CS/ECE 374 ✧ Spring 2021

Homework 8

Due Thursday, April 8, 2021 at 10am

Groups of up to three people can submit joint solutions. Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

The following unnumbered problems are not for submission or grading. No solutions for them will be provided but you can discuss them on Piazza.

- In the lab you saw how to compute s-t shortest walks efficiently when the graph has a single negative length edge. The running time is asymptotically the same as using Dijkstra's algorithm. Generalize this to the setting where the graph has *two* negative length edges. Can you generalize the approach to any fixed number of edges? If there are k negative length edges what would be the running time of your algorithm? Is this better than using Bellman-Ford?
- See problems in HW 8 from Spring 2018. https://courses.engr.illinois.edu/cs374/sp2018/A/homework/hw8.pdf

1. Let $G = (V, E)$ be a directed graph with non-negative edge lengths $\ell(e), e \in E$ that represents a road network. Alice is invited to a party at her friend Bob's house and she wants to buy dessert at a grocery store on the drive to his house. Just as she is about to leave she realizes that she may not have sufficient gas in her car to drive all the way to Bob's house. Her car can go at most a distance of R before the gas runs out. Describe an efficient algorithm to help Alice accomplish the task of reaching Bob's house with as little travel as feasible; she needs to buy dessert but may or may not need to fill gap. Assume Alice's house is at node s and Bob's house is at node t and that the grocery 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 Alice fills gas she can travel an infinite distance. Note that Alice could buy dessert either 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.

   Try not to modify the internals of existing algorithms. Use them on modified graphs or multiple times on different instances to deduce the information needed for your problem. You may want to first find a simple algorithm whose correctness that you are confident about and then try to improve the running time. You will lose at most 2 out of 10 for a sub-optimal running time so do not make that your priority.

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 $v_1, v_2, \ldots, v_h$ 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.

- For a parameter $k \geq 1$ the k-norm length of a path $v_1, v_2, \ldots, v_k$ 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 $k \geq 1$ 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.
- Consider the previous part but now suppose we set k to be a very large number. As $k \to \infty$ 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.
- **Not to submit:** 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 reachbility 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.
- **Not to submit:** Now consider another motivaton. Suppose each edge $e \in E$ has a probability $p(e)$ of failing. Given a path $v_1, v_2, \ldots, v_h$, 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.

No proofs necessary but the algorithm should be clear.

3. Since you are taking an algorithms class you decided to create a fun candy hunting game for Halloween. You set up a maze with one way streets that can be thought of as a directed graph $G = (V, E)$. Each node v in the maze has $w(v)$ amount of candy located at v.

   - Each of your friends, starting at a given node s, has to figure out the maximum amount of candy they can collect. Note that candy at node v can be collected only once even if the node v is visited again on the way to some other place.
   - Your friends complain that they can collect more candy if they get to choose the starting node. You agree to their their request and ask them to maximize the amount of candy they can collect starting at any node they choose.

   Before you ask your friends to solve the game you need to know how to do it yourself! Describe efficient algorithms for both variants. Ideally your algorithm should run in linear time. *Hint:* Consider what happens if G is strongly connected and if it is a DAG.

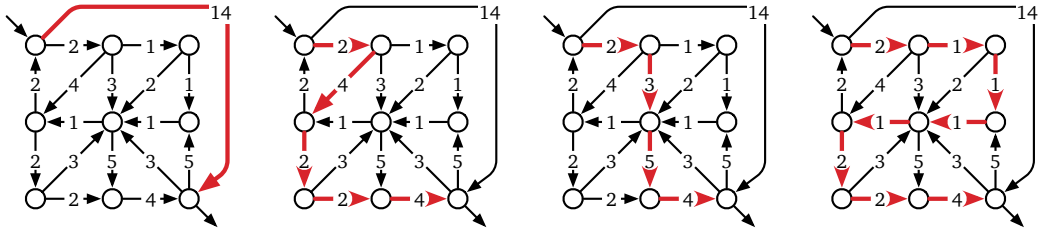
   No proof necessary if you use reductions to standard algorithms via graph transformations and simple steps. Otherwise you need to prove the correctness.

4. **Not to submit but strongly encouraged to solve:** 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.

5. **Not to submit but strongly encouraged to solve:** Read the notes to see the connection between DP and DAGs. Solve the McKing problem from HW 7 via a reduction to a shortest path problem on DAGs without invoking DP.

Solved Problem

4. 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 $u \to v$ ***tight*** if $dist(u) + w(u \to v) = dist(v)$. 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:

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

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 $PathsToT(v) = \sum \varnothing = 0$.

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 $\boldsymbol{O(E \log V)}$ ***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)