Arbori parțiali de cost minim

Algoritmul lui Prim

Algoritmul lui Prim

- Se porneşte de la un vârf (care formează arborele iniţial)
- La un pas este selectată o muchie de cost minim de la un vârf deja adăugat la arbore la unul neadăugat

O primă formă a algoritmului

Kruskal

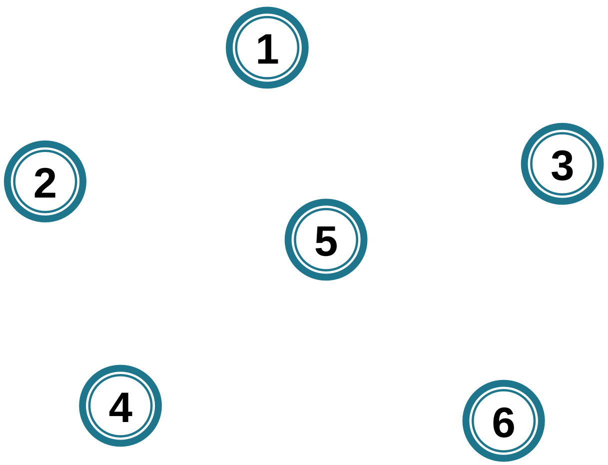
- Inițial T= (V; ∅)
- **pentru i = 1, n-1**
  - alege o muchie uv cu **cost minim** a.î. u,v sunt în **componente conexe diferite (T+uv aciclic)**
  - $E(T) = E(T) \cup uv$

Prim

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

Kruskal

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



- **Se încearcă unirea acestor componente prin muchii de cost minim**

Prim

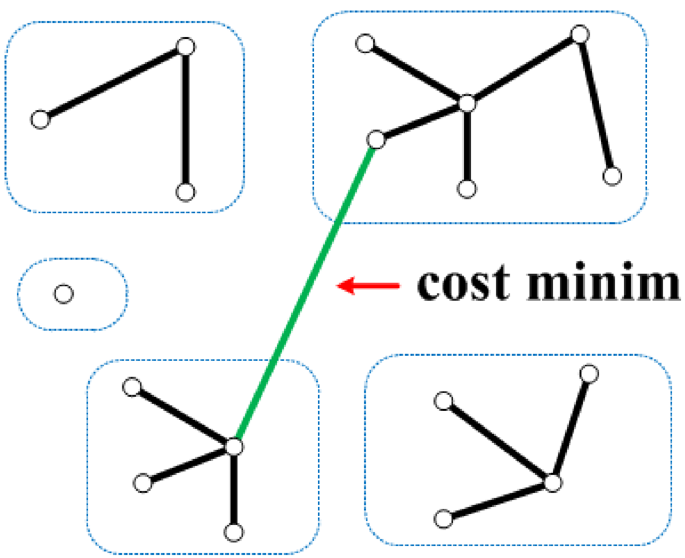
- **Inițial**: se pornește de la un vârf de start

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

Kruskal

- <u>La un pas</u>:

Muchiile selectate formează o **pădure**

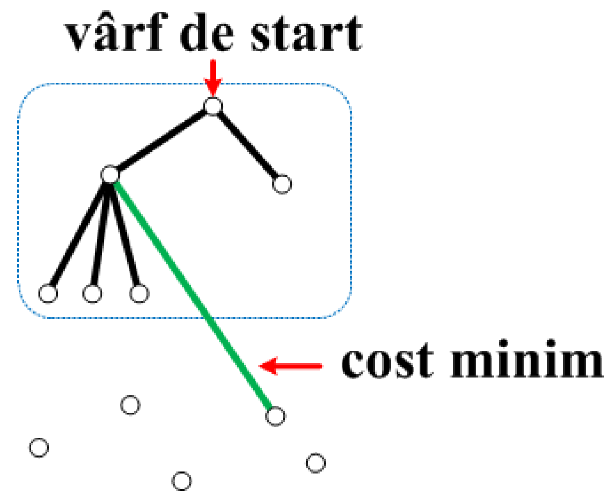


Este selectată o muchie de cost minim care unește doi arbori din pădurea curentă (două componente conexe)

Prim

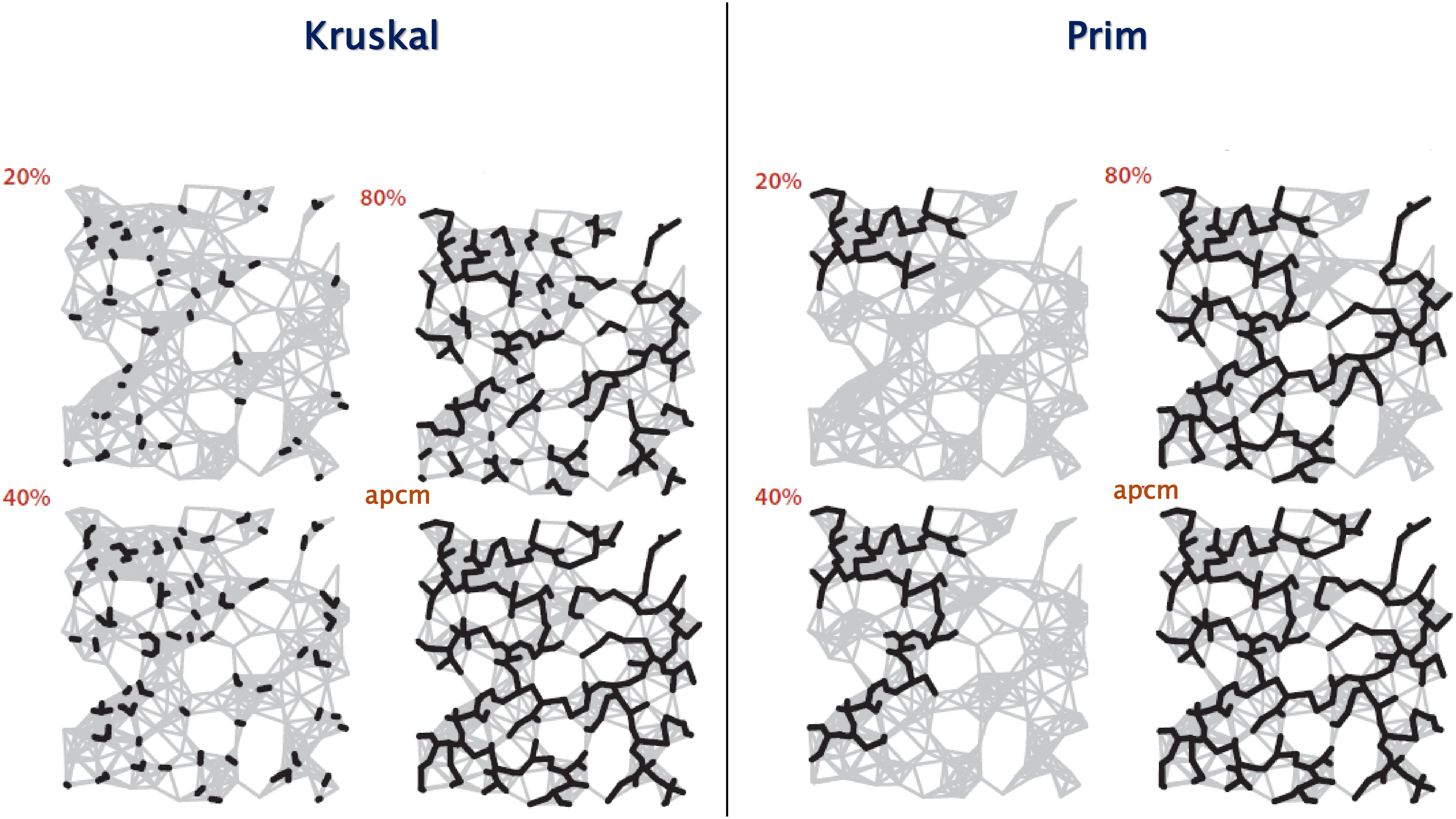
- <u>La un pas</u>:

Muchiile selectate formează un **arbore**

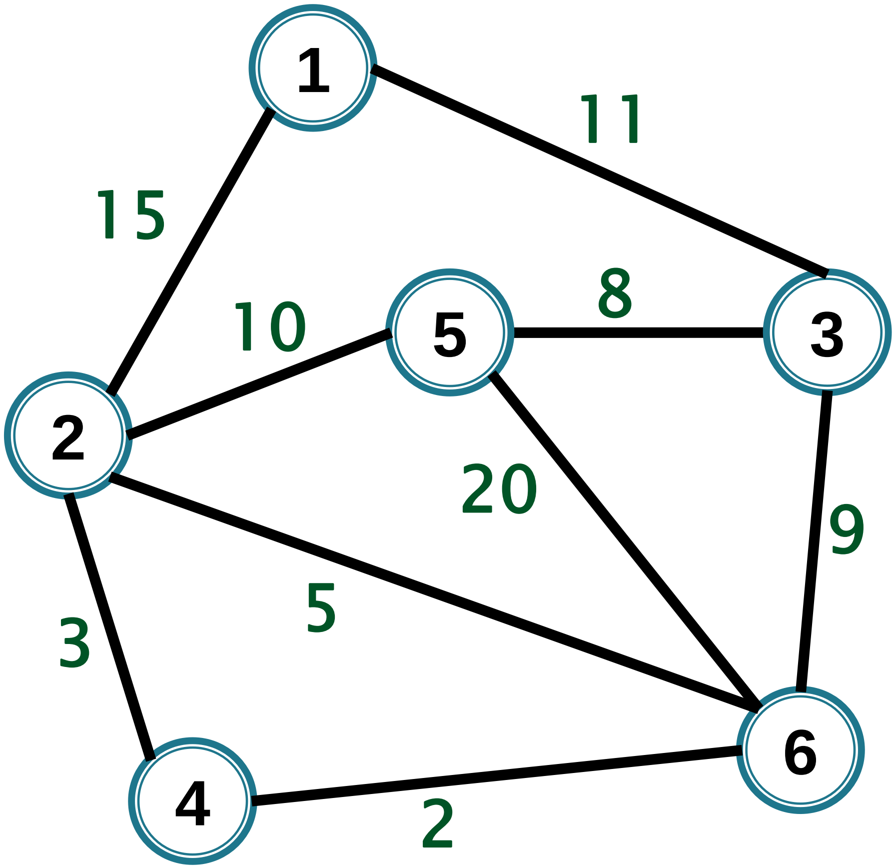


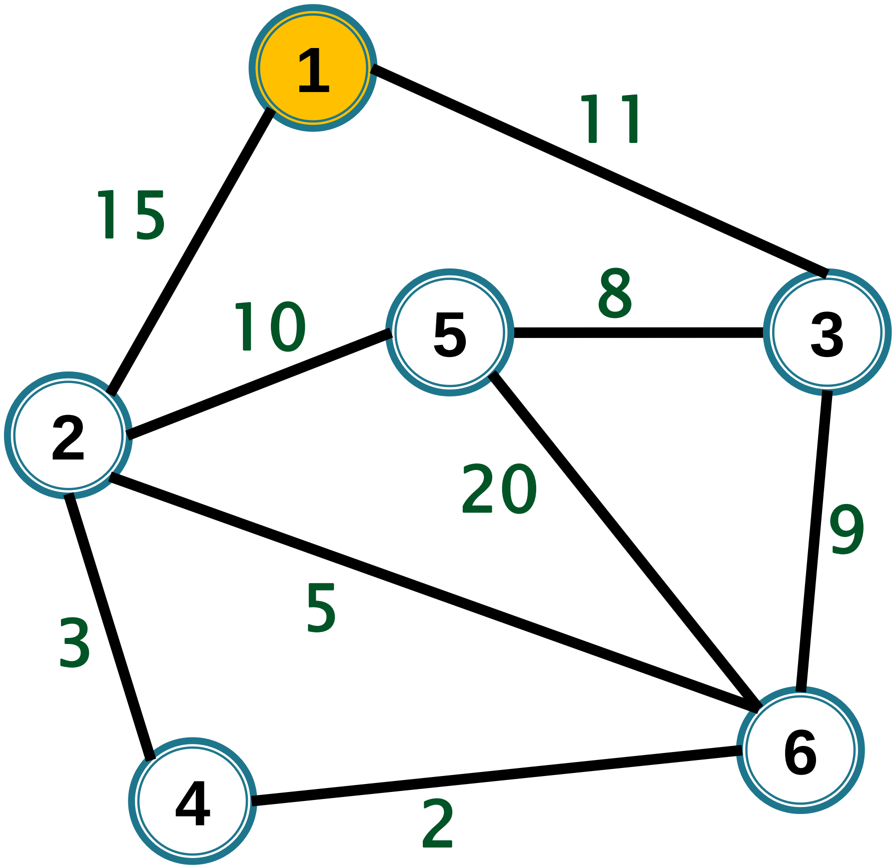
Este selectată o muchie de cost minim care unește un vârf din arbore cu unul care nu este în arbore(neselectat)

