Proiectarea Algoritmilor

Curs 5 – Introducere in grafuri

Bibliografie

- Giumale – Introducere in Analiza Algoritmilor cap 5 si 5.1
- Cormen – Introducere în Algoritmi cap 22, 22.1, 22.2, 22.3 si 22.4
- http://ist.marshall.edu/ist362/pics/OSPF.gif
- http://ashitani.jp/gv/
- http://en.wikipedia.org/wiki/PageRank

Plan curs

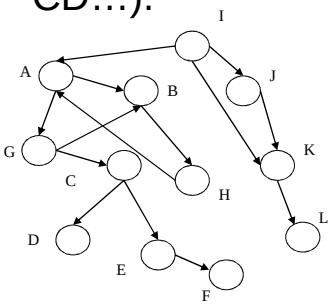
- Introducere
- Modalități de reprezentare
- Exemple de probleme practice
- Algoritmi de parcurgere
  - BFS
  - DFS
- Sortare topologică

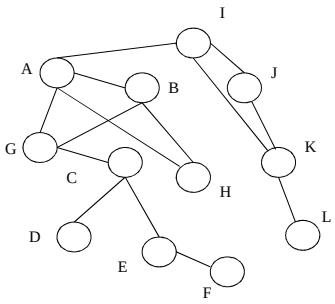
Introducere

- Circa 3 cursuri in 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 cat mai practice.

Tipuri de grafuri

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

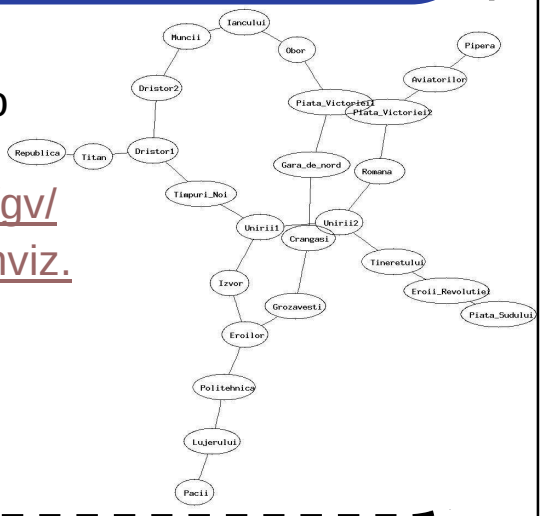




- Neorientate: noduri (A-I) + muchii (AB, BC, CD…).

Exemplu graf neorientat

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



Modalitati de descriere ale grafurilor

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

Formate de reprezentare

- Listă adiacență :
  - Dristor1: Titan, Timpuri_noi, Dristor2
  - Muncii: Dristor2, Obor…
- Matrice adiacență:

