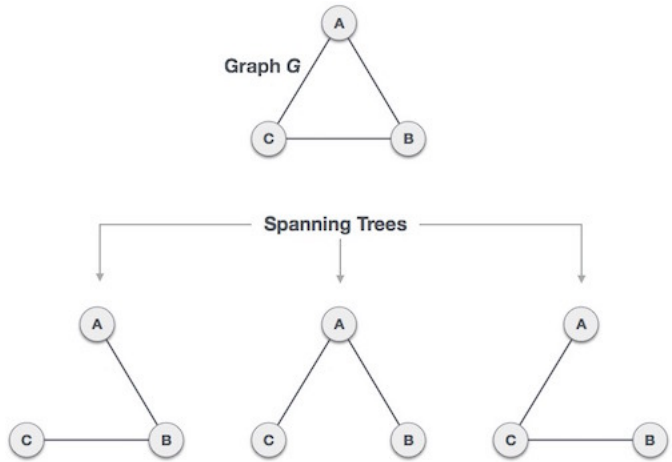
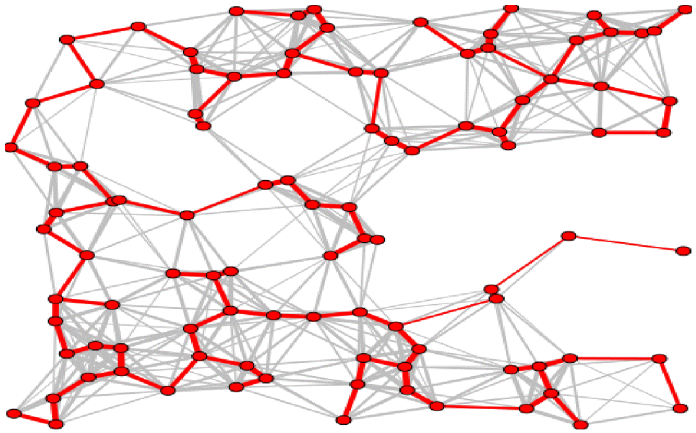
Algorithms

Spanning Tree

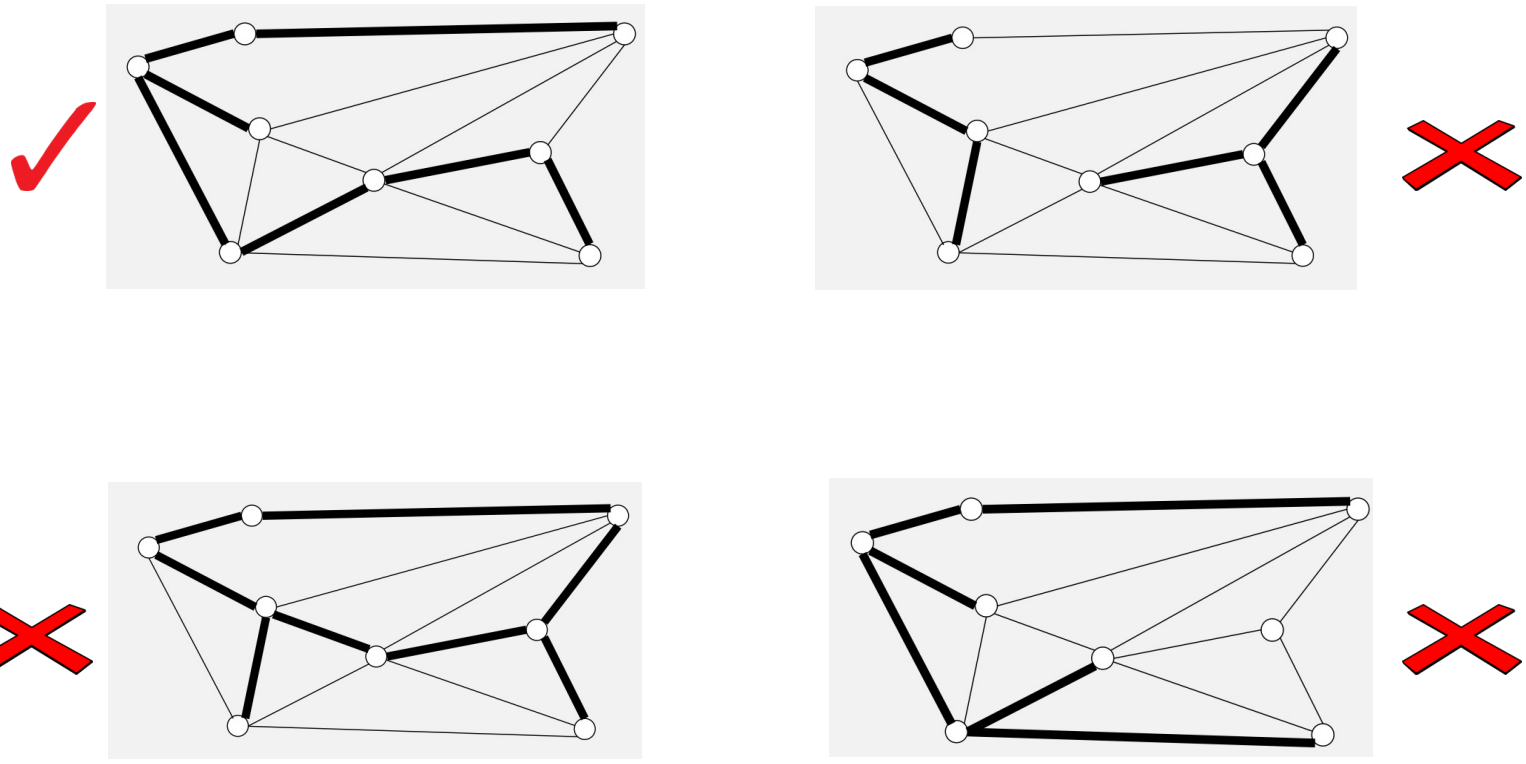
Given a connected graph G, a **spanning tree** of G is a subgraph of G which includes all vertices and forms a tree.





