Cursul nr. 6

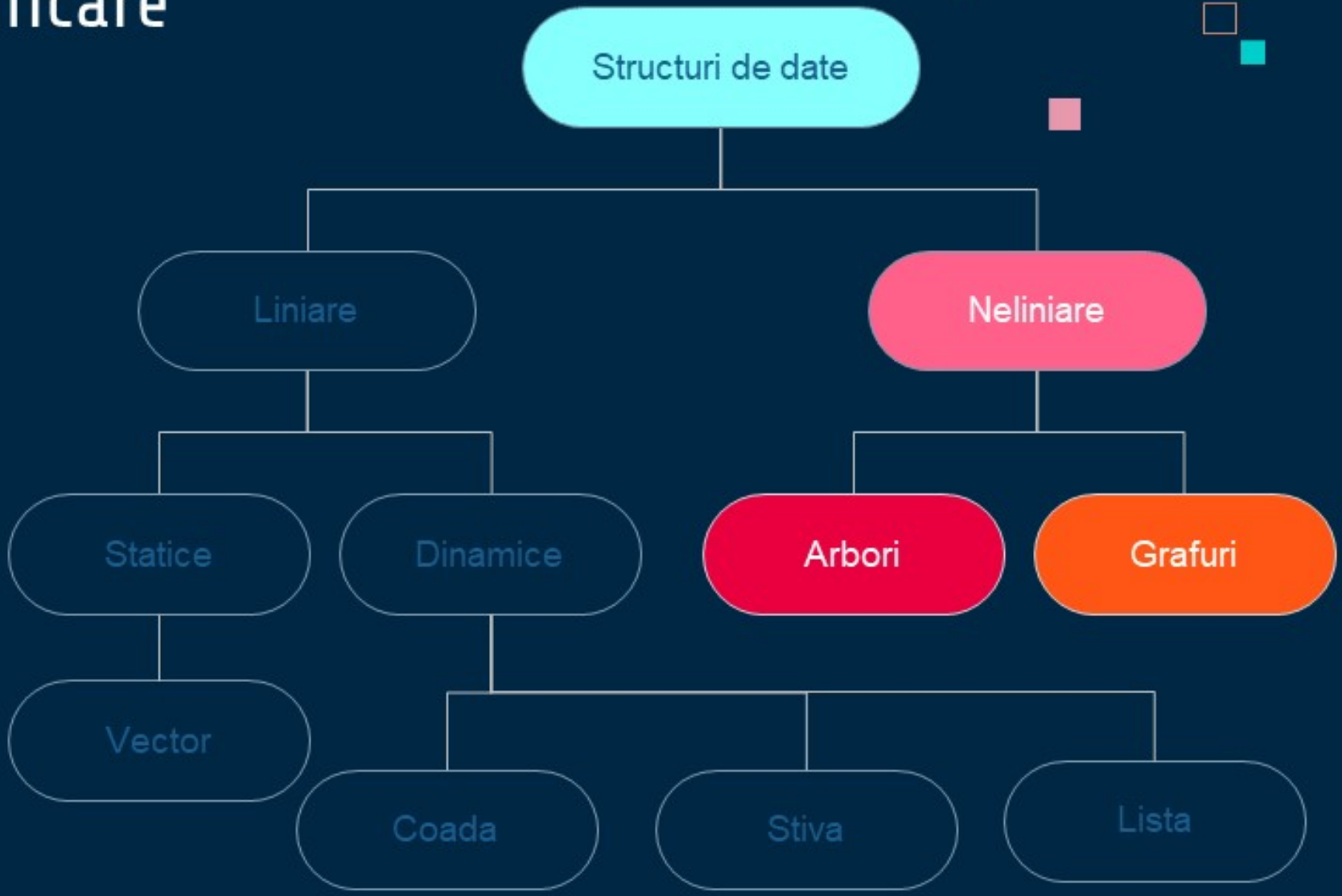
STRUCTURI DE DATE

neliniare

Grafuri

Lector dr. Dorin IORDACHE

Clasificare



Agenda

01

Generalitati

02

Reprezentare

03

Căutare

Generalități

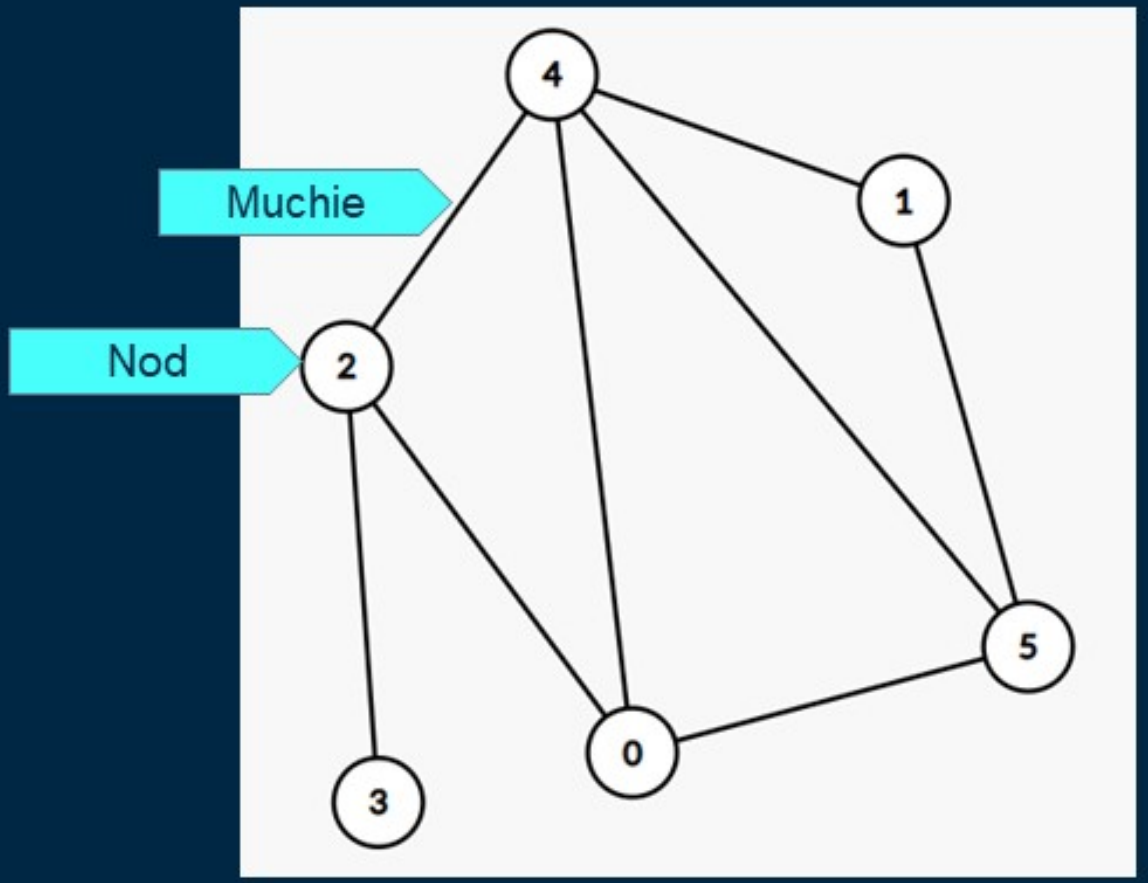
01

Graf

Un graf este o structură de date neliniară constând din vârfuri (noduri) și muchii (arce).

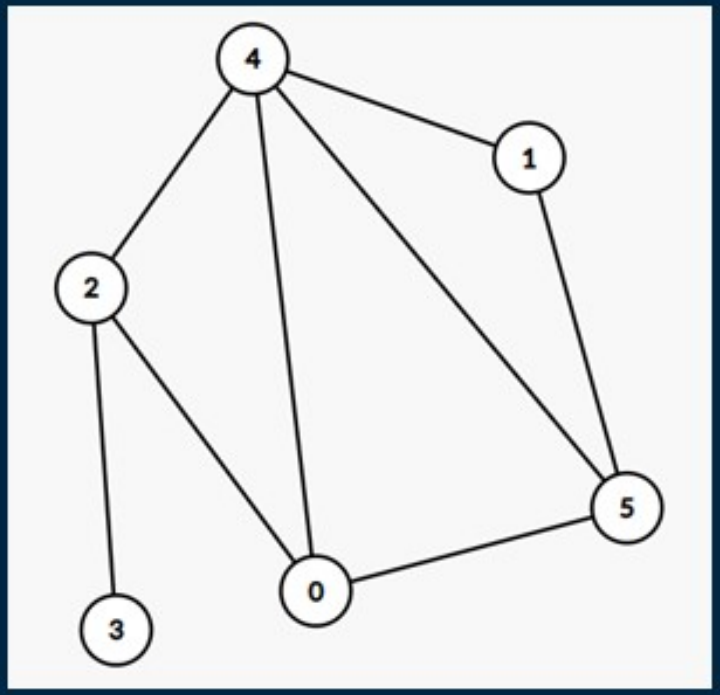
Notam:

G(Varfuri, Muchii)



Graf G(V,M)

