Algorytmy grafowe cz. 1

DFS, przykłady zastosowań (depth-first search – przegląd w głąb)

G=(V, E) - graf skierowany lub nieskierowany
$|V| = n$, $|E| = m$

Reprezentacja: listy sąsiedztwa

- tablica Adj[1..n] (nagłówków) list
- Adj[i] = lista wszystkich j, takich że (i,j) jest krawędzią.
- kolejność na listach dowolna.
- dla grafu nieskierowanego każda krawędź (i, j) reprezentowana jest w Adj[i] oraz w Adj[j].

Fakt: Rozmiar pamięci potrzebnej dla list sąsiedztwa wynosi $\Theta(n + m)$.

Wersja I - najprostsza

```
Algorytm DFS(G):
  FOR EACH u ∈ V(G)
     visited[u] ← false
  FOR EACH u ∈ V(G)
     IF not visited[u] THEN
        DFS_visit(u)
END.

procedure DFS_visit(u)
   visited[u] ← true
   preorder_visit(u)      //badanie "na wejściu"
   FOR EACH v ∈ Adj[u]    //badaj krawędź (u,v)
      IF not visited[v] THEN  DFS_visit (v)

   postorder_visit(u)     //badanie "na wyjściu"
END.
```

Wersja II

Obliczane atrybuty:

col[v]=
- white – wierzchołek nieodwiedzony
- grey – odwiedzanie się rozpoczęło, v jest na stosie
- black – odwiedzanie zakończone

p[v] – poprzednik w drzewie rozpinającym (drzewo DFS)
d[v] – moment rozpoczęcia odwiedzania v
f[v] – moment zakończenia odwiedzania v

```
Algorytm DFS (G):
  FOR EACH u ∈ V(G)
     col[u] ← white
     p[u] ← NIL  //poprzednik w drzewie rozpinającym

  time ← 0
  FOR EACH u ∈ V(G)
     IF col[u] = white THEN DFS_visit (u)
END

procedure DFS_Visit (u)
   col[u] ← grey
   d[u] ← ++time       // moment odkrycia u
   FOR EACH v ∈ Adj[u] // badaj krawędź (u,v)
      IF col[v] = white THEN
         p[v] ← u
         DFS_visit (v)
      else
         zależnie od zastosowań, być może puste

   col[u] ← black
   f[u] ← ++time       // moment zakończenia dla u
END.
```

Złożoność: $\Theta(n+m)$

Zastosowania:

- problem osiągalności: dla dowolnych wierzchołków u, v, czy v jest osiągalny z u?

- wyznaczanie spójnych składowych grafu nieskierowanego:

Obliczyć tablicę liczbową ss[u], przy czym ss[u] = ss[v] wtw. gdy u i v są w tej samej składowej

- istnienie cykli

Klasyfikacja krawędzi algorytmem DFS dla grafu skierowanego.
Węzeł aktualny = u, badana krawędź = (u,v).

1. drzewowa (do następnika w drzewie): col[v]=white
2. powrotna (do przodka w drzewie): col[v]=grey
3. wzdłużna (w przód - do potomka w drzewie):
   col[v]=black oraz d[u]<d[v]
4. poprzeczna (do starszego poddrzewa):
   col[v]=black oraz d[u]>d[v]

Dla grafu nieskierowanego tylko dwa rodzaje krawędzi:
1. drzewowa
2. pozostałe (umownie: powrotne)

Fakt.
(1) Graf zawiera cykl wtedy i tylko wtedy gdy w dowolnym drzewie DFS istnieje krawędź wsteczna (w grafie nieskierowanym: wsteczna prowadząca do węzła innego niż poprzednik)

(2) Dla ustalonego drzewa (lasu) DFS każdy cykl w grafie zwiera co najmniej jedną krawędź wsteczną.

Wniosek:
Acykliczność grafu można sprawdzić w czasie
- $\Theta(n+m)$ – graf skierowany
- $\Theta(n)$ – graf nieskierowany

- sortowanie topologiczne grafu skierowanego

Wejście: Graf skierowany acykliczny (czyli DAG)
Wyjście:
uporządkowanie wierzchołków takie, że dla każdej krawędzi (u,v) wierzchołek v występuje później niż u

Zastosowanie: szeregowanie czynności – w produkcji, w obliczeniach zależności funkcyjnych etc.

Rozwiązanie:

- zastosuj DFS,
- podczas kolorowania na czarno wstawiaj na stos
- na końcu wypisz zawartość stosu

Równoważnie:
DFS + wypisanie wierzchołków w kolejności malejących numerów postorder f[u].

- znajdowanie silnie spójnych składowych

Silnie spójna składowa (sss) grafu skierowanego G=(V,E): każdy maksymalny w sensie zawierania podgraf indukowany G'=(V', E') taki, że dla każdej pary wierzchołków u,v w V' istnieje ścieżka z u do v.

