Written Assignment 1

CS 362/561

February 1, 2022

Damian Franco

1 4.18

***Q*uestion:**

Your friends are planning an expedition to a small town deep in the Canadian north next winter break. They've researched all the travel options and have drawn up a directed graph whose nodes represent intermediate destinations and edges represent the roads between them.

In the course of this, they've also learned that extreme weather causes roads in this part of the world to become quite slow in the winter and may cause large travel delays. They've found an excellent travel Web site that can accurately predict how fast they'll be able to travel along the roads; however, the speed of travel depends on the time of year. More precisely, the Web site answers queries of the following form: given an edge e = (v, w) connecting two sites v and w, and given a proposed starting time t from location v, the site will return a value $f_e(t)$, the predicted arrival time at w. The Web site guarantees that $f_e(t)$ t for all edges e and all times t (you can't travel backward in time), and that $f_e(t)$ is a monotone increasing function of t (that is, you do not arrive earlier by starting later). Other than that, the functions $f_e(t)$ may be arbitrary. For example, in areas where the travel time does not vary with the season, we would have $f_e(t)$ = t + l_e, where l_e is the time needed to travel from the beginning to the end of edge e.

Your friends want to use the Web site to determine the fastest way to travel through the directed graph from their starting point to their intended destination. (You should assume that they start at time 0, and that all predictions made by the Web site are completely correct.) Give a polynomial-time algorithm to do this, where we treat a single query to the Web site (based on a specific edge e and a time t) as taking a single computational step.

***S*olution:**

Within this problem, we are asked to find the smallest/fastest time to travel through the Canadian north. The algorithm will have to accurately predict the smallest "path" and the minimum time of arrival which will be denoted by $f_e(t)$. Time t will be considered edge weights between the edges $e = (v, w)$ and will start at 0. If traveling times vary within each season then the new function for time of arrival will be $f_e(t) = 1 + l_e$ Lastly, we must note that we cannot travel backward in time and that time is always

To provide a clear and efficient algorithm, I first focused on the main goal which was to find the minimum time of arrival. The destinations here create a directed graph. The graph will include the time it will take to traverse through each destination. Knowing that, we will now have a start and last node in which both are indicated to the algorithm by the user. A greedy algorithm similar to Dijkstra's or Prim's algorithms will be used in this case to find the fastest time. My take on the algorithm will assume that the graph has set distances/time to travel between each node. Another note, the graph being directed will not impact the way the algorithm functions. Now the algorithm can begin.

First, the algorithm needs to keep track of the ending node by setting the

time traveled of the node to zero. This is where the algorithm will begin and end after all nodes have been visited. Set the start node to the current node. All nodes will now be marked as not visited. Next, the algorithm will find all the neighbors of the current node and calculate the distances/time it takes to travel to each neighbor. The node with the minimum time to travel to will be the next node in the algorithms step. Set the current node to visited and traverse to next node (with the smallest time to travel towards) and set to current. Since we need to find the fastest full traversal path to the end node, after each step in the algorithm each node with the smallest time to travel towards will be added to a list full of the shortest path nodes. This process will then repeat with each node until all nodes have been visited. After all nodes have been visited, the algorithm will then output the minimum time it will take to traverse to the ending destination and the fastest path. Since this algorithm is very similar to Dijkstra's or Prim's algorithms, the output of the fastest path could be thought of as a Minimum Spanning Tree.

Algorithm 1 Pseudocode for the Fastest Way algorithm

$G \leftarrow$ Set of all nodes within the graph
$Visited \leftarrow$ Set of all visited nodes
$QuickPath \leftarrow$ Set of the quickest path nodes found
$currNode \leftarrow$ Start node indicated by user
Add start node to QuickPath
$t = 0 \leftarrow$ Time is set to 0
while all nodes in G != Visited **do**
    $nextNode =$ neighbor of currNode with at least one edge between itself and the current node and with the minimum time to travel ($f_e(t)$)
    $t = t+ (f_e(t)) \leftarrow$ Add time to travel to the overall time
    Add nextNode to QuickPath
    Add currNode to Visited
    $currNode = nextNode$
end while
Return t
Return $QuickPath$

The algorithm will loop until all nodes have been visited in the graph. Once all nodes have been visited, the shortest path and minimum time traveled will be found and returned to the user. Overall, this algorithm should efficiently fulfil all the goals set. Run-time here will be based on the amount of edges that are traversed through. Each operation within the algorithm takes *O(logn)*. Knowing that it will take n operations, total running time should be *O(nlogn)* which adheres to the polynomial-timed requirement.

