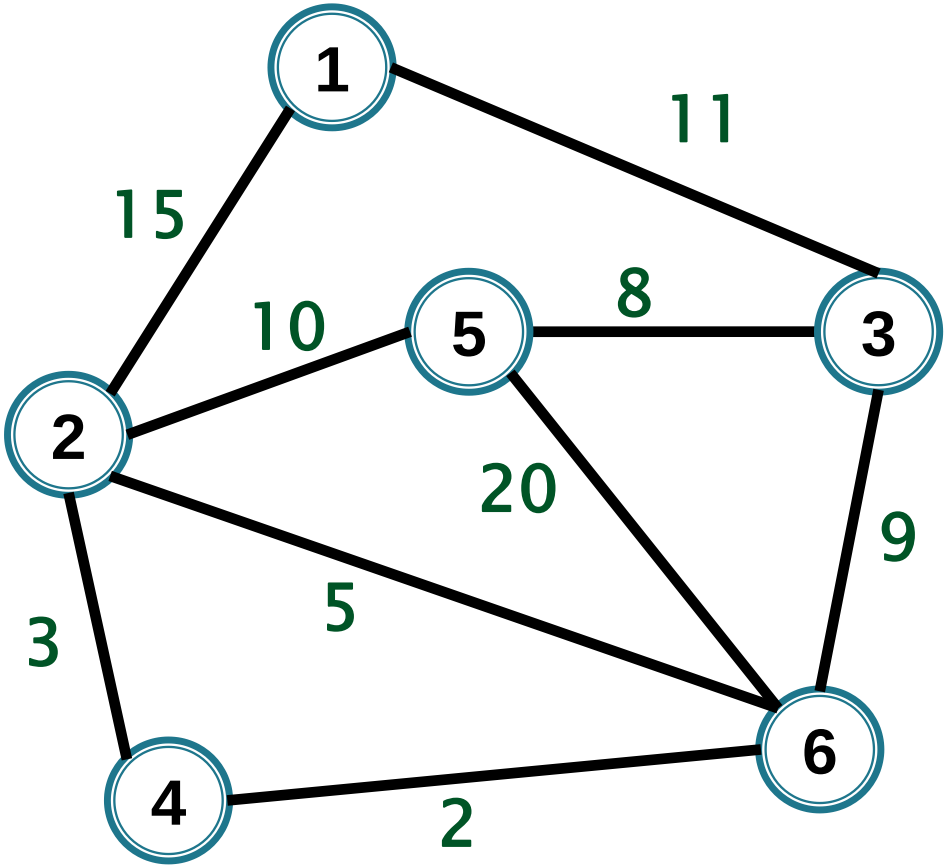
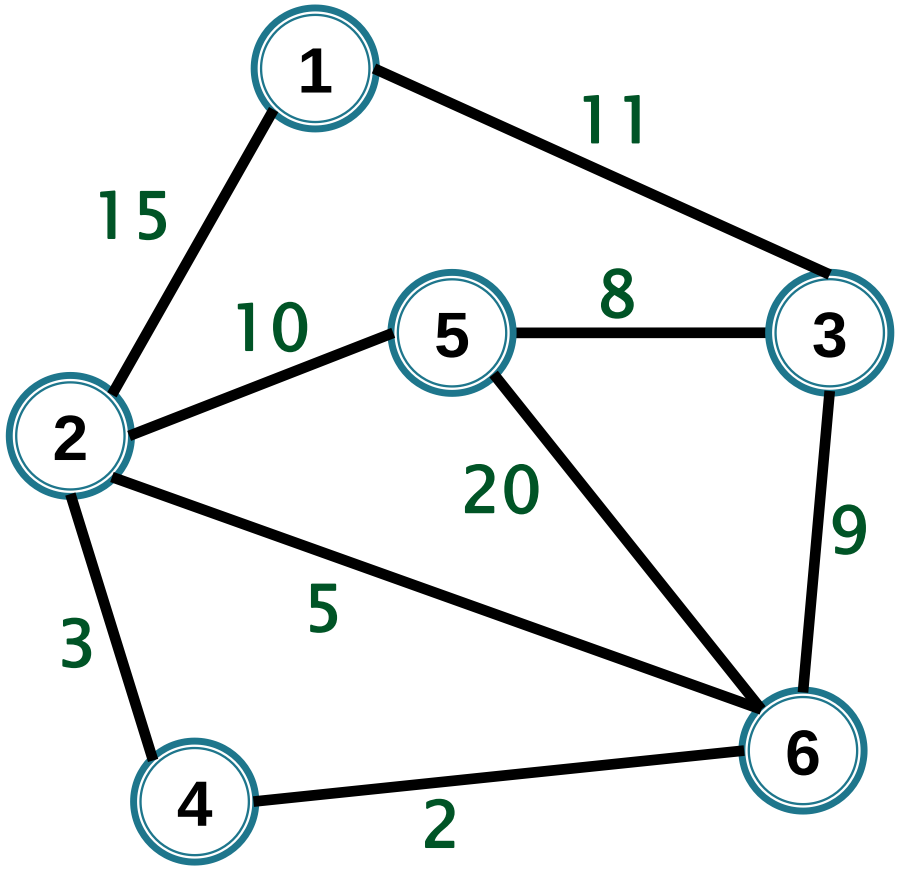
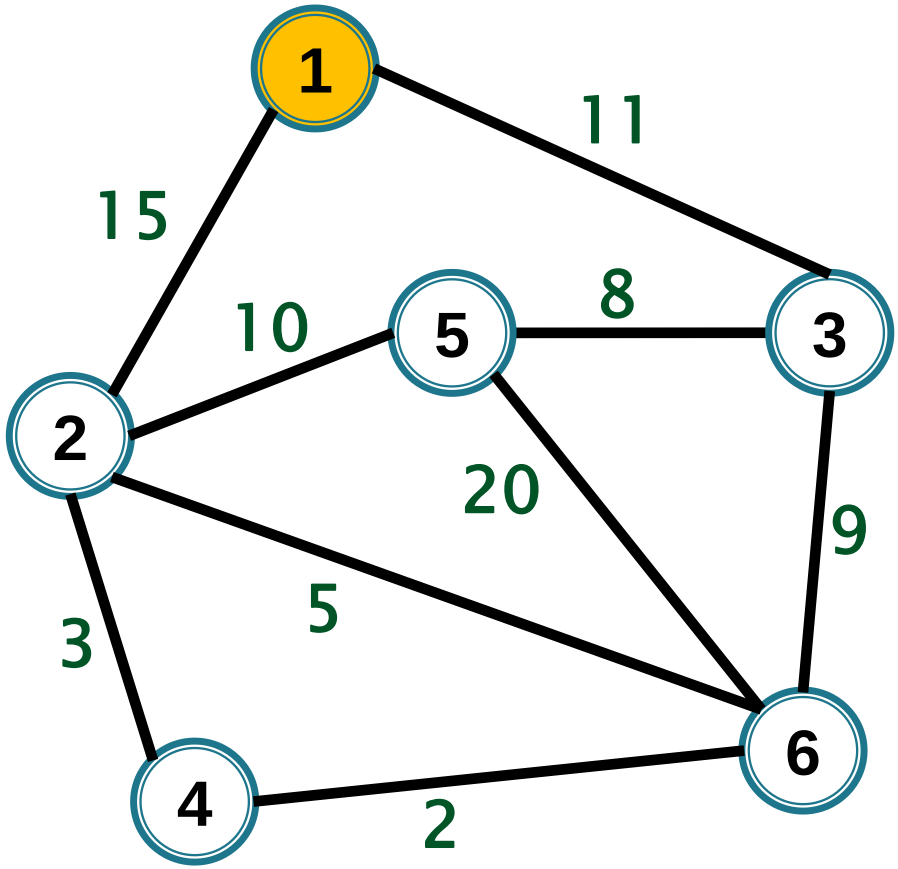
Algoritmul lui Prim
Implementare

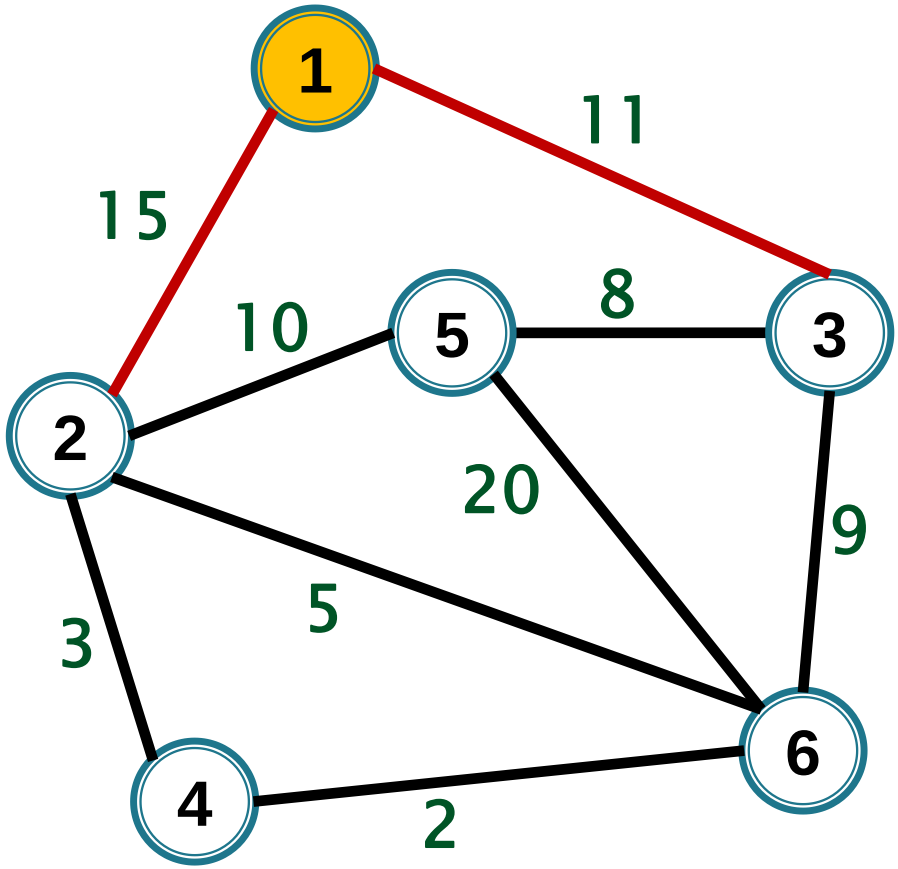
Exemplu – curs


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

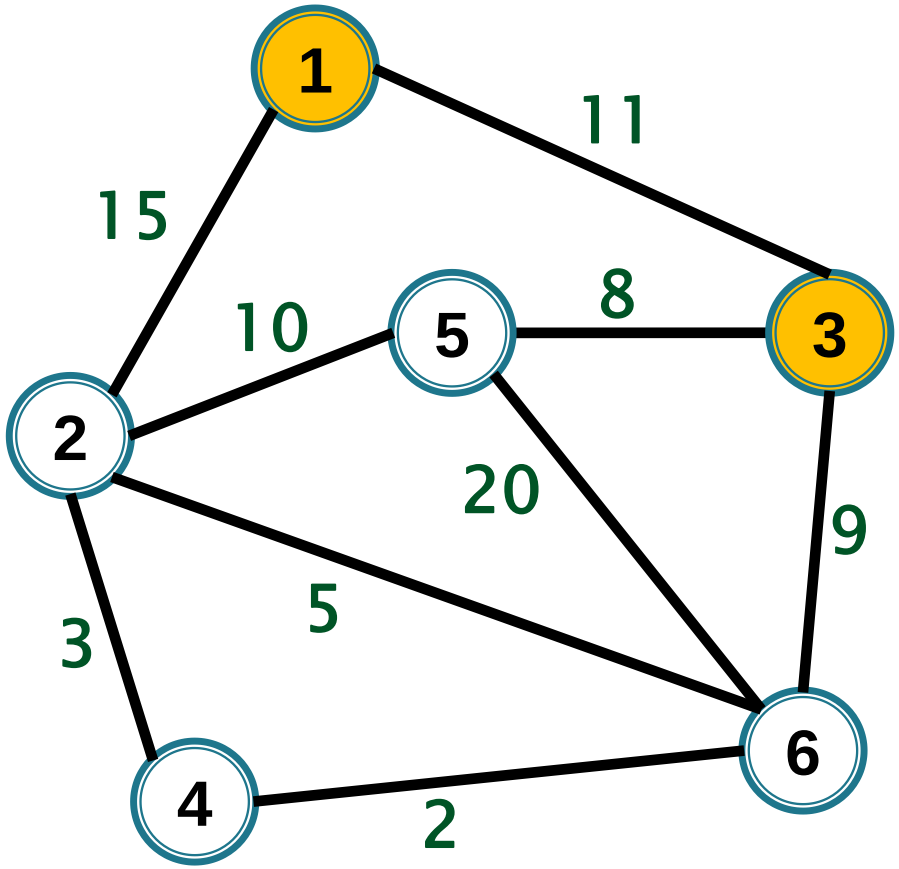

1
11
15
8
10
5
3
2
20
9
5
3
6
4
2

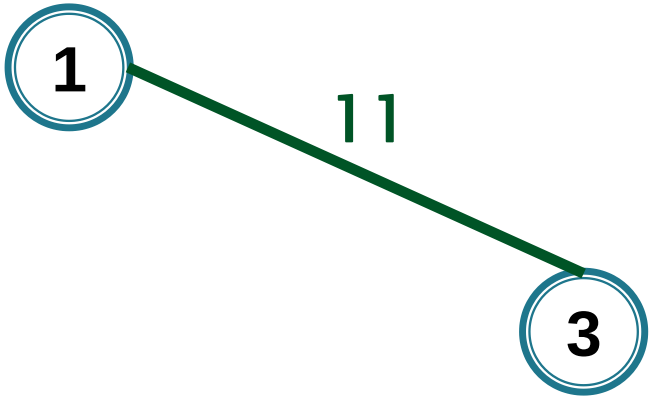


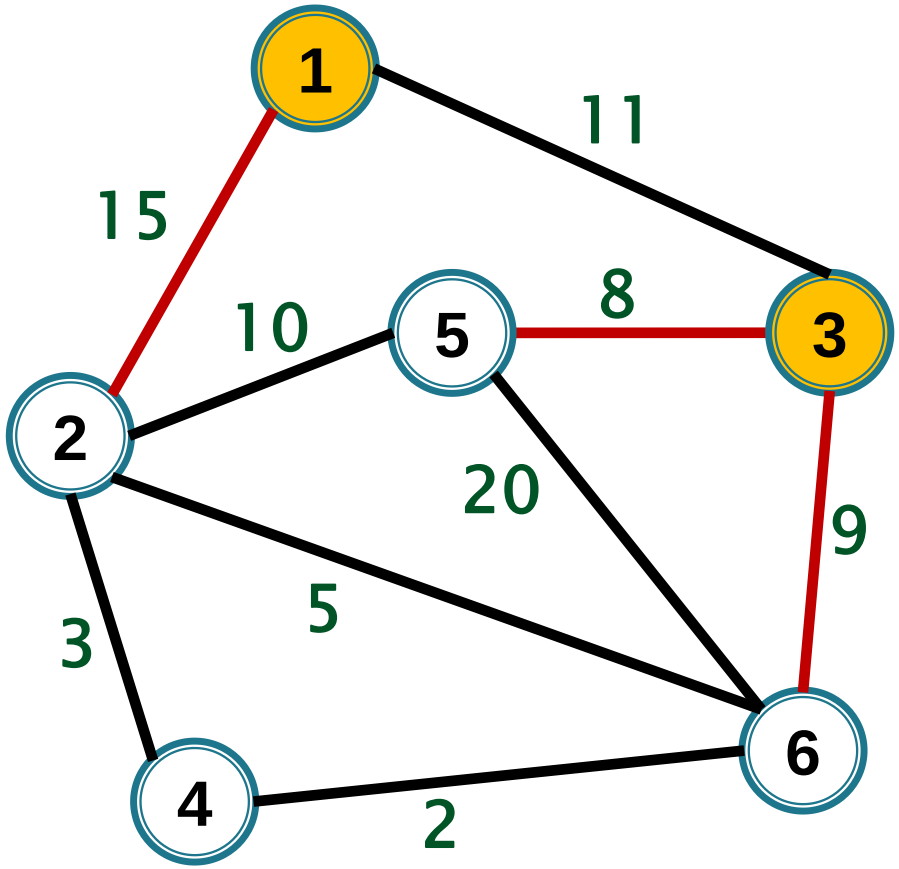
s = 1

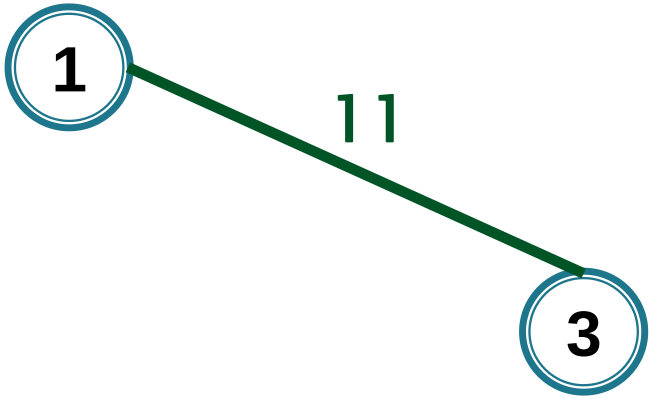

1
11
15
10
5
8
3
2
20
9
5
3
6
4
2

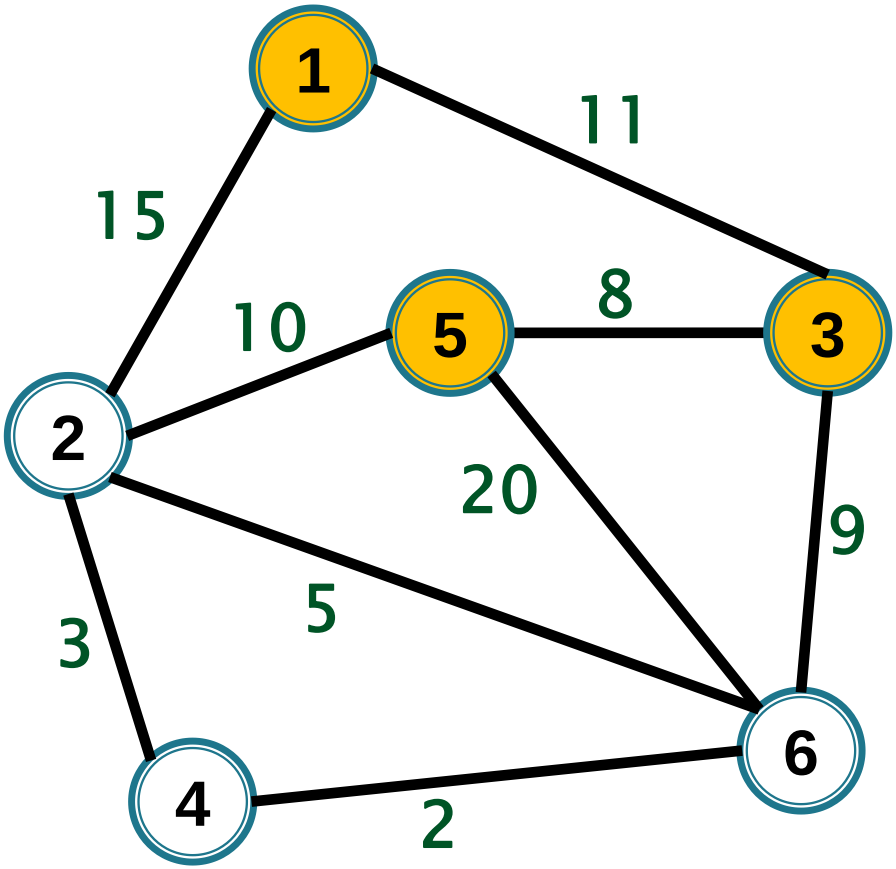

1

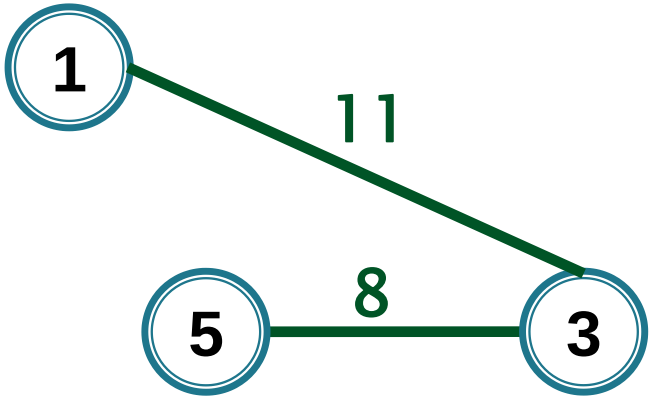

1
11
15
10
5
8
3
2
20
9
5
3
6
4
2

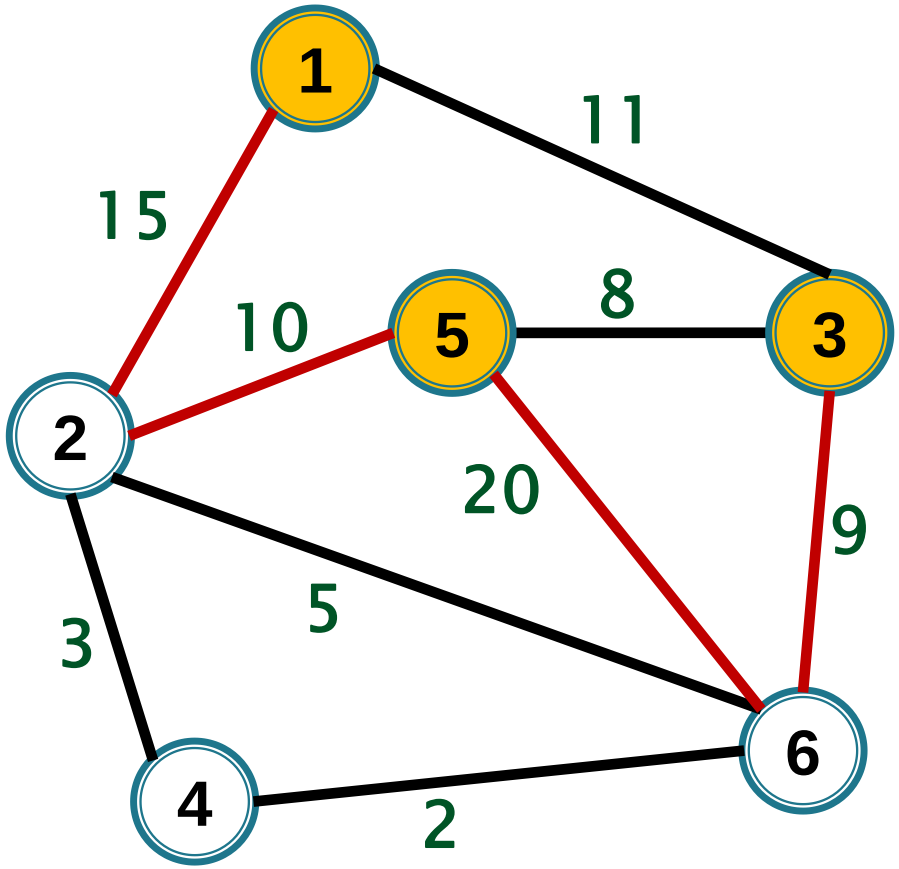

1
11
3

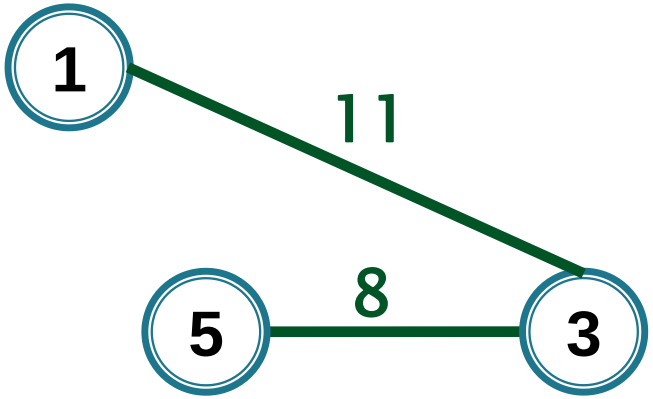

1
11
15
10
5
8
3
2
20
9
5
3
6
4
2

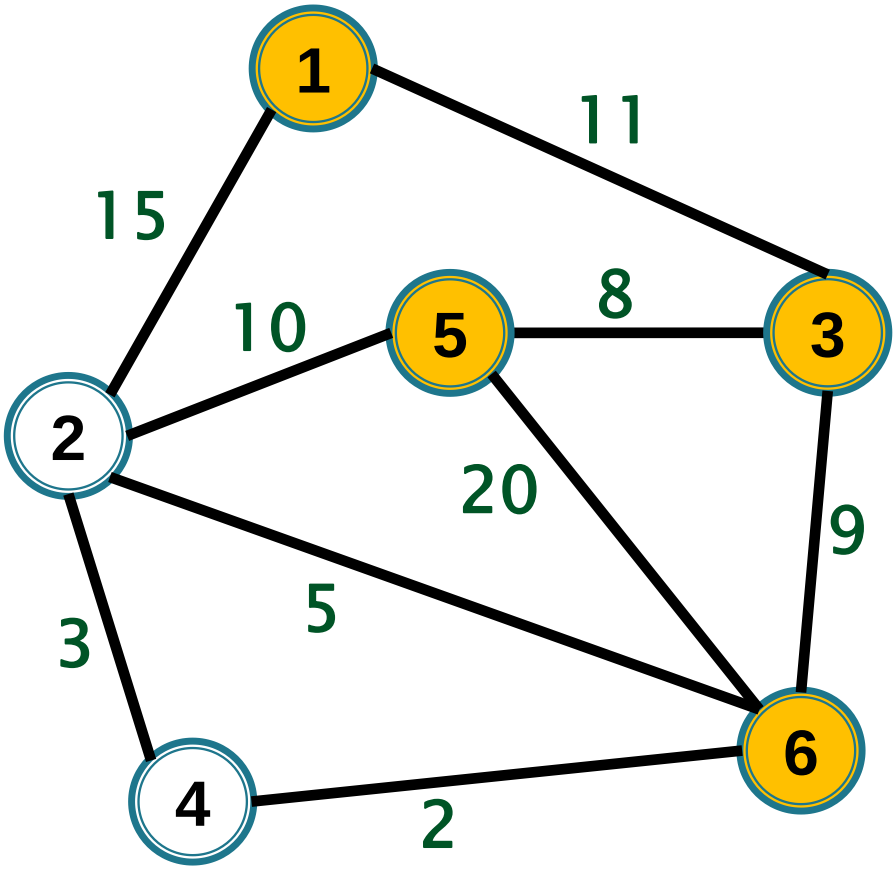

1
11
3

