UNIVERSITAT DE BARCELONA

Greedy

Algorísmica Avançada | Enginyeria Informàtica

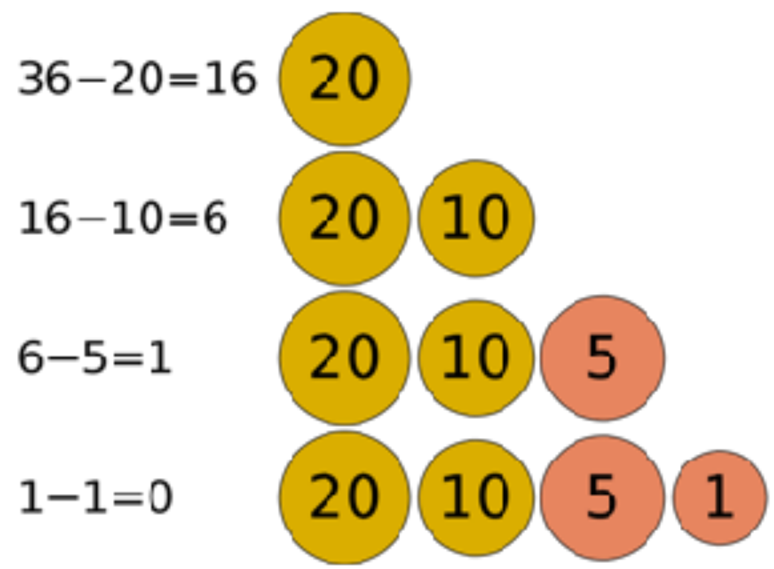
Santi Seguí | 2019-2020

Greedy?

- L’**algorisme greddy (o voraç)** és un algorisme que, per resoldre un problema d'optimització, fa una seqüència d'eleccions, prenent en cada pas un òptim local, amb l'esperança (no sempre complerta) d'arribar a un òptim global. L'algorisme greddy no torna mai enrere per reavaluar les eleccions ja preses

Exemple

- Tornar un canvi amb el mínim de monedes. El conjunt de candidats és {20, 10, 5, 1}

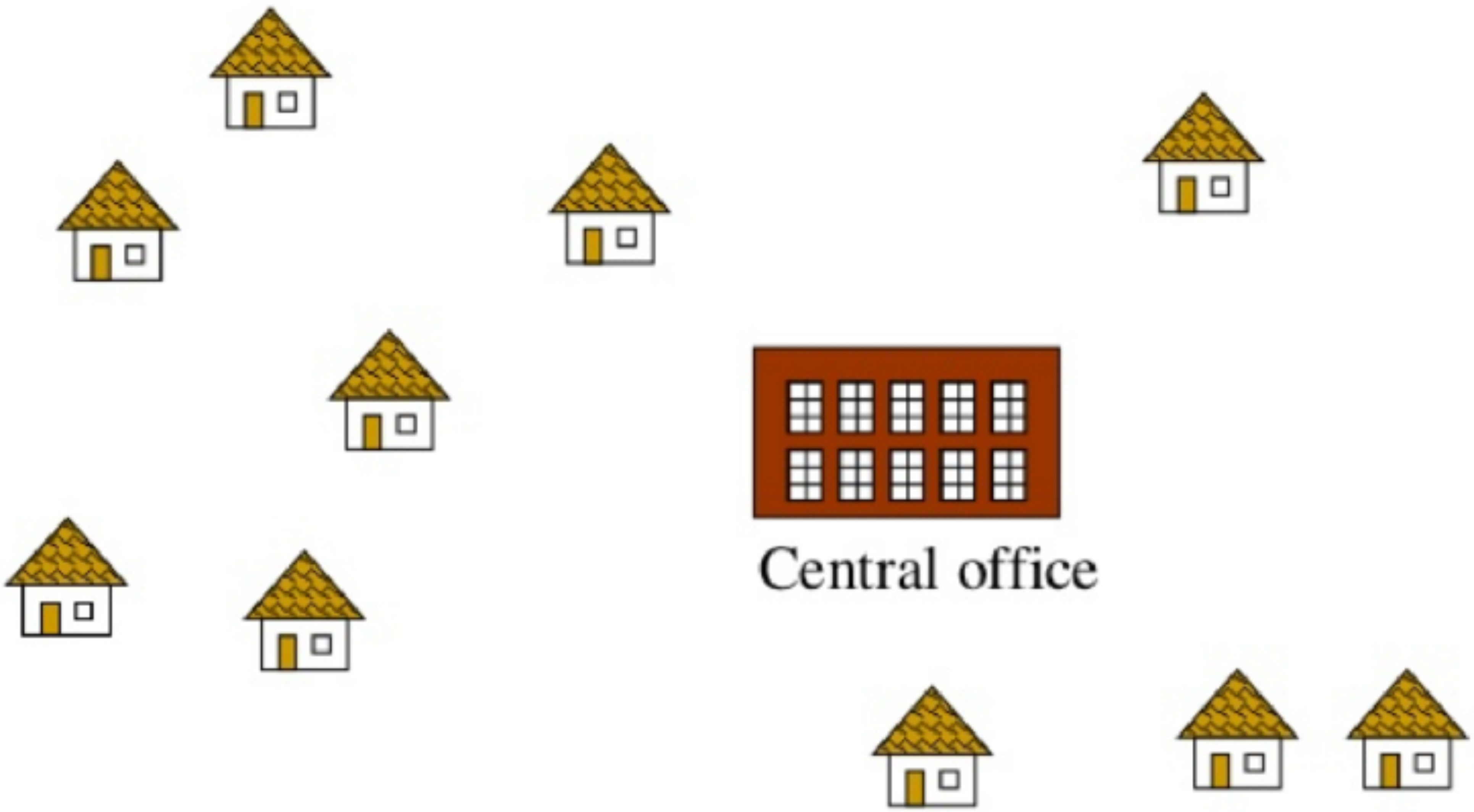


Els algorismes voraços **no garanteixen** sempre **una solució**, **ni** que la **solució** obtinguda sigui **l'òptima**

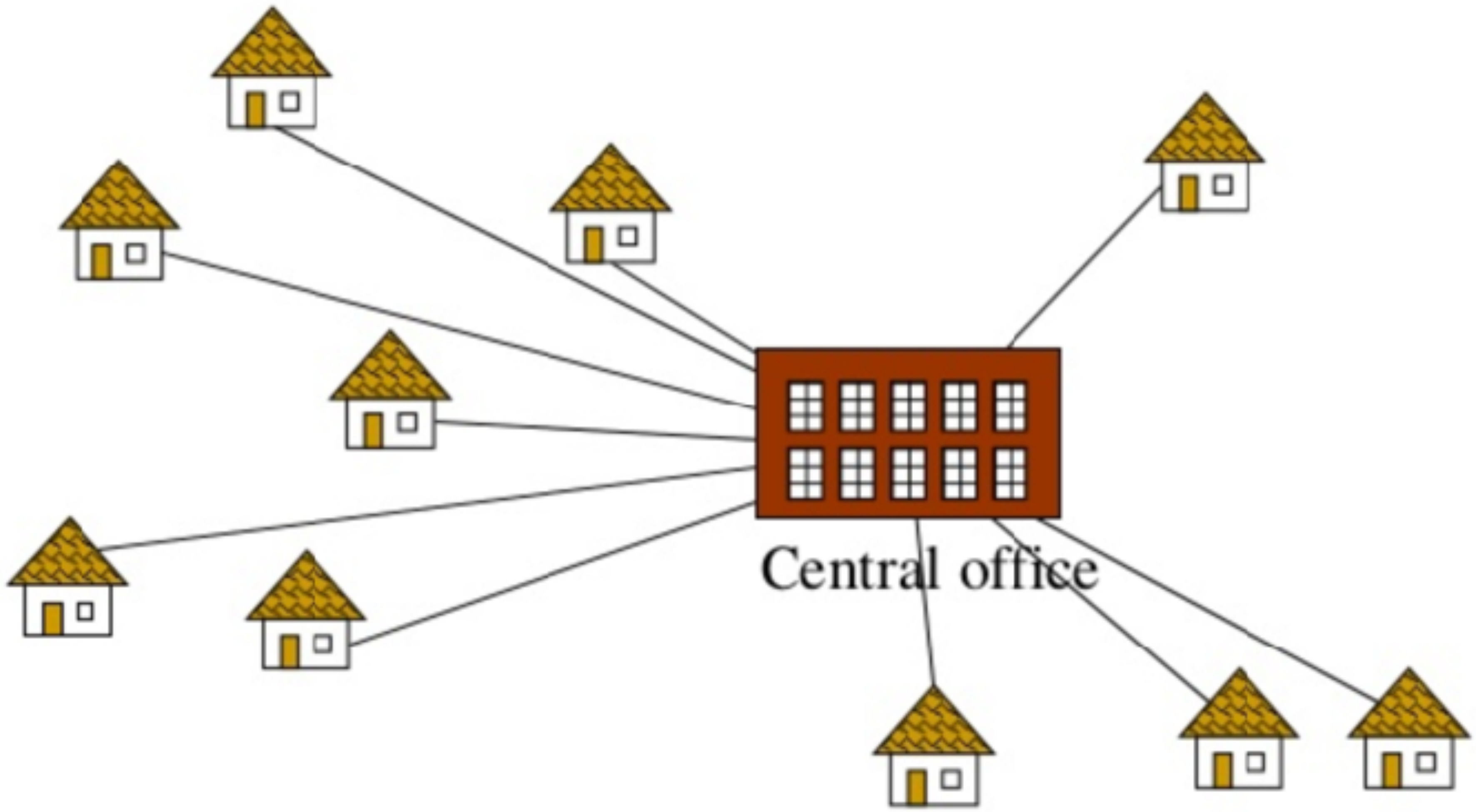
Greedy

- Podem guanyar als escacs pensant només en la següent jugada?
- I al scrabble? → algorisme greedy?
- **Algoritmes greedy troben la millor “jugada” a cada pas**

Problem: Laying Telephone Wire

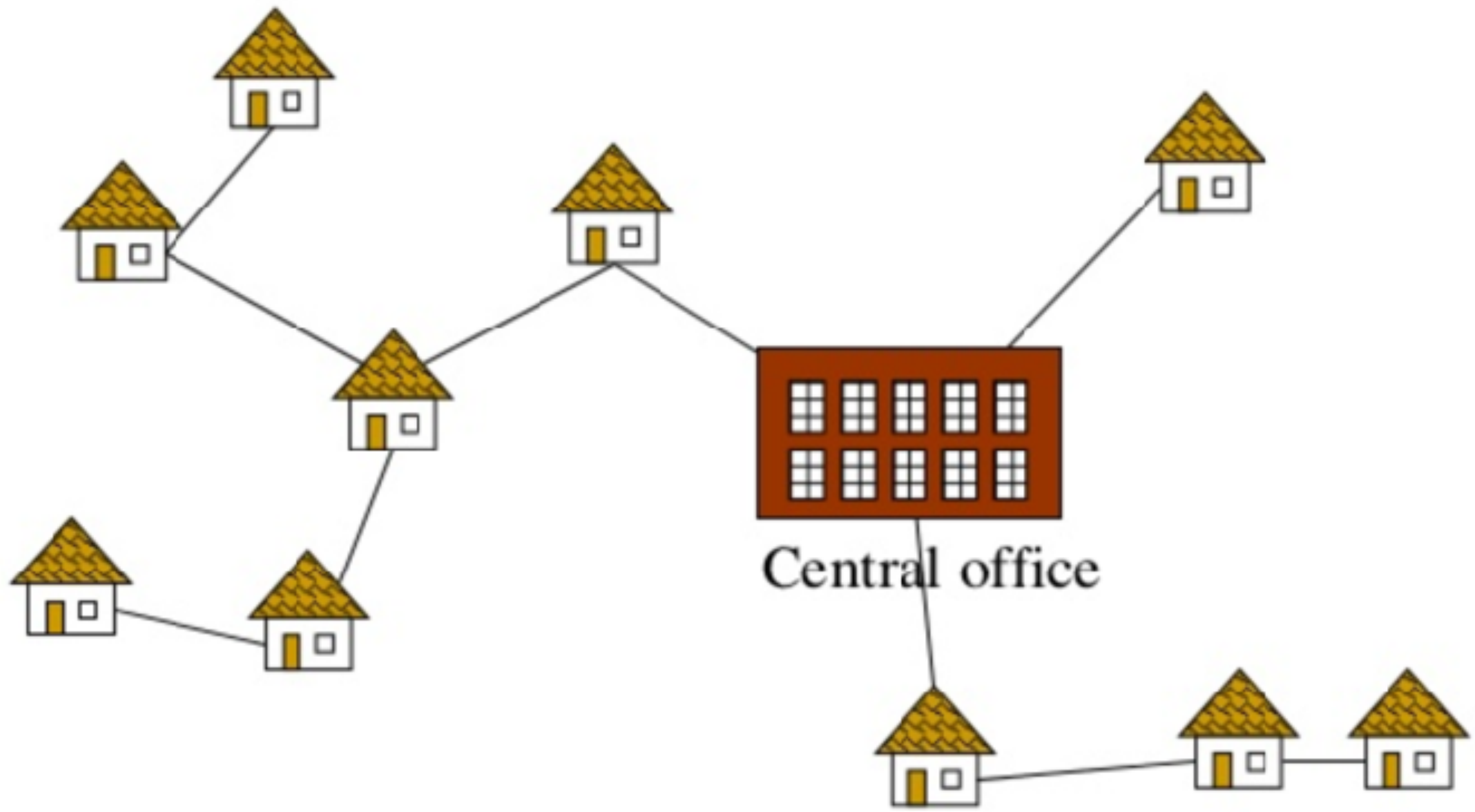


Wiring: Naïve Approach



Expensive!