2 4.20

Question:

Every September, somewhere in a far-away mountainous part of the world, the county highway crews get together and decide which roads to keep clear through the coming winter. There are n towns in this county, and the road system can be viewed as a (connected) graph G = (V, E) on this set of towns, each edge representing a road joining two of them. In the winter, people are high enough up in the mountains that they stop worrying about the length of roads and start worrying about their altitude—this is really what determines how difficult the trip will be.

So each road—each edge e in the graph—is annotated with a number a_e that gives the altitude of the highest point on the road. We'll assume that no two edges have exactly the same altitude value a_e. The height of a path P in the graph is then the maximum of a_e over all edges e on P. Finally, a path between towns i and j is declared to be winter-optimal if it achieves the minimum possible height over all paths from i to j.

The highway crews are going to select a set E' in E of the roads to keep clear through the winter; the rest will be left not maintained and kept off limits to travelers. They all agree that whichever subset of roads E' they decide to keep clear, it should have the property that (V, E') is a connected subgraph; and more strongly, for every pair of towns i and j, the height of the winter-optimal path in (V, E') should be no greater than it is in the full graph G = (V, E). We'll say that (V, E') is a minimum-altitude connected subgraph if it has this property.

Given that they're going to maintain this key property, however, they otherwise want to keep as few roads clear as possible. One year, they hit upon the following conjecture:

The minimum spanning tree of G, with respect to the edge weights a_e, is a minimum-altitude connected subgraph.

(In an earlier problem, we claimed that there is a unique minimum spanning tree when the edge weights are distinct. Thus, thanks to the assumption that all a_e are distinct, it is okay for us to speak of the minimum spanning tree.)

Initially, this conjecture is somewhat counter-intuitive, since the minimum spanning tree is trying to minimize the sum of the values a_e, while the goal of minimizing altitude seems to be asking for a fairly different thing. But lacking an argument to the contrary, they begin considering an even bolder second conjecture:

A subgraph (V, E') is a minimum-altitude connected subgraph if and only if it contains the edges of the minimum spanning tree.

Note that this second conjecture would immediately imply the first one, since a minimum spanning tree contains its own edges.

So here's the questions:

(a) Is the first conjecture true, for all choices of G and distinct altitudes a_e? Give a proof or a counterexample with explanation.

(b) Is the second conjecture true, for all choices of G and distinct altitudes a_e? Give a proof or a counterexample with explanation.

***S*olution(s):**

In this problem, an algorithm is presented that will see highway crews selecting the edges/roads of a graph by the minimum altitude a_e. These selected edges will be the roads the crew will keep clear of ice and snow throughout the winter seasons. Altitude of each edge a_e will be the main factor that will indicate which roads need to be kept clear. This in turn will create a subgraph of the roads that are kept clear and have the minimum altitude out of the whole graph *G*. The subgraph will be denoted by (V, E') and should be connected to the full graph *G*. The algorithm here will have an output of a Minimum Spanning Tree, hence the subgraph of edges that will need to be cleared. There are two conjectures given and we have to decide whether each conjecture is true or false.

The first conjecture states that the minimum spanning tree of G, with respect to the edge weights/altitude level a_e, is a minimum-altitude connected subgraph. Since the algorithm here has a Minimum Spanning Tree as the output, thinking about this as an algorithm like Dijskra's or Kruskal's helps with understanding whether these conjectures are true or false. Lets say that this algorithm runs on Kruskal's algorithm to find the MST to help figure out which roads need to be clear. In the graph G we create a subgraph and call it T which will be the indicator to see if it is of minimum altitude and connected. Lets suppose that T is connected to G. Knowing this, these components must be connected by some edge in G itself. This proves that the subgraph will be connected. Let T be the subgraph output of the algorithm and lets add a new edge e in the tree but not connected to T. The edge e here will will then create two groups, S and T. S will contain the new edge. Another edge is selected from the subgroup S which we will call edge f. Since we know that T is an MST and S is not a MST, we come to the conclusion that the altitude of $e > f$. That helps find that e must be the minimum weight throughout the entire graph G and $e < f$ which is contradicting our original claim. This proves by contradiction that the output will also be minimal, therefore the first conjecture is *true*.

