86/100 Didn't know how to do graph transform for (5)

Ravi Dayabhai

Introduction to Algorithms: 6.006
Massachusetts Institute of Technology
Instructors: Erik Demaine, Jason Ku, and Justin Solomon
Problem Set 6

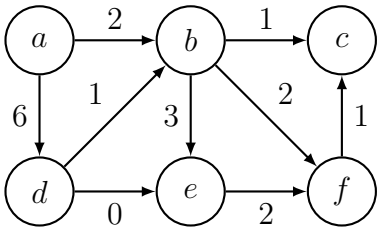
Problem Set 6

Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct.

Problem 6-1. [15 points] **Dijkstra Practice**

Consider weighted graph $G = (V, E, w)$ below, which is acyclic and has nonnegative edge weights.

(a) Run both DAG Relaxation and Dijkstra on G from vertex a. Each algorithm tries to relax every edge $(u, v) \in E$, but does so in a different order. For each algorithm, write down a list of all edges in **relaxation order**: the order the algorithm tries to relax them. If there is ambiguity on which vertex or outgoing adjacency to process next, process them in **alphabetical order**.



(b) List $\delta(a, v)$ for each $v \in V$ (via either algorithm).

Problem 6-2. [10 points] **Short Circuits**

Given a weighted directed graph $G = (V, E, w)$ and vertex $s \in V$, with the property that, for every vertex $v \in V$, some minimum-weight path from s to v traverses at most k edges, describe an algorithm to find the shortest-path weight from s to each $v \in V$ in $O(|V| + k|E|)$ time.

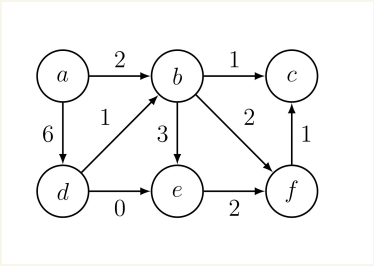
Problem 6-3. [20 points] **Dynamite Detonation**

Superspy Bames Jond is fleeing the alpine mountain lair of Silvertoe, the evil tycoon. Bames has stolen a pair of skis and a trail map listing the mountain's clearings and slopes (n in total), and she wants to ski from the clearing L by the lair to a clearing S where a snowmobile awaits.

- Each **clearing** $c_i \in C$ has an integer **elevation** e_i above sea level.
- Each **slope** (c_i, c_j, ℓ_{ij}) connects a pair of clearings c_i and c_j with a **monotonic** trail (strictly decreasing or increasing in elevation) with positive integer length ℓ_{ij}. Bames doesn't have time to ski uphill, so she will only traverse slopes so as to decrease her elevation.
- On her way up the mountain, Bames laced the mountain with **dynamite** at known locations $C_D \subset C$. Detonating the dynamite will immediately change the elevation of clearings $c_i \in C_D$ by a known amount, from e_i to a lower elevation $e'_i < e_i$. The **detonator** exists at clearing $D \in C$ (where there is no dynamite). If she reaches clearing D, she can choose to detonate the dynamite before continuing on.

Given Bames' map and dynamite data, describe an $O(n)$-time algorithm to find the minimum distance she must ski to reach the snowmobile (possibly detonating the dynamite along the way).

① [A] Relaxation order (where ties are broken alphabetically, when vertices are being processed:



DAG Relaxation: [(a,b), (a,d), (d,b), (d,e), (b,c), (b,c), (b,f), (e,f), (f,c)]

(from topological sort of vertices: [a, d, b, e, f, c])

Dijkstra: [(a,b), (a,d), (b,c), (b,e), (b,f), (f,c), (c,f), (d,b), (d,c)]

(greedy in the "frontier")

Those edges (u,v) did not update d[v], the estimated SSSP distances.

[B]

$\delta(a,a) = 0$

$\delta(a,b) = 2$

$\delta(a,c) = 3$

$\delta(a,d) = 6$

$\delta(a,e) = 5$

$\delta(a,f) = 4$

② ✓ Solving SSSP problem for source vertex s to all vertices $v \in V$ can be done in $O(|V| + |E| \cdot k)$ time by running a modified Bellman-Ford, i.e., relaxing all edges k times. This works because G has no negative cycles (since some min. weight path $\pi^*(s,v)$ consisting of at most k edges exists for all $v \in V$) and any finite $\pi^*(s,v)$ must be simple. $O(|E|k)$ time to do these exhaustive relaxations, $O(|V|)$ time to recover $\delta(s,v)$ for all $v \in V$. (Note: this requires Bellman-Ford to be implemented not using graph duplication, instead by looping k times, relaxing all outgoing edges for all vertices.)

