CS5050 Advanced Algorithms

Fall Semester, 2015

Assignment 6: Graph Algorithms I

Due Date: 8:30 a.m., Monday, Nov. 23, 2015 (**at the beginning of CS5050 class**)

Note: In this assignment, we assume all input graphs are represented by adjacency lists.

1. We say that a **directed** graph G is *strongly connected* if for any two vertices u and v, there exists a path from u to v and there also exists a path from v to u in G.

   Given a directed graph $G = (V, E)$, let $n = |V|$ and $m = |E|$. Let s be an arbitrary vertex of G.

   (a) Design an $O(m + n)$ time algorithm to determine whether there exists a path from s to every other vertex of G. **(10 points)**

   (b) Design an $O(m+n)$ time algorithm to determine whether there exists a path from every other vertex of G to s. **(10 points)**

   (c) Prove the following statement: *G is strongly connected if and only if there is a path in G from s to every other vertex and there is a path from every other vertex to s.* **(10 points)**

   Remark: According to the above statement, we can determine whether G is strongly connected in $O(m + n)$ time by using your algorithms for the above two questions (a) and (b).

2. Given a graph $G = (V, E)$ (either directed or undirected), let $n = |V|$ and $m = |E|$. In class we gave a depth-first-search (DFS) algorithm that is recursive and runs in $O(m + n)$ time. Usually a recursive algorithm can be replaced by a non-recursive algorithm by using a stack.

   Design a *non-recursive* DFS algorithm by using a stack, and your algorithm should also run in $O(m + n)$ time. As we did in class, your algorithm should compute the predecessor $v.pre$ for each vertex v of the graph, i.e., if v is visited by the algorithm through the edge (u, v), then $v.pre = u$. As discussed in class, the predecessor information essentially maintains the DFS tree. **(20 points)**

3. Given an undirected graph $G = (V, E)$, let $n = |V|$ and $m = |E|$. Suppose s and t are two vertices in G. We have already known that we can find a shortest path from s to t by using the breadth-first-search (BFS) algorithm. However, there might be multiple different shortest paths from s to t (e.g., see Fig. 1 as an example).

   Design an $O(m + n)$ time algorithm to compute the number of different shortest paths from s to t (your algorithm does not need to list all the paths, but only report the number of different paths). (Hint: Modify the BFS algorithm.) **(20 points)**

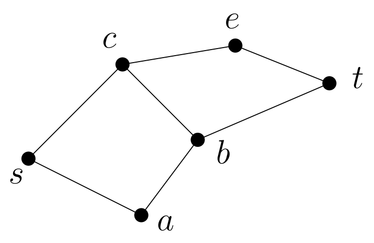

Figure 1: There are three different shortest paths from s to t: $(s, a, b, t), (s, c, b, t), (s, c, e, t)$.

Note: In case you are interested, the following is an application of the above problem in social networks.

A number of art museums around the country have been featuring work by an artist named Mark Lombardi (1951–2000), consisting of a set of intricately rendered graphs. Building on a great deal of research, these graphs encode the relationships among people involved in major political scandals over the past several decades: the vertices correspond to participants, and each edge indicates some type of relationship between a pair of participants. And so, if you peer closely enough at the drawings, you can trace out ominous-looking paths from a high-ranking U.S. government official, to a former business partner, to a bank in Switzerland, to a shadowy arms dealer.

Such pictures form striking examples of *social networks*, which have vertices representing people and organizations, and edges representing relationships of various kinds. And the short paths that abound in these networks have attracted considerable attention recently, as people ponder what they mean. In the case of Mark Lombardis graphs, they hint at the short set of steps that can carry you from the reputable to the disreputable.

Of course, a single, spurious short path between vertices s and t in such a network may be more coincidental than anything else; a large number of short paths between s and t can be much more convincing. So in addition to the problem of computing a single shortest s-t path in a graph G, social networks researchers have looked at the problem of determining the number of shortest s-t paths, which is the problem we are now considering.

4. Given a directed-acyclic-graph (DAG) $G = (V, E)$, let $n = |V|$ and $m = |E|$. Let s and t be two vertices of G. There might be multiple different paths (not necessarily shortest paths) from s to t (e.g., see Fig. 2 for an example). Design an $O(m + n)$ time algorithm to compute the number of different paths in G from s to t. **(20 points)**

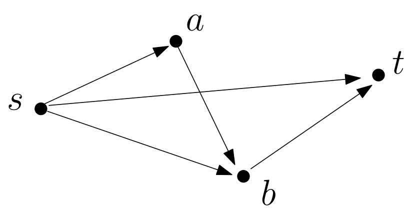

Figure 2: There are three different paths from s to t: $s \to t$, $s \to b \to t$, and $s \to a \to b \to t$.

Total Points: 90