UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI

Instrumente Structurale
2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Proiectarea Algoritmilor

22. Algoritmul lui Kruskal

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. 24

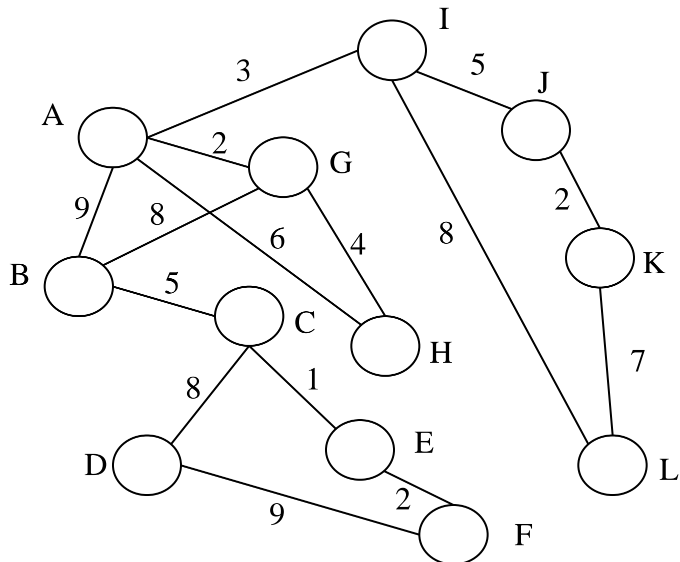
PA

Algoritmul lui Kruskal

Implementare în Java la [4] !

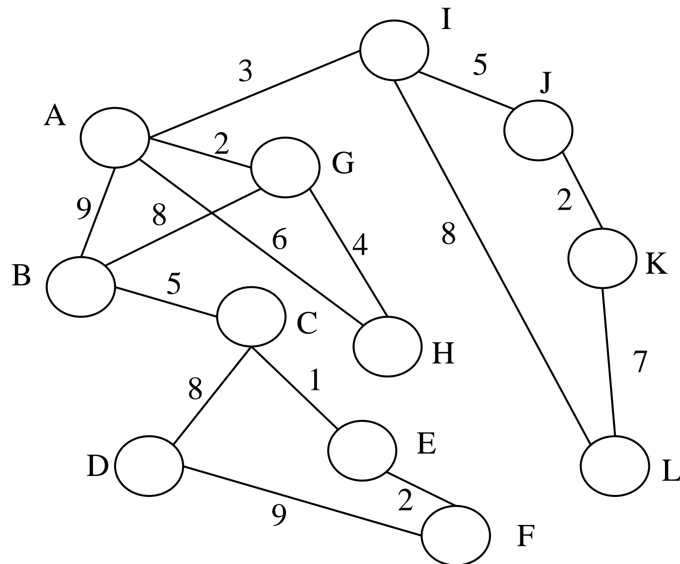
- Kruskal(G,w)
  - A = ∅; // AMA
  - **Pentru fiecare** (v ∊ V)
    - Constr_Arb(v) // creează o mulțime formată din nodul respectiv // (un arbore cu un singur nod)
  - Sortează_asc(E,w) // se sortează muchiile în funcție de // costul lor
  - **Pentru fiecare** ((u,v) ∊ E) // muchiile se extrag în ordinea // 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)

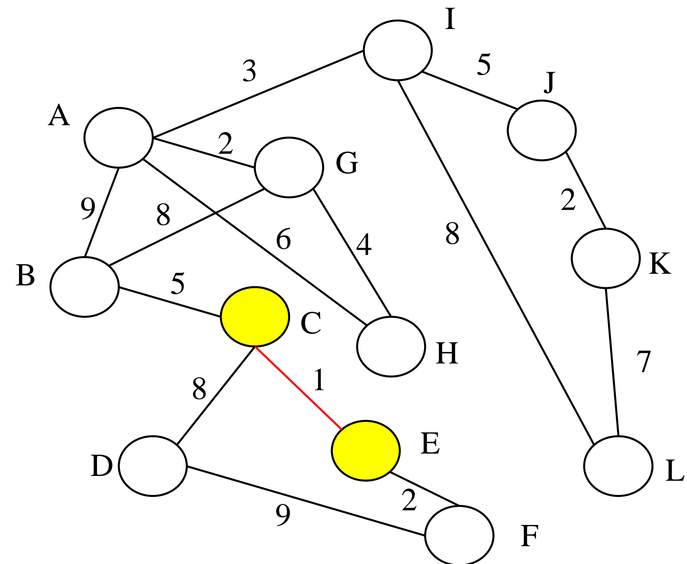


- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9

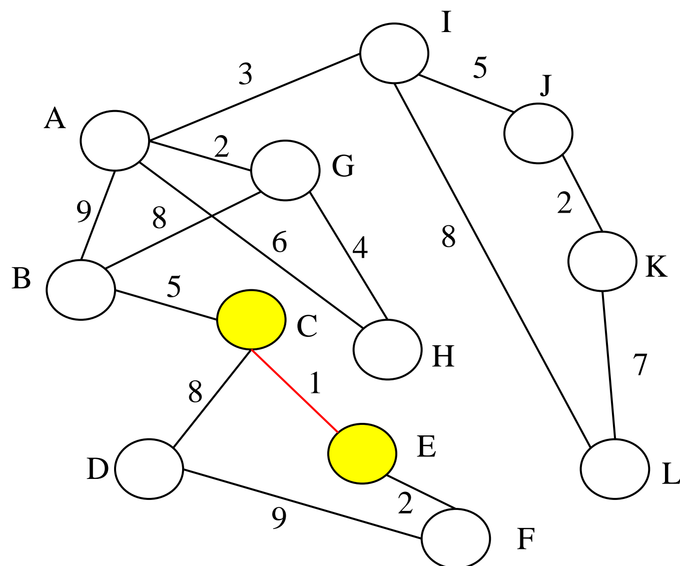
Exemplu (II)