Wiring: Better Approach



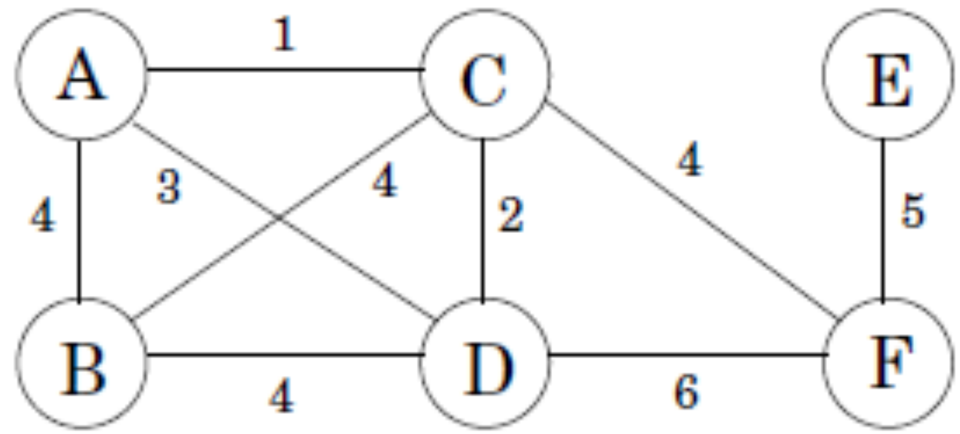
Minimize the total length of wire connecting the customers

Exemple

Teniu un negoci amb diverses oficines; voleu llogar línies de telèfon per connectar-les entre elles; i l'empresa de telefonia cobra diferents quantitats de diners per connectar les oficines. Volem un conjunt de línies que **connecti totes** les seves oficines amb un **cost total mínim**. Hauria de ser un arbre extensiu, ja que si una xarxa no és un arbre, sempre podeu treure algunes vores i estalviar diners.

Algoritmes Greedy

- Exemple

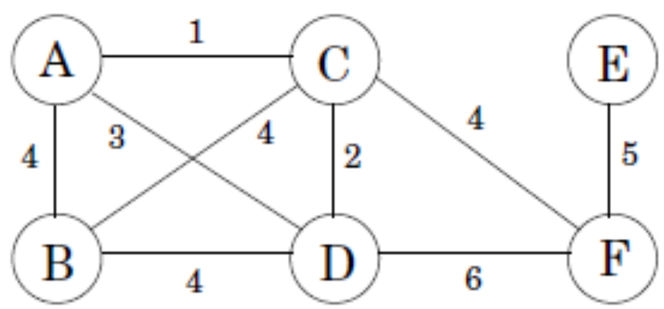


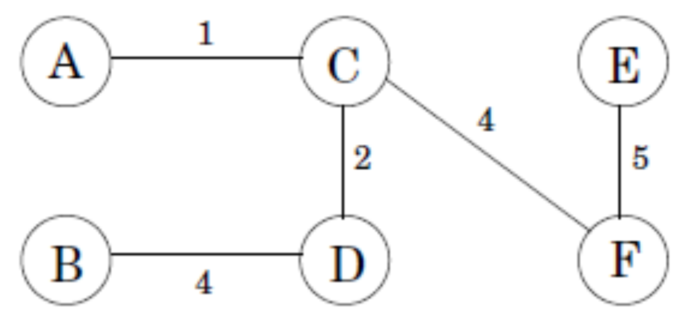
- Volem connectar els oficines (nodes) d'una empresa. Les connexions són les arestes. Cadascuna té un cost. Volem el mínim cost.
  - → llavors no volem cicles
  - → volem un graf no dirigit acíclic connectat
    - → arbre !!!
  - → de mínim cost: **Minimum Spanning Tree** (MST)

Kruskal

Algoritmes Greedy

- MST amb cost 16 (un dels possibles)



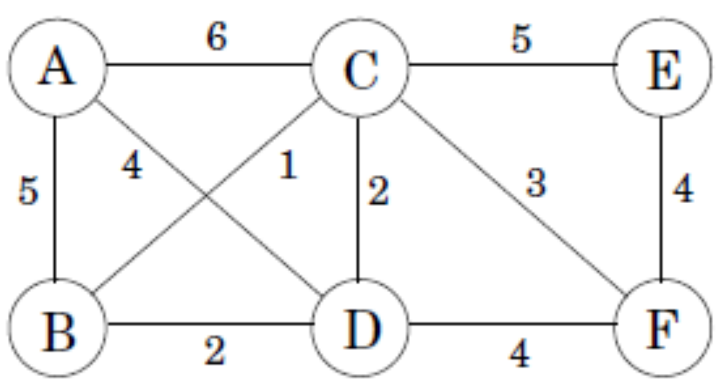


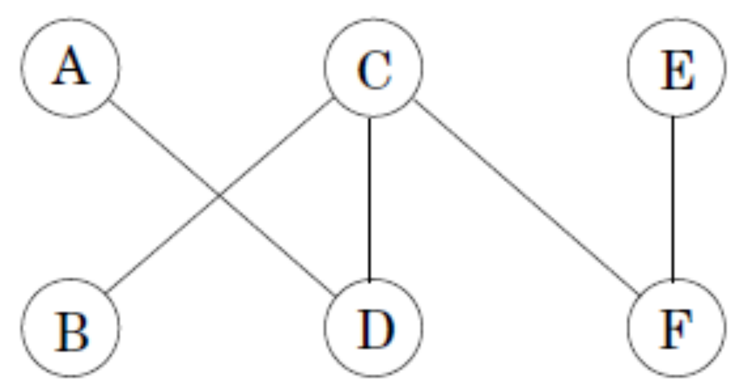
- Algorisme greedy: **Kruskal**
  - Començar amb arbre buit
  - Mentre no estiguin tots els nodes connectats
    - Incloure aresta de cost mínim que no produeix un cicle

Algoritmes Greedy

- Cost 14!

