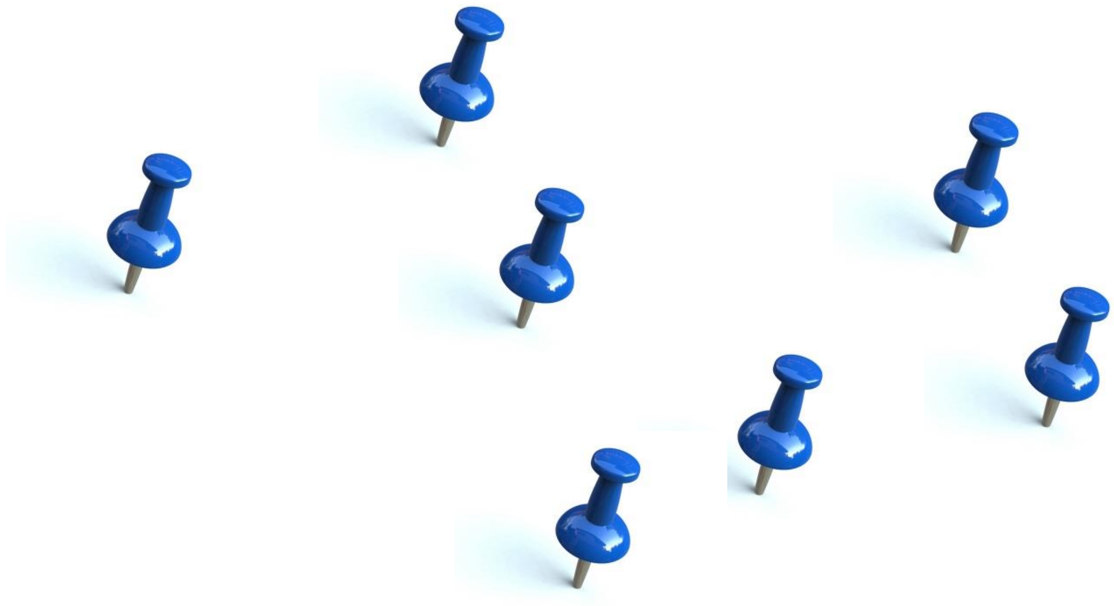
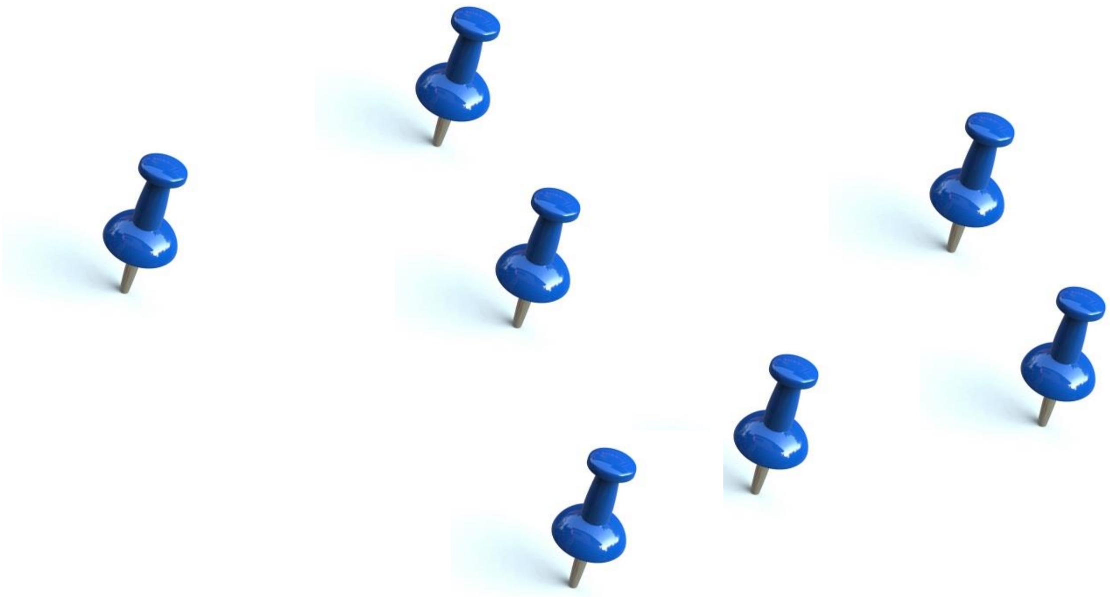
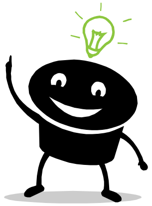
Arbori parțiali de cost minim

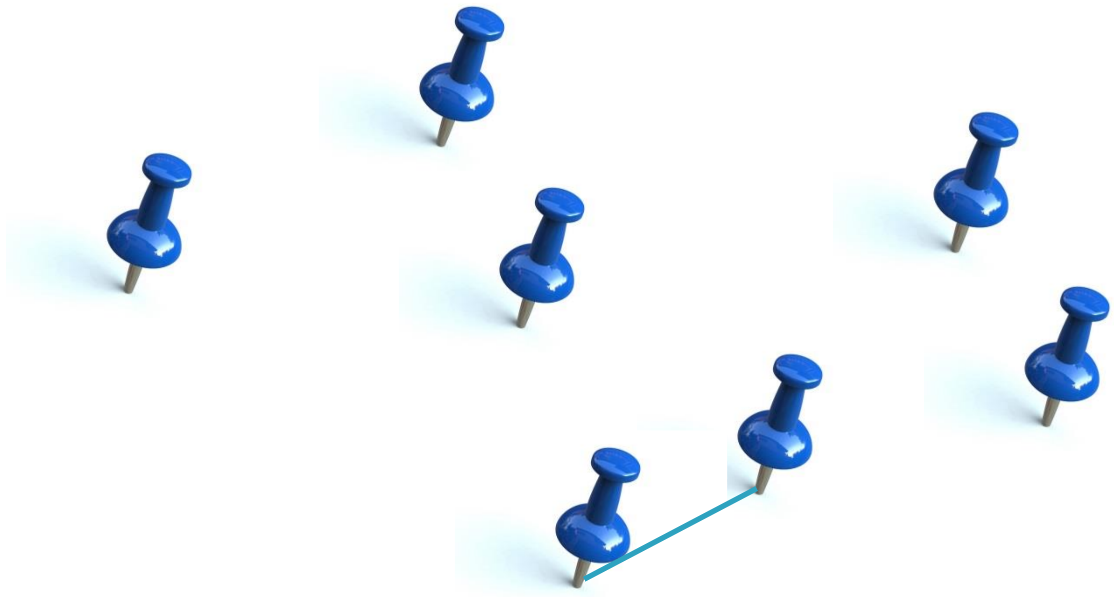


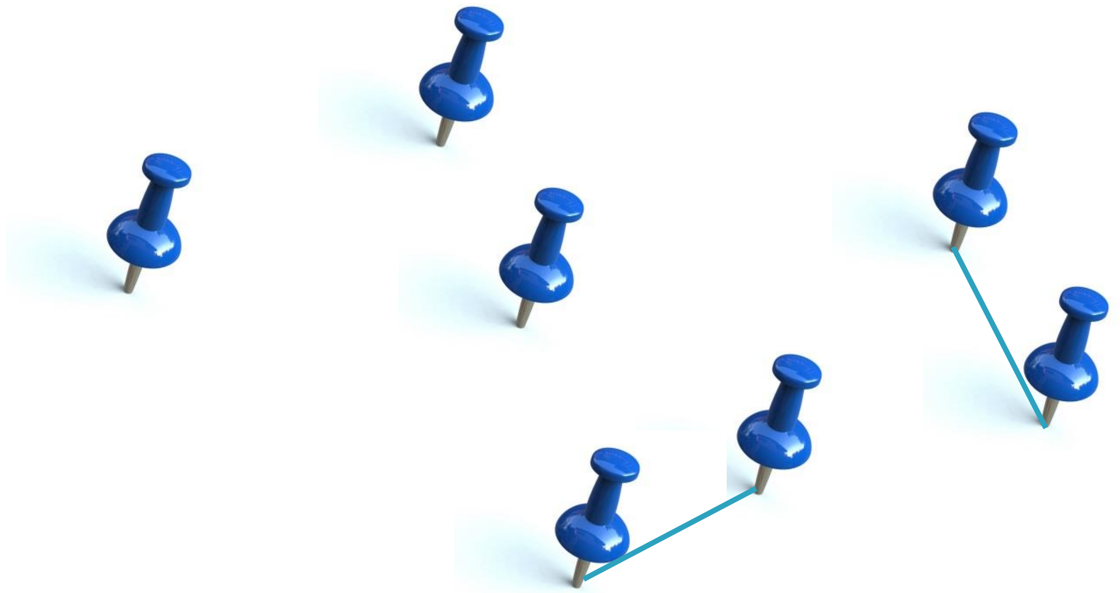
Conectați pinii astfel încât să folosiți cât mai puțin cablu

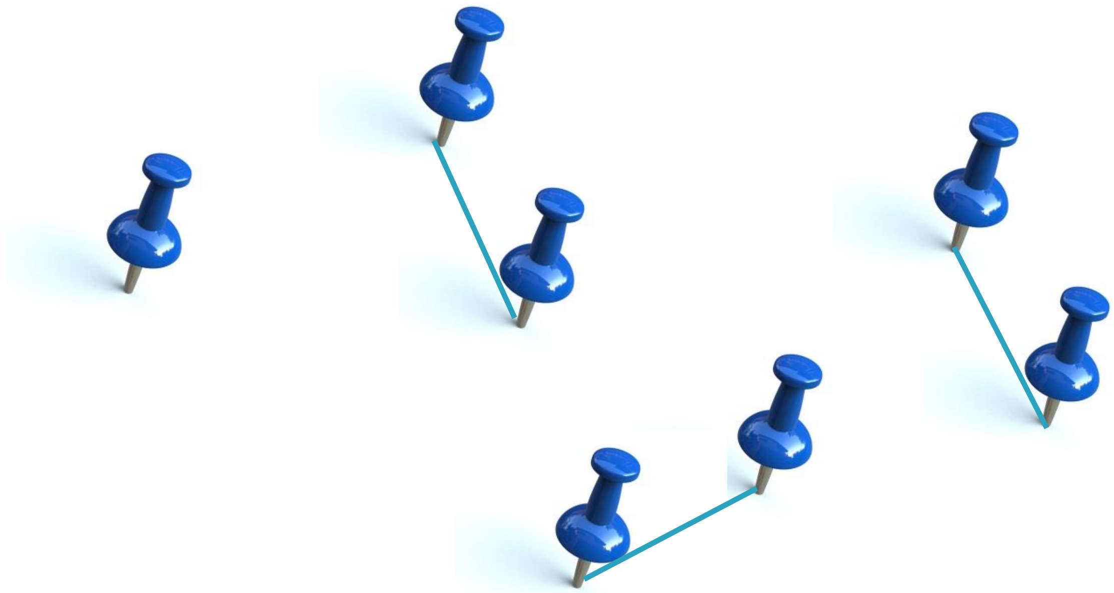


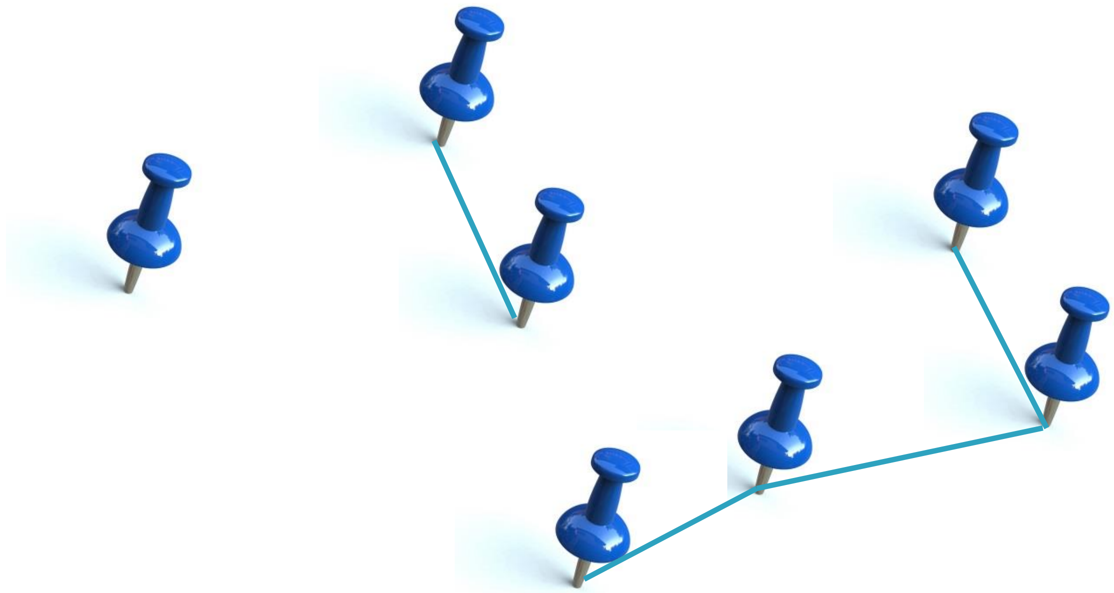


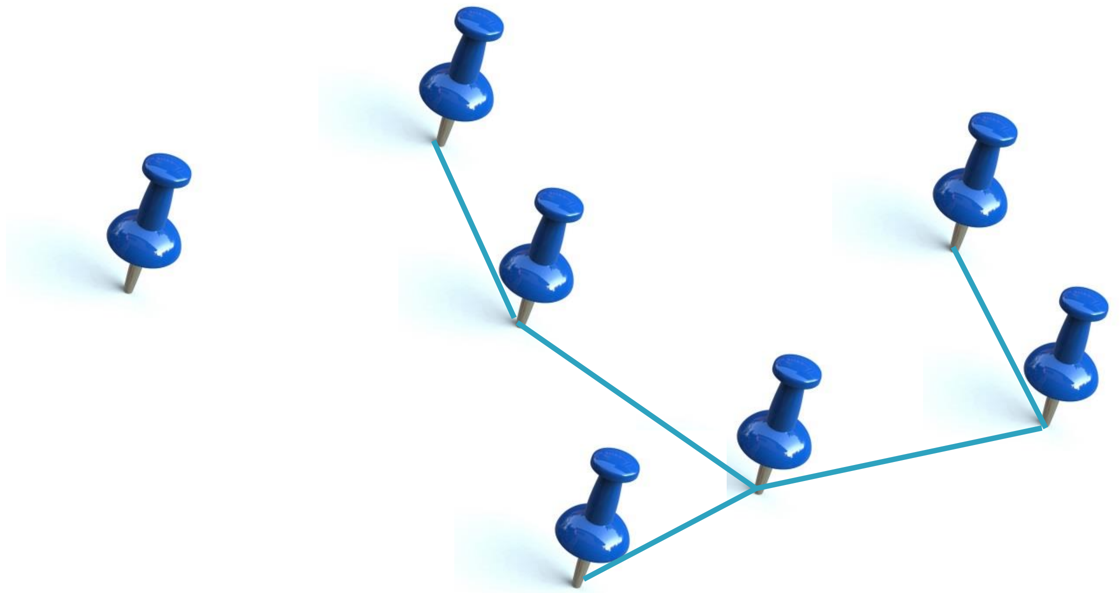
- Legăm pini apropiați
- Nu închidem cicluri

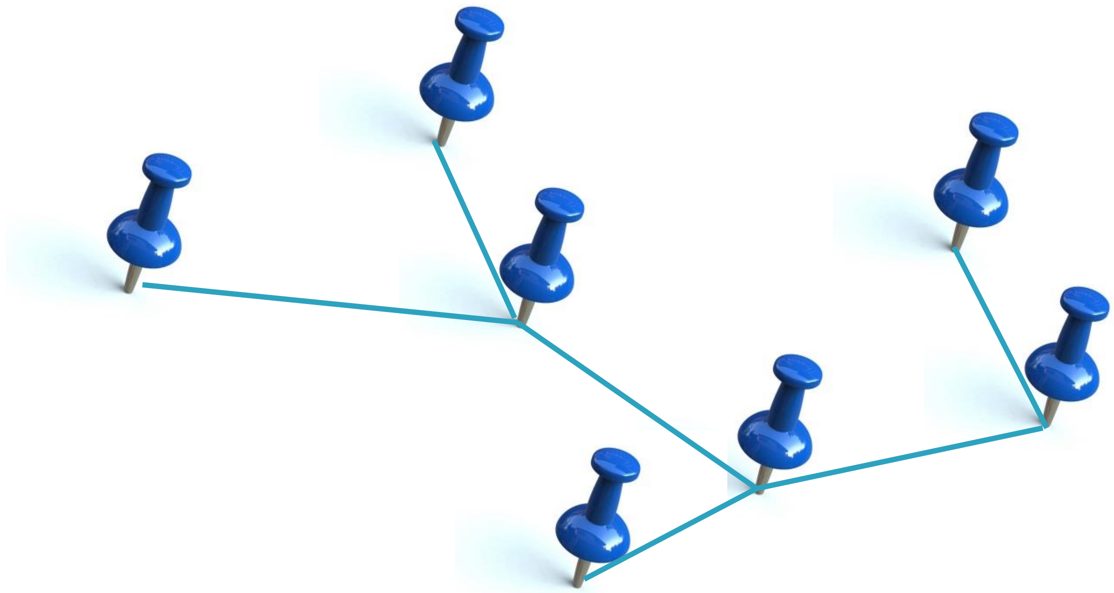


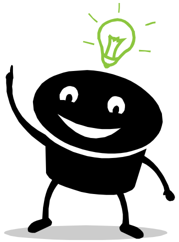












conectare cu cost minim $\Rightarrow$ evităm ciclurile

Deci trebuie să construim

graf conex + fără cicluri $\Rightarrow$ arbore

cu suma costurilor muchiilor minimă

Grafuri ponderate

Grafuri ponderate

- **$G = (V, E)$ conex ponderat**
  - $w : E \to \mathbb{R}$ funcție **pondere** (**cost**)
  - notat $G = (V, E, w)$

Grafuri ponderate

- G = (V, E, w) conex ponderat

- Pentru A $\subseteq$ E

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

Grafuri ponderate

- $G = (V, E, w)$ **conex ponderat**

- Pentru $A \subseteq E$

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

- Pentru T subgraf al lui G

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

Reprezentarea grafurilor ponderate

Grafuri ponderate

Reprezentarea grafurilor ponderate

- Matrice de costuri (ponderi)
-
-

Reprezentarea grafurilor ponderate

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

Grafuri ponderate

Reprezentarea grafurilor ponderate

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

A.p.c.m

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

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

Aplicații a.p.c.m.

- Construcția/renovarea unui sistem de căi ferate a.î.:
  - oricare două stații să fie conectate (prin căi renovate)
  - sistem economic

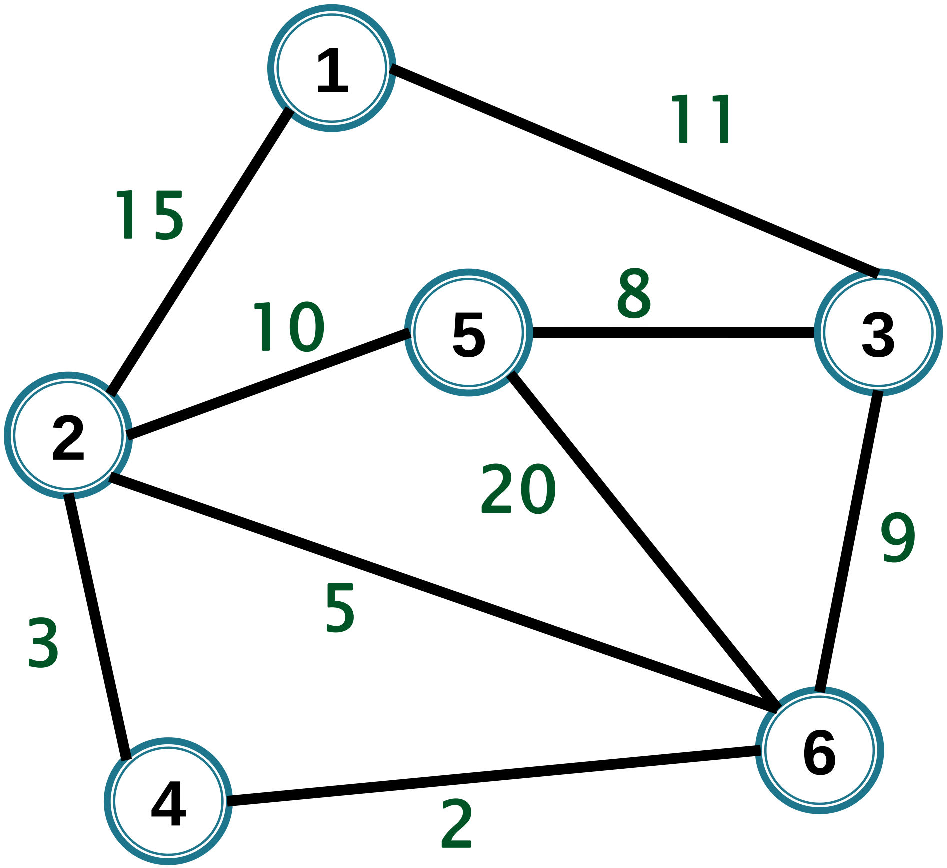
- Proiectarea circuitelor electronice
  - conectarea pinilor cu cost minim

- Clustering …

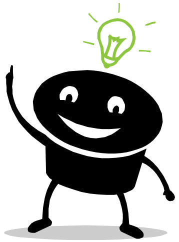
Algoritmi de determinare a unui arbore parțial de cost minim

Arbori parțiali de cost minim

Cum determinăm un arbore parțial de cost minim al unui graf conex ponderat?


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

Arbori parțiali de cost minim



Idee: **Prin adăugare succesivă de muchii,** astfel încât mulțimea de muchii selectate

- să aibă costul cât mai mic
- să fie submulțime a mulțimii muchiilor unui arbore parțial de cost minim (apcm)

Arbori parțiali de cost minim

După ce criteriu selectăm muchiile?

Algoritmul lui Kruskal

- **La un pas este selectată o muchie de cost minim care nu formează cicluri cu muchiile deja selectate (care unește două componente din graful deja construit)**

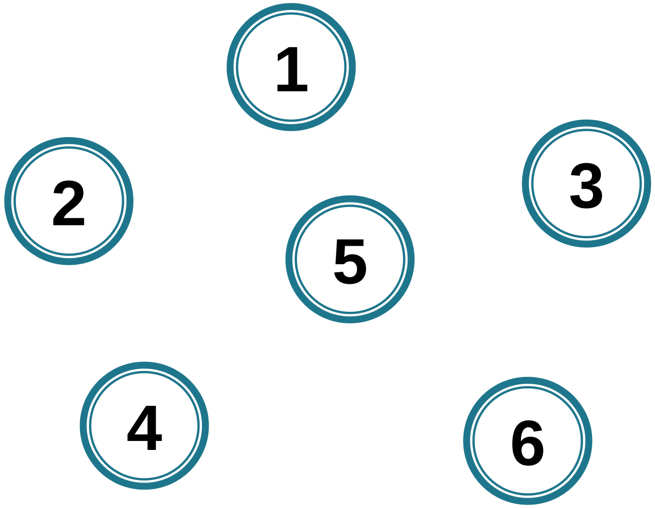
O primă formă a algoritmului

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) ∪ {uv}

Kruskal

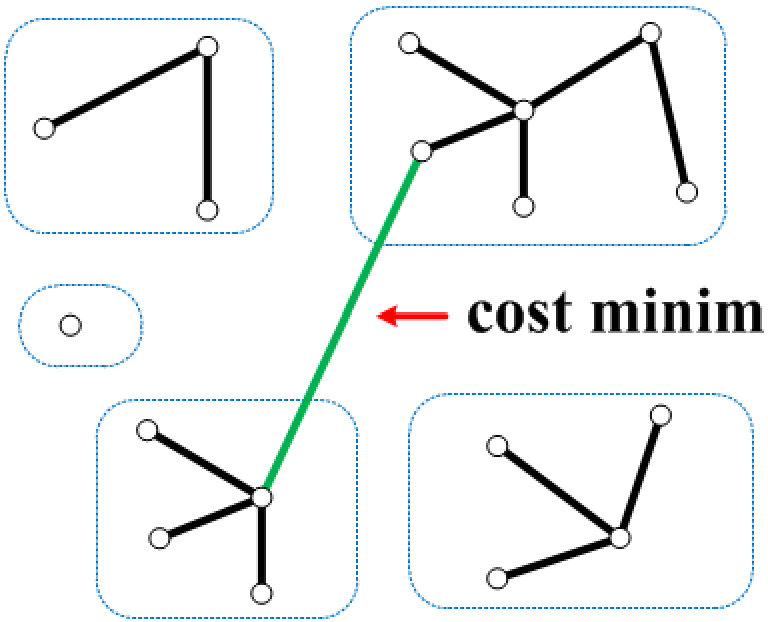
- **Inițial**: cele n vârfuri sunt **izolate**, fiecare formând o componentă conexă



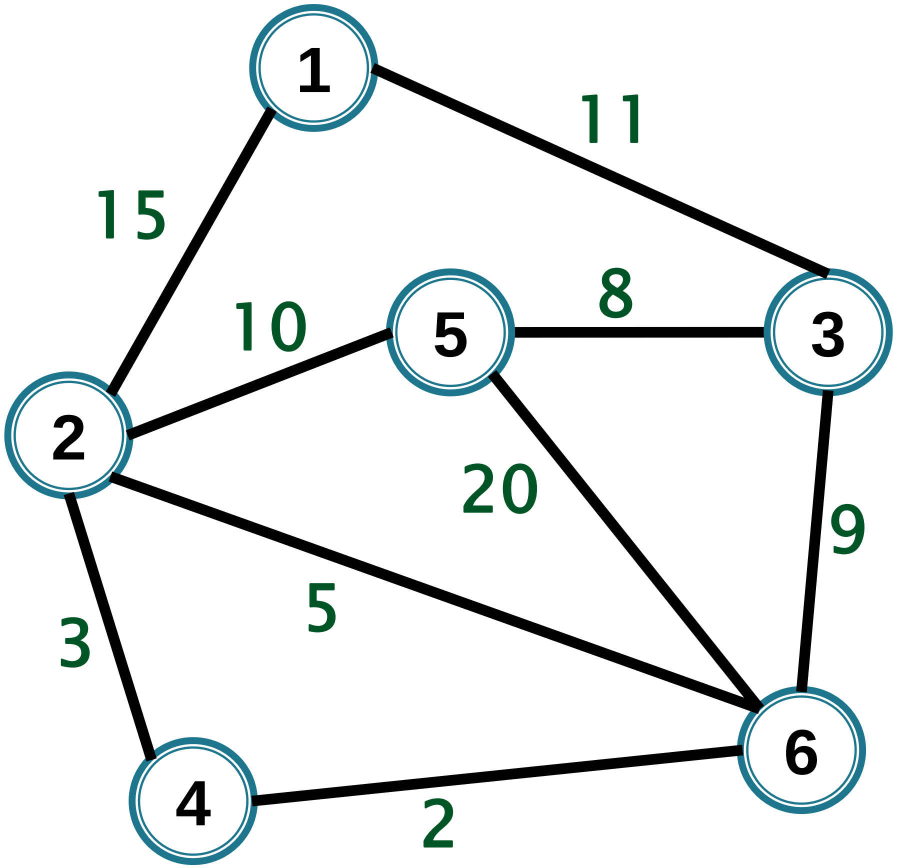
Kruskal

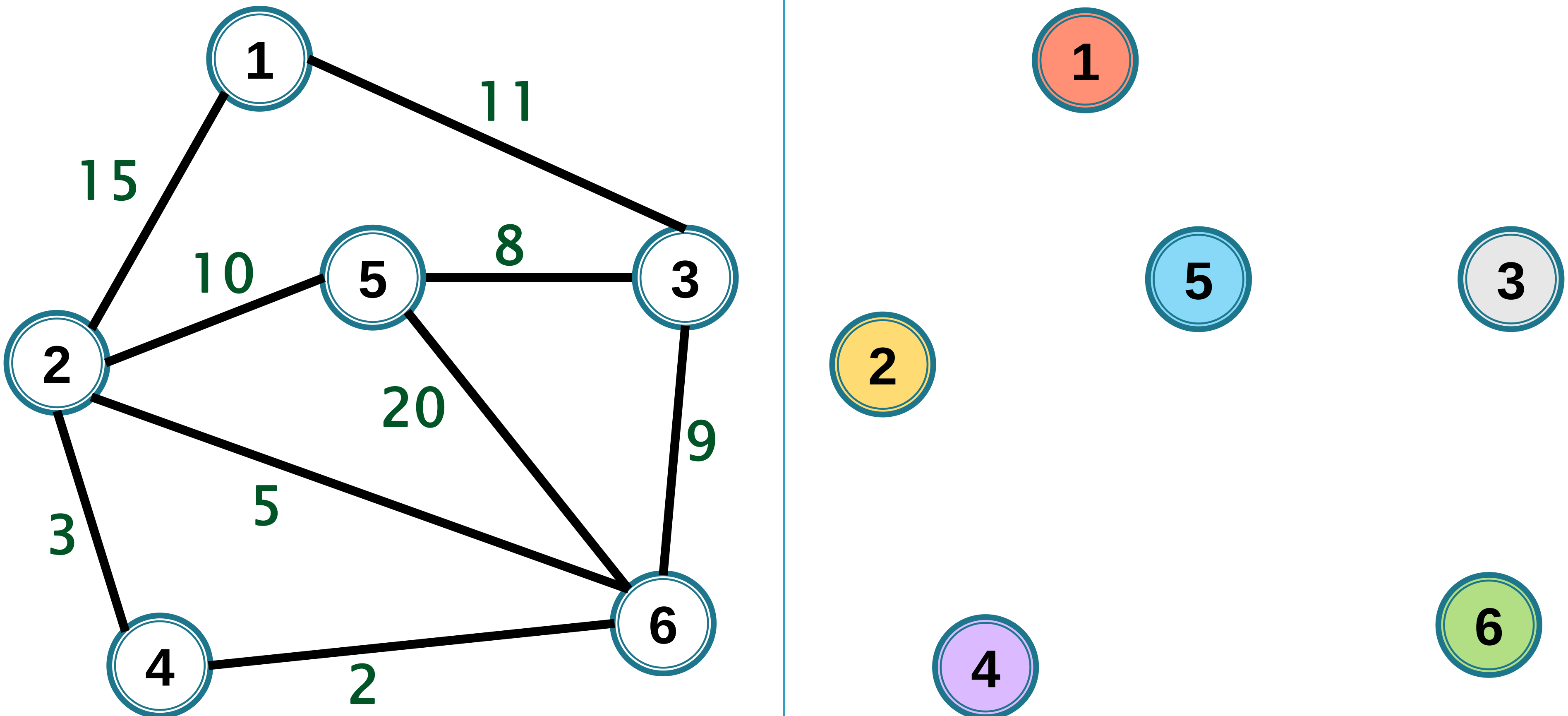
- **La un pas**:

