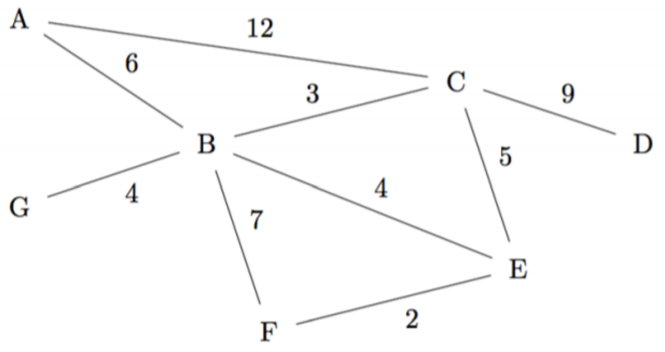
CS 61B Exam Prep 13: Graphs Spring 2020

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 smallext 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 grpah 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 Graph Conceptuals

Answer the following questions as either **True** or **False** and provide a brief explanation -

(a) If a graph with n vertices has $n-1$ edges, it **must** be a tree.
False. The graph **must** be connected.

(b) The adjacency matrix representation is **typically** better than the adjacency list representation when the graph is very connected.
True. The adjacency matrix representation is usually worse than the adjacency list representation with regards to space, scanning a vertex's neighbors, and full graph scans. However, when the graph is very connected, the adjacency matrix representation has roughly same asymptotic runtime in these operations, while "winning" in operations like hasEdge.

(c) Every edge is looked at exactly twice in **every** iteration of DFS on a connected, undirected graph.
True. The two vertices the edge is connecting will look at that edge when it's their turn.

(d) In BFS, let $d(v)$ be the minimum number of edges between a vertex v and the start vertex. For any two vertices u, v in the fringe, $|d(u) - d(v)|$ is **always less than** 2.
True. Suppose this wasn't the case. Then, we could have a vertex 2 edges away and a vertex 4 edges away in the fringe at the same time. But, the only way to have a vertex 4 edges away is if a vertex 3 edges away was removed from the fringe. We see this could never occur because the vertex 2 edges away would be removed before the vertex 3 edges away!

(e) Given a fully connected, directed graph (a directed edge exists between every pair of vertices), a topological sort can never exist.
False. Consider the graph constructed as follows: for all vertices i, j such that $i < j$, draw a directed edge from i to j. A valid topological ordering of this graph is simply enumerating the vertices: $1, 2, 3, \ldots .N$.

3 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/marked yet. In a sorted list of nodes, we can simply traverse through the nodes in order. Therefore, our algorithm is simply to run Dijkastra's, but instead of keeping a priority queue we go through our sorted list of nodes in order. Our runtime is $O(V+E)$

4 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.