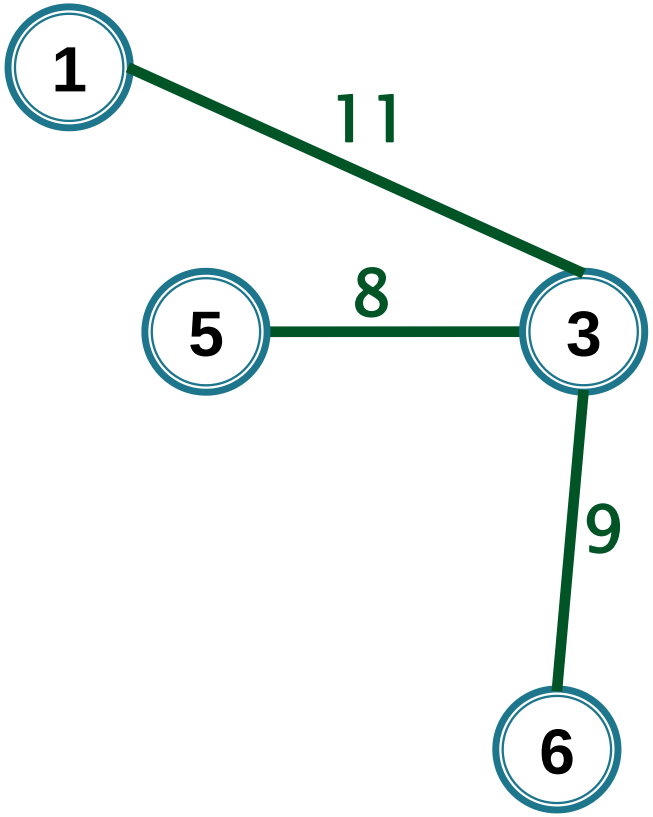

1
11
15
10
5
8
3
2
20
9
5
3
6
4
2

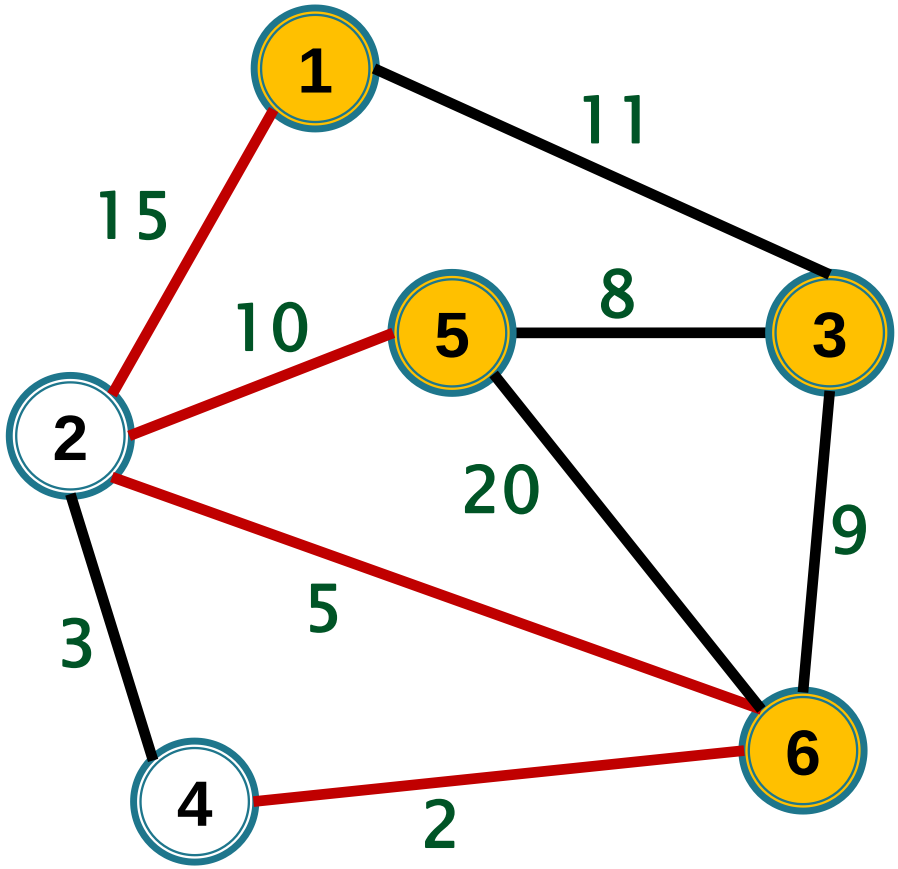

1
11
5
8
3

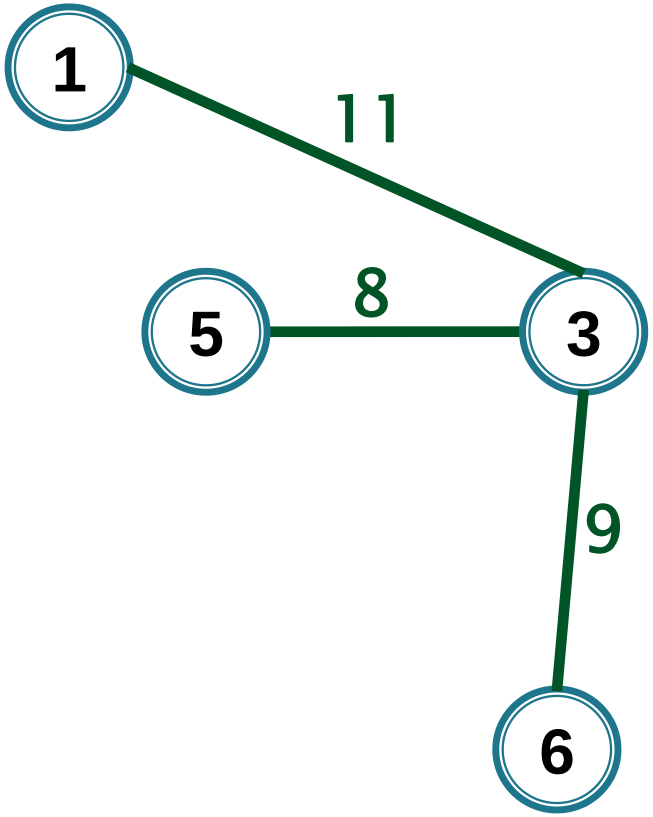

1
11
15
10
5
8
3
2
20
9
5
3
6
4
2

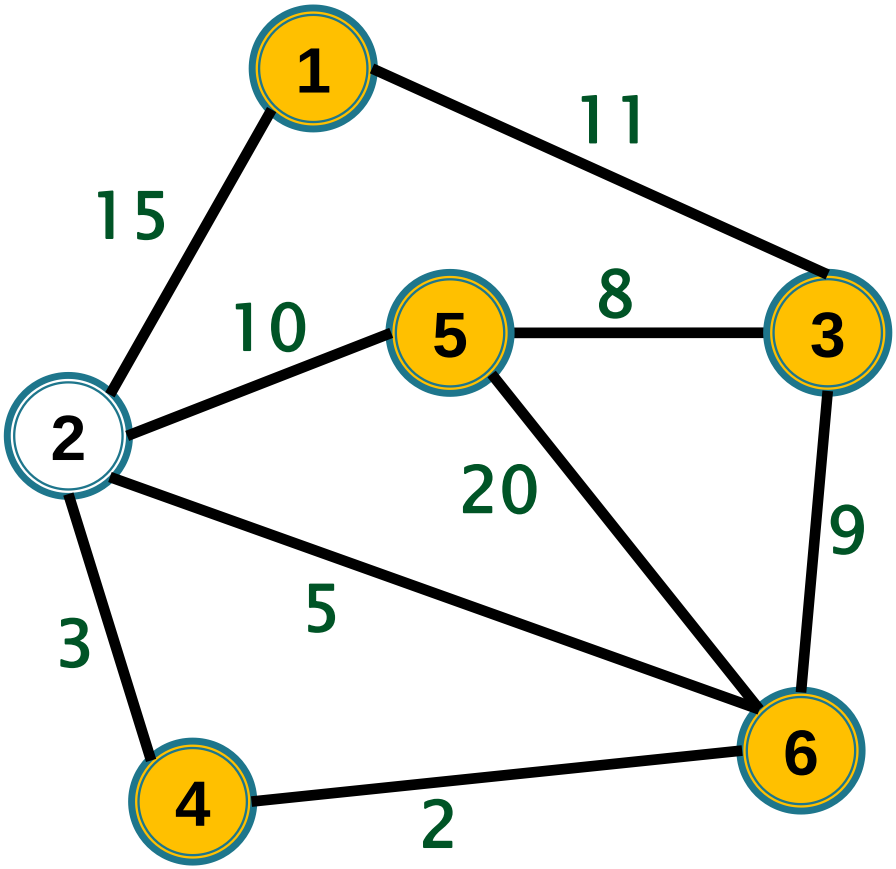

1
11
5
8
3

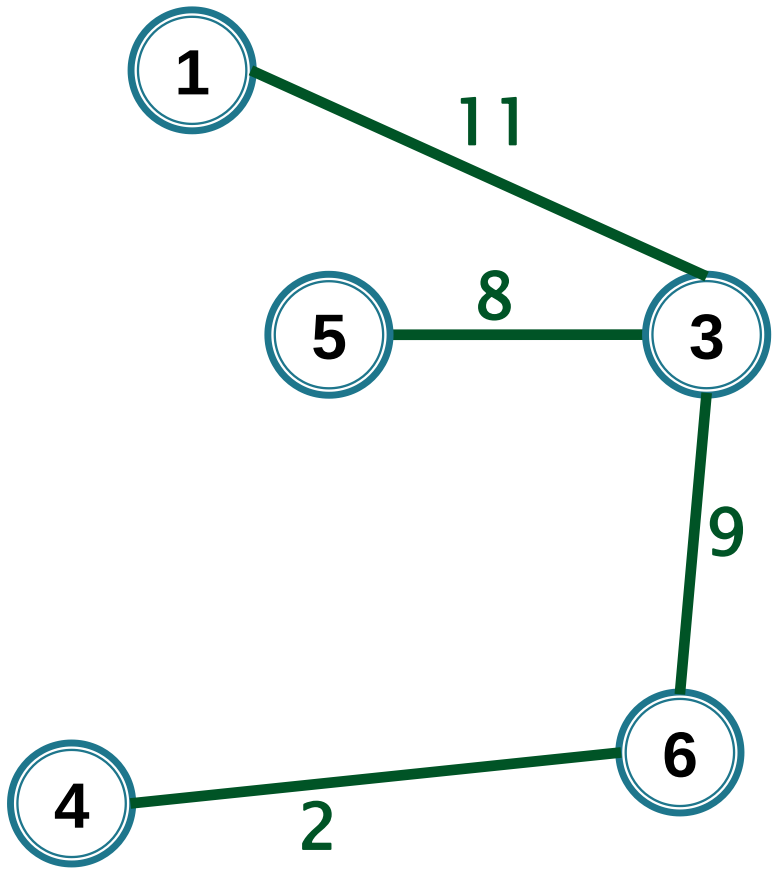

1
11
15
10
5
8
3
2
20
9
5
3
6
4
2

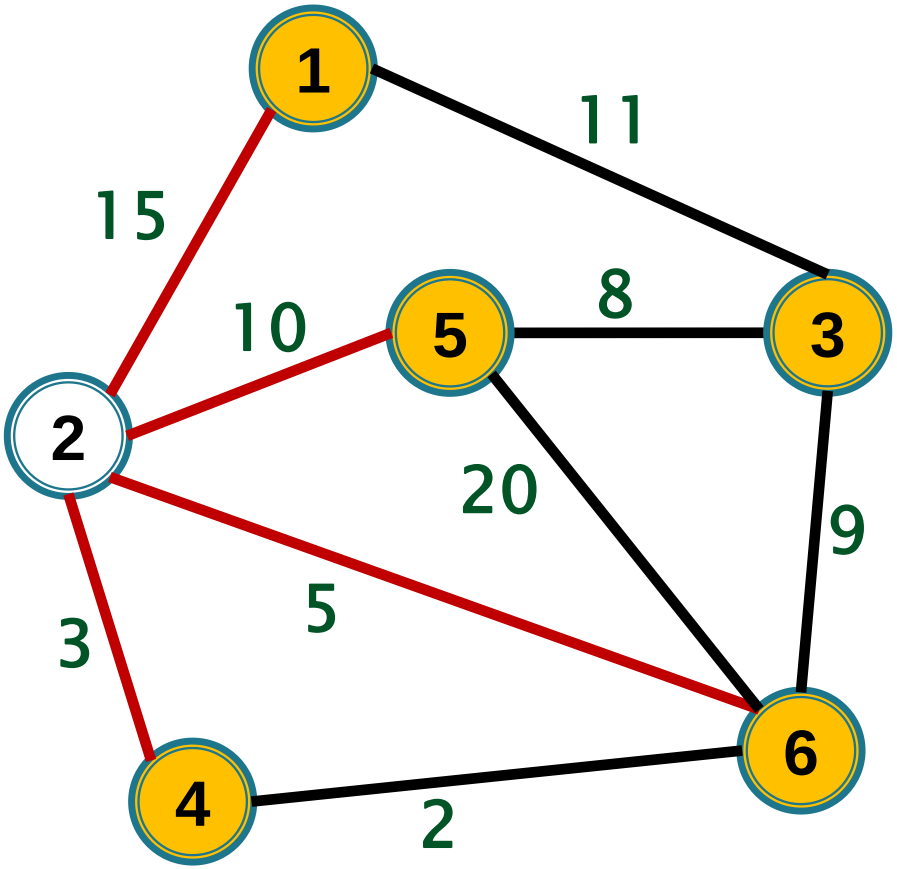

1
11
5
8
3
9
6

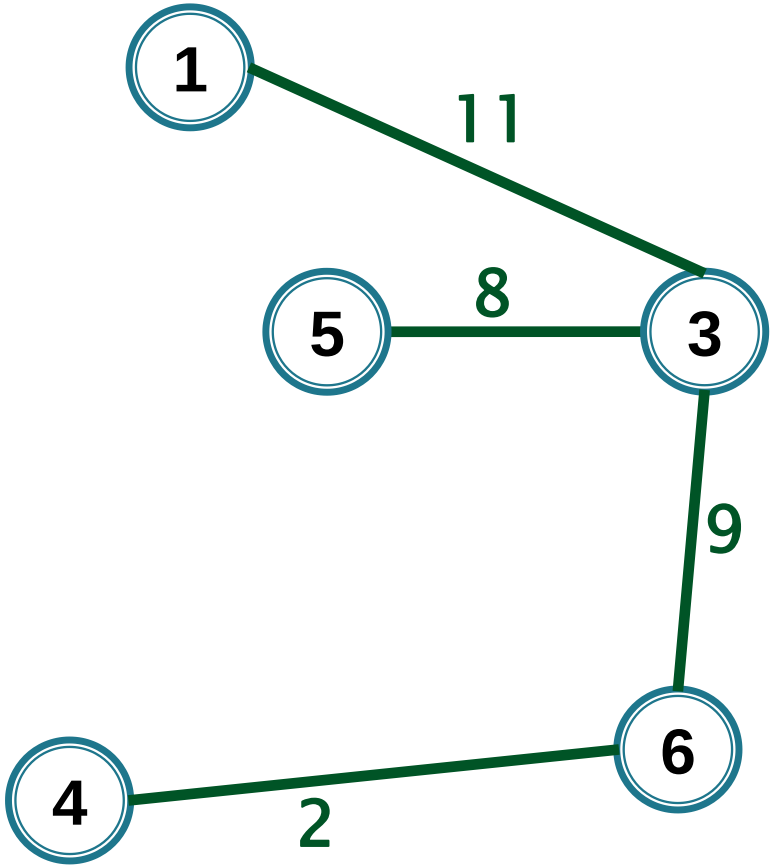

1
11
15
10
5
8
3
2
20
9
5
3
4
2
6

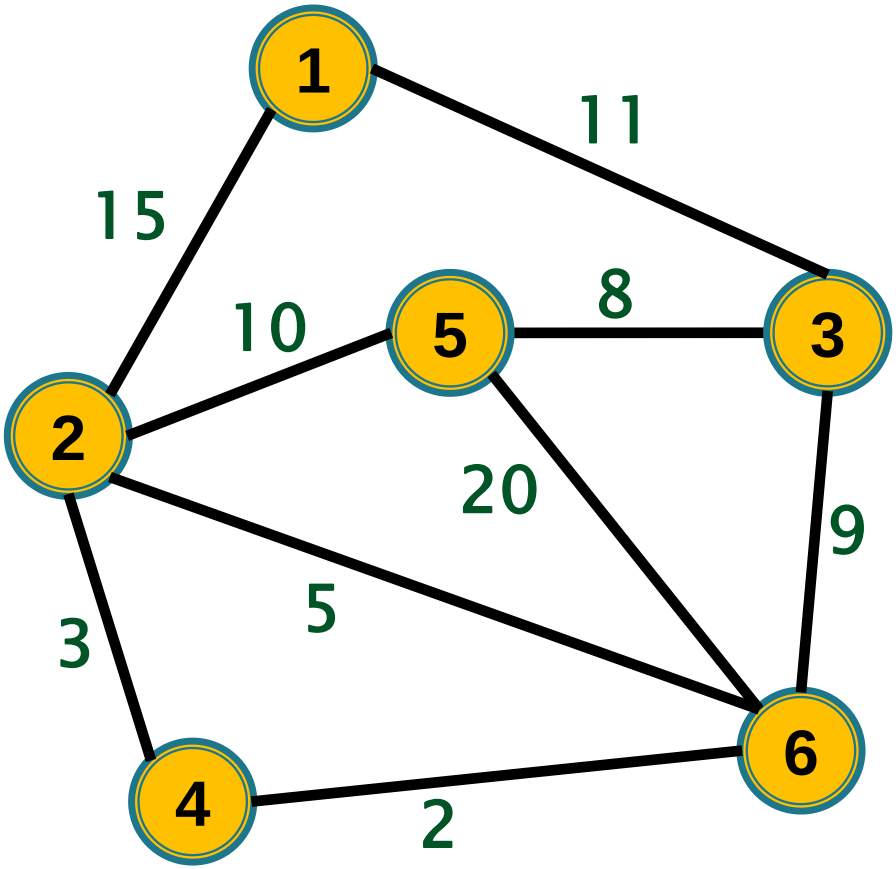

1
11
5
8
3
9
6

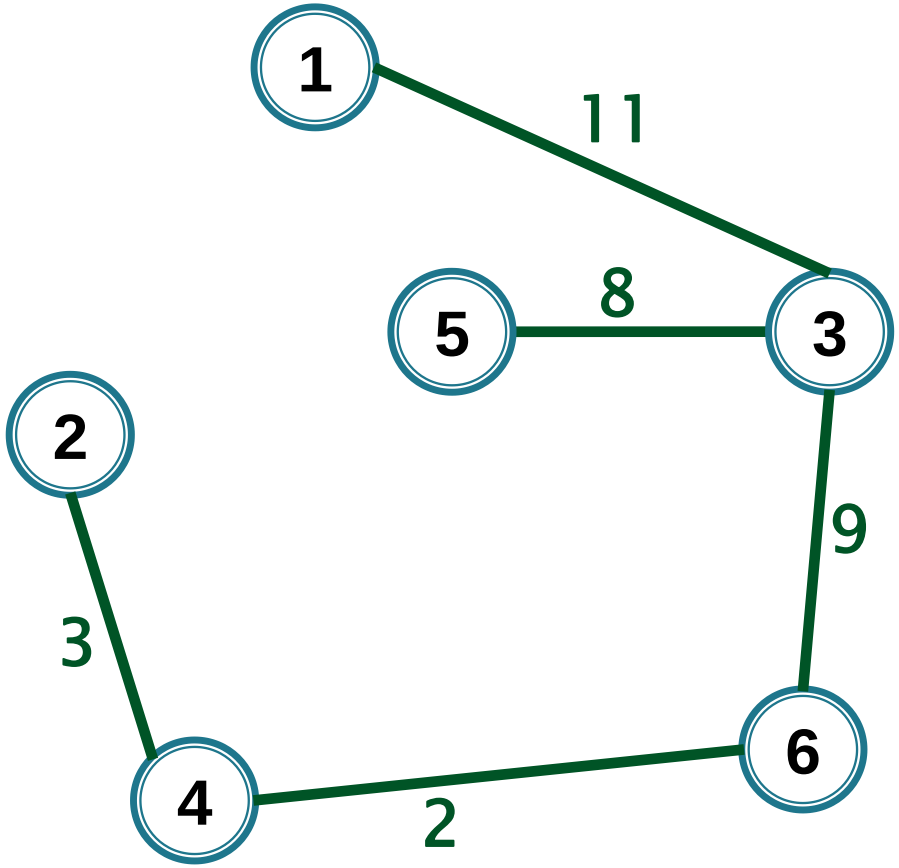

1
11
15
10
5
8
3
2
20
9
5
3
6
4
2

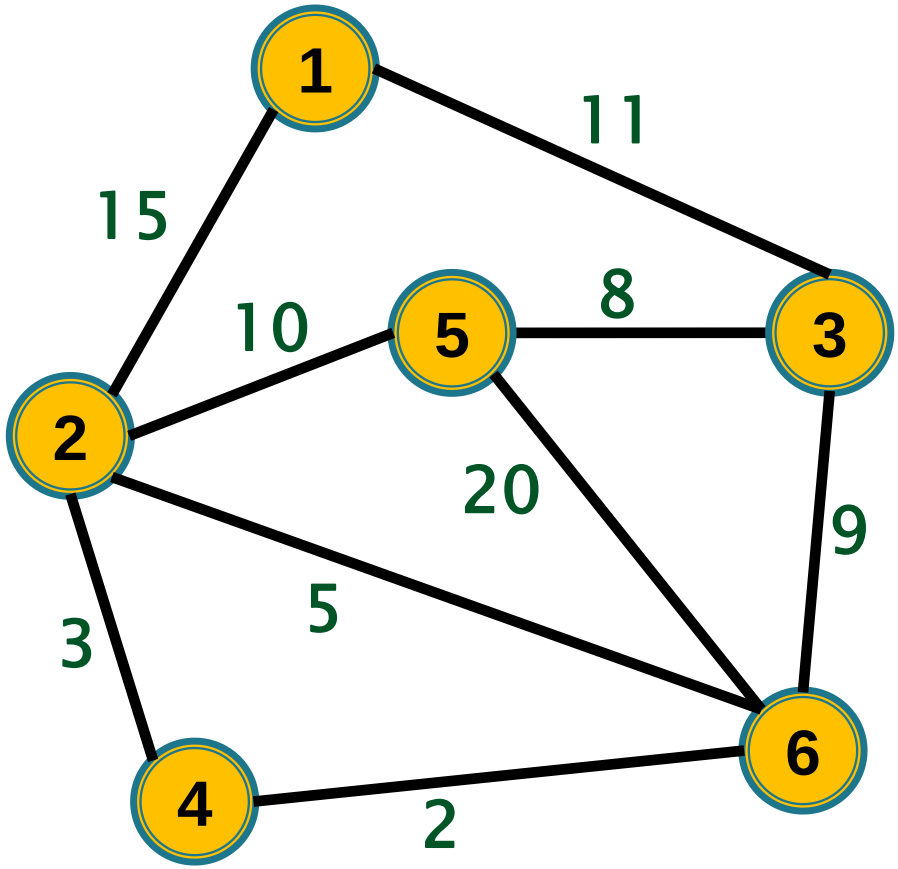

1
11
5
8
3
9
6
4
2

