Lab 5-1 Minimum Spanning Trees

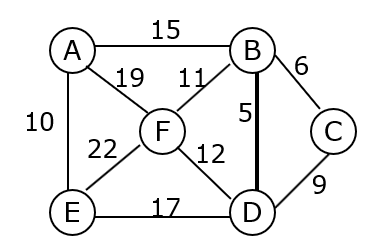
1. Consider an undirected graph g = ( V, E ) with non-negative edge weights we ≥ 0. Suppose you have computed a minimum spanning tree for G, call it Tmst . Suppose each edge of the graph is now increased by 1, the new weights are now we‘ = we + 1 . Does the minimum spanning tree always contain the same edges?   
   If yes, justify (not a formal proof) your answer. If no, give an example where the edges in the MST change.

Yes, because if you increase the weights of all the edges, the cut property still holds, and the same minimum edges will be chosen. You will still pick the next lowest edge since all edges were increased.

1. Does Prim’s algorithm always work correctly on a connected graph with negative edge weights?   
   If yes, justify (not a formal proof) your answer. If no, give an example where the edges in the MST are different from those found by Prim’s Algorithm.

Yes, since Prim’s takes the lowest weighted edge that does not form a cycle in the tree. The negative edge weights also follow this algorithm since we just want the smallest edge that does not form a cycle with the current MST.

1. Trace Prim;s algorithm as it constructs the Minimum Spanning Tree for the following graph.



Each line of the following show the state of a Priority Queue implemented as an unsorted list where the first entry is the other vertex on the shortest edge that connects to the existing tree. The first column contains the vertices that are already in the tree constructed by Prim

MST A B C D E F

INIT ( ꟷ, 0) ( ꟷ, ∞ ) ( ꟷ, ∞) ( ꟷ, ∞) ( ꟷ,∞) ( ꟷ,∞)

A ( ꟷ, 0) (A, 15) ( ꟷ, ∞) ( ꟷ, ∞) (A, 10) (A, 19)

E (DONE) (A, 15) ( ꟷ, ∞) ( E, 17) (ꟷ, 0) (A, 19)

B (DONE) (ꟷ, 0) (B, 6) (B, 5) (DONE) (B, 11)

D (DONE) (DONE) (B, 6) (ꟷ, 0) (DONE) (B, 11)

C (DONE) (DONE) (ꟷ, 0) (DONE) (DONE) (B, 11)

F (DONE) (DONE) (DONE) (DONE) (DONE) (ꟷ, 0)

1. Determine the computational complexity of Prim’s Algorithm where the Priority Queue is implemented as an unorder list of vertices. Justify your answer.

The complexity would be O(|E| \* |V|) since you have to go through the list of vertices each iteration and update the priority by checking each edge.

1. Suppose we are given a minimum spanning tree problem on a graph G with all edge costs positive and distinct. Let T be the MST for this instance. Now replace each edge cost by its square that is ce is replaced by is ce2. This creates a new instance of the MST problem. Must T still be a minimum spanning tree for this problem. Justify your answer.

Yes, T is still the MST for this problem. Since you are squaring all of the edge weights, when you compare the edges to find which edge is the lowest and doesn’t form a cycle with the existing MST in Prim’s algorithm, the lowest edge does not change. Even though the total weight of the MST is increased, you are still using the same edges from T because the closest edge did not change.

1. Suppose that we represent the graph G- (V,E) as an adjacency matrix, Find a simple implementation of Prim’s algorithm that runs in O(|V| 2).

Start with a vertex v in tree T. Then go through the adj matrix of all the vertices in T and get the minimum edge of a vertex, w, that is not already in T (takes O(|V|) time). Add vertex, w to the spanning tree of T. Repeat until all vertices added to T (repeat |V| times). Total complexity is O(|V| 2).

1. Trace Kruskal’s Algorithm as constructs the Minimum Spanning Tree for the same graph using the last union find structure discussed.

Initialize

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Symbol | A | B | C | D | E | F |
| Parent | A | B | C | D | E | F |

Process edge (B,D) breaking the tie alphabetically by making the earlier vertex the root

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Symbol | A | B | C | D | E | F |
| Parent | A | B | C | B | E | F |

Process edge: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Symbol | A | B | C | D | E | F |
| Parent |  |  |  |  |  |  |

Process edge: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Symbol | A | B | C | D | E | F |
| Parent |  |  |  |  |  |  |

Process edge: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Symbol | A | B | C | D | E | F |
| Parent |  |  |  |  |  |  |

Process edge: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Symbol | A | B | C | D | E | F |
| Parent |  |  |  |  |  |  |

Process edge: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Symbol | A | B | C | D | E | F |
| Parent |  |  |  |  |  |  |

1. Determine the computational complexity of Kruskal’s Algorithm using different implementations of the **Union-Find** data structure.
   1. Quick Find
   2. Quick Union
   3. Weighted Quick Union