Spanning Tree

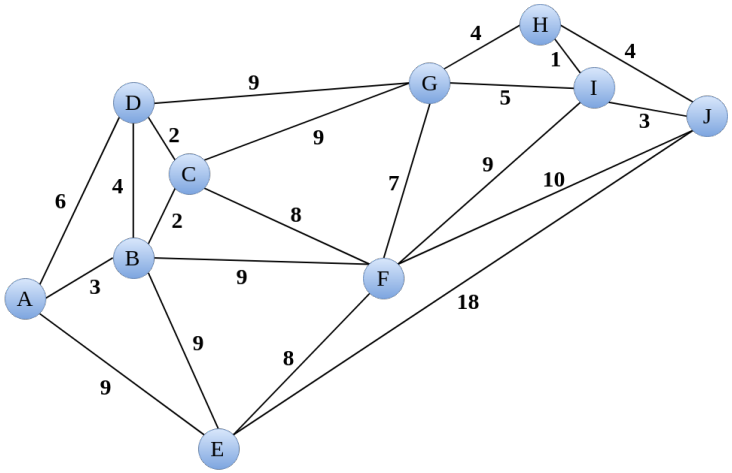
Which of the following show a spanning tree?



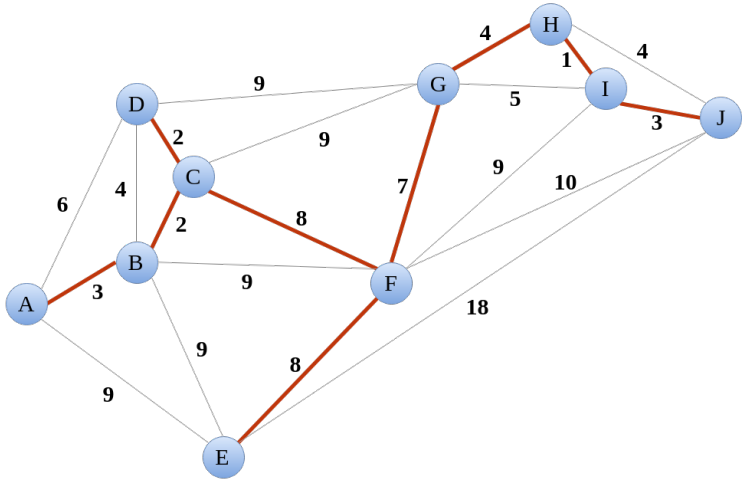
Minimum Spanning Tree

Let G be a connected undirected graph with +**ve** weights on the edges.

