CSCI 3110 Assignment 4 Solutions

In many of the questions on this assignment you are asked to *describe* an algorithm. This means to explain in words how your algorithm would solve the problem and then give a brief justification that your method has the claimed running time and is correct. You may give pseudocode but it is not necessary.

1. Consider the following task.

   Input: A connected, undirected graph G.

   Question: Is there an edge you can remove from G while still leaving G connected?

   (a) (10 pts) Describe a linear-time algorithm for solving this question.

   ANSWER: This problem is equivalent to cycle detection. To see this, suppose that we have a path from vertex u to vertex v and we can remove an edge on this path from G while still leaving G connected. Then there must be another path from u to v! The graph is undirected so this is a cycle. Conversely, if there is no cycle then any edge we remove will disconnect the graph. Thus, we can simply use the cycle detection algorithm shown in class which runs in linear time.

   (b) (Bonus: 5 pts) Can you reduce the running time of your algorithm to $\mathrm{O}(n)$?

   ANSWER: What do we call a connected, undirected graph G that has no cycles? A tree! Any tree has exactly $n-1$ edges, so we can simply traverse the edge list of the graph and count the edges. If we count $n-1$ edges then we return "yes" but if we reach the nth edge then we return "no". This takes $\mathrm{O}(n)$ time because we look at at most n edges.

2. Consider an undirected graph $G = (V, E)$ with distinct nonnegative edge weights $w_e \geq 0$. Suppose that you have computed a minimum spanning tree of G, and that you have also computed shortest paths to all nodes from a particular node $s \in V$.

   Now suppose each edge weight is increased by 1: the new weights are $w'_e = w_e + 1$.

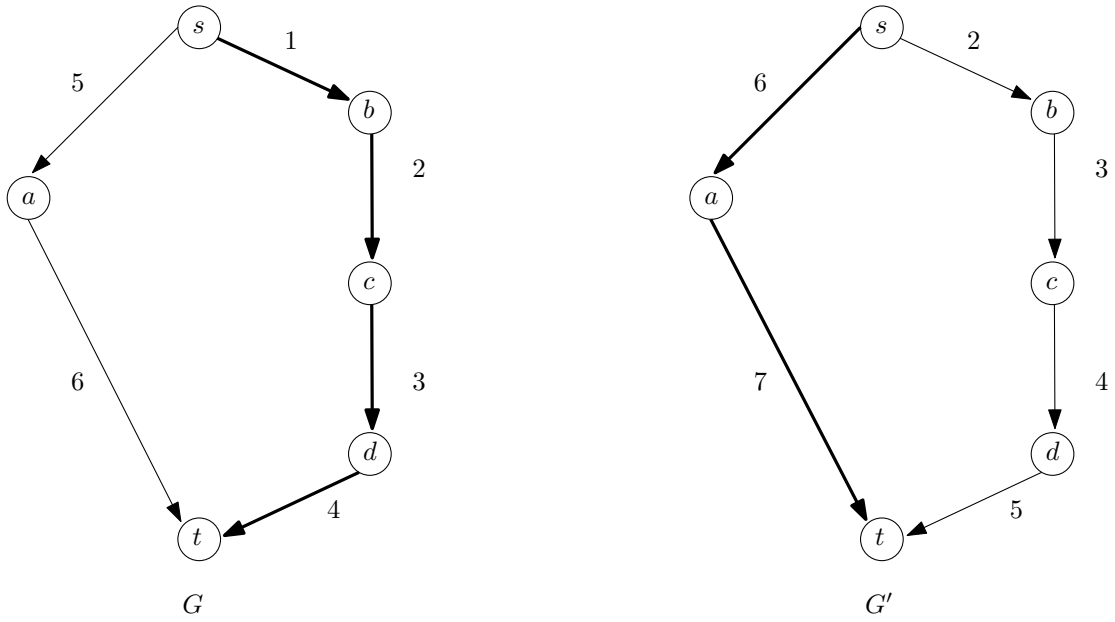
   (a) (10 pts) Does the minimum spanning tree change? Give an example where it changes or prove it cannot change.

   ANSWER: No, the minimum spanning tree does not change. To see this, lets run Kruskal's algorithm to find the MST. Because there are distinct edge weights (and they remain distinct after being increased by 1), there is exactly one MST. Kruskal's algorithm will consider the edges in the same order, because we

increased all of the edge weights by the same amount. We have the same graph structure, so Kruskal's algorithm will add the same edges to the MST. We proved in class that Kruskal's algorithm is correct, so, therefore, the MST cannot change.

