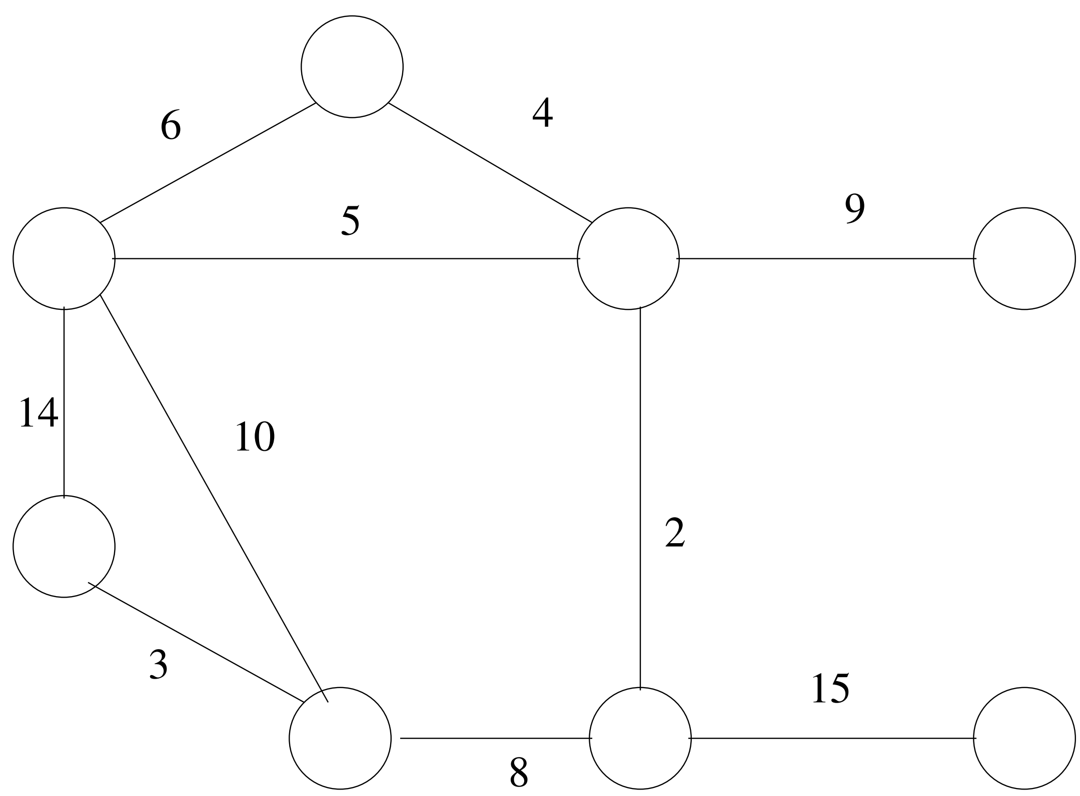
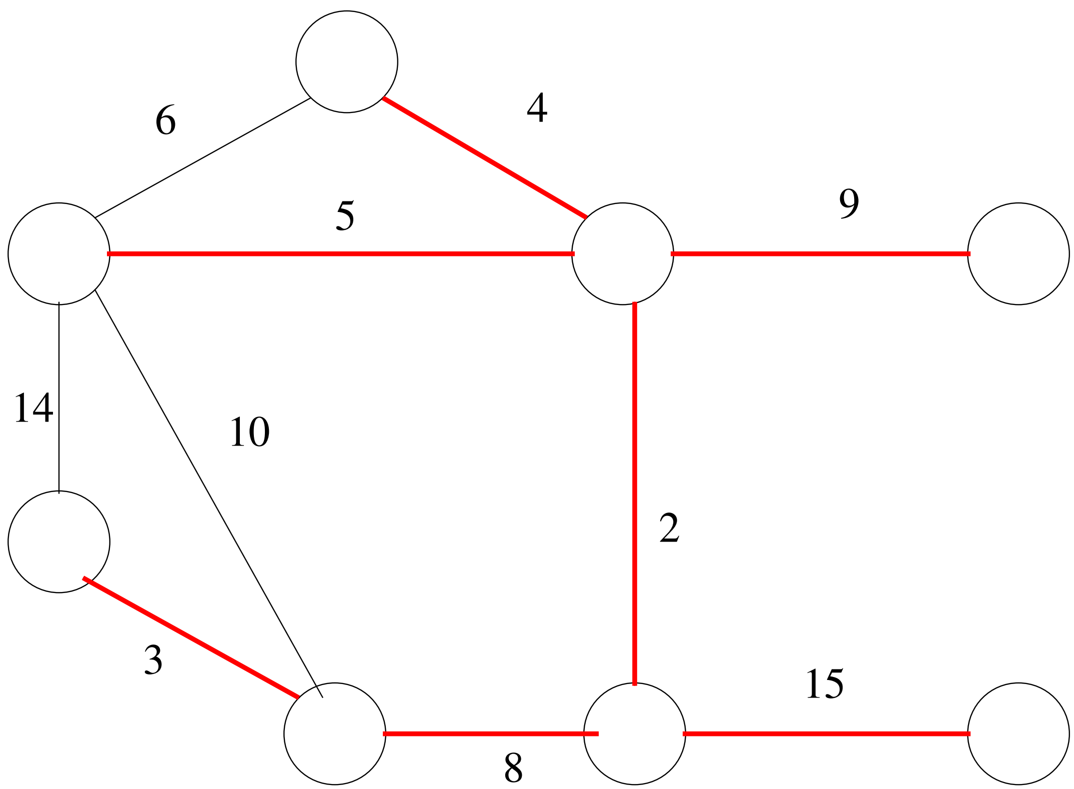
Minimum Spanning Trees