The second conjecture states that the subgraph *(V, E')* is a minimum-altitude connected subgraph if and only if it contains the edges of the minimum spanning tree. Lets say that we have five edges within our subgraph *(V, E')*. Four of the edges have an altitude of 1000 and the other has an altitude of 4000. In this instance, the shortest path will continue to be the same with the subgraph being connected with the edges of the MST, and the conjecture will hold true. Now lets have the same graphs but double the altitude for each edge, this now would see a change in the shortest path that may not have all the edges that the original MST contained which is why this conjecture is *false*.

(a) This conjecture is *TRUE*.

(b) This conjecture is *FALSE*.

3 4.28

Question:

Suppose you're a consultant for the networking company CluNet, and they have the following problem. The network that they're currently working on is modeled by a connected graph G = (V, E) with n nodes. Each edge e is a fiber-optic cable that is owned by one of two companies— creatively named X and Y—and leased to CluNet.

Their plan is to choose a spanning tree T of G and upgrade the links corresponding to the edges of T. Their business relations people have already concluded an agreement with companies X and Y stipulating a number k so that in the tree T that is chosen, k of the edges will be owned by X and n−k−1 of the edges will be owned by Y.

CluNet management now faces the following problem. It is not at all clear to them whether there even exists a spanning tree T meeting these conditions, or how to find one if it exists. So this is the problem they put to you: Give a polynomial-time algorithm that takes G, with each edge labeled X or Y, and either does (i) or (ii).

(i) returns a spanning tree with exactly k edges labeled X

(ii) reports correctly that no such tree exists.

Solution:

Within this problem, we are assigned the role of a consultant that wants to help write an algorithm for the company CluNet. It is given that their plan is to generate a spanning tree with exactly k edges. The number of edges k is the amount of edges the generated tree T must have. An agreement between the companies X and Y states that k is the number of edges owned by the company X and $n-k-1$ are owned by company Y. The companies do not know if a spanning tree that meet these conditions even exists, which is why I am here to provide an algorithm that either returns a spanning tree if it exists, or report that no such tree exists.

To correctly predict if it is possible to construct a spanning tree with k edges labeled X, we must find if there is any tree that exists in the first place. Knowing that there will be two tree's constructed, one for company X and the other for company Y, the following trees are generated - T_X and T_Y. The tree we are focusing on is T_X. Knowing that T_X must have k $X-$edges allows us to think of how we might solve the problem. The first step will run Dijkstra's or Kruskal's algorithm to find a minimum spanning tree, we call this tree T_Z. For this problem, I will be using Kruskal's algorithm to generate the MST. The goal for T_Z is to find an MST with k amount of $X-$edges. Kruskal's algorithm will used multiple times to generate all the possible MST's within the graph G. If there is an MST that is found that has k $X-$edges then we will stop the algorithm there an return that MST since there happens to be a spanning tree that exists with exactly k edges labeled X.

The algorithm could stop after it finds a spanning tree with $X-$edges, but this algorithm only handles part (i) in the question so far and the run-time here would be atrocious. To handle part (ii) and make it a bit faster is a bit more complicated. First, instead of finding all MST's until the spanning tree we want is generated, we must instead focus on finding two trees T_{Z1} and T_{Z2}. The tree T_{Z1} will find the MST with the least amount of $X-$edges and T_{Z2} will be the opposite with the most amount of $X-$edges. These two trees can now be used to find the spanning tree that is needed. Similar to the first iteration of this algorithm, there must be a check to see if either tree has k X-edges. If so, then either T_{Z1} or T_{Z2} could be returned. Next, a quick check could be done to see if there is no possible spanning tree that could be found. If the number of X-edges in T_{Z1} is higher than k, and T_{Z2} is lower than k then there is no possible way to have a spanning tree with k X-edges. This will result in returning that no spanning trees exist with k X-edges. With that, this algorithm should effectively handle parts (i) and (ii) of the question and improve the run-time immensely.

Algorithm 2 Pseudocode for the Spanning Tree with k X-edges algorithm

```
G ← Set of all nodes within the graph
Run Kruskal's algorithm to find:
    Z1 ← MST with min X-edges
    Z2 ← MST with max X-edges
if X-edges in Z1 == k then
    return Z1
else if X-edges in Z2 == k then
    return Z2
else
    return nothing and state that no such tree exists
end if
```

Overall, this algorithm should run effectively if there is an efficient way to find the MST's with the minimum and maximum X-edges. Doing some research allowed me to look a bit deeper and I found that there are ways to do so, but they seem to get very complicated and thus are not reflected in the assignment. Run-time here would be similar to the first problem where each operation costs *O(logn)* with n operations with an overall run-time of *O(nlogn)*.