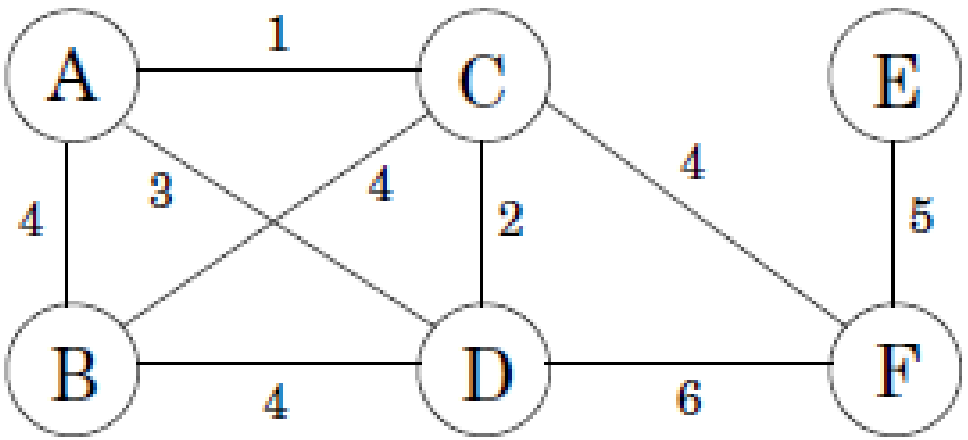
Greedy Algorithms

Greedy algorithms

- Algorithm design paradigm
- Idea : when we have a choice to make, make the one that looks best right now. Make a locally optimal choice in hope of getting a globally optimal solution.

Problem

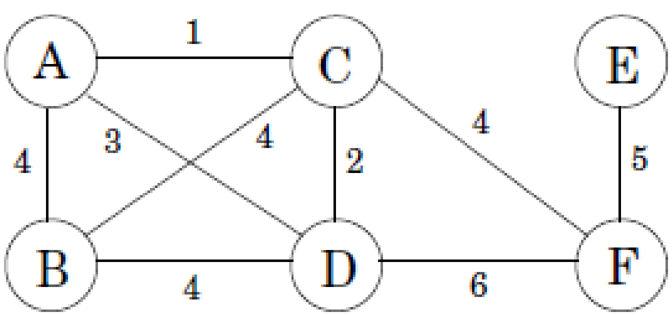
- Network a collection of computers by linking selected pairs of them.
- Each link has a maintenance cost.
- What is the cheapest network?

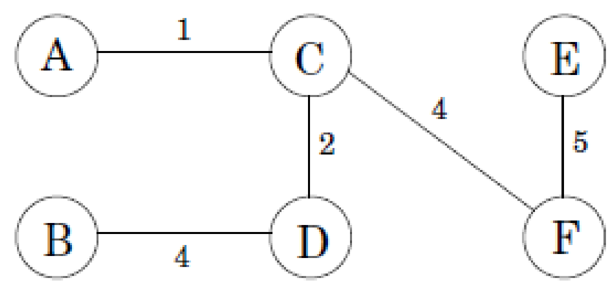


Minimum spanning tree

- Removing a cycle edge cannot disconnect a graph.
- So the solution must be connected and *acyclic* : undirected connected acyclic graphs = *trees*.
- Input: An undirected graph $G = (V, E)$, edge weights w_e
- Output: A tree $T = (V, E')$, with $E' \subseteq E$, that minimizes