	Unirii2	Tineretului	Romana	…
Unirii2	-	1	1	
Tineretului	1	-		
Romana	1		-	
…				

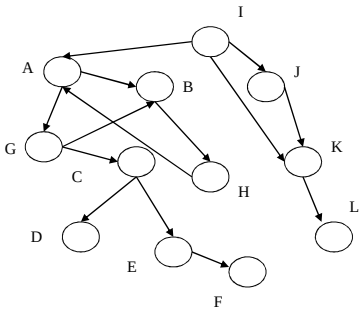
- Tupluri:
  - (Dristor1;Dristor2)
  - (Eroilor;Grozavesti)
  - …
- Dot:

```
graph G {node;
   Dristor2--Muncii--Iancului—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

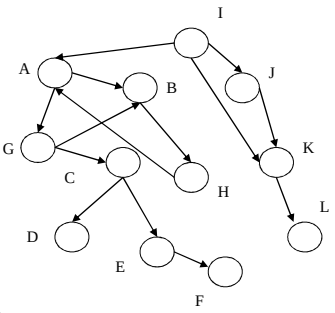


Cum se determină matricea de adiacență?

Matrice de adiacență

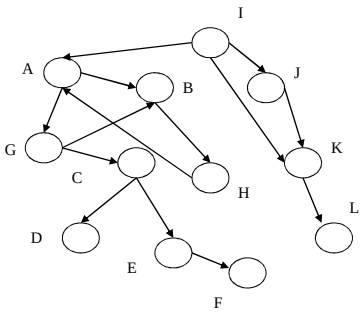
Matricea este rară?

- G – rar
- G – dens



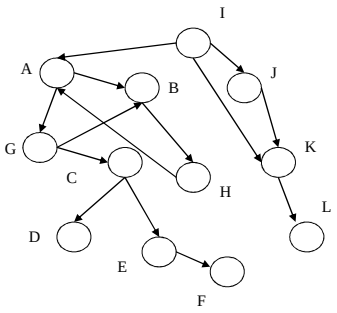
	A	B	C	D	E	F	G	H	I	J	K	L
A		1					1					
B								1				
C				1	1							
D												
E						1						
F												
G		1	1									
H	1											
I	1									1	1	
J											1	
K												1
L												

Vector de adiacență



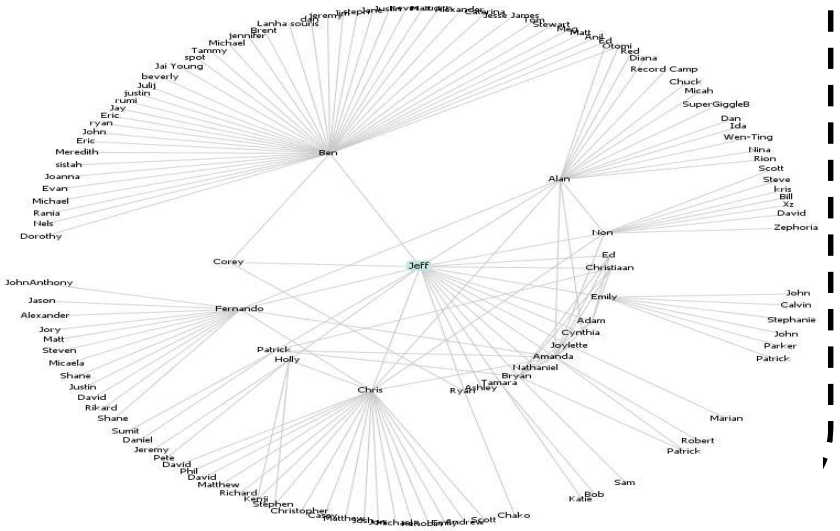
Cum se determină vectorul de adiacență?

Vector de adiacență

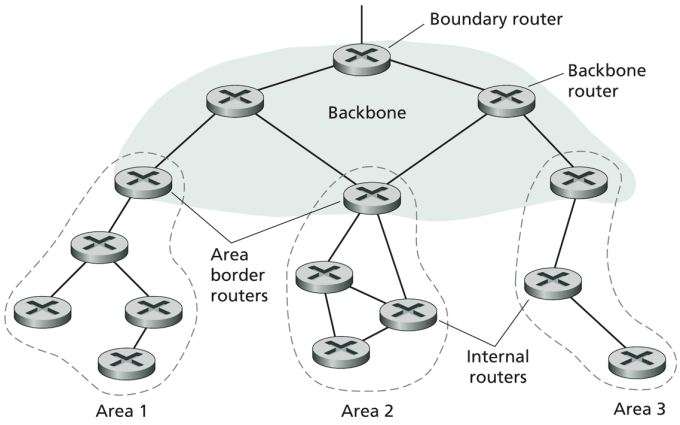


A	**B**	**G**	
B	**H**		
C	**D**	**E**	
D			
E	**F**		
F			
G	**B**	**C**	
H	**A**		
I	**A**	**J**	**K**
J	**K**		
K	**L**		
L			

Utilizări practice - Rețele sociale

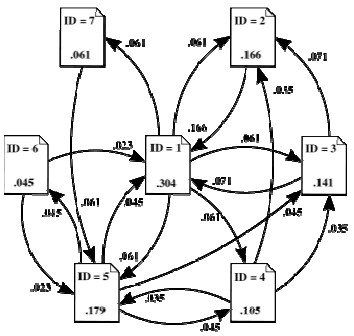


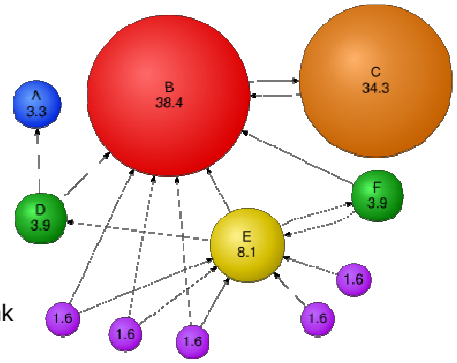
Utilizări practice – Rețele de calculatoare



http://ist.marshall.edu/ist362/pics/OSPF.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 intre punctele A si B pe o hartă.
  - Radialitate – in rețea socială: gradul in care rețeaua socială a unui individ se întinde in rețeaua globala pentru a schimba date si influență.
  - Page Rank (Google). http://www.iprcom.com/papers/pagerank/

Algoritmi de parcurgere – Notatii (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 – Notatii (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, in curs de prelucrare;
  - Negru – descoperit si terminat (cu semnificații diferite pentru BFS si DFS).
- p(u) – "părintele lui u" – identificator al nodului din care s-a ajuns in nodul u prima oară.

Parcurgere in lățime (BFS)

