CS 3510: Homework 3A

Due on **Friday** Mar 18

Professor Faulkner

CS 3510 Staff

Notes:

a.) For graph algorithms you should briefly describe your algorithm in English (pseudocode will not be graded), formally prove correctness, and prove the run-time complexity.

b.) You may use any algorithm in class as a black box if you are not modifying it; if you are modifying it, you need to explain your modification in detail, clearly.

c.) You may use any fact from class without reproving it.

d.) You must provide an optimal run-time solution for full credit.

Problem 1. *(25 points)*
You are given a strongly connected graph $G = (V, E)$ with positive edge weights along with a particular node $v_0 \in V$. Give an efficient algorithm for finding shortest paths between *all pairs of nodes*, with the one restriction that these paths must all pass through v_0. Give an $O((|V| + |E|) \log |V|)$ algorithm.

a.) Describe your algorithm in plain English. Use no more than a single paragraph.

For this situation, we are able to make great utilization of Djikstra's algorithm. We can first run an implementation of Djikstra's algorithm from node v_0 to every other node to find the shortest distance between v_0 and all other nodes. Then we are able to again use Djikstra's algorithm from every other node to end at node v_0 which will find the shortest paths from any node to v_0. Then in order to find the shortest path between arbitrary nodes $v_i, v_j \in V$ we will just take the Djikstra result from $v_i \to v_0$ and concat on $v_0 \to v_j$ to get the shortest path from $v_i \to v_j$ that passes through the point v_0 as desired.

b.) Prove your algorithm's correctness.

This works because Djikstra's algorithm is an algorithm that is created in order to find the most optimal paths between two points as is desired for this problem, and we can just run it twice for $v_i \to v_0$ and again for $v_0 \to v_j$, add them together, and have the shortest path between $v_i \to v_j$ that will pass through v_0 as desired.

c.) Justify that your algorithm's runtime is $O((|V| + |E|) \log |V|)$.

This algorithm will run Djikstra's algorithm two times, $v_i \to v_0$ and $v_0 \to v_j$. This implies that it will be doing $\mathcal{O}((|V| + |E|) \log |V|)$ work $\mathcal{O}(2)$ times. Therefore, the total running time of this algorithm is $\mathcal{O}(2((|V| + |E|) \log |V|)) = \mathcal{O}((|V| + |E|) \log |V|)$ as desired.

Problem 2. *(25 points)*
$G = (V, E)$ is an undirected graph where all the edges are either red or blue. Given two vertices u, v determine if there exists a path between u, v such that the edge colors are alternating. Give an $O(|V| + |E|)$ algorithm.

a.) Describe your algorithm in plain English. Use no more than a single paragraph.

Let us label edges that are blue as True and red as False. We are going to start from vertex u, and from here we are going to check each edge. From the first node we can travel on any edge. We are going to create a switch that will be either True or False but will be initialized as Null because from the first node we can go on any edge regardless if it is red or blue. When we travel on our first edge we will set the value of the switch to True or False. Then we will travel to that node and check for each edge that is the opposite value to our switch and travel to them and check each of them subsequently checking each edge until we are able to make it to our desired vertex v. If we make it to our desired vertex we will return that it is possible to make it. Otherwise, if we make it to a vertex that does not have an alternate color edge then we will go back and check the other edges until we ether exhaust options or we make it to the desired vertex. Lastly, we will prioritize going to unvisited nodes because we need to make sure we don't accidentally go in a cycle. Lastly, we will check if a cyclic pattern occurs on one path so we can break it and prevent infinite loop.

b.) Prove your algorithm's correctness.

Using this algorithm our switch allows us make sure that we are always alternating. And since we are basically using a DFS algorithm with this additional requirement if we can travel on each edge (since they are undirected). Additionally, since if we return to a node, we could be on the different color from what we were on before, so we are able to then check this node again to see if we can now travel to new nodes.

c.) Justify that your algorithm's runtime is $O(|V| + |E|)$.

Since we are simply using DFS algorithm with an additional criteria that would be simple to add in, we would be using the running time of DFS which is $\mathcal{O}(|V| + |E|)$.

Problem 3. *(25 points)*
Consider a **connected** undirected graph $G = (V, E)$ where the weights of all the edges are non-negative and distinct. Each of the following two parts outlines a strategy for finding an MST in a graph with unique edge weights.

