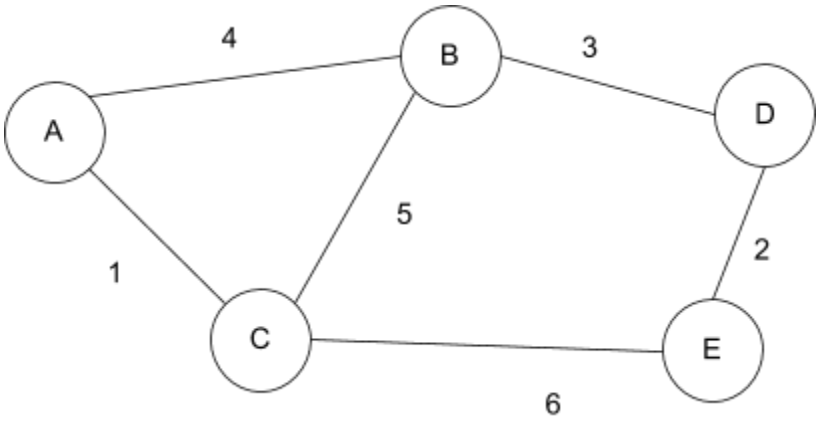
DSA Homework

Katie Foster

Part 1



Suppose there are two minimum spanning trees for a graph in which all edge weights are unique.

That means that T1* has an edge that T2* does not have. Consider the edge e_i that is is in T1* but not T2*. If e_i were added to T2*, since T2* is already a minimum spanning tree, then adding one extra edge would create a cycle. Once a cycle is created, the greatest cost edge can be removed in order to create a minimum spanning tree. If you removed the maximum edge from T2*+e_i, you would either have to remove e_i, or some other edge. All edge weights are unique, so it is not possible that e_i and one of the other edges in the cycle have the same weight. If you removed e_i, that would mean that e_i had a greater weight, which means that T1* was not a minimum spanning tree. This is a contradiction, since it was stated that T1* and T2* were both minimum spanning trees.

Part 2

a)

Consider the closest node v to source i, whose BFS path is not one of the possible shortest paths. In order for v to be part of the path, v would have to be marked as a child of u, so one of two cases would happen.

The first case is that v would have to be dequeued before u was dequeued. This means that another node x was processed before u, despite u-v being the last edge in the path.. If x was processed before u, that means that i->x->v is a possible shortest path, which is a contradiction.

The other case is u is added to the queue before processing v. We assume that up until u, the path is correct. If v is added to the queue while u is being processed, this means that the distance from i to v is the closest path. If u to v is the last edge, that means that the path outputted by the algorithm is actually one of the shortest possible paths, which is a contradiction.

b)

I would turn a weighted graph into an unweighted graph by adding n-1 nodes along each weighted line, which n is equal to the weight of each line. Since each edge in an unweighted graph has a weight of 1, adding n-1 extra nodes along each edge would make each edge effectively have a weight of n.

c)

Using your construction above, explain how to find the shortest path from i to all other nodes j in a weighted graph G = (V, E) using BFS. What is the runtime of your overall algorithm? How does it compare to the runtime of Dijkstra's algorithm?

Pseudocode

- Mark all nodes as unvisited (create dictionary, o(|V|))
- Create queue Q (represents nodes with unvisited neighbors) O(1)
- Add v to Q, and mark visited O(1)
- Create a linked list to keep track of path O(1)
- While Q not empty: O(|E|) * (everything in loop)
    - Mark u as v's parent O(1)
    - Dequeue u from Q, mark u visited O(1)
    - Add U's unvisited neighbors to Q O(1)
- Return the linked list that keeps track of path

The runtime of Breadth First Search would be O(|V|+|E|). The runtime of Dijkstra's algorithm can get down to O(|E| + |V|log|V|). It seems like BFS would be faster than Dijkstra's algorithm, however, when comparing the two, it is important to keep in mind that the number of vertices and edges for BFS is really

$$|V| \text{ or } |E| \text{ of original graph} + \sum \text{weights}$$

since according to the design in part b, n-1 nodes would need to be added to the graph in order to create an unweighted representation of a weighted graph. Therefore BFS might appear slightly faster, but it is likely that in reality the algorithm would be much slower on a weighted graph with big weights.