

Minimum Spanning Trees

CMSC 420: Lecture 11

Following Dave Mount's
lecture in his lecture notes.

Skew Heaps

- Self-adjusting version of leftist heaps
- Don't store npl (or any other auxiliary information at the nodes)
- **Difference:**
 - *always* swap the left & right subtrees at each step of meld
 - old rightmost path becomes new leftmost path
- Can show (beyond the scope of this class) that a series of m *insert*, *findmin*, *meld* operations take $O(m \log n)$ time.
 - like splay trees, each operation takes $O(\log n)$ amortized time.

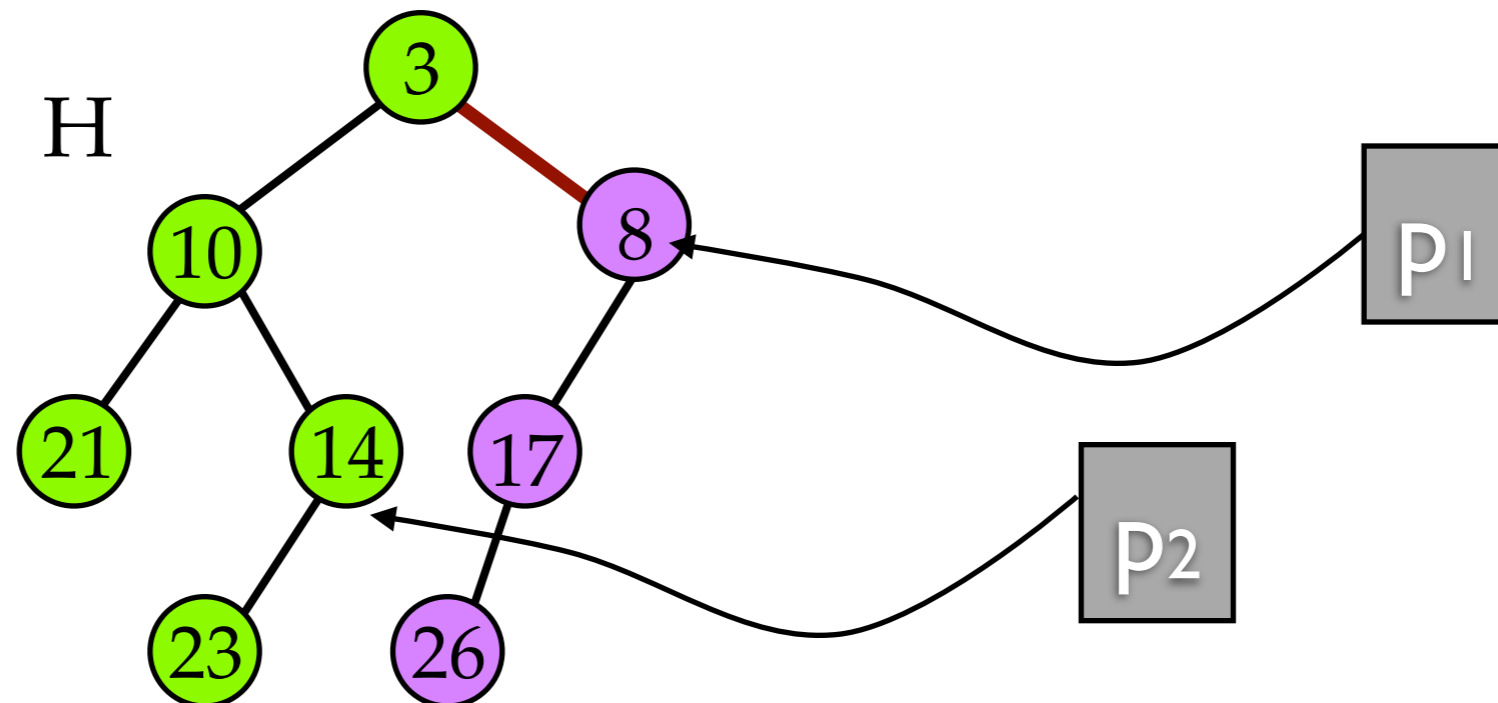
Decrease Key

decreasekey(pointer, d): subtract d from the key at the node pointed to by *pointer*.

- How would you implement this to run in $O(\log n)$ time?
 1. Subtract d from the key @ *pointer*
 2. sift up

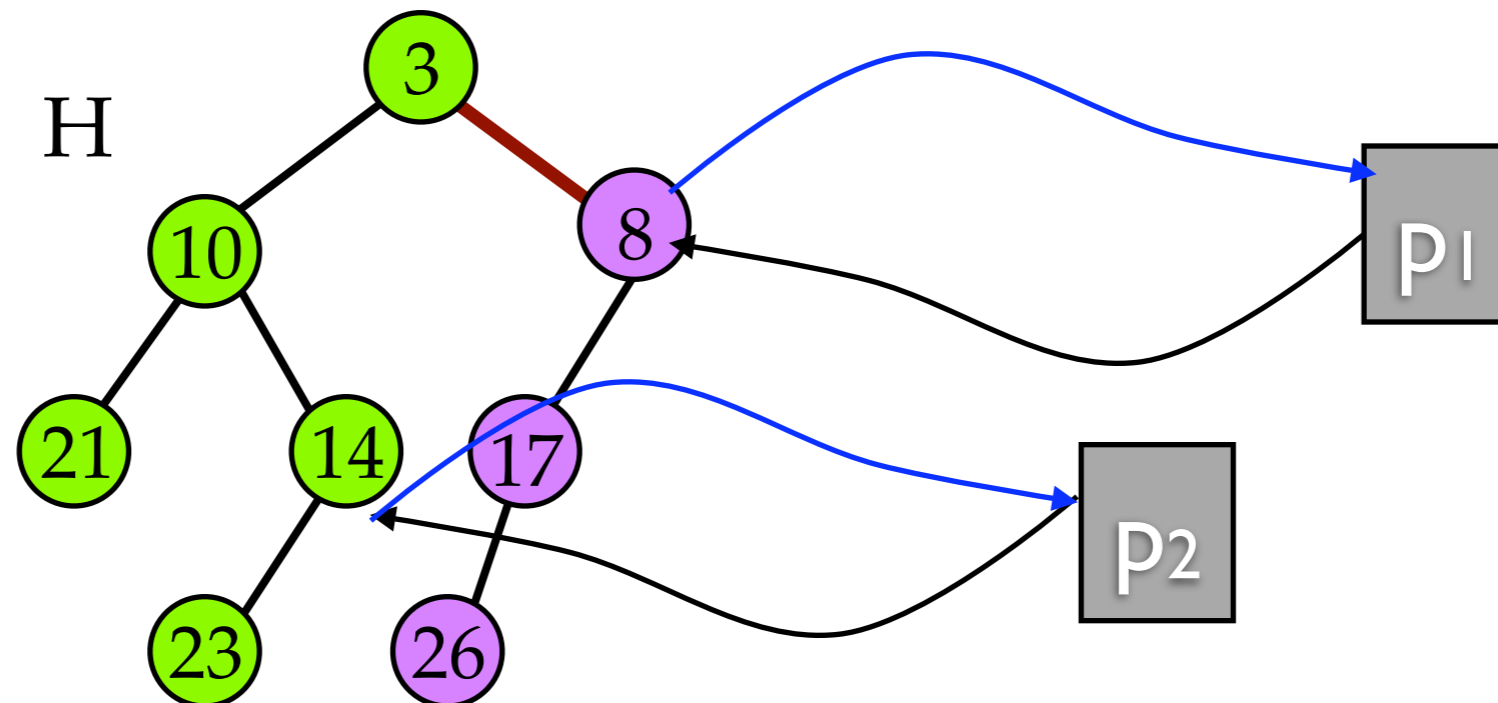
Handling Moving References

- **Problem:** Suppose you have a heap and are also keeping pointers to objects stored in the heap (to use for *delete* & *decrease_key*)
- When you sift up, nodes are going to move around, and the pointers will become invalid...
- How would you solve this?

