Grafuri ponderate

-arbori parțiali de cost minim-

Ștefan Popescu

PopescuStefanBDV@gmail.com

Grafuri ponderate

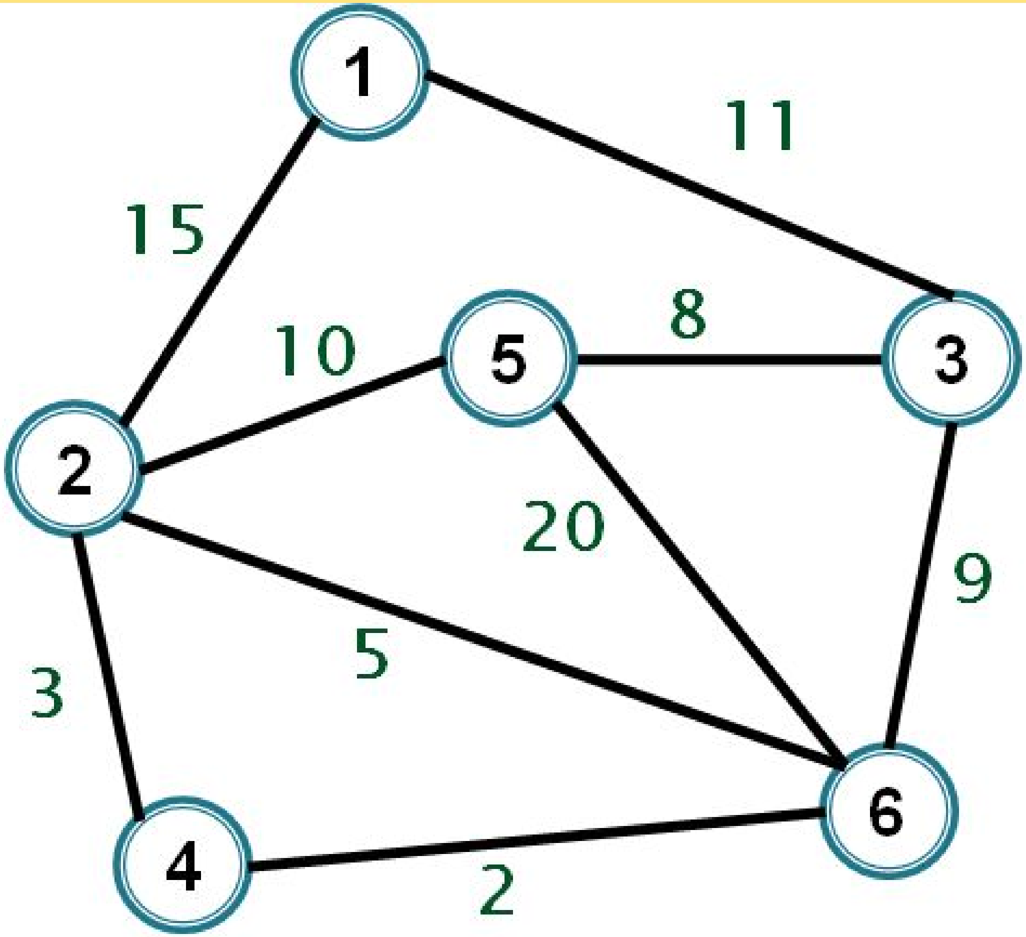
- **G = (V, E, w)** **conex ponderat**
  - **w:E→R - ** funcție de pondere (cost)
  - Pentru **A⊆E**

$$w(A) = \sum_{e \in A} w(e)$$

  - Pentru T - subgraf al lui G

$$w(T) = \sum_{e \in E(T)} w(e)$$

Grafuri ponderate - Exemplu



Grafuri ponderate - Reprezentare

- **Matrice de costuri:**

Grafuri ponderate - Reprezentare

- **Matrice de costuri:**
  - m[i][j]= w(i,j) dacă (i,j)-muchie / 0 dacă i=j / ∞ altfel

Grafuri ponderate - Reprezentare

- **Matrice de costuri:**
  - m[i][j]= w(i,j) dacă (i,j)-muchie / 0 dacă i=j / ∞ altfel
- **Liste de vecini** (array de perechi nod/cost)

Grafuri ponderate - Reprezentare

- **Matrice de costuri:**
  - m[i][j]= w(i,j) dacă (i,j)-muchie / 0 dacă i=j / ∞ altfel
- **Liste de vecini** (array de perechi nod/cost)
- **Lisă de muchii** (triplete nod-nod-cost)

Arbori parțiali de cost minim (APM)

- Dându-se un graf ponderat conex $G=(V,E,w)$, să se găsească un subgraf $T=(V,E')$ astfel încât T - conex, iar $w(T)$ - minimă

Aplicații pt. APM

- Întreținerea drumurilor auto și de cale ferată
- Proiectarea de rețele
- Proiectarea cicruitelor electrice
- Clustering

Aplicații pt. APM

Algoritmi pentru determinarea de APM

Algoritmi pentru determinarea de APM

Cum putem determina un APM într-un graf conex ponderat?

Algoritmi pentru determinarea de APM

Hint: Adăugarea succesivă de muchii, până când construcția devine conexă iar suma costurilor muchiilor selectate în construcție să fie minimă

Consecință: Nu vom selecta niciodată o muchie care să închidă un ciclu

Algoritmi pentru determinarea de APM

După ce criteriu alegem muchiile?

Algoritmi pentru determinarea de APM

După ce criteriu alegem muchiile?

Răspunsuri multiple posibile!!!!

Algoritmi pentru determinarea de APM

Algoritmul lui Kruskal

Algoritmul lui Kruskal

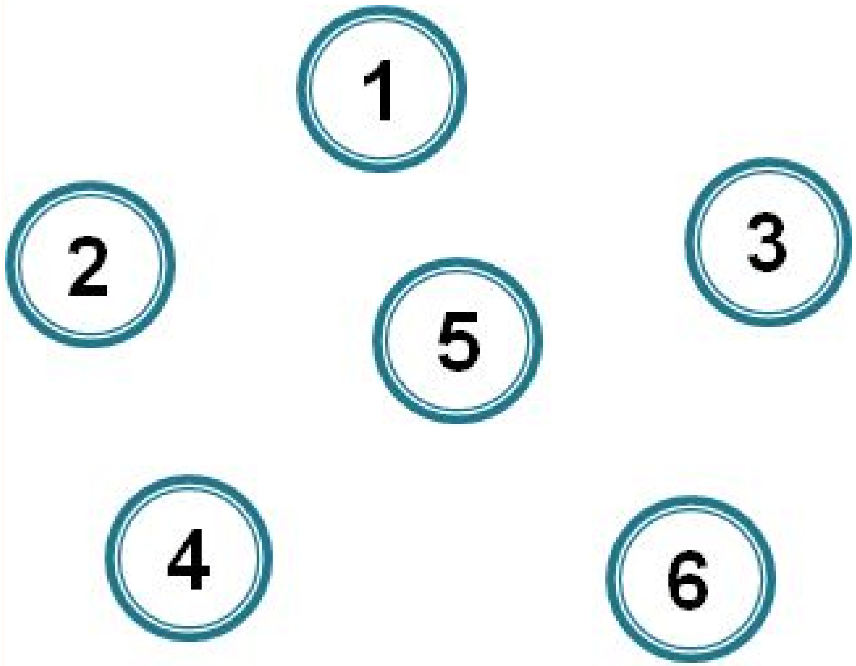
La un <u>pas</u> este selectată o <u>muchie de cost minim</u> care <u>nu formează cicluri</u> cu muchiile deja selectate (care unește două componente din graful deja construit)

Algoritmul lui Kruskal - intuitiv

- **Inițial T= (V; Ø)**
- **pentru i = 1, n-1**
  - alege o muchie uv cu **cost minim** a.î. u,v sunt în **componente conexe diferite** (**T+uv aciclic**)
  - E(T) = E(T) U uv

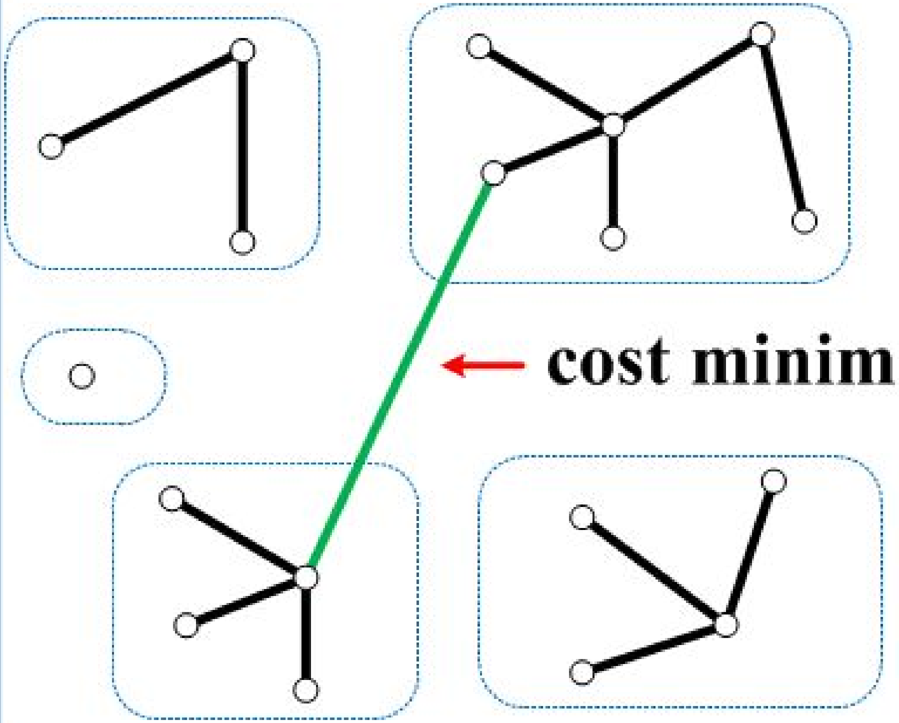
Algoritmul lui Kruskal - intuitiv

- **Pasul inițial:** Toate cele *n* noduri sunt <u>izolate</u>, fiecare formând proria-i <u>componentă conexă</u>

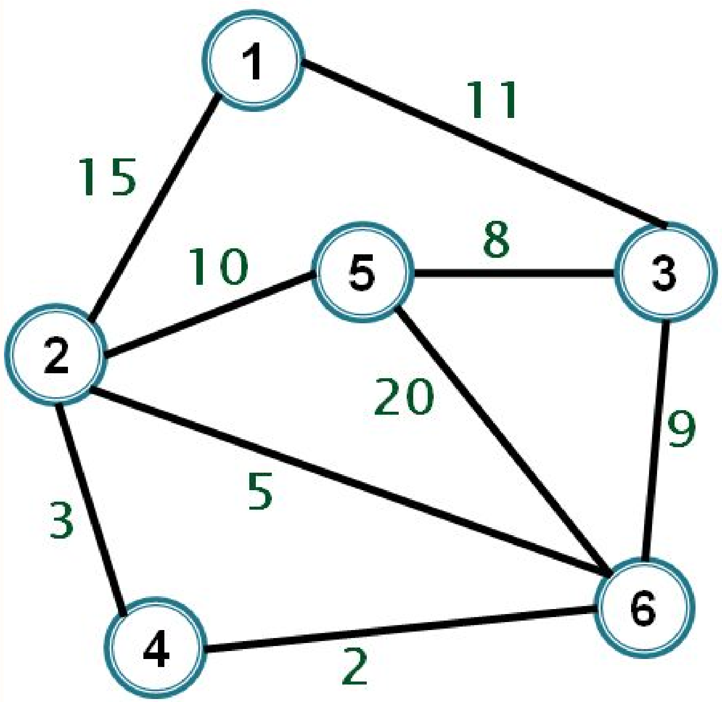


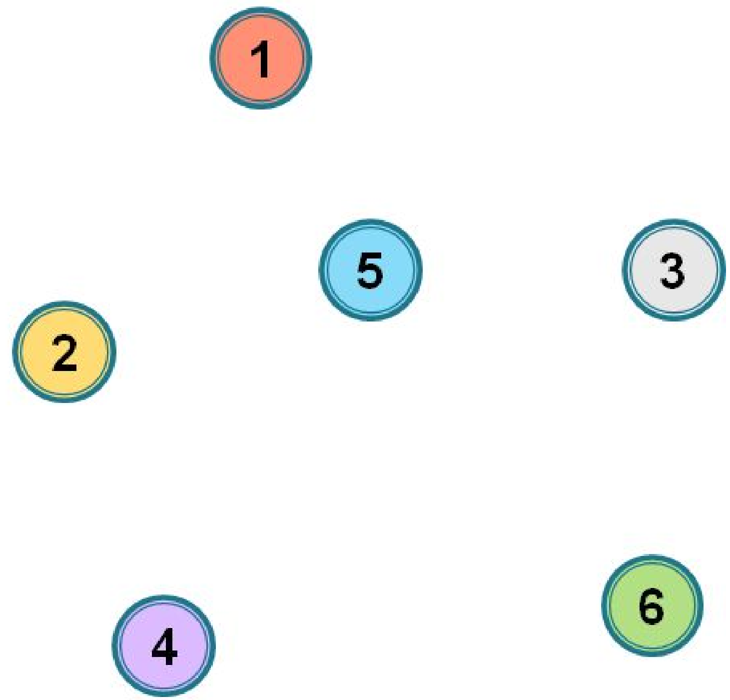
Algoritmul lui Kruskal - intuitiv

- **Pas oarecare:** Muchiile deja selectată formează mai multe componente conexe. Muchia selectată la pasul curent, va fi muchia minimă cu capetele în componente conexe diferite (nu închide ciclu).

