

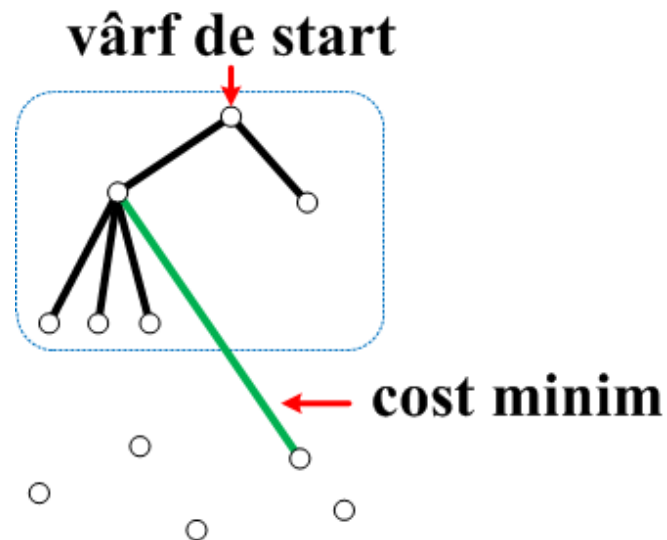
Arbori parțiali de cost minim



Algoritmul lui Prim

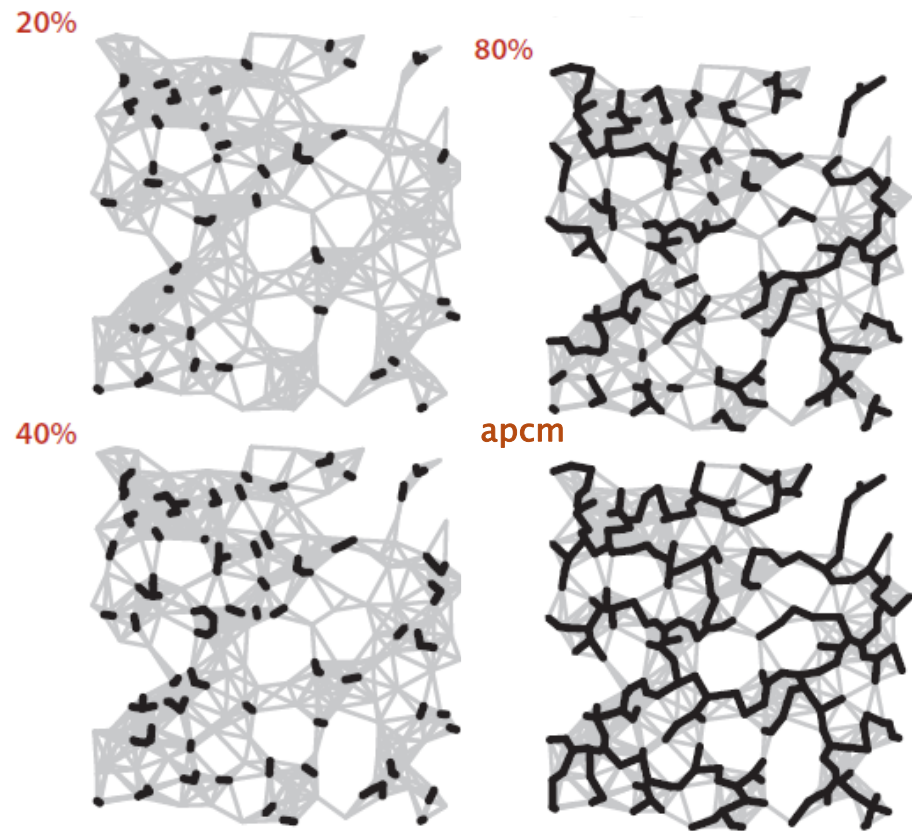
Algoritmul lui Prim

- ▶ 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 la arbore la unul neadăugat

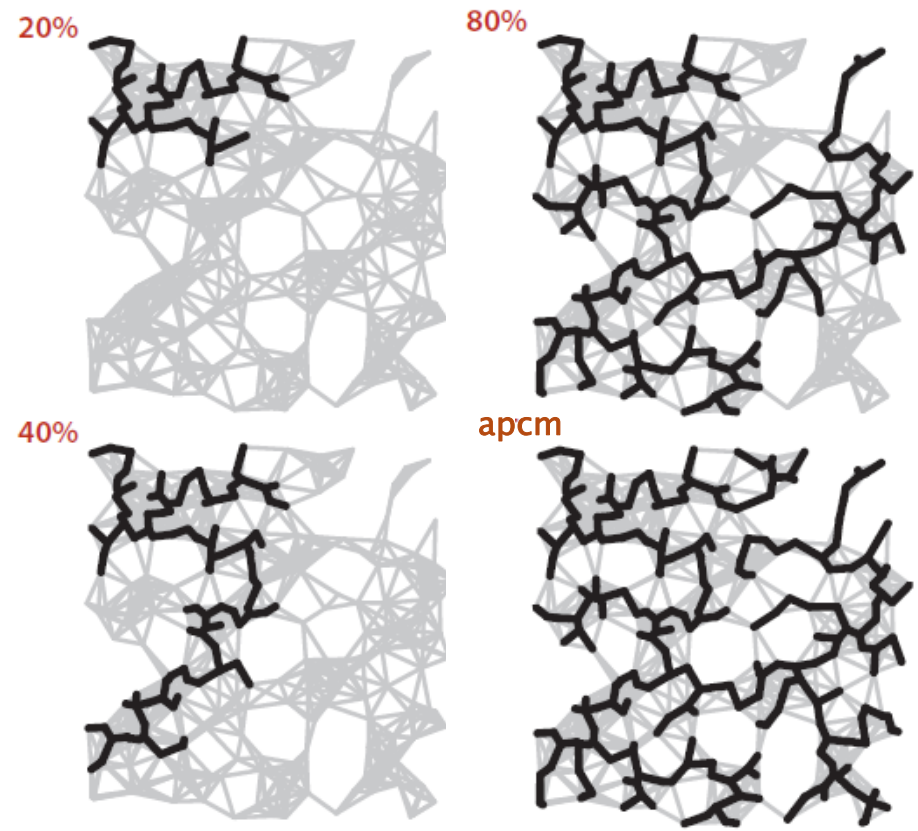


Arbori parțiali de cost minim

Kruskal



Prim



Imagine din

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

► O primă formă a algoritmului

Kruskal

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

Prim

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

► O primă formă a algoritmului

Kruskal

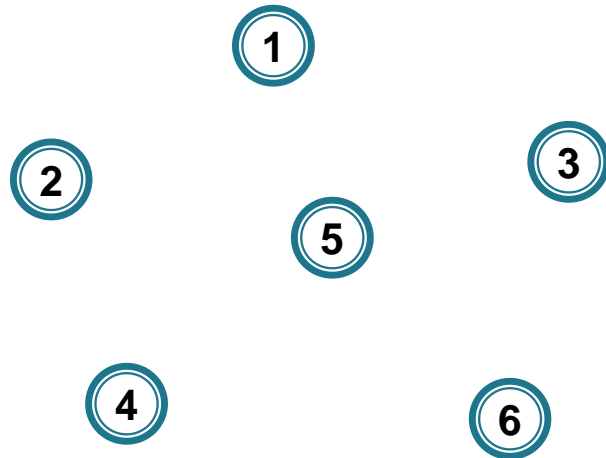
- Inițial $T = (V; \emptyset)$
- pentru $i = 1, n-1$
 - alege o muchie uv cu **cost minim** a.î. u, v sunt în **componente conexe diferite** ($T+uv$ aciclic)
 - $E(T) = E(T) \cup uv$

Prim

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

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

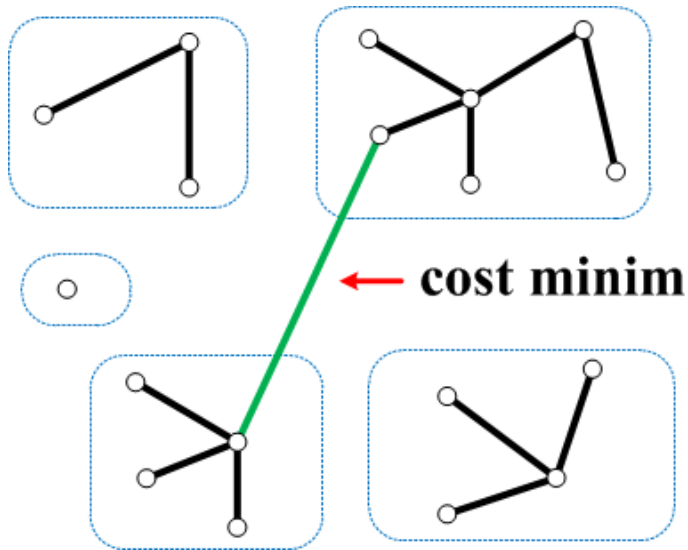


- Se adăugă pe rând câte un vârf la arborele deja construit, folosind muchii de cost minim

Kruskal

- La un pas:

Muchiile selectate formează o pădure

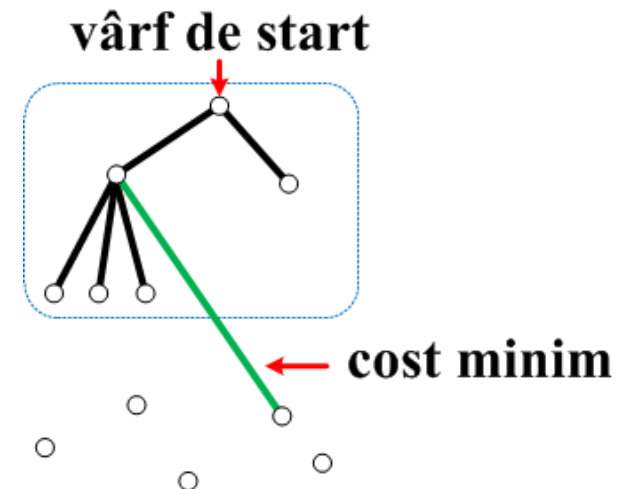


Este selectată o muchie de cost minim care unește doi arbori din pădurea curentă (două componente conexe)

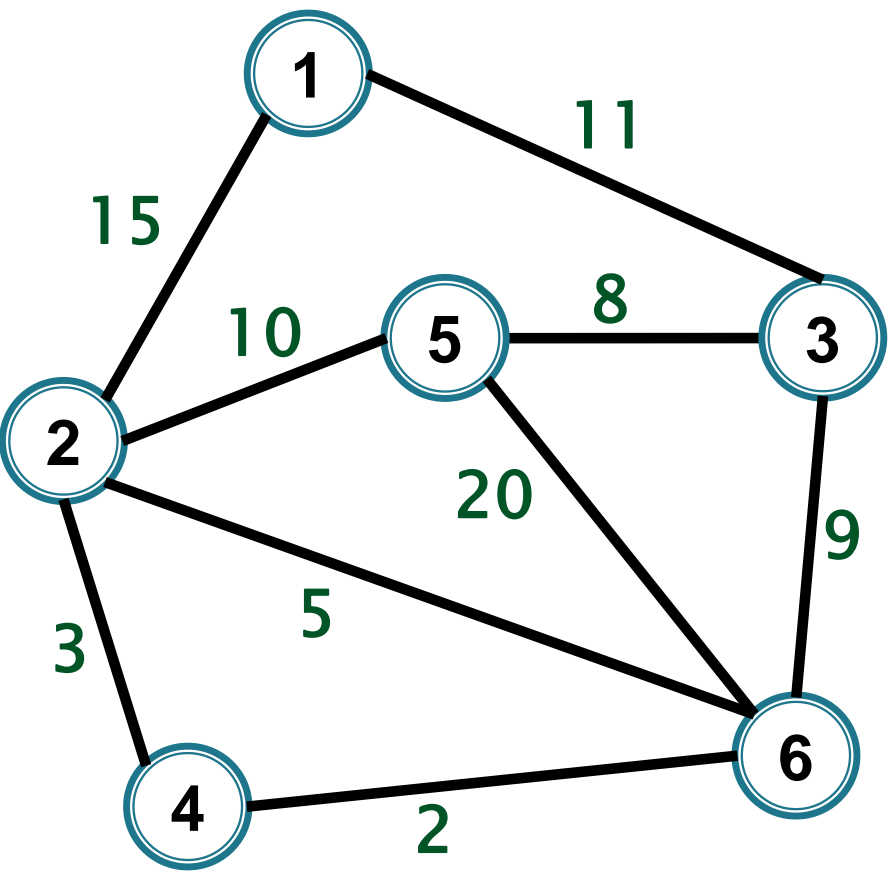
Prim

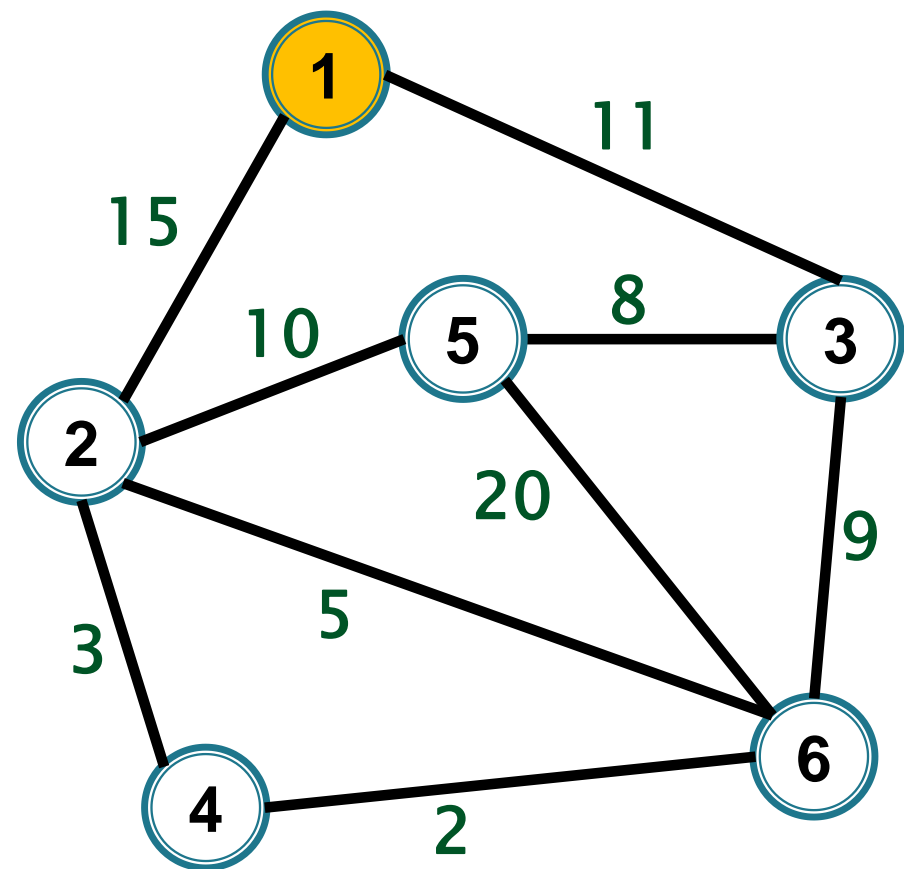
- La un pas:

