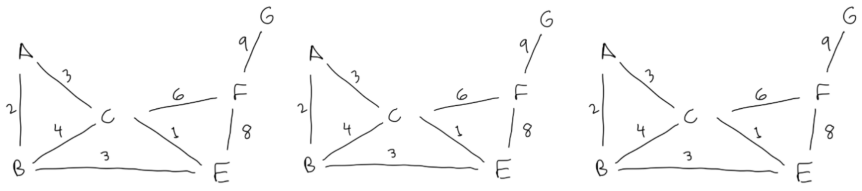
1: Minimum Spanning Trees

- Tree:
- Minimum Spanning Tree (MST):
- Cut:
- Crossing Edge:
- MST cut property:

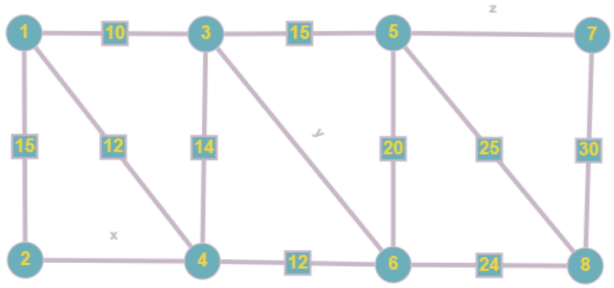
1. Find three MSTs corresponding to the weighted graph below using Prim's, Kruskal's, and the Reverse-Delete algorithm:



2. When encountering edges of the same weight, does it matter which one you add to the MST (assuming neither will create a cycle)? Why or why not?
3. Show the following statement is false: The largest edge in any cut of a graph G cannot be in the MST of G.
4. True or False: The cheapest edge of a cycle in a graph G must be in the MST of G.

2: Cut and Cycle Property

1. Design an algorithm that finds the Maximal Spanning Tree (the tree wherby the total edge weight connecting nodes is maximized) by modifying either Kruskal's or Prim's algorithm.
2. Let G be a graph of nodes and undirected weighted edges. Let's say we have already obtained the Minimum Spanning Tree for this graph G. However, one of the edges that is in G but not in this MST is randomly decreased by an unknown amount (eg: an edge of weight 5 not in the MST is now of weight 2). Design an algorithm that can find the new MST in this altered graph.
3. You are given the graph below with unknown weights on some edges. Let's say each of these unknown weight edges are known to be in the MST of the graph. What are the possible weight values for each of the unknown edges?



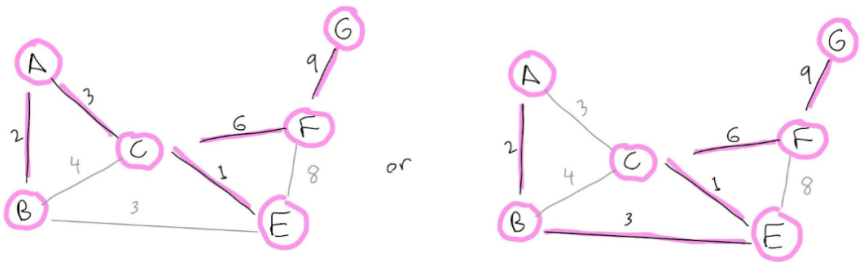
Helpful Videos:

- Prim's algorithm in 2 minutes, Micheal Sambol, https://www.youtube.com/watch?v=cplfcGZmX7I
- Kruskal's algorithm in 2 minutes, Micheal Sambol, https://www.youtube.com/watch?v=71UQH7Pr9kU

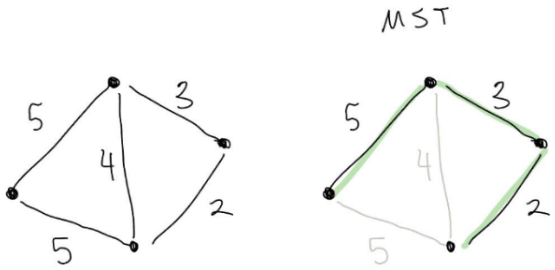
3: Solutions

Minimum Spanning Trees

- Tree: A connected acyclic graph. A connected graph with no cycles, meaning it contains exactly $|V| - 1$ edges.
- Minimum Spanning Tree (MST): given a connected undirected weighted graph G = (V, E, w), find a spanning tree of minimum weight, where the weight of a tree T is defined as the sum of the edge weights in the tree.
- Cut: splitting the graph's vertices into two non-empty sets, usually denoted as S and $V - S$
- Crossing Edge: an edge with one end in S and one end in $V - S$, an edge that crosses the cut.
- MST cut property: in any cut of the graph, the crossing edge with the smallest weight must be in the MST

