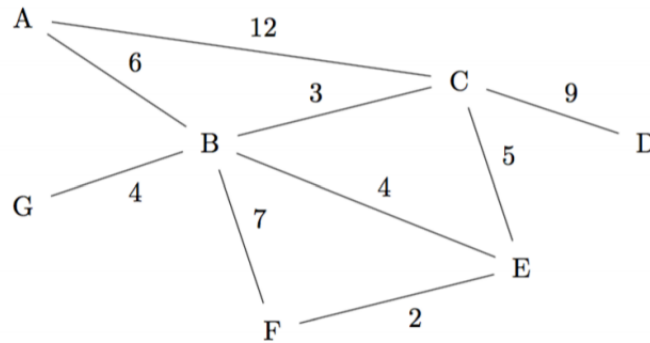


## 1 Warmup with MSTs



- (a) For the graph above, list the edges in the order they're added to the MST by Kruskal's and Prim's algorithm. Assume Prim's algorithm starts at vertex A. Assume ties are broken in alphabetical order. Denote each edge as a pair of vertices (e.g. AB is the edge from A to B)

Prim's algorithm order: **AB, BC, BE, EF, BG, CD**

Kruskal's algorithm order: **EF, BC, BE, BG, AB, CD**

- (b) Is there any vertex for which the shortest paths tree from that vertex is the same as your Prim MST?

**Vertex B, A, or G**

- (c) True/False: Adding 1 to the smallest edge of a graph G with unique edge weights must change the total weight of its MST

**True, either this smallest edge (now with weight +1) is included, or this smallest edge is not included and some larger edge takes its place. Either way, total weight increases**

- (d) True/False: The shortest path from vertex A to vertex B in a graph G is the same as the shortest path from A to B using only edges in T, where T is the MST of G.

**No, consider vertices C and E in the graph above**

- (e) True/False: Given any cut, the maximum-weight crossing edge is in the maximum spanning tree.

**True, We can use the cut-property proof as seen in class, but replace "smallest" with "largest"**

## 2 Oracle Dijkstra's

In some graph  $G$ , we are given a sorted list of nodes, sorted by their distances from some start vertex  $A$ . Design an algorithm to find the shortest paths tree starting from  $A$  in linear  $O(V+E)$  time. This algorithm essentially removes the purpose of the priority queue in normal Dijkstra's. When a node is removed from the PQ normally, this signifies we have found the shortest path from the source to that node, AND that this node is the next closest node to the source that hasn't been visited/visited yet. In a sorted list of nodes, we can simply traverse through the nodes in order. Therefore, our algorithm is simply to run Dijkstra's, but instead of keeping a priority queue we go through our sorted list of nodes in order. Our runtime is  $O(V+E)$

## 3 Graph Algorithm Design

For each of the following scenarios, write a brief description for an algorithm for finding the MST in an undirected, connected graph  $G$ .

- (a) If all edges have edge weight 1. Hint: Runtime is  $O(V+E)$  The key idea here is that any tree which connects all nodes is an MST. We can run DFS and take the DFS tree. You could also take a BFS tree, or run Prim's algorithm with a queue or stack instead of a priority queue (this would be equivalent to BFS/DFS). Unfortunately, a modified Kruskal's will be slightly slower, because even if we don't need to sort edges, the union-find operations will take additional time.
- (b) If all edges have edge weight 1 or 2. Hint: Use your algorithm from part (a) Remove weight 2 edges from the graph so only weight 1 edges remain. Now run an algorithm from part (a) as far as possible (e.g. find a DFS forest). We will have some number of connected components. Use these connected components as nodes in a new graph  $G^*$ . Look at the weight 2 edges in  $G$ . For each edge, if the nodes containing the two endpoints are not already connected in  $G^*$ , add an edge between the two containing nodes in  $G^*$ . Now we can run our algorithm from part (a) again to complete the MST.