# CS5050 Advanced Algorithms

Fall 2023

## Assignment 6: Algorithm Analysis

Due Date: 11:59:59 p.m., *Sunday, Dec 3, 2023*

**Total Points: 70**

**Note:** In this assignment, we assume that all input graphs are represented by **adjacency lists**. If you want to start early, you should be able to do Problems 1 and 2 now. For Problems 3 and 4, you may want to wait until after Thursday's class on Nov 30.

1. Given a **directed** graph $G$ of $n$ vertices and $m$ edges, let $s$ be a vertex of $G$.

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

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

   **Note:** The input is the adjacency lists for $G$. This means that all information needed in your algorithm must be computed from the adjacency lists of $G$. For example, if you want to convert $G$ to a new graph $G'$, then you must compute the adjacency lists of $G'$ using the adjacency lists of $G$.

   **Note:** Here is an application of your algorithms for (a) and (b). We say that a directed graph $G$ is *strongly connected* if for every pair of vertices $u$ and $v$, there exists a path from $u$ to $v$ and there also exists a path from $v$ to $u$ in $G$. An interesting observation is that *$G$ is strongly connected if and only if there exists a path in $G$ from $s$ to $v$ and there is also a path from $v$ to $s$ for every vertex $v$ of $G$* (you may think about how to prove this observation). In light of the observation, 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. **(20 points)** Given a **directed-acyclic-graph (DAG)** $G$ of $n$ vertices and $m$ edges, 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. 1 for an example). Design an $O(m+n)$ time algorithm to compute the number of different paths in $G$ from $s$ to $t$.

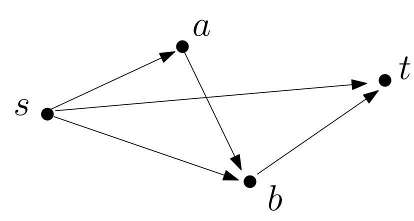


Figure 1: There are three different paths from $s$ to $t$: $s \to t$, $s \to b \to t$, and $s \to a \to b \to t$.

3. **(20 points)** Given a directed graph $G$ of $n$ vertices and $m$ edges, each edge $(u, v)$ has a weight $w(u, v)$, which can be positive, zero, or negative. The *bottleneck-weight* of any path in $G$ is defined to be the **largest** weight of all edges in the path. Let $s$ and $t$ be two vertices of $G$. A *minimum bottleneck-weight path* from $s$ to $t$ is a path with the smallest bottleneck-weight among all paths from $s$ to $t$ in $G$. Refer to Figure 2 for an example.

Modify Dijkstra's algorithm to compute minimum bottleneck-weight paths from $s$ to all other vertices of $G$. Your algorithm does not have to output all paths but only need to compute the correct predecessor information for all vertices (which forms a *minimum bottleneck-weight path tree* with $s$ as the root), as we did in class for Dijkstra's algorithm. Your algorithm is required to have the same time complexity as Dijkstra's algorithm, i.e., $O((n+m)\log n)$ time.

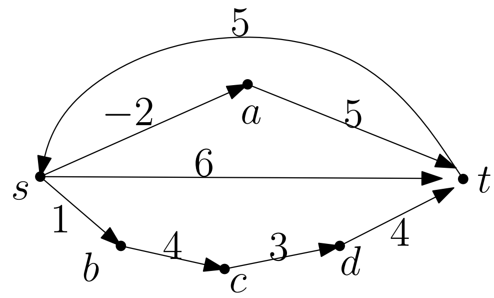


Figure 2: Here is a minimum bottleneck-weight path from $s$ to $t$: $s, b, c, d, t$, whose bottleneck-weight is 4.

4. **(10 points)** Let $G$ be an undirected connected graph of $n$ vertices and $m$ edges. Suppose each edge of $G$ has a color of either *blue* or *red*. Design an algorithm to find a spanning tree $T$ of $G$ such that $T$ has as few red edges as possible. Your algorithm should run in $O((n+m)\log n)$ time.