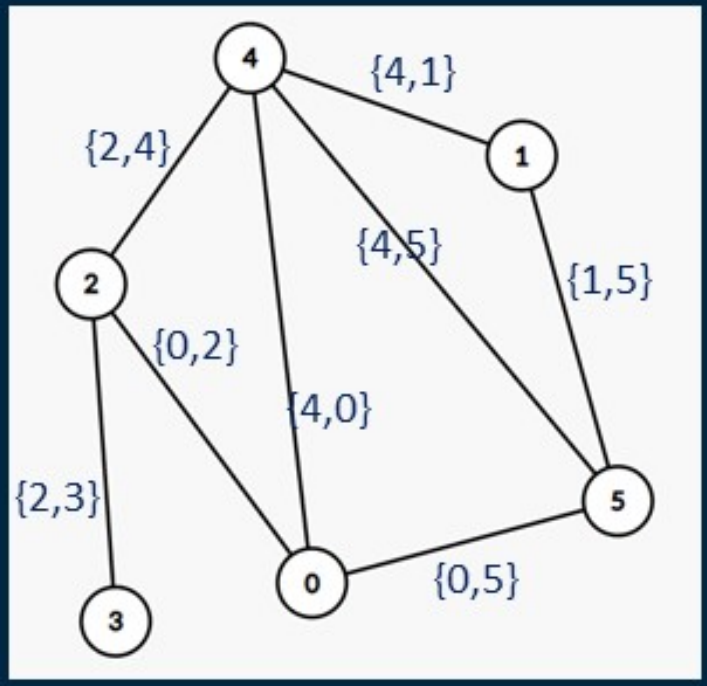
= o pereche G (V, M), unde V este o multime de varfuri, iar M o submultime inclusa in V × V este o multime de muchii. O muchie de la varful a la varful b este notata cu perechea ordonata (a, b), daca graful este orientat, si cu multimea {a, b}, daca graful este neorientat.
= Doua varfuri unite printr-o muchie se numesc adiacente.
= Un varf care este extremitatea unei singure muchii se numeste varf terminal.