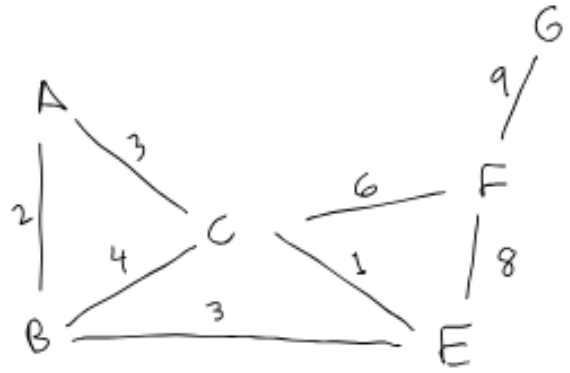


1. Order of edges used to arrive at MST may vary; final MSTs should look like the following. Walk through of the answer for each method is attached at the end of this document.
2. As long as the addition of an edge does not create a cycle, edges of equal weight are equally considered by Prim's and Kruskal's. Thus, it is arbitrary amongst equal-weight edges for which is first added to an MST.
3. A simple counterexample to this statement would be to have a node in a graph that is only connected to the rest of the graph by a single edge. This edge would have to be in the MST, but would technically be the largest edge across the cut separating this node from the rest of the graph.
4. False; if the cheapest edge of one cycle is the most expensive edge of another cycle, it cannot be in the MST. See counterexample:



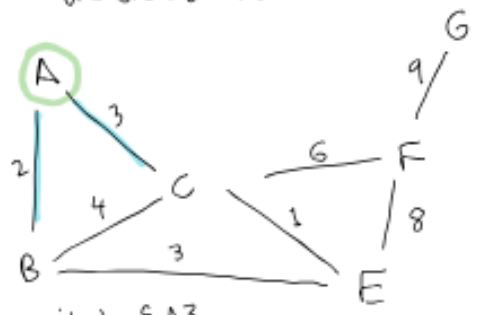
Cut and Cycle Property

1. The cut property is true for both the minimum edge spanning a cut and the maximum edge spanning a cut (ie: if the smallest edge in a cut is in the minimum spanning tree, the largest edge in a cut is in the maximum spanning tree). Thus, we could simply use Prim's algorithm which continually adds the smallest edge, and modify it to continually add the largest edge. The same proof by the cut property that ensures Prim's algorithm yields a minimum spanning tree would also ensure the modified algorithm yields a maximum spanning tree.
2. We will need to consider the impact of this edge on the MST; the easiest way to do so is to add the altered edge to the existing MST. According to the cycle property, adding an edge to an existing MST will create exactly one cycle. We can then detect the cycle within this MST, loop over every edge in the cycle, and remove the one with the maximum-weight to recreate the MST.
3. According to the cycle property, the edges *x*, *y*, and *z* must be lower than or equal to the maximum value in every cycle. Thus, the edge weights could be $x \leq 15, y \leq 12, z \leq 30$. (notice that you have to check multiple cycles in some cases)



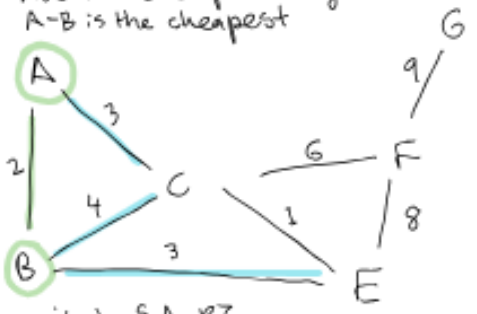
Prim's algorithm walkthrough

1. choose arbitrary node.
   we choose A



visited = {A}

2. look at all edges of visited nodes.
   Add the cheapest edge to the tree
   A-B is the cheapest

