26 (100 PTS.) Stay safe

We are given an *undirected* graph with n vertices and m edges ($m \geq n$), where each edge e has a positive real weight $w(e)$, and each vertex is marked as either "safe" or "dangerous".

26.A. (35 PTS.) Given safe vertices s and t, describe an $O(m)$-time algorithm to find a path from s to t that passes through the smallest number of dangerous vertices.

26.B. (65 PTS.) Given safe vertices s and t and a value W, describe an algorithm, as fast as possible, to find a path from s to t that passes through the smallest number of dangerous vertices, subject to the constraint that the total weight of the path is at most W.

26

Solution:

1. **Idea:** The idea behind the algorithm is to follow a modified version of BFS operating on a new graph weighted by its proximity to dangerous nodes. First we start off by making the weight of each edge connected to a dangerous node as 1 and all other edges that connect safe nodes as 0. We then run BFS in a way similar to Dijkstra's by checking edge weights and updating the minimum value at each node when needed. We enqueue and dequeue edges as in BFS using a double edged queue. One difference is that we enqueue at the front of the queue when we come across an edge to a safe node and enqueue at the end of the queue when the edge goes towards a dangerous node. This keeps the queue sorted by each level which is a requirement for BFS to operate correctly. We can do this since we only have 2 options for each edge based on the value of the edge, either u is in the same level as v or the next one.

   Psuedocode: See image 1.

   Analysis: This algorithm utilizes the BFS algorithm to search for the shortest path. A major difference is the utilization of a double ended queue which allows enqueues and dequeues at both the beginning and end. These operations are supported in O(1) time using an implementation such as C++ STL's version therefore not affecting run time. This allows the algorithm to retain BFS's runtime of O(m+n). Since (m≥n) for this question $O(m+n) = O(m)$, satisfying the runtime requirement of this problem.

2. **Idea:** The approach we took for this problem is a modified algorithm from part A. We started with the same graph pre-processing and setup. The first part of the approach was to think about "removing" all dangerous nodes then using a modified BFS/Dijkstras to find a path that had a total weight less than W. If this did not exist meaning we have to go through at least one dangerous node to get to T, then we cut the traversal short and increment the "allowed" number of dangerous nodes by one. We then restart our BFS/Dijkstras traversal with this criteria. If a possible path to T is achieved with the criteria that it is under weight W on the current iterations restraint of maximum dangerous nodes, then we return the path and the algorithm finishes. We iterate up until the worst case such that all nodes are dangerous. If a path still cannot be achieved, we return an error.

Psuedocode: see image 2

Analysis: For every maximum dangerous iteration, we run a dijkstras call. Every Dijkstras call takes O(M + NlogN). If every node is a dangrous node, the maximum dangrous iterator will go through N times. This makes the worst case runtime O(N*MlogN).
