

CS/ECE 374 Sec A ✦ Spring 2025

🌀 Homework 9 🌀

Due Wednesday, April 9th, 2025 at 9am

- You can work in a group of up to **three** students. Read the instructions on the course website for additional details.
 - **Submit your solutions electronically on the course Gradescope site as PDF files.** Submit a separate PDF file for each numbered problem. If you plan to typeset your solutions, please use the \LaTeX solution template on the course web site. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app (or an actual scanner, not just a phone camera).
 - Late submissions will be accepted (for 75% credit) until midnight the day of the deadline.
-

🔔 **Some important course policies** 🔔

- **You may use any source at your disposal**—paper, electronic, or human—but you *must* cite *every* source that you use, and you *must* write everything yourself in your own words. See the academic integrity policies on the course web site for more details including the policy on using AI tools.
 - **Avoid the Two Deadly Sins!** Any homework or exam solution that breaks any of the following rules will be given an **automatic zero**, unless the solution is otherwise perfect. Yes, we really mean it. We're not trying to be scary or petty (Honest!), but we do want to break a few common bad habits that seriously impede mastery of the course material.
 - Always give complete solutions, not just examples.
 - Always declare all your variables, in English. In particular, always describe the specific problem your algorithm is supposed to solve.
-

See the course web site for more information.

If you have any questions about these policies, please don't hesitate to ask in class, in office hours, or on Ed.

1. Several problems can be solved on directed graphs by first solving the corresponding problem on a DAG, and then reducing the problem on a general graph to that on a DAG via the meta-graph. On a DAG one can use topological sort or some other useful property. Solve each of the following problems using this general paradigm.
 - (a) (3 pts) Let $G = (V, E)$ be a directed graph such that each vertex v has a number $b(v)$ associated with it. The goal is to compute for each $u \in V$ the number $a(u) = \min\{b(v) \mid v \text{ can reach } u \text{ in } G\}$. Describe a linear time algorithm that given G and b values in an array B outputs $a(u)$ for each $u \in V$.
 - (b) (3 pts) Let $G = (V, E)$ be a directed graph. We have seen a linear-time algorithm that checks whether G is strongly connected. Suppose G is not strongly connected. Describe a linear time algorithm to check if one can add at most *one* edge to G to make it strongly connected.
 - (c) (4 pts) Let $G = (V, E)$ be a directed graph in which each edge $e \in E$ has a non-negative reward $p(e)$ that can be collected by traversing it. Describe a linear time algorithm that given G , the edge rewards, and a starting node s , computes the maximum reward that a walk starting at s can collect. Note the the reward on an edge e is counted only the first time it is traversed in the walk since it is gone after it is picked up.

2. Let $G = (V, E)$ be a directed graph with non-negative edge lengths $\ell(e), e \in E$. Dijkstra's algorithm can be used to find the shortest path tree rooted at any given node $s \in V$. In the standard shortest path problem the length of a path v_1, v_2, \dots, v_h is defined as $\sum_{i=1}^{h-1} \ell(v_i, v_{i+1})$ which is simply the sum of the lengths of the edges in the path. In various situations one needs different measures.
 - (a) (2.5 pts) For a parameter $k \geq 1$ the k -norm length of a path v_1, v_2, \dots, v_k is defined to be $(\sum_{i=1}^{h-1} \ell(v_i, v_{i+1})^k)^{1/k}$. If $k = 1$ we get the standard length. Given $G, s, t \in V$ and an integer $k \geq 1$ describe an algorithm to find the shortest k -norm length path from s to t in G . Give a small example where 2-norm s - t shortest path is different from the standard shortest path.
 - (b) (2.5 pts) Consider the previous part but now suppose we set k to be a very large number. As $k \rightarrow \infty$ the k -norm of a path can be seen to be the maximum length of the edges in the path (assume that edge lengths are distinct). This corresponds to the ∞ norm of a vector which is the largest coordinate. In the context of paths, the length of the longest edge length in a path is called its *bottleneck* length. Describe an algorithm to compute the bottleneck shortest path distances from s to every node in G by adapting Dijkstra's algorithm.
 - (c) (2.5 pts) We will consider an alternate algorithm to compute the s - t bottleneck shortest path distance. Given G, s, t and a value $\lambda \geq 0$, describe a reduction to s - t reachability to decide whether there is a path from s - t with bottleneck length at most λ . Use this and binary search to find the s - t bottleneck distance.
 - (d) (2.5 pts) Now consider another motivaton. Suppose each edge $e \in E$ has a probability $p(e)$ of failing. Given a path v_1, v_2, \dots, v_h , what is the probability that none of the edges in the path fail assuming that the edges fail independently? Describe an algorithm to find the s - t path with the least probability of failing.

3. **Not to submit:** Let $G = (V, E)$ a directed graph with non-negative edge lengths. Let $R \subset E$ and $B \subset E$ be red and blue edges (the rest are not colored). Given s, t and integers h_r and h_b describe an efficient algorithm to find the length of a shortest s - t path that contains at most h_r red edges and at most h_b blue edges.

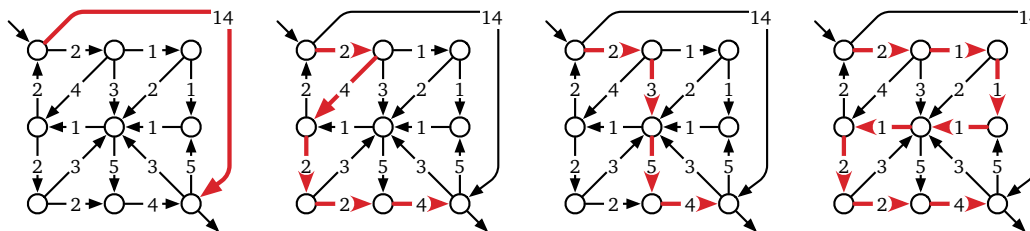
4. **Not to submit:**

- (a) Aaron and Alex decided to host a party for the CAs at Aaron's place. Yulie knew that Aaron likes empanadas and decided to buy them at one of the many places that sell them in Champaign (lucky for us!) on her way from her place to Aaron's place. Let $G = (V, E)$ be a directed graph with non-negative edge lengths $\ell(e), e \in E$ that represents the roads in town. Yulie entered her car to drive over and noticed that she may not have sufficient gas to drive all the way to Aaron's place. She estimated that her car can go D miles before her gas runs out. She wants to get to Aaron's place as fast as she can. Describe an efficient algorithm to help her accomplish the task of reaching Aaron's place with as little travel as feasible; she needs to buy empanadas but may or may not need to fill gas. Assume Yulie's house is at node s and Aaron's place is at node t and that the empanada shops are given by a set $X \subset V$ and the gas stations by a set $Y \subset V$. Assume that X, Y are disjoint sets. Also assume, for simplicity, that once Yulie fills gas she can travel an infinite distance. Note that Yulie could buy empanadas before or after filling up gas, as long as she does not run out of fuel on the way to the gas station. Express your running time as a function of n , the number of nodes, and m , the number of edges.
- (b) In the previous part we assumed that Yulie's car can go an infinite distance after filling up on gas. This is reasonable assumption for visits in the town. However, Yulie is now driving to visit a friend in a different state and has to fill up gas multiple times before reaching her destination. Assume that each full tank of gas lets her drive R miles. Describe an efficient algorithm that minimizes her driving distance. For simplicity assume that she starts with a full tank of gas.

For both parts your algorithm should return ∞ if there is no way to reach the destination given the graph and the constraints.

Solved Problem

- Although we typically speak of “the” shortest path between two nodes, a single graph could contain several minimum-length paths with the same endpoints.



Four (of many) equal-length shortest paths.

Describe and analyze an algorithm to determine the *number* of shortest paths from a source vertex s to a target vertex t in an arbitrary directed graph G with weighted edges. You may assume that all edge weights are positive and that all necessary arithmetic operations can be performed in $O(1)$ time.

[Hint: Compute shortest path distances from s to every other vertex. Throw away all edges that cannot be part of a shortest path from s to another vertex. What’s left?]

Solution: We start by computing shortest-path distances $dist(v)$ from s to v , for every vertex v , using Dijkstra’s algorithm. Call an edge $u \rightarrow v$ **tight** if $dist(u) + w(u \rightarrow v) = dist(v)$. Every edge in a shortest path from s to t must be tight. Conversely, every path from s to t that uses only tight edges has total length $dist(t)$ and is therefore a shortest path!

Let H be the subgraph of all tight edges in G . We can easily construct H in $O(V + E)$ time. Because all edge weights are positive, H is a directed acyclic graph. It remains only to count the number of paths from s to t in H .

For any vertex v , let $PathsToT(v)$ denote the number of paths in H from v to t ; we need to compute $PathsToT(s)$. This function satisfies the following simple recurrence:

$$PathsToT(v) = \begin{cases} 1 & \text{if } v = t \\ \sum_{v \rightarrow w} PathsToT(w) & \text{otherwise} \end{cases}$$

In particular, if v is a sink but $v \neq t$ (and thus there are no paths from v to t), this recurrence correctly gives us $PathsToT(v) = \sum \emptyset = 0$.

We can memoize this function into the graph itself, storing each value $PathsToT(v)$ at the corresponding vertex v . Since each subproblem depends only on its successors in H , we can compute $PathsToT(v)$ for all vertices v by considering the vertices in reverse topological order, or equivalently, by performing a depth-first search of H starting at s . The resulting algorithm runs in $O(V + E)$ time.

The overall running time of the algorithm is dominated by Dijkstra’s algorithm in the preprocessing phase, which runs in $O(E \log V)$ time. ■

Rubric: 10 points = 5 points for reduction to counting paths in a dag + 5 points for the path-counting algorithm (standard dynamic programming rubric)