Graph Algorithms:
Minimum Spanning Tree (Ch 23)

Minimum spanning tree

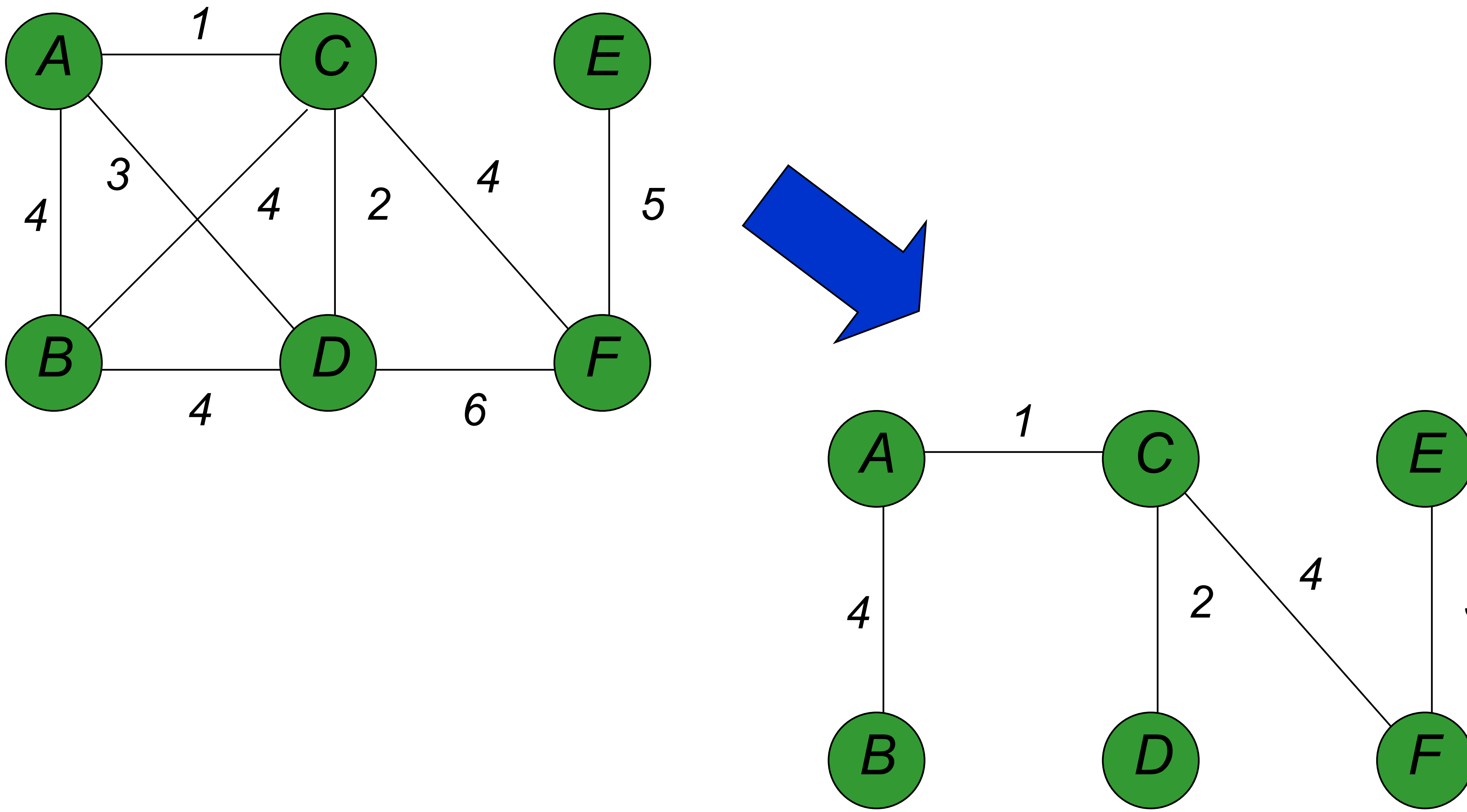
What is the lowest weight set of edges that connects all vertices of an undirected graph with positive weights

Input: An undirected, positive weight graph, G=(V,E)

Output: A tree T=(V,E') where E' $\subseteq$ E that minimizes

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

MST example



Applications?

Connectivity

- Networks (e.g. communications)
- Circuit design/wiring

hub/spoke models (e.g. flights, transportation)

Traveling salesman problem?

