Tema 5

7 Decembrie 2020

Deadline: 12 ianuarie 2021.

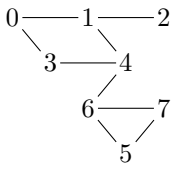
Aplicaţii ale traversării în adâncime

Problema 1 (40 puncte)

Prima problema este de detecţie a nodurilor critice, punţilor, şi componentelor biconexe ale unui graf neorientat conex.

Dacă $G = (V, E)$ este un graf neorientat conex atunci

- Un **nod critic** este un nod $x \in V$ care, decă se elimină din G, rezultă un graf neconex. Spunem că G este **biconex** dacă nu are noduri critice.
- O **punte** este o muchie $e \in E$ care, dacă se elimină din G, rezultă un graf neconex.
- O **componentă biconexă** a lui G este o submulţime maximală de noduri $V \subseteq V$ astfel încât subgraful lui G cu nodurile V' este biconex.

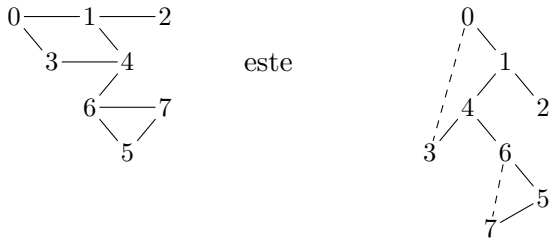
De exemplu, graful conex 

are

- trei noduri critice: 1, 4, 6,
- două punţi: $1-2$ şi $4-6$, şi
- patru componente biconexe: $\{0, 1, 3, 4\}$, $\{1, 2\}$, $\{4, 6\}$ şi $\{5, 6, 7\}$.

Traversarea în adâncime se poate folosi pentru a determina simultan componentele biconexe, nodurile critice şi punţile unui graf neorientat. Complexitatea acestei metode este aceeaşi cu a traversării în adâncime. În cele ce urmează vom descrie cum funcţionează această metodă pentru un graf neorientat conex $G = (V, E)$.

Fie A un arbore DFS al lui G. Reamintim că muchiile lui G care nu sunt muchii în A sunt muchii de întoarcere, adică de forma $y-x$ unde x este un ascendent al lui y diferit de $\mathtt{p}[y]$. De exemplu, un arborele DFS cu muchii de întoarcere al grafului conex



Observăm că

- Un nod x al grafului, diferit de rădăcina lui A, este nod critic dacă şi numai dacă are un descendent direct y în arborele A cu proprietatea că de la niciun descendent al lui y sau de la y nu există muchie de întoarcere la niciun ascendent al lui x;

- Rădăcina lui A este nod critic dacă şi numai dacă are cel puţin doi fii.

Pentru a determina nodurile critice, punţile şi componentele biconexe ale lui G vom calcula pentru fiecare nod x următoarele informaţii:

- $\mathtt{nivel}[x]$: nivelul la care se află nodul x în arborele DFS:

  $\mathtt{nivel}[\text{rădăcină}] = 0,$

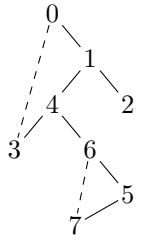
  $\mathtt{nivel}[x] = \mathtt{nivel}[\mathtt{p}[x]] + 1$ dacă $x \neq$ rădăcină.

- $\mathtt{nma}[x]$: nivelul minim la care se poate ajunge din x, mergând numai pe muchii de arbore şi folosind ca ultimă muchie o muchie de întoarcere. Îl vom numi **nivelul minim accesibil** şi este egal cu minimul dintre următoarele 3 valori:

  1. nivelul nodului curent,
  2. minimul dintre nivelurile nodurilor cu care este legat nodul curent printr-o muchie de întoarcere,

3. minimul dintre nivelurile minime accesibile ale fiilor nodului curent din cadrul arborelui DFS.

Pentru nodurile x din arborele DFS A ilustrat anterior cu muchii de întoarcere, aceste valori sunt următoarele:



x	0	1	2	3	4	5	6	7
`nivel`$[x]$	1	2	3	4	3	5	4	6
`nma`$[x]$	1	1	3	1	1	4	4	4

Cu ajutorul valorilor `nivel`$[x]$ şi `nma`$[x]$ putem detecta nodurile critice şi punţile lui G în felul următor:

- Un nod y diferit de rădăcina arborelui DFS este nod critic dacă şi numai dacă are cel puţin un fiu x astfel încât `nivel`$[y] \leq$ `nma`$[x]$.
- Punţile sunt muchiile de forma `p`$[x]-x$ cu x diferit de rădăcina arborelui DFS şi `nivel`$[$`p`$[x]] <$ `nma`$[x]$.

  Muchiile de întoarcere nu sunt niciodată muchii critice fiindcă aparţin întotdeauna unor cicluri.

Valorile tablourilor `nivel` şi `nma` se pot determina în timpul traversării DFS. Fie x nodul curent în traversarea DFS:

- Dacă x este rădăcina, setăm `nivel`$[x] = 1$. În caz contrar, setăm `nivel`$[x] =$ `nivel`$[$`p`$[x]] + 1$.

