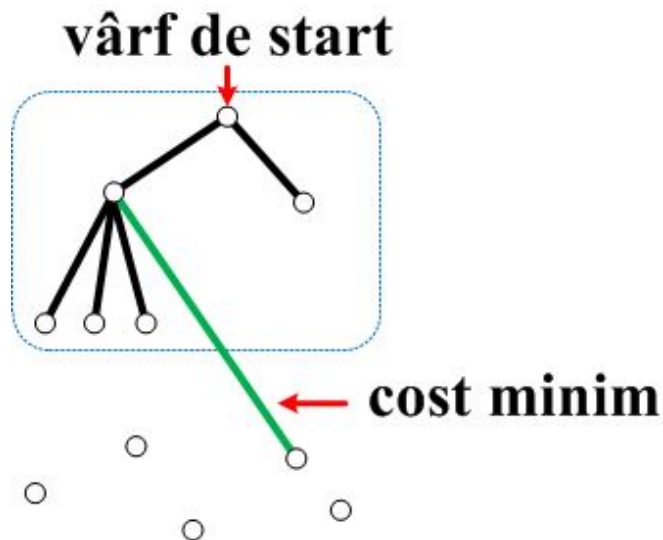


# Algoritmul lui Prim

A dark blue diagonal gradient bar that starts from the bottom-left corner and extends towards the top-right corner, covering the lower half of the slide.

# Algoritmul lui Prim

- Se pornește de la un vârf, care formează arborele inițial
- **La un pas**, este selectată o muchie de cost minim, de la un vârf deja adăugat în arbore, la un vârf neadăugat



# 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)
  - $E(T) = E(T) \cup \{uv\}$

## PRIM

- $s$  - vârful de start
- Inițial,  $T = (\{s\}, \emptyset)$

# 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)
  - $E(T) = E(T) \cup \{uv\}$

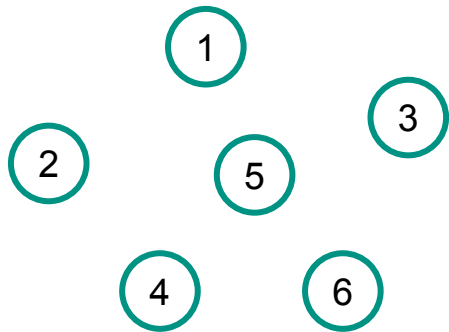
## PRIM

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

# O primă formă a algoritmului

## KRUSKAL

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



- Se încearcă unirea acestor componente prin muchii de cost minim

## PRIM

- **Inițial:** se pornește de la un vârf de start



- Se adaugă, pe rând, câte un vârf la arborele deja construit, folosind muchii de cost minim

# O primă formă a algoritmului

## KRUSKAL

- La un pas:
  - muchiile selectate formează o pădure

## PRIM

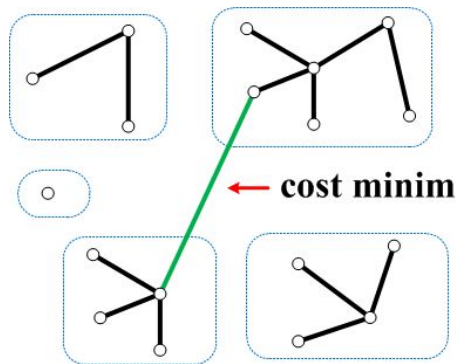
- La un pas:
  - muchiile selectate formează un arbore

# O primă formă a algoritmului

## KRUSKAL

### □ La un pas:

- muchiile selectate formează o **pădure**

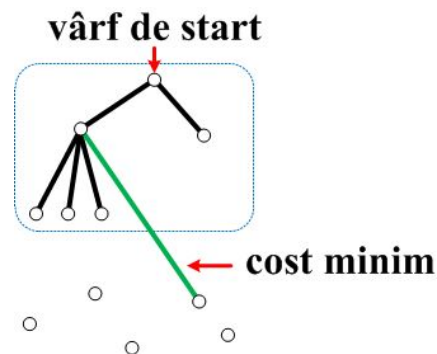


Este selectată o muchie de cost minim, care unește doi arbori din pădurea curentă (două componente conexe).

## PRIM

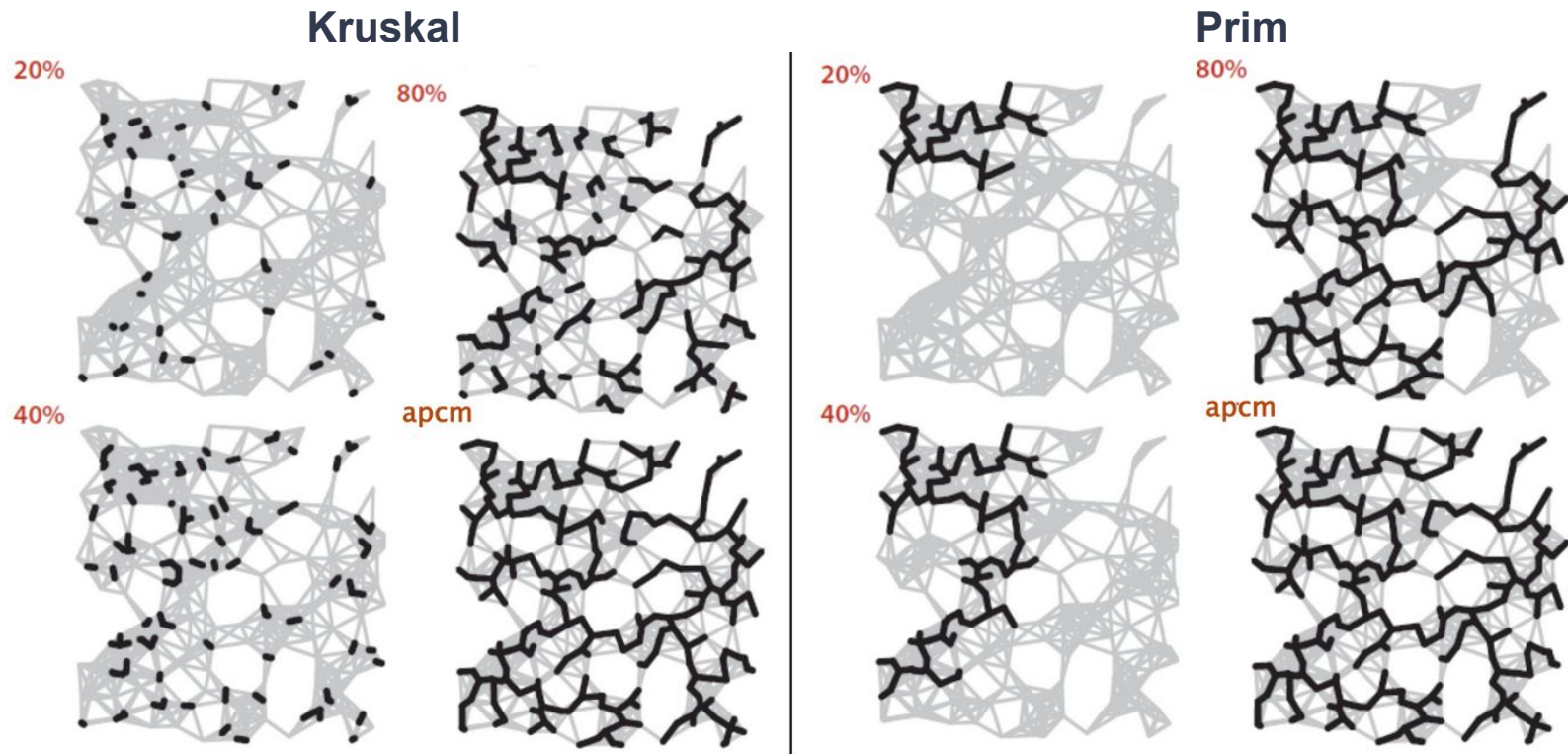
### □ La un pas:

- muchiile selectate formează un **arbore**



Este selectată o muchie de cost minim, care unește un vârf din arbore cu unul care nu este în arbore (neselectat).

# Arbori parțiali de cost minim

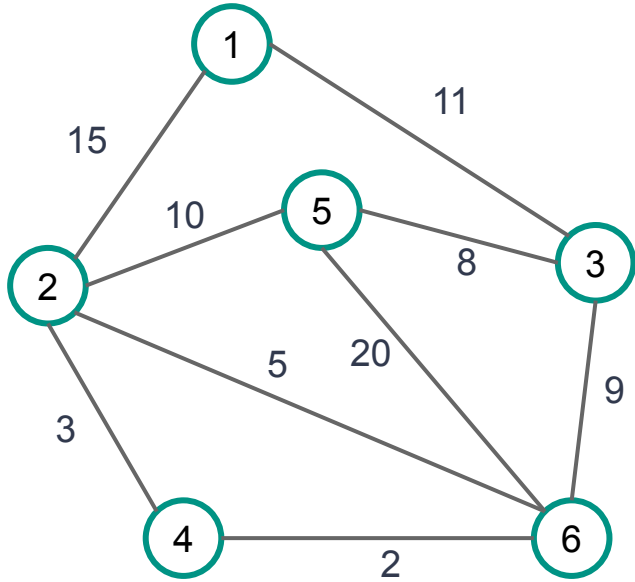


Imagine din

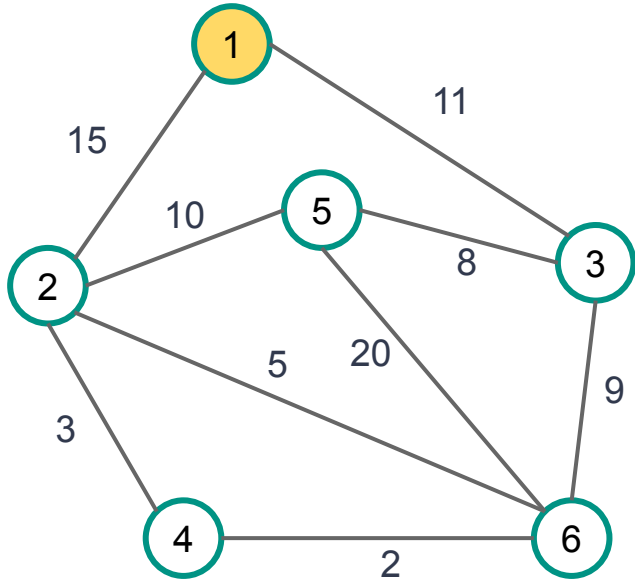
R. Sedgewick, K. Wayne – **Algorithms, 4th edition**, Pearson Education, 2011




