**CS315-001**

**Caleb Daniel Burnett**

**Assignment: Program 3**

Purpose: To implement an undirected weighted graph map of middle earth, and demonstrate how Dijkstra’s and BFS choose different paths.

**COMPILE INSTRUCTIONS:**

This was compiled on a linux machine using bash commands.

Step 1: ensure all files in this .zip file are in the same file on the Virtual Machine. You should see

* MiddleEarthEdges.txt
* MiddleEarthVertices.txt
* map.cpp
* Project 3 Report word document. (not needed for compiling).

Step 2:

Use the following g++ command:

g++ -o map map.cpp

Step 3:

Type the following command:

./map MiddleEarthVertices.txt MiddleEarthEdges.txt

The output of the code is this:

**BFS path: Hobbiton -> Southfarthing -> Isengard -> Edoras -> Rauros -> BlackGate -> MountDoom**

**Dijkstra path edge cost = 175): Dijkstra path: Hobbiton -> Bree -> Rivendell -> Moria -> Lorien -> Rauros -> BlackGate -> CirithUngol -> MountDoom**

**BFS cost count: 36**

**Dijkstra cost count: 36**

* **IMPLEMENTATION CHOICES**

The code reads in from the vertices.txt file and the edges.txt middle earth files, and populates the graph, treating edges as undirected, and inserting them into an adjacency list.

A priority queue is made using the Binary Min-Heap, using a vector of min heap nodes.

It preserves order by inserting each node and heapifying up.

The get\_min\_distance function swaps the root node of the tree with the last appended node, and then returns the root, which is known to be the shortest distance. It then calls the heapify up function.

* + BFS

For BFS, an unordered map (suggested by ChatGPT) takes a string, and a vector of pairs of strings and integers. The weights are ignored for BFS.

It then uses a string for its path search, and marks each neighbor as visited using another unordered map of string and bool.

Once a path is found, it calls PRINT PATH function from the textbook to print the path.

Ties are not handled, as it is assumed that there are not double values in the input text files.

To count the cost, a counter is incremented every time a neighbor is checked.

* + Dijkstra’s

Begins by initializing every vertex to the highest possible value for comparing minimum paths, and uses the priority queue of the min-heap function.

It iterates through the adjacent nodes, checking their weights, then chooses the adjacent node with the minimum weight.

Once the path is found, it uses the same PRINT PATH from the textbook.

It also counts its cost by counting how many edge checks it makes.

**Conclusions:**

Table from chat GPT for best and worst theoretical cases of algorithms, along with my cost counter result. It was prompted with my code and my code output. And told to make a table to show the theoretical worst case cost of the BFS algorithm using priority queue of min heap, and best case, and my “actual cost” counter, as well as the theoretical worst and best case for Dijkstras, and the values of V and E.

To measure the “actual cost” of both methods, I incremented my counters every time a neighbor was checked in the BFS, and in my edge-relaxation loop in Dijkstras.

The total edge cost of Dijkstra’s is 10+30+10+40+5+20+20+40 = 175

As can be seen, Dijkstra’s is within the worst case, and higher than the best case, as to be expected. As for BFS, I believe the discrepancy here is due to one more count than needed, or else my incrementer was incorrectly placed, or due to vertices being bi-directional for the BF search.

These algorithms choose different paths, but why?

The short answer is, one is taking the weights of the edges into consideration, (Dijkstra’s) as it travels through the graph, looking for the lowest cost path.

Being a LOTR fan, I knew my output was correct when I was able to get it to print, because it is book-accurate path of Frodo. In the story, Frodo encounters obstacles on the path to mount doom. He is not able to go near Isengard or through the Black Gate. In Dijkstra’s, this is seen in that those edges have higher costs, so a different path is chosen than the shortest, most direct path. Dijkstra’s minimizes the weight of the path, rather than the length.

The main function returns zero, much like Frodo returns the O shaped ring to the fire. :)