MST construction: Kruskal's method

- create a forest F (a set of trees), where each node in the graph is a separate (trivial) tree.
- create a set S containing all the links in the network.
- while S is nonempty and F is not yet spanned
  - remove a link with minimum cost from S
  - if that link connects two different trees, then add it to the forest, combining two trees into a single tree

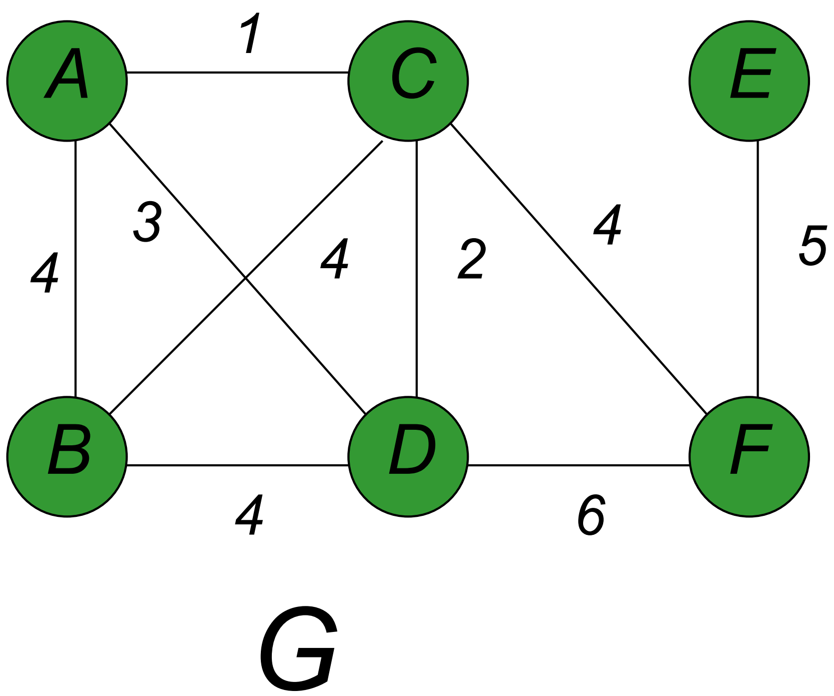
Greedy algorithm

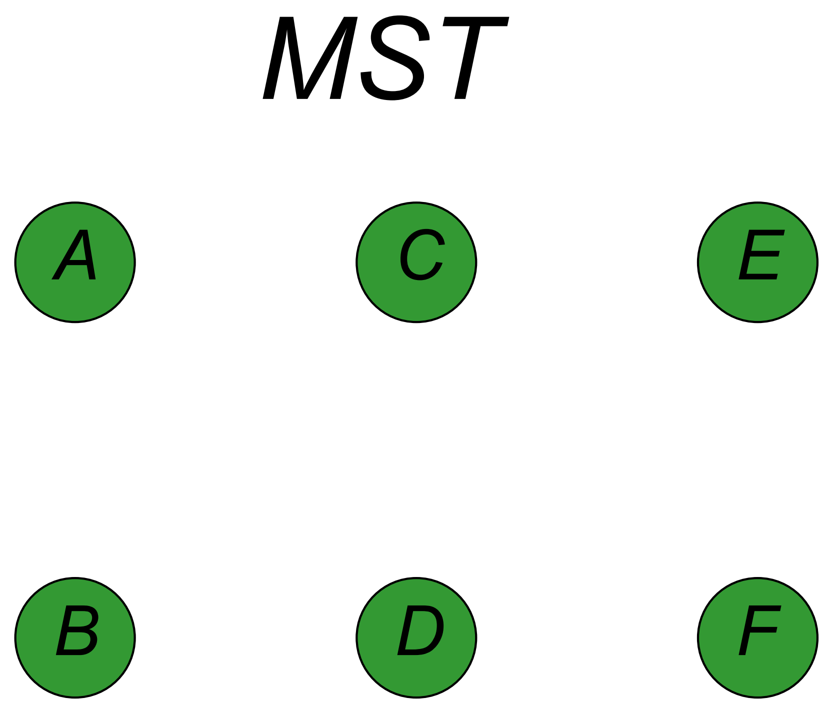
MST construction: Kruskal's method

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)   // Test if u and v belong to same tree
        A = A ∪ {(u, v)}
        UNION(u, v)    // Combine two trees
return A
```

Kruskal's algorithm

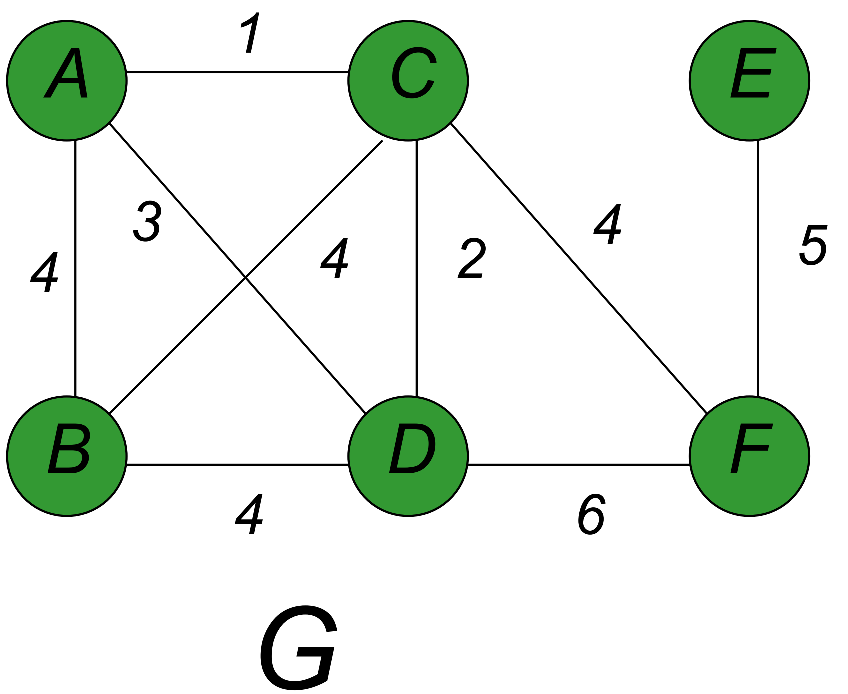
