CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2021**

Homework #3

April 7, 2021

Name: Ryan Coslove *Extension: Yes*

Homework Policy

- If you leave a question completely blank, you will receive 25% of the grade for that question. This however does not apply to the extra credit questions.
- You are allowed to discuss the homework problems with other students in the class. **But you must write your solutions independently.** You may also consult all the materials used in this course (video recordings, notes, textbook, etc.) while writing your solution, but no other resources are allowed.
- Do not forget to write down your name and whether or not you are using one of your two extensions. Submit your homework on Canvas.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "use Prim's or Kruskal's algorithm to find an MST of the input graph in $O(m \log m)$ time". You are **strongly encouraged to use graph reductions** instead of designing an algorithm from scratch whenever possible (even when the question does not ask you to do so explicitly).
- Remember to always **prove the correctness** of your algorithms and **analyze their running time** (or any other efficiency measure asked in the question).
- The "Challenge yourself" and "Fun with algorithms" are both extra credit. These problems are significantly more challenging than the standard problems you see in this course (including lectures, homeworks, and exams). As a general rule, only attempt to solve these problems if you enjoy them.

Problem 1. You are given the map of n cities with m bidirectional roads between different cities. You are asked to construct airports in some of the cities such that each city either has an airport itself or there is a way to go from this city to a city with an airport using the given roads—moreover, you can also construct a road between any two cities if there is no road between them already. Finally, you are told that the cost of constructing an airport is a and the cost of connecting any two cities by a new road is r.

Design and analyze an $O(n+m)$ time algorithm that given the number n of the cities, the m roads between them, and the costs a and r, outputs the locations of airports and roads to be constructed to satisfy the conditions above, while having the minimum possible cost. **(25 points)**

Examples:

1. *Input:* $n = 4$ cities with $m = 2$ roads $(1, 2), (2, 3)$, cost of constructing an airport $a = 7$, and constructing a road is $r = 5$.

   Output: The minimum cost needed is 12 – we construct an airport in city 4 and connect it this city via a road to any of the cities 1, 2, or 3 (chosen arbitrarily).

2. *Input:* $n = 4$ cities with $m = 2$ roads $(1, 2), (2, 3)$, cost of constructing an airport $a = 7$, and constructing a road is $r = 8$.

   Output: The minimum cost needed is 14 – we construct an airport in city 4 and another one in any one of the cities 1, 2, or 3 (chosen arbitrarily).

Solution. Algorithm

First we will use DFS to find the number of connected cities, and we know that DFS runs an average of O(n+m) time.

. Second, we can arbitrarily place an airport at any city, which will cost a.

If we make the number of connected cities as c, we can use the following as a way to find the minimum: $a + (c-1) * min\{a, r\}$

Proof of correctness

If we use the first example as a base case of $n = 4$ cities with $m = 2$ roads $(1, 2), (2, 3)$, cost of constructing an airport $a = 7$, and constructing a road is $r = 5$, let's check if the formula works.

Given the roads, 2 is connected to 1 and 3 so $c = 2$. Using our equation we have $7 + (2-1) * min\{7, 5\}$. This would equal $7 + 1 * (5) = 12$, which is expected.

If we use the second example given, the equation is $7+(2-1)*min\{7, 8\}$. This is would equal $7+1*(7) = 14$, which is expected.

Runtime

Using DFS to find the number of connected cities will use a run time of $O(n+m)$. Implementing the equation that will tell us the minimum possible cost will take $O(1)$ time.

So, total runtime is $O(n + m)$.

Problem 2. We say that an undirected graph $G = (V, E)$ is **2-edge-connected** if we need to remove *at least two* edges from G to make it disconnected. Prove that a graph $G = (V, E)$ is 2-edge-connected if and only if for every cut $(S, V - S)$ in G, there are *at least two cut edges*, i.e., $|\delta(S)| \geq 2$. **(25 points)**

Solution.

Since the given graph is undirected, the problem can be solved only by counting the number of edges connected to the nodes. If for any of the nodes, the number of edges connected to it is 1 it means on removing this edge the node becomes disconnected and it can't be reached from any other node therefore the graph is not 2-edge connected. Here are the steps:

1. Create an array $noOfEdges[]$ of size V which will store the number of edges connected to a node.
2. For every edge (u, v) increment the number of edges for node u and v.
3. Now iterate over the array $noOfEdges[]$ and check if any of the edge has only 1 edge connected to it. If yes then the graph is not 2-edge connected.
4. Otherwise, the graph is 2-edge connected.