Fakt: Wierzchołki u, v są w tej samej sss wtedy i tylko wtedy gdy leżą na cyklu.

Wniosek (uboczny): Jeśli szukamy cyklu (lub listujemy cykle) wystarcza ograniczyć się do każdej sss oddzielnie.

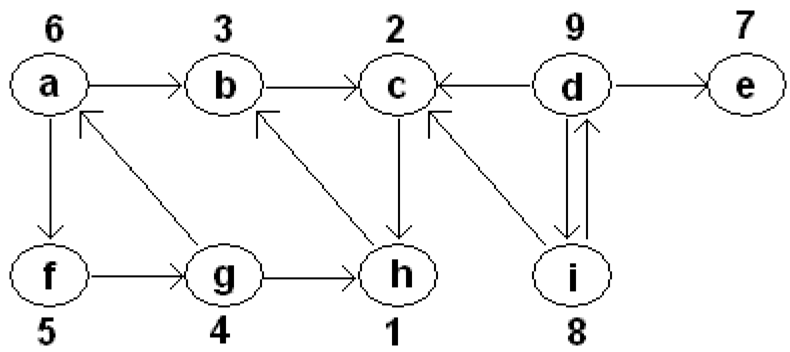
Algorytm silnie spójnych składowych:
We: G=(V,E) – graf skierowany
Wy: Etykiety s(v), dla każdego wierzchołka v, identyfikujące silnie spójną składowa (tzn. s(v)=s(u) wtw. u,v leżą w jednej sss).

1. Algorytmem DFS dla grafu G oblicz czas f(u) (postorder) dla każdego wierzchołka u.

2. Oblicz G^T - graf transponowany (czyli wszystkie krawędzie skierowane w przeciwną stronę)

3. Wykonaj DFS(G^T), wywołując DFS_visit(u) w głównej pętli w kolejności malejących f(u), wyliczonych w kroku 1.

4. Każde drzewo lasu rozpinającego wygenerowane w kroku 3 wypisz jako oddzielną silnie spójną składową.



Przykładowy graf, w wartościami funkcji f(v). Kolejność na listach sąsiedztwa alfabetyczna. Korzenie drzew w kroku 4 i związane z nimi silnie spójne składowe wypisywane są w następującej kolejności:

d : {i, d}
e : {e}
a : {g, f, a}
b : {h, c, b}

Poprawność wynika z następujących obserwacji:

1. Jeśli wierzchołki u, v leżą w jednej sss, to żadna ścieżka z u do v nie opuszcza tej składowej.

2. W dowolnym przeszukiwaniu w głąb wierzchołki jednej sss leżą w tym samym drzewie rozpinającym.

Def. $w=\varphi(u)$ – praojciec wierzchołka u: węzeł w taki, że w osiągalny z u oraz f[w] największy z możliwych.

3. Dla dowolnego wierzchołka u, $\varphi(u)$ jest przodkiem u.

4. Dowolne wierzchołki u, v leżą w tej samej sss wtedy i tylko wtedy gdy mają wspólnego praojca.

5. Niech r – wierzchołek taki, że f[r] największe.

- r jest praojcem dla samego siebie, więc jest praojcem pewnej sss.

- do jego sss należą wierzchołki z których r jest osiągalny (oraz nie jest osiągalny wierzchołek o większym f[] – ale takiego wierzchołka nie ma, bo f[r] jest największe)

- wierzchołki z których r jest osiągalny, to wszystkie wierzchołki osiągalne z r w grafie G^T

- w ten sposób wywołanie DFS(r) w G^T zbuduje drzewo wszystkich węzłów w sss, do której należy r

- indukcyjnie, powtarzanie DFS(u) w punkcie 3 algorytmu dla wierzchołka u o największym f() spośród wierzchołków pozostałych generować będzie kolejne sss.

Złożoność
W kroku 1 kolejne wierzchołki odwiedzane "postorder" dodajemy do listy, w ten sposób unikamy sortowania wg f(v) w punkcie 3.

Zatem: $\Theta(n+m)$

- dwuspójne składowe

G = (V, E) graf nieskierowany, spójny

w $\in$ V
w jest punktem artykulacji (czyli wierzchołkiem rozdzielającym), jeśli dla pewnych dwóch różnych wierzchołków v, u, każda ścieżka łącząca v i u przechodzi przez w; inaczej: G – {v} jest niespójny

e $\in$ E
e jest mostem jeśli G – { e } jest niespójny

Relacja R między krawędziami:
eRf <==>
e=f lub
e i f leżą na jednym cyklu elementarnym

Dwuspójna składowa:

podgraf indukowany w którym:

zbiór krawędzi:
klasa równoważności względem relacji R

zbiór wierzchołków:
wierzchołki incydentne z krawędziami w klasie

Dwuspójna składowa – inaczej:

podgraf indukowany w którym krawędzie są maksymalnym w sensie inkluzji zbiorem takim, że dowolne dwie różne krawędzie leżą na pewnym cyklu elementarnym