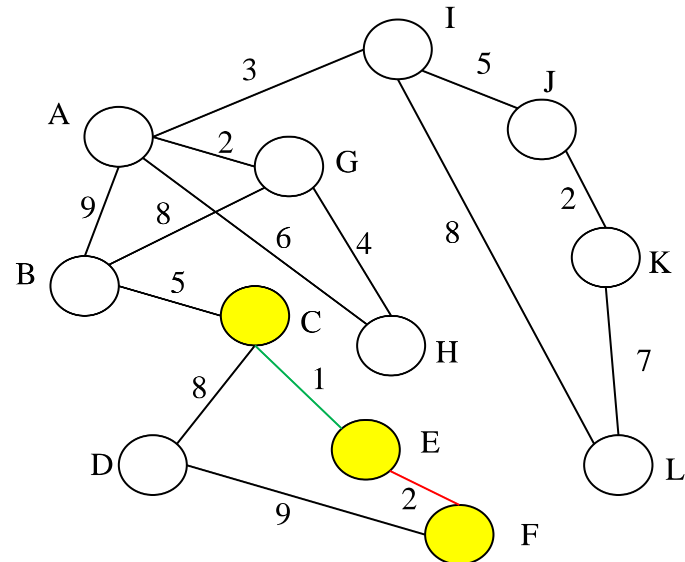
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



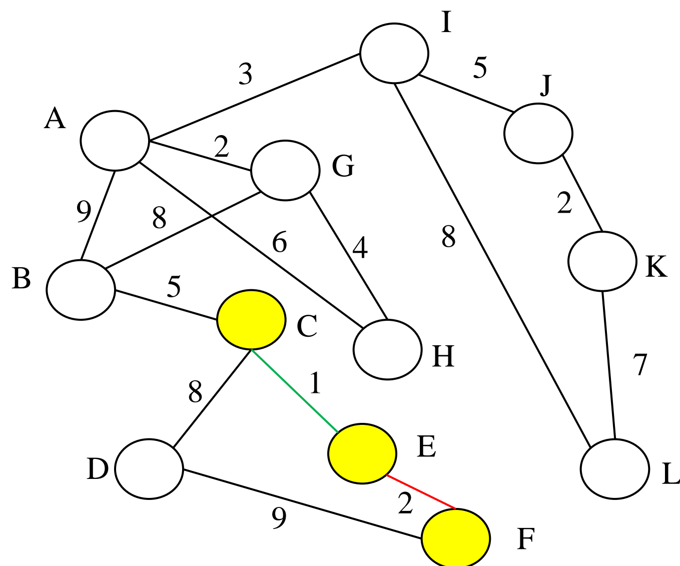
Exemplu (III)



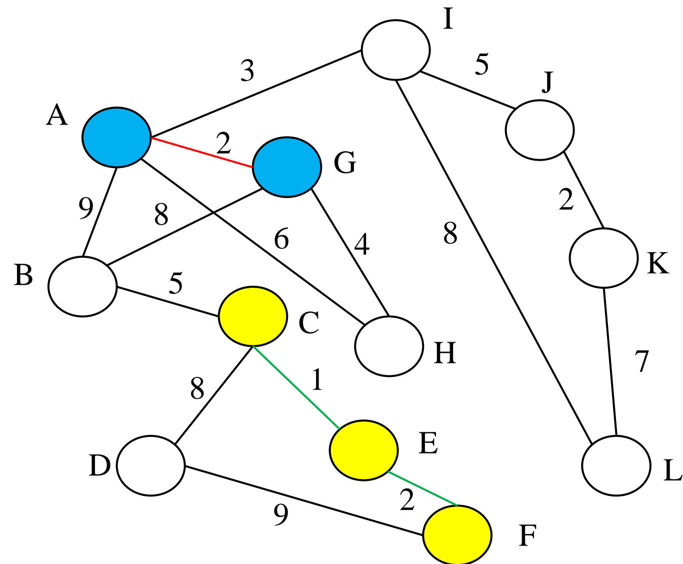
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



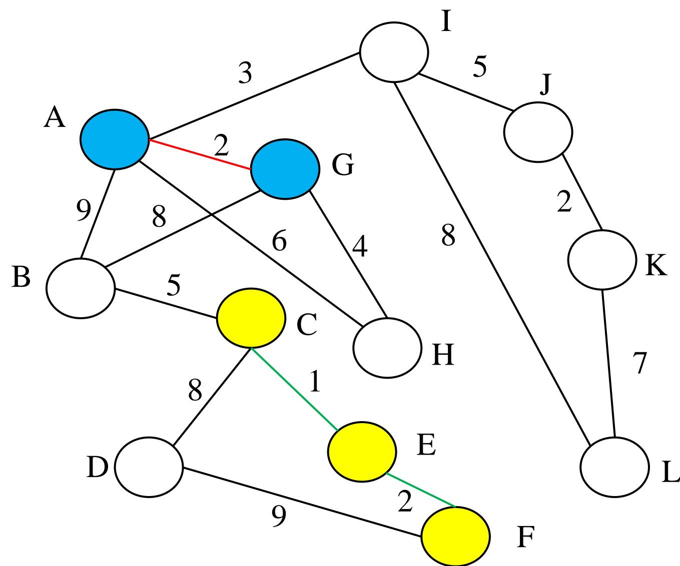
Exemplu (IV)



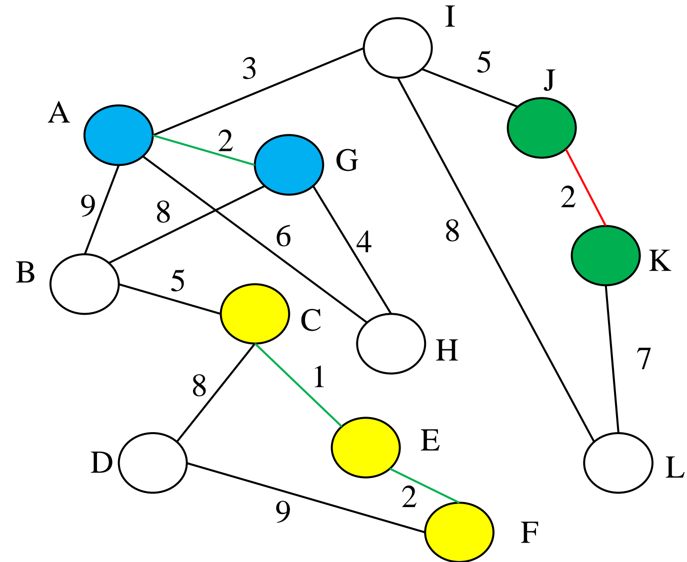
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