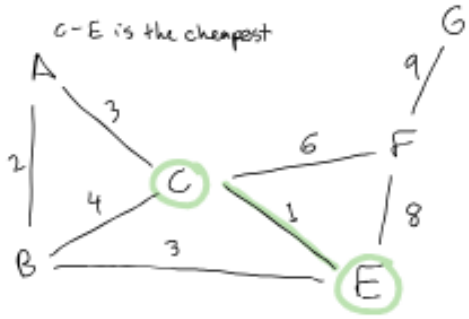


visited = {A, B}

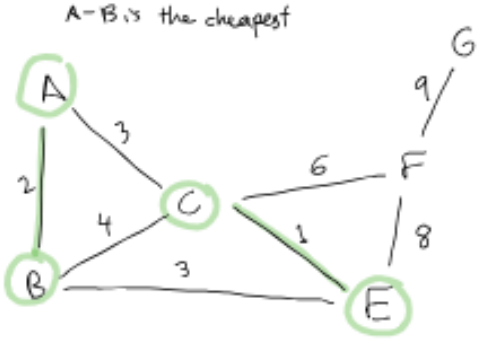
Kruskal's algorithm walkthrough

1. Find the smallest edge in the graph
   and add it

   C-E is the cheapest

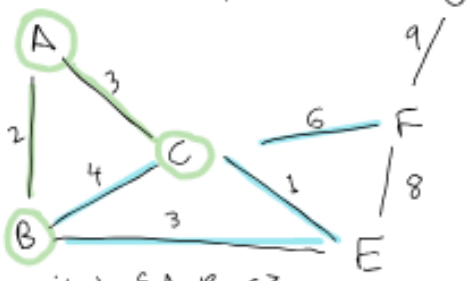


A-B is the cheapest



3. Repeat

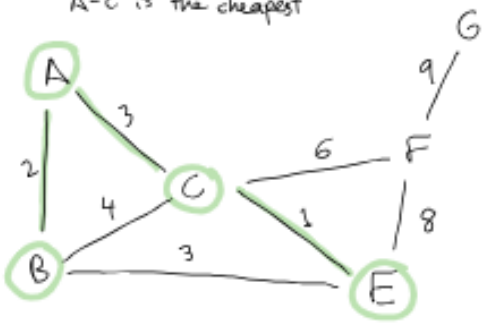
A-C is the cheapest



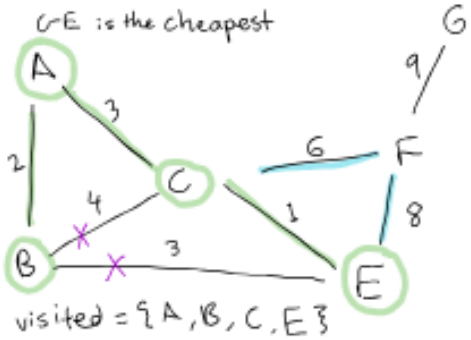
visited = {A, B, C}

(when weights are the same, just choose any)

A-C is the cheapest

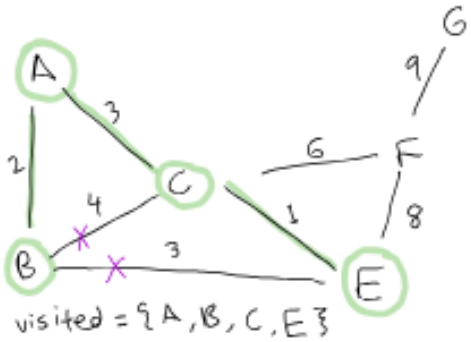


C-E is the cheapest



visited = {A, B, C, E}

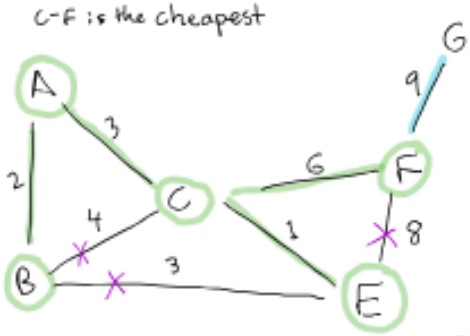
cannot add B-E in the next step, even though it is the lowest edge, because it will create a cycle A-C-E-B-A. We cannot add B-C, as it will create B-C-A-B



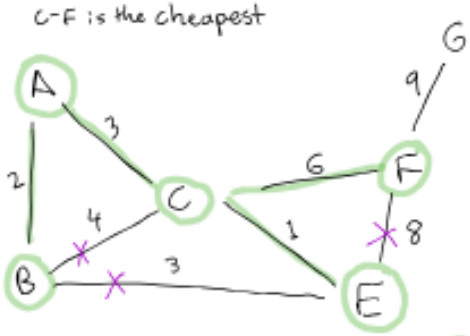
visited = {A, B, C, E}

cannot add B-E in the next step, even though it is the lowest edge, because it will create a cycle A-C-E-B-A. We cannot add B-C, as it will create B-C-A-B

