CS 577 Homework 4

Alex Pizzuto

Question One: 4.18

We seek to create a modification of Dijkstra's Algorithm that can account for potentially varying edge costs. To do this, we will store not only the total summed edge cost to travel to a given node, $v \in V$, but we also keep track of the last node visited before v, and will store the set of nodes that have already been explored as S. Some notation, we will denote the earliest possible time that has been found for a given node, v, as $t_{min}(v)$, and the node immediately preceding v in the fastest path as $\alpha(v)$. With this, we have the following algorithm:

```
Start with $S = s$;
while $S \neq V$ do
    Find a node $v \in V - S$ that is connected to at least one node in $S$;
    Minimize $t_{min}(v)$ by searching for the minimum of $\min_{e=(u,v), u \in S} f_e(t_{min}(u))$;
    Add this node, $v$, to $S$, and set $t_{min}(v)$ as the minimum found in the last step;
    Set $\alpha(v)$ as the node that produced this minimum time;
end
```

Algorithm 1: FastestTravel Algorithm to find the fastest route to a destination given the possibility of predictably changing edge costs

Now, we must still prove correctness and optimality as well as analyze its complexity. We will prove the optimality and correctness via induction on the size of the explored set, proving at each level that all minimum times are the minimum attainable times given our root node.

Proof: By Induction. *Basis of Induction.* First, we look at when $|S| = 1$. Clearly, we have the minimum size for all elements because our set only contains node s, and $t_{min}(s) = 0$.

Inductive Step: Now, assume that $\exists k \in \mathbb{N},\ k \geq 0$, such that all nodes in S have associated with them a t_min that is the true minimum time for arrival at that node. We will show that this implies the correctness of $|S| = k + 1$. In order to do this, we consider the t_{min} obtained from adding the new node, v, and prove that the time given by any other algorithm (we will just use $t(x)$ to denote the time to encounter node x in this algorithm) is at least t_{min}. Clearly, in order for the path from our root, s, to v to be different from the path from our algorithm but still be a contender for an optimal path, the path must contain at least one node, $w \notin S$ (returning a different path completely within S is not a contender for an optimal path based on our inductive hypothesis). However, our algorithm guarantees that the travel time to w is at least as large as the travel time to v, otherwise we would have added w to our set S before v. This means that $t(w) \geq t_{min}(v)$, and as w is encountered before v in this purported optimal solution, we know that the $t(v) > t(w) \geq t_{min}(v)$, and thus our time is truly the minimum, and thus our inductive claim holds.

Finally, we must analyze the complexity. We have m iterations of our loop, as we must add nodes until our set is the size of V. In each loop iteration, we consider all edges connected to v, and if implemented as a priority queue as it is suggested for the normal Dijkstra's algorithm, then each iteration takes $\mathcal{O}(\log n)$ time. thus, overall, we have achieved $\mathcal{O}(m \log n)$ complexity, which is better than the question asks, but if we did not use a data structure like a priority queue for implementation, this would likely end up being a quadratic complexity time.

Question Two: 4.21

We seek to find an $\mathcal{O}(n)$ algorithm that returns a MST of the graph G, where $n = |V|$, and we know that the graph is connected and has *at most* $n + 8$ edges. If we seek to find an MST, then we know that we must have exactly $n - 1$ edges. In order to do this, we make use of the cycle property k times, where $k = |E| - n + 1$, so we will be doing this at most 9 times. Each iteration will be a breadth first search for cycle detection, and then once we have found a cycle, we will delete the edge with the highest cost. By the Cycle property, the omission of this edge does not alter the connectedness of the tree. We know that the resulting tree is optimal based on the various proofs that we have done on MSTs in class, ie, the optimal MST is obtained when we delete or add edges based off of the cycle or cut properties, respectively.

In order to analyze the complexity, we will analyze the worst case complexity (ie when we have $m = n + 8$ edges). In this case, we must perform 9 iterations of a breadth first search, each of which being $\mathcal{O}(m + n)$, and then we must find and delete the maximum from each cycle, which is also linear complexity. Thus, our overall complexity, as in this problem we are constrained to let the number of edges grow, at worst, linearly with the number of nodes, we have $\mathcal{O}(n)$.

Question Three: 4.10

a We seek to write an efficient algorithm to test if a given MST T from a graph G is still an MST if we add a new edge connecting two nodes in G. In order to do this, we just incite the Cycle Property. More specifically, call the new edge, $e = (u, v)$. Then if T is an MST, then there already exists a path, P, in T that connects u to v. We have seen that if we use an adjacency list, then we can find a path such as P in linear time in the number of vertices (i.e. $\mathcal{O}(|V|)$), so we will use this as the data structure to represent our tree. Then, if all edges on P have a cost less than e, then by the Cycle Property e is the edge with the maximum cost in a cycle, and is therefore not in the MST, and therefore T is still an MST. However, if there is any edge with a cost greater than e on P, then it no longer belongs in the MST, as it is now the maximum edge in a cycle, and by the Cycle Property must be deleted. In this case, T is no longer the MST.

b If it is the case that T is no longer an MST, then we create the algorithm to update T to be an MST in time $\mathcal{O}(|E|)$. Clearly, in this case, there is an edge in T that would be in a cycle if we add e to G with a larger cost than e. Our way of updating the tree is to replace this edge with e, and then our algorithm terminates after having replaced the heavier edge, call this edge d, with e. First, a note on complexity. We must find the maximum edge cost, which means a scan over the edges, and thus an $\mathcal{O}(|E|)$ complexity. Additionally, to show our algorithm is correct, we prove that the resulting tree is the MST. In order to do this, we prove why all edges that are excluded by our new tree do not belong in the MST. Clearly, edge d does not belong, we have already eliminated it by the Cycle Property. Now, consider all edges that are not d or edges in our new tree. Then these edges also do not belong in the MST as the edges selected to create an MST before the addition of e generates the lowest cost subset of edges, thus excluding the other edges in the original G. Thus, we have the lowest cost subset of edges, and our graph is connected, and thus we have an MST.