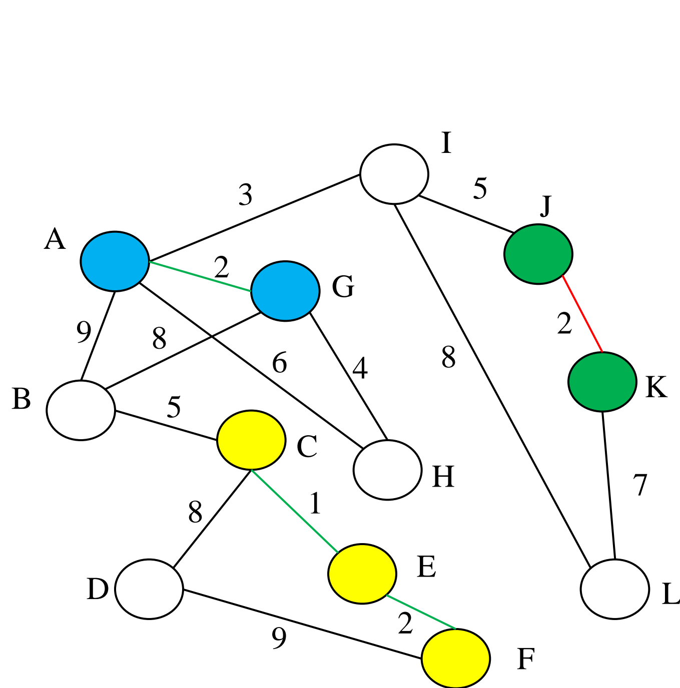
Exemplu (V)



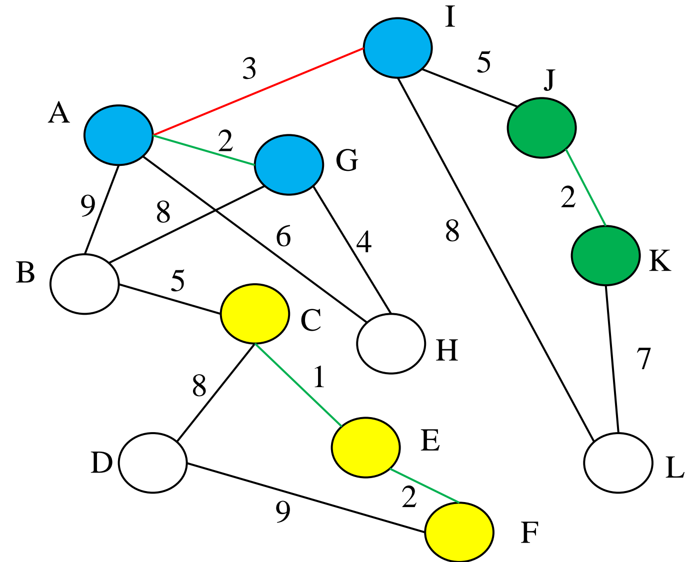
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



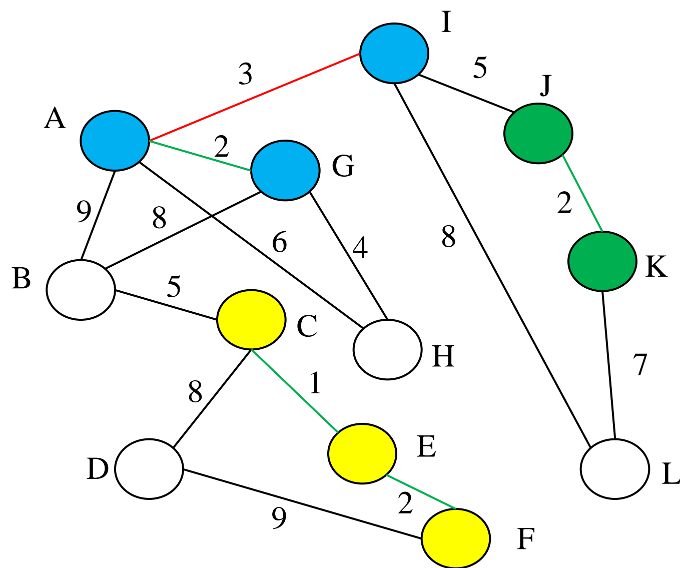
Exemplu (VI)



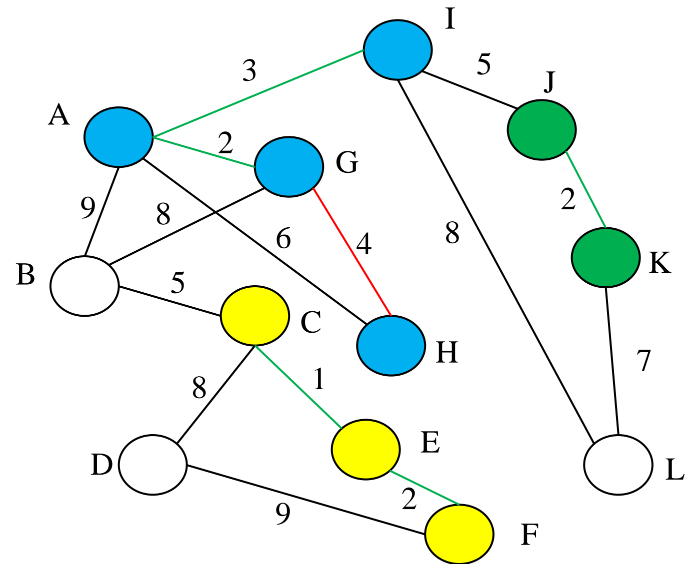
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



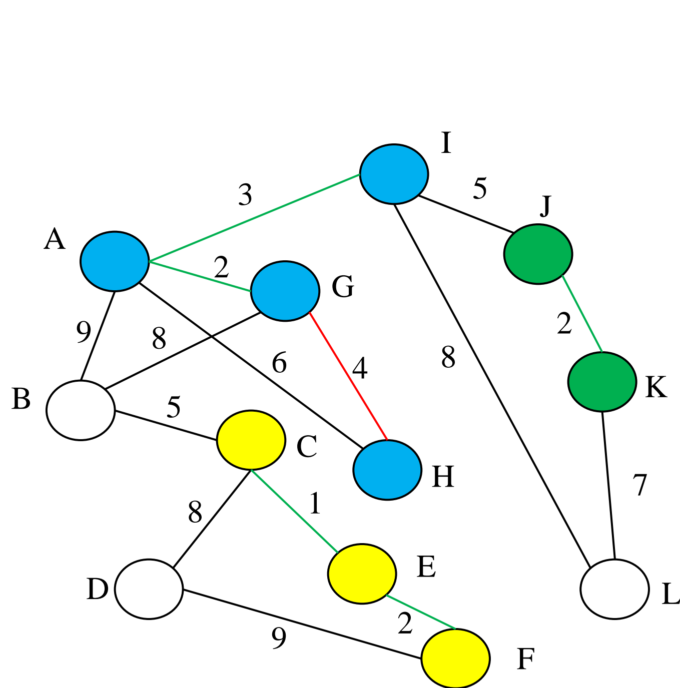
Exemplu (VII)



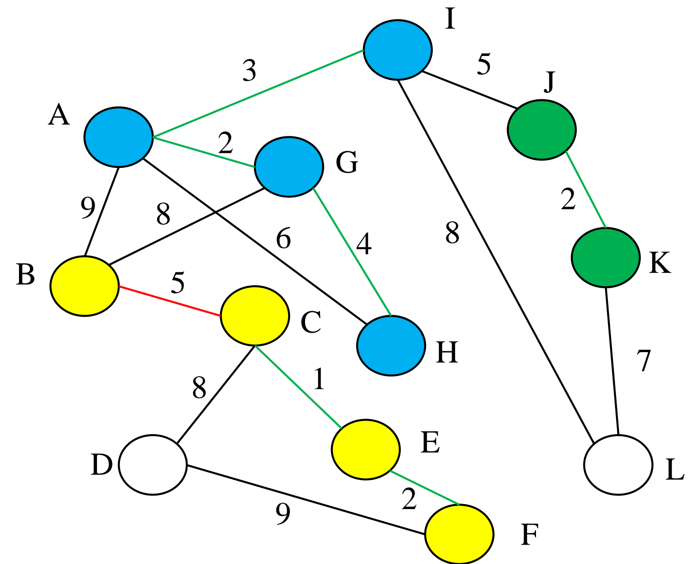
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



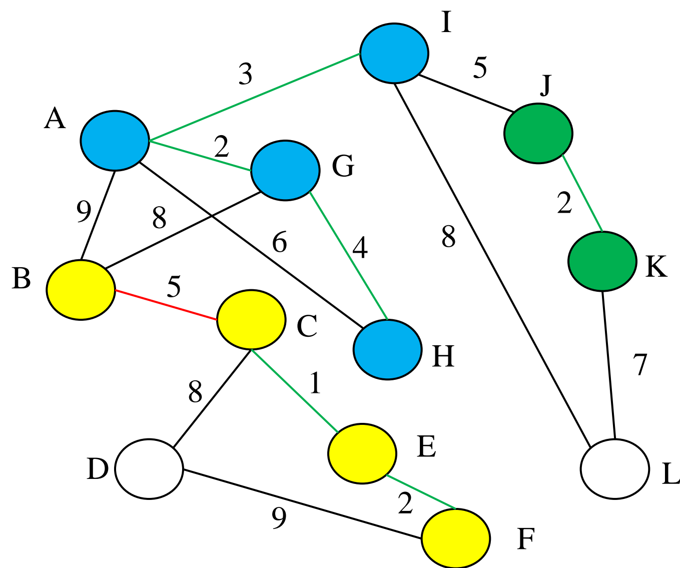
Exemplu (VIII)



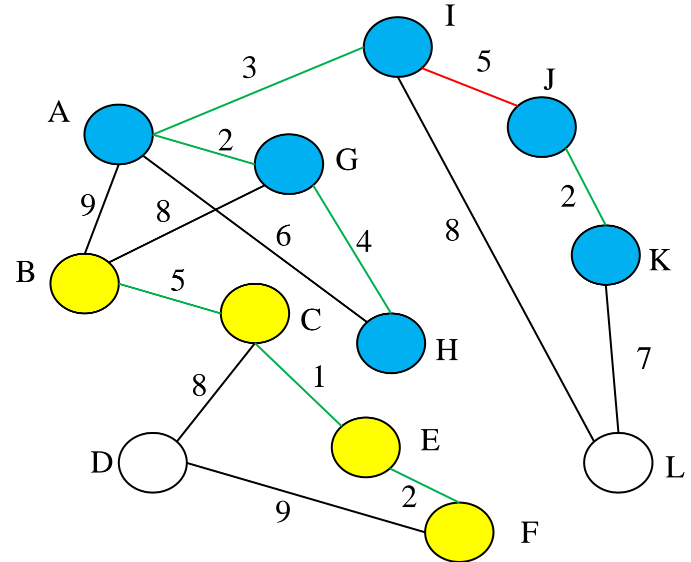
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



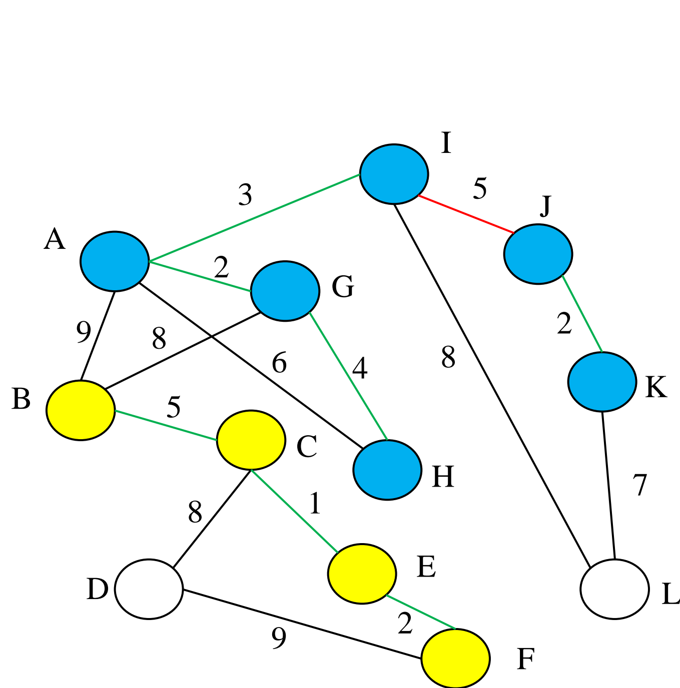
Exemplu (IX)



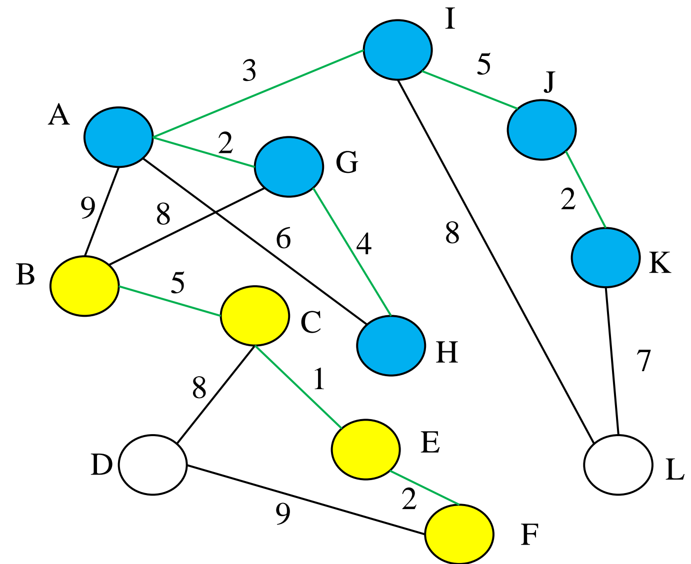
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



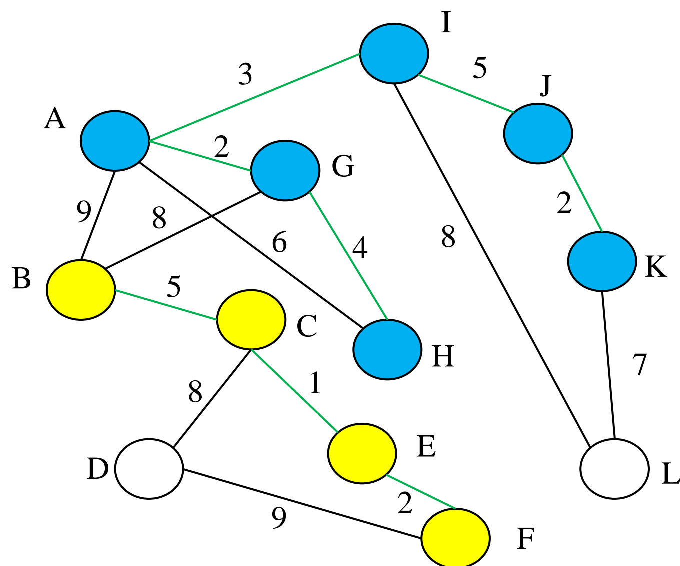
Exemplu (X)



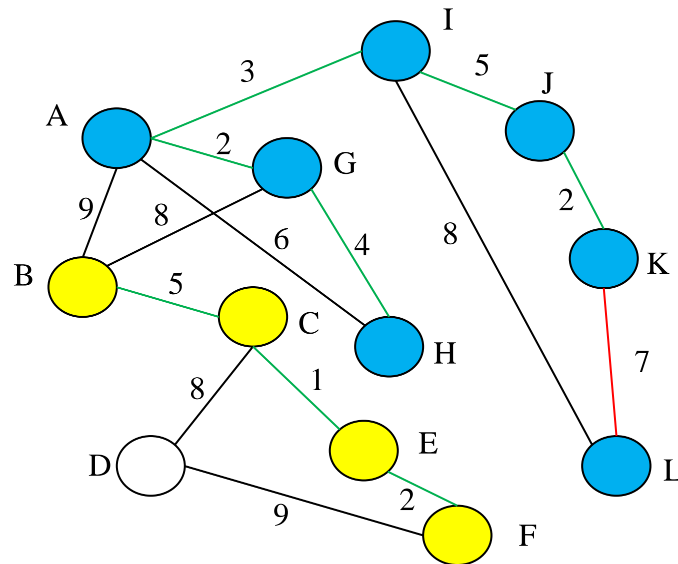
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



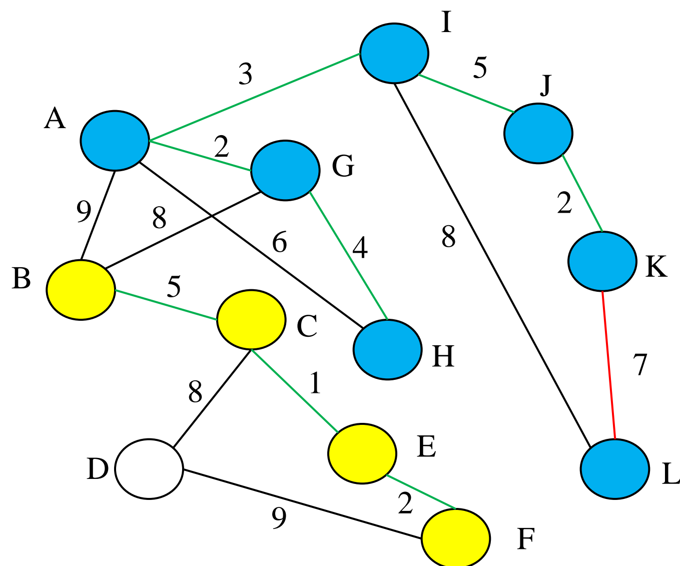
Exemplu (XI)



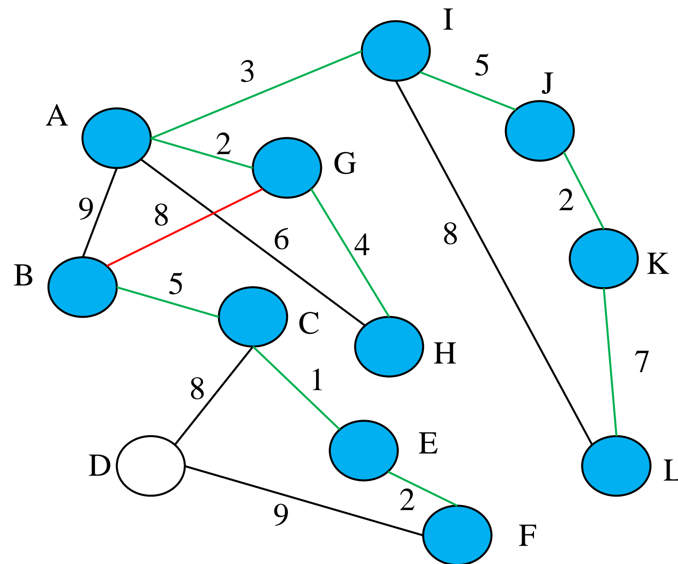
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



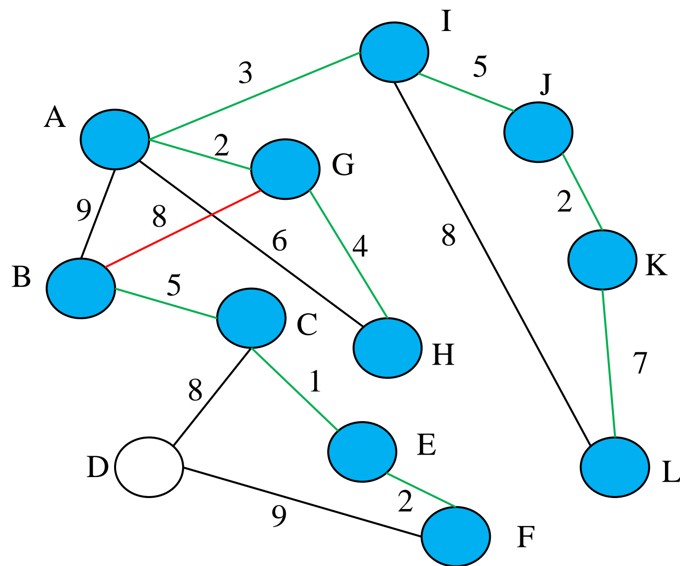
Exemplu (XII)



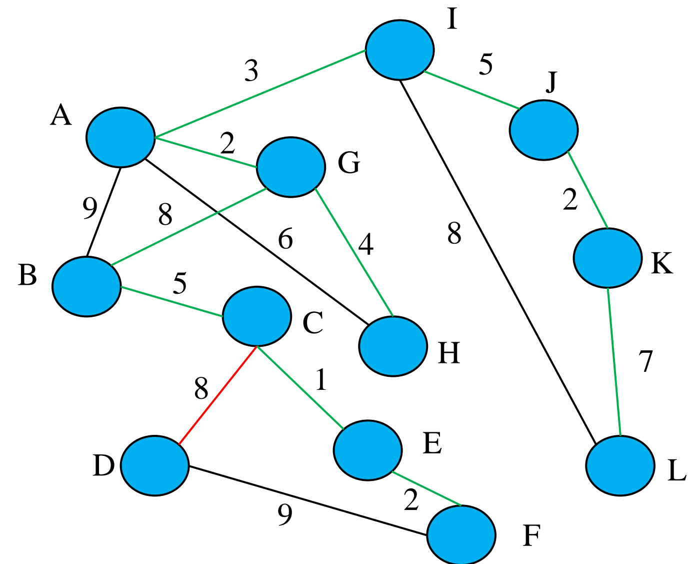
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



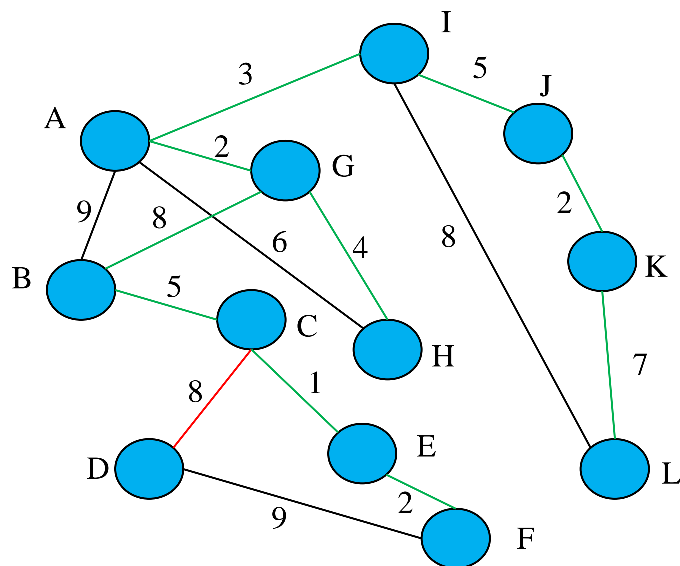
Exemplu (XIII)



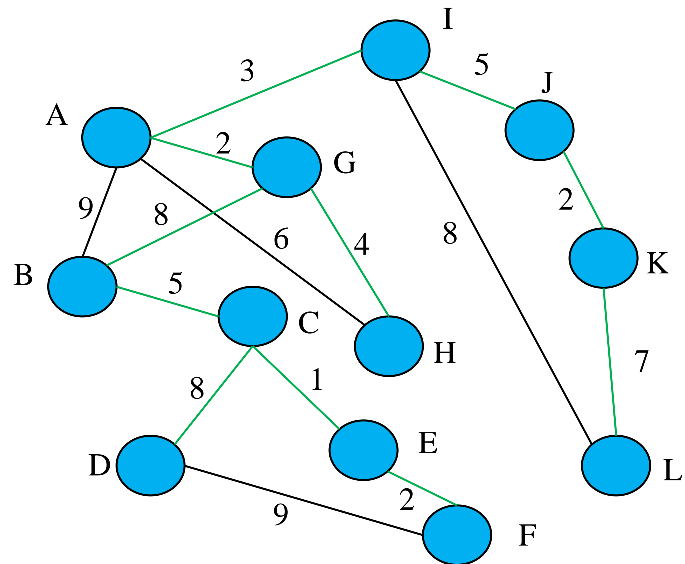
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



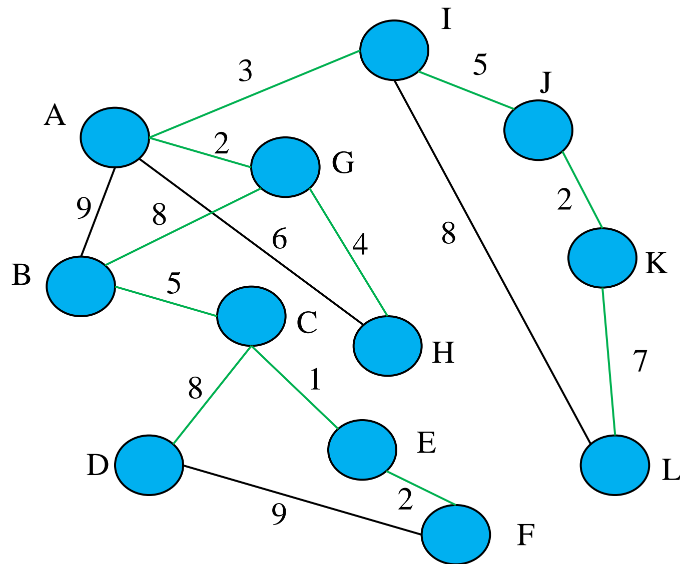
Exemplu (XIV)



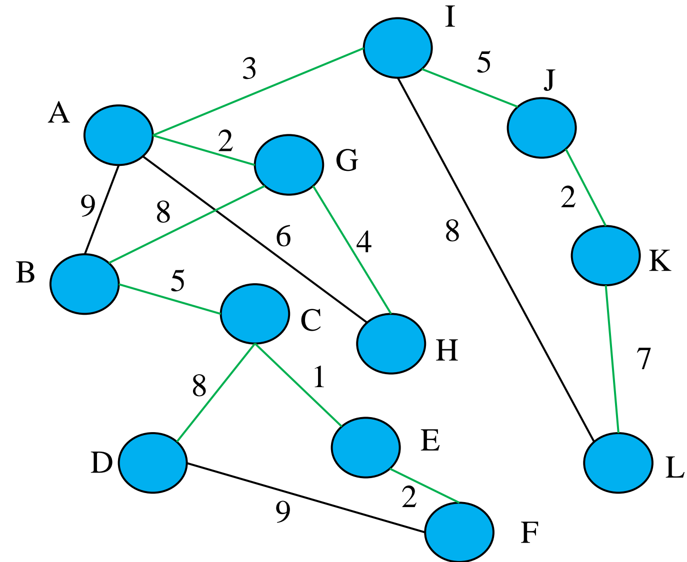
- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



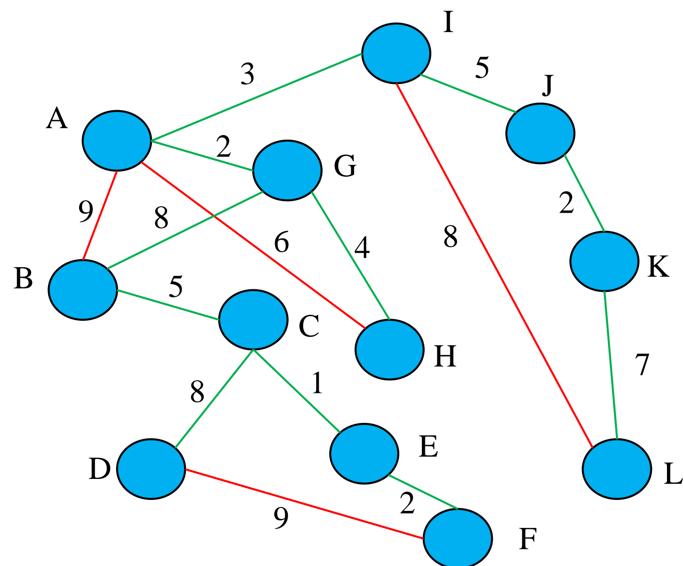
Exemplu (XV)

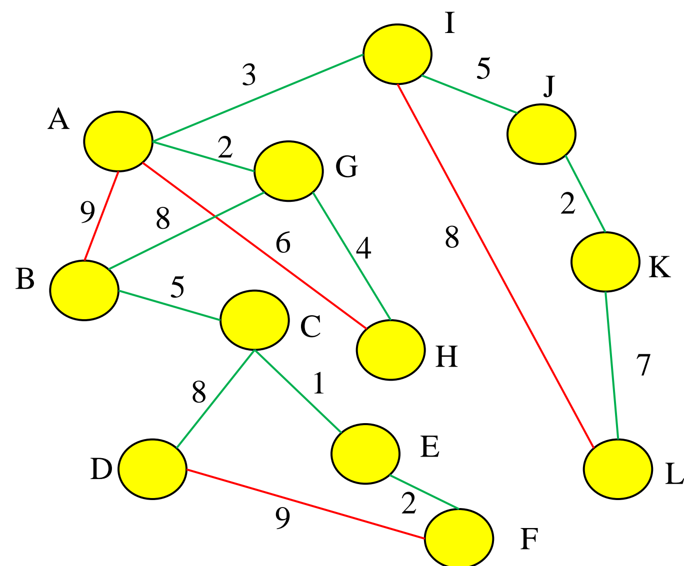


- CE -1
- EF -2
- AG-2
- JK-2
- AI-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9



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)
    - $w(u,v) = \max \{w(u',v') \mid (u',v') \in Arb(u)\}$ (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 in 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

Complexitate 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

PA

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

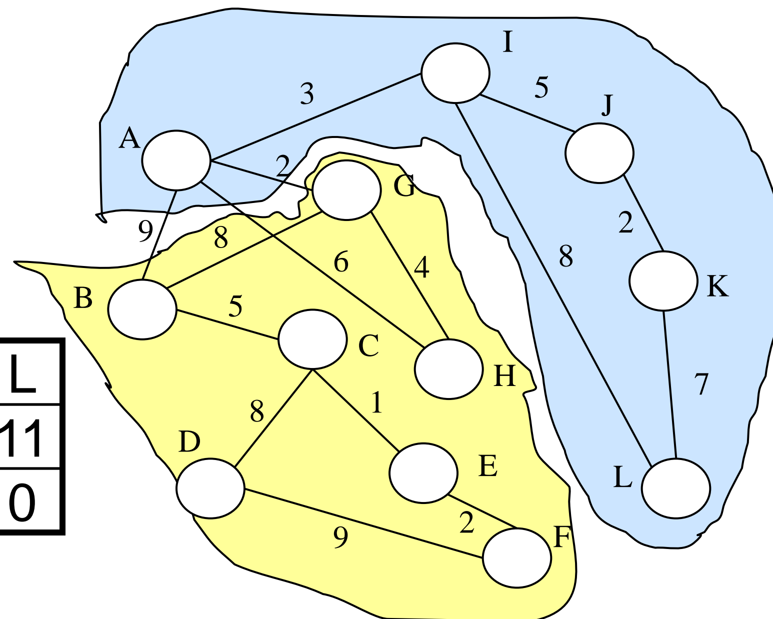
- Reuniune (M1, M2)
  - **Pentru** i **de la** length(M_1) **la** length(M_1) + length(M_2)
    - $M_1[i] = M_2[i - length(M_1)]$
  - **Întoarce** M_1
- Complexitate: V

- numărul de apelări – E
- Complexitate totală: $E*V^2$

PA

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

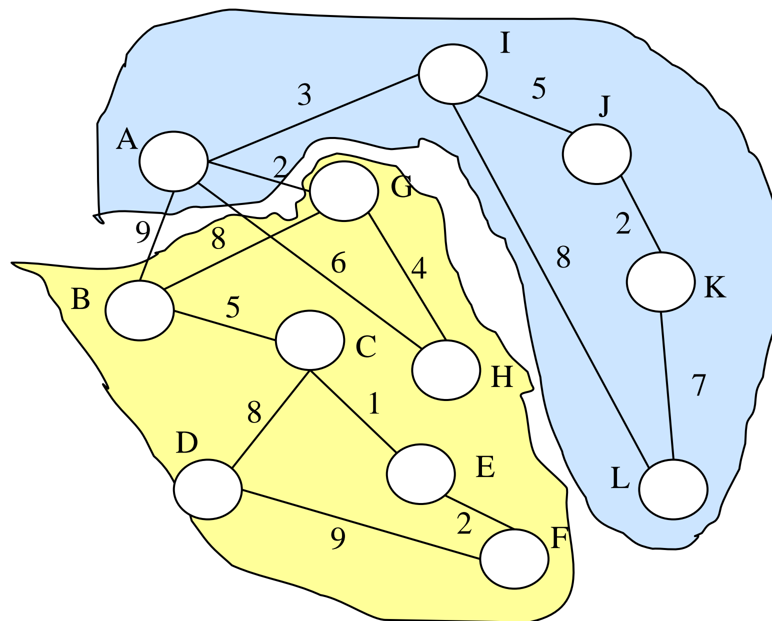
- Complexitate maximă
  - O(V * E) // E = numărul de reuniuni

- 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)
  - Arb(u) != Arb(v)

Complexitate?

- Reuniune (u,v) // implică identificarea rădăcinii
  - v = Arb(v)
  - id[v] = u;

Reuniune rapidă (Complexitate)

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

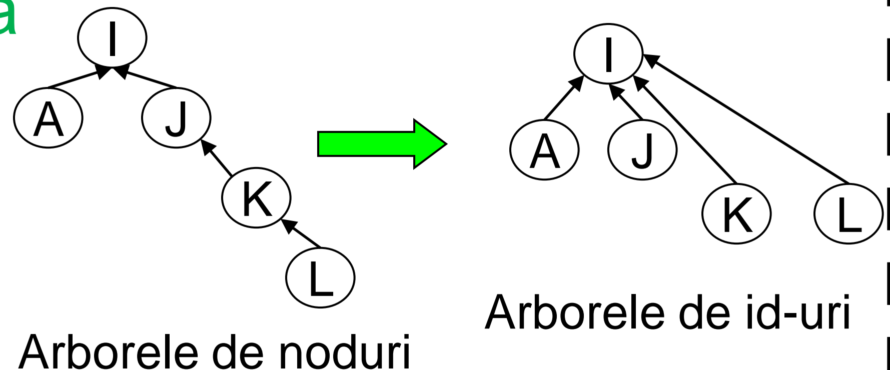
Reuniunea – O(V) // implică regăsirea rădăcinii pentru a ști unde se face modificarea

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 cate 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

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