CS 344 - Sections 5,6,7,8 - Spring 2021
Homework 6
Due Monday, April 26, 4:00pm
100 points total

General Assumption: You can assume that all graphs given in this problem set have zero isolated vertices, so $|E| \geq |V|/2$.

1 Problem 1 (25 points)

Say that you are given a directed graph $G = (V, E)$ and a source s. Say that all the edge weights of G are non-negative except that there is EXACTLY ONE edge (x, y) that has negative weight.

Part 1 (10 points) Write pseudocode for an algorithm that in $O(|E| \log(|V|))$ time detects whether G contains a negative-weight cycle. The algorithm should return "YES CYCLE" if there is a negative-weight cylce and "NO CYCLE" if there is not a negative weight cycle.

Part 2 (15 points) Consider the graph G above and say that you have the additional guarantee that G does NOT contain a negative-weight cycle. Write pseudocode for an algorithm that in $O(|E| \log(|V|))$ time computes dist(s,v) for all vertices v.

HINT: your solution to both parts will want create a new graph G' which is very slightly different from G and then run several instances of one of the algorithms in our library on the graph G'. You can then use the distances you learn in G' to figure out the answer you need about G.

2 Problem 2 (10 points)

In the Bellman ford algorithm from class, if we detected a negative cycle we just returned "ERROR" and gave up on finding distances. But this is not quite justified, because even if there a negative cycle then for some vertex x we might have $\text{dist}(s, x) = -\infty$ (because the negative cycle is between s and

x), but there could be another vertex y for which dist$(s, y) = 100$ (because there is no negative cycle between s and y.)

Consider the following problem

- INPUT: a directed graph $G = (V, E)$ that can have negative edge weights and might have negative cycles.
- OUTPUT: dist(s,v) for every vertex $v \in V$, where for some vertices x you might have dist$(s, x) = -\infty$

Write pseudocode for an algorithm that solves the above problem in $O(|V||E|)$ time.

NOTE: all you need to do is write pseudocode; no explanation needed

HINT: most of your pseudocode can be copied directly from the Bellman-Ford algorithm in class.

3 Problem 3 (20 points)

Recall the topological ordering problem from class.

INPUT: a DAG $G = (V, E)$

OUTPUT: a topological ordering $v_1, ..., v_n$ of the vertices.

Write pseudocode for an algorithm that solves this problem in $O(|E|)$ time.

IMPORTANT NOTE: to keep notation consistent, please use the following notation for this problem. Assume that the original vertex set is $V = v'_1, ..., v'_n$, which are NOT in topological order. (I am using v'_i to avoid confusion with the v_i of a topological ordering.) Your goal is then to sort the vertices so they are in topological order. So for example, if the correct topological order is v'_2, v'_3, v'_1, then you should output the list v'_2, v'_3, v'_1. Alternatively, you could output

- $v_1 \leftarrow v'_2$
- $v_2 \leftarrow v'_3$
- $v_3 \leftarrow v'_1$,

You can use either of the two output methods described above.

HINT: you will want to follow the approach from class of repeatedly finding zero-degree vertices. Think about how to implement that efficiently.

Note that it's not necessary to literally remove edges/vertices as long as you have an efficient way of finding the next vertex in the topological order in each step. But you are welcome to remove vertices if that's easier for you, as long as the overall runtime is $O(|E|)$.

4 Problem 4 (20 points)

Say that G is NOT a DAG. Now, say that I try running an analogue of the topological ordering algorithm from class

RemoveZeroDegree(G)

- While there exists a vertex x with $|In(x)| = 0$
  - Remove x from G
  - Remove all edges in $Out(x)$ from G

Part 1 (10 points) Argue that if G contains a cycle, then when the algorithm terminates there will be at least one edge remaining in the graph. A proof by contradiction is ideal, but a more informal argument will also get you fell credit, as long as you clearly state the main ideas. Your answer should be 4-5 sentences MAX (shorter is better.)

ASIDE: this algorithm can be used to detect if the graph is a DAG. We showed in class that if G is a DAG then RemoveZeroDegree(G) will eventually remove every vertex/edge. Now in this problem you are showing that if G is not a DAG then there will be at least one edge remaining, so we will know G is not a DAG.

Part 2 (10 points) Say that G is NOT a DAG and let G' be the graph remaining after we run RemoveZeroDegree(G).

QUESTION: is it necessarily the case that *every* vertex in G' is on some directed cycle? If yes, give a brief argument that this is always the case. In no, show a counterexample.

5 Problem 5 (10 points)

Consider the following problem:

- INPUT
  - A directed graph G where all edges have weight -1, 0, or 1
  - Two fixed vertices s,t.
- OUTPUT: dist(s,t)

One way to solve this problem is to run BellmanFord(G,s), but this is somewhat slow. Professor F. Lake. suggests the following alternative algorithm:

- Create a new graph G' which is the same as G except that all weights are increased by 1. So G' uses weight function w', where $w'(x, y) = w(x, y) + 1$.
- Run Dijkstra(G',s)
- Let P be the shortest s-t path in G' returned by Dijkstra(G',s)
- Output $w(P)$ as your final answer, where $w(P)$ is the weight of path P in the original graph G.

The Problem: Give an example graph G where the above algorithm produces an incorrect solution. Concretely, you need to show a graph G such that the shortest $s - t$ path in G' is NOT the shortest $s - t$ path in G.

6 Problem 6 (15 points)

Part 1 (10 points) Recall that in Johnson's algorithm, we used node potentials $\phi(v)$. In particular, the algorithm creates a dummy source s and sets $\phi(v) = \text{dist(s,v)}$.

In this problem, the goal is to show why we need a dummy source. Consider the following alternative approach to computing node potentials

- Let s be an arbitrary vertex in V. So s is not a dummy source, but rather one of the vertices in the graph.
- do NOT add any edge edges to the graph
- Run BellmanFord(G,s)

- Let $\phi(v) = \text{dist(s,v)}$

The Problem: Give an example of a graph $G = (V, E)$ with no negative-weight cycles and a vertex $s \in V$, such that the alternative approach above fails. That is, the node potentials it produces will not lead to the kind of reduced weights we need for Johnson's algorithm to go through.

Then, briefly (one or two sentences) describe for what kinds of graphs the alternative approach is guaranteed to work. That is, for what kinds of graph will the above approach no matter which vertex $s \in V$ the algorithm picks in the first step.

Part 2 (5 points) Let G be a directed graph with (possibly negative) edge weights $w(x, y)$ but no negative weight cycles. Let s be the dummy source that we add in Johnson's algorithm and let G^+ be the graph G with this additional dummy source, along with the weight-0 edges Johnson's algorithm adds from the dummy source. Recall that Johnson's algorithm then runs BellmanFord(G^+, s) and sets $\phi(v) = dist_{G^+}(s, v)$ and then for each edge (x, y) defines reduced weight $w'(x, y) = w(x, y) + \phi(x) - \phi(y)$.

The Problem Say that some edge (u, v) is on the shortest path from s to v in G^+. What does this imply about $w'(u, v)$? Briefly justify your answer. The justification does not have to be a formal proof; just a one to two sentence description that should convince someone of your claim about $w'(u, v)$.

HINT: the question may seem open ended, but it is not. There is an extremely concrete thing you can say about $w'(u, v)$.

NOTE: just saying that $w'(u, v) \geq 0$ will get you no credit, since Johnson's algorithm guarantees that ALL reduces weights are non-negative. The point of this problem is that because we know (u, v) is on the shortest path from s to v, we can say something much more specific about $w'(u, v)$.