CMPSC 465 Data Structures & Algorithms
Fall 2025 Ke Chen and Yana Safonova

HW 6

Due October 20th, 10:00 pm

Instructions: You are encouraged to solve the problem sets on your own, or in groups of three to five people, but you must write your solutions strictly by yourself. You must explicitly acknowledge in your write-up all your collaborators, as well as any books, papers, web pages, etc. you got ideas from.

Formatting: Each part of each problem should begin on a new page. Each page should be clearly labeled with the problem number and the problem part. The pages of your homework submissions must be in order. **When submitting in Gradescope, make sure that you assign pages to problems from the rubric. You risk receiving no credit for it if you do not adhere to these guidelines.**

Late homework will not be accepted. Please, do not ask for extensions since we will provide solutions shortly after the due date. Remember that we will drop your lowest three scores.

This homework is due Monday, October 20, at 10:00 pm electronically. You need to submit it via Gradescope. Please ask on Campuswire about any details concerning Gradescope.

1. (10 pts.) **Dijkstra with Reinsertion.** Let $G = (V, E)$ be a directed graph with weights $\ell : E \to \mathbb{R}$ and let $s \in V$ be the starting vertex. Consider the variant of Dijkstra that, after a vertex has been extracted, allows it to be reinserted into the priority queue if a later Update call reduces its distance from s.

(a) Does this algorithm correctly compute single-source shortest paths from s when negative edge weights are allowed but there are no negative cycles?

(b) Analyze the runtime complexity of the algorithm.

2. (20 pts.) **Budget Bellman-Ford.** Given a directed graph $G = (V, E)$, a weight function of the edges $\ell : E \to \mathbb{R}$ (you can assume there is no negative cycles in G), a starting vertex s, and an integer budget $k \geq 0$, we would like to compute for every vertex $v \in V$ the shortest path from s to v that **uses at most k edges**. Give an algorithm for this problem, argue for its correctness, and analyze its time complexity.

3. (30 pts.) **Ordered Bellman-Ford.** Bellman–Ford performs $|V| - 1$ rounds of Update calls on all edges; a practical improvement is to stop early when no Update takes effect in the previous round.

The following variant aims to reduce the number of rounds required: in each round, call Update on the edges in nondecreasing order of their weights, so that edges with smaller weights are updated earlier.

(a) Does this variant still correctly compute the shortest distances from the starting vertex s?

(b) Give an example where this variant reduces the number of rounds required (namely, there is some order of updates that needs $|V| - 1$ rounds but this method requires fewer), or argue that no such instance exists.

(c) Give an instance where this variant does not reduce the number of rounds required (namely, this method needs $|V| - 1$ updates), or argue that no such instance exists.

4. (20 pts.) **Pinned Nodes.** Given a directed graph $G = (V, E)$, with n vertices (namely, $n = |V|$), a weight function on edges $\ell : E \to \mathbb{R}$ (you can assume the graph has no negative cycles), and a subset $S \subseteq V$ of special vertices, the task is to compute $d(i, j)$, the length of the shortest path from i to j that **includes at least one vertex from** S, for all pairs of vertices $(i, j) \in V \times V$. If no such path exists, the length is defined to be ∞.

(a) Design an algorithm, based on the Floyd–Warshall method (or using it as a subroutine), that computes $d(i, j)$ as defined above for all pairs (i, j).

(b) Prove the correctness of your algorithm and analyze its time and space complexity.

5. (20 pts.) **Bottleneck Path.** Consider a directed weighted graph $G = (V, E)$, where V is a set of n vertices, and $E \subseteq V \times V$ is a set of directed edges, each associated with a non-negative weight $w : E \to \mathbb{R}_{\geq 0}$. The bottleneck weight of a path from vertex i to vertex j is defined as the maximum weight of any edge in the path. The objective is to compute, for each pair of vertices $(i, j) \in V \times V$, the path from i to j with the minimum bottleneck weight. If no path exists from i to j, the bottleneck weight is defined to be ∞. Formally, for each pair (i, j), we seek the value $b(i, j)$, defined as:

$$b(i, j) = \min_{\substack{\text{path} \\ p: i \to j}} \left(\max_{(u,v) \in p} w(u, v) \right),$$

where the minimum is taken over all simple paths p from i to j, and $\max_{(u,v) \in p} w(u, v)$ is the bottleneck weight of a path p. Design an algorithm based on the Floyd-Warshall algorithm to find minimum bottleneck weight for all pairs of vertices in the graph. Argue for its correctness and analyze its time complexity.