Graf - definire

Un graf G=(V, E) o mulime de varfuri, V si una de muchii, M.
Fiecare muchie este o pereche *(v, w)*, unde v, w apartin multimii V
Dac[pereche (v,w) nu este ordonata(nu conteaza sensul d eparcurgere de la nodul v la w) atunci graful este **neorientat**, atlfet este graf **orientat**.

V= { 0,1,2,3,4,5}
M = {{2,3}, {2,4}, {0,2}, {4.0}, {4,5}, {0,5}, {4,1}, {1,5}}

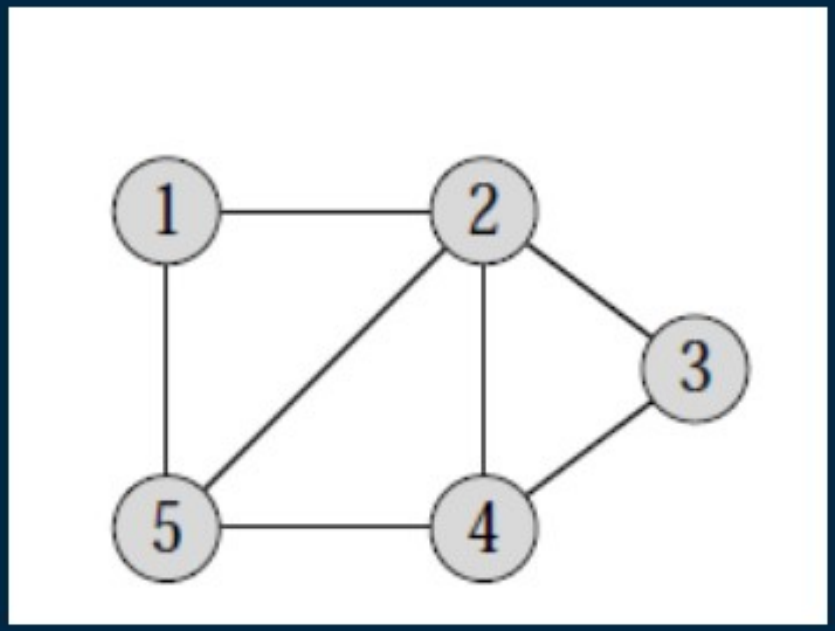


Graf - variante

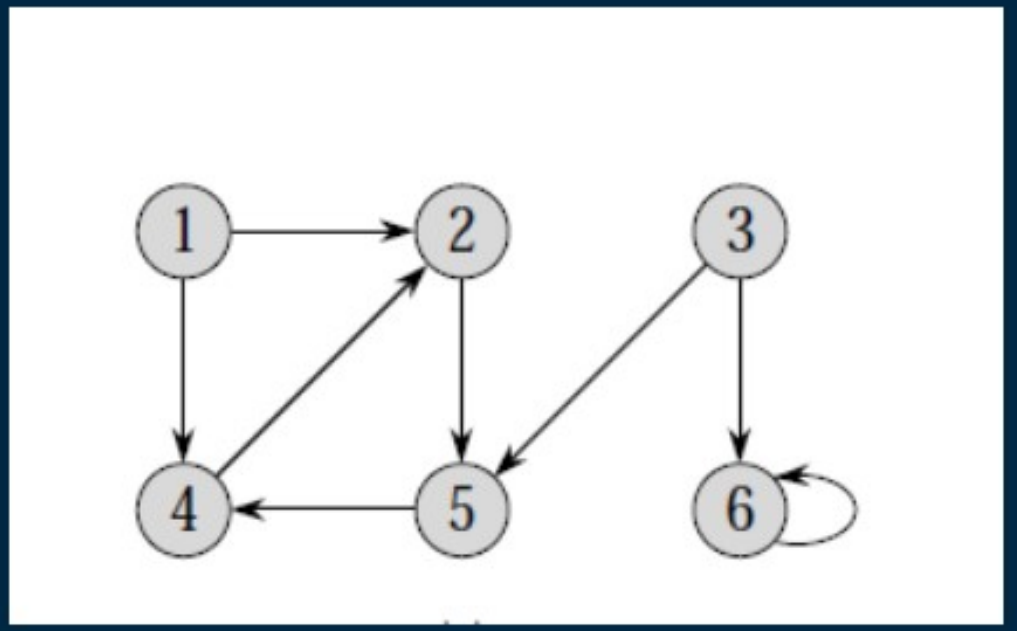
- Graf **conex** are muchii de la fiecare vârf c[tre celellate
- Graf neorientat:
    - Muchia (u,v) = Muchia (v,u)
    - Fără auto-bucle
- Graf orientat:
    - Muchia (u,v) are sensul de parcurgere de la vârful u la vârful v, notat u $\rightarrow$ v

