Project 4: Maze

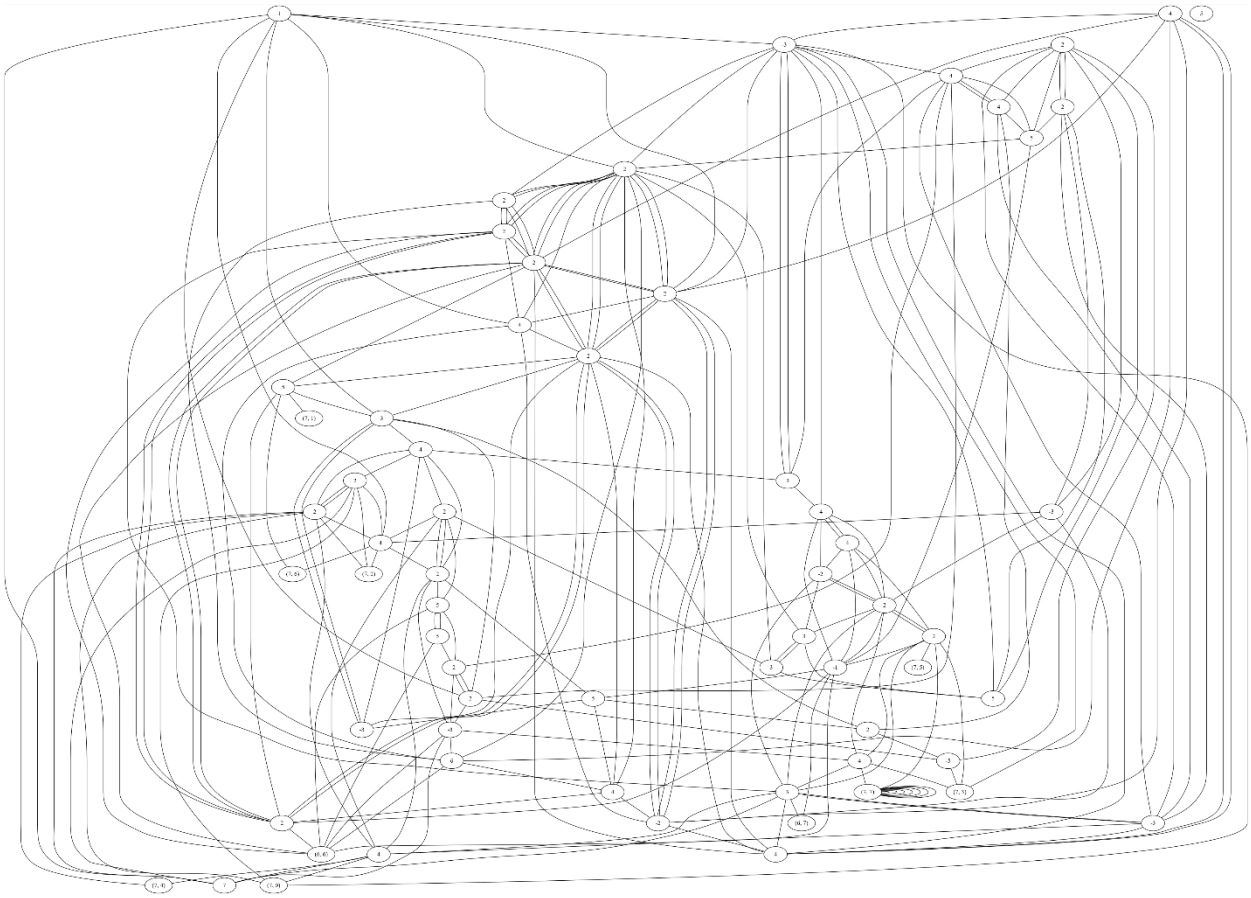
MAZE 15

Jimmy Truong

Colorado School of Mines

CSCI 406: Algorithms

Problem Modeling

1. Modeling Explanation I modeled the graph by using the cells of the maze as vertexes and the possible moves are undirected edges. In the code I created an adjacency list using a Python dictionary with the keys as vertexes and a 2D list with the 0 index as the possible horizontal moves and 1 index with possible diagonal moves.
2. 
3. I used DFS to solve the problem.
4. This algorithm will actually solve this problem because I created an adjacency list of the whole graph. Afterwards, I created a dictionary that contains parents of nodes that get us there the fastest. I then pre-order traverse the dictionary to get the path.