A **minimum spanning tree** of G is a spanning tree of G that minimizes the total weight of the edges in the tree.



G



MST of G

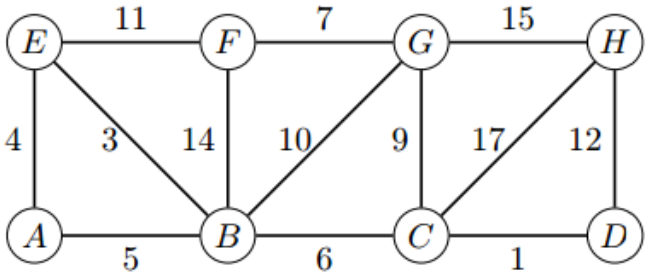
Many applications including network design, cluster analysis, image processing, error correction etc.

See https://en.wikipedia.org/wiki/Minimum_spanning_tree

Minimum Spanning Tree

What is the weight of the minimum spanning tree in this graph?

Exercise



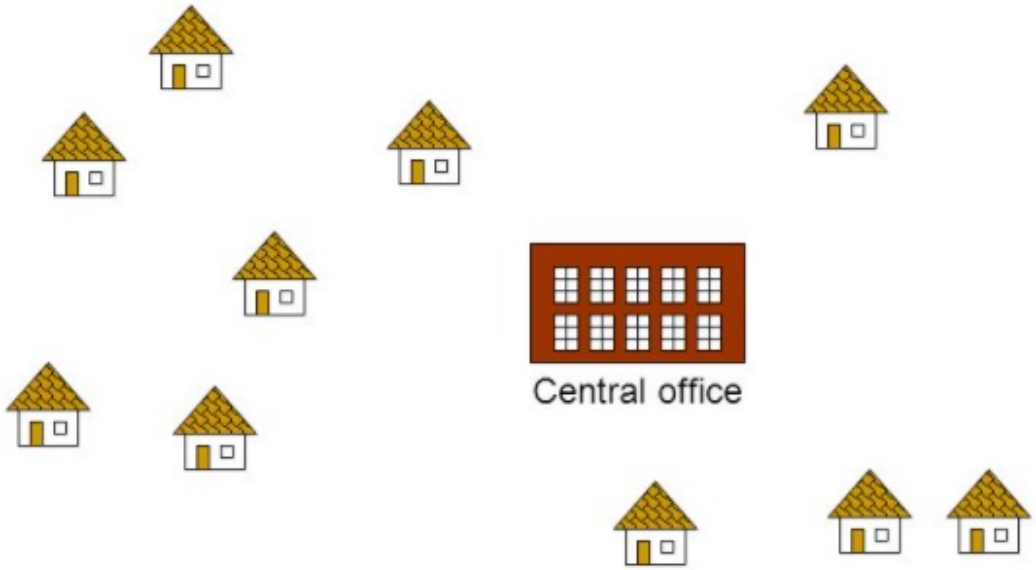
A. 41

B. 43

C. 45

D. none of the above

Minimum Spanning Tree



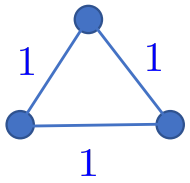
Suppose that we want a wired connection the central office to all office.

How do we minimize the total length of the wire used? *MST in the complete graph*

We can join any two buildings with a straight wired connection.

Minimum Spanning Tree (MST)

Is there always a unique MST? *No!*



Any two of the three edges form a spanning tree.

Fact. If all the edge weights are distinct, then there is a unique MST.

We can assume w.l.o.g. that the edge weights are distinct. *Why?*

We just need a consistent way of breaking ties among edges of equal weight.

This assumption is not necessary for the algorithms but makes proofs easier.

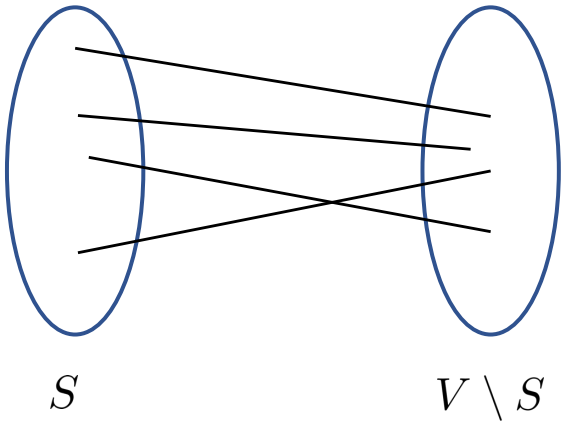
Minimum Spanning Tree (MST)

Connected undirected

Let $G = (V, E)$ be an input graph.

