CS3000: Algorithms & Data — Spring 2019 — Paul Hand

Homework 7
Due Wednesday 4/10/2019 at 2:50pm via Gradescope

Name:
Collaborators:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- This assignment is due Wednesday 4/10/2019 at 2:50pm via Gradescope. No late assignments will be accepted. Make sure to submit something before the deadline.
- Solutions must be typeset in LaTeX. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF. I recommend using the source file for this assignment to get started.
- I encourage you to work with your classmates on the homework problems. *If you do collaborate, you must write all solutions by yourself, in your own words.* Do not submit anything you cannot explain. Please list all your collaborators in your solution for each problem by filling in the `yourcollaborators` command.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class is strictly forbidden.

Problem 1. *All-Pairs Shortest Paths*

In the *all-pairs shortest paths* problem, you are given a directed, weighted graph with edge lengths $G = (V, E, \{\ell_e\})$, and have to find the length of the shortest path from s to t for *every pair* $s, t \in V$. For this HW we only want the *length* of the shortest path and not the path itself.

If all edge lengths are non-negative ($\ell_e \geq 0$), then we can solve this problem by running Dijkstra's algorithm from every source node $s \in V$, incurring running time $O(nm \log n)$. However, if lengths can be negative, then running Bellman-Ford from each source node $s \in V$ incurs running time $O(n^2 m)$. In this question we will study the following algorithm for solving all-pairs shortest paths in graphs with negative-length edges, but no negative-length cycles.

- Modify the input graph by adding an additional node z connected to every other node v by a zero-length edge (z, v).
- Run the Bellman-Ford algorithm on the modified graph with source z to find the length $f(v)$ of the shortest $z \to v$ path in the modified graph.
- Define new edge lengths $\ell'_{u,v} = \ell_{u,v} + f(u) - f(v)$ and let $G' = (V, E, \{\ell'_e\})$ be the input graph with these modified edge weights.
- For each source $s \in V$, run Dijkstra's algorithm on the graph G' with source s to find the length $d'(s, v)$ of the shortest $s \to v$ path in G' for every node v.
- For every $u, v \in V$, let $d(u, v) = d'(u, v) - f(u) + f(v)$. Output the values $\{d(u, v)\}$.

In this problem, we will show correctness and analyze the running time of this algorithm. The final three steps of the problem form the proof of correctness.

(a) What is the running time of this algorithm? Briefly explain your answer.

Solution:

(b) Prove that every edge in G' has non-negative length. That is, $\forall u, v \in V, \ell'_{u,v} \geq 0$. (Thus, Dijkstra's algorithm will correctly find the length $d'(u, v)$ of the shortest $u \to v$ path in G'.)

Solution:

(c) Prove that for every $u \to v$ path $P = u \to w_1 \to \cdots \to w_{k-1} \to v$, we have $\ell'_P = \ell_P + f(u) - f(v)$ where ℓ'_P, ℓ_P are the length of the path in G' and G, respectively.

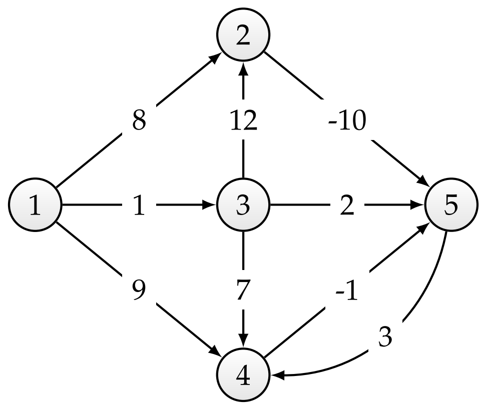
Solution:

(d) Prove that for every $u, v \in V$, $d'(u, v) - f(u) + f(v)$ is the length of the shortest $u \to v$ path in the original graph G. Thus, the final lengths output by this algorithm are correct.

Solution:

Problem 2. *Shortest Paths Practice*

Solve the single-source shortest path problem on the following graph, using node 1 as the source. Write the distance from s to each node and write the parent of each node in the shortest path tree. **Hint:** Write your solution using the skeleton table provided.



Solution:

Node	1	2	3	4	5
Distance Parent	 $\perp$				

Problem 3. *Priority Queues and Heaps*

Write pseudocode for a priority queue implemented via a heap. The only data structures you are allowed to use are arrays. You should keep track of the length of the arrays using a variable. You may assume that the priority queue will never hold more than N items, and you may initialize arrays to have length N. You may assume that all keys are an integer from 0 to $N-1$. You do not need to implement error checking with user input (for example if a user calls ExtractMin on a queue with no entries, or if a user attempts to add an entry for a key that is already in the queue, or if a user attempts to add an entry to a queue that already contains N entries, etc.).

Use the global variable V for the array of (key,value) pairs in the heap. Use the notation $V[k]$.key and $V[k]$.value to refer to the key and value of the kth entry of V. Use the global variable K for the array of indexes within V corresponding to each key. Use the global variable L for the current number of items in the queue.

You must implement the following operations:

- INITIALIZE(N). Initializes a priority queue that can handle at most N items.
- INSERT(k, v). Adds the key-value pair (k, v) to the priority queue.
- LOOKUP(k). Returns the value corresponding to the key k.
- EXTRACTMIN(). Returns the key-value pair (k, v) of minimal value and removes it from the priority queue.
- DECREASEKEY(k, v). Decreases the value of k to v, which you may assume is less than its previous value.

Solution: