Proiectarea Algoritmilor

Curs 6 – Parcurgere în adâncime (DFS), Sortare Topologica & Componente Tare Conexe

Bibliografie

- 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

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

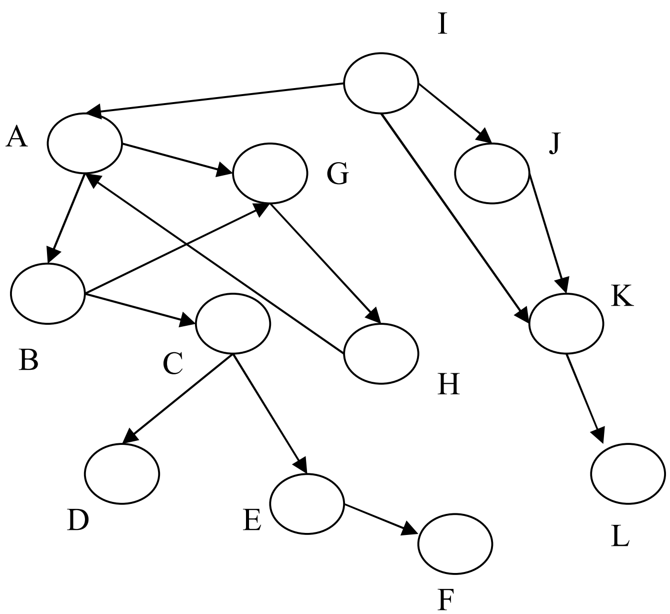
- Folosește o stiva (LIFO) pentru a reține nodurile ce trebuie prelucrate
  - În implementările uzuale, stiva este rareori folosită explicit;
  - Se apelează la recursivitate pentru a simula stiva.
- Folosește o variabilă globală *timp* pe baza căreia se calculează timpii de descoperire și de finalizare ai fiecărui nod.
- Pentru fiecare nod se rețin:
  - 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 (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

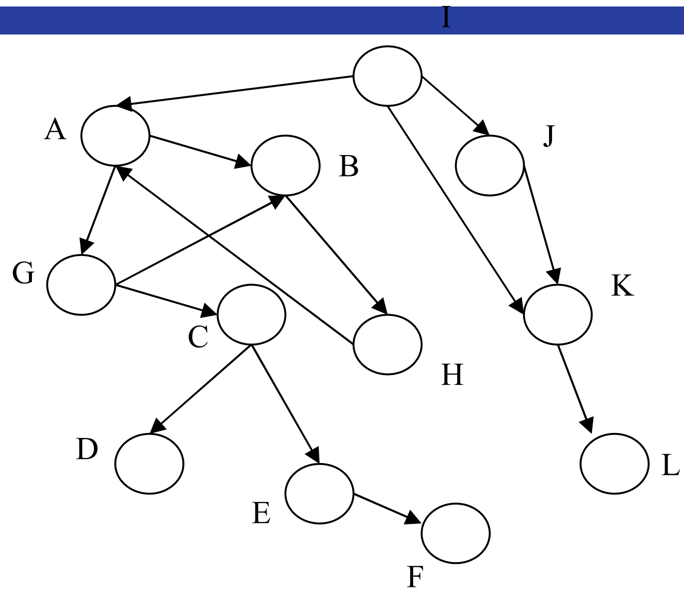
DFS – Exemplu

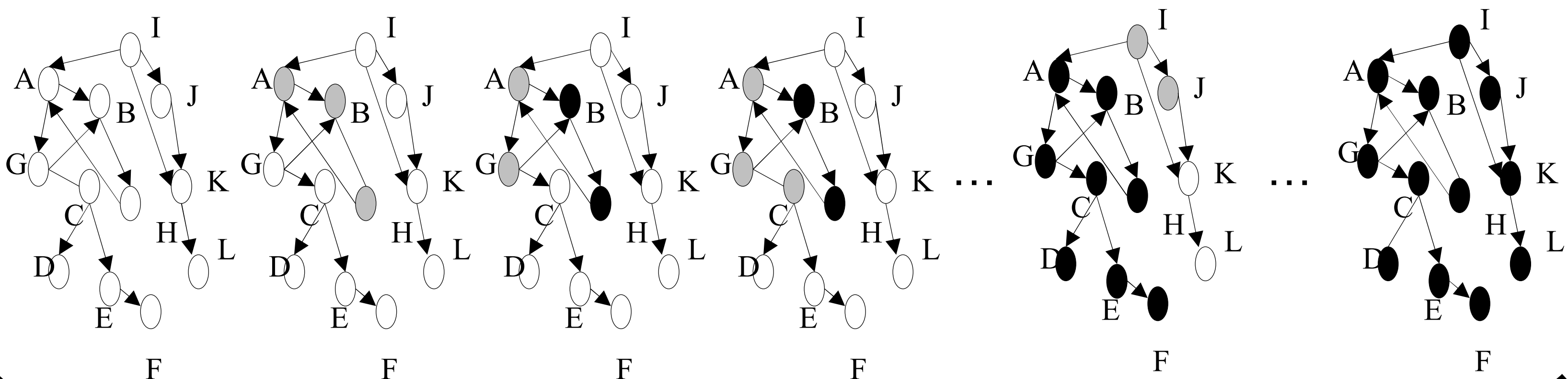


- DFS(G)
  - V = noduri(G)
  - **Pentru fiecare** nod u (u ∈ V)
    - c(u) = alb; p(u) = null; // inițializare structură date
  - timp = 0; // reține distanța de la rădăcina arborelui // DFS pană la nodul curent
  - **Pentru fiecare** nod u (u ∈ V)
    - **Dacă** c(u) este alb
      - **Atunci** explorare(u);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

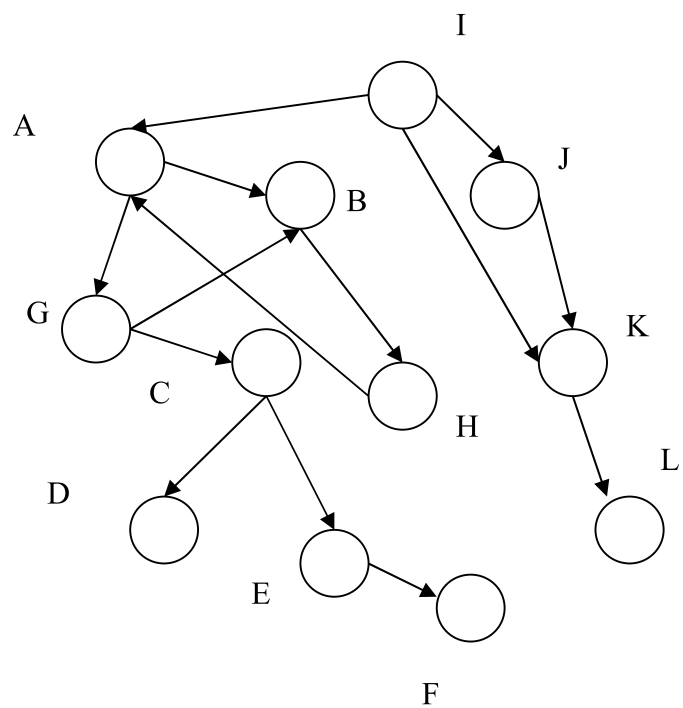
- explorare(u)
  - d(u) = ++ timp; // timpul de descoperire al nodului u
  - c(u) = gri; // nod in curs de explorare
  - **Pentru fiecare** nod v (v ∈ succs(u)) // încerc sa // prelucrez vecinii
    - **Dacă** c(v) este alb
      - **Atunci** {p(v) = u; explorare(v);} // dacă nu au // fost prelucrați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

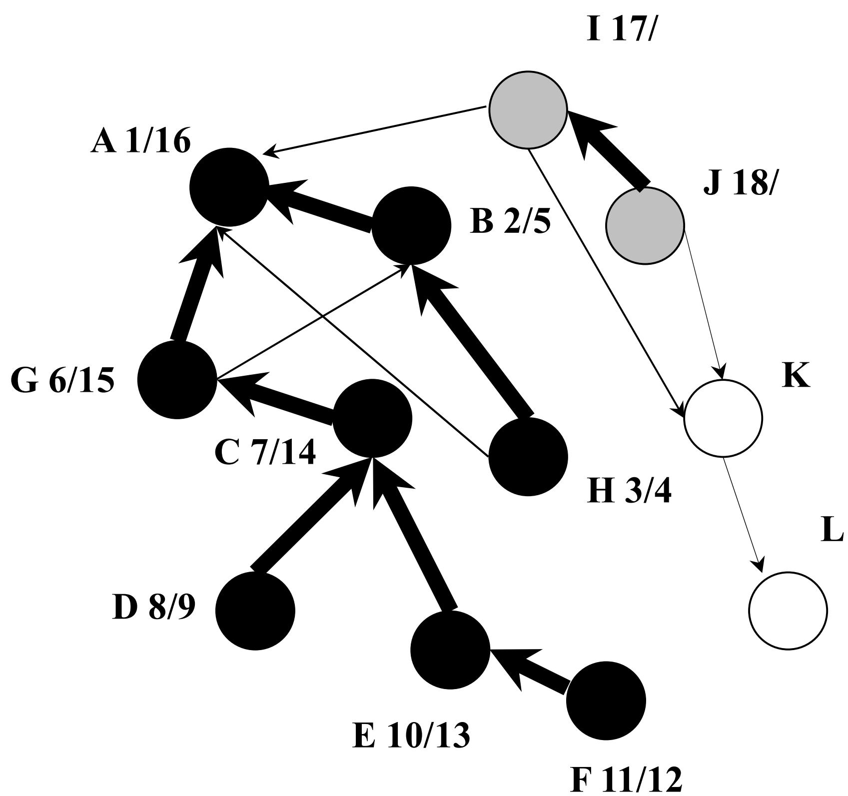
DFS – Evoluția explorării



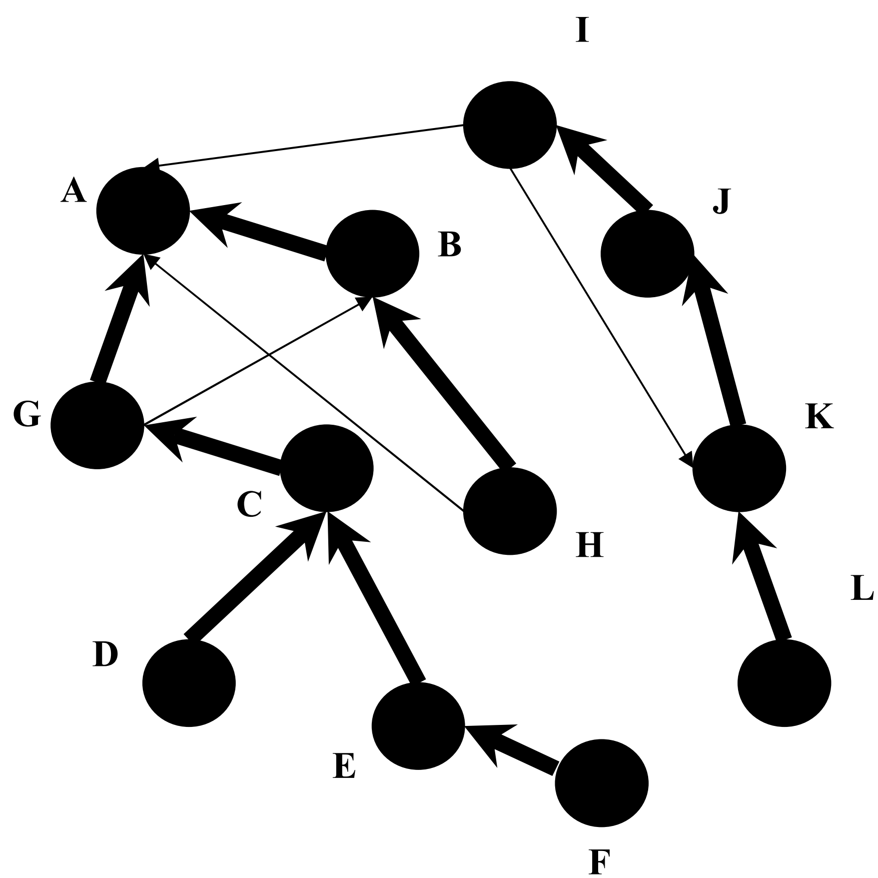


Calcului timpilor

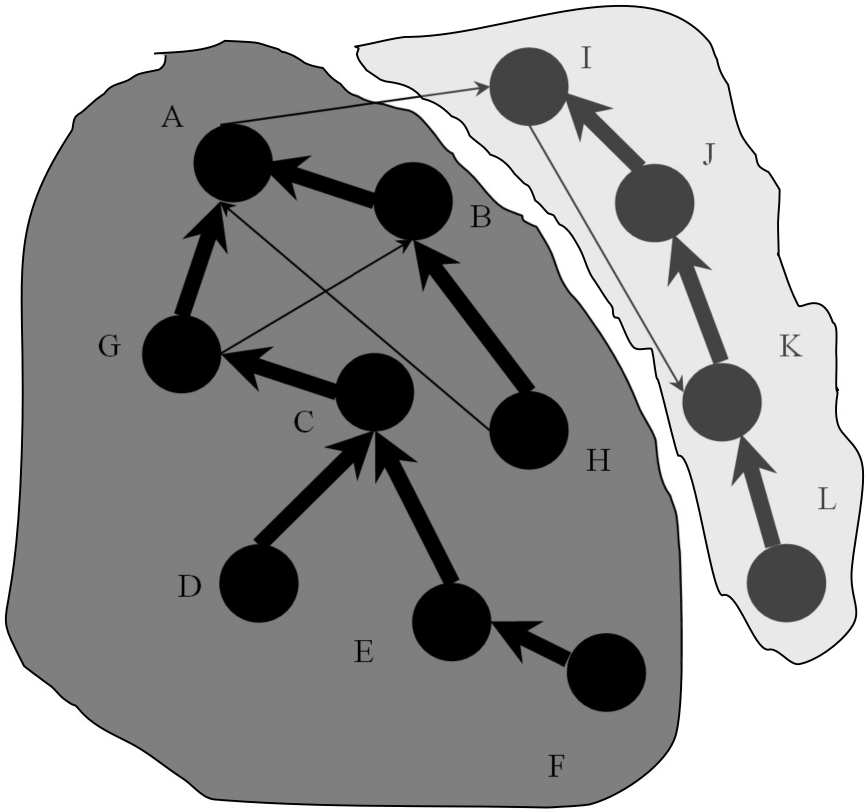




Arborele de parcurgere în adâncime



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 }.
  - Dem: Conform algoritmului, se pot identifica noduri în ciclul principal sau din funcția de explorare. Dacă u e descoperit în ciclul principal, atunci ∃ o cale către toți succesorii dată de părinți → V(u) = { v | d(u) < d(v) < f(u) } iar arcele sunt chiar cele ce desemnează părinții

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 $I(u) \cap I(v) = \emptyset$ sau $I(u) \subset I(v)$ sau $I(v) \subset I(u)$.
  - Dem prin considerarea tuturor combinațiilor posibile!
  - a) $I(u) < I(v)$: $d(u) < f(u) < d(v) < f(v)$: $v \notin R(u)$ $\rightarrow$ v rămâne alb pe durata prelucrării lui u $\rightarrow$ $f(u) < d(v)$
  - b) $I(v) \subset I(u)$: $d(u) < d(v) < f(v) < f(u)$: $v \in R(u)$ $\rightarrow$ v este descoperit din u și devine negru înaintea terminării prelucrării lui u $\rightarrow$ $f(u) < f(v)$
  - c) $I(v) < I(u)$: $d(v) < f(v) < d(u) < f(u)$ Analog a)
  - d) $I(u) \subset I(v)$: $d(v) < f(u) < f(u) < f(v)$ Analog b)

DFS – Proprietăți (III)

- 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.
  - Demonstrație prin inducție!
  - v ∈ V(u) → la momentul d(u) există o cale numai cu noduri albe u..v
    - Dacă v ∈ V(u) → $\exists$ o cale unică v.. α..u de pointeri π. Fie un nod oarecare z din calea α. → (Teorema 5.5) d(u) < d(z) < f(z) < f(u) → la d(u), c(z) = alb și cum z a fost ales la întâmplare → toate nodurile de pe calea α sunt albe la d(u).
  - la momentul d(u) există o cale numai cu noduri albe u..v → v ∈ V(u)
    - Fie u = $v_0 v_1 \ldots v_p$ = v o cale din G, a.î. La d(u) avem $c(v_i)$ = alb, $\forall$i $\in$ 0,p. Dem prin inducție după i că v_i) este descendentul lui u în Arb(u).
    - Caz de bază: d(u) < $d(v_1)$ < f(u) → (Teorema 5.5) v_1 descendent al lui u Adevărat
    - Pas inducție: v_i descendent al lui u → v_{i+1} descendent al lui u
      - v_i descendent al lui u → d(u) < $d(v_i)$ < $f(v_i)$ < f(u) . Cum v_{i+1} este alb la d(u) și este succesorul lui v_i → v_{i+1} este descoperit după d(u), dar înainte de $f(v_i)$. → d(u) < $d(v_{i+1})$ < $f(v_i)$ < f(u) → v_{i+1} descendent al lui u

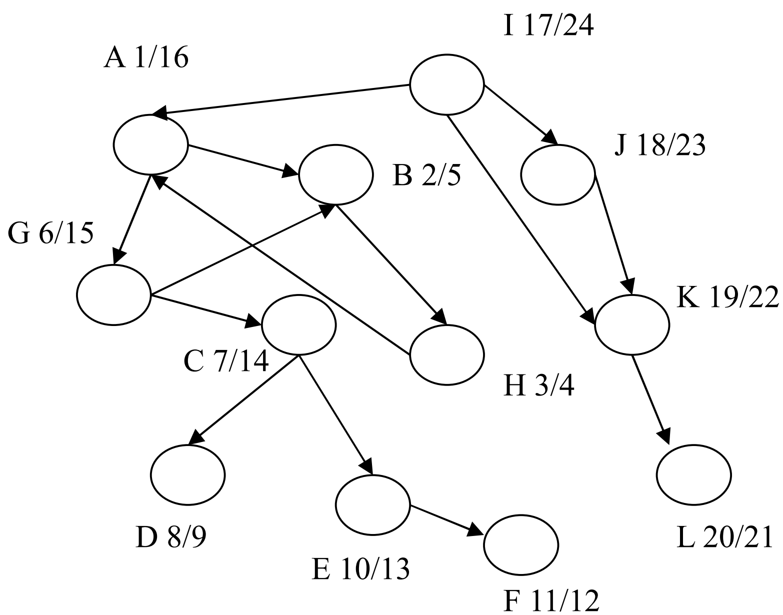
Clasificări ale arcelor grafului (I)

- 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)

întree 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 (IV)

- Teorema 5.7. Într-un graf neorientat, DFS poate descoperi doar muchii directe și inverse.
  - Dem prin considerarea cazurilor posibile!
  - Fie muchia (u,v) ∈ E și pp. d(u) < d(v). Muchia poate fi străbătută din u sau din v:
    - Caz (u,v): c(u) = gri, c(v) = alb → muchie directă
    - Caz (v,u): la d(u) ∃ o cale cu noduri albe u..v → (Teorema drumurilor albe) v este descendent al lui u în Arb(u) → (Teorema 5.5) d(u) < d(v) < f(v) < f(u) → în intervalul (d(v), f(v)) când se investighează (v,u) c(u) = c(v) = gri → muchie inversă

DFS – Proprietăți (V)

- 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!
  - G ciclic → DFS descoperă arce inverse
    - Fie un ciclu și u primul nod din ciclu descoperit de DFS. Atunci ∃ u..v si (v,u) arcul care închide ciclul. d(u) < d(v) (ipoteză) iar u..v conține doar noduri albe → (Teorema drumurilor albe & Teorema 5.5) d(u) < d(v) < f(v) < f(u) → în intervalul (d(v), f(v)) când se explorează (v,u) c(u) = c(v) = gri → arc invers
  - DFS descoperă arce inverse → G ciclic
    - Fie (v,u) arc invers → d(u) < d(v) < f(v) < f(u) → (Teorema 5.5) → v este descendent al lui u în Arb(u) → ∃ calea u..v care inchide ciclul

DFS – Complexitate și Optimalitate

Complexitate:

$O(n+m)$

n = număr noduri

m = număr muchii

Optimalitate: NU

Parcurge tot graful? DA

Sortare topologică

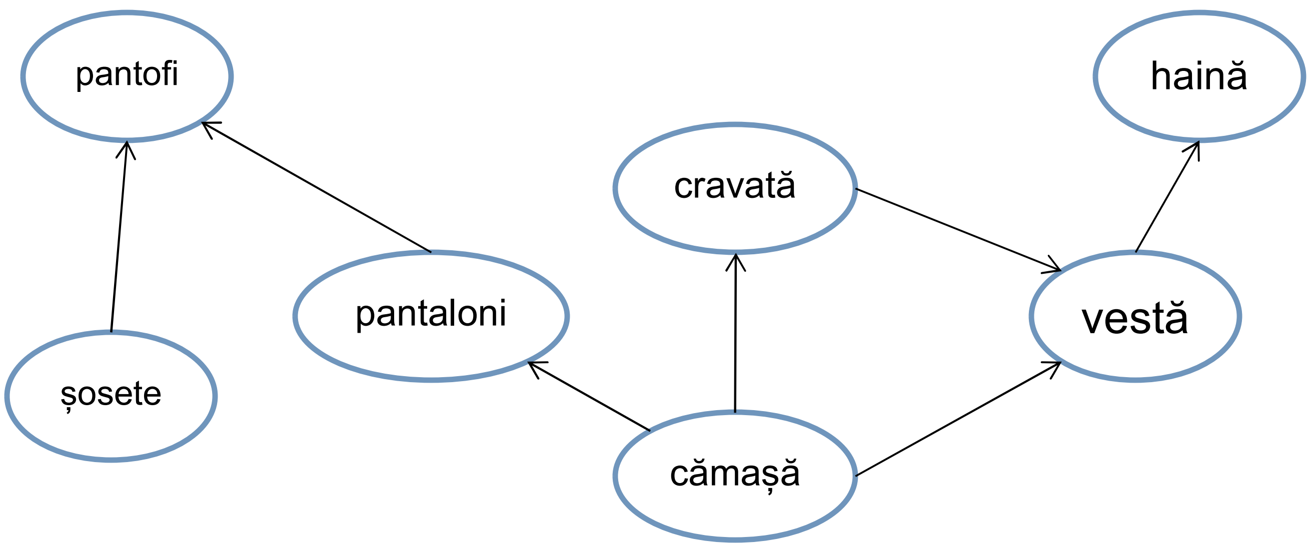
- Se folosește la sortarea unei mulțimi parțial ordonate (nu orice pereche de elemente pot fi comparate).

