Proiectarea Algoritmilor

Curs 10 – Arbori minimi de acoperire (continuare)

Rețele de flux. Flux maxim.

Bibliografie

[1] C. Giumale – Introducere în Analiza Algoritmilor - cap. 5.5 si 5.6

[2] Cormen – Introducere în algoritmi - cap. Arbori de acoperire minimi si Flux Maxim (24 si 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

Arbori minimi de acoperire (reminder)

Arbori minimi de acoperire – Definiții

- Fie G = (V,E) graf neorientat și conex, iar w: E → $\Re$ o funcție de cost (w(u,v) = costul muchiei (u,v)).

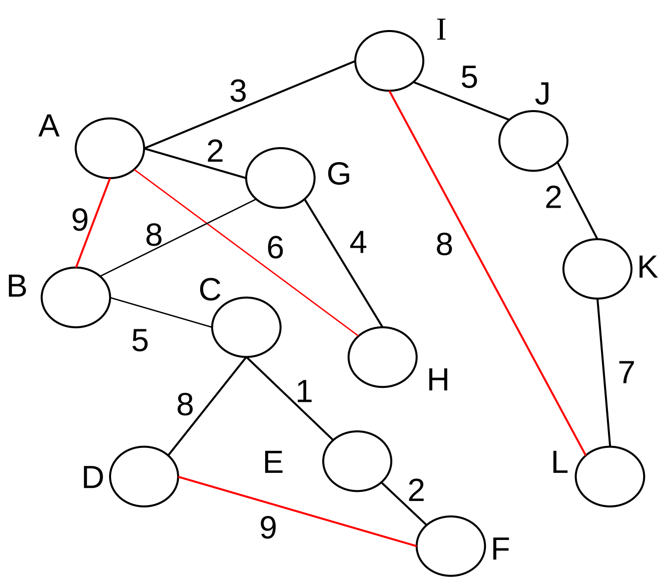
- Definiție: Un arbore liber al lui G este un graf neorientat conex și aciclic Arb = (V',E'); V' $\subseteq$ V, E' $\subseteq$ E. Costul arborelui este: C(Arb) = Σ w(e), e $\in$ E'.

- 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 Arb $\in$ ARB(G) a.î. C(Arb) = min{C(Arb') | Arb' $\in$ ARB(G)}.

Proprietăți (I)

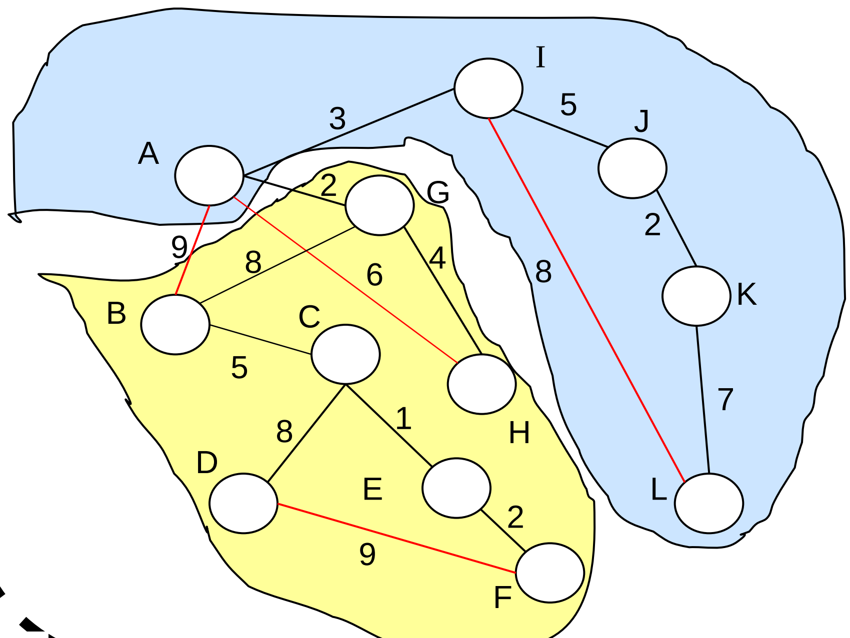
$G = (V,E)$, $C = (V',E')$ – ciclu în G; $e \in E'$ a.î. $w(e) = \max \{w(e') \mid e' \in E'\}$ => $e \notin Arb(G)$ unde Arb(G) = AMA în G.



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

Proprietăți (II)

G = (V,E), S = (V',E'), 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 ∉ Arb(G).
- Arb' = Arb(G) – e' + e (unde e' o muchie similară cu e).
- Arb'= arbore de acoperire.
- Cost(Arb') < Cost(Arb) → Arb nu este arbore minim.

AMA

- Bazați pe ideea de muchie sigură – se identifică o muchie sigură și se adaugă în AMA.

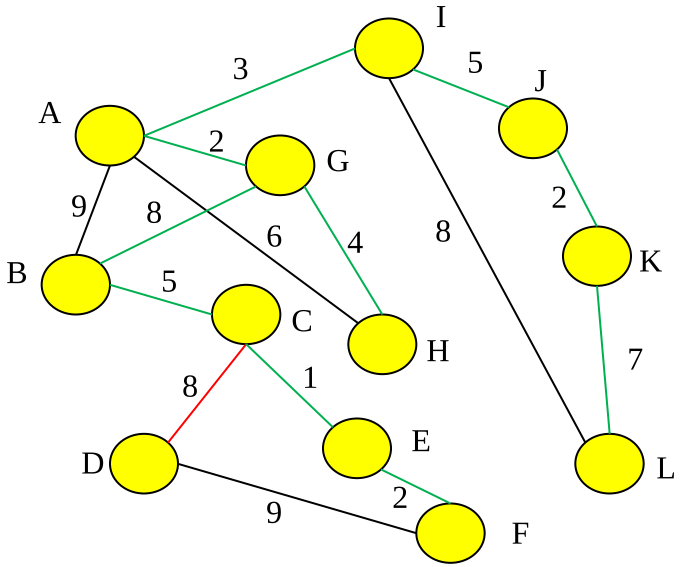
- 2 algoritmi de tip greedy:
  - Prim: se pornește cu un nod și se extinde pe rând cu muchiile cele mai ieftine care au un singur capăt în mulțimea de muchii deja formată (Proprietatea 2). Algoritmul este asemănător algoritmului Dijkstra.

  - 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

- Prim(G,w,s) Implementare în Java la [4] !
  - A = ∅ // AMA
  - **Pentru fiecare** (u ∈ V)
    - d[u] = ∞; p[u] = null // inițializăm distanța și părintele
  - d[s] = 0; // nodul de start are distanța 0
  - Q = constrQ(V, d); // ordonată după costul muchiei
    // care unește nodul de AMA deja creat
  - **Cât timp** (Q != ∅) // cât timp mai sunt noduri neadăugate
    - u = ExtrageMin(Q); // extrag nodul aflat cel mai aproape
    - A = A ∪ {(u,p[u])}; // adaug muchia în AMA
    - **Pentru fiecare** (v ∈ succs(u))
      - **Dacă** d[v] > w(u,v) **atunci**
        - d[v] = w(u,v); //+ d[u] // actualizăm distanțele și părinții nodurilor
        - p[v] = u; // adiacente care nu sunt în AMA încă
  - **Întoarce** A - {(s,p(s))} // prima muchie adăugată

Exemplu (XII)



- Q: Ø

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' = ∅. Adaug muchia (u,s), u = nod adiacent sursei aflat cel mai aproape de aceasta → din Propr. 2 → (u,s) ∈ Arb.
- $P_n \rightarrow P_{n+1}$:
  - S = (V',E') mulțimea vârfurilor și muchiilor adăugate deja în arbore înainte de a adăuga (u,p[u]).
  - p[u] ∈ V', u ∉ V'; (u,p[u]) are cost minim dintre muchiile care au un capăt în S (conform extrage minim)
  - din Propr. 2 → (u,p[u]) ∈ Arb

Corectitudine (II)

- 2. arătăm că muchiile ignorate 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. Arb(u) = Arb(v)
    - → (u,v) creează un ciclu în Arb(u) (arborii sunt aciclici) – fie ciclul format din u..x..v și (u,v).
    - w(u,v) = max {w(u',v') | (u',v') $\in$ 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 → (u,v) are costul maxim din ciclu
    - → din Propr. 1 → (u,v) $\notin$ Arb

Complexitate Prim

- Depinde de implementare (vezi Dijkstra)
  - Matrice de adiacență $O(V^2)$
  - Heap binar $O(ElogV)$
  - Heap Fibonacci $O(VlogV+E)$

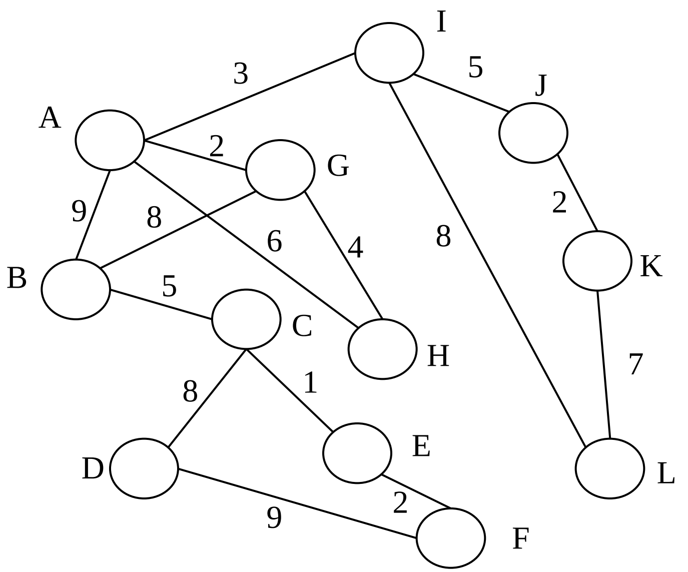
- Concluzii
  - Grafuri dese
    - Matrice de adiacență preferată
  - Grafuri rare
    - Heap binar sau Fibonacci

Algoritmul lui Kruskal

Implementare în Java la [4] !

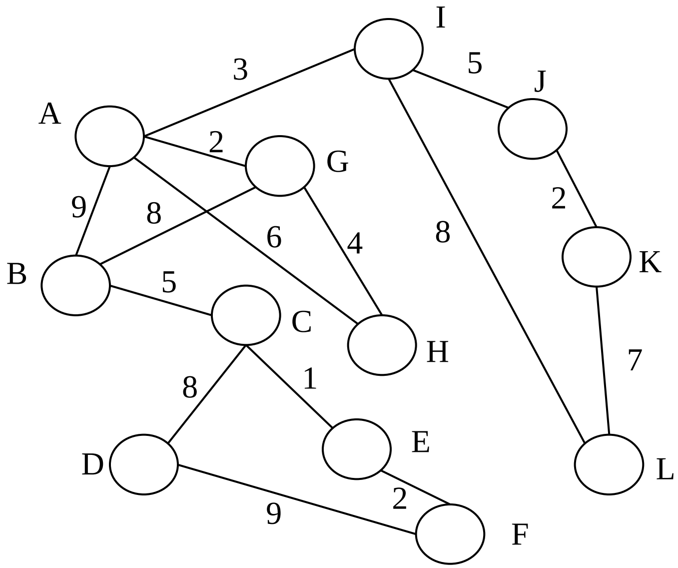
- Kruskal(G,w)
  - A = ∅; // AMA
  - **Pentru fiecare** (v ∊ V)
    - Constr_Arb(v) // creează o mulțime formată din nodul respectiv // (un arbore cu un singur nod)
  - Sortează_asc(E,w) // se sortează muchiile în funcție de // costul lor
  - **Pentru fiecare** ((u,v) ∊ E) // muchiile se extrag în ordinea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

Exemplu (I)

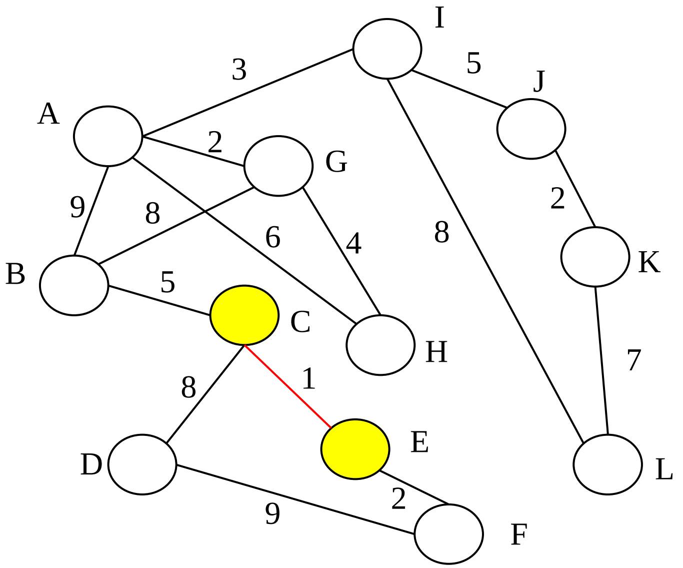


- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9

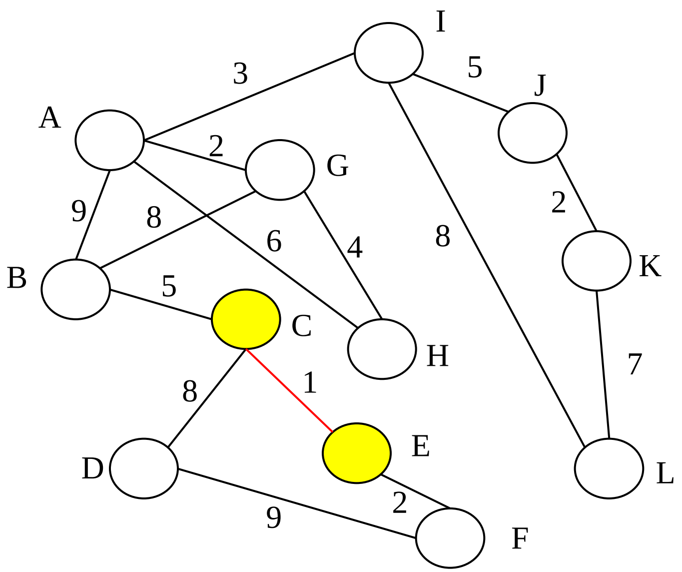
Exemplu (II)