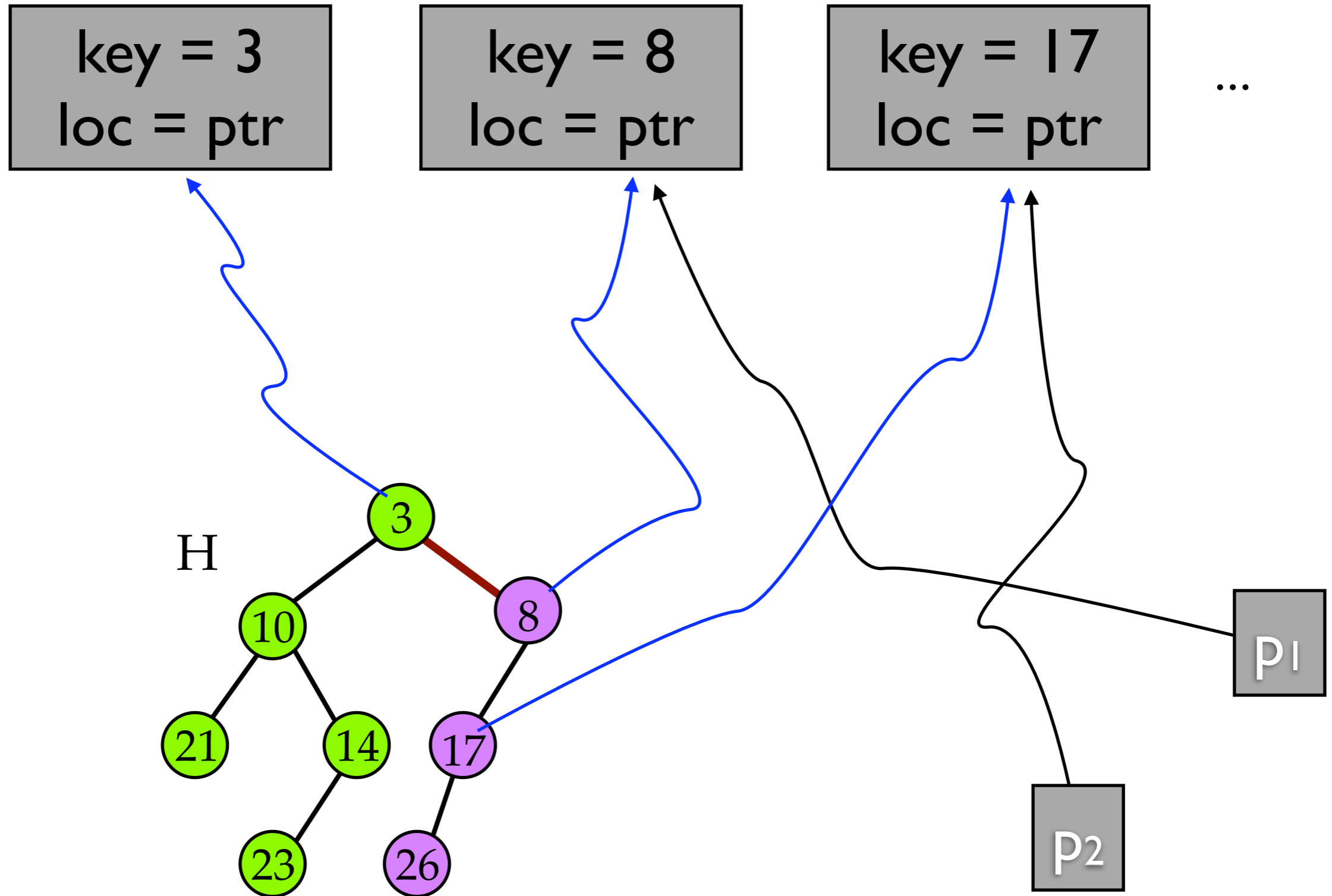


Handling Moving References

- **Problem:** Suppose you have a heap and are also keeping pointers to objects stored in the heap (to use for *delete* & *decrease_key*)
- When you sift up, nodes are going to move around, and the pointers will become invalid...
- How would you solve this?



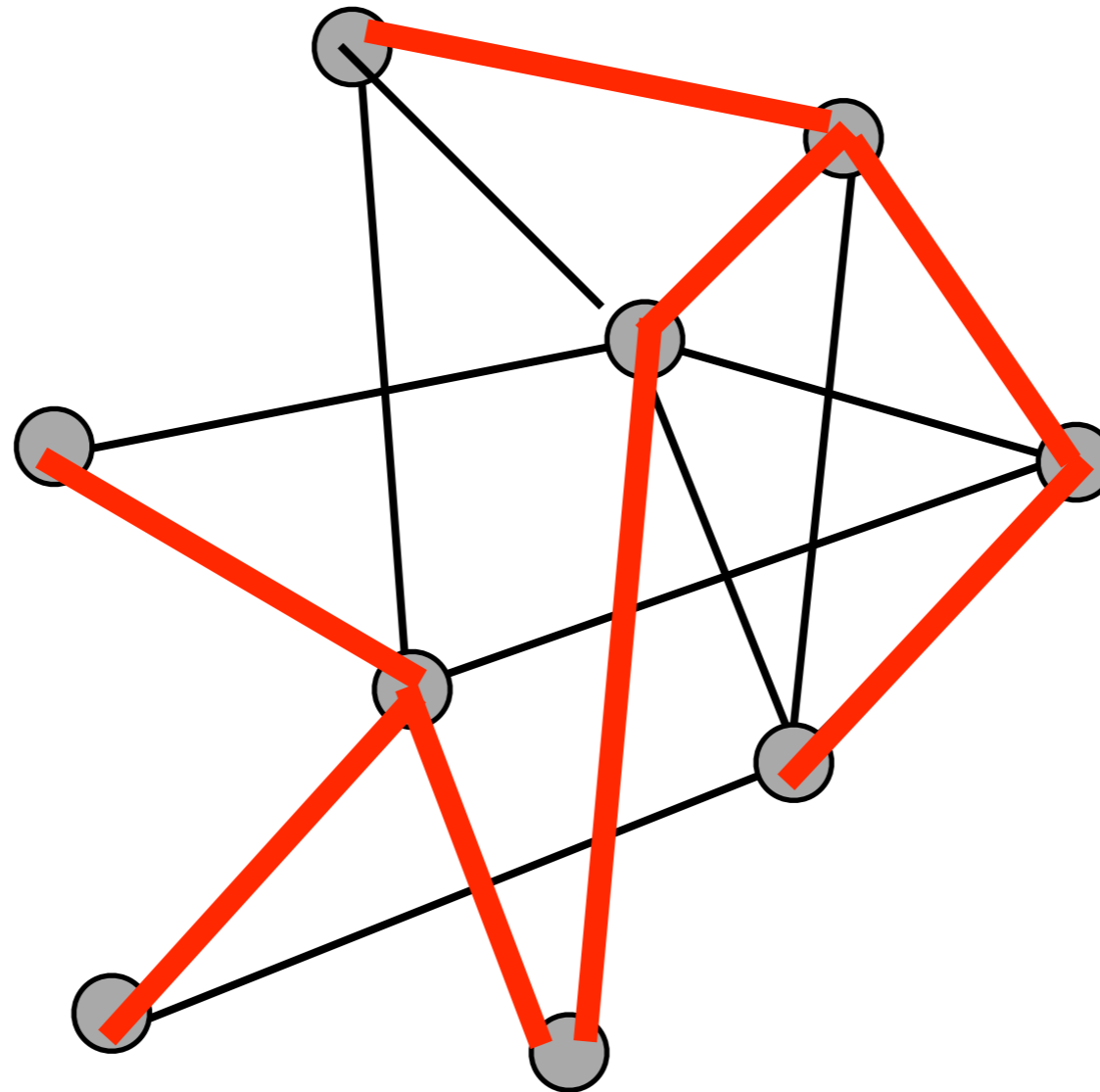
Handling Moving References



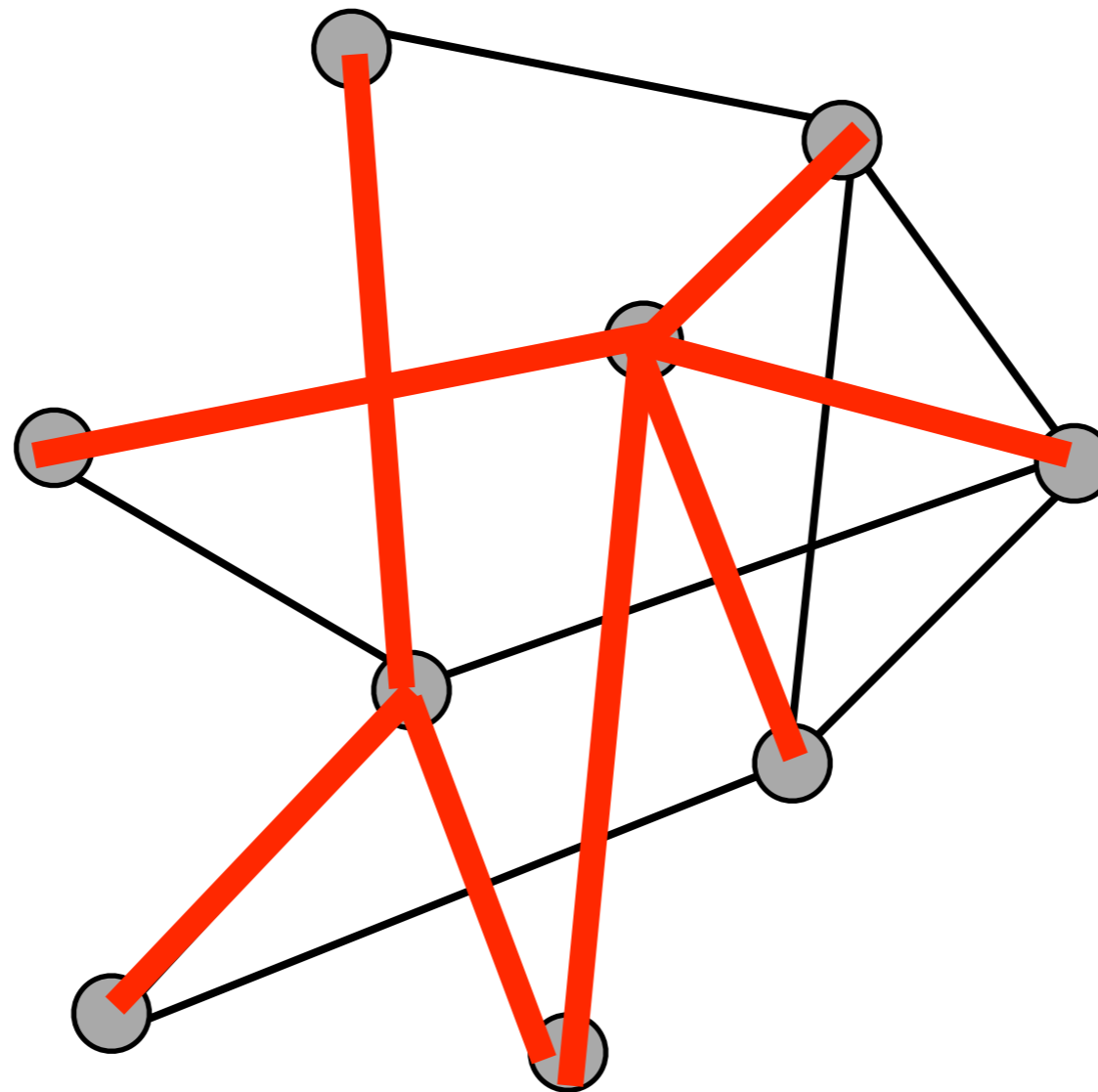
Spanning Trees

- Given a graph $G = (V, E)$
- A spanning tree is a connected, acyclic subset of edges that connects together all the vertices.
- To connect all vertices with the fewest number of edges, it never makes sense to include a cycle.
- If graph is weighed by putting real numbers on each edge, then we can look for a minimum cost spanning tree (called the minimum spanning tree)