- Fie A o mulțime parțial ordonată față de o relație de ordine $\propto$ ($\propto \subseteq A*A$) atunci $\exists$ e_1 și e_2 astfel încât 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.
- V_S – secvența de noduri a.î. $\forall(u,v) \in E$, avem index(u) < index(v).
- Scop: Sortare_topologică(G) => V_S.
- Idee bazată pe DFS:
  - G = (V,E) orientat, aciclic; la sfârșitul DFS avem $\forall(u,v) \in E$, f(v) < f(u)
  - => colectăm în V_S vârfurile în ordinea descrescătoare a timpilor f

Algoritm sortare topologică

- sortare_topologică (G)
  - **Pentru fiecare** nod u (u ∈ V) {c(u) = alb;} // inițializări
  - $V_S = \varnothing$;
  - **Pentru fiecare** nod u (u ∈ V) // pentru fiecare componentă conexă
    - **Dacă** c(u) este alb
      - V_S = explorează (u, V_S) // prelucrez componenta conexă
  - **Întoarce** V_S

- explorează (u, V_S)
  - c(u) = gri // prelucrez nodul, deci îi actualizez culoarea
  - **Pentru fiecare** nod v (v ∈ succs(u))
    - **Dacă** c(v) este alb **atunci** V_S = explorează (v, V_S) // recursivitate
    - **Dacă** c(v) este gri **atunci Întoarce** Eroare: graf ciclic
  - c(u) = negru // am terminat prelucrarea nodului
  - **Întoarce** cons(u, V_S) // inserează nodul u la începutul lui V_S

Sortare topologică – Observație

- Observație: În 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
- Care e numărul maxim de sortări topologice?
- Dar numărul minim?
- Când se obțin aceste valori?

Complexitate?

Sortare topologică – Complexitate

Complexitate:

$O(n+m)$

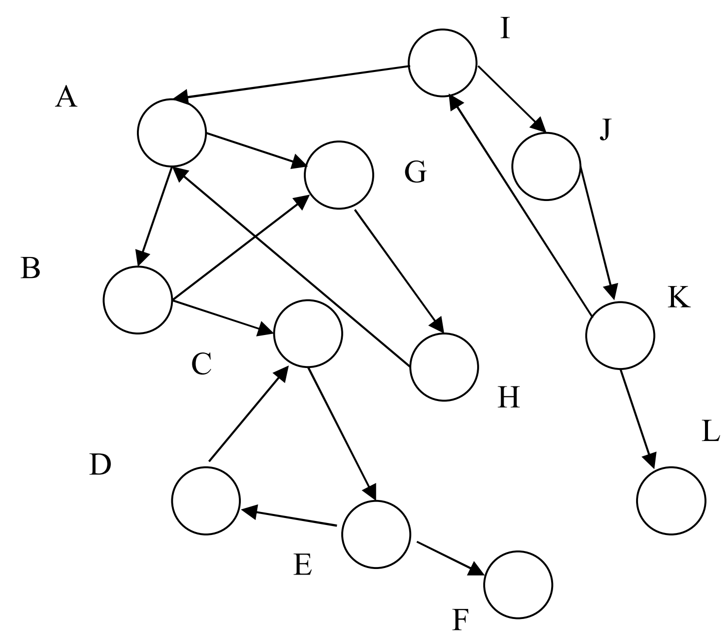
n = număr noduri

m = număr muchii

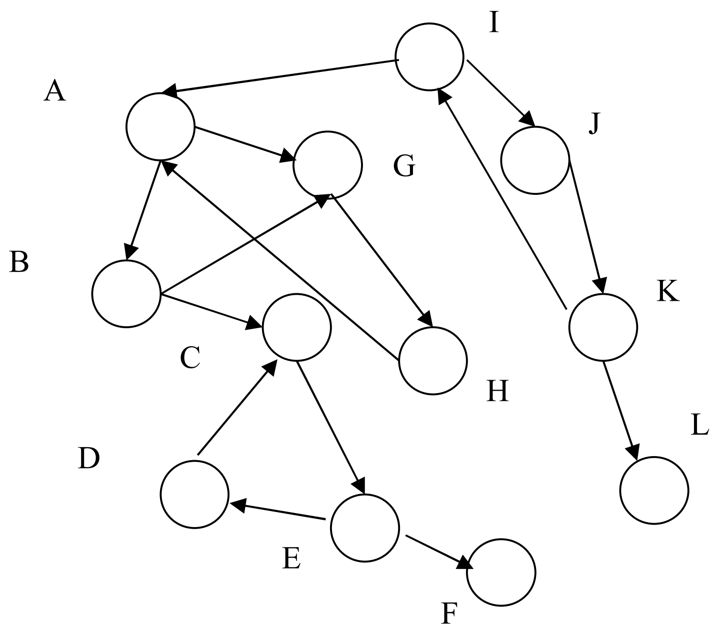
Componente Tare Conexe (CTC)

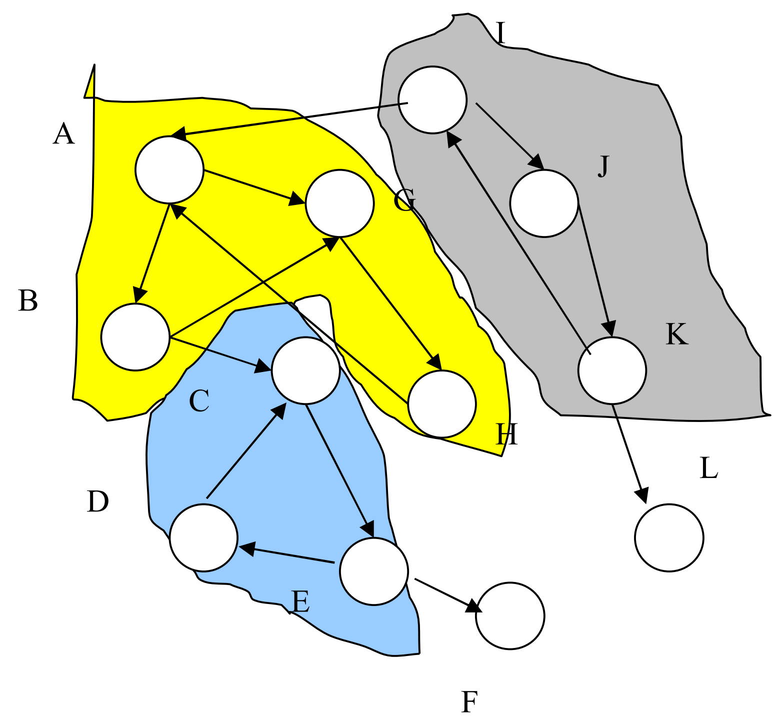
- Definiție: Fie G = (V,E) un graf orientat. G este tare-conex $\Leftrightarrow \forall$ u,v$\in$V $\exists$ o cale u..v și o cale v..u (u$\in$R(v) și v$\in$R(u)).
- Definiție: G = (V,E) graf orientat. G' = (V',E'), V'$\subseteq$V, E'$\subseteq$E. G' este o CTC a lui G $\Leftrightarrow$ G' e tare-conex ($\forall$ u,v$\in$V', u$\in$R(v) și v$\in$R(u)) și G' este maximal.
- Lema 5.6: G = (V,E) graf orientat, G' CTC, $\forall$ u,v$\in$V' => $\forall$ u..v din G are noduri exclusiv in V'
  - Dem: $\forall$ z a.î. u..z..v => z$\in$R(u) și v$\in$R(z). Dar u$\in$R(v) => z$\in$R(v) => v și z sunt în aceeași CTC.

Exemplu (I) – determinare CTC



Exemplu (II) – determinare CTC(2)



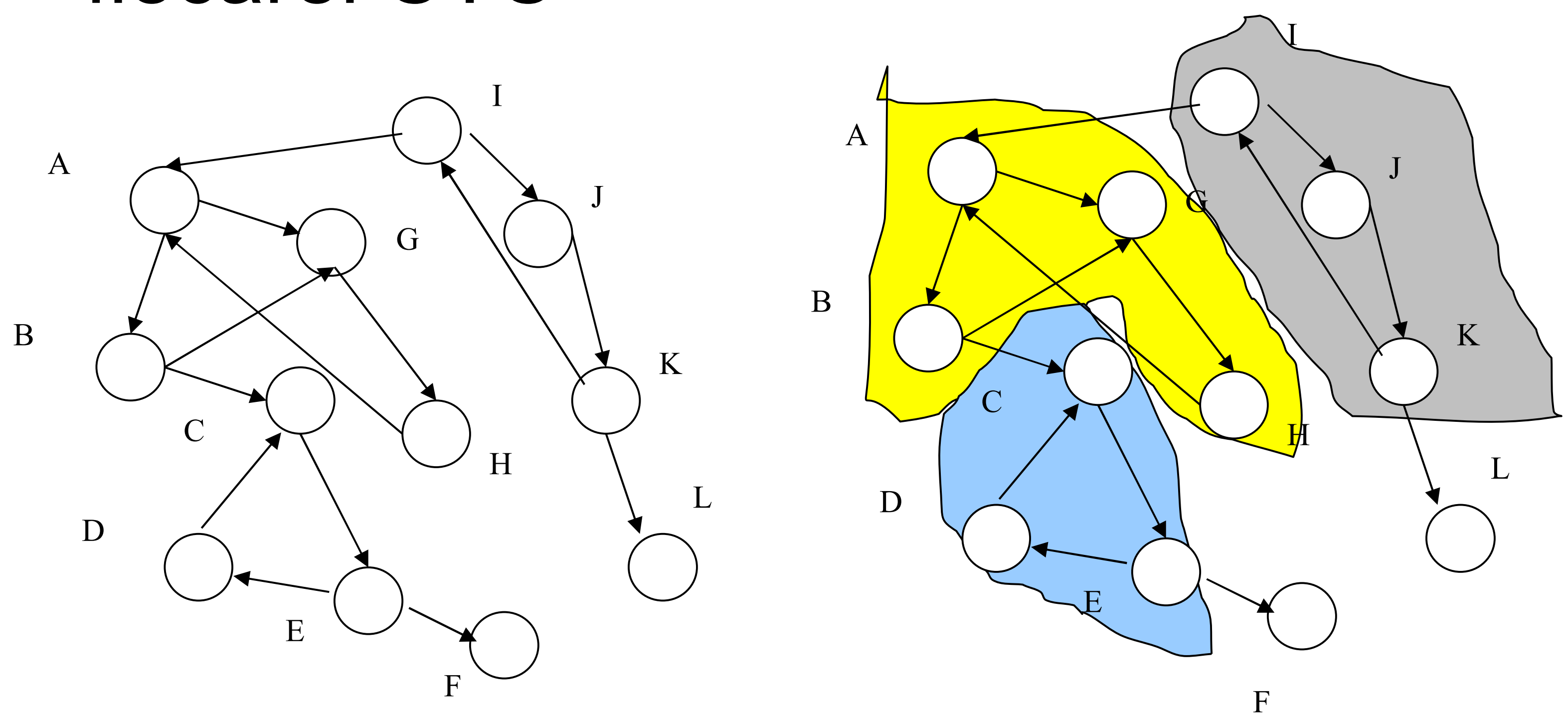


Componente Tare Conexe (CTC)

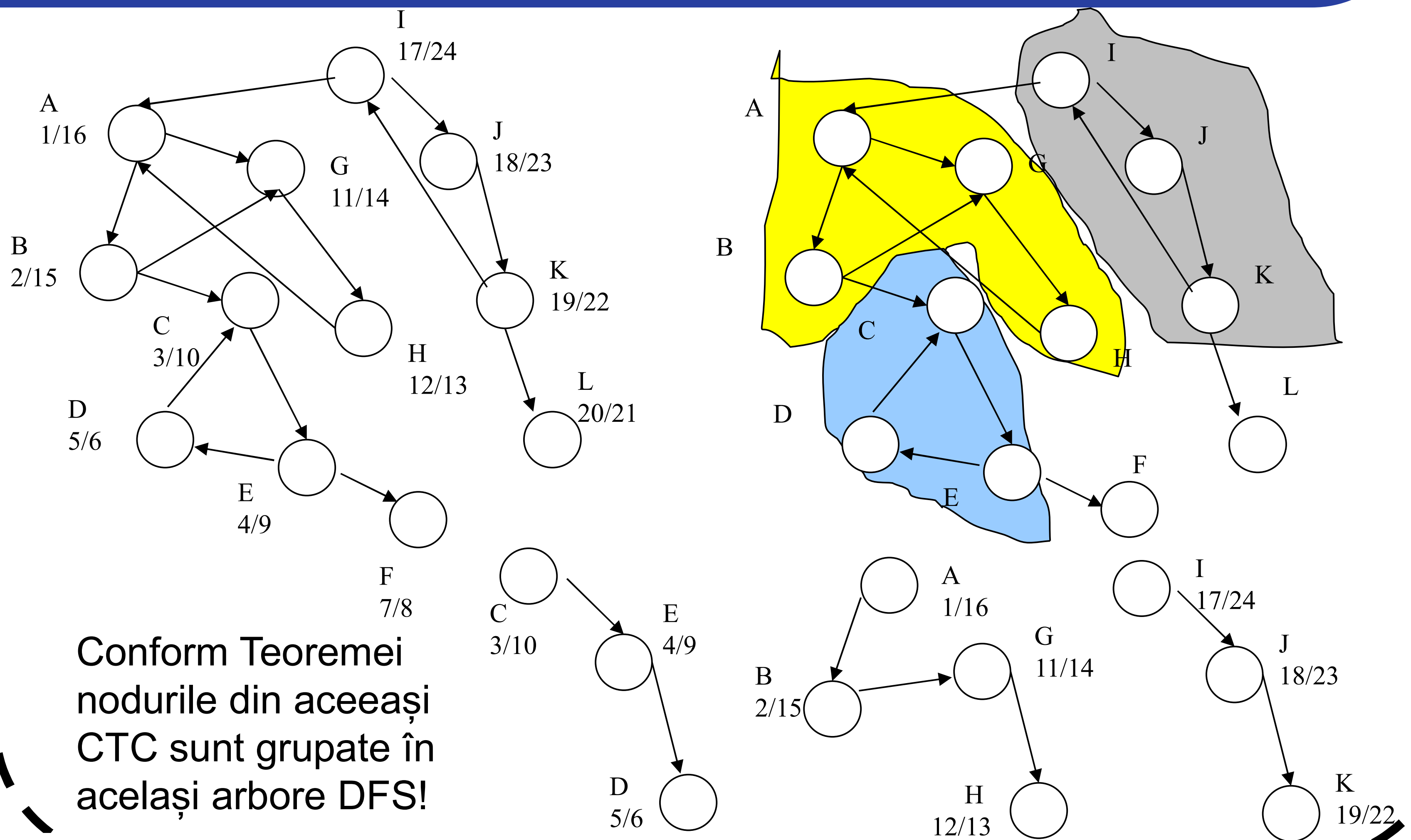
- Teorema 5.10: $G = (V,E)$ orientat, $G' = (V',E')$ o CTC a lui G. Toate nodurile $v \in V'$ sunt grupate in același Arb(u) construit de DFS(G), unde u este primul nod descoperit al componentei.
  - Dem: $\forall\ v \in V'$, $v \neq u$, $\exists$ u..v drum cu noduri albe la momentul descoperirii d(u); toate nodurile drumului sunt în V' (conf. Lema 5.6) => (din Teorema drumurilor albe) v este descendent al lui u în Arb(u)

Exemplu (III) – DFS

- Aplicare DFS pornind din primul nod al fiecărei CTC



Exemplu (IV) - DFS(2)



Conform Teoremei nodurile din aceeași CTC sunt grupate în același arbore DFS!