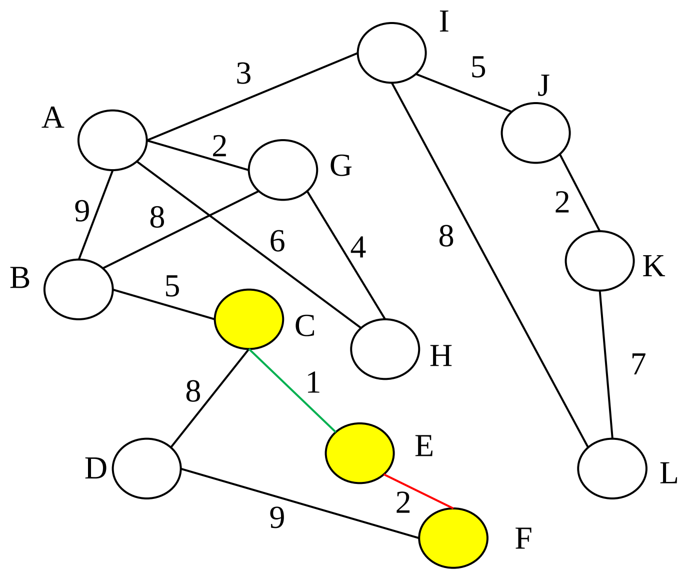
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



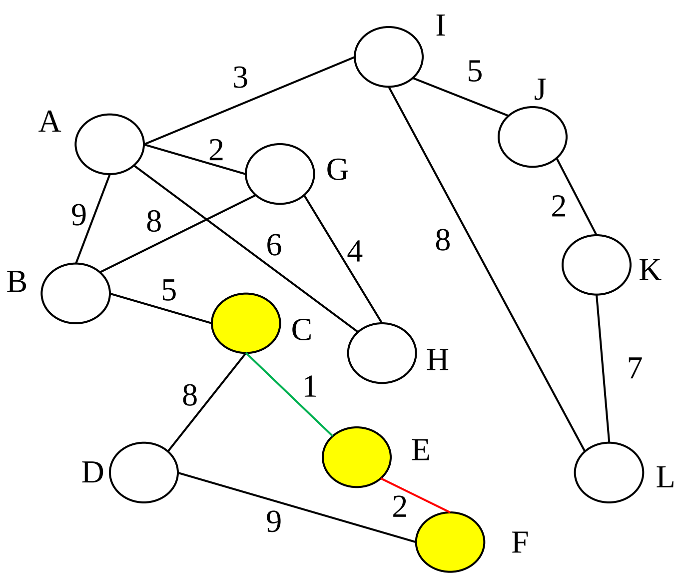
Exemplu (III)



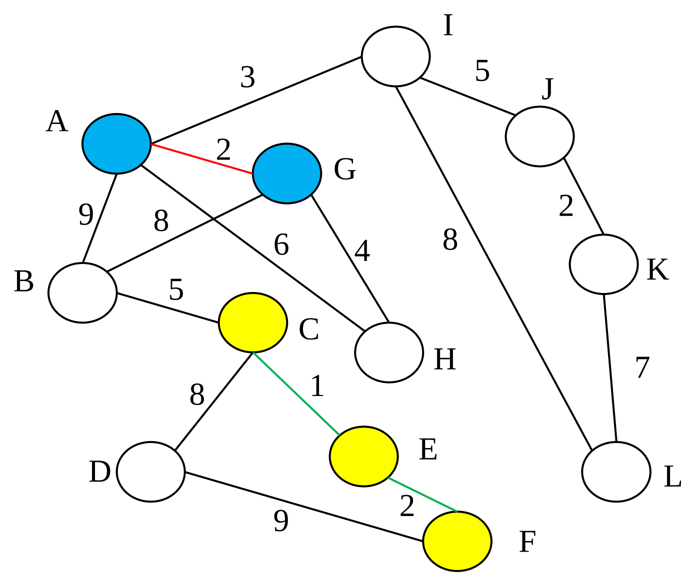
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



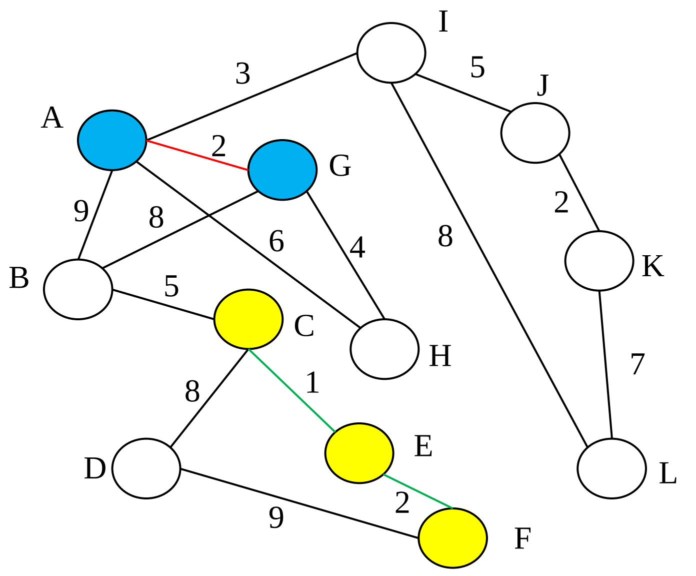
Exemplu (IV)



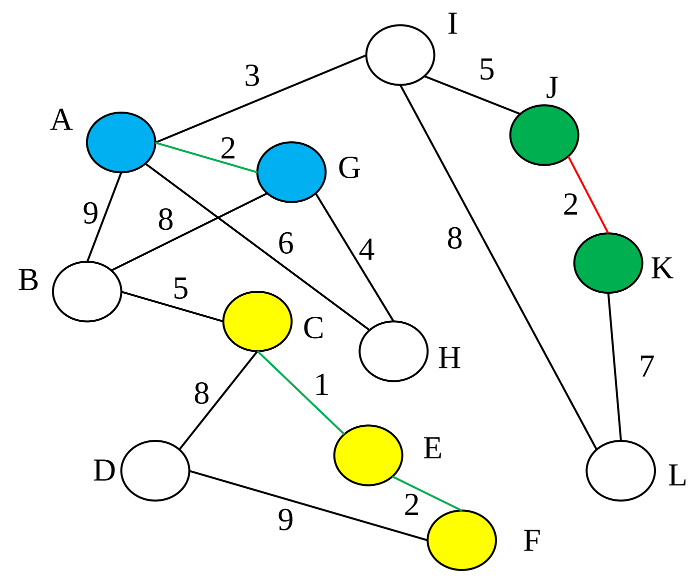
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



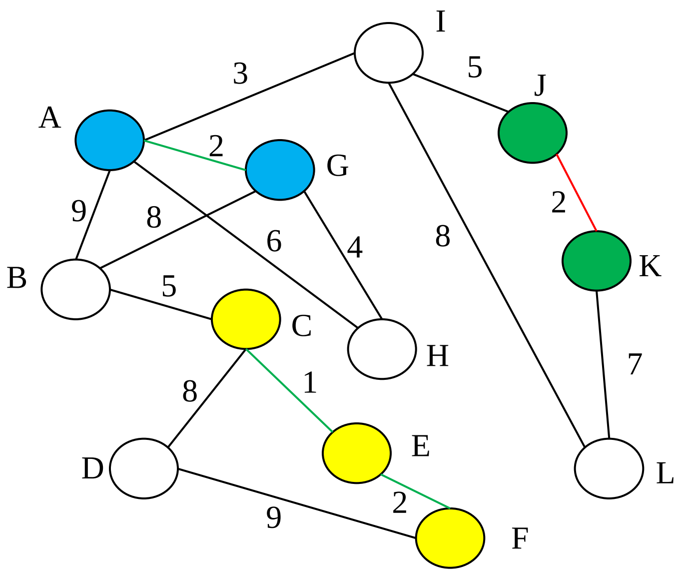
Exemplu (V)



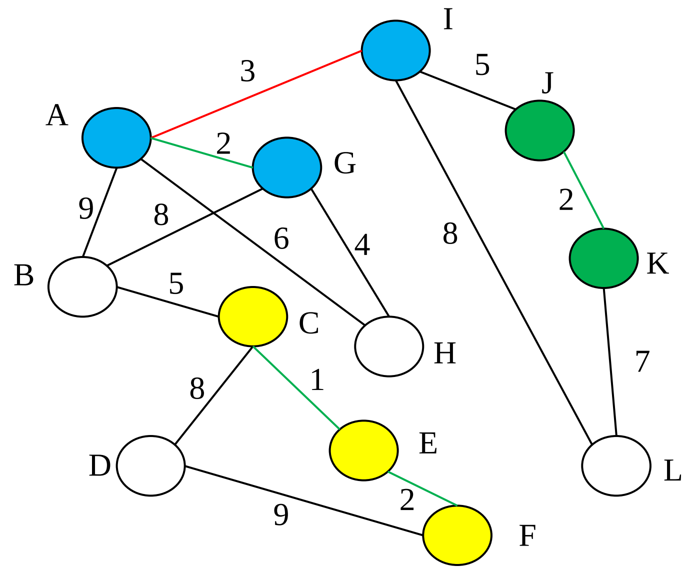
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



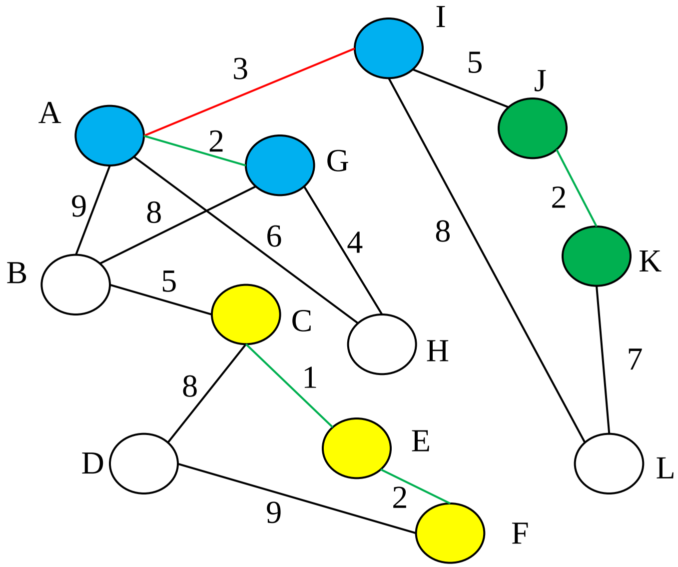
Exemplu (VI)



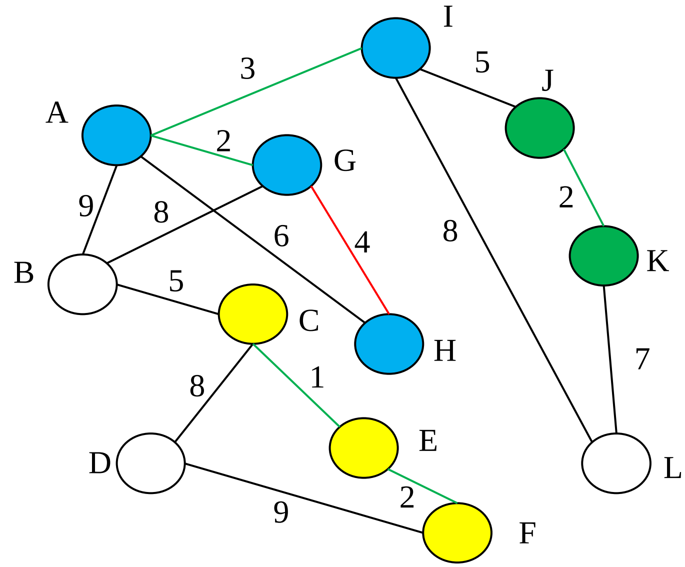
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



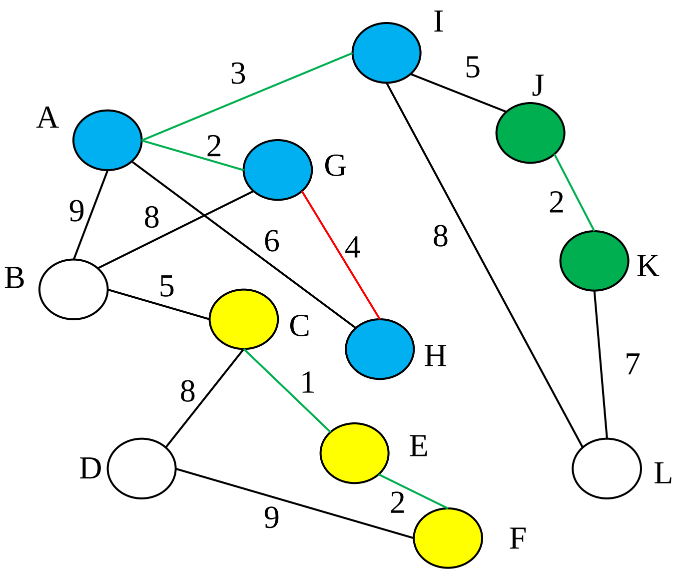
Exemplu (VII)



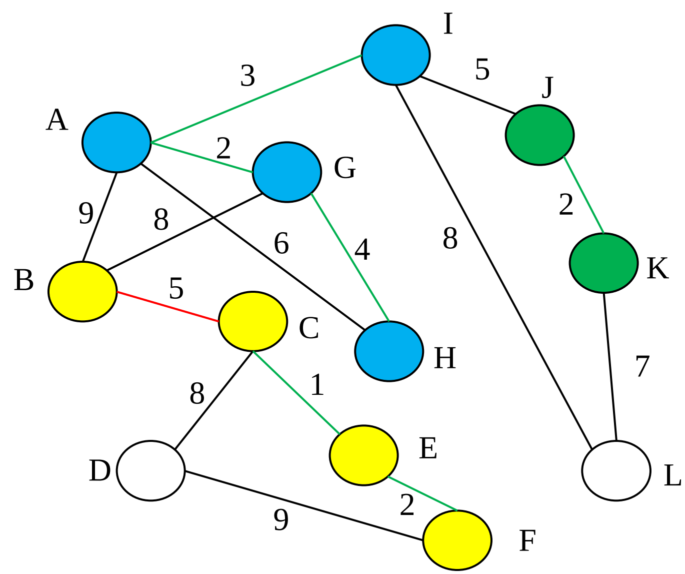
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



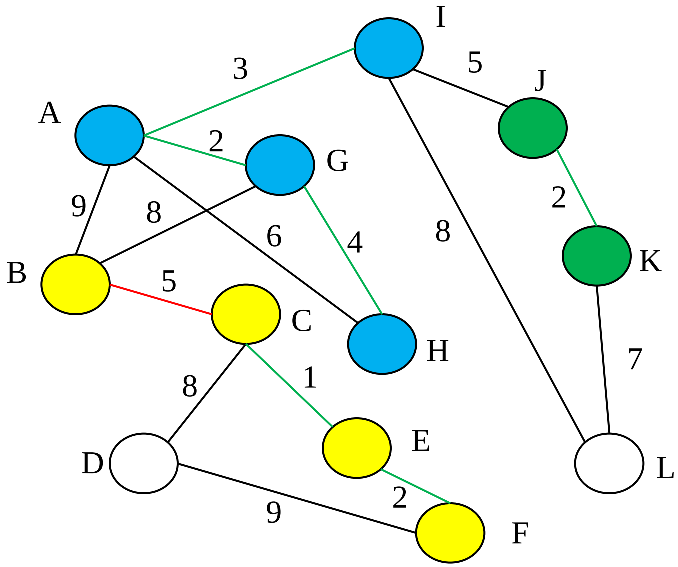
Exemplu (VIII)



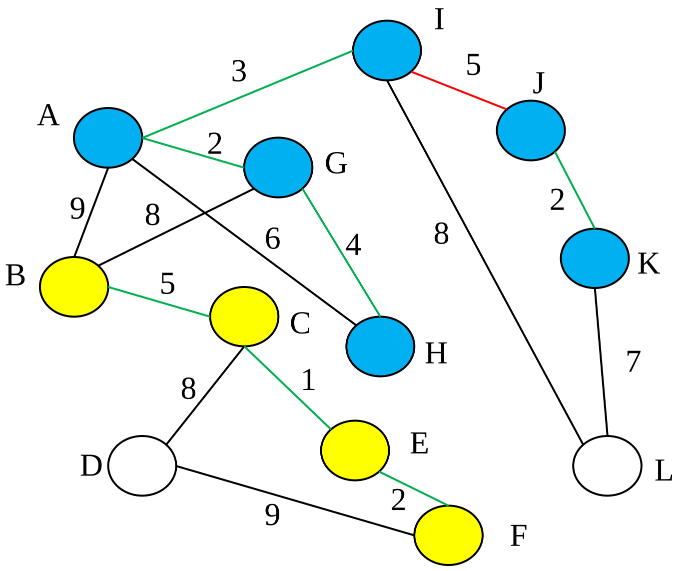
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



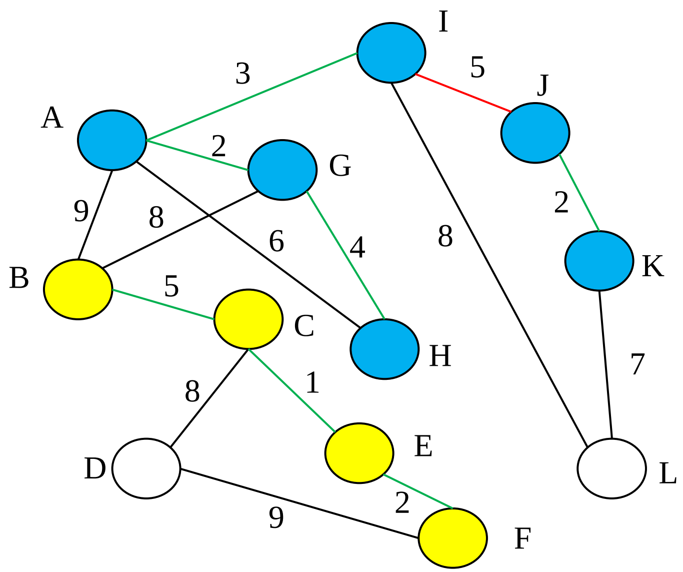
Exemplu (IX)



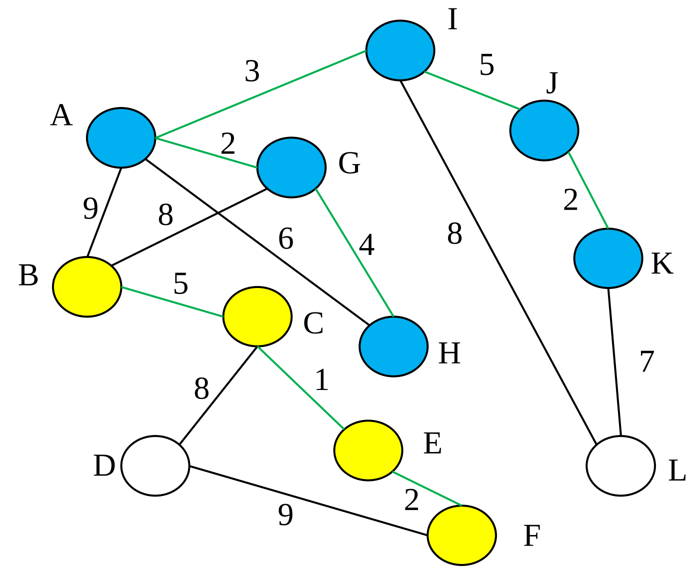
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



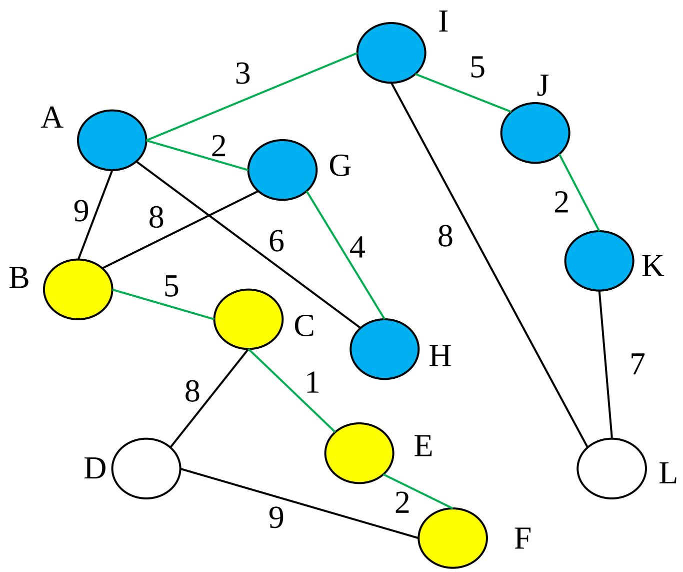
Exemplu (X)



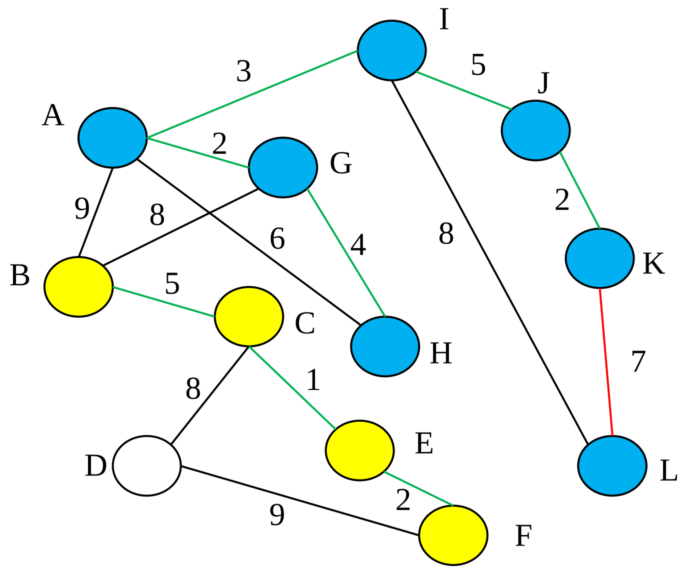
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



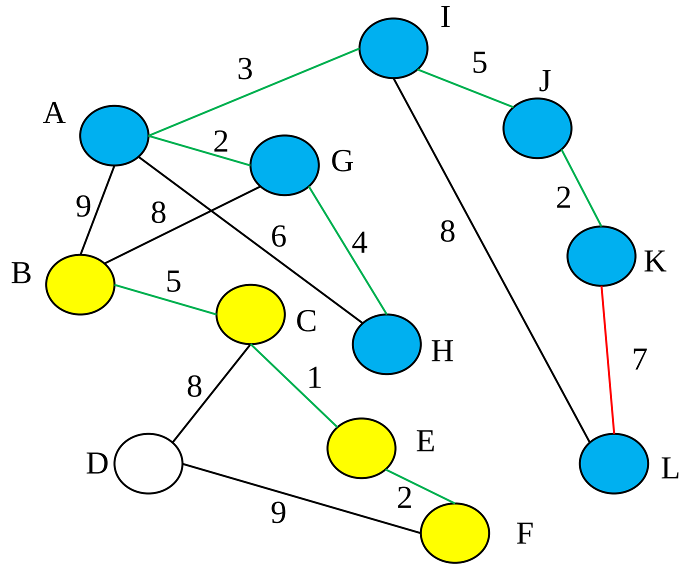
Exemplu (XI)



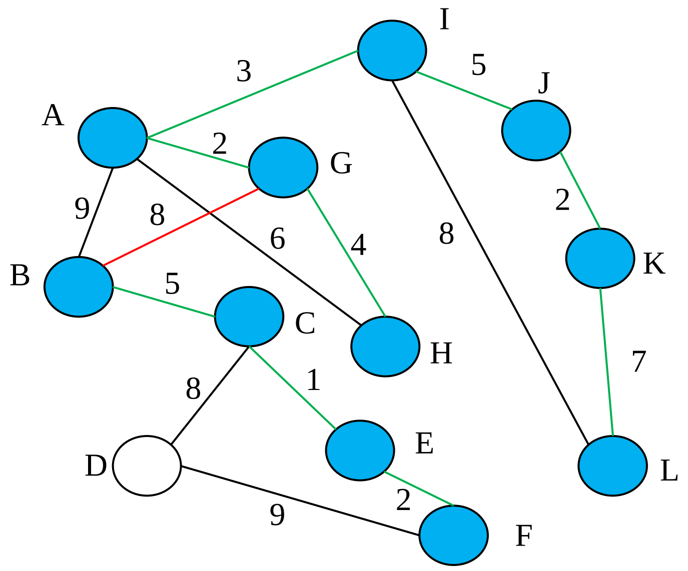
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



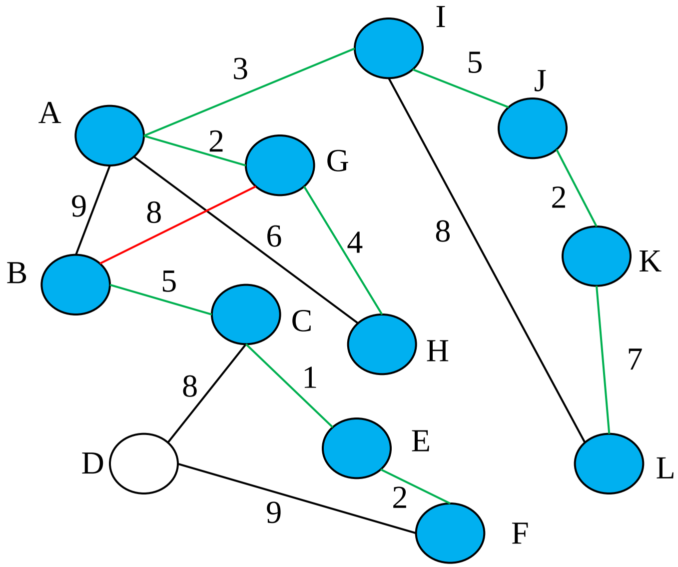
Exemplu (XII)



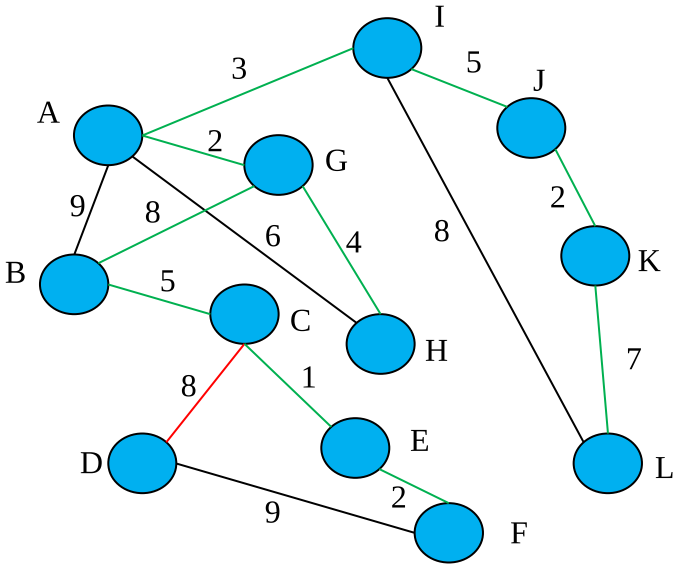
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



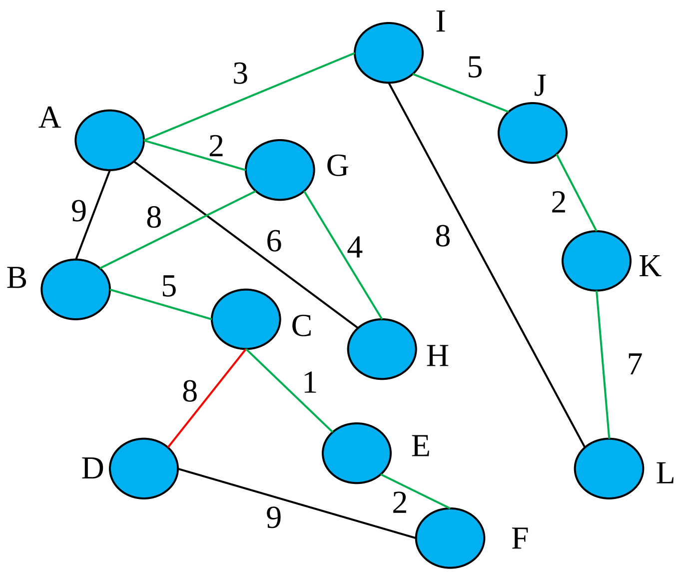
Exemplu (XIII)



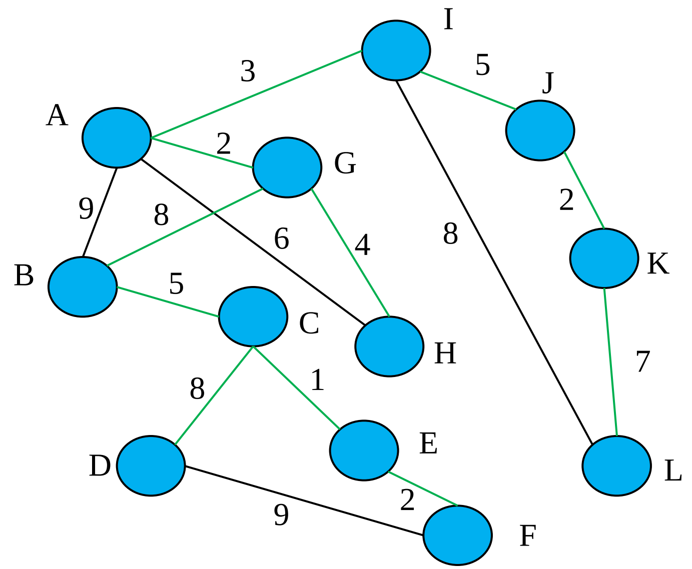
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



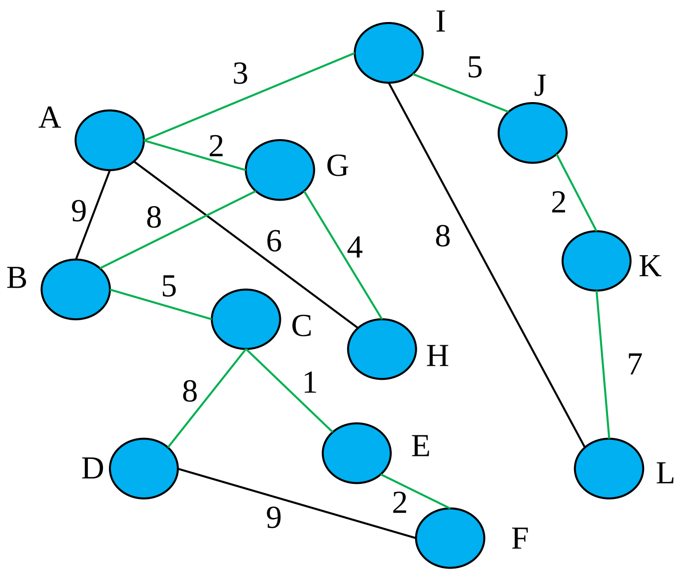
Exemplu (XIV)



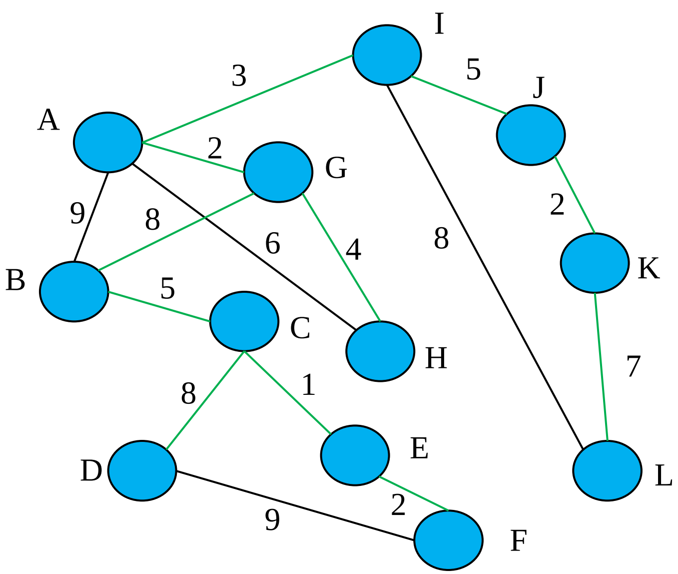
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



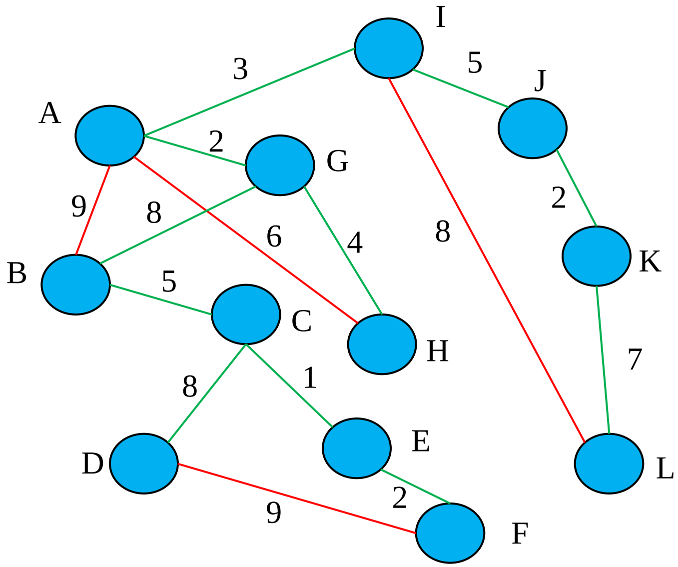
Exemplu (XV)

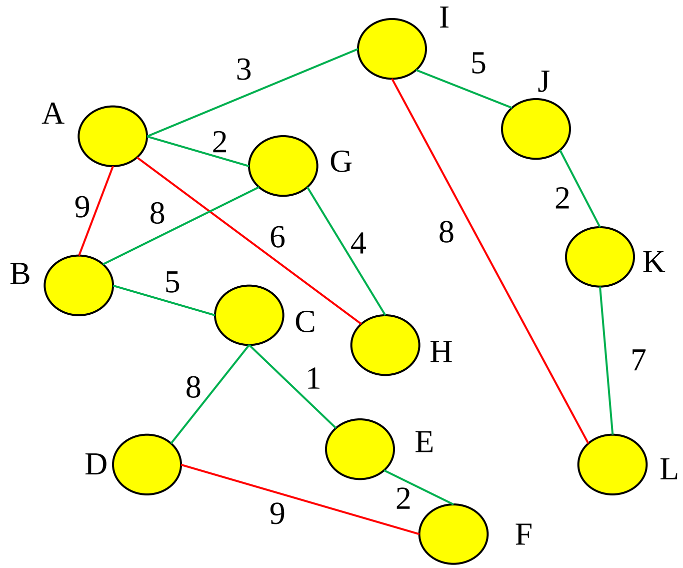


- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



Comparație Prim - Kruskal





Corectitudine (I)

- 1. arătăm că muchiile ignorate nu fac parte din Arb:
  - Pp. (u,v) a.î. Arb(u) = Arb(v)
    - → (u,v) creează un ciclu în Arb(u) (arborii sunt aciclici)
    - w(u,v) = max {w(u',v') | (u',v') $\in$ Arb(u)} (din faptul că muchiile sunt sortate crescător)
    - → din Propr. 1 → (u,v) $\notin$ 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') | (u',v') $\in$ E} → din Propr. 2 → (u,v) $\in$ Arb.
- $P_n \rightarrow P_{n+1}$:
  - Arb(u) != Arb(v)
  - → (u,v) muchie cu un capăt în Arb(u)
  - (u,v) are cel mai mic cost din muchiile cu un capăt în u (din faptul că muchiile sunt sortate crescător)
  - → din Propr. 2 → (u,v) $\in$ Arb

Algoritmul lui Kruskal

- Kruskal(G,w)
  - A = ∅; // AMA
  - **Pentru fiecare** (v ∊ 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) ∊ E) // muchiile se extrag în ordinea // costului
    - **Dacă** Arb(u) != Arb(v) **atunci** // verificăm dacă se creează ciclu
      - Arb(u) = Arb(u) ∪ Arb(v) // se reunesc mulțimile de noduri (arborii)
      - A = A ∪ {(u,v)} // se adaugă muchia sigură în AMA
  - **Întoarce** A

Complexitate?

Complexitate Kruskal

- Elementele algoritmului:
  - Sortarea muchiilor: O(ElogE) ≈ O(ElogV)
  - Arb(u) = Arb(v) – compararea a 2 mulțimi disjuncte {1,2,3} {4,5,6} – mai precis trebuie identificat dacă 2 elemente sunt în aceeași mulțime
  - Arb(u) ∪ Arb(v) – reuniunea a 2 mulțimi disjuncte într-una singură
- → depinde de implementarea mulțimilor disjuncte

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

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

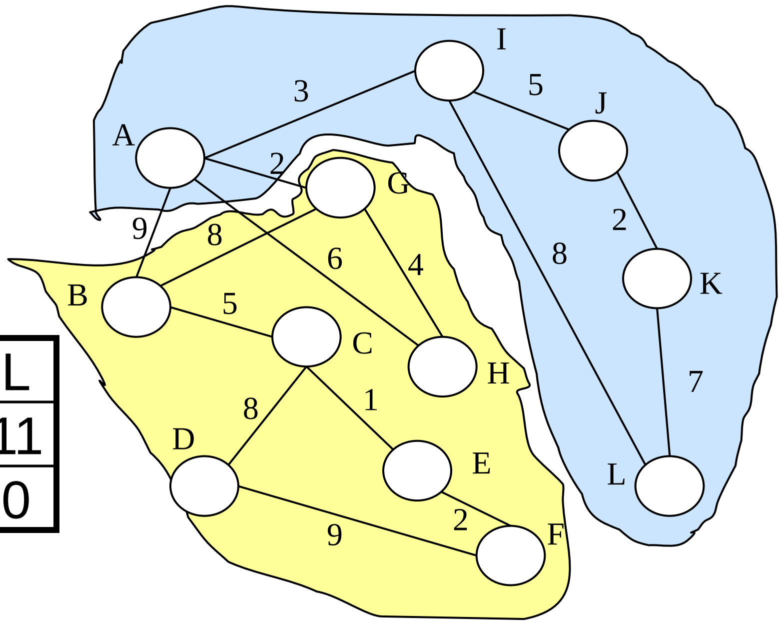
- Comparare (M_1, M_2)
  - **Pentru fiecare** ($u \in M_1$)
    - **Pentru fiecare** ($v \in M_2$)
      - **Dacă** ($u = v$) **întoarce true**
  - **Întoarce false**
- Complexitate: V^2

- Reuniune (M1, M2)
  - **Pentru** i **de la** length(M_1) **la** length(M_1) + 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 mulțimi disjuncte (Var. 2) – Regăsire Rapidă

- Mulțimile - 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
0	1	1	1	1	1	1	1	0	0	0	0

- Arb(u) != Arb(v)
  - Complexitate?
- Arb(u) = Arb(u) ∪ Arb(v)
  - Complexitate?

Complexitate maximă?

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 mulțimi

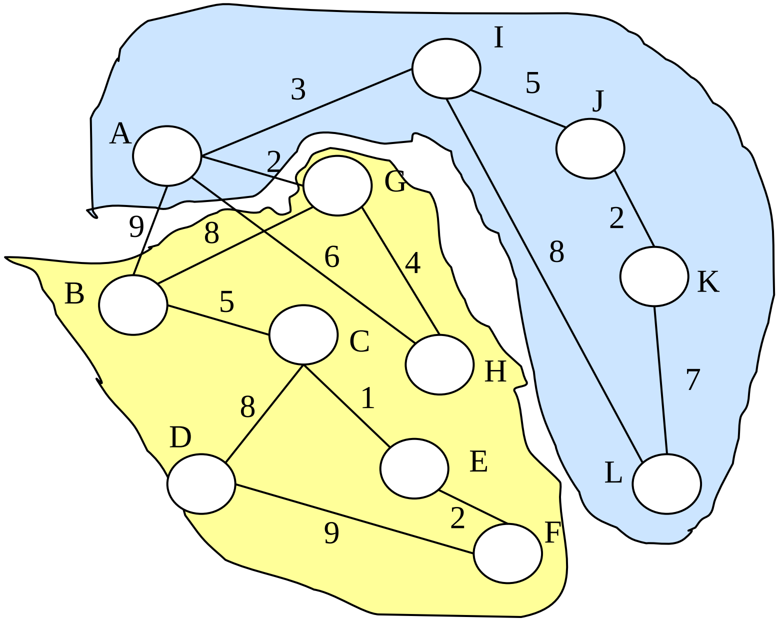
- Complexitate maximă
  - O(V * E) // E = numărul de reuniuni

- Inacceptabil pentru grafuri f mari

Variante de implementare mulțimi 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
8	1	1	2	2	4	1	6	8	8	9	10



- id[i] reprezintă părintele lui i
- pentru rădăcina arborelui id[i] = i

Variante de implementare mulțimi disjuncte – reuniune rapidă

- Comparare (u, v)
  - Verifică dacă 2 noduri au aceeași rădăcină;
  - Implică identificarea rădăcinii:
- Arb(u) // identificarea rădăcinii unei componente
  - **Cât timp** (i != id[i]) i = id[i];
  - **Întoarce** i
- Comparare (u, v)
  - **Întoarce** Arb(u) != Arb(v)
- Reuniune (u,v) // implică identificarea rădăcinii
  - v = Arb(v)
  - 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.

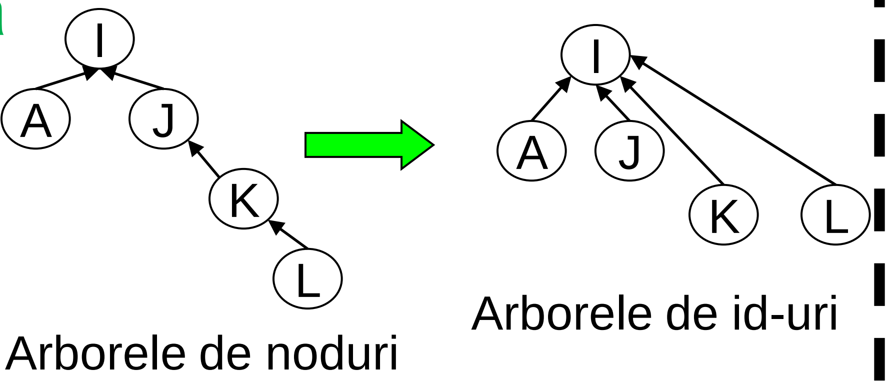
- Reuniunea – O(V) // implică regăsirea rădăcinii pentru a ști unde se face modificarea

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 != id[i])
      - id[i] = id[id[i]];
      - i = id[i];
    - **Întoarce** i

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

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

- Menține o înălțime redusă a arborilor.

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 lg pentru a ajunge la 1.
  - În practică este ≤ 5.

- → În practică $O(E)$

Complexitate Kruskal

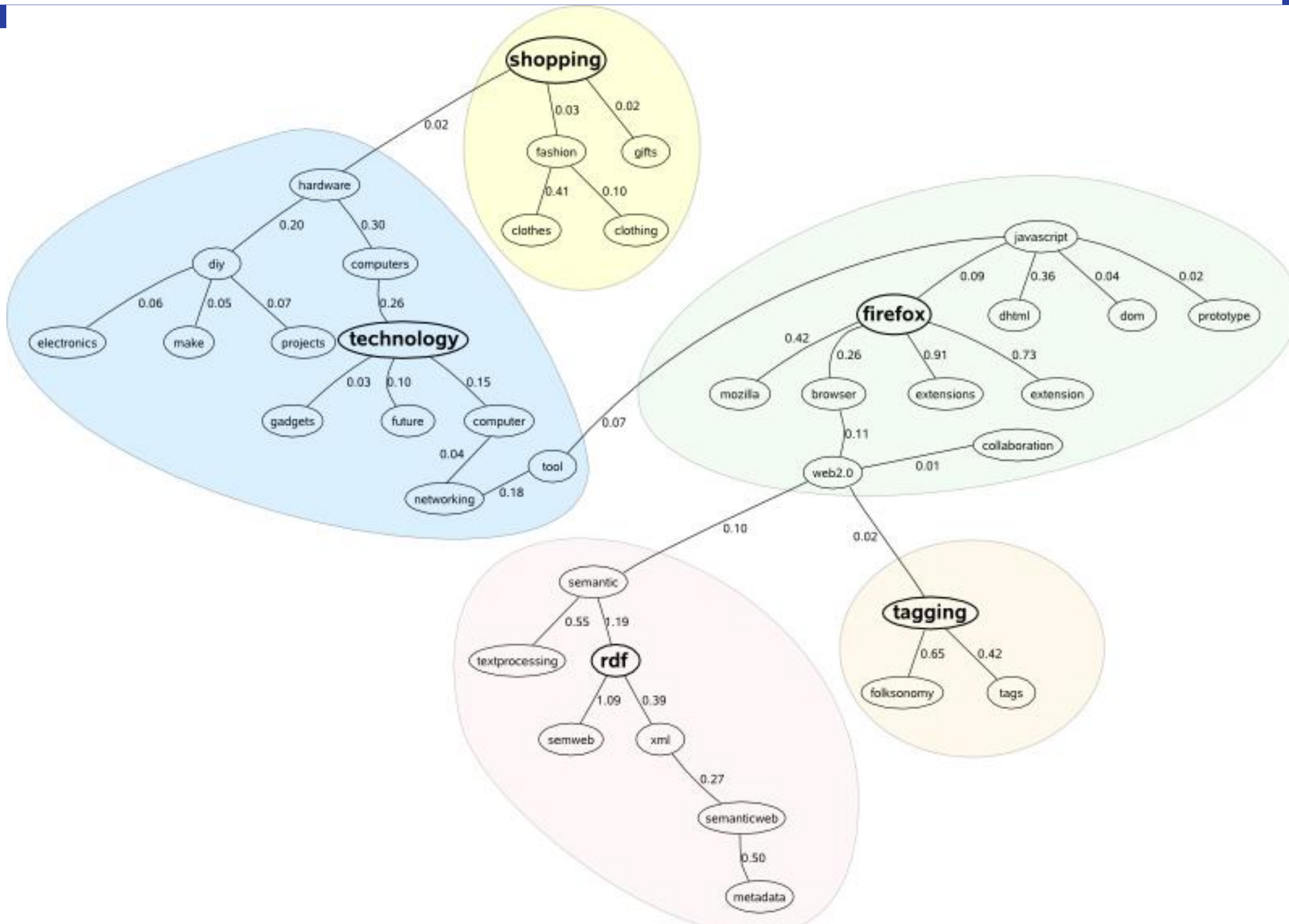
- Max (complexitate sortare, complexitate operații mulțimi) = max(O(ElogV), O(E)) = O(ElogV)

- → 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



shopping
0.02
0.03
0.02
fashion
gifts
0.41
0.10
clothes
clothing
hardware
0.20
0.30
diy
computers
0.06
0.05
0.07
0.26
electronics
make
projects
technology
0.03
0.10
0.15
gadgets
future
computer
0.04
tool
networking
0.18
0.07
javascript
0.09
0.36
0.04
0.02
firefox
dhtml
dom
prototype
0.42
0.26
0.91
0.73
mozilla
browser
extensions
extension
0.11
collaboration
web2.0
0.01
0.10
0.02
semantic
0.55
1.19
textprocessing
rdf
1.09
0.39
semweb
xml
0.27
semanticweb
0.50
metadata
tagging
0.65
0.42
folksonomy
tags

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

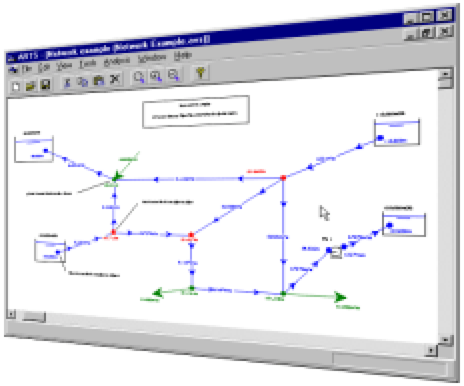
Rețele de flux. Flux maxim.

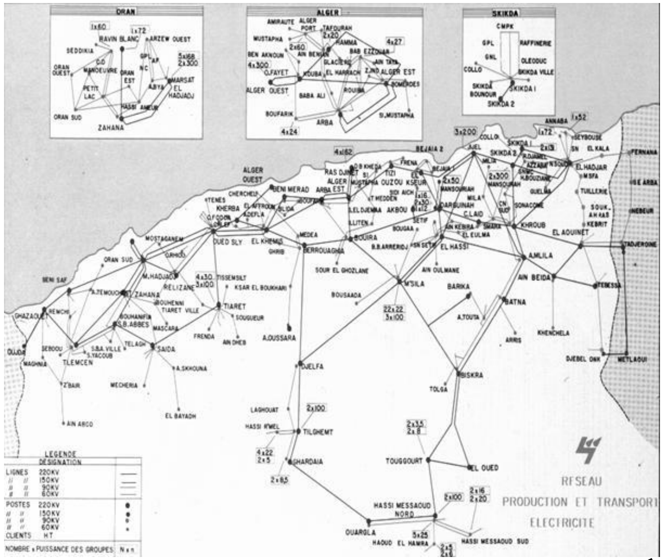
Obiective

- Definirea conceptului de rețea de flux (sau de transport).
- Identificarea principalilor 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.

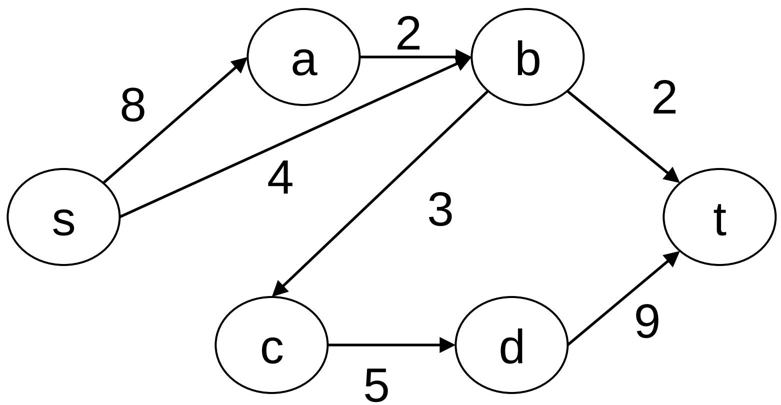




Rețea de flux – Definiție

- $G(V,E)$ graf orientat;
- $c(u,v) \geq 0 \ \ \forall(u,v)$ – 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

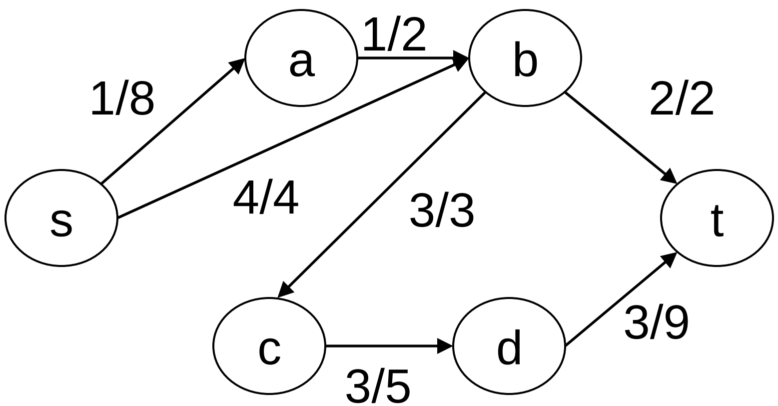


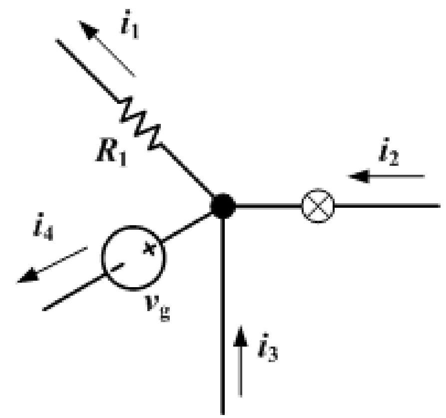
- s – sursa, t – destinația.
- Pe arce este reprezentată capacitatea arcului.

