Assignment 3

CSCI 270

Paul Kim | pbkim@usc.edu | 1723717002

Question 1 1

Question 2 4

Question 1

*Collaborated and discussed ideas with Matt Quan and James Fan on Friday, Feb 11 with the partial presence of CP Aviv at Cardinal Gardens. Discussed Q1, regarding whether to use Prim's or Kruskal's (we determined we can use both). We also discussed what happens to a spanning tree when you add an edge in Q2, and discovered a key observation of cycles.

(1) [10 points]
Recall that we motivated the Minimum Spanning Tree problem as follows: There is a graph $G = (V, E)$, with edge costs c_e which we assumed (and will continue to assume) to be all distinct. We then were talking about building the cheapest possible connected "backbone" $E' \subseteq E$, i.e., the cheapest set of edges such that the graph (V, E') is still connected. We then mentioned that this cheapest edge set E' always forms a tree.

Now suppose that for some subset of edges $\hat{E}$, you are *required* to include them in the solution.[1] So now, you want to find the cheapest set $E' \supseteq \hat{E}$ such that the graph (V, E') is connected. It is not ruled out that $\hat{E}$ might contain cycles already, so the output may not be a tree any more.

Give an efficient algorithm (running time no worse than $O(mn)$, though ideally, it should be $O(m \log n)$) that computes the cheapest such edge set E'.

Answer:

*We will continue to assume that all edges in graph G = (V, E) are distinct (Prompt).

A minimum spanning tree is the cheapest possible connected ”backbone”, i.e, the cheapest set of edges such that the graph (V,E) is still connected. However, in this instance, we assume that there is a subset of edges (we will call this edge ”*e*”) that are required to be in the solution of the MST, and therefore, must be included edges. Our goal is to find the minimum cost subset of edges in the graph such that every edge of *e* is in the solution and that the graph defined by (V,E) is connected. We want to give an efficient algorithm that ideally runs in O(m log n) such that it computes the cheapest edge set E'. (*M is the number of edges, and N is the number of nodes).

Such an algorithm that can be used to achieve this task is in fact, Kruskal's Algorithm, which is a greedy algorithm used to produce a minimum spanning tree of G. The idea of Kruskal's is that the initial constructed graph starts without any edges at all, and builds a spanning tree by successively inserting edges ”e” in order of increasing cost. As we move through the edges in this order, we insert every edge e as long as it does not create a cycle when added to the edges we've already added. If it does create a cycle, we discard and move to the next. In each iteration of the algorithm, will see that this general procedure will create a MST (edges are added to minimize costs). However, for the case of this problem, we don't want to follow the precise definition of the algorithm, and instead desire to make a few modifications. Since we are given an additional input of edges *e*, in our algorithm, we first want to connect and add all these *e* edges in our solution, because it must be true that all of these edges must be in the "tree". From here, we can use our intuition and think of every *e* edge and it's connection to other nodes as a connected component (for intuitively a cluster if there are multiple nodes connected by *e*). For example, if there are 3 (*e*) edges that connect 3 nodes together (completely valid if a cycle as well), we can possibly imagine this connected component as a singular node (basically, a cluster). Then at this instance, it must be true that we included all of the *e* edges and now we can construct/add edges to connect all these "components" in the cheapest way possible. We can achieve this by simply using Kruskal's algorithm, and adding the edges in increasing order that will connect the components that will not form a cycle with the connected components. Generally, we are trying to connect all these connected components of *e*. Therefore, by this algorithm, we will construct a minimum connected graph (that adds edges as cheapest as possible, while ensuring no cycles).

*We also make note that there is a possibility that the solution could also not be a Minimum Spanning "Tree", because there is a chance that the solution can contain a cycle (if *e* (the required edges) is a cycle) because it must be true that all of *e* is included. Therefore, what we can do, is imagine that this "cycle"/connected-components is itself a singular node. Then what we can just use Kruskal's to find the minimum cost edge that connects all of these "Components" together, without forming any "cycles."

