2.2 Reprezentatea grafurilor în calculator

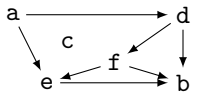
Această secţiune prezintă cele mai populare moduri de reprezentare a grafurilor în clculatoe, şi descrierea unui API Java pe care-l vom folosi pentru lucrul cu grafuri.

2.2.1 Liste de noduri şi liste de muchii

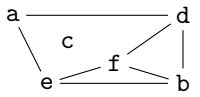
În această reprezentare, un graf G cu n noduri şi m muchii este descris de

1. O listă $V = [x_1, x_2, \ldots, x_n]$ care enumeră toate nodurile grafului,
2. O listă $E = [e_1, e_2, \ldots, e_m]$ care enumeră toate muchiile grafului.

Exemplul 1. Reprezentarea cu liste de noduri şi muchii a grafului



este $V = [\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}]$, $E = [\mathtt{a}{\to}\mathtt{d}, \mathtt{a}{\to}\mathtt{e}, \mathtt{d}{\to}\mathtt{b}, \mathtt{d}{\to}\mathtt{f}, \mathtt{f}{\to}\mathtt{b}, \mathtt{f}{\to}\mathtt{e}]$, iar reprezentarea cu liste de adiacenţă a grafului



este $V = [\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}]$, $E = [\mathtt{a}{-}\mathtt{d}, \mathtt{a}{-}\mathtt{e}, \mathtt{d}{-}\mathtt{b}, \mathtt{d}{-}\mathtt{f}, \mathtt{f}{-}\mathtt{b}, \mathtt{f}{-}\mathtt{e}]$. □

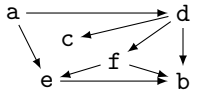
Dacă G nu are noduri izolate atunci îl putem reprezenta doar cu lista de muchii E fiindcă, în acest caz, V se poate calcula din E.

2.2.2 Liste de adiacenţă

În această reprezentare, un graf G cu n noduri şi m muchii este descris de

1. O listă $V = [x_1, x_2, \ldots, x_n]$ care enumeră toate nodurile grafului,
2. Pentru fiecare nod $x_i \in V$, o listă $\mathtt{adj}[x_i]$ a vecinilor lui x_i. Acestea se numesc *listele de adiacenţă* ale lui G.

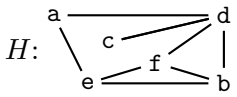
Exemplul 2. Reprezentarea cu liste de adiacenţă a digrafului



este

$$\texttt{adj}[\texttt{a}] = [\texttt{d}, \texttt{e}] \qquad \texttt{adj}[\texttt{c}] = [\,] \qquad \texttt{adj}[\texttt{d}] = [\texttt{c}, \texttt{f}, \texttt{b}]$$
$$\texttt{adj}[\texttt{f}] = [\texttt{b}, \texttt{e}] \qquad \texttt{adj}[\texttt{b}] = [\,] \qquad \texttt{adj}[\texttt{e}] = [\texttt{b}]$$

iar reprezentarea cu liste de adiacenţă a grafului neorientat



este

$$\texttt{adj}[\texttt{a}] = [\texttt{d}, \texttt{e}] \qquad \texttt{adj}[\texttt{c}] = [\texttt{d}] \qquad \texttt{adj}[\texttt{d}] = [\texttt{a}, \texttt{c}, \texttt{f}, \texttt{b}]$$
$$\texttt{adj}[\texttt{f}] = [\texttt{d}, \texttt{b}, \texttt{e}] \qquad \texttt{adj}[\texttt{b}] = [\texttt{d}, \texttt{f}, \texttt{e}] \qquad \texttt{adj}[\texttt{e}] = [\texttt{a}, \texttt{f}, \texttt{b}]$$

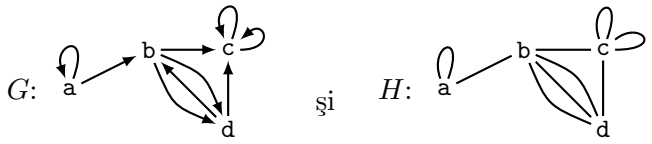
□

Pentru grafuri neorientate, această reprezentare este redundantă: faptul că $a{-}b \in E$ este reţinut de două ori: a este element al listei $\texttt{adj}[b]$, şi b este element al listei $\texttt{adj}[a]$.

2.2.3 Matrice de adiacenţă

Matricea de adiacenţă A_G a unui graf G se defineşte pentru o enumerare $V = [x_1, x_2, \ldots, x_n]$ a nodurilor lui G în felul următor: $A_G = (a_{ij})$ este matricea de dimensiune $n \times n$ unde a_{ij} este numărul de muchii de la x_i la x_j în E.

Exemplul 3. Grafurile



cu lista de noduri $V = [\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}]$ au matricile de adiacenţă

$$A_G = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \text{ şi } A_H = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 0 & 1 & 3 \\ 0 & 1 & 4 & 1 \\ 0 & 3 & 1 & 0 \end{pmatrix}.$$

□

Se observă că, dacă G este graf neorientat atunci A_G este matrice simetrică. Rezultă că, pentru grafuri neorientate, reprezentarea cu matrice de adiacenţă are acelaşi neajuns ca şi reprezentarea cu liste de adiacenţă: reţine de 2 ori informaţia despre muchiile dintre noduri.

Proprietăţi ale matricilor de adiacenţă

În cele ce urmează presupunem implicit că I_n este matricea identitate de dimensiune $n \times n$.

1. Dacă $k > 0$ atunci elementul de la poziţia (i, j) în A_G^k este numărul de drumuri de lungime k de la x_i la x_j în G.

2. Dacă A, B sunt două matrici de dimensiune $n \times n$ atunci $A \cdot B$ se poate calcula în timp $O(n^3)$ iar $A + B$ în timp $O(n^2)$. Rezultă că

   - A_G^k se poate calcula în timp $O((k-1) \cdot n^2)$
   - Matricea $I_n + A + A^2 + \ldots + A^{n-1}$ se poate calcula în timp $O(n^4)$ cu formula $I_n + A \cdot (I_n + \ldots (I_n + A) \ldots)$ care necesită $n-1$ adunări şi $n-2$ înmulţiri.

3. $x_i \rightsquigarrow x_j$ dacă şi numai dacă există un drum elementar de lungime cel mult $n-1$ de la x_i la x_j. Această condiţie este echivalentă cu condiţia

   - elementul de la poziţia (i, j) în matricea $I_n + A_G + \ldots + A_G^{n-1}$ este mai mare ca 0.

Definim matricea $A_G^* := I_n + A_G + \ldots + A_G^{n-1}$. Deoarece A_G^* se poate calcula în timp $O(n^4)$, obţinem un algoritm cu complexitatea $O(n^4)$ pentru problema de decizie "$x_i \rightsquigarrow x_j$?".

Un fapt remarcabil este că putem folosi decide dacă $x_i \rightsquigarrow x_j$ în timp $O(n^3)$. În acest scop, considerăm mulţimea $\mathbf{B}_n$ a matricilor de biţi de dimensiune $n \times n$ împreună cu operaţia $A \odot_k B$ care pentru $A, B \in \mathbf{B}_n$ şi $1 \leq k \leq n$ produce matricea $C = (c_{ij}) \in \mathbf{B}_n$ cu $c_{ij} = \max(a_{i,j}, \min(a_{ik}, b_{kj}))$. Apoi definim matricile recursiv matricile $B_G^{[}k] \in \mathbf{B}_n$ pentru $0 \leq k \leq n$:

- $B_G^{[0]} = (b_{ij})$ cu $b_{ij} = 0$ dacă $a_{ij} = 0$ şi $b_{ij} = 1$ în caz contrar.
- Dacă $k > 0$ atunci $B_G^{[k]} = B_G^{[k-1]} \odot_k B_G^{[k-1]}$.

Se observă că

1. Elementul lui $B_G^{[k]}$ la poziţia (i, j) este 1 dacă şi numai dacă există un drum de la x_i la x_j care trece prin noduri intermediare din mulţimea $\{x_1, x_2, \ldots, x_k\}$.

   ▶ Rezultă că $x_i \rightsquigarrow x_j$ dacă şi numai dacă elementul lui $B_G^{[n]}$ la poziţia (i, j) este 1.

2. $B_G^{[k]}$ se poate calcula în timp $O(k \cdot n^2)$ iar $B_G^{[n]}$ se poate calcula în timp $O(n^3)$.

2.2.4 Matrice de incidenţă

Matricea de incidenţă a unui graf G cu n noduri şi m muchii este o matrice de dimensiune $n \times m$ care se construieşte în felul următor:

1. Fixăm o enumerare $V = [x_1, \ldots, x_n]$ pentru mulţimea de noduri, şi $E = [E_1, \ldots, e_m]$ pentru lista de muchii a lui G.

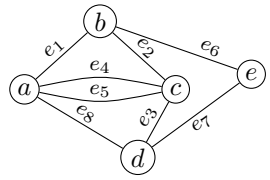
2. Pentru un graf neorientat definim matricea $M_G = (m_{ij})$ de dimensiune $n \times n$ cu

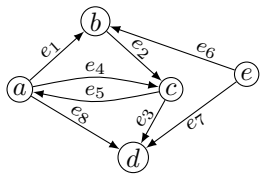
$$m_{ij} = \begin{cases} 1 & \text{dacă } e_j \text{ este incident la nodul } i \\ 0 & \text{în caz contrar.} \end{cases}$$

3. Pentru un graf orientat definim

$$m_{ij} = \begin{cases} 1 & \text{dacă } e_j \text{ are nodul sursă } i, \\ -1 & \text{dacă } e_j \text{ are nodul destinaţie } i, \\ 0 & \text{în celelalte cazuri.} \end{cases}$$

Exemplul 4. Reprezentările cu matrice de incidenţă ale grafurilor

G: 

şi H: 

pentru enumerările $V = [a, b, c, d, e]$ şi $E = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8]$ sunt

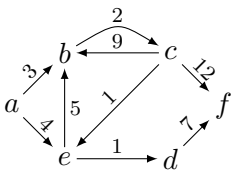
$$M_G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix},$$

$$M_H = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 1 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

2.2.5 Matrice de ponderi

Matricea de ponderi se foloseşte pentru a reprezenta grafuri ponderate simple. Matricea de ponderi a unui graf ponderat simplu (G, w) cu n noduri este o matrice $W_G = (w_{ij})$ de dimensiune $n \times n$ care se defineşte astfel:

1. Fixăm o enumerare $V = [x_1, \ldots, x_n]$ a nodurilor lui G.
2. $w_{ij} = \begin{cases} 0 & \text{dacă } i = j, \\ w(x_i, x_j) & \text{dacă } G \text{ are muchia } x_i{-}x_j \text{ sau } x_i{\to}x_j, \\ +\infty & \text{în celelalte cazuri.} \end{cases}$

Exemplul 5. Matricea de ponderi a grafului G : 

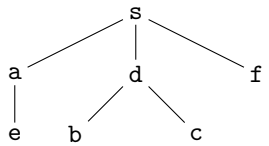
pentru enumerarea $V = [a, b, c, d, e, f]$ este $W_G = \begin{pmatrix} 0 & 3 & \infty & \infty & 4 & \infty \\ \infty & 0 & 2 & \infty & \infty & \infty \\ \infty & 9 & 0 & \infty & 1 & 12 \\ \infty & \infty & \infty & 0 & \infty & 7 \\ \infty & 5 & \infty & 1 & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$. □

2.2.6 Reprezentarea cu predecesori a arborilor cu rădăcină

Reamintim că un arbore este un graf simplu neorientat care este conex şi nu are cicluri. Un **arbore cu rădăcină** este o pereche (T, r) unde $T = (V, E)$ este un arbore iar $r \in V$ este un nod desemnat să fie rădăcina arborelui.

Pentru orice arbore cu rădăcină $((V, E), r)$ putem defini o funcţie $\mathtt{p} : V - \{r\} \to V$ astfel încât $E = \{\mathtt{p}(x){-}x \mid x \in V - \{r\}\}$. Funcţia $\mathtt{p}$ este reprezentarea cu predecesori a arborelui cu rădăcină, iar $\mathtt{p}(x)$ este predecesorul direct al nodului $x \in V - \{s\}$.