- $G = (V, E)$ is an undirected graph with non-negative edge weights $w : E \rightarrow Z^+$
- We assume wlog that edge weights are distinct
- A spanning tree is a tree with $V - 1$ edges, i.e. a tree that connects all the vertices.
- The total cost (weight) of a spanning tree T is defined as $\Sigma_{e \in T}\, w(e)$
- A minimum spanning tree is a tree of minimum total weight.



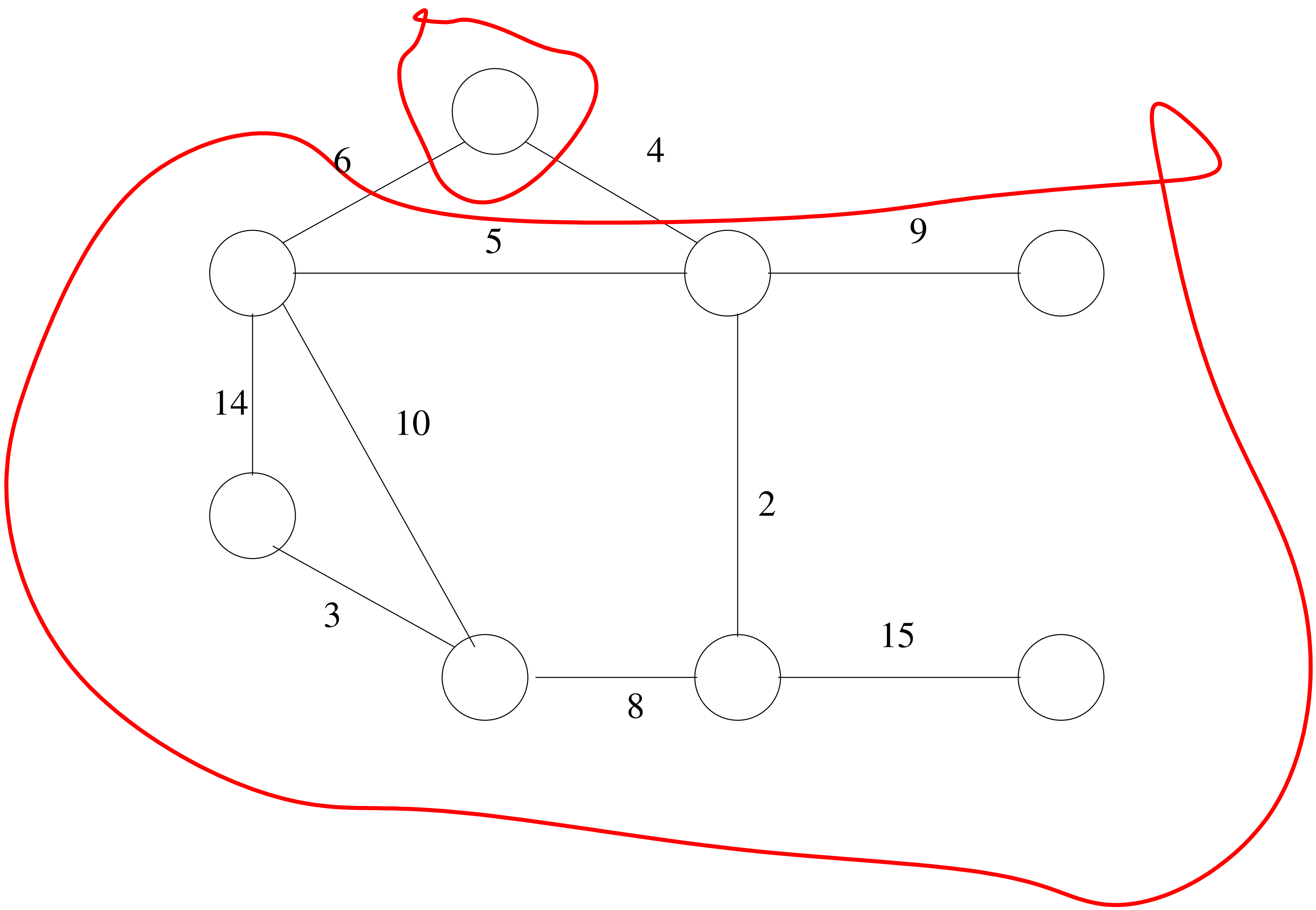
Minimum Spanning Trees

- $G = (V, E)$ is an undirected graph with non-negative edge weights $w : E \rightarrow Z^+$
- We assume wlog that edge weights are distinct
- A **spanning tree** is a tree with $V - 1$ edges, i.e. a tree that connects all the vertices.
- The total cost (weight) of a spanning tree T is defined as $\Sigma_{e \in T}\, w(e)$
- A **minimum spanning tree** is a tree of minimum total weight.



