Proiectarea Algoritmilor

Curs 5 – Introducere în grafuri

Bibliografie

- Giumale – Introducere în Analiza Algoritmilor cap 5 și 5.1
- Cormen – Introducere în Algoritmi cap Algoritmi elementari de grafuri (23) – Reprezentări + Căutări
- http://www.h3c.com/portal/res/200706/01/20070601_108959_image001_201240_57_0.gif
- http://ashitani.jp/gv/
- http://en.wikipedia.org/wiki/PageRank

Plan curs

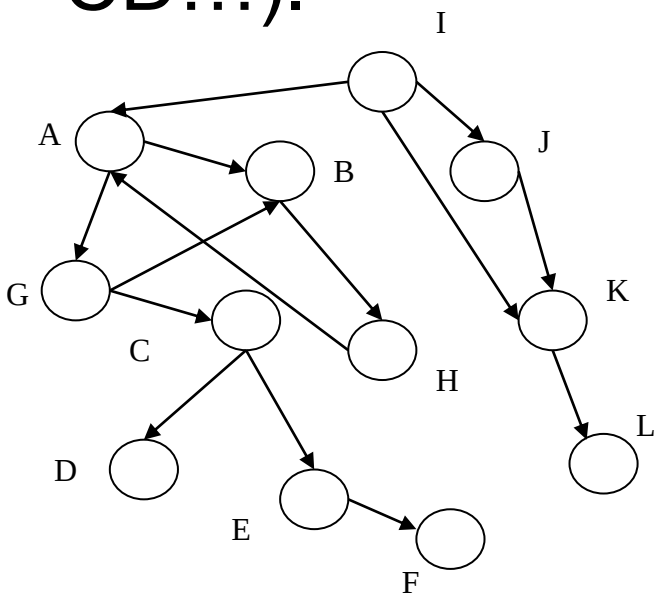
- Introducere
- Modalități de reprezentare
- Exemple de probleme practice
- Algoritmi de parcurgere
  - BFS
  - DFS
- Sortare topologică

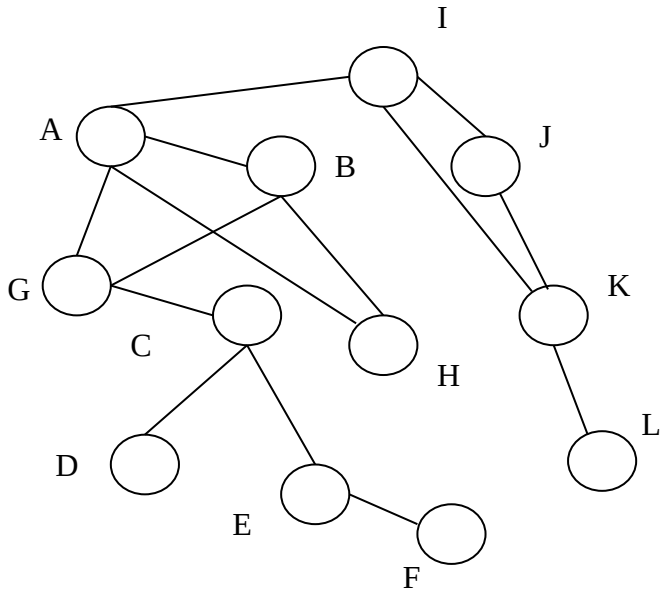
Introducere

- Circa 6 cursuri în care sunt prezentați algoritmii cei mai importanți pentru prelucrarea grafurilor:
  - Parcurgere
  - Sortare topologică
  - Componente tare conexe
  - Puncte de articulație
  - Punți
  - Arbori minimi de acoperire
  - Drumuri de cost minim
  - Fluxuri maxime
- Încercăm să legăm algoritmii de aplicații cât mai practice.

Tipuri de grafuri

- Orientate: noduri (A-L) + arce (AB, BC, CD…).

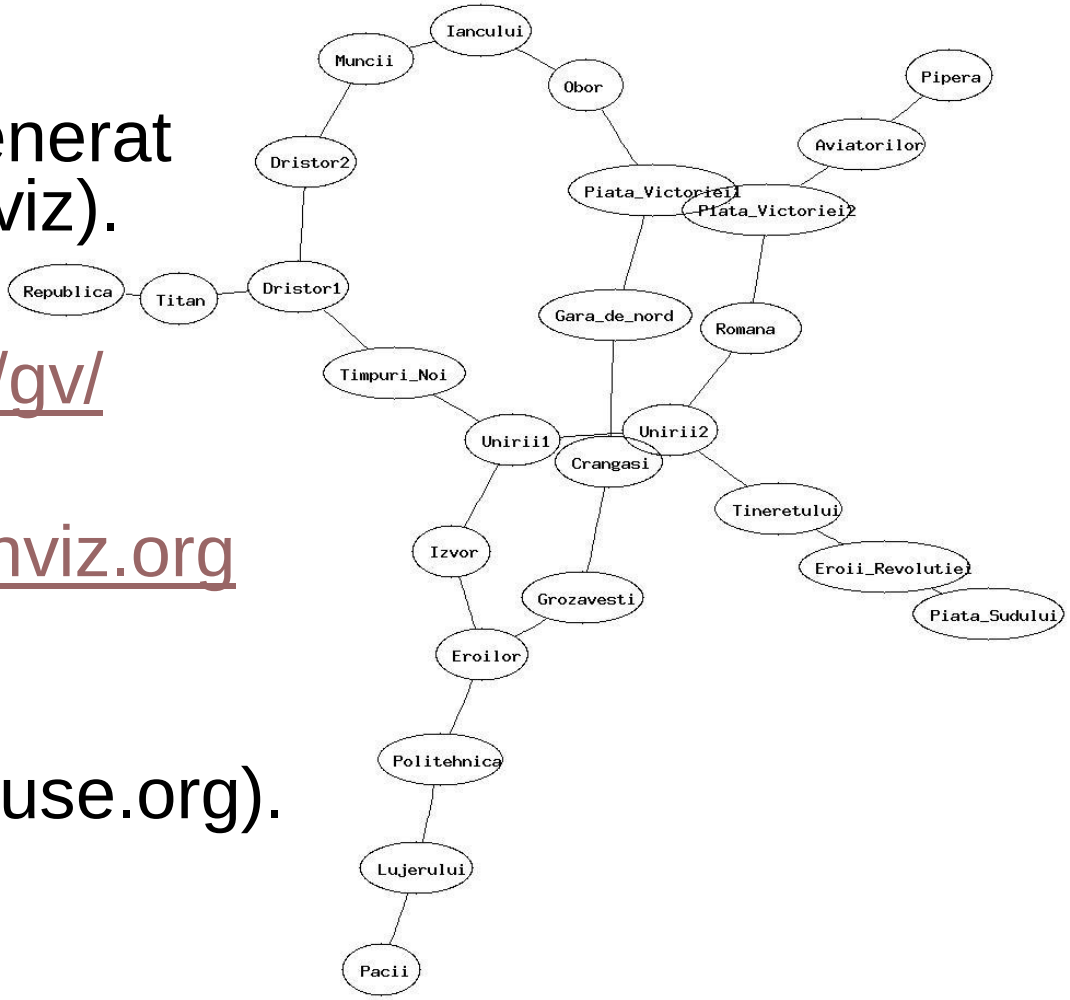




- Neorientate: noduri (A-L) + muchii (AB, BC, CD…).

Exemplu graf neorientat

- Exemplu graf generat cu neato (graphviz).
- http://ashitani.jp/gv/
- http://www.graphviz.org
- Biblioteci pentru vizualizare (Prefuse.org).



Modalități de descriere ale grafurilor

- Reprezentare în memorie:
  - Liste de adiacență;
  - Matrice de adiacență.
- Reprezentarea datelor de intrare:
  - Tupluri (sursă, destinație);
    - Întâlnite mai ales în descrierile folosind baze de date.
  - Limbaje specializate (ex: dot, GraphML, rdf).