  Această valoare va rămâne neschimbată până la final.
- Iniţializăm `nma`$[x] =$ `nivel`$[x]$.
- Parcurgem toate nodurile $y \in$ `adj`$[x]$:
  - ▶ Dacă y a fost deja vizitat şi $y \neq$ `p`$[x]$, atunci $x-y$ este muchie de întoarcere. Dacă `nivel`$[y] <$ `nma`$[x]$, actualizăm `nma`$[x] =$ `nivel`$[y]$.
  - ▶ Dacă y nu a fost vizitat, continuăm recursiv cu traversarea în adâncime de la y. La revenirea din apelul recursiv, dacă `nma`$[x] >$ `nma`$[y]$, actualizăm `nma`$[x] =$ `nma`$[y]$.

Traversarea DFS poate fi ajustată să determine, la revenirea din apelul recursiv, nodurile critice, punţile şi componentele biconexe ale lui G.

Pentru determinarea nodurilor critice verificăm următoarele:

1. La revenirea din un apel recursiv pentru un nod $y \in \texttt{adj}[x]$, verificăm dacă $\texttt{nivel}[x] \leq \texttt{nma}[y]$. În caz afirmativ, decidem că x este nod critic.

2. Pentru a stabili dacă rădăcina arborelui A este nod critic, verificăm dacă are cel puţin doi fii. În caz afirmativ, decidem că rădăcina arborelui A este nod critic.

Pentru determinarea punţilor verificăm dacă $\texttt{nivel}[x] < \texttt{nma}[y]$ la revenirea din un apel recursiv pentru un nod $y \in \texttt{adj}[x]$. În caz afirmativ, decidem că $x-y$ este punte.

Pentru determinarea componentelor biconexe folosim o stivă suplimentarš S în care adăugăm noduri în ordinea vizitării lor de către traversarea DFS şi le eliminăm la detecţia unei componente biconexe. Detecţia unei componente biconexe apare când $\texttt{nivel}[x] \leq \texttt{nma}[y]$ la revenirea din un apel recursiv pentru un nod $y \in \texttt{adj}[x]$. În acest caz

- ▶ eliminăm din S toate nodurile până la nodul y, inclusiv. Nodurile eliminate, împreună cu nodul x reprezintă o componentă biconexă. Eliminarea se face până la y, nu până la x, deoarece între acestea, pe stivă, pot fi elemente din altă componentă conexă.

OBSERVAŢIE: condiţia $\texttt{nivel}[x] \leq \texttt{nma}[y]$ are loc întotdeauna când x este rădăcina lui A, chiar dacă x nu este nod critic. În acest cas se determină o componentă biconexă din care face parte rădăcina.

Folosiţi clasele din `algs4.jar` precum şi clasele Java de pe site-ul

`https://staff.fmi.uvt.ro/~mircea.marin/lectures/EduGraph/`

pentru a implementa o clasă Java `BCC` cu următorul API:

`public class BCC`	
`BCC(Graph G)`	Constructor de detecţie a componentelor biconexe şi punţilor lui `G`
`int count()`	Numărul componentelor biconexe
`boolean critical(v)`	Este `v` nod critic?
`Iterable<Edge> bridges()`	Punţile din `G`
`Iterable<Integer> comp(int c)`	Nodurile din componenta biconexă cu identificatorul `c`

Problema 2 (30 puncte)

Problema găsirii unor drumuri elementare de lungime maximă de la un nod sursă într-un graf este următoarea:

Se dau un graf G cu n noduri, m muchii, şi un nod sursă $s \in V(G)$.

Să se găsească drumuri elementare de lungime maximă la $s \overset{\pi}{\rightsquigarrow} x$ pentru toate nodurile din $\{x \in V(G) \mid s \rightsquigarrow x\}$.

Dacă G este DAG, problema se poate rezolva uşor cu ajutorul sortării topologice:

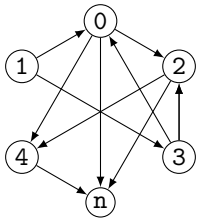
1. Se calculează o sortare topologică $[x_0, x_1, \ldots, x_k]$ a nodurilor la care se poate ajunge din s. Observaţi că $x_0 = s$.
2. Pentru $0 \leq i \leq k$, fie
   - $\texttt{L}[i]$: lungimea celui mai lung drum elementar de s la x_i, şi
   - $\texttt{d}[i]$: o coadă care reţine nodurile unui drum elementar de lungime maximă de la s la x_i.

   Valorile lui $\texttt{L}[i]$ şi $\texttt{d}[i]$ se pot calcula astfel:
   - Se setează valorile iniţiale $\texttt{L}[0] = 0$, $\texttt{d}[0] = [s]$, $\texttt{L}[i] = -\infty$ şi $\texttt{d}[i] = []$ pentru $1 \leq i \leq k$.
   - Se actualizează valorile iniţiale în felul următor:

     for $i = 0$ to k **do**
       for $j \in \texttt{adj}[i]$ **do**
         if $\texttt{L}[j] < \texttt{L}[i] + 1$ **then**
           $\texttt{L}[j] = \texttt{L}[i] + 1$;
           $\texttt{d}[j] = \texttt{d}[i]$ cu nodul j adăugat la sfârşit;
         endif
3. În final, fiecare element $\texttt{d}[i]$ va conţine nodurile unui drum elementar de lungime maximă de la s la x_i.

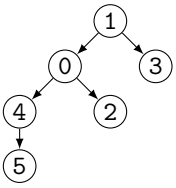
Algoritmul se termină în $k+m+1$ paşi, deci are complexitate liniară $O(n+m)$, unde n este numărul de noduri şi m este numărul de muchii.

Exemplu. Să se calculeze drumuri elementare de lungime maximă de la nodul sursă `c` în digraful



reprezentat cu listele de adiacenţă
$\texttt{adj}[0] = [4, 2, 5], \texttt{adj}[1] = [0, 3], \texttt{adj}[2] = [4, 5],$
$\texttt{adj}[3] = [0, 2], \texttt{adj}[4] = [5], \texttt{adj}[5] = []$.

Arborele de căutare în adâncime cu sursa `1` este

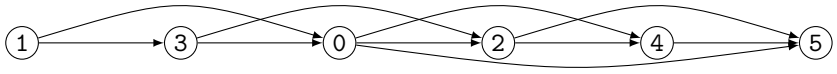


iar traversarea în postordine inversă a acestui arbore produce sortarea topologică $[1, 3, 0, 2, 4, 5]$. În acest exemplu avem $k = 5$.

Valorile iniţiale ale tablourilor `L` şi `d` sunt

$\texttt{L}[0] = 0, \texttt{d}[0] = [\texttt{c}]$, şi
$\texttt{L}[i] = -\infty$ şi $\texttt{d}[i] = []$ pentru $1 \leq i \leq 5$.

Evoluţia valorilor elementelor tabloului `d` este ilustrată mai jos:



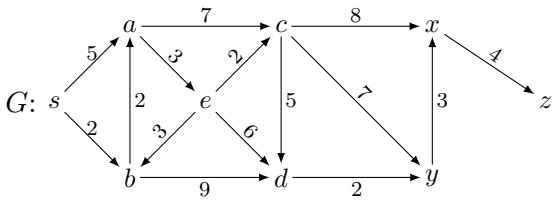
d[0]	d[1]	d[2]	d[3]	d[4]	d[5]
[1]	[]	[]	[]	[]	[]
[1]	[1, 3]	[1, 0]	[]	[]	[]
[1]	[1, 3]	[1, 3, 0]	[1, 3, 2]	[]	[]
[1]	[1, 3]	[1, 3, 0]	[1, 3, 0, 2]	[1, 3, 0, 4]	[1, 3, 0, 5]
[1]	[1, 3]	[1, 3, 0]	[1, 3, 0, 2]	[1, 3, 0, 2, 4]	[1, 3, 0, 2, 5]
[1]	[1, 3]	[1, 3, 0]	[1, 3, 0, 2]	[1, 3, 0, 2, 4]	[1, 3.0, 2, 4, 5]

Să se implementeze în java o clasă `MaxPaths` cu următorul API de detecţie a drumurilor elementare de lungime maximă de la un nod sursă în un DAG:

public class MaxPaths	
`MaxPaths(Digraph G,int s)`	Constructor de detecţie a drumurilor simple de lungime maximă de la nodul sursă `s`
`boolean connected(int x)`	Există drum de la sursă la `x`?
`Iterable<Integer> maxPath(int x)`	Nodurile unui drum elementar de lungime maximă de la sursă la `x`

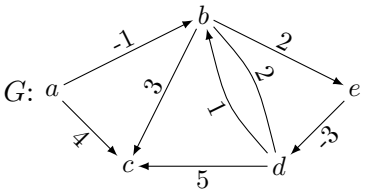
Algoritmii lui Dijkstra, Bellman-Ford şi Floyd-Warshall

1. **(10 puncte)** Fie digraful ponderat



Folosiţi algoritmul lui Dijkstra pentru a calcula căile cu lungime ponderată minimă de la sursa s, şi desenaţi arborele de căi cu lungimi ponderate minime calculat de algoritm.

2. **(20 puncte)** Fie digraful ponderat



   (a) Desenaţi arborele de căi cu lungimi ponderate minime calculat de algoritmul Bellman-Ford de la sursa a.

   (b) Indicaţi tabloul `P`[5] calculat de algoritmul Floyd-Warshall pentru enumerarea de noduri $[a, b, c, d, e]$.

   (c) Indicaţi arborele de căi cu lungimi ponderate minime de la sursa b, determinat de matricea `P`[5].

   (d) Indicaţi arborele de căi cu lungimi ponderate minime de la sursa f, determinat de matricea `P`[5].