*We will use the Union-Find data structure which stores representation of components in a way that supports rapid searching and updating. As each edge e = (V,W) is considered, we need to efficiently find the identities of the connected components containing v and w. If these components are different, and there is no path from v and w, then edge e should be included; if the components are the same, then the edge should be emitted. In sum, the Union-Find data structure allows us to maintain disjoint sets. (Textbook CH 4.6).

*I will not describe how UnionFind works, but merely just use the data structure and assume it works. (Referenced Textbook, Kempe Notes, Piazza @177)

To Formally State The Algorithm: // Modified Kruskal's Algorithm

```
    Let S be the MakeUnionFind(S), where all elements are in separate sets (singular). (O(n)
    where n is the number of nodes)

    Let T be the empty set of edges, this will be our "minimum tree". (O (1))

    We sort all of the edges (they are distinct) by cost in increasing order. We can achieve
    this by using Merge Sort (O (m log m)).

    FOR each edge *e* = (u,v) (edges that must be included) (E Carrot)

        Use the Union-Find data structure to add each edge *e* to T, and form the
        necessary unions between the sets.
```

```
*At this point, we will have included all edges of *e* and form the necessary "sets." *The
basic idea is that we are trying to "connect these clusters of nodes". These edges do not
have to be sorted, since it must be so that all edges are connected.
FOR each edge e = (u,v) from smallest to biggest (regular edges that we sorted earlier)
    if U.find(u) != U.find(v)
        Add the edge to T.
        Call U.Union (u,v)
        *We are using the Union-Find data structure to add each edge e to
        T, and forming the necessary unions between the sets, if they are not
        yet connected.* (This is our condition to ensure there is no cycle)
*As each edge e = (u,v) is considered, we compute to see if the following connected nodes
belong to different components, and if the algorithm decides to include the edge e in the
tree, we merge these two nodes into the same set, in increasing edge costs weights. (Thus
we will have formed our minimum tree)
Return T // This will be our solution
//
```

Total Runtime: **O (M log N)** (*Where M is the number of edges, and N is the number of Vertices) *Runtime of Kruskal's using Union-Find derived by textbook (4.25). The general runtime will be the same as proved as stated in Textbook 4.25, since we are using the Union-Find Data structure which supports the necessary operations.

Proof: // We need to prove that our solution is connected and our solution constructs a connection as cheap as possible.

Requirements: The algorithm computes the cheapest edge set E', and the algorithm includes all of *e* (the subset of edges that are required to be included in the solution), and the algorithm is connected.

*To prove that the algorithm includes all of the edges of *e*, we can say that our algorithm directly proves this because in our FIRST for loop, we explicitly iterate through through all edges *e* and form the necessary union between the sets/nodes. **Proved (because our algorithm does this**).

*To prove that the algorithm computes the cheapest edge set E' and our solution is connected. (Very Similar to Textbook/Lecture Proof of Kruskal's and MST's).

Formally:

We see that the algorithm explicitly chooses the cheapest edge for each cut along the way, so the generic analysis of MST's applies. Because each edge e added by the algorithm is cheapest across the cut (we will assume we know the Cut Property), by the cut property, they are all in the MST. However, in this instantiation, if there is a cut with edge(s) that include *e*, then the algorithm must be so that it includes those edge(s) and no others that are in the cut (these edges must be added). So T is a subset of the MST and in the end, T is **connected** by the cheapest cost... T = MST.

We also want to show that our algorithm is a special case of our generic algorithm as proved above.

We want to show that for every edge that our algorithm adds, we need to find a cut for which that edge is the cheapest. After some iterations, the set T is a forest, creating 2 or more connected components through the Union-Find data structure. When the edge e = (u,v) is added, it connects the two connect components (whether these components would be single nodes or clusters), however, because e is the cheapest edge (as we sorted the edges in increasing cost), it is the cheapest cut between our two connected components. We do this until all components are connected, thus we prove that our solution is connected because we know all the nodes are connected through the UnionFind sets, and it is the most cheapest through the Kruskal's property.

