CS 170 HW 4 (Optional; 0 Credits)

Due on 2018-02-18, at 10:00 pm

1 Study Group

List the names and SIDs of the members in your study group.

2 All Roads Lead to Rome

You are the chief trade minister under Emperor Caesar Augustus with the job of directing trade in the ancient world. The Emperor has proclaimed that *all roads lead to (and from) Rome*; that is, all trade must go through Rome. In particular, you are given a strongly connected directed graph $G = (V, E)$ with positive edge weights, and there is a particular node $v_0 \in V$ (Rome). Give an efficient algorithm for finding shortest path between *all pairs of nodes*, with the one restriction that these paths must all pass through v_0 (Rome). Make your algorithm as efficient as you can, perhaps as fast as Dijkstra's algorithm.

(a) Give the efficient algorithm.

[Provide 3 part solution.]

(b) Occasionally, Augustus will ask you for the (smallest) distance between two vertices. You want to do this as quickly as possible, so that Augustus does not have your head.

This is called a *distance query*: Given a pair of vertices (u, v), give the the distance of the shortest path that passes through v_0. Describe how you might store the results such that you require $O(|V|)$ storage and from your data structure you can compute the result in $O(1)$ time. For your answer, a clear description of the data structure and its usage is sufficient.

(c) On the other hand, the traders need to know the paths themselves.

This is called a *path query*: Given a part of vertices (u, v), give the shortest path itself, that passes through v_0. Describe how you might store the results such that you require $O(|V|)$ storage and from your data structure you can compute the result in $O(|V|)$ time. Again, a clear description of the data structure and its usage is sufficient.

3 Fixing Dijsktra's Algorithm with Negative Weights

Dijkstra's algorithm doesn't work on graphs with negative edge weights. Here is one attempt to fix it:

1. Add a large number M to every edge so that there are no negative weights left.
2. Run Dijkstra to find the shortest path in the new graph.
3. Return the path Dijkstra found, but with the old edge weights (i.e. subtract M from the weight of each edge).

Show that this algorithm doesn't work by finding a graph for which it must give the wrong answer.

4 Connectivity vs Strong Connectivity

(a) Prove that in any connected undirected graph $G = (V, E)$ there is a vertex $v \in V$ such that removing v from G gives another connected graph.

(b) Give an example of a strongly connected directed graph $G = (V, E)$ such that, for *every* $v \in V$, removing v from G gives a directed graph that is not strongly connected.

(c) Let $G = (V, E)$ be a connected undirected graph such that G remains connected after removing any vertex. Show that for every pair of vertices u, v where $(u, v) \notin E$ there exist two different u-v paths.

5 Disrupting a Network of Spies

Let $G = (V, E)$ denote the "social network" of a group of spies. In other words, G is an undirected graph where each vertex $v \in V$ corresponds to a spy, and we introduce the edge $\{u, v\}$ if spies u and v have had contact with each other. The police would like to determine which spy they should try to capture, to disrupt the coordination of the group of spies as much as possible. More precisely, the goal is to find a single vertex $v \in V$ whose removal from the graph splits the graph into as many different connected components as possible. This problem will walk you through the design of a linear-time algorithm to solve this problem. In other words, the running time will be $O(|V| + |E|)$.

In the following, let $f(v)$ denote the number of connected components in the graph obtained after deleting vertex v from G. Also, assume that initial graph G is connected (before any vertex is deleted), has at least two vertices, and is represented in adjacency list format.

For each part, prove that your answer is correct (some parts are simple enough that the proof can be a brief justification; others will be more involved).

(a) Let T be a tree produced by running DFS on G with root $r \in V$. (In particular, $T = (V, E_T)$ is a spanning tree of G.) Given T, find an efficient way to calculate $f(r)$.

(b) Let $v \in V$ be some vertex that is not the root of T (i.e., $v \neq r$). Suppose further that no descendant of v in T has any non-tree edge (i.e. edge in $E \setminus E_T$) to any ancestor of v in T. How could you calculate $f(v)$ from T in an efficient way?

(c) For $w \in V$, let $D_T(w)$ be the set of descendants of w in T including w itself. For a set $S \subseteq V$, let $N_G(S)$ be the set of *neighbours* of S in G, i.e. $N_G(S) = \{y \in V : \exists x \in S \text{ s.t. } \{x, y\} \in E\}$. We define $\mathsf{up}_T(w) := \min_{y \in N_G(D_T(w))} \mathsf{depth}_T(y)$, i.e. the smallest depth in T of any neighbour in G of any descendant of w in T.

Now suppose v is an arbitrary non-root node in T, with children $w_1, \ldots, w_k$. Describe how to compute $f(v)$ as a function of k, $\mathsf{up}_T(w_1), \ldots, \mathsf{up}_T(w_k)$, and $\mathsf{depth}_T(v)$.

Hint: Think about what happened in part (b); think about what changes when we can have non-tree edges that go up from one of v's descendants to one of v's ancestors; and think about how you can detect it from the information provided.

(d) Design an algorithm which, on input G, T, computes $\mathtt{up}_T(v)$ for all vertices $v \in V$, in linear time.

(e) Given G, describe how to compute $f(v)$ for all vertices $v \in V$, in linear time.

6 Unique Shortest Path

Shortest paths are not always unique: sometimes there are two or more different paths with the minimum possible length. Show how to solve the following problem in $O((|V| + |E|) \log |V|)$ time.

Input: An undirected graph $G = (V, E)$; edge lengths $l_e > 0$; starting vertex $s \in V$.

Output: A Boolean array `usp[·]`: for each node u, the entry `usp[`u`]` should be `true` if and only if there is a *unique* shortest path from s to u. (Note: `usp[`s`] = true`.)

[Provide 3 part solution.]