- Nod de start (sursă): s.
- Determină numărul minim de arce / muchii intre s si ∀ u ∈ V = numărul de pași intre sursă si orice alt nod din graf (acesta este cel mai scurt drum in condițiile in care nu există o funcție de cost asociată grafului).
- δ(s,u) – costul optim al s..u; δ(s,u) = ∞ => u ∉ 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

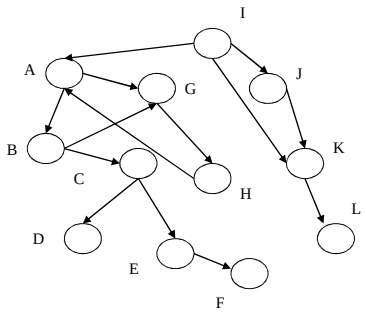
- Folosește o coadă (FIFO) pentru a reține nodurile ce trebuie prelucrate.
- Pentru fiecare nod se reține:
  - Părintele;
  - Dist(s,u) – distanța pană la nodul sursă;
  - culoarea nodului.

BFS – Algoritm

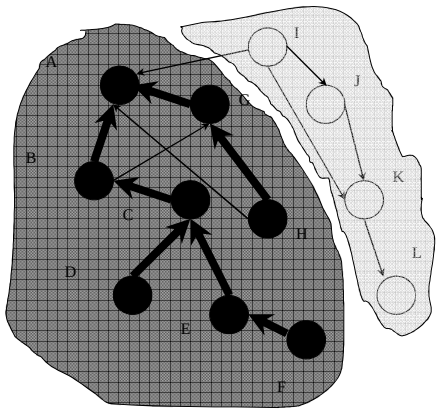
- BFS(s,G)
  - *Pentru fiecare* nod u (u ∈ V)
    - p(u) = null; dist(s,u) = inf; c(u) = alb; // inițializări
  - Q = (); // se folosește o coada in care reținem nodurile de prelucrat
  - dist(s) = 0; // actualizări: distanța de la sursă până la sursă este 0
  - Q ← Q + s; // adăugam sursa in coadă → începem prelucrarea lui s
  - c(s) = gri; // si 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 in coadă
        - *Atunci* { dist(v) = dist(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

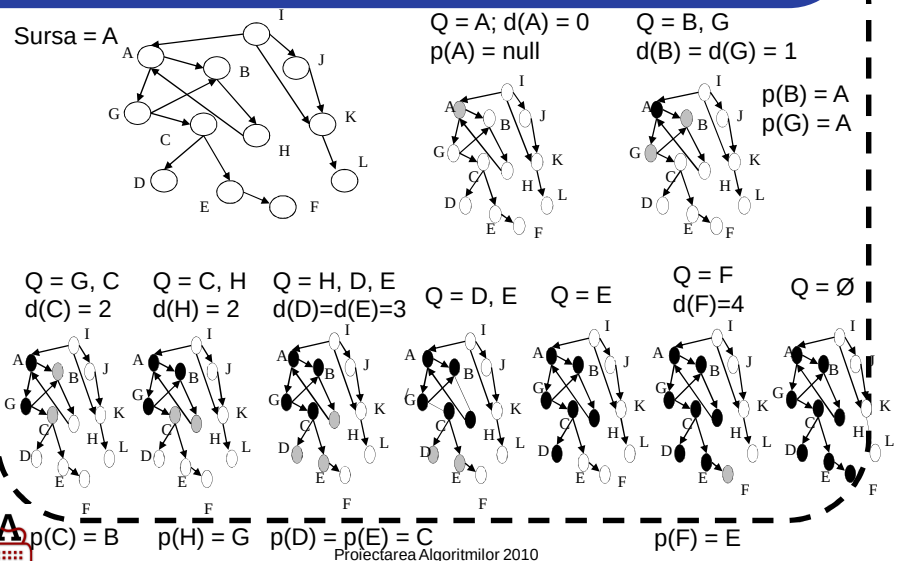
Sursa = A



BFS – Zona de explorare



BFS – Evoluția explorării



BFS – Proprietati (I)

- Lema 5.1. In cursul execuției BFS(s,G) $v \in Q \Leftrightarrow v \in R(s)$.
  - →BFS parcurge toate nodurile ce pot fi atinse din s;
  - ← toate nodurile ce pot fi atinse din s vor fi introduse cândva in coadă.
  - Dem prin inducție!
- Lema 5.2. $\forall (u,v) \in E, \delta(s,v) \leq \delta(s,u) + 1$
  - $\delta(s,v) \leq \delta(s,u) + 1$ in general sunt = când v este descoperit din u; < când v deja a fost descoperit înainte sa se ajungă in u.
  - Dem prin reducere prin absurd!

BFS – Proprietati (II)

- Lema 5.3. La terminarea BFS(s,G) există proprietatea $dist(s,u) \geq \delta(s,u)$.