Graf

Neorientat



Orientat



Calea între muchii

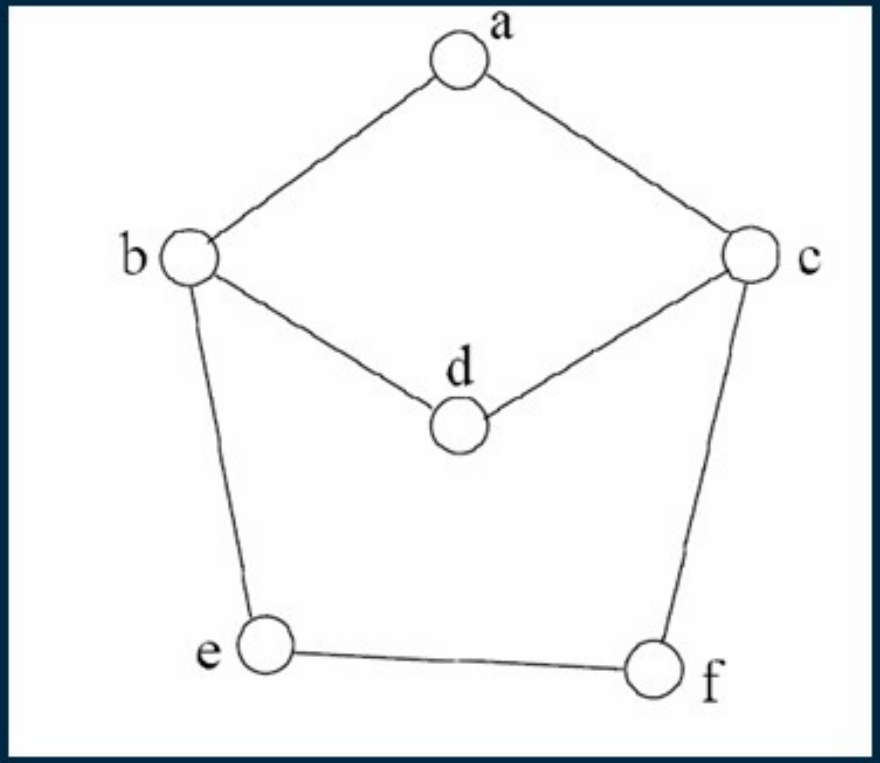
O cale este o succesiune de vârfuri (v0, v1, v2,... vk) astfel încât:

Pentru $0 \leq i < k$, {vi, vi+1} este o muchie

Notă: o cale are voie să treacă prin același vârf sau aceeași muchie de oricâte ori!

Lungimea unei căi este numărul de muchii ale căii

Cale - exemple



1. {a,c,f,e}
2. {a,b,d,c,f,e}
3. {a, c, d, b, d, c, f, e}
4. {a,c,d,b,a}
5. {a,c,f,e,b,d,c,a}

Tipuri de cale

O cale este **simplă** dacă și numai dacă nu conține un vârf de mai multe ori.

O cale este un **ciclu** dacă și numai dacă $v_0 = v_k$

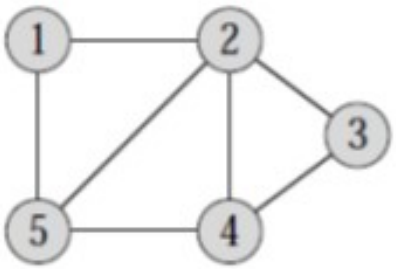
Începutul și sfârșitul sunt același vârf!

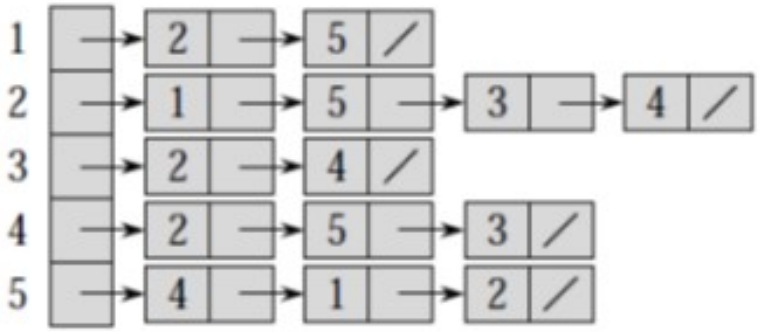
O cale conține un ciclu ca traseu secundar dacă un vârf apare de două sau mai multe ori

Reprezentare

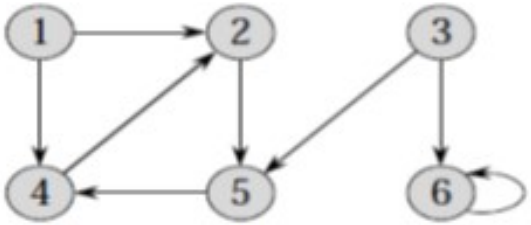
02

Reprezentarea unui graf

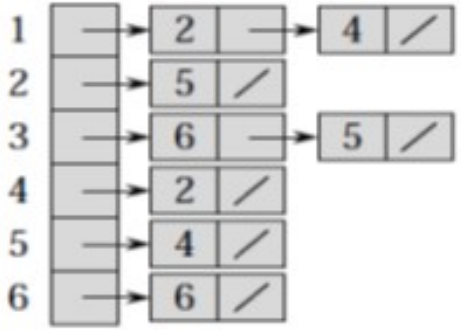




	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



(a)