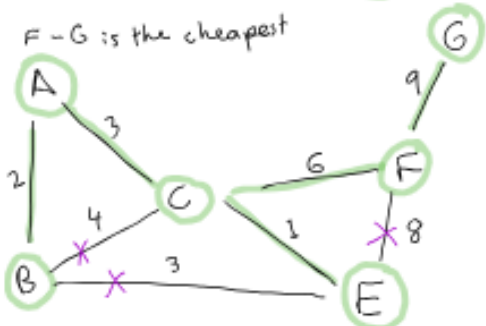
C-F is the cheapest



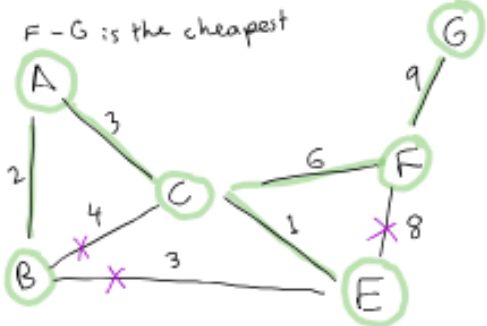
C-F is the cheapest



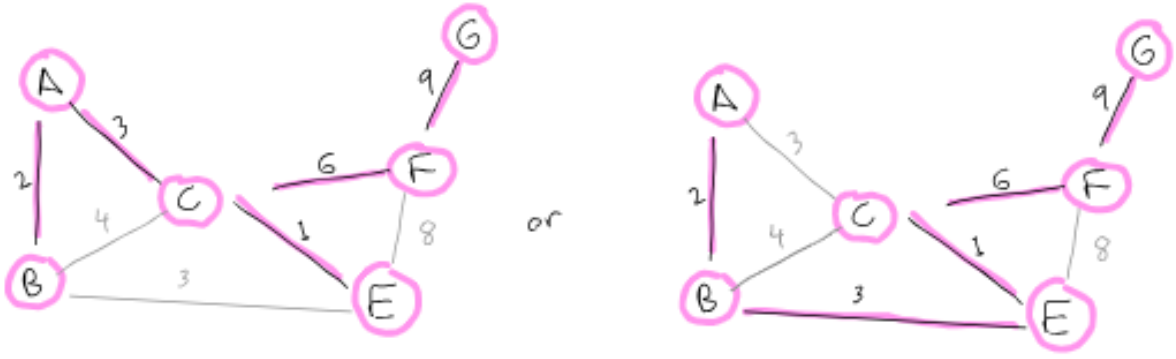
F-G is the cheapest



F-G is the cheapest



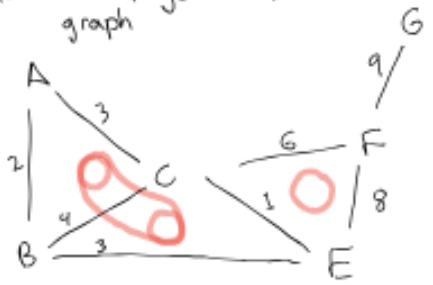
Answer:



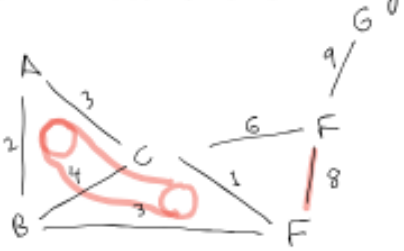
Reverse Delete

Delete heaviest edges until there are no more cycles, without disconnecting the graph

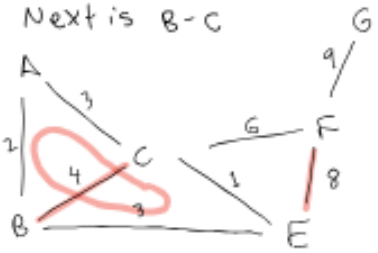
1) There are 4 cycles in this graph



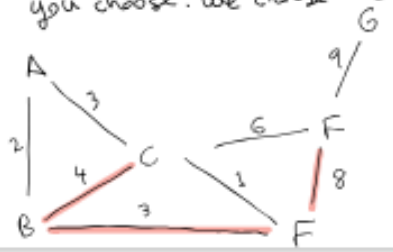
2) F-E is heaviest edge that can be deleted without disconnecting the graph



3) Next is B-C



4) Next is A-C or B-E, doesn't matter which one you choose. We choose B-E



Final graph: