

Proiectarea Algoritmilor

| CA: Stefan Trausan-Matu stefan.trausan@cs.pub.ro

| CB: Traian Rebedea traian.rebedea@cs.pub.ro

| CC: Costin Chiru costin.chiru@cs.pub.ro

Obiectivele cursului (I)

- Cunoașterea unui set important de algoritmi și metode de rezolvare a problemelor de algoritmică → curs
- Dezvoltarea abilităților de adaptare a unui algoritm la o problemă din viața reală. → laborator/teme/proiect
- Dezvoltarea abilităților de lucru în echipă. → proiect

Obiectivele cursului (II)

- Utilizarea teoriei predate la curs pentru proiectarea algoritmilor de rezolvare pentru probleme tipice întâlnite în practica dezvoltării sistemelor de programe.
- Discutarea relației dintre caracteristicile problemelor, modul de rezolvare și calitatea soluțiilor.
- Compararea variantelor unor algoritmi pentru rezolvarea problemelor dificile.

De ce să învăț PA?

- Exemple de utilizări ale PA-ului în diferite meserii:
 - **web developer** – web social, teoria grafurilor, data mining, clustering;
 - **game dev** – căutări, grafuri, inteligență artificială;
 - **project manager** – fluxuri, grafuri de activități;
 - **dezvoltator de sisteme de operare** – structuri de date avansate, scheme de algoritmi;
 - **programator** – tot ce tine de algoritmi, în special complexitate și eficiență;
 - **tester** – tot ce tine de algoritmi, în special complexitate, eficiență și debugging;



Planul cursului (I)

• Scheme de algoritmi

- Caracteristici ale problemelor și tehnici asociate de rezolvare: divide & impera (Merge Sort, puterea unui număr), rezolvare Iacomă (arbori Hufmann, problema rucsacului continuă), programare dinamică (înmulțirea matricelor, AOC, problema rucsacului discretă). Backtracking cu optimizări. Propagarea restricțiilor.

• Algoritmi pentru jocuri – Minimax și α - β .

• Algoritmi pentru grafuri

- Algoritmi pentru grafuri: parcurgeri, sortare topologică, componente tare conexe, articulații, punți, arbori minimi de acoperire, drumuri de cost minim, fluxuri.

Planul cursului (II)

- **Rezolvarea problemelor prin căutare euristică**
 - Rezolvarea problemelor prin căutarea euristică A*. Completitudine și optimalitate, caracteristici ale heuristicilor.
- **Algoritmi aleatorii**
 - Algoritmi aleatori. Las Vegas și Monte Carlo, aproximare probabilistică.

Evaluarea

- Citiți documentul “Regulament Proiectarea Algoritmilor 2015” de pe site!
<http://ocw.cs.pub.ro/courses/pa/regulament-general>
- Examen 4 p
- Laborator 6 p ☺
 - 3p teme (3 teme punctate egal) // incepând de anul acesta, nu mai există temă de recuperare
 - 2p laborator (1p activitate laborator, 1p test practic)
 - 2p proiect
- Activitate științifică – maxim 0,5 – 1 p bonus (în funcție de realizări)
- Obs. 1 - Nu se copiază în facultate!
- Obs. 2 - Prima temă copiată se punctează cu minus valoarea maximă a temei. La a doua temă copiată, se repetă materia!

Proiect PA

- Tema proiectului: TBA (To Be Announced)
- Obiective:
 - Rezolvarea unei probleme interesante din viața reală;
 - Găsirea și aplicarea unor euristici pentru această problemă;
 - Dezvoltarea abilităților de lucru în echipă;
 - Dezvoltarea spiritului competitiv.

Proiect PA (2)

- **Etape:**
 - 1. Formarea echipelor.
 - 2. Familiarizarea cu frameworkul cu care se va lucra (mișcarea furnicilor pe hartă, acumularea de hrănă și înmulțirea lor).
 - 3. Lupta cu un bot extrem de slab pe diverse planșe.
 - 4. Lupta cu un bot ceva mai intelligent pe diverse planșe.
 - 5. Finalizare proiectului și concursul pentru stabilirea celei mai bune colonii de furnici din an.
- Mai multe informații în “Regulament Proiect PA 2015”

Feedback 2008, 2009

- Idei preluate din feedback 2008:

- Schimbarea modului de organizare al proiectului (etape, mod de lucru în echipă).
- Schimbarea orientării temelor (variante mai ușoare, notare parțială).
- Subliniată importanța bibliografiei.

- Idei preluate din feedback 2009:

- Eliminare restricții echipa proiect la nivel de grupă.
- Publicare teme pe infoarena / checker teme.
- Fără net în laboratoare.
- Laboratoare mai bune, teme mai atent elaborate, organizare mai bună.
- Evitată ora 8 dimineața pentru curs. ☺

Feedback 2010 (1)

- Curs:
 - Prea puține 2 ore. (**nu avem ce face, aşa e în programă**)
 - Evitată ora 8 dimineața pentru curs. (**Am reușit!!!**)
 - Pseudocod uniform - română sau engleză. (**Am trecut totul în română**)
 - Evitarea greșelilor din cursuri. (**îmi cer scuze de pe acum pentru eventualitatea în care mai apar**)
- Laborator:
 - Prea grele. (**am încercat să le ușurăm prin introducerea de schelete de cod**)
 - Furnizare de rezolvări. (**am început anul trecut – sperăm să fie ok anul acesta**)
 - Laboratoare mai clare și mai uniforme din punctul de vedere al scheletului de cod. (**două persoane vor scrie codul de la laboratoare – C++/Java**)
 - Păreri împărtășite referitoare la absența internetului din laborator. (**niciodată nu vom putea să împăcăm pe toată lumea!**)

Feedback 2010 (2)

- Teme:
 - Mai puține (3 în loc de 4) și mai utile. (vor fi 3 teme)
 - Corectate mai repede și uniform pe serii. (fiecare temă va fi corectată de către o singură persoană pe serie)
 - Temă de recuperare peste vară. (va fi disponibilă o astfel de temă)
- Proiect:
 - Schimbarea șahului. (F1- ants)
 - Interesantă competiția finală. (ne bucurăm!, am păstrat-o)
 - E bine că este pe grupe – abilități de lucru în echipă. (asta ne și dorim!)
 - Punctați și cei care nu intră în grupe. (încercăm să schimbăm punctarea – se va face un clasament general deoarece vom încerca să facem meciuri fiecare cu fiecare)
- Examen:
 - Timp prea scurt. (deja am dublat timpul fata de acum doi ani, mai mult de atât nu vi se va pune la dispoziție!)
 - Examen greu!

Feedback 2011 (1)

- Curs:

- Demonstrații prea multe/puține în curs – să se dea mai multe demonstrații la examen! (**părerile sunt împărtite!**)
- Introducere de flashuri pentru a demonstra algoritmii (**anul acesta nu avem cum, dar sperăm să le introducem la anul!** – **Dacă veți găsi astfel de flash-uri sau tool-uri, va rog să mă anunțați și pe mine ca să le pun la bibliografia cursurilor și să le poată folosi și ceilalți colegi de-a voștri**)
- Timp prea puțin la curs (2 ore) fiind necesare 3 ore cel puțin, prea mulți algoritmi (**nu avem ce face, aşa e în programă**)
- Bazat pe cărțile lui Cormen și Giumale (**Așa este! Este necesară citirea capitolelor din aceste cărți! Vezi slide 22!**)

- Laborator

- Introducere de schelete de cod în Java pentru laborator (**sperăm că veți fi mulțumiți de ele**)
- Toți studenții să aibă timp să vadă din timp problemele de laborator (**vor fi disponibile din timp și vă încurajăm să vă uitați peste probleme și peste laborator înainte de a face laboratorul**)

Feedback 2011 (2)

- Teme
 - Teme bine alese, care reflectă probleme reale și sunt în concordanță cu materia predată (vom încerca și anul acesta să dăm teme cel puțin la fel de interesante)
- Proiect
 - Acordare premii pentru câștigătorii concursului de la proiect (încercăm să găsim sponsori pentru o astfel de inițiativă)
 - Erori în serverul de proiect (au fost eliminate astfel de erori întrucât anul acesta am schimbat proiectul și vom folosi o platformă care deja și-a demonstrat valoarea într-un concurs internațional)
- Overall
 - Entuziasm din partea echipei (ne pare bine că a fost remarcat acest aspect! Sperăm să aveți parte de același entuziasm)

Feedback 2012 (1)

- **Curs**

- Complexitatea materiei este cu mult peste numarul alocat in orar (2 ore curs, 2 ore laborator).
- Timpul dedicat cursului a fost prea putin si cateodata se mergea foarte repede, se explica rapid doar pentru a ne incadra in timp. → **trebuie sa acopar materia**
- mult mai mult de 3 ore; → **asa e in programa**
- Daca nu se face de 3 ore sugerez sa se predea mai succint totul, ramanand sa aprofundeze copiii acasa.. ca doar sunt la facultate nu la clasa a patra.
- ar putea fi scoase multe din teoremele si demonstratiile din slide-urile de curs → **nu se poate**
- Putine probleme (aplicatii, exemple de probleme) discutate la curs. → **asta se face la laborator**

Feedback 2012 (2)

- Examen

- Timp mai mult la examene si mai putine subiecte, nu trebuie sa fim presati de timp, in plus nu trebuie ca dupa fiecare subiect sa ni se ia foile. → **ne scuteste de facut politie + veti lucra mereu sub presiunea timpului, nu ar strica sa va obisnuiti**

- Laborator

- scheletul de cod ar fi bine sa fie pus pe site din timp.
- notele de la teme si laborator apar cu intarziere (4-5 saptamani); → **sper ca nu vor mai fi probleme**
- unele exercitii sa fie mai usoare pentru ca 2 ore nu sunt suficiente pentru unii sa rezolve probleme propuse la laborator. → **am transmis asistentilor acest lucru**
- sa nu se mai suprapuna temele si projectul ca deadline cu celelalte teme. → **problema mai complexa**

Feedback 2013

- Cadrul didactic a fost bine pregatit pentru sustinerea?
Cursuri: **4.76 / 6**
Laboratoare:/Seminarii: **4.78 / 6**
- Pot fi obtinute explicatii suplimentare in afara orelor de?
Cursuri: **4.31 / 6**
Laboratoare:/Seminarii: **4.51 / 6**
- Materialele puse la dispozitie sunt suficiente pentru intelegerarea?
Cursuri: **4.33 / 6**
Laboratoare:/Seminarii: **4.42 / 6**
- Numarul si dificultatea temelor au fost adecvate? **3.76 / 6**

Feedback 2013 (2)

- **Aspecte pozitive**

- Materia este frumoasa, importanta, bine predata, utila in multe domenii nu neaparat programare
- Exemple relevante la curs. Explicatii foarte bune atat la curs, cat si la orele de laborator
- Cunostintele invatate au aplicabilitate directa
- Organizarea cursului, sistemul de notare, numarul de teme
- Faptul ca nota maxima este 12 (incluzand tema de recuperare)
- Proiectul este o idee foarte buna, antreneaza si spiritul de echipa
- Echipa tanara, entuziasta, profesorul si asistentii sunt foarte bine pregatiti

Feedback 2013 (3)

- **Aspecte negative**

- Cantitate mare de informatie (numar mare de algoritmi noi) in timp scurt
- Ar fi util o ora in plus atat pentru curs, cat si pentru laborator
- Studentii vor sa ia un punctaj cat mai mare din laboratoare, asa ca fac aceasta prin orice mijloace (corecte si incorecte)
- Laboratoare dificile, dureaza mult intelegerarea scheletului de cod, necesita pregatirea (citirea, intelegerea) laboratorului de acasa de multe ori
- Temele un pic cam dificile si corectate tarziu
- Deadline-uri suprapuse cu alte materii
- Proiectul organizat cam prost (inclusiv probleme cu platforma aleasa)

Feedback 2014 (1)

- | • Cadrul didactic a fost bine pregatit pentru sustinerea?
 - | Cursuri: 4.65 / 6
 - | Laboratoare:/Seminarii: 4.86 / 6
- | • Pot fi obtinute explicatii suplimentare in afara orelor de?
 - | Cursuri: 4.13 / 6
 - | Laboratoare:/Seminarii: 4.48 / 6
- | • Materialele puse la dispozitie sunt suficiente pentru intelegerarea?
 - | Cursuri: 4.03 / 6
 - | Laboratoare:/Seminarii: 4.31 / 6
- | • Numarul si dificultatea temelor au fost adecvate? 3.42 / 6

Feedback 2014 (2)

- Aspecte:
 - Multe observații similare cu cele din 2013, referitor la dificultate, număr mare de teme, etc. → Este o materie dificilă, nu o putem face mai ușoară
 - Studenții au reclamat ca laboratorul s-a transformat într-o alergătură după puncte, sunt prea dificile, iar studenții nu sunt atenți la explicații, au tendința să mai copieze de la colegi, etc.
- Drept urmare:
 - Laboratorul valorează doar 1p în acest an
 - A fost introdus un test practic care va fi susținut aproximativ în săptămâna a 10a → valorează 1p

Bibliografie

- Introducere in Analiza Algoritmilor de *Cristian Giumeș* – Ed. Polirom 2004
- Introducere in Algoritmi de *Thomas H. Cormen, Charles E. Leiserson, Ronald R. Rivest* – Ed. Agora
- **Vedeți și recomandările din Regulament!**
- **Mențiune importantă – slide-urile reprezintă doar un suport pentru prezentare!**

Proiectarea Algoritmilor

Curs 1 – Scheme de algoritmi –
Divide et impera

Curs 1 – Cuprins

- Scheme de algoritmi
- Divide et impera
- Exemplificare folosind Merge sort
- Alte exemple de algoritmi divide et impera
- Greedy
- Exemplificare folosind arbori Huffman
- Demonstrația corectitudinii algoritmului Huffman

Curs 1 – Bibliografie

- Giumale – Introducere in Analiza Algoritmilor cap 4.4
- Cormen – Introducere în Algoritmi cap. Algoritmi Greedy (17)
- <http://www.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf>
- <http://www.cs.umass.edu/~barring/cs611/lecture/4.pdf>
- <http://thor.info.uaic.ro/~dlucanu/cursuri/tpaa/resurse/Curs6.pps>
- <http://www.math.fau.edu/locke/Greedy.htm>
- http://www.cs.rit.edu/~ib/Classes/CS515_Spring12-13/Slides/022-SelectMasterThm.pdf

Scheme de algoritmi

- Prin **scheme de algoritmi** înțelegem **tipare comune** pe care le putem aplica în rezolvarea unor **probleme similare**.
- O gamă largă de probleme se pot rezolva folosind un număr relativ mic de scheme.
- => Cunoașterea schemelor determină o rezolvare mai rapidă și mai eficientă a problemelor.

Divide et Impera (I)

- Ideea (divide și cucerește) este atribuită lui Filip al II-lea, regele Macedoniei (382-336 i.e.n.), tatăl lui Alexandru cel Mare și se referă la politica acestuia față de statele grecești.
- În CS – **Divide et impera** se referă la o clasă de algoritmi care au ca **principale caracteristici** faptul că **împart problema în subprobleme similare cu problema inițială** dar mai mici ca dimensiune, **rezolvă problemele recursiv** și apoi **combină soluțiile** pentru a crea o soluție pentru problema originală.

Divide et Impera (II)

- Schema **Divide et impera** constă în **3 pași** la fiecare nivel al recurenței:
 - **Divide** problema dată într-un număr de subprobleme;
 - **Impera (cucerește)** – subproblemele sunt rezolvate recursiv. Dacă subproblemele sunt suficient de mici ca date de intrare se rezolvă direct (**ieșirea din recurență**);
 - **Combină** – soluțiile subproblemelor sunt combinate pentru a obține soluția problemei inițiale.

Divide et Impera – Avantaje și Dezavantaje

- **Avantaje:**

- Produce **algoritmi eficienți**.
- Descompunerea problemei în subprobleme facilitează **paralelizarea algoritmului** în vederea execuției sale pe mai multe procesoare.

- **Dezavantaje:**

- Se adaugă un **overhead datorat recursivității** (reținerea pe stivă a apelurilor funcțiilor).

Merge Sort (I)

- Algoritmul **Merge Sort** este un exemplu clasic de rezolvare cu D&I.
- **Divide**: Divide cele n elemente ce trebuie sortate în 2 secvențe de lungime $n/2$.
- **Impera**: Sortează secvențele recursiv folosind *merge sort*.
- **Combină**: Secvențele sortate sunt asamblate pentru a obține vectorul sortat.
- Recurența se oprește când secvența ce trebuie sortată are lungimea 1 (un vector cu un singur element este întotdeauna sortat ☺) .
- Operația cheie este: **asamblarea soluțiilor parțiale**.

Merge Sort (II)

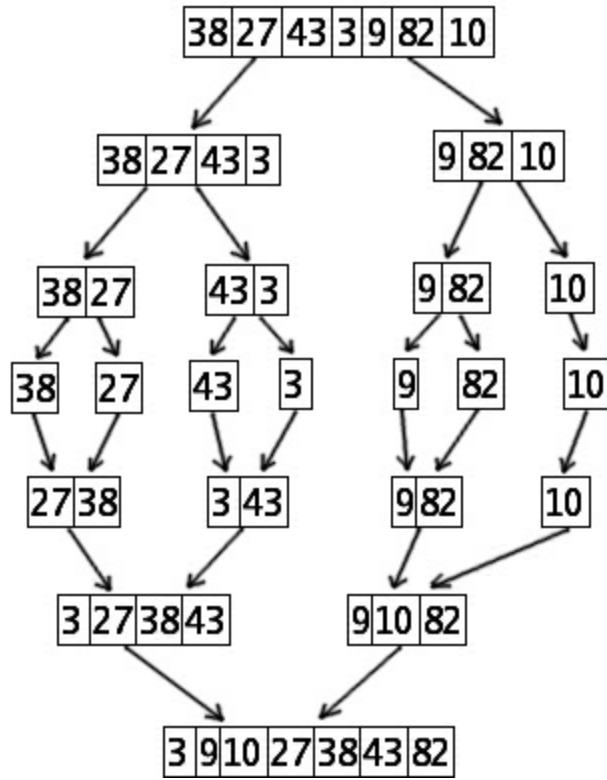
Merge Sort (III) – Algoritmul de interclasare

- Algoritm [Cormen]

- MERGE(A, p, q, r) // p și r sunt capetele intervalului, q este “mijlocul”
 - 1 $n_1 \leftarrow q - p + 1$ // numărul de elemente din partea stânga
 - 2 $n_2 \leftarrow r - q$ // numărul de elemente din partea dreapta
 - 3 creează vectorii $S[1 \rightarrow n_1 + 1]$ și $D[1 \rightarrow n_2 + 1]$
 - 4 Pentru i de la 1 la n_1
 - 5 $S[i] \leftarrow A[p + i - 1]$ // se copiază partea stânga în S
 - 6 Pentru j de la 1 la n_2
 - 7 $D[j] \leftarrow A[q + j]$ // si partea dreapta în D
 - 8 $S[n_1 + 1] \leftarrow \infty$
 - 9 $D[n_2 + 1] \leftarrow \infty$
 - 10 $i \leftarrow 1$
 - 11 $j \leftarrow 1$
 - 12 Pentru k de la p la r // se copiază înapoi în vectorul de
 - 13 Dacă $S[i] \leq D[j]$ // sortat elementul mai mic din cei
 - 14 Atunci $A[k] \leftarrow S[i]$ // doi vectori sortați deja
 - 15 $i \leftarrow i + 1$
 - 16 Altfel $A[k] \leftarrow D[j]$
 - 17 $j \leftarrow j + 1$

Exemplu funcționare Merge Sort

- Exemplu funcționare [Wikipedia]:



Merge Sort - Complexitate

- $T(n) = 2 * T(n/2) + \Theta(n)$

număr de subprobleme

dimensiunea subproblemelor

complexitatea interclasării

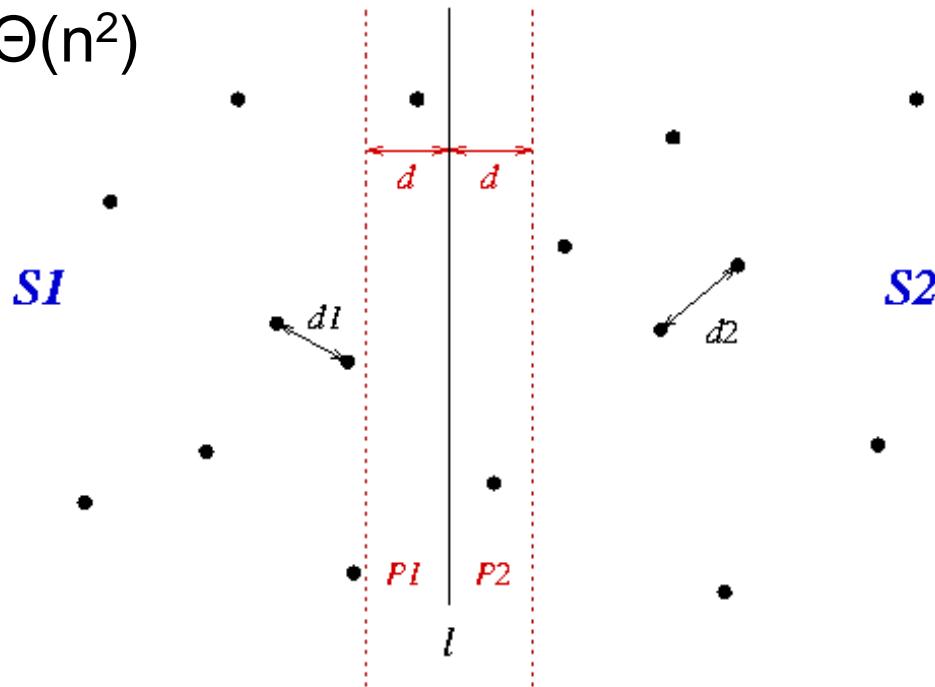
$$\Rightarrow (\text{din T. Master}) T(n) = \Theta(n * \log n)$$

Divide et Impera – alte exemple (I)

- Calculul puterii unui număr: x^n
 - Algoritm “clasic”:
 - Pentru i de la 1 la n rez = rez * x;
 - Întoarce rez.
 - Complexitate: $\Theta(n)$
- Algoritm Divide et Impera:
 - Dacă n este par
 - Atunci Întoarce $x^{n/2} * x^{n/2}$
 - Altfel (n este impar) Întoarce $x * x^{(n-1)/2} * x^{(n-1)/2}$
 - Complexitate: $T(n) = T(n/2) + \Theta(1) = \Theta(\log n)$

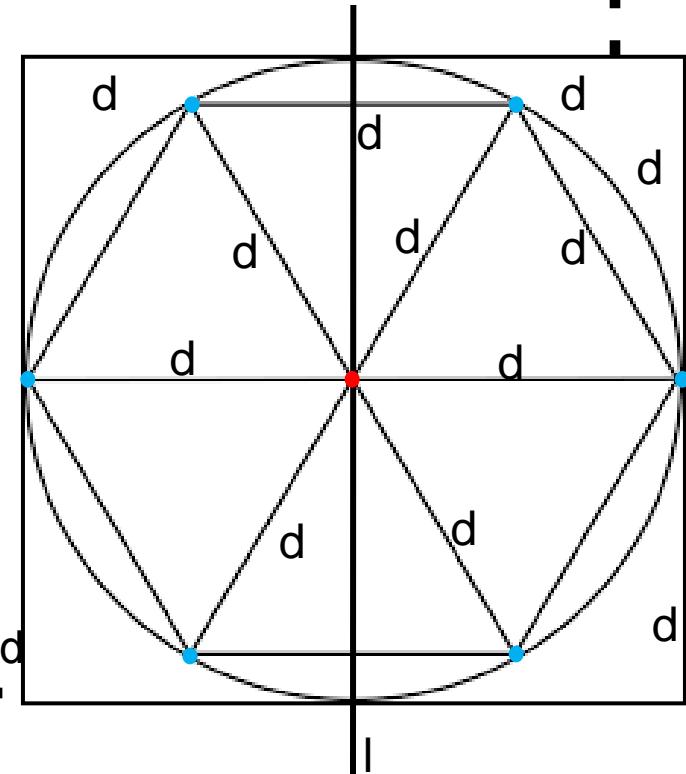
Divide et Impera – alte exemple (II)

- Calculul celei mai scurte distanțe între 2 puncte din plan (<http://www.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf>)
 - algoritmul naiv – $\Theta(n^2)$



Divide et Impera – alte exemple (III)

- Sortează punctele în ordinea crescătoare a coordonatei x ($\Theta(n \log n)$);
- Împărțim setul de puncte în 2 seturi de dimensiune egală și calculăm recursiv distanța minimă în fiecare set (l = linia ce împarte cele 2 seturi, d = distanța minimă calculată în cele 2 seturi);
- Elimină punctele care sunt plasate la distanță de $l > d$;
- Sortează punctele rămase după coordonata y;
- Calculează distanțele de la fiecare punct rămas la cei 6 vecini (nu pot fi mai mulți);
- Dacă găsește o distanță $< d$, atunci actualizează d .



Divide et Impera – Temă de gândire (1)

- Se dă o mulțime M de numere întregi și un număr x . Se cere să se determine dacă există $a, b \in M$ a.î. $a + b = x$.
- Algoritmul propus trebuie să aibă complexitatea $\Theta(n \log n)$.
- Temele de la curs sunt **facultative!** 😊

Divide et Impera – Temă de gândire (2)

- Cum sortăm un vector fără interclasare sau alte operații complexe în etapele de împărțire a problemei și de combinare a soluțiilor?
- Hint: se folosesc trei apeluri recursive!
- Observație! Algoritmul este inefficient, dar este interesant!

Divide et Impera – Temă de gândire (3)

- Cum se găseste eficient elementul median al unui vector?
- Problema mai generală: cum se găsește al k-lea cel mai mic (sau cel mai mare) element dintr-un vector?
- Eficient înseamnă $\Theta(n)$
- http://www.cs.rit.edu/~ib/Classes/CS515_Spring12-13/Slides/022-SelectMasterThm.pdf

Divide et Impera – Temă de gândire

- Se dă o mulțime M de numere întregi și un număr x . Se cere să se determine dacă există $a, b \in M$ a.î. $a + b = x$.
- Algoritmul propus trebuie să aibă complexitatea $\Theta(n \log n)$.
- Temele de la curs sunt **facultative!** 😊

ÎNTREBĂRI?

Proiectarea Algoritmilor

Curs 2 – Scheme de algoritmi –
Greedy

Bibliografie

- Cormen – Introducere în Algoritmi cap. Algoritmi Greedy (17) și Programare dinamică (16)
- Giumale – Introducere în Analiza Algoritmilor cap 4.4 ,4.5
- <http://www.cs.umass.edu/~barring/cs611/lecture/4.pdf>
- <http://thor.info.uaic.ro/~dlucanu/cursuri/tpaa/resurse/Curs6.pps>
- <http://www.math.fau.edu/locke/Greedy.htm>
- <http://en.wikipedia.org/wiki/Greedoid>
- <http://www.cse.ust.hk/~dekai/271/notes/L12/L12.pdf>

Greedy (I)

- Metodă de rezolvare eficientă a unor probleme de optimizare.
- Soluția trebuie să satisfacă un criteriu de optim global (greu de verificat) → optim local mai ușor de verificat.
- Se aleg soluții parțiale ce sunt îmbunătățite repetat pe baza criteriului de optim local până ce se obțin soluții finale.
- Soluțiile parțiale ce nu pot fi îmbunătățite sunt abandonate → proces de rezolvare irevocabil (fără reveniri)!

Greedy (II)

- Schema generală de rezolvare a unei probleme folosind Greedy (programarea lacomă):
- Rezolvare_lacomă(Crit_optim, Problemă)
 - 1. sol_partiale = sol_initiale(Problemă); // determinarea soluțiilor parțiale
 - 2. sol_fin = \emptyset ;
 - 3. **Cât timp** ($\text{sol_partiale} \neq \emptyset$)
 - 4. **Pentru fiecare** (s în sol_partiale)
 - 5. **Dacă** (s este o soluție a problemei) { // dacă e soluție
 - 6. $\text{sol_fin} = \text{sol_fin} \cup \{s\}$; // finală se salvează
 - 7. $\text{sol_partiale} = \text{sol_partiale} \setminus \{s\}$;
 - 8. } **Altfel** // se poate optimiza?
 - 9. **Dacă** (optimizare_posibilă (s , Crit_optim, Problemă))
 - 10. $\text{sol_partiale} = \text{sol_partiale} \setminus \{s\} \cup$ // da optimizare(s , Crit_optim, Problemă)
 - 11. **Altfel** $\text{sol_partiale} = \text{sol_partiale} \setminus \{s\}$; // nu
 - 12. **Întoarce** sol_fin ;

Comparație D&I și Greedy

- Tip abordare
 - D&I: top-down;
 - Greedy: top-down.
- Criteriu de optim
 - D&I: nu;
 - Greedy: da.

Arborei Huffman

- Metodă de codificare folosită la compresia fișierelor.
- Construcția unui astfel de arbore se realizează printr-un algoritm Greedy.
- Considerăm un text, de exemplu:
 - **ana are mere**
- Vom exemplifica pas cu pas construcția arborelui de codificare pentru acest text și vom defini pe parcurs conceptele de care avem nevoie.

Arborei Huffman – Definiții (I)

- K – multimea de simboluri ce vor fi codificate. ($a, n, " ", r, e, m$)
- Arbore de codificare a cheilor K este un arbore binar ordonat cu proprietăți:
 - Doar frunzele conțin cheile din K ; nu există mai mult de o cheie într-o frunză;
 - Toate nodurile interne au exact 2 copii;
 - Arcele sunt codificate cu 0 și 1 (arcul din stânga unui nod – codificat cu 0).
- $k = \text{Codul unei chei}$ – este sirul etichetelor de pe calea de la rădăcina arborelui la frunza care conține cheia k (k este din K).
- $p(k)$ – frecvența de apariție a cheii k în textul ce trebuie comprimat.
- Ex pentru “ana are mere”:
 - $p(a) = p(e) = 0.25; p(n) = p(m) = 0.083; p(r) = p(" ") = 0.166$

Arborei Huffman – Definiții (II)

- A – arborele de codificare a cheilor.
- lg_cod(k) – lungimea codului cheii k conform A.
- nivel(k,A) – nivelul pe care apare în A frunza ce conține cheia K.
- Costul unui arbore de codificare A al unor chei K relativ la o frecvență p este:

$$Cost(A) = \sum_{k \in K} lg_cod(k) * p(k) = \sum_{k \in K} nivel(k, A) * p(k)$$

- Un arbore de codificare cu cost minim al unor chei K, relativ la o frecvență p este un arbore Huffman, iar codurile cheilor sunt coduri Huffman.

Arborei Huffman – algoritm de construcție (I)

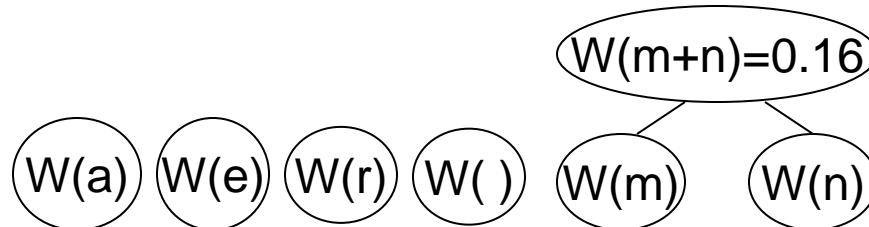
- 1. Pentru fiecare k din K se construiește un **arbore cu un singur nod** care conține cheia k și este caracterizat de **ponderea** $w = p(k)$. Subarborii construiți formează o mulțime numită Arb.
- 2. Se aleg doi subarbori a și b din Arb astfel încât **a și b au pondere minimă**.

Arborei Huffman – algoritm de construcție (II)

- 3. Se construiește un **arbore binar** cu o rădăcina r care nu conține nici o cheie și cu **descendenții** a și b. **Ponderea arborelui** este definită ca $w(r) = w(a) + w(b)$.
- 4. **Arborii a și b sunt eliminați** din Arb iar r este inserat în Arb.
- 5. **Se repetă procesul** de construcție descris de pașii 2-4 până când **mulțimea Arb conține un singur arbore** – **Arborele Huffman** pentru cheile K.

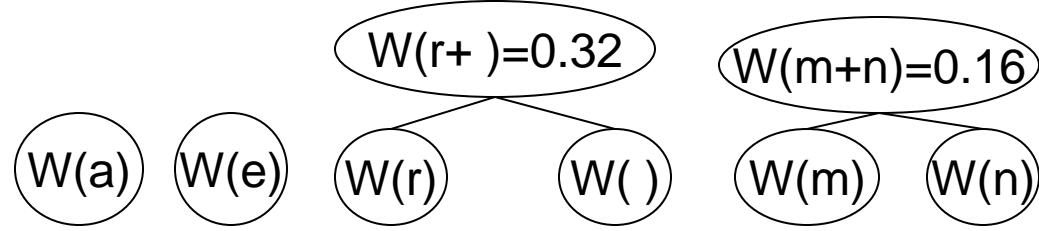
Arbore Huffman – Exemplu (I)

- Text: ana are mere
- $p(a) = p(e) = 0.25; p(n) = p(m) = 0.083; p(r) = p() = 0.166$
- Pasul 1:
 - $W(a)=0.25$
 - $W(e)=0.25$
 - $W(r)=0.16$
 - $W()=0.16$
 - $W(m)=0.08$
 - $W(n)=0.08$
- Pasii 2-4:

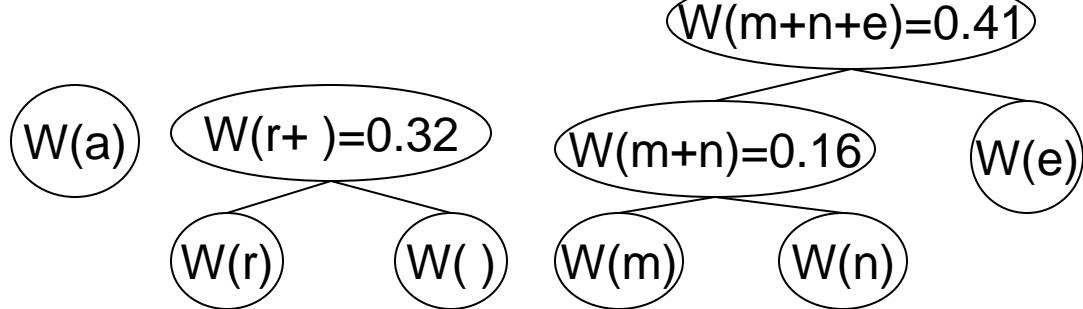


Arbore Huffman – Exemplu (II)

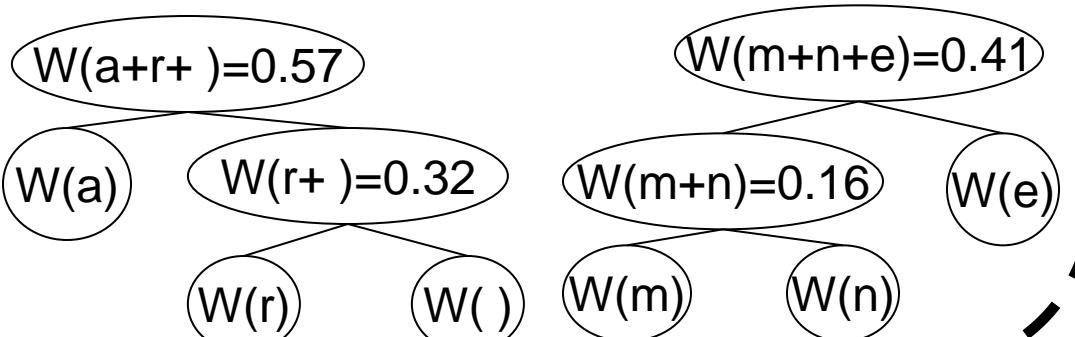
- Pasii 2-4 (II):



- Pasii 2-4 (III):

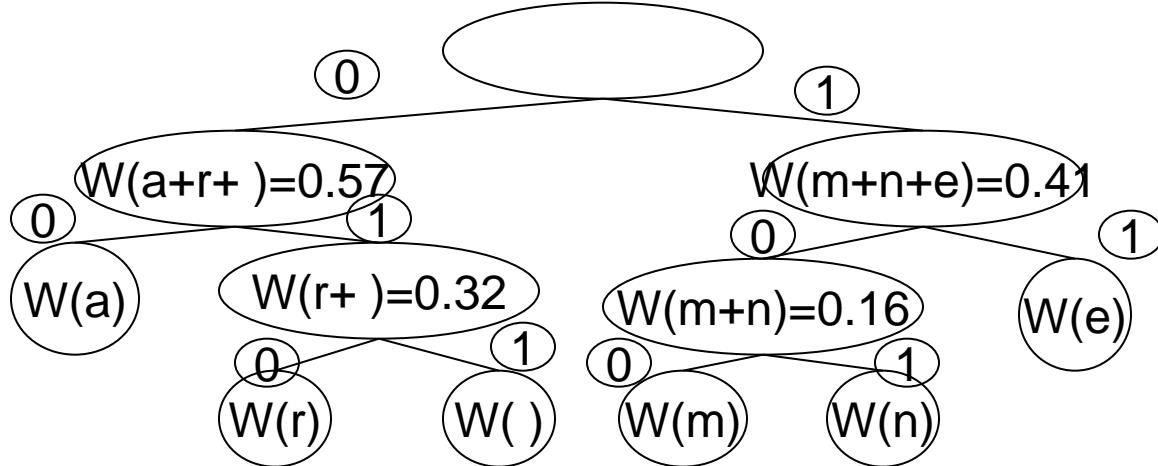


- Pasii 2-4 (IV):



Arborei Huffman – Exemplu (III)

- Pasii 2-4 (V):



- Codificare: a - 00; e - 11; r - 010; ' ' - 011; m - 100; n - 101;

$$Cost(A) = \sum_{k \in K} \lg_cod(k) * p(k) = \sum_{k \in K} nivel(k, A) * p(k)$$

- $p(a) = p(e) = 0.25$; $p(n) = p(m) = 0.083$; $p(r) = p() = 0.166$
- $Cost(A) = 2 * 0.25 + 2 * 0.25 + 3 * 0.083 + 3 * 0.083 + 3 * 0.166 + 3 * 0.166 = 1 + 1.2 = 2.2$ biti.

Arborei Huffman - pseudocod

- Huffman(K, p){
 - 1. $\text{Arb} = \{k \in K \mid \text{frunză}(k, p(k))\};$
 - 2. **Cât timp** ($\text{card}(\text{Arb}) > 1$) // am mai mulți subarbore
 - 3. Fie a_1 și a_2 arbori din Arb a.i. $\forall a \in \text{Arb} a \neq a_1$ și $a \neq a_2$, avem
 $w(a_1) \leq w(a)$ și $w(a_2) \leq w(a)$; // practic se extrage
// de două ori minimul și se salvează în a_1 și a_2
 - 4. $\text{Arb} = \text{Arb} \setminus \{a_1, a_2\} \cup \text{nod_intern}(a_1, a_2, w(a_1) + w(a_2));$
 - 5. **Dacă** ($\text{Arb} = \emptyset$)
 - 6. Întoarce arb_vid ;
 - 6. **Altfel**
 - 7. fie A singurul arbore din multimea Arb ;
 - 8. Întoarce A ;
- Notații folosite:
 - $a = \text{frunză}(k, p(k))$ – subarbore cu un singur nod care conține cheia k , iar $w(a) = p(k)$;
 - $a = \text{nod_intern}(a_1, a_2, x)$ – subarbore format dintr-un nod intern cu descendenții a_1 și a_2 și $w(a) = x$.

Arborei Huffman - Decodificare

- Se încarcă arborele și se decodifică textul din fisier conform algoritmului:
- Decodificare (in, out)
A = restaurare_arbore (in) // reconstruiesc arborele
Cât timp (! terminare_cod(in)) // mai am caractere de citit
 nod = A // pornesc din rădăcină
 Cât timp (! frunză(nod)) // cât timp nu am determinat caracterul
 Dacă (bit(in) = 1) nod = dreapta(nod) // avansez în arbore
 Altfel nod = stânga(nod)
 Scrie (out, cheie(nod)) // am determinat caracterul și îl scriu la // ieșire

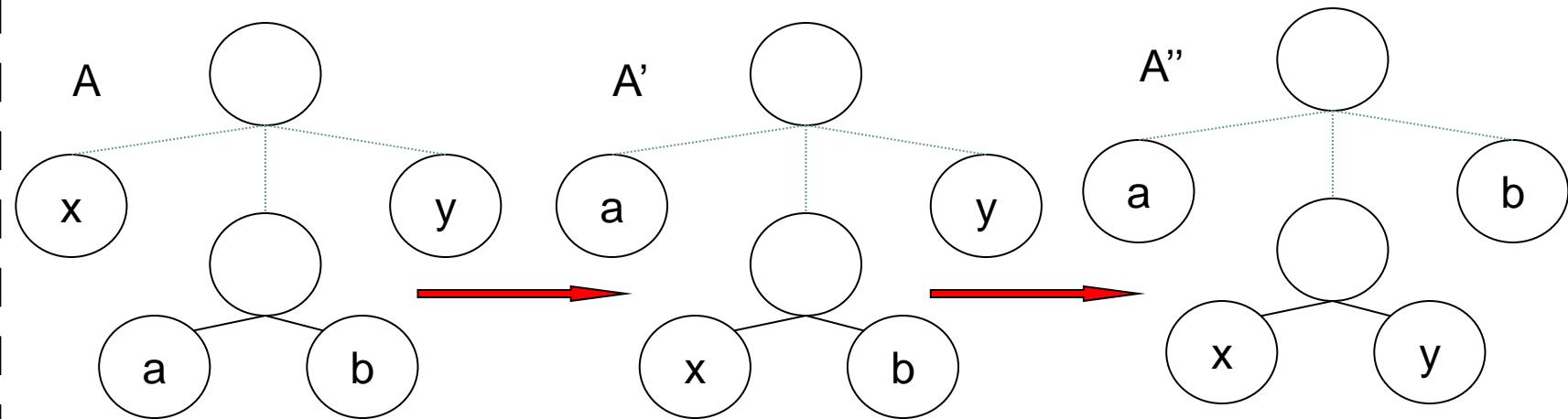
Demonstrație (I)

- Arboarele de codificare construit trebuie să aibă cost minim pentru a fi arbore Huffman.
- **Lema 1.** Fie K mulțimea cheilor dintr-un arbore de codificare, $\text{card}(K) \geq 2$, x, y două chei cu pondere minimă. \exists un arbore Huffman de înălțime h în care cheile x și y apar pe nivelul h fiind descendente ale aceluiași nod intern.

Demonstrație (II)

- Demonstrație Lema 1:

$$Cost(A) = \sum_{k \in K} \lg_cod(k) * p(k) = \sum_{k \in K} nivel(k, A) * p(k)$$



- Se interschimbă a cu x și b cu y și din definiția costului arborelui $\Rightarrow \text{cost}(A'') \leq \text{cost}(A') \leq \text{cost}(A)$
 \Rightarrow A'' arbore Huffman

Demonstrație (III)

- Lema 2. Fie A un arbore Huffman cu cheile K , iar x și y două chei direct descendente ale aceluiași nod intern a . Fie $K' = K \setminus \{x, y\} \cup \{z\}$ unde z este o cheie fictivă cu ponderea $w(z) = w(x) + w(y)$. Atunci arborele A' rezultat din A prin înlocuirea subarborelui cu rădăcina a și frunzele x, y cu subarborele cu un singur nod care conține frunza z , este un arbore Huffman cu cheile K' .
- Demonstrație:
 - 1) analog $\text{Cost}(A') \leq \text{Cost}(A)$ ($\text{Cost}(A) = \text{Cost}(A') + w(x) + w(y)$)
 - 2) pp există A'' a.i. $\text{Cost}(A'') < \text{Cost}(A') \Rightarrow$
 - $\text{Cost}(A'') < \text{Cost}(A) - (w(x) + w(y))$;
 - $\text{Cost}(A'') + w(x) + w(y) < \text{Cost}(A)$; $\Rightarrow A$ nu este Huffman (contradicție)

Demonstrație (IV)

- Teoremă – Algoritmul Huffman construiește un arbore Huffman.
- Demonstrație: prin inducție după numărul de chei din multimea K.
- $n \leq 2 \Rightarrow$ evident
- $n > 2$
 - Ip. Inductivă: algoritmul Huffman construiește arbori Huffman pentru orice multime cu $n-1$ chei
 - Fie $K = \{k_1, k_2, \dots, k_n\}$ a.i. $w(k_1) \leq w(k_2) \leq \dots \leq w(k_n)$

Demonstrație (V)

- Cf. [Lema 1](#), \exists Un arbore Huffman unde cheile k_1, k_2 sunt pe același nivel și descendente ale aceluiași nod.
- A_{n-1} – arborele cu $n-1$ chei $K' = K - \{k_1, k_2\} \cup z$ unde $w(z) = w(k_1) + w(k_2)$.
- A_{n-1} rezultă din A_n prin modificările prezentate în [Lema 2](#) $\Rightarrow A_{n-1}$ este Huffman, și cf. ipotezei inducitive e construit prin algoritmul $Huffman(K', p')$.
- \Rightarrow Algoritmul $Huffman(K, p)$ construiește arborele format din k_1 și k_2 și apoi lucrează ca și algoritmul $Huffman(K', p')$ ce construiește A_{n-1} \Rightarrow construiește arborele $Huffman(K, p)$.

Alt exemplu (I)

Problema rucsacului

Trebuie să umplem un rucsac de capacitate maximă M kg cu obiecte care au greutatea m_i și valoarea v_i . Putem alege mai multe obiecte din fiecare tip cu scopul de a maximiza valoarea obiectelor din rucsac.

- **Varianta 1:** putem alege fracții de obiect – “problema continuă”
- **Varianta 2:** nu putem alege decât obiecte întregi (număr natural de obiecte din fiecare tip) – “problema 0-1”

Alt exemplu (II)

- **Varianta 1: Algoritm Greedy**
 - sortăm obiectele după raportul v_i/m_i ;
 - adăugăm fracțiuni din obiectul cu cea mai mare valoare per kg până epuizăm stocul și apoi adăugăm fracțiuni din obiectul cu valoarea următoare.
 - **Exemplu:** $M = 10$; $m_1 = 5 \text{ kg}$, $v_1 = 10$, $m_2 = 8 \text{ kg}$, $v_2 = 19$, $m_3 = 4 \text{ kg}$, $v_3 = 4$
 - **Soluție:** (m_2, v_2) 8kg și 2kg din (m_1, v_1) – valoarea totală: $19 + 2 * 10 / 5 = 23$
- **Varianta 2: Algoritmul Greedy nu funcționează => contraexemplu**
 - **Exemplu:** $M = 10$; $m_1 = 5 \text{ kg}$, $v_1 = 10$, $m_2 = 8 \text{ kg}$, $v_2 = 19$, $m_3 = 4 \text{ kg}$, $v_3 = 4$
 - **Rezultat corect** – 2 obiecte (m_1, v_1) – valoarea totală: **20**
 - **Rezultat algoritm Greedy** – 1 obiect (m_2, v_2) – valoarea totală: **19**

Când funcționează algoritmii Greedy? (I)

- Problema are proprietatea de substructură optimă
 - Soluția problemei conține soluțiile subproblemelor.
- Problema are proprietatea alegerii locale
 - Alegând soluția optimă local se ajunge la soluția optimă global.

Când funcționează algoritmii Greedy? (II)

- Fie E o mulțime finită nevidă și $I \subset \mathcal{P}(E)$ a.i. $\emptyset \in I$, $\forall X \subseteq Y$ și $Y \subseteq I \Rightarrow X \subseteq I$.
Atunci spunem că (E, I) este un **sistem accesibil**.
- Submulțimile din I sunt numite submulțimi “**independente**”.
- **Exemple:**
 - Ex1: $E = \{e_1, e_2, e_3\}$ și $I = \{\emptyset, \{e_1\}, \{e_2\}, \{e_3\}, \{e_1, e_2\}, \{e_2, e_3\}\}$ – mulțimile ce nu conțin e_1 și e_3 .
 - Ex2: E – muchiile unui graf neorientat și I mulțimea mulțimilor de muchii ce nu conțin un ciclu (mulțimea arborilor).
 - Ex3: E set de vectori dintr-un spațiu vectorial, I mulțimea mulțimilor de vectori linear independenți.
 - Ex4: E – muchiile unui graf neorientat și I mulțimea mulțimilor de muchii în care oricare 2 muchii nu au un vârf comun.

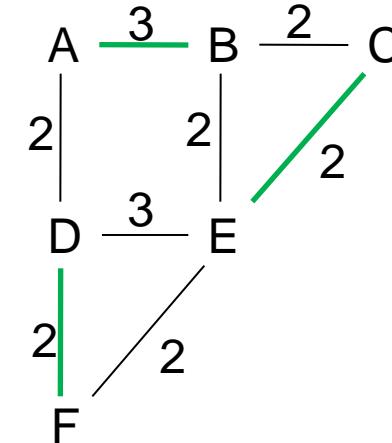
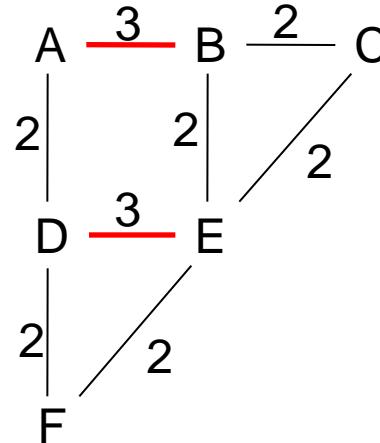
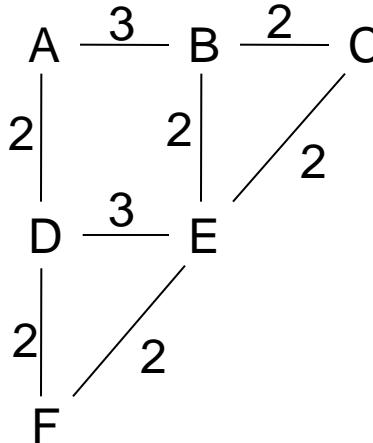
Când funcționează algoritmii Greedy? (III)

- Un **sistem accesibil** este un **matroid** dacă satisface proprietatea de **interschimbare**:
 $X, Y \subseteq I$ și $|X| < |Y| \Rightarrow \exists e \in Y \setminus X$ a.i. $X \cup \{e\} \subseteq I$
- **Teoremă.** Pentru orice **subset accesibil** (E, I) **algoritmul Greedy rezolvă problema de optimizare** dacă și numai dacă (E, I) este **matroid**.

Verificăm exemplele

- Ex1: $I = \{\emptyset, \{e_1\}, \{e_2\}, \{e_3\}, \{e_1, e_2\}, \{e_2, e_3\}\}$
Fie $Y = \{\{e_1\}, \{e_2\}, \{e_3\}, \{e_1, e_2\}\}$ și $X = \{\{e_1\}, \{e_3\}\}$
 $\rightarrow Y \setminus X = \{\{e_2\}, \{e_1, e_2\}\} \rightarrow X \cup \{e_2\} \subseteq I \rightarrow$ matroid

- Ex4:



Problema rucsacului discretă (varianta 2)

- **Exemplu:** $M = 10$; $m_1 = 5 \text{ kg}$, $v_1 = 10$, $m_2 = 8 \text{ kg}$, $v_2 = 19$, $m_3 = 4 \text{ kg}$, $v_3 = 4$
- Fie $I = \text{mulțimea subsoluțiilor}$ și $X = \{m_2\}$ și $Y = \{m_1, m_1\} \rightarrow m_1 \in Y \setminus X$ dar $X \cup \{m_1\} \notin I$
 \rightarrow problema **nu respectă proprietatea de matroid** \rightarrow problema **nu se poate rezolva folosind tehnica Greedy!**

Greedy – tema de gândire

- Se dă un număr natural n . Să se găsească cel mai mare subset S din $\{1, 2, \dots, n\}$ astfel încât nici un element din S să nu fie divizibil cu nici un alt element din S .
- În plus, dacă există mai multe subseturi maximale ce respectă proprietatea de mai sus, să se determine întotdeauna cel mai mic din punct de vedere lexicografic.
- S este **lexicografic mai mic decât T** dacă cel mai mic element care este membru din S sau T , dar nu din amândouă, face parte din S .

ÎNTREBĂRI?

Proiectarea Algoritmilor

Curs 3 – Programare dinamică

Programare dinamică

- Programare dinamică
 - Descriere generală
 - Algoritm generic
 - Caracteristici
- Exemplificare: Înmulțirea matricilor
- Exemplificare: Arboi optimi la căutare (AOC)
 - Definiții
 - Construcția AOC

Programare dinamică

- Descriere generală
 - Soluții optime construite iterativ asamblând soluții optime ale unor probleme similare de dimensiuni mai mici.
- Algoritmi “clasici”
 - Înmulțirea unui sir de matrici
 - AOC
 - Algoritmul Floyd-Warshall care determină drumurile de cost minim dintre toate perechile de noduri ale unui graf.
 - Numere catalane
 - Viterbi

Algoritm generic

- Programare dinamică (crit_optim, problema)
 - // fie problema₀ problema₁ ... problema_n astfel încât
 - // problema_n = problema; problema_i mai simplă decât problema_{i+1}
 - 1. Sol = soluții_iniciale(crit_optim, problema₀);
 - 2. **Pentru i de la 1 la n // construcție soluții pentru**
 // problema_i folosind soluțiile problemelor precedente
 - 3. Sol_i = calcul_soluții(Sol, Crit_optim, Problema_i);
 // determin soluția problemei_i
 - 4. Sol = Sol U Sol_i;
 // noile soluții se adaugă pentru a fi refolosite pe viitor
 - 5. s = soluție_pentru_problema_n(Sol);
 // selecție / construcție soluție finală
 - 6. **Întoarce s;**

Caracteristici

- O soluție optimă a unei probleme conține soluții optime ale subproblemelor.
- Decompozabilitatea recursivă a problemei P în subprobleme similare $P = P_n, P_{n-1}, \dots, P_0$ care acceptă soluții din ce în ce mai simple.
- Suprapunerea problemelor (soluția unei probleme P_i participă în procesul de construcție a soluțiilor mai multor probleme P_k de talie mai mare $k > i$) – memoizare (se folosește un tablou pentru salvarea soluțiilor subproblemelor cu scopul de a nu le recalcula).
- În general se folosește o abordare bottom-up, de la subprobleme la probleme.

Diferențe Greedy – Programare dinamică

Programare Iacomă

- Sunt menținute doar soluțiile parțiale curente din care evoluează soluțiile parțiale următoare
- Soluțiile parțiale anterioare sunt eliminate
- Se poate obține o soluție neoptimă. (trebuie demonstrat că se poate aplica).

Programare dinamică

- La construcția unei soluții noi poate contribui orice altă soluție parțială generată anterior
- Se păstrează toate soluțiile parțiale
- Se obține soluția optimă.

Diferențe divide et impera – programare dinamică

Divide et impera

- abordare top-down – problema este descompusă în subprobleme care sunt rezolvate independent
- putem rezolva aceeași problemă de mai multe ori (dezavantaj potențial foarte mare)

Programare dinamică

- abordare bottom-up - se pornește de la sub-soluții elementare și se combină sub-soluțiile mai simple în sub-soluții mai complicate, pe baza criteriului de optim
- se evită calculul repetat al aceleiași subprobleme prin memorarea rezultatelor intermediare (memoizare)

Exemplu: Parantezarea matricilor (Chain Matrix Multiplication)

- Se dă un sir de matrice: A_1, A_2, \dots, A_n .
- Care este **numărul minim de înmulțiri** de scalari pentru a calcula produsul:
$$A_1 \times A_2 \times \dots \times A_n ?$$
- Să se determine una dintre **parantezările care minimizează** numărul de înmulțiri de scalari.

Înmulțirea matricilor

- $A(p, q) \times B (q, r) \Rightarrow pqr$ înmulțiri de scalari.
- Dar înmulțirea matricilor este **asociativă** (deși **nu este comutativă**).
- $A(p, q) \times B (q, r) \times C(r, s)$
 $(AB)C \Rightarrow pqr + prs$ înmulțiri
 $A(BC) \Rightarrow qrs + pqs$ înmulțiri
- Ex: $p = 5, q = 4, r = 6, s = 2$
 $(AB)C \Rightarrow 180$ înmulțiri
 $A(BC) \Rightarrow 88$ înmulțiri
- **Concluzie:** Parantezarea este foarte importantă!

Soluția banală

- Matrici: A_1, A_2, \dots, A_n .
- Vector de dimensiuni: $p_0, p_1, p_2, \dots, p_n$.
- $A_i(p_{i-1}, p_i) \rightarrow A_1(p_0, p_1), A_2(p_1, p_2), \dots$
- Dacă folosim căutare exhaustivă și vrem să construim toate parantezările posibile pentru a determina minimul: $\Omega(4^n / n^{3/2})$.
- Vrem o **soluție polinomială** folosind P.D.

Descompunere în subprobleme

- Încercăm să definim subprobleme identice cu problema originală, dar de dimensiune mai mică.
- $\forall 1 \leq i \leq j \leq n$:
 - Notăm $A_{i,j} = A_i \times \dots \times A_j$. $A_{i,j}$ are p_{i-1} linii și p_j coloane: $A_{i,j}(p_{i-1}, p_j)$
 - $m[i, j]$ = numărul optim de înmulțiri pentru a rezolva subproblema $A_{i,j}$
 - $s[i, j]$ = poziția primei paranteze pentru subproblema $A_{i,j}$
 - **Care e parantezarea optimă pentru $A_{i,j}$?**
- Problema inițială: $A_{1,n}$

Combinarea subproblemelor

- Pentru a rezolva $A_{i,j}$
 - Trebuie găsit acel indice $i \leq k < j$ care asigură parantezarea optimă:

$$A_{i,j} = (A_i \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_j)$$

$$A_{i,j} = A_{i,k} \times A_{k+1,j}$$

Alegerea optimală

- Căutăm optimul dintre toate variantele posibile de alegere ($i \leq k < j$)
- Pentru aceasta, trebuie însă ca și subproblemele folosite să aibă soluție optimală (adică $A_{i, k}$ și $A_{k+1, j}$ să aibă soluție optimă).

Substructura optimală

- Dacă știm că alegerea optimală a soluției pentru problema $A_{i, j}$ implică folosirea subproblemelor ($A_{i, k}$ și $A_{k+1, j}$) și soluția pentru $A_{i, j}$ este optimală, atunci și soluțiile subproblemelor $A_{i, k}$ și $A_{k+1, j}$ trebuie să fie optimale!
- **Demonstrație:** Folosind **metoda cut-and-paste** (metodă standard de demonstrare a substructurii optimale pentru problemele de programare dinamică).
- **Observație:** Nu toate problemele de optim posedă această proprietate! Ex: drumul maxim dintr-un graf orientat.

Definirea recursivă

- Folosind **descompunerea în subprobleme**, **combinarea subproblemelor**, **alegerea optimală** și **substructura optimală** putem să rezolvăm problema prin programare dinamică.
- Următorul pas este **să definim recursiv soluția unei subprobleme**.
- Vrem să găsim o **formulă recursivă** pentru $m[i, j]$ și $s[i, j]$.

Definirea recursivă (II)

$$m[i, j] = \begin{cases} 0 & i = j, \\ \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j) & i < j \end{cases}$$

- Cazurile de bază sunt $m[i, i]$
- Noi vrem să calculăm $m[1, n]$
- Cum alegem $s[i, j]$?
- Bottom-up de la cele mai mici subprobleme la cea inițială.

Rezolvare bottom-up

$m[1, 2], m[2, 3], m[3, 4], \dots, m[n - 3, n - 2], m[n - 2, n - 1], m[n - 1, n]$

$m[1, 3], m[2, 4], m[3, 5], \dots, m[n - 3, n - 1], m[n - 2, n]$

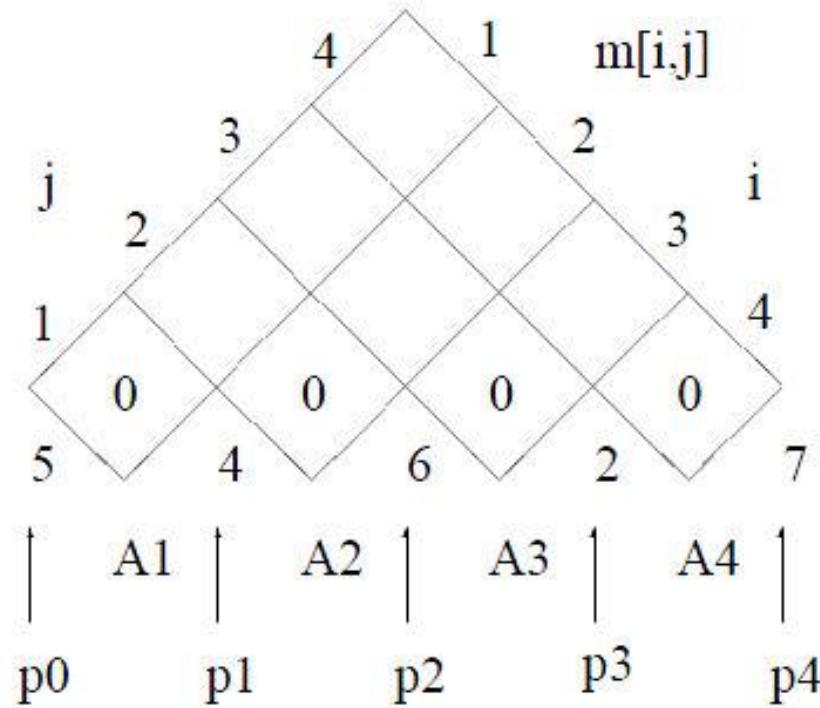
$m[1, 4], m[2, 5], m[3, 6], \dots, m[n - 3, n]$

\vdots

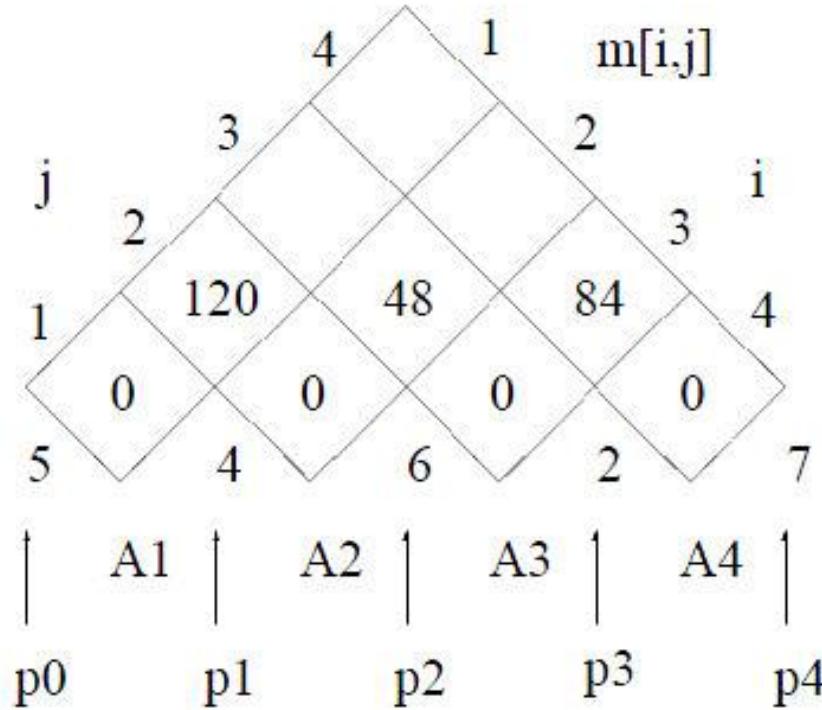
$m[1, n - 1], m[2, n]$

$m[1, n]$

Rezolvare - inițializare



Rezolvare – pas intermediar (I)



Rezolvare – pas intermediar (II)

$$m[i, j] = \begin{cases} 0 & i = j, \\ \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j) & i < j \end{cases}$$

$$A_{1,3} = \{A_1 * A_{2,3}, A_{1,2} * A_3\}$$

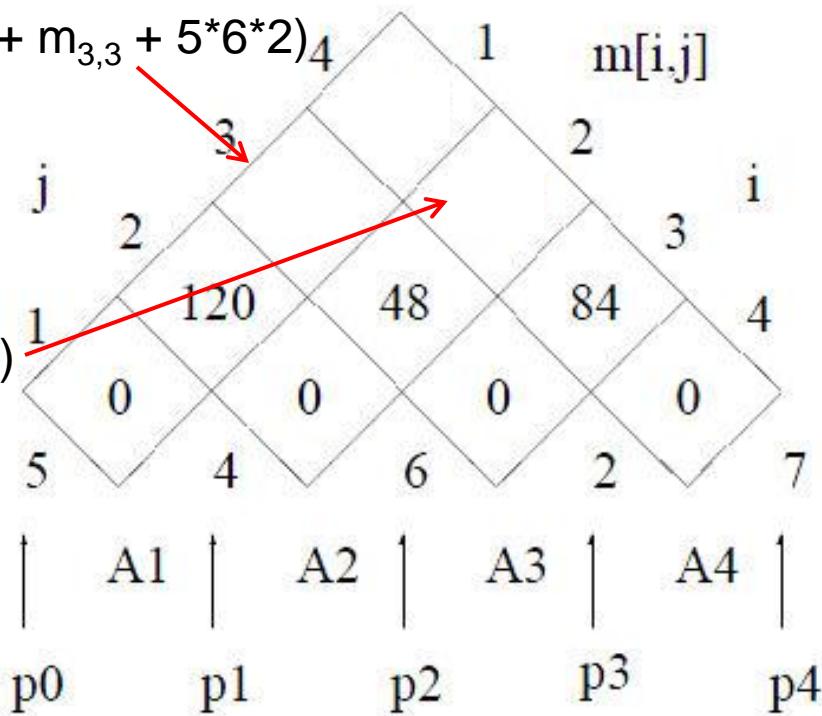
$$m_{1,3} = \min(m_{1,1} + m_{2,3} + 5*4*2, m_{1,2} + m_{3,3} + 5*6*2)$$

$$s_{1,3} = 1$$

$$A_{2,4} = \{A_2 * A_{3,4}, A_{2,3} * A_4\}$$

$$m_{2,4} = \min(m_{3,4} + 4*6*2, m_{2,3} + 4*2*7)$$

$$s_{2,4} = 3$$



Rezolvare – final

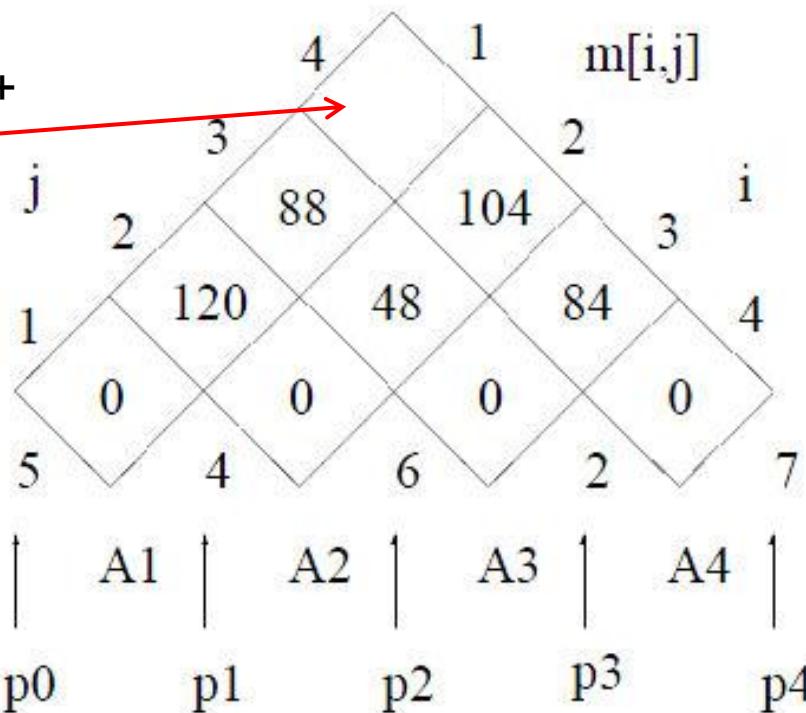
$$m[i, j] = \begin{cases} 0 & i = j, \\ \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j) & i < j \end{cases}$$

$A_{1,4} = \{A_1 * A_{2,4}, A_{1,2} * A_{3,4}, A_{1,3} * A_4\}$
 $m_{1,3} = \min(m_{2,4} + 5*4*7, m_{1,2} + m_{3,4} + 5*6*7, m_{1,3} + 5*2*7) = 158$
 $s_{1,4} = 3$

Parantezarea optimă este:

$$(A_1(A_2 A_3)) A_4$$

Numărul optim de operații
este: 158



Pseudocod

- Înmulțire_matrici (p, n)
 - Pentru i de la 1 la n // initializare
 - $m[i, i] = s[i, j] = 0$
 - Pentru $/$ de la 2 la n // dimensiune problema
 - Pentru i de la 1 la $n - l + 1$ // indice stânga
 - $j = i + l - 1$ // indice dreapta
 - $m[i, j] = \infty$ // pentru determinare minim
 - Pentru k de la i la $j - 1$
 - $q = m[i, k] + m[k + 1, j] + p[i - 1] * p[k] * p[j]$
 - Dacă $q < m[i, j]$
 - $m[i, j] = q$
 - $s[i, j] = k$

• Întoarce m și s

Complexitate

- Spațială: $\Theta(n^2)$
 - Pentru memorarea soluțiilor subproblemelor
- Temporală: $O(n^3)$
 - N_s : Număr total de subprobleme: $O(n^2)$
 - N_a : Număr total de alegeri la fiecare pas: $O(n)$
 - Complexitatea: $O(n^3)$ este de obicei egală cu $N_s \times N_a$

Alt exemplu: Arbori optimi la căutare (AOC)

- **Def 2.1:** Fie K o mulțime de chei. Un **arbore binar cu cheile K** este un **graf orientat și aciclic** $A = (V, E)$ a.î.:
 - Fiecare nod $u \in V$ conține o singură cheie $k(u) \in K$ iar cheile din **noduri sunt distințe**.
 - Există un **nod unic** $r \in V$ a.î. $i\text{-grad}(r) = 0$ și $\forall u \neq r, i\text{-grad}(u) = 1$.
 - $\forall u \in V, e\text{-grad}(u) \leq 2$; $S(u) / D(u) =$ succesorul stânga / dreapta.
- **Def 2.2:** Fie K o mulțime de chei peste care există o **relație de ordine** \prec . Un **arbore binar de căutare** satisface:
 - $\forall u, v, w \in V$ avem $(v \in S(u) \Rightarrow \text{cheie}(v) \prec \text{cheie}(u)) \wedge (w \in D(u) \Rightarrow \text{cheie}(u) \prec \text{cheie}(w))$

Căutare într-un arbore de căutare

- Caută(elem, Arb)
 - Dacă Arb = null
 - Întoarce null
 - Dacă elem = Arb.val // valoarea din nodul crt.
 - Întoarce Arb
 - Dacă elem < Arb.val
 - Întoarce Caută(elem, Arb.st)
 - Întoarce Caută(elem, Arb.dr)

Complexitate: $\Theta(\log n)$

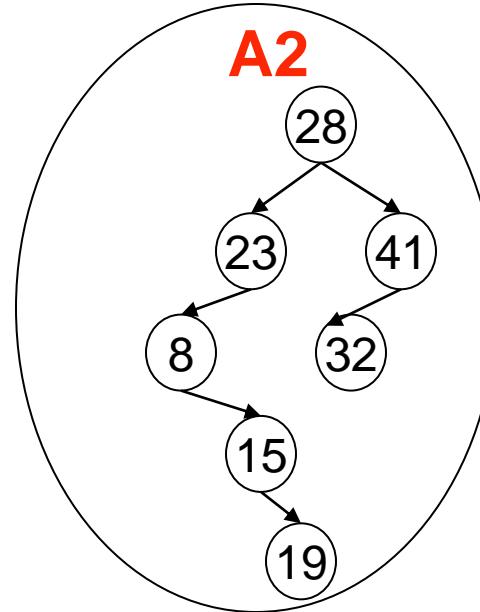
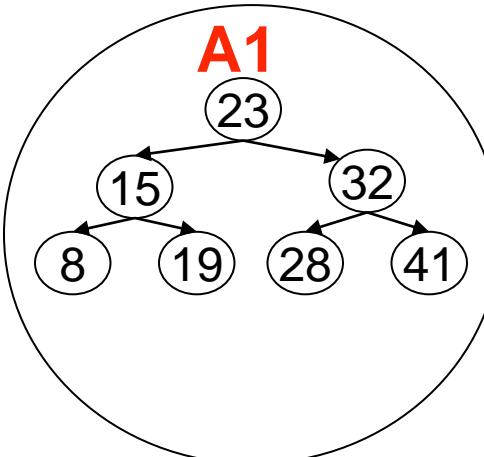
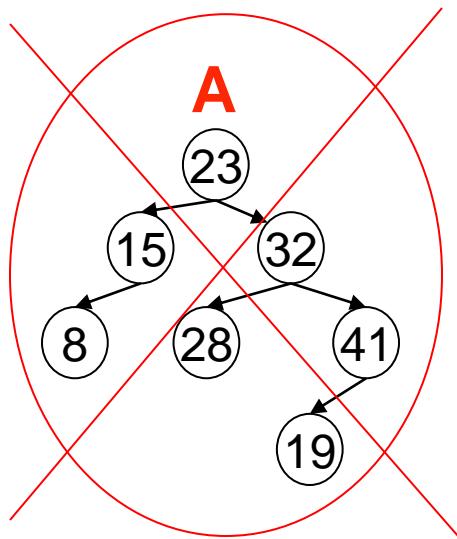
Inserție într-un arbore de căutare

- Inserare(elem, Arb)
 - Dacă Arb = vid // adaug cheia în arbore
 - nod_nou(elem, null, null)
 - Dacă elem = Arb.val // valoarea există deja
 - Întoarce Arb
 - Dacă elem < Arb.val
 - Întoarce Inserare(elem, Arb.st) // adaugă în stânga
 - Întoarce Inserare(elem, Arb.dr) // sau în dreapta

Complexitate: $\Theta(\log n)$

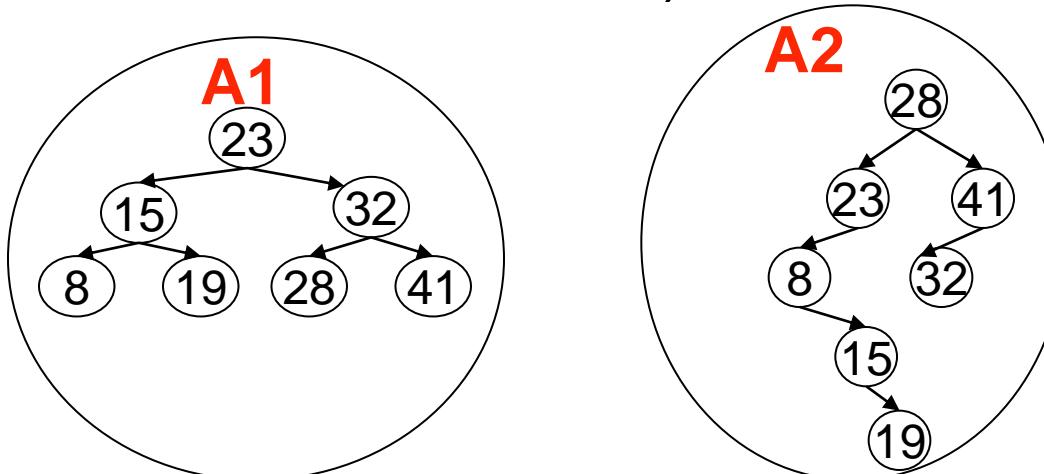
Exemplu de arbori de căutare

- Cu aceleasi chei se pot construi arbori distincți



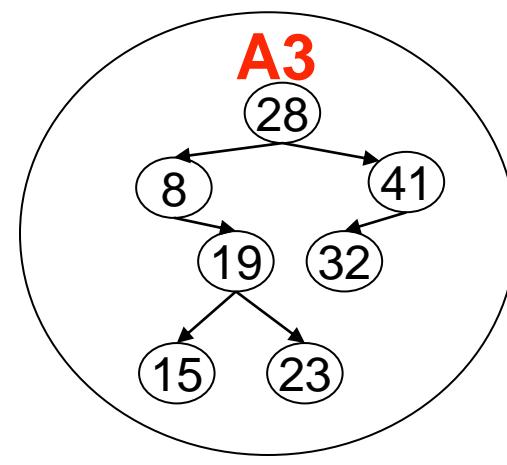
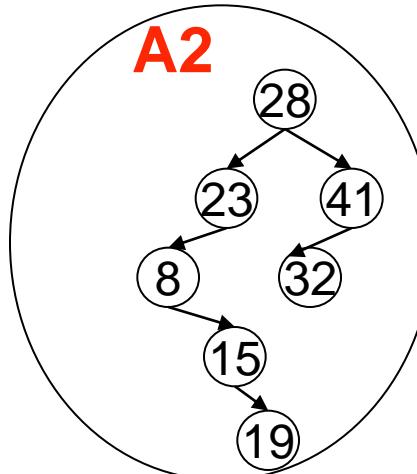
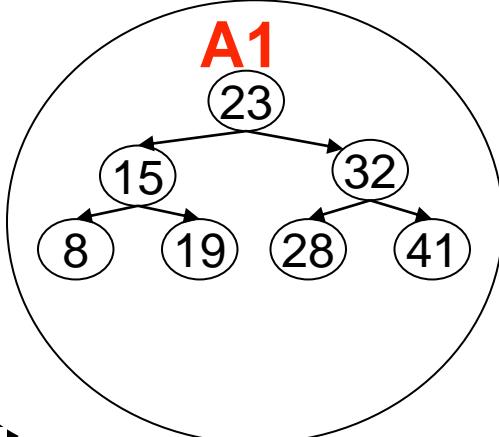
Exemplu (I)

- Presupunem că elementele din A1 și A2 au **probabilități de căutare egale**:
 - Numărul mediu de comparații pentru A1 va fi:
 $(1 + 2 + 2 + 3 + 3 + 3 + 3) / 7 = 2.42$
 - Numărul mediu de comparații pentru A2 va fi:
 $(1 + 2 + 2 + 3 + 3 + 4 + 5) / 7 = 2.85$



Exemplu (II)

- Presupunem că elementele au **următoarele probabilități**:
 - 8: 0.2; 15: 0.01; 19: 0.1; 23: 0.02; 28: 0.25; 32: 0.2; 41: 0.22;
 - Numărul mediu de comparații pentru A1:
 - $0.02*1+0.01*2+0.2*2+0.2*3+0.1*4+0.25*3+0.22*3=2.85$
 - Numărul mediu de comparații pentru A2:
 - $0.25*1+0.02*2+0.22*2+0.2*3+0.2*3+0.01*4+0.1*5=2.47$



Probleme

- Costul căutării **depinde de frecvența** cu care este căutat fiecare termen.
- → Ne dorim ca **termenii cei mai des căutați** să fie **cât mai aproape de vârful arborelui** pentru a micșora numărul de apeluri recursive.
- Dacă arborele este **construit prin sosirea aleatorie a cheilor** putem ajunge la o simplă listă cu **n elemente**.

Definiție AOC

- Definiție: Fie A un arbore binar de căutare cu chei într-o mulțime K; fie $\{x_1, x_2, \dots, x_n\}$ cheile conținute în A, iar $\{y_0, y_1, \dots, y_n\}$ chei reprezentante ale cheilor din K ce nu sunt în A astfel încât: $y_{i-1} < x_i < y_i, i = \overline{1, n}$. Fie p_i , $i = 1, n$ probabilitatea de a căuta cheia x_i și q_j , $j = 0, n$ probabilitatea de a căuta o cheie reprezentată de y_j . Vom avea relația: $\sum_{i=1}^n p_i + \sum_{j=0}^n q_j = 1$. Se numește arbore de căutare probabilistică, un arbore cu costul:

$$Cost(A) = \sum_{i=1}^n (nivel(x_i, A) + 1) * p_i + \sum_{j=0}^n nivel(y_j, A) * q_j$$

- Definiție: Un arbore de căutare probabilistică având cost minim este un arbore optim la căutare (AOC).

Algoritm AOC naiv

- Generarea permutărilor $x_1, \dots x_n$.
- Construcția arborilor de căutare corespunzători.
- Calcularea costului pentru fiecare arbore.
- Alegerea arborelui de cost minim.
- Complexitate: $\Theta(n!)$ (deoarece sunt $n!$ permutări).
- → căutăm altă variantă!!!

Construcția AOC – Notații

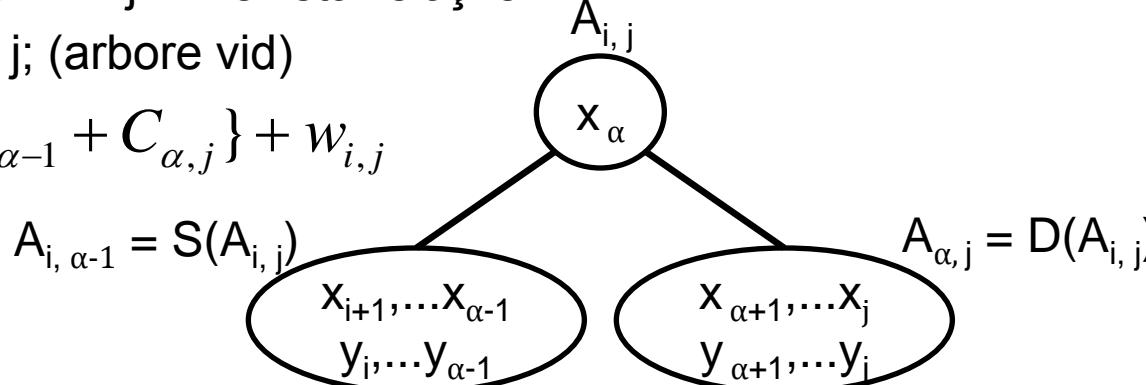
- $A_{i,j}$ desemnează un AOC cu cheile $\{x_{i+1}, x_{i+2}, \dots, x_j\}$ în noduri și cu cheile $\{y_i, y_{i+1}, \dots, y_j\}$ în frunzele fictive.
 - $C_{i,j} = \text{Cost } (A_{i,j})$. $Cost(A_{ij}) = \sum_{k=i+1}^j (\text{nivel}(x_k, A_{ij}) + 1) * p_k + \sum_{k=i}^j \text{nivel}(y_k, A_{ij}) * q_k$
 - $R_{i,j}$ este indicele α al cheii x_α din rădăcina arborelui $A_{i,j}$.
- $$w_{i,j} = \sum_{k=i+1}^j p_k + \sum_{k=i}^j q_k \quad w_{i,j} = \sum_{k=i+1}^{j-1} p_k + p_j + \sum_{k=i}^{j-1} q_k + q_j = w_{i,j-1} + p_j + q_j$$
- **Observație:** $A_{0,n}$ este chiar arborele A , $C_{0,n} = \text{Cost } (A)$ iar $w_{0,n} = 1$.

Construcția AOC - Demonstrație

- Lemă: Pentru orice $0 \leq i \leq j \leq n$ există relațiile:

- $C_{i,j} = 0$, dacă $i = j$; (arbore vid)

- $C_{i,j} = \min_{i < \alpha \leq j} \{C_{i,\alpha-1} + C_{\alpha,j}\} + w_{i,j}$



- Demonstrație:

$$Cost(A_{ij}) = \sum_{k=i+1}^j (nivel(x_k, A_{ij}) + 1) * p_k + \sum_{k=i}^j nivel(y_k, A_{ij}) * q_k$$

$$\rightarrow C_{i,j} = C_{i,\alpha-1} + C_{\alpha,j} + w_{i,j}$$

- $C_{i,j}$ depinde de indicele α al nodului rădăcină.

- dacă $C_{i,\alpha-1}$ și $C_{\alpha,j}$ sunt minime (costurile unor AOC) $\rightarrow C_{i,j}$ este minim.

Construcția AOC

- 1. În etapa d , $d = 1, 2, \dots, n$ se calculează costurile și indicele cheilor din rădăcina arborilor AOC $A_{i, i+d}$, $i = 0, n-d$ cu d noduri și $d + 1$ frunze fictive.
- Arboarele $A_{i, i+d}$ conține în noduri cheile $\{x_{i+1}, x_{i+2}, \dots, x_{i+d}\}$, iar în frunzele fictive sunt cheile $\{y_i, y_{i+1}, \dots, y_{i+d}\}$. Calculul este efectuat pe baza rezultatelor obținute în etapele anterioare.
- Conform lemei avem
$$C_{i,i+d} = \min_{i < \alpha \leq i+d} \{C_{i,\alpha-1} + C_{\alpha,i+d}\} + w_{i,i+d}$$
- Rădăcina $A_{i, i+d}$ are indicele $R_{i,j} = \alpha$ care minimizează $C_{i, i+d}$.
- 2. Pentru $d = n$, $C_{0,n}$ corespunde arborelui AOC $A_{0,n}$ cu cheile $\{x_1, x_2, \dots, x_n\}$ în noduri și cheile $\{y_0, y_1, \dots, y_n\}$ în frunzele fictive.

Algoritm AOC

```
| AOC(x, p, q, n)
|   Pentru i de la 0 la n
|     {Ci,i = 0, Ri,i = 0, wi,i = qi} // inițializare costuri AOC vid Ai,i
|   Pentru d de la 1 la n
|     Pentru i de la 0 la n-d // calcul indice rădăcină și cost pentru Ai,i+d
|       j = i + d, Ci,j = ∞, wi,j = wi,j-1 + pj + qj
|       Pentru α de la i + 1 la j // ciclul critic – operații intensive
|         Dacă (Ci,α-1 + Cα,j < Ci,j) // cost mai mic?
|           { Ci,j = Ci,α-1 + Cα,j; Ri,j = α } // update
|           Ci,j = Ci,j + wi,j // update
|       Întoarce gen_AOC(C, R, x, 0, n) // construcție efectivă arbore A0,n
|                               // cunoscând indicii
```

Complexitate???

Exemplu constructie AOC (I)

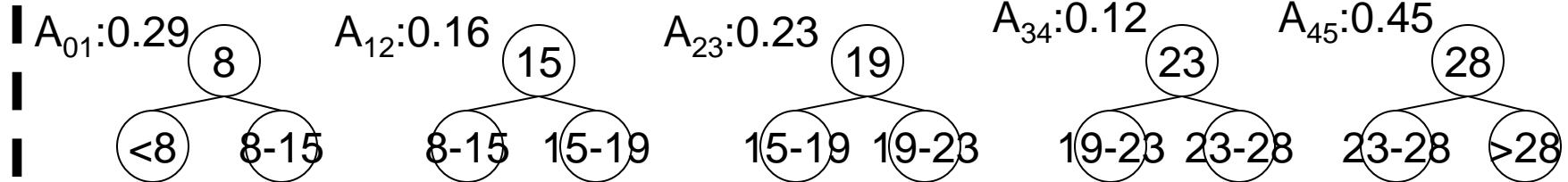
- 8: 0.2; 15: 0.01; 19: 0.1; 23: 0.02; 28: 0.25; (58%)
- [0:8]: 0.02; (8:15): 0.07; (15:19): 0.08; (19:23): 0.05; (23:28): 0.05; (28, ∞): 0.15 (42%)
- $C_{01} = p_1 + q_0 + q_1 = 0.2 + 0.02 + 0.07 = 0.29$
- $C_{12} = p_2 + q_1 + q_2 = 0.01 + 0.07 + 0.08 = 0.16$
- $C_{23} = p_3 + q_2 + q_3 = 0.1 + 0.08 + 0.05 = 0.23$
- $C_{34} = p_4 + q_3 + q_4 = 0.02 + 0.05 + 0.05 = 0.12$
- $C_{45} = p_5 + q_4 + q_5 = 0.25 + 0.05 + 0.15 = 0.45$

$$w_{i,j} = \sum_{k=i+1}^j p_k + \sum_{k=i}^j q_k$$

$$w_{i,j} = w_{i,j-1} + p_j + q_j$$

$$C_{i,i+d} = \min_{i < \alpha \leq i+d} \{ C_{i,\alpha-1} + C_{\alpha,i+d} \} + w_{i,i+d}$$

Exemplu constructie AOC (II)

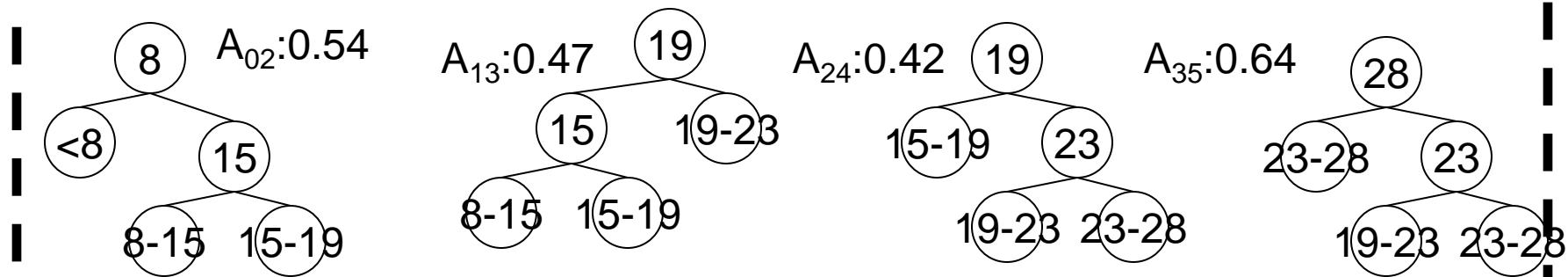


$$C_{02} = \min \{ (C_{00} + C_{12}), (C_{01} + C_{22}) \} + w_{02} = \min(0.16, 0.29) + 0.38 = 0.54 \quad R_{02} = 1 \ (\alpha=1)$$

$$C_{13} = \min \{ C_{11} + C_{23}, C_{12} + C_{33} \} + w_{13} = \min(0.23, 0.16) + 0.31 = 0.47 \quad R_{13} = 3 \ (\alpha=3)$$

$$C_{24} = \min \{ C_{34}, C_{23} \} + w_{24} = \min(0.12, 0.23) + 0.3 = 0.42 \quad R_{24} = 3$$

$$C_{35} = \min \{ C_{45}, C_{34} \} + w_{35} = \min(0.45, 0.12) + 0.52 = 0.64 \quad R_{35} = 5$$

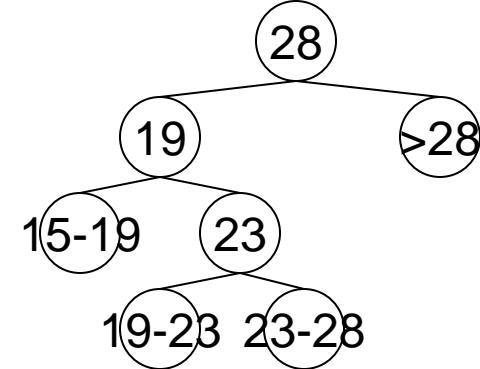
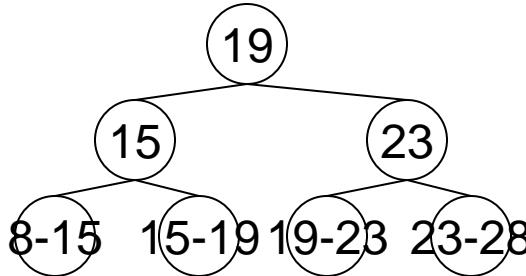
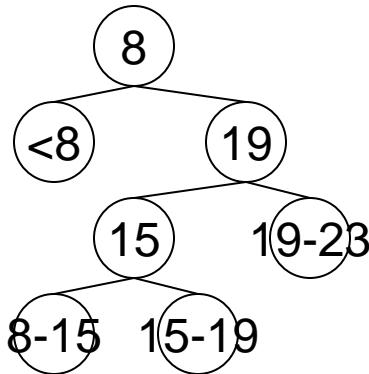


$$w_{i,j} = w_{i,j-1} + p_j + q_j$$

$$C_{i,i+d} = \min_{i < \alpha \leq i+d} \{ C_{i,\alpha-1} + C_{\alpha,i+d} \} + w_{i,i+d}$$

Exemplu constructie AOC (III)

- $C_{03} = \min(C_{00}+C_{13}, C_{01}+C_{23}, C_{02}+C_{33}) + w_{03} = \min(0.47, 0.52, 0.54) + w_{03} \Rightarrow R_{03} = 1$
- $C_{14} = \min(C_{11}+C_{24}, C_{12}+C_{34}, C_{13}+C_{44}) + w_{14} = \min(0.42, 0.28, 0.47) + w_{14} \Rightarrow R_{14} = 3$
- $C_{25} = \min(C_{22}+C_{35}, C_{23}+C_{45}, C_{24}+C_{55}) + w_{25} = \min(0.64, 0.67, 0.42) + w_{25} \Rightarrow R_{25} = 5$



$$w_{i,j} = \sum_{k=i+1}^j p_k + \sum_{k=i}^j q_k$$

$$C_{i,i+d} = \min_{i < \alpha \leq i+d} \{C_{i,\alpha-1} + C_{\alpha,i+d}\} + w_{i,i+d}$$

AOC – Corectitudine (I)

- **Teoremă:** Algoritmul AOC construiește un arbore AOC A cu cheile $x = \{x_1, x_2, \dots, x_n\}$ conform probabilităților de căutare p_i , $i = 1, n$ și q_j , $j = 0, n$.
- **Demonstrație:** prin inducție după etapa de calcul a costurilor arborilor cu d noduri.
- **Caz de bază:** $d = 0$. Costurile $C_{i,i}$ ale arborilor vizi $A_{i,i}$, $i = 0, n$ sunt 0, aşa cum sunt inițializate de algoritm.
- **Pas de inducție:** $d \geq 1$. **Ip. ind.:** pentru orice $d' < d$, algoritmul AOC calculează costurile $C_{i, i+d'}$ și indicii $R_{i, i+d'}$, ai rădăcinilor unor AOC $A_{i, i+d'}$ cu $i = 0, n-d'$ cu cheile $\{x_{i+1}, x_{i+2}, \dots, x_{i+d'}\}$. Trebuie să arătam că valorile $C_{i, i+d}$ și $R_{i, i+d}$ corespund unor AOC $A_{i, i+d}$ cu $i = 0, n-d$ cu cheile $\{x_{i+1}, x_{i+2}, \dots, x_{i+d}\}$.

AOC – Corectitudine (II)

- Pentru d și i fixate, algoritmul calculează:

$$C_{i,i+d} = \min_{i < \alpha \leq i+d} \{C_{i,\alpha-1} + C_{\alpha,i+d}\} + w_{i,i+d}$$

- unde costurile $C_{i,\alpha-1}$ și $C_{\alpha,i+d}$ corespund unor arbori cu un număr de noduri $d' = \alpha - 1 - i$ în cazul $C_{i,\alpha-1}$ și $d' = i + d - \alpha$ în cazul $C_{\alpha,i+d}$.
- $0 \leq d' \leq d - 1 \rightarrow$ aceste valori au fost deja calculate în etapele $d' < d$ și conform ipotezei inducitive \rightarrow sunt costuri și indici ai rădăcinilor unor AOC.
- Conform Lemei anterioare, $C_{i,j}$ este costul unui AOC. Conform algoritmului \rightarrow rădâcina acestui arbore are indicele $r = R_{i,j}$, iar cheile sunt $\{x_{i+1}, x_{i+2}, \dots, x_{r-1}\}$
 $\{x_r\} \{x_{r+1}, x_{r+2}, \dots, x_j\} = \{x_{i+1}, x_{i+2}, \dots, x_j\}$
- Pentru $d = n$, costul $C_{0,n}$ corespunde unui AOC $A_{0,n}$ cu cheile x și cu rădâcina de indice $R_{0,n}$.

AOC – Concluzii

- Câte subprobleme sunt folosite în soluția optimală într-un anumit pas?
AOC: pentru fiecare din cei $n - d + 1$ arbori sunt folosite 2 subprobleme $C_{i, \alpha-1}$ și $C_{\alpha, i+d}$
- Câte variante de ales avem de făcut pentru determinarea alegerii optimale într-un anumit pas?
AOC: pentru fiecare din cei $n - d + 1$ arbori avem $j - i$ candidați pentru rădăcină
- Informal, complexitatea = $N_s * N_a$ (N_s = număr subprobleme; N_a = număr alegeri)

Complexitate AOC: $O(n) * O(n^2) = O(n^3)$

Optimizare – Knuth: $O(n^2)$

ÎNTREBĂRI?

Proiectarea Algoritmilor

Curs 5 – Backtracking și
propagarea restricțiilor

Bibliografie

<http://ktiml.mff.cuni.cz/~bartak/constraints/intro.html>

Problema

	2		8	1		7	4	
7					3	1		
	9				2	8		5
		9		4			8	7
4			2		8			3
1	6			3		2		
3		2	7				6	
		5	6					8
	7	6		5	1		9	

SUDOKU

- Joc foarte la modă cu reguli foarte simple.
- Fiecare rând, coloană sau regiune nu trebuie să conțină decât o dată cifrele de la unu la nouă (Wikipedia).
- Prin trecerea în revistă a soluțiilor posibile pentru acest joc vom explora tehniciile de rezolvare **backtracking și propagarea restricțiilor**.