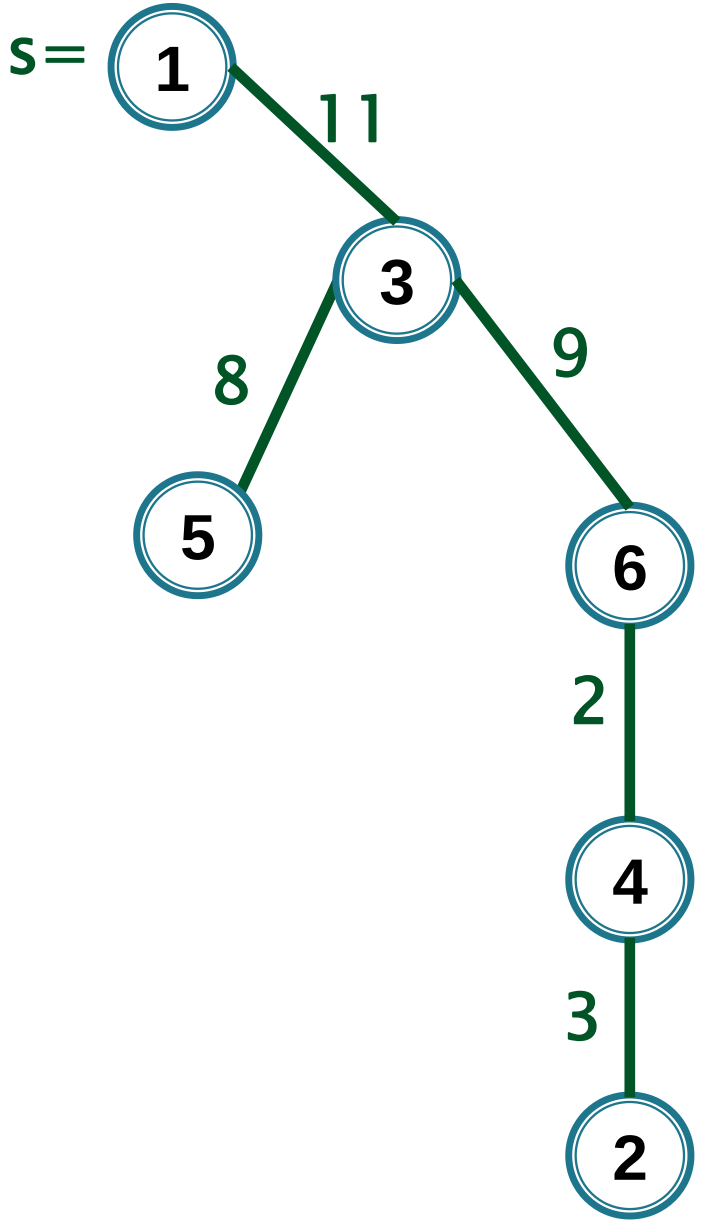

1
11
15
10
5
8
3
2
20
9
5
3
6
4
2


1
11
5
8
3
9
6
4
2


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


1
11
5
8
3
2
9
3
6
4
2


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


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

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

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

Implementare Prim

Asociem fiecărui vârf următoarele informaţii (etichete):

- **d[u]** = costul minim al unei muchii de la u la un vârf selectat deja în arbore
-

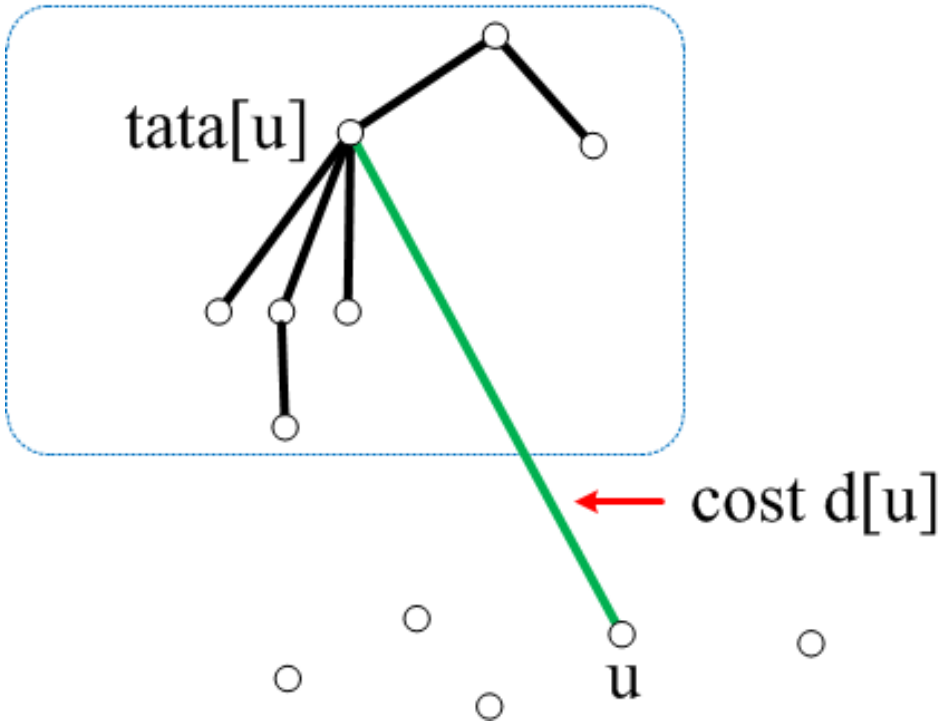
Implementare Prim

Asociem fiecărui vârf următoarele informaţii (etichete):