$$weight(T) = \Sigma_{e \in E'} w_e$$



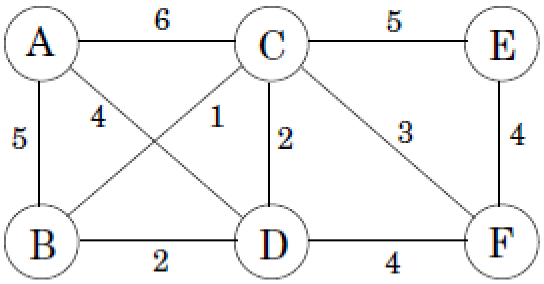


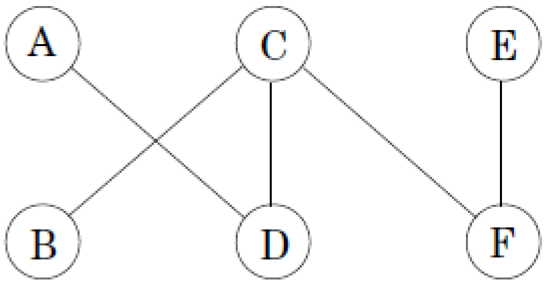
Trees

- A tree on n nodes has n - 1 edges.
- Any connected, undirected graph $G = (V, E)$ with $|E| = |V| - 1$ is a tree.
- An undirected graph is a tree if and only if there is a unique path between any pair of nodes.

Kruskal’s algorithm

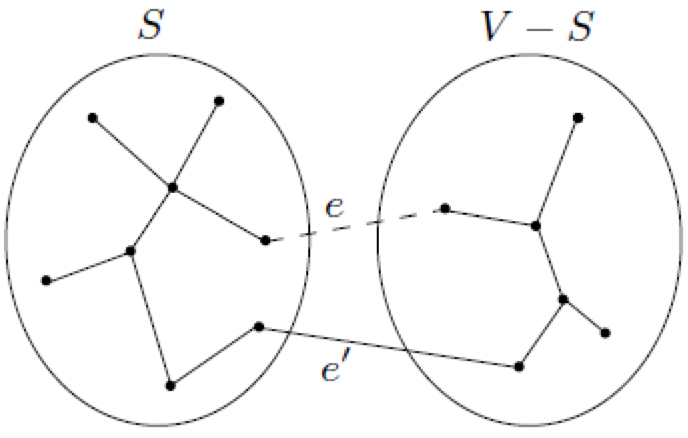
- Kruskal's minimum spanning tree algorithm starts with the empty graph and then selects edges from *E* according to the following rule.
- Repeatedly add the next *lightest* edge that doesn't produce a cycle.
- This is a *greedy algorithm.*





Cut property

- Suppose edges X are part of a minimum spanning tree 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.
- Pf) Let T be an MST that includes X.



If e is in T, done.

Otherwise, add e to T. It creates a cycle.

This cycle must have another edge e' across the cut (S, V-S).

Remove e'. Then, we have a new spanning tree $T' = T \cup \{e\} - \{e'\}$.

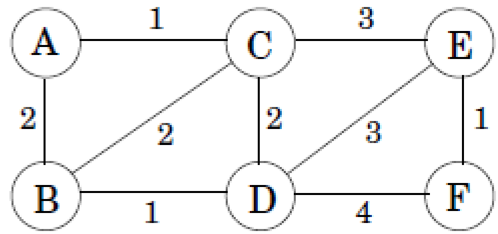
(why is T' a spanning tree?)

weight(T') = weight(T) +w(e) – w(e')

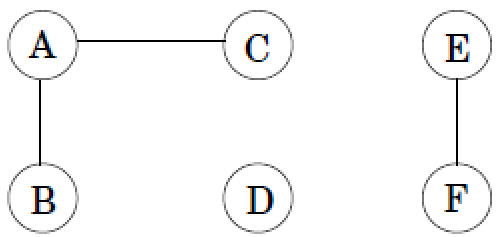
Since e is the lightest edge crossing the cut (S, V-S), w(e) $\leq$ w(e')

Thus, weight(T') $\leq$ weight(T).

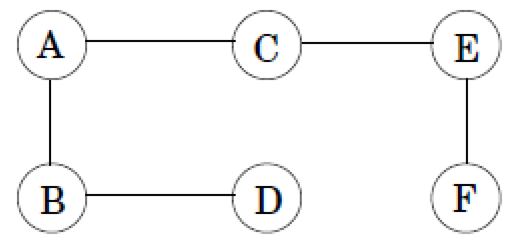
Since T is a MST, T' is also a MST.



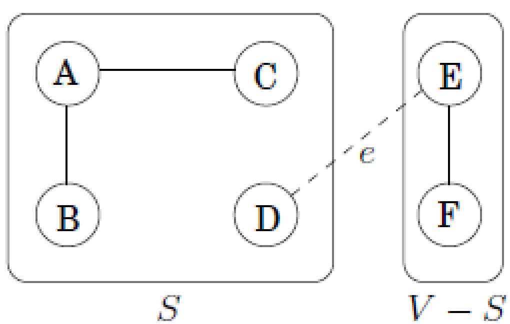
Edges X:



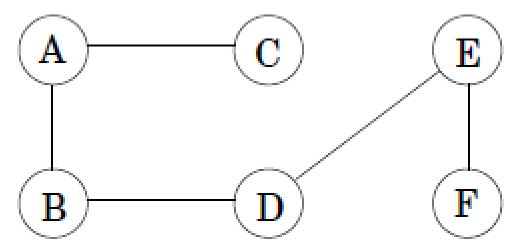
MST T:



The cut:



MST T':



Correctness of Kruskal's algorithm

- At any given moment, the edges already chosen form a partial solution, a collection of connected components (trees).
- The next edge e to be added connects two of these components; call them T_1 and T_2 .
- Since e is the lightest edge that doesn't produce a cycle, it is certain to be the lightest edge between T_1 and V- T_1
- Therefore, it satisfies the cut property.

Implementation

- Need to test each candidate edge u - v to see whether the endpoints u and v lie in different components, not producing a cycle.
- Need a *disjoint-set data structure* supporting the following :

  - makeset(x): create a singleton set containing just x.

  - find(x): to which set does x belong?

  - union(x, y): merge the sets containing x and y.

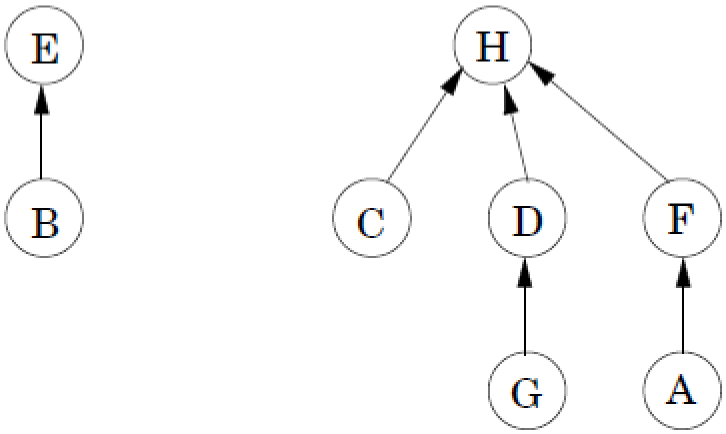
Kruskal's algorithm

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

for all u ∈ V:
   makeset(u)

X = {}
Sort the edges E by weight
for all edges {u, v} ∈ E, in increasing order of weight:
   if find(u) ≠ find(v):
      add edge {u, v} to X
      union(u, v)
```

- Uses |V| makeset, 2|E| find, |V|-1 union operations.

Disjoint-set data structure

- Store a set as a directed tree.
- Nodes of the tree are elements of the set, arranged in no particular order.
- Each has parent pointers π that eventually lead up to the root of the tree.
- The root is a *representative*, or *name*, for the set.
- The root has a parent pointer π pointing itself.
- Each node has *rank* representing the height of the subtree from the node.



Makeset and find

- *makeset* is a constant-time operation
- *find* follows parent pointers to the root of the tree : takes O(height of the tree).

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

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

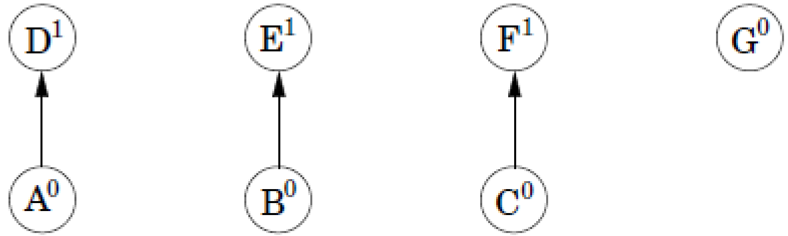
Union by rank

- Make the root of the shorter tree point to the root of the taller tree.
- Then, the overall height increases only if the two trees being merged are equally tall.
- Instead of explicitly computing heights of trees, we will use the *rank* numbers of their root nodes ***- union by rank.***

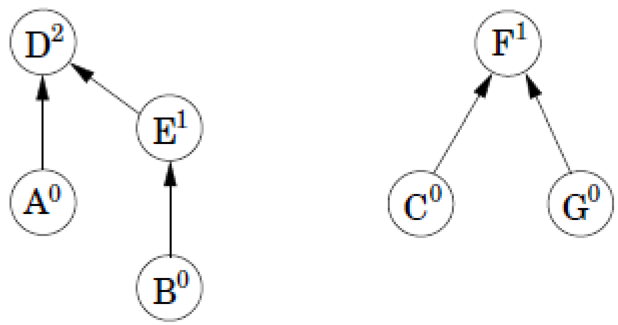
```
procedure union(x, y)
r_x = find(x)
r_y = find(y)
if r_x = r_y:   return
if rank(r_x) > rank(r_y):
    π(r_y) = r_x
else:
    π(r_x) = r_y
    if rank(r_x) = rank(r_y):  rank(r_y) = rank(r_y) + 1
```

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

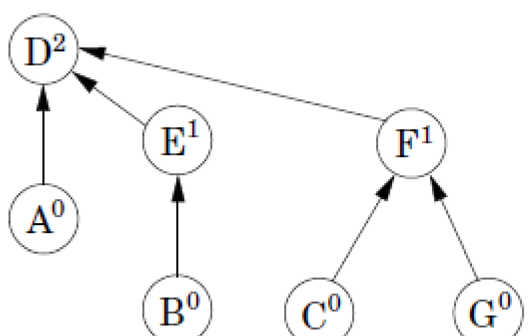
After $\texttt{union}(A, D), \texttt{union}(B, E), \texttt{union}(C, F)$:



After $\texttt{union}(C, G), \texttt{union}(E, A)$:



After $\texttt{union}(B, G)$:



Analysis

- By design, the rank of a node is exactly the height of the subtree rooted at that node.
- As you move up a path toward a root node, the rank values are strictly increasing.
- **Property 1 For any x, rank(x) < rank($\pi(x)$).**
- **Property 2 Any root node of rank k has at least 2^k nodes in its tree.**

  - Prove by induction
- **Property 3 If there are n elements overall, there can be at most $n/2^k$ nodes of rank k.**
- The maximum rank is log n.
- Therefore, all the trees have height $\leq \log n$, and this is an upper bound on the running time of find and union.

Analysis of Kruskal's algorithm

- Kruskal's algorithm uses $|V|$ makeset, $2|E|$ find, $|V|$-1 union operations.
- We need $O(|E| \lg |V|)$ to sort the edges. ($\lg |E| = \Theta(\lg |V|)$)
- $O(|E| \log |V|)$ for find and union operations.

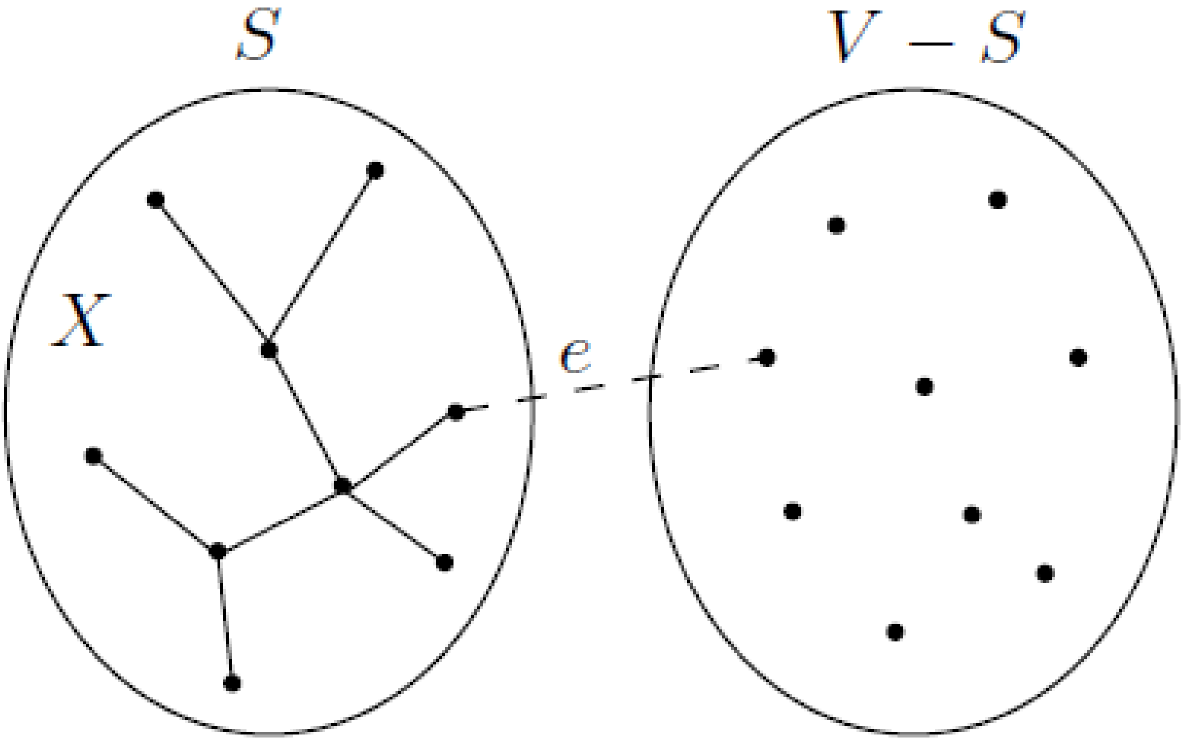
Greedy algorithm

- The cut property suggests that the following greedy schema works to find MST.

$X = \{\ \}$ (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 $V - S$
  $X = X \cup \{e\}$

- Prim's algorithm
  - the intermediate set of edges *X* always forms a subtree, and *S* is the set of this tree's vertices.

Prim's algorithm

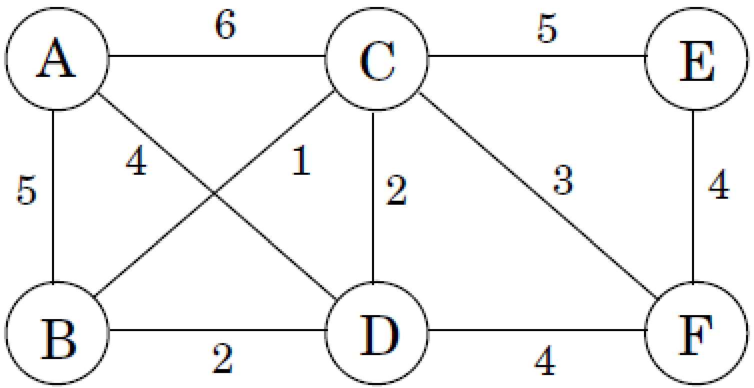


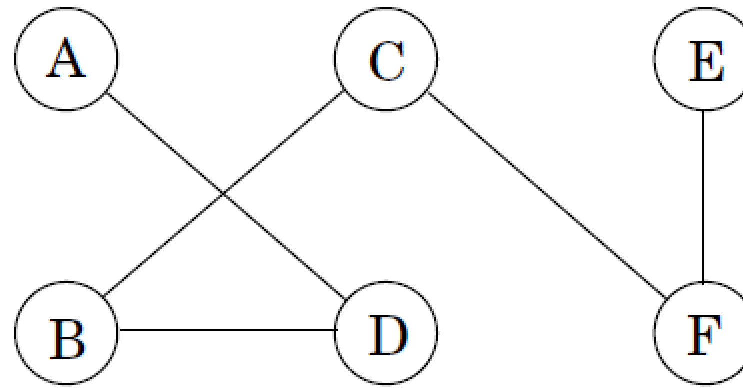
- the intermediate set of edges X always forms a subtree, and S is the set of this tree's vertices.
- Use *priority queue* to find the lightest edge between a vertex in S and a vertex outside S - grow S to include the vertex $v \notin S$ of smallest **cost** :

$$\mathbf{cost}(v) = \min_{u \in S} w(u, v)$$

Prim's algorithm

```
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)
```





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					$4/F$	