Soluția 1 – generează și testează

- Generăm toate soluțiile posibile și le testăm.
- 45 spații de completat, 9 posibilități de completare pentru fiecare căsuță => 9^{45} soluții de testat.

1-9	2	1- 9	8	1	1-9	7	4	1-9
7	1-9	1-9	1-9	1-9	3	1	1-9	1-9
1-9	9	1-9	1-9	1-9	2	8	1-9	5
1-9	1-9	9	1-9	4	1-9	1-9	8	7
4	1-9	1-9	2	1-9	8	1-9	1-9	3
1	6	1-9	1-9	3	1-9	2	1-9	1-9
3	1-9	2	7	1-9	1-9	1-9	6	1-9
1-9	1-9	5	6	1-9	1-9	1-9	1-9	8
1-9	7	6	1-9	5	1	1-9	9	1-9

Soluția 2 – Backtracking cronologic (orb) (I)

- Construiește soluțiile iterativ.
- Menține evidența alegerilor făcute.
- În momentul în care se ajunge la o **contradicție** se revine **la ultima decizie** luată și se încearcă alegerea unei alte variante.

Soluția 2 – Backtracking cronologic (orb)

(II)

1	2	3	4	5	6	7	8	9
2	3	4	5	6	7	8	9	1
3	4	5	6	7	8	9	1	2
4	5	6	7	8	9	1	2	3
5	6	7	8	9	1	2	3	4

1	2	3	4	5	6	7	8	9
2	3	4	5	6	7	8	9	1
3	4	5	6	7	8	9	1	2
4	5	6	7	8	9	1	2	3
5	6	7	8	9	1	2	3	4

1
2
3
4
5
6
7
8
9
Nu se găsește nici o soluție
pe această cale
deci trebuie să revenim.

Soluția 2 – Backtracking cronologic (orb)

(III)

5	2	3	8	1	6	7	4	9
7	4	8	5	9	3	1	2	6
6	9	1	4	7	2	8	3	5
2	3	9	1	4	5	6	8	7
4	5	7	2	6	8	9	∅	3
1	6	∅		3		2		
3	2	7			6			
	5	6		5	1		9	
PA	7	6						

1 Se încearcă schimbarea
 2 ultimei alegeri (în acest
 3 caz nu se poate deci
 4 revenim iar la alegerea
 precedență).

1 Găsim o nouă soluție
 2 posibilă și reluăm
 3 avansul.

5	2	3	8	1	6	7	4	9
7	4	8	5	9	3	1	2	6
6	9	1	4	7	2	8	3	5
2	3	9	1	4	5	6	8	7
4	5	7	2	6	8	9	∅	3
1	6	∅		3		2		
3	2	7			6			
	5	6		5	1		9	
7	6							



Soluția 2 – Backtracking cronologic (orb) (IV)

Schema Backtracking

- Soluție-parțială \leftarrow INIT // initializez
- EŞEC-DEFINITIV \leftarrow fals // nu am ajuns (încă) la eșec
- Cât timp Soluție-parțială nu este soluție finală și nu avem EŞEC-DEFINITIV
 - Soluție-parțială \leftarrow AVANS (Soluție-parțială) // avansez
 - Dacă EŞEC (Soluție-parțială) // nu mai pot avansa
 - Atunci REVENIRE (Soluție-parțială) // mă întorc
- Dacă EŞEC-DEFINITIV
 - Atunci Întoarce EŞEC // nu s-a găsit nicio soluție
 - Altfel Întoarce SUCCES // am ajuns la soluția problemei
- Sfârșit.

Procedura AVANS (Soluție-parțială)

- Dacă există alternativă de extindere // pot avansa?
 - Atunci Soluție-parțială \leftarrow Soluție-parțială \cup alternativă de extindere // avansez
 - Altfel Dacă Soluție-parțială este INIT
 - Atunci EŞEC-DEFINITIV \leftarrow adevărat // nu s-au găsit soluții pentru problemă
 - Altfel EŞEC (Soluție-parțială) // ramura curentă a dus la eșec

Backtracking – optimizări posibile (I)

- Alegerea variabilelor în altă ordine.
- Îmbunătățirea revenirilor.
 - Necesită detectarea **cauzei** producerii erorii.
- Evitarea redundanțelor în spațiul de căutare (**îmbunătățirea avansului**).
 - Evitarea repetării unei căutări care știm că va duce la un rezultat greșit.

Backtracking – optimizări posibile (II)

Îmbunătățirea revenirilor

Revenire la alegerea variabilei care a cauzat eșecul (8 nu poate fi pus decât la poziția indicată).

The diagram shows a 9x9 Sudoku grid during the solving process. The grid contains the following values:

- Row 1: 5, 2, 3, 8, 1, 6, 7, 4, 9
- Row 2: 7, 4, 8, 5, 9, 3, 1, 2, 6
- Row 3: 6, 9, 1, 4, 7, 2, 8, 3, 5
- Row 4: 2, 3, 9, 1, 4, 5, 6, 8, 7
- Row 5: 4, 5, 7, 2, 6, 8, 9, 1, 3
- Row 6: 1, 6, 0 (empty), 3, 2
- Row 7: 3, 2, 7, 6, 8
- Row 8: 7, 6, 5, 1, 9
- Row 9: 8, 6, 1, 3, 2, 4, 5, 7, 9

Red numbers indicate values that have been eliminated by row, column, or 3x3 grid rules. Red crossed-out numbers indicate previous wrong choices. A red arrow points from the text "Revenire la alegerea variabilei care a cauzat eșecul (8 nu poate fi pus decât la poziția indicată)." to the cell at position (6,2) which contains a circled 0.

Backtracking – optimizări posibile (III)

Evitarea redundanțelor în spațiul de căutare

Alegerea lui 8 pe această poziție va produce un eșec în viitor indiferent de celelalte alegeri făcute deci în cazul revenirii în această poziție nu are sens să mai facem această alegere.

The diagram shows a 9x9 Sudoku grid during a backtracking search. The grid contains the following values:

- Row 1: 5, 2, 3, 8, 1, 6, 7, 4, 9
- Row 2: 7, 4, 8, 5, 9, 3, 1, 2, 6
- Row 3: 6, 9, 1, 4, 7, 2, 8, 3, 5
- Row 4: 2, 3, 9, 1, 4, 5, 6, 8, 7
- Row 5: 4, 5, 7, 2, 6, 8, 9, 1, 3
- Row 6: 1, 6, Ø, 3, 2
- Row 7: 3, 2, 7, 6, 8
- Row 8: 5, 6, 5, 1, 9
- Row 9: 7, 6, 5, 1, 9

Red numbers indicate invalid or pruned values. A red arrow points from the text "Alegerea lui 8 pe această poziție va produce un eșec în viitor indiferent de celelalte alegeri făcute deci în cazul revenirii în această poziție nu are sens să mai facem această alegere." to the value 8 in the second row, third column.

Restricții, rețele de restricții, probleme de prelucrarea restricțiilor

- **Definiție:** O **restricție** c este o relație între una sau mai multe **variabile** v_1, \dots, v_m , (denumite **nodurile** sau **celulele** restricției). Fiecare variabilă v_i poate lua valori într-o anumită mulțime D_i , denumită **domeniul** ei (ce poate fi finit sau nu, numeric sau nu).
- **Definiție:** Se spune că un **tuplu** (o atribuire) de valori (x_1, \dots, x_m) din domeniile corespunzătoare celor m variabile **satisfac** restricția $c(v_1, \dots, v_m)$, dacă $(x_1, \dots, x_m) \in c(v_1, \dots, v_m)$.

Exprimarea restricțiilor

- Enumerarea tupluri restricției.
 - $(5,2,3,8,1,6,7,4,9)$;
 $(6,2,3,8,1,5,7,4,9)$; etc.
- Formule matematice, cum ar fi ecuațiile sau inecuațiile.
 - $0 < V_{1j} < 10; V_{1j} \neq V_{1k}; \forall j \neq k, 0 < j, k < 10$
- Precizarea unei multimi de reguli.

	2		8	1		7	4	
7					3	1		
	9				2	8		5
		9		4			8	7
4			2		8			3
1	6			3		2		
3		2	7				6	
		5	6					8
	7	6		5	1		9	

Tipuri de restricții

- **Unare**

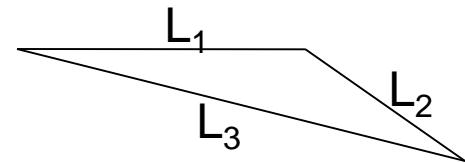
- Specificarea domeniului variabilei.
- $V_{11} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \setminus \{2, 8, 1, 7, 4, 3, 9\}$

- **Binare**

- Între 2 variabile.
- $V_{1j} \neq V_{1k} \quad \forall j \neq k, 0 < j, k < 10$

- **N-are**

- Între n variabile.
- Regula triunghiului: $L_1 + L_2 > L_3$.



Probleme de satisfacerea restricțiilor

- Definiție: O problemă de satisfacere a restricțiilor (PSR) este un triplet $\langle V, D, C \rangle$, format din:
 - o mulțime V formată din n variabile $V = \{v_1, \dots, v_n\}$;
 - mulțimea D a domeniilor de valori corespunzătoare acestor variabile: $D = \{D_1, \dots, D_n\}$;
 - o mulțime C de restricții $C = \{c_1, \dots, c_p\}$ peste submulțimi ale mulțimii V ($c_i(v_{i1}, \dots, v_{ij}) \subseteq D_{i1} \times D_{i2} \times \dots \times D_{ij}$).
- Conform Definiției tuplului, o restricție $c_i(v_{i1}, \dots, v_{ij})$, este o submulțime a produsului cartezian $D_{i1} \times D_{i2} \times \dots \times D_{ij}$, constând din toate tuplurile de valori considerate că satisfac restricția pentru (v_{i1}, \dots, v_{ij}) .

Soluții ale PSR

- Definiție: O soluție a unei PSR $\langle V, D, C \rangle$ este un **tuplu de valori** $\langle x_1, \dots, x_n \rangle$ care conține toate variabilele din V , din domeniile corespunzătoare din D , astfel încât **toate restricțiile din C să fie satisfăcute**.
- Definiție: PSR binară este o PSR ce conține **doar restricții unare și binare**.

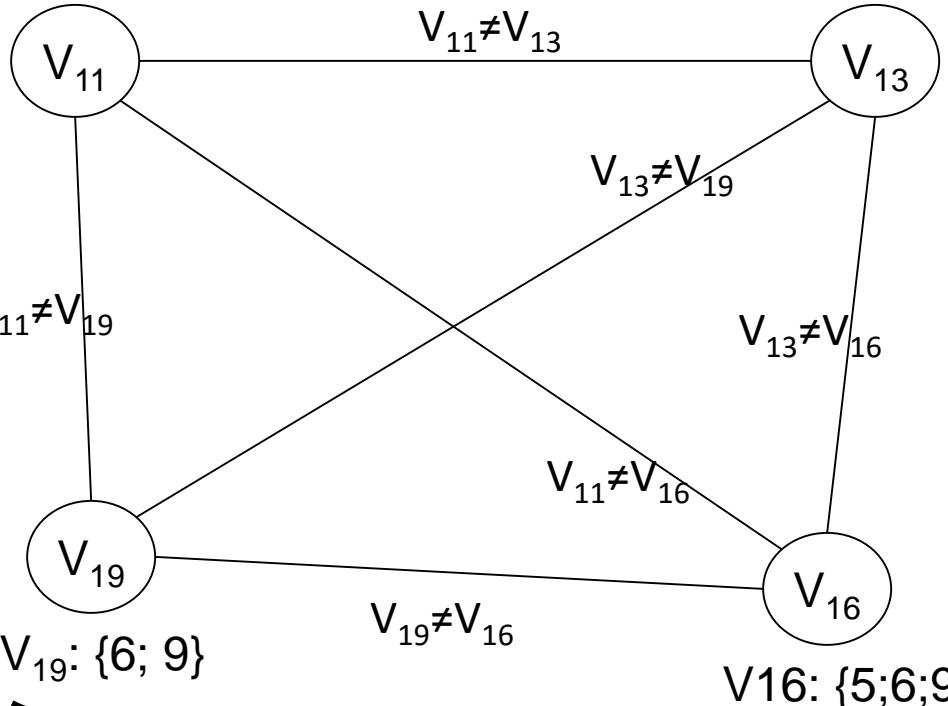
Probleme de satisfacere a restricțiilor - Reprezentare

- Reprezentare PSR prin **rețele de restricții**.
- Reprezentarea folosită: **graf**
 - **Nodurile**: variabilele restricției
 - **Arcele**: restricțiile problemei
 - Valabilă doar pentru PSR binare
- Altă reprezentare posibilă: **graf**
 - **Nodurile**: restricțiile problemei
 - **Arcele**: variabilele restricției
 - Valabilă pentru orice tip de PSR

Exemplu de reprezentare a PSR

- $0 < V_{1j} < 10; V_{1j} \neq V_{1k};$
 $\forall j \neq k, 0 < j, k < 10$

$V_{11}: \{5;6\}$



5;6	2	3	8	1	5;6; 9	7	4	6;9
7						3	1	
9						2	8	5
	9		4				8	7
4		2	8					3
1	6		3			2		
3		2	7				6	
	5	6						8
7	6		5	1			9	

Algoritmi de rezolvare a PSR – Propagarea restricțiilor

- Caracteristici

- Rezolvă PSR binare;
- Variabilele au domenii finite de valori;
- Prin propagarea restricțiilor se **filtrează multimile de valori** (se elimină elementele din domeniu conform unui criteriu dat);
- **Procesul de propagare se oprește când:**
 - O mulțime de valori este vidă → **ESEC**;
 - Nu se mai modifică domeniul vreunei multimi.

Algoritmi de rezolvare a PSR – Propagarea restricțiilor

Notări:

- n = număr variabile = număr **restricții unare**;
- r = număr **restricții binare**;
- G = **rețeaua de restricții** cu variabile drept noduri și restricții drept arce;
- D_i = **domeniul** variabilei i ;
- Q_i = **predicat care verifică restricția unară** pe variabila i ;
- P_{ij} = **predicatul care reprezintă restricția binară** pe variabilele i și j (O muchie între i și j se înlocuiește cu arcele orientate de la i la j și de la j la i);
- $a = \max |D_i|$.

NC-1 (Node Consistency -1)

- Algoritm de consistență nodurilor (pentru restricții unare).
- Procedura NC(i) este:
 - Pentru fiecare $x \in D_i$
 - Dacă $\text{not } Q_i(x)$ // nu este satisfăcută restricția unară
 - Atunci șterge x din D_i
 - Sfârșit.
- Algoritm NC-1 este:
 - Pentru i de la 1 la n Execută NC(i) // pentru fiecare var
 - Sfârșit.

Complexitate NC-1? na

NC-1: Exemplu (I)

- Elimină din domeniul de valori al fiecărui nod valorile care nu satisfac restricțiile care au ca argument variabila din nodul respectiv.
- În cazul Sudoku variabilele inițial iau valori între 1-9.
- Algoritmul NC-1 elimină pentru fiecare variabilă acele valori din domeniu care nu sunt consistente cu valorile fixe (celulele deja fixate).

NC-1: Exemplu (II)

	5;6	2	3;	8	1	5;6; 9	7	4	6;9;
	7	4;5; 8;	4;8;			3	1		
	6;	9	1;3; 4;			2	8		5
		9		4			8	7	
	4			2		8			3
	1	6			3		2		
	3		2	7				6	
			5	6					8
		7	6		5	1		9	

Algoritmi de consistență a arcelor

- Algoritmii de consistență a arcelor înlătură toate inconsistentele submulțimilor de 2 elemente ale rețelei de restricții.
- Funcția REVISE ((i,j)) este:
 - **ȘTERS** \leftarrow fals // nu am modificat domeniul de valori
 - Pentru fiecare $x \in D_i$
 - Dacă nu există $y \in D_j$ a.î. $P_{ij}(x,y)$ // nu se respectă restricția
 - Atunci
 - Șterge x din D_i ;
 - **ȘTERS** \leftarrow adevărat; // am făcut modificări
 - Întoarce **ȘTERS**.

Exemplu funcționare REVISE

x	5;6;	2	3;	8	1	5;6; 9	7	4	6;9;
y	7	4;5; 8;	4;8;			3	1		
	6;	9	1;3;4;			2	8		5
	5;6;2;		9		4			8	7
	4			2		8			3
	1	6			3		2		
	3		2	7				6	
	9;		5	6					8
	8;	7	6		5	1		9	

Complexitate Revise ? a^2

REVISE - Concluzie

- Funcția **Revise** este apelată pentru un arc al grafului de restricții (binare) și **șterge acele valori** din **domeniul** de definiție al **unei variabile** pentru care **nu este satisfăcută restricția** pentru **nici o valoare** corespunzătoare celeilalte variabile a restricției.
- Complexitate Revise: $O(a^2)$

AC-1 (Arc Consistency -1)

- Algoritm AC-1 este:
 - NC-1; // reduc domeniul de valori
 - $Q \leftarrow \{(i,j) \mid (i,j) \in \text{arce}(G), i \neq j\}$ // adaug restricțiile
 - **Repetă**
 - SCHIMBAT \leftarrow fals // nu am modificat niciun domeniu
 - **Pentru fiecare** $(i,j) \in Q$ // pentru fiecare restricție
 - SCHIMBAT \leftarrow (REVISE ((i,j)) sau SCHIMBAT)
 - **Până când non** SCHIMBAT // nu am mai făcut // modificări
 - **Sfârșit.**

AC-1 Caracteristici & Complexitate

- Se aplică algoritmul de consistență nodurilor și apoi se aplică REVISE până nu se mai realizează nici o schimbare.
- Complexitate: $O(na * 2r * a^2)$

La fiecare iterație eliminăm o singură valoare (și avem maxim na valori posibile).

Numărul maxim de apelări al Revise.

Complexitate Revise.

Exemplu AC-1

5;6	2	3;	8	1	5;6; 9	7	4	6;9;
7	4;5; 8;	4;8;			3	1		
6;	9	1;3,4;			2	8		5
5;6;2;	3;5	9		4			8	7
4	5	7	2		8			3
1	6	8		3		2		
3		2	7				6	
9;		5	6					8
8;	7	6		5	1		9	

AC-3 (Arc Consistency -3)

- Algoritm AC-3 este:
 - NC-1; // reduc domeniul de valori
 - $Q \leftarrow \{(i,j) \mid (i,j) \in \text{arce}(G), i \neq j\}$ // adaug restricțiile
 - Cât timp Q nevid
 - Selectează și șterge un arc (k,m) din Q;
 - Dacă REVISE $((k,m))$ // am modificat domeniu - Atunci $Q \leftarrow Q \cup \{(i,k) \mid (i,k) \in \text{arce}(G), i \neq k, i \neq m\}$
// verific dacă nu se modifică și alte domenii

AC-3 Caracteristici

- Se elimină pe rând arcele (constrângerile).
- Dacă o constrângere aduce modificări în rețea adăugăm pentru reverificare nodurile care punctează către nodul de plecare al restricției verificate.
 - Scopul: Reverificarea nodurilor direct implicate de o constrângere din rețea.
- Avantaj: Se fac mult mai puține apeluri ale funcției REVISE.
- Complexitate: $O(a^3r)$.

Backtracking + Propagarea restricțiilor

- În general, propagarea restricțiilor **nu poate rezolva complet** problema dată.
- Metoda ajută la **limitarea spațiului de căutare** (foarte importantă în condițiile în care backtracking-ul are complexitate exponențială!).
- În cazul în care propagarea restricțiilor nu rezolvă problema se folosește:
 - Backtracking pentru **a genera soluții parțiale**;
 - Propagarea restricțiilor după fiecare pas de backtracking pentru a **limita spațiul de căutare** (și eventual a găsi că soluția nu este validă).

ÎNTREBĂRI?

Proiectarea Algoritmilor

Curs 6 – 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://www.graphviz.org>
- <http://en.wikipedia.org/wiki/PageRank>

Plan curs

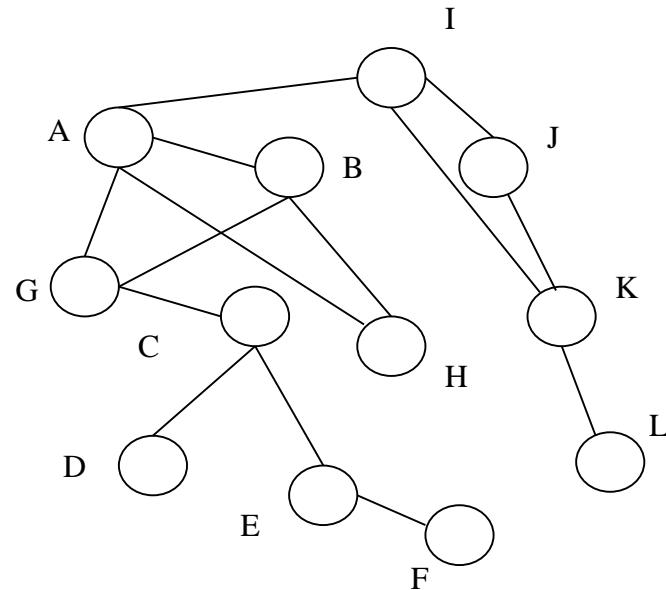
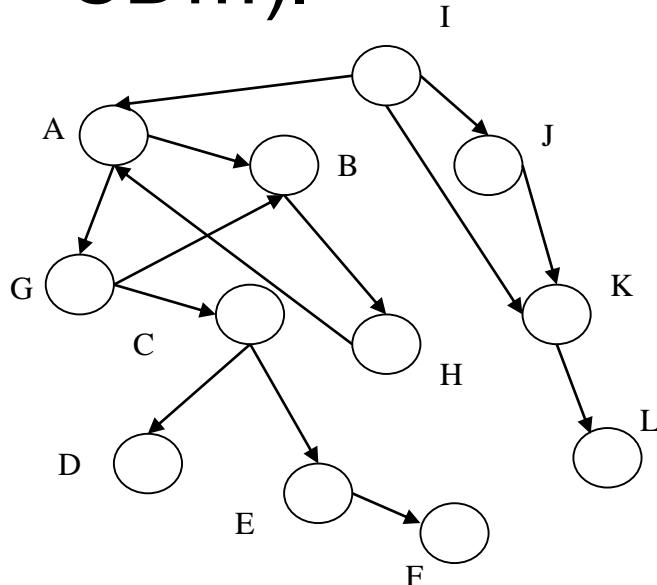
- Introducere
- Modalități de reprezentare
- Exemple de probleme practice
- Algoritmi de parcursere
 - BFS
 - DFS

Introducere

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

Tipuri de grafuri

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

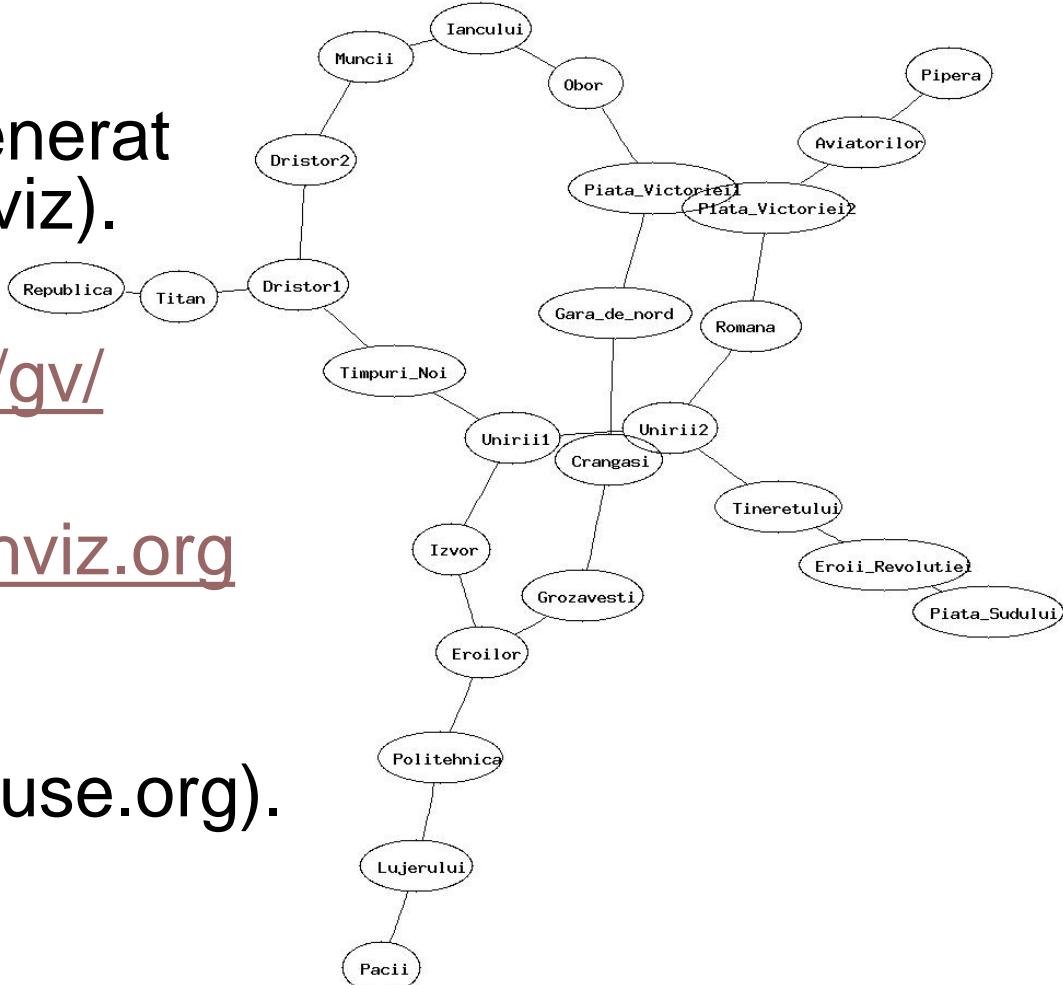


- Neorientate: noduri (A-L) + muchii (AB, BC, CD...).



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, Iancului...

- **Matrice adiacență:**

	Unirii2	Tineretului	Romană	...
Unirii2	-	1	1	
Tineretului	1	-		
Romană	1		-	
...				

- **Tpluri:**

- (Dristor1; Dristor2)
- (Eroilor; Grozăvești)
- ...

- **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--Romană--Piata_Victoriei2--Aviatorilor--Pipera;
- }

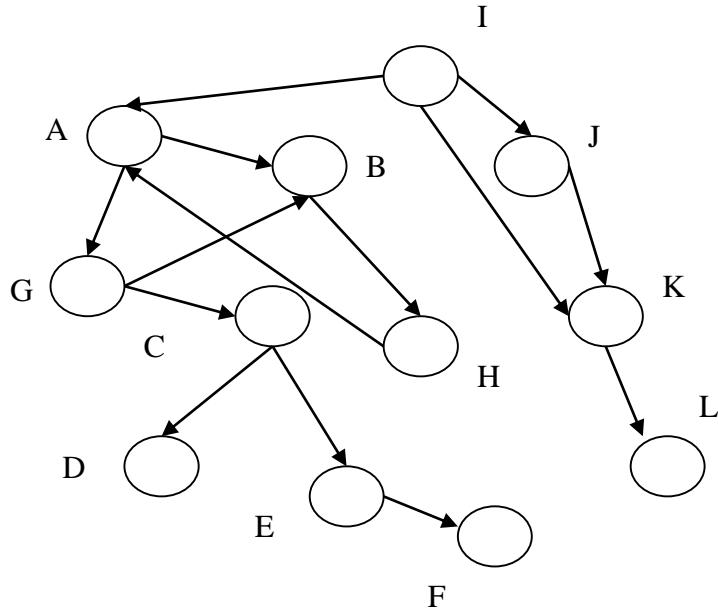
Formate de reprezentare - GraphML

- **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

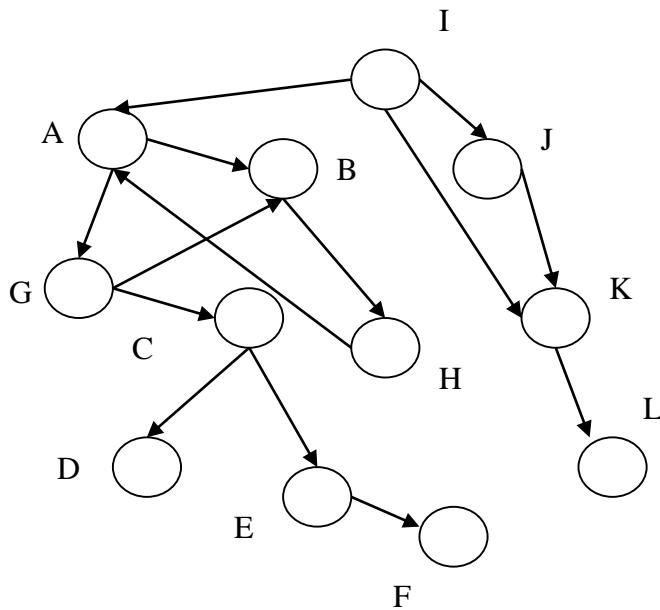


Cum se determină matricea de adiacență?

Matrice de adiacență

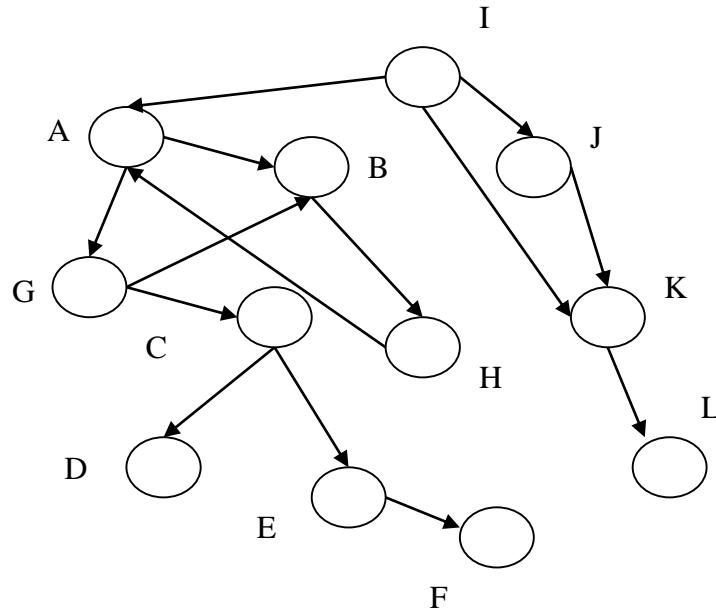
Matricea este **rară**?

- G – rar
- G – dens



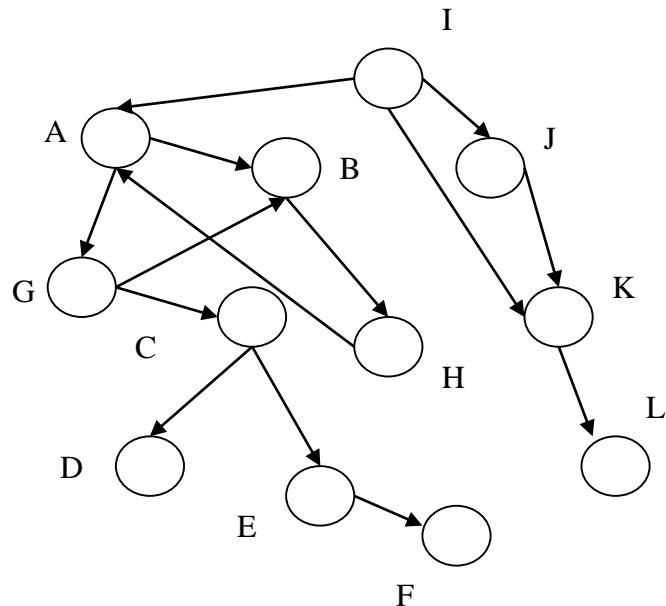
	A	B	C	D	E	F	G	H	I	J	K	L
A	1						1					
B								1				
C								1	1			
D												
E										1		
F												
G		1					1					
H	1											
I	1											
J											1	
K												1
L												

Vector de adiacență



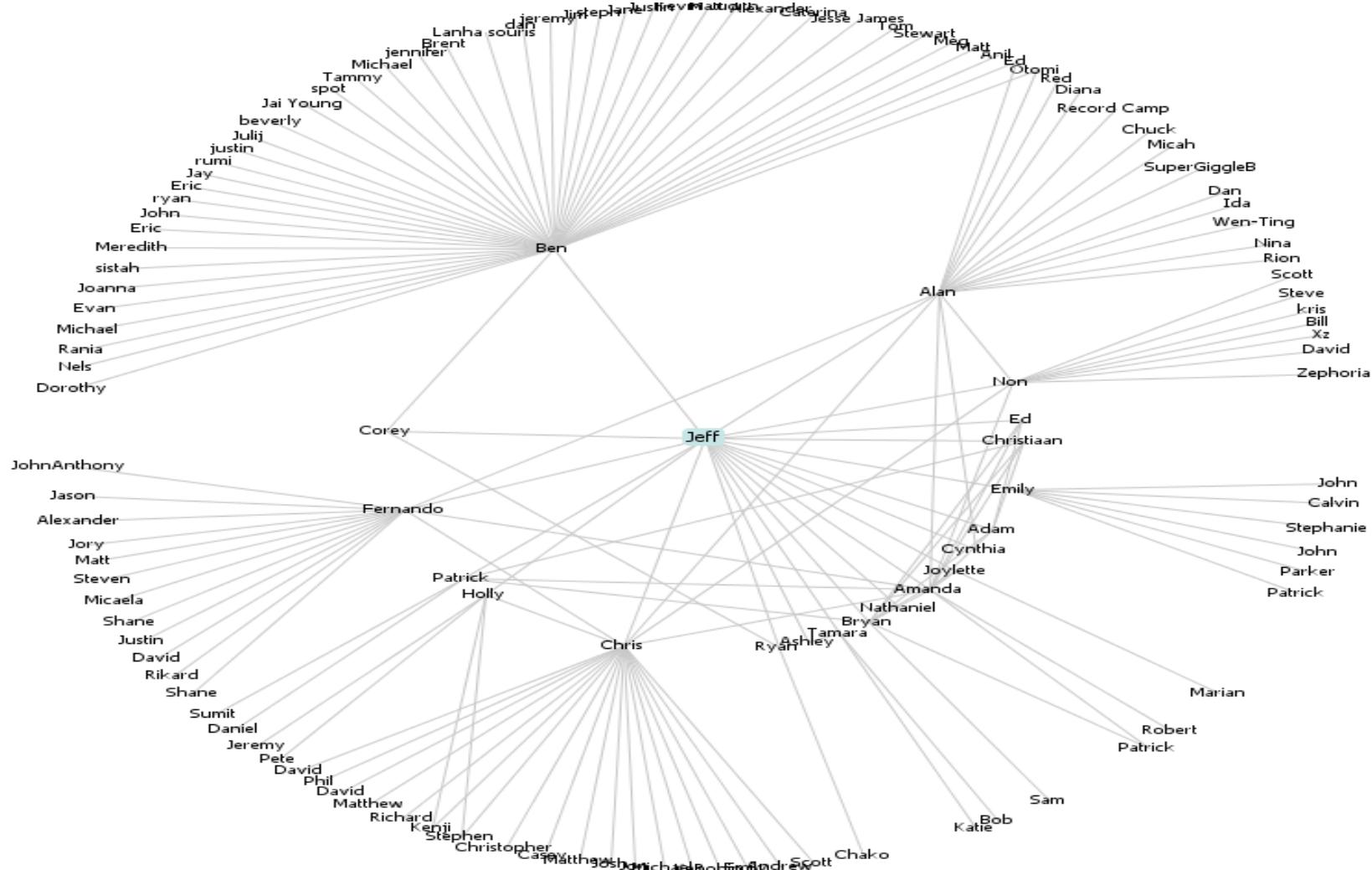
Cum se determină vectorul de adiacență?

Vector de adiacență

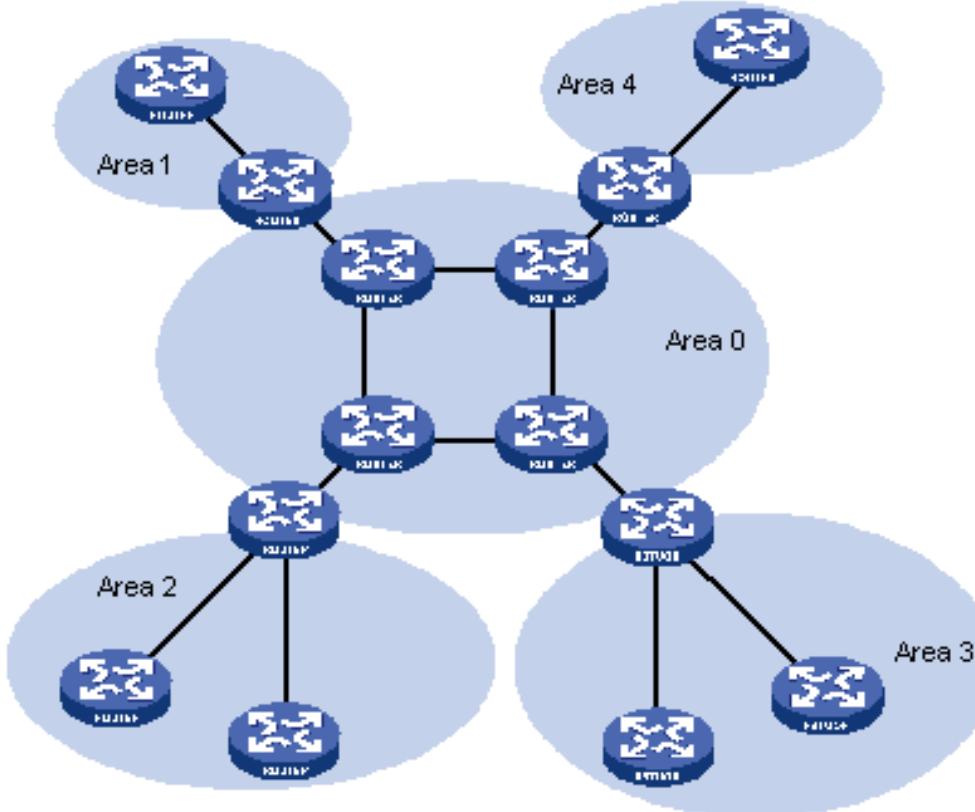


A	B	G	
B	H		
C	D	E	
D			
E	F		
F			
G	B	C	
H	A		
I	A	J	K
J	K		
K	L		
L			

Utilizări practice - Rețele sociale

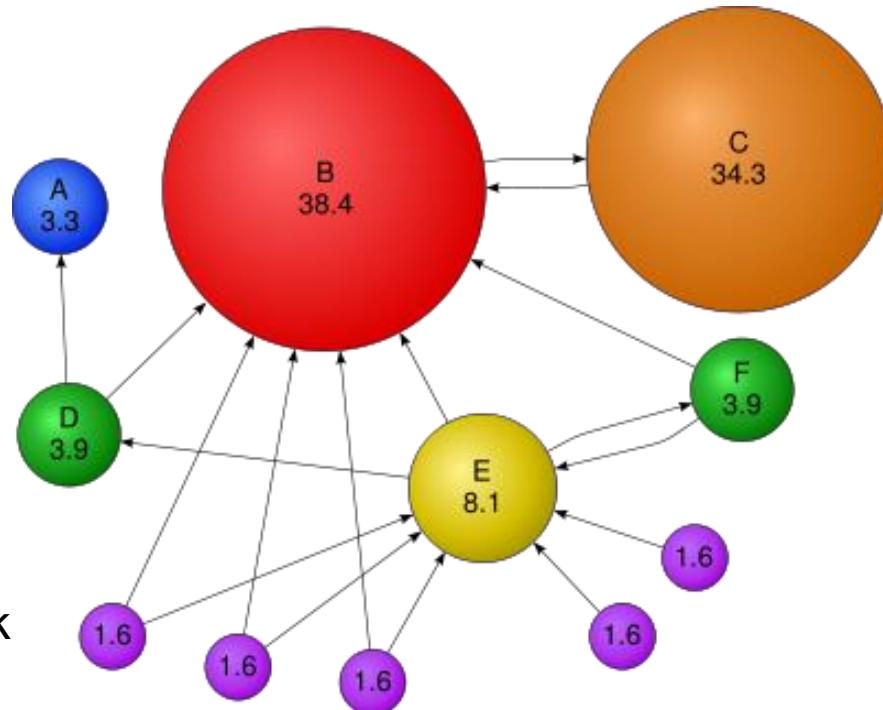
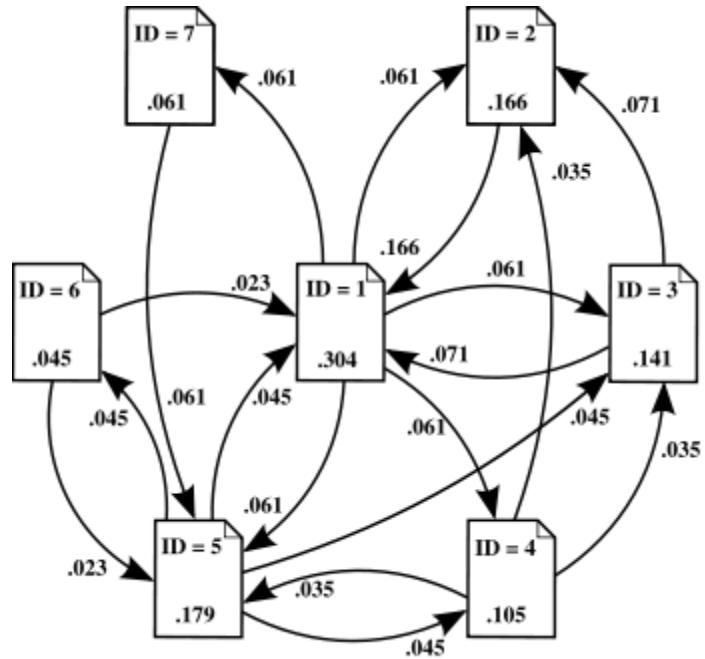


Utilizări practice – Rețele de calculatoare



[http://www.h3c.com/portal/res/200706/01/20070601_108959
image001_201240_57_0.gif](http://www.h3c.com/portal/res/200706/01/20070601_108959_image001_201240_57_0.gif)

Utilizári 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).

Algoritmi de parcurgere – Notații (1)

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

Algoritmi de parcurgere – Notații (2)

- $\text{succs}(u)$ – multimea **succesorilor** lui u (graf orientat) sau multimea 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).
- $p(u)$ ($\pi(u)$) – “**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).
- $\delta(s,u)$ – costul optim al s..u; $\delta(s,u) = \infty \Leftrightarrow u \notin R(s)$.
- **Dist(s,u)** – costul drumului **descoperit** s..u.
- Ex: Politehnica \rightarrow 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 rețin:
 - Părintele – $\pi(u)$ ($p(u)$);
 - $\text{Dist}(s,u)$ – distanța până la nodul sursă;
 - Culoarea nodului.

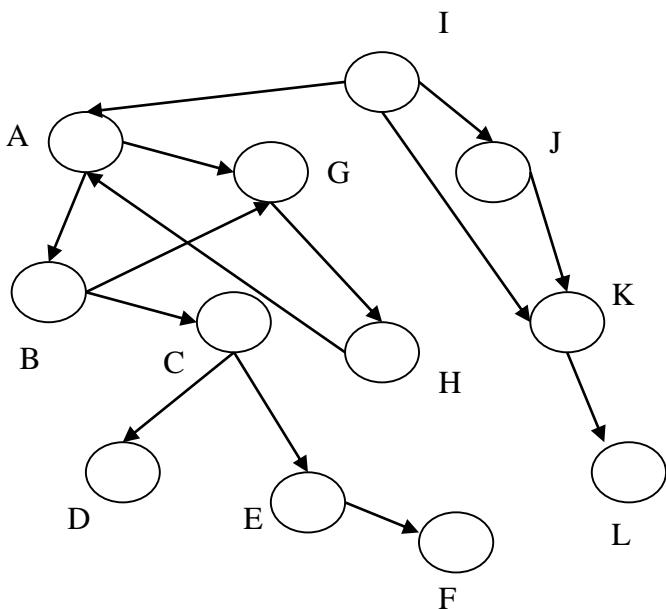
BFS – Algoritm

- BFS(s, G)

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

BFS – Exemplu

Sursa = A

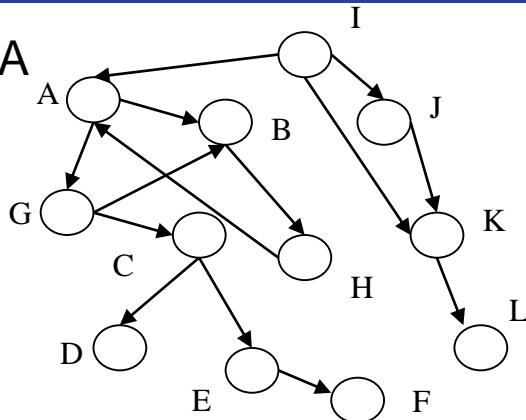


- $\text{BFS}(s, G)$

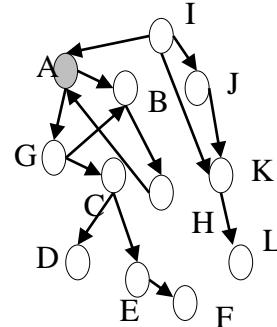
- **Pentru fiecare** nod $u (u \in V)$
 - $p(u) = \text{null}$; $\text{dist}(s,u) = \text{inf}$; $c(u) = \text{alb}$; // inițializări
- $Q = ()$; // se folosește o coadă pentru nodurile de prelucrat
 - $\text{dist}(s,s) = 0$; // actualizări: distanța de la s la s e 0
 - $Q \leftarrow Q + s$; // adăugăm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

BFS – Evoluția explorării

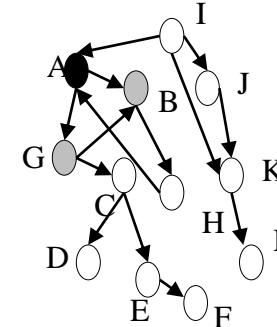
Sursa = A



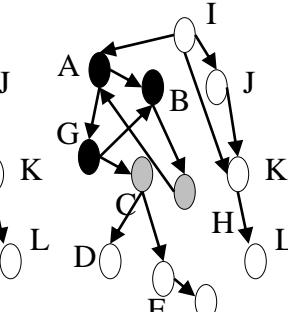
$Q = A$; $d(A) = 0$
 $p(A) = \text{null}$



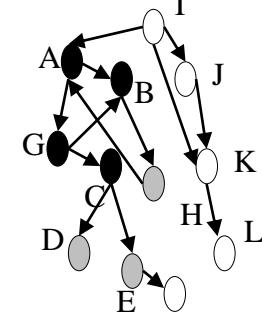
$Q = G, B$
 $d(B) = d(G) = 1$



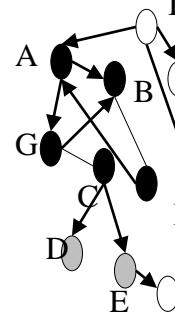
$Q = B, C$
 $d(C) = 2$



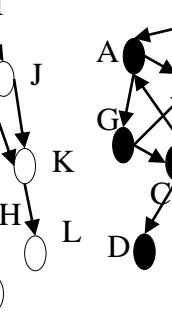
$Q = C, H$
 $d(H) = 2$



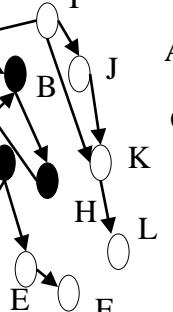
$Q = H, D, E$
 $d(D)=d(E)=3$



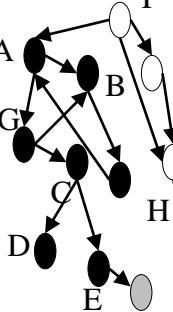
$Q = D, E$



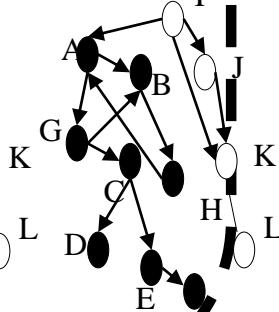
$Q = E$



$Q = F$
 $d(F)=4$



$Q = \emptyset$



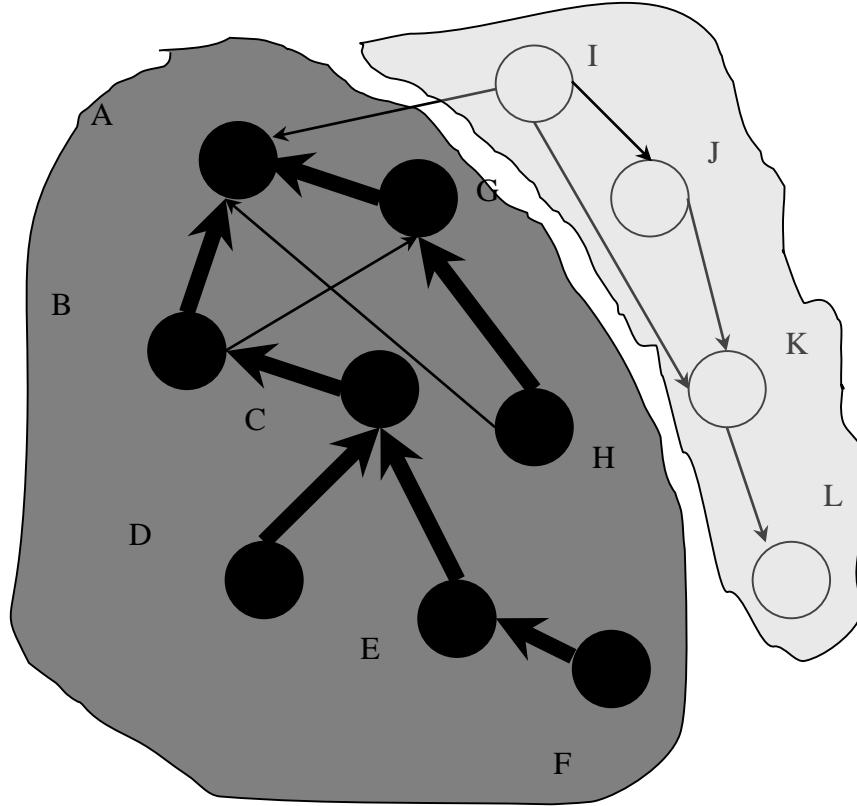
PA
 $p(C) = G$

$p(H) = B$

$p(D) = p(E) = C$

$p(F) = E$

BFS – Zona de explorare



BFS – Proprietăți (I)

• **Lema 5.1.** În cursul execuției $\text{BFS}(s, G)$ $v \in Q \Leftrightarrow v \in R(s)$.

- $\rightarrow v \in Q \rightarrow v \in R(s)$: Q conține exclusiv noduri din $R(s)$ (BFS parcurge toate nodurile ce pot fi atinse din s);
 - Caz de bază: $s \in R(s)$ **Adevărat**
 - Pas inducție: Q' conține doar noduri din $R(s) \rightarrow Q \leftarrow Q' + v$ conține doar noduri din $R(s)$ ($v \in R(s)$)
 - Dacă $u = \text{top}(Q') \rightarrow u \in R(s) \rightarrow v \in \text{succs}(u) \rightarrow (u, v) \in E \rightarrow v \in R(s)$
- \leftarrow toate nodurile ce pot fi atinse din s vor fi introduse cândva în coadă. (drum $s = v_0v_1\dots v_p = v$)
- **Dem prin inducție!** $P(i) = \forall i \in 0..p$, se execută $Q \leftarrow Q + v_i$
 - Caz de bază: $i = 0$, se execută $Q \leftarrow Q + s$ **Adevărat**
 - Pas inducție: $P(i)$ Adevărat $\rightarrow P(i+1)$ Adevărat
 - Conform ipotezei inductive, la un moment dat avem $Q \leftarrow Q + v_i \rightarrow$ cândva, $v_i = \text{top}(Q) \rightarrow$ dacă $c(v_{i+1}) = \text{alb}$, atunci este introdus în Q ($Q \leftarrow Q + v_{i+1}$). Altfel, $c(v_{i+1}) \neq \text{alb} \rightarrow v_{i+1}$ este/a fost deja în coadă pt. că un nod își schimbă culoarea numai la inserție.



BFS – Proprietăți (II)

Lema 5.2. $\forall (u,v) \in E, \delta(s,v) \leq \delta(s,u) + 1$

- $\delta(s,v) \leq \delta(s,u) + 1$ 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! Pp. $\delta(s,v) > \delta(s,u) + 1 \rightarrow \delta(s,u) + 1$ este costul unui drum optim.

• 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!
 - Caz de bază: $dist(s,s) = 0 = \delta(s,s)$ Adevărat
 - Pas inducție: pt. \forall nod din Q': $dist(s,u) \geq \delta(s,u)$.
 - Cum $dist(s,v) = dist(s,u) + 1$ și $dist(s,u) \geq \delta(s,u) \rightarrow dist(s,v) \geq \delta(s,u) + 1 \geq \delta(s,v)$
 - Conf. Lema 5.1, $v \in Q \Leftrightarrow v \in R(s)$ și $dist(s,u) \geq \delta(s,u)$.
 - Dacă $v \notin R(s)$, $dist(s,v) = \delta(s,v) = \infty$



BFS – Proprietăți (III)

- **Lema 5.4.** După orice execuție a ciclului principal al BFS, Q conține v_1, v_2, \dots, v_p ai:
 - $\text{Prop}(Q) = \text{dist}(s, v_1) \leq \text{dist}(s, v_2) \leq \dots \leq \text{dist}(s, v_p) \leq \text{dist}(s, v_1) + 1$
 - \Rightarrow la un moment dat în 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 invarianta $\text{Prop}(Q)$ la inserare și eliminare de elemente în/din Q.)
 - Fie $u = v_1 = \text{top}(Q)$
 - **Inserție:** fie v succ lui u și $\text{cul}(v) = \text{alb}$ \rightarrow coada devine: $v_1, v_2, \dots, v_p, v_{p+1} = v$. $\text{dist}(s, v) = \text{dist}(s, v_1) + 1 \rightarrow \text{dist}(s, v_1) \leq \text{dist}(s, v_2) \leq \dots \leq \text{dist}(s, v_p) \leq \text{dist}(s, v_{p+1}) \leq \text{dist}(s, v_1) + 1$
 - **Eliminare:** $\text{dist}(s, v_1) \leq \text{dist}(s, v_2) \rightarrow \text{dist}(s, v_1) + 1 \leq \text{dist}(s, v_2) + 1 \rightarrow$ coada devine: $v_2, \dots, v_p \rightarrow \text{dist}(s, v_2) \leq \dots \leq \text{dist}(s, v_p) \leq \text{dist}(s, v_1) + 1 \leq \text{dist}(s, v_2) + 1$
 - Cum relația este respectată, înseamnă că $\text{Prop}(Q)$ este **adevărată**.



BFS – Proprietăți (IV)

- **Corolar**

- $d(u) = \text{momentul în care nodul } u \text{ este inserat în coada Q. Atunci:}$

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

- **Teorema 5.1. BFS este corect și după terminare $\delta(s,u) = \text{dist}(s,u)$, $\forall u \in V$.**

- Utilizăm notația $V_k = \{ u \in V \mid \delta(s,u) = k \}$
 - Dem prin inducție $P(k) = \{ u \in V_k \mid \delta(s,u) = \text{dist}(s,u) \ \&\& \ (k > 0 \Rightarrow \pi(u) \in V_{k-1}) \ \&\& \ (k = 0 \Rightarrow \pi(u) = \text{null}) \}$
 - Caz de bază: $k = 0$, $V_0 = \{s\}$, $\delta(s,s) = \text{dist}(s,s) = 0$ și $\pi(s) = \text{null}$ Adevărat



BFS – Proprietăți (V)

- | Pas de inducție: $k > 0$, $P(k)$ adevărat $\rightarrow P(k+1)$ adevărat
- | • Alegem la întâmplare un nod $v \in V_{k+1} \rightarrow \delta(s,v) = k+1$ și fie $u \in V_k$ predecesorul lui v pe drumul cel mai scurt $s..v$.
- | • Caz 1: u este descoperit după v
 - $\text{dist}(s,v) \leq \text{dist}(s,u)$ (Corolar 5.1) iar $\text{dist}(s,u) = \delta(s,u) \rightarrow \text{dist}(s,v) \leq k$
 - $\text{dist}(s,v) \geq \delta(s,v) = k+1$ (Lema 5.3)
- | • Caz 2: u este descoperit înainte de v și v e descoperit pe arcul (u,v)
 - $\text{dist}(s,v) = \text{dist}(s,u) + 1$ și $\pi(v) = u$, $u \in V_k \rightarrow P(k+1)$ este adevărat
- | • Caz 3: u este descoperit înainte de v și v NU e descoperit pe arcul (u,v) . Fie $z \neq u$, a.î. v e descoperit pe arcul (z,v)
 - a) $d(u) < d(z) < d(v) \rightarrow$ cul(v) = alb la $d(u) \rightarrow v$ e descoperit pe arcul (u,v)
 - b) $d(z) < d(u) \rightarrow \text{dist}(s,z) \leq \text{dist}(s,u) = \delta(s,u) \rightarrow \text{dist}(s,z) \leq k$
 $\text{dist}(s,v) = \text{dist}(s,z) + 1$ și $k + 1 = \delta(s,v) \leq \text{dist}(s,v) \rightarrow \text{dist}(s,z) \geq k$
 $\rightarrow \text{dist}(s,z) = k \rightarrow \text{dist}(s,v) = k + 1 = \delta(s,v) \rightarrow$ drumul $s..z..v$ e optim
 $\rightarrow z \in V_k \rightarrow \pi(v) = z \in V_k \rightarrow P(k+1)$ este adevărat

Complexitate? Optimalitate? Completitudine?

BFS – Complexitate și Optimalitate

Complexitate:
 $O(n+m)$

n = număr noduri

m = număr muchii

Optimalitate: DA

Parcurge tot graful? NU

Parcuregere în adâncime (DFS)

- Nu mai avem nod de start, nodurile fiind parcuse 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

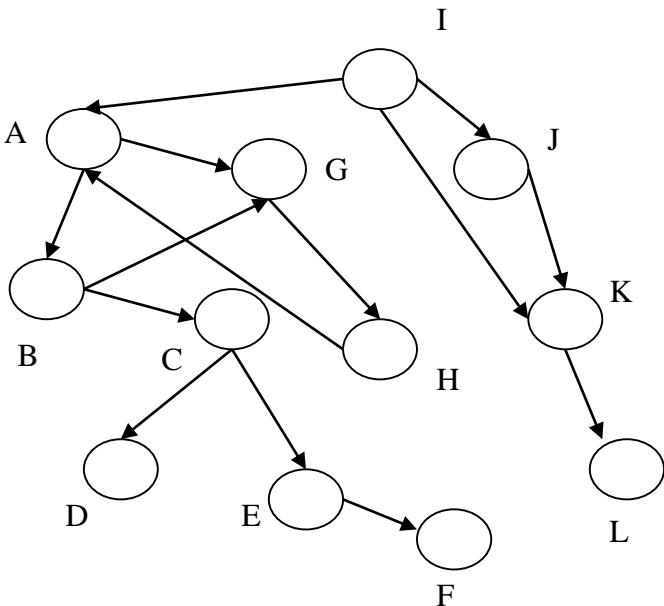
- Folosește o **stiva (LIFO)** pentru a reține nodurile ce trebuie prelucrate
 - În implementările uzuale, stiva este rareori folosită explicit;
 - Se apelează la recursivitate pentru a simula stiva.
- Folosește **o variabilă globală *timp*** pe baza căreia **se calculează** **timpii de descoperire și 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)
 - $V = \text{noduri}(G)$
 - **Pentru fiecare** nod u ($u \in V$)
 - $c(u) = \text{alb}; p(u) = \text{null}$; // inițializare structură date
 - $\text{timp} = 0$; // reține distanța de la rădăcina arborelui DFS pană la nodul curent
 - **Pentru fiecare** nod u ($u \in V$)
 - Dacă $c(u)$ este alb
 - **Atunci** $\text{explorare}(u)$; // explorez nodul
- $\text{explorare}(u)$
 - $d(u) = ++ \text{timp}$; // timpul de descoperire al nodului u
 - $c(u) = \text{gri}$; // nod în curs de explorare
 - **Pentru fiecare** nod v ($v \in \text{succs}(u)$) // încerc sa prelucrez vecinii
 - Dacă $c(v)$ este alb
 - **Atunci** $\{p(v) = u; \text{explorare}(v)\}$ // dacă nu au fost prelucrați deja
 - $c(u) = \text{negru}$; // am terminat de explorat nodul u
 - $f(u) = ++ \text{timp}$; // timpul de finalizare al nodului u



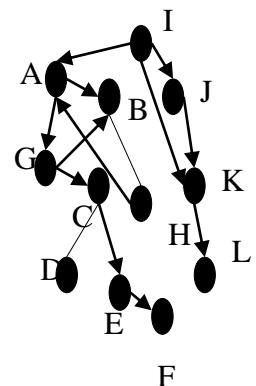
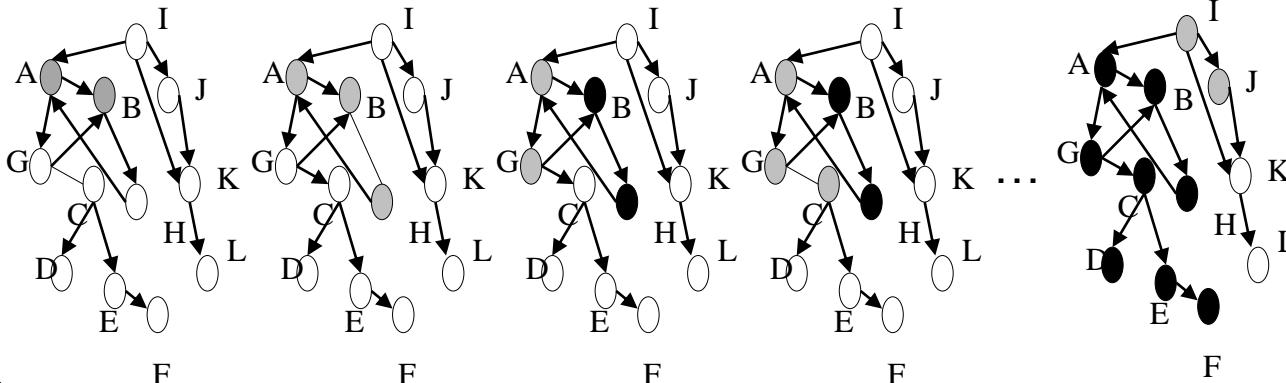
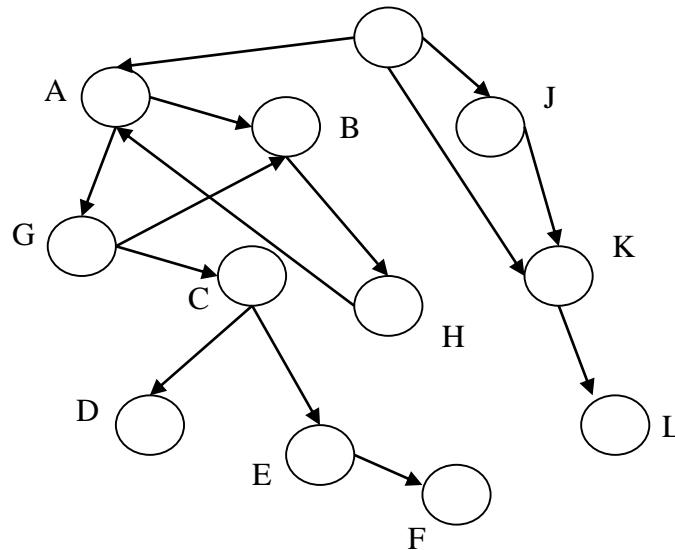
DFS – Exemplu



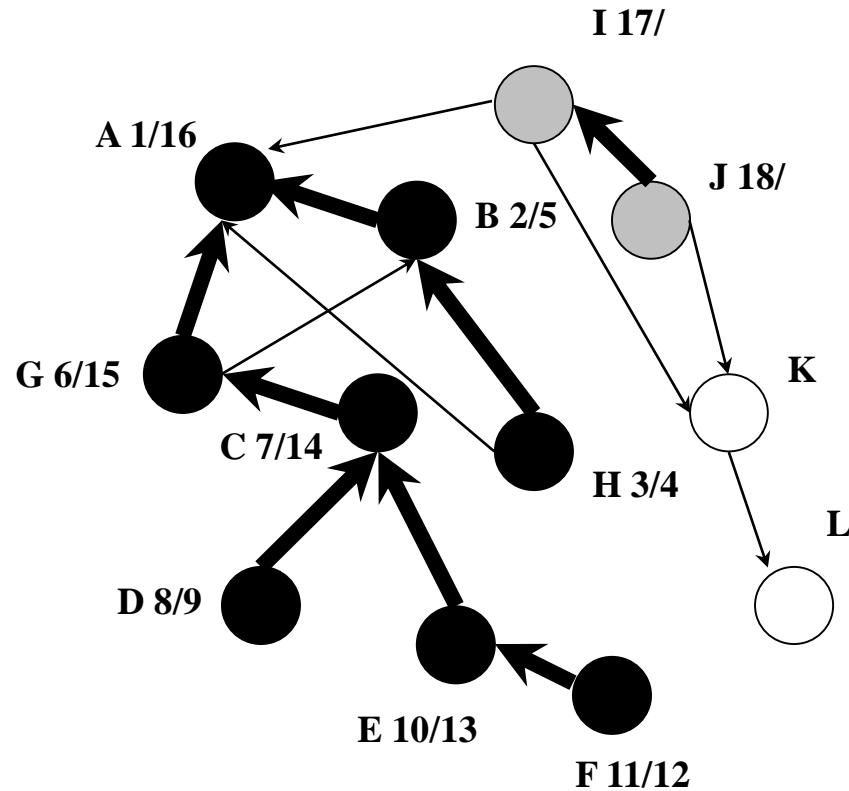
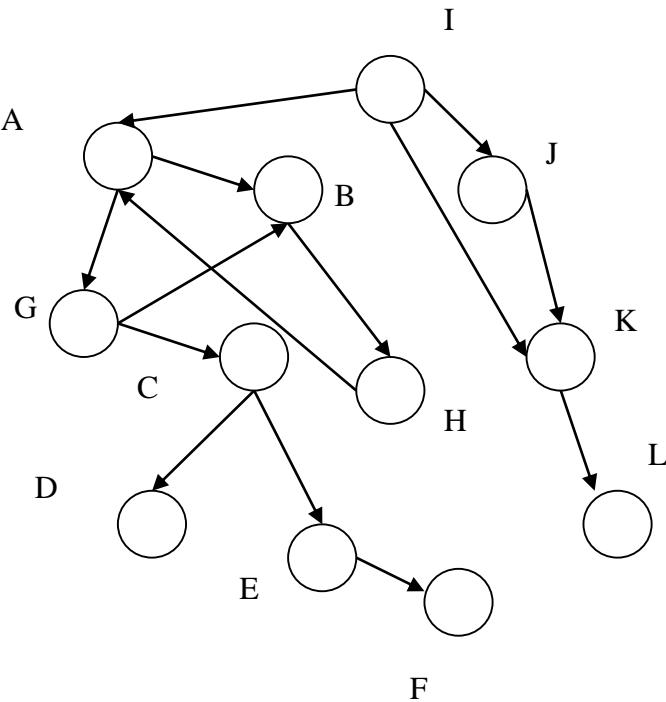
- **DFS(G)**
 - $V = \text{noduri}(G)$
 - **Pentru fiecare** nod u ($u \in V$)
 - $c(u) = \text{alb}; p(u) = \text{null}$; // inițializare structură date
 - $\text{timp} = 0$; // reține distanța de la rădăcina arborelui // DFS pană la nodul curent
 - **Pentru fiecare** nod u ($u \in V$)
 - **Dacă** $c(u)$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

DFS – Evoluția explorării

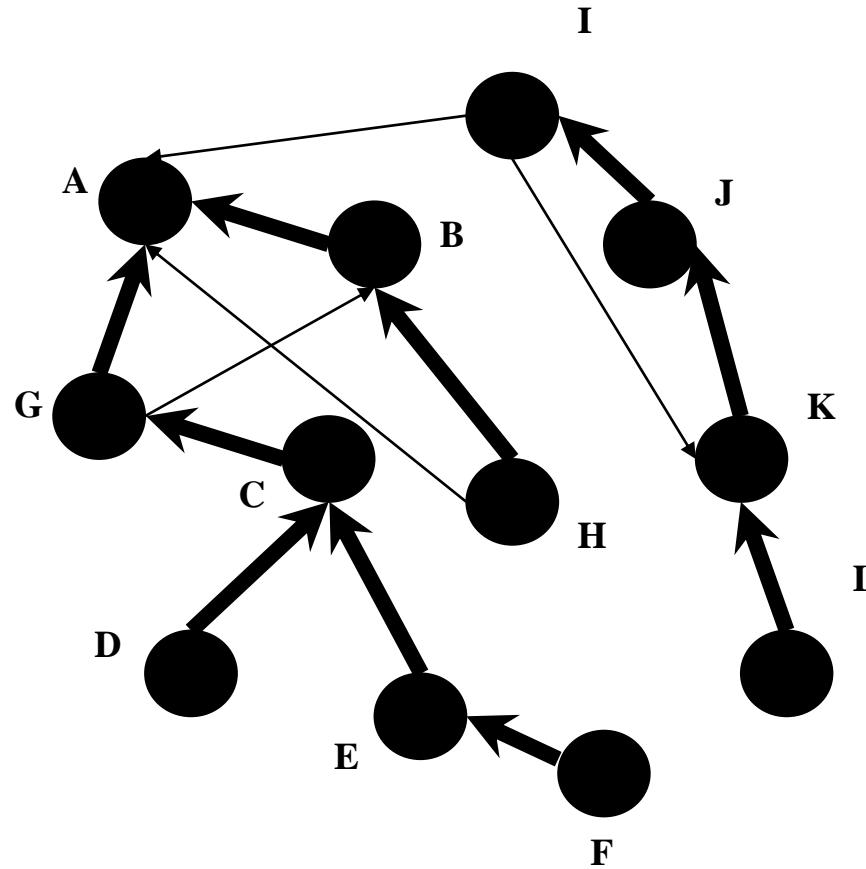
I



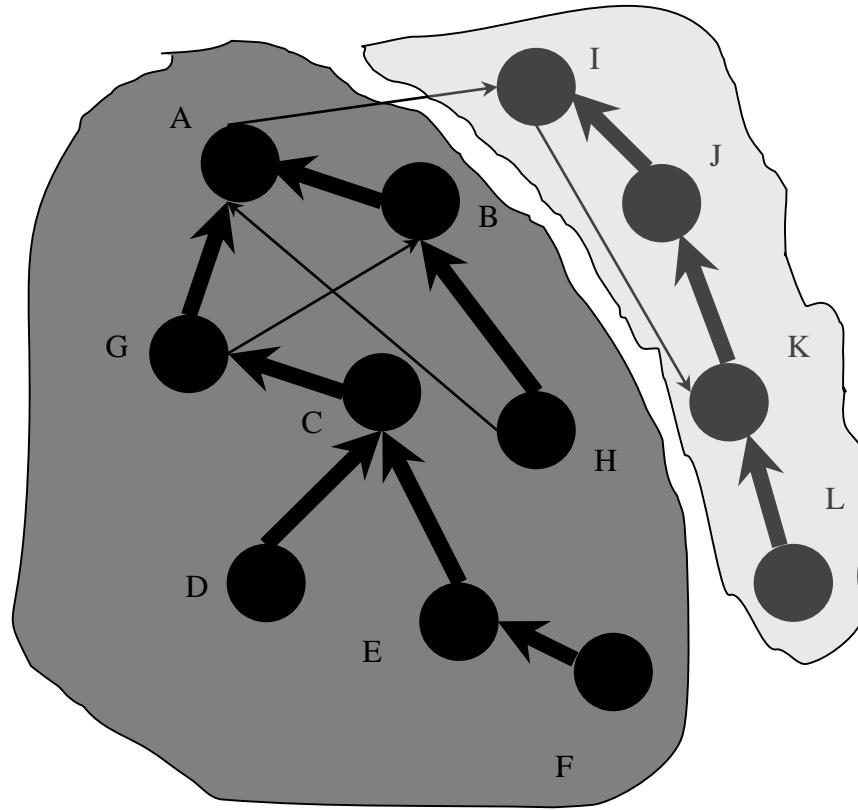
Calculul timpilor



Arborele de parcurgere în adâncime



DFS – Zone de explorare



DFS – Proprietăți (I)

- $I(u) = \text{intervalul de prelucrare al nodului } (d(u), f(u)).$
- **Lema 5.5.** $G = (V, E); u \in V$; pentru fiecare v descoperit de DFS pornind din u este construită o cale $v, p(v), p(p(v)), \dots, u$.
 - Fie calea $u = v_0v_1\dots v_n = v$. Dem prin inducție ca $\pi(v_i) = v_{i-1}$!
- **Teorema 5.2.** $G = (V, E); \text{DFS}(G)$ sparge graful G într-o pădure de arbori $\text{Arb}(G) = \{ \text{Arb}(u); p(u) = \text{null} \}$ unde $\text{Arb}(u) = (V(u), E(u))$;
 - $V(u) = \{ v \mid d(u) < d(v) < f(u) \} + \{u\}$;
 - $E(u) = \{ (v, z) \mid v, z \in V(u) \text{ && } p(z) = v \}$.
 - **Dem:** Conform algoritmului, se pot identifica noduri în ciclul principal sau din funcția de explorare. Dacă u e descoperit în ciclul principal, atunci \exists o cale către toți succesorii date de părinti $\rightarrow V(u) = \{ v \mid d(u) < d(v) < f(u) \}$ iar arcele sunt chiar cele ce desemnează părintii



DFS – Proprietăți (II)

- Teorema 5.3. Dacă $\text{DFS}(G)$ generează 1 singur arbore $\Rightarrow G$ este conex. (Reciproca este adevărată?)
- Teorema 5.4. Teorema parantezelor:
 - $\forall u, v$ avem $I(u) \cap I(v) = \emptyset$ 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 $u \rightarrow f(v) < f(u)$
 - 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:**

- $G = (V, E)$; $\text{Arb}(u)$; v este descendenter al lui u în $\text{Arb}(u) \Leftrightarrow$ la momentul $d(u)$ există o cale numai cu noduri albe $u..v$.
- **Demonstrație prin inducție!**
- $v \in V(u) \rightarrow$ la momentul $d(u)$ există o cale numai cu noduri albe $u..v$
 - Dacă $v \in V(u) \rightarrow \exists$ o cale unică $v .. \alpha .. u$ de pointeri π . Fie un nod oarecare z din calea α . \rightarrow (Teorema 5.5) $d(u) < d(z) < f(z) < f(u) \rightarrow$ la $d(u)$, $c(z) =$ alb și cum z a fost ales la întămplare \rightarrow toate nodurile de pe calea α sunt albe la $d(u)$.
- La momentul $d(u)$ există o cale numai cu noduri albe $u..v \rightarrow v \in V(u)$
 - Fie $u = v_0v_1\dots v_p = v$ o cale din G , a.î. La $d(u)$ avem $c(v_i) =$ alb, $i = 0, p$. Dem prin inducție după i că v_i este descendenter al lui u în $\text{Arb}(u)$.
 - Caz de bază: $d(u) < d(v_1) < f(u) \rightarrow$ (Teorema 5.5) v_1 descendenter al lui u **Adevărat**
 - Pas inducție: v_i descendenter al lui $u \rightarrow v_{i+1}$ descendenter al lui u
 - v_i descendenter al lui $u \rightarrow d(u) < d(v_i) < f(v_i) < f(u)$. Cum v_{i+1} este alb la $d(u)$ și este succesorul lui $v_i \rightarrow v_{i+1}$ este descoperit după $d(u)$, dar înainte de $f(v_i)$. $\rightarrow d(u) < d(v_{i+1}) < f(v_i) < f(u) \rightarrow v_{i+1}$ descendenter al lui u

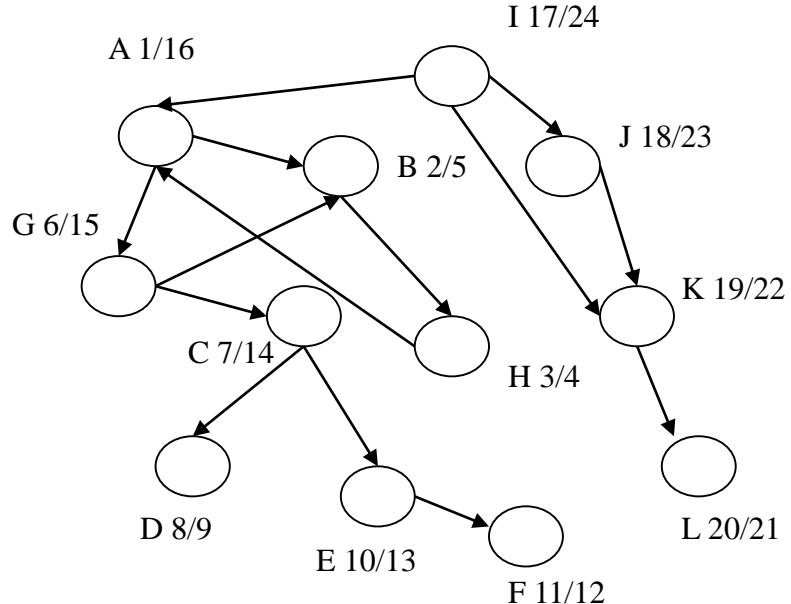
Clasificări ale arcelor grafului (I)

- 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)

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 (IV)

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



DFS – Proprietăți (V)

- **Teorema 5.8.** $G =$ graf orientat; G ciclic \Leftrightarrow în timpul execuției DFS găsim arce inverse.
 - Dem prin exploatarea proprietăților de ciclu și de arc invers!
 - G ciclic \rightarrow DFS descoperă arce inverse
 - Fie un ciclu și u primul nod din ciclu descoperit de DFS. Atunci $\exists u..v$ și (v,u) arcul care închide ciclul. $d(u) < d(v)$ (ipoteză) iar $u..v$ conține doar noduri albe \rightarrow (Teorema drumurilor albe & Teorema 5.5) $d(u) < d(v) < f(v) < f(u) \rightarrow$ în intervalul $(d(v), f(v))$ când se explorează (v,u) $c(u) = c(v) =$ gri \rightarrow arc invers
 - DFS descoperă arce inverse $\rightarrow G$ ciclic
 - Fie (v,u) arc invers $\rightarrow d(u) < d(v) < f(v) < f(u) \rightarrow$ (Teorema 5.5) $\rightarrow v$ este descendenta lui u în $\text{Arb}(u) \rightarrow \exists$ calea $u..v$ care închide ciclul

DFS – Complexitate și Optimalitate

Complexitate:

$$O(n+m)$$

n = număr noduri

m = număr muchii

Optimalitate: NU

Parcurge tot graful? DA

ÎNTREBĂRI?

Proiectarea Algoritmilor

Curs 7 – Parcuregere în adâncime
(DFS), Sortare Topologică &
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 + Componente Tare Conexe
- http://en.wikipedia.org/wiki/Tarjan%27s_stongly_connected_components_algorithm

Parcuregere în adâncime (DFS)

- Nu mai avem nod de start, nodurile fiind parcuse **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.

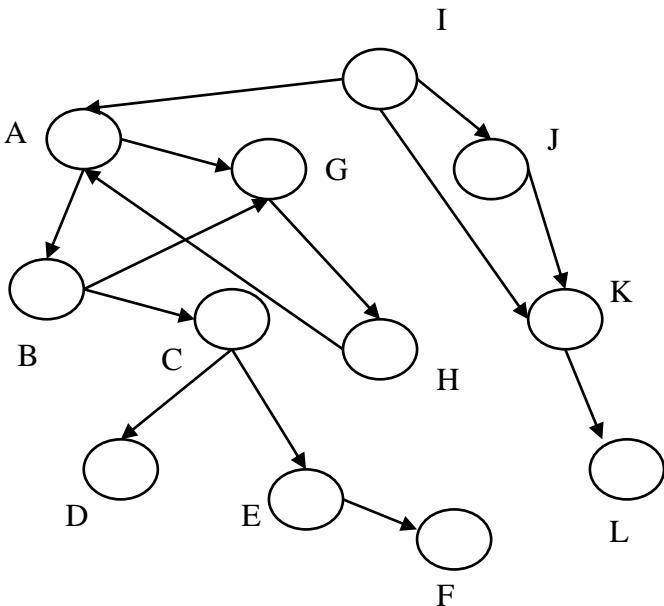
DFS – Structura de date

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

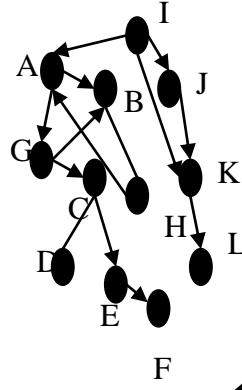
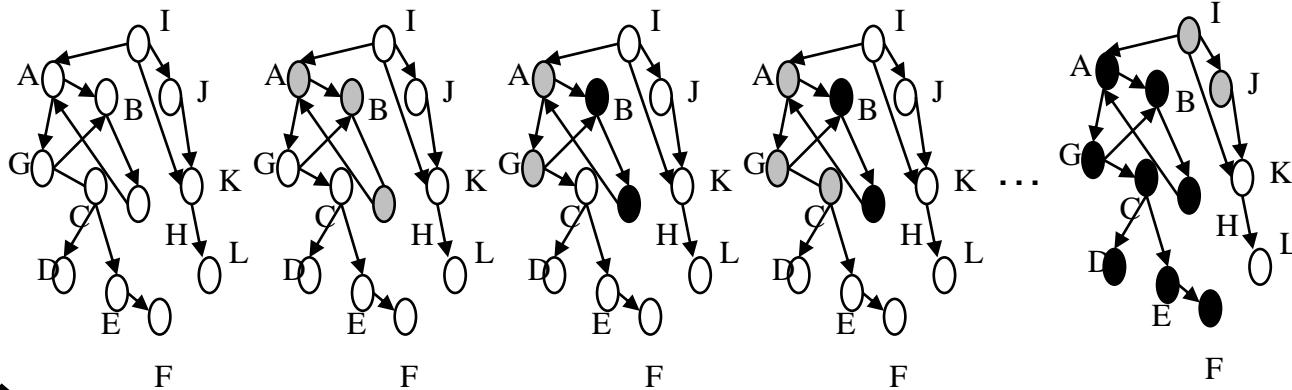
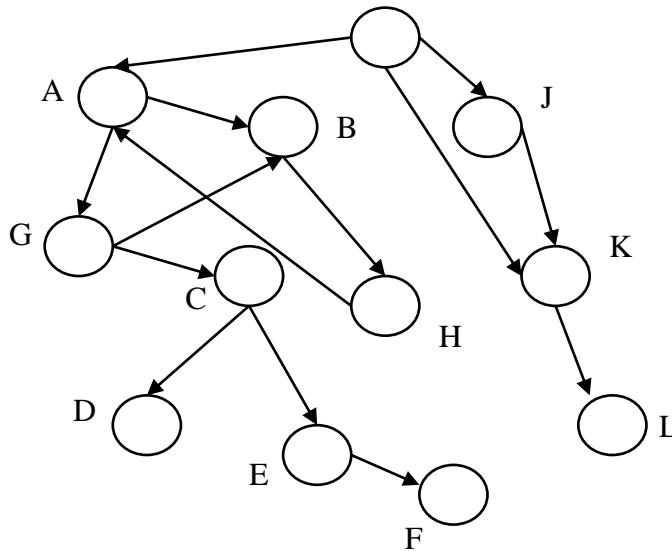
DFS – Exemplu



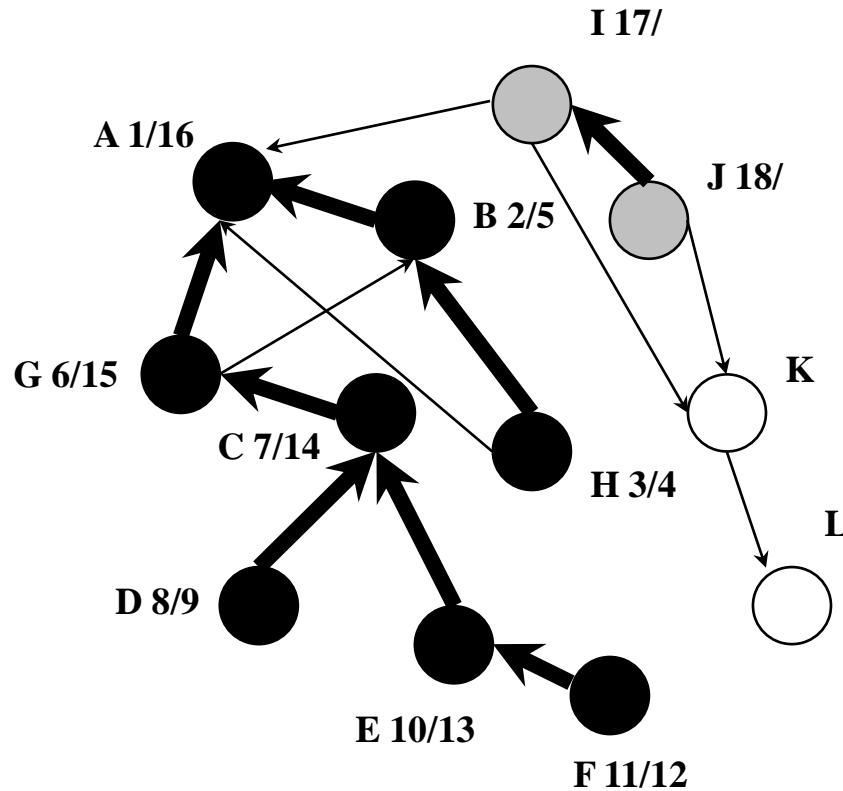
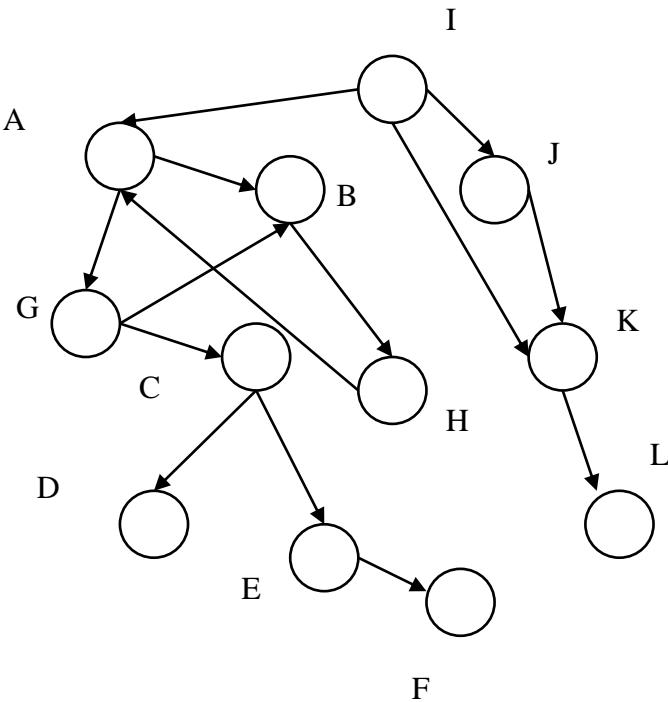
- **DFS(G)**
 - $V = \text{noduri}(G)$
 - **Pentru fiecare** nod u ($u \in V$)
 - $c(u) = \text{alb}; p(u) = \text{null}$; // inițializare structură date
 - $\text{timp} = 0$; // reține distanța de la rădăcina arborelui // DFS pană la nodul curent
 - **Pentru fiecare** nod u ($u \in V$)
 - **Dacă** $c(u)$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

DFS – Evoluția explorării

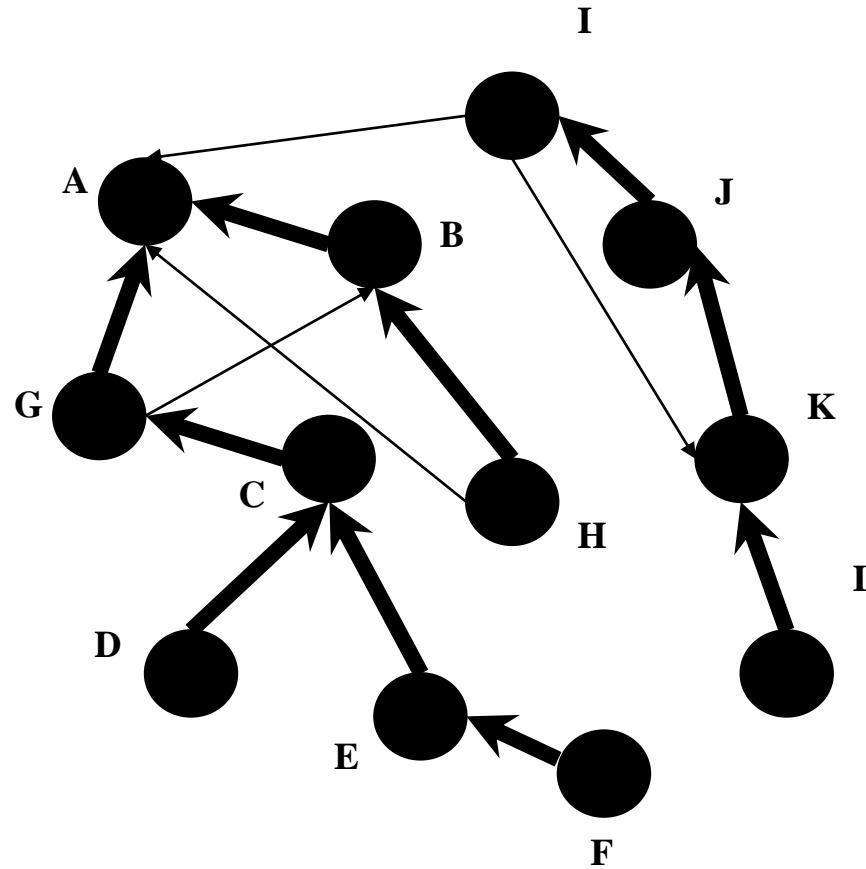
I



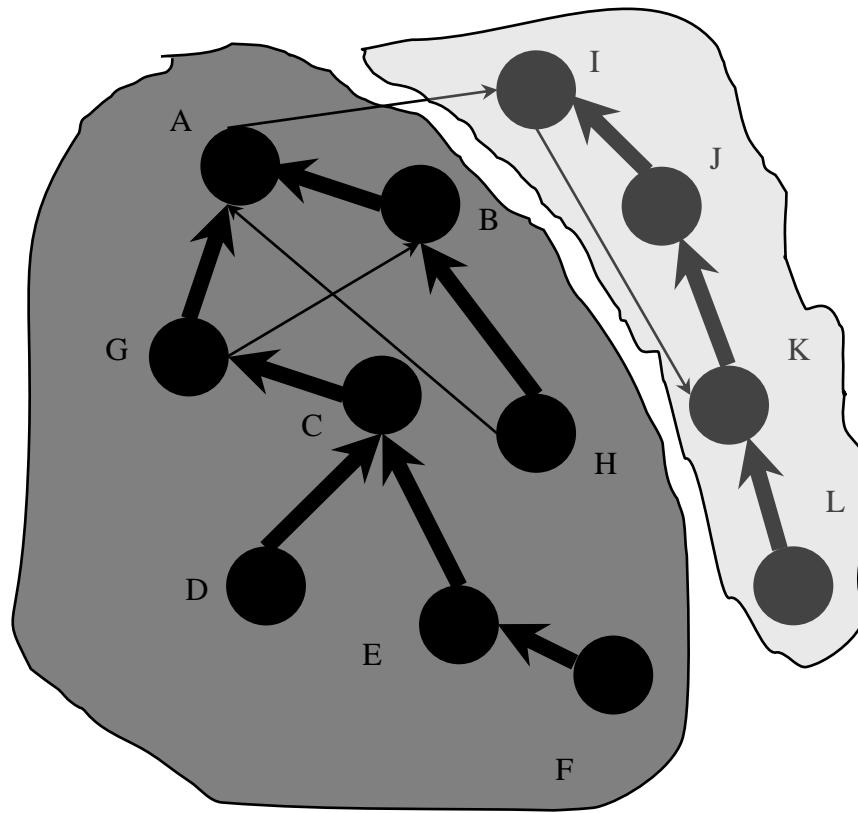
Calculul timpilor



Arborele de parcurgere în adâncime



DFS – Zone de explorare



DFS – Proprietăți (I)

- $I(u) = \text{intervalul de prelucrare al nodului } (d(u), f(u))$.
- **Lema 5.5.** $G = (V, E)$; $u \in V$; pentru fiecare v descoperit de DFS pornind din u este construită o cale $v, p(v), p(p(v)), \dots, u$.
 - Fie calea $u = v_0 v_1 \dots v_n = v$. Dem prin inducție ca $\pi(v_i) = v_{i-1}$!
- **Teorema 5.2.** $G = (V, E)$; $\text{DFS}(G)$ sparge graful G într-o pădure de arbori $\text{Arb}(G) = \{ \text{Arb}(u); p(u) = \text{null} \}$ unde $\text{Arb}(u) = (V(u), E(u))$;
 - $V(u) = \{ v \mid d(u) < d(v) < f(u) \} + \{u\}$;
 - $E(u) = \{ (v, z) \mid v, z \in V(u) \text{ && } p(z) = v \}$.
 - **Dem:** Conform algoritmului, se pot identifica noduri în ciclul principal sau din funcția de explorare. Dacă u e descoperit în ciclul principal, atunci \exists o cale către toți succesorii dată de părinți $\rightarrow V(u) = \{ v \mid d(u) < d(v) < f(u) \}$ iar arcele sunt chiar cele ce desemnează părinții

DFS – Proprietăți (II)

- Teorema 5.3. Dacă $\text{DFS}(G)$ generează 1 singur arbore $\Rightarrow G$ 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:**
 - $G = (V, E)$; $\text{Arb}(u)$; v este descendant al lui u în $\text{Arb}(u) \Leftrightarrow$ la momentul $d(u)$ există o cale numai cu noduri albe $u..v$.
 - **Demonstrație prin inducție!**
 - $v \in V(u) \rightarrow$ la momentul $d(u)$ există o cale numai cu noduri albe $u..v$
 - Dacă $v \in V(u) \rightarrow \exists$ o cale unică $v.. \alpha..u$ de pointeri π . Fie un nod oarecare z din calea α . \rightarrow (Teorema 5.5) $d(u) < d(z) < f(z) < f(u) \rightarrow$ la $d(u)$, $c(z) =$ alb și cum z a fost ales la întămplare \rightarrow toate nodurile de pe calea α sunt albe la $d(u)$.
 - La momentul $d(u)$ există o cale numai cu noduri albe $u..v \rightarrow v \in V(u)$
 - Fie $u = v_0v_1...v_p = v$ 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 descendantul lui u în $\text{Arb}(u)$.
 - Caz de bază: $d(u) < d(v_1) < f(u) \rightarrow$ (Teorema 5.5) v_1 descendant al lui u **Adevărat**
 - Pas inducție: v_i descendant al lui $u \rightarrow v_{i+1}$ descendant al lui u
 - v_i descendant al lui $u \rightarrow d(u) < d(v_i) < f(v_i) < f(u)$. Cum v_{i+1} este alb la $d(u)$ și este succesorul lui $v_i \rightarrow v_i \rightarrow v_{i+1}$ este descoperit după $d(u)$, dar înainte de $f(v_i)$. $\rightarrow d(u) < d(v_{i+1}) < f(v_i) < f(u) \rightarrow v_{i+1}$ descendant al lui u

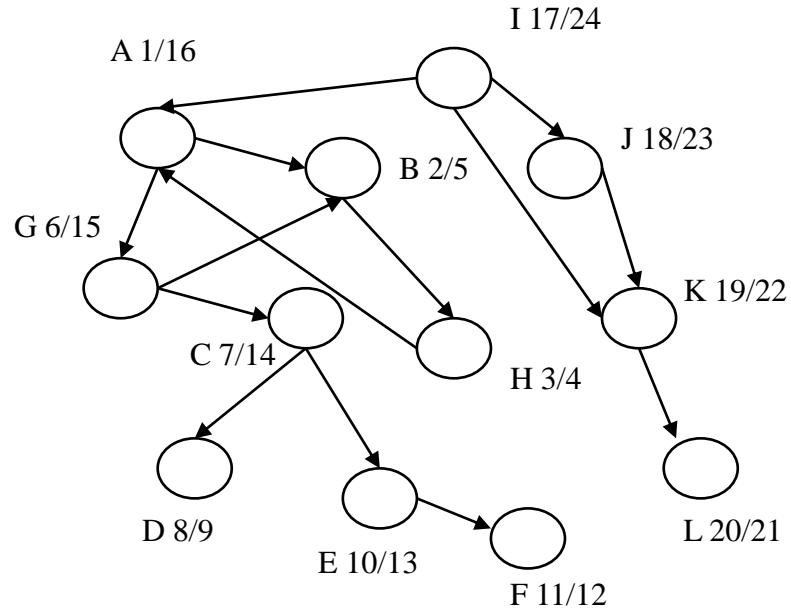
Clasificări ale arcelor grafului (I)

- 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)

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 (IV)

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

DFS – Proprietăți (V)

- **Teorema 5.8.** $G =$ graf orientat; G ciclic \Leftrightarrow în timpul execuției DFS găsim arce inverse.
 - Dem prin exploatarea proprietăților de ciclu și de arc invers!
 - G ciclic \rightarrow DFS descoperă arce inverse
 - Fie un ciclu și u primul nod din ciclu descoperit de DFS. Atunci $\exists u..v$ și (v,u) arcul care închide ciclul. $d(u) < d(v)$ (ipoteză) iar $u..v$ conține doar noduri albe \rightarrow (Teorema drumurilor albe & Teorema 5.5) $d(u) < d(v) < f(v) < f(u) \rightarrow$ în intervalul $(d(v), f(v))$ când se explorează (v,u) $c(u) = c(v) =$ gri \rightarrow arc invers
 - DFS descoperă arce inverse $\rightarrow G$ ciclic
 - Fie (v,u) arc invers $\rightarrow d(u) < d(v) < f(v) < f(u) \rightarrow$ (Teorema 5.5) $\rightarrow v$ este descendenta lui u în $\text{Arb}(u) \rightarrow \exists$ calea $u..v$ care închide ciclul

DFS – Complexitate și Optimalitate

Complexitate:
 $O(n+m)$

n = număr noduri

m = număr muchii

Optimalitate: NU

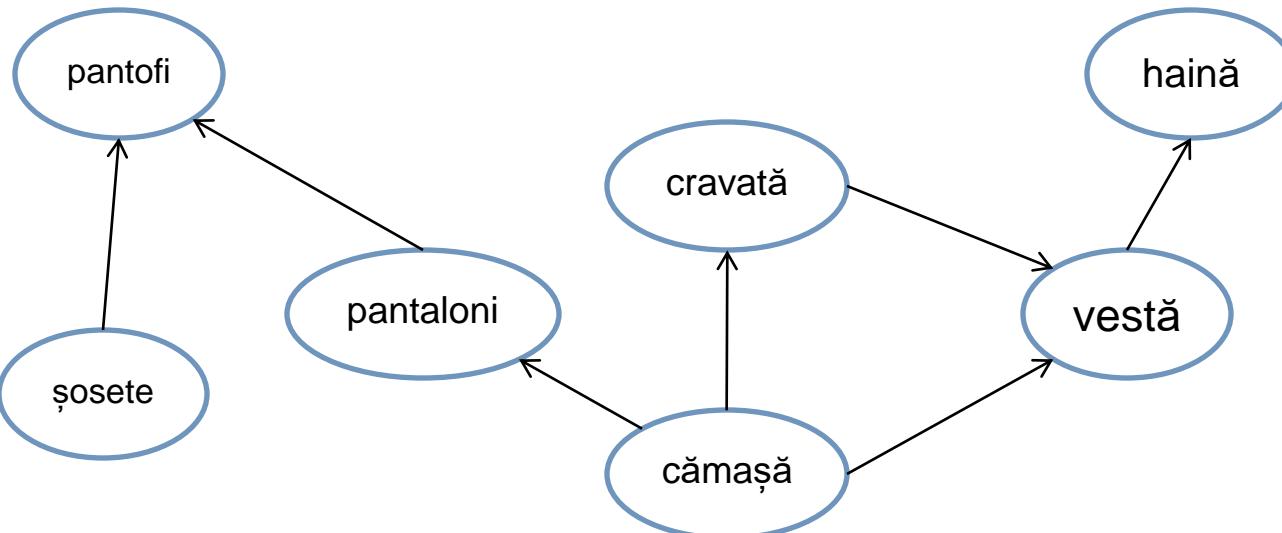
Parcurge tot graful? DA

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 α ($\alpha \subseteq A^*A$) 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 $L = <e_1, e_2 \dots e_n>$, cu proprietatea că $\forall i, j$, dacă $e_i \alpha e_j$, atunci $i < j$.

Sortare topologică - Exemplu

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



Sortare topologică

- $G = (V, E)$ orientat, **aciclic**.
- V_S – secvența de noduri a.î. $\forall (u, v) \in E$, avem $\text{index}(u) < \text{index}(v)$.
- **Scop:** Sortare_topologică($G \Rightarrow V_S$).
- **Idee bazată pe DFS:**
 - $G = (V, E)$ orientat, aciclic; la sfârșitul DFS avem $\forall (u, v) \in E$, $f(v) < f(u)$
 - \Rightarrow colectăm în V_S vârfurile în ordinea descrescătoare a timpilor f

Algoritm sortare topologică

- Sortare_topologică (G)
 - **Pentru fiecare** nod u ($u \in V$) $\{c(u) = \text{alb};\}$ // inițializări
 - $V_S = \emptyset;$
 - **Pentru fiecare** nod u ($u \in V$) // pentru fiecare componentă conexă
 - **Dacă** $c(v)$ este alb
 - $V_S = \text{explorează } (u, V_S)$ // prelucrez componenta conexă
 - **Întoarce** V_S
- Explorează (u, V_S)
 - $c(u) = \text{gri}$ // prelucrez nodul, deci îi actualizez culoarea
 - **Pentru fiecare** nod v ($v \in \text{succs}(u)$)
 - **Dacă** $c(v)$ este alb **atunci** $V_S = \text{explorează } (u, V_S)$ // recursivitate
 - **Dacă** $c(v)$ este gri **atunci** **Întoarce** Eroare: graf ciclic
 - $c(u) = \text{negru}$ // am terminat prelucrarea nodului
 - **Întoarce** $\text{cons}(u, V_S)$ // 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:

$$O(n+m)$$

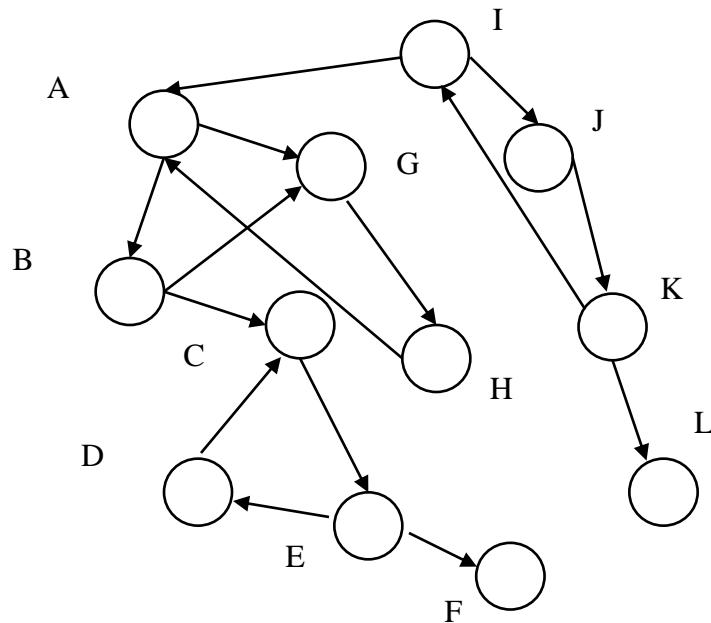
n = număr noduri

m = număr muchii

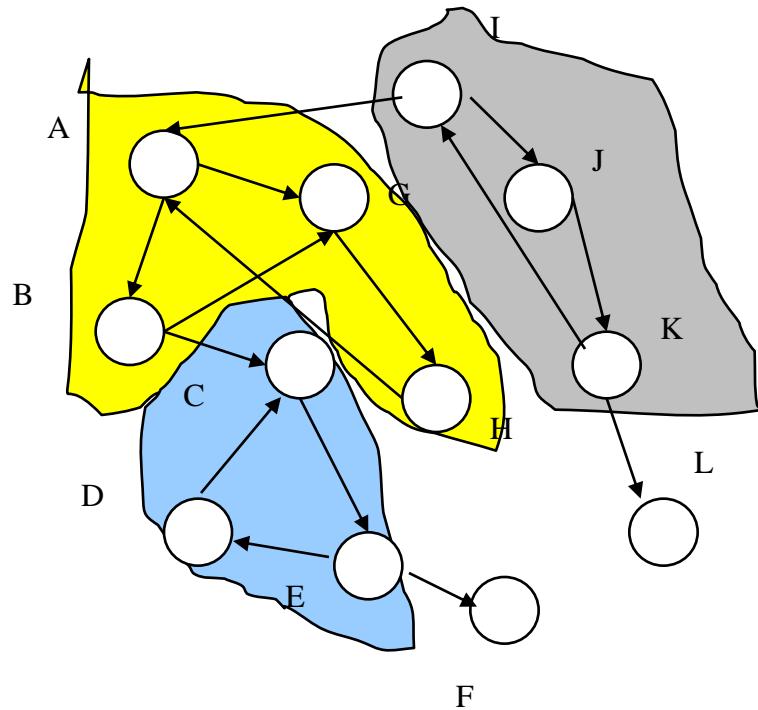
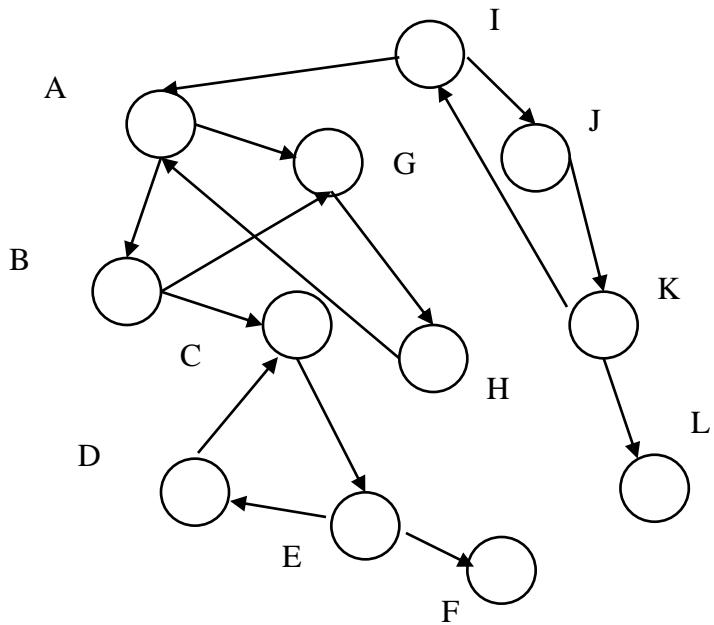
Componente Tare Conexe (CTC)

- **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, $\forall u, v \in V'$
 $\Rightarrow \forall u..v$ din G are noduri exclusiv în V'
 - **Dem:** $\forall z$ a.î. $u..z..v \Rightarrow z \in R(u)$ și $v \in R(z)$. Dar $u \in R(v)$
 $\Rightarrow z \in R(v) \Rightarrow v$ și z sunt în aceeași CTC.

Exemplu (I) – determinare CTC



Exemplu (II) – determinare CTC(2)

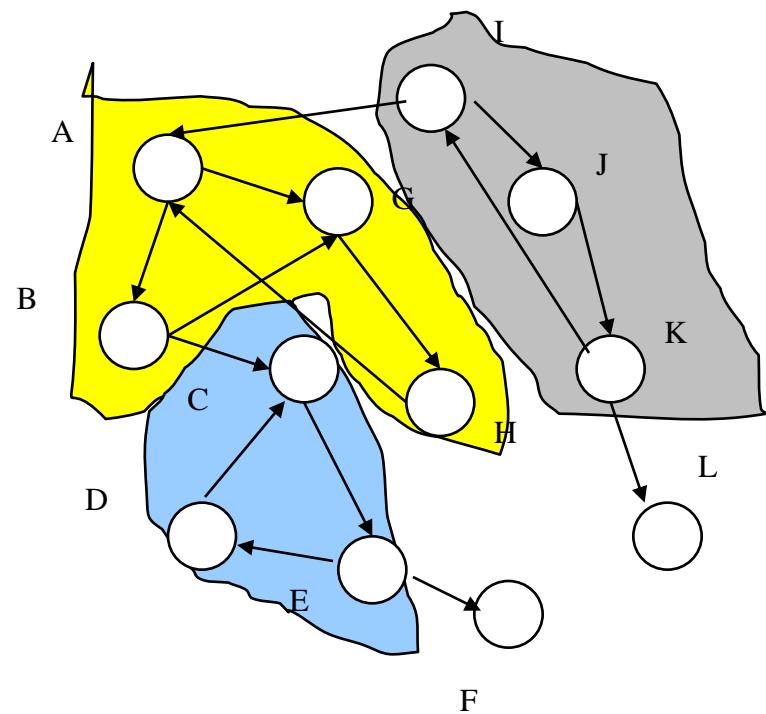
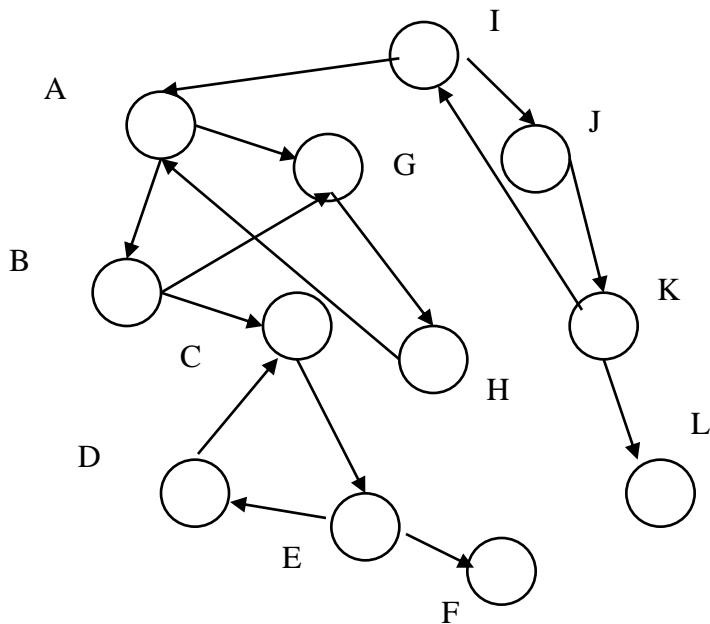


Componente Tare Conexe (CTC)

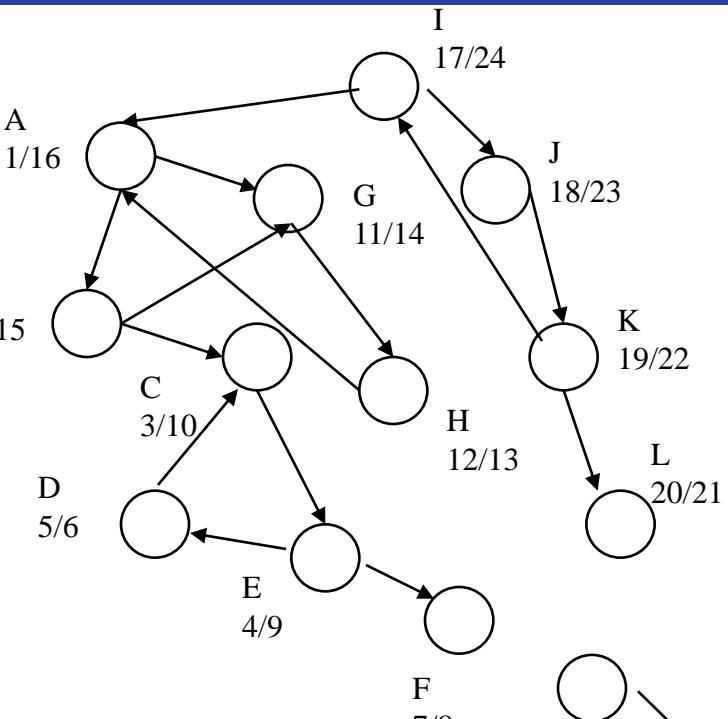
- **Teorema 5.10:** $G = (V, E)$ orientat, $G' = (V', E')$ o CTC a lui G . Toate nodurile $v \in V'$ sunt grupate in același $\text{Arb}(u)$ construit de $\text{DFS}(G)$, unde u este primul nod descoperit al componentei.
 - **Dem:** $\forall v \in V', v \neq u, \exists u..v$ drum cu noduri albe la momentul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

Exemplu (III) – DFS

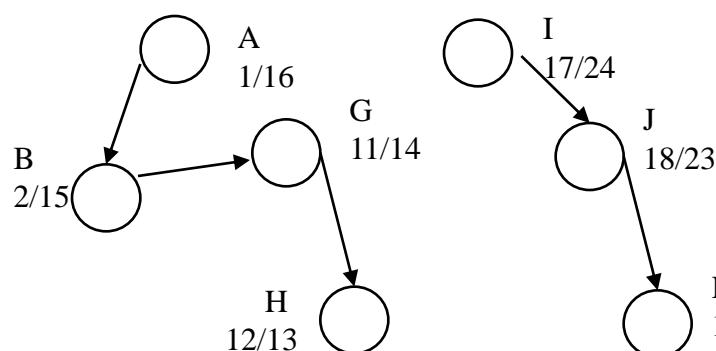
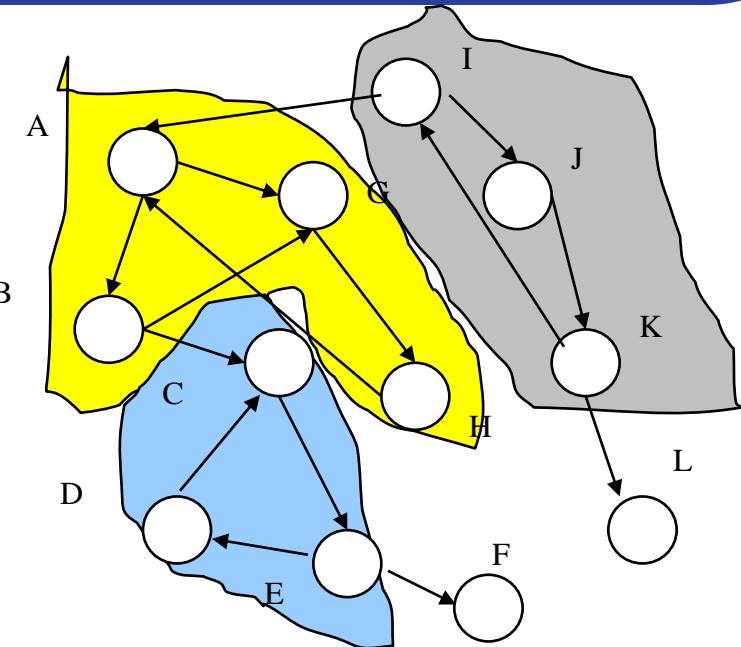
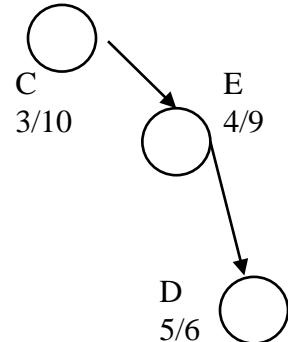
- Aplicare DFS pornind din primul nod al fiecărei CTC



Exemplu (IV) - DFS(2)



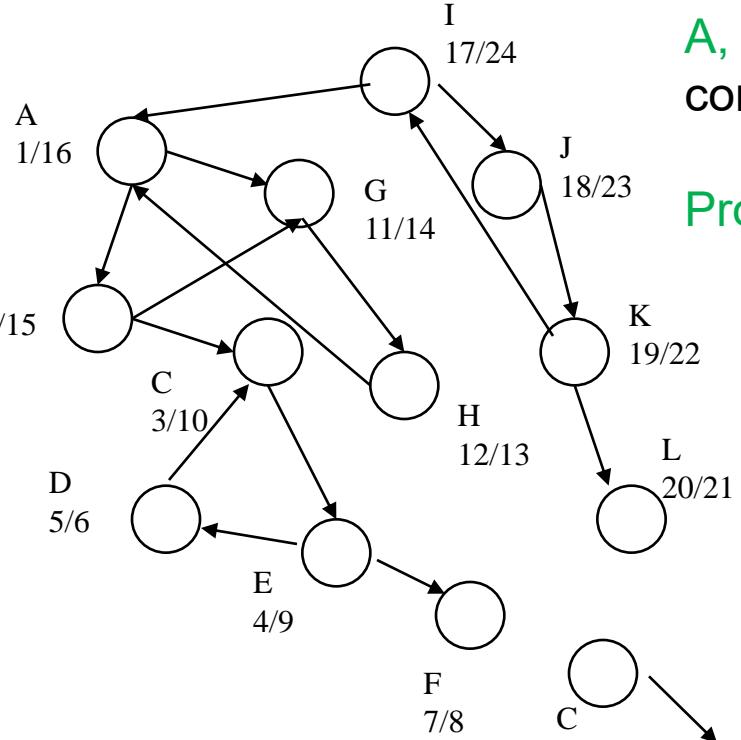
Conform Teoremei
nodurile din aceeași
CTC sunt grupate în
același arbore DFS!



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: $\Phi(u)$ satisfac 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 și v sunt în 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) – strămosi



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

Proprietățile strămoșilor:

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

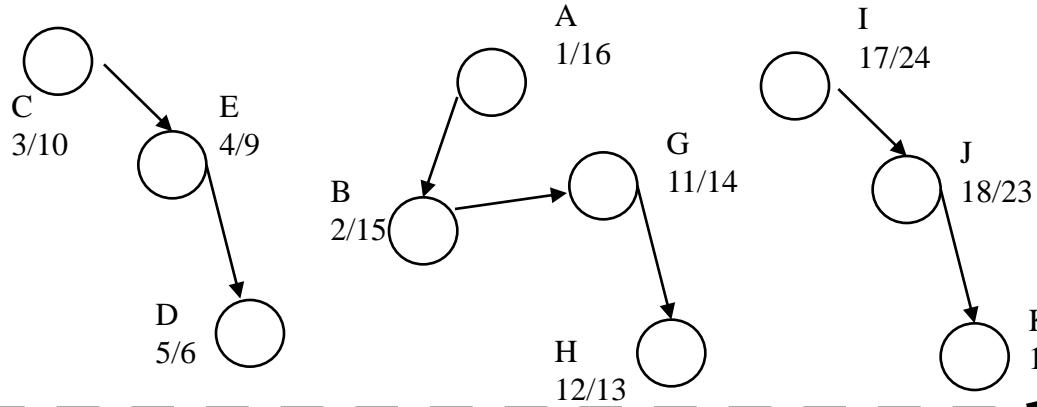
Ex: $f(B) \leq f(\Phi(B)) = f(A)$

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

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

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

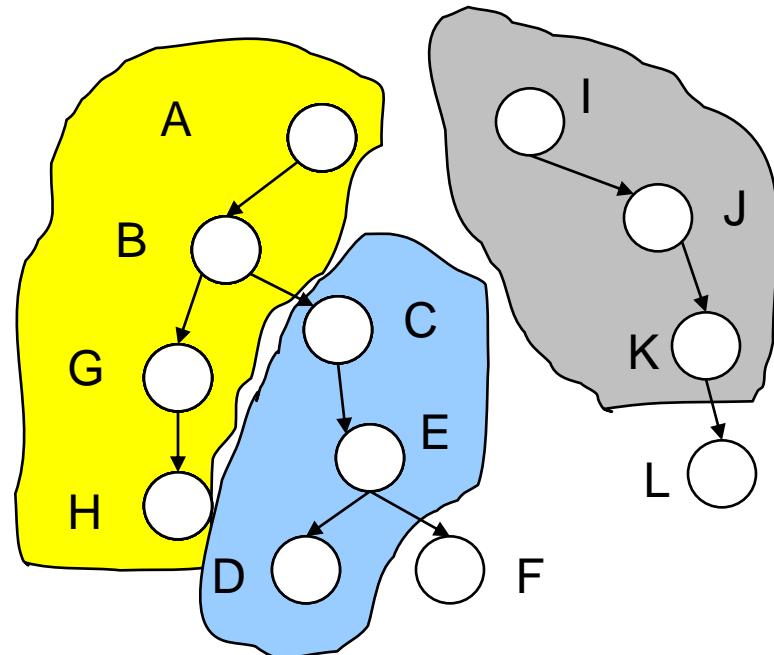
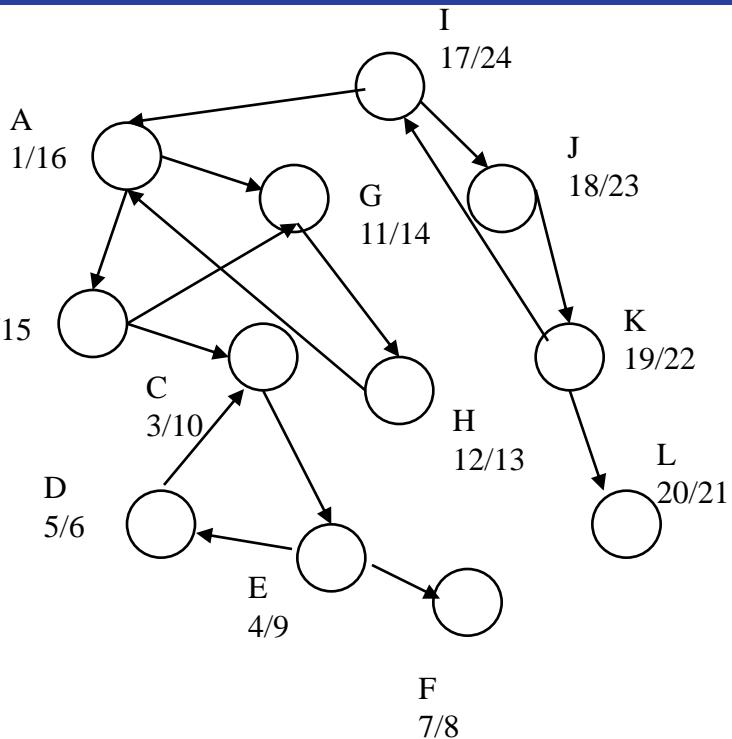
Ex: $\Phi(\Phi(E)) = \Phi(C) = C = \Phi(E)$



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 $\text{Arb}(\Phi(u))$ construit de DFS.**
 - **Dem:** prin considerarea tuturor cularilor 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 $\Rightarrow \Phi(u) = \Phi(v)$: $v \in R(u)$, $f(\Phi(v)) \leq f(\Phi(u))$ și $u \in R(v)$, $f(\Phi(u)) \leq f(\Phi(v)) \Rightarrow f(\Phi(u)) = f(\Phi(v)) \rightarrow \Phi(u) = \Phi(v)$
 - $\Phi(u) = \Phi(v) \Rightarrow \Phi(u) \in R(u) \Rightarrow u$ și $\Phi(u) \in$ aceleiași CTC și $\Phi(u) \in R(v) \Rightarrow v$ și $\Phi(u) \in$ aceleiași CTC
 $\Rightarrow u$ și $v \in$ aceleiași CTC

Exemplu (VI)

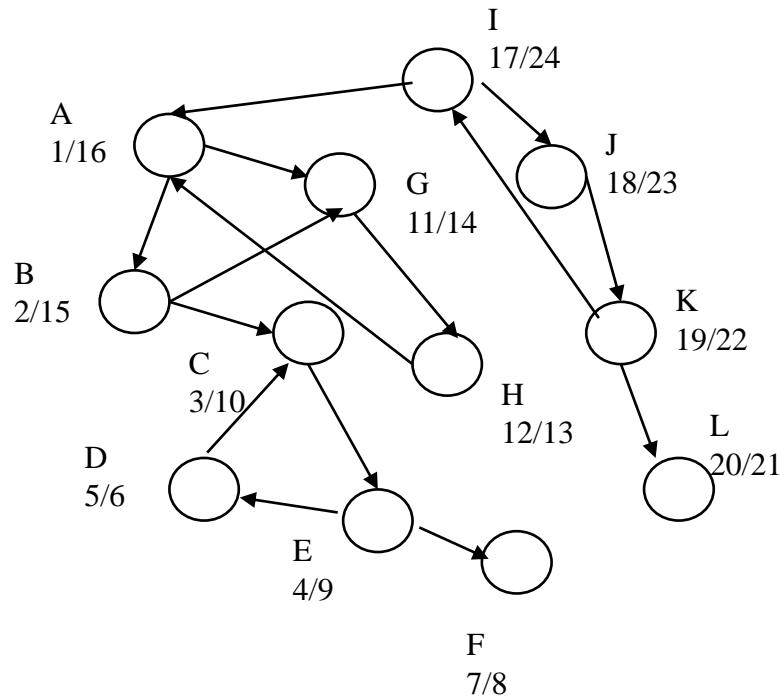


Primul nod dintr-o CTC descoperit prin DFS va avea drept copii în arborele generat de DFS toate elementele componentei conexe!

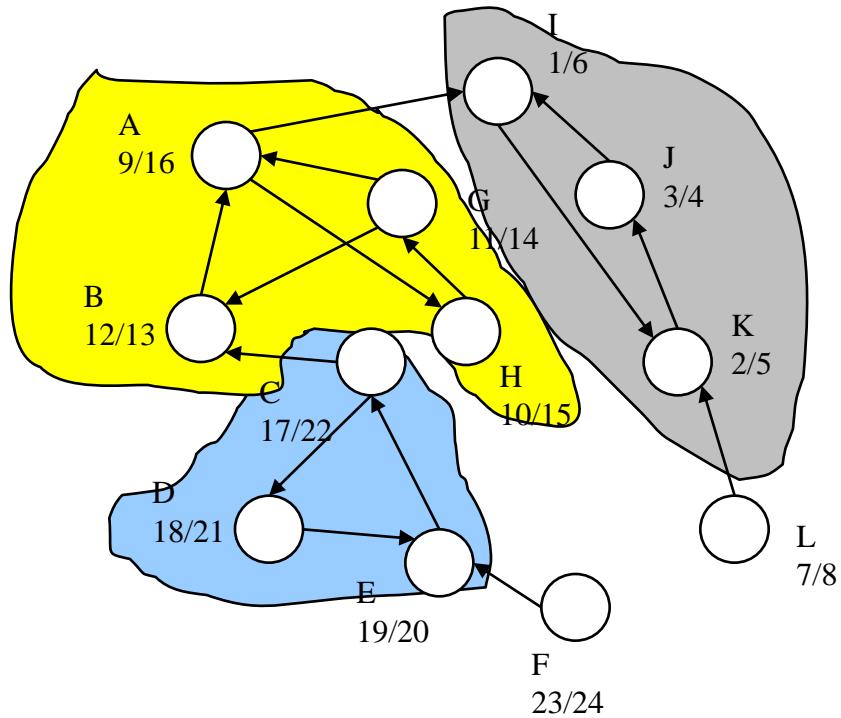
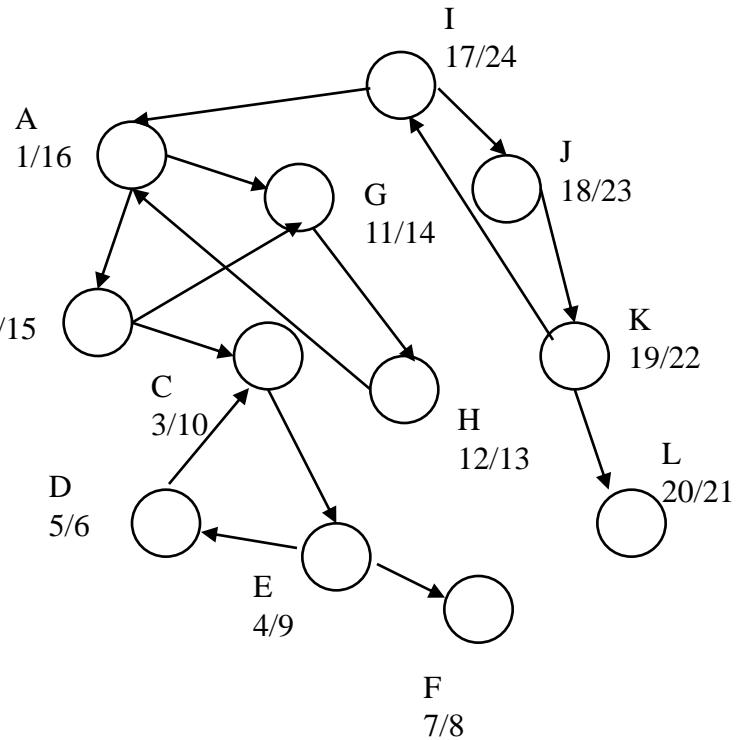
Componente Tare Conexe (CTC)

- 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 $\text{Arb}(u)$ și nu CTC $\Rightarrow \exists u..v$ și $\nexists v..u$.
 - \rightarrow DFS pe graful transpus, în ordinea inversă a timpilor de finalizare obținuți din DFS pe graful normal!

Exemplu (VII) – DFS (G^T)



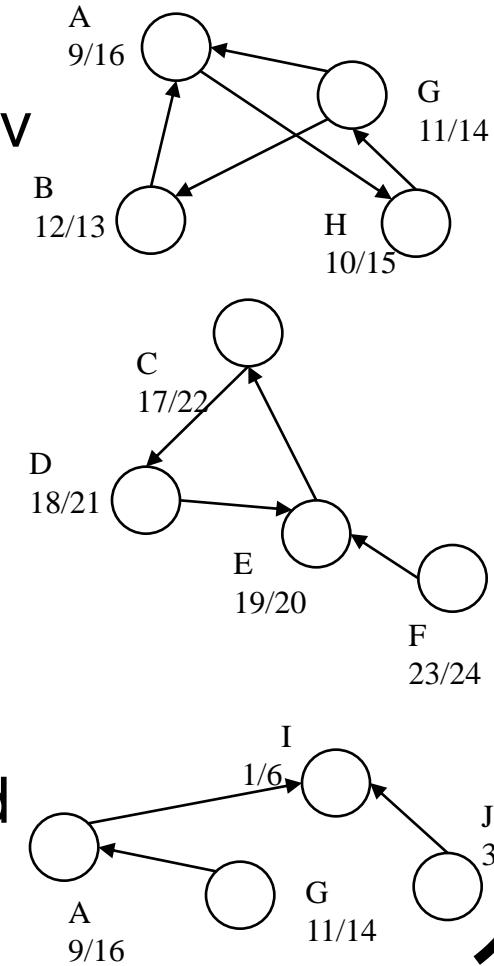
Exemplu (VIII) – DFS (G^T) (2)



Componente Tare Conexe (CTC)

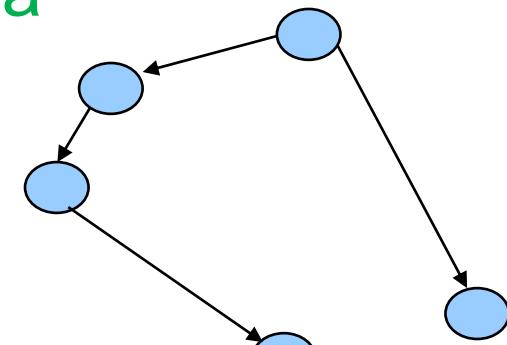
- Cazuri în $\text{DFS}(G^T)$:

- v este în CTC descoperita din $u \rightarrow v$ poate fi descoperit din u și în $\text{DFS}(G^T)$.
- $v \notin \text{CTC}$ dar $v \in \text{Arb}(u)$ în $\text{DFS}(G) \rightarrow$ nu va fi atins în $\text{DFS}(G^T)$ din u .
- $v \notin \text{CTC}$ dar $\exists v..u$ în $G \rightarrow f(v) > f(u)$ $\rightarrow v$ 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 inversă a timpilor de finalizare a strămoșilor fiecărei CTC!



Pseudocod algoritm CTC

- | • Algoritmul lui Kosaraju:
- | |
- | | • CTC(G)
 - $\text{DFS}(G)$
 - $G^T = \text{transpune}(G)$
 - $\text{DFS}(G^T)$ (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 $\text{DFS}(G^T)$.
- | |

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 nu arbori anteriori.
 - $P_n \rightarrow P_{n+1}$: Fie arborele T obținut în 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 ($\text{Arb}(r)$). 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 unică
- 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>

Proiectarea Algoritmilor

Curs 8 – Puncte de articulație,
Punți, Drumuri minime

Bibliografie

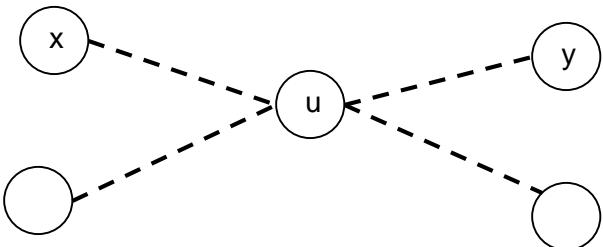
- | [1] Giumale – Introducere în Analiza Algoritmilor cap. 5.3, 5.4,
| 5.4.1
- | [2] Cormen – Introducere în Algoritmi cap. Heap-uri binomiale
(20), Heap-uri Fibonacci (21), Drumuri minime de sursă unică
- primele 2 subcapitole (25.1 și 25.2)
- | [3] R. Sedgewick, K. Wayne - Algorithms and Data Structures
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>

Obiective

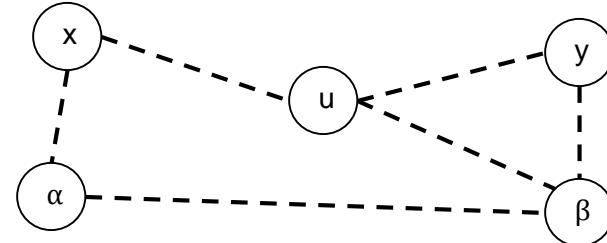
- “Descoperirea” algoritmilor de:
 - Identificare a punctelor de articulație;
 - Identificare a punțiilor;
 - Identificare a drumurilor de cost minim.
- Identificarea structurilor de date necesare pentru reducerea complexității acestor algoritmi.

Puncte de articulație. Def. Exemple

- Definiție: $G = (V, E)$ graf neorientat, $u \in V$.
 U este punct de articulație dacă $\exists x, y \in V$, $x \neq y$, $x \neq u$, $y \neq u$, a.î. $\forall x..y$ în G trece prin u .



Orice drum $x..y$ trece prin $u \rightarrow$ u este punct de articulație.



Exista $x..\alpha..y$ care nu trece prin $u \rightarrow$ u nu mai este punct de articulație!

Algoritm naiv de detectare a punctelor de articulație

- Elimină fiecare nod și verifică conectivitatea grafului rezultat:
 - Graf conex → nodul nu e punct de articulație.
 - Altfel → punct de articulație.

Complexitate?

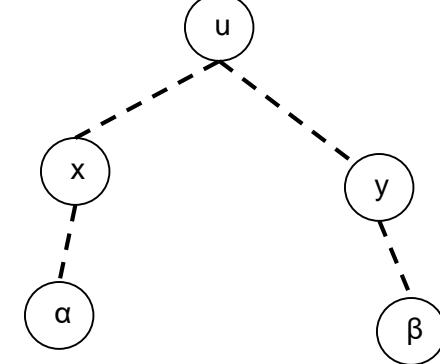
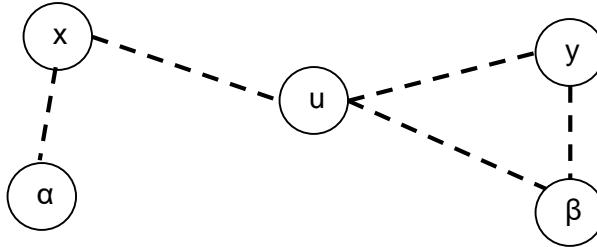
$O(V(V+E))$

Puncte de articulație. Teoremă

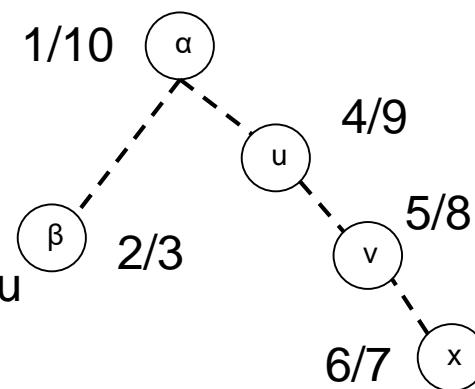
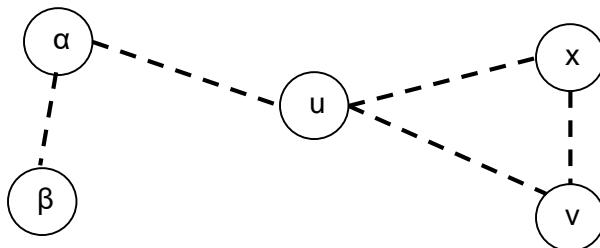
- **Teorema 5.15:** $G = (V, E)$, graf neorientat și $u \in V$. U este punct de articulație în G
 \Leftrightarrow în urma DFS-ului din u sunt satisfăcute unele proprietăți:
 - $p(u) = \text{null}$ și u domină cel puțin 2 subarbori;
 - $p(u) \neq \text{null}$ și $\exists v$ descendent al lui u în $\text{Arb}(u)$ a.i. $\forall x \in \text{Arb}(v)$ și $\forall (x, z)$ parcursă de DFS(G) avem $d(z) \geq d(u)$.

Situări posibile

- 1) $p(u) = \text{null}$ și u domina cel puțin 2 subbarburi:



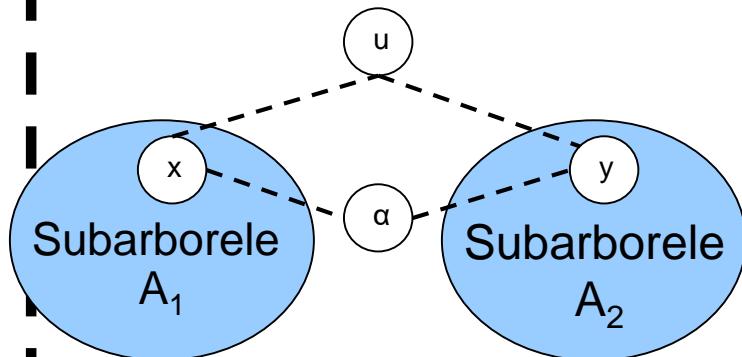
- 2) $p(u) \neq \text{null}$ și $\exists v$ descendent al lui u în $\text{Arb}(u)$ a.î.
 $\forall x \in \text{Arb}(v)$ și $\forall (x,z)$ parcursă de $\text{DFS}(G)$ $d(z) \geq d(u)$:



Pentru orice muchie din subarborele lui v nu există nici o **muchie înapoi** spre un nod descoperit înaintea lui u .

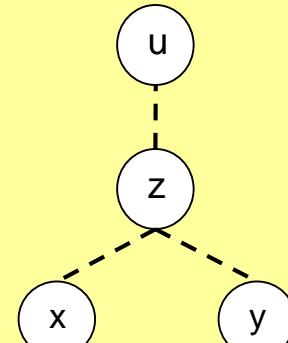
Puncte de articulație. Demonstrație teoremă (Ia)

- $p(u) = \text{null}$ și u domină cel puțin 2 subbarbri $\Rightarrow u$ este punct de articulație.
- **Dem (Reducere la absurd):** Fie A_1 și A_2 cei 2 subbarbri, $x \in A_1$, $y \in A_2$. Pp $\exists x..a..y$ și $u \notin x..a..y$.
- $z = \text{primul nod descoperit de DFS din care se poate ajunge la } x \text{ și la } y$. Cf. **T drumurilor albe** $x, y \in \text{Arb}(z)$.
- Dar $x, y \in \text{Arb}(u) \rightarrow$ 2 cazuri:



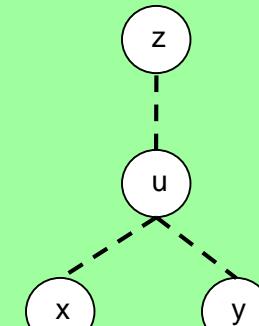
Pp $\exists x..a..y$ și $u \notin x..a..y$.

Caz 1: $d(u) < d(z)$:



Contradicție (1) x, y nu sunt în subbarbri diferenți ai lui $\text{Arb}(u)$.

Caz 2: $d(z) < d(u)$:



Contradicție (1), $p(u) \neq \text{null}$.

Puncte de articulație. Demonstrație teoremă (Ib)

- u este punct de articulație și este descoperit în ciclul principal al DFS $\Rightarrow p(u) = \text{null}$ și u domină cel puțin 2 subbarori.
- **Dem (Reducere la absurd):** Fie nodurile x și y a.î. $u \in \forall x..y$. u = primul nod descoperit din cale (altfel u nu mai e descoperit în ciclul principal al DFS) $\Rightarrow p(u) = \text{null}$ și $x, y \in \text{Arb}(u)$.

DFS(G)

$V = \text{noduri}(G)$

Pentru fiecare nod u ($u \in V$)

$c(u) = \text{alb}; p(u) = \text{null}$; // inițializare structură date

$\text{timp} = 0$; // reține distanța de la rădăcina arborelui

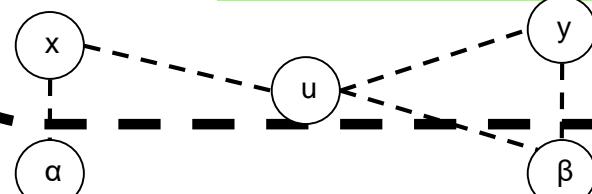
DFS până la nodul curent

Pentru fiecare nod u ($u \in V$)

Dacă $c(u)$ este alb

Atunci explorare(u); // explorez nodul

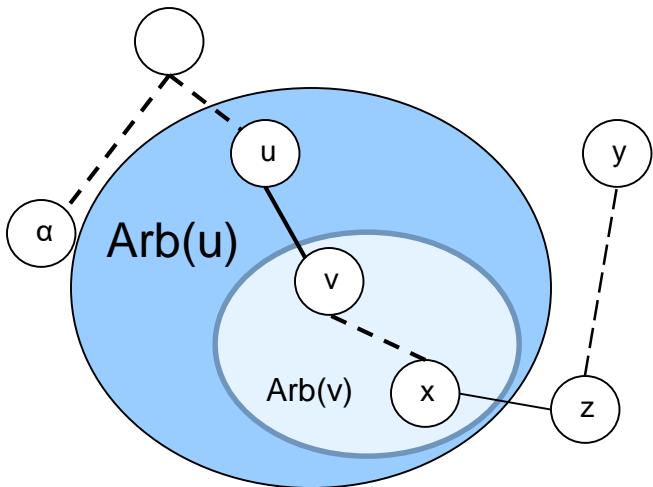
celăși subarbore



Contradicție $\exists x..z..y \Rightarrow$
u nu este punct de articulație

Puncte de articulație. Demonstrație teoremă (IIa)

- $p(u) \neq \text{null}$ și $\exists v$ descendent al lui u în $\text{Arb}(u)$ a.î. $\forall x \in \text{Arb}(v)$ și $\forall (x,z)$ parcursă de $\text{DFS}(G)$ are $d(z) \geq d(u)$
 $\Rightarrow u$ este punct de articulație.



Dem (Reducere la absurd): Pp. u nu e punct de articulație $\rightarrow \exists w \in \text{Arb}(v)$, $y \notin \text{Arb}(u)$ a.î. $y..w$. Fie z primul nod din $y..w$ a.î. $z \notin \text{Arb}(u)$ și x ultimul nod din $w..y$ a.î. $x \in \text{Arb}(u) \rightarrow (x,z)$ taie frontiera $\text{Arb}(u)$.

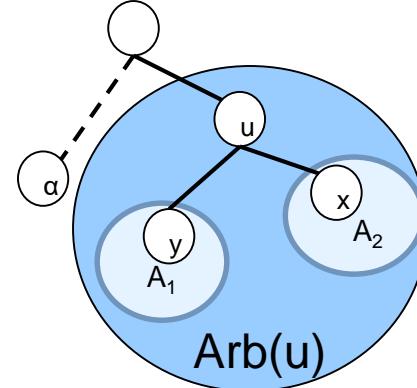
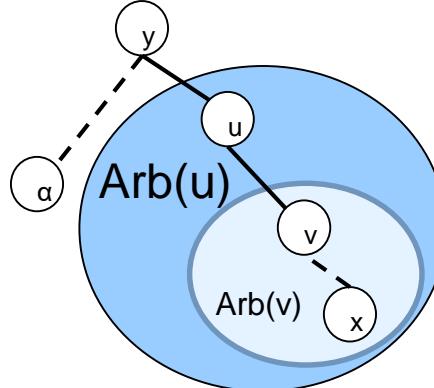
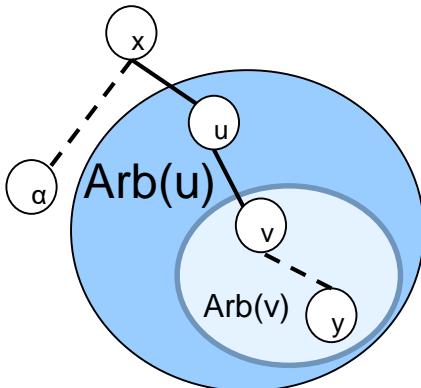
Dacă $d(z) > d(u) \rightarrow u..x, z$ alb la $d(u) \rightarrow z \in \text{Arb}(u) \rightarrow$ contradicție ($z \notin \text{Arb}(u)$)

Dacă $d(z) < d(u) \rightarrow$ contradicție (ipoteza)

$\rightarrow \nexists y..w \rightarrow u$ punct de articulație

Puncte de articulație. Demonstrație teoremă (IIb)

- u este punct de articulație și nu este descoperit în ciclul principal al DFS $\Rightarrow p(u) \neq \text{null}$ și $\exists v$ descendenter al lui u în $\text{Arb}(u)$ a.î. $\forall x \in \text{Arb}(v)$ și $\forall (x,z)$ parcursă de $\text{DFS}(G)$ având $d(z) \geq d(u)$.
- **Dem:** Fie nodurile x și y a.î. $u \in \forall x..y$ și $p(u) \neq \text{null}$. Se pot forma 3 tipuri de structuri:



- Pentru primele 2 structuri, nu trebuie să existe muchie care să formeze ciclu de la nici un nod din $\text{Arb}(v)$ către vreun predecesor al lui u . Altfel $\exists x..y$ a.î. $u \notin x..y$.
- Pentru a 3-a structură, trebuie să \nexists muchie care să formeze ciclu către un predecesor al lui u de la niciun nod din cel puțin un subarbore A_1 sau A_2 .

Puncte de articulație. Structuri de date.

- Structura de date de la DFS + pentru fiecare nod $u \in V$ se rețin:
 - $\text{Low}(u) = \min\{\text{d}(v) \mid v \text{ descoperit pornind din } u \text{ în cursul DFS și } c(v) \neq \text{alb}\}$
 - $\text{Subarb}(u) = \text{numărul subarborilor dominati de } u$ (dacă este ≥ 2 , atunci avem un punct de articulație).

Idee algoritm

- Se aplică DFS și se salvează pentru fiecare nod până unde merge înapoi (low):
$$\text{low}[u] = \min \{d(u), d(v) \text{ pentru toate muchiile înapoi } (u,v), \text{low}(w) \text{ pentru toți fiile } w \text{ ai lui } u\}.$$
- Pentru **eficiență**, trebuie ca fiile să se parcurgă înaintea părintilor \rightarrow ordinea inversă a $d(u)$.

Algoritm Tarjan (I)

- Articulații (G)

- $V = \text{noduri}(G)$ // inițializări

- Timp = 0;

- **Pentru fiecare** ($u \in V$)

- $c(u) = \text{alb}$;

- $d(u) = 0$;

- $p(u) = \text{null}$;

- $\text{low}(u) = 0$;

- $\text{subarb}(u) = 0$; // reține numărul de subbarbi dominați de u

- $\text{art}(u) = 0$; // reține punctele de articulație

- **Pentru fiecare** ($u \in V$)

- **Dacă** $c(u)$ e alb

- Exploreaza(u);

- **Dacă** ($\text{subarb}(u) > 1$) // cazul în care u este rădăcina în arborele

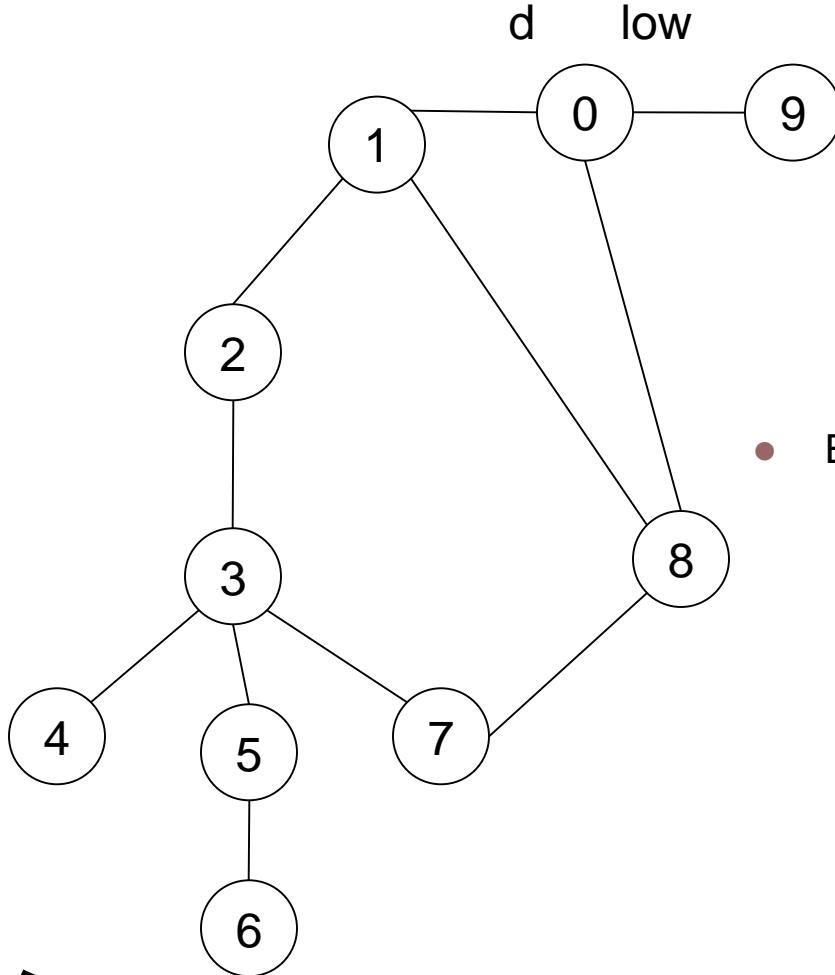
- $\text{art}(u) = 1$ // DFS și are mai mulți subbarbi \rightarrow cazul

- // 1 al teoremei

Algoritm Tarjan (II)

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - **Pentru fiecare** nod $v \in \text{succs}(u)$
 - **Dacă** ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr subarbori
// dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - **Dacă** ($p(u) \neq \text{null}$ && $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$
// cazul 2 al teoremei
 - **Altfel** $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

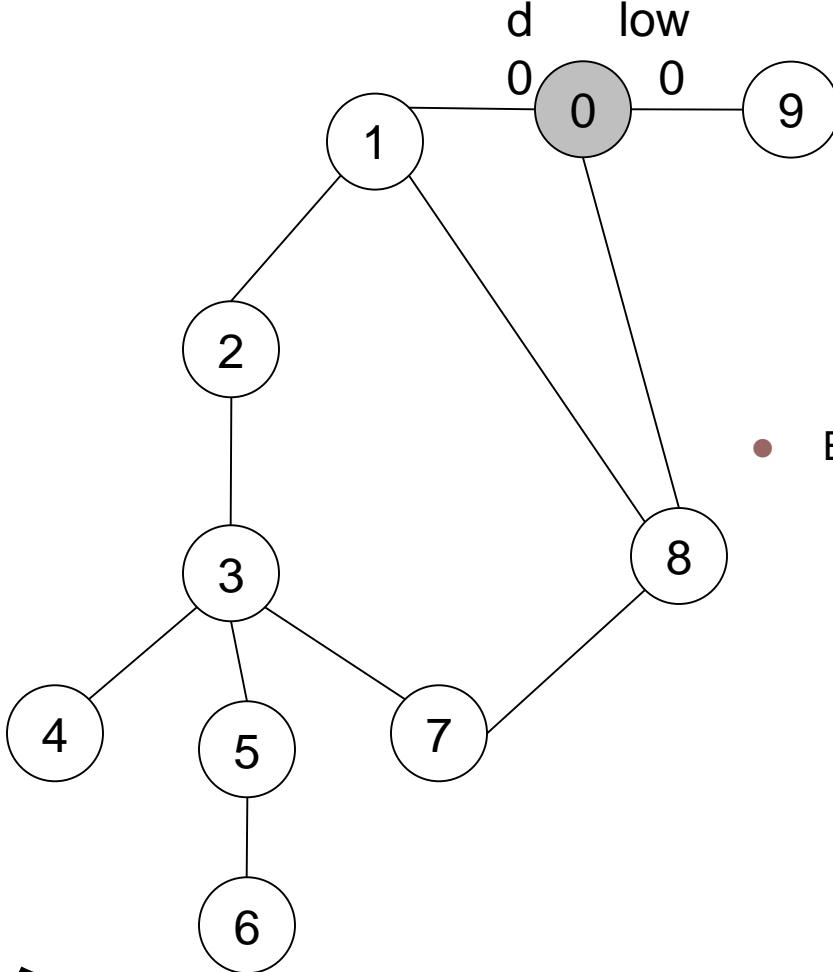
Exemplu rulare (1)



Timp = 0
C(i) = alb
D(i) = 0
Low(i) = 0
P(i) = null
Subarb(i) = 0
Art(i) = 0
Exploreaza (0)

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++$; // inițializări
 - $c(u) = \text{gri}$;
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u$; $\text{subarb}(u)++$; // actualizare nr // subarbوري dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(v) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1$; // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

Exemplu rulare (2)



$$\text{Low}(0) = d(0) = 0$$

$$\text{Timp} = 1$$

$$C(0) = \text{gri}$$

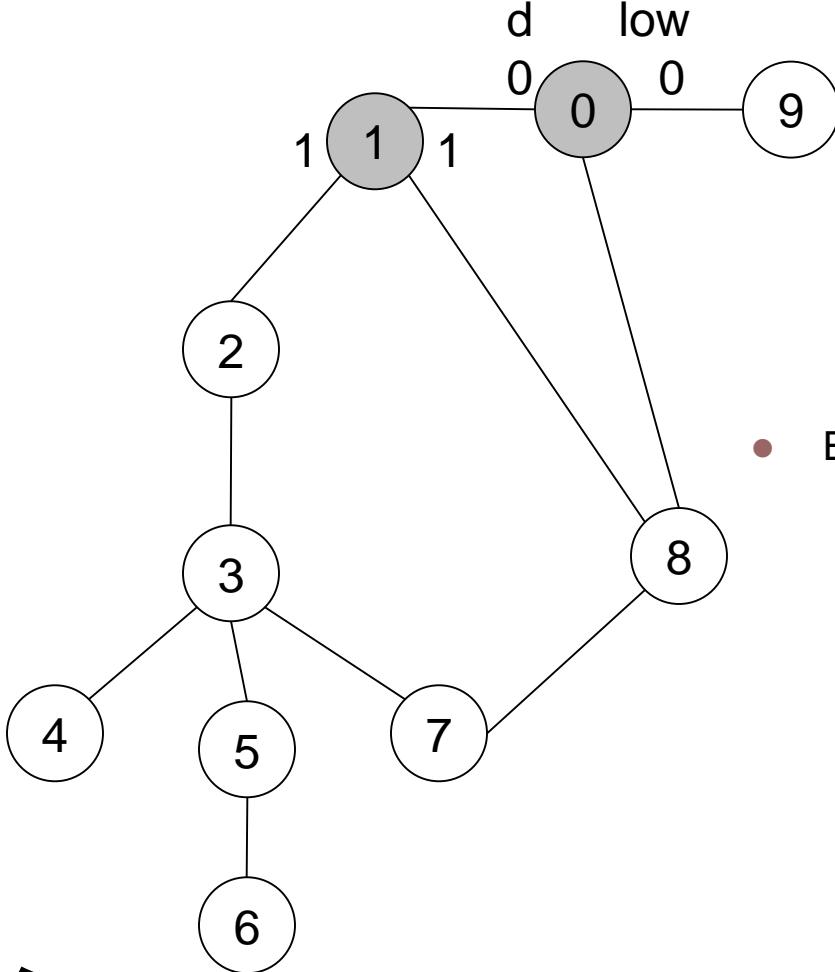
$$P(1) = 0$$

$$\text{Subarb}(0) = 1$$

Exploreaza (1)

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarbوري dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(v) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

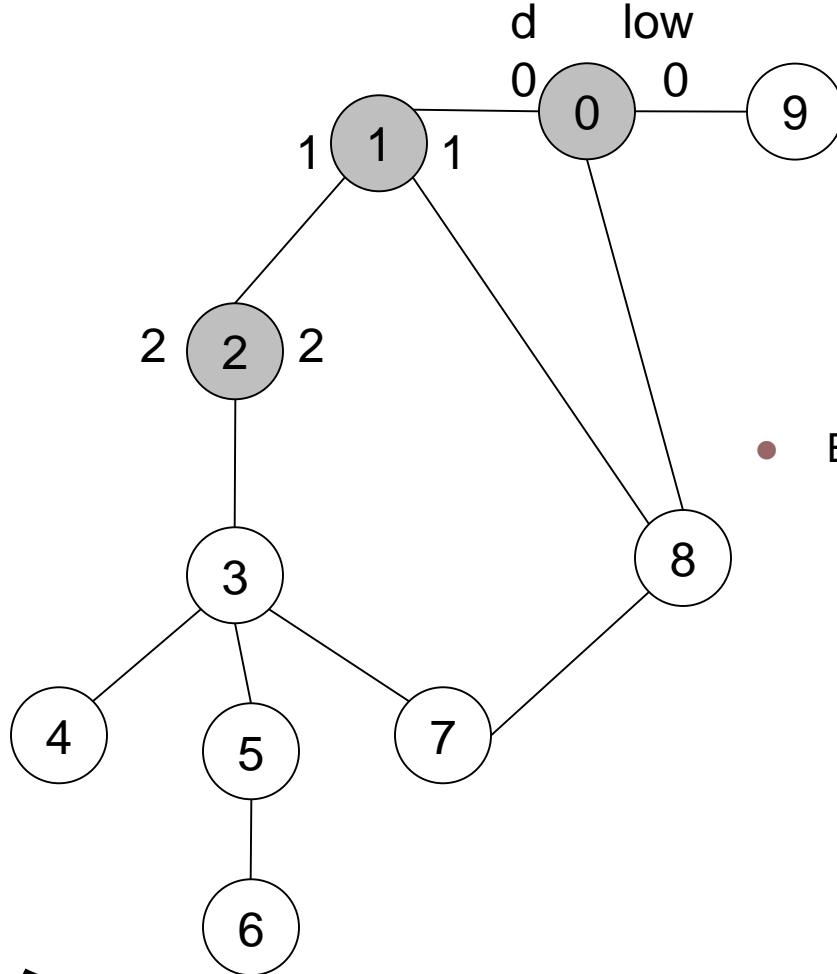
Exemplu rulare (3)



Low(1) = d(1) = 1
Timp = 2
C(1) = gri
P(2) = 1
Subarb(1) = 1
Exploreaza (2)

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++$; // inițializări
 - $c(u) = \text{gri}$;
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u$; $\text{subarb}(u)++$; // actualizare nr // subarbوري dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(u) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1$; // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

Exemplu rulare (4)



$$\text{Low}(2) = d(2) = 2$$

Timp = 3

C(2) = gri

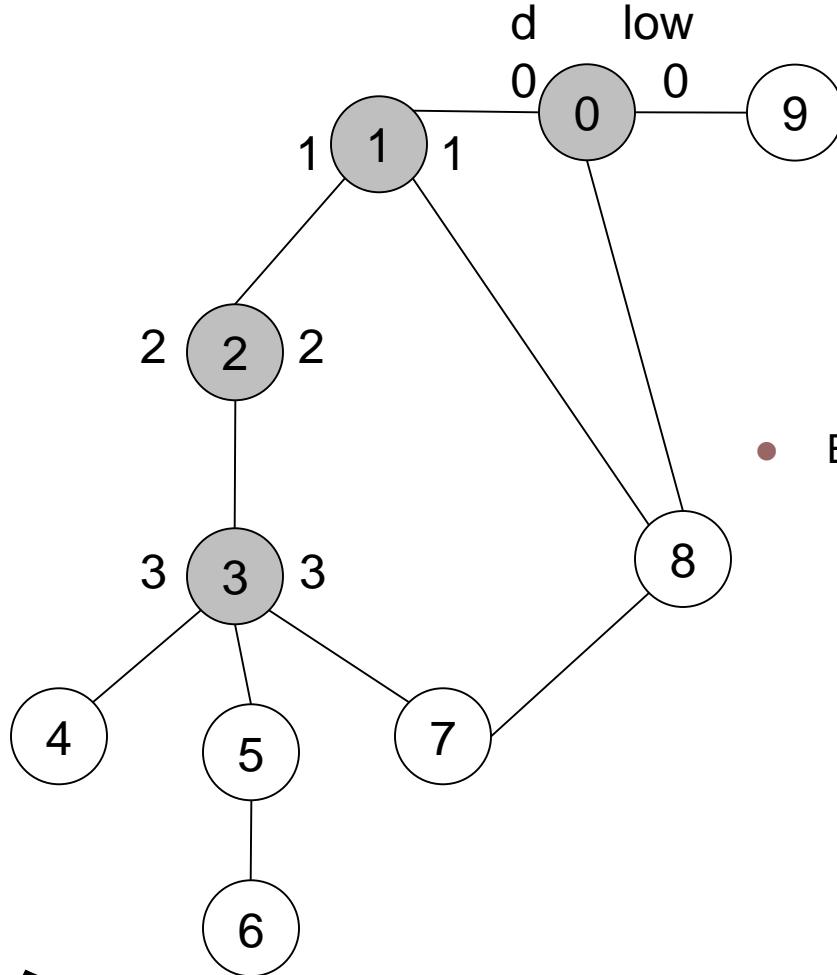
P(3) = 2

Subarb(2) = 1

Exploreaza (3)

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(u) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

Exemplu rulare (5)



$$\text{Low}(3) = d(3) = 3$$

Timp = 4

C(3) = gri

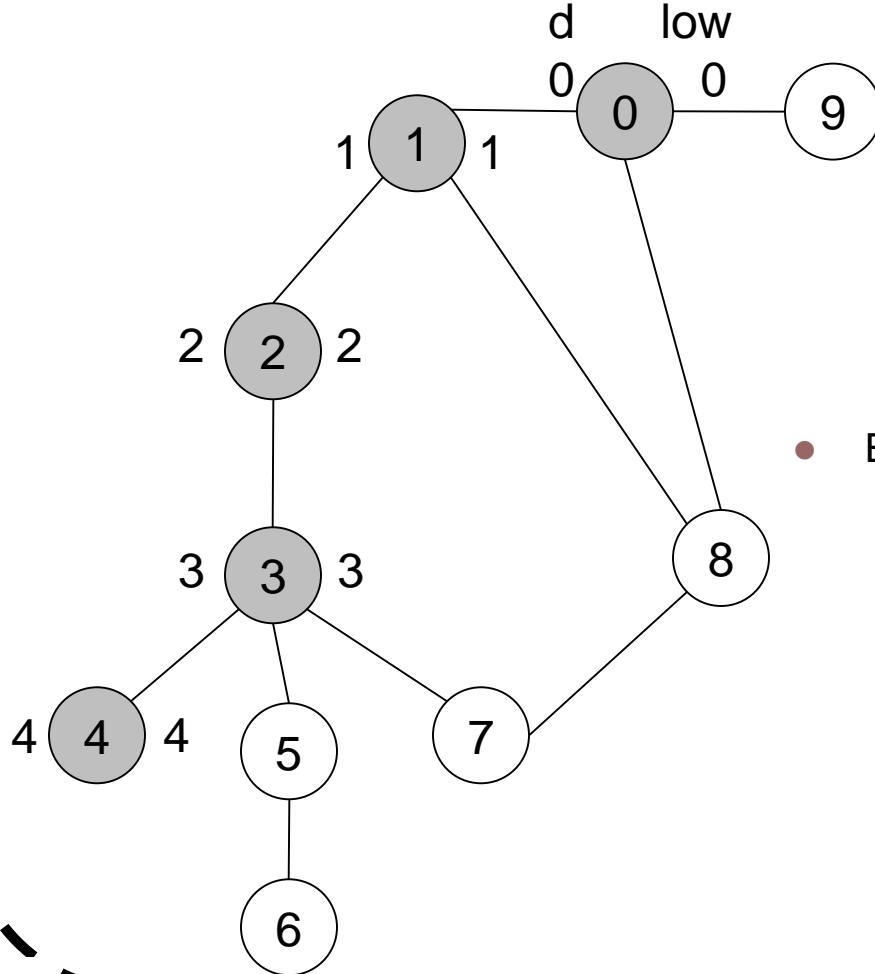
P(4) = 3

Subarb(3) = 1

Exploreaza (4)

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - **Pentru fiecare** nod $v \in \text{succs}(u)$
 - **Dacă** ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarbوري dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - **Dacă** ($p(u) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - **Altfel** $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

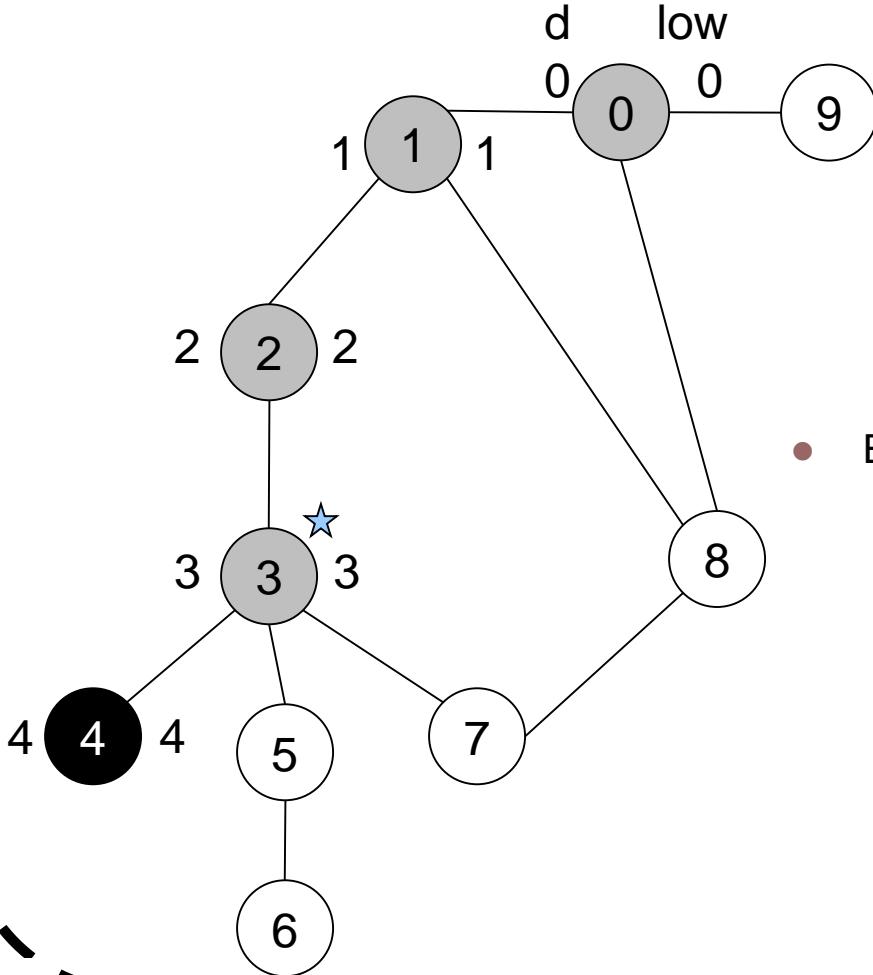
Exemplu rulare (6)



Low(4) = d(4)=4
Timp =5
C(4) =gri
revenire

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarbوري dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(u) \neq \text{null} \& \text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

Exemplu rulare (7)



$$\text{Low}(4) = d(4) = 4$$

$$\text{Timp} = 5$$

$C(4) = \text{gri}$
revenire

$$\text{Low}(3) = \min\{\text{low}(3), \text{low}(4)\} = 3$$

$$\text{Low}(4) > d(3) \rightarrow \text{art}(3) = 1$$

$$P(5) = 3$$

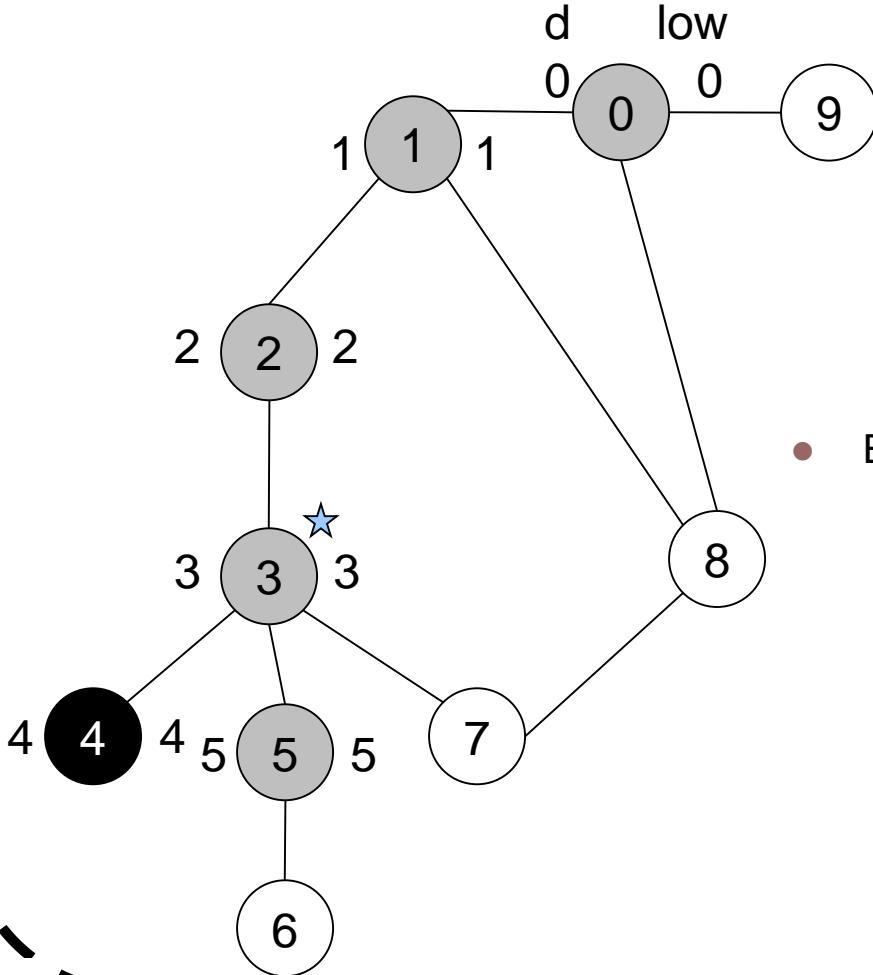
$$\text{Subarb}(3) = 2$$

Exploreaza (5)

- Explorează(u)

- $d(u) = \text{low}(u) = \text{timp}++; // \text{initializare}$
- $c(u) = \text{gri};$
- **Pentru fiecare** nod $v \in \text{succs}(u)$
 - **Dacă** ($c(v)$ e alb)
 - $p(v) = u; \text{subarb}(u)++; // \text{actualizare nr subarbوري dominați de } u$
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\} // \text{actualizare low}$
 - **Dacă** ($p(u) \neq \text{null} \& \text{low}(v) \geq d(u)$) $\text{art}(u) = 1; // \text{cazul 2 al teoremei}$
 - **Altfel** $\text{low}(u) = \min\{\text{low}(u), d(v)\} // \text{actualizare low}$

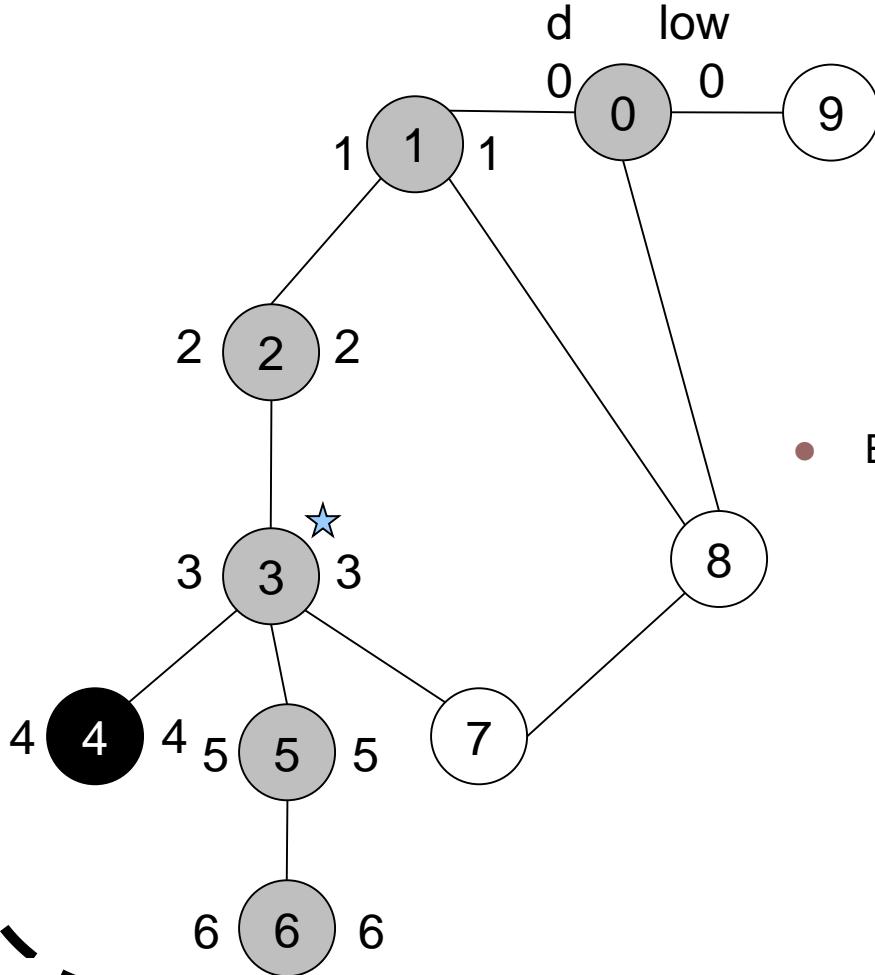
Exemplu rulare (8)



Low(5) = d(5) = 5
Timp = 6
C(5) = gri
P(6) = 5
Subarb(5) = 1
Exploreaza (6)

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++$; // inițializări
 - $c(u) = \text{gri}$;
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u$; $\text{subarb}(u)++$; // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(v) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1$; // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

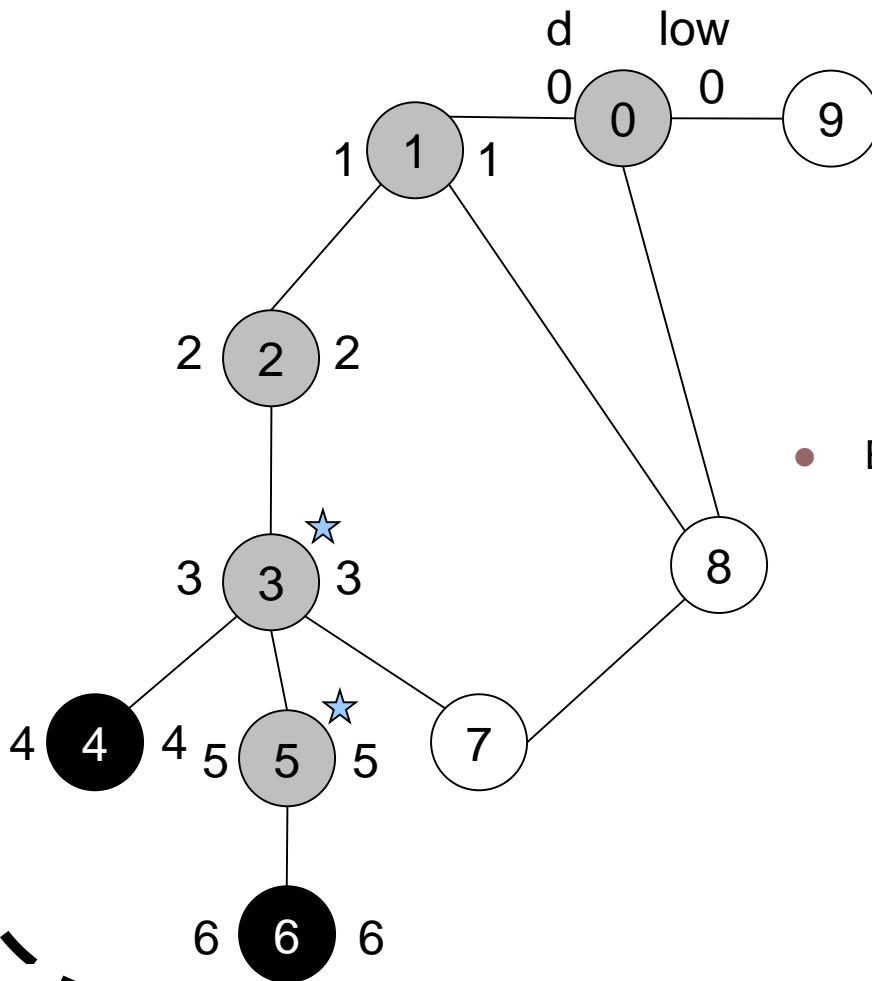
Exemplu rulare (9)



Low(6) = d(6) = 6
Timp = 7
C(6) = gri
revenire

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++$; // inițializări
 - $c(u) = \text{gri}$;
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u$; $\text{subarb}(u)++$; // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(v) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1$; // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

Exemplu rulare (10)



$$\text{Low}(6) = d(6) = 6$$

$$\text{Timp} = 7$$

$C(6) = \text{gri}$

revenire

$$\text{Low}(5) = \min\{\text{low}(5), \text{low}(6)\} = 5$$

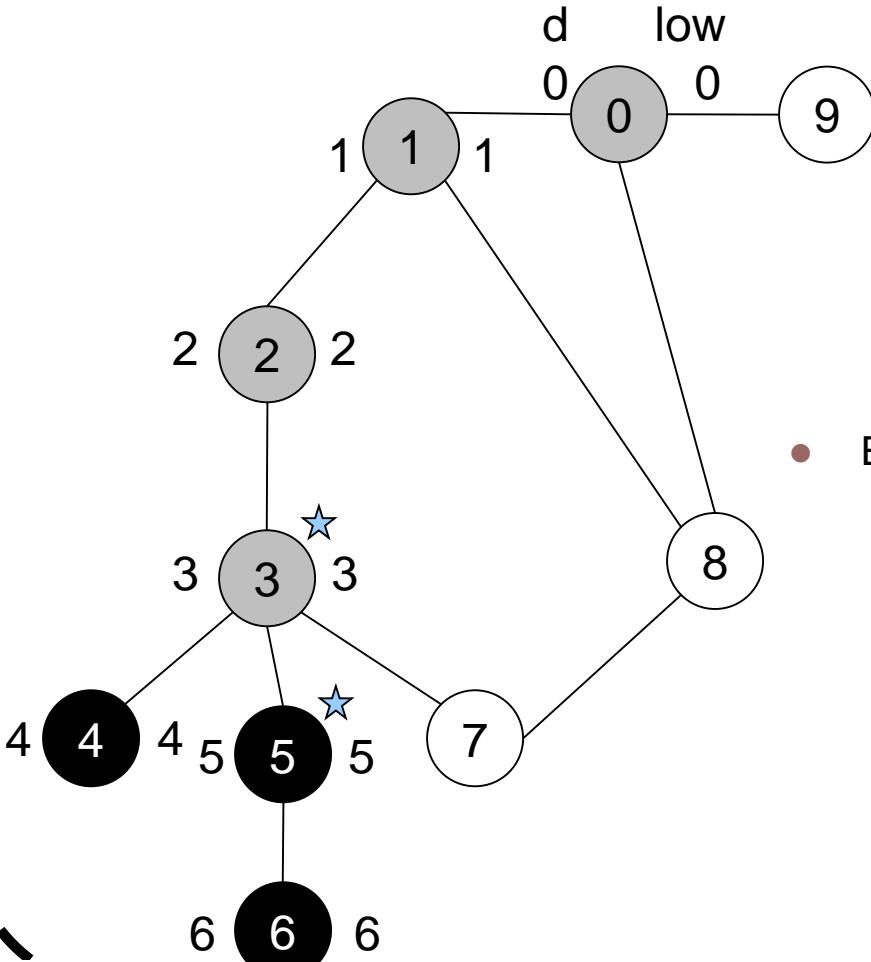
$$\text{Low}(6) > d(5) \rightarrow \text{art}(5) = 1$$

revenire

- Explorează(u)

- $d(u) = \text{low}(u) = \text{timp}++$; // inițializări
- $c(u) = \text{gri}$;
- **Pentru fiecare** nod $v \in \text{succs}(u)$
 - **Dacă** ($c(v)$ e alb)
 - $p(v) = u$; $\text{subarb}(u)++$; // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - **Dacă** ($p(u) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1$; // cazul 2 al teoremei
 - **Altfel** $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

Exemplu rulare (11)



$$\text{Low}(5) = d(5) = 5$$

$$\text{Timp} = 7$$

$$C(5) = \text{gri}$$

revenire

$$\text{Low}(3) = \min\{\text{low}(3), \text{low}(5)\} = 3$$

$$\text{Low}(5) > d(3) \rightarrow \text{art}(3) = 1$$

$$P(7) = 3$$

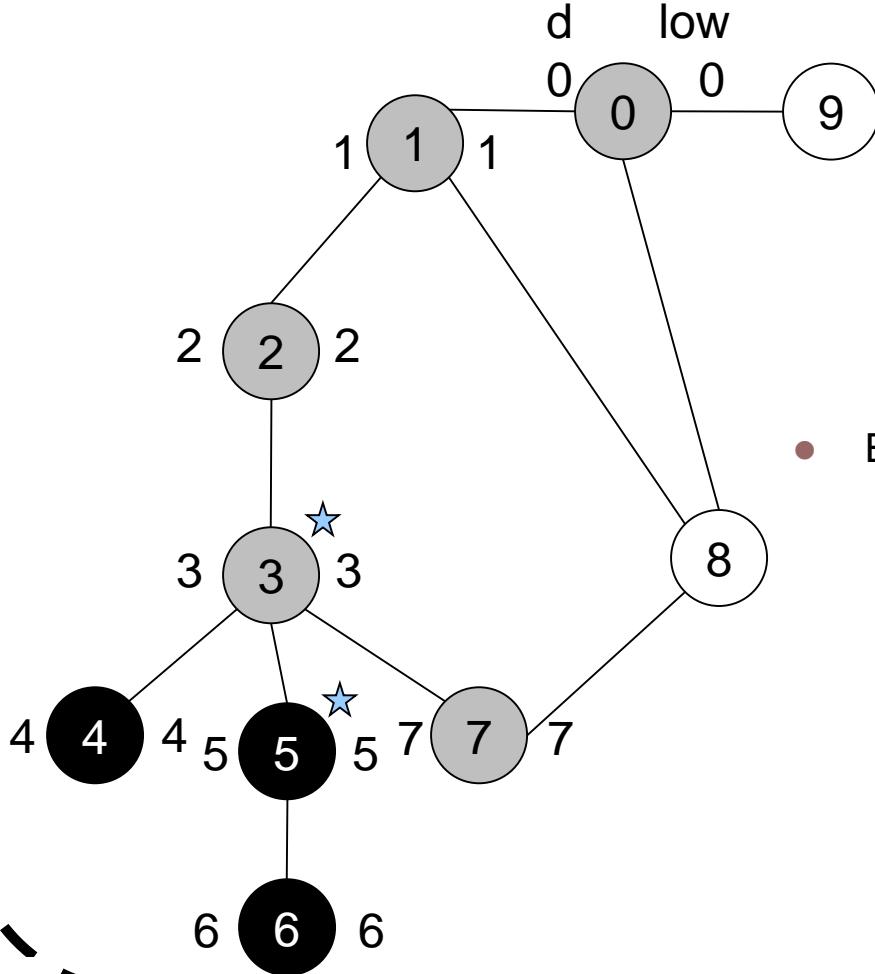
$$\text{Subarb}(3) = 3$$

Exploreaza (7)

- Explorează(u)

- $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
- $c(u) = \text{gri};$
- **Pentru fiecare** nod $v \in \text{succs}(u)$
 - **Dacă** ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarbوري dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - **Dacă** ($p(u) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - **Altfel** $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

Exemplu rulare (12)



$$\text{Low}(7) = d(7) = 7$$

$$\text{Timp} = 8$$

$$C(7) = \text{gri}$$

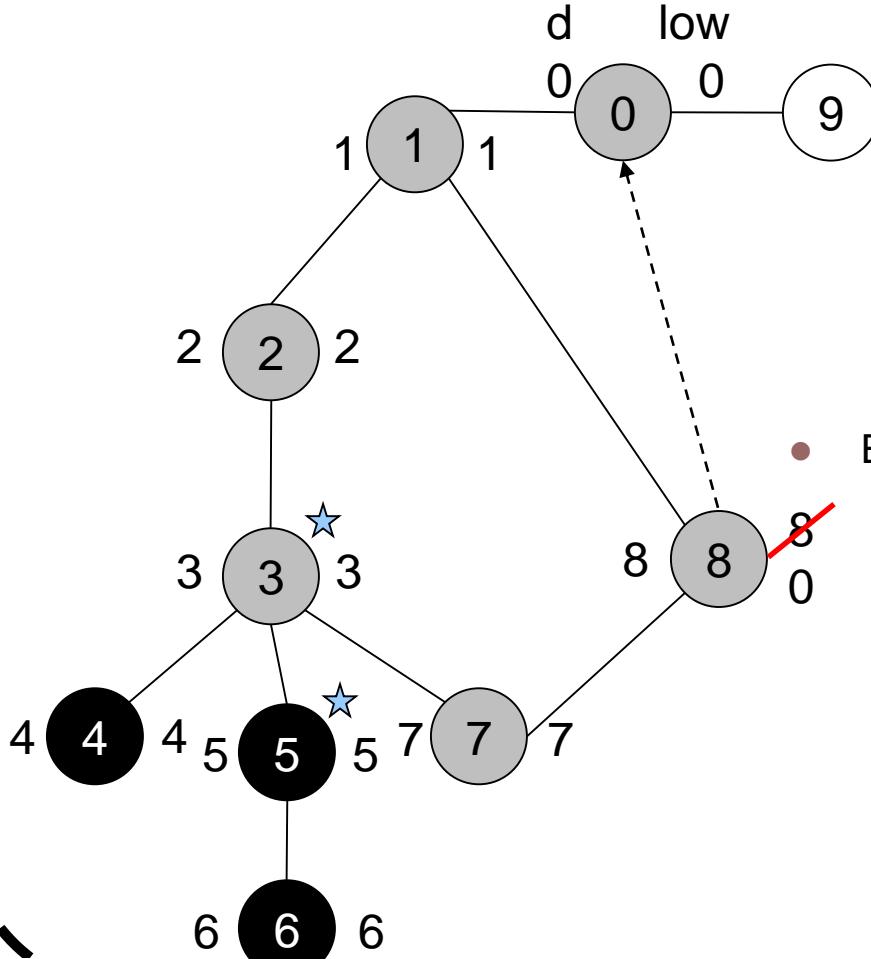
$$P(8) = 7$$

$$\text{Subarb}(7) = 1$$

Exploreaza (8)

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarbوري dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(u) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

Exemplu rulare (13)



$$\text{Low}(8) = d(8) = 8$$

$$\text{Timp} = 9$$

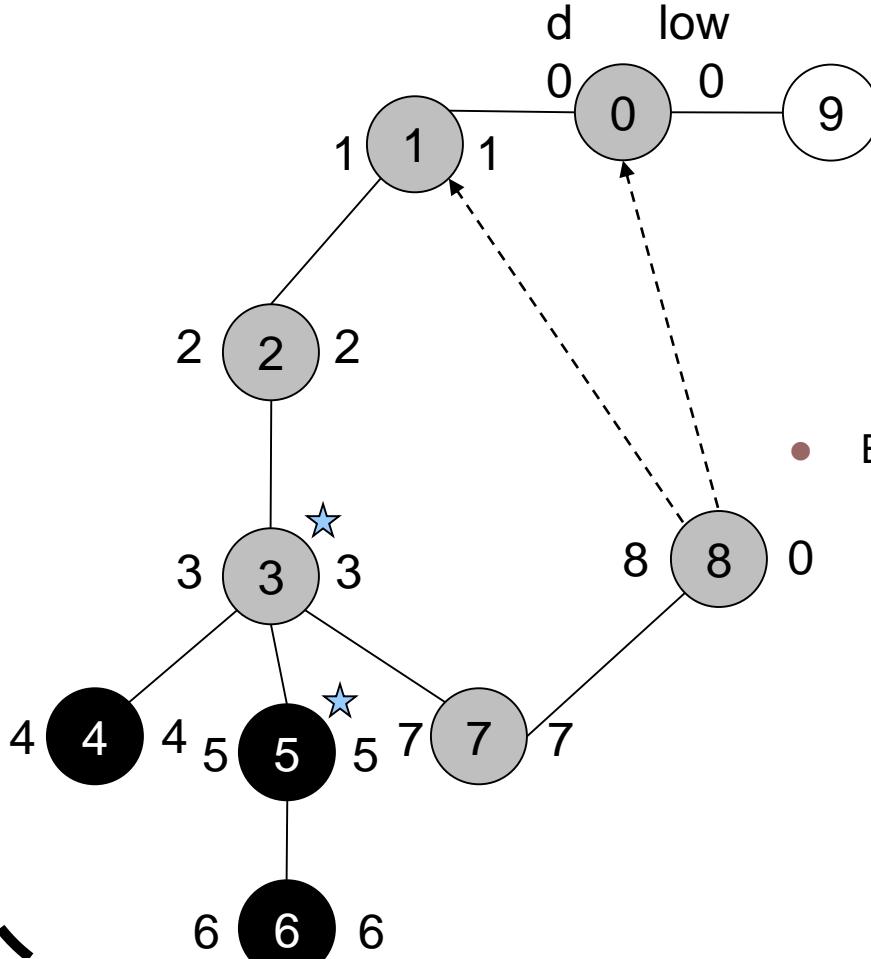
$$C(8) = \text{gri}$$

$$\text{Low}(8) = \min\{d(0), \text{low}(8)\} = 0$$

Explorează(u)

- $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
- $c(u) = \text{gri};$
- **Pentru fiecare** nod $v \in \text{succs}(u)$
 - **Dacă** ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - **Dacă** ($p(u) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - **Altfel** $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

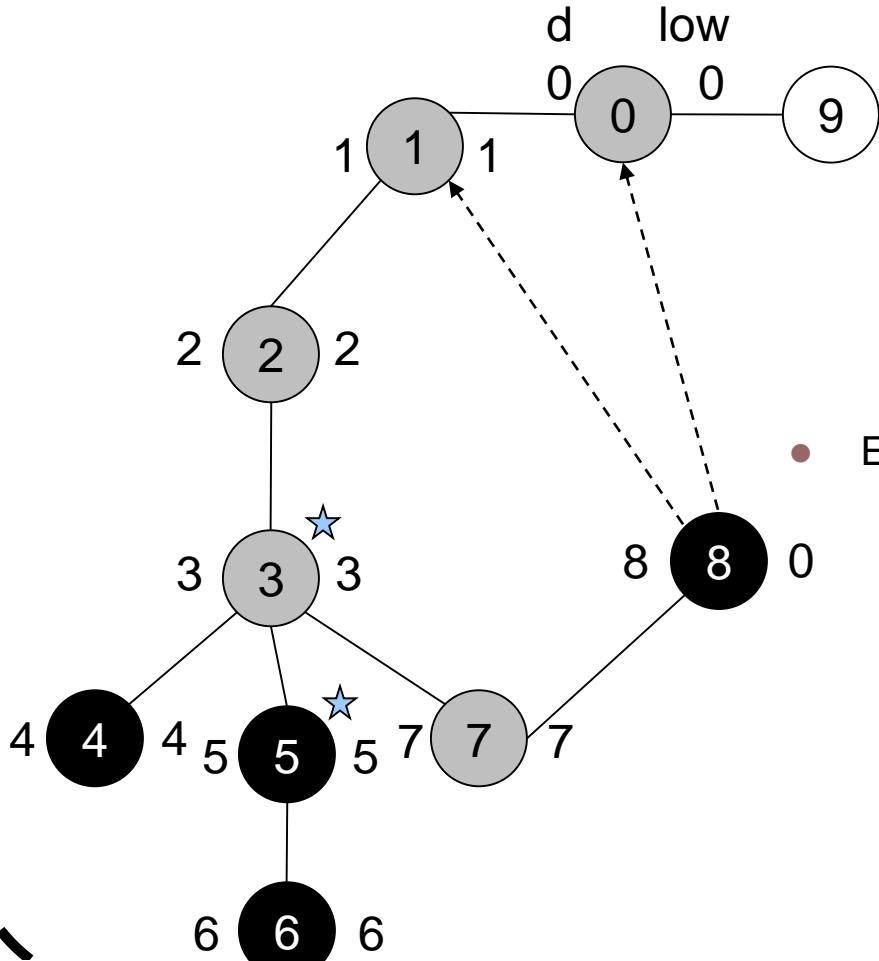
Exemplu rulare (14)



$d(8) = 8$
 $\text{Low}(8) = 0$
 $\text{Timp} = 9$
 $C(8) = \text{gri}$
 $\text{Low}(8) = \min\{d(1), \text{low}(8)\} = 0$

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++$; // inițializări
 - $c(u) = \text{gri}$;
 - **Pentru fiecare** nod $v \in \text{succs}(u)$
 - **Dacă** ($c(v)$ e alb)
 - $p(v) = u$; $\text{subarb}(u)++$; // actualizare nr // subarbوري dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - **Dacă** ($p(u) \neq \text{null}$ && $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1$; // cazul 2 al teoremei
 - **Altfel** $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

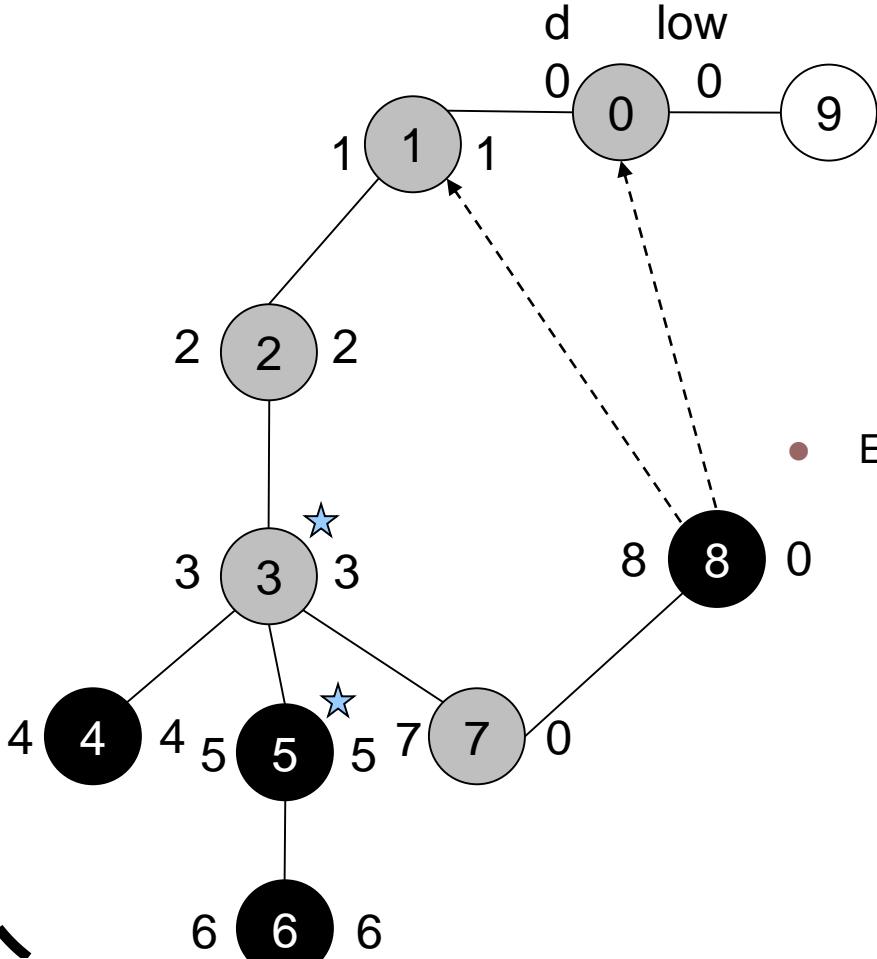
Exemplu rulare (15)



$d(8) = 8$
 $\text{Low}(8) = 0$
 $\text{Timp} = 9$
 $C(8) = \text{gri}$
revenire

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - **Pentru fiecare** nod $v \in \text{succs}(u)$
 - **Dacă** ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - **Dacă** ($p(v) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - **Altfel** $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

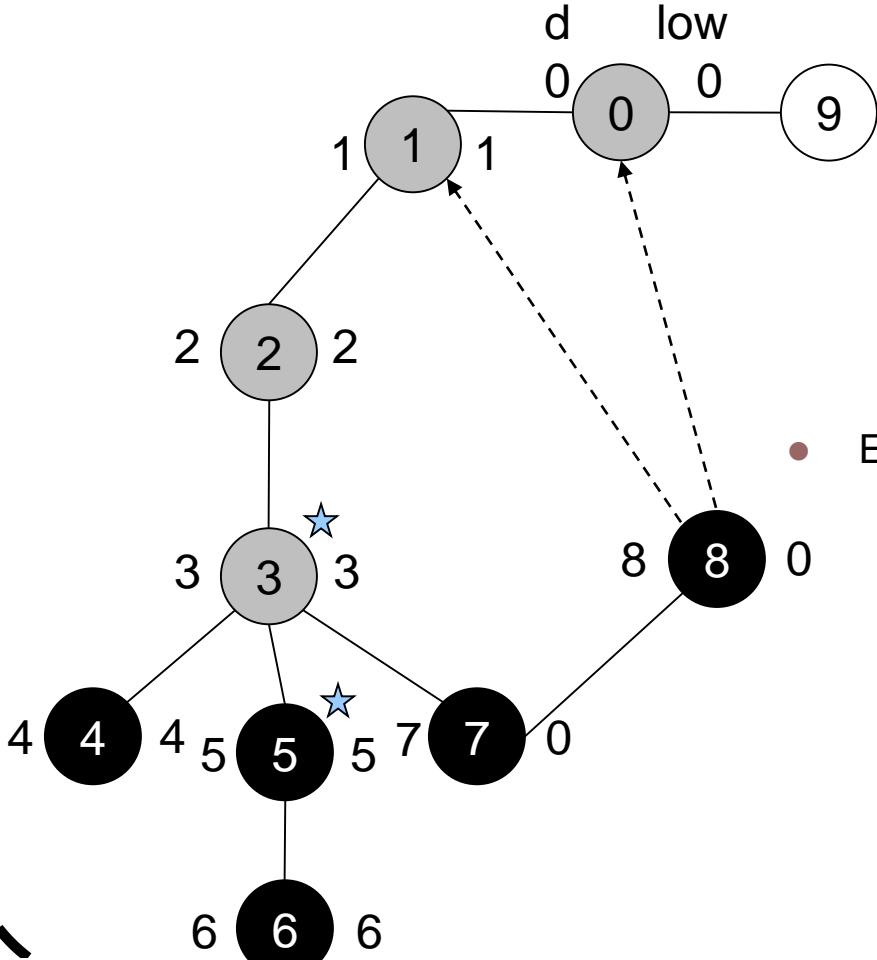
Exemplu rulare (16)



$\text{low}(7) = \min(\text{low}(7), \text{low}(8)) = 0$
 $\text{low}(8) < d(7) \rightarrow \text{nu se modifica art}(7)$

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++$; // inițializări
 - $c(u) = \text{gri}$;
 - **Pentru fiecare** nod $v \in \text{succs}(u)$
 - **Dacă** ($c(v)$ e alb)
 - $p(v) = u$; $\text{subarb}(u)++$; // actualizare nr subarbوري dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - **Dacă** ($p(u) \neq \text{null} \& \text{low}(v) \geq d(u)$) $\text{art}(u) = 1$; // cazul 2 al teoremei
 - **Altfel** $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

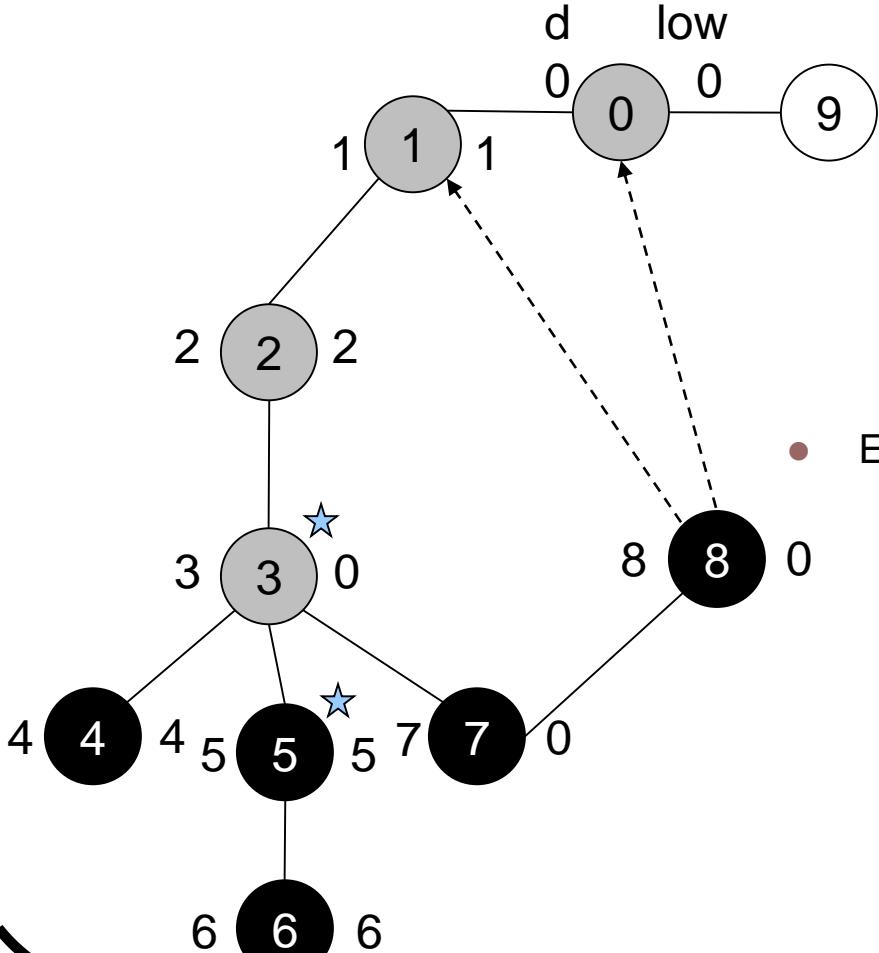
Exemplu rulare (17)



$\text{low}(7) = \min(\text{low}(7), \text{low}(8)) = 0$
revenire

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarbori dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(u) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

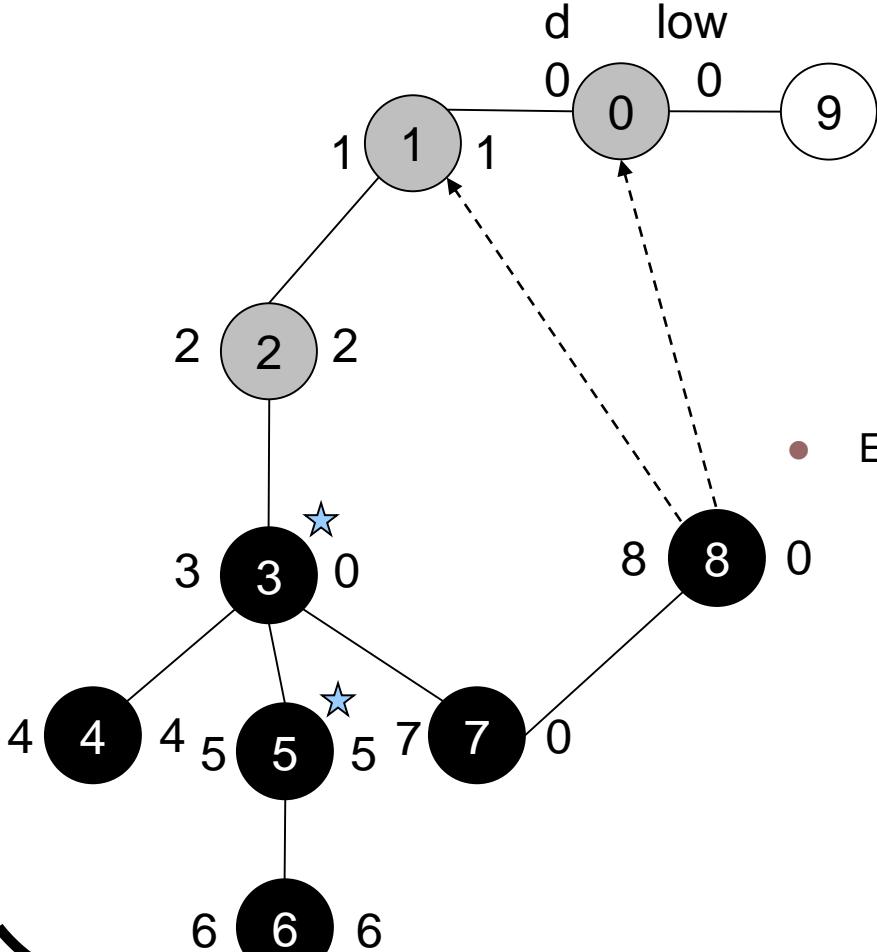
Exemplu rulare (18)



$\text{low}(3) = \min(\text{low}(3), \text{low}(7)) = 0$
 $\text{low}(7) < d(3) \rightarrow \text{nu se modifică}$
 $\text{art}(3)$

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++$; // inițializări
 - $c(u) = \text{gri}$;
 - **Pentru fiecare** nod $v \in \text{succs}(u)$
 - **Dacă** ($c(v)$ e alb)
 - $p(v) = u$; $\text{subarb}(u)++$; // actualizare nr // subarbوري dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - **Dacă** ($p(u) \neq \text{null} \& \text{low}(v) \geq d(u)$) $\text{art}(u) = 1$; // cazul 2 al teoremei
 - **Altfel** $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

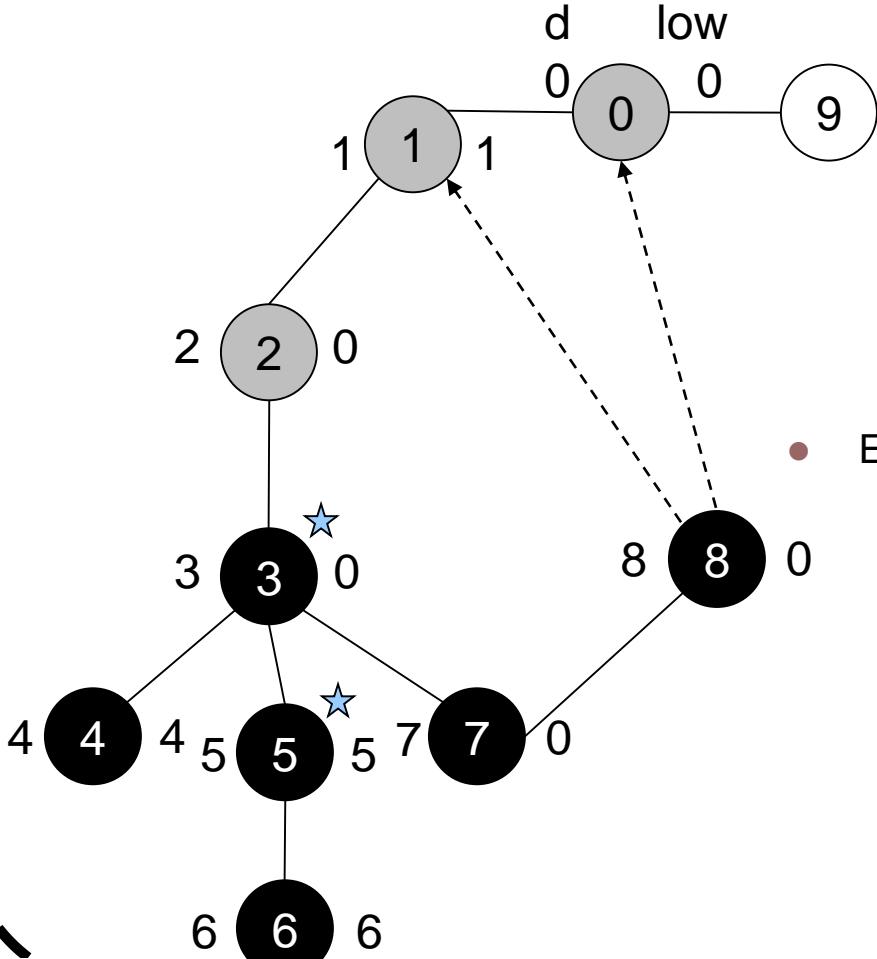
Exemplu rulare (19)



$\text{low}(3) = \min(\text{low}(3), \text{low}(7)) = 0$
 $\text{low}(7) < d(3) \rightarrow \text{nu se modifică}$
 $\text{art}(3)$
revenire

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(v) \neq \text{null} \& \text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

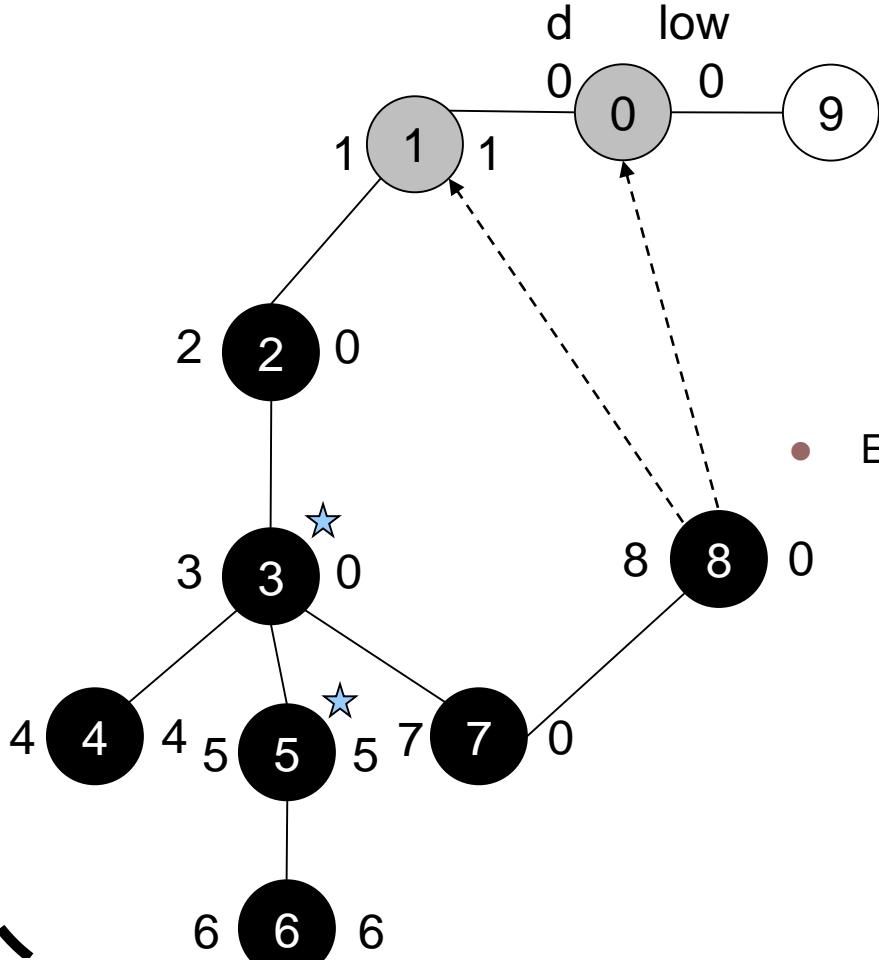
Exemplu rulare (20)



$\text{low}(2) = \min(\text{low}(2), \text{low}(3)) = 0$
 $\text{low}(3) < d(2) \rightarrow \text{nu se modifică}$
 $\text{art}(2)$

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(v) \neq \text{null} \& \text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

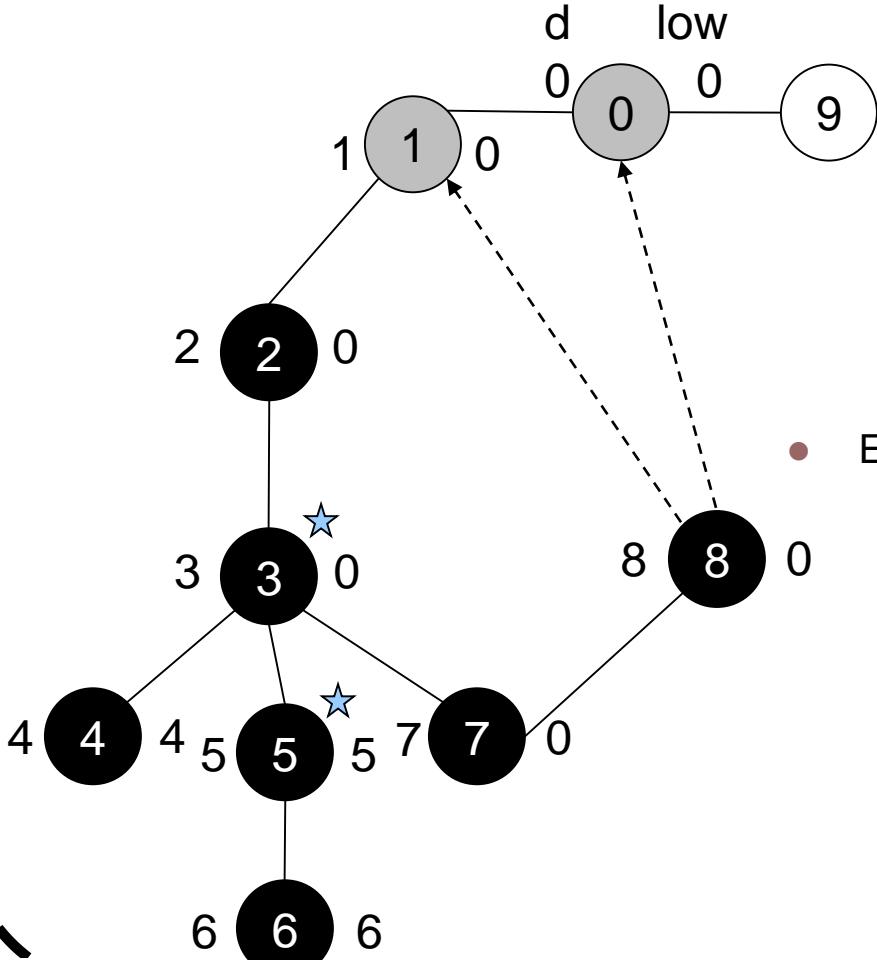
Exemplu rulare (21)



$\text{low}(2) = \min(\text{low}(2), \text{low}(3)) = 0$
 $\text{low}(3) < d(2) \rightarrow \text{nu se modifică}$
 $\text{art}(2)$
revenire

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(u) \neq \text{null} \& \text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

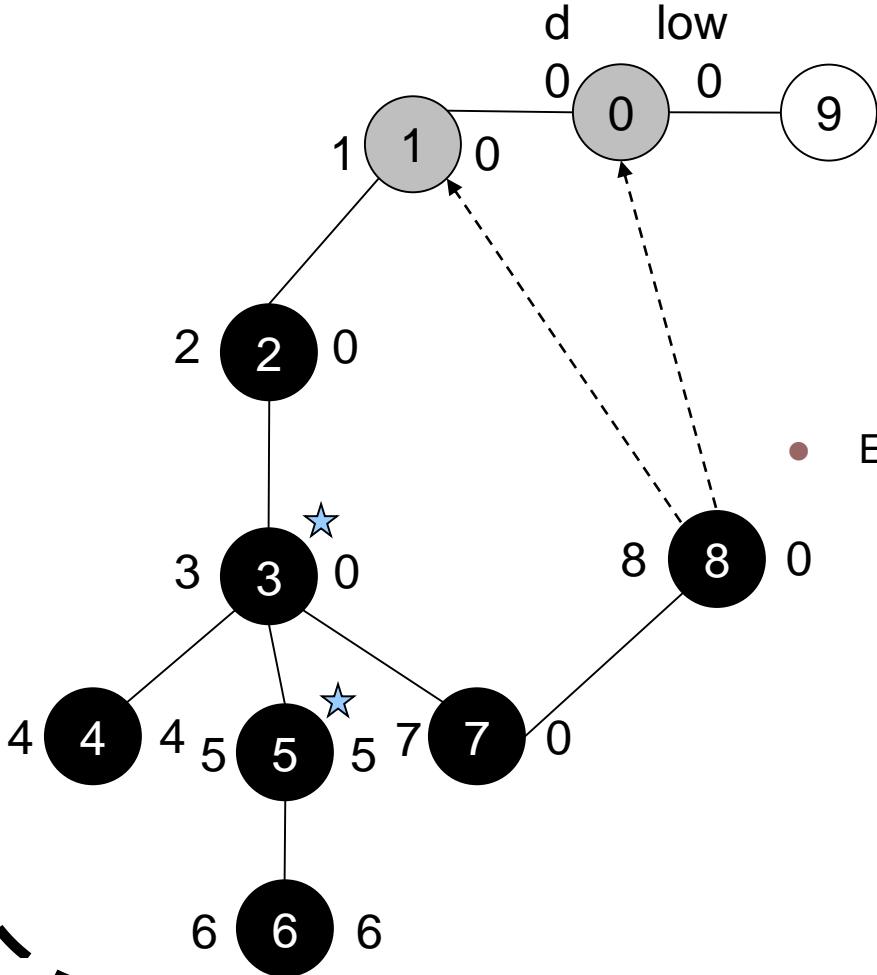
Exemplu rulare (22)



$low(1) = \min(low(1), low(2)) = 0$
 $low(2) < d(1) \rightarrow$ nu se modifică
 $art(1)$

- Explorează(u)
 - $d(u) = low(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $low(u) = \min\{low(u), low(v)\}$ // actualizare low
 - Dacă ($p(v) \neq \text{null} \& low(v) \geq d(u)$) $art(u) = 1;$ // cazul 2 al teoremei
 - Altfel $low(u) = \min\{low(u), d(v)\}$ // actualizare low

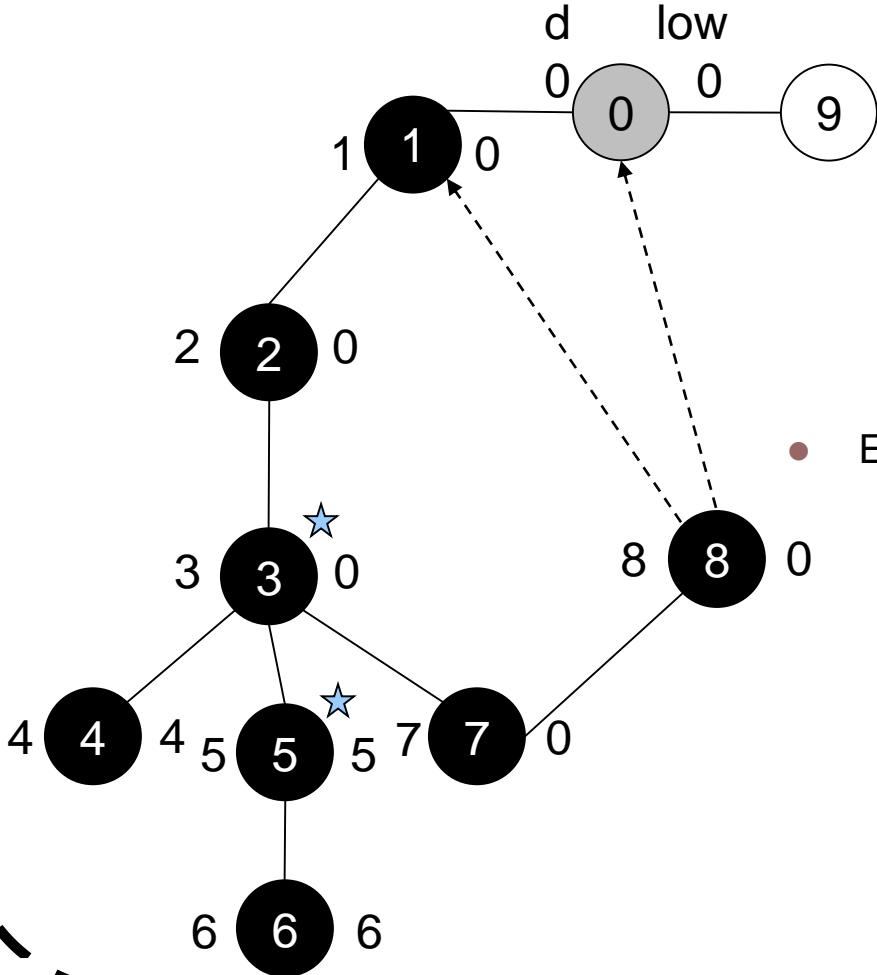
Exemplu rulare (23)



$$\begin{aligned} \text{low}(1) &= 0 \\ \text{low}(1) &= \min(\text{low}(1), d(8)) = 0 \end{aligned}$$

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(v) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

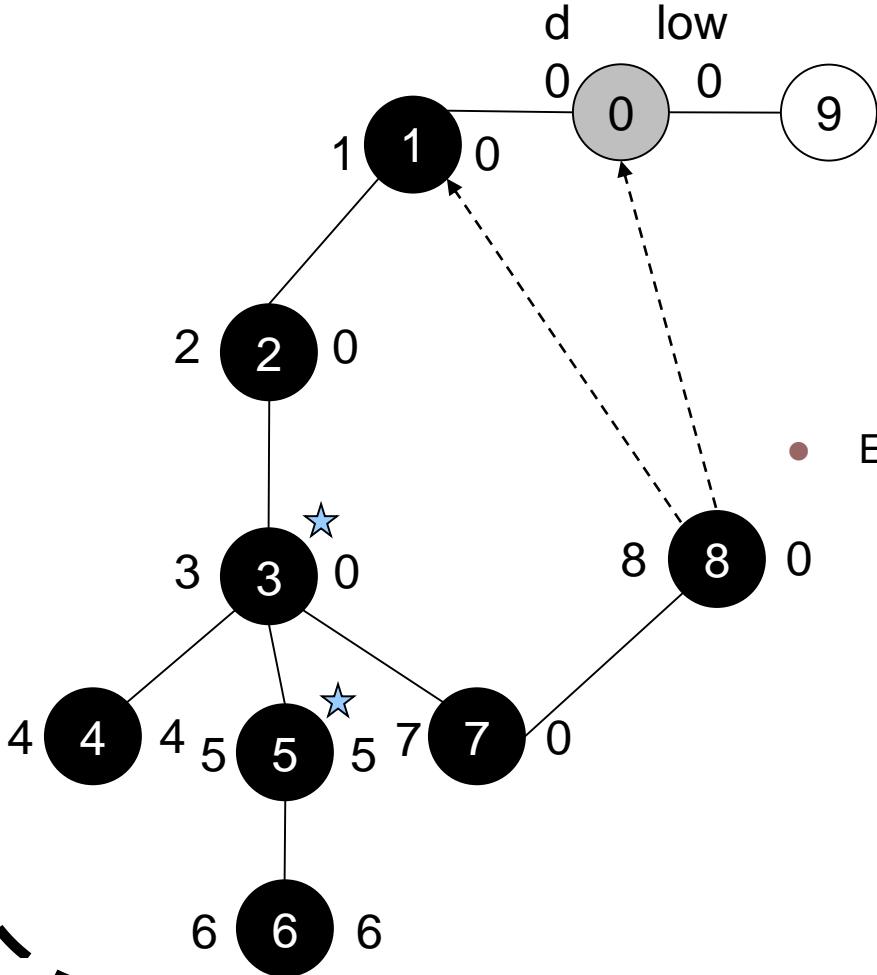
Exemplu rulare (24)



$\text{low}(1) = 0$
revenire

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++$; // inițializări
 - $c(u) = \text{gri}$;
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u$; $\text{subarb}(u)++$; // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(u) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1$; // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

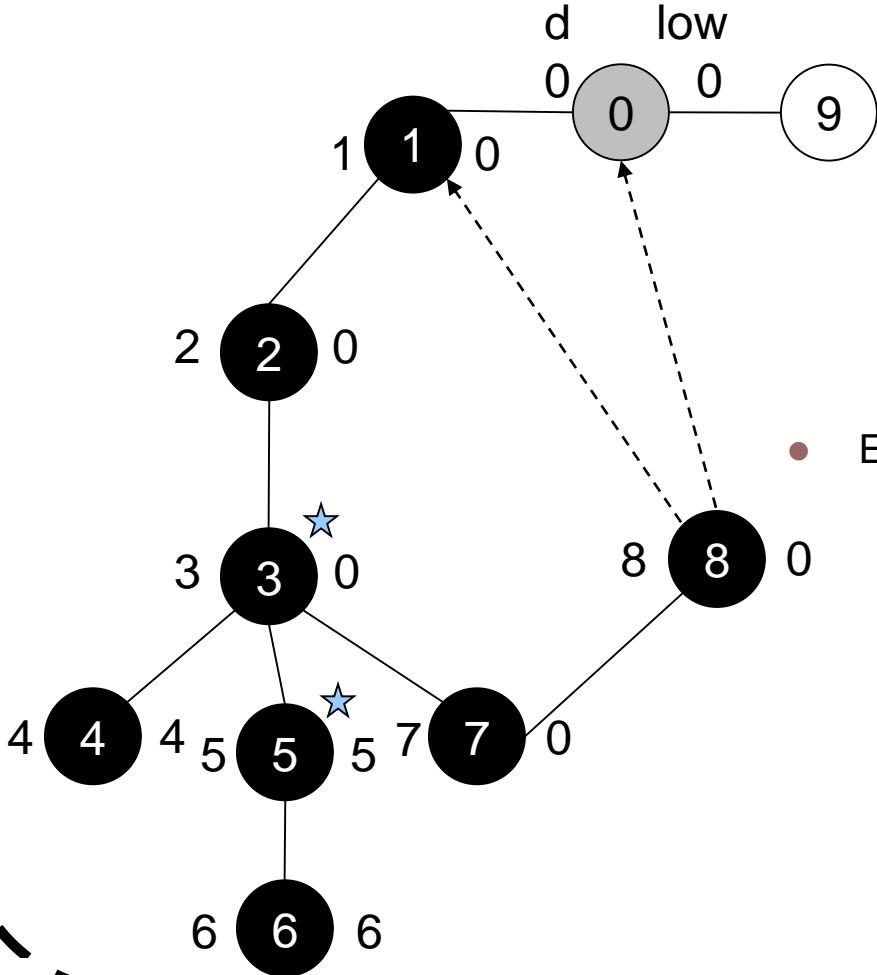
Exemplu rulare (25)



$\text{low}(0) = \min\{\text{low}(1), \text{low}(0)\} = 0$
 $P(0) = \text{null} \rightarrow$ continuă cu următorul copil

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(u) \neq \text{null} \& \text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

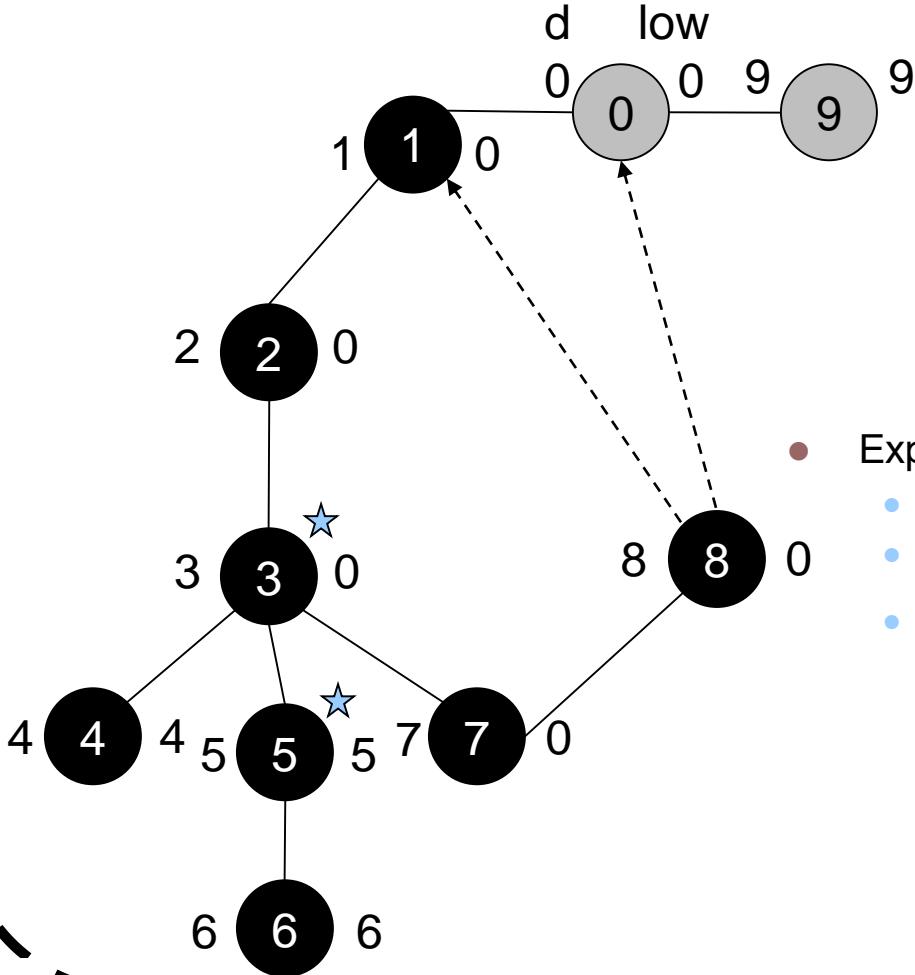
Exemplu rulare (26)



$low(0) = \min(low(0), d(8)) = 0$
 $P(9) = 0$
 $Subarb(0) = 2$
Exploreaza (9)

- Explorează(u)
 - $d(u) = low(u) = \text{timp}++$; // inițializări
 - $c(u) = \text{gri}$;
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u$; $\text{subarb}(u)++$; // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $low(u) = \min\{low(u), low(v)\}$ // actualizare low
 - Dacă ($p(v) \neq \text{null}$ & $low(v) \geq d(u)$) $art(u) = 1$; // cazul 2 al teoremei
 - Altfel $low(u) = \min\{low(u), d(v)\}$ // actualizare low

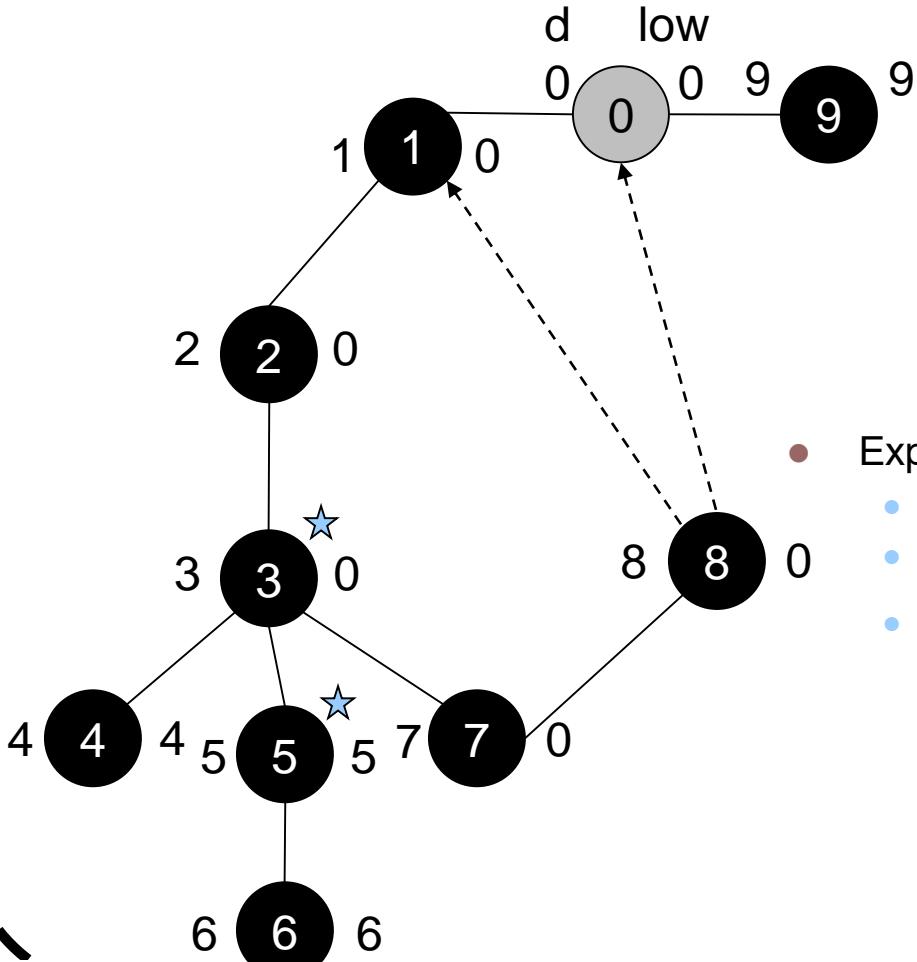
Exemplu rulare (27)



Low(9) = d(9) = 9
Timp = 10
C(9) = gri
revenire

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(u) \neq \text{null}$ & $\text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

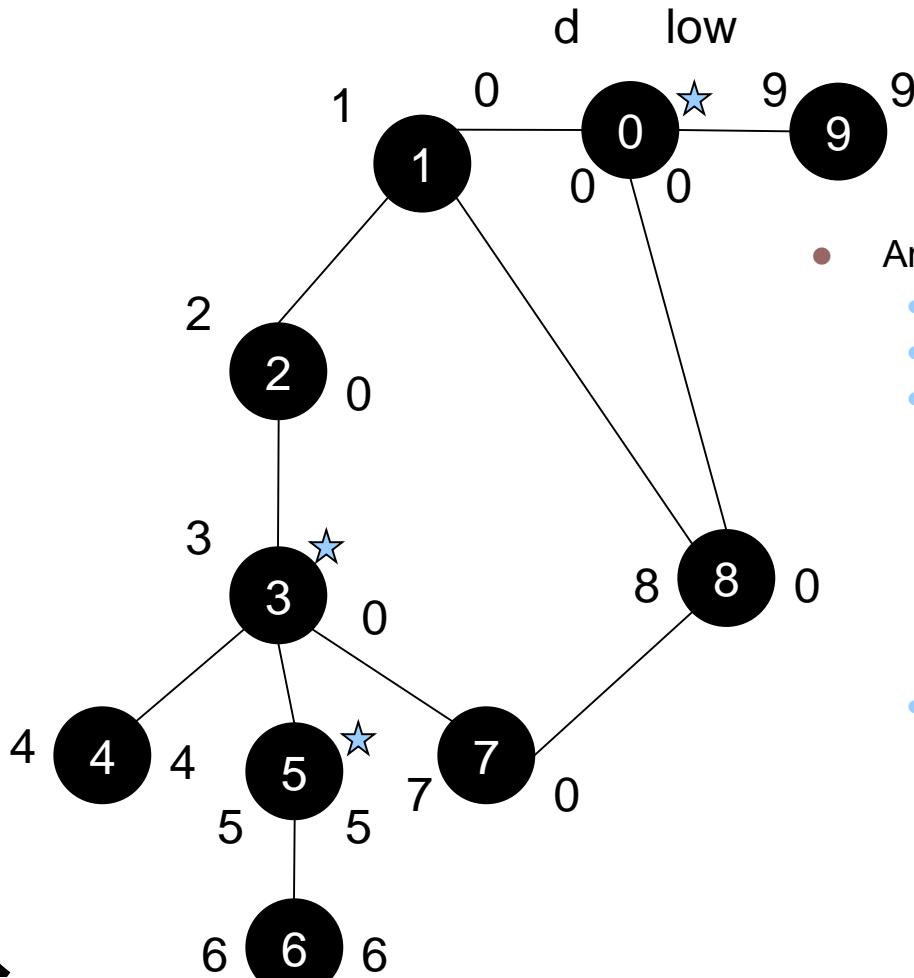
Exemplu rulare (28)



$\text{low}(0) = \min(\text{low}(0), \text{low}(9)) = 0$
 $P(0) = \text{null} \rightarrow \text{revenire}$

- Explorează(u)
 - $d(u) = \text{low}(u) = \text{timp}++;$ // inițializări
 - $c(u) = \text{gri};$
 - Pentru fiecare nod $v \in \text{succs}(u)$
 - Dacă ($c(v)$ e alb)
 - $p(v) = u;$ $\text{subarb}(u)++;$ // actualizare nr // subarburi dominați de u
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - Dacă ($p(u) \neq \text{null} \& \text{low}(v) \geq d(u)$) $\text{art}(u) = 1;$ // cazul 2 al teoremei
 - Altfel $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

Exemplu rulare (29)



$$\text{Subarb}(0) = 2 > 1 \rightarrow \text{art}(0) = 1$$

- Articulații (G)

- $V = \text{noduri}(G)$ // inițializări

- Timp = 0;

- Pentru fiecare ($u \in V$)

- $c(u) = \text{alb}$;

- $d(u) = 0$;

- $p(u) = \text{null}$;

- $\text{low}(u) = 0$;

- $\text{subarb}(u) = 0$; // reține numărul de subbarbi
// dominați de u

- $\text{art}(u) = 0$; // reține punctele de articulație

- Pentru fiecare ($u \in V$)

- Dacă $c(u)$ e alb

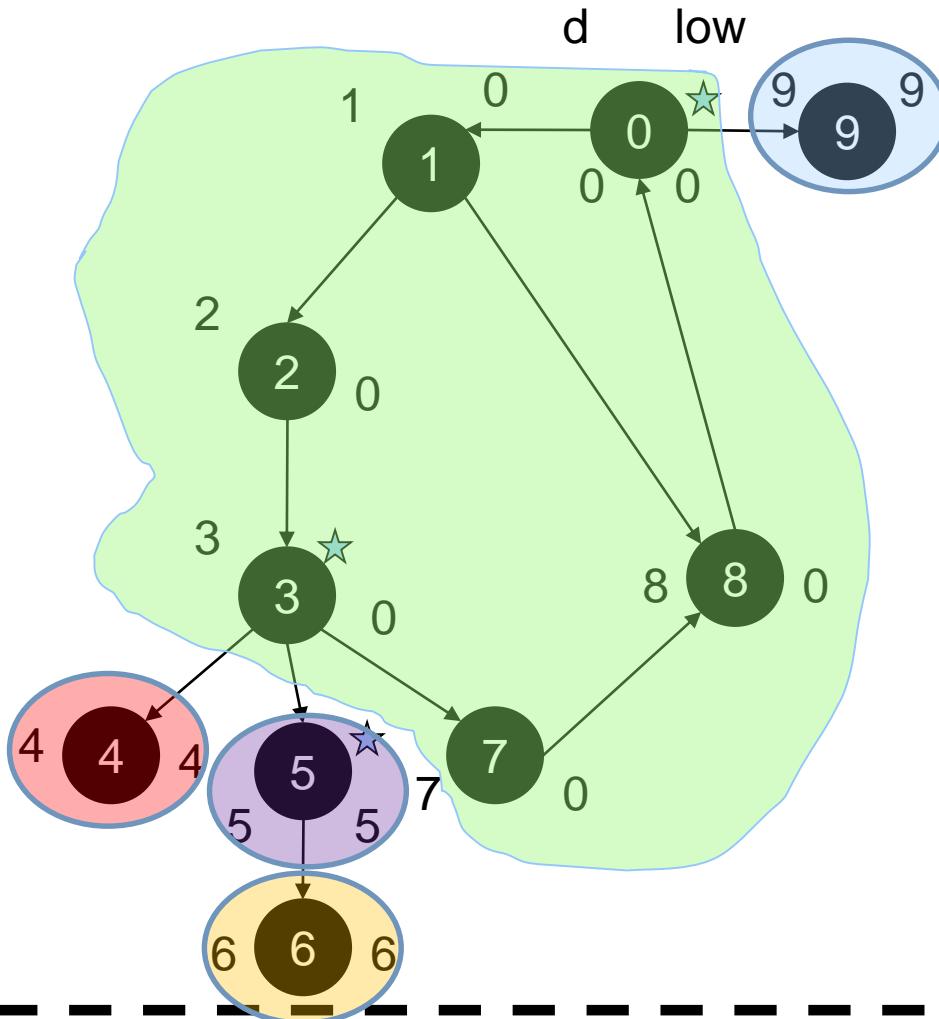
- Exploreaza(u);

- Dacă ($\text{subarb}(u) > 1$) // cazul în care u este
// rădăcina în arborele
 $\text{art}(u) = 1$ // DFS și are mai mulți subbarbi
// \rightarrow cazul 1 al teoremei

Algoritmul lui Tarjan adaptat pentru determinarea CTC

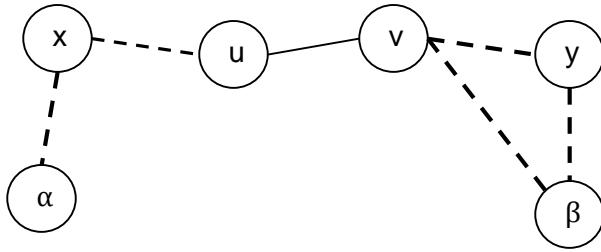
- `index = 0 // nivelul pe care este nodul în arborele DFS`
- `S = empty // se folosește o stivă care se initializează cu Ø`
- **Pentru fiecare** v din V
 - **Dacă** ($v.index$ e nedefinit) **atunci** `// se pornește DFS din fiecare nod pe care`
 - `Tarjan(v)` `// nu l-am vizitat încă`
- **Tarjan(v)**
 - `v.index = index // se setează nivelul nodului v`
 - `v.lowlink = index // reține strămoșul nodului v`
 - `index = index + 1 // incrementez nivelul`
 - `S.push(v) // introduc v în stivă`
 - **Pentru fiecare** (v, v') din E **// se prelucrează succesorii lui v**
 - **Dacă** ($v'.index$ e nedefinit sau v' e în S) **atunci** `// CTC deja identificate sunt ignoreate`
 - **Dacă** ($v'.index$ e nedefinit) **atunci** `Tarjan(v')` `// dacă nu a fost vizitat v' intru în recursivitate`
 - `v'.lowlink = min(v.lowlink, v'.lowlink) //actualizez strămoșul`
 - **Dacă** ($v'.lowlink == v.index$) **atunci** `// printez CTC începând de la coadă spre rădăcină`
 - `print "CTC:"`
 - **Repetă**
 - `v' = S.pop // extrag nodul din stiva și îl printez`
 - `print v'`
 - **Până când** ($v' == v$) `// până când extrag rădăcina`

Exemplu rularare (CTC)

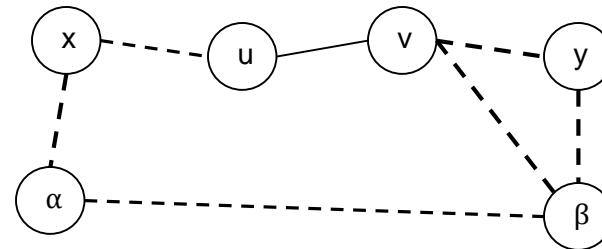


Punți

- Definiție: $G = (V, E)$, graf neorientat și $(u, v) \in E$. (u, v) este punte în G dacă $\exists x, y \in V$, $x \neq y$, a.î. $\forall x..y$ conține muchia (u, v) .



Orice drum $x..y$ trece prin (u, v)
 $\Rightarrow (u, v)$ este punte



(u, v) nu este punte

Algoritm punți (I)

- Punți(G)

- $V = \text{noduri}(G)$ // inițializări
- $\text{Timp} = 0;$
- **Pentru fiecare** nod u ($u \in V$)
 - $c(u) = \text{alb};$
 - $d(u) = 0;$
 - $p(u) = \text{null};$
 - $\text{low}(u) = 0;$
 - $\text{punte}(u) = 0;$ // înlocuieste: $\text{subarb}(u) = 0; \text{art}(u) = 0;$
- **Pentru fiecare** nod u ($u \in V$)
 - **Dacă** $c(u)$ e alb
 - Explorează(u)

Algoritm punți (II)

- Explorează(u)

- $d(u) = \text{low}(u) = \text{timp}++;$ // initializări
- $c(u) = \text{gri};$
- **Pentru fiecare** nod v ($v \in \text{succs}(u)$)

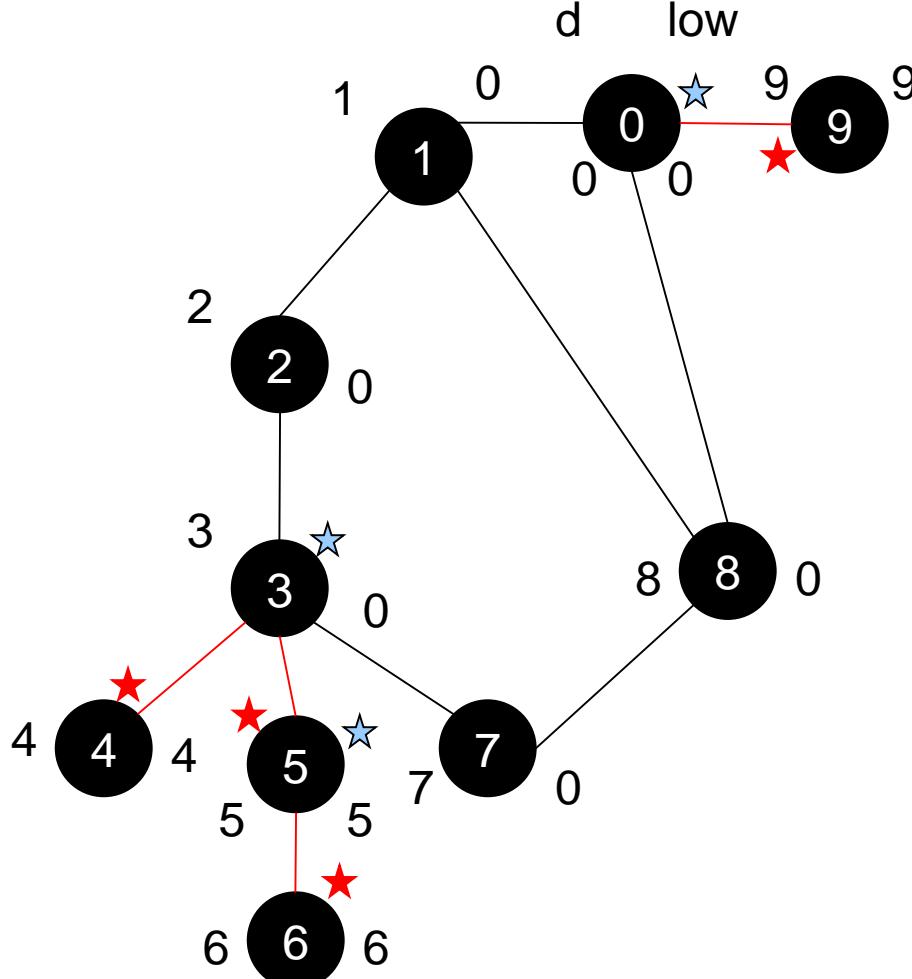
- **Dacă** $c(v)$ e alb

- $p(v) = u;$ // se elimină: $\text{subarb}(u)++;$
 - Explorează(v);
 - $\text{low}(u) = \min\{\text{low}(u), \text{low}(v)\}$ // actualizare low
 - **Dacă** ($\text{low}(v) > d(u)$) $\text{punte}(v) = 1;$
// în loc de: **Dacă**($p(u) \neq \text{null} \&& \text{low}(v) \geq d(u)$)

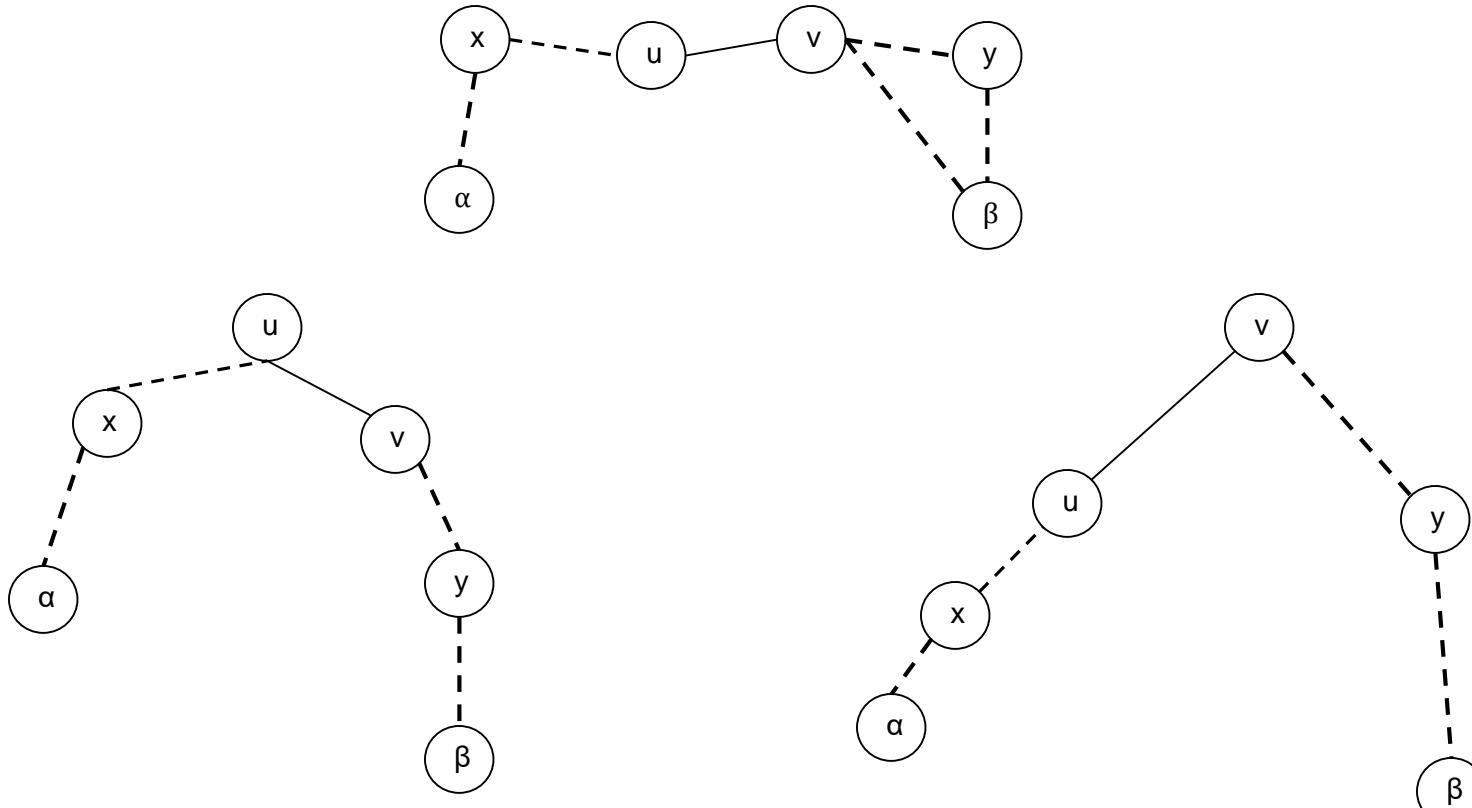
- **Altfel**

- **Dacă** ($p(u) \neq v$) $\text{low}(u) = \min\{\text{low}(u), d(v)\}$ // actualizare low

Exemplu rulare (Punți)



Exemplu



DFS din u ; puntea este detectată în v

DFS din v ; puntea este detectată în u

Drumuri de cost minim

- $G = (V, E)$ un graf, iar $w:E \rightarrow \mathbb{R}$ o funcție de cost asociată arcelor grafului ($w(u,v) = \text{costul arcului } (u,v)$).
- $\text{Cost}(u..v) = \text{costul drumului } u..v$ (este aditiv – costul drumului = suma costurilor arcelor).
- Variante:
 1. Drumuri punct – multipunct: pentru un nod dat $s \in V$, să se găsească un drum de cost minim de la s la $\forall u \in V$; **Dijkstra, Bellman-Ford**
 2. Drumuri multipunct – punct: pentru un nod dat $e \in V$, să se găsească un drum de cost minim de la $\forall u \in V$ la e ; **G^T și apoi 1**
 3. Drumuri punct – punct: pentru două noduri date u și $v \in V$, să se găsească un drum $u..v$ de cost minim; **Folosind 1**
 4. Drumuri multipunct – multipunct: $\forall u, v \in V$, să se găsească un drum $u..v$ de cost minim. **Floyd-Warshall**
 5. Drumuri de cost maxim!

Temă de gândire pentru acasă – posibil subiect de examen!

Optimalitatea drumurilor minime (I)

- Lemă 25.1 (Subdrumurile unui drum minim sunt drumuri optimale): $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. Fie $p = v_1v_2\dots v_k$ un drum optim de la v_1 la v_k . Atunci pentru orice i și j cu $1 \leq i \leq j \leq k$, subdrumul lui p de la v_i la v_j este un drum minim.
- Dem: Fie $p_{ij} = v_i..v_j$ subdrumul din p dintre v_i și v_j . $\rightarrow p = v_1..v_i..v_j..v_k \Rightarrow \text{cost}(p) = \text{cost}(v_1..v_i) + \text{cost}(v_i..v_j) + \text{cost}(v_j..v_k)$.
- Pp. prin absurd că $v_i..v_j$ nu e optim $\Rightarrow \exists p'$ a.î. $\text{cost}(p') < \text{cost}(v_i..v_j) \Rightarrow p$ nu e drum minim \rightarrow Contrazice ipoteza $\rightarrow p_{ij}$ este drum minim.

Optimalitatea drumurilor minime (II)

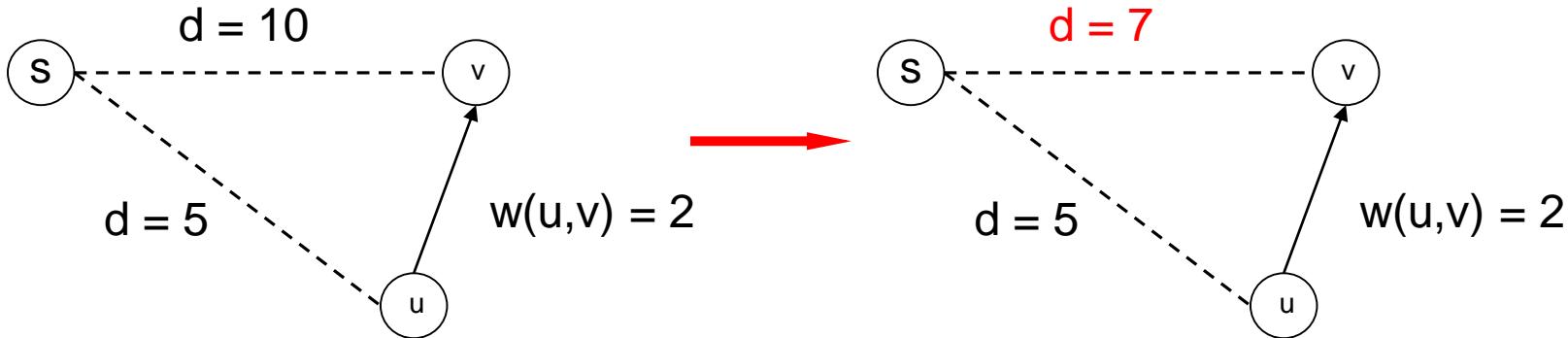
- **Corolar 25.2:** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată.
Fie $p = s..uv$ un drum optim de la s la v . Atunci costul optim al acestui drum poate fi scris ca $\delta(s,v) = \delta(s,u) + w(u,v)$.
- **Dem:** Conform teoremei anterioare, $s..u$ e un drum optim
 \Rightarrow cost $(s..u) = \delta(s,u)$.
- **Lemă 25.3:** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată.
 $\forall (u,v) \in E$ avem $\delta(s,v) \leq \delta(s,u) + w(u,v)$.
- **Dem:** Orice drum optim are costul mai mic ca al oricărui alt drum.

Drumuri minime de sursă unică

- Sunt concepuți pentru **grafuri orientate**.
- Bazați pe **algoritmi Greedy**.
- Se pornește de la nodul de start și pe baza unui optim local, drumurile sunt extinse și optimizate până la soluția finală.
- Notații:
 - $d[v]$ = costul drumului descoperit $s..v$;
 - $\delta(u,v)$ = costul drumului optim $u..v$; $\delta(u,v)=\infty$ dacă $v \notin R(u)$;
 - $p(v)$ = predecesorul lui v pe drumul $s..v$.

Drumuri minime de sursă unică

- Relaxarea arcelor \rightarrow dacă $d[v] > d[u] + w(u,v)$, atunci actualizează $d[v]$.



- Exemple: Dijkstra și Bellman–Ford.

Algoritmul lui Dijkstra (I)

- Folosește o coadă de priorități în care se adaugă nodurile în funcție de distanță cunoscută în momentul respectiv de la s până la nod.
- Se folosește **NUMAI** pentru costuri pozitive ($w(u,v) > 0, \forall u,v \in V$).
- Dijkstra_generic (G,s)
 - $V =$ nodurile lui G
 - **Cât timp** ($V \neq \emptyset$)
 - $u =$ nod din V cu $d[u]$ min
 - $V = V - \{u\}$
 - **Pentru fiecare** ($v \in$ succesorii lui u) $\text{relaxare_arc}(u,v)$
// optimizare drum s..v pentru v ∈ succesorilor lui u

Relaxarea arcelor (I)

- Lemă 25.5: $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. $\forall v \in V$, $d[v]$ obținut de algoritmul lui Dijkstra respectă $d[v] \geq \delta(s, v)$. În plus, odată atinsă valoarea $\delta(s, v)$, ea nu se mai modifică.
- Dem:
 - $\forall v \in V, v \notin R(s) \rightarrow d[v] = \delta(s, v) = \infty; d[s] = \delta(s, s) = 0$ (initializare)
 - Pt $v \in R(s)$, initializare $\rightarrow d[v] = \infty \geq \delta(s, v)$. Dem. prin reducere la absurd că după oricâte relaxări, relația se menține. Fie v primul vârf pentru care relaxarea (u, v) determină $d[v] < \delta(s, v) \rightarrow$ după relaxarea (u, v) : $d[u] + w(u, v) = d[v] < \delta(s, v) \leq \delta(s, u) + w(u, v) \rightarrow d[u] < \delta(s, u)$. Dar relaxarea nu modifică $d[u]$, iar v e primul pentru care $d[v] < \delta(s, v)$. Contrazice presupunerea! $\Rightarrow d[v] \geq \delta(s, v), \forall v \in V$
 - Cum $d[v] \geq \delta(s, v) \Rightarrow$ odată ajuns la $d[v] = \delta(s, v)$, ea nu mai scade. Cum relaxarea nu crește valorile $\Rightarrow d[v]$ nu se mai modifică.

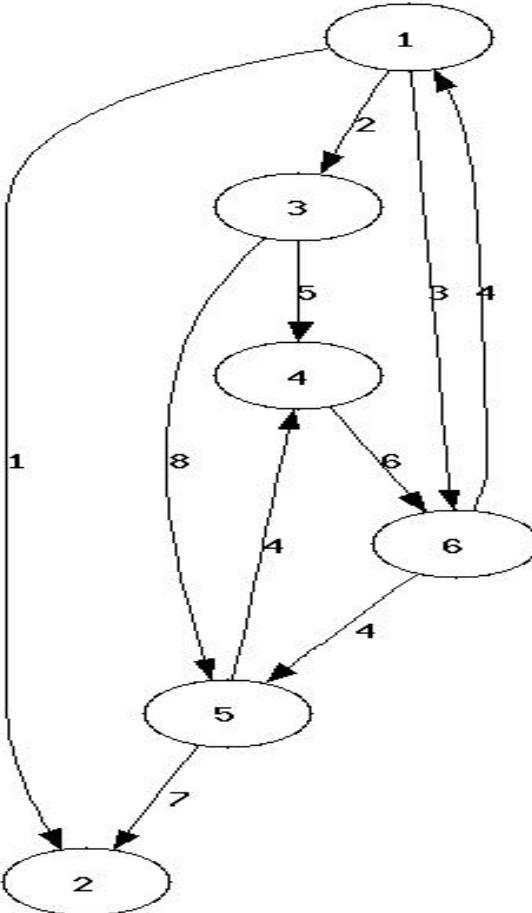
Relaxarea arcelor (II)

- Lemă 25.7: $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată.
Fie $p = s \dots uv$ un drum optim de la s la v . Dacă $d[u] = \delta(s, u)$ la un moment dat, atunci începând cu momentul imediat următor relaxării arcului (u, v) avem $d[v] = \delta(s, v)$
- Dem:
 - Dacă înainte de relaxare $d[v] > d[u] + w(u, v)$, prin relaxare $\rightarrow d[v] = d[u] + w(u, v)$. Altfel, $d[v] \leq d[u] + w(u, v) \Rightarrow$ după relaxare avem $d[v] \leq d[u] + w(u, v)$.
 - Cum $d[u] = \delta(s, u)$ și relaxarea (u, v) nu modifică $d[u] \Rightarrow d[v] \leq d[u] + w(u, v) = \delta(s, u) + w(u, v) = \delta(s, v)$ (conf. Corolar 25.2) $\rightarrow d[v] = \delta(s, v)$

Algoritmul lui Dijkstra (II)

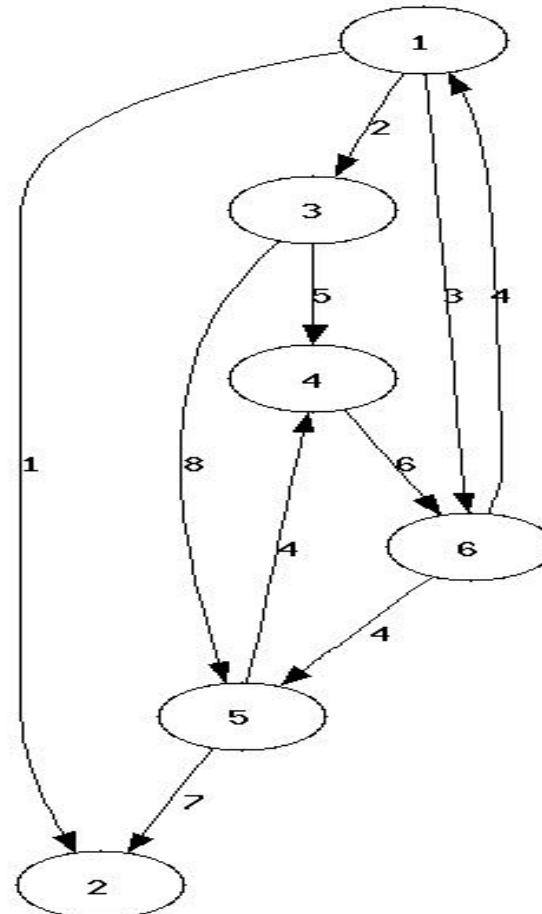
- **Dijkstra(G, s)**
 - **Pentru fiecare** ($u \in V$) // initializări
 - $d[u] = \infty$; $p[u] = \text{null}$;
 - $d[s] = 0$;
 - $Q = \text{construiește_coada}(V)$ // coadă cu priorități
 - **Cât timp** ($Q \neq \emptyset$)
 - $u = \text{ExtrageMin}(Q)$; // extrage din V elementul cu $d[u]$ minim
 - // $Q = Q - \{u\}$ – se execută în cadrul lui `ExtrageMin`
 - **Pentru fiecare** ($v \in Q$ și v din succesorii lui u)
 - **Dacă** ($d[v] > d[u] + w(u,v)$)
 - $d[v] = d[u] + w(u,v)$ // actualizez distanța
 - $p[v] = u$ // și părintele

Exemplu (I)



Exemplu (II)

- $d[1] = 0;$
- (1): $d[2] = 1; d[3] = 2; d[6] = 3;$
- (2): $d[4] = 7; d[5] = 10;$
- (3): $d[5] = 7;$
- Dijkstra(G, s)
 - Pentru fiecare ($u \in V$)
 - $d[u] = \infty; p[u] = \text{null};$
 - $d[s] = 0;$
 - $Q = \text{construiește_coada}(V) // coadă cu priorități$
 - Cât timp ($Q \neq \emptyset$)
 - $u = \text{ExtragăMin}(Q); // extrage din V elementul cu } d[u] \\ // minim$
 - $// Q = Q - \{u\} - \text{se execută în cadrul lui ExtragăMin}$
 - Pentru fiecare ($v \in Q$ și v din succesorii lui u)
 - Dacă ($d[v] > d[u] + w(u,v)$)
 - $d[v] = d[u] + w(u,v) // actualizez distanța$
 - $p[v] = u // și părintele$



Complexitate Dijkstra

- Depinde de ExtragEMin – coadă cu priorități.
- Operații ce trebuie realizate pe coadă + frecvența lor:
 - insert – V;
 - delete – V;
 - conține? – E;
 - micsorează_val – E;
 - este_vidă? – V.
- Dijkstra(G,s)
 - Pentru fiecare $(u \in V)$
 - $d[u] = \infty$; $p[u] = \text{null}$;
 - $d[s] = 0$;
 - $Q = \text{construieste_coada}(V)$ // coadă cu priorități
 - Cât timp ($Q \neq \emptyset$)
 - $u = \text{ExtragEMin}(Q)$; // extrage din V elementul cu $d[u]$ minim
 - // $Q = Q - \{u\}$ – se execută în cadrul lui ExtragEMin
 - Pentru fiecare $(v \in Q \text{ și } v \text{ din succesorii lui } u)$
 - Dacă ($d[v] > d[u] + w(u,v)$)
 - $d[v] = d[u] + w(u,v)$ // actualizez distanța
 - $p[v] = u$ // si părintele

Implementare cu vectori

- Costuri:

- insert – $1 * V = V$;
- delete – $V * V = V^2$ (necesită căutarea minimului);
- conține? – $1 * E = E$;
- micșorează_val – $1 * E = E$;
- este_vidă? – $1 * V = V$;

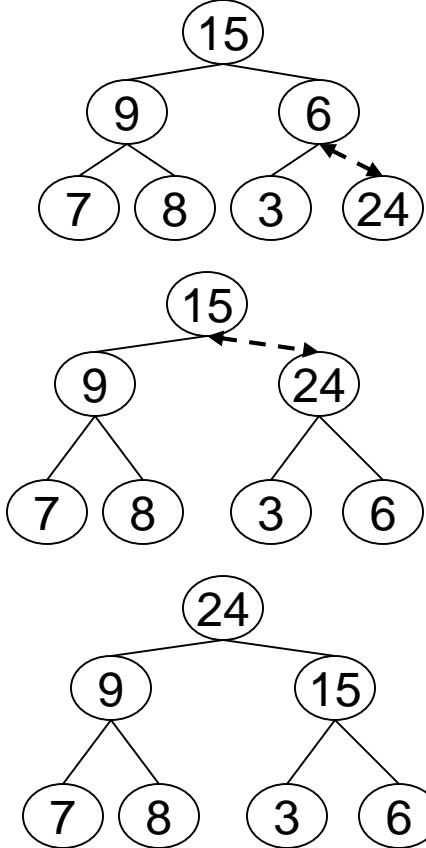
- Cea mai bună metodă pentru grafuri “dese” ($E \approx V^2$)!

Implementare cu heap binar

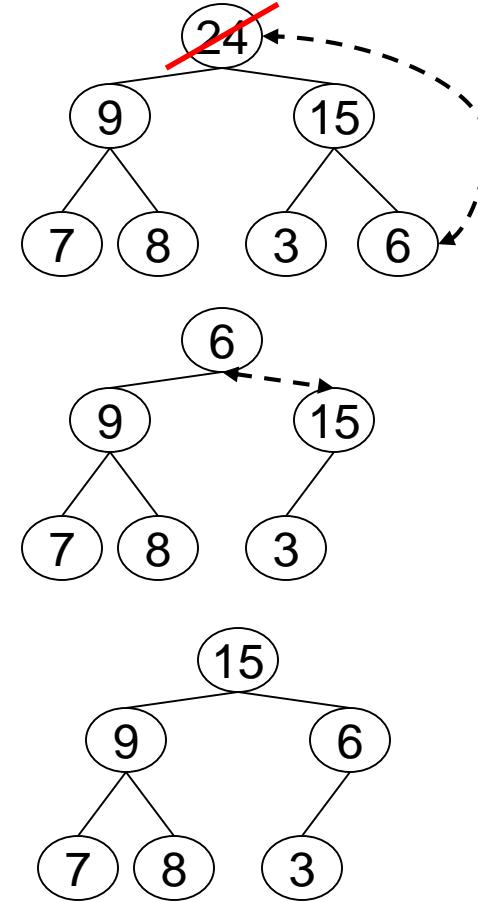
- Heap binar – structură de date de tip arbore binar + 2 constrângeri:
 - Fiecare nivel este complet; ultimul se umple de la stânga la dreapta;
 - $\forall u \in \text{Heap}; u \geq \text{răd(st}(u))$ și $u \geq \text{răd(dr}(u))$ (u este \geq decât ambii copii ai săi) unde \geq este o relație de ordine pe mulțimea pe care sunt definite elementele heapului.

Operatii pe Heap Binar

insert



delete



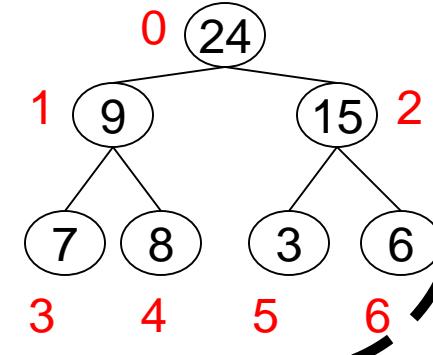
PA



Implementare Heap Binar

- Implementare folosind vectori.
- Poziție[i] = unde se găsește în indexul de valori elementul de pe poziția i din heap.
- Reverse[i] = unde se găsește în heap elementul de pe poziția i din valoare.
- Implementare disponibila la [3].

Index	0	1	2	3	4	5	6
Valoare	7	6	15	8	24	9	3
Poziție							
Reverse							



Heap Binar

- Costuri:

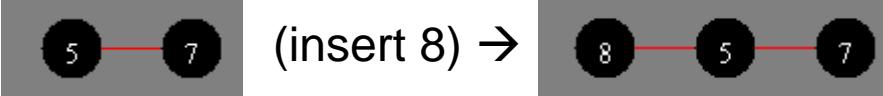
- insert – $\log V * V = V \log V$;
- delete – $\log V * V = V \log V$;
- conține? – $1 * E = E$;
- micșorează_val – $\log V * E = E \log V$;
- este_vidă? – $1 * V = V$.

- Eficient dacă graful are arce puține comparativ cu numărul de noduri.

Heap Fibonacci

- Poate fi format din mai mulți arbori.
- Cheia unui părinte \leq cheia oricărui copil.
- Fiind dat un nod u și un heap H :
 - $p(u)$ – părintele lui u ;
 - $\text{copil}(u)$ – legătura către unul din copiii lui u ;
 - $\text{st}(u)$, $\text{dr}(u)$ – legătura la frații din stânga și din dreapta (cei de pe primul nivel sunt legați între ei astfel);
 - $\text{grad}(u)$ – numărul de copii ai lui u ;
 - $\text{min}(H)$ – cel mai mic nod din H ;
 - $n(H)$ – numărul de noduri din H .

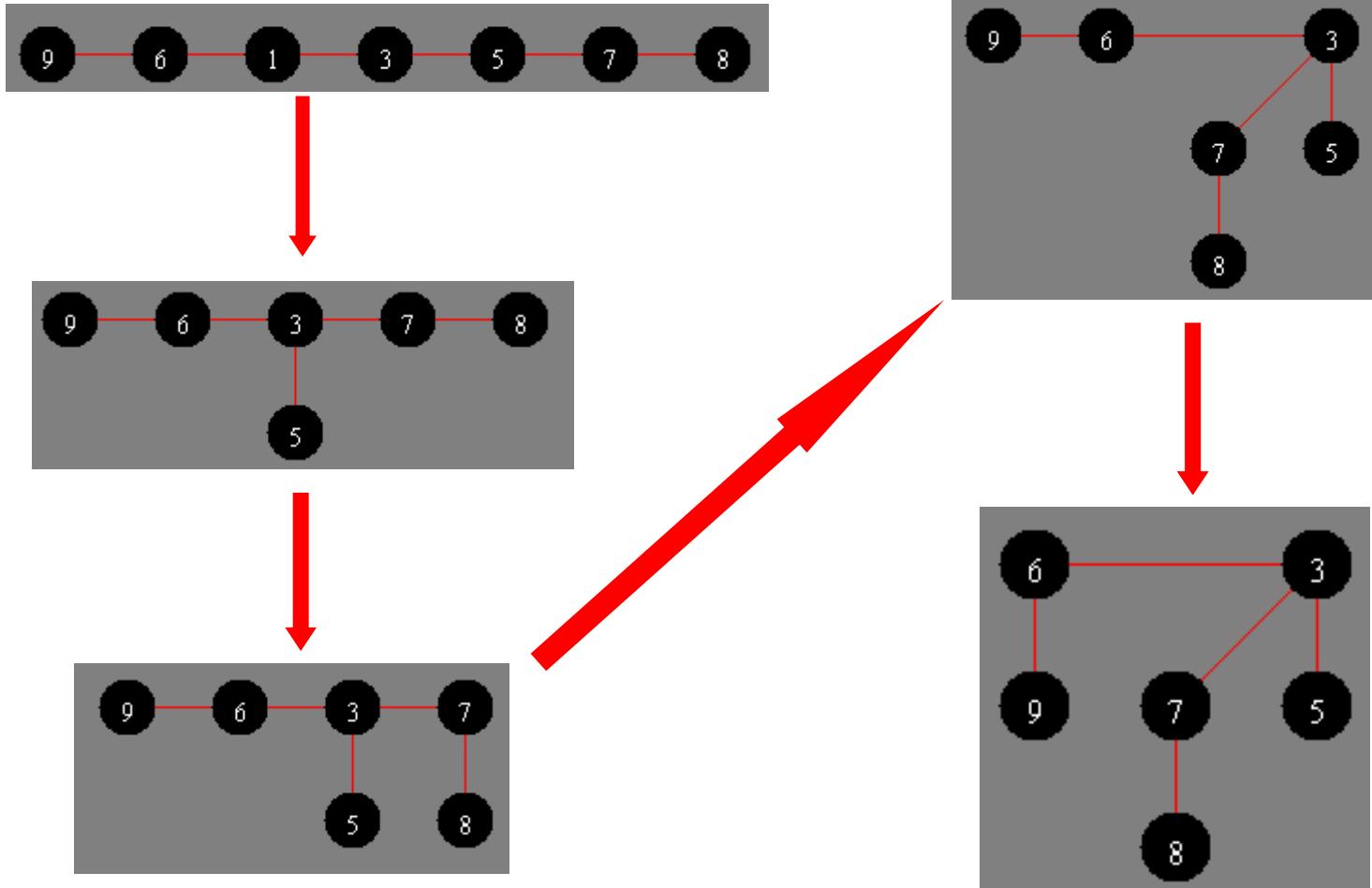
Operatii Heap Fibonacci

- Inserare nod – $O(1)$
 - construiește un nou arbore cu un singur nod
- Min – accesibil direct - $\min(H) - O(1)$
- ExtrageMin $O(\log n)$ – **cost amortizat!**
 - Mută copiii minimului pe prima coloană;
 - **Consolidează** heap-ul.

Operatii Heap Fibonacci

- Consolidare Heap
 - Cât timp există 2 arbori cu grade egale $\text{Arb}(x)$ și $\text{Arb}(y)$, $x < y$:
 - $\text{Arb}(y)$ transformat în copil al lui x ;
 - $\text{grad}[x] ++$;
- Applet și implementare disponibile la [4].

Consolidare Heap



Costuri Heap Fibonacci

- Costuri:
 - insert – $1 * V = V$;
 - delete – $\log V * V = V \log V$ (amortizat!);
 - micșorează_val – $1 * E = E$;
 - este_vidă? – $1 * V = V$.
- Cea mai rapidă structură dpdv teoretic.

Concluzii Dijkstra

- Implementarea trebuie realizată în funcție de tipul grafului pe care lucrăm:
 - vectori pentru grafuri “dese” - $O(V^2)$;
 - heap pentru grafuri “rare”: HB - $O(E \log V)$, HF - $O(V \log V+E)$
- Heapul Fibonacci este mai eficient decât heapul binar dar mai dificil de implementat.

ÎNTREBĂRI?

Proiectarea Algoritmilor

Curs 9 – Drumuri de cost minim

Bibliografie

- [1] R. Sedgewick, K. Wayne - Algorithms and Data Structures Fall 2007 – Curs Princeton -
<http://www.cs.princeton.edu/~rs/AlgsDS07/>
- [2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*

Obiective

- “Descoperirea” algoritmilor de identificare a drumurilor de cost minim.
- Recunoașterea caracteristicilor acestor algoritmi.

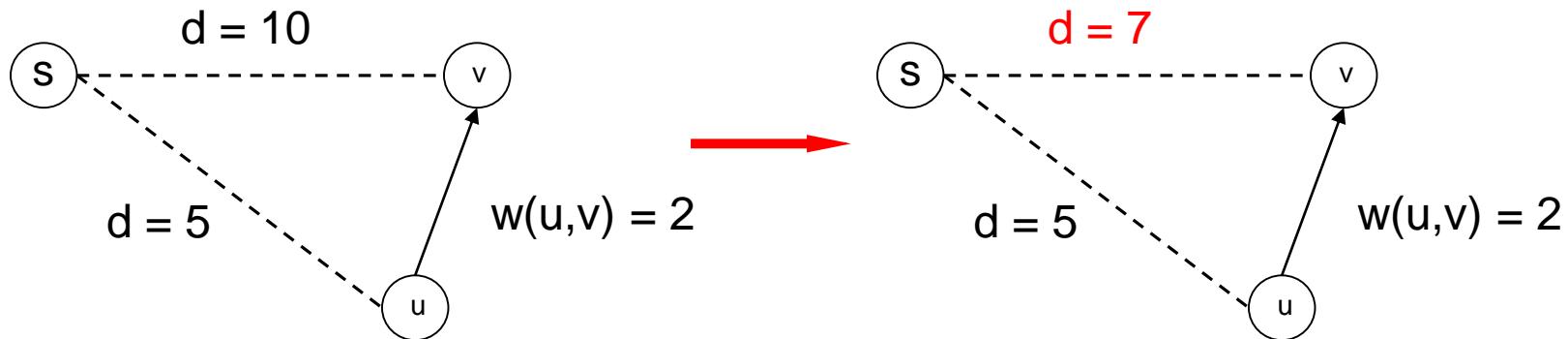
Reminder (I)

- $G = (V, E)$;
- $s \in V$ – nodul sursă;
- $w : E \rightarrow \mathbb{R}$ funcție de cost asociată arcelor grafului;
- $\text{cost}(u..v) = \text{costul drumului } u..v$ (aditiv);
- $d[v] = \text{costul drumului descoperit } s..v$;
- $\delta(u,v) = \text{costul drumului optim } u..v$:
 - $\delta(u,v) = \infty$ dacă $v \notin R(u)$
 - $\delta(u,v) = \sum w(x,y), (x,y) \in u..v$ ($u..v$ fiind drumul optim);
- $p[v] = \text{predecesorul lui } v \text{ pe drumul } s..v$.

Reminder (II)

- Relaxarea arcelor:

- Dacă $d[v] > d[u] + w(u,v)$, atunci
 - $d[v] = d[u] + w(u,v);$
 - $p[v] = u$



Concluzii Dijkstra (I)

- **Dijkstra(G, s)**
 - **Pentru fiecare nod u ($u \in V$)**
 - $d[u] = \infty$; $p[u] = \text{null}$;
 - $d[s] = 0$;
 - $Q = \text{construieste_coada}(V)$ // coadă cu priorități
 - **Cât timp ($Q \neq \emptyset$)**
 - $u = \text{ExtrageMin}(Q)$; // extrage din V elementul cu $d[u]$ minim
 - // $Q = Q - \{u\}$ – se execută în cadrul lui `ExtrageMin`
 - **Pentru fiecare nod v ($v \in Q$ și v din succesorii lui u)**
 - **Dacă** ($d[v] > d[u] + w(u,v)$)
 - $d[v] = d[u] + w(u,v)$ // actualizez distanța
 - $p[v] = u$ // și părintele

Complexitate?

Vectori - $O(V^2)$

HB - $O(E \log V)$

HF - $O(V \log V + E)$

Concluzii Dijkstra (II)

- Implementarea trebuie realizată în funcție de tipul grafului pe care lucrăm:
 - vectori pentru grafuri “dese”;
 - heap pentru grafuri “rare”.
- Heapul Fibonacci este mai eficient decât heapul binar dar mai dificil de implementat.

Corectitudine Dijkstra – Reminder(I)

- Lemă 25.1 (Subdrumurile unui drum minim sunt drumuri optimale): $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. Fie $p = v_1v_2\dots v_k$ un drum optim de la v_1 la v_k . Atunci pentru orice i și j cu $1 \leq i \leq j \leq k$, subdrumul lui p de la v_i la v_j este un drum minim.
- Corolar 25.2: $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. Fie $p = s\dots uv$ un drum optim de la s la v . Atunci costul optim al acestui drum poate fi scris ca $\delta(s, v) = \delta(s, u) + w(u, v)$.
- Lemă 25.3: $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. $\forall (u, v) \in E$ avem $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Corectitudine Dijkstra – Reminder(II)

- Lemă 25.5: $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. $\forall v \in V$, $d[v]$ obținut de algoritmul lui Dijkstra respectă $d[v] \geq \delta(s, v)$. În plus, odată atinsă valoarea $\delta(s, v)$, ea nu se mai modifică.
- Lemă 25.7: $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. Fie $p = s..uv$ un drum optim de la s la v . Dacă $d[u] = \delta(s, u)$ la un moment dat, atunci începând cu momentul imediat următor relaxării arcului (u, v) avem $d[v] = \delta(s, v)$.

Corectitudine Dijkstra

- **Teoremă.** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată **nenegativă**. La terminarea aplicării algoritmului Dijkstra pe acest graf plecând din sursa s vom avea $d[v] = \delta(s, v)$ pentru $\forall v \in V$.
- **Dem:** prin reducere la absurd se demonstrează că la scoaterea din Q a fiecărui nod v avem $d[v] = \delta(s, v)$ și egalitatea se menține și ulterior.
 - Pp. u e primul nod pt. care $d[u] \neq \delta(s, u)$ la scoaterea din Q . $u \neq s$ pt. că altfel $d[u] = \delta(s, u) = 0$ și $u \in R(s)$ pt. că altfel $d[u] = \delta(s, u) = \infty$. \Rightarrow La scoaterea lui u din Q , există drum $s..u$ și fie p drumul optim $s..u$ a.î. $p = s..xy..u$, unde $x \notin Q$ iar $y \in Q$.
 - (Dacă nu impunem $y \in Q$, atunci toate nodurile până la u au fost scoase din Q și u a rămas în vârful cozii. Cum u e primul nod pentru care $d[u] \neq \delta(s, u)$, iar calea e optimă, avem situația din [Lema 25.7](#) \Rightarrow La scoaterea lui u din Q , $d[u] = \delta(s, u)$.)
 - Cum u e primul nod pt. care $d[u] \neq \delta(s, u) \Rightarrow d[x] = \delta(s, x)$ la momentul extragerii lui u din $Q \rightarrow d[y] = \delta(s, y)$ prin relaxarea (x, y) (conf. [Lema 25.7](#)).
 - y precede u pe drumul $p \Rightarrow d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$ (conf. [Lema 25.5](#)).
 - Cum $y \in Q$ la momentul scoaterii lui u din $Q \Rightarrow d[u] \leq d[y]$
 - $\Rightarrow d[y] = \delta(s, y) = \delta(s, u) = d[u] -$ Contrazice ipoteza! $\Rightarrow d[u] = \delta(s, u)$ și conf. [Lema 25.5](#), egalitatea se menține și ulterior.

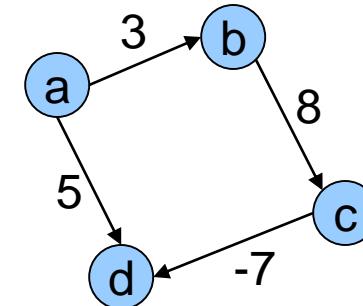
Problema Dijkstra (I)

- **Dijkstra(G, s)**
 - **Pentru fiecare nod u ($u \in V$)**
 - $d[u] = \infty$; $p[u] = \text{null}$;
 - $d[s] = 0$;
 - $Q = \text{construieste_coada}(V)$ // coadă cu priorități
 - **Cât timp ($Q \neq \emptyset$)**
 - $u = \text{ExtrageMin}(Q)$; // extrage din V elementul cu $d[u]$ minim
 - // $Q = Q - \{u\}$ – se execută în cadrul lui `ExtrageMin`
 - **Pentru fiecare nod v ($v \in Q$ și v din succesorii lui u)**
 - **Dacă** ($d[v] > d[u] + w(u,v)$)
 - $d[v] = d[u] + w(u,v)$ // actualizez distanța
 - $p[v] = u$ // și părintele

Problemă Dijkstra (II)

- Exemplu rulare:

- $d[a] = 0; d[b] = d[c] = d[d] = \infty$
- $d[b] = 3; d[d] = 5;$
- $d[c] = 11;$



- **d este extras din coadă!** În momentul extragerii din coadă distanța pană la nodul d se consideră a fi calculată și a fi optimă.
- Se extrage nodul c; $d[d]$ nu va mai fi actualizată – nodul d fiind deja eliminat din coadă.

- → Algoritmul nu funcționează pentru grafuri ce conțin arce de cost negativ!

Exemplu practic – arce de cost negativ (I)

Currency conversion. Given currencies and exchange rates, what is best way to convert one ounce of gold to US dollars?

- 1 oz. gold $\Rightarrow \$327.25$. [208.10×1.5714]
- 1 oz. gold $\Rightarrow £208.10 \Rightarrow \327.00 .
- 1 oz. gold $\Rightarrow 455.2$ Francs $\Rightarrow 304.39$ Euros $\Rightarrow \$327.28$.

[$455.2 \times .6677 \times 1.0752$]

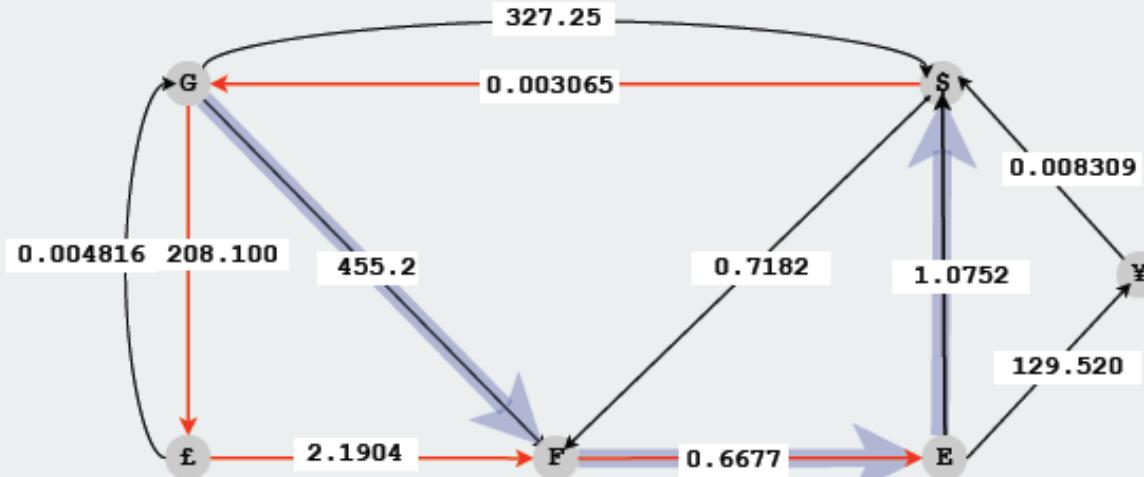
Currency	£	Euro	¥	Franc	\$	Gold
UK Pound	1.0000	0.6853	0.005290	0.4569	0.6368	208.100
Euro	1.4599	1.0000	0.007721	0.6677	0.9303	304.028
Japanese Yen	189.050	129.520	1.0000	85.4694	120.400	39346.7
Swiss Franc	2.1904	1.4978	0.011574	1.0000	1.3929	455.200
US Dollar	1.5714	1.0752	0.008309	0.7182	1.0000	327.250
Gold (oz.)	0.004816	0.003295	0.0000255	0.002201	0.003065	1.0000

*slide din cursul de algoritmi de la Princeton – Sedgewick&Wayne[1]

Exemplu practic – arce de cost negativ (II)

Graph formulation.

- Vertex = currency.
- Edge = transaction, with weight equal to exchange rate.
- Find path that maximizes **product** of weights.



*slide din cursul de algoritmi de la Princeton – Sedgewick&Wayne[1]

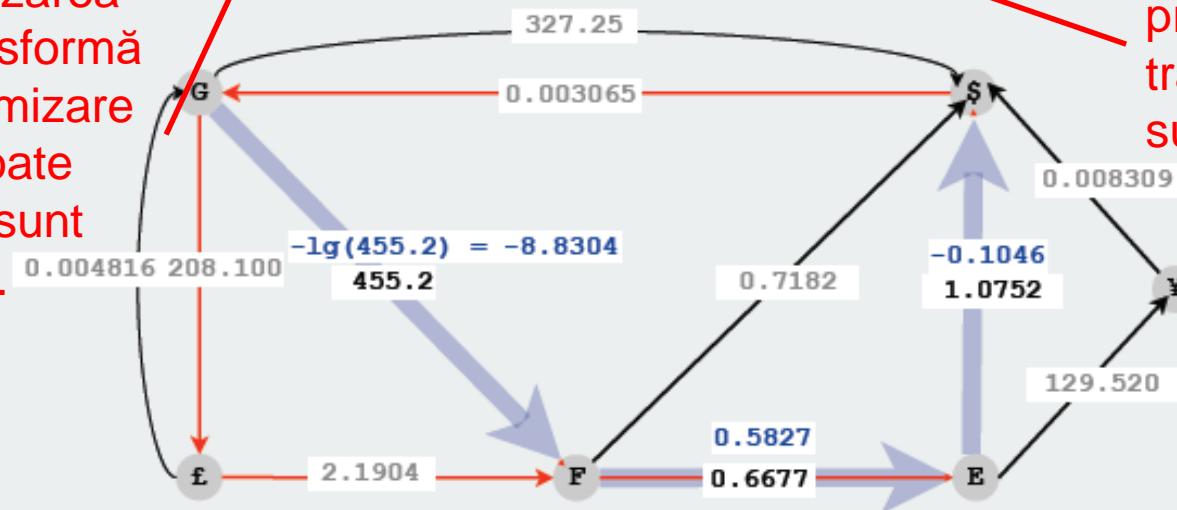
Exemplu practic – arce de cost negativ (III)

Reduce to shortest path problem by taking logs

- Let $\text{weight}(v-w) = -\lg$ (exchange rate from currency v to w)
- multiplication turns to addition
- Shortest path with costs c corresponds to best exchange sequence.

Maximizarea se transformă în minimizare dacă toate arcele sunt negative.

Prin logaritmare produsul se transformă în sumă.

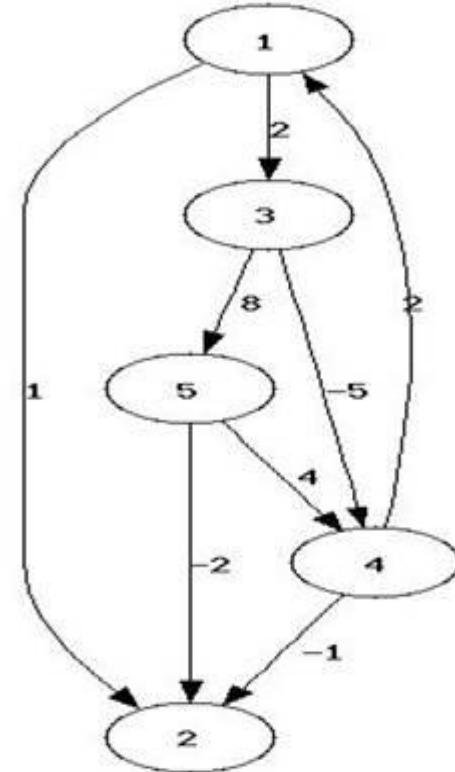


*slide din cursul de algoritmi de la Princeton – Sedgewick&Wayne[1]

Cicluri de cost negativ

$$\delta(u, v) = \begin{cases} \sum w(x, y), (x, y) \in u..v \\ (u..v \text{ fiind drumul optim}); \\ \infty, \text{ dacă nu există drum } u..v. \end{cases}$$

- Dacă există pe drumul $u..v$ un ciclu de cost negativ $x..y \rightarrow$
 - $\delta(u, v) = \delta(u, v) + \text{cost}(x..y) < \delta(u, v)$
 - \rightarrow valoarea lui $\delta(u, v)$ va scădea continuu \rightarrow costul este $-\infty$
 - $\rightarrow \delta(u, v) = -\infty$

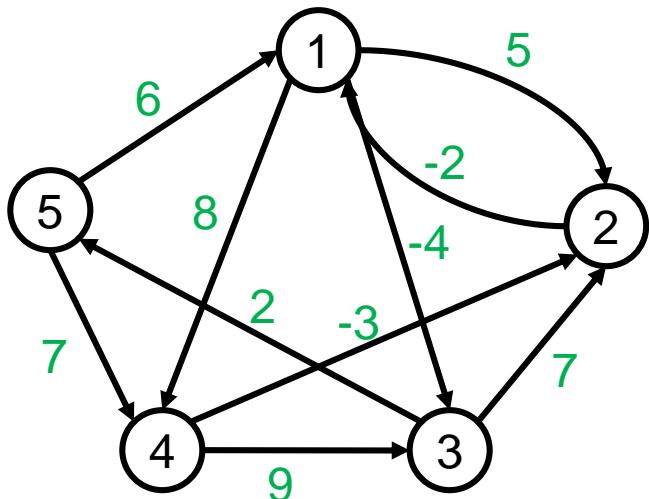


1-3-4 ciclu de cost negativ(-1) \rightarrow toate costurile din graf sunt $-\infty$

Algoritmul Bellman-Ford

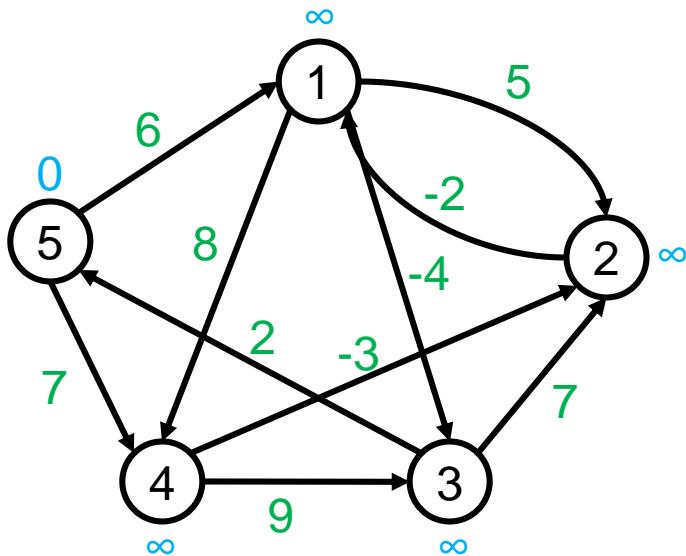
- **BellmanFord(G,s)** // $G=(V,E)$, s =sursa
 - Pentru fiecare nod v ($v \in V$) // inițializări
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță de la s la s
 - Pentru i de la 1 la $|V| - 1$ // pentru fiecare pas pornind din s
// spre restul nodurilor se încearcă construcția unor drumuri
// optime de dimensiune i
 - Pentru fiecare (u,v) din E
// pentru arcele ce pleacă de la nodurile deja considerate
 - Dacă $d[v] > d[u] + w(u,v)$ atunci // se relaxează arcele corespunzătoare
 - $d[v] = d[u] + w(u,v)$;
 - $p[v] = u$;
 - Pentru fiecare (u,v) din E
 - Dacă $d[v] > d[u] + w(u,v)$ atunci
 - Eroare ("ciclu negativ");

Exemplu Bellman-Ford (I)



- **BellmanFord(G, s)**
 - **Pentru fiecare** v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță pană la s
 - **Pentru i de la 1 la $|V| - 1$** // pt // fiecare pas de la s spre $V - s$
 - **Pentru fiecare** (u, v) din E // pt. // arcele ce pleacă de la nodurile // deja considerate
 - **Dacă** $d[v] > d[u] + w(u, v)$ atunci // se relaxează arcele // corespunzătoare
 - $d[v] = d[u] + w(u, v);$
 - $p[v] = u;$
 - **Pentru fiecare** (u, v) din E
 - **Dacă** $d[v] > d[u] + w(u, v)$ atunci
 - **Eroare** ("ciclu negativ");

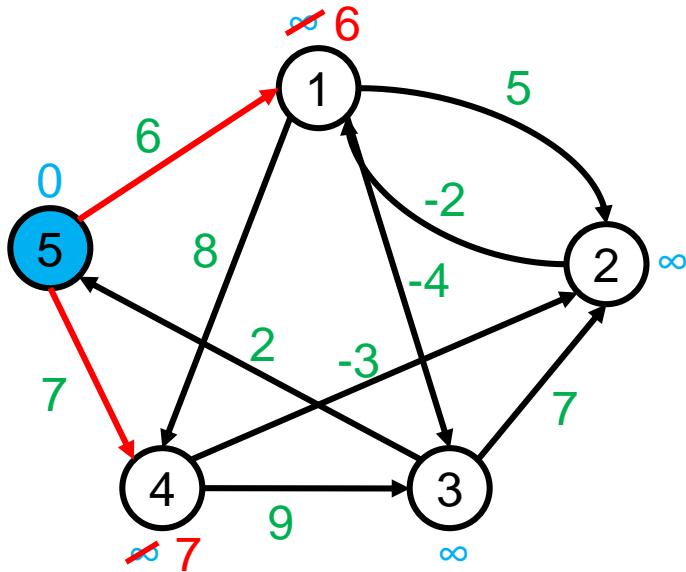
Exemplu Bellman-Ford (II)



$$d[1] = d[2] = d[3] = d[4] = d[5] = \infty$$
$$d[5] = 0$$

- **BellmanFord(G, s)**
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță pană la s
 - Pentru i de la 1 la $|V| - 1$ // pt // fiecare pas de la s spre $V - s$
 - Pentru fiecare (u, v) din E // pt. // arcele ce pleacă de la nodurile // deja considerate
 - Dacă $d[v] > d[u] + w(u, v)$ atunci // se relaxează arcele // corespunzătoare
 - $d[v] = d[u] + w(u, v)$;
 - $p[v] = u$;
 - Pentru fiecare (u, v) din E
 - Dacă $d[v] > d[u] + w(u, v)$ atunci
 - Eroare ("ciclu negativ");

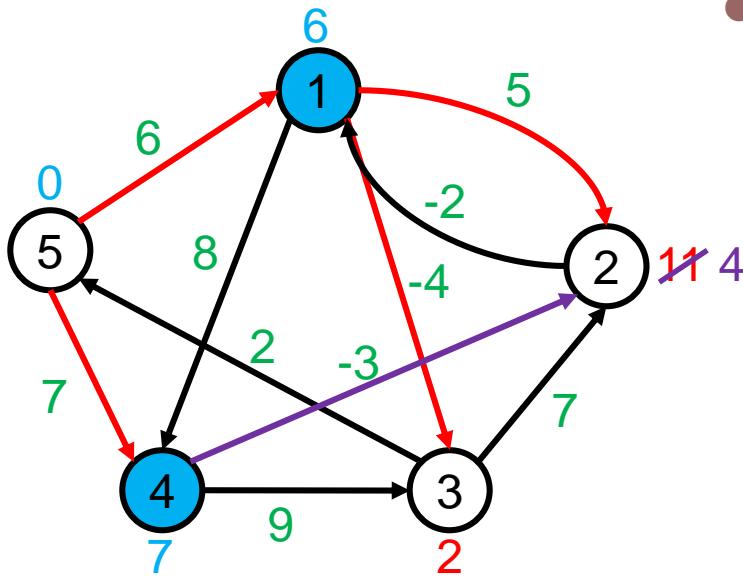
Exemplu Bellman-Ford (III)



Pas 1: relaxare $(5,1)$ și $(5,4)$
 $d[1] = 6$, $p[1] = 5$
 $d[4] = 7$, $p[4] = 5$

- **BellmanFord(G,s)**
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță pană la s
 - Pentru i de la 1 la $|V| - 1$ // pt fiecare pas de la s spre $V - s$
 - Pentru fiecare (u,v) din E // pt. arcele ce pleacă de la nodurile deja considerate
 - Dacă $d[v] > d[u] + w(u,v)$ atunci // se relaxează arcele corespunzătoare
 - $d[v] = d[u] + w(u,v);$
 - $p[v] = u;$
 - Pentru fiecare (u,v) din E
 - Dacă $d[v] > d[u] + w(u,v)$ atunci
 - Eroare ("ciclu negativ");

Exemplu Bellman-Ford (IV)



Pas 2: relaxare (1,2) și (1,3)

$d[2] = 11$, $p[2] = 1$

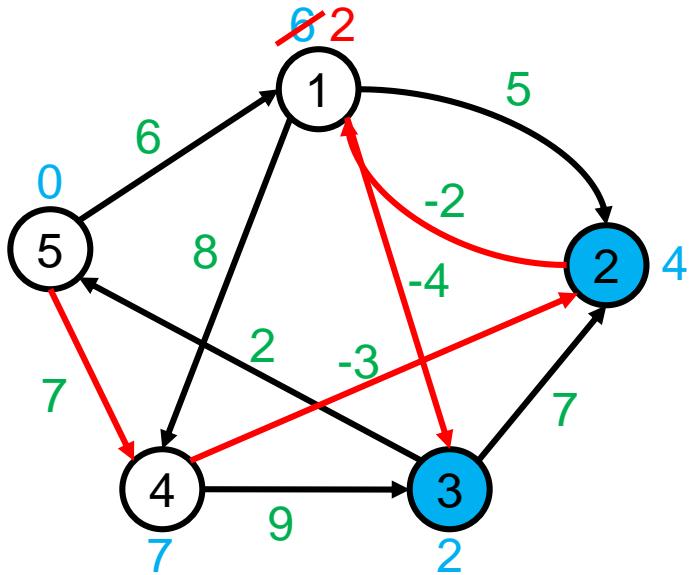
$d[3] = 2$, $p[3] = 1$

relaxare (4,2)

$d[2] = 4$, $p[2] = 4$

- **BellmanFord(G, s)**
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță pană la s
 - Pentru i de la 1 la $|V| - 1$ // pt // fiecare pas de la s spre $V - s$
 - Pentru fiecare (u, v) din E // pt. // arcele ce pleacă de la nodurile // deja considerate
 - Dacă $d[v] > d[u] + w(u, v)$ atunci // se relaxează arcele // corespunzătoare
 - $d[v] = d[u] + w(u, v);$
 - $p[v] = u;$
 - Pentru fiecare (u, v) din E
 - Dacă $d[v] > d[u] + w(u, v)$ atunci
 - Eroare ("ciclu negativ");

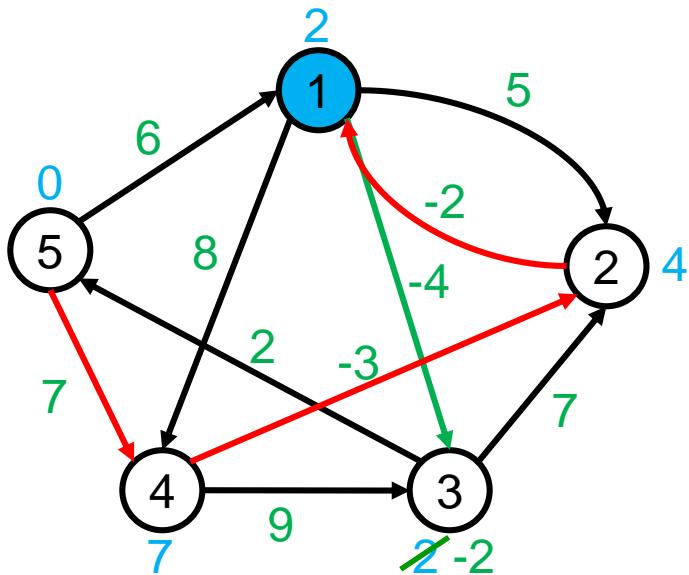
Exemplu Bellman-Ford (V)



Pas 3: relaxare (2,1)
 $d[1] = 2$, $p[1] = 2$

- **BellmanFord(G, s)**
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță pană la s
 - Pentru i de la 1 la $|V| - 1$ // pt fiecare pas de la s spre $V - s$
 - Pentru fiecare (u, v) din E // pt. arcele ce pleacă de la nodurile deja considerate
 - Dacă $d[v] > d[u] + w(u, v)$ atunci // se relaxează arcele corespunzătoare
 - $d[v] = d[u] + w(u, v)$;
 - $p[v] = u$;
 - Pentru fiecare (u, v) din E
 - Dacă $d[v] > d[u] + w(u, v)$ atunci
 - Eroare ("ciclu negativ");

Exemplu Bellman-Ford (VI)



Pas 4: relaxare $(1, 3)$
 $d[3] = -2$, $p[3] = 1$

- **BellmanFord(G, s)**
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță pană la s
 - Pentru i de la 1 la $|V| - 1$ // pt // fiecare pas de la s spre $V - s$
 - Pentru fiecare (u, v) din E // pt. // arcele ce pleacă de la nodurile // deja considerate
 - Dacă $d[v] > d[u] + w(u, v)$ atunci // se relaxează arcele // corespunzătoare
 - $d[v] = d[u] + w(u, v);$
 - $p[v] = u;$
 - Pentru fiecare (u, v) din E
 - Dacă $d[v] > d[u] + w(u, v)$ atunci
 - Eroare ("ciclu negativ");

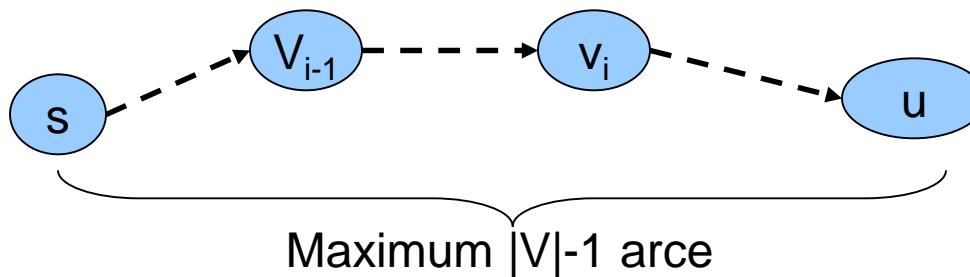
Complexitate Bellman-Ford

- cazul defavorabil:
 - Pentru i de la 1 la $|V| - 1$ $\overbrace{\hspace{10em}}$ V^*
 - Pentru fiecare (u, v) din E $\overbrace{\hspace{10em}}$ E
 - Dacă $d[v] > d[u] + w(u, v)$ atunci $\overbrace{\hspace{10em}}$ $O(VE)$
 - $d[v] = d[u] + w(u, v);$
 - $p[v] = u;$

Corectitudine Bellman-Ford (I)

- Lemă 25.12: $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată; dacă G nu conține ciclu de cost negativ atunci după $|V| - 1$ iterații ale relaxării fiecărui arc avem $d[v] = \delta(s, v)$ pentru $\forall v \in R(s)$.
- Dem prin inducție:
 - Fie $s = v_0, v_1 \dots v_k = u$ un drum minim în graf cu $k \leq |V| - 1$.

La pasul i va fi relaxat arcul v_{i-1}, v_i



Corectitudine Bellman-Ford (II)

- **Demonstrăm că** în pasul i : $d[v_i] = \delta(s, v_i)$ și se menține până la sfârșit.
- P_0 : (initializare) $\rightarrow d[s] = d[v_0] = 0 = \delta(s, s) = \delta(s, v_0)$ și conf. **Lema 25.5**, relația se menține până la sfârșit.
- $P_{i-1} \rightarrow P_i$:
 - P_{i-1} : $d[v_{i-1}] = \delta(s, v_{i-1})$,
 - În pasul i se relaxează arcul (v_{i-1}, v_i) , \Rightarrow conf. **Lema 25.7** $\Rightarrow d[v_i] = d[v_{i-1}] + \delta(v_{i-1}, v_i) = \delta(s, v_{i-1}) + \delta(v_i, v_{i-1}) = \delta(s, v_i)$. Conf. **Lema 25.5**, relația se menține până la sfârșit.
 - Cum $i \in (1, |V|-1)$ \rightarrow relația e adevarată pentru toate nodurile accesibile din $s \rightarrow d[v] = \delta(s, v)$, $\forall v \in R(s)$.

Corectitudine Bellman-Ford (III)

- **Teorema.** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată.
Algoritmul Bellman-Ford aplicat acestui graf plecând din sursa s nu returnează EROARE dacă G nu conține cicluri negative, iar la terminare $d[v] = \delta(s, v)$ pentru $\forall v \in V$. Dacă G conține cel puțin un ciclu negativ accesibil din s , atunci algoritmul întoarce EROARE.
- **Dem:** pe baza [Lemei 25.12](#).
 - Dacă \nexists ciclu negativ:
 - $d[v] = \delta(s, v) \quad \forall v \in R(s)$
 - $d[v] = \delta(s, v) = \infty, \forall v \notin R(s)$ (initializare)
 - $\rightarrow d[v] \leq d[u] + w(u, v) \rightarrow$ nu se întoarce eroare (conf. [Lema 25.3](#))
 - Dacă există ciclu negativ \rightarrow în cei $|V| - 1$ pași se scad costurile drumurilor, iar în final ciclul se menține \rightarrow Eroare

Optimizări Bellman-Ford

● Observație!

- Dacă $d[v]$ nu se modifică la pasul i atunci nu trebuie sa relaxăm niciunul din arcele care pleacă din v la pasul $i + 1$.
- => păstrăm o coadă cu vârfurile modificate (o singură copie).

Bellman-Ford optimizat

- BellmanFordOpt(G, s)
 - Pentru fiecare nod v ($v \in V$)
 - $d[v] = \infty;$
 - $p[v] = \text{null};$
 - $\text{marcat}[v] = \text{false}; //$ marcăm nodurile pentru care am făcut relaxare
 - $Q = \emptyset; //$ coadă cu priorități
 - $d[s] = 0;$ $\text{marcat}[s] = \text{true};$ Introdu(Q, s);
 - Cât timp ($Q \neq \emptyset$)
 - $u = \text{ExtragMin}(Q);$ $\text{marcat}[u] = \text{false}; //$ extrag minimul
 - Pentru fiecare nod v (v din succesorii lui u)
 - Dacă $d[v] > d[u] + w(u,v)$ atunci // relaxez arcele ce pleacă din u
 - $d[v] = d[u] + w(u,v);$
 - $p[v] = u;$
 - Dacă ($\text{marcat}[v] == \text{false}$) { $\text{marcat}[v] = \text{true};$ Introdu(Q, v);}

● Observație: nu mai detectează cicluri negative!

Floyd-Warshall (Roy-Floyd)

- Algoritm prin care se calculează distanțele minime între oricare 2 noduri dintr-un graf (drumuri optime multipunct-multipunct).
- Exemplu clasic de programare dinamică.
- Idee: la pasul k se calculează cel mai bun cost între u și v folosind cel mai bun cost $u..k$ și cel mai bun cost $k..v$ calculat până în momentul respectiv.
- Se aplică pentru grafuri ce nu conțin cicluri de cost negativ.

Notări

- | • $G = (V, E)$; $V = \{1, 2, \dots, n\}$;
- | • $w : V \times V \rightarrow \mathbb{R}$; $w(i, i) = 0$; $w(i, j) = \infty$ dacă $(i, j) \notin E$;
- | • $d^k(i, j) = \text{costul drumului } i..j \text{ construit astfel încât drumul (nodurile intermediare) trece doar prin noduri din multimea } \{1, 2, \dots, k\}$;
- | • $\delta(i, j) = \text{costul drumului optim } i..j$; $\delta(i, j) = \infty$ dacă $\nexists i..j$;
- | • $\delta^k(i, j) = \text{costul drumului optim } i..j \text{ ce trece doar prin noduri intermediare din multimea } \{1, 2, \dots, k\}$; $\delta^k(i, j) = \infty$ dacă $\nexists i..j$ cu aceasta proprietate;
- | • $p^k(i, j) = \text{predecesorul lui } j \text{ pe drumul } i..j \text{ ce trece doar prin noduri din multimea } \{1, 2, \dots, k\}$.

Teorema Floyd - Warshall

- **Teoremă:** Fie formulele de mai jos pentru calculul valorii $d^k(i,j)$, $0 < k \leq n$:
 - $d^0(i,j) = w(i,j)$;
 - $d^k(i,j) = \min\{d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j)\}$, pentru $0 < k \leq n$;
- Atunci $d^n(i,j) = \delta(i,j)$, pentru $\forall i, j \in V$
- **Dem:**
 - Prin inducție după k dem. că $d^k(i,j) = \delta^k(i,j)$. (next slide)
 - Pt. $k = n$, i..j trece prin $\forall v \in V$ și avem $d^n(i,j) \leq d^{n-1}(i,j)$,
 $\forall k = 1,n \rightarrow d^n(i,j) \leq d^{k-1}(i,j)$, $\forall k = 1,n$
 - Din $d^k(i,j) = \delta^k(i,j) \rightarrow d^n(i,j) = \delta^n(i,j) \leq d^{k-1}(i,j) = \delta^{k-1}(i,j)$, $\forall k = 1,n \rightarrow d^n(i,j) = \delta^n(i,j) = \delta(i,j)$

Demonstrație teorema Floyd - Warshall

- $K = 0$: 0 noduri intermediare $\rightarrow i..j = (i,j)$, la fel ca inițializarea $d^0(i,j) = w(i,j)$;
- $0 < k \leq n$: $d^{k-1}(i,j) = \delta^{k-1}(i,j) \rightarrow d^k(i,j) = \delta^k(i,j)$
- a) $k \notin$ drumului optim $i..j$: drumul optim nu se modifică
 $(\delta^{k-1}(i,j) = \delta^k(i,j) \leq \delta^{k-1}(i,k) + \delta^{k-1}(k,j))$
- $d^k(i,j) = \min\{d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j)\}$
 $d^k(i,j) = \min\{\delta^{k-1}(i,j), \delta^{k-1}(i,k) + \delta^{k-1}(k,j)\} = \delta^{k-1}(i,j) = \delta^k(i,j)$
- b) $k \in$ drumului optim $i..j$: $i..j$ se descompune în $i..k$ și $k..j$ optime
 $(\delta^{k-1}(i,k) = d^{k-1}(i,k)$ și $\delta^{k-1}(k,j) = d^{k-1}(k,j)$) și
 $\delta^k(i,j) = \delta^{k-1}(i,k) + \delta^{k-1}(k,j).$
- $i..j$ optim $\rightarrow \delta^k(i,j) \leq \delta^{k-1}(i,j)$
- $d^k(i,j) = \min\{d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j)\}$
- $d^k(i,j) = \min\{\delta^{k-1}(i,j), \delta^{k-1}(i,k) + \delta^{k-1}(k,j)\} = \delta^{k-1}(i,k) + \delta^{k-1}(k,j) = \delta^k(i,j)$

Algoritm Floyd-Warshall

I Floyd-Warshall(G)

- Pentru i de la 1 la n

- Pentru j de la 1 la n // inițializări

- $d^0(i,j) = w(i,j)$
 - Dacă ($w(i,j) == \infty$)
 - $p^0(i,j) = \text{null};$
 - Altfel $p^0(i,j) = i;$

- Pentru k de la 1 la n

- Pentru i de la 1 la n

- Pentru j de la 1 la n

- Dacă ($d^{k-1}(i,j) > d^{k-1}(i,k) + d^{k-1}(k,j)$) // determinăm minimul

- $d^k(i,j) = d^{k-1}(i,k) + d^{k-1}(k,j)$

- $p^k(i,j) = p^{k-1}(k,j); // și actualizăm părintele$

- Altfel

- $d^k(i,j) = d^{k-1}(i,j)$

- $p^k(i,j) = p^{k-1}(i,j);$

Complexitate?

$O(V^3)$

Complexitate
spațială?

$O(V^3)$

Observație

- Putem folosi o singură matrice în loc de n ?
- **Problemă:** în pasul k , pentru $k < i$ și $k < j$, $d(i,k)$ și $d(k,j)$ folosite la calculul $d(i,j)$ sunt $d^k(k,j)$ și $d^k(i,k)$ în loc de $d^{k-1}(k,j)$ și $d^{k-1}(i,k)$. Dacă dem. că $d^k(k,j) = d^{k-1}(k,j)$ și $d^k(i,k) = d^{k-1}(i,k)$, atunci putem folosi o singură matrice.
- Dar:
 - $d^k(k,j) = d^{k-1}(k,k) + d^{k-1}(k,j) = d^{k-1}(k,j)$
 - $d^k(i,k) = d^{k-1}(i,k) + d^{k-1}(k,k) = d^{k-1}(i,k)$
- → Algoritm modificat pentru a folosi o singura matrice → **complexitate spațială: $O(n^2)$** .

Algoritm Floyd-Warshall

Floyd-Warshall2(G)

- Pentru i de la 1 la n

- Pentru j de la 1 la n // inițializări

- $d(i,j) = w(i,j)$
 - Dacă ($w(i,j) == \infty$)
 - $p(i,j) = \text{null};$
 - Altfel $p(i,j) = i;$

- Pentru k de la 1 la n

- Pentru i de la 1 la n

- Pentru j de la 1 la n

- Dacă ($d(i,j) > d(i,k) + d(k,j)$) // determinăm minimul
 - $d(i,j) = d(i,k) + d(k,j)$
 - $p(i,j) = p(k,j); // și actualizăm părintele$

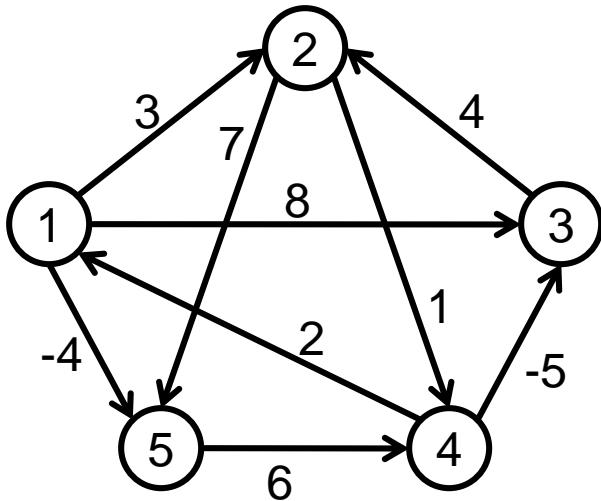
Complexitate?

$O(V^3)$

Complexitate
spațială?

$O(V^2)$

Exemplu (I)

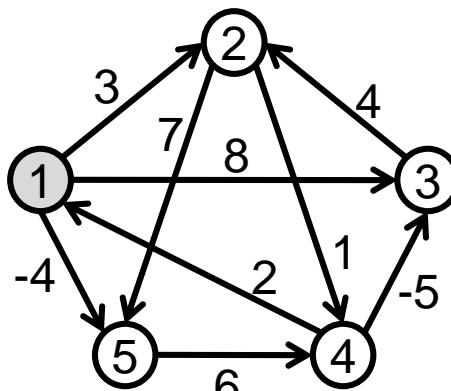


$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & \text{nil} & 4 & \text{nil} & \text{nil} \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

Exemplu (II)

$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$



$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \textcolor{red}{5} & -5 & 0 & \textcolor{red}{-2} \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & \text{nil} & 4 & \text{nil} & \text{nil} \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

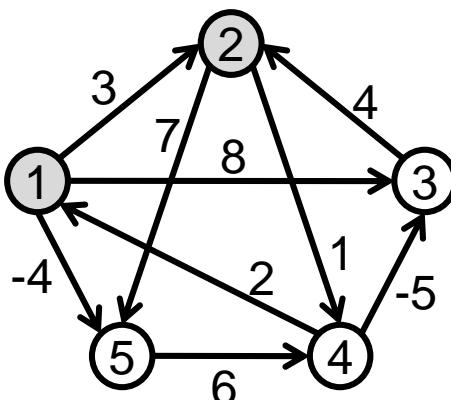
$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & \textcolor{red}{1} & 4 & \text{nil} & \textcolor{red}{1} \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

Exemplu (III)

$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & 1 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

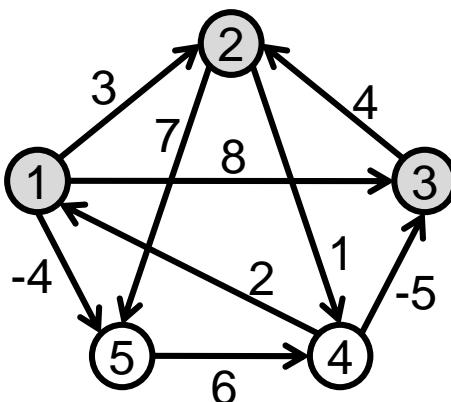
$$D = \begin{bmatrix} 0 & 3 & 8 & \textcolor{red}{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \textcolor{red}{5} & \textcolor{red}{11} \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$



$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \textcolor{red}{2} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \textcolor{red}{2} & \textcolor{red}{2} \\ 4 & 1 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

Exemplu (IV)

$$D = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$



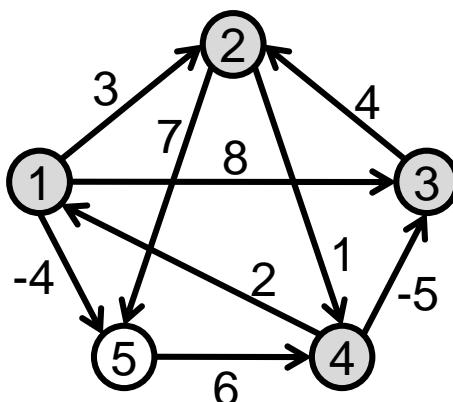
$$D = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & \textcolor{red}{-1} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & 2 & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & 2 & 2 \\ 4 & 1 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & 2 & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & 2 & 2 \\ 4 & \textcolor{red}{3} & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

Exemplu (V)

$$D = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$



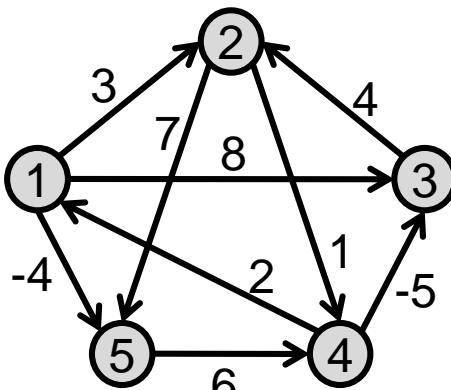
$$D = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & 2 & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & 2 & 2 \\ 4 & 3 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 4 & 2 & 1 \\ 4 & \text{nil} & 4 & 2 & 1 \\ 4 & 3 & \text{nil} & 2 & 1 \\ 4 & 3 & 4 & \text{nil} & 1 \\ 4 & 3 & 4 & 5 & \text{nil} \end{bmatrix}$$

Exemplu (VI)

$$D = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$



$$D = \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 4 & 2 & 1 \\ 4 & \text{nil} & 4 & 2 & 1 \\ 4 & 3 & \text{nil} & 2 & 1 \\ 4 & 3 & 4 & \text{nil} & 1 \\ 4 & 3 & 4 & 5 & \text{nil} \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 3 & 4 & 5 & 1 \\ 4 & \text{nil} & 4 & 2 & 1 \\ 4 & 3 & \text{nil} & 2 & 1 \\ 4 & 3 & 4 & \text{nil} & 1 \\ 4 & 3 & 4 & 5 & \text{nil} \end{bmatrix}$$

Închiderea tranzitivă (I)

- Fie $G = (V, E)$. Închiderea tranzitivă a lui E e un $G^* = (V, E^*)$, unde

$$E^*(i,j) = \begin{cases} 1, & \text{dacă } \exists i..j \\ 0, & \text{dacă } \nexists i..j \end{cases}$$

- Poate fi determinată prin modificarea algoritmului Floyd-Warshall:
 - $\min \Rightarrow$ operatorul boolean **sau** (\vee)
 - $+$ \Rightarrow operatorul boolean **și** (\wedge)

Închiderea tranzitivă (II)

Închidere_tranzitivă(G)

- Pentru i de la 1 la n
 - Pentru j de la 1 la n
 - $E^* (i,j) = (((i,j) \in E) \vee (i = j))$ // inițializări
- Pentru k de la 1 la n
 - Pentru i de la 1 la n
 - Pentru j de la 1 la n
 - $E^* (i,j) = E^* (i,j) \vee (E^* (i,k) \wedge E^* (k,j))$

Complexitate?

$O(V^3)$

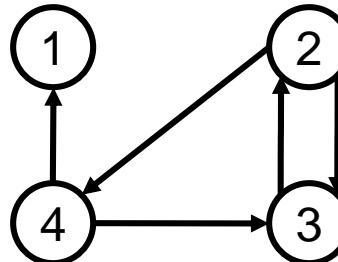
Complexitate spațială?

$O(V^2)$

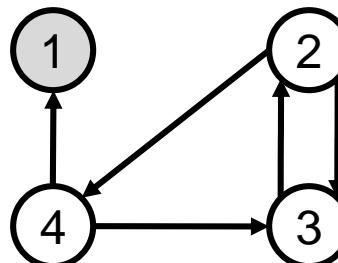
Exemplu (I)

| Închidere_tranzitivă(G)

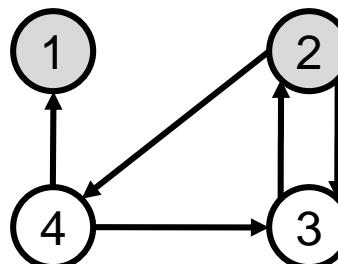
- **Pentru i de la 1 la n**
 - **Pentru j de la 1 la n**
 - $E^*(i,j) = (i,j) \in E \vee i = j$
// initializări
 - **Pentru k de la 1 la n**
 - **Pentru i de la 1 la n**
 - **Pentru j de la 1 la n**
 - $E^*(i,j) = E^*(i,j) \vee (E^*(i,k) \wedge E^*(k,j))$



$$T^{(0)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$



$$T^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

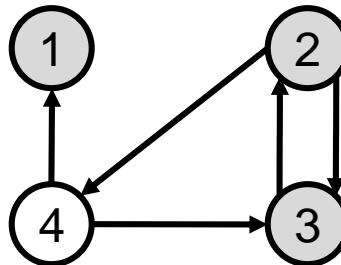


$$T^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

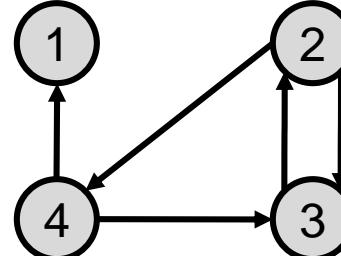
Exemplu (II)

Închidere_tranzitivă(G)

- Pentru i de la 1 la n
 - Pentru j de la 1 la n
 - $E^*(i,j) = (i,j) \in E \vee i = j$
// initializări
- Pentru k de la 1 la n
 - Pentru i de la 1 la n
 - Pentru j de la 1 la n
 - $E^*(i,j) = E^*(i,j) \vee (E^*(i,k) \wedge E^*(k,j))$



$$T^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$



$$T^{(4)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Algoritmul lui Johnson

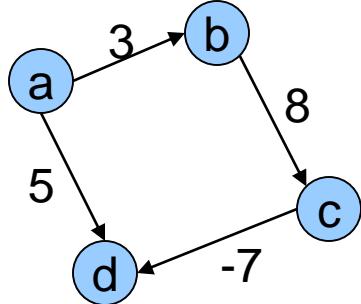
- Pentru grafuri rare.
- Folosește liste de adiacență.
- Bazat pe Dijkstra și Bellman-Ford.
- Complexitate: $O(V^2 \log V + VE)$
 - Mai bună decât Floyd-Warshall pentru grafuri rare.

Idee algoritm Johnson

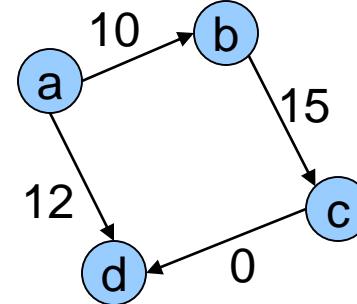
- Dacă graful are **numai arce pozitive**:
 - se aplică **Dijkstra** pentru fiecare nod => cost $V^2 \log V$.
- Altfel **se calculează costuri pozitive pentru fiecare arc menținând proprietățile**:
 - $w_1(u,v) \geq 0, \forall (u,v) \in E$;
 - p este drum minim utilizând $w \Leftrightarrow p$ este drum minim utilizând w_1 .

Construcție w_1 (I)

- Idee 1: identificare arcul cu cel mai mic cost – c; adunare la costul fiecărui arc valoarea c;



$$\text{cost}(a..b..d) < \text{cost}(a,d)$$



$$\text{cost}(a..b..d) > \text{cost}(a,d)$$

Nu funcționează!!!

Construcție w_1 (II)

- Idee 2: $w_1(u,v) = w(u,v) + h(u) - h(v)$;
- unde $h : V \rightarrow \mathbb{R}$;
- se adaugă un nod s ;
- se unește s cu toate nodurile grafului prin arce de cost 0;
- se aplică BF pe acest graf $\Rightarrow h(v) = \delta(s,v)$;
- $\rightarrow w_1(u,v) = w(u,v) + h(u) - h(v)$.

Algoritm Johnson

- Johnson(G)
 - // Construim $G' = (V', E')$;
 - $V' = V \cup \{s\}$; // adăugăm nodul s
 - $E' = E \cup (s, u)$, $\forall u \in V$; $w(s, u) = 0$; // și îl legăm de toate nodurile
 - Dacă $BF(G', s)$ e fals // aplic BF pe G'
 - Eroare “ciclu negativ”
 - Altfel
 - Pentru fiecare $v \in V$
 - $h(v) = \delta(s, v)$; // calculat prin BF
 - Pentru fiecare $(u, v) \in E$
 - $w_1(u, v) = w(u, v) + h(u) - h(v)$ // calculez noile costuri pozitive
 - Pentru fiecare $(u \in V)$
 - Dijkstra(G, w_1, u) // aplic Dijkstra pentru fiecare nod
 - Pentru fiecare $(v \in V)$
 - $d(u, v) = \delta_1(u, v) + h(v) - h(u)$ // calculez costurile pe graful inițial

Exemplu (I)

BellmanFord(G, s) // $G = (V, E)$, s =sursă

Pentru fiecare v din V // inițializări

$d[v] = \infty$; $p[v] = \text{null}$;

$d[s] = 0$; // actualizare distanță de la s la s

Pentru i de la 1 la $|V| - 1$ // se construiesc drumuri optime de dimensiune i

Pentru fiecare (u, v) din E // pentru arcele ce pleacă de la nodurile deja considerate

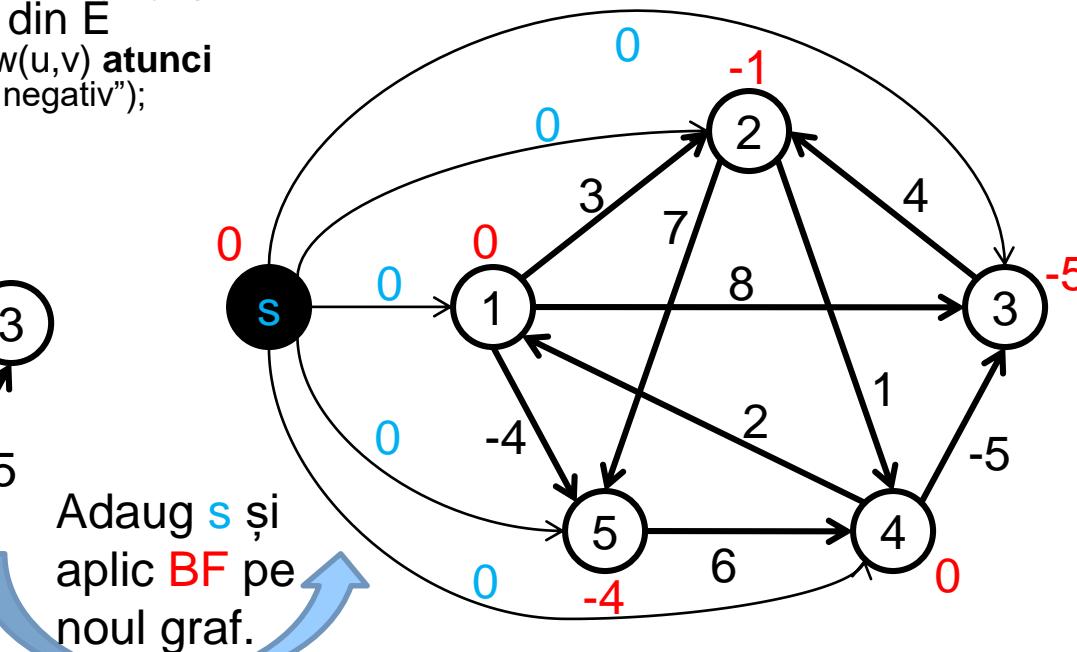
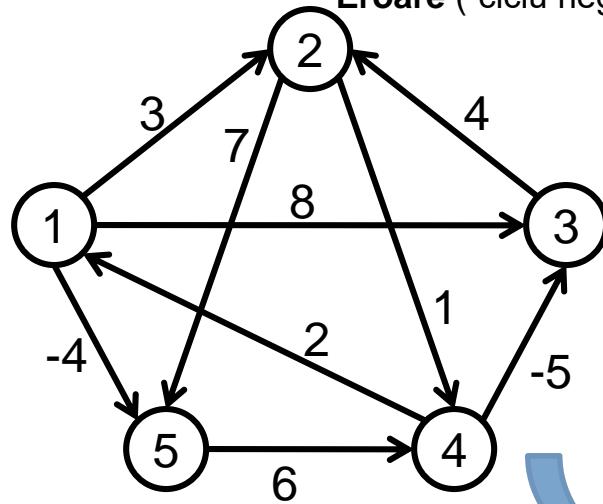
Dacă $d[v] > d[u] + w(u, v)$ atunci // se relaxează arcele corespunzătoare

$d[v] = d[u] + w(u, v)$; $p[v] = u$;

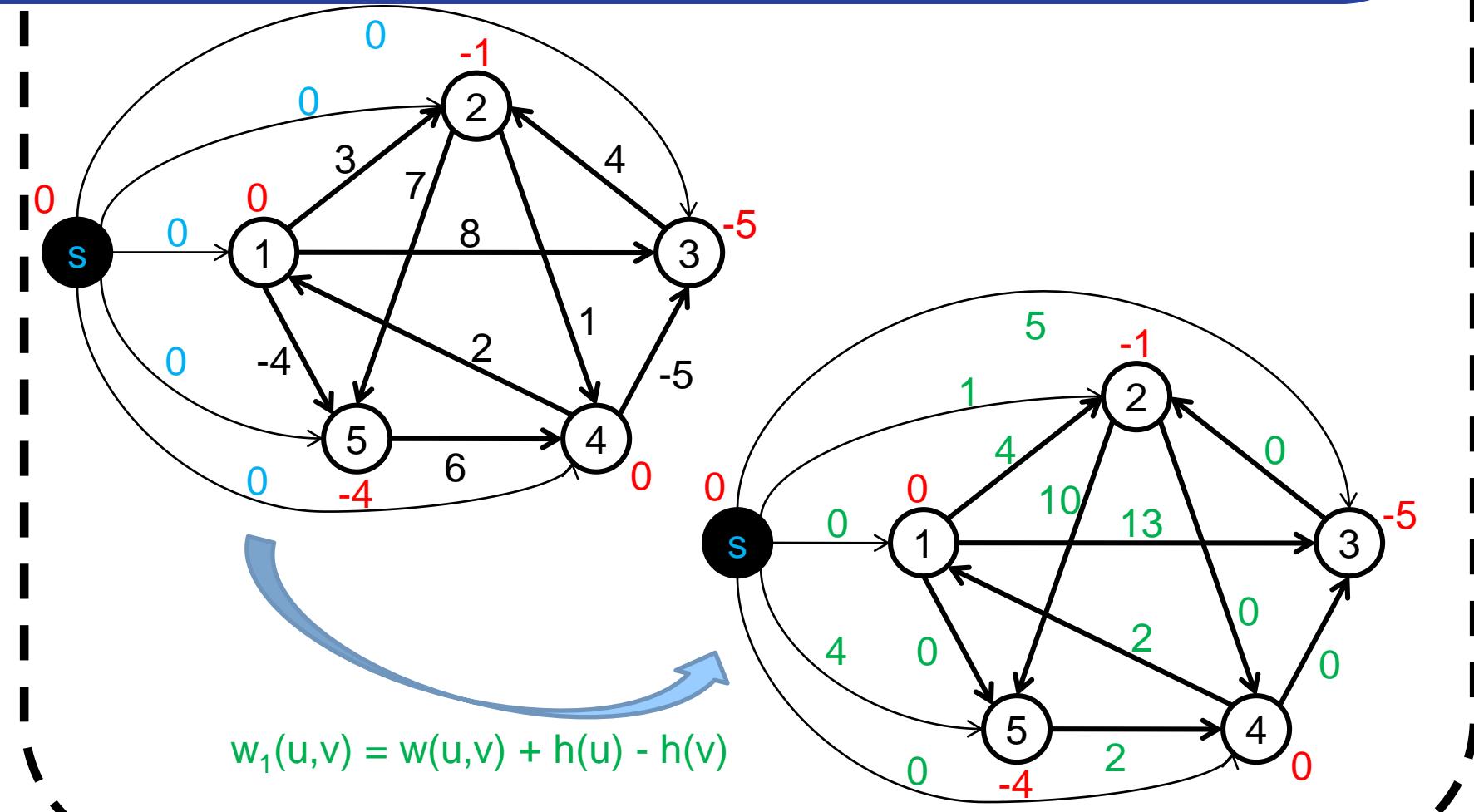
Pentru fiecare (u, v) din E

Dacă $d[v] > d[u] + w(u, v)$ atunci

Eroare ("ciclu negativ");

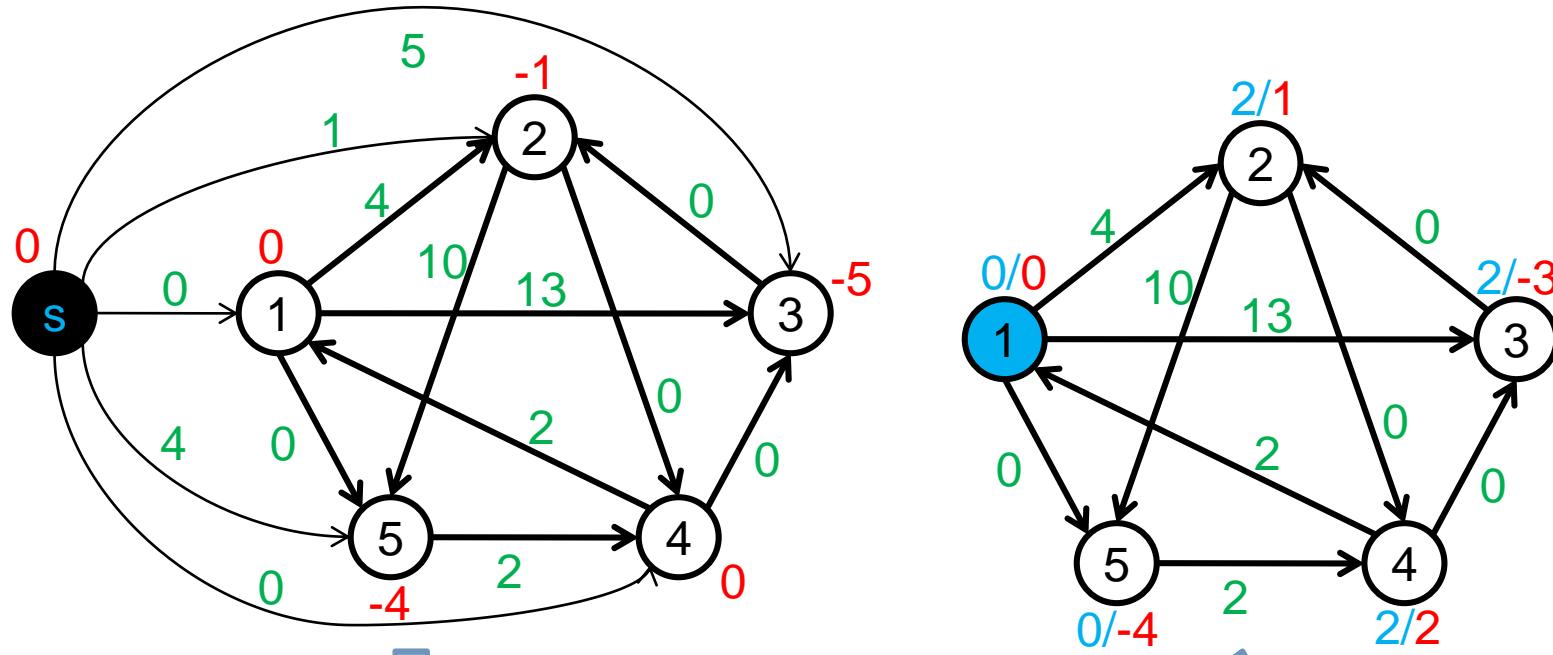


Exemplu (II)



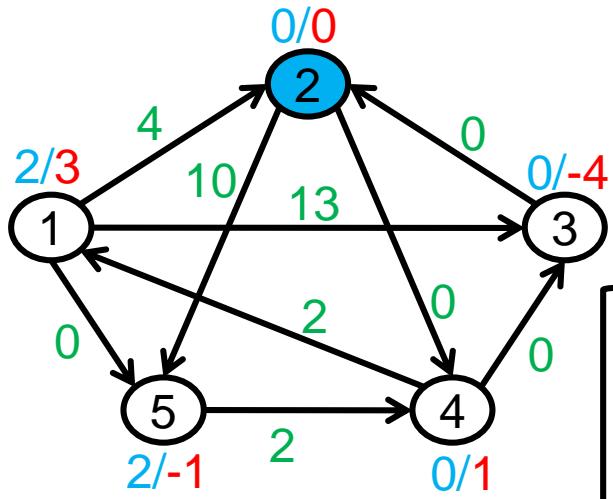
$$w_1(u,v) = w(u,v) + h(u) - h(v)$$

Exemplu (III)

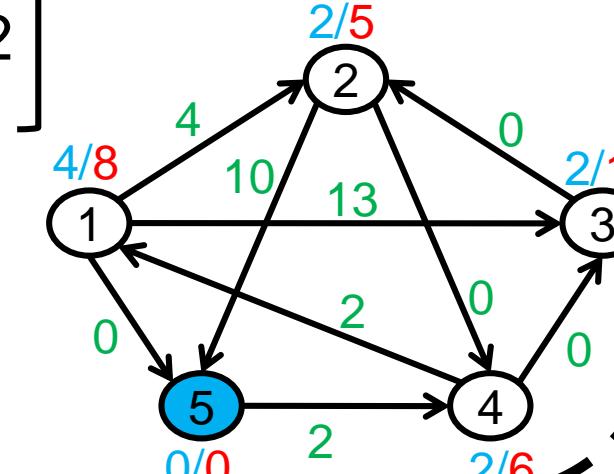
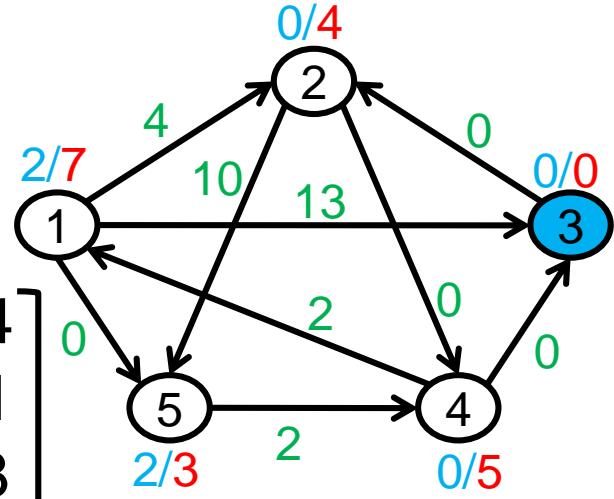
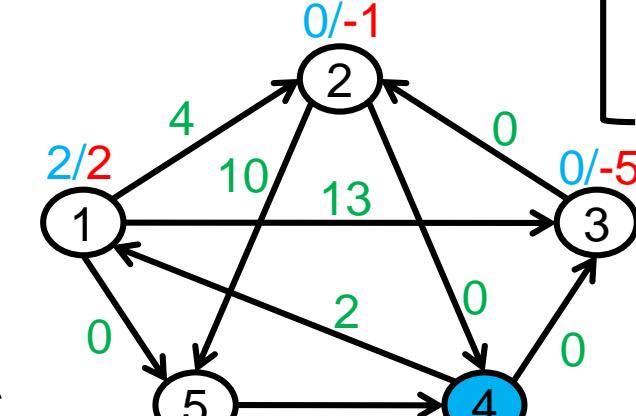


Aplicăm Dijkstra din fiecare nod ($\delta_1(u,v)$).
Refacem distanțele:
 $d(u,v) = \delta_1(u,v) + h(v) - h(u)$

Exemplu (IV)



$$\begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & 0 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$



Concluzii Floyd-Warshall & Johnson

- Algoritmi ce găsesc **drumurile minime** între oricare **2 noduri** din graf.
- Funcționează pe grafuri cu arce **ce au costuri negative** (dar care **nu au cicluri de cost negativ**).
- Floyd-Warshall e **optim** pentru grafuri dese.
- Johnson e **mai bun** pentru grafuri rare.

ÎNTREBĂRI?

Proiectarea Algoritmilor

Curs 10 – Arbori minimi de acoperire

Bibliografie

- | [1] http://monalisa.cacr.caltech.edu/monalisa__Service_Applications__Monitoring_VRVS.html
- | [2] <http://www.cobblestoneconcepts.com/ucgis2summer2002/guo/guo.html>
- | [3] Giumale – Introducere in Analiza Algoritmilor cap. 5.5
- | [4] R. Sedgewick, K Wayne – curs de algoritmi Princeton 2007
www.cs.princeton.edu/~rs/AlgsDS07/ 01UnionFind si 14MST
- | [5] http://www.pui.ch/phred/automated_tag_clustering/
- | [6] Cormen – Introducere în Algoritmi cap. Arboi de acoperire minimi
(24)

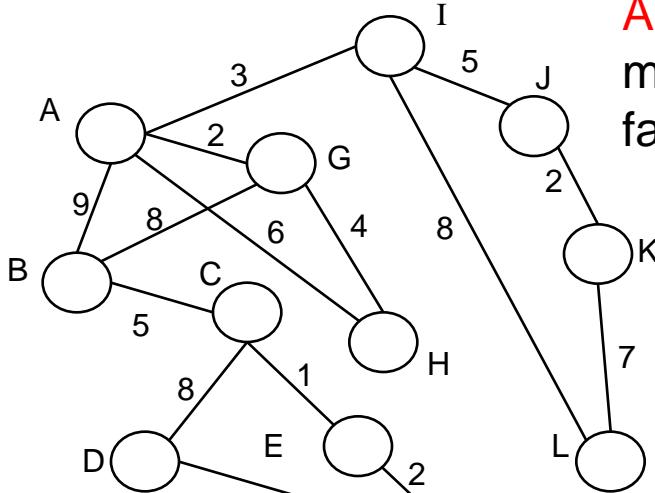
Planul cursului

- **Arbore minimi de acoperire:**
 - Definiție;
 - Utilizare;
 - Algoritmi.
- **Operații cu multimi disjuncte:**
 - Structuri de date pentru reprezentarea multimilor disjuncte;
 - Algoritmi pentru reuniune și căutare;
 - Calcul de complexitate.

Arborei minimi de acoperire – Definiții

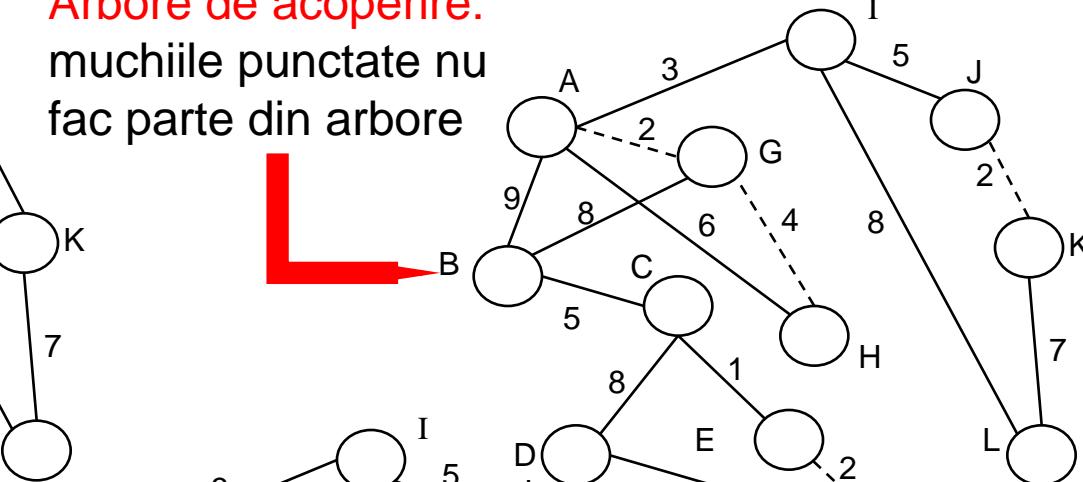
- Fie $G = (V, E)$ graf neorientat și conex, iar $w: E \rightarrow \mathbb{R}$ o funcție de cost ($w(u,v) =$ costul muchiei (u,v)).
- Definiție: Un arbore liber al lui G este un graf neorientat conex și aciclic $\text{Arb} = (V', E')$; $V' \subseteq V$, $E' \subseteq E$. Costul arborelui este: $C(\text{Arb}) = \sum w(e)$, $e \in E'$.
- Definiție: Un arbore liber se numește arbore de acoperire dacă $V' = V$.
- Definiție: Un arbore de acoperire (Arb) se numește arbore minim de acoperire (notăm AMA) dacă $\text{Arb} \in \text{ARB}(G)$ a.î. $C(\text{Arb}) = \min\{C(\text{Arb}') \mid \text{Arb}' \in \text{ARB}(G)\}$.

Exemplu

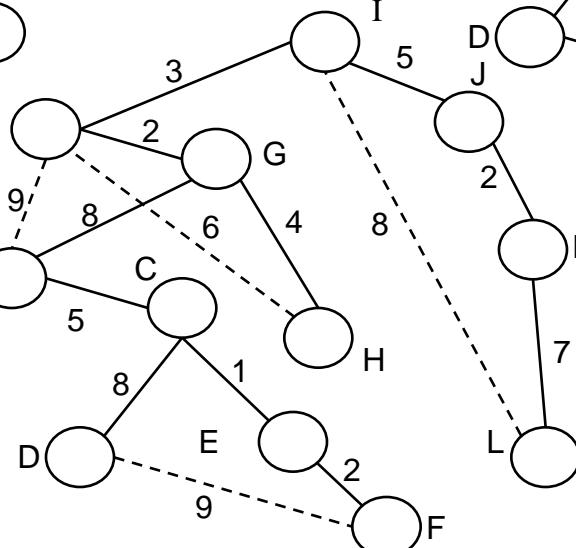


Graf neorientat
conex

Arbore de acoperire:
muchiile punctate nu
fac parte din arbore



Arbore minim de acoperire:
muchiile punctate au fost
eliminate din graf



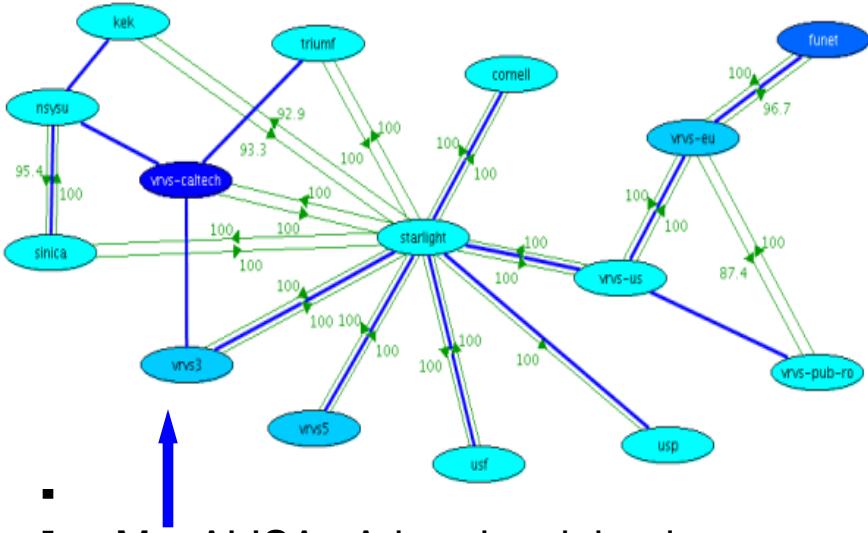
PA



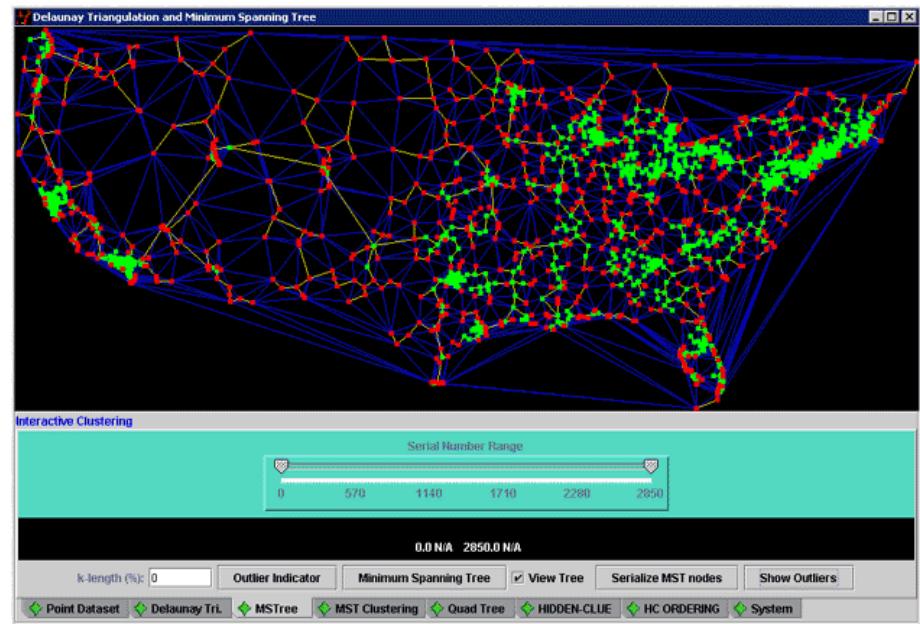
Utilizări

- Proiectarea rețelelor:
 - Electrice, calculatoare, drumuri.
- Clustering.
- Algoritmi de aproximare pentru probleme NP-complete.

Exemple de utilizare



MonALISA - Arborele minim de acoperire al conexiunilor si calitatea conexiunilor peer-to-peer pentru un set de relee VRVS (caltech) [1]



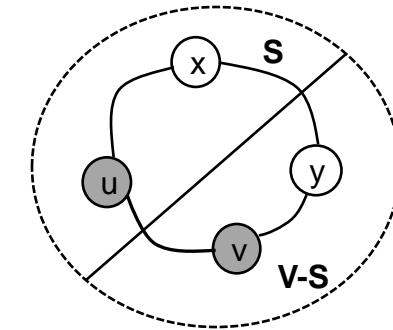
Arbore minim de acoperire pentru cca 2850 de orașe din USA [2]

AMA – Definiții (II)

- **Definiție:** Fie $A \subseteq E$ o mulțime de muchii ale unui graf $G = (V, E)$ și $(S, V-S)$ o **partiționare a lui V** . Partiționarea respectă mulțimea A dacă $\nexists e \in A$ care taie frontiera dintre S și $V-S$ ($\forall (u, v) \in A \rightarrow u, v \in S$ sau $u, v \in V-S$).
- **Definiție:** Fie $A \subseteq E'$ o mulțime de muchii ale unui AMA parțial $\text{Arb} = (V, E')$ al grafului $G = (V, E)$, iar $e \in E$ o muchie oarecare din G . Muchia e este **sigură în raport cu A** dacă **mulțimea $A \cup \{e\}$ face parte dintr-un AMA al lui G** .

AMA – Teoremă

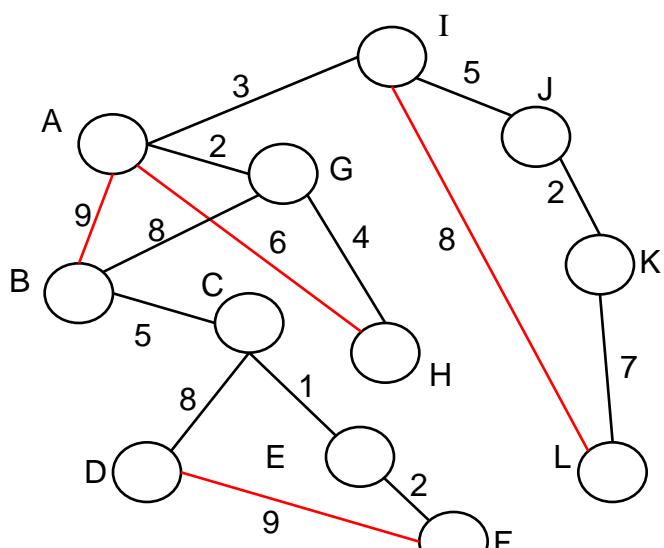
- **Teorema 5.23:** Fie A o mulțime de muchii ale unui AMA parțial al grafului $G = (V, E)$. Fie $(S, V-S)$ o partiționare care respectă A, iar $(u,v) \in E$ o muchie care taie frontiera dintre S și $V-S$ a.î.
 $w(u,v) = \min\{w(x,y) \mid (x,y) \in E \text{ și } (x \in S, y \in V-S) \text{ sau } (x \in V-S, y \in S)\}$
Muchia (u,v) este **sigură in raport cu A**.



- **Dem (Reducere la absurd):**
 - pp (u,v) nu e muchie sigură.
 - (I) $\rightarrow \exists$ AMA $\text{Arb}' = (V, E')$, a.î. $A \subseteq E'$. Pp $(u,v) \notin \text{Arb}'$
 - În $\text{Arb}' \exists$ cale $u..v \rightarrow \exists (x,y) \in u..v$ care taie partiționarea și $(x,y) \in \text{Arb}'$
 - $(x,y) \notin A$, $(u,v) \notin A$ pt. că partiționarea respectă A, iar $w(u,v) \leq w(x,y)$ (I)
 - Dacă în Arb' eliminăm (x,y) și adăugăm $(u,v) \rightarrow \text{Arb}'' = (V, E'')$, $E'' = E' - \{(x,y)\} + \{(u,v)\}$
 - $C(\text{Arb}'') \leq C(\text{Arb}')$, $\text{Arb}' - \text{AMA} \rightarrow C(\text{Arb}') = C(\text{Arb}'') \rightarrow \text{Arb}'' - \text{AMA} \rightarrow (u,v)$ – muchie sigură.

Proprietăți (I)

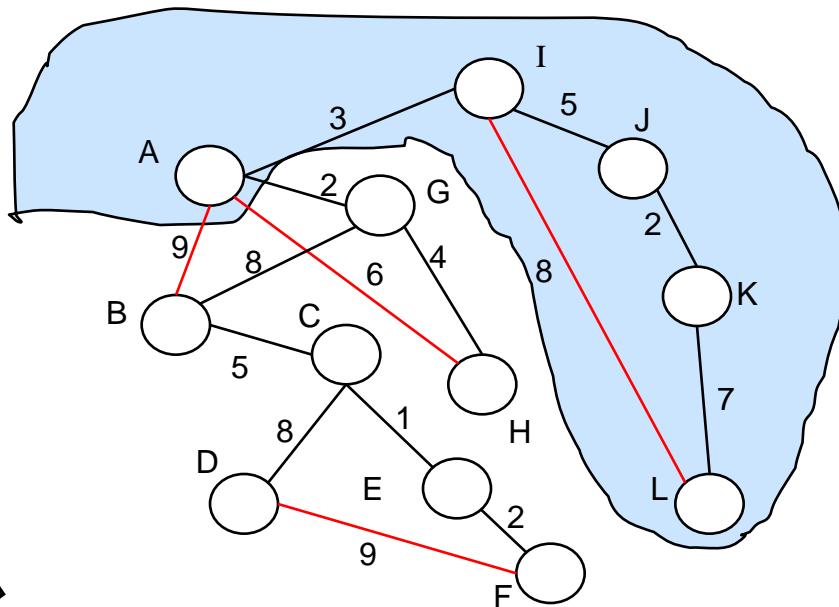
- | $G = (V, E)$, $C = (V', E')$ – ciclu în G ; $e \in E'$
- | a.î. $w(e) = \max \{w(e') \mid e' \in E'\} \Rightarrow e \notin$
- | $\text{Arb}(G)$ unde $\text{Arb}(G) = \text{AMA}$ în G .



- Dem (Reducere la absurd): Pp $e \in \text{Arb}(G)$.
- Eliminând e din $\text{Arb}(G) \rightarrow 2$ multimi de muchii: S_1, S_2 .
- $e \in E'$ (ciclu) $\rightarrow \exists e' \in E', w(e) > w(e')$ a.î. un capăt din e' este în S_1 și celalalt în S_2 .
- $\text{Arb}(G) - e + e' = \text{arbore de acoperire}$.
- $\text{Cost}(\text{Arb}(G) - e + e') < \text{Cost}(\text{Arb}(G)) \Rightarrow \text{Arb}(G)$ nu este arbore minim.

Proprietăți (II)

| G = (V,E), S = (V',E') un AMA parțial al lui G, V' ⊂ V; e =
| (u,v) a.î. e ∉ E' și (u ∈ V' și v ∉ V') sau (u ∉ V' și v ∈ V')
| cu proprietatea că: w(u,v) = min{w(u',v')| (u' ∈ V' și v' ∉ V')
| sau (u' ∉ V' și v' ∈ V')} } => (u,v) ∈ AMA.



• Dem (Reducere la absurd): Pp e ∉ AMA Arb(G).

• Arb' = Arb(G) – e' + e (unde e' o muchie similară cu e).

• Arb' = arbore de acoperire.

• Cost(Arb') < Cost(Arb) → Arb(G) nu este AMA.

AMA

- Bazați pe ideea de **muchie sigură** – se identifică o muchie sigură și se adaugă în AMA.
- 2 algoritmi de tip **greedy**:
 - **Prim**: se pornește cu un nod și se extinde pe rând cu muchiile cele mai ieftine care au un singur capăt în mulțimea de muchii deja formată (**Proprietatea 2**). Algoritmul este asemănător algoritmului Dijkstra.
 - **Kruskal**: inițial toate nodurile formează câte o mulțime și la fiecare pas se reunesc 2 mulțimi printr-o muchie. Muchiile sunt considerate în ordinea costurilor și sunt adăugate în arbore doar dacă nu creează ciclu (**Proprietatea 1**).

Algoritmul lui Prim

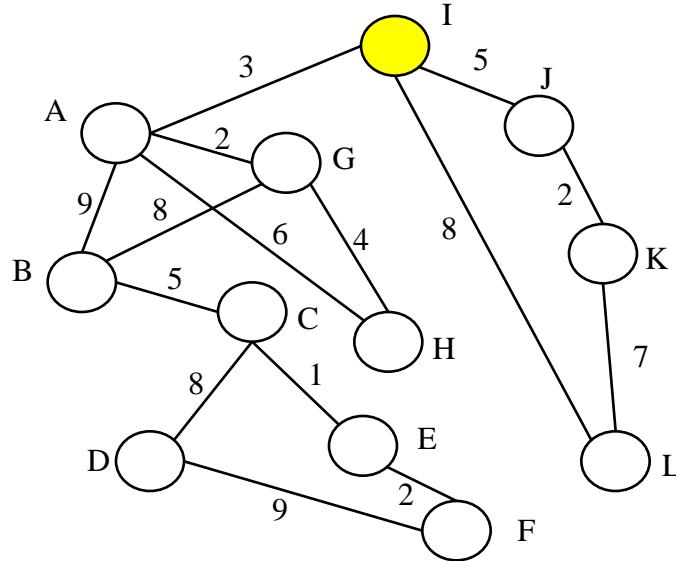
- $\text{Prim}(G, w, s)$

Implementare în Java la [4] !

- $A = \emptyset$ // initializare AMA
- Pentru fiecare $(u \in V)$
 - $d[u] = \infty$; $p[u] = \text{null}$ // initializăm distanța și părintele
- $d[s] = 0$; // nodul de start are distanța 0
- $Q = \text{constrQ}(V, d)$; // ordonată după costul muchiei
// care unește nodul de AMA deja creat
- Cât timp ($Q \neq \emptyset$) // cât timp mai sunt noduri neadăugate
 - $u = \text{ExtragMin}(Q)$; // extrag nodul aflat cel mai aproape
 - $A = A \cup \{(u, p[u])\}$; // adaug muchia în AMA
 - Pentru fiecare $(v \in \text{succs}(u))$
 - Dacă $d[v] > w(u, v)$ atunci
 - $d[v] = w(u, v)$; // actualizăm distanțele și părinții nodurilor
 - $p[v] = u$; // adiacente care nu sunt în AMA încă
- Întoarce $A - \{(s, p(s))\}$ // prima muchie adăugată

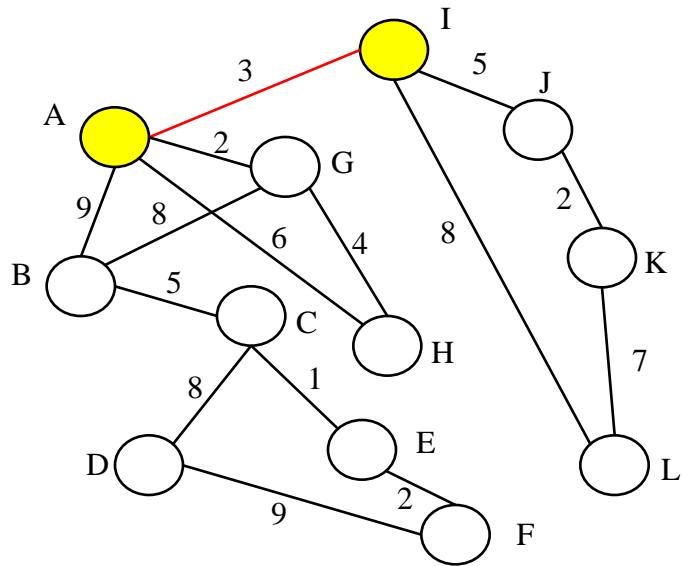
Exemplu (I)

- Pornim din I



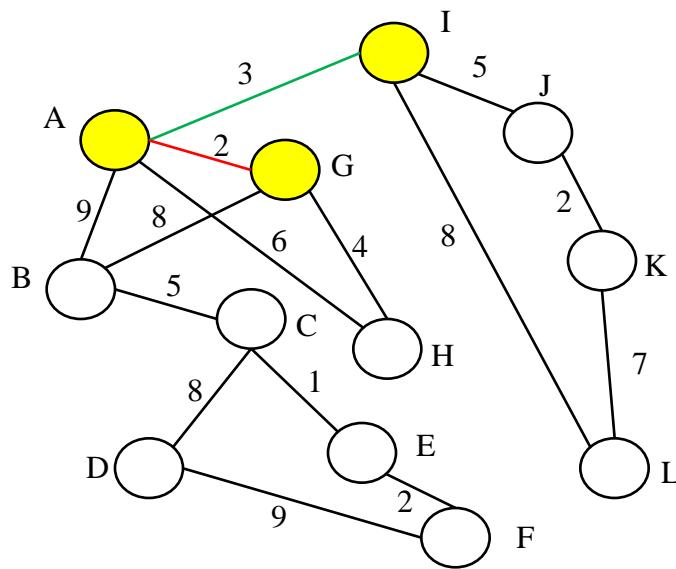
- Q: A(3), J(5), L(8),
B(∞), C(∞), D(∞), E(∞),
F(∞), G(∞), H(∞), K(∞)
 \rightarrow A

Exemplu (II)



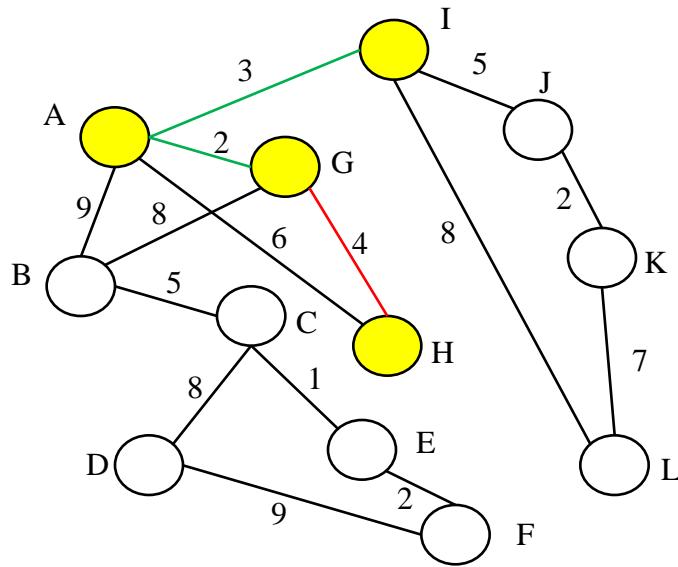
- Q: $G(2), J(5), H(6), L(8), B(9), C(\infty), D(\infty), E(\infty), F(\infty), K(\infty) \rightarrow G$

Exemplu (III)



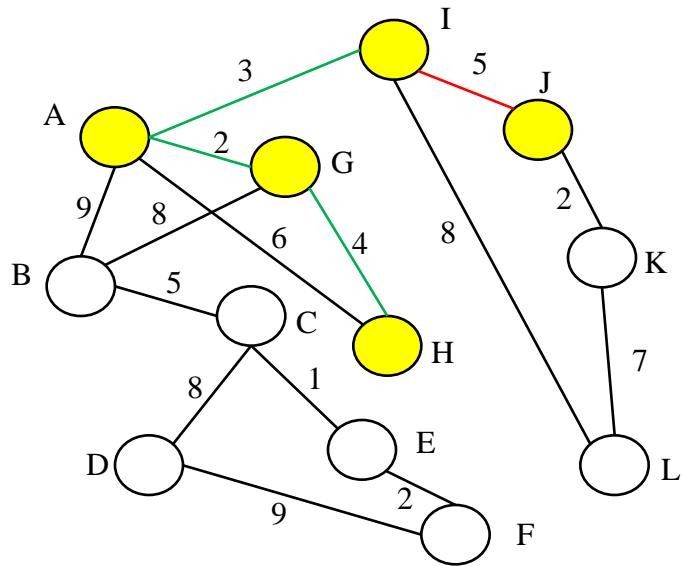
- Q: G(2), J(5), H(6), L(8), B(9), C(∞), D(∞), E(∞), F(∞), K(∞) \rightarrow G
-
- Q: H(4), J(5), L(8), B(8), C(∞), D(∞), E(∞), F(∞), K(∞) \rightarrow H

Exemplu (IV)



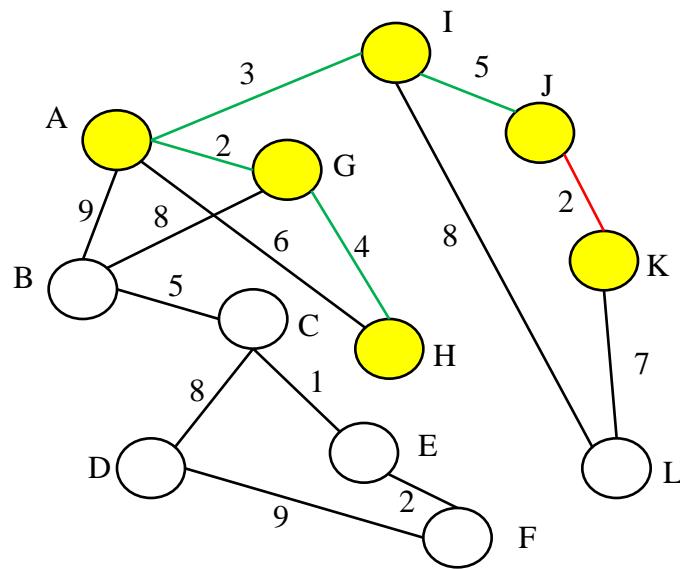
- Q: J(5), L(8), B(8), C(∞), D(∞), E(∞), F(∞), K(∞) \rightarrow J

Exemplu (V)



- Q: K(2), L(8), B(8),
C(∞), D(∞), E(∞), F(∞)
 \rightarrow K

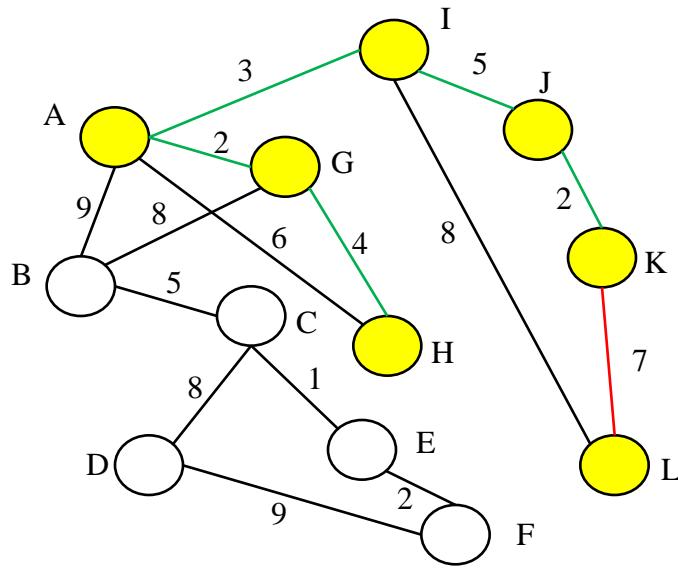
Exemplu (VI)



- Q: K(2), L(8), B(8), C(∞), D(∞), E(∞), F(∞)
→ K
- Q: L(7), B(8), C(∞), D(∞), E(∞), F(∞) → L

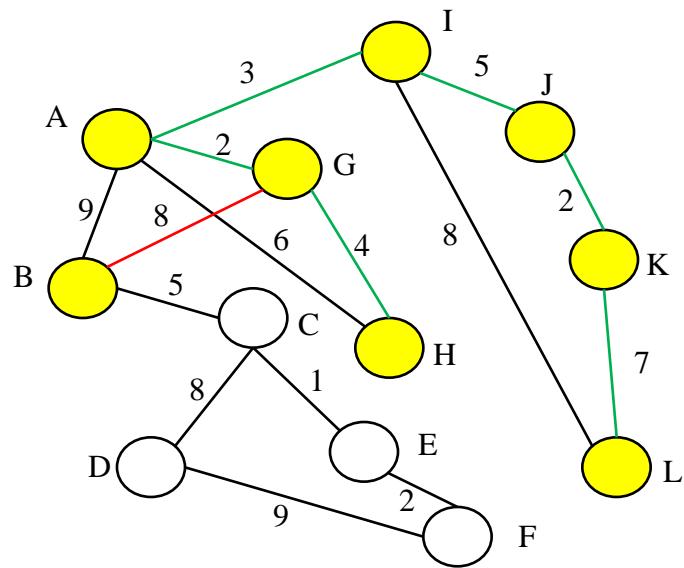


Exemplu (VII)



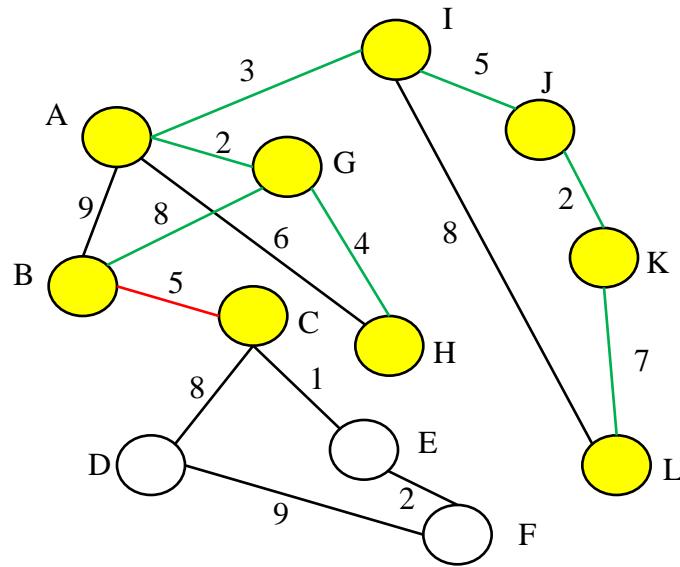
- $Q: B(8), C(\infty), D(\infty), E(\infty), F(\infty) \rightarrow B$

Exemplu (VIII)



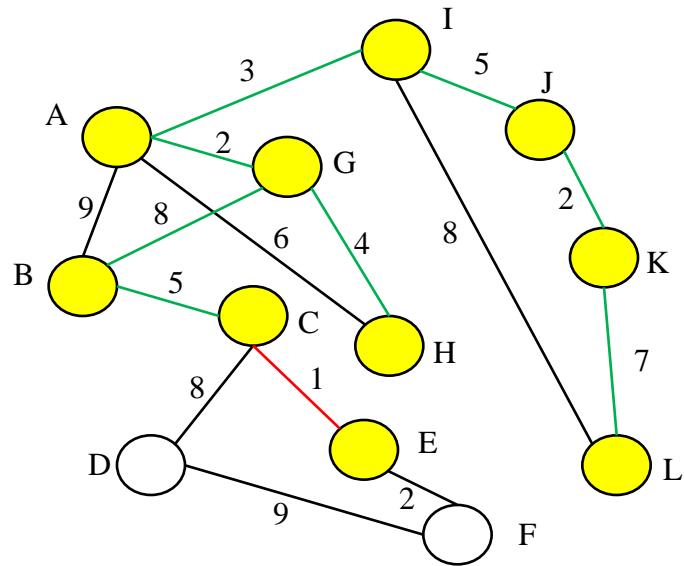
- $Q: C(5), D(\infty), E(\infty), F(\infty) \rightarrow C$

Exemplu (IX)



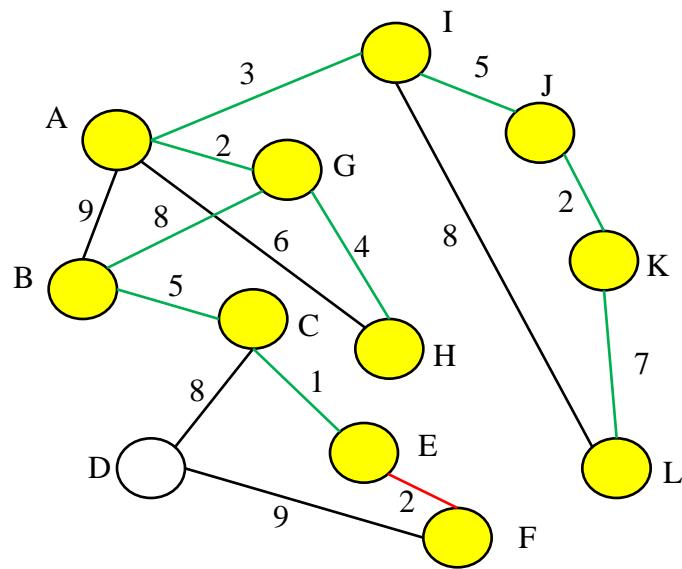
- $Q: E(1), D(8), F(\infty) \rightarrow E$

Exemplu (X)



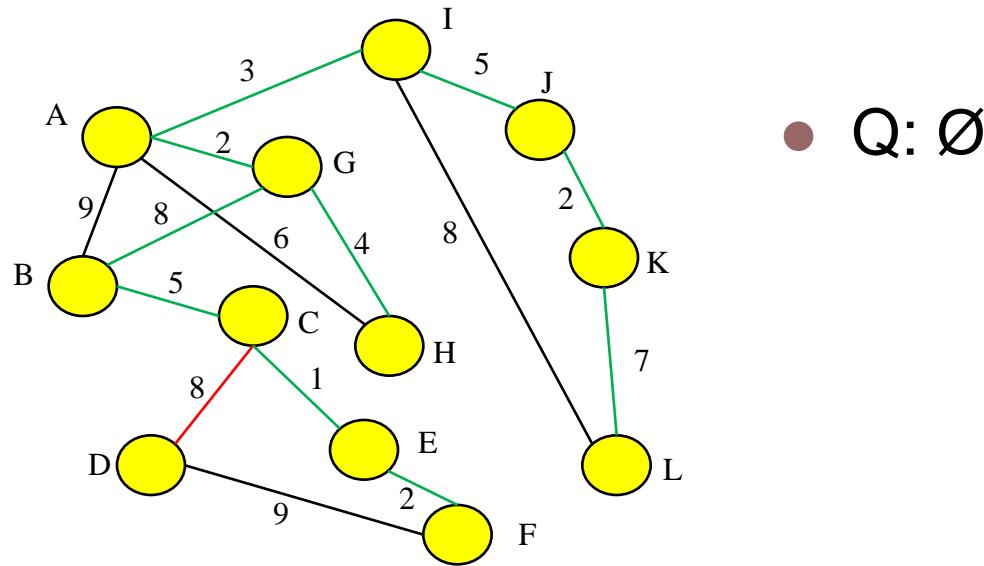
- Q: $F(2), D(8) \rightarrow F$

Exemplu (XI)



- $Q: D(8) \rightarrow D$

Exemplu (XII)



• Q: \emptyset

Corectitudine (I)

- 1. Arătăm că muchiile pe care le adăugăm aparțin Arb:
- Dem prin inducție după muchiile adăugate în AMA:
 - P_1 : avem $V' = s$, $E' = \emptyset$. Adaug muchia (u,s) , $u = \text{nod adjacent sursei aflat cel mai aproape de aceasta} \rightarrow \text{din Propr. 2} \rightarrow (u,s) \in \text{Arb.}$
 - $P_n \rightarrow P_{n+1}$:
 - $S = (V', E')$ multimea vârfurilor și muchiilor adăugate deja în arbore înainte de a adăuga $(u,p[u])$.
 - $p[u] \in V'$, $u \notin V'$; $(u,p[u])$ are cost minim dintre muchiile care au un capăt în S (conform extrage minim)
 - din Propr. 2 $\rightarrow (u,p[u]) \in \text{Arb}$

Corectitudine (II)

- 2. arătăm că muchiile ignoreate nu fac parte din Arb:
 - $d[v]$ scade tot timpul de-a lungul algoritmului până când v este adăugat în AMA. În momentul adăugării, s-a găsit muchia de cost minim ce conectează nodul v la AMA;
 - Pp. (u,v) a.î. $\text{Arb}(u) = \text{Arb}(v)$
 - $\rightarrow (u,v)$ creează un ciclu în $\text{Arb}(u)$ (arborii sunt aciclici) – fie ciclul format din $u..x..v$ și (u,v) .
 - $w(u,v) = \max \{w(u',v') \mid (u',v') \in \text{Arb}(u)\}$ DE CE?
 - Nodul u i-a fost adiacent nodului v , dar nu a fost ales la niciunul din momentele ulterioare de timp, când au fost parcurse muchiile din $u..x..v \rightarrow (u,v)$ are costul maxim din ciclu
 - \rightarrow din Prop. 1 $\rightarrow (u,v) \notin \text{Arb}$

Algoritmul lui Prim

- $\text{Prim}(G, w, s)$
 - $A = \emptyset // \text{AMA}$
 - **Pentru fiecare** ($u \in V$)
 - $d[u] = \infty; p[u] = \text{null} // \text{initializăm distanța și părintele}$
 - $d[s] = 0; // \text{nodul de start are distanța 0}$
 - $Q = \text{constrQ}(V, d); // \text{ordonată după costul muchiei}$
 $// \text{care unește nodul de AMA deja creat}$
 - **Cât timp** ($Q \neq \emptyset$) $// \text{cât timp mai sunt noduri neadăugate}$
 - $u = \text{ExtragMin}(Q); // \text{extrag nodul aflat cel mai aproape}$
 - $A = A \cup \{(u, p[u])\}; // \text{adaug muchia în AMA}$
 - **Pentru fiecare** ($v \in \text{succs}(u)$)
 - **Dacă** $d[v] > w(u, v)$ **atunci**
 - $d[v] = w(u, v); // + d[u] // \text{actualizăm distanțele și părintii nodurilor}$
 - $p[v] = u; // \text{adiacente care nu sunt în AMA încă}$
 - **Întoarce** $A - \{(s, p(s))\} // \text{prima muchie adăugată}$

Reminder Dijkstra (II)

- **Dijkstra(G, s)**
 - **Pentru fiecare** ($u \in V$)
 - $d[u] = \infty$; $p[u] = \text{null}$;
 - $d[s] = 0$;
 - $Q = \text{construieste_coada}(V)$ // coadă cu priorități
 - **Cât timp** ($Q \neq \emptyset$)
 - $u = \text{ExtrageMin}(Q)$; // extrage din V elementul cu $d[u]$ minim
 - // $Q = Q - \{u\}$ – se execută în cadrul lui `ExtrageMin`
 - **Pentru fiecare** ($v \in Q$ și v din succesorii lui u)
 - **Dacă** ($d[v] > d[u] + w(u,v)$)
 - $d[v] = d[u] + w(u,v)$ // actualizez distanță
 - $p[v] = u$ // și părintele

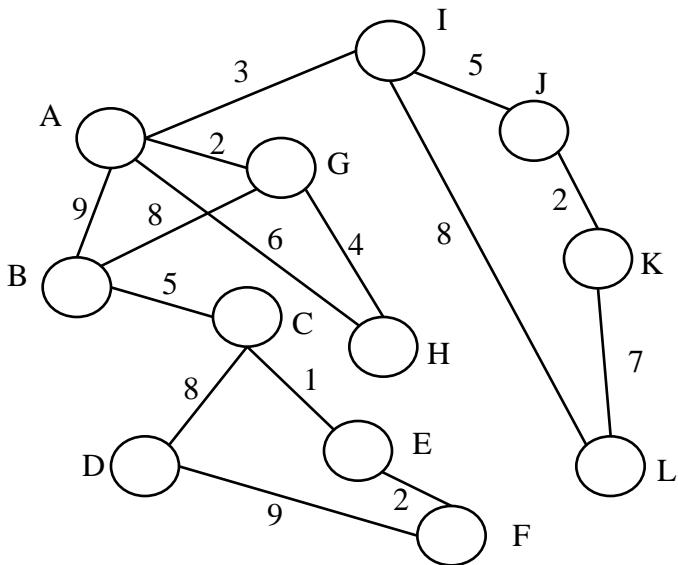
Complexitate Prim

- Depinde de implementare (vezi Dijkstra)
 - Matrice de adiacență $O(V^2)$
 - Heap binar $O(E \log V)$
 - Heap Fibonacci $O(V \log V + E)$
- Concluzii
- Grafuri dese
 - Matrice de adiacență preferată
- Grafuri rare
 - Heap binar sau Fibonacci

