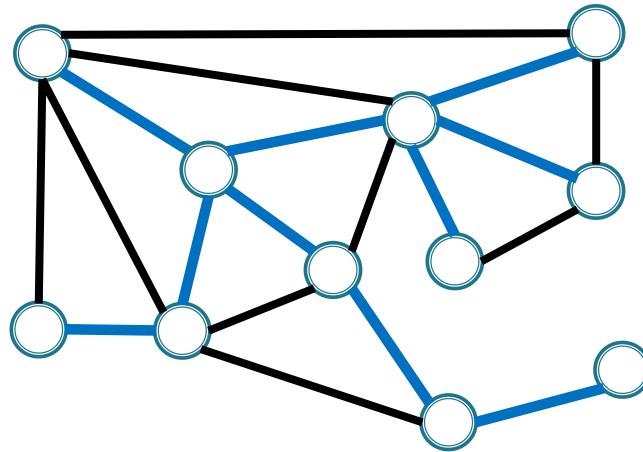


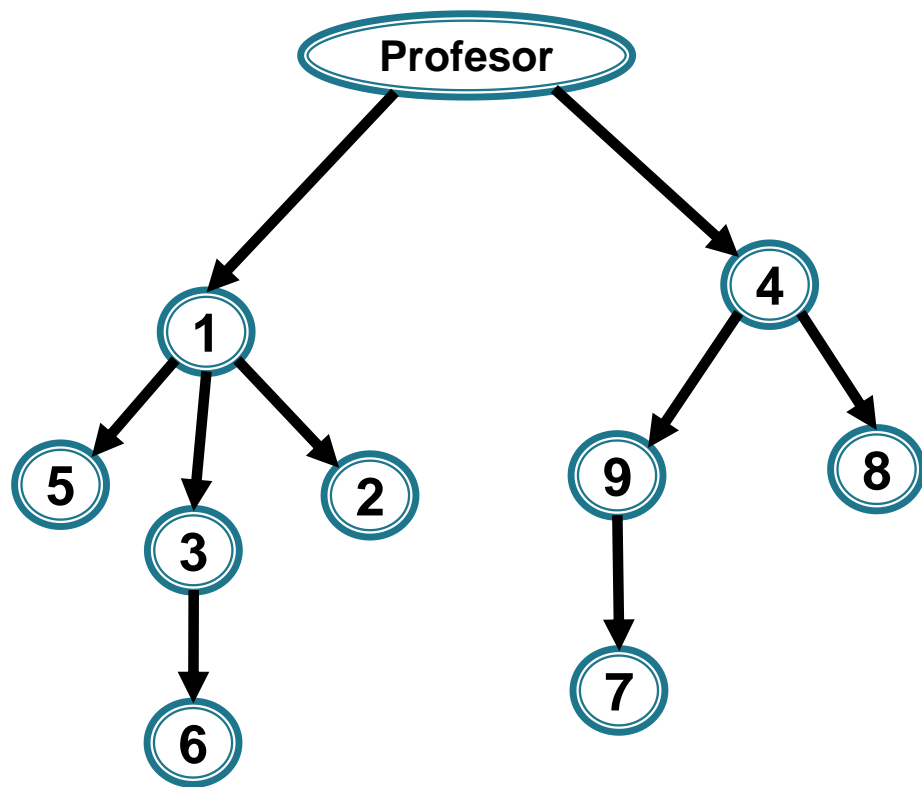
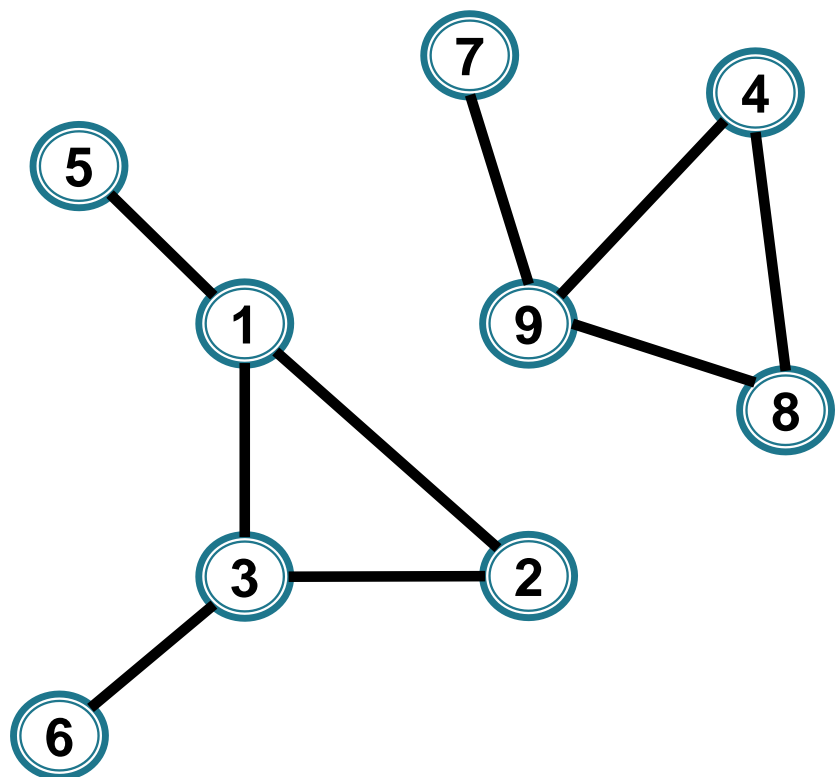
# Arbori parțiali de cost minim

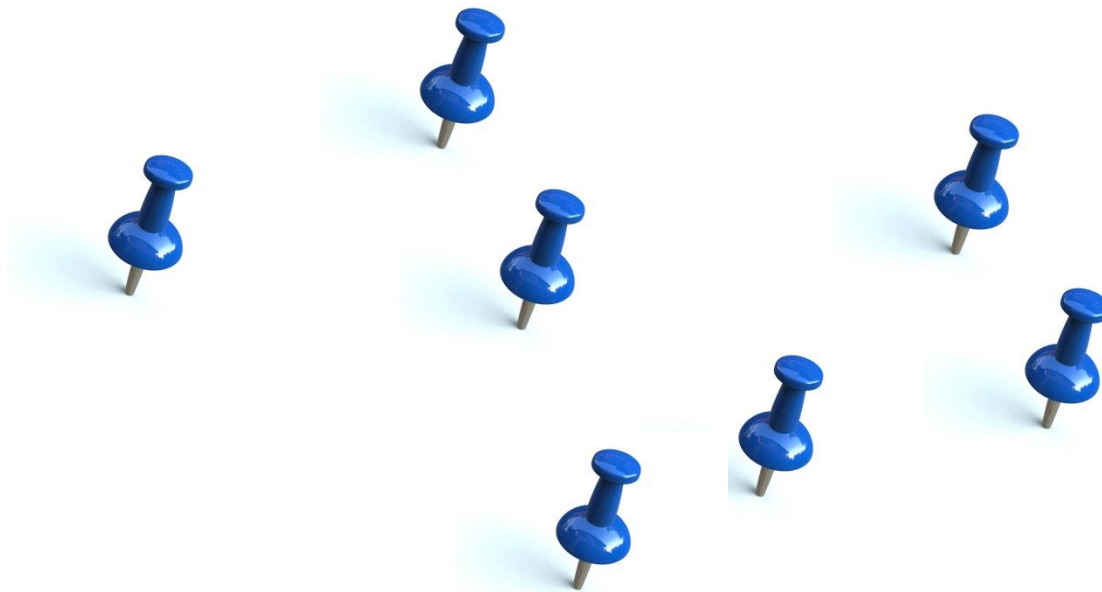


# Arbori parțiali

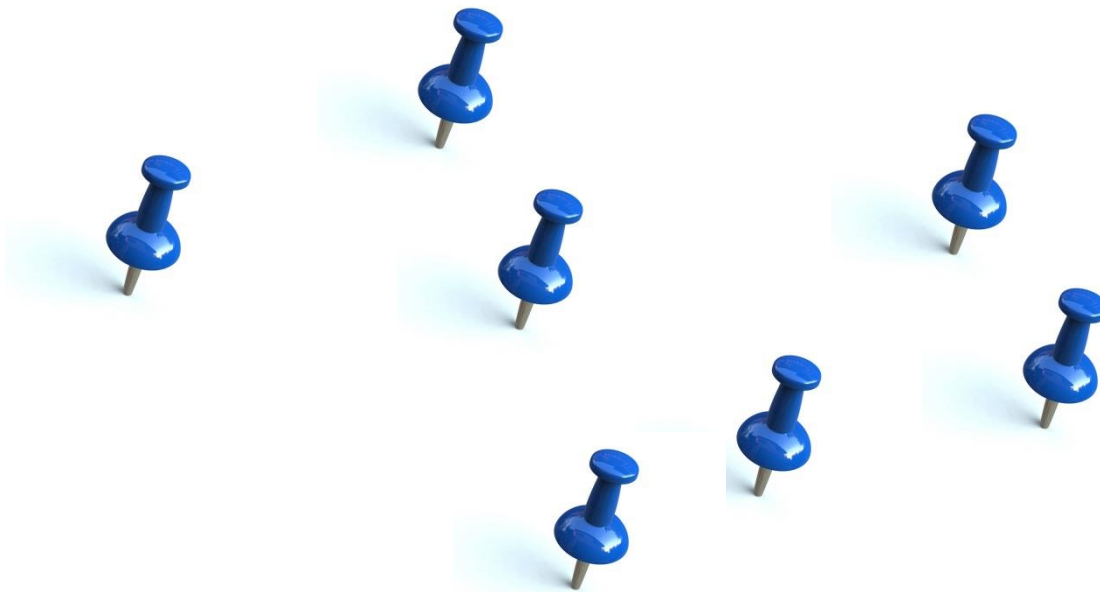


- “Scheletul” grafului
- Transmiterea de mesaje în rețea astfel încât mesajul să ajungă o singură dată în fiecare vârf
- Conectare fără redundanță + cu cost minim

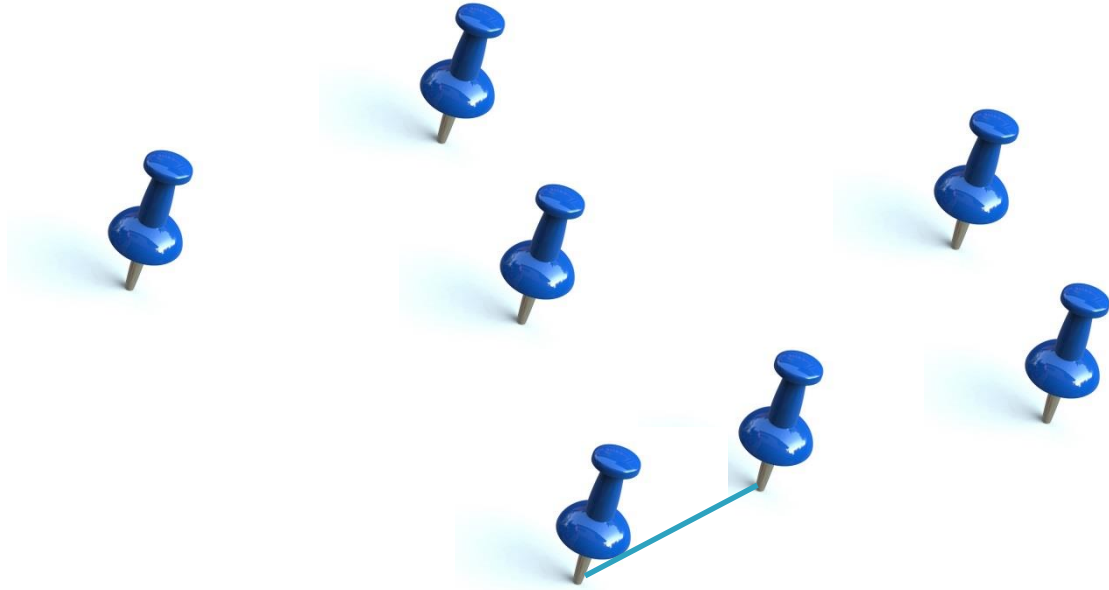


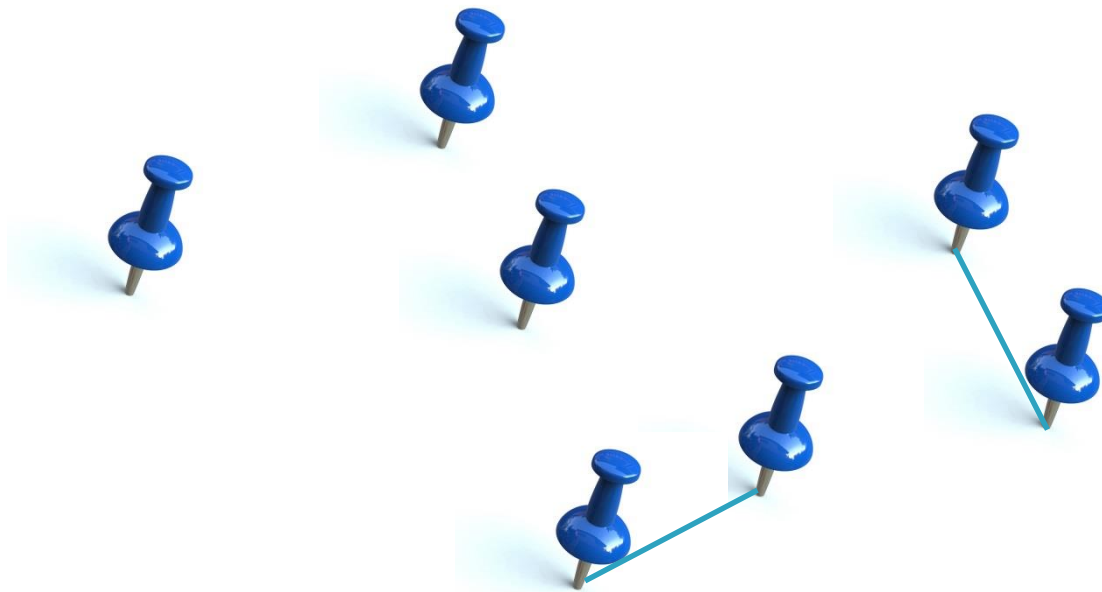


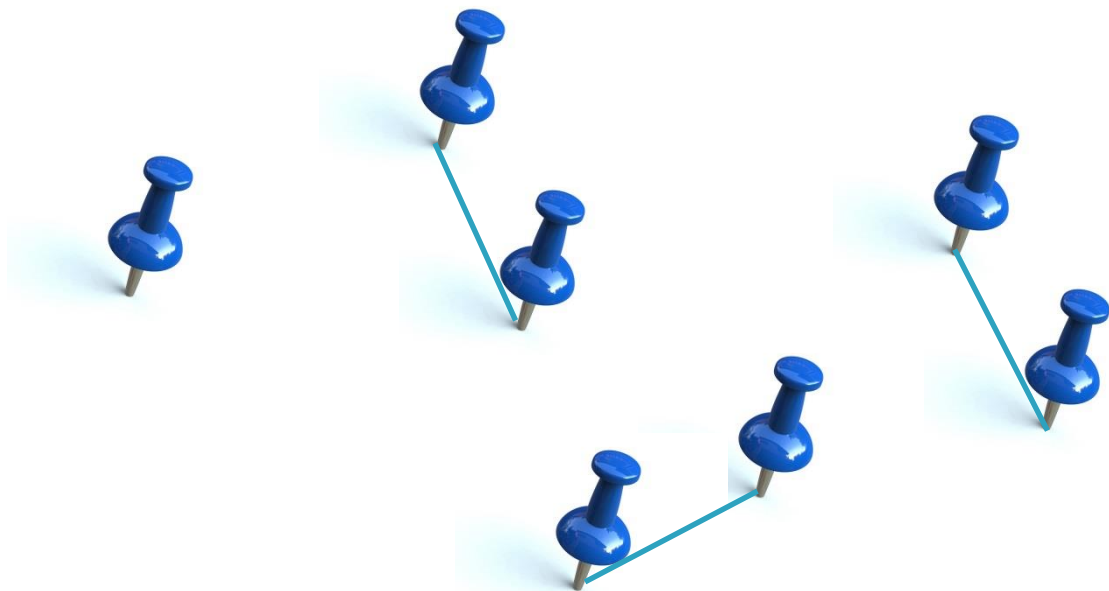
**Conectați pinii astfel încât să folosiți cât mai puțin cablu**



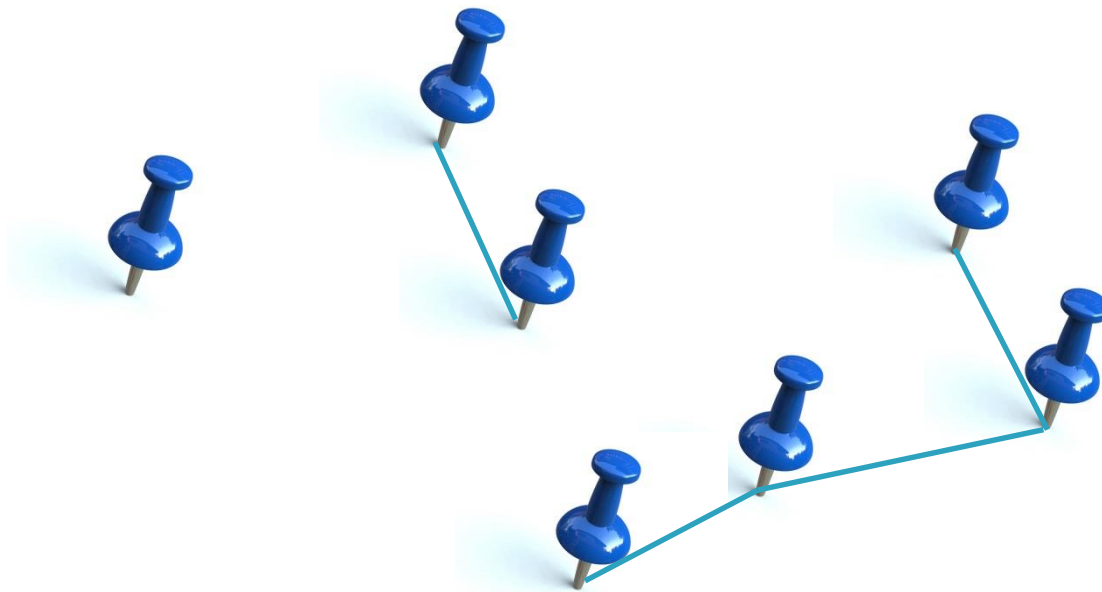
- Legăm pini apropiati
- Nu închidem cicluri

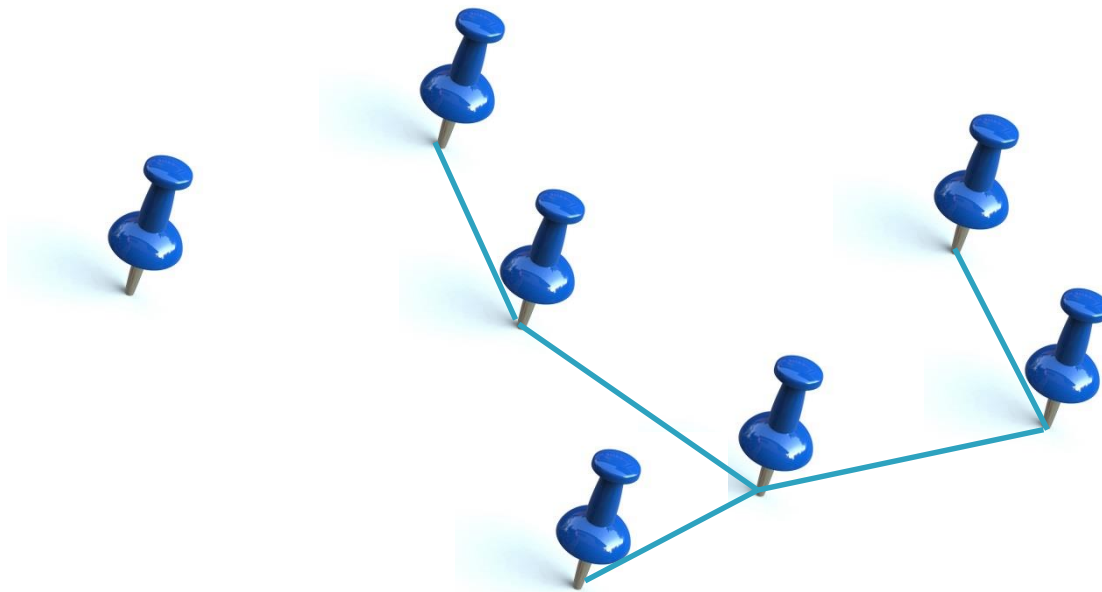


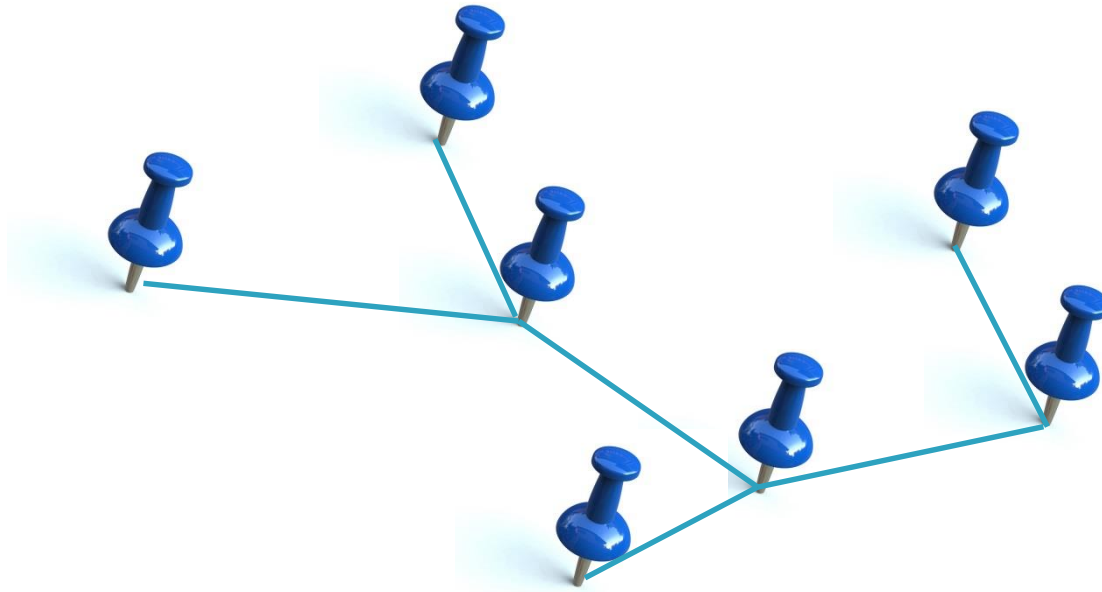














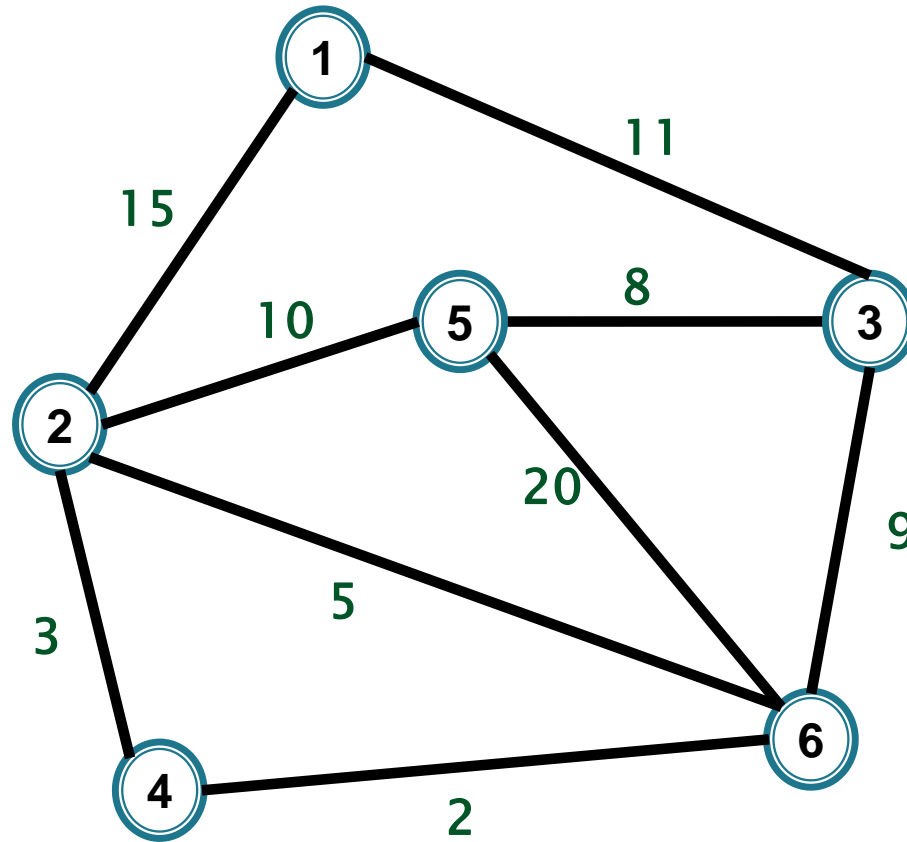
conectare cu cost minim  $\Rightarrow$  evităm ciclurile

