UNIVERSITAT DE BARCELONA

Greedy

Algorísmica Avançada | Enginyeria Informàtica

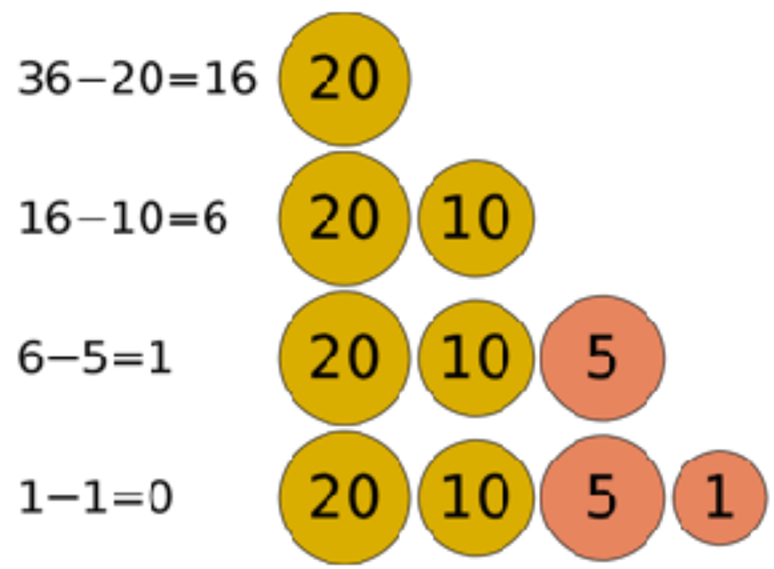
Santi Seguí | 2019-2020

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}

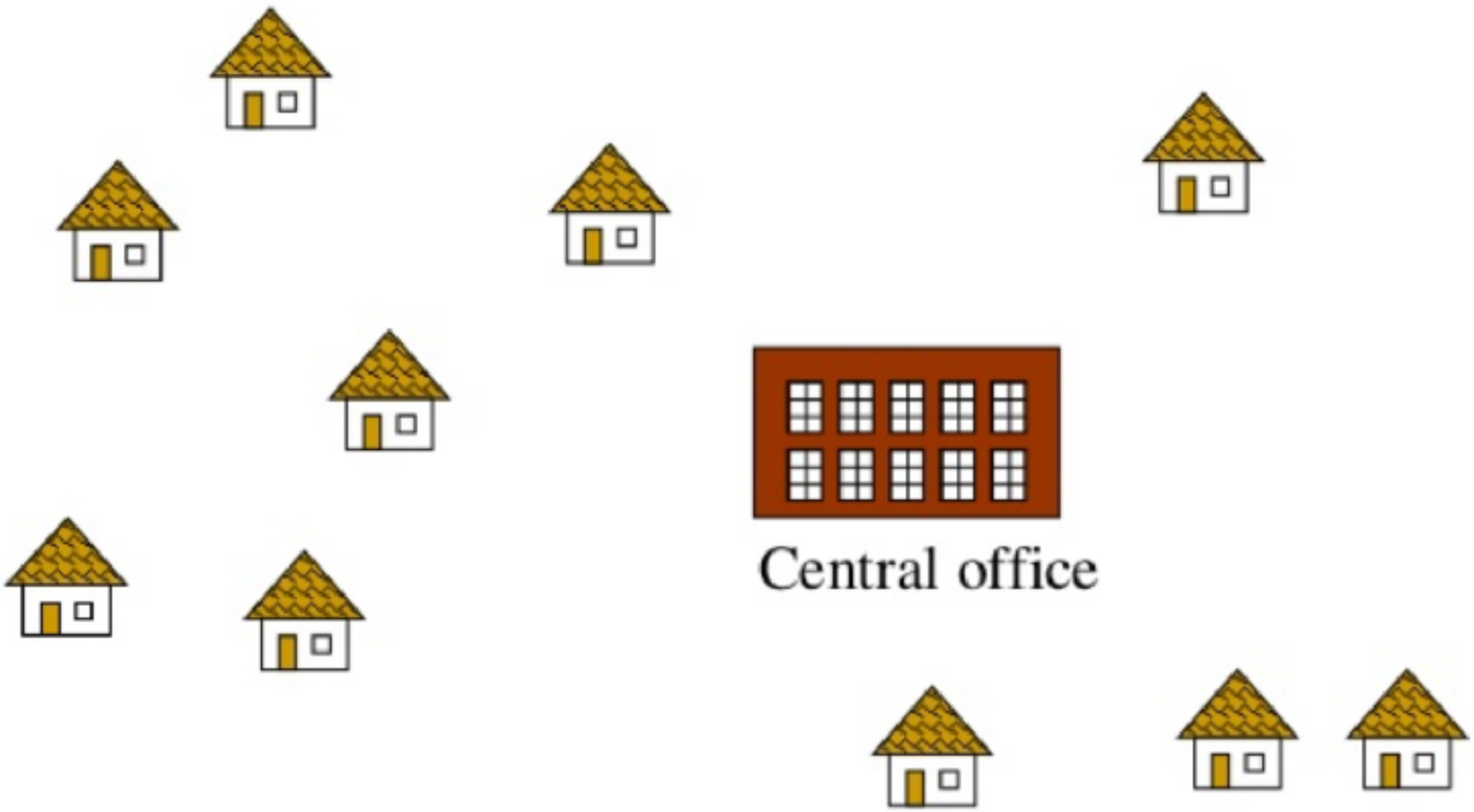


Els algorismes voraços **no garanteixen** sempre **una solució**, **ni** que la **solució** obtinguda sigui **l'òptima**

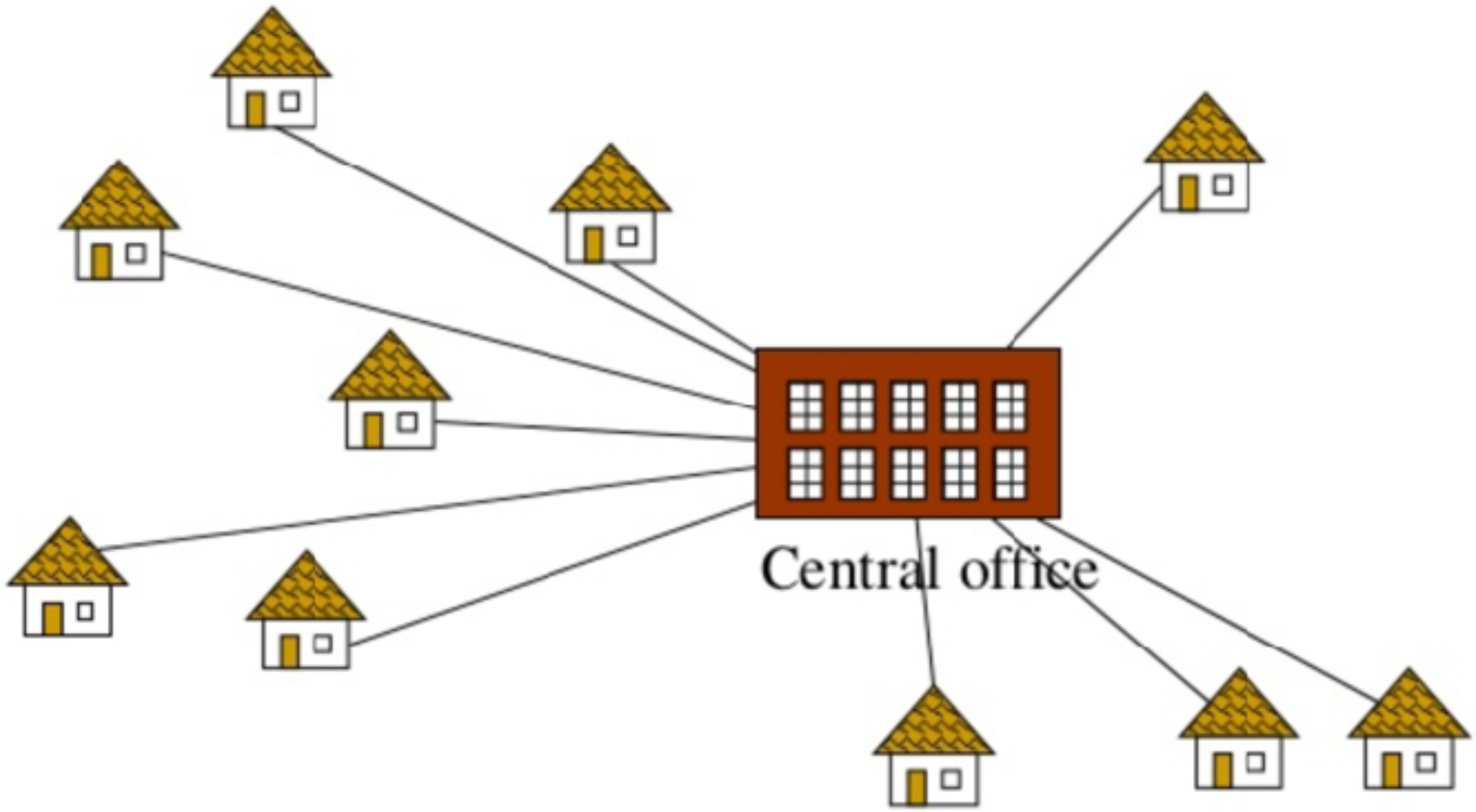
Greedy

- 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

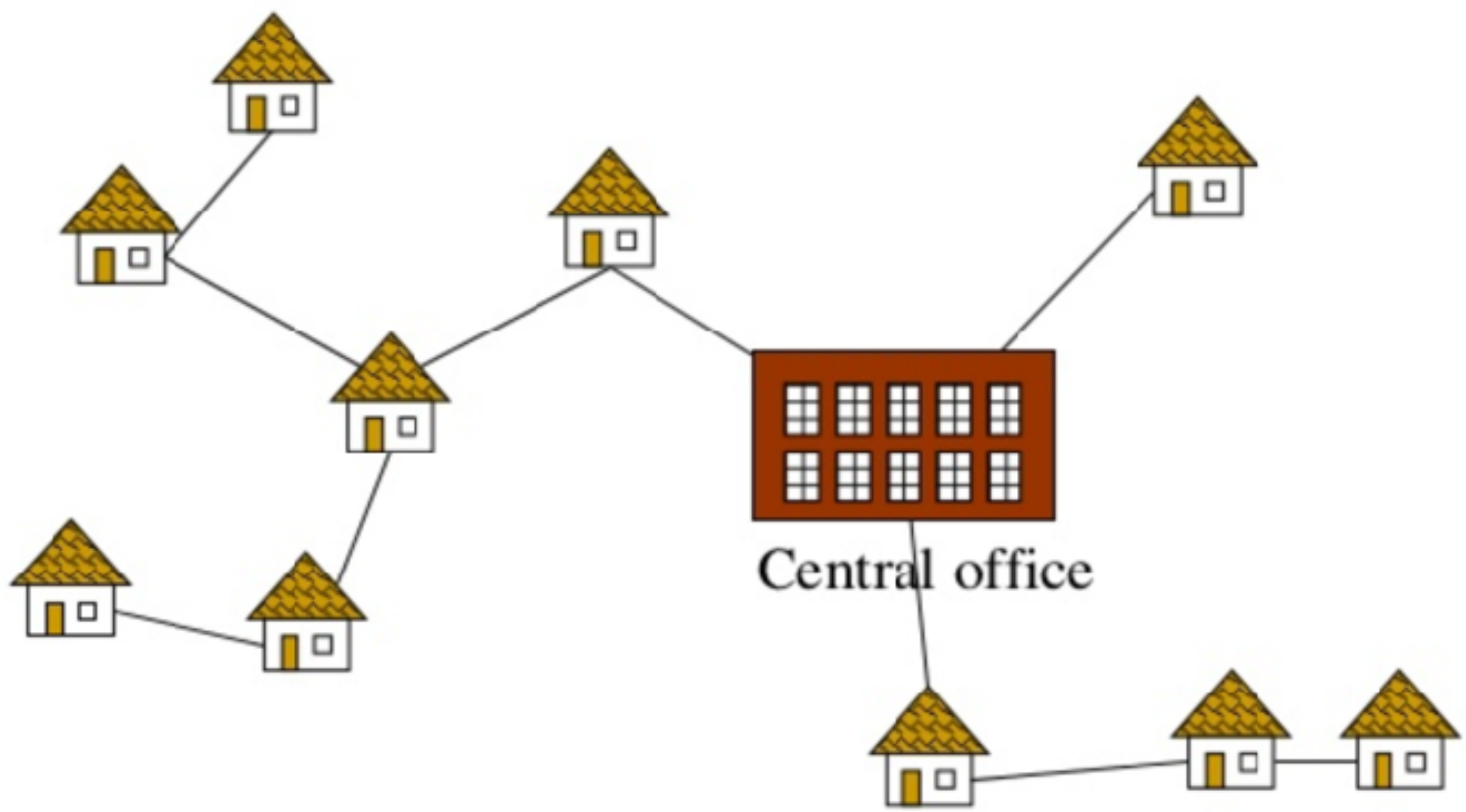


Wiring: Naïve Approach



Expensive!

Wiring: Better Approach



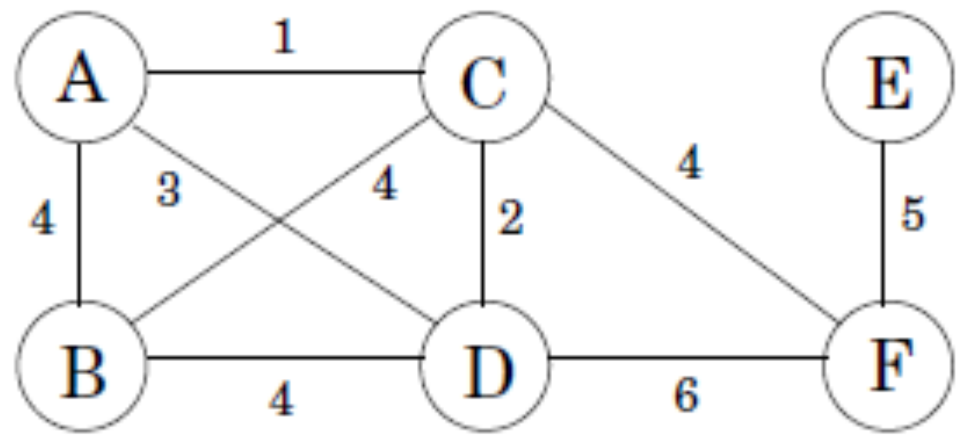
Minimize the total length of wire connecting the customers

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

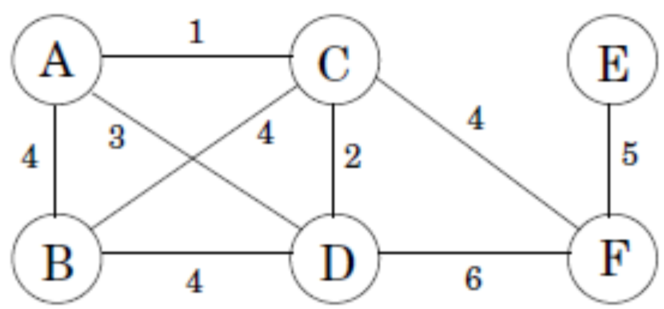


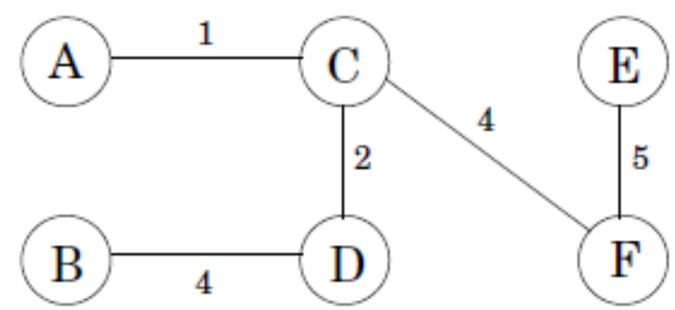
- Volem connectar els oficines (nodes) d'una empresa. Les connexions són les arestes. Cadascuna té un cost. Volem el mínim cost.
  - → llavors no volem cicles
  - → volem un graf no dirigit acíclic connectat
    - → arbre !!!
  - → de mínim cost: **Minimum Spanning Tree** (MST)

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?

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
```

$|V|$ makeset,
$2|E|$ find
$|V|-1$ union

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 $\texttt{makeset}(A), \texttt{makeset}(B), \ldots, \texttt{makeset}(G)$:

Algoritmes Greedy

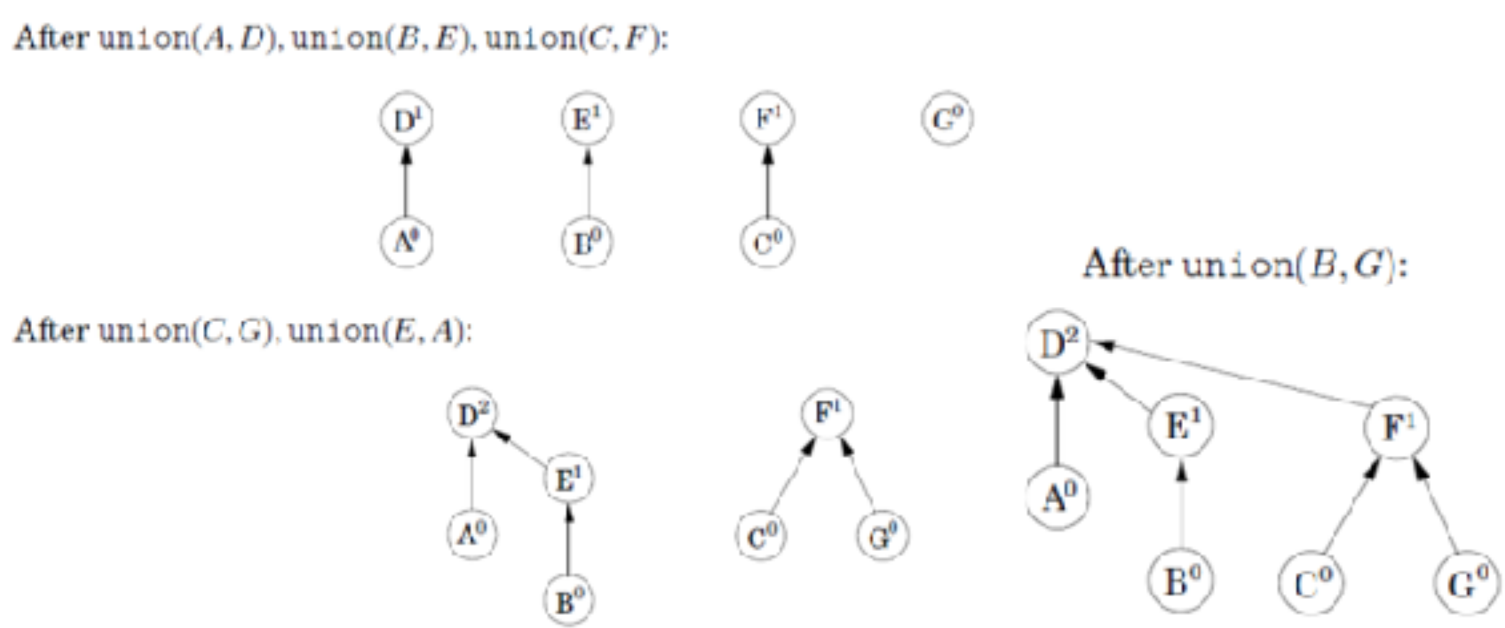
```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

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

Algoritmes Greedy

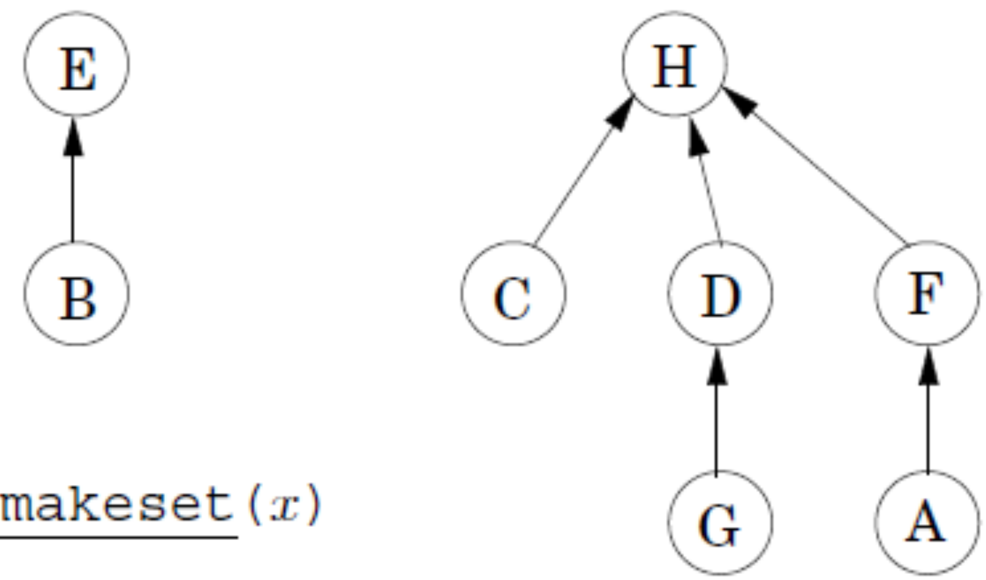
```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

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



Algoritmes Greedy

- Representació dels conjunts: arbres dirigits



$\underline{\texttt{procedure makeset}}(x)$
$\pi(x) = x$
$\texttt{rank}(x) = 0$

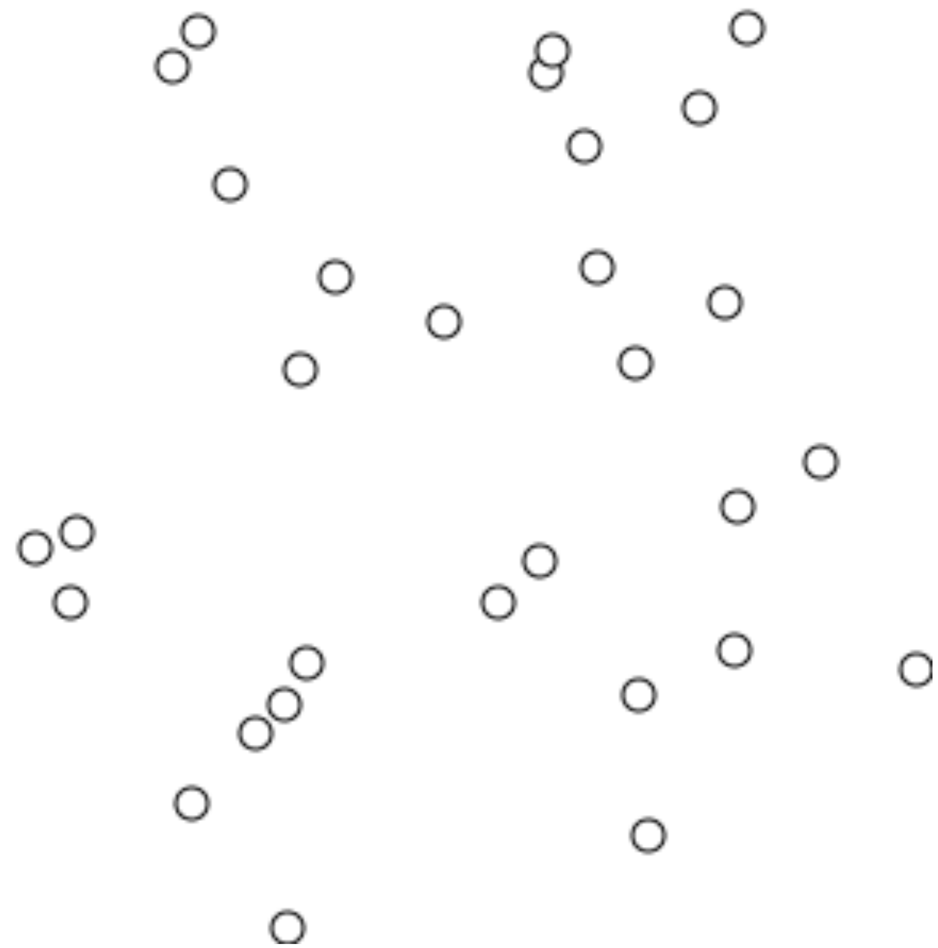
$\underline{\texttt{function find}}(x)$
$\texttt{while } x \neq \pi(x): \quad x = \pi(x)$
$\texttt{return } x$

punter
rank: altura dins de l'arbre

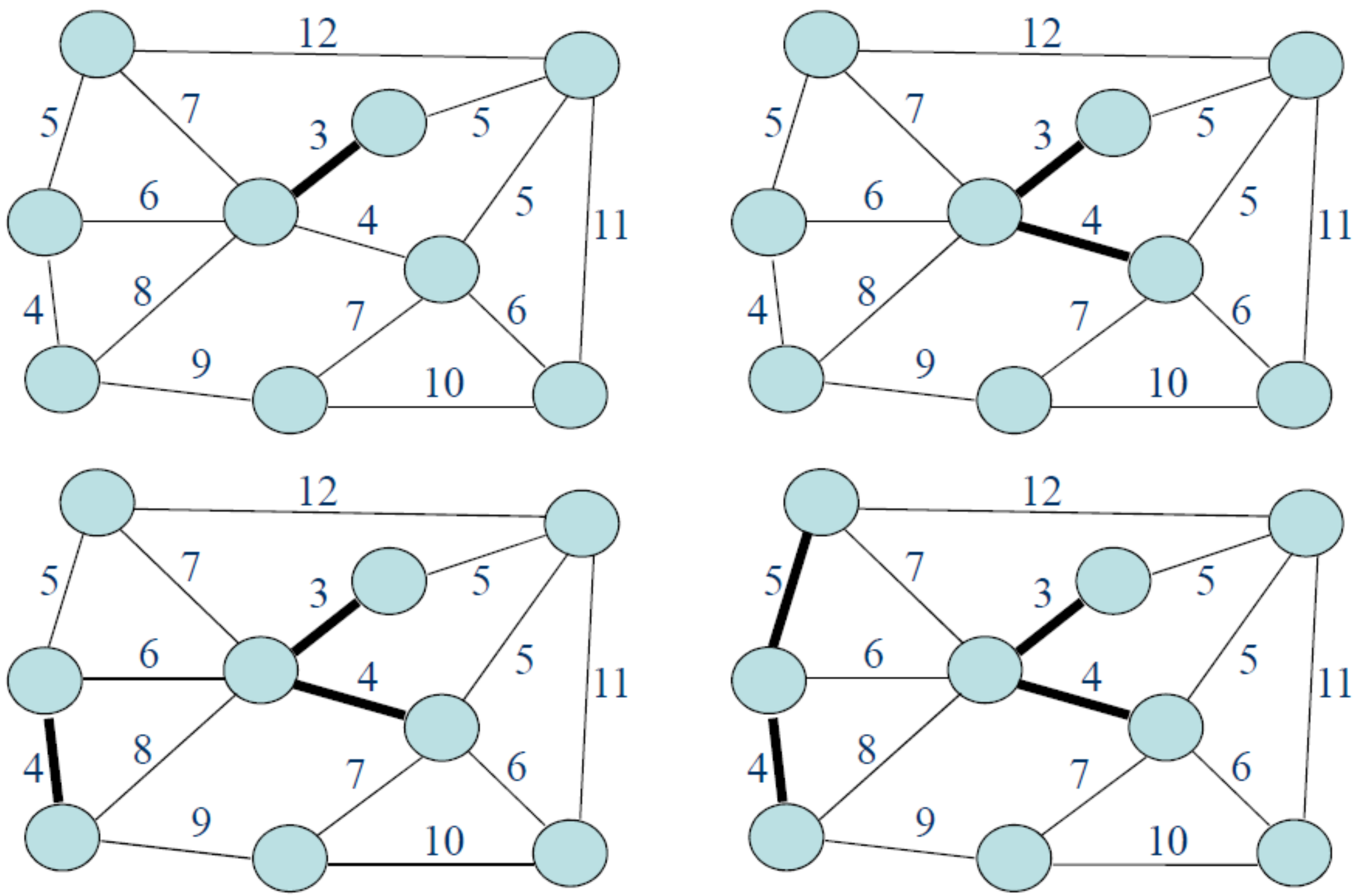
Algoritmes Greedy

- **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

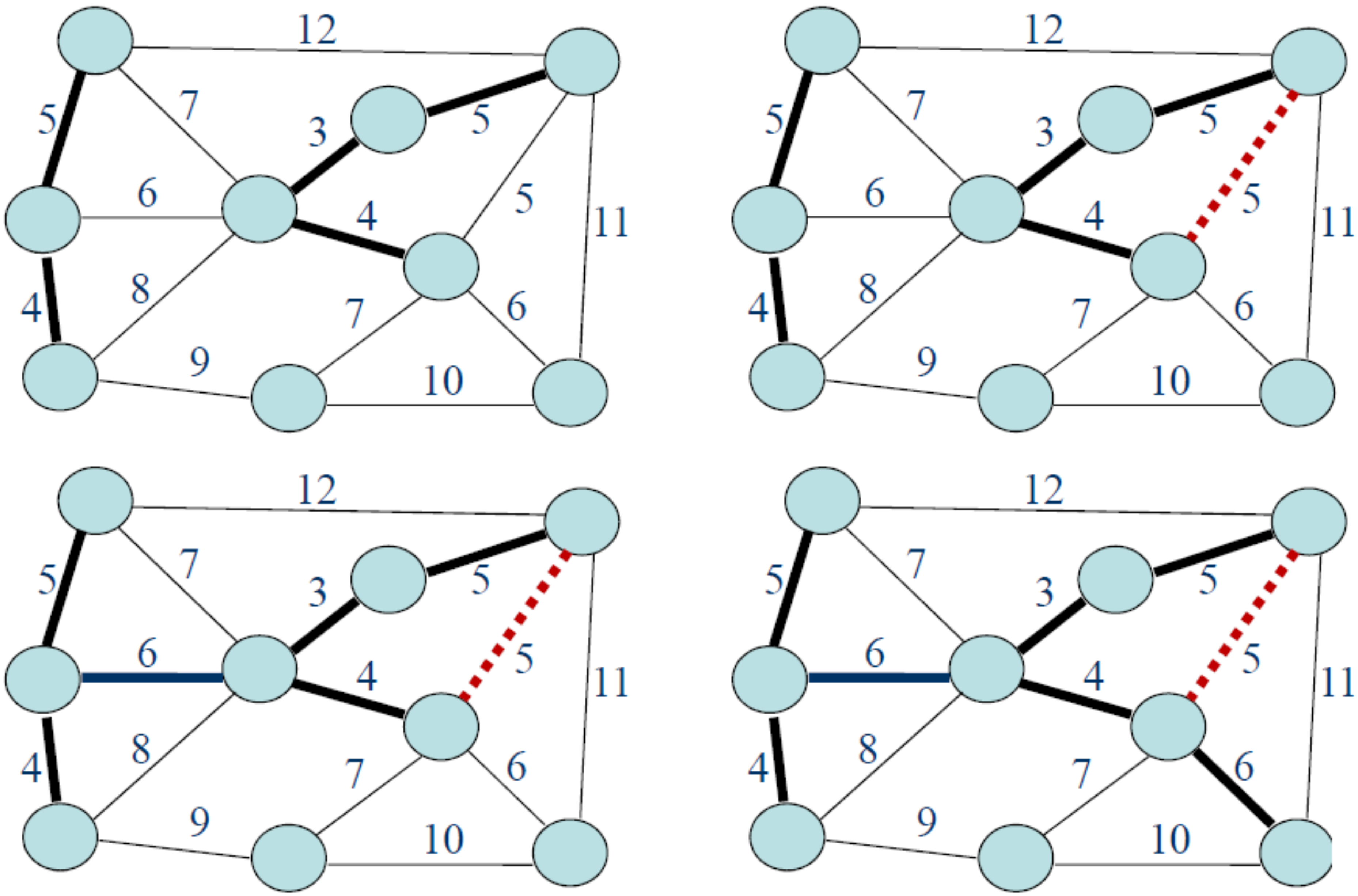
<u>procedure union</u>(x, y)
$r_x = \texttt{find}(x)$
$r_y = \texttt{find}(y)$
if $r_x = r_y$: return
if $\texttt{rank}(r_x) > \texttt{rank}(r_y)$:
    $\pi(r_y) = r_x$
else:
    $\pi(r_x) = r_y$
    if $\texttt{rank}(r_x) = \texttt{rank}(r_y)$: $\texttt{rank}(r_y) = \texttt{rank}(r_y) + 1$



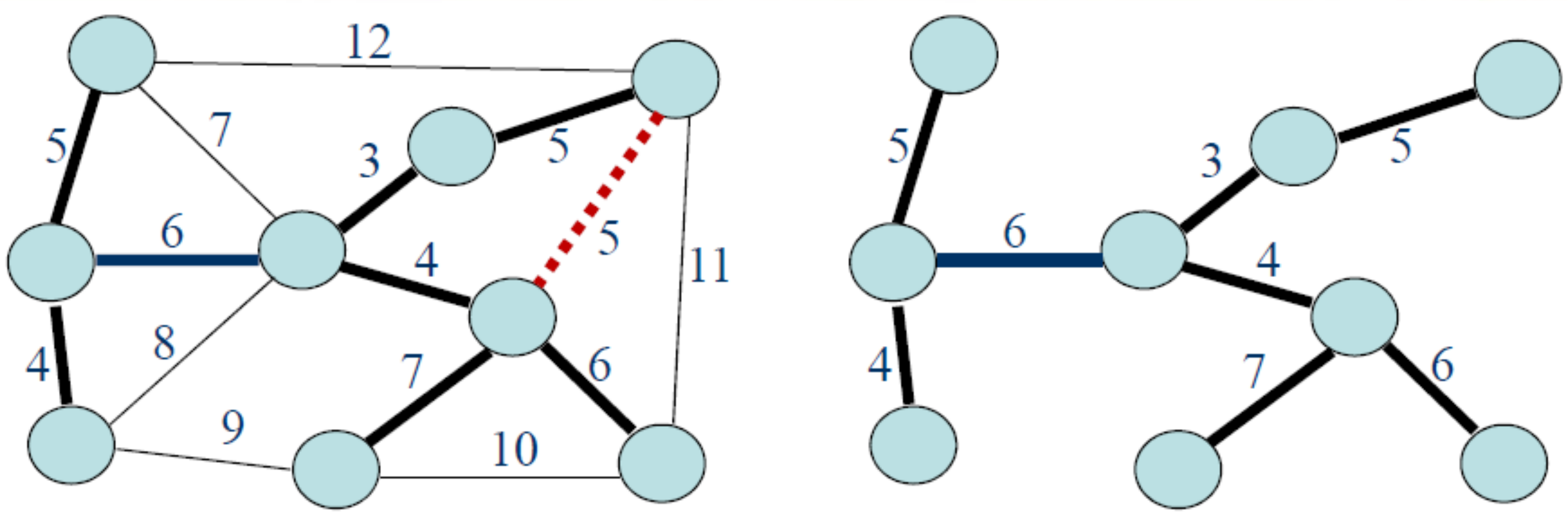
Kruskal example



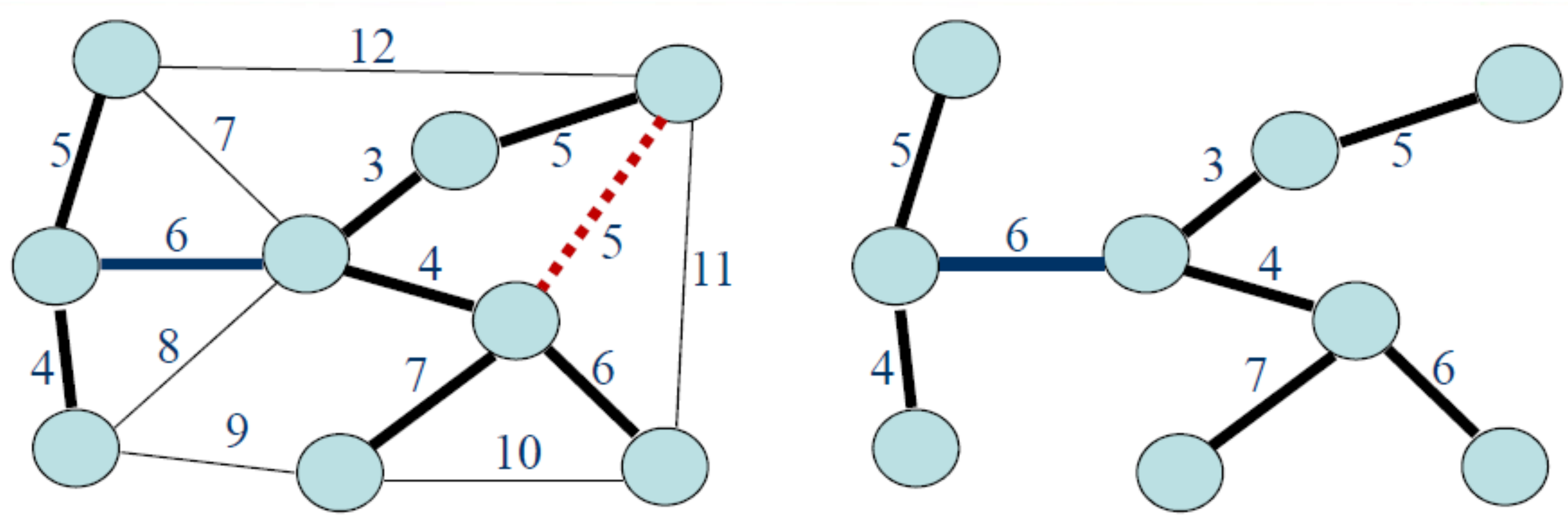
Kruskal example



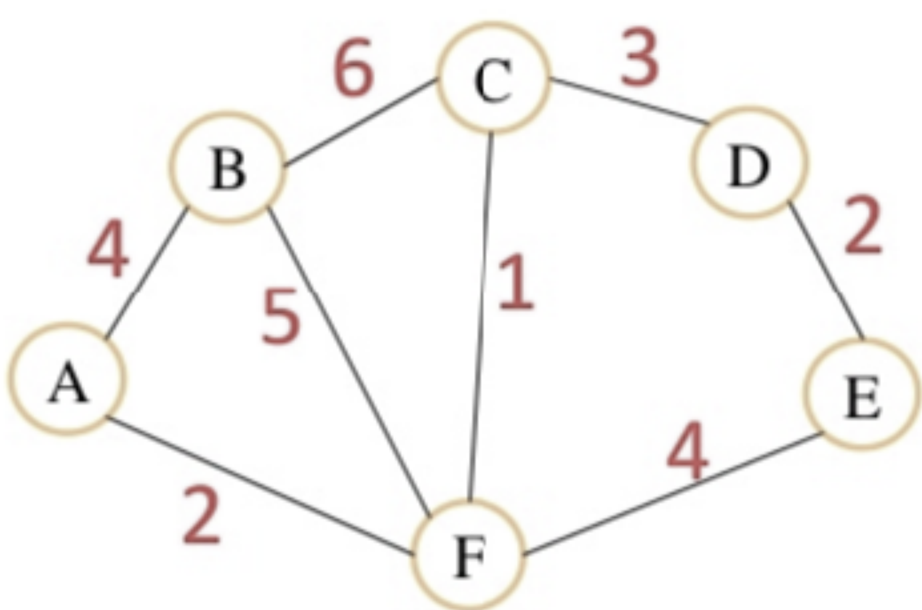
Kruskal example



Kruskal example

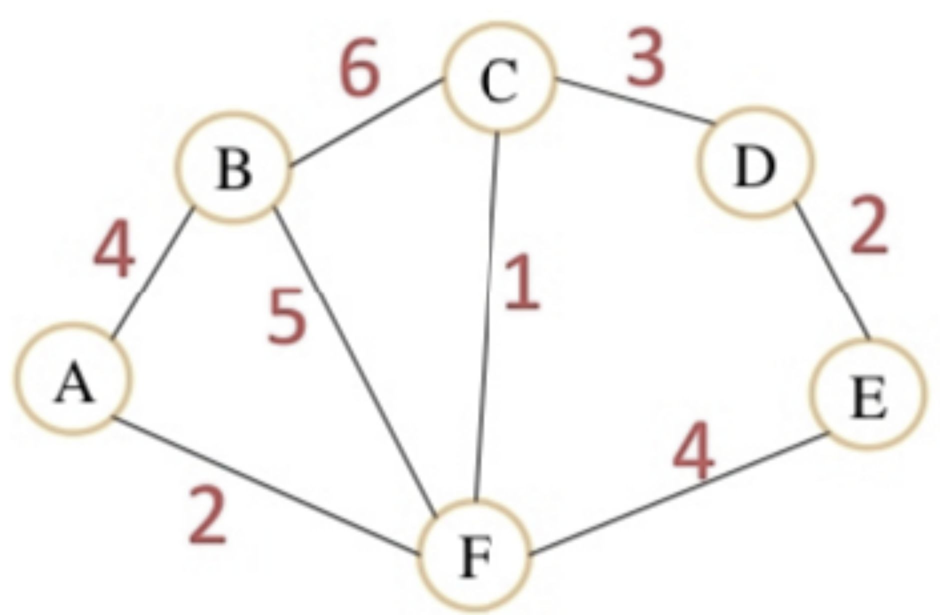


```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```



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

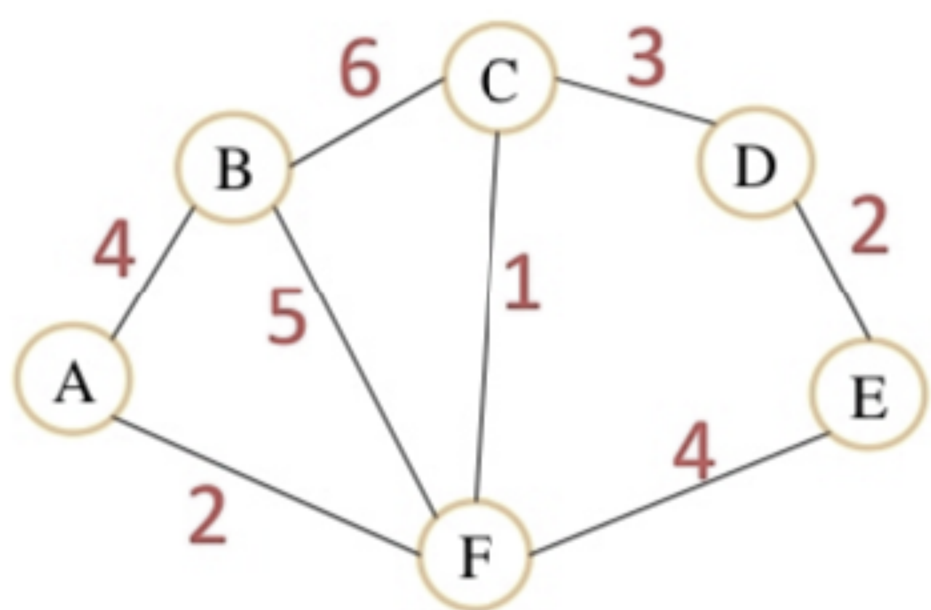
```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
      A = A U {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```



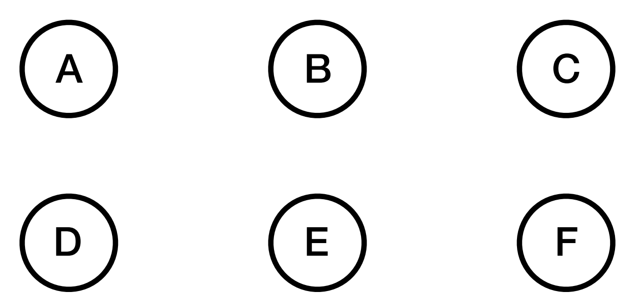
A = {}

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
```

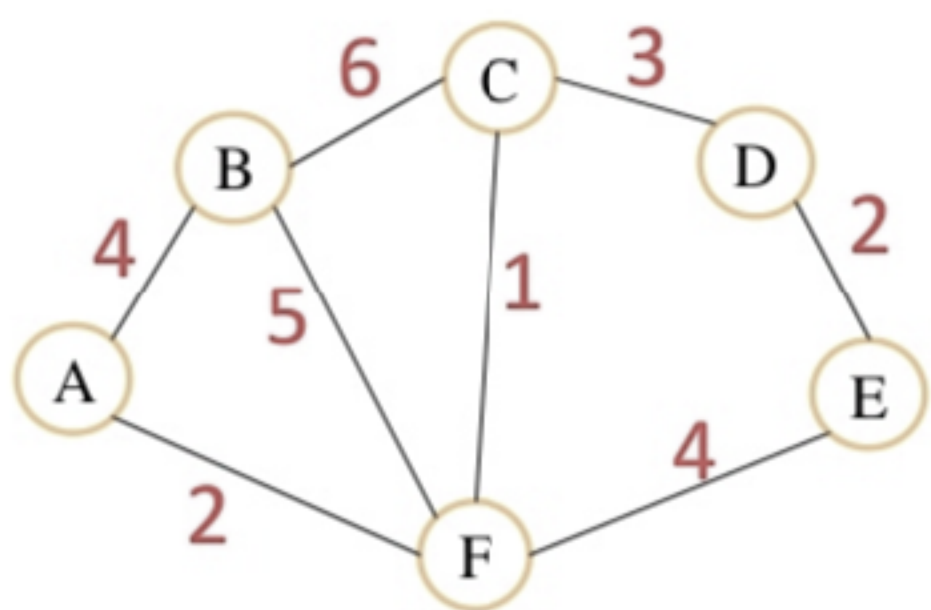


A = {}

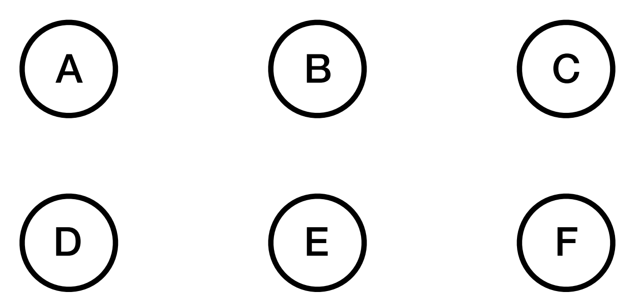


Edges	Weight
AB	4
BC	6
CD	3
DE	2
EF	4
AF	2
BF	5
CF	1

```
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
```

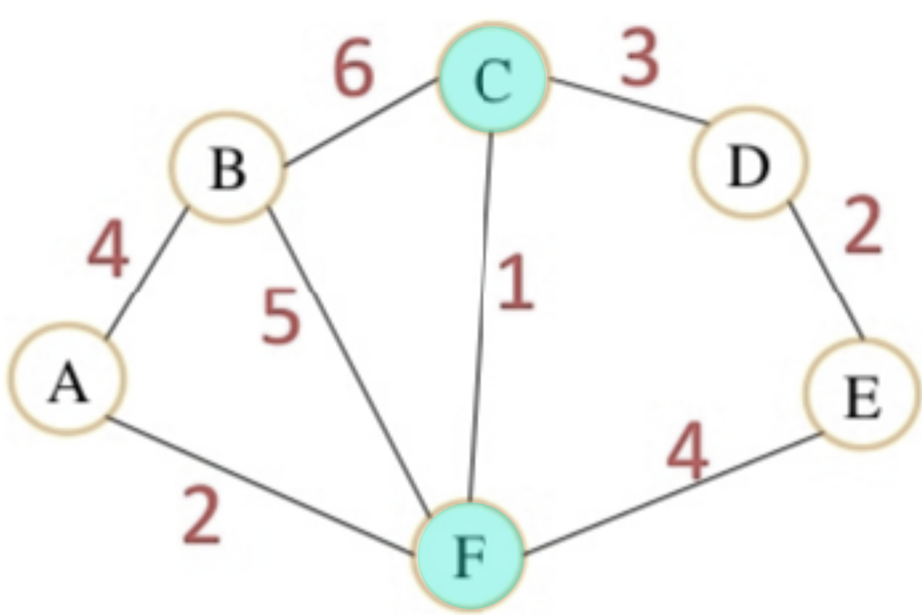


A = {}

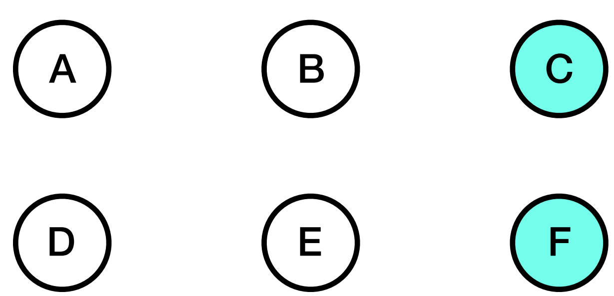


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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
```

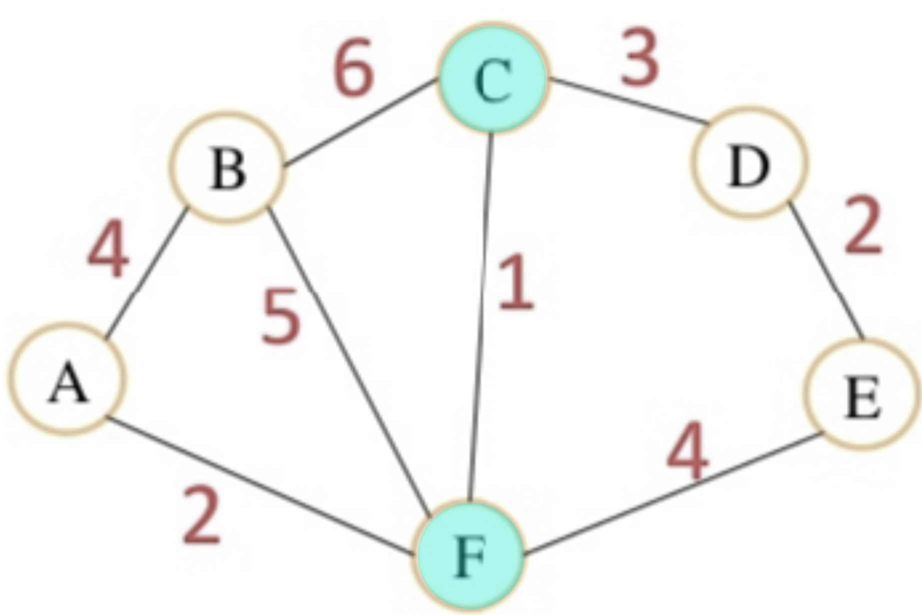


A = {}

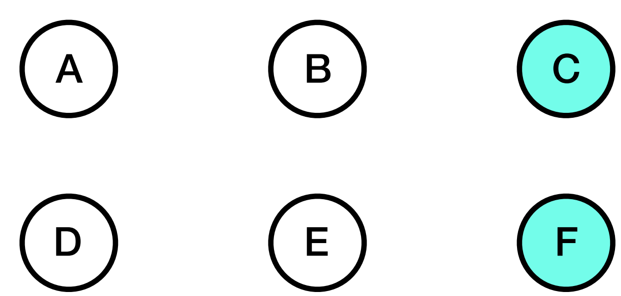


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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
```

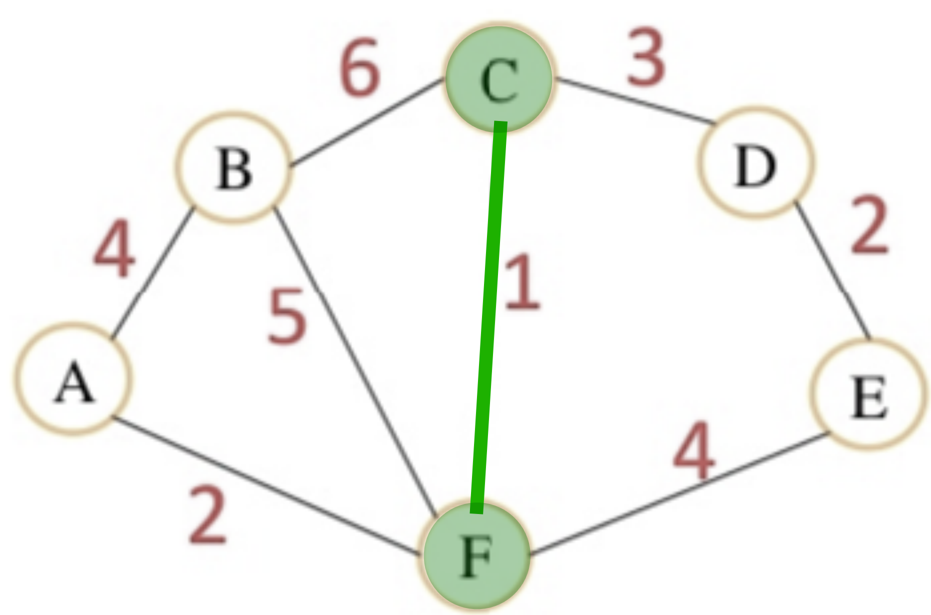


A = {(C,F)}

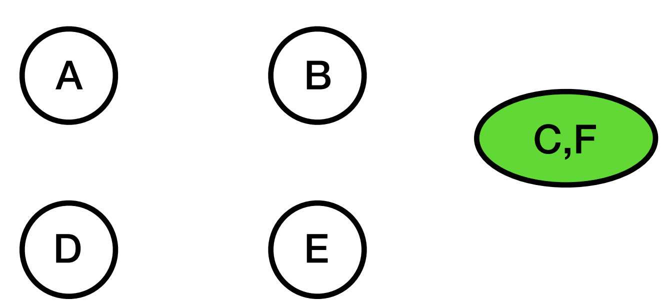


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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 {(u, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

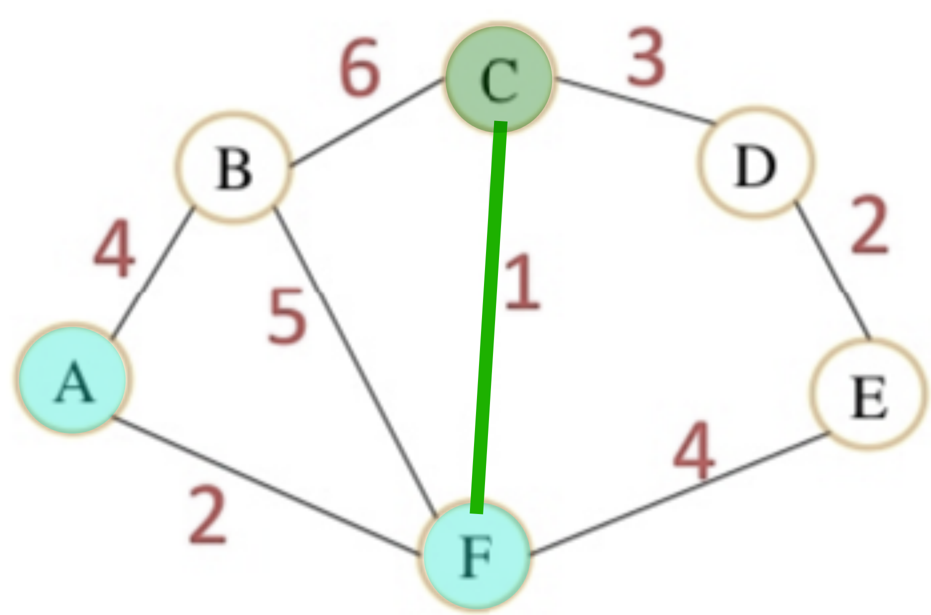


A = {(C,F)}

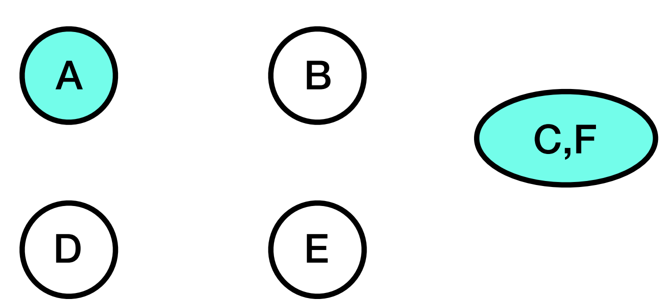


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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
```

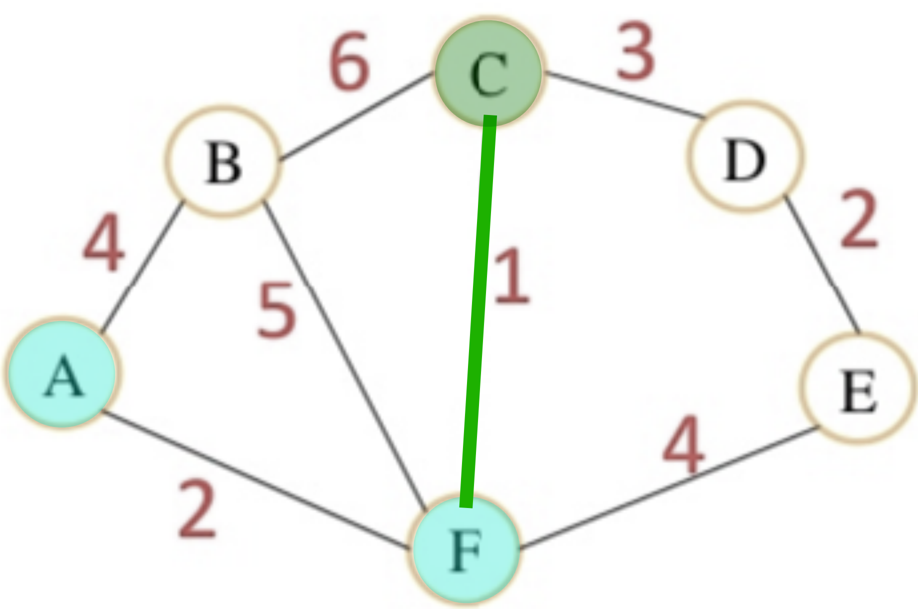


A = {(C,F)}

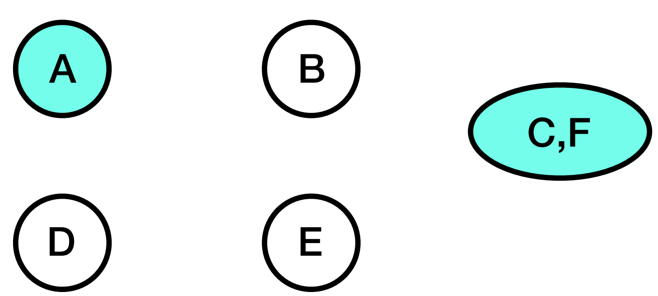


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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
```

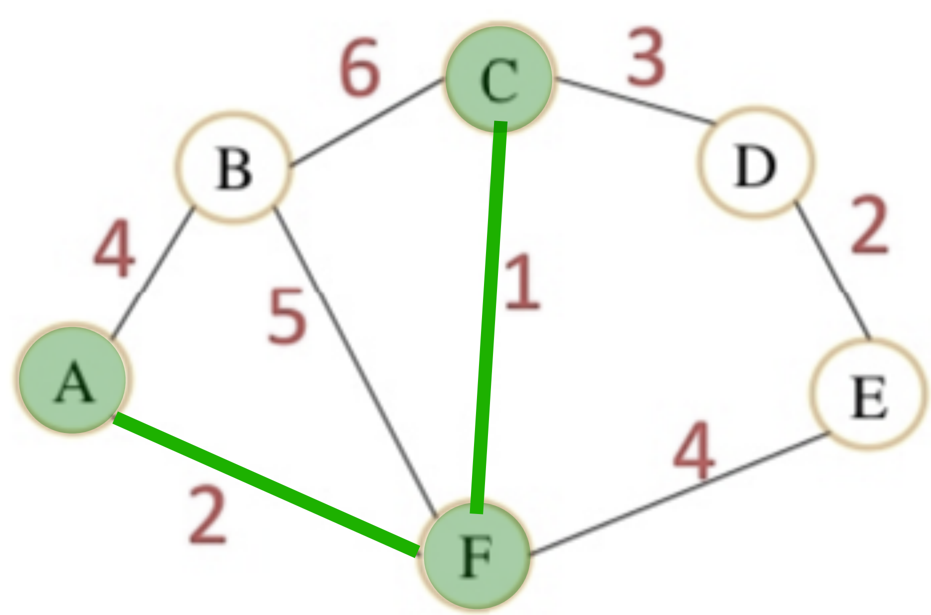


A = {(C,F),(A,F)}

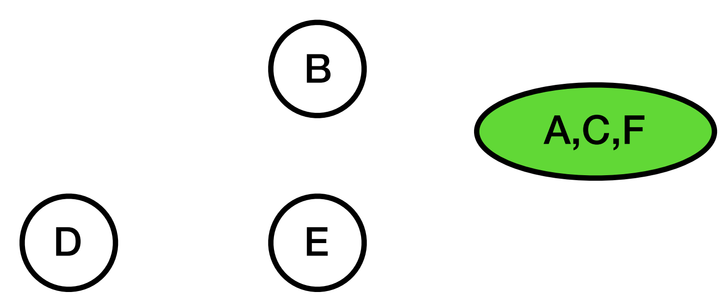


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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
```

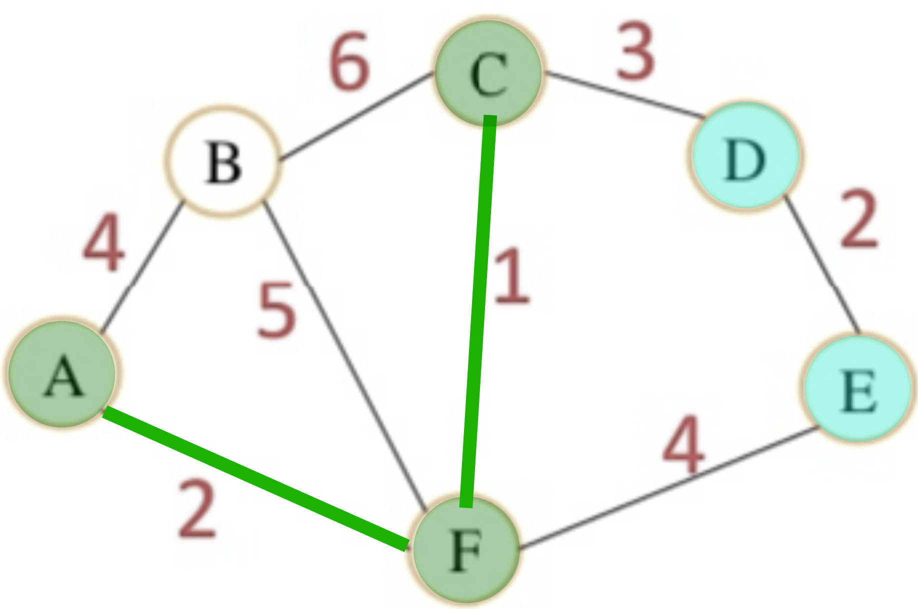


A = {(C,F),(A,F)}

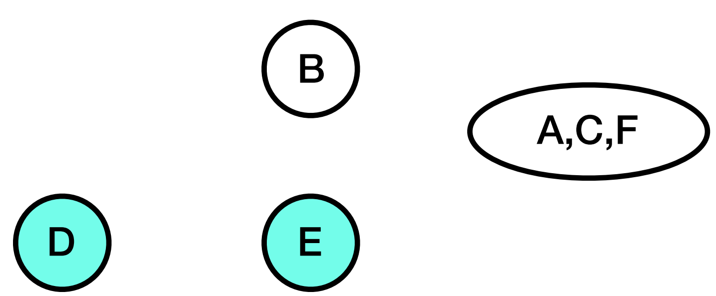


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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
```

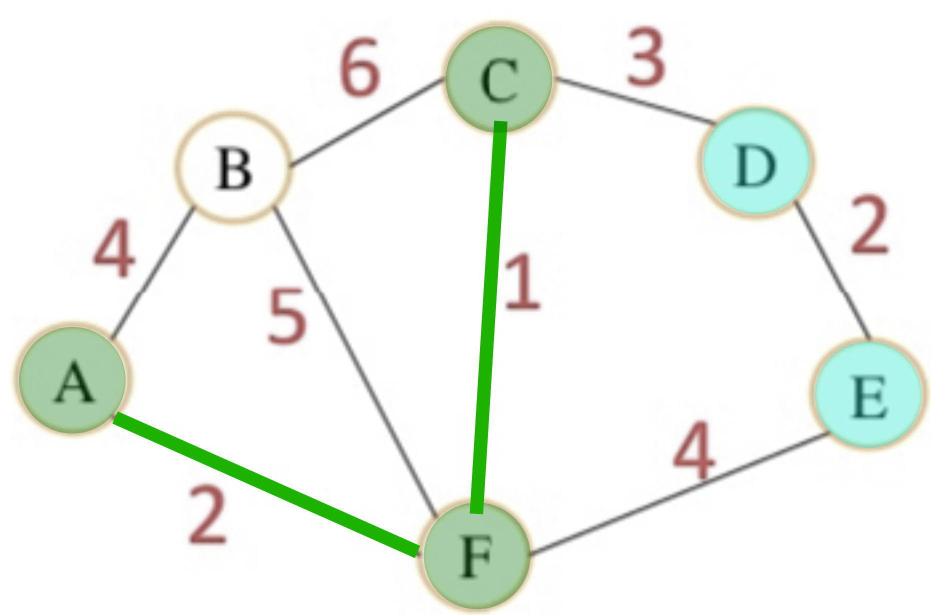


A = {(C,F),(A,F)}

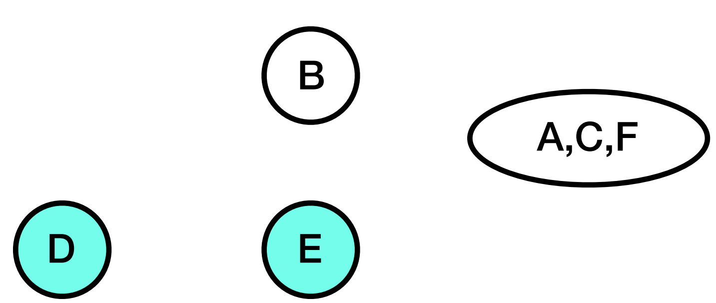


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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
```

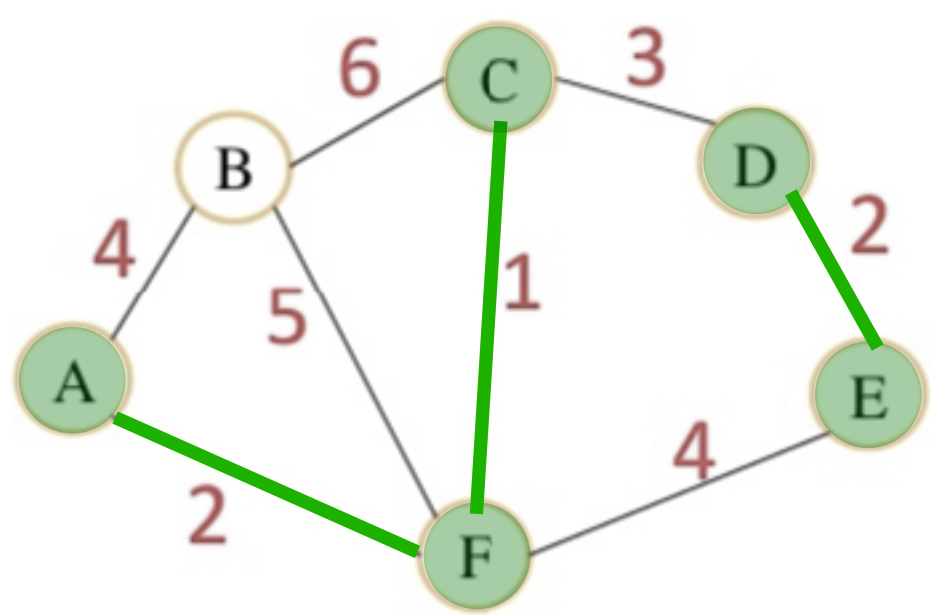


A = {(C,F),(A,F),(D,E)}

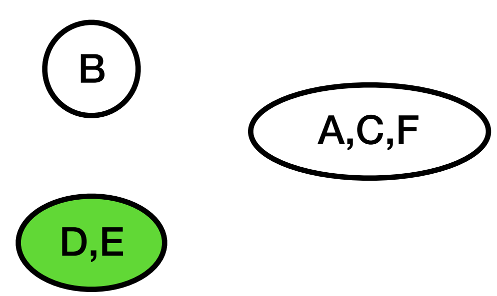


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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
```

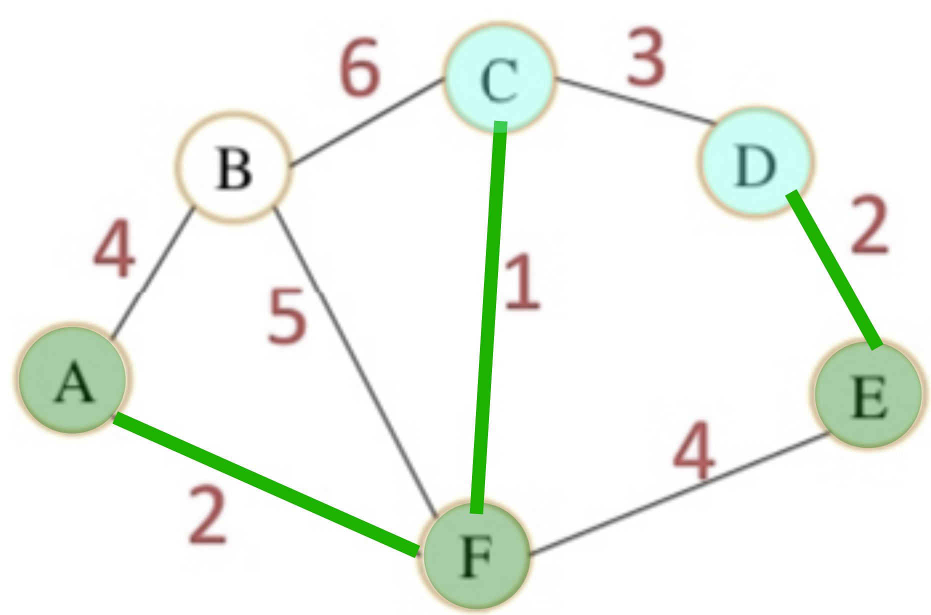


A = {(C,F),(A,F),(D,E)}

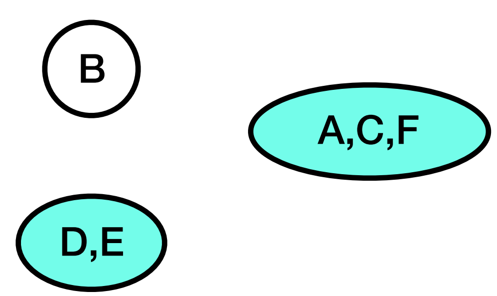


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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, v)}
      UNION(FIND-SET(u), FIND-SET(v))
return A
```

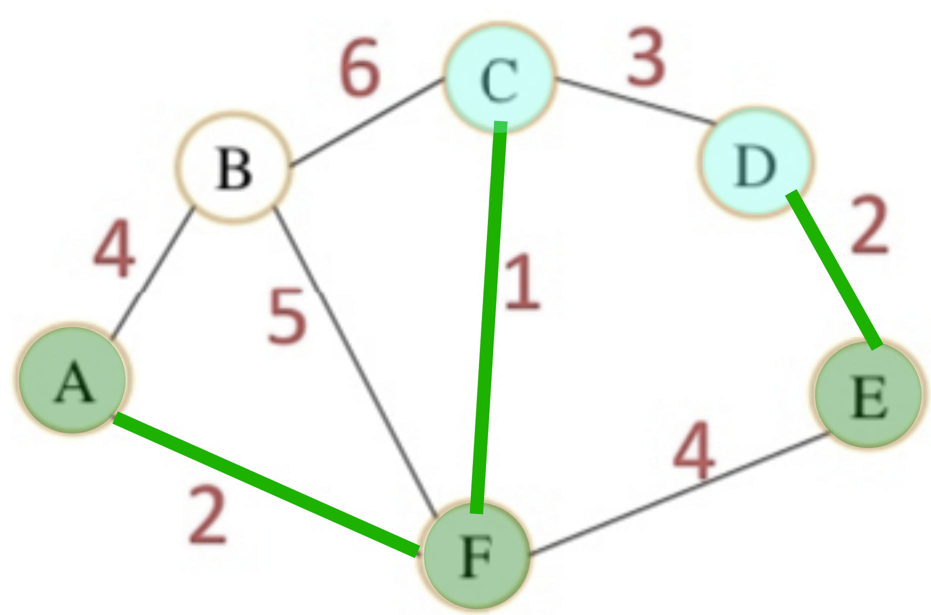


A = {(C,F),(A,F),(D,E)}

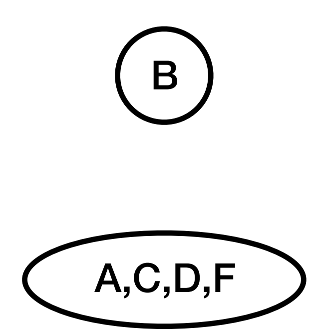


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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
```

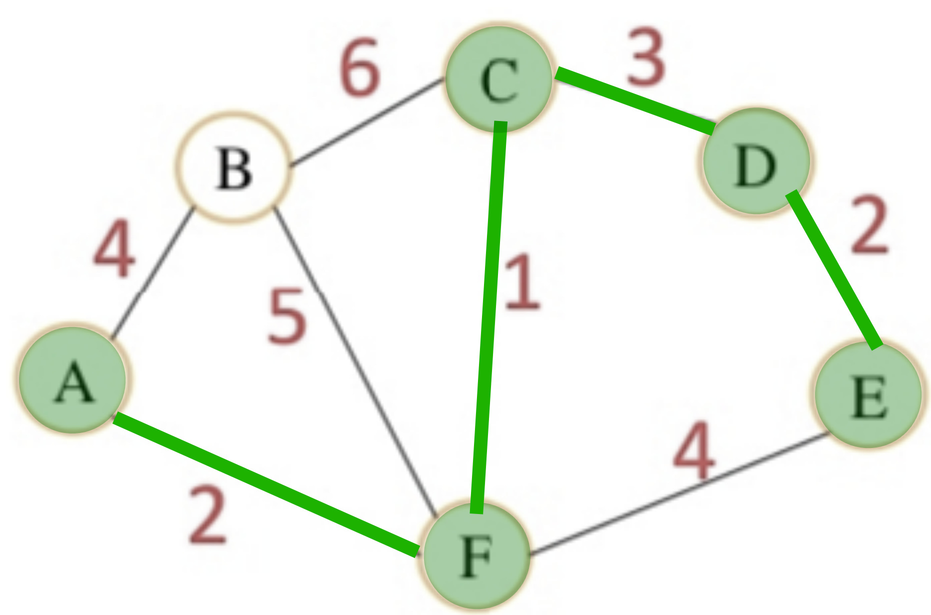


A = {(C,F),(A,F),(D,E),(C,D)}

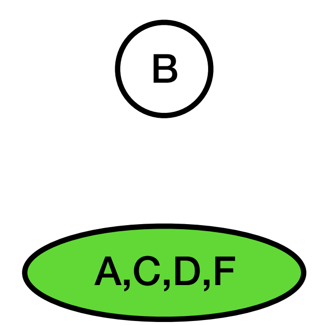


Edges	Weight
CF	1
AF	2
DE	2
CD	3
AB	4
FE	4
BF	5
BC	6

```
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
```



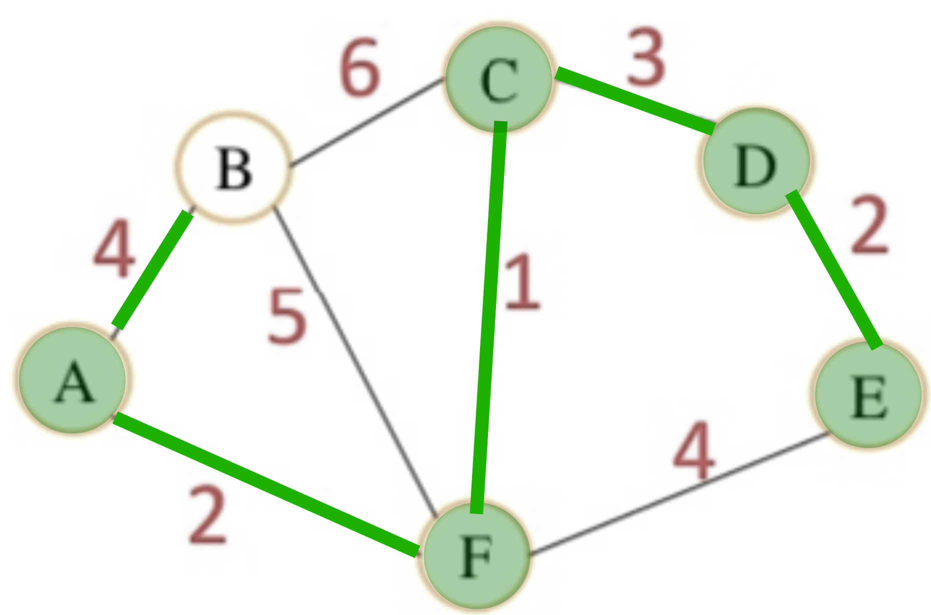
A = {(C,F),(A,F),(D,E), (C,D)}



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 = ∅
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 = ∅
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)

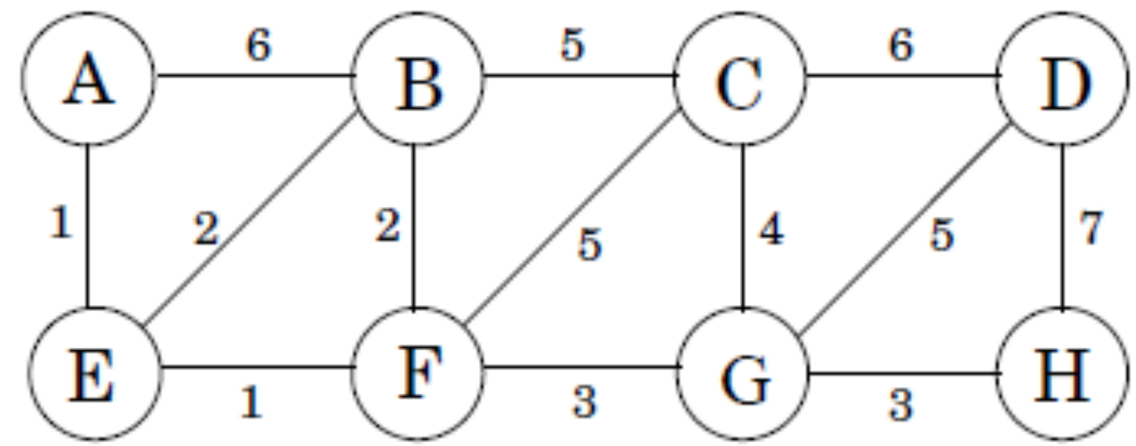
O(ELovV)

Complexitat Kruskal

$$\begin{aligned} \text{Complexitat} &= O(1) + O(V) + O(E\ \log E) + O(E\ \mathrm{Log} V) \\ &= O(E\ \log E) + O(E\ \log V) \\ &= O(E\ \log V) \end{aligned}$$

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?