Arbori parțiali de cost minim

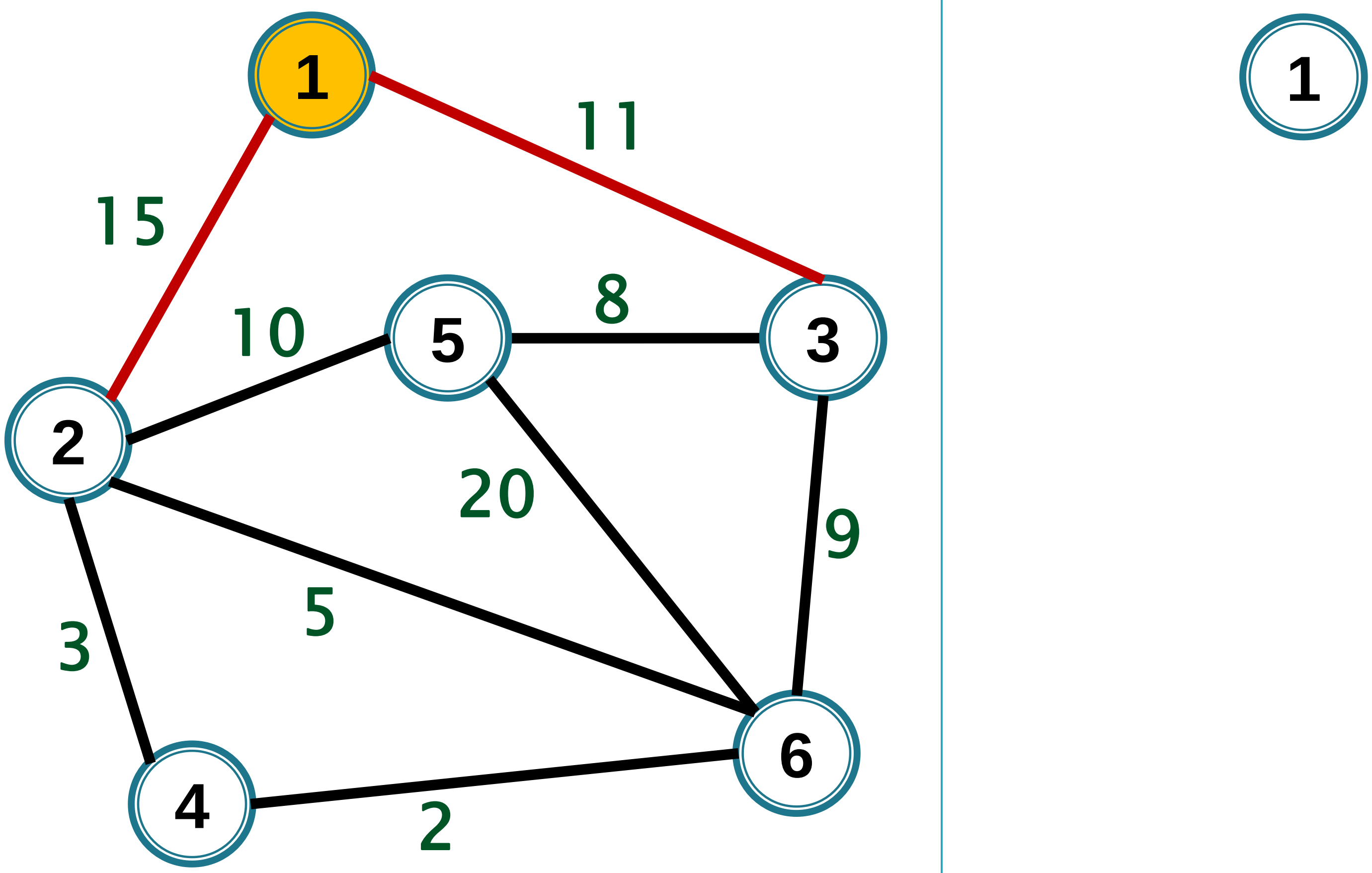


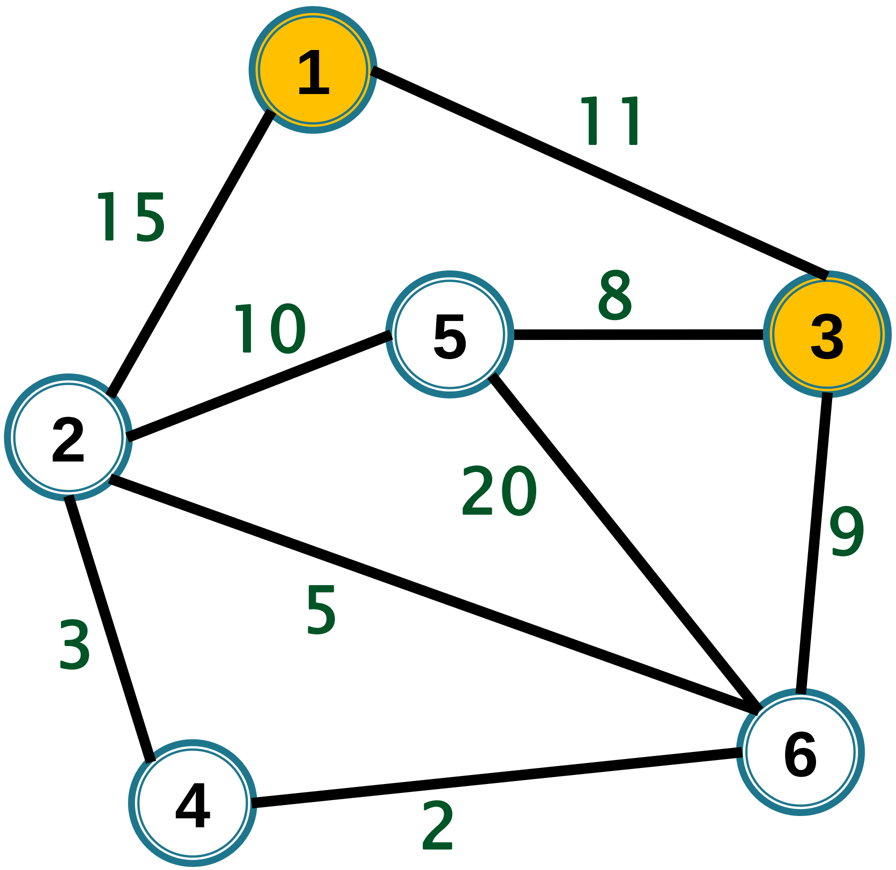
Imagine din
R. Sedgewick, K. Wayne – **Algorithms, 4th edition, Pearson** Education, 2011


1
2
3
4
5
6
11
15
10
8
20
9
5
3
2

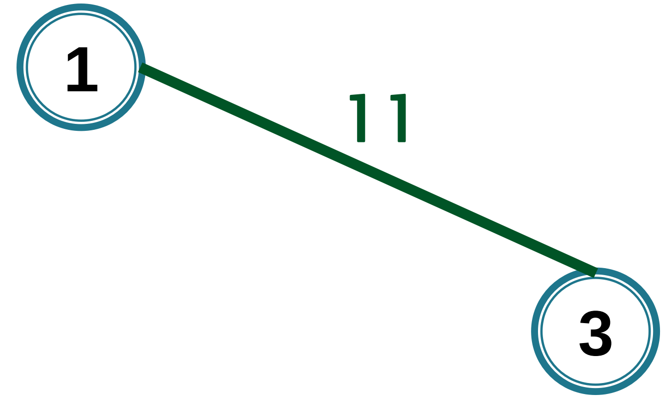


s = 1

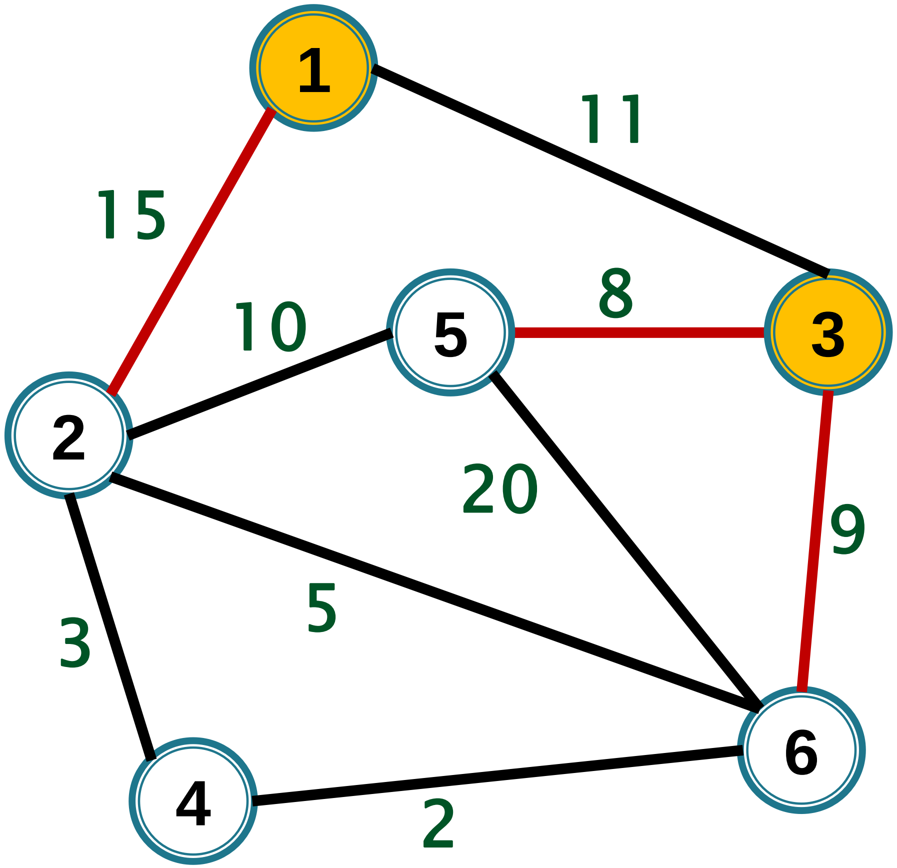

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2
1

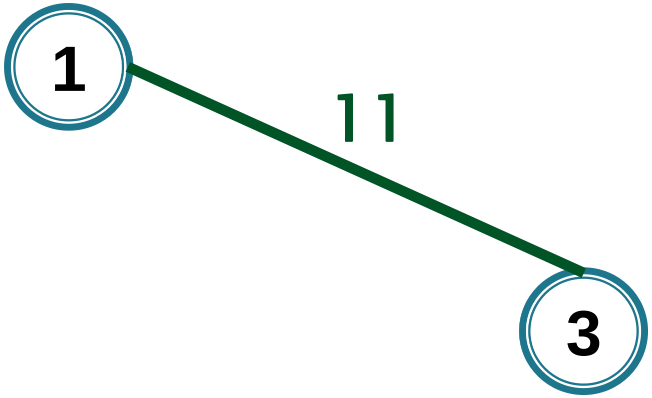


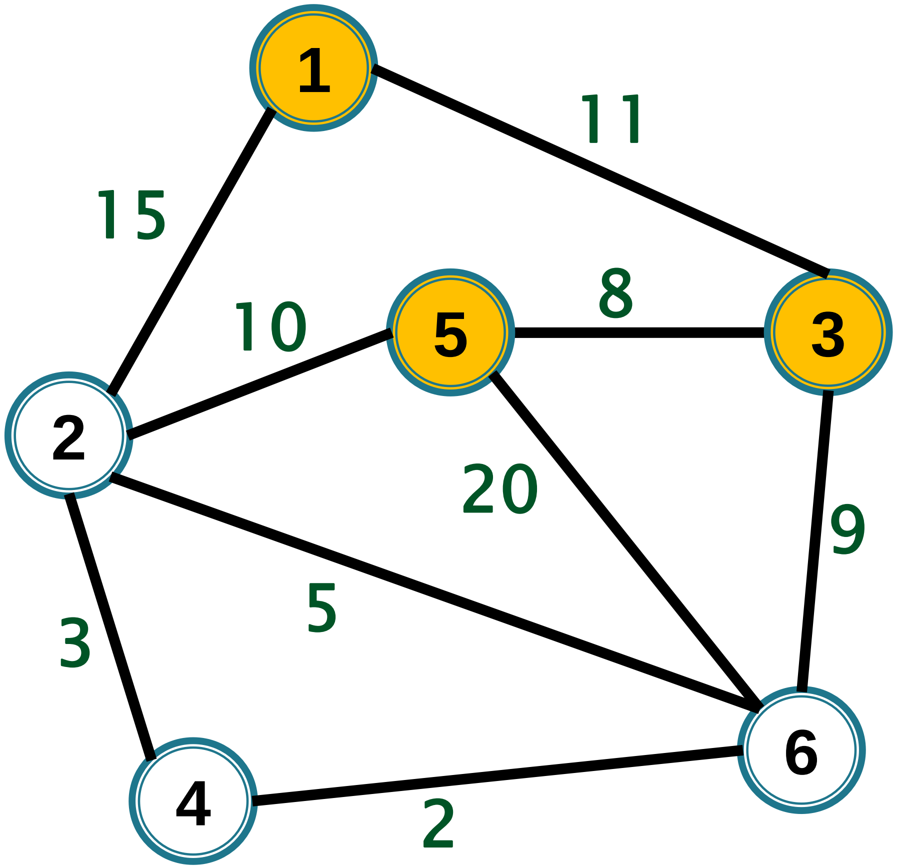
1
2
3
4
5
6
15
11
10
8
20
9
5
3
2

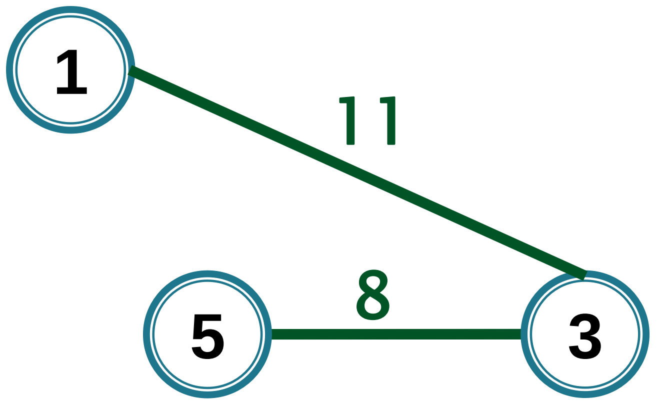


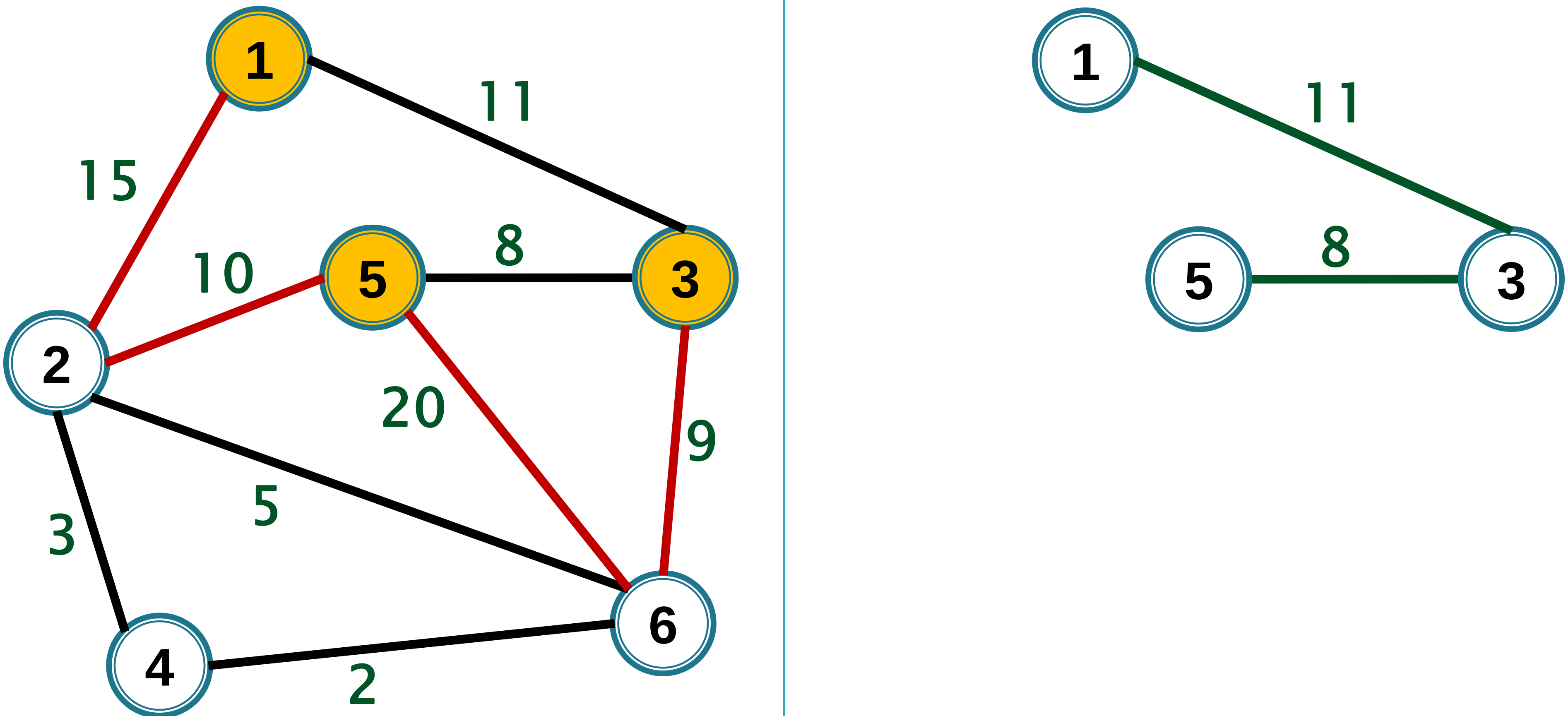
1
3
11

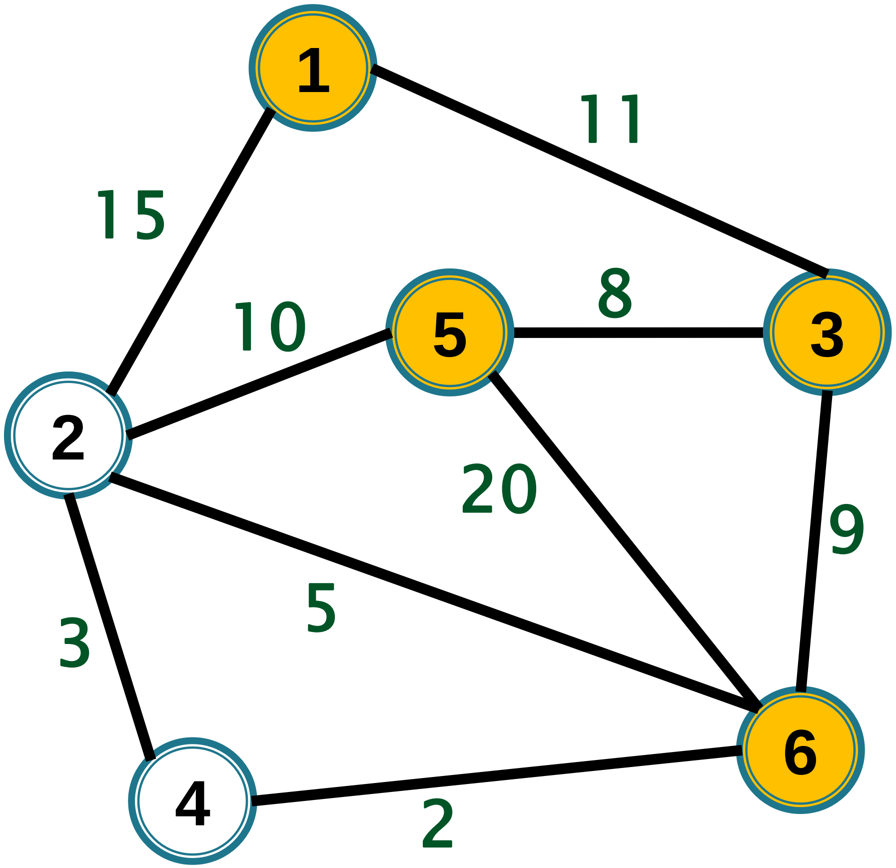

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2

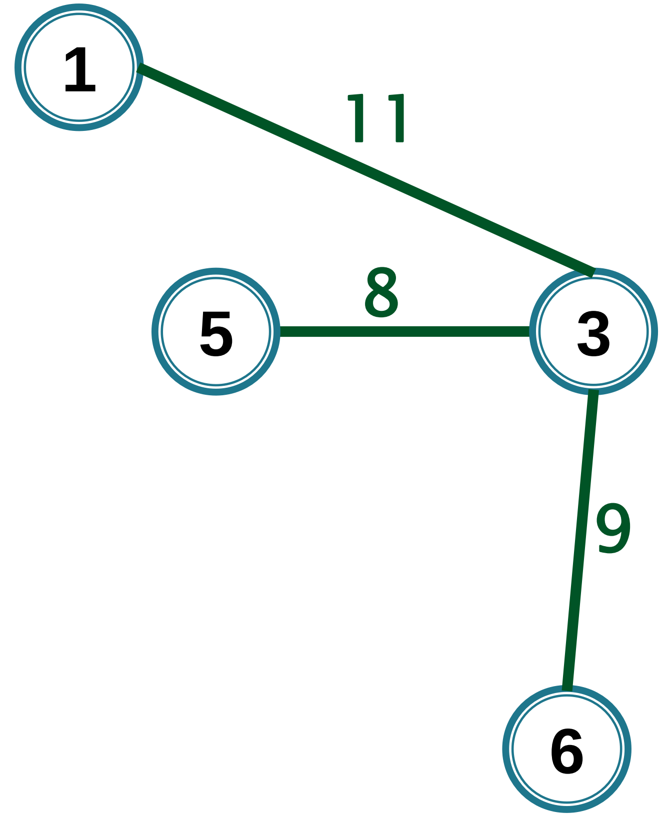

1
3
11

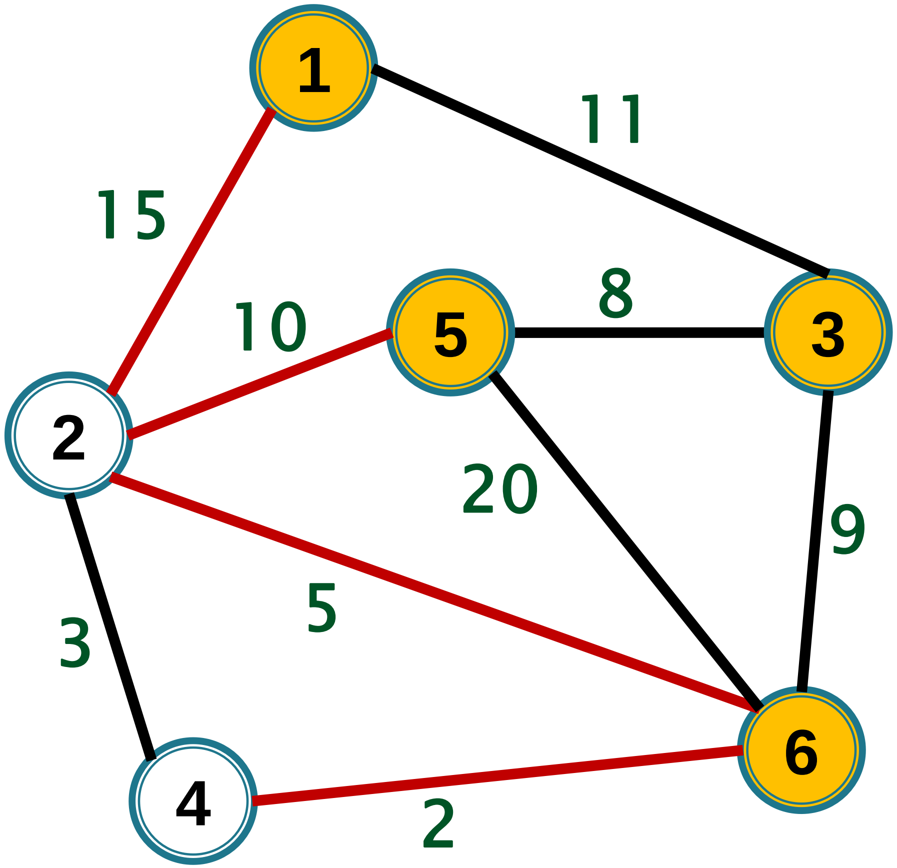

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2


1
5
3
11
8

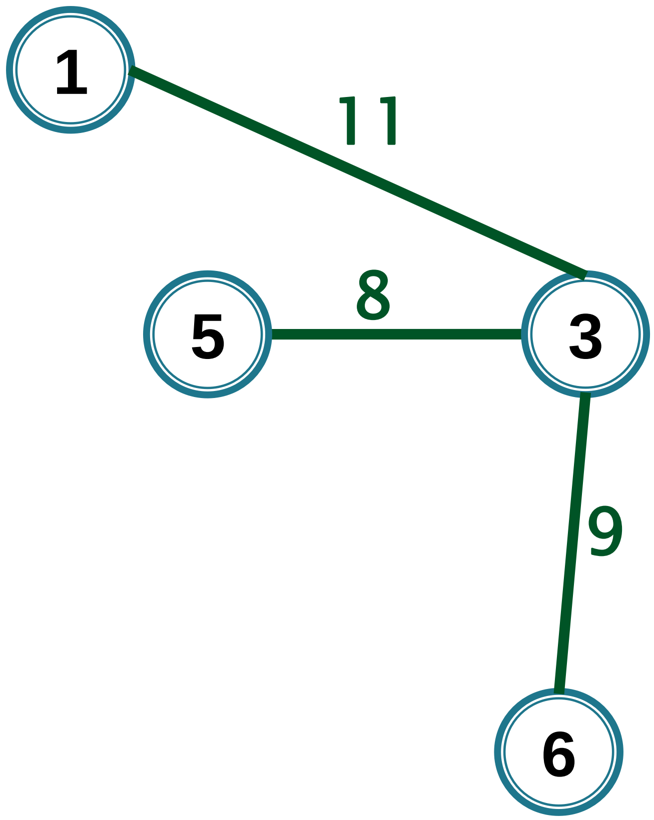

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2
1
5
3
11
8


1
2
3
4
5
6
15
11
10
8
20
9
5
3
2

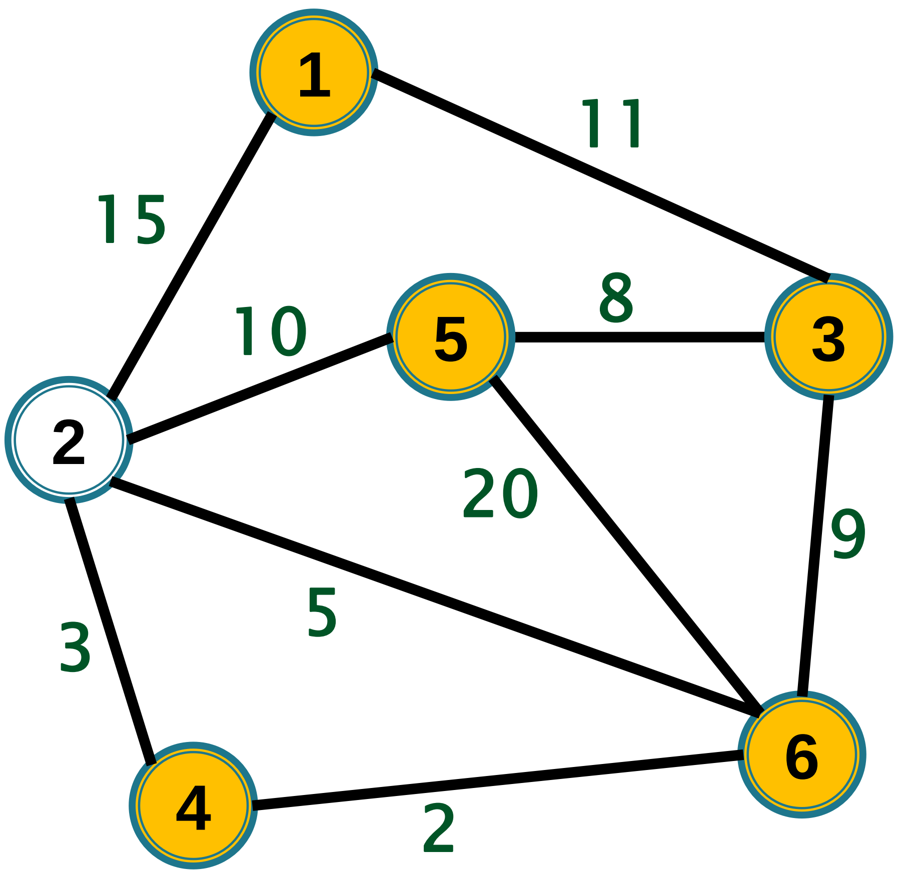