$$B-C,\ C-D,\ B-D,\ C-F,\ D-F,\ E-F,\ A-D,\ A-B,\ C-E,\ A-C.$$

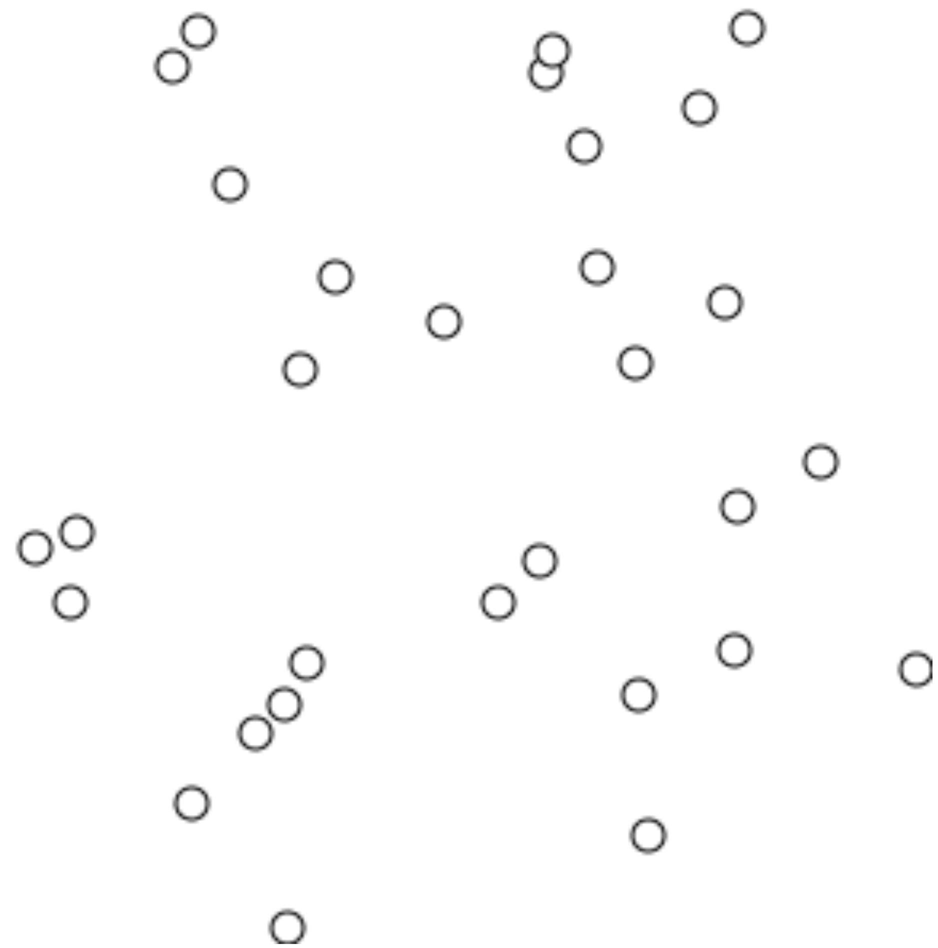




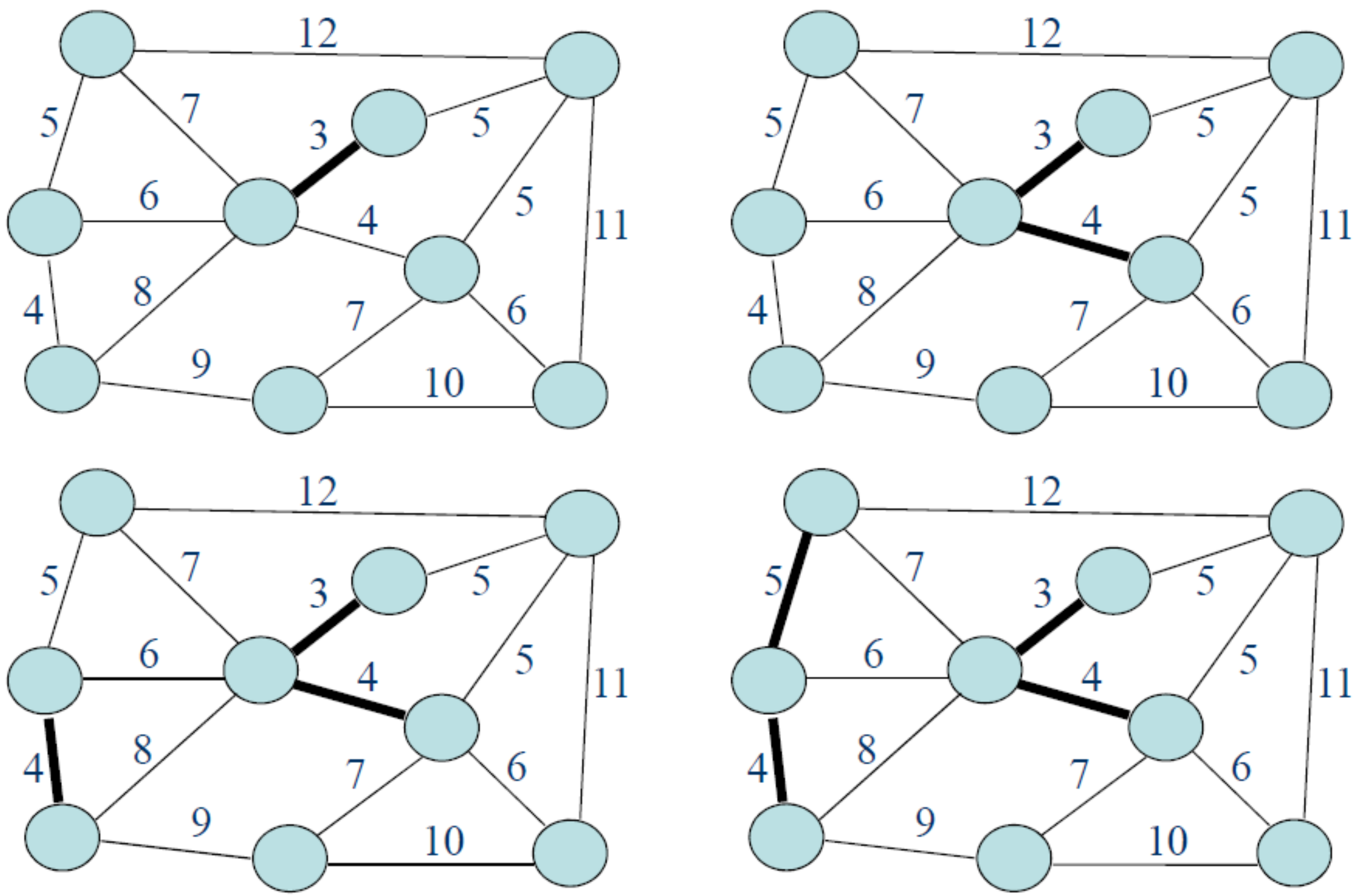
- Aquest algoritme és **òptim!**

Algoritmes Greedy

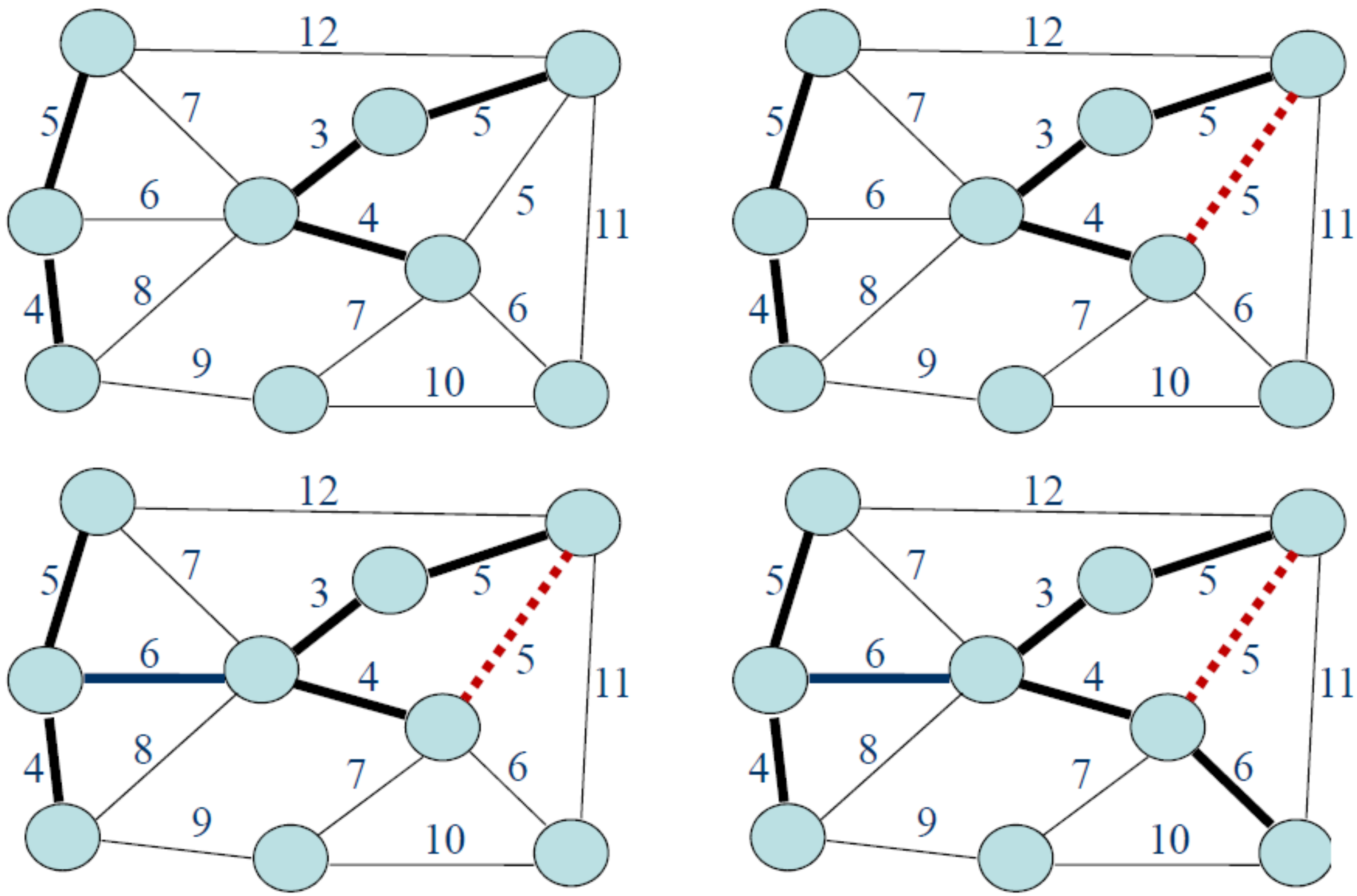
- Per què? Propietat de tall ("cut"):
  - Un **tall** és aquella aresta que si la traiem es **genera una nova component connexa**.
  - El que fem amb Kruskal és anar connectant elements amb el tall de cost mínim.
  - Cóm ho podem implementar eficientment?



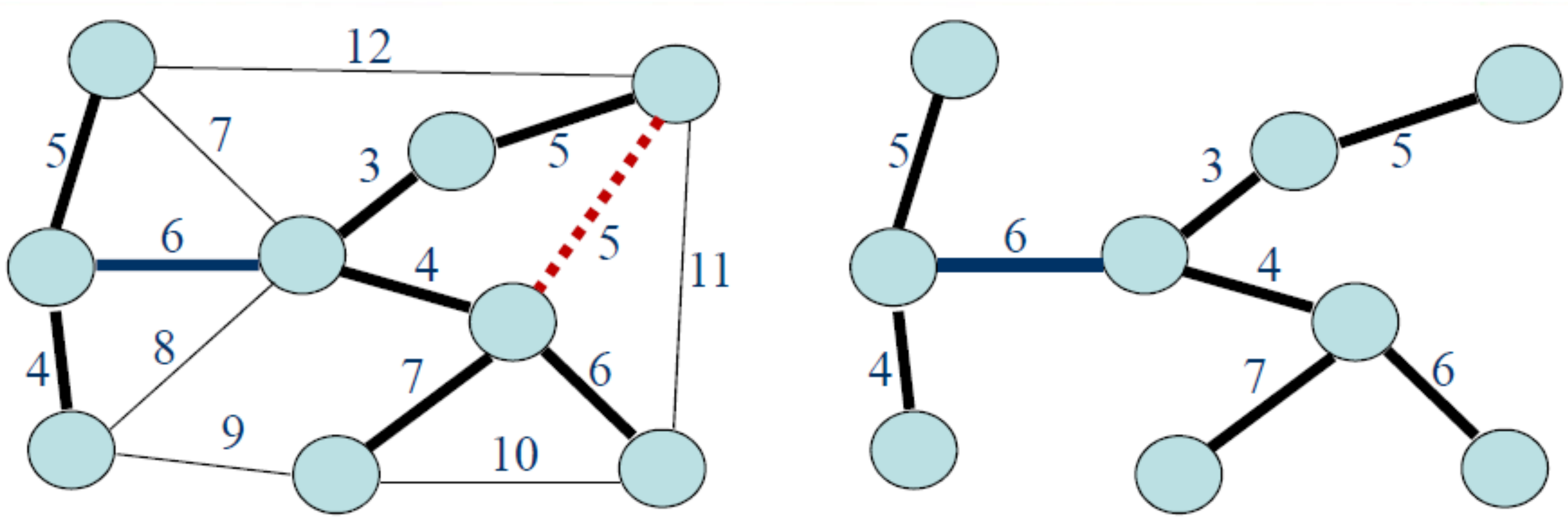
Kruskal example



Kruskal example



Kruskal example



Algoritmes Greedy

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

Algoritmes Greedy

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

- **Makeset**: Construir un nou conjunt a partir d'un simple node. Temps constant

After makeset(A), makeset(B), . . . , makeset(G):

Algoritmes Greedy

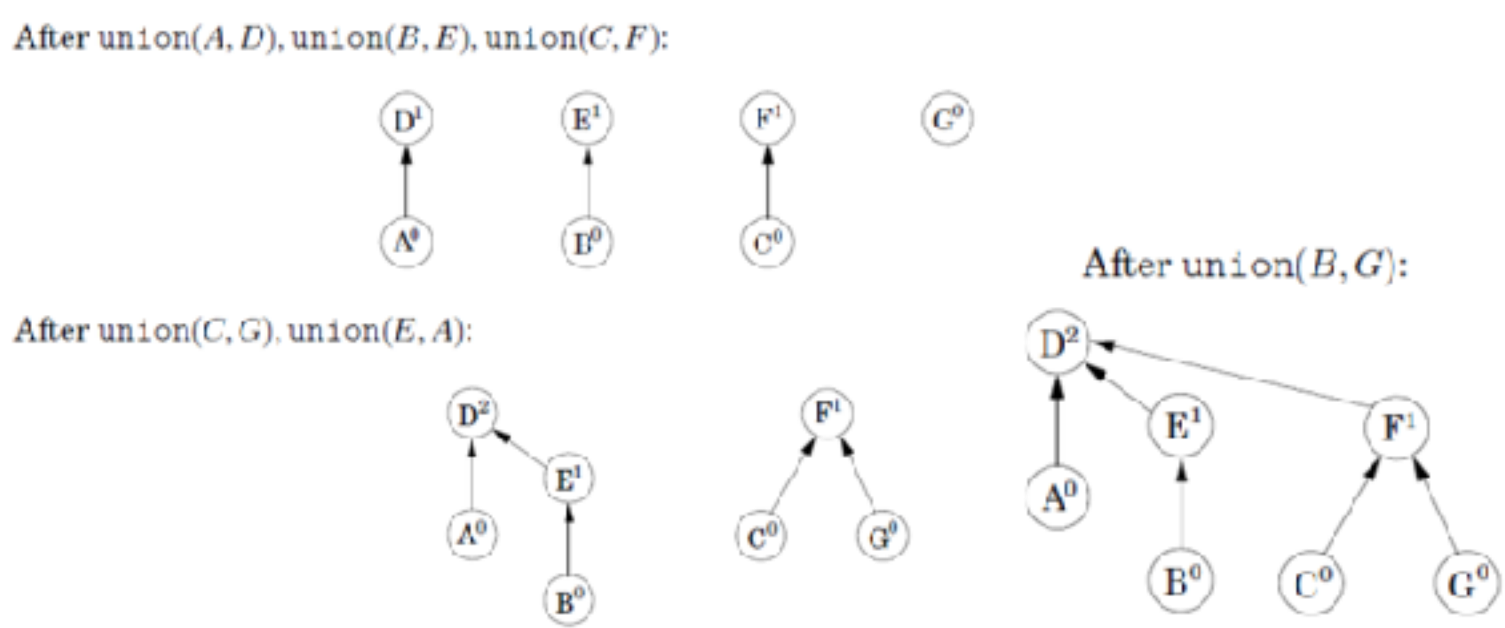
```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

- **Find-Set(u)** : Busca el conjunt que conté l'element **u.**

Algoritmes Greedy

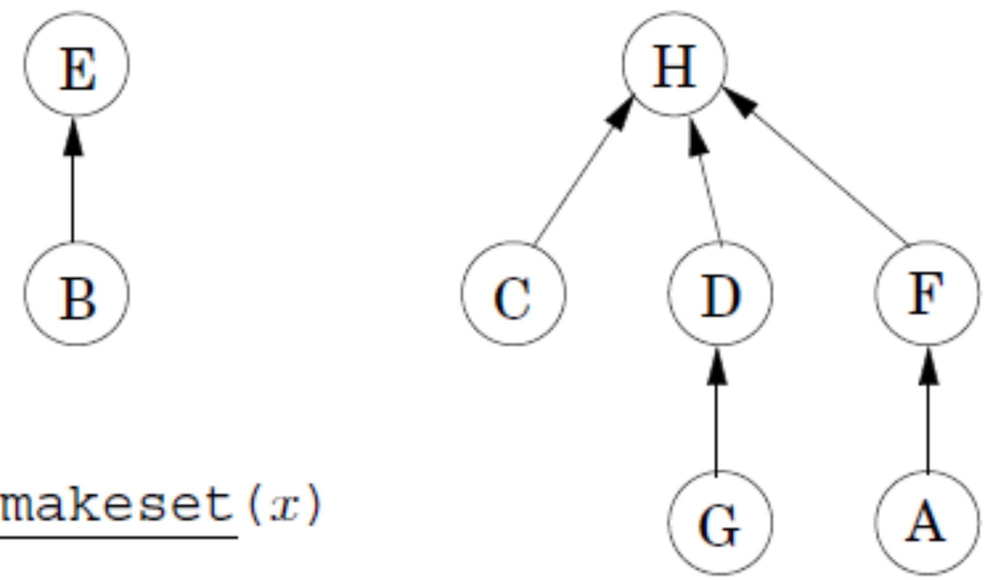
```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

- **Union(S1, S2)** : Crea un nou conjunt amb l'unió dels conjunts S1 i S2.



Algoritmes Greedy

- Representació dels conjunts: arbres dirigits




```
procedure makeset(x)
π(x) = x
rank(x) = 0

function find(x)
while x ≠ π(x):  x = π(x)
return x
```

punter

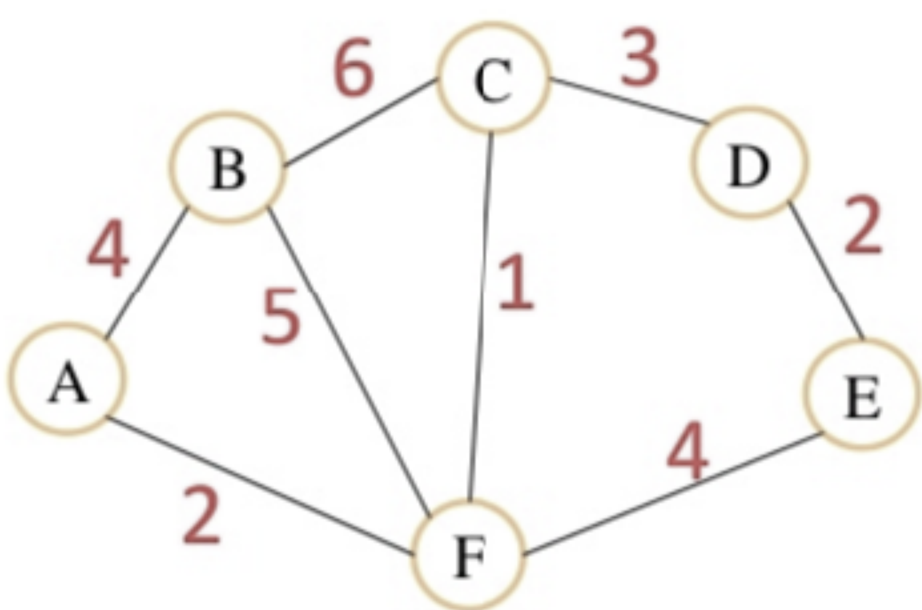
rank: altura dins de l'arbre

Algoritmes Greedy

- **Makeset**: temps constant
- **Find**: segueix punters dels pares als roots, per tant el temps és proporcional a l'altura
- **Union**: com l'altura ens defineix la complexitat, posem el punter de l'arbre més curt apuntant al punter de l'arbre amb més altura

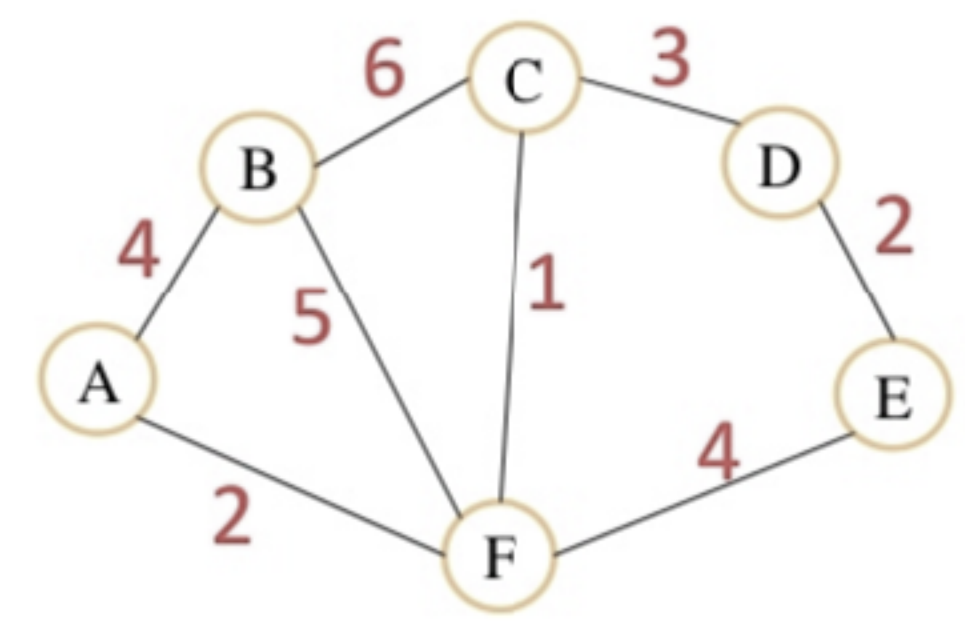
$\underline{\texttt{procedure union}}(x, y)$
$r_x = \texttt{find}(x)$
$r_y = \texttt{find}(y)$
$\texttt{if } r_x = r_y\texttt{:} \quad \texttt{return}$
$\texttt{if rank}(r_x) > \texttt{rank}(r_y)\texttt{:}$
$\quad \pi(r_y) = r_x$
$\texttt{else:}$
$\quad \pi(r_x) = r_y$
$\quad \texttt{if rank}(r_x) = \texttt{rank}(r_y)\texttt{:} \quad \texttt{rank}(r_y) = \texttt{rank}(r_y) + 1$

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```



Edges	Weight
AB	4
BC	6
CD	3
DE	2
EF	4
AF	2
BF	5
CF	1

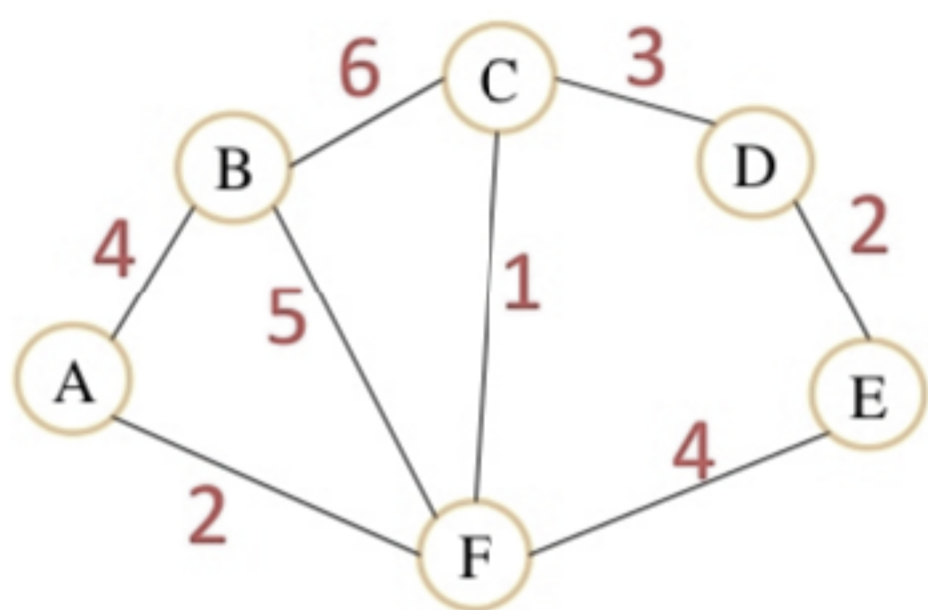
```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A ∪ {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```



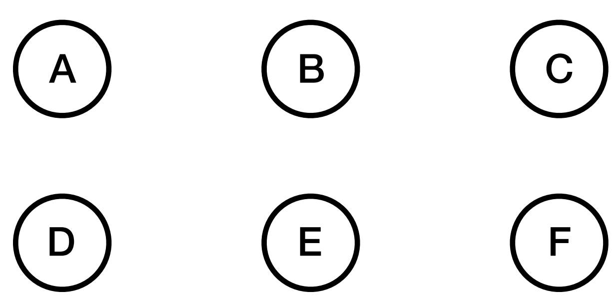
A = {}

Edges	Weight
AB	4
BC	6
CD	3
DE	2
EF	4
AF	2
BF	5
CF	1

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
    MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
    if FIND-SET(u) ≠ FIND-SET(v):
        A = A U {(u, v)}
        UNION(FIND-SET(u), FIND-SET(v))
return A
```

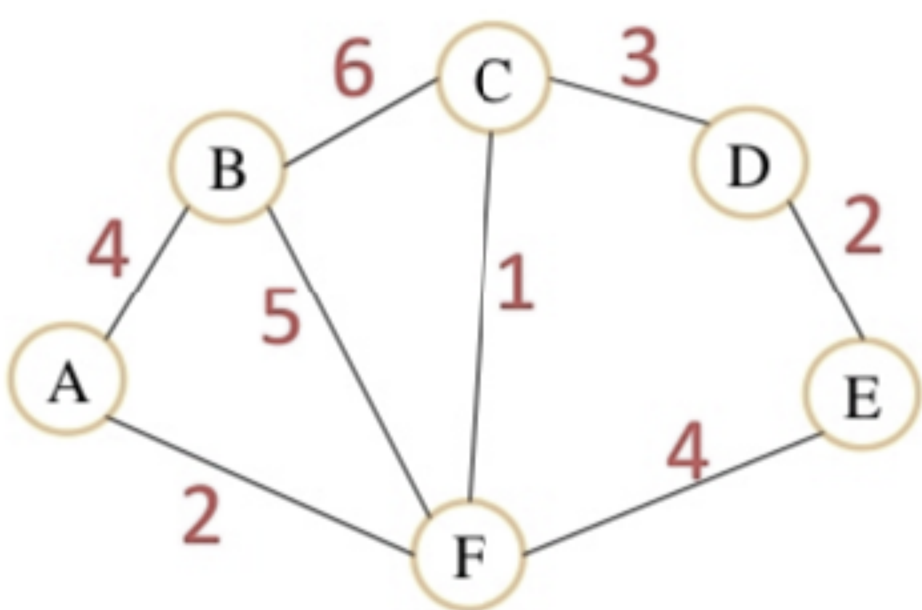


A = {}

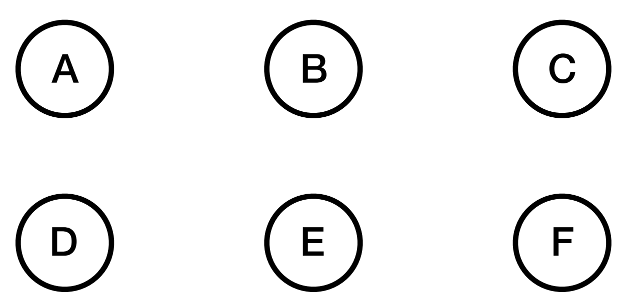


Edges	Weight
AB	4
BC	6
CD	3
DE	2
EF	4
AF	2
BF	5
CF	1

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

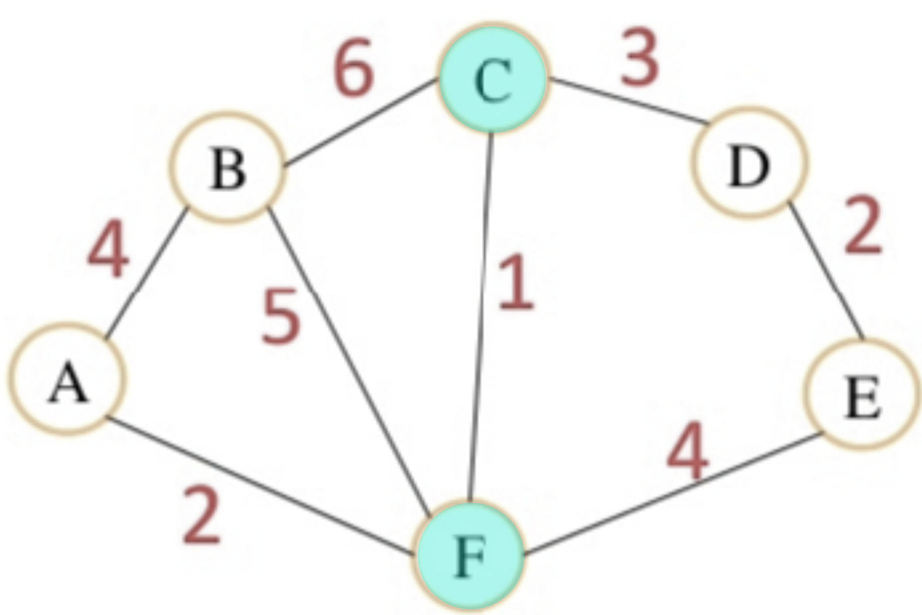


A = {}

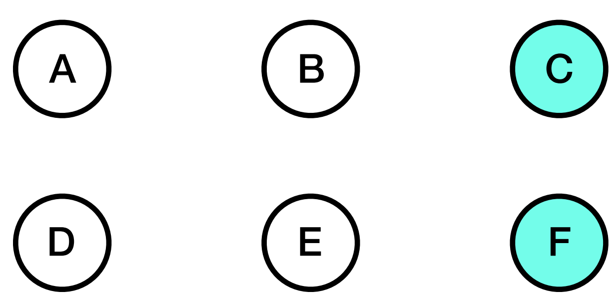


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

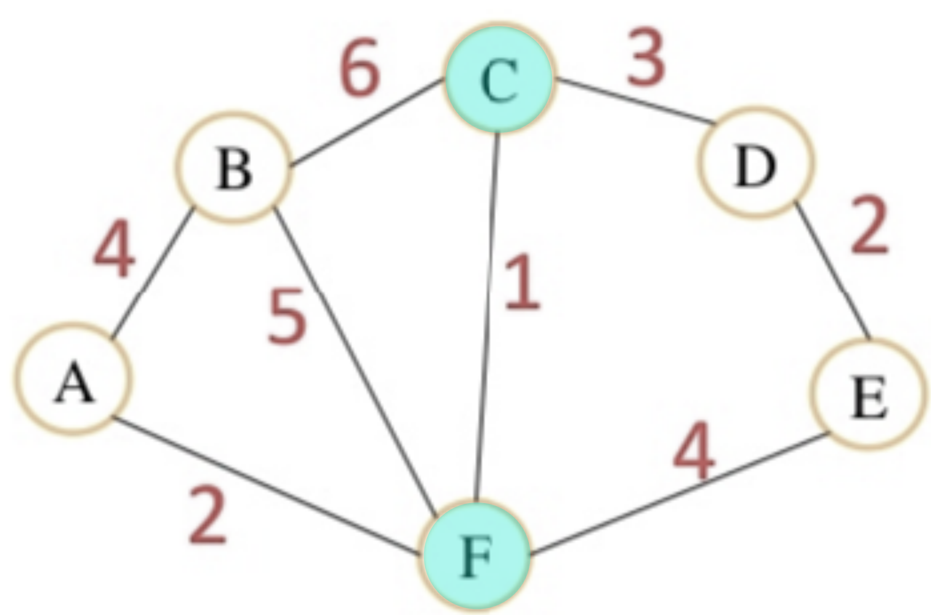


A = {}

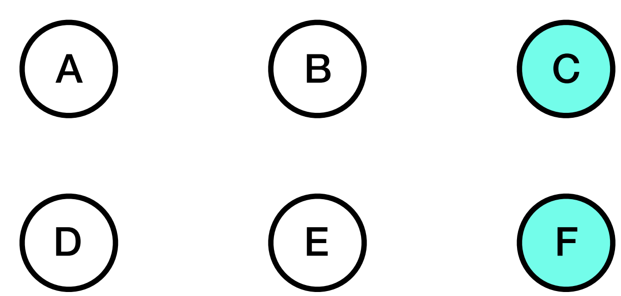


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

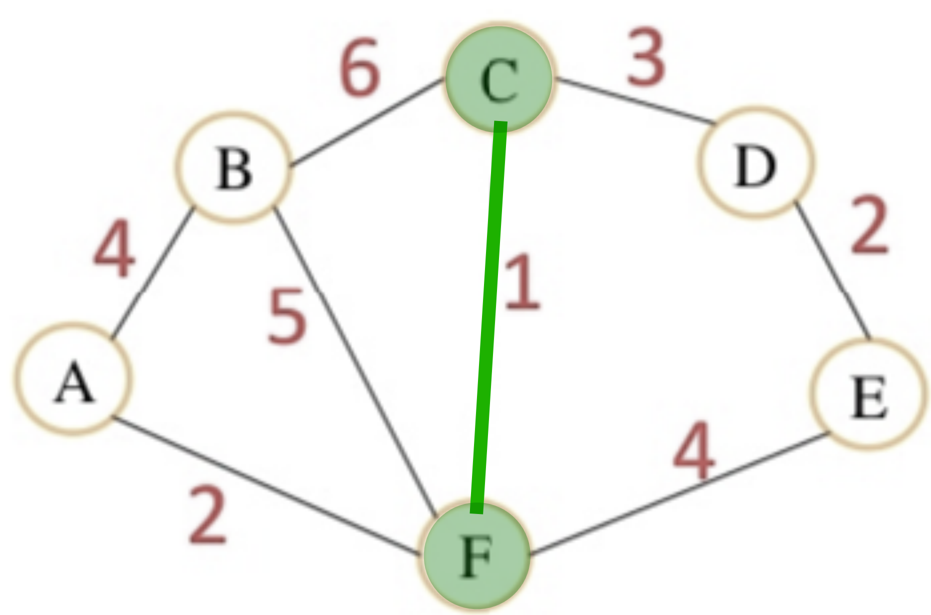


A = {(C,F)}

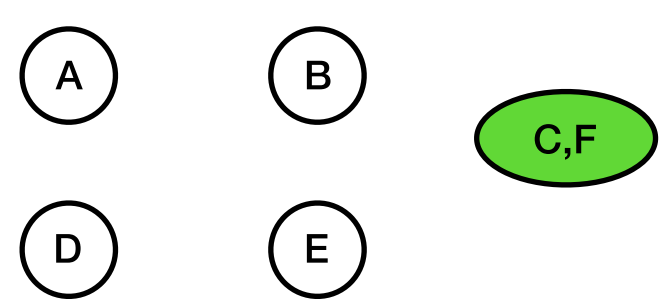


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A ∪ {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

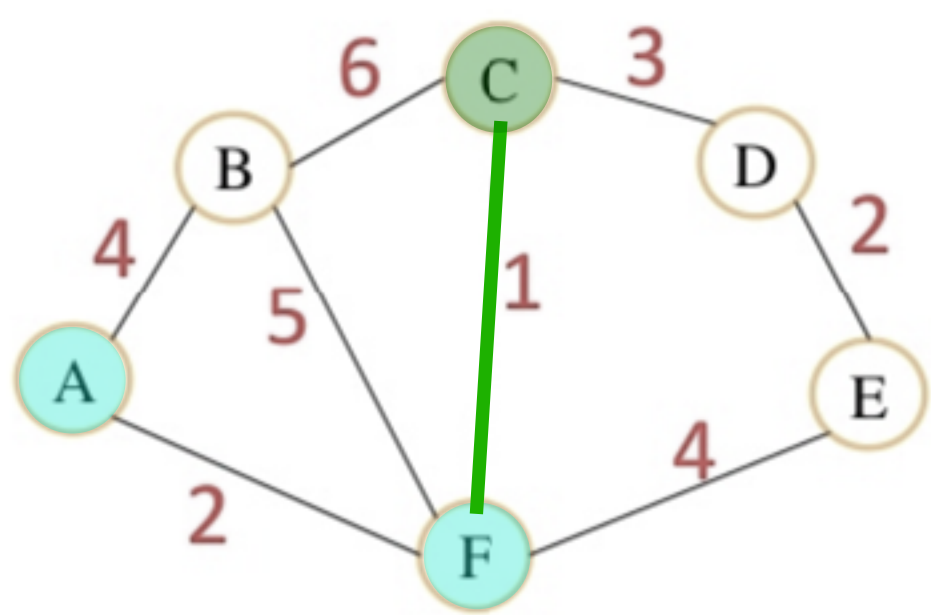


A = {(C,F)}

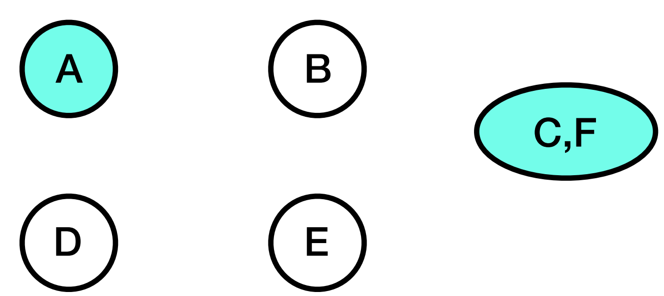


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

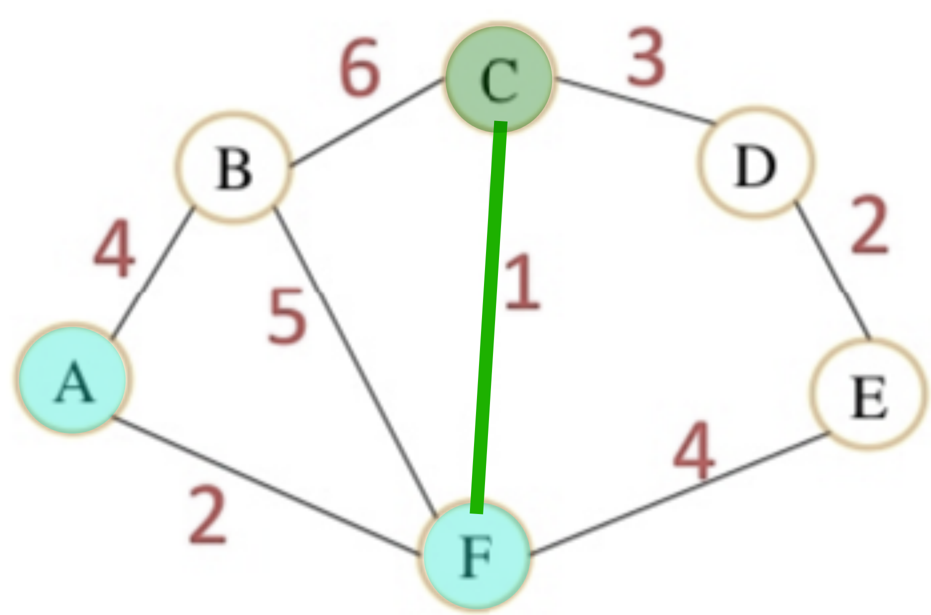


A = {(C,F)}

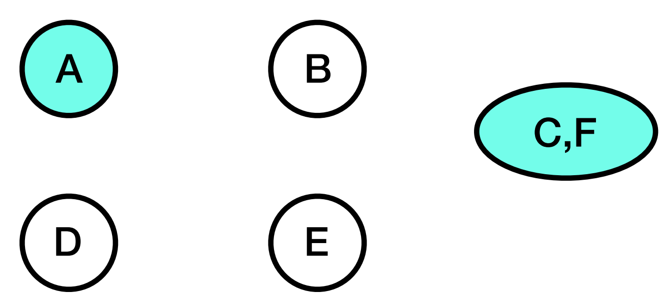


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A ∪ {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

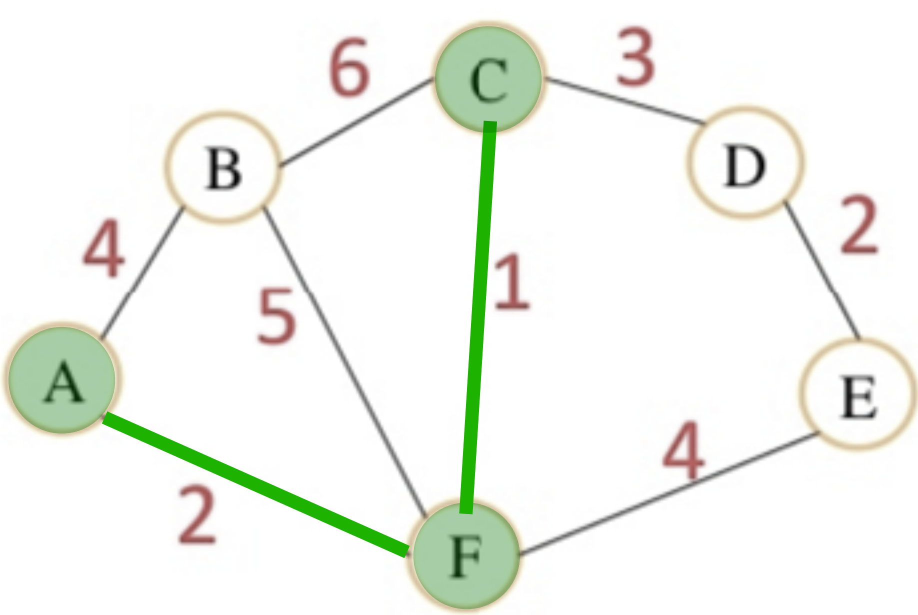


A = {(C,F),(A,F)}

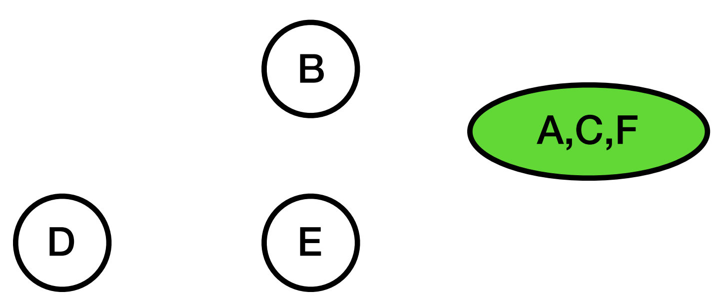


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

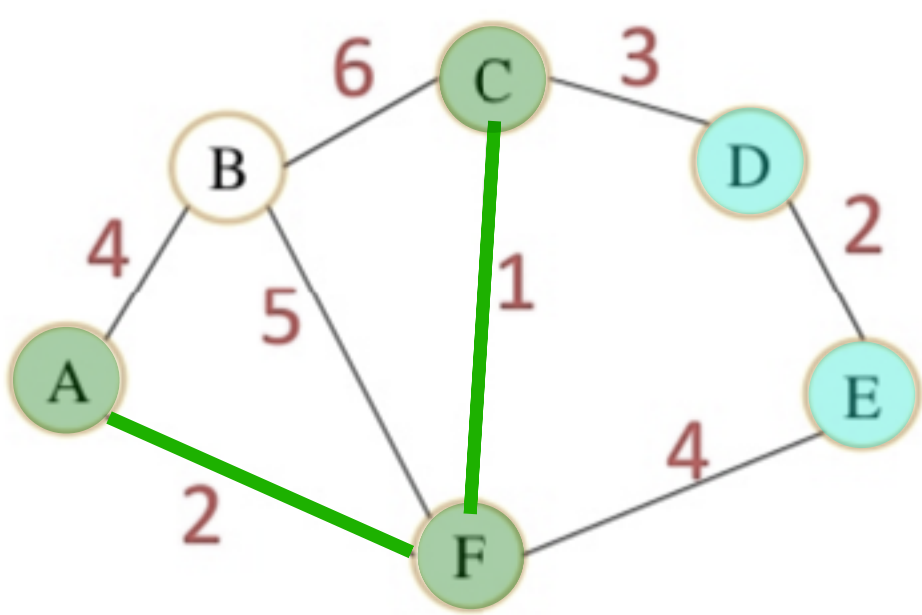


A = {(C,F),(A,F)}

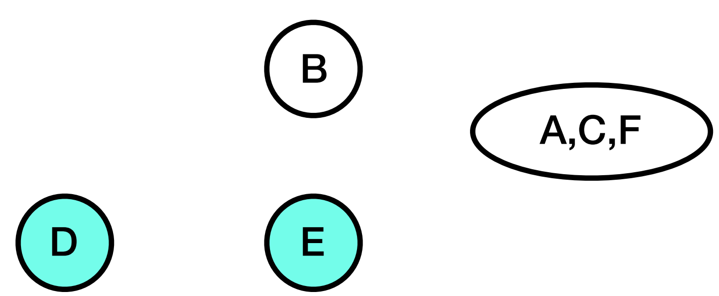


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

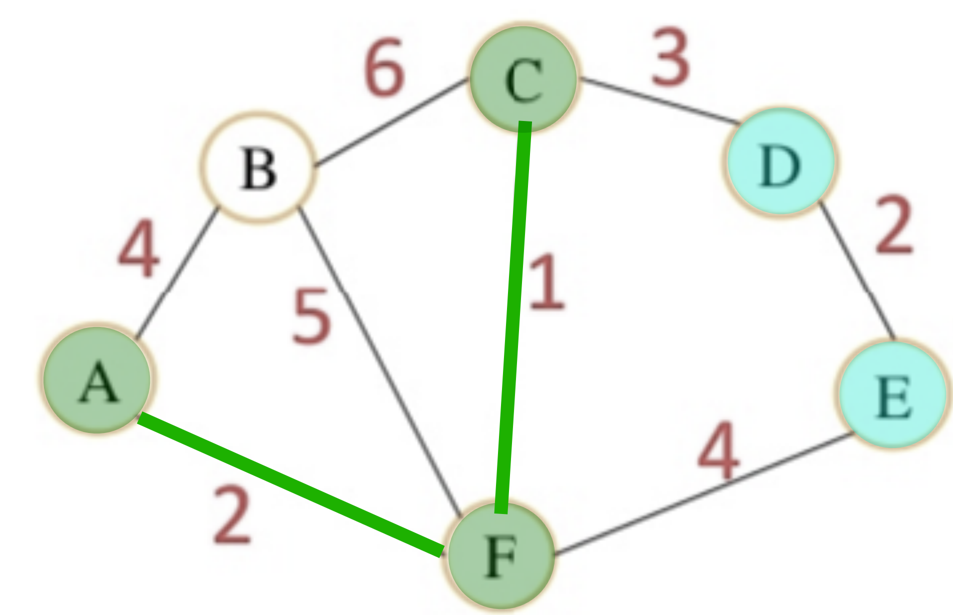


A = {(C,F),(A,F)}

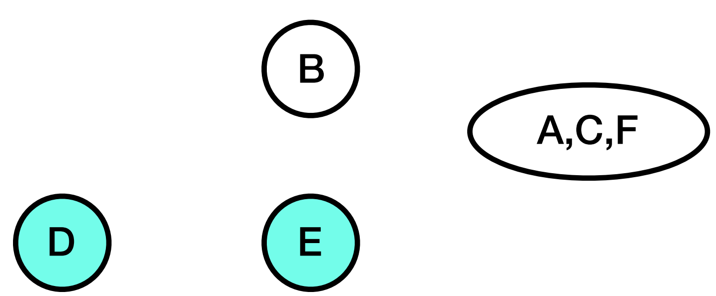


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
    MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
    if FIND-SET(u) ≠ FIND-SET(v):
        A = A U {(u, v)}
        UNION(FIND-SET(u), FIND-SET(v))
return A
```

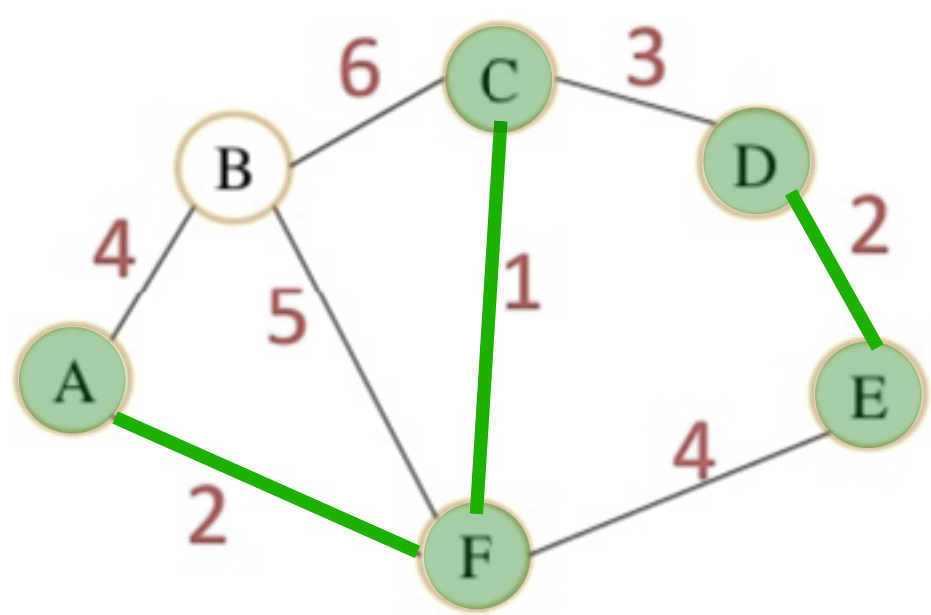


A = {(C,F),(A,F),(D,E)}

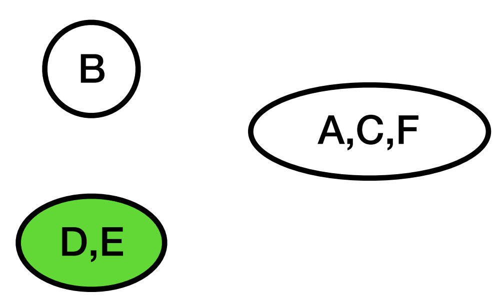


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

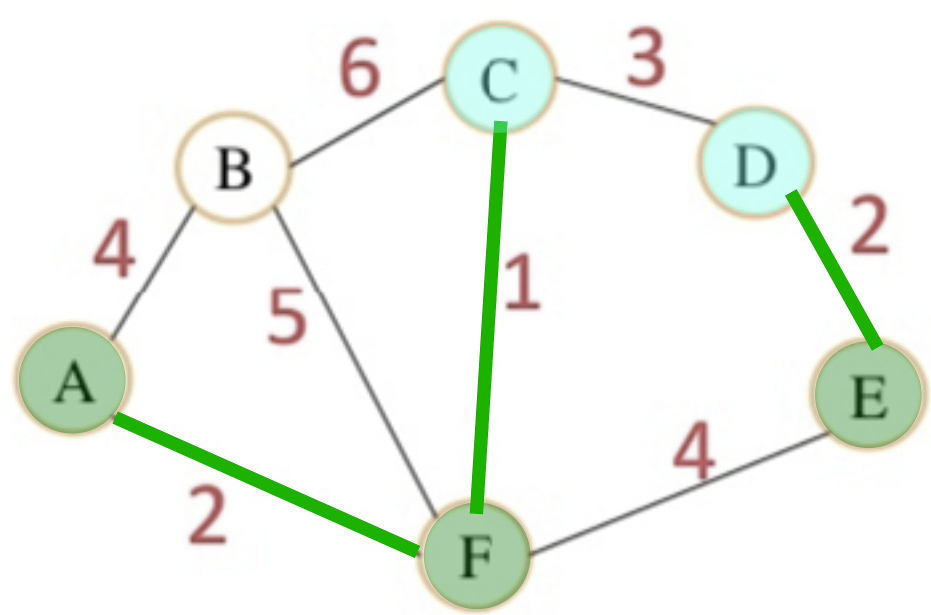


A = {(C,F),(A,F),(D,E)}

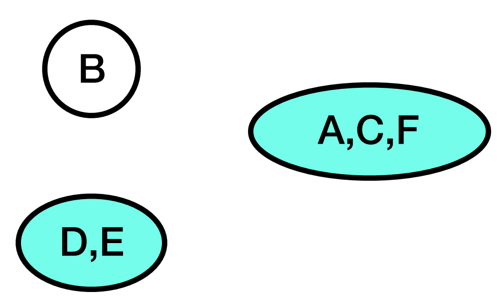


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

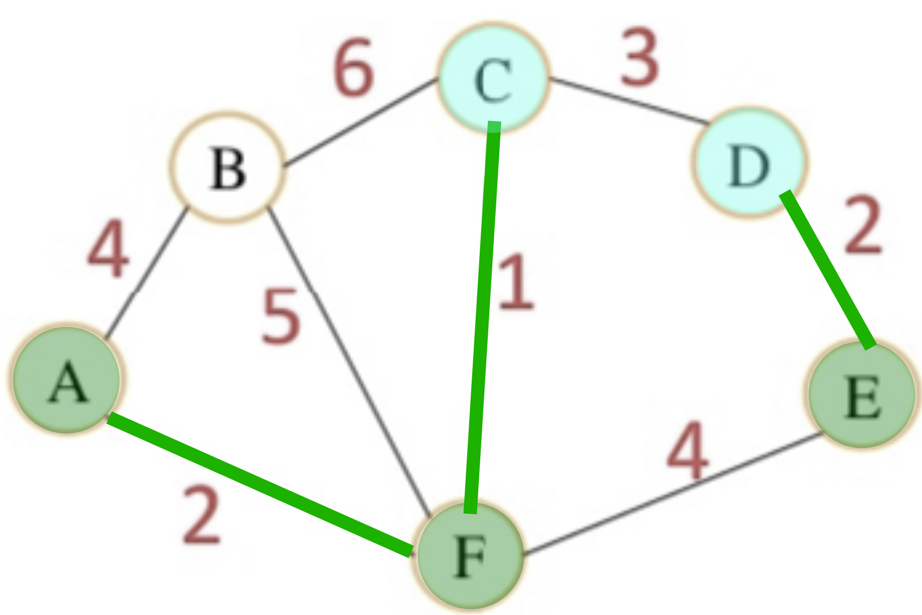


A = {(C,F),(A,F),(D,E)}



Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```



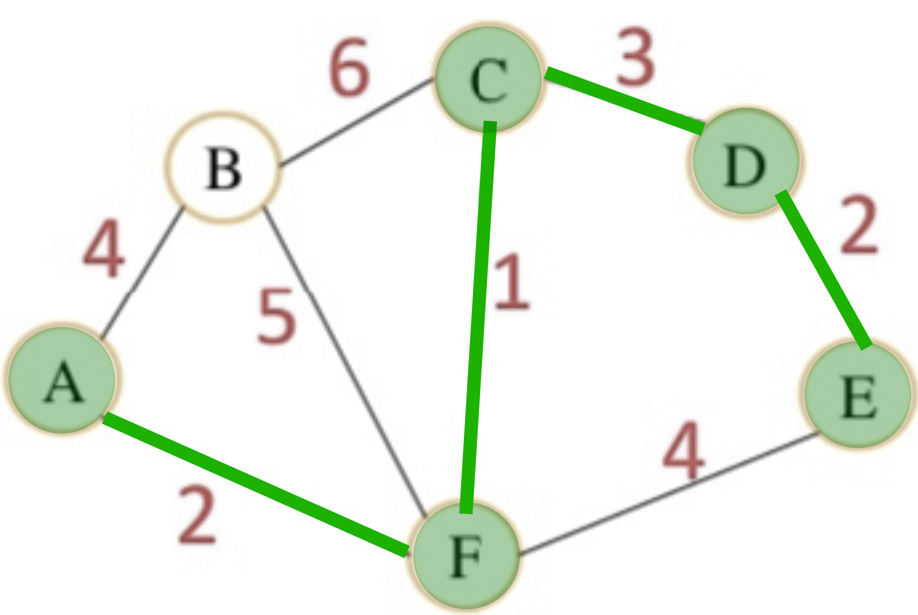
A = {(C,F),(A,F),(D,E),(C,D)}

B

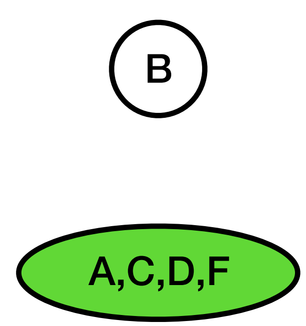
A,C,D,F

Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
    MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
    if FIND-SET(u) ≠ FIND-SET(v):
        A = A U {(u, v)}
        UNION(FIND-SET(u), FIND-SET(v))
return A
```



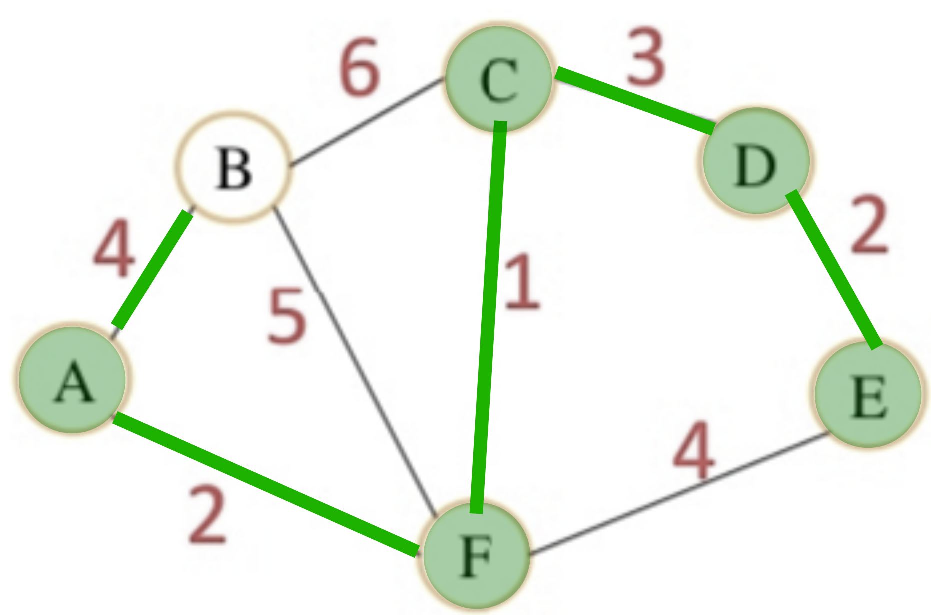
A = {(C,F),(A,F),(D,E),(C,D)}



Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

i continua amb la següent solució:

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```



A = {(C,F),(A,F),(D,E),(C,D),(A,B)}

A,B,C,D,F

Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

Complexitat Kruskal

```
KRUSKAL(G):
A = Ø
foreach v ∈ G.V:
    MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
    if FIND-SET(u) ≠ FIND-SET(v):
        A = A U {(u, v)}
        UNION(FIND-SET(u), FIND-SET(v))
return A
```

?

?

?

Complexitat Kruskal

```
KRUSKAL(G):
A = Ø
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

O(V)

O(ELogE)

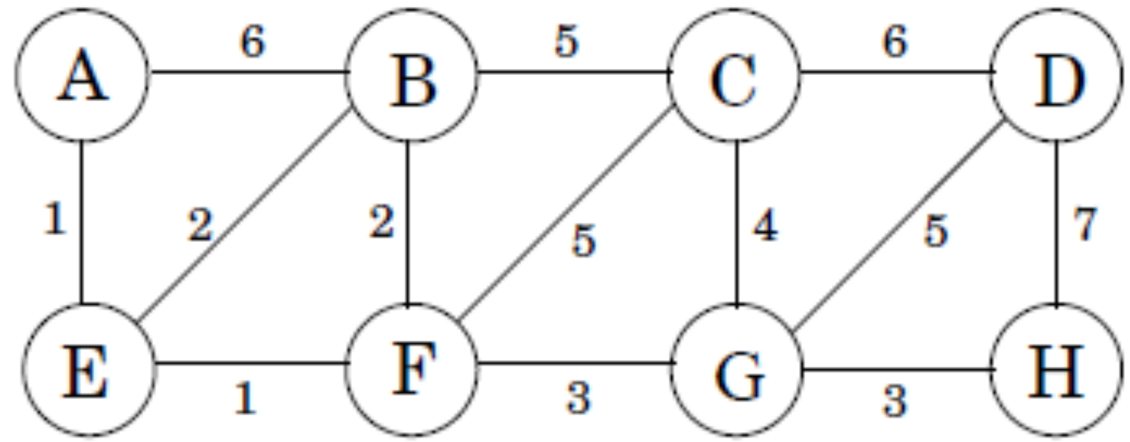
O(ELovV)

Complexitat Kruskal

Complexitat = O(1) + O(V) + O(E logE) + O(E LogV)
= O(E logE) + O(E logV)
= O(E logE)

Exercici: MST i Kruskal

- Exercicis (1):



  - A) Quin és el cost del MST?
  - B) En quin ordre les arestes són incloses en el MST usant l'algorisme Kruskal?

Prim

Algoritmes greedy

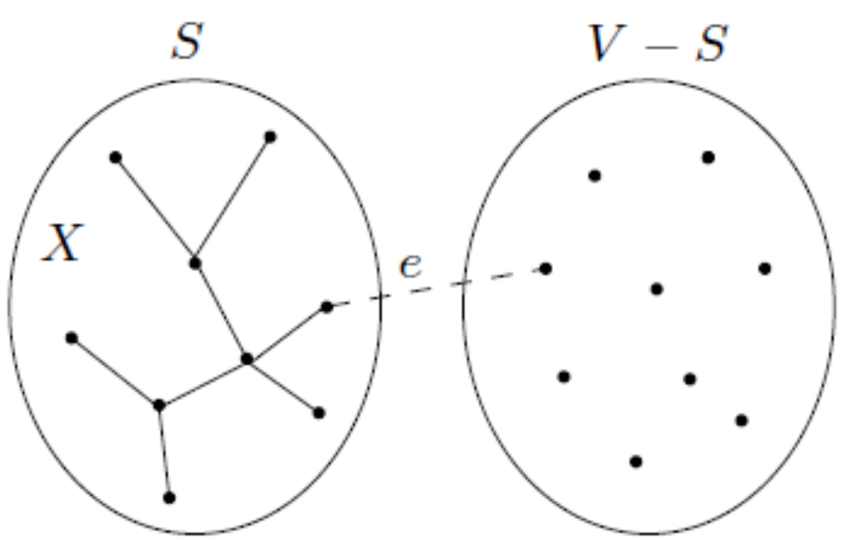
- Exemple: Algorisme de **Prim**
  - **Alternativa a Kruskal**
- La propietat de tall ens diu que qualsevol algorisme que segueix el següent procediment hauria de funcionar :

$X = \{\ \}$ (edges picked so far)
repeat until $|X| = |V| - 1$:
  pick a set $S \subset V$ for which X has no edges between S and $V - S$
  let $e \in E$ be the minimum-weight edge between S and $V - S$
  $X = X \cup \{e\}$

Algoritmes greedy

- Algoritme de **Prim**

$$\texttt{cost}(v) \;=\; \min_{u\in S}\; w(u,v).$$



La complexitat és similar a l'algorisme de kruskal

Algoritmes greedy

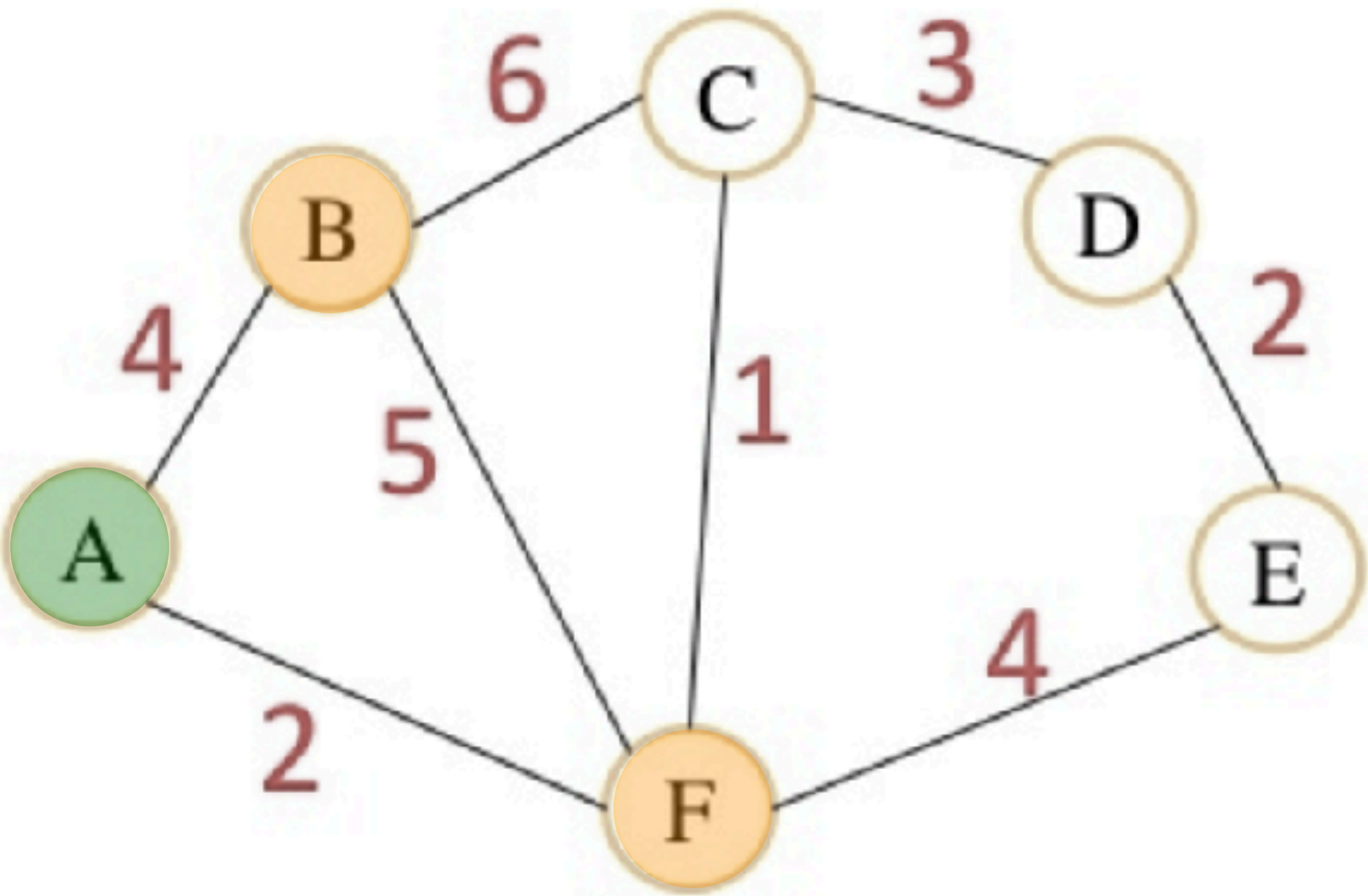
- Algorisme de **Prim**

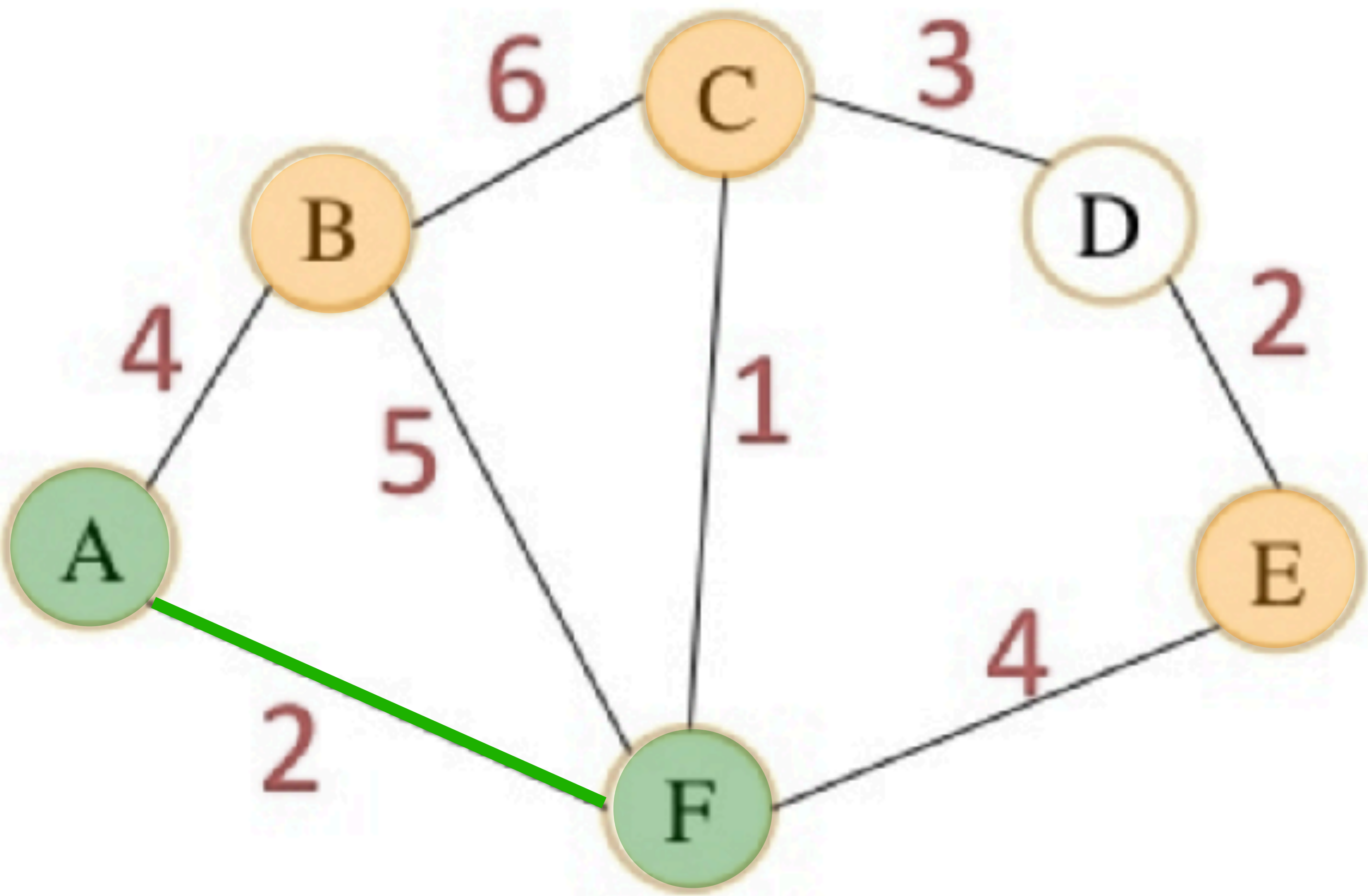
```
procedure prim(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the array prev

for all u ∈ V:
   cost(u) = ∞
   prev(u) = nil
Pick any initial node u_0
cost(u_0) = 0

H = makequeue(V)    (priority queue, using cost-values as keys)
while H is not empty:
   v = deletemin(H)
   for each {v, z} ∈ E:
      if cost(z) > w(v, z):
         cost(z) = w(v, z)
         prev(z) = v
         decreasekey(H, z)
```

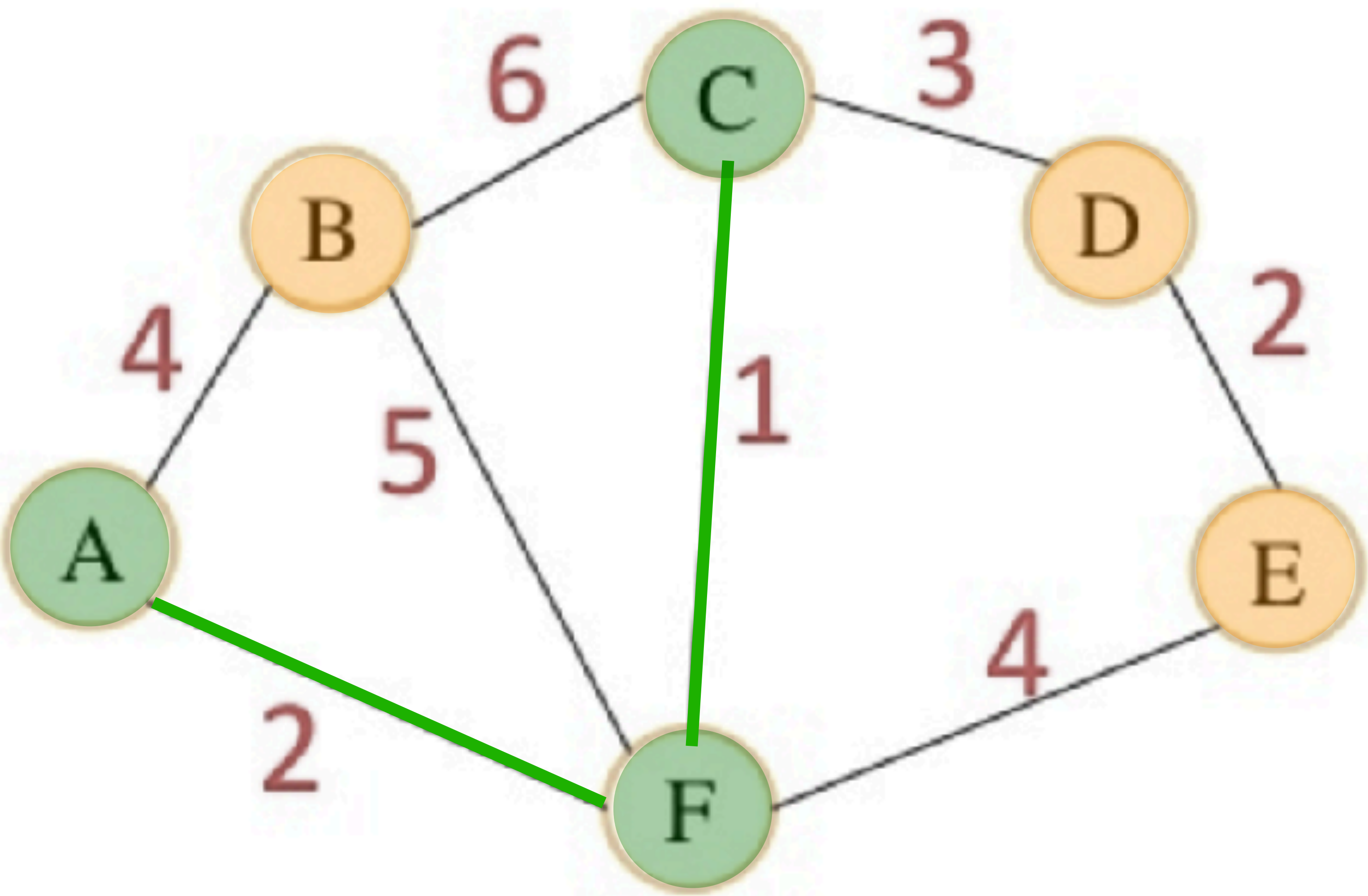
Algoritmes greedy

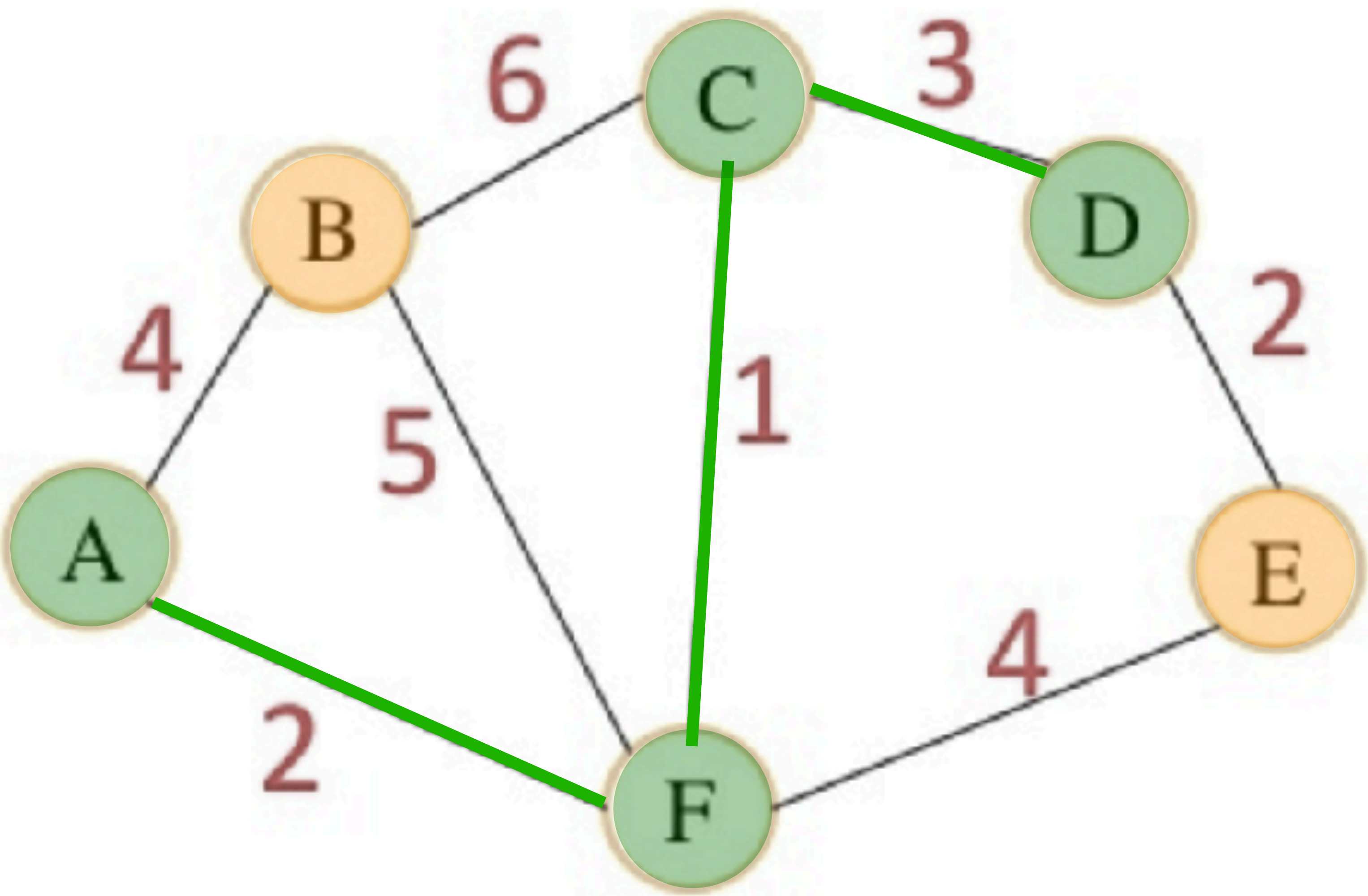
- Algorisme de **Prim**

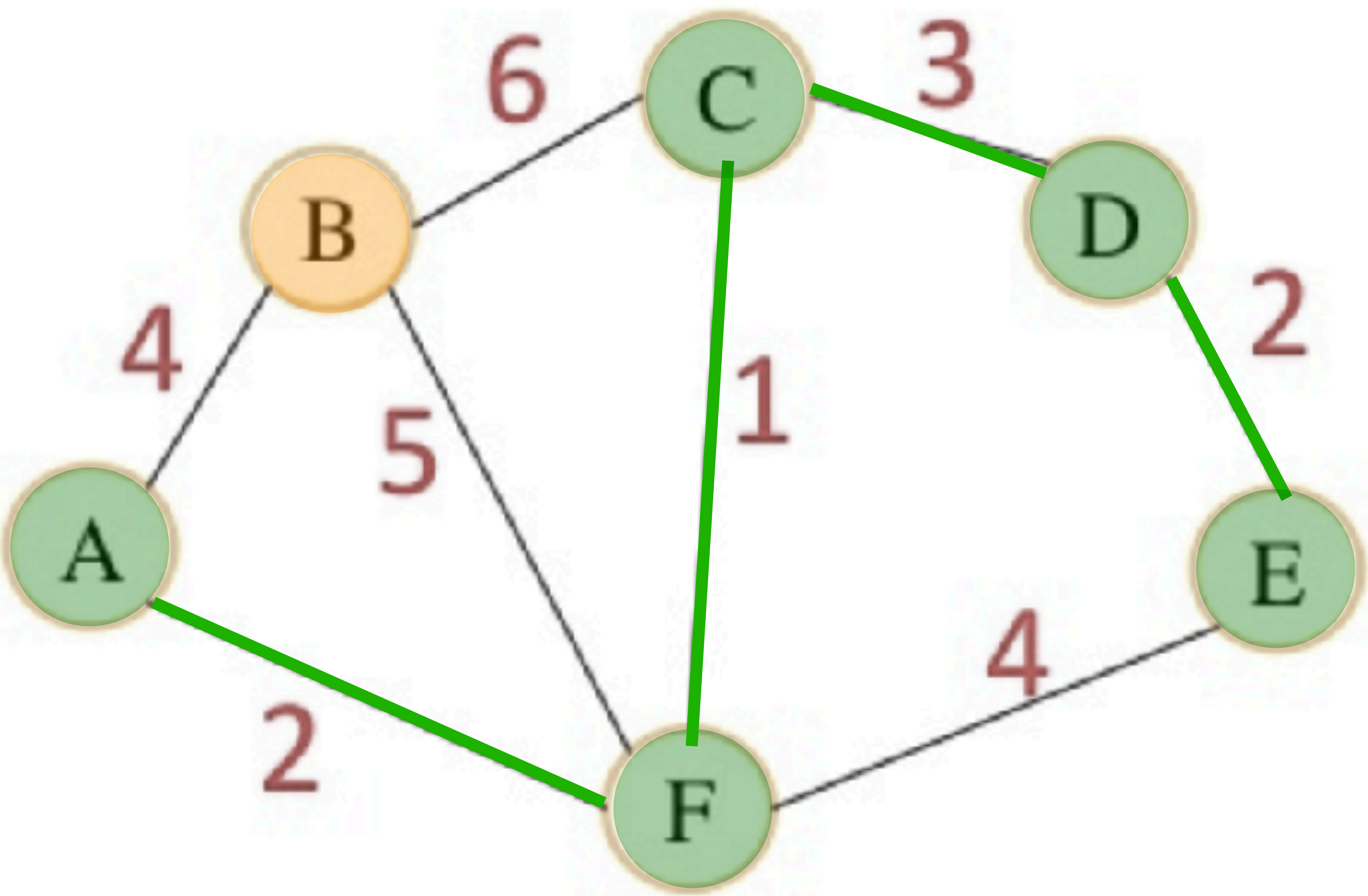
Set S	A	B	C	D	E	F
$\{\}$	$0/\text{nil}$	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
A		$5/A$	$6/A$	$4/A$	∞/nil	∞/nil
A, D		$2/D$	$2/D$		∞/nil	$4/D$
A, D, B			$1/B$		∞/nil	$4/D$
A, D, B, C					$5/C$	$3/C$
A, D, B, C, F					$4/F$	


A
B
C
D
E
F
4
6
3
5
1
2
2
4

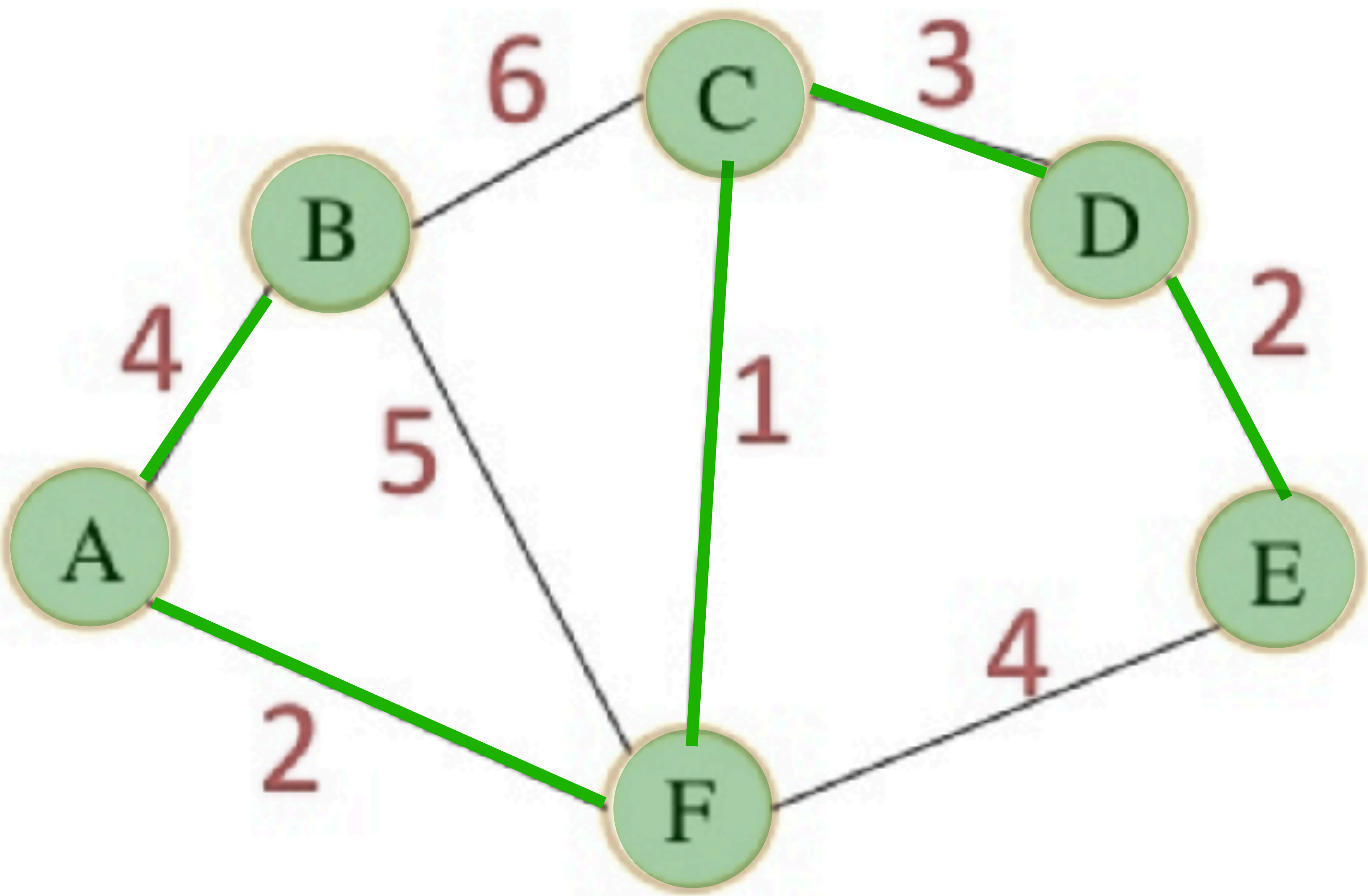

A
B
C
D
E
F
6
3
4
5
1
2
2
4


6
C
3
B
D
4
2
1
5
A
E
4
2
F

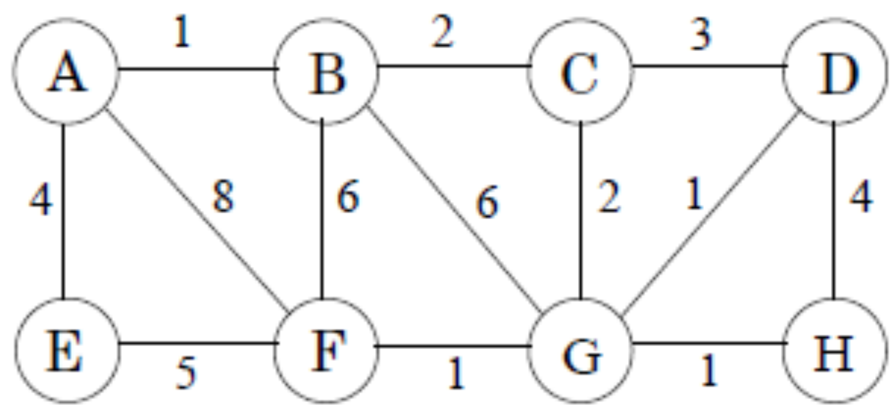

A
B
C
D
E
F
6
3
4
5
1
2
2
4



6
3
B
C
D
4
5
1
2
A
E
4
2
F


A
B
C
D
E
F
4
6
3
5
1
2
2
4

Exercici: PRIM



- Aplica l'algorisme Prim (order alfabètic)
    - Escriu la taula de costos intermedis
- Aplica l'algorisme Kruskal i mostra els diferents arbres intermedis.

Algoritmes greedy

- En altres casos, els algorismes greedy obtenen respostes aproximades
  - → factor d'aproximació
- No són òptimes, però no existeixen algorismes lineals que solucionen el problema
  - → Ho veurem a problemes NP