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



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)**

Implementare Prim

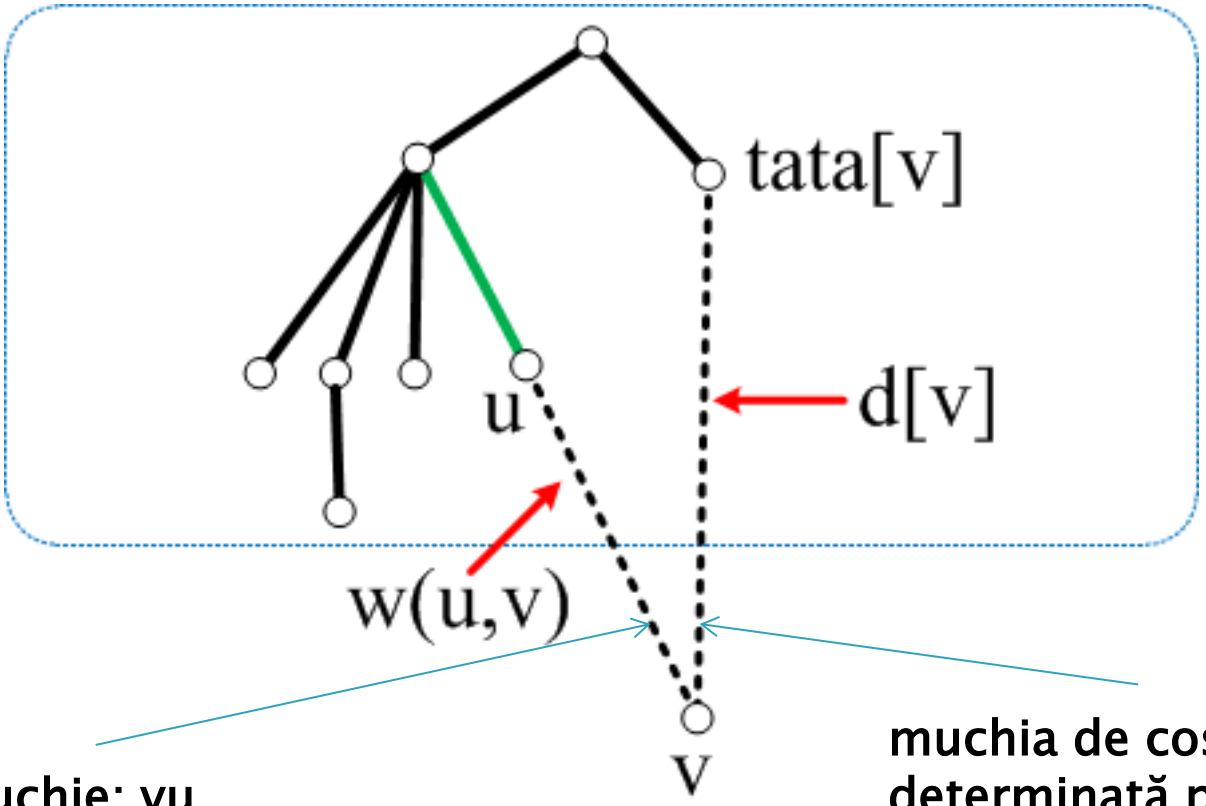
Atunci algoritmul se modifică astfel:

- **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:

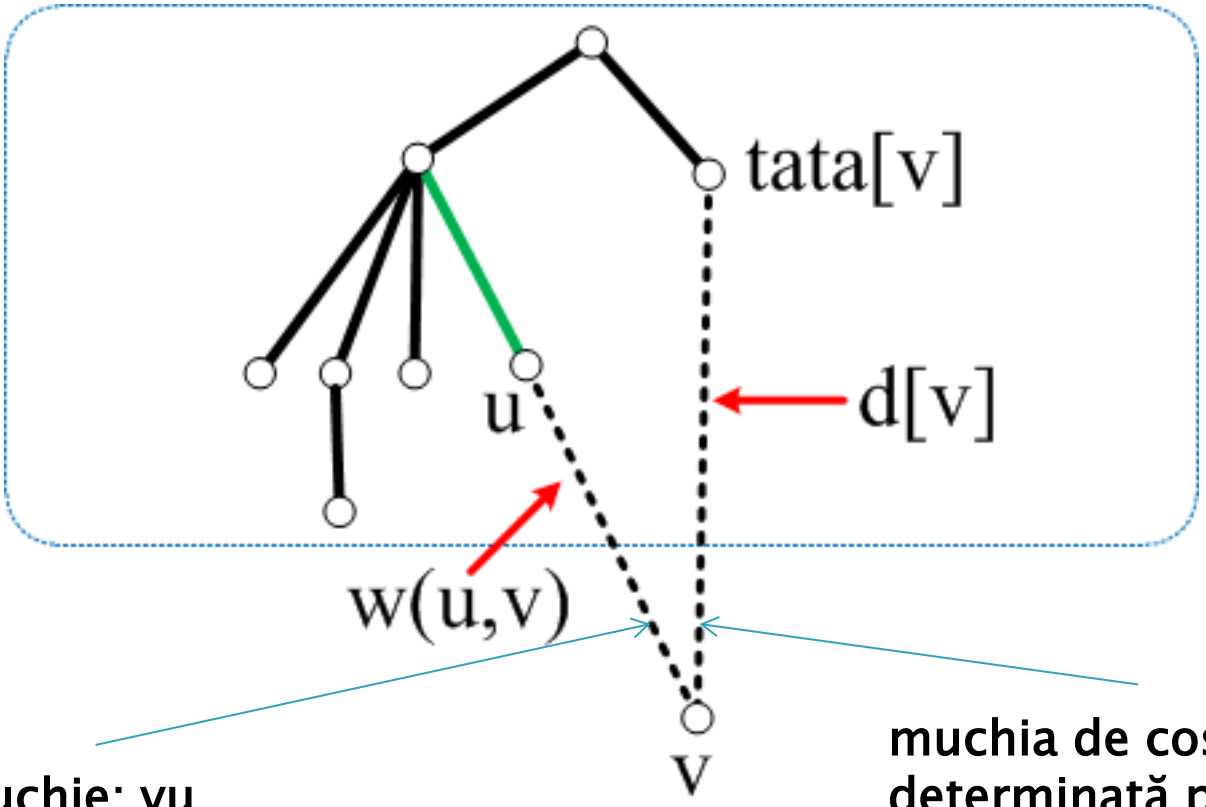
Prim



noua muchie: vu

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

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≠ s

Prim

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

Prim

- s- vârful de start
- **inițializează Q cu V**
- **pentru fiecare u∈V executa**

```
    d[u] = ∞; tata[u]=0
d[s] = 0
```

Prim

- s- vârful de start
- **inițializează Q cu V**
- **pentru fiecare u∈V executa**

  d[u] = ∞; tata[u]=0

  d[s] = 0
- **cat timp Q ≠ ∅ executa⇔** pentru i = 1, n -1

Prim

- s- vârful de start
- **inițializează Q cu V**
- **pentru fiecare u∈V executa**

  d[u] = ∞; tata[u]=0

  d[s] = 0
- **cat timp Q ≠ ∅ executa**

  extrage un vârf u∈Q cu eticheta d[u] minimă

Prim

- s- vârful de start
- inițializează Q cu V
- pentru fiecare u∈V executa

```
        d[u] = ∞; tata[u]=0
    d[s] = 0
```

- cat timp Q ≠ ∅ executa

```
        extrage un vârf u∈Q cu eticheta d[u] minimă
        pentru fiecare uv∈E executa
                daca v∈Q si w(u,v)<d[v] atunci
                    d[v] = w(u,v)
                    tata[v] = u
```

Prim

- s- vârful de start
- **inițializează Q cu V**
- **pentru fiecare u∈V executa**

  d[u] = ∞; tata[u]=0

  d[s] = 0
- **cat timp Q ≠ ∅ executa**

  extrage un vârf u∈Q cu eticheta d[u] minimă

  pentru fiecare uv∈E executa

  daca v∈Q si w(u,v)<d[v] atunci

  d[v] = w(u,v)

  tata[v] = u
- scrie (u, tata[u]), pentru u≠ s

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

- Inițializare
- n * extragere vârf minim
- actualizare etichete vecini

Prim

Complexitate

Varianta 1 – cu vector de vizitat

- **Inițializări** ->
- **n * extragere vârf minim** ->
- **actualizare etichete vecini** ->

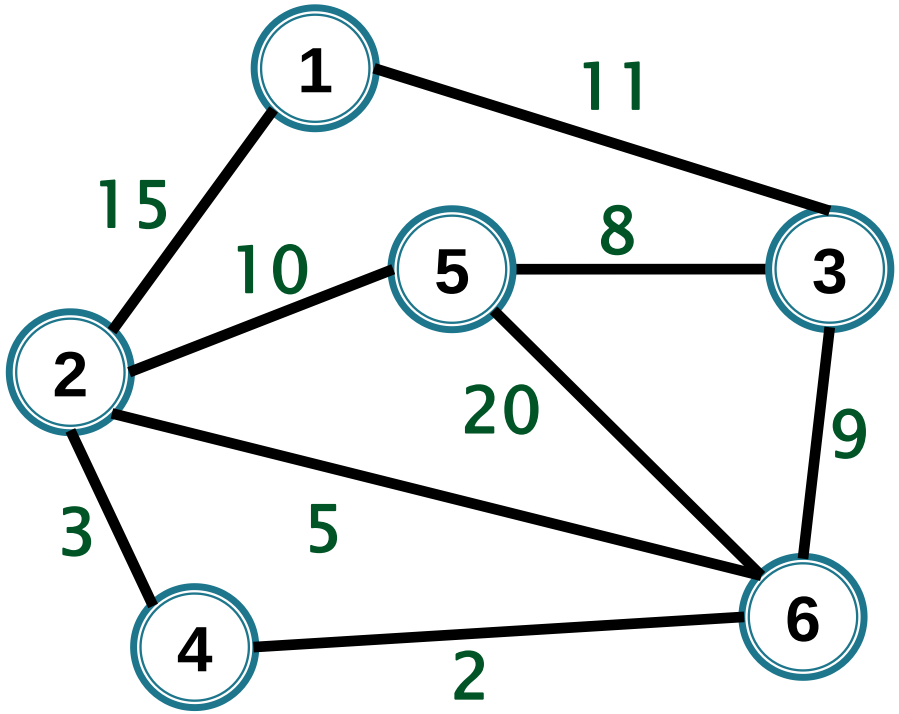
Prim

Complexitate

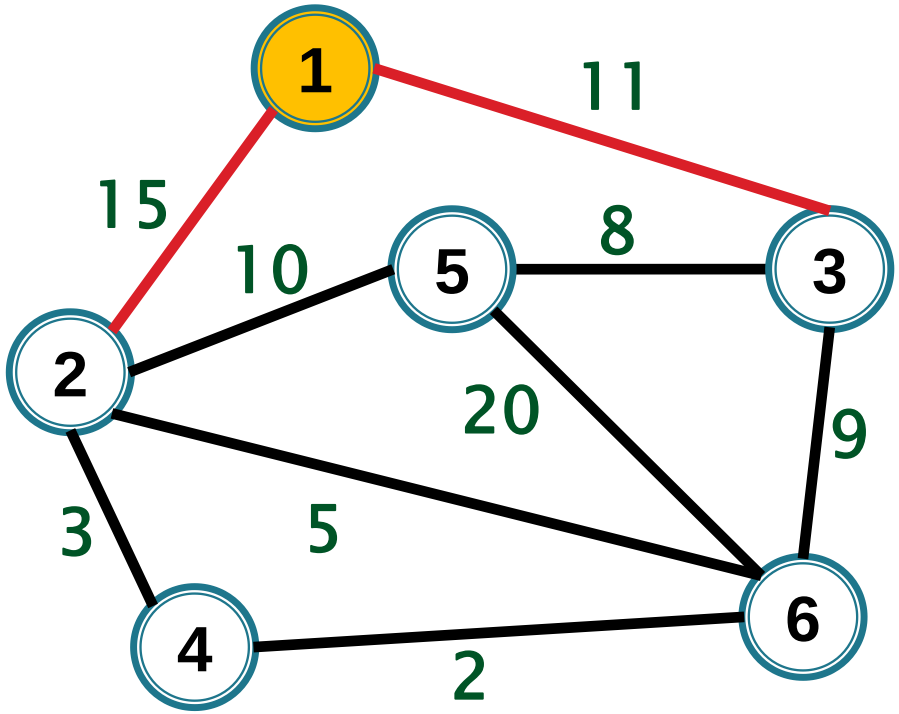
Varianta 1 – cu vector de vizitat

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

$O(n^2)$

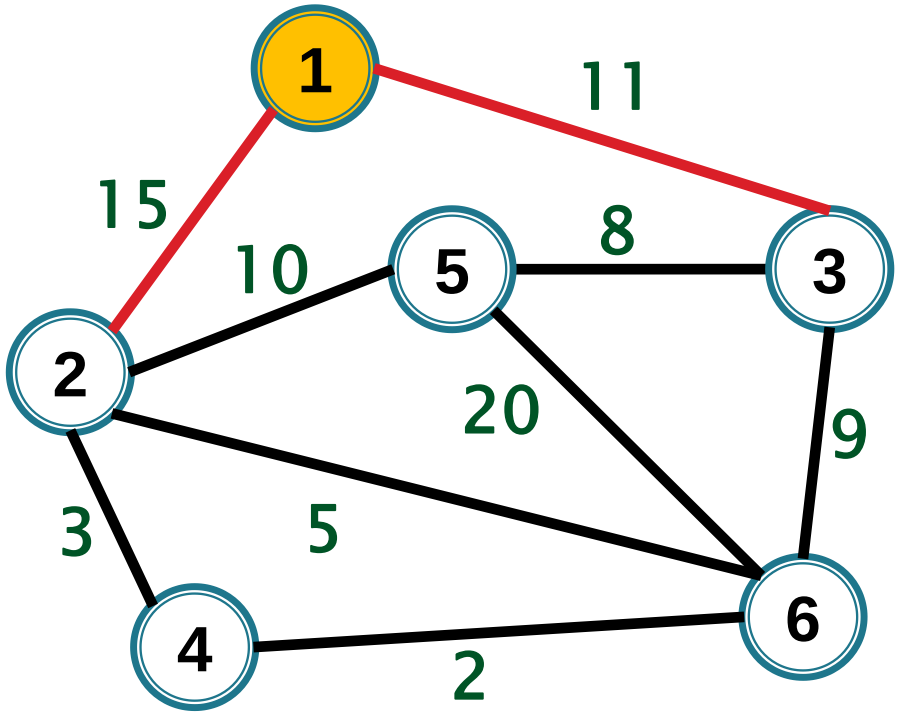


1	2	3	4	5	6
d/tata= [0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]



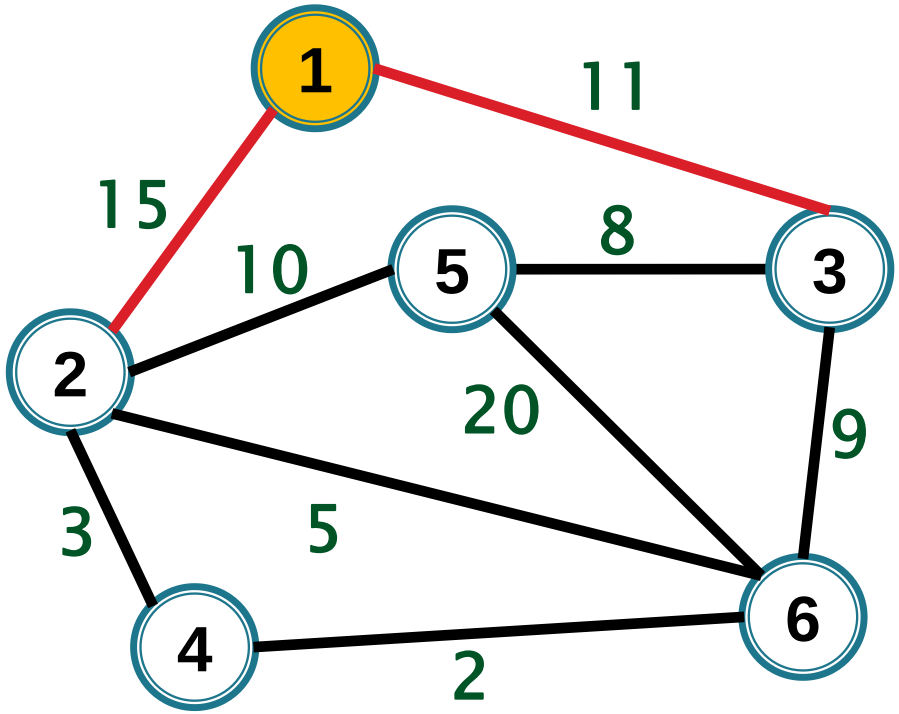
1	2	3	4	5	6
[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]

Sel. 1:



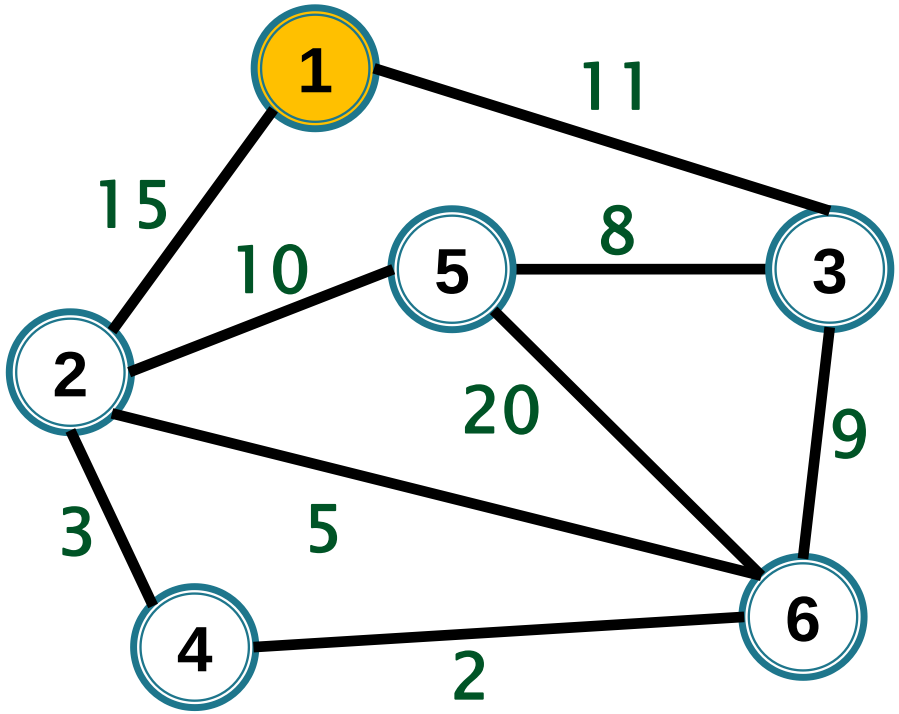
1	2	3	4	5	6
[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]

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

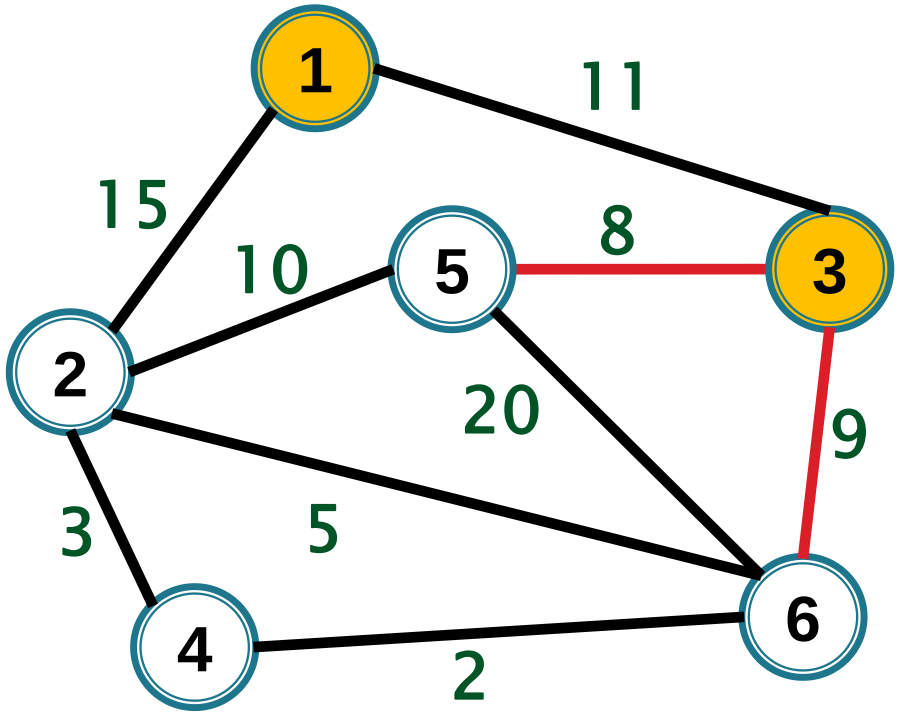


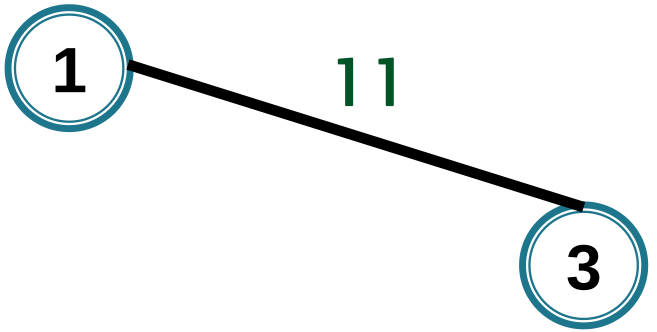
1	2	3	4	5	6
[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]

Sel. 1: actualizăm etichetele vecinilor

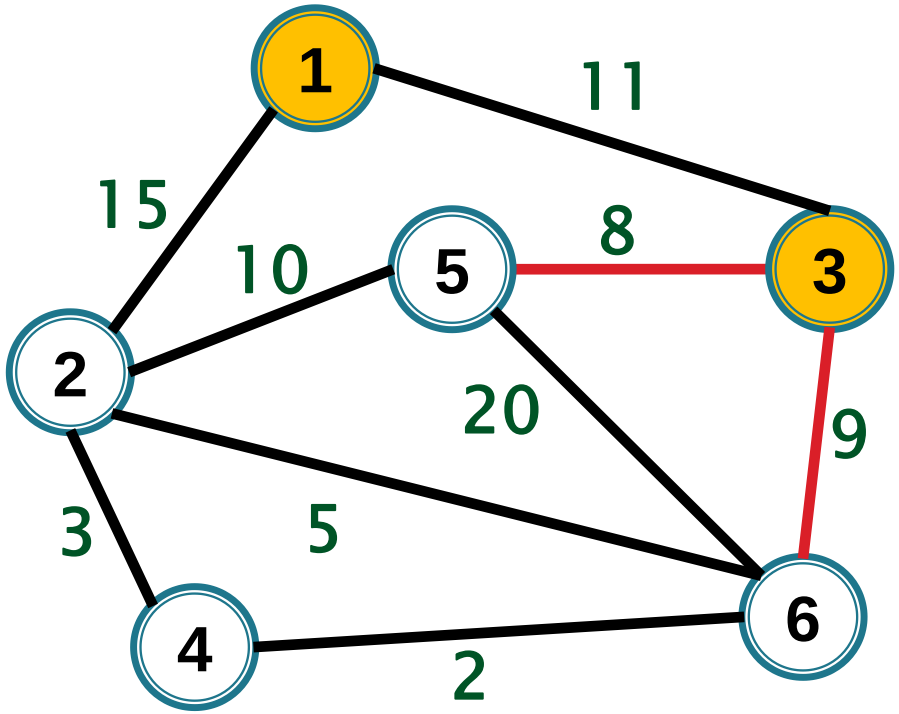


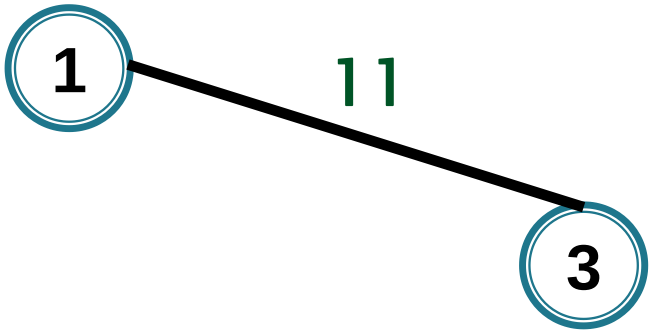
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]



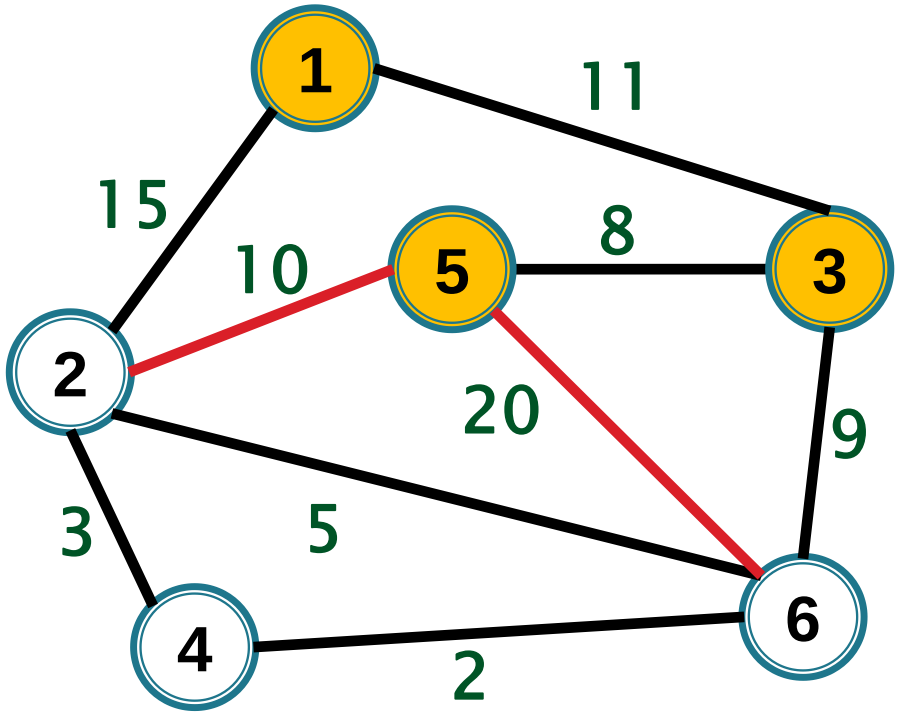


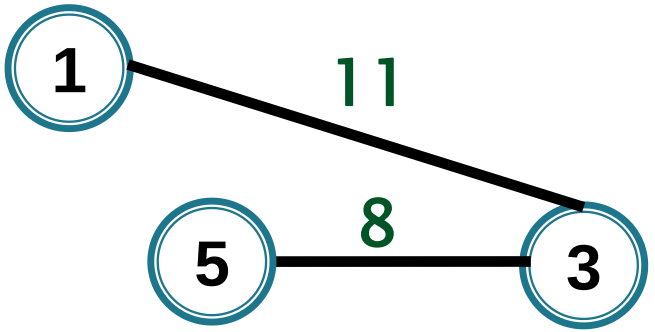
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:						



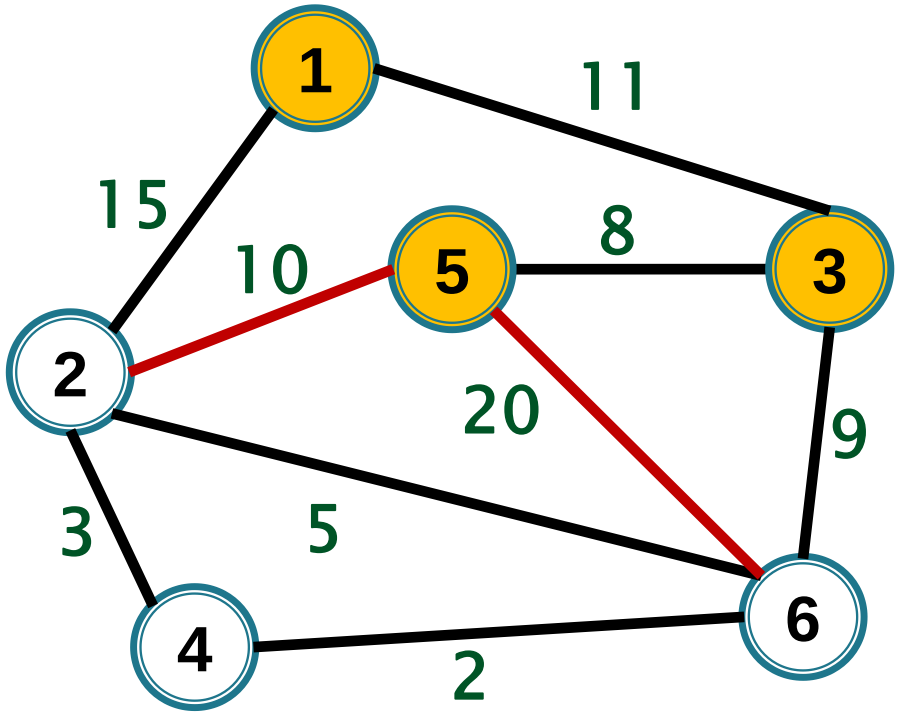


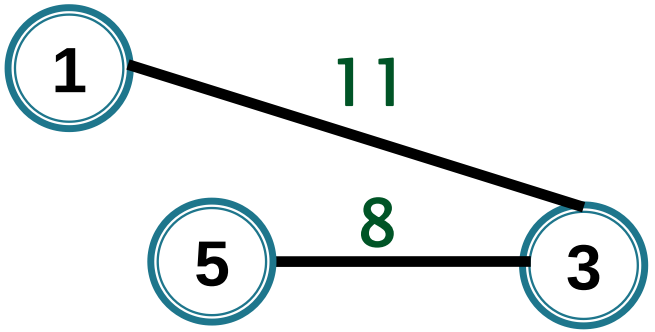
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]



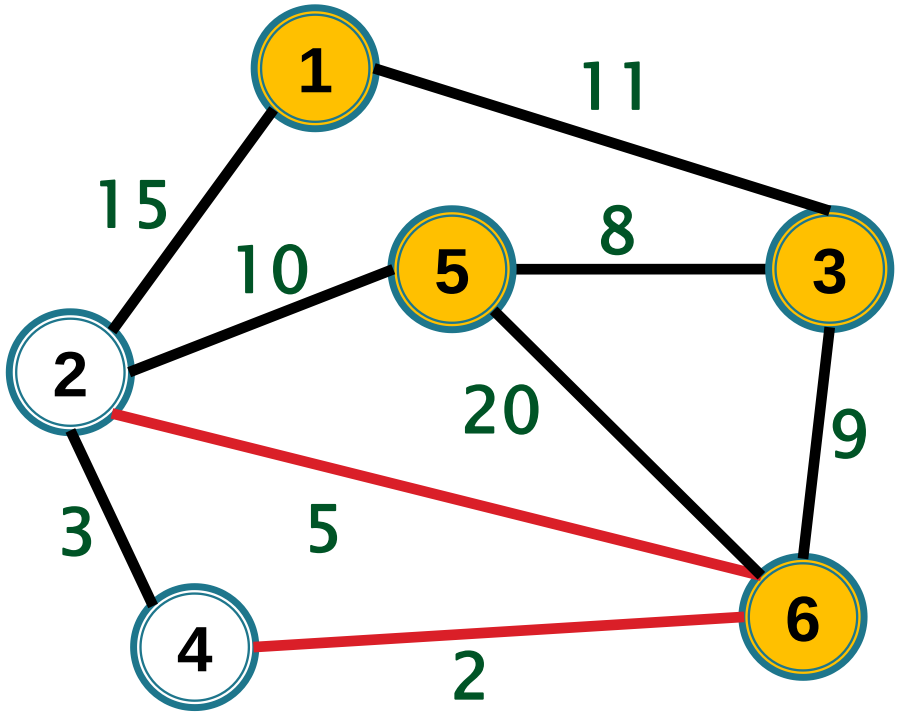


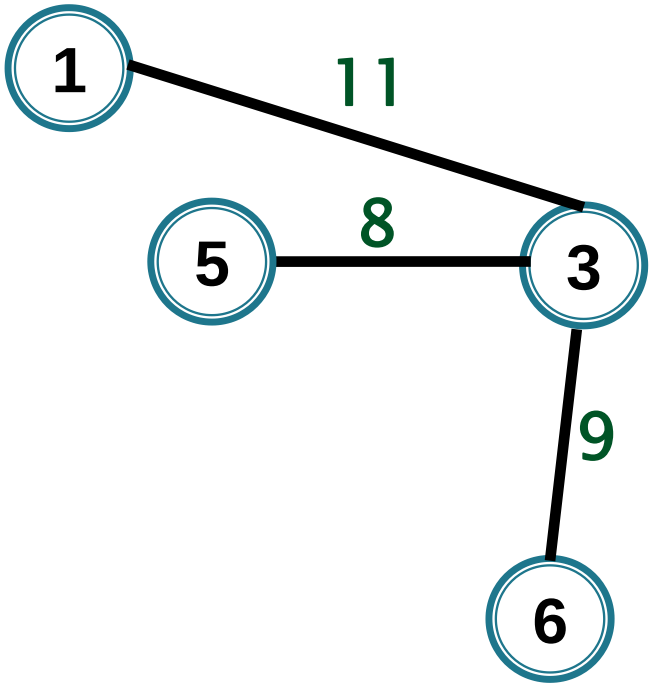
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:						



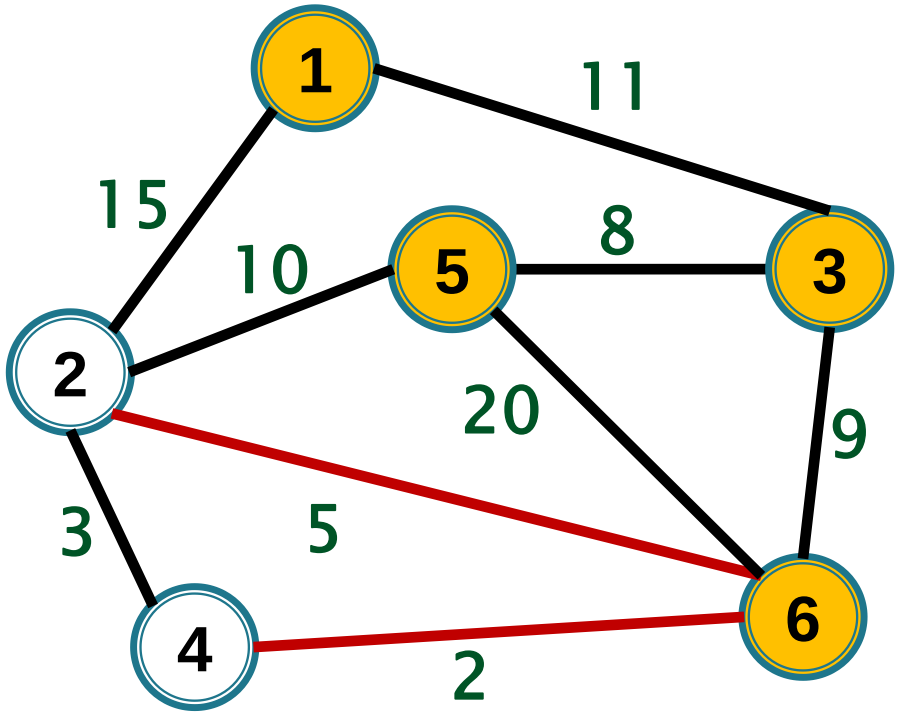


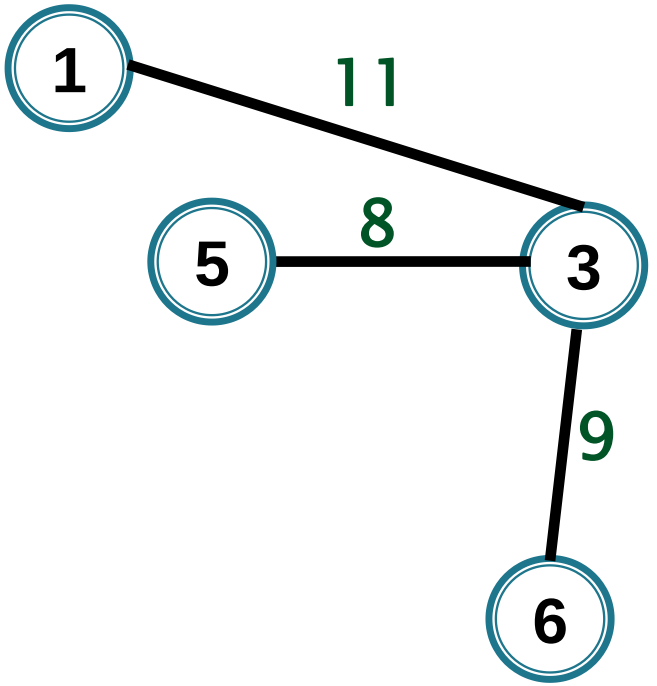
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]



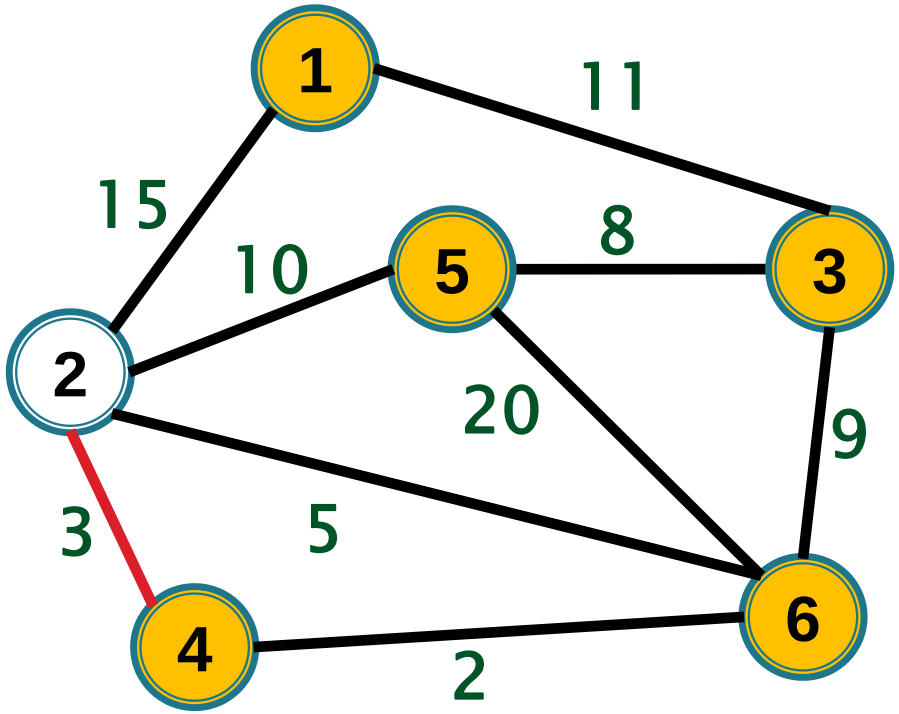


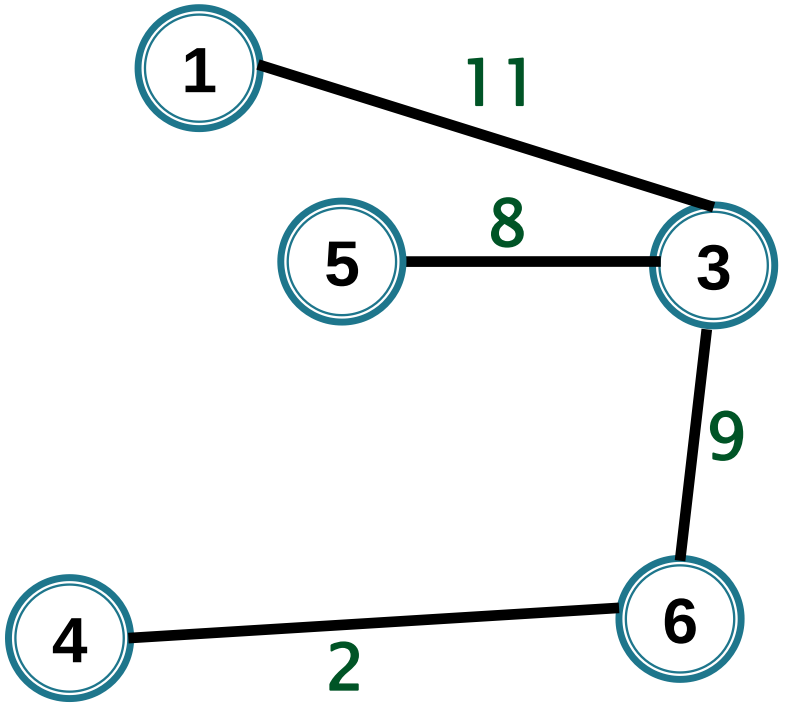
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]
Sel. 6:						



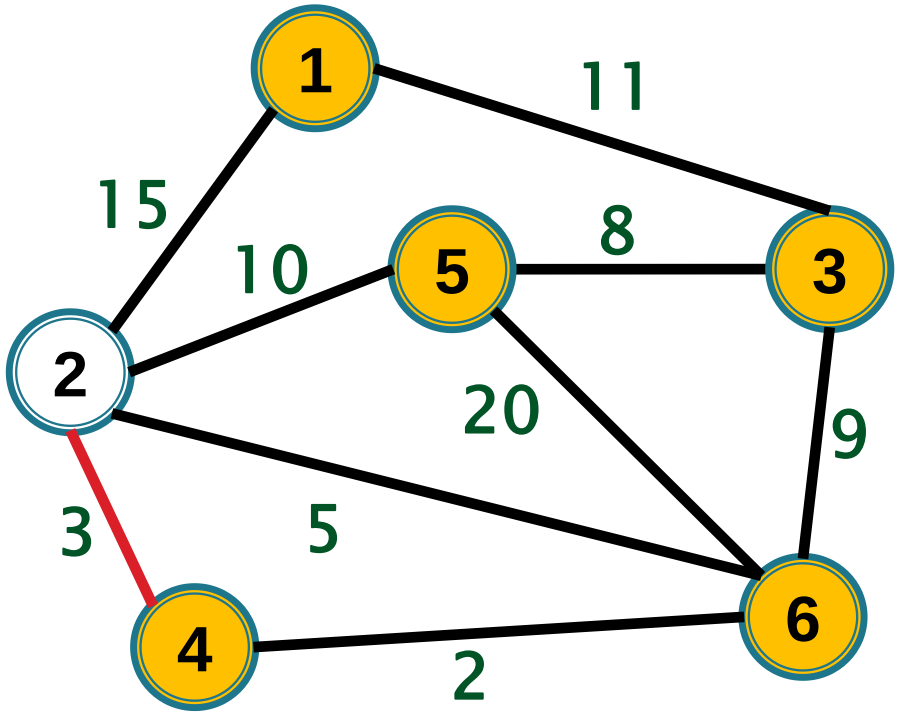


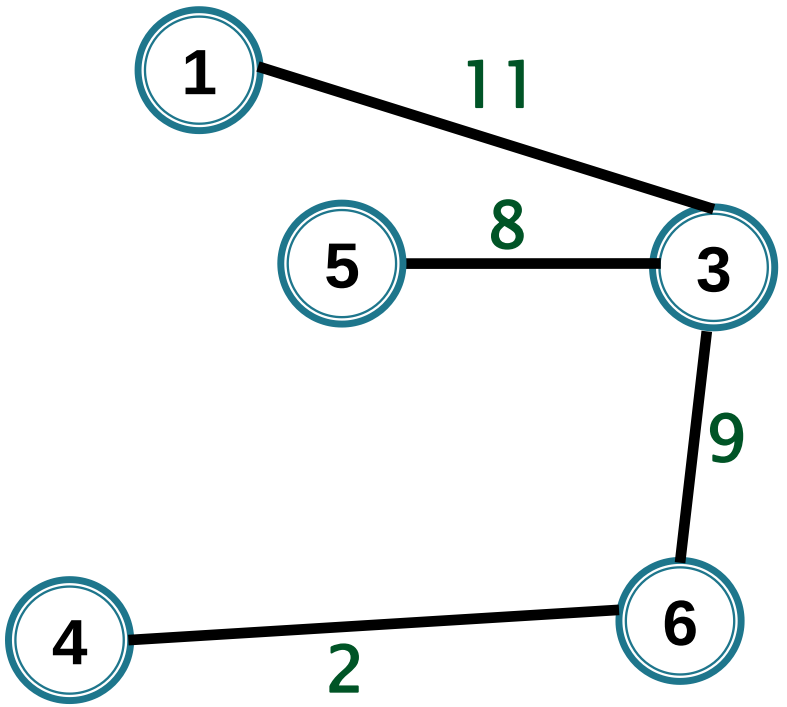
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]
Sel. 6:	[– ,	5/6 ,	– ,	2/6,	– ,	–]



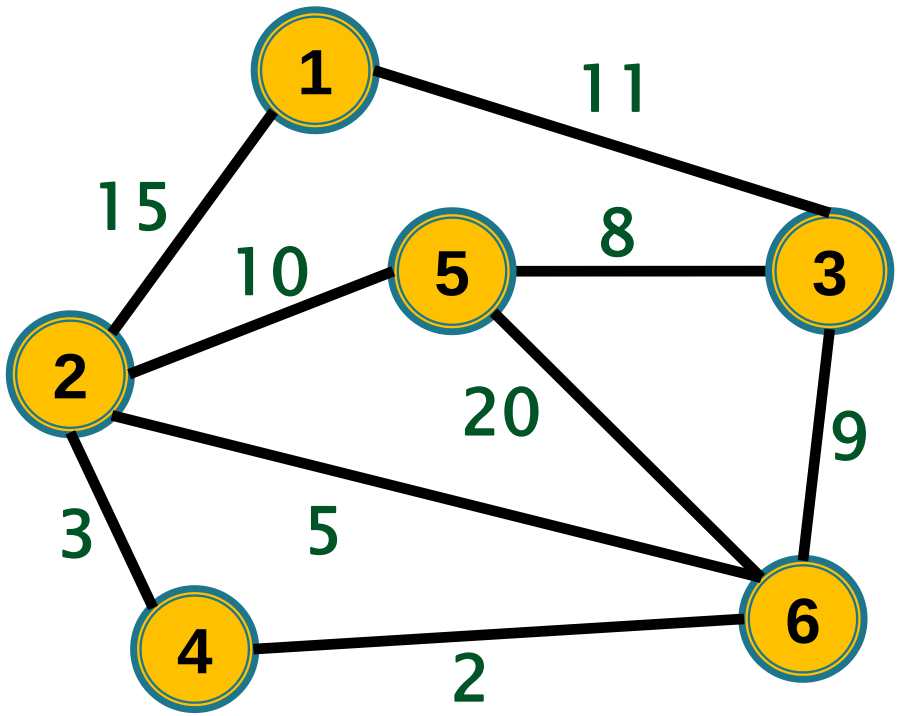


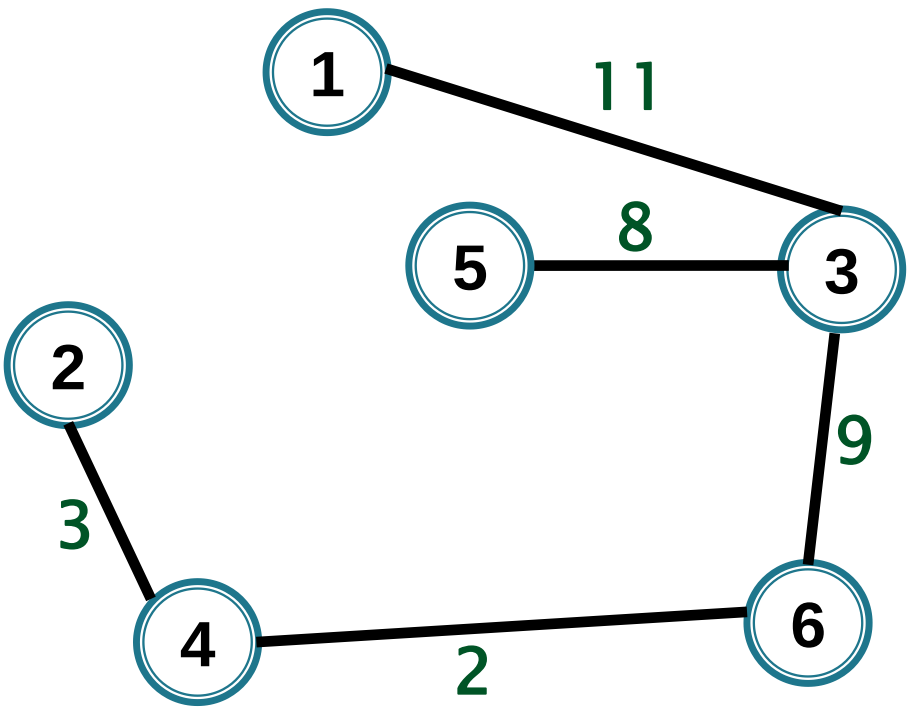
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]
Sel. 6:	[– ,	5/6 ,	– ,	2/6,	– ,	–]
Sel. 4:						



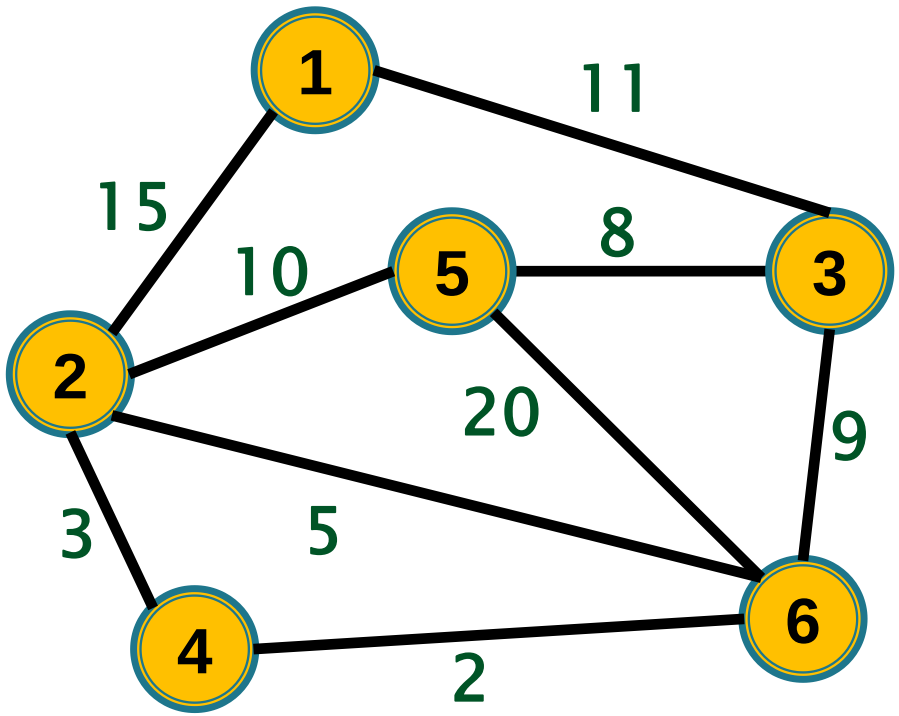


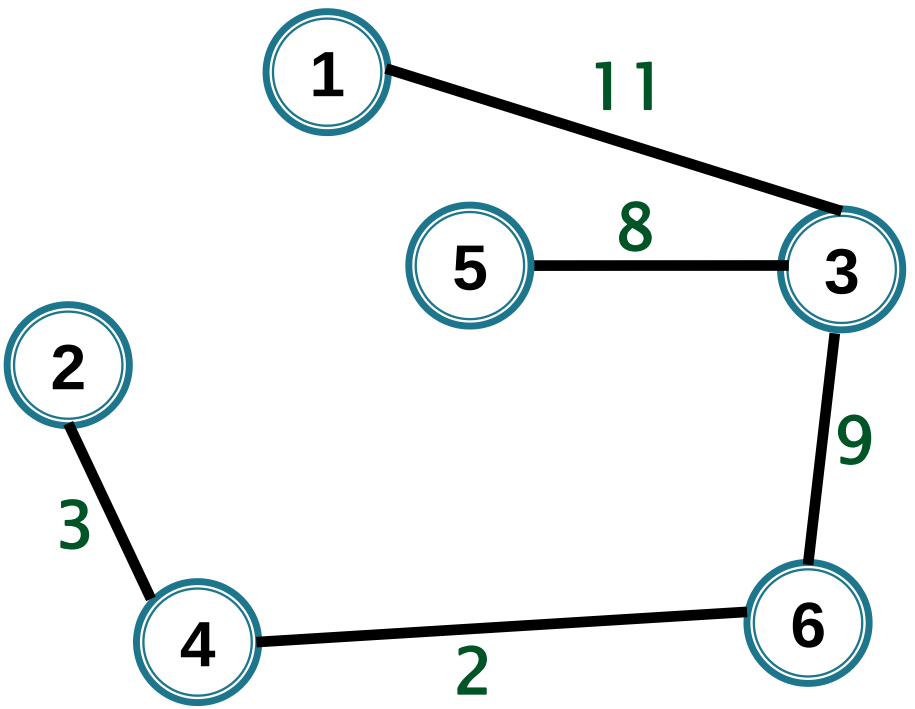
	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	8/3,	9/3]
Sel. 5:	[– ,	10/5 ,	– ,	∞/0,	– ,	9/3]
Sel. 6:	[– ,	5/6 ,	– ,	2/6,	– ,	–]
Sel. 4:	[– ,	3/4 ,	– ,	– ,	– ,	–]





	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

Varianta 2 – memorarea vârfurilor din într-un min-heap Q (min-ansamblu)

- **Inițializare Q** ->
- **n * extragere vârf minim** ->
- **actualizare etichete vecini** ->

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