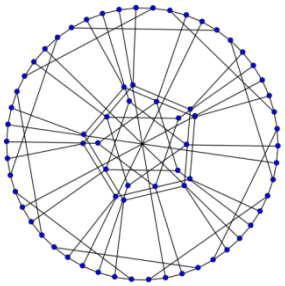
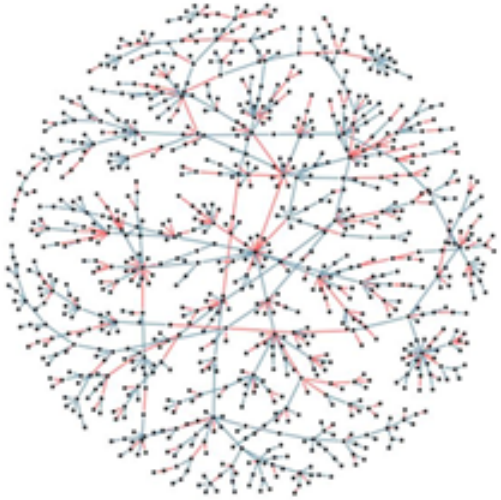


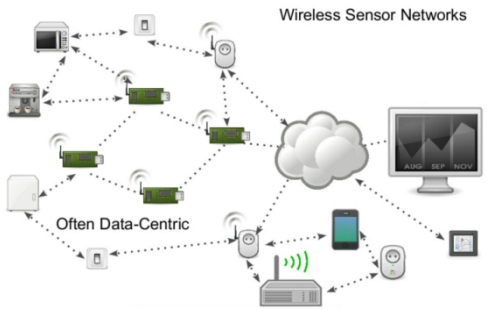
# Traversarea grafurilor

*SD.11*

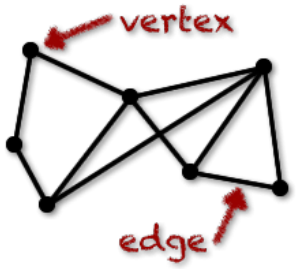
# Grafuri – aplicații



- Hărți cu drumuri (drum minim, accesibilitate noduri)
- Rețele de calculatoare (Spanning Tree Protocol - Cisco )
- Rețele energetice (transportul energiei)
- Rețele de senzori (data fusion)
- Web (FaceBook, Google PageRank)
- Rețele sociale (centralitate)
- Modelări grafice

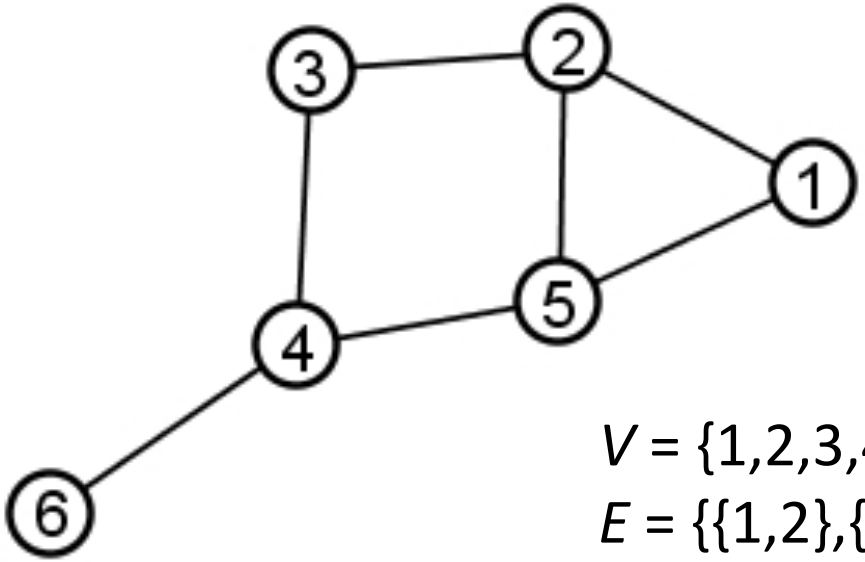



| Graph | Vertices | Edges |
|---|---|---|
| communication | telephones, computers | fiber optic cables |
| circuits | gates, registers, processors | wires |
| mechanical | joints | rods, beams, springs |
| hydraulic | reservoirs, pumping stations | pipelines |
| financial | stocks, currency | transactions |
| transportation | street intersections, airports | highways, airway routes |
| scheduling | tasks | precedence constraints |
| software systems | functions | function calls |
| internet | web pages | hyperlinks |
| games | board positions | legal moves |
| social relationship | people, actors | friendships, movie casts |
| neural networks | neurons | synapses |
| protein networks | proteins | protein-protein interactions |
| chemical compounds | molecules | bonds |




# Graf

- graf G = pereche ordonată de mulţimi (V,E), unde
  - V = mulţime nevidă şi finită de elemente denumite vârfurile grafului.
  - E = mulţime de perechi de vârfuri din graf.



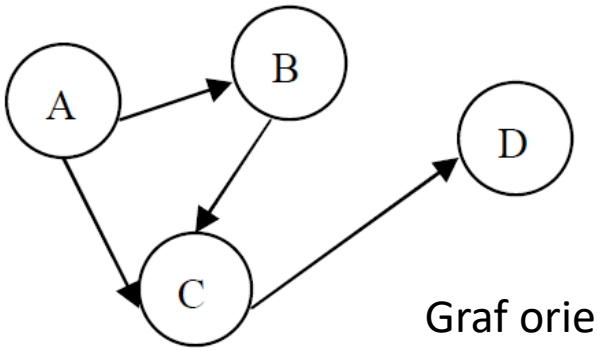

$V = \{1,2,3,4,5,6\}$

$E = \{\{1,2\},\{1,5\},\{2,3\},\{2,5\},\{3,4\},\{4,5\},\{4,6\}\}$

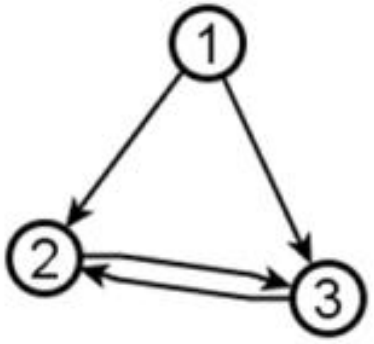
G = (V,E)

# Graf orientat

- perechile de vârfuri din mulţimea E sunt ordonate si sunt denumite **arce**.
- Un arc se parcurge doar de la un anumit nod la altul: de la extremitatea iniţială a arcului la extremitatea finală a arcului .



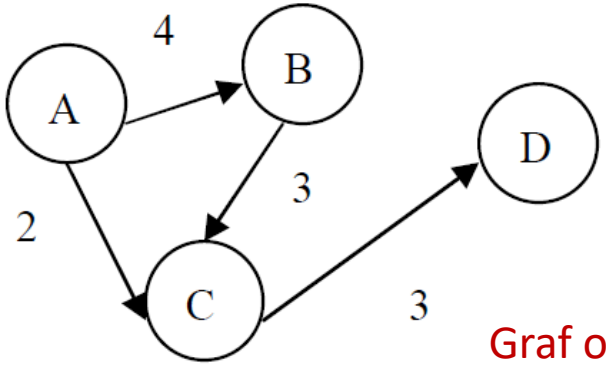

Graf orientat




Directed graph $(V_2, E_2)$

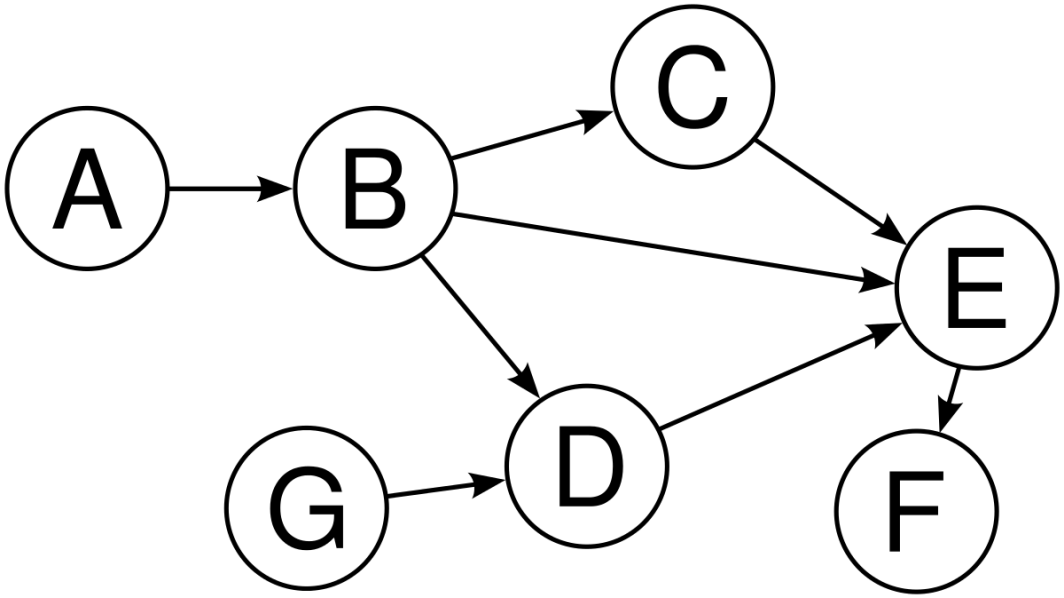
$V_2 = \{1,2,3\}$

$E_2 = \{(1,2),(2,3),(3,2),(1,3)\}$




Graf orientat ponderat

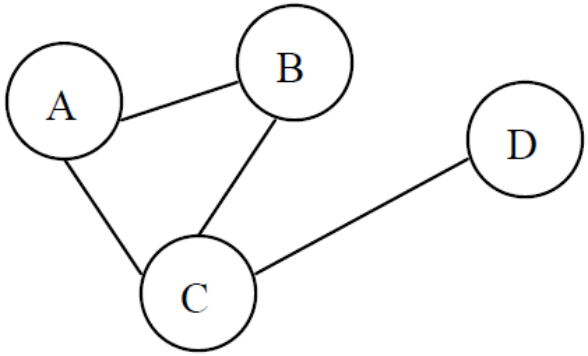
# Directed Acyclic Graph (DAG)



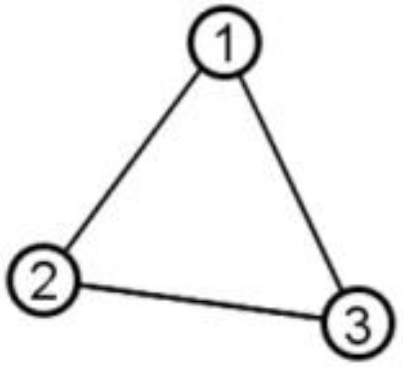

In a directed graph, the edges are connected so that each edge only goes one way.
A directed acyclic graph means that the graph is not cyclic, or that it is impossible to start at one point in the graph and traverse the entire graph. Each edge is directed from an earlier edge to a later edge. This is also known as a topological ordering of a graph.

# Graf neorientat

- perechile de vârfuri din mulţimea E sunt neordonate şi sunt denumite **muchii** (o muchie se poate parcurge in ambele sensuri).




Graf neorientat




Undirected graph $(V_1, E_1)$

$V_1 = \{1,2,3\}$

$E_1 = \{\{1,2\},\{2,3\},\{3,1\}\}$

# Adiacenta

- Dacă există un arc sau o muchie cu extremitățile x și y, atunci vârfurile x și y sunt adiacente; fiecare extremitate a unei muchii/unui arc este considerată incidentă cu muchia/arcul respectiv.

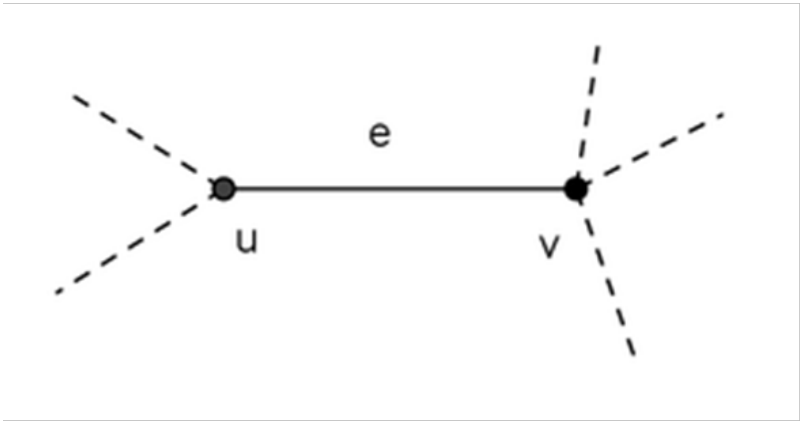
- **Adiacența**: într-un graf neorientat existența muchiei (v,w) presupune că w este adiacent cu v şi v adiacent cu w.

- Adjacent Vertices: two vertices that are connected by an edge

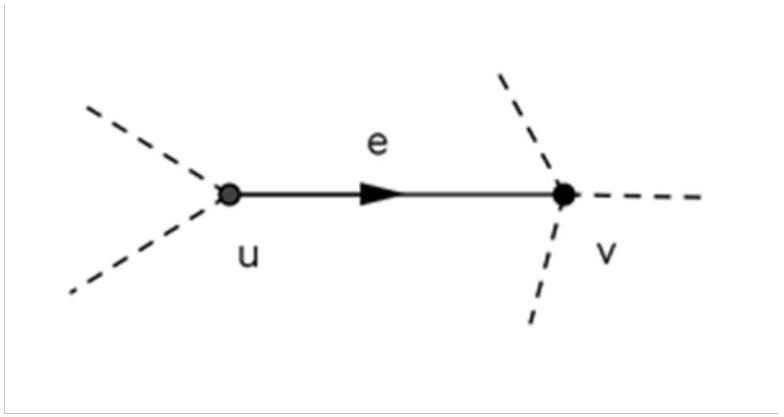
- Adjacent Edges: two edges that share a common vertex

# Incidenta

**Incidență** : o muchie este incidentă cu un nod dacă îl are pe acesta ca extremitate. Muchia (u,v) este incidentă în nodul *u* respectiv *v*.




*u* and *v* are each incident with *e*
*e* is incident with *u* and incident with *v*.




*e* is incident from *u*
*v* is incident from *e*.

*e* is incident to *v*
*u* is incident to *e*.

# Gradul unui nod

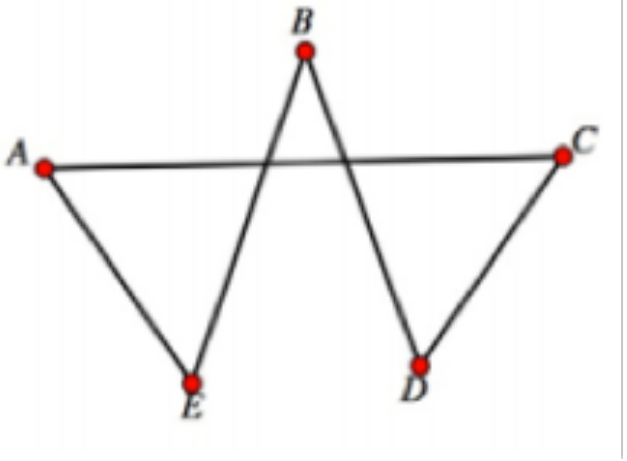
- **Gradul** unui nod v, dintr-un graf neorientat, este un număr natural ce reprezintă numărul de noduri adiacente cu acesta (sau numarul de muchii incidente cu nodul respectiv)

- Nod izolat = Un nod cu gradul 0.

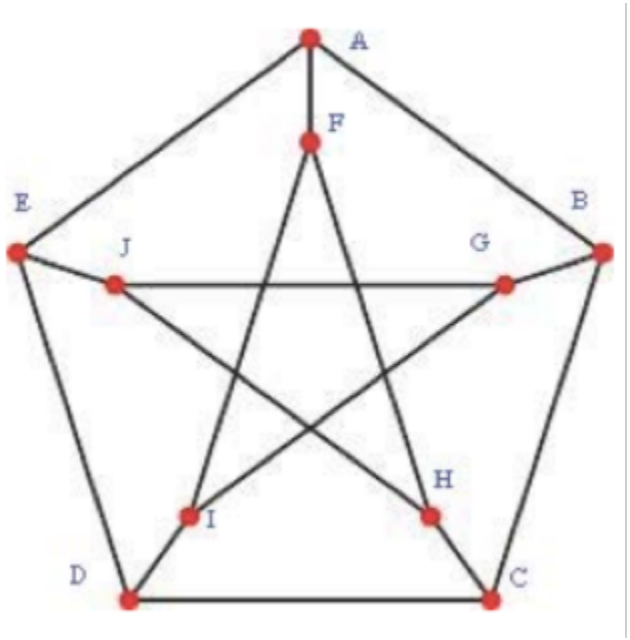
- Nod terminal= un nod cu gradul 1

# Ciclu,lant Hamiltonian

- *Lant hamiltonian* = un lant elementar care contine toate nodurile unui graf
- *Ciclu hamiltonian* = un ciclu elementar care conţine toate nodurile grafului




Hamilton circuit: ACDBEA

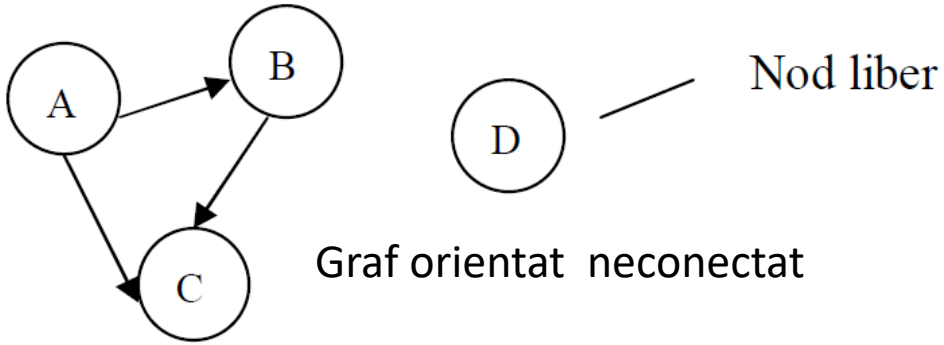



Hamilton path: ABCDEJGIFH
(but no Hamilton circuits)

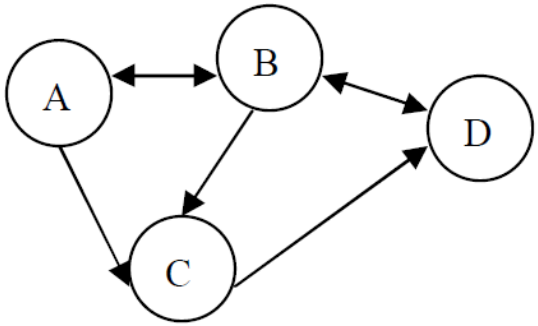
**Hamilton Circuit**: a circuit that must pass through each vertex of a graph once and only once
**Hamilton Path**: a path that must pass through each vertex of a graph once and only once

# Graf conex

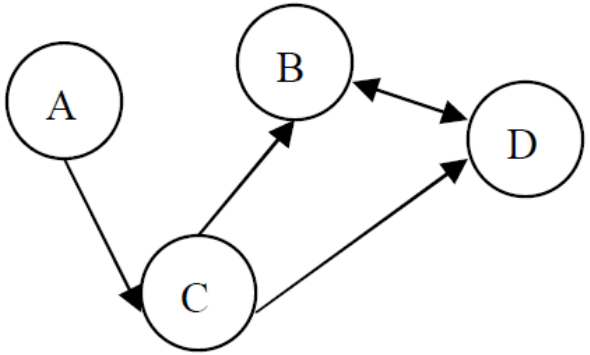
- Un graf neorientat este conex dacă există un drum între oricare două noduri distincte.
- Un graf orientat este tare conex dacă, oricare ar fi două noduri ale acestuia *u* și *v*, există drum și de la *u* la *v*, și de la *v* la *u*.




Graf orientat neconectat



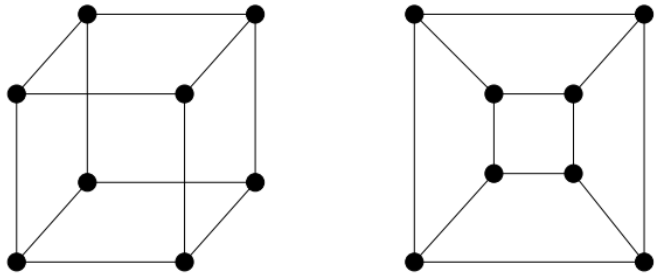

Graf orientat puternic conectat

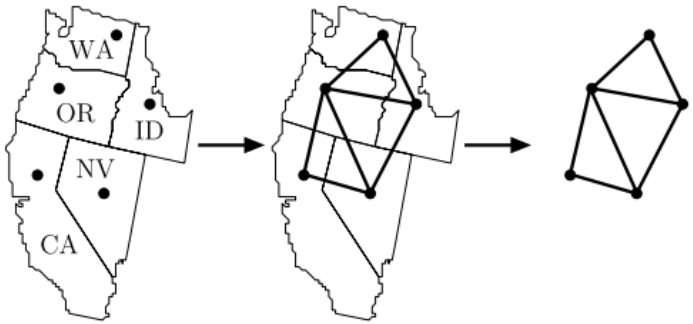


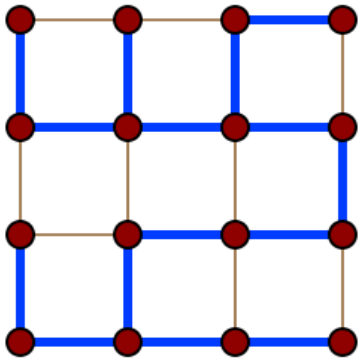

Graf orientat slab conectat

# Problematica

- reprezentare
- colorare
- **traversare**
- cautare
- concatenare
- arbori de acoperire (spanning tree)
- drum minim

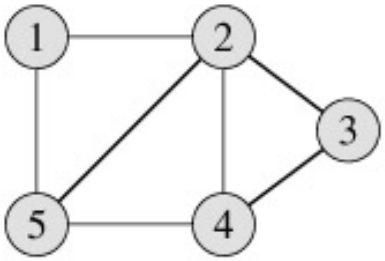




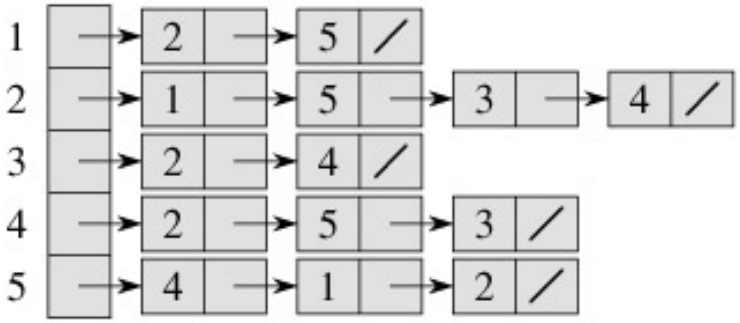



# Reprezentarea grafurilor

- **Matrice de adiacență**
- **Liste de adiacență**
- *lista muchiilor*
- *matricea vârfuri-arce (m. de incidență)*
- *matricea drumurilor*




graph

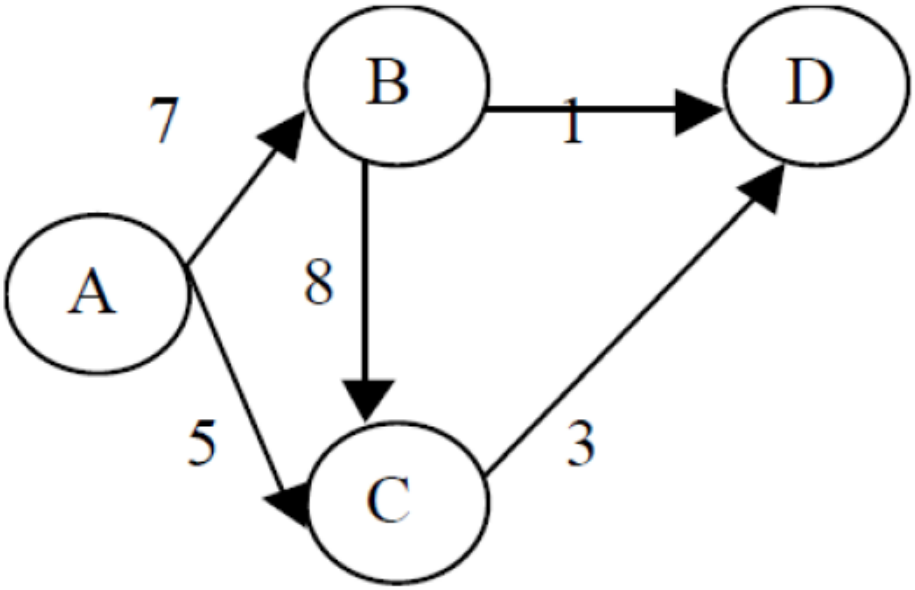



Adjacency list

| | 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 |

Adjacency matrix

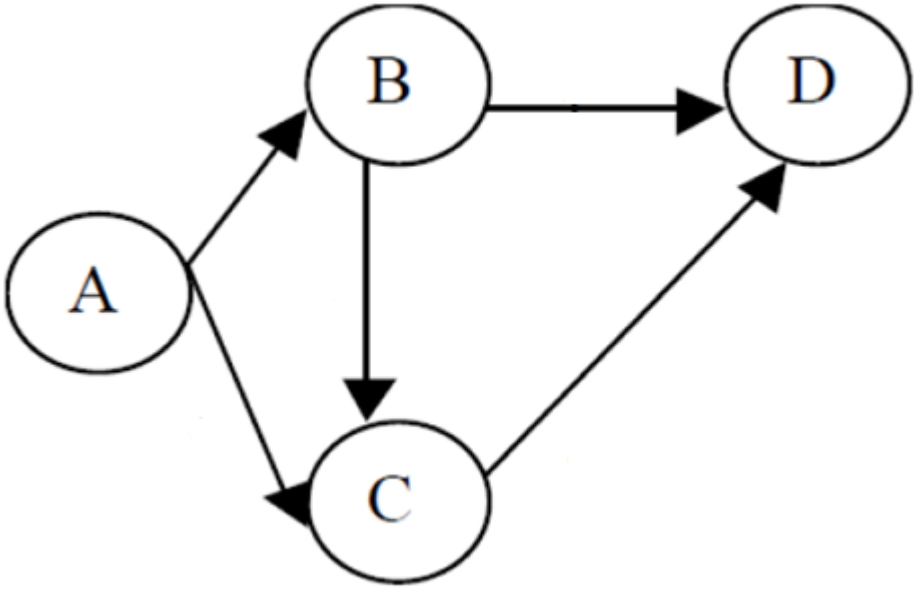
# Matricea de adiacență




| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 7 | 5 | 0 |
| B | 0 | 0 | 8 | 1 |
| C | 0 | 0 | 0 | 3 |
| D | 0 | 0 | 0 | 0 |

graf orientat ponderat

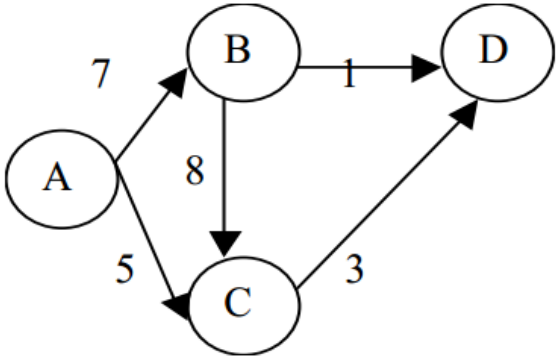
# Matricea de adiacență



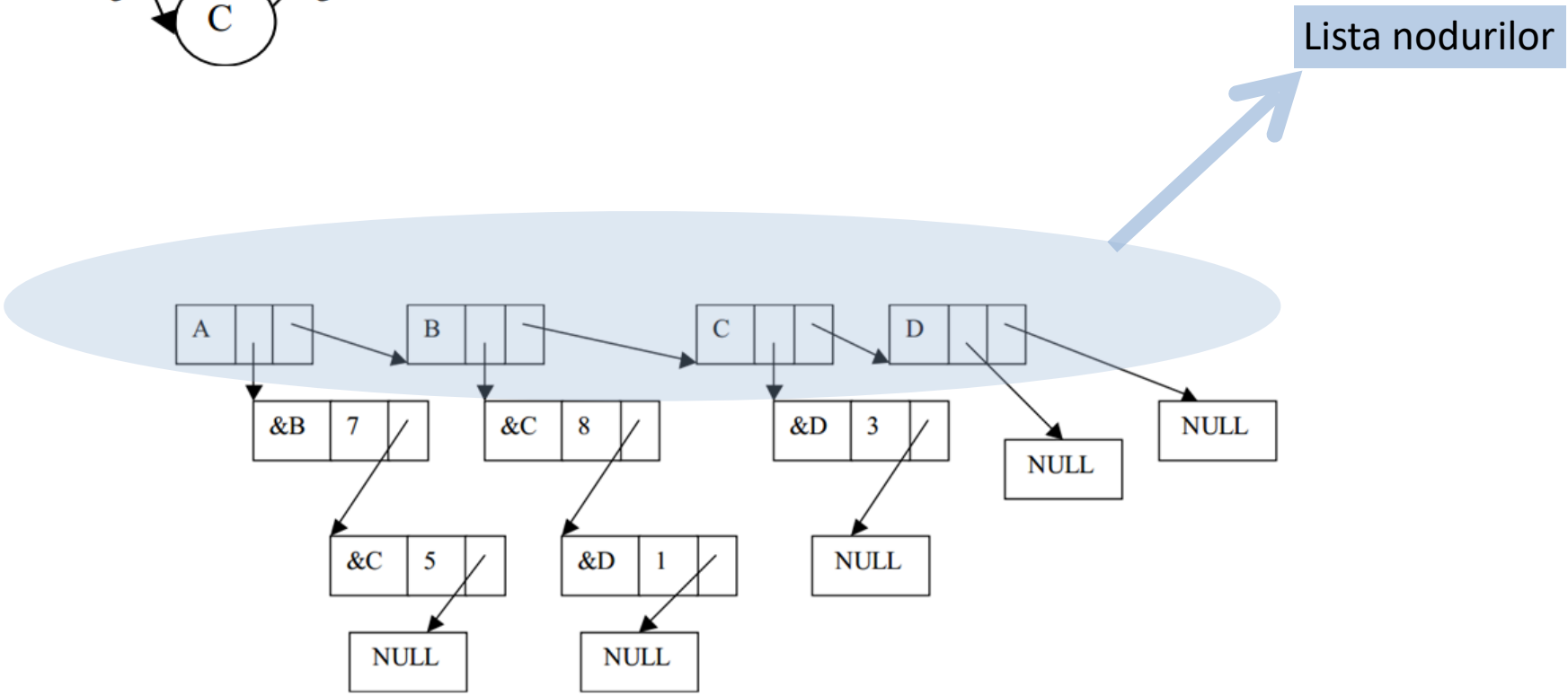

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 |
| C | 0 | 0 | 0 | 1 |
| D | 0 | 0 | 0 | 0 |

graf orientat neponderat

```
// structura Graf cu matrice de adiacenta alocata dinamic
typedef struct {
        int n,m; // n=nr de noduri, m=nr de arce
        int **a; // adresa matrice de adiacente
} Graf

void initGraf (Graf &g, int n) { // initializare graf cu n noduri
        int i;
        g.n=n; g.m=0;
        g.a = (int**) malloc( (n+1)*sizeof(int*));
        // daca varfurile sunt numerotate 1,2,3..n
        for (i=1;i<=n;i++)
                g.a[i]= (int*) calloc( (n+1),sizeof(int));
}

void addArc (Graf &g, int x,int y) {  // adauga arcul (x,y)
        g.a[x][y]=1; g.m++;
}

int arc (Graf &g, int x, int y) {     // verifica daca exista arcul (x,
        return g.a[x][y];
}

void delArc (Graf &g, int x, int y) {         // elimina arcul (x,y)
        g.a[x][y]=0; g.m--;
}
```



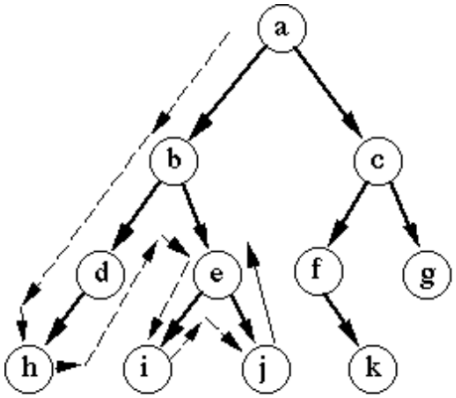

# Liste de adiacență




Atunci când numărul nodurilor din graf este mult mai mare ca numărul de arce (matrice rară - foarte multe elemente nule) se preferă reprezentarea prin liste de adiacențe.

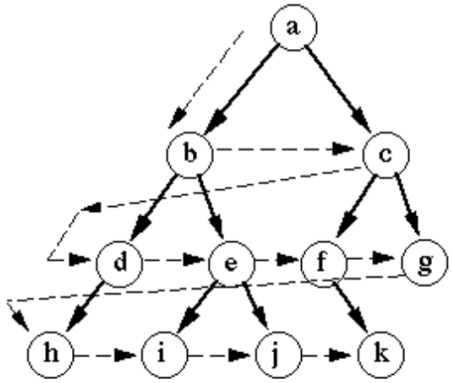
# Traversarea arborilor

- **Depth-First Search**
  - recursiv
    - în cazul arborilor binari:
      - inordine,
      - preordine,
      - postordine
  - iterativ: utilizează stiva



Depth-first search

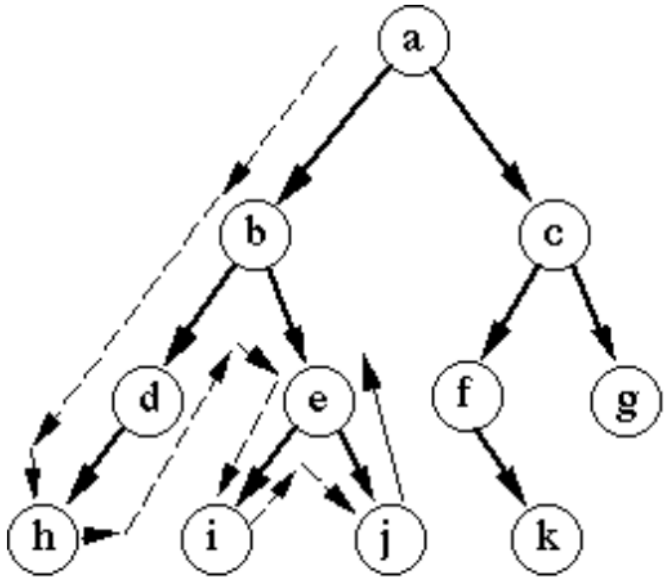
- **Breadth-First Search**
  - iterativ: utilizează coada



Breadth-first search

# Depth-First Search (recursiv)

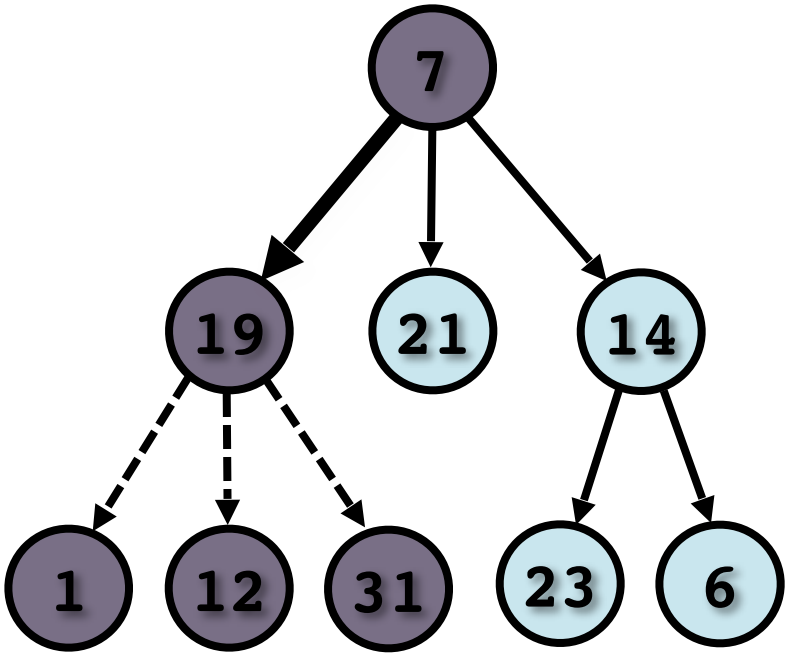
```
DFS(node)
{
    for each child c of node
        DFS( c );
    print the current node;
}
```




Depth-first search

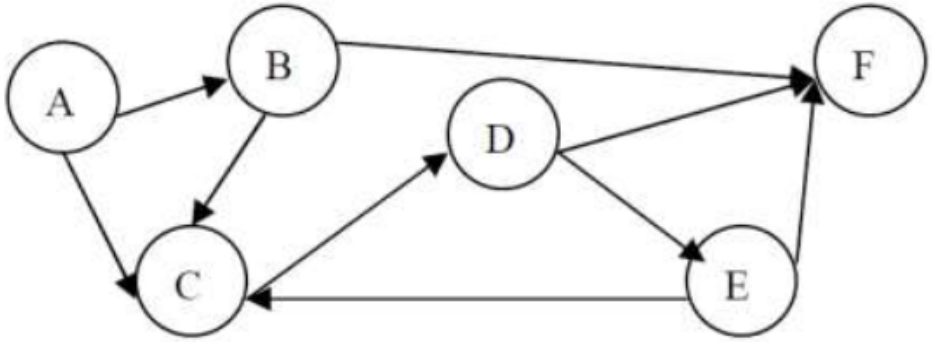
# DFS iterativ : stack

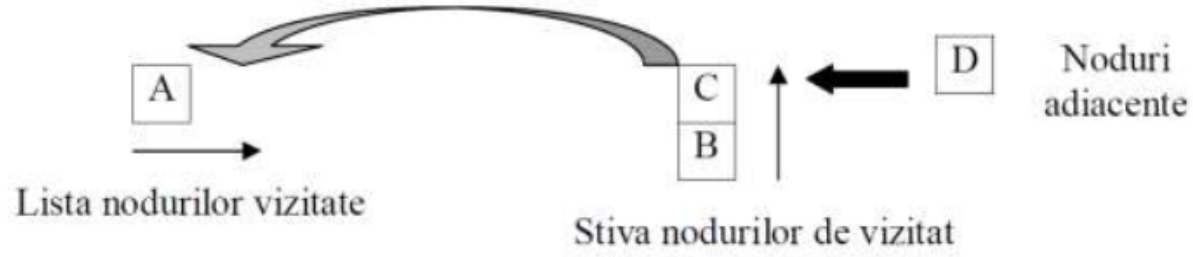
- Stack: 21,14
- Output: 7, 19 , 1, 12, 31



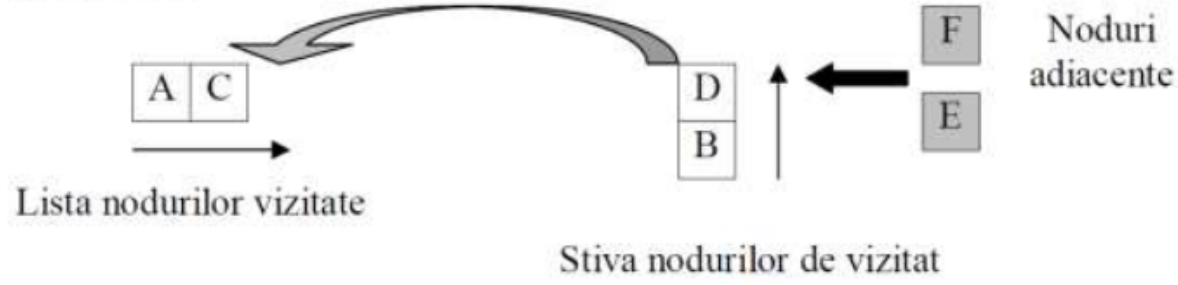

```
DFS(node)
{
  stack ← node
  while stack not empty
    v ← stack
    print v
    for each child c of
v
      stack ← c
}
```

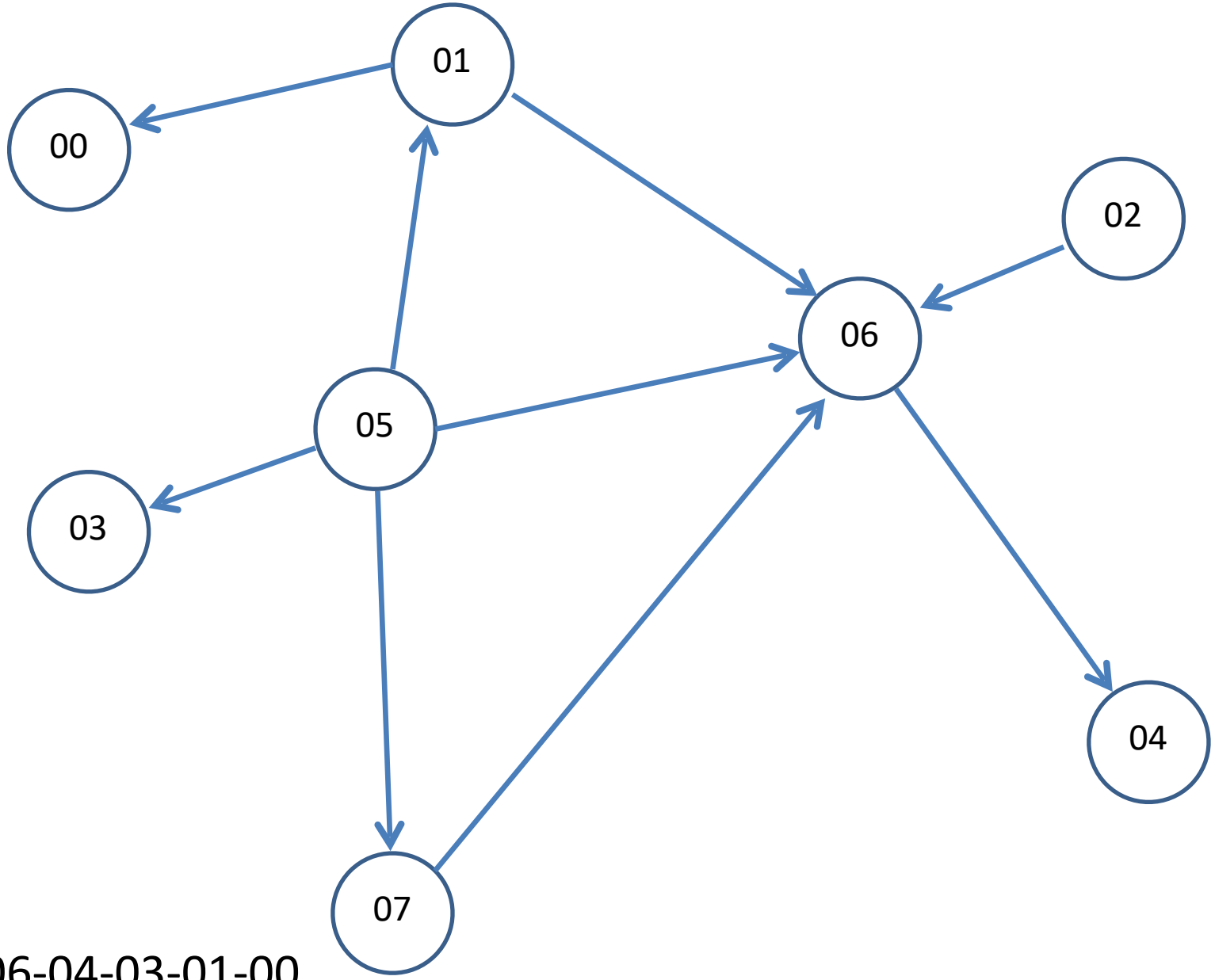
# traversarea în adâncime (**depth-first traversal**)




se scoate primul nod din stivă, C, şi cum acesta nu a fost vizitat (nu se afla în lista nodurilor vizitate) este trecut acum în listă. Nodurile sale adiacente, doar D, sunt trecute în stivă;

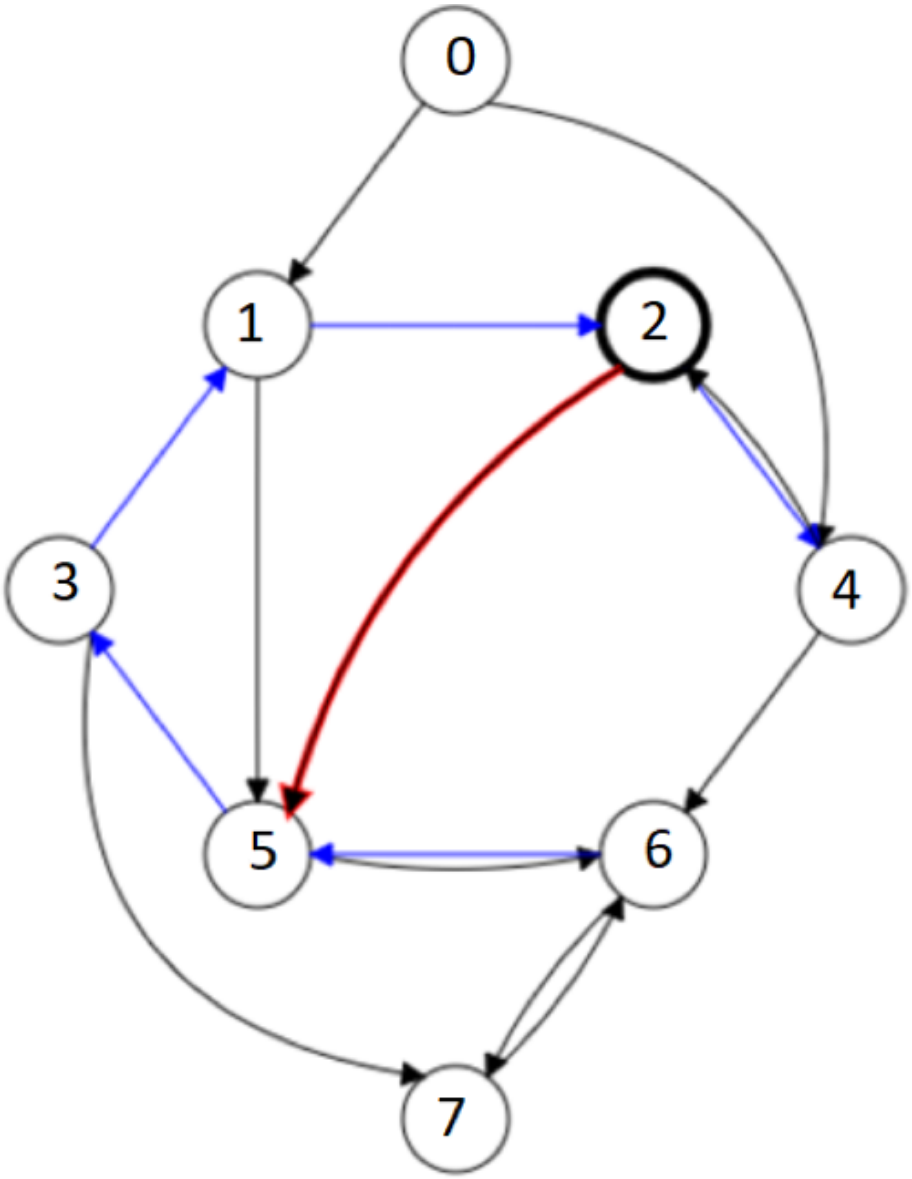





DFS:

05-07-06-04-03-01-00

# Depth-First Search




| | Parent | Visited |
|---|---|---|
| 0 | | f |
| 1 | 3 | T |
| 2 | 1 | T |
| 3 | 5 | T |
| 4 | 2 | T |
| 5 | 6 | T |
| 6 | | T |
| 7 | | f |

DFS(6)

DFS(5)

DFS(3)

DFS(1)

DFS(2)

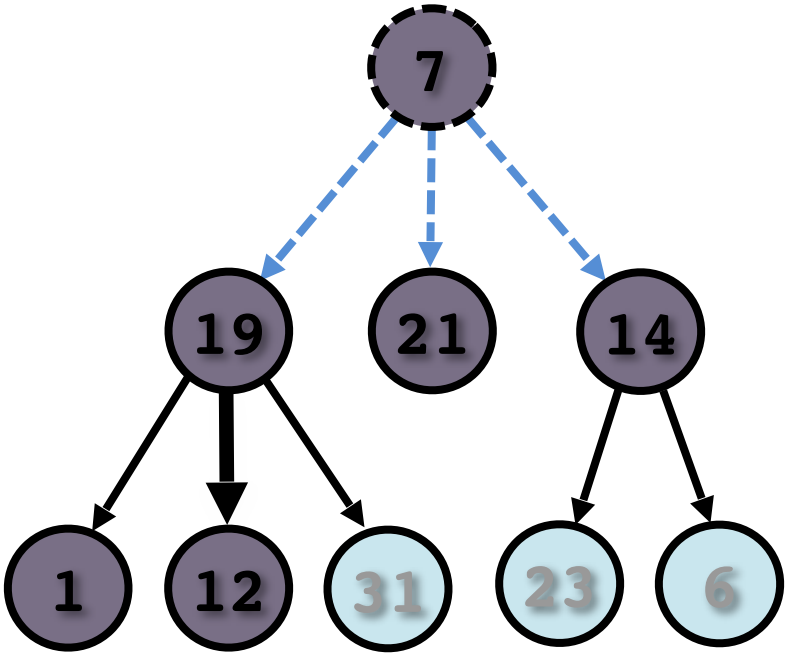
DFS(4)

Vertex 5 already visited.

http://www.cs.usfca.edu/~galles/visualization/DFS.html

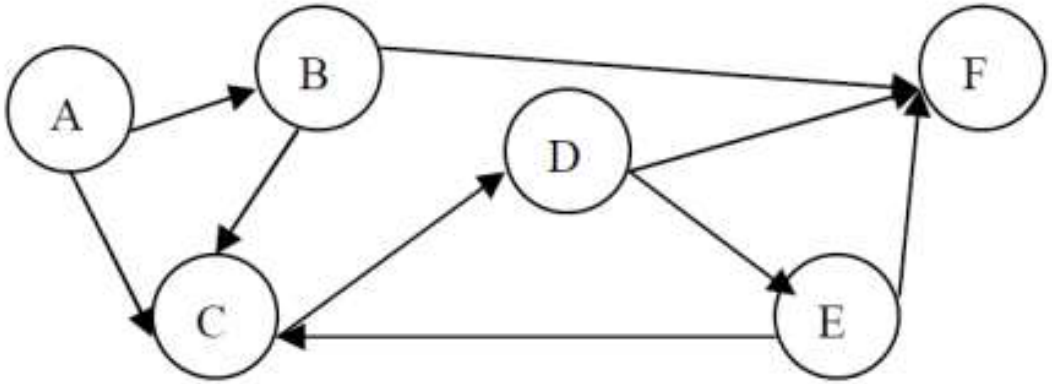
# BFS - iterative

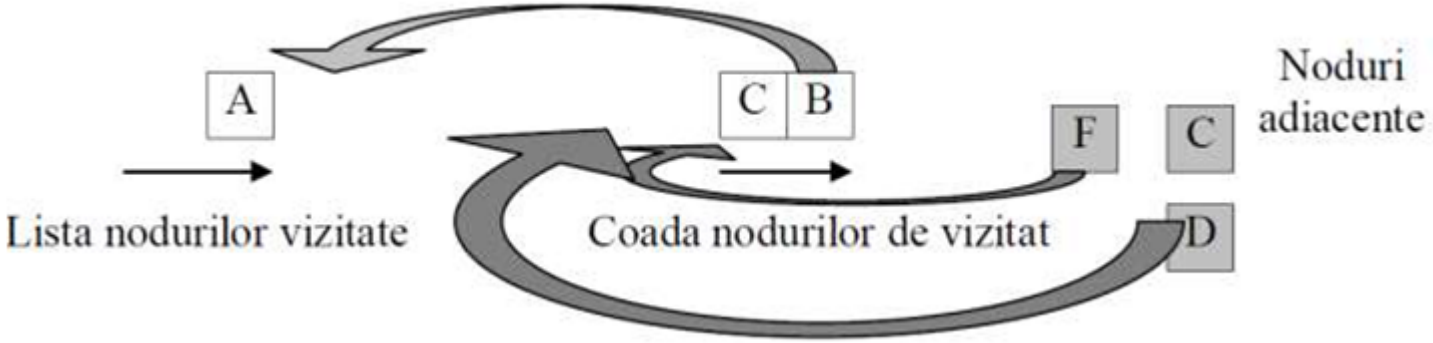
- Queue: 21, 14, 1, 12, 31
- Output: 7, 19

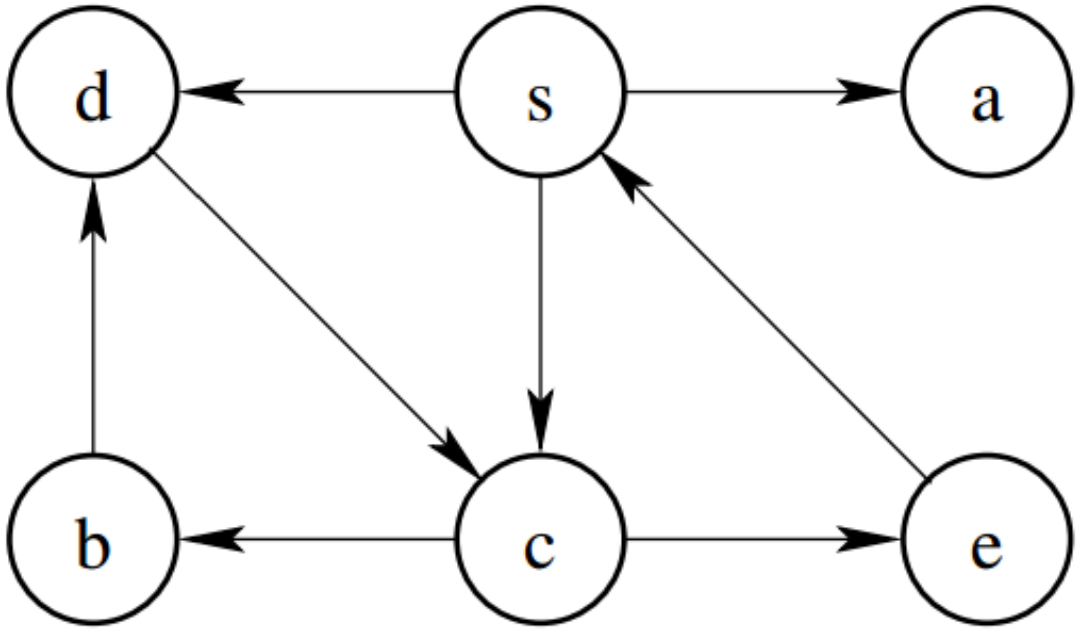



```
BFS(node)
{
  queue ← node
  while queue not empty
    v ← queue
    print v
    for each child c of
v
      queue ← c
}
```

# traversarea în latime (**breadth-first traversal**)






Breadth First Search: s a c d e b

Depth First Search: s a c e b d

# tema: parcurgere BFS, DFS