Muchiile selectate formează o **pădure**

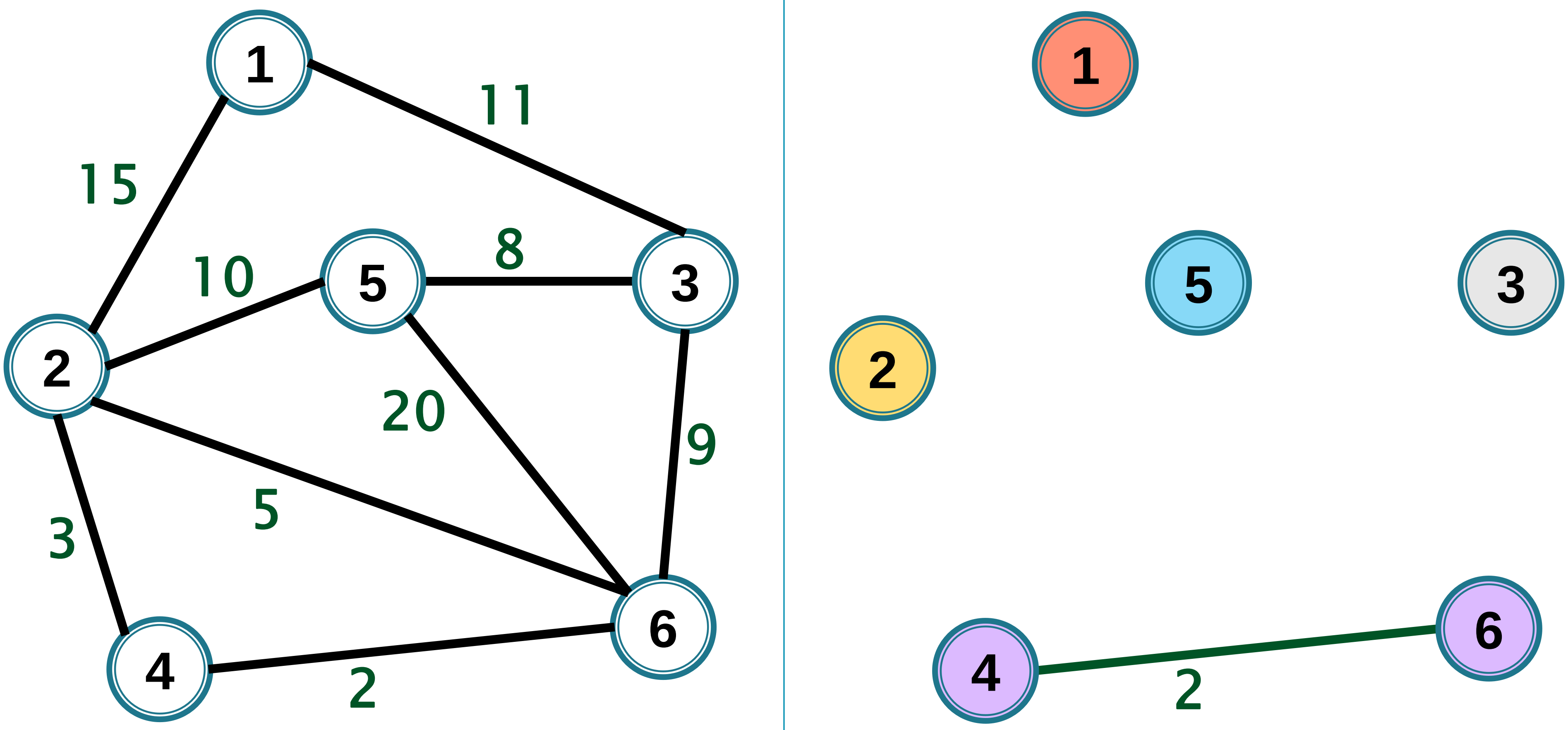


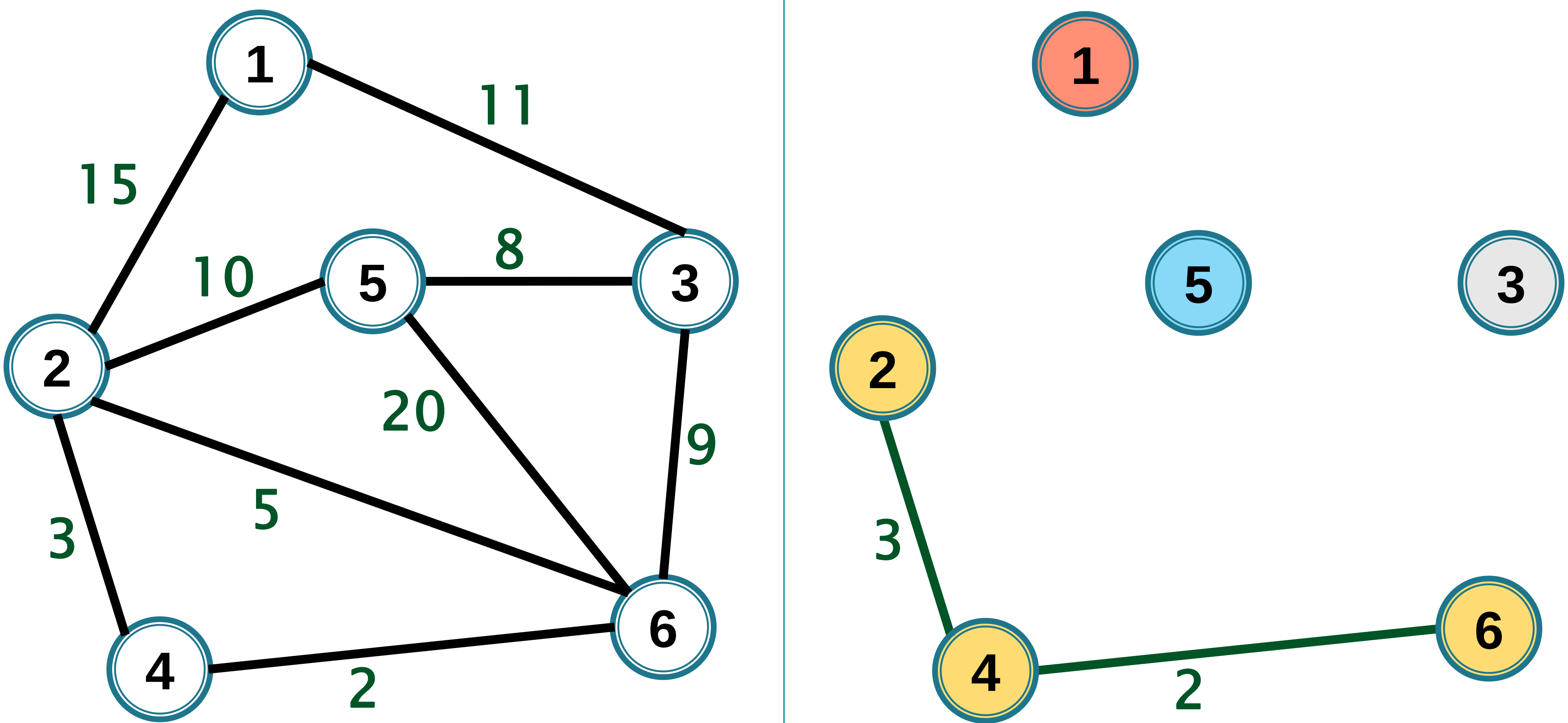
Este selectată o muchie de cost minim care unește doi arbori din pădurea curentă (două componente conexe)

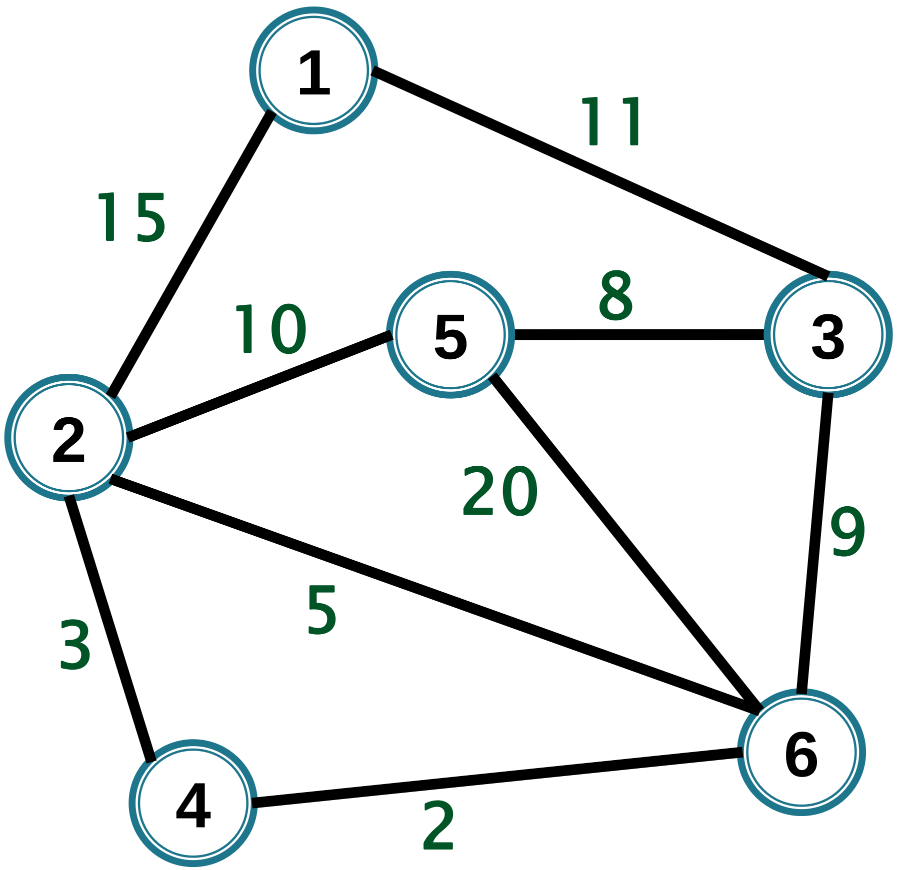

1
11
15
10
5
8
3
2
20
9
5
3
4
6
2

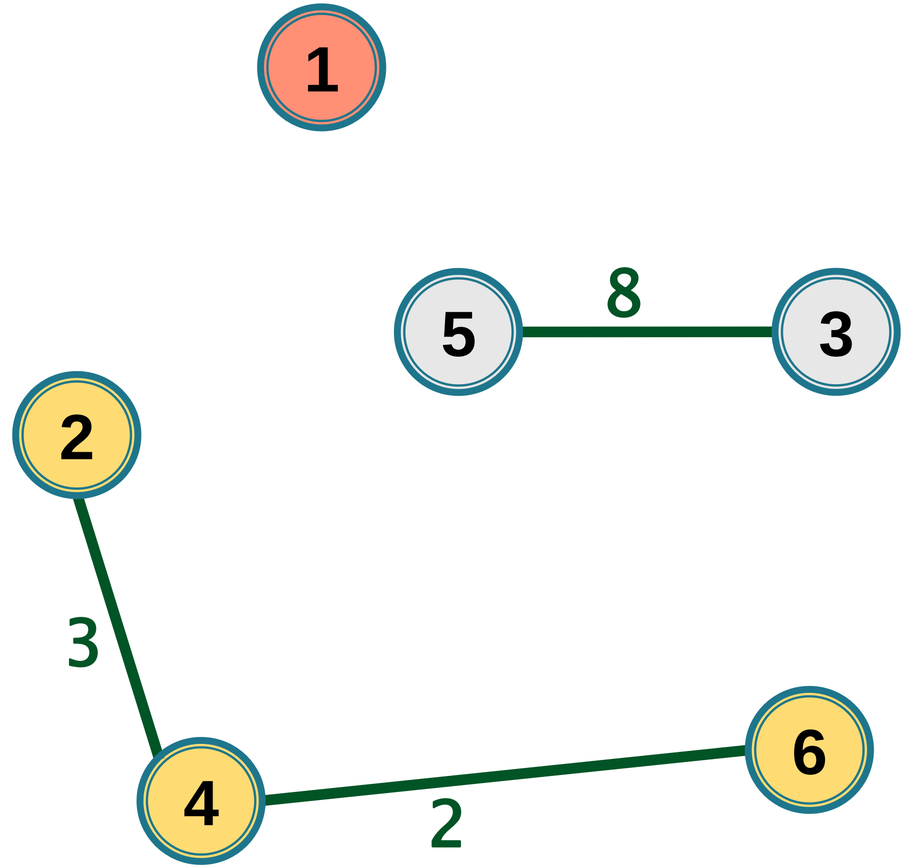


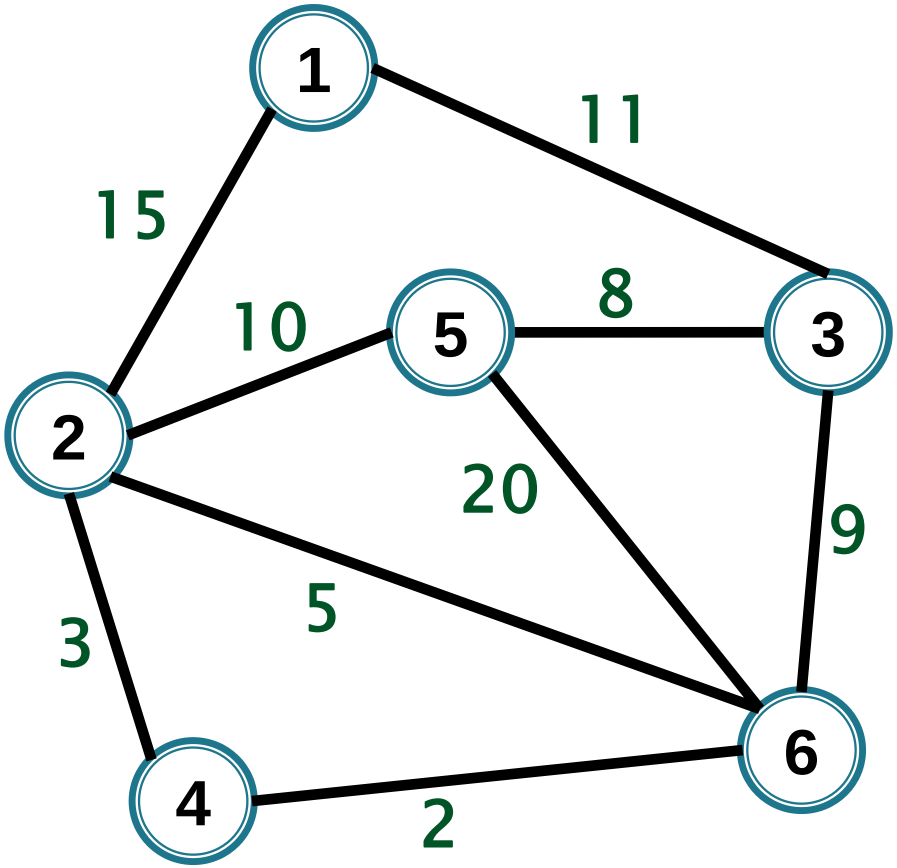
1
2
3
4
5
6
15
11
10
8
20
9
5
3
2
1
5
3
2
4
6

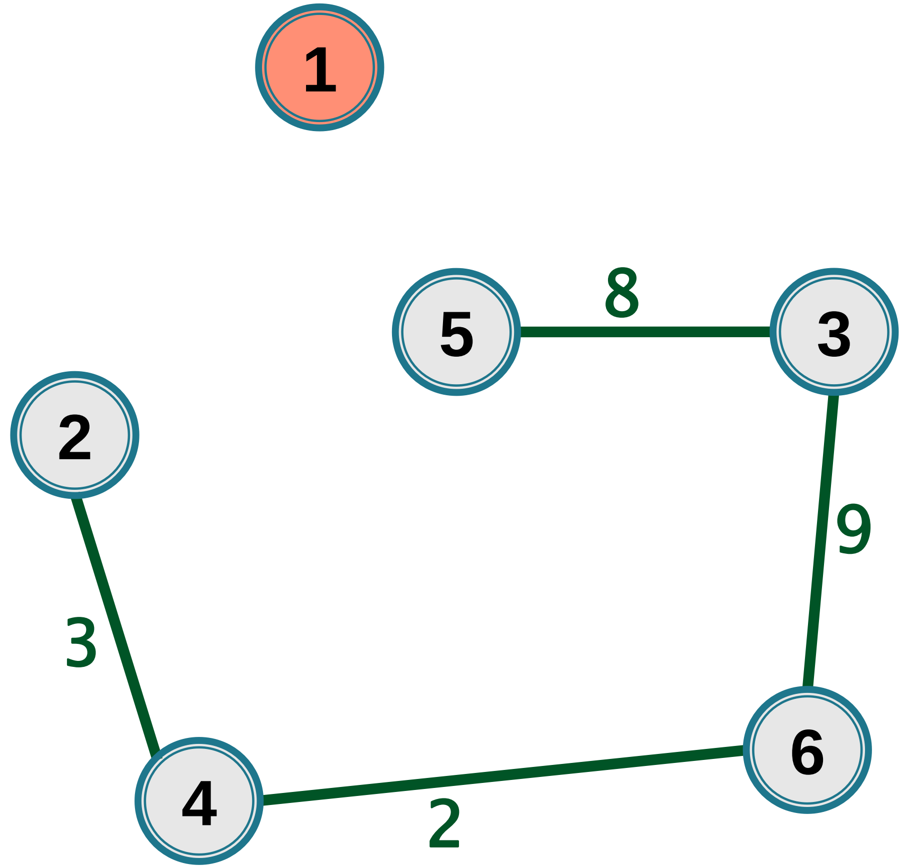

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2
1
5
3
2
4
6
2

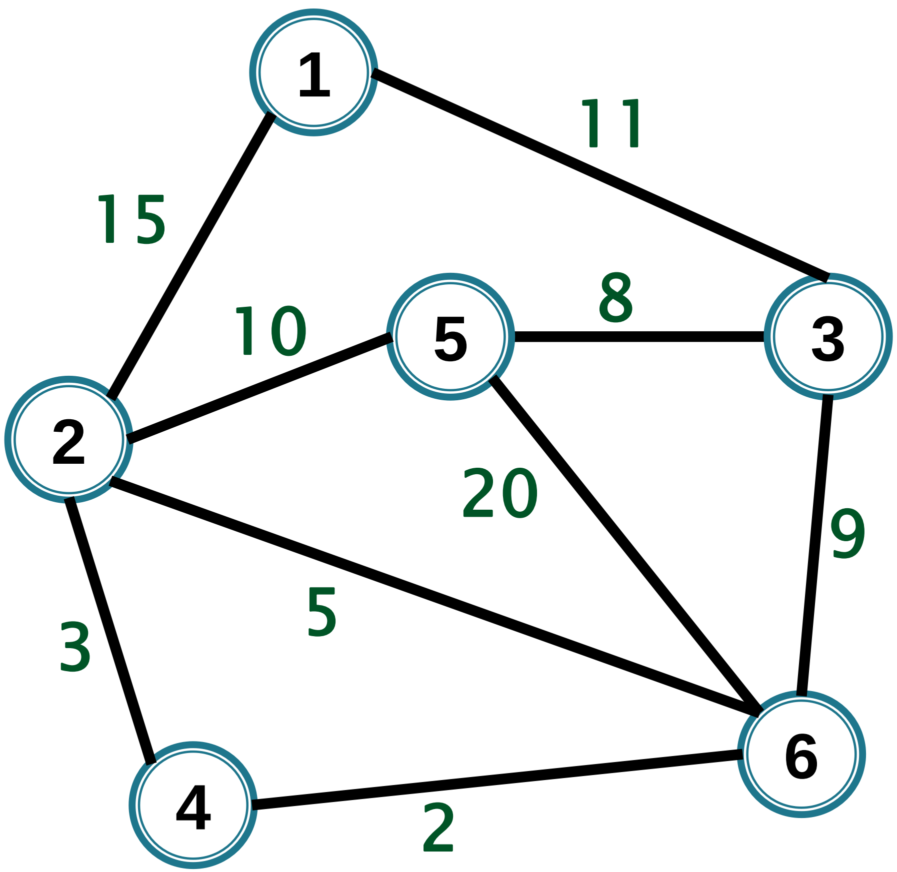

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2
1
5
3
2
4
6
3
2

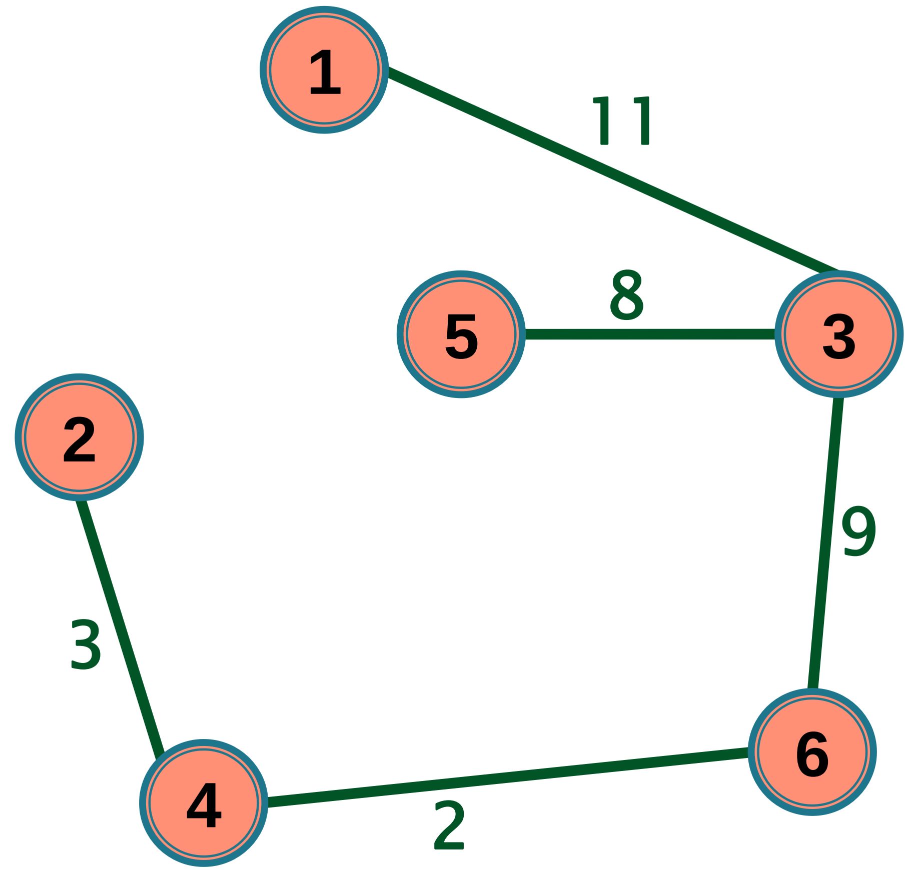

1
11
15
10
5
8
3
2
20
9
5
3
6
4
2


1
5
8
3
2
3
4
6
2


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


1
2
3
4
5
6
8
9
3
2

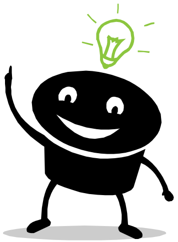

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2


1
2
3
4
5
6
11
8
9
3
2

Kruskal – Implementare

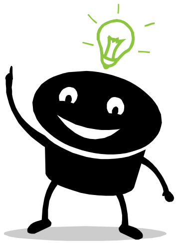
1. Cum reprezentăm graful în memorie?
2. Cum selectăm ușor o muchie:
   - de cost minim
   - care unește două componente (nu formează cicluri cu muchiile deja selectate)

Kruskal



Pentru a selecta ușor o muchie de cost minim cu proprietatea dorită **ordonăm crescător muchiile după cost şi considerăm muchiile în această ordine**

Kruskal



Reprezentarea grafului ponderat

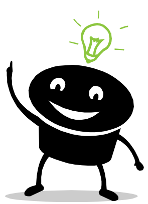
- **Listă de muchii:** memorăm pentru fiecare muchie extremitățile și costul

Kruskal

Cum testăm dacă muchia curentă unește două componente ($\Leftrightarrow$ nu formează cicluri cu muchiile deja selectate) ?

Kruskal

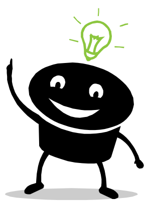
Cum testăm dacă muchia curentă unește două componente ($\Leftrightarrow$ nu formează cicluri cu muchiile deja selectate) ?



verificăm printr-o parcurgere dacă extremităţile muchiei sunt deja unite printr-un lanţ

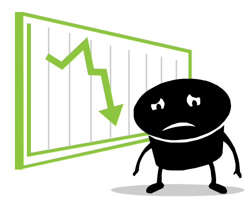
Kruskal

Cum testăm dacă muchia curentă unește două componente (⇔ nu formează cicluri cu muchiile deja selectate) ?

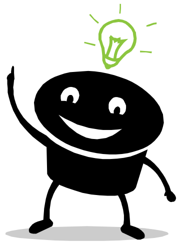


verificăm printr-o parcurgere dacă extremităţile muchiei sunt deja unite printr-un lanţ

O(mn) – ineficient



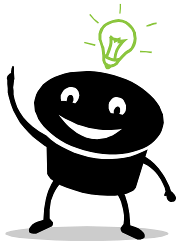
Kruskal



Componentele sunt mulţimi disjuncte din V (partiţie a lui V) – **structuri pentru mulţimi disjuncte**

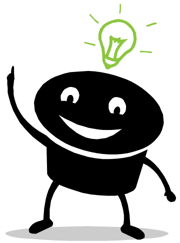
- **asociem fiecărei componente un reprezentant (o culoare)**

Kruskal



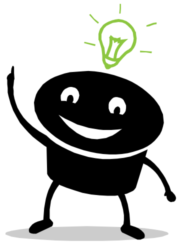
- **Operații necesare**:
  - **Initializare(u)** – creează o componentă cu un singur vârf, u

Kruskal



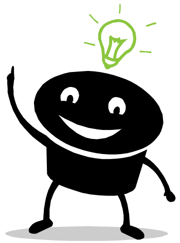
- **Operații necesare**:
  - **Initializare(u)** – creează o componentă cu un singur vârf, u
  - **Reprez(u)** – returnează reprezentantul (culoarea) componentei care conține pe u

Kruskal



- **Operații necesare**:
  - **Initializare(u)** – creează o componentă cu un singur vârf, u
  - **Reprez(u)** – returnează reprezentantul (culoarea) componentei care conține pe u
  - **Reuneste(u,v)** – unește componenta care conține u cu cea care conține v

Kruskal



- O muchie uv uneşte două componente dacă

  Reprez(u) ≠ **Reprez**(v)

Kruskal

```
sorteaza(E)
for(v=1;v<=n;v++)
     Initializare(v);
```

Kruskal

```
sorteaza(E)
for(v=1;v<=n;v++)
    Initializare(v);
nrmsel=0
for(uv ∈ E)
 if(Reprez(u)!=Reprez(v))
  {


   }
```

Kruskal

```
sorteaza(E)
for(v=1;v<=n;v++)
    Initializare(v);
nrmsel=0
for(uv ∈ E)
 if(Reprez(u)!=Reprez(v))
  {
     E(T) = E(T) ∪ {uv};
     Reuneste(u,v);
     nrmsel=nrmsel+1;
     if(nrmsel==n-1)
          STOP; //break;
  }
```

Kruskal

Complexitate

Kruskal

Complexitate

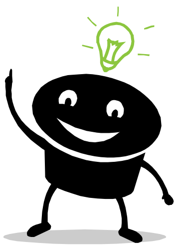
- Sortare -> O(m log m) = O(m log n)
- n * Initializare
- 2m * Reprez
- (n-1) * Reuneste

Depinde de memorarea componentelor conexe

Kruskal

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

Kruskal



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

r[u] = culoarea componentei care conține vârful u

Kruskal

- **Initializare**

- **Reprez**

- **Reuneste**

Kruskal

- Initializare – O(1)

```
void Initializare(int u){
    r[u]=u;
}
```

- Reprez
- Reuneste

Kruskal

- Initializare – O(1)

```
void Initializare(int u){
    r[u]=u;
}
```

- Reprez – O(1)

```
int Reprez(int u){
    return r[u];
}
```

- Reuneste

Kruskal

- Initializare – O(1)

```
void Initializare(int u){
    r[u]=u;
}
```

- Reprez – O(1)

```
int Reprez(int u){
    return r[u];
}
```

- Reuneste – O(n)

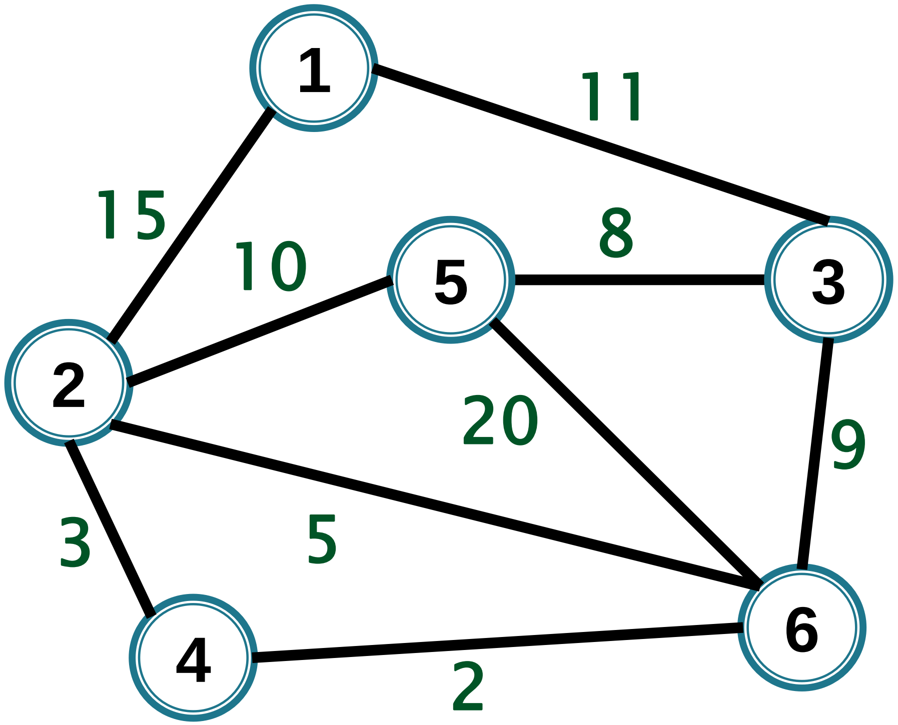
```
void Reuneste(int u,int v)
{
  r1=Reprez(u);//r1=r[u]
  r2=Reprez(v);//r2=r[v]
  for(k=1;k<=n;k++)
    if(r[k]==r2)
       r[k]=r1;
}
```