For any subset $S \subset V$, consider the two groups of vertices S and $V \setminus S$.

Such a partition of V is called a ***cut*** and denoted $(S, V \setminus S)$.



The edges going *across* the cut are called ***cut edges***.

Since the graph is connected, there are cut edges.

One of the cut edges should belong to the tree

Claim. The lightest cut edge belongs to the MST. ***Cut property.***

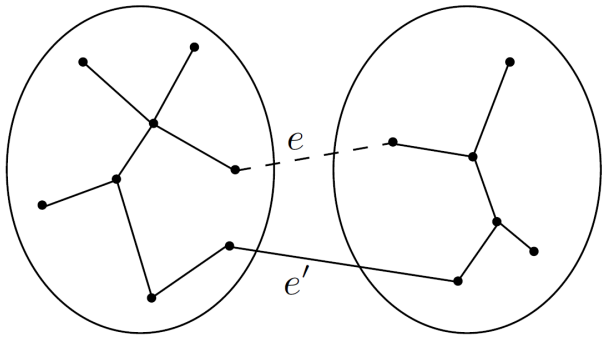
Minimum Spanning Tree (MST)

Claim. The lightest cut edge belongs to the MST. ***Cut property.***

Proof. Let e be the lightest edge.

Suppose that e is not present in the MST.

Then, we can add e to the tree and remove the heavier cut edge e' in the cycle that adding e creates.



This yields a spanning tree of weight smaller than the MST!

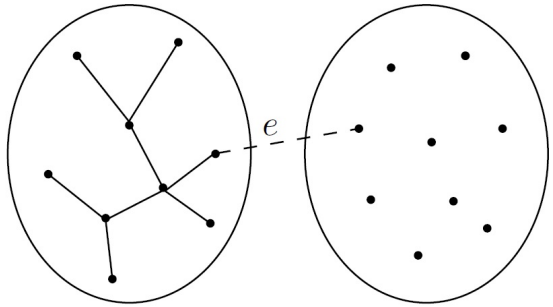
Contradiction! ■

Greedy Algorithms for MST

Prim's Algorithm.

Initial tree = any single vertex.

Repeatedly add to the tree the lightest edge connecting a tree vertex to a non-tree vertex until all vertices belong to the tree.

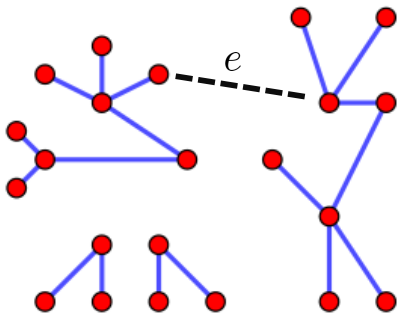


Kruskal's Algorithm.

At any time, we have a *forest* i.e. a collection of trees.

Initially, each vertex is a tree by itself.

We repeatedly add the lightest edge that joins two of the trees until we have a single tree.



The correctness of both algorithms follows from the cut property.

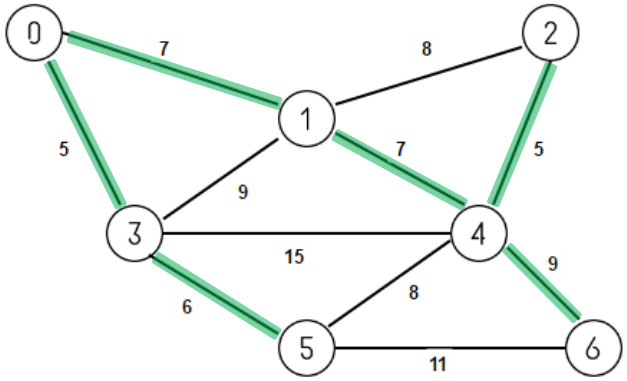
Greedy Algorithms for MST

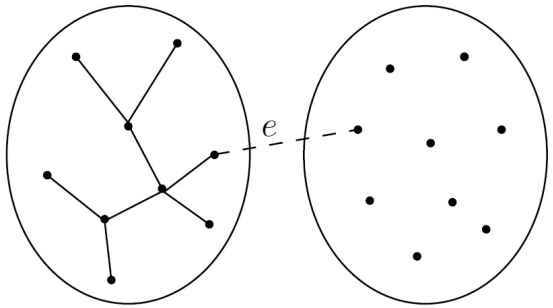
Prim's Algorithm.

Initial tree = any single vertex.

Repeatedly add to the tree the lightest edge connecting a tree vertex to a non-tree vertex until all vertices belong to the tree.

Example. *Start at* 1





Correctness follows from the cut property.

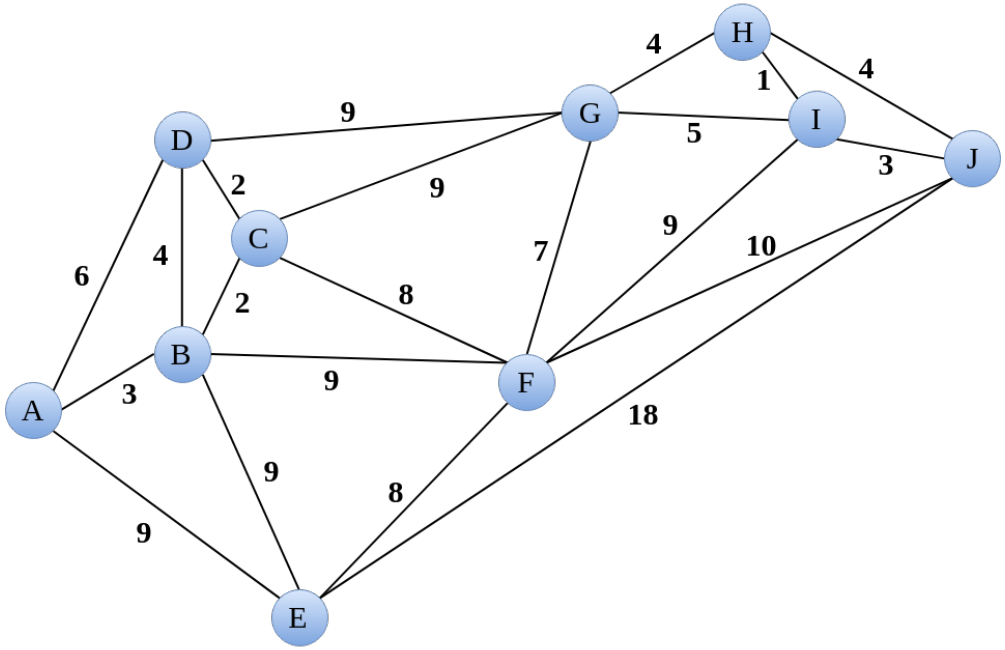
At any point in time, the tree vertices and the non-tree vertices form a cut.

We always pick the lightest cut edge.

Prim's Algorithm

Exercise.

Execute Prim's algorithm on the graph below with $\{C\}$ as the initial tree.



Prim's Algorithm

At each step of the algorithm, we need to find the lightest edge connecting a tree vertex to a non-tree vertex.

Implementation using Priority Queues:

Whenever we add a vertex v to the tree, we insert each edge $e = (v, w)$ joining v to a non-tree vertex w to a priority queue with the weight of e as priority. Log m

ExtractMin always gives us the lightest edge. Log m

We add this edge to our MST ***iff*** *it is a tree to non-tree edge.*
We discard edges connecting two tree vertices.

Log m

Running time of the algorithm on a graph with n vertices and m edges?

- m insertions into the priority queue
- $\leq m$ ExtractMin operations
- Deleting edges between tree vertices <= m times

- additional $O(1)$ work for each edge and vertex

Time:

$$O(n + m \log m)$$

$$= O(n + m \log n)$$

since $m = O(n^2)$

Greedy Algorithms for MST

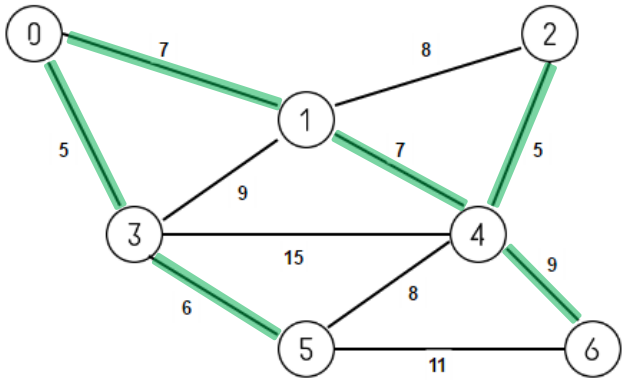
Kruskal's Algorithm.