Algoritmul lui Kruskal

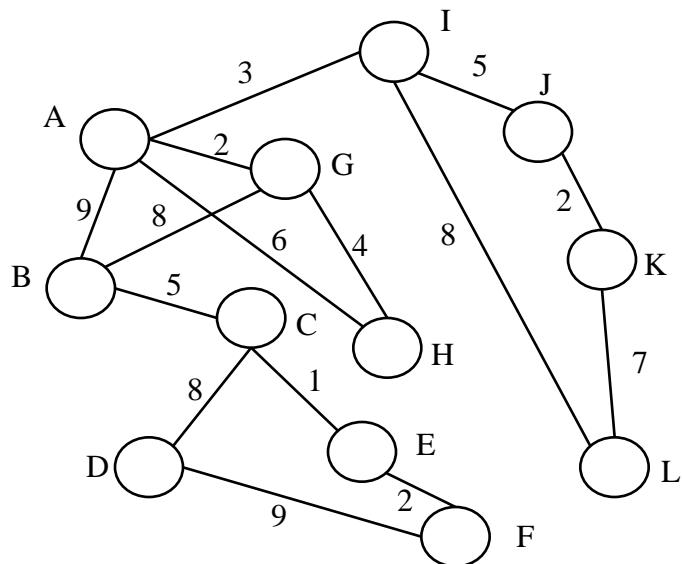
- Kruskal(G, w) Implementare în Java la [4] !
 - $A = \emptyset$; // inițializare AMA
 - **Pentru fiecare** ($v \in V$)
 - Constr_Arb(v) // creează o mulțime formată din nodul respectiv // (un arbore cu un singur nod)
 - Sortează_asc(E, w) // se sortează muchiile în funcție de // costul lor
 - **Pentru fiecare** $((u, v) \in E)$ // muchiile se extrag în ordinea // costului
 - **Dacă** $\text{Arb}(u) \neq \text{Arb}(v)$ **atunci** // verificăm dacă se creează ciclu
 - $\text{Arb}(u) = \text{Arb}(u) \cup \text{Arb}(v)$ // se reunesc mulțimile de noduri (arborii)
 - $A = A \cup \{(u, v)\}$ // se adaugă muchia sigură în AMA
 - **Întoarce** A

Exemplu (I)

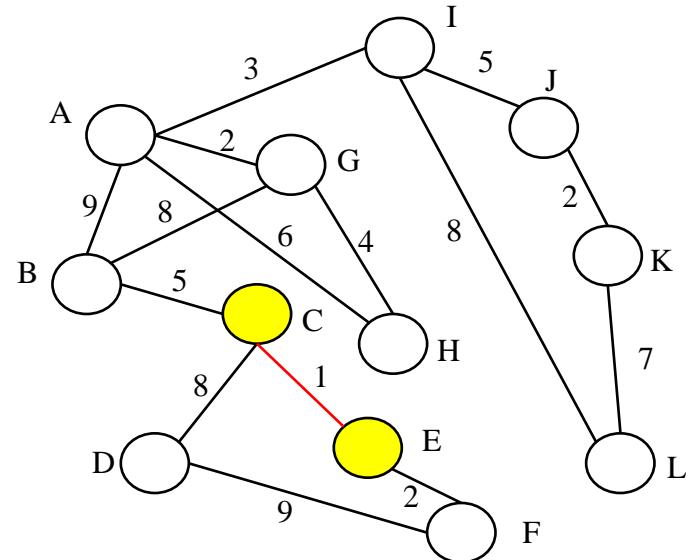


- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9

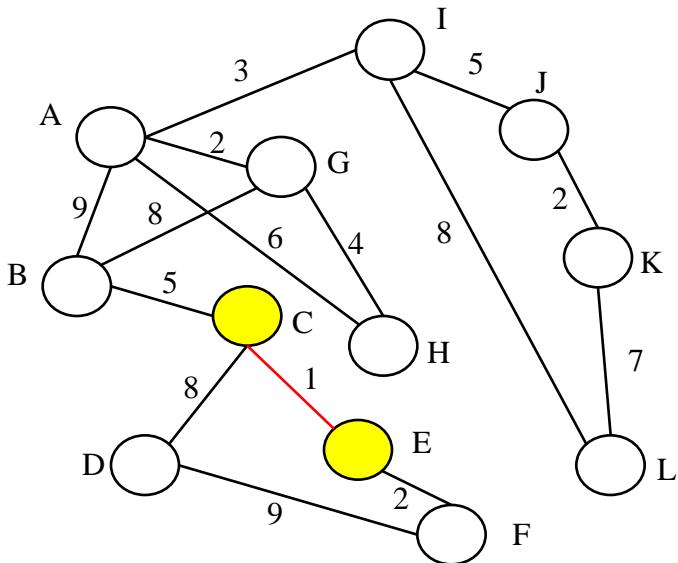
Exemplu (II)



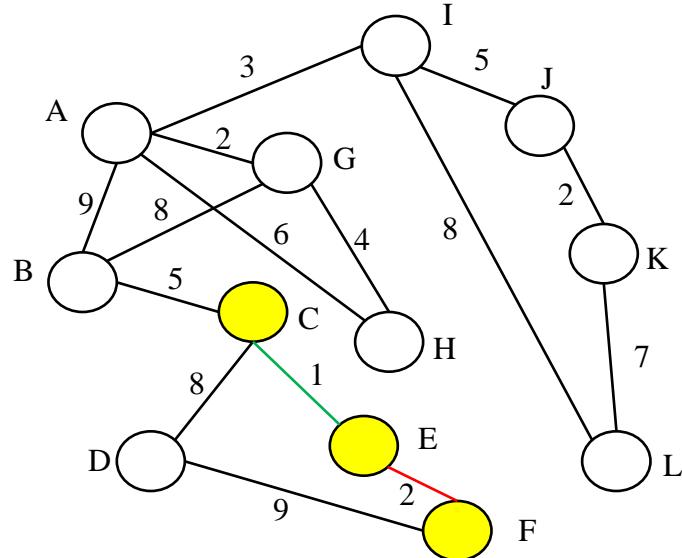
- CE -1
 - EF -2
 - AG-2
 - JK-2
 - AI-3
 - GH-4
 - BC-5
 - IJ-5
 - AH-6
 - KL-7
 - BG-8
 - CD-8
 - IL-8
 - AB-9



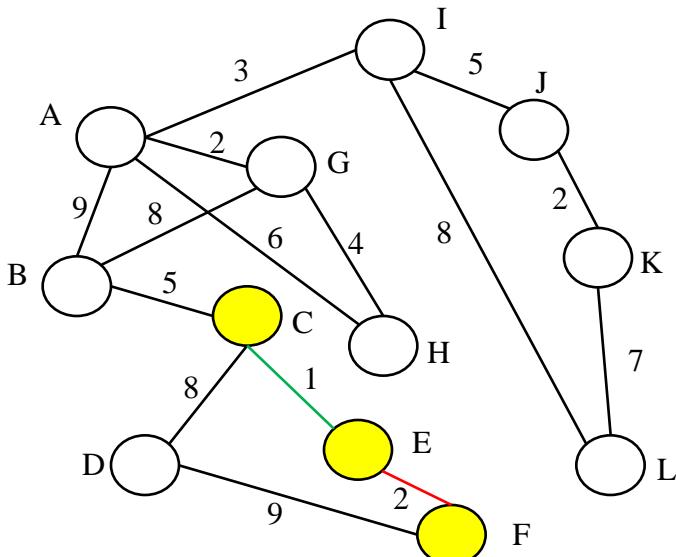
Exemplu (III)



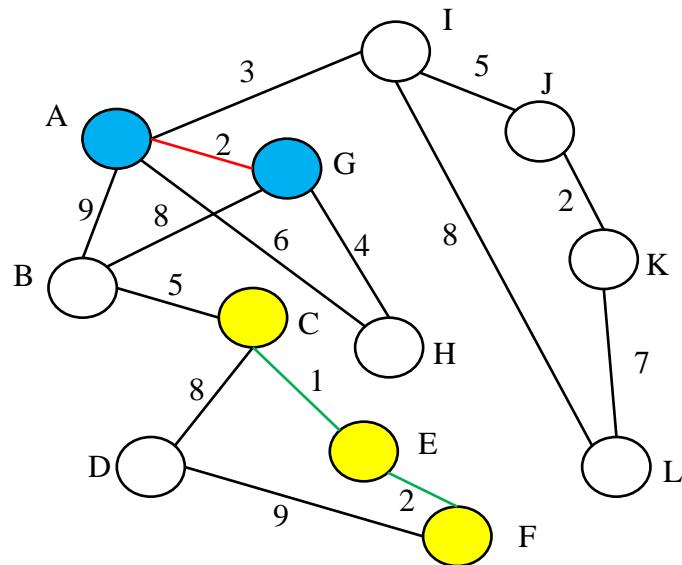
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



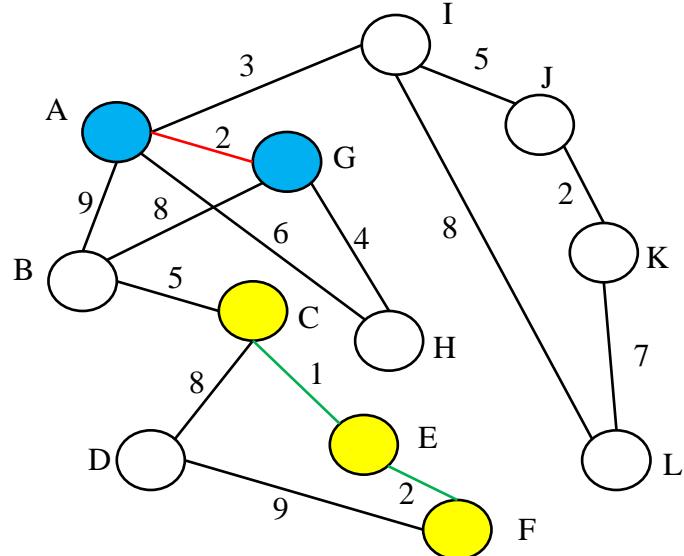
Exemplu (IV)



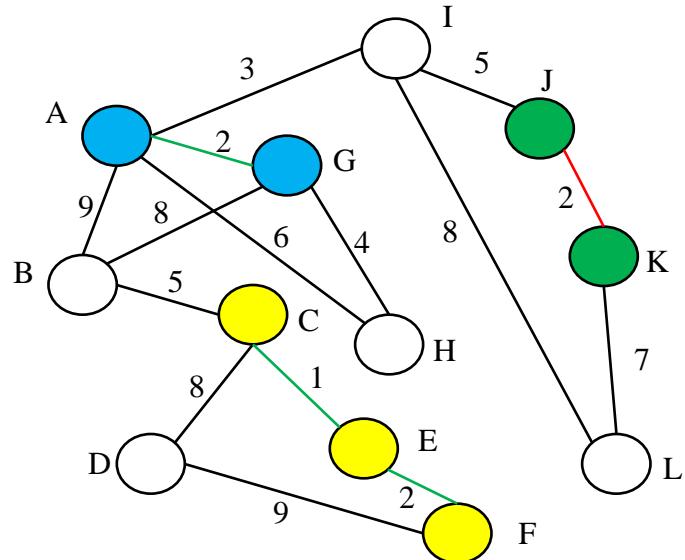
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



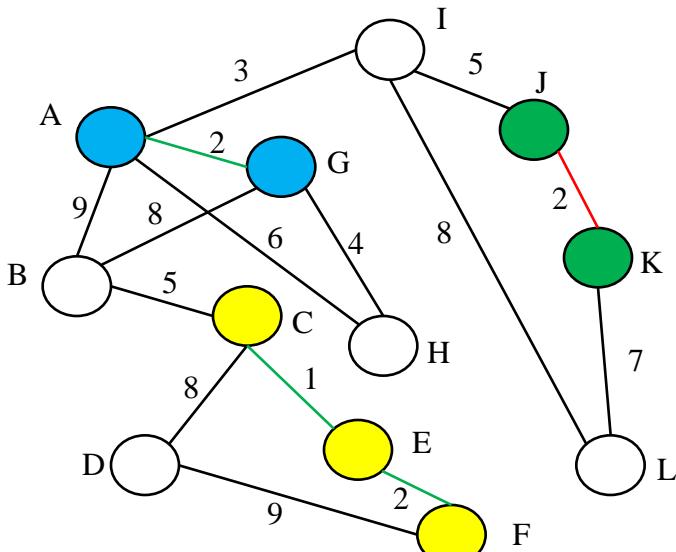
Exemplu (V)



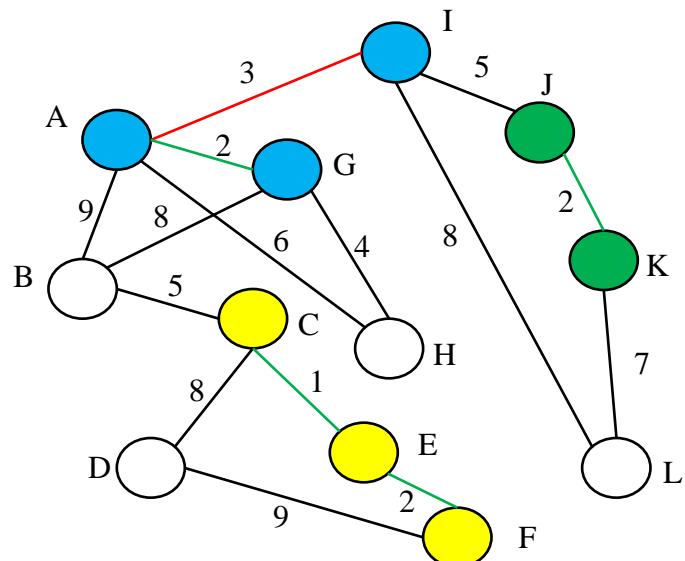
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



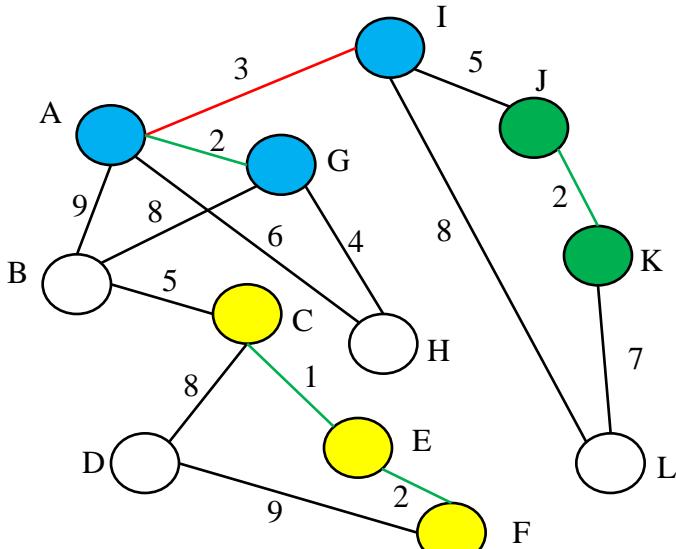
Exemplu (VI)



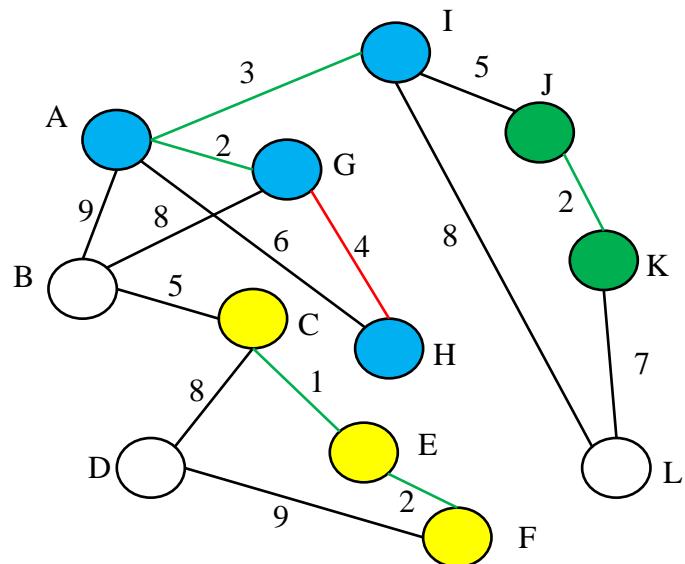
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



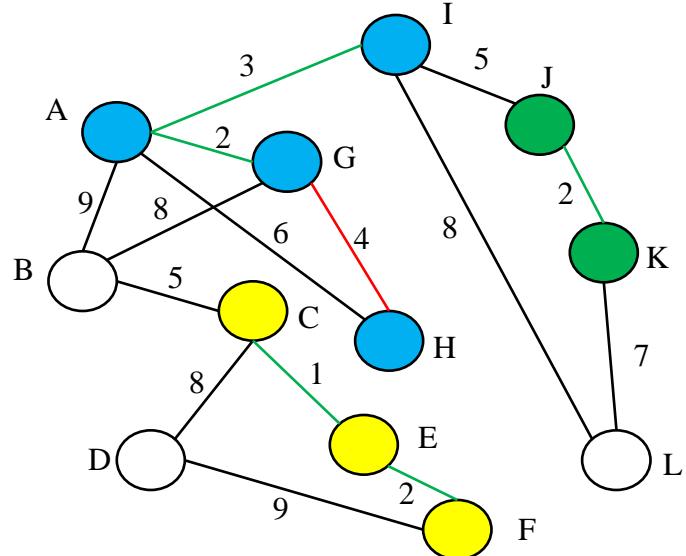
Exemplu (VII)



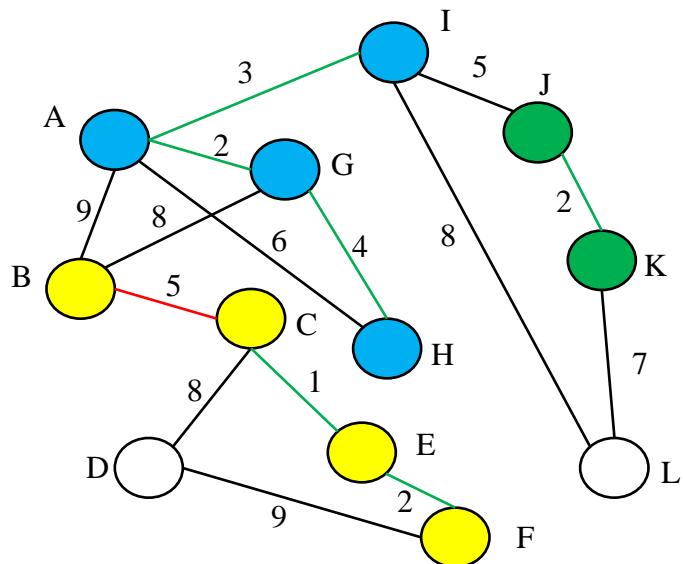
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



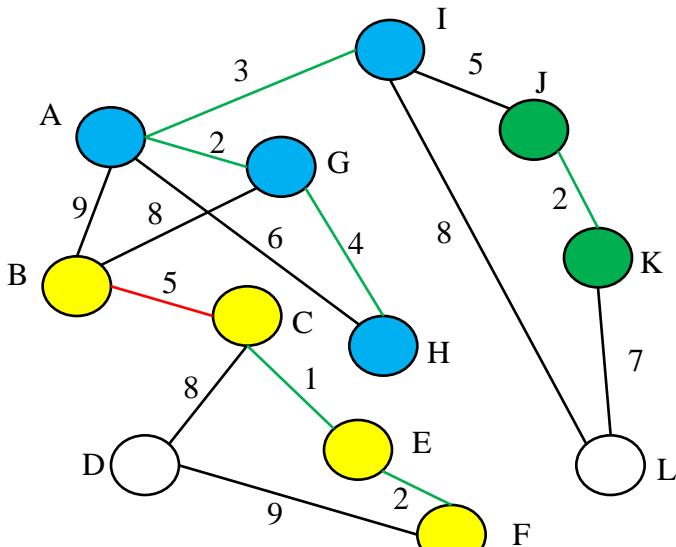
Exemplu (VIII)



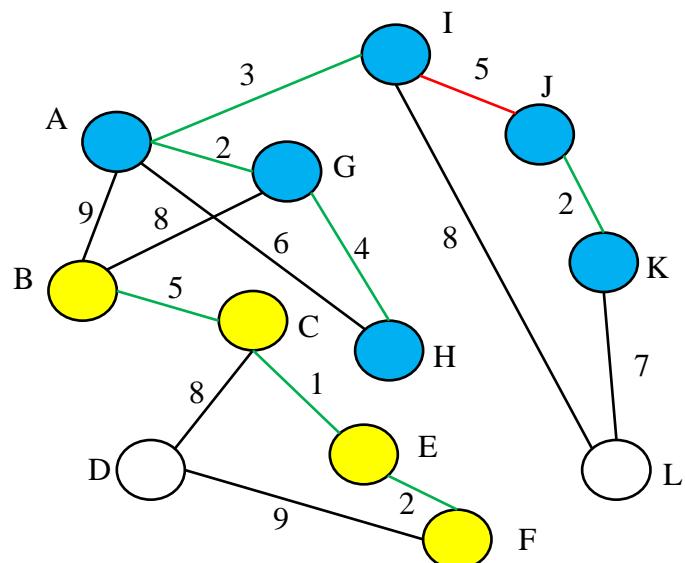
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5**
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



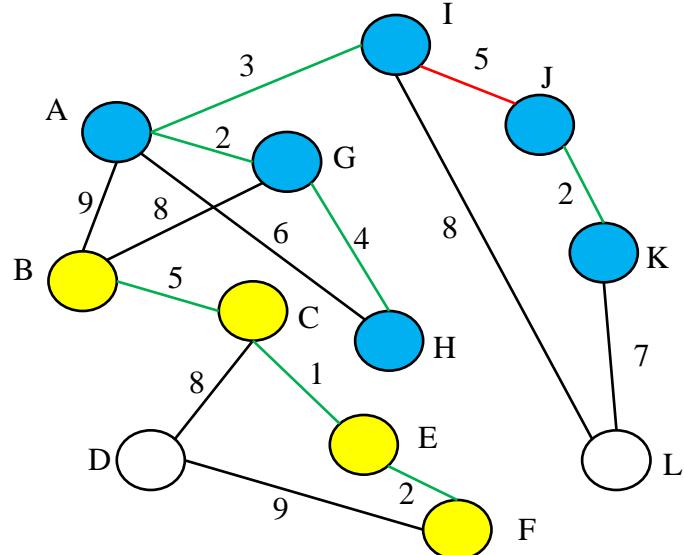
Exemplu (IX)



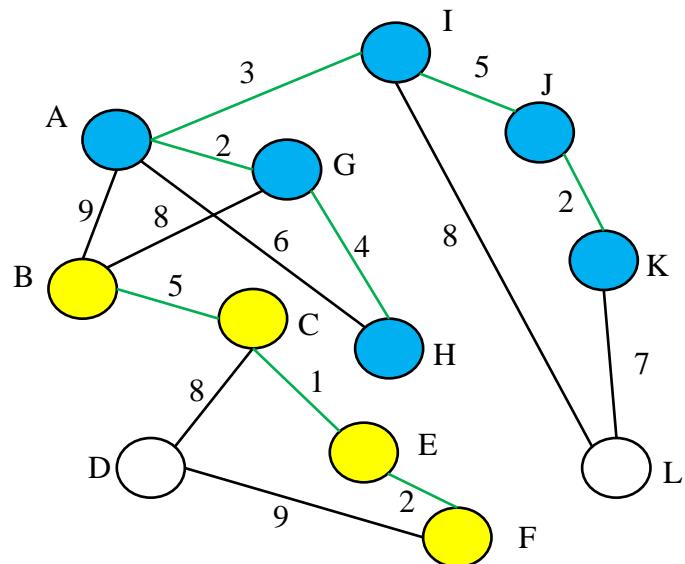
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



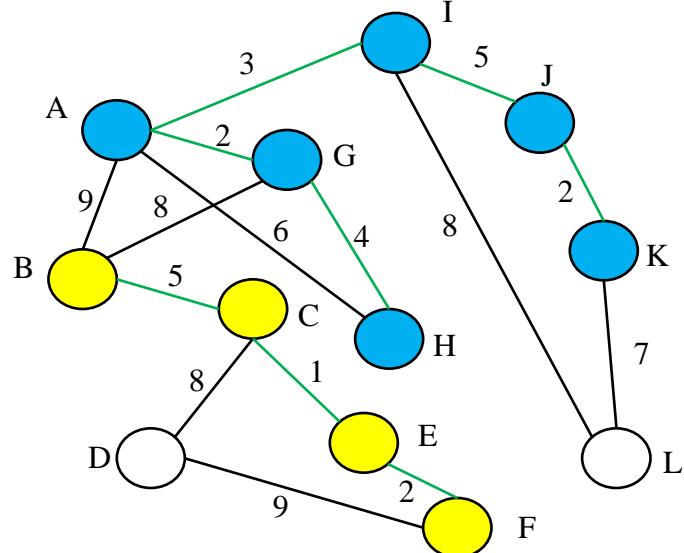
Exemplu (X)



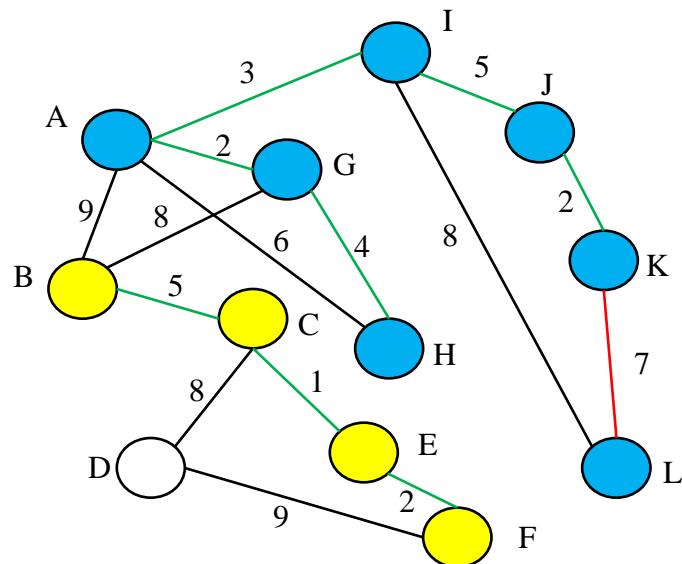
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



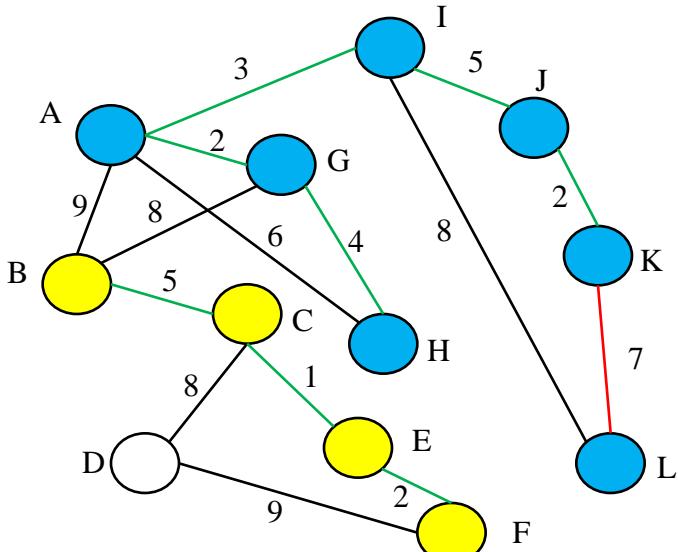
Exemplu (XI)



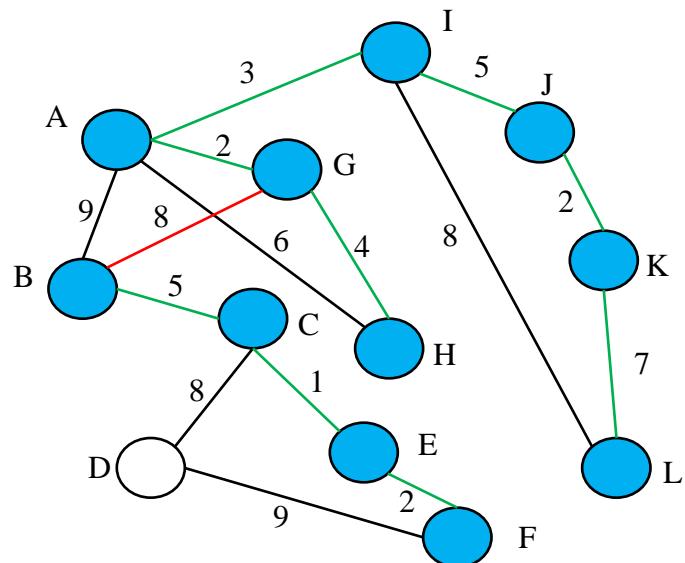
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



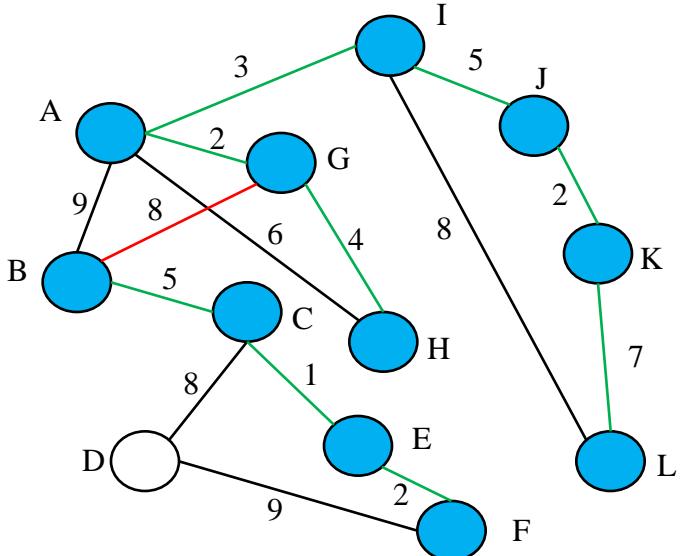
Exemplu (XII)



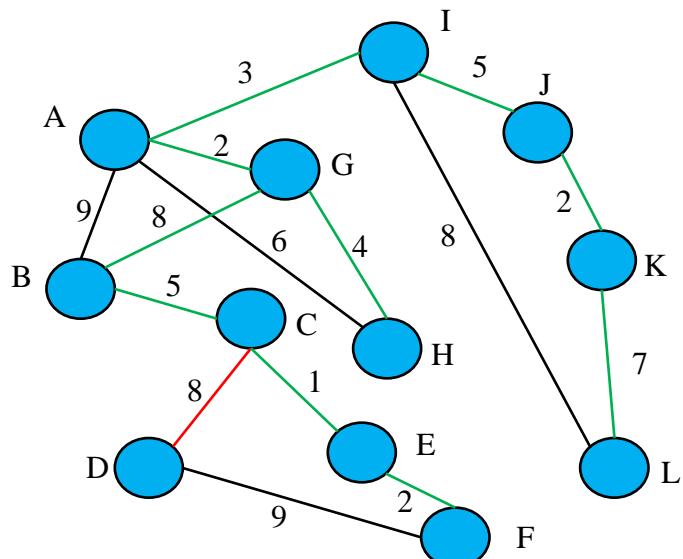
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



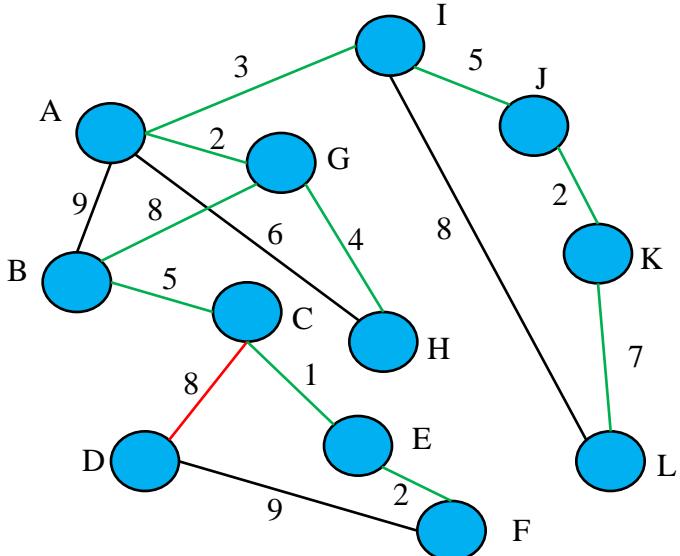
Exemplu (XIII)



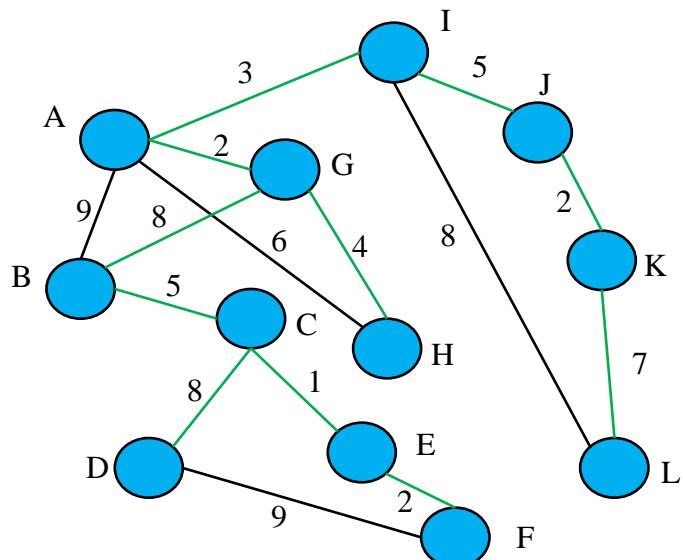
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



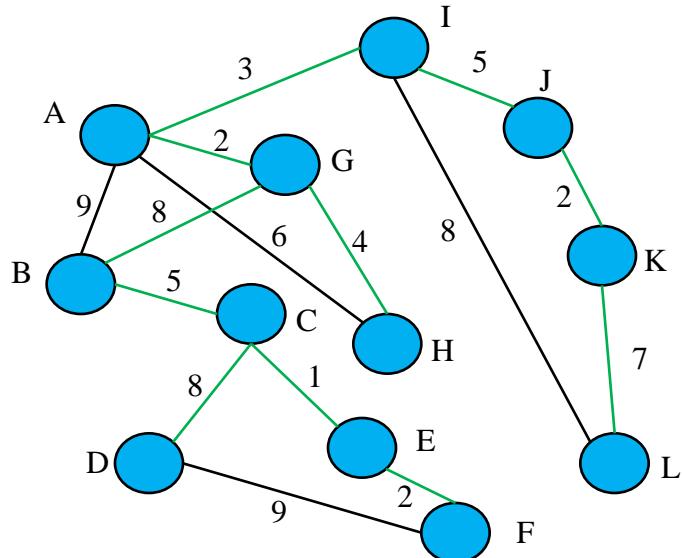
Exemplu (XIV)



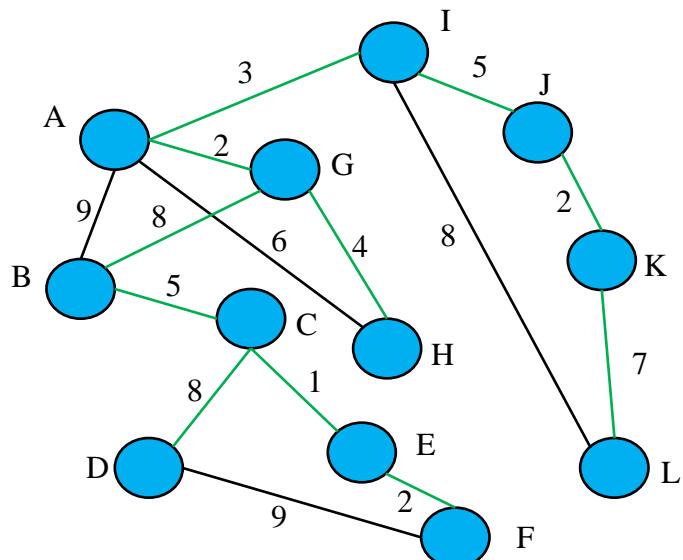
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



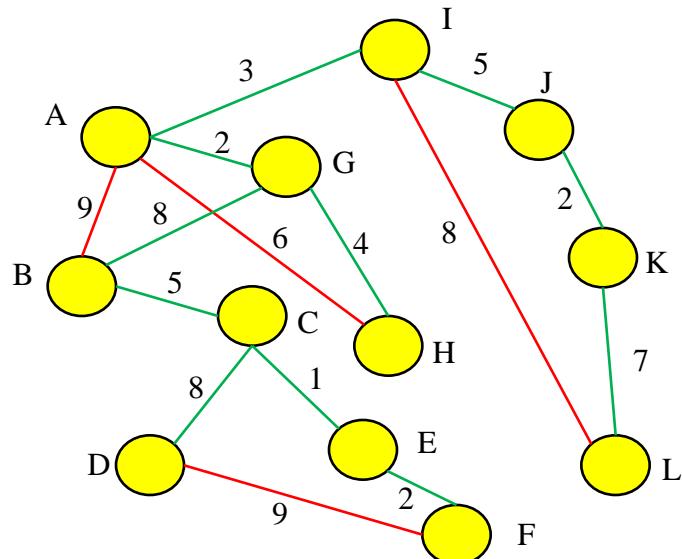
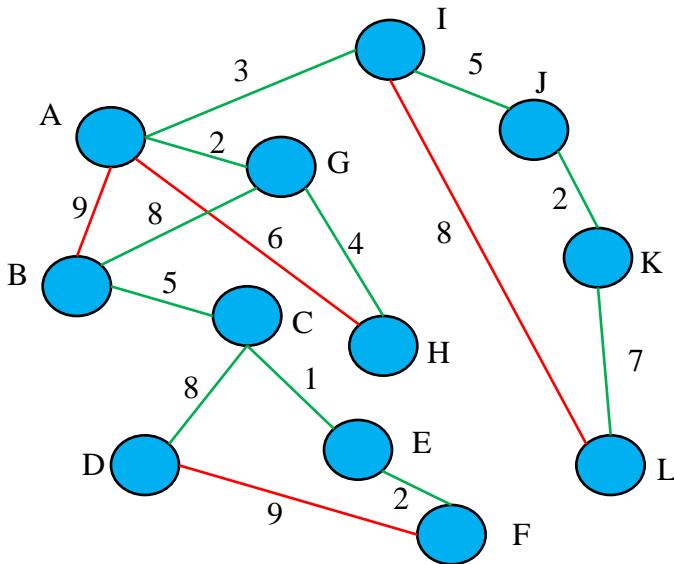
Exemplu (XV)



- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



Comparație Prim - Kruskal



Corectitudine (I)

- 1. arătăm că muchiile ignoreate nu fac parte din Arb:
 - Pp. (u,v) a.î. $\text{Arb}(u) = \text{Arb}(v)$
 - $\rightarrow (u,v)$ creează un ciclu în $\text{Arb}(u)$ (arborii sunt aciclici)
 - $w(u,v) = \max \{w(u',v') \mid (u',v') \in \text{Arb}(u)\}$ (din faptul că muchiile sunt sortate crescător)
 - \rightarrow din Propri. 1 $\rightarrow (u,v) \notin \text{Arb}$

Corectitudine (II)

- 2. arătăm că muchiile pe care le adăugăm aparțin Arb:
- Dem prin inducție după muchiile adăugate în AMA:
 - P_1 : Avem nodurile u și v , cu muchia (u,v) având proprietatea $w(u,v) = \min \{w(u',v') \mid (u',v') \in E\} \rightarrow$ din Propr. 2 $\rightarrow (u,v) \in \text{Arb.}$
 - $P_n \rightarrow P_{n+1}$:
 - $\text{Arb}(u) \neq \text{Arb}(v)$
 - $\rightarrow (u,v)$ muchie cu un capăt în $\text{Arb}(u)$
 - (u,v) are cel mai mic cost din muchiile cu un capăt în u (din faptul că **muchii sunt sortate crescător**)
 - \rightarrow din Propr. 2 $\rightarrow (u,v) \in \text{Arb}$

Algoritmul lui Kruskal

Complexitate?

- $\text{Kruskal}(G, w)$
 - $A = \emptyset$; // AMA
 - **Pentru fiecare** ($v \in V$)
 - $\text{Constr_Arb}(v)$ // creează o mulțime formată din nodul respectiv // (un arbore cu un singur nod)
 - $\text{Sortează_asc}(E, w)$ // se sortează muchiile în funcție de // costul lor
 - **Pentru fiecare** $((u, v) \in E)$ // muchiile se extrag în ordinea // costului
 - **Dacă** $\text{Arb}(u) \neq \text{Arb}(v)$ **atunci** // verificăm dacă se creează ciclu
 - $\text{Arb}(u) = \text{Arb}(u) \cup \text{Arb}(v)$ // se reunesc mulțimile de noduri (arborii)
 - $A = A \cup \{(u, v)\}$ // se adaugă muchia sigură în AMA
 - **Întoarce** A

Complexitate Kruskal

- Elementele algoritmului:
 - Sortarea muchiilor: $O(E \log E) \approx O(E \log V)$
 - $\text{Arb}(u) = \text{Arb}(v)$ – compararea a 2 multimi disjuncte
 $\{1,2,3\} \{4,5,6\}$ – mai precis trebuie identificat dacă 2 elemente sunt în aceeași multime
 - $\text{Arb}(u) \cup \text{Arb}(v)$ – reuniunea a 2 multimi disjuncte într-una singură
- → depinde de implementarea multimilor disjuncte

Variante de implementare multimi disjuncte (Var. 1) – contraexemplu

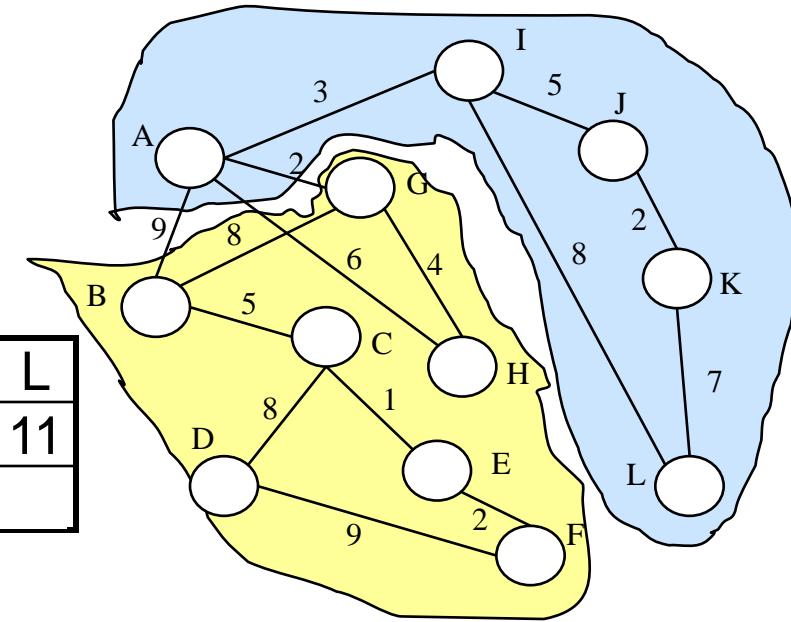
Multimile implementate ca vectori (populară la laborator ☺) –
NERECOMANDATĂ ☹

- Comparare (M_1, M_2)
 - Pentru fiecare ($u \in M_1$)
 - Pentru fiecare ($v \in M_2$)
 - Dacă ($u = v$) Întoarce true
 - Întoarce false
- Complexitate: V^2
- Reuniune (M_1, M_2)
 - Pentru i de la $\text{length}(M_1)$ la $\text{length}(M_1) + \text{length}(M_2)$
 - $M_1[i] = M_2[i - \text{length}(M_1)]$
 - Întoarce M_1
- Complexitate: V
 - numărul de apelări – E
- Complexitate totală: $E * V^2$

Variante de implementare multimi disjuncte (Var. 2) – Regăsire Rapidă

- Multimile - vectori
- Id - vector de id-uri conținând id-ul primului nod din componentă

A	B	C	D	E	F	G	H	I	J	K	L
0	1	2	3	4	5	6	7	8	9	10	11



Complexitate maximă?

- $\text{Arb}(u) \neq \text{Arb}(v)$
 - Complexitate?
- $\text{Arb}(u) = \text{Arb}(u) \cup \text{Arb}(v)$
 - Complexitate?

Regăsire rapidă (Complexitate)

- Compararea – $O(1)$ // Căutare în vector și verificare dacă au același id
- Reuniunea – $O(V)$ // trebuie să modifice toate id-urile nodurilor din una din multimi
- Complexitate maximă
 - $O(V * E)$ // E = numărul de reuniuni
- Inaceptabil pentru grafuri f mari

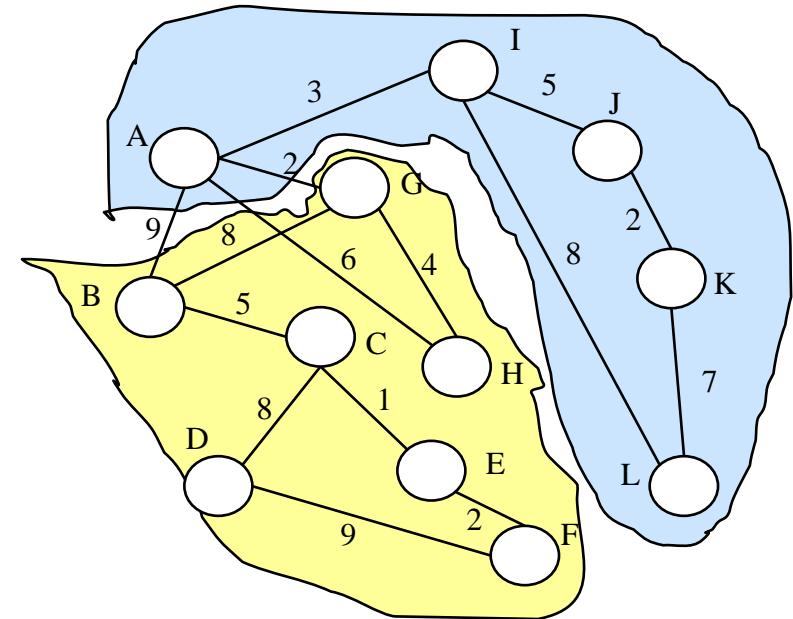
Variante de implementare multimi disjuncte (Var. 3) – Reuniune Rapidă

- se folosește tot un vector auxiliar de id-uri

A	B	C	D	E	F	G	H	I	J	K	L
0	1	2	3	4	5	6	7	8	9	10	11

- $\text{id}[i]$ reprezintă părintele lui i

- pentru rădăcina arborelui $\text{id}[i] = i$



Variante de implementare mulțimi disjuncte – reuniune rapidă

- Comparare (u, v)
 - Verifică dacă 2 noduri au aceeași rădăcină;
 - Implică identificarea rădăcinii:
- $\text{Arb}(u)$ // identificarea rădăcinii unei componente
 - Cât timp ($i \neq \text{id}[i]$) $i = \text{id}[i]$;
 - Întoarce i
- Comparare (u, v)
 - Întoarce $\text{Arb}(u) \neq \text{Arb}(v)$
- Reuniune (u, v) // implică identificarea rădăcinii
 - $v = \text{Arb}(v)$
 - $\text{id}[v] = u;$

Complexitate?

Reuniune rapidă (Complexitate)

Compararea – $O(V)$ // în cel mai rău caz, am o lista și trebuie să trec din părinte în părinte.

Reuniunea – $O(V)$ // implică regăsirea rădăcinii pentru a ști unde se face modificarea

Optimizarea reuniunii rapide (1)

- Reuniune rapidă balansată
- Se menține numărul de noduri din fiecare subarbore.
- Se adaugă arborele mic la cel mare pentru a face mai puține căutări → înălțimea arborelui e mai mică și numărul de căutări scade de la V la $\lg V$.
- Complexitate:
 - Compararea – $O(\lg V)$
 - Reuniune – $O(\lg V)$

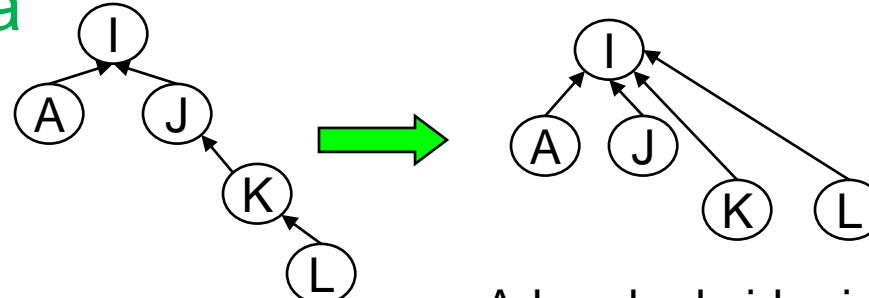
Optimizarea reuniunii rapide (2)

- Reuniune rapidă balansată cu compresia căii:

- Identificarea rădăcinii:

- Arb(u)
 - Cât timp ($i \neq id[i]$)
 - $id[i] = id[id[i]]$;
 - $i = id[i]$;
 - Întoarce i

- Menține o înălțime redusă a arborilor.



Arborele de noduri

Arborele de id-uri

K: $id[K] = id[J] = I$
L: $id[L] = id[K] = I$

Implementare
în Java și
exemplu la [4]

Complexitate după optimizări

- Orice secvență de E operații de căutare și reuniune asupra unui graf cu V noduri consumă $O(V + E^* \alpha(V, E))$.
- α – de câte ori trebuie aplicat Ig pentru a ajunge la 1.
 - În practică este ≤ 5 .
- \rightarrow În practică $O(E)$

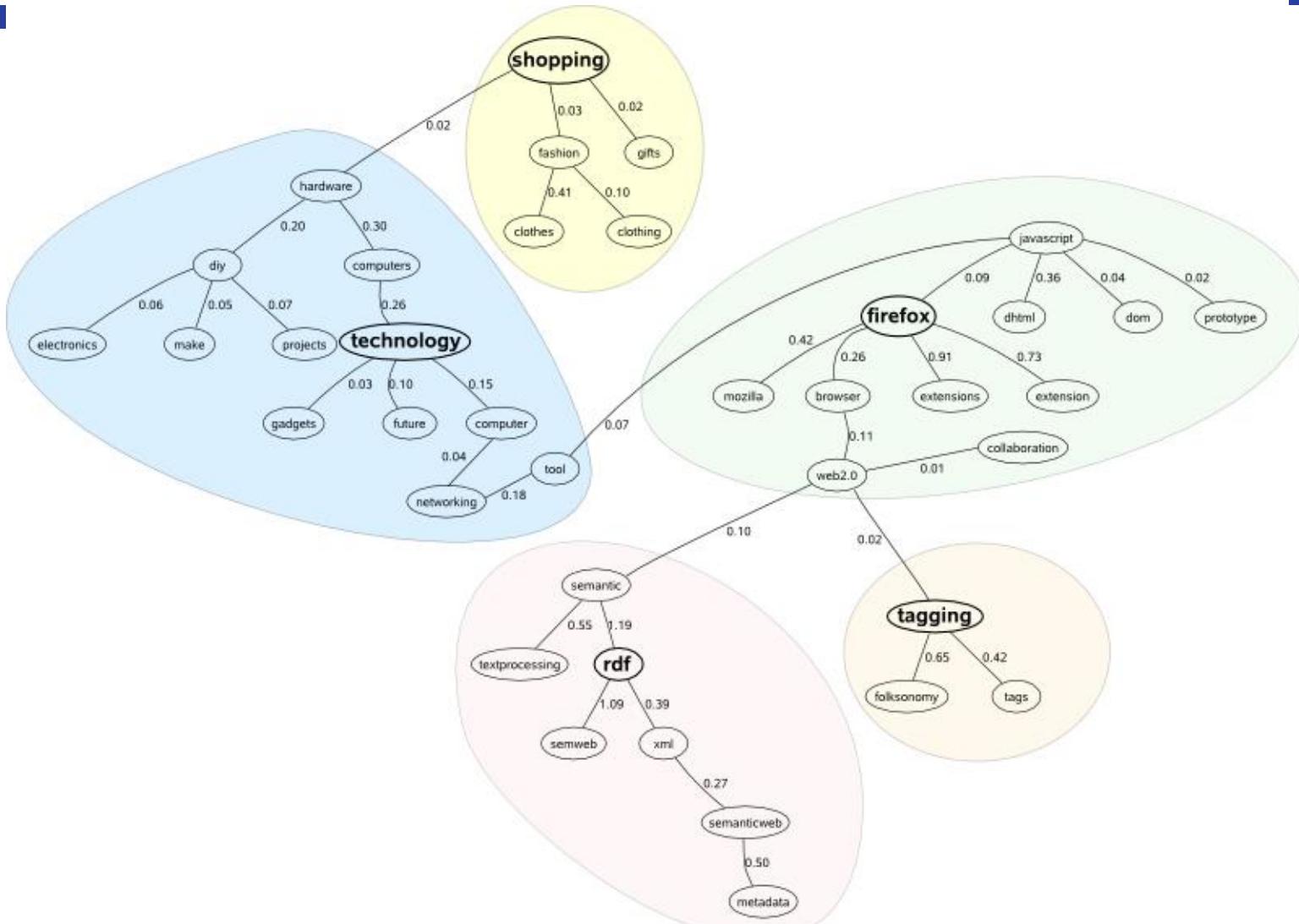
Complexitate Kruskal

- Max (complexitate sortare, complexitate operații multimi) = $\max(O(E \log V), O(E)) = O(E \log V)$
- → Complexitatea algoritmului Kruskal este dată de complexitatea sortării costurilor muchiilor.

Aplicație practică

- K-clustering
 - Împărțirea unui set de obiecte în grupuri astfel încât obiectele din cadrul unui grup să fie “apropiate” considerând o “distanță” dată.
- Utilizat în clasificare, căutare (web search de exemplu).
- Dându-se un întreg K să se împartă grupul de obiecte în K grupuri astfel încât spațiul dintre grupuri să fie maximizat.

Exemplu



Algoritm

- Se formează V clustere (un cluster per obiect).
- Găsește cele mai apropiate 2 obiecte din clustere diferite și unește cele 2 clustere.
- Se oprește când au mai rămas k clustere.
- → chiar algoritmul Kruskal

ÎNTREBĂRI?

Bibliografie curs 11

- | [1] C. Giumale – Introducere in Analiza Algoritmilor - cap. 5.6
- |
- | [2] Cormen – Introducere in algoritmi - cap. 27
- |
- | [3] Wikipedia -
http://en.wikipedia.org/wiki/Ford-Fulkerson_algorithm

Proiectarea Algoritmilor

Curs 11 – Rețele de flux. Flux maxim.

Bibliografie

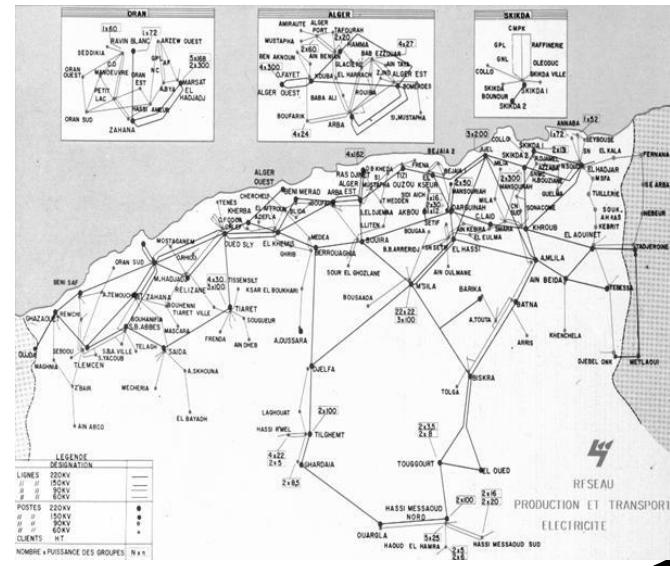
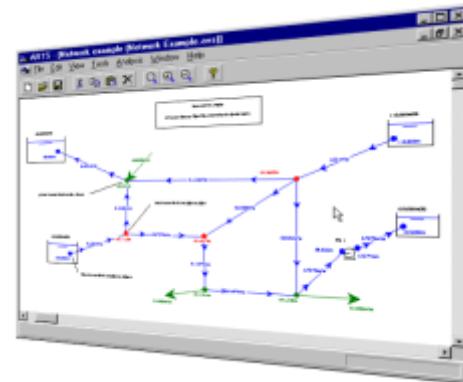
- [1] C. Giumale – Introducere în Analiza Algoritmilor - cap. 5.6
- [2] Cormen – Introducere în algoritmi - cap. Flux Maxim (27)
- [3] Wikipedia - http://en.wikipedia.org/wiki/Ford-Fulkerson_algorithm
- [4] R. Sedgewick, K Wayne – curs de algoritmi Princeton 2007 www.cs.princeton.edu/~rs/AlgsDS07/
01UnionFind si 14MST

Obiective

- Definirea conceptului de rețea de flux (sau de transport).
- Identificarea principaliilor algoritmi ce calculează fluxul maxim printr-o rețea.

Definirea problemei

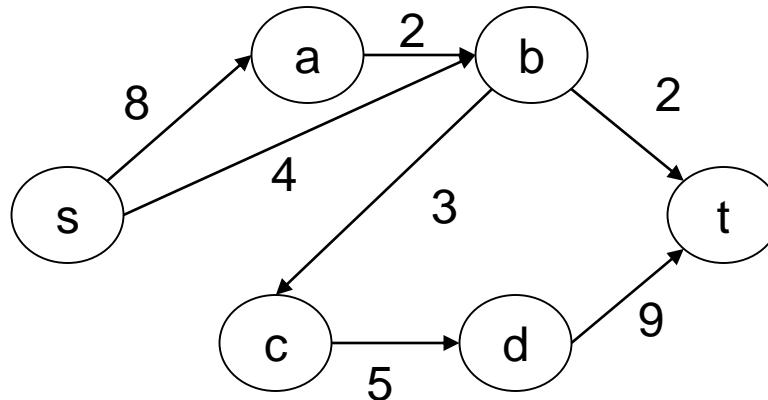
- Rețea ce transportă diferite materiale între un producător și o destinație.
 - Fiecare arc are o capacitate maximă de transport.
 - Trebuie identificat fluxul maxim ce poate fi transportat prin rețea.
 - Rețele:
 - Electrice;
 - Apă;
 - Informații;
 - Drumuri.



Rețea de flux – Definiție

- $G(V, E)$ graf **orientat**;
- $c(u, v) \geq 0 \quad \forall (u, v) - c =$ **capacitatea arcelor**;
- Dacă $(u, v) \notin E \rightarrow c(u, v) = 0$;
- S – **sursa traficului**;
- T – **destinația traficului (drena)**;
- Presupunem că $\forall u \in V \setminus \{s, t\} \exists s..u..t.$

Exemplu de rețea de flux

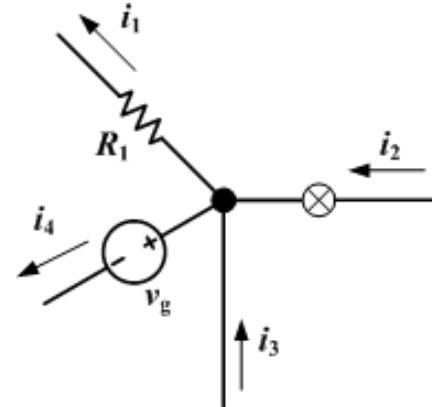
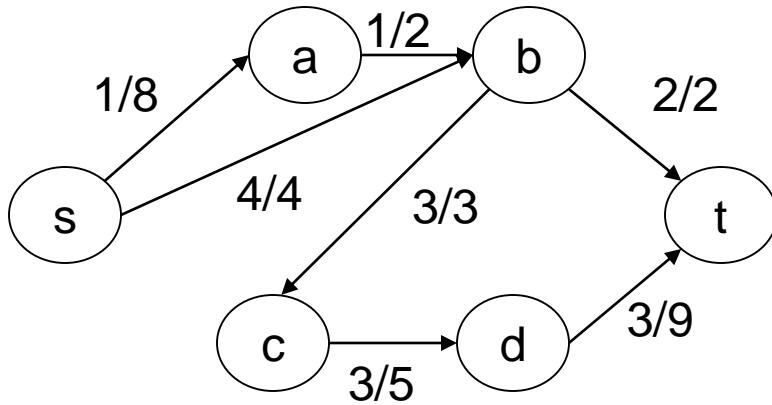


- s – sursa, t – destinația.
- Pe arce este reprezentată capacitatea arcului.

Flux. Definiție. Proprietăți.

- $G = (V, E)$ – rețea de flux;
- $c: V \times V \rightarrow \mathbb{R}$ - capacitatea rețelei;
- $f: V \times V \rightarrow \mathbb{R}$ - fluxul prin rețeaua G ;
- Proprietăți:
 - $\forall u, v \in V, f(u,v) \leq c(u,v)$ (fluxul printr-un arc este mai mic sau egal cu capacitatea arcului) – respectarea capacitații arcelor;
 - $\forall u, v \in V, f(u,v) = -f(v,u)$ – simetria fluxului;
 - $\sum f(u,v) = 0$ pentru $\forall u \in V \setminus \{s,t\}$ – conservarea fluxului.

Exemplu de fluxuri



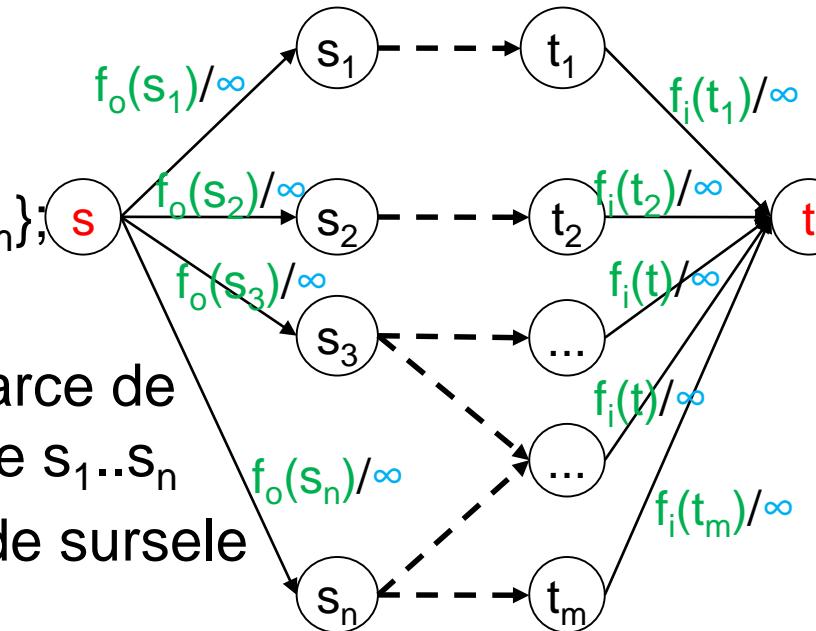
- $\sum f(u,v) = 0$ pentru $\forall u \in V \setminus \{s,t\}$ – fluxul se conservă;
- Proprietatea 3 = **legea curentului (Kirchoff)** ☺ - suma I. curenților ce intră într-un nod = suma I. curenților ce ieș din nodul respectiv.

Flux. Notații.

- $f(u,v)$ – fluxul din u spre v ;
- $f_i(u) = \sum f(v,u)$ – fluxul total care intra în nodul u ;
- $f_o(u) = \sum f(u,v)$ – fluxul total careiese din nodul u ;
- Valoarea totală a fluxului:
 - $|f| = \sum f(s,v) = f_o(s)$;
 - $|f| =$ fluxul ce părăsește sursa;
 - Cf. proprietăților P1-P3: $|f| = \sum f(s,v) = \sum f(v,t) = f_i(t)$.

Surse multiple, destinații multiple

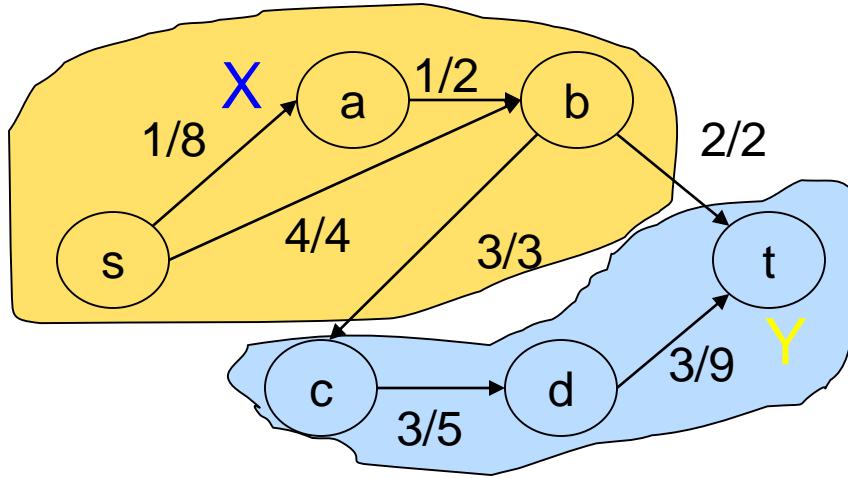
- Surse multiple $\{s_1, s_2, \dots, s_n\}$;
- Destinații multiple $\{t_1, t_2, \dots, t_m\}$;
- Se adaugă o **sursă unică** cu arce de **capacitate infinită** spre sursele $s_1..s_n$ și **flux egal cu fluxul generat** de sursele respective;
- Se adaugă o **destinație unică** t și arce de **capacitate infinită** între $t_1..t_m$ și t și **flux egal cu fluxul ce intră** în destinațiile respective.



Operații cu fluxuri

- X, Y – multimi de noduri;
- $f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y)$ = fluxul între X și Y ;
- **Operații:**
 - $\forall X \in V: f(X, X) = 0$;
 - $\forall X, Y \in V: f(X, Y) = -f(Y, X)$;
 - $\forall X, Y, Z \in V$ și Y inclus în X :
 - $f(X \setminus Y, Z) = f(X, Z) - f(Y, Z)$;
 - $f(Z, X \setminus Y) = f(Z, X) - f(Z, Y)$;
 - $\forall X, Y, Z \in V$ și $X \cap Y = \emptyset$:
 - $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$;
 - $f(Z, X \cup Y) = f(Z, X) + f(Z, Y)$
 - $f(s, V) = f(V, t)$

Exemplu operații fluxuri (1)

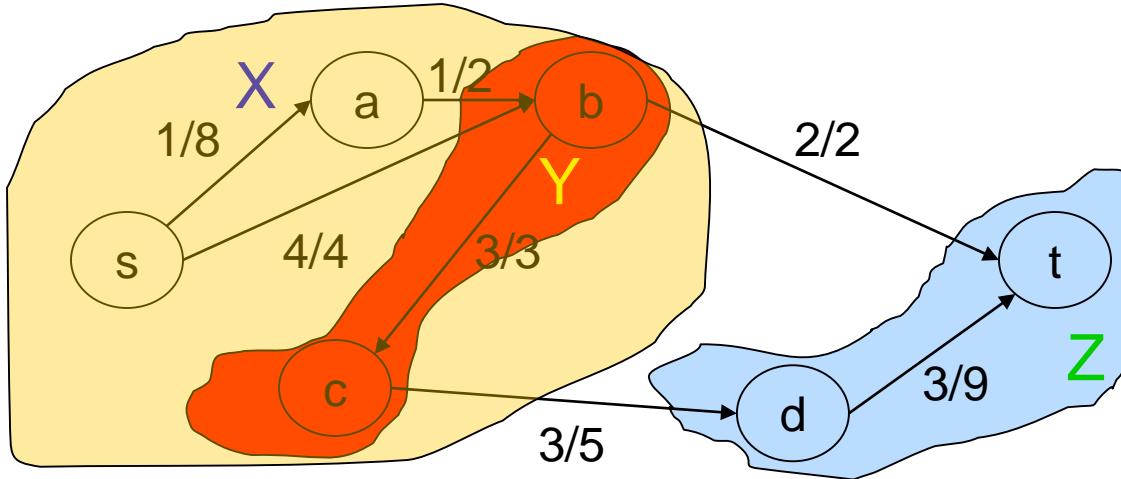


$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y)$$

$$f(X, X) = f(s, a) + f(a, s) + f(s, b) + f(b, s) + f(a, b) + f(b, a) = 0$$

$$f(X, Y) = f(b, c) + f(b, t) = -f(c, b) - f(t, b) = -f(Y, X)$$

Exemplu operații fluxuri (2)



$\forall X, Y, Z \in V$ și Y inclus în X

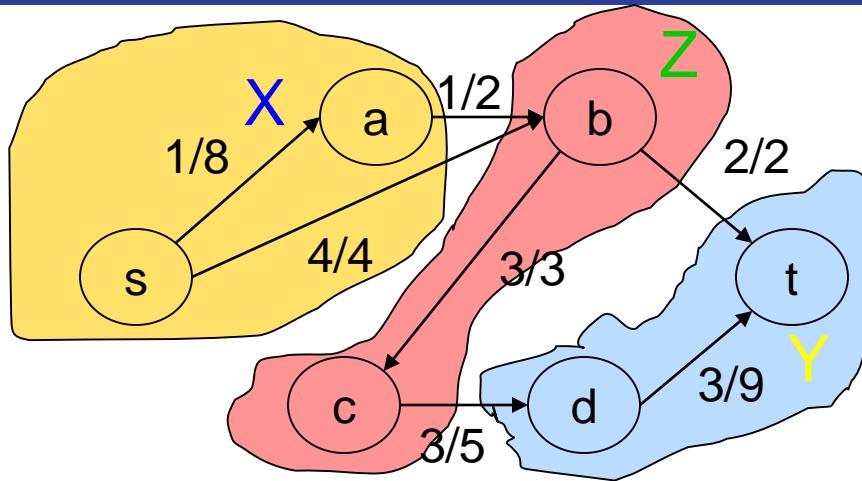
$$f(X \setminus Y, Z) = f(X, Z) - f(Y, Z)$$

$$f(Z, X \setminus Y) = f(Z, X) - f(Z, Y)$$

$$f(X \setminus Y, Z) = 0 = f(b,t) + f(c,d) - f(b,t) - f(c,d) = f(X,Z) - f(Y,Z)$$

$$f(Z, X \setminus Y) = 0 = f(t,b) + f(d,c) - f(t,b) - f(d,c) = f(Z,X) - f(Z,Y)$$

Exemplu operații fluxuri (3)



$\forall X, Y, Z \in V$ și $X \cap Y = \emptyset$

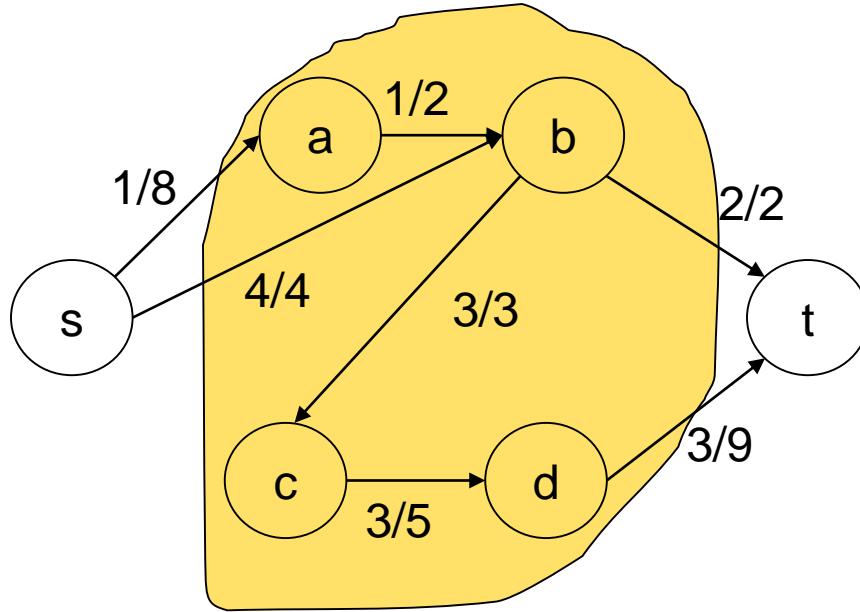
$$f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$$

$$f(Z, X \cup Y) = f(Z, X) + f(Z, Y)$$

$$f(X \cup Y, Z) = f(s,b) + f(a,b) + f(t,b) + f(d,c) = f(X, Z) + f(Y, Z)$$

$$f(Z, X \cup Y) = f(b,a) + f(b,s) + f(b,t) + f(c,d) = f(Z, X) + f(Z, Y)$$

Exemplu operații fluxuri (4)



$$f(s, V) = f(V, t)$$

$$f(s, V) = f(s,a) + f(s,b) = 5 = f(d,t) + f(b,t) = f(V, t)$$

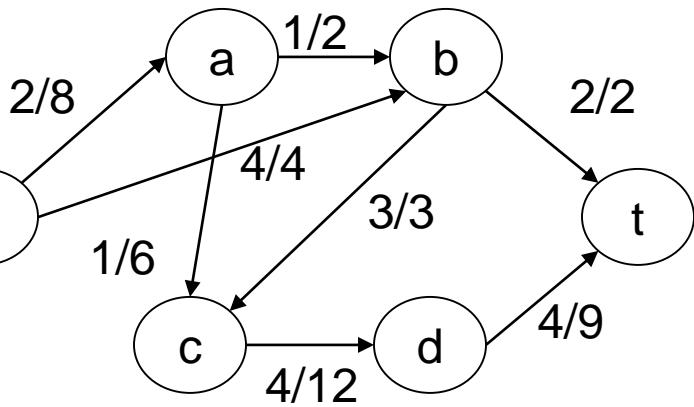
Arc rezidual. Capacitate reziduală.

- Definiție: Un arc (u,v) pentru care $f(u,v) < c(u,v)$ se numește **arc rezidual**.
- → Fluxul pe acest arc se poate mări.
- Definiție: Cantitatea cu care se poate mări fluxul pe arcul (u,v) se numește **capacitatea reziduală a arcului (u,v)** ($c_f(u,v)$):
$$c_f(u,v) = c(u,v) - f(u,v)$$

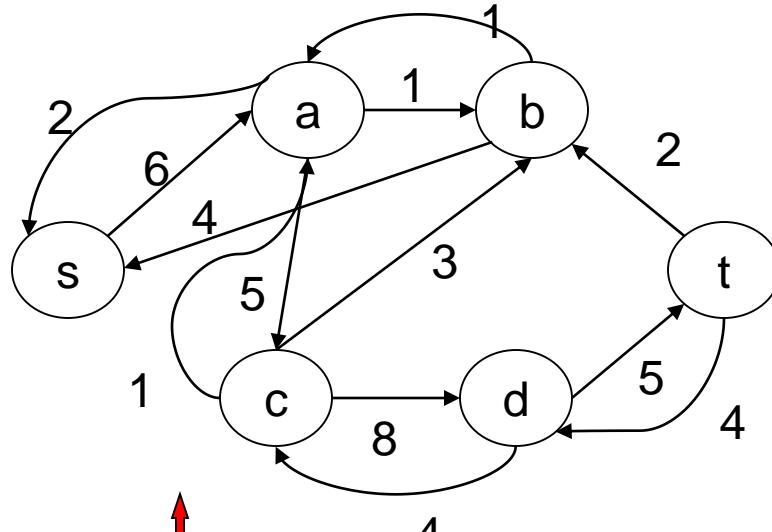
Rețea reziduală. Cale reziduală.

- $G = (V, E)$ rețea de flux cu funcția de capacitate c .
- Definiție: Rețeaua reziduală ($G_f = (V, E_f)$) este o rețea de flux formată din arcele ce admit creșterea fluxului:
$$E_f = \{(u, v) \in V \times V \mid c_f(u, v) > 0\}.$$
- Observație: $E_f \not\subseteq E!!!$
- Definiție: Cale reziduală e un drum s..t din G_f , unde $c_f(u, v)$ este capacitatea reziduală a arcului (u, v) .
- Definiție: Capacitatea reziduală a căii = capacitatea reziduală minimă de pe calea s..t descoperită.

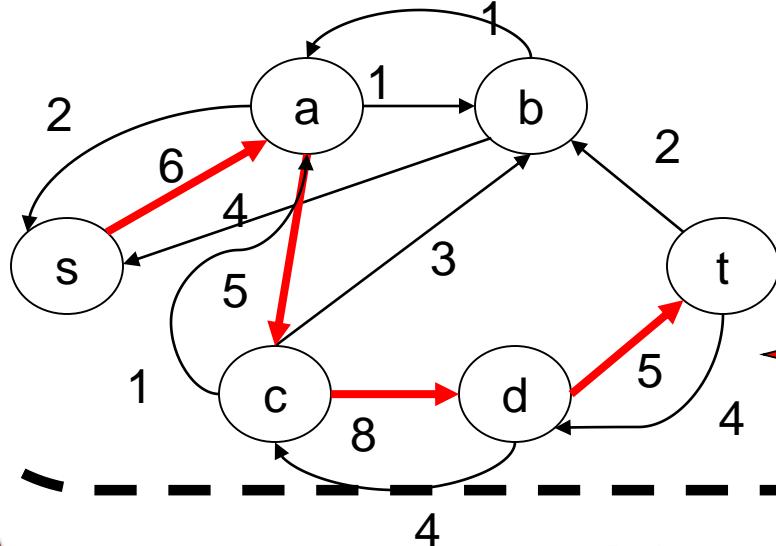
Exemplu rețea reziduală



$$c_f(u,v) = c(u,v) - f(u,v)$$



Rețeaua reziduală $G_f = (V, E_f)$ unde
 $E_f = \{(u,v) \in V \times V \mid c_f(u,v) > 0\}$



Calea reziduală: $s \rightarrow a \rightarrow c \rightarrow d \rightarrow t$
 Capacitatea reziduală a căii:
 $c_f(p) = \min\{6, 5, 8, 5\} = 5$

Rețea reziduală

- Lemă 5.16: Fie $G = (V, E)$ rețea de flux, f fluxul în G , G_f rețeaua reziduală a lui G . Fie f' un flux prin G_f și $f+f'$ o funcție definită astfel:
$$f+f' (u, v) = f(u, v) + f'(u, v).$$
- Atunci $f+f'$ reprezintă un flux în G și
$$|f+f'| = |f| + |f'|$$
- Această Lemă ne spune cum putem mări fluxul printr-o rețea de flux.

Flux în rețeaua reziduală

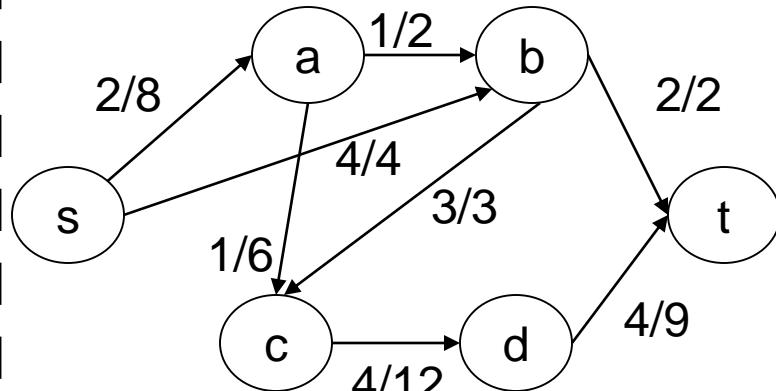
- Lemă 5.17: G – rețea de flux, f flux în G , $p = s..t$ – cale reziduală în G_f , $f_p: V \times V \rightarrow \mathbb{R}$ se definește ca fiind:

$$f_p(u,v) = \begin{cases} c_f(p), & \text{dacă } (u,v) \in p \\ -c_f(p), & \text{dacă } (v,u) \in p \\ 0, & \text{dacă } (u,v) \text{ și } (v,u) \notin p \end{cases}$$

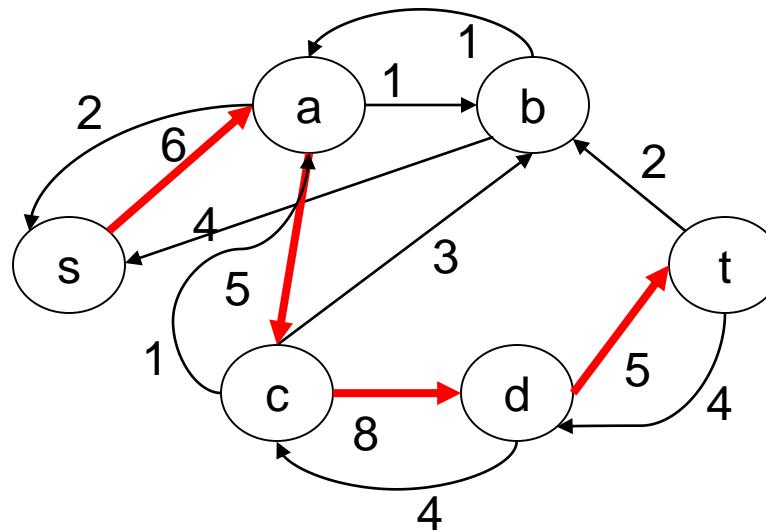
f_p = flux în G_f ; $|f_p| = c_f(p)$

- Corolar 5.4: $f' = f + f_p$ = flux în G , astfel încât $|f'| = |f| + |f_p| > |f|$.
- Această Lemă ne spune cum se definește fluxul printr-o rețea reziduală.

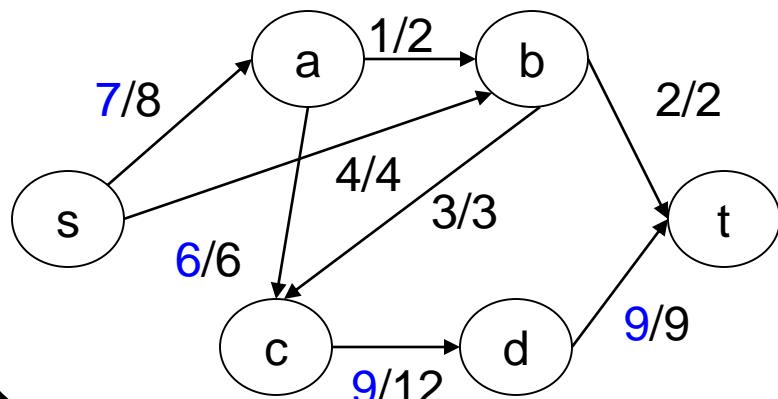
Exemplu maximizare flux



$$f(s..t) = 2 + 4 = 6$$



$$|f_p(s..t)| = c_f(s..t) = 5$$



$$f'(s..t) = f + f_p = 6 + 5 = 11$$

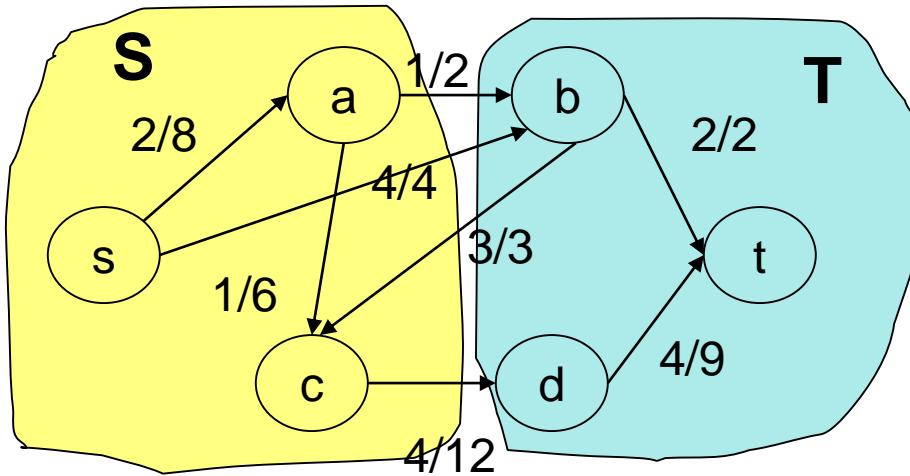
Calculul fluxului maxim

- Metoda Ford-Fulkerson
 - $f(u,v) = 0 \forall (u,v)$ // inițializarea fluxului
 - Repetă // creștere iterativă a fluxului
 - găsește un drum s..p..t pe care se poate mări fluxul (cale reziduală)
 - $f = f + \text{flux}(s..p..t)$
 - Până când nu se mai poate găsi nici un drum s..p..t
 - Întoarce f
- În funcție de metodele de identificare a căii reziduale există mai mulți algoritmi ce urmează această metodă.

Tăieturi în rețele de flux

- **Definiție:** O tăietură (S, T) a unei rețele de flux $G = \langle V, E, f, c \rangle$ este o partiționare a nodurilor în 2 multimi disjuncte S și $T = V \setminus S$ astfel încât $s \in S$ și $t \in T$.
 - $f(S, T) = \sum_{x \in S} \sum_{y \in T} f(x, y)$ – fluxul prin tăietura (S, T)
 - $c(S, T) = \sum_{x \in S} \sum_{y \in T} c(x, y)$ – capacitatea tăieturii (S, T)
- **Lema 5.18:** Fluxul prin tăietură = fluxul prin rețea – $f(S, T) = |f|$
- **Corolar 5.5:** S, T – tăietură oarecare – fluxul maxim este limitat superior de capacitatea tăieturii $|f| \leq c(S, T)$

Exemplu de tăietură într-o rețea de flux



- $f(S, T) = f(s, b) + f(a, b) + f(c, d) + f(c, b)$
 $= 4 + 1 + 4 - 3 = 6 = f(s, V)$
- $c(S, T) = c(a, b) + c(s, b) + c(c, d) = 18$

Flux maxim – tăietură minimă

- Teorema 5.25 (Flux maxim – tăietură minimă): $G = (V, E)$ rețea de flux – următoarele afirmații sunt echivalente:
 - f este o funcție de flux în G astfel încât $|f|$ este flux maxim total în G ;
 - rețeaua reziduală G_f nu are căi reziduale;
 - există o tăietură (S, T) astfel încât $|f| = c(S, T)$.

Algoritmul Ford – Fulkerson

- Ford – Fulkerson(G, s, t)
 - Pentru fiecare (u, v) din E
 - $f(u, v) = f(v, u) = 0$ // inițializare
 - Cât timp
 - Există o cale reziduală p între $s..t$ în G_f
 - $c_f(p) = \min\{c_f(u, v) \mid (u, v) \text{ din } p\}$ // capacitatea reziduală
 - Pentru fiecare (u, v) din p
 - $f(u, v) = f(u, v) + c_f(p)$
 - $f(v, u) = -f(u, v)$
 - Întoarce $|f|$

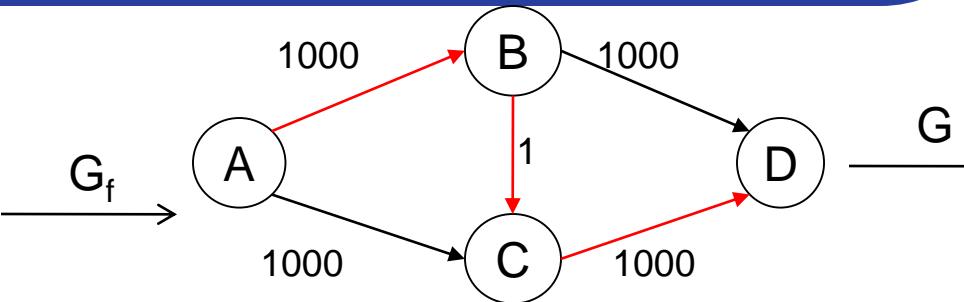
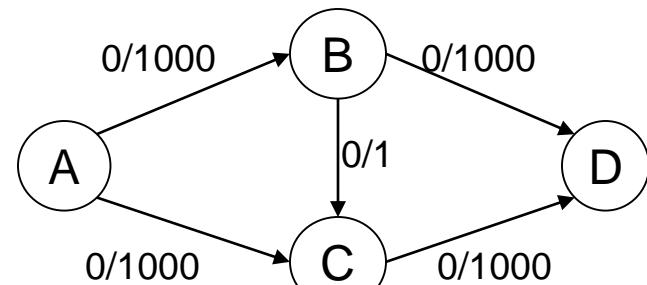
Complexitate?

Algoritmul Ford – Fulkerson (2)

- Ford – Fulkerson(G, s, t)
 - Pentru fiecare (u, v) din E
 - $f(u, v) = f(v, u) = 0$ // $O(E)$
 - Cât timp // $O(?)$
 - Există o cale reziduală p între $s..t$ în G_f // $O(E)$
 - $c_f(p) = \min\{c_f(u, v) \mid (u, v) \text{ din } p\}$ // $O(E)$
 - Pentru fiecare (u, v) din p // $O(E)$
 - $f(u, v) = f(u, v) + c_f(p)$
 - $f(v, u) = -f(u, v)$
 - Întoarce $|f|$

Complexitate?

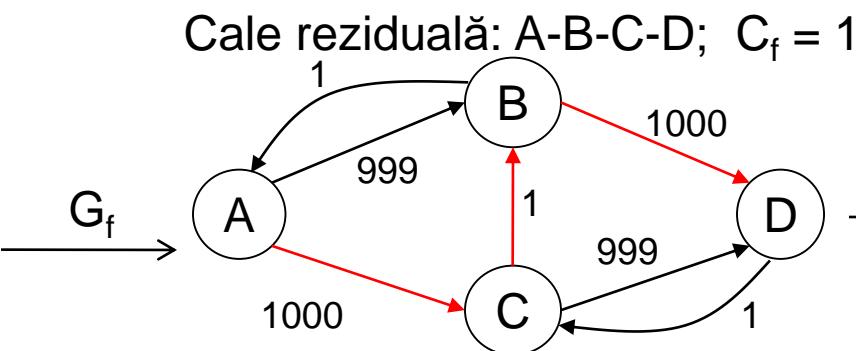
Exemplu Ford – Fulkerson (1)



```

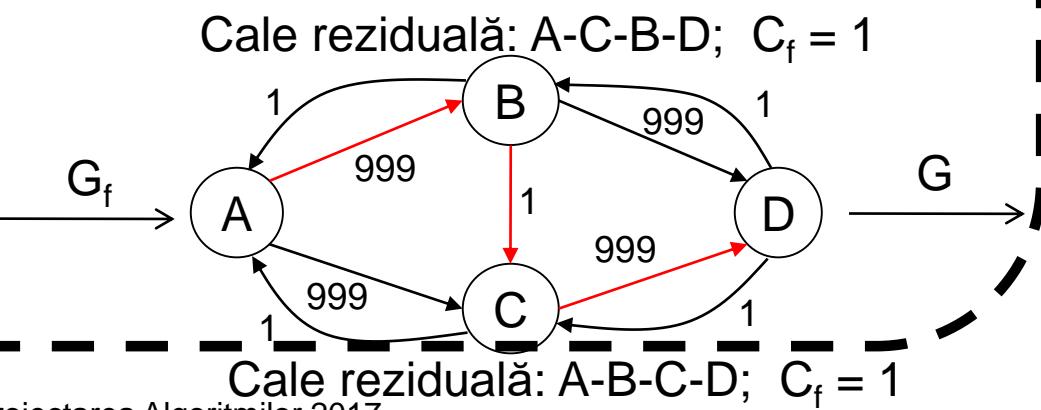
graph LR
    A((A)) -- "1/1000" --> B((B))
    A((A)) -- "0/1000" --> C((C))
    B((B)) -- "0/1000" --> D((D))
    B((B)) -- "1/1" --> C((C))
    C((C)) -- "1/1000" --> D((D))
    C((C)) -- "0/1000" --> B((B))

```

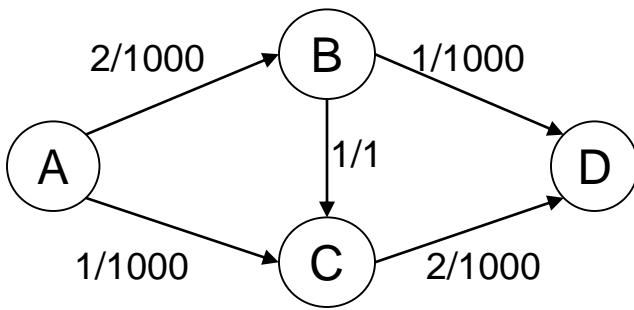


```

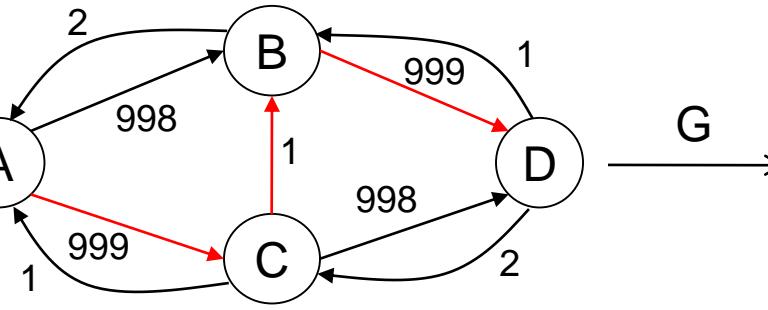
graph TD
    A((A)) -- "1/1000" --> B((B))
    A((A)) -- "1/1000" --> C((C))
    B((B)) -- "1/1000" --> C((C))
    B((B)) -- "1/1000" --> D((D))
    C((C)) -- "1/1000" --> D((D))
    C((C)) -- "1/1000" --> E((E))
    D((D)) -- "0/1" --> E((E))
  
```



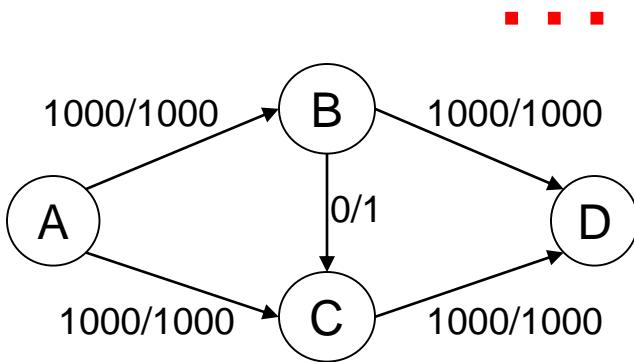
Exemplu Ford – Fulkerson (2)



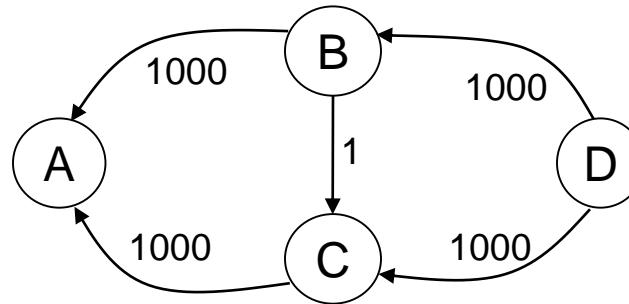
G_f



Cale reziduală: A-C-B-D; $C_f = 1$



G_f



Cale reziduală: \emptyset

După câți pași se ajunge la forma finală?

Complexitate Ford – Fulkerson

- Complexitate $O(E * f_{\max})$
- f_{\max} = fluxul maxim

Algoritmul Ford – Fulkerson – discuție

- Probleme ce pot să apară:
 - Se folosesc căi cu capacitate mică;
 - Se pun fluxuri pe mai multe arce decât este nevoie.
- Îmbunătățiri:
 - Se aleg căile reziduale cu capacitate maximă – complexitatea va depinde în continuare de f_{\max} și de valoarea capacitaților;
 - Se aleg căile reziduale cele mai scurte → în acest caz complexitatea nu mai depinde de f_{\max} ci numai de numărul de arce (ex. Edmonds-Karp: identificarea căilor reziduale minime prin aplicarea unui BFS)

Algoritmul Edmonds – Karp (1)

- **Edmonds – Karp(G, s, t)**
 - **Pentru fiecare** (u,v) din E
 - $f(u,v) = f(v,u) = 0$ // inițializare
 - **Cât timp**
 - Există căi reziduale între $s..t$ în G_f
 - Determină calea reziduală minimă p aplicând BFS
 - $c_f(p) = \min\{c_f(u,v) \mid (u,v) \text{ din } p\}$ // capacitatea reziduală
 - **Pentru fiecare** (u,v) din p
 - $f(u,v) = f(u,v) + c_f(p)$
 - $f(v,u) = -f(u,v)$
 - **Întoarce** $|f|$

Complexitate?

Algoritmul Edmonds – Karp (2)

- $\text{Edmonds – Karp}(G, s, t)$

- **Pentru fiecare** (u,v) din E

- $f(u,v) = f(v,u) = 0$ // $O(E)$

- **Cât timp** // $O(E^*V)$ [vezi Cormen]

- Există căi reziduale între $s..t$ în G_f // $O(E)$

- Determină calea reziduală minimă p aplicând BFS // $O(E)$

- $c_f(p) = \min\{c_f(u,v) \mid (u,v) \text{ din } p\}$ // $O(E)$

- **Pentru fiecare** (u,v) din p // $O(E)$

- $f(u,v) = f(u,v) + c_f(p)$

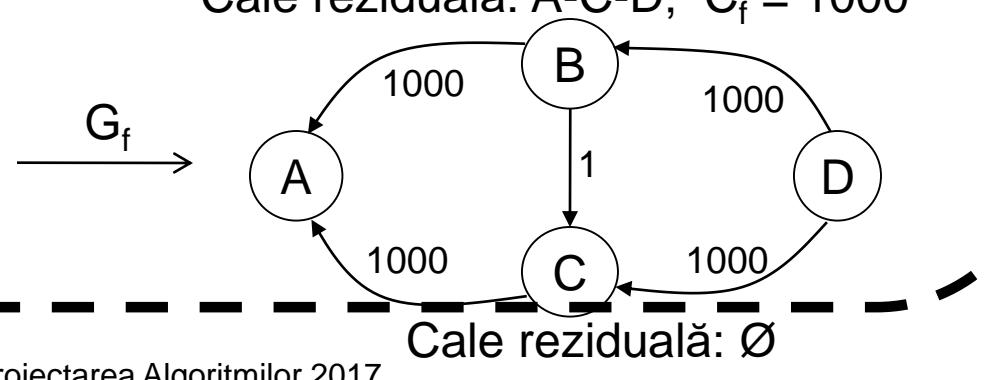
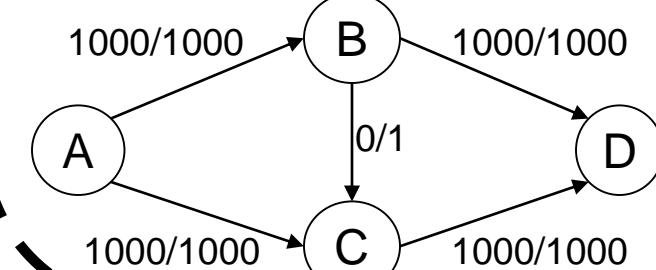
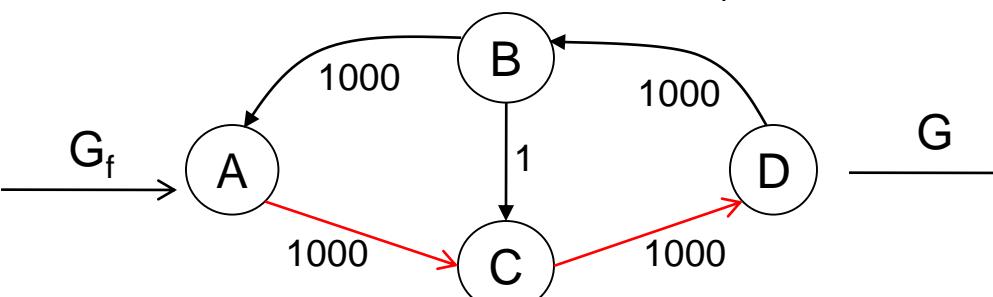
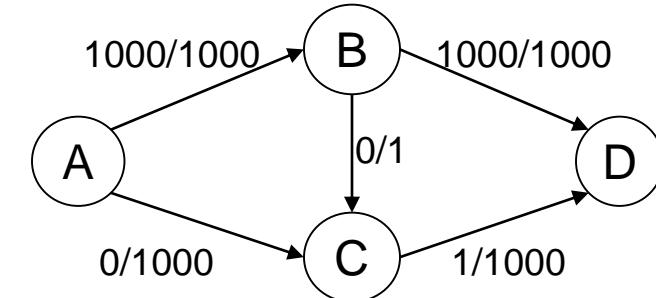
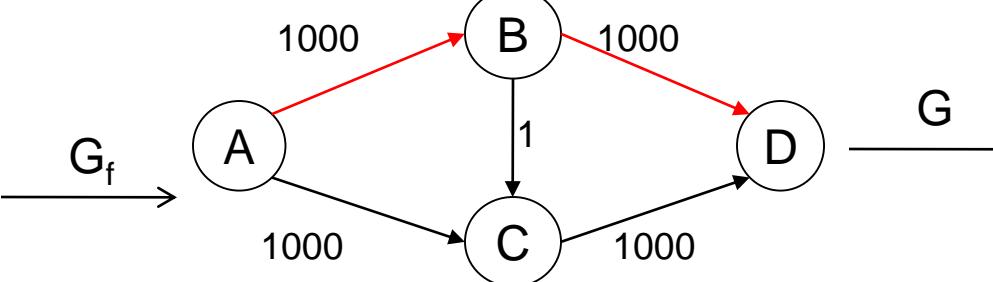
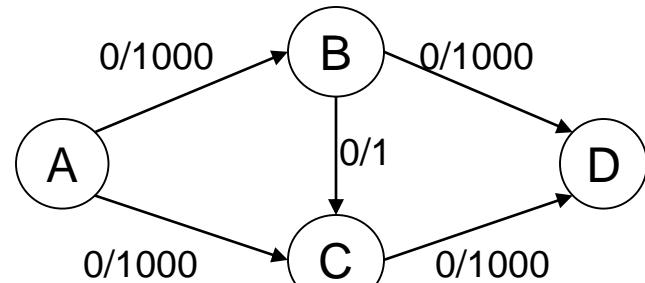
- $f(v,u) = -f(u,v)$

- **Întoarce** $|f|$

De câte ori un arc poate fi critic în rețeaua reziduală? $O(V)$
Câte arce? $O(E)$

Complexitate?
 $O(E^2 * V)$

Exemplu Edmonds-Karp



Pompare preflux (1)

- **Idee:** Simularea curgerii lichidelor într-un sistem de conducte ce leagă noduri aflate la diverse înălțimi;
- **Sursa – înălțime maximă;**
- **Inițial** toate nodurile exceptând sursa sunt la **înălțime 0**;
- **Destinația** rămâne în permanentă la **înălțimea 0!**

Pompare preflux (2)

- Există un preflux inițial în rețea obținut prin încărcarea la capacitate maximă a tuturor conductelor ce pleacă din s;
- Excesul de flux dintr-un nod poate fi stocat într-un rezervor al nodului (**Notat $e(u)$**);
- Când un nod u are flux disponibil în rezervor și o conductă spre un alt nod v nu este încărcată complet \rightarrow înălțimea lui u este crescută pentru a permite curgerea din u în v .

Pompare preflux – Definiții (1)

- $G = (V, E)$ rețea de flux;
- Definiție: **Preflux** = $f: V \times V \rightarrow \mathbb{R}$ astfel încât să fie satisfăcute restricțiile:
 - $f(u, v) \leq c(u, v), \forall (u, v) \in E$ – respectarea capacitatei arcelor;
 - $f(u, v) = -f(v, u), \forall u, v \in V$ – simetria fluxului;
 - $\sum_{v \in V} f(v, u) \geq 0, \forall u \in V \setminus \{s\}$ – conservarea fluxului.
- Definiție: Supraîncărcare a unui nod:
 - $e(u) = f(V, u) \geq 0, \forall u \in V \setminus \{s\}$.

Pompare preflux – Definiții (2)

- **Definiție:** O funcție $h: V \rightarrow N$ este o **funcție de înălțime** dacă îndeplinește restricțiile:
 - $h(s) = |V|$ – fixă;
 - $h(t) = 0$ – fixă;
 - $h(u) \leq h(v) + 1$ pentru orice arc rezidual $(u,v) \in G_f$ – variabilă.
- **Lema 5.19:** G – rețea de flux, $h: V \rightarrow N$ este o funcție de înălțime. Dacă $\forall u, v \in V, h(u) > h(v) + 1$ atunci arcul (u,v) nu este arc rezidual.

Pompare preflux – Metode folosite

- **Pompare(u,v)** // pompează fluxul în exces ($e(u) > 0$)
// are loc doar dacă diferența de înălțime dintre u și v este 1
// ($h(u) = h(v) + 1$), altfel nu e arc rezidual și nu ne interesează
 - $d = \min(e(u), c_f(u,v))$; // cantitatea de flux pompată
 - $f(u,v) = f(u,v) + d$; // actualizare flux pe arcul (u,v)
 - $f(v,u) = -f(u,v)$; // respectarea simetriei
 - $e(u) = e(u) - d$; // actualizare supraîncărcare la sursă
 - $e(v) = e(v) + d$; // actualizare supraîncărcare la destinație
- **Înălțare(u)** // mărește $h(u)$ dacă u are flux în exces
// ($e(u) > 0$) și $u \notin \{s, t\}$ $\forall (u,v) \in G_f$ avem $h(u) \leq h(v)$
 - $h(u) = 1 + \min\{h(v) \mid (u,v) \in G_f\}$

Complexitate?

Pompare preflux – Inițializare

- **Init_preflux(G, s, t)**
 - **Pentru fiecare** ($u \in V$)
 - $e(u) = 0$ // inițializare exces flux în nodul u
 - $h(u) = 0$ // inițializare înălțime nod u
 - **Pentru fiecare** ($v \in V$) // inițializare fluxuri
 - $f(u,v) = 0$
 - $f(v,u) = 0$
 - $h(s) = |V|$ // inițializare înălțime sursă
 - **Pentru fiecare** ($u \in \text{succs}(s) \setminus \{s\}$)
// actualizare flux + exces
 - $f(s,u) = c(s,u);$
 - $f(u,s) = -c(s,u);$
 - $e(u) = c(s,u)$

Complexitate?

Pompare preflux – Algoritm

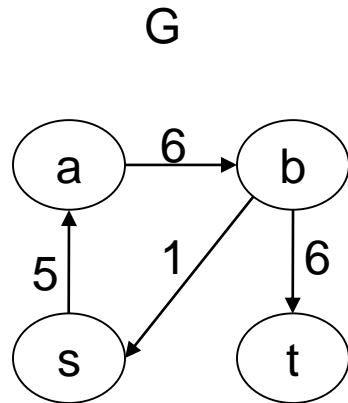
- Pompare_prelux(G, s, t)
 - Init_prelux(G, s, t) // initializarea prefluxului
 - **Cât timp** (1) // cât timp pot face pompări sau înălțări
 - **Dacă** ($\exists u \in V \setminus \{s, t\}, v \in V \mid e(u) > 0$ și $c_f(u,v) > 0$ și $h(u) = h(v) + 1$) // încerc să pompeze
 - Pompare(u,v); **continuă**;
 - **Dacă** ($\exists u \in V \setminus \{s, t\}, v \in V \mid e(u) > 0$ și $\forall (u,v) \in E_f, h(u) \leq h(v)$)
 - Înălțare(u); **continuă**; // încerc să înalță;
 - **Întrerupe**; // nu mai pot face nimic → am ajuns la flux max
 - **Întoarce** $e(t)$ // $e(t) = |f| =$ fluxul total în rețea

● Complexitate?

Pompare preflux – Complexitate

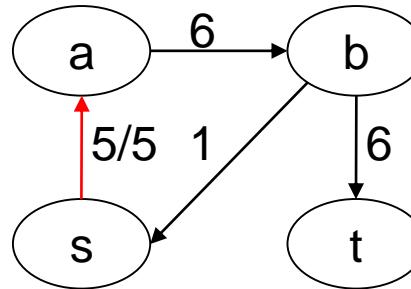
- Init_preflux: $O(V * E)$
- Pompare(u,v): $O(1)$
- Înălțare(u): $O(V)$ – implică găsirea minimului dintre nodurile succesoare
- Cât timp: [vezi Cormen]
 - Câte înălțări?
 - Care e înălțimea maximă? $2 |V| - 1$ – drum rezidual de lungime maximă
 - Care este numărul maxim total de înălțări? $(2 |V| - 1) (|V| - 2)$
 - Câte pompări?
 - Pompări saturate: $2 |V| |E|$ - de câte ori un arc poate fi saturat? (în funcție de suma $h(u) + h(v)$)
 - Pompări nesaturate: $4 |V|^2 (|V| + |E|)$ – sumă înălțimi noduri excedentare
- Complexitate totală: $O(V^2 * E)$ [vezi Cormen]

Exemplu Pompare preflux (1)

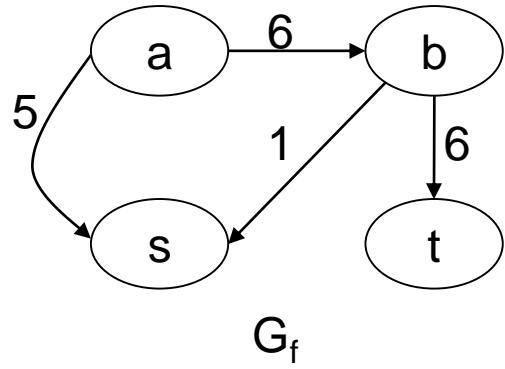


Init_preflux

$$\begin{aligned} h(s) &= 4 \\ h(a) &= h(b) = h(t) = 0 \\ e(a) &= 5 \\ e(s) &= e(b) = e(t) = 0 \end{aligned}$$



G_f



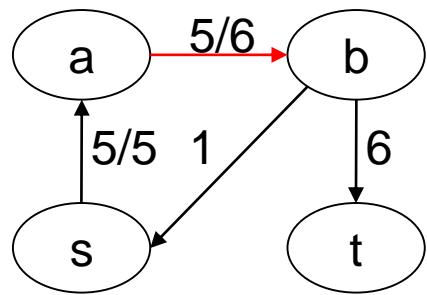
$\xrightarrow{\text{Înălțare}} (a)$

$$\begin{aligned} h(s) &= 4 \\ h(a) &= 1 \\ h(b) &= h(t) = 0 \\ e(a) &= 5 \\ e(s) &= e(b) = e(t) = 0 \end{aligned}$$

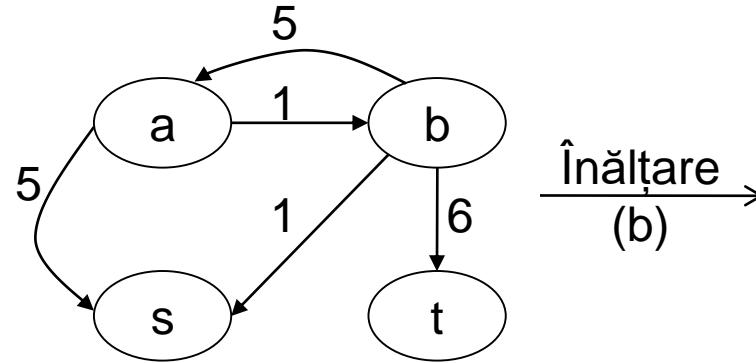
$\xrightarrow{\text{Pompare}} (a,b)$

Exemplu Pompare preflux (2)

$h(s) = 4$
 $h(a) = 1$
 $h(b) = h(t) = 0$
 $e(b) = 5$
 $e(s) = e(a) = e(t) = 0$

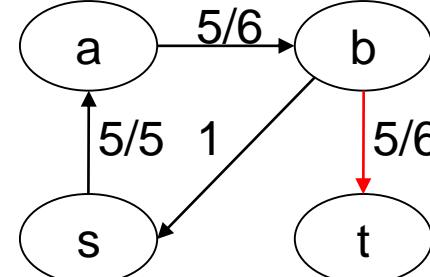


$G_f \rightarrow$



$h(s) = 4$
 $h(a) = h(b) = 1$
 $h(t) = 0$
 $e(b) = 5$
 $e(s) = e(a) = e(t) = 0$

$\xrightarrow{\text{Pompare (b,t)}}$



$h(s) = 4$
 $h(a) = h(b) = 1$
 $h(t) = 0$
 $e(t) = 5$
 $e(s) = e(a) = e(b) = 0$

ÎNTREBĂRI?

Proiectarea Algoritmilor

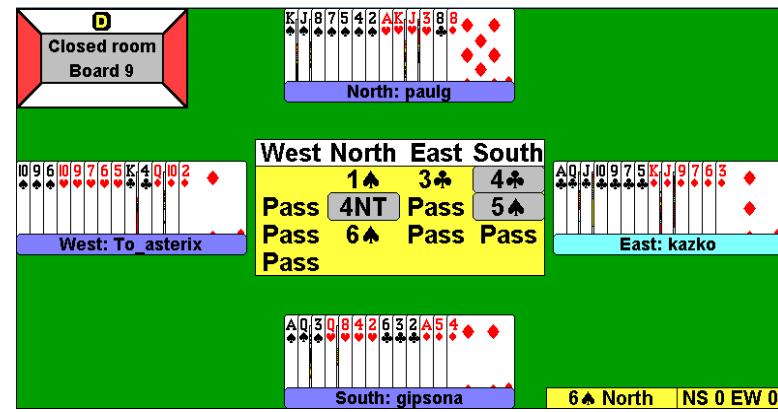
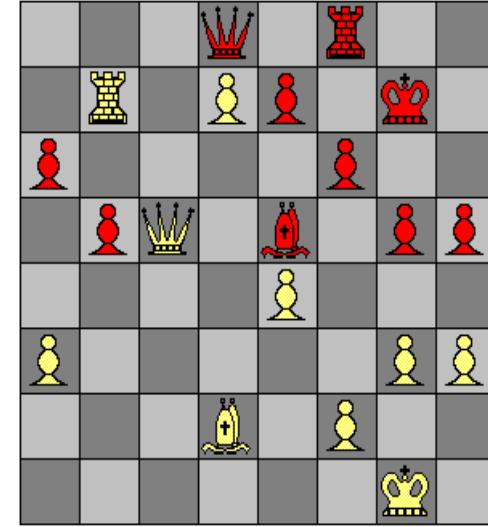
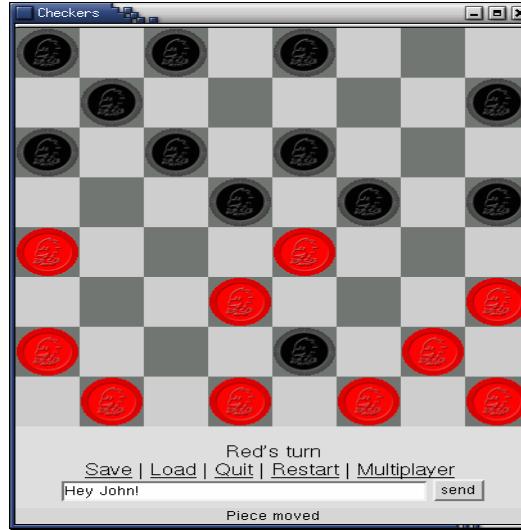
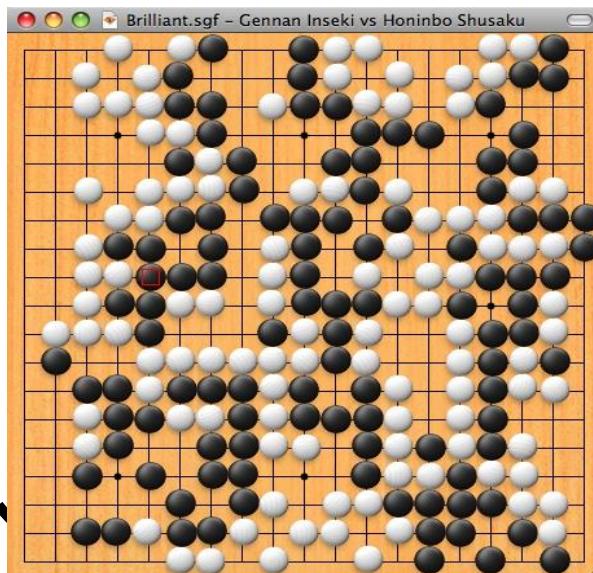
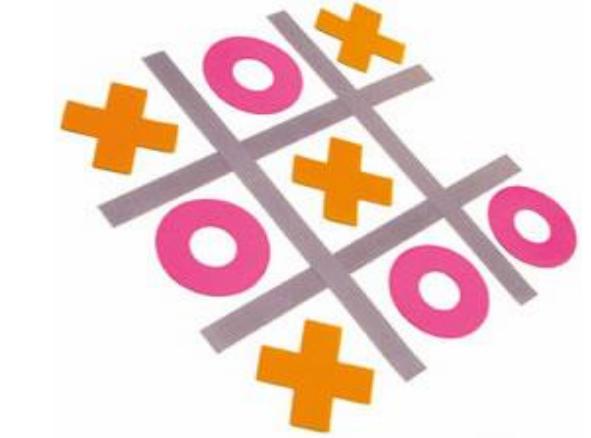
Curs 12 – Algoritmi pentru jocuri

Minimax, α - β

Bibliografie

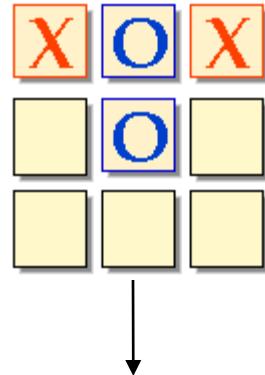
- Giumale – Introducere in Analiza Algoritmilor cap 7.6
- <http://www.dwheeler.com/chess-openings/#Sicilian%20Defense>
- http://mouserunner.com/MozllaTicTacToe/Mozilla_Tic_Tac_Toé.htm
- http://www.emunix.emich.edu/~evett/AI/AlphaBeta_movie/index_movie.htm

Problema

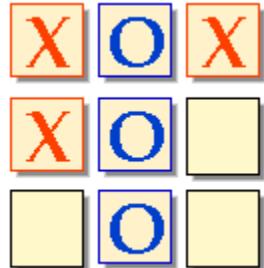


PA

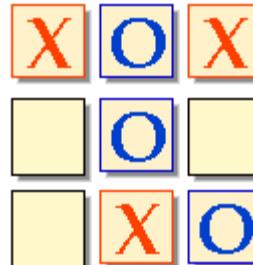
Cum gândim noi?



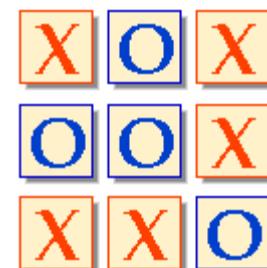
Analizăm posibilitățile și evaluăm fiecare mutare în funcție de consecințe.



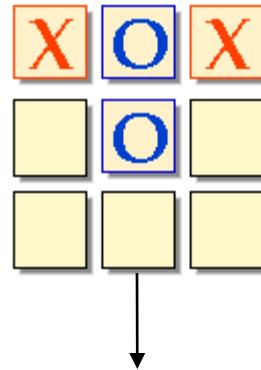
Am pierdut! :(



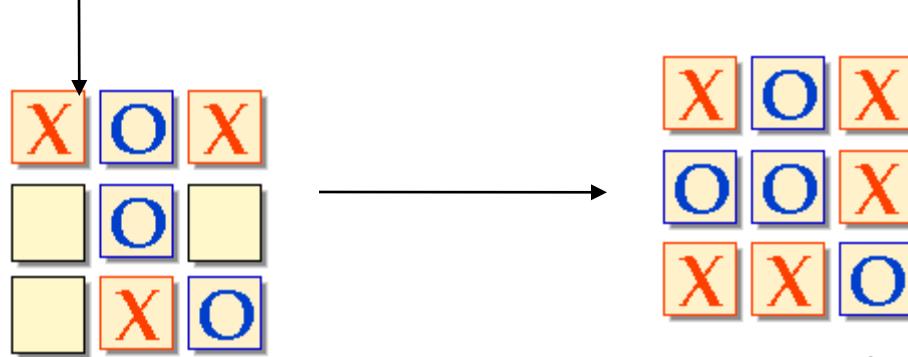
Egal! :(



Cum gândim noi?



Ne dăm seama "instinctiv" că avem o singură opțiune pentru a nu pierde partida și mutăm în consecință!

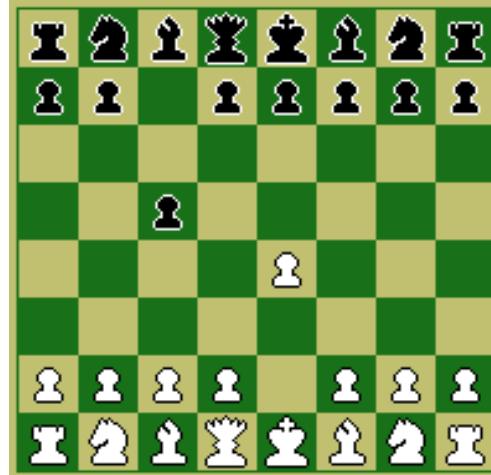


Egal! :(

Cum gândim noi?

[http://www.dwheeler.com/
/chess-openings/
#Sicilian%20Defense](http://www.dwheeler.com/chess-openings/#Sicilian%20Defense)

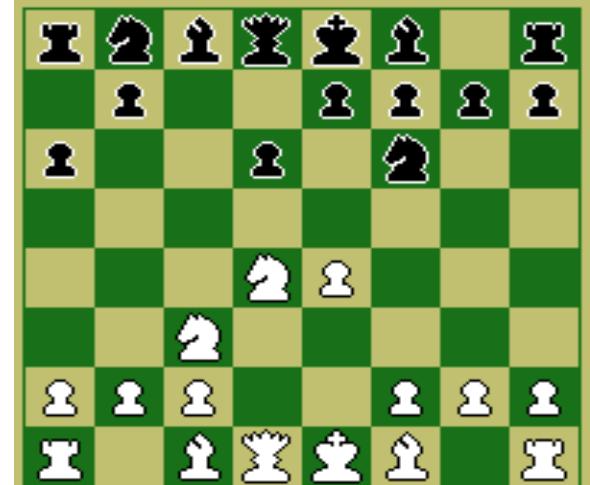
Varianta Najdorf



Apărarea siciliană!

Varianta Dragon

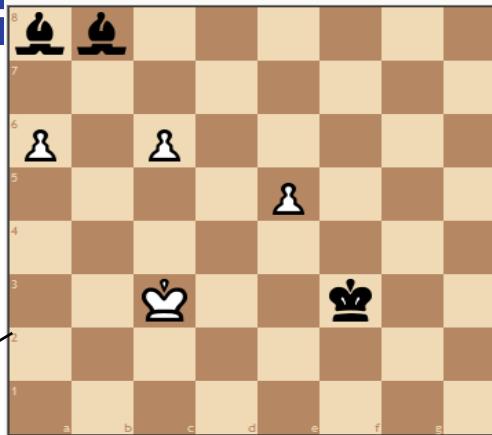
Când avem foarte
multe posibilități la
dispoziție încercăm
să folosim **poziții**
(pattern-uri)
cunoscute.



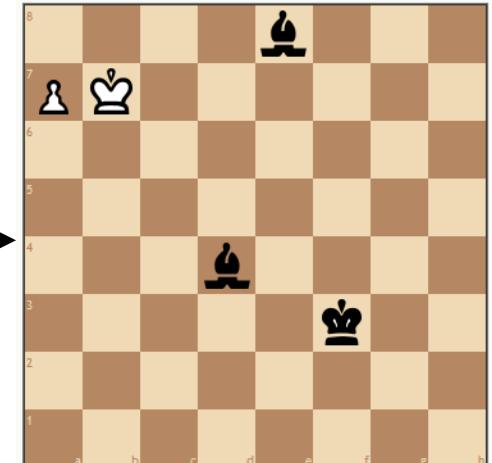
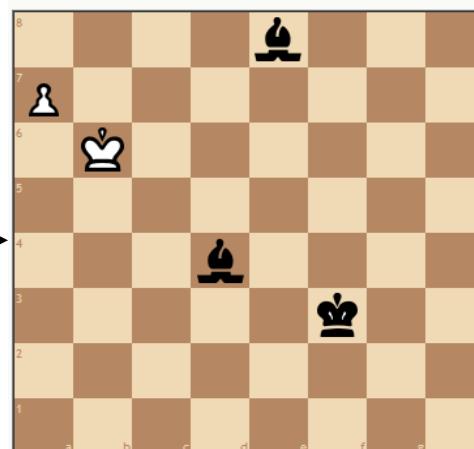
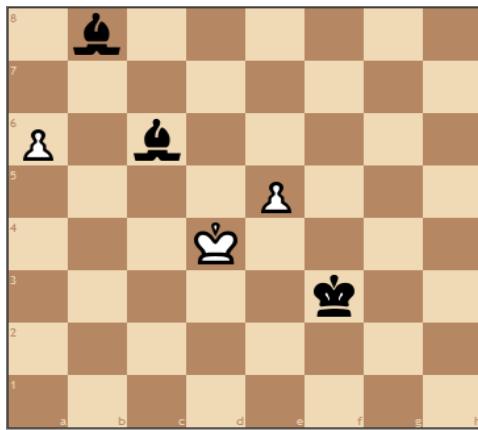
Cum gândim noi?

Albul la mutare

- 11 posibilități de mutare;
- le putem încerca pe toate
- să vedem ce se întâmplă.



Circa 15.000 de mutări de analizat – ușor pentru calculator. Noi eliminăm mutările ce ni se par fără sens (mai mult de jumătate).



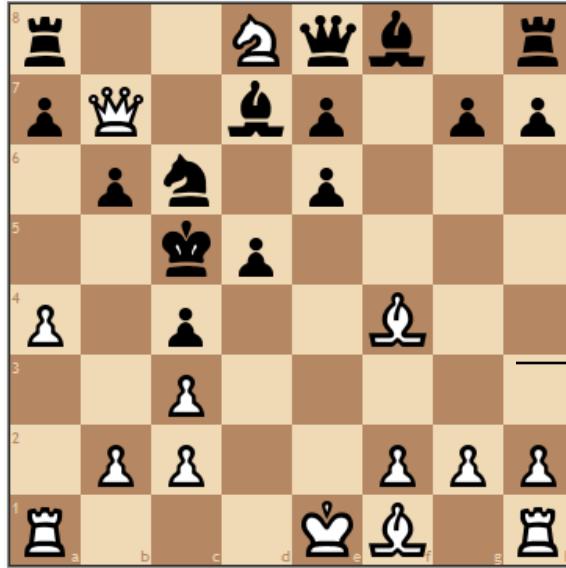
Numărul mutărilor posibile se reduce la 6.

Doar 4 mutări posibile.

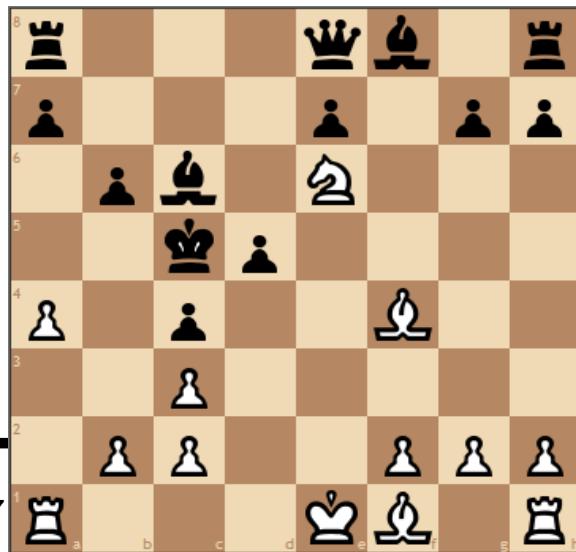
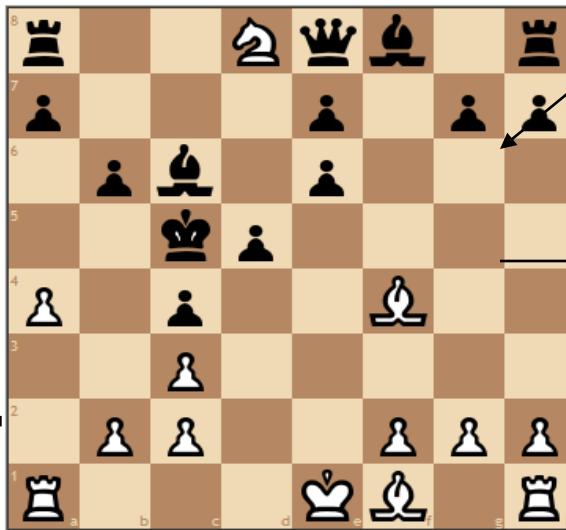
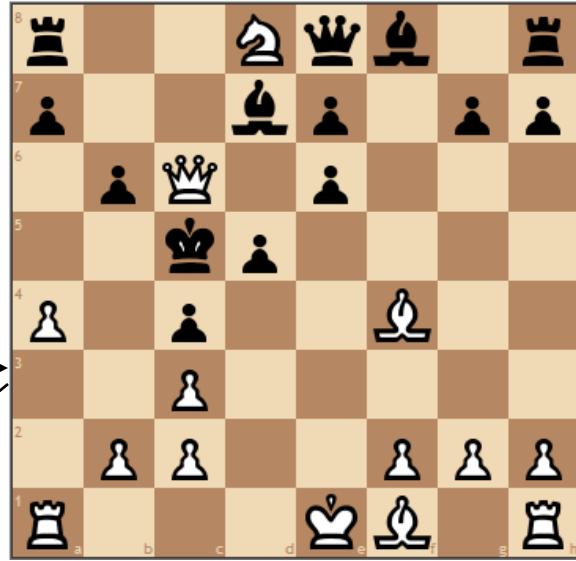
Remiză asigurată!

Cum gândim noi?

Cam 35
de
mutări
posibile



Varianta câștigătoare presupune sacrificarea unei piese valoroase:



Cum gândim noi?

- **Evaluăm amenințările:**
 - Căutăm mutări care să minimizeze pierderile;
 - Căutăm mutări care să maximizeze câștigul.
- **Alegem mutările ce ni se "par" cele mai bune pe moment:**
 - Explorăm în adâncime **graful mutărilor**;
 - **Numărul de niveluri** = **minim** dintre:
 - Terminarea jocului;
 - Obținerea unui avantaj consistent fără pericol aparent de a-l pierde;
 - Nivelul maxim al **capacității noastre de calcul**.

Abordări posibile pentru calculator

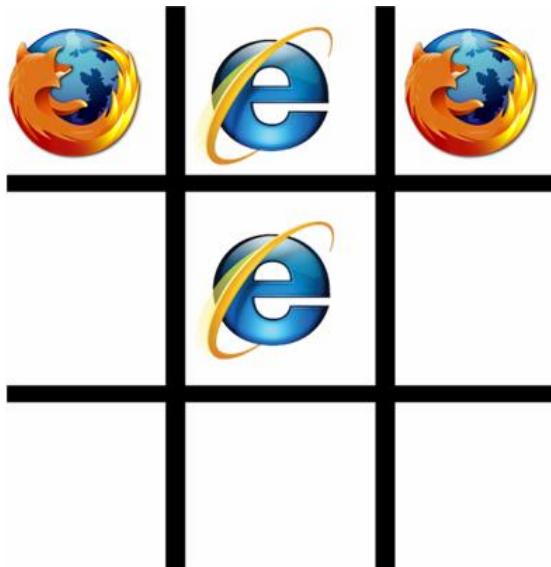
- **Şabloane** pentru poziţii standard.
- **Căutare** în spaţiul de poziţii.
- **Utilizare euristici** pentru evaluarea poziţiei curente.
- Ne vom concentra asupra căutărilor.

Metoda Minimax

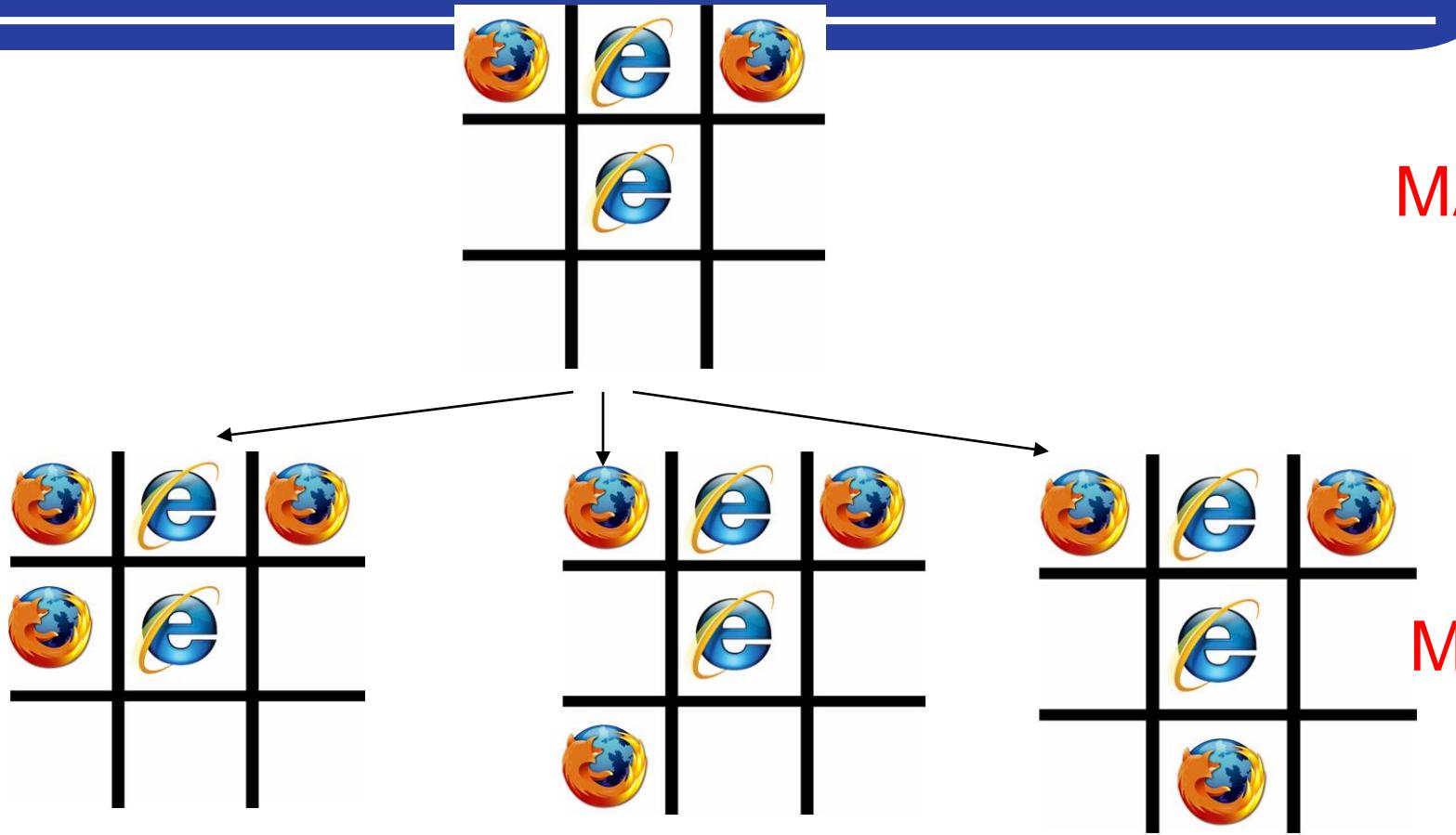
- 2 jucători: Max și Min care mută pe rând (Max mută primul).
- Max urmărește să-și maximizeze câștigul.
- Min urmărește să-și minimizeze pierderea.
- Se construiește un arbore AND-OR:
- Nivelurile impare → mutările jucătorului Max.
- Nivelurile pare → mutările jucătorului Min.
- Frunzele desemnează câștigul/pierderea lui Max.
- Arcele reprezintă mutările propriu-zise.

Exemplu (I)

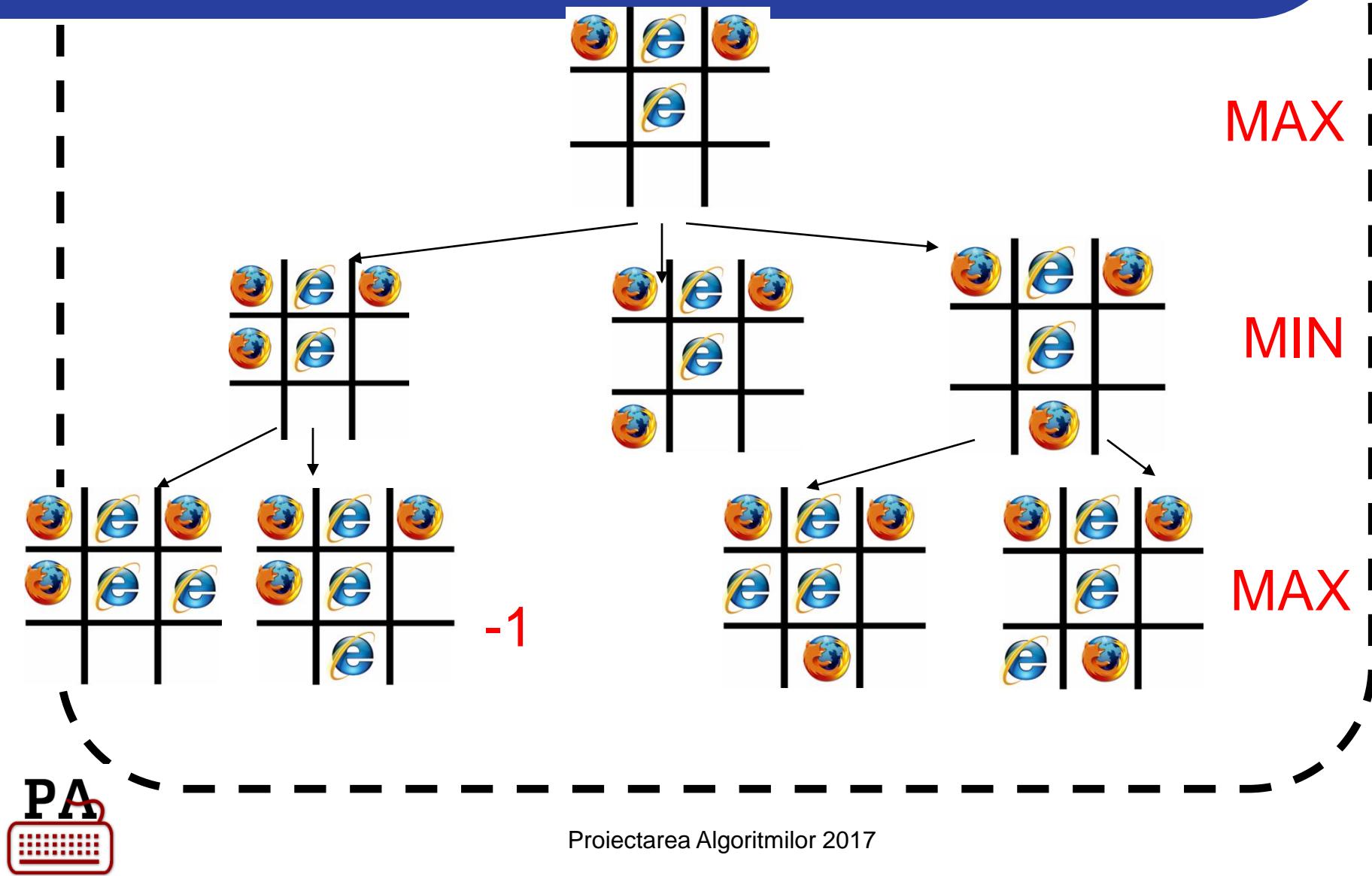
MAX (Firefox) trebuie să mute:



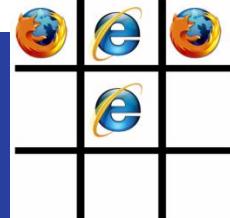
Exemplu (II)



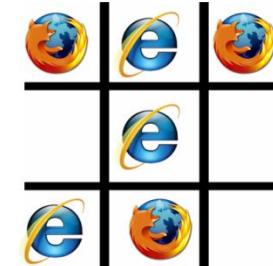
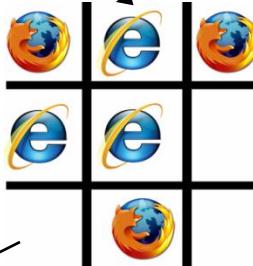
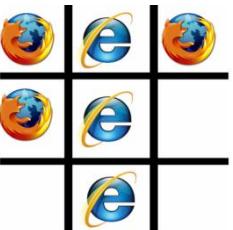
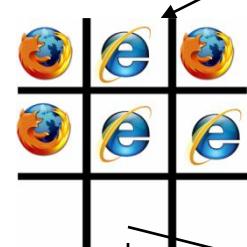
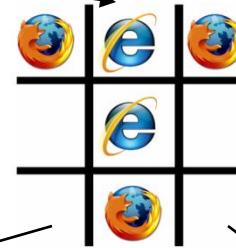
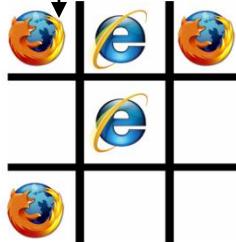
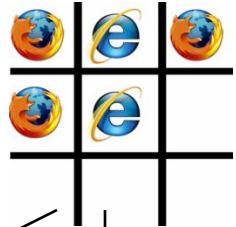
Exemplu (III)



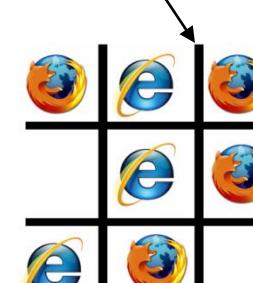
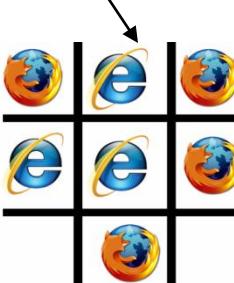
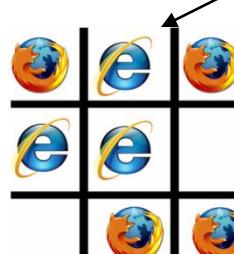
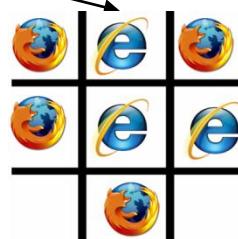
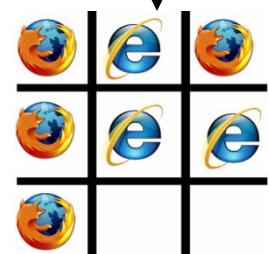
Exemplu (IV)



MAX



MAX



MIN

+1

0

... -1

... 0

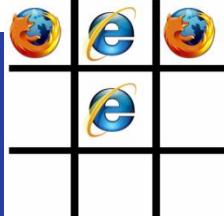
... 0

http://mouserunner.com/MozillaTicTacToe/Mozilla_Tic_Tac_To.htm

Funcționare Minimax

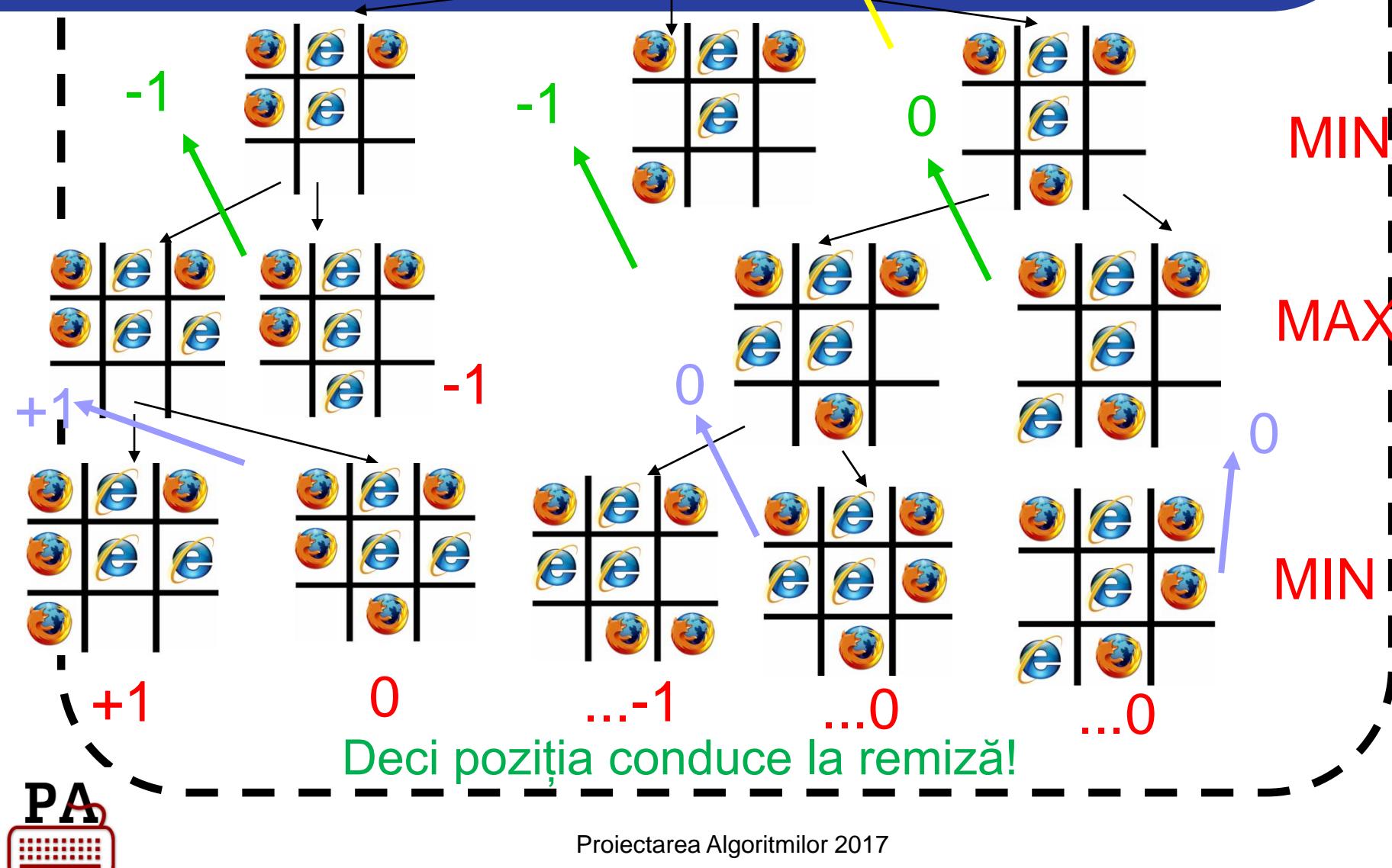
- 1) Se generează întregul arbore;
- 2) Se evaluatează frunzele și li se asociază valori;
- 3) Se propagă rezultatele dinspre frunze spre rădăcină astfel:
 - Nivelul MIN alege cea mai mică valoare dintre cele ale copiilor.
 - Nivelul MAX alege cea mai mare valoare dintre cele ale copiilor.

Exemplu (V)

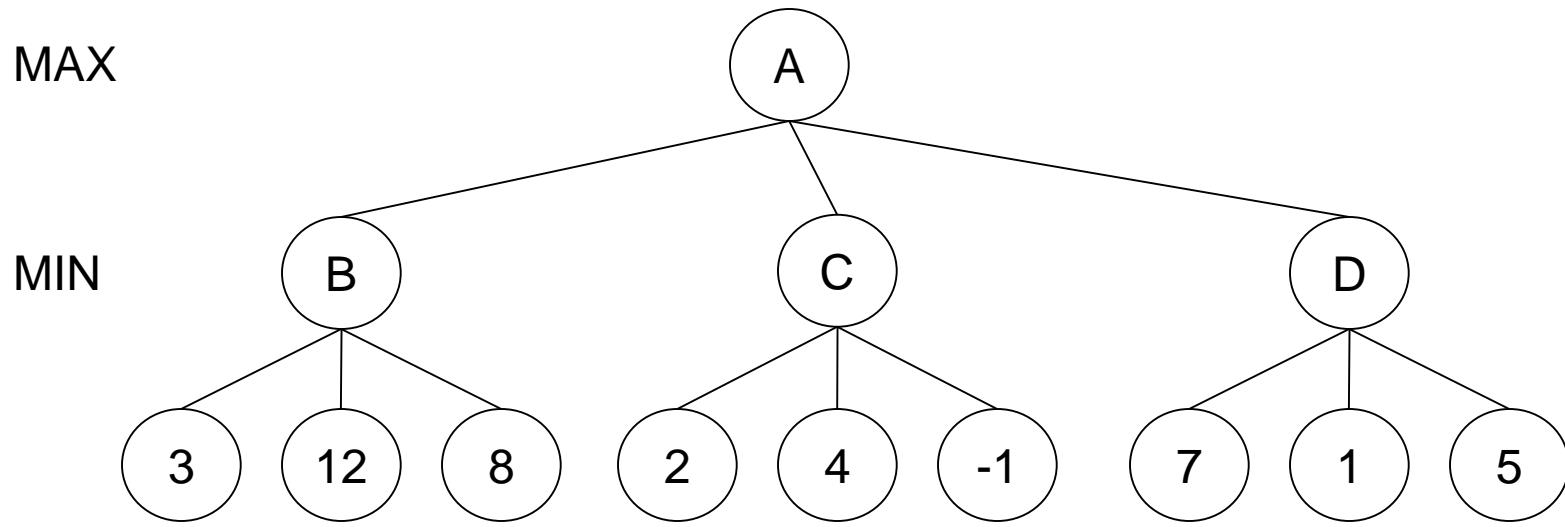


0

MAX



Alt exemplu (I)



Probleme

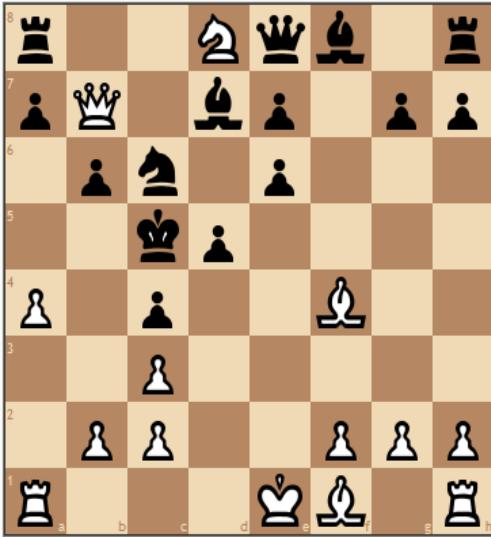
- Dimensiunea arborelui pentru “X și 0” e $\leq 9!$
- Pentru Șah fiecare nod are în medie 35 copii!
- Pentru Go ramificarea este de cca. 150 – 250!
- Complexitatea arborelui:
 - pentru Șah – 10^{123} noduri;
 - pentru Go – 10^{360} noduri.
- Limitări: → Nu putem să construim întregul arbore → Nu putem ajunge de fiecare dată la stările finale pentru a le putea evalua.

Optimizări minimax

• Limitarea adâncimii căutării

- Trebuie să construim o **funcție euristică** care **să estimeze** şansele de câștig pentru o poziție dată.
 - Ex. pentru sah:
 - Regină: 10p; Turn: 5p; Cal, Nebun: 3p; Pion: 1p;
 - Ex: Funcție de evaluare a poziției = suma pieselor proprii – suma pieselor adversarului.
- **Oprirea căutării:**
 - Limitare **statică**: după un număr maxim de nivele/intervall de timp.
 - Limitare **dinamică**: când profitul obținut din continuarea căutării devine foarte mic (scade sub o valoare fixată).
- Se **estimează valoarea funcției de evaluare** la nivelul respectiv.
- Apoi **propagăm valorile** conform principiului enunțat anterior.

Exemplu și contraexemplu



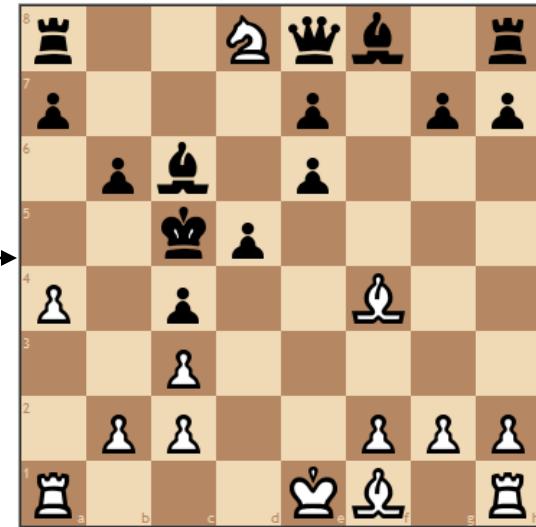
Eval: 36-37=-1

Funcția nu ține cont de poziție – albul are o poziție net superioară dar funcția de evaluare o ignoră



Eval: 36-34=2

Dacă căutarea se oprește la acest nivel atunci aparent albul iese în câștig material ignorându-se faptul că la mutarea următoare se pierde dama



Eval: 26-34=-8

În cazul în care căutarea se oprește la acest nivel aparent albul iese în dezavantaj deoarece a pierdut dama

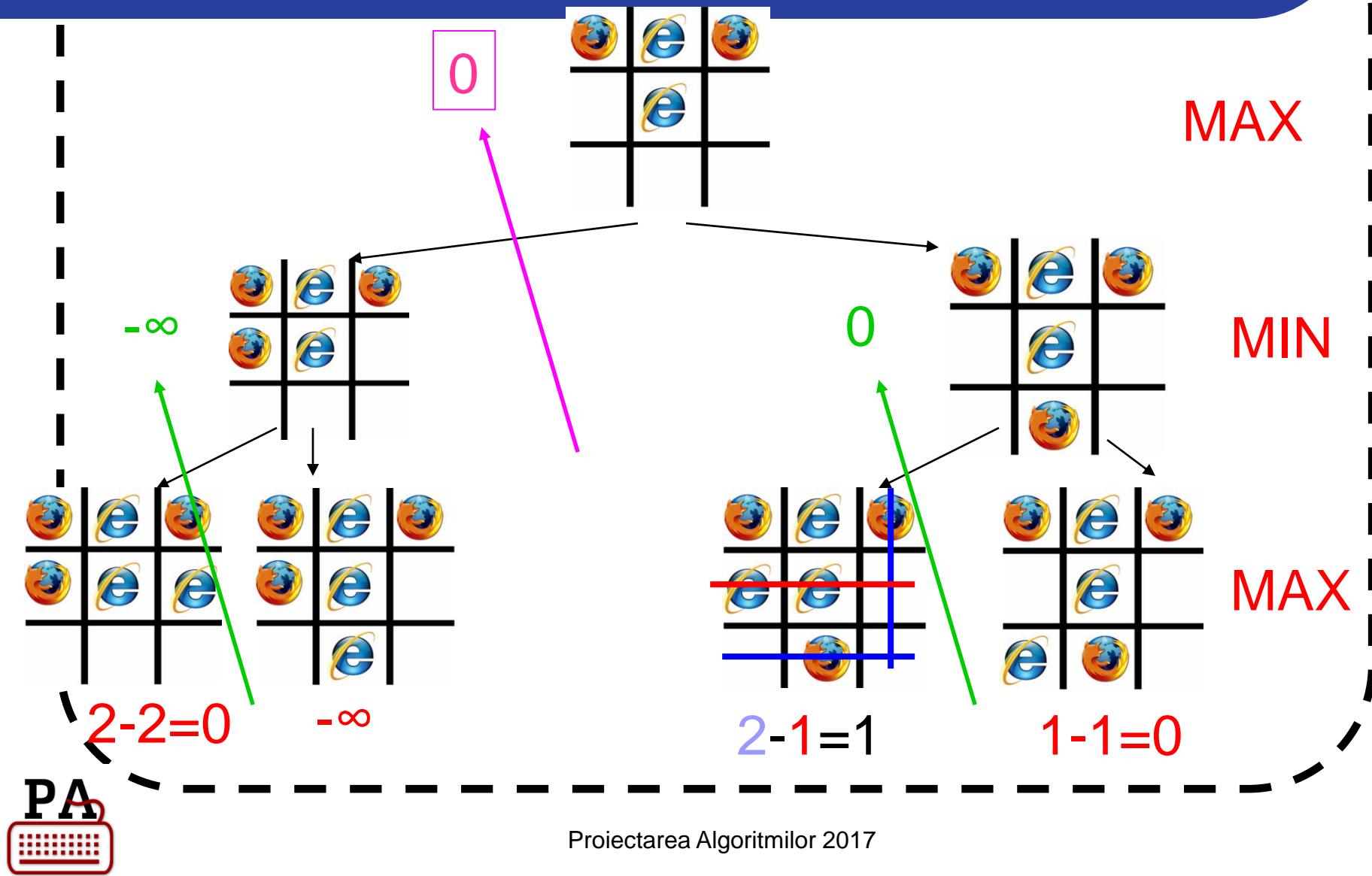
Minimax – funcții de evaluare

- Funcția euristică trebuie să **cuantifice** "poziția".
 - Chiar în dauna avantajului material.
- Trebuie să ia în calcul **potențialele amenințări!**

Exemplu funcție euristică X și 0

- F = numărul de linii/coloane/diagonale posibil câștigătoare pentru MAX – numărul de linii/coloane/diagonale posibil câștigătoare pentru MIN.
- Dacă MAX poate să mute și să câștige atunci $F = +\infty$; dacă MIN poate să mute și să câștige $F = -\infty$.

Exemplu funcție euristică X și 0



Algoritm MINIMAX

- **MINIMAX_limitat (n, nivel_limită)**
 - **Pentru fiecare** $n' \in \text{succs}(n)$ // pentru toate mutările
 - Fie $m = \text{mutarea corespunzătoare arcului } (n, n')$
 - $\text{VAL}(m) = w(n', \text{nivel_limită}, 1)$ // determin valoarea mutării
 - **Întoarce** m a.î. $\text{VAL}(m) = \max \{\text{VAL}(x) | x \in \text{mutări}(n)\}$
- **W(n, limită, nivel)**
 - **Dacă** n este frunză **Întoarce** $\text{cost}(n)$
 - **Dacă** $\text{nivel} \geq \text{limită}$ **Întoarce** $\text{euristică}(n)$
 - **Dacă** jucătorul MAX este la mutare **Întoarce**
 - $\max \{w(n', \text{limită}, \text{nivel} + 1) | n' \in \text{succs}(n)\}$
 - **Dacă** jucătorul MIN este la mutare **Întoarce**
 - $\min \{w(n', \text{limită}, \text{nivel} + 1) | n' \in \text{succs}(n)\}$

Caz special - Minimax 3 jucători (1)

Jucătorii vor alege pe rând valoarea care le maximizează câștigul propriu

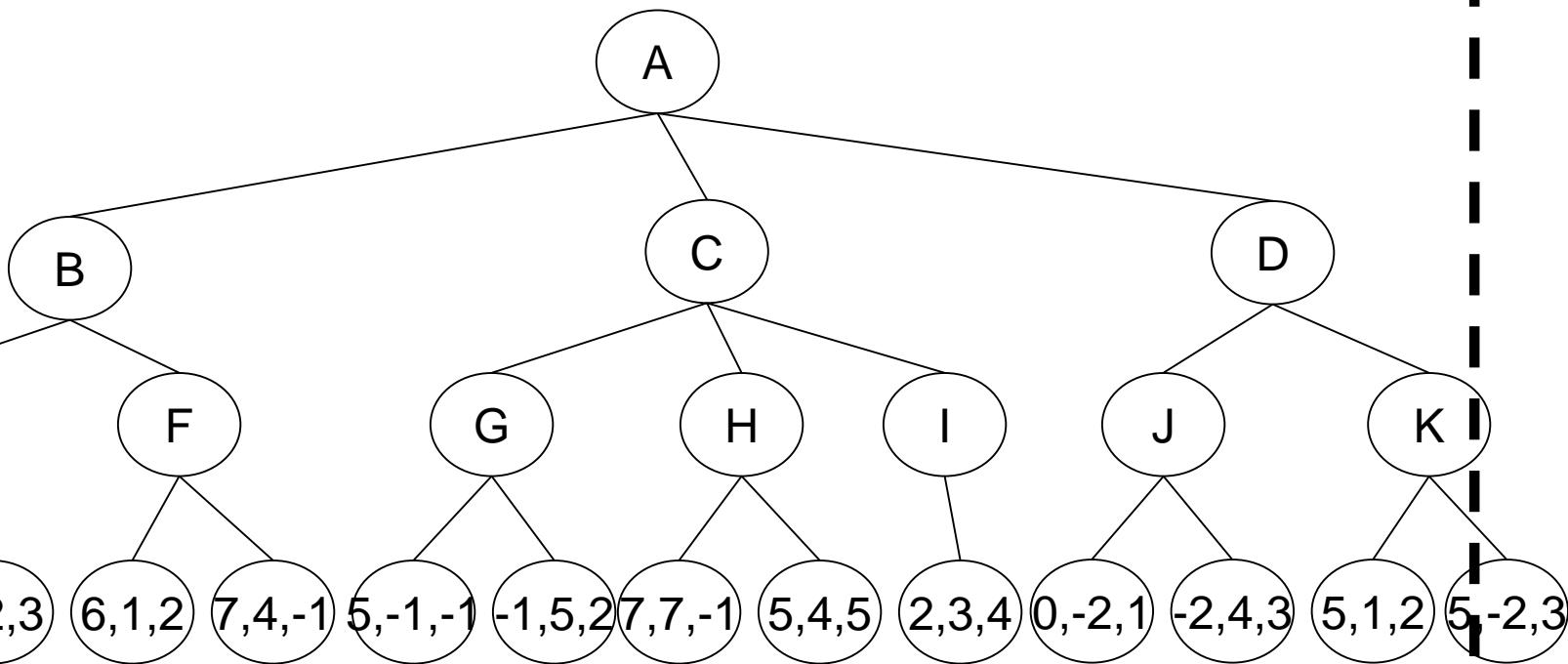
Juc 1

Juc 2

Juc 3

Juc 1

PA



Caz special - Minimax 3 jucători (2)

Jucătorii vor alege pe rând valoarea care le maximizează câștigul propriu

Juc 1

Juc 2

Juc 3

Juc 1

A

1,2,6

-1,5,2

1,4,3

1,2,6

6,1,2

5,4,5

5,-2,3

1,2,6

4,2,3

6,1,2

7,4,-1

5,-1,-1

-1,5,2

-1,5,2

7,7,-1

5,4,5

2,3,4

1,4,3

0,-2,1

1,4,3

5,1,2

5,-2,3

PA

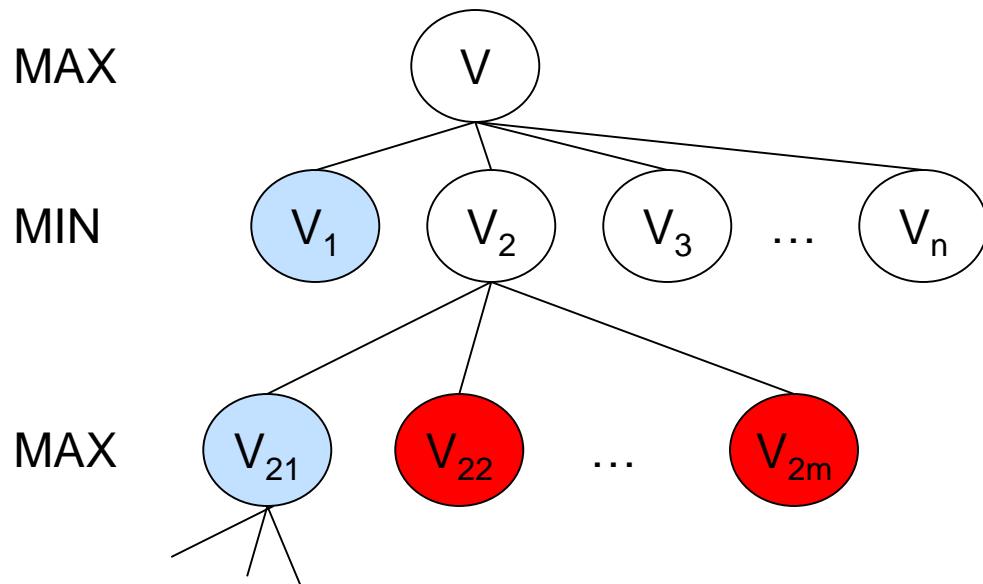
Caz special (2) – Minimax Probabilistic

- La unele jocuri, mutările sunt guvernate de șansă.
- Ex: Jocul de Table – mulțimea mutărilor este limitată de:
 - starea curentă a jocului;
 - combinația zarurilor în starea curentă.
- Arborele MINIMAX este **completat cu noduri suplimentare (noduri șansă)** plasate între nodurile MIN/MAX (MIN – șansă – MAX și MAX – șansă – MIN).
- Valorile se calculează ca sumă ponderată între probabilitatea nodului și evaluarea acestuia (prin cost sau euristică).

Tăiere α - β

- Încercăm să limităm spațiul de căutare prin eliminarea variantelor ce nu au cum să fie alese.

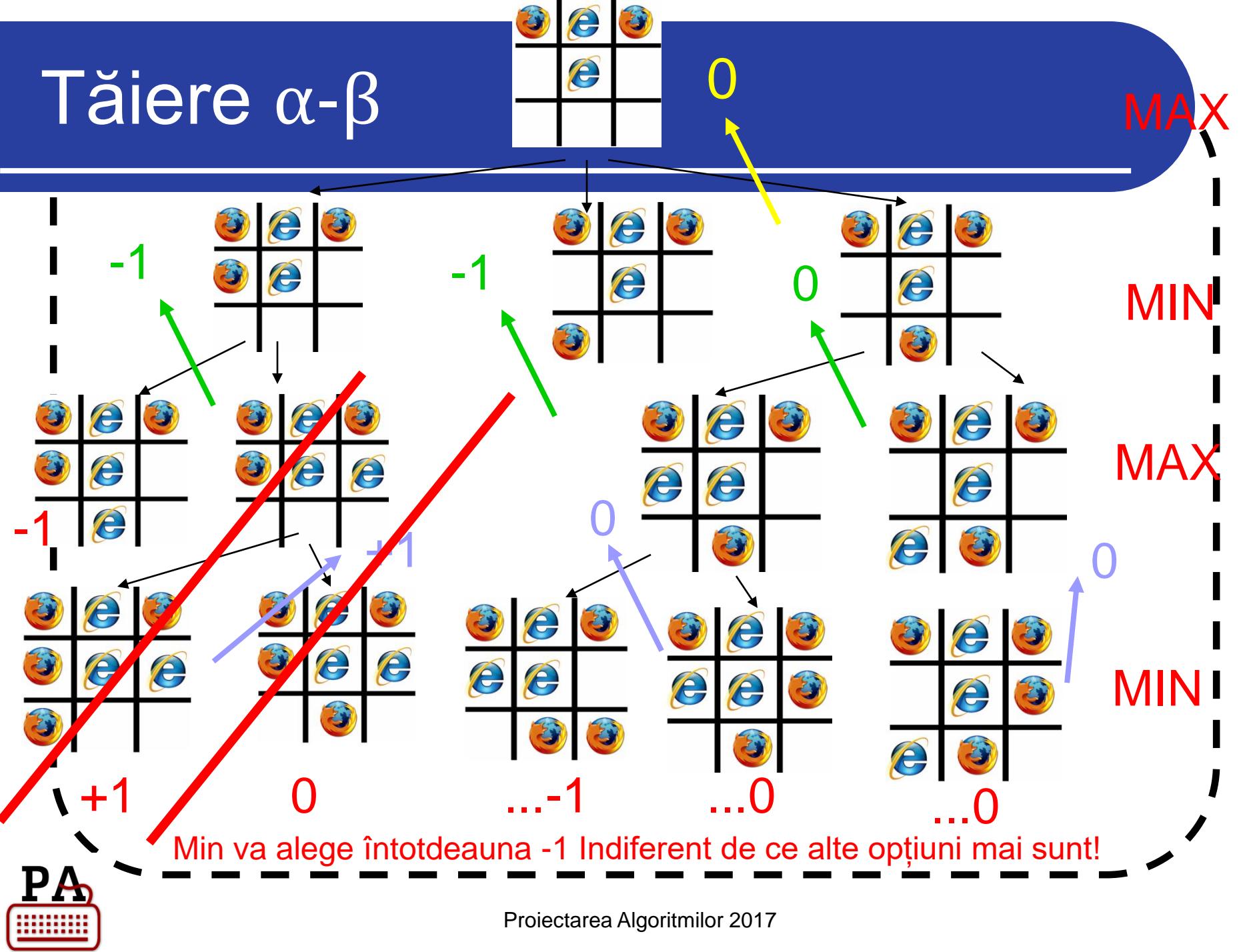
- Idee:
 - Dacă $V_{21} < V_1$ toată ramura V_2 poate fi ignorată.



Tăiere α - β

- α = max dintre valorile găsite pentru un nod MAX.
- β = min dintre valorile găsite pentru un nod MIN.
- Tăiem o ramură dacă:
 - am găsit un nod pe nivelul MAX cu valoare $\beta \leq$ oricare din valorile α calculate anterior;
 - am găsit un nod pe nivelul MIN cu valoare $\alpha \geq$ oricare din valorile β calculate anterior.
- Teorema α - β . Fie J un nod din arborele MINIMAX explorat. Daca $\alpha(J) \geq \beta(J)$, atunci explorarea nodului J nu este necesară.

Tăiere α - β



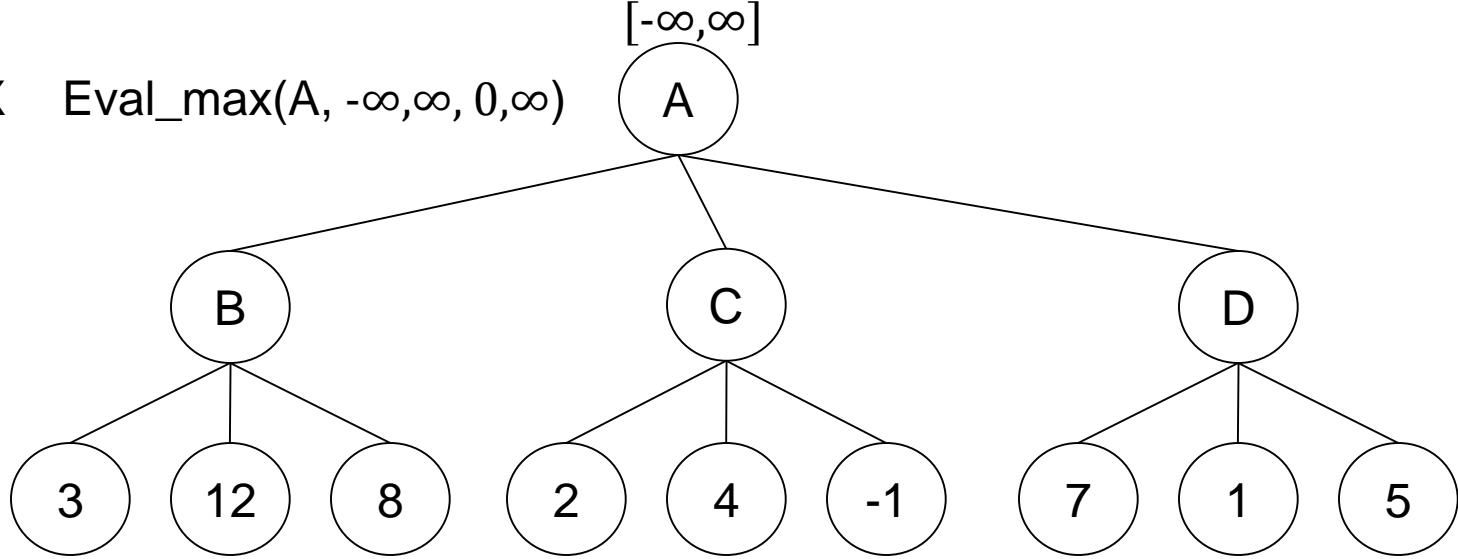
Algoritm α - β

- α - β (n, limită)
 - w = eval_max(n, $-\infty$, ∞ , 0, limită)
 - Întoarce m \in mutări(n) a.i. VAL(m) = w
- eval_max(n, α , β , nivel, limită)
 - Dacă n este frunză Întoarce cost(n)
 - Dacă (nivel \geq limită) Întoarce euristică(n) // sunt limitat
 - a = $-\infty$ // valoarea curentă a nodului de tip Max
 - Pentru fiecare ($n' \in \text{succs}(n)$)
 - a = max(a, eval_min(n' , max(α , a), β , nivel+1, limită)); // propag
 - Dacă ($a \geq \beta$) oprire;
 - Întoarce a
- similar eval_min

Alt exemplu (II)

MAX Eval_max(A, $-\infty, \infty$, 0, ∞)

MIN



eval_max($n, \alpha, \beta, \text{nivel}, \text{limită}$)

Dacă n este frunză **Întoarce** cost(n)

Dacă ($\text{nivel} \geq \text{limită}$) **Întoarce** euristică(n) // sunt limitat

$a = -\infty$ // valoarea curentă a nodului de tip max

Pentru fiecare ($n' \in \text{succs}(n)$) {

$a = \max(a, \text{eval_min}(n', \max(\alpha, a), \beta, \text{nivel}+1, \text{limită}))$; // propag

Dacă ($a \geq \beta$) **oprire**; }

Întoarce a

Alt exemplu (III)

MAX $\text{Eval_max}(A, -\infty, \infty, 0, \infty)$

$[-\infty, \infty]$

A

$a = -\infty$

MIN $[-\infty, \infty]$

$\text{Eval_min}(B, -\infty, \infty, 1, \infty)$

B

C

D

3

12

8

2

4

-1

7

1

5

$\text{eval_max}(n, \alpha, \beta, \text{nivel}, \text{limită})$

Dacă n este frunză **Întoarce** $\text{cost}(n)$

Dacă $(\text{nivel} \geq \text{limită})$ **Întoarce** $\text{euristică}(n)$ // sunt limitat

$a = -\infty$ // valoarea curentă a nodului de tip max

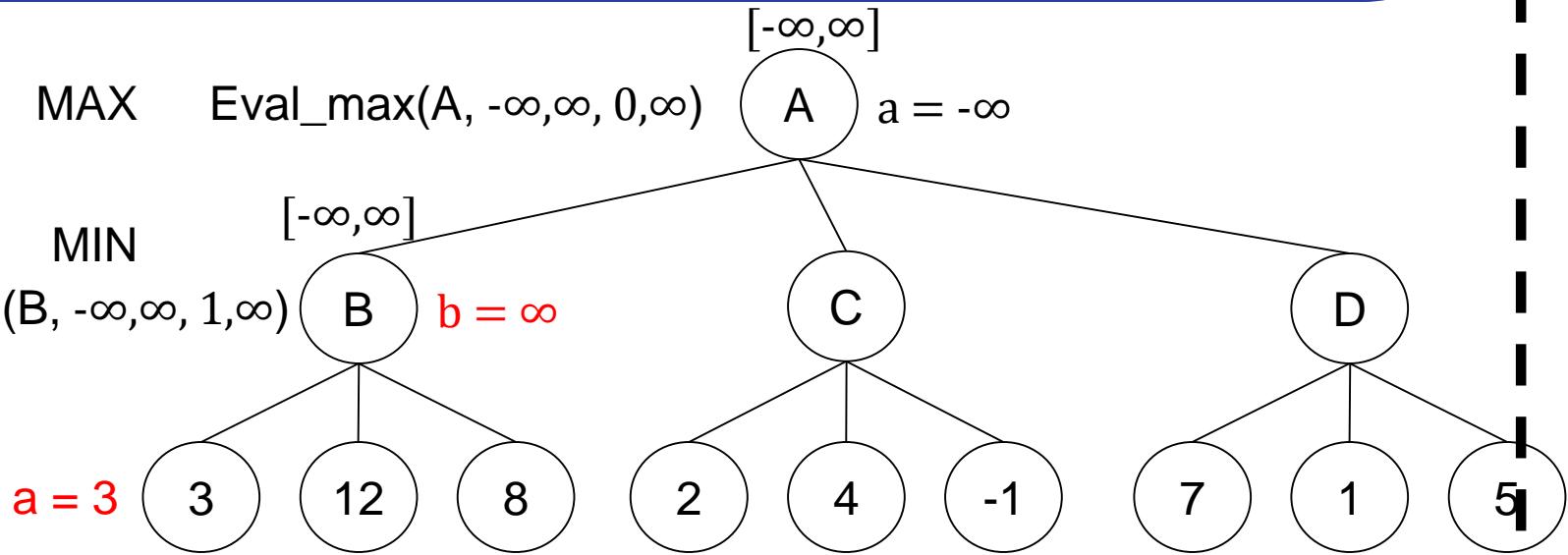
Pentru fiecare $(n' \in \text{succs}(n))$ {

$a = \max(a, \text{eval_min}(n', \max(\alpha, a), \beta, \text{nivel}+1, \text{limită}))$; // propag

Dacă $(a \geq \beta)$ **oprire**; }

Întoarce a

Alt exemplu (IV)



`eval_min(n, α, β, nivel, limită)`

Dacă n este frunză **Întoarce** $\text{cost}(n)$

Dacă $(\text{nivel} \geq \text{limită})$ **Întoarce** $\text{euristică}(n)$ // sunt limitat

$b = \infty$ // valoarea curentă a nodului de tip min

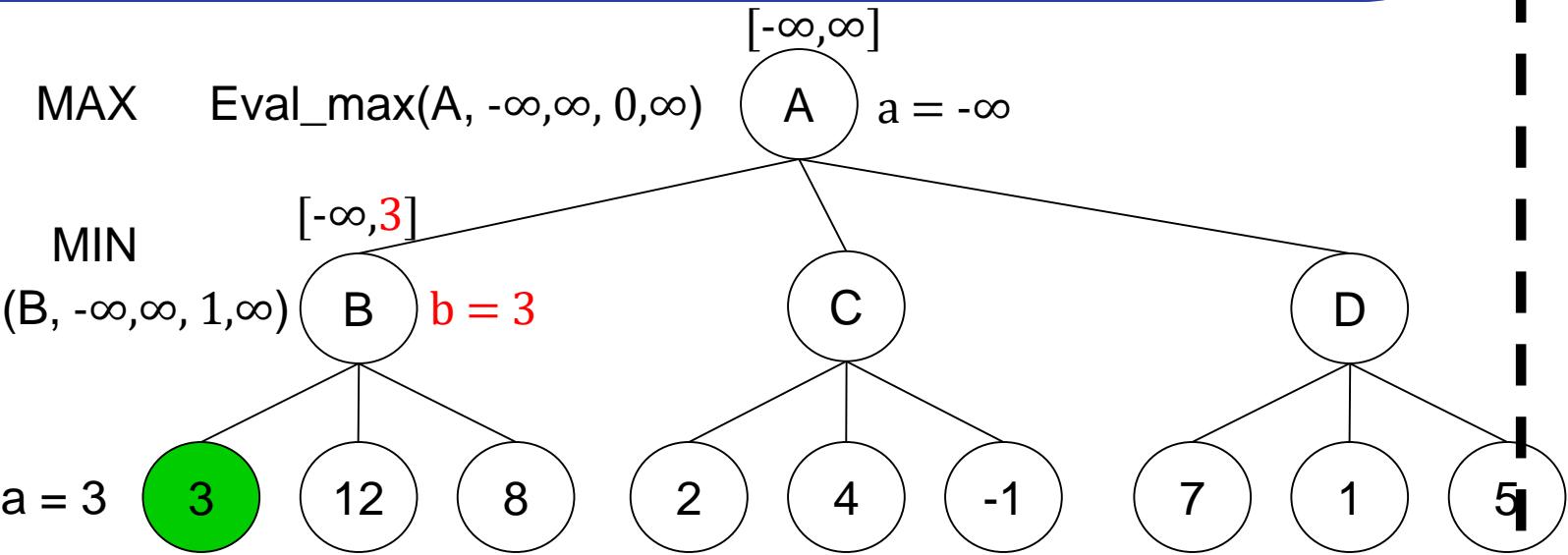
Pentru fiecare $(n' \in \text{succs}(n))$ {

$b = \min(b, \text{eval_max}(n', \alpha, \min(\beta, b), \text{nivel}+1, \text{limită}))$; // propag

Dacă $(b \leq \alpha)$ **oprire**; }

Întoarce b

Alt exemplu (V)



`eval_min(n, α, β, nivel, limită)`

Dacă n este frunză **Întoarce** cost(n)

Dacă (nivel ≥ limită) **Întoarce** euristică(n) // sunt limitat

$b = \infty$ // valoarea curentă a nodului de tip min

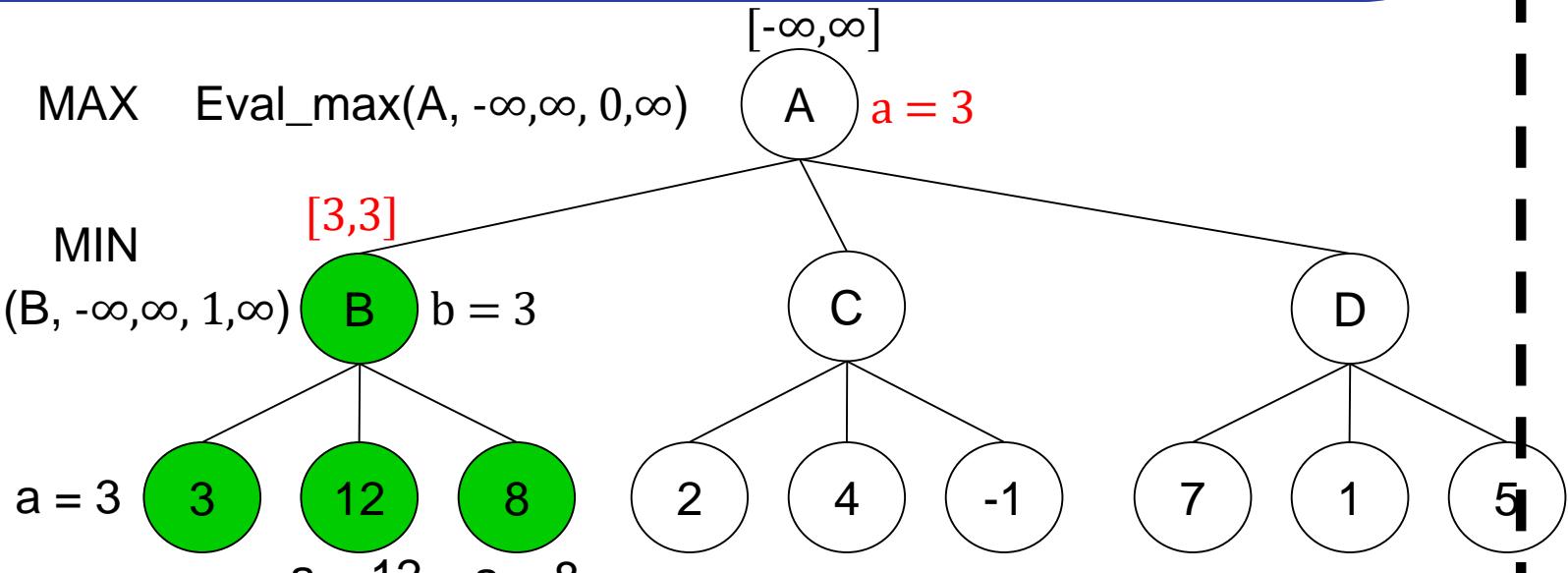
Pentru fiecare ($n' \in \text{succs}(n)$) {

$b = \min(b, \text{eval_max}(n', \alpha, \min(\beta, b), \text{nivel}+1, \text{limită}))$; // propag

Dacă ($b \leq \alpha$) **oprire**; }

Întoarce b

Alt exemplu (VI)



`eval_max(n, α, β, nivel, limită)`

Dacă n este frunză **Întoarce** $\text{cost}(n)$

Dacă $(\text{nivel} \geq \text{limită})$ **Întoarce** $\text{euristică}(n)$ // sunt limitat

$a = -\infty$ // valoarea curentă a nodului de tip max

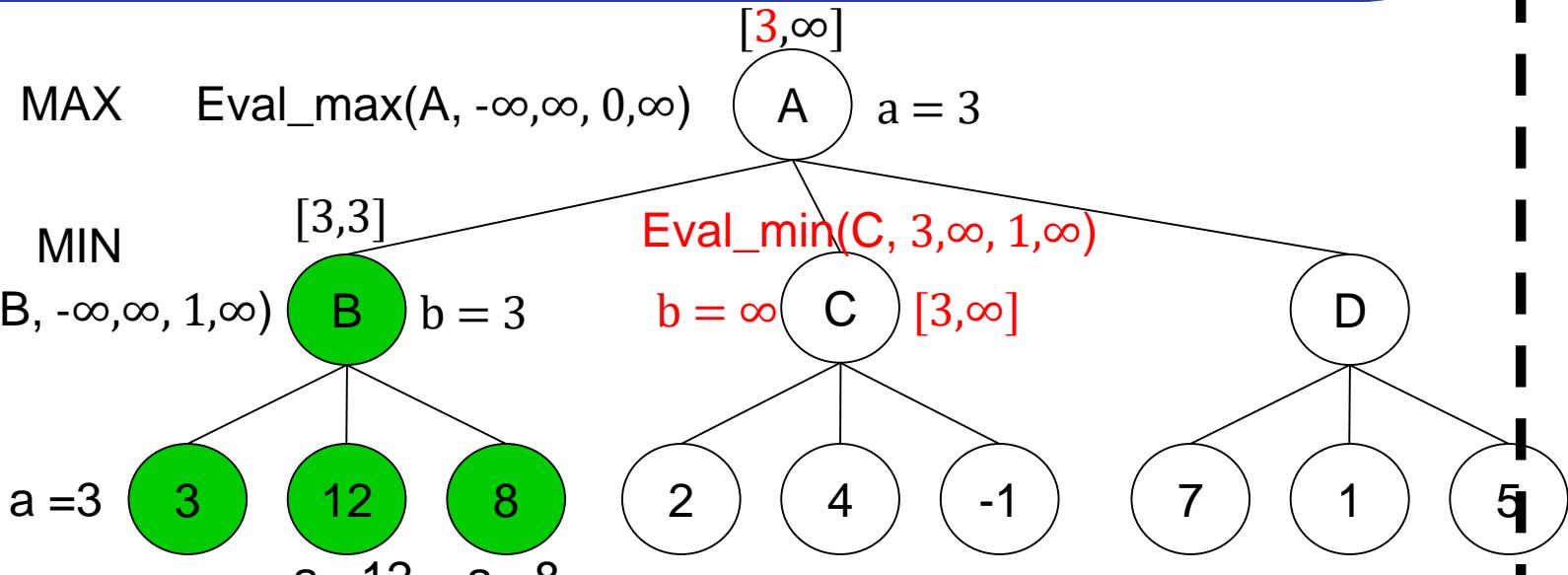
Pentru fiecare $(n' \in \text{succs}(n))$ {

$a = \max(a, \text{eval_min}(n', \max(\alpha, a), \beta, \text{nivel}+1, \text{limită}))$; // propag

Dacă $(a \geq \beta)$ **oprire**;

Întoarce a

Alt exemplu (VII)



Dacă n este frunză **Întoarce** $\text{cost}(n)$

Dacă ($\text{nivel} \geq \text{limita}$) **Întoarce** $\text{euristică}(n)$ // sunt limitat

$b = \infty$ // valoarea curentă a nodului de tip min

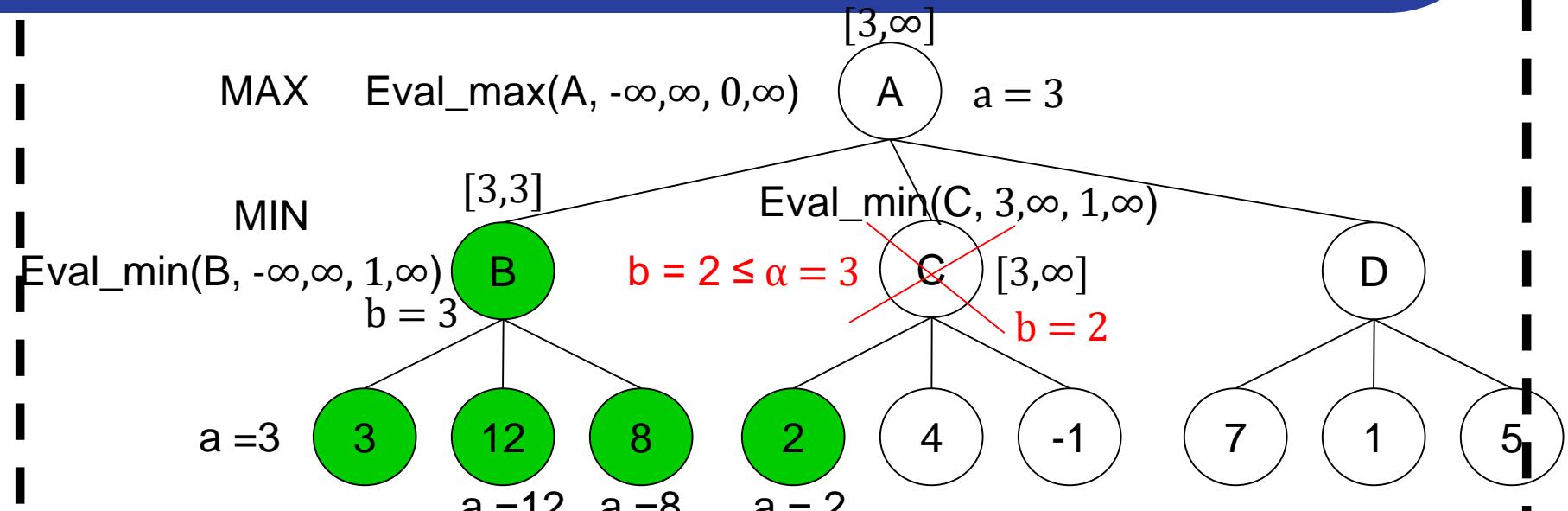
Pentru fiecare ($n' \in \text{succs}(n)$) {

$b = \min(b, \text{eval_max}(n', \alpha, \min(\beta, b), \text{nivel}+1, \text{limită}))$; // propag

Dacă ($b \leq \alpha$) **oprire**; }

Întoarce b

Alt exemplu (VIII)



eval_min($n, \alpha, \beta, \text{nivel}, \text{limită}$)

Dacă n este frunză **Întoarce** cost(n)

Dacă ($\text{nivel} \geq \text{limită}$) **Întoarce** euristică(n) // sunt limitat

$b = \infty$ // valoarea curentă a nodului de tip min

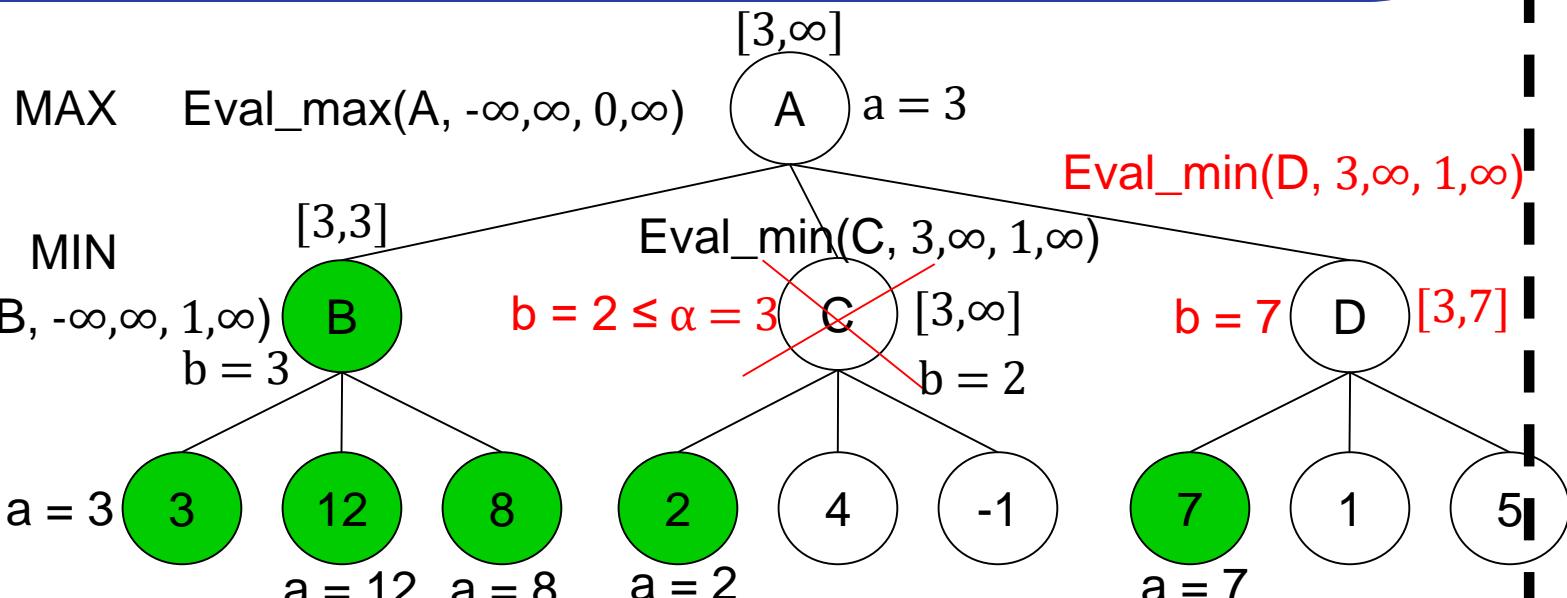
Pentru fiecare ($n' \in \text{succs}(n)$) {

$b = \min(b, \text{eval_max}(n', \alpha, \min(\beta, b), \text{nivel}+1, \text{limită}))$; // propag

Dacă ($b \leq \alpha$) **oprire**; }

Întoarce b

Alt exemplu (IX)



`eval_min(n, α, β, nivel, limită)`

Dacă n este frunză **Întoarce** $\text{cost}(n)$

Dacă ($\text{nivel} \geq \text{limită}$) **Întoarce** $\text{euristică}(n)$ // sunt limitat

$b = \infty$ // valoarea curentă a nodului de tip min

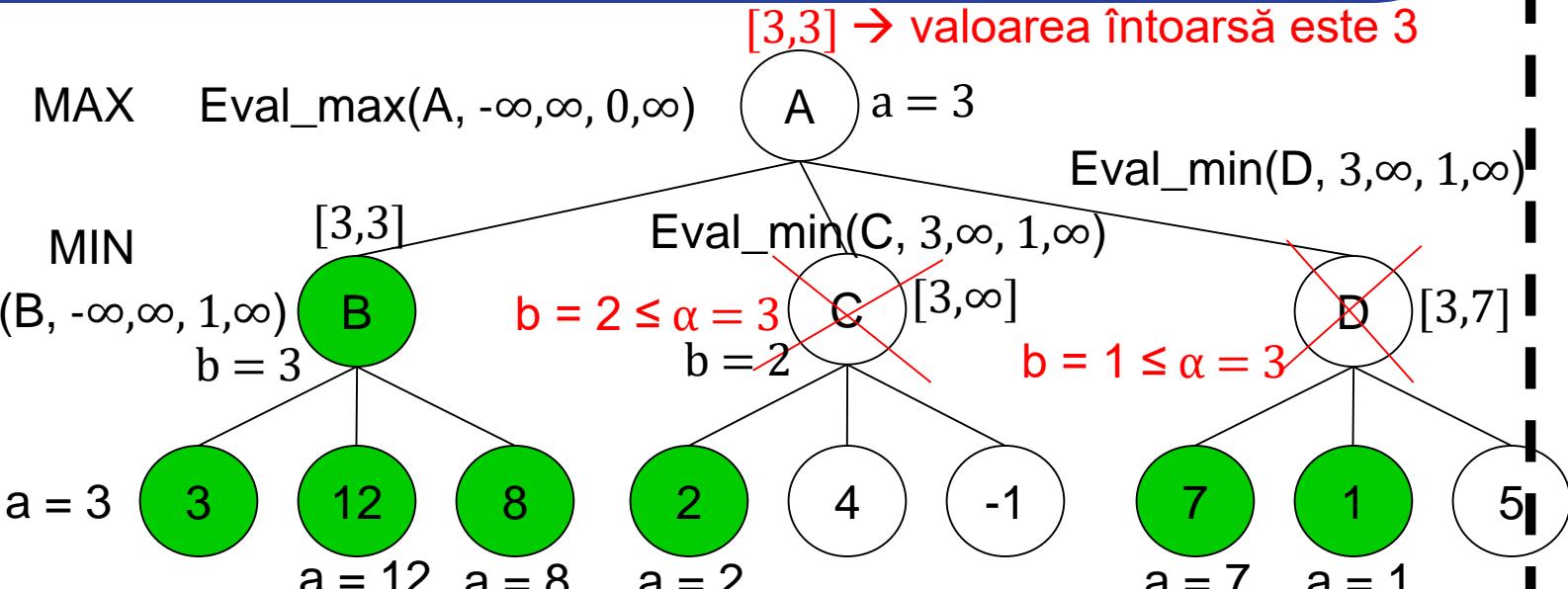
Pentru fiecare ($n' \in \text{succs}(n)$) {

$b = \min(b, \text{eval_max}(n', \alpha, \min(\beta, b), \text{nivel}+1, \text{limită}))$; // propag

Dacă ($b \leq \alpha$) **oprire**; }

Întoarce b

Alt exemplu (X)



`eval_max(n, α, β, nivel, limită)`

Dacă n este frunză **Întoarce** $\text{cost}(n)$

Dacă $(\text{nivel} \geq \text{limita})$ **Întoarce** $\text{euristică}(n)$ // sunt limitat

$a = -\infty$ // valoarea curentă a nodului de tip max

Pentru fiecare $(n' \in \text{succs}(n))$ {

$a = \max(a, \text{eval_min}(n', \max(\alpha, a), \beta, \text{nivel}+1, \text{limita}));$ // propag

Dacă $(a \geq \beta)$ **oprire**;

Întoarce a

Observații α - β

- Reduce complexitatea minimax în cazul ideal de la
 - Număr_ramificări^{număr_nivele} la Număr_ramificări^{număr_nivele/2}
- Contează foarte mult **ordinea** în care analizăm mutările!
 - **Sortarea mutărilor** după un criteriu dat **nu este costisitoare** comparativ cu costul exponential al algoritmului.
- Se folosesc euristici pentru a alege mutările examineate mai întâi:
 - ex: la șah se aleg întâi mutările în care se iau piese;
 - sau se aleg mai întâi mutările cu scor bun în parcurgeri precedente;
 - sau se aleg mutările care au mai generat tăieri.