Proof

If our graph does not have any edges with only 1 edge connection, we can confirm that $|\delta(S)| \geq 2$

Any connected graph with at least two vertices can be disconnected by removing edges: by removing all edges incident with a single vertex the graph is disconnected. Thus, $\lambda(G) \leq \sigma(G)$, where $\sigma(G)$ is the minimum degree of any vertex in G. Note that $\sigma(G) \leq n-1$, so $\lambda(G) \leq n-1$.

Removing a vertex also removes all of the edges incident with it, which suggests that $k(G) \leq \lambda(G)$. This turns out to be true. We write $G-v$ to mean G with vertex v removed, and $G-\{v_1, v_2, \ldots, v_k\}$ (k is cutsize) to mean G with all of $\{v_1, v_2, \ldots, v_k\}$ removed, and similarly for edges.

Problem 3. The Muddy City consists of n houses but no proper streets; instead, the different houses are connected to each other via m bidirectional muddy roads. The newly elected mayor of the city aims to pave some of these roads to ease the travel inside the city but also does not want to spend too much money on this project, as paving each road e between houses u and v has a certain cost c_e (different across the muddy roads). The mayor thus specifies two conditions:

- Enough streets must be paved so that everyone can travel from any house to another one using only the paved roads (you may assume that this is always possible);
- The paving should cost as little as possible.

You are chosen to help the mayor in this endeavor.

(a) Design and analyze an $O(m \log m)$ time algorithm for this problem. **(10 points)**

Solution.

We will use Kruskal's algorithm to create a MST constructed where it starts with the lowest cost edge, then continually adding the next lowest cost edge/road until all vertices have been reached.

This is the algorithm:

(a) Sort the edges of G in increasing (non-decreasing) order of their weights.
(b) Let $F = \emptyset$.
(c) For $i = 1$ to m (in the sorted ordering of edges): If adding c_e to F does not create a cycle, let $F \leftarrow F \cup \{c_e\}$.
(d) Return F

Proof

The proof of correctness is done in lecture for Kruskal's algorithm. Consider the forest F maintained by the algorithm. We prove that if F is MST-good at some iteration i and in that iteration we inserted an edge c_e to F, then c_e was safe with respect to F. This then implies that we only added safe edges to F. Moreover, the output of this algorithm isalways a tree since whenever we see an edge that does not create a cycle we add it to F (and since G was connected, we will find a tree eventually this way). That means that we added $n-1$ safe edges. This would imply the correctness of the algorithm as we now can say that this algorithm is an implementation of the meta-algorithm. Since the edges are sorted in non-decreasing order of their weights, we have that c_e has the minimum weight among cut edges of $(S, V-S)$. We are now done since we can apply Theorem 1 and argue that c_e is a safe edge. This finalizes the proof.

Runtime

The first line of the algorithm takes $O(m \log m)$ time to sort all the edges (using, say, merge sort). We then have m iteration, each iteration involve deciding whether adding the edge c_e to the graph creates a cycle or not. The easiest way to implement this step is to run DFS/BFS - that will take $O(n+m)$ time per each iteration, making the total runtime of the algorithm $O(m*(n+m)) = O(mn+m^2) = O(m^2)$ since we know that $m \geq n-1$ as G was connected (any connected graph needs at least $n-1$ edges). Thus, overall, the runtime of the algorithm is $O(m \log m)$.

(b) The mayor of a neighboring city is feeling particularly generous and has made the following offer to Muddy City: they have identified a list of $O(\log m)$ different muddy roads in the city and are willing to entirely pay the cost of paving *exactly one* of them (in exchange for calling the new street after the neighboring city).

Design and analyze an $O(m \log m)$ time algorithm that identifies the paving of which of these roads, if any, the mayor should delegate to the neighboring city to further minimize the total cost—note that if you decide to pave one of the roads payed by the neighboring city, you only need to pay a cost of 1 (for making a plaque of the name of the street). **(15 points)**

Hint: Design an algorithm that given a MST T of a graph G, and a single edge e, in only $O(m)$ time finds an MST T' for the graph G' obtained by changing the weight of the edge e to 1.

Solution.

We will use a similar approach to part (a). We will use Kruskal's algorithm again, and use a sorting algorithm like merge sort to order them in increasing (non-decreasing order).

