Lista VIII Z: 3

1 Treść:

Często w grafie występuje kilka najkrótszych ścieżek (czyli o tej samej długości) między dwoma wierzchołkami. Pokaz algorytm o złożoności liniowej dla następującego zadania:
Input: Nieskierowany graf G = (V, E) gdzie każda krawędź ma długość 1, węzły u, v $\in$ V.
Output: Liczba różnych najkrótszych ścieżek z u do v.

2 Rozwiązanie:

W rozwiązaniu spróbujmy użyć lekko zmodyfikowanego algorytmu BFS (przeszukiwania wszerz) pozwalającego na wyznaczenie najkrótszej ścieżki między wierzchołkami. Będziemy chcieli żeby nasz algorytm dla każdego odwiedzanego wierzchołka liczył ilość sposobów na który możemy się do niego dostać (liczymy jedynie najkrótsze ścieżki). Nasz algorytm będzie bazował na tym, że BFS odwiedza wierzchołki poziomami - tzn. najpierw odwiedzamy wierzchołki których odległość od startu wynosi 1, potem 2 itd. Nasz zmodyfikowany BFS zamiast wyznaczać poprzedników będzie przypisywał wierzchołkom parametr paths mówiący ile jest najkrótszych ścieżek prowadzących do danego wierzchołka od startu. Zauważmy, że jeśli mamy już dla każdego wierzchołka odległego od startu o 1 wyznaczony parametr paths to możemy wyznaczyć paths dla dowolnego wierzchołka x odległego od startu o 2. wystarczy zsumować parametry paths wierzchołków o odległości 1 z których istnieje krawędź do wierzchołka x - w ten sposób wyznaczymy wszystkie możliwe najkrótsze ścieżki do x (inne krawędzie prowadzące do x pochodzą z wierzchołków których odległość od startu jest $\geqslant$ niż 2 - żadna najkrótsza droga nie może przez nie przechodzić bo miałaby długość przynajmniej 3). W podobny sposób jeśli mamy już wyznaczone paths dla wierzchołków odległych od startu o n, możemy wyznaczyć paths dla wierzchołków odległych o n + 1. Jeszcze jedna drobna uwaga - spróbujemy podać pseudokod algorytmu wyznaczającego paths dla wszystkich wierzchołków grafu, chociaż zgodnie z treścią wystarczyłby nam wyznaczenie liczby najkrótszych ścieżek pomiędzy dwoma konkretnymi wierzchołkami. Taki bardziej ogólny algorytm będzie nam troszeczkę łatwiej napisać a jego koszt - jak spróbujemy pokazać - też będzie liniowy. Podajmy pseudokod:

```
BFS-COUNT-PATHS(G, s)
for all v ∈ V
    dist(v) = ∞
    paths(v) = 0
dist(s) = 0
s.paths = 1 // do wierzchołka startowego możemy dostać się na jeden sposób
Q = [s] // Inicjujemy kolejkę Q
while Q is not empty
    u = Q.eject()
    for all edges (u, v) ∈ E
        if dist(v) == ∞
            Q.inject(v)
```

```
12:         dist(v) = dist(u) + 1
13:         v.paths = u.paths // pierwsze najkrótsze ścieżki jakie znaleźliśmy prowadzą z u
14:      else if dist(v) = dist(u) + 1 // czy przechodząc przez u będziemy mieli jedną z najkrótszych ścieżek do v
15:         v.paths = v.paths + u.paths // jeśli tak aktualizujemy paths
```

Koszt takiego lekko zmodyfikowanego algorytmu BFS dalej będzie O(n + m) - gdzie n to liczba wierzchołków grafu a m liczba krawędzi. (Łączny koszt wywołań linijek 10-15 jest O(1) a w całym algorytmie wykonujemy je najwyżej raz dla każdej krawędzi grafu - stąd mamy w koszcie czynnik O(m), podobnie łączny koszt wywołań linijki 9 jest O(m) - BFS przechodzi przez każdą krawędź tylko jeden raz, dodatkowo koszt linijek 7-8 jest O(n) - w kolejce każdy wierzchołek może znaleźć się najwyżej raz a linijka 8 zmniejsza rozmiar kolejki o 1 dodatkowo koszt wykonywanych na początku linijek 4-5 to O(1) a 1-3 to O(n), zsumowanie tych częściowych kosztów daje nam O(n+m) - bardziej formalne szacowanie kosztu możemy uzyskać powtarzając analizę algorytmu BFS i lekko ją modyfikując). Jeszcze jedna uwaga techniczna - zakładamy że graf przechowujemy korzystając z list sąsiedztwa, gdybyśmy używali reprezentacji macierzowej koszt linijki 9 mógłby być O(n) i łączny koszt algorytmu wynosiłby O(n^2)