Question 2

(2) [5+5=10 points]
In this question, we will look at arbitrary spanning trees, not necessarily minimal ones. Unless the input graph is itself a tree, there will now be multiple spanning trees, and they may look very different. For an input graph $G = (V, E)$, let T and T' be two different (possibly very different) spanning trees. Prove the following two exchange properties:

(a) For every edge $e \in T \setminus T'$, there exists an edge $e' \in T' \setminus T$ such that $T \cup \{e'\} \setminus \{e\}$ is also a spanning tree.

(b) For every edge $e' \in T' \setminus T$, there exists an edge $e \in T \setminus T'$ such that $T \cup \{e'\} \setminus \{e\}$ is also a spanning tree. [Notice that the quantifier order for e, e' is switched compared to the previous subproblem.]

[Note: the exchange properties you are asked to prove imply that you can gradually go from T to T', or from T' to T, by exchanging one edge at a time.]

Answer:

We look at arbitrary spanning trees, not minimal ones. This implies that there are multiple spanning trees that could look very different with different paths/combinations of edge costs. We keep in mind that a tree contains no cycles at all.

1. *Prove that for every edge of T, you can take it out and replace it with an edge of T' to get another spanning tree.*

   We use our intuition and know that since T and T' are both spanning trees of the same vertices, then that means that the number of edges between both trees are the same (cardinality). If we take a valid edges from T and replace it with a valid edges from T' and replace them, then it is obvious that we will get another spanning tree because there is always one path and were never cycles to begin with.

   Formally: Proof By Contradiction:

   Suppose that there exists an edge e that is in T and not in T', and there does not exist a suitable replaceable edge e' in T' that is not in T. When we remove T - e, we can conclude that T is not a spanning tree anymore, because it was so that T was already a spanning tree and we just removed an edge. Therefore, now we have a disconnected graph. We have two situations here if we add this unique edge e' into T. If there is no way to replace e with e' then that implies there is no other edge connecting that vertex from the graph; and therefore, we will not have a minimal spanning tree and our graph is disconnected... e' must exist. If there does exist a path between the two components, then it must be the case that every single possible spanning tree must have this edge and therefore... e' must have existed. In conclusion, we shown that it is not possible to find an edge for which e' is not suitable by proof of contradiction.

2. *Prove and show that every edge that you want to use as replacement can be used as replacement by taking out the right edge e from T.*

   For every edge that is in T' there exists an edge in T such that you can just swap them and T remains a spanning tree.

   *Should be very similar to proof above (quantifiers are different)

   The general intuition here is that if you add an edge to a minimum spanning tree, you will evidently create a cycle. However, if you were to take out an edge from this cycle, you will still have connected components. This also satisfies the above intuition that the number of vertices and number of edges should and will be the same for both spanning trees.

   Formally:(Direct Proof)

   Let us consider a cycle C as we add T U e'. By doing this, we are adding an edge that defeats the spanning tree definition, and now there exists two such edges such that one of them is e and another is e' that creates a cycle. Now let us consider removing the edge e from this newly formed cycle. What we will continue to get here, is that as we remove this edge, since we have already added e' that formed a cycle, removing this edge, will end the cyclical behavior and the nodes will still be connected through the spanning tree property. In addition, the cardinality will still be the same as a result of this subtraction of edges, and therefore has the same number of edges that a spanning tree will have. By the property/definition of a spanning tree, we will still have connected components as a result of adding an edge to form a cycle, and taking out an edge from that same cycle to have connected components. The solution therefore is a spanning tree. To sum the idea, we know that e and e' belong to the same cycle, and it is completely valid to remove e from this cycle, to still hold the spanning tree property with the addition of e'.

   *General Idea: Think about a spanning tree. If you add any edges to it, you will create a cycle. When you think about a cycle, you can remove any edge from the cycle, and you will still have a spanning tree.

   *More generally, we can use the same intuition and apply this idea if it creates a circuit. Adding an edge that makes a circuit, and taking out another edge from this circuit will remove the ”circuitry”.

   *Cycles and Circuits and the proof's properties are synonymous and interchangeable.