1. We use a **divide and conquer** strategy to find the MST. Divide V arbitrarily into disjoint sets V_1 and V_2, each of size roughly $\frac{V}{2}$. Define $G(V_1, E_1)$ as the graph with the set of edges E_1 where E_1 is the subset of E where the endpoints are in V_1. We define $G(V_2, E_2)$ in a similar fashion, where where E_2 is the subset of E where the endpoints are in V_2. We then *recrusively* find the MSTs for G_1 and G_2, labeled as T_1 and T_2 respectively. Find the edge with the least weight that crosses the cut between V_1 and V_2 and add that edge to the final MST.

2. We use a **cycle-breaking** strategy to find the MST. The strategy runs iterations until it has a spanning tree (while $|E| > |V| - 1$). In each iteration, the strategy finds some simple cycle in the graph and removes the edge with the maximum weight in that cycle. The strategy leaves us with an MST.

For each of the two strategies outlined above, prove (or disprove) whether or not the strategy returns the correct MST. If the strategy is incorrect, provide a specific counterexample.

1. This will not work. Consider if we had a graph $G = (V, E)$ for vertices $V = \{A, B, C, D\}$ with weights $w(A, B) = 1, w(B, C) = 100, w(C, D) = 2, W(D, A) = 100$. If we split this graph into $V_1 = \{A, D\}$ and $V_2 = \{B, C\}$ then we would get half weights of both 100. Then if we find the least weight that crosses these two sets, then we would add $w(A, B) = 1$ which would result in $100 + 100 + 1 = 201$. However, we could easily construct a much better tree as $w(A, B) + w(B, C) + w(C, D) = 1 + 100 + 2 = 103$.

2. This method will yield the MST of the graph. Let us denote this graph as M. Since all of the edges have distinct values, then there is only one edge that has the greatest value. Since, if we find a cycle in the graph, then we need to remove one of the edges to create an MST, so we will remove the most expensive one from it which is what the process says. Otherwise, if there is no cycle then we have to accept their values anyways.

Problem 4. *(25 points)*
Consider some digraph $G(V, E)$. We say that G is psuedo-connected if for every pair of vertices $s, t \in V$ with $s \neq t$, there is a path from s to t or a path from t to s. Design an algorithm that runs in $O(V + E)$ time that determines whether a digraph $G(V, E)$ is psuedo-connected. Give an $O(|V| + |E|)$ algorithm.

a.) Describe your algorithm in plain English. Use no more than a single paragraph.

We will design an algorithm by first putting each vertex into one of two categories, visited and unvisited. The algorithm will then work by putting the first vertex, s, into the visited list. Then we will look to see which adjacent vertices we can travel to and add the ones we haven't traveled to to the visited list. Once we have travelled to all nodes then we know that this node is pseudo-connected and we can repeat this process for the last node that gets added to the visited list from each branch we find (so every time we break from a loop and return to a previous node to check another branch), we will call t. Repeat the process starting from t and if the process is a success, then we know that the whole graph is pseudo-connected as desired.

b.) Prove your algorithm's correctness. Make sure you show both directions of the proof if necessary.

At the beginning, this is just a simple DFS algorithm on a directed graph. We simply just add a checker to make sure that we are able to travel to each node, and if we are then it is a success. The reason we are only going to repeat this process with the last node added to the original visited list is because from any node before it we know that we are able to reach that last node from some path so we are just checking if we can proceed back around to the original node which would again allow us to make it back to any other node. If this fails at any point then we know that the graph is not pseudo-connected.

c.) Justify that your algorithm's runtime is $O(|V| + |E|)$.

Since this algorithm is just a DFS on a directed graph that we are running finitely many times then the running time of this algorithm is just $\mathcal{O}(|V| + |E|)$ which is the running time of a DFS algorithm.

Problem 5. *(EXTRA CREDIT!)*
Let's get some coding practice as well! For each of the following questions you solve we'll add 2% to your exam 3 grade. The below items are hyperlinks, click them! Submit these problems on the DMOJ site.

a.) Problem 1 Djikstra's application

b.) Problem 2 Fancy Djikstra's

c.) Problem 3 DFS

d.) Problem 4 MST

e.) Problem 5 Bridges

In order to get credit for each of these problems, you need to submit a file to Gradescope under the `Exam #3 Extra Credit` assignment. You should submit a simple Python script like below named `username.py`.

```
1 def get_username():
2     """
3         Simply return your DMOJ username so we can check which problems you've solved!
4         ex) return "SimarKareer"
5     """
6
7     return "yourUsernameHere"
```

This extra credit is due a week after the exam (see Gradescope): **April 7th**.