Exercises 8 – minimal spanning trees (Prim and Kruskal)

Questions

1. Suppose we have an undirected graph with weights that can be either positive or negative. Do Prim's and Kruskal's algorithim produce a MST for such a graph?

2. Consider the problem of computing a *maximum* spanning tree, namely the spanning tree that maximizes the sum of edge costs. Do Prim and Kruskal's algorithm work for this problem (assuming of course that we choose the crossing edge with maximum cost)?

3. Prove that for any weighted undirected graph such that the weights are distinct (no two edges have the same weight), the minimal spanning tree is unique. (See lecture 8, slide ~15).

4. Cycle Property: Let G be an undirected connected weighted graph. Suppose the graph has at least one cycle (choose one). For that chosen cycle, let edge e be an edge that has strictly greater cost than all other edges in the cycle. (Such an edge might not exist, e.g. there might be two edges that have the same greatest cost). Show that e does not belong to any MST of G.

5. Consider a "reversed" Kruskal's algorithm for computing a MST. Initialize T to be the set of all edges in the graph. Now, consider edges from largest to smallest cost. For each edge, delete it from T if that edge belongs to a cycle in T. (Never mind how to implement this. Just note that union-find does not allow deletions, so an inefficient implementation of this reversed Kruskal is not obvious.) Assuming all the edge costs are distinct, does this new algorithm correctly compute a MST?

6. (From Sedgewick's course) Given a connected graph G with positive weights, which of the following will compute an MST of G?

    a) Change the weight of each edge from c(e) to c(e) + 17.
    b) Change the weight of each edge from c(e) to 17 * c(e).
    c) Change the weight of each edge from c(e) to c(e)*c(e).
    d) All of the above.
    e) None of the above.

7. If a connected undirected graph has n vertices, then any spanning tree has n-1 edges.

8. Suppose you have n objects and you define a distance between them. We can represent these distances as edges in a weighted graph. An interesting and useful problem is to group these objects such that objects within a group have a small distance between them, and objects across groups have a large distance between them. This is sometimes called "clustering".

    One way to compute clusters is to run Kruskal's algorithm, but stop it before the MST has been computed. For example, you could stop it after k edges have been added to T. How many connected components will you have, in that case? What can you say about the distances of the crossing edges between these components?

Answers

1. Yes. Add some sufficient large positive constant (the absolute value of the smallest cost edge) to all edge costs so that all edge costs become non-negative. Then apply either MST algorithm. It should be clear that Kruskal's algorithm will pick the same set of edges, regardless of whether we add a constant to them since the selection of edges only depends on the ordering of costs. For Prim, the same argument can be made. At each stage of Prim, a crossing edge of least cost is chosen. The choice won't change if there is a constant added to all edges.

   The issue we had with Dijkstra's algorithm and negative edges was that some paths might be longer than others. In Dijkstra's algorithm, we chose crossing edges based on the sum of costs along a path plus the cost of a crossing edge. That choice might be different if a constant is added to each edge in the graph.

2. Yes, all the proofs are the same. Just flip the inequalities. Another way of seeing that there's no difference in the problems is to just flip the sign on all the edges. Finding the maximum spanning tree is the same problem as finding the minimum spanning tree in a graph which had costs negated (relative to the originals). As we saw from the previous question, having negative costs doesn't change the correctness of the MST algorithms.

3. Suppose there are two MSTs, call them T1 and T2. Let the edges of T1 be {e_i1, e_i2, … e_i_n-1} and the edges of T2 be {e_k1, e_k2, … e_k_n-1}. If the trees are different, then these sets of edges must be different. So, let e* be the smallest cost edge in T1 that is not in T2. Deleting that edge from T1 would disconnect T1 into two components. Since T1 chose that edge, it must be the smallest cost crossing edge for those two components. But by the cut property, that edge must therefore belong to every minimum spanning tree. Thus it must belong to T2 as well. But this contradicts the definition of e*.

4. Suppose e does belong to a MST, and derive a contradiction. Consider the disconnected subgraph with edges T \ {e} which has two connected components. Since G has a cycle containing e, there must be another edge in the cycle that connects the two components of T \ {e}. Let this edge be (u,v). This edge has smaller cost than e (since otherwise we would have removed it earlier). But then one could then define a spanning tree of lower cost by deleting e from T and adding this (u,v) edge, which contradicts our assumption that e belongs to the MST.

5. Yes it does. At stage k (starting a k=1), the algorithm considers the kth largest cost edge. If that edge belongs to a cycle in the remaining graph T, then all edges in that cycle (and indeed in T) must have smaller cost than the edge being considered. Thus, the edge cannot belong to the MST (by the previous question).

   The algorithm cannot terminate with T having a cycle, since the algorithm would have considered each edge in such a cycle and would have removed the edge with the largest cost when it considered that edge. The algorithm also cannot terminate with T being disconnected, since edges are only removed when they belong to a cycle, and disconnecting an edge that belongs to a cycle does not disconnect the graph. Thus the algorithm terminates with T being a spanning tree. It is the MST because all the edges that were removed have the property that they cannot belong to the MST. Since the only edges that could belong the MST are the ones that remain, and they indeed define a spanning tree, it must be the MST.

6. The answer is (d) all of the above. The choices made by Prim's and Krustal's algorithms only depend on the ordering of edge costs. The changes in edge weights in (a)-(c) don't change the ordering of edge costs.

7. Consider the edges in a spanning tree T and consider a graph with no edges, but all n vertices. Now add the edges of the spanning tree one by one. Each edge is a crossing edge between two connected components and adding the edge reduces the number of connected components by 1. Since adding all the edges of T (one by one) reduces the number of connected components from n down to 1, it must be that T has n-1 edges.

8. Kruskal's algorithm starts off with |V| components. Each edge that it adds is a crossing edge between two connected components, and merges these two components into 1. Hence each edge reduces the number of connected components by 1. Thus, after adding k edges, we have |V|- k connected components. If the kth added edge has cost c, then all edges that haven't been considered by Kruskal's algorithm must have cost greater than c. In particular, all crossing edges must have cross greater cost than this. Moreover, all edges within components (that have been added) must have cost less than c.