CS5050 Advanced Algorithms

Fall Semester, 2017

Assignment 6: Graph Algorithms I

Due Date: 8:30 a.m., Monday, Nov. 27, 2017 (**at the beginning of CS5050 class**)

Note: In this assignment, we assume that 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 a vertex of G.

   (a) Design an $O(m+n)$ time algorithm to determine whether the following is true: for each vertex v of G, there exists a path from s to v. **(10 points)**

   (b) Design an $O(m+n)$ time algorithm to determine whether the following is true: for each vertex v of G, there exists a path from v to s. **(10 points)**

   (c) Prove the following statement: *G is strongly connected if and only if for each vertex v of G, there is a path in G from s to v and there is a path from v 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 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 shortest paths, but only report their number). (Hint: Modify the BFS algorithm.) **(20 points)**

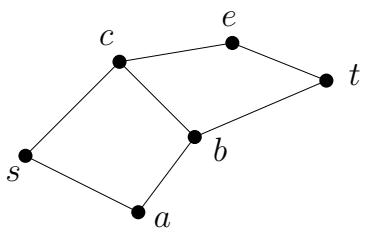

Figure 1: There are three different shortest paths from s to t: $(s, a, b, t), (s, c, b, t), (s, c, e, t)$.

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 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.

3. 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)**

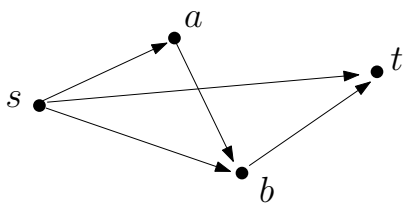

Figure 2: There are three different paths from s to t: $s \to t$, $s \to b \to t$, and $s \to a \to b \to t$.

4. In class we have studied an algorithm for computing shortest paths in DAGs. Particularly, the *length* of a path is defined to be the total sum of the weights of all edges in the path.

   Sometimes we are not only interested in a shortest path but also care about the number of edges of the path. For example, consider a flight graph in which each vertex is an airport and each edge represents a flight segment from one airport to another, and the weight of every edge represents the price of that flight segment. If you want to fly from airport s to another one t, you are certainly interested in a shortest path from s to t to minimize the total price you have to pay. But you probably do not want a path with too many stops (i.e., too many edges) even if it is relatively cheap. For example, suppose you can only accept a path with at most two stops (i.e., three edges). Then, your goal is essentially to find a shortest path from

s to t such that the path has at most three edges. This is the problem we are considering in this exercise.

Let $G = (V, E)$ be a DAG (directed-acyclic-graph) of n vertices and m edges. Each edge (u, v) of E has a weight $w(u, v)$ (which can be positive, zero, or negative). Let k be a positive integer, which is given in the input. A path in G is called a *k-link* path if the path has **no more than** k edges. Let s and t be two vertices of G. A *k-link shortest path* from s to t is defined as a k-link path from s to t that has the minimum total sum of edge weights among all possible k-link s-to-t paths in G. Refer to Fig. 3 for an example.

Design an $O(k(m + n))$ time algorithm to compute a k-link shortest path from s to t. For simplicity, you only need to return the length of the path and do not need to report the actual path. (Hint: use dynamic programming or modify the shortest path algorithm on DAG we discussed in class.) **(20 points)**

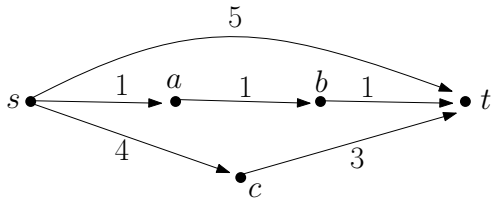

Figure 3: Illustrating a DAG: the number besides each edge is the weight of the edge. If we are looking for a 2-link (i.e., $k = 2$) shortest path from s to t, then it is the path consisting of the only edge (s, t), whose length is 5. Although the path $s \to a \to b \to t$ is shorter, it is not a 2-link path since it has three edges. Note that the path with the only edge (s, t) is a 2-link path because it has one edge, which is indeed **no more than** two edges.

Total Points: 90