1 Lista 8, Zadanie 6

1.1 Wstęp

W poleceniu zadania 6 nie jest to jasno zaznaczone, ale chodzi o to: dla każdej pary wierzchołków $u, v \in V$, niech $m_{u,v}$ będzie liczbą krawędzi na najmniejszej (w sensie liczby krawędzi) z najkrótszych ścieżek pomiędzy u, v. Niech k będzie największą ze wszystkich $m_{u,v}$. Pondato nasz algorytm ma mieć złożoność $O(k \cdot |E|)$, innymi słowy ma wykonać się k razy i zatrzymać. Zwróćmy uwagę, że kolejne wykonania procedury są bezcelowe, ponieważ, kiedy obliczyliśmy najdłuższą z najkrótszych tras to już nic nie pozostało do policzenia. Łatwiej będzie czytać ten wywód kiedy "najkrótszą trasę" uznamy za cechę dwóch wierzchołków, a nie epitet.
Użyję do tego zmodyfikowanego algorytmu Bellmana-Forda ze "Wstępu do Algorytmów" Cormena.

1.2 Algorytm Bellmana-Forda Przerobiony

```
counter = 0
for v ∈ V do
    v.dist = ∞
    v.prev = null
end
s.dist = 0
change = true
while change == true do
    change = false
    for each (u,v) ∈ E do
        if v.dist > u.dist + l(u,v) then
            v.dist = u.dist + l(u,v)
            v.prev = u
            change = true
        end
    end
    counter = counter + 1
end
```

1.3 Analiza

Powyższy algorytm, daje nam dwie potrzebne rzeczy. Po pierszwe mamy *counter*, który jest równy $k+1$. Oraz mamy jednoznacznie wyznaczoną najkrótszą trasę pomiędzy dowolnymi $u, v \in V$. Aby znaleźć te trasę wsytarczy od tyłu (to zanczy od v) rekurencyjnie przejść po atrybucie *prev*.
U Cormena występuje test, na cykle, których sumaryczna waga jest ujemna. W tym rozwiązaniu zakładam, że graf takich cykli nie ma. W przeciwnym razie procedura Bellmana-Forda Przerobiony niegdy by się nie zatrzymała.

1.4 Złożoność

k jest górnym ograniczeniem, na długość najkrótszych tras w naszym grafie. Jako, że algorytm wykonuje się tak długo aż nie ustali najkrótszej trasy dla wierzchołków dla których trasa będzie najdłuższa, przestanie się wykonywać po $k+1$ iteracjach. (k + jedna iteracja, aby sprzawdzić, że nic się nie zmieniło). Pętla $foreach(u, v) \in E$ ma rzecz jasna złożoność $O(|E|)$, a podczas wykonywania całej procedury wywoływana jest $k+1$ razy. Tym samym złożoność algorytmu wynosi $O((k+1) \cdot |E|) = O(k \cdot |E|)$.