Add smallest edge that connects two sets not already connected

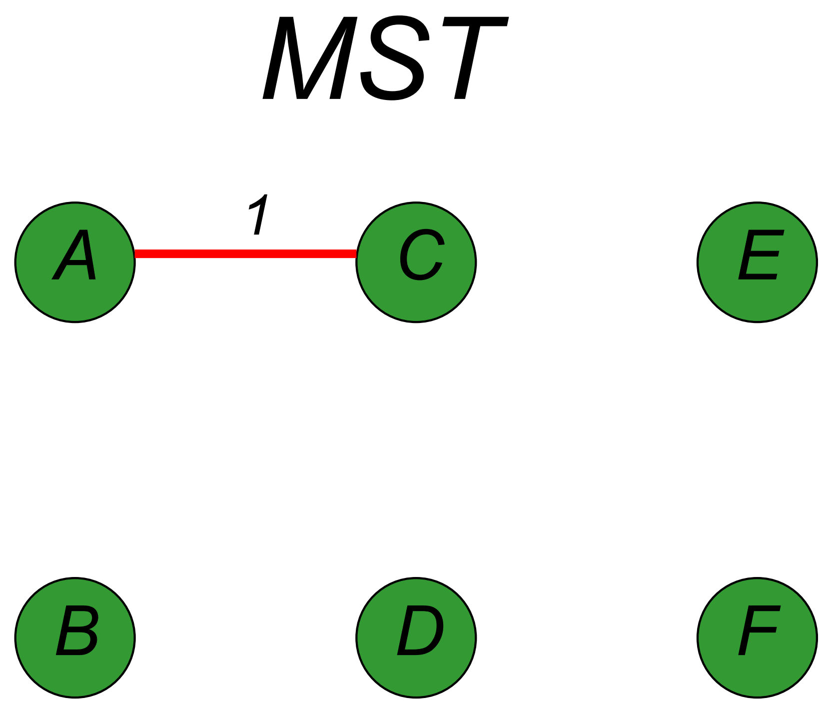




Kruskal's algorithm

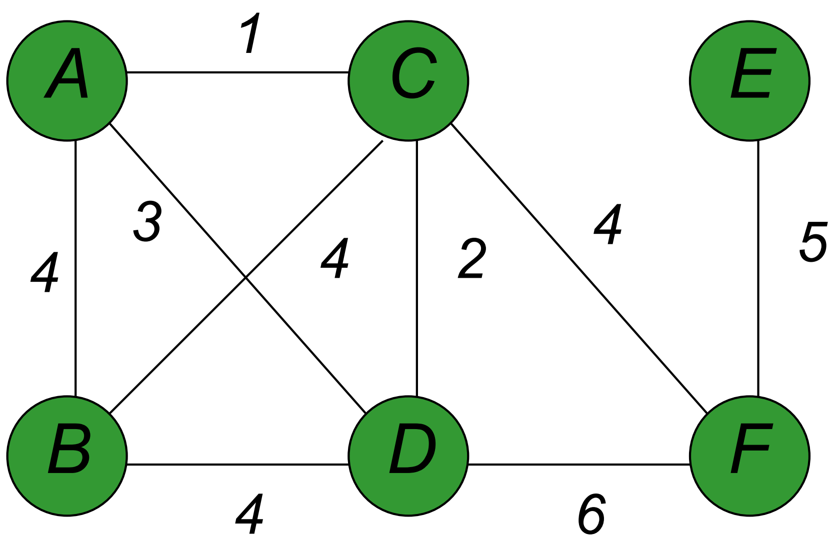
Add smallest edge that connects two sets not already connected





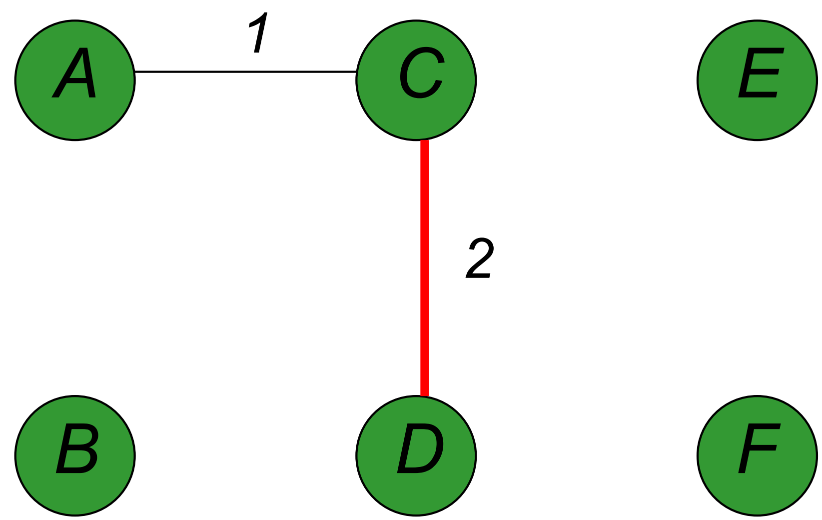
Kruskal's algorithm

Add smallest edge that connects two sets not already connected