Formate de reprezentare

- Listă adiacență :
  - Eroilor: Politehnica, Grozăvești, Izvor
  - Muncii: Dristor2, Ianculului…
- Matrice adiacență:

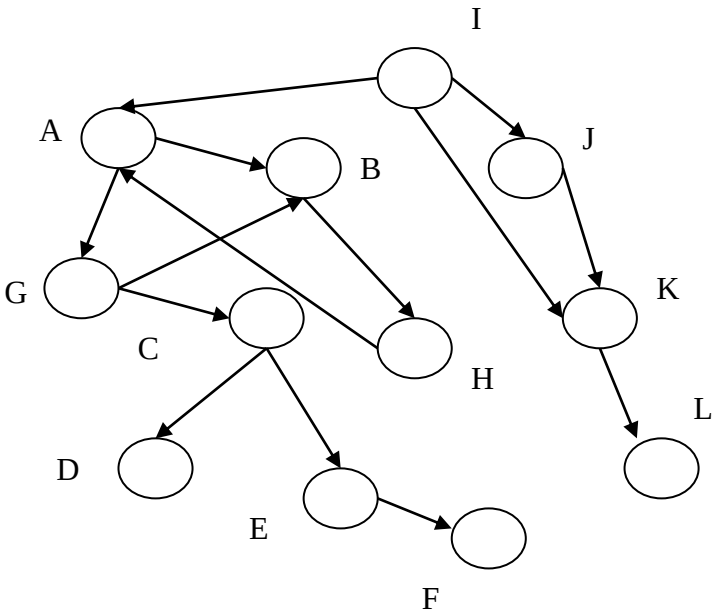
	Unirii2	Tineretului	Romană	…
Unirii2	-	1	1	
Tineretului	1	-		
Romană	1		-	
…				

- Tupluri:
  - (Dristor1; Dristor2)
  - (Eroilor; Grozăvești)
  - …
- Dot:
  - graph G {node;
  - Dristor2--Muncii--Ianculuì—Obor;
  - Piata_Victoriei1--Gara_de_nord--Crangasi--Grozavesti--Eroilor;
  - Pacii--Lujerului--Politehnica--Eroilor;
  - Republica--Titan--Dristor1--Timpuri_Noi--Unirii1--Izvor--Eroilor;
  - Dristor1--Dristor2;
  - Unirii1--Unirii2;
  - Piata_Victoriei1--Piata_Victoriei2;
  - Piata_Sudului--Eroii_Revolutiei--Tineretului--Unirii2--Romana--Piata_Victoriei2--Aviatorilor--Pipera;
  - }

Formate de reprezentare - GraphML

- GraphML
  - <graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  - <graph edgedefault="undirected">
  - <!-- data schema -->
  - <key id="name" for="node" attr.name="name" attr.type="string"/>
  - <key id="gender" for="node" attr.name="gender" attr.type="string"/>
  - <!-- nodes -->
  - <node id="1">
  - <data key="name">Jeff</data>
  - <data key="gender">M</data>
  - </node>
  - <node id="2">
  - <data key="name">Ed</data>
  - <data key="gender">M</data>
  - </node>
  - <edge source="1" target="2"></edge>
  - </graph>
  - </graphml>

Matrice de adiacență

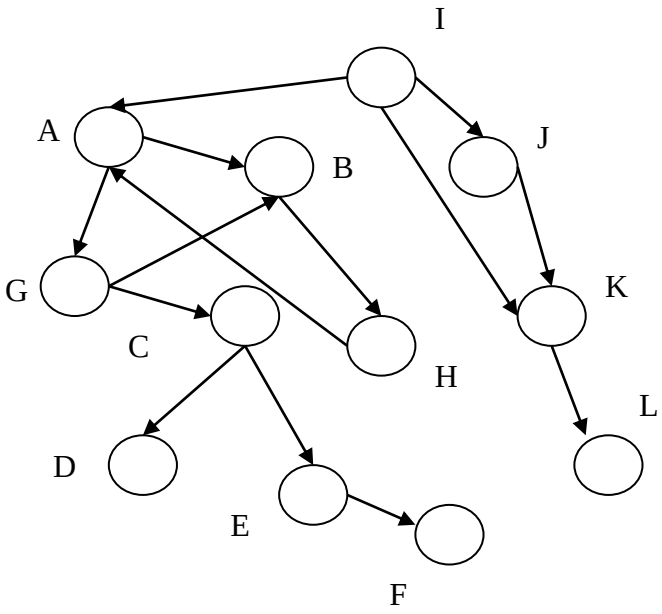


Cum se determină matricea de adiacență?

Matrice de adiacență

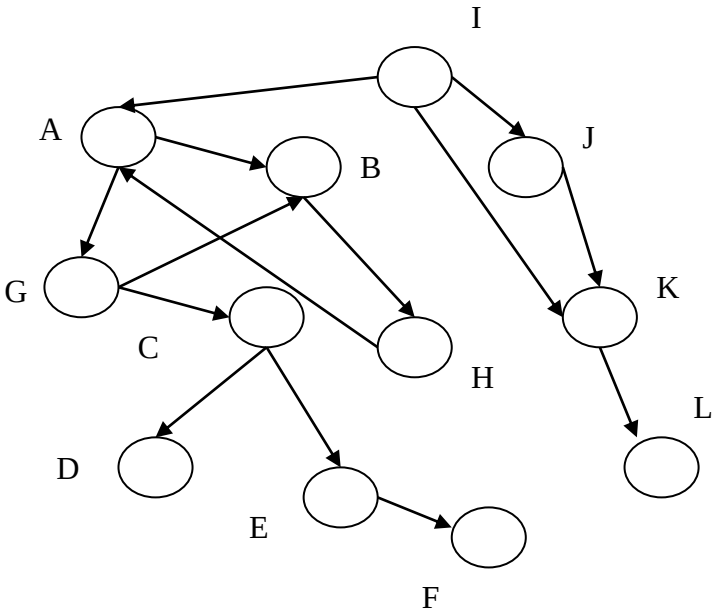
Matricea este rară?

- G – rar
- G – dens



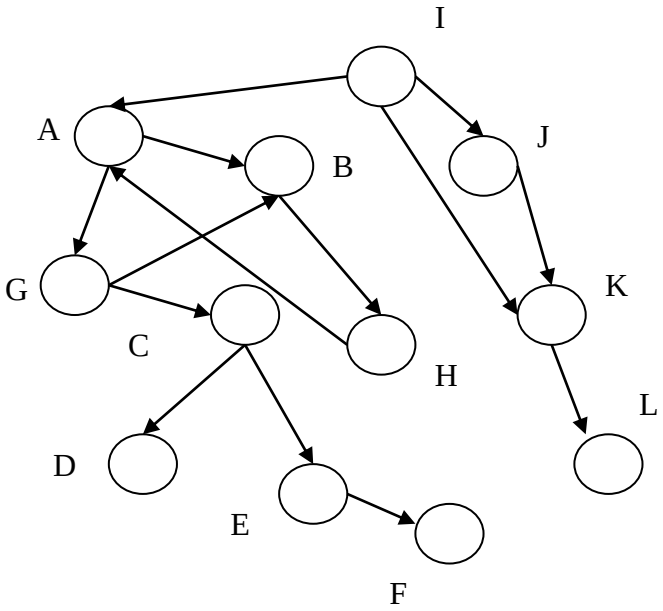
	A	B	C	D	E	F	G	H	I	J	K	L
A		1					1					
B								1				
C				1	1							
D												
E						1						
F												
G		1	1									
H	1											
I	1									1	1	
J											1	
K												1
L												