Muchiile selectate formează un arbore

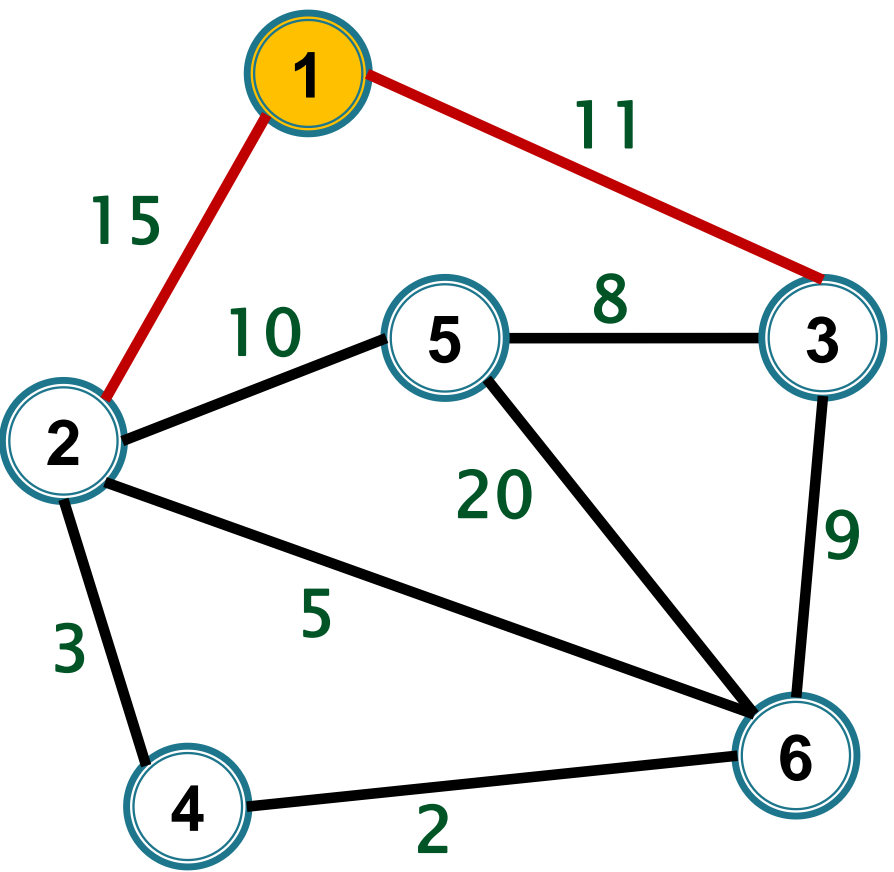


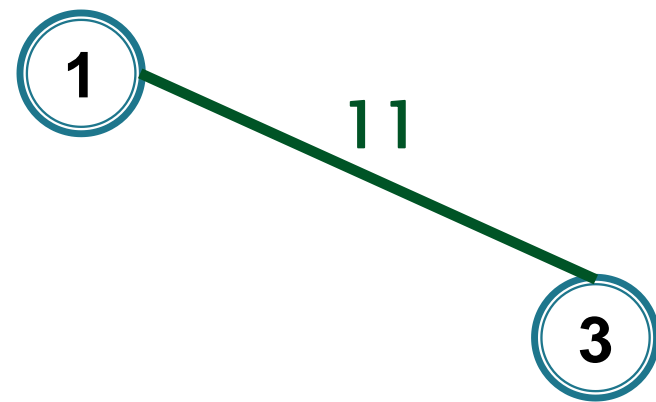
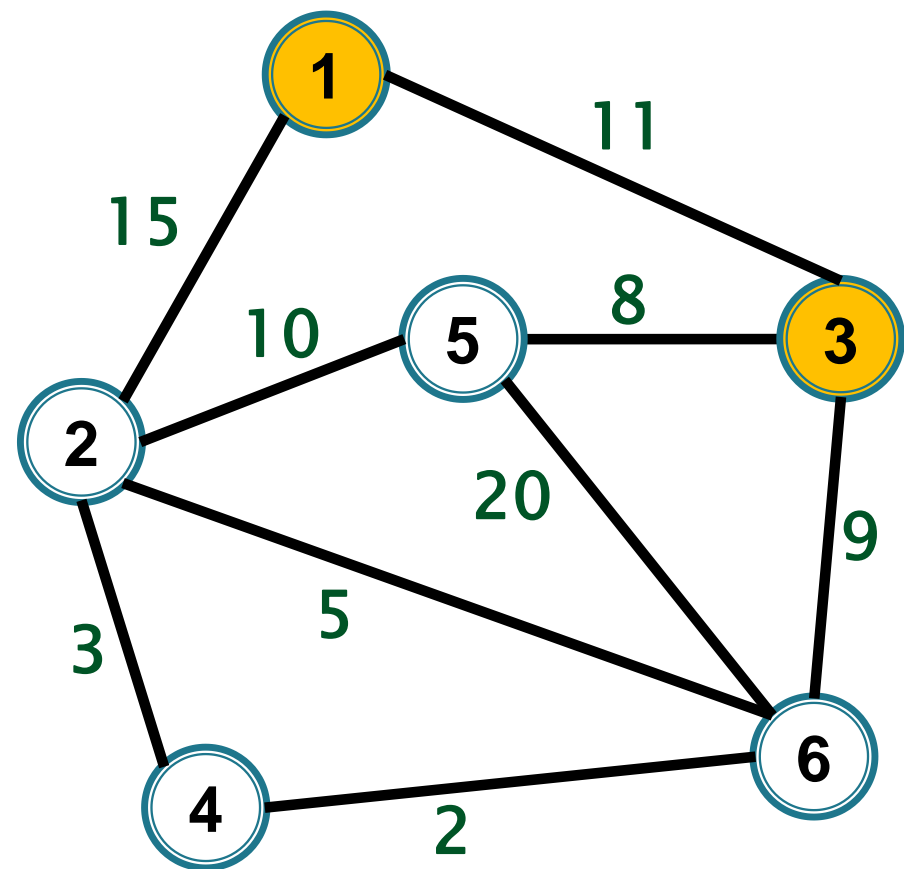
Este selectată o muchie de cost minim care unește un vârf din arbore cu unul care nu este în arbore(neselectat)

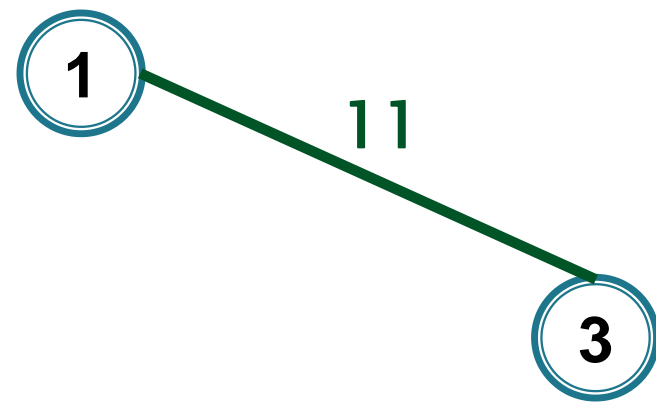
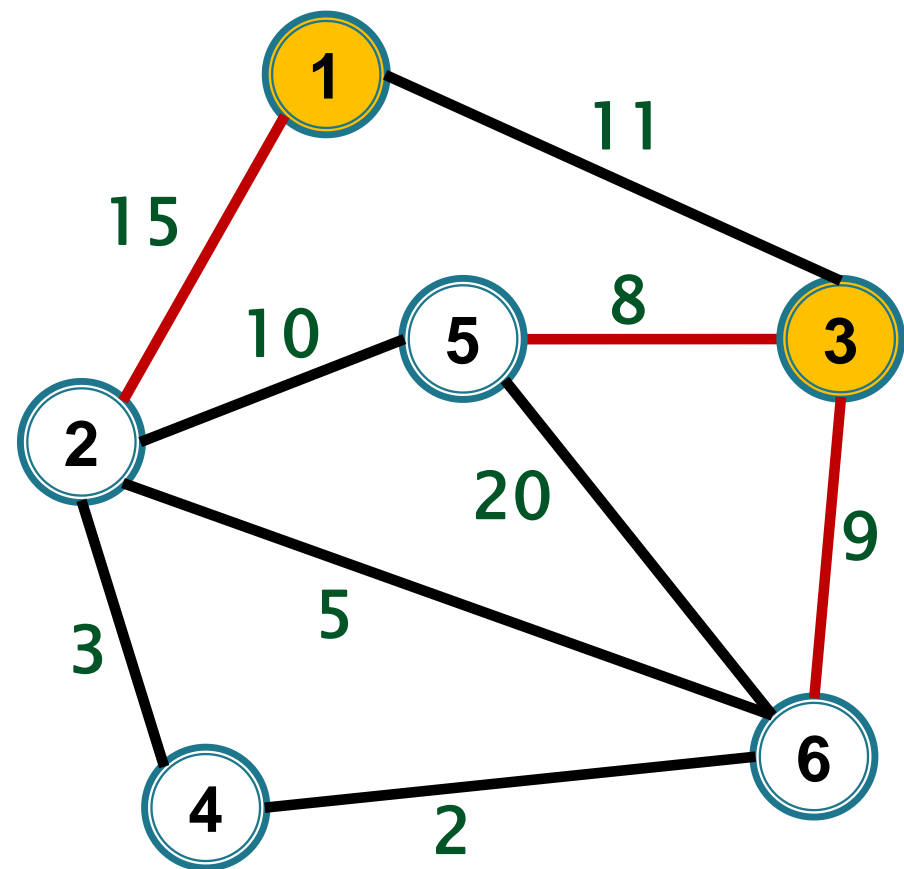


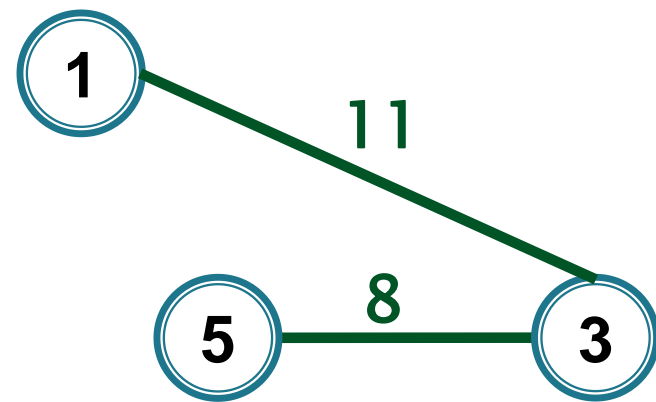
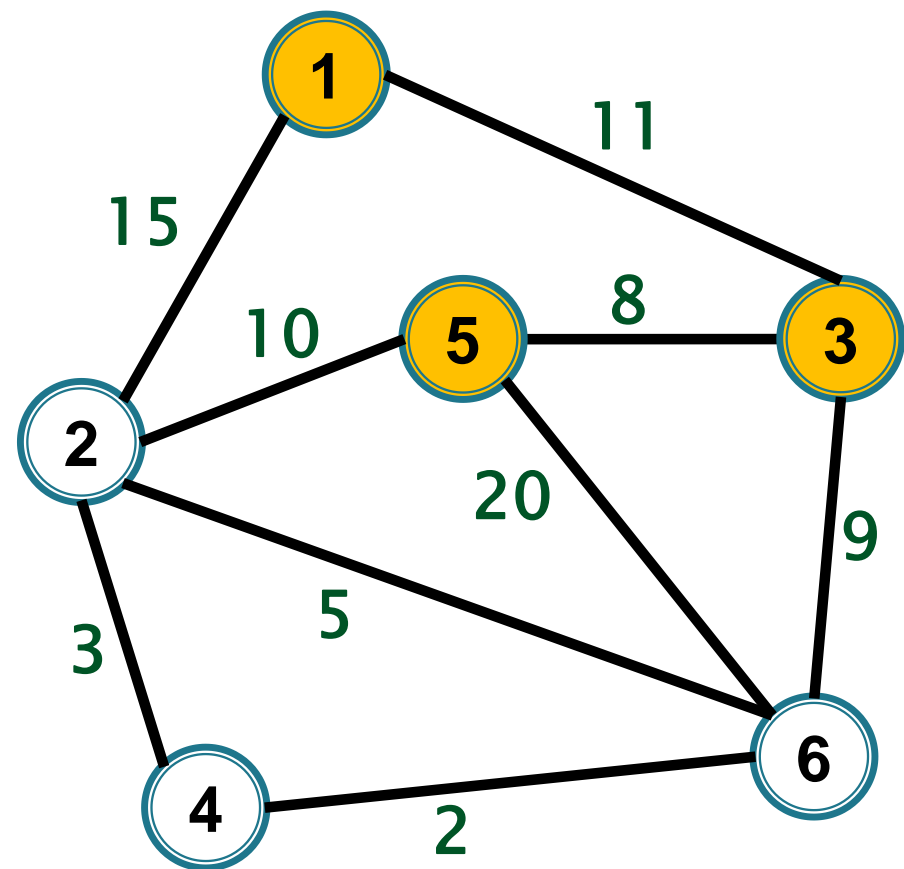


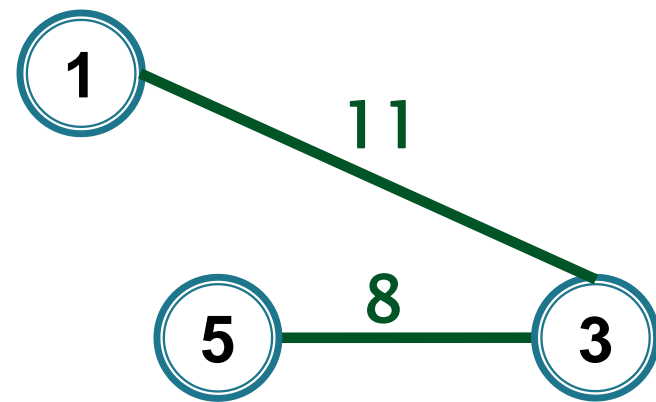
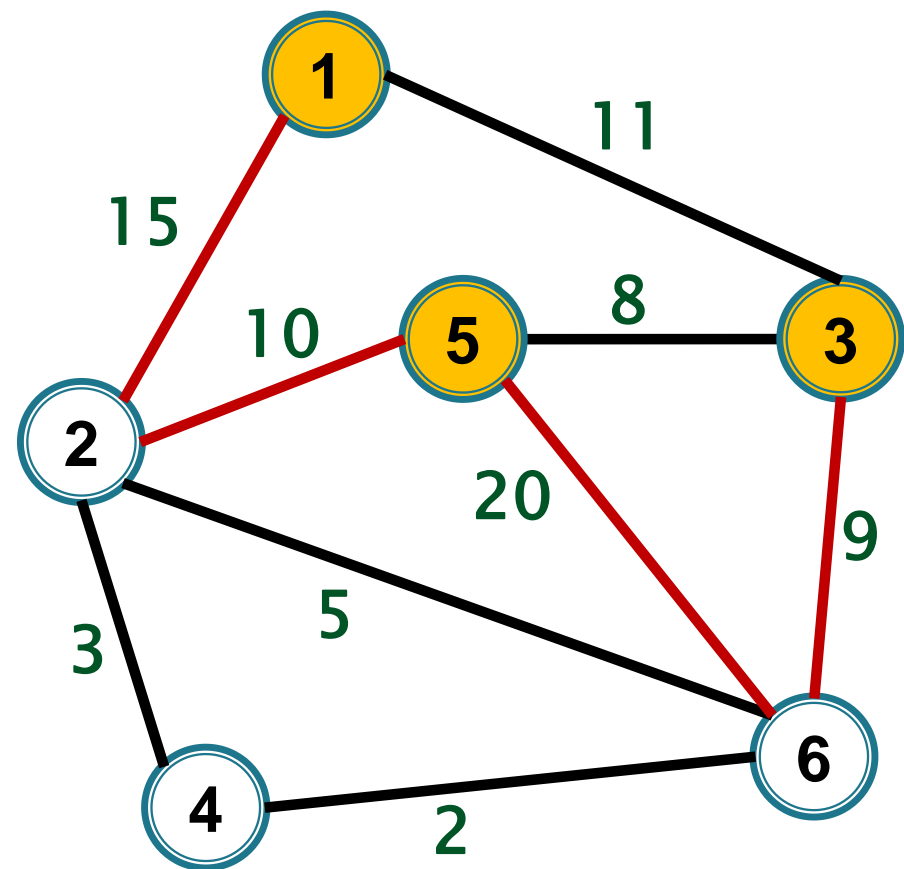
$s =$ 

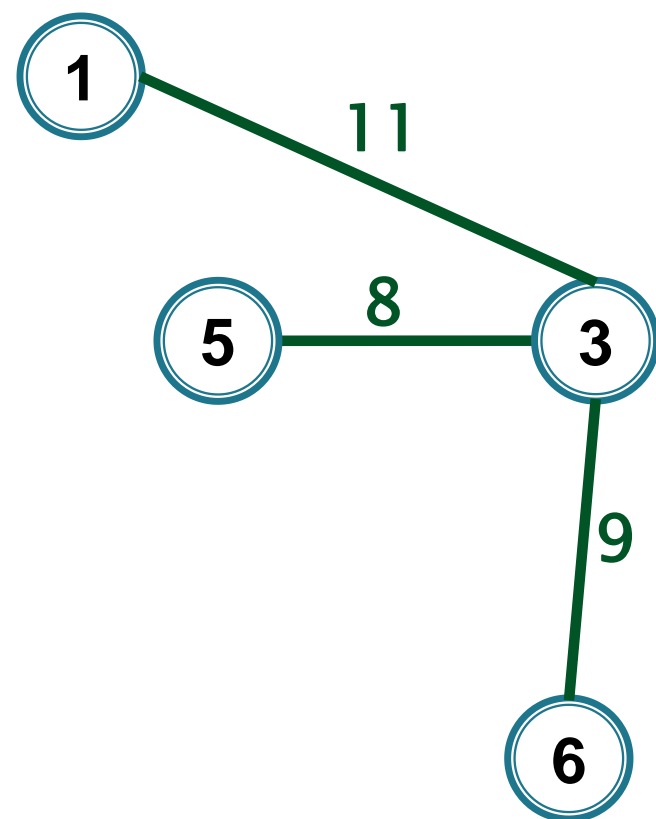
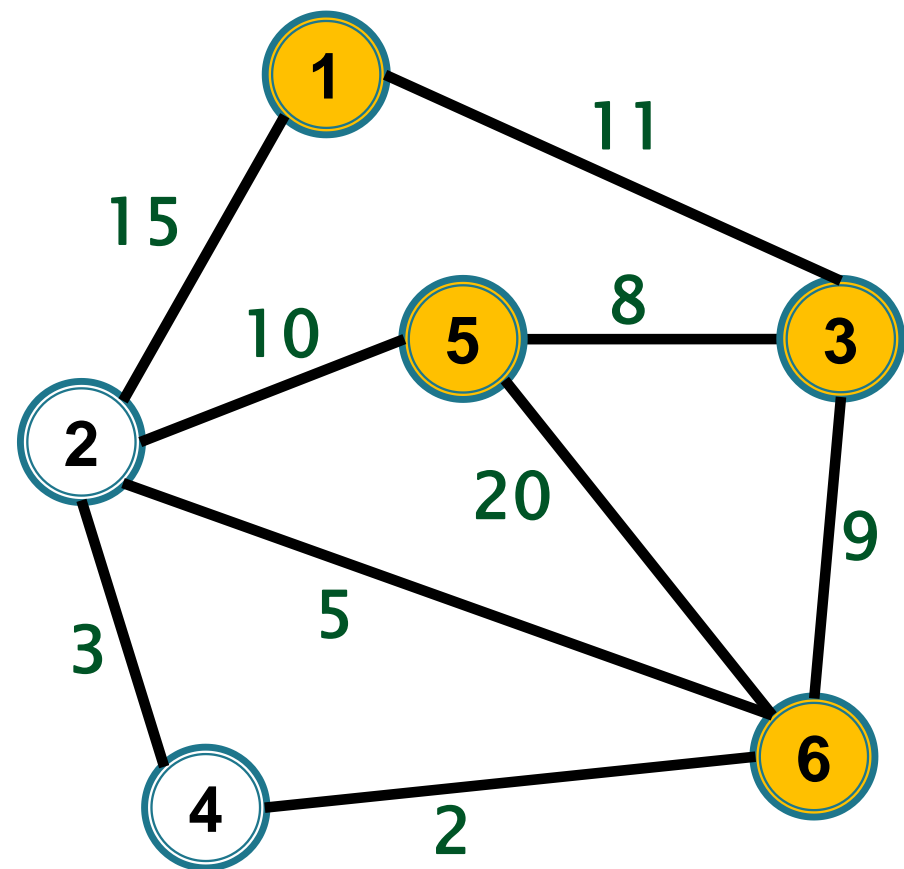


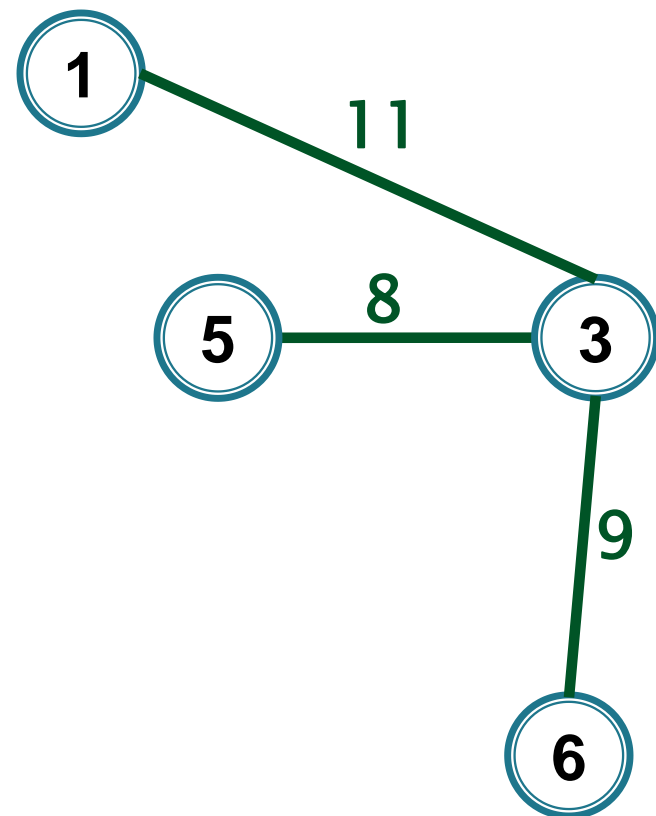
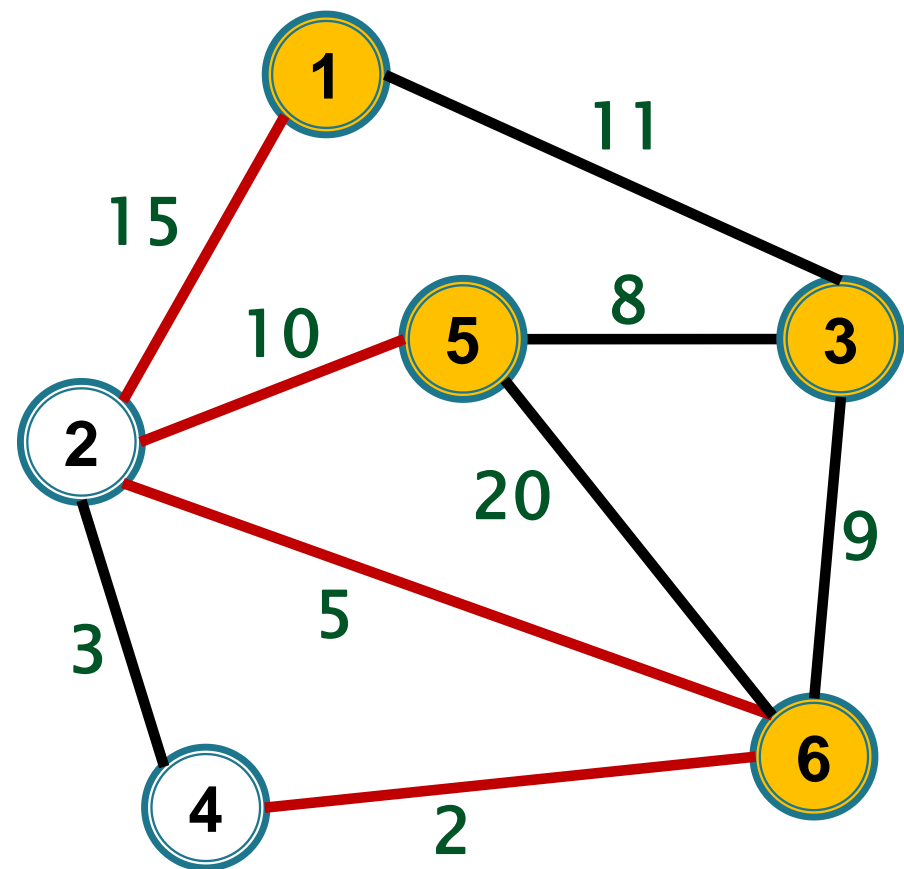


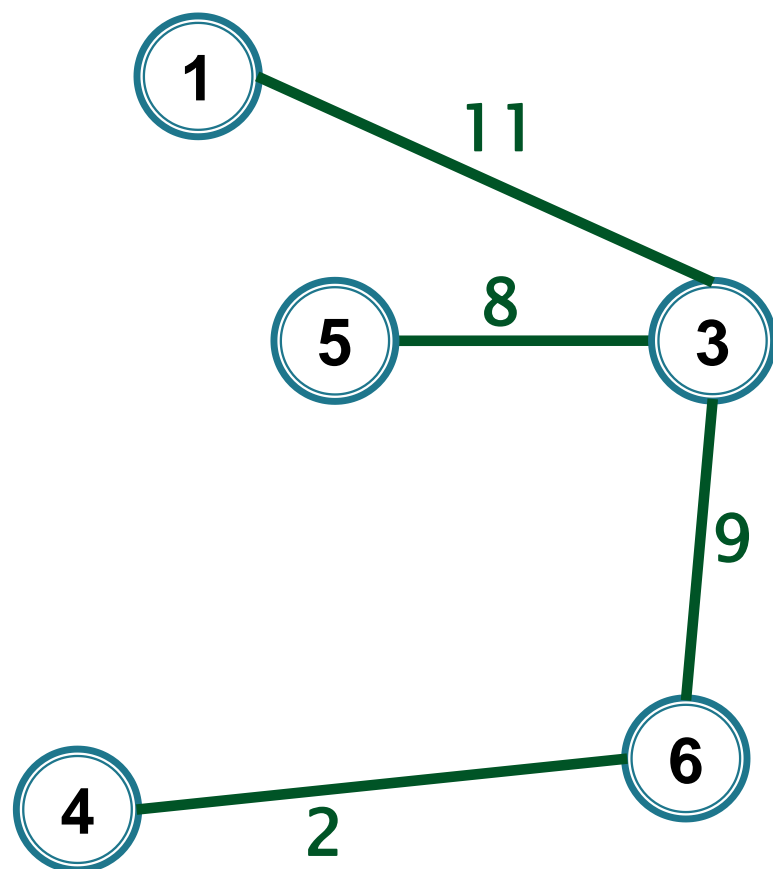
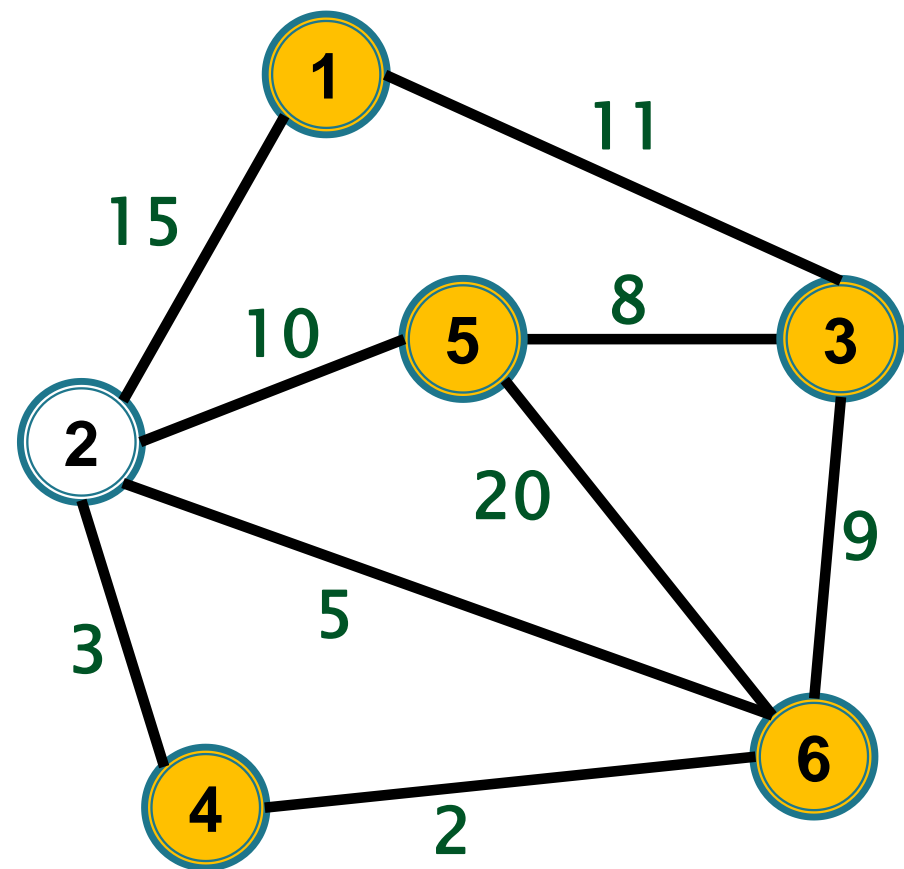


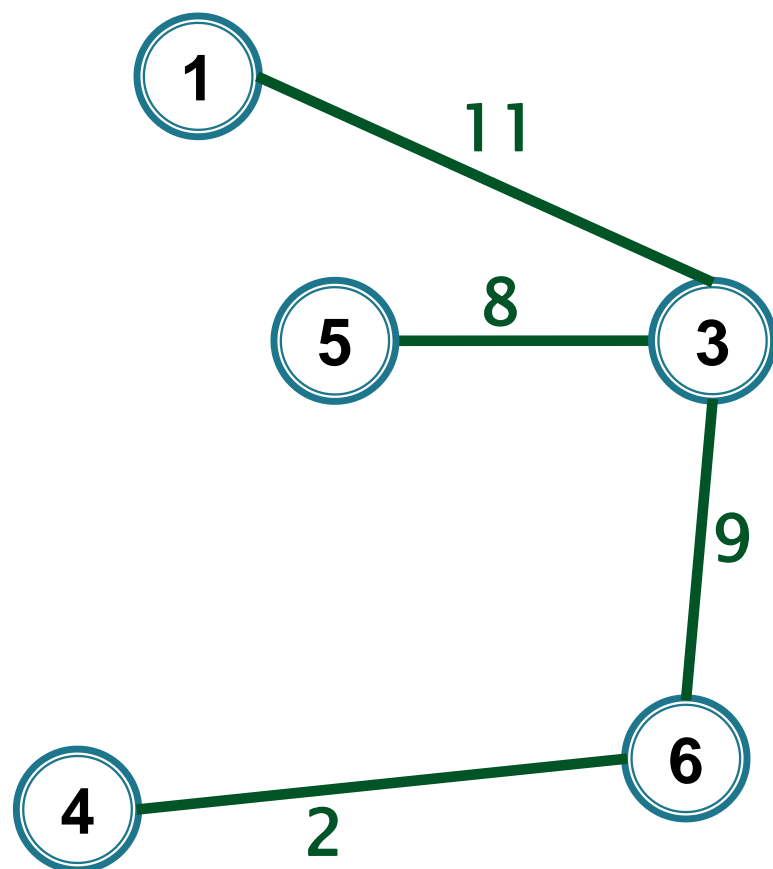
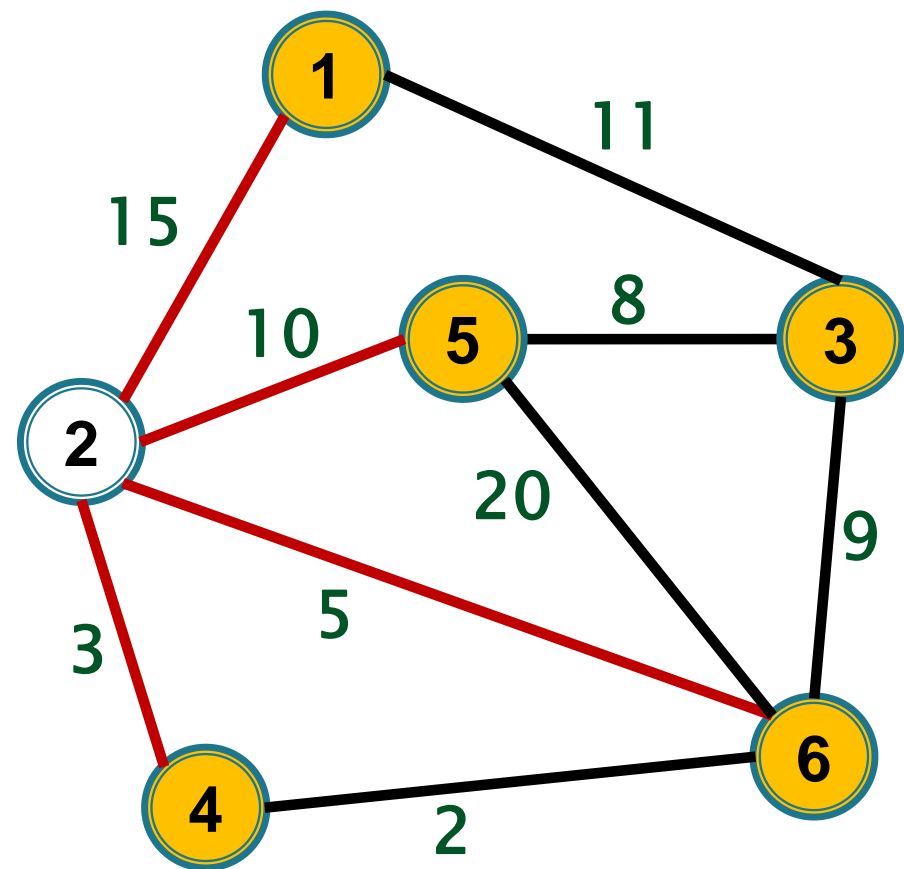