Graf dwuspójny:

Def. 1: eRf, dla wszystkich krawędzi e, f

Def. 2: Dla dowolnych różnych wierzchołków u, v istnieją co najmniej dwie rozłączne (wierzchołkowo) ścieżki z u do v.

punkt artykulacji inaczej:

wierzchołek wspólny dla różnych dwuspójnych składowych

most inaczej:

krawędź stanowiąca klasę równoważności względem relacji R

Problem:

znaleźć dwuspójne składowe grafu

postać rozwiązania:

dla każdej krawędzi e etykieta bcc[e] taka, że

bcc[e]=bcc[e'] ⇔
e, e' należą do tej samej dwuspójnej składowej

Własności:

T := drzewo rozpinające generowane przez DFS
r := root(T)

1. r jest punktem artykulacji ⇔
   r ma co najmniej dwa następniki w T

2. $v \neq r$, v jest punktem artykulacji ⇔
   v posiada następnik s taki, że nie istnieje krawędź wsteczna z poddrzewa o korzeniu s do jakiegokolwiek właściwego przodka węzła v

Pomocnicza funkcja, reprezentowana tablicą:

low[u] := min
({ d[u] } $\cup$ { d[w] : (v, w) jest krawędzią powrotną dla pewnego potomka v węzła u i przodka w węzła u})
// każdy węzeł jest własnym przodkiem i potomkiem

czyli:
low[u] = numer preorder najwyższego wierzchołka osiągalnego krawędzią powrotną z poddrzewa o korzeniu u

Fakt. Załóżmy, że $u \neq$ root(T).
u jest punktem artykulacji ⇔
low[v] >= d[u] dla pewnego następnika v wierzchołka u

Algorytm
znajdowania dwuspójnych składowych

notacja:
(u, v) oznacza krawędź nieskierowaną {u, v}

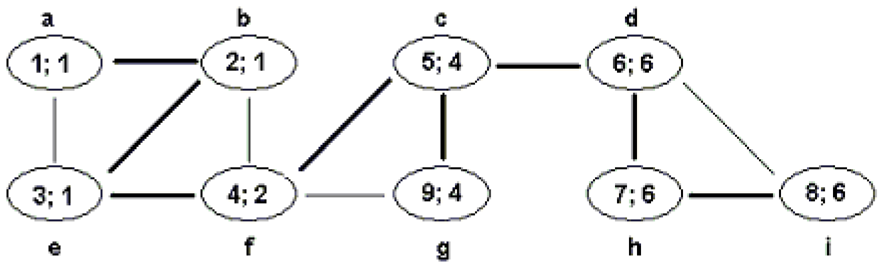
We: G = (V, E) graf nieskierowany, spójny

Wy: dla każdej krawędzi e, bcc[e] = numer dwuspójnej składowej zawierającej krawędź e.

```
Algorytm BIC(G):

FOR EACH u ∈ V(G) color[u] ← white
time ← 0
cur_bcc ← 0    // numer 2składowej
S.create       // pomocniczy stos krawędzi
select u ∈ V(G)
BIC_Visit (u, NULL)
end.
```

```
procedure BIC_Visit (u, p)
// p = rodzic węzła u w drzewie
   color[u] ← red // używamy tylko 2 kolorów
   d[u] ← ++time
   low[u] ← d[u]
   FOR EACH v ∈ Adj[u]        // badaj krawędź (u,v)
         IF color[v] = white THEN
            S.push(<u,v>)
            BIC_Visit (v, u)

            // a po powrocie z rekurencji:

            IF low[v] >= d[u] THEN
            // zdejmij nową 2składową
            // gdy low[v] > d[u] jest to most
               cur_bcc++
               REPEAT
                  <y,z> ← S.pop
                  bcc[<y,z>] ← cur_bcc
               UNTIL <y,z>=<u,v>

            low[u] ← min (low[u], low[v])

         ELSE
            IF d[v] < d[u] && v ≠ p THEN
               // (u,v) jest krawędzią wsteczną
               S.push(<u,v>)
               low[u] ← min (low[u], d[v])

end BIC_Visit
```

Przykład:
Przegląd wierzchołków w pętli głównej oraz na listach sąsiedztwa w kolejności alfabetycznej.



Pogrubione krawędzie należą do drzewa rozpinającego. W każdym węźle wpisano numer d[v] oraz ostateczną wartość low[v].

Kolejno ściągane ze stosu krawędzie drzewa i ich podział na dwuspójne składowe:

Składowa 1: {(i,d), (h,i), (d,h)}
Składowa 2: {(c,d)} (most)
Składowa 3: {(g,f), (c,g), (f,c)}
Składowa 4: {(f,b), (e,f), (e,a), (b,e), (a,b)}.

Złożoność: $\Theta(n+m)$