Justin-Anthony Joco 0028483667
ECE 368

Programming Assignment 6 Report

The purpose of this program was to minimize the amount of turns needed for one person to cross one side of a river to the other. My implementation might have been more complicated than it needed to be since I used three data structures: Graph, AdjNode, and AdjListHead. The entire idea was using adjacency lists to implement a graph in order to perform Dijkstra's algorithm.

For file I/O, after reading the number of pole rows and pole columns, I stored each succeeding character vertex (except '\n') as a head of its own adjacency list into a matrix, whose indices are equal to the vertex coordinates found in the input file. Thus, for an input file with N pole rows and M pole columns, I created a graph matrix of size N-1 by M elements, and each element was one adjacency list.

To create the edges, I filled in each list. For every node in the matrix, I checked if any of its neighbors existed. If a neighbor exists, then I append that neighbor into the node's adjacency list. Hence, I checked all eight directions around the node; as a result, nodes at the matrix's corners and borders lacked some neighbors.

Let's say the file had V = (N-1)*M total characters and E edges. My time complexity for creating the data structure would be about O(V + E) because I create the lists for each vertex. Then, I add the edges to each list.

While I was appending each adjacency list, I calculated the number of turns (weight) for a node to traverse to a neighboring node. If a plank existed up or down, the weight of the edge would be 0. If a plank existed on an a corner or on the left or right, the weight of that particular edge would be 1. Otherwise, for each case, the weight of the edge would be 2.

After creating the edge graph, I conducted Dijkstra's algorithm for each starting side coordinate in order to get the shortest path for that starting point. When extracting the minimum node, I did not implement a priority queue. Rather, I searched for the minimum node in linear time and kept track of the min nodes I already used. To extract_min V times, my time complexity was O(V). Thus, my calculation time complexity was about O(V*logV) due to repeatedly re-visiting nodes