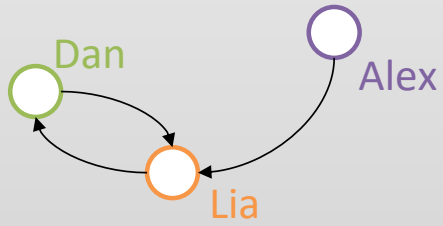
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Curs 7 SDA: Grafuri

Definitie. Tipuri de grafuri. Terminologie. Reprezentari. Parcurgeri. Aplicatii

Definitie

- Un graf consta dintr-o multime de noduri si o multime de muchii, muchiile = relatii intre noduri.
- Arborii si listele sunt cazuri speciale de grafuri.




```
V = {Dan,Lia,Alex}
E = {(Alex,Lia),
     (Dan,Lia),
     (Lia,Dan)}
```

Exemple de utilizare

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Relatii sociale (retele sociale)

Noduri: indivizi

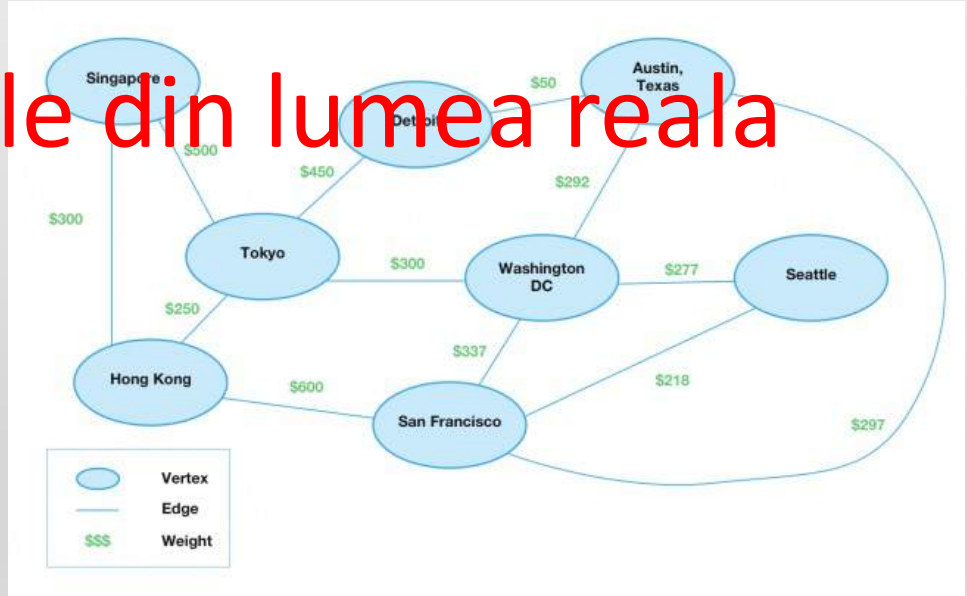
Muchii: relatii de prietenie, like, follow, etc.

http://www.messersmith.name/wordpress/tag/friend-wheel/

Retele de transport

Noduri: aeroporturi, intersectii, porturi

Muchii: zboruri, drumuri, rute de navigatie



https://koenig-media.raywenderlich.com/uploads/2017/01/graph2-2.png

... si multe alte exemple din lumea reala

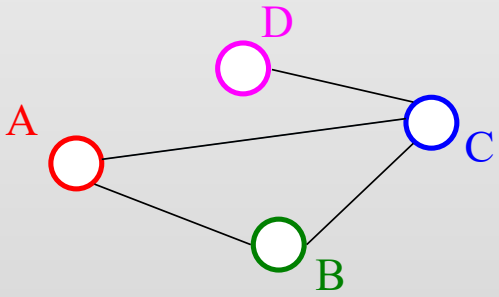
Definitie

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

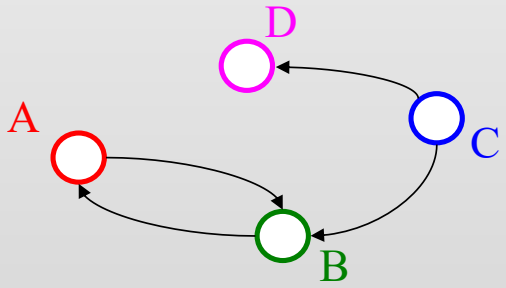
- Un **graf G=(V,E)** – pereche ordonata de multimi care cuprinde:
  - o multime **V** finita si nevida de ***varfuri*** (***noduri***)
  - o multime **E** de perechi (ordonate sau neordonate) de elemente din V numite:
    - ***muchii*** – daca sunt perechi neordonate (graf neorientat)
    - ***arce*** – daca sunt perechi ordonate (graf orientat)
  - Fiecare muchie leaga o pereche de varfuri din V.
- **Ordinul** unui graf: numarul de varfuri $n = |V|$
- Numarul de muchii $e = |E|$
  - e poate lua valori intre 0 si $|V|^2-|V|$

Tipuri de grafuri

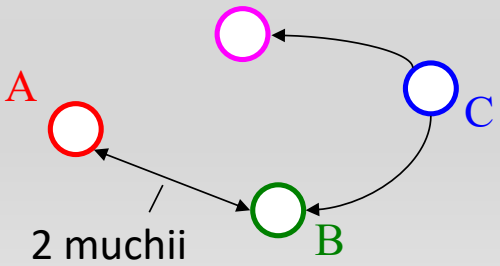
Graf **neorientat**– graf ale carui muchii nu au o directie (implicit un graf e neorientat)



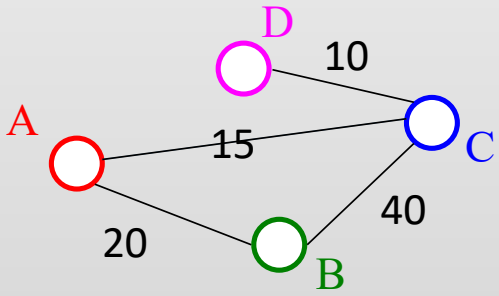
Graf **orientat** – un graf ale carui muchii au o directie definite de la un varf la alt varf.

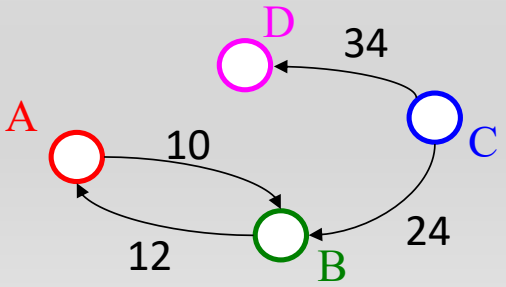


sau



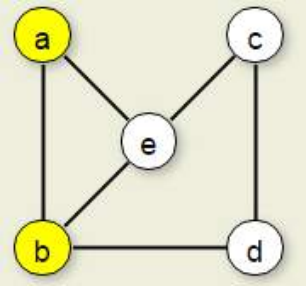
Graf **etichetat** – un graf care are atasata o eticheta, o pondere sau un cost la fiecare muchie. (poate fi neorientat sau orientat)

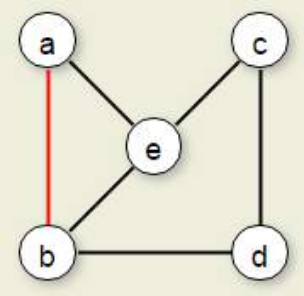




Notiuni fundamentale

- O muchie care leaga varfurile a si b ale unui graf se noteaza (a,b)
  - O astfel de muchie este **incidenta** la varfurile a si b.
  - Varfurile a si b sunt varfuri **adiacente** sau **vecine.**





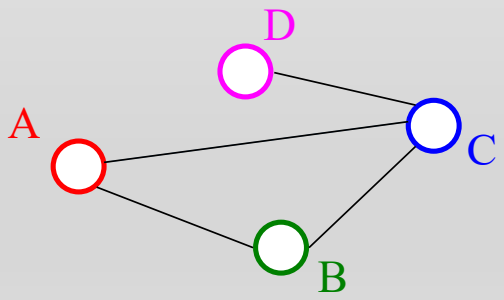
- Nr maxim de muchii pentru un graf neorientat: $(|V|*(|V|-1))/2 = O(|V|^2)$
- Nr maxim de muchii pentru un graf orientat: $|V|*(|V|-1) = O(|V|^2)$

Notiuni fundamentale

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Intr-un graf neorientat:

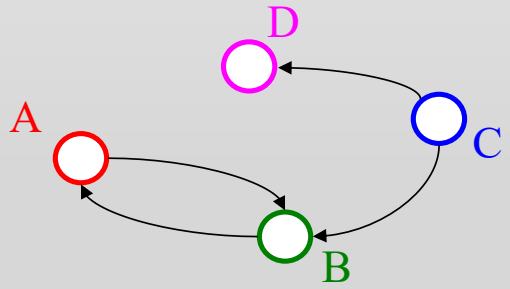
- **gradul unui varf** este numarul de muchii incidente acelui varf.
- Cat este suma gradelor tuturor nodurilor din graful G in functie de numarul de muchii? (Handshaking lemma)



Grad(A) = 2;
Grad(C) = 3

Intr-un graf orientat:

- **Gradul exterior** (out degree) al unui varf este dat de numarul de vecini adiacenti de la el (numarul de muchii care ies din varf).
- **Gradul interior** (in degree) este numarul de vecini adiacenti varfului (numarul de muchii care intra in varf)

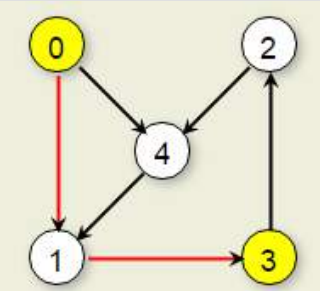


GradExterior(A) = 1; GradInterior(A) = 1;
GradExterior(C) = 2; GradInterior(C) = 0;

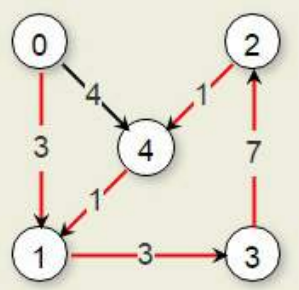
Notiuni fundamentale

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- **Drum (cale)** – o secventa de varfuri v_1, v_2, … v_n formeaza un drum de lungime n-1 daca exista muchii de la v_i la v_{i+1} pentru $1 \leq i < n$
- **Lungimea drumului** – numarul de muchii ale drumului.
- **Drum simplu** – drum cu varfuri distincte.



