Nathaniel Daniel

Yang

CS 365

7 May 2020

Bonus Assignment

I chose Rust to make Dijkstra's Algorithm. The code may be run by installing Rust, then running “cargo run” inside the directory “cs365-bonus”. The edge data is supplied in the format “{edge_name_src} {edge_name_dest} {cost}” per line. This will add a bidirectional relationship between both nodes, though the code may be trivially modified to support one-way relationships as well. An example input file is “cs365-bonus/input.txt”. This file contains the problem data for section 10.6 problem 4. In the input file, nodes are not explicitly declared, but implicitly through edge declarations. The program will try to load the input file stated above and find the shortest path using Dijkstra's Algorithm. It assumes that the edge named “a” is the start and that the edge named “z” is the end. The code may be trivially changed to allow different starting and terminal node definitions. The code is all contained in the file “main.rs”.

The code calculated the shortest path with a cost of 16. The path was a, b, e, h, l, m, p, s, z. The cost to reach b is 2. The cost to get to e is 3. The cost to get to h is 6. The cost to get to l is 7. The cost to get to m is 10. The cost to get to p is 12. The cost to get to s is 14. The program begins by loading all edges into a graph bidirectionally. The initial node is set to have no cost, and an empty “state” var is inserted into a priority queue, where a state var contains the corresponding node index and total cost to reach. For each state in the queue, the lowest is processed first. If the cost it has is greater than the current cost to the node, it is not processed

again as another state with a lower distance value has already been processed. Then, all connecting nodes are added to the queue if it could potentially have a lower cost than the already recorded cost for a given node. It also records each added node's parent so the program can later backtrack and rebuild the path. After the queue is empty, the program begins to rebuild the path. If the final node has no associated cost value, then it was not processed. The program exits and alerts the user a path could not be created. Otherwise, the program backtracks from the end to the beginning, recovering the parent of each node and the associated cost until it reaches the beginning node. The recovered data is then reversed to match the path a user would consider, and returns a structure with the total cost, distance values of each node, and the path. The program then converts the node indexes from the path back into their original names and prints to the console the original name and associated cost at that node.

Terminal Output:

Located a minimum path of cost: 16

a (0) -> b (2) -> e (3) -> h (6) -> l (7) -> m (10) -> p (12) -> s (14) -> z (16)