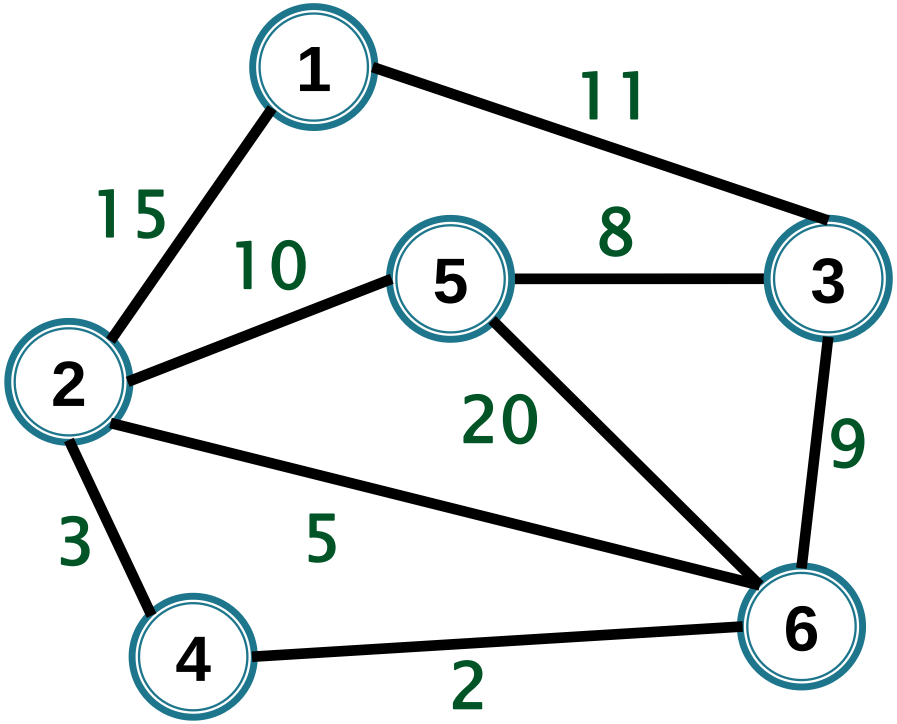
Kruskal

Varianta 1– dacă folosim vector de reprezentanți

- **Sortare** –> $O(m \log m) = O(m \log n)$
- **n * Initializare** –> $O(n)$
- **2m * Reprez** –> $O(m)$
- **(n-1) * Reuneste** –> $O(n^2)$

$$\mathbf{O(m \log n + n^2)}$$


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



(4,6)

(2,4)

(2,6)

(3,5)

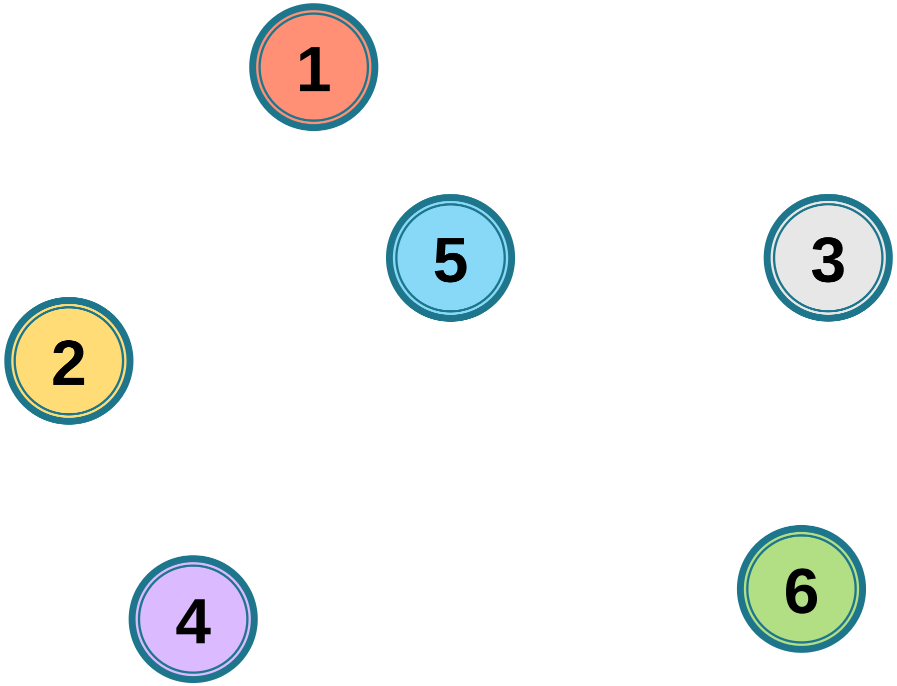
(3,6)

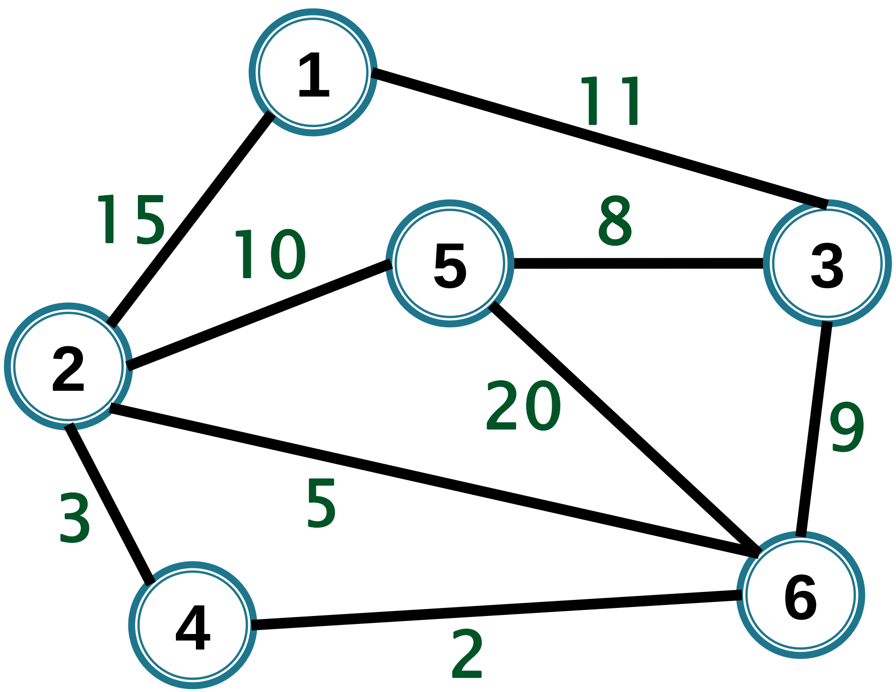
(2,5)

(1,3)

(1,2)

(5,6)





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

(4,6)

(2,4)

(2,6)

(3,5)

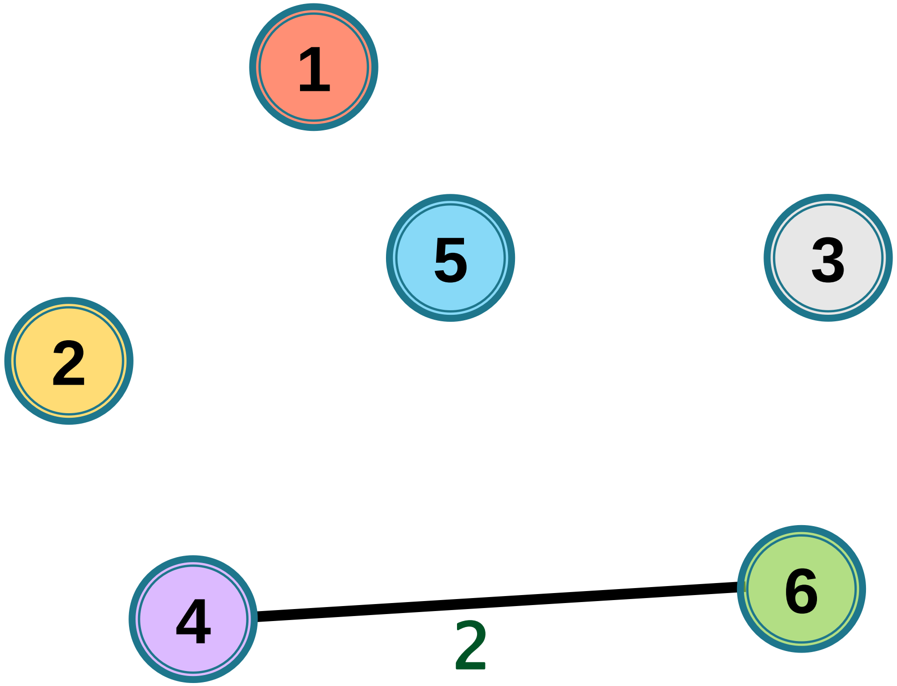
(3,6)

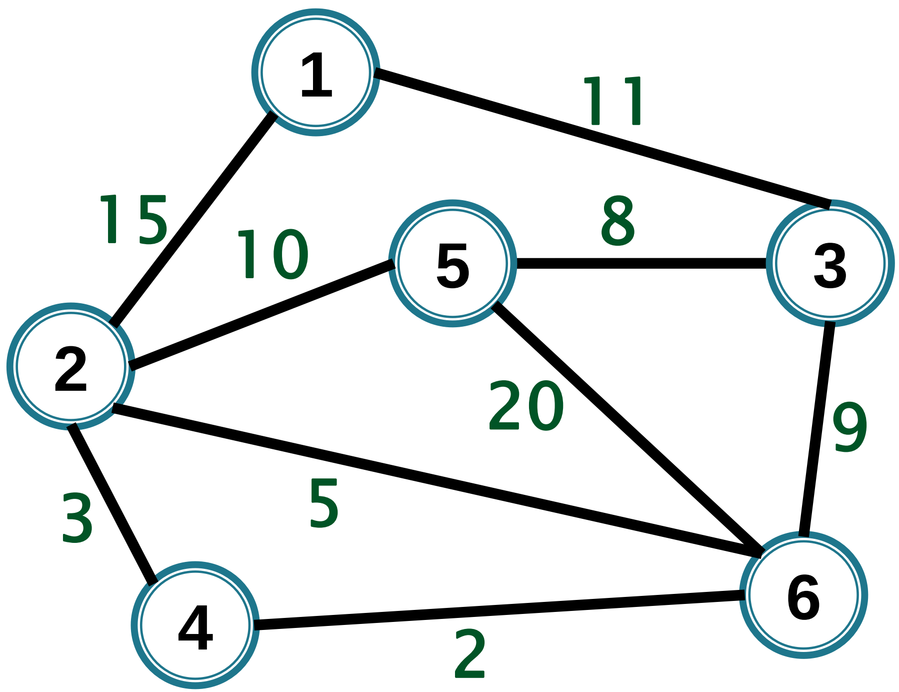
(2,5)

(1,3)

(1,2)

(5,6)





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

(4,6) $r(4) \neq r(6)$

(2,4)

(2,6)

(3,5)

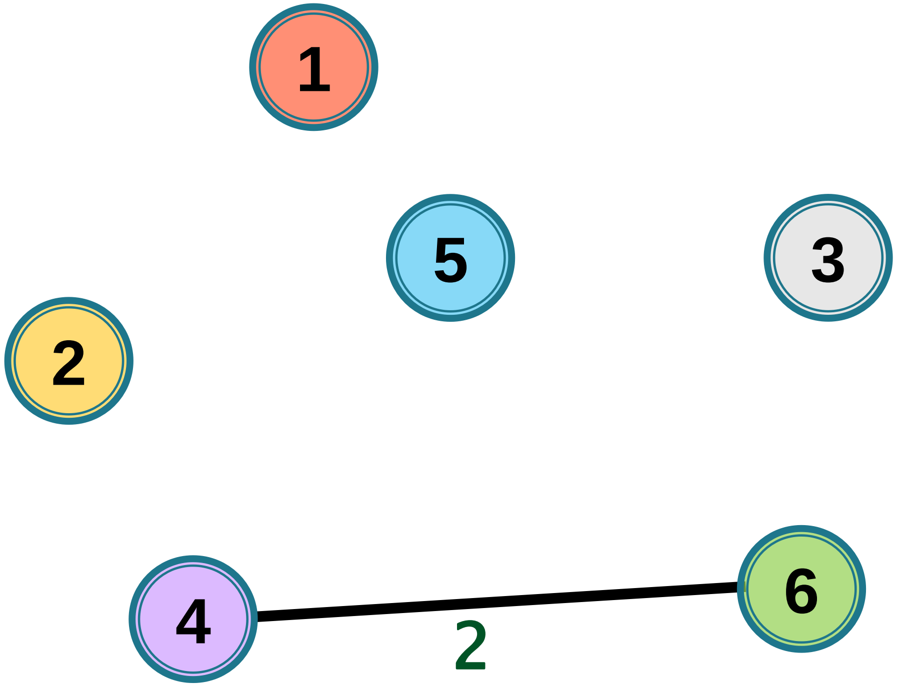
(3,6)

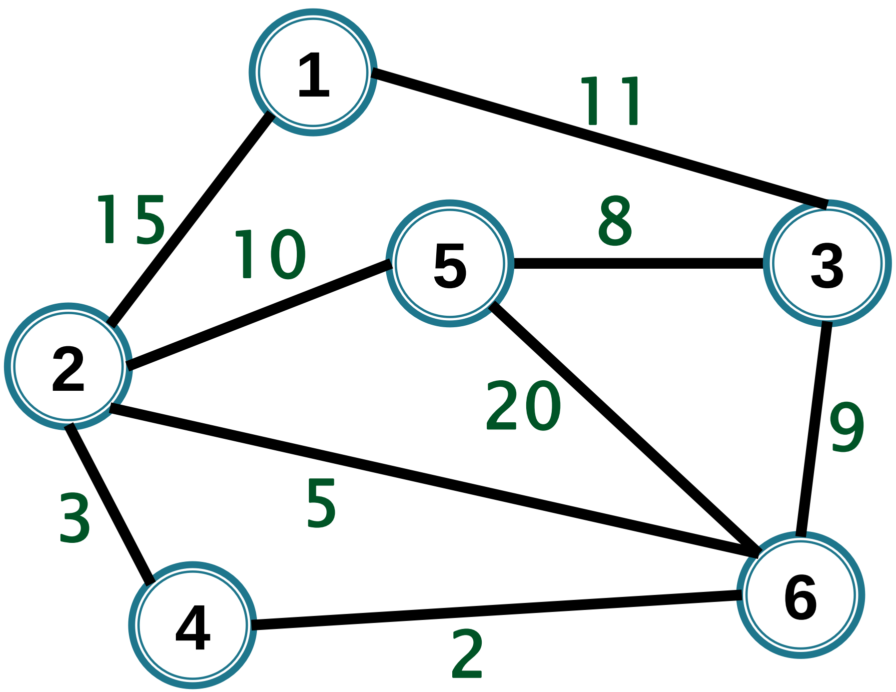
(2,5)

(1,3)

(1,2)

(5,6)





r = [1,2,3,<u>4</u>,5,<u>6</u>]

(4,6) Reuneste(4, 6)

(2,4)

(2,6)

(3,5)

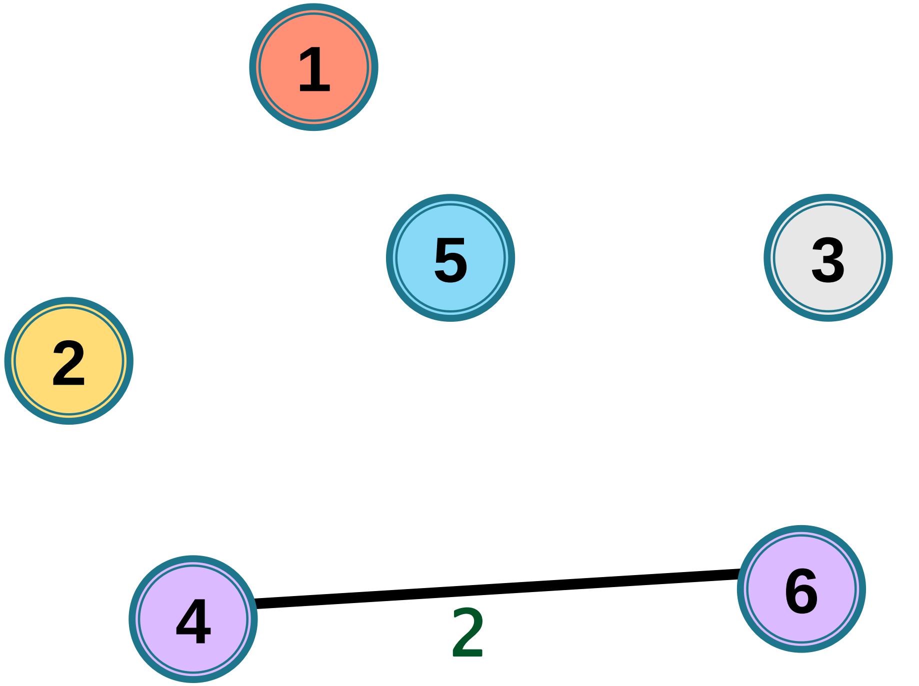
(3,6)

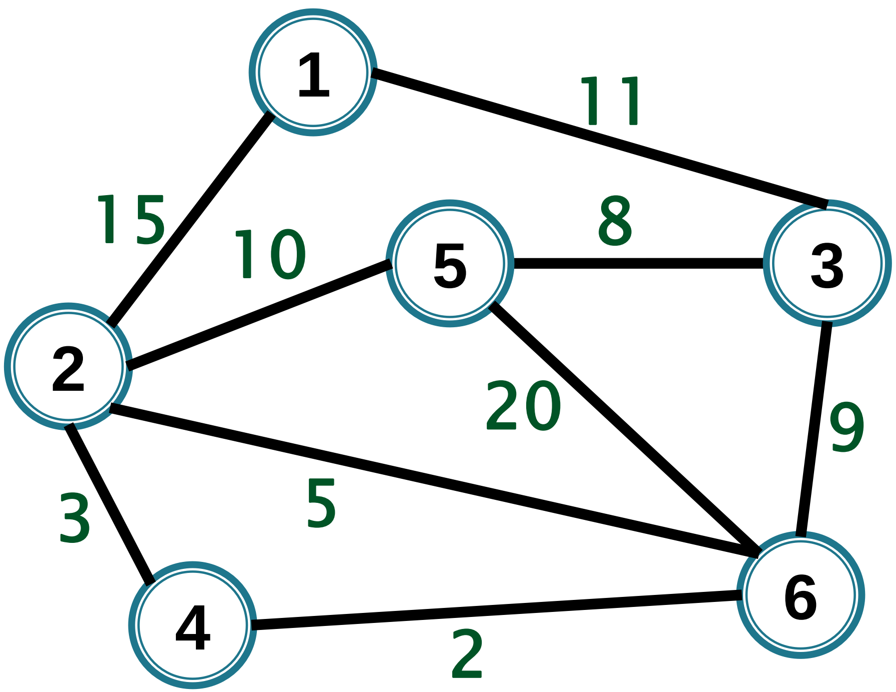
(2,5)

(1,3)

(1,2)

(5,6)





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

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

(2,4)

(2,6)

(3,5)

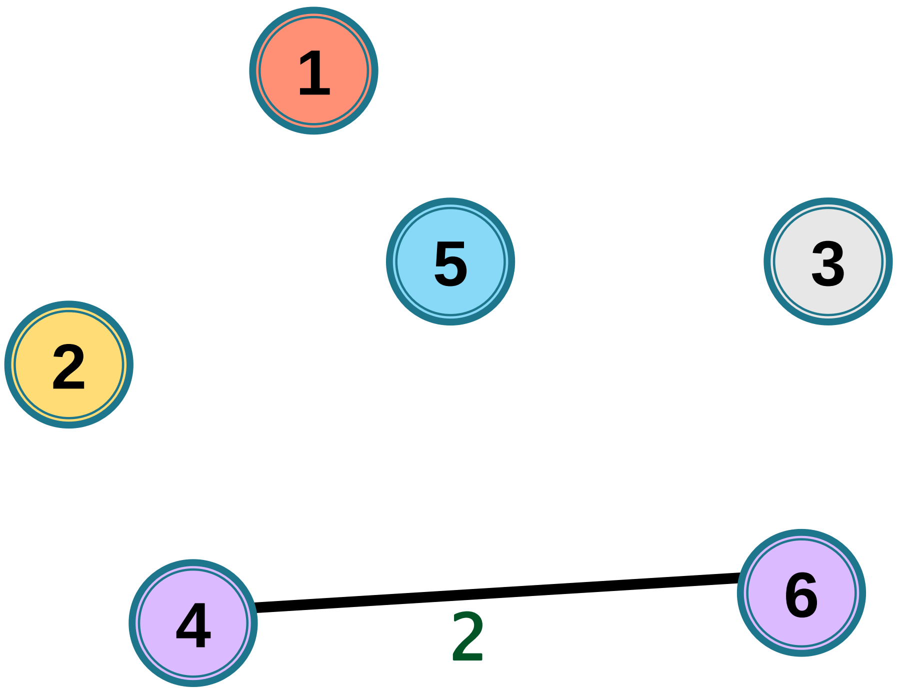
(3,6)

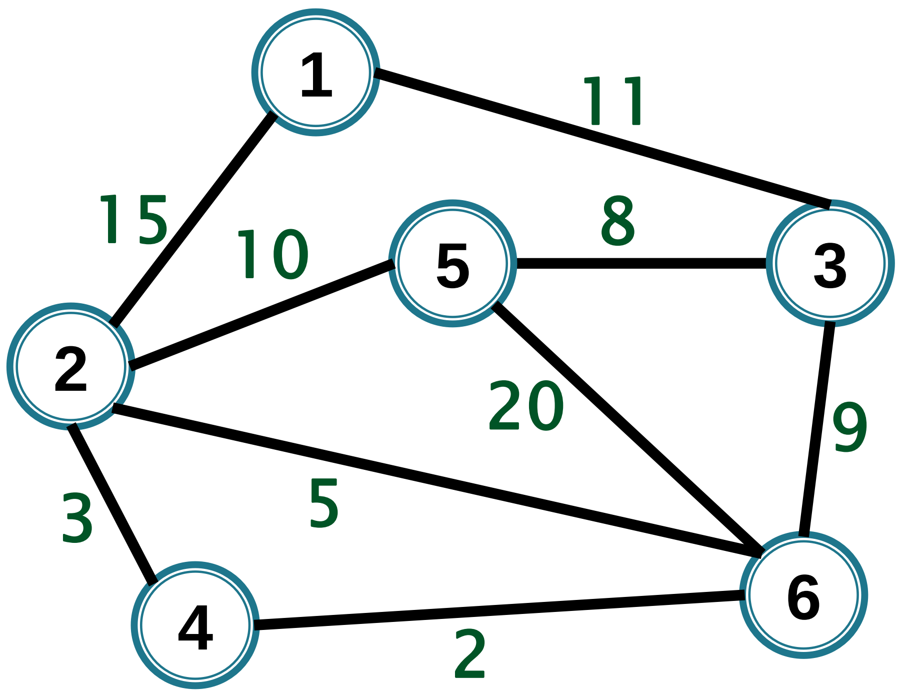
(2,5)

(1,3)

(1,2)

(5,6)





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

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

(2,4)

(2,6)

(3,5)

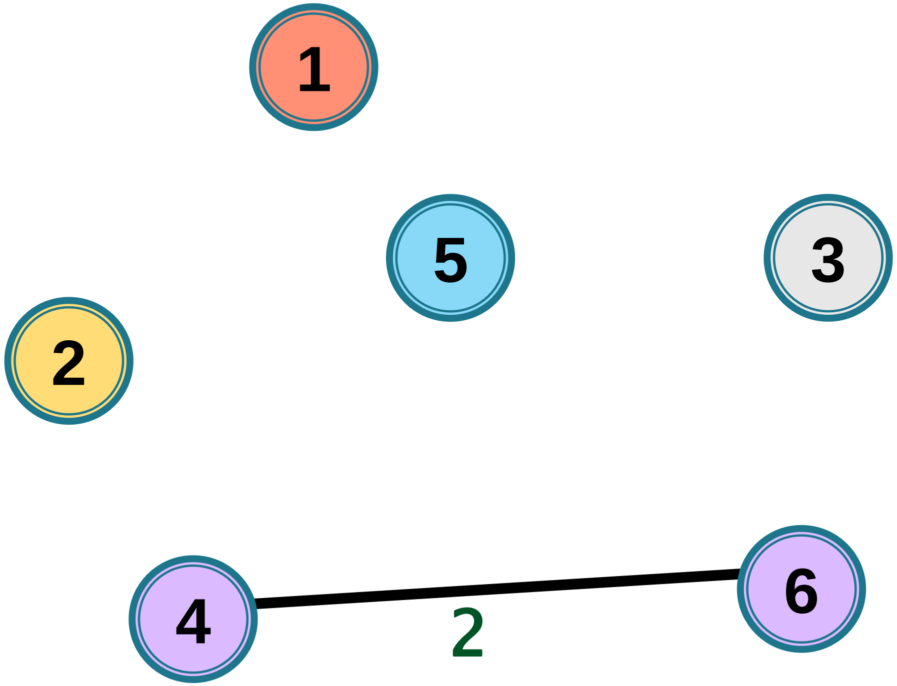
(3,6)

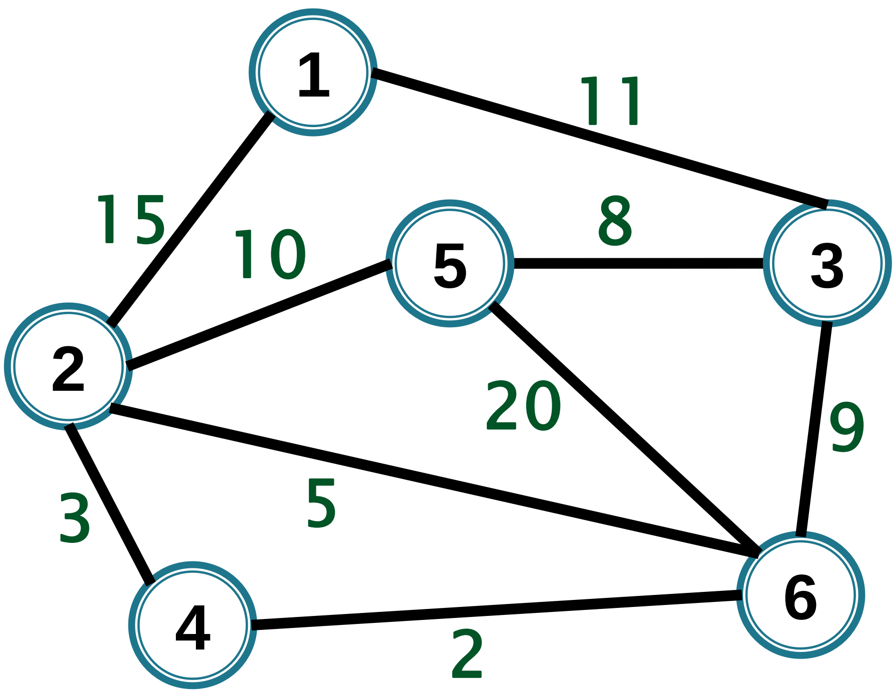
(2,5)

(1,3)

(1,2)

(5,6)





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

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

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

(2,6)

(3,5)

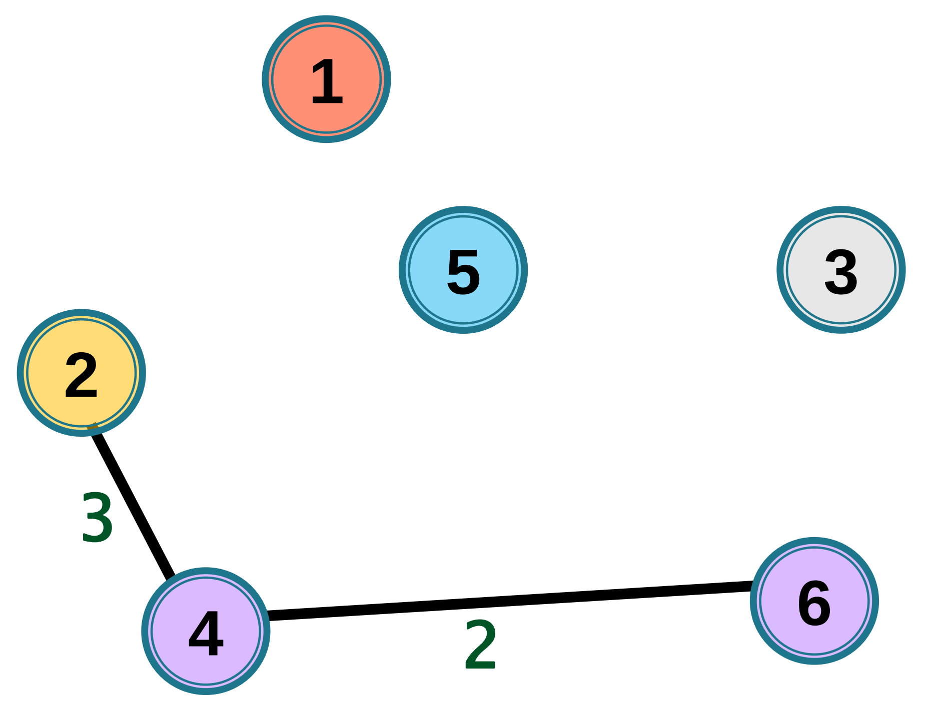
(3,6)

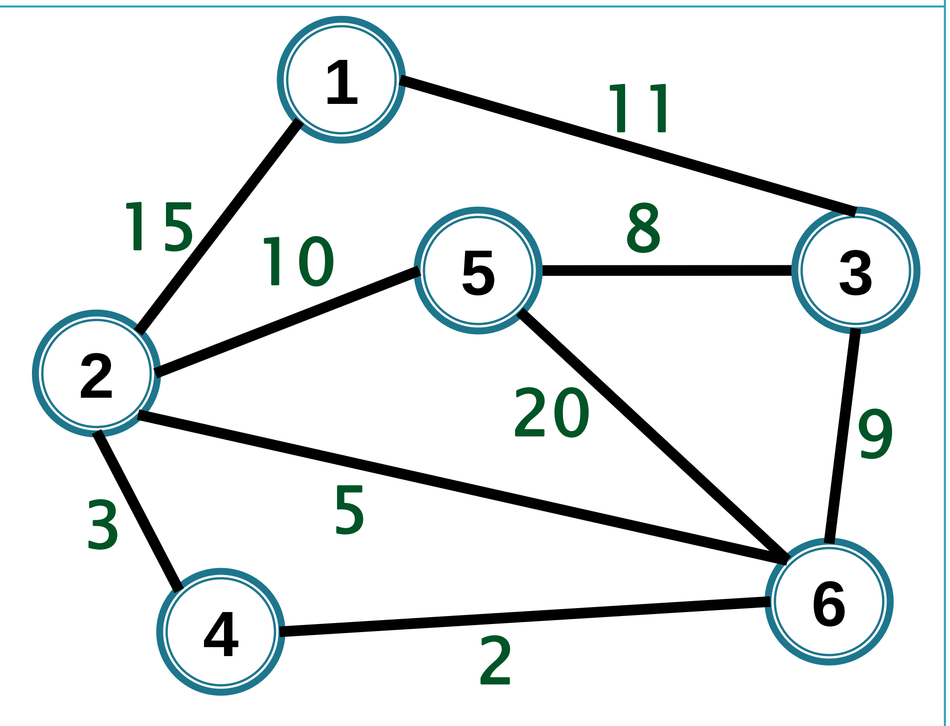
(2,5)

(1,3)

(1,2)

(5,6)





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

(4,6) r = [1,<u>2</u>,3,<u>4</u>,5,4]

(2,4)

(2,6)

(3,5)

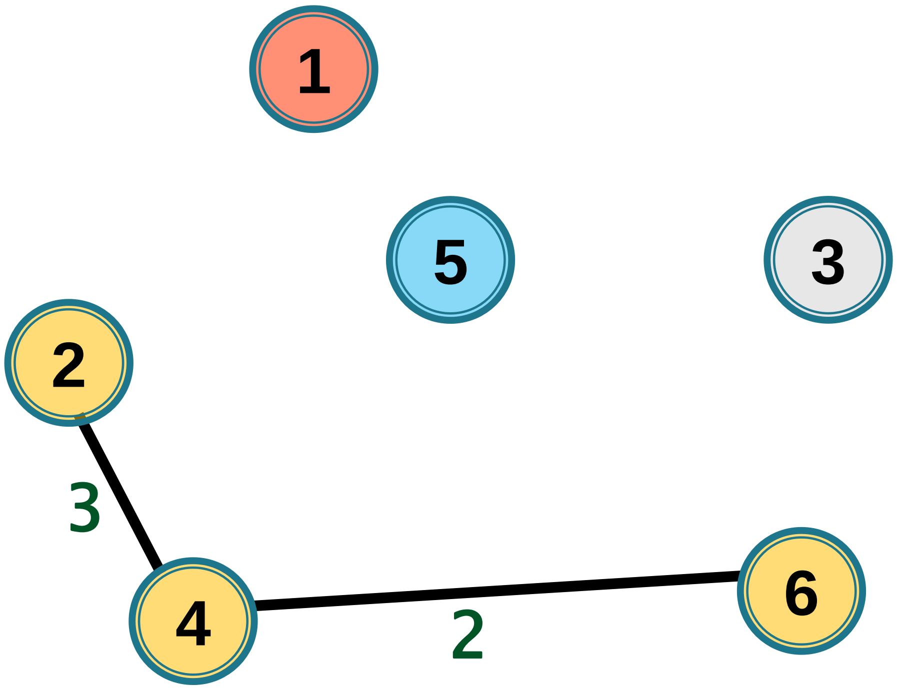
(3,6)

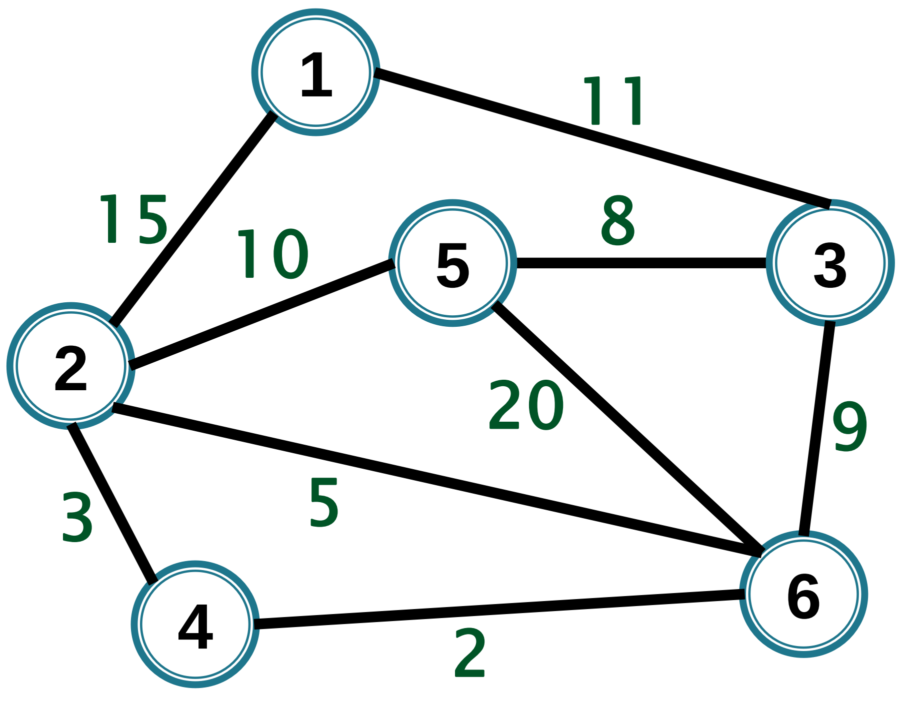
(2,5)

(1,3)

(1,2)

(5,6)





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

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

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

(2,6)

(3,5)

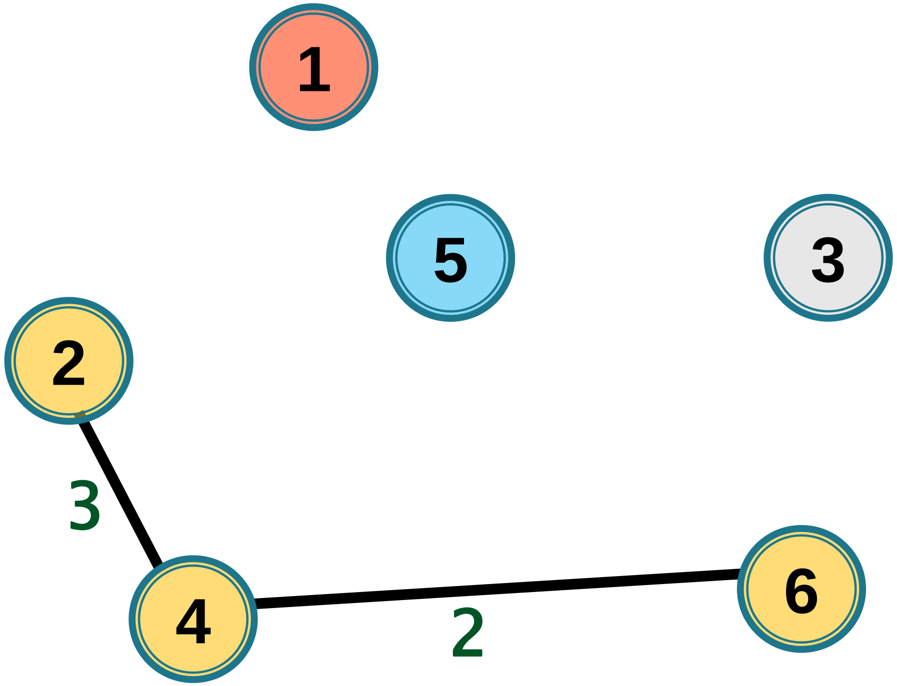
(3,6)

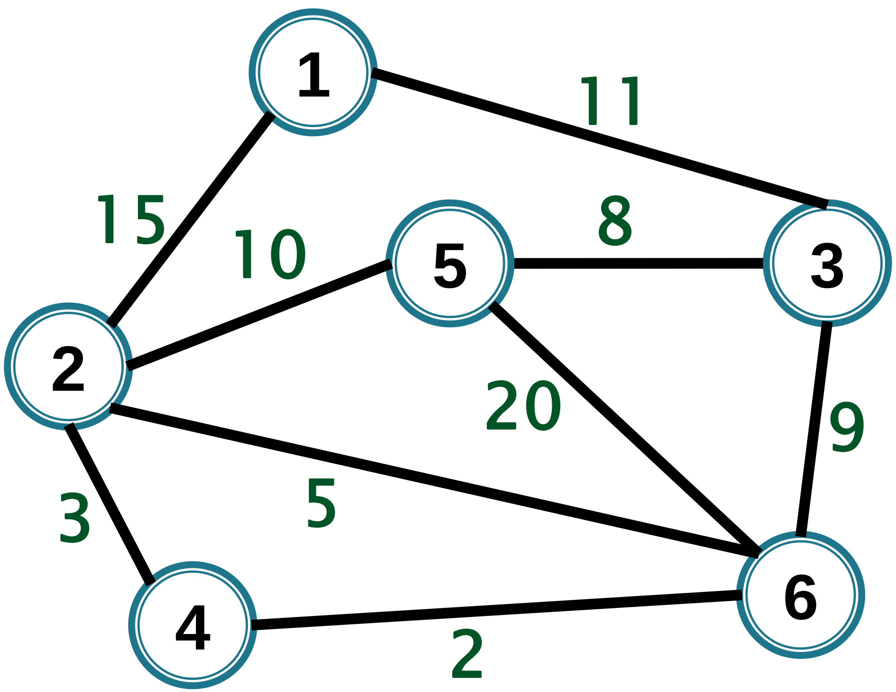
(2,5)

(1,3)

(1,2)

(5,6)





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

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

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

(2,6)

(3,5)

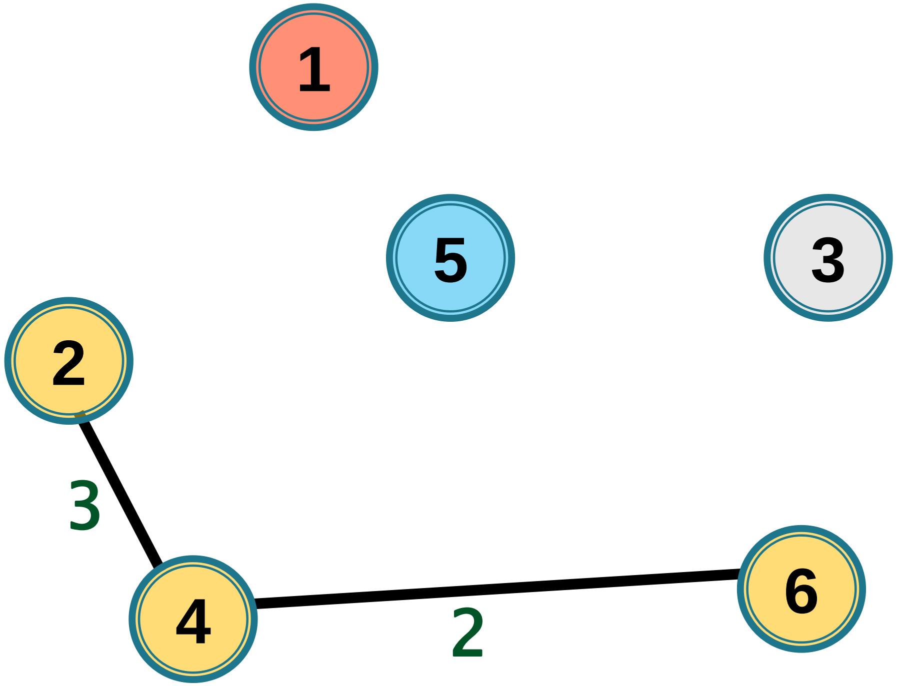
(3,6)

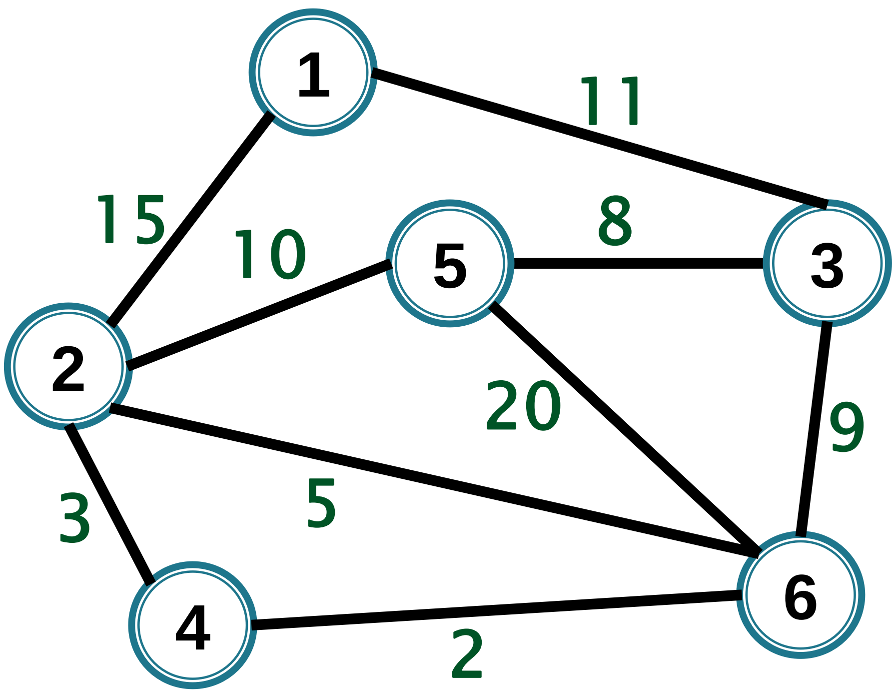
(2,5)

(1,3)

(1,2)

(5,6)





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

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

(2,4) r = [1,<u>2</u>,3,2,5,<u>2</u>]

(2,6) r(2) =r(6) -> NU

(3,5)

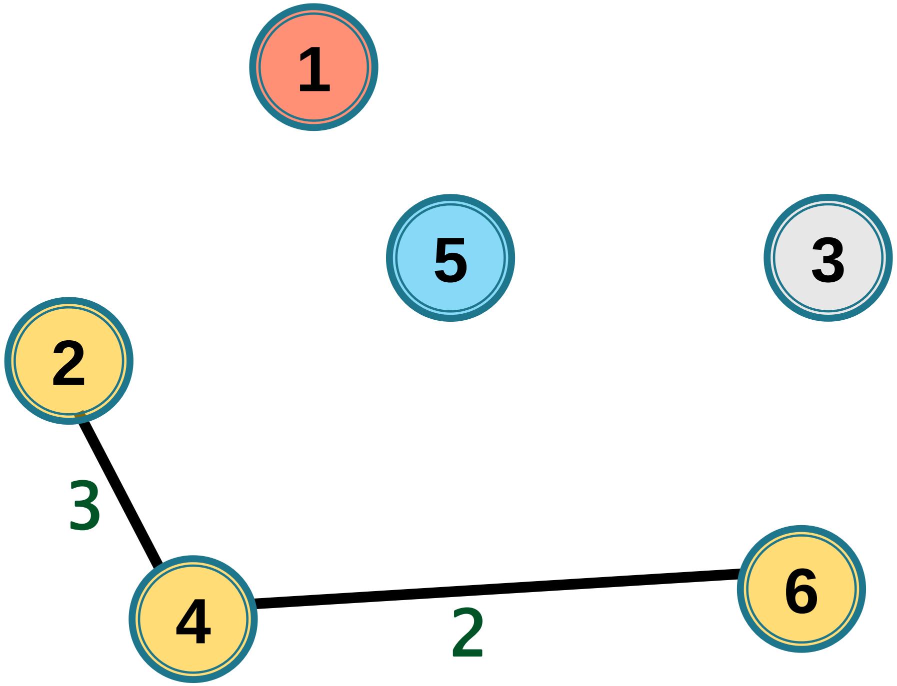
(3,6)

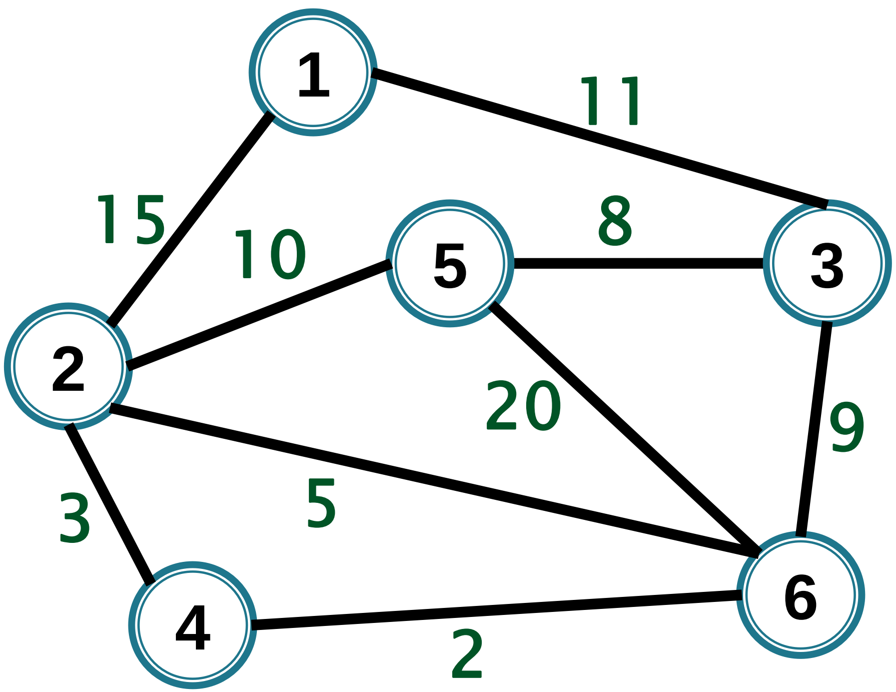
(2,5)

(1,3)

(1,2)

(5,6)





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

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

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

(2,6) r(2) =r(6) -> NU

(3,5)

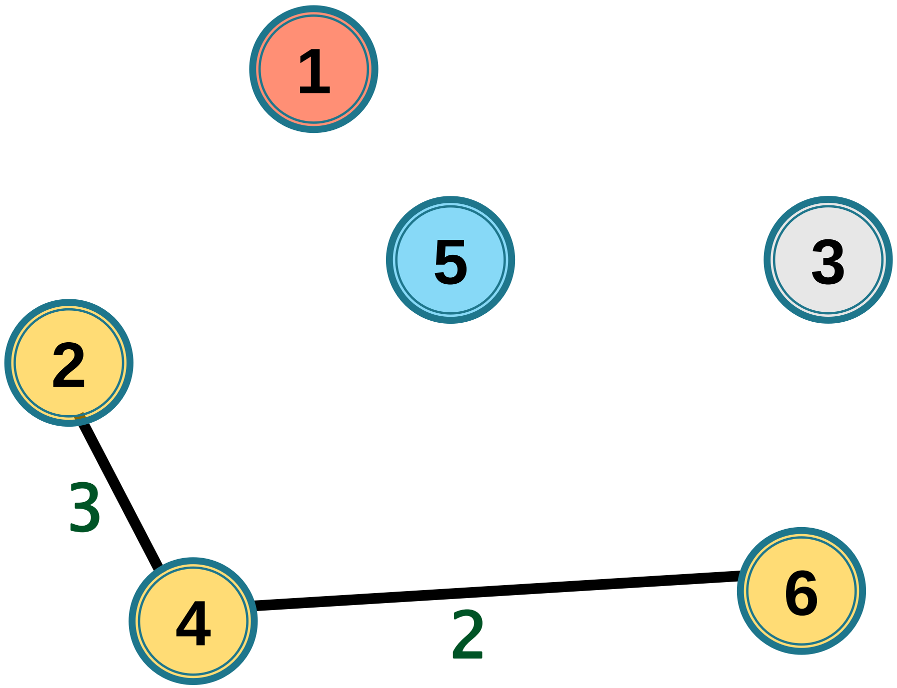
(3,6)

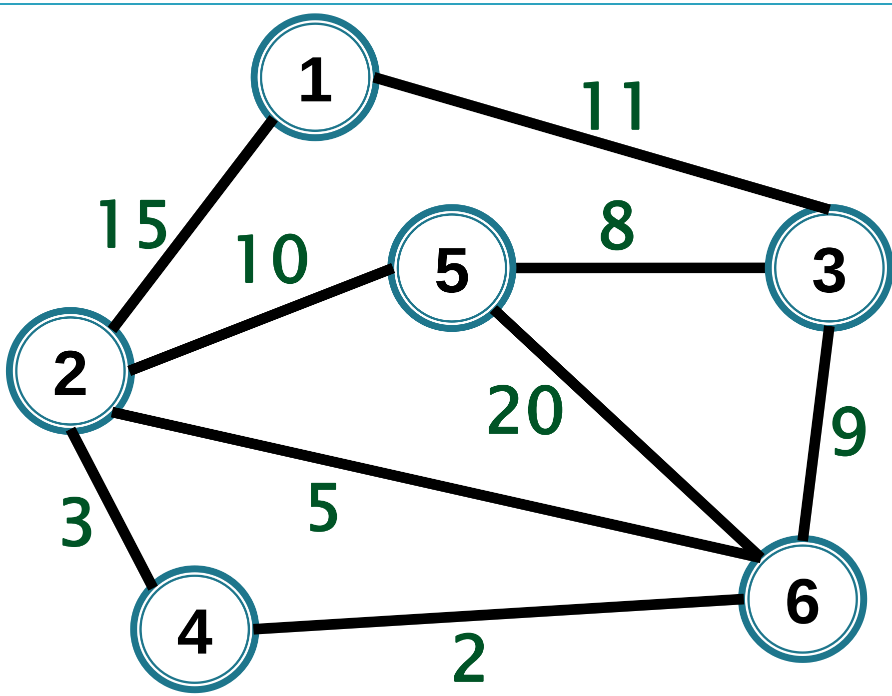
(2,5)

(1,3)

(1,2)

(5,6)





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

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

(2,4) r = [1,2,<u>3</u>,2,<u>5</u>,2]

(2,6) r(2) =r(6) -> NU

(3,5) r(3) ≠ r(5)

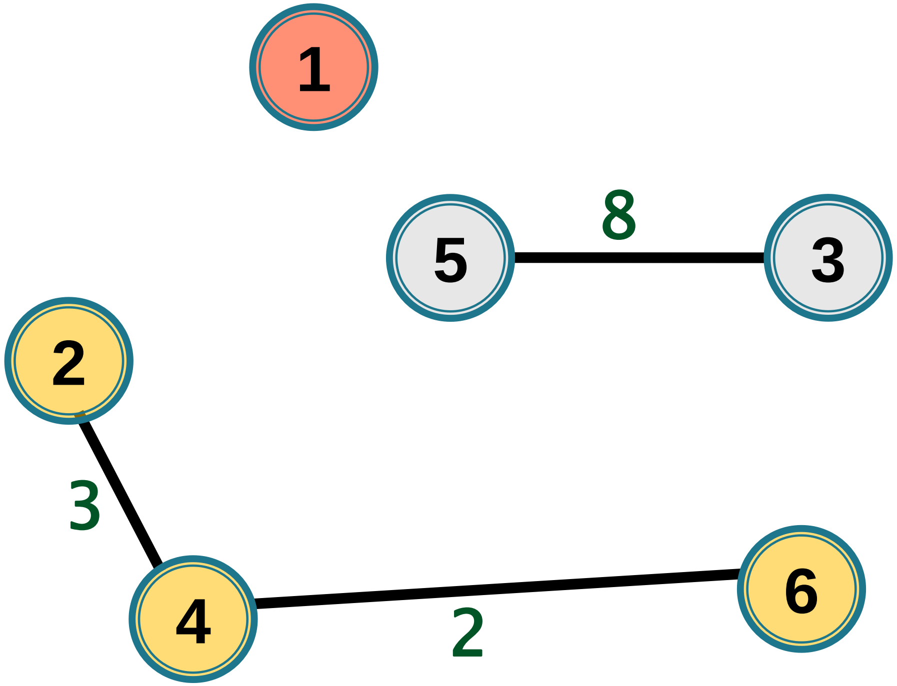
(3,6)

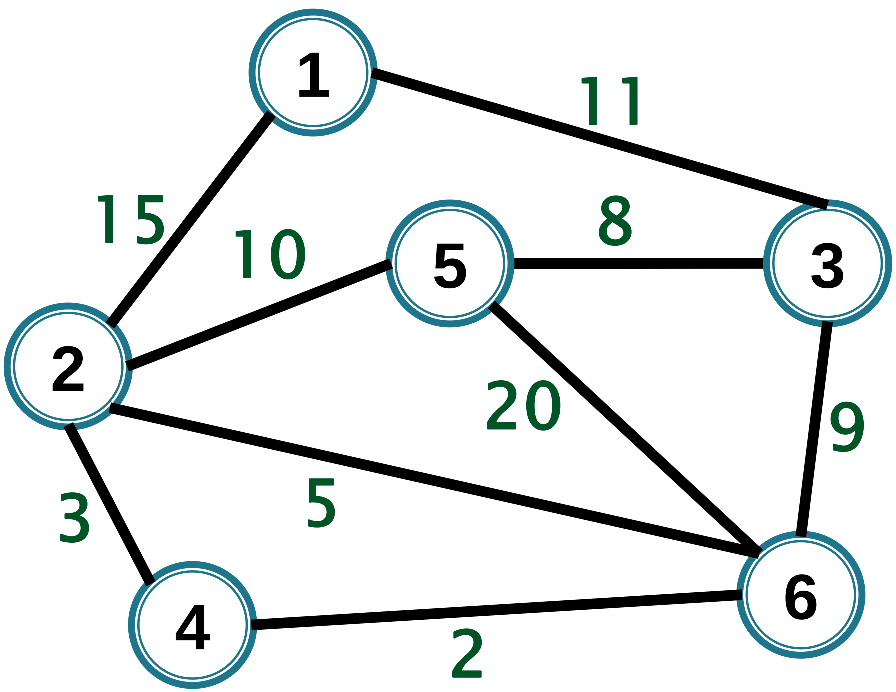
(2,5)

(1,3)

(1,2)

(5,6)





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

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

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

(2,6) r(2) =r(6) -> NU

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

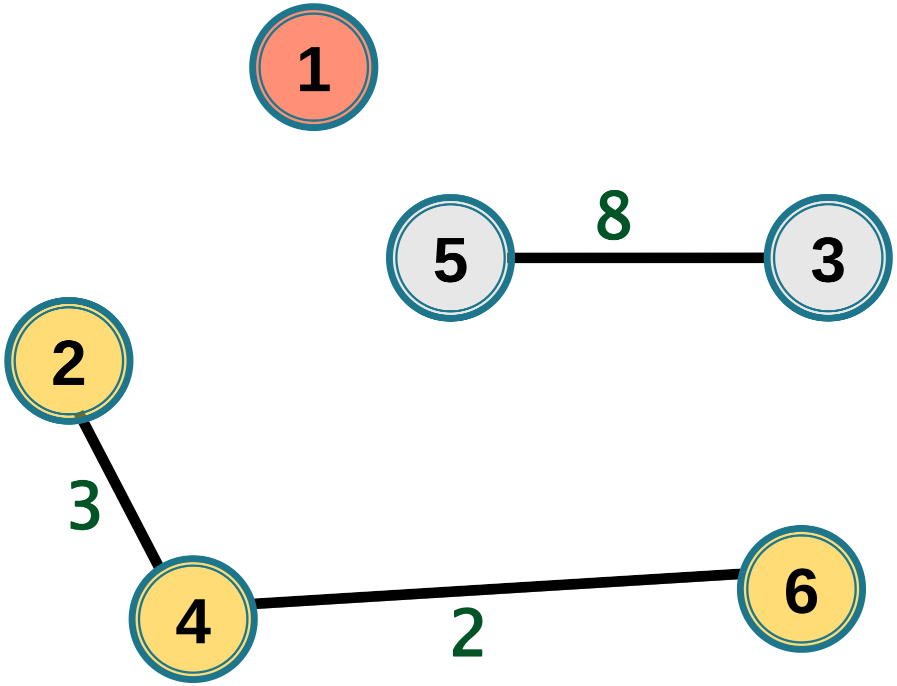
(3,6)

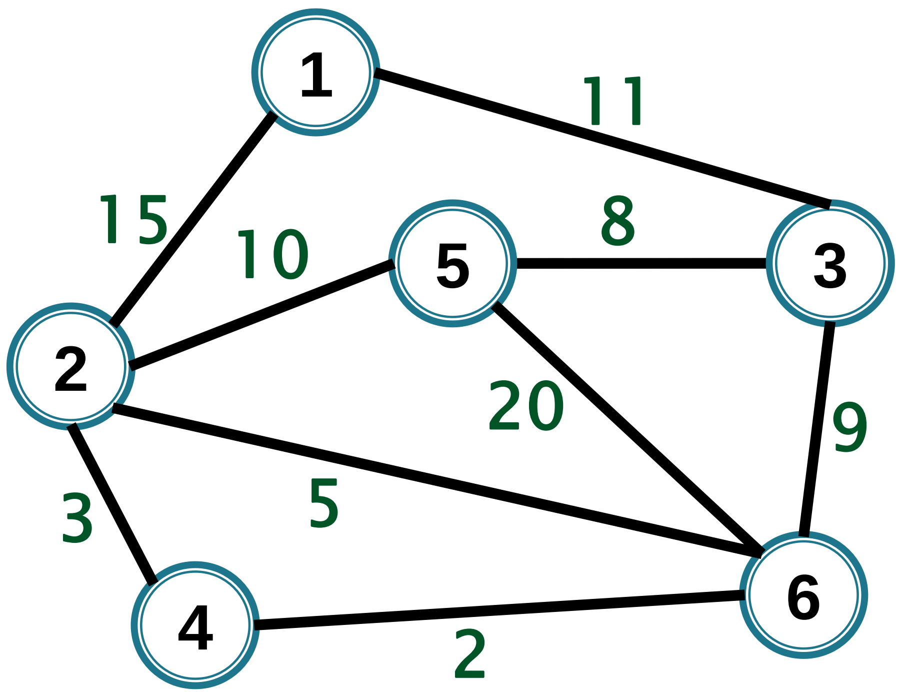
(2,5)

(1,3)

(1,2)

(5,6)





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

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

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

(2,6) r(2) =r(6) -> NU

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

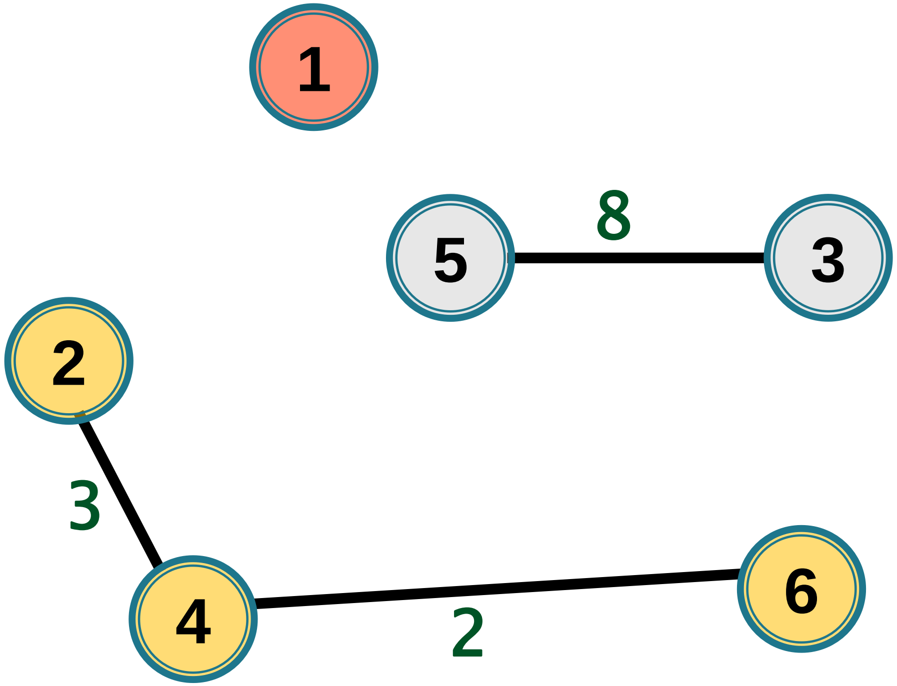
(3,6)

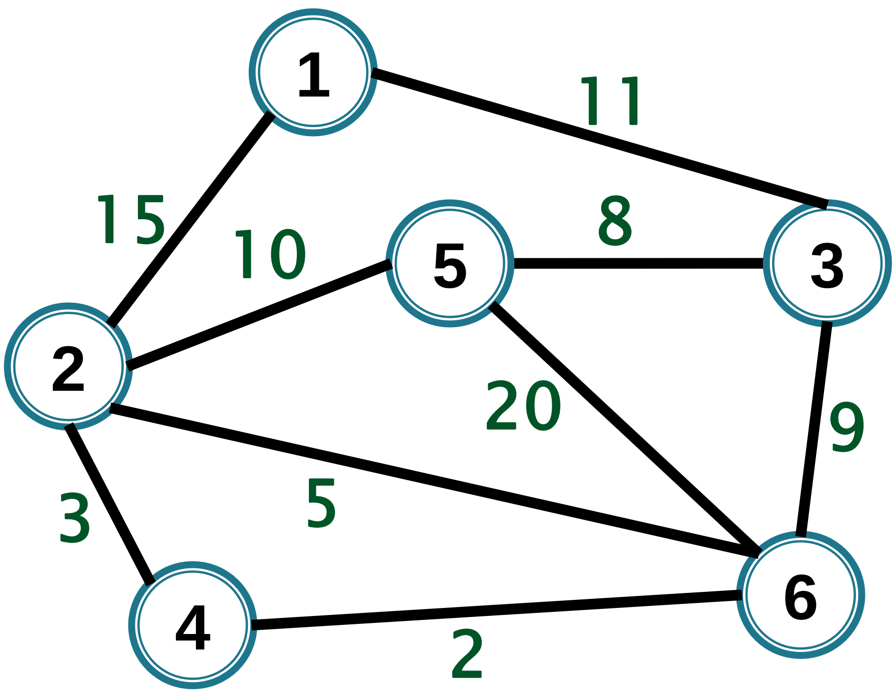
(2,5)

(1,3)

(1,2)

(5,6)





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

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

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

(2,6) r(2) =r(6) -> NU

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

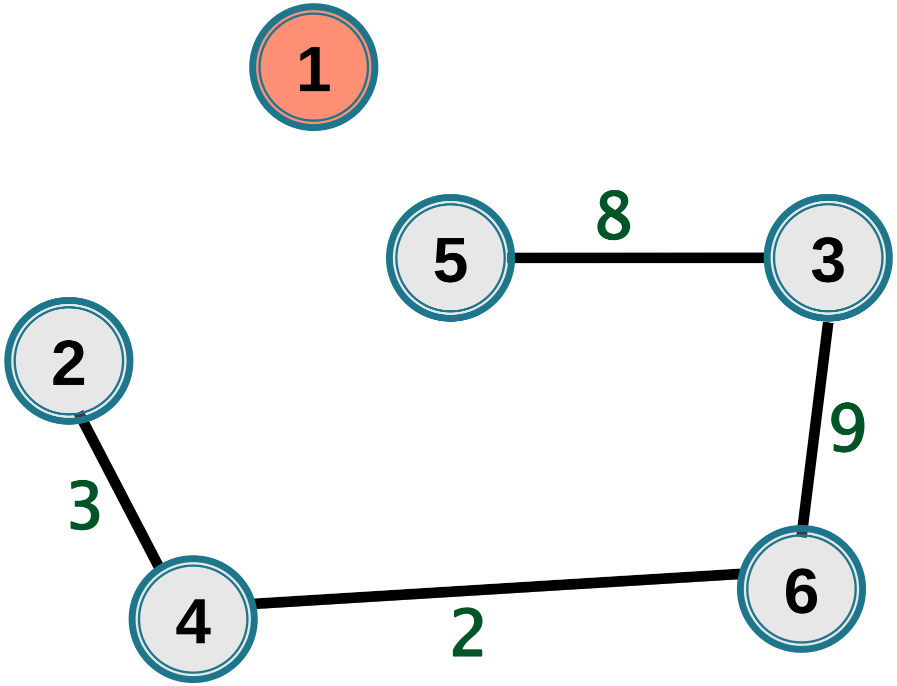
(3,6) $r(3) \neq r(6)$

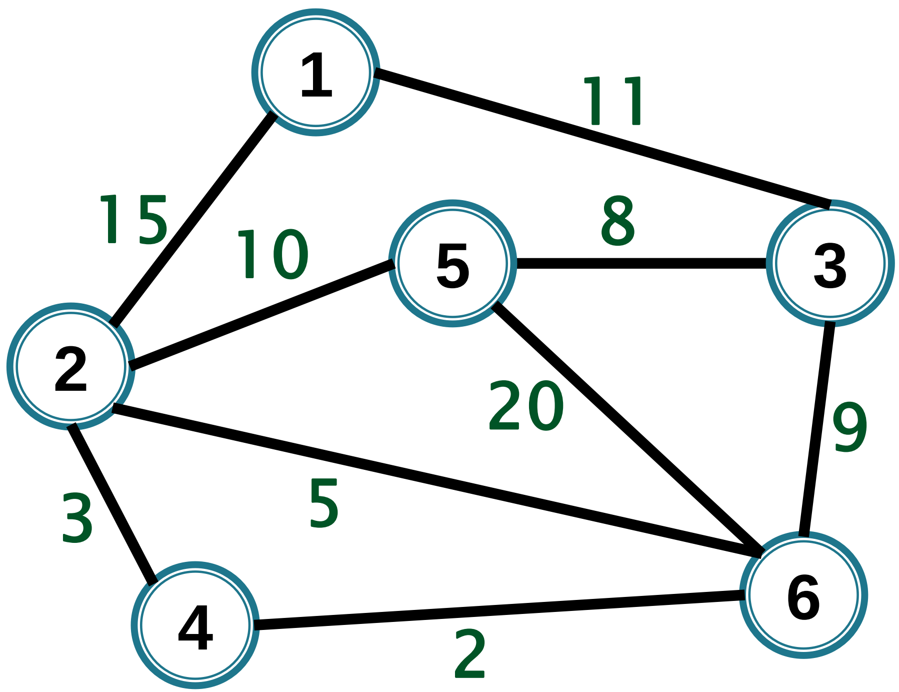
(2,5)

(1,3)

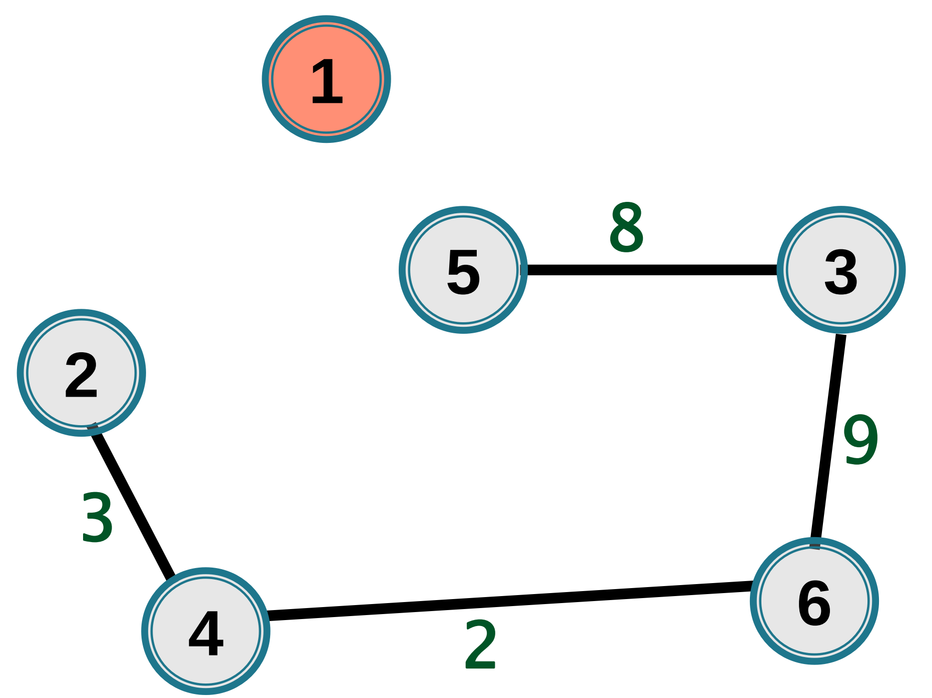
(1,2)

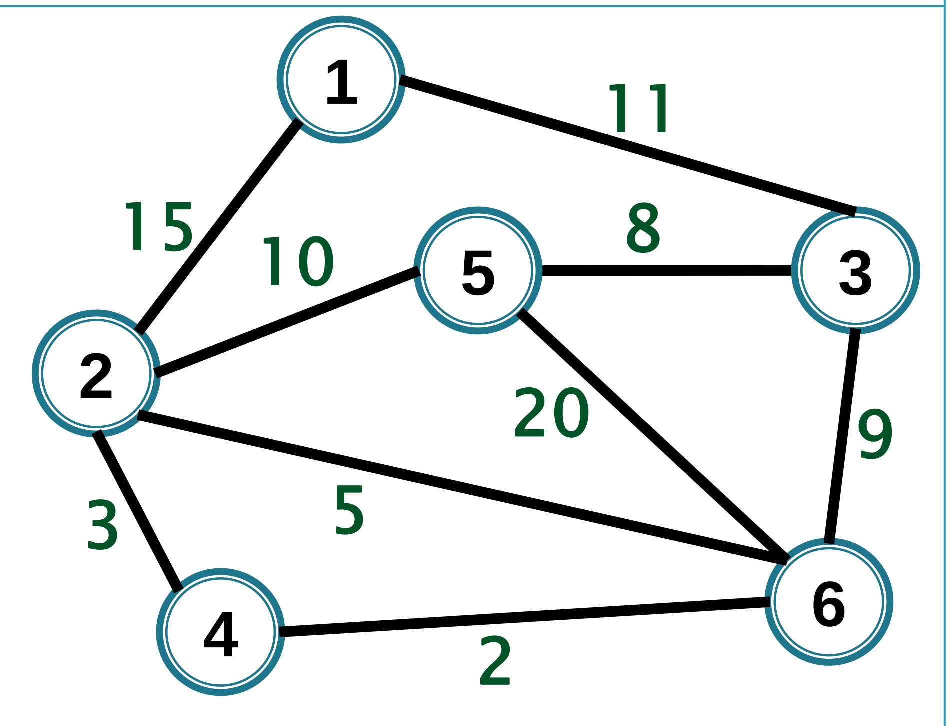
(5,6)





	r = [1,2,3,4,5,6]
(4,6)	r = [1,2,3,4,5,4]
(2,4)	r = [1,2,3,2,5,2]
(2,6)	r(2) =r(6) -> NU
(3,5)	r = [1,2,<u>3</u>,2,3,<u>2</u>]
(3,6)	r = [1,3,3,3,3,3]
(2,5)	
(1,3)	
(1,2)	
(5,6)	





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

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

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

(2,6) r(2) =r(6) -> NU

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

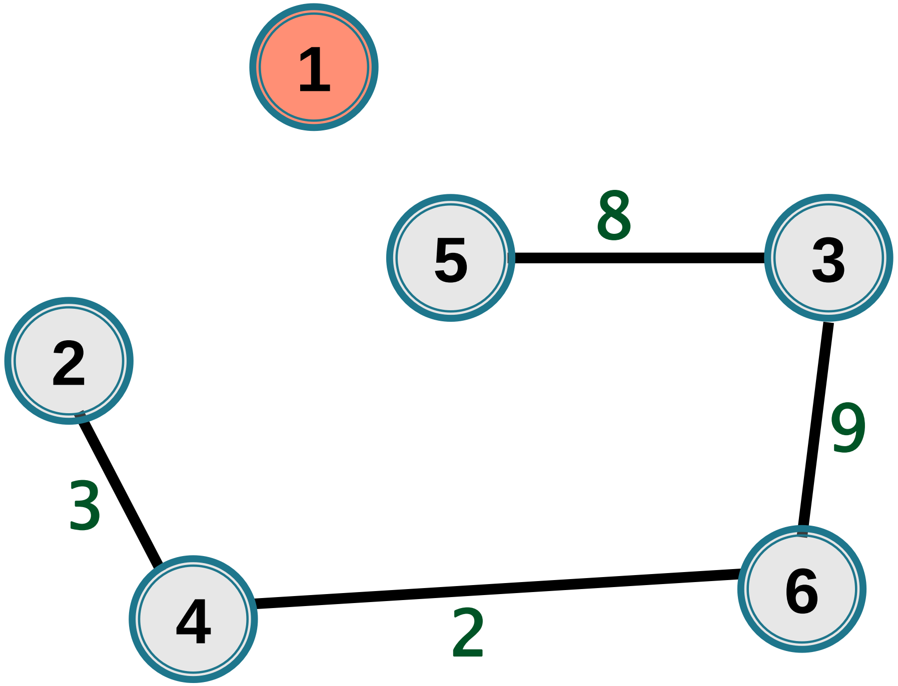
(3,6) r = [1,3,3,3,3,3]

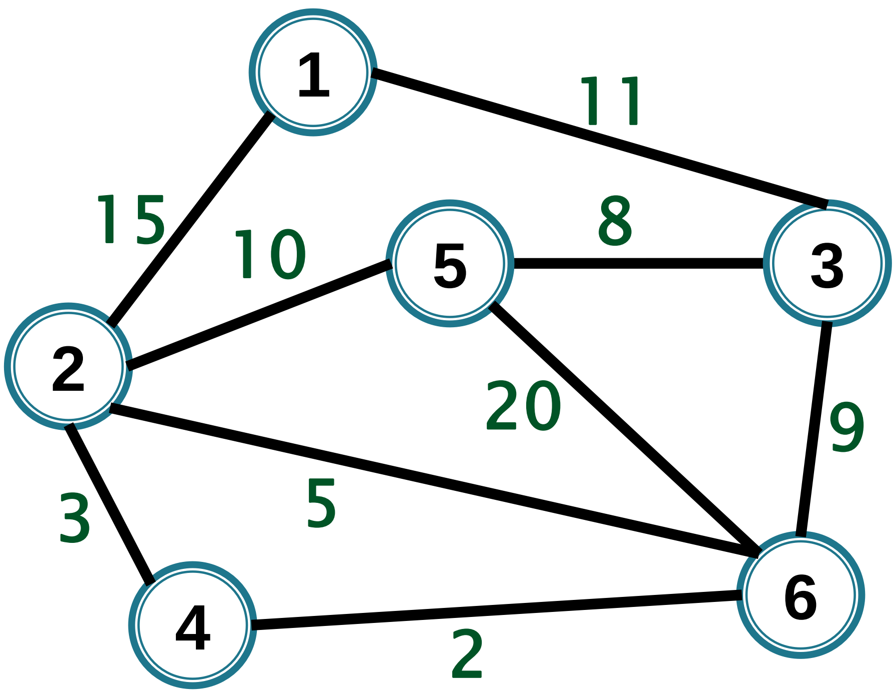
(2,5)

(1,3)

(1,2)

(5,6)





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

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

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

(2,6) r(2) =r(6) -> NU

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

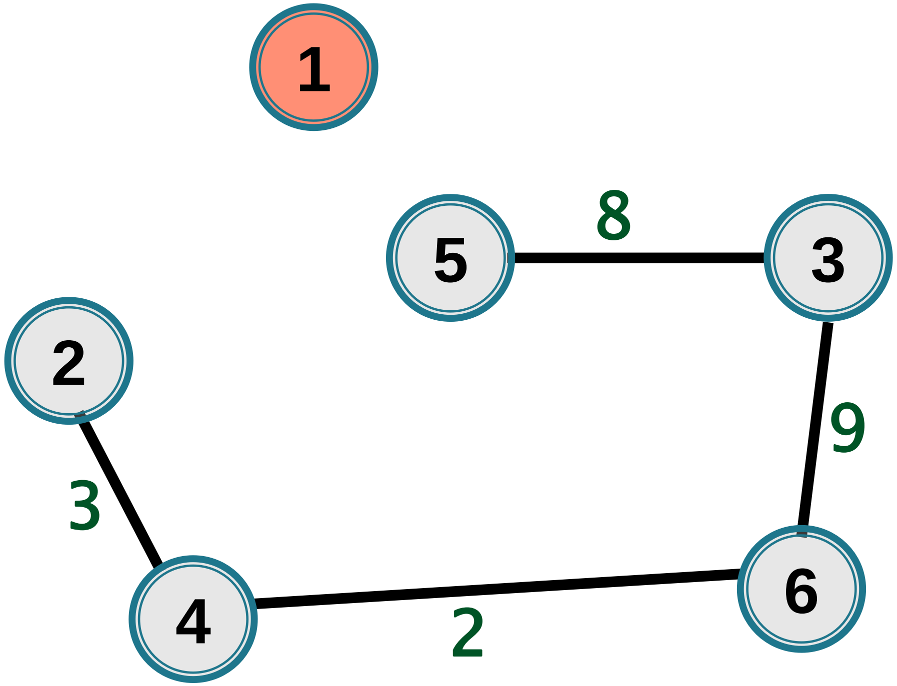
(3,6) r = [1,3,3,3,3,3]

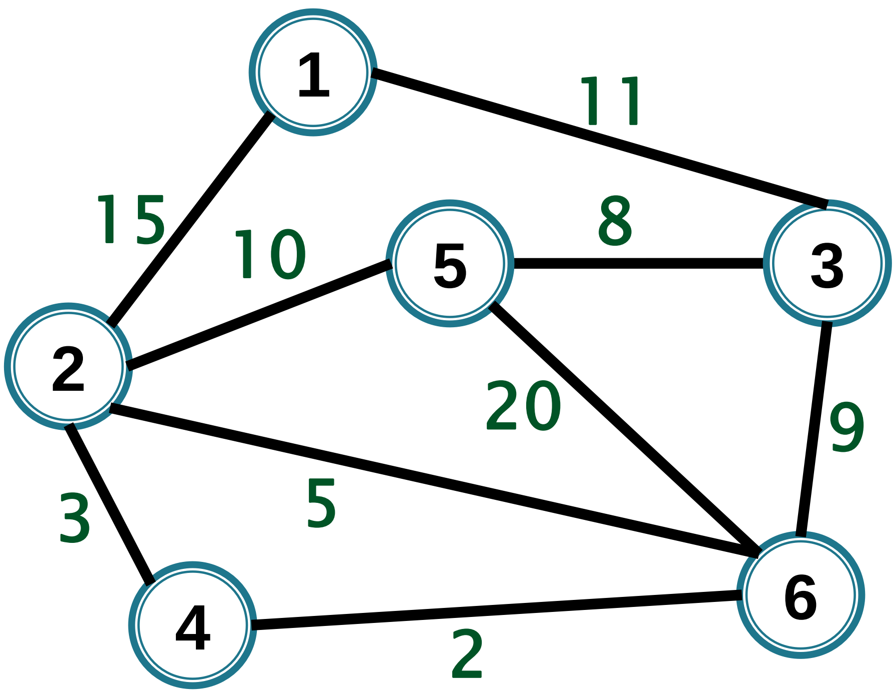
(2,5) r(2) =r(5) -> NU

(1,3)

(1,2)

(5,6)





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

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

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

(2,6) r(2) =r(6) -> NU

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

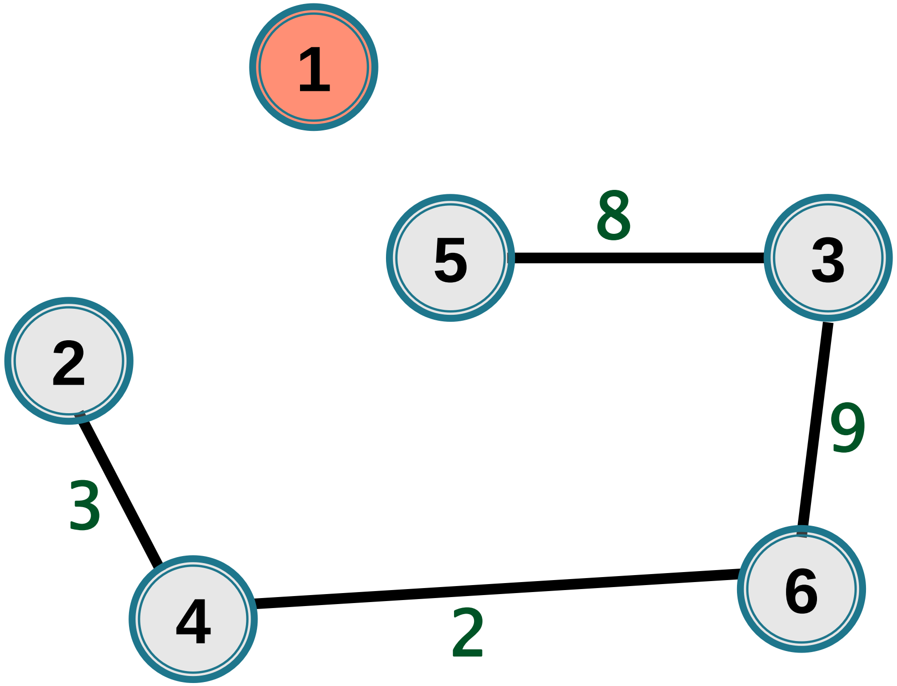
(3,6) r = [1,3,3,3,3,3]

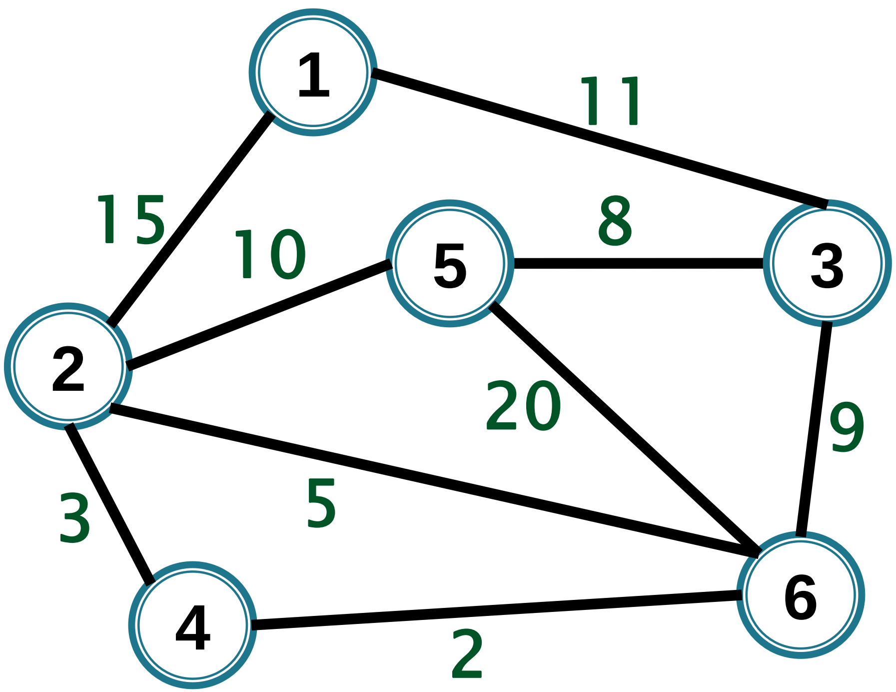
(2,5) r(2) =r(5) -> NU

(1,3)

(1,2)

(5,6)





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

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

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

(2,6) r(2) =r(6) -> NU

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

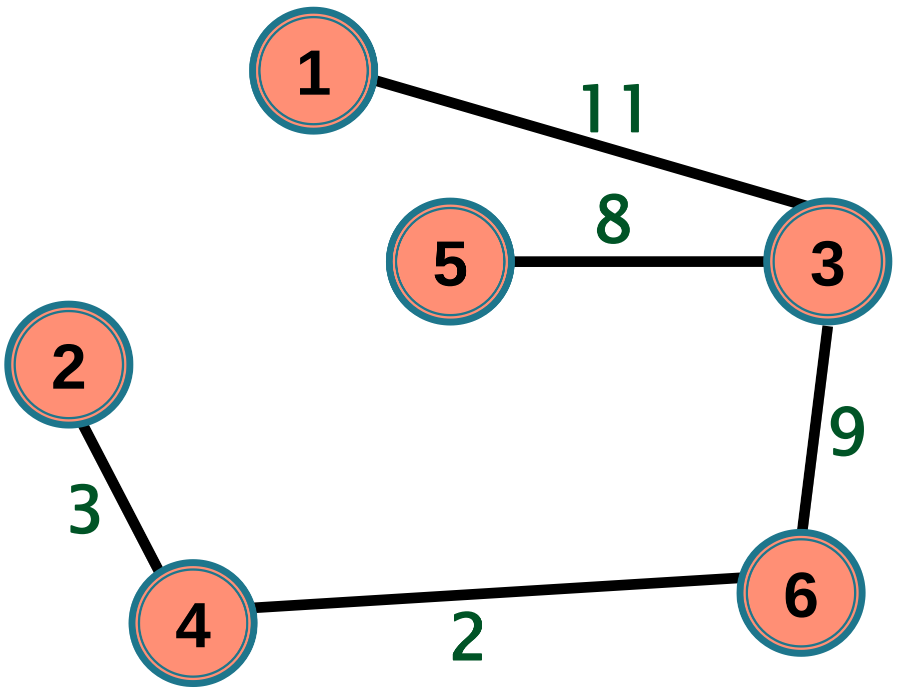
(3,6) r = [1,3,3,3,3,3]

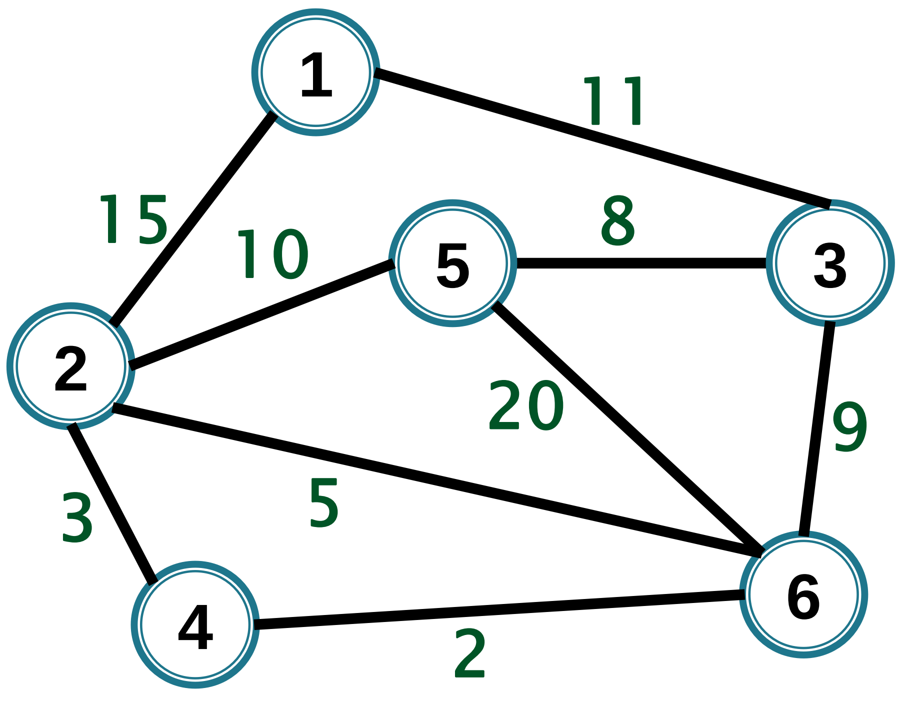
(2,5) r(2) =r(5) -> NU

(1,3) r(1) ≠ r(3)

(1,2)

(5,6)





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

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

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

(2,6) r(2) =r(6) -> NU

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

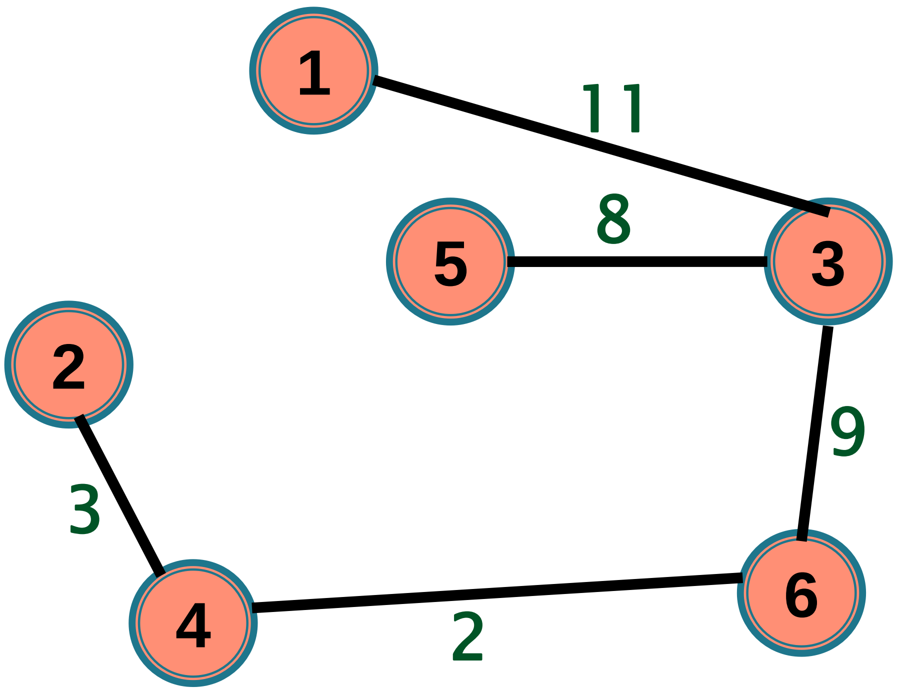
(3,6) r = [1,3,3,3,3,3]

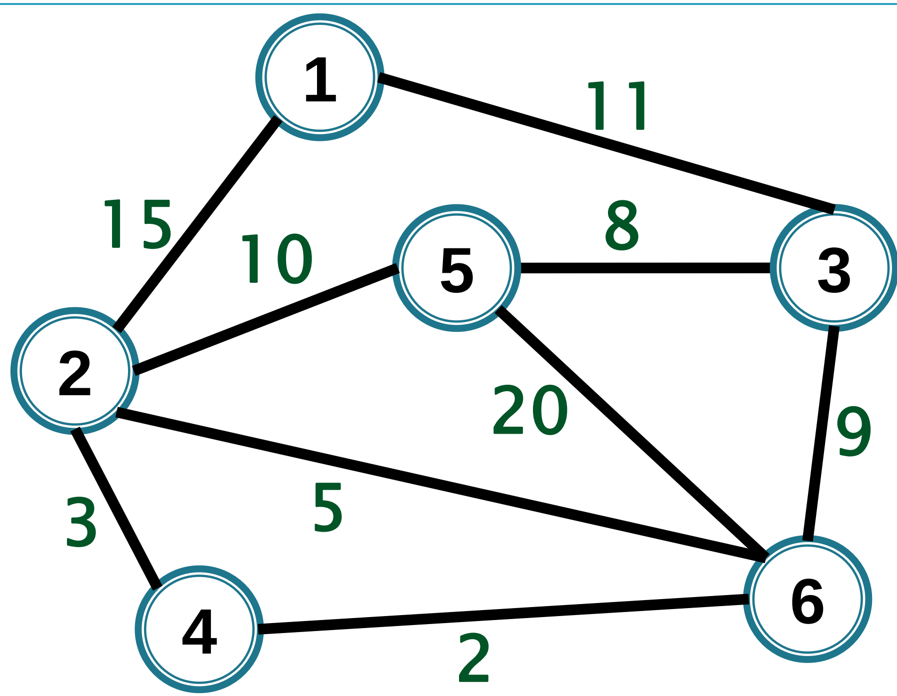
(2,5) r(2) =r(5) -> NU

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

(1,2)

(5,6)





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

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

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

(2,6) r(2) =r(6) -> NU

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

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

(2,5) r(2) =r(5) -> NU

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

<u>STOP</u>

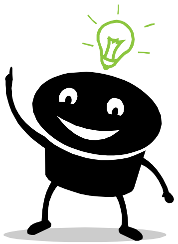
(1,2)

(5,6)

Kruskal

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

Kruskal

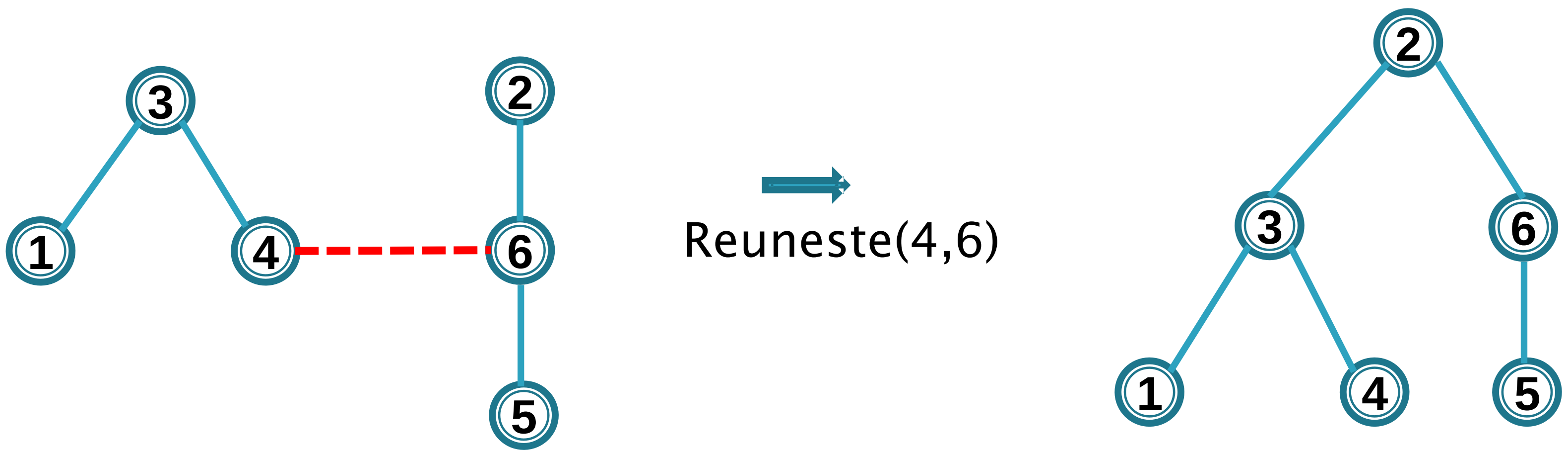


Varianta 2 – Structuri pentru mulțimi disjuncte Union/Find – arbori

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

Kruskal

- **Reuniunea se va face în funcţie de înălţimea arborilor (reuniune ponderată) ⇒ arbori de înălțime logaritmică**



- arborele cu înălţimea mai mică devine subarbore al rădăcinii celuilalt arbore

Kruskal

Complexitate – dacă folosim arbori

- **Sortare** -> O(m log m) = O(m log n)
- **n * Initializare** ->
- **2m * Reprez** ->
- **(n-1) * Reuneste** ->

Kruskal

Complexitate – dacă folosim arbori

- **Sortare** -> $O(m \log m) = O(m \log n)$
- **n * Initializare** -> $O(n)$
- **2m * Reprez** -> $O(m \log n)$
- **(n-1) * Reuneste** -> $O(n \log n)$

$O(m \log n)$

(cu compresie de cale – Reprez+Reuneste $O(n+m)$)

Kruskal

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

Aplicații – Clustering

Gruparea unor obiecte în k clase cât mai *bine separate* (k dat)

- obiecte din clase diferite să fie cât mai diferite

Aplicații – Clustering

Cuvinte – distanța de editare

care

martian

este

apa

ana

sinonim

minim

partial

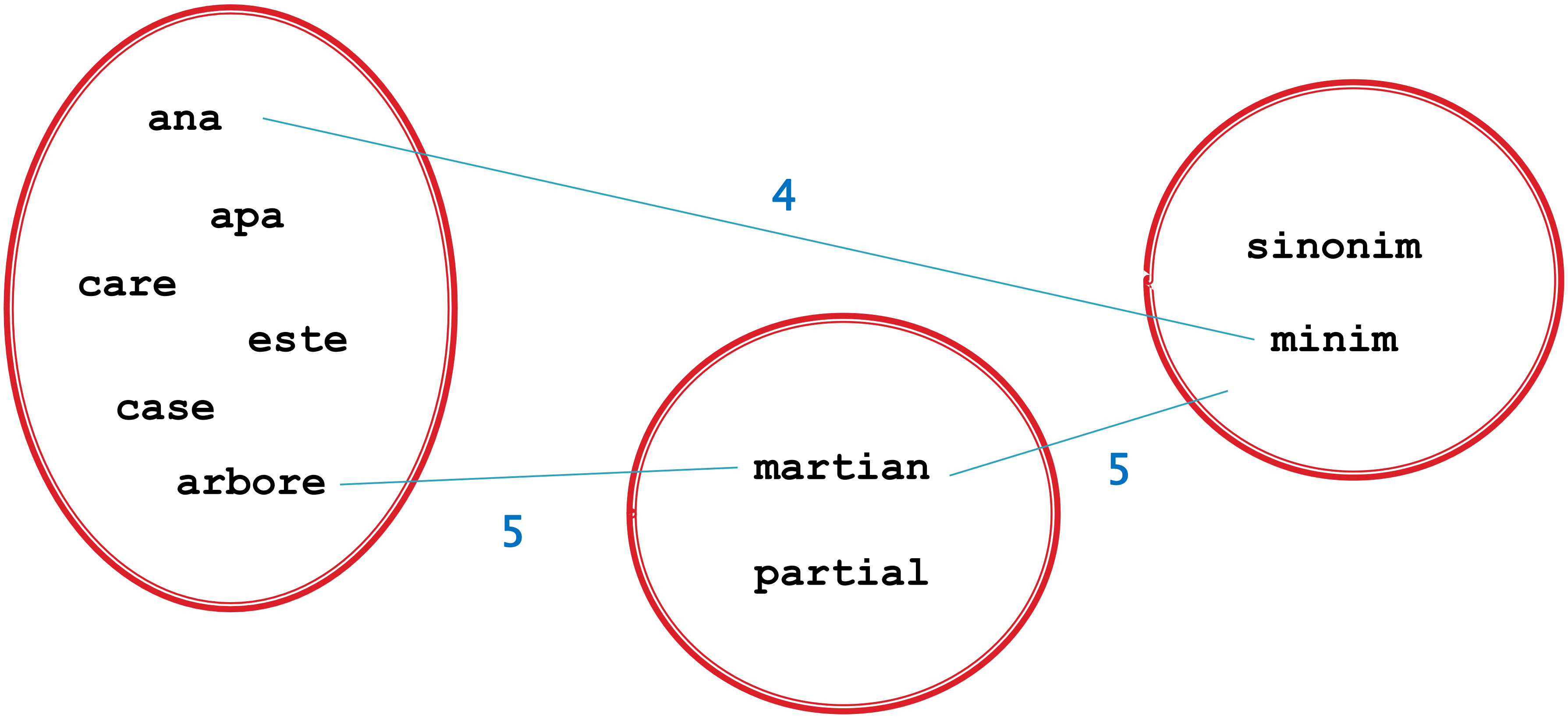
arbore

case

K = 3 clustere

Soluţie posibilă

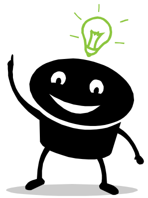
Cuvinte – distanța de editare



Aplicații - Clustering

Formal

Aplicații – Clustering



Idee

Aplicații – Clustering

Idee

- Iniţial fiecare obiect (cuvânt) formează o clasă
- La un pas determinăm cele mai apropiate două obiecte aflate în clase diferite (cu distanţa cea mai mică între ele) şi unim clasele lor

Aplicații – Clustering

Idee

- Iniţial fiecare obiect (cuvânt) formează o clasă
- La un pas determinăm cele mai apropiate două obiecte aflate în clase diferite (cu distanţa cea mai mică între ele) şi unim clasele lor
- n – k paşi

Aplicații – Clustering

Cuvinte – distanța de editare

care

martian

este

apa

ana

sinonim

minim

partial

arbore

case

K = 3 clustere

Aplicații – Clustering

care 1 case

martian

este

apa

ana

sinonim

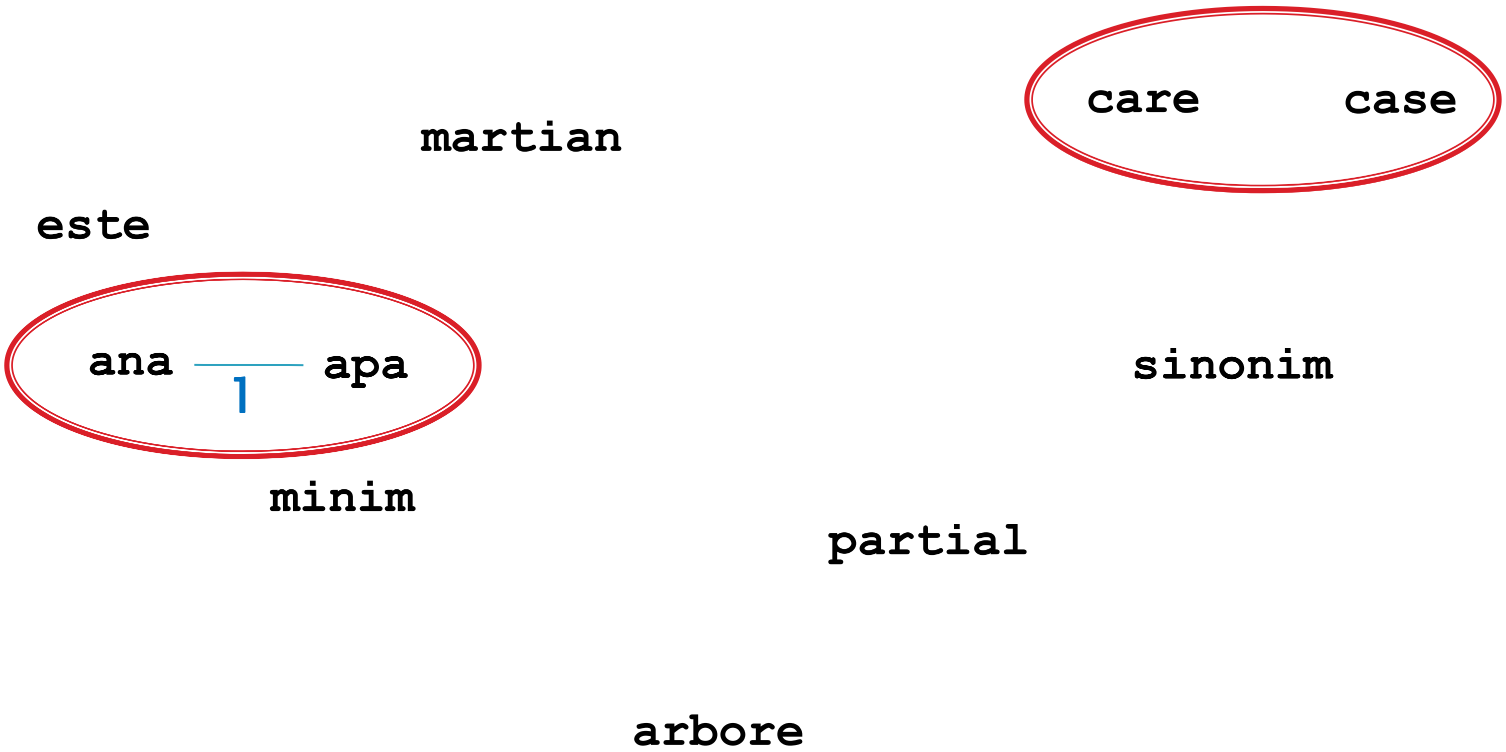
minim

partial

arbore

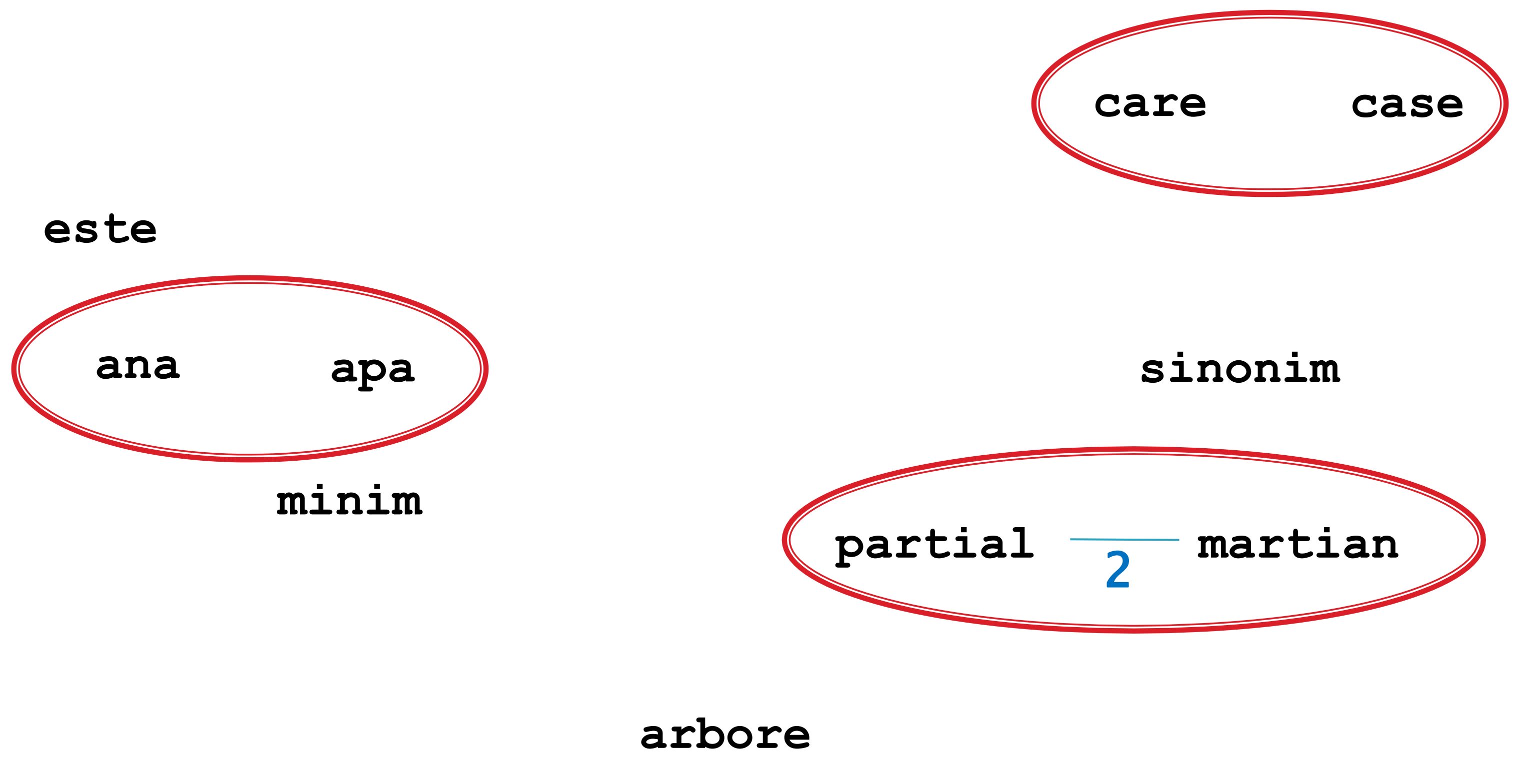
K = 3 clustere

Aplicații – Clustering



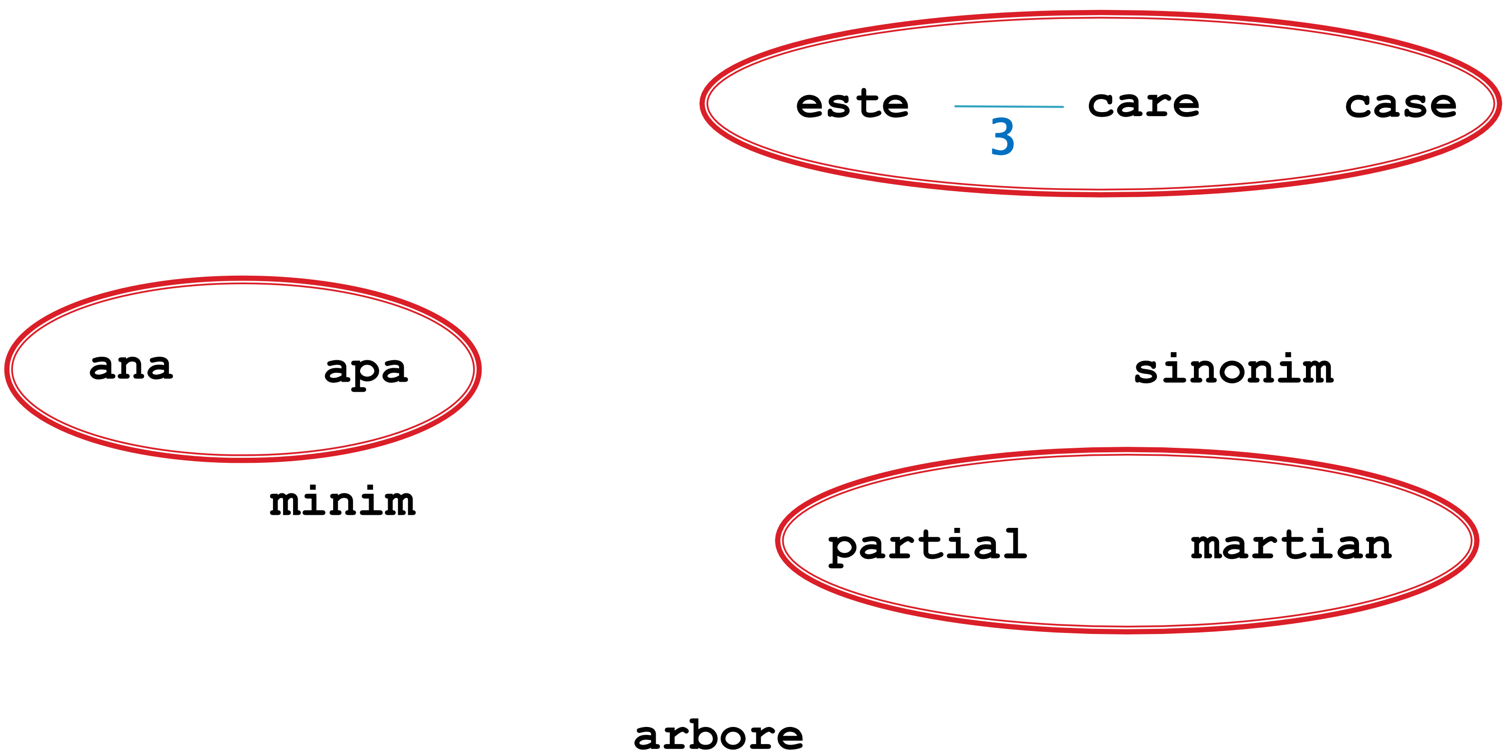
K = 3 clustere

Aplicații – Clustering



K = 3 clustere

Aplicații – Clustering



K = 3 clustere

Aplicații – Clustering

este care case

ana apa

sinonim

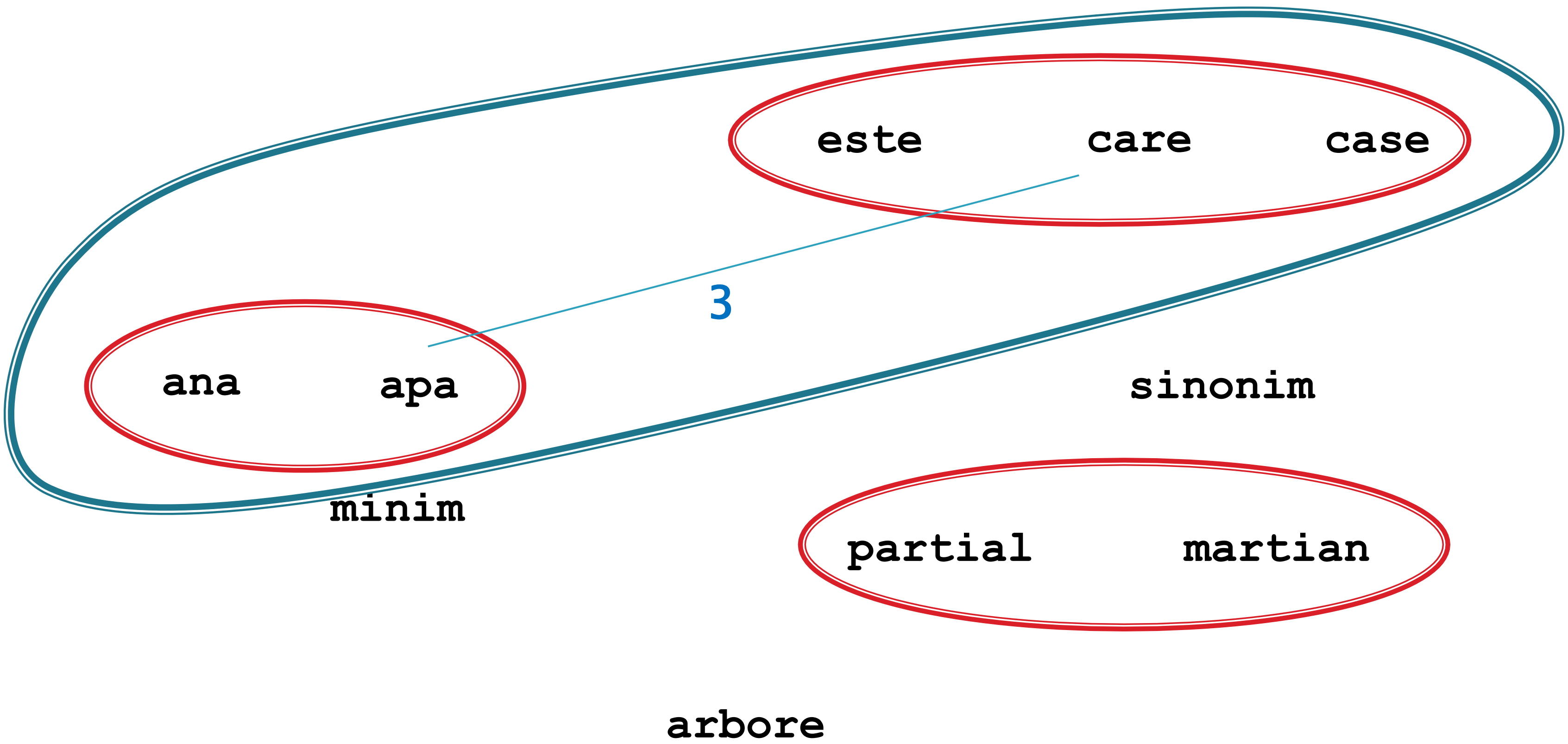
minim

partial martian

arbore

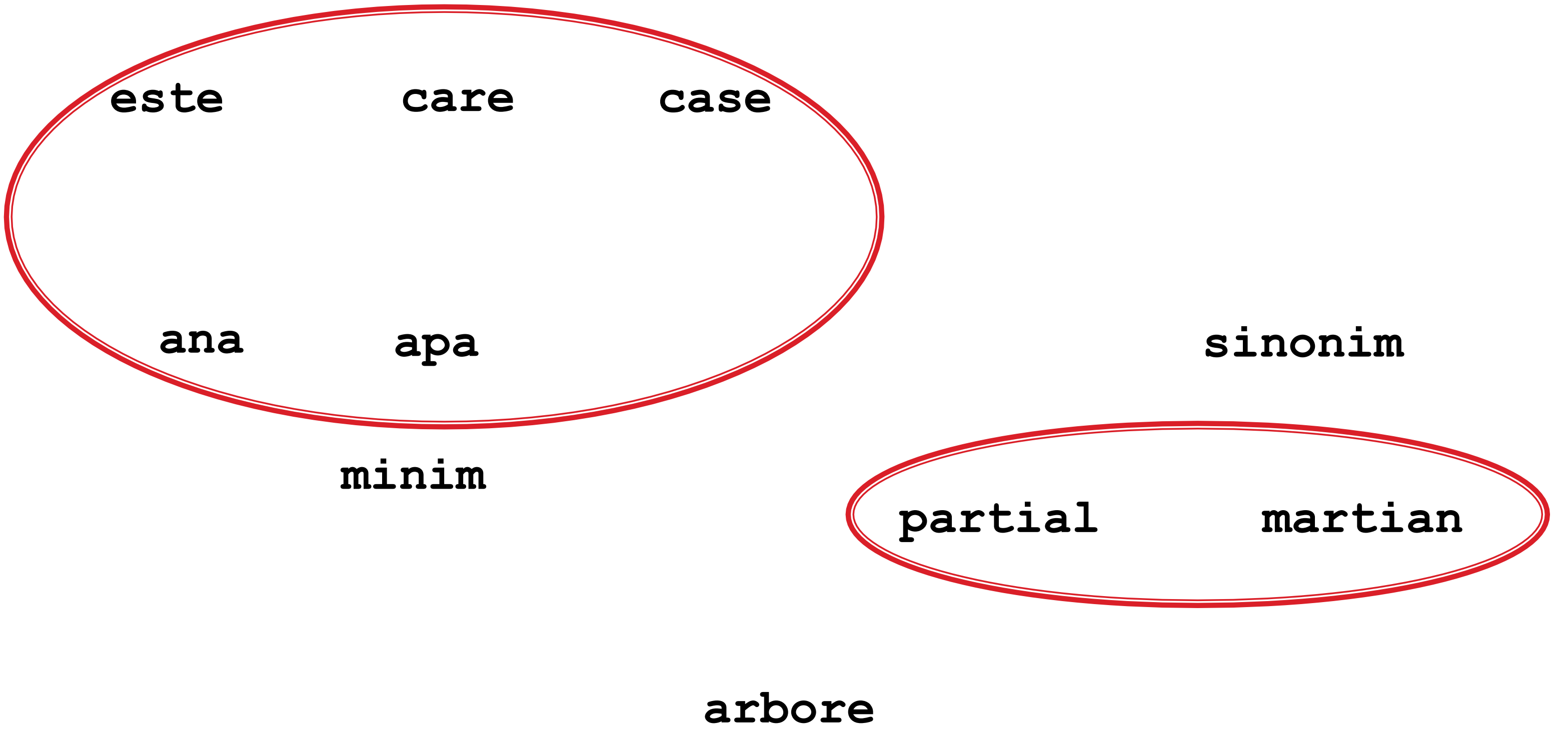
K = 3 clustere

Aplicații – Clustering



K = 3 clustere

Aplicații – Clustering



K = 3 clustere

Aplicații – Clustering

Cuvinte – distanța de editare

este care case

ana apa

minim

3

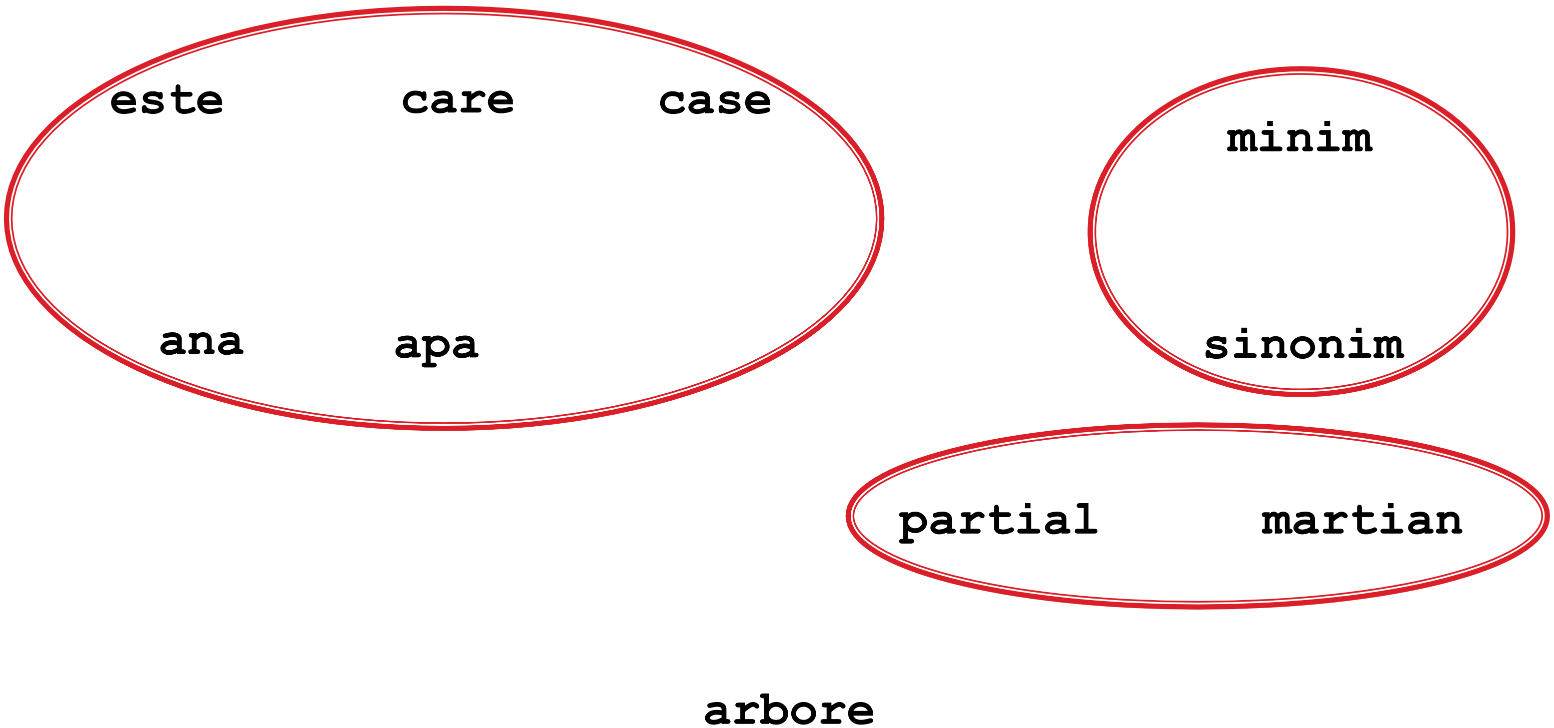
sinonim

partial martian

arbore

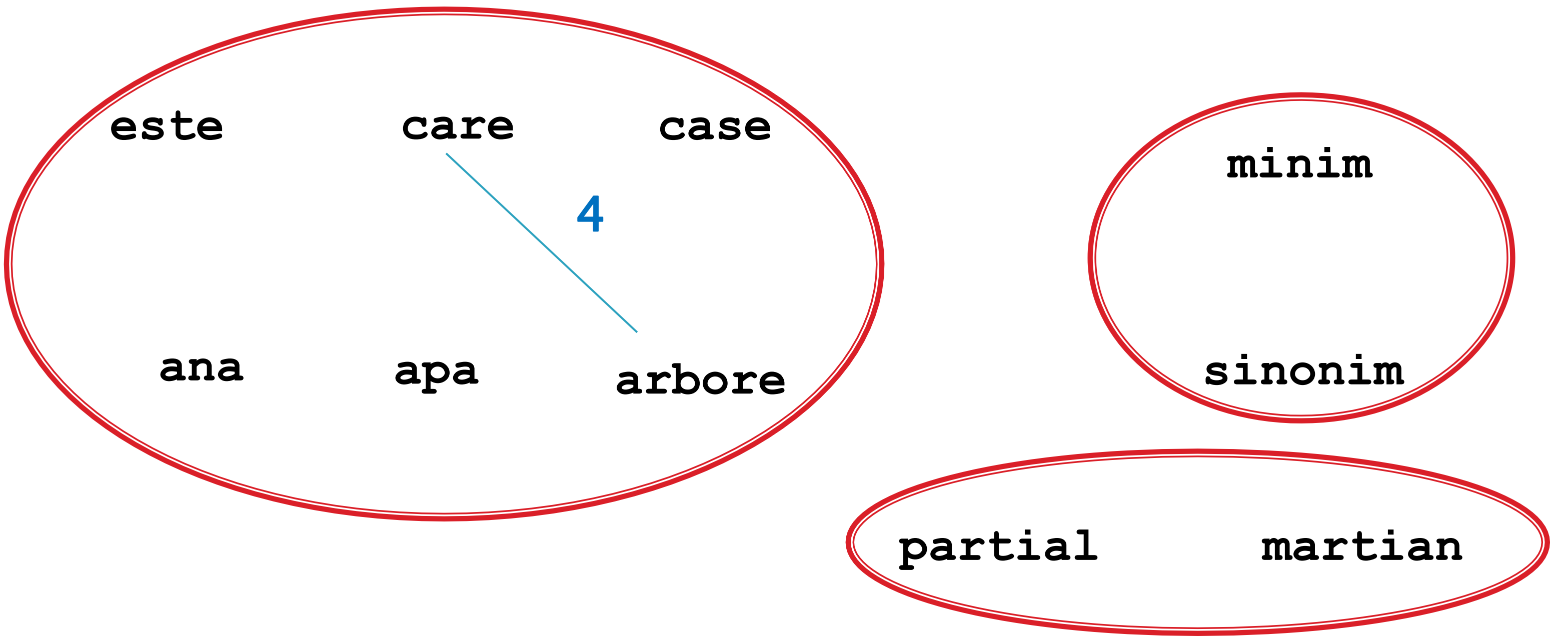
K = 3 clustere

Aplicații – Clustering



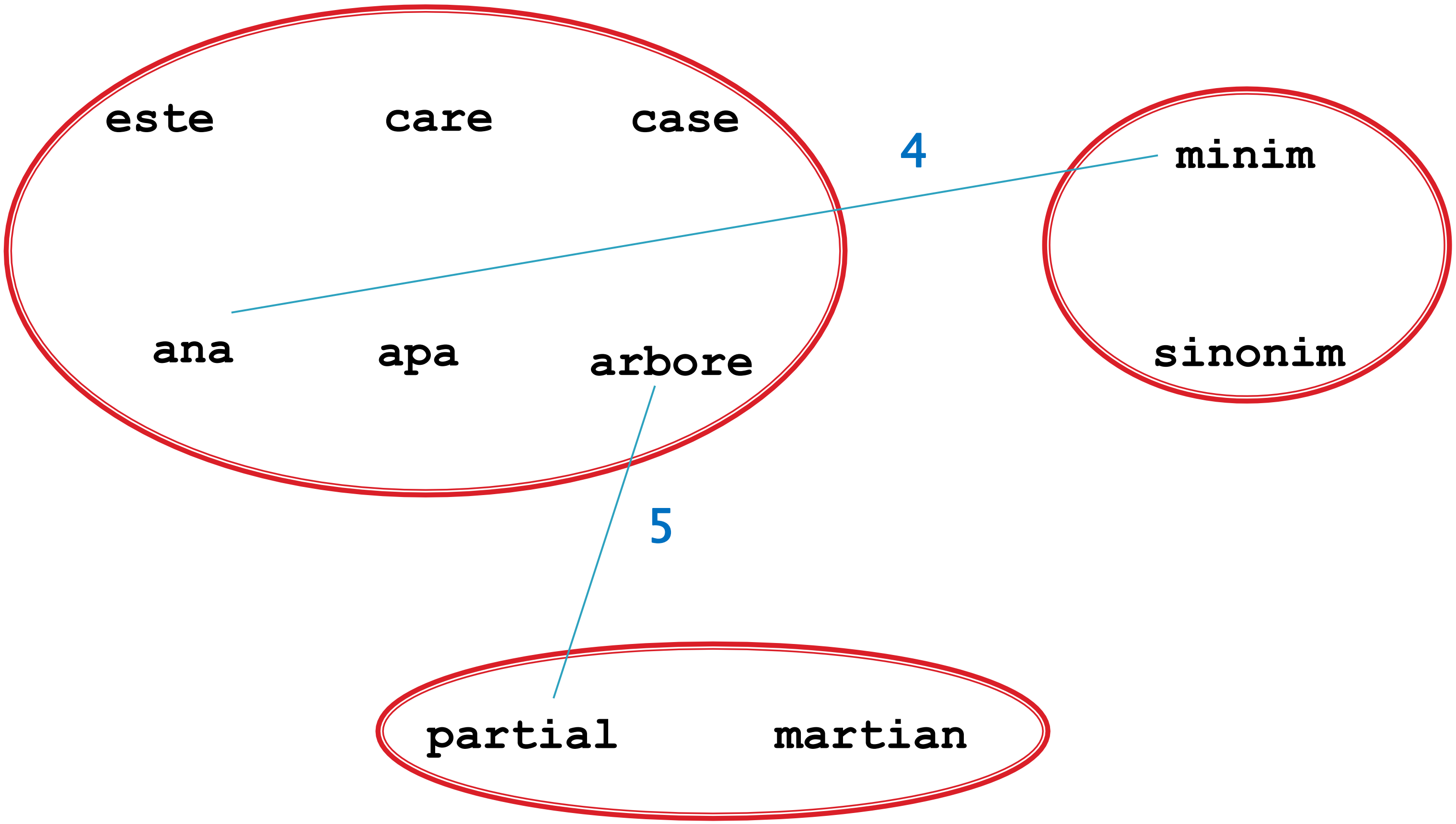
K = 3 clustere

Aplicații – Clustering



K = 3 clustere

Aplicații – Clustering



K = 3 clustere

Aplicații – Clustering

este care case

ana apa arbore

minim

sinonim

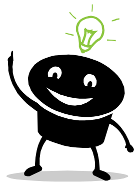
partial martian

K = 3 clustere

Aplicații – Clustering

Idee

- Iniţial fiecare obiect (cuvânt) formează o clasă
- La un pas determinăm cele mai apropiate două obiecte aflate în clase diferite (cu distanţa cea mai mică între ele şi unim clasele lor)
- n – k paşi



Modelare cu graf $\Rightarrow$ n – k paşi din algoritmul lui Kruskal

Aplicații - Clustering

Corectitudine

- k-clusteringul obţinut are grad de separare maxim

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

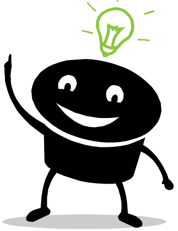
O primă formă a algoritmului

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**)
  - $E(T) = E(T) \cup uv$

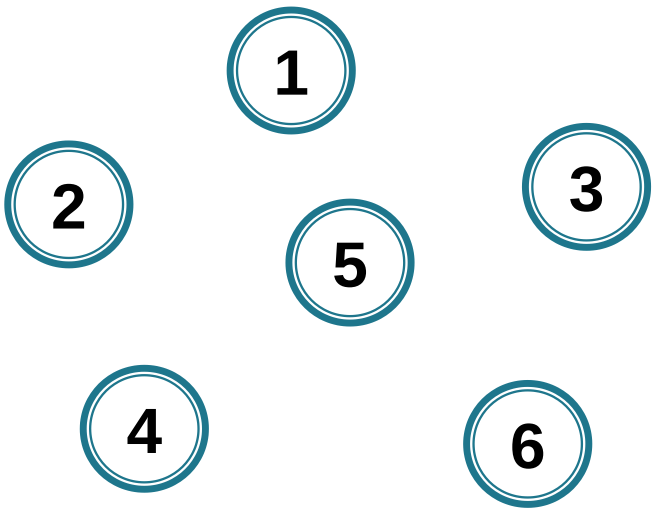
Prim

- **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

- **La un pas**:

Muchiile selectate formează o **pădure**

Prim

- **La un pas**:

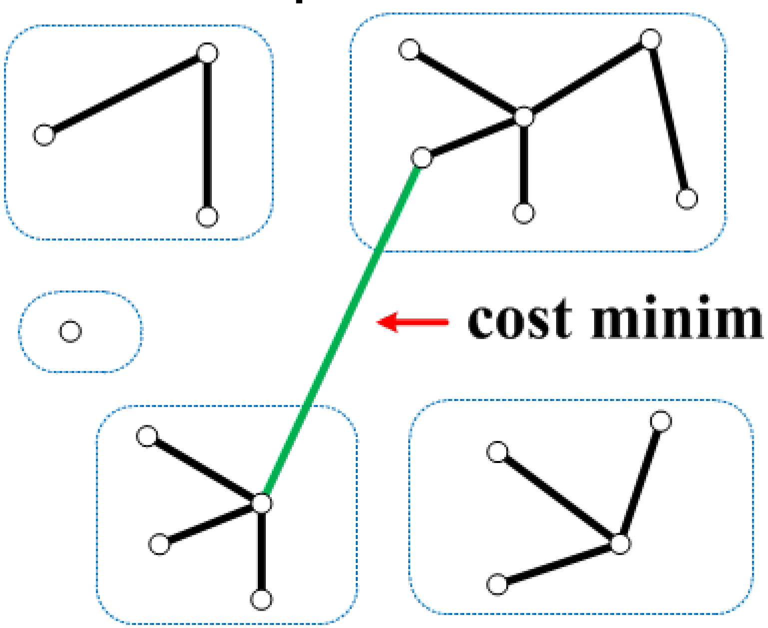
Muchiile selectate formează un **arbore**

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)

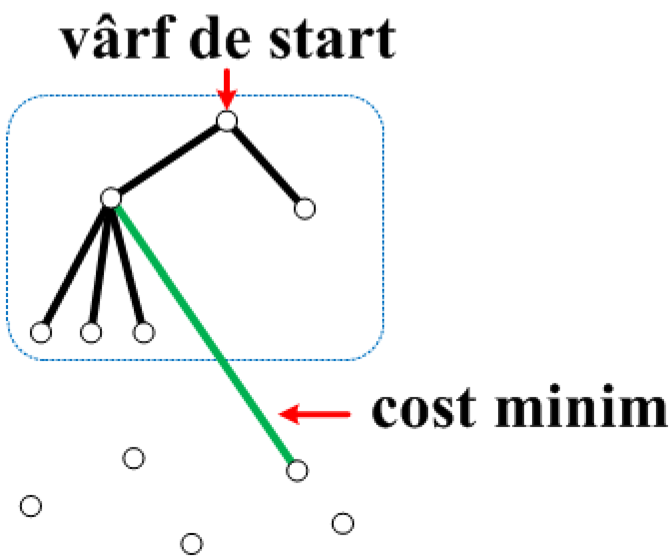


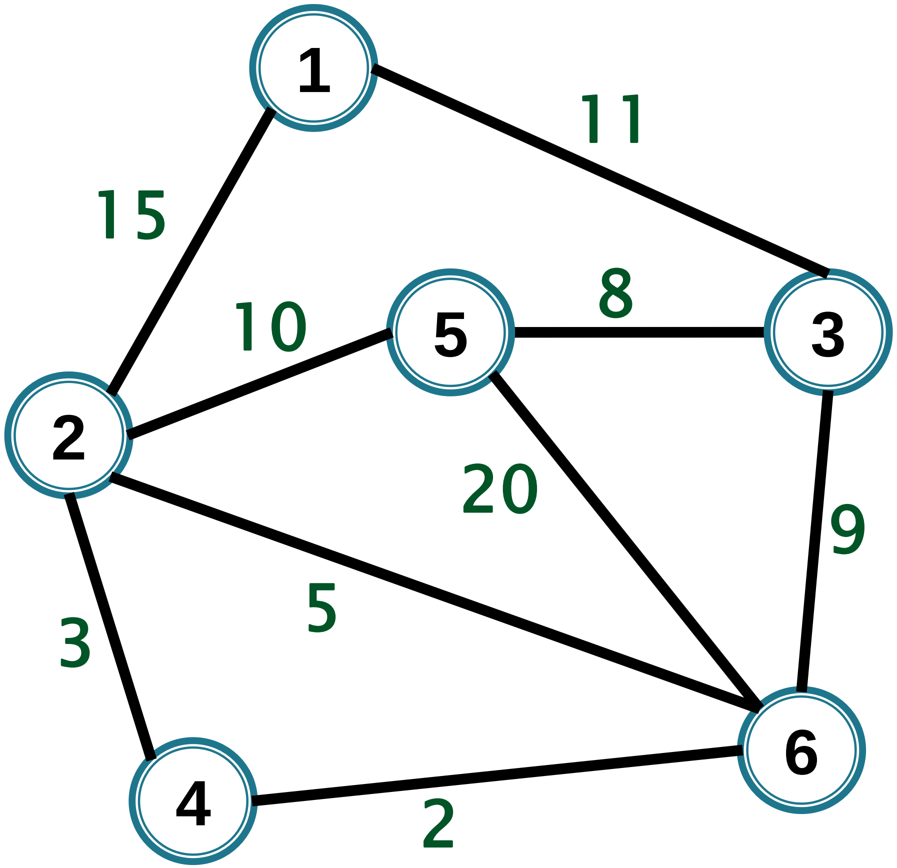
Prim

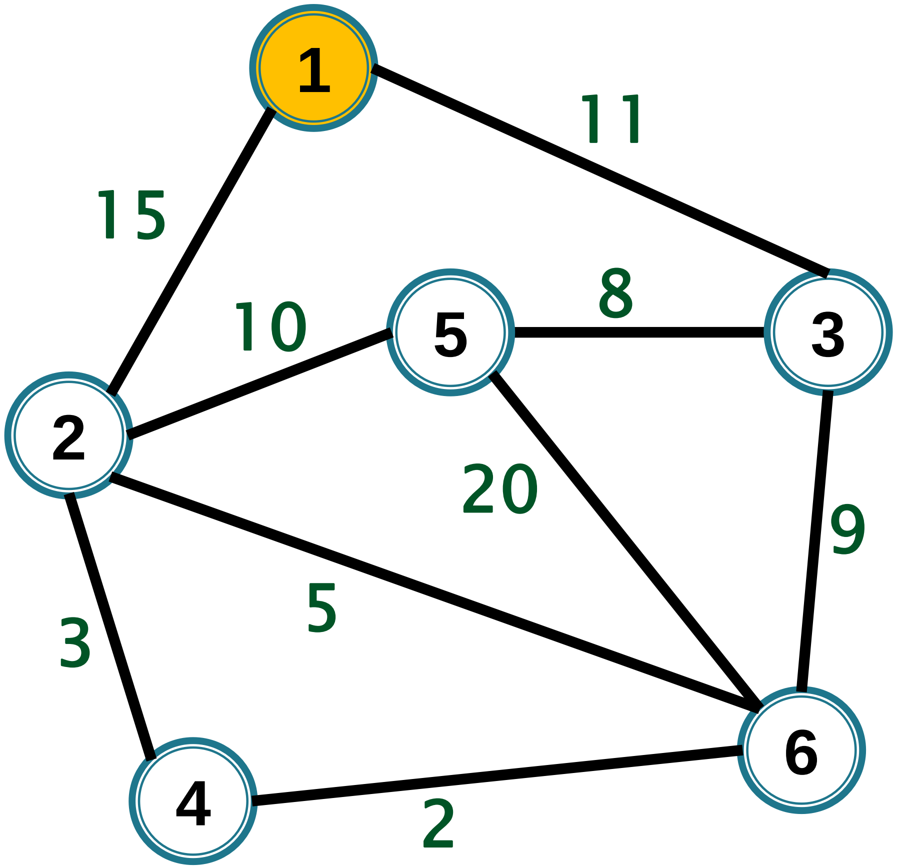
- **<u>La un pas</u>**:

Muchiile selectate formează un **arbore**

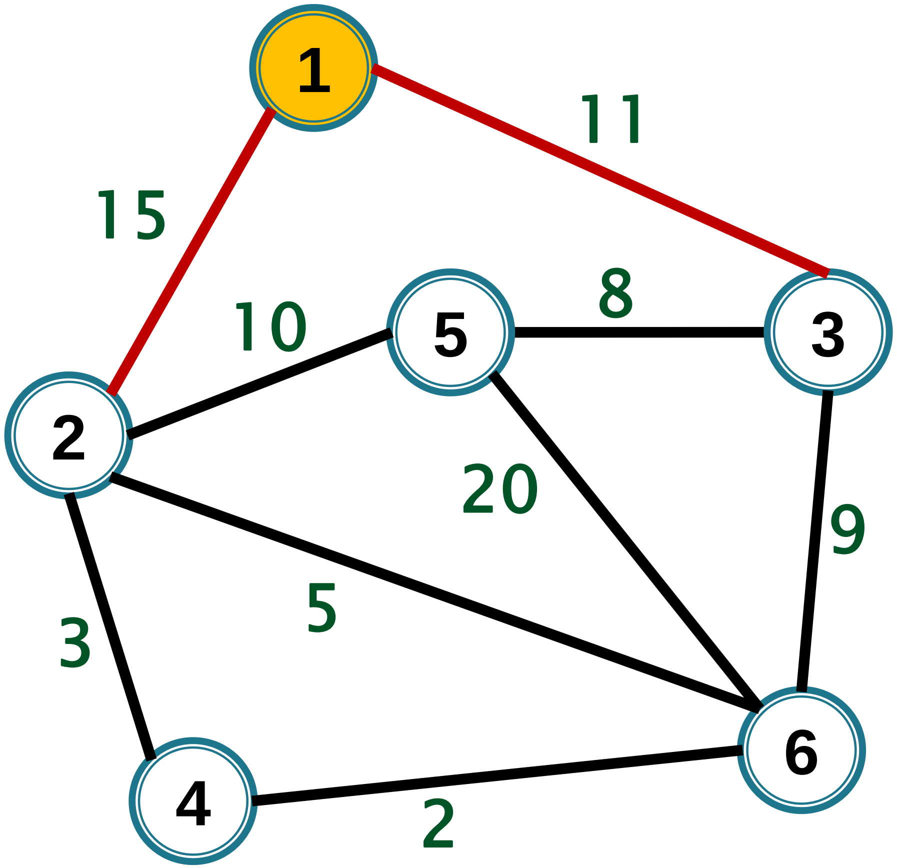
Este selectată o muchie de cost minim care unește un vârf din arbore cu unul care nu este în arbore(neselectat)



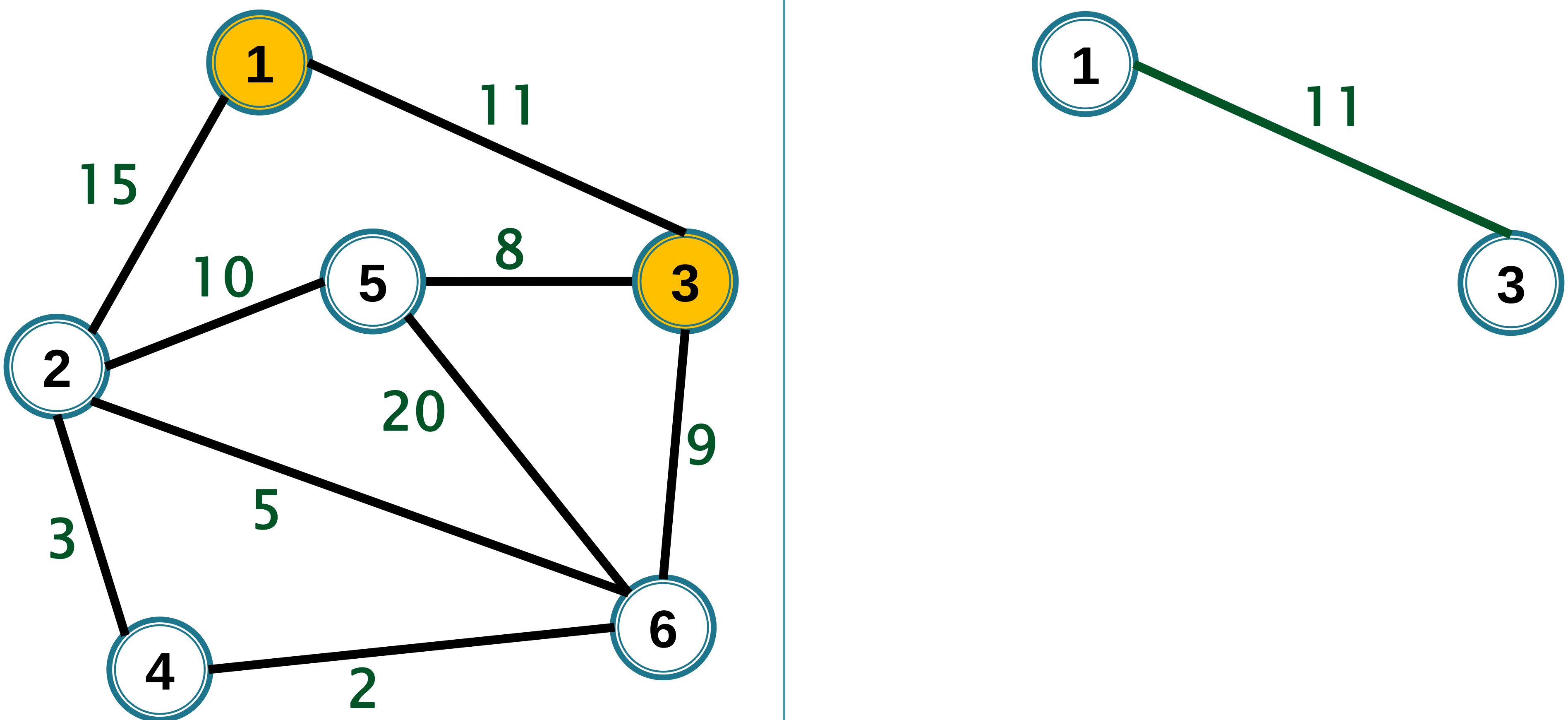

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2

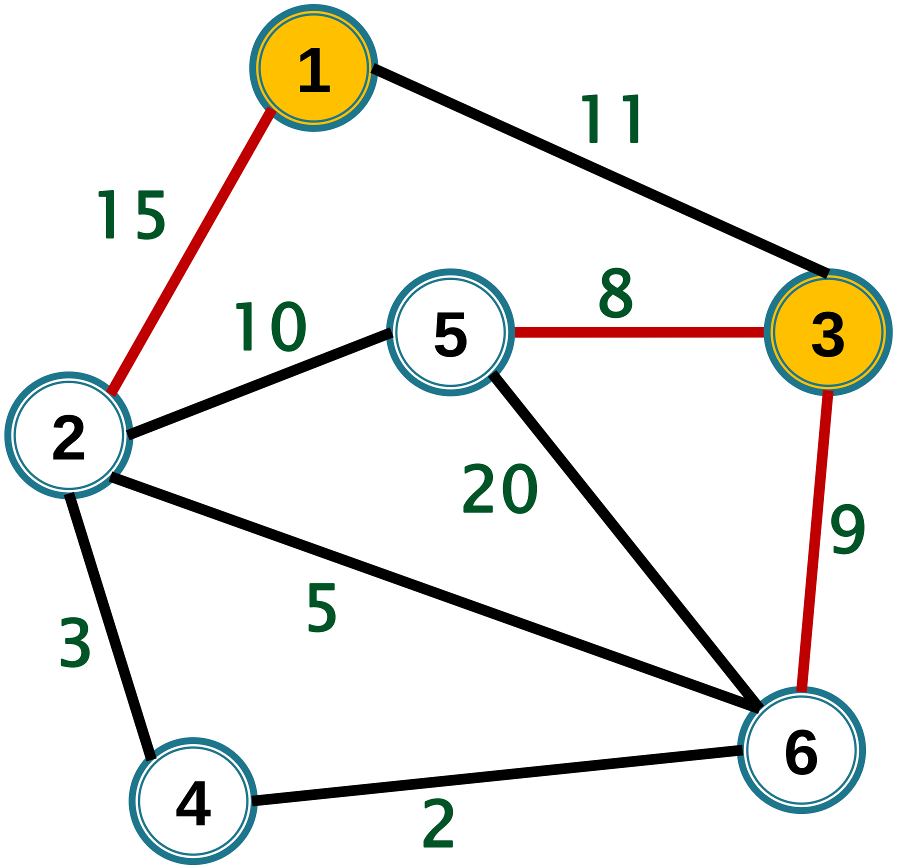


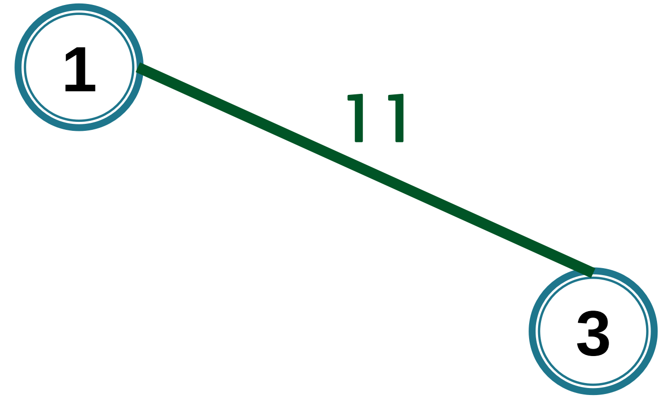
s = 1

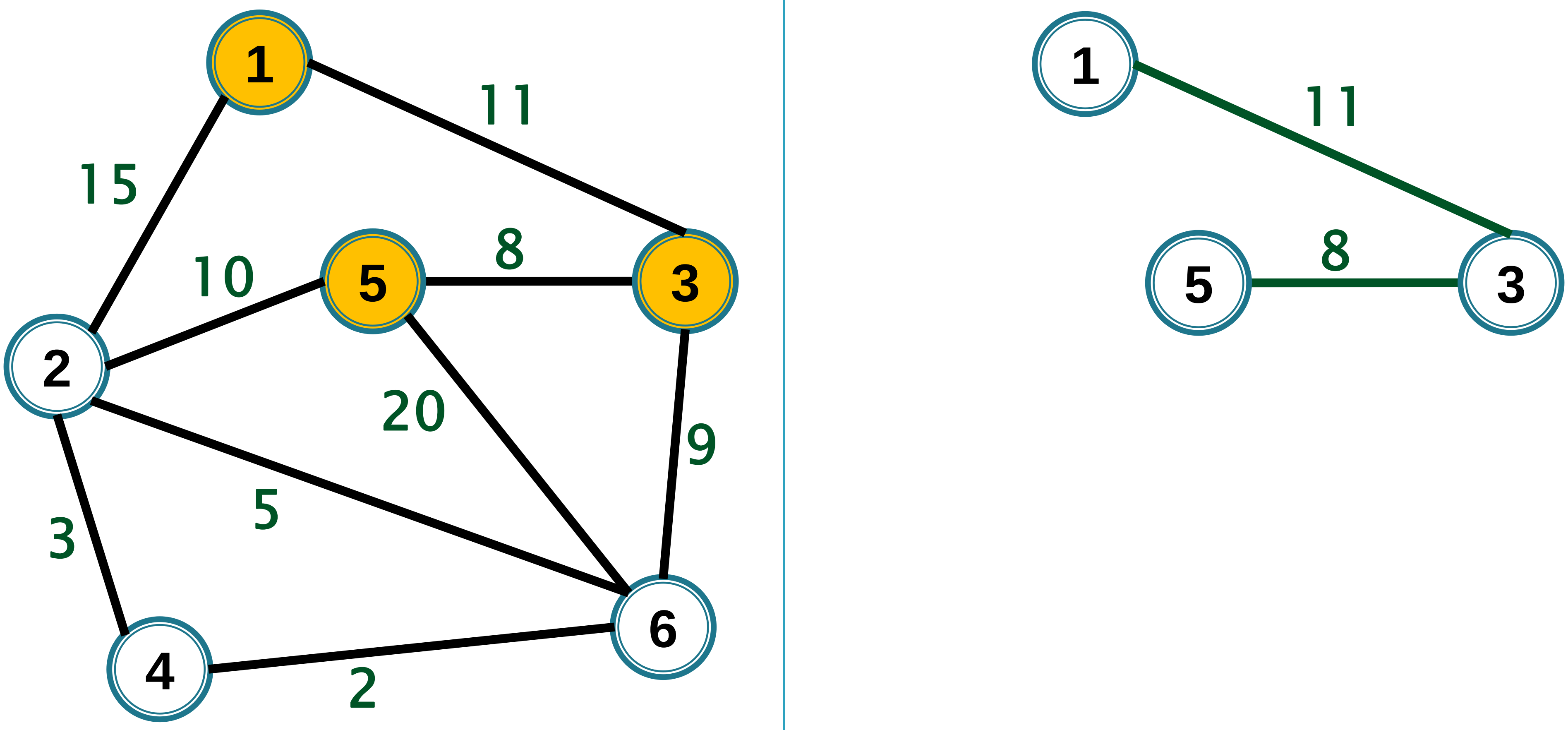

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2

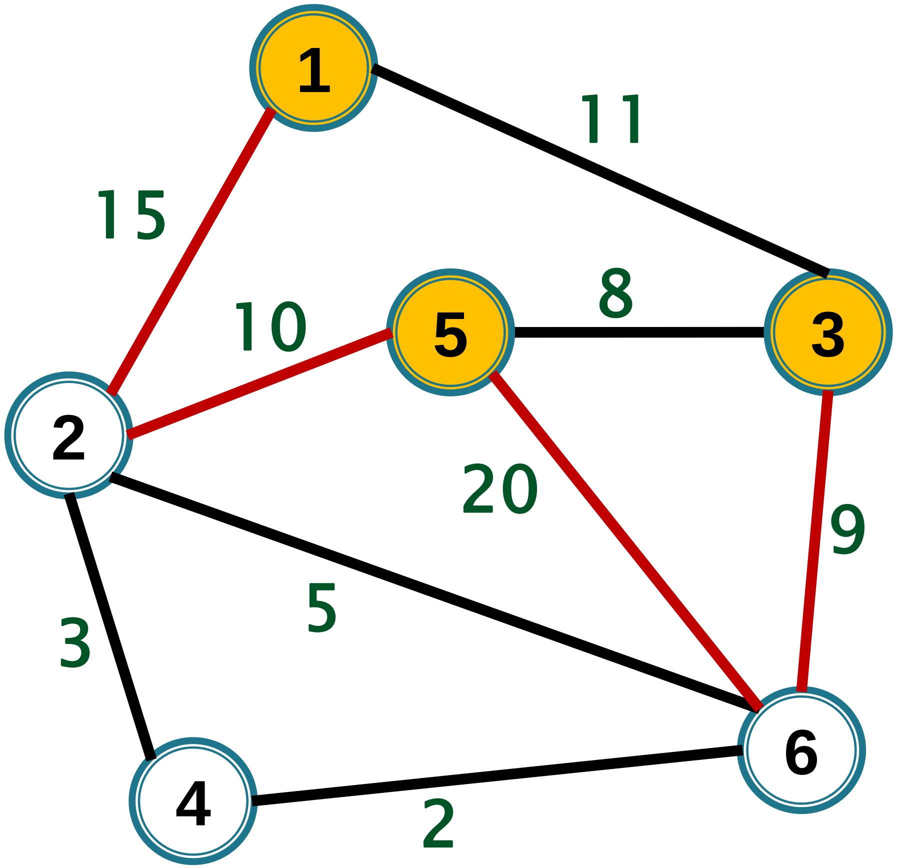

1

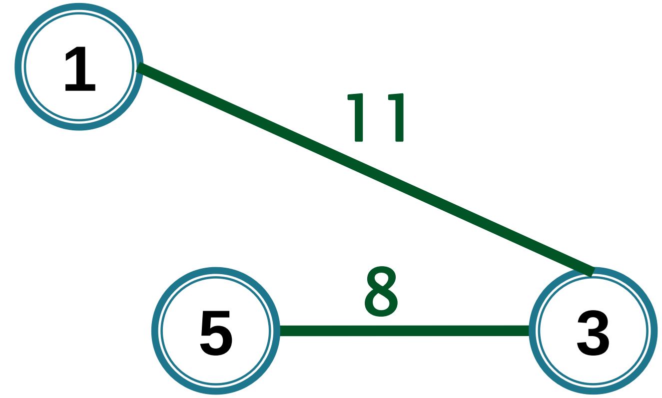

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2
1
3
11

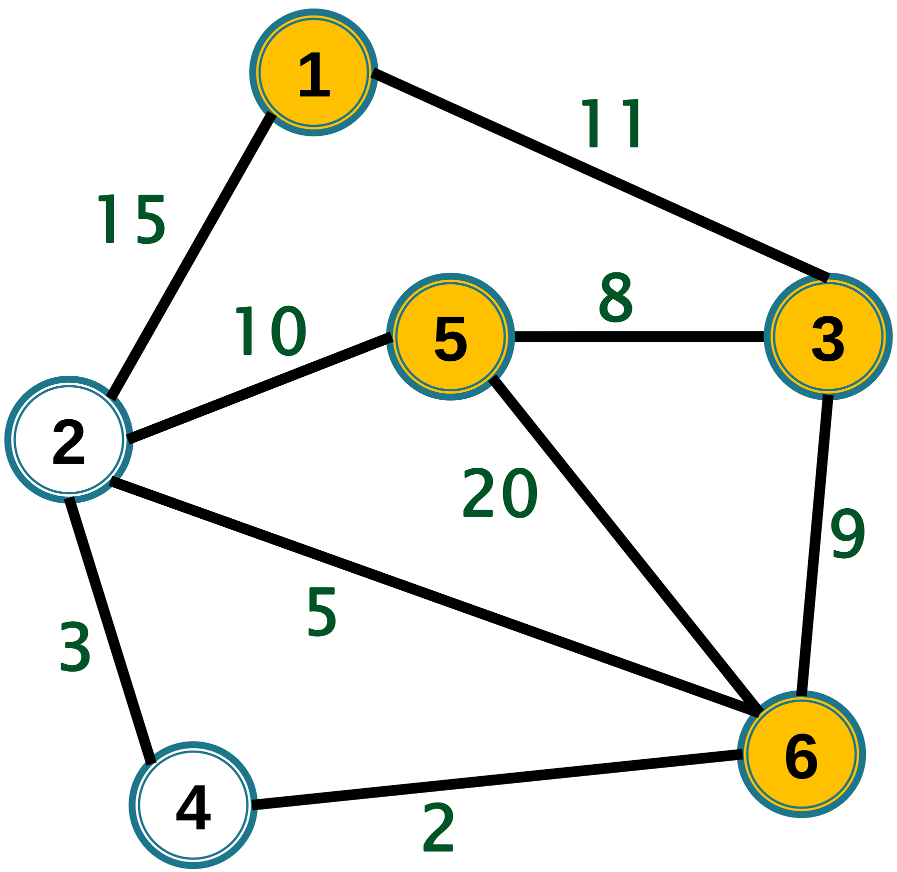

1
11
15
10
5
8
3
2
20
9
5
3
6
4
2

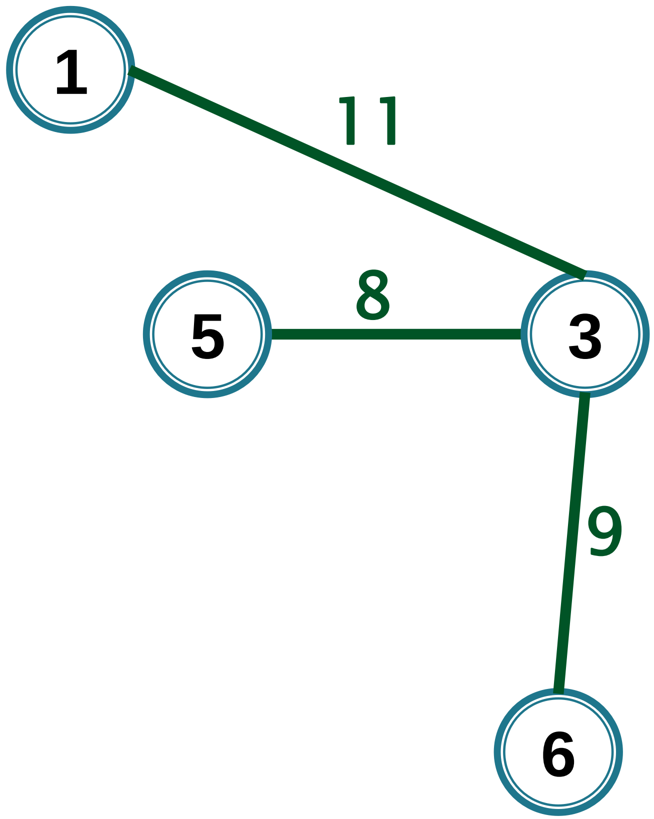

1
11
3

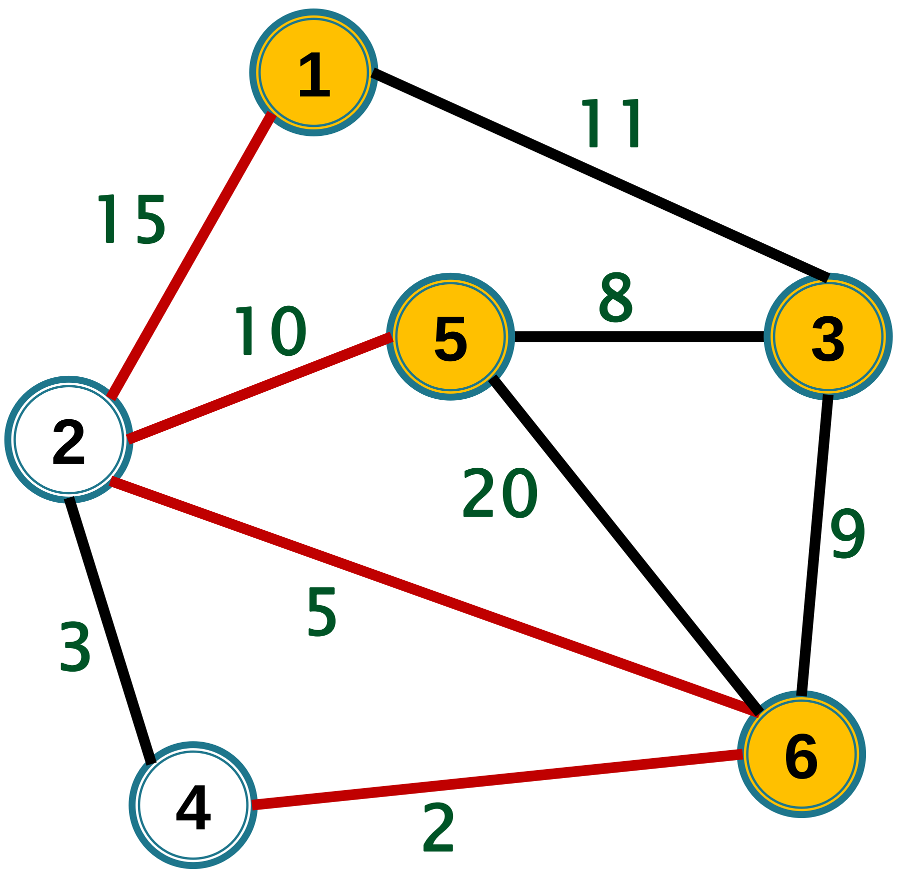

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2
1
5
3
11
8

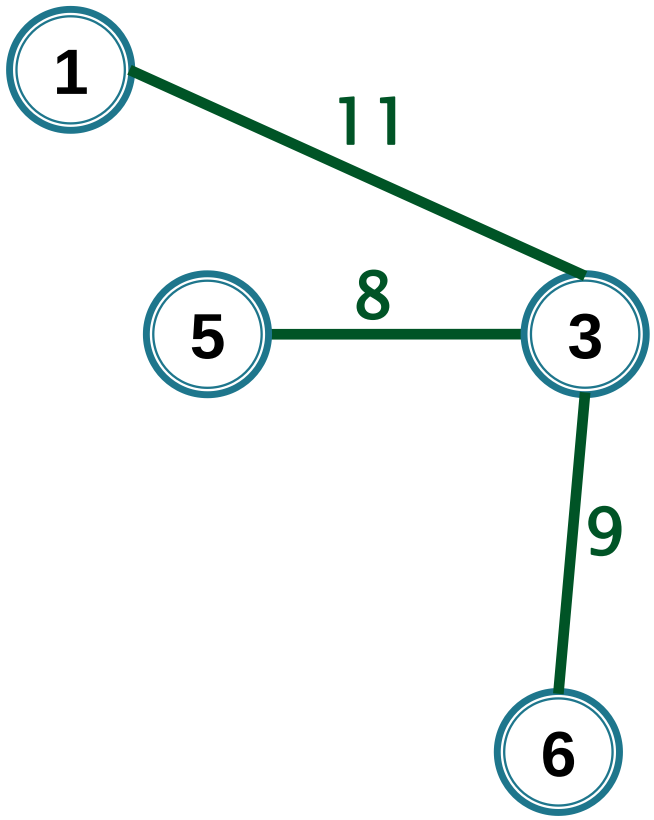

1
2
3
4
5
6
11
15
10
8
20
9
5
3
2

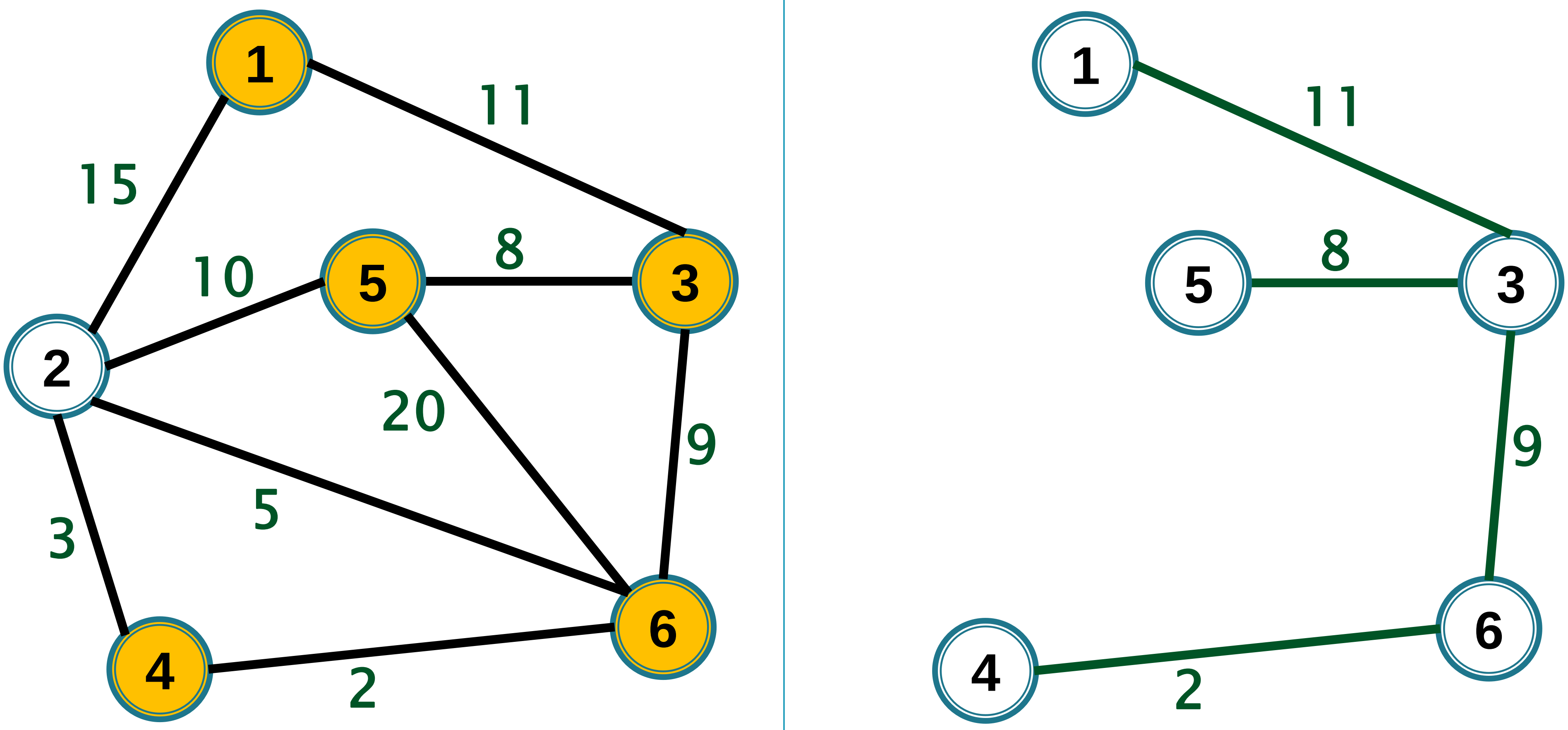

1
5
3
11
8

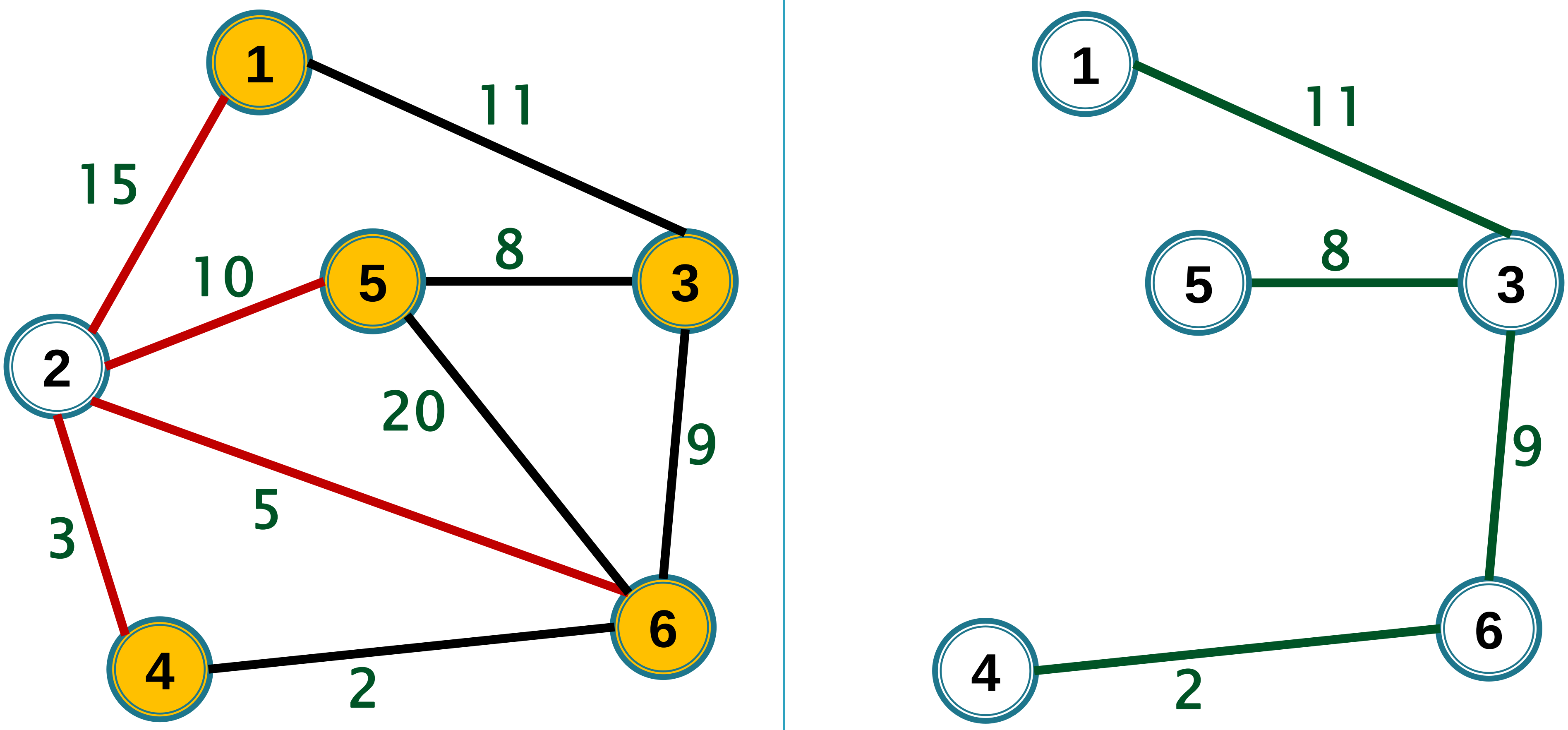

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2

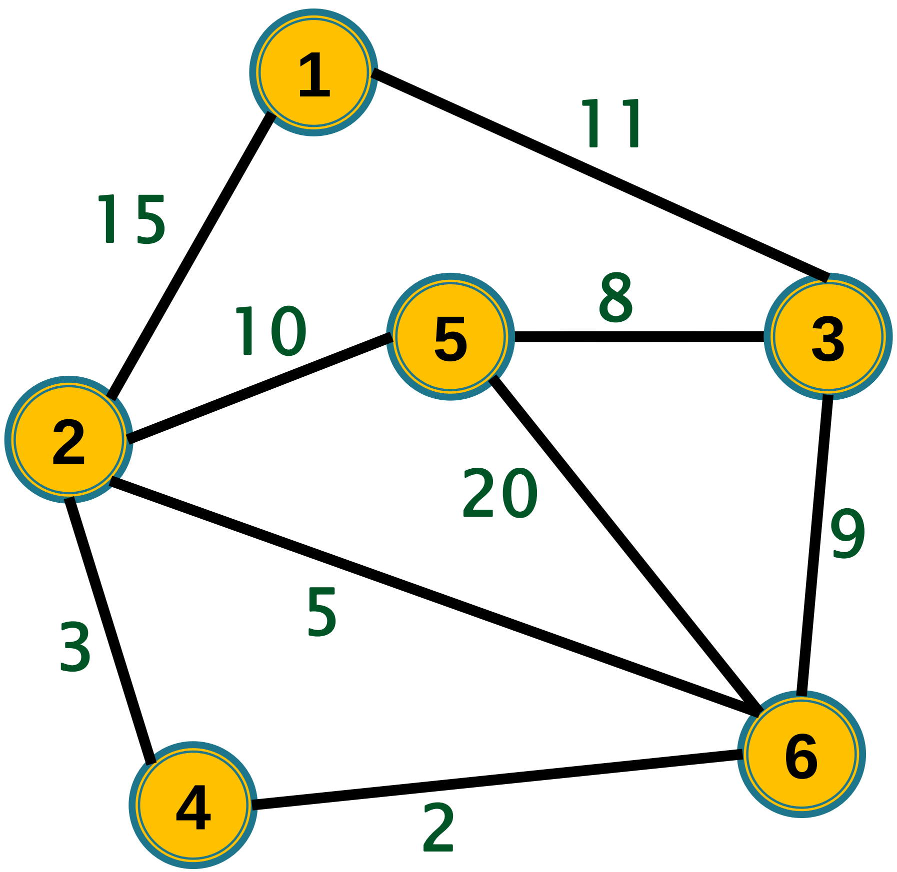

1
3
5
6
11
8
9

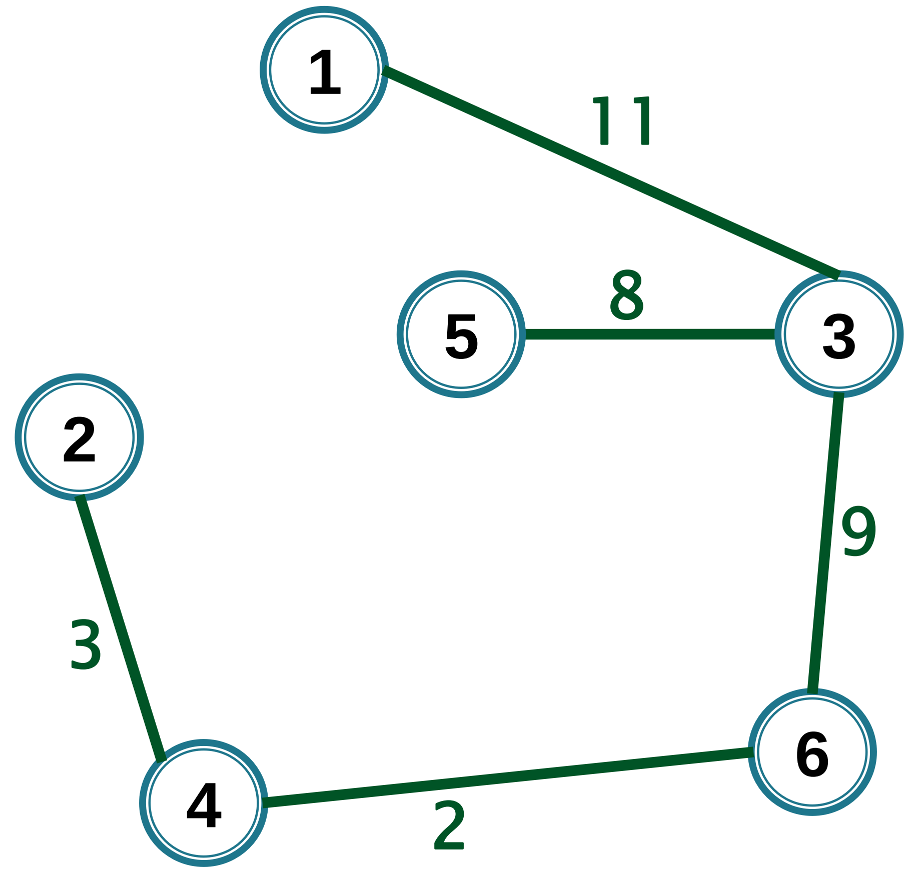

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2

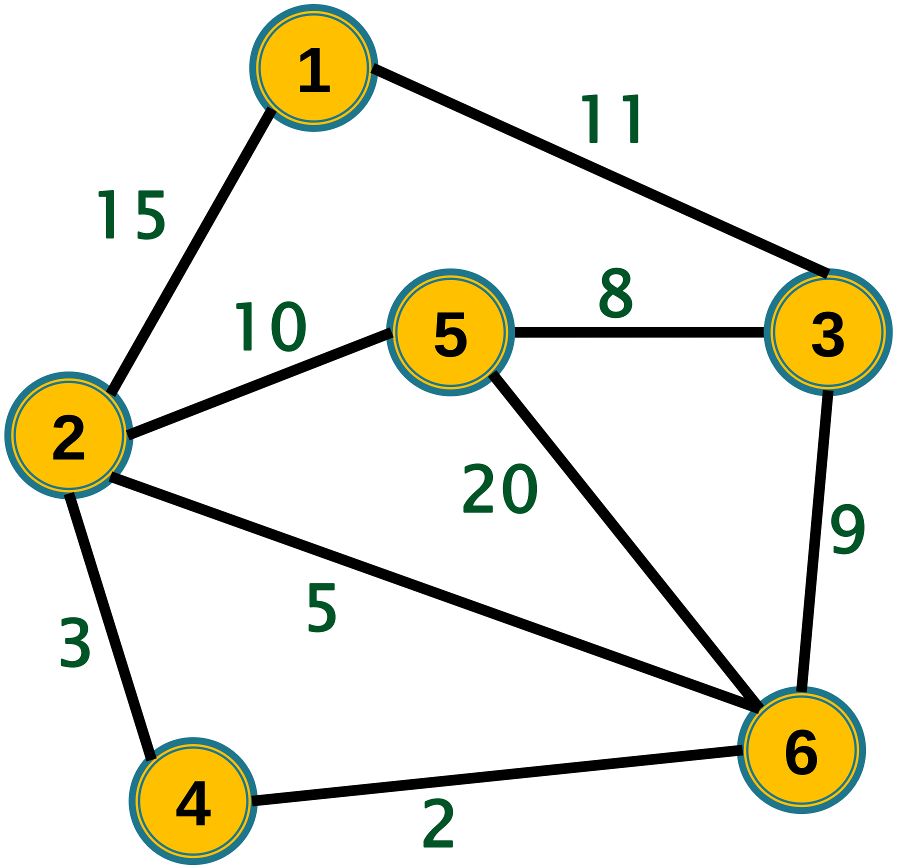

1
3
5
6
11
8
9

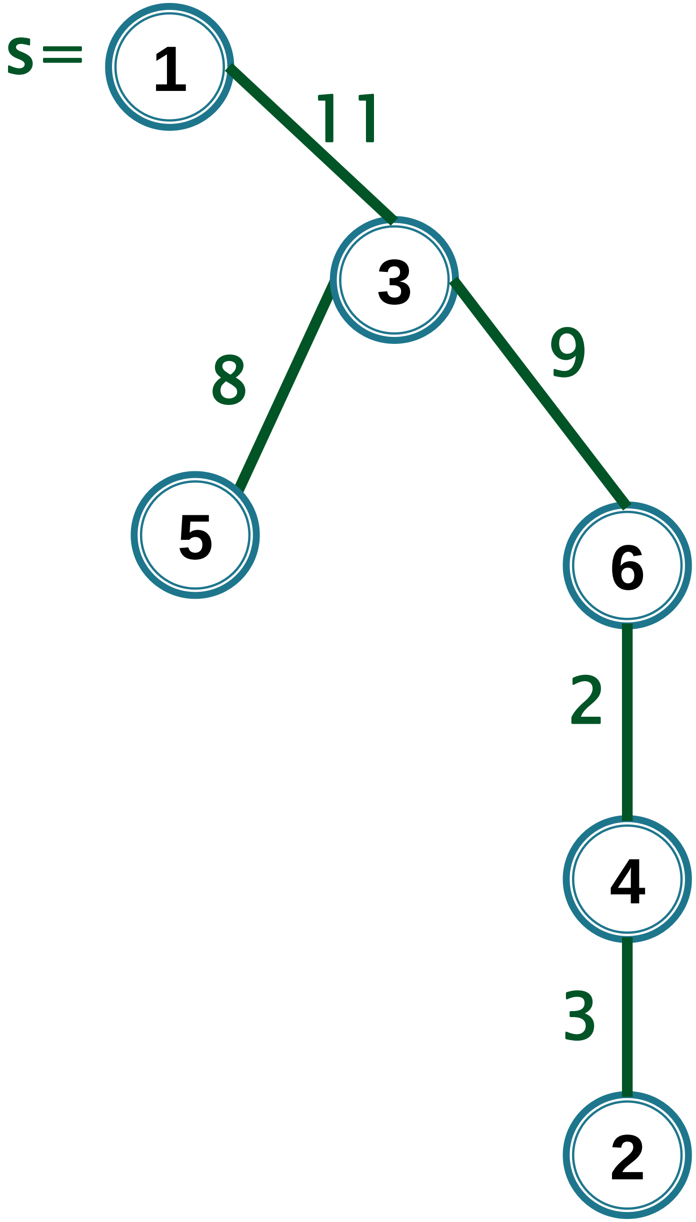

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2
1
3
5
6
4
11
8
9
2


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


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


1
2
3
4
5
6
11
8
9
3
2

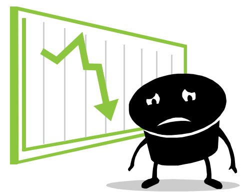

1
2
3
4
5
6
15
11
10
8
20
9
5
3
2


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

Implementare

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

O(nm)



Implementare

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

- heap de muchii

sau

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

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

Algoritmi bazaţi pe eliminare de muchii

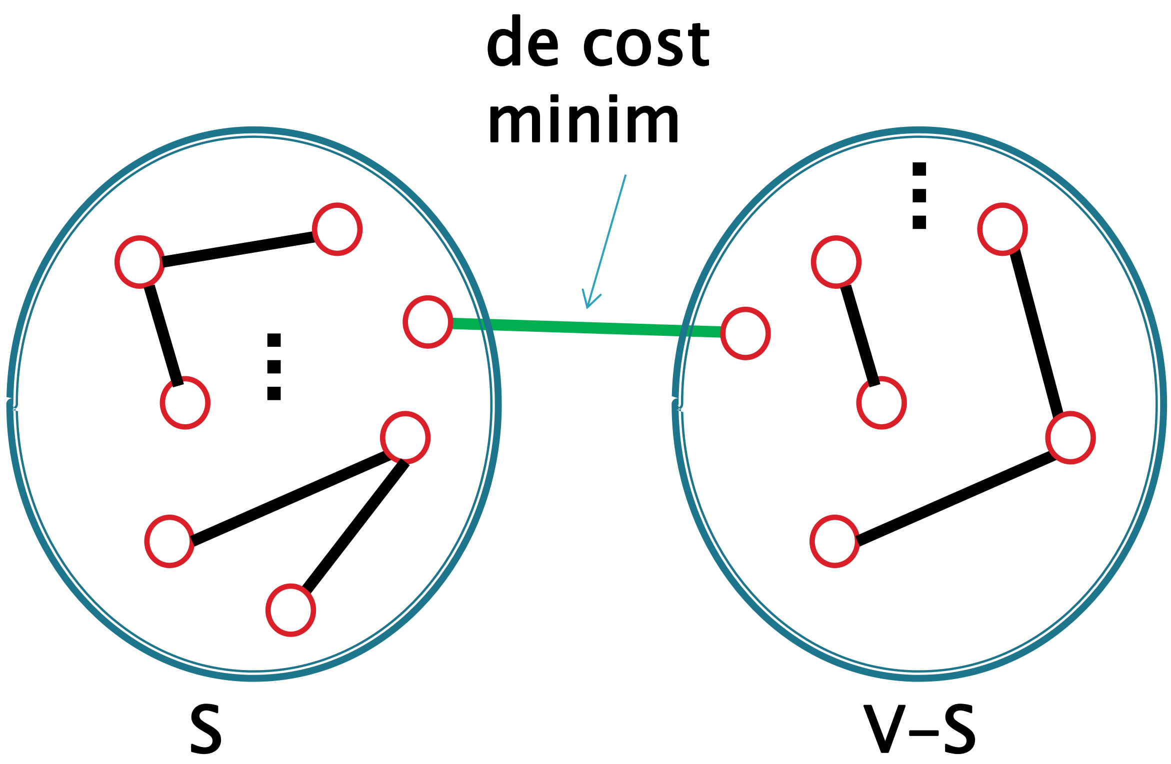
Temă – Care dintre următorii algoritmi determină corect un arbore parţial de cost minim (justificaţi)? Pentru fiecare algoritm corect precizaţi ce complexitate are.

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

Corectitudinea algoritmilor



Corectitudinea algoritmilor

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

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