Code Submission

```
from pprint import pprint
import graphviz

with(open("input.txt", "r")) as f:
    maze = [[int(num) for num in line.split(' ')] for line in f]

rows = maze[0][0]
cols = maze[0][1]
maze = maze[1:]
adj_list = {}
# print(rows, cols)
# Create adjacency list for positive cells
for i in range(rows):
    for j in range(cols):
        if (i, j) not in adj_list:
            adj_list[(i, j)] = [[], []]
        value = maze[i][j]
        value = abs(value)
        # North East
        ne = (i - value, j + value)
        # South East
        se = (i + value, j + value)
        # South West
        sw = (i + value, j - value)
        # North West
        nw = (i - value, j - value)
        if ne[0] >= 0 and ne[1] < cols:
            adj_list[(i, j)][1].append(ne)
        if se[0] < rows and se[1] < cols:
            adj_list[(i, j)][1].append(se)
        if sw[0] < rows and sw[1] >= 0:
            adj_list[(i, j)][1].append(sw)
        if nw[0] >= 0 and nw[1] >= 0:
            adj_list[(i, j)][1].append(nw)
        up = (i - value, j)
        down = (i + value, j)
        left = (i, j - value)
        right = (i, j + value)
        if up[0] >= 0:
            adj_list[(i, j)][0].append(up)
        if down[0] < rows:
            adj_list[(i, j)][0].append(down)
```

```
        if left[1] >= 0:
            adj_list[(i, j)][0].append(left)
        if right[1] < cols:
            adj_list[(i, j)][0].append(right)
# pprint(adj_list)

# Set of negative points in the maze
diagPoints = set()
for i in range(rows):
    for j in range(cols):
        if maze[i][j] < 0:
            diagPoints.add((i, j))
# Depth First Search the adjacency list
# adj_list[(i, j)][0] = list of nodes that can be reached horizontally
and vertically from (i, j)
# path = list of nodes that form the path from start to end
# https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
def DFSUtil(vertex, visited, diagonal=0, diagonal_visited=set(),
parent=None, caller=None):
    if not diagonal:
        visited.add(vertex)
    else:
        diagonal_visited.add(vertex)
    if maze[vertex[0]][vertex[1]] < 0:
        if diagonal == 0:
            diagonal = 1
        else:
            diagonal = 0
    for v in adj_list[vertex][diagonal]:
        if diagonal == 1:
            if v not in diagonal_visited:
                if v not in parent:
                    parent[v] = {vertex}
                else:
                    parent[v].add(vertex)
                DFSUtil(v, visited, diagonal, diagonal_visited,
parent,vertex)
        else:
            if v not in visited:
                if v not in parent:
                    parent[v] = {vertex}
                else:
                    parent[v].add(vertex)
                DFSUtil(v, visited, diagonal, diagonal_visited,
parent,vertex)
```

```
def dfs(vertex):
    visited = set()
    diagonal_visited = set()
    parent = {}
    DFSUtil(vertex, visited, 0, diagonal_visited, parent)
    # pprint(parent)
    curr = (7,7)
    path = [curr]
    diagonal = False
    # Trace back the path from end to start
    while curr != (0,0) or (curr == (0, 0) and diagonal):
        if curr in parent:
            if curr in diagPoints:
                diagonal = not diagonal
            for p in parent[curr]:
                if not diagonal:
                    if curr in adj_list[p][0]:
                        path.append(p)
                        curr = p
                        break
                else:
                    if curr in adj_list[p][1]:
                        path.append(p)
                        curr = p
                        break
    path.reverse()
    return path
mypath = dfs((0, 0))

# reverse each element in the list and add 1
for x in mypath:
    x = (x[1] + 1, x[0] + 1)
    print(x, end=" ")
# Creates a graphical representation of the maze using graphviz
dot = graphviz.Digraph(comment='Maze')
for i in adj_list:
    dot.node(str(i), str(maze[i[0]][i[1]]))
    for j in adj_list[i][0]:
        dot.edge(str(i), str(j), dir='none')
    for j in adj_list[i][1]:
        dot.edge(str(i), str(j), dir='none')
# print(dot.source)
```

Results

```
(1, 1) (1, 5) (4, 8) (8, 4) (5, 1) (1, 5) (1, 2) (4, 2) (4, 6)
(8, 6) (7, 7) (1, 1) (5, 5) (7, 3) (2, 8) (4, 8) (8, 8)
```