Drum simplu de la 0 la 3

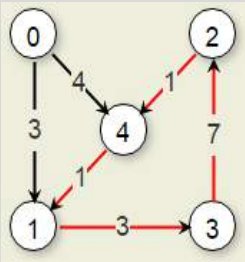


Drumul 0, 1, 3, 2, 4, 1
nu este simplu

Notiuni fundamentale

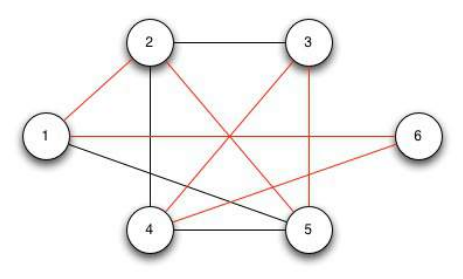
- **Ciclu** – drum de lungime minim 3 care conecteaza un varf v_1 la el insusi (drum care incepe si se termina cu acelasi nod).

Ciclu **simplu**: ciclu care are un drum simplu, exceptie facand primul si ultimul varf care sunt identice.



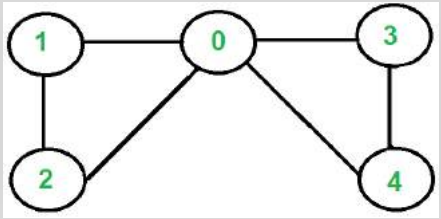
Ciclu simplu: 1, 3, 2, 4, 1

Ciclu **hamiltonian**: ciclu simplu care trece prin toate nodurile grafului G, exact o dată.



Ciclu hamiltonian:
1, 2, 5, 3, 4, 6, 1

Ciclu **eulerian**: ciclu care trece prin toate muchiile grafului G, exact o dată.

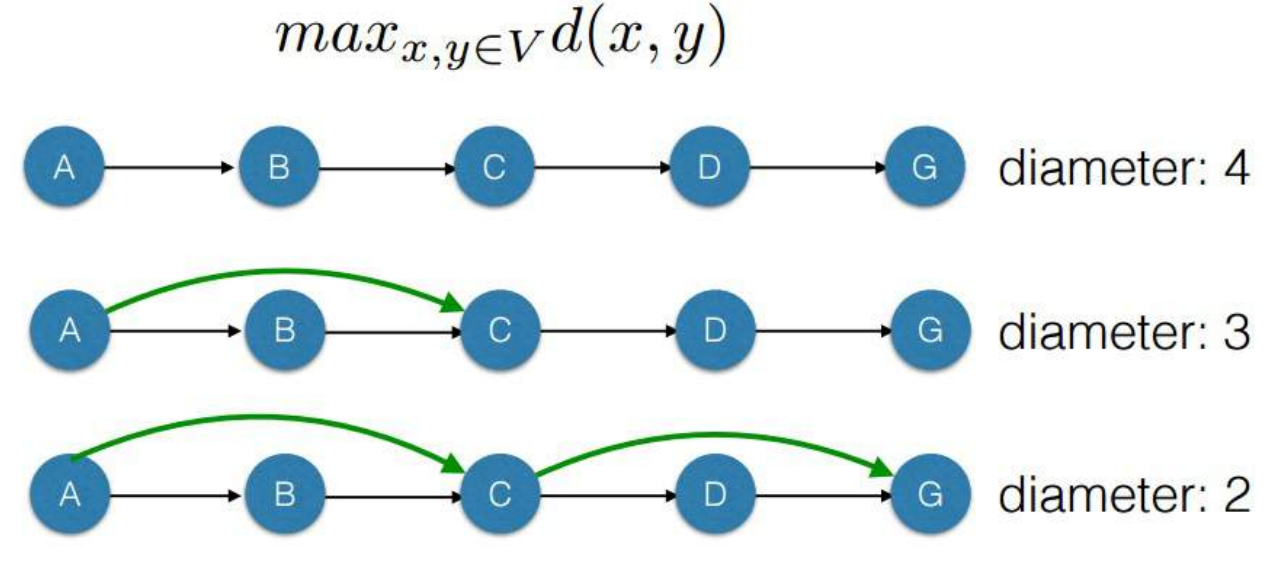


Ciclu eulerian:
2,1,0,3,4,0,2

Notiuni fundamentale

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

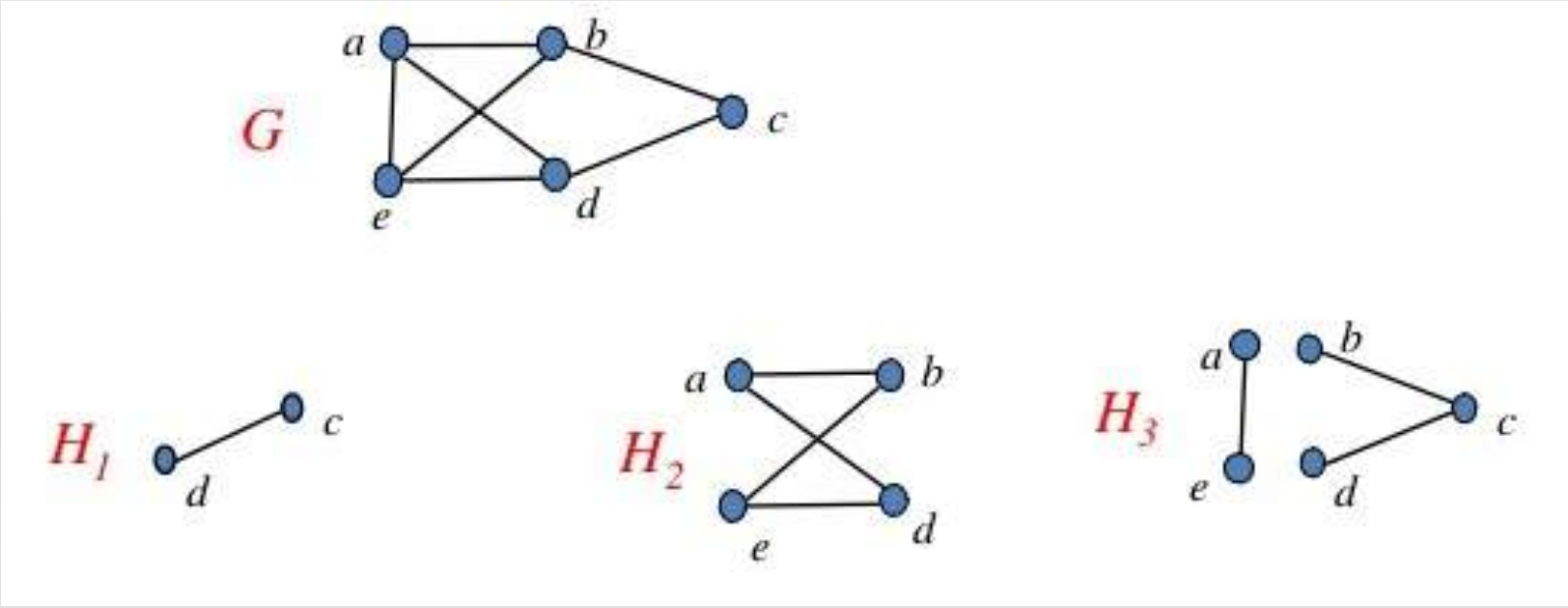
- **Diametrul unui graf** este lungimea drumului cel mai scurt dintre cele mai indepartate noduri ale grafului.



http://www.st.ewi.tudelft.nl/~hauff/BDP-Lectures/11_graph.pdf

Subgraf

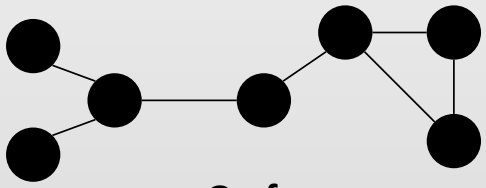
- Fiind dat graful G=(V,E), **Gs= (Vs, Es)** este un **subgraf** al lui G daca:
  $$\boldsymbol{Vs \subseteq V \; si} \; \forall \; (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{E}: (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{Es} \; \Leftrightarrow \boldsymbol{x \in Vs \; si \; y \in Vs}$$
- ex H1, H2, H3 sunt subgrafuri ale lui G.



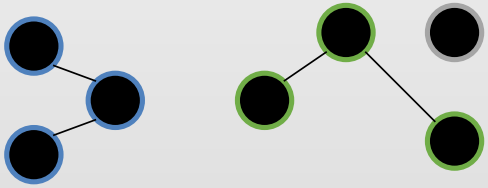
Notiuni fundamentale

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- **Un graf neorientat este conex** daca intre orice pereche de varfuri (u, v) din graf exista cel putin un drum.

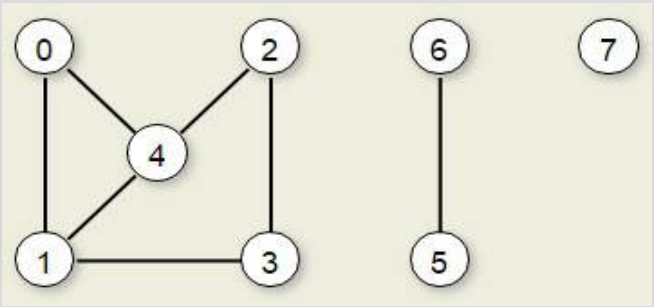


Graf conex



Graf neconex

- **Componente conexe** – subgrafurile maximal conectate ale unui graf neorientat.

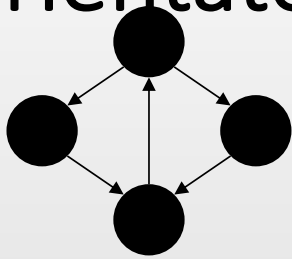


Graf neorientat care are 3 componente conexe.