  - Dem prin inducție folosind 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 si eliminare de elemente in/din Q.)

BFS – Proprietati (III)

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

    $d(u) < d(v) \Rightarrow dist(s,u) \leq dist(s,v)$.
- Teorema 5.1. BFS este corect si după terminare $\delta(s,u) = dist(s,u)$, $\forall$ u din V.
  - Dem prin inducție!

Complexitate ? Optimalitate?

BFS – Complexitate si Optimalitate

Complexitate:

O(n+m)

n = număr noduri

m = număr muchii

Optimalitate: DA

Parcurge tot graful? NU

Parcurgere in adancime (DFS)

- Nu mai avem nod de start, nodurile fiind parcurse in ordine.
- d(u) = momentul descoperirii nodului (se trece prima oară prin u si e totodată si momentul începerii explorării zonei din graf ce poate fi atinsă din u).
- f(u) = timpul de finalizare al nodului (momentul in care prelucrarea nodului a luat sfârșit)
  - Tot subarborele de adâncime dominat de u a fost explorat
  - Alternativ: tot subgraful accesibil din u a fost descoperit sau finalizat deja

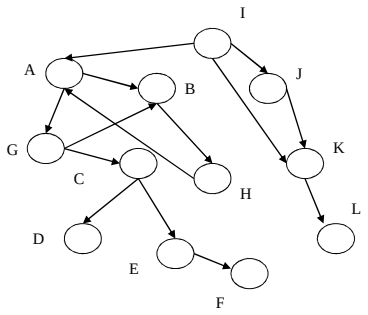
DFS – Structura de date

- Folosește o stiva (LIFO) pentru a reține nodurile ce trebuie prelucrate
  - In implementarile uzuale, stiva este rareori folosită explicit
  - Se apelează la recursivitate pentru a simula stiva
- Folosește o variabila globala timp pe baza căreia se calculează timpii de descoperire si de finalizare ai fiecărui nod.
- Pentru fiecare nod se reține:
  - Părintele;
  - d(u) – timpul de descoperire;
  - f(u)– timpul de finalizare;
  - culoarea nodului.

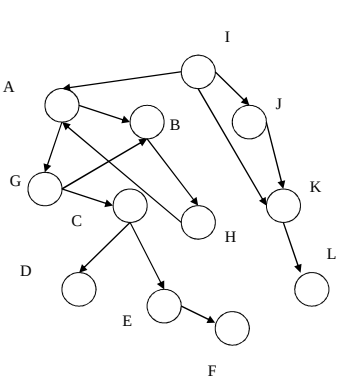
DFS – Algoritm

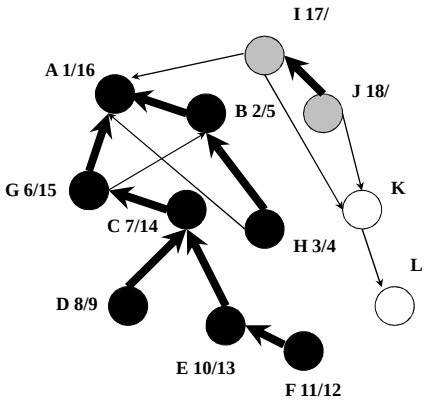
- DFS(G)
  - V = noduri(G)
  - *Pentru fiecare* nod u (u ∈ V)
    - c(u) = alb; p(u) = null; // inițializare structură date
  - timp = 0; // reține distanța de la rădăcina arborelui DFS pană la nodul curent
  - *Pentru fiecare* nod u (u ∈ V)
    - *Dacă* c(u) este alb
      - *Atunci* explorare(u); //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

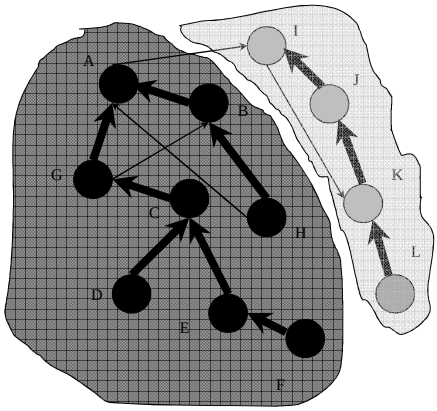


Calculul timpilor

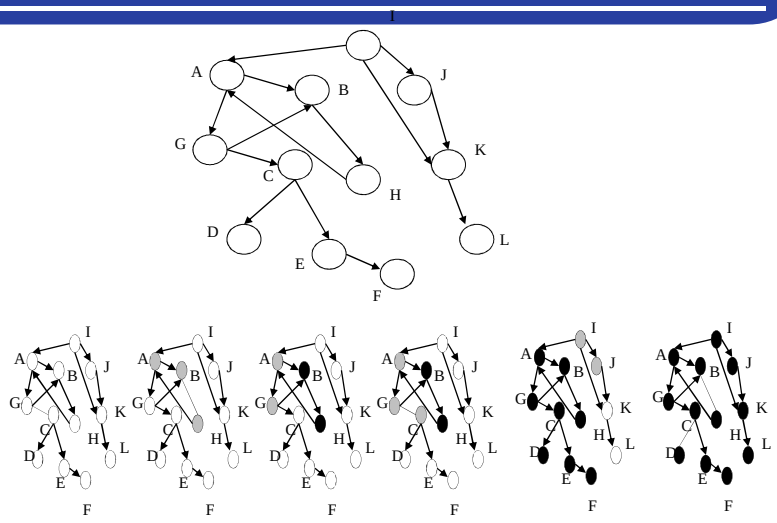




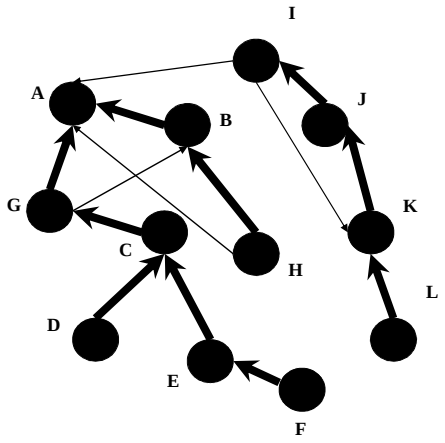
DFS – Zone de explorare



DFS – Evoluția explorării



Arborele de parcurgere în adâncime



DFS – Proprietati (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 este construită o cale v, p(v), p(p(v)),…, u.
  - Dem prin inducție!
- Teorema 5.2. G = (V,E); DFS(G) sparge graful G intr-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 in V(u), z in V(u) && p(z) = v}.

DFS – Proprietati (II)

- Teorema 5.3. Daca DFS(G) generează 1 singur arbore => G este conex. (Reciproca este adevărata?)
- Teorema 5.4. Teorema parantezelor:
  - $\forall$ u, v atunci I(u) $\cap$ I(v) = $\emptyset$ 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) exista o cale numai cu noduri albe u..v.
  - Dem prin inducție!

Clasificari ale arcelor grafului

- 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?

Clasificari ale arcelor grafului

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

DFS – Proprietati (III)

- Teorema 5.7. Intr-un graf neorientat, DFS poate descoperi doar arce directe si inverse.
  - Dem prin considerarea cazurilor posibile!
- Teorema 5.8. G = graf orientat; G ciclic ⇔ in timpul execuției DFS găsim arce inverse.
  - Dem prin exploatarea proprietăților de ciclu si de arc invers!

DFS – Complexitate si Optimalitate

Complexitate:

O(n+m)

n = număr noduri

m = număr muchii

Optimalitate: NU

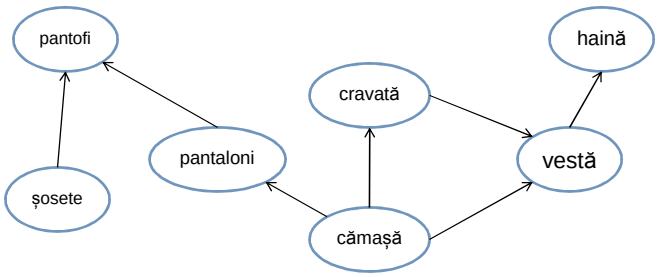
Parcurge tot graful? DA

Sortare topologică

- Se folosește la sortarea unei mulțimi parțial ordonată (nu orice pereche de elemente pot fi comparate).
- Fie A o mulțime parțial ordonată față de o relație de ordine $\propto$ ($\propto \subseteq A*A$) atunci $\exists$ e_1 si e_2 astfel incat e_1, e_2 nu pot fi comparate.