(5) We can treat each clearing as a vertex in a graph whose edges are given by the slopes and the relative elevation of the incident clearings, i.e., a slope (c_i, c_j, b_j) $\in$ for graph $G(C, E)$ iff $e_i > e_j$ since Bames only cares about losing elevation $\Rightarrow$ G is a DAG. In $O(n)$ time, graph construction and DAG relaxation would give $\delta^*(L, S)$ when Bames doesn't use dynamite. If she does, she needs to find $\delta(L, D)$ first (also done in linear time w/ DAG Relaxation), then set up a new graph $G'(C', E')$ representing a new DAG after detonating the dynamite, then run DAG Relaxation with source vertex D (note: C' excludes clearings that are higher than D). The shortest path distance will be the smaller of $\delta^*(L, S)$ and $\delta^*(L, D) + \delta^{**}(D, S)$ where * values come from SSSP over G, and ** values come from SSSP over G'. (Note: edge weights are given by the positive integer trail lengths.)

Can run DAG relaxation once by connecting D^* D^{**} by 0-weight edge and making terminus node T connected to S^*, S^{**}, but this also abides by run time constraint!

Problem 6-4. [10 points] **Conservative Cycles**

In Wentonian physics, a force field acts on the motion of a particle by requiring a certain positive or negative amount of **work** to move along any path. A force field is **conservative** if the total sum of work is zero **along every closed loop**. Given a discrete force field represented by n possible particle locations ℓ_i and $O(n)$ possible particle transitions[1], where transition (ℓ_i, ℓ_j, w_{ij}) moves a particle along a **directed** path from location ℓ_i to location ℓ_j requiring (positive or negative) integer work w_{ij}, describe an $O(n^2)$-time algorithm to determine whether the force field is conservative.

Problem 6-5. [20 points] **SparkPlug Derby**

LightQueen McNing is a race car that wants to drive to California to compete in a road race: the annual SparkPlug Derby. She has a map depicting:

- the n intersections in the country, where each intersection x_i is marked with its positive integer **elevation** e_i and whether it contains a **gas station**; and
- the r roads connecting pairs of them, where each road r_j is marked with the positive integer t_j denoting the **driving time** it will take LightQueen to drive along it in either direction.

Some intersections connect to many roads, but the average number of roads at any intersection is less than 5. LightQueen needs to get from Carburetor Falls at intersection s to the SparkPlug Derby race track at intersection t, subject to the following conditions:

- LightQueen's gas tank has a positive integer **capacity** $g < n$: it can hold up to g units of gas at a time (starting full). Along the way, she can refill her tank (by any integer amount) at any intersection marked with a gas station. It takes exactly t_G time for her to fill up 1 unit of gas.
- LightQueen uses gas only when **driving uphill**. Specifically, if she drives on a road from intersection x_i to x_j at elevations e_i and e_j respectively, LightQueen will use exactly $e_j - e_i$ units of gas to travel along it if $e_j > e_i$, and will use zero units of gas otherwise.

Given LightQueen's map, describe an $O(n^2 \log n)$-time algorithm to return a fastest route to the race that keeps a **strictly positive** amount of gas in her tank at all times (if such a route exists).

Problem 6-6. [25 points] **Johnson's Algorithm**

In this problem, you will implement Johnson's Algorithm to compute all-pairs shortest-path weights in a general weighted directed graph G, as described in Lecture 14. The input to your algorithm will be: a positive integer n representing the number of vertices of your graph identified by consecutive integers 0 to $n-1$, and a tuple S of triples, where each triple (u, v, w) corresponds to a directed edge from vertex u to v of weight w. Your output should be a length-n tuple D of length-n tuples where $D[u][v] = \delta(u, v)$ for all $u, v \in \{0, \ldots, n-1\}$, except when the input graph contains a **negative-weight cycle** when you should return `None`.

Please implement the `johnson(n, S)` function in the template code provided. Your code template contains working implementations of Bellman–Ford and Dijkstra (using a binary heap as a priority queue) modified from the recitation notes. Note that you will have to construct your own adjacency list and weight function if you would like to use this code. You can download the code template and some test cases from the website.

[1] Note that a transition from ℓ_i to ℓ_j does not imply the existence of a transition from ℓ_j to ℓ_i.

④ A force field is a digraph whose nodes are particle locations and whose edges are transitions (where the transition's required work is the edge weight). Constructing this graph takes linear time, and running Bellman-Ford (implemented without graph duplication up front) takes $O(n^2)$ time — if a negative weight cycle exists, the force field is not conservative. Repeating the above with the signs of all edges flipped (if the first pass did not invalidate the force field as conservative) will tell us if there exists positive length loops in the original graph because they will present as negative weight cycles in the transformed graph. If both passes of Bellman-Ford yield no presence of negative weight cycles, the force field is conservative.

Yes, but state explicitly creation of super node s that serves as the single source when using SSSP algorithm.

6.006 Introduction to Algorithms

Spring 2020