Exemplul 6. Reprezentarea cu predecesori a arborelui



cu rădăcina $\mathtt{s}$ este funcţia $\mathtt{p} : \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\} \to \{\mathtt{s}, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$ cu

$$\mathtt{p[a]} = \mathtt{s}, \quad \mathtt{p[d]} = \mathtt{s}, \quad \mathtt{p[f]} = \mathtt{s}, \quad \mathtt{p[e]} = \mathtt{c}, \quad \mathtt{p[b]} = \mathtt{d}, \quad \mathtt{p[c]} = \mathtt{d}.$$

2.2.7 Studiu comparativ

Alegerea modului de reprezentare a unui graf trebuie să ţină cont de operaţiile pe care vrem să le facem cel mai frecvent cu graful respectiv. Dacă G este un graf cu n noduri şi m muchii atunci:

- Reprezentarea cu listă de muchii este adecvată pentru reprezentarea grafurilor simple fără noduri izolate, cu $m \ll n^2$. Complexitatea spaţială (memoria ocupată) a acestei reprezentări este $O(m)$.
- Reprezentarea cu liste de adiacenţă permite enumerarea rapidă a vecinilor unui nod. Complexitatea spaţială (memoria ocupată) a acestei reprezentări este $O(n + m)$.
- Reprezentarea cu matrice de adiacenţă sau cu matrice de ponderi are avantajul că permite un test în timp constant $O(1)$ al conectivităţii directe dintre 2 noduri. Ca dezavantaje menţionăm
  - Complexitatea spaţială (memoria ocupată) a acestei reprezentări este $\Omega(n^2)$. Această reprezentare este neadecvată când $m \ll n^2$.
  - Detecţia şi parcurgerea mulţimii de vecini a unui nod se face în timp $\Omega(n)$ – prea mult pentru grafuri cu număr mare de noduri. Detecţia şi parcurgerea mulţimii de vecini $V(x)$ a unui nod x se face în timp $\Omega(\deg(x))$ şi nu $\Omega(n)$ ca în cazul matricii de adiacenţă.
- Reprezentarea cu matrice de incidenţă M_G are complexitatea temporală $O(n \cdot m)$.

În majoritatea cazurilor, listele de adiacenţă ocupă mult mai puţin spaţiu decât matricea de adiacenţă. De exemplu, un graf simplu cu $n = 10^4$ noduri şi $m = 10^5$ muchii ocupă ≈ 800 KB cu liste de adiacenţă şi ≈ 400 MB cu matrice de adiacenţă. Din acest motiv, structurile de date pentru grafuri implementează cel mai adesea reprezentarea cu liste de adiacenţă.

2.2.8 Un API Java de lucru cu grafuri

Pentru lucrul cu grafuri vom folosi biblioteca java `algs4.jar` oferită ca suport de programare pentru cartea *Algorithms, 4th Edition* de R. Sedgewick şi K. Wayne [13]. Biblioteca poate fi descărcată de pe site-ul

`https://algs4.cs.princeton.edu/`

care oferă instrucţiuni de instalare a acestei bibloteci pe Windows, Linux şi MacOS, precum şi o versiune condensată a cărţii. Secţiunea *1.1 Basic Programming Model* este un ghid concis de familiarizare cu Java şi cu această bibliotecă de clase java.

Material suplimentar pentru lucrul cu grafuri este postat pe site-ul

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

Interfaţa *IGraph* oferă următorul API de lucru cu grafuri:

`public`	`interface` *IGraph*	
`int`	`V()`	Numărul de noduri
`int`	`E()`	Numărul de muchii
`void`	`addEdge(int` i`, int` j`)`	Adaugă muchia i–j la graf
`Iterable<Integer>`	`adj(int` i`)`	Iterator la vecinii nodului i
`String`	`toString()`	Reprezentarea grafului ca şir

Clasele care implementează această interfaţă sunt

- `Graph` pentru grafuri neorientate şi
- `Digraph` pentru grafuri orientate.

