Ron Bakrac
COP4400 – Dr. Hendrix
11/28/2018

Project 3 questions

I made every tile be its own vertex (in the 7x7 example, the tiles would be vertex 0-48) and created a directed graph of these vertices. I made a directed graph because you can travel from one tile to another but you can't necessarily travel back (unless you land on the same number). I first stored the input matrix, then created an adjacency list of the out degrees. I opted to create an adjacency list instead of matrix because of the better space complexity, since each tile will at most have 4 neighbors. To create the adjacency list, I created an array of all the vertices (array of 0-48 in project example). The elements of this array were arrays that took in an ordered pair of the vertex and the direction traveled to get to that vertex. For the ordered pair in C++, I had a struct that had members int num (which vertex) and char direction (direction traveled from). So my adjacency list was a vector<vector<node*>>. I iterated through all the vertices of input matrix, checking to see if the vertex to the N E S and W were adjacent by simply adding the tile value and checking if it was a valid location. If it was a valid location, I created a new node* where num was the vertex number of that valid location and direction was the direction traveled to get that tile.

I created my own breadth first search that outputs the predecessor array of the vertices. I opted for bfs because it would give me the shortest possible route to my exit, if one exists. Dfs will give me a solution but it may not be the shortest path to the exit. My bfs works like any normal bfs, where I enqueue the starting vertex, created a while loop that repeats until the queue is empty, within the loop dequeue from the queue then add all the adjacent elements of that removed item to the queue. The predecessor array ensures vertices aren't repeatedly added and keeps track of the vertex it came from. If I encounter the exit vertex in the queue then I save this exit vertex (otherwise I wouldn't know which direction to travel from the 2nd last node to exit) and I terminate bfs early returning the predecessor array. Printing the predecessor path, I don't want to print the vertex order you have to traverse, instead I must print the direction traveled to get to each vertex. This function requires the starting vertex and the exit vertex. The start vertex should have no direction and a num being 0 (assuming the first tile is always my start, can be changed). I simply output the direction of each node (the char direction member in node) traversing the predecessor array, until I get to start.