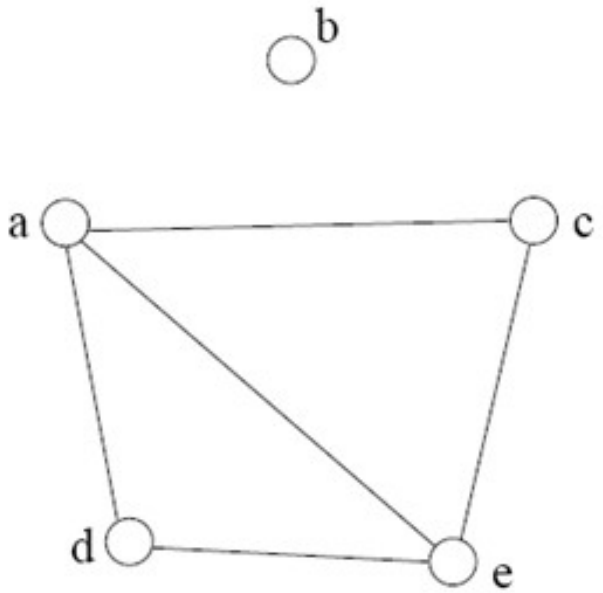


(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

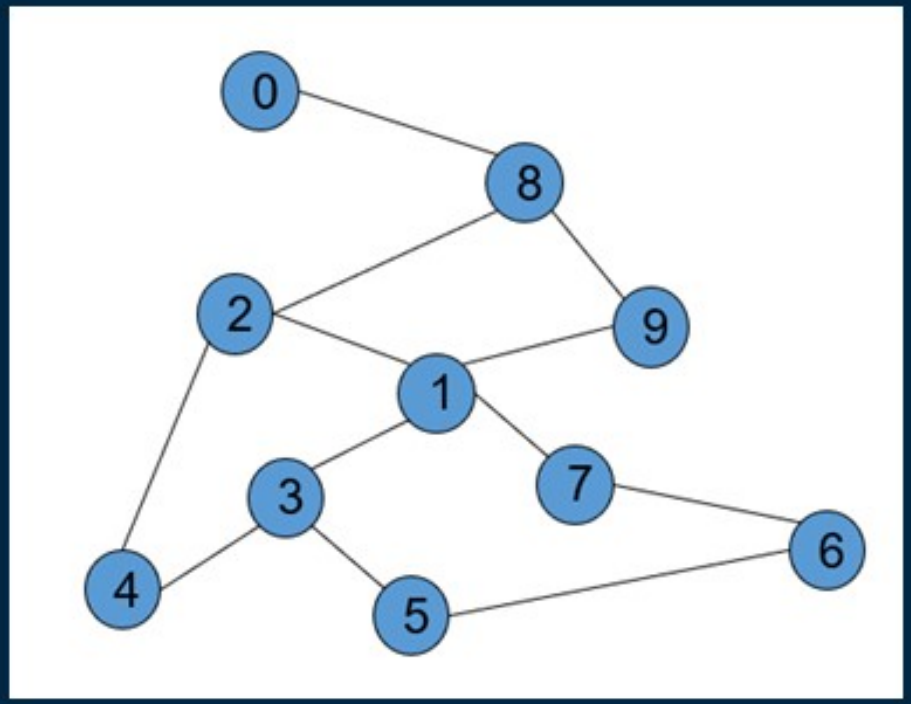
Reprezentarea unui graf – Matrice de adiacență



	a	b	c	d	e
a	0	0	1	1	1
b	0	0	0	0	0
c	1	0	0	0	1
d	1	0	0	0	1
e	1	0	1	1	0

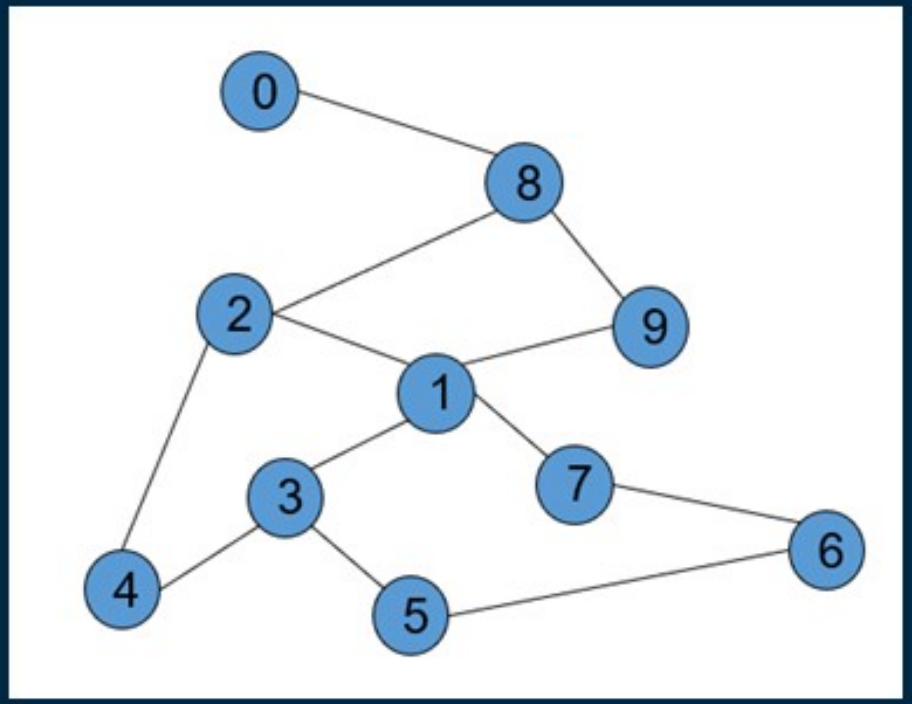
Matrice (vector bidimensional) M[0..n-1, 0..n-1], unde n este numărul de vârfuri din graf
Fiecare rând și coloană este indexată după id-ul vârfului
e,g a=0, b=1, c=2, d=3, e=4
M[i][j]=1 dacă există o muchie care leagă vârfurile i și j;
M[i][j]=0, în caz contrar

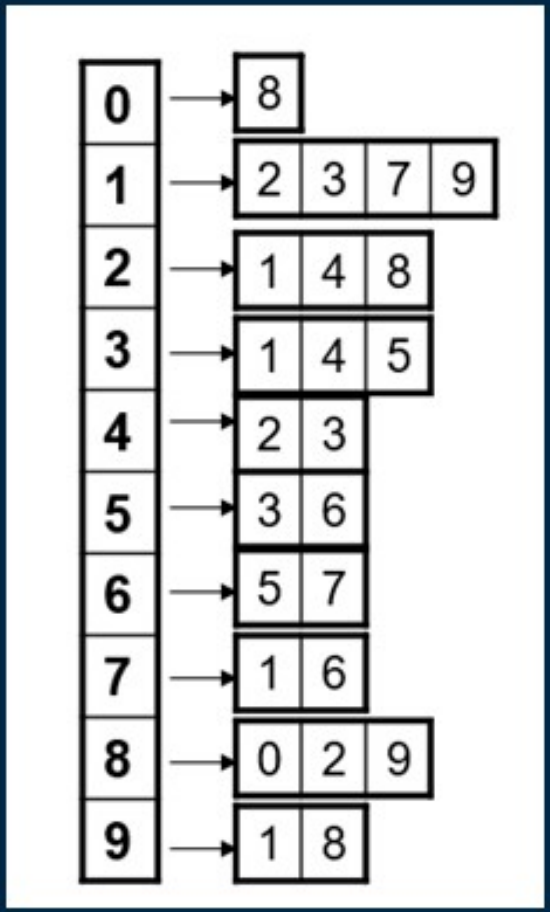
Matrice de adiacență - exemplu



	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	0	1	0	1
2	0	1	0	0	1	0	0	0	1	0
3	0	1	0	0	1	1	0	0	0	0
4	0	0	1	1	0	0	0	0	0	0
5	0	0	0	1	0	0	1	0	0	0
6	0	0	0	0	0	1	0	1	0	0
7	0	1	0	0	0	0	1	0	0	0
8	1	0	1	0	0	0	0	0	0	1
9	0	1	0	0	0	0	0	0	1	0

Listă de adiacență - exemplu





Matrice vs Listă de adiacență

Listă de adiacență:

Mai compact decât matricele de adiacență dacă graficul are puține muchii

Necesită mai mult timp pentru a afla dacă există o muchie

Matrice de adiacență:

Necesită întotdeauna spațiu n^2

Acest lucru poate pierde mult spațiu dacă numărul de margini este rar

Poate găsi rapid dacă există o muchie

Căutare

03

Tipuri de parcurgeri

Breadth-First Search (BFS) (în lățime)

- calea cea mai scrută într-un graf neponderat

Depth-First Search (DFS) (în adâncime)

- sortare topologica
- cautare componente puternic conectat

Breadth-First Search (BFS) (în lățime)

Are la bază operarea cu mulțimile nodurilor: vizitate si evizitate
- a marca fiecare vârf ca vizitat, evitând în același timp ciclurile.

Algoritmul funcționează după cum urmează:

Pas 1: Se alege oricare dintre nodurile graficului ca punct de start.
Pas 2: Se adaugă vârful în lista nodurilor vizitate.
Pas 3: Se creează lista nodurilor adiacente acelui vârf
adăugându-le în lista nodurilor nevizitate.
Pas 4: Se repetă pașii 2 și 3 până când lista nodurilor nevizitate este vidă.

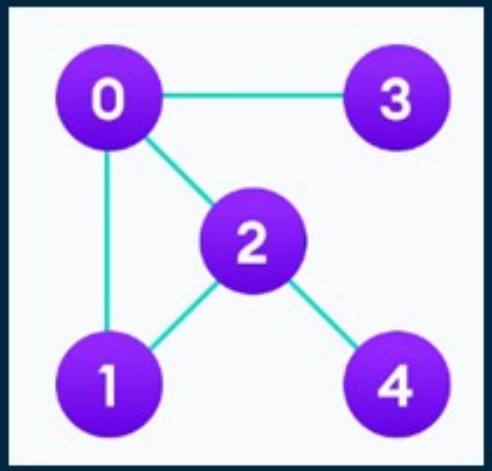
https://www.programiz.com/dsa/graph-bfs

Breadth-First Search (BFS) (în lățime)

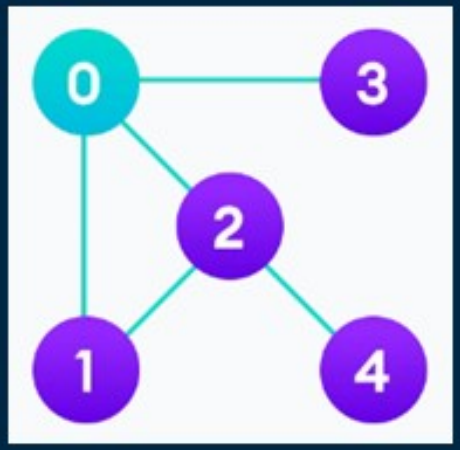
Are la bază operarea cu mulțimile **nodurilor**:

vizitate

nevizitate



Breadth-First Search (BFS) (în lățime)



Noduri

0				

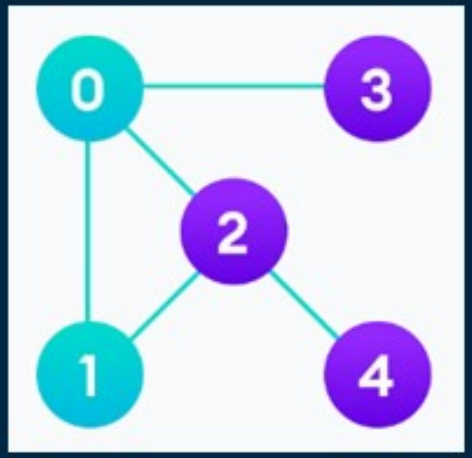
vizitate

1	2	3		

nevizitate

prim

Breadth-First Search (BFS) (în lățime)



Noduri

0	1			

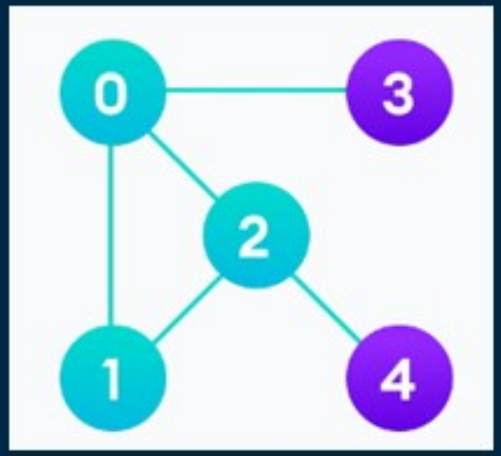
vizitate

2	3			

nevizitate

prim

Breadth-First Search (BFS) (în lățime)



Noduri

0	1	2		

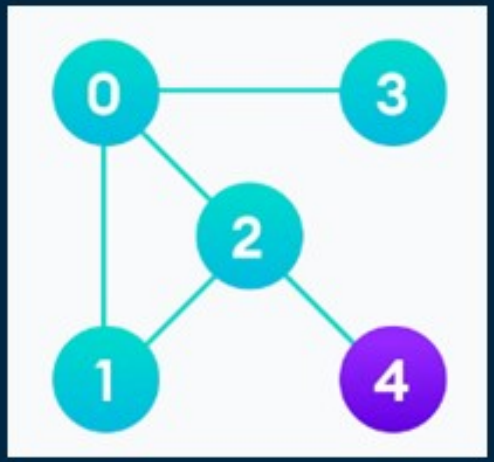
vizitate

3	4			

nevizitate

prim

Breadth-First Search (BFS) (în lățime)



Noduri

0	1	2	3	

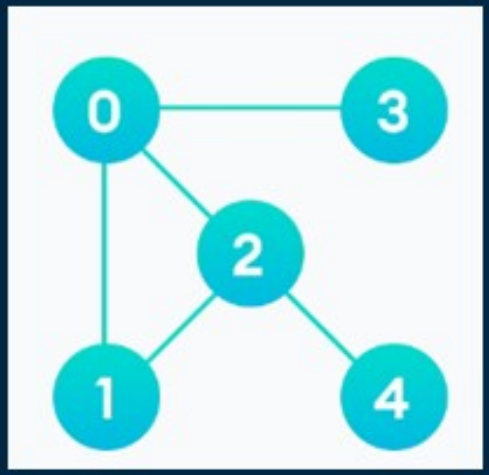
vizitate

4				

nevizitate

prim

Breadth-First Search (BFS) (în lățime)



Noduri

0	1	2	3	4

vizitate

nevizitate

prim

BFS – implememtare

```
create  queue Q
mark v as visited and put v into Q
 while Q is noEmpty  do
            remove the head u of Q
            mark and enqueue all (unvisited) neighbours of u
end while
```

BFS - aplicatii

1. construire unui index după criteriu de căutare
2. navigare GPS
3. algoritmi de căutare a căilor
4. algoritmul Ford-Fulkerson pentru a găsi fluxul maxim într-o rețea
5. detectarea ciclului într-un grafic neorientat
6. arbore de drum minim

Depth-First Search (BFS) (în adâncime)

Are la bază operarea cu mulțimile nodurilor: vizitate si evizitate
- a marca fiecare vârf ca vizitat, evitând în același timp ciclurile.

Algoritmul funcționează după cum urmează:

Pas 1: Se alege oricare dintre nodurile graficului ca punct de start.
Pas 2: Se adaugă vârful în lista nodurilor vizitate.
Pas 3: Se creează lista nodurilor adiacente acelui vârf
adăugându-le în stiva nodurilor nevizitate.
Pas 4: Se repetă pașii 2 și 3 până când lista nodurilor nevizitate este vidă.

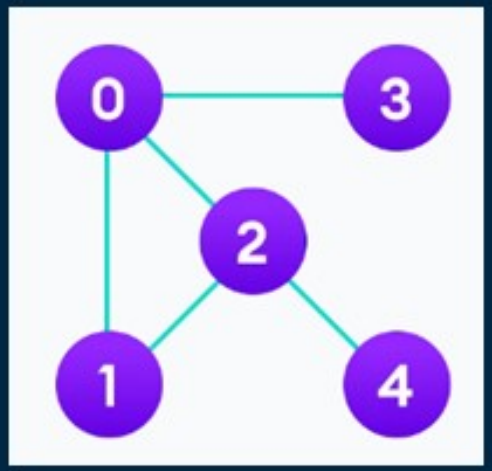
https://www.programiz.com/dsa/graph-bfs

Depth-First Search (BFS) (în adâncime)

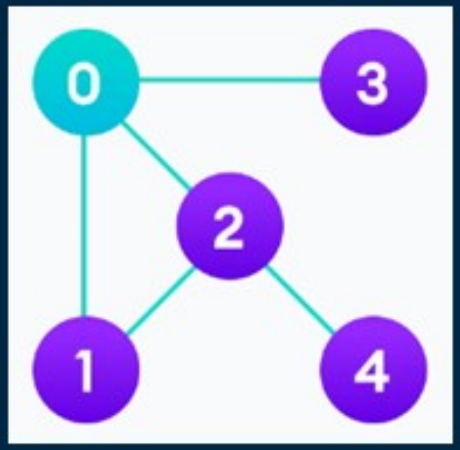
Are la bază operarea cu mulțimile **nodurilor**:

vizitate

nevizitate



Depth-First Search (BFS) (în adâncime)



Noduri

0				

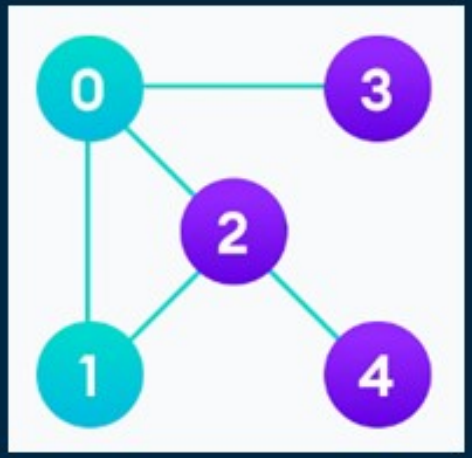
vizitate

3	2	1		

stivă

top

Depth-First Search (BFS) (în adâncime)



Noduri

0	1			

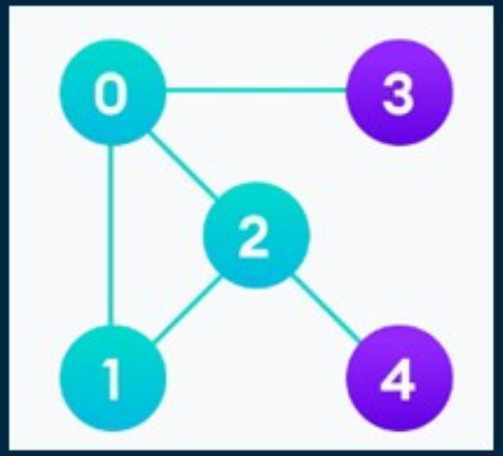
vizitate

3	2			

nevizitate

top

Depth-First Search (BFS) (în adâncime)



Noduri

0	1	2		

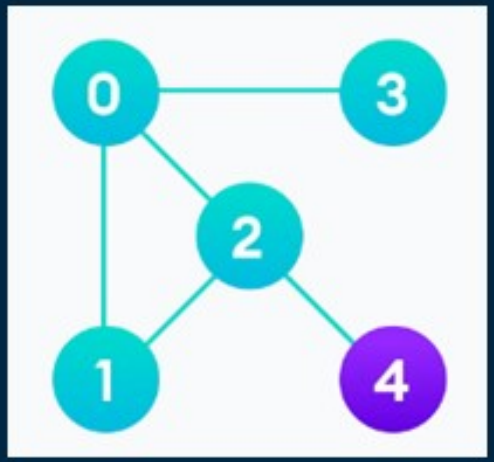
vizitate

3	4			

nevizitate

top

Depth-First Search (BFS) (în adâncime)



Noduri

0	1	2	3	

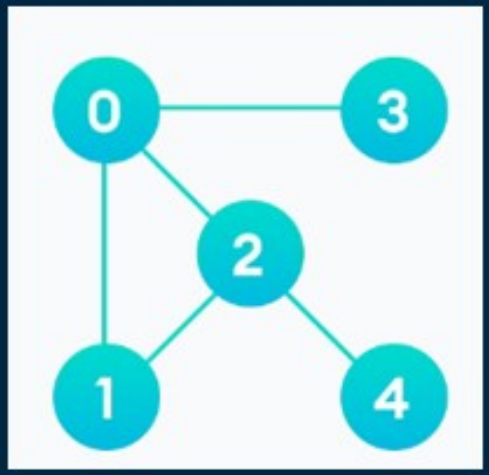
vizitate

4				

nevizitate

top

Depth-First Search (BFS) (în adâncime)



Noduri

0	1	2	3	4	vizitate
					nevizitate

DFS - implememtare

```
create  stack Q
mark v as visited and put v into Q
 while Q is noEmpty  do
            remove the top u of Q
            mark and enqueue all (unvisited) neighbours of u
end while
```

DFS - aplicatii

1. algoritmi de căutare a căilor
2. testare dacă un graf este bipartit
3. detectare subgraf puternic conectat
4. detectarea circuitelor din graf

Aplicații

- crawlere web (cum găsește Google paginile)
- rețele sociale (căutare/regăsire prieteni Facebook)
- rutarea de pachete în rețea
- "garbage collector" – eliminare secvente de memorie
- control integritate model (mașină cu stări finite)
- verificarea conjecturilor matematice
- rezolvarea de puzzle-uri și jocuri

Intrebari?

dorin.lordache@365.univ-ovidius.ro

Mulțumesc