Department of Electrical and Computer Engineering
EECE 2560: Fundamentals of Engineering Algorithms

Project #4

Part c

In this part, evaluate the effectiveness of the Bellman-Ford algorithm and Dijkstra's algorithm for finding shortest paths in graphs.

1. Write a global function `relax(Graph &g, vertex_descriptor u, vertex_descriptor v)` that relaxes the edge between nodes `u` and `v`.

2. Write a global function `bellmanFord(Graph &g, vertex_descriptor s)` that uses the Bellman-Ford algorithm to search for a shortest path from node `s` to every node in `g`, where the graph can have negative edge weights. If a shortest path exists to every node, returns true and sets each node's `pred` field equal to the predecessor node in a shortest path from node `s`. If a negative cycle is reachable from `s`, returns false. Use node weights to store the node values.

3. Write a global function `dijkstra(Graph &g, vertex_descriptor)` that uses Dijkstra's algorithm to search for a shortest path from node `s` to every node in `g`, where all node weights must be positive. If a path exists to every node, returns true and sets each node's `pred` field equal to the predecessor node in a shortest path from node `s`.

   Use a priority queue based on a min-heap to store the nodes and find the one with the smallest value.

   Use the weight property of nodes to store the shortest path estimates. Use `heapV::minHeapDecreaseKey(v)` to update the estimates for node v. The function is passed the index `i` of a node. Use `heapV::getIndex(v)` to find this index value.

Apply your algorithms to the graph files. The files include the number of nodes, the start node and the end node.

The code you submit should read a graph's file name from the keyboard, apply both shortest path functions to the graph, and then print the sequence of nodes in a shortest path. If a shortest path does not exist, your code should print "no shortest path exists".