CS/ECE 374 A 💀 Fall 2021

🦇🦇 Homework 9 🦇🦇

Due Tuesday, November 2, 2021 at 8pm Central Time

This is the last homework before Midterm 2.

1. Morty needs to retrieve a stabilized plumbus from the Clackspire Labyrinth. He must enter the labyrinth using Rick's interdimensional portal gun, traverse the Labyrinth to a plumbus, then take that plumbus through the Labyrinth to a fleeb to be stabilized, and finally take the stabilized plumbus back to the original portal to return home. Plumbuses are stabilized by fleeb juice, which any fleeb will release immediately after being removed from its fleebhole. An unstabilized plumbus will explode if it is carried more than 137 flinks from its original storage unit. The Clackspire Labyrinth smells like farts, so Morty wants to spend as little time there as possible.

   Rick has given Morty a detailed map of the Clackspire Labyrinth, which consist of a directed graph $G = (V, E)$ with non-negative edge weights (indicating distance in flinks), along with two disjoint subsets $P \subset V$ and $F \subset V$, indicating the plumbus storage units and fleebholes, respectively. Morty needs to identify a start vertex s, a plumbus storage unit $p \in P$, and a fleebhole $f \in F$, such that the shortest-path distance from p to f is at most 137 flinks long, and the length of the shortest walk $s \rightsquigarrow p \rightsquigarrow f \rightsquigarrow s$ is as short as possible.

   Describe and analyze an algo(burp)rithm to so(burp)olve Morty's problem. You can assume that it is in fact possible for Morty to succeed. As usual, do not assume that edge weights are integers.

2. You are planning a hiking trip in Jasper National Park in British Columbia over winter break. You have a complete map of the park's trails, which indicates that hikers on certain trails have a higher chance of encountering a sasquatch. All visitors to the park are required to purchase a canister of sasquatch repellent. You can safely traverse a high-risk trail segment only by *completely* using up a *full* canister of sasquatch repellent. The park rangers have helpfully installed several refilling stations around the park, where you can refill empty canisters at no cost. The canisters themselves are expensive and heavy, so you can only carry one. The trails are narrow, so each trail segment allows traffic in only one direction.

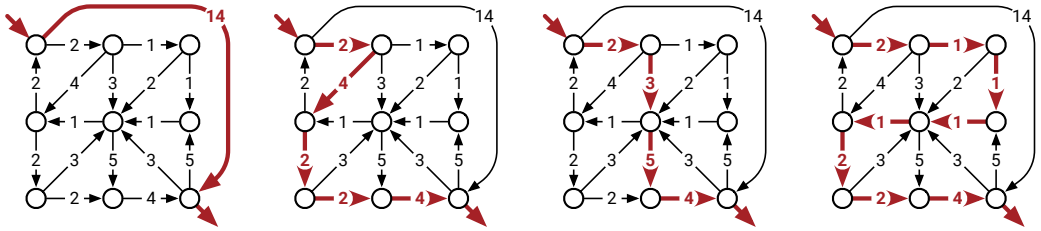
   You have converted the trail map into a directed graph $G = (V, E)$, whose vertices represent trail intersections, and whose edges represent trail segments. A subset $R \subseteq V$ of the vertices indicate the locations of the *R*epellent *R*efilling stations, and a subset $H \subseteq E$ of the edges are marked as *H*igh-risk. Each edge e is labeled with the length $\ell(e)$ of the corresponding trail segment. Your campsite appears on the map as a particular vertex $s \in V$, and the visitor center is another vertex $t \in V$.

   (a) Describe and analyze an algorithm that finds the shortest *safe* hike from your campsite s to the visitor center t. Assume there is a refill station at your campsite, and another refill station at the visitor center.

   (b) Describe and analyze an algorithm to decide if you can safely hike from any refill station any other refill station. In other words, for *every* pair of vertices u and v in R, is there a safe hike from u to v?

Solved Problem

3. Although we typically speak of "the" shortest path from one vertex to another, 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 compute the *number* of shortest paths from a source vertex s to a target vertex t in an arbitrary directed graph G with weighted edges. Assume that all edge weights are positive and that any necessary arithmetic operations can be performed in $O(1)$ time each.

[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 $NumPaths(v)$ denote the number of paths in H from v to t; we need to compute $NumPaths(s)$. This function satisfies the following simple recurrence:

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

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

We can memoize this function into the graph itself, storing each value $NumPaths(v)$ at the corresponding vertex v. Since each subproblem depends only on its successors in H, we can compute $NumPaths(v)$ 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 (standard graph reduction rubric) + 5 points for the path-counting algorithm (standard dynamic programming rubric)