Flux. Definiție. Proprietăți.

- $G = (V,E)$ – rețea de flux;
- $c: V \times V \rightarrow \Re$ - capacitatea rețelei;
- $f: V \times V \rightarrow \Re$ - 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 capacității arcelor;
  - $\forall u, v \in V, f(u,v) = -f(v,u)$ – simetria fluxului;
  - $\Sigma f(u,v) = 0$ pentru $\forall u \in V \setminus \{s,t\}$ – conservarea fluxului.

Exemplu de fluxuri





i2+i3-i4-i1=0 (P3)

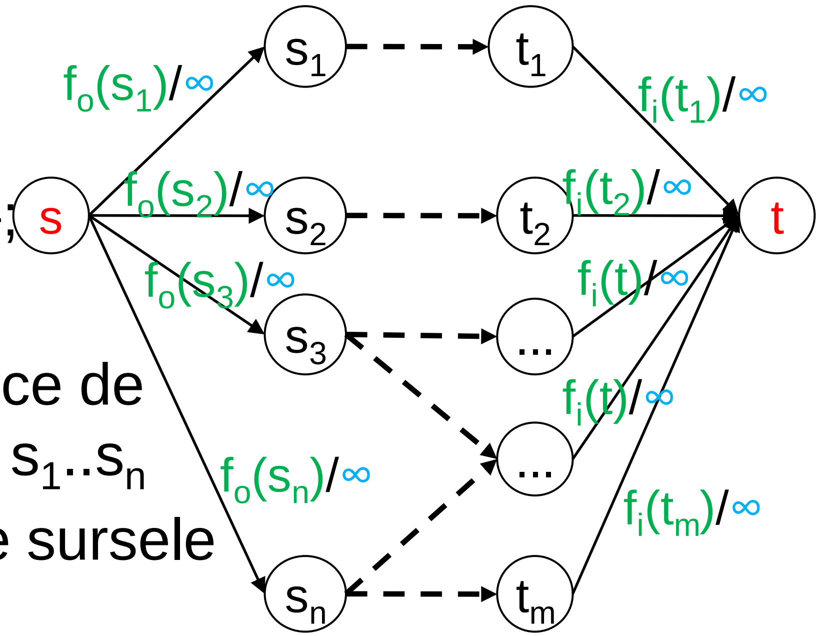
- $\Sigma f(u,v) = 0$ pentru $\forall u \in V \setminus \{s,t\}$ – fluxul se conservă;
- Proprietatea 3 = legea curentului (Kirchoff) ☺ - suma I. curenților ce intră într-un nod = suma I. curenților ce ies din nodul respectiv.

Flux. Notații.

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

Surse multiple, destinații multiple

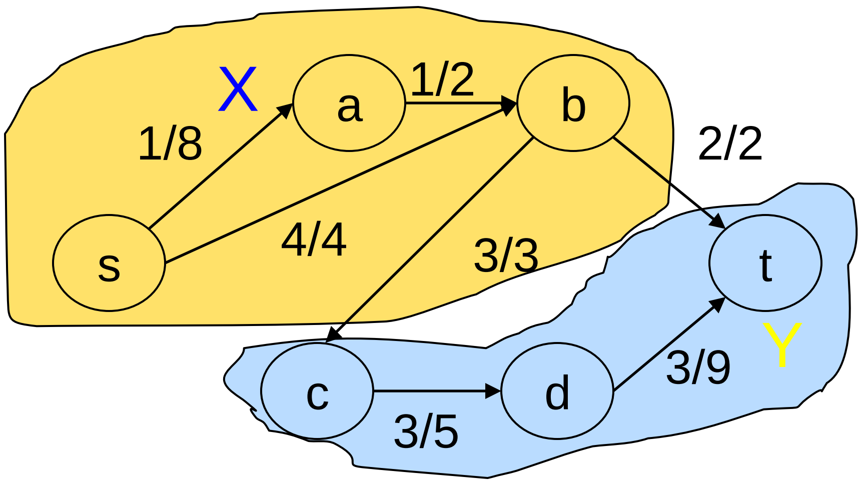
- Surse multiple $\{s_1, s_2, \ldots, s_n\}$;
- Destinații multiple $\{t_1, t_2, \ldots, t_m\}$;
- Se adaugă o sursă unică cu arce de capacitate infinită spre sursele $s_1..s_n$ și flux egal cu fluxul generat de sursele respective;
- Se adaugă o destinație unică t și arce de capacitate infinită între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



Operații cu fluxuri

- X,Y – mulțimi de noduri;
- $f(X,Y) = \Sigma_{x \in X} \Sigma_{y \in Y} f(x,y)$ = 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 $Y \subseteq X$:
    - $f(X \setminus Y, Z) = f(X,Z) - f(Y,Z)$;
    - $f(Z, X \setminus Y) = f(Z,X) - f(Z,Y)$;
  - $\forall X,Y,Z \in V$ 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)

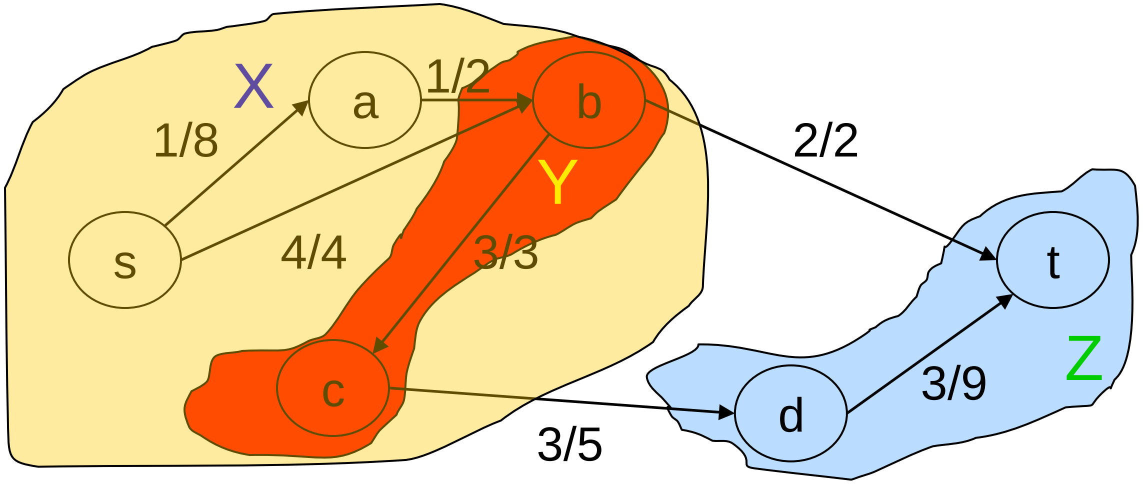


$f(X,Y) = \Sigma_{x \in X}\Sigma_{y \in Y}\ f(x,y)$

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

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

Exemplu operații fluxuri (2)

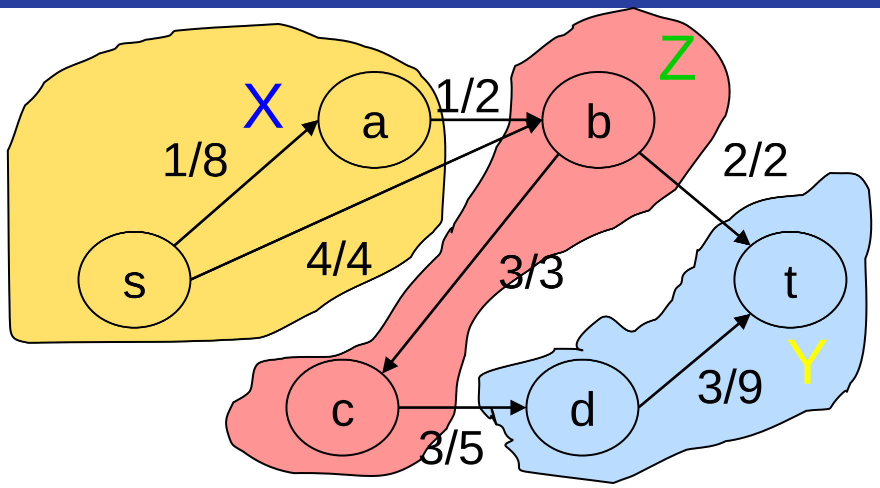


$\forall X, Y, Z \in V$ si $Y \subseteq X$

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

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

Exemplu operații fluxuri (3)

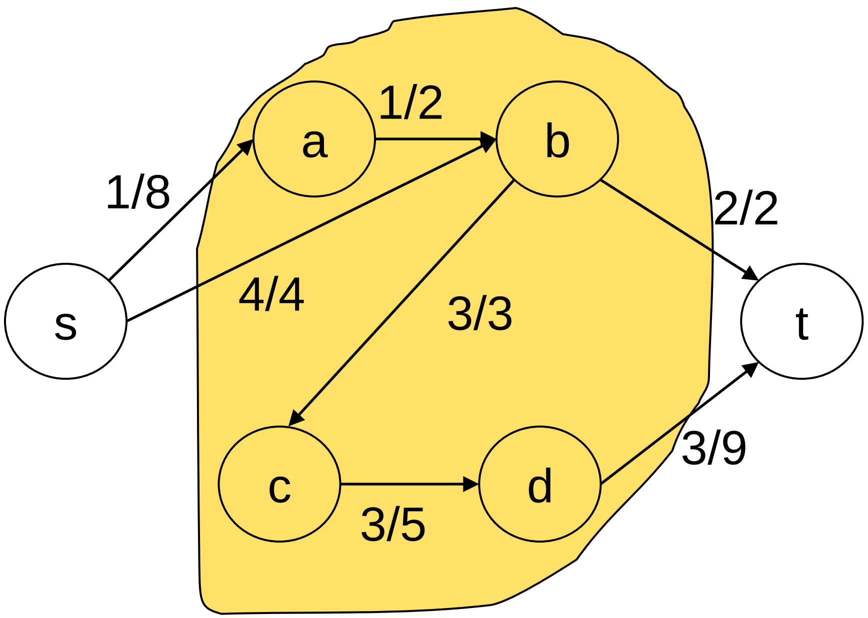


$\forall X, Y, Z \in V$ si $X \cap Y = \varnothing$

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

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

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) ($c_f(u,v)$):

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

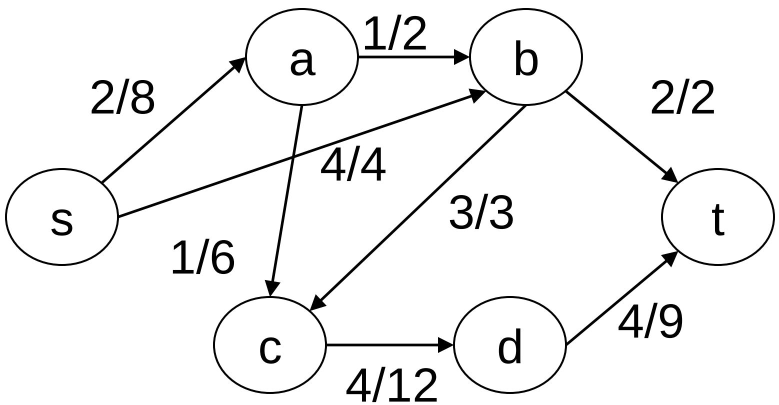
Rețea reziduală. Cale reziduală.

- 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:

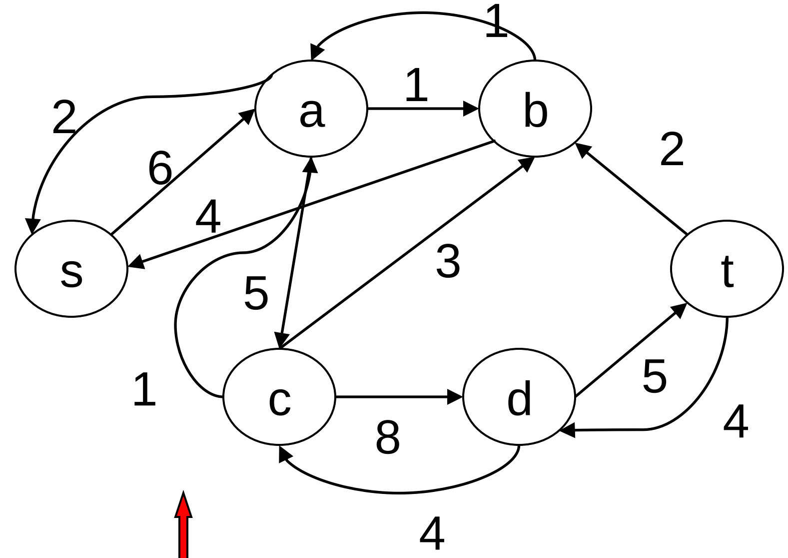
$$E_f = \{(u,v) \in V \times V \mid c_f(u,v) > 0\}.$$

- Observație: $E_f \not\subseteq E$!!!
- Definiție: Cale reziduală e un drum s..t ⊆ 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

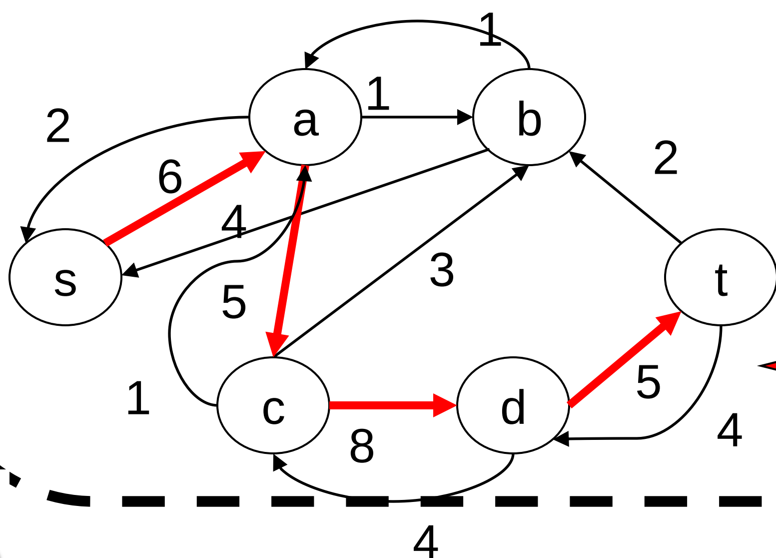


$c_f(u,v) = c(u,v) - f(u,v)$



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 → a → c → d → 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 x V->$\Re$ se definește ca fiind:

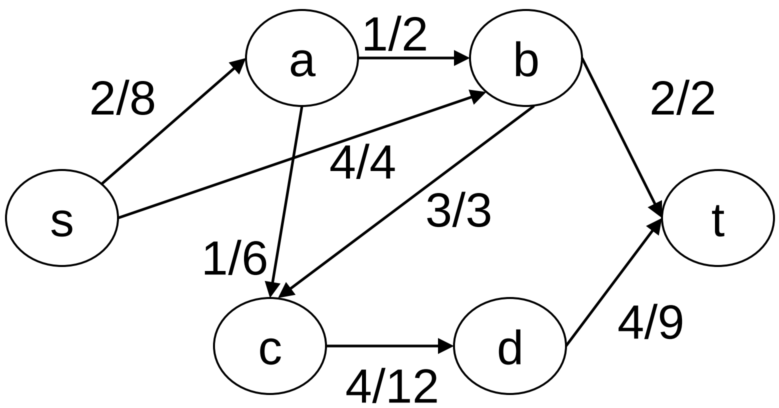
$$f_p(u,v) = \begin{cases} c_f(p), \text{ dacă } (u,v) \in p \\ -c_f(p), \text{ dacă } (v,u) \in p \\ 0, \text{ dacă } (u,v) \text{ și } (v,u) \notin p \end{cases}$$

f_p = flux în G_f; $|f_p| = c_f(p)$

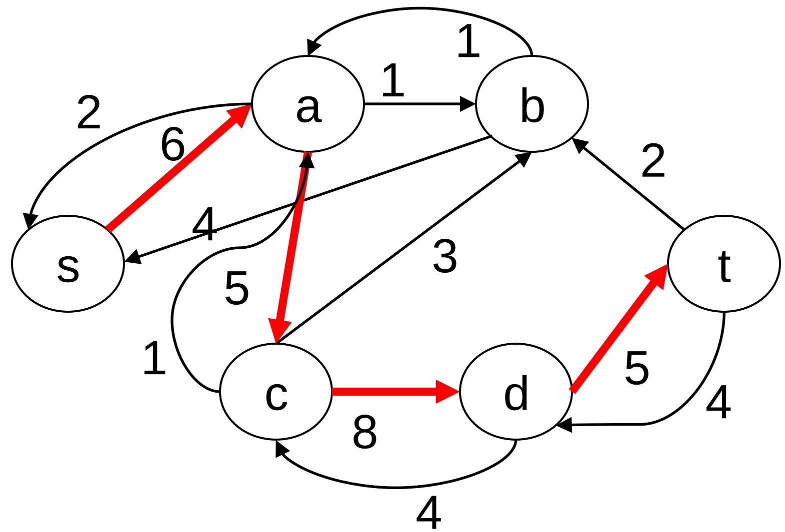
- Corolar 5.4: f' = f + f_p = flux în G, astfel încât |f'| = |f| + $|f_p|$ > |f|.

- Această Lemă ne spune cum se definește fluxul printr-o rețea reziduală.

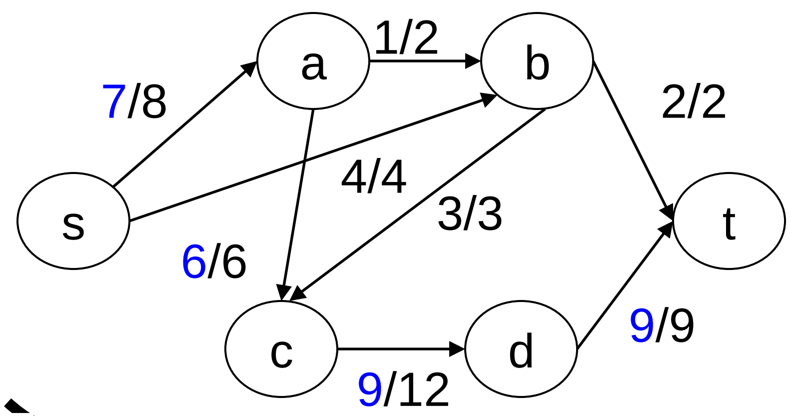
Exemplu maximizare flux



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



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



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

Calculul fluxului maxim

- Metoda Ford-Fulkerson
  - f(u,v) = 0 ∀ (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 + 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 există mai mulți algoritmi ce urmează această metodă.

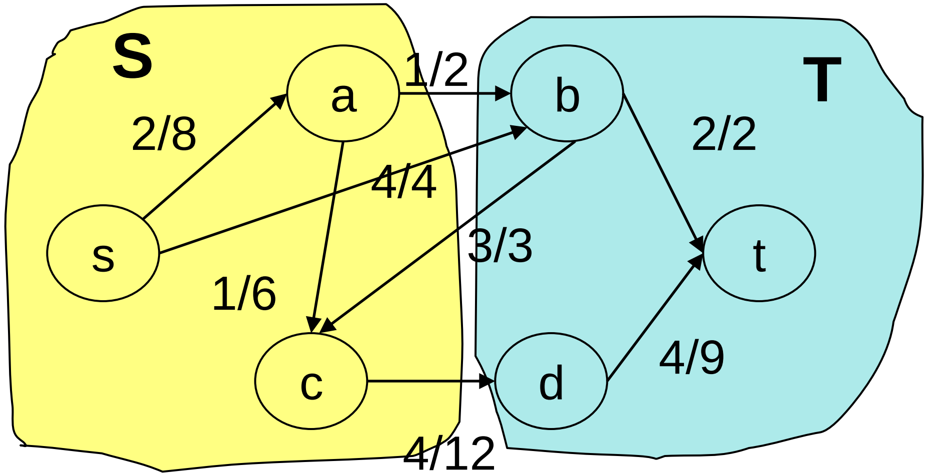
Tăieturi în rețele de flux

- Definiție: O tăietură (S,T) a unei rețele de flux G = partiționare a nodurilor în 2 mulțimi disjuncte S și T = V \ S astfel încât $s \in S$ și $t \in T$.
  - $f(S,T) = \Sigma_{x \in S}\Sigma_{y \in T}\ f(x,y)$ – fluxul prin tăietura
  - $c(S,T) = \Sigma_{x \in S}\Sigma_{y \in T}\ c(x,y)$ – capacitatea tăieturii

- 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|

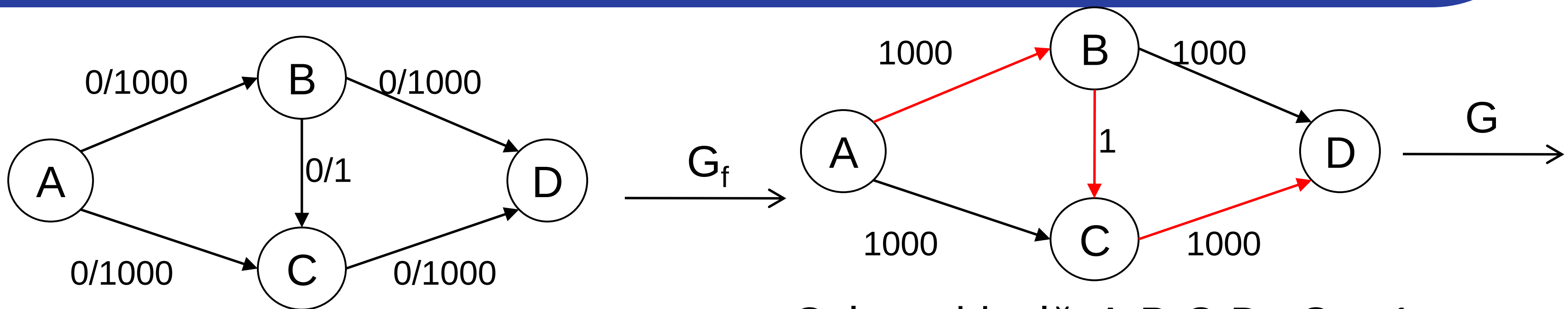
Complexitate?

Algoritmul Ford – Fulkerson (2)

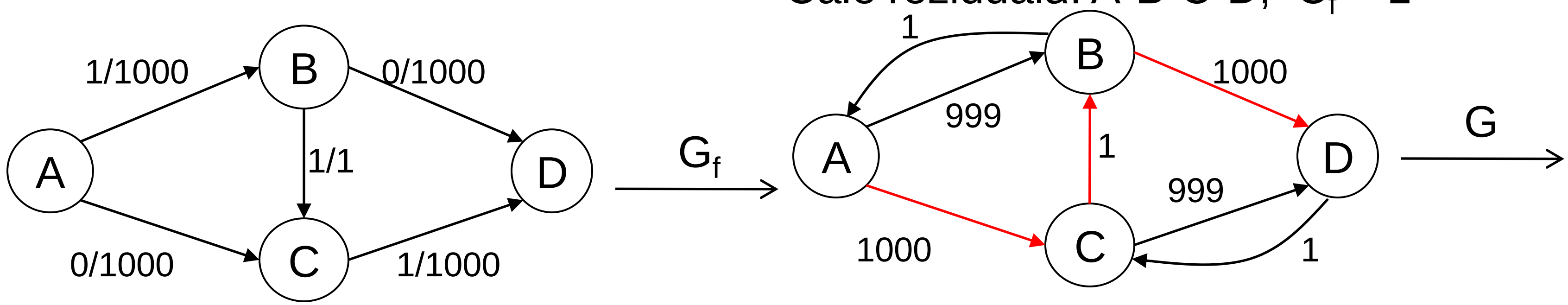
- Ford – Fulkerson(G,s,t)
  - **Pentru fiecare** (u,v) din E
    - f(u,v) = f(v,u) = 0 // O(E)
  - **Cât timp** // O(?)
    - Există o cale reziduală p între s..t în G_f // O(E)
      - $c_f(p) = \min\{c_f(u,v) \mid (u,v) \text{ din } p\}$ // O(E)
      - **Pentru fiecare** (u,v) din p // O(E)
     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

Complexitate?

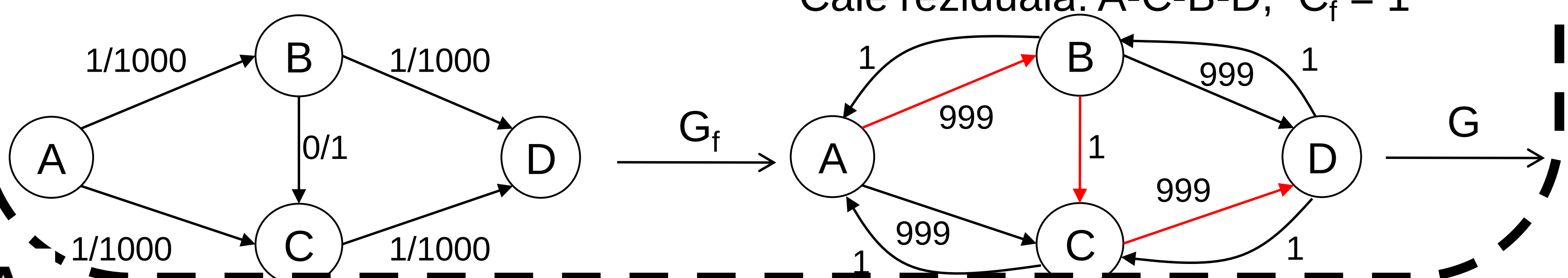
Exemplu Ford – Fulkerson (1)



Cale reziduală: A-B-C-D; $C_f = 1$

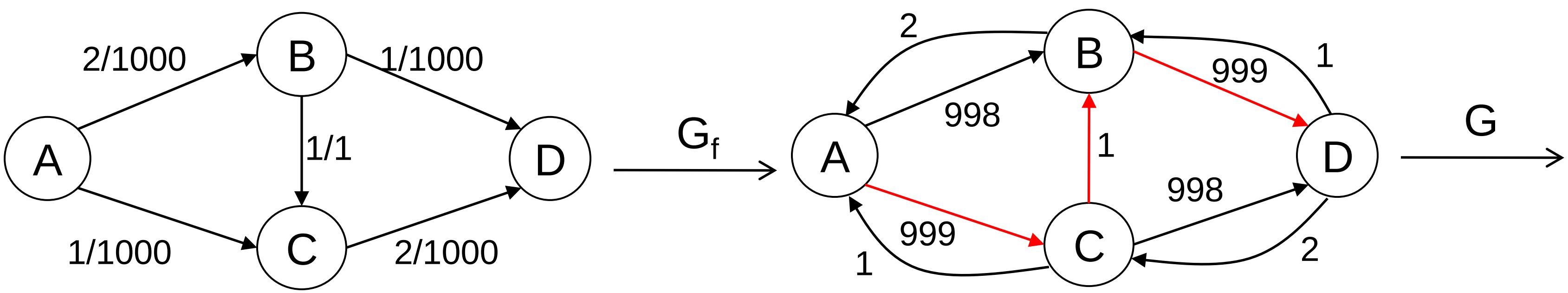


Cale reziduală: A-C-B-D; $C_f = 1$



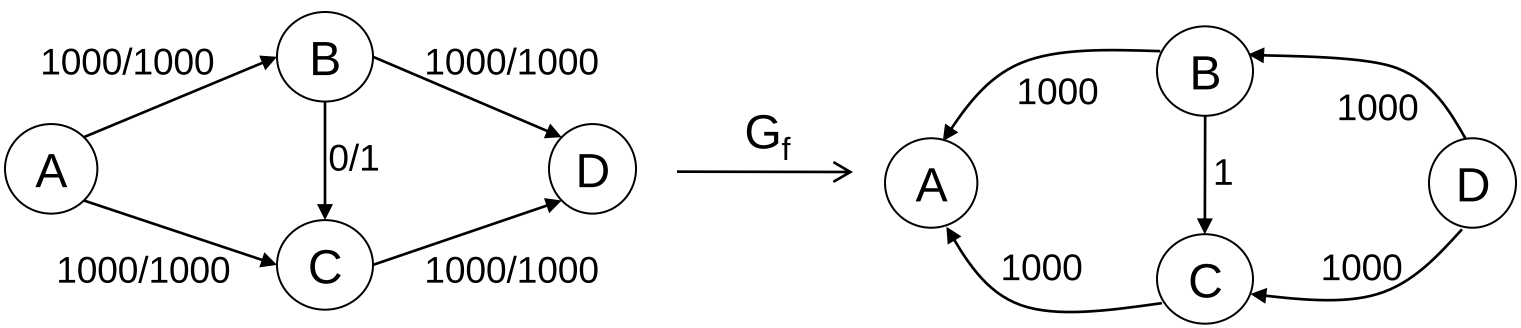
Cale reziduală: A-B-C-D; $C_f = 1$

Exemplu Ford – Fulkerson (2)



Cale reziduală: A-C-B-D; $C_f = 1$

...



Cale reziduală:Ø

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 capacităților;
  - 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|

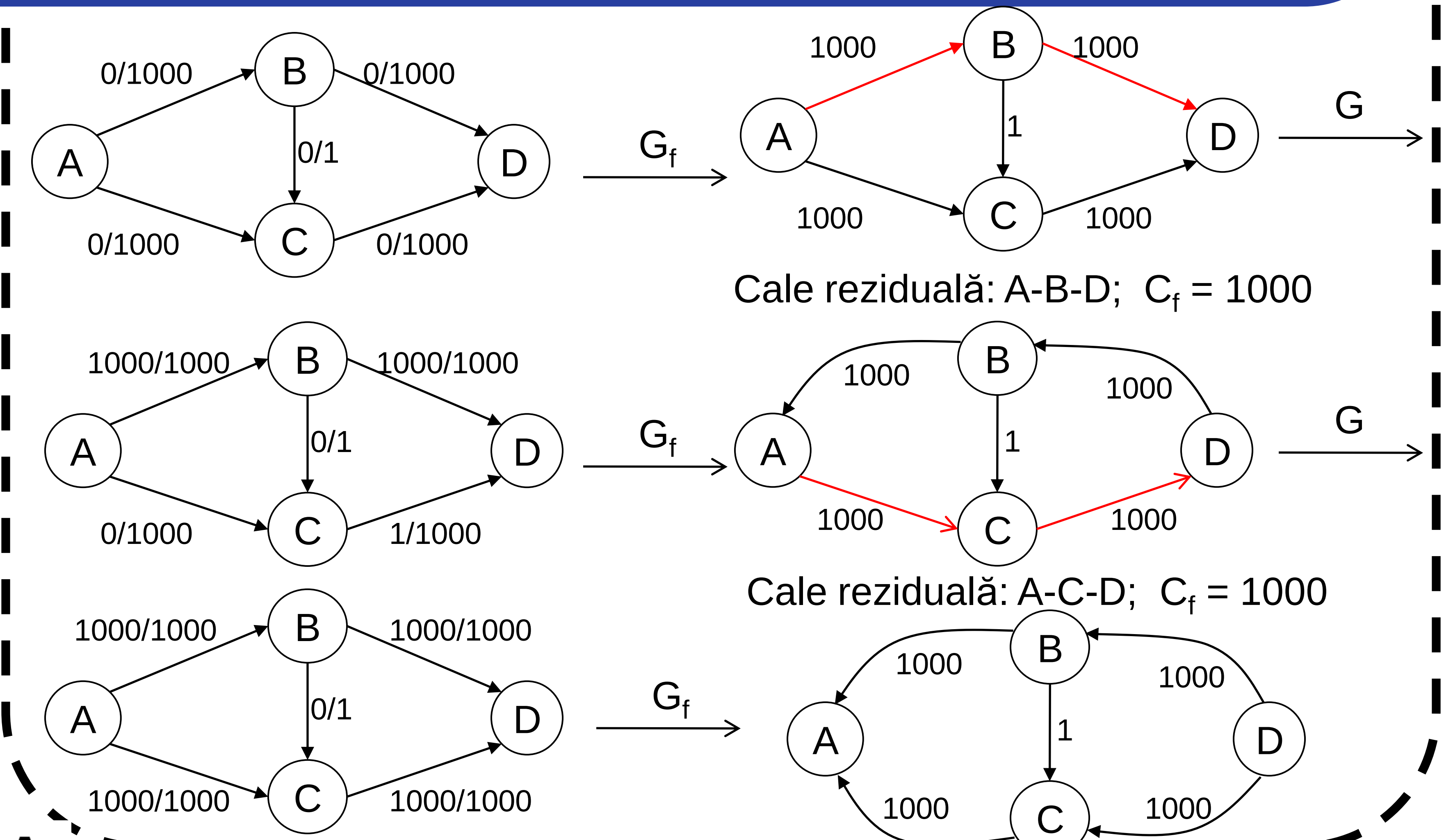
Complexitate?

Algoritmul Edmonds – Karp (2)

- 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|

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

Exemplu Edmonds-Karp



Cale reziduală: A-B-D; C_f = 1000

Cale reziduală: A-C-D; C_f = 1000

Cale reziduală: Ø

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 permanență 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 → 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 x V → $\Re$ astfel încât să fie satisfăcute restricțiile:
  - $f(u,v) \leq c(u,v), \forall (u,v) \in E$ – respectarea capacității arcelor;
  - $f(u,v) = -f(v,u), \forall u,v \in V$ – simetria fluxului;
  - $\Sigma_{v \in V} f(v,u) \geq 0, \forall u \in V \setminus \{s\}$ – ~~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 → 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 → 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 $h(u) \leq h(v)$
  - $h(u) = 1 + \min\{h(v) \mid (u,v) \in G_f\}$

Pompare preflux – Inițializare

- Init_preflux(G, s, t)
  - **Pentru fiecare** (u ∈ V)
    - e(u) = 0 // inițializare exces flux în nodul u
    - h(u) = 0 // inițializare înălțime nod u
    - **Pentru fiecare** (v ∈ V) // inițializare fluxuri
      - f(u,v) = 0
      - f(v,u) = 0
  - h(s) = |V| // inițializare înălțime sursă
  - **Pentru fiecare** (u ∈ succs(s) \ {s})

    // actualizare flux + exces
    - f(s,u) = c(s,u);
    - f(u,s) = -c(s,u);
    - e(u) = c(s,u)

Pompare preflux – Algoritm

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

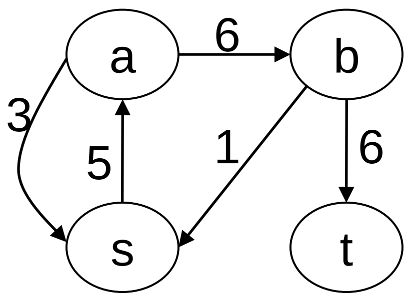
- Complexitate?

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
    - Care este numărul maxim total de înălțări? (2 |V| - 1) (|V| - 2)
  - Câte pompări?
    - Pompări saturate: 2 |V| |E|
    - Pompări nesaturate: 4 $|V|^2$ (|V| + |E|)
- Complexitate totală: O(V^2 * E) [vezi Cormen]

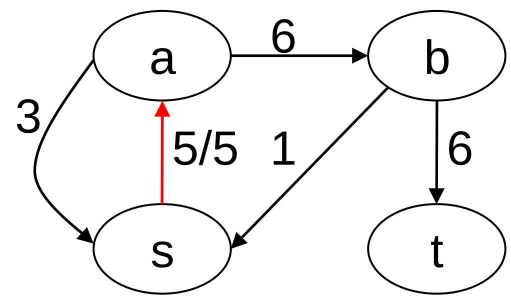
Exemplu Pompare preflux (1)

G

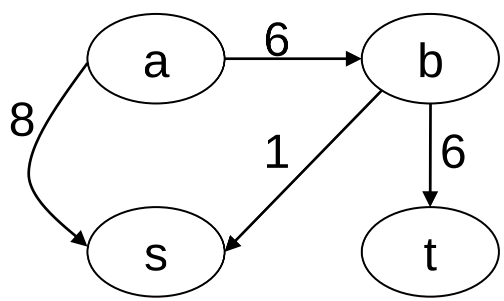


Init_preflux →

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



G_f →



G_f

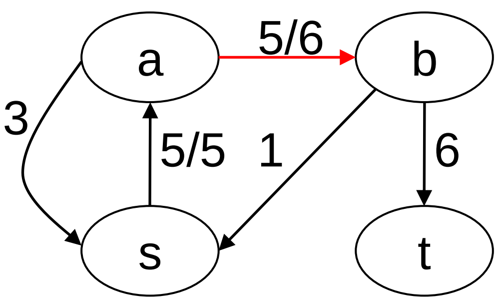
Înălțare (a) →

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

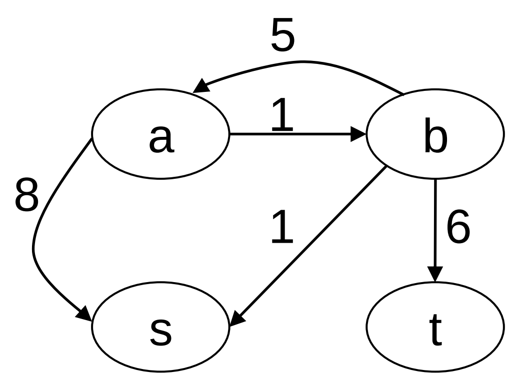
Pompare (a,b) →

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



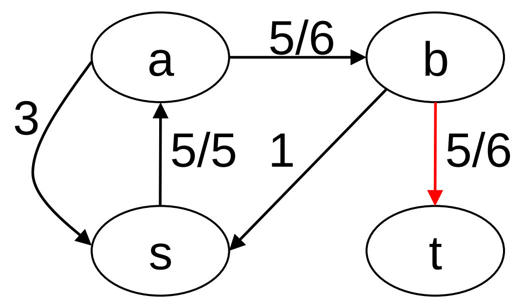
$\xrightarrow{G_f}$



$\xrightarrow[\text{(b)}]{\text{Înălțare}}$

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

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



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

ÎNTREBĂRI?

Bibliografie

[1] C. Giumale – Introducere in Analiza Algoritmilor - cap. 7

[2] http://www.gamasutra.com/features/19990212/pathdemo.zip

[3] http://www.policyalmanac.org/games/aStarTutorial.htm