Fundamental Algorithms, Section 003
Homework 11, Fall 22. Due date: Wednesday, December 07, 4:55pm on Gradescope

1. Let $G = (V, E)$ be a weighted directed graph and let $s \in V$ be a designated vertex; assume that all the edge weights are non-negative. The task is to provide an algorithm to determine for each vertex $v \in V$ whether there is a unique shortest path from s to v. There is a solution which avoids counting the number of paths; try to find it. Argue that your solution is correct. The resulting algorithm should have the same running time as Dijkstra's algorithm on a graph of n vertices and m edges, up to constant factors; argue that your algorithm does so. You may assume $n \leq m$.

2. Taxi and subway. Let $G = (V, E)$ be a weighted directed graph and let $s \in V$ be a designated vertex. All the weights are positive and indicate the dollar cost of traversing an edge by taxi. Let S be a subset of the vertices, the subway stops. You can travel from any vertex in S to any other vertex in S for a cost of \$2.75. Your task is to compute, for every vertex $v \in V$, the cost of the least cost path from s to v, using any mode of transport (some combination of taxis and subway). You are to do this by means of a reduction to the single source shortest path problem. Argue that your reduction is correct. Your algorithm should run in the same time as Dijkstra's algorithm on a graph of n vertices and m edges, up to constant factors; argue that it does so.

3. Let $G = (V, E)$ be a directed graph with two weights, w_e and r_e, on each edge e; assume that all the edge weights are non-negative. There are two special vertices in this graph: c, the chemical plant, and a, the amphitheater. The task is to create an evacuation plan from a to v for every vertex v in V in the event that the chemical plant catches fire. The time it takes the fire to spread along an edge e is r_e. The time to travel along an edge is w_e. We want to find the shortest safe path from a to v for each $v \in V$: a path is safe if it does not include any vertex on fire at the time that vertex is traversed. Your algorithm should run in the same time as Dijkstra's algorithm on a graph of n vertices and m edges, up to constant factors. Justify your running time. Also, argue that your construction is correct.
Hint. i. First compute the time at which each vertex catches fire, assuming the chemical plant catches fire at time 0.
ii. Now modify Dijkstra's algorithm so that it does not use any vertex on fire on the paths it computes.

4. There is an algorithm for the all pairs shortest path problem that runs in $O(n^3)$ time, called the Floyd-Warshall algorithm. Here, the input is a directed weighted graph $H = (V_H, E_H)$ given in adjacency matrix format. You may assume there is a matrix $W_H[1 : n, 1 : n]$ which stores the weights of the edges, where, for pairs of vertices u, v with $(u, v) \notin E_H$, $W_H[u, v] = \infty$. The output is the length of the shortest path from u to v for each pair u, v of vertices. The assumption needed here is that there are no negative length cycles.

Suppose you are given a directed weighted graph $G = (V_G, E_G)$, with the edge weights stored in matrix $W_G[1 : n, 1 : n]$. The task is to give an algorithm to determine, for every pair of vertices $u, v \in V_G$, the length of a shortest path from u to v that uses an even number of edges. Solve this problem by means of a reduction to the standard all pairs shortest path algorithm. Your algorithm should run in $O(n^3)$ time; justify your running time.

You do not need to know the Floyd-Warshall algorithm in order to solve this problem. All you need to know is the input-output definition of the all pairs shortest path problem.

Challenge problem. Do not submit.

5. This problem will develop a version of the relaxed heap.

We start by introducing binomial trees. Every binomial tree has an integer rank r. A rank r binomial tree has 2^r nodes. We will also say that the root of a rank r binomial tree has rank r. Binomial trees are defined as follows. A rank 0 tree comprises a single node. For $r \geq 0$, a rank $r+1$ binomial tree consists of two rank r binomial trees, T_1 and T_2, with T_2's root being an additional child of T_1's root. Another way of viewing this is that a rank $r+1$ binomial tree has $r+1$ children, which are the roots of binomial trees of ranks $0, 1, 2, \ldots, r$, respectively. Prove that these views are equivalent.

To store a binomial tree we maintain the following pointers. Each node has a pointer to its parent, if any. The children of a node are kept in a doubly linked list, in rank order. Each non-leaf node has a pointer to its largest rank child.

A binomial heap on n items consists of a collection of at most $\log n$ binomial trees, all having distinct ranks. Each node stores one item, with the items obeying the heap property: if e is the item at node v, the item e' at v's parent will have a smaller key than e (if equal valued keys could occur, then we allow ties).

The following Merge(T_1, T_2) operation is very useful in implementing the standard priority queue operations. Merge is defined on trees of equal rank, r say. WLOG, suppose that $\text{root}(T_1).\,\text{key} \leq \text{root}(T_2).\,\text{key}$. Then the mege operation makes the root of T_2 a child of T_1's root, thereby creating a rank $r+1$ binomial tree. This can be done in $O(1)$ time. Verify this.

We implement the standard operations as follows.

- Insert(e): Create a new 1-item binomial tree. Then repeatedly merge equal rank trees, if any, until all the remaining trees have distinct ranks.
- DeleteMin: Examine the items at the roots of all $O(\log n)$ trees in the heap; remove the root holding the item e with the smallest key; if the root had rank $r+1$, this creates $r+1$ binomial trees. Now, repeatedly merge equal rank trees, if any, until all the remaining trees have distinct ranks.
- ReduceKey(e, k): This operation assumes one is given a pointer to the node holding item e. Repeatedly swap e with the item at its parent, continuing until the heap property is restored.

Argue that each heap operation can be implemented in $O(\log n)$ time. In addition, argue that starting with an empty heap, if one performs n insert operations right away, they will take a total of $O(n)$ time.

The relaxed heap.

In a relaxed heap, we allow the heap property to be violated at up to $\log n$ nodes of a binomial heap; also, we will maintain a bound of $\log n$ on the number of binomial trees in the relaxed heap, but possibly some of these trees have equal rank. The goal is to perform Insert and ReduceKey operations in $O(1)$ worst case time, and DeleteMin operations in

$O(\log n)$ worst case time. DeleteMins are performed as before. An insert is simplified to be the creation of a one node tree, together with one merge of equal rank trees, if any such pair is present. Now, a ReduceKey operation is performed in place, which may add one violation, but if there are violations at equal rank nodes, then it will remove one of them. It is your task to figure out how to remove a violation in $O(1)$ time.

Suppose the equal rank violations of rank r are both the largest rank children of their parents. Let the roots of the equal rank children be nodes u and v. Let their parents be named w and x. If we remove u and v from their parents, we obtain 4 rank r trees, rooted at u v, w, and x, respectively. We now merge the trees rooted at u and v and the trees rooted at w and x. Finally, we reinsert these trees in such a way that there is at most 1 violation where before there were 2.

Now show how to handle the other two cases: neither u nor v is the largest rank child of its parent; one of u and v is the largest rank child and the other is not.

Fill in any other missing details.