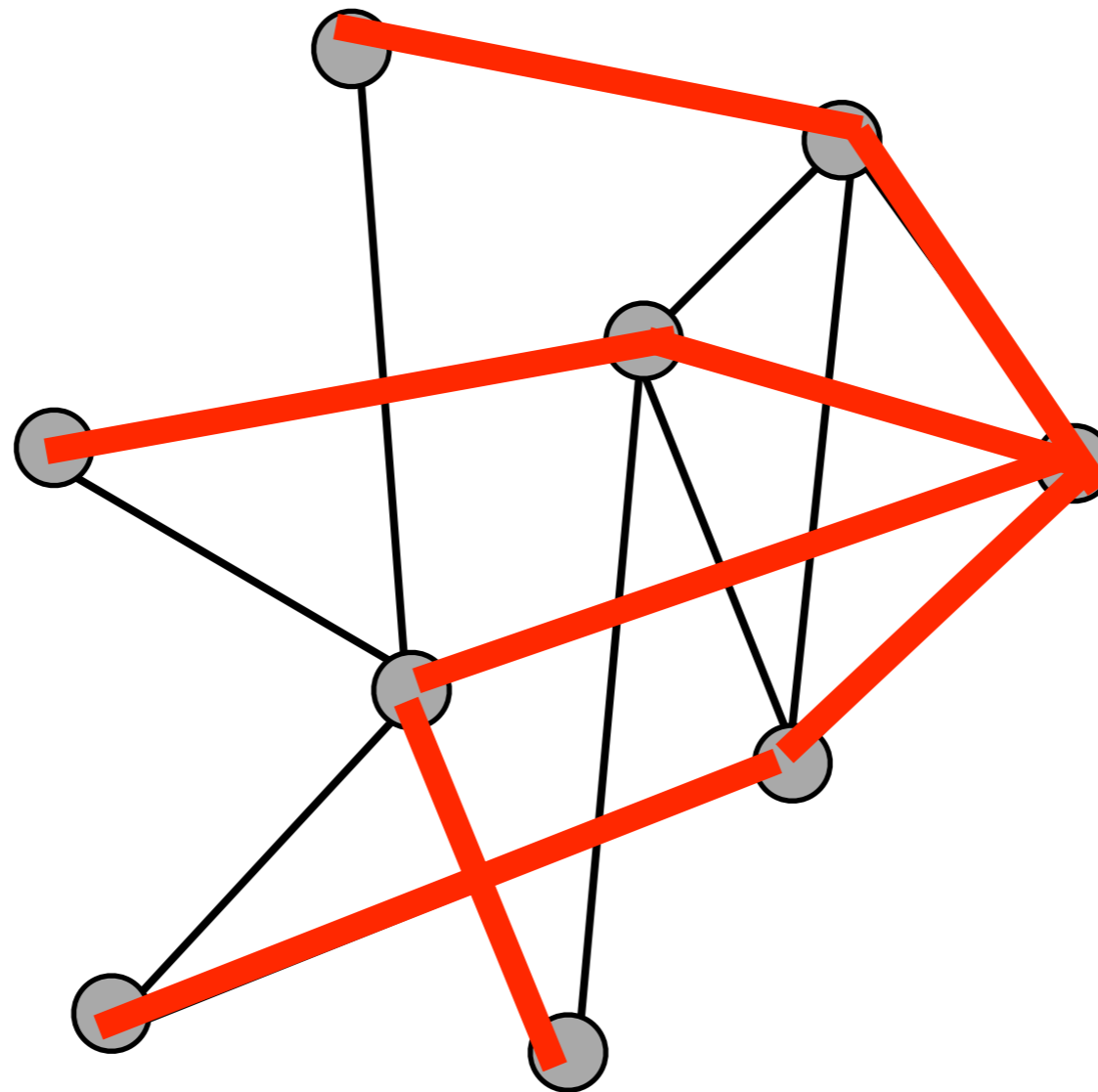
Example – Spanning Trees



Example – Spanning Tree



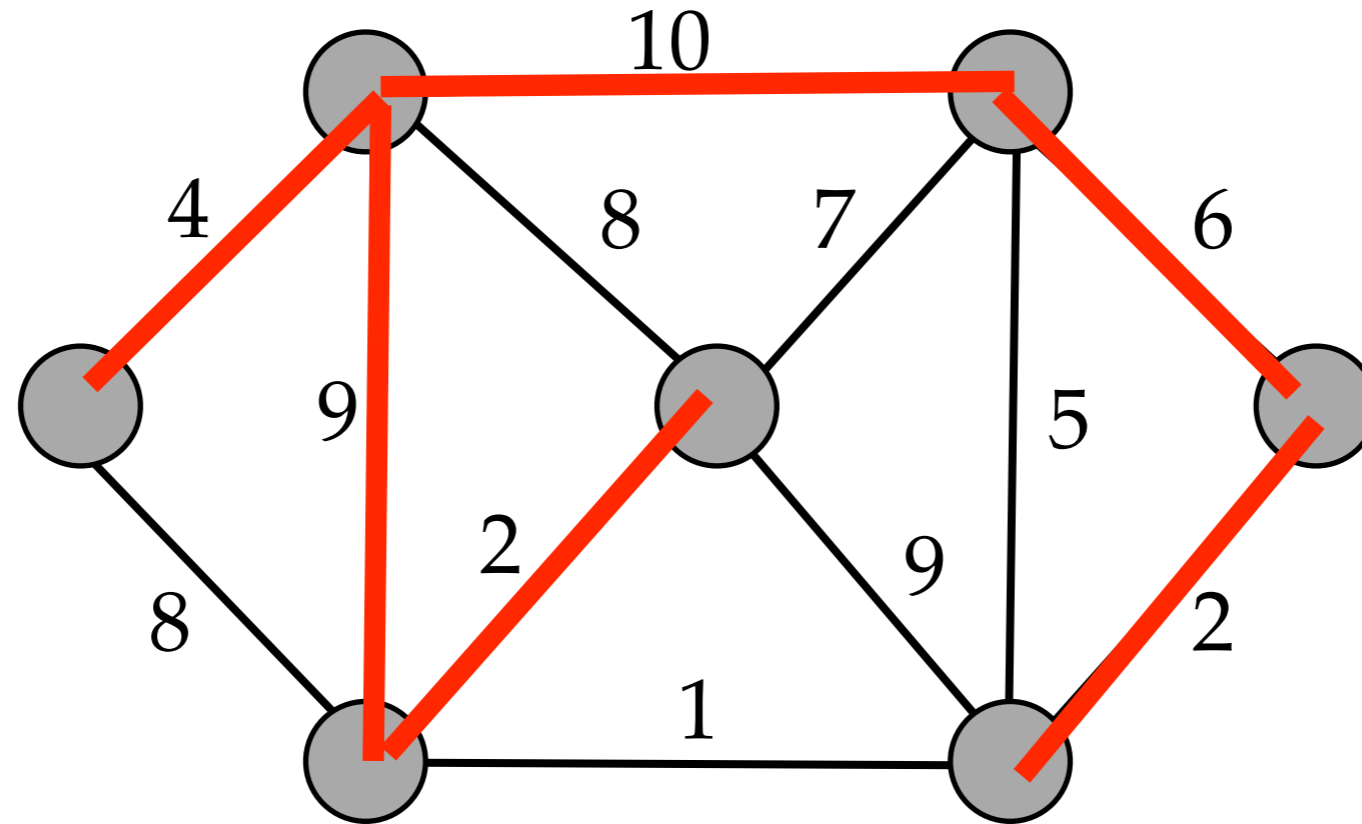
Example – Spanning Tree



Example Uses

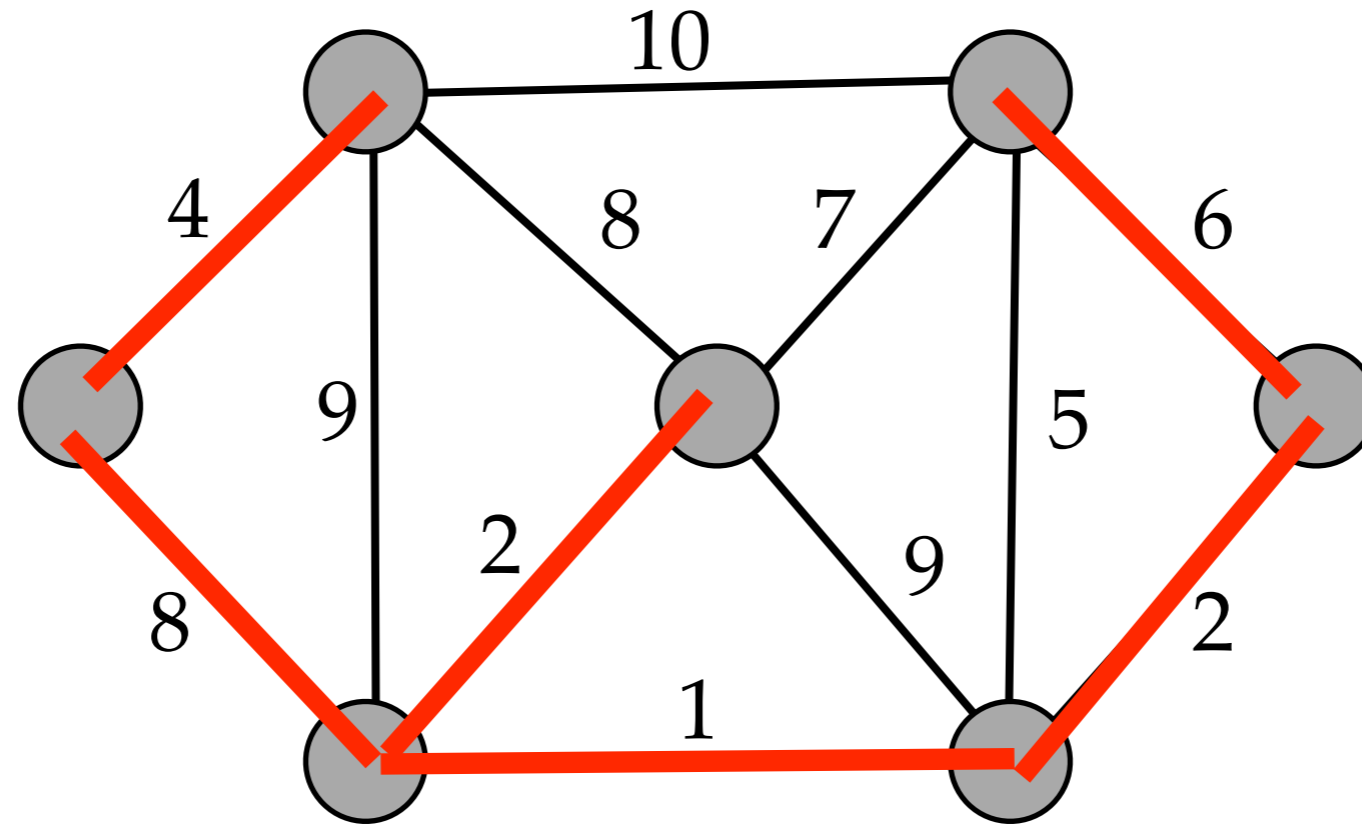
- **Cable routing:** connect houses with telephone or television cable.
- **Circuit Wiring:** minimum amount of solder or heat.

Minimum Spanning Trees



Cost is 33

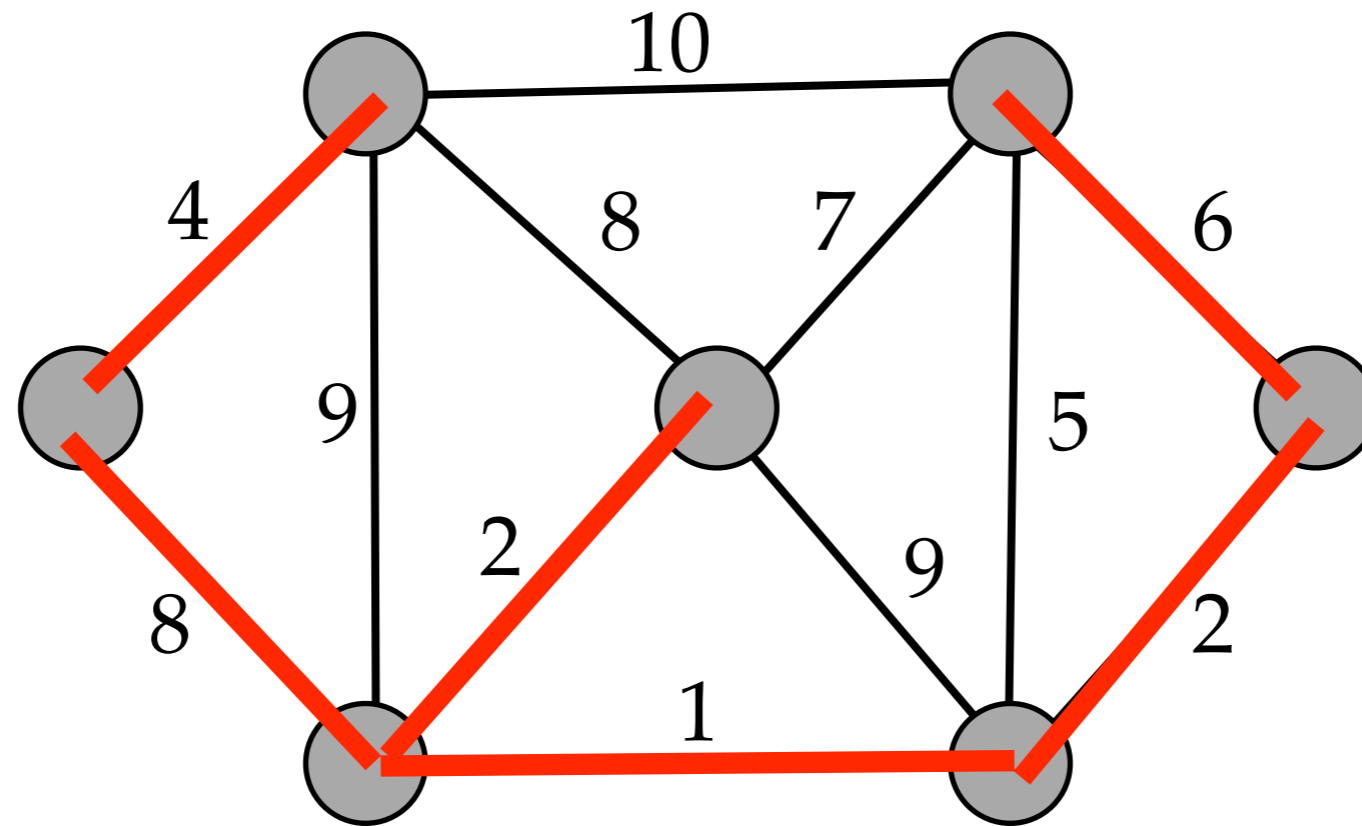
Minimum Spanning Trees



Cost is 22

Minimum Spanning Trees

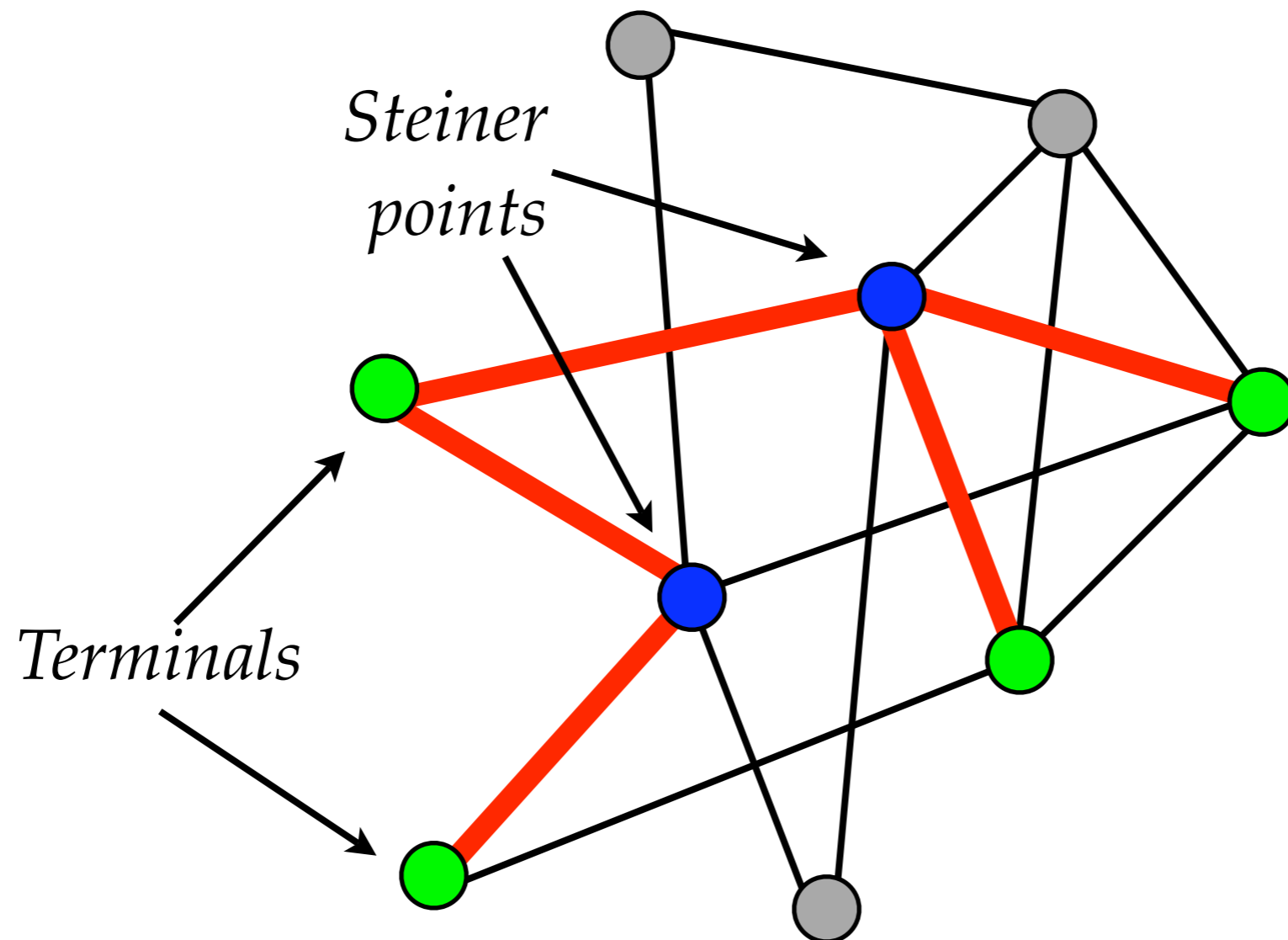
There may be several MSTs



Cost is 22

Steiner Trees

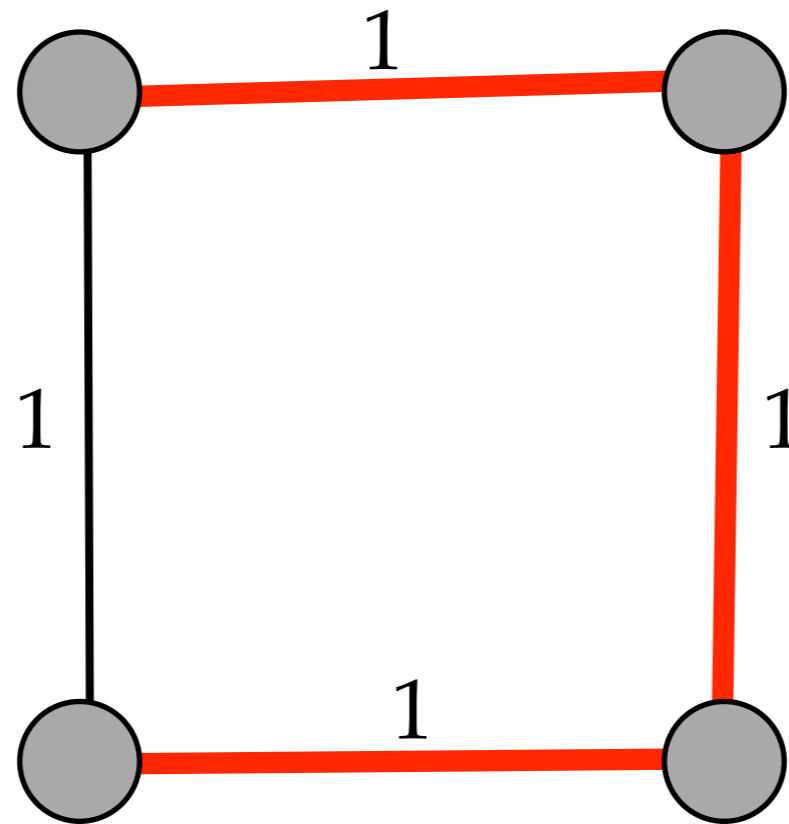
- A Steiner tree is a tree that connects a given subset of vertices (called terminal nodes)



Example – Euclidian MST

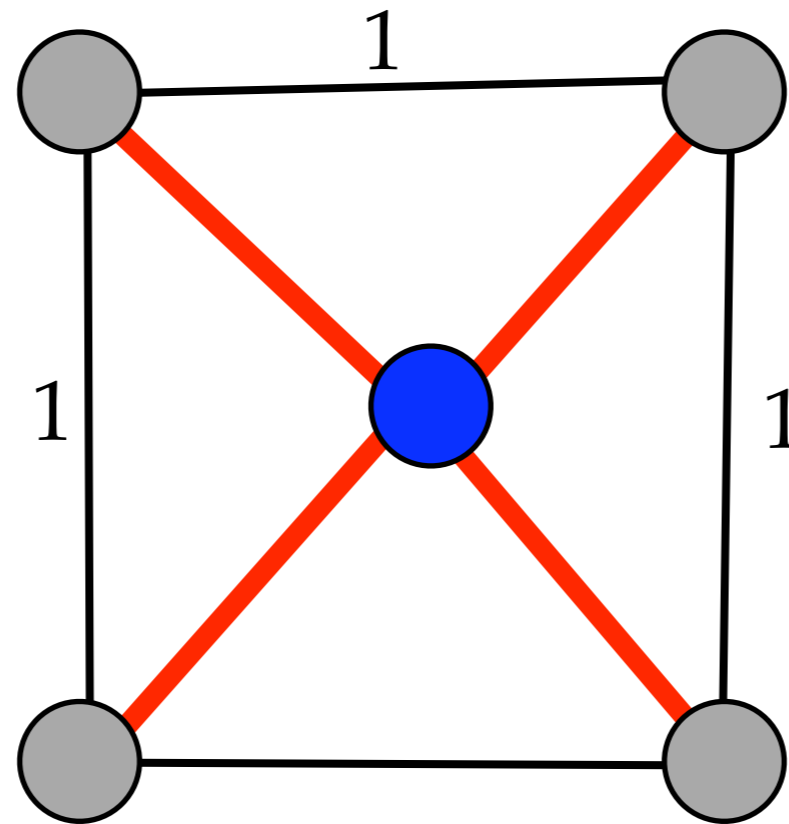
Often vertices are embedded in plane and distance is Euclidian distance

Example of *metric MST*



Cost of MST = 3

Example Metric Steiner Tree



$$\text{Cost} = 2\sqrt{2} = 2.83$$

Example of *metric Steiner Tree*

If allowed to add points, then cost can be reduced.

Which is computationally harder?

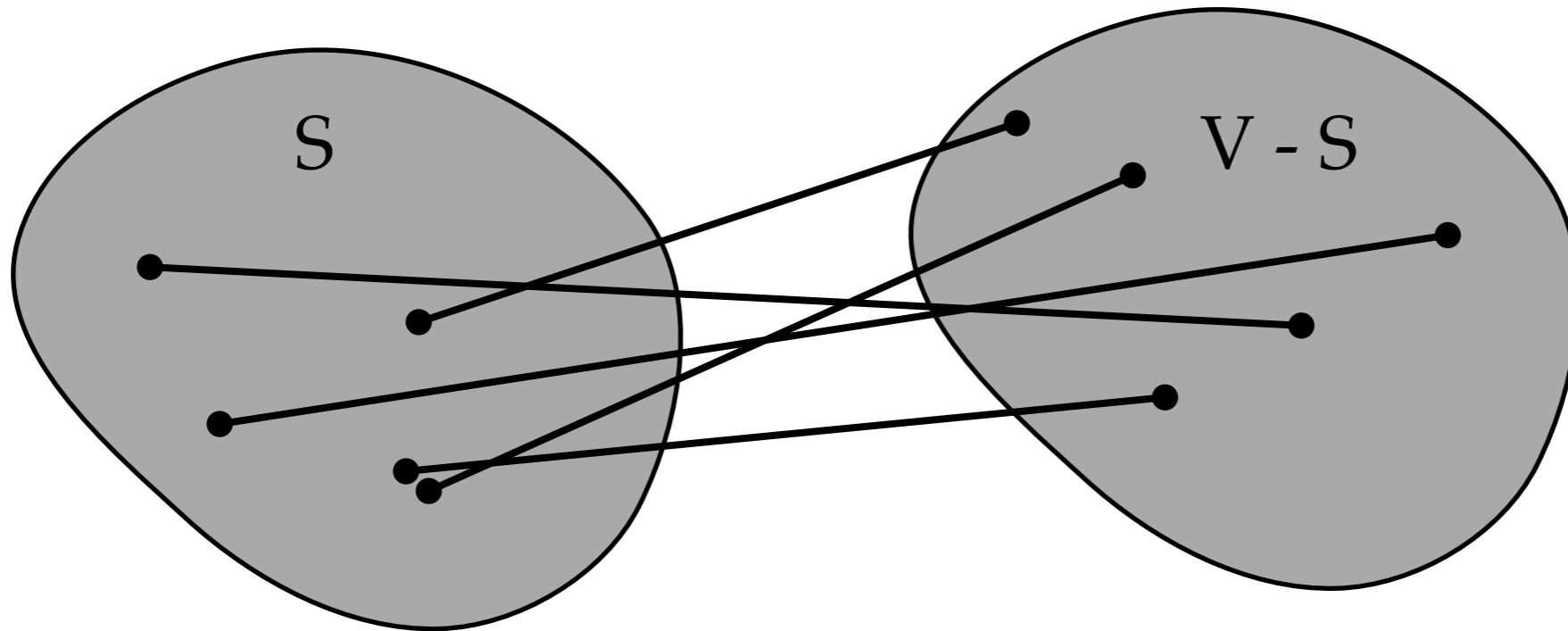
- $\text{MST}(G) = \text{Steiner}(G, V)$
 - Steiner tree where every vertex is a terminal is the same as MST.
- For which do you think there are faster algorithms?
 - A. MST?
 - B. Steiner?
 - C. Trick question?
 - D. Hard question?

MST is fast!

- MST is fast --- best is nearly linear.
- But no known fast algorithms for Steiner tree (NP-hard)
- How do we find the minimum spanning tree?

Cuts

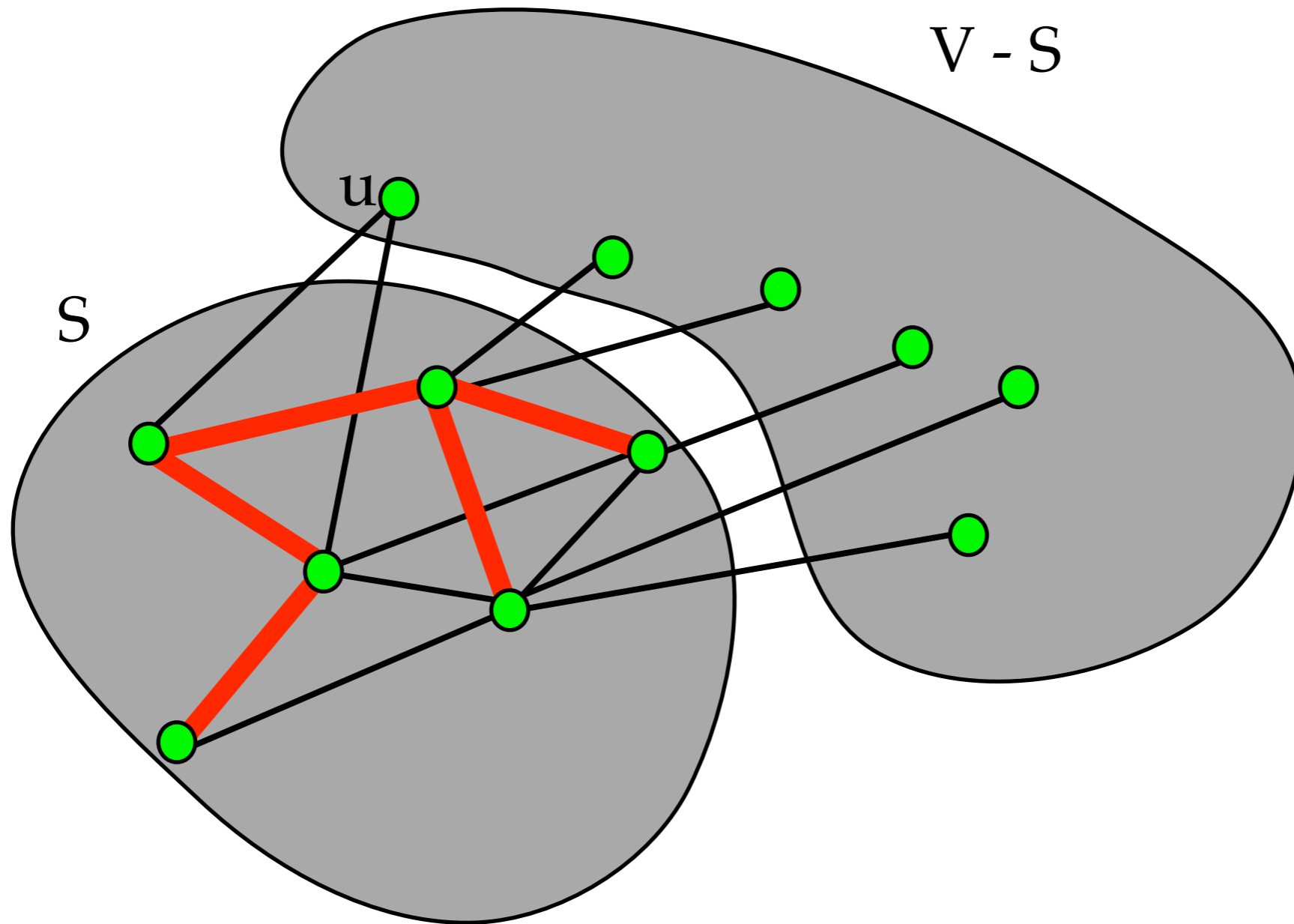
- A cut of a graph is a partition of its vertices into two groups.
- Sometimes we refer to the edges that go from one group to another as the cut.
- A edge $\{u,v\}$ with u in S and v in $V-S$ crosses the cut.



