CS/ECE 374 A ✧ Spring 2018

Homework 8

Due Tuesday, April 3, 2018 at 8pm

This is the last homework before Midterm 2.

---

1. After moving to a new city, you decide to choose a walking route from your home to your new office. To get a good daily workout, your route must consist of an uphill path (for exercise) followed by a downhill path (to cool down), or just an uphill path, or just a downhill path.[1] (You'll walk the same path home, so you'll get exercise one way or the other.) But you also want the *shortest* path that satisfies these conditions, so that you actually get to work on time.

   Your input consists of an undirected graph G, whose vertices represent intersections and whose edges represent road segments, along with a start vertex s and a target vertex t. Every vertex v has a value $h(v)$, which is the height of that intersection above sea level, and each edge uv has a value $\ell(uv)$, which is the length of that road segment.

   (a) Describe and analyze an algorithm to find the shortest uphill–downhill walk from s to t. Assume all vertex heights are distinct.

   (b) Suppose you discover that there is no path from s to t with the structure you want. Describe an algorithm to find a path from s to t that alternates between "uphill" and "downhill" subpaths as few times as possible, and has minimum length among all such paths. (There may be even shorter paths with more alternations, but you don't care about them.) Again, assume all vertex heights are distinct.

2. Let $G = (V, E)$ be a directed graph with weighted edges; edge weights could be positive, negative, or zero.

   (a) How could we delete an arbitrary vertex v from this graph, without changing the shortest-path distance between any other pair of vertices? Describe an algorithm that constructs a directed graph $G' = (V', E')$ with weighted edges, where $V' = V \setminus \{v\}$, and the shortest-path distance between any two nodes in G' is equal to the shortest-path distance between the same two nodes in G, in $O(V^2)$ time.

   (b) Now suppose we have already computed all shortest-path distances in G'. Describe an algorithm to compute the shortest-path distances in the original graph G from v to every other vertex, and from every other vertex to v, all in $O(V^2)$ time.

   (c) Combine parts (a) and (b) into another all-pairs shortest path algorithm that runs in $O(V^3)$ time. (The resulting algorithm is *almost* the same as Floyd-Warshall!)

[1] A *hill* is an area of land that extends above the surrounding terrain, usually at a fairly gentle gradient. Like a building, but smoother and made of dirt and rock and trees instead of steel and concrete. It's hard to explain.

3. The first morning after returning from a glorious spring break, Alice wakes to discover that her car won't start, so she has to get to her classes at Sham-Poobanana University by public transit. She has a complete transit schedule for Poobanana County. The bus routes are represented in the schedule by a directed graph G, whose vertices represent bus stops and whose edges represent bus routes between those stops. For each edge $u \to v$, the schedule records three positive real numbers:

   - $\ell(u \to v)$ is the length of the bus ride from stop u to stop v (in minutes)
   - $f(u \to v)$ is the first time (in minutes past 12am) that a bus leaves stop u for stop v.
   - $\Delta(u \to v)$ is the time between successive departures from stop u to stop v (in minutes).

   Thus, the first bus for this route leaves u at time $f(u \to v)$ and arrives at v at time $f(u \to v) + \ell(u \to v)$, the second bus leaves u at time $f(u \to v) + \Delta(u \to v)$ and arrives at v at time $f(u \to v) + \Delta(u \to v) + \ell(u \to v)$, the third bus leaves u at time $f(u \to v) + 2 \cdot \Delta(u \to v)$ and arrives at v at time $f(u \to v) + 2 \cdot \Delta(u \to v) + \ell(u \to v)$, and so on.

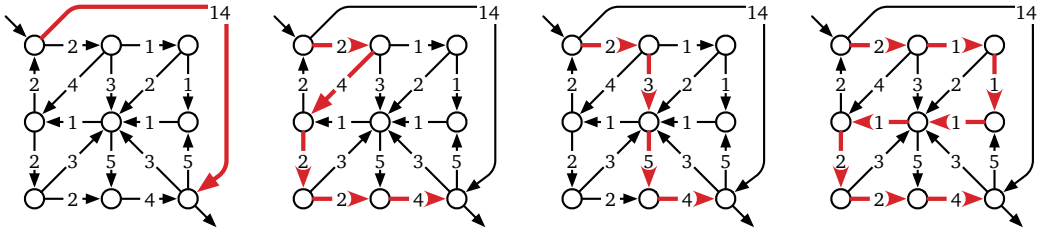
   Alice wants to leaves from stop s (her home) at a certain time and arrive at stop t (The See-Bull Center for Fake News Detection) as quickly as possible. If Alice arrives at a stop on one bus at the exact time that another bus is scheduled to leave, she can catch the second bus. Because she's a student at SPU, Alice can ride the bus for free, so she doesn't care how many times she has to change buses.

   Describe and analyze an algorithm to find the earliest time Alice can reach her destination. Your input consists of the directed graph $G = (V, E)$, the vertices s and t, the values $\ell(e), f(e), \Delta(e)$ for each edge $e \in E$, and Alice's starting time (in minutes past 12am).

   [Hint: In this rare instance, modifying the algorithm may be more efficient than modifying the input graph. Don't describe the algorithm from scratch; just describe your changes.]

Solved Problem

4. 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 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 the 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)