Algorytmy do zestawu nr 4[1]

1. Algorytm Kosaraju

Algorytm Kosaraju, oznaczający silnie spójne składowe w grafie skierowanym, polega na dwóch przeszukiwaniach w głąb: w liniach 1–10 oraz 12–22.

- $d[v]$ – czas odwiedzenia wierzchołka v.
- $f[v]$ – czas przetworzenia wierzchołka v.
- $comp[v]$ – numer silnie spójnej składowej, do której należy wierzchołek v.

Algorytm 1: `kosaraju`(G) ▷ *G – skierowany graf wejściowy*

1: **for** każdy wierzchołek v należący do grafu G **do**
2:     $d[v] \leftarrow -1$ ▷ *Oznaczenie wierzchołka nieodwiedzonego.*
3:     $f[v] \leftarrow -1$
4: **end for**
5: $t \leftarrow 0$
6: **for** każdy wierzchołek v należący do grafu G **do**
7:     **if** $d[v] = -1$ **then**
8:         `DFS_visit`$(\boldsymbol{v}, G, d, f, t)$
9:     **end if**
10: **end for**
11: Utwórz graf G^{T}, będący transpozycją G ▷ *W G^{T} zwroty krawędzi są odwrócone.*
12: $nr \leftarrow 0$ ▷ *Numer silnie spójnej składowej.*
13: **for** każdy wierzchołek v należący do grafu G^{T} **do**
14:     $comp[v] \leftarrow -1$ ▷ *Wszystkie wierzchołki są nieodwiedzone.*
15: **end for**
16: **for** każdy wierzchołek v należący do G^{T} w kolejności malejących czasów $f[v]$ **do**
17:     **if** $comp[v] = -1$ **then**
18:         $nr \leftarrow nr + 1$
19:         $comp[v] \leftarrow nr$
20:         `components_r`$(nr, \boldsymbol{v}, G^{\mathrm{T}}, comp)$
21:     **end if**
22: **end for**
23: **return** $comp$

[1] Na podstawie: Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., *Wprowadzenie do algorytmów*, Wyd. 4, Warszawa, Wydawnictwo Naukowo – Techniczne, 2001, ISBN 83-204-2665-0.

Uwaga: algorytm rekurencyjny `DFS_visit`$(\boldsymbol{v}, G, d, f, t)$ musi mieć bezpośredni dostęp do zmiennej t, żeby móc ją edytować – nie wystarczy przekazanie parametru t przez wartość.

Algorytm 2: `DFS_visit`$(\boldsymbol{v}, G, d, f, t)$

1: $t \leftarrow t + 1$
2: $d[v] \leftarrow t$
3: **for** każdy wierzchołek $u \in G$ będący sąsiadem v **do** ▷ *Przejście po krawędzi* (v, u)
4:     **if** $d[u] = -1$ **then**
5:         `DFS_visit`$(\boldsymbol{u}, G, d, f, t)$
6:     **end if**
7: **end for**
8: $t \leftarrow t + 1$
9: $f[v] \leftarrow t$ ▷ *W implementacji ze stosem: tutaj należy dodać v do stosu.*

Algorytm 3: `components_r`$(nr, \boldsymbol{v}, G^{\mathrm{T}}, comp)$

for każdy wierzchołek $u \in G^{\mathrm{T}}$ będący sąsiadem v **do**
    if $comp[u] = -1$ **then**
        $comp[u] \leftarrow nr$
        `components_r`$(nr, \boldsymbol{u}, G^{\mathrm{T}}, comp)$
    end if
end for

W implementacji ze stosem wierzchołek v należy dodać do stosu w momencie zakończenia jego przetwarzania przez pierwsze przeszukiwanie w głąb, tj. w 9. kroku procedury `DFS_visit`$(\boldsymbol{v}, G, d, f, t)$. Wówczas główna pętla drugiego przeszukiwania w głąb, tj. pętla z 16. kroku procedury `kosaraju`(G), przechodzi po wierzchołkach w kolejności ściągania ich ze stosu.

2. Algorytm Bellmana-Forda

Algorytm 4: `bellman_ford`(G, w, s) ▷ *Graf G ma n wierzchołków. w – macierz wag krawędzi, s – wierzchołek startowy.*

`init`(G, s)
for $i \leftarrow 1$ **to** $n - 1$ **do** ▷ $n - 1$ iteracji.
    for każda krawędź (u, v) należąca do grafu G **do**
        `relax`(u, v, w)
    end for
end for
for każda krawędź (u, v) należąca do grafu G **do**
    if $d_s[v] > d_s[u] + w[u][v]$ **then**
        return FALSE ▷ *W grafie jest cykl o ujemnej wadze osiągalny ze źródła s.*
    end if
end for
return TRUE

3. Algorytm Johnsona

- $d_s[v]$ – długość najkrótszej ścieżki z dodatkowego wierzchołka s do v, obliczona w 2. kroku.
- D – wynikowa macierz odległości w grafie G (długości najkrótszych ścieżek między każdą parą wierzchołków).
- $\widehat{w}$ – macierz z przeskalowanymi wagami (nieujemnymi).
- $\widehat{d_u}[v]$ – długość najkrótszej ścieżki z wierzchołka u do v dla zmodyfikowanych wag $\widehat{w}$.

Algorytm 5: johnson(G, w)

$G' \leftarrow$ add_s(G)
if bellman_ford(G', w, s) = **FALSE then**
    ERROR ▷ *G zawiera cykl o ujemnej wadze.*
else
    for każdy wierzchołek v należący do G' **do**
        $h[v] \leftarrow d_s[v]$ ▷ Tablica odległości d_s pochodzi z algorytmu Bellmana-Forda.
    end for
    for każda krawędź (u, v) należąca do grafu G' **do**
        $\widehat{w}[u][v] \leftarrow w[u][v] + h[u] - h[v]$
    end for
    Utwórz macierz D rozmiaru $n \times n$
    for każdy wierzchołek u należący do G **do**
        dijkstra($G, \widehat{w}, u$) ▷ *Aby obliczyć $\widehat{d_u}[v]$ dla każdego v należącego do G.*
        for każdy wierzchołek v należący do G **do**
            $D[u][v] \leftarrow \widehat{d_u}[v] - h[u] + h[v]$
        end for
    end for
    return D
end if

Algorytm 6: add_s(G)

Utwórz $G' \leftarrow G \cup s$
for każdy wierzchołek v należący do grafu G **do**
    Dodaj krawędź (s, v) do G'
    $w[s][v] \leftarrow 0$
end for
return G'