Viability Sets & Safe Edges

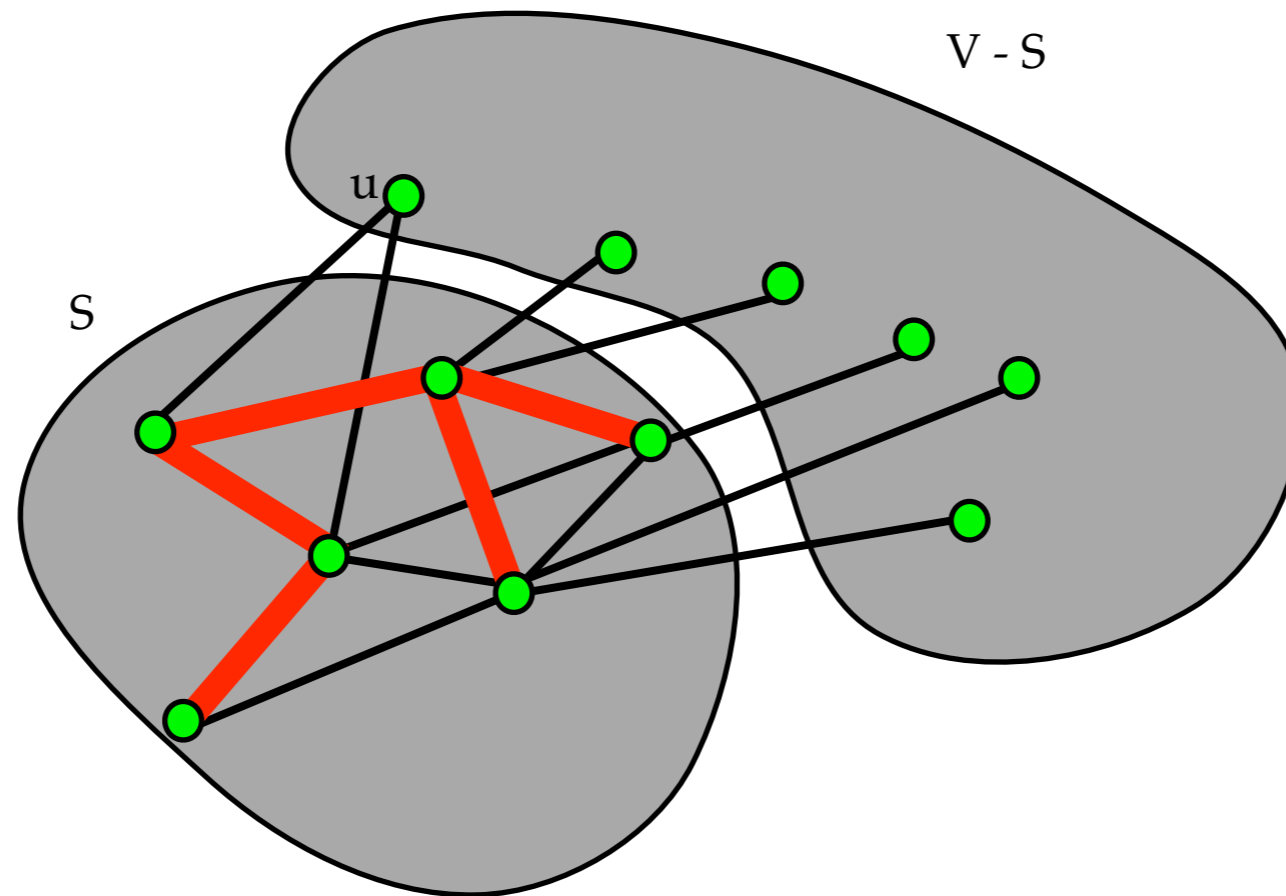
- An set of edges is *viable* if it can be extended to form a MST.
- An edge is safe if adding it to a viable set creates another viable set.
- If we can start with an empty edge set A (which is viable) and successively add $n-1$ safe edges, we'll end up with a MST.
- Which edges are safe and how do we find them quickly?

Respecting Cuts



A cut respects a set of edges A if no edge in A crosses the cut.

Respecting Cuts



If

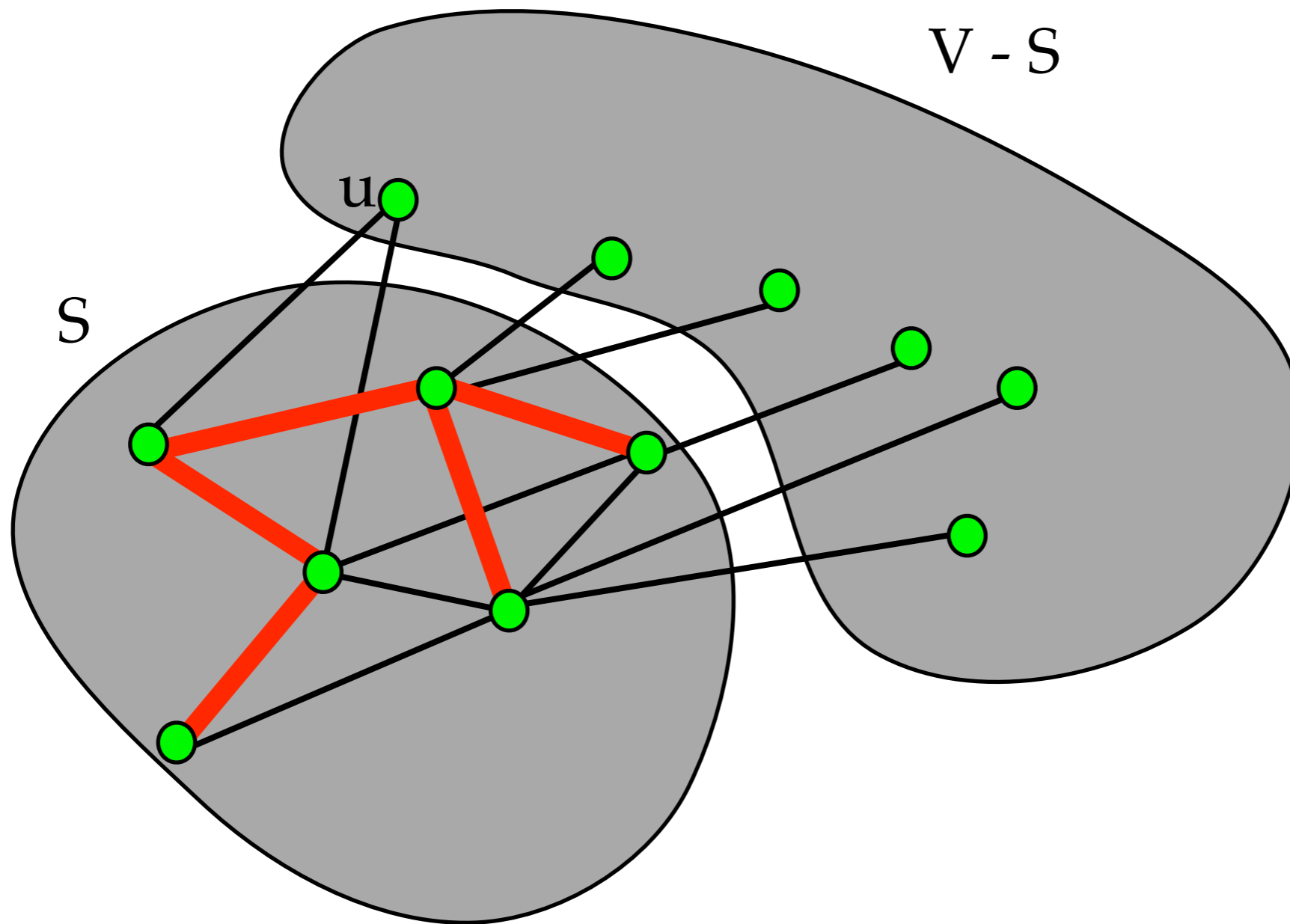
T is the set of tree edges added so far, and
 $(S, V-S)$ is an T -respecting cut

then

edges that cross $(S, V-S)$ can be added to the tree
without creating a cycle.

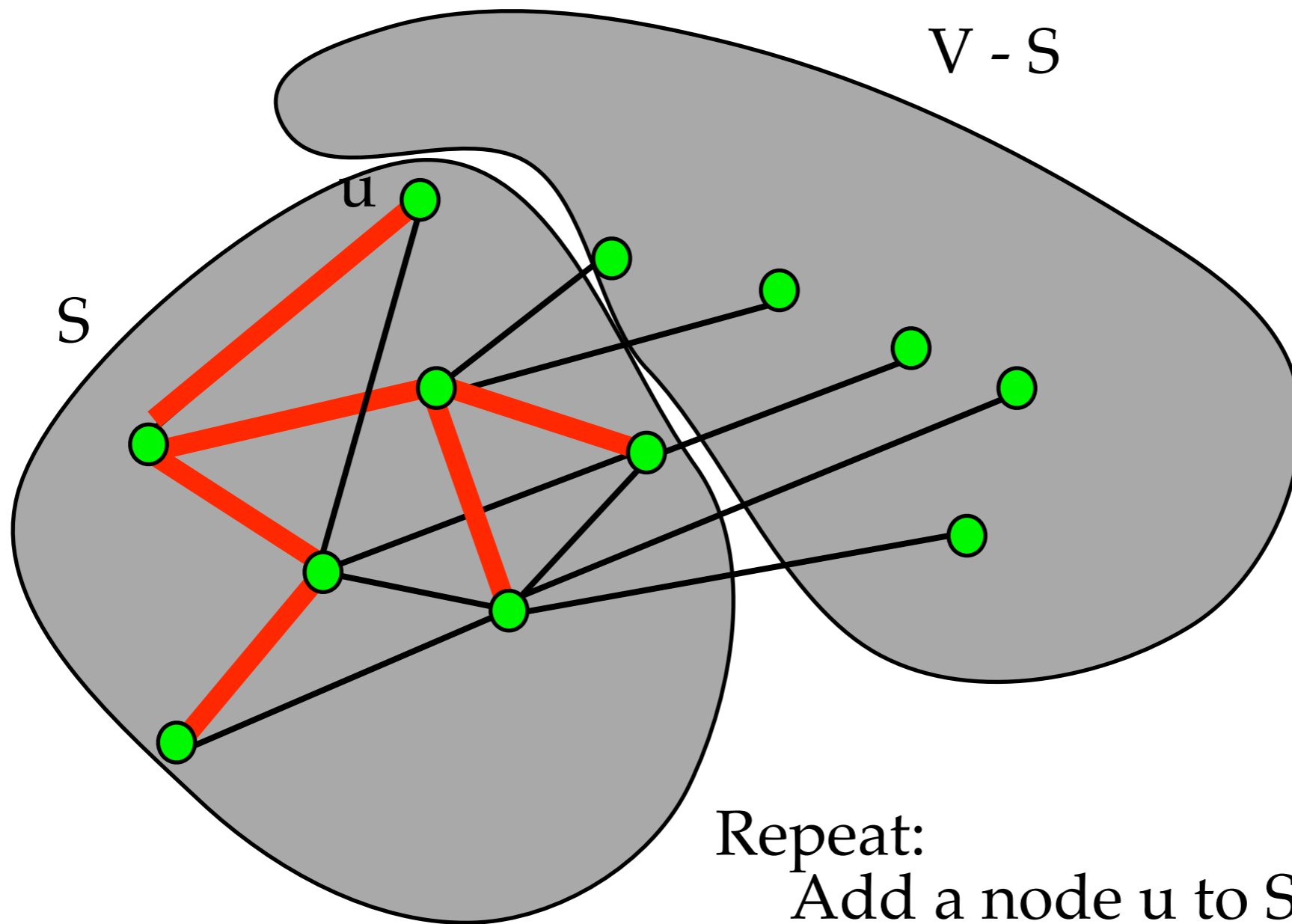
So: these are candidates to
add to the growing MST

Prim's Algorithm: Idea



S = the set of nodes already in the tree;
 S defines a cut of the graph.

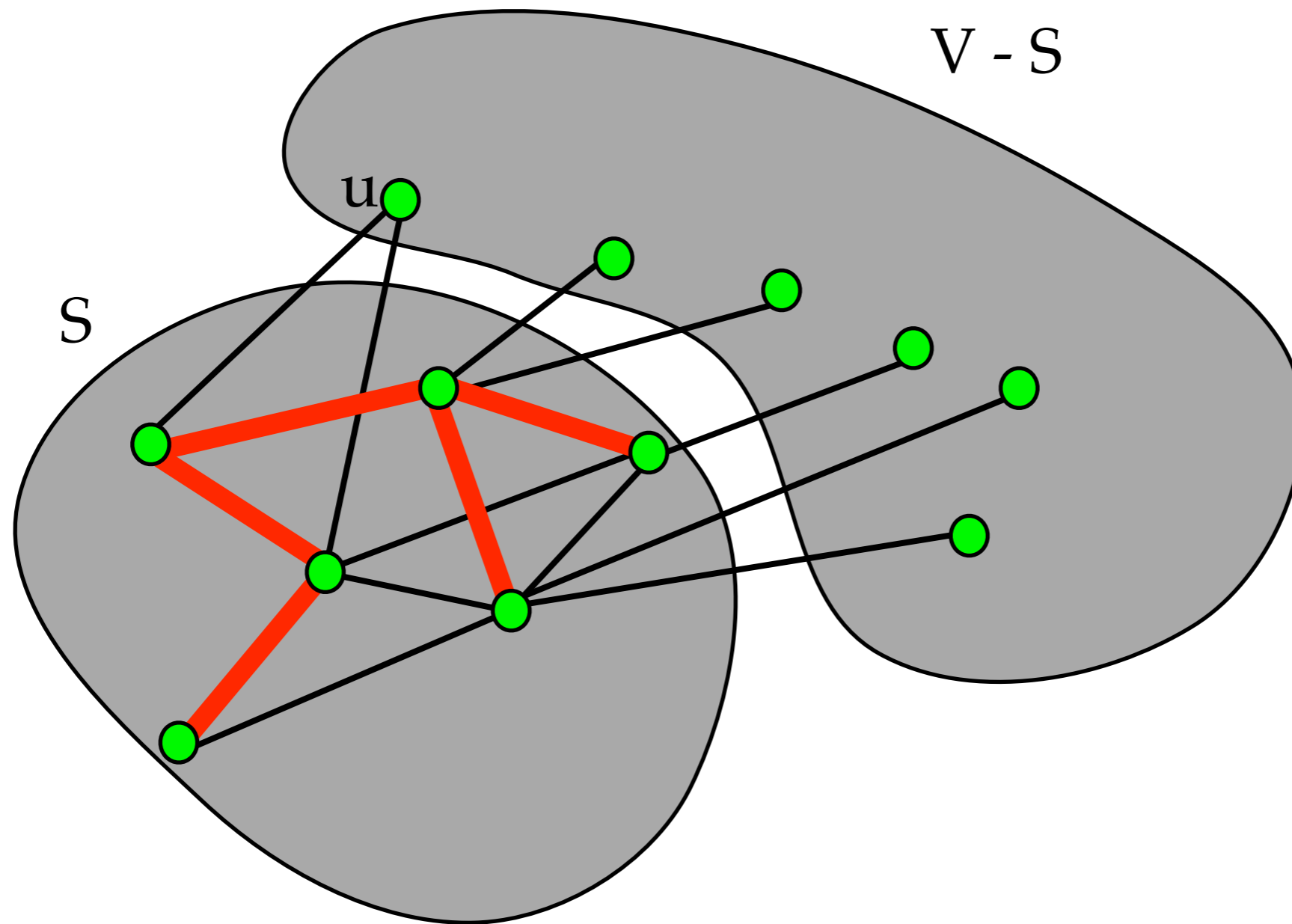
Prim's Algorithm: Idea



Repeat:
Add a node u to S from $V - S$.
 $S := S + \{u\}$

How do you choose u ?

Prim's Algorithm: Idea



Greedy idea: Choose the edge with the smallest weight that cross the T -respecting cut.

Greedy Algorithms

- Repeatedly selected a “locally optimal” choice.
- Never undo a choice you’ve already made.

*How do you know this will result in
the minimum cost spanning tree?*

Free Tree Facts

- Tree of n nodes has $n - 1$ edges.
- There is a unique path between any pair of nodes
 - if there were two paths, then there would be a cycle
 - if there were 0 paths, wouldn't be connected
- Adding any additional edges creates a cycle!

MST Lemma

Thm. If T is a *viable* subset of E and $(S, V-S)$ is a T -respecting cut, then the minimum weight edge that crosses $(S, V-S)$ is safe.

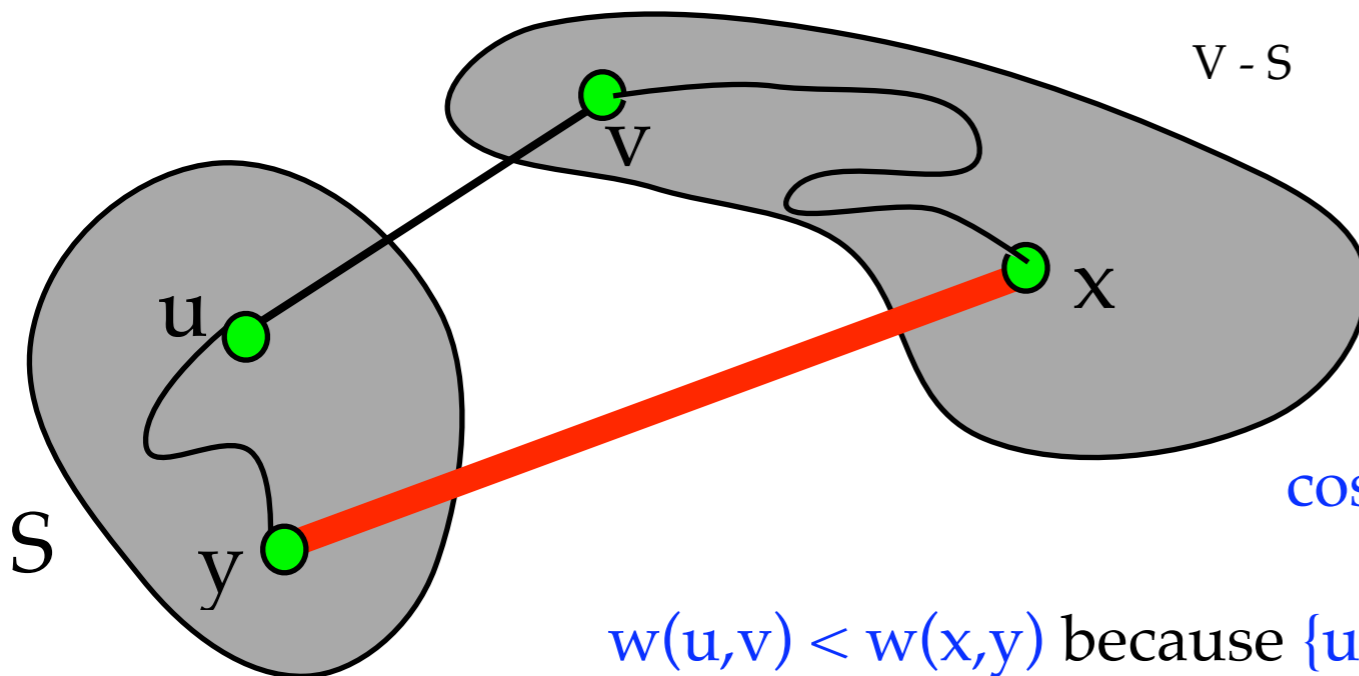
Proof. Suppose $\{u,v\}$ is the minimum weight edge and it is NOT safe.

Then: it is in NO MST.

So, let Q be a MST.

Add $\{u,v\}$ to $Q \Rightarrow Q'$, a subgraph with a cycle.

At least two edges of the cycle in Q' cross $(S, V-S)$ [why?]



(x,y) is not in T because the cut was T -respecting

So: remove (x,y) from Q' to get an NEW spanning tree Q''

$$\text{cost}(Q'') = \text{cost}(Q) - w(x,y) + w(u,v)$$

$w(u,v) < w(x,y)$ because $\{u,v\}$ was the minimum weight edge.

Therefore, Q'' is a lower-cost spanning tree than Q

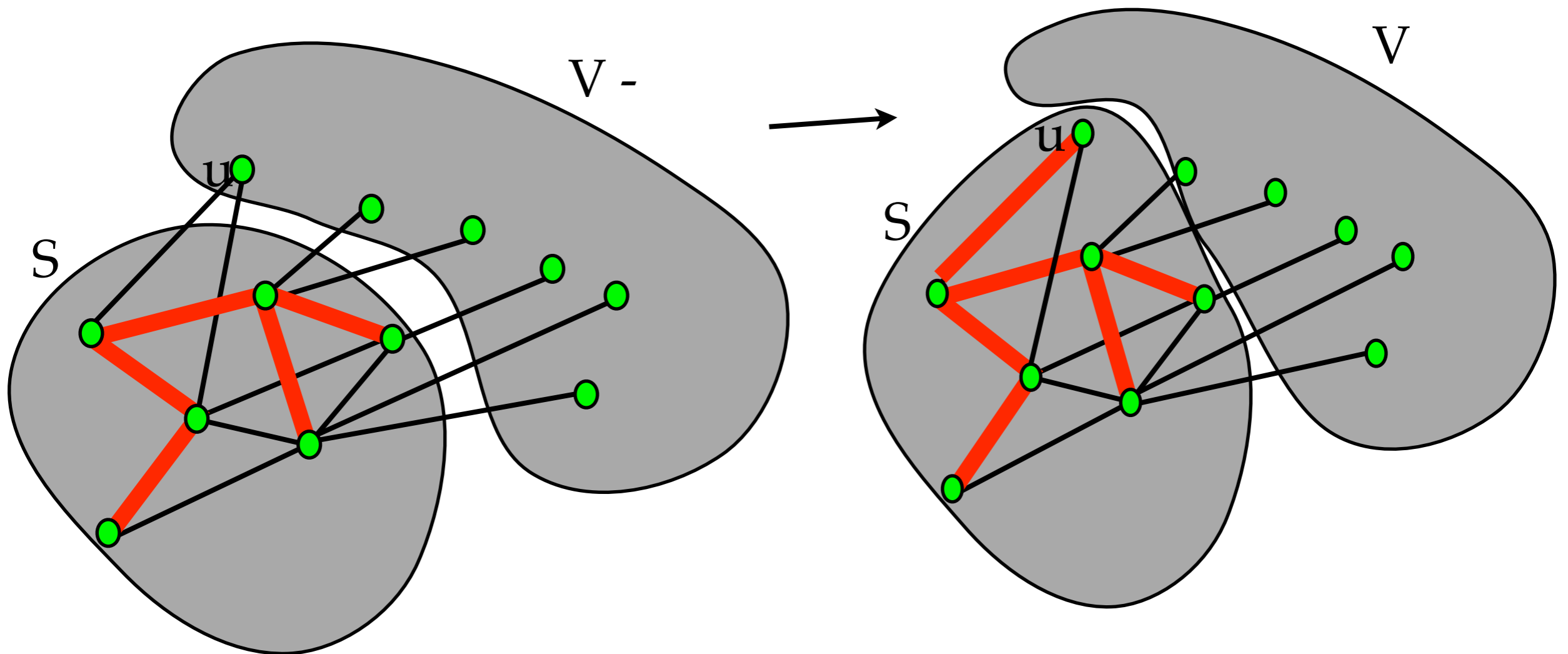
But Q was a MST. Contradiction!

Prim's Algorithm – The heap!

- At every step, we're adding the minimum weight edge that crosses the cut.
- Is there a way to find this minimum edge quickly?
- A heap!
- What do we store in the heap?

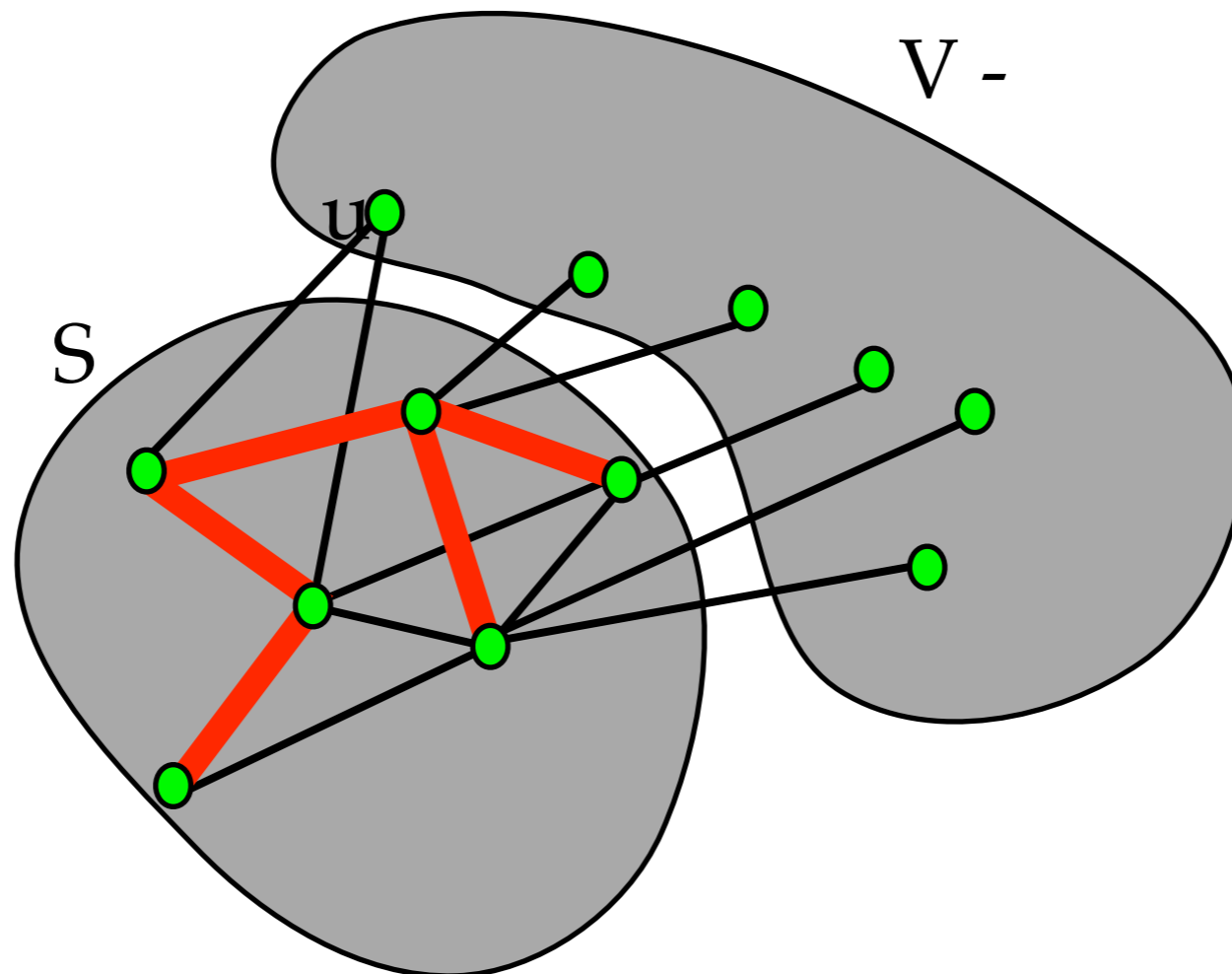
Storing edges is problematic

- If we put edges into the heap, we have to do complicated updates when we add a node to S :



Better idea: Store **NODES** in heap

- Heap stores nodes that are not yet in the spanning tree (aka nodes in $V-S$)
- $\text{key}(u) = \text{weight of lowest cost edge going from current } T \text{ to } u$.
- $\text{key}(u) = \text{INFINITY}$ if there is no edge from T to u .



```

def MST_Prim(G, root):
    # initialize
    for u in Vertices(G):
        u.key = infinity
        u.intree = False
        u.parent = None

    # root has the smallest key (so extracted first)
    root.key = 0;

    # create a heap of the vertices not in the tree
    Q = Heap(Vertices(G))

    # while there are vertices not in the tree
    while not Empty(Q):

        # find vertex that can be connected to S most cheaply
        u = Q.deletemin()

        # update its neighbors that are not in the tree
        for v in Neighbors(u):
            if not v.intree and w(u,v) < v.key:
                Q.decrease_key(v, v.key - w(u,v))
                v.key = w(u,v)
                v.parent = u
        u.intree = True

```

Prim's Running Time

- Each time you add a vertex to the tree, you spend:
 - $O(\log n)$ extracting the minimum vertex u
 - $\text{DEGREE}(u) * O(\log n)$ updating its neighbors

- Have to add all vertices to the graph eventually:

$$\begin{aligned} & \sum_u \log n + \text{DEGREE}(u) * \log n \\ = & \sum_u (1 + \text{DEGREE}(u)) * \log n \\ = & (\log n) \sum_u (1 + \text{DEGREE}(u)) \\ = & (\log n) (n + 2E) \quad \leftarrow E = \# \text{ of edges} \\ = & O(E \log n) \quad \leftarrow \begin{array}{l} \text{(sum of degrees counts} \\ \text{every edge twice)} \end{array} \\ & \quad \uparrow \\ & E > n \text{ because graph is} \\ & \text{connected} \end{aligned}$$