Vector de adiacență



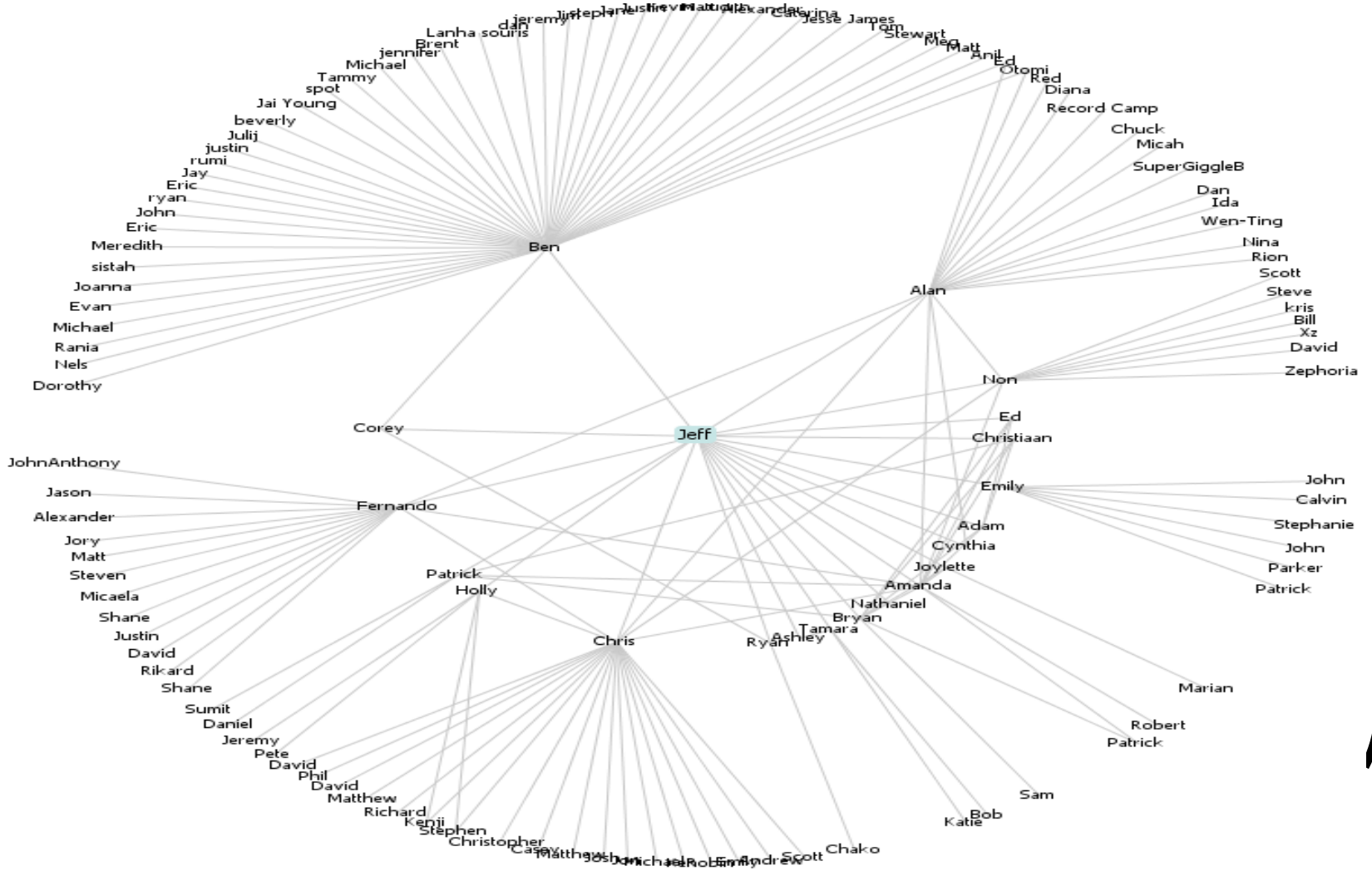
Cum se determină vectorul de adiacență?

Vector de adiacență

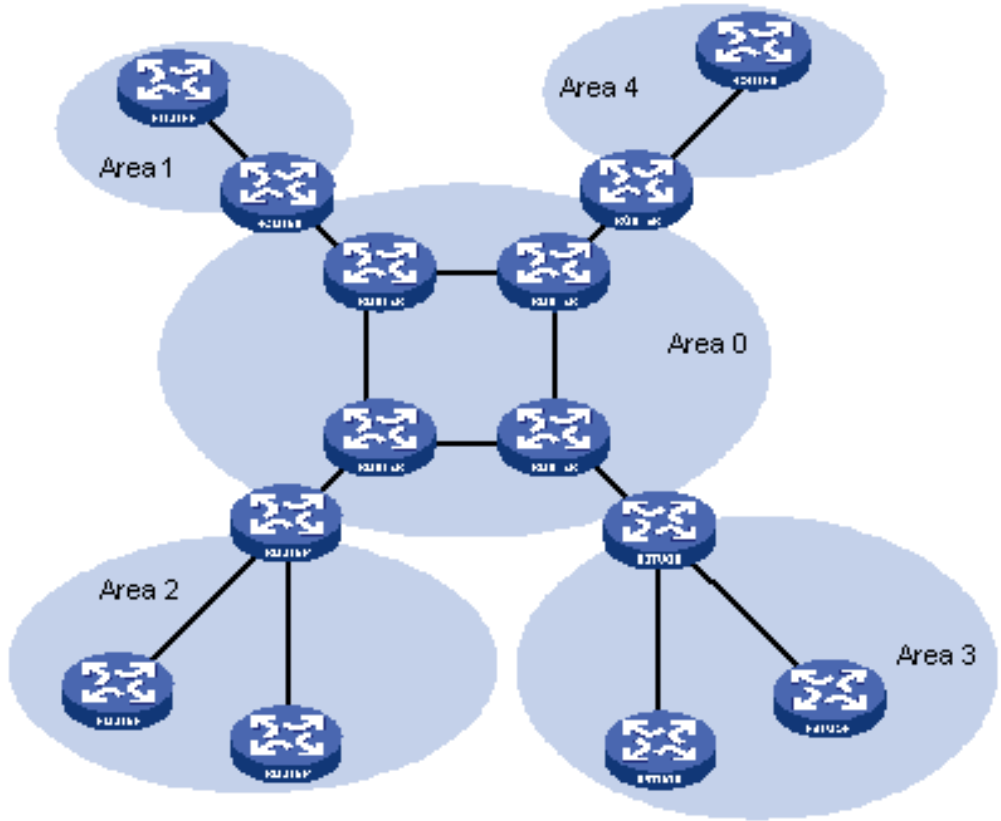


A	**B**	**G**	
B	**H**		
C	**D**	**E**	
D			
E	**F**		
F			
G	**B**	**C**	
H	**A**		
I	**A**	**J**	**K**
J	**K**		
K	**L**		
L			