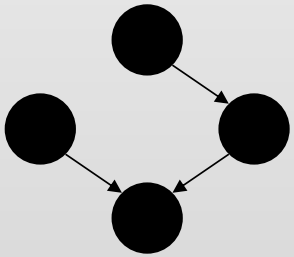
Notiuni fundamentale grafuri orientate

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

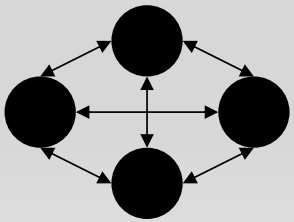
- Un graf **orientat** este **puternic conex** daca exista un drum intre oricare doua noduri din graf.



- Un graf **orientat** este **slab conex** daca exista un drum intre oricare noduri din graf – fara a lua in considerare directia muchiilor.

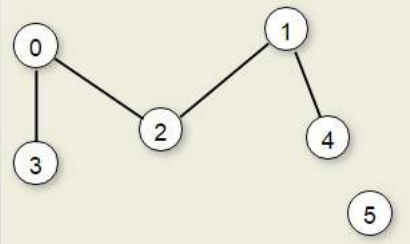


- Un graf **orientat** este **complet** daca exista o muchie intre oricare doua varfuri.

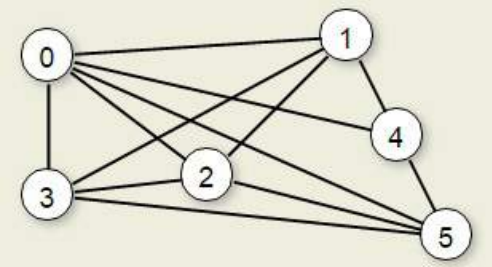


Notiuni fundamentale – grafuri neorientate

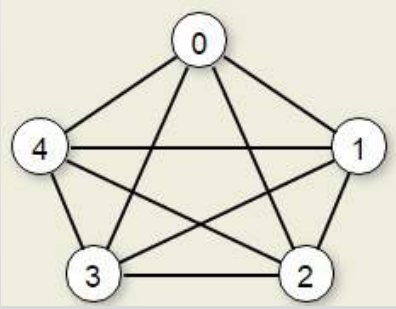
Graf rar (sparse graf) – graf cu putine muchii



Graf dens – graf cu multe muchii

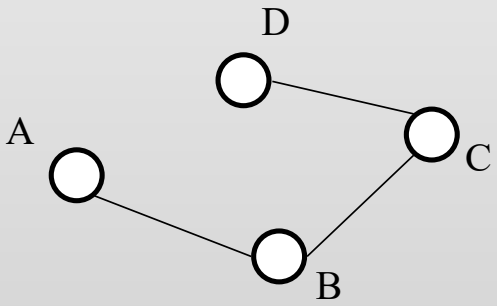


Graf complet – graf care are toate muchiile posibile.

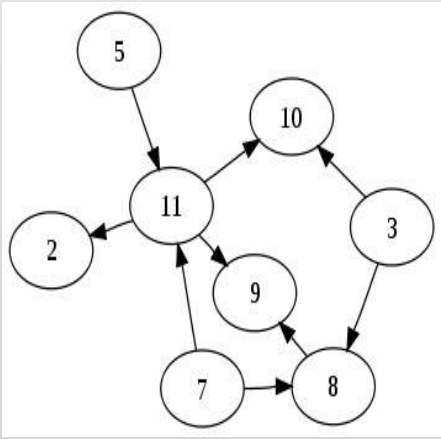


Notiuni fundamentale

- **Graf aciclic** – graf fara cicluri.
- **Graf aciclic directionat (orientat)** – un graf directionat fara cicluri (a.k.a. DAG).



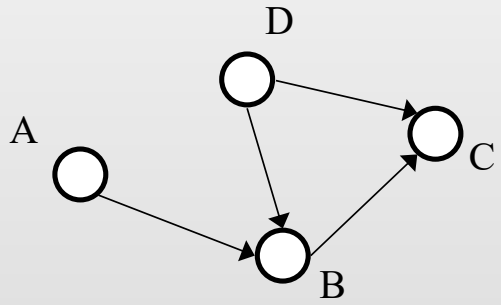
Graf aciclic neorientat



Graf aciclic orientat (DAG)

Drumuri si cicluri in grafuri orientate

Exemplu:

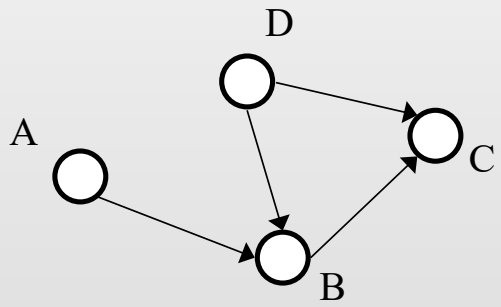


Exista drum de la A la D ?

Exista cicluri in graf ?

Drumuri si cicluri in grafuri orientate

Exemplu:

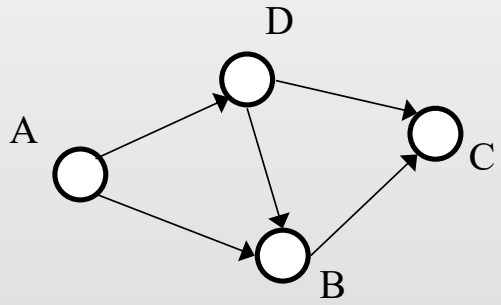


Exista drum de la A la D ? Nu

Exista cicluri in graf ? Nu

Drumuri si cicluri in grafuri orientate

Exemplu:

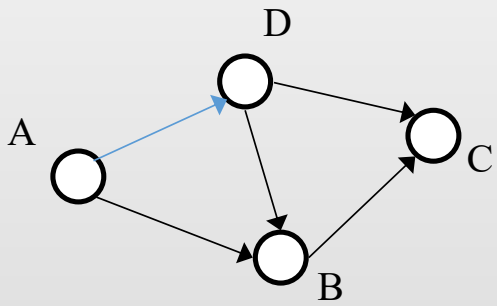


Exista drum de la A la D ?

Exista cicluri in graf ?

Drumuri si cicluri in grafuri orientate

Exemplu:

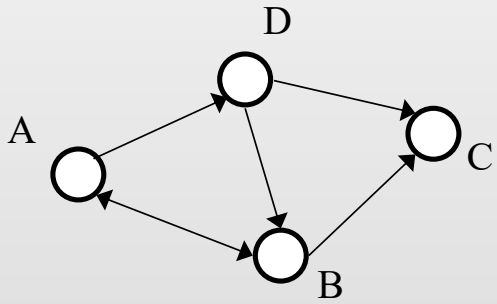


Exista drum de la A la D ? Da

Exista cicluri in graf ? Nu

Drumuri si cicluri in grafuri orientate

Exemplu:

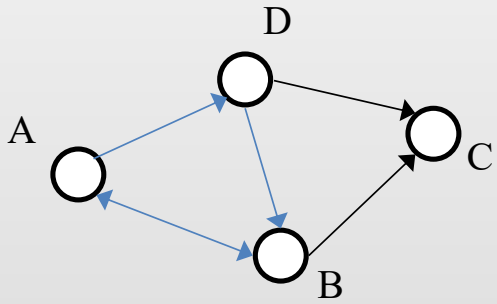


Exista drum de la A la D ?

Exista cicluri in graf ?

Drumuri si cicluri in grafuri orientate

Exemplu:



Exista drum de la A la D ? Da

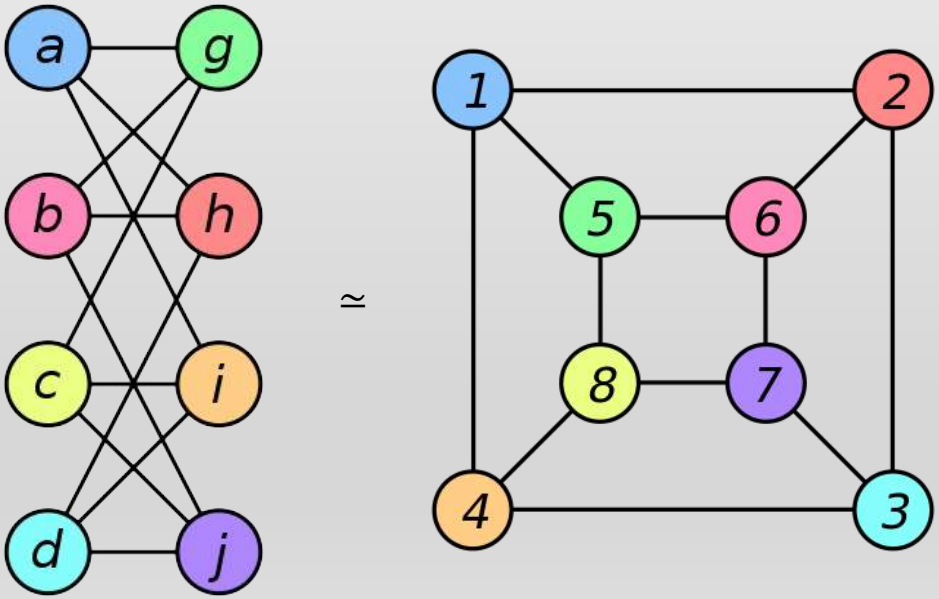
Exista cicluri in graf ? Da

Grafuri izomorfe

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- G=(V,E) si G'=(V',E') sunt **izomorfe** (notatie $G \simeq G'$), daca:

$$\exists\, f: V \to V' \; - bijectiva, a.i. \forall\, (x,y) \in E, (f(x), (f(y)) \in E'$$



$f(a) = 1$
$f(b) = 6$
$f(c) = 8$
$f(d) = 3$
$f(g) = 5$
$f(h) = 2$
$f(i) = 4$
$f(j) = 7$

https://en.wikipedia.org/wiki/Graph_isomorphism

Modalitati de reprezentare a grafurilor

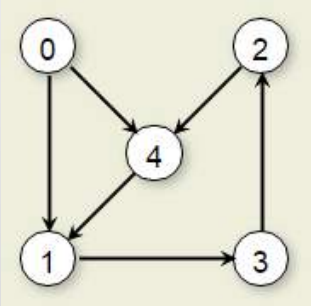
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

1. **Matrice de adiacenta**:
   - Matrice de dimensiune |V|x|V|
   - Randul *i* din matrice contine intrarile pentru varful v_i
   - Coloana *j* din randul i este marcata (fie 1, True, pondere, cost) daca exista o muchie de la varful v_i la varful v_j.
2. **Lista de adiacenta**:
   - Sir (vector) care contine |V| liste inlantuite
   - Pe pozitia *i* din sir contine un pointer la lista varfurilor adiacente varfului v_i.