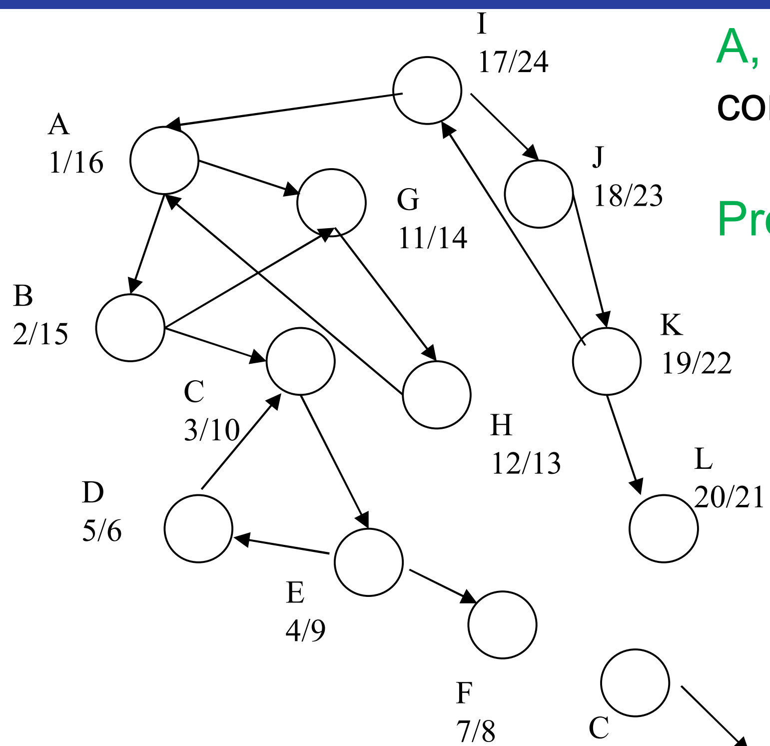
Componente Tare Conexe (CTC)

- Definiție: G = (V,E) orientat, $u \in V$. $\Phi(u)$ = strămoș DFS al lui u determinat în cursul DFS(G) dacă:
  - $\Phi(u) \in R(u)$
  - $f(\Phi(u)) = \max\{f(v) \mid v \in R(u)\}$

- Ce e $\Phi(u)$? $\Phi(u)$ este primul nod din CTC descoperit de DFS(G)

- Teorema 5.11: $\Phi(u)$ satisface următoarele proprietăți:
  1. $f(u) \leq f(\Phi(u))$ când e egalitate? u este primul nod din CTC
  2. $\forall v \in R(u)$, $f(\Phi(v)) \leq f(\Phi(u))$ ce înseamnă ca e egalitate? u si v sunt in aceeași CTC
  3. $\Phi(\Phi(u)) = \Phi(u)$ Dem: $\Phi(\Phi(u)) \in R(\Phi(u)) \xrightarrow{1} f(\Phi(\Phi(u))) \leq f(\Phi(u))$ și $\xrightarrow{2} f(\Phi(\Phi(u))) \geq f(\Phi(u)) \rightarrow f(\Phi(\Phi(u))) = f(\Phi(u)) \rightarrow \Phi(\Phi(u)) = \Phi(u)$

Exemplu (V) – stramosi



A, C, I sunt strămoși DFS ai nodurilor din componenta conexă din care fac parte.

Proprietățile strămoșilor:

- $f(u) \leq f(\Phi(u))$

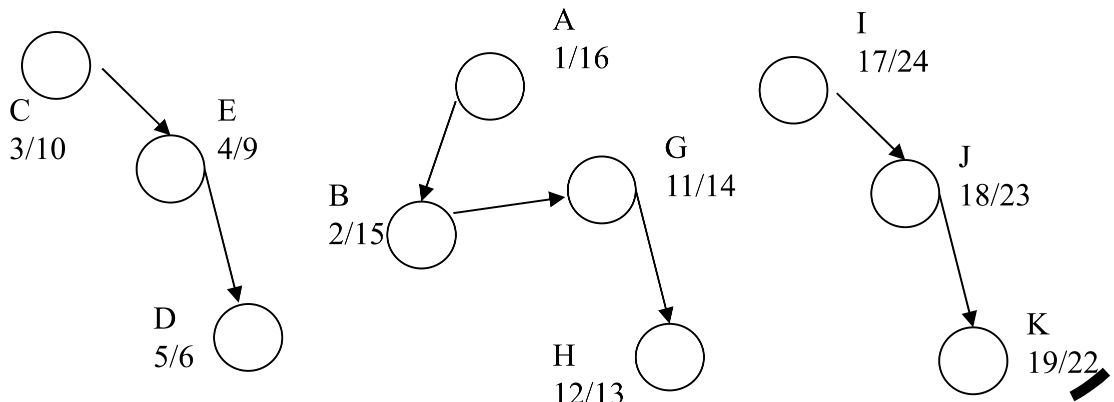
Ex: $f(B) \leq f(\Phi(B)) = f(A)$

- $\forall\ v \in R(u),\ f(\Phi(v)) \leq f(\Phi(u))$

Ex: $E \in R(B),\ f(C) = f(\Phi(E)) \leq f(\Phi(B)) = f(A)$

- $\Phi(\Phi(u)) = \Phi(u)$

Ex: $\Phi\ (\Phi(E)) = \Phi(C) = C = \Phi(E)$



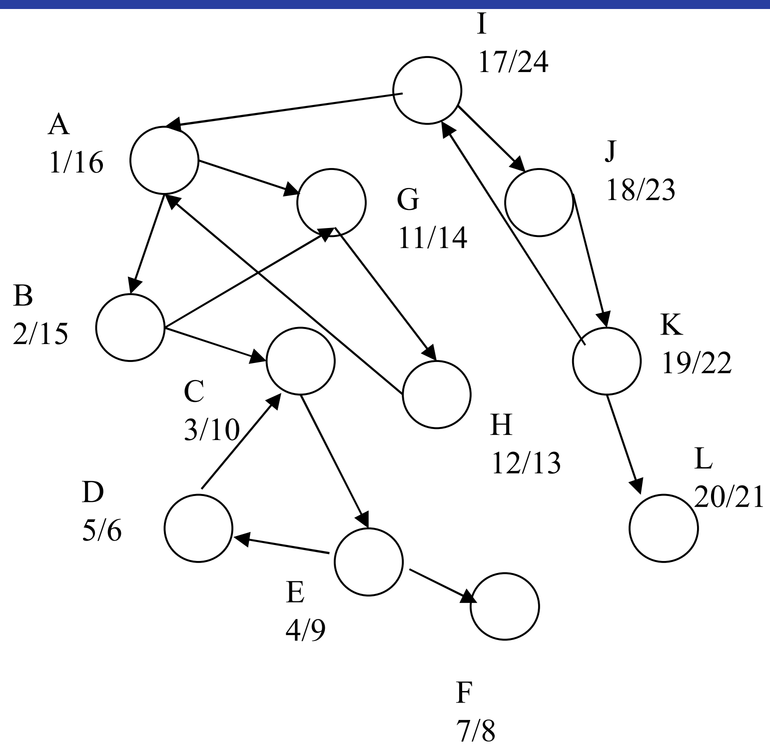
Componente Tare Conexe (CTC)

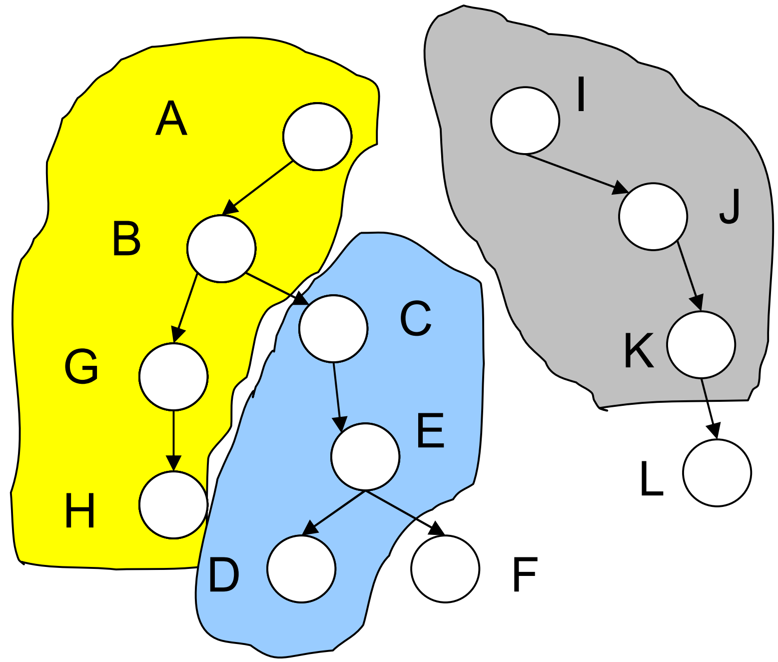
- Din Definiție: $\Phi(u) \in R(u)$ și $f(\Phi(u)) = \max\{f(v) \mid v \in R(u)\}$.

- Teorema 5.12. $G = (V,E)$ orientat, $\forall\ u \in V$, u este descendent al lui $\Phi(u)$ în $Arb(\Phi(u))$ construit de DFS.
  - Dem: prin considerarea tuturor culorilor posibile ale lui $\Phi(u)$ la momentul $d(u)$.

- Teorema 5.13. $G = (V,E)$ orientat, $\forall\ u,v \in V$; u și v aparțin aceleiași CTC $\Leftrightarrow \Phi(u) = \Phi(v)$.
  - Dem folosind proprietățile strămoșilor :
    - $\forall$ $u,v \in$ aceleiași CTC => $\Phi(u) = \Phi(v)$: $v \in R(u)$, $f(\Phi(v)) \le f(\Phi(u))$ și $u \in R(v)$, $f(\Phi(u)) \le f(\Phi(v))$ => $f(\Phi(u)) = f(\Phi(v)) \rightarrow \Phi(u) = \Phi(v)$
    - $\Phi(u) = \Phi(v)$ => $\Phi(u) \in R(u)$ => u și $\Phi(u) \in$ aceleiași CTC
și $\Phi(u) \in R(v)$ => v și $\Phi(u) \in$ aceleiași CTC
=> u și v $\in$ aceleiași CTC

Exemplu (VI)



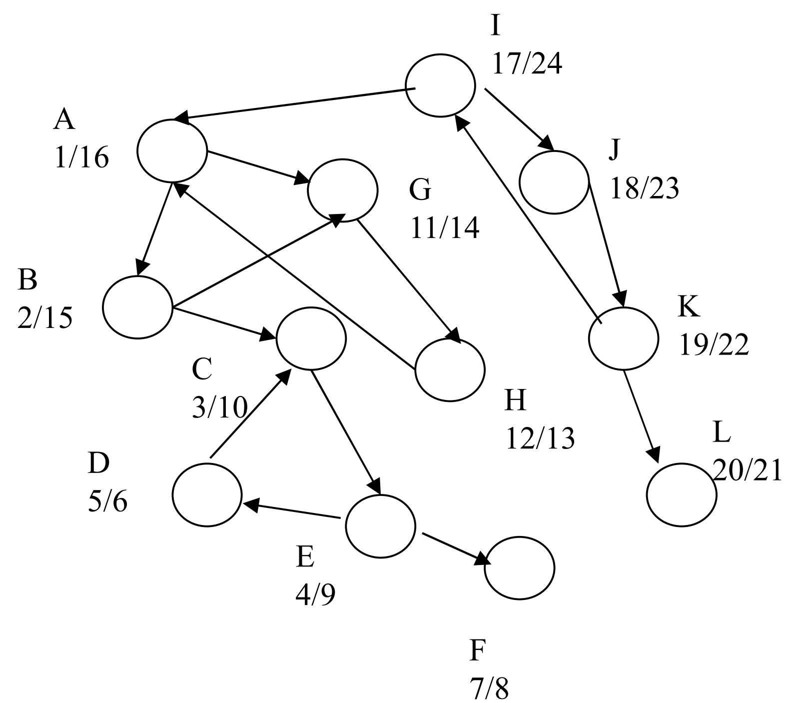


Primul nod dintr-o CTC descoperit prin DFS va avea drept copii în arborele generat de DFS toate elementele componentei conexe!

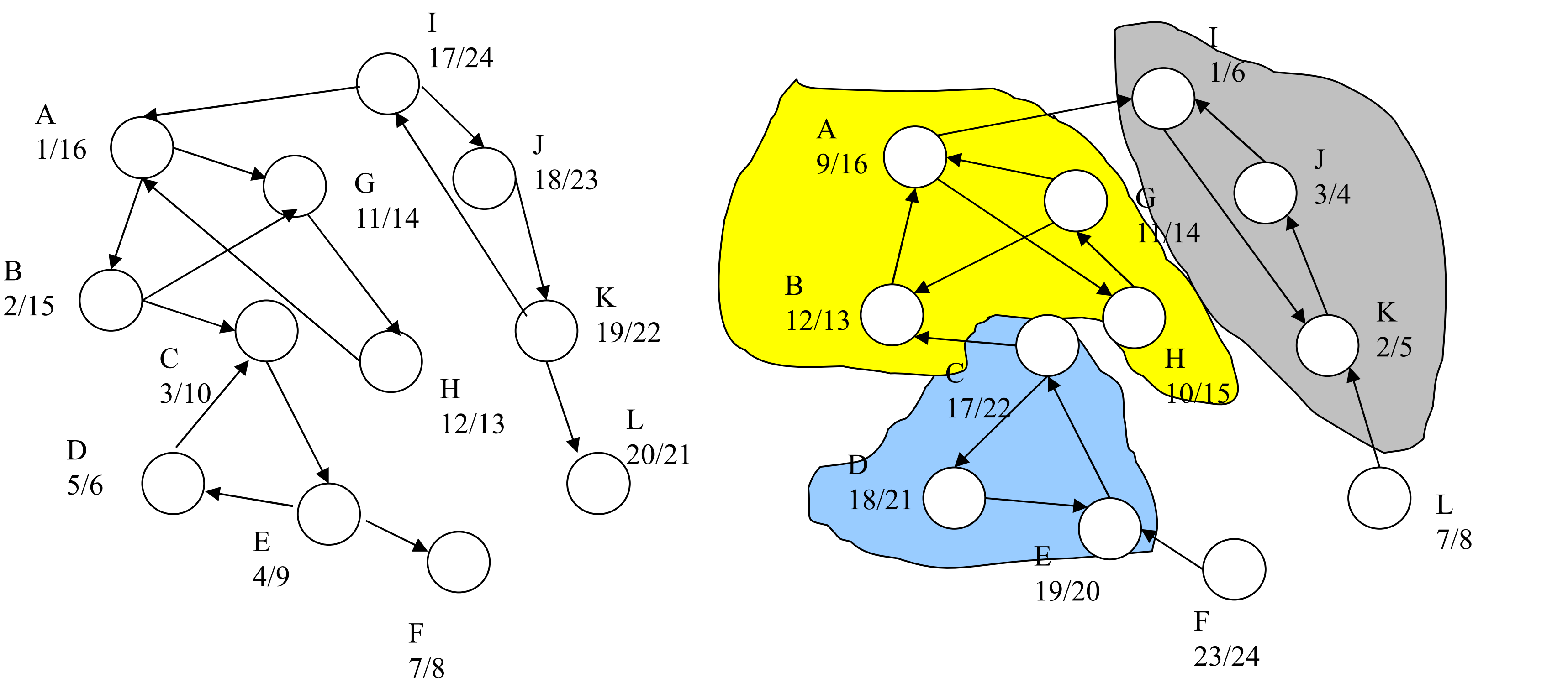
Componente Tare Conexe (CTC)

- Probleme:
  - vrem ca fiecare arbore construit să conțină o CTC.
  - trebuie să eliminăm nodurile care nu sunt în componenta conexă.
- Idee – eliminăm nodurile ce nu aparțin CTC! Cum?
  - Dacă aparțin Arb(u) și nu CTC => ∃ u..v și ∄ v..u.
  - → DFS pe graful transpus, în ordinea inversă a timpilor de finalizare obținuți din DFS pe graful normal!

Exemplu (VII) – DFS (G^T)

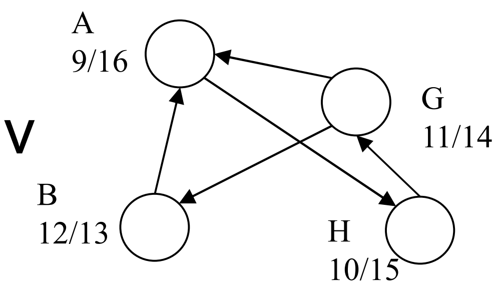


Exemplu (VIII) – DFS (G^T) (2)

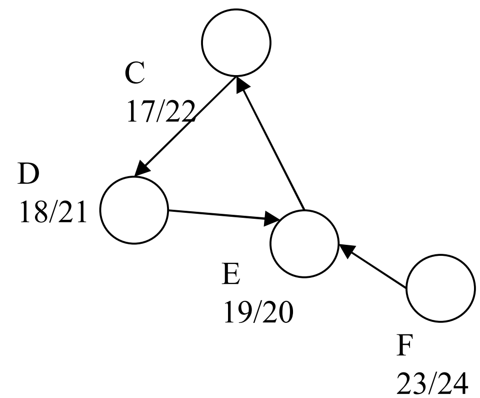


Componente Tare Conexe (CTC)

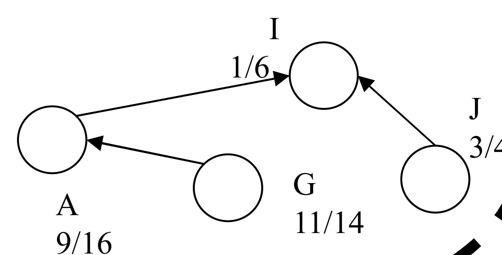
- Cazuri în DFS(G^T):
  - v este în CTC descoperita din u → v poate fi descoperit din u și în DFS(G^T).



  - v$\notin$CTC dar v$\in$Arb(u) în DFS(G)→ nu va fi atins în DFS(G^T) din u.

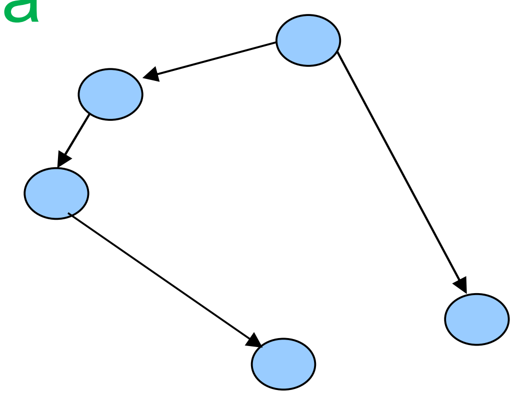


  - v$\notin$CTC dar $\exists$ v..u în G → f(v) > f(u) → v va fi deja colorat în negru când se explorează u.



Observatii

- Înlocuind componentele tare conexe cu noduri obținem un graf aciclic. De ce?
- Pentru că altfel am avea o singură CTC!



- Prima parcurgere DFS este o sortare topologică. De ce?
- Pentru că sortează nodurile în ordinea descrescătoare a timpilor de finalizare a strămoșilor fiecărei CTC!

Pseudocod algoritm CTC

- Algoritmul lui Kosaraju:
- CTC(G)
  - DFS(G)
  - G^T = transpune(G)
  - DFS(G^T) (în bucla principală se tratează nodurile în ordinea descrescătoare a timpilor de finalizare de la primul DFS)
- Componentele conexe sunt reprezentate de pădurea de arbori generați de DFS(G^T).

Complexitate?

Algoritmul lui Kosaraju

Complexitate

$O(n+m)$

n = numar noduri

m = numar muchii

Corectitudine algoritm CTC (1)

- Teoremă: Algoritmul CTC calculează corect componentele tare conexe ale unui graf G = (V,E).
- Dem. prin inducție după nr. de arbori de adâncime găsiți de DFS al G^T că vârfurile din fiecare arbore formează o CTC:
  - Fiecare pas demonstrează că arborele format în acel pas e o CTC, presupunând că toți arborii produși deja sunt CTC.
  - P_1: trivial pentru că $\nexists$ arbori anteriori.
  - $P_n \rightarrow P_{n+1}$: Fie arborele T obținut in pasul curent având rădăcina r. Notăm $C_r = \{v \in V \mid \Phi(v) = r\}$.

Corectitudine algoritm CTC (2)

- Demonstrăm că $u \in T \Leftrightarrow u \in C_r$:
- $u \in C_r \rightarrow u \in T$:
  - $u \in C_r \rightarrow \exists$ r..u $\rightarrow$ toate nodurile din C_r ajung în același arbore DFS (Arb(r)). Dar $r \in C_r$ și r e rădăcina lui T $\rightarrow \forall u \in C_r \Rightarrow u \in T$
- $u \in T \rightarrow u \in C_r$: demonstrăm că $\forall$ w a.î. $f(\Phi(w)) > f(r)$ sau $f(\Phi(w)) < f(r)$, $w \notin T$
  - Dacă $f(\Phi(w)) > f(r) \rightarrow$ la d(r), w e deja pus în CTC cu rădăcina $\Phi(w)$ pt. că nodurile sunt considerate în ordinea inversă a timpilor de finalizare $\rightarrow w \notin T$
  - Dacă $f(\Phi(w)) < f(r) \rightarrow w \notin T$ pt. că altfel ($w \in T$) $\rightarrow \exists$ r..w în $G^T \rightarrow \exists$ w..r în G $\rightarrow r \in R(w)$, $\rightarrow f(\Phi(w)) \geq f(\Phi(r)) = f(r)$ (F)
  - $\rightarrow$ T conține doar nodurile pt. care $\Phi(w) = r \rightarrow T = C_r$

ÎNTREBĂRI?

Bibliografie curs 8

[1] Giumale – Introducere în Analiza Algoritmilor cap. 5.3, 5.4, 5.4.1

[2] Cormen – Introducere în Algoritmi cap. Heap-uri binomiale (20), Heap-uri Fibonacci (21), Drumuri minime de sursă unica - primele 2 subcapitole (25.1 și 25.2)

[3] R. Sedgewick, K. Wayne - Algorithms and Data Structures Fall 2007 – Curs Princeton - http://www.cs.princeton.edu/~rs/AlgsDS07/06PriorityQueues.pdf

[4] Heap Fibonacci: http://www.cse.yorku.ca/~aaw/Jason/FibonacciHeapAnimation.html