- O sortare topologică a lui A este o listă L = $<e_1,e_2 \ldots e_n>$, cu proprietatea că $\forall$ i, j, dacă $e_i \propto e_j$, atunci i < j.

Sortare topologică - Exemplu

- A = {pantofi, șosete, cravată, haină, vestă, pantaloni, cămașă}.



Sortare topologică

- G = (V,E) orientat, aciclic.
- Vs – secvența de noduri ai $\forall(u,v) \in E$, avem index(u) < index(v).
- Scop: Sortare_topologică(G) => Vs.
- Idee bazată pe DFS:
  - G = (V,E) orientat, aciclic; la sfârșitul DFS avem $\forall(u,v) \in E$, f(v) < f(u)
  - => colectam in Vs vârfurile in ordinea descrescătoare a timpilor f

Algoritm sortare topologică

- Sortare_topologică (G)
  - *Pentru fiecare* nod u (u ∈ V) {c(u) = alb;} // inițializări
  - Vs= ∅;
  - *Pentru fiecare* nod u (u ∈ V) // pentru fiecare componenta conexă
    - *Dacă* c(v) este alb
      - Vs = explorează (u,Vs) // prelucrez componenta conexă
  - *Întoarce* Vs
- Explorează (u,Vs)
  - c(u) = gri // prelucrez nodul, deci ii actualizez culoarea
  - *Pentru fiecare* nod v ∈ succs(u)
    - *Dacă* c(v) este alb *atunci* Vs = explorează (u,Vs) // recursivitate
    - *Dacă* c(v) este gri *atunci întoarce* Eroare: graf ciclic
  - c(u) = negru // am terminat prelucrarea nodului
  - *Întoarce* cons(u, Vs) // inserează nodul u la începutul lui Vs

Sortare topologică – Observație

- Observație: In general există mai multe sortări posibile!
- Ex:
  - cămașă, cravată, vestă, haină, șosete, pantaloni, pantofi
  - cămașă, pantaloni, cravată, vestă, haină, șosete, pantofi
  - șosete, cămașă, cravată, vestă, haină, pantaloni, pantofi
  - șosete, cămașă, pantaloni, cravată, vestă, haină, pantofi
  - șosete, cămașă, pantaloni, pantofi, cravată, vestă, haină

Complexitate?

Sortare topologică – Complexitate

Complexitate:

O(n+m)

n = număr noduri

m = număr muchii

Întrebări?

Bibliografie curs 6

- Giumale – Introducere in Analiza Algoritmilor cap. 5.2
- Cormen – Introducere în Algoritmi cap. 23.5
- http://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm