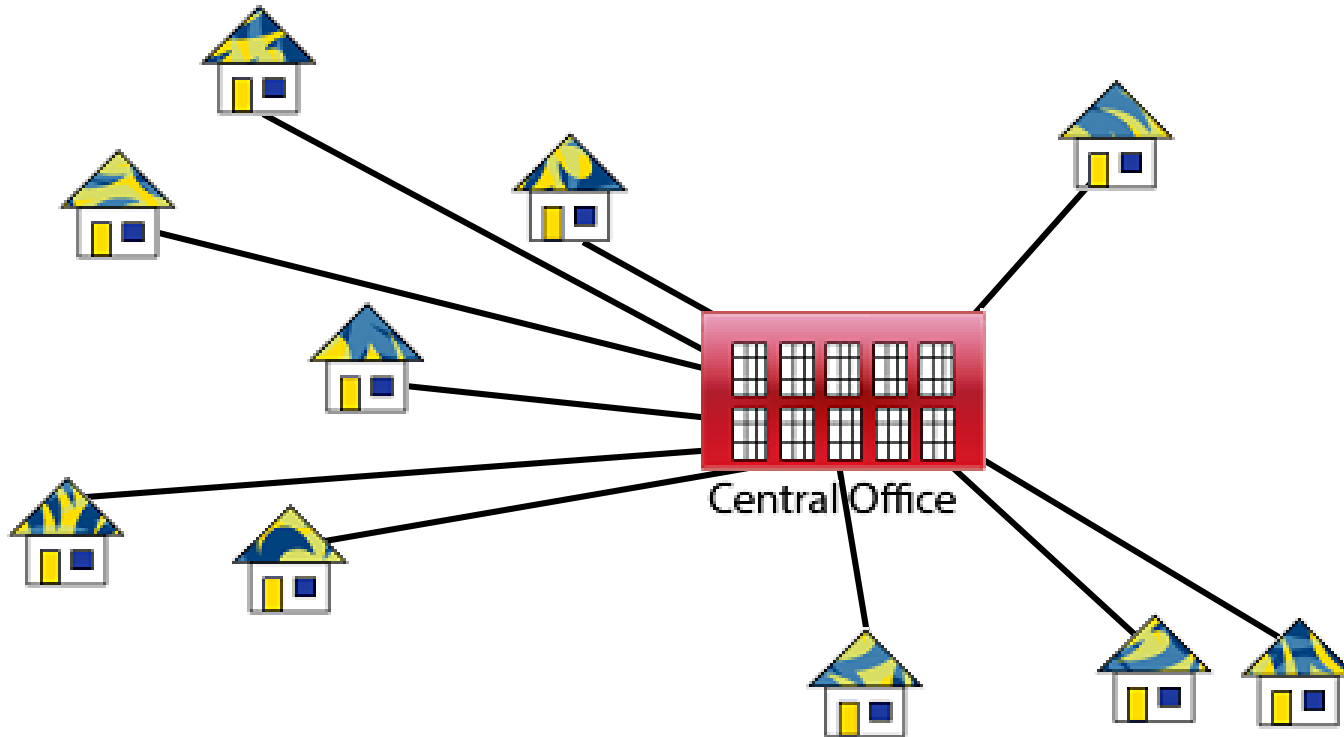


Graphs: Minimum Spanning Trees



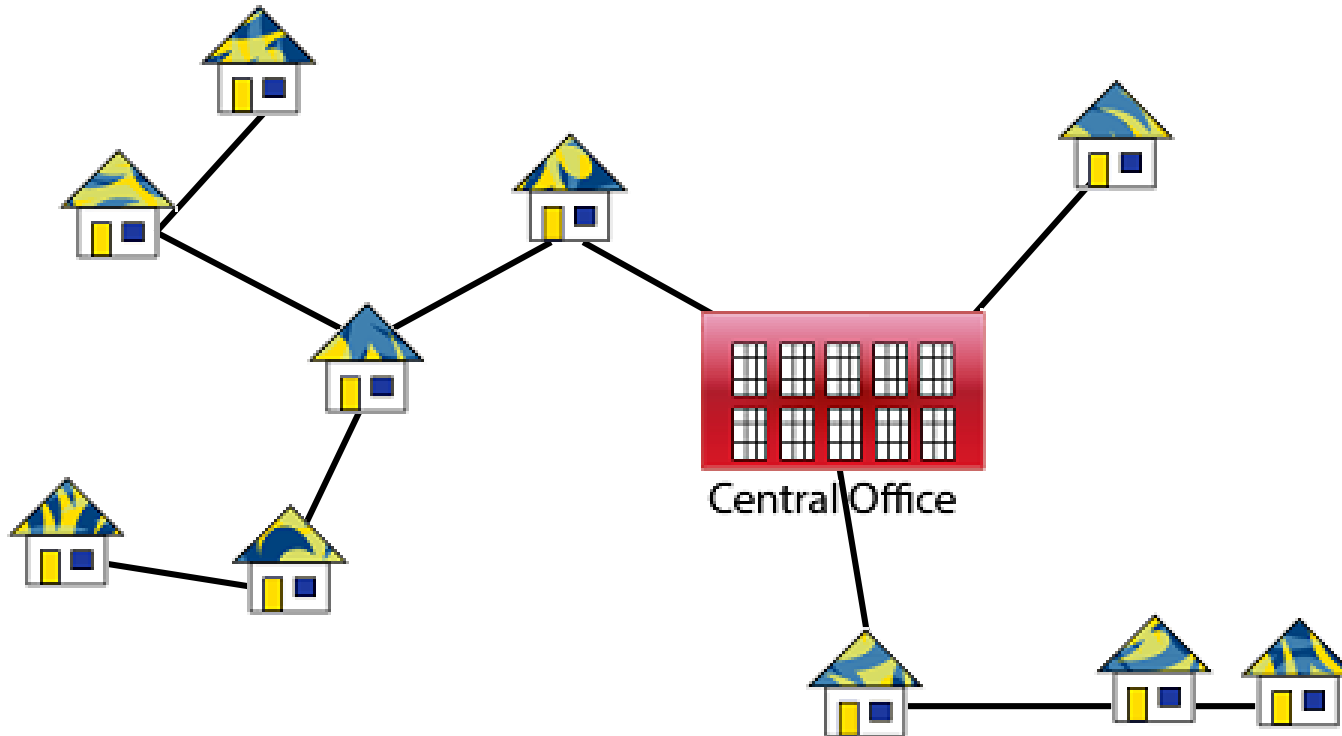
Jordi Cortadella and Jordi Petit
Department of Computer Science

Laying a communication network



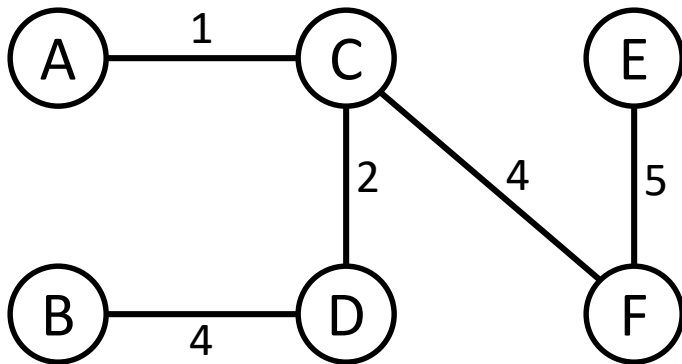
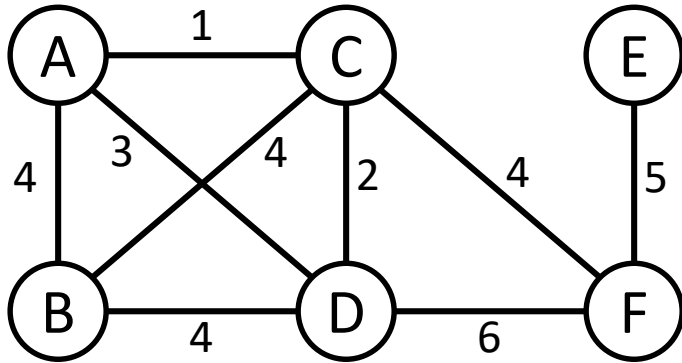
Source: <https://www.javatpoint.com/applications-of-minimum-spanning-tree>

Laying a communication network



Source: <https://www.javatpoint.com/applications-of-minimum-spanning-tree>

Minimum Spanning Trees



- Nodes are computers
- Edges are links
- Weights are maintenance cost
- Goal: pick a subset of edges such that
 - the nodes are connected
 - the maintenance cost is minimum

The solution is not unique.
Find another one !

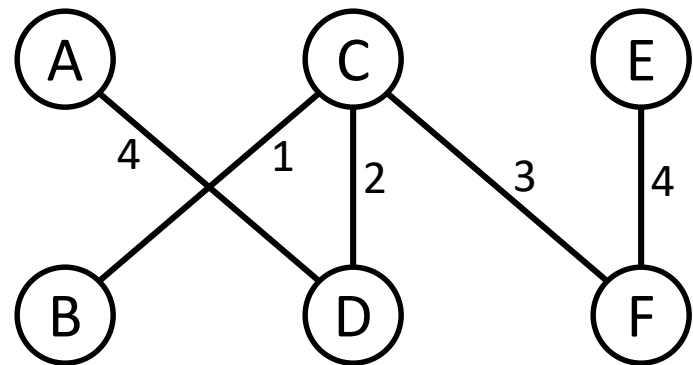
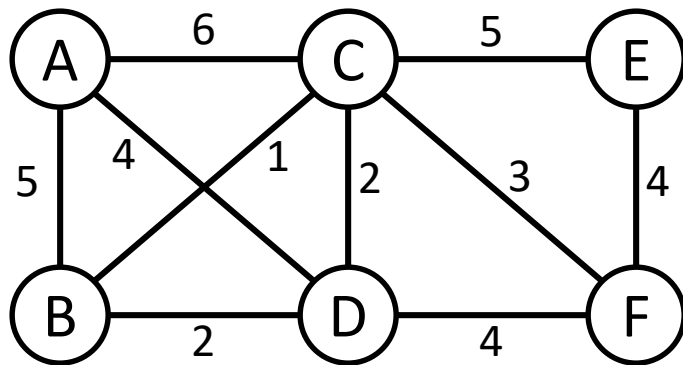
Property:
An optimal solution cannot contain a cycle.

Minimum Spanning Tree

- Given an undirected graph $G = (V, E)$ with edge weights w_e , find a tree $T = (V, E')$, with $E' \subseteq E$, that minimizes

$$\text{weight}(T) = \sum_{e \in E'} w_e .$$

- Greedy algorithm: repeatedly add the next lightest edge that does not produce a cycle.

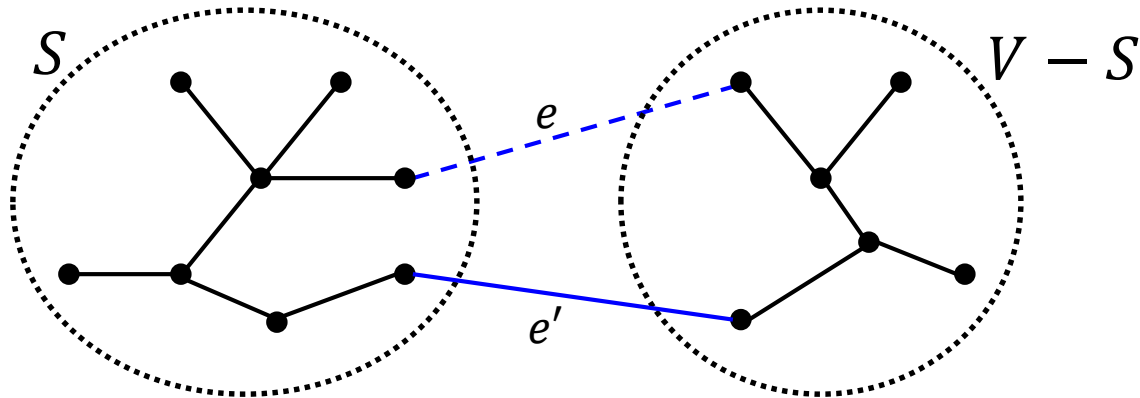


Note: We will now see that this strategy guarantees an MST.

Properties of trees

- **Definition:** A tree is an undirected graph that is connected and acyclic.
- **Property:** Any connected, undirected graph $G = (V, E)$ has $|E| \geq |V| - 1$ edges.
- **Property:** A tree on n nodes has $n - 1$ edges.
 - Start from an empty graph. Add one edge at a time making sure that it connects two disconnected components. After having added $n - 1$ edges, a tree has been formed.
- **Property:** Any connected, undirected graph $G = (V, E)$ with $|E| = |V| - 1$ is a tree.
 - It is sufficient to prove that G is acyclic. If not, we can always remove edges from cycles until the graph becomes acyclic.
- **Property:** Any undirected graph is a tree iff there is a unique path between any pair of nodes.
 - If there would be two paths between two nodes, the union of the paths would contain a cycle.

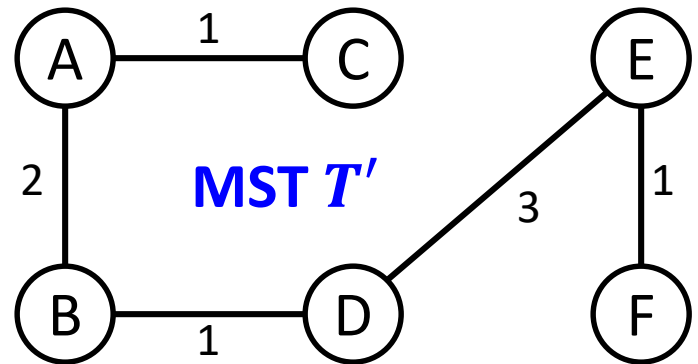
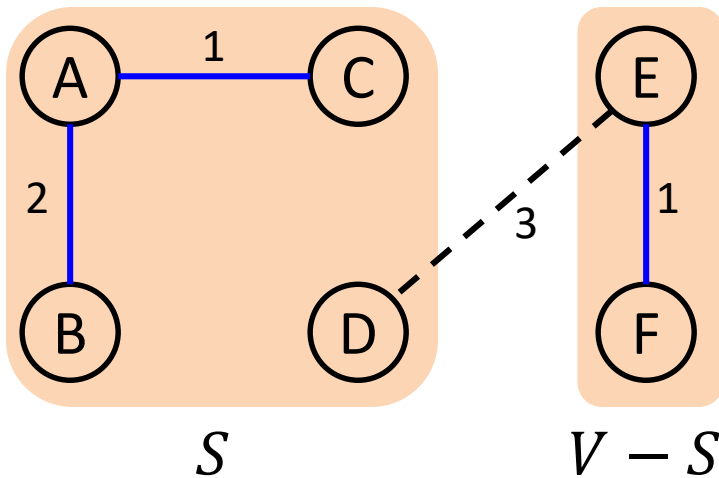
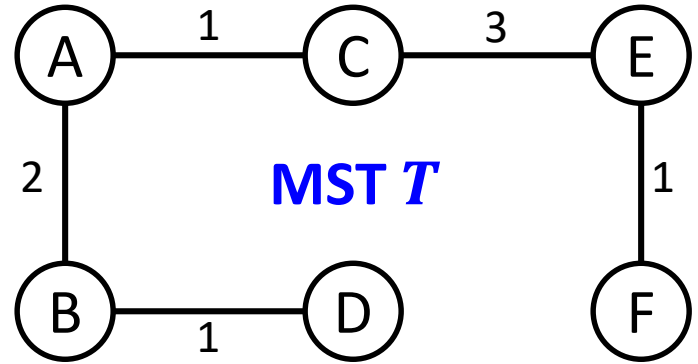
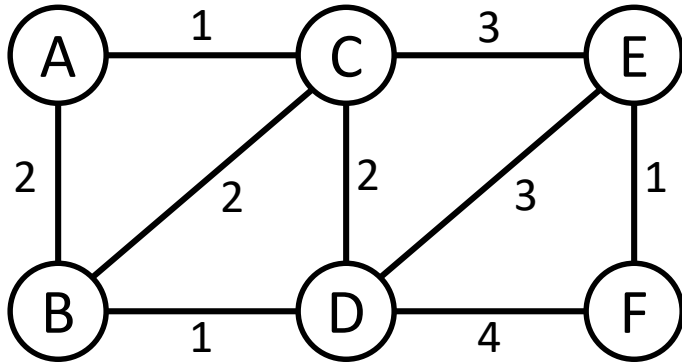
The cut property



Suppose edges X are part of an MST of $G = (V, E)$. Pick any subset of nodes S for which X does not cross between S and $V - S$, and let e be the lightest edge across this partition. Then $X \cup \{e\}$ is part of some MST.

Proof (sketch): Let T be an MST and assume e is not in T . If we add e to T , a cycle will be created with another edge e' across the cut $(S, V - S)$. We can now remove e' and obtain another tree T' with $\text{weight}(T') \leq \text{weight}(T)$. Since T is an MST, then the weights must be equal.

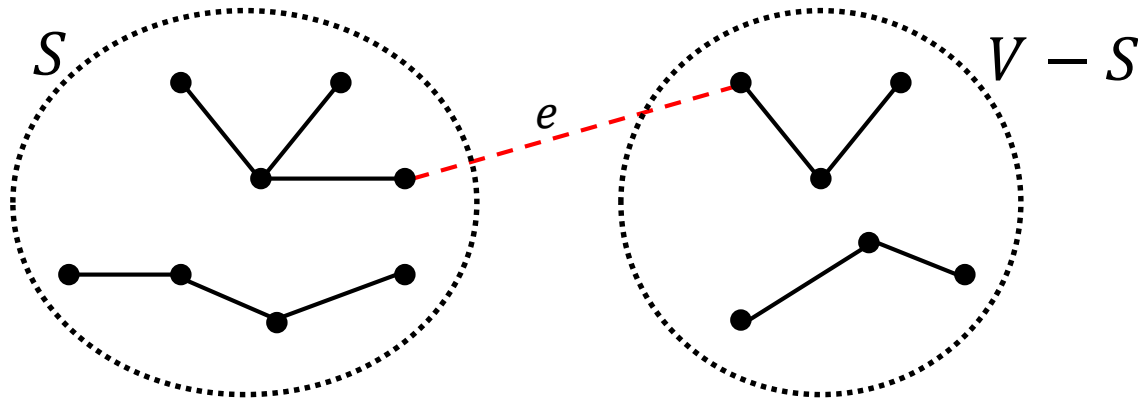
The cut property: example



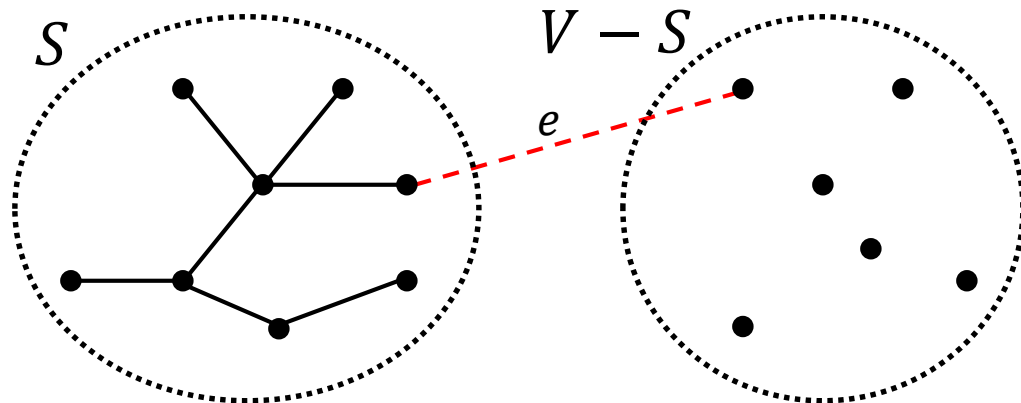
Minimum Spanning Tree

Any scheme like this works (because of the properties of trees):

```
 $X = \{\}$  # The set of edges of the MST  
repeat  $|V| - 1$  times:  
  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  $V - S$   
   $X = X \cup \{e\}$ 
```



MST: two strategies



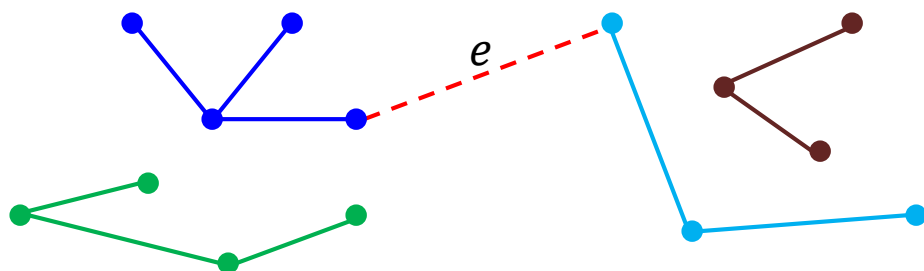
Prim's algorithm

Invariant:

- A set of nodes (S) is in the tree.

Progress:

- The lightest edge with exactly one endpoint in S is added.



Kruskal's algorithm

Invariant:

- A set of trees (forest) has been constructed.

Progress:

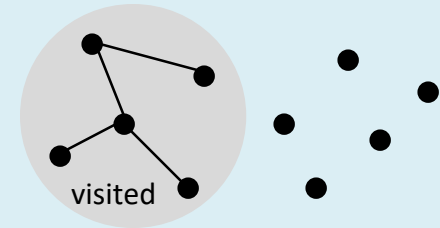
- The lightest edge between two trees is added.

Prim's algorithm

```
def Prim( $G, w$ ) → prev:
    """Input: A connected undirected Graph  $G(V, E)$ 
       with edge weights  $w(e)$ .
       Output: An MST defined by the vector prev."""
    for all  $u \in V$ :
        visited[ $u$ ] = False
        prev[ $u$ ] = nil
    pick any initial node  $u_0$ 
    visited[ $u_0$ ] = True
     $n = 1$ 

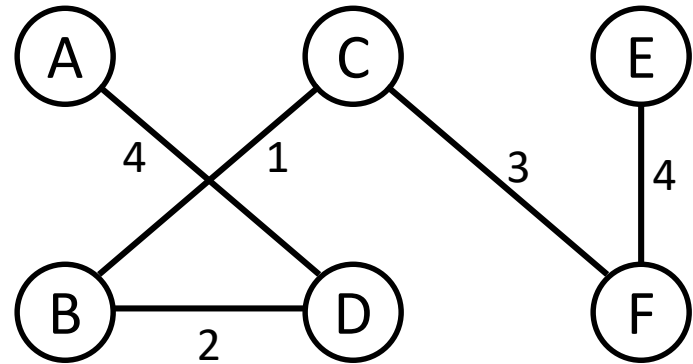
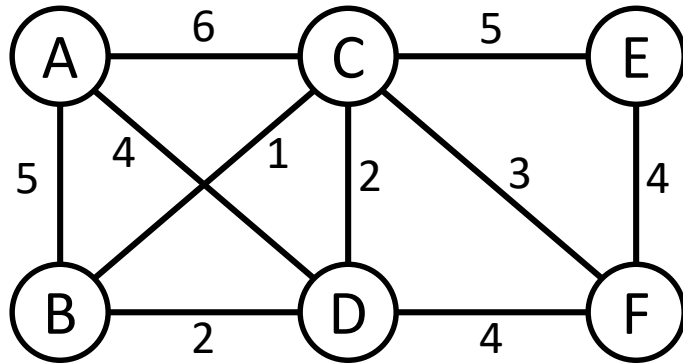
    #  $Q$ : priority queue of edges using  $w(e)$  as priority
     $Q = \text{makequeue}()$ 
    for each  $(u_0, v) \in E$ :  $Q.\text{insert}(u_0, v)$ 

    while  $n < |V|$ :
         $(u, v) = \text{deletemin}(Q)$  # Edge with smallest weight
        if not visited[ $v$ ]:
            visited[ $v$ ] = True
            prev[ $v$ ] =  $u$ 
             $n = n + 1$ 
            for each  $(v, x) \in E$ :
                if not visited[ $x$ ]:  $Q.\text{insert}(v, x)$ 
```



Complexity: $O(|E| \log |V|)$

Prim's algorithm



Q: (AD,4) (AB,5) (AC,6)

(DB,2) (DC,2) (DF,4) (AB,5) (AC,6)

(BC,1) (DC,2) (DF,4) (AB,5) (AC,6)

(DC,2) (CF,3) (DF,4) (AB,5) (CE,5) (AC,6)

(CF,3) (DF,4) (AB,5) (CE,5) (AC,6)

(DF,4) (FE,4) (AB,5) (CE,5) (AC,6)

(FE,4) (AB,5) (CE,5) (AC,6)

Kruskal's algorithm

Informal algorithm:

- Sort edges by weight.
- Visit edges in ascending order of weight and add them as long as they do not create a cycle.

How do we know whether a new edge will create a cycle?

```
def Kruskal( $G, w$ )  $\rightarrow$  MST:
```

```
    """Input: A connected undirected Graph  $G(V, E)$   
           with edge weights  $w_e$ .
```

```
    Output: An MST defined by the edges in MST."""
```

```
MST = {}
```

```
sort the edges in  $E$  by weight
```

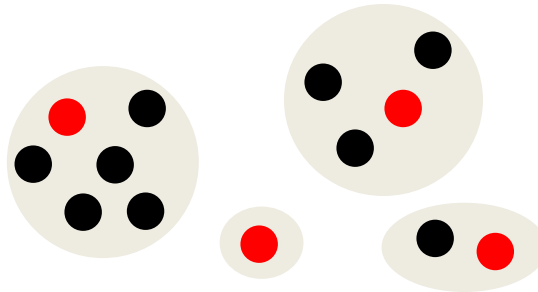
```
for all  $(u, v) \in E$ , in ascending order of weight:
```

```
    if (MST has no path connecting  $u$  and  $v$ ):
```

```
        MST = MST  $\cup$   $\{(u, v)\}$ 
```

Disjoint sets

- A data structure to store a collection of disjoint sets.



- Operations:
 - $\text{makeset}(x)$: creates a singleton set containing just x .
 - $\text{find}(x)$: returns the identifier of the set containing x .
 - $\text{union}(x, y)$: merges the sets containing x and y .
- Kruskal's algorithm uses disjoint sets and calls
 - makeset : $|V|$ times
 - find : $2 \cdot |E|$ times
 - union : $|V| - 1$ times

Kruskal's algorithm

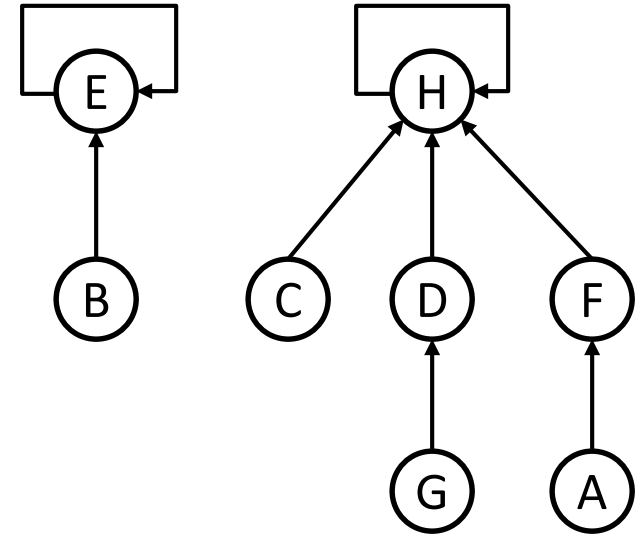
```
def Kruskal( $G, w$ )  $\rightarrow$  MST:
    """Input: A connected undirected Graph  $G(V, E)$ 
        with edge weights  $w_e$ .
        Output: An MST defined by the edges in MST."""

    for all  $u \in V$ : makeset( $u$ )

    MST = {}
    sort the edges in  $E$  by weight
    for all  $(u, v) \in E$ , in ascending order of weight:
        if (find( $u$ )  $\neq$  find( $v$ )):
            MST = MST  $\cup$   $\{(u, v)\}$ 
            union( $u, v$ )
```

Disjoint sets

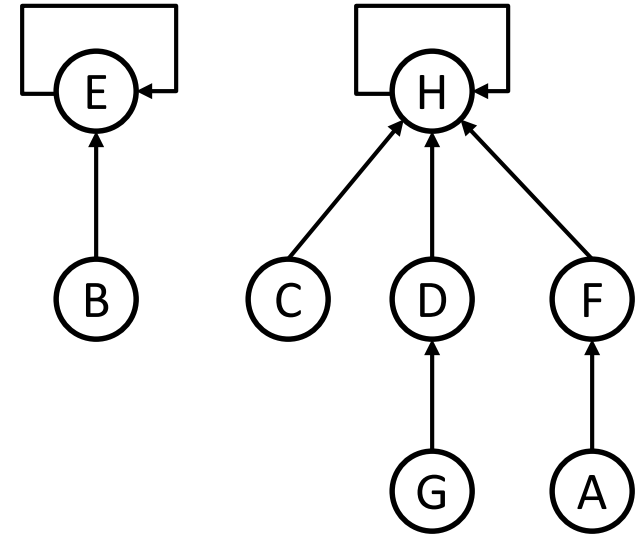
- The nodes are organized as a set of trees. Each tree represents a set.
- Each node has two attributes:
 - parent (π): ancestor in the tree
 - rank: height of the subtree
- The root element is the representative for the set: its parent pointer is itself (self-loop).
- The efficiency of the operations depends on the height of the trees.



```
def makeset(x):  
     $\pi(x) = x$   
    rank(x) = 0  
  
def find(x):  
    while  $x \neq \pi(x)$ :  $x = \pi(x)$   
    return x
```


Disjoint sets

```
def union(x, y):  
    rx = find(x)  
    ry = find(y)  
    if rx = ry: return  
  
    if rank(rx) > rank(ry):  
        π(ry) = rx  
    else:  
        π(rx) = ry  
        if rank(rx) = rank(ry):  
            rank(ry) = rank(ry) + 1
```



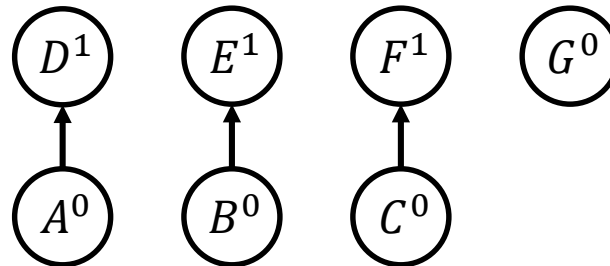
```
def makeset(x):  
    π(x) = x  
    rank(x) = 0  
  
def find(x):  
    while x ≠ π(x): x = π(x)  
    return x
```

Disjoint sets

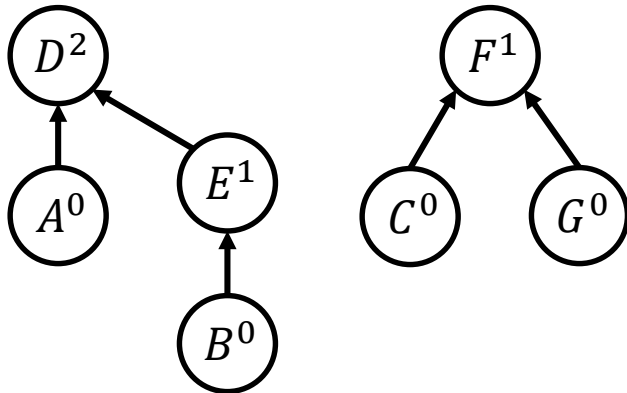
After `makeset(A), ..., makeset(G)`:



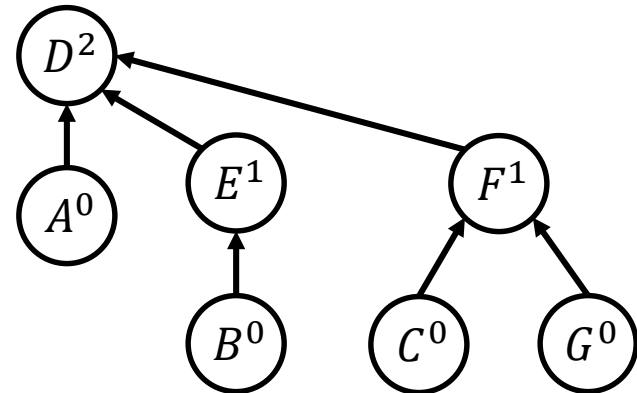
After `union(A,D), union(B,E), union(C,F)`:



After `union(C,G), union(E,A)`:



After `union(B,G)`:

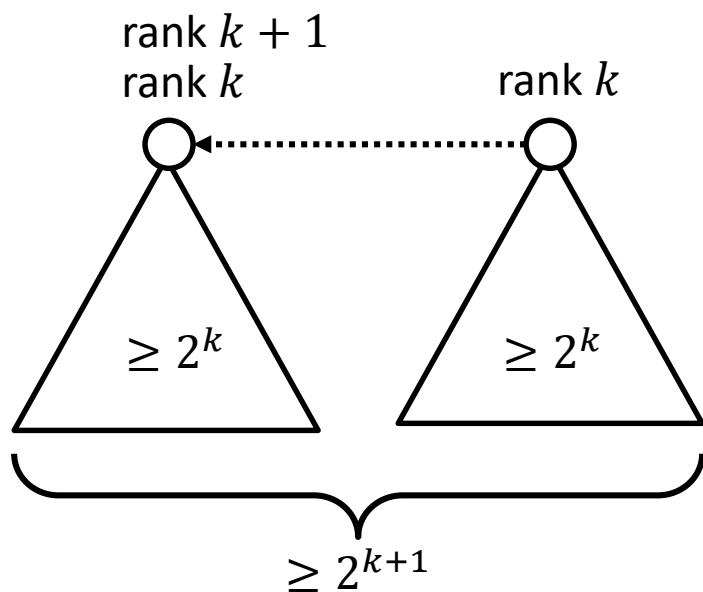


Property: Any root node of rank k has at least 2^k nodes in its tree.

Property: If there are n elements overall, there can be at most $n/2^k$ nodes of rank k . Therefore, all trees have height $\leq \log n$.

Disjoint sets

Property 1: proof by induction



Property 2:

For n nodes, the tallest possible tree could have rank k , such that:

$$n \geq 2^k$$



$$k \leq \log_2 n$$

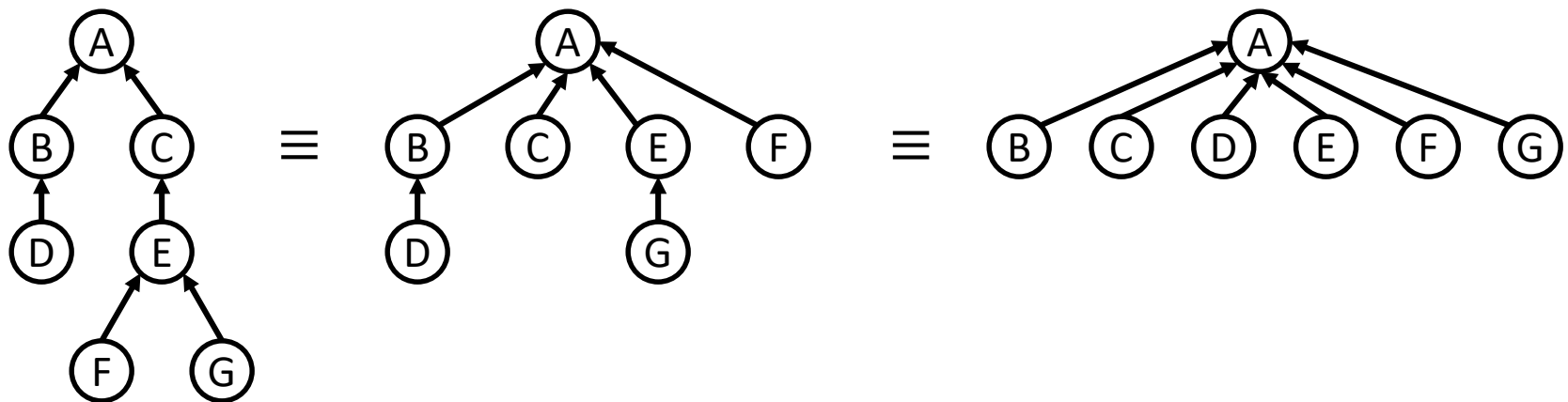
Therefore, $\text{find}(x)$ is $O(\log n)$

Property 1: Any root node of rank k has at least 2^k nodes in its tree.

Property 2: If there are n elements overall, there can be at most $n/2^k$ nodes of rank k . Therefore, all trees have height $\leq \log n$.

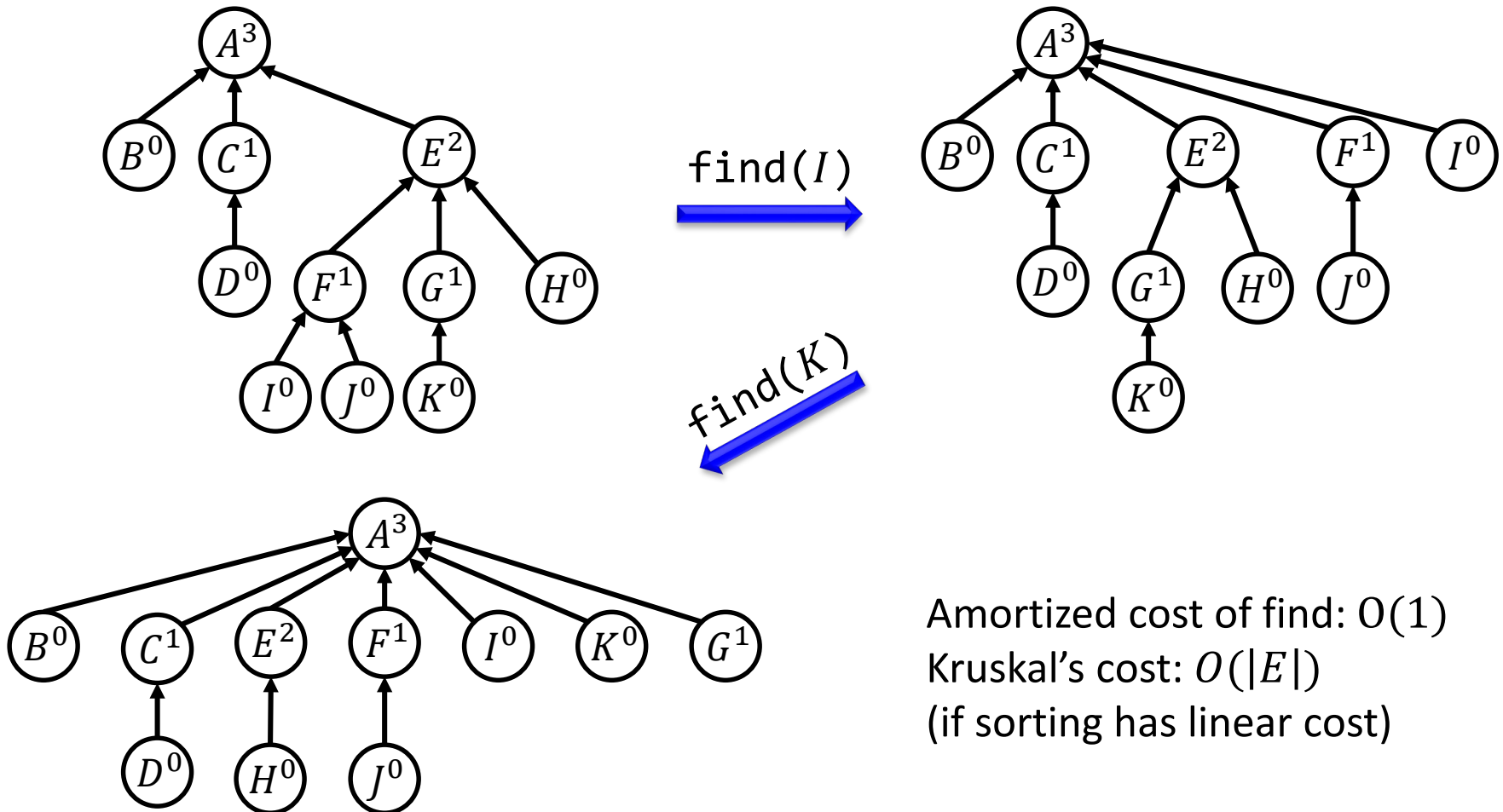
Disjoint sets: path compression

- Complexity of Kruskal's algorithm: $O(|E| \log |V|)$.
 - Sorting edges: $O(|E| \log |E|) = O(|E| \log |V|)$.
 - Find + union ($2 \cdot |E|$ times): $O(|E| \log |V|)$.
- How about if the edges are already sorted or sorting can be done in linear time (weights are integer and small)?
- Path compression:



Disjoint sets: path compression

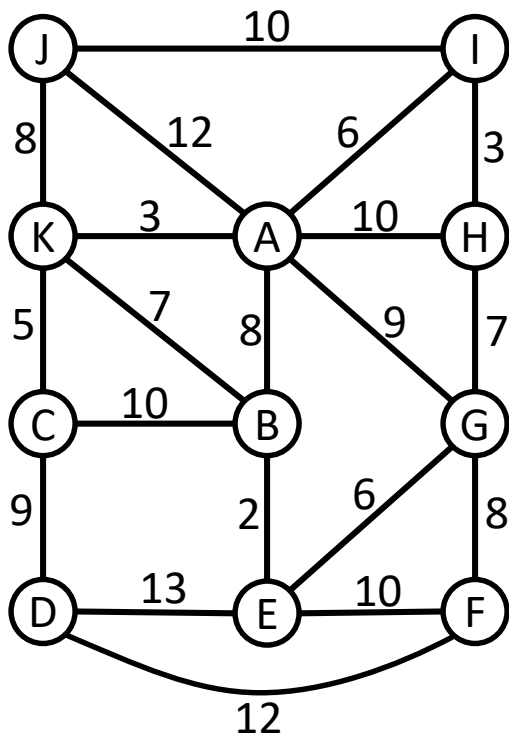
```
def find(x):  
    if x ≠ π(x): π(x) = find(π(x))  
    return π(x)
```



Amortized cost of find: $O(1)$
Kruskal's cost: $O(|E|)$
(if sorting has linear cost)

EXERCISES

Minimum Spanning Trees



- Calculate the shortest path tree from node A using Dijkstra's algorithm.
- Calculate the MST using Prim's algorithm. Indicate the sequence of edges added to the tree and the evolution of the priority queue.
- Calculate the MST using Kruskal's algorithm. Indicate the sequence of edges added to the tree and the evolution of the disjoint sets. In case of a tie between two edges, try to select the one that is not in Prim's tree.