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.-7. oraz 9.-16.

KOSARAJU(G)

1. `for` każdy wierzchołek v należący do G
2. `do` $d[v] = -1$ *// $d[v]$ - czas odwiedzenia wierzchołka v. $d[v] == -1$ oznacza nieodwiedzony.*
3. $f[v] = -1$ *// $f[v]$ – czas przetworzenia wierzchołka v.*
4. $t = 0$
5. `for` każdy wierzchołek v należący do G
6. `do if` $d[v] == -1$
7. `then` DFS_VISIT($\boldsymbol{v}, G, d, f, t$)
8. Utwórz graf G^{T}, będący transpozycją grafu G *// W G^{T} zwroty krawędzi są odwrócone.*
9. $nr = 0$ *// nr – numer spójnej składowej.*
10. `for` każdy wierzchołek v należący do grafu G^{T}
11. `do` $comp[v] = -1$ *// Wszystkie wierzchołki są nieodwiedzone.*
12. `for` każdy wierzchołek v należący do grafu G^{T} **w kolejności malejących czasów** $f[v]$
13. `do if` $comp[v] == -1$
14. `then` $nr = nr + 1$
15. $comp[v] = nr$
16. COMPONENTS_R($nr, \boldsymbol{v}, G^{\mathrm{T}}, comp$)
17. `return` $comp$

DFS_VISIT($\boldsymbol{v}, G, d, f, t$)

1. $t = t + 1$
2. $d[v] = t$
3. `for` każdy wierzchołek $u \in G$ będący sąsiadem v *// Przejście po krawędzi (v, u)*
4. `do if` $d[u] == -1$
5. `then` DFS_VISIT($\boldsymbol{u}, G, d, f, t$)
6. $t = t + 1$ *// Zwiększona wartość t musi być zapamiętana – można np. przekazywać t przez referencję.*
7. $f[v] = t$ *// W implementacji ze stosem: tutaj należy dodać v do stosu.*

---

[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.

COMPONENTS_R($nr, \boldsymbol{v}, G^{\mathrm{T}}, comp$)
1. `for` każdy wierzchołek $u \in G^{\mathrm{T}}$ będący sąsiadem v
2.   `do if` $comp[u] == -1$
3.   `then` $comp[u] = nr$
4.   COMPONENTS_R($nr, \boldsymbol{u}, G^{\mathrm{T}}, comp$)

W implementacji ze stosem wierzchołek v należy dodać do stosu w momencie zakończenia jego przetwarzania przez pierwsze przeszukiwanie w głąb, tj. w 7. kroku procedury DFS_VISIT($\boldsymbol{v}, G, d, f, t$). Wówczas główna pętla drugiego przeszukiwania w głąb, tj. pętla z 12. kroku procedury KOSARAJU(G), przechodzi po wierzchołkach w kolejności ściągania ich ze stosu.

2. Algorytm Bellmana-Forda

BELLMAN_FORD(G, w, s) // n – liczba wierzchołków
1. INIT(G, s)
2. `for` $i = 1$ `to` $n - 1$
3.   `do for` każda krawędź (u, v) należąca do grafu G
4.   `do` RELAX(u, v, w)
5. `for` każda krawędź (u, v) należąca do grafu G
6.   `do if` $d_s(v) > d_s(u) + w(u, v)$
7.   `then return` FALSE   *// W grafie jest cykl o ujemnej wadze osiągalny ze źródła s*
8. `return` TRUE

3. Algorytm Johnsona

JOHNSON(G, w)   // n – liczba wierzchołków grafu G
1. G' =ADD_S(G)
2. `if` BELLMAN_FORD(G', w, s)==FALSE
3.   `then ERROR`   *// G zawiera cykl o ujemnej wadze*
4.   `else for` każdy wierzchołek v należący do G'
5.   `do` $h(v) = d_s(v)$ *// $d_s(v)$ - długość najkrótszej ścieżki $s \to v$ obliczona w 2. kroku*
6.   `for` każda krawędź (u, v) należąca do grafu G'
7.   `do` $\widehat{w}(u, v) = w(u, v) + h(u) - h(v)$
8.   Utwórz macierz $\mathbf{D}$ rozmiaru $n \times n$, gdzie $D_{u,v}$ to długość najkrótszej ścieżki $u \to v$ w G
9.   `for` każdy wierzchołek u należący do G
10.   `do` DIJKSTRA($G, \widehat{w}, u$)   *// aby obliczyć $\widehat{d_u}(v)$ dla każdego v należącego do G*
11.   `for` każdy wierzchołek v należący do G
12.   `do` $D_{u,v} = \widehat{d_u}(v) - h(u) + h(v)$
13.   `return` $\mathbf{D}$

ADD_S(G) // n – liczba wierzchołków grafu G

Utwórz $G' = G \cup s$

for każdy wierzchołek v należący do G

  do Dodaj krawędź (s, v) do G'

    $w(s, v) = 0$

return G'