Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare

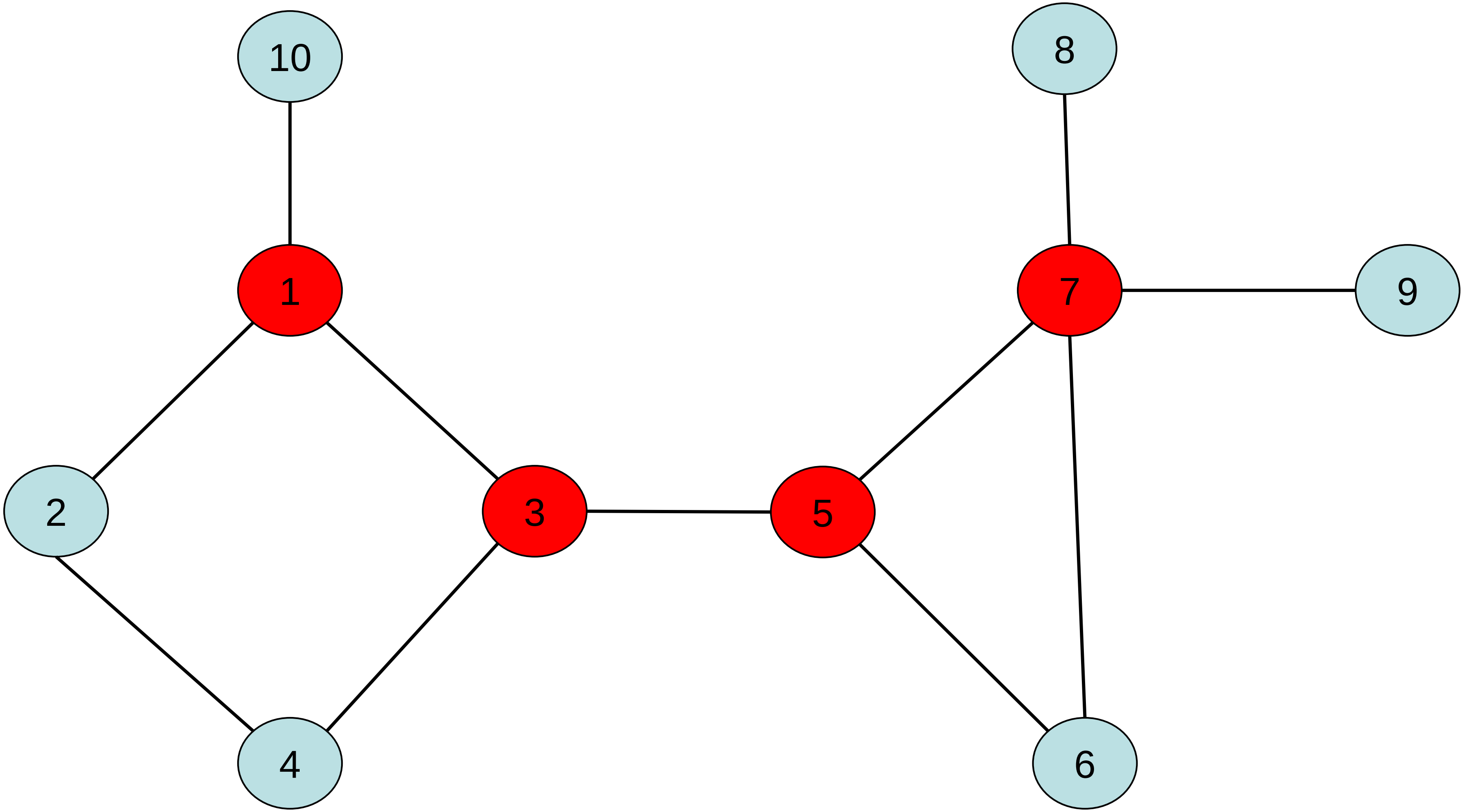
BICONEXITATE

Punct de articulație

- Fie `G=(V,E)` un graf neorientat conex
- Vârful `v∈V` se numește **punct de articulație** dacă subgraful obținut prin eliminarea vârfului `v` și a muchiilor incidente cu acesta nu mai este conex