(b) (10 pts) Do the shortest paths change? Give an example where they change or prove they cannot change.

ANSWER:



Yes, the shortest paths may change. The length of a path increases by the number of edges on that path so paths with a shorter number of edges may be shortest paths in the modified graph but not the original graph. Look at the path from s to t in the above graphs G and G'. The path $s \to b \to c \to d \to t$ is the original shortest path but the path $s \to a \to t$ becomes the new shortest path when the weight of each edge increases by 1.

3. (10 pts) You are given a strongly connected directed graph $G = (V, E)$ with positive edge weights along with a particular node $v_0 \in V$. Describe an $\mathrm{O}\left((n+m)\lg n\right)$ time algorithm for finding shortest paths between all pairs of nodes, with the one restriction that these paths must all pass through v_0.

ANSWER: The shortest path between two nodes u and v that passes through v_0 must be the shortest path from u to v_0 and the shortest path from v_0 to v. We can easily compute the latter using Dijkstra's algorithm in $\mathrm{O}\left((n+m)\lg n\right)$ time. To compute the former, observe that we want to compute the shortest paths from every vertex to v_0. This is exactly the opposite of the shortest path problem, so we can solve it easily by first reversing the graph and then running Dijkstra's algorithm to compute the shortest paths from v_0 to each vertex in G^R. Reversing the graph takes linear time (as shown in the previous assignment) so this also takes $\mathrm{O}\left((n+m)\lg n\right)$ time. Storing all of the shortest path lengths would require $\mathrm{O}\left(n^2\right)$ time and space, so

we simply keep these two arrays of shortest paths. Using these we can answer a shortest path length query from u to v by adding the shortest path length from u to v_0 (in the second array we created) and the shorteste path length from v_0 to v (using the first array).

4. (10 pts) Here is a problem that occurs in automatic program analysis. For a set of variables $x_1, \ldots, x_n$, you are given some equality constraints, of the form $x_i = x_j$ and some disequality constraints, of the form $x_i \neq x_j$. Is it possible to satisfy all of them? For instance, the constraints $x_1 = x_2, x_2 = x_3, x_3 = x_4, x_1 \neq x_4$ cannot be satisfied.

   (a) Describe an $O(m + n \lg n)$ time algorithm that takes as input m constraints over n variables and decides whether the constraints can be satisfied.

   ANSWER: We can solve this greedily by assigning variables the same value (note that we do not care what the value is) according to the equality constraints and then testing whether any of the disequality constraints can not be satisfied. We assign each variable its own set and then join together sets that are equal. A Union-Find structure allows us to do this easily, similar to how we implemented Kruskal's algorithm.

   Note that we must first check whether two elements of an equality $x_i = x_j$ are already in the same set (i.e. does $\text{Find}(x_i) = \text{Find}(x_j)$?). If they are not, we apply $\text{Union}(x_i, x_j)$. By the same argument we used for Kruskal's algorithm, the at most $n - 1$ union operations take $O(n \lg n)$ time. Scanning through the list of equalities and applying the at most $2m$ Find operations takes $O(m)$ time.

   After each equal variable has been placed in the proper sets, we scan through the list of disequality constraints. The constraints can be satisfied if, and only if, for every constraint $x_i \neq x_j$, we have that $\text{Find}(x_i) \neq \text{Find}(x_j)$, so we check this in linear time. Thus, the total running time is $O(m + n \lg n)$.

   (b) (Bonus: 5 pts) Can you reduce the running time of your algorithm to $O(n + m)$?

   ANSWER: Although the Union-Find structure discussed in class appears to be necessary at first, we do not actually care whether two variables x_i and x_j are in the same set until we have finished processing the equality constraints, unlike Kruskal's algorithm. Instead, we can use a simple undirected graph structure where each variable is a vertex and we add an edge between two variables x_i and x_j for each equality constraint $x_i = x_j$. This takes linear time because we add at most m edges to the graph. We then compute the connected components of the graph in linear time and check each disequality constraint $x_i \neq x_j$ to determine if x_i and x_j are in the same connected component.

   In other words, we apply the same algorithm as in part(a) but we use a graph structure where $\text{Union}(x_i, x_j)$ adds an edge (x_i, x_j) (and we do *not* check whether they are already in the same set) and $\text{Find}(x_i, x_j)$ compares the component numbers of x_i and x_j (which must be computed before we begin checking the disequality constraints). Union and Find operations take constant time and we have a linear number of them. Numbering the connected components also takes linear time so this takes $O(n + m)$ time in total.