Ambele clase implementează o reprezentare cu liste de adiacenţă în care nodurile unui graf cu ordinul n sunt 0, 1, 2, ..., $n-1$. În plus, ele mai au următoarele funcţionalităţi:

`public`	`class Graph`	
	`Graph(In` n`)`	Crează un graf cu n noduri şi nici o muchie
	`Graph(int` in`)`	Citeşte un graf din stream-ul de intrare in
	`Graph(Graph G)`	Crează o copie adâncă a grafului `G`

`public`	`class Digraph`	
	`Digraph(In` n`)`	Crează un digraf cu n noduri şi nici un arc
	`Digraph(int` in`)`	Citeşte un digraf din stream-ul de intrare in
	`Digraph(Digraph G)`	Crează o copie adâncă a digrafului `G`
`Digraph`	`reverse()`	Inversul acestui digraf

Constructorii `Graph(In` in`)` şi `Digraph(In` in`)` citesc un graf cu m muchii de la un stream de intrare format din o secvenţă de $2m+2$ întregi: numărul n de noduri, numărul m de muchii, apoi m perechi de întregi care reprezintă capetele muchiilor grafului.

Dacă obiectul `G` reprezintă un digraf $G = (V, E)$, atunci

```
Digraph RG = G.reverse();
```

crează obiectul `RG` pentru digraful $G^r = (V, E^r)$ a cărui mulţime de arce este $E^r = \{j \to i \mid (i \to j) \in E\}$. Digraful G^r se numeşte **inversul** lui G.

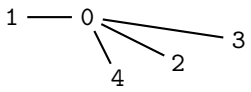
De exemplu, dacă `grafMic.txt` este un fişier text cu conţinutul

```
5
4
0 4
0 1
0 2
3 0
```

atunci comanda

```
Graph G = new Graph(new In("grafMic.txt"));
```

crează o instanţă `G` a clasei `Graph` care reprezintă graful



Obiectul `G` este reprezentarea internă (în memoria calculatorului) a acestui graf, iar fişierul `graficMic.txt` este reprezentarea externă a acestui graf.

Exemplele următoare ilustrează cum poate fi folosit `Graph` în cod client:

- Calculul gradului unui nod:

```
public static int deg(Graph G, int x) {
    int grad = 0;
    for (int y : G.adj(x))
        grad++;
    return grad;
}
```

- Calculul gradului maxim:

```
public static int Delta(Graph G) {
    int max=0;
    for (int x=0; x<G.V(); x++)
        if (deg(G,x)>max)
           max = deg(G,x);
    return max;
}
```

- Calculul numărului de bucle din graf:

```
public static int numarBucle(Graph G) {
   int nr = 0;
   for (int i=0; i<G.V(); i++)
       for (int j : adj(i))
           if (i==j) nr++;
   return nr/2;  // buclele au fost numărate de 2 ori
}
```

2.2.9 Concluzii

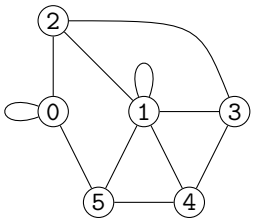
Cele mai populare reprezentări ale grafurilor sunt:

1. liste de noduri şi liste de muchii
2. liste de adiacenţă
3. matrice de adiacenţă
4. matrice de incidenţă
5. matrice de ponderi
6. reprezentarea cu predecesori, pentru arbori cu rădăcină.

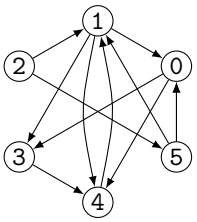
- Problema "$x \rightsquigarrow y$?" într-un graf de ordinul n se poate decide în timp $O(n^3)$ cu metoda descrisă în secţiunea 2.2.3.
- Alegerea modului de reprezentare a unui graf trebuie să ţină cont de nărimea şi ordinul grafului, precum şi de operaţiile care urmează să fie efectuate cel mai frecvent asupra grafului.
- Biblioteca java `algs4.jar` oferă un API simplu de lucrul cu grafuri orientate şi neorientate. Acest API va fi folosit pentru a descrie algoritmii prezentaţi în această lucrare.

2.2.10 Exerciţii

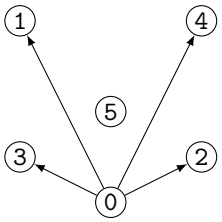
1. Pentru fiecare din grafurile G_1, G_2 şi G_3 de mai jos să se indice:
   (a) O reprezentare cu liste de noduri şi de muchii.
   (b) O reprezentare cu liste de adiacenţă,
   (c) O reprezentare cu matrice de adiacenţă.
   (d) Dacă este graf simplu, pseudograf sau multigraf.



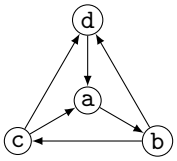
G_1



G_2



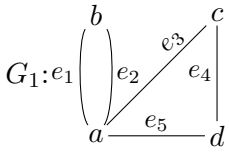
G_3

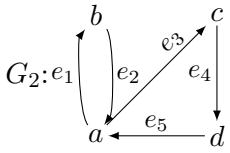
2. Fie digraful G:  cu enumerarea de noduri $[\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}]$.

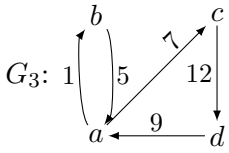
   (a) Indicaţi valorile marticilor $B_G^{[k]}$ pentru $0 \leq k \leq 4$.

   (b) Este G graf conex? Motivaţi răspunsul dat cu ajutorul matricii $B_G^{[4]}$.

3. Se consideră grafurile







   (a) Indicaţi matricile de incidenţă M_{G_1} şi M_{G_2} pentru $V = [a, b, c, d]$ şi $E = [e_1, e_2, e_3, e_4, e_5]$.

   (b) Indicaţi matricea de ponderi W_{G_3} pentru $V = [a, b, c, d]$.

4. Definiţi o clasă `GraphTests` cu urmatoarele metode statice:

   (a) `public static boolean multigraph(Graph G)`
   care returnează `true` dacă `G` este multigraf. Reamintim că un multigraf trebuie să conţină cel puţin o buclă şi cel puţin o muchie cu multiplicitatea mai mare ca 1.

   (b) `public static boolean hamCycle(Digraph G,int[] c)`
   care ia ca argumente de intrare un graf `G` de ordinul n şi un tablou, şi returnează `true` dacă şi numai dacă `c` are $n + 1$ elemente şi $[\mathtt{c}[0], \mathtt{c}[1], \ldots, \mathtt{c}[n]]$ este un ciclu hamiltonian în `G`.

(c) `public static boolean eulerCycle(Graph G,int[] c)`
care ia ca argumente de intrare un graf `G` cu m muchii şi un tablou, şi returnează `true` dacă şi numai dacă `c` are $m+1$ elemente şi $[\texttt{c}[0], \texttt{c}[1], \ldots, \texttt{c}[m]]$ este un ciclu eulerian în `G`.

(d) `public static boolean multimeStabila(Graph G,int[] x)`
care returnează `true` dacă şi numai dacă `x` este un tablou de p numere distincte iar $\{\texttt{x}[0], \texttt{x}[1], \ldots, \texttt{x}[p-1]\}$ este o mulţime stabilă în `G`.

(e) `public static boolean clica(Graph G,int[] x)`
care returnează `true` dacă şi numai dacă `x` este un tablou de p numere distincte care sunt sunt nodurile unei p-clici în `G`.

5. Presupunem că `p` este reprezentarea cu predecesori a unui arbore cu n noduri numerotate de la 0 la $n-1$. Singurul nod x pentru care $\texttt{p}[x]$ este `null` este nodul rădăcină. Definiţi metodele clasei

```
class RootedTree {
   private int[] p; // reprezentarea cu predecesori

   public RootedTree(int [] q) {
      p = new int[q.length];
      p = Arrays.copyOf(q,q.length);
    }
   public int root() { ... }
   public int depth { ... }
   public Iterator<Integer> cale(int x) { ... }
}
```

astfel încât să satisfacă cerinţele următoare:

(a) `root()` returnează nodul rădăcină al arborelui.

(b) `depth()` returnează adâncimea arborelui.

(c) `cale(int x)` returnează o colecţie iterabilă care enumeră toate nodurile de pe ramura arborelui de la rădăcină la x. Sugestie: folosiţi clasa `Stack<E>` din `algs4.jar`.