Lista 01 zadanie 04

Wiktor Hamberger
308982

2 kwietnia 2020

1 Treść

Niech u i v będą dwoma wierzchołkami w grafie nieskierowanym $G = (V, E; c)$, gdzie c : E $\rightarrow R_+$ jest funkcją wagową. Mówimy, że droga $u = u_1, u_2, ..., u_{k-1}, u_k = v$ z u do v jest sensowna, jeśli dla każdego $i = 2, ..., k$ istnieje droga z u_i do v krótsza od każdej drogi z u_{i-1} do v (przez długość drogi rozumiemy sumę wag jej krawędzi).

Ułóż algorytm, który dla danego G oraz wierzchołków u i v wyznaczy liczbę sensownych dróg z u do v.

2 Idea rozwiązania

Rozwiązanie opiera się na spostrzeżeniu, że liczba moich (danego wierzchołka) sensownych dróg do v jest równa sumie sensownych dróg do v moich dzieci, będących bliżej v t.j. mających krótszą najkrótszą drogę do v niż ja. To powoduje podzielenie rozwiązania na dwie części. Pierwsza to obliczenie najkrótszej odległości od każdego wierzchołka do wierzchołka v. Jako, że jest to graf nieskierowany, możemy policzyć odległości od wierzchołka v do każdego innego i wyjdzie na to samo. Z tym zadaniem poradzi sobie algorytm dijkstry. Druga część to zliczenie sensownych dróg odpowiednich dzieci wierzchołka u. Defnicja użyta w pierwszym zdaniu tej sekcji nadaje się do stworzenia rekurencji t.j. wierczhołek u sumuje wyniki swoich dzieci, których wynik bierze się z sumowania wyników ich dzieci itd.. Z tym zadaniem poradzi sobie lekko zmodyfikowany algorytm dfs, który po wejściu do danego wierzchołka, dla każdego jego dziecka leżącego "bliżej"v, policzy ilość jego sensownych dróg, a następnie, przy wychodzeniu z wierzchołka, zsumuje wyniki dzieci.

3 Pseudokod i złożoność

Data:
G - graf,
u, v - jak w zadaniu,
G[x][y] - 0 jeżeli nie istnieje krawędź z x do y, waga krawędzi w p.p.,
Wyn[x] - ilość sensownych dróg do v z x, tablica początkowo wypełniona zerami,
dist[] - tablica wypełniana przez dijkstrę, zapamiętująca koszt dojścia z danego wierzchołka do v;
Result: Wyn[u] - ilość sensownych dróg z u do v;

```
Function fun(int u, int v) is
    Wyn[v]:=1;
    dijkstra(v);
    mydfs(u);
    return Wyn[u];
end
Function mydfs(int x) is
    for i:=0 to G[x].size() do
        if dist[G[x][i]]<dist[x] then
            if Wyn[G[x][i]]==0 then
                mydfs(G[x][i]);
            end
            Wyn[x]+=Wyn[G[x][i]];
        end
    end
end
```

Złożoność tego algorytmu zależy od liczby krawędzi E oraz liczby wierzchołków V. W algorytmie używamy algorytmu Dijkstry $O(E * log(V))$ oraz DFS $O(E + V)$, jako że używamy ich jeden po drugim to sumaryczna złożoność czasowa będzie rzędu $O(E * log(V) + E + V)$.

4 Dowód poprawności

Trzeba udowodnić dwie rzeczy. Po pierwsze trzeba pokazać, że jeżeli istnieje jakaś sensowana droga z u do v, to ten alorytm ją rozpatrzy. Weźmy drogę $u = u_1, u_2, \ldots, u_k = v$. Jeżeli ta ścieżka jest sensowna to dla każdego $i = 2, \ldots, k$ $dist[u_i] < dist[u_{i-1}]$. Skoro tak jest to funkcja *mydfs* doliczy do sumy $Wyn[u_{i-1}]$ $Wyn[u_i]$, więc utworzy ścieżkę pomiędzy każdym $i - 1$ i i. Co oznacza że utworzy drogę $u = u_1, u_2, \ldots, u_k = v$. Teraz pozostaje sprawdzić, czy algorytm nie policzy za dużo ścieżek. Znowu mamy dwie opcje. Po pierwsze jeżeli droga nie jest sensowna to na pewno nie zostanie wzięta pod uwagę funkcja $mydfs$ w pewnym momencie nie przejdzie z ojca do syna będącego dalej od wierzchołka v, co spowoduje że cała droga również nie będzie istnieć. Pozostaje sprawdzić, czy algorytm nie liczy niektórych dróg wielokrotnie. Zauważmy, że w trakcie działania programu, każdy wierzchorzek odwiedzony jest dokładnie raz. To oznacza, że jeżeli miałaby istnieć identyczne drogi zawierające dany wierzchołek, to musiałby zoastać one policzone w trakcie przetwrzania danego wierzchołka przez funkcję $mydfs$. Jednak każdego z sąsiadów danego wierzchołka też rozpatrujemy dokłanie raz. I tak indukcyjnie aż do samego wierzchołka v, którego wartość ustawiamy ręcznie na 1. Tak więc żadna droga nie zostanie policzona więcej niż jeden raz.