Utilizări practice - Rețele sociale

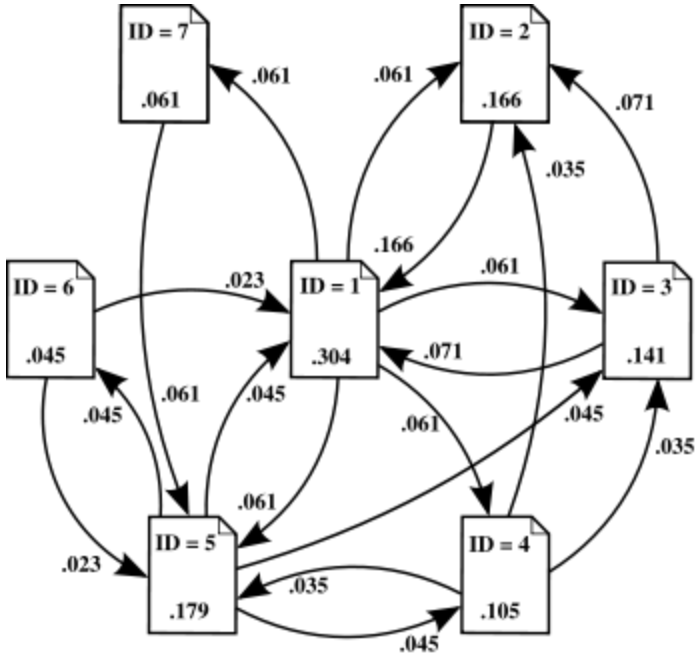


Utilizări practice – Rețele de calculatoare

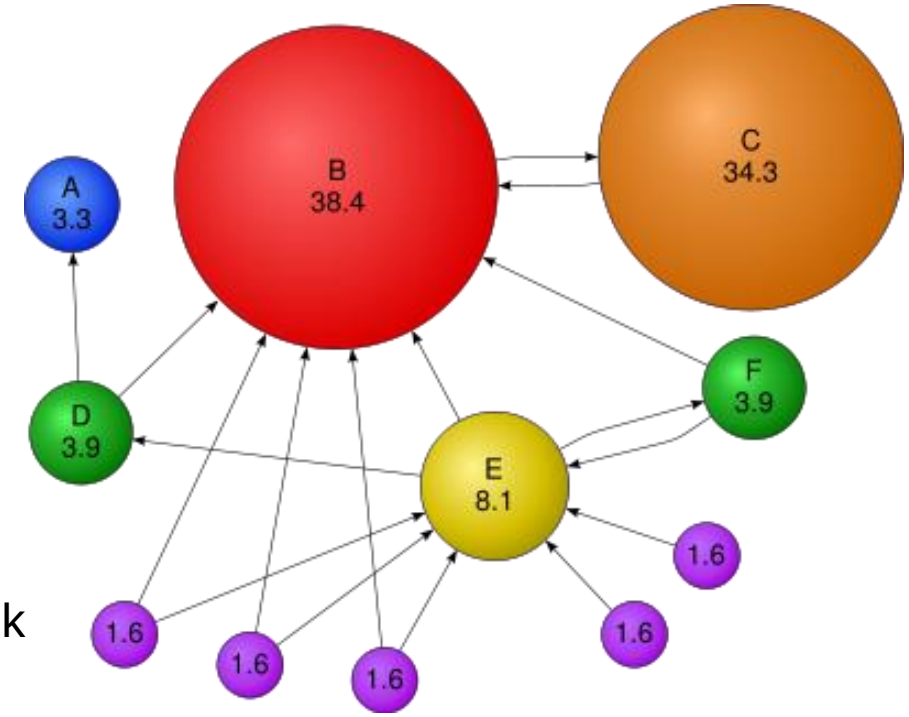


http://www.h3c.com/portal/res/200706/01/20070601_108959_image001_201240_57_0.gif

Utilizări practice - Web



http://en.wikipedia.org/wiki/PageRank



Utilizări practice

- Hărți, rețele (calculatoare, instalații, etc.), rețele sociale, analiza fluxurilor (semaforizare, proiectarea dimensiunii țevilor de apă).

- Exemple simple:
  - Cel mai scurt drum între punctele A și B pe o hartă.
  - Radialitate – în rețele sociale: gradul în care rețeaua socială a unui individ se întinde în rețeaua globală pentru a schimba date și influență.
  - Page Rank (Google). http://www.sirgroane.net/google-page-rank/

Algoritmi de parcurgere – Notații (1)

- G = (V,E);
- V – mulțimea de noduri;
- E – mulțimea de muchii / arce;
- (u,v) – arcul / muchia u,v;
- u..v – drum de la u la v; dacă există mai multe variante notăm u..x..v, u..y..v;
- R(u) - reachable(u) = mulțimea nodurilor ce pot fi atinse pe căi ce pleacă din u;

Algoritmi de parcurgere – Notații (2)

- succs(u) – mulțimea succesorilor lui u (graf orientat) sau mulțimea nodurilor adiacente lui u (graf neorientat);
- c(u) – culoarea nodului – specifică starea nodului la un anumit moment al parcurgerii:
  - Alb – nedescoperit;
  - Gri – descoperit, în curs de prelucrare;
  - Negru – descoperit și terminat (cu semnificații diferite pentru BFS si DFS).
- p(u) (π(u)) – "părintele lui u" – identificator al nodului din care s-a ajuns în nodul u prima oară.

Parcurgere în lățime (BFS)

- Nod de start (sursă): s.
- Determină numărul minim de arce / muchii între s și $\forall$ u $\in$ V = numărul de pași între sursă și orice alt nod din graf (acesta este cel mai scurt drum în condițiile în care nu există o funcție de cost asociată grafului).
- δ(s,u) – costul optim al s..u; $\delta(s,u) = \infty <=> u \notin R(s)$.
- Dist(s,u) – costul drumului descoperit s..u.
- Ex: Politehnica → restul stațiilor de autobuz (de câte bilete am nevoie?)

BFS – Structura de date

- Folosește o coadă (FIFO) pentru a reține nodurile ce trebuie prelucrate.

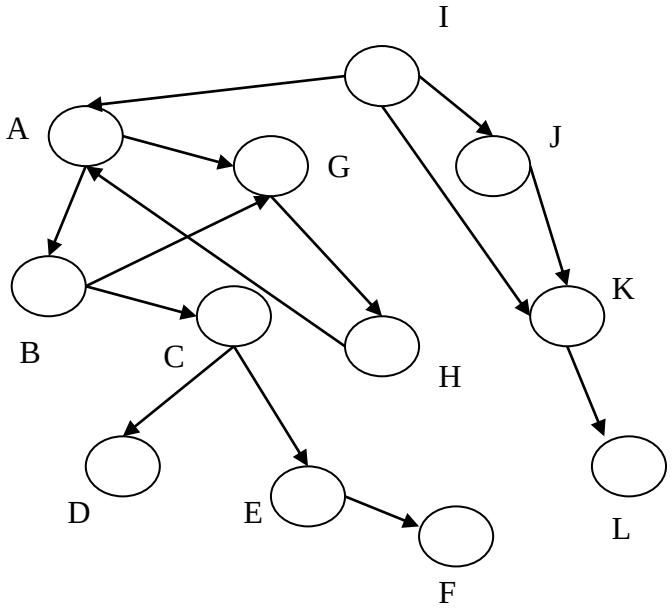
- Pentru fiecare nod se reține:
  - Părintele – $\pi(u)$ ($p(u)$);
  - Dist(s,u) – distanța până la nodul sursă;
  - Culoarea nodului.

BFS – Algoritm

- BFS(s,G)
  - **Pentru fiecare** nod u (u ∈ V)
    - p(u) = null; dist(s,u) = inf; c(u) = alb; // inițializări
  - Q = (); // se folosește o coadă în care reținem nodurile de prelucrat
  - dist(s,s) = 0; // actualizări: distanța de la sursă până la sursă este 0
  - Q ← Q + s; // adăugăm sursa în coadă → începem prelucrarea lui s
  - c(s) = gri; // și atunci culoarea lui devine gri
  - **Cât timp** (!empty(Q)) // cât timp mai am noduri de prelucrat
    - u = top(Q); // se determină nodul din vârful cozii
    - **Pentru fiecare** nod v ∈ succs(u) // pentru toți vecinii
      - **Dacă** c(v) este alb // nodul nu a mai fost găsit, nu e în coadă
        - **Atunci** { dist(s,v) = dist(s,u) + 1; p(v) = u; c(v) = gri; Q ← Q + v;} // actualizăm structura date
    - c(u) = negru; // am terminat de prelucrat nodul curent
    - Q ← Q - u; // nodul este eliminat din coadă

BFS – Exemplu

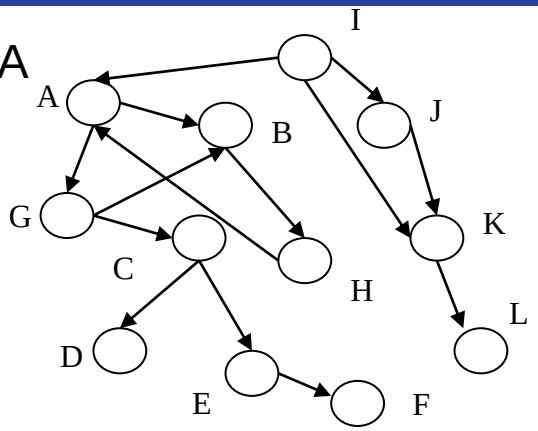
Sursa = A



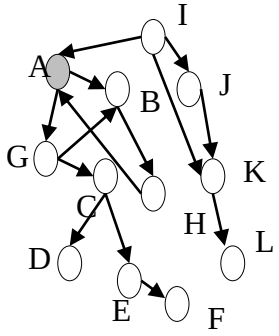
- BFS(s,G)
  - **Pentru fiecare** nod u (u ∈ V)
    - p(u) = null; dist(s,u) = inf; c(u) = alb; // inițializări
  - Q = (); // se folosește o coadă pentru nodurile de prelucrat
  - dist(s,s) = 0; // actualizări: distanța de la s la s e 0
  - Q ← Q + s; // adăugăm sursa în coadă → începem cu s
  - c(s) = gri; // și atunci culoarea lui devine gri
  - **Cât timp** (!empty(Q)) // cât timp am noduri de prelucrat
    - u = top(Q); // se determină nodul din vârful cozii
    - **Pentru fiecare** nod v ∈ succs(u) // pentru toți vecinii
      - **Dacă** c(v) este alb // nodul nu a mai fost găsit, nu e în Q
        Atunci { dist(s,v) = dist(s,u) + 1; p(v) = u; c(v) = gri;
        Q ← Q + v;} // actualizăm structura date
    - c(u) = negru; // am terminat de prelucrat nodul curent
    - Q ← Q - u; // nodul este eliminat din coadă

BFS – Evoluția explorării

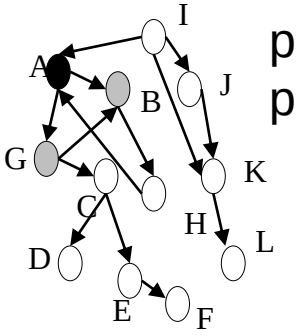
Sursa = A



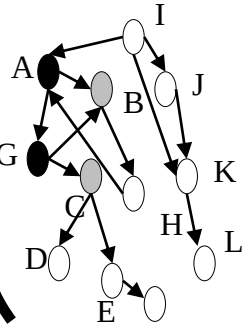
Q = A; d(A) = 0
p(A) = null



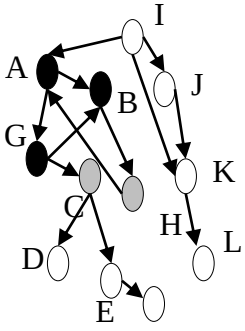
Q = G, B
d(B) = d(G) = 1
p(B) = A
p(G) = A



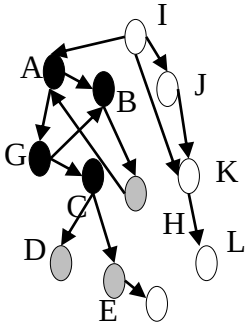
Q = B, C
d(C) = 2



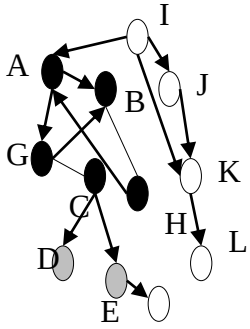
Q = C, H
d(H) = 2



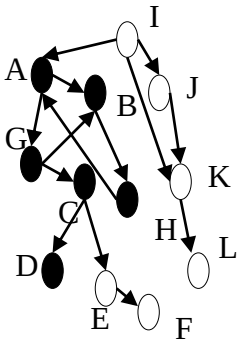
Q = H, D, E
d(D)=d(E)=3



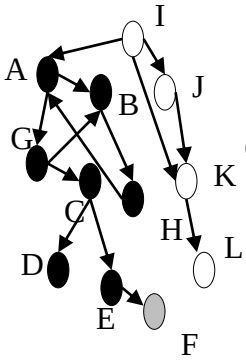
Q = D, E



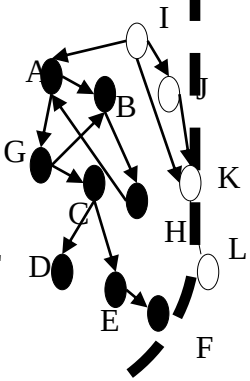
Q = E



Q = F
d(F)=4

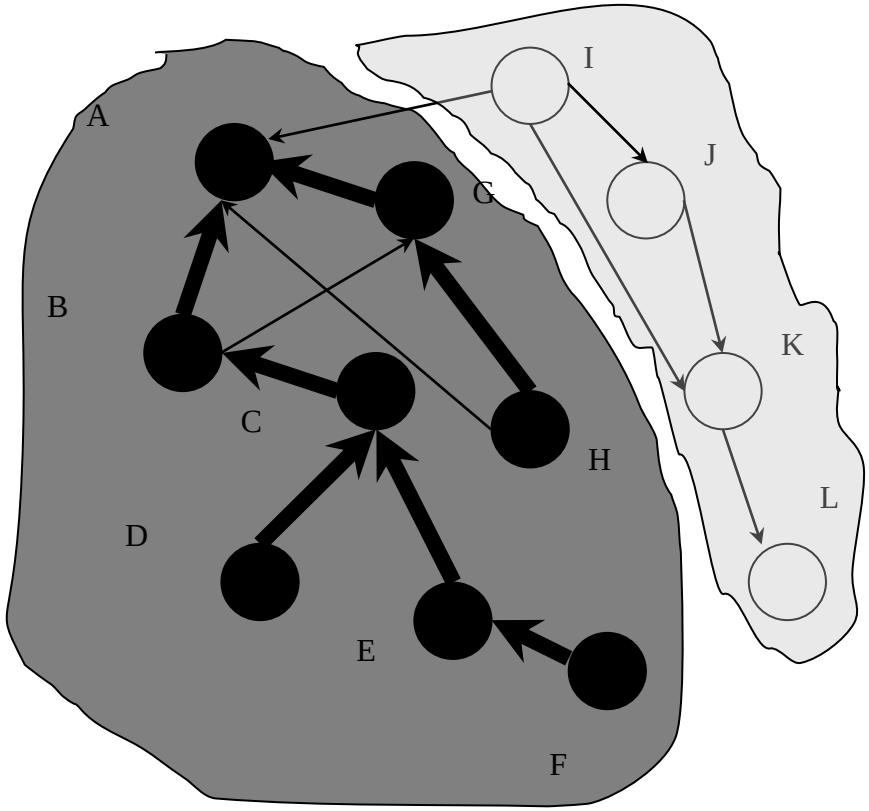


Q = Ø



p(C) = G p(H) = B p(D) = p(E) = C p(F) = E

BFS – Zona de explorare



BFS – Proprietăți (I)

- Lema 5.1. În cursul execuției BFS(s,G) $v \in Q \Leftrightarrow v \in R(s)$.
  - →Q conține exclusiv noduri din R(s) (BFS parcurge toate nodurile ce pot fi atinse din s);
  - ← toate nodurile ce pot fi atinse din s vor fi introduse cândva în coadă. (drum $s = v_0 v_1 \ldots v_p = v$)
  - Dem prin inducție!

- Lema 5.2. $\forall (u,v) \in E$, $\delta(s,v) \leq \delta(s,u) + 1$
  - $\delta(s,v) \leq \delta(s,u) + 1$ în general este egalitate când v este descoperit din u; < când v deja a fost descoperit înainte să se ajungă în u.
  - Dem prin reducere prin absurd!

BFS – Proprietăți (II)

- Lema 5.3. La terminarea BFS(s,G) există proprietatea $dist(s,u) \geq \delta(s,u)$.
  - Dem prin inducție folosind Lema 5.1 și Lema 5.2!