Deci trebuie să construim

**graf conex + fără cicluri  $\Rightarrow$  arbore**

cu suma **costurilor muchiilor** minimă

# Grafuri ponderate



# Grafuri ponderate

- ▶  $G = (V, E)$  **ponderat** =
  - $w : E \rightarrow \mathbb{R}$  funcție **pondere** (**cost**)
- ▶ notat  $G = (V, E, w)$

# Grafuri ponderate

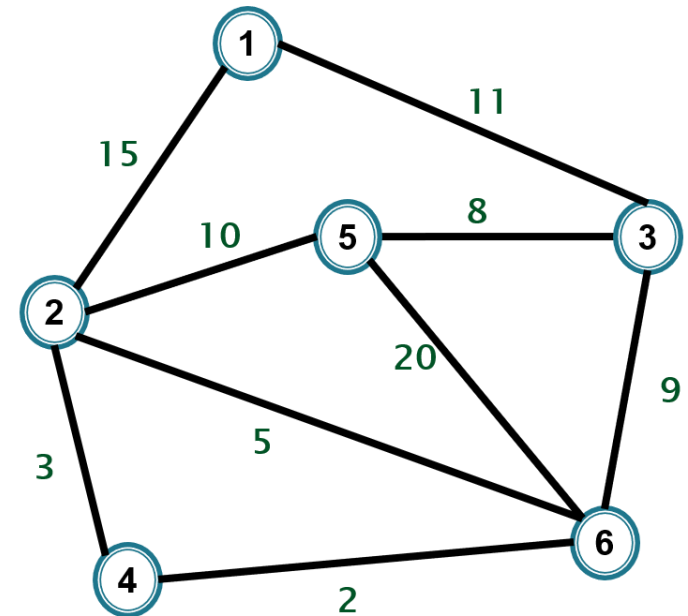
►  $G = (V, E, w)$  **graf ponderat**

► Pentru  $A \subseteq E$

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

► Pentru  $T$  subgraf al lui  $G$

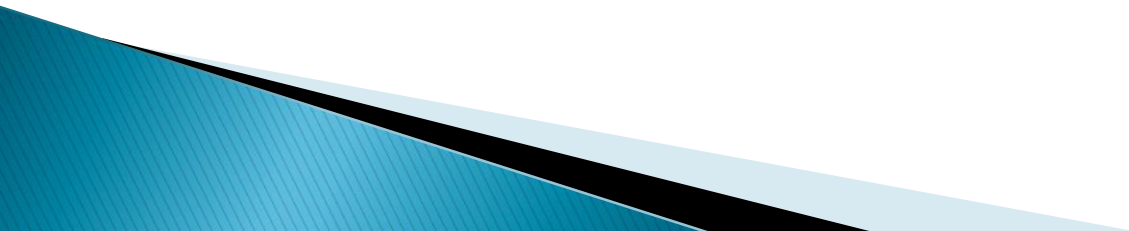
$$w(T) = \sum_{e \in E(T)} w(e)$$



$$w([1, 2, 5]) = w(1,2) + w(2,5) = 15 + 10 = 25$$

# Grafuri ponderate

## Reprezentarea grafurilor ponderate



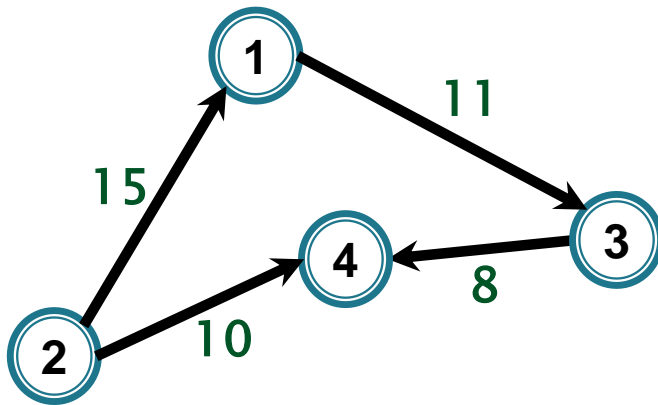


# Grafuri ponderate

## Reprezentarea grafurilor ponderate

- ▶ Matrice de costuri (ponderi)  $W = (w_{ij})_{i,j=1..n}$

$$w_{ij} = \begin{cases} 0, & \text{daca } i = j \\ w(i,j), & \text{daca } ij \in E \\ \infty, & \text{daca } ij \notin E \end{cases}$$

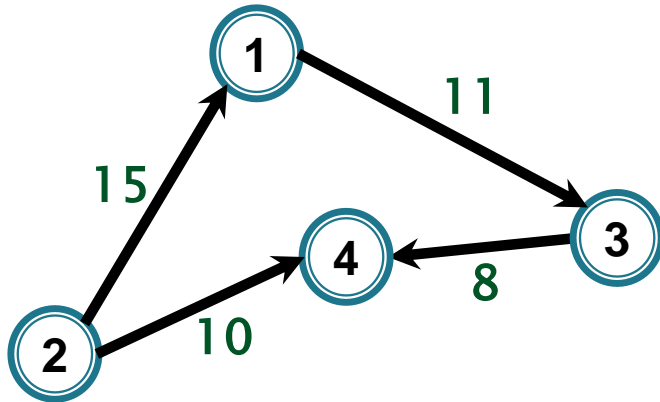


0	$\infty$	11	$\infty$
15	0	$\infty$	10
$\infty$	$\infty$	0	8
$\infty$	$\infty$	$\infty$	0

# Grafuri ponderate

## Reprezentarea grafurilor ponderate

- ▶ Liste de adiacență



1: 3 / 11

2: 1 / 15, 4 / 10

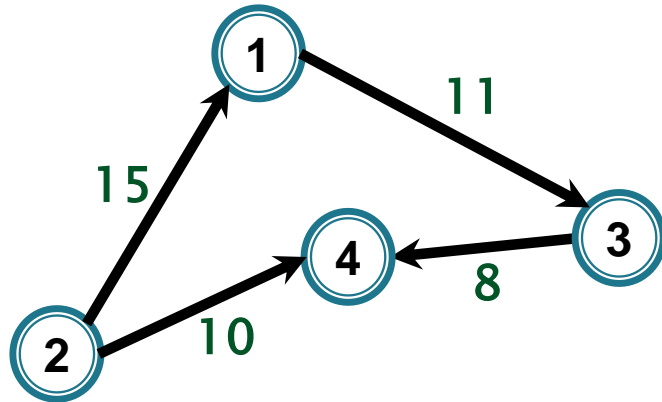
3: 4 / 8

4:

# Grafuri ponderate

## Reprezentarea grafurilor ponderate

- ▶ Liste de muchii/arce

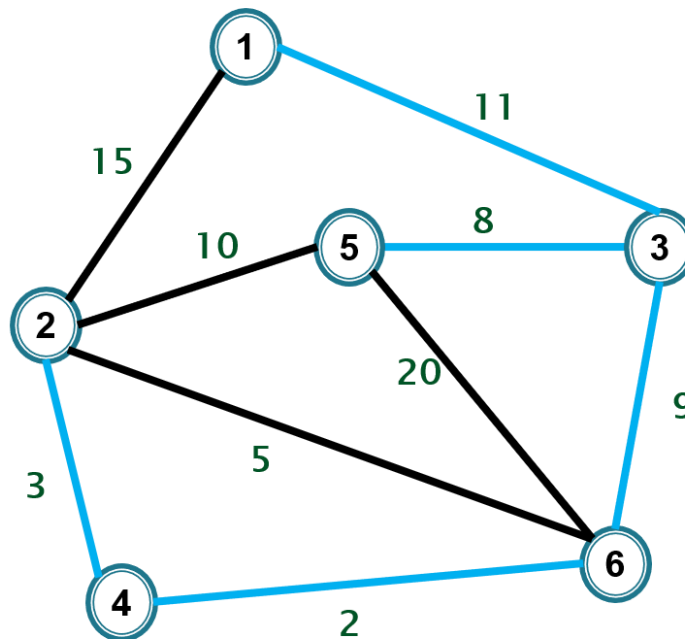


1	3	11
2	1	15
2	4	10
3	4	8

# A.p.c.m

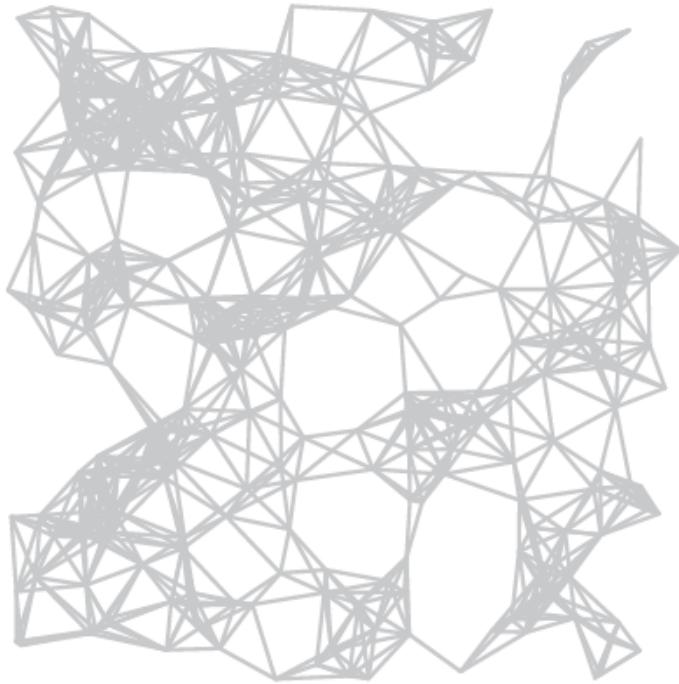
- ▶  $G = (V, E, w)$  **conex ponderat**
- ▶ **Arbore parțial de cost minim** al lui  $G$  = un arbore parțial  $T_{\min}$  al lui  $G$  cu

$$w(T_{\min}) = \min \{ w(T) \mid T \text{ arbore partial al lui } G \}$$

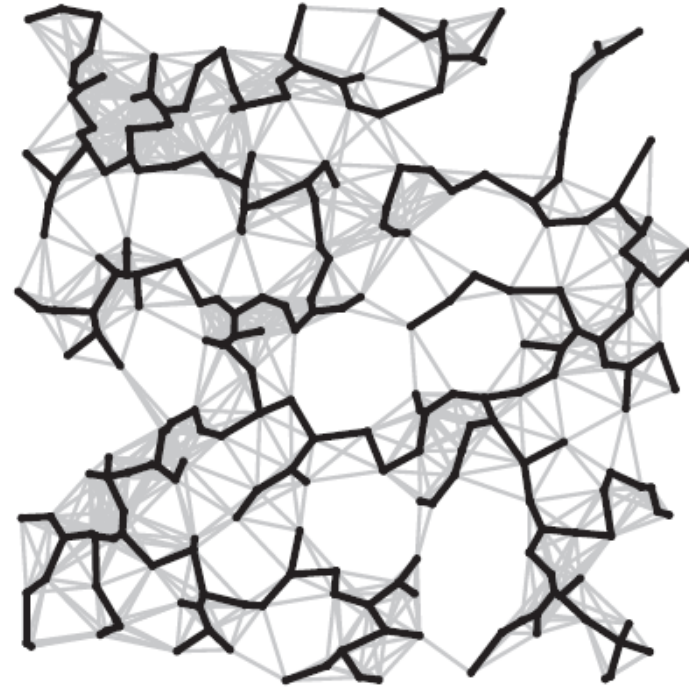


# A.p.c.m.

graf 250 noduri



apcm



**Imagine din**

R. Sedgewick, K. Wayne – **Algorithms**, 4th edition, Pearson Education, 2011

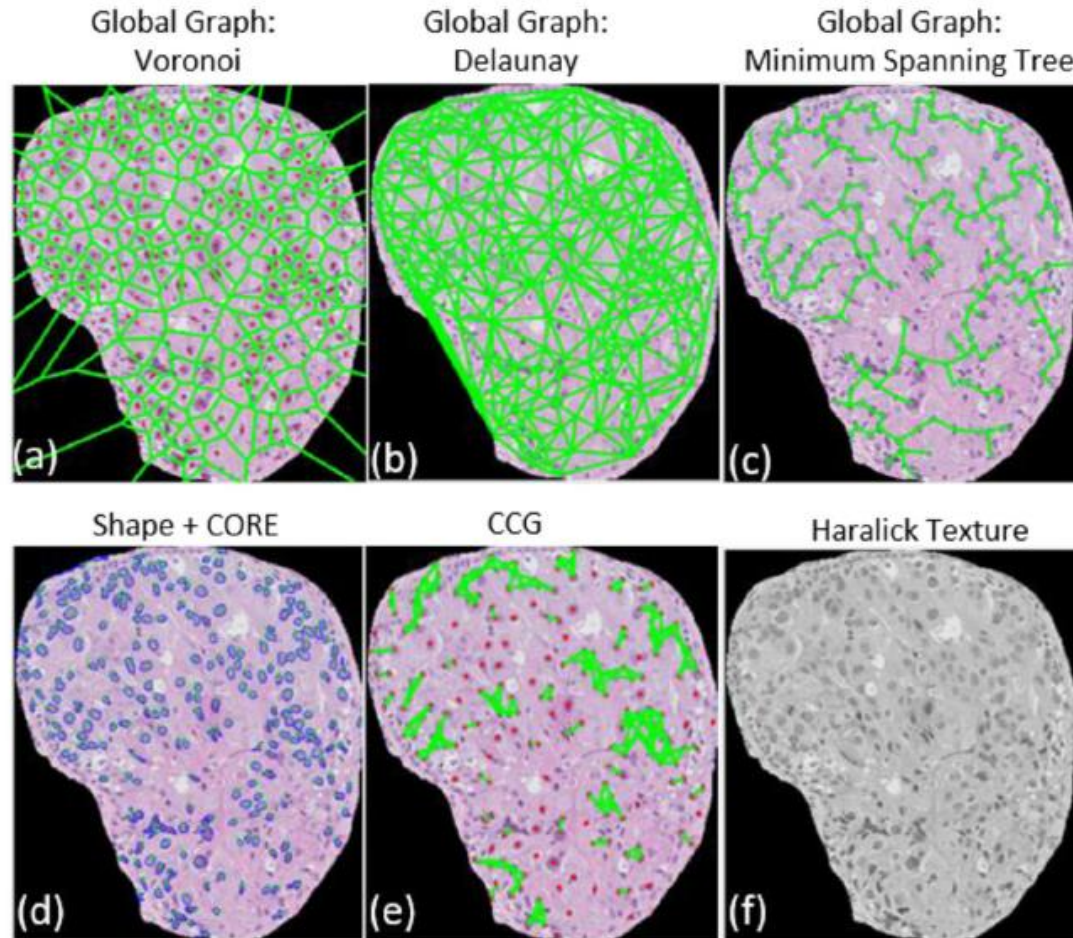
# Aplicații a.p.c.m.

- ▶ **Construcția/renovarea unui sistem de căi ferate a.î.:**
  - oricare două stații să fie conectate (prin căi renovate)
  - sistem economic (costul total minim)
- ▶ **Proiectarea de rețele, circuite electronice**
  - conectarea pinilor cu cost minim/ fără cicluri
- ▶ **Clustering**
- ▶ **Subrutină în alți algoritmi (trasee hamiltoniene)**
- ▶ **Proiectarea de rețele**
- ▶ **Protocoale de rutare cu evitarea ciclurilor, analiza imaginilor...**

<https://www.ics.uci.edu/~eppstein/gina/mst.html>



# Aplicații a.p.c.m.



**Fig. 4** Illustration of the feature maps corresponding to global graph (Voronoi (a), Delaunay (b), and Minimum Spanning Tree (c)), Shape (d), CORE (d), CCG (e), and Haralick Texture (f) features, capturing respectively spatial arrangement, shape, orientation, local arrangement, and heterogeneity of nuclei within a tissue image of a DCIS patient corresponding to the high ODx risk category the feature maps corresponding to the intermediate- and low-risk categories were included in S.3 as Figure 2 (II) and Figure 2 (III) respectively



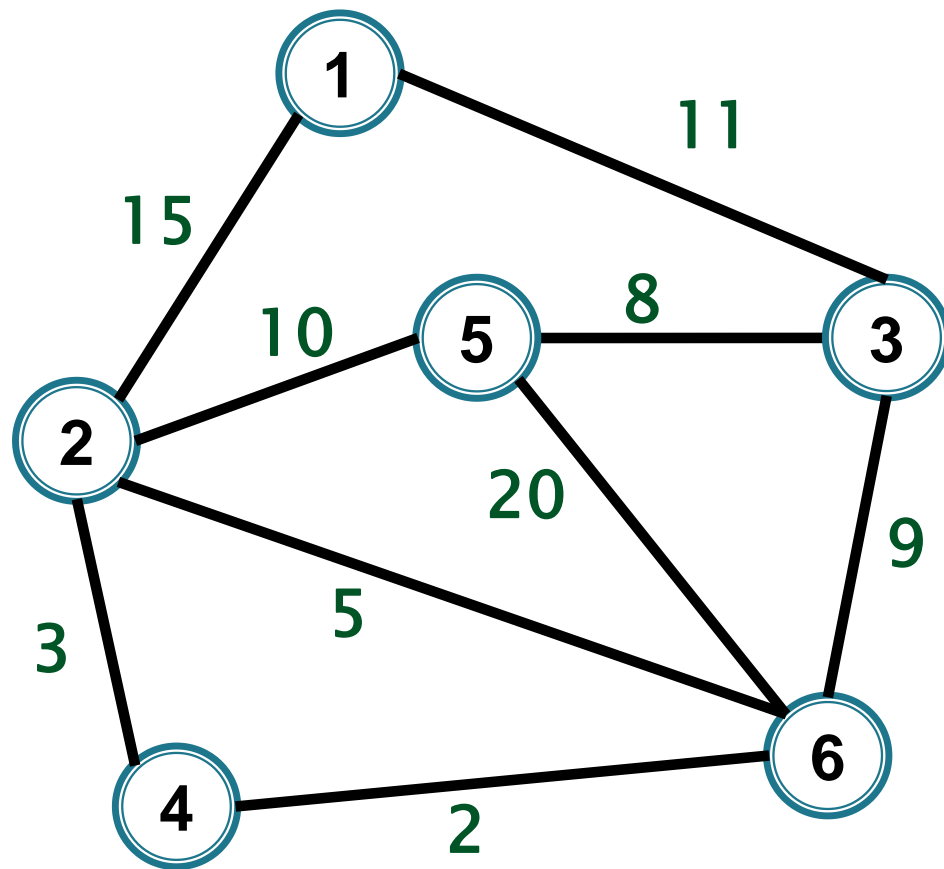
# Algoritmi de determinare a unui arbore parțial de cost minim



# Arbori parțiali de cost minim



Cum determinăm un arbore parțial de cost minim al unui graf conex ponderat?



# Arbori parțiali de cost minim



**Idee:** Prin adăugare succesivă de muchii, astfel încât mulțimea de muchii selectate

- ▶ să aibă costul cât mai mic
- ▶ să fie submulțime a mulțimii muchiilor unui arbore parțial de cost minim (apcm)

# Arbori parțiali de cost minim



După ce criteriu selectăm muchiile?

# Arbori parțiali de cost minim



După ce criteriu selectăm muchiile?

⇒ diverși algoritmi

# Arbori parțiali de cost minim

## Kruskal

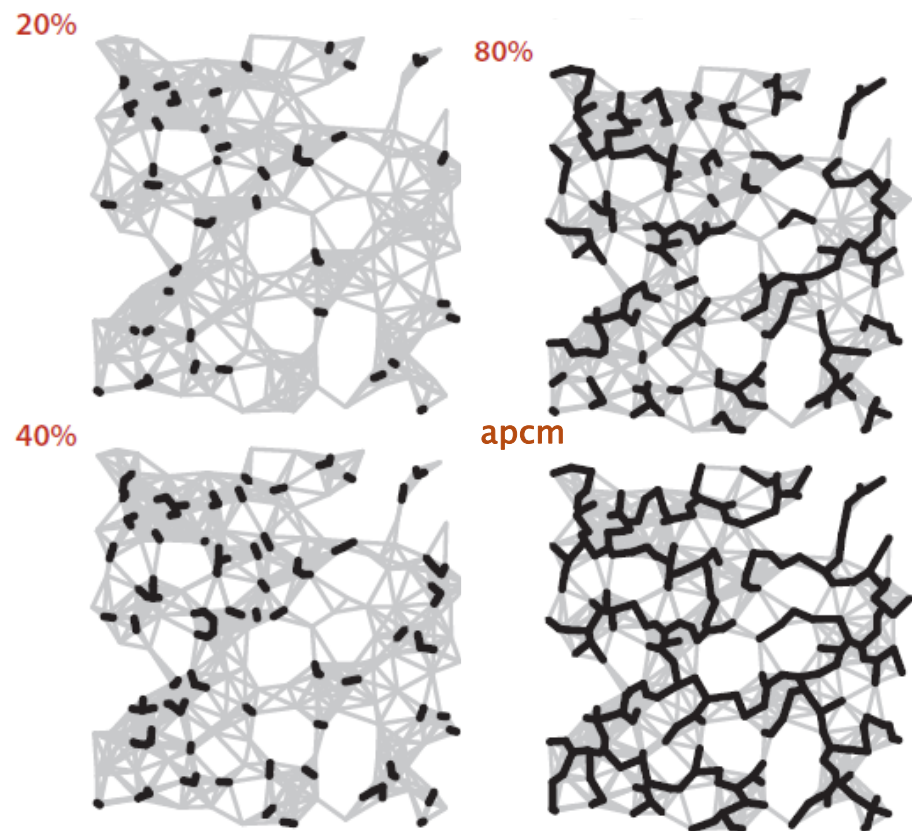
- Inițial  $T = (V; \emptyset)$
- pentru  $i = 1, n-1$ 
  - alege o muchie  $uv$  cu cost minim a.î.  $u, v$  sunt în componente conexe diferite ( $T+uv$  aciclic)
  - $E(T) = E(T) \cup uv$

## Prim

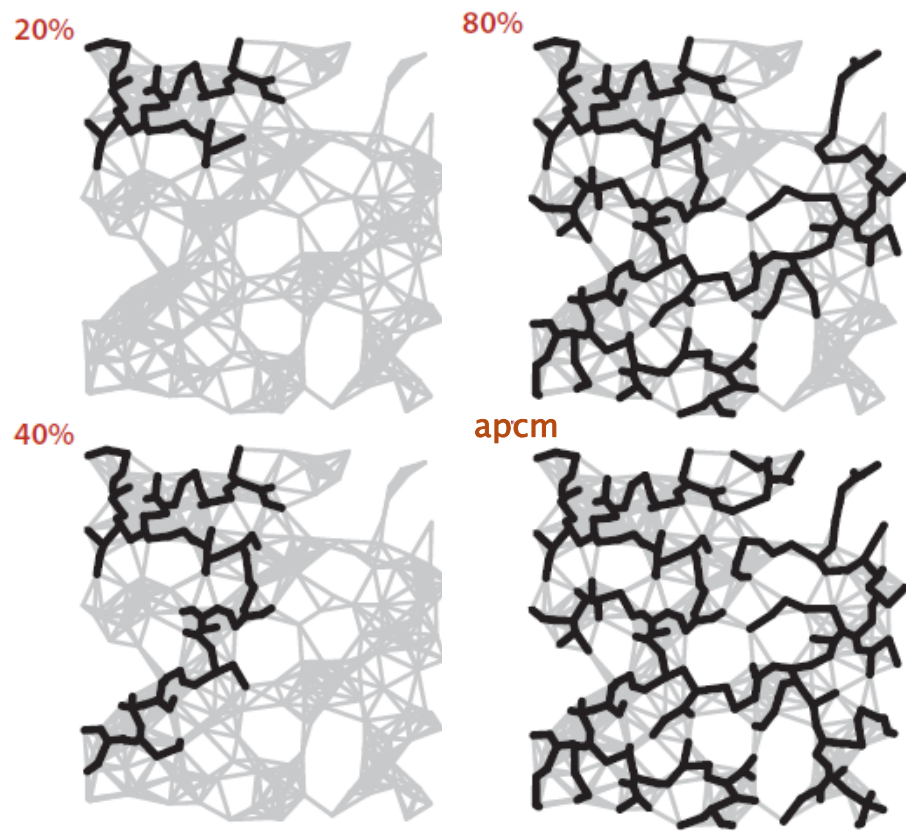
- $s$  – vârful de start
- Inițial  $T = (\{s\}; \emptyset)$
- pentru  $i = 1, n-1$ 
  - alege o muchie  $uv$  cu cost minim a.î.  $u \in V(T)$  și  $v \notin V(T)$
  - $V(T) = V(T) \cup \{v\}$
  - $E(T) = E(T) \cup uv$

# Arbori parțiali de cost minim

## Kruskal



## Prim



Imagine din

R. Sedgewick, K. Wayne – Algorithms, 4th edition, Pearson Education, 2011

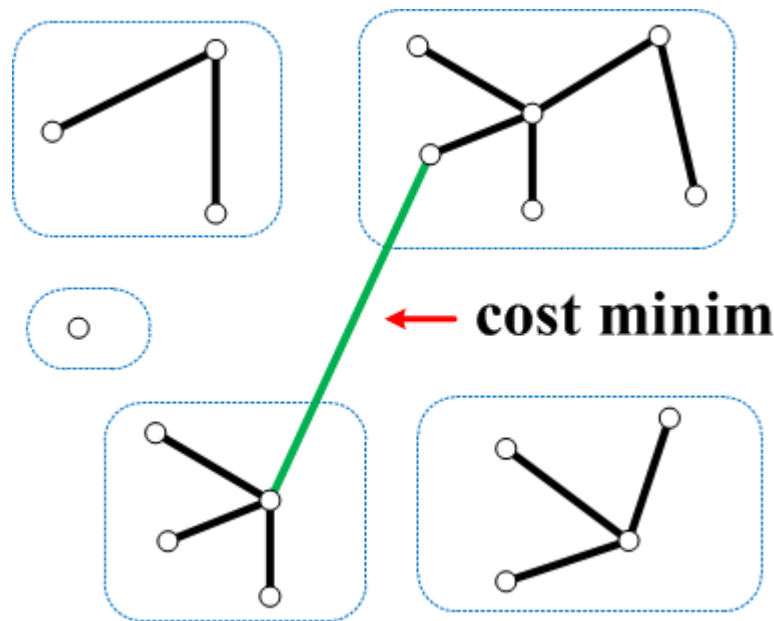
# Algoritmul lui Kruskal





# Algoritmul lui Kruskal

- ▶ La un pas este selectată o muchie de cost minim din  $G$  care nu formează cicluri cu muchiile deja selectate (care unește două componente conexe din graful deja construit)



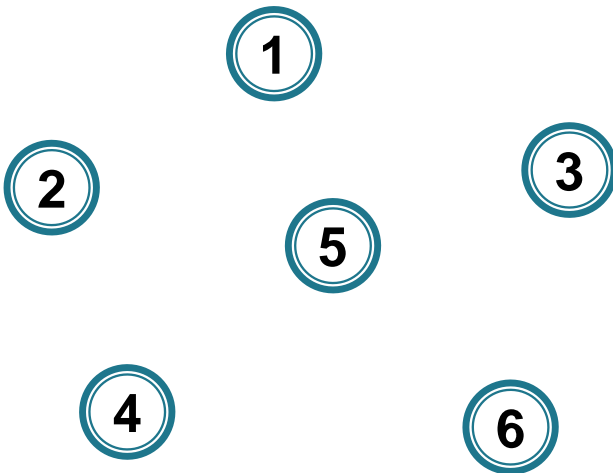
## ► O primă formă a algoritmului

### Kruskal

- Inițial  $T = (V; \emptyset)$
- pentru  $i = 1, n-1$ 
  - alege o muchie  $uv$  cu **cost minim din  $G$**  a.î.  $u, v$  sunt în **componente conexe diferite** ( $T+uv$  aciclic)
  - $E(T) = E(T) \cup \{uv\}$

# Kruskal

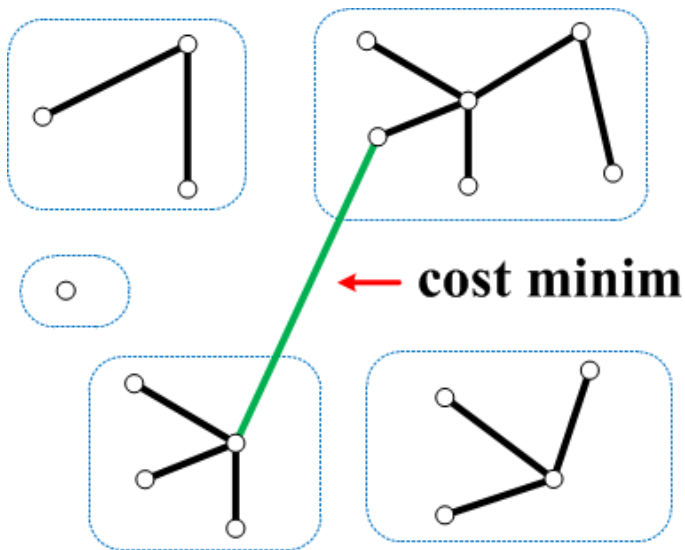
- Inițial: cele  $n$  vârfuri sunt **izolate**, fiecare formând o componentă conexă



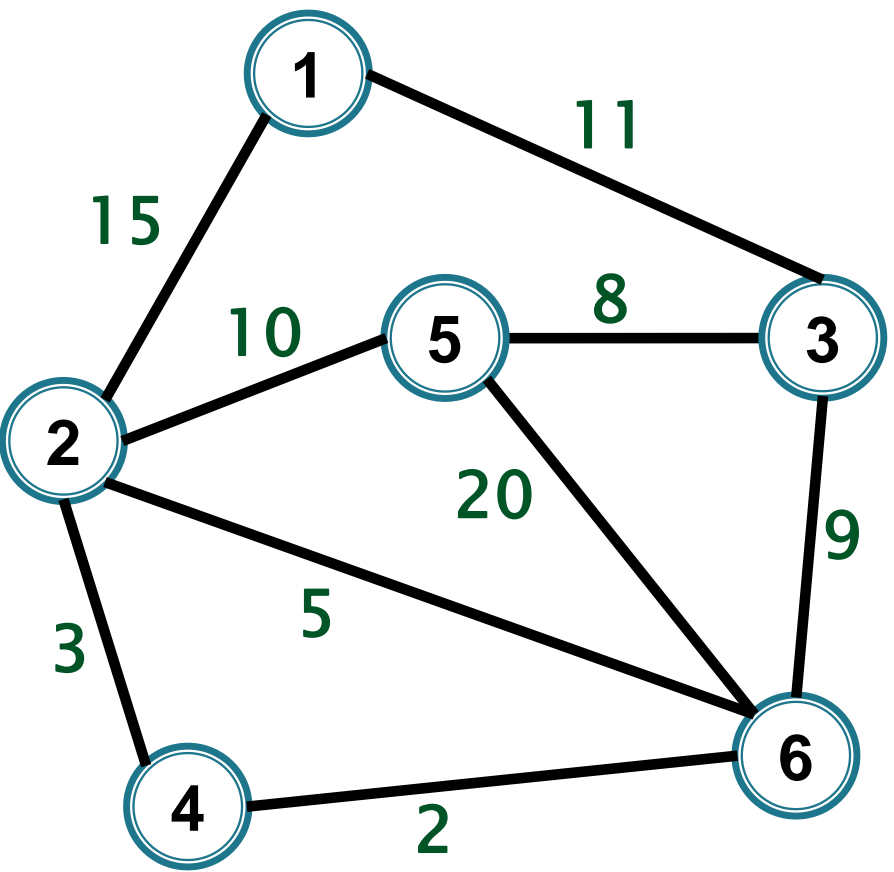
# Kruskal

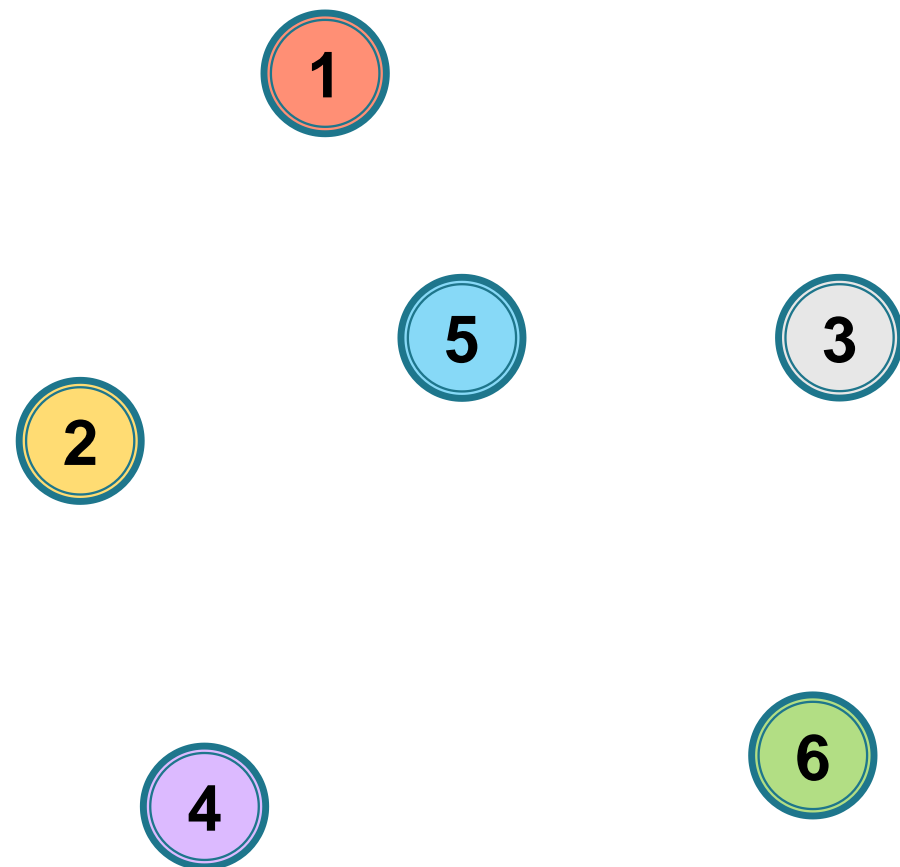
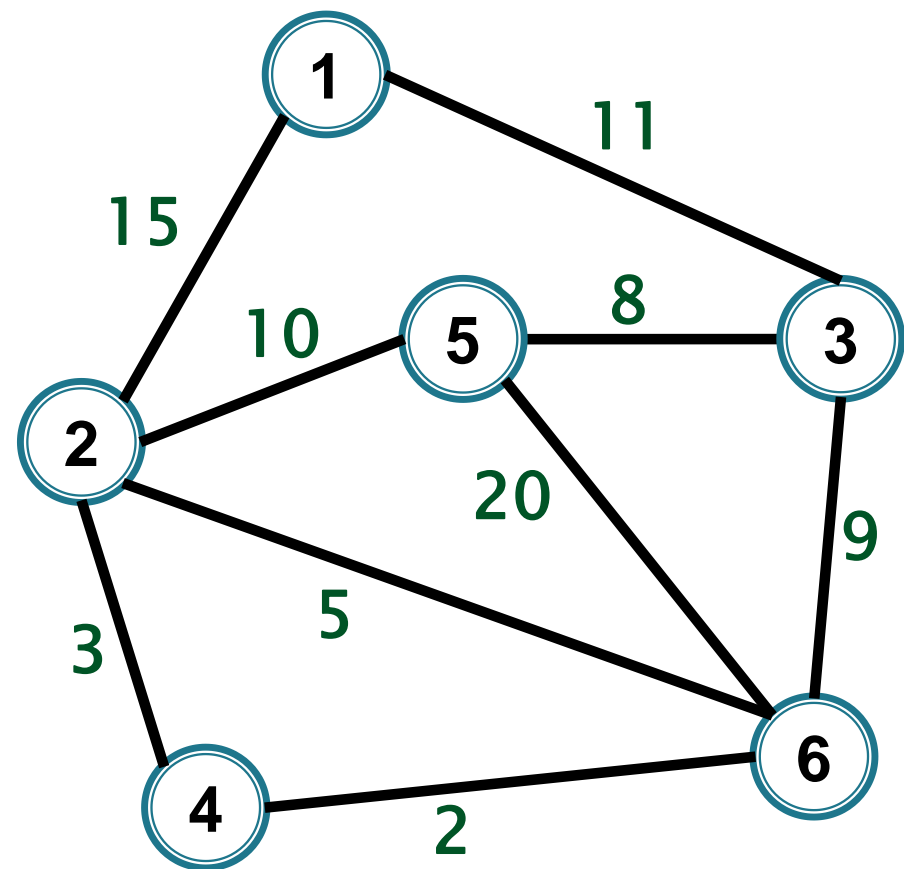
- La un pas:

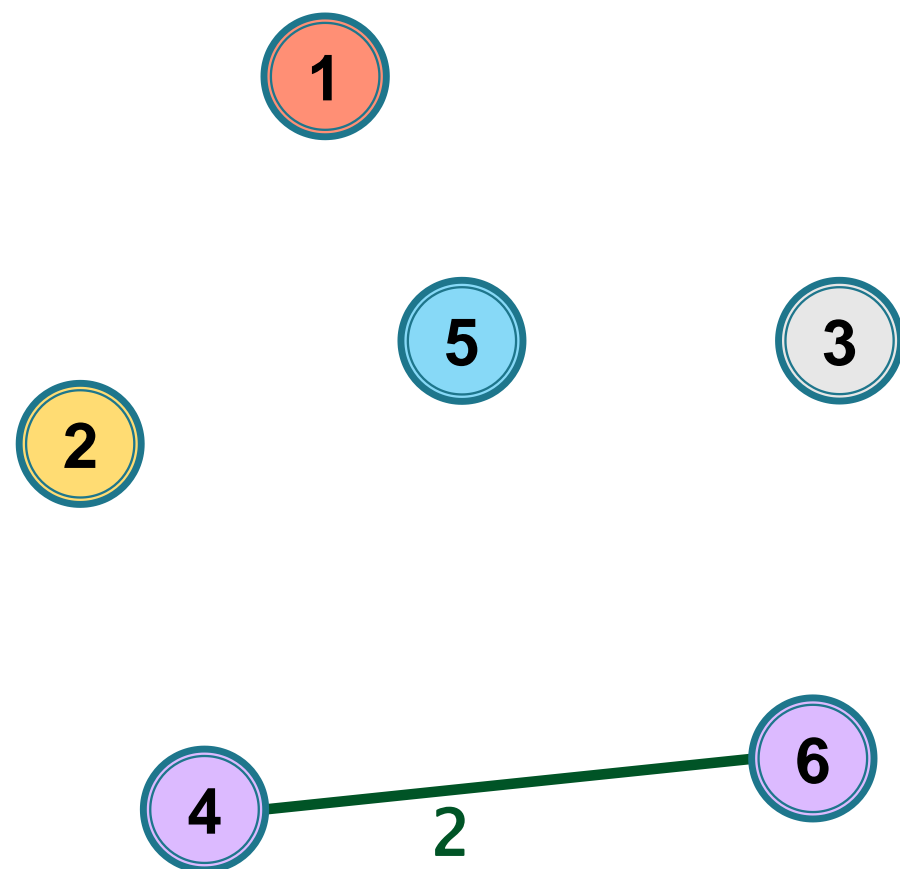
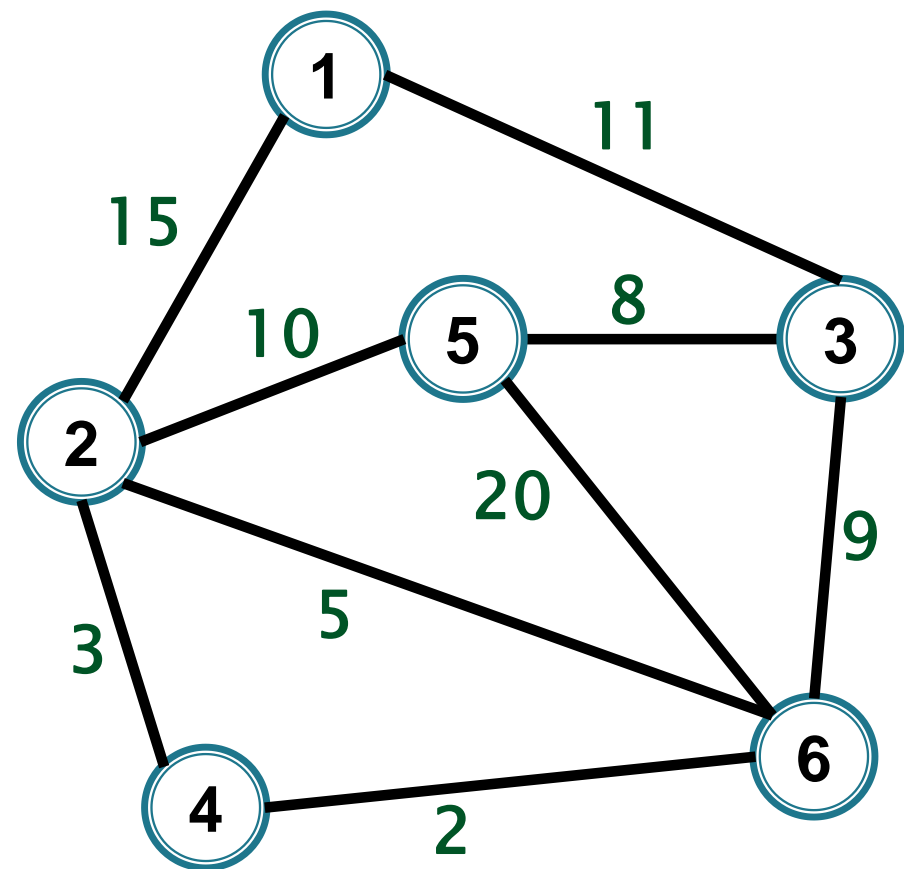
Muchiile selectate formează o pădure

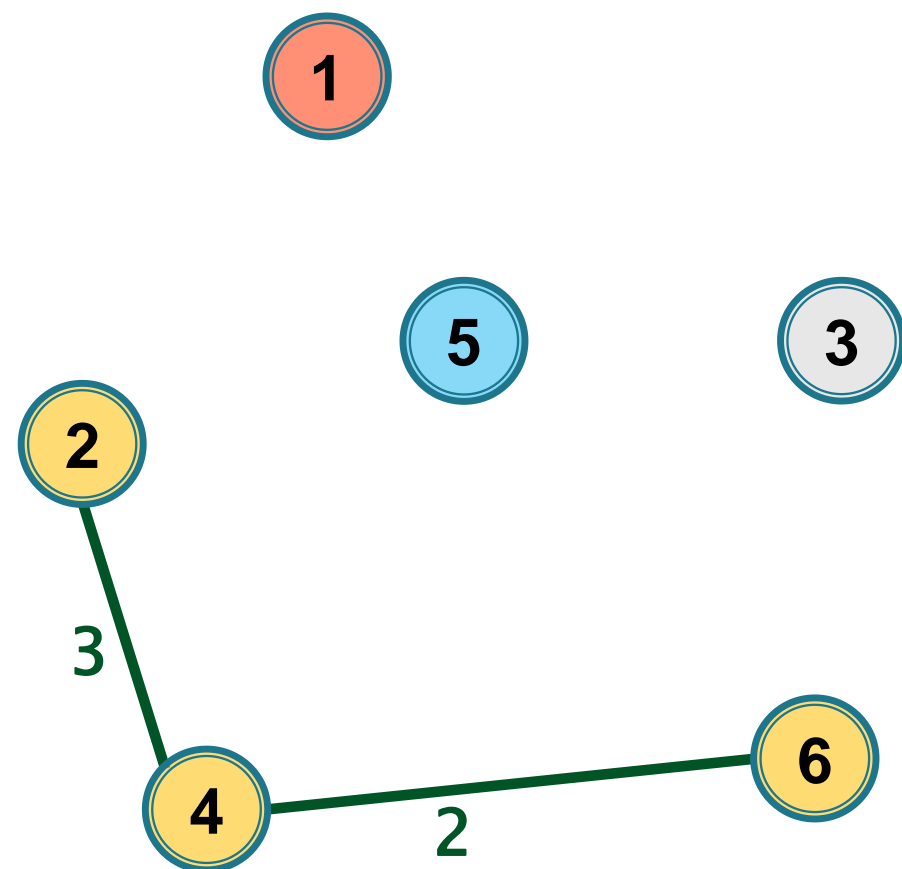
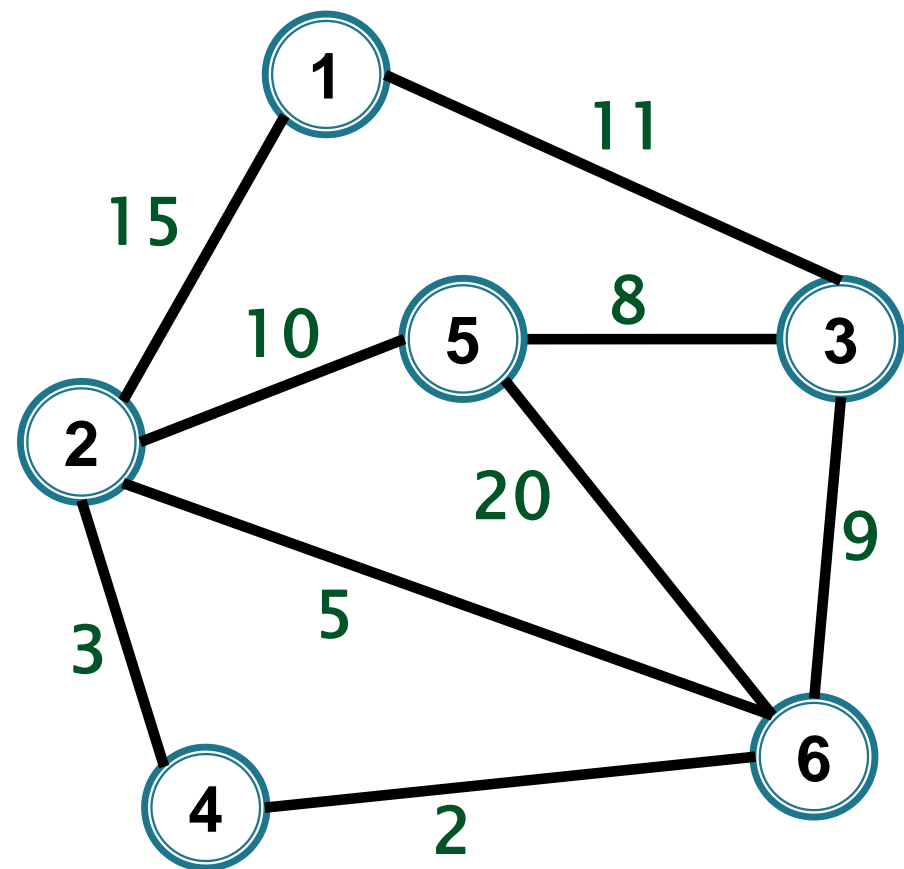


Este selectată o muchie de cost minim care unește doi arbori din pădurea curentă (două componente conexe)

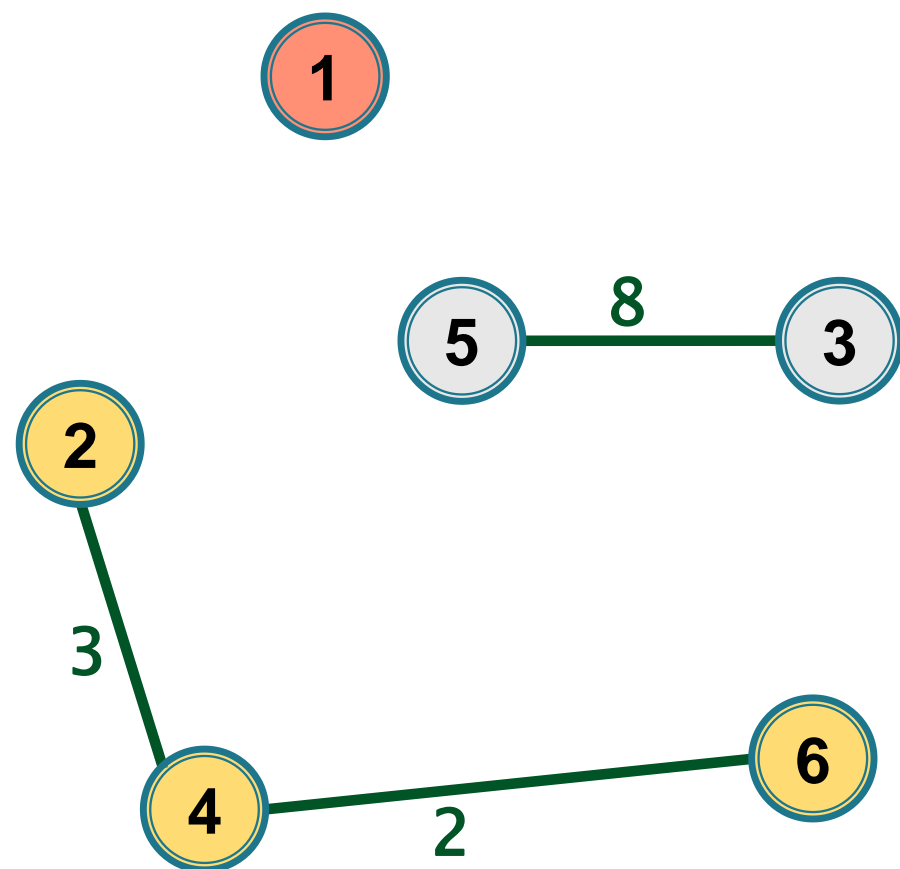
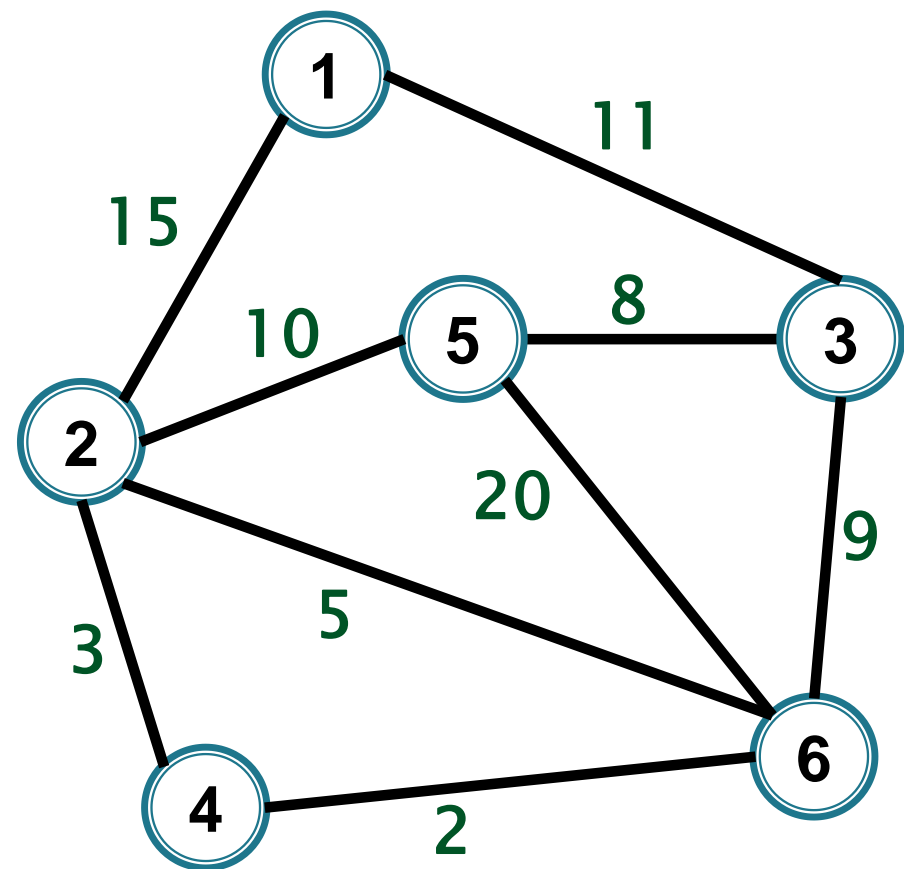


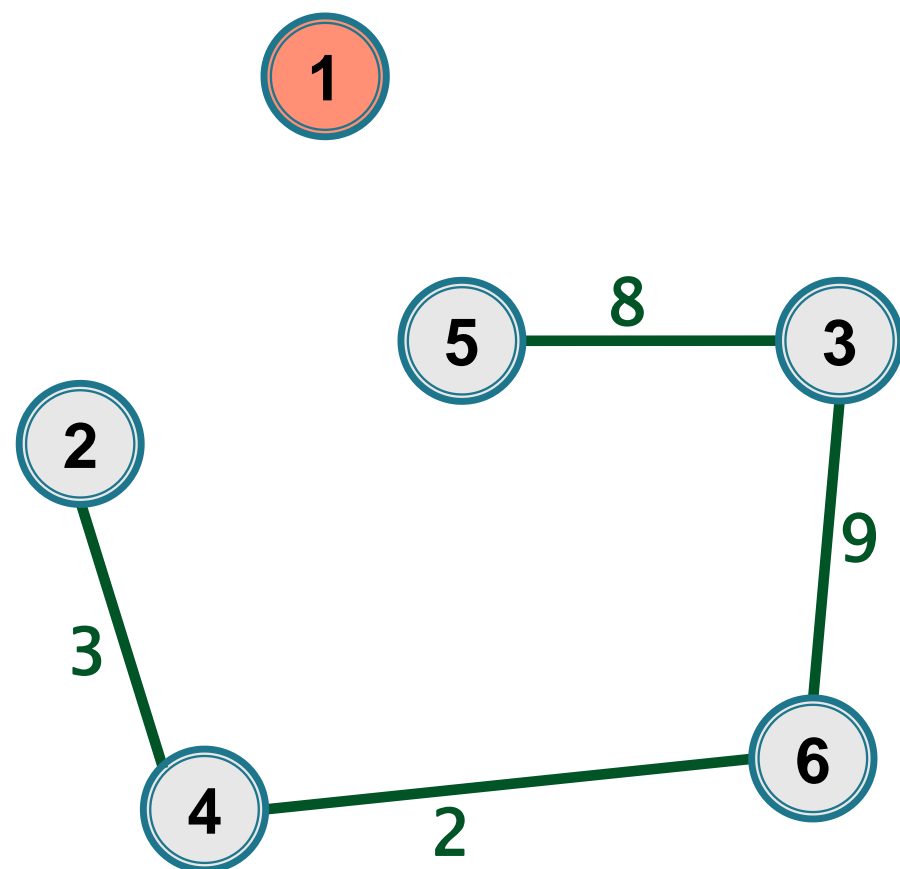
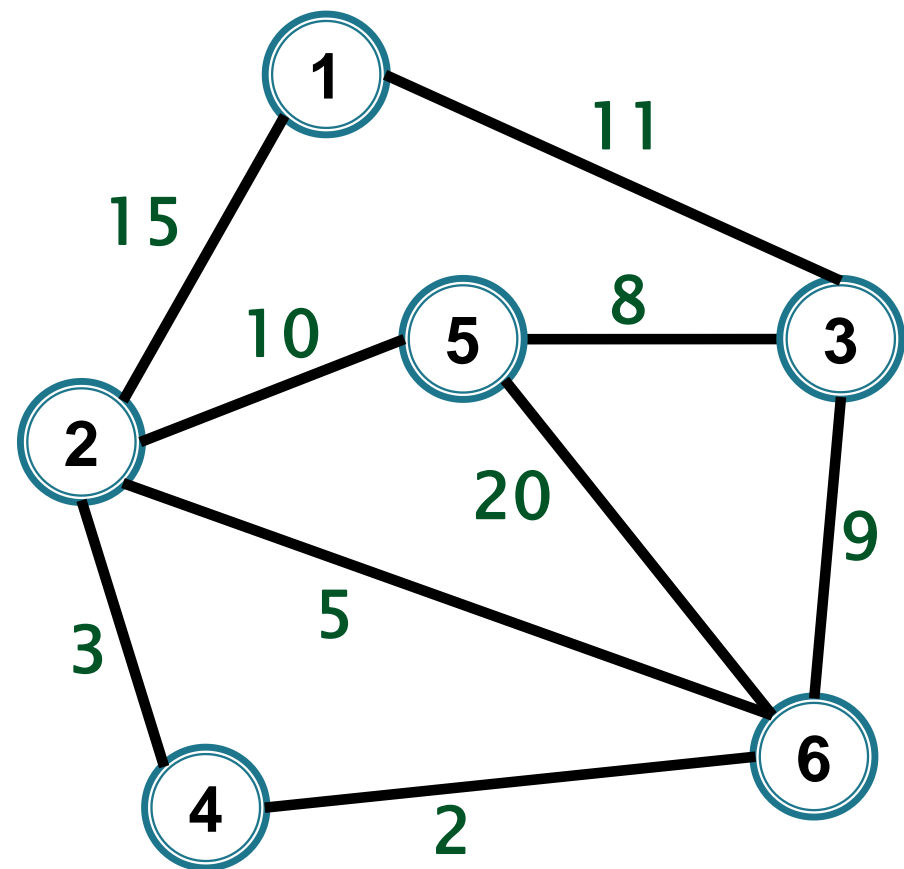


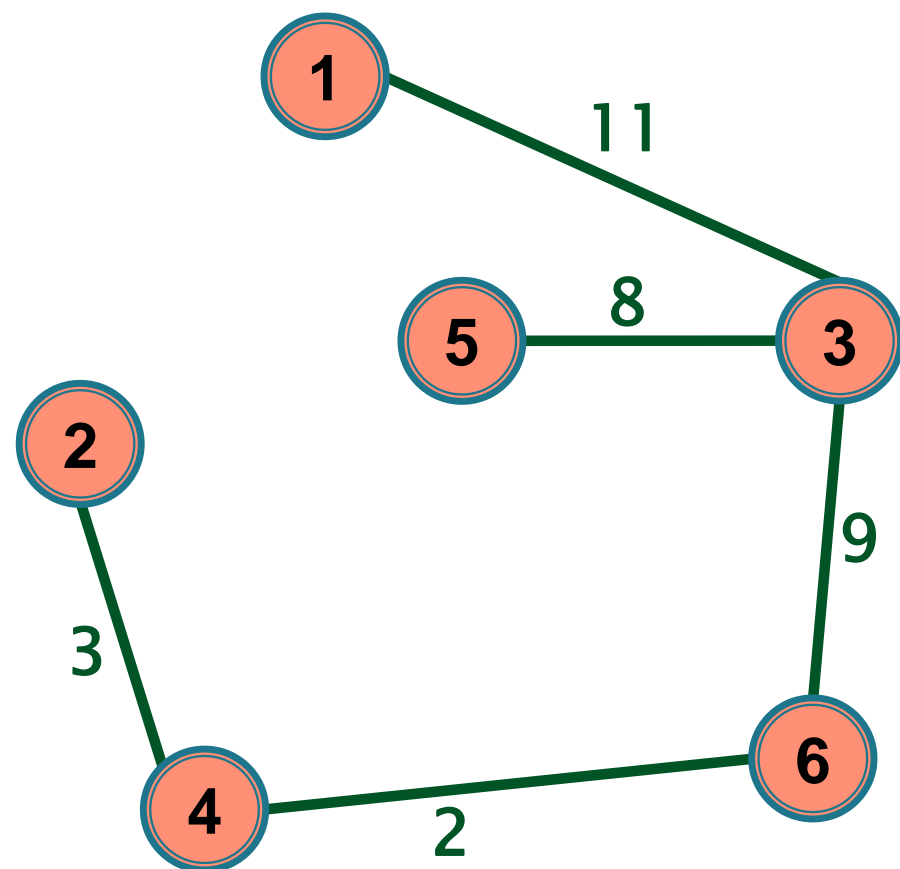
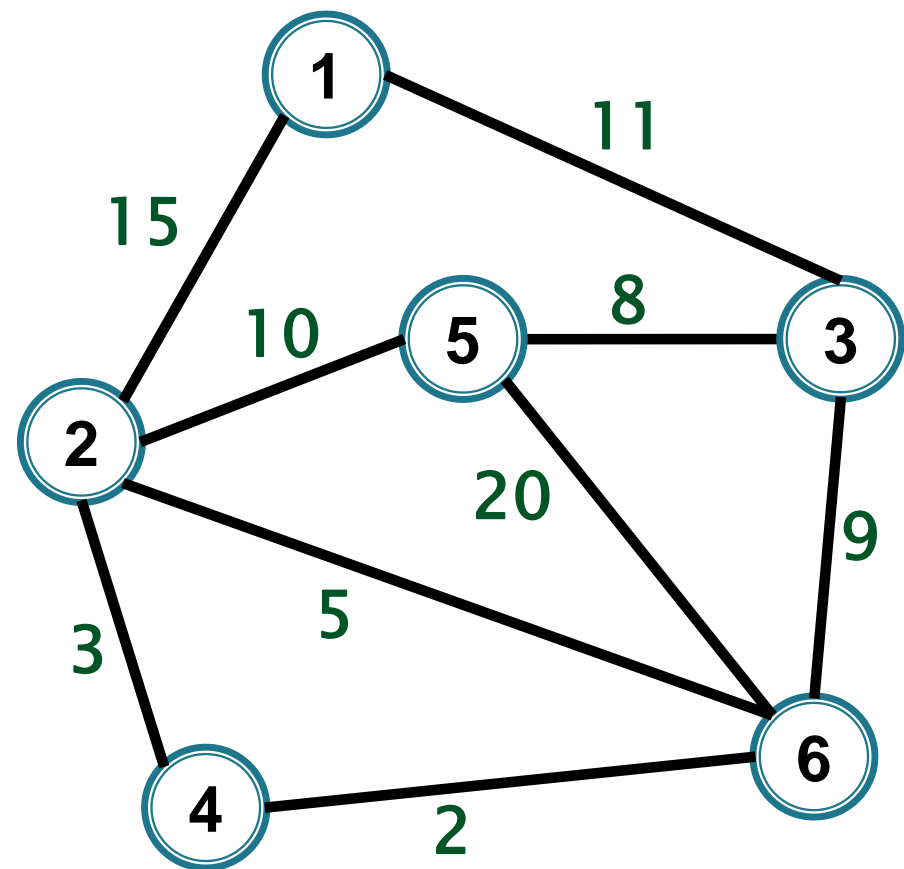












# Kruskal – Implementare



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)

# Kruskal



Pentru a selecta ușor o muchie de cost minim cu proprietatea dorită **ordonăm crescător muchiile după cost și considerăm muchiile în această ordine**

# Kruskal



## Reprezentarea grafului ponderat

- **Listă de muchii:** memorăm pentru fiecare muchie extremitățile și costul

# Kruskal



Cum testăm dacă muchia curentă unește două componente (  $\Leftrightarrow$  nu formează cicluri cu muchiile deja selectate) ?

# Kruskal



Cum testăm dacă muchia curentă unește două componente (  $\Leftrightarrow$  nu formează cicluri cu muchiile deja selectate) ?



Verificăm printr-o parcurgere dacă extremitățile muchiei sunt deja unite printr-un lanț



# Kruskal



Cum testăm dacă muchia curentă unește două componente (  $\Leftrightarrow$  nu formează cicluri cu muchiile deja selectate) ?



Verificăm printr-o parcurgere dacă extremitățile muchiei sunt deja unite printr-un lanț

$\Rightarrow O(mn)$  – ineficient



# Kruskal



La un pas componentele conexe din  $T$  sunt mulțimi disjuncte din  $V$  (partiție a lui  $V$ )

⇒ **structuri pentru mulțimi disjuncte**

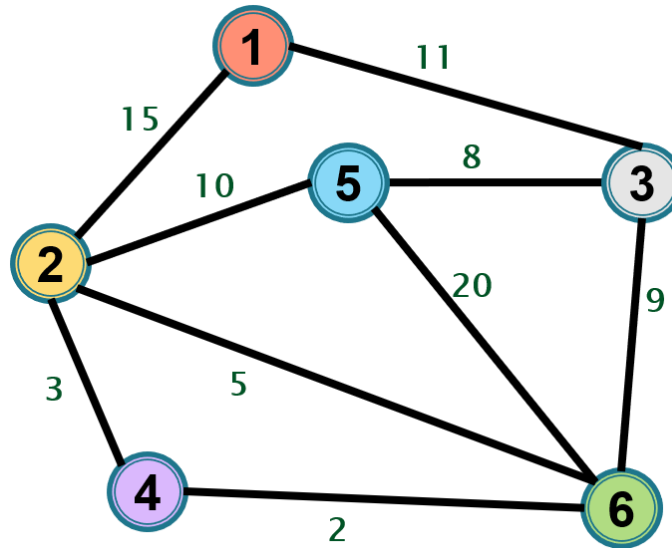
# Kruskal



La un pas componentele conexe din  $T$  sunt mulțimi disjuncte din  $V$  (partiție a lui  $V$ )

⇒ **structuri pentru mulțimi disjuncte**

- asociem fiecărei componente un reprezentant (o culoare)



# Kruskal

## ► Operații necesare:

- **Initializare(u)** –
- **Reprez(u)** –
- **Reuneste(u,v)** –

# Kruskal

## ► 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

# Kruskal

- ▶ O muchie  $uv$  unește două componente dacă și numai dacă

# Kruskal

- ▶ O muchie  $uv$  unește două componente dacă și numai dacă  
 **$\text{Reprez}(u) \neq \text{Reprez}(v)$**

# Kruskal

- ▶ O muchie  $uv$  unește două componente dacă și numai dacă  
 **$\text{Reprez}(u) \neq \text{Reprez}(v)$**



# Kruskal

**sorteaza** (E)

for (v=1 ; v<=n ; v++)

**Initializare** (v) ;

# Kruskal

**sorteaza** (E)

for (v=1 ; v<=n ; v++)

**Initializare** (v) ;

nrmsel=0

for (uv ∈ E)

    if (**Reprez** (u) != **Reprez** (v) )

    {

    }

# Kruskal

```
sorteaza (E)
```

```
for (v=1 ; v<=n ; v++)
```

```
    Initializare (v) ;
```

```
nrmse1=0
```

```
for (uv ∈ E)
```

```
    if (Reprez (u) !=Reprez (v) )
```

```
    {
```

```
        E (T)  = E (T)  ∪  {uv} ;
```

```
    }
```

# Kruskal

**sorteaza** (E)

for (v=1 ; v<=n ; v++)

**Initializare** (v) ;

nrmsel=0

for (uv ∈ E)

    if (**Reprez** (u) != **Reprez** (v) )

    {

$E(T) = E(T) \cup \{uv\};$

**Reuneste** (u,v) ;

        nrmsel=nrmsel+1;

        if (nrmsel==n-1)

            STOP;

    }

# Kruskal

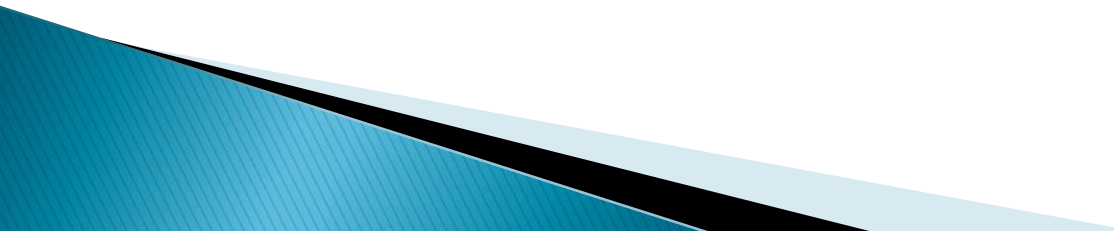
## Complexitate



De câte ori se execută fiecare operație?

# Kruskal

## Complexitate

- **Sortare**  $\rightarrow O(m \log m) = O(m \log n)$
  - ? \* **Initializare**
  - ? \* **Reprez**
  - ? \* **Reuneste**
- 

# Kruskal

## Complexitate

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

Depinde de modalitatea de memorare a componentelor

# Kruskal



Cum memorăm componentele + reprezentantul / culoarea componentei în care se află un vârf?

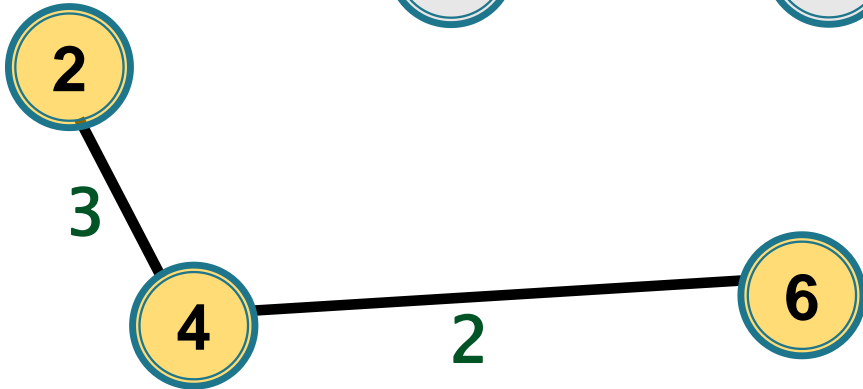


# Kruskal

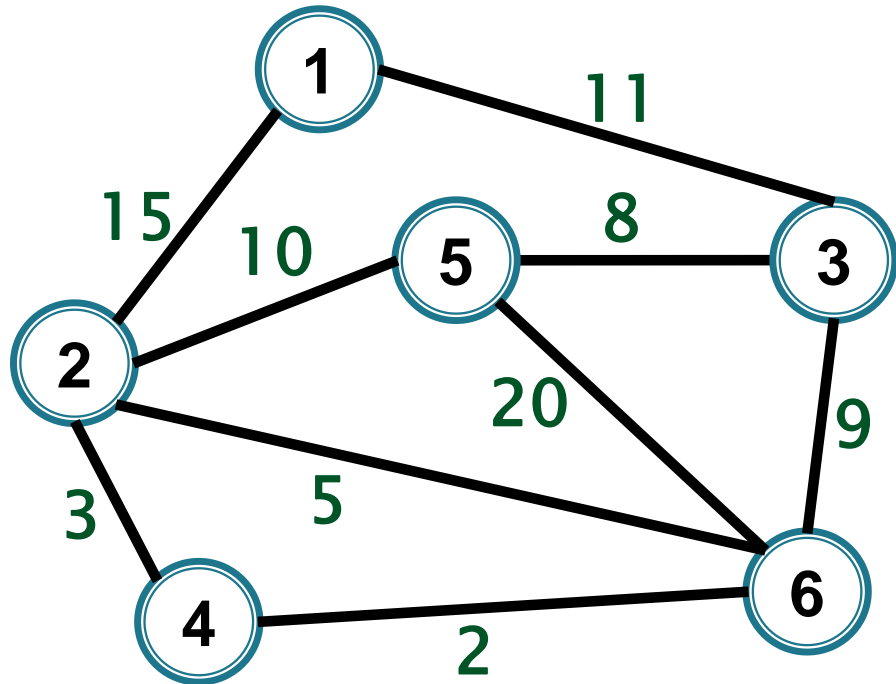


**Varianta 1** – memorăm într-un vector pentru fiecare vârf reprezentantul/culoarea componentei din care face parte

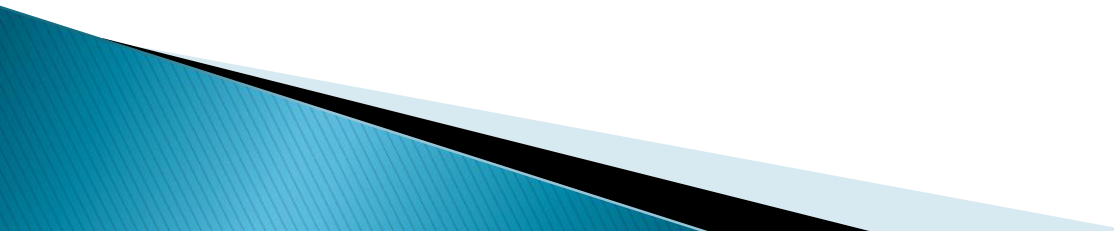
**$r[u]$  = culoarea (reprezentantul) componentei  
care conține vârful  $u$**



$r = [1, 2, 3, 2, 3, 2]$



# Kruskal

- ▶ **Initializare**
  - ▶ **Reprez**
  - ▶ **Reuneste**
- 

# Kruskal

- ▶ **Initializare –  $O(1)$**

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

- ▶ **Reprez**

- ▶ **Reuneste**

# Kruskal

- ▶ **Initializare –  $O(1)$**

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

- ▶ **Reprez –  $O(1)$**

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

- ▶ **Reuneste**

# Kruskal

## ► Initialize – $O(1)$

```
void Initialize(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;  
}
```

# Kruskal

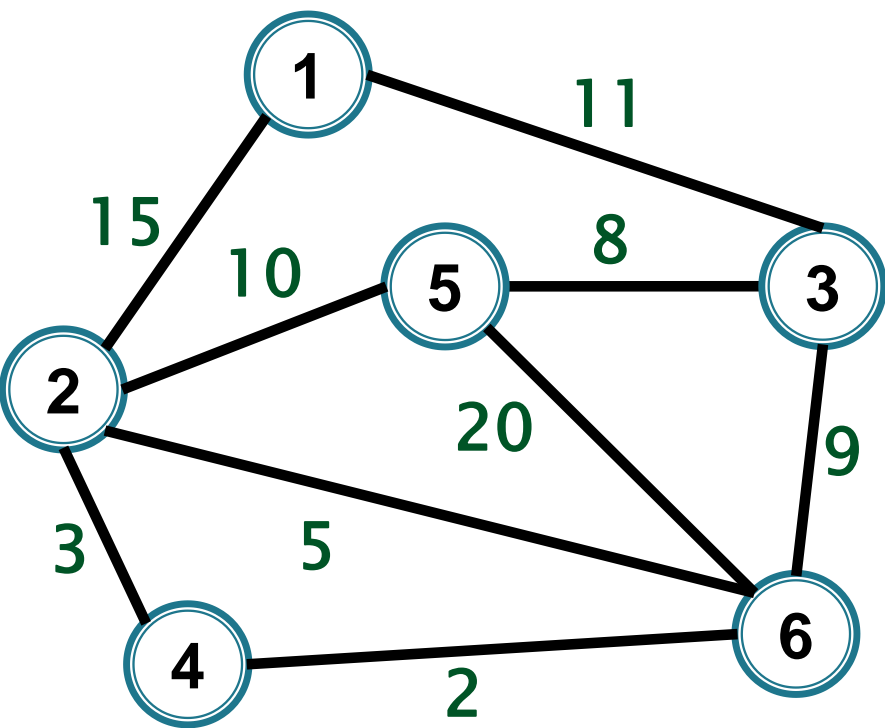
## Complexitate

**Varianta 1** – dacă folosim vector de reprezentanți

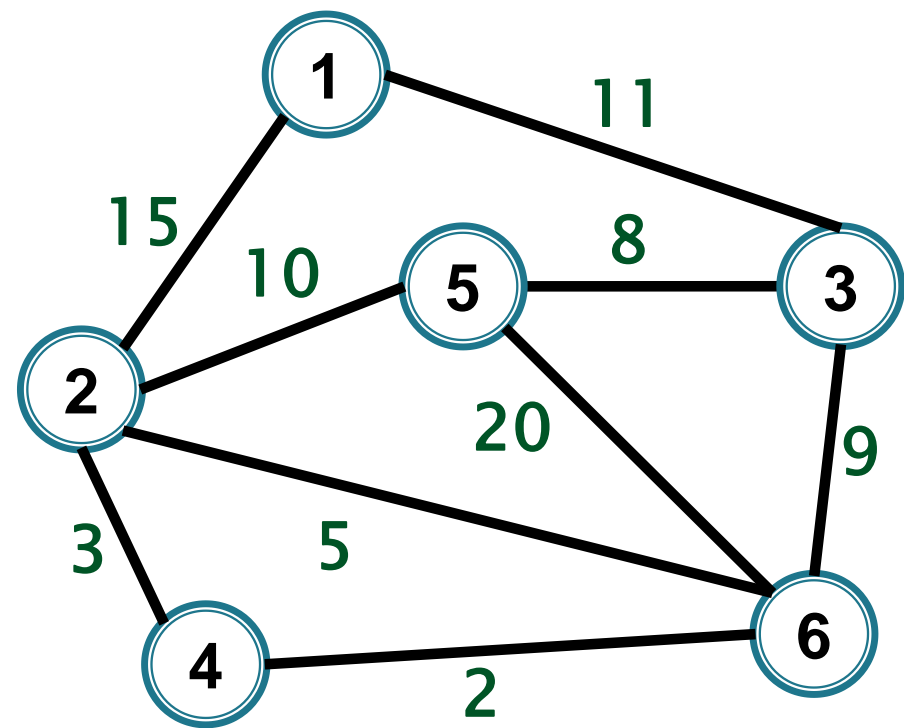
- **Sortare**  $\rightarrow O(m \log m) = O(m \log n)$
- $n$  \* **Initializare**  $\rightarrow O(n)$
- $2m$  \* **Reprez**  $\rightarrow O(m)$
- $(n-1)$  \* **Reuneste**  $\rightarrow O(n^2)$

---

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







(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

**(4,6)**

(2,4)

(2,6)

(3,5)

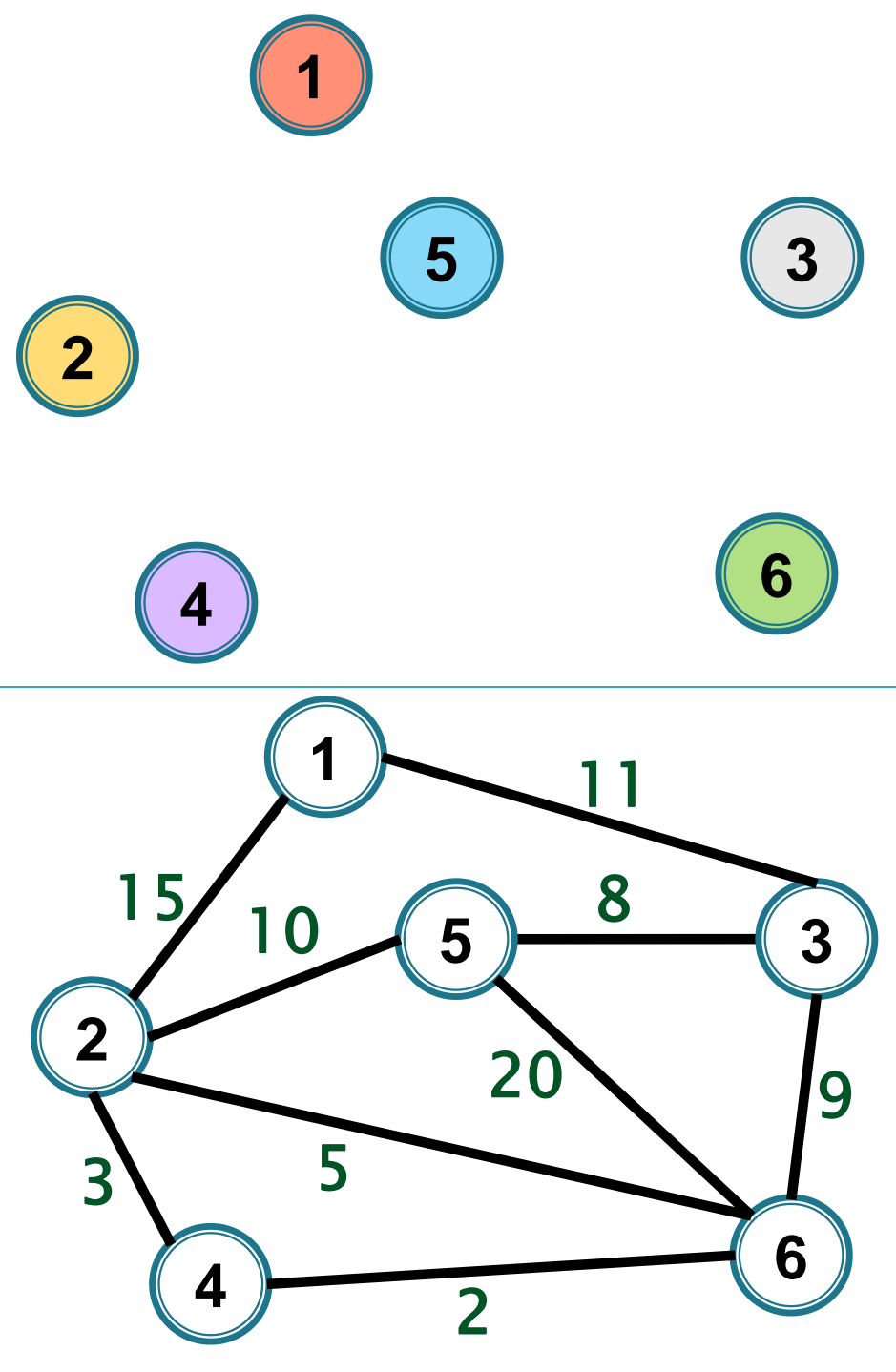
(3,6)

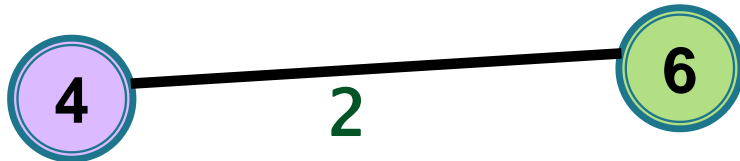
(2,5)

(1,3)

(1,2)

(5,6)





$r = [1, 2, 3, \underline{4}, 5, \underline{6}]$

**(4,6)**

$r(4) \neq r(6)$

(2,4)

(2,6)

(3,5)

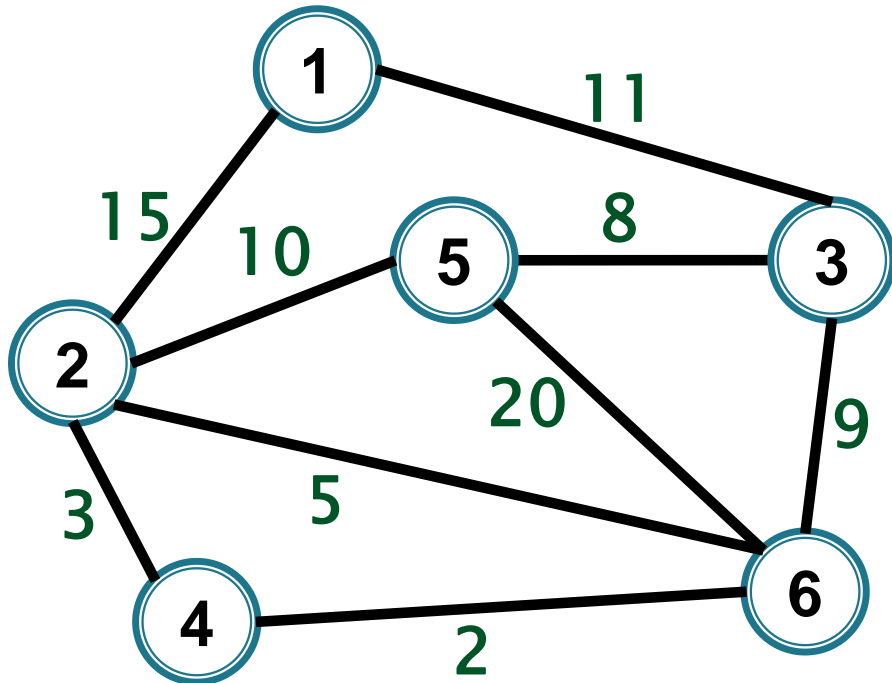
(3,6)

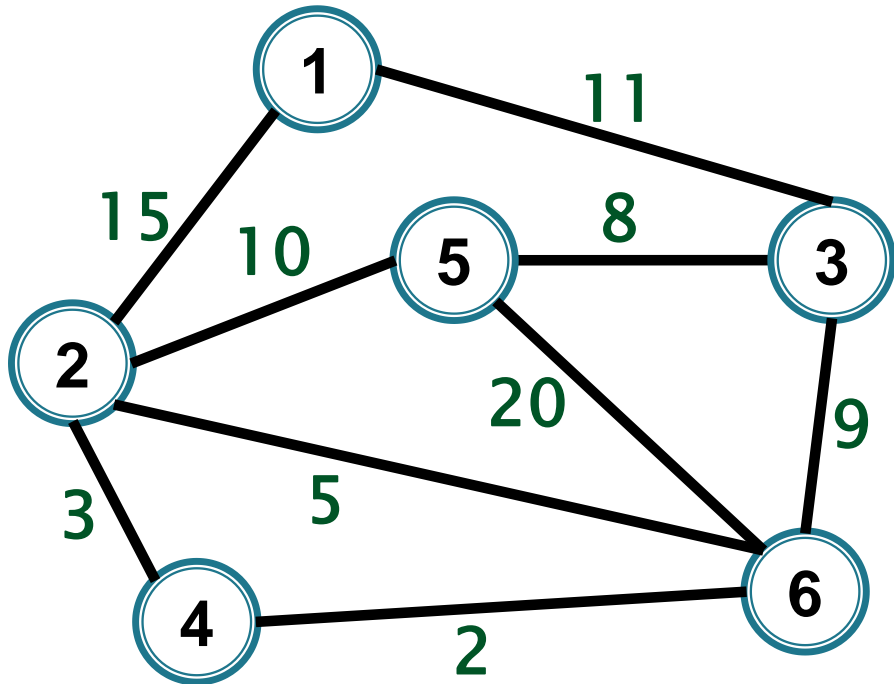
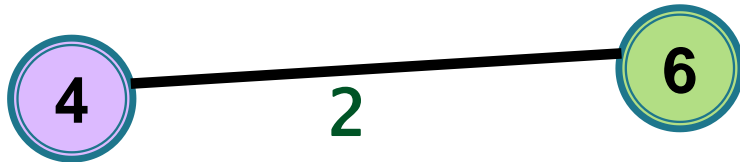
(2,5)

(1,3)

(1,2)

(5,6)





$r = [1, 2, 3, \underline{4}, 5, \underline{6}]$

**(4,6)**

Reuneste(4, 6)

(2,4)

(2,6)

(3,5)

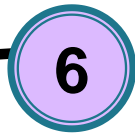
(3,6)

(2,5)

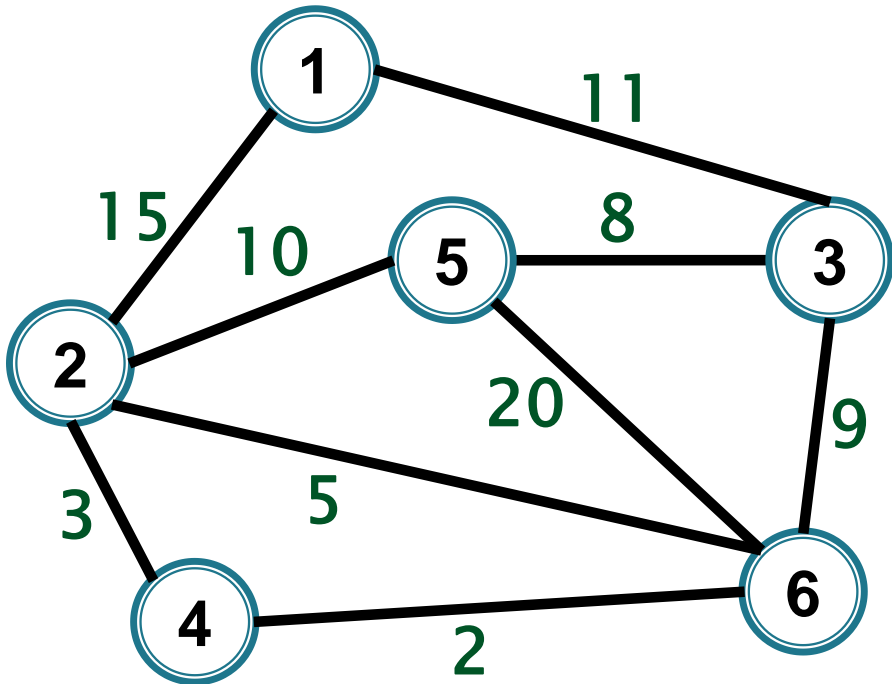
(1,3)

(1,2)

(5,6)



2



$r = [1, 2, 3, 4, 5, \underline{6}]$

**(4,6)**

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

(2,4)

(2,6)

(3,5)

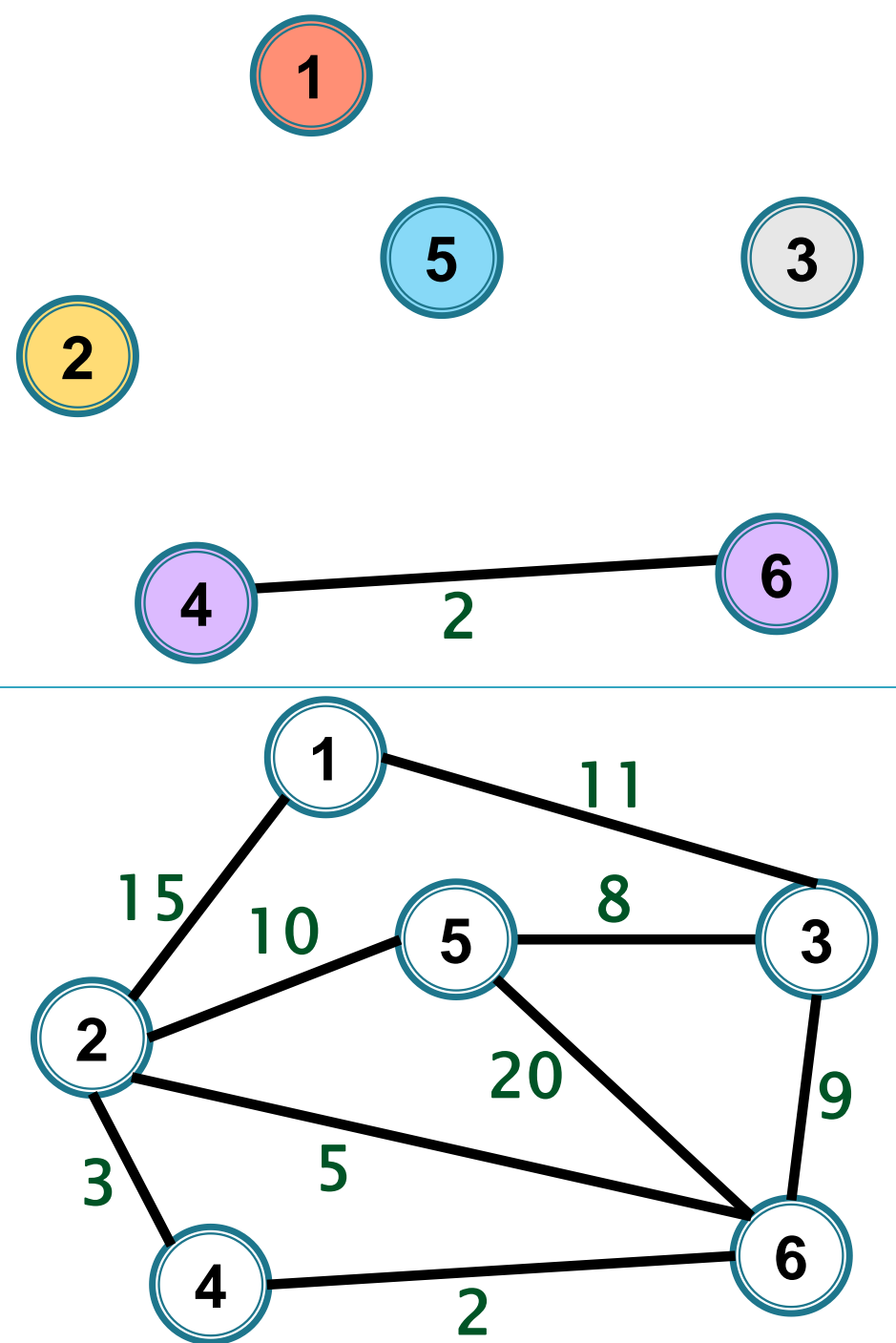
(3,6)

(2,5)

(1,3)

(1,2)

(5,6)



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

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

(4,6)

(2,4)

(2,6)

(3,5)

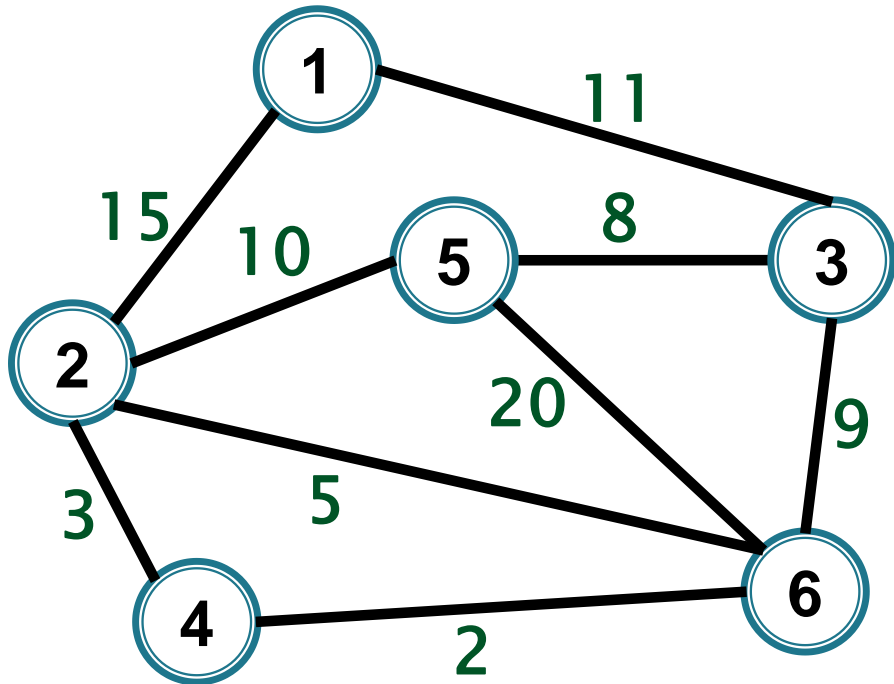
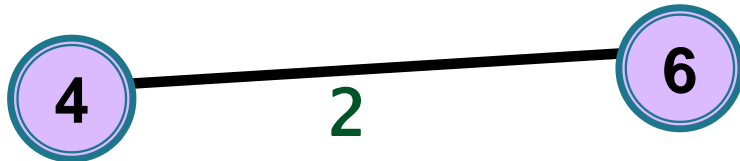
(3,6)

(2,5)

(1,3)

(1,2)

(5,6)



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

$(4, 6)$   $r = [1, \underline{2}, 3, \underline{4}, 5, 4]$

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

$(2, 6)$

$(3, 5)$

$(3, 6)$

$(2, 5)$

$(1, 3)$

$(1, 2)$

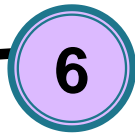
$(5, 6)$



3



2



11



15

10



8



20

3

5



2



9

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

$r = [1, \underline{2}, 3, \underline{4}, 5, 4]$

(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

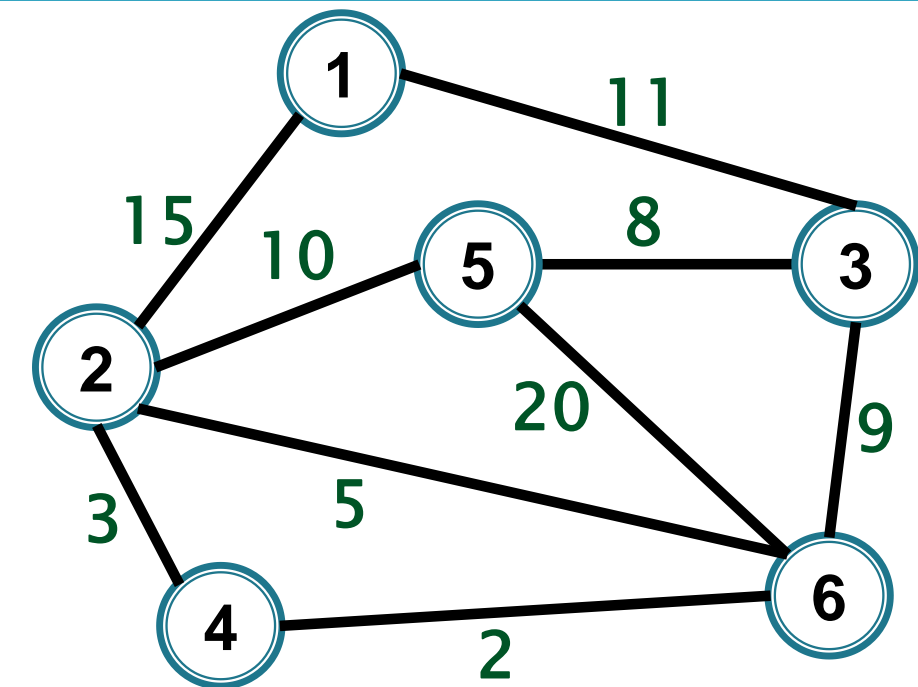
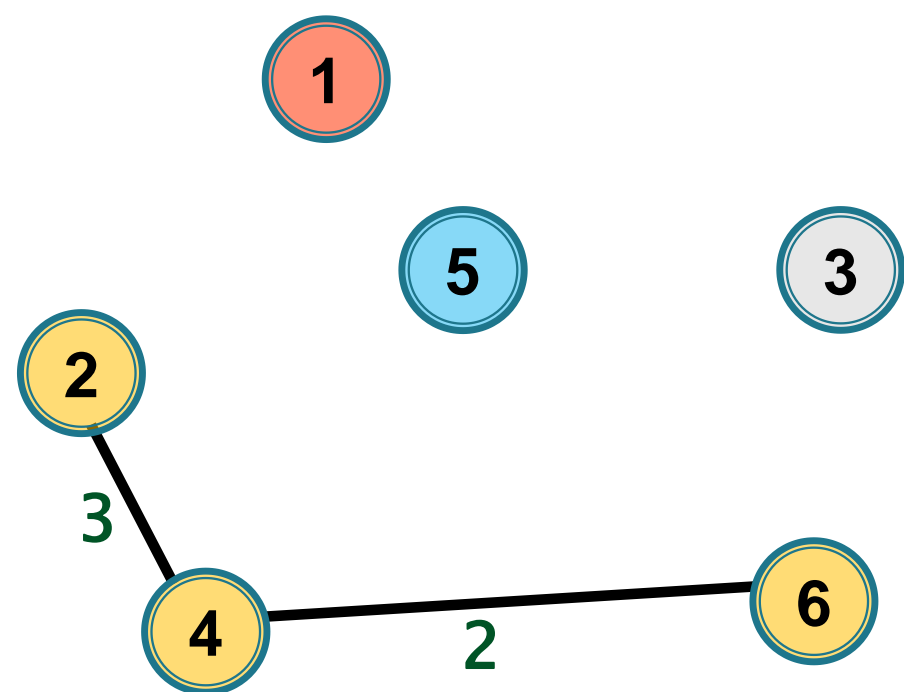
(2,5)

(1,3)

(1,2)

(5,6)





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

(4,6)  $r = [1, \underline{2}, 3, \underline{4}, 5, 4]$

(2,4)  $r = [1, 2, 3, \underline{2}, 5, 2]$

(2,6)

(3,5)

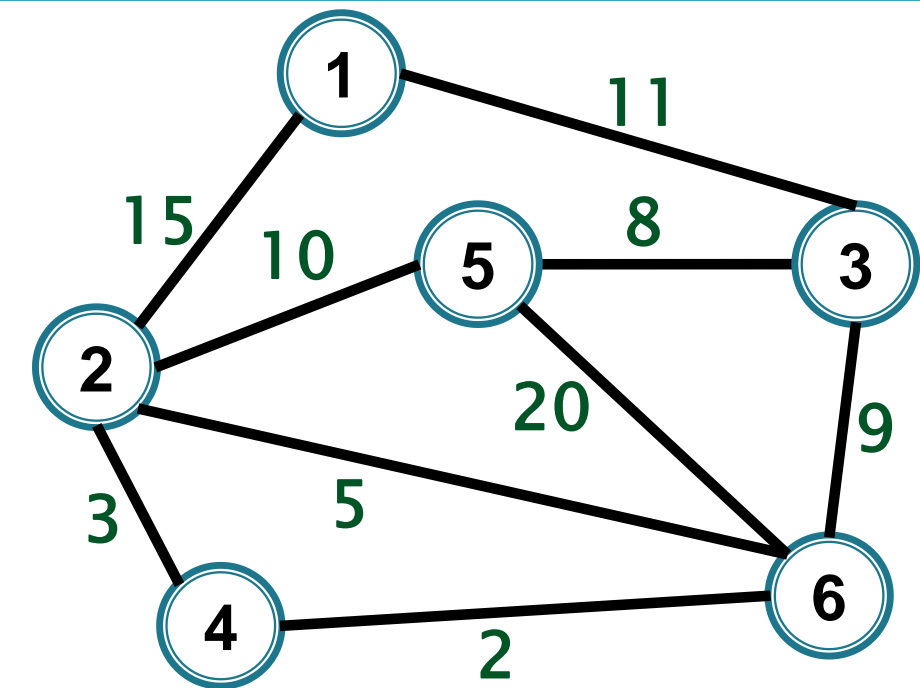
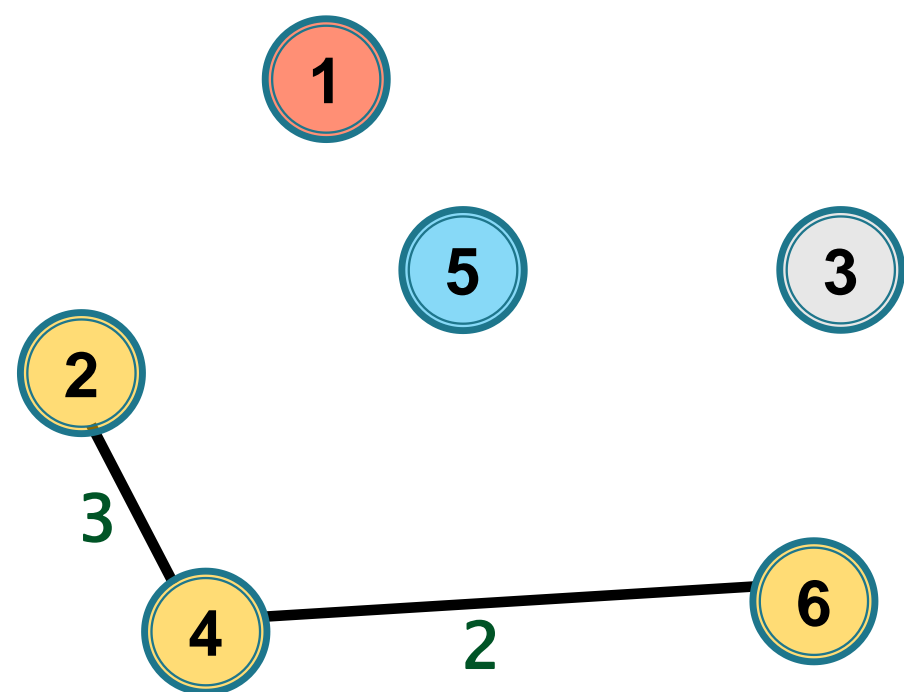
(3,6)

(2,5)

(1,3)

(1,2)

(5,6)



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

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

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

$(2, 6)$

$(3, 5)$

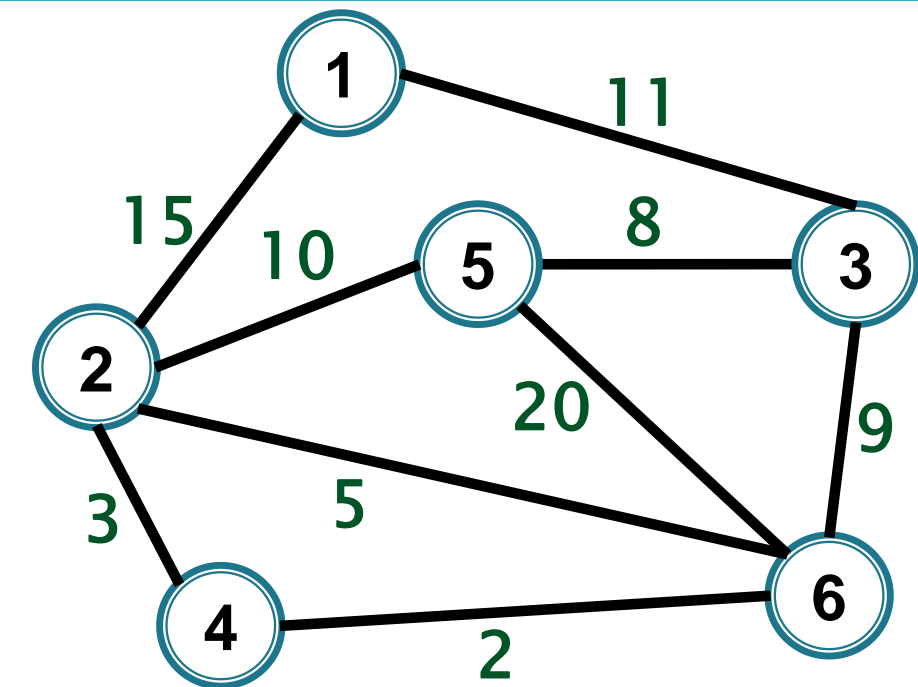
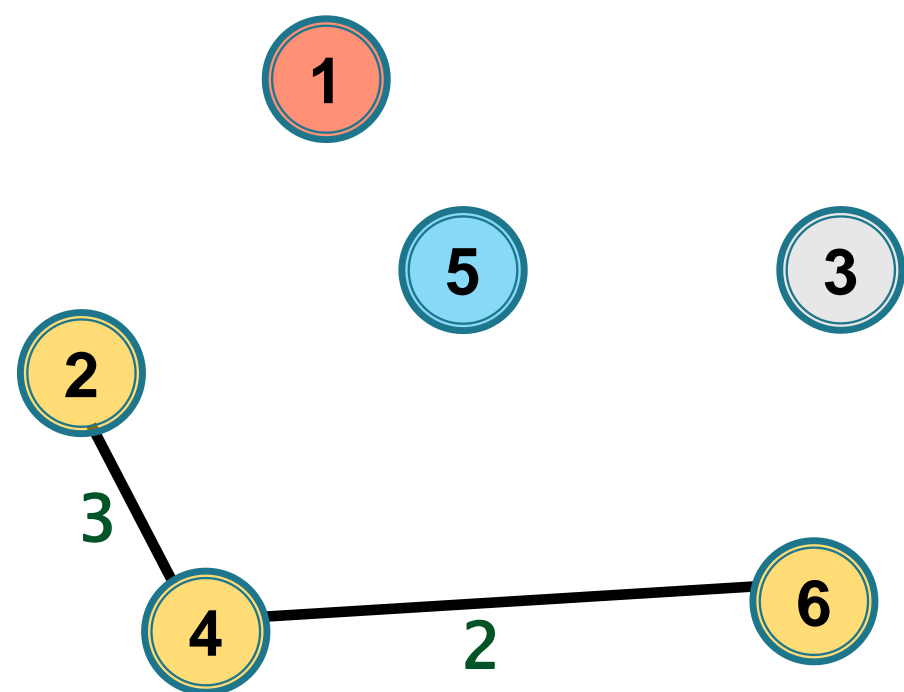
$(3, 6)$

$(2, 5)$

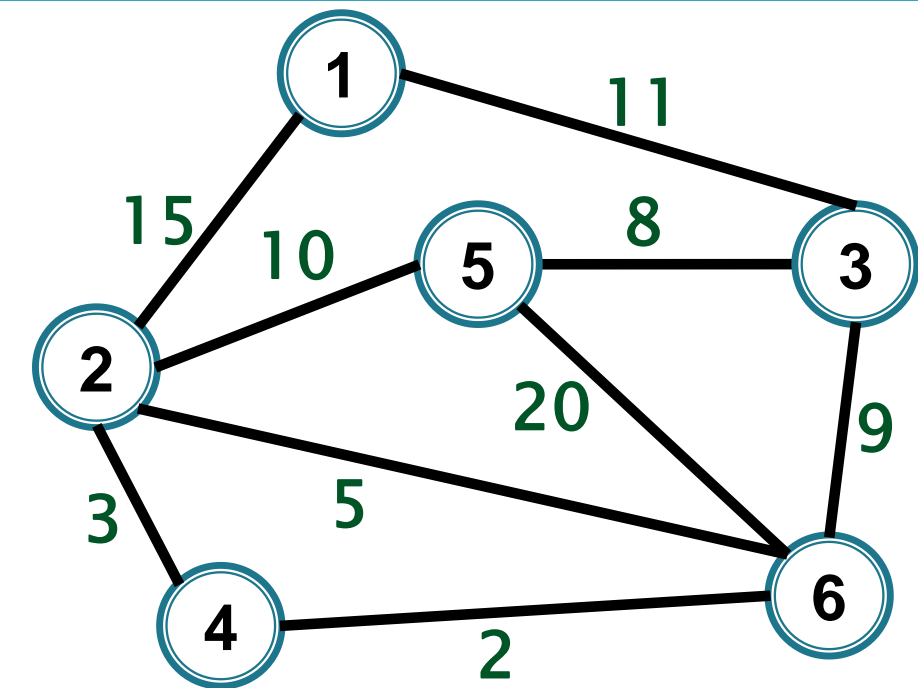
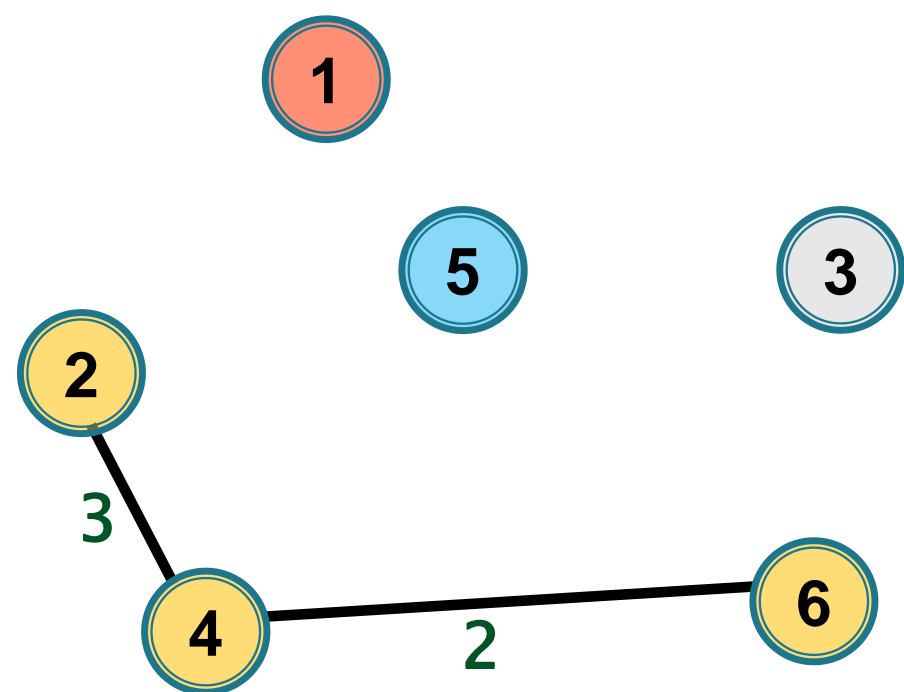
$(1, 3)$

$(1, 2)$

$(5, 6)$



$r = [1, 2, 3, 4, 5, 6]$   
 (4,6)  $r = [1, 2, 3, 4, 5, 4]$   
 (2,4)  $r = [1, \underline{2}, 3, 2, 5, \underline{2}]$   
 (2,6)  $r(2) = r(6) \rightarrow \text{NU}$   
 (3,5)  
 (3,6)  
 (2,5)  
 (1,3)  
 (1,2)  
 (5,6)



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

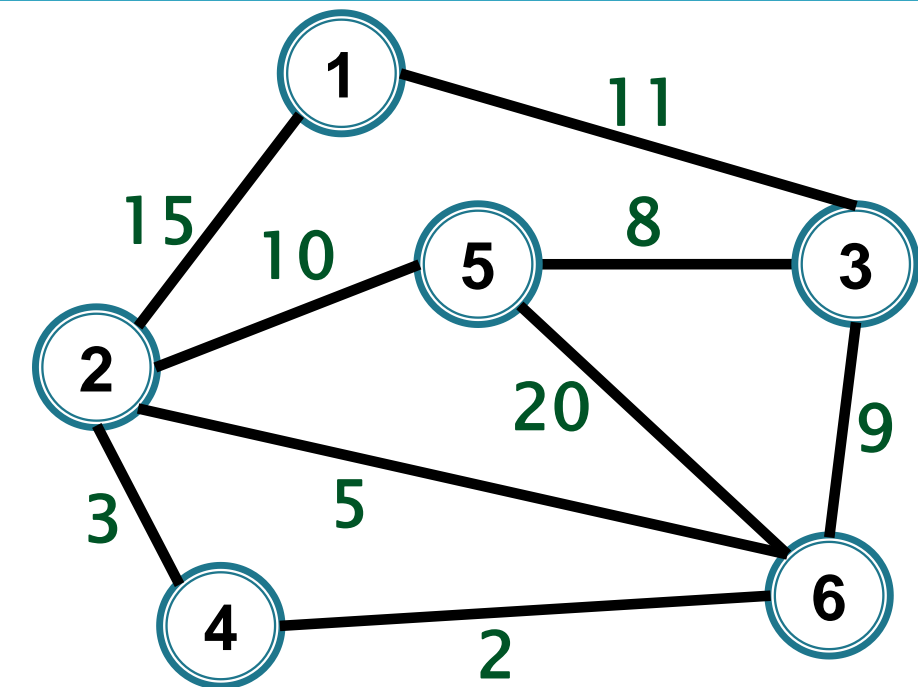
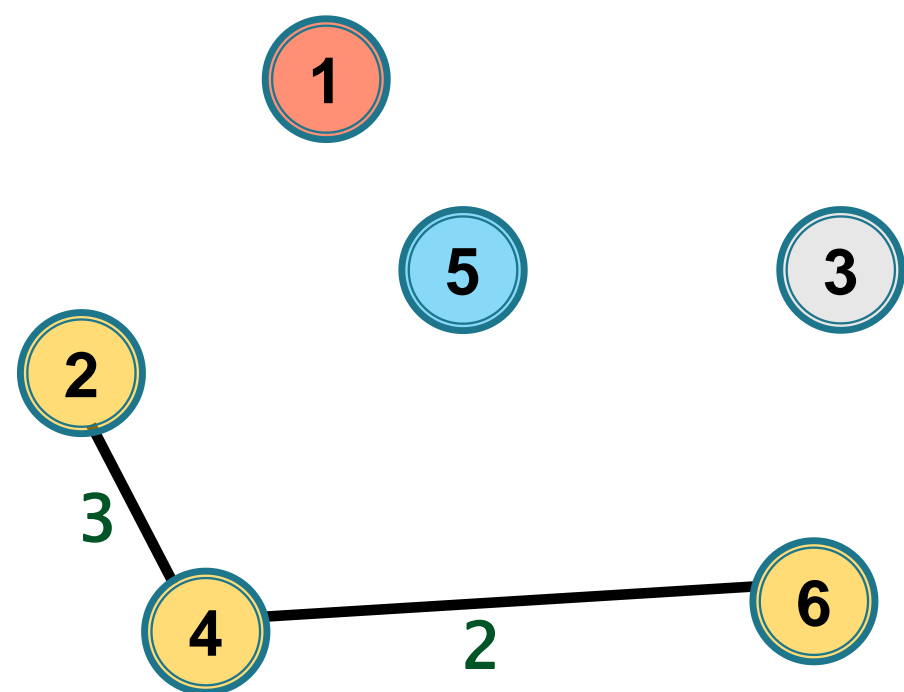
(5,6)

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

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

$r = [1, 2, 3, 2, 5, 2]$

$r(2) = r(6) \rightarrow \text{NU}$



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

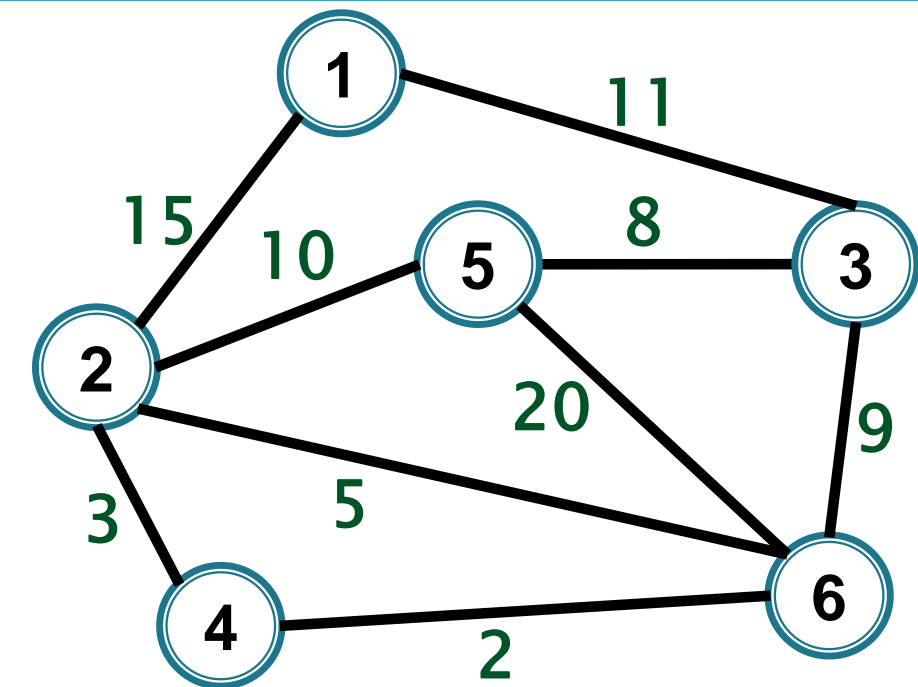
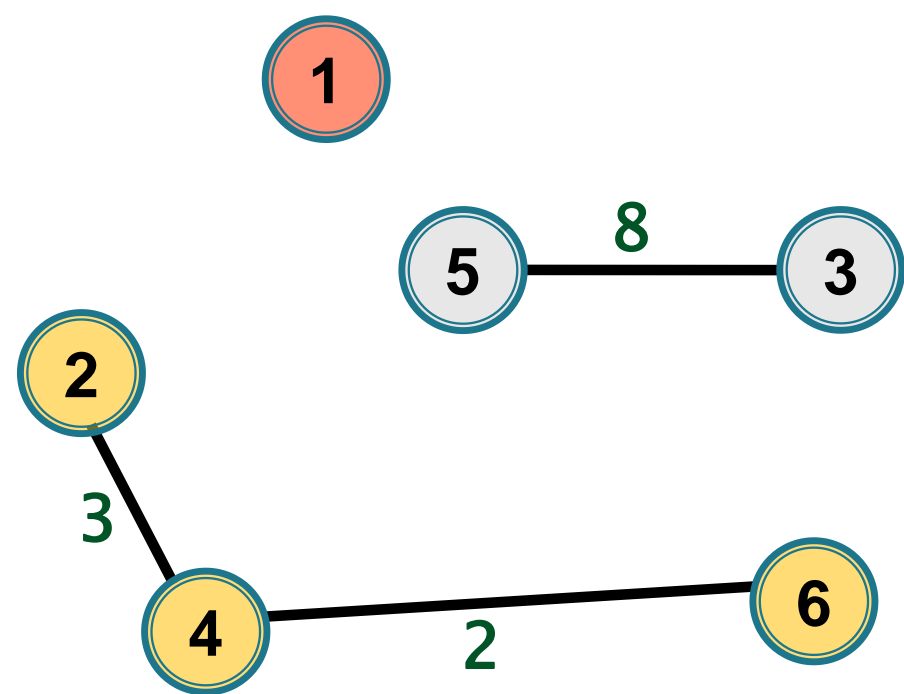
$r = [1, 2, 3, 4, 5, 6]$

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

$r = [1, 2, \underline{3}, 2, \underline{5}, 2]$

$r(2) = r(6) \rightarrow \text{NU}$

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



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

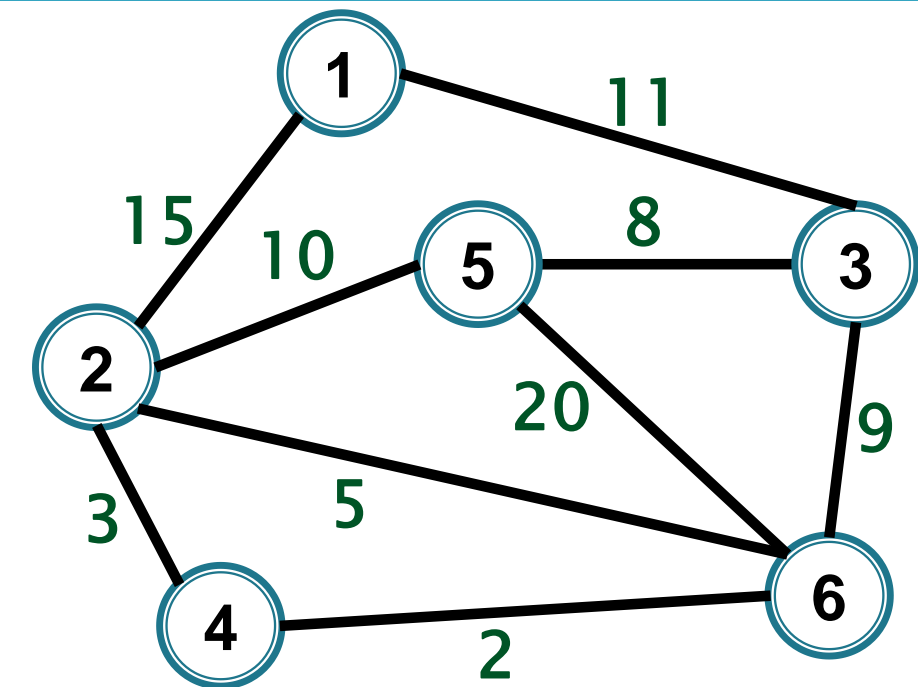
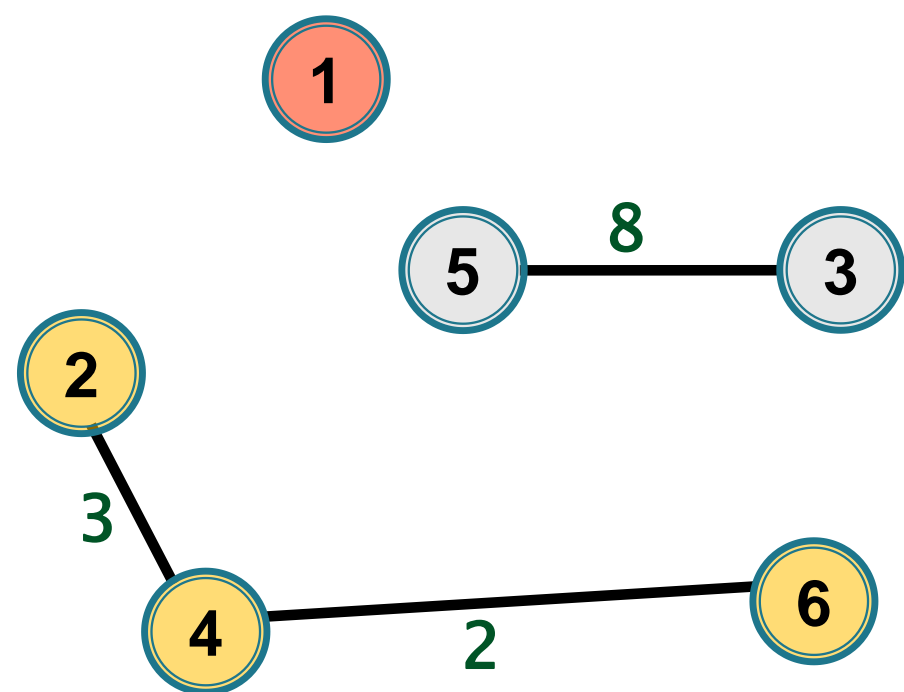
$r = [1, 2, 3, 4, 5, 6]$

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

$r = [1, 2, \underline{3}, 2, \underline{5}, 2]$

$r(2) = r(6) \rightarrow \text{NU}$

$r = [1, 2, 3, 2, \underline{3}, 2]$



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

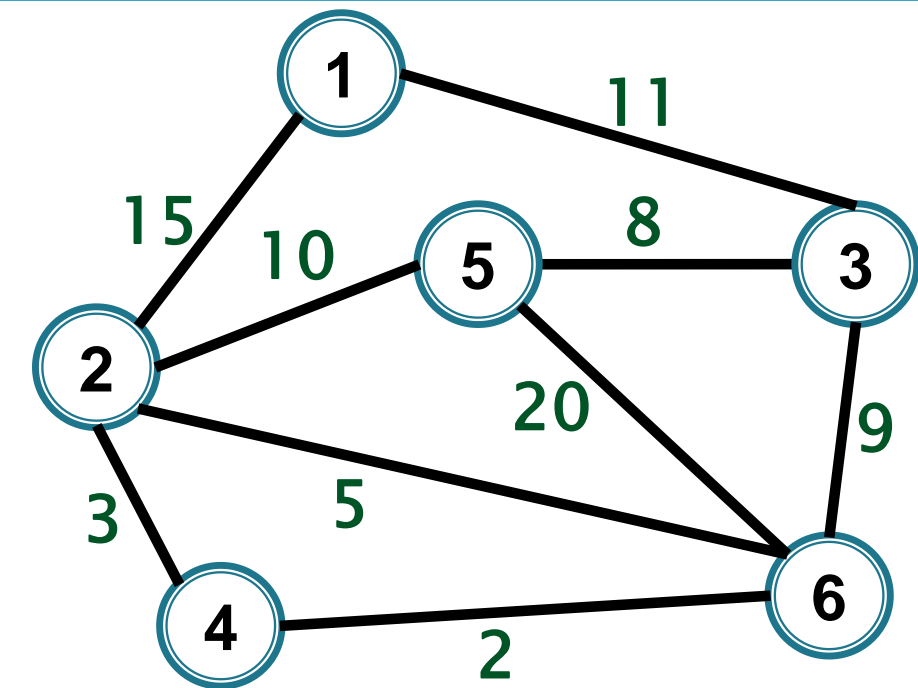
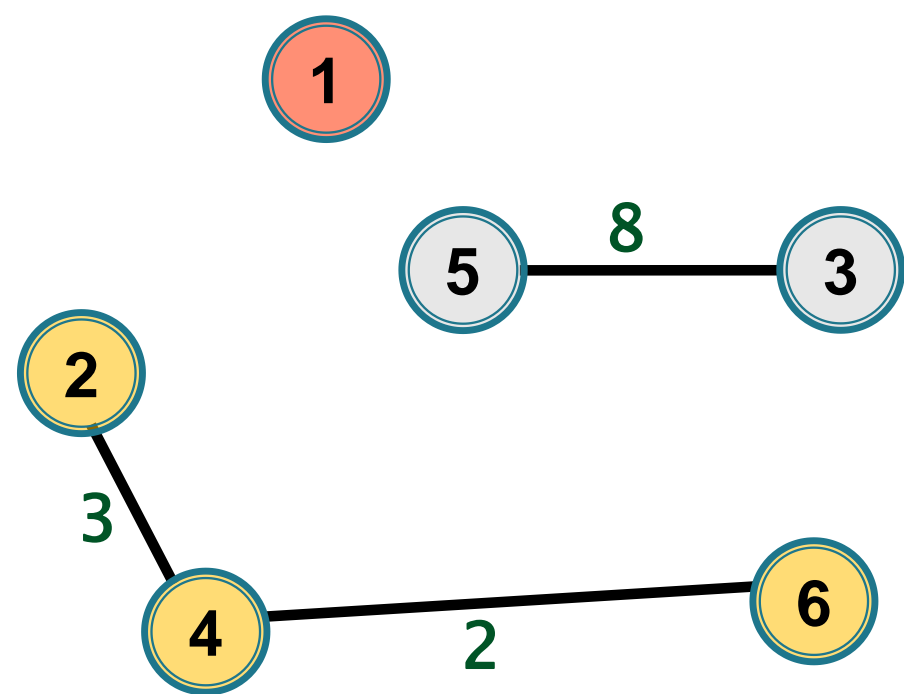
$r = [1, 2, 3, 4, 5, 6]$

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

$r = [1, 2, 3, 2, 5, 2]$

$r(2) = r(6) \rightarrow \text{NU}$

$r = [1, 2, 3, 2, 3, 2]$



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

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

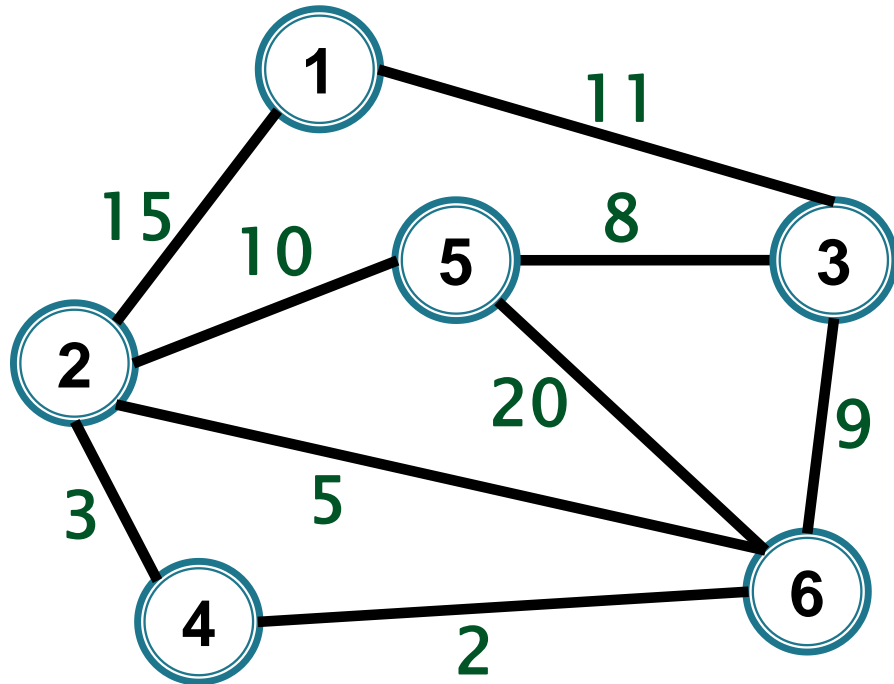
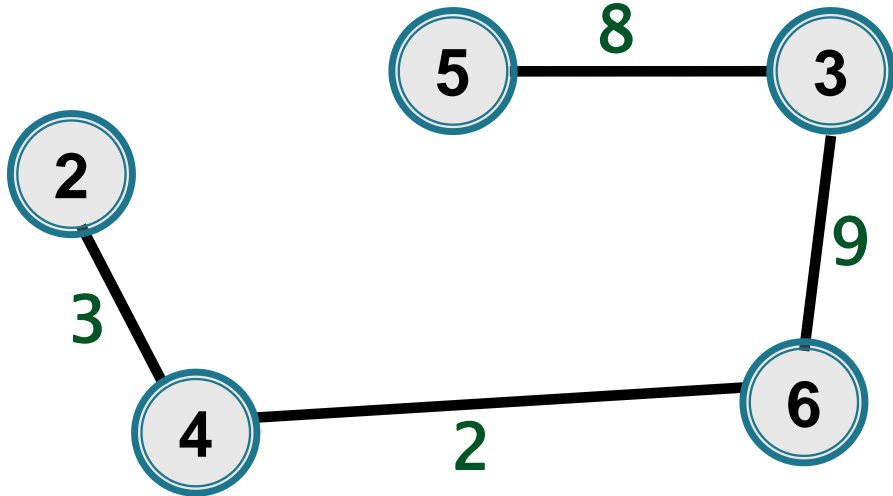
$r = [1, 2, 3, 2, 5, 2]$

$r(2) = r(6) \rightarrow \text{NU}$

$r = [1, 2, \underline{3}, 2, 3, \underline{2}]$

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





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

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

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

$(2, 6)$   $r(2) = r(6) \rightarrow \text{NU}$

$(3, 5)$   $r = [1, 2, \underline{3}, 2, 3, \underline{2}]$

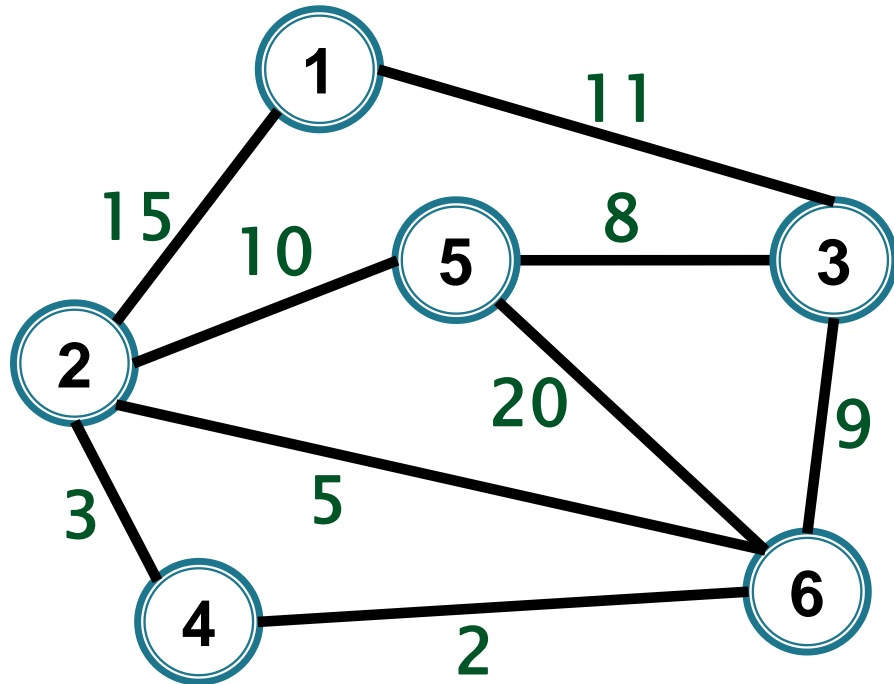
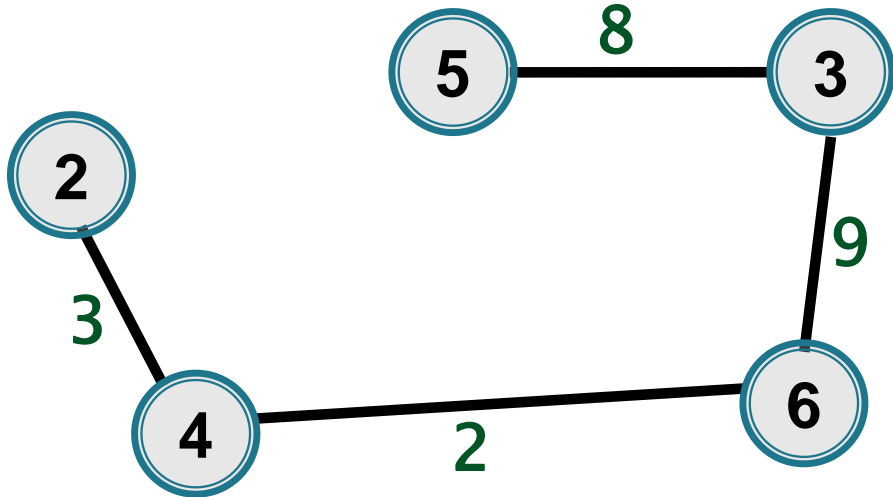
$(3, 6)$   $r = [1, \underline{3}, 3, \underline{3}, 3, \underline{3}]$

$(2, 5)$

$(1, 3)$

$(1, 2)$

$(5, 6)$



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

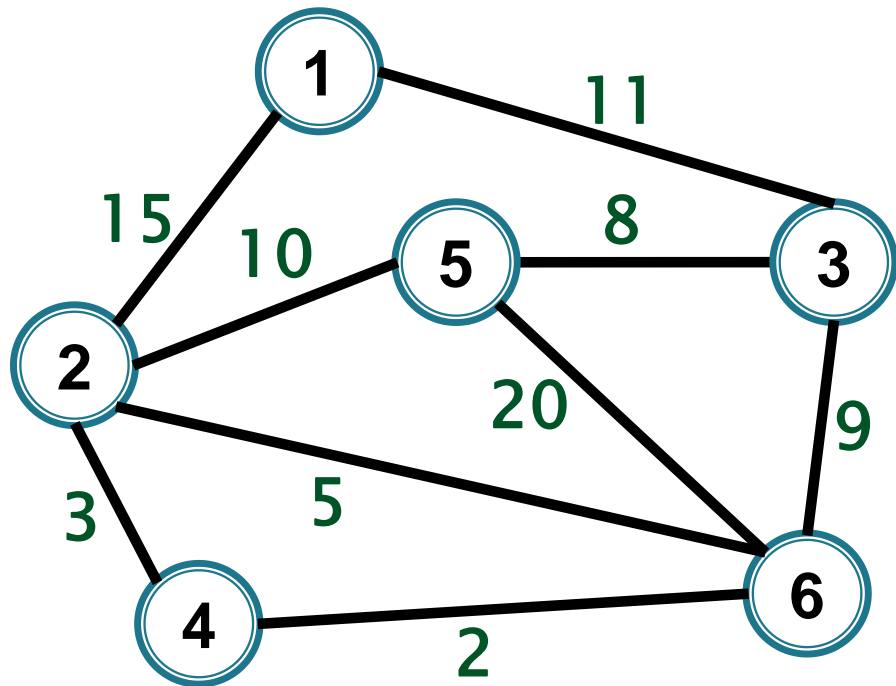
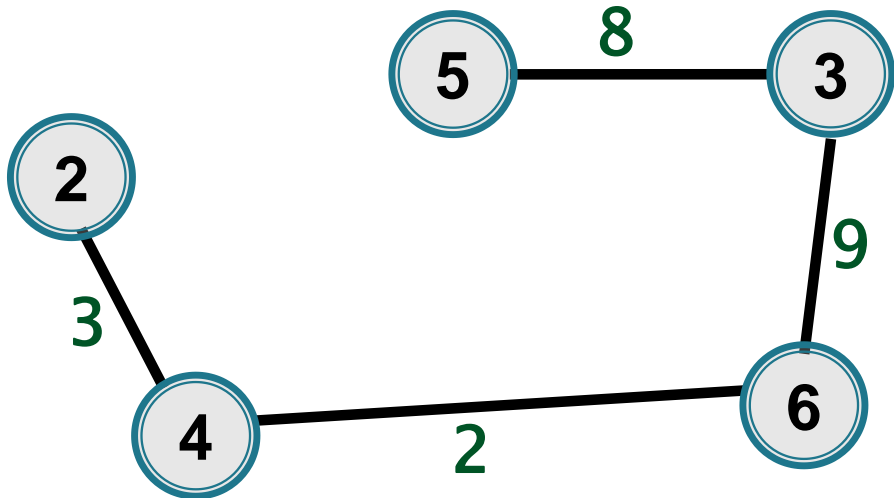
$r = [1, 2, 3, 4, 5, 4]$

$r = [1, 2, 3, 2, 5, 2]$

$r(2) = r(6) \rightarrow \text{NU}$

$r = [1, 2, 3, 2, 3, 2]$

$r = [1, 3, 3, 3, 3, 3]$



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

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

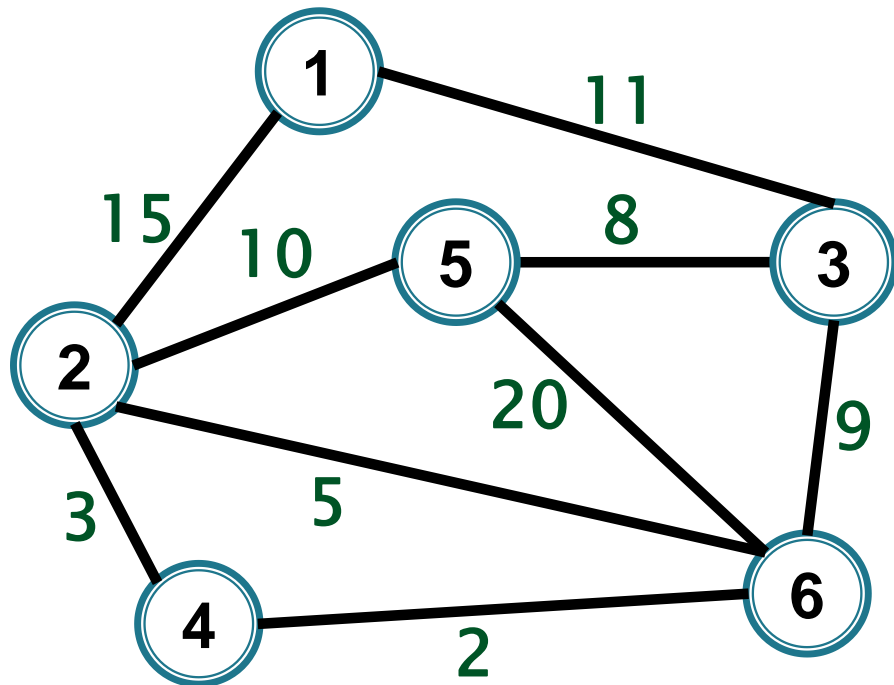
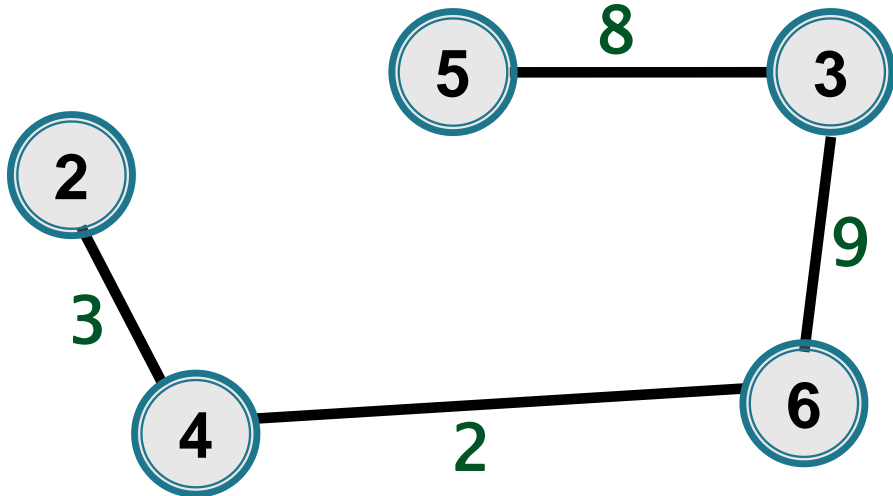
$r = [1, 2, 3, 2, 5, 2]$

$r(2) = r(6) \rightarrow \text{NU}$

$r = [1, 2, 3, 2, 3, 2]$

$r = [1, \underline{3}, 3, 3, \underline{3}, 3]$

$r(2) = r(5) \rightarrow \text{NU}$



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

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

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

$(2, 6)$   $r(2) = r(6) \rightarrow \text{NU}$

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

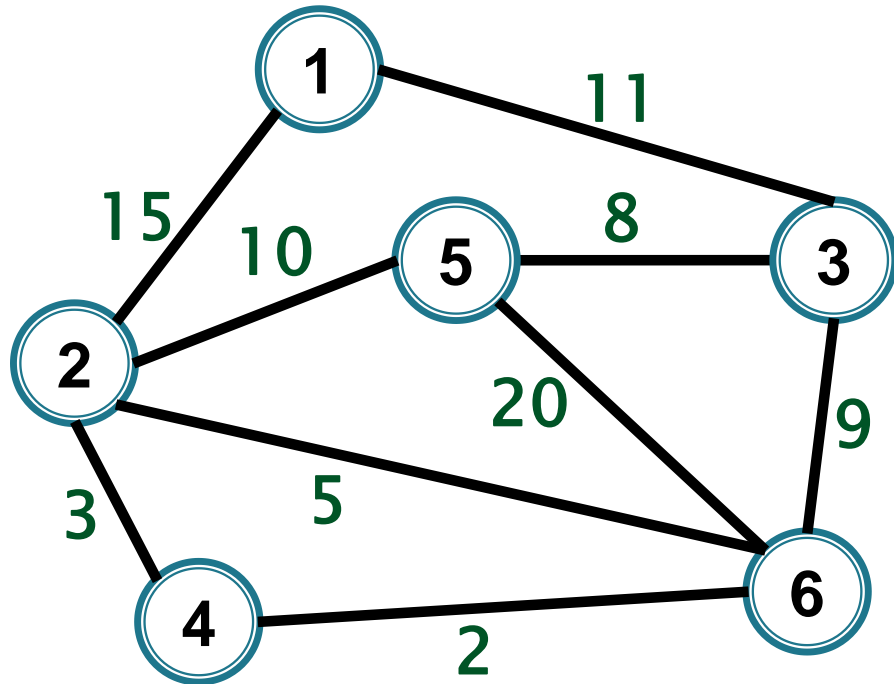
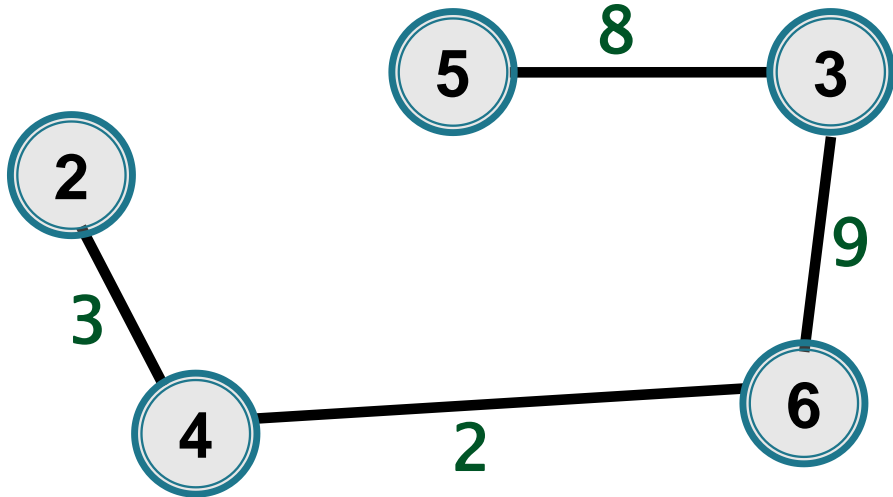
$(3, 6)$   $r = [1, 3, 3, 3, 3, 3]$

$(2, 5)$   $r(2) = r(5) \rightarrow \text{NU}$

$(1, 3)$

$(1, 2)$

$(5, 6)$



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

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

$r = [1, 2, 3, 2, 5, 2]$

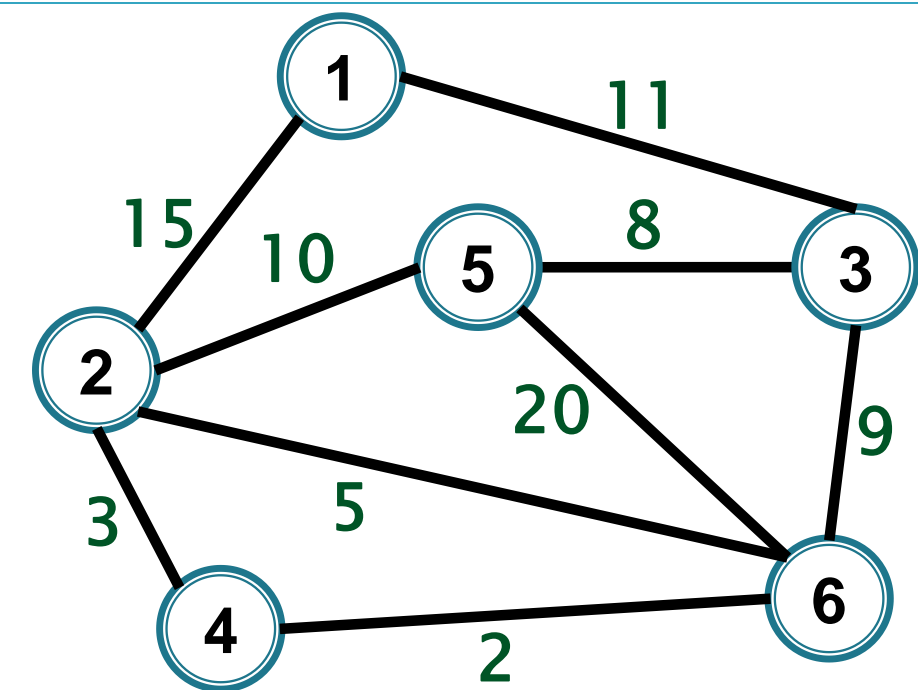
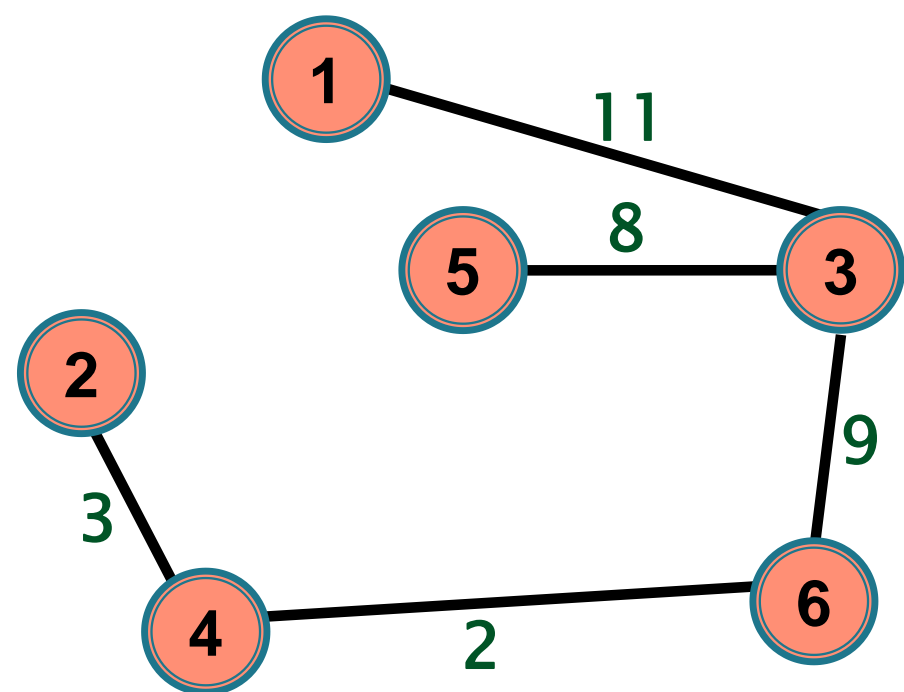
$r(2) = r(6) \rightarrow \text{NU}$

$r = [1, 2, 3, 2, 3, 2]$

$r = [1, 3, 3, 3, 3, 3]$

$r(2) = r(5) \rightarrow \text{NU}$

$r(1) \neq r(3)$



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

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

$r = [1, 2, 3, 2, 5, 2]$

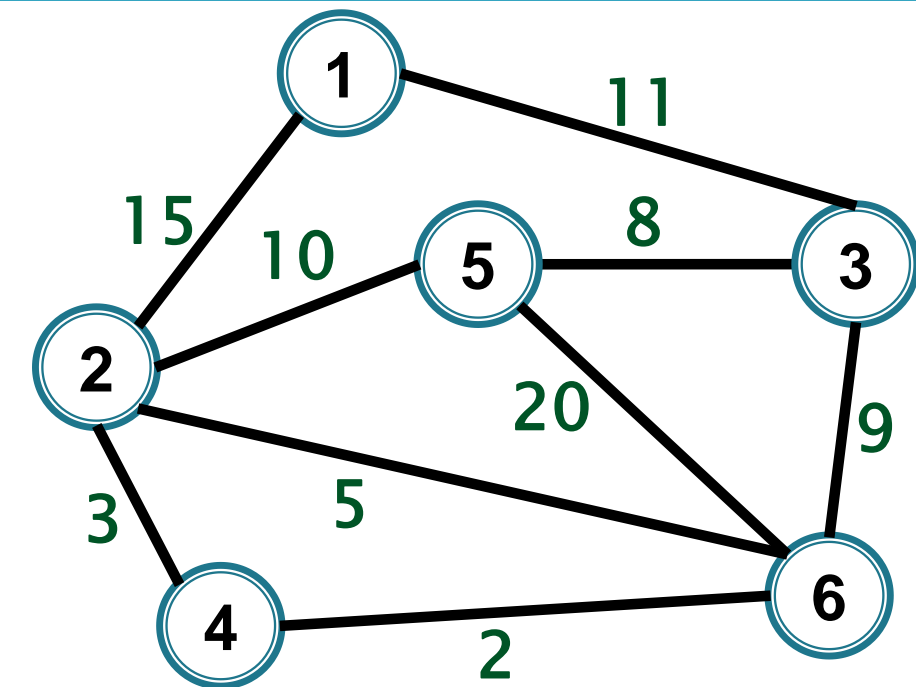
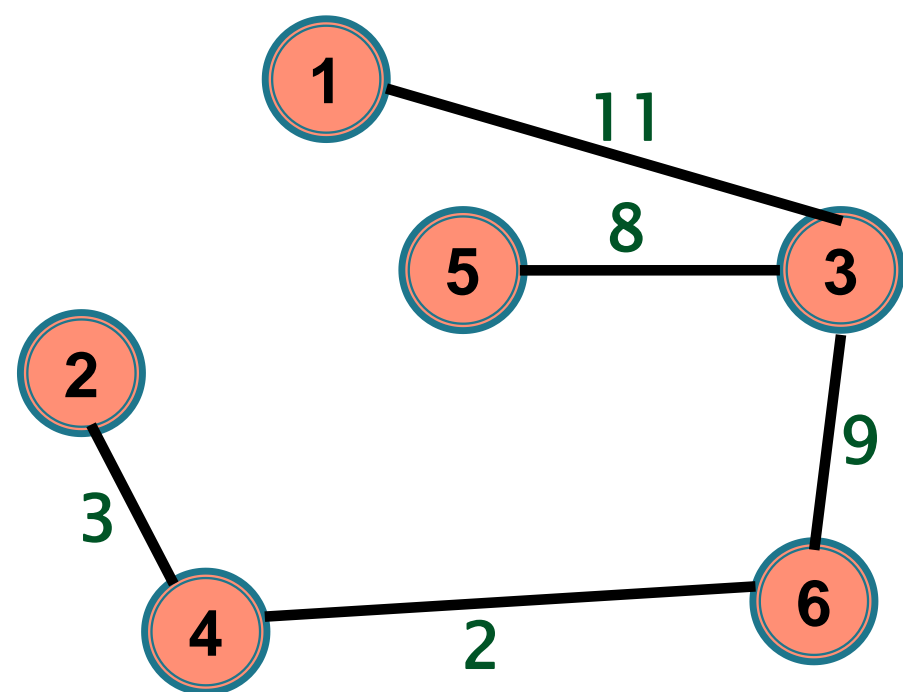
$r(2) = r(6) \rightarrow \text{NU}$

$r = [1, 2, 3, 2, 3, 2]$

$r = [1, 3, 3, 3, 3, 3]$

$r(2) = r(5) \rightarrow \text{NU}$

$r = [1, 1, 1, 1, 1, 1]$



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

**STOP**

(1,2)

(5,6)

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

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

$r = [1, 2, 3, 2, 5, 2]$

$r(2) = r(6) \rightarrow \text{NU}$

$r = [1, 2, 3, 2, 3, 2]$

$r = [1, 3, 3, 3, 3, 3]$

$r(2) = r(5) \rightarrow \text{NU}$

$r = [1, 1, 1, 1, 1, 1]$

# Kruskal



## **Varianta 2 – Structuri pentru mulțimi disjuncte Union/Find**

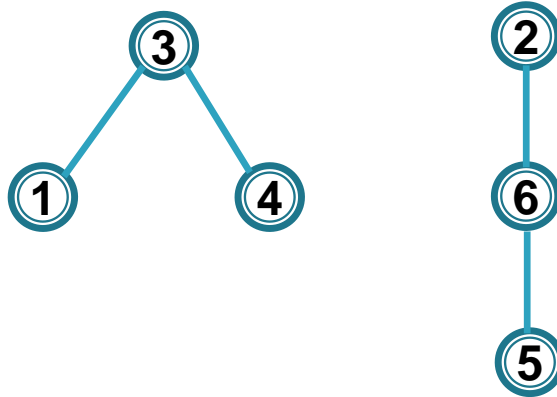


# Kruskal



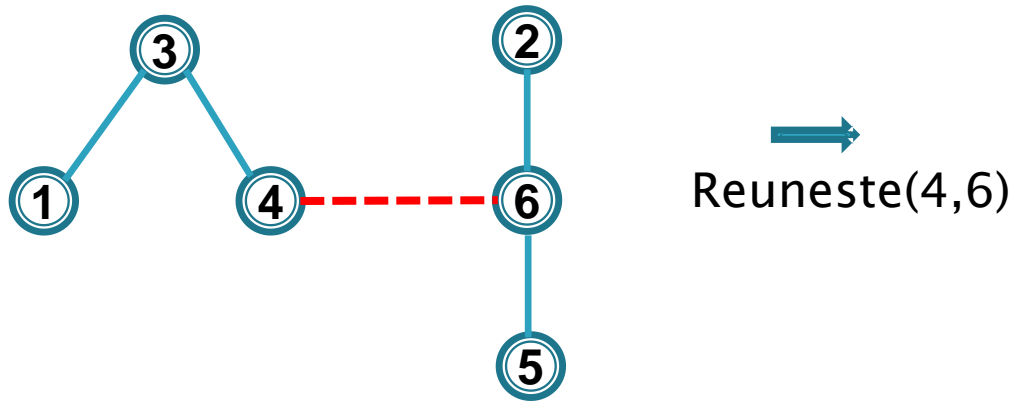
## Varianta 2 – Structuri pentru mulțimi disjuncte Union/Find – arbori cu rădăcină

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



# Kruskal

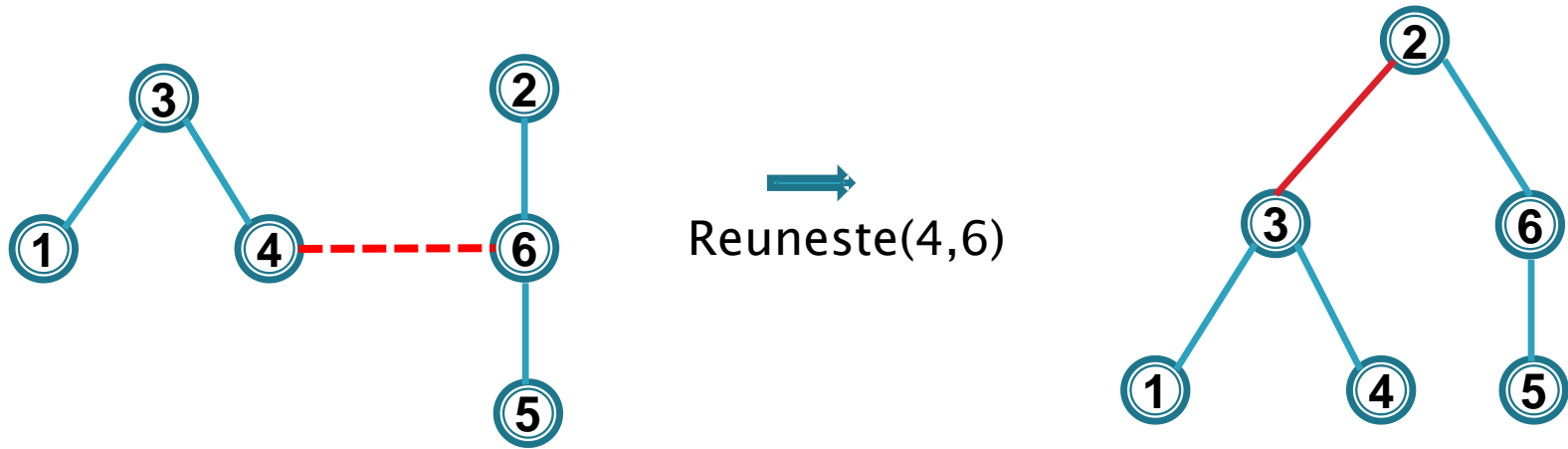
- **Reuniunea** a doi arbori  $\Rightarrow$  rădăcina unui arbore devine fiu al rădăcinii celuilalt arbore



# Kruskal

- Reuniunea se va face în funcție de înălțimea/dimensiunea arborilor (reuniune ponderată), pentru a obține arbori de înălțime mică

⇒ **arbori de înălțime logaritmică**



- arborele cu înălțimea mai mică devine subarbore al rădăcinii celuilalt arbore

# Kruskal

Detalii de implementare operații cu structuri Union/Find pentru mulțimi disjuncte:

- **Initializare**
- **Reprez(u)**  $\Rightarrow$  determinarea rădăcinii arborelui care conține u  
+ **compresie de cale** (v. seminar+laborator)
- **Reuneste(u,v)**  $\Rightarrow$  reuniune ponderată

# Kruskal

```
void Initializare(int u){  
    tata[u]=h[u]=0;  
}
```

```
int Reprez(int u){  
    while(tata[u]!=0)  
        u = tata[u];  
    return u;  
}
```

# Kruskal

```
void Initializare(int u){  
    tata[u]=h[u]=0;  
}
```

```
int Reprez(int u){  
    while(tata[u]!=0)  
        u = tata[u];  
    return u;  
}
```

```
void Reuneste(int u,int v)  
{  
    int ru,rv;  
    ru = Reprez(u);  
    rv = Reprez(v);  
    if (h[ru] > h[rv])  
        tata[rv] = ru;  
    else{  
        tata[ru] = rv;  
  
    }  
}
```

# Kruskal

```
void Initializare(int u){  
    tata[u]=h[u]=0;  
}
```

```
int Reprez(int u){  
    while(tata[u]!=0)  
        u = tata[u];  
    return u;  
}
```

```
void Reuneste(int u,int v)  
{  
    int ru,rv;  
    ru = Reprez(u);  
    rv = Reprez(v);  
    if (h[ru] > h[rv])  
        tata[rv] = ru;  
    else{  
        tata[ru] = rv;  
        if(h[ru] == h[rv])  
            h[rv] = h[rv]+1;  
    }  
}
```

# Kruskal

## Complexitate

**Varianta 2** – dacă folosim arbori Union/Find

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



# Kruskal

## Complexitate

**Varianta 2** – dacă folosim arbori Union/Find

- **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**  $\rightarrow O(n \log n)$
-

# Kruskal

## Complexitate

**Varianta 2** – dacă folosim arbori Union/Find

- **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**  $\rightarrow O(n \log n)$
- mai mică dacă folosim și compresie de cale

---

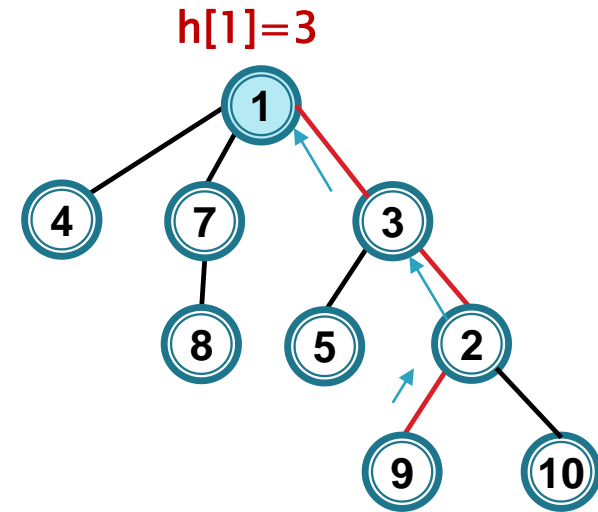
$O(m \log n)$

# Kruskal

## Compresie de cale

```
int Reprez(int u){  
    if (tata[u]==0)  
        return u;  
    tata[u]=Reprez(tata[u]);  
    return tata[u];  
}
```

După apelul **Reprez(9)** pentru arborele



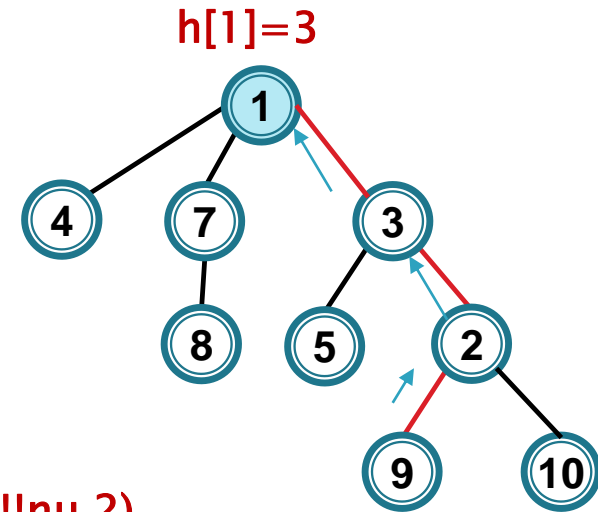
rezultatul va fi 1, iar arborele devine

# Kruskal

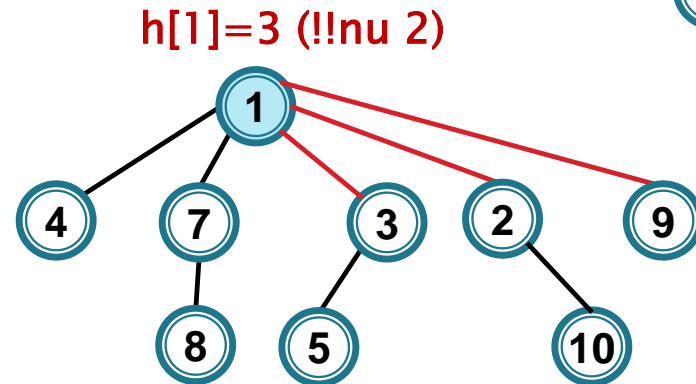
## Compresie de cale

```
int Re prez(int u){  
    if (tata[u]==0)  
        return u;  
    tata[u]=Re prez(tata[u]);  
    return tata[u];  
}
```

După apelul `Re prez(9)` pentru arborele



rezultatul va fi 1, iar arborele devine



# Kruskal

## Concluzii complexitate

- Vector de culori  $O(m \log n + n^2)$
- Structuri union/find  $O(m \log n)$

# Kruskal

## Temă

- ▶ Pentru grafuri dense ( $m$  de ordin  $n^2$ ) ce complexitate are algoritmul și pentru ce structuri se obține?
- ▶ Dacă ponderile sunt în mulțimea  $\{1, \dots, k\}$ ,  $k < 100$ , ce complexitate are algoritmul lui Kruskal?
  - se pot sorta muchiile cu o complexitate mai mică decât  $O(m \log(n))$ ?
- ▶ Dacă ponderile sunt în mulțimea  $\{1, \dots, |V|\}$  ce complexitate are algoritmul lui Kruskal?