Exemplu

- Punctele de articulație sunt `1`, `3`, `5`, `7`



Graf biconex

- Un graf se numește **biconex** dacă nu are puncte de articulație

Observații

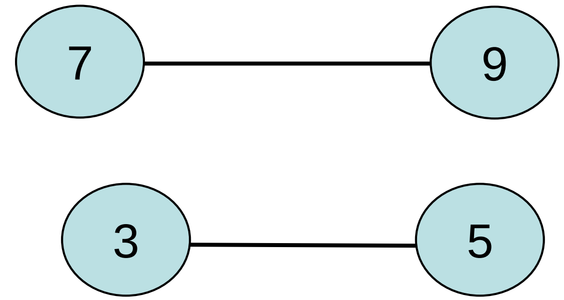
- În multe aplicații practice care se pot modela cu ajutorul grafurilor, nu sunt de dorit punctele de articulație
- Într-o rețea de telecomunicații, dacă o centrală dintr-un punct de articulație se defectează, rezultatul este nu doar întreruperea comunicării cu centrala respectivă, ci și cu alte centrale

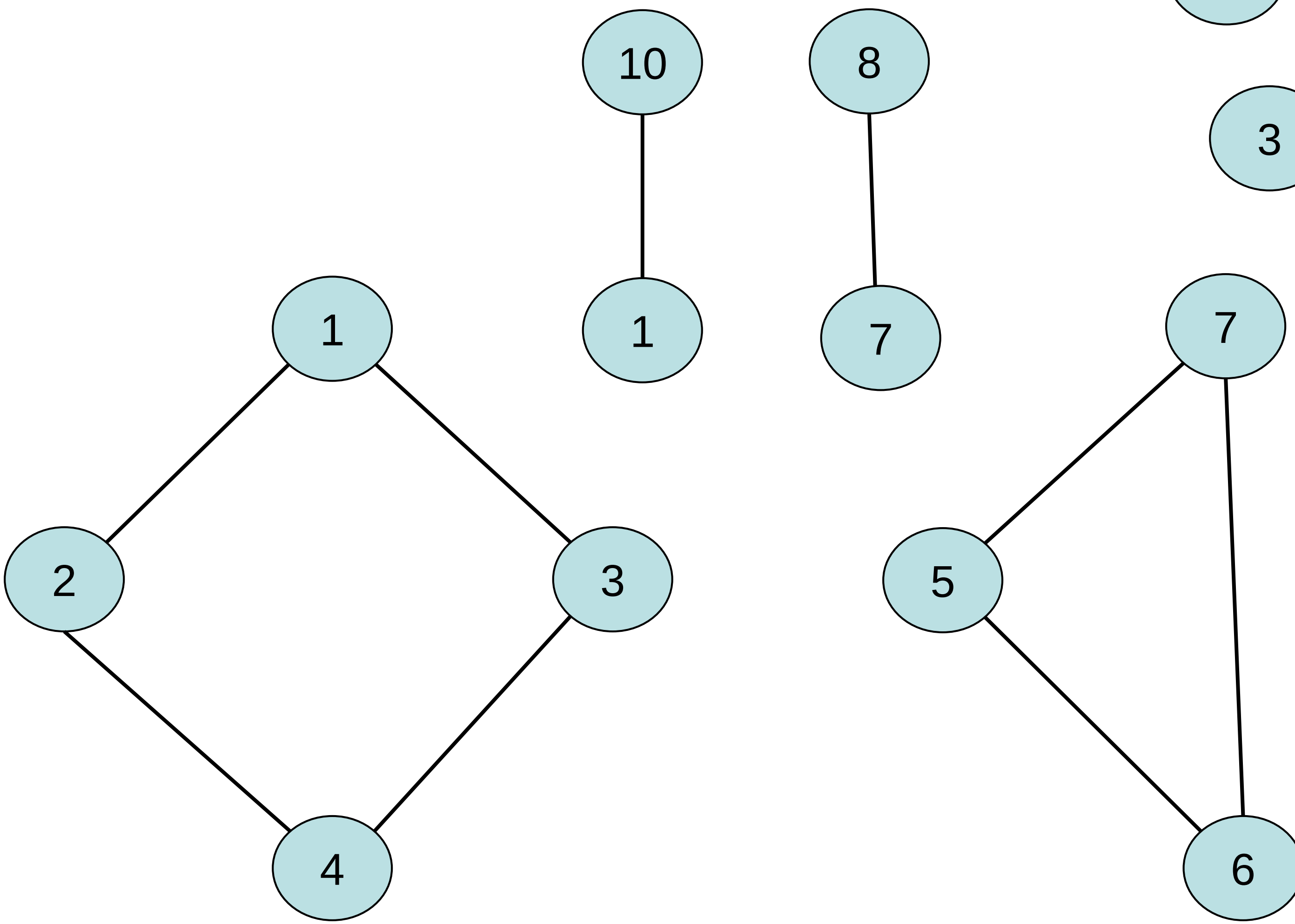
Componentă biconexă

- O **componentă biconexă** a unui graf este un subgraf biconex maximal cu această proprietate

Exemplu

- Pentru graful precedent, componentele biconexe sunt:





Observații

- Componentele biconexe ale unui graf reprezintă o partiție a mulțimii muchiilor grafului
- Punctele de articulație aparțin la cel puțin două componente biconexe

Descompunere în componente biconexe

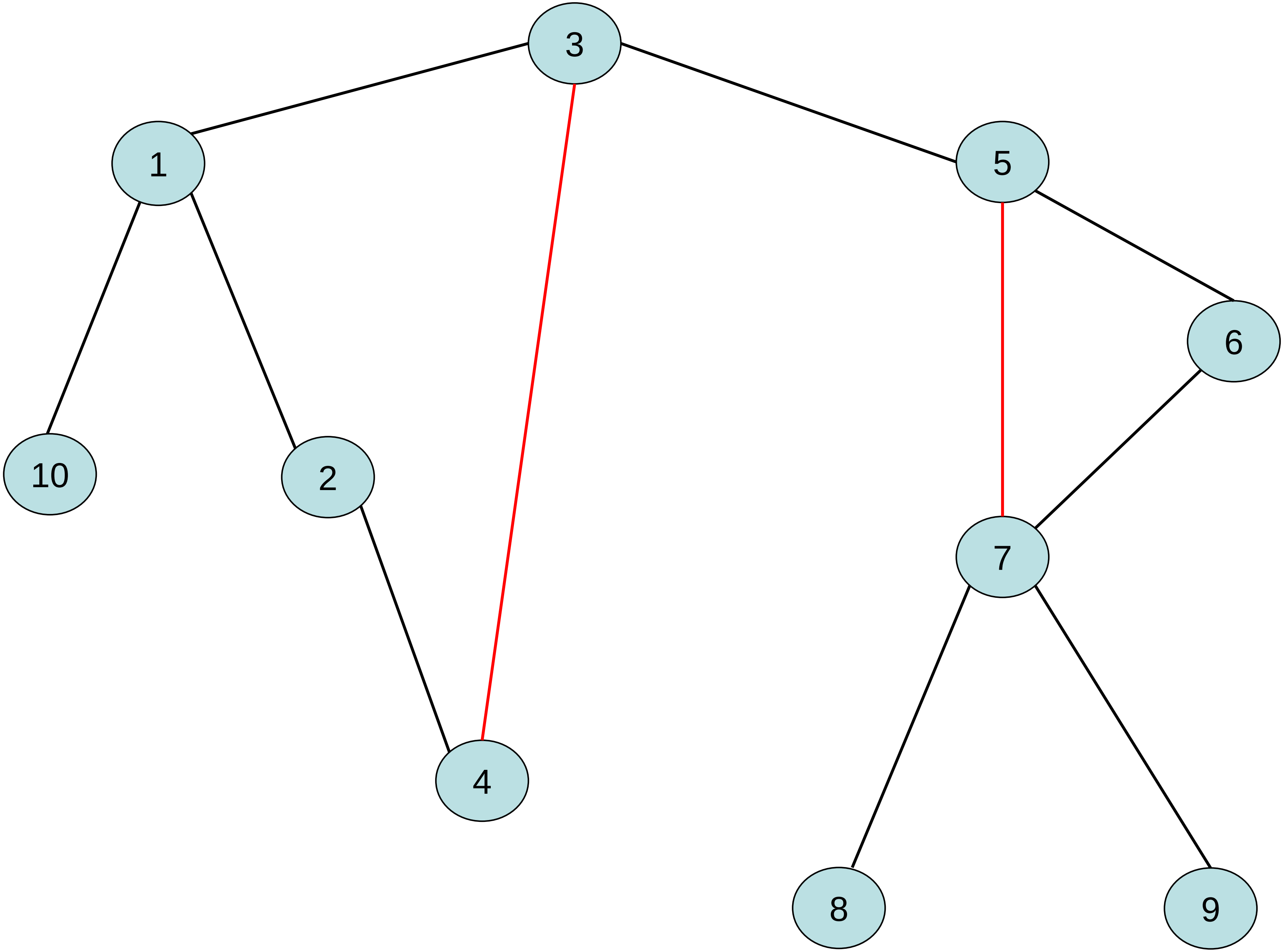
- Pentru a descompune un graf în componente biconexe se utilizează parcurgerea în adâncime

Observații

- Prin parcurgerea grafului în adâncime, se pot clasifica muchiile grafului în:
- **1**. Muchii care aparțin arborelui parțial DFS (*tree edges*)
- **2**. Muchii `(u,v)` care nu aparțin arborelui și care unesc vârful `u` cu un strămoș al său `v` în arborele parțial DFS, numite muchii de revenire (*back edges*)

Exemplu

- Se parcurge în adâncime graful, începând cu rădăcina `3`
- Muchiile negre reprezintă muchiile din arborele parțial obținut prin parcurgerea în adâncime (*tree edges*)
- Muchiile roșii reprezintă muchiile de revenire (*back edges*)


3
1
5
6
10
2
7
4
8
9

Observații

- Rădăcina arborelui parțial DFS este punct de articulație dacă și numai dacă are cel puțin doi descendenți, între vârfuri din subarbori diferiți ai rădăcinii neexistând muchii
- Un vârf `x` oarecare nu este punct de articulație dacă și numai dacă din orice descendent `y` al lui `x` poate fi atins un strămoș al lui `x` pe un lanț format din descendenți ai lui `x` și o muchie de revenire (un drum de “siguranță” între `x` și `y`)

Definiție

- Pentru fiecare vârf `x` al grafului se definește:
- `dfn(x)`=numărul de ordine al vârfului `x` în parcurgerea DFS a grafului (*depth-first-number*)
- Dacă `x` este un strămoș al lui `y` în arborele parțial DFS, atunci:
- `dfn(x)<dfn(y)`

Exemplu

x	1	2	3	4	5	6	7	8	9	10
dfn(x)	2	4	1	5	6	7	8	9	10	3

Definiție

- Pentru fiecare vârf `x` al grafului se definește:
- `low(x)` =numărul de ordine al primului vârf din parcurgerea DFS ce poate fi atins din `x` pe un alt lanț decât lanțul unic din arborele parțial DFS

Observații

- `low(x)=min{dfn(x),min{low(y)|y descendent al lui x},min{dfn(y)|(x,y) muchie de revenire}}`

Exemplu

x	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
dfn(x)	2	4	1	5	6	7	8	9	10	3
low(x)	*1*	*1*	*1*	*1*	*6*	*6*	*6*	*9*	*10*	*3*

Explicații

- `low(1)=1`, pentru că se consideră
- `min{low(y)|y descendent al lui x}`, pentru `x=1` și `y=4`, iar `low(4)=1`
- `low(4)=1`, pentru că se consideră
- `min{dfn(y)|(x,y) muchie de revenire}`, pentru `x=4` și `y=3`, iar `dfn(3)=1`

Explicații

- `low(2)=1`, pentru că se consideră
- `min{low(y)|y descendent al lui x}`, pentru `x=2` și `y=4`, iar `low(4)=1`
- `low(4)=1`, pentru că se consideră
- `min{dfn(y)|(x,y) muchie de revenire}`, pentru `x=4` și `y=3`, iar `dfn(3)=1`

Explicații

- `low(3)=1`, pentru că se consideră `min{dfn(3)}`, iar `dfn(3)=1`
- `low(4)=1`, pentru că se consideră `min{dfn(y)|(x,y) muchie de revenire}`, pentru `x=4` și `y=3`, iar `dfn(3)=1`

Explicații

- `low(5)=6`, pentru că se consideră
- `min{dfn(5)}`, iar `dfn(5)=6`
- `low(7)=6`, pentru că se consideră
- `min{dfn(y)|(x,y) muchie de revenire}`, pentru `x=7` și `y=5`, iar `dfn(5)=6`

Explicații

- `low(6)=6`, pentru că se consideră
- `min{low(y)|y descendent al lui x}`, pentru `x=6` și `y=7`, iar `low(7)=6`
- `low(8)=dfn(8)=9`
- `low(9)=dfn(9)=10`
- `low(10)=dfn(10)=3`

Observații

- Punctele de articulație dintr-un graf se pot caracteriza astfel: `x` este punct de articulație dacă și numai dacă este rădăcina unui arbore parțial DFS cu cel puțin doi descendenți sau, dacă nu este rădăcină, are un fiu `y` astfel încât `low(y)≥dfn(x)`

Exemplu

- Nodul `3` este punct de articulație, deoarece este rădăcina arborelui parțial DFS și are doi descendenți
- Nodul `7` este punct de articulație, deoarece `low(8)=9≥dfn(7)=8`
- Nodul `5` este punct de articulație, deoarece `low(6)=6≥dfn(5)=6`
- Nodul `1` este punct de articulație, deoarece `low(10)=3≥dfn(1)=2`

Idei de implementare

- Se folosesc 3 vectori de noduri:
- **1**. `dfn[x]` este momentul vizitării (descoperirii) vârfului `x` în explorarea DFS
- **2**. `p[x]` este predecesorul vârfului `x` în arborele de explorare DFS
- **3**. `low[x]` este numărul de ordine al primului vârf din parcurgerea DFS ce poate fi atins din `x` pe un alt lanț decât lanțul unic din arborele parțial DFS

Observații

- Vectorul `low` se determină la vizitarea DFS
- Funcția ce determină **punctele de articulație** verifică, pe rând, pentru fiecare vârf din graf, ce statut are în arborele DFS

Funcție care numără fii lui **x** în arborele descris prin vectorul de predecesori **p**

```
int fii (int x, int p[], int n) {
int i, m = 0;
for (i = 1; i <= n; i++)
        if ( i != x && p[i] == x) // dacă i are ca părinte pe x
        m++;
return m;
}
```

```
//Funcție de parcurgere în adâncime din vârful x, cu
//crearea vectorilor dfn, p, low
void dfs (Graf g, int x, int t, int dfn[], int p[], int low[]) {
int w;
low[x] = dfn[x] = ++t;
for (w = 1; w <= g.n; w++) {
        if (g.a[x][w]) // dacă w este vecin cu x
                if (dfn[w] == 0) { // dacă w nevizitat
                p[w] = x; // w are ca predecesor pe x
                dfs(g,w,t,dfn,p,low); //continuă vizitarea din w
                low[x]=min(low[x],low[w]); //actualizare low[x]
                }
                else // dacă w deja vizitat
                if ( w != p[x]) // dacă muchie de revenire (x,w)
                low[x]=min(low[x],dfn[w]); // actualizare low[x]
        }
}
```

```
//Funcție de găsire a punctelor de articulație
void articulatie (Graf g, int dfn[], int p[], int low[]) {
int x, w, t = 0; // t = moment vizitare (descoperire vârf)
dfs(g,1,t,dfn,p,low); // vizitare din 1 (graf conex)
for (x = 1; x <= g.n; x++) {
if (p[x] == 0) {
        if (fii(x, p, g.n) > 1) //dacă rădăcină cu cel puțin 2 fii
                printf("%d ", x); //este punct de articulație
        }
        else // dacă nu e rădăcina
        for (w = 1; w <= g.n; w++) {
        // dacă x are un fiu w în arborele DFS
        if (p[w]==x && low[w]>=dfn[x]) // cu low[w]>=dfn[x]
        printf("%d ", x); // atunci x este punct de articulație
} } }
```