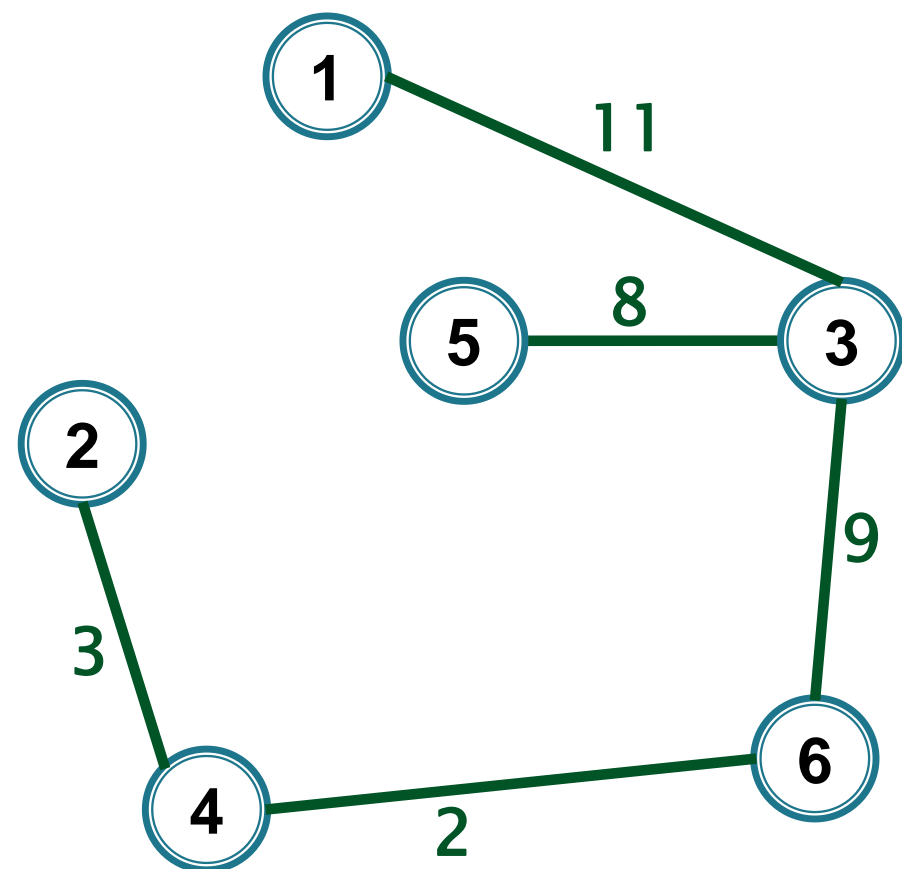
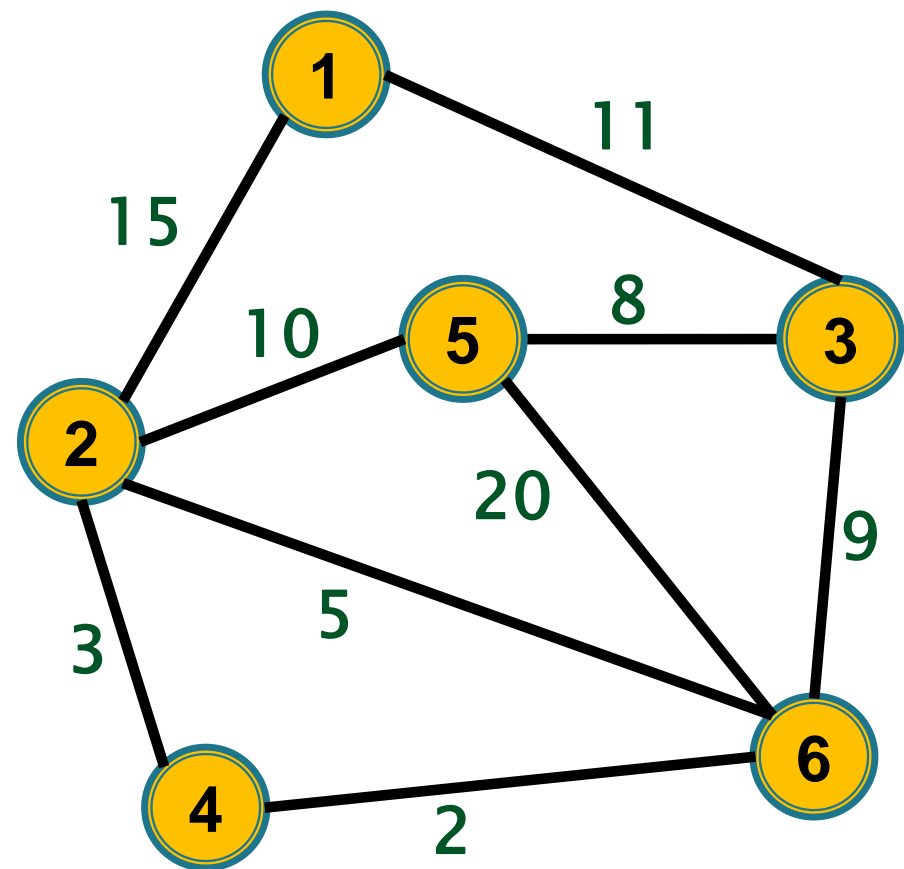


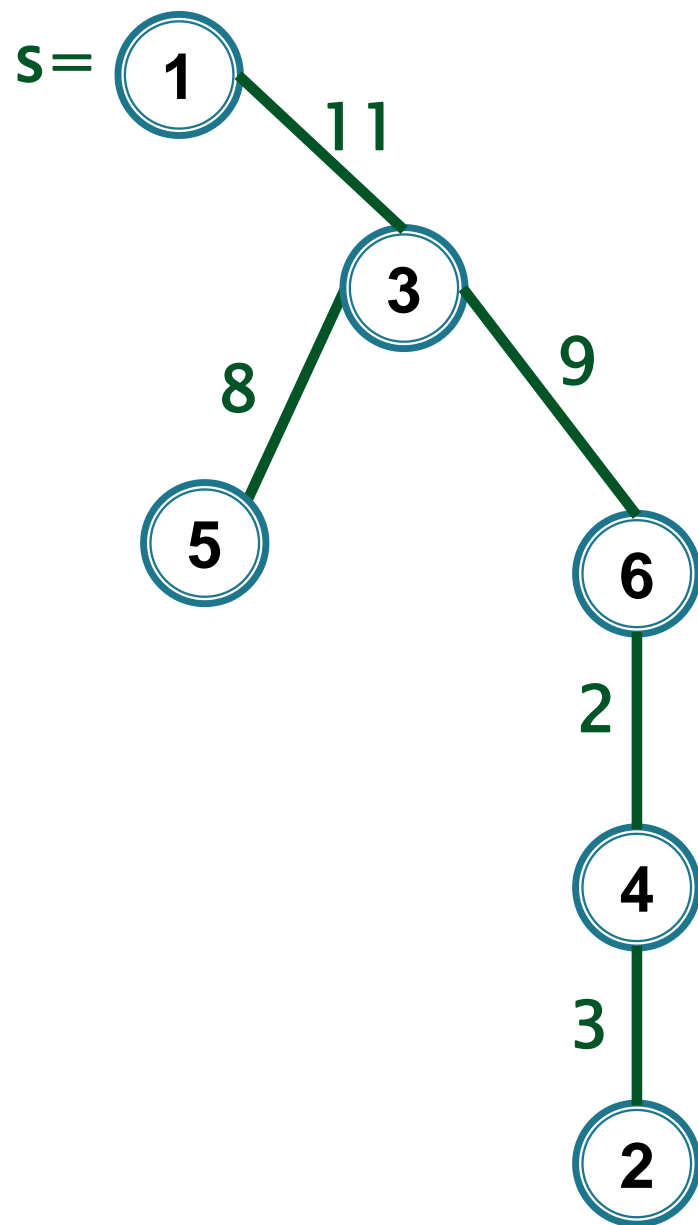
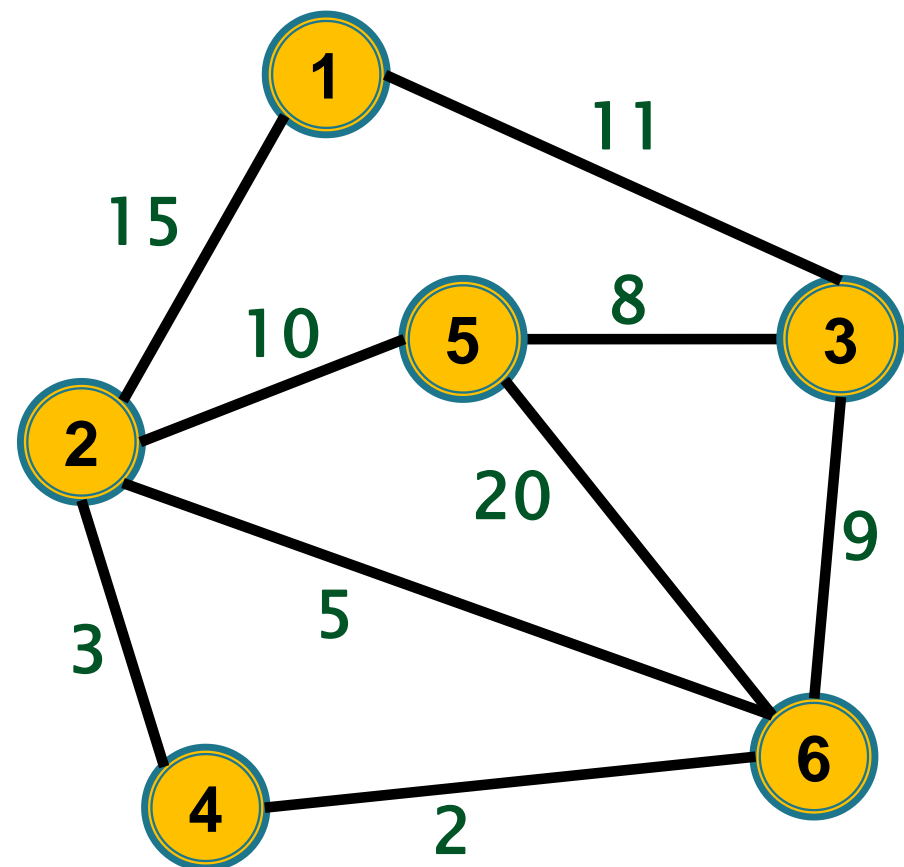












Implementare+Complexitate



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 și o alegem pe cea de cost minim cu o extremitate selectată și una neselectată

Implementare+Complexitate



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

$O(nm)$



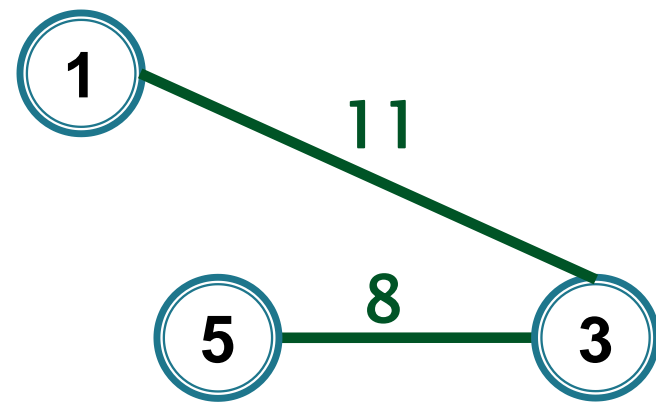
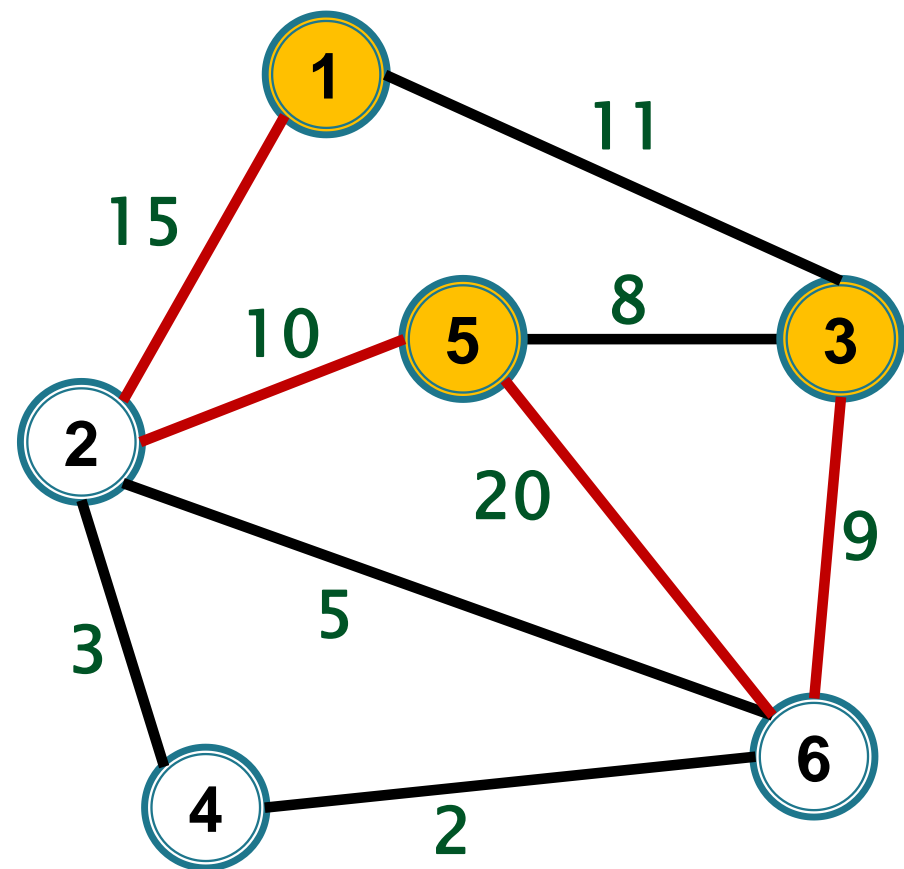
Implementare Prim



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



Implementare Prim

Cum evităm să comparăm 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 unește de un vârf care este deja în arbore

sau

- heap de muchii

(v. laborator+seminar)

Detalii implementare

Algoritmul lui Prim

Implementare Prim

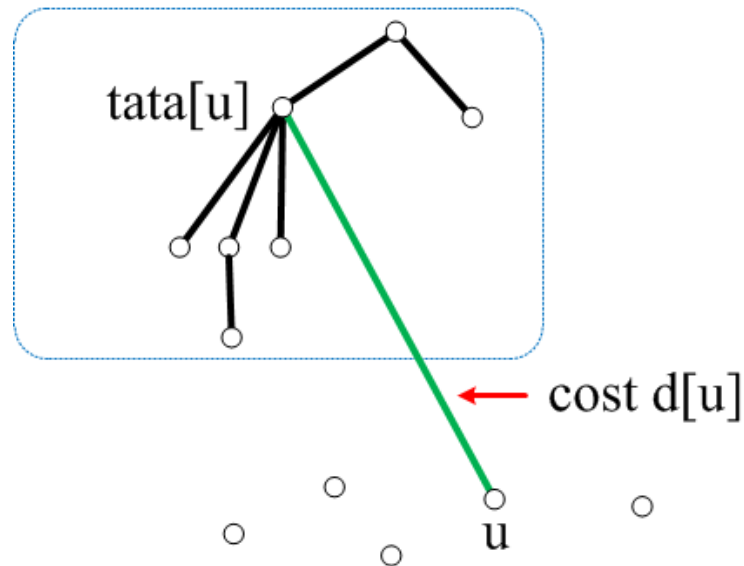
Asociem fiecărui vârf u următoarele informații (etichete) – pentru a reține **muchia de cost minim care îl unește de un vârf selectat deja în arbore**:



Implementare Prim

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

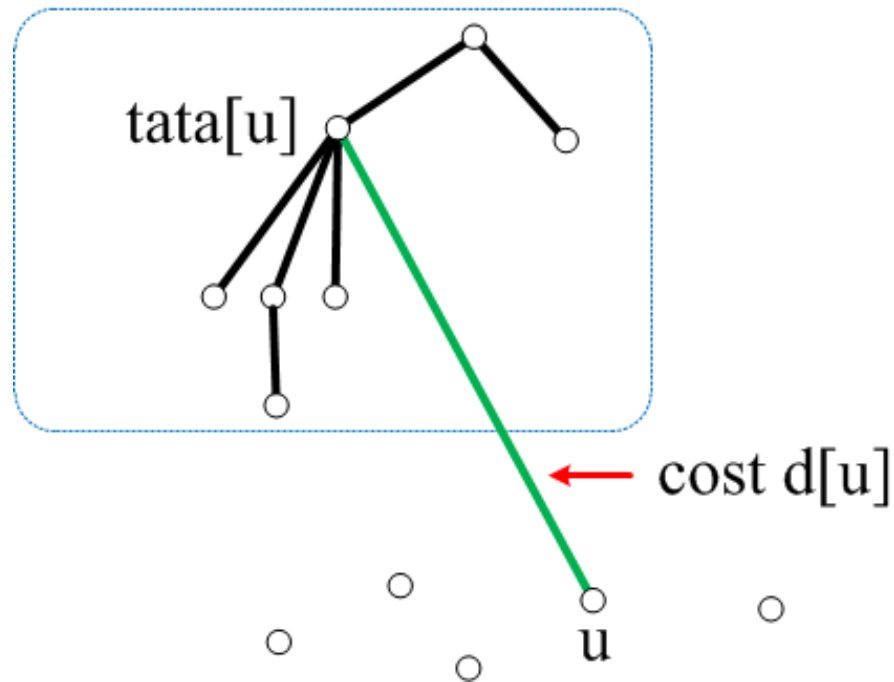
- ▶ $d[u]$ = costul minim al unei muchii de la u la un vârf selectat deja în arbore
- ▶ $tata[u]$ = acest vârf din arbore pentru care se realizează minimul



Implementare Prim

► Avem

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



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 și se adaugă la arbore muchia (**$tata[u]$, u**)

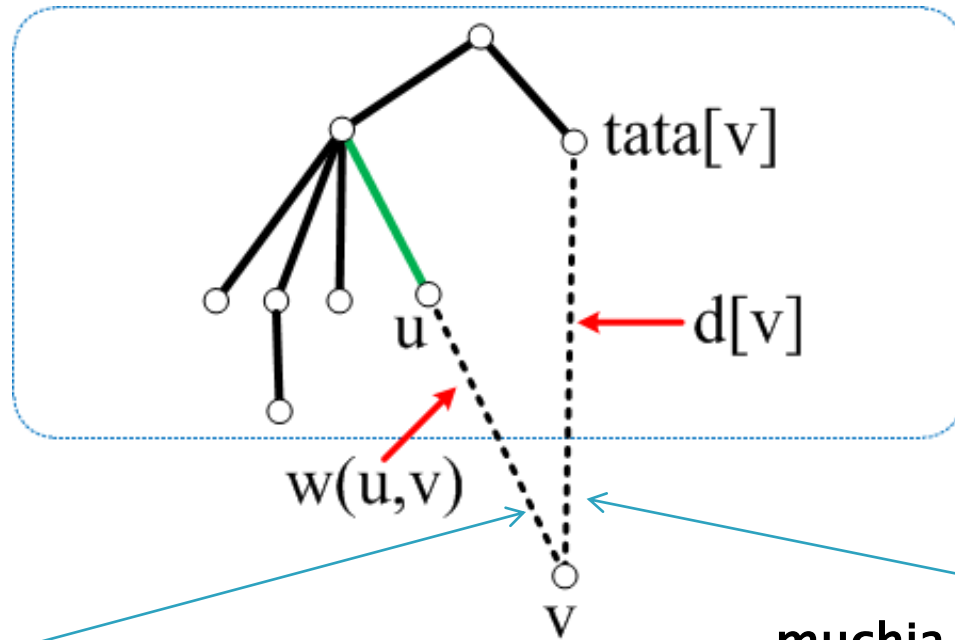
Implementare Prim

Atunci algoritmul se modifică astfel:

▶ **La un pas**

- se alege un vârf u cu eticheta d minimă care nu este încă în arbore și se adaugă la arbore muchia $(\text{tata}[u], u)$
- se actualizează etichetele vârfurilor $v \notin V(T)$ vecine cu u astfel:

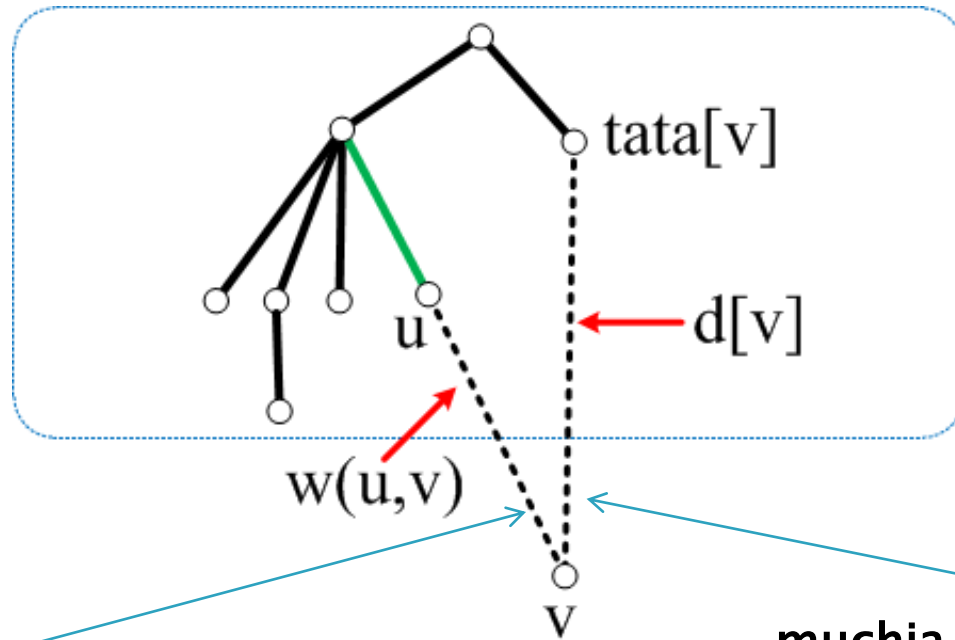
Prim



noua muchie: vu

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

Prim



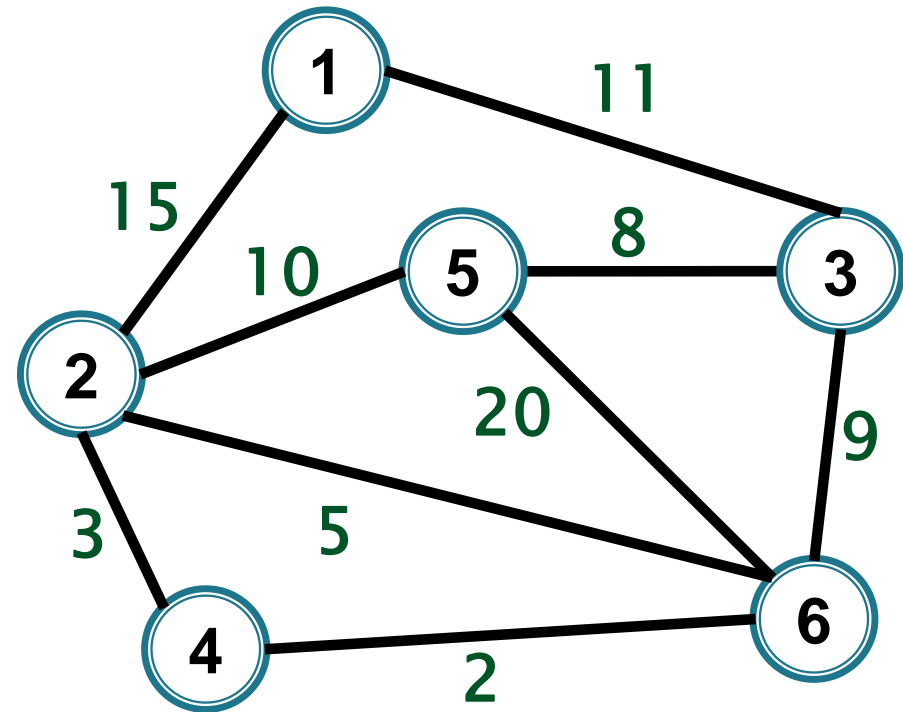
noua muchie: vu

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

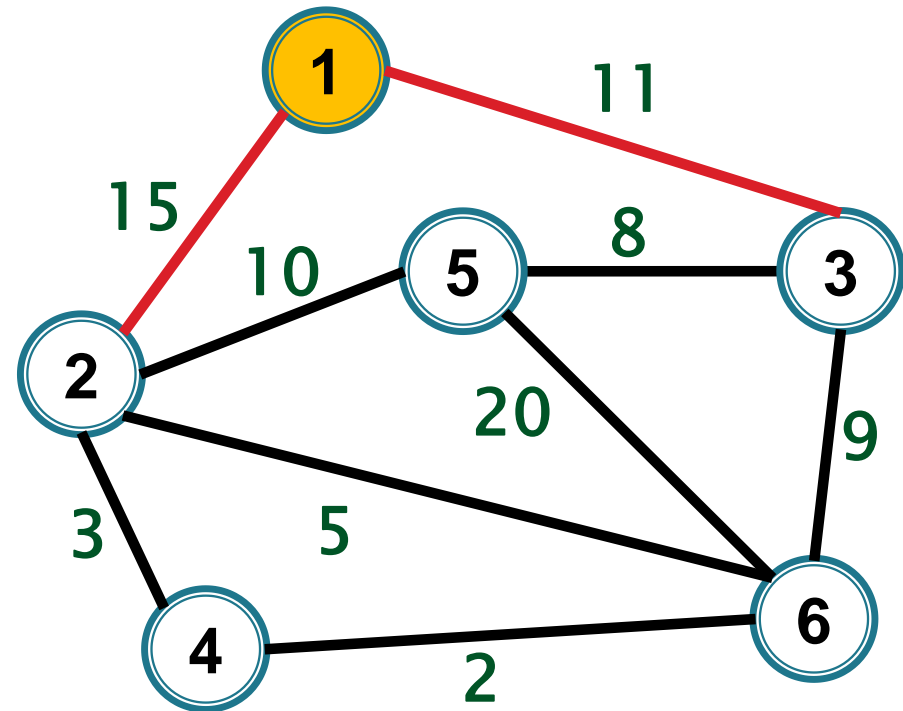
dacă $w(u, v) < d[v]$ atunci

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

$tata[v] = u$

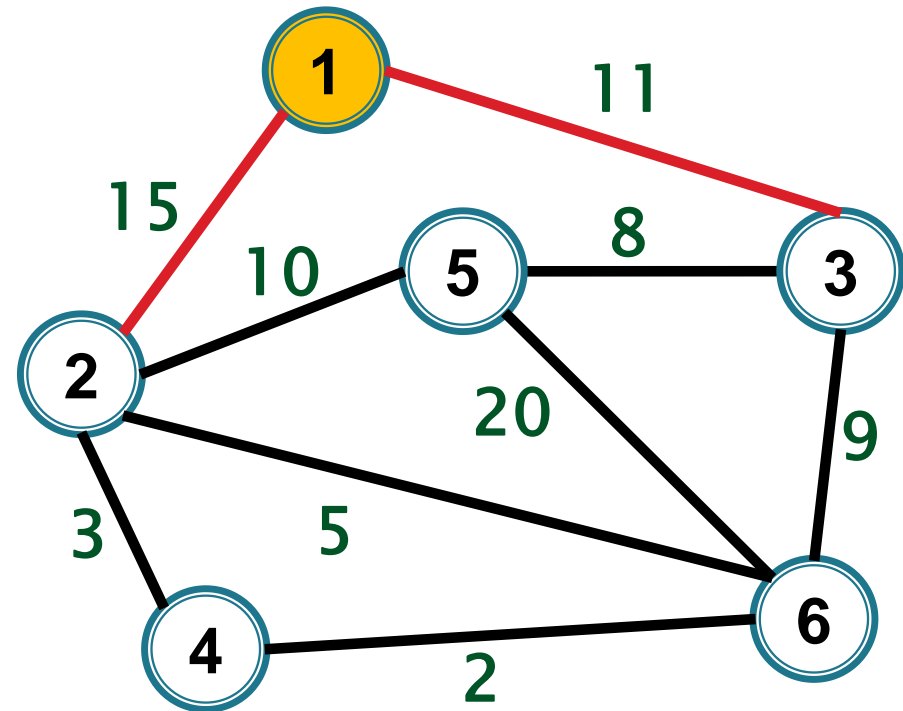


1	2	3	4	5	6
d/tata= [0/0,	∞ /0,	∞ /0,	∞ /0,	∞ /0,	∞ /0]



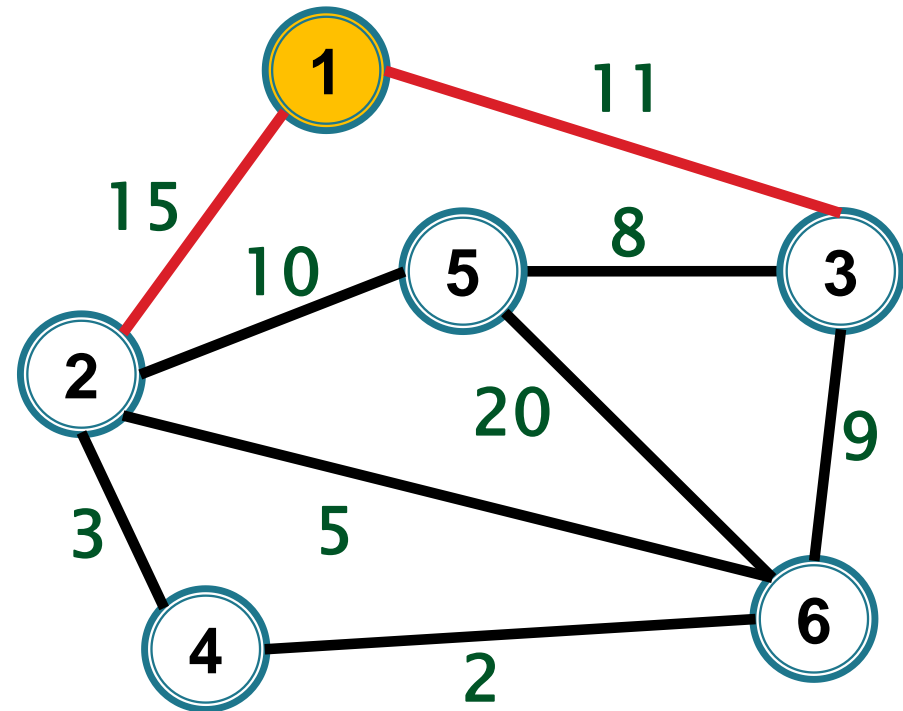
Sel. 1:

1	2	3	4	5	6
[0/0,	$\infty/0,$	$\infty/0,$	$\infty/0,$	$\infty/0,$	$\infty/0$]



1	2	3	4	5	6
[0/0,	$\infty/0,$	$\infty/0,$	$\infty/0,$	$\infty/0,$	$\infty/0$]

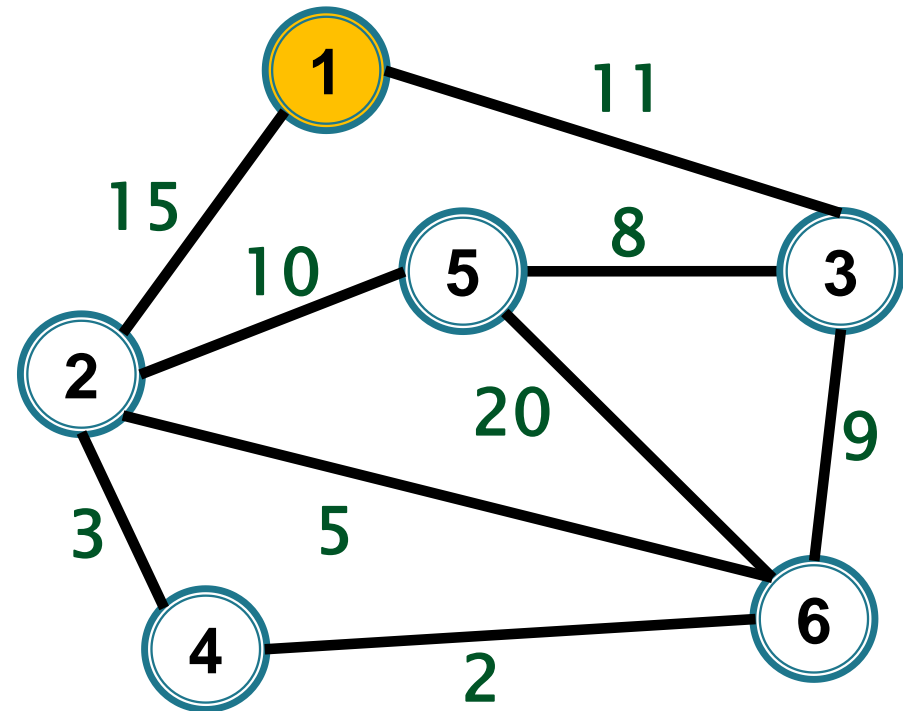
Sel. 1: viz [1] = 1 (vom reprezenta prin -)



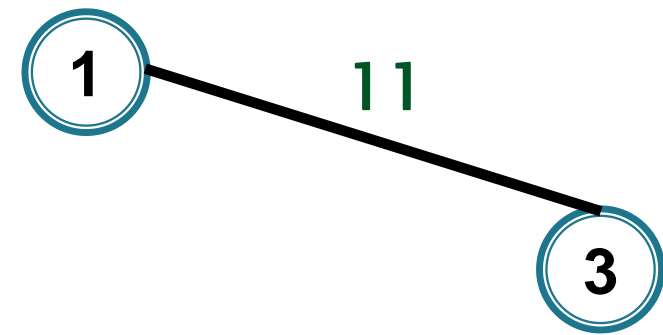
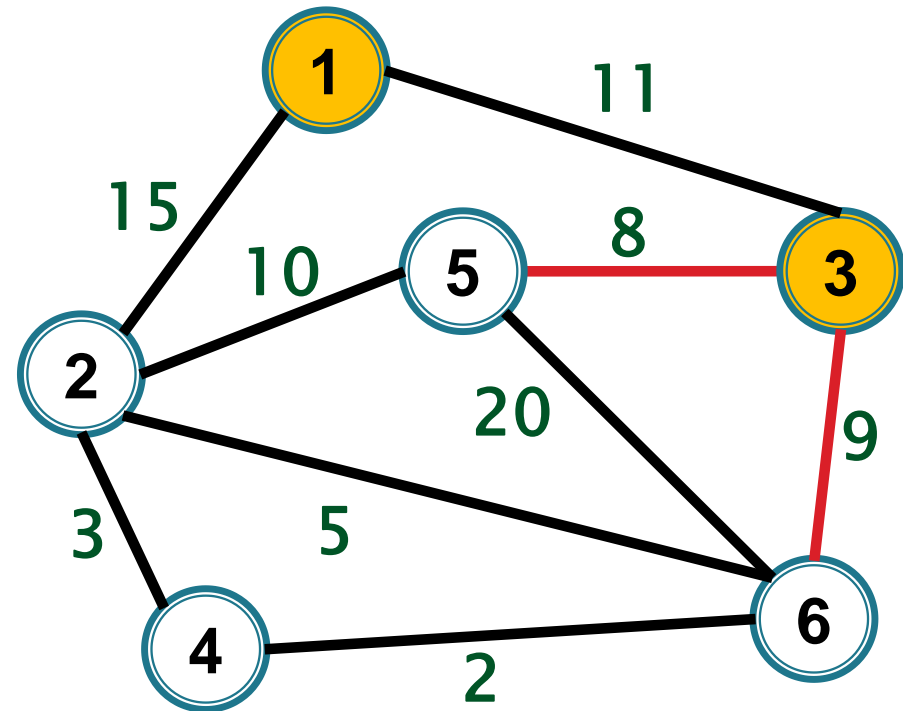
1

1	2	3	4	5	6
[0/0,	$\infty/0$,	$\infty/0$,	$\infty/0$,	$\infty/0$,	$\infty/0$]

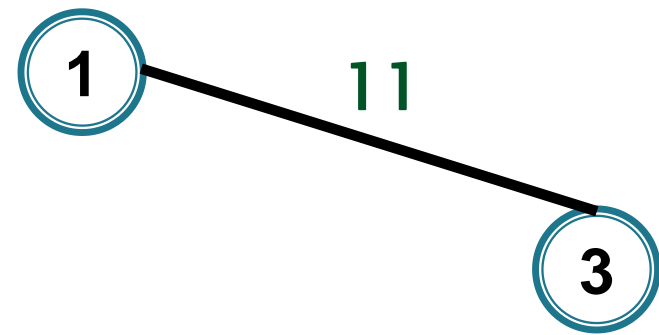
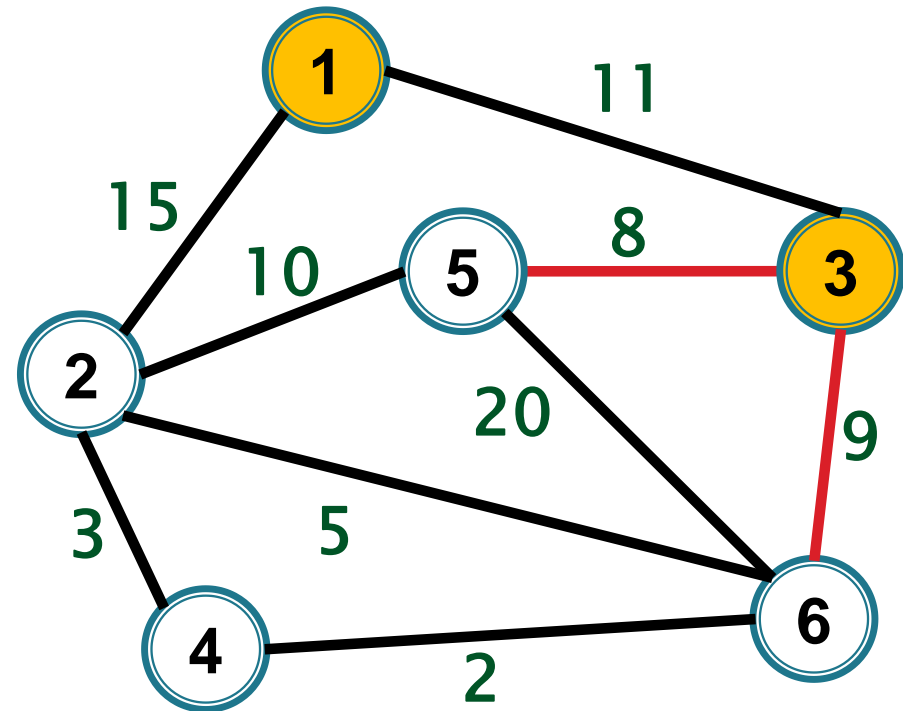
Sel. 1: actualizăm etichetele vecinilor



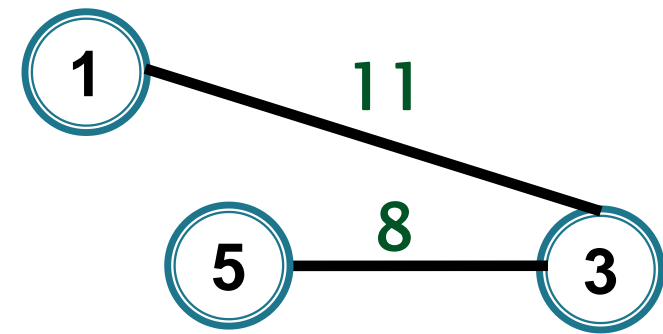
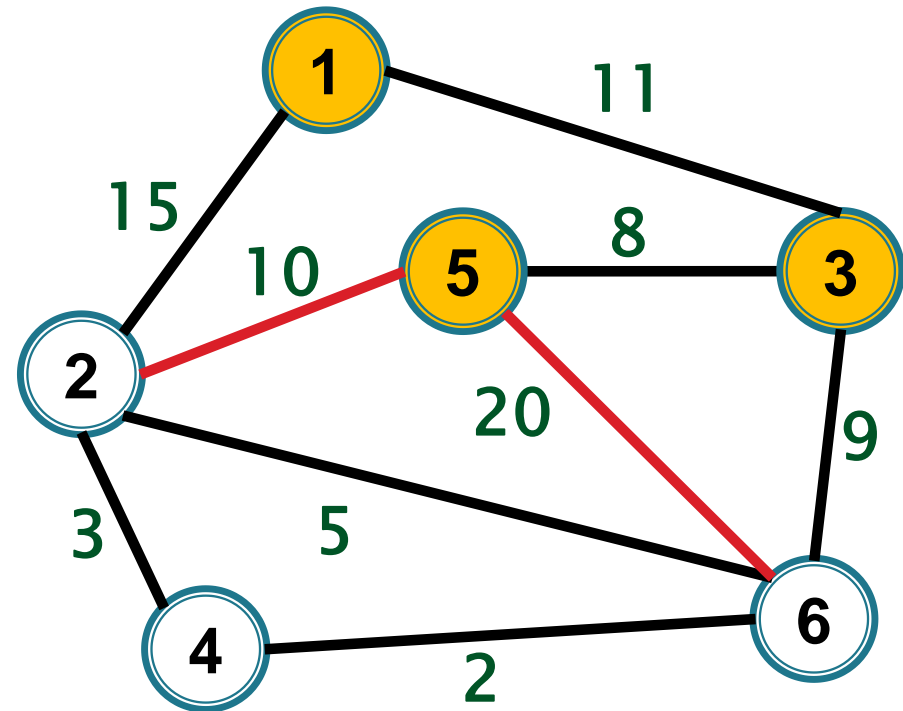
	1	2	3	4	5	6
	[0/0, ∞/0, ∞/0, ∞/0, ∞/0]					
Sel. 1:	[- , 15/1, 11/1, ∞/0, ∞/0, ∞/0]					



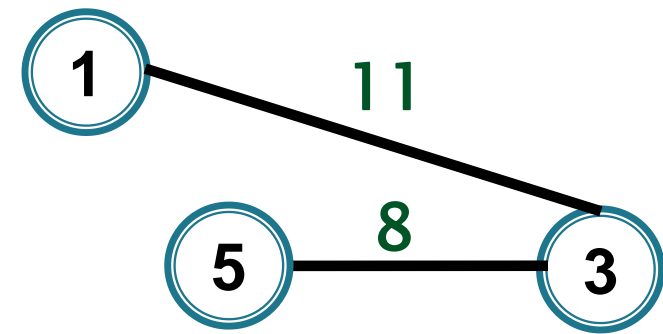
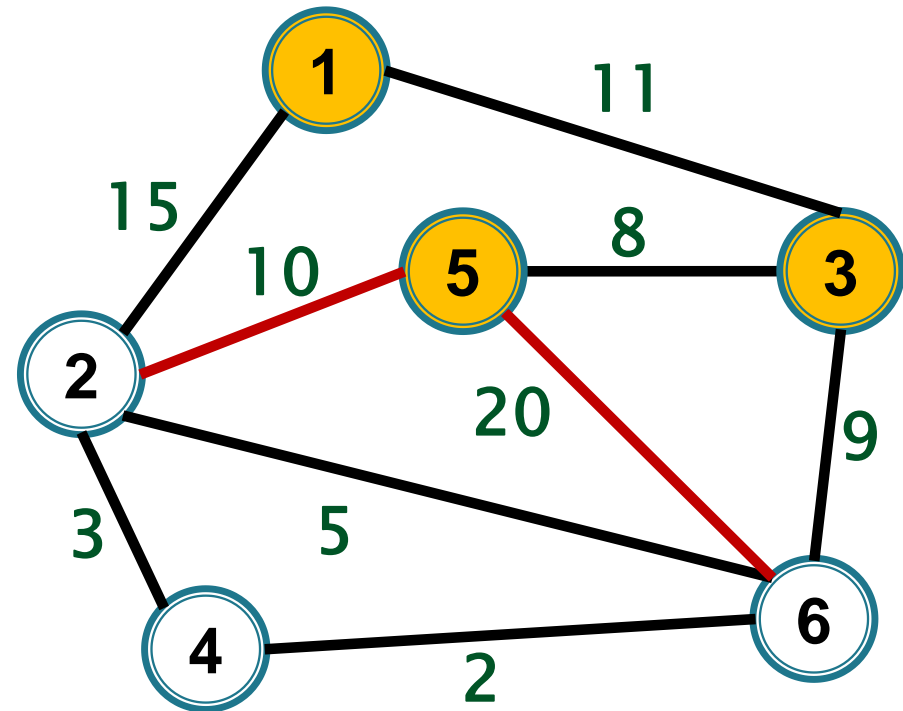
	1	2	3	4	5	6
	[0/0 , $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]	[$\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]	[$\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]	[$\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]	[$\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]	[$\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]
Sel. 1:	[- , 15/1, 11/1 , $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 3:						



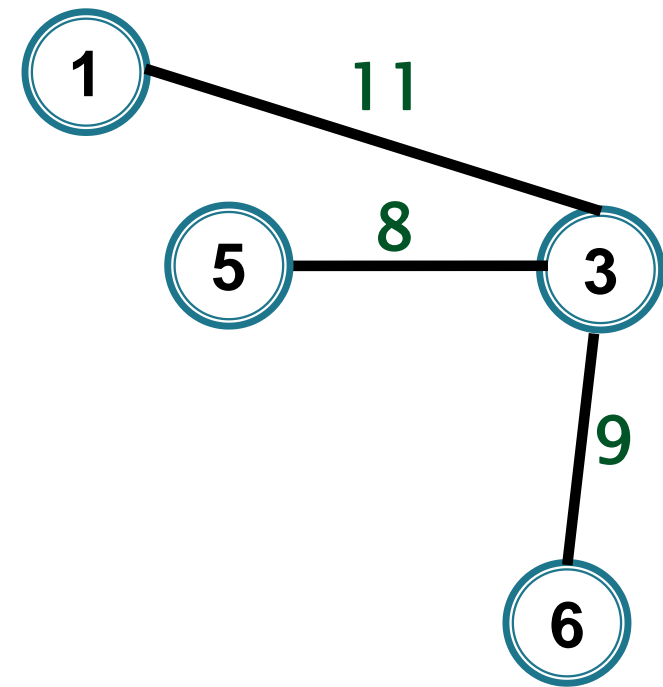
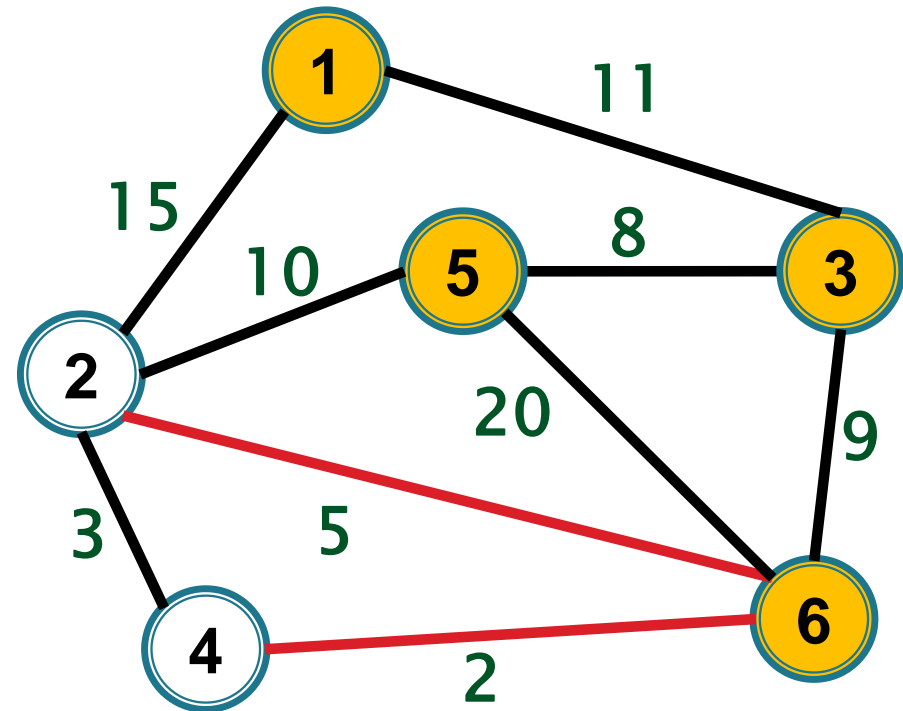
	1	2	3	4	5	6
	[0/0 , $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 1:	[– , 15/1, 11/1 , $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 3:	[– , 15/1, – , $\infty/0$, 8/3 , 9/3]					



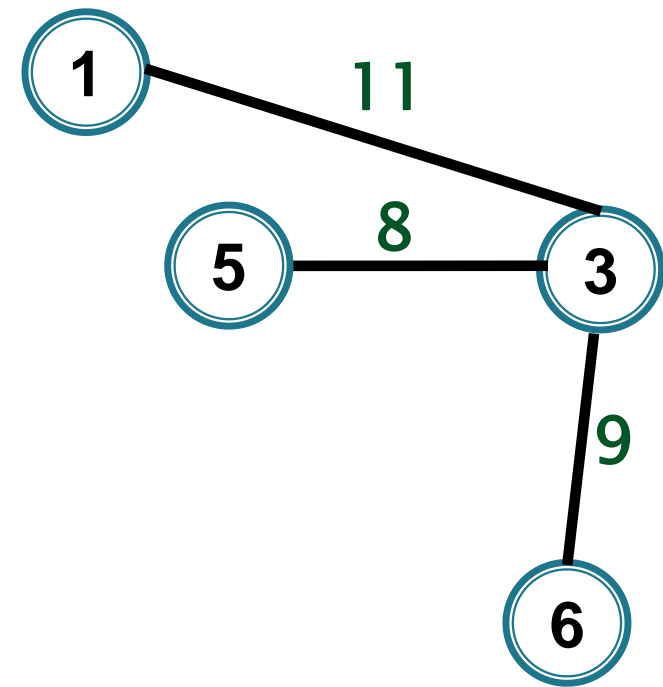
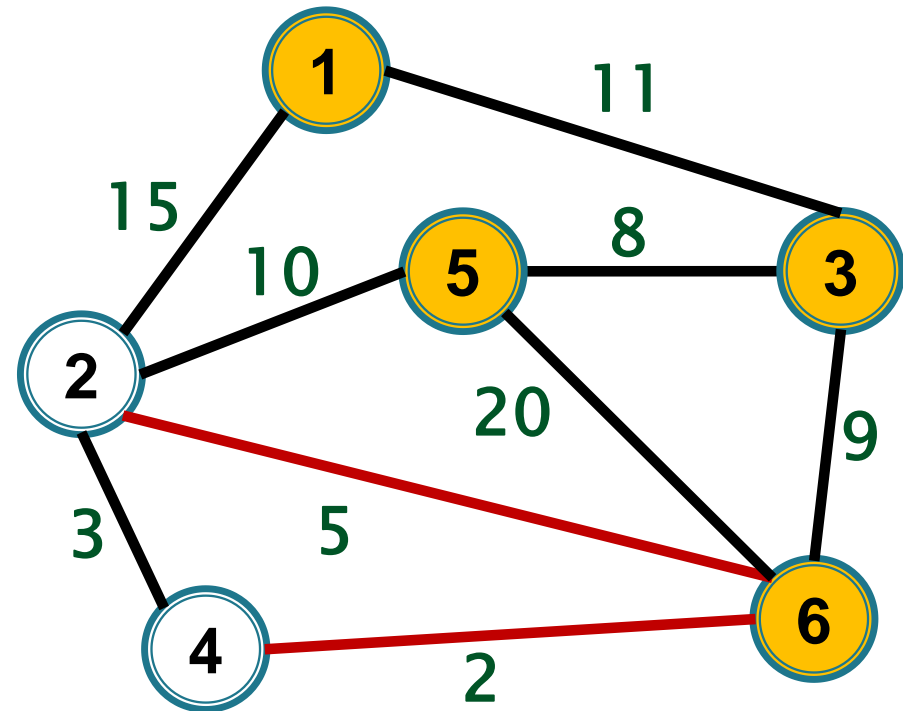
	1	2	3	4	5	6
	[0/0 , $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 1:	[– , 15/1, 11/1 , $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 3:	[– , 15/1, – , $\infty/0$, 8/3 , 9/3]					
Sel. 5:						



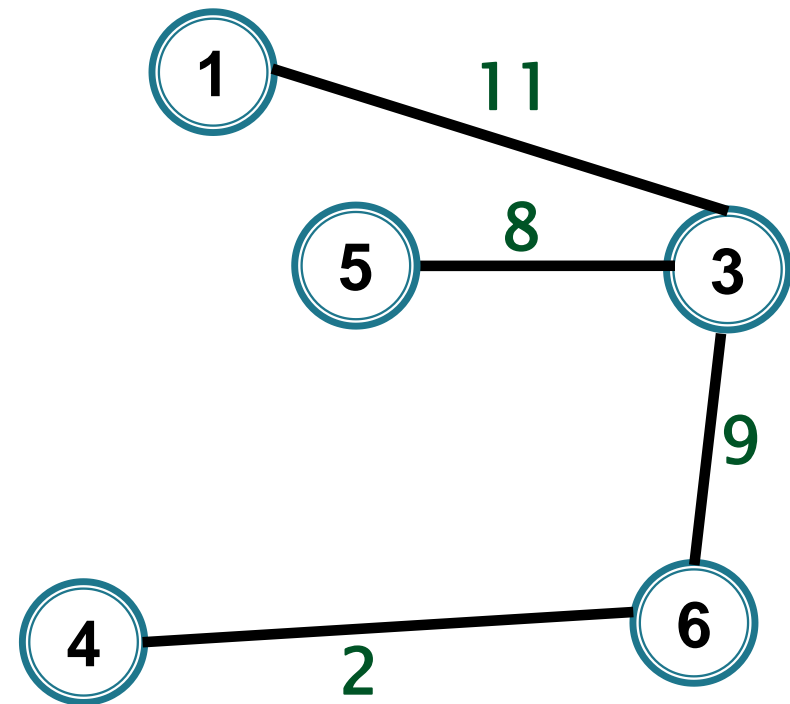
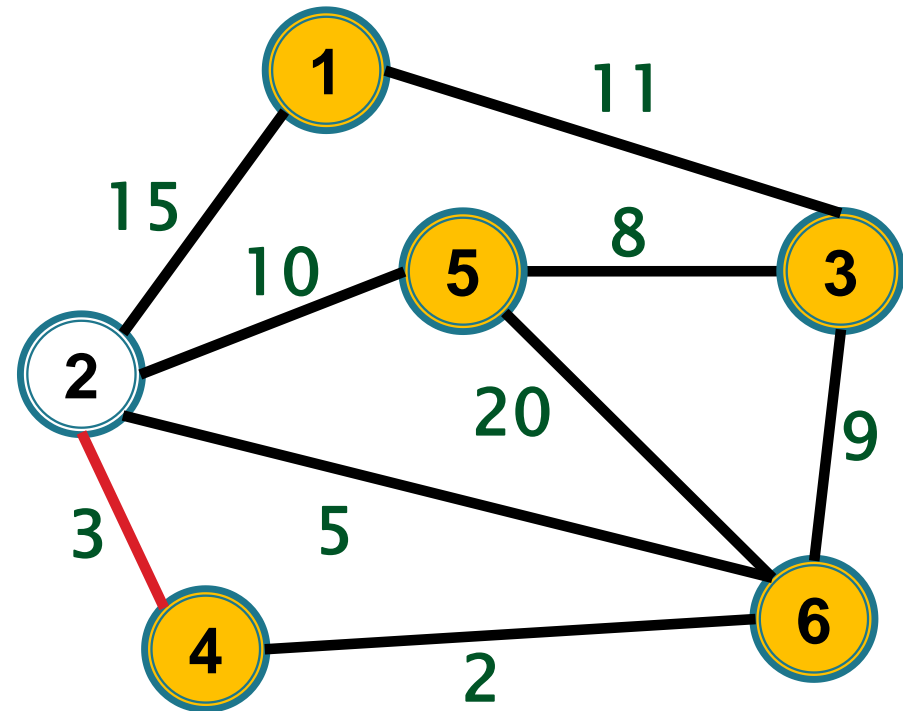
	1	2	3	4	5	6
	[0/0 , $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 1:	[– , 15/1, 11/1 , $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 3:	[– , 15/1, – , $\infty/0$, 8/3 , 9/3]					
Sel. 5:	[– , 10/5 , – , $\infty/0$, – , 9/3]					



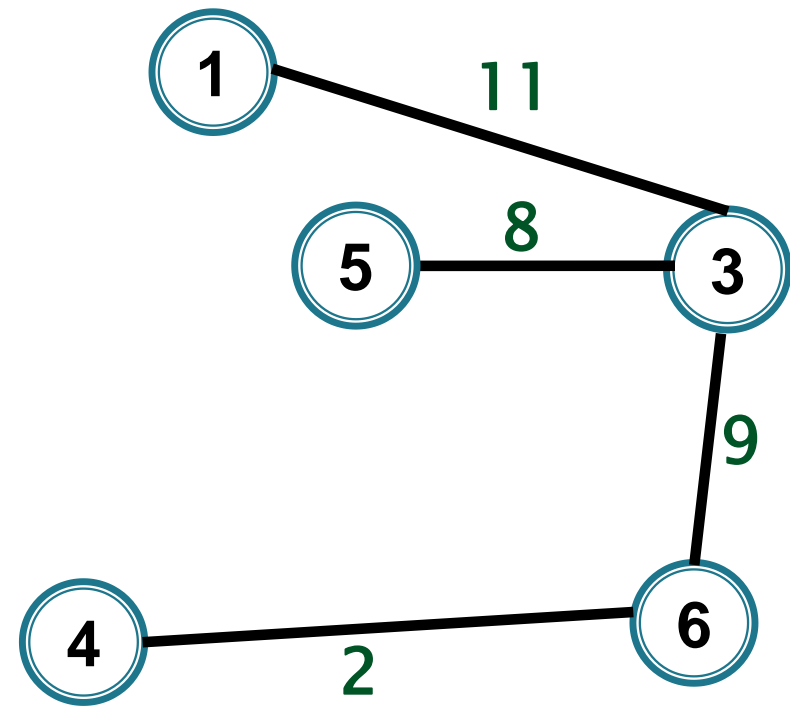
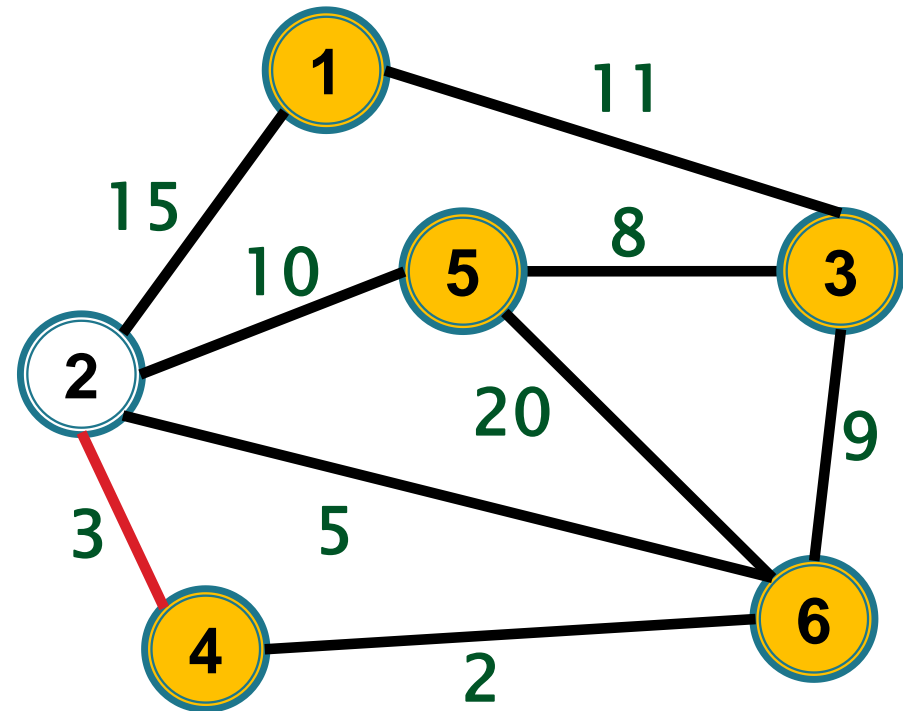
	1	2	3	4	5	6
	[0/0 , $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 1:	[– , 15/1, 11/1 , $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 3:	[– , 15/1, – , $\infty/0$, 8/3 , 9/3]					
Sel. 5:	[– , 10/5 , – , $\infty/0$, – , 9/3]					
Sel. 6:						



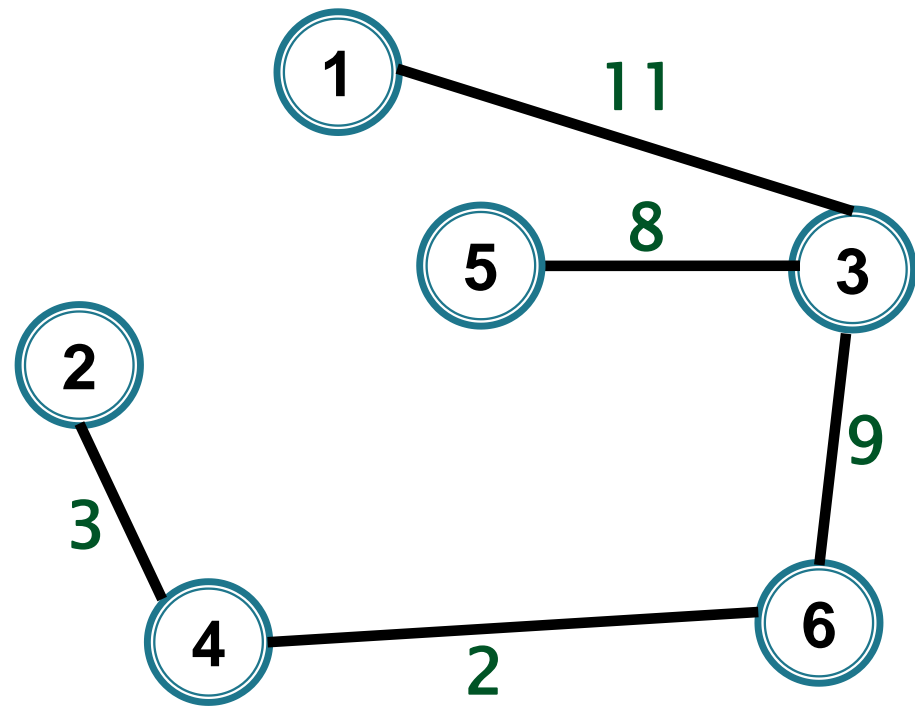
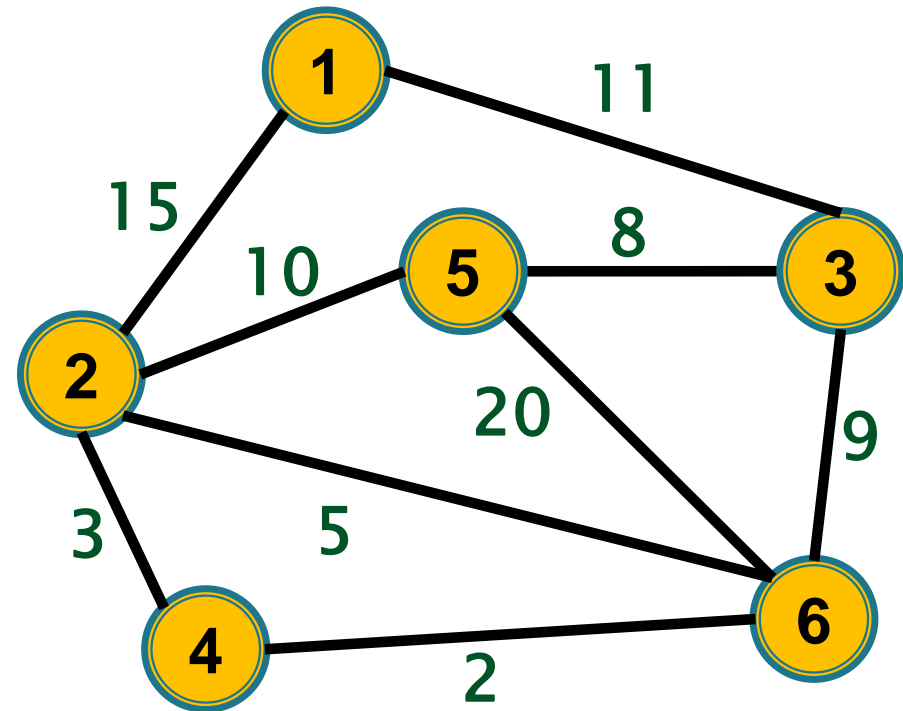
	1	2	3	4	5	6
	[0/0 , $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 1:	[– , 15/1, 11/1 , $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 3:	[– , 15/1, – , $\infty/0$, 8/3 , 9/3]					
Sel. 5:	[– , 10/5, – , $\infty/0$, – , 9/3]					
Sel. 6:	[– , 5/6 , – , 2/6 , – , –]					



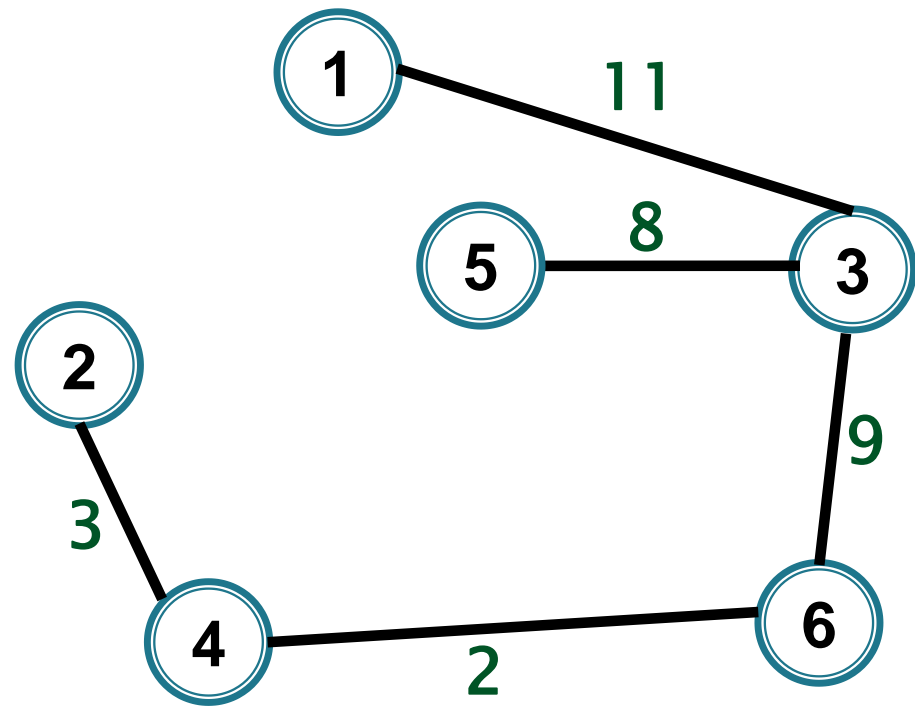
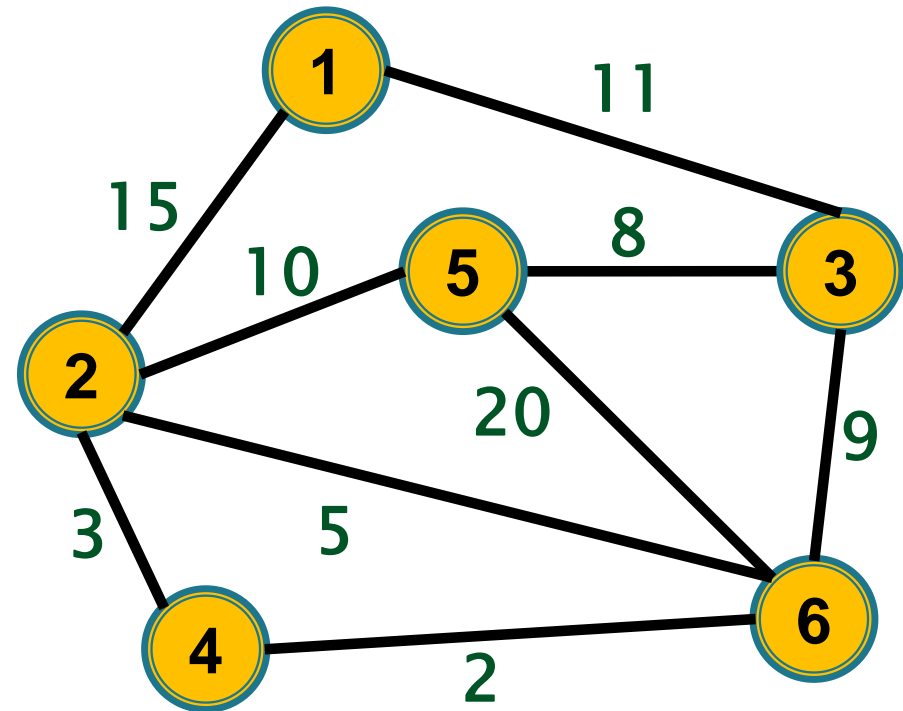
	1	2	3	4	5	6
	[0/0 , $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 1:	[– , 15/1, 11/1 , $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 3:	[– , 15/1, – , $\infty/0$, 8/3 , 9/3]					
Sel. 5:	[– , 10/5, – , $\infty/0$, – , 9/3]					
Sel. 6:	[– , 5/6, – , 2/6 , – , –]					
Sel. 4:						



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



	1	2	3	4	5	6
	[0/0 , $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 1:	[- , 15/1, 11/1 , $\infty/0$, $\infty/0$, $\infty/0$]					
Sel. 3:	[- , 15/1, - , $\infty/0$, 8/3 , 9/3]					
Sel. 5:	[- , 10/5 , - , $\infty/0$, - , 9/3]					
Sel. 6:	[- , 5/6 , - , 2/6 , - , -]					
Sel. 4:	[- , 3/4 , - , - , - , -]					
Sel. 2:						



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

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

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

► Prim

- s – vârful de start
- inițializează Q cu V
- pentru fiecare $u \in V$ executa
 - $d[u] = \infty$; $tata[u] = 0$
 - $d[s] = 0$
- cat timp $Q \neq \emptyset$ executa \Leftrightarrow pentru $i = 1, n - 1$

► Prim

- s – vârful de start
- inițializează Q cu V
- pentru fiecare $u \in V$ executa
 - $d[u] = \infty$; $tata[u] = 0$
 - $d[s] = 0$
- cat timp $Q \neq \emptyset$ executa
 - extrage un vârf $u \in Q$ cu eticheta $d[u]$ minimă

► Prim

- s – vârful de start
- inițializează Q cu V
- pentru fiecare $u \in V$ executa
 - $d[u] = \infty$; $tata[u] = 0$
 - $d[s] = 0$
- cat timp $Q \neq \emptyset$ executa
 - extrage un vârf $u \in Q$ cu eticheta $d[u]$ minimă
 - pentru fiecare $uv \in E$ executa

► Prim

- s – vârful de start
- inițializează Q cu V
- pentru fiecare $u \in V$ executa
 - $d[u] = \infty$; $tata[u] = 0$
 - $d[s] = 0$
- cat timp $Q \neq \emptyset$ executa
 - extrage un vârf $u \in Q$ cu eticheta $d[u]$ minimă
 - pentru fiecare $uv \in E$ executa
 - daca $v \in Q$ 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 $u \in V$ executa
 - $d[u] = \infty$; $tata[u] = 0$
 - $d[s] = 0$
- cat timp $Q \neq \emptyset$ executa
 - extrage un vârf $u \in Q$ cu eticheta $d[u]$ minimă
 - pentru fiecare $uv \in E$ 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

Complexitate

- ▶ Inițializări →
- ▶ n * extragere vârf minim →
- ▶ actualizare etichete vecini →

Prim



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

Prim

Varianta 1 – Folosim vector de vizitat

$Q[u] = 1$, dacă $u \notin Q$
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 $\rightarrow O(n)$
 - ▶ n * extragere vârf minim $\rightarrow O(n^2)$
 - ▶ actualizare etichete vecini $\rightarrow O(m)$
-
- $O(n^2)$

Prim

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

- ▶ Inițializare Q \rightarrow
 - ▶ n * extragere vârf minim \rightarrow
 - ▶ actualizare etichete vecini \rightarrow
-

Prim(G, w, s)

pentru fiecare $u \in V$ executa

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

$d[s] = 0$

inițializează Q cu V

cat timp $Q \neq \emptyset$ executa

$u = \text{extrage vârf cu eticheta } d \text{ minimă din } Q$

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

Prim(G, w, s)

pentru fiecare $u \in V$ executa

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

$d[s] = 0$

inițializează Q cu V

cat timp $Q \neq \emptyset$ executa

$u = \text{extrage vârf cu eticheta } d \text{ minimă din } Q$

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

Prim

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

- ▶ Inițializare Q $\rightarrow O(n)$
 - ▶ n * extragere vârf minim $\rightarrow O(n \log n)$
 - ▶ actualizare etichete vecini $\rightarrow O(m \log n)$
-
- $O(m \log n)$

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

Corectitudine

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 ?

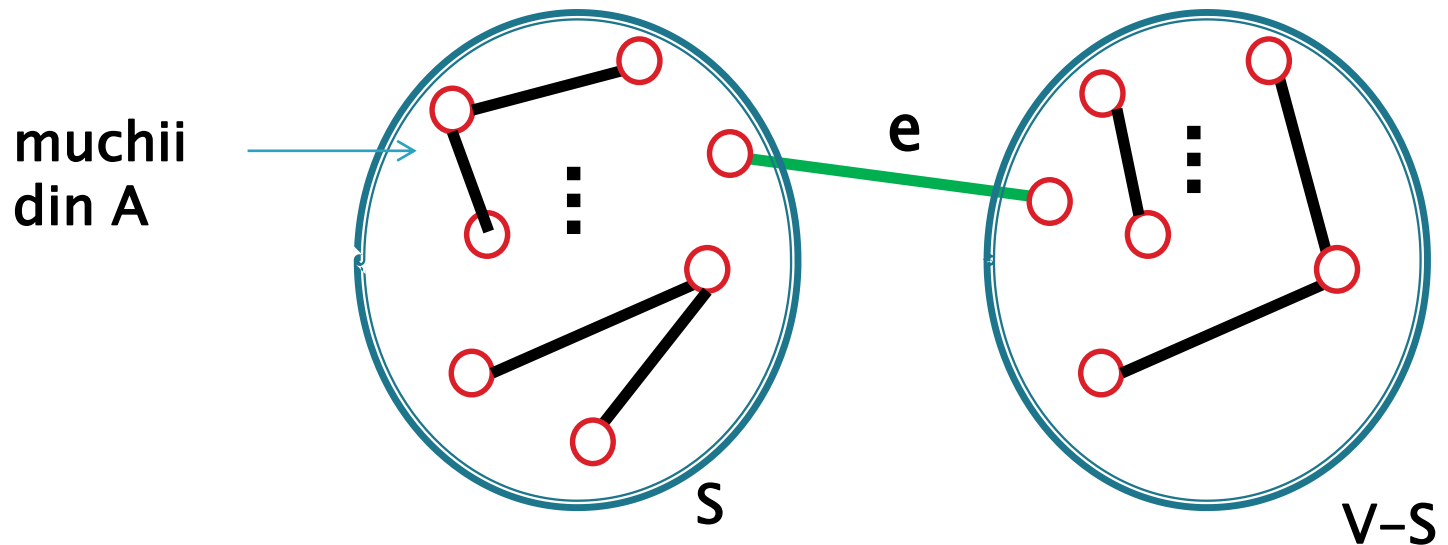
Corectitudine Kruskal + Prim

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

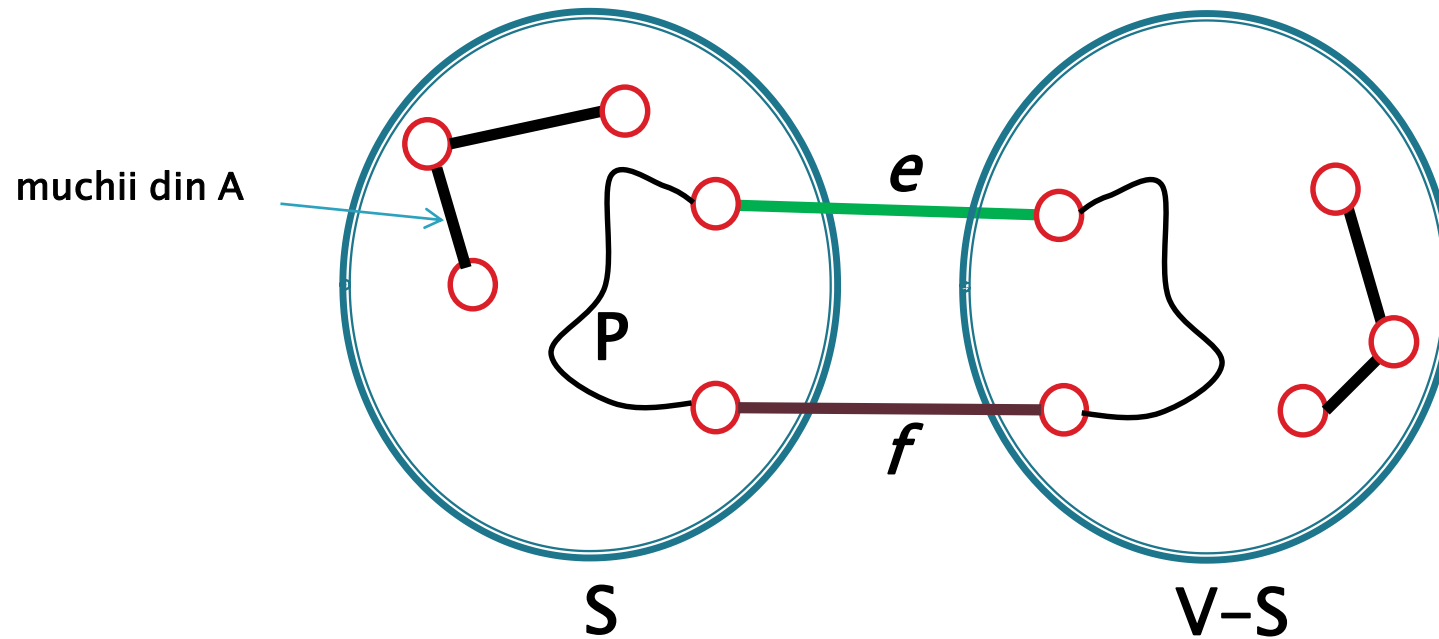
- $A \leftarrow \emptyset$ (mulțimea muchiilor selectate în arborele construit)
- pentru $i = 1, n-1$ execută
 - alege o muchie e astfel încât $A \cup \{e\} \subseteq \text{apcm}$
 - $A = A \cup \{e\}$
- returnează $T = (V, A)$

Corectitudinea algoritmilor

- **Propoziție.** Fie $G=(V, E, w)$ un graf conex ponderat și $A \subseteq E$ o submulțime a mulțimii muchiilor unui apcm al lui G .
Fie $S \subseteq V$ a.î. orice muchie din A are ambele extremități în S sau ambele extremități în $V-S$.
Fie $e=uv$ o muchie de cost minim cu o extremitate în S și cealaltă în $V-S$.
Atunci $A \cup \{e\} \subseteq \text{apcm}$.



Corectitudinea algoritmilor



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 maxim 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

$T \leftarrow T - e$