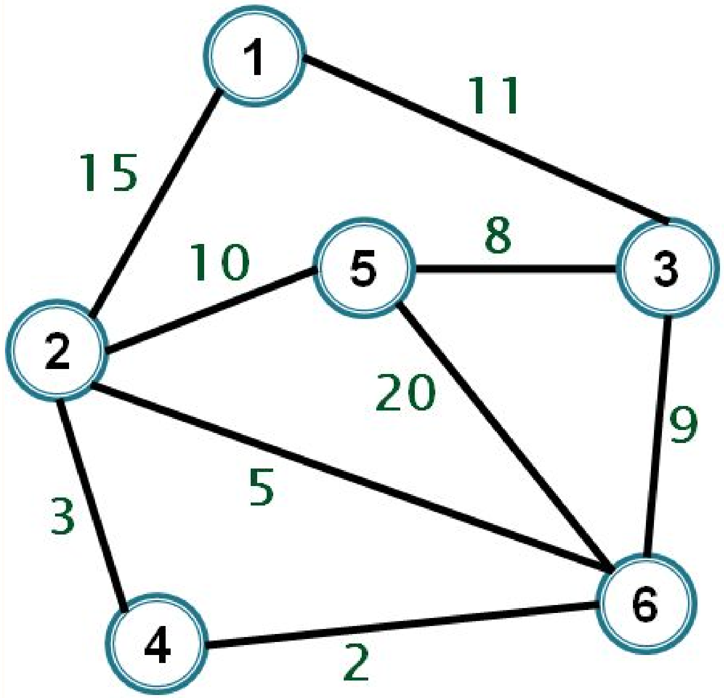


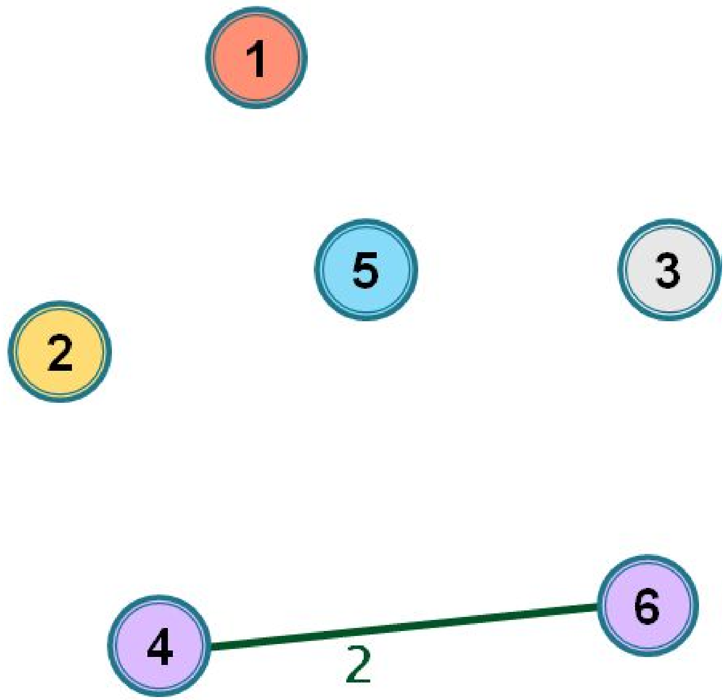
Algoritmul lui Kruskal



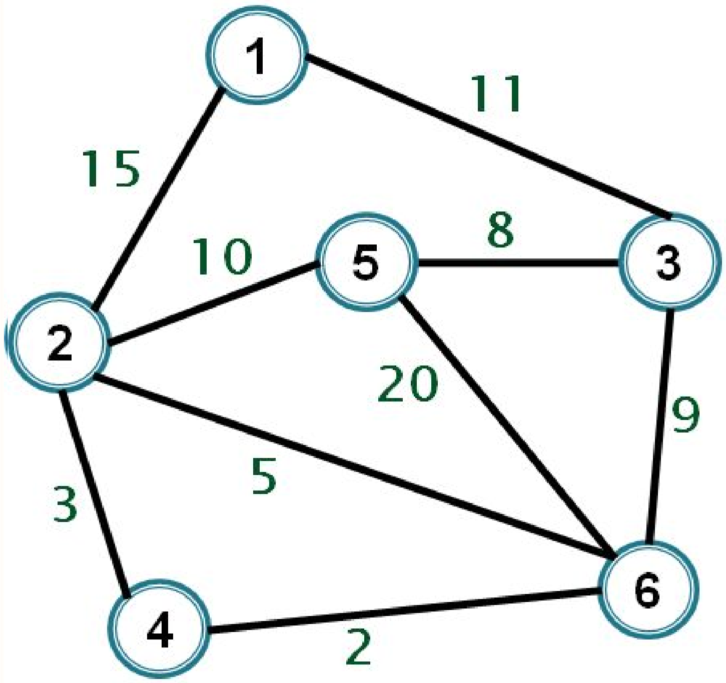


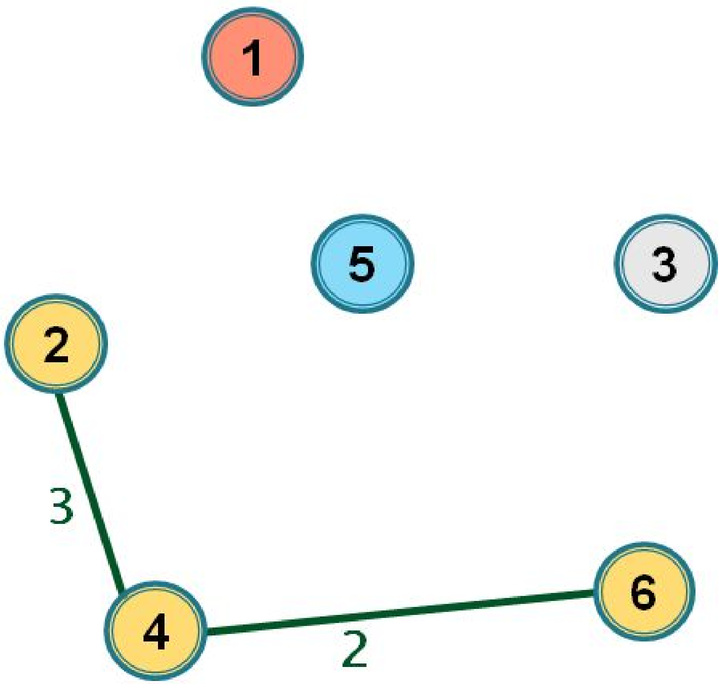
Algoritmul lui Kruskal



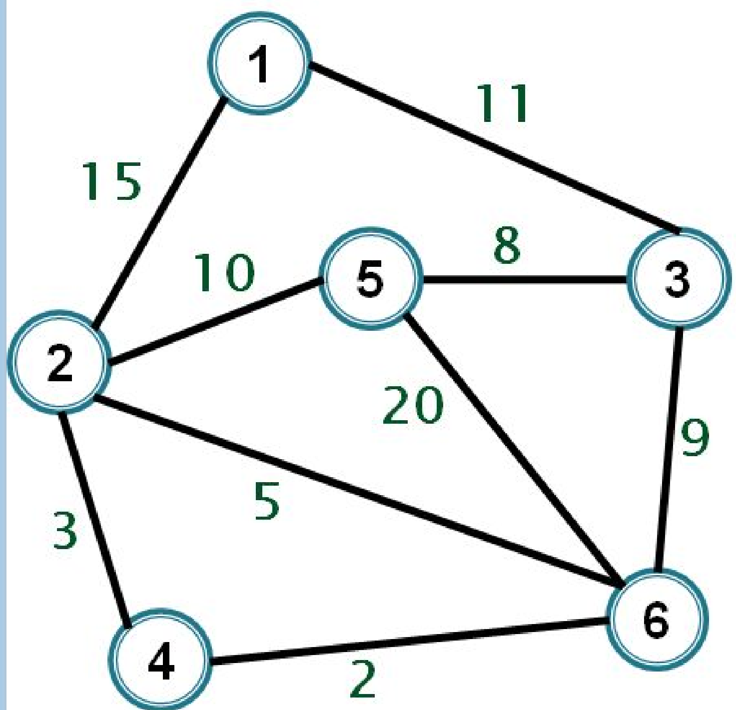


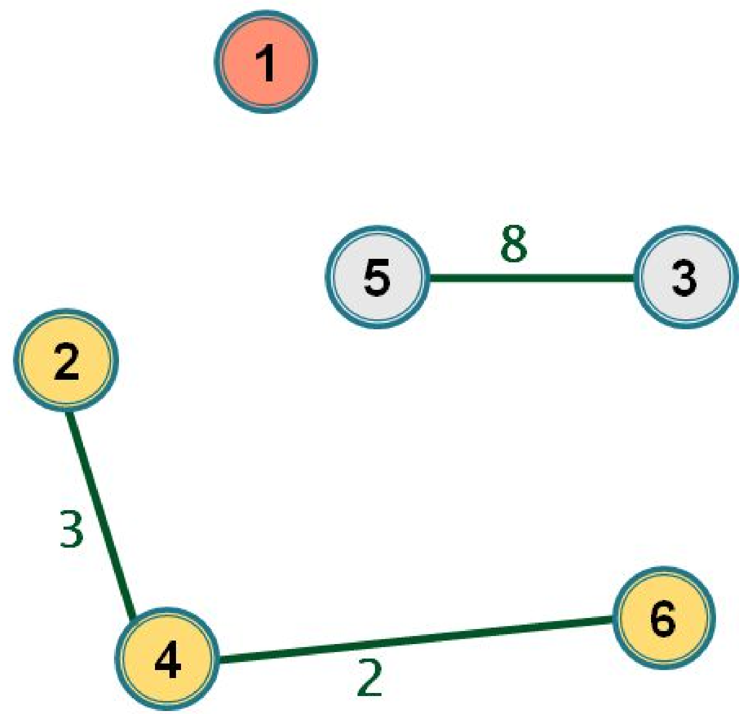
Algoritmul lui Kruskal



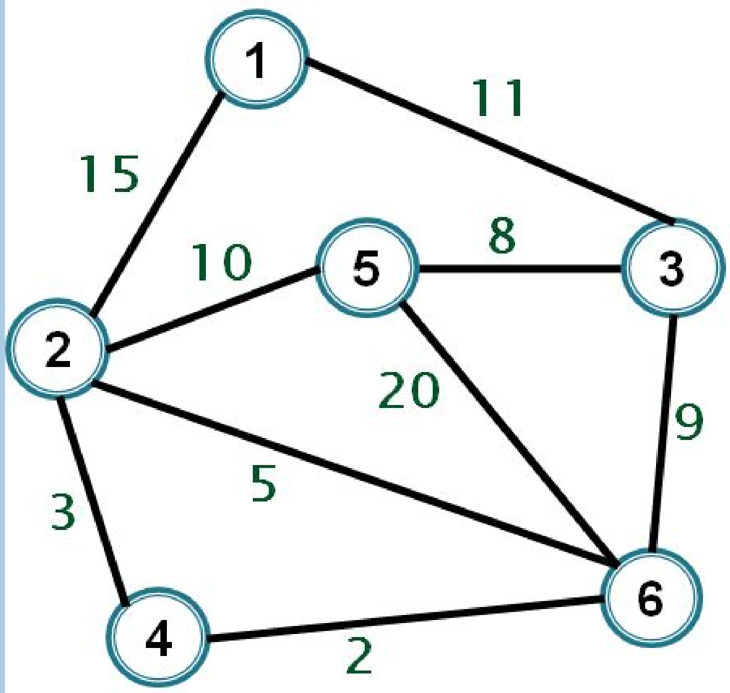


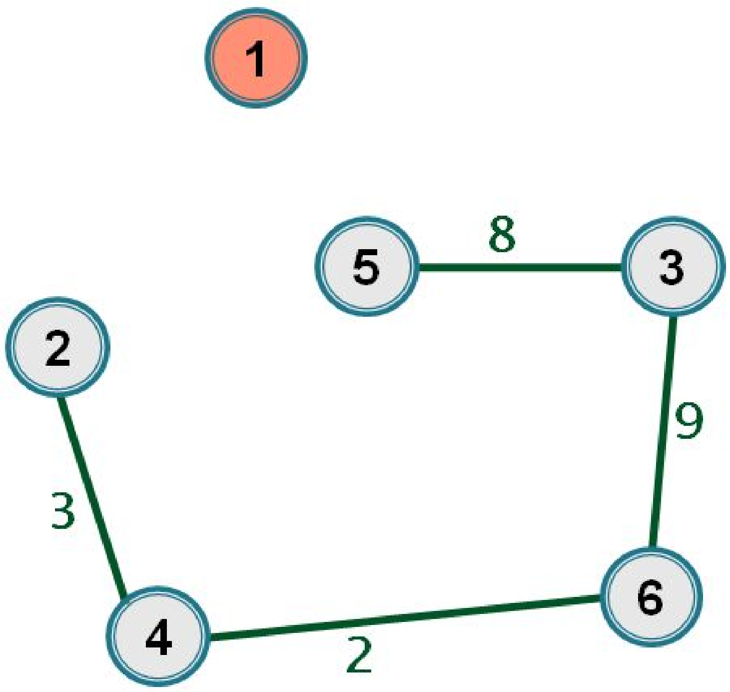
Algoritmul lui Kruskal



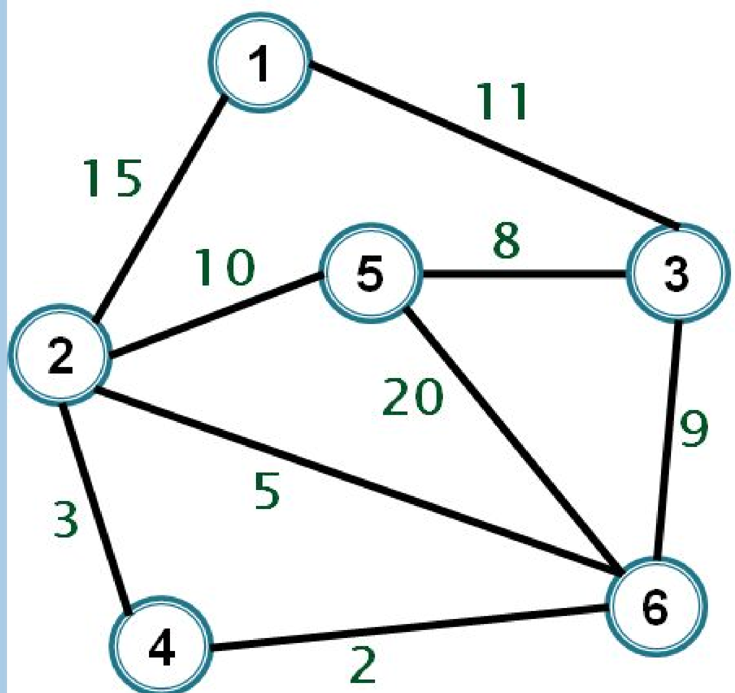


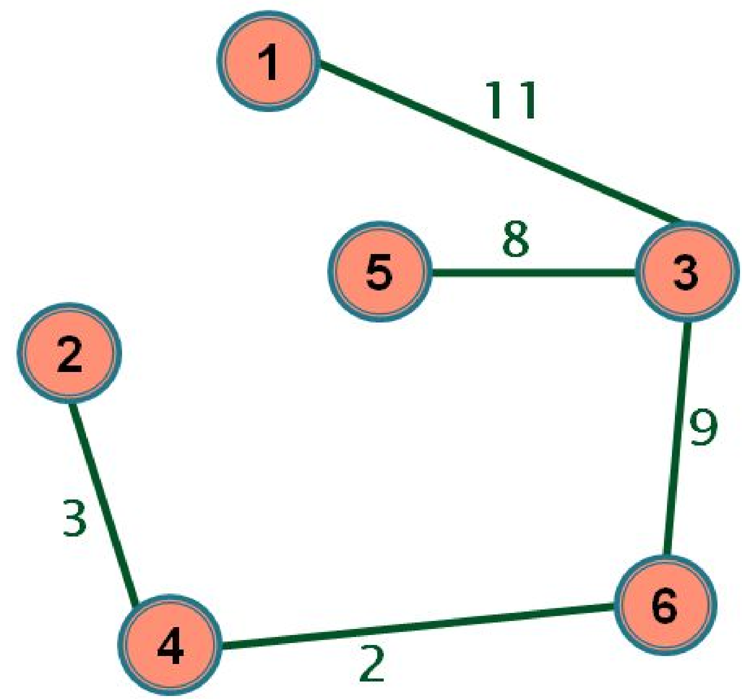
Algoritmul lui Kruskal





Algoritmul lui Kruskal





Algoritmul lui Kruskal - Corectitudine

//TODO @ Blackboard

Algoritmul lui Kruskal - Implementare

1. **Cum reprezentăm graful în memorie?**
2. **Cum selectăm ușor o muchie?**
   - de cost minim
   - care unește două componente (nu formează cicluri cu muchiile deja selectate)

Algoritmul lui Kruskal - Implementare

1. Cum reprezentăm graful în memorie?

Raspuns:

Algoritmul lui Kruskal - Implementare

1. Cum reprezentăm graful în memorie?

Raspuns: lista de muchii.

Pentru fiecare muchie reținem un triplet reprezentând extremitățile și costul muchiei.

Algoritmul lui Kruskal - Implementare

2. Cum selectăm ușor o muchie de cost minim?

Raspuns:

Algoritmul lui Kruskal - Implementare

2. Cum selectăm ușor o muchie de cost minim?

Raspuns: Inițial sortăm crescător lista de muchii după cost, apoi considerăm muchiile în această ordine.

Algoritmul lui Kruskal - Implementare

3. Cum ne asigurăm că o muchie nu închide un ciclu?

Raspuns:

Algoritmul lui Kruskal - Implementare

3. Cum ne asigurăm că o muchie nu închide un ciclu?

Raspuns: Asociem fiecărei componente conexe, un reprezentant (o culoare)

Algoritmul lui Kruskal - Implementare

Operații necesare:

- **Initializare(u)** **-** creează o componentă cu un singur vârf, u

Algoritmul lui Kruskal - Implementare

Operații necesare:

- **Initializare(u)** **-** creează o componentă cu un singur vârf, u
- **Reprez(u)** – returnează reprezentantul (culoarea) componentei care conține pe u

Algoritmul lui Kruskal - Implementare

Operații necesare:

- **Initializare(u)** **-** creează o componentă cu un singur vârf, u
- **Reprez(u)** – returnează reprezentantul (culoarea) componentei care conține pe u
- **Reuneste(u,v)** – unește componenta care conține u cu cea care conține v

Algoritmul lui Kruskal - Implementare

O **muchie (u,v) poate fi adăgată** în construcția APM-lui doar dacă:

Algoritmul lui Kruskal - Implementare

O **muchie (u,v) poate fi adăgată** în construcția APM-lui doar dacă:

Reprez(u) ≠ Reprez(v)

Algoritmul lui Kruskal - Pseudocod

```
sorteaza(E)
```

Algoritmul lui Kruskal - Pseudocod

```
sorteaza(E)
for(v=1;v<=n;v++)
    Initializare(v);
```

Algoritmul lui Kruskal - Pseudocod

```
sorteaza(E)
for(v=1;v<=n;v++)
    Initializare(v);
nrmsel=0
for(uv : E)
 if(Reprez(u)!=Reprez(v))
  {



    }
```

Algoritmul lui Kruskal - Pseudocod

```
sorteaza(E)
for(v=1;v<=n;v++)
     Initializare(v);
nrmsel=0
for(uv : E)
 if(Reprez(u)!=Reprez(v))
  {
  scrie uv;
  Reuneste(u,v);
      nrmsel=nrmsel+1;
  if(nrmsel==n-1)
   STOP; //break;
   }
```

Algoritmul lui Kruskal - Complexitate

Complexitate

Sortare $\rightarrow$ O(m log m) **=** O(m log n)

n * Initializare

2m * Reprez

(n-1) * Reuneste

Algoritmul lui Kruskal - Varianta 1

Varianta 1 – memorăm într-un vector pentru fiecare vârf reprezentantul/culoarea componentei din care face parte

r[u] = culoarea componentei care conține vârful u

Algoritmul lui Kruskal - Varianta 1

Initializare - O(1)

```
void Initializare(int u){
    r[u]=u;}
```

Reprez

Reuneste

Algoritmul lui Kruskal - Varianta 1

Initializare - O(1)

```
void Initializare(int u){
                    r[u]=u;}
```

Reprez - O(1)

```
int Reprez(int u){
    return r[u];
}
```

Reuneste

Algoritmul lui Kruskal - Varianta 1

Initializare - O(1)

```
void Initializare(int u){
                r[u]=u;}
```

Reprez - O(1)

```
int Reprez(int u){
    return r[u];
}
```

Reuneste – O(n)

```
void Reuneste(int u,int v)
{
  r1=Reprez(u);//r1=r[u]
  r2=Reprez(v);//r2=r[v]
  for(k=1;k<=n;k++)
    if(r[k]==r2)
       r[k]=r1;
}
```

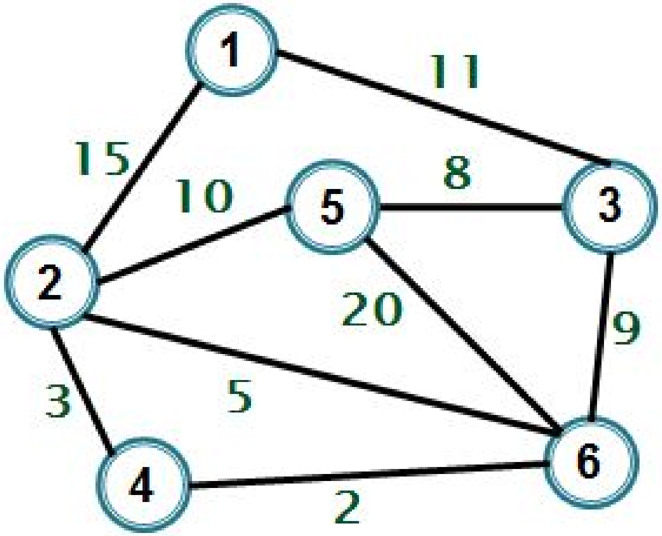
Algoritmul lui Kruskal - Complexitate 1

Varianta 1– dacă folosim vector de reprezentanți

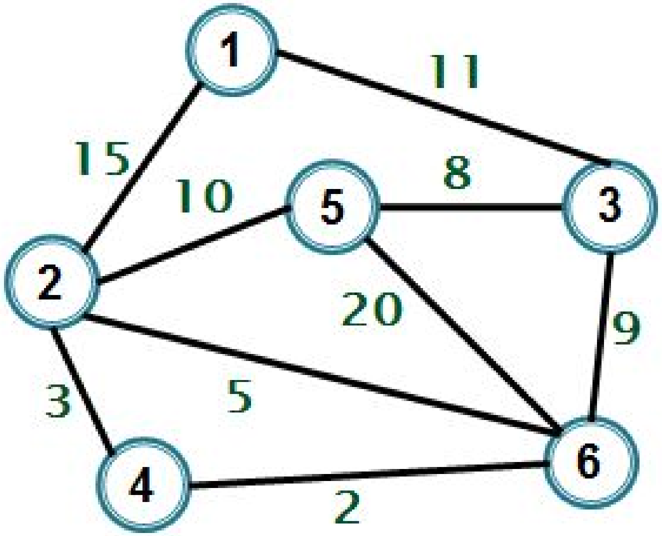
- **Sortare** -> $O(m \log m) = O(m \log n)$
- **n * Initializare** -> $O(n)$
- **2m * Reprez** -> $O(m)$
- **(n–1) * Reuneste** -> $O(n^2)$

$O(m \log n + n^2)$

Algoritmul lui Kruskal - Simulare 1

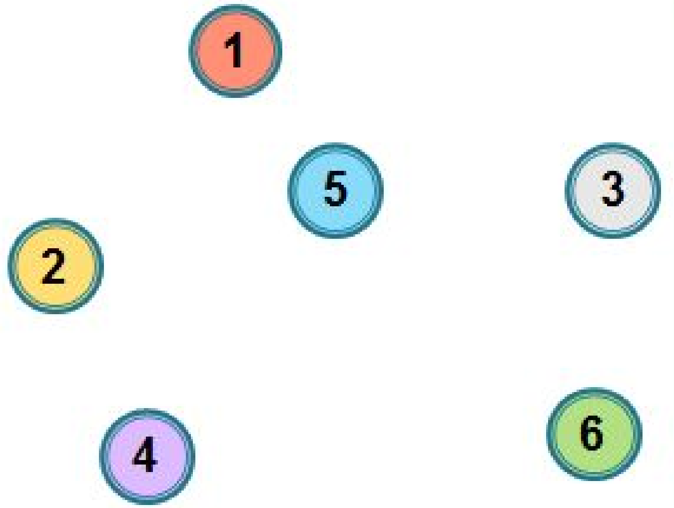


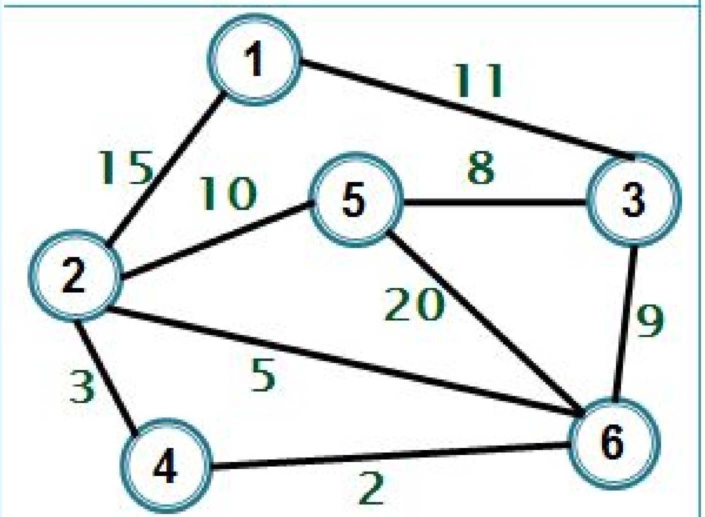
Algoritmul lui Kruskal - Simulare 1



(4,6)
(2,4)
(2,6)
(3,5)
(3,6)
(2,5)
(1,3)
(1,2)
(5,6)

Algoritmul lui Kruskal - Simulare 1

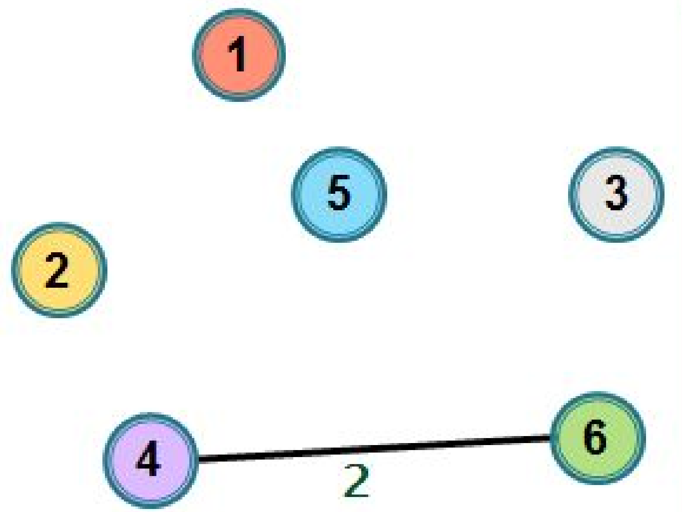


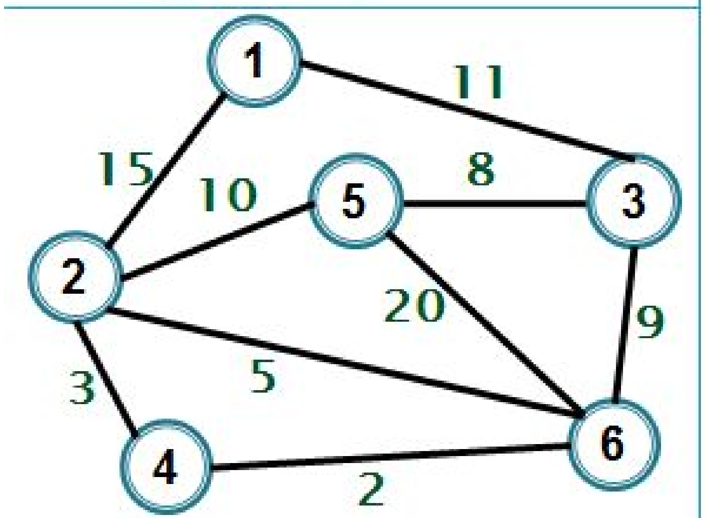


r = [1,2,3,4,5,6]

(4,6)
(2,4)
(2,6)
(3,5)
(3,6)
(2,5)
(1,3)
(1,2)
(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,<u>4</u>,5,<u>6</u>]

(4,6) r(4) ≠ r(6)

(2,4)

(2,6)

(3,5)

(3,6)

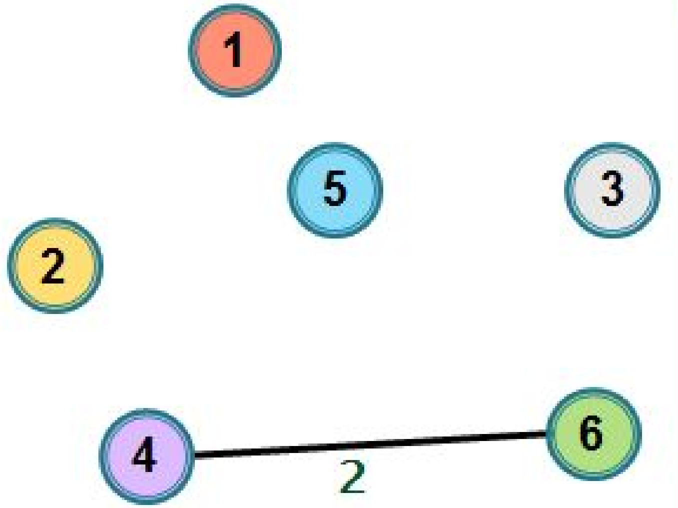
(2,5)

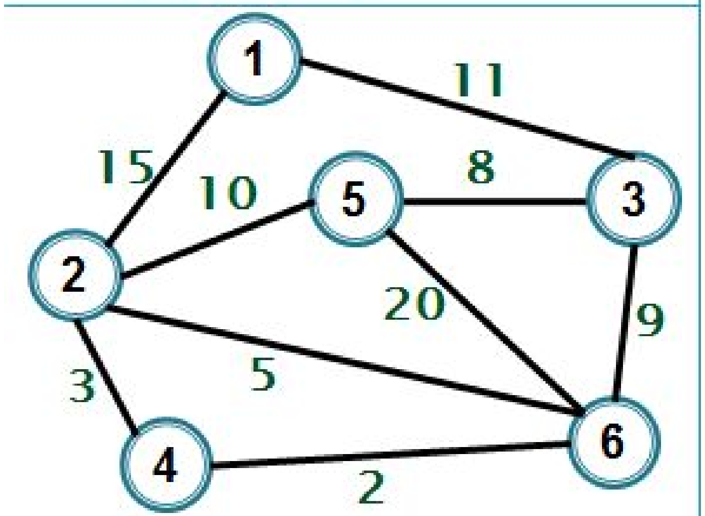
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) Reuneste(4, 6)

(2,4)

(2,6)

(3,5)

(3,6)

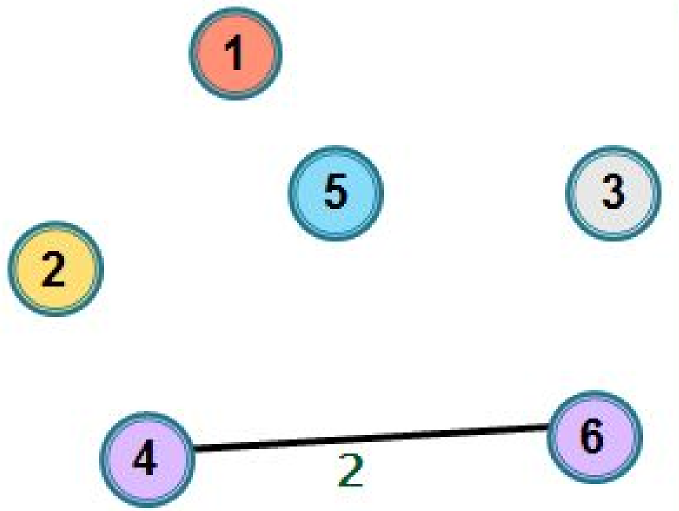
(2,5)

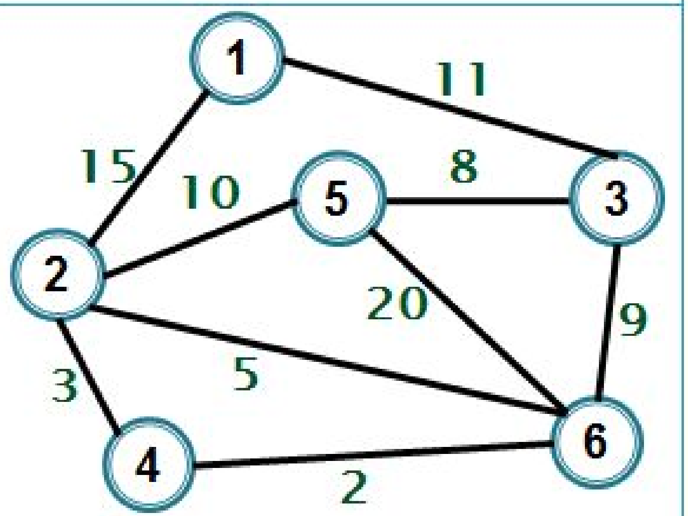
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4)

(2,6)

(3,5)

(3,6)

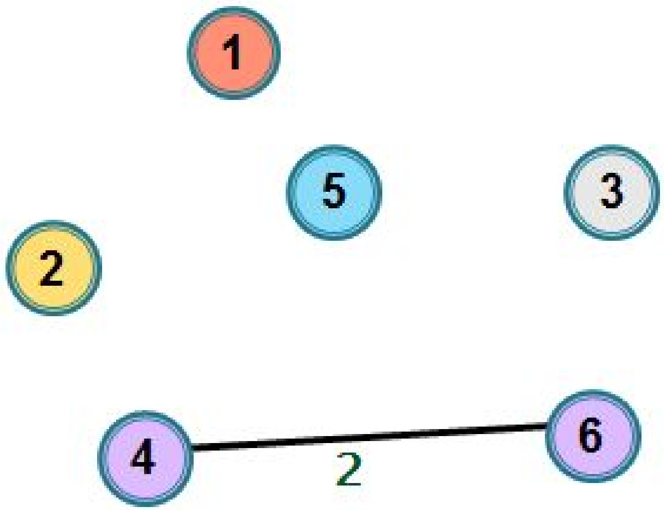
(2,5)

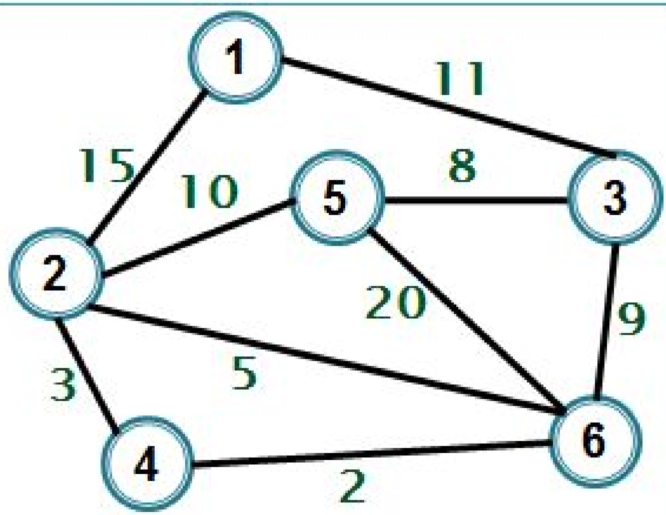
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4)

(2,6)

(3,5)

(3,6)

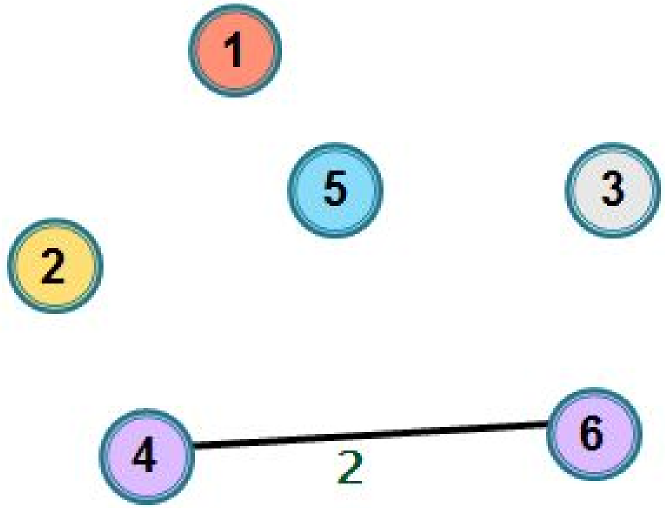
(2,5)

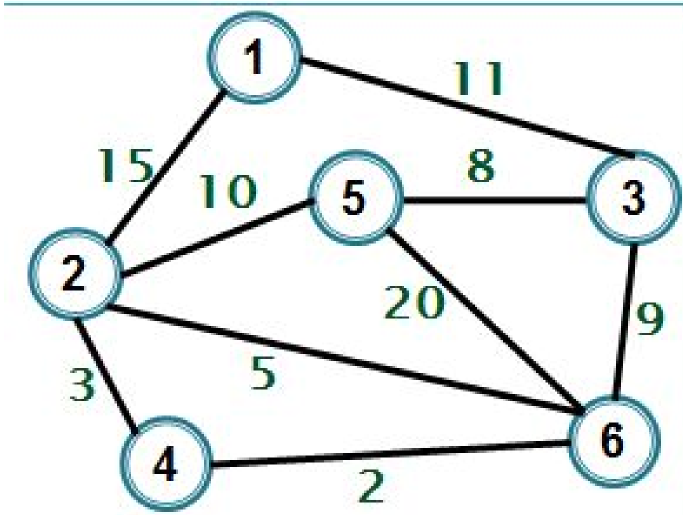
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) $r(2) \neq r(4)$

(2,6)

(3,5)

(3,6)

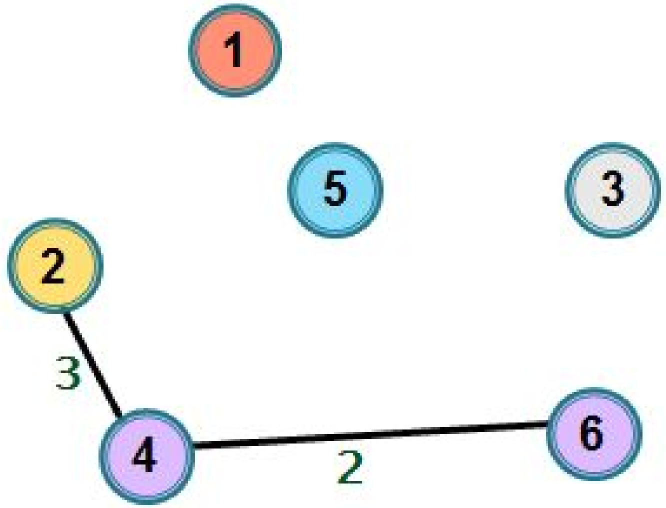
(2,5)

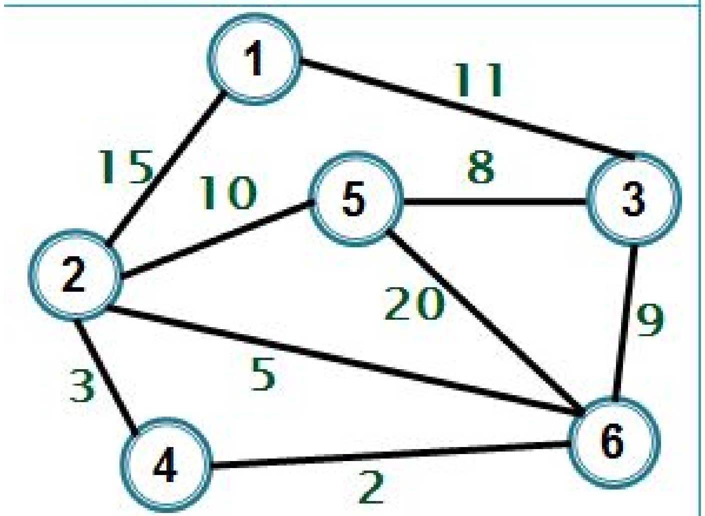
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4)

(2,6)

(3,5)

(3,6)

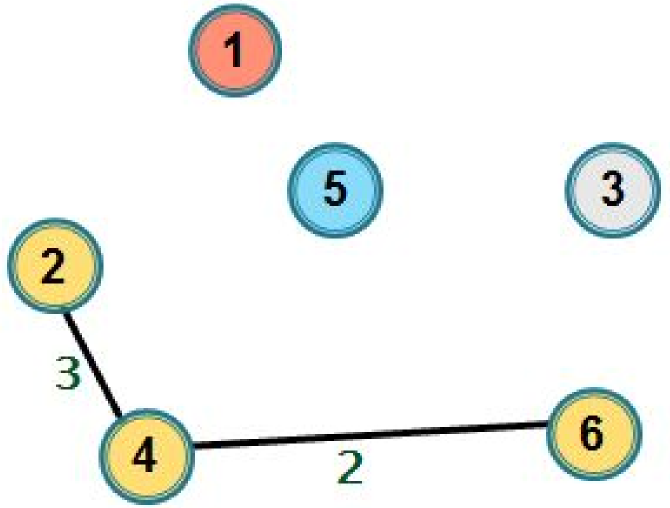
(2,5)

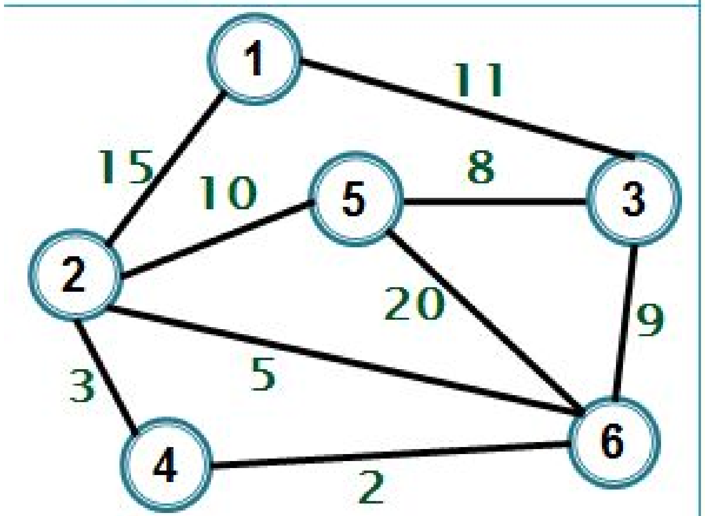
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6)

(3,5)

(3,6)

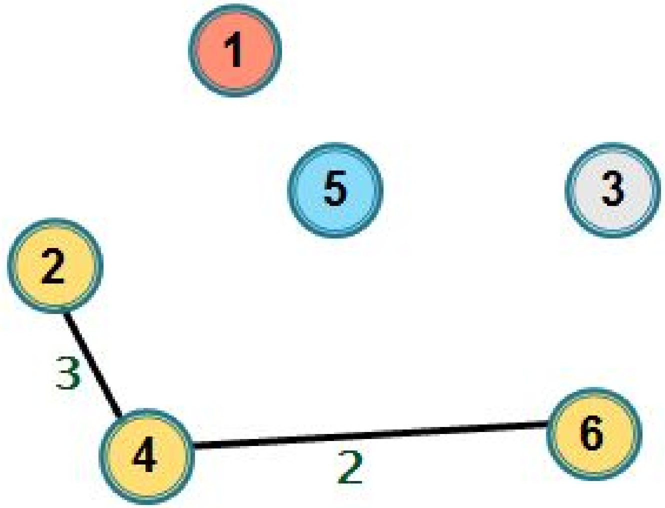
(2,5)

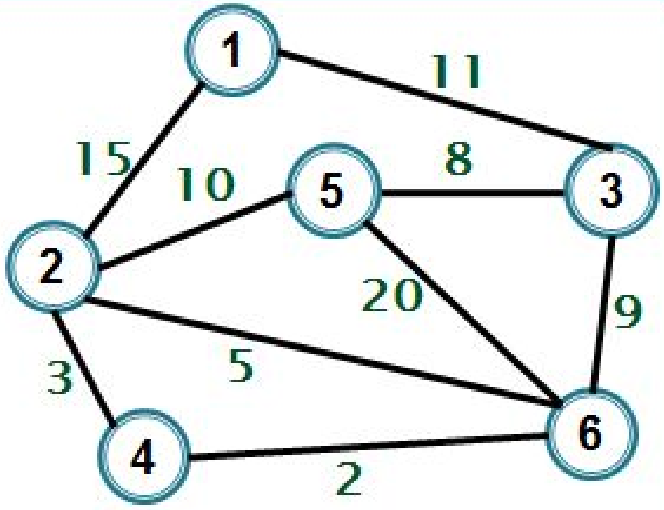
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6)

(3,5)

(3,6)

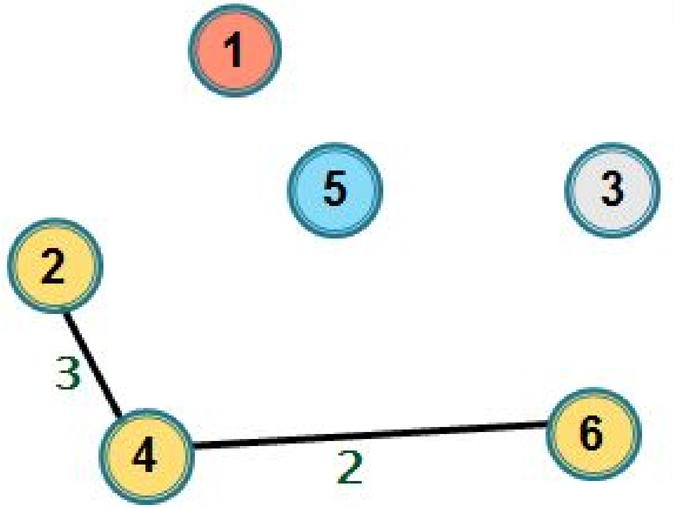
(2,5)

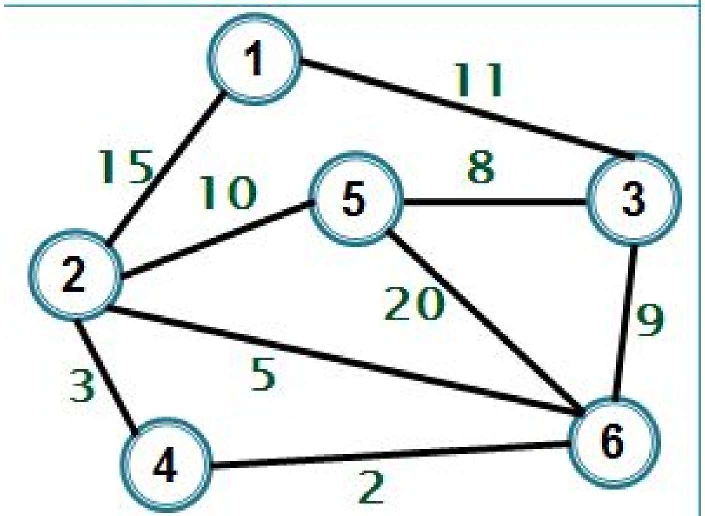
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6) r(2) =r(6) –> NU

(3,5)

(3,6)

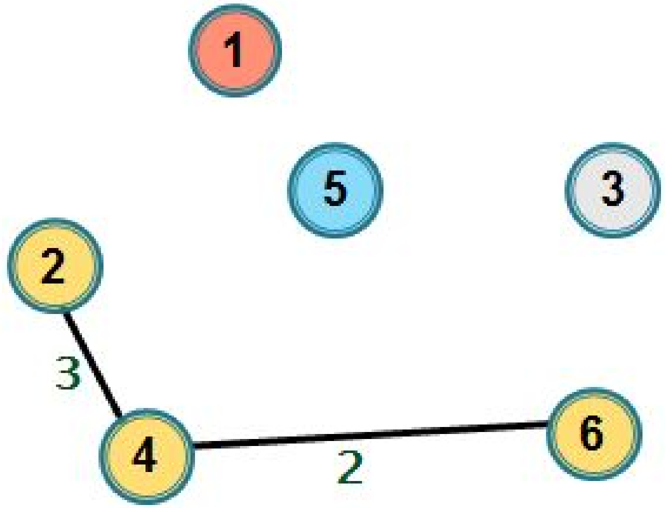
(2,5)

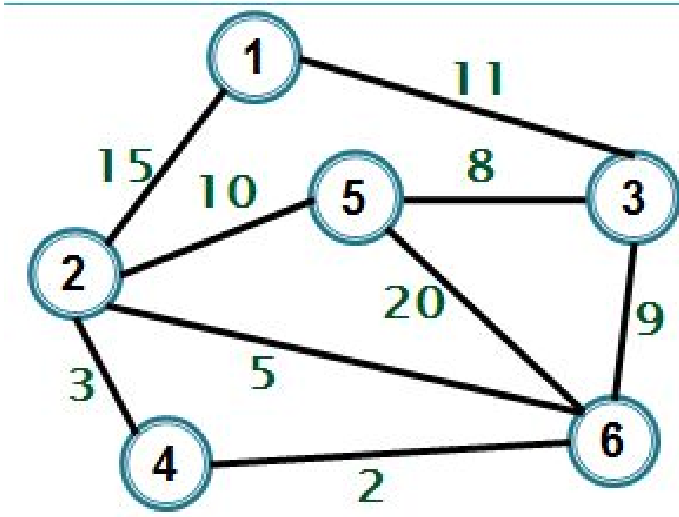
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6) r(2) =r(6) –> NU

(3,5)

(3,6)

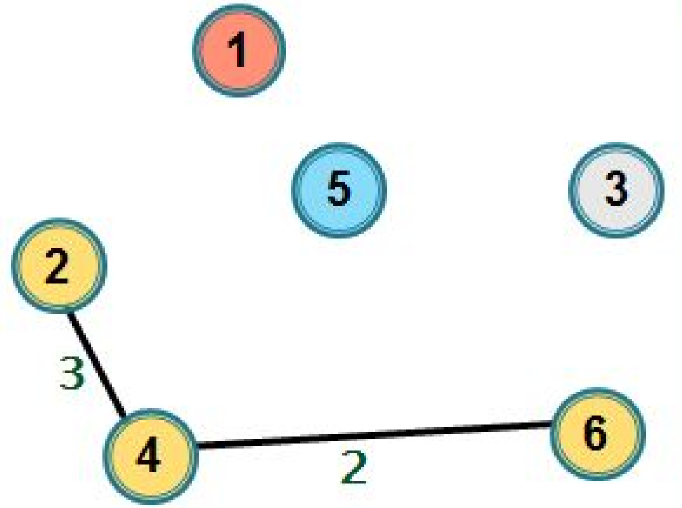
(2,5)

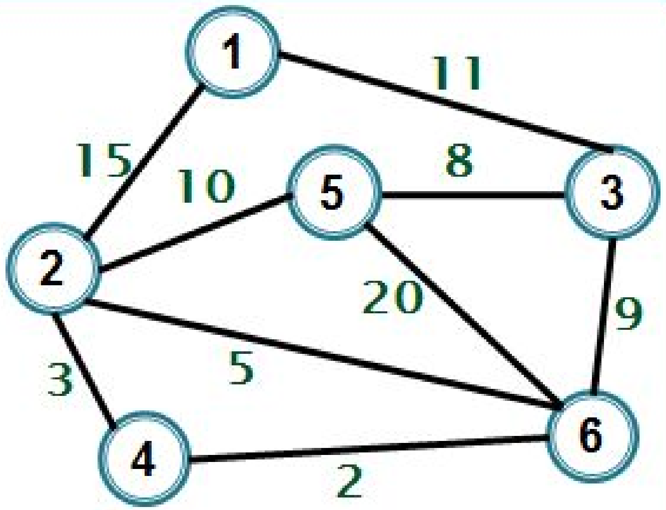
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6) r(2) = r(6) -> NU

(3,5) $r(3) \neq r(5)$

(3,6)

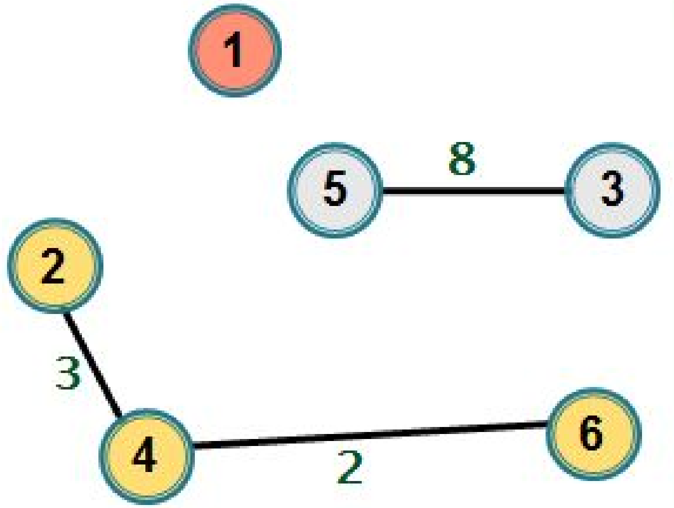
(2,5)

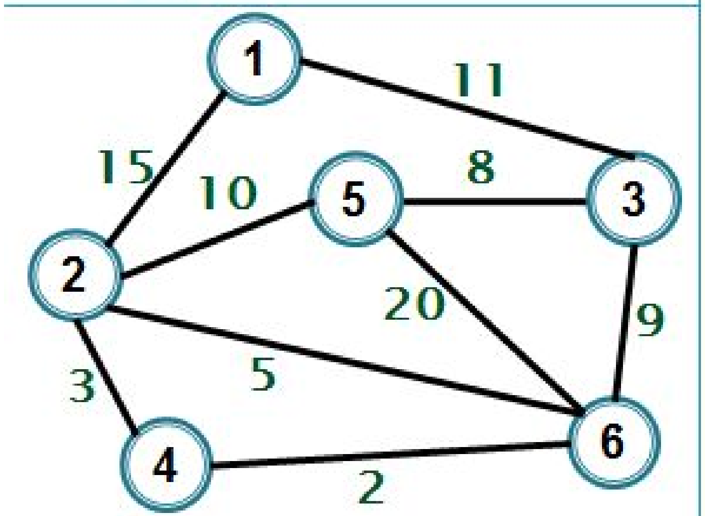
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6) r(2) = r(6) -> NU

(3,5) r = [1,2,3,2,3,2]

(3,6)

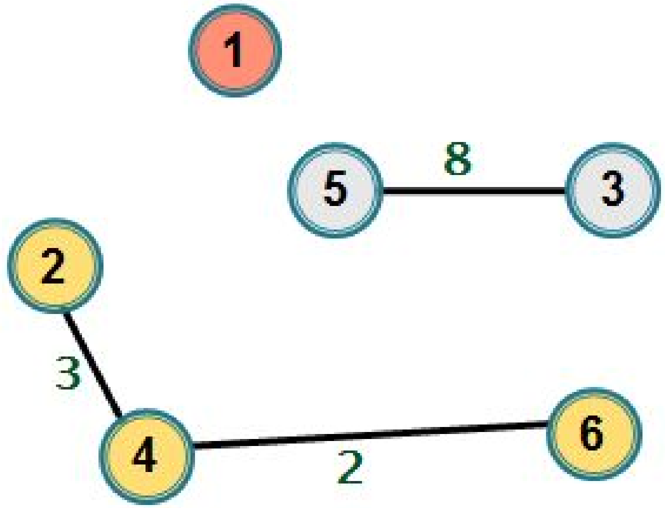
(2,5)

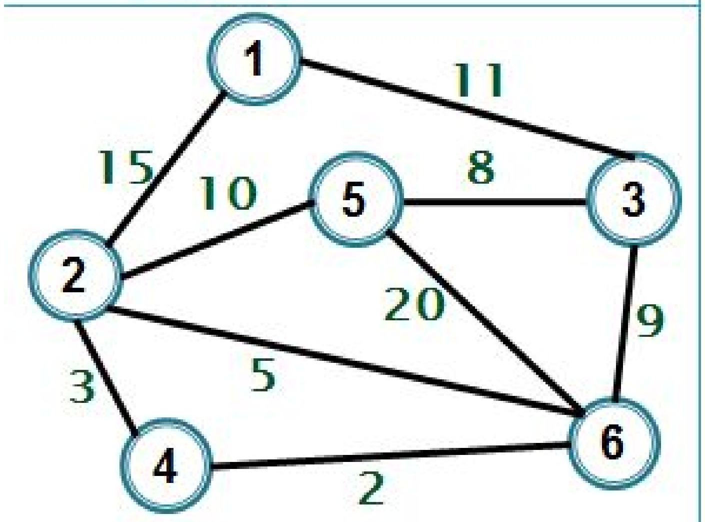
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6) r(2) =r(6) -> NU

(3,5) r = [1,2,3,2,3,2]

(3,6)

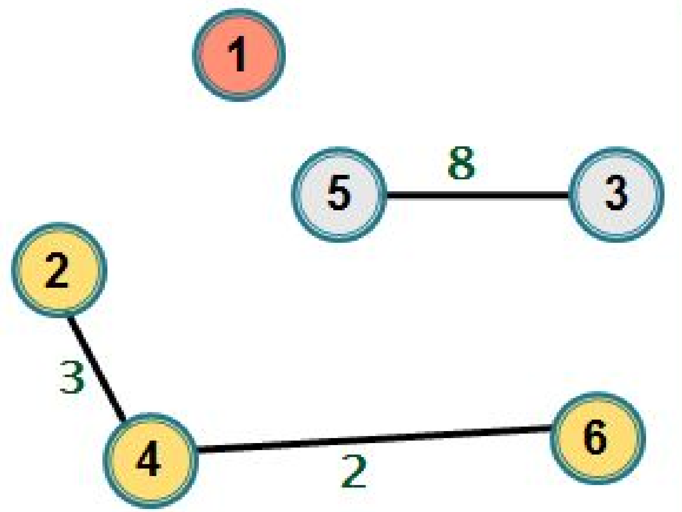
(2,5)

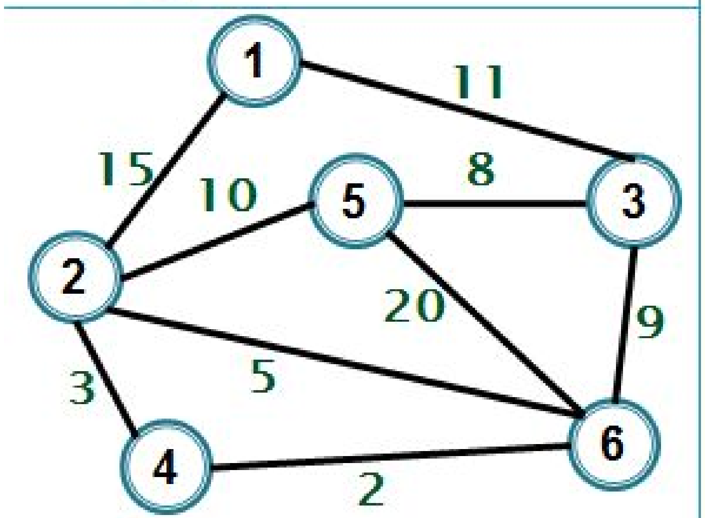
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6) r(2) = r(6) -> NU

(3,5) r = [1,2,3,2,3,2]

(3,6) $r(3) \neq r(6)$

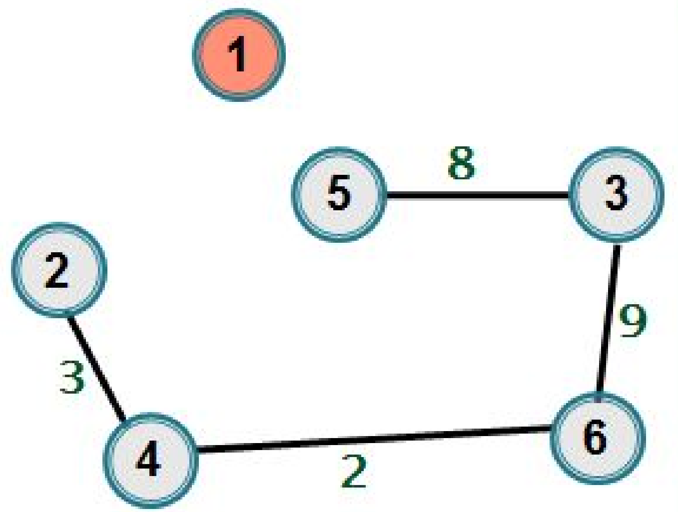
(2,5)

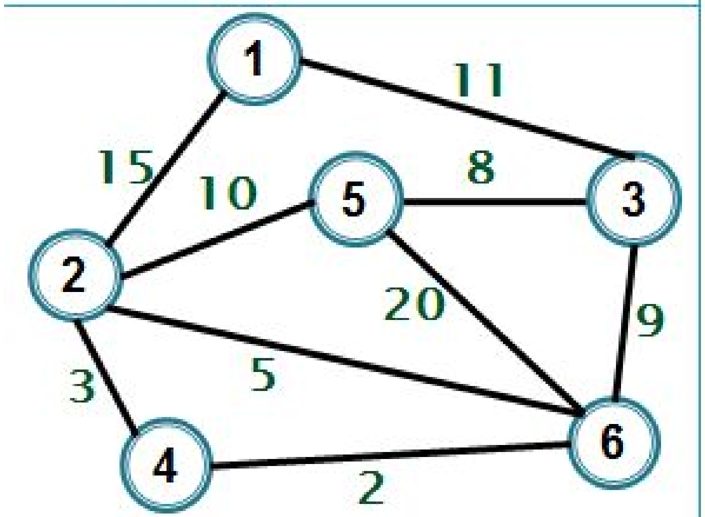
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6) r(2) =r(6) -> NU

(3,5) r = [1,2,3,2,3,2]

(3,6) r = [1,3,3,3,3,3]

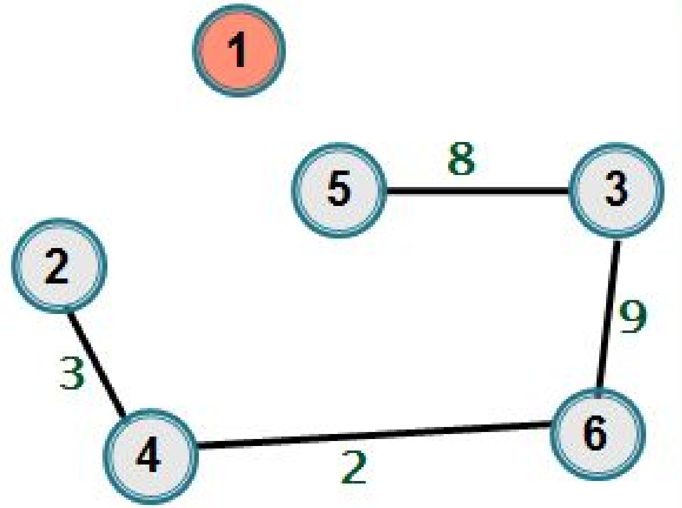
(2,5)

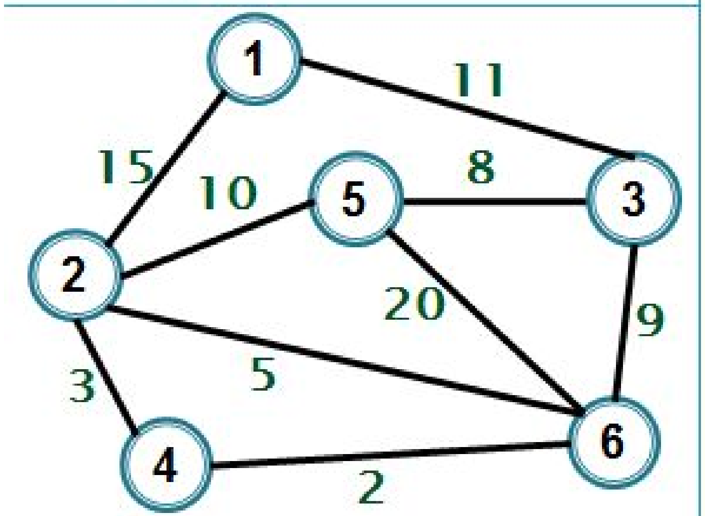
(1,3)

(1,2)

(5,6)

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6) r(2) = r(6) -> NU

(3,5) r = [1,2,3,2,3,2]

(3,6) r = [1,3,3,3,3,3]

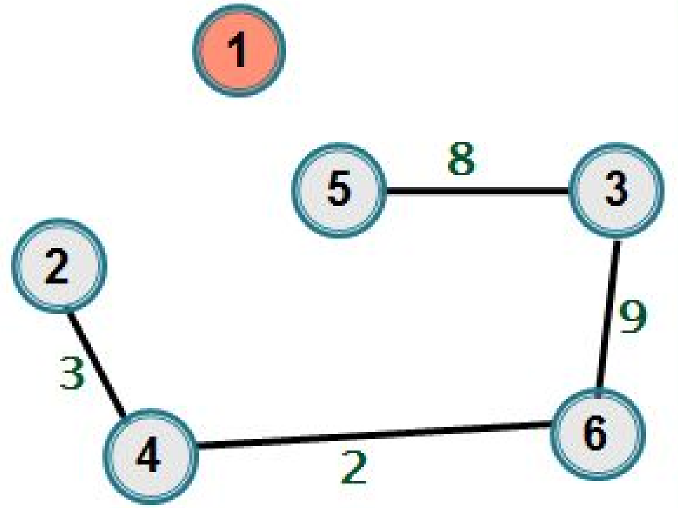
(2,5)

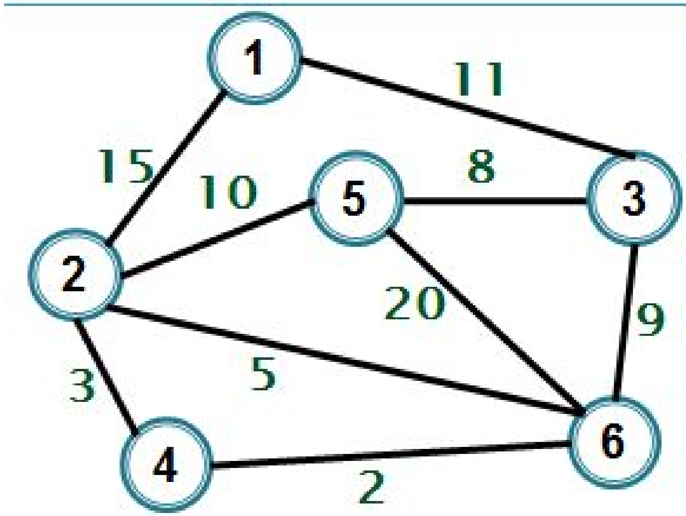
(1,3)

(1,2)

(5,6)

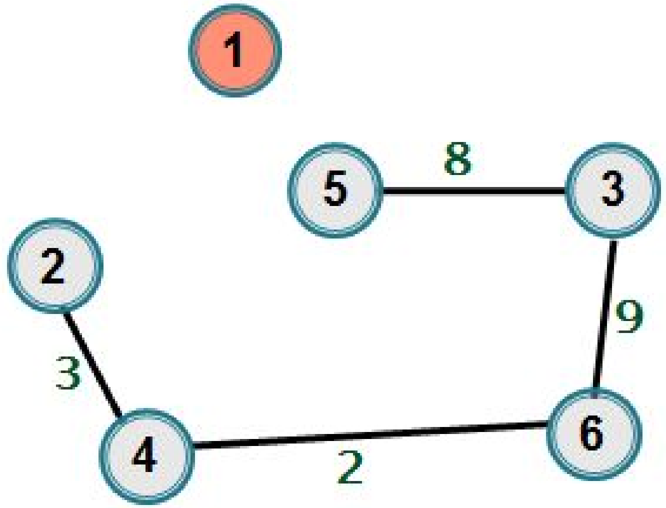
Algoritmul lui Kruskal - Simulare 1

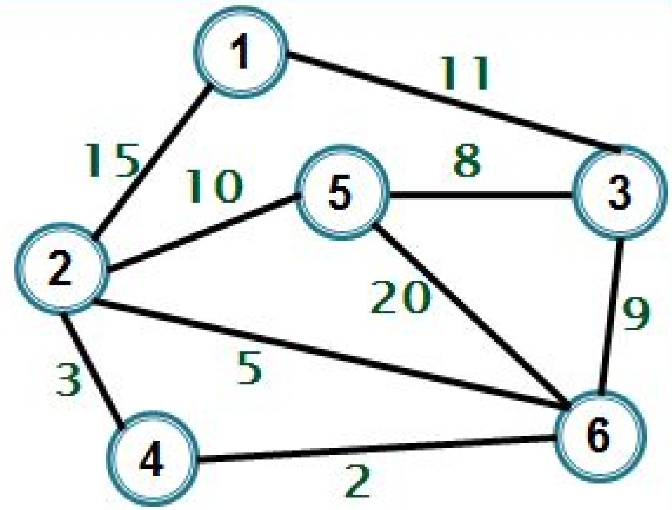




	r = [1,2,3,4,5,6]
(4,6)	r = [1,2,3,4,5,4]
(2,4)	r = [1,2,3,2,5,2]
(2,6)	r(2) =r(6) -> NU
(3,5)	r = [1,2,3,2,3,2]
(3,6)	r = [1,3,3,3,3,3]
(2,5)	r(2) =r(5) -> NU
(1,3)	
(1,2)	
(5,6)	

Algoritmul lui Kruskal - Simulare 1





r = [1,2,3,4,5,6]

(4,6) r = [1,2,3,4,5,4]

(2,4) r = [1,2,3,2,5,2]

(2,6) r(2) =r(6) −> NU

(3,5) r = [1,2,3,2,3,2]

(3,6) r = [1,3,3,3,3,3]

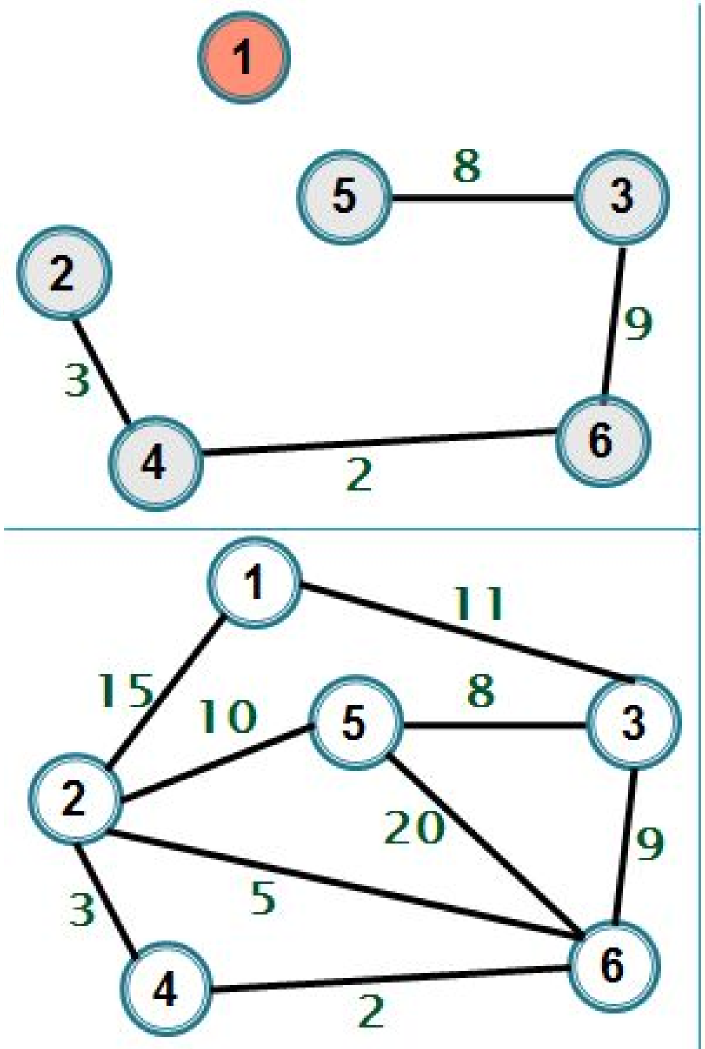
(2,5) r(2) =r(5) −> NU

(1,3)

(1,2)

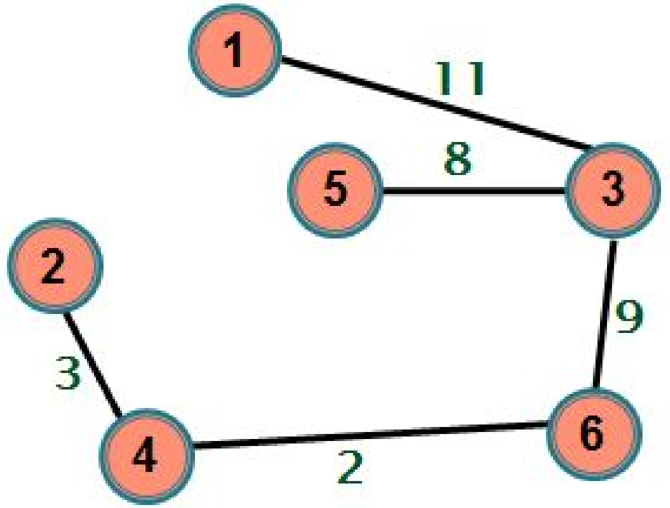
(5,6)

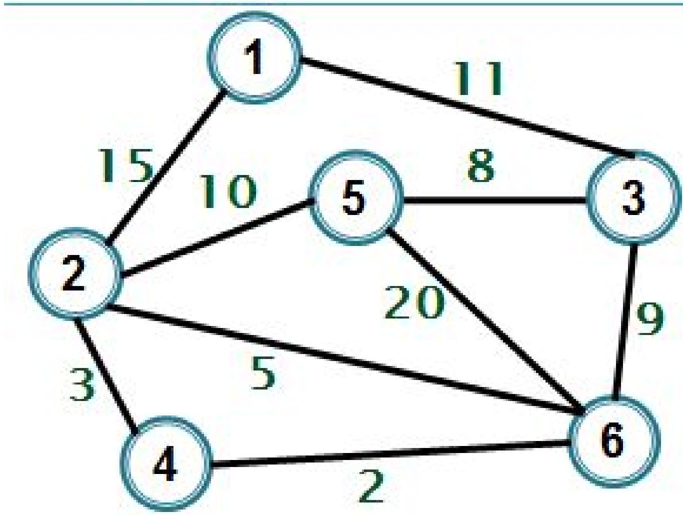
Algoritmul lui Kruskal - Simulare 1



	$r = [1,2,3,4,5,6]$
(4,6)	$r = [1,2,3,4,5,4]$
(2,4)	$r = [1,2,3,2,5,2]$
(2,6)	$r(2) = r(6) \to$ NU
(3,5)	$r = [1,2,3,2,3,2]$
(3,6)	$r = [1,3,3,3,3,3]$
(2,5)	$r(2) = r(5) \to$ NU
(1,3)	$r(1) \neq r(3)$
(1,2)	
(5,6)	

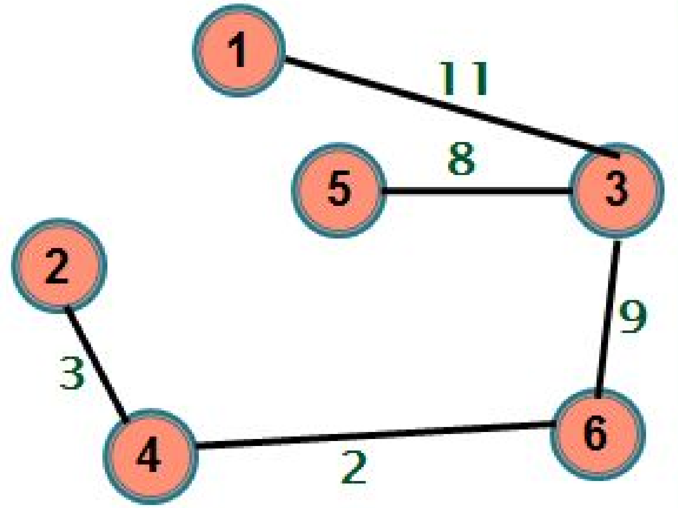
Algoritmul lui Kruskal - Simulare 1

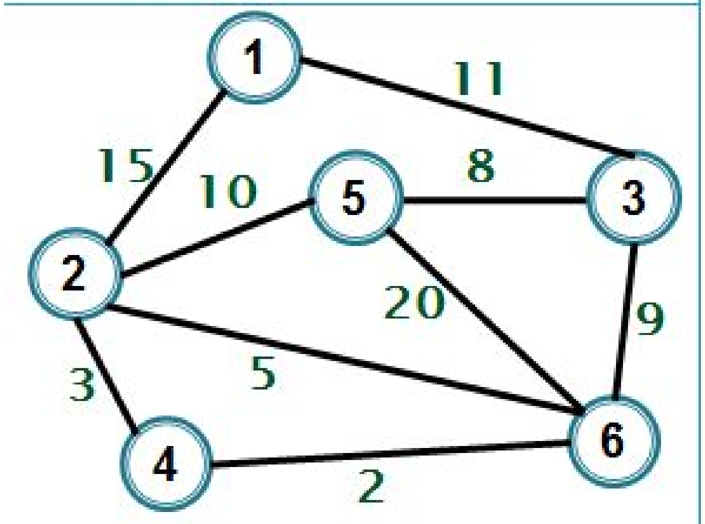




	r = [1,2,3,4,5,6]
(4,6)	r = [1,2,3,4,5,4]
(2,4)	r = [1,2,3,2,5,2]
(2,6)	r(2) = r(6) -> NU
(3,5)	r = [1,2,3,2,3,2]
(3,6)	r = [1,3,3,3,3,3]
(2,5)	r(2) = r(5) -> NU
(1,3)	r = [1,1,1,1,1,1]
(1,2)	
(5,6)	

Algoritmul lui Kruskal - Simulare 1





	r = [1,2,3,4,5,6]
(4,6)	r = [1,2,3,4,5,4]
(2,4)	r = [1,2,3,2,5,2]
(2,6)	r(2) = r(6) -> NU
(3,5)	r = [1,2,3,2,3,2]
(3,6)	r = [1,3,3,3,3,3]
(2,5)	r(2) = r(5) -> NU
(1,3)	r = [1,1,1,1,1,1]

<u>STOP</u>

(1,2)

(5,6)

Algoritmul lui Kruskal

Observație:

Reprezentarea cu vector de culori este ineficientă

Algoritmul lui Kruskal - Varianta 2

Soluție:

Structuri pentru mulțimi disjuncte Union/Find - <u>arbori</u>

Algoritmul lui Kruskal - Varianta 2

Soluție:

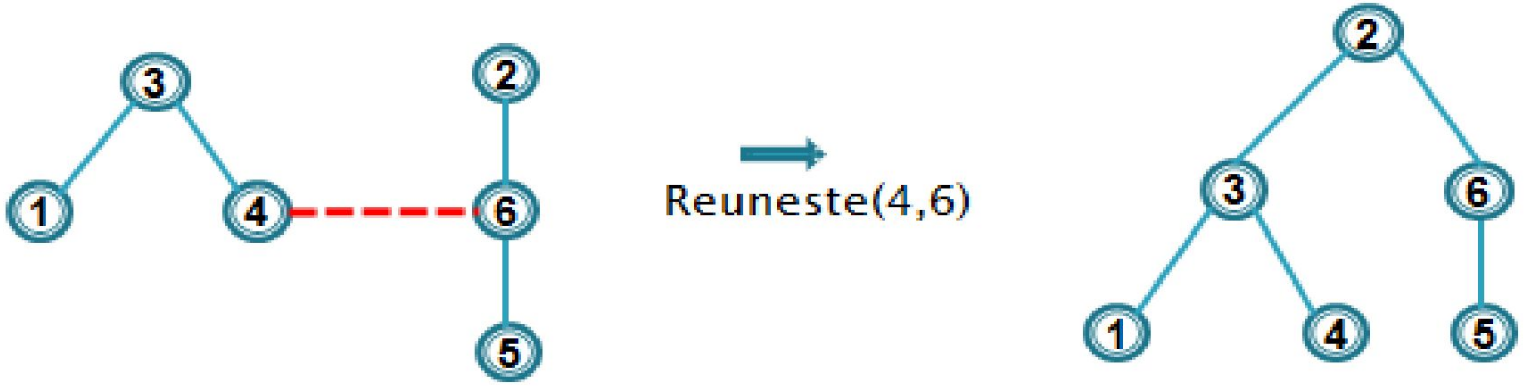
Structuri pentru mulțimi disjuncte Union/Find - arbori

- memorăm componentele conexe ca arbori, folosind **vectorul tata**; **reprezentantul componentei va fi rădăcina arborelui**

Algoritmul lui Kruskal - Varianta 2

Soluție:

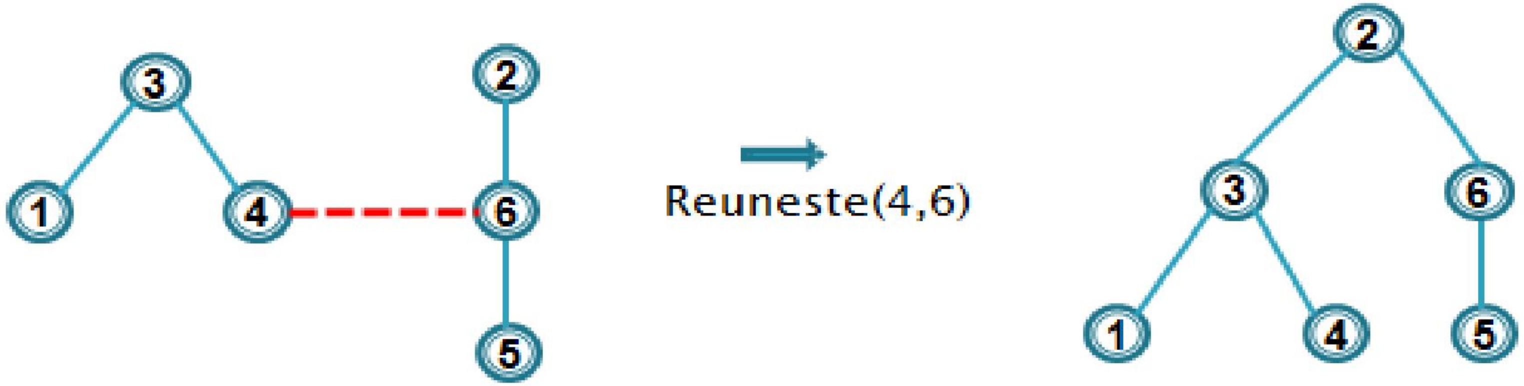
- **Reuniunea se va face în funcţie de înălţimea arborilor (reuniune ponderată) ⇒ arbori de înălțime logaritmică**



Algoritmul lui Kruskal - Varianta 2

Soluție:

arborele cu înălţimea mai mică devine subarbore al rădăcinii celuilalt arbore



Algoritmul lui Kruskal - Varianta 2

Complexitate – dacă folosim arbori

Sortare $\rightarrow$ O(m log m) **=** O(m log n)

n * Initializare $\rightarrow$

2m * Reprez $\rightarrow$

(n-1) * Reuneste $\rightarrow$

Algoritmul lui Kruskal - Varianta 2

Complexitate – dacă folosim arbori

Sortare $\rightarrow$ O(m log m) **=** O(m log n)

n * Initializare $\rightarrow$ O(n)

2m * Reprez $\rightarrow$ O(m log n)

(n-1) * Reuneste $\rightarrow$ O(m log n)

Aplicaţie: k-Clustering

Gruparea unor obiecte în k clase cât mai *bine separate* (k dat)

- obiecte din clase diferite să fie cât mai diferite

//TODO Defninire Cluster - prop, etc

Aplicație: k-Clustering

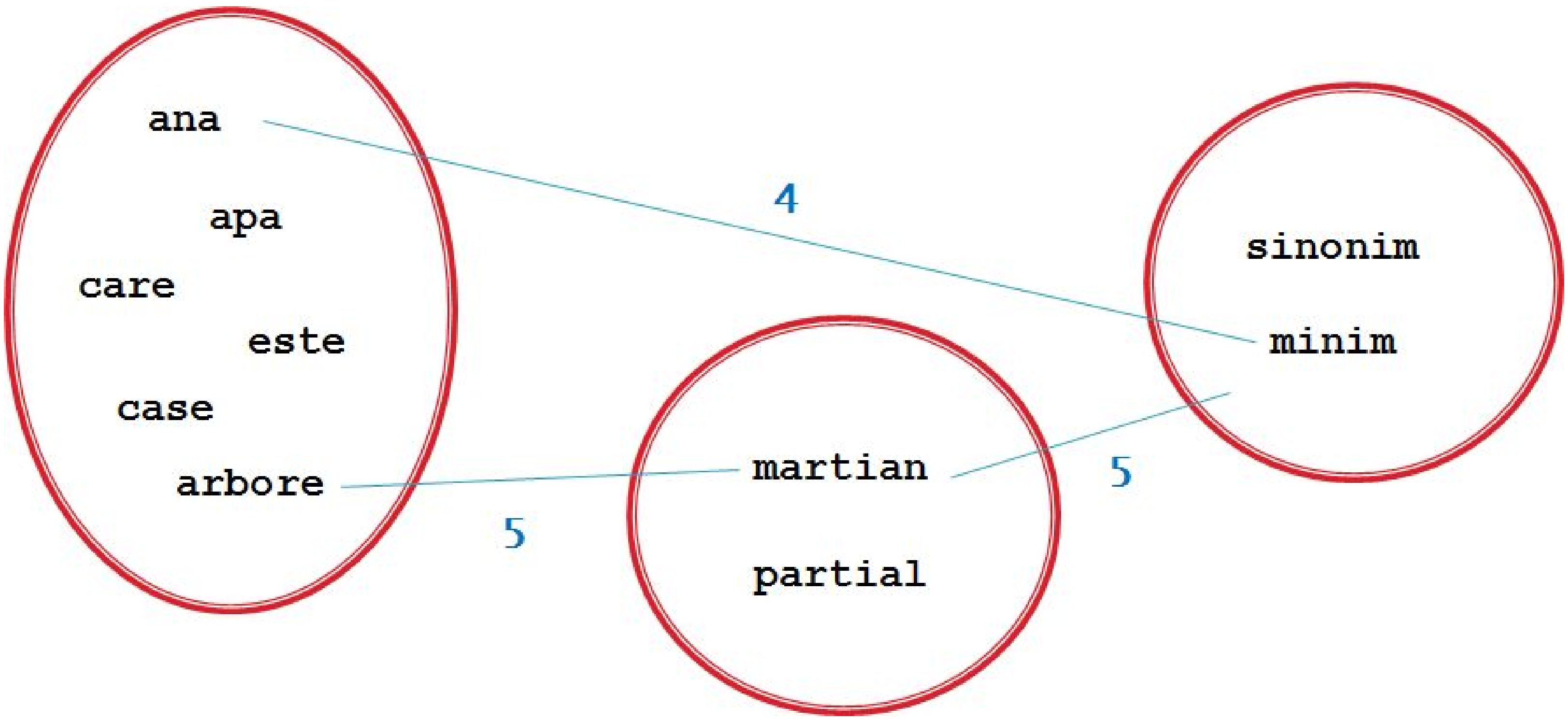
Cuvinte - Distanța de editare

care
martian
este
apa
ana
sinonim
minim
partial
arbore
case

K = 3 clustere

Aplicație: k-Clustering

Cuvinte - Distanța de editare



K = 3 clustere

Aplicaţie: k-Clustering

Idee:

- Cuvintele pot fi asemănate unor noduri

Aplicație: k-Clustering

Idee:

- Cuvintele pot fi asemănate unor noduri
- Distanța de editare dintre două cuvinte poate fi văzută ca un cost al muchiei dintre noduri

Aplicație: k-Clustering

Idee:

- Cuvintele pot fi asemănate unor noduri
- Distanța de editare dintre două cuvinte poate fi văzută ca un cost al muchiei dintre noduri
- Inițial fiecare cuvânt face parte din propriul cluster

Aplicație: k-Clustering

Idee:

- Cuvintele pot fi asemănate unor noduri
- Distanța de editare dintre două cuvinte poate fi văzută ca un cost al muchiei dintre noduri
- Inițial fiecare cuvânt face parte din propriul cluster
- La fiecare pas, determinăm cele mai apropiate două cuvinte aflate în cluster diferite și reunim cele două clustre intr-unul singur

Aplicație: k-Clustering

Idee:

- Cuvintele pot fi asemănate unor noduri
- Distanța de editare dintre două cuvinte poate fi văzută ca un cost al muchiei dintre noduri
- Inițial fiecare cuvânt face parte din propriul cluster
- La fiecare pas, determinăm cele mai apropiate două cuvinte aflate în cluster diferite și reunim cele două clustre intr-unul singur
- n-k pași

Aplicație: k-Clustering

Corectitudine:

//TODO @ Blackboard

Aplicație: k-Clustering

care

martian

este

apa

ana

sinonim

minim

partial

arbore

case

K = 3 clustere

Aplicație: k-Clustering

care $\underset{1}{\text{———}}$ case

martian

este

apa

ana

sinonim

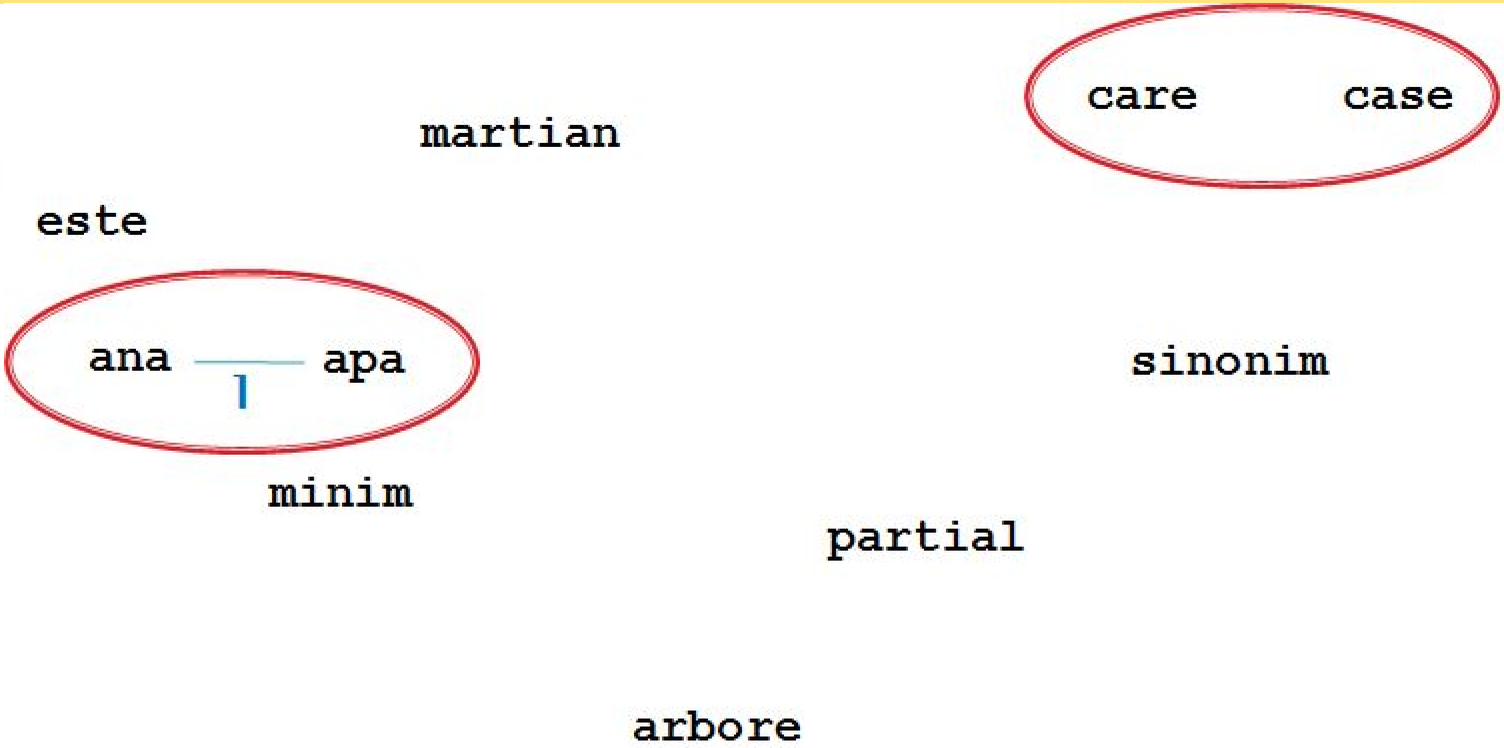
minim

partial

arbore

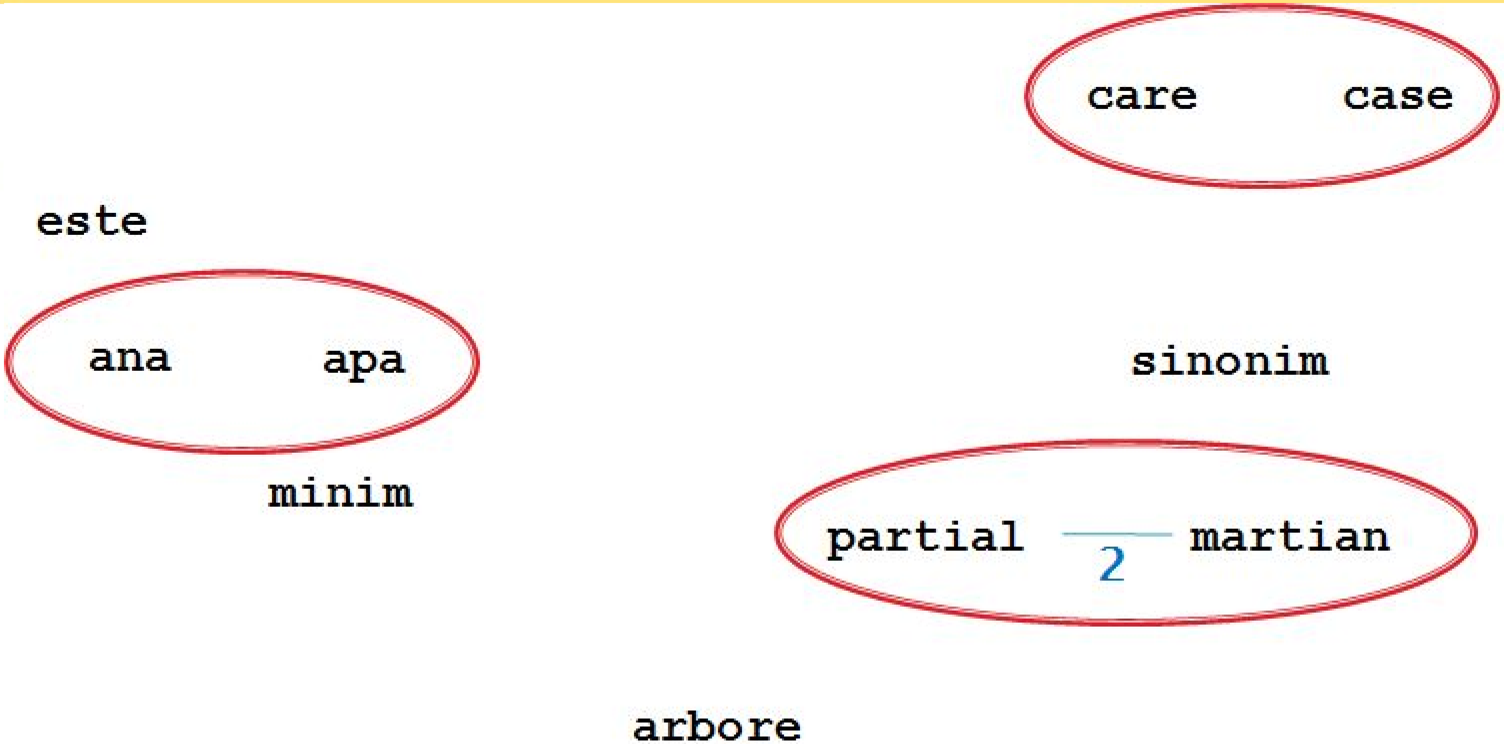
K = 3 clustere

Aplicație: k-Clustering



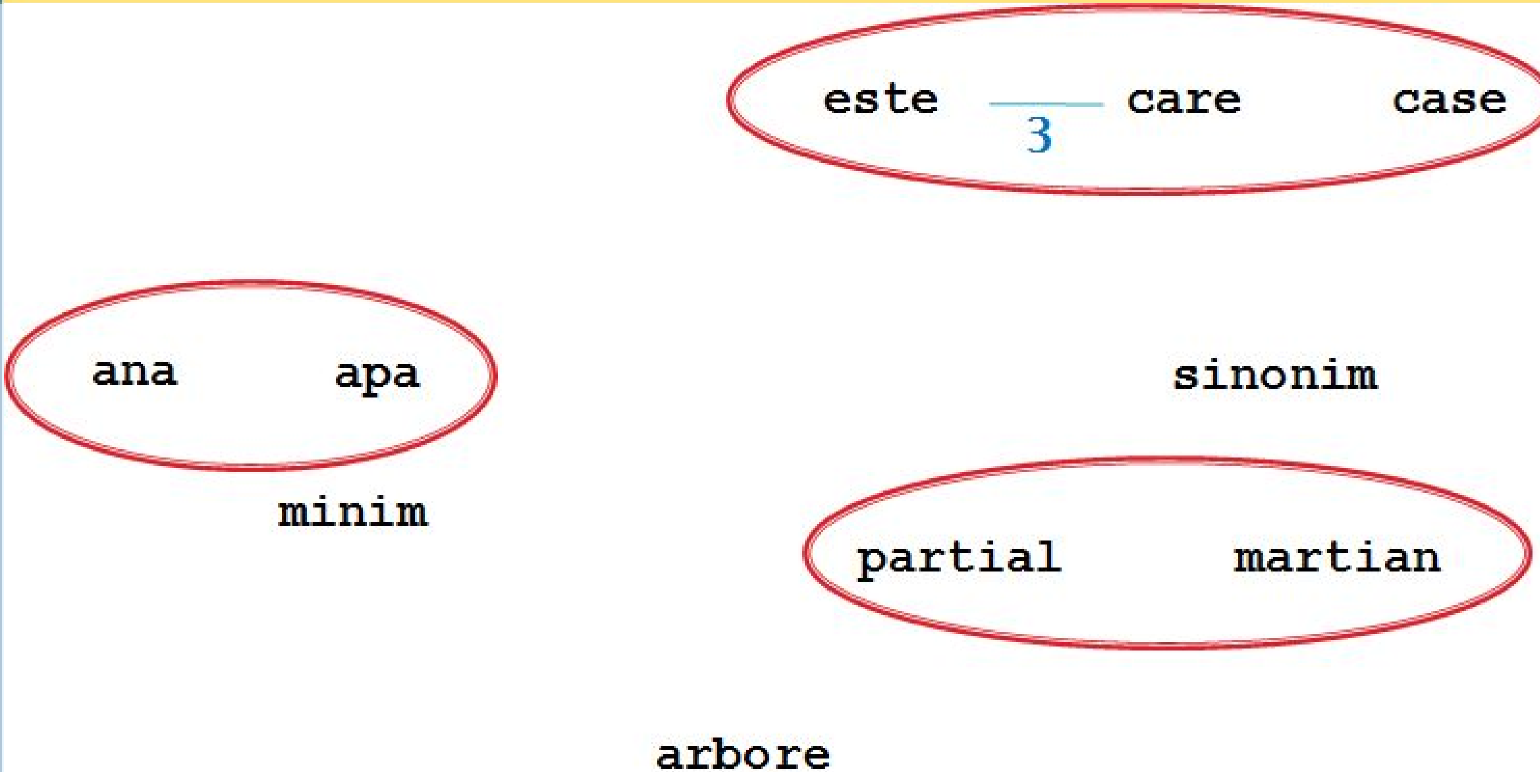
K = 3 clustere

Aplicație: k-Clustering



K = 3 clustere

Aplicație: k-Clustering



K = 3 clustere

Aplicație: k-Clustering

este care case

ana apa

sinonim

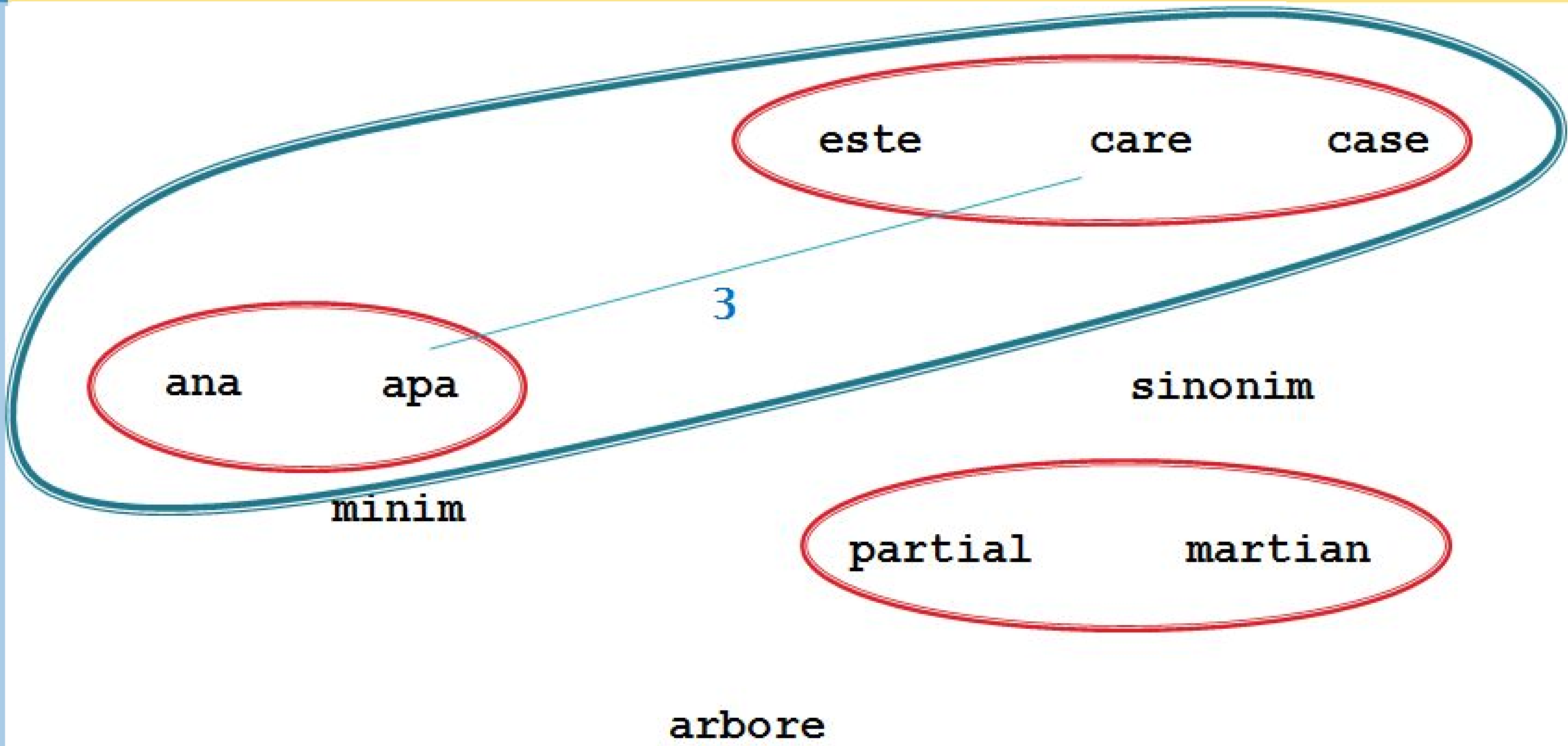
minim

partial martian

arbore

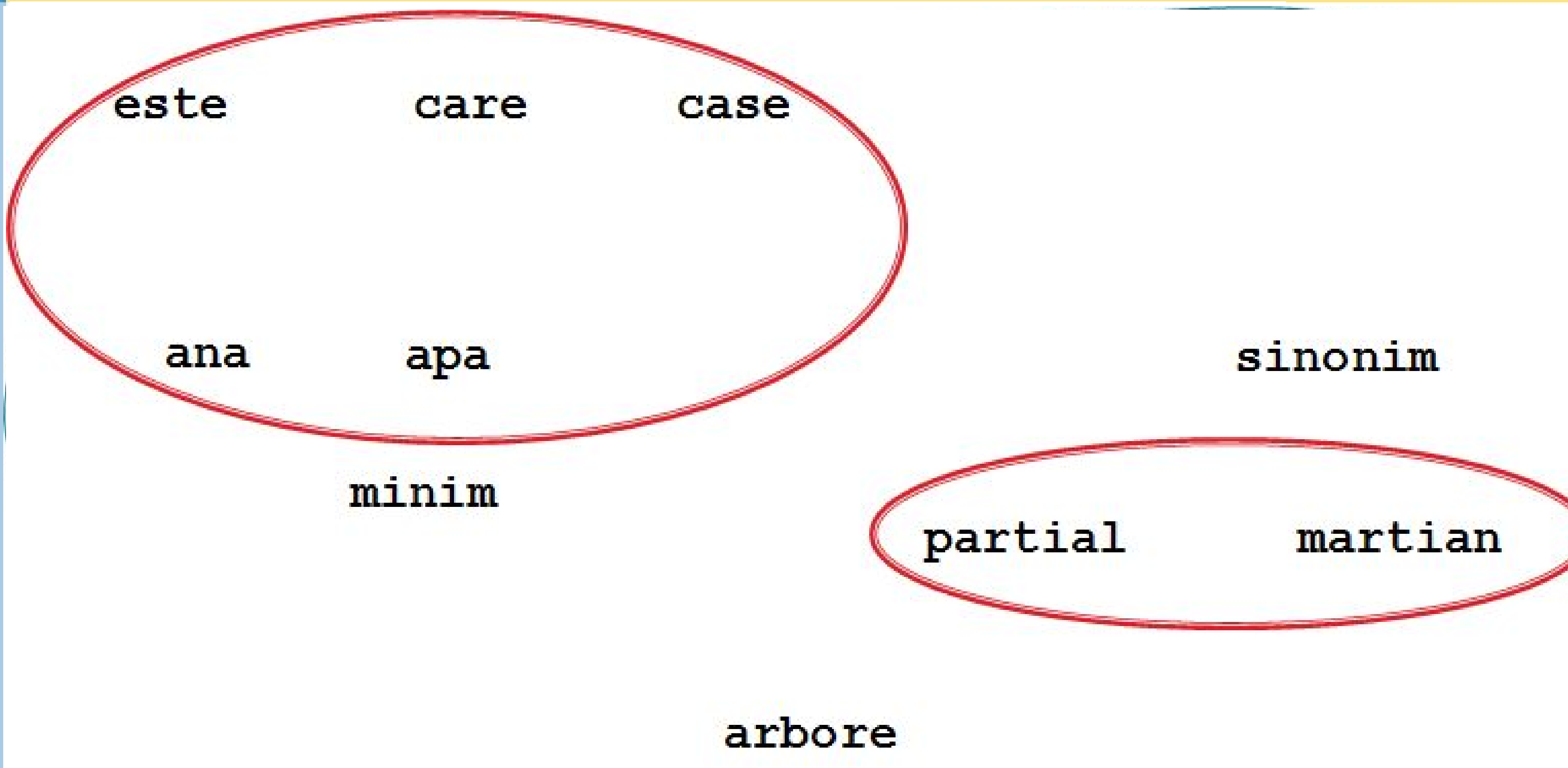
K = 3 clustere

Aplicație: k-Clustering



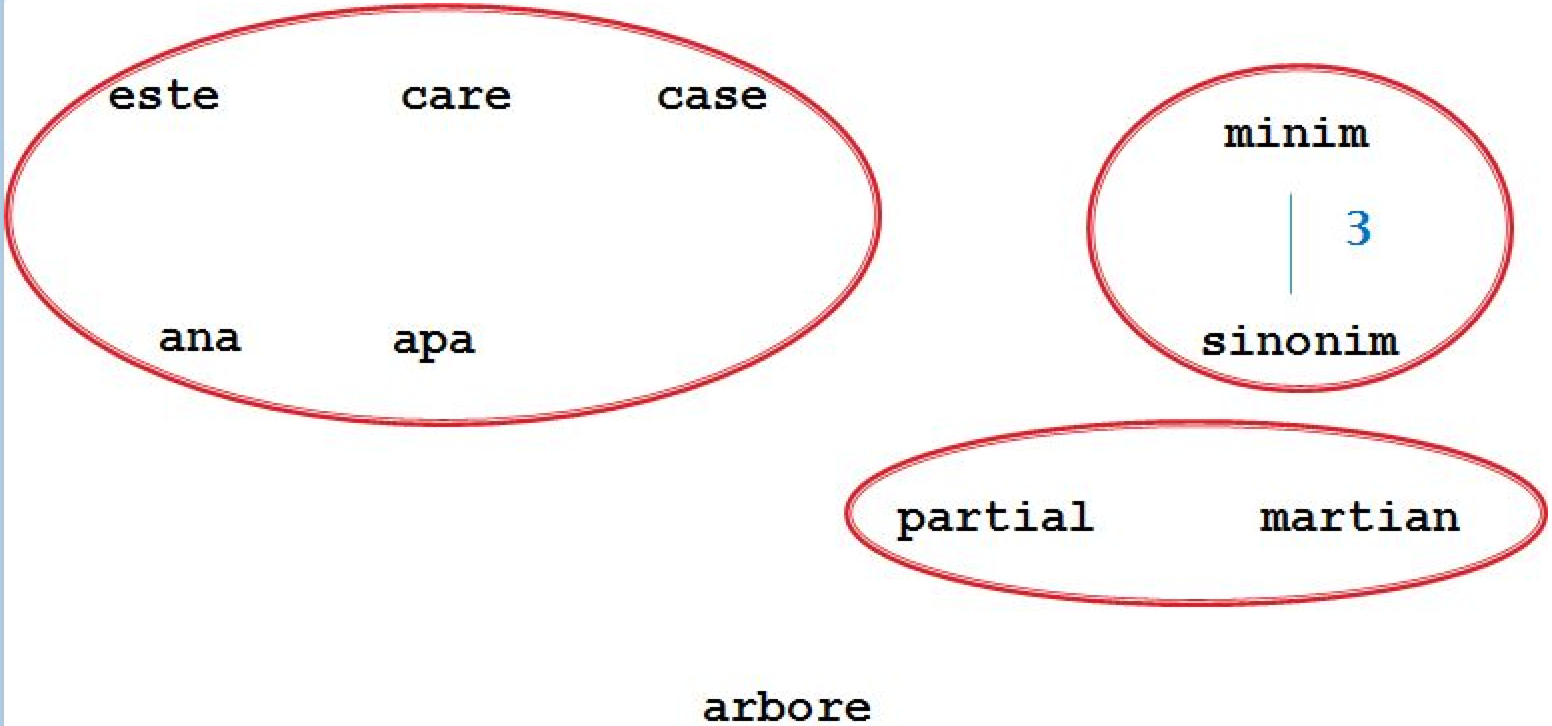
K = 3 clustere

Aplicație: k-Clustering



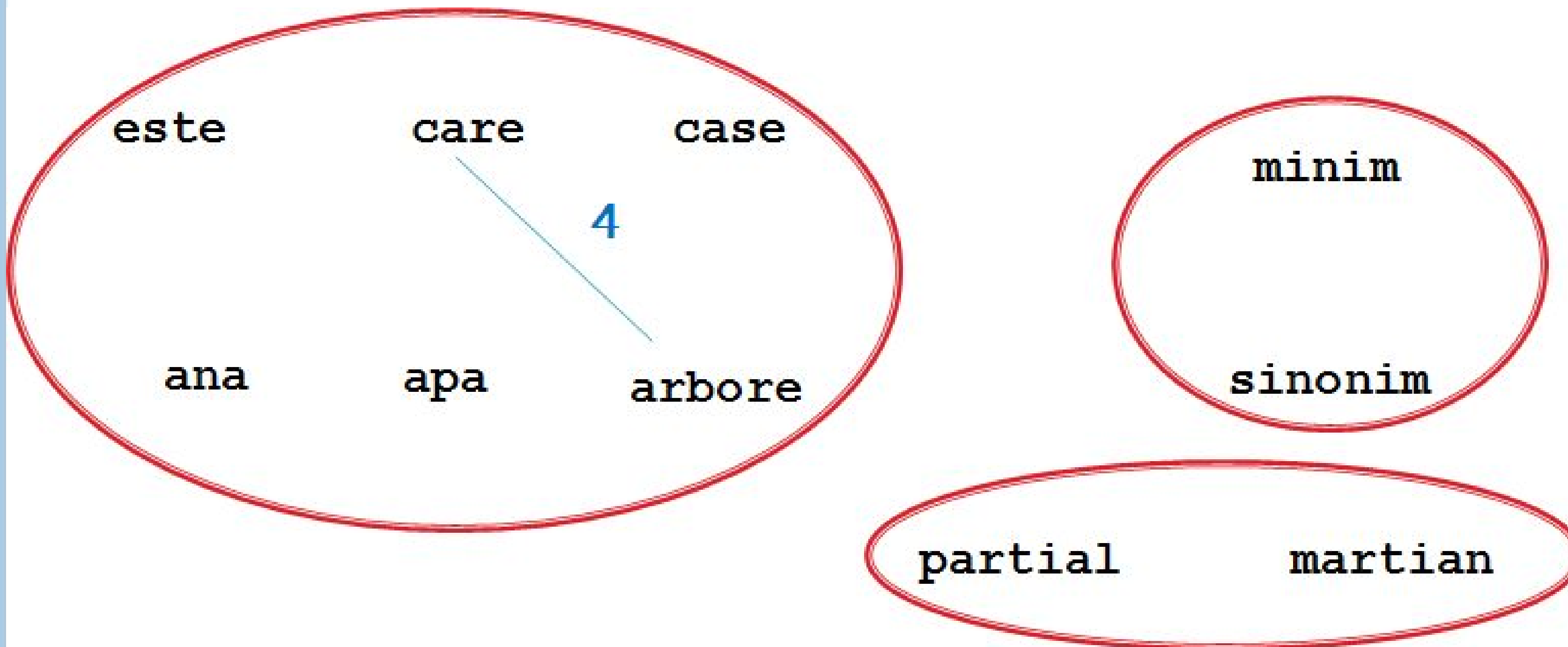
K = 3 clustere

Aplicație: k-Clustering



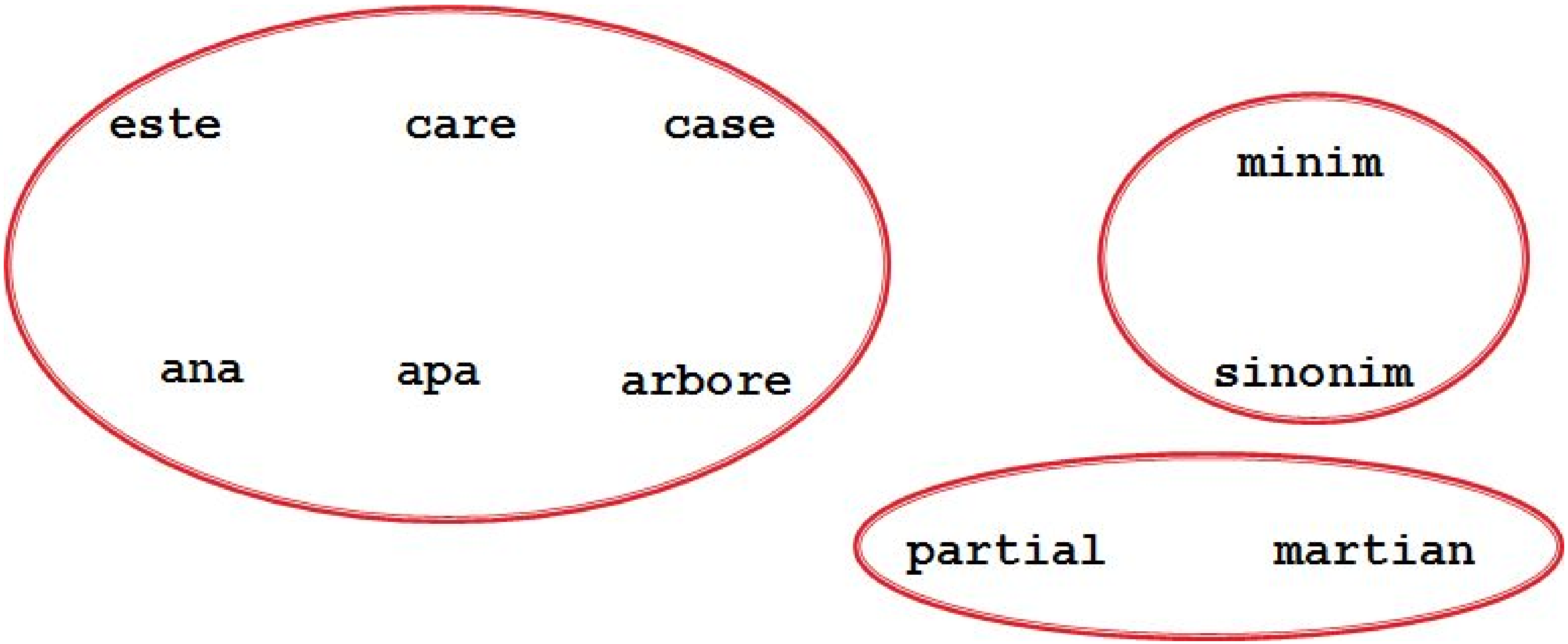
K = 3 clustere

Aplicație: k-Clustering



K = 3 clustere

Aplicație: k-Clustering



K = 3 clustere

Aplicație: k-Clustering