1
3
5
6
11
8
9

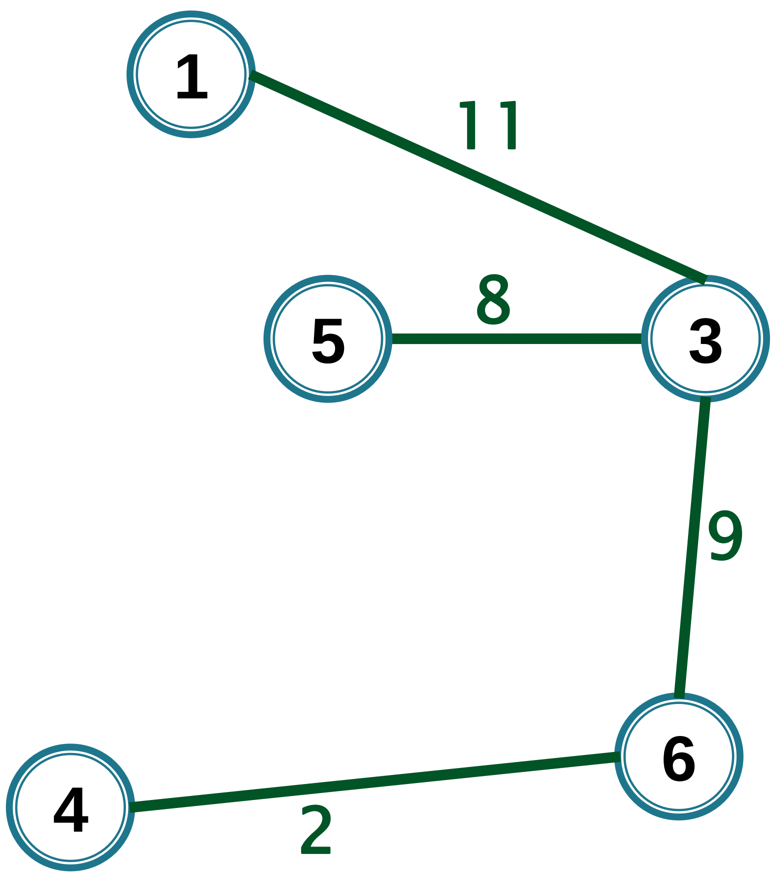


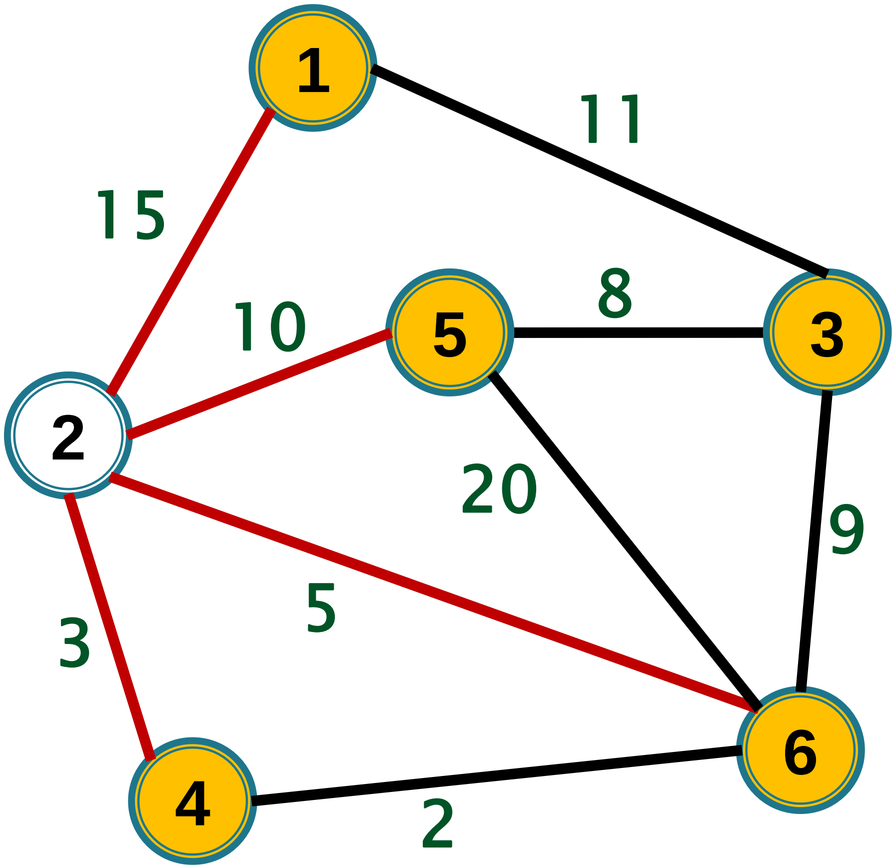
1
2
3
4
5
6
11
15
10
8
20
9
5
3
2

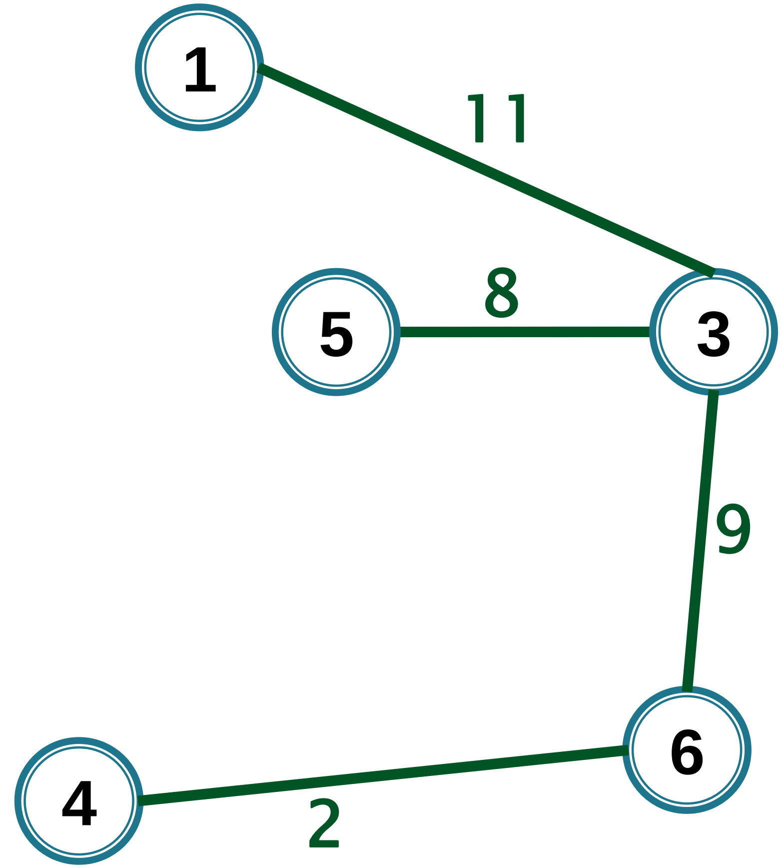


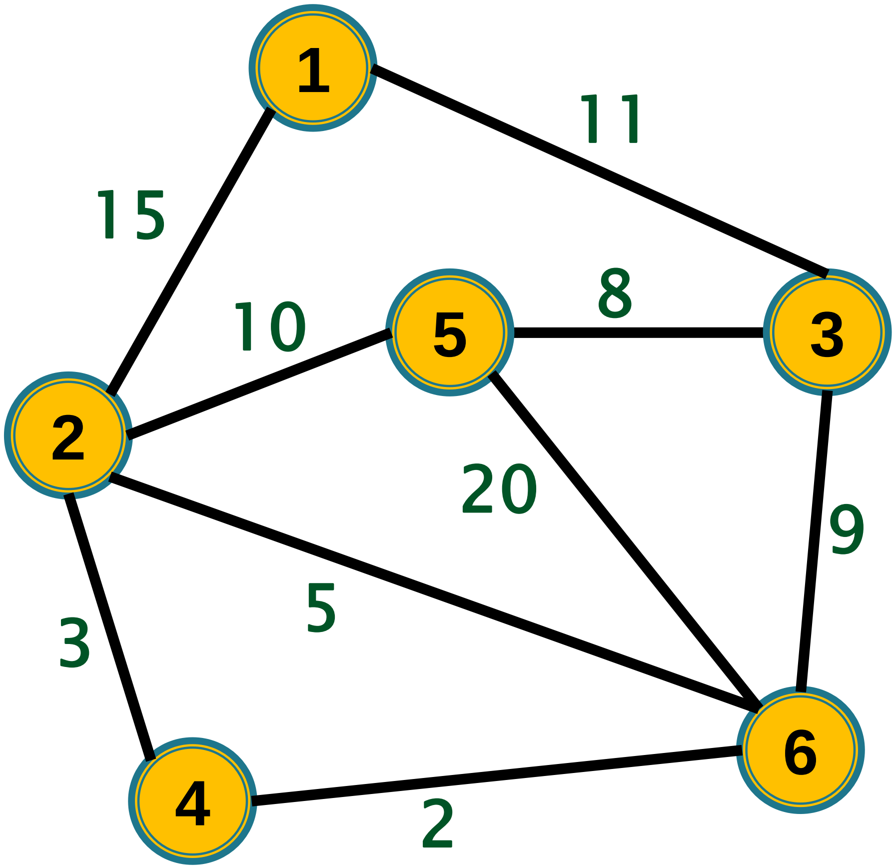
1
3
5
6
11
8
9

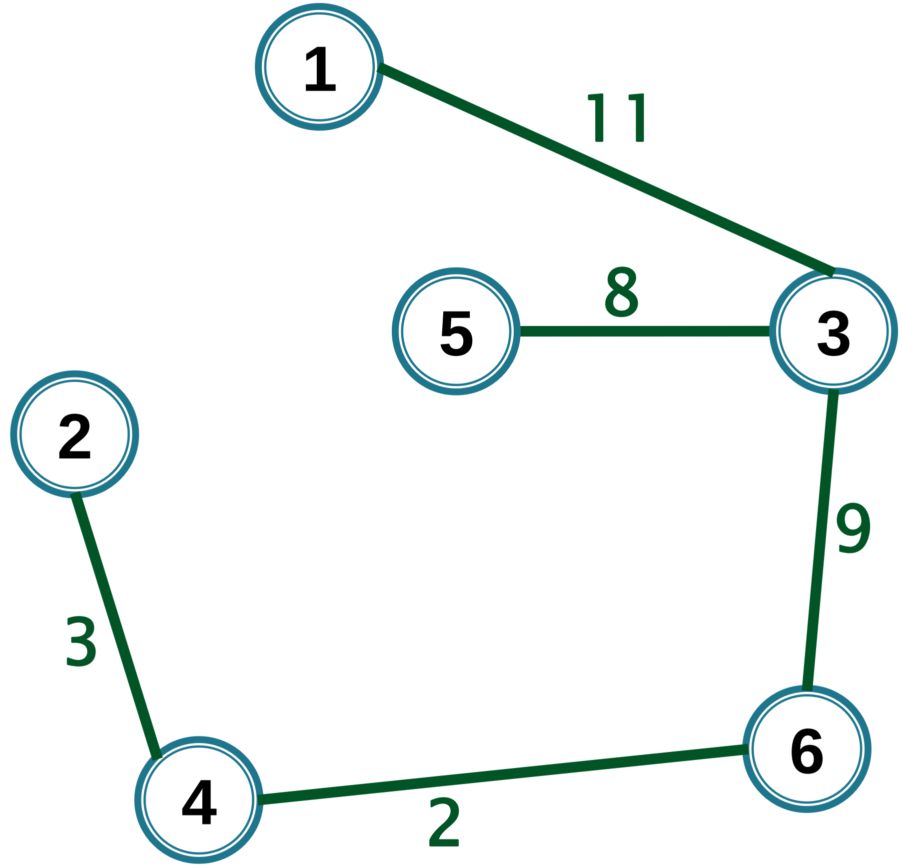

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2

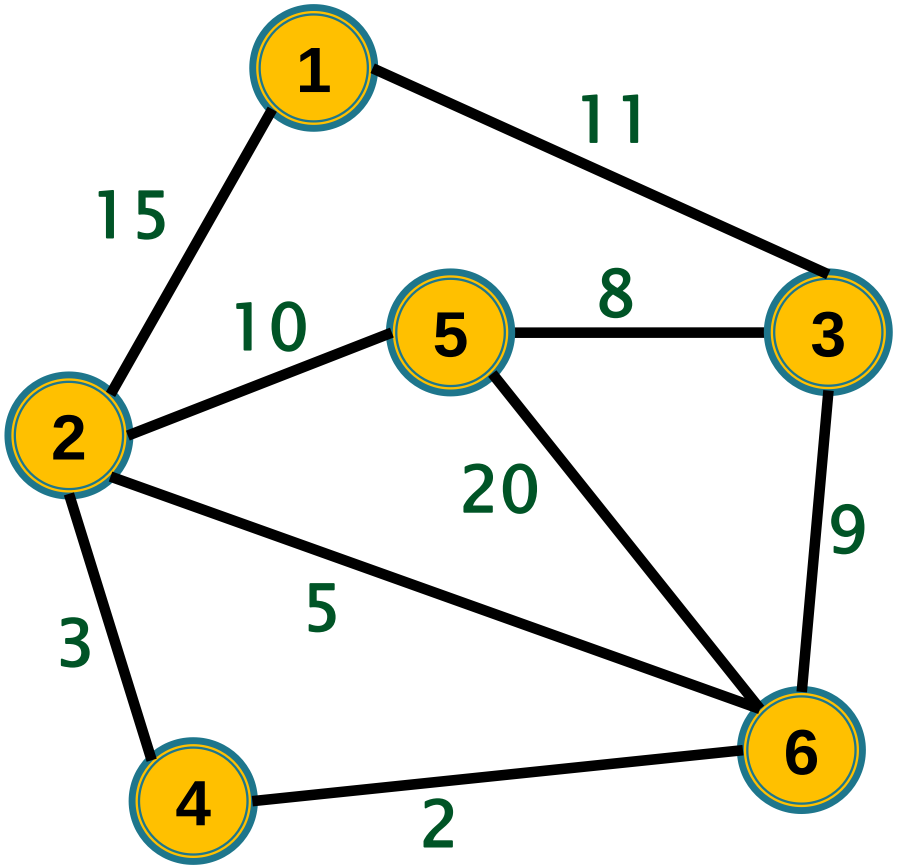

1
3
4
5
6
11
8
9
2


1
2
3
4
5
6
15
11
10
8
20
9
5
3
2

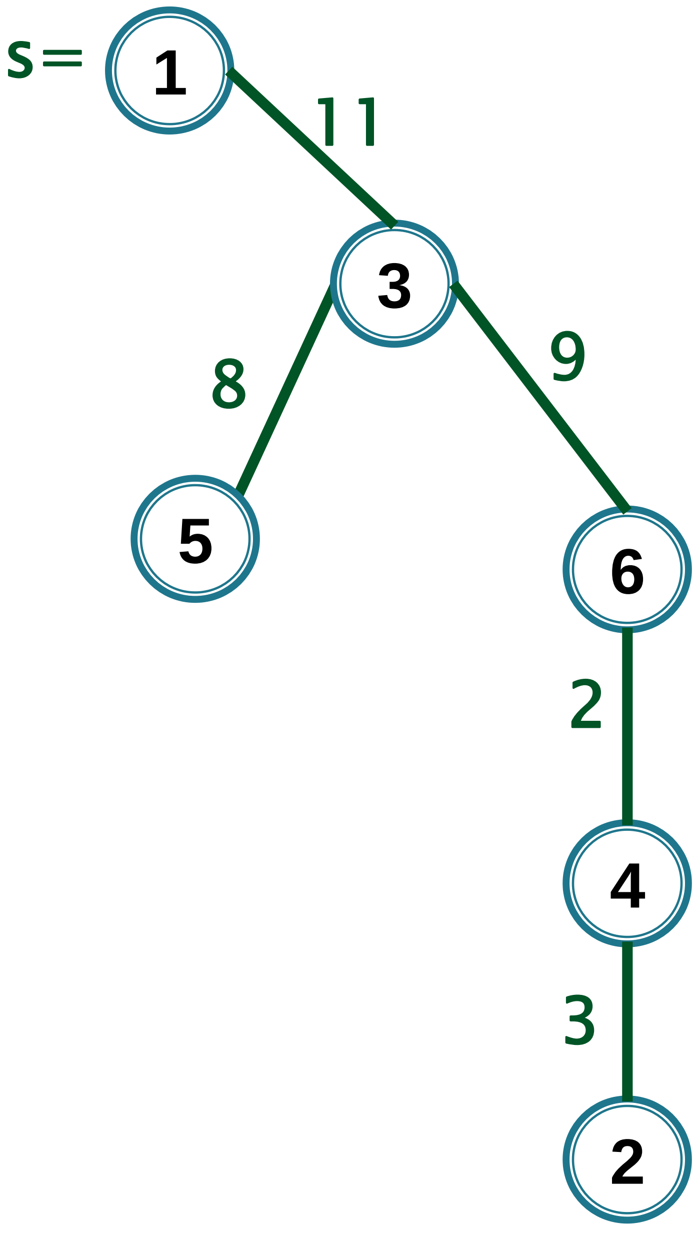

1
3
4
5
6
11
8
9
2


1
11
15
10
5
8
3
2
20
9
5
3
4
6
2


1
11
5
8
3
2
9
3
4
6
2



1
2
3
4
5
6
15
11
10
8
20
9
5
3
2

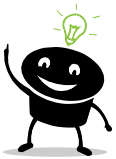


s=
1
3
5
6
4
2
11
8
9
2
3

Implementare+Complexitate

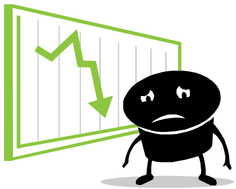
Cum alegem *eficient* o muchie de cost minim cu o extremitate selectată (deja în arbore) și cealaltă nu?

Implementare+Complexitate



- La fiecare pas parcurgem toate muchiile şi o alegem pe cea de cost minim cu o extremitate selectată şi una neselectată

O(nm)



Implementare Prim

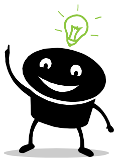
Cum evităm să comparam de fiecare dată toate muchiile cu o extremitate în arbore și cealaltă nu.

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**

Implementare Prim

Cum evităm să comparam de fiecare dată toate muchiile cu o extremitate în arbore și cealaltă nu.



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

Implementare+Complexitate

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

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

sau

- heap de muchii

(v. laborator+seminar + slideuri implementare+ alg. Dijkstra)

Algoritmi bazaţi pe eliminare de muchii

Temă – Care dintre următorii algoritmi determină corect un arbore parţial de cost minim (justificaţi)? Pentru fiecare algoritm corect precizaţi ce complexitate are.

1.
```
T ← G
cât timp T conţine cicluri execută
      alege e o muchie de cost maxim care este
      conţinută într-un ciclu din T
      T ← T - e
```

2.
```
T ← G
cât timp T conţine cicluri execută
      alege C un ciclu oarecare din T şi fie e
      muchia de cost maxim din C
      T ← T - e
```

Corectitudine

Corectitudine Kruskal + Prim

- Cei doi algoritmi determină corect un apcm?
  Chiar dacă muchiile au şi 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 $A \subseteq$ apcm $\Leftrightarrow \exists$ T un apcm astfel încât $A \subseteq E(T)$

Corectitudine Kruskal + Prim

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

- $A \leftarrow \varnothing$ (mulțimea muchiilor selectate în arborele construit)
- pentru i = 1, n−1 execută

  alege o muchie *e* astfel încât $A \cup \{e\} \subseteq$ apcm

  $A = A \cup \{e\}$
- returnează $T = (V, A)$

Corectitudine Kruskal + Prim

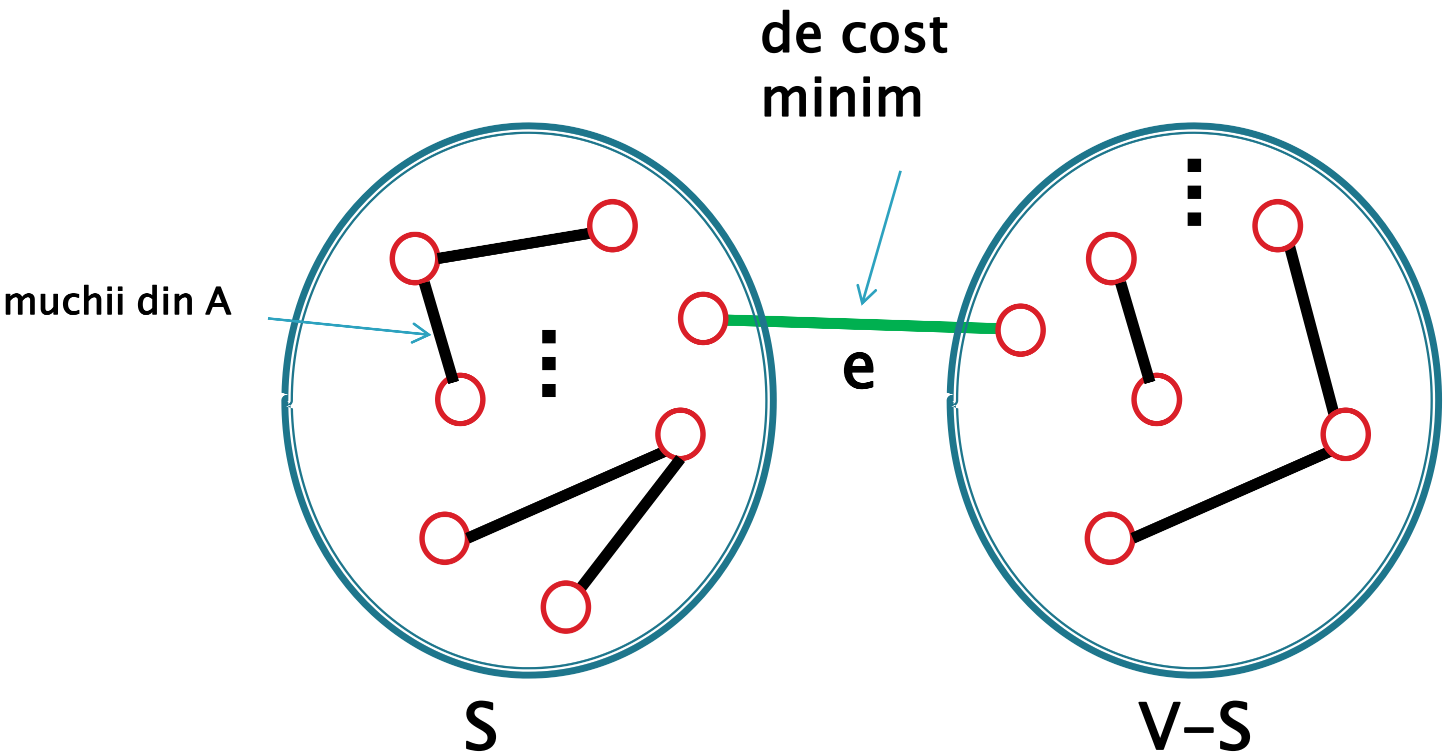
- vom demonstra un **criteriu de alegere a muchiei e** la un pas astfel încât:

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

și

- **vom demonstra că algoritmii lui Kruskal și Prim aplică acest criteriu.**

Corectitudine Kruskal + Prim



Corectitudine Kruskal + Prim

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

- **Propoziție.** Algoritmul Kruskal determină un apcm
- **Propoziție.** Algoritmul Prim determină un apcm

Detalii implementare
Algoritmul lui Prim

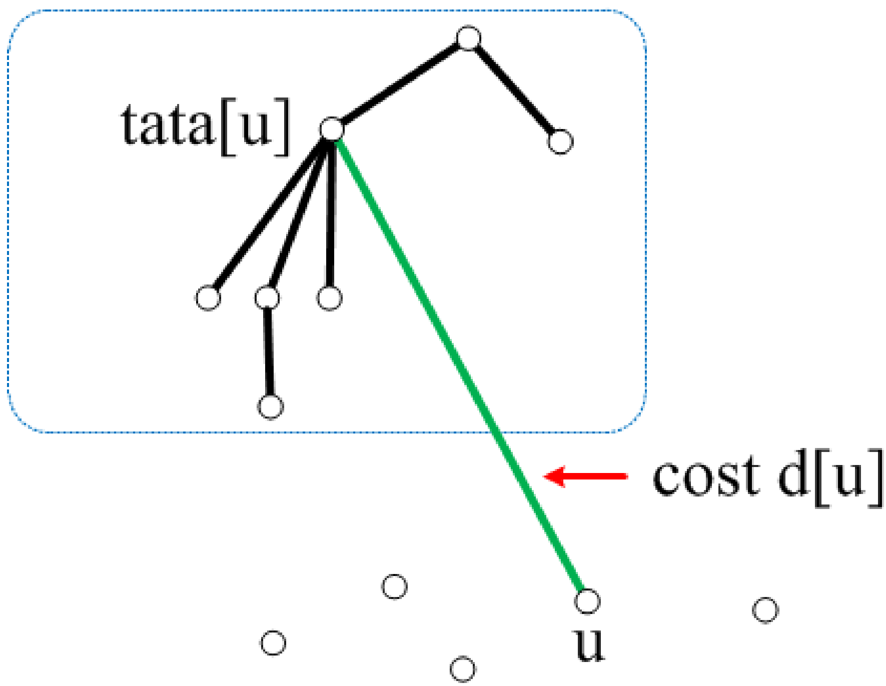
Implementare Prim

Asociem fiecărui vârf u următoarele informaţii (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 selectat deja în arbore
- **tata[u]** = acest vârf din arbore pentru care se realizează minimul

Implementare Prim

- **Avem**
  - **(u, tata[u])** este muchia de cost minim de la u la un vârf din arbore
  - **d[u] = w(u, tata[u])**



Implementare Prim

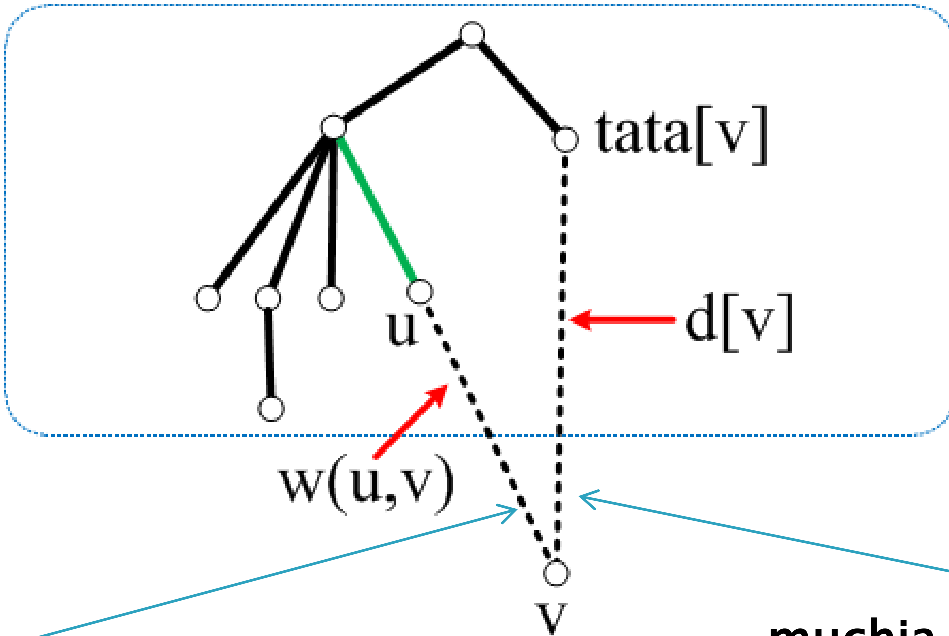
Atunci algoritmul se modifică astfel:

- **La un pas**

– se alege **un vârf** u cu **eticheta d minimă** care nu este încă în arbore şi se adaugă la arbore muchia **(tata[u], u)**

– se actualizează etichetele vârfurilor $v \notin V(T)$ vecine cu u astfel:

Prim



noua muchie: vu

muchia de cost minim determinată până acum: (v,tata[v])

```
dacă w(u,v) < d[v] atunci
      d[v] = w(u,v)
      tata[v] = u
```

Implementare Prim

- Muchiile arborelui vor fi în final

  $(u, tata[u]), u \neq s$

Prim

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

Prim

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

Prim

Varianta 1– Folosim vector de vizitat

$$Q[u] = \begin{cases} 1, & \text{dacă } u \notin Q \\ 0, & \text{altfel} \end{cases}$$

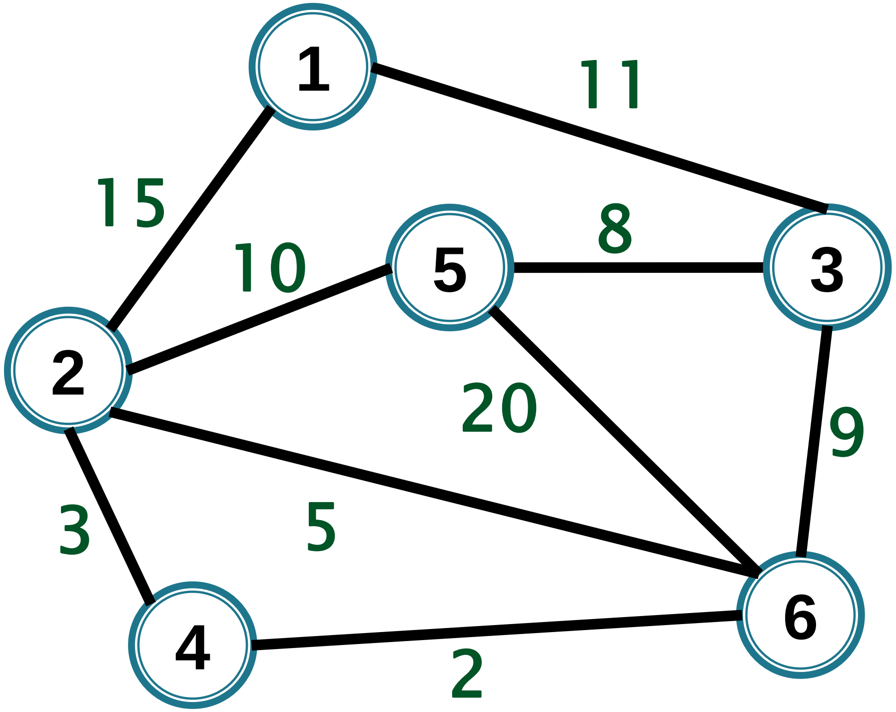
Prim

Complexitate

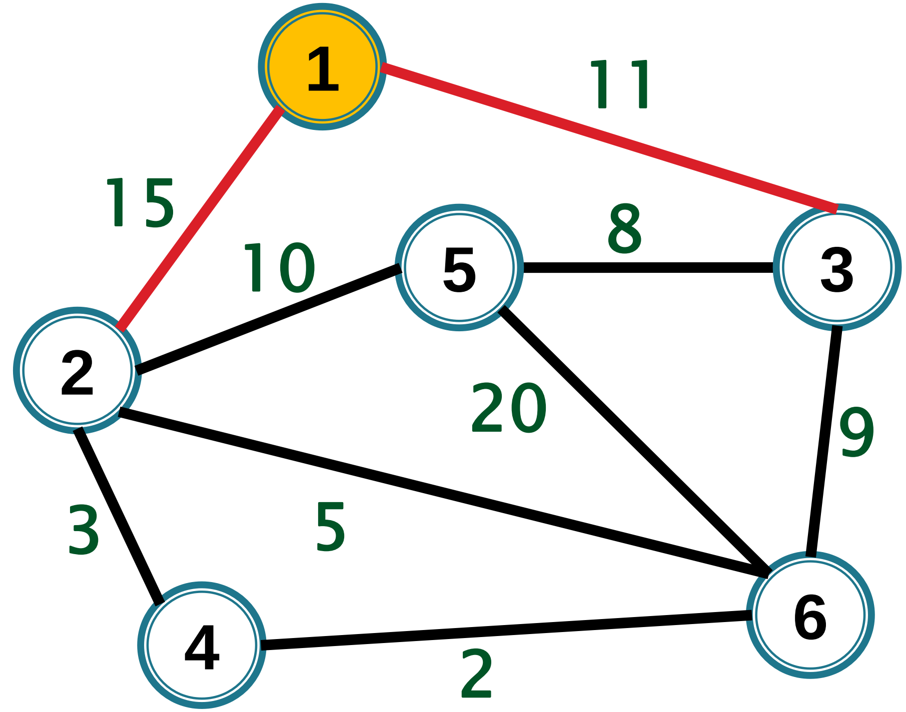
Varianta 1 – cu vector de vizitat

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

$O(n^2)$

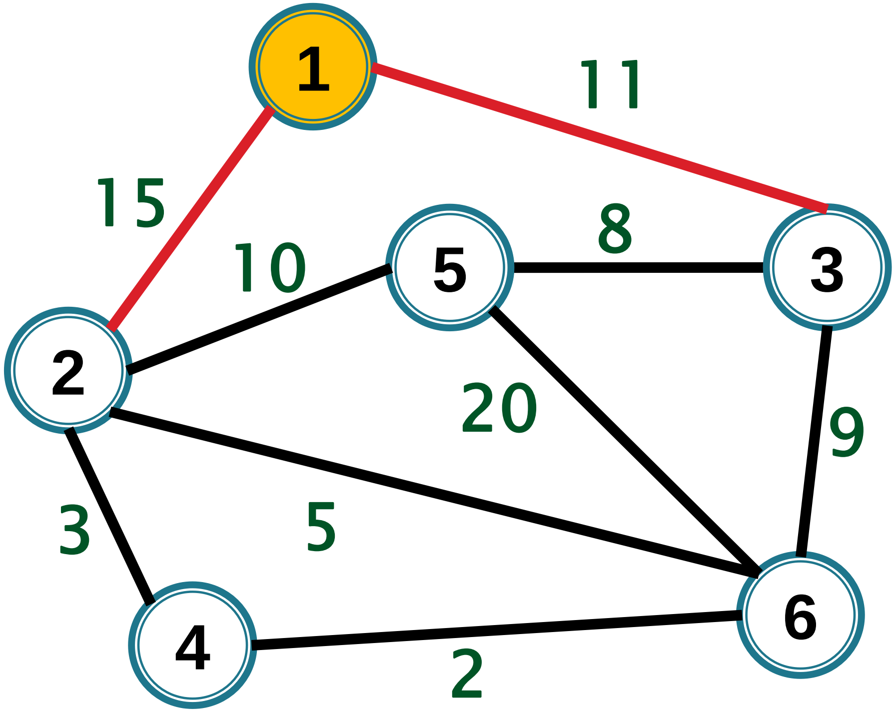


	1	2	3	4	5	6
d/tata=	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]



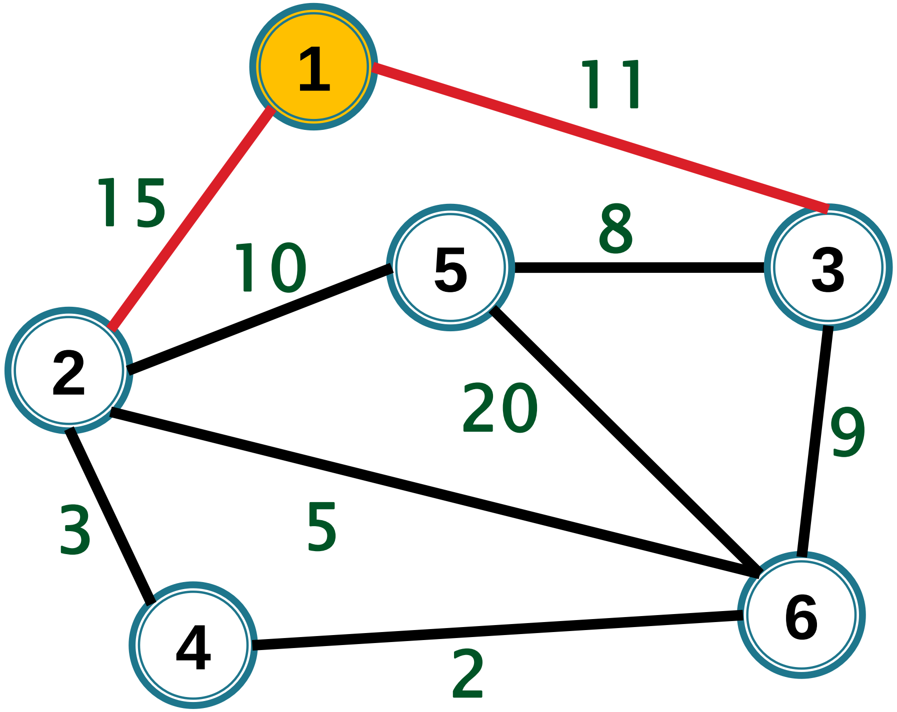
1	2	3	4	5	6
[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]

Sel. 1:



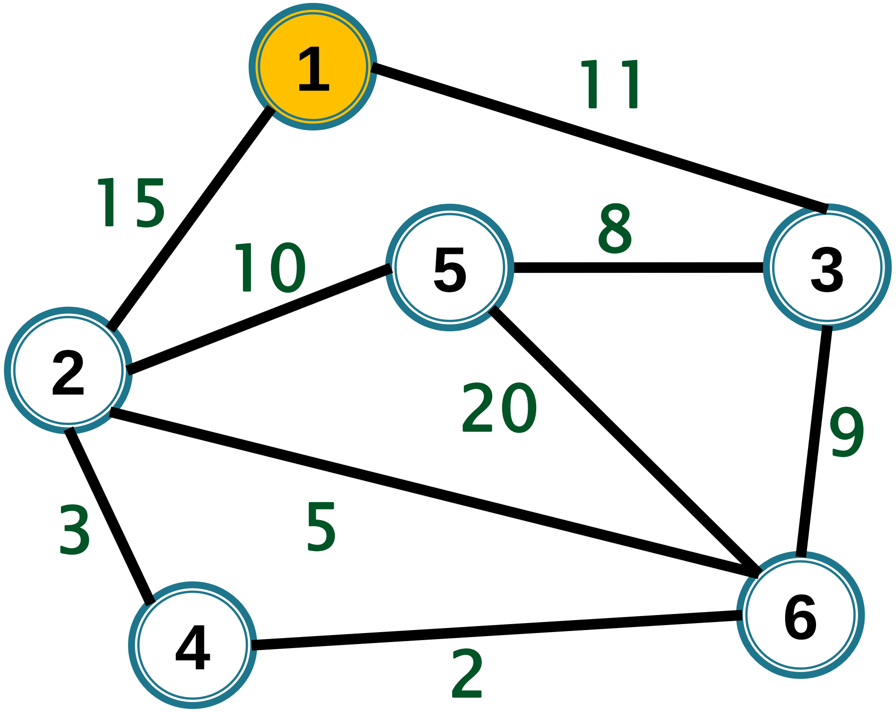
1	2	3	4	5	6
[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]

Sel. 1: viz [1] = 1 (vom reprezenta prin –)

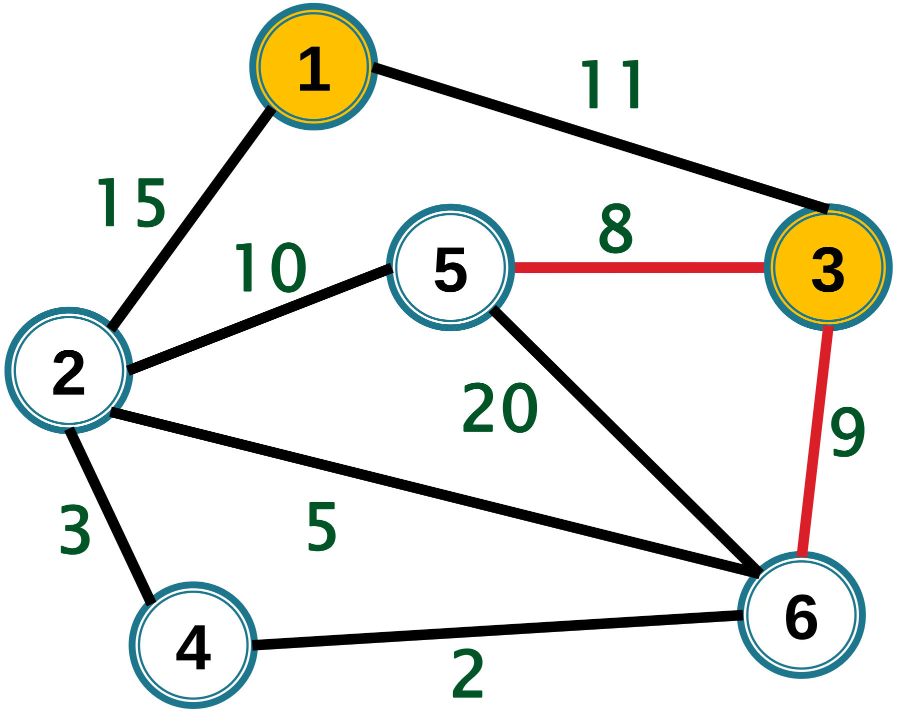


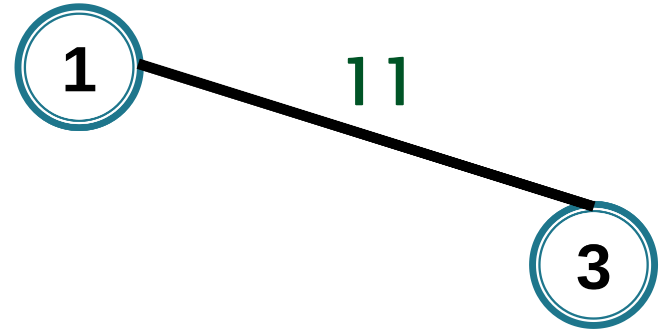
1	2	3	4	5	6
[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]

Sel. 1: actualizăm etichetele vecinilor

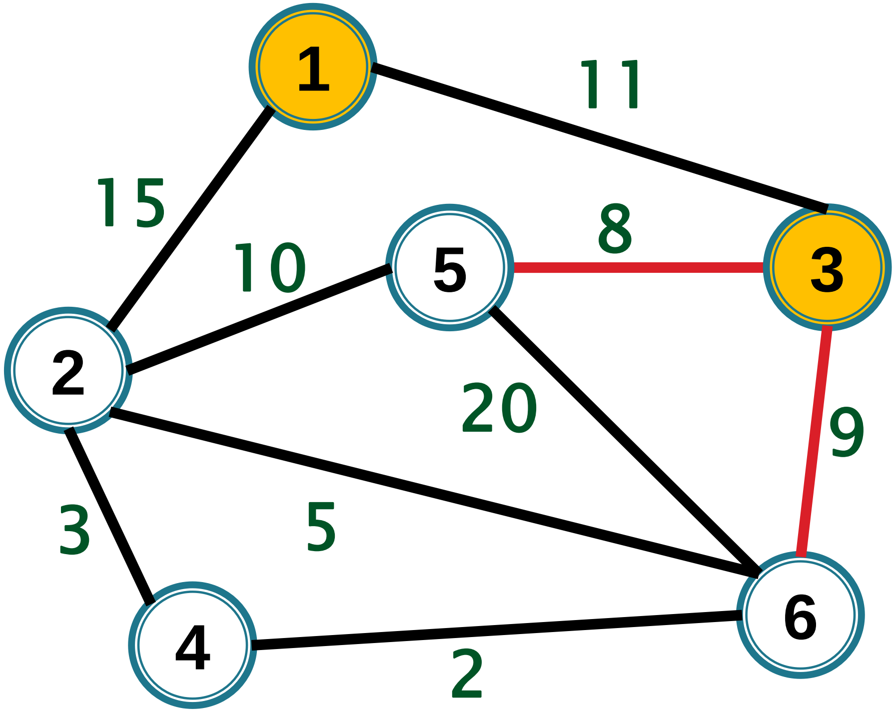


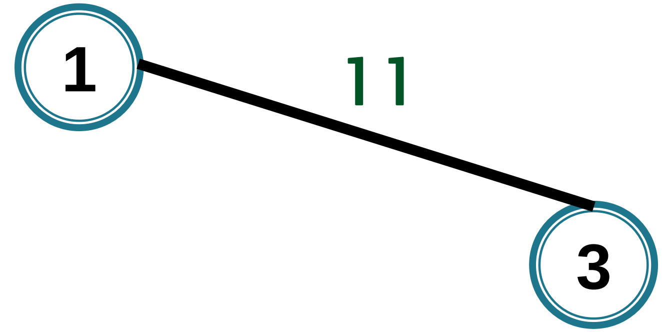
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]



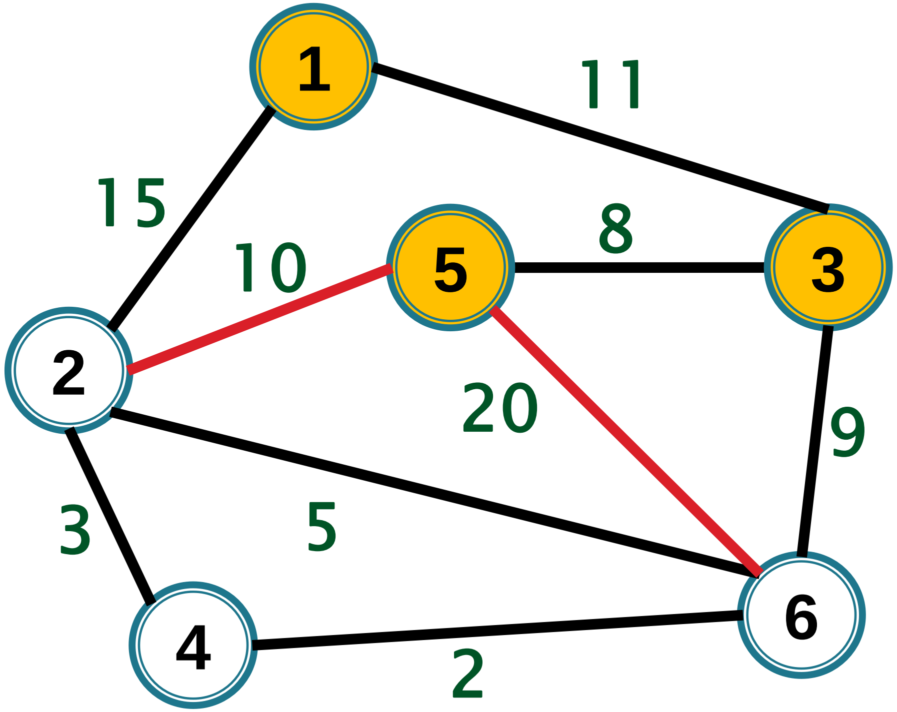


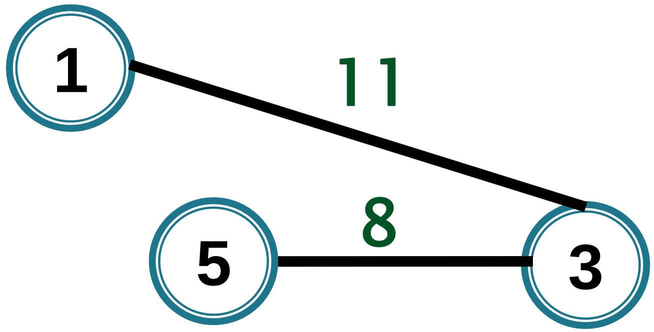
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:						



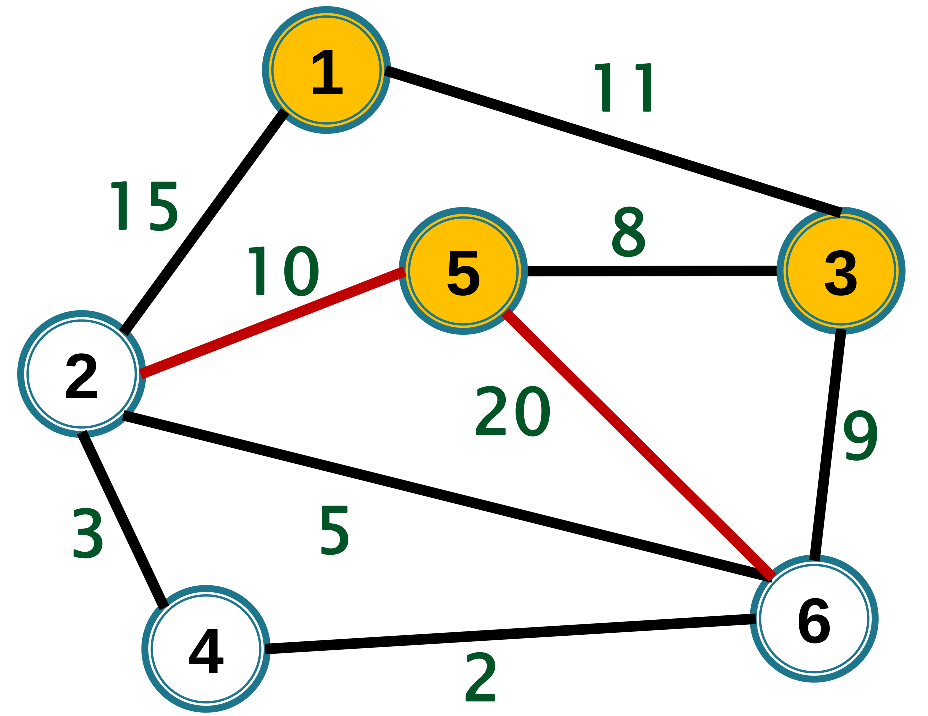


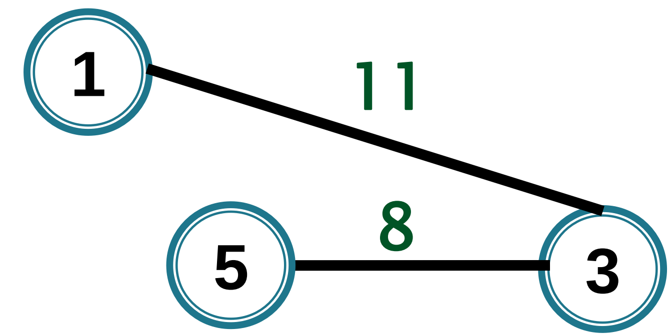
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]



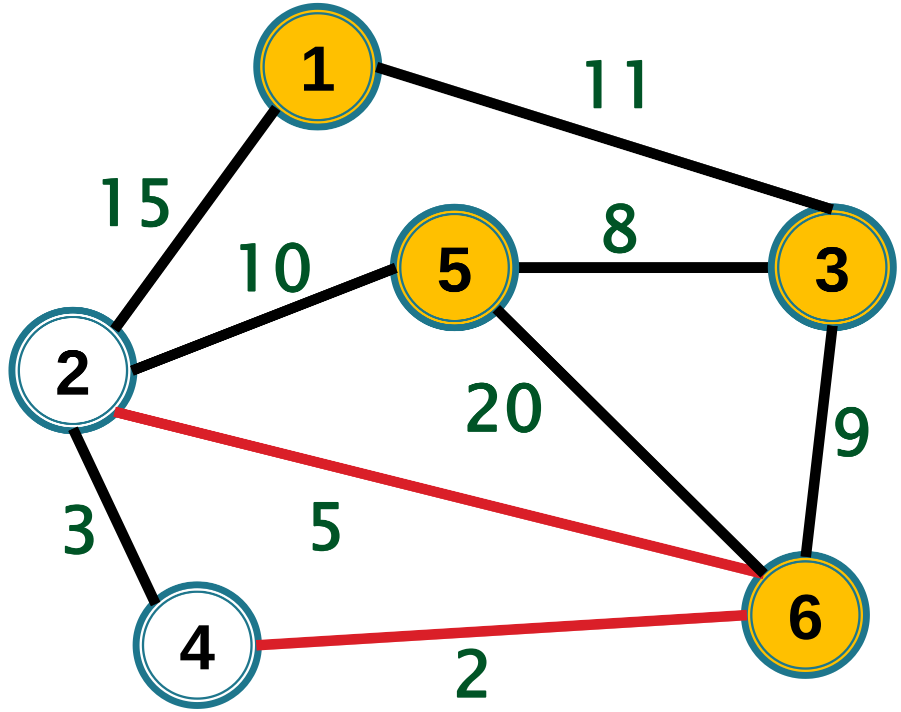


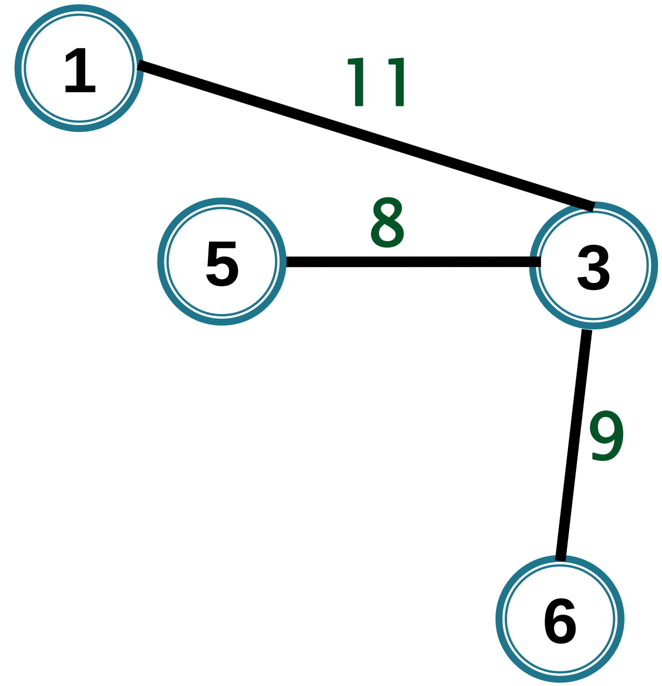
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:						



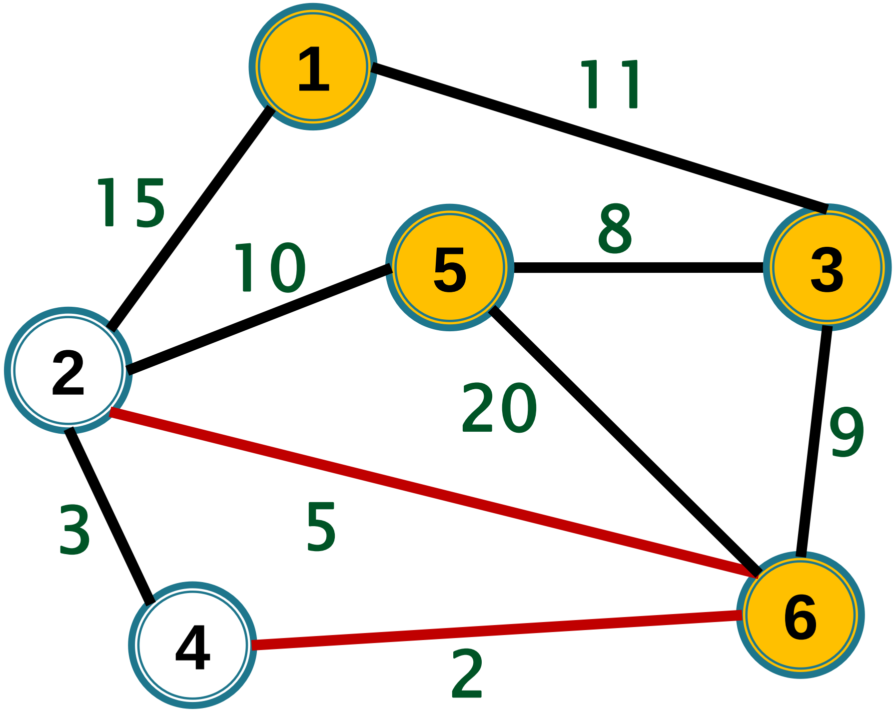


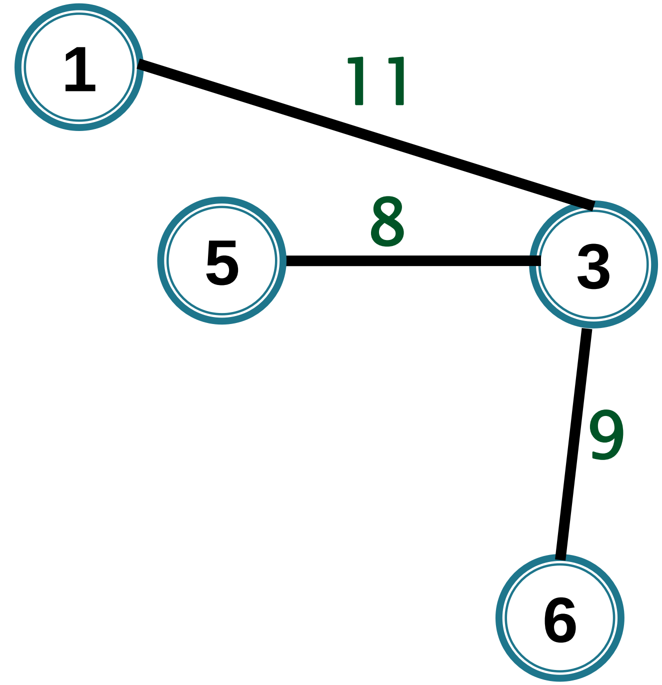
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]



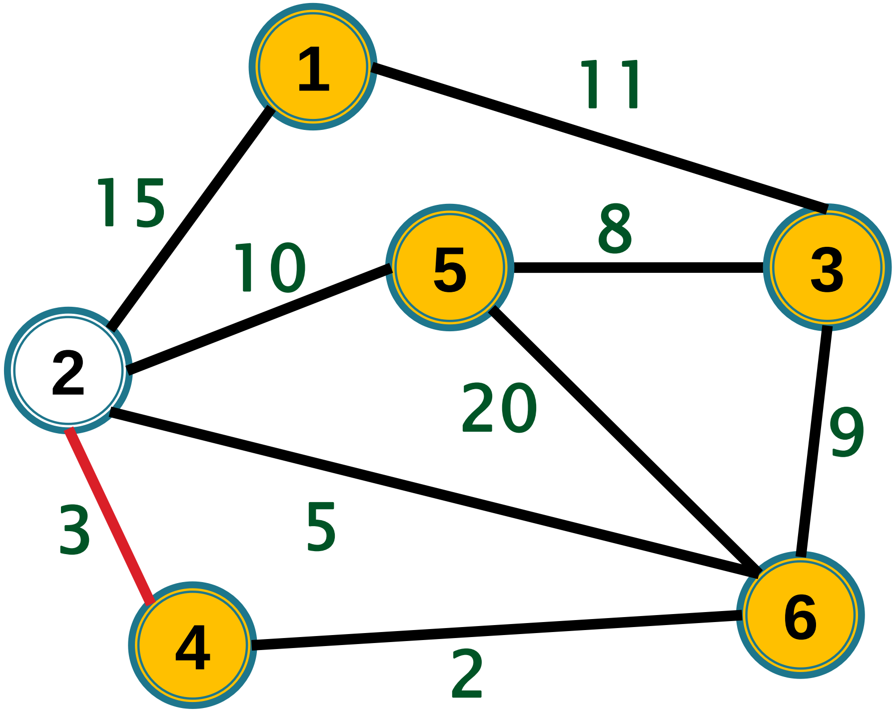


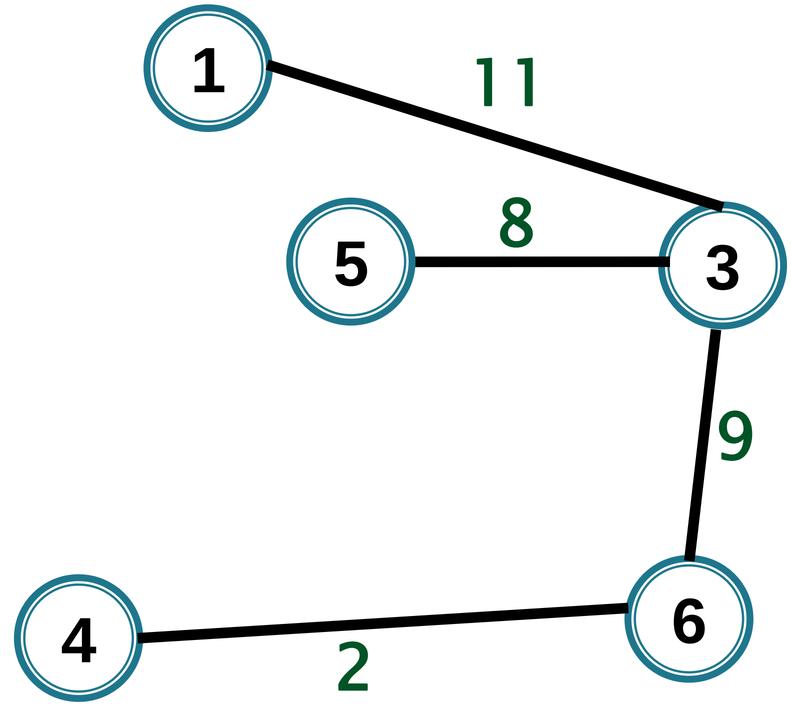
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]
Sel. 6:						



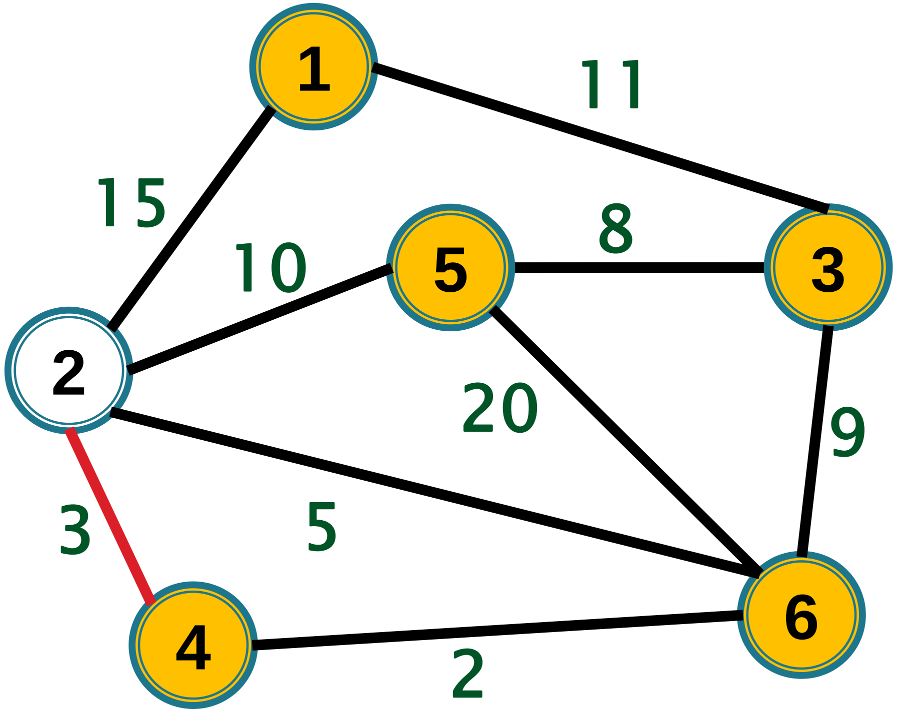


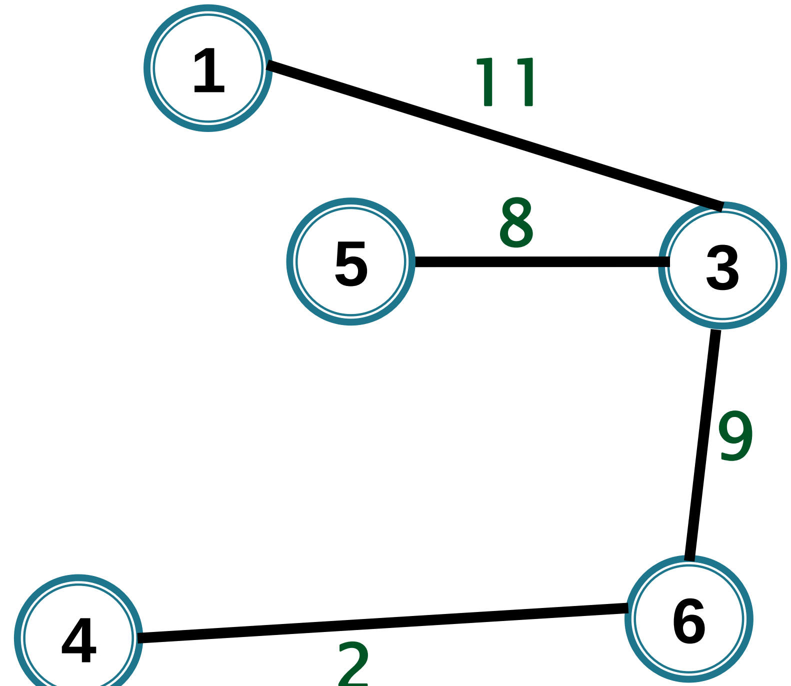
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]
Sel. 6:	[– ,	5/6 ,	– ,	2/6,	– ,	–]



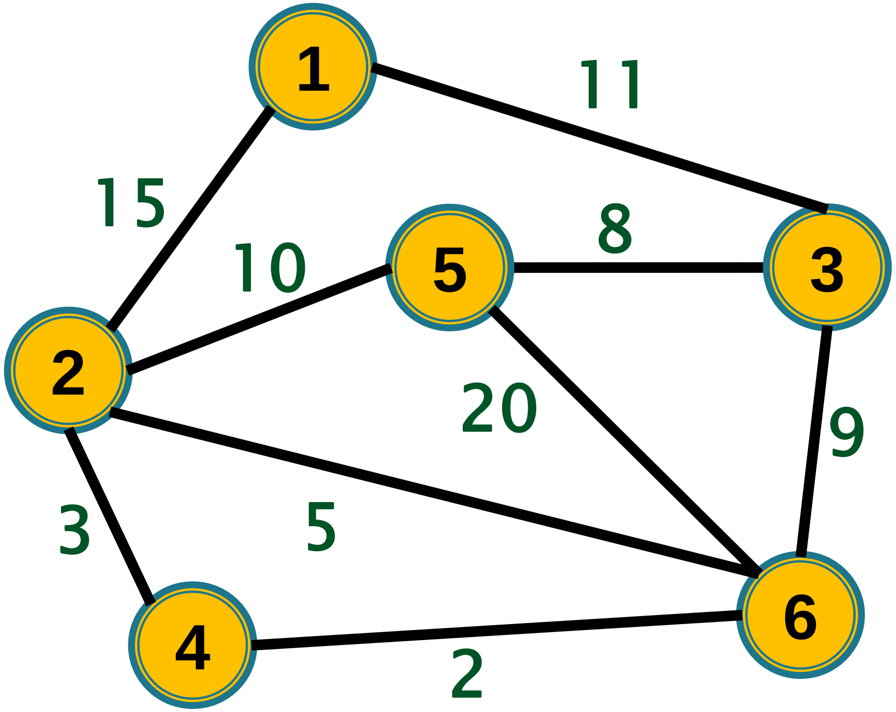


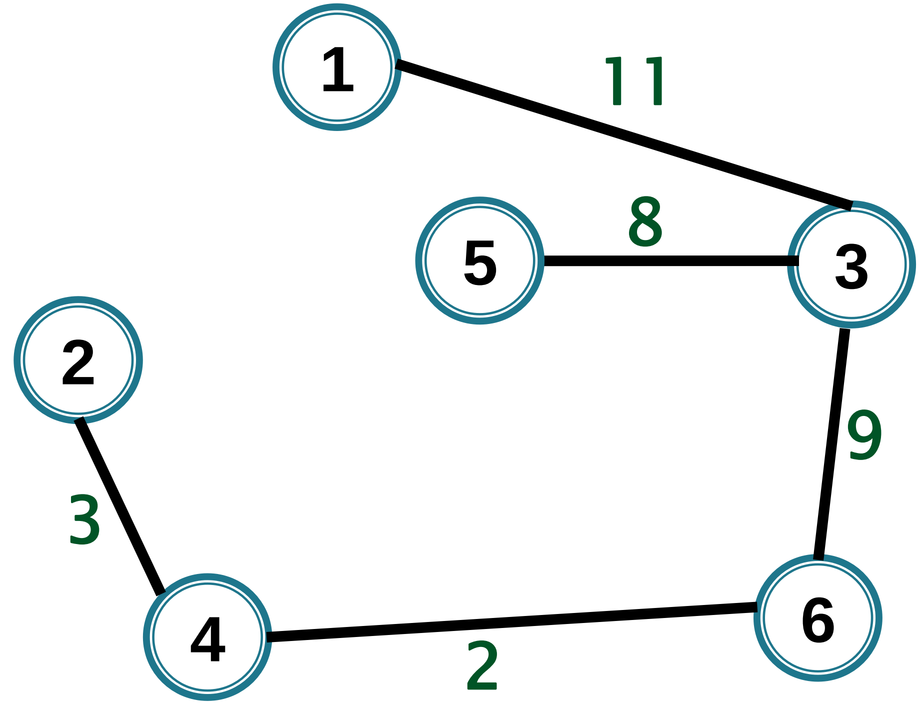
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]
Sel. 6:	[– ,	5/6 ,	– ,	2/6,	– ,	–]
Sel. 4:						



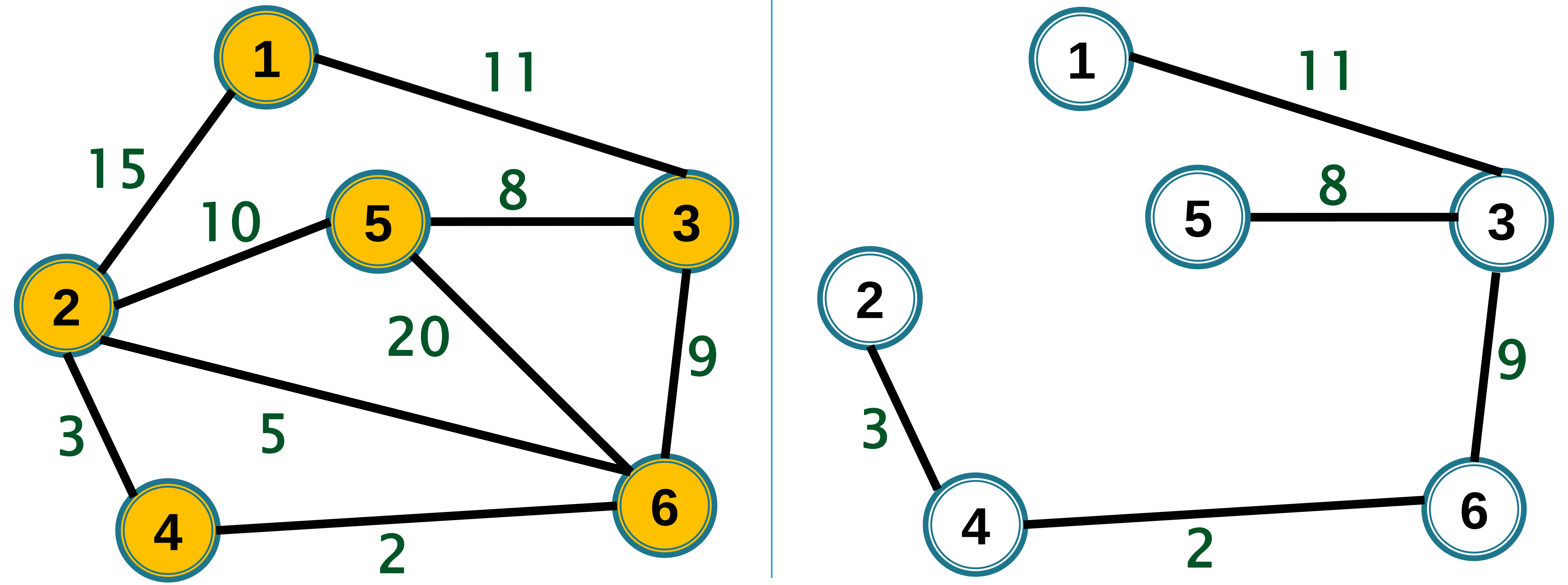


	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]
Sel. 6:	[– ,	5/6 ,	– ,	2/6,	– ,	–]
Sel. 4:	[– ,	3/4 ,	– ,	– ,	– ,	–]





	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]
Sel. 6:	[– ,	5/6 ,	– ,	2/6,	– ,	–]
Sel. 4:	[– ,	3/4 ,	– ,	– ,	– ,	–]
Sel. 2:						



	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]
Sel. 6:	[– ,	5/6 ,	– ,	2/6,	– ,	–]
Sel. 4:	[– ,	3/4 ,	– ,	– ,	– ,	–]
Sel. 2:	[– ,	– ,	– ,	– ,	– ,	–]

Prim(G, w, s)

```
pentru fiecare u∈V executa
      d[u] = ∞; tata[u]=0
 d[s] = 0
 inițializează Q cu V
 cat timp Q ≠ ∅ executa
        u=extrage vârf cu eticheta d minimă din Q
        pentru fiecare v adiacent cu u executa
              daca v∈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≠ s
```

Prim

Varianta 2 – memorarea vârfurilor din 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

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

$\Rightarrow$ **$O(n^2)$** mai eficient