Observații MINIMAX și α - β

- Algoritmi de căutare în adâncime.
- Pot cauza probleme când avem un timp limită.
- → Soluție posibilă IDDFS (căutare în adâncime mărind iterativ adâncimea maximă până la care căutăm).

Concluzii

- Algoritmi cu **complexitate foarte mare**.
- **Soluții euristică** pentru **limitarea complexității**.
- Recomandabil să se **combine** cu alte **strategii** – baze de date cu pozitii, pattern-matching.

ÎNTREBĂRI?

Proiectarea Algoritmilor

Curs 13 - Algoritmi euristici de
explorare

Bibliografie

- [1] C. Giu male – Introducere in Analiza Algoritmilor - cap. 7
- [2] <http://www.policyalmanac.org/games/aStarTutorial.htm>
- [3] http://www.ai.mit.edu/courses/6.034b/searchcompl_ex.pdf

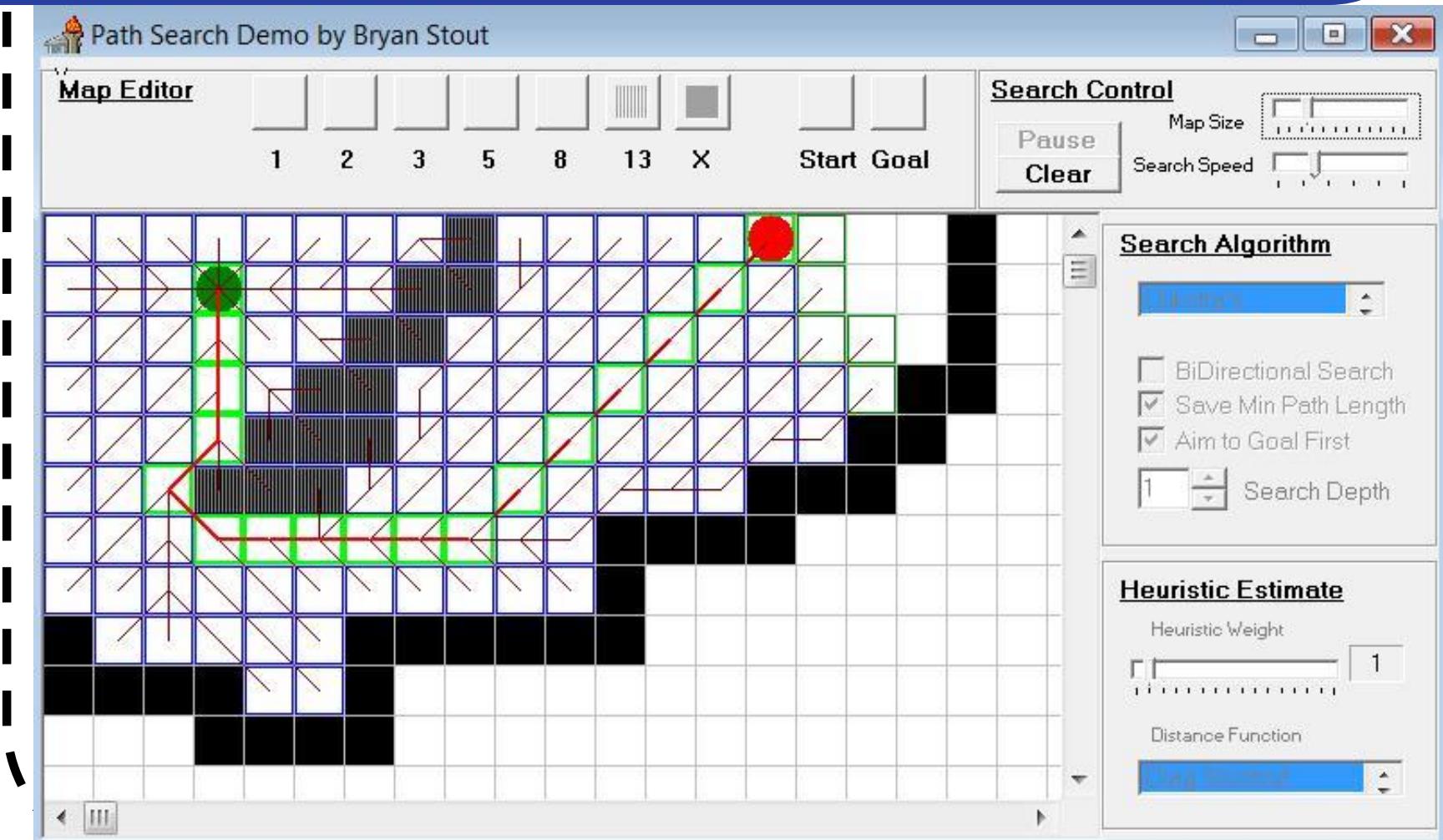
Cuprins

- Explorarea spațiului stărilor problemei
- Explorare informată irevocabilă
- Explorări tentative informate
 - Explorare lacomă
 - Explorare tentativă completă
 - Explorare A*

Probleme cu căutările neinformate

- Modelul unor probleme este prea complicat → variantele de rezolvare se bazează pe explorarea spațiului stărilor.
- Probleme:
 - Deseori se calculează prea mult (ex: drumul optim între 2 puncte folosind Dijkstra) - ex: Dijkstra.
 - În cazul grafurilor infinite sau nedescoperite încă, algoritmii clasici fie sunt ineficienți, fie nu garantează găsirea soluției.
- Soluție:
 - Rezolvarea să nu se mai bazeze numai pe calculele exacte ci și pe experiența anterioară (euristici) → direcționarea căutării.

Exemplu Dijkstra



Explorarea spațiului stărilor problemei

Spațiul stărilor unei probleme

- Definiție: Stare a problemei = abstractizare a unei configurații valide a universului problemei, configurație ce determină univoc comportarea locală a fenomenului descris de problemă.
- Definiție: Spațiul stărilor = graf în care nodurile corespund stărilor problemei, iar arcele desemnează tranzițiile valide între stări.
 - Caracteristică importantă: nu este cunoscut apriori, ci este descoperit pe măsura explorării!
 - Descriere
 - Nodul de start (starea inițială);
 - Funcție de expandare a nodurilor (produce lista nodurilor asociate stărilor valide în care se poate ajunge din starea curentă);
 - Predicat de testare dacă un nod corespunde unei stări soluție.

Obiectivele navigării prin spațiul stărilor

- **Cartografierea** sistematică a spațiului stărilor.
- Asamblarea soluțiilor parțiale care în final conduc la soluția finală. Această soluție finală poate fi:
 - Identificarea stărilor soluție (poziționarea a n regine pe tablă de șah fără să se atace);
 - Drumul străbătut de la starea inițială spre o stare soluție (acoperirea tablei de șah cu un cal);
 - Strategia de rezolvare = arbore multicăi în care rădăcina este starea inițială, iar frunzele sunt stări soluție. În acest arbore, unele noduri corespund unor **evenimente neprevăzute care influențează calea de urmat** în rezolvare (identificarea monedei false dintr-un grup de 3 monede).

Căutări informate/neinformate; Algoritmi tentativi/irevocabili

- **Definiție:** Dacă explorarea se bazează pe informația acumulată în cursul explorării, informație prelucrată **euristic** (costuri) → **algoritm informat**.
- **Definiție:** Dacă explorarea este ‘la întâmplare’ → **algoritm neinformat**.
- **Definiție:** Dacă algoritmul de explorare are posibilitatea să abandoneze calea curentă de rezolvare și să revină la o cale anterioară → **algoritmi tentativi**.
- **Definiție:** Altfel (algoritmul avansează pe o singură direcție) → **algoritmi irevocabili**.

Căutări informate vs neinformate

- Căutările informate beneficiază de **informații suplimentare** pe care le colectează și le utilizează în încercarea de a ghici direcția în care trebuie explorat spațiul stărilor pentru a găsi soluția.
- Aceste informații sunt stocate:
 - În **nodurile din spațiul stărilor**:
 - Starea problemei reprezentată de nod;
 - Părintele nodului curent;
 - Copii nodului curent (obținuți prin expandarea acestuia);
 - Costul asociat nodului curent care **estimează calitatea nodului $f(n)$** ;
 - Adâncimea de explorare.
 - În **structuri auxiliare pentru diferențierea nodurilor în raport cu gradul de prelucrare**:
 - **Expandat (închis)** – toți succesorii nodului sunt cunoscuți;
 - **Explorat (deschis)** – nodul e cunoscut, dar succesorii săi nu;
 - **Neexplorat** – nodul nu e cunoscut.

Listele CLOSED și OPEN

- **OPEN** = multimea (lista) nodurilor **explorate** (frontiera dintre zona cunoscută și cea necunoscută).
- **CLOSED** = multimea (lista) nodurilor **expandate** (regiunea cunoscută în totalitate).
- Explorarea zonelor necunoscute se face prin **alegerea** și **expandarea** unui nod din **OPEN**. După expandare, nodul respectiv e trecut în **CLOSED**.
- Majoritatea algoritmilor tentativi folosesc lista **OPEN**, dar doar o parte folosesc lista **CLOSED**.

Completitudine și optimalitate

- Definiție: Algoritm complet = algoritm de explorare care garantează descoperirea unei soluții, dacă problema acceptă soluție.
 - Algoritmii irevocabili sunt mai rapizi și consumă mai puține resurse decât cei tentativi, dar nu sunt compleți pentru că pierd informație.
- Definiție: Algoritm optimal = algoritm de explorare care descoperă soluția optimă a problemei.

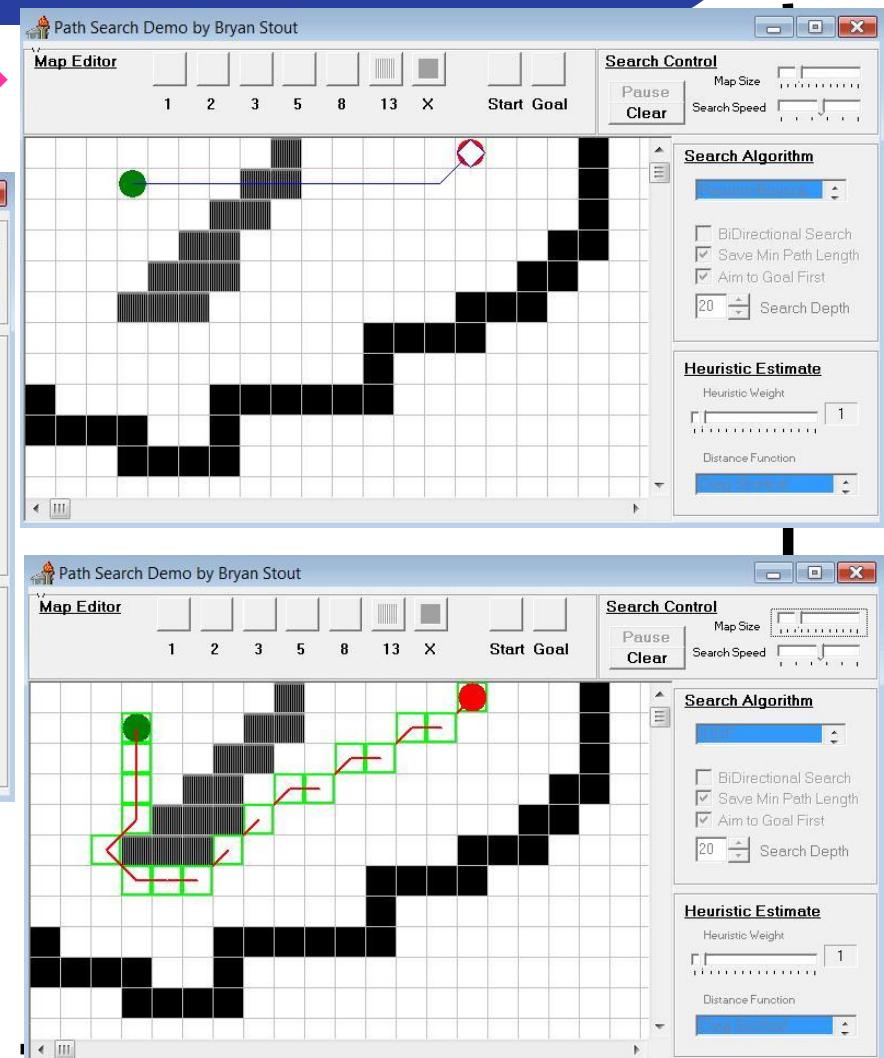
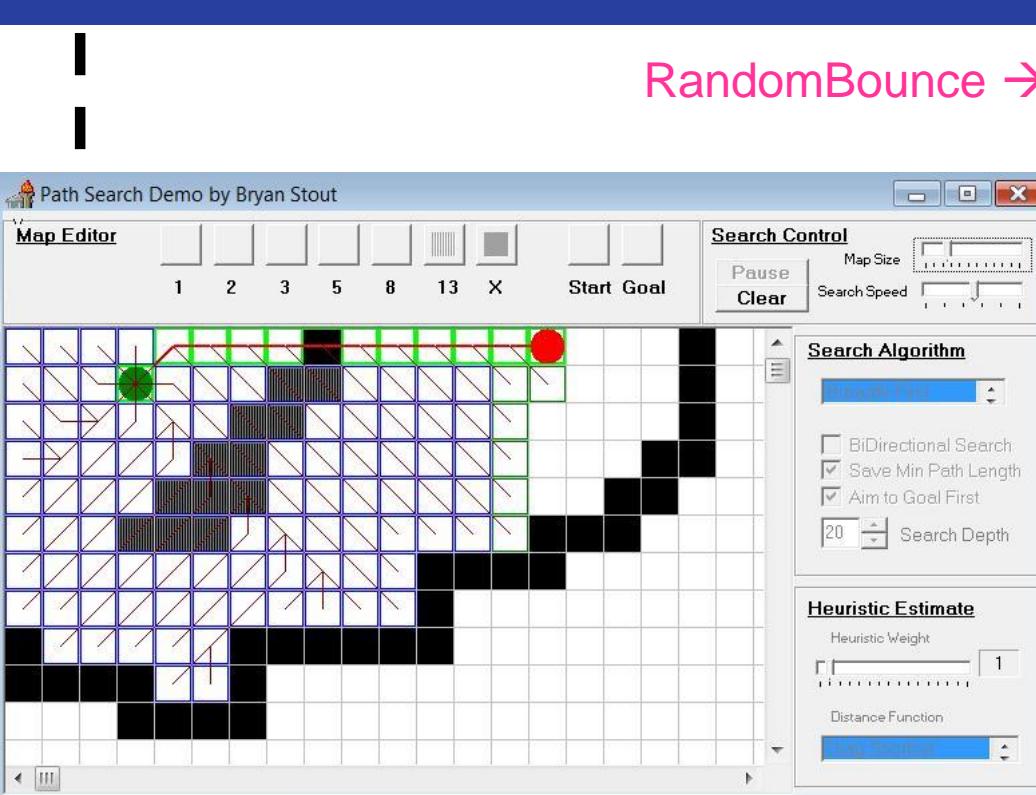
Algoritm generic de explorare

- Explorare(StInit, test_sol)
 - OPEN = {constr_nod(StInit)}; // starea inițială
 - Cât timp ($OPEN \neq \emptyset$)
 - nod = selecție_nod(OPEN); // aleg un nod
 - Dacă (test_sol(nod)) Întoarce nod;
 - // am găsit o soluție
 - OPEN = $OPEN \setminus \{nod\} \cup \text{expandare}\{nod\}$;
 - // extind căutarea
 - Întoarce insucces; // nu s-a găsit nicio soluție

Discuție pe baza algoritmului

- Dacă **selecție_nod** se realizează independent de costul nodurilor din graful stărilor → **căutare neinformată**:
 - Dacă e de tip “random” → algoritm aleator - ex: RandomBounce
 - Dacă e de tip “primul venit, primul servit” → OPEN e coadă → **BFS**
 - ex: Breadth-first
 - Dacă e de tip “ultimul venit, primul servit” → OPEN e stivă → **DFS**
 - ex: Depth-first limitat / IDDFS
- Dacă **selecție_nod** se bazează pe un cost exact sau estimat (euristic) al stărilor problemei → **căutare informată**:
 - Estimarea costului și folosirea sa în procesul de selecție → **esențiale pentru completitudinea, optimalitatea și complexitatea algoritmilor de explorare!**

Exemplu de căutări neinformate



Explorare informață irevocabilă

Algoritm de explorare informată irevocabilă

- Ex: algoritmul alpinistului = **algoritmul gradientului maxim.**
- Fiecărui nod i se asociază o valoare $f(\text{nod}) \geq 0 \rightarrow$ calitatea soluției parțiale din care face parte nodul.
- Se păstrează doar cel cu valoare maximă \Rightarrow **OPEN are un singur element!**

Gradientul Maxim

- Gradient_maxim(StInit, f, test_sol)

- nod = constr_nod(StInit); // starea initial
 - $\pi(\text{nod}) = \text{null};$

Initializări

- Cât timp ($\text{!test_sol}(\text{nod})$)

Testez soluția

- succs = expandare(nod); // nodurile au o valoare estimată
// prin f

• Dacă ($\text{succs} = \emptyset$) Întoarce insucces; Insucces

// nu mai am noduri de prelucrat

- succ = selecție_nod(succs); // $f(\text{succ}) = \max \{f(n) \mid n \in \text{succs}\}$
 - $\pi(\text{succ}) = \text{nod};$
 - nod = succ;

Găsesc calea de continuat

- Întoarce nod; // am ajuns la soluție

Soluția

Gradientul Maxim

Optimalitate?

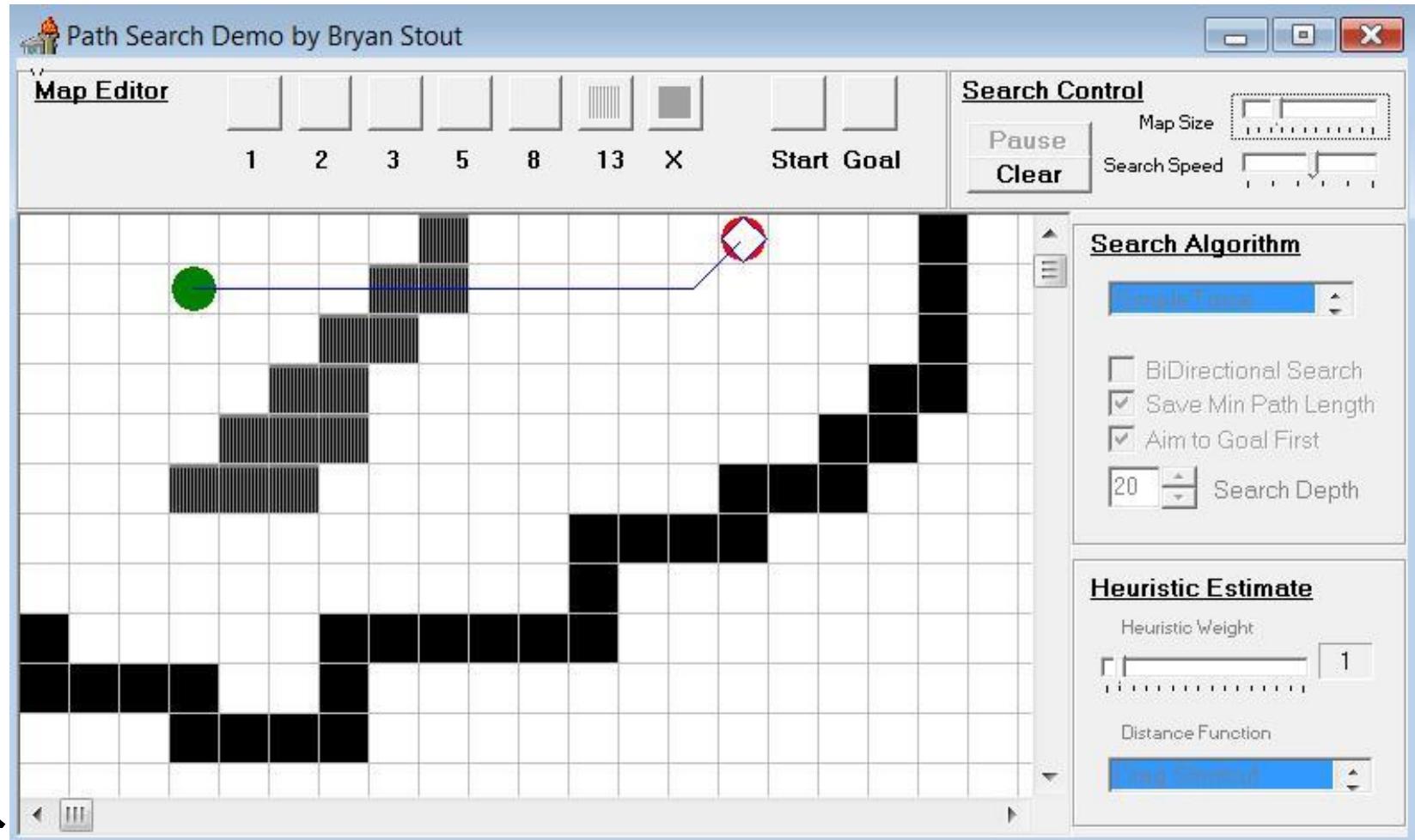
Compleitudine?

Complexitate?

Ex: SimpleTrace



Exemplu Gradient Maxim



Discuție algoritmul gradientului maxim

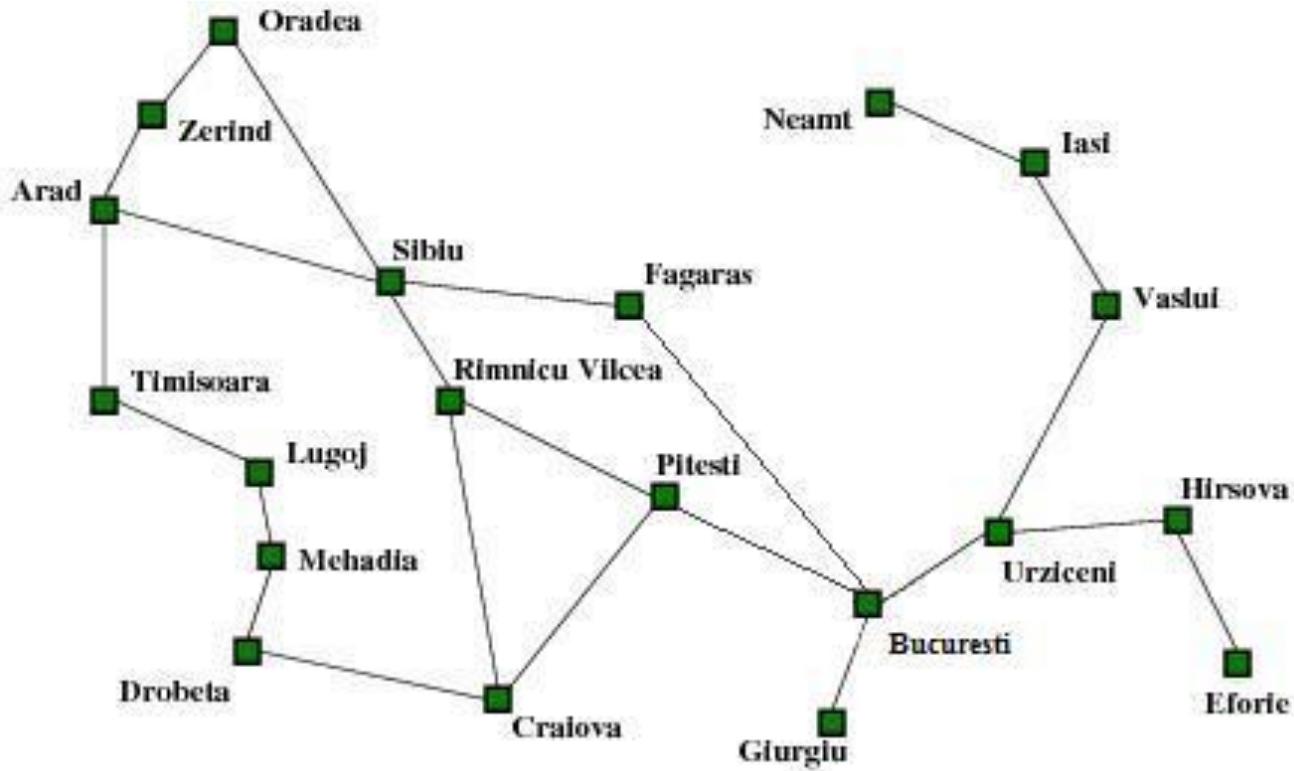
- Algoritmul **nu e complet și nu e optimal!**
- **Complexitate scăzută:** $O(bd)$ - b = branching factor, iar d = depth!
- Performanțele algoritmului depind foarte tare de forma teritoriului **explorat** și de **euristica folosită** (de dorit să existe puține optime locale și o **uristică de evaluare** cât mai bună).
- **Pseudo-soluție eliminare optim local:** se lansează algoritmul de mai multe ori plecând din stări initiale diferite și se alege cea mai bună soluție obținută.

Explorări tentative informate

Detalii generale

- Păstrează toate nodurile de pe frontieră (**OPEN**), unii păstrând și nodurile expandate (**CLOSED**).
- Fiecare nod are un cost asociat $f(n) \geq 0$ care **estimează calitatea nodului** (distanța de la nodul respectiv până la un nod soluție).
- Cu cât **$f(n)$ este mai mic**, cu atât **nodul este mai bun**.

Prezentarea problemei



- Trebuie să ajungem în Bucureşti din diverse puncte ale țării pe ruta cea mai scurtă.

Explorare lacomă

- Explorare_lacomă (StInit, f, test_sol)

- nod = constr_nod(StInit); // starea inițială
- $\pi(\text{nod}) = \text{null};$
- OPEN = {nod};

Inițializări

- Cât timp ($\text{OPEN} \neq \emptyset$) // mai am noduri de prelucrat

- nod = selecție_nod (OPEN); // $f(\text{nod}) = \min \{f(n) \mid n \in \text{OPEN}\}$

- Dacă (test_sol(nod)) Întoarce nod:

Solutia

- OPEN = OPEN \ {nod}; // nodul nu e soluție, trebuie expandat

- succs = expand(nod); // expandare nod

- Pentru fiecare ($\text{succ} \in \text{succs}$) // actualizare succesori

- OPEN = OPEN U {succ};

- $\pi(\text{succ}) = \text{nod};$

Continuarea căutării

- Întoarce insucces; Insucces

Optimalitate?

Completitudine?

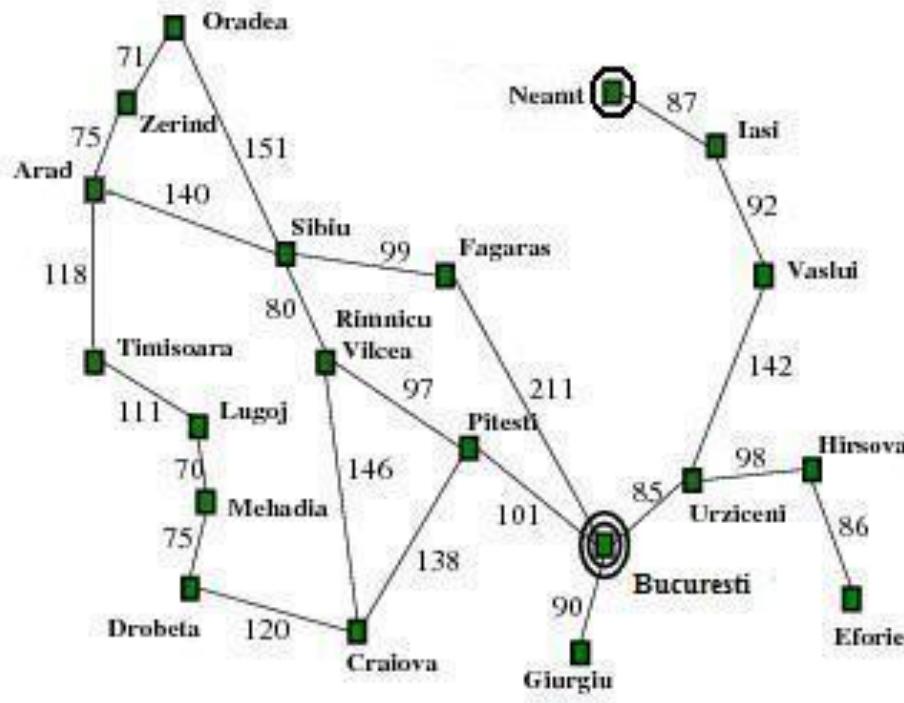
Problema?

$f(\text{nod}) = \text{distanța de la nodul curent până la nodul nod}$

$f(\text{Iași}) = 87 \rightarrow \text{Iași}$

$f(\text{Neamț}) = 87$

$f(\text{Vaslui}) = 92 \rightarrow \text{Neamț}$



- Drumul Neamț-București? → nu se termină algoritmul!
- → Explorarea lacomă nu e completă → trebuie să se rețină teritoriul deja parcurs ca să se evite ciclurile!

Explorare tentativă completă BF* (BEST FIRST) (1)

- **BF*(StInit, f, test_sol)**

- nod = constr_nod(StInit); // starea inițială
- $\pi(\text{nod}) = \text{null};$
- OPEN = {nod}; // noduri explorate dar neexpandate
- CLOSED = $\emptyset;$ // noduri expandate **Inițializări**

- **Cât timp** ($\text{OPEN} \neq \emptyset$)

- nod = selecție_nod(OPEN); // $f(\text{nod}) = \min \{f(n) \mid n \in \text{OPEN}\}$

- **Dacă** (test_sol(nod)) **Întoarce** nod;

Soluția

- OPEN = OPEN \ {nod};
- CLOSED = CLOSED U {nod};
- succs = expand(nod);

Continuarea căutării

Explorare tentativă completă BF* (BEST FIRST) (2)

- Pentru fiecare ($\text{succ} \in \text{succs}$)

- Dacă ($\text{succ} \notin \text{CLOSED} \cup \text{OPEN}$) atunci
 - $\text{OPEN} = \text{OPEN} \cup \{\text{succ}\}; \pi(\text{succ}) = \text{nod};$

Nod nou

- Altfel
 - $\text{succ}' = \text{apariția lui succ în } \text{CLOSED} \cup \text{OPEN}$
 - Dacă ($f(\text{succ}) < f(\text{succ}')$) // am găsit o cale mai bună către succ și // redeschidem nodul
 - $\pi(\text{succ}') = \text{nod}; // \text{actualizez părintele}$
 - $f(\text{succ}') = f(\text{succ}); // și costul nodului$

// redeschidem nodul

$\pi(\text{succ}') = \text{nod}; // \text{actualizez părintele}$

$f(\text{succ}') = f(\text{succ}); // și costul nodului$

Actualizări

Reprelucrare

Dacă ($\text{succ}' \in \text{CLOSED}$) // dacă era considerat expandat, îl redeschid

$\text{CLOSED} = \text{CLOSED} \setminus \{\text{succ}'\}; \text{OPEN} = \text{OPEN} \cup \{\text{succ}'\};$

- Întoarce insucces;

Insucces

ex: Best-first cu diverse euristici

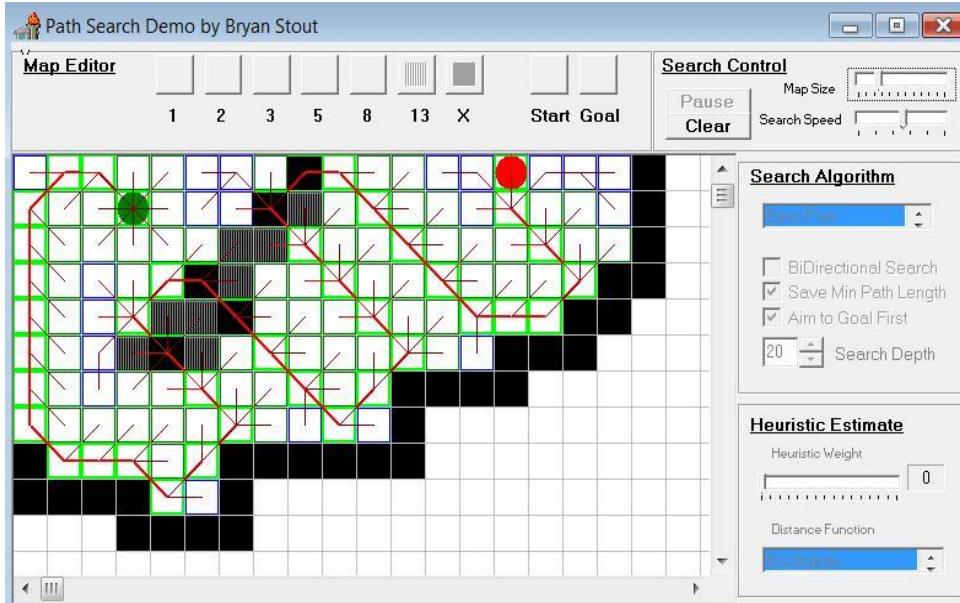
Optimalitate?

Completitudine?

Complexitate?

Exemple rulare BF* cu diferite euristici

BF* - Euristică Distanță Euclidiană (11 pași, cost 23) →

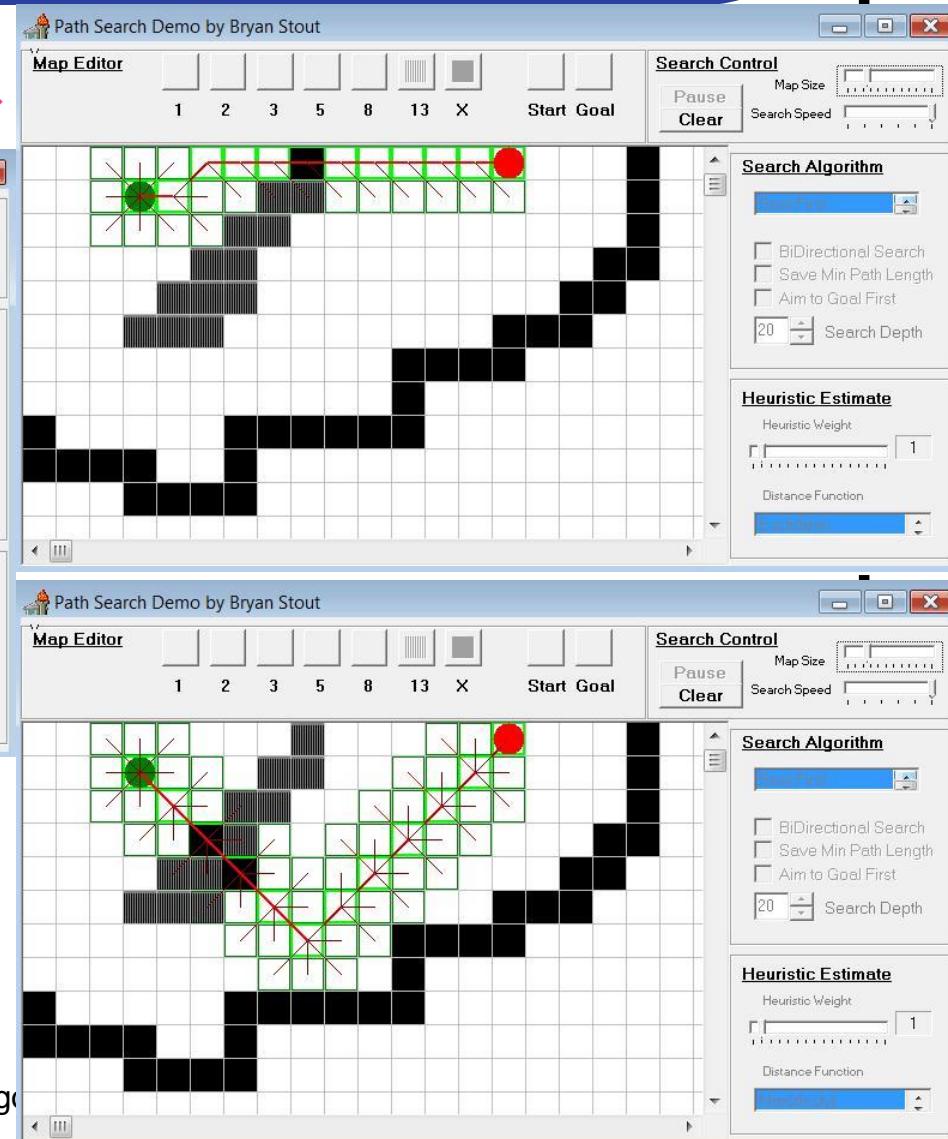


BF* fără euristici ↑

BF* - Euristică maximul distanței pe x și pe y (11 pași, cost 35) →



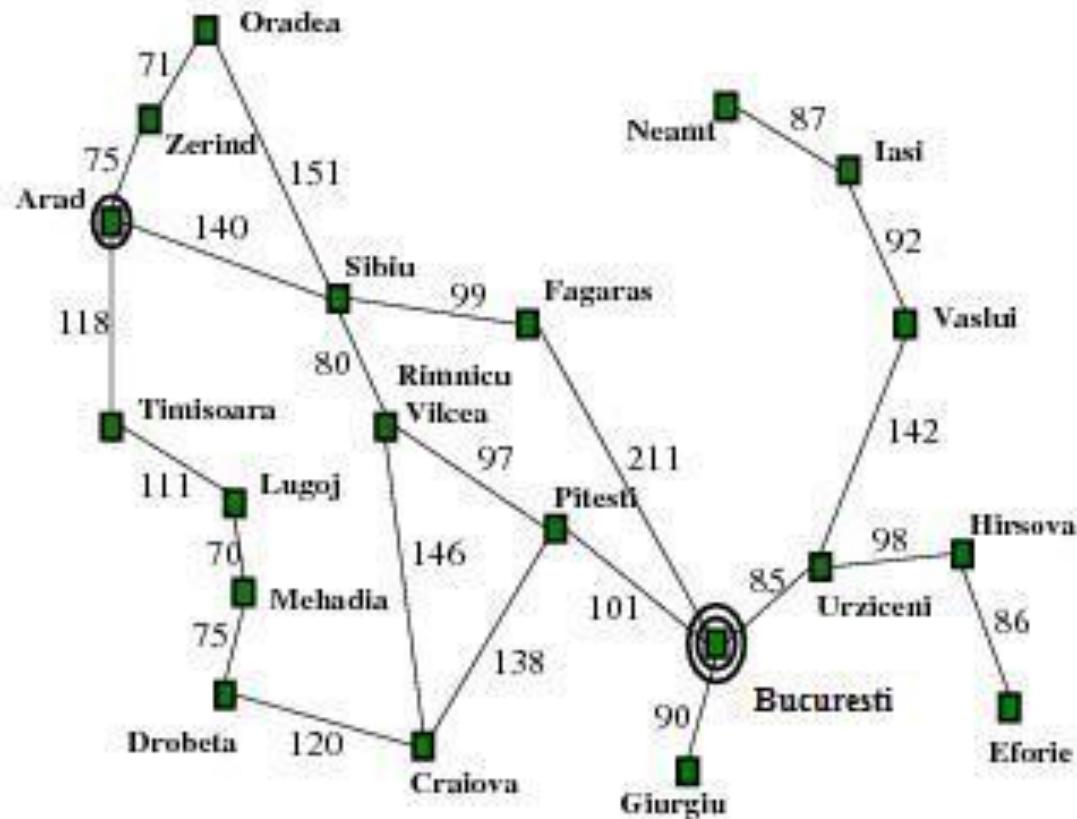
Proiectarea Algo



BF* - completitudine, optimalitate și complexitate

- Păstrează întreg teritoriul explorat:
 - OPEN – nodurile de pe frontieră;
 - CLOSED – nodurile expandate (unele noduri pot fi redeschise) → se evită ciclurile.
- Algoritmul este complet dar nu este optim
→ optimalitatea depinde de euristica $f!$
- Complexitate: $O(b^{d+1})$

Aplicație BF*



Distanța în linie dreaptă până la București

Arad	366
Craiova	160
Drobeta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Drumul optim Arad-București ($f(\text{nod}) = \text{distanța în linie dreaptă până la București}$)

A*

- Variantă a BF*.
- Nu poate fi aplicat mereu → trebuie demonstrat că păstrează ordinea soluțiilor unde soluțiile problemelor sunt drumuri în spațiul stărilor! ([vezi Giumale pentru detalii!](#))
- Costul unui drum este **aditiv** (= suma costurilor arcelor) și **crescător** în lungul drumului.
- Folosește două funcții de cost:
 - $h(n)$ - distanța estimată de la nodul curent până la nodul țintă;
 - $g(n)$ - distanța parcursă de la nodul initial până la nodul curent;
 - $f(n) = g(n) + h(n)$.

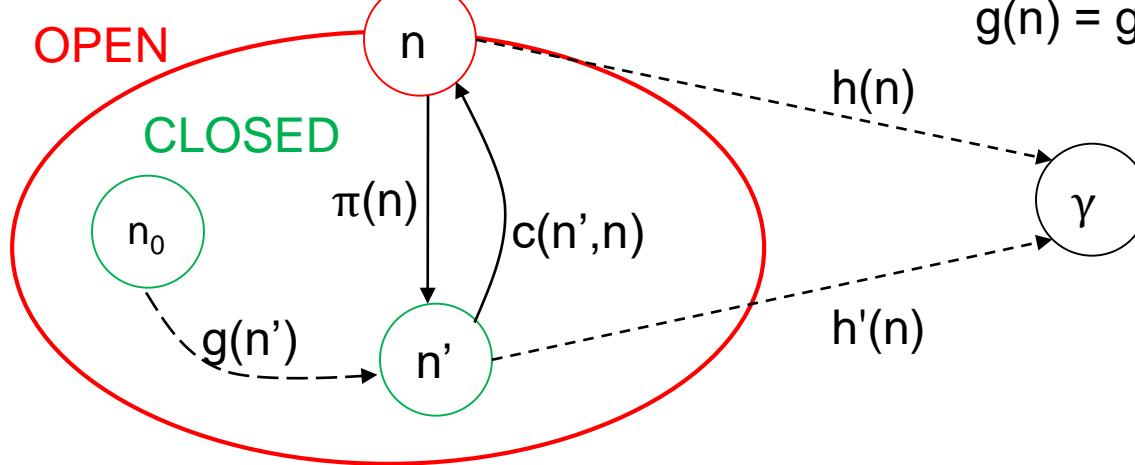
Notări (1)

- $S = (V, E)$ – **graful** asociat spațiului stărilor problemei;
- n_0 – **nodul de start** asociat stării inițiale a problemei;
- $\Gamma \subseteq V$ – **mulțimea nodurilor soluție**. Un nod soluție se notează γ ;
- $c(n, n') > 0$ – **costul arcului** (n, n') ;
- $\pi(n)$ – **părintele** lui n ;
- $g(n)$ – **costul** drumului $n_0..n$ **descoperit de algoritm** la momentul curent de timp;
- $g_p(n)$ – **costul exact** al porțiunii $n_0..n$ din lungul unei căi date P ;
- $g^*(n)$ – **costul exact** al unui **drum optim** $n_0..n$;

Notări (2)

- $h(n) \geq 0$ – costul estimat al drumului optim de la nodul n la cel mai favorabil nod soluție $\gamma \in \Gamma$. În plus $h(\gamma) = 0$, pentru orice $\gamma \in \Gamma$;
- $h^*(n)$ – costul exact al porțiunii de drum optim $n.. \gamma$, pentru cel mai favorabil nod $\gamma \in \Gamma$ ($h^*(n) = \min \{\text{cost}(n.. \gamma) | \gamma \in \Gamma\}$);
- $f(n) = g(n) + h(n)$ – costul estimat al întregului drum $n_0..n.. \gamma$, pentru cel mai favorabil nod $\gamma \in \Gamma$, unde porțiunea de drum $n_0..n$ este cea descoperită de algoritm la momentul curent de timp în cursul execuției;
- $f^*(n) = g^*(n) + h^*(n)$ – costul exact al unui drum optim $n_0..n.. \gamma$, pentru cel mai favorabil nod $\gamma \in \Gamma$;
- $C = \min\{f^*(\gamma) | \gamma \in \Gamma\}$ – costul exact al unui drum optim $n_0.. \gamma$, $\gamma \in \Gamma$. (C = costul soluției optime);

Funcția de evaluare A*



$$f(n) = g(n) + h(n)$$
$$g(n) = g(n') + c(n',n)$$

A* (1)

- A*(StInit, h, test_sol)

- $n_0 = \text{constr_nod}(StInit);$ // starea inițială

Initializări

- $f(n_0) = h(n_0); g(n_0) = 0; \pi(n_0) = \text{null};$ // euristici
 - $\text{OPEN} = \{n_0\}; \text{CLOSED} = \emptyset;$ // și cozi

- **Cât timp** ($\text{OPEN} \neq \emptyset$) // mai am noduri de prelucrat

- nod = selecție_nod (OPEN); // $f(\text{nod}) = \min \{f(n) \mid n \in \text{OPEN}\}$

- **Dacă** ($\text{test_sol}(\text{nod})$) **Întoarce** nod;

Soluția

- $\text{OPEN} = \text{OPEN} \setminus \{\text{nod}\};$ // updatez OPEN

- $\text{CLOSED} = \text{CLOSED} \cup \{\text{nod}\};$ // și CLOSE

- $\text{succs} = \text{expand}(\text{nod});$ // determin nodurile succesoare

Continuarea căutării

A* (2)

- Pentru fiecare ($\text{succ} \in \text{succs}$) { // prelucrare succs

• $g_{\text{succ}} = g(\text{nod}) + c(\text{nod}, \text{succ})$; // calculez g

• $f_{\text{succ}} = g_{\text{succ}} + h(\text{succ})$; // calculez $f = g + h$

Prelucrare
succesori

• Dacă ($\text{succ} \notin \text{CLOSED} \cup \text{OPEN}$) atunci // nod nou descoperit →

• $\text{OPEN} = \text{OPEN} \cup \{\text{succ}\}$; // îl bag în OPEN

$g(\text{succ}) = g_{\text{succ}}$; $f(\text{succ}) = f_{\text{succ}}$; $\pi(\text{succ}) = \text{nod}$;

Nod nou

• altfel // a mai fost prelucrat

• Dacă ($g_{\text{succ}} < g(\text{succ})$) // verific dacă noul g este mai mic decât
// anteriorul

$g(\text{succ}) = g_{\text{succ}}$; $f(\text{succ}) = f_{\text{succ}}$; $\pi(\text{succ}) = \text{nod}$; // cale mai bună

Actualizări

Reprelucrare

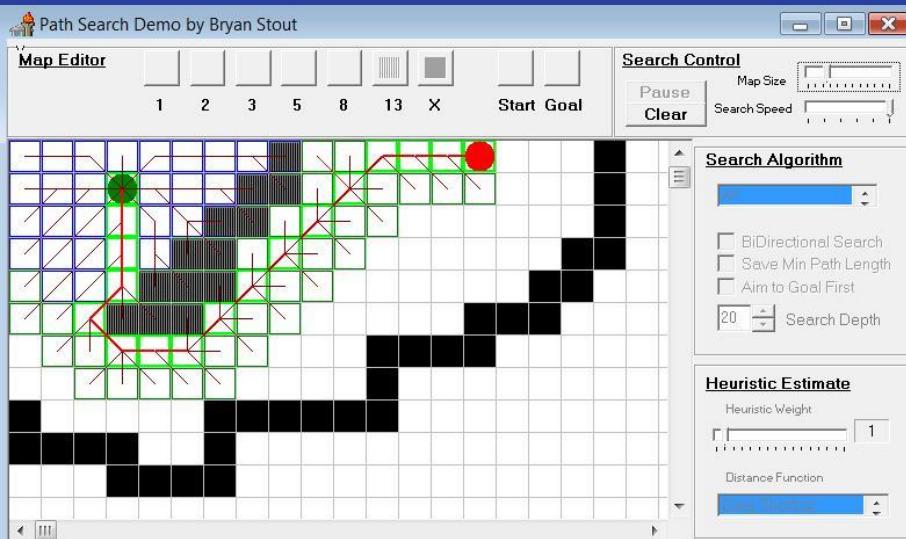
Dacă ($\text{succ} \in \text{CLOSED}$) // dacă era considerat expandat, îl redeschid

$\text{CLOSED} = \text{CLOSED} \setminus \{\text{succ}'\}$; $\text{OPEN} = \text{OPEN} \cup \{\text{succ}'\}$

- Întoarce Insucces; Insucces

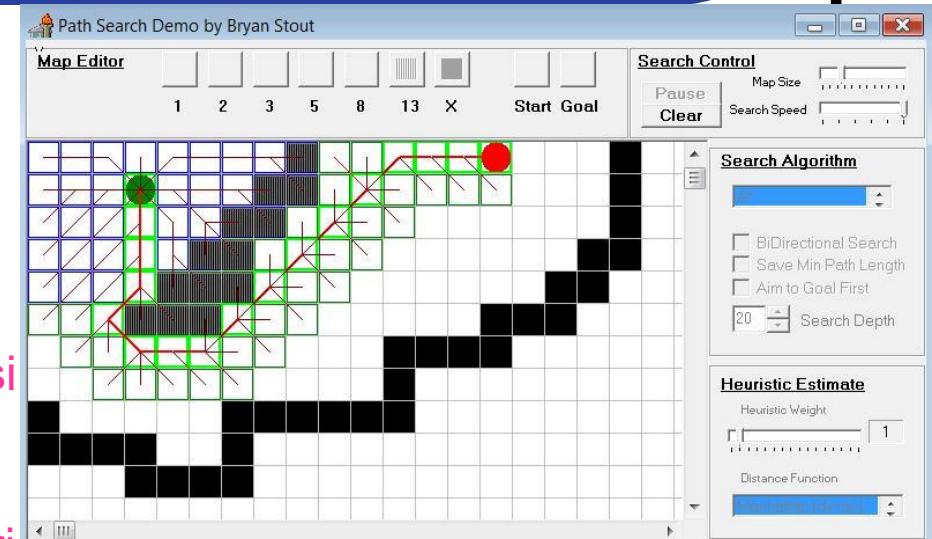
ex: A* cu diverse
euristici

Exemplu A* cu diverse euristici



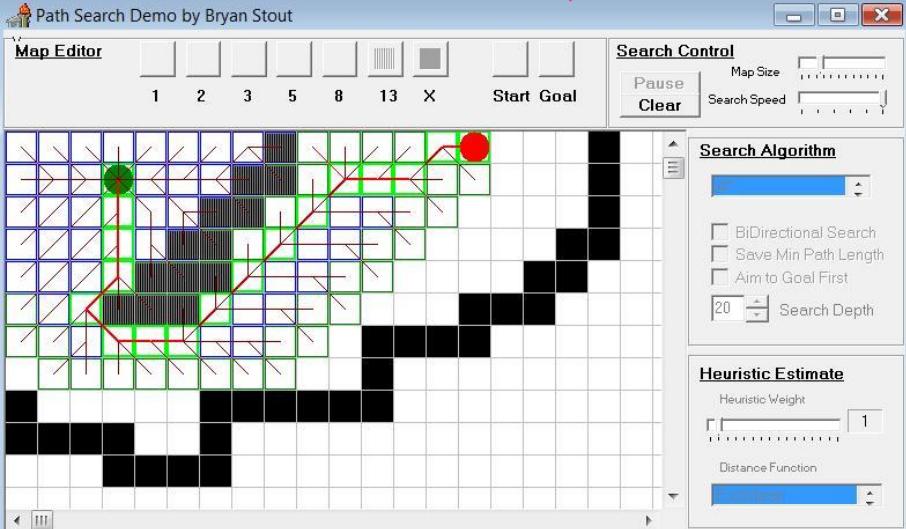
A*
16
pași

Diagonala ↑ Distanța Euclidiană ↓

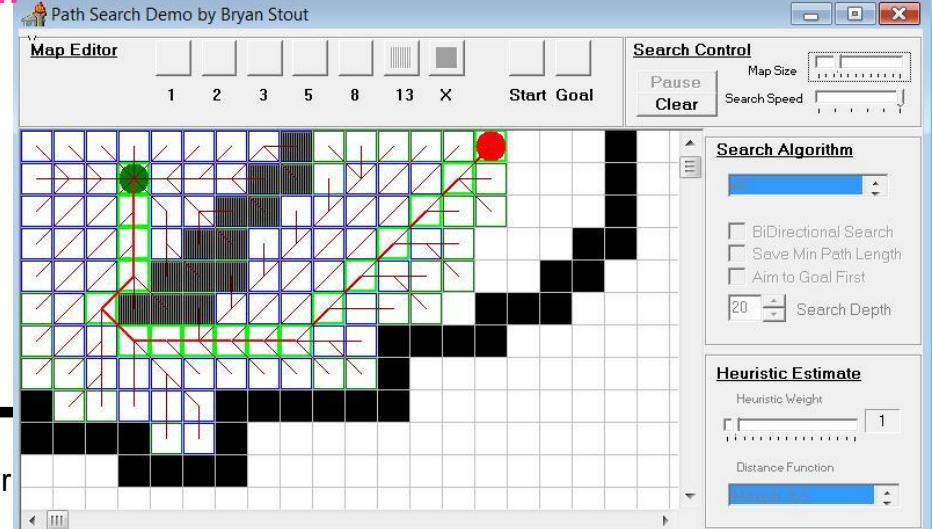


Euri
stici:

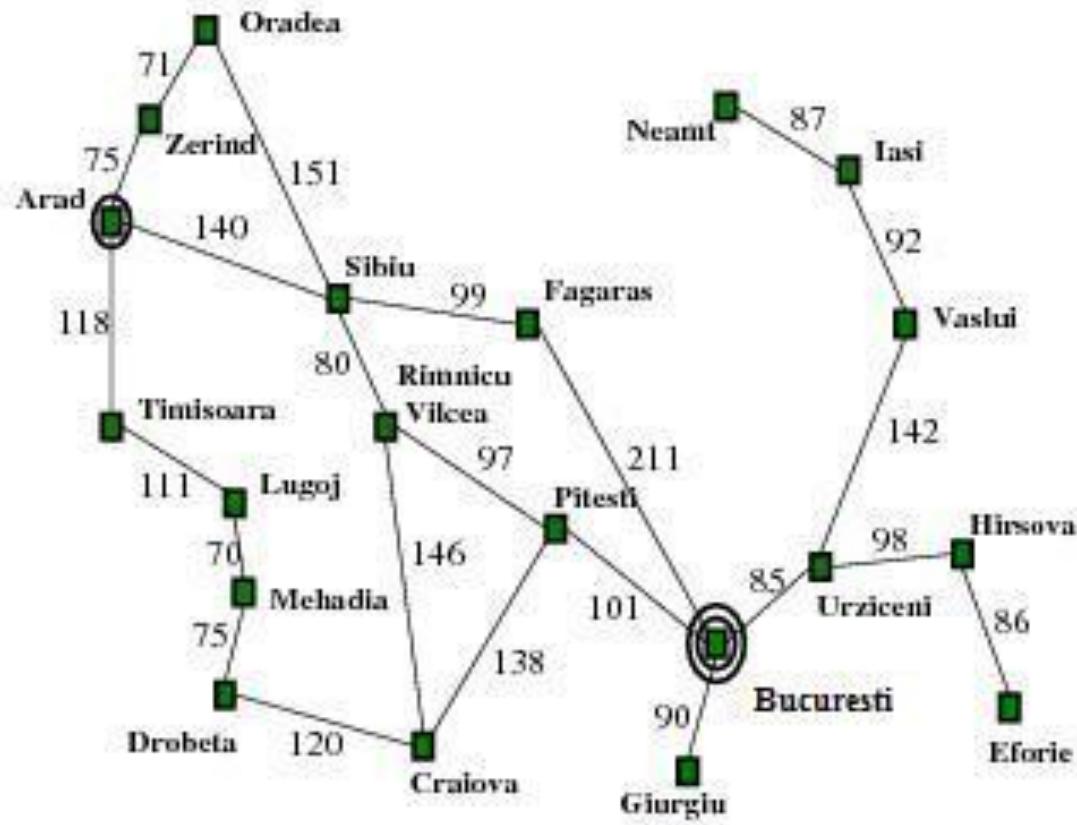
Distanța Manhattan ↑ Max(dx,dy) ↓



Algor



Aplicație A*

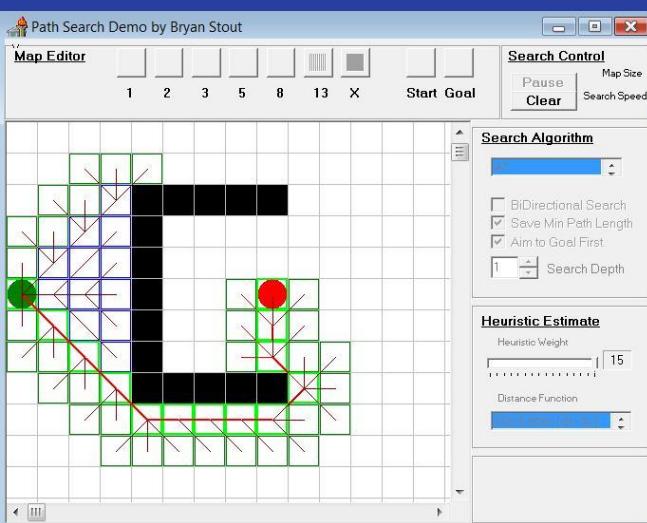


Distanță în linie dreaptă
pană la București

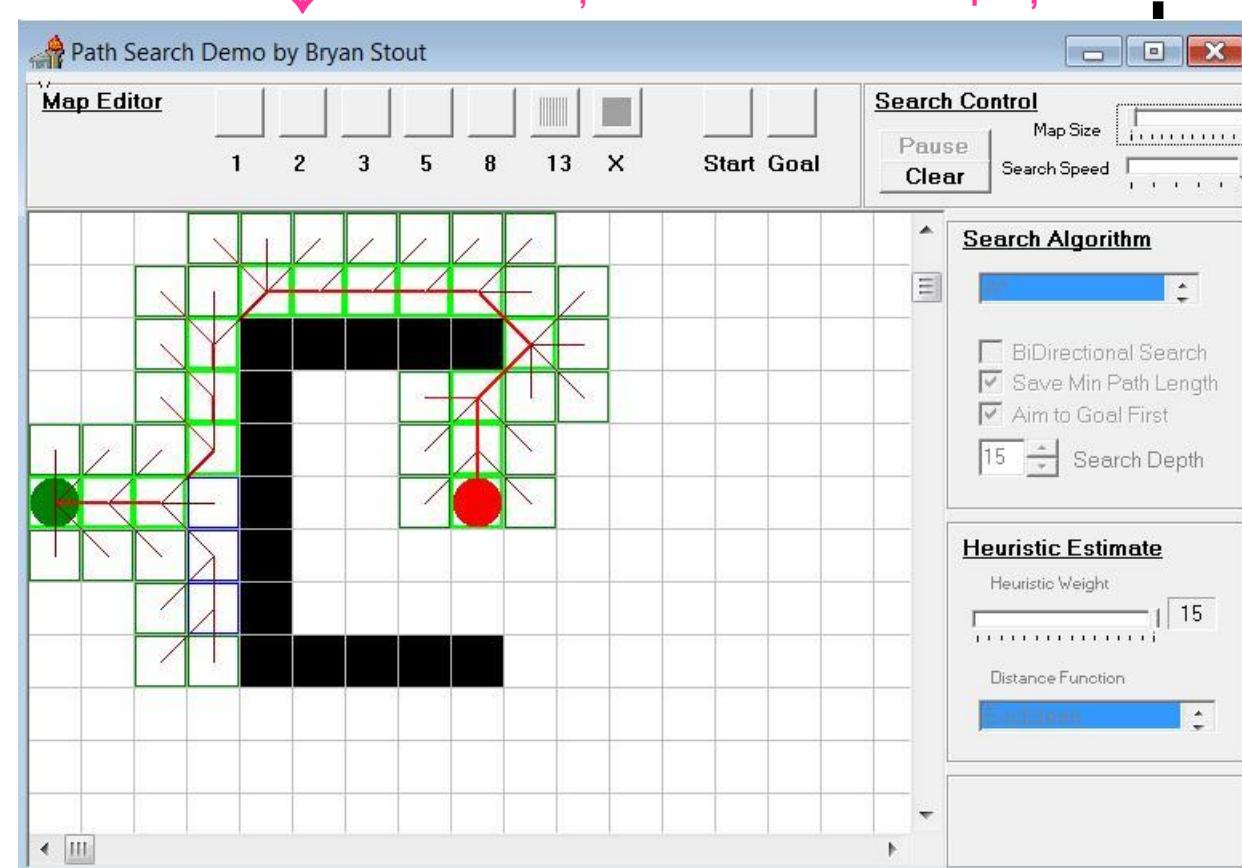
Arad	366
Craiova	160
Drobeta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Drumul optim Arad-București ($h(n)$ = distanță în linie dreaptă până la București, $g(n)$ = distanță parcursă)

Problemă



← A* – Distanța Manhattan – 12 pași



Cum se explică???



Algoritmul A* - completitudine și optimalitate (1)

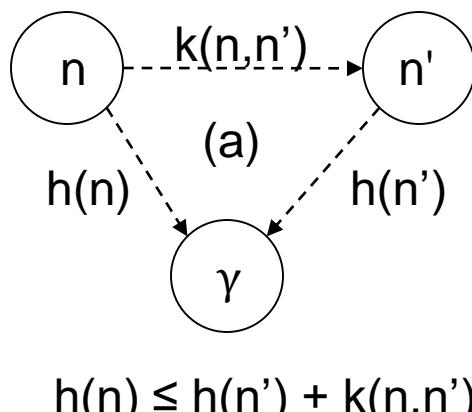
- **Teorema 7.1:** Algoritmul A* este **complet** chiar dacă graful explorat nu este finit.
- **Lema 7.1:** Fie $P = n_0, n_1, \dots, n_m$ un drum oarecare în graful explorat de A^* , astfel încât la un moment T al explorării toate nodurile din P sunt în CLOSED. Atunci, la orice moment de timp egal sau superior lui T , există inegalitatea $g(n_i) \leq g_p(n_i)$, $i = 0, m$:
 - Costul nodurilor din CLOSED poate să scadă, dar de fiecare dată când acest lucru se întâmplă, se pierde timp → scoaterea nodului din CLOSED, punerea în OPEN, prelucrarea acestuia încă o dată → trebuie evitate aceste situații → alegerea unei euristici cât mai bune care să minimizeze numărul acestor actualizări!

Algoritmul A* - completitudine și optimalitate (2)

- Definiție 7.2: Funcția heuristică h este **admisibilă** dacă pentru orice nod n din spațiul stărilor $h(n) \leq h^*(n)$. Cu alte cuvinte, o **heuristică admisibilă** h este **optimistă** și $h(\gamma) = 0$ pentru orice nod $\gamma \in \Gamma$.
- Teorema 7.2: Algoritmul A* ghidat printr-o **heuristică admisibilă** descoperă **soluția optimă** dacă există soluții.

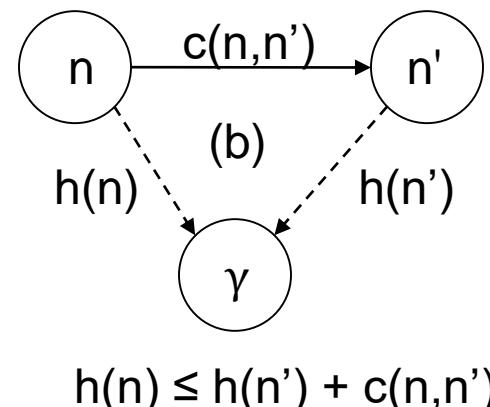
Euristici – consistență și monotonie

- Definiție 7.4: O euristică h este **consistentă** dacă pentru oricare două noduri n și n' ale grafului explorat, astfel încât n' este accesibil din n , există inegalitatea: $h(n) \leq h(n') + k(n,n')$, unde $k(n,n')$ este costul unui drum optim de la n la n' .
- Definiție 7.5: O euristică h este **monotonă** dacă pentru oricare două noduri n și n' ale grafului explorat, astfel încât n' este succesorul lui n , există inegalitatea $h(n) \leq h(n') + c(n,n')$, unde $c(n,n')$ este costul arcului (n,n') .



Regula
triunghiului
pentru euristici:

← Consistență
→ Monotone



$h(n) \leq h(n') + c(n,n')$

Consistență = monotonie

- Teorema 7.5: O heuristică este consistentă \Leftrightarrow este monotonă.

- Demonstrație:

- h – consistentă $\rightarrow h$ – monotonă. Alegem $n' \in \text{succs}(n) \rightarrow k(n,n') = c(n,n') \rightarrow h(n) \leq h(n') + c(n,n') \rightarrow h$ – monotonă.
- h – monotonă $\rightarrow h$ – consistentă. Fie $n = n_1, n_2, \dots, n_q = n'$, un drum optim $n..n'$ cu cost $k(n,n')$. $\rightarrow h(n) = h(n_1) \leq h(n_2) + c(n_1, n_2) \leq h(n_3) + c(n_1, n_2) + c(n_2, n_3) \dots \leq h(n_q) + c(n_1, n_2) + c(n_2, n_3) + \dots + c(n_{q-1}, n_q) = h(n_q) + k(n_1, n_q) \rightarrow h(n) \leq h(n') + k(n,n')$ $\rightarrow h$ – consistentă.

Consistență → admisibilitate

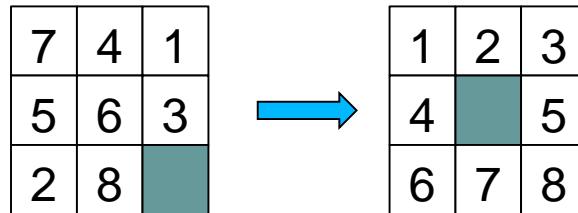
- Teorema 7.6: O euristică consistentă este admisibilă.
 - Demonstrație:
 - Fie h o euristică consistentă $\rightarrow h(n) \leq h(n') + k(n, n')$, $\forall n'$ accesibil din n . Fie $n' = \gamma \in \Gamma \rightarrow k(n, \gamma) = \min \{ k(n, \gamma') \mid \gamma' \in \Gamma \} = h^*(n) \rightarrow h(n) \leq h(\gamma) + h^*(n)$, dar $h(\gamma) = 0 \rightarrow h(n) \leq h^*(n) \rightarrow$ euristică admisibilă.
- Corolar 7.2: O euristică monotonă este admisibilă.

Dominanță - Definiții și teoremă

- **Definiție 7.6:** Fie h_1 și h_2 două euristici admisibile. Spunem că h_1 este **mai informată** decât h_2 dacă $h_2(n) < h_1(n)$ pentru orice nod $n \notin \Gamma$ din graful spațiului de stare explorat.
- **Definiție 7.7:** Un algoritm A_1^* **domină** un algoritm A_2^* dacă orice nod expandat de A_1^* este expandat și de A_2^* . (eventual, A_2^* expandează noduri suplimentare față de A_1^* , deci A_1^* poate fi mai rapid ca A_2^* .)
- **Teorema 7.11:** Dacă o euristică monotonă h_1 este mai informată decât o euristică monotonă h_2 , atunci un algoritm A_1^* condus de h_1 domină un algoritm A_2^* condus de h_2 .

Dominanță - Exemplu

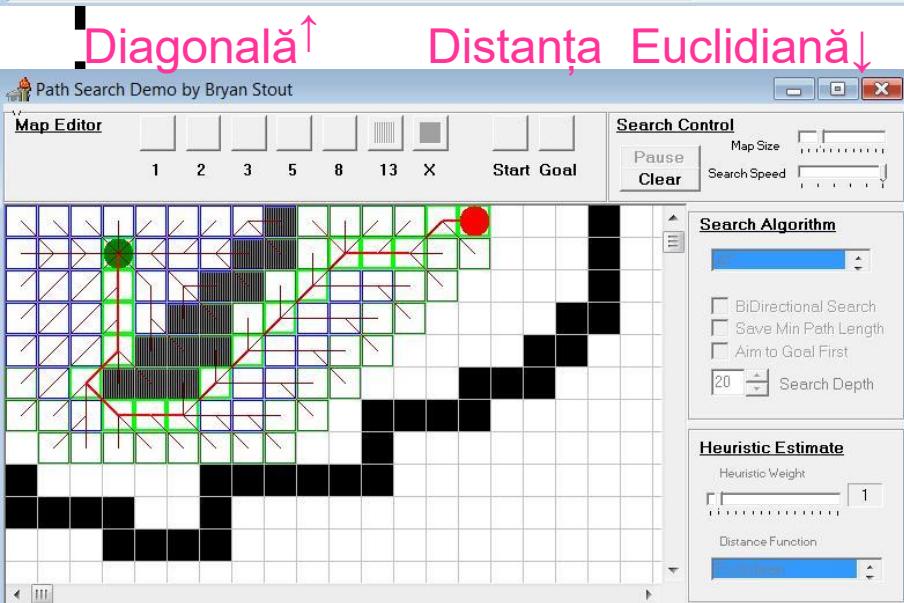
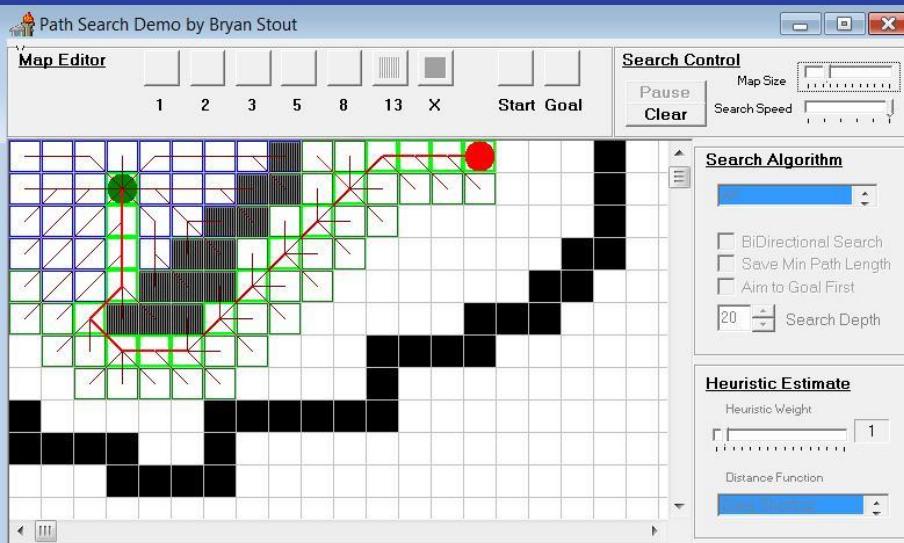
- Considerăm jocul 8-pătrățele care trebuie aranjat pornind de la forma inițială prin mutarea locului 'liber' astfel încât să ajungem la forma finală:



- Două euristici posibile:
 - h_1 = numărul pătrățelor a căror poziție curentă diferă de poziția finală;
 - $h_1 = \sum_{p \in \text{piese}} (\delta_p)$, unde $\delta_p = 0$ dacă poziția curentă coincide cu cea finală și $\delta_p = 1$, altfel
 - h_2 = distanța Manhattan = suma distanțelor pe verticală și orizontală între pozițiile curente ale pătrățelor și pozițiile lor finale
 - $h_2 = \sum_{p \in \text{piese}} (\text{dist_h}_p + \text{dist_v}_p)$

Admisibilitate? Monotonie? Dominanță? Care euristică va fi aleasă pentru A*?

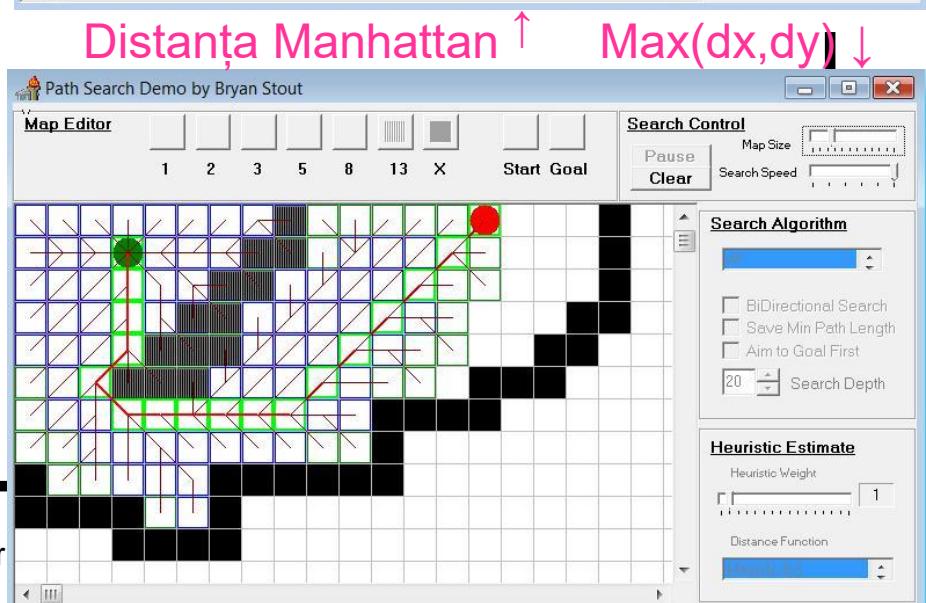
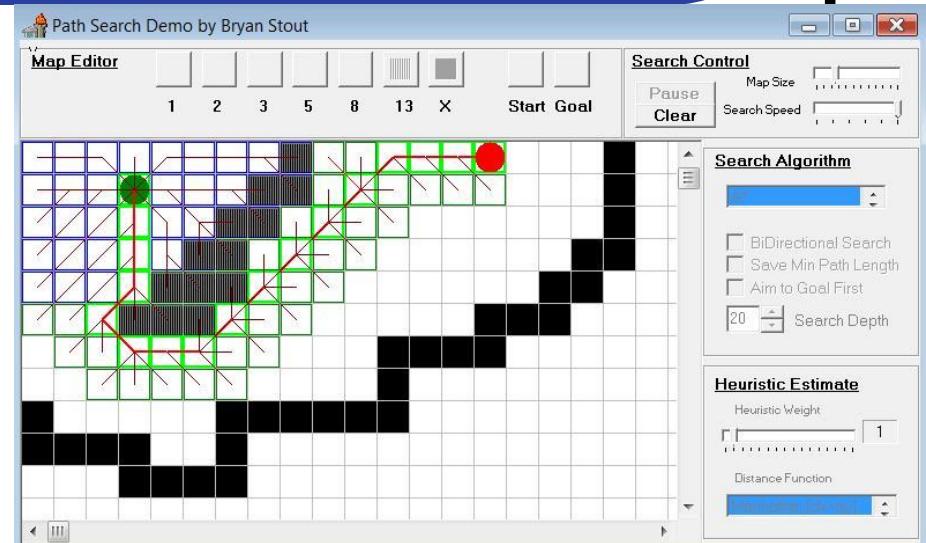
Exemplu A* cu diverse euristici



Do
mi
na
nț
ă

ce
uri
sti
ci

Algor



Complexitate A*

- Liniară dacă $|h^*(n) - h(n)| \leq \delta$, unde $\delta \geq 0$ este o constantă.
- Subexponențială, dacă $|h^*(n) - h(n)| \leq O(\log(h^*(n)))$.
- Exponențială, altfel, (dar mult mai bună decât a căutărilor neinformate).
- Mai multe explicații găsiți în Giumele 7.4.4!

ÎNTRERBĂRI?

Bibliografie

[1] C. Giu male – Introducere in Analiza Algoritmilor – cap. 6.1

[2] Cormen – Introducere in algoritmi – cap. 8.3

[3]
<http://www.soe.ucsc.edu/classes/cmps102/Spring04/TantaloAsymp.pdf>

[4] <http://www.mersenne.org/>

Proiectarea Algoritmilor

Curs 14 - Algoritmi aleatorii

Bibliografie

- [1] C. Giu male – Introducere in Analiza Algoritmilor – cap. 6.1
- [2] Cormen – Introducere in algoritmi – cap. 8.3
- [3] [http://classes.soe.ucsc.edu/cmps102/
Spring04/TantaloAsymp.pdf](http://classes.soe.ucsc.edu/cmps102/Spring04/TantaloAsymp.pdf)
- [4] <http://www.mersenne.org/>

Obiective

- Definirea conceptului de algoritm aleatoriu
- Algoritmi Las Vegas
- Algoritmi Monte Carlo
- Analiza algoritmilor aleatorii

Algoritmi aleatorii

- Micșorăm timpul de rezolvare a problemei relaxând restricțiile impuse soluțiilor.
- Determinarea soluției optime:
 - Renunțăm la optimalitate (soluția suboptimală are o marjă de eroare garantată prin calcul probabilistic).
- Găsirea unei singure soluții:
 - Găsim o soluție ce se apropie cu o probabilitate măsurabilă de soluția exactă.

Algoritmi Las Vegas

- Găsesc soluția corectă a problemei, însă **timpul de rezolvare** nu poate fi determinat cu exactitate.
- Creșterea timpului de rezolvare → creșterea probabilității de terminare a algoritmului (găsirea soluției optime).
- **Timp = ∞** → algoritmul se termină sigur (soluția e optimă).
- Probabilitatea de găsire a soluției crește extrem de repede astfel încât să se determine soluția corectă într-un timp suficient de scurt.

Algoritmi Monte Carlo

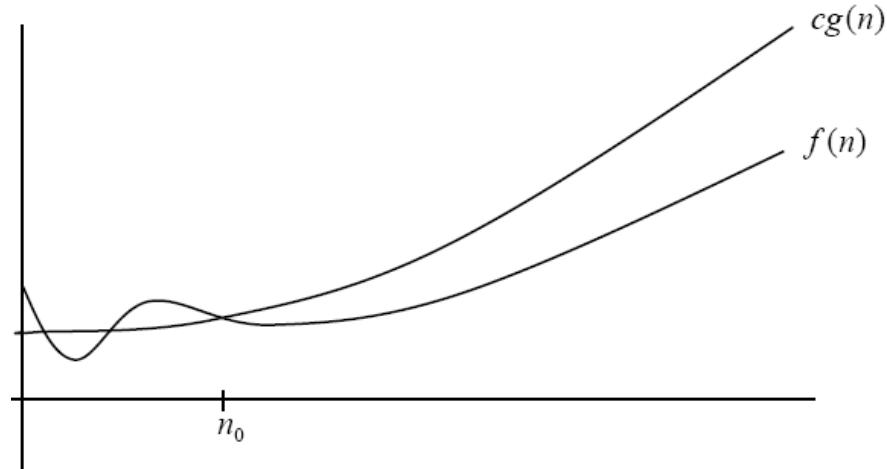
- Găsesc o soluție a problemei care nu e garantat corectă (soluție aproximativă).
- $\text{Timp} = \infty \rightarrow$ soluția corectă a problemei.
- Probabilitatea ca soluția să fie corectă crește o dată cu timpul de rezolvare.
- Soluția găsită într-un timp acceptabil este aproape sigur corectă.

Reminder – complexitatea algoritmilor

$$O(g(n)) = \{ f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) \leq cg(n) \}$$

$f(n) = O(g(n))$ says:

- $g(n)$ - limită **asimptotică superioară** pentru $f(n)$



<http://classes.soe.ucsc.edu/cmps102/Spring04/TantaloAsymp.pdf>

Complexitatea algoritmilor Las Vegas

- Definiția 6.1: Algoritmii Las Vegas au complexitatea $f(n) = O(g(n))$ dacă $\exists c > 0$ și $n_0 > 0$ a.î. $\forall n \geq n_0$ avem $0 < f(n) < c \alpha g(n)$ cu o probabilitate de cel puțin $1 - n^{-\alpha}$ pentru $\alpha > 0$ fixat și suficient de mare.

Complexitate algoritmi Monte Carlo

- Definiția 6.1': Algoritmii Monte Carlo au complexitatea $f(n) = O(g(n))$ dacă $\exists c > 0$ și $n_0 > 0$ astfel încât:
 - $\forall n \geq n_0, 0 < f(n) \leq c \alpha g(n)$ cu o probabilitate de cel puțin $1 - n^{-\alpha}$ pentru $\alpha > 0$ fixat și suficient de mare;
 - Probabilitatea ca soluția determinată de algoritm să fie corectă este cel puțin $1 - n^{-\alpha}$.

Exemplu algoritm Las Vegas

• Problemă:

- Capitolele unei cărți sunt stocate într-un fișier text sub forma unei secvențe nevide de linii;
- Fiecare secvență este precedată de o linie contor ce indică numărul de linii din secvență;
- Fiecare linie din fișier este terminată prin CR, LF;
- Toate liniile din secvență au aceeași lungime;
- Fiecare secvență de linii conține o linie (titlul capitolului) ce se repetă și care apare în cel puțin 10% din numărul de linii al secvenței.
- Secvențele sunt lungi.☺

• Cerință:

- Pentru fiecare secvență de linii să se tipărească titlul capitolului (linia care se repetă).

Rezolvare “clasică”

Detectează_linii(fisier)

• Pentru fiecare Secv ∈ fisier

• Pentru i de la 0 la dim(Secv)

• Pentru j de la i + 1 la dim(Secv)

• Dacă linie(i,Secv) = linie(j,Secv) atunci

• Întoarce (linie(i,Secv));

prelucrare
secvență

Complexitate – $O(\dim(\text{Secv})^2)$

Algoritm Las Vegas pentru rezolvarea problemei

- Secțiunea “prelucrare secvență” se înlocuiește cu următoarea funcție:
- Selectie_linii(n,secv) // n = dim secv
 - Cât timp(1) // mereu
 - i = random(0,n-1) // selectez o linie
 - j = random(0,n-1) // și încă una
 - Dacă i != j și linie(i,Secv) = linie(j,Secv) atunci // le compar
 - Întoarce linie(i,Secv) // am găsit linia

Analiza algoritmului Las Vegas (I)

- **Notății:**

- n = numărul de linii din secvența curentă;
- q = ponderea liniei repetate în secvență;
- r = numărul de apariții al liniei repetate: $r = n * q / 100$;
- m = numărul de pași necesari terminării algoritmului;
- P_k = probabilitatea ca la pasul k să fie satisfăcută condiția de terminare a algoritmului;
- $P(m)$ = probabilitatea ca algoritmul să se termine după m pași.

Analiza algoritmului Las Vegas (II)

- Probabilitatea ca la pasul k linia i să fie una din liniile repetitive este r / n .
- Probabilitatea ca la pasul k linia j să fie una din liniile repetitive (diferită de i) este $(r - 1) / n$.
- Condiția de terminare: cele 2 evenimente trebuie să se producă simultan:

$$P_k = r / n * (r-1) / n = q / 100 * (q / 100 - 1 / n)$$

Analiza algoritmului Las Vegas (III)

- Probabilitatea ca algoritmul să NU se termine după m pași:
 - $\prod_{k=1 \rightarrow m} (1 - P_k) = \prod_{k=1 \rightarrow m} [1 - q / 100 * (q / 100 - 1 / n)] = [1 - q / 100 * (q / 100 - 1 / n)]^m$
- $\rightarrow P(m) = 1 - [1 - q / 100 * (q / 100 - 1 / n)]^m$
- Pp: $n > 100$; $q > 10\%$
- $\rightarrow P(m) \geq 1 - [1 - q * (q - 1) / 10.000]^m$

Comparație timp de rulare

- $q = 10\%:$
 - 3500 pași $P = 1;$
 - 1000 pași – $P = 0,9988.$
- $q = 20\%:$
 - 1000 pași $P = 1.$
- $q = 30\%:$
 - 500 pași $P = 1.$
- Varianta clasică: cazul cel mai defavorabil – 10000 pași.

Complexitate algoritm Las Vegas

- Algoritmii Las Vegas au complexitatea $f(n) = O(g(n))$ dacă $\exists c > 0$ și $n_0 > 0$ a.i. $\forall n \geq n_0$ avem $0 < f(n) < c \alpha g(n)$ cu o probabilitate de cel puțin $1 - n^{-\alpha}$ pentru $\alpha > 0$ fixat și suficient de mare.
- Arătăm că $f(n) = O(\lg(n))$:
 - Notăm: $a = 1 - q * (q - 1) / 10.000$; $\rightarrow P(m) \geq 1 - a^m$
 - $P(c \alpha \lg(n))$ = probabilitatea ca algoritmul să se termine în $c \alpha \lg(n)$ pași;
 - $P(c \alpha \lg(n)) \geq 1 - a^{c \alpha \lg(n)} = 1 - n^{c \alpha \lg(a)} = 1 - n^{-c \alpha \lg(1/a)}$ pentru că $0 < a < 1$;
 - Dacă alegem $c \geq \lg^{-1}(1/a)$ $\rightarrow P(c \alpha \lg(n)) \geq 1 - n^{-\alpha} \rightarrow$ algoritmul se termină în $\lg^{-1}(1/a) \alpha \lg(n)$ pași cu o probabilitate $\geq 1 - n^{-\alpha} \rightarrow$ (definiție) $f(n) = O(\lg(n))$.

Exemplu algoritm Monte Carlo

- **Problemă:** testarea dacă un număr n dat este prim.

- Rezolvare “clasică”:

Complexitate:

$O(\sqrt{n})$

- Prim-clasic(n)

- Pentru i de la 2 la \sqrt{n}

- Dacă $(n \bmod i == 0)$ întoarce fals;

- Întoarce adevărat

Determinarea numerelor prime - complexitate

- Observație: pentru numere mari – operațiile nu mai durează $O(1)$!
- → Estimăm numărul de operații în funcție de numărul de biți pe care este exprimat numărul.
- → Prim_clasic – $O(2^{k/2})$ unde $k = \text{nr. de biți ocupat de } n$.

Complexitate nesatisfătoare!

- “On **September 4, 2006**, in the same room just a few feet away from their last find, Dr. Curtis Cooper and Dr. Steven Boone's CMSU team broke their own world record, discovering the 44th known Mersenne prime, $2^{32,582,657}-1$. The new prime at 9,808,358 digits is 650,000 digits larger than their previous record prime found **last December.**”
- “On **April 12th (2009)**, the 46th known Mersenne prime, $2^{42,643,801}-1$, a 12,837,064 digit number was found by Odd Magnar Strindmo from Melhus, Norway! This prime is the second largest known prime number, a "mere" 141,125 digits smaller than the Mersenne prime found last August.”
- As of **October 2009**, 47 Mersenne primes are known. The largest known prime number ($2^{43,112,609}-1$) is a Mersenne prime. [Wikipedia]
- As of **October 2014**, 48 Mersenne primes are known. The largest known prime number $2^{57,885,161}-1$ is a Mersenne prime.
- **January 7, 2016:** the discovery of the largest known prime number, $2^{74,207,281}-1$. The prime number has 22,338,618 digits -- almost 5 million digits longer than the previous record prime number

Algoritm aleatoriu (I)

- **Teorema 6.1 (mica teoremă a lui Fermat):** Dacă n este prim $\rightarrow \forall 0 < x < n, x^{n-1} \bmod n = 1$.
- **Prim1(n, α)** // detectează dacă n e număr prim
 - **Dacă** ($n \leq 1$ sau $n \bmod 2 = 0$) **Întoarce** fals
 - **Limit** = **limită_calcul(n, α)** // numărul minim de pași pentru // soluția corectă cu $P = 1 - n^{-\alpha}$
 - **Pentru i de la 0 la limit**
 - $x = \text{random}(1, n-1)$ // aleg un număr oarecare
 - **Dacă** ($\text{pow_mod}(x, n) \neq 1$) **Întoarce** fals // testez teorema // Fermat
 - **Întoarce** adevărat

Complexitate?

Algoritm aleatoriu (II)

- Pow_mod(x,n) // calculează $x^{n-1} \text{ mod } n$
 - $r = 1$ // restul
 - Pentru m de la $n-1$ la 0
 - Dacă $(m \text{ mod } 2 \neq 0)$ // testează dacă puterea e pară // sau nu
 - $r = x * r \text{ mod } n$
 - $x = (x * x) \text{ mod } n$ // calculează $x^2 \text{ mod } n$
 - $m = m \text{ div } 2$ // înjumătățesc puterea
 - Întoarce r

Complexitate:
 $O(\lg(n))$

Algoritm aleatoriu (III)

- **Problemă:** nu putem stabili cu exactitate care este **limita de calcul**:
 - Nu se poate estima pentru un număr compus n numărul de numere x , $2 < x < n$ pentru care nu se verifică ecuația;
 - Există numere compuse pentru care orice număr $x < n$ și prim în raport cu n satisfac ecuația lui Fermat (ex: nr. Carmichael $\rightarrow 561$).
- \rightarrow Nu știm cu exactitate câte numere sunt!
 \rightarrow Nu putem calcula probabilitatea!

Altă variantă de algoritm aleatoriu

- **Teorema 6.2:** Pentru orice număr prim n ecuația $x^2 \bmod n = 1$ are exact 2 soluții:

$$x_1 = 1 \quad \text{și} \quad x_2 = n - 1.$$

- **Definiție 6.2:** Fie $n > 1$ și $0 < x < n$ două numere astfel încât $x^{n-1} \bmod n \neq 1$ sau $x^2 \bmod n = 1$, $x \neq 1$ și $x \neq n - 1$. X se numește **mărtor al divizibilității lui n** .

Algoritmul Miller-Rabin

- Prim2(n, α)
 - Dacă ($n \leq 1$ sau $n \bmod 2 = 0$) **Întoarce fals**
 - $\text{limit} = \text{limita_calcul}(n, \alpha)$
 - **Pentru i de la 0 la limit**
 - $x = \text{random}(1, n-1)$
 - Dacă (`martor_div(x,n)`) **Întoarce fals**
 - **Întoarce adevărat**

Complexitate?

Algoritmul Miller-Rabin (II)

- martor_div(x,n) // determină dacă x este
// martor al divizibilității lui n

- $r = 1$; $y = x$;
- Pentru m de la $n-1$ la 0 // puterea
 - Dacă $(m \bmod 2 \neq 0)$ // putere impară
 - $r = y * r \bmod n$
 - $z = y$ // salvez valoarea lui x
 - $y = y * y \bmod n$ // calculez $y^2 \bmod n$
 - Dacă $(y = 1 \text{ și } z \neq 1 \text{ și } z \neq n-1)$ // verific teorema 6.2
 - Întoarce 1
 - $m = m \bmod 2$ // înjumătățesc puterea
 - Întoarce $r \neq 1$ // mica teoremă Fermat ($x^{n-1} \bmod n \neq 1$)

Complexitate:
 $O(\lg(n))$

Calcularea numărului de pași

- **Teorema 6.3:** Pentru orice număr n , impar și compus există cel puțin $(n-1) / 2$ martori ai divizibilității lui n .
- **Caz neinteresant:** număr prim pentru că oricum algoritmul întoarce adevărat ($P_{\text{corect}}(n) = 1$)!
- **Caz interesant:** număr compus (impar) ($P_{\text{corect}}(n) = ?$):
 - x = element generat la un pas al algoritmului ($0 < x < n$);
 - $P(x)$ = probabilitatea ca numărul x generat din cele $n-1$ posibilități să fie martor al divizibilității;
 - $P(x) \geq (n-1) / 2 * 1 / (n-1) = 0.5$;
 - $P_{\text{incorrect}}(n) = \prod_{1 \rightarrow \text{limit}} (1 - P(x)) \leq 1/2^{\text{limit}}$;
 - $\rightarrow P_{\text{correct}}(n) \geq 1 - 2^{-\text{limit}} = 1 - n^{-\alpha} \rightarrow \text{limit} = \alpha \lg(n)$; \rightarrow după $\alpha \lg(n)$ pași $P_{\text{correct}}(n) \geq 1 - n^{-\alpha}$;
 - \rightarrow **Complexitate: $O(\lg^2(n))$** \rightarrow în funcție de **numărul de biți k** \rightarrow **Complexitate: $O(k^2)$**

Exemplu de utilizare practică

- Quicksort(A, st, dr)
 - Dacă $st < dr$
 - $q \leftarrow \text{Partiție}(A, st, dr)$
 - Quicksort($A, st, q - 1$)
 - Quicksort($A, q + 1, dr$)

Cazul
defavorabil?

- Partiție(A, st, dr)
 - $x \leftarrow A[st]$
 - $i \leftarrow st - 1$
 - Pentru j de la st la $dr - 1$
 - Dacă $A[j] \leq x$
 - $i \leftarrow i + 1$
 - Interschimbă $A[i] \leftrightarrow A[j]$
 - Interschimbă $A[i + 1] \leftrightarrow A[dr]$
 - Întoarce $i + 1$

Complexitate
?

Exemplu de utilizare practică (II)

- Problema Quicksort – **cazul defavorabil** – datele de intrare sunt sortate în ordine inversă.
- **Complexitate Quicksort: $O(n^2)$.**
- Folosind algoritmi aleatorii eliminăm acest caz.

Quicksort-aleatoriu

- Quicksort-Randomizat(A , st , dr)
 - Dacă $st < dr$
 - $q \leftarrow$ Partiție-Randomizată(A , st , dr)
 - Quicksort-Randomizat(A , st , $q - 1$)
 - Quicksort-Randomizat(A , $q + 1$, dr)
- Partiție-Randomizată(A , st , dr)
 - $i \leftarrow$ Random(st , dr)
 - **Interschimbă** $A[dr] \leftrightarrow A[i]$
 - **Întoarce** Partiție(A , st , dr)

INTREBĂRI?