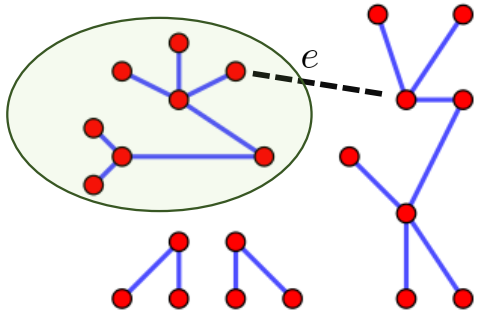
At any time, we have a *forest*
i.e. a collection of trees.

Initially, each vertex is a tree by itself.

We repeatedly add the lightest edge that joins two of the trees until we have a single tree.

Example.





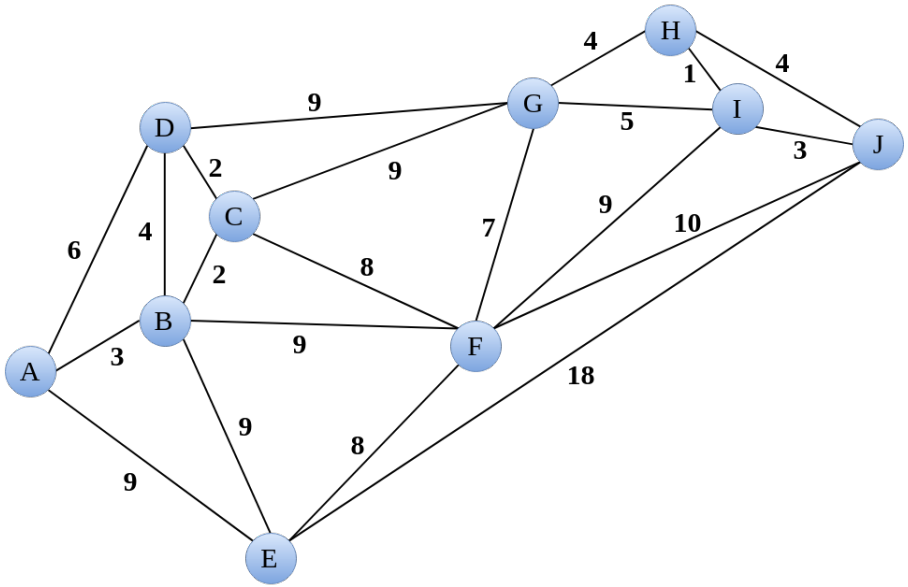
Correctness follows from the cut property.

Let T be the set of vertices in the tree containing one of the end-points of u.

Then e is the lightest edge in the cut $(T, V \setminus T)$.

Kruskal's Algorithm

Exercise. Execute Kruskal's algorithm on the graph below.



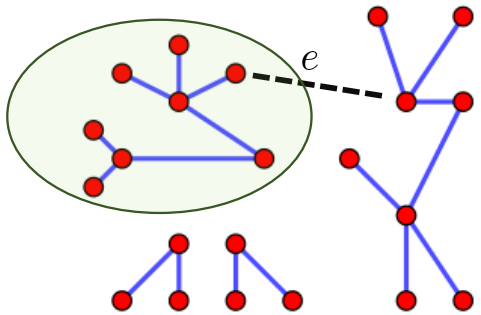
Kruskal's Algorithm

Kruskal's Algorithm.

At any time, we have a *forest* i.e. a collection of trees.

Initially, each vertex is a tree by itself.

We repeatedly add the lightest edge that joins two of the trees until we have a single tree.



Implementation?

Pseudocode.

```
Sort the edges in non-decreasing order of their weights.
Let e_1, ··· , e_m be the sorted order of edges.
for i = 1 to m:
    if the end-points of e_i are in different trees:
        add e_i to the MST
```

How do we implement this?