(a) Sort the edges of G in increasing (non-decreasing) order of their weights.
(b) Let $F = \emptyset$.
(c) For $i = 1$ to m (in the sorted ordering of edges): If adding c_e to F does not create a cycle, let $F \leftarrow F \cup \{c_e\}$.
(d) Return F
(e) The next step to the algorithm is to find the most expensive road c_e. We will iterate through the array of c_e's and find the max value (the last safe edge which will be e).
(f) We create an MST T' for the graph G' by changing the weight/cost of this edge e to 1.

Now, we have our new MST T' for graph G' which has taken the most expensive road e from MST T of graph G and replaced it with a value of 1.

Proof of correctness

The proof for the Kruskal's algorithm and its correctness is done in part (a). In addition to that proof, we know this is correct because we have iterated through the sorted list of c_e values, found the most expensive (maximum value) road and have given it to the neighboring city, thus reducing the cost of G and producing G'. For example if our G contained c_e's of {1, 2, 3, 4, 5} whose total cost is 15, we search for the highest value (5 in this case) and change it to 1 and produce G'. G' will now contain c_e's of {1, 1, 2, 3, 4} whose total cost is 11. With the knowledge that G' always produces a lower total cost than G, we prove that our algorithm works and that we will have the minimum possible cost.

Runtime

The first line of the algorithm takes $O(m \log m)$ time to sort all the edges (using, say, merge sort). We then have m iteration, each iteration involve deciding whether adding the edge c_e to the graph creates a cycle or not. The easiest way to implement this step is to run DFS/BFS - that will take $O(n+m)$ time per each iteration, making the total runtime of the algorithm $O(m*(n+m)) = O(mn+m^2) = O(m^2)$ since we know that $m \geq n-1$ as G was connected (any connected graph needs at least $n-1$ edges). It also only takes $O(m)$ time to find an MST' for graph G'. Thus, overall, the runtime of the algorithm is $O(m \log m)$.

Problem 4. You are given a weighted undirected graph $G = (V, E)$ with integer weights $w_e \in \{1, 2, \ldots, W\}$ on each edge e, where $W = O(1)$. Given two vertices $s, t \in V$, the goal is to find the minimum weight path (or shortest path) from s to t. Recall that Dijkstra's algorithm solves this problem in $O(n + m \log m)$ time even if we do not have the condition that $W = O(1)$. However, we now want to use this extra condition to design an even faster algorithm.

Design and analyze an algorithm to find the minimum weight (shortest) s-t path in $O(n + m)$ time.

Solution.

We will utilize BFS for this problem. This is the algorithm:

1. Initialize an array $mark[1 : t]$ with 'FALSE' and s be the designated source vertex.
2. Create a queue data structure, set $mark[s] =' TRUE'$ and let S (initially) be the set of edges incident on s and assign a value $value(e) = d[s] + w_e$ to each of these edges.
3. While S is not empty:
   - Let $e = (u, v)$ be the minimum value edge in S and remove e from S.
   - If $mark[v] = 'TRUE'$; continue to the next iteration of the while-loop.
   - Otherwise, let $mark[v] = 'TRUE'$ and $d[v] = value(e)$, and insert all edges e' incident on v to S with $value(e') = d[v] + w'_e$.
4. Return d

Proof

The proof of BFS is done in lecture. The alteration to this from our normal BFS is we are queueing the integer weights, iterating through the minimum weights in the queue.

Runtime

Because we are calling this algorithm on disconnected sub graphs, doing each one once, and knowing the general time complexity of running a BFS, we can assure that to find the minimum weight (shortest) $s - t$ path is in $O(n + m)$ time.

Challenge Yourself. A **bottleneck spanning tree (BST)** of a undirected connected graph $G = (V, E)$ with positive weights w_e over each edge e, is a spanning tree T of G such that the weight of maximum-weight edge in T is minimized across all spanning trees of G. In other words, if we define the cost of T as $\max_{e \in T} w_e$, a BST has minimum cost across all spanning trees of G. Design and analyze an $O(n + m)$ time algorithm for finding a BST of a given graph. **(+10 points)**

Fun with Algorithms. Let us go back to the bottleneck spanning tree (BST) problem defined above.

(a) Prove that any MST of any graph G is also a BST. Use this to obtain an $O(m \log m)$ time algorithm for the BST problem (notice that this is slower than the algorithm from the previous question). **(+8 points)**

(b) Give an example of a BST of some graph G which is *not* an MST in G. **(+2 points)**