Algorithms Graph
Modeling Project

A detailed description of the maze problem including a figure is provided separately[1]. You may want to look at that first to get a sense of what this project involves! Your assignment is to (1) model the maze problem as a graph (2) use an appropriate graph algorithm that we've encountered in class to solve the problem (3) code the algorithm and (4) provide the output on an example input. **Friendly reminder: This is an individual assignment. The departmental collaboration policy (the document you signed at the start of the semester) will be enforced.**

1 Report

1.1 Problem Modeling[40 pts]

- [10] Explain how you modeled the problem as a graph (typically, a few sentences)

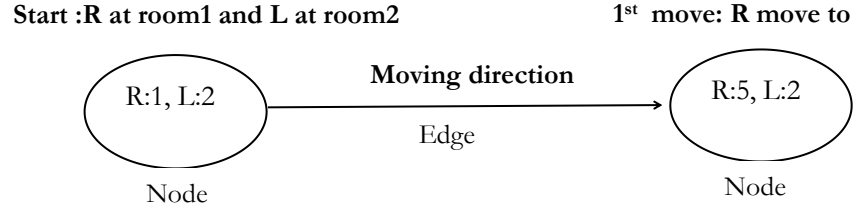
Based on the original graph provided by the instructor whose format is .in file, **we create another graph named resulting graph to model this maze problem.**

Resulting graph:

- **Each node presents the current state of both Rocket and Lucky** where their locations/room numbers are saved in the node.
- **Each edge denotes the connections between two different states.** Given two states/nodes A and B, and we know that A and B contains two different combinations of position information for Rocket and Lucky.
- **By utilizing the color information of edges and nodes embedded in the original graph, if the state of A can be changed to the state of B by moving either Rocket or Lucky,** we can build the edge between A and B, and its direction is from A to B.
- **Using recursive manner to search and obtainall all moving paths**/all possible moving directions on the resulting graph for both Rocket and Lucky.

Example:

Since the start positions for Rocket and Lucky are 1 and 2, respectively. So, the first node in the resulting graph is the upper one (R:1, L:2) where "R" denotes Rocket and L represents "Lucky". Then we hold L and try to move R. Fortunately, we notice that 1 is connected to 5. The color of the edge between 1 and 5 and the color of node 2 are all Red. That means Rocket can be moved from 1 to 5 when Lucky is in 2. Thus, we will have another node (R:5, L:2) and an edge between these two nodes. During the procedure of generating such a resulting graph, we would like to obtain all moving paths/all possible moving directions for both Rocket and Lucky, and its searching strategy is built in a recursive manner.



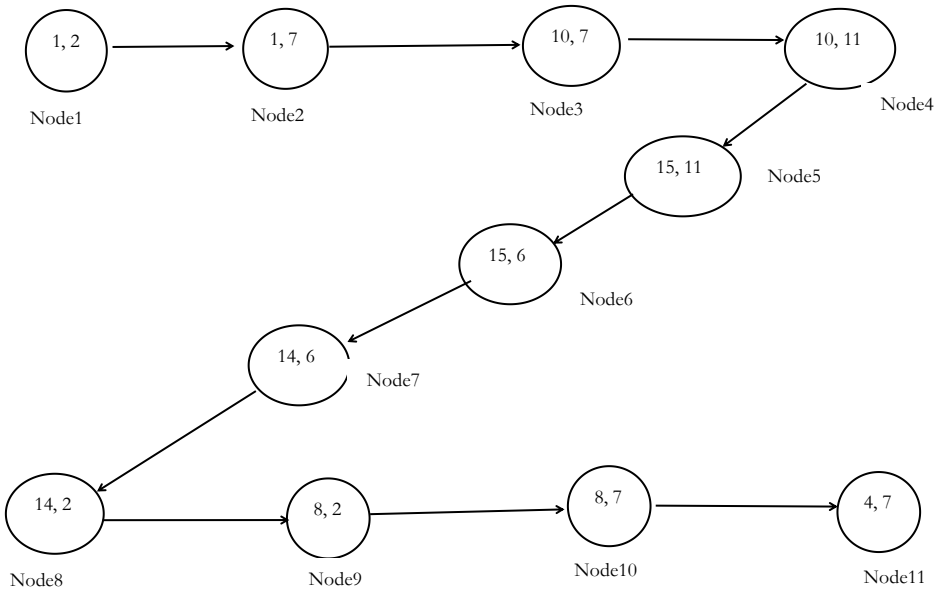
A example path in the resulting graph

- [10] Draw enough of the resulting graph to convince us that you have modeled the graph correctly (you don't have to draw the whole graph).

Following the color rulues of nodes and edges, partial moving steps for Rocket or Lucky as below:

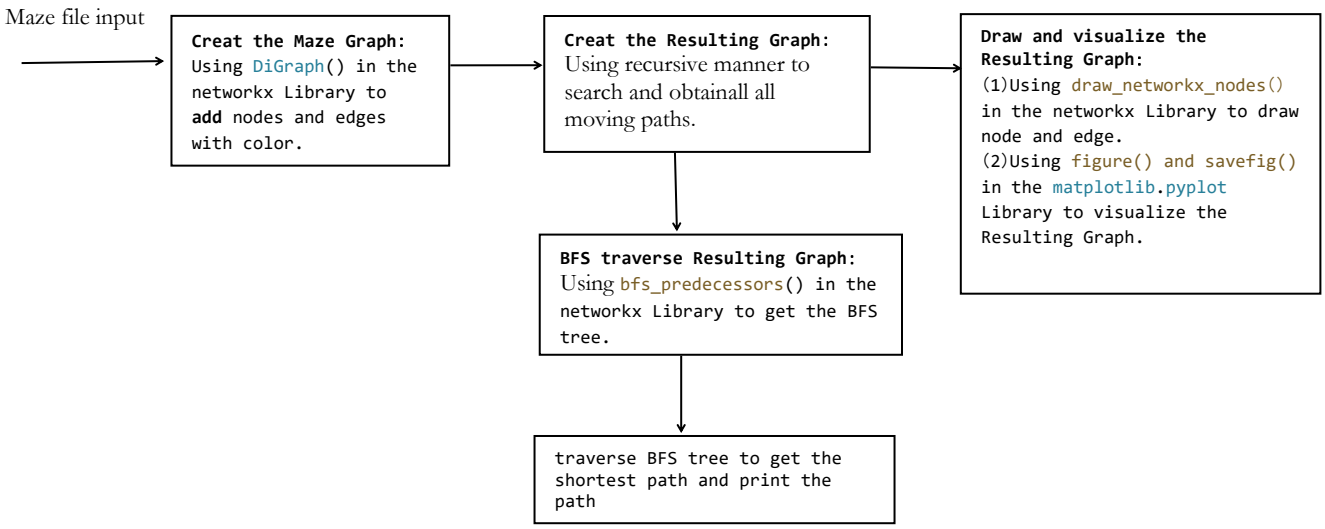
Node1 :R at room1 and L at room2 (Start Location)
Node2: R stay at room1 and L move to room7
Node3: R move to room10 and L stay at room7
Node4: R stay at room10 and L move to room11
Node5: R move to room15 and L stay at room11
Node6: R stay at room15 and L move to room6
Node7: R move to room14s and L stay at room6
Node8: R stay at room14 and L move to room2
Node9: R move to room8 and L stay at room2
Node10: R stay at room8 and L move to room7
Node11: R move to room4 and L stay at room7

The Space Wreck resulting graph for above moving steps as below:



- [10] Identify the graph algorithm needed to solve the problem (one sentence).

We first build the original graph based on the .in file provided by the instructor. And then create the resulting graph described above. **For this resulting graph, we use BFS to obtain the shortest path between the final state to the start state.**



- [10] Convince us that this algorithm will actually solve the problem. This means that it will find the *shortest path* if a path exists (typically, a few sentences, depending on your approach).

First, based on the aforementioned statement for the first question, **the resulting graph is an unweighted directed graph.** Specifically**, the edges of resulting graph represent moving directions** for either Rocket or Lucky. And we don't need to assign weights for them as they all share the same weights. **So BFS is an appropriate algorithm to find the shortest path for unweighted directed graph if a path exists.**

1.2 Code Submission [25 pts]

Please include your code in the report. 10 points will be allocated for quality and readability of the code. The code should read in the provided input file and print the output.

1.3 Extra Credit [5 pts]

You will receive a bonus of up to 5 points if you can show that you have done something above and beyond what was asked. Possible examples include but are not limited to (1) using an algorithmic library (e.g., Boost in C++) to implement the graph functionality (2) augmenting your algorithm with a visualization

(1) Using an algorithmic library to implement the graph functionality

I use the networkx Library in python to implemen the graph functionality.

- Using DiGraph() in the networkx Library to add node and edge with color for the Maze Graph.
- Using draw_networkx_nodes（） in the networkx Library to draw node and edge for the Resulting Graph.
- Using bfs_predecessors() in the networkx Library to traverse the BFS tree for the Resulting Graph.

(2) Augmenting your algorithm with a visualization.

I use two Function Draw and visualize the Resulting Graph:

- Using **draw_networkx_nodes()in the networkx Library** to draw nodes and edges**.**
- Using **plt.figure() and plt.show()** to display the Resulting Graph..

Space Wreck Resulting Graph