UNIVERSITY OF SOUTH FLORIDA

Project 3: Graph modeling and graph algorithms

Jumping Jim maze problem

Analysis of Algorithms (COT4400)

John Cameron

November 28, 2018

1 MODELING THE PROBLEM

Below is a summary of the type of graph and the data structure used to represent it.

- Directed - Because jumps are only one-way because the destination space may have a different amount of jumps permitted than the source space.
- Unweighted - Because all jumps are equivalent regardless of the number of jumps permitted for each space.
- Vertex Labeled - In order for each node to represent a position in the matrix.
- Edge Labeled - In order to identify the direction each edge is representing.
- Adjacency List - Because the graph is dense on average.

1.1 GRAPH DATA STRUCTURE

For this problem, there can be at most four outgoing edges per vertex because there are only four possible directions: north, south, east, and west. Based on the Jumping Jim matrix from the problem, the average number of outgoing degrees per vertex was approximately 2.27. From a lower bound of 0 to an upper bound of 4, the value 2.27 is closer to the upper bound. Therefore, the graph is considered dense. Because of this, an adjacency list would be the best suited data structure. See Figure 1.1 for a sample representation.

(0,0)	→	S: (0,3)	E: (3,0)
(0,1)	→	S: (0,3)	E: (2,1)
(0,4)	→	N: (0,0)	E: (4,4)
(2,2)	→	S: (2,6)	E: (6,2)
(0,5)	→	N: (0,1)	E: (4,5)

Figure 1.1: Sample Adjacency List Data Structure

1.2 Component Representation

The model that was decided for this problem was to have the vertices be the positions i and j in the matrix and the edges be labeled with the direction of the jump. Figure 1.2 on page 5 is a graph visualization that models the Jumping Jim matrix from the problem.

1.3 Algorithm

The graph algorithm that will be used to find the path is Dijkstra's Algorithm. To determine the directions Jim would take, it would record each edge traversed that leads to the goal. Each edge in the graph must be labeled with a direction (e.g. north, south, east, or west).

Algorithm 1: Builds the graph given a 2D matrix

Input: *rows* by *columns* integer matrix
Output: Graph representation of the matrix

```
Function BuildGraph(matrix, rows, columns):
    graph = DirectedUnweightedGraph()
    for i = 0 to rows do
        for j = 0 to columns do
            spaces = matrix[i][j]
            if spaces = 0 then
                continue
            end
            for d ∈ N, S, E, W do
                target_i = i
                target_j = j
                switch d do
                    case N do
                        target_j = target_j − spaces
                    end
                    case S do
                        target_j = target_j + spaces
                    end
                    case E do
                        target_i = target_i + spaces
                    end
                    case W do
                        target_i = target_i − spaces
                    end
                end
                if target_i and target_j is within the matrix bounds then
                    Add vertex (i, j) to graph if not present
                    Add vertex (target_i, target_j) to graph if not present
                    Add directed edge with label d from (i, j) to (target_i, target_j) in graph
                end
            end
        end
    end
```

Algorithm 2: Finds a solution path for the Jumping Jim problem given a graph representation

Input: $graph$

Output: A solution path for the Jumping Jim problem

```
Function jumpPath(graph):
    start = vertex representing the top left of matrix
    end = vertex representing the bottom right of matrix
    path = Dijkstra's Algorithm on graph with start vertex start and goal vertex end
    return The traversed edges in path
```

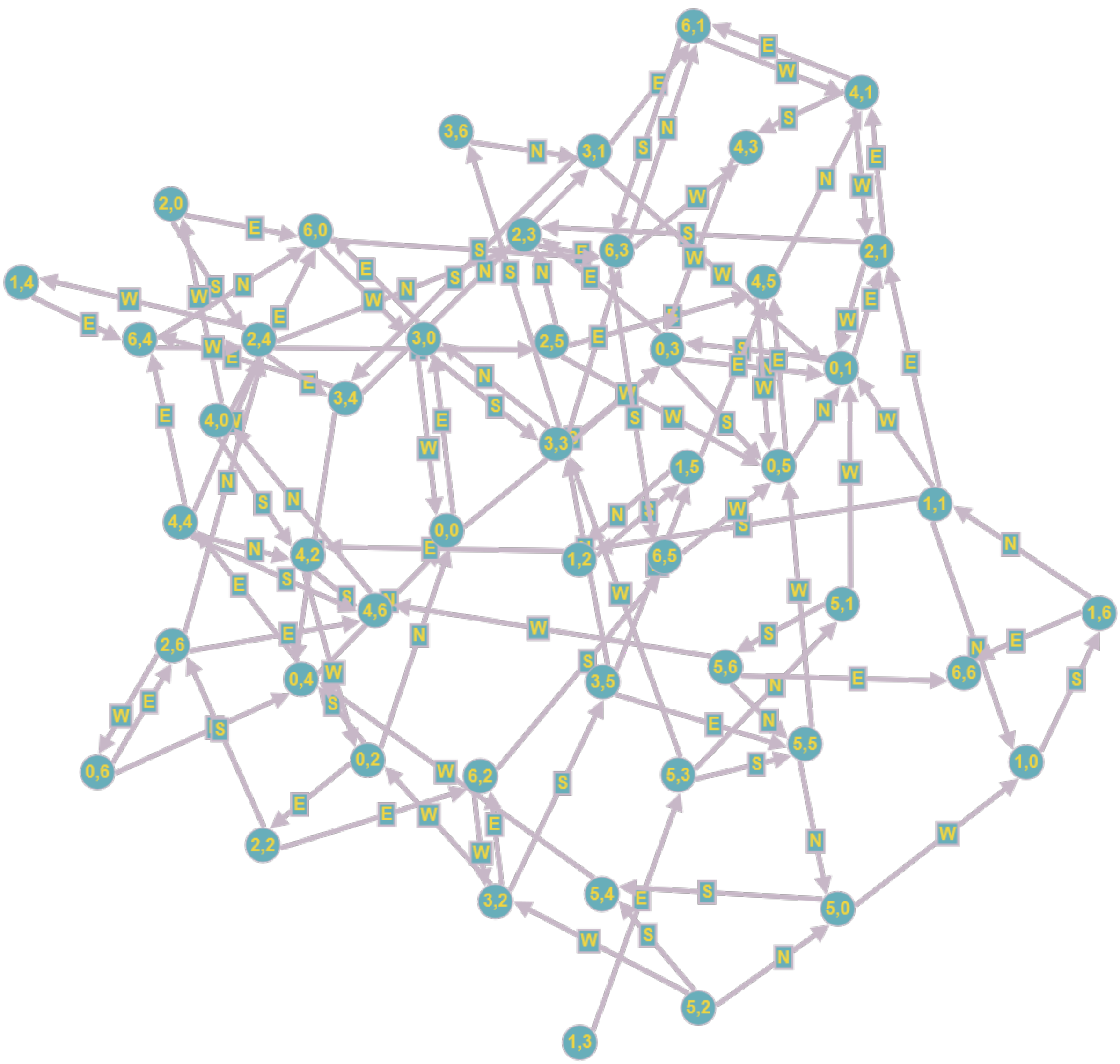

Figure 1.2: Visualization of the graph model