniowa i pamięciowa

4.1 Złożoność pamięciowa

Graf możemy przechowywać w postaci listy sąsiedztwa, ale oprócz sąsiadów, możemy dodatkowo trzymać również wagę krawędzi łączącej dane wierzchołki. Wymaga to $\mathcal{O}(n+m)$ pamięci, czyli dla drzewa $\mathcal{O}(n)$. Informacje o odwiedzonych wierzchołkach można trzymać w zwykłej tablicy, co wymaga $\mathcal{O}(n)$ pamięci. Pojedyncze wywołanie $DFS(v, d)$ wymaga stałej ilości pamięci stosu. Mamy jednak wywołania rekurencyjne, więc zużywana pamięć będzie wprost proporcjonalna do głębokości wywołań rekurencyjnych, a skoro mamy n wierzchołków, to będzie ona wynosiła co najwyżej $\mathcal{O}(n)$. Zatem całkowita złożoność pamięciowa algorytmu wynosi $\mathcal{O}(n)$.

4.2 Złożoność obliczeniowa

Dla drzewa złożoność czasowa DFS'a wynosi $\mathcal{O}(n)$. Być może drobnego wyjaśnienia wymaga czas sprawdzenia warunku $d + d(v, u) \leqslant C$. Zauważmy, że iterując się po liście sąsiadów v jesteśmy obecnie w elemencie odpowiadającym wierzchołkowi u, zatem dostęp do wagi związanej z krawędzią (v, u) jest w czasie stałym (waga ta jest wpisana w tym elemencie listy sąsiadów). Zastanówmy się więc, jaka jest złożoność procedury Algorytm. Oznaczenie wszystkich wierzchołków jako nieodwiedzonych wymaga każdorazowo $\mathcal{O}(n)$ operacji. DFS tak samo. Pętla, w której znajdują się te instrukcje wykonuje się tyle razy, ile jest wierzchołków w naszym grafie, czyli n. Zatem wykonanie się wszystkich obrotów pętli wymaga $\mathcal{O}(n^2)$ czasu. Instrukcje wykonywane przed i po pętli wykonują się w czasie stałym. Stąd ogólna złożoność obliczeniowa algorytmu wynosi $\mathcal{O}(n^2)$.