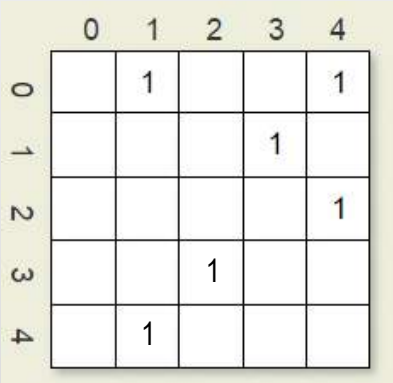
- Ambele reprezentari sunt potrivite atat pentru grafuri orientate cat si pentru grafuri neorientate.

Modalitati de reprezentare a grafurilor

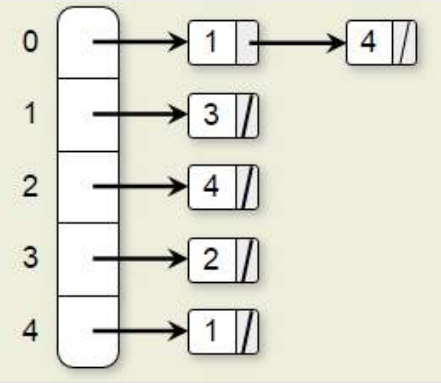
- Exemplu pentru graf orientat:



Graf orientat



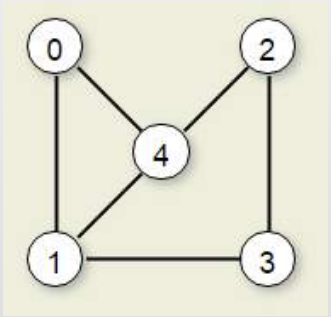
Matricea de adiacenta



Lista de adiacenta

Modalitati de reprezentare a grafurilor

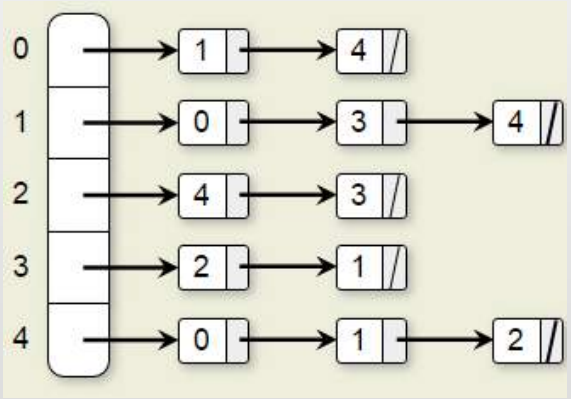
- Exemplu pentru graf neorientat:



Graf neorientat

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

Matricea de adiacenta

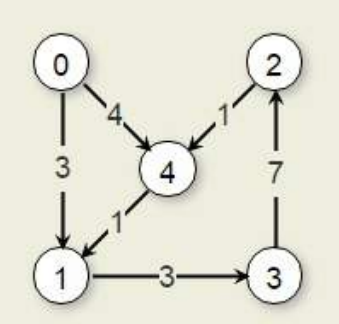


Lista de adiacenta

Fiecare muchie dintre varfurile u si v a unui graf neorientat este reprezentata de doua muchii orientate: una de la u la v si alta de la v la u.

Modalitati de reprezentare a grafurilor

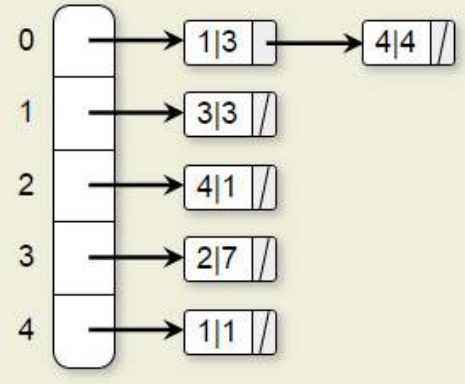
- Exemplu pentru graf etichetat:



Graf etichetat

	0	1	2	3	4
0		3			4
1				3	
2					4
3			7		
4		1			

Matricea de adiacenta



Liste de adiacenta

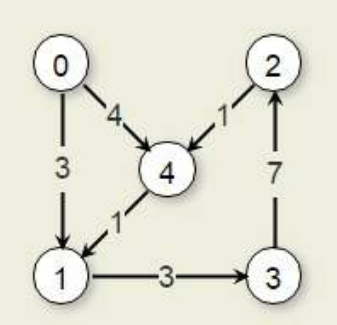
Modalitati de reprezentare a grafurilor

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Care reprezentare este mai eficienta ca spatiu de stocare?
  - Depinde de numarul de muchii
  - Lista de adiacenta stocheaza informatii doar pentru acele muchii care apar in graf
  - Matricea de adiacenta aloca spatiu pentru orice muchie posibila din graf, dar nu necesita o incarcare suplimentara de pointeri
  - Cu cat graful devine mai dens cu atat matricea de adiacenta devine mai eficienta ca spatiu alocat.
  - Cu cat graful este mai rar cu atat este mai indicate o reprezentare cu liste de adiacenta.

Modalitati de reprezentare a grafurilor

- Eficienta reprezentarilor - exemplu



Graf etichetat

- Se da graful din figura alaturata.
- Se cunosc urmatoarele detalii de stocare:
  - Indicele unui varf – 2 bytes
  - Pointer – 4 bytes
  - Eticheta unui arc – 2 bytes

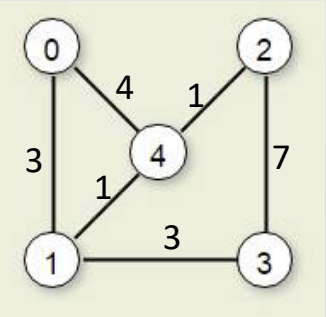
  Care e spatiul necesar pentru stocarea prin matrice de adiacenta si prin liste de adiacenta ?

Matricea de adiacenta: $2|V|^2 = 50$ bytes
Lista de adiacenta: $4|V| + 8|E| = 68$ bytes

Modalitati de reprezentare a grafurilor

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Eficienta reprezentarilor - exemplu



Graf neorientat

- Se da graful din figura alaturata.
- Se cunosc urmatoarele detalii de stocare:
  - Indicele unui varf – 2 bytes
  - Pointer – 4 bytes
  - Eticheta unui arc – 2 bytes.

  Care e spatiul necesar pentru stocarea prin matrice de adiacenta si prin liste de adiacenta ?

Matricea de adiacenta: $2|V|^2 = 50$ bytes
Lista de adiacenta: $4|V| + 2*8\ |E| = 116$ bytes

Parcurgerea unui graf - idee

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Fiind dat un graf oarecare si un nod de pornire v, sa se determine toate nodurile la care se poate ajunge pornind de la v (adica acele noduri de la care exista un drum pana la v)

- Este posibil sa “facem ceva” cand vizitam fiecare nod
- De exemplu sa afisam valorile din noduri, sa marcam un camp etc.

- Poate **raspunde la** ‘Este graful conex?’
- **Problema diferita:** este graful puternic conex?

Ideea de baza a parcurgerii:

- Urmarim vizitarea nodurilor
- Dar “marcam” nodurile dupa ce le vizitam pentru ca parcurgerea sa se termine si pentru ca fiecare nod sa fie parcurs o singura data

Definitie

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- **Parcurgerea unui graf**: procesul de vizitare (actualizare, verificare) a fiecarui nod din graf.
- Parcurgerile sunt clasificate dupa ordinea in care sunt vizitate varfurile din graf:
  - **Parcurgere in adancime** (depth-first search (DFS))
  - **Parcurgere in latime** (breadth-first search (BFS))

Metodologie

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Algoritmii de parcurgere a grafurilor incep cu un varf de start si incearca sa viziteze restul varfurilor incepand de la varful de start.
- Cazuri exceptionale:
  - Incepand de la un varf dat e posibil sa nu se poata ajunge la toate celelalte varfuri – graf ne-conex.
  - Daca exista cicluri in graf – ne asiguram ca algoritmul evita intrarea intr-o bucla infinita de parcurgere.
- Pentru a evita cazurile de mai sus se foloseste o eticheta – VIZITAT pentru a marca varfurile parcurse

Metodologie

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
void graphTraverse(Graph G) {
    int v;
    for (v=0; v<G.nodeCount(); v++)
        G.setValue(v, null); // set all nodes to not visited
    for (v=0; v<G.nodeCount(); v++)
        if (G.getValue(v) != VISITED)
                doTraversal(G, v);
}
```

Functia doTraversal se poate implementa folosind:

- **parcurgere in adancime**
- **parcurgere in latime**

Parcurgerea in latime (BFS)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- *Breadth first search (BFS)*
- Se poate aplica atat pe grafuri orientate cat si pe grafuri neorientate
- Parcurge toate varfurile conectate la nodul curent inainte de a trece mai departe
- Foloseste o ***coada*** pentru a mentine ordinea de parcurgere
- Verifica daca un nod a fost vizitat inainte sa il introduca in coada

Parcurgere in latime - pseudocod

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
BFS(G, s)  //doTraverse
 1  for each vertex u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

- Culori pentru a tine evidenta starii de parcurgere a nodurilor
- Calculeaza distanta de la nodul sursa la toate nodurile conectate cu el
- Produce un arbore de parcurgere in latime pentru fiecare nod sursa, *s*
- Expandeaza frontiera dintre nodurile descoperite si cele nedescoperite in mod uniform pe latimea ei (toate nodurile de la dist *k* sunt descoperite inainte de cele de la dist *k+1*)

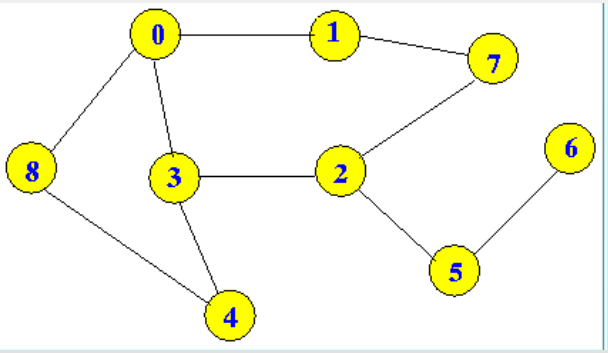
Parcurgere in latime

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

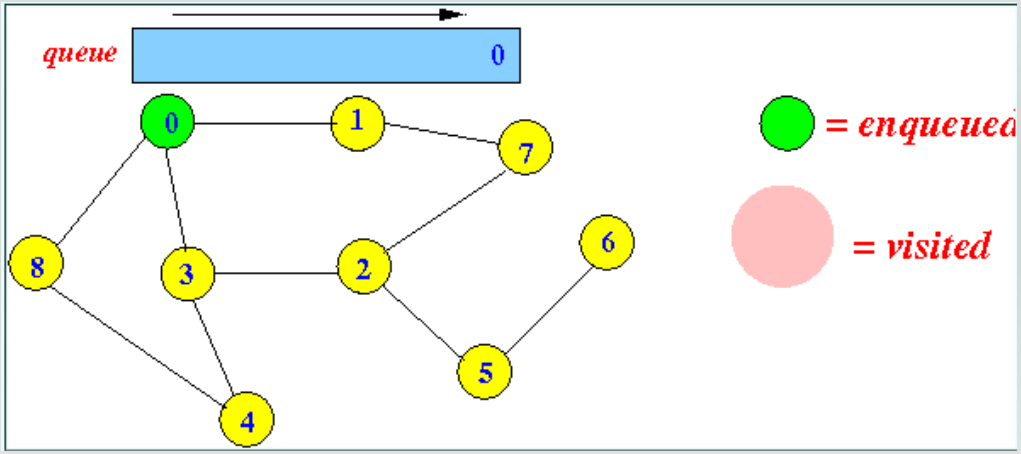
- Atributul *distanta* al fiecarui nod este folosit pentru a cauta cel mai scurt drum intre doua noduri din graf.
- La inceputul algoritmului distanta pentru fiecare nod este infinit – ceea ce semnifica faptul ca nodul respectiv nu a fost vizitat.
- Atributul *parinte* al fiecarui nod poate fi folosit pentru a accesa nodurile din drumul cel mai scurt (ne da arborele, sau padurea de arbori de parcurgere in latime)
- *Complexitatea?*

Parcurgere in latime - trasare

1. Se da graful neorientat:

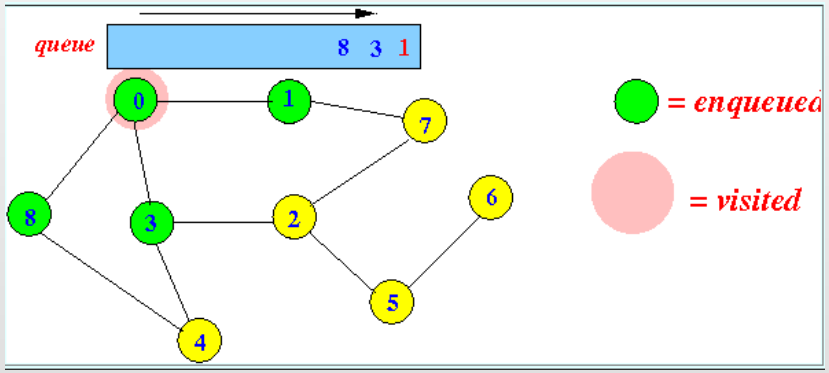


2. Starea initiala: nodul 0 este adaugat in coada



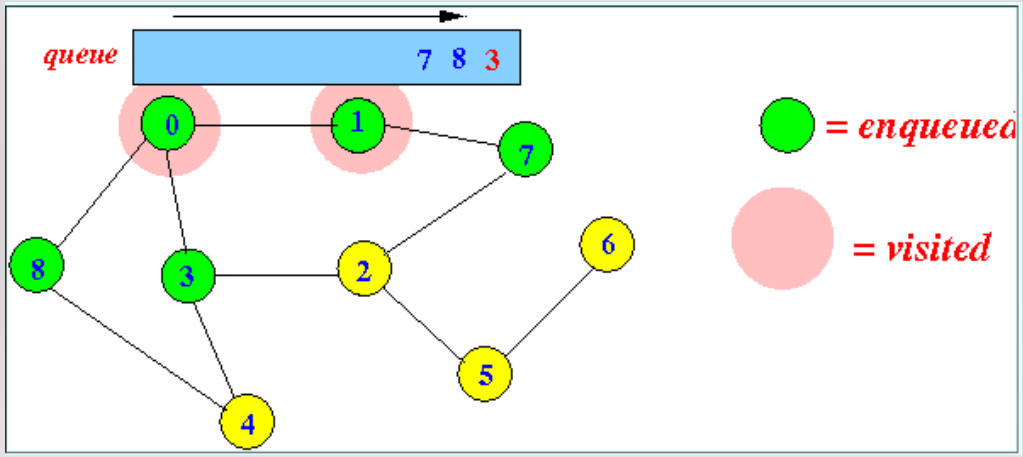
Parcurgere in latime - trasare

3. Starea dupa vizitarea nodului 0:



Se pun in coada vecinii nevizitati: 1, 3, 8
Apoi se viziteaza primul nod din coada, adica nodul 1.

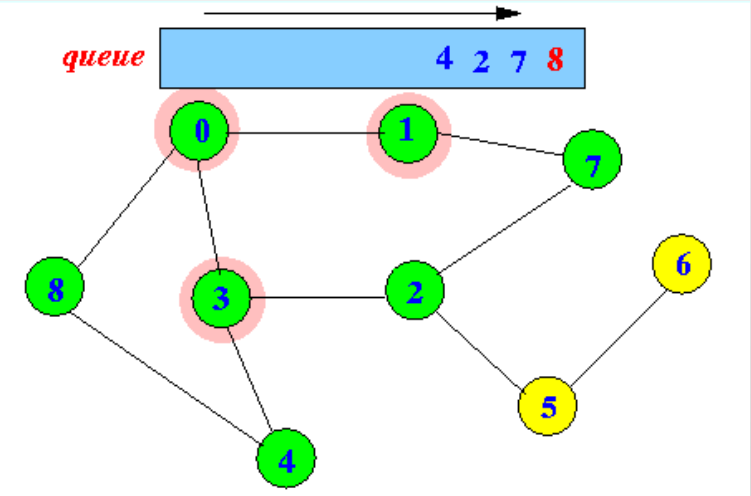
4. Starea dupa vizitarea nodului 1:



Se pun in coada vecinii nevizitat ai nodului 1: 7
Apoi se viziteaza primul nod din coada, adica nodul 3.

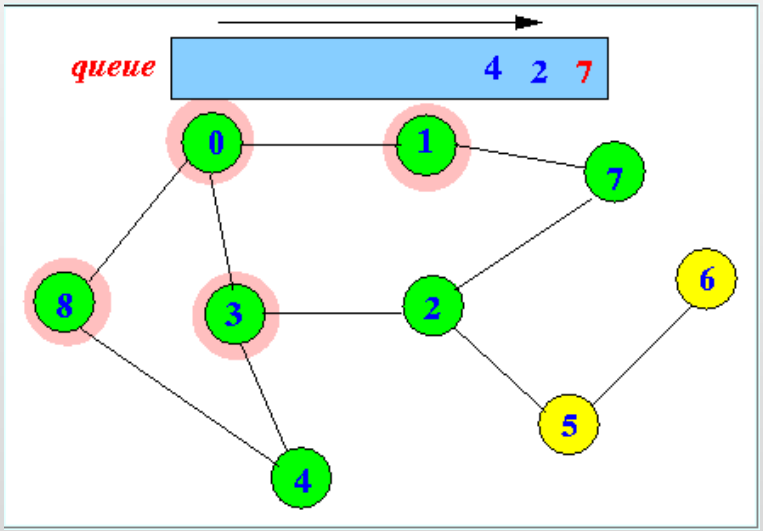
Parcurgere in latime - trasare

5. Starea dupa vizitarea nodului 3:



Se pun in coada vecinii nevizitati ai nodului 3, adica: 2 si 4
Apoi se viziteaza primul nod din coada, adica nodul 8.

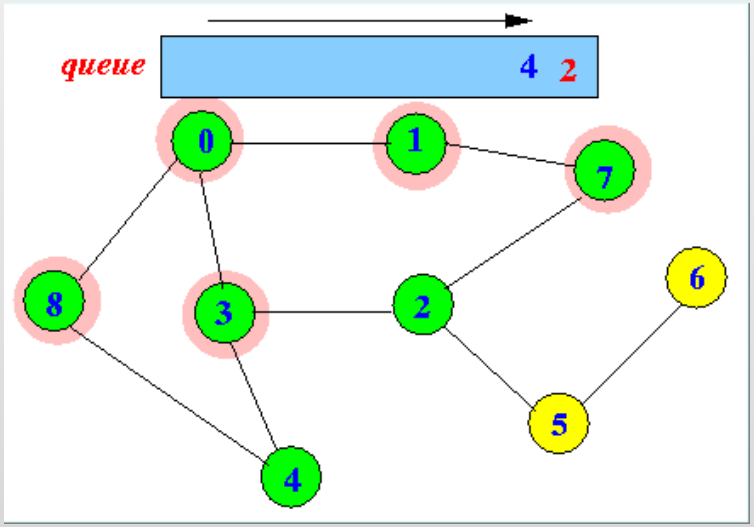
6. Starea dupa vizitarea nodului 8:



Se pun in coada vecinii nevizitati ai nodului 8: - nu sunt
Apoi se viziteaza primul nod din coada, adica nodul 7.

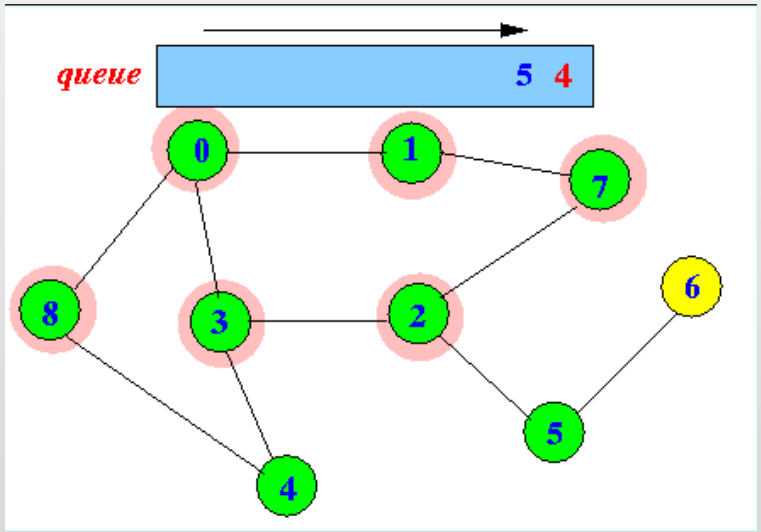
Parcurgere in latime - trasare

7. Starea dupa vizitarea nodului 7:



Se pun in coada vecinii nevizitati ai nodului 7: - nu sunt
Apoi se viziteaza primul nod din coada, adica nodul 2.

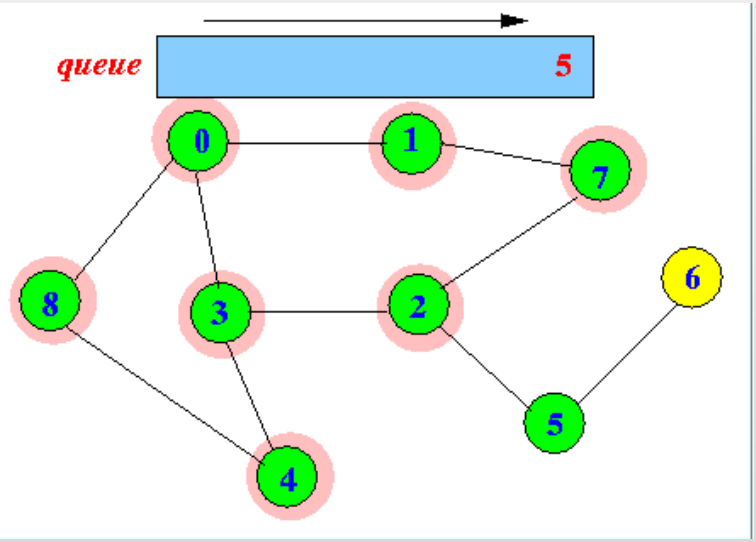
8. Starea dupa vizitarea nodului 2:



Se pun in coada vecinii nevizitati ai nodului 2: nodul 5.
Apoi se viziteaza primul nod din coada, nodul 4.

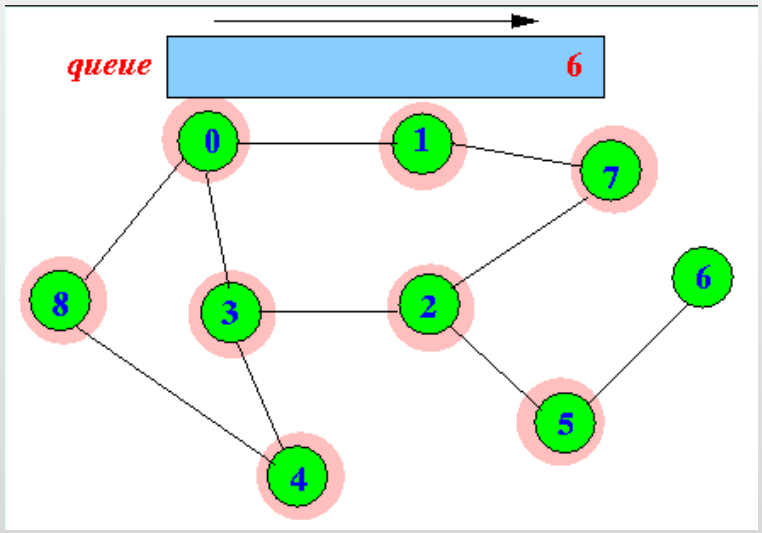
Parcurgere in latime - trasare

9. Starea dupa vizitarea nodului 4:



Se pun in coada vecinii nevizitati ai nodului 4: nu sunt
Apoi se viziteaza primul nod din coada, adica nodul 5.

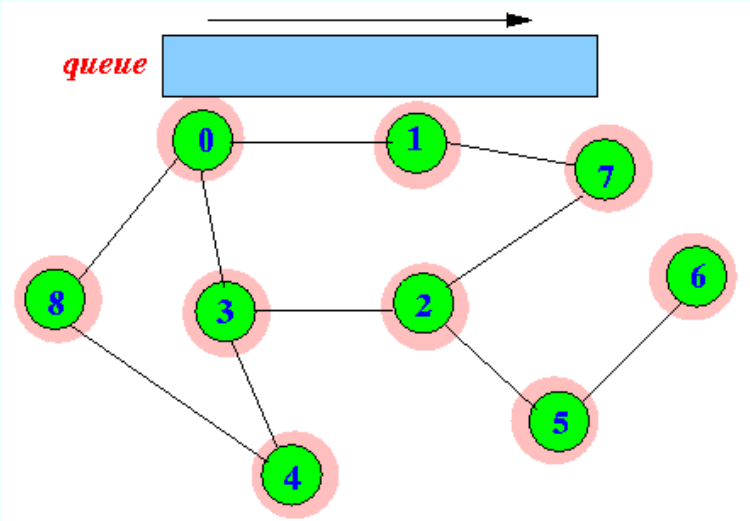
10. Starea dupa vizitarea nodului 5:



Se pun in coada vecinii nevizitati ai nodului 5: nodul 6
Apoi se viziteaza primul nod din coada, adica nodul 6.

Parcurgere in latime - trasare

11. Starea dupa vizitarea nodului 6

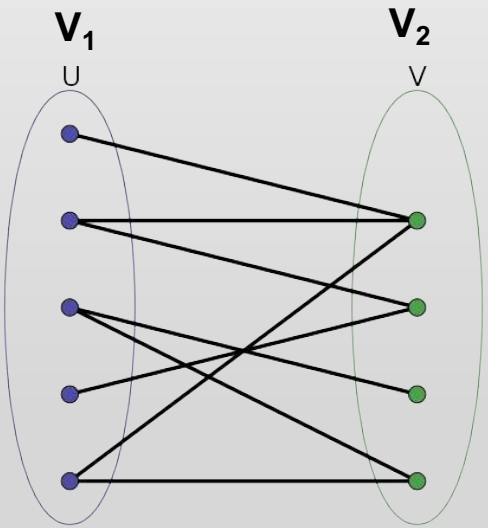


Coada este vida ! Algoritmul s-a terminat.

Parcurgerea in latime - aplicatii

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Gasirea drumurilor de lungime minima
- Graf bipartit
  - Gasirea unui ciclu de lungime impara



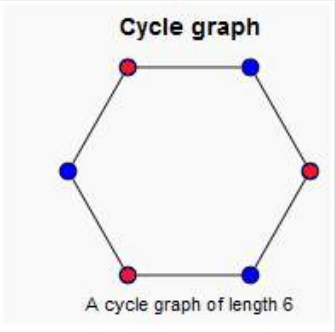
G=(V,E) – bipartit iff
V = V_1 ∪ V_2, V_1 ∩ V_2= ∅,
$\forall(u,v) \in E \Rightarrow u \in V_1, v \in V_2 \text{ or } u \in V_2, v \in V_1$

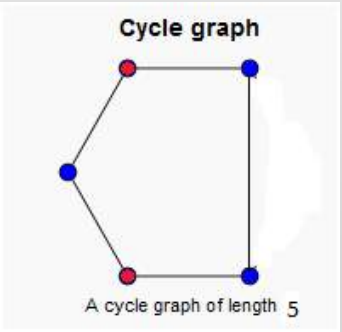
Un graf bipartit nu poate contine un ciclu de lungime impara!!

Graf bipartit – poate fi colorat in 2 culori ?

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
1  Assume graph G is connected. Otherwise, we can run the algorithm for each connected
2  Let q be an empty queue
3
4  Pick any vertex s ∈ V and color it Red
5  q.enqueue(s)
6
7  while !q.empty()
8    u = q.dequeue()
9    foreach v in u.adjList:
10     if v.color is nil:
11       v.color = (u.color == Red) ? Black : Red
12       q.enqueue(v)
13     elif v.color == u.color:
14       return "Not Bipartite"
15 return "Bipartite"
```





Se poate aplica si DFS!

Tema: Cum se modifica algoritmul pentru a detecta ciclul de lungime impara?

Parcurgere in adancime (DFS)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- *Depth first search (DFS)*
- Se poate aplica pe grafuri orientate si neorientate
- Muchiile sunt explorate pornind din varful *v* cel mai *recent* descoperit, care mai are inca muchii neexplorate ce pleaca din el
- Cand toate muchiile care pleaca din *v* au fost explorate, parcurgerea revine pe propriile urme pentru a explora muchiile care pleaca din varful din care a fost descoperit *v* (parintele lui *v*)
- Parcurgerea foloseste o structura de tip ***stiva*** pentru a mentine ordinea de parcurgere

Pseudocod – parcurgere in adancime

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- **Intrare**: Un graf G si un varf *v* din G
- **Iesire**: Arborele de parcurgere in adancime, care contine varfurile la care se poate ajunge pornind de la *v*
- Varianta iterativa:

```
1  procedure DFS-iterative(G,v):
2      let S be a stack
3      S.push(v)
4      while S is not empty
5            v = S.pop()
6            if v is not labeled as discovered:
7                label v as discovered
8                for all edges from v to w in
G.adjacentEdges(v) do
9                    S.push(w)
```

Pseudocod – parcurgere in adancime

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- **Intrare**: Un graf G si un varf *u* din G, de pornire
- **Iesire**: Arborele de parcurgere in adancime, care contine varfurile la care se poate ajunge pornind de la u
  - Inregistreaza momentele de timp – descoperire, finalizare
- Varianta recursiva:

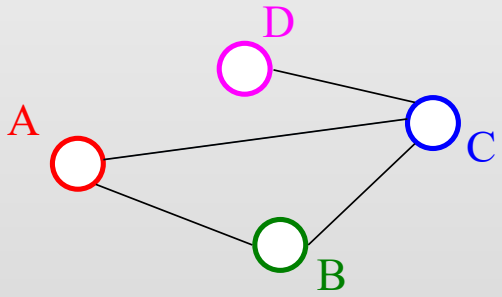
```
DFS-VISIT(G, u)
1   time = time + 1                  // white vertex u has just been discovered
2   u.d = time
3   u.color = GRAY
4   for each v ∈ G.Adj[u]            // explore edge (u, v)
5       if v.color == WHITE
6           v.π = u
7           DFS-VISIT(G, v)
8   u.color = BLACK                  // blacken u; it is finished
9   time = time + 1
10  u.f = time
```

Eficienta?

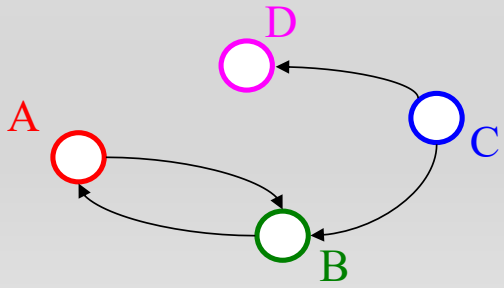
Parcurgere in adancime - trasare

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Sa se realizeze la tabla evolutia parcurgerii in adancime pentru cele doua grafuri de mai jos:
- Graf neorientat:



- Graf orientat:



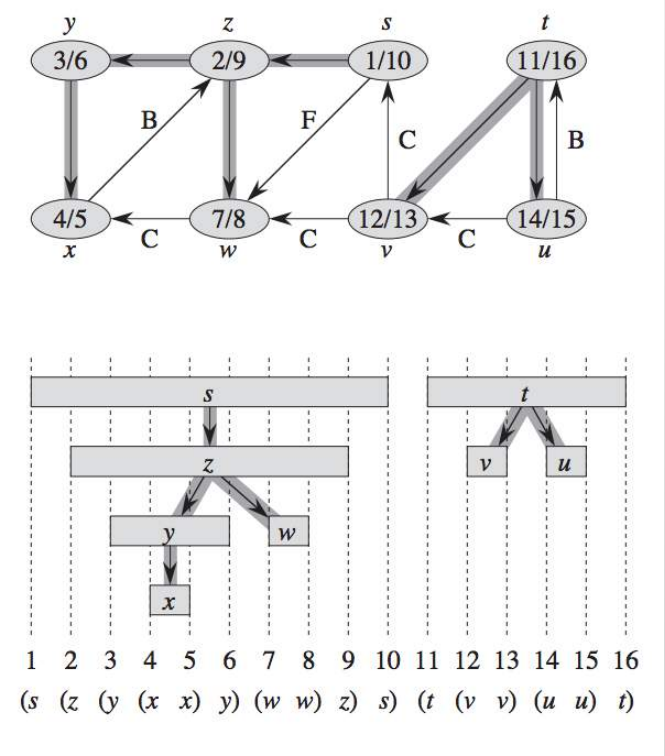
Proprietati ale parcurgerii in adancime

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Cautarea in adancime ofera multe informatii despre structura unui graf.
- ***Structura de paranteza*** a timpilor de descoperire si terminare:
  - Daca reprezentam descoperirea unui varf u printr-o paranteza deschisa ***(u***, si finalizarea sa printr-o paranteza inchisa ***u)***, atunci istoria descoperirilor si a finalizarilor formeaza o expresie bine formata (i.e. parantezele sunt corect imperecheate)

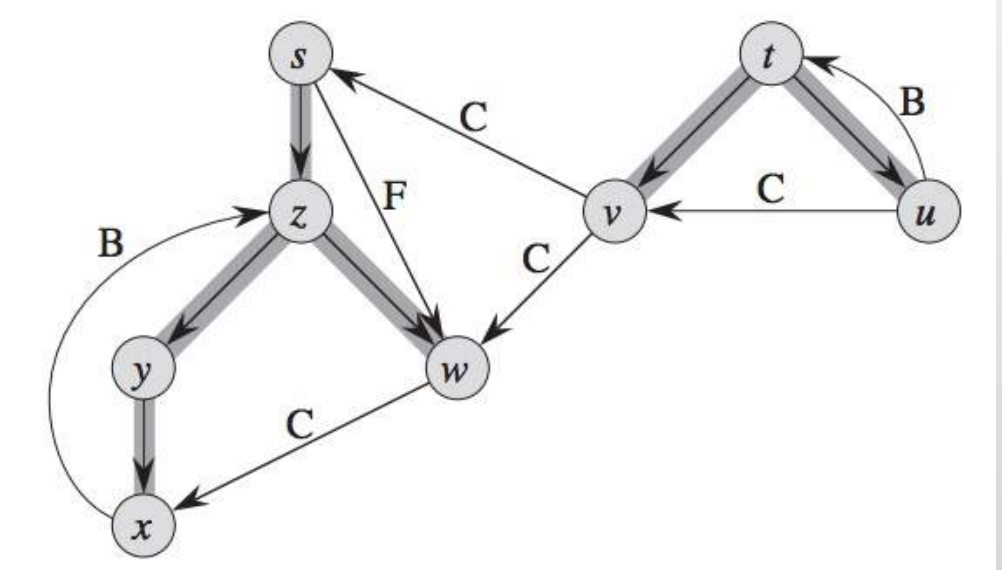
Teorema parantezelor

- In orice traversare DFS a unui graf G=(V,E), pentru oricare 2 varfuri – *u* si *v*, exact una din urmatoarele 3 conditii are loc:
  - Intervalele ***[u.d, u.f]*** si ***[v.d, v.f]*** sunt complet ***disjuncte*** (nici u, nici v nu este descendent al celuilalt nod in padurea de arbori de parcurgere in adancime)
  - ***[u.d, u.f]*** e inclus in ***[v.d, v.f]*** (*u* este descendent al lui *v* intr-un arbore DFS)
  - ***[v.d, v.f]*** e inclus in ***[u.d, u.f]*** (*v* este descendent al lui *u*)



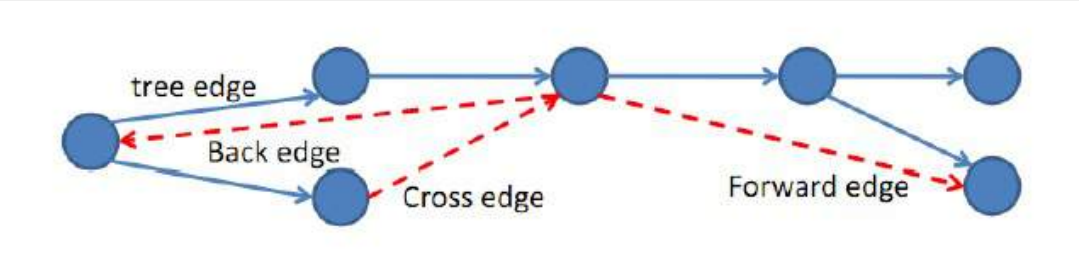
Proprietati ale parcurgerii in adancime

- Clasificarea muchiilor:
- Tipuri de muchii:
  - **Muchii arbore** (marcate cu linie gri ingrosata)**:** determinate la parcurgerea in adancime prin explorarea varfurilor muchiei (u,v).
  - **Muchii inapoi (B)**: (*v*,*u*), unde *v* este descendentul lui *u* in arborele de parcurgere in adancime.
  - **Muchii inainte (F)**: (*u*,*v*), unde *v* este descendentul lui *u* in arborele de parcurgere in adancime, dar nu este o muchie arbore
  - **Muchii transversale(C)**: restul muchiilor (intre ramuri, sau intre arbori diferiti).
- Algoritmul DFS poate fi modificat pentru a clasifica muchiile pe masura ce le intalneste.



https://www2.hawaii.edu/~janst/311/Notes/Topic-14.html

DFS clasificarea muchiilor



1. If v is visited for the first time as we traverse the edge (u, v), then the edge is a **tree edge**.
2. Else, v has already been visited:
   (a) If v is an ancestor of u, then edge (u, v) is a **back edge**.
   (b) Else, if v is a descendant of u, then edge (u, v) is a **forward edge**.
   (c) Else, if v is neither an ancestor or descendant of u, then edge (u, v) is a **cross edge**.

https://courses.csail.mit.edu/6.006/spring11/rec/rec13.pdf

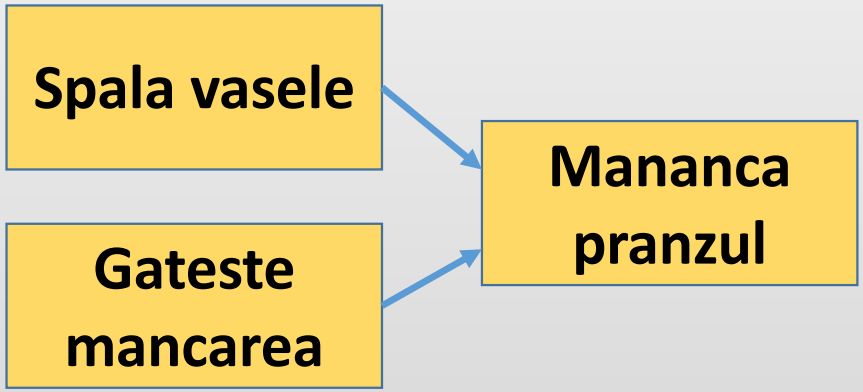
Aplicatii ale parcurgerii in adancime

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Determinarea ciclurilor intr-un graf
- Determinarea componentelor conexe/ puternic conexe ale unui graf
- Sortare topologica
- Gasirea puntilor (*bridges*) intr-un graf
  - Punte = muchie a carei stergere creste numarul de componente conexe ale unui graf nedirectionat
- Testarea planaritatii
  - Poate fi desenat in plan fara a exista intersectii de muchii
- Etc.

