UNIVERSITY OF SOUTH FLORIDA

Project 3
GRAPH MODELING AND GRAPH ALGORITHMS

Huong Duong

COT 4400 - Analysis of Algorithms

Fall 2022

Professor William Hendrix

What type of graph would you use to model the problem input and how would you construct this graph? Be specific here; we discussed a number of different types of graphs in class

We are given some input data:

- First line: The number of mazes
- Second line: the number of levels, rows, and columns in the maze *(l, r,* and *c*)
- Third line: locations of the start point
- Fourth line: locations of the endpoint
- The remaining lines: directional information for each cell in the maze. The bottom level is described first, and all rows for one level of the mazer are described before the next level appears. Each line has *c* 6-digit binary values where each bit represents whether the spider can travel in that direction (1) or not (0). The direction bits are given in the order north, east, south, west, up, and down

I decided to use an adjacency matrix to model the problem input. The first four lines will be read line by line manually. Next, an adjacency matrix size *level * row * column* will be used to store the input's 6-bit direction (the remaining lines). In this graph, a vertex will store the position of a level, row, and column of itself, boolean discovered and 6-bit binary directional value. An edge will be an undirected path (symmetric) between two vertices which are the two locations where the spider can travel based on the directional information (when the direction binary is 1). By using an adjacency matrix, the program will look at the 6-bit binary directional value of a vertex to indicate its neighbors if the value is 1. I will store the 6-bit binary as a string. Then, I can check the value at each position of the binary value, if the direction_value[position] = 1, the neighbor will be found.

What algorithm will you use to solve the problem? Be sure to describe not just the general algorithm you will use, but how you will identify the sequence of moves the spider must take in order to reach the goal.

I will choose the Breadth-first search (BFS) strategy as the main method for my graph traversal. I choose BFS instead of Depth-first search (DFS) because after analyzing the given input and sample output, I recognized that the spider only traveled to one position one time, which means that the maximum visit for a n^{th} *level* - m^{th} *row* - k^{th} *column* is one (because we will skip the discovered vertex). Besides, the graph is unweighted and the goal is to find the shortest length of a path between two vertices so BFS will be the better option. Therefore, the program will be implemented by using BFS with an adjacency matrix. Queue will be used to implement the BFS. Besides, I want to when a path can not find the goal position, we will use recursion to go to the different directions to find different paths until we reach the goal (in case one vertex has a lot of directions that it can go).

To identify the sequence of moves the spider must take in order to reach the goal, I will create a queue to keep track of BFS. I will mark all vertices as undiscovered and add the start point vertex (v_0) to the queue. Then, I marked the v_0 as discovered. Next, I will find all of the neighbors of v_0 that are undiscovered to enqueue and mark them discovered, and deque the vertex v_0. With the newly discovered vertices/vertex, we will repeat the whole process to find the other undiscovered vertices/vertex until we reach the goal/endpoint.

Pseudocode:

input:

l, r, c: the number of levels, rows, and columns in the maze *(l, r,* and *c*)
sl. sr, sc: locations of the start point (start level. start row, start column)
el, er. ec: locations of the goal (goal level, goal row, goal column)
6-digit binary values to directional information for each cell in the maze (north, east, south, west, up, and down)

output:

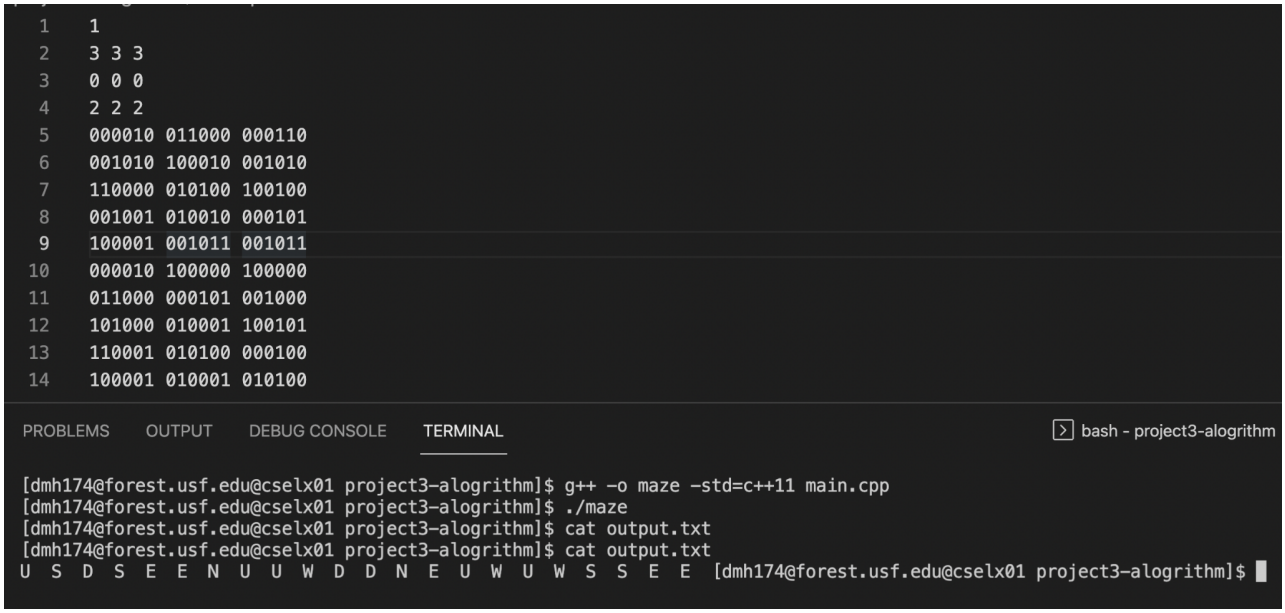
the path from the start and the goal

algorithm: maze_graph

```
Create a queue for BFS
Read the data from the input file in a 3D matrix
Mark all vertices are undiscovered
Create a 3D boolean matrix to store the discovered value. Initialize it to undiscovered
Enqueue() the start location sl, sr, sc to the queue

while queue is not empty:
    Create a temp variable to store the information of the first element in the queue
    Mark the vertex as discovered
    Dequeue();
    if the current vertex has the same locations as goal point (el, er, ec):
        end program;
    else:
        Use temp variable to find all the neighbors of the vertex:
            if the neighbor is not visited:
                add the neighbor vertex to the queue;
                print the path;
return 0;
```

Result and Discussion



However, I had no experience with implementing a 3D array into a function so even though I tried my best I can not make a separate function for the recursion call. I tried to use an adjacency list instead of the adjacency matrix and I encountered trouble with storing the input to the list. Therefore, my code is only working for the small maze with a simple direction and giving me the correct result. On the other hand, for the big maze in the example, my code still works and finds the path but it also prints the neighbors of a vertex (which had a lot of neighbors) which makes the path look not correct but I still can not find a way to fix it. My code is working great on the C4 machine lab.