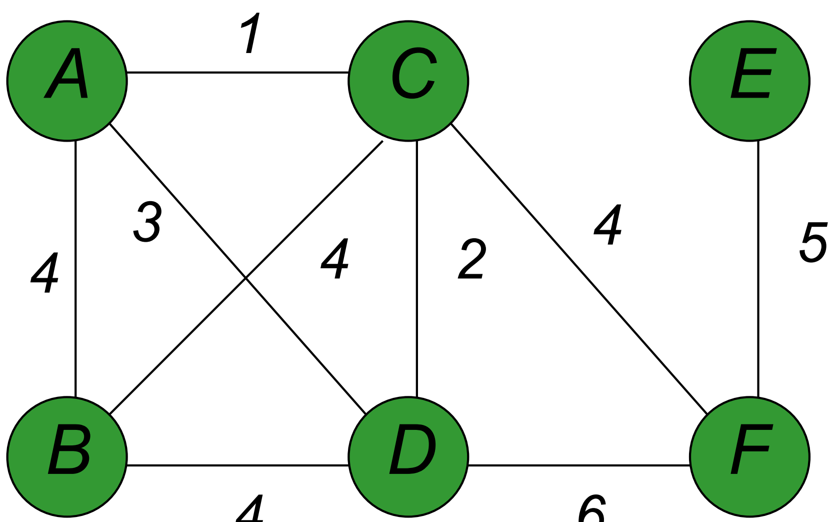
G

MST



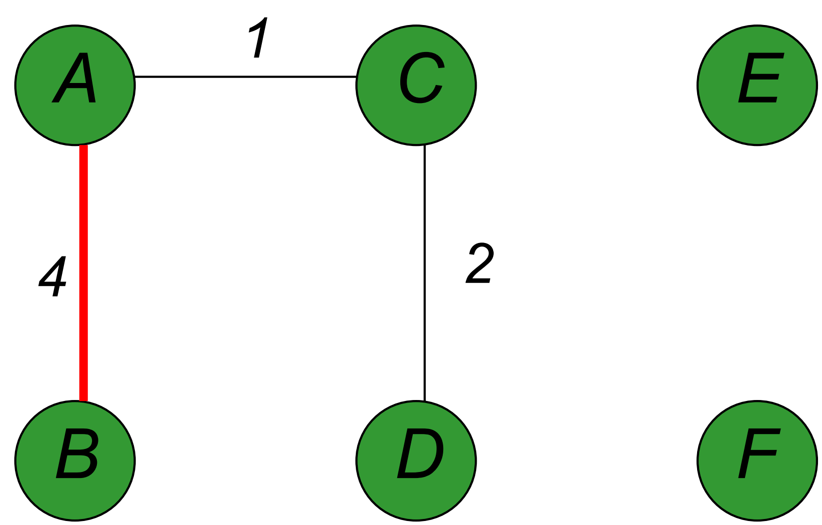
Kruskal's algorithm

Add smallest edge that connects two sets not already connected



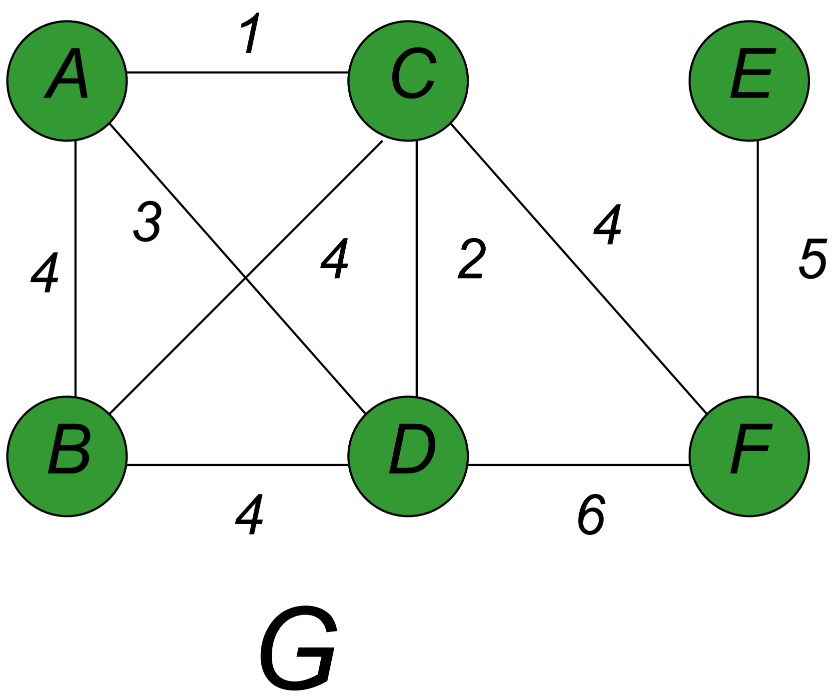
G

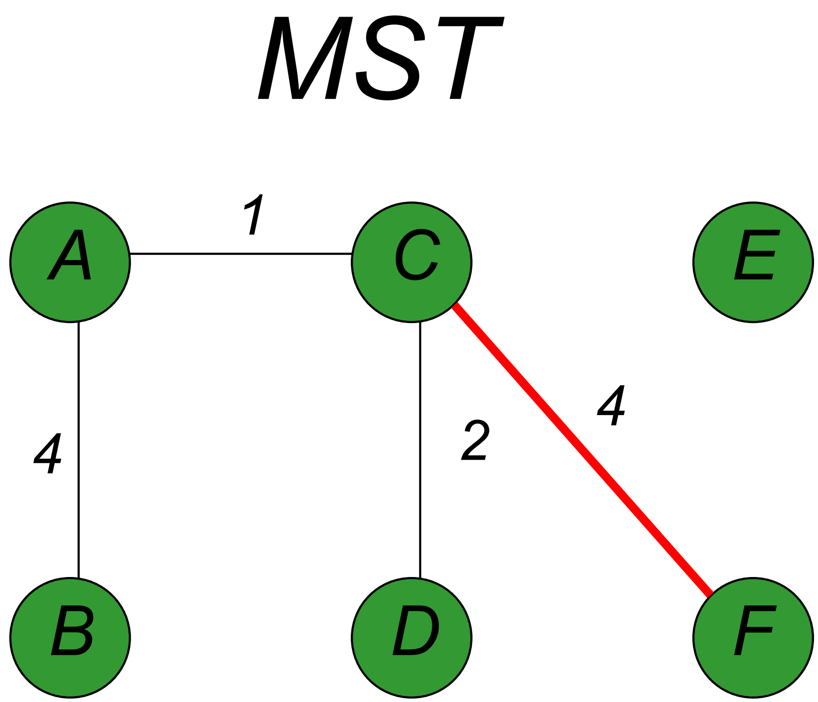
MST



Kruskal’s algorithm

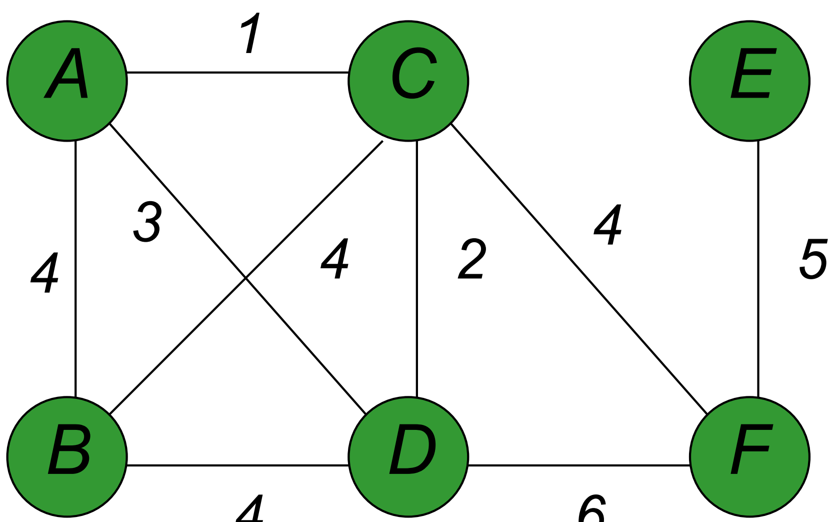
Add smallest edge that connects two sets not already connected





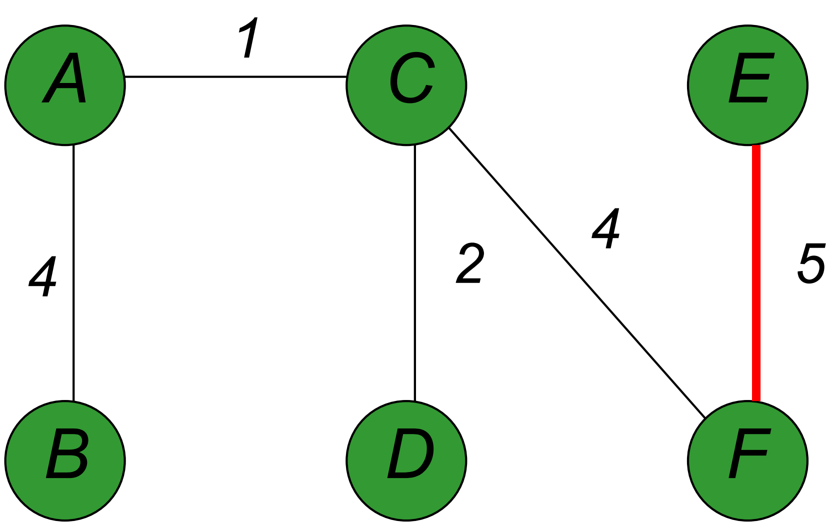
Kruskal's algorithm

Add smallest edge that connects two sets not already connected



G

MST



Kruskal Alg's Complexity

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)      O(1)
        A = A ∪ {(u, v)}
        UNION(u, v)                   O(lg |V|) per iteration (amortized)
return A
```

Sorting: $O(|E| \lg |E|)$

For loop: $O(|E| \lg |E|)$

Thus, total time= $O(|E| \lg |E|)$

$= O(|E| \lg |V|)$ since $(\lg |E|) = O(\lg |V|)$ as $|E| < |V|^2$

MST construction: Prim's method

- Initialize a tree with a single node, chosen arbitrarily from the network.
- Grow the tree by one edge: of the edges that connect the tree to nodes not yet in the tree, find the minimum-cost link, and transfer it to the tree.
- Repeat step 2 (until all nodes are in the tree).

Greedy Algorithm

Prim's Algorithm

MST-PRIM(G, w, r)

for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$ *Parent*
$r.key = 0$
$Q = G.V$ *Priority queue*
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$

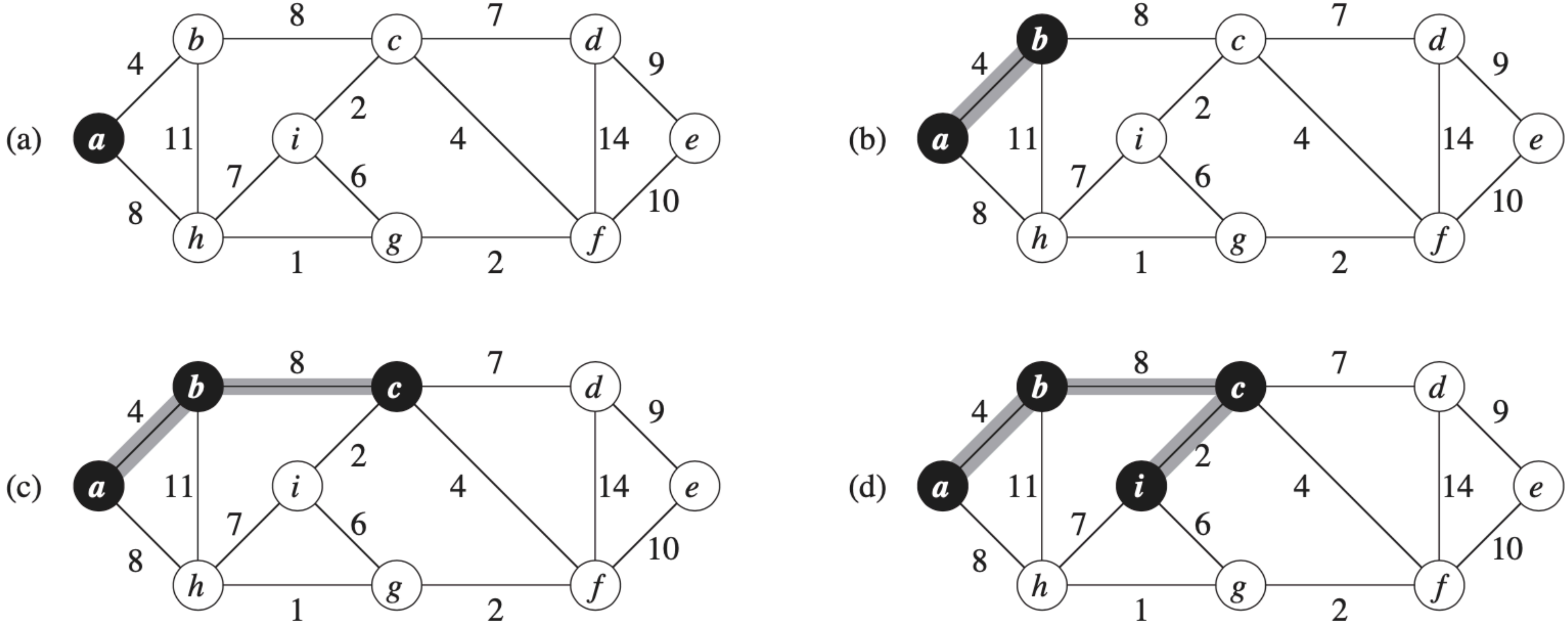
Set of vertices is partitioned into two subsets: V-Q and Q.

Set V-Q indicates current MST.

Extract u in Q incident on least weight edge between V-Q and Q. And add u in V-Q.

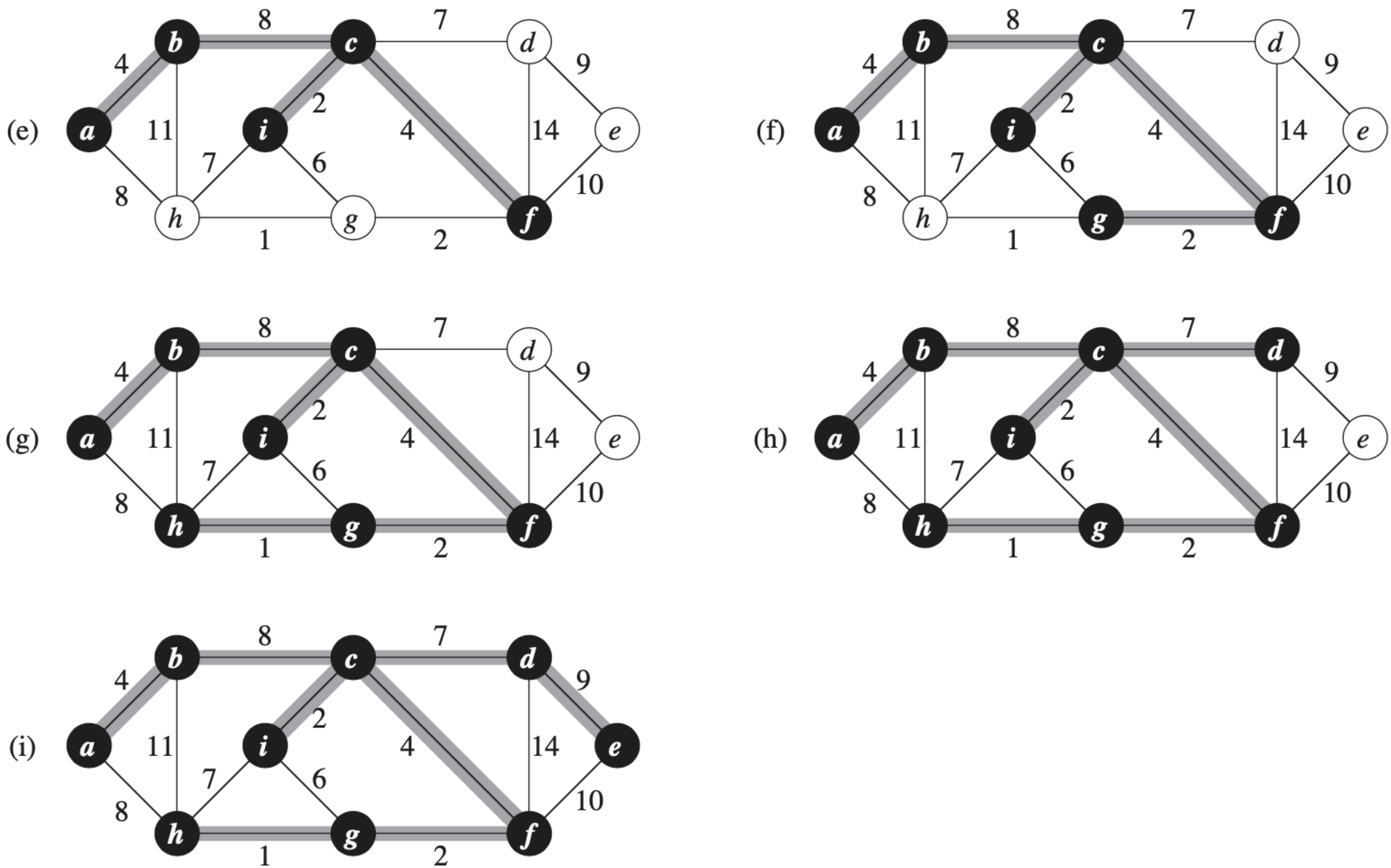
update the key and parent of each node v adjacent to u but not in the tree.

Prim's Algorithm



V-Q indicates current MST and is shaded in black.

Prim's Algorithm



Prim's Algorithm: Time Complexity

```
MST-PRIM(G, w, r)
1   for each u ∈ G.V
2       u.key = ∞
3       u.π = NIL
4   r.key = 0
5   Q = G.V
6   while Q ≠ ∅
7       u = EXTRACT-MIN(Q)
8       for each v ∈ G.Adj[u]
9           if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

Depends on how we implement priority queue

Buildheap:O(|V|)

|V| calls to Extract-Min, each of O(lg|V|)

lines 8–11 executes O(|E|) times altogether, since the sum of the lengths of all adjacency lists is 2|E|

Line 11 involves Decrease-Key: O(lg|V| for binary heap and O(1) for fibonacci heap.

Using binary heap, *time=* $O(|V| \lg |V|) + O(|E| \lg |V|) = O(|E| \lg |V|)$

Using fibonacci heap, *time=* $O(|V| \lg |V| + |E|)$

Prim's Algorithm: Correctness

- Straightforward using induction