Sortare Topologica – exemplu

Presupunem ca e nevoie sa facem anumite sarcini care depind unele de altele.
Relatia dintre ele este modelata sub forma unui graf orientat ca in figura de mai jos.



Care este ordinea in care se vor executa ?
Eficienta?

Sortare Topologica

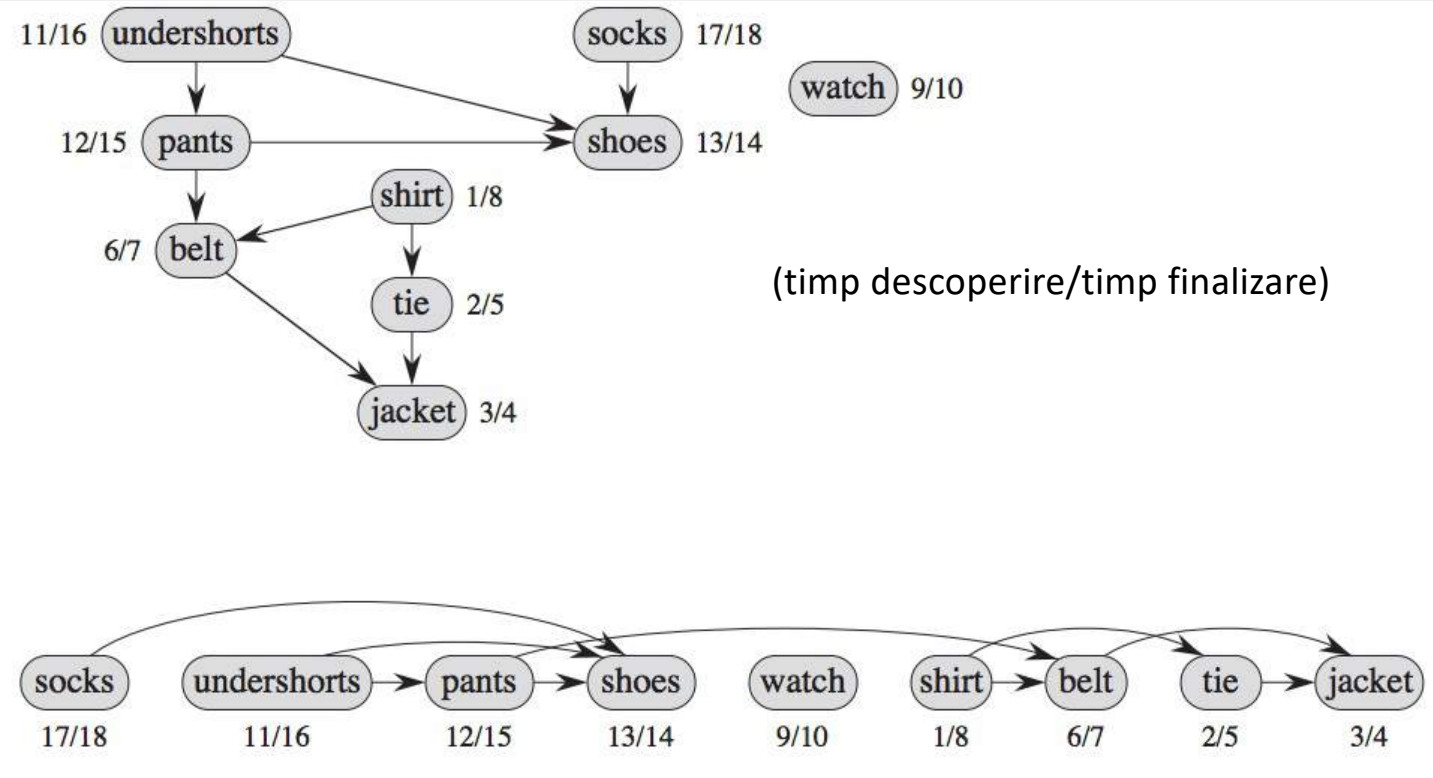
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- ***Sortare topologica*** a unui DAG – G=(V,E): ordonare liniara a tuturor varfurilor, astfel incat daca G contine muchia *(u,v)* => *u* apare inainte lui *v* in ordonare
  - **Daca graful contine cicluri, nu are sortare topo**!
  - Precedenta intre evenimente

```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times v.f for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

Altfel spus: determinam initial timpii de finalizare ai varfurilor si apoi afisam varfurile in ordinea inversa a timpilor de finalizare

Sortare Topologica – exemplu



(timp descoperire/timp finalizare)

Complexitate?

Determinarea componentelor puternic conexe – graf orientat

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Componenta puternic conexa: multime maximala de varfuri astfel incat oricare 2 varfuri din multime sunt conectate.

Algoritmul lui Kosaraju

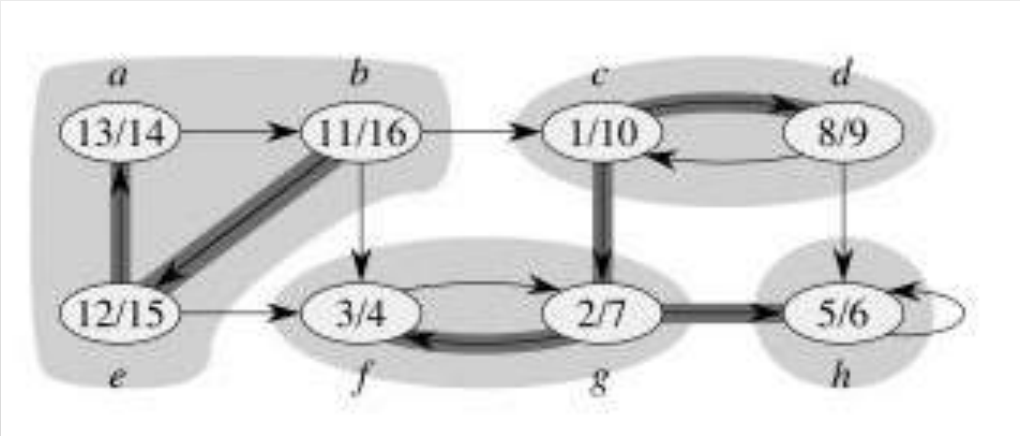
```
STRONGLY-CONNECTED-COMPONENTS(G)
1  call DFS(G) to compute finishing times u.f for each vertex u
2  compute G^T
3  call DFS(G^T), but in the main loop of DFS, consider the vertices
       in order of decreasing u.f (as computed in line 1)
4  output the vertices of each tree in the depth-first forest formed in line 3 as a
       separate strongly connected component
```

G^T este graful transpus si se obtine prin inversarea directiilor muchiilor.

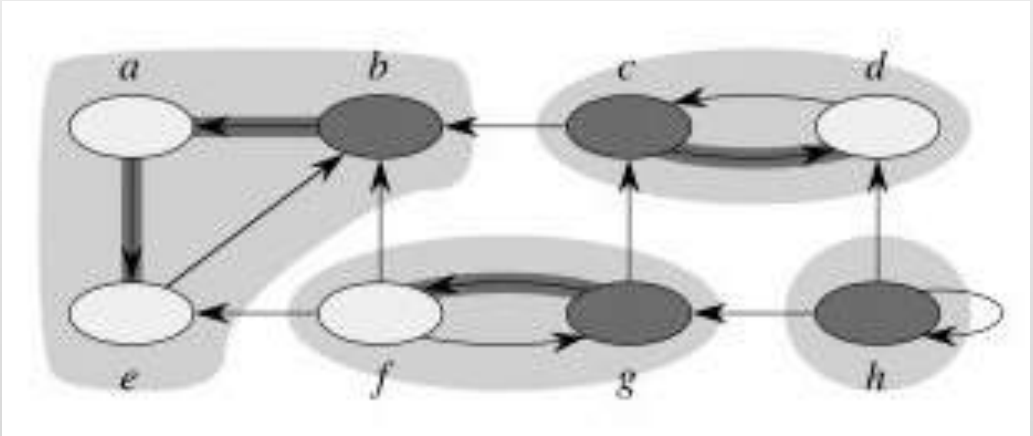
Determinarea componentelor puternic conexe

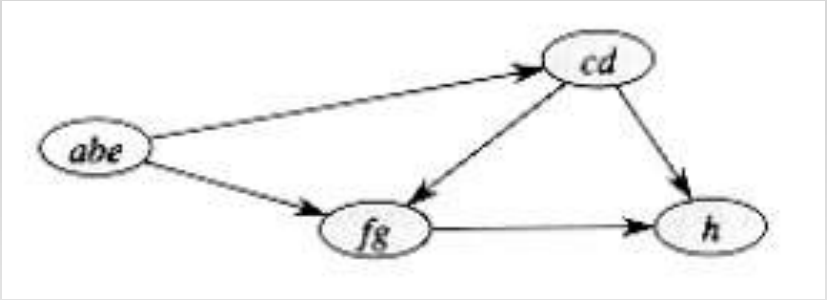
UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

DFS parcurgere 1 pe G:



DFS parcurgere 2 pe G^T:





Componentele puternic conexe formeaza si ele un graf orientat.
Acest graf nu are cicluri.

Bibliografie

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- CLR, “Introduction to Algorithms”, chapter 22 - Elementary Graph Algorithms

- Sursa figuri: http://algoviz.org/OpenDSA/Books/Everything/html/GraphIntro.html

- https://courses.cs.washington.edu/courses/cse373/17wi/#lectures
- http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/bfs.html
- https://www2.hawaii.edu/~janst/311/Notes/Topic-14.html
- https://courses.csail.mit.edu/6.006/spring11/rec/rec13.pdf
- Trasare exemplu sortare topologica: https://www.hackerearth.com/practice/algorithms/graphs/topological-sort/tutorial/