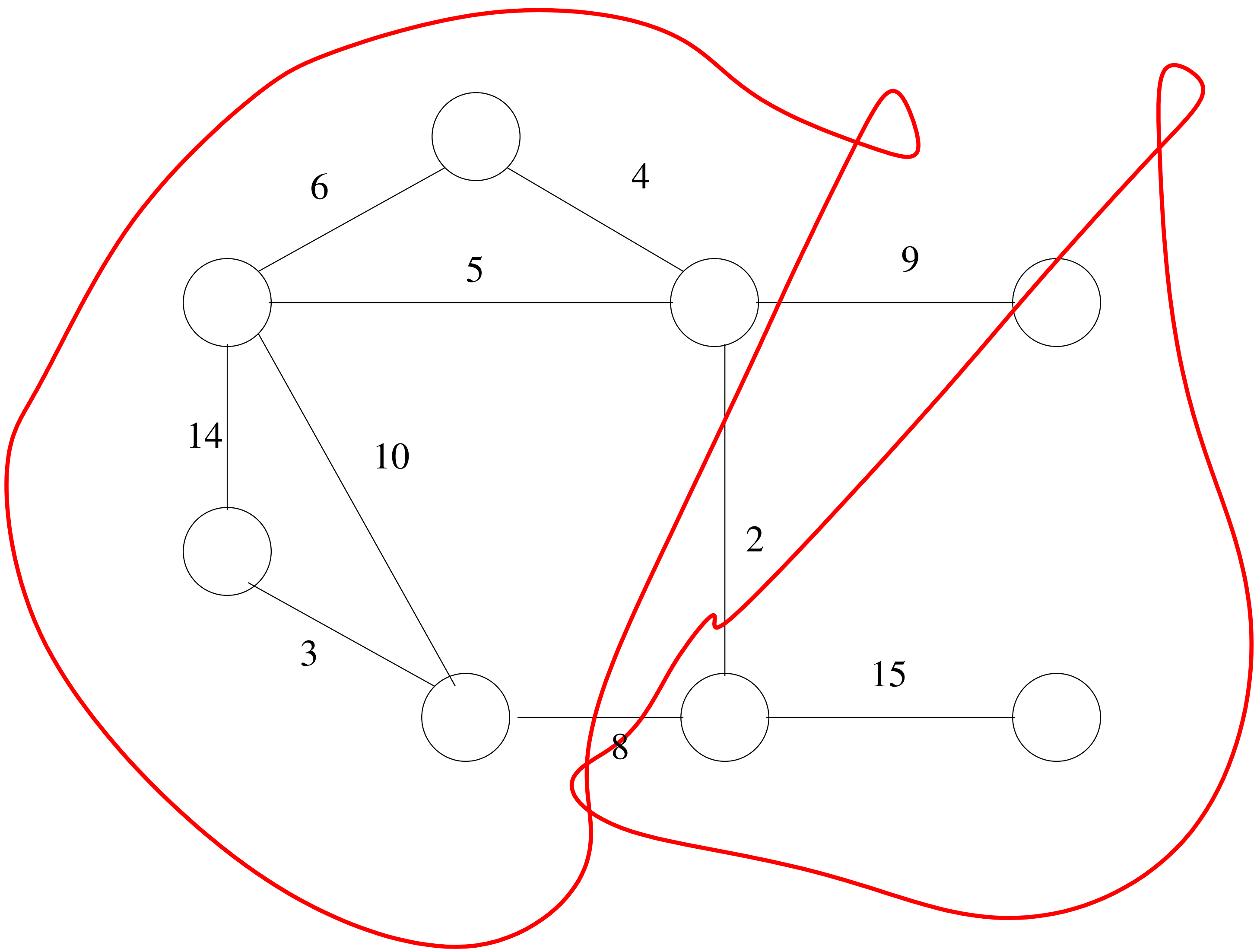
Cuts

- **A cut in a graph is a partition of the vertices into two sets S and T.**
- **An edge (u, v) with $u \in S$ and $v \in T$ is said to cross the cut.**



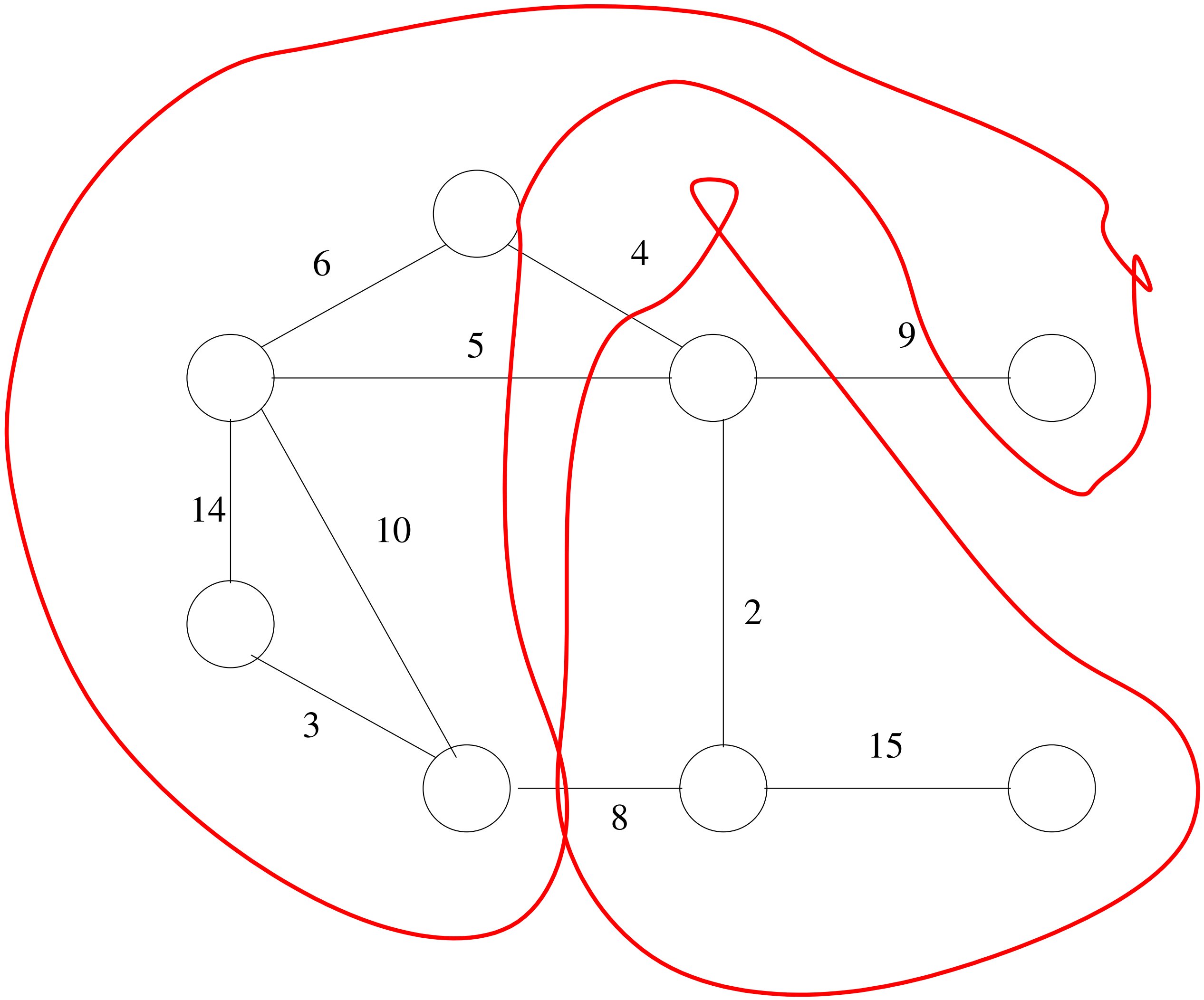
Cuts

- A **cut** in a graph is a partition of the vertices into two sets S and T.
- An edge (u, v) with $u \in S$ and $v \in T$ is said to **cross the cut**.



Cuts

- **A cut in a graph is a partition of the vertices into two sets S and T.**
- **An edge (u, v) with $u \in S$ and $v \in T$ is said to cross the cut.**



Greedy Property

Recall that we assume all edges weights are unique.

Greedy Property: **The minimum weight edge crossing a cut is in the minimum spanning tree.**

Proof Idea: **Assume not, then remove an edge crossing the cut and replace it with the minimum weight edge.**

Restatement Lemma: **Let $G = (V, E)$ be an undirected graph with edge weights w. Let $A \subseteq E$ be a set of edges that are part of a minimum spanning tree. Let (S, T) be a cut with no edges from A crossing it. Then the minimum weight edge crossing (S, T) can be added to A.**

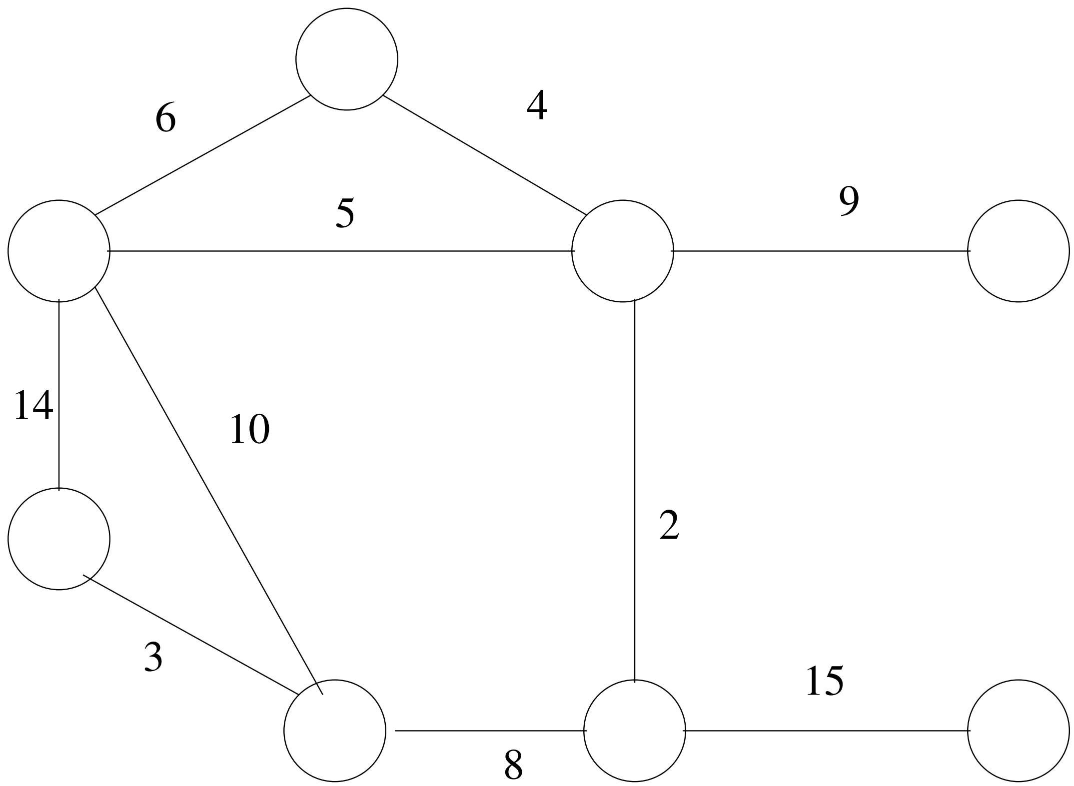
Algorithm Idea: **Repeatedly choose an edge according to the Lemma, add to MST.**

Challenge: **Finding the edge to add.**

Two standard algorithms:

- **Kruskal – consider the edges in increasing order of weight**
- **Prim – start at one vertex and grow the tree.**

Example: Run both agorithms



Kruskal's Algorithm: detailed implementation

Idea: **Consider edges in increasing order.**

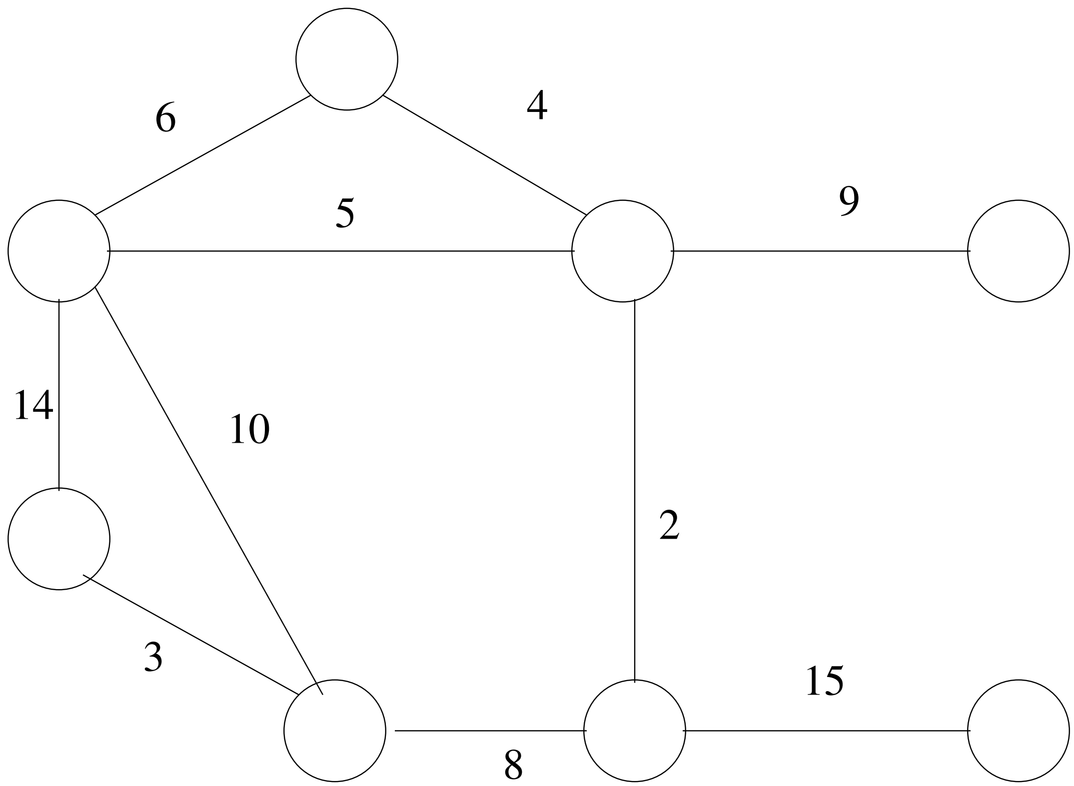
Need: **a data structure to maintain the sets of vertices in each component of the current forrest**

- MAKE-SET(v) **puts** v **in a set by itself**
- FIND-SET(v) **returns the name of** v**'s set**
- UNION(u, v) **combines the sets that** u **and** v **are in**

MST-Kruskal(G, w)

$A \leftarrow \emptyset$
for each vertex $v \in V[G]$
  do MAKE-SET(v)
sort the edges of E **into nondecreasing order by weight** w
for each edge $(u, v) \in E$**, taken in nondecreasing order by weight**
  do if FIND-SET(u) $\neq$ FIND-SET(v)
    then $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
return A

Example



Prim's Algorithm

Idea: Grow the MST from one node going out

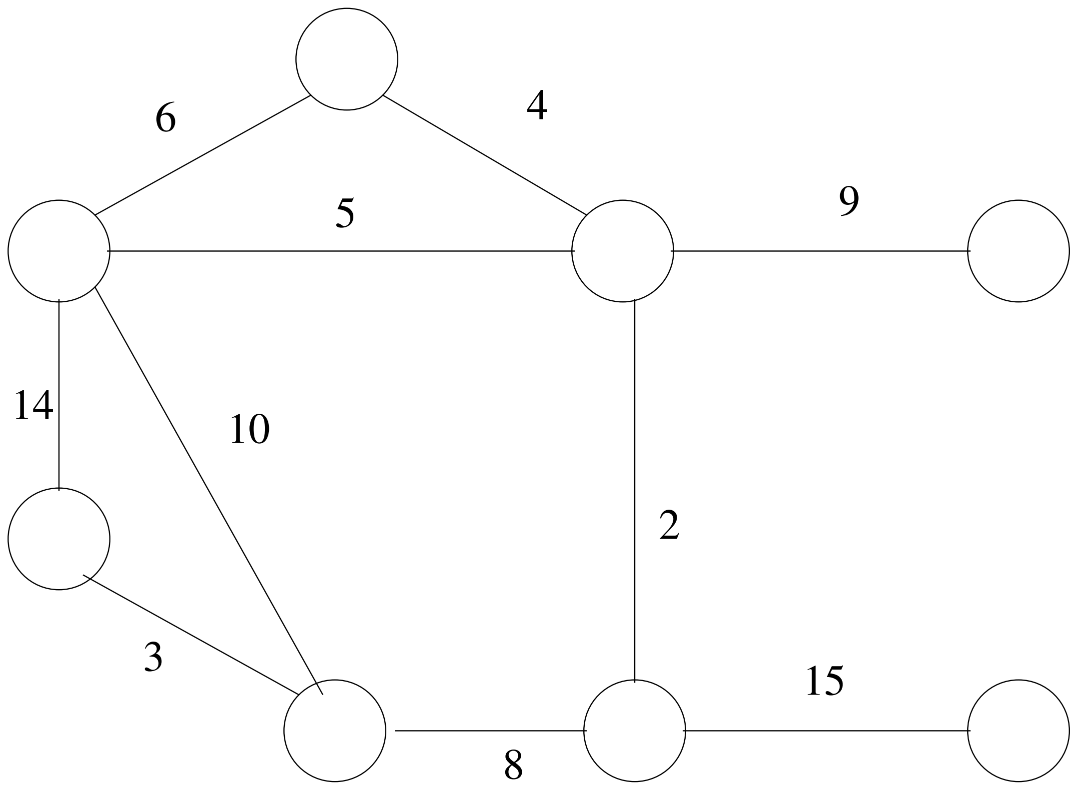
Need: a data structure to maintain the edges crossing the cut, and choose minimum. We will maintain, for each vertex, the minimum weight incident edge crossing the cut

- INSERT(v) puts v in the structure
- EXTRACT-MIN() finds and returns the node with minimum key value
- DECREASE-KEY(v, w) updates (decreases) the key of v

MST-Prim(G, w, r)

for each $u \in V[G]$
    do $key[u] \leftarrow \infty$
        $\pi[u] \leftarrow$ NIL
$key[r] \leftarrow 0$
$Q \leftarrow V[G]$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        for each $v \in Adj[u]$
            do if $v \in Q$ **and** $w(u, v) < key[v]$
                then $\pi[v] \leftarrow u$
                    $key[v] \leftarrow w(u, v)$

Example



Baruvka's Algorithm

- Repeat
  - Every node picks its minimum incoming edge and adds to the spanning tree T
  - Contract all edges in T

- How much progress is made?
- How implement an iteration?
- Total running time?

Baruvka's Algorithm

- **Repeat**
  - **Every node picks its minimum incoming edge and adds to the spanning tree T**
  - **Contract all edges in T**

- **How much progress is made?**
- **How implement an iteration?**
- **Total running time?**

$$T(n, m) = T(n/2, m - n) + O(n + m)$$

Problem: **Edges don't decrease fast enough**

Eliminating Edges

- **The heaviest edge on any cycle is not in the MST.**
- **Let F be a forest**
- **Let $w_F(u, v)$ be the maximum weight of an edge on the path from u to v in F (or ∞ if the path does not exist.**
- **Edge (u, v) is F-heavy if $w(u, v) > w_F(u, v)$ and F-light otherwise.**

Claim: Let F be any forest, let (u, v) be any edge. If (u, v) is F-heavy, then (u, v) is not in the MST.

Ideas

- **It is good to eliminate F-heavy edges**
- **The MST T would let us eliminate all non-MST edges.**
- **What can an F that is not an MST eliminate**

Fact to accept without proof: **Given G and a forrest F, we can eliminate all F-heavy edges in** $O(n+m)$ **time (spanning tree verification).**

Algorithm $MST(G)$

1. Run 3 Bruvka phases to get G' . Let C be the contracted edges.
2. Let G'' be G' with each edge included with prob. $1/2$.
3. Recursively compute $F'' = MST(G'')$.
4. Identify the F”-heavy edges in G' . Delete them to obtain G''' .
5. Recursively compute $F''' = MST(G''')$
6. Return $F''' \cup C$

Note: The recursion bottons out on a graph with $O(1)$ nodes.

Key Lemma: Let H be a subgraph of G where each edge is included with probability p . Let F be a Minimum Spanning Forest of H . Then the expected number of F-light edges in G is at most n/p .

Recurrence $T(n, m) \leq O(n+m) + T(n/8, m/2) + T(n/8, n/4) = O(n+m)$