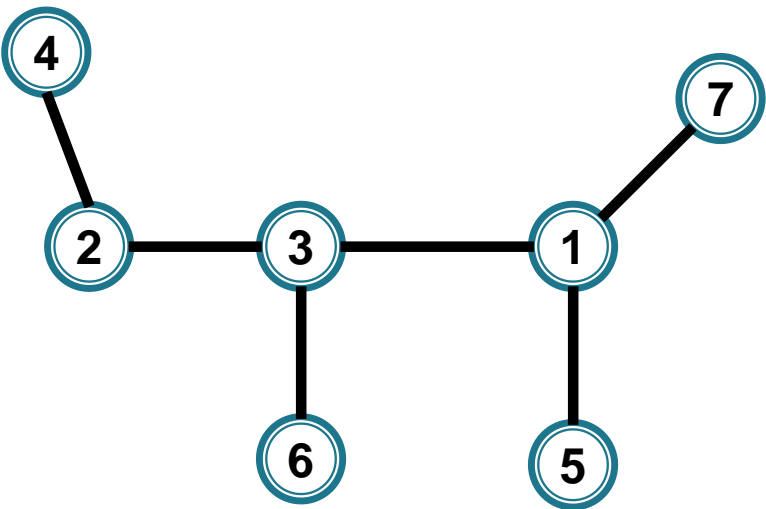
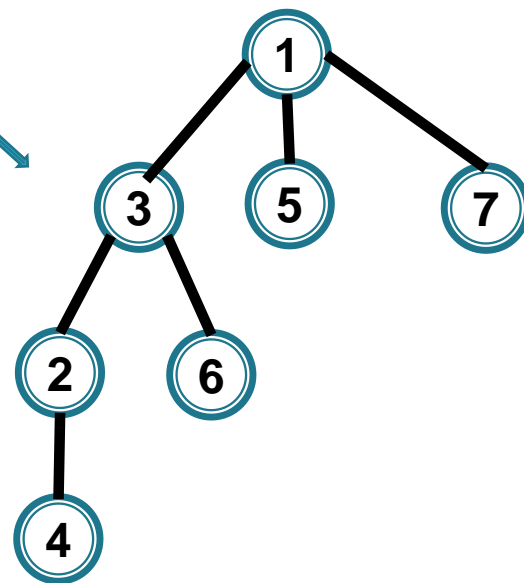
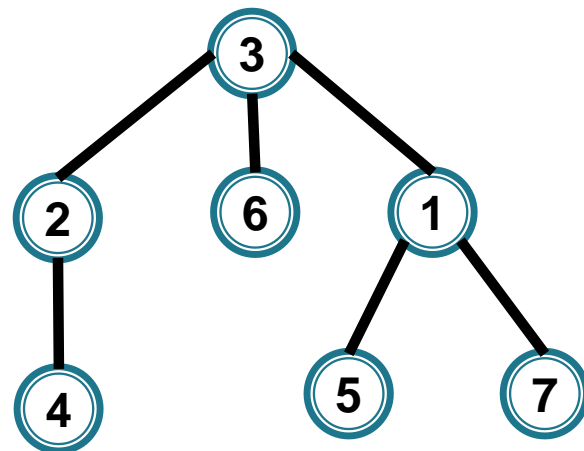
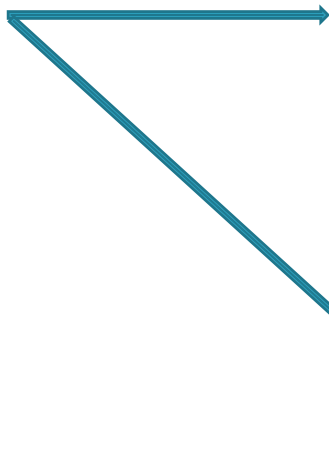


# Arbori



Arbore = conex + aciclic



Arbore cu rădăcină

# Arbori

- **Arbore** = graf neorientat **conex** și **aciclic**

# Arbori

➤ **Arbore** = graf neorientat **conex** și **aciclic**

➤ **Arbori filogenetici** – ilustrează evoluții

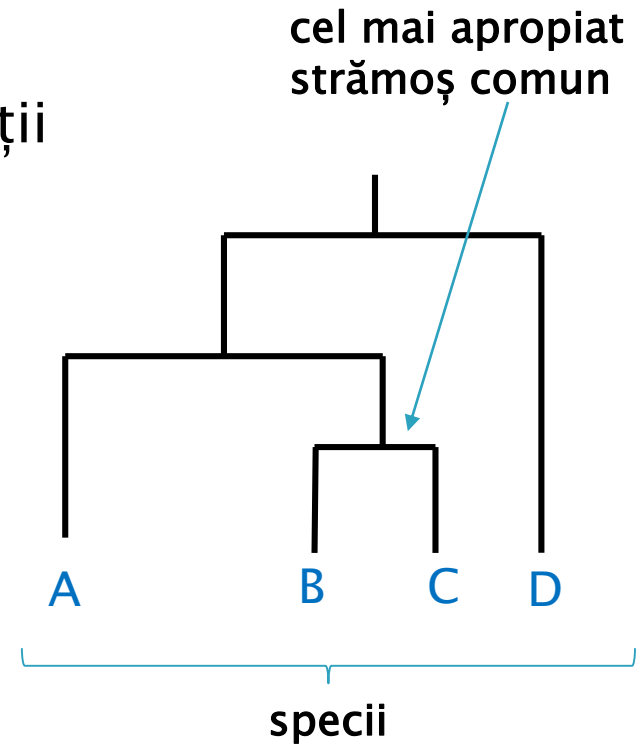
➤ **Arbori de dependențe, de joc**

➤ **Probleme de rutare**

➤ **Arbori aleatorii**

➤ **Arbori economici (cu costul minim)**

➤ **Structuri de date...**



# Arbori

## Leme

1. Orice arbore  $T$  cu  $n > 1$  are cel puțin două vârfuri terminale (de grad 1)

# Arbori

## Leme

1. Orice arbore  $T$  cu  $n > 1$  are cel puțin două vârfuri terminale (de grad 1)



Fie  $P$  un lanț elementar maxim în  $T$

# Arbori

## Leme

1. Orice arbore  $T$  cu  $n > 1$  are cel puțin două vârfuri terminale (de grad 1)



Fie  $P$  un lanț elementar maxim în  $T$

Extremitățile lui  $P$  sunt vârfuri terminale, altfel:

# Arbori

## Leme

1. Orice arbore  $T$  cu  $n > 1$  are cel puțin două vârfuri terminale (de grad 1)



Fie  $P$  un lanț elementar maxim în  $T$

Extremitățile lui  $P$  sunt vârfuri terminale, altfel:

– putem extinde lanțul cu o muchie 

sau



# Arbori

## Leme

1. Orice arbore  $T$  cu  $n > 1$  are cel puțin două vârfuri terminale (de grad 1)



Fie  $P$  un lanț elementar maxim în  $T$

Extremitățile lui  $P$  sunt vârfuri terminale, altfel:

– putem extinde lanțul cu o muchie



sau

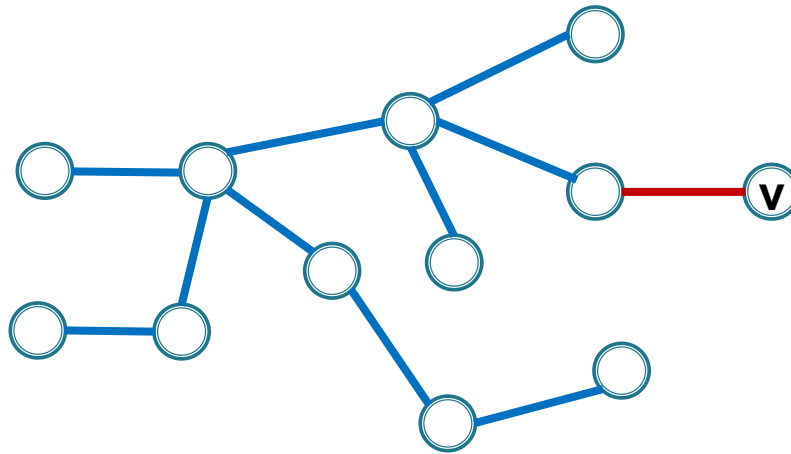
– se închide un ciclu în  $T$



# Arbori

## Leme

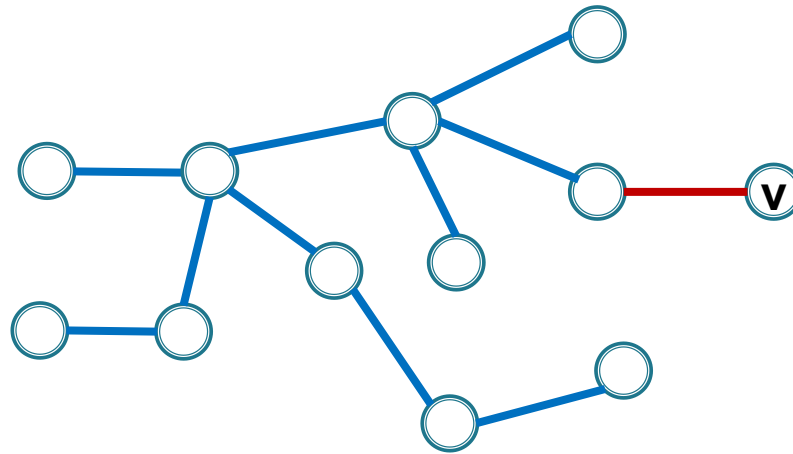
2. Fie  $T$  un arbore cu  $n > 1$  vârfuri și  $v$  un vârf terminal în  $T$ .  
Atunci  $T - v$  este arbore.



# Arbori

## Leme

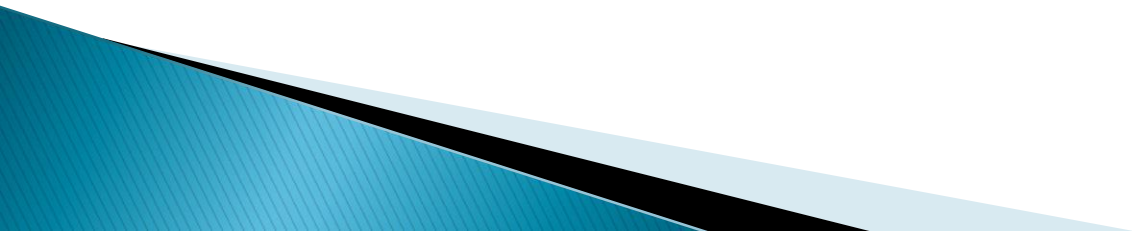
2. Fie  $T$  un arbore cu  $n > 1$  vârfuri și  $v$  un vârf terminal în  $T$ .  
Atunci  $T - v$  este arbore.



Rezultă din definiția conexității + un vârf terminal nu poate fi vârf intern al unui lanț elementar

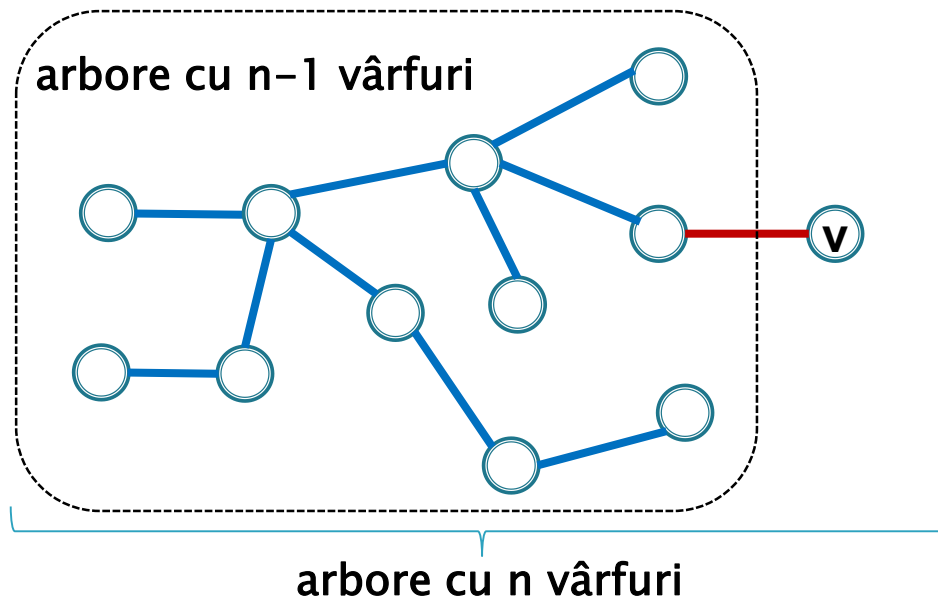
# Arbori

3. Un arbore cu  $n$  vârfuri are  $n-1$  muchii.



# Arbori

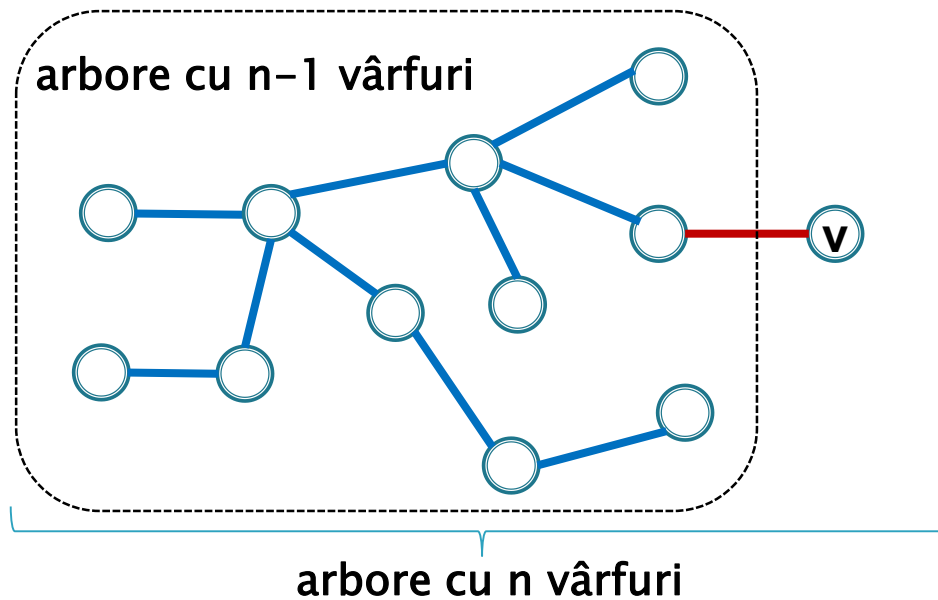
3. Un arbore cu  $n$  vârfuri are  $n-1$  muchii.



Inducție după  $n$

# Arbori

3. Un arbore cu  $n$  vârfuri are  $n-1$  muchii.

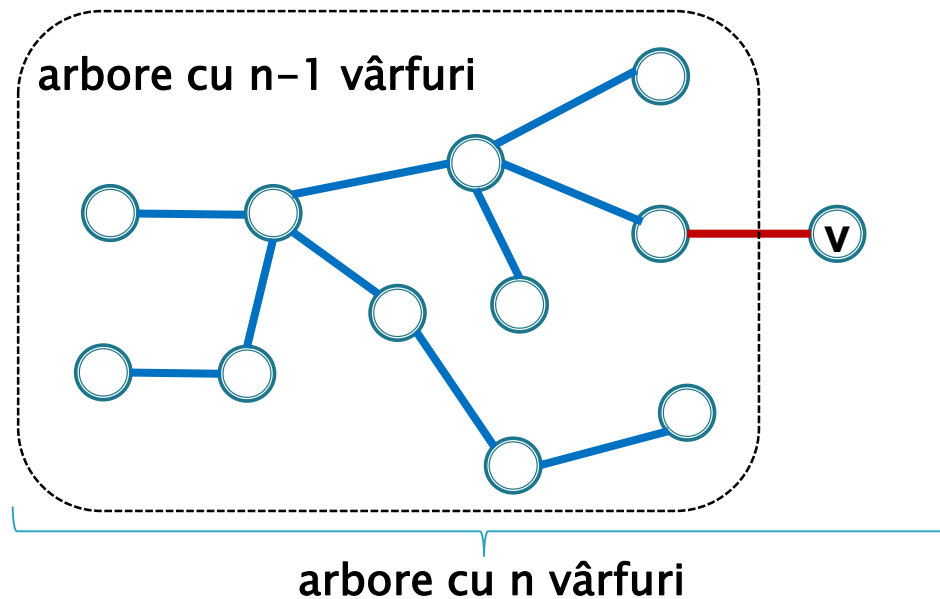


Inducție după  $n$

Verificare -  $n=1$

# Arbori

3. Un arbore cu  $n$  vârfuri are  $n-1$  muchii.

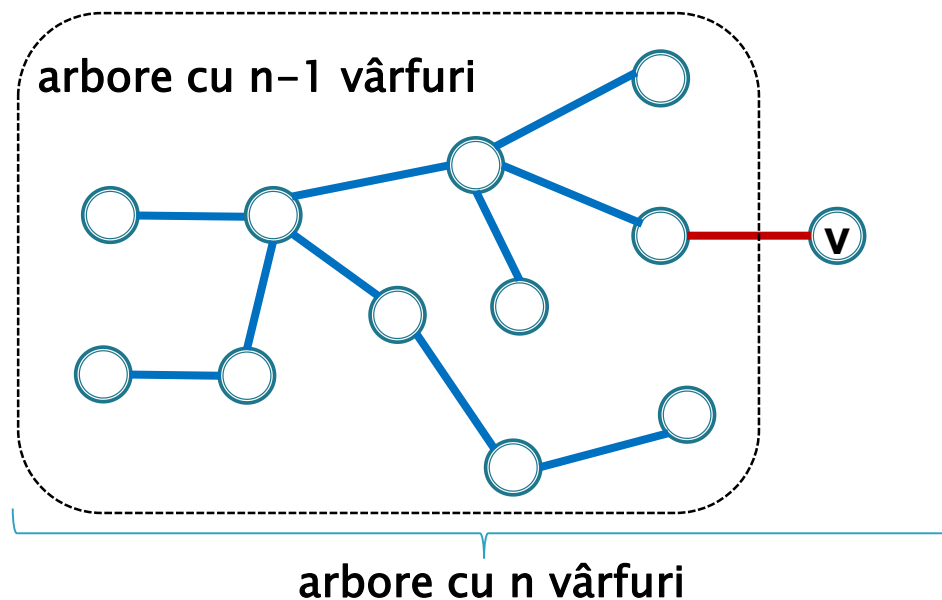


**Inducție după  $n$**

- Fie  $T$  este un arbore cu  $n$  vârfuri. Fie  $v$  este vârf terminal în  $T$  (!există, lema 1)

# Arbori

3. Un arbore cu  $n$  vârfuri are  $n-1$  muchii.



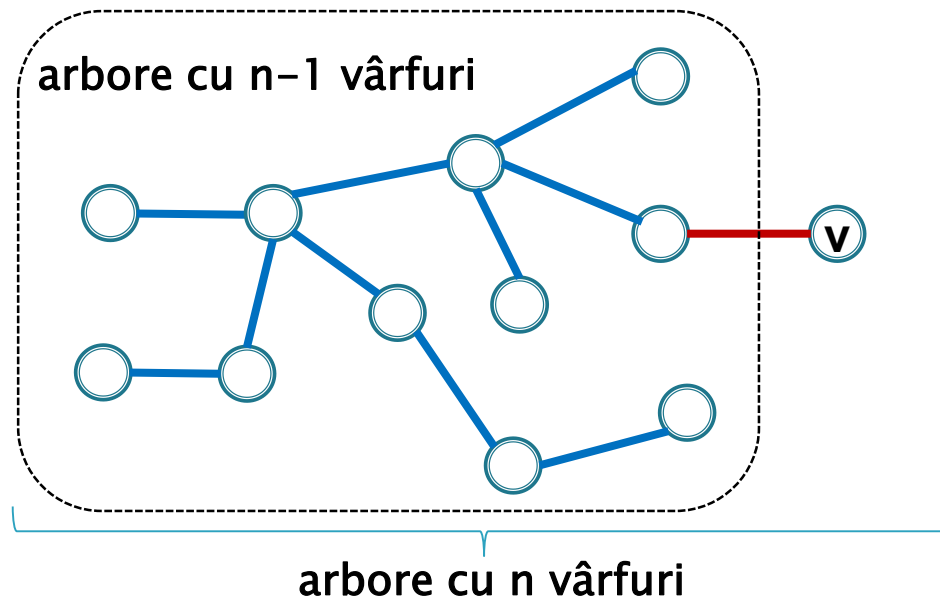
**Inducție după  $n$**

- Fie  $T$  este un arbore cu  $n$  vârfuri. Fie  $v$  este vârf terminal în  $T$  (!există, lema 1); atunci  $T - v$  este arbore cu  $n-1$  vârfuri (Lema 2) și



# Arbori

## 3. Un arbore cu $n$ vârfuri are $n-1$ muchii.



### Inducție după $n$

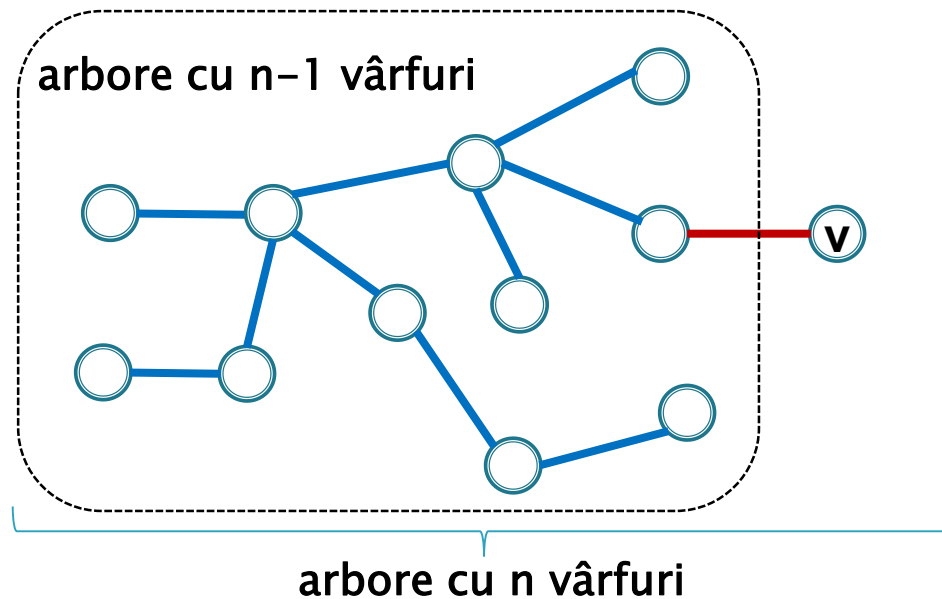
- Fie  $T$  este un arbore cu  $n$  vârfuri. Fie  $v$  este vârf terminal în  $T$  (!există, lema 1); atunci  $T - v$  este arbore cu  $n-1$  vârfuri (Lema 2)

$$|E(T-v)| = |E(T)| - 1$$

- Aplicăm ipoteza de inducție pentru  $T-v$

# Arbori

3. Un arbore cu  $n$  vârfuri are  $n-1$  muchii.



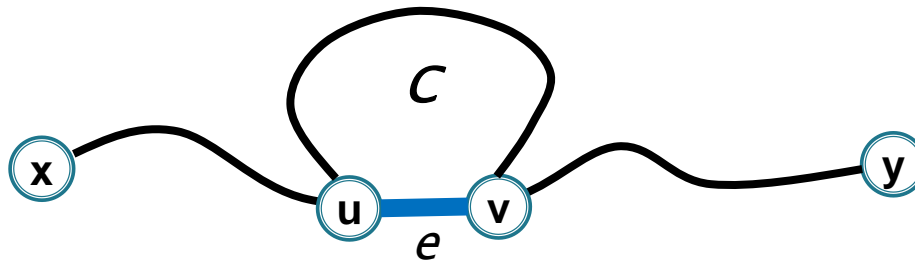
$$\Rightarrow |E(T-v)| = |V(T-v)| - 1 = (n-1) - 1 = n-2$$

$$\text{Rezultă } |E(T)| = |E(T-v)| + 1 = n-1$$

# Arbori

## Leme

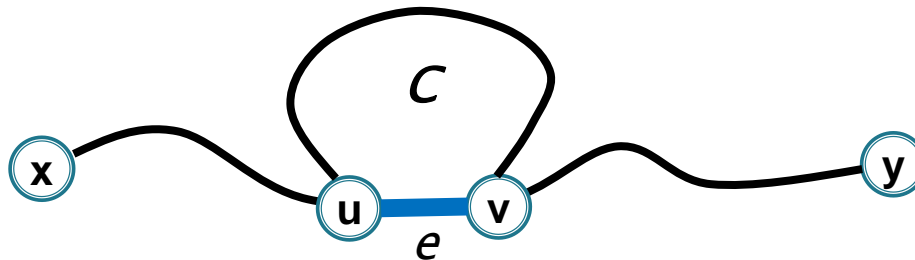
4. Fie  $G$  un graf neorientat conex și  $C$  un ciclu în  $G$ .  
Fie  $e \in E(C)$  o muchie din ciclul  $C$ .  
Atunci  $G - e$  este tot un graf conex.



# Arbori

## Leme

4. Fie  $G$  un graf neorientat conex și  $C$  un ciclu în  $G$ .  
Fie  $e \in E(C)$  o muchie din ciclul  $C$ .  
Atunci  $G - e$  este tot un graf conex.

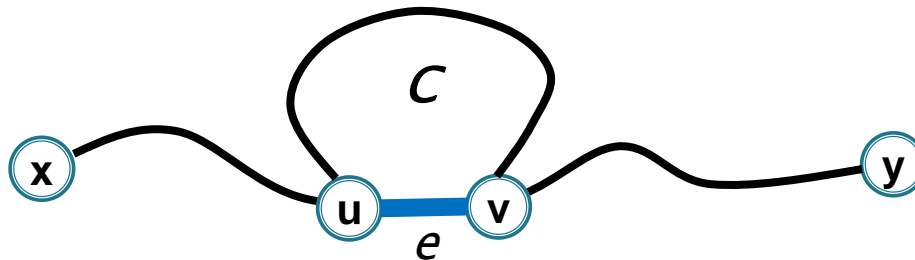


Rezultă din definiția conexității + observația:

# Arbori

## Leme

4. Fie  $G$  un graf neorientat conex și  $C$  un ciclu în  $G$ .  
Fie  $e \in E(C)$  o muchie din ciclul  $C$ .  
Atunci  $G-e$  este tot un graf conex.



Rezultă din definiția conexității + observația:

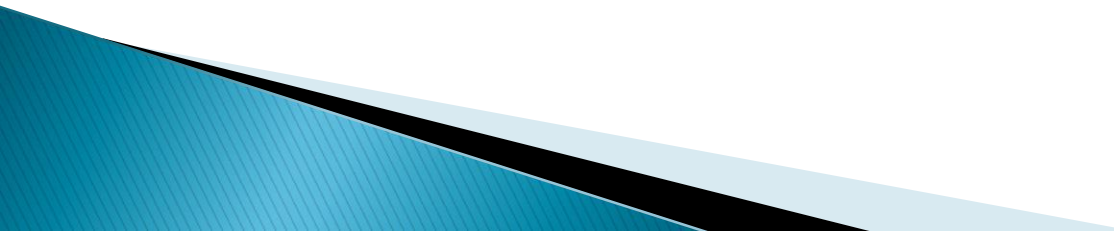
- dintr-un  $x$ - $y$  lanț în  $G$  care conține muchia  $e$  se poate obține un  $x$ - $y$  lanț în  $G-e$  înlocuind muchia  $e$  cu lanțul  $C-e$ .

# Arbori

## Definiții echivalente

Fie  $T$  un graf neorientat cu  $n > 1$  vârfuri.

Următoarele afirmații sunt echivalente.

1.  $T$  este arbore (conex și aciclic)
  - 2.
  - 3.
  - 4.
  - 5.
  - 6.
- 

# Arbori

## Definiții echivalente

Fie  $T$  un graf neorientat cu  $n > 1$  vârfuri.

Următoarele afirmații sunt echivalente.

1.  $T$  este arbore (conex și aciclic)
2.  $T$  este conex muchie-minimal
- 3.
- 4.
- 5.
- 6.

prin eliminarea unei muchii din  $T$  se obține un graf care nu mai este conex

# Arbori

## Definiții echivalente

Fie  $T$  un graf neorientat cu  $n > 1$  vârfuri.

Următoarele afirmații sunt echivalente.

1.  $T$  este arbore (conex și aciclic)
2.  $T$  este conex muchie-minimal
3.  $T$  este aciclic muchie-maximal
- 4.
- 5.
- 6.

prin eliminarea unei muchii din  $T$  se obține un graf care nu mai este conex

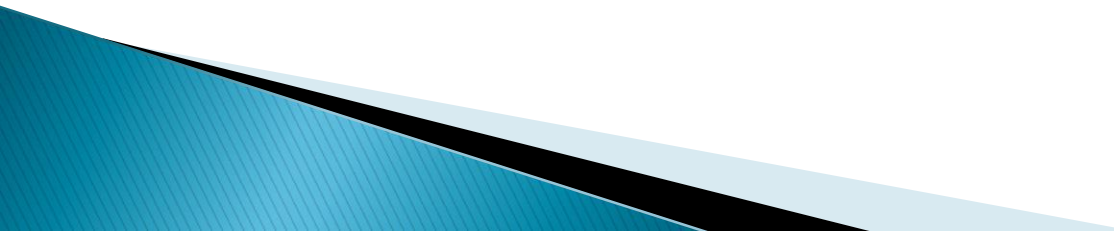


# Arbori

## Definiții echivalente

Fie  $T$  un graf neorientat cu  $n > 1$  vârfuri.

Următoarele afirmații sunt echivalente.

1.  $T$  este arbore (conex și aciclic)
  2.  $T$  este conex muchie-minimal
  3.  $T$  este aciclic muchie-maximal
  4.  $T$  este conex și are  $n-1$  muchii
  5.  $T$  este aciclic și are  $n-1$  muchii
  - 6.
- 

# Arbori

## Definiții echivalente

Fie  $T$  un graf neorientat cu  $n > 1$  vârfuri.

Următoarele afirmații sunt echivalente.

1.  $T$  este arbore (conex și aciclic)
2.  $T$  este conex muchie-minimal
3.  $T$  este aciclic muchie-maximal
4.  $T$  este conex și are  $n-1$  muchii
5.  $T$  este aciclic și are  $n-1$  muchii
6. Între oricare două vârfuri din  $T$  există un unic lanț elementar.

# Arbori

Demonstrații echivalențe – Temă (seminar)

Exemplu 1:  $T$  este arbore (conex și aciclic)  $\Leftrightarrow$

2:  $T$  este conex muchie-minimal

# Arbori

## Demonstrații echivalențe – Temă (seminar)

**Exemplu 1:**  $T$  este arbore (conex și aciclic)  $\Leftrightarrow$

**2:**  $T$  este conex muchie-minimal

$1 \Rightarrow 2$ : Presupunem  $T$  este arbore.

Fie  $e = xy \in E(T)$ .

Arătăm că  $T - e$  nu este conex (deci  $T$  este conex muchie-minimal).

# Arbori

## Demonstrații echivalențe – Temă (seminar)

**Exemplu 1:**  $T$  este arbore (conex și aciclic)  $\Leftrightarrow$

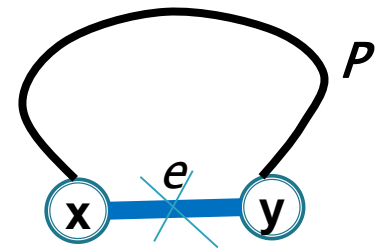
**2:**  $T$  este conex muchie-minimal

$1 \Rightarrow 2$ : Presupunem  $T$  este arbore.

Fie  $e = xy \in E(T)$ .

Arătăm că  $T - e$  nu este conex (deci  $T$  este conex muchie-minimal).

Presupunem prin absurd că  $T - e$  este conex.



# Arbori

## Demonstrații echivalențe – Temă (seminar)

**Exemplu 1:**  $T$  este arbore (conex și aciclic)  $\Leftrightarrow$

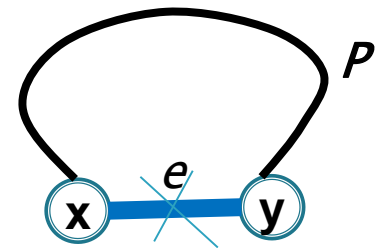
**2:**  $T$  este conex muchie-minimal

$1 \Rightarrow 2$ : Presupunem  $T$  este arbore.

Fie  $e = xy \in E(T)$ .

Arătăm că  $T - e$  nu este conex (deci  $T$  este conex muchie-minimal).

Presupunem prin absurd că  $T - e$  este conex. Atunci există un lanț elementar  $P$  în  $T - e$  de la  $x$  la  $y$  (care unește extremitățile lui  $e$ ).



# Arbori

## Demonstrații echivalențe – Temă (seminar)

**Exemplu 1:**  $T$  este arbore (conex și aciclic)  $\Leftrightarrow$

**2:**  $T$  este conex muchie-minimal

$1 \Rightarrow 2$ : Presupunem  $T$  este arbore.

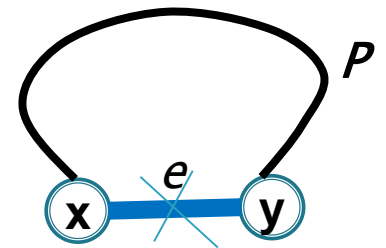
Fie  $e = xy \in E(T)$ .

Arătăm că  $T - e$  nu este conex (deci  $T$  este conex muchie-minimal).

Presupunem prin absurd că  $T - e$  este conex. Atunci există un lanț elementar  $P$  în  $T - e$  de la  $x$  la  $y$  (care unește extremitățile lui  $e$ ).

Atunci lanțul

$P + xy = [x \underline{P} y; x]$  este ciclu în  $T$ , contradicție.



# Arbori

Demonstrații echivalențe – Temă (seminar)

**Exemplu 1:**  $T$  este arbore (conex și aciclic)  $\Leftrightarrow$

**2:**  $T$  este conex muchie-minimal

$2 \Rightarrow 1$ : Presupunem că  $T$  este conex muchie-minimal.

Demonstrăm că  $T$  este aciclic.



# Arbori

## Demonstrații echivalențe – Temă (seminar)

**Exemplu 1:**  $T$  este arbore (conex și aciclic)  $\Leftrightarrow$

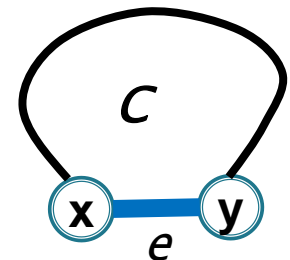
**2:**  $T$  este conex muchie-minimal

$2 \Rightarrow 1$ : Presupunem că  $T$  este conex muchie-minimal.

Demonstrăm că  $T$  este aciclic.

Presupunem prin absurd că  $T$  conține cicluri.

Fie  $C$  un ciclu elementar în  $T$ .



# Arbori

## Demonstrații echivalențe – Temă (seminar)

**Exemplu 1:**  $T$  este arbore (conex și aciclic)  $\Leftrightarrow$

**2:**  $T$  este conex muchie-minimal

$2 \Rightarrow 1$ : Presupunem că  $T$  este conex muchie-minimal.

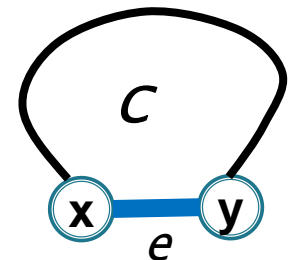
Demonstrăm că  $T$  este aciclic.

Presupunem prin absurd că  $T$  conține cicluri.

Fie  $C$  un ciclu elementar în  $T$ .

Fie  $e \in E(C)$ .

Din Lema 4 rezultă că  $T - e$  este tot conex,  
contradicție ( $T$  este conex muchie-minimal).

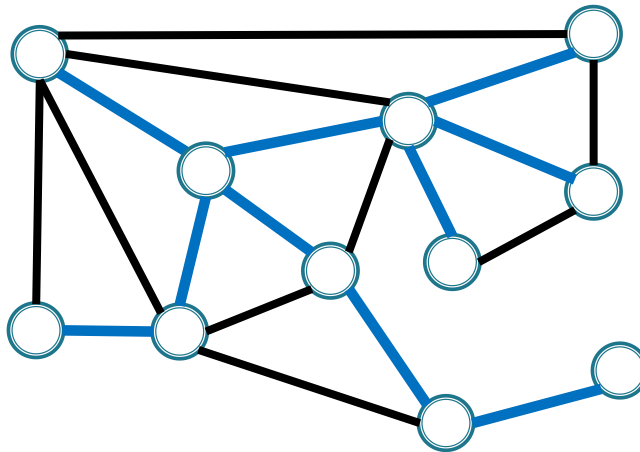


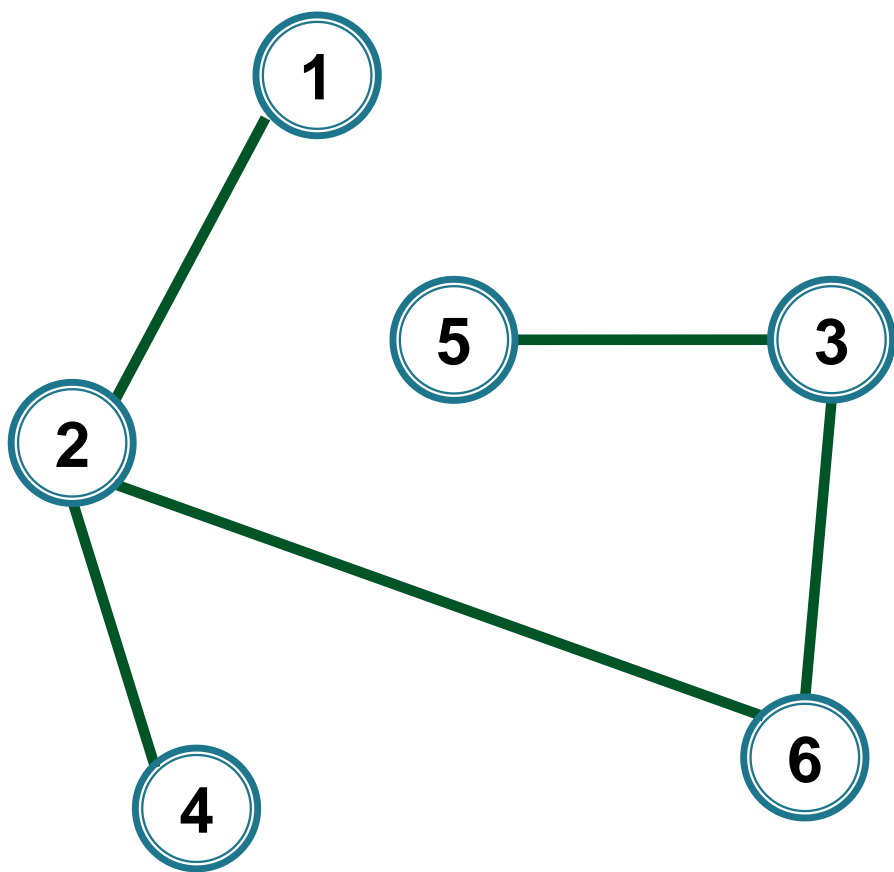
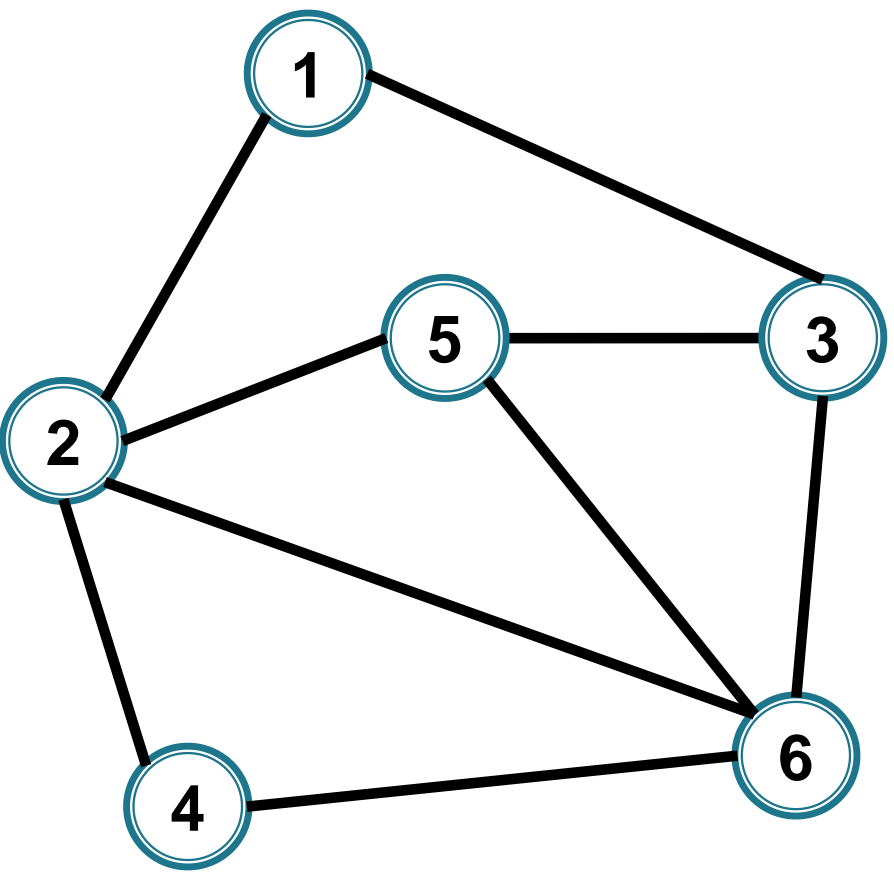
# Arbori parțiali ai unui graf

# Arbori parțiali

## Proprietate

Orice graf neorientat conex conține un arbore parțial  
(un graf parțial care este arbore).





# Arbori parțiali

## Proprietate

Orice graf neorientat conex conține un arbore parțial

**Demonstrație** – două tipuri de algoritmi de construcție a unui arbore parțial al unui graf conex  $G=(V,E)$ :

Prin adăugare de muchii ( <b>bottom – up</b> )	Prin eliminare de muchii ( <b>cut –down</b> )

# Arbori parțiali

## Proprietate

Orice graf neorientat conex conține un arbore parțial

**Demonstrație** – două tipuri de algoritmi de construcție a unui arbore parțial al unui graf conex  $G=(V,E)$ :

Prin adăugare de muchii ( <b>bottom – up</b> )	Prin eliminare de muchii ( <b>cut –down</b> )
$T \leftarrow (V, \emptyset)$	

# Arbori parțiali

## Proprietate

Orice graf neorientat conex conține un arbore parțial

**Demonstrație** – două tipuri de algoritmi de construcție a unui arbore parțial al unui graf conex  $G=(V,E)$ :

Prin adăugare de muchii ( <b>bottom – up</b> )	Prin eliminare de muchii ( <b>cut –down</b> )
$T \leftarrow (V, \emptyset)$ cat timp $T$ nu este conex executa <ul style="list-style-type: none"><li>alege <math>e \in E(G) - E(T)</math> care <b>unește două componente conexe</b> din <math>T</math> (nu formează cicluri cu muchiile din <math>T</math>)</li><li><math>E(T) \leftarrow E(T) \cup \{e\}</math></li></ul> returneaza $T$	



# Arbori parțiali

## Proprietate

Orice graf neorientat conex conține un arbore parțial

**Demonstrație** – două tipuri de algoritmi de construcție a unui arbore parțial al unui graf conex  $G=(V,E)$ :

Prin adăugare de muchii (bottom – up)	Prin eliminare de muchii (cut –down)
$T \leftarrow (V, \emptyset)$ cat timp $T$ nu este conex executa <ul style="list-style-type: none"><li>alege <math>e \in E(G) - E(T)</math> care unește două componente conexe din <math>T</math> (nu formează cicluri cu muchiile din <math>T</math>)</li><li><math>E(T) \leftarrow E(T) \cup \{e\}</math></li></ul> returneaza $T$	
În final $T$ este conex și aciclic, deci arbore	

# Arbori parțiali

## Proprietate

Orice graf neorientat conex conține un arbore parțial

**Demonstrație** – două tipuri de algoritmi de construcție a unui arbore parțial al unui graf conex  $G=(V,E)$ :

Prin adăugare de muchii ( <b>bottom – up</b> )	Prin eliminare de muchii ( <b>cut –down</b> )
$T \leftarrow (V, \emptyset)$ cat timp $T$ <b>nu este conex</b> executa <ul style="list-style-type: none"><li>alege <math>e \in E(G) - E(T)</math> care <b>unește două componente conexe</b> din <math>T</math> (nu formează cicluri cu muchiile din <math>T</math>)</li><li><math>E(T) \leftarrow E(T) \cup \{e\}</math></li></ul> returneaza $T$	$T \leftarrow (V, E)$ cat timp $T$ <b>conține cicluri</b> executa <ul style="list-style-type: none"><li>alege <math>e \in E(T)</math> o muchie <b>dintr-un ciclu</b></li><li><math>E(T) \leftarrow E(T) - \{e\}</math></li></ul> returneaza $T$
În final $T$ este conex și aciclic, deci arbore	

# Arbori parțiali

## Proprietate

Orice graf neorientat conex conține un arbore parțial

**Demonstrație** – două tipuri de algoritmi de construcție a unui arbore parțial al unui graf conex  $G=(V,E)$ :

Prin adăugare de muchii ( <b>bottom – up</b> )	Prin eliminare de muchii ( <b>cut –down</b> )
$T \leftarrow (V, \emptyset)$ cat timp $T$ <b>nu este conex</b> executa <ul style="list-style-type: none"><li>alege <math>e \in E(G) - E(T)</math> care <b>unește două componente conexe</b> din <math>T</math> (nu formează cicluri cu muchiile din <math>T</math>)</li><li><math>E(T) \leftarrow E(T) \cup \{e\}</math></li></ul> returneaza $T$	$T \leftarrow (V, E)$ cat timp $T$ <b>conține cicluri</b> executa <ul style="list-style-type: none"><li>alege <math>e \in E(T)</math> o muchie <b>dintr-un ciclu</b></li><li><math>E(T) \leftarrow E(T) - \{e\}</math></li></ul> returneaza $T$
În final $T$ este conex și aciclic, deci arbore	În final $T$ este aciclic și conex (s-au eliminat doar muchii din ciclu), deci arbore

# Arbori parțiali

Algoritmi de determinare a unui arbore parțial al unui graf conex



Algoritm de determinare a unui arbore parțial?

Complexitate algoritm?

# Arbori parțiali

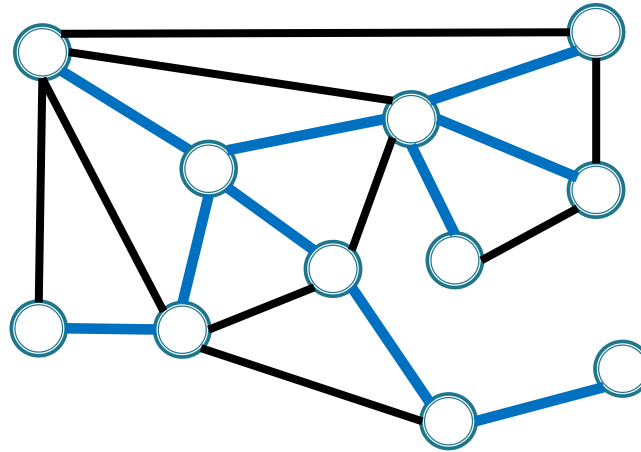
Algoritmi de determinare a unui arbore parțial al unui graf conex

Complexitate algoritm?



arborele asociat unei parcurgeri este arbore parțial  $\Rightarrow$   
determinăm un arbore parțial printr-o parcurgere

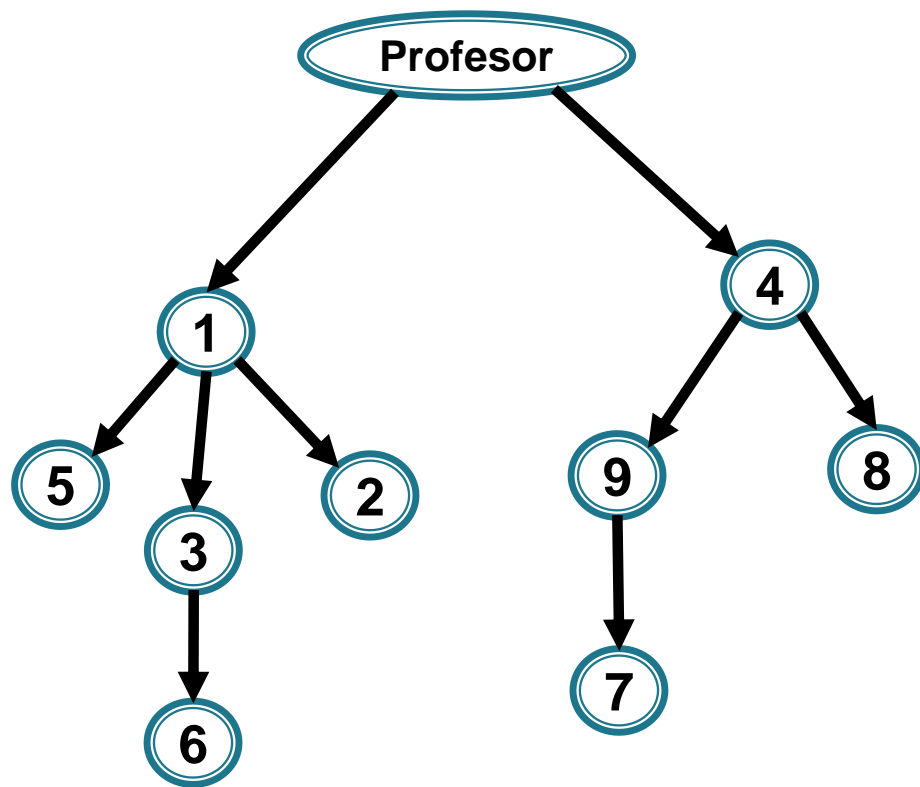
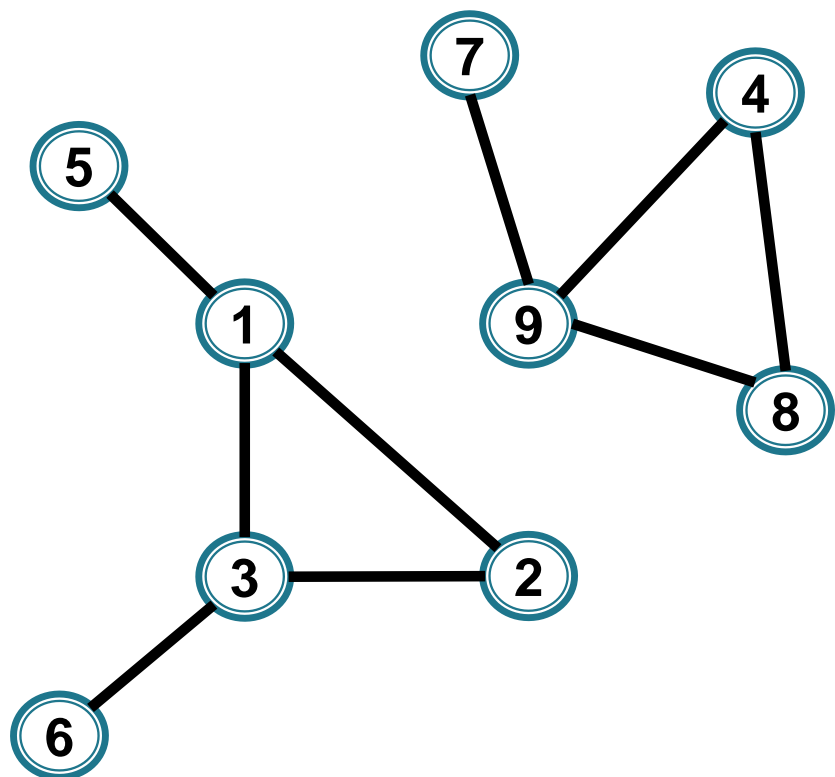
# Arbori parțiali



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

# Aplicații

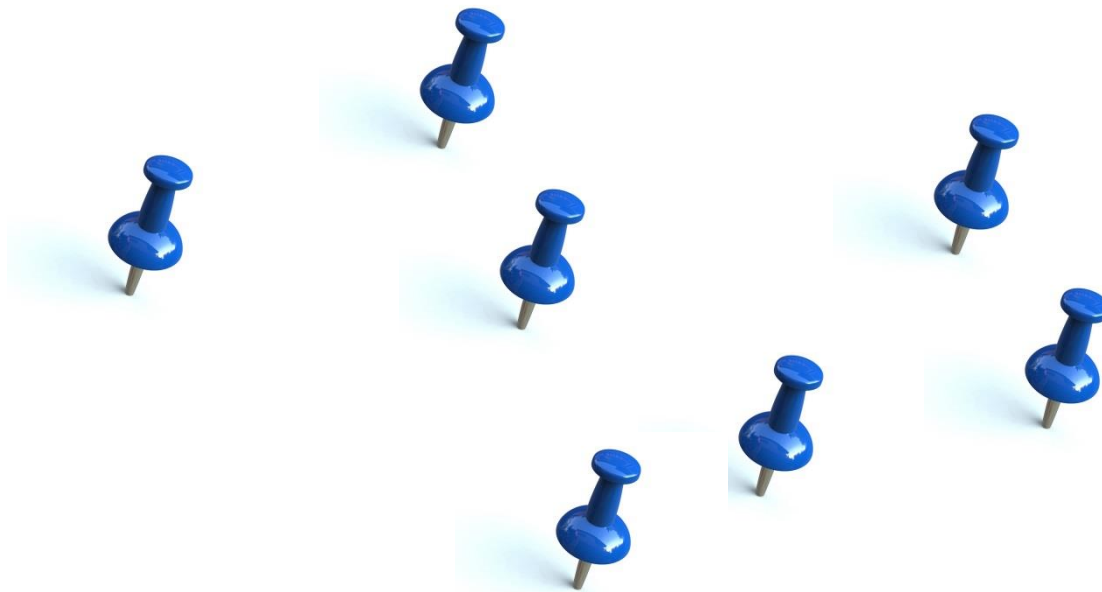
- ▶ Determinarea unui arbore parțial al unui graf conex
- ▶ Transmiterea unui mesaj în rețea: Între participanții la un curs s-au legat relații de prietenie și comunică și în afara cursului. Profesorul vrea să transmită un mesaj participanților și știe ce relații de prietenie s-au stabilit între ei. El vrea să contacteze cât mai puțini participanți, urmând ca aceștia să transmită mesajul între ei. Ajutați-l pe profesor să decidă cui trebuie să transmită inițial mesajul și să atașeze la mesaj o listă în care să arate fiecărui participant către ce prieteni trebuie să trimită mai departe mesajul, astfel încât mesajul să ajungă la fiecare participant la curs o singură dată.



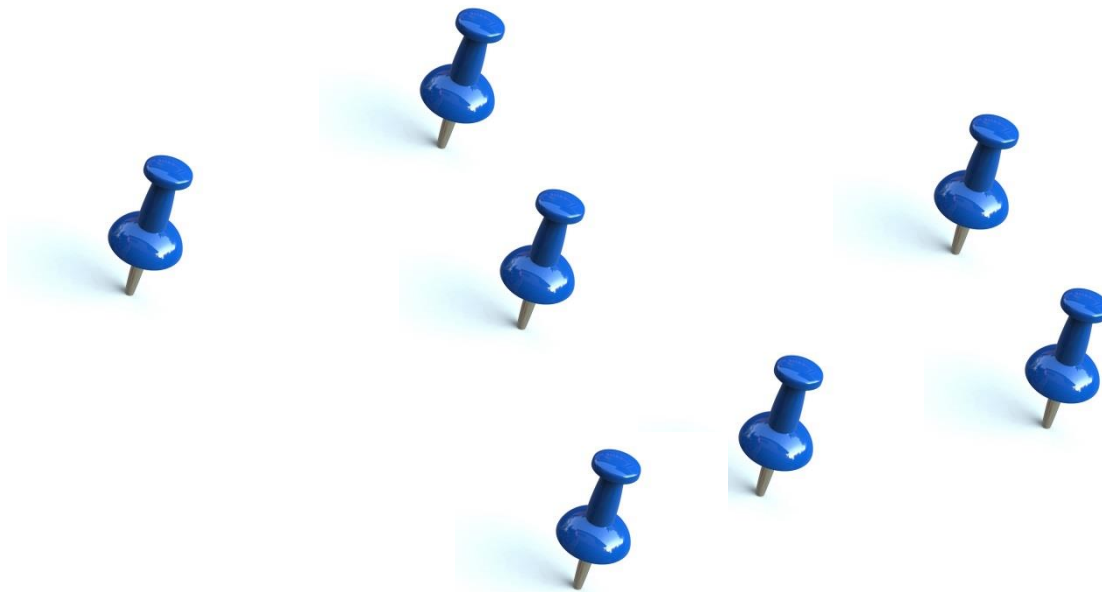


# Arbori parțiali de cost minim

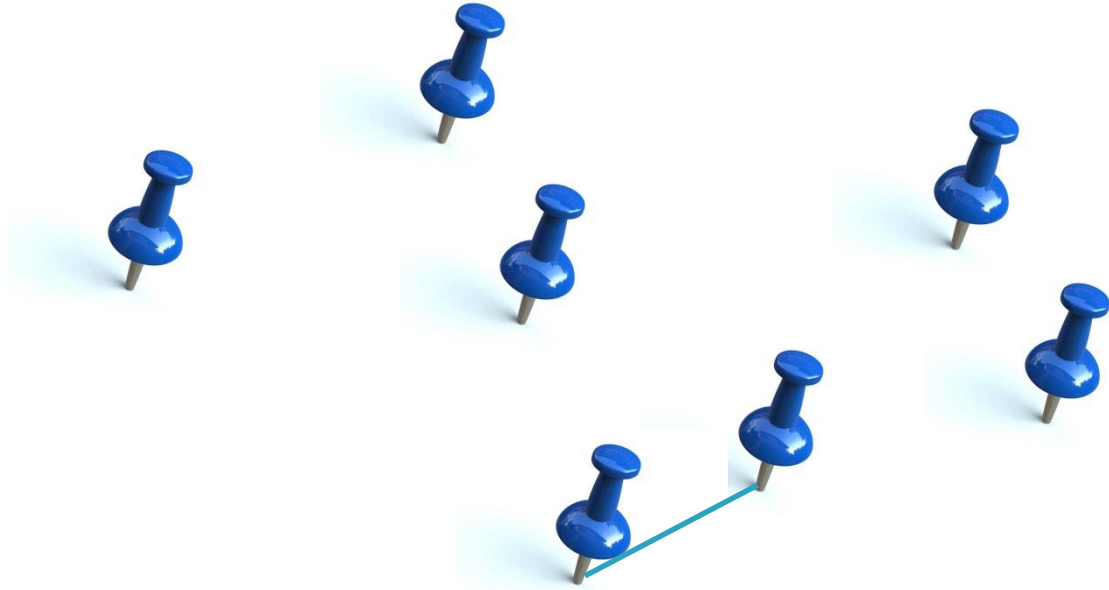


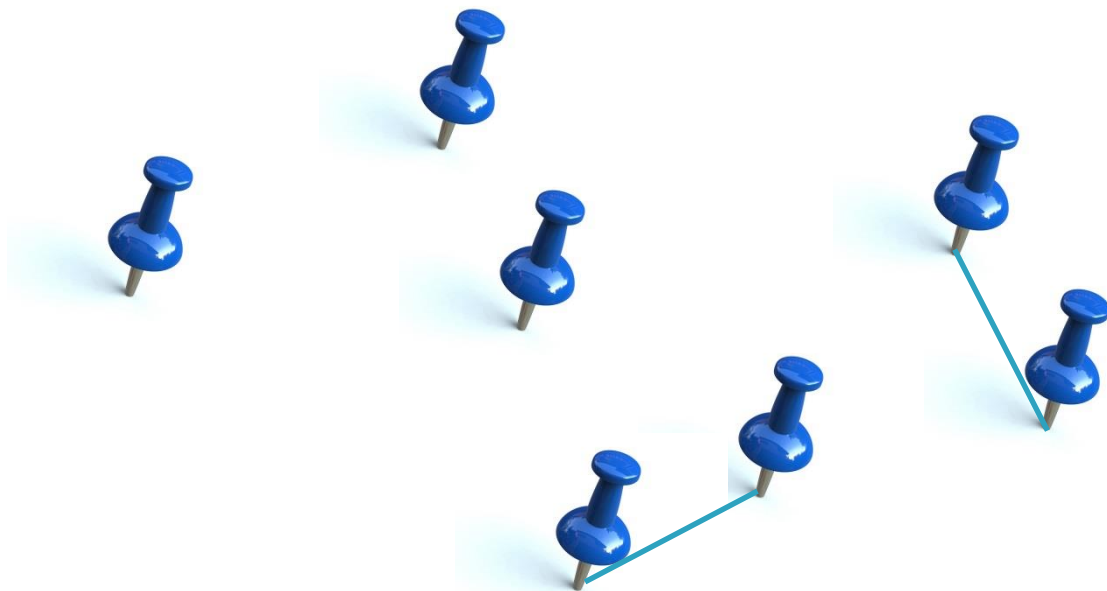


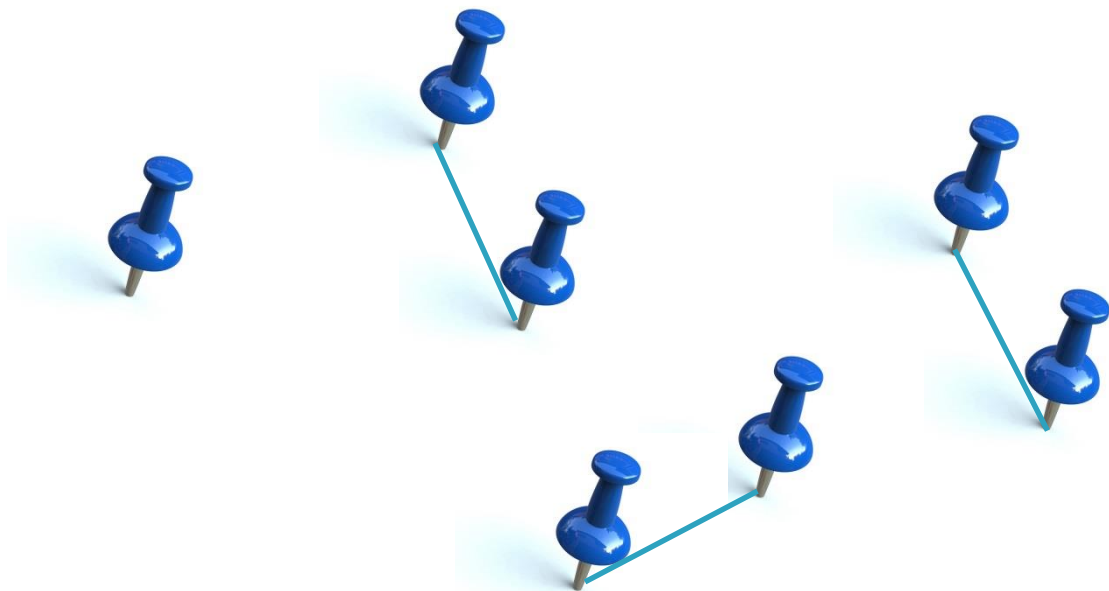
**Conectați pinii astfel încât să folosiți cât mai puțin cablu**

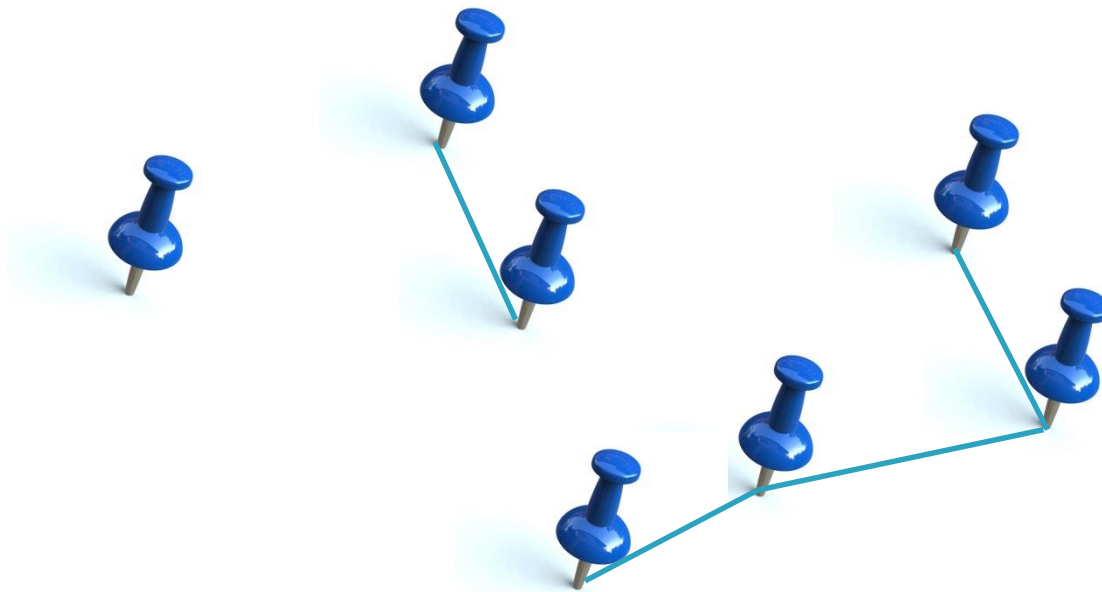


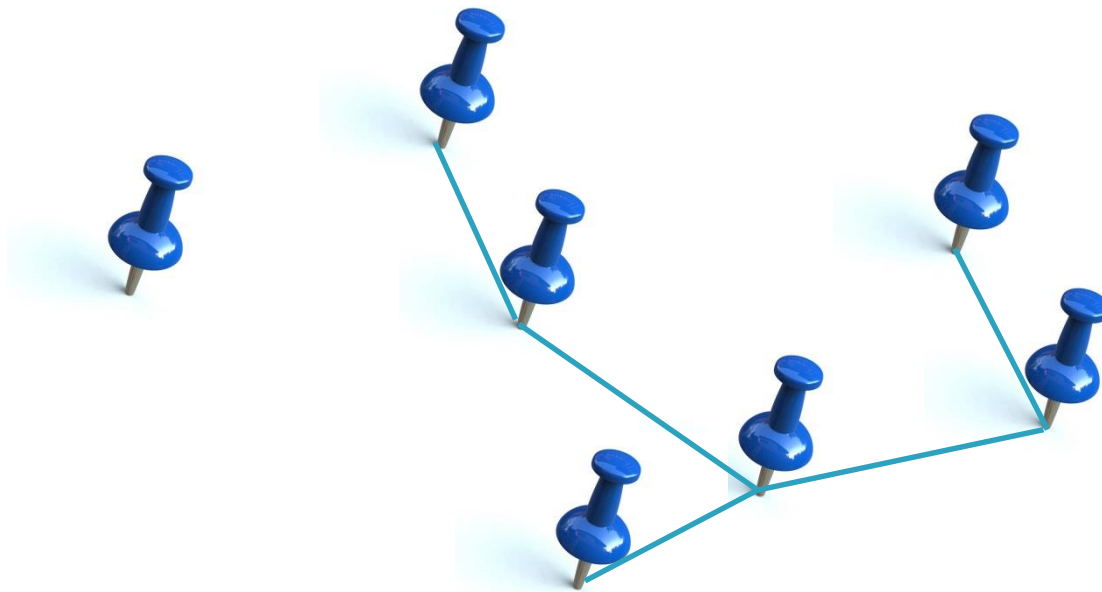
- Legăm pini apropiati
- Nu închidem cicluri



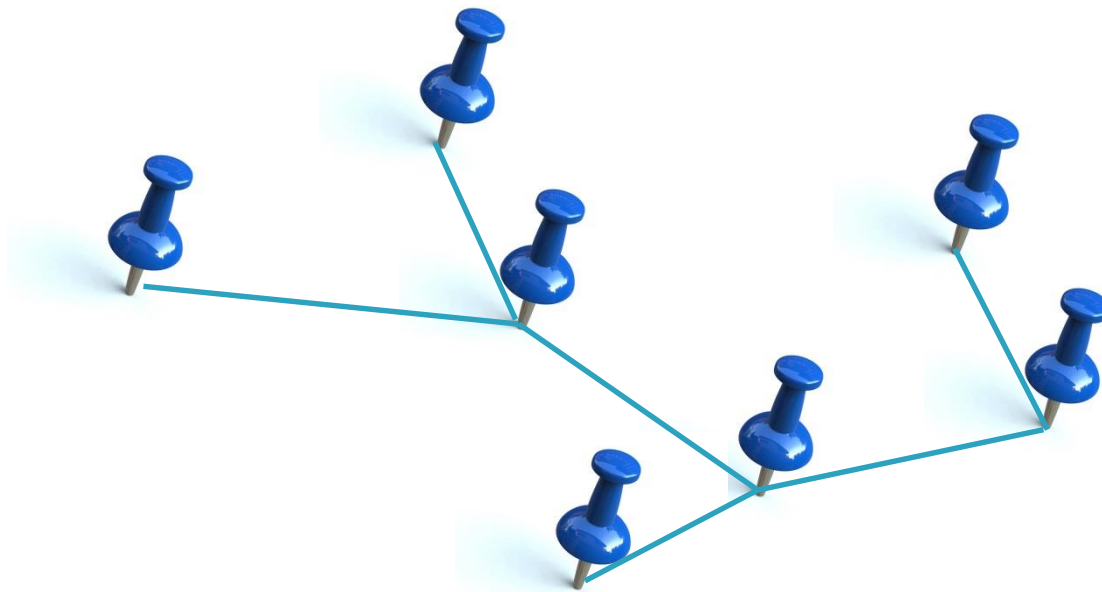














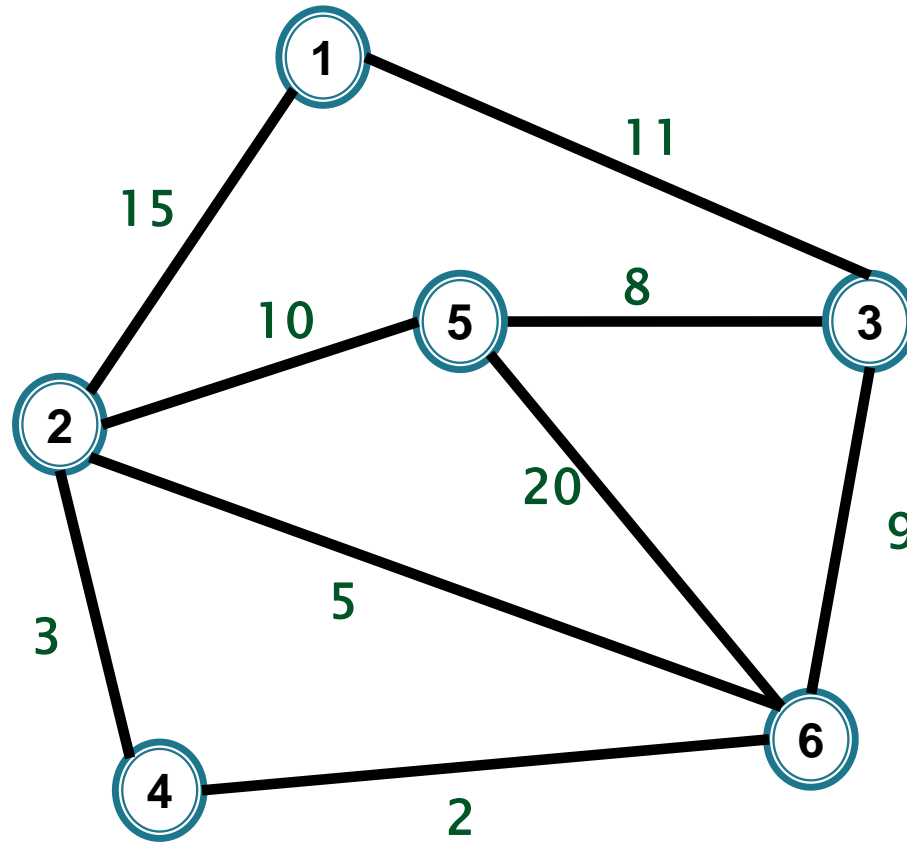
conectare cu cost minim  $\Rightarrow$  evităm ciclurile

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  $A \subseteq E$

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

# 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  $G$

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

# Grafuri ponderate

## Reprezentarea grafurilor ponderate

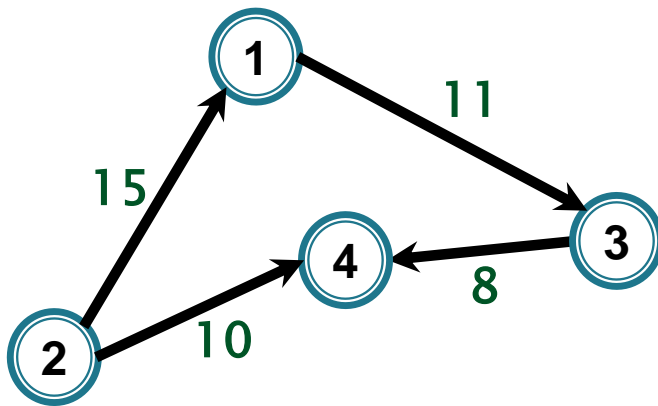


# Grafuri ponderate

## Reprezentarea grafurilor ponderate

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

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



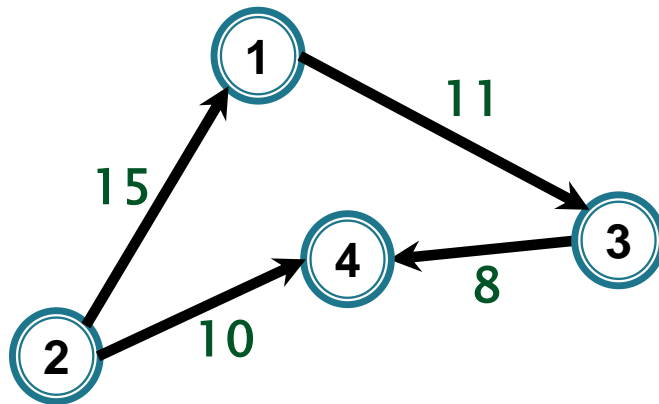
0	$\infty$	11	$\infty$
15	0	$\infty$	10
$\infty$	$\infty$	0	8
$\infty$	$\infty$	$\infty$	0



# Grafuri ponderate

## Reprezentarea grafurilor ponderate

- ▶ Matrice de costuri (ponderi)
- ▶ Liste de adiacență



1: 3 / 11

2: 1 / 15, 4 / 10

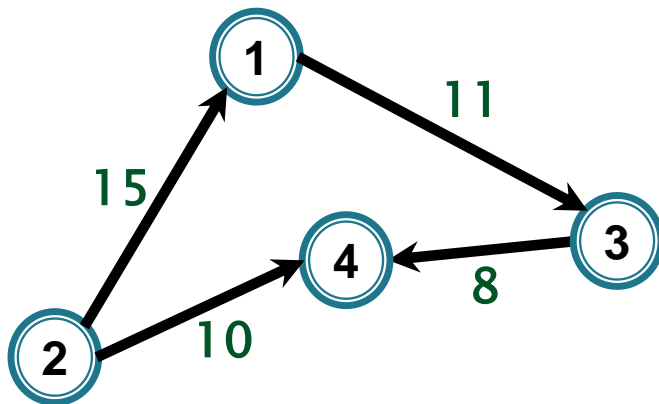
3: 4 / 8

4:

# Grafuri ponderate

## Reprezentarea grafurilor ponderate

- ▶ Matrice de costuri (ponderi)
- ▶ Liste de adiacență
- ▶ Liste de muchii/arce



1 3 11

2 1 15

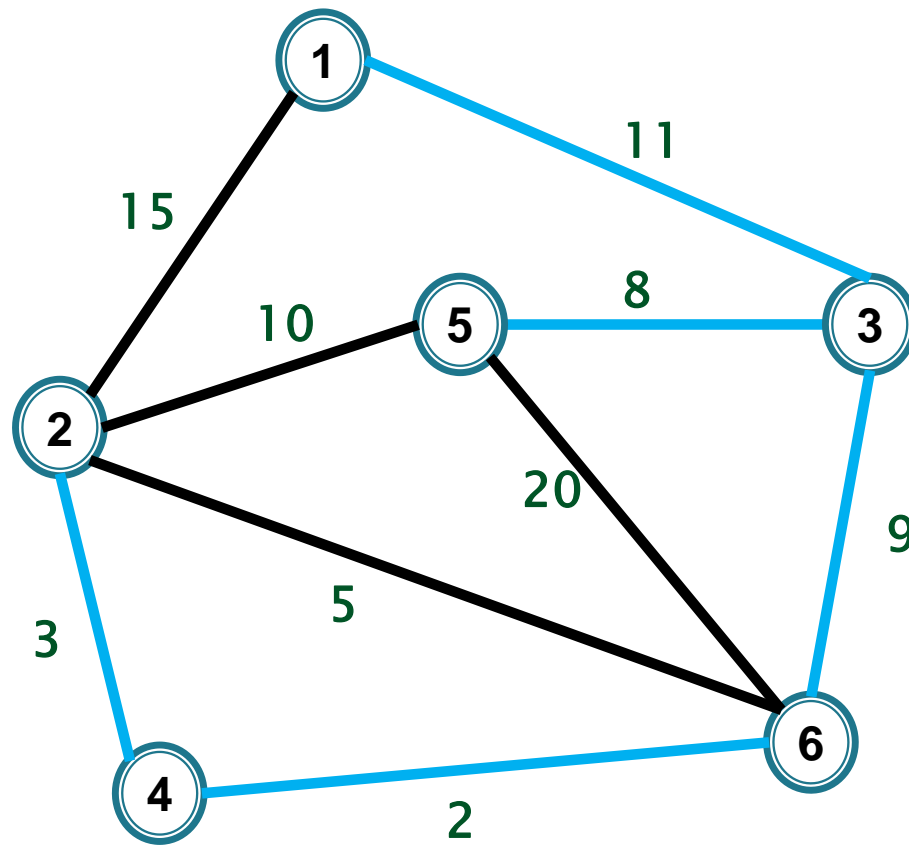
2 4 10

3 4 8

# A.p.c.m

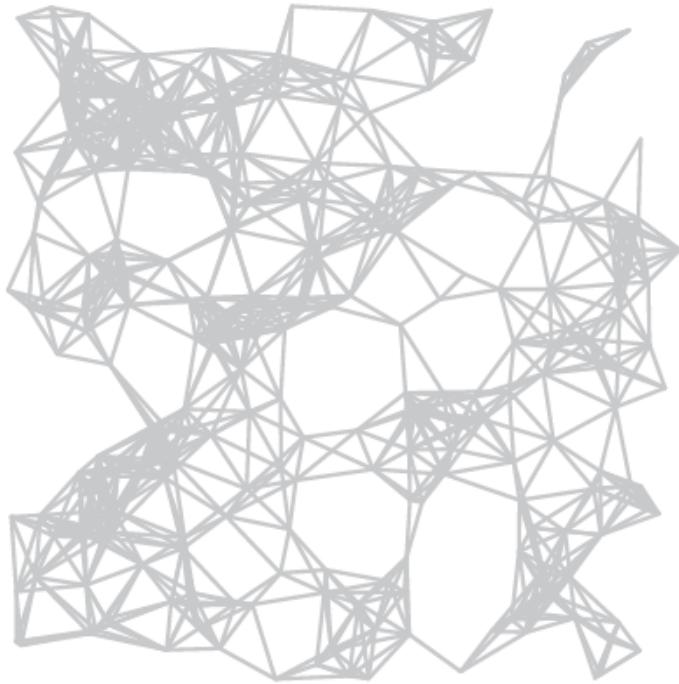
- ▶  $G = (V, E, w)$  **conex ponderat**
- ▶ **Arbore parțial de cost minim** al lui  $G$  = un arbore parțial  $T_{\min}$  al lui  $G$  cu

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

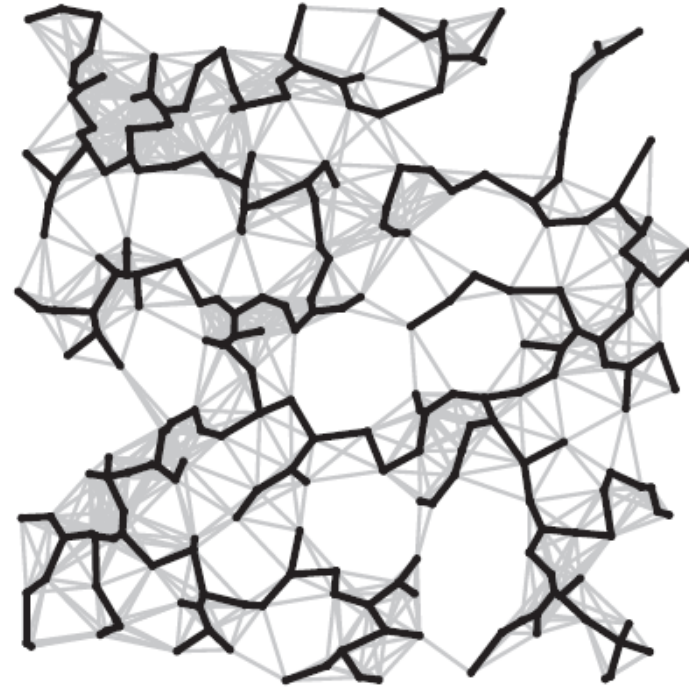


# A.p.c.m.

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)**
- ▶ ...

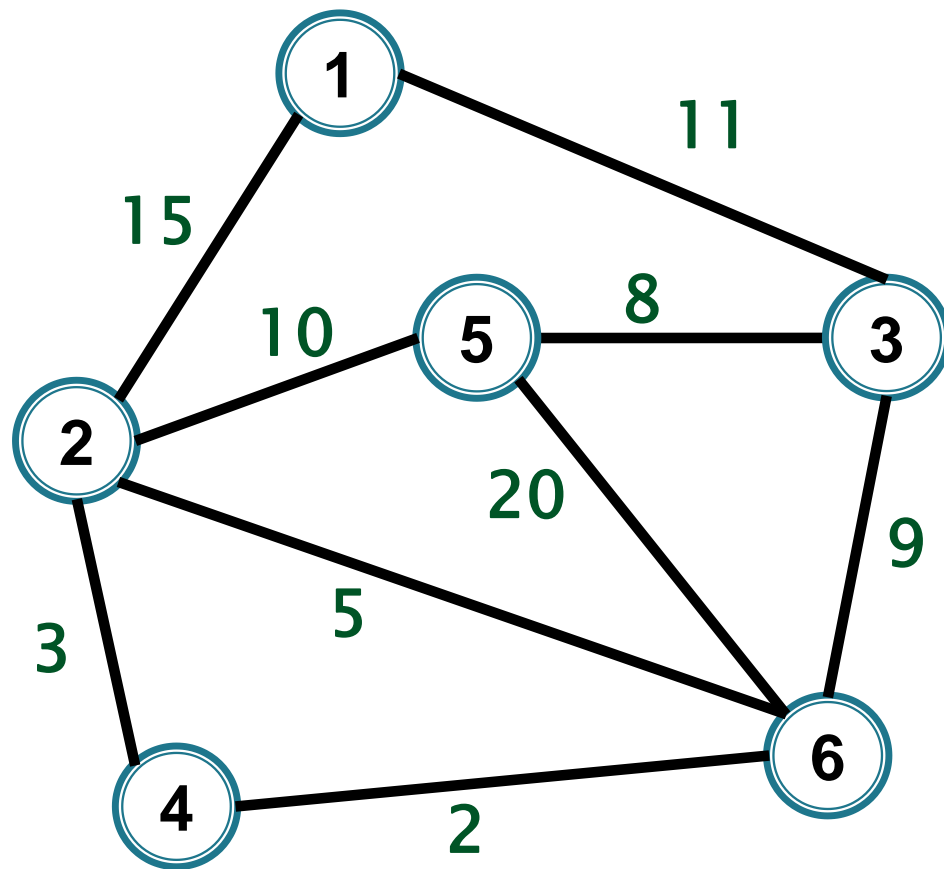
# Algoritmi de determinare a unui arbore parțial de cost minim

# 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

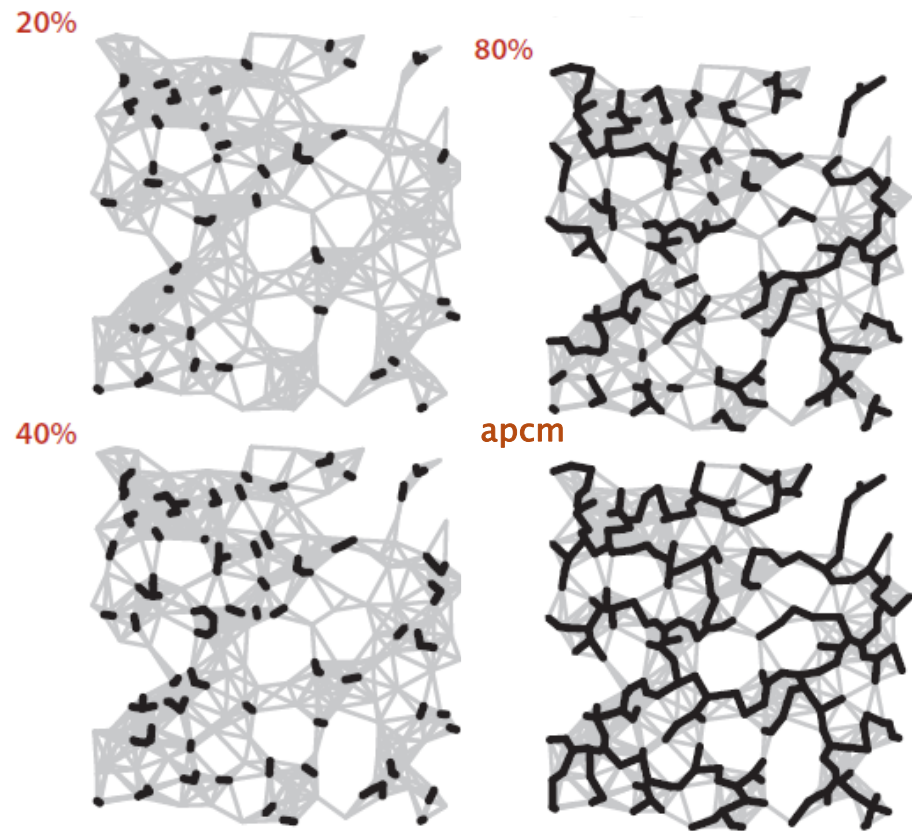


După ce criteriu selectăm muchiile?

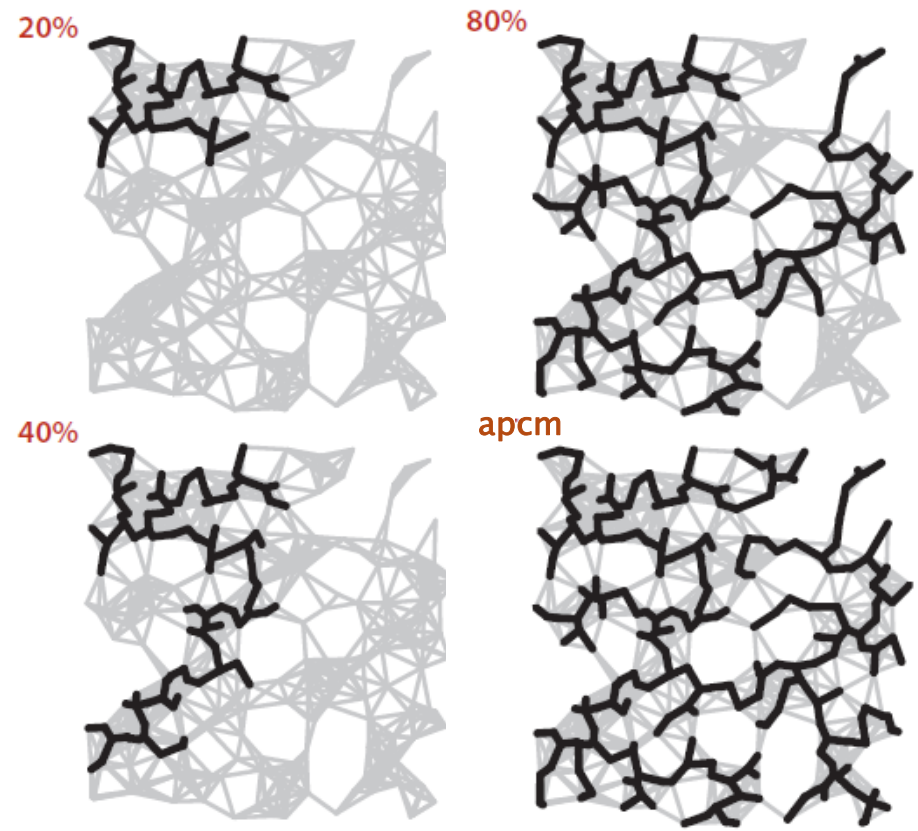
⇒ diverși algoritmi

# Arbori parțiali de cost minim

## Kruskal



## Prim



Imagine din

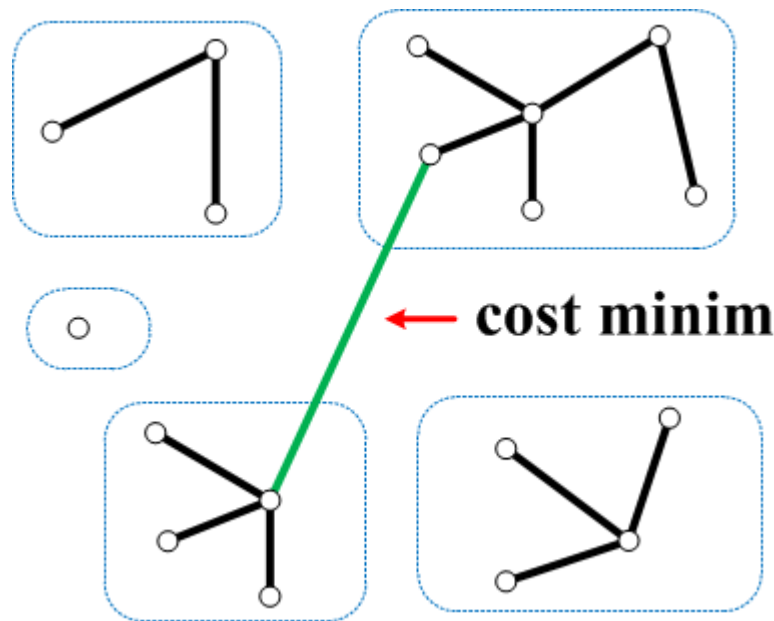
R. Sedgewick, K. Wayne – Algorithms, 4th edition, Pearson Education, 2011

# 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

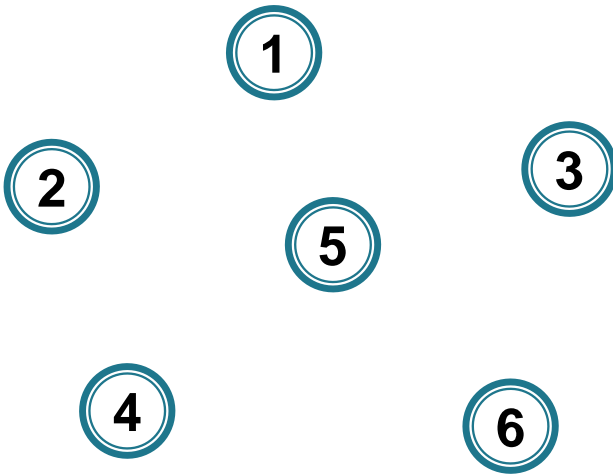
### Kruskal

- 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

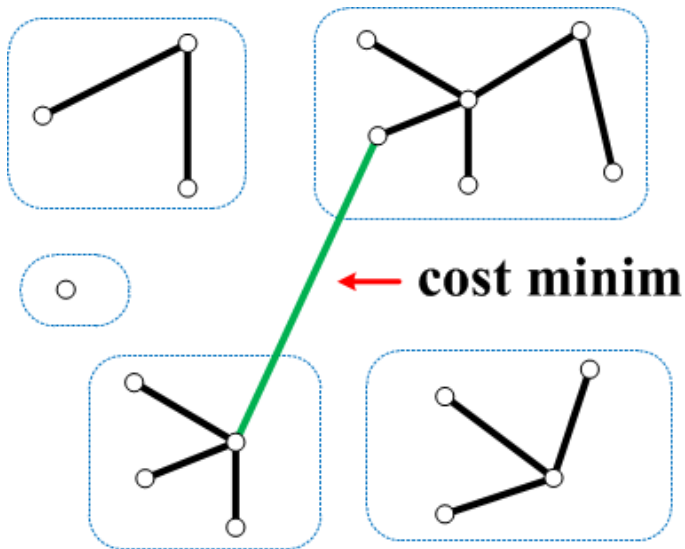
- Inițial: cele  $n$  vârfuri sunt **izolate**, fiecare formând o componentă conexă



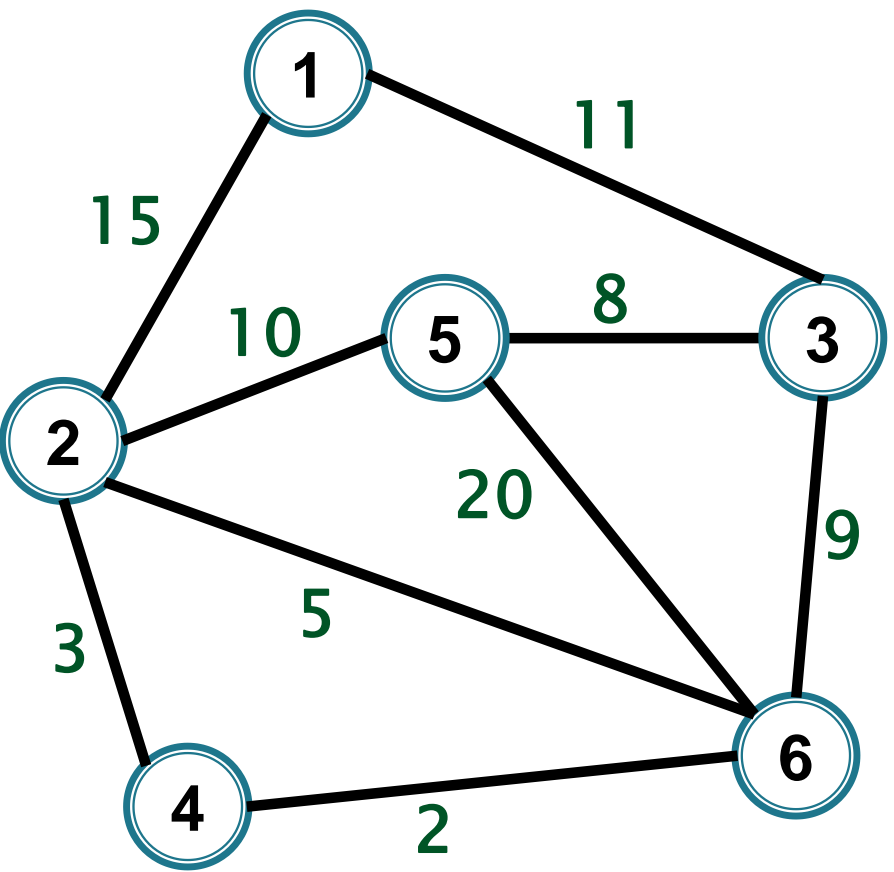
# Kruskal

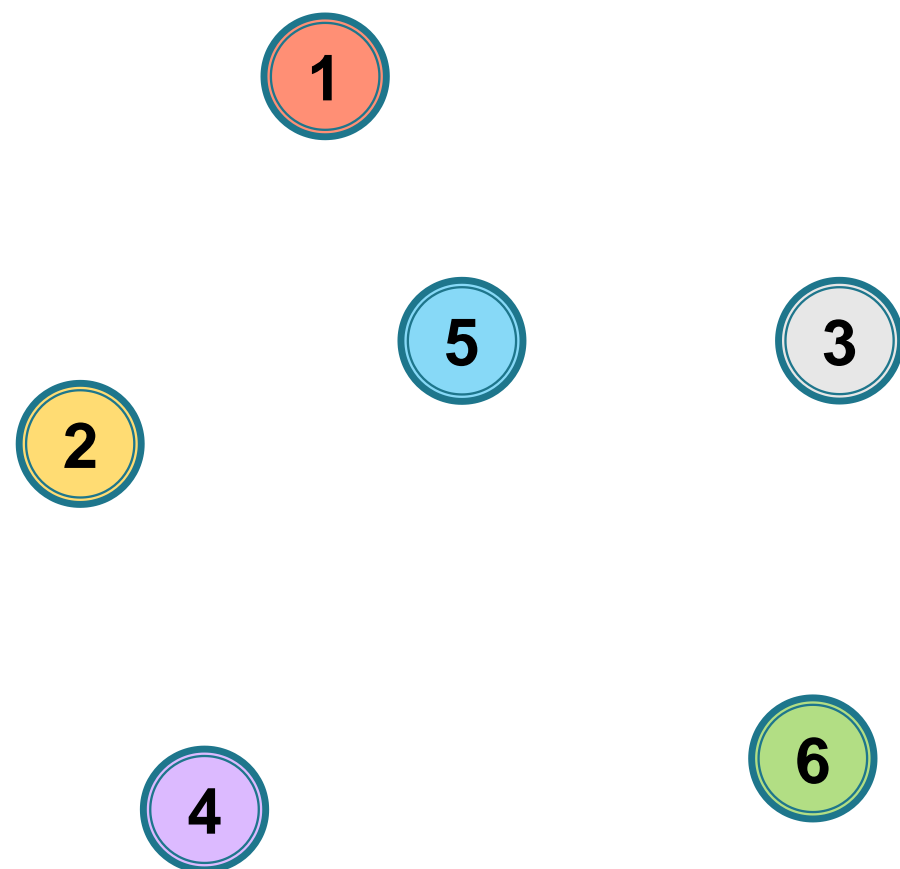
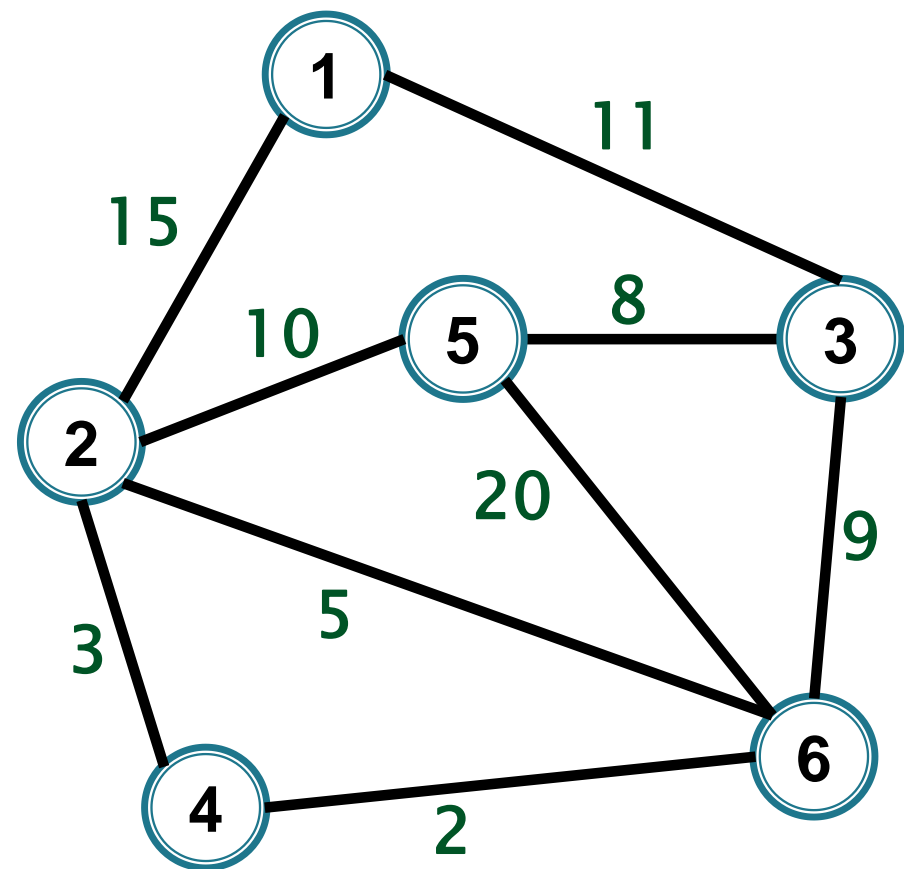
- La un pas:

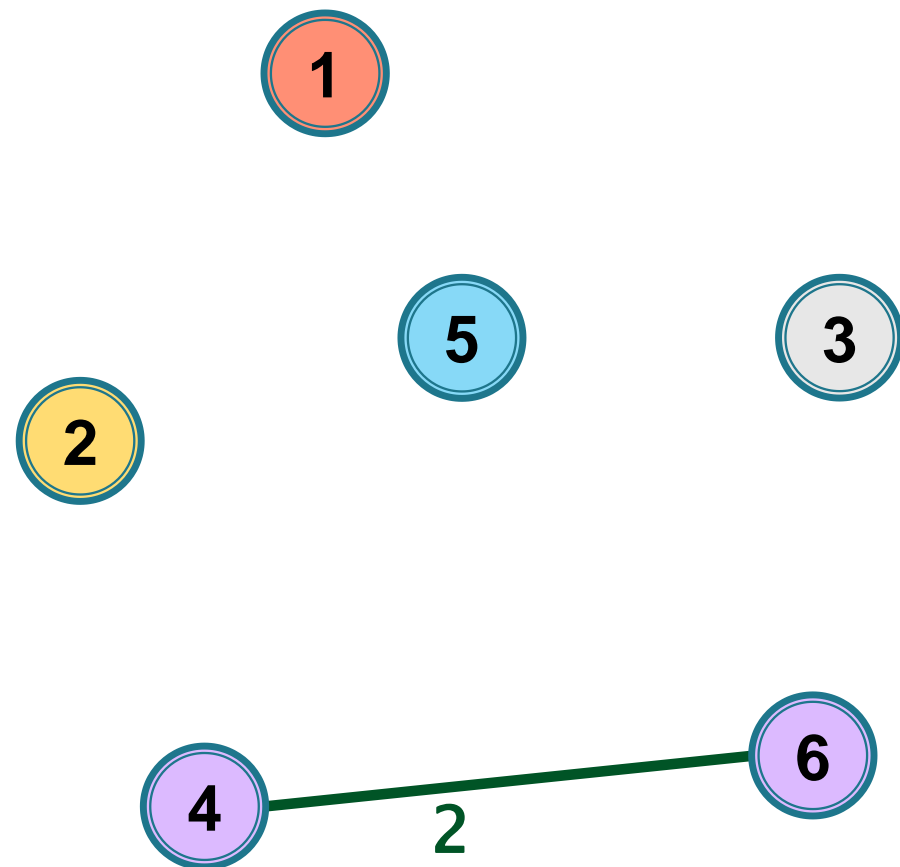
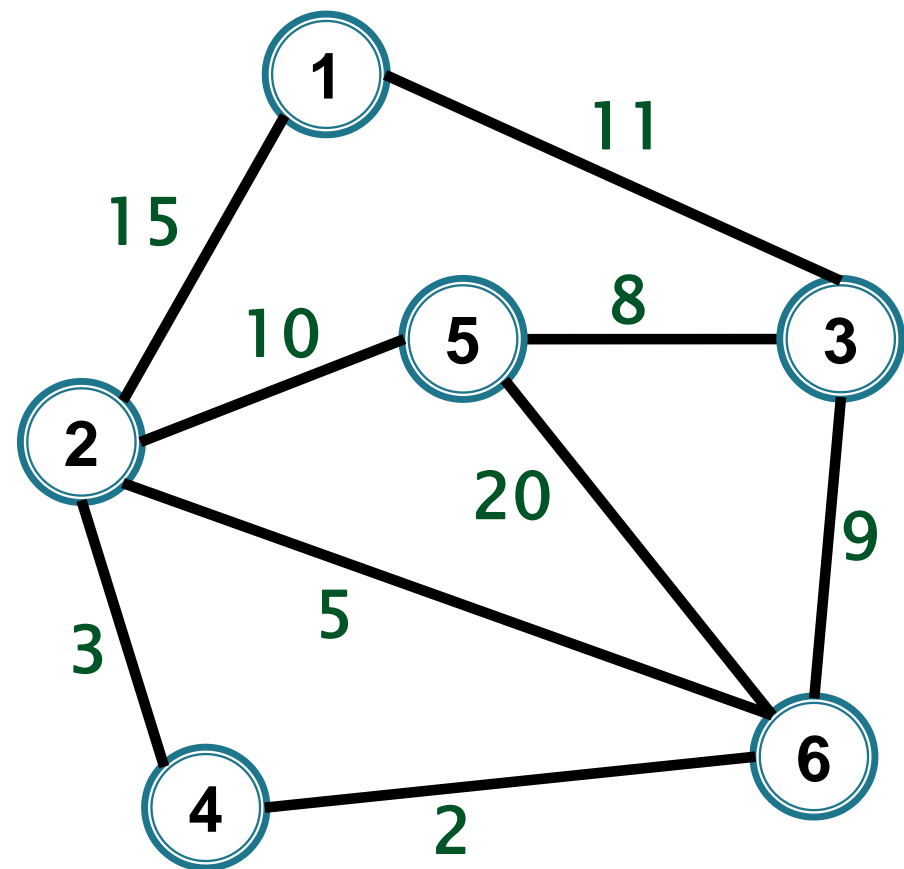
Muchiile selectate formează o pădure

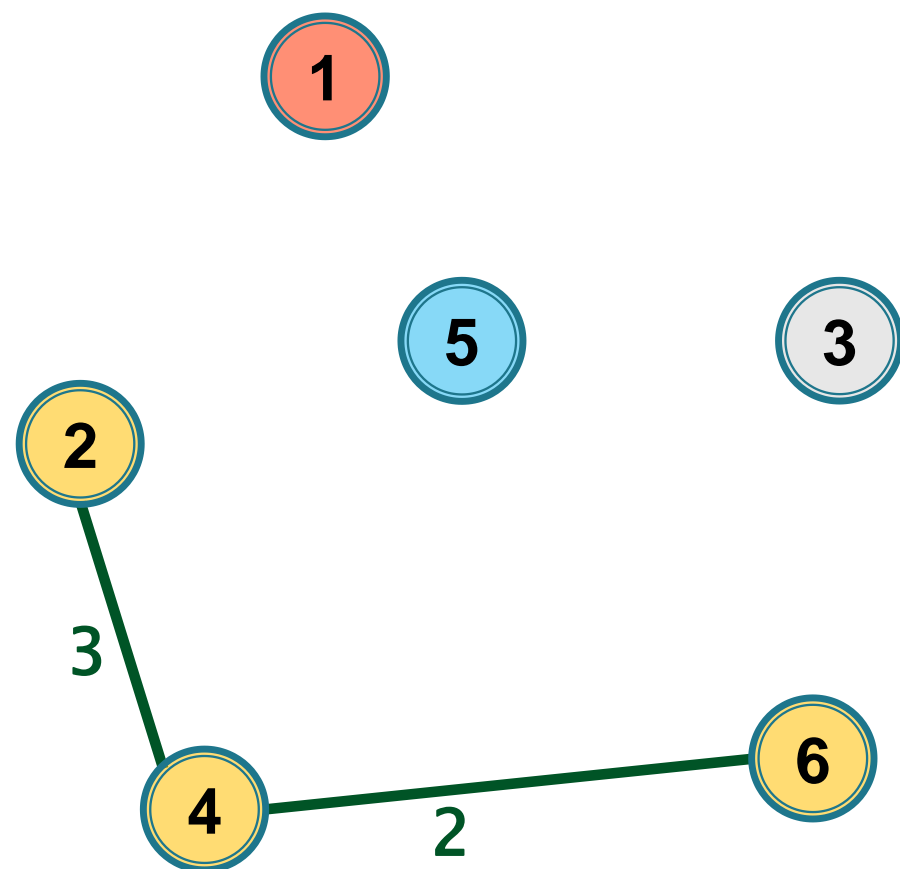
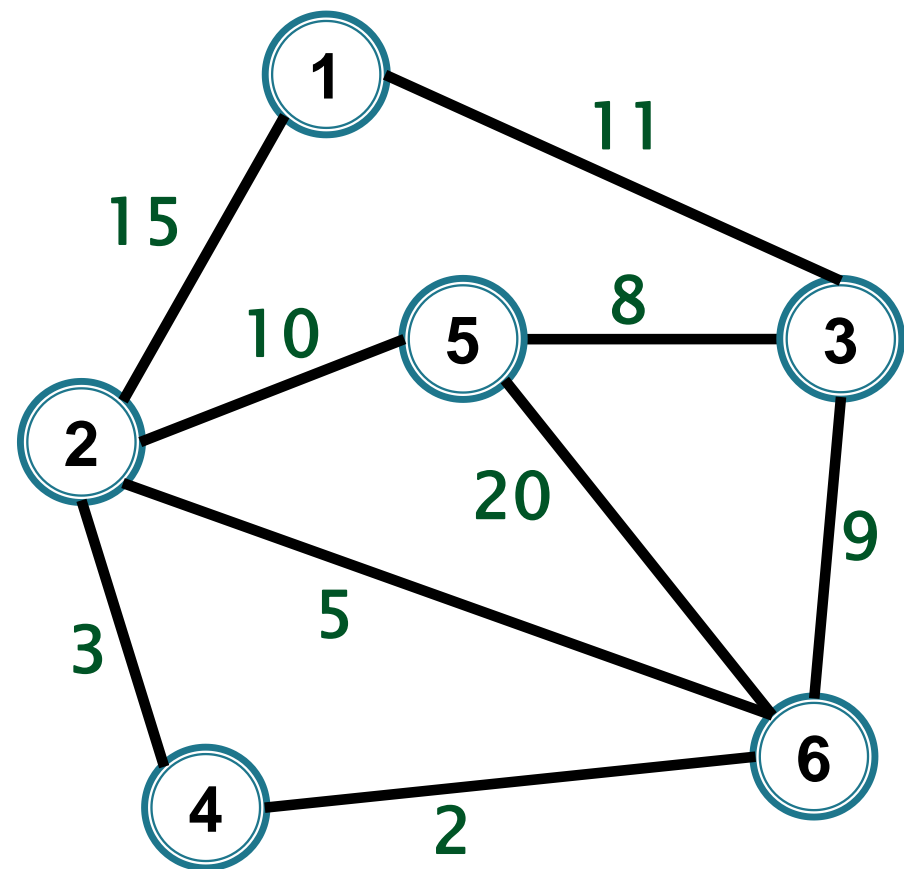


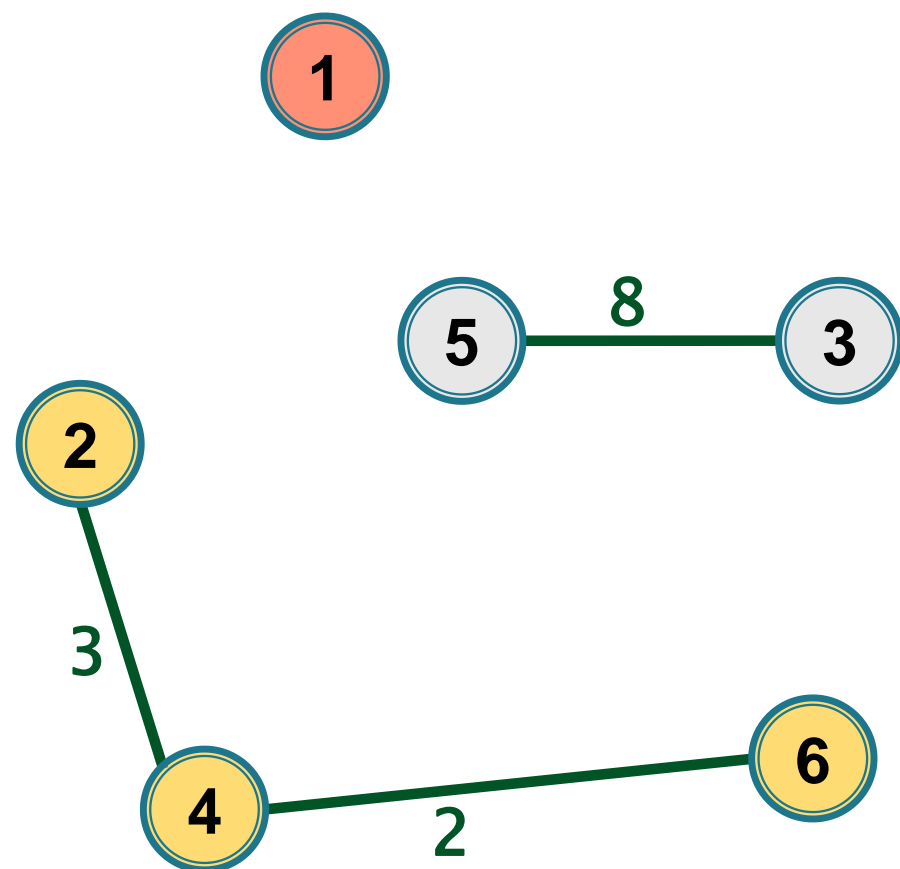
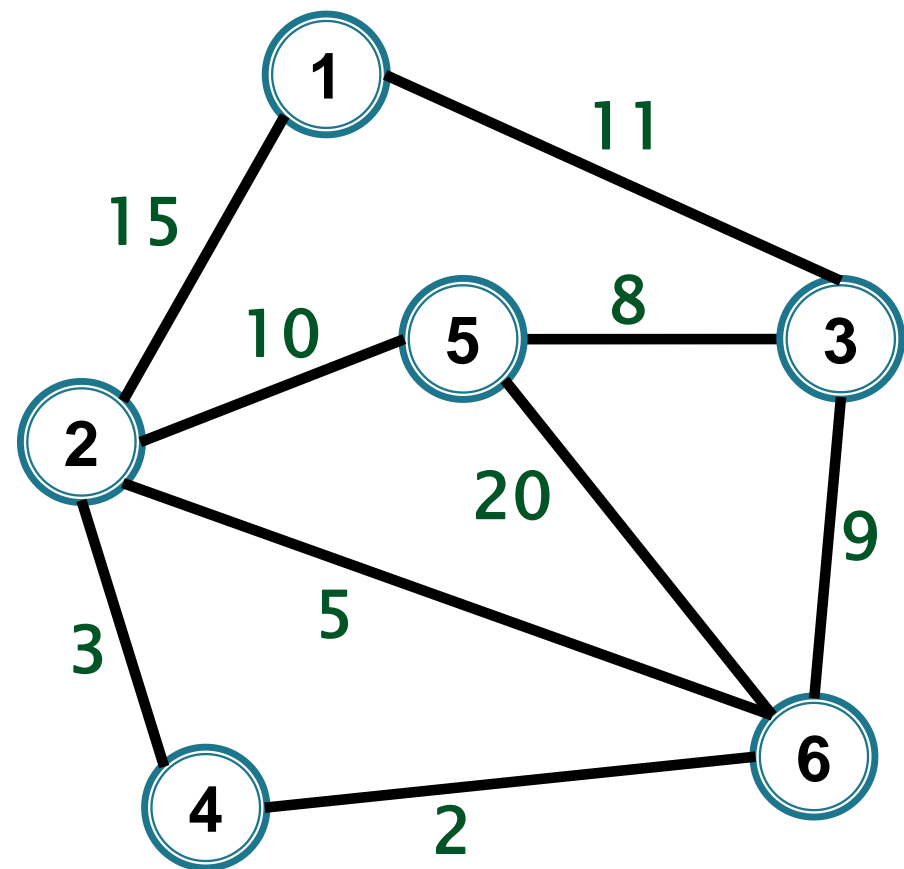
Este selectată o muchie de cost minim care unește doi arbori din pădurea curentă (două componente conexe)

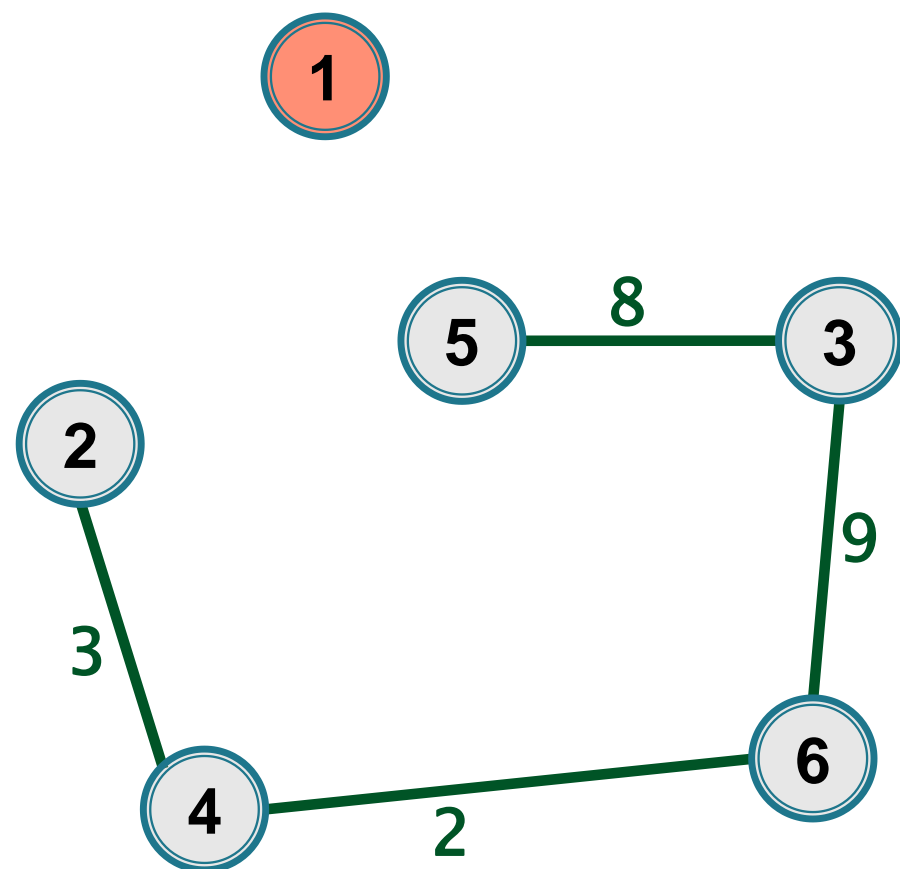
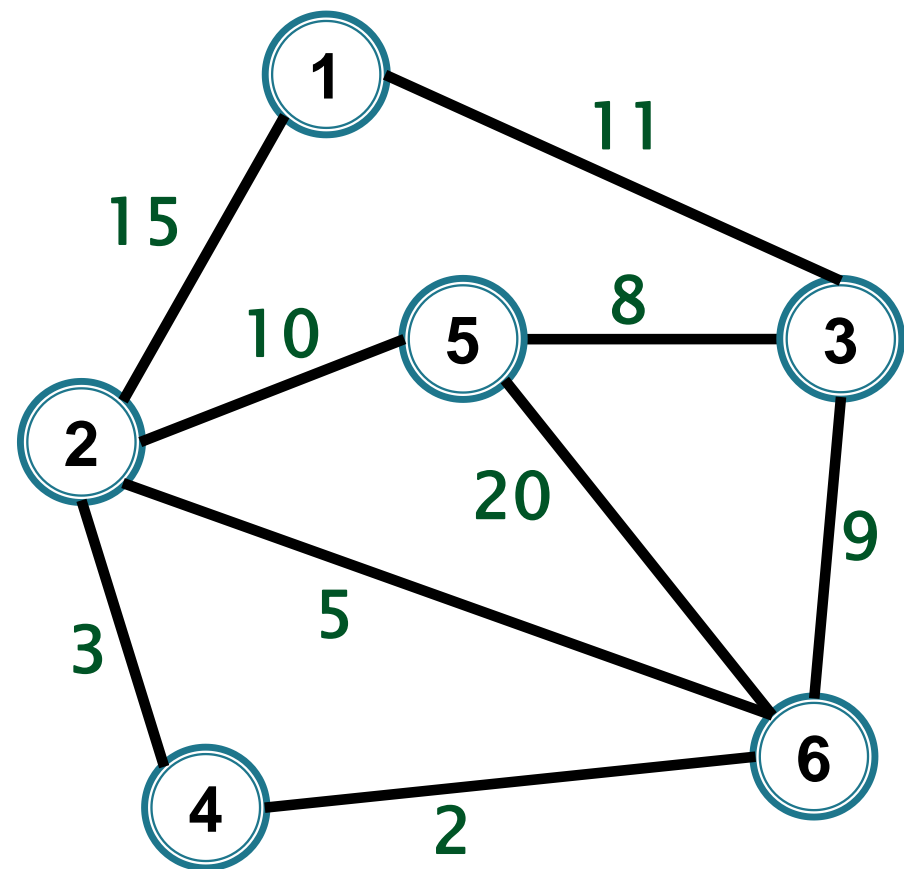




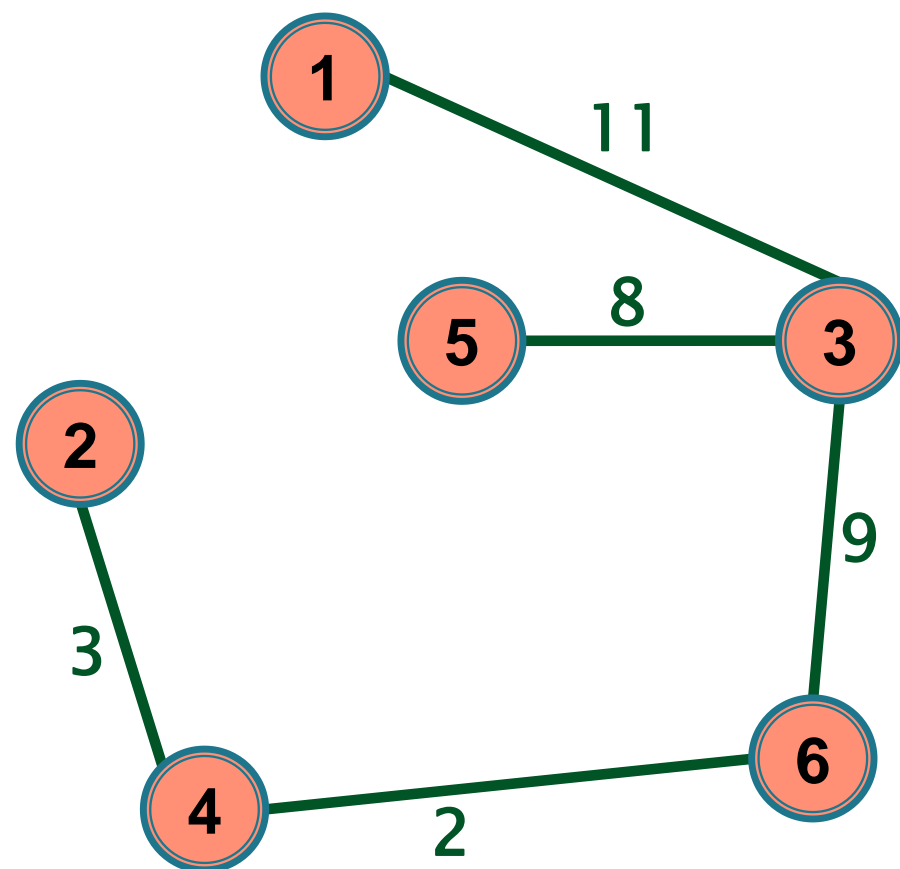
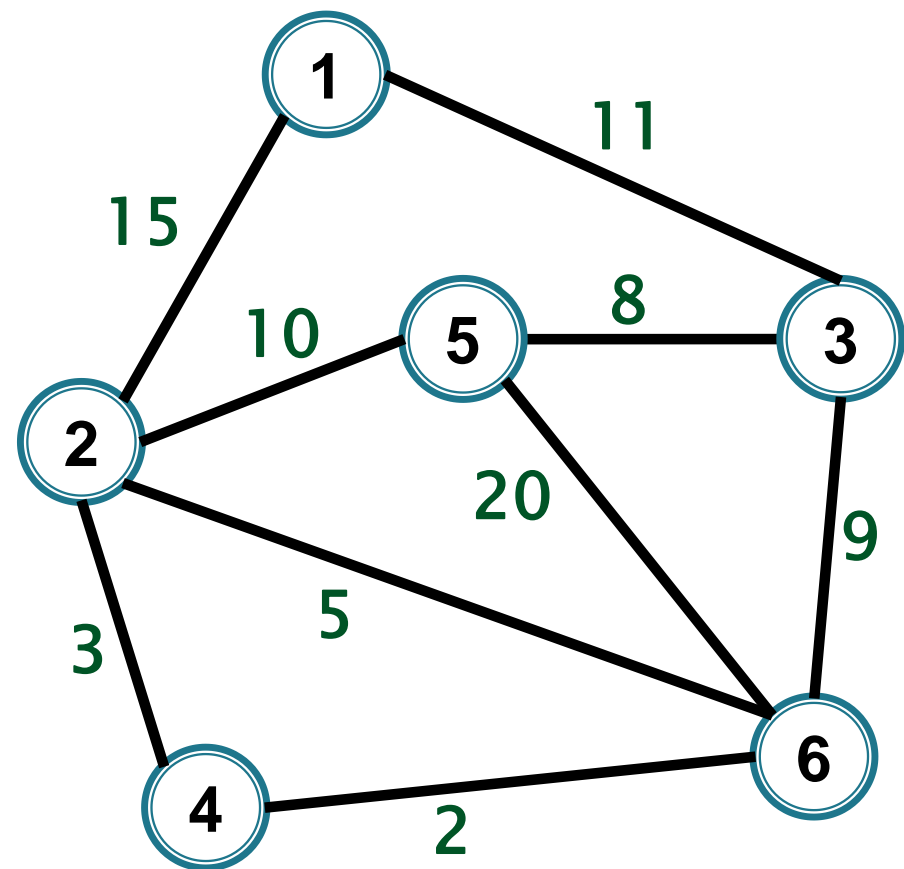












# Kruskal – Implementare



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



Componentele sunt mulțimi disjuncte din  $V$   
(partiție a lui  $V$ )

⇒ **structuri pentru mulțimi disjuncte**



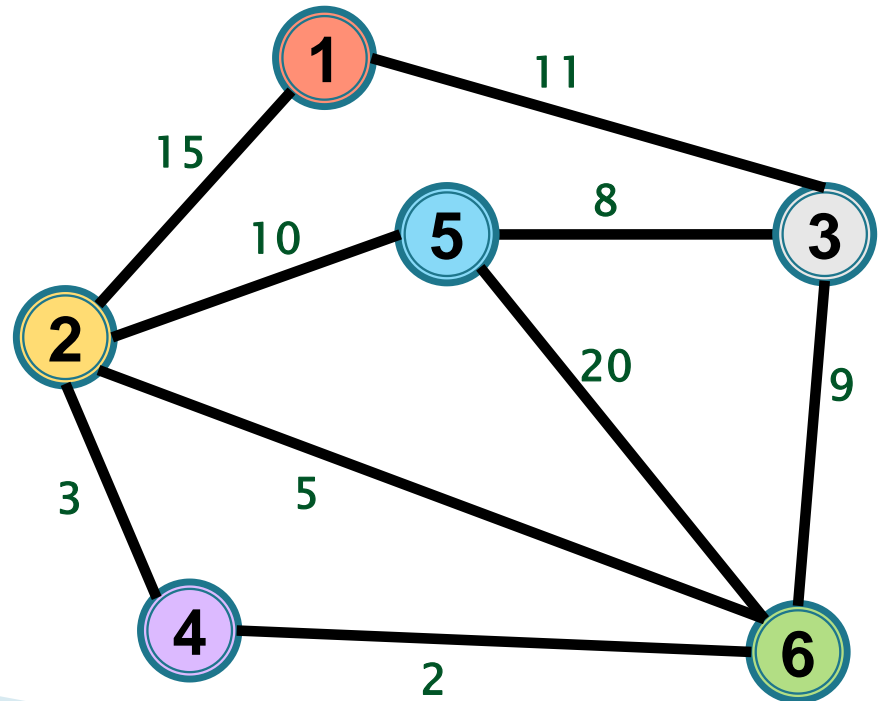
# Kruskal



Componentele 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)** –
- **Reunește(u,v)** –

# Kruskal

## ► Operații necesare:

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

**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) ;
nrmsel=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) ∪ {uv} ;
        Reuneste (u,v) ;
        nrmsel=nrmsel+1;
        if (nrmsel==n-1)
            STOP;
    }
```

# Kruskal

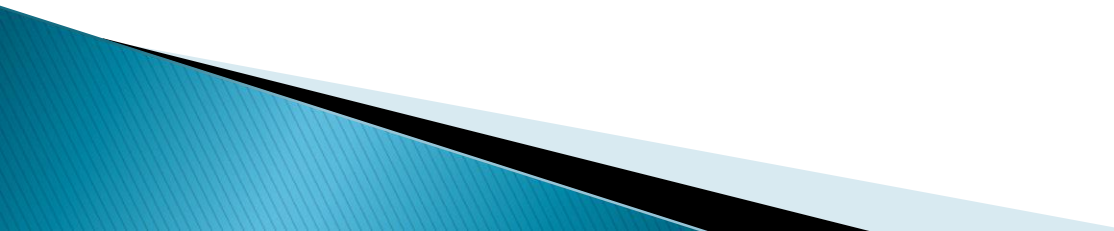
## Complexitate



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



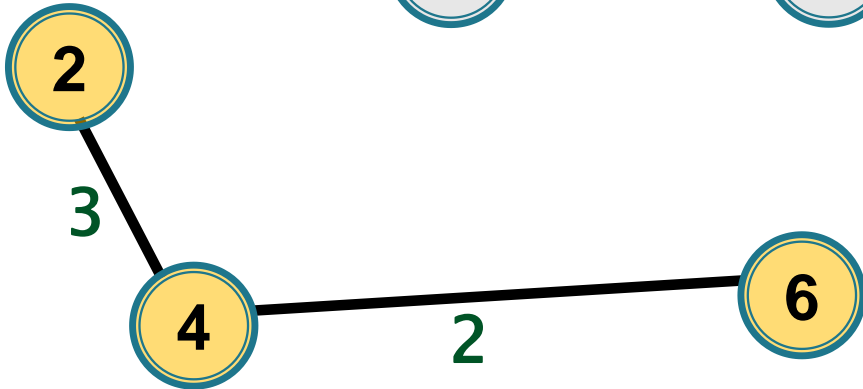
Cum memorăm componentele +  
reprezentantul / culoarea componentei în  
care se află un vârf?

# Kruskal

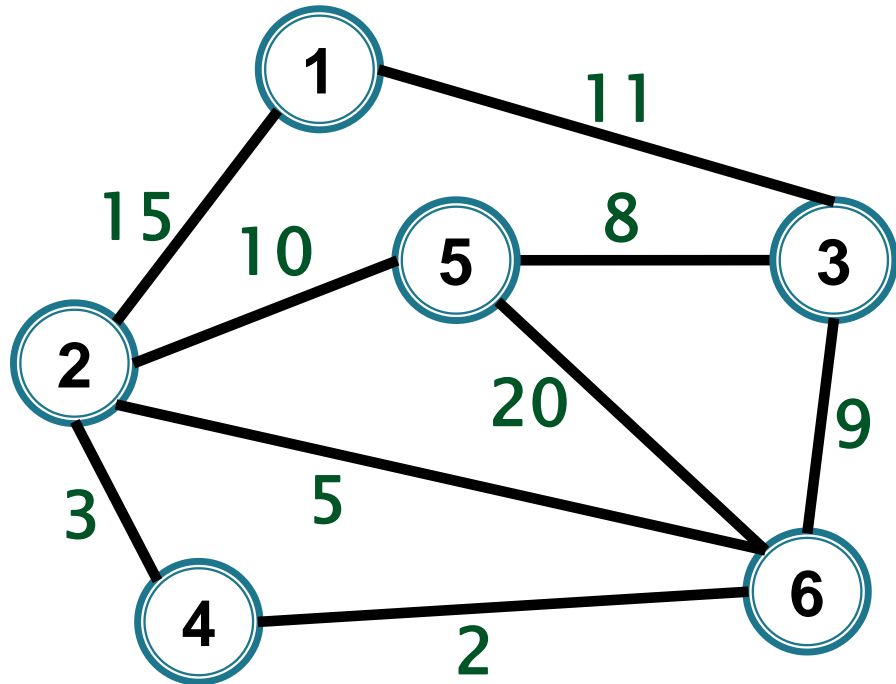


**Varianta 1** – memorăm într-un vector pentru fiecare vârf reprezentantul/culoarea componentei din care face parte

$r[u]$  = 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

- ▶ Initialize –  $O(1)$

```
void Initialize(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

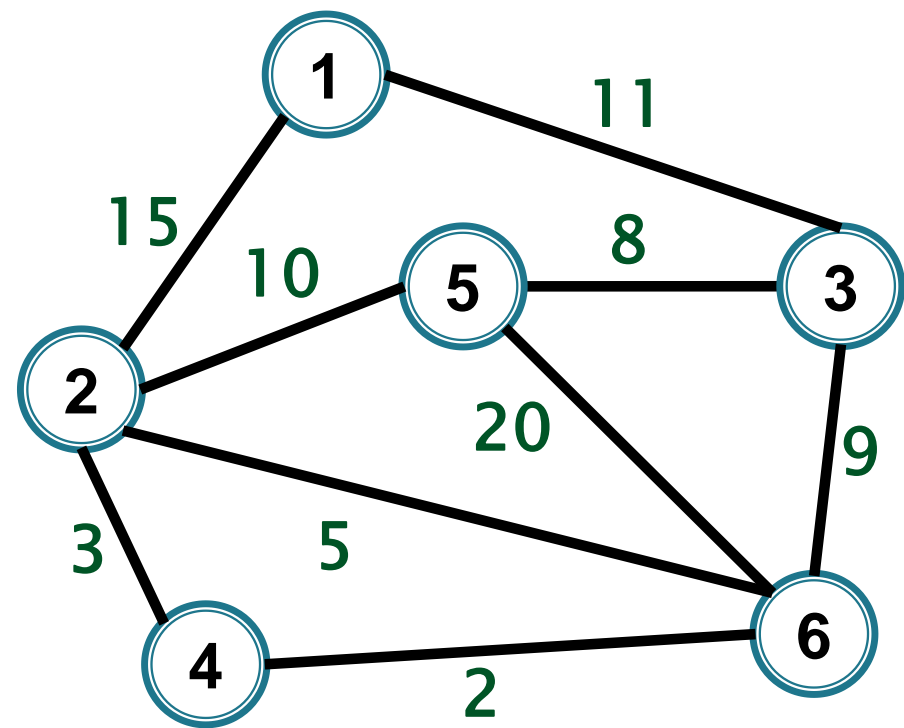
## Complexitate

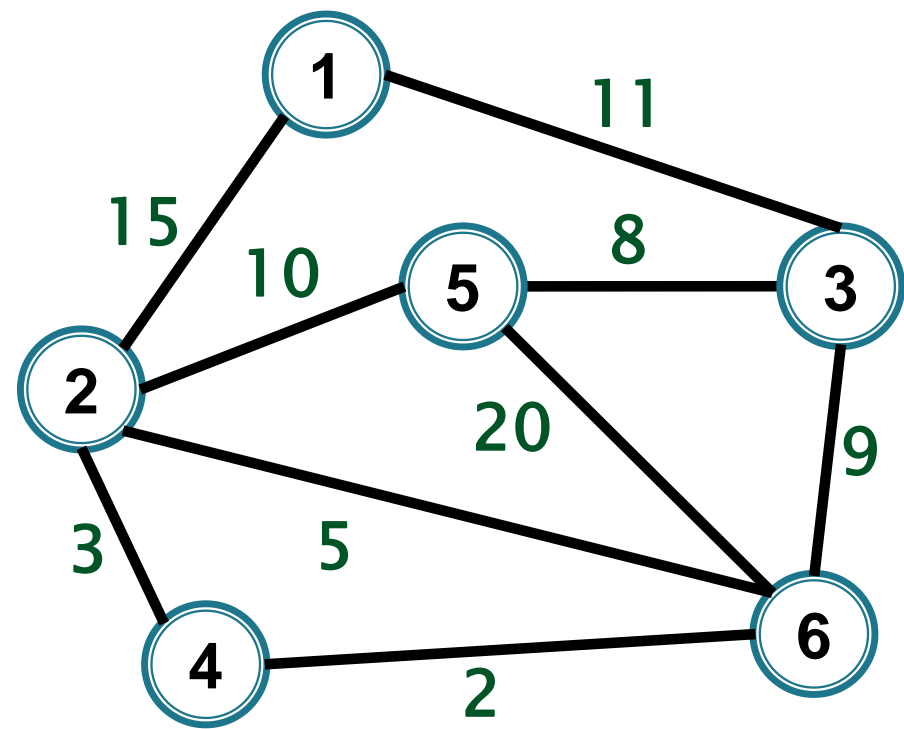
**Varianta 1** – dacă folosim vector de reprezentanți

- **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)

1

2

5

3

4

6

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

(4,6)

(2,4)

(2,6)

(3,5)

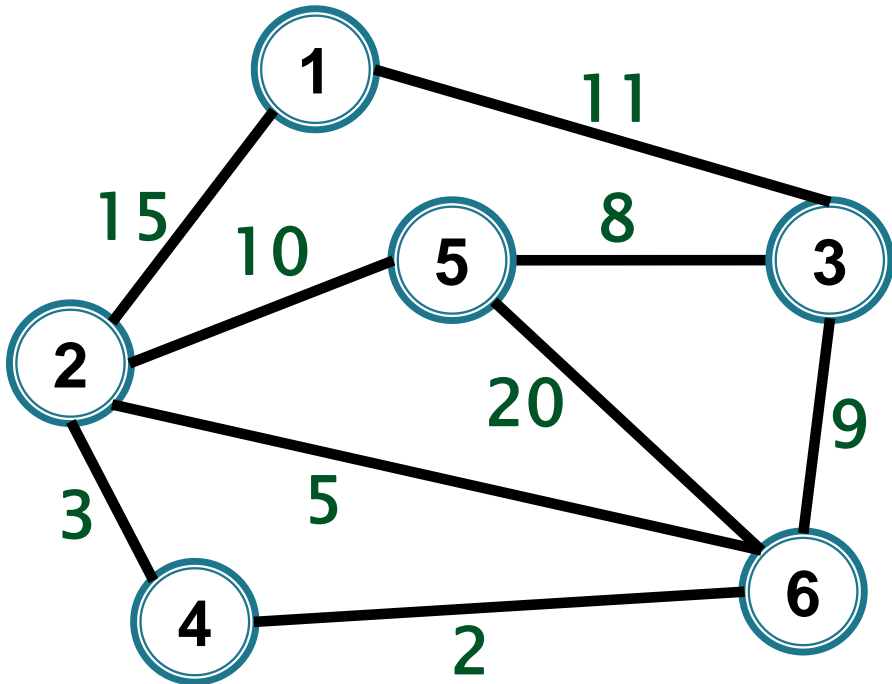
(3,6)

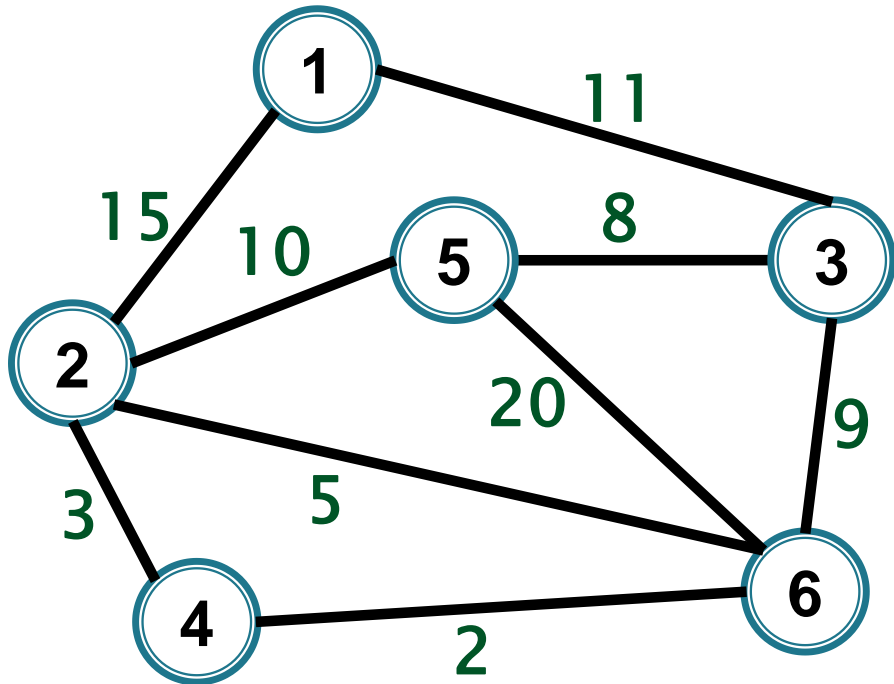
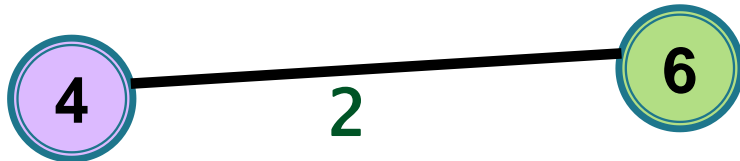
(2,5)

(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)

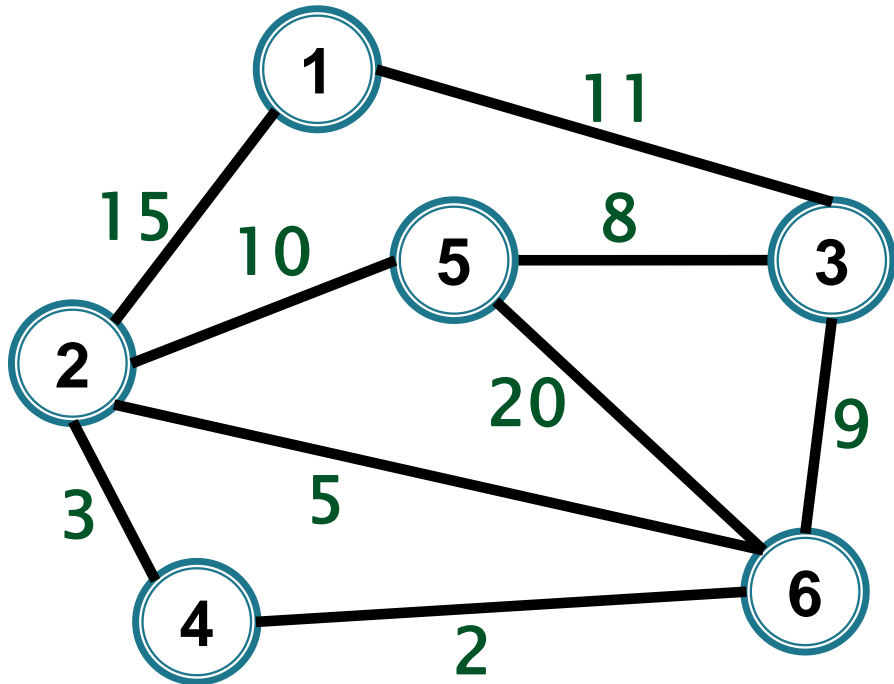
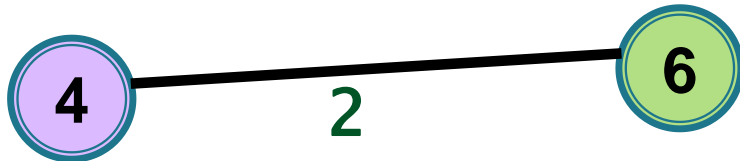
(2,5)

(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)

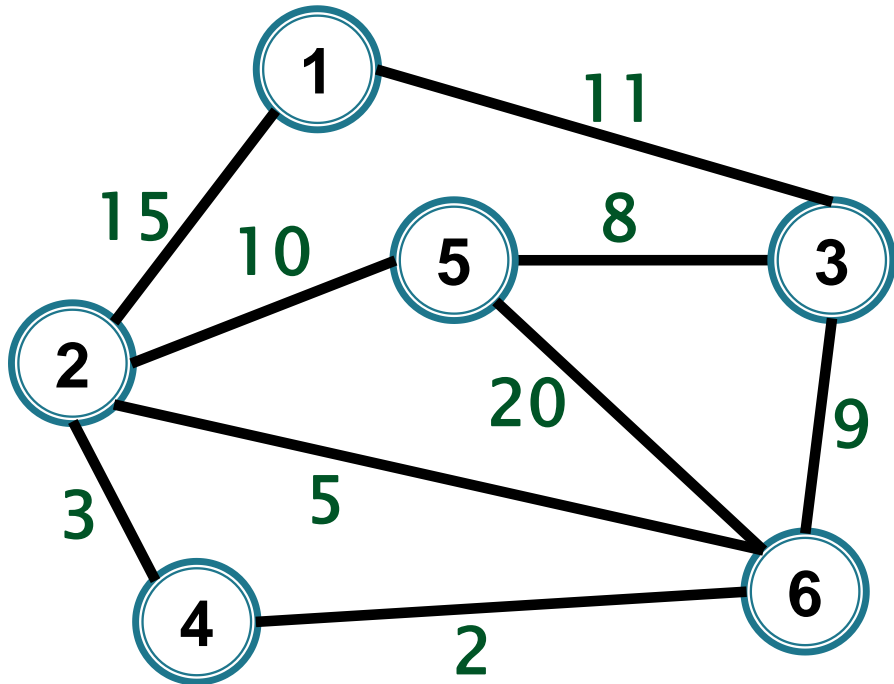
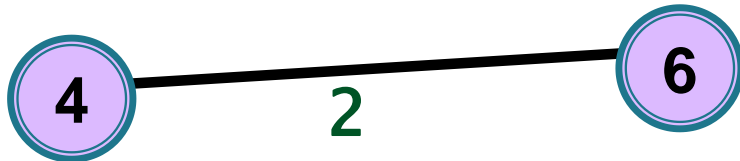
(3,6)

(2,5)

(1,3)

(1,2)

(5,6)



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

**(4,6)**

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

(2,4)

(2,6)

(3,5)

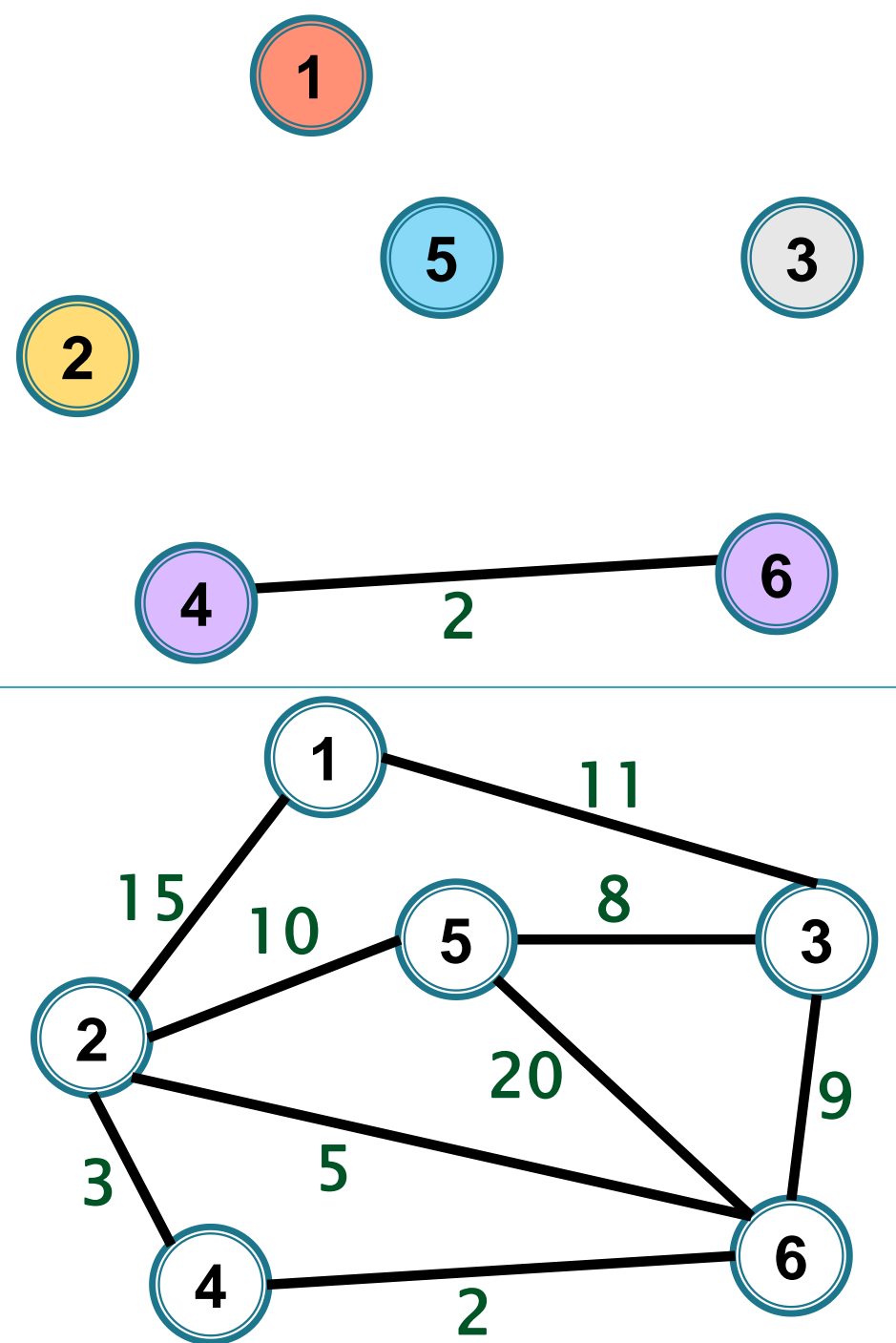
(3,6)

(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)

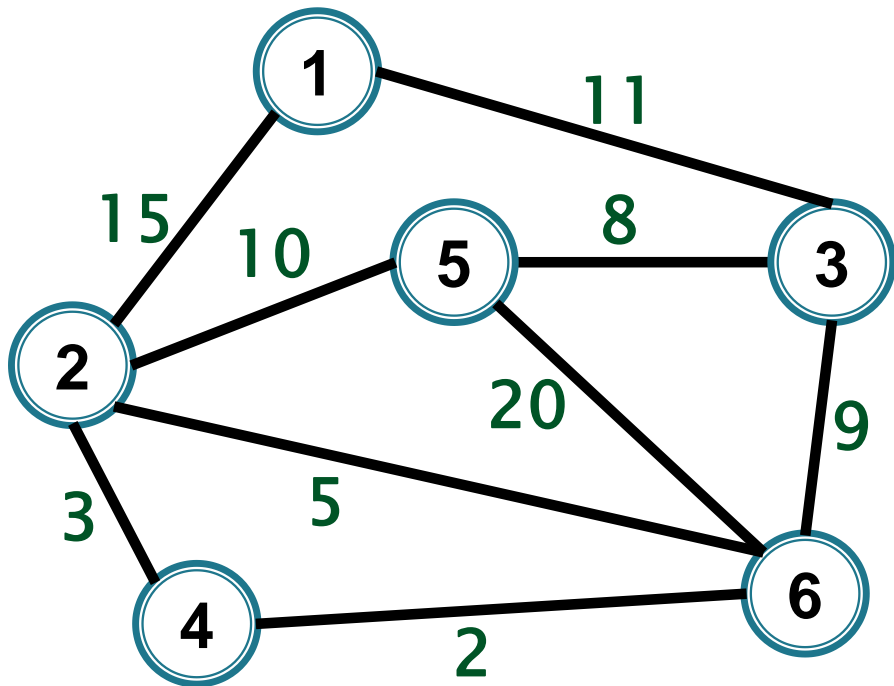
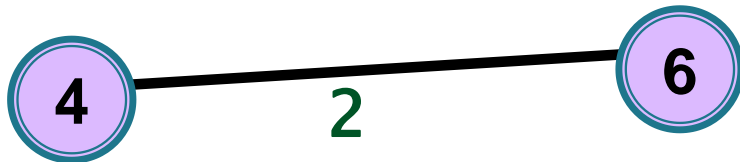
(3,6)

(2,5)

(1,3)

(1,2)

(5,6)



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

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

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

$(2, 6)$

$(3, 5)$

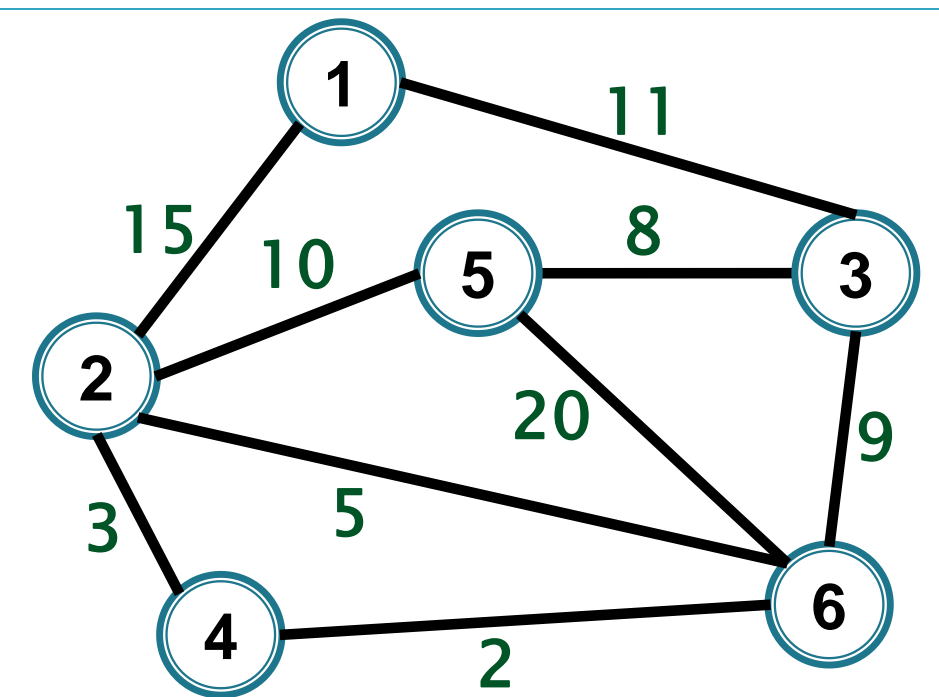
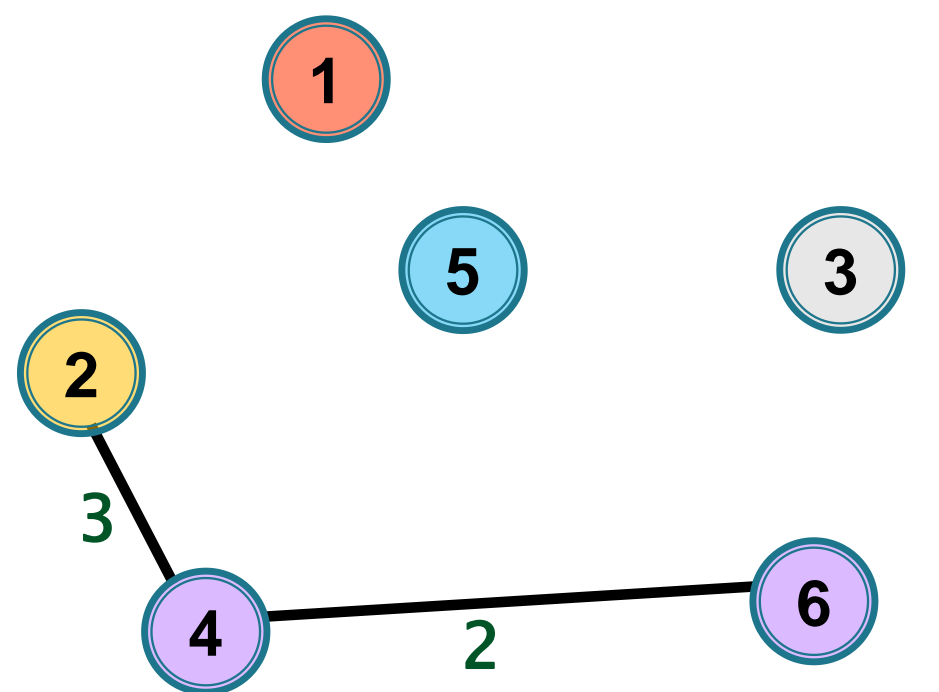
$(3, 6)$

$(2, 5)$

$(1, 3)$

$(1, 2)$

$(5, 6)$



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

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

$(4, 6)$

$(2, 4)$

$(2, 6)$

$(3, 5)$

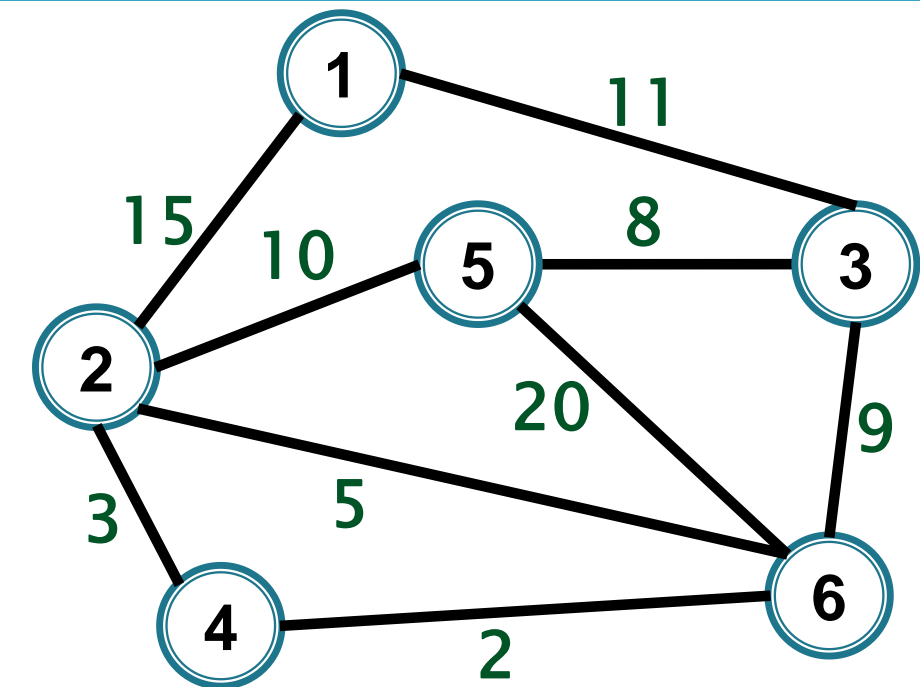
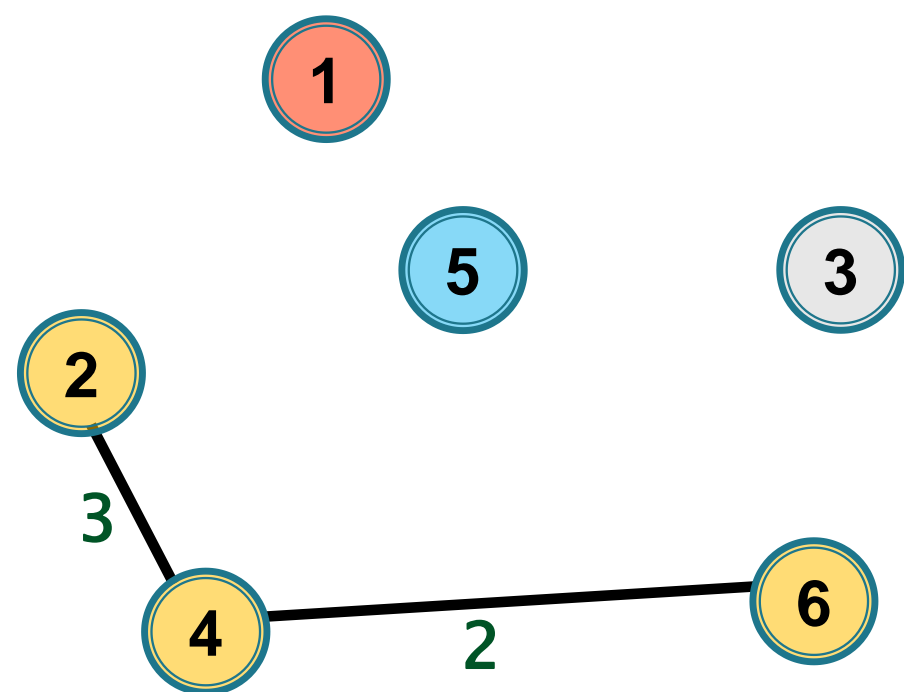
$(3, 6)$

$(2, 5)$

$(1, 3)$

$(1, 2)$

$(5, 6)$



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

(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)

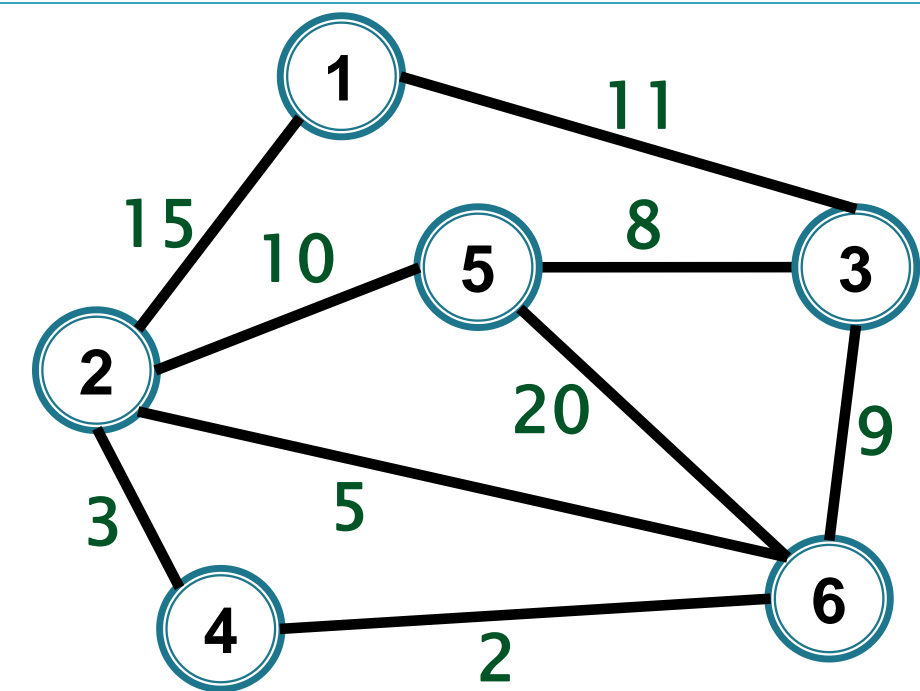
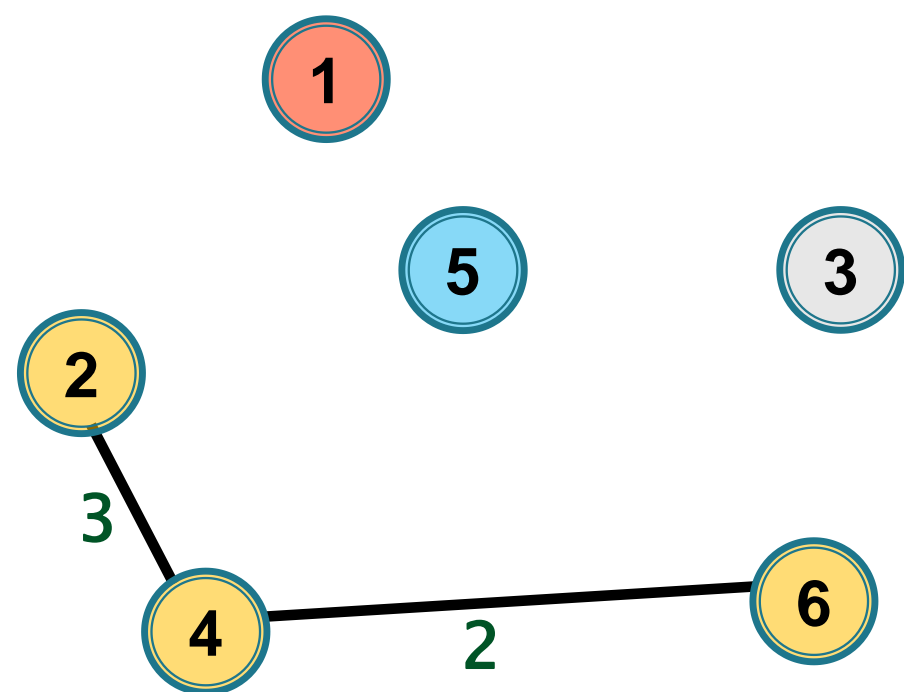
(3,6)

(2,5)

(1,3)

(1,2)

(5,6)



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

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

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

$(2, 6)$

$(3, 5)$

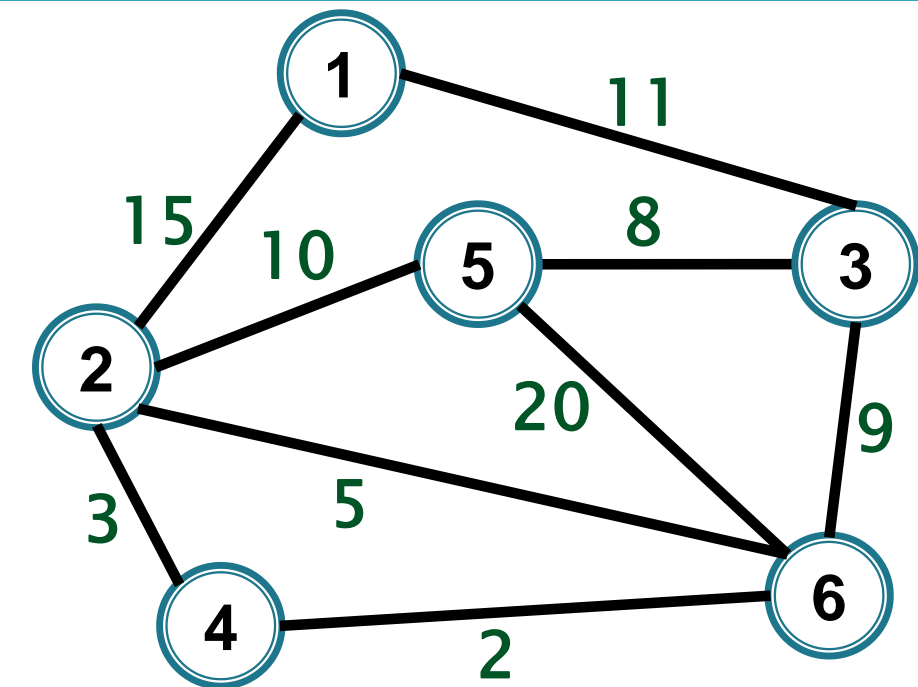
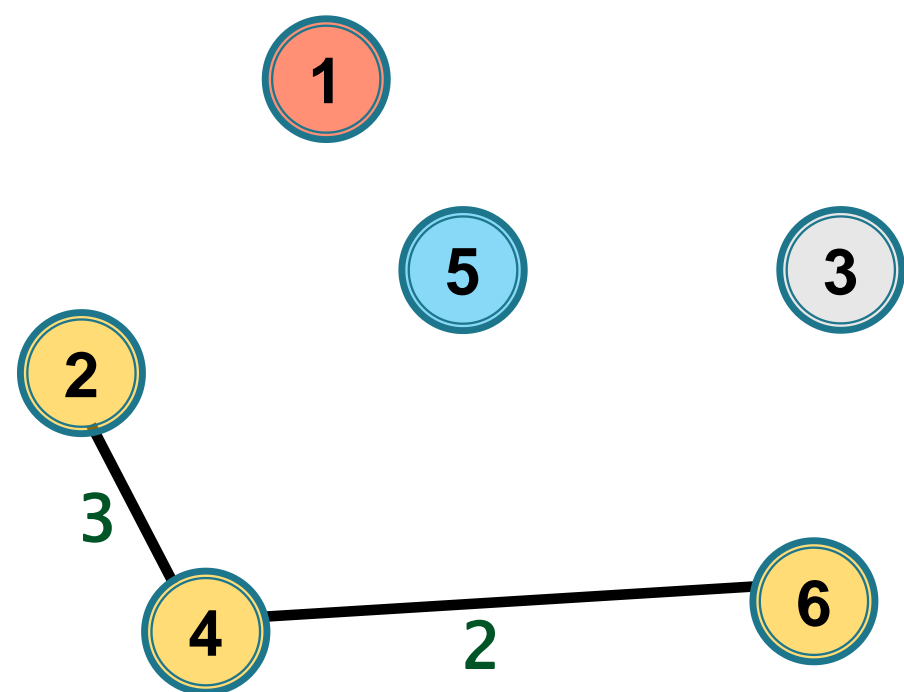
$(3, 6)$

$(2, 5)$

$(1, 3)$

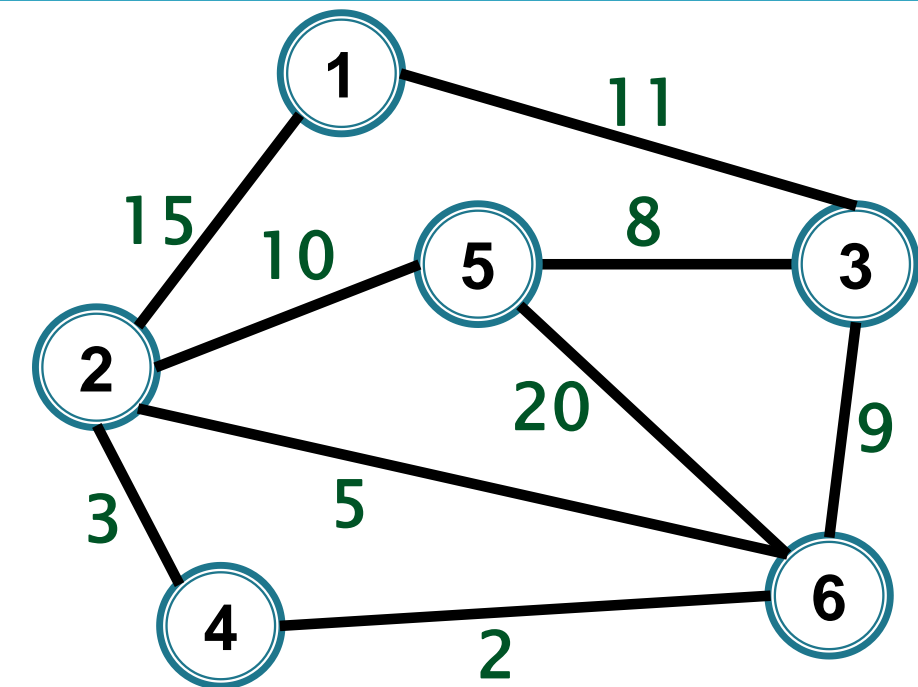
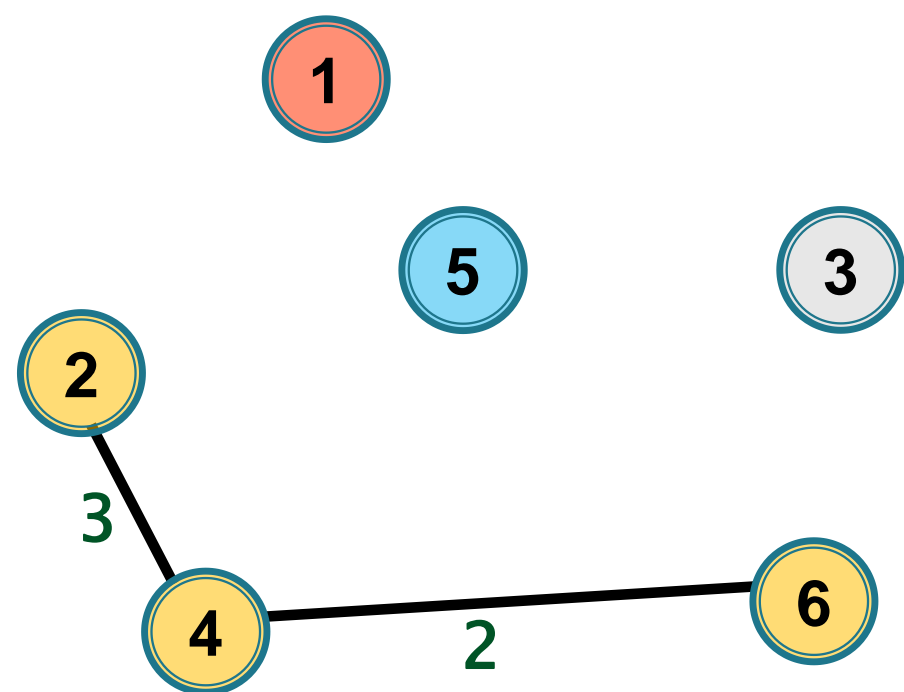
$(1, 2)$

$(5, 6)$



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

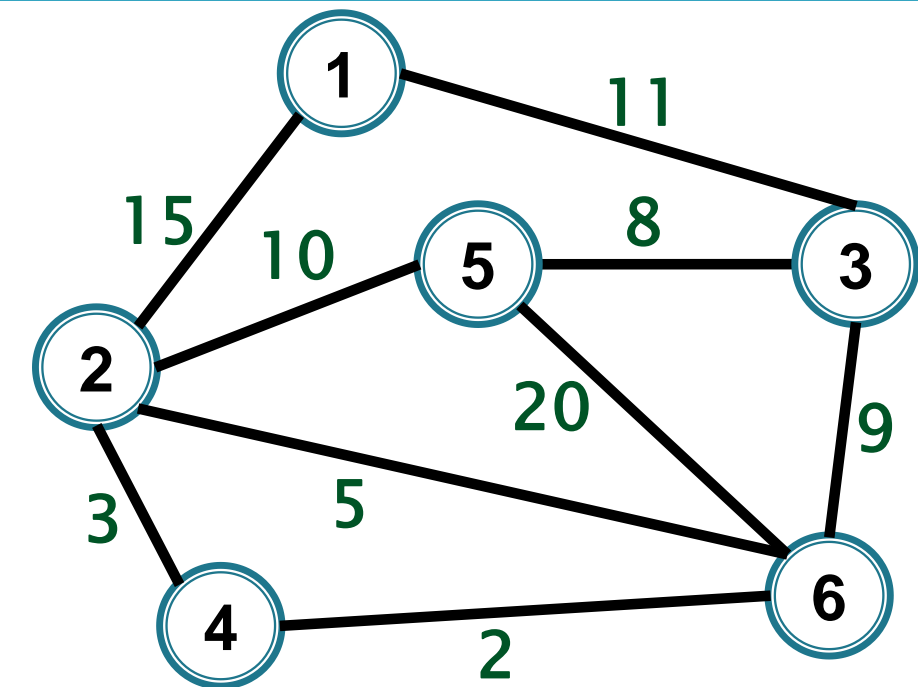
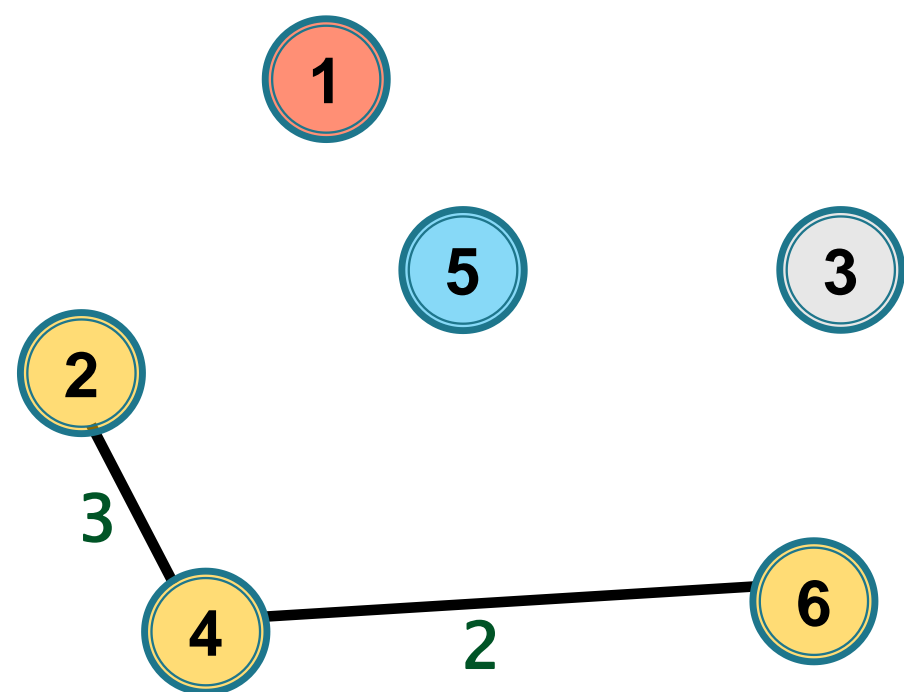
(5,6)

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

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

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

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



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

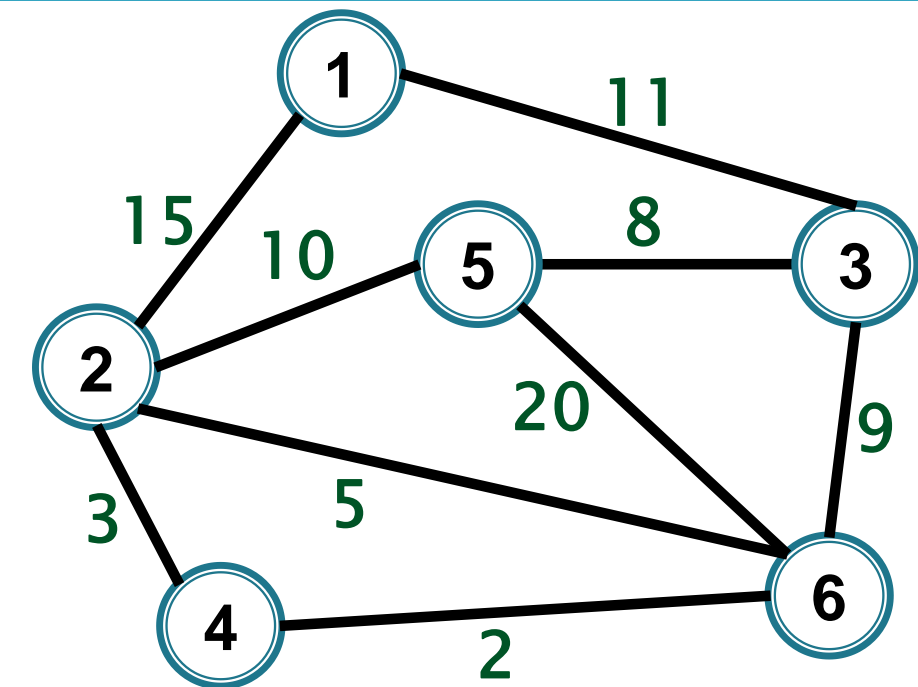
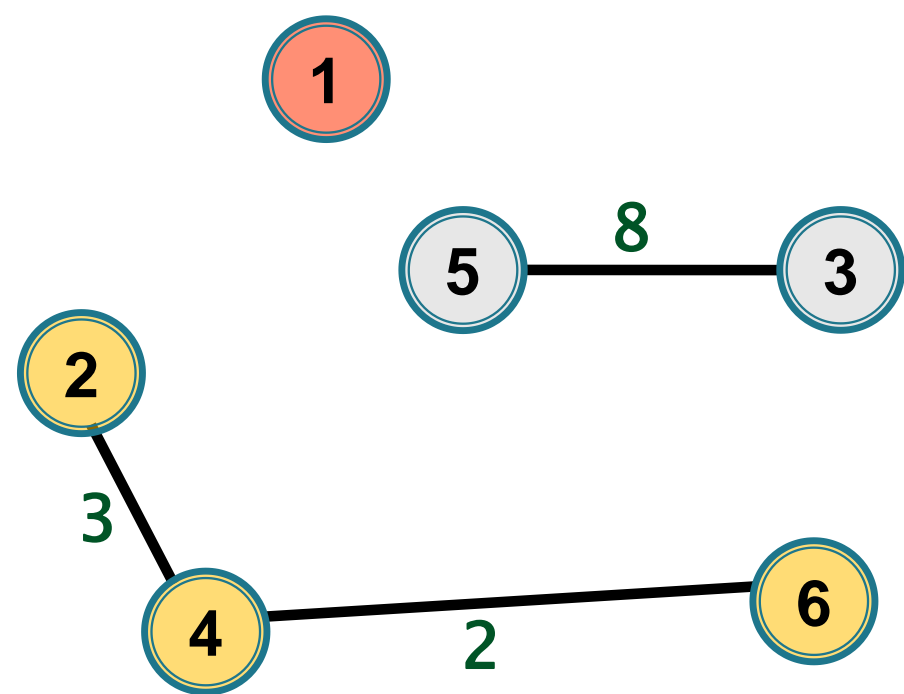
$r = [1, 2, 3, 4, 5, 6]$

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

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

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

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



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

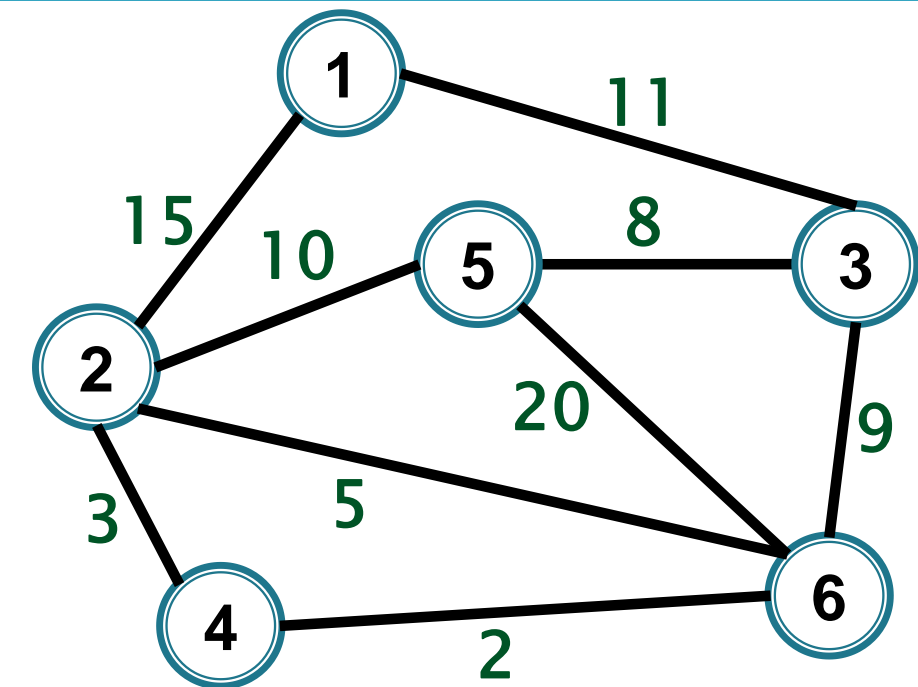
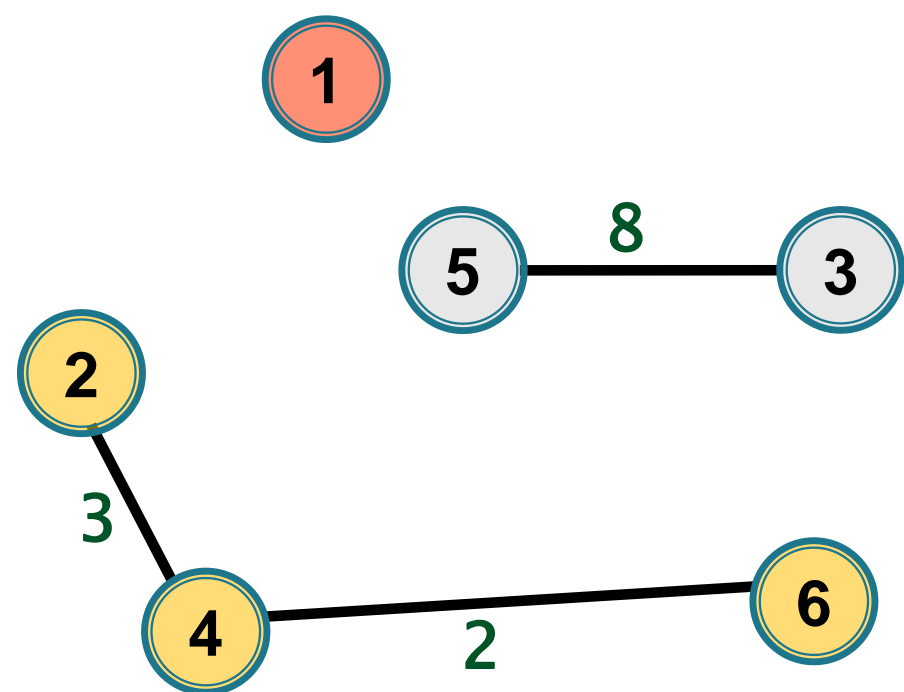
$r = [1, 2, 3, 4, 5, 6]$

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

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

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

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



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

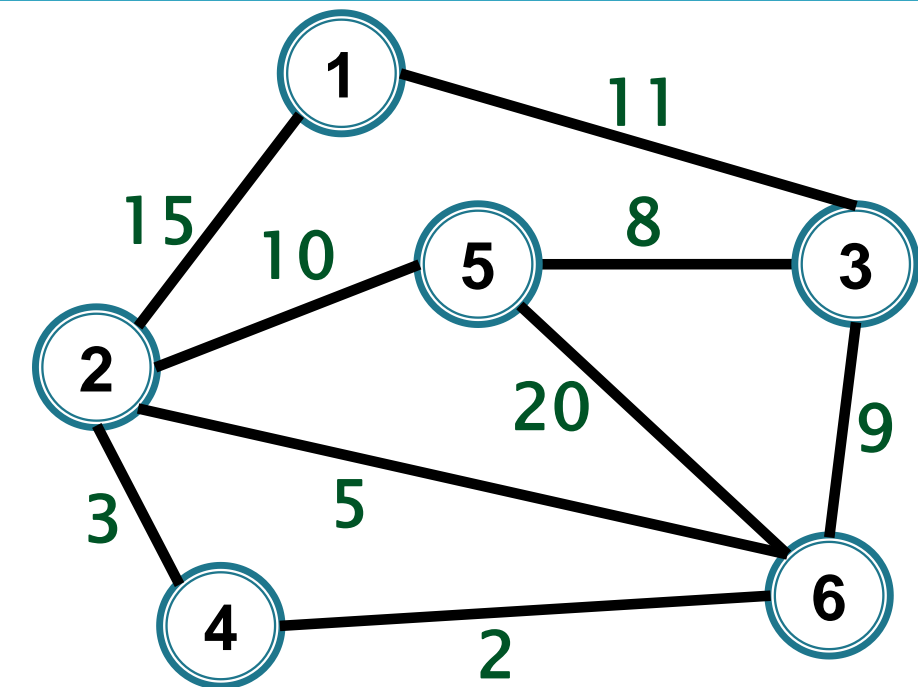
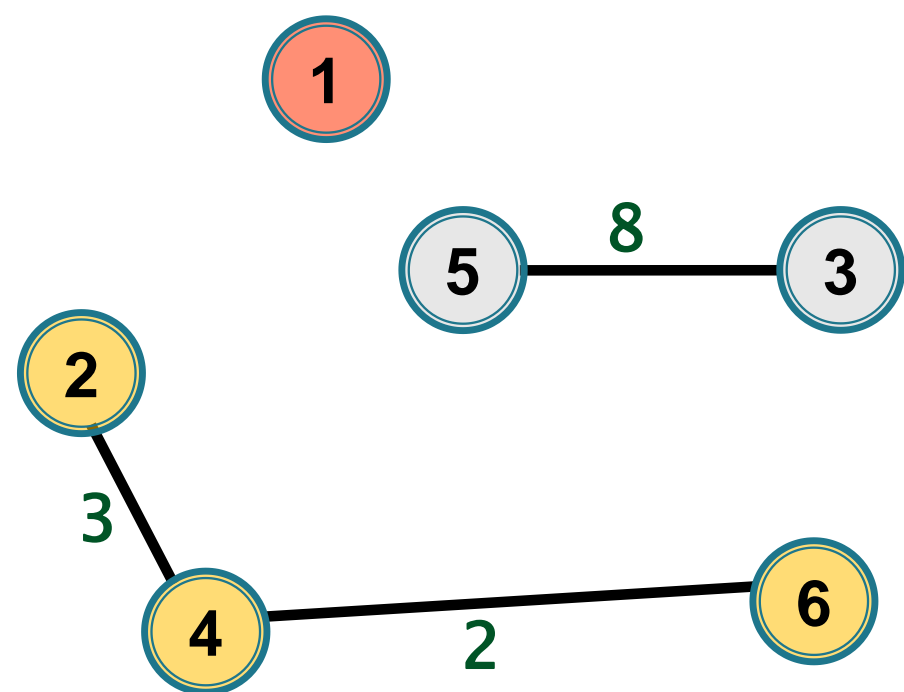
$r = [1, 2, 3, 4, 5, 6]$

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

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

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

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



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

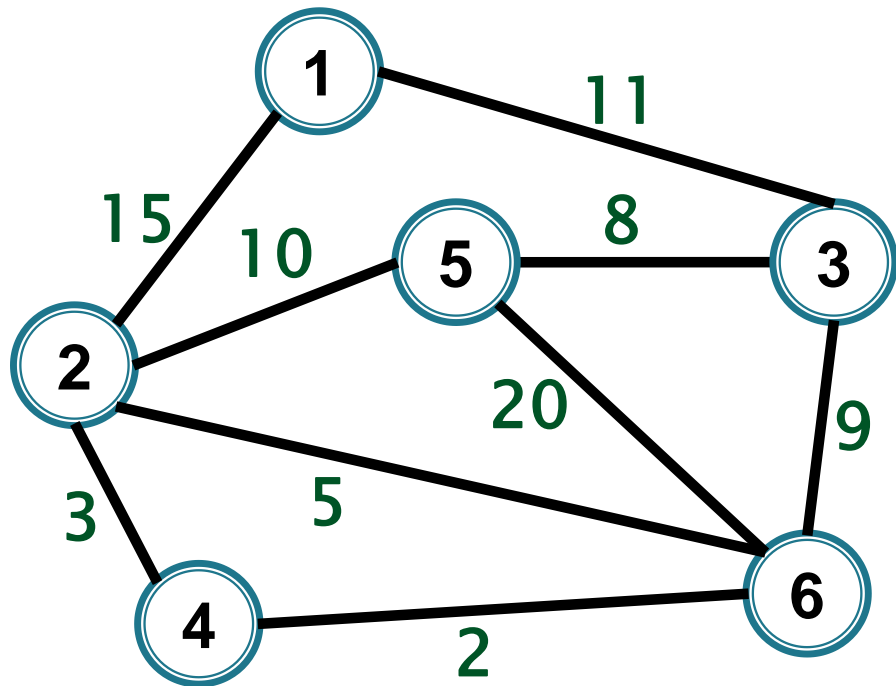
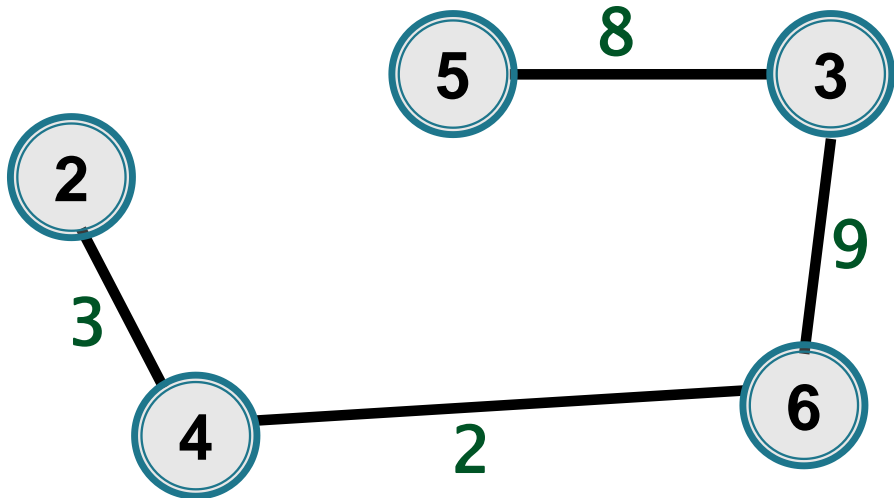
$r = [1, 2, 3, 4, 5, 4]$

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

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

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

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



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

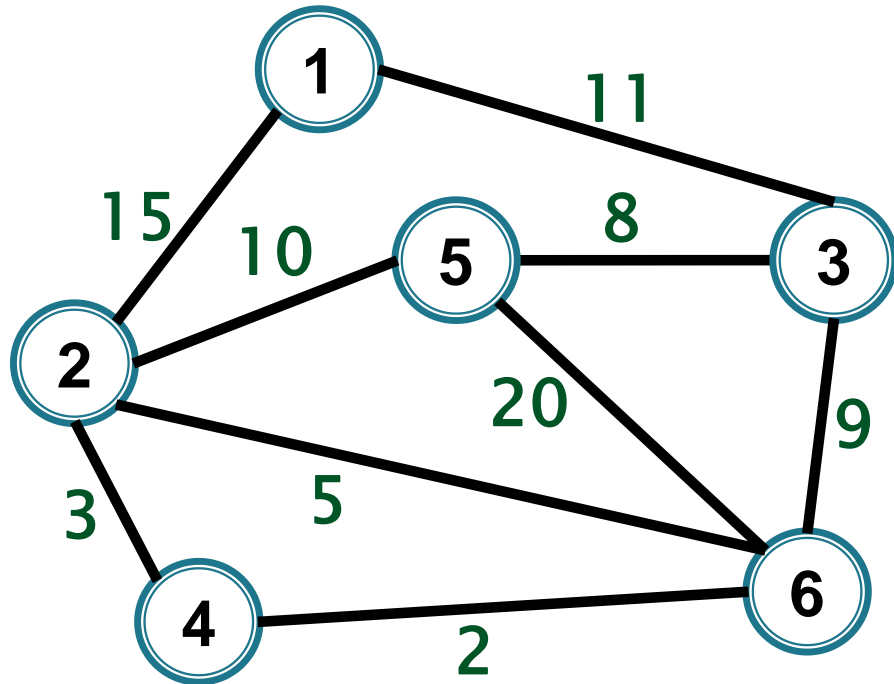
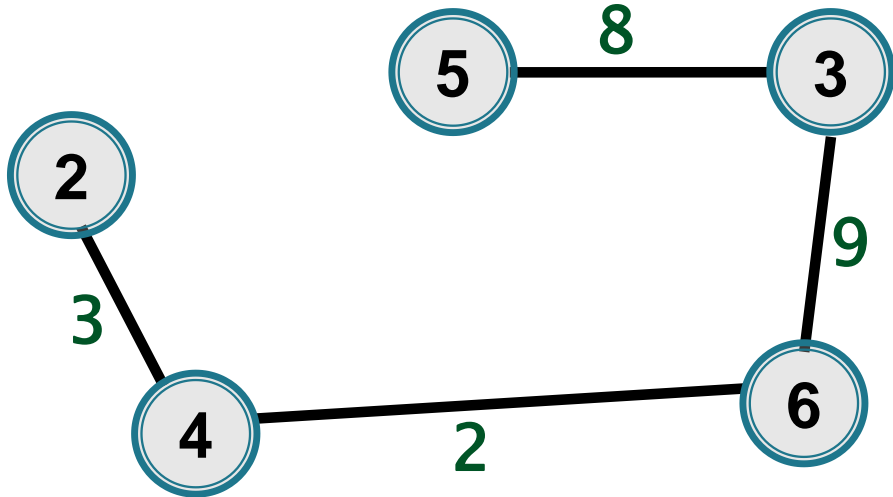
$r = [1, 2, 3, 4, 5, 4]$

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

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

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

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



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

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

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

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

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

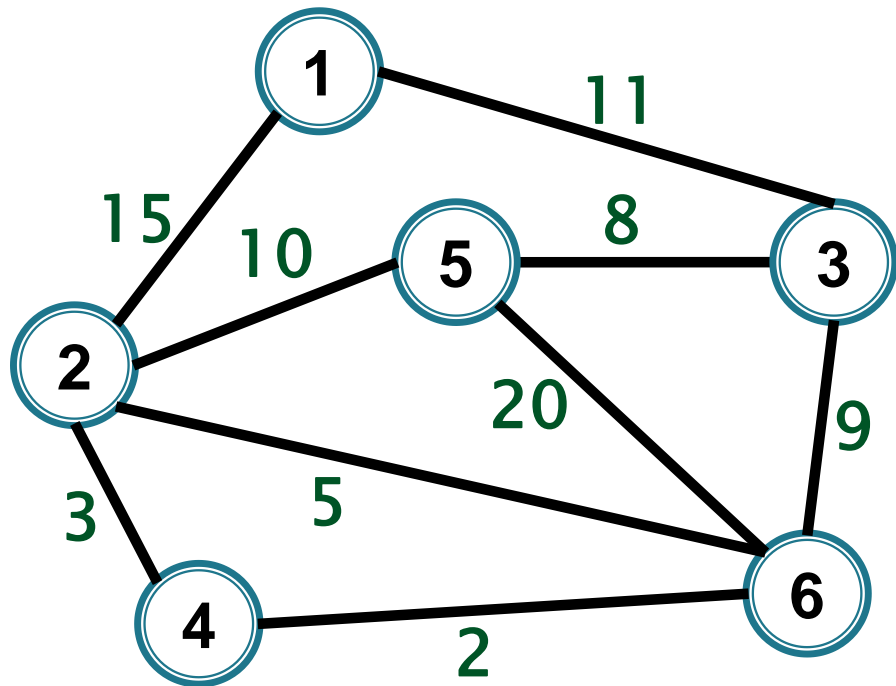
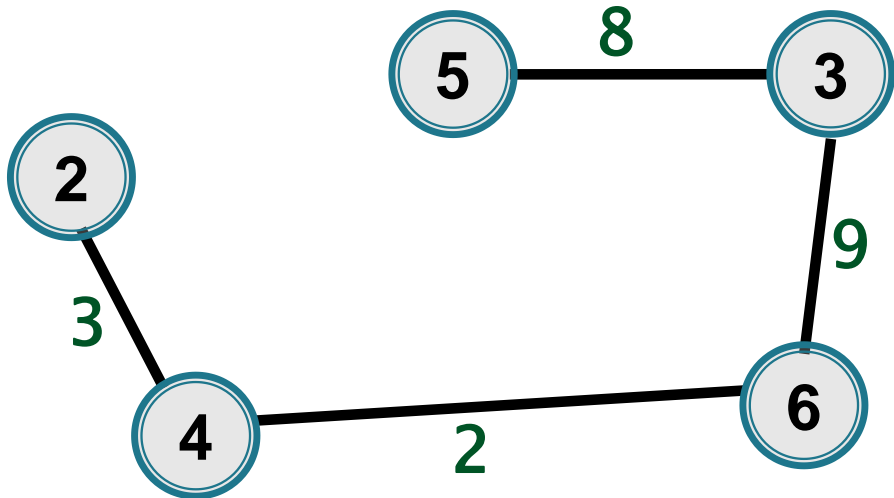
$(3, 6)$   $r = [1, 3, 3, 3, 3, 3]$

$(2, 5)$

$(1, 3)$

$(1, 2)$

$(5, 6)$



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

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

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

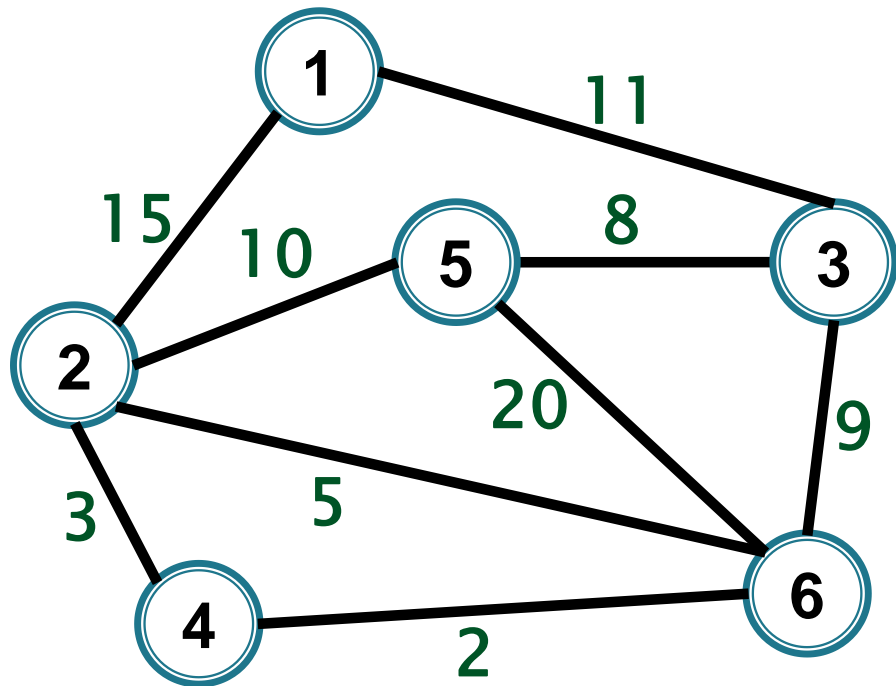
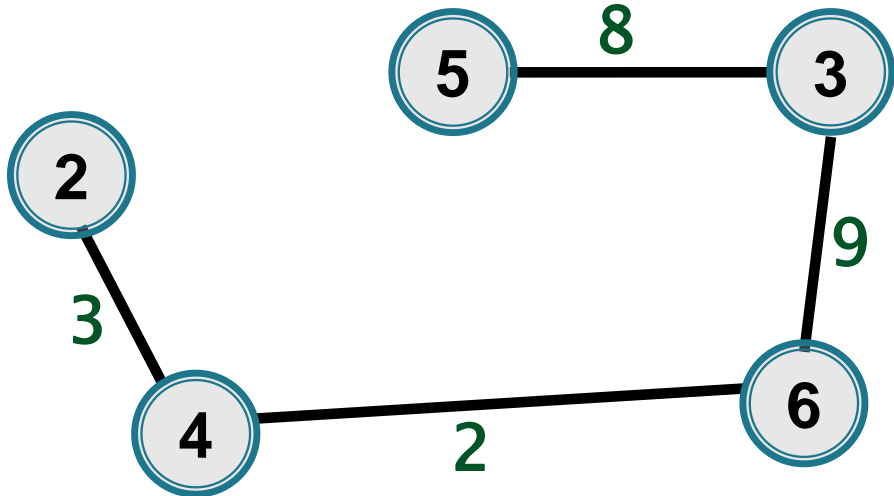
$r(2) = r(6) \rightarrow \text{NU}$

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

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

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





(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

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

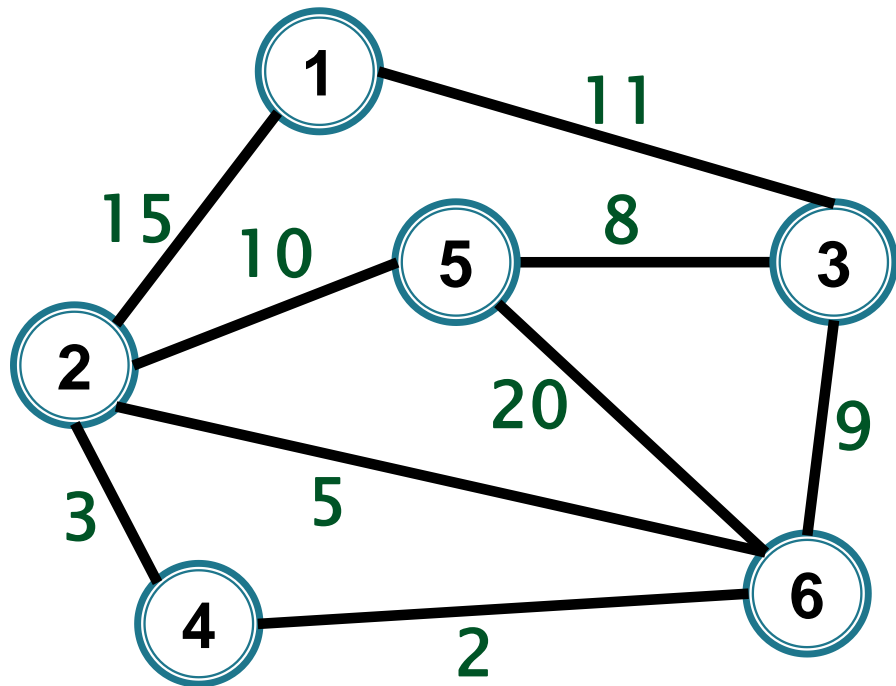
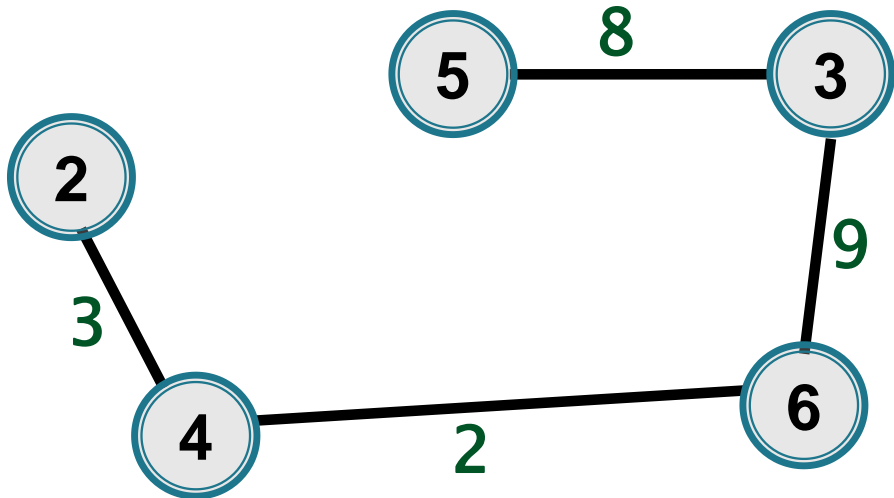
$r = [1, 2, 3, 2, 5, 2]$

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

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

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

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



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

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

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

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

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

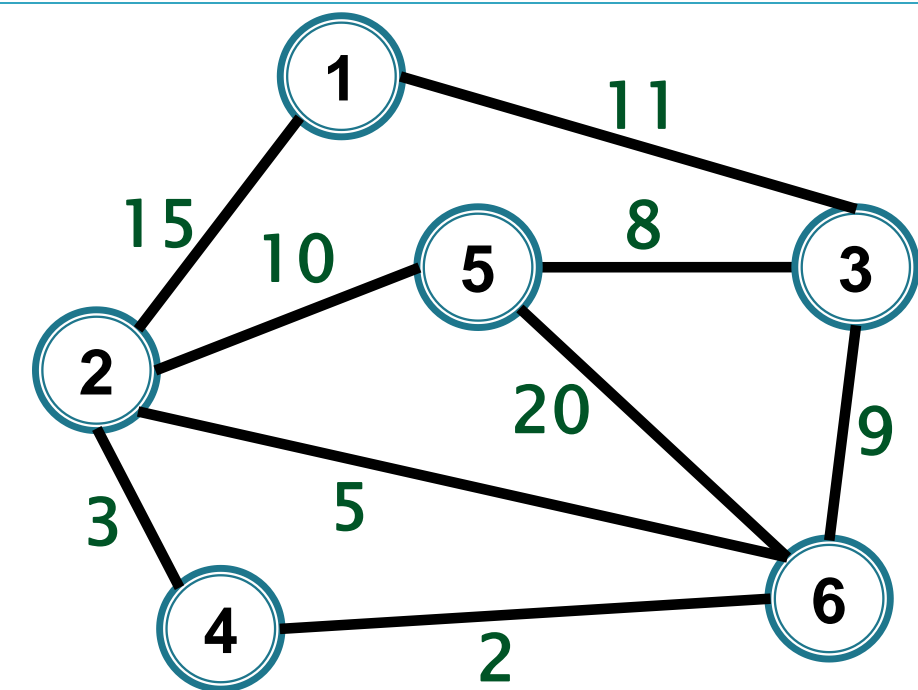
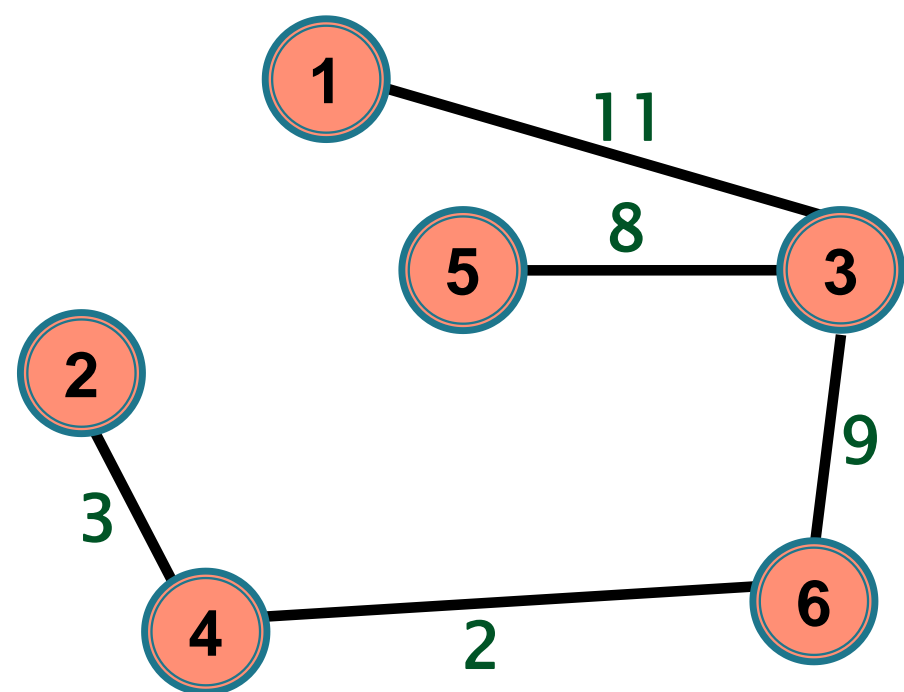
$(3, 6)$   $r = [1, 3, 3, 3, 3, 3]$

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

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

$(1, 2)$

$(5, 6)$



(4,6)

(2,4)

(2,6)

(3,5)

(3,6)

(2,5)

(1,3)

(1,2)

(5,6)

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

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

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

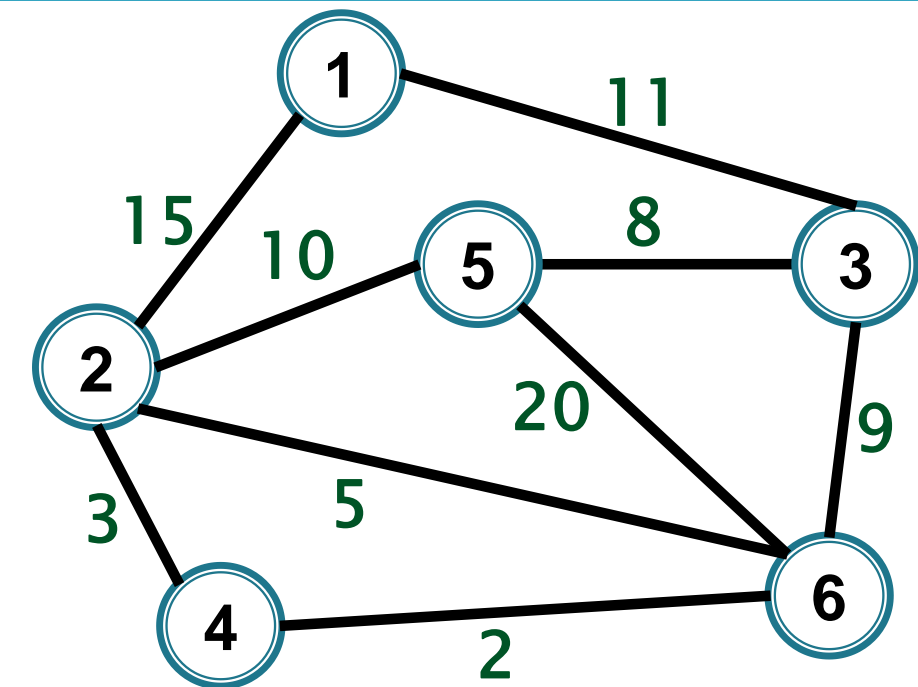
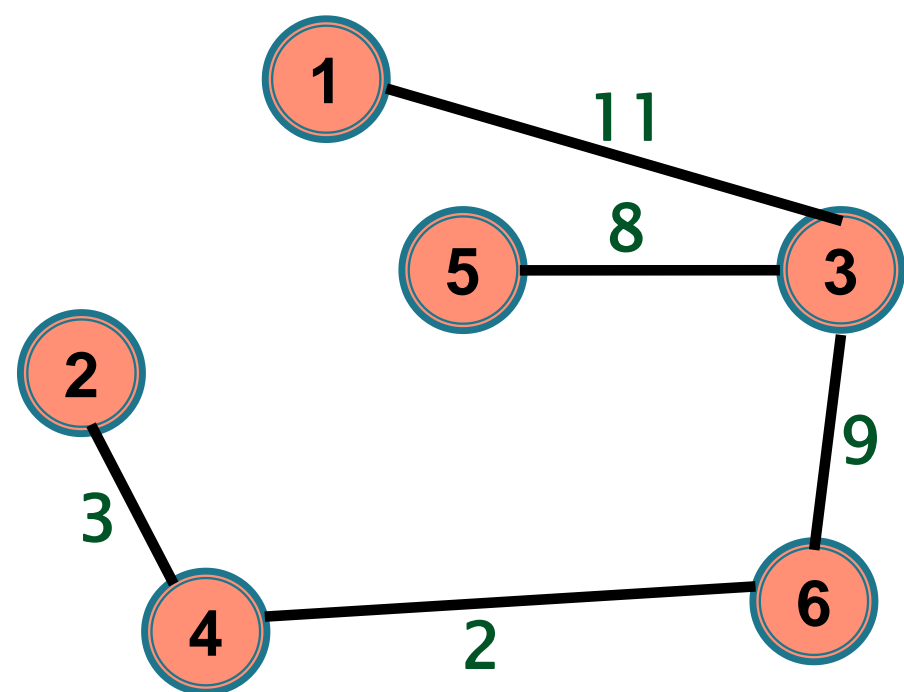
$r(2) = r(6) \rightarrow \text{NU}$

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

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

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

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



(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



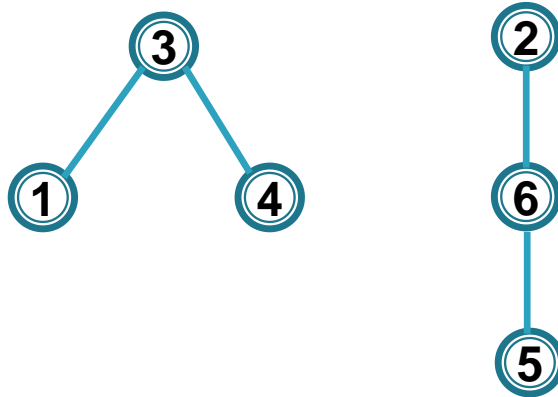
## **Varianta 2 – Structuri pentru mulțimi disjuncte Union/Find**

# Kruskal



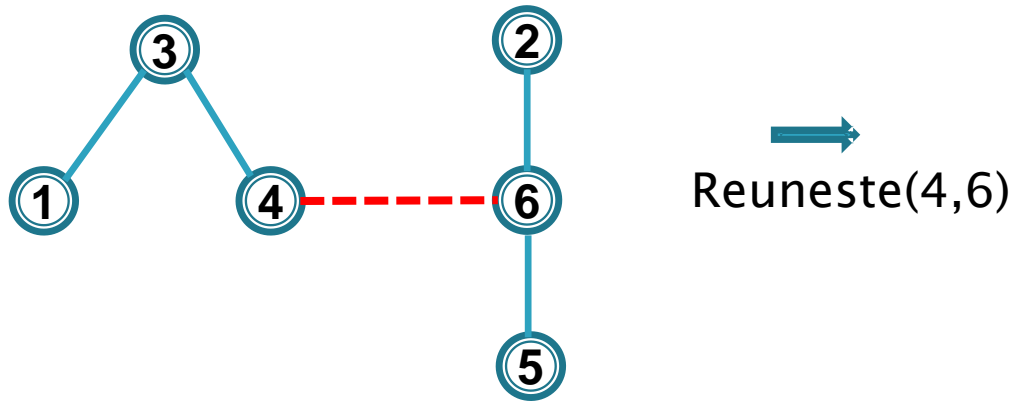
## Varianta 2 – Structuri pentru mulțimi disjuncte Union/Find – arbori

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



# Kruskal

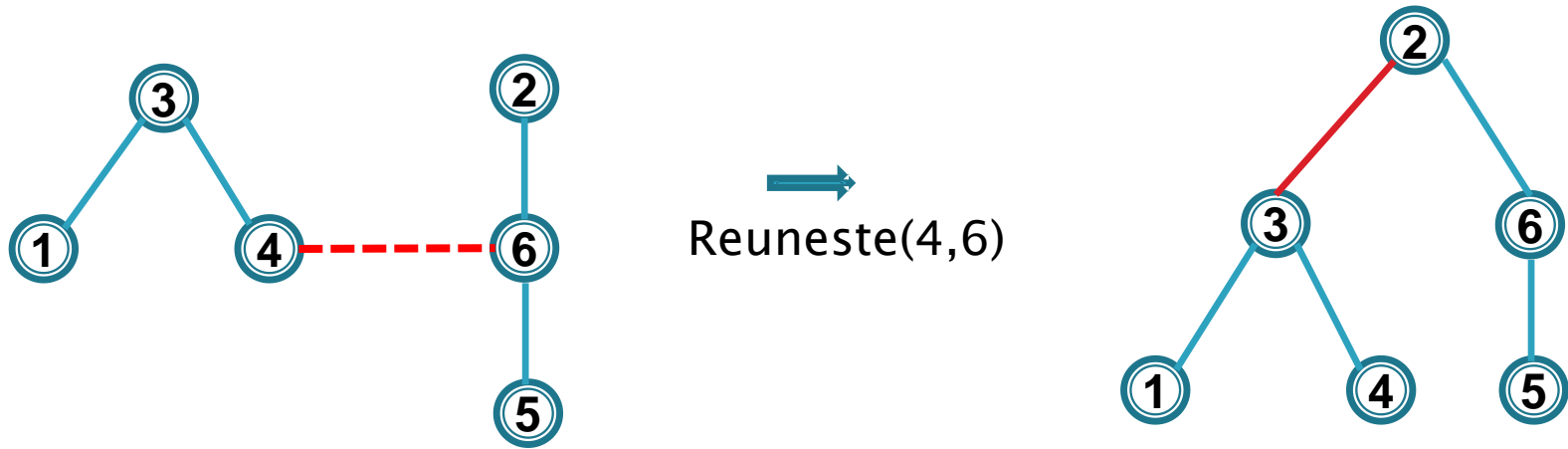
- **Reuniunea** a doi arbori  $\Rightarrow$  rădăcina unui arbore devine fiu al rădăcinii celuilalt arbore



# Kruskal

- Reuniunea se va face în funcție de înălțimea arborilor (reuniune ponderată)

⇒ **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
- **Reunește(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 Inizializare(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])

```

# Kruskal

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

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

```
void Reunește(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 Reunește(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**  $\rightarrow O(n \log n)$

---

$O(m \log n)$



# Kruskal

**Concluzii complexitate –  $O(m \log n)$**