# Exemplu

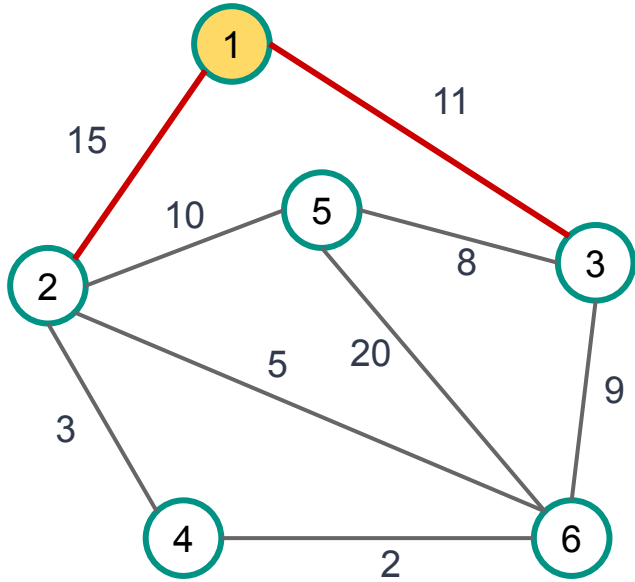


# Exemplu

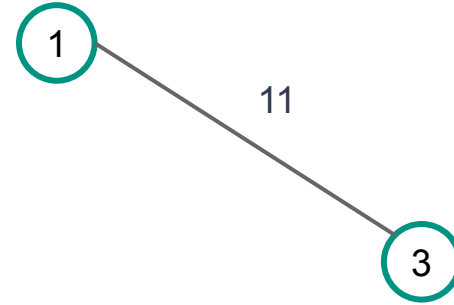
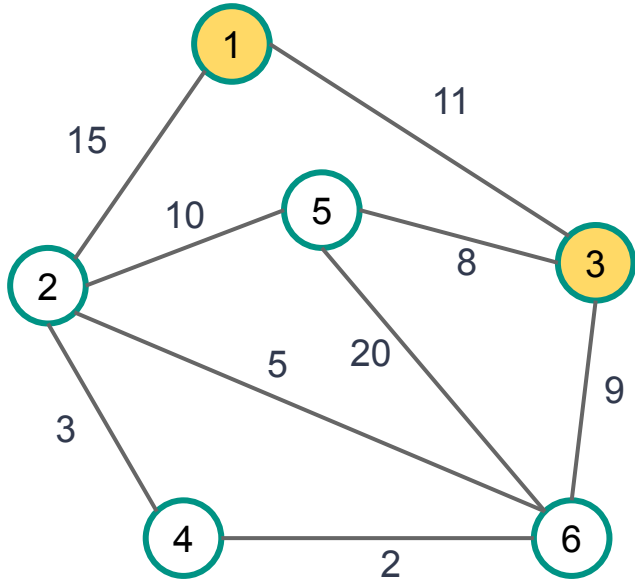


$s =$  

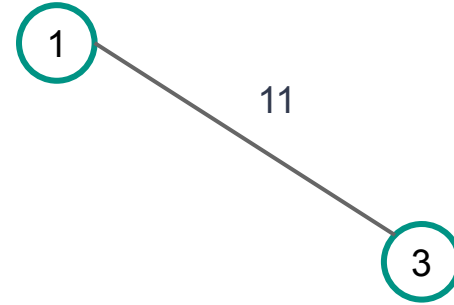
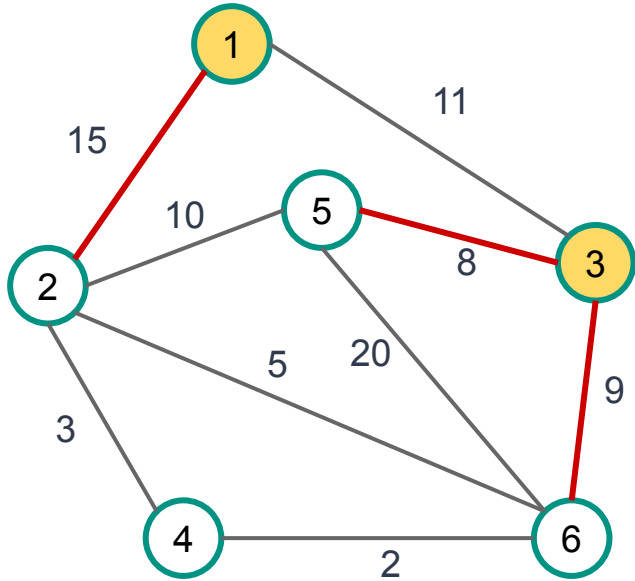
# Exemplu



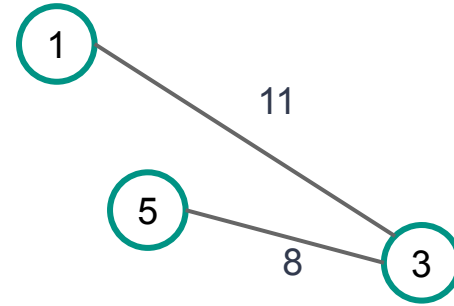
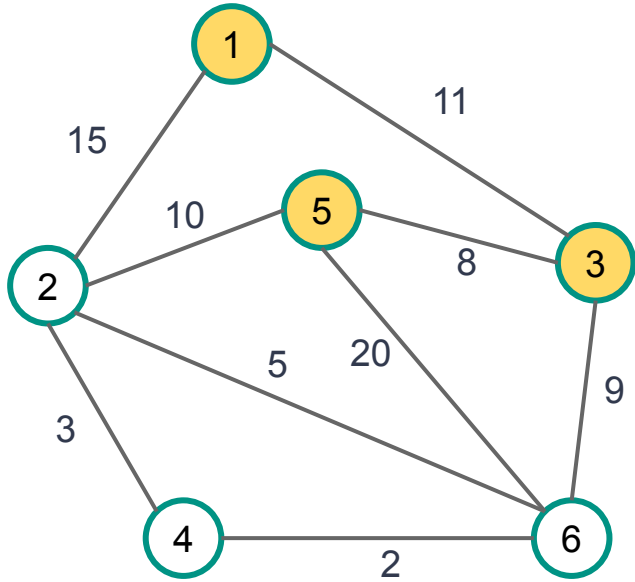
# Exemplu



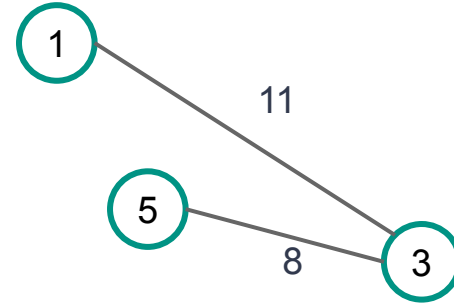
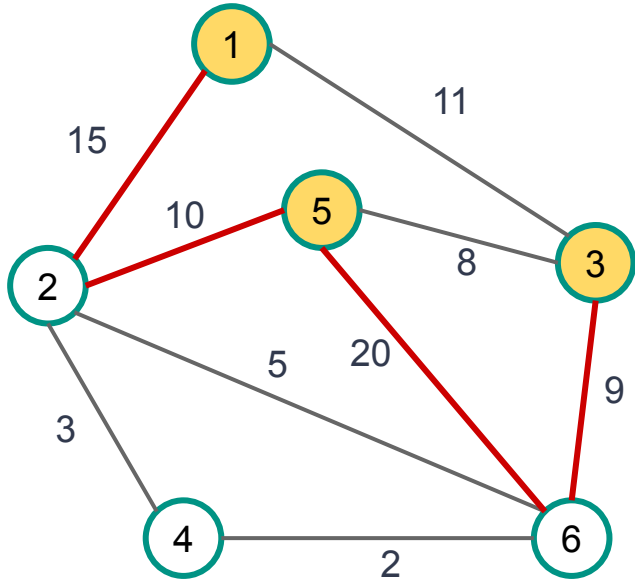
# Exemplu



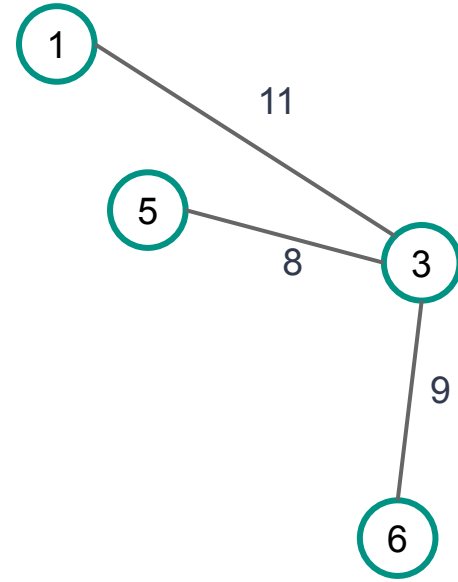
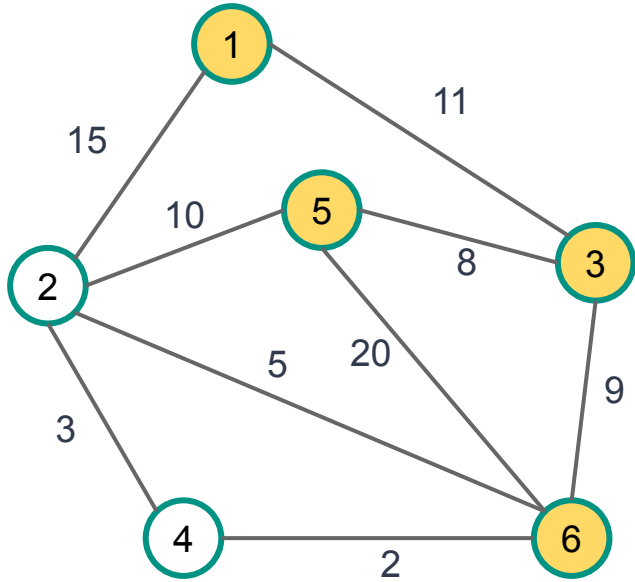
# Exemplu



# Exemplu

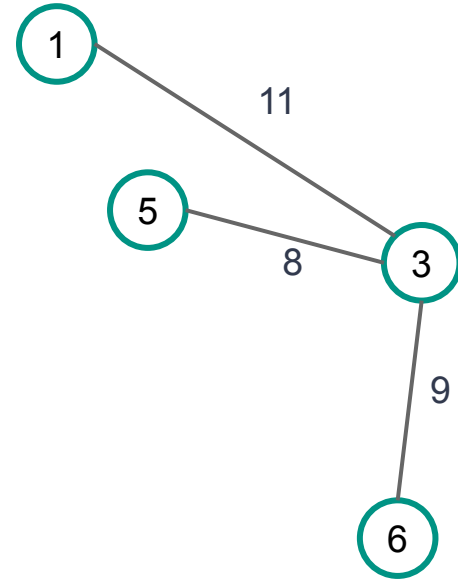
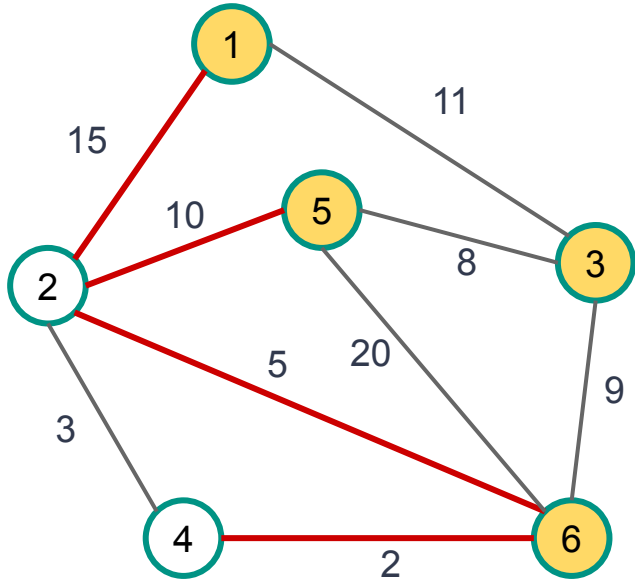


# Exemplu

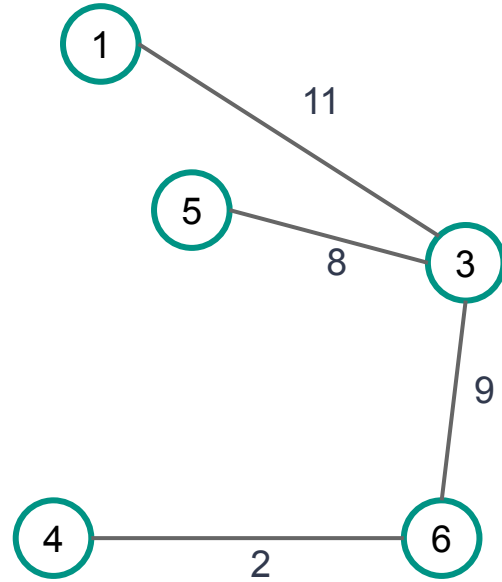
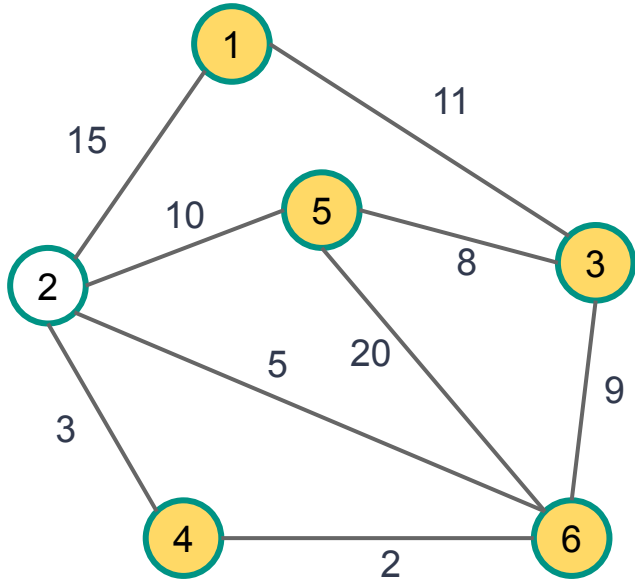




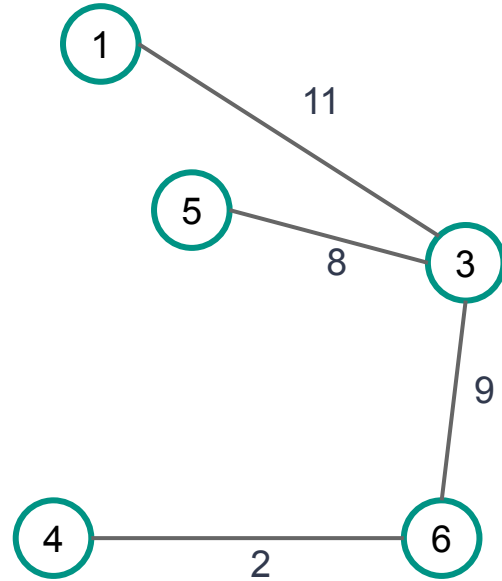
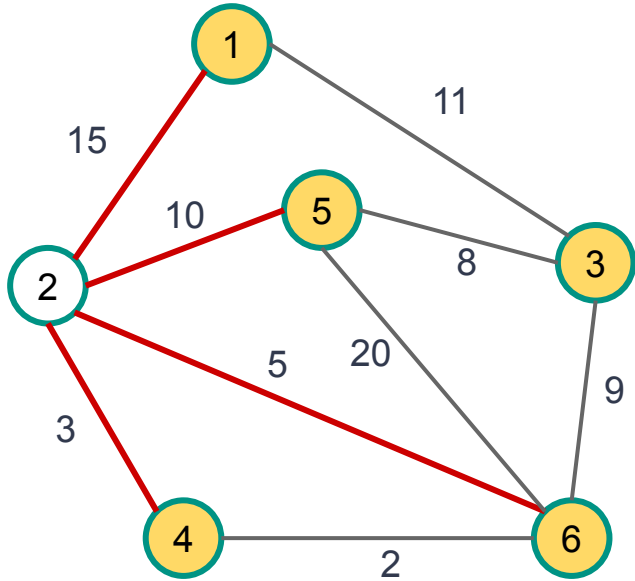
# Exemplu



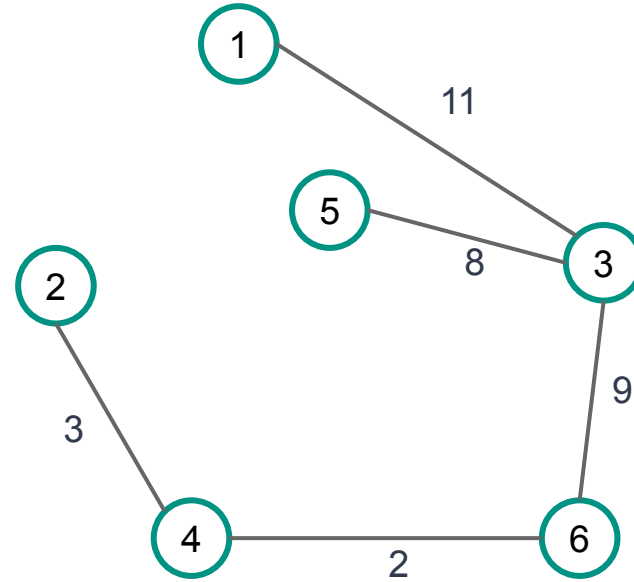
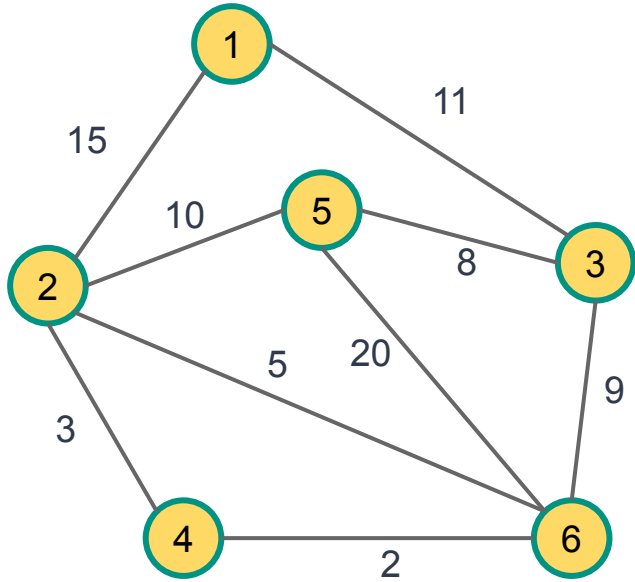
# Exemplu



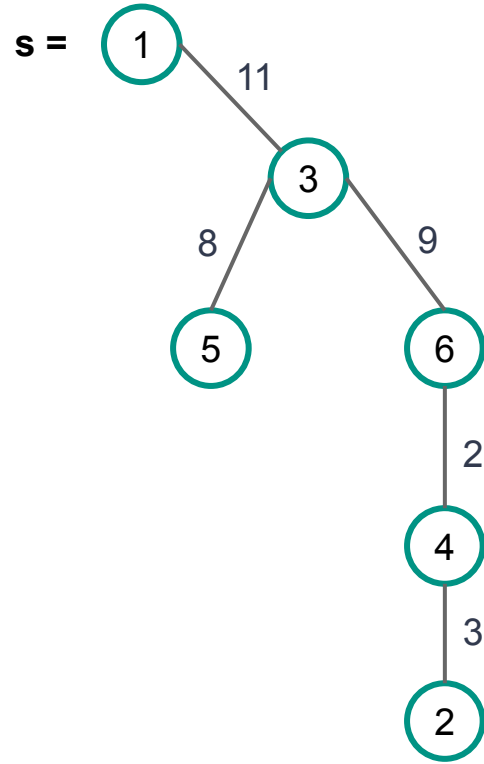
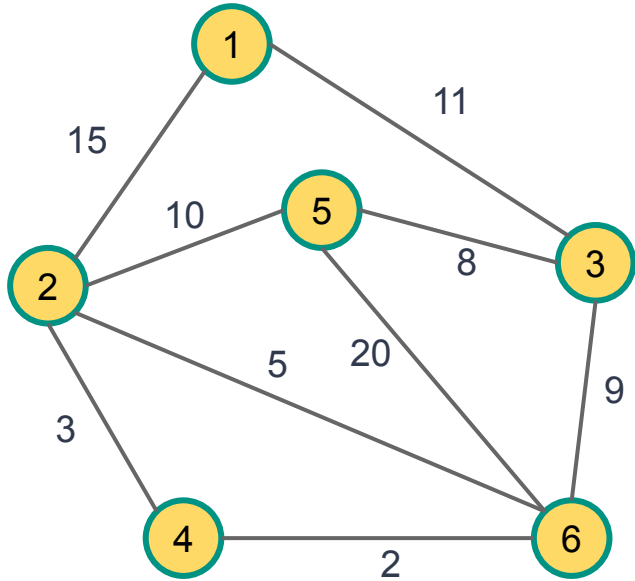
# Exemplu



# Exemplu



# Exemplu



# Prim – Implementare



- Cum alegem *eficient* o muchie de cost minim cu o extremitate selectată (deja în arbore) și cealaltă nu?

# Prim – Implementare



La fiecare pas, parcurgem toate muchiile și o alegem pe cea de cost minim cu o extremitate selectată și una neselectată.

# Prim – Implementare



La fiecare pas, parcurgem toate muchiile și o alegem pe cea de cost minim cu o extremitate selectată și una neselectată.

⇒  **$O(mn)$  - ineficient**





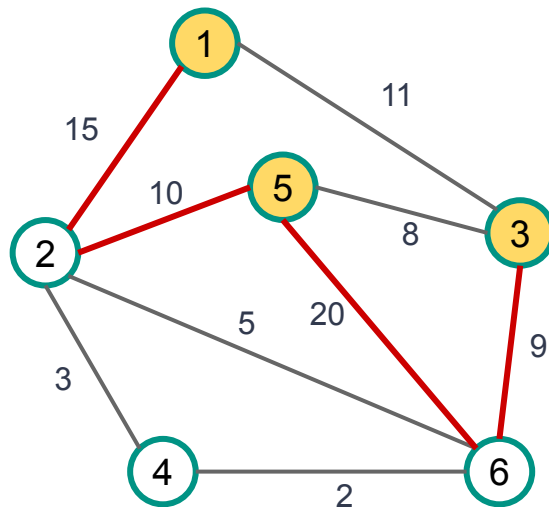
# Prim – Implementare



- Cum evităm să comparăm de fiecare dată toate muchiile cu o extremitate în arbore și cealaltă nu?

## Exemplu:

După ce vârfurile 1 și 5 au fost adăugate în arbore, muchiile **(2, 1)** și **(2, 5)** sunt comparate la fiecare pas, deși  $w(2, 1) > w(2, 5)$ , deci **(2, 1)** **nu va fi selectată niciodată**.



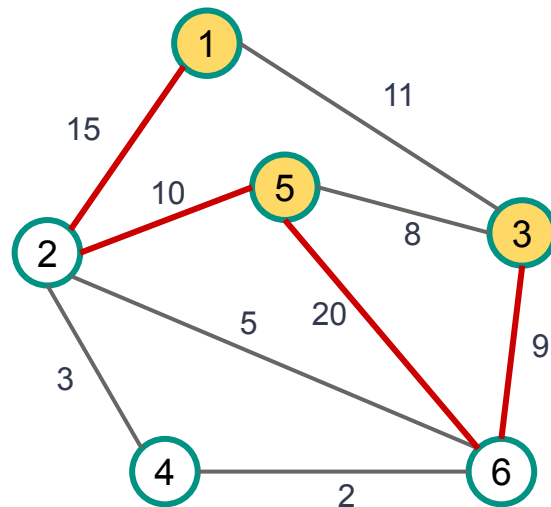
# Prim – Implementare



- Cum evităm să comparăm de fiecare dată toate muchiile cu o extremitate în arbore și cealaltă nu?

Pentru fiecare vârf (neselectat), memorăm **doar muchia de cost minim** care îl unește cu un vârf din arbore (selectat).

pentru vârful 2, va fi memorată, la acest pas, muchia (2, 5)



# Prim – Complexitate

Variante:  $O(n^2)$  /  $O(m \log n)$

- **memorăm, la fiecare pas, pentru fiecare vârf, muchia de cost minim care îl unește de un vârf care este deja în arbore**

sau

- **heap de muchii**

(v. și laborator + seminar + alg. Dijkstra)

# Detalii de implementare

## Algoritmul lui Prim



# Prim – Implementare

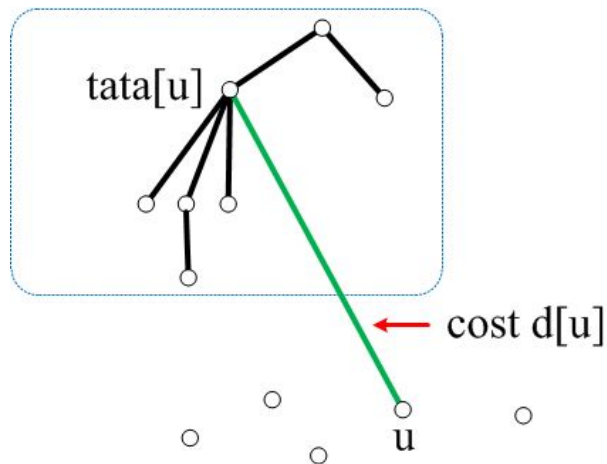
Asociem fiecărui vârf  $u$  următoarele informații (etichete) - pentru a reține **muchia de cost minim care îl unește de un vârf selectat deja în arbore**:



# Prim – Implementare

Asociem fiecărui vârf  $u$  următoarele informații (etichete) - pentru a reține **muchia de cost minim care îl unește de un vârf selectat deja în arbore**:

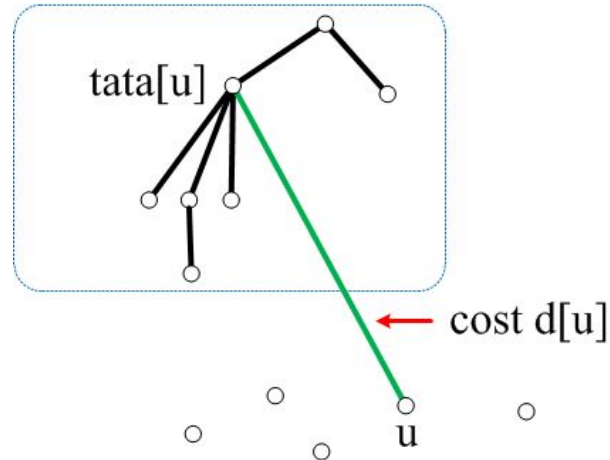
- $d[u]$  = costul minim al unei muchii de la  $u$  la un vârf deja selectat în arbore
- $tata[u]$  = acel vârf din arbore pentru care se realizează minimul



# Prim – Implementare

**Avem:**

- $(u, \text{tata}[u])$  este muchia de cost minim de la  $u$  la un vârf din arbore
- $d[u] = w(u, \text{tata}[u])$



# Prim – Implementare

**Algoritmul se modifică astfel:**

La un pas:

- se alege **un vârf**  $u$  cu **eticheta  $d$  minimă**, care nu este încă în arbore și se adaugă la arbore muchia **(tata[ $u$ ],  $u$ )**
  - **aceasta este muchia de cost minim care unește un vârf neselectat de un vârf din arbore**





# Prim – Implementare

Muchiile arborelui vor fi, în final:

$$(u, \text{tata}[u]), \quad u \neq s$$

# Prim

Notăm  $Q = V(G) - V(T) =$  mulțimea vârfurilor neselectate încă în arbore

# Prim – Algoritm

- **s** - vârful de start
- **inițializează** Q cu V
- **pentru** fiecare  $u \in V$  **execută**
  - $d[u] = \infty$
  - $tata[u] = 0$
- **$d[s] = 0$**

# Prim - Algoritm

- **s** - vârful de start
- **inițializează**  $Q$  cu  $V$
- **pentru** fiecare  $u \in V$  **execută**
  - $d[u] = \infty$
  - $tata[u] = 0$
- **$d[s] = 0$**
- **cât timp**  $Q \neq \emptyset$  **execută** // pentru  $i=1, n$  (suficient  $n-1$ )

# Prim - Algoritm

- **s** - vârful de start
- **inițializează**  $Q$  cu  $V$
- **pentru** fiecare  $u \in V$  **execută**
  - $d[u] = \infty$
  - $tata[u] = 0$
- **$d[s] = 0$**
- **cât timp**  $Q \neq \emptyset$  **execută** // pentru  $i=1, n$  (suficient  $n-1$ )
  - extrage un vârf**  $u \in Q$  **cu eticheta**  $d[u]$  **minimă**

# Prim - Algoritm

- **s** - vârful de start
- **inițializează**  $Q$  cu  $V$
- **pentru** fiecare  $u \in V$  **execută**
  - $d[u] = \infty$
  - $tata[u] = 0$
- **$d[s] = 0$**
- **cât timp**  $Q \neq \emptyset$  **execută** // pentru  $i=1, n$  (suficient  $n-1$ )
  - extrage un vârf  $u \in Q$  cu eticheta  $d[u]$  minimă
  - pentru** fiecare  $uv \in E$  **execută**

# Prim - Algoritm

- **s** - vârful de start
- **inițializează**  $Q$  cu  $V$
- **pentru** fiecare  $u \in V$  **execută**
  - $d[u] = \infty$
  - $tata[u] = 0$
- **$d[s] = 0$**
- **cât timp**  $Q \neq \emptyset$  **execută** // pentru  $i=1, n$  (suficient  $n-1$ )
  - extrage un vârf  $u \in Q$  cu eticheta  $d[u]$  minimă
  - pentru** fiecare  $uv \in E$  **execută**
    - dacă**  $v \in Q$  și  $w(u, v) < d[v]$  **atunci**
      - $d[v] = w(u, v)$
      - $tata[v] = u$



# Prim - Algoritm

- **s** - vârful de start
- **inițializează**  $Q$  cu  $V$
- **pentru** fiecare  $u \in V$  **execută**
  - $d[u] = \infty$
  - $tata[u] = 0$
- **$d[s] = 0$**
- **cât timp**  $Q \neq \emptyset$  **execută** // pentru  $i=1, n$  (suficient  $n-1$ )
  - extrage un vârf  $u \in Q$  cu eticheta  $d[u]$  minimă
  - pentru** fiecare  $uv \in E$  **execută**
    - dacă**  $v \in Q$  și  $w(u, v) < d[v]$  **atunci**
      - $d[v] = w(u, v)$
      - $tata[v] = u$
- **scrie** (**u**, **tata[u]**) pentru  $u \neq s$

# Prim – Complexitate

□ **Inițializări** →

□  **$n$  \* extragere vârf minim** →

□ **actualizare etichete vecini** →

# Prim – Complexitate



- Cum putem memora  $Q$  pentru a determina eficient vârful  $u \in Q$  cu eticheta minimă?

# Prim – Complexitate



- Cum putem memora  $Q$  pentru a determina eficient vârful  $u \in Q$  cu eticheta minimă?
  - vector?
  - heap?

# Prim – Complexitate

**Varianta 1 - Folosim vector de vizitat**

**$Q[u] = 1$ , dacă  $u \in Q$**

**0, altfel**

# Prim – Complexitate

## Varianta 1 - cu vector de vizitat

☐ Inițializări →

☐  $n$  \* extragere vârf minim →

☐ actualizare etichete vecini →

---

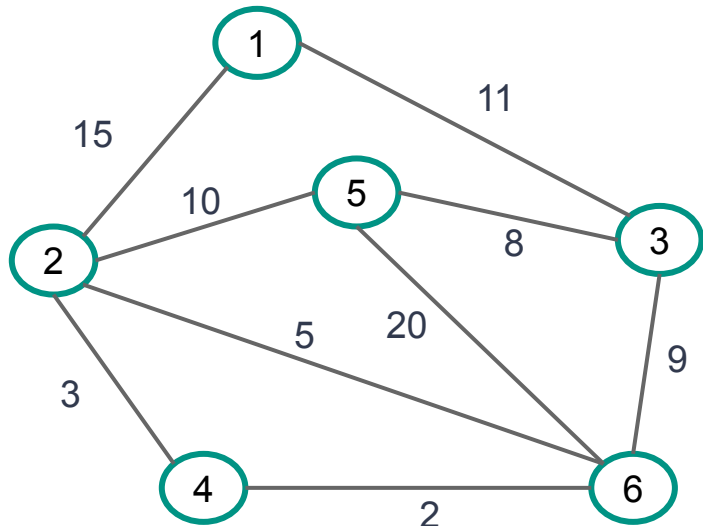
# Prim – Complexitate

## Varianta 1 - cu vector de vizitat

- **Inițializări** →  **$O(n)$**
- **$n$  \* extragere vârf minim** →  **$O(n^2)$**
- **actualizare etichete vecini** →  **$O(m)$**

---

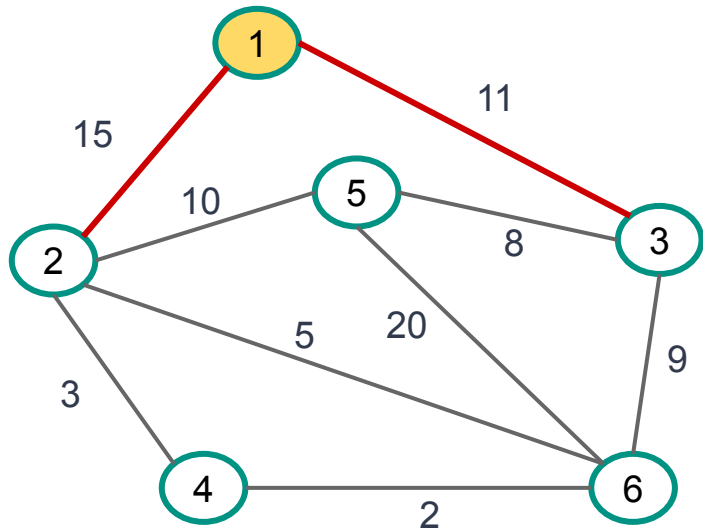
**$O(n^2)$**



$d/tata =$ 

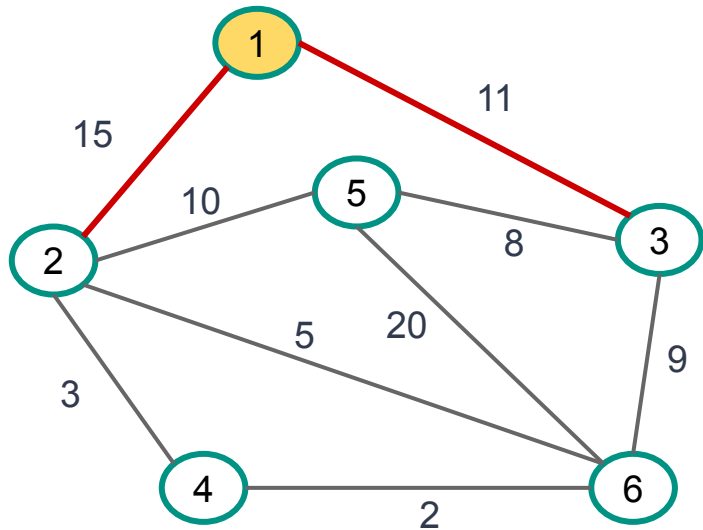
<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
<b>[ 0/0,</b>	<b><math>\infty/0,</math></b>	<b><math>\infty/0,</math></b>	<b><math>\infty/0,</math></b>	<b><math>\infty/0,</math></b>	<b><math>\infty/0</math> ]</b>





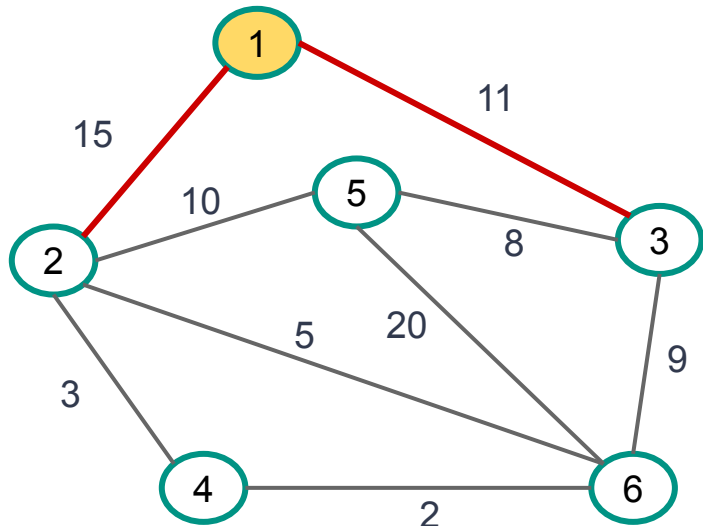

---

	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1						

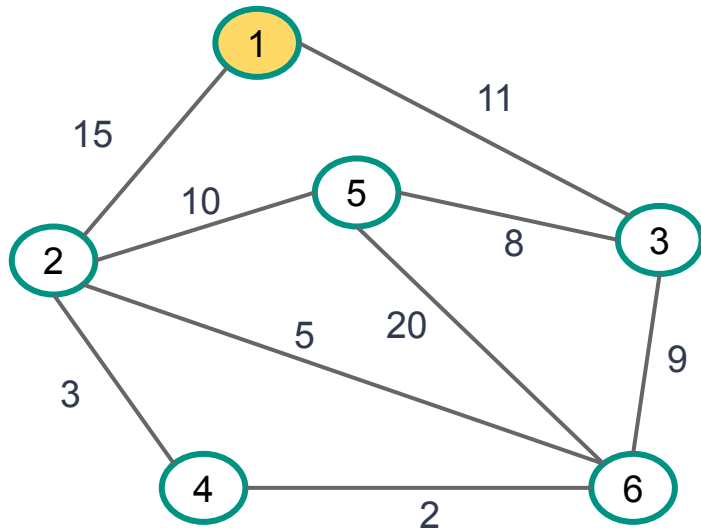


1

	1	2	3	4	5	6
d/tata	[ 0/0,	$\infty$ /0,	$\infty$ /0,	$\infty$ /0,	$\infty$ /0,	$\infty$ /0 ]
Selectăm 1	viz[1] = 1 (vom reprezenta prin - )					

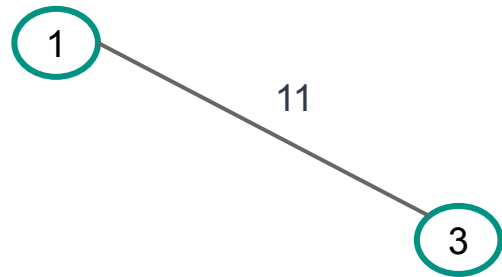
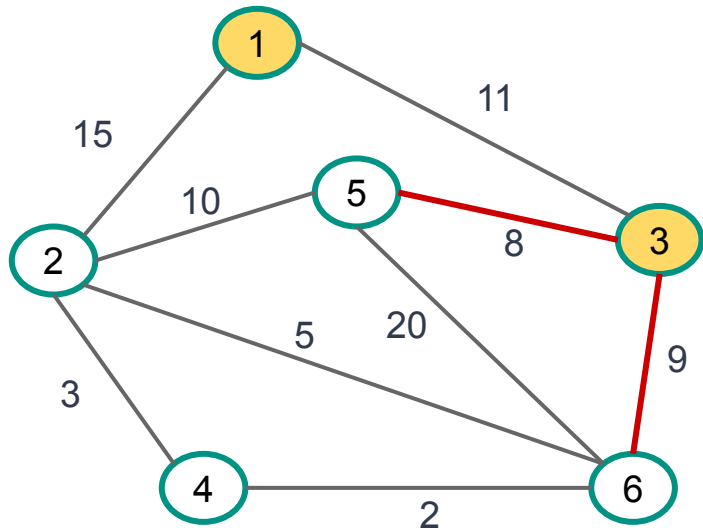


	1	2	3	4	5	6
d/tata	[ 0/0,	$\infty$ /0,	$\infty$ /0,	$\infty$ /0,	$\infty$ /0,	$\infty$ /0 ]
Selectăm 1	actualizăm etichetele vecinilor					

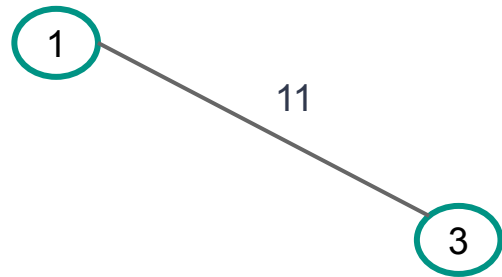
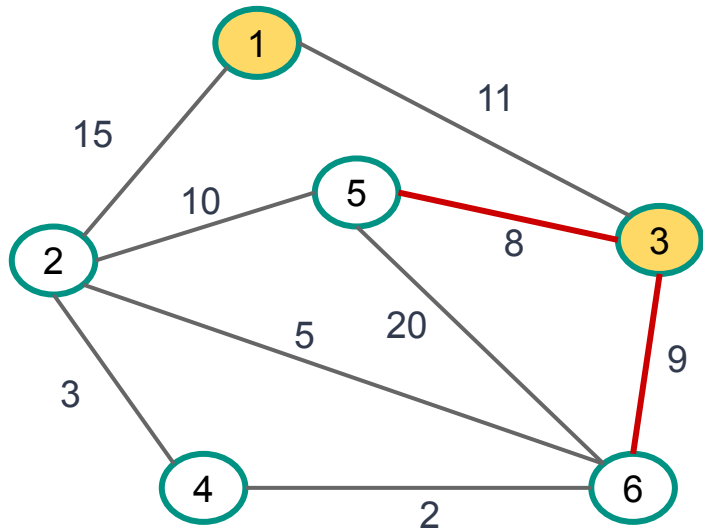


	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ - ,	<b>15/1</b> ,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]

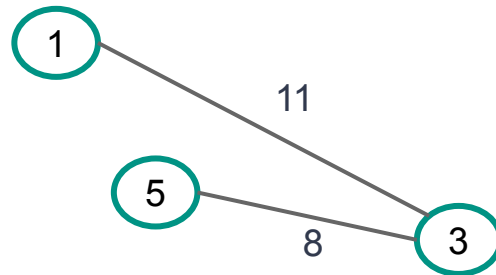
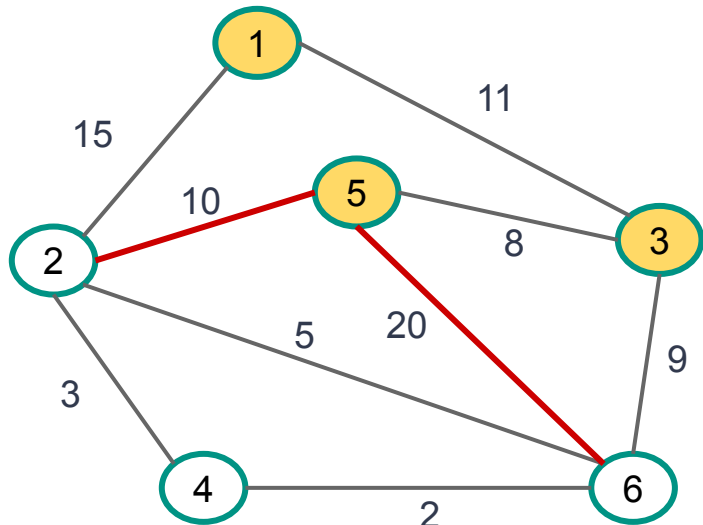
Pentru vârful 2, este memorată muchia (2,1) de cost 15, mai exact costul muchiei în **d[2]** și cealaltă extremitate în **tata[2]**



	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ - ,	15/1,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 3						



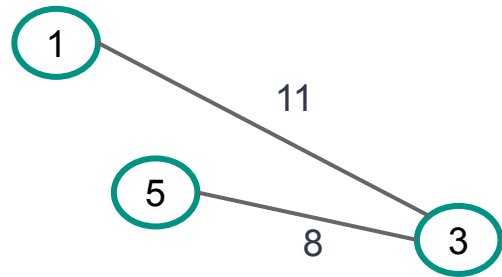
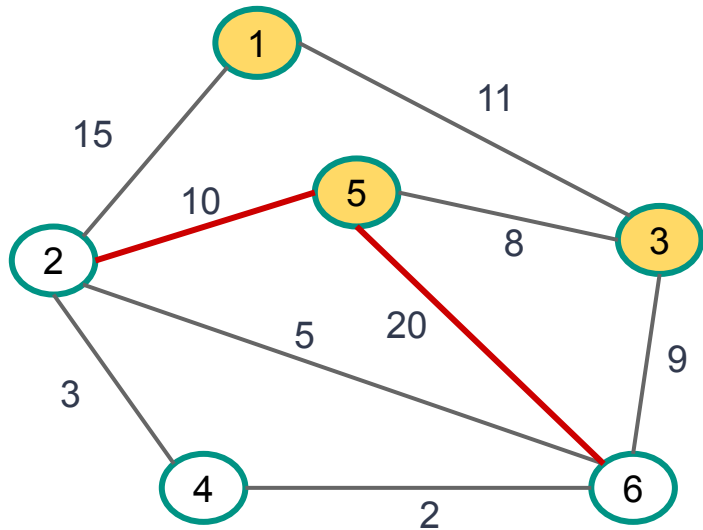
	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ -,	15/1,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 3	[ -,	15/1,	-,	$\infty/0$ ,	<b>8/3</b> ,	<b>9/3</b> ]



	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ -,	15/1,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 3	[ -,	15/1,	-,	$\infty/0$ ,	<b>8/3</b> ,	9/3 ]
Selectăm 5						

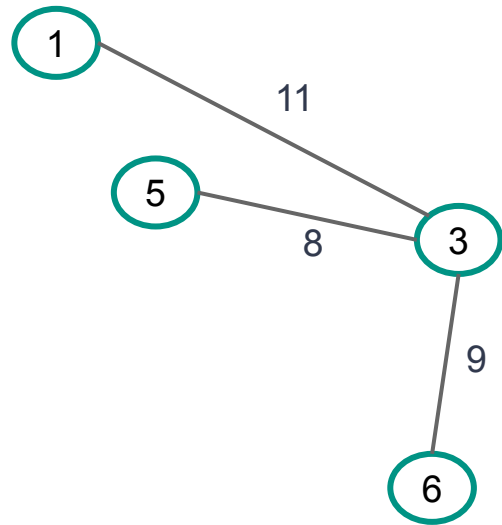
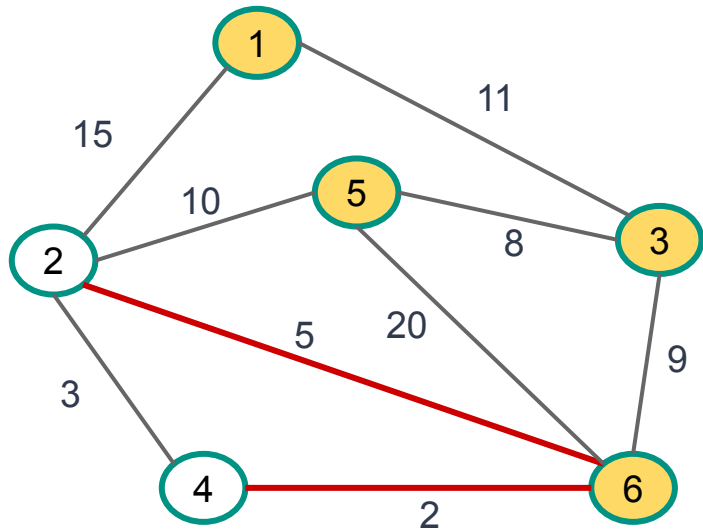
$d[2] = \min \{ d[2], w(2, 5) \}$

$d[6] = \min \{ d[6], w(6, 5) \}$

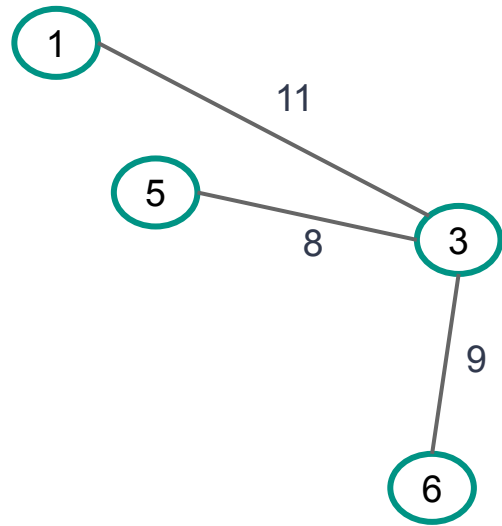
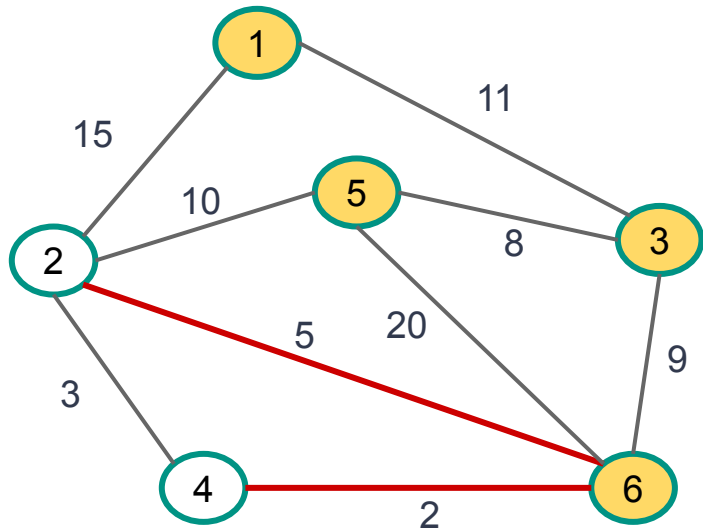


	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ -,	15/1,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 3	[ -,	15/1,	-,	$\infty/0$ ,	<b>8/3</b> ,	9/3 ]
Selectăm 5	[ -,	<b>10/5</b> ,	-,	$\infty/0$ ,	-,	<b>9/3</b> ]

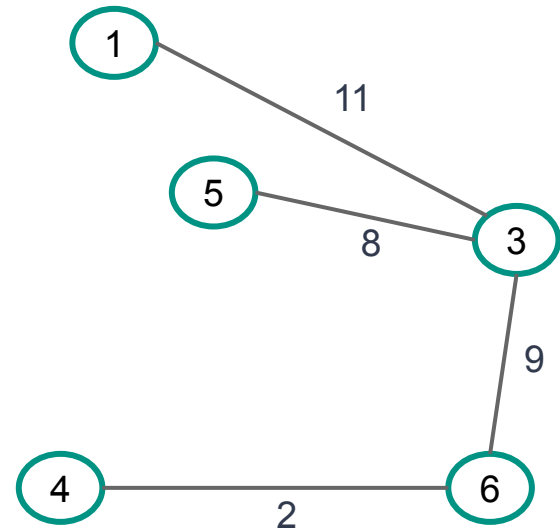
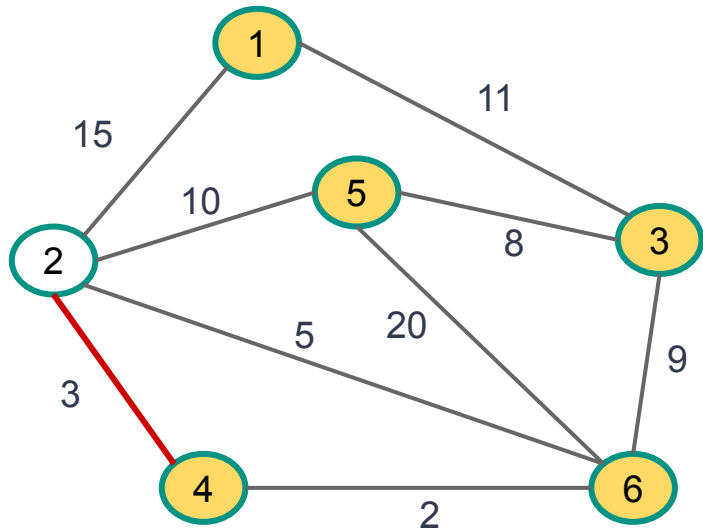




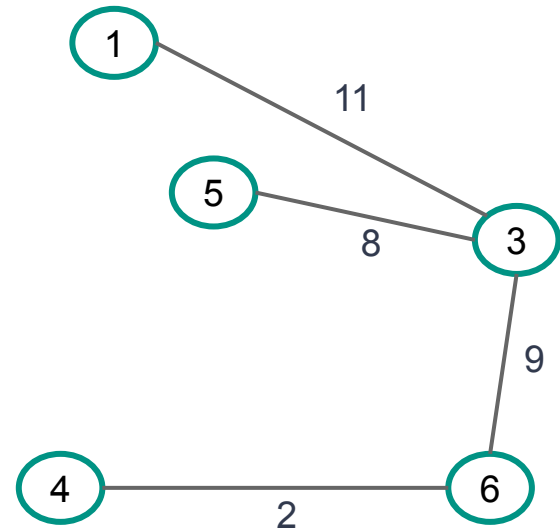
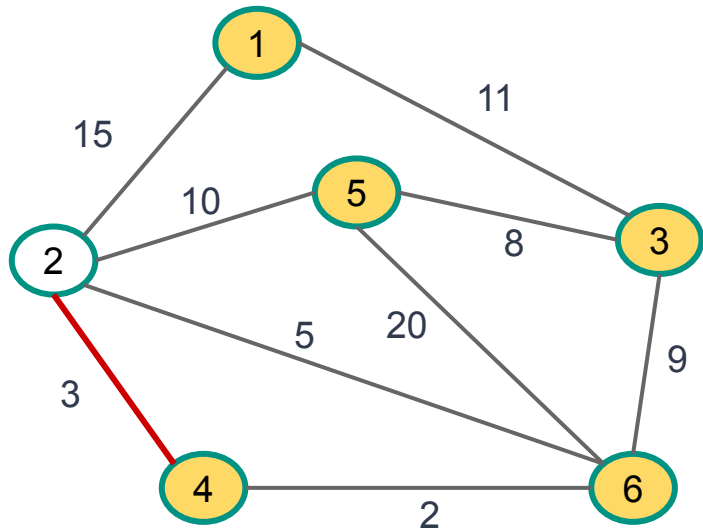
	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ -,	15/1,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 3	[ -,	15/1,	-,	$\infty/0$ ,	<b>8/3</b> ,	9/3 ]
Selectăm 5	[ -,	10/5,	-,	$\infty/0$ ,	-,	<b>9/3</b> ]
Selectăm 6						



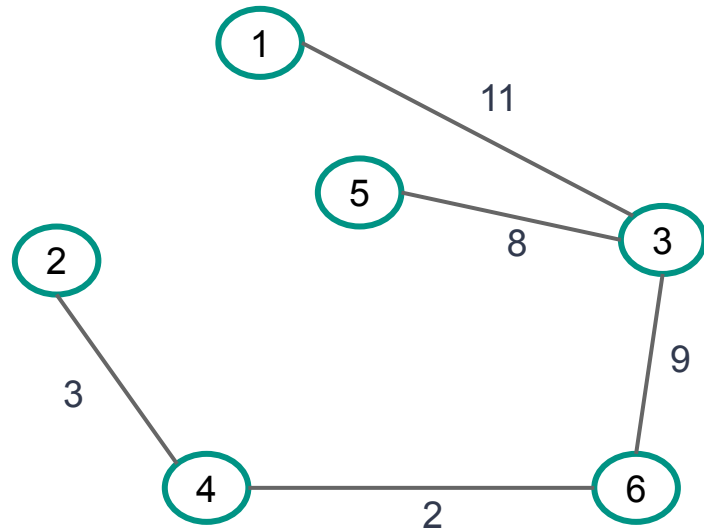
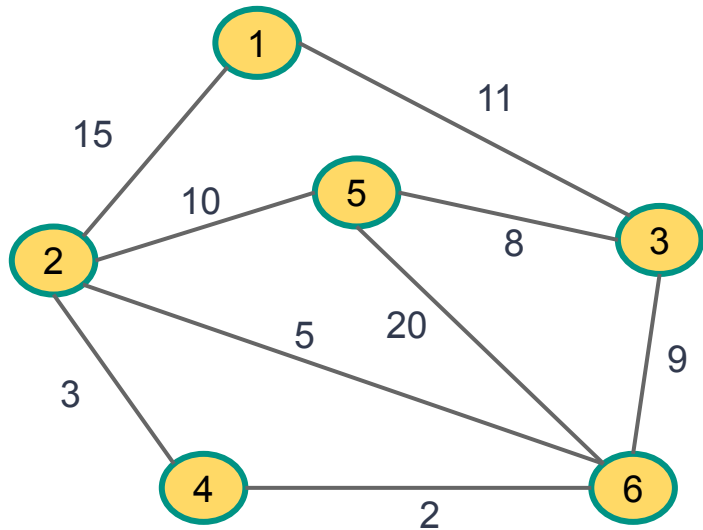
	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ -,	15/1,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 3	[ -,	15/1,	-,	$\infty/0$ ,	<b>8/3</b> ,	9/3 ]
Selectăm 5	[ -,	10/5,	-,	$\infty/0$ ,	-,	<b>9/3</b> ]
Selectăm 6	[ -,	<b>5/6</b> ,	-,	<b>2/6</b> ,	-,	- ]



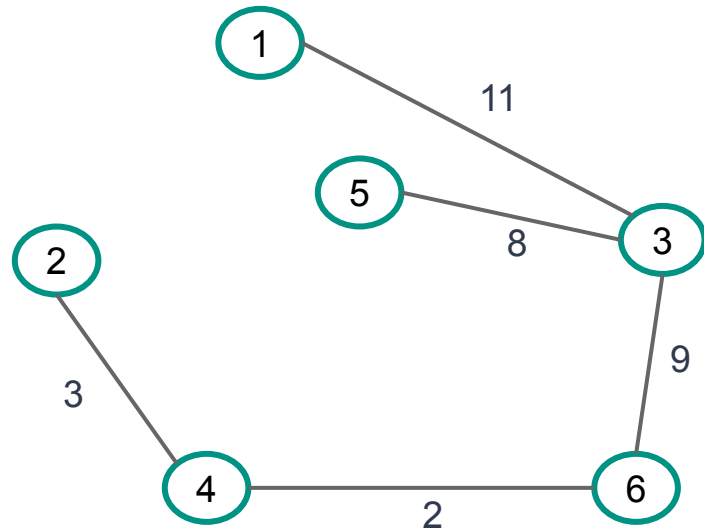
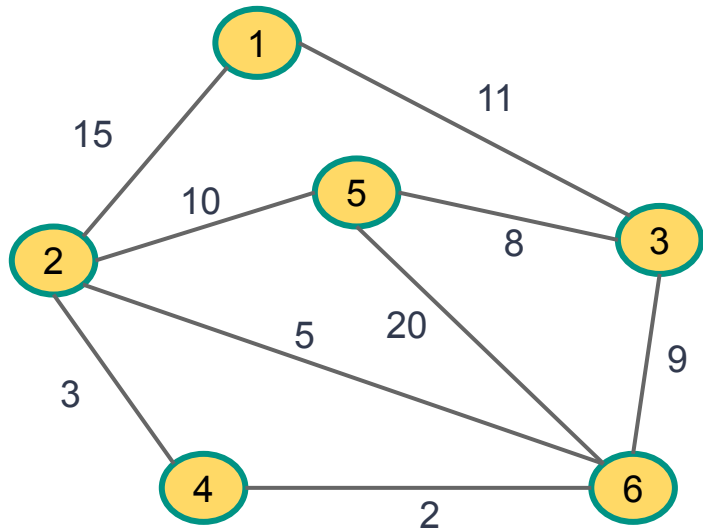
	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ -,	15/1,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 3	[ -,	15/1,	-,	$\infty/0$ ,	<b>8/3</b> ,	9/3 ]
Selectăm 5	[ -,	10/5,	-,	$\infty/0$ ,	-,	<b>9/3</b> ]
Selectăm 6	[ -,	5/6,	-,	<b>2/6</b> ,	-,	- ]
Selectăm 4						



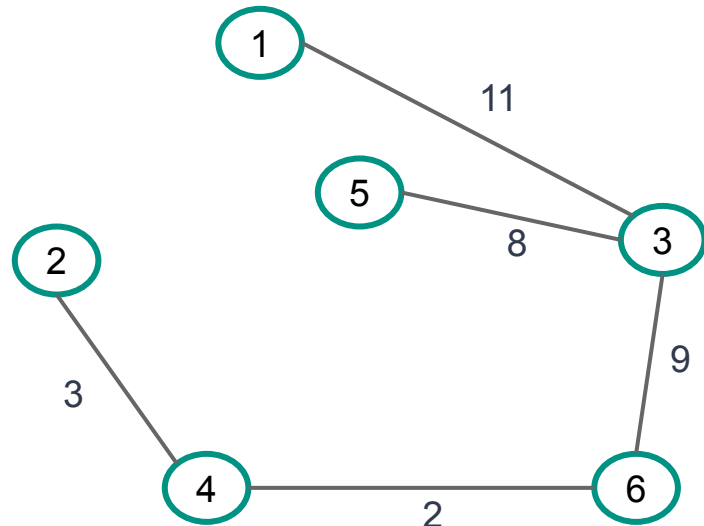
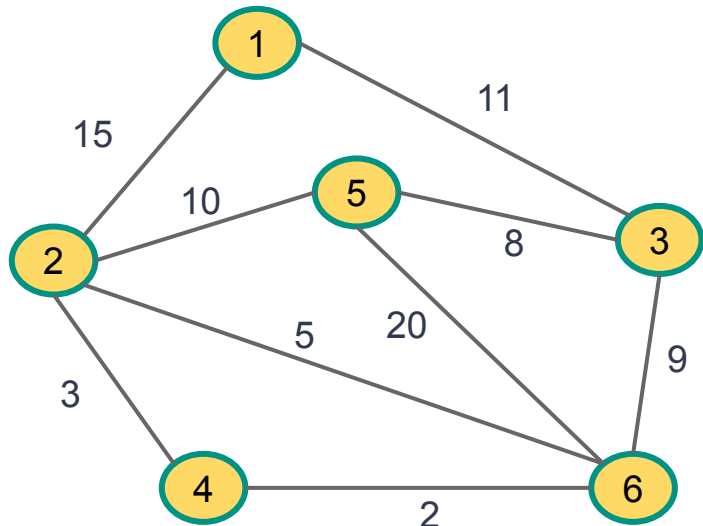
	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ -,	15/1,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 3	[ -,	15/1,	-,	$\infty/0$ ,	<b>8/3</b> ,	9/3 ]
Selectăm 5	[ -,	10/5,	-,	$\infty/0$ ,	-,	<b>9/3</b> ]
Selectăm 6	[ -,	5/6,	-,	<b>2/6</b> ,	-,	- ]
Selectăm 4	[ -,	<b>3/4</b> ,	-,	-,	-,	- ]



	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ -,	15/1,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 3	[ -,	15/1,	-,	$\infty/0$ ,	<b>8/3</b> ,	9/3 ]
Selectăm 5	[ -,	10/5,	-,	$\infty/0$ ,	-,	<b>9/3</b> ]
Selectăm 6	[ -,	5/6,	-,	<b>2/6</b> ,	-,	- ]
Selectăm 4	[ -,	<b>3/4</b> ,	-,	-,	-,	- ]
Selectăm 2						



	1	2	3	4	5	6
d/tata	[ <b>0/0</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 1	[ -,	15/1,	<b>11/1</b> ,	$\infty/0$ ,	$\infty/0$ ,	$\infty/0$ ]
Selectăm 3	[ -,	15/1,	-,	$\infty/0$ ,	<b>8/3</b> ,	9/3 ]
Selectăm 5	[ -,	10/5,	-,	$\infty/0$ ,	-,	<b>9/3</b> ]
Selectăm 6	[ -,	5/6,	-,	<b>2/6</b> ,	-,	- ]
Selectăm 4	[ -,	<b>3/4</b> ,	-,	-,	-,	- ]
Selectăm 2	[ -,	-,	-,	-,	-,	- ]



d/tata  
 FINAL

1	2	3	4	5	6
[ 0/0,	3/4,	11/1,	2/6,	8/3,	9/3 ]

Vectorul tata  $\Rightarrow$  muchiile arborelui  $(u, \text{tata}[u]), u \neq s$

# Prim – Complexitate

**Varianta 2** - cu memorarea vârfurilor într-un min-heap  $Q$  (min-ansamblu)

☐ **Inițializare  $Q$**  →

☐  **$n$  \* extragere vârf minim** →

☐ **actualizare etichete vecini** →

---



# Prim – Algoritm

Prim( $G, w, s$ )

**pentru** fiecare  $u \in V$  **executa**

$d[u] = \infty$ ;  $tata[u] = 0$

$d[s] = 0$

**initializeaza**  $Q$  cu  $V$

**cat timp**  $Q \neq \emptyset$  **executa**

$u =$  extrage varf cu eticheta minima din  $Q$

**pentru** fiecare  $v$  adiacent cu  $u$  **executa**

**daca**  $v \in Q$  si  $w(u, v) < d[v]$  **atunci**

$d[v] = w(u, v)$

$tata[v] = u$

???

**scrie** ( $u, tata[u]$ ), pentru  $u \neq s$

# Prim – Algoritm

Prim( $G, w, s$ )

**pentru** fiecare  $u \in V$  **executa**

$d[u] = \infty$ ;  $tata[u] = \emptyset$

$d[s] = 0$

**initializeaza**  $Q$  cu  $V$

**cat timp**  $Q \neq \emptyset$  **executa**

$u =$  extrage varf cu eticheta minima din  $Q$

**pentru** fiecare  $v$  adiacent cu  $u$  **executa**

**daca**  $v \in Q$  si  $w(u, v) < d[v]$  **atunci**

$d[v] = w(u, v)$

$tata[v] = u$

**// actualizeaza  $Q$  - pentru  $Q$  heap**

**scrie** ( $u, tata[u]$ ), pentru  $u \neq s$

# Prim – Complexitate

**Varianta 2** - cu memorarea vârfurilor într-un min-heap  $Q$  (min-ansamblu)

□ **Inițializare  $Q$**  →  **$O(n)$**

□  **$n$  \* extragere vârf minim** →  **$O(n \log n)$**

□ **actualizare etichete vecini** →  **$O(m \log n)$**

---

**$O(m \log n)$**

# Prim – Complexitate

**Observație** - Dacă graful este complet (spre exemplu, dacă toate punctele se pot conecta și distanța dintre puncte este distanța euclidiană), atunci  $m = n(n-1) / 2$  este de ordin  $n^2$

⇒  $O(n^2)$  mai eficient

# Algoritmi bazați pe eliminare de muchii



**Temă:** Care dintre următorii algoritmi determină corect un arbore parțial de cost minim? Justificați. Pentru fiecare algoritm corect, precizați ce complexitate are.

1.  $T \leftarrow G$   
cât timp  $T$  conține cicluri execută  
    alege  $e$  o muchie de cost minim care este conținută într-un ciclu din  $T$   
     $T \leftarrow T - e$
2.  $T \leftarrow G$   
cât timp  $T$  conține cicluri execută  
    alege  $C$  un ciclu oarecare din  $T$  și fie  $e$  muchia de cost maxim din  $C$   
     $T \leftarrow T - e$

# Corectitudine Algoritmii Kruskal + Prim



# Corectitudine Kruskal + Prim



- ☐ Cei doi algoritmi determină corect un apcm?  
**Chiar dacă muchiile au și costuri negative?**
- ☐ Costul arborelui obținut de algoritmul lui Prim nu depinde de vârful de start?

# Corectitudine Kruskal + Prim

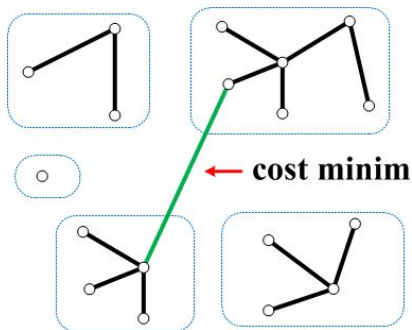
- Fie  $A \subseteq E$  o mulțime de muchii
- Notăm  $A \subseteq \text{apcm} \Leftrightarrow \exists T$  un apcm astfel încât  $A \subseteq E(T)$



# Amintim

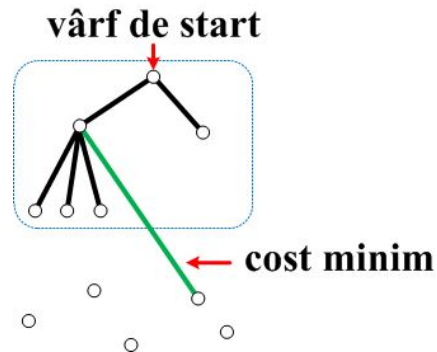
## KRUSKAL

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



## PRIM

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



# Corectitudine Kruskal + Prim

Atât algoritmul lui Kruskal, cât și cel al lui Prim, funcționează după următoarea schemă:

- $A = \emptyset$  (mulțimea muchiilor selectate în arborele construit)
- pentru  $i = 1, n-1$  execută
  - alege o muchie  $e$  astfel încât  $A \cup \{e\} \subseteq \text{apcm}$
  - $A = A \cup \{e\}$
- returnează  $T = (V, A)$

# Corectitudine Kruskal + Prim

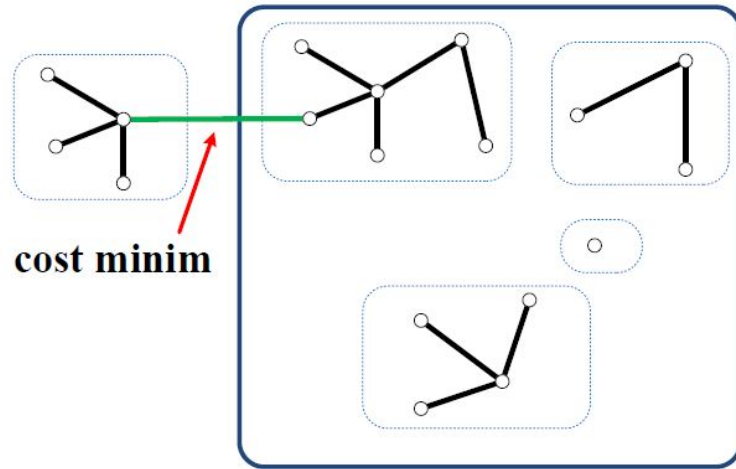
Vom demonstra cu **criteriu de alegere a muchiei e** la un pas astfel încât:

$$A \subseteq \text{apcm} \Rightarrow A \cup \{e\} \subseteq \text{apcm}$$

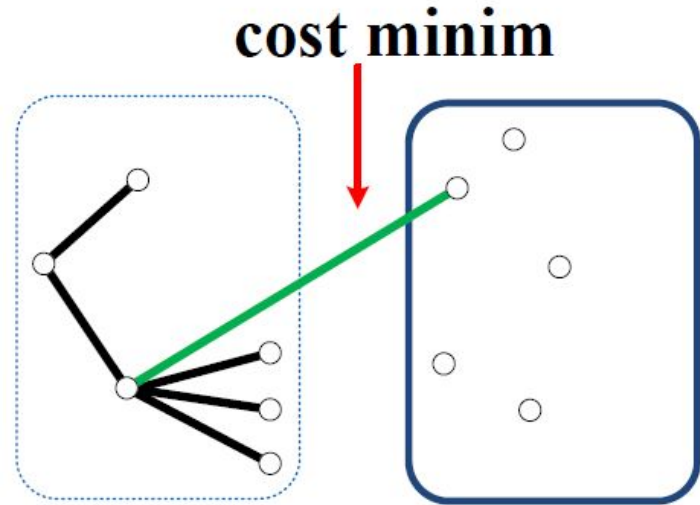
Vom demonstra că algoritmiile lui Kruskal și Prim aplică acest criteriu.

# Corectitudine Kruskal + Prim

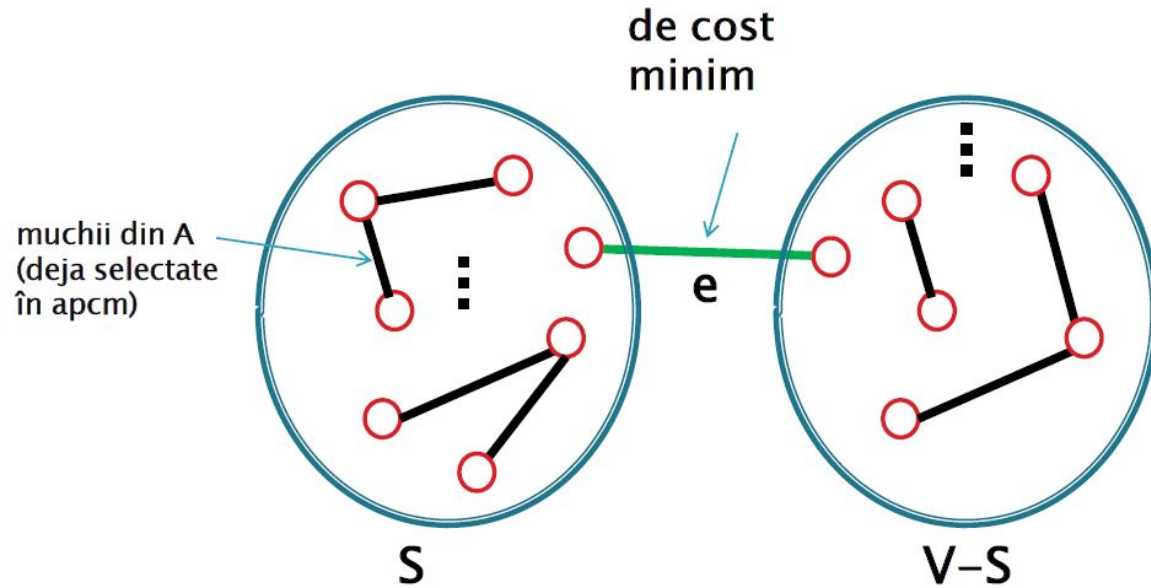
## KRUSKAL



## PRIM



# Corectitudine Kruskal + Prim



$$A \subseteq \text{apcm} \Rightarrow A \cup \{e\} \subseteq \text{apcm}$$

# Corectitudine Kruskal + Prim

Fie  $G = (V, E, w)$  un **graf conex ponderat**.

**Propoziție.** Algoritmul lui Kruskal determină un apcm.

**Propoziție.** Algoritmul lui Prim determină un apcm.