- Lema 5.4. După orice execuție a ciclului principal al BFS, Q conține $v_1, v_2, \ldots, v_p$ ai:
  - $Prop(Q) = dist(s,v_1) \leq dist(s,v_2) \leq \ldots \leq dist(s,v_p) \leq dist(s,v_1) + 1$
  - => la un moment dat in coadă sunt elemente de pe același nivel din arborele generat de BFS (sau maxim 1 nivel diferență).
  - Dem prin inducție după numărul de elemente din Q! (demonstrăm invarianța Prop(Q) la inserare și eliminare de elemente în/din Q.)

BFS – Proprietăți (III)

- Corolar
  - d(u) = momentul în care nodul u este inserat în coada Q. Atunci:

$$d(u) < d(v) \Rightarrow dist(s,u) \leq dist(s,v).$$

- Teorema 5.1. BFS este corect și după terminare $\delta(s,u) = dist(s,u)$, $\forall$ u din V.
  - Utilizăm notația $V_k = \{ u \in V \mid \delta(s,u) = k \}$
  - Dem prin inducție $P(k) = \{ u \in V_k \mid \delta(s,u) = dist(s,u) \;\&\&\; (k > 0 \Rightarrow \pi(u) \in V_{k-1}) \;\&\&\; (k = 0 \Rightarrow \pi(u) = null) \}$

Complexitate? Optimalitate? Completitudine?

BFS – Complexitate și Optimalitate

Complexitate:

O(n+m)

n = număr noduri

m = număr muchii

Optimalitate: DA

Parcurge tot graful? NU

Parcurgere în adâncime (DFS)

- Nu mai avem nod de start, nodurile fiind parcurse în ordine.

- d(u) = momentul descoperirii nodului (se trece prima oară prin u și e totodată și momentul începerii explorării zonei din graf ce poate fi atinsă din u).
- f(u) = timpul de finalizare al nodului (momentul în care prelucrarea nodului u a luat sfârșit)
  - Tot subarborele de adâncime dominat de u a fost explorat.
  - Alternativ: tot subgraful accesibil din u a fost descoperit și finalizat deja.

DFS – Structura de date

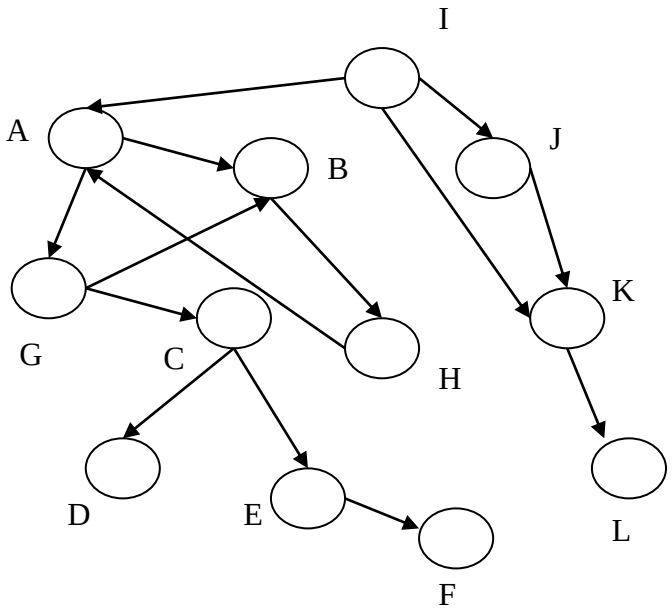
- Folosește o stiva (LIFO) pentru a reține nodurile ce trebuie prelucrate
  - În implementările uzuale, stiva este rareori folosită explicit;
  - Se apelează la recursivitate pentru a simula stiva.
- Folosește o variabilă globală *timp* pe baza căreia se calculează timpii de descoperire și de finalizare ai fiecărui nod.
- Pentru fiecare nod se reține:
  - Părintele – $\pi(u)$ ($p(u)$);
  - Timpul de descoperire – $d(u)$;
  - Timpul de finalizare – $f(u)$;
  - Culoarea nodului.

DFS – Algoritm

- DFS(G)
  - V = noduri(G)
  - **Pentru fiecare** nod u (u ∈ V)
    - c(u) = alb; p(u) = null; // inițializare structură date
  - timp = 0; // reține distanța de la rădăcina arborelui DFS pană la nodul curent
  - **Pentru fiecare** nod u (u ∈ V)
    - **Dacă** c(u) este alb
      - **Atunci** explorare(u); // explorez nodul

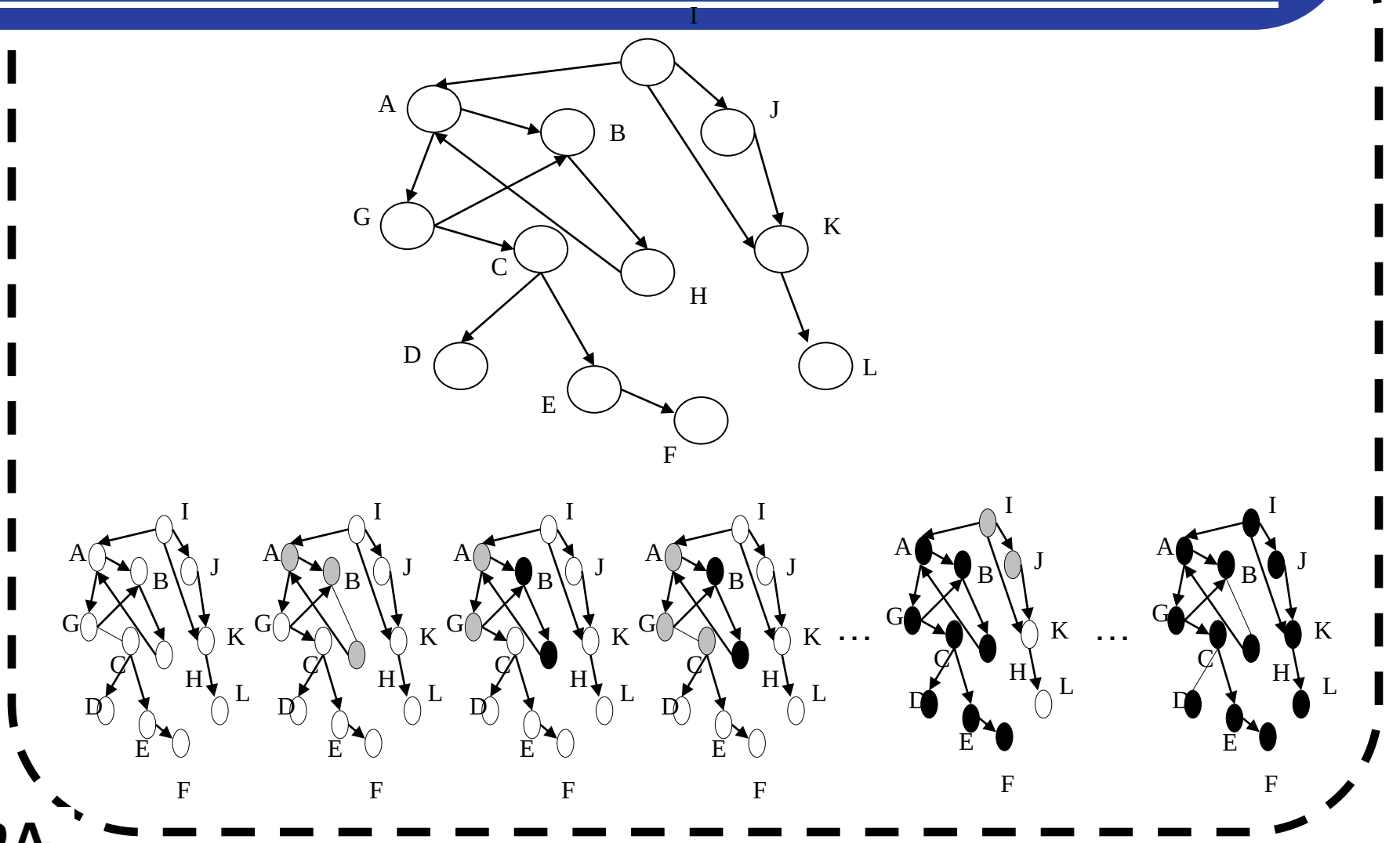
- explorare(u)
  - d(u) = ++ timp; // timpul de descoperire al nodului u
  - c(u) = gri; // nod în curs de explorare
  - **Pentru fiecare** nod v ∈ succs(u) // încerc sa prelucrez vecinii
    - **Dacă** c(v) este alb
      - **Atunci** {p(v) = u; explorare(v);} // dacă nu au fost prelucrați deja
  - c(u) = negru; // am terminat de explorat nodul u
  - f(u) = ++ timp; // timpul de finalizare al nodului u

DFS – Exemplu

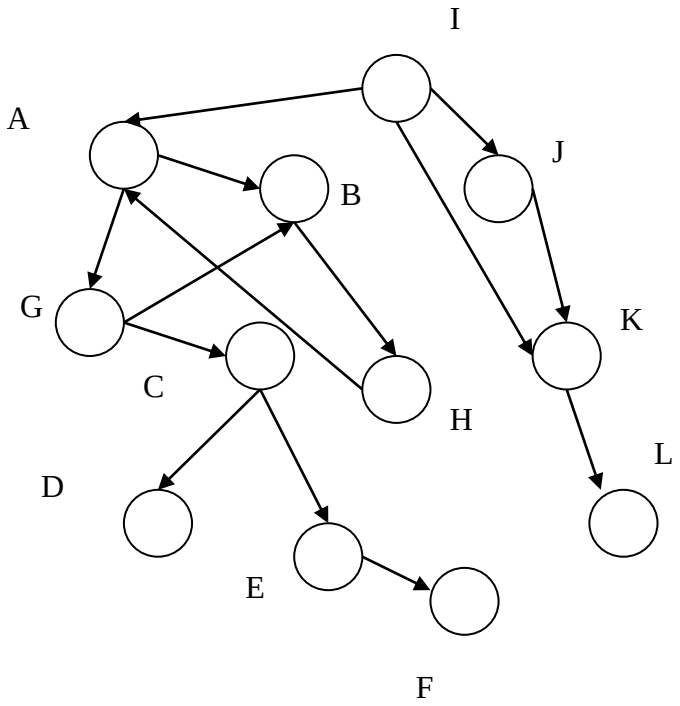


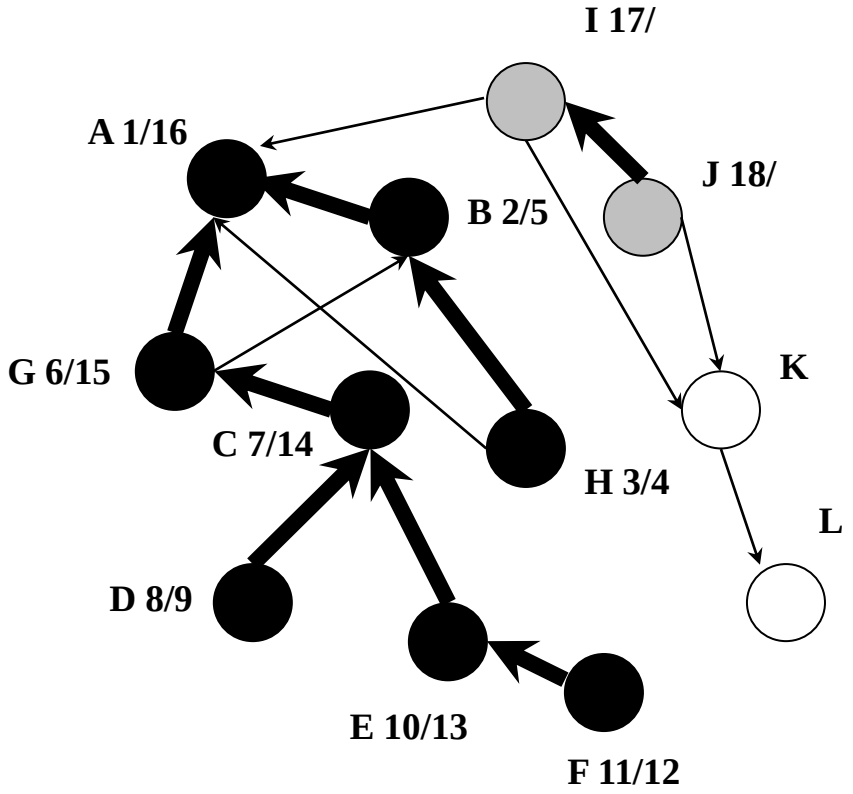
- DFS(G)
  - V = noduri(G)
  - **Pentru fiecare** nod u (u ∈ V)
    - c(u) = alb; p(u) = null; // inițializare structură date
  - timp = 0; // reține distanța de la rădăcina arborelui DFS pană la nodul curent
  - **Pentru fiecare** nod u (u ∈ V)
    - **Dacă** c(u) este alb
      - **Atunci** explorare(u); // explorez nodul
- explorare(u)
  - d(u) = ++ timp; // timpul de descoperire al nodului u
  - c(u) = gri; // nod in curs de explorare
  - **Pentru fiecare** nod v ∈ succs(u) // încerc sa prelucrez vecinii
    - **Dacă** c(v) este alb
      - **Atunci** {p(v) = u; explorare(v);} // dacă nu au fost prelucrați deja
  - c(u) = negru; // am terminat de explorat nodul u
  - f(u) = ++ timp; // timpul de finalizare al nodului u

DFS – Evoluția explorării

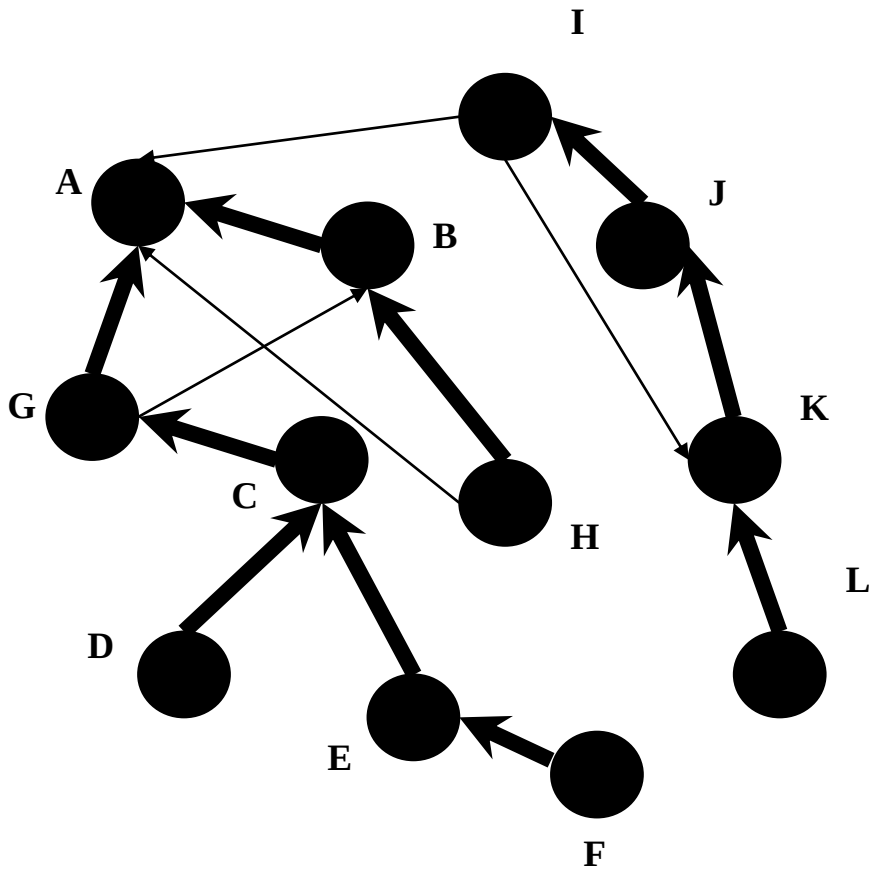


Calculul timpilor

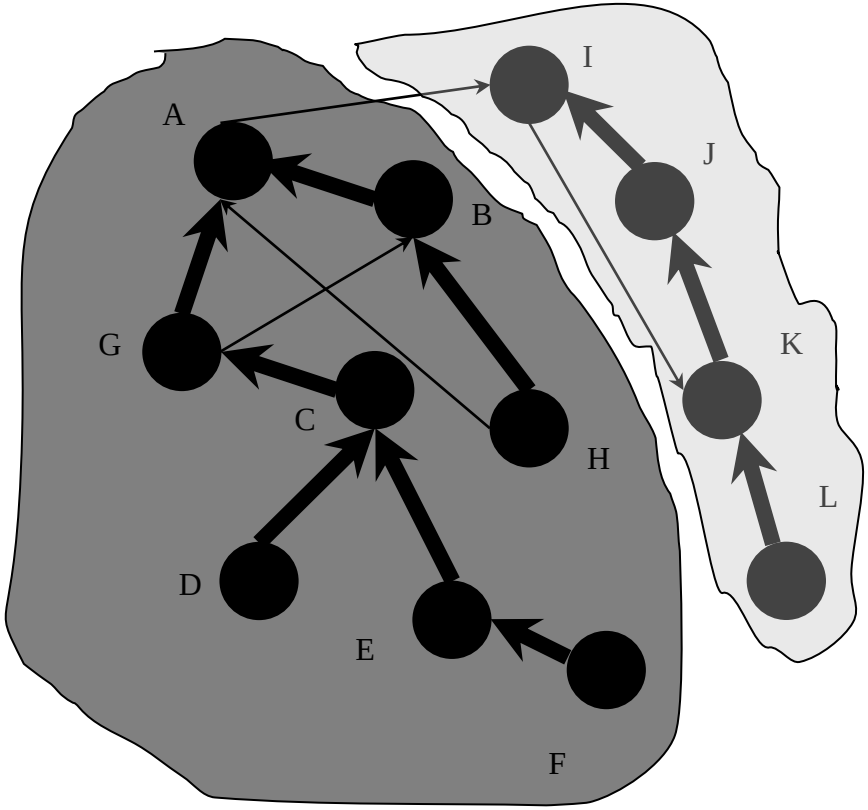




Arborele de parcurgere în adâncime



DFS – Zone de explorare



DFS – Proprietăți (I)

- I(u) = intervalul de prelucrare al nodului (d(u),f(u)).
- Lema 5.5. G = (V,E); u ∈ V; pentru fiecare v descoperit de DFS pornind din u este construită o cale v, p(v), p(p(v)),…, u.
  - Fie calea u = $v_0v_1…v_n$ = v. Dem prin inducție ca $\pi(v_i) = v_{i-1}$!
- Teorema 5.2. G = (V,E); DFS(G) sparge graful G într-o pădure de arbori Arb(G) = { Arb(u); p(u) = null } unde Arb(u) = (V(u),E(u));
  - V(u) = { v | d(u) < d(v) < f(u) } + {u};
  - E(u) = { (v, z) | v, z ∈ V(u) && p(z) = v }.

DFS – Proprietăți (II)

- Teorema 5.3. Dacă DFS(G) generează 1 singur arbore => G este conex. (Reciproca este adevărată?)
- Teorema 5.4. Teorema parantezelor:
  - $\forall$ u, v atunci I(u) $\cap$ I(v) $\neq \emptyset$ sau I(u) $\subset$ I(v) sau I(v) $\subset$ I(u).
  - Dem prin considerarea tuturor combinațiilor posibile!
- Teorema 5.5. $\forall$ u, v $\in$ V, atunci v $\in$ V(u) $\Leftrightarrow$ I(v) $\subset$ I(u).
- Teorema 5.6. Teorema drumurilor albe:
  - G = (V,E); Arb(u); v este descendent al lui u in Arb(u) $\Leftrightarrow$ la momentul d(u) există o cale numai cu noduri albe u..v.
  - Dem prin inducție!

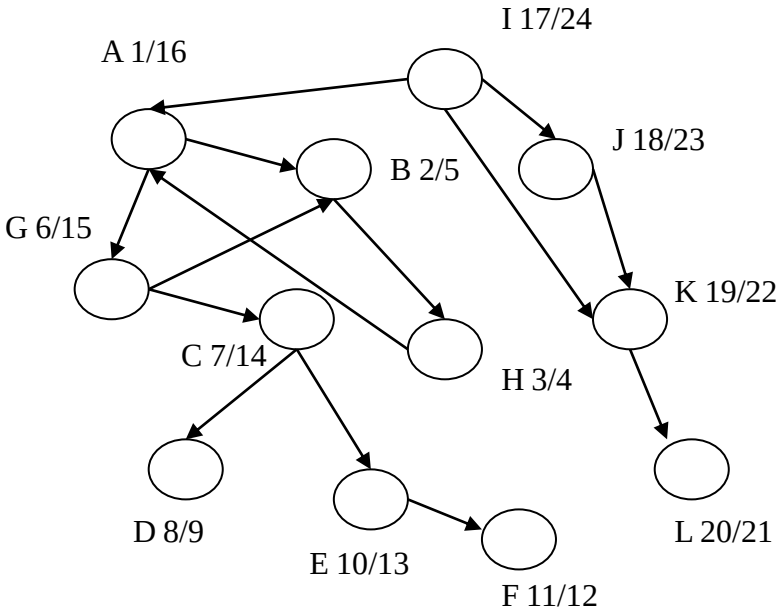
Clasificări ale arcelor grafului (I)

- Arc direct (de arbore)
  - Ce fel de noduri?
- Arc invers (de ciclu)
  - Ce fel de noduri?
- Arc înainte
  - Ce fel de noduri?
- Arc transversal
  - Ce fel de noduri?

Clasificări ale arcelor grafului (II)

- Arc direct (de arbore)
  - între nod gri și nod alb;
- Arc invers (de ciclu)
  - între nod gri și nod gri;
- Arc înainte
  - nod gri și nod negru și d(u) < d(v);
- Arc transversal
  - nod gri și nod negru și d(u) > d(v).

Clasificări ale arcelor grafului (III)



Arc direct (de arbore)
între nod gri și nod alb;
Arc invers (de ciclu)
între nod gri și nod gri;
Arc înainte
nod gri și nod negru și d(u) < d(v);
Arc transversal
nod gri și nod negru și d(u) > d(v).

Arc direct (de arbore):
AB, BH, AG, GC, CD, CE, EF, IJ, JK, KL
Arc invers (de ciclu):
HA
Arc înainte:
IK
Arc transversal:
GB, IA

DFS – Proprietăți (III)

- Teorema 5.7. Într-un graf neorientat, DFS poate descoperi doar arce directe și inverse.
  - Dem prin considerarea cazurilor posibile!

- Teorema 5.8. G = graf orientat; G ciclic ⇔ în timpul execuției DFS găsim arce inverse.
  - Dem prin exploatarea proprietăților de ciclu și de arc invers!

DFS – Complexitate și Optimalitate

Complexitate:

O(n+m)

n = număr noduri

m = număr muchii

Optimalitate: NU

Parcurge tot graful? DA

ÎNTREBĂRI?

Bibliografie curs 6

- Giumale – Introducere in Analiza Algoritmilor cap. 5.1 și 5.2
- Cormen – Introducere în Algoritmi cap. Algoritmi elementari de grafuri (23) – Sortare topologică + ComponenteTare Conexe
- http://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm