MAZE GENERATOR AND SOLVER

INTRODUCTION

This project is centred around the implementation and analysis of maze generation and solving algorithms, leveraging the principles of Graph Theory. The algorithms featured in this project include binary tree maze generation, Prim's algorithm, left wall follower maze solver, Dijkstra's algorithm, and depth-first search (DFS) maze solver.

PREVIOUS LITERATURE

Numerous books have examined the topic of creating and solving mazes, including Jamis Buck's "Mazes for Programmers", which offers helpful insights about coding maze algorithms. Dan Gusfield's book "Algorithms on Strings, Trees, and Sequences" provides basic ideas related to maze algorithms. Understanding geometric problem-solving is provided by "Computational Geometry: Algorithms and Applications". Hermann Kern's book "Labyrinth: A Comprehensive Survey of the Maze" examines the cultural and historical dimensions of mazes. Furthermore, conference proceedings and scholarly periodicals support current research. Even while these publications may not be solely algorithmic in nature, taken as a whole, they provide a thorough grasp of a variety of maze-related subjects.

TOPIC OF STUDY

I. MAZE GENERATION ALGORITHMS.

1. **Binary Tree Maze Generation** creates mazes through a straightforward process. The algorithm operates on a grid of cells, initializing each with information about carved paths and visitation status. Starting with a random cell, the algorithm iterates through each cell, choosing a random direction (North, East, South, or West). If moving in that direction is valid, a passage is carved between the current cell and the neighbouring one. Backtracking is employed when a cell has no valid neighbours, ensuring all cells are visited, resulting in a fully connected maze. The algorithm uses a stack to keep track of the current cell being processed, and the grid structure maintains information about the maze's carved paths and visited cells.

2. **Prim's Algorithm** is a maze generation technique that creates mazes by randomly selecting walls and carving passages between cells. The algorithm ensures a fully connected maze, where every cell is reachable from every other cell. Notably, The process begins with the initialization of a grid of cells, each representing a part of the maze. A set keeps track of visited cells, and a list is used for processing cells. The algorithm starts by selecting a random cell as the starting point and adds neighbouring walls to the cell list. Subsequently, a random wall is chosen, and a passage is carved through it to the neighbouring cell. The neighbouring walls of the new cell are added to the list if they have not been visited. This process repeats until the list is empty, resulting in a fully connected maze with passages carved in a manner that avoids loops.

II. MAZE SOLVING ALGORITHMS.

1. **Left Wall Follower Maze Solver** follows a strategy of always keeping the left side of the wall to its left. Starting at the entrance, the solver makes decisions based on the presence of walls to its left, front, and right. If there is an open path to the left, the solver turns left; if there is no left path but an open path in front, it moves forward. The solver turns right if there is neither a left nor front path and reverses direction if no paths are open. This continues until the solver reaches the maze's exit, and the algorithm visualizes the path taken by drawing it on the maze.

2. **Dijkstra's algorithm** for pathfinding, the program explores the maze by visiting neighbouring cells and updating the minimum distance to each cell. Backtracking from the exit to the entrance reconstructs the shortest path, visualized by drawing it on the maze. The algorithm employs data structures such as a priority queue (unvisited), a path reconstruction dictionary, and a visited set to achieve efficient pathfinding.

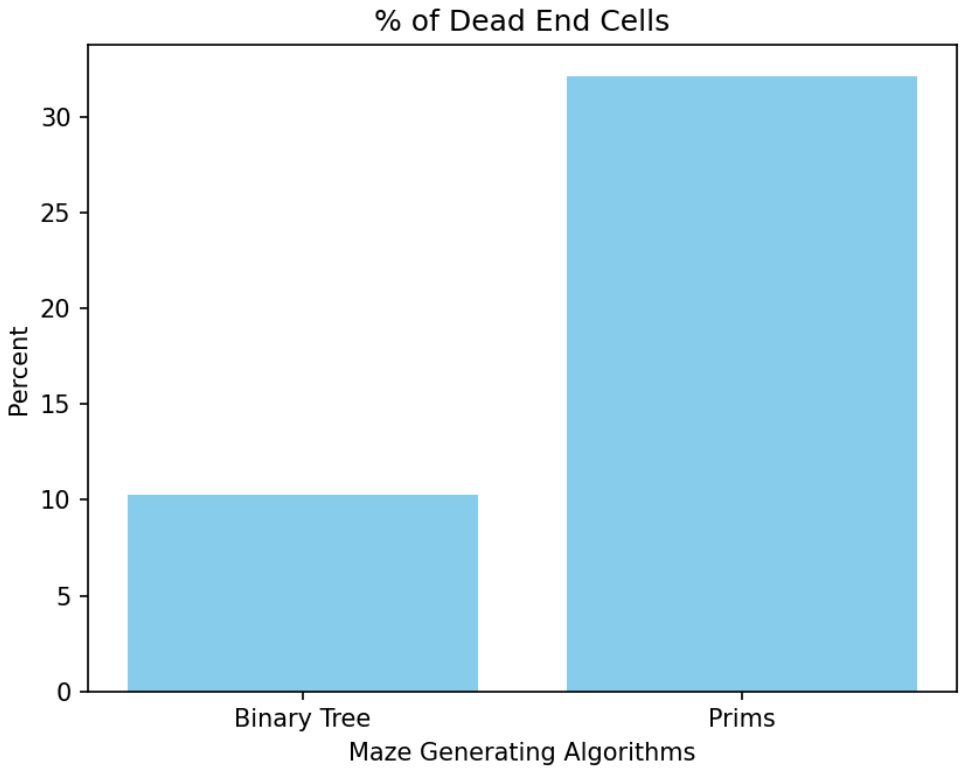
3. **Depth-First Search (DFS) Maze Solver** navigates mazes using a stack-based approach. Starting with the entrance, the algorithm identifies unvisited neighbouring cells at each step. If no unvisited neighbours are found, it backtracks to the previous cell. This process continues until the exit is reached, visualizing the

path by drawing it on the maze. The algorithm relies on a set to keep track of visited cells and an array-based stack to process cells in a specific order.

ANALYSIS

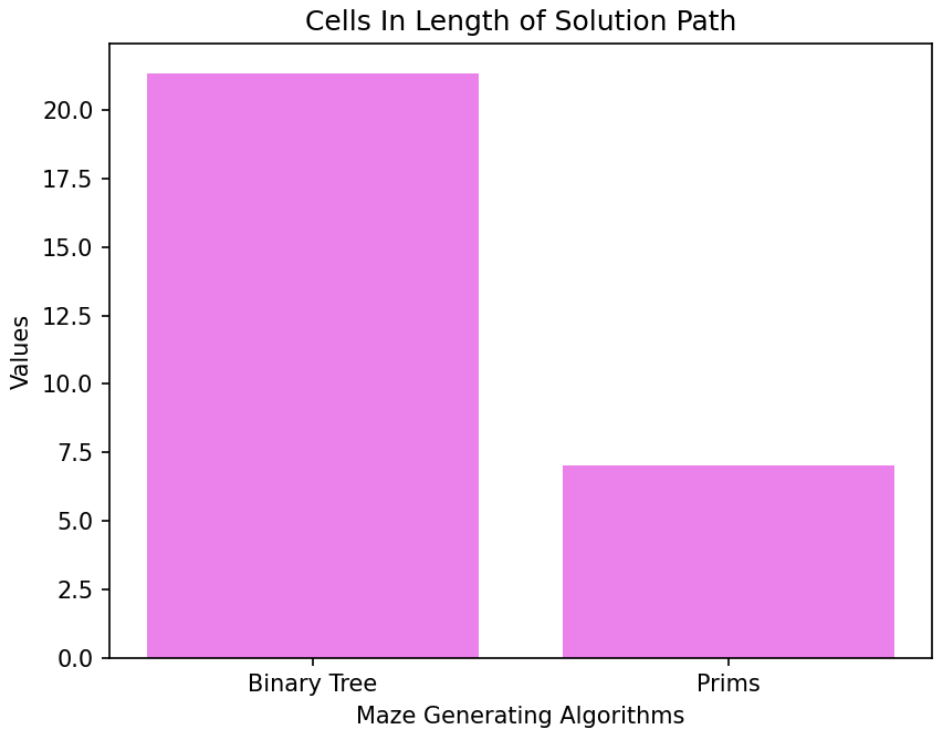
I. MAZE GENERATION ALGORITHMS.

1. DEAD-END CELLS: Prim's Algorithm produces a significantly larger number of dead-end cells (32.12% of cells in the maze have 3 walls) than the binary tree algorithm. (10.24%)



Method: Both algorithms were run 100 times on a 32x32 grid. The resulting maze was stored as a grid of cells. The summation of Each cell in the grid with 3 walls (and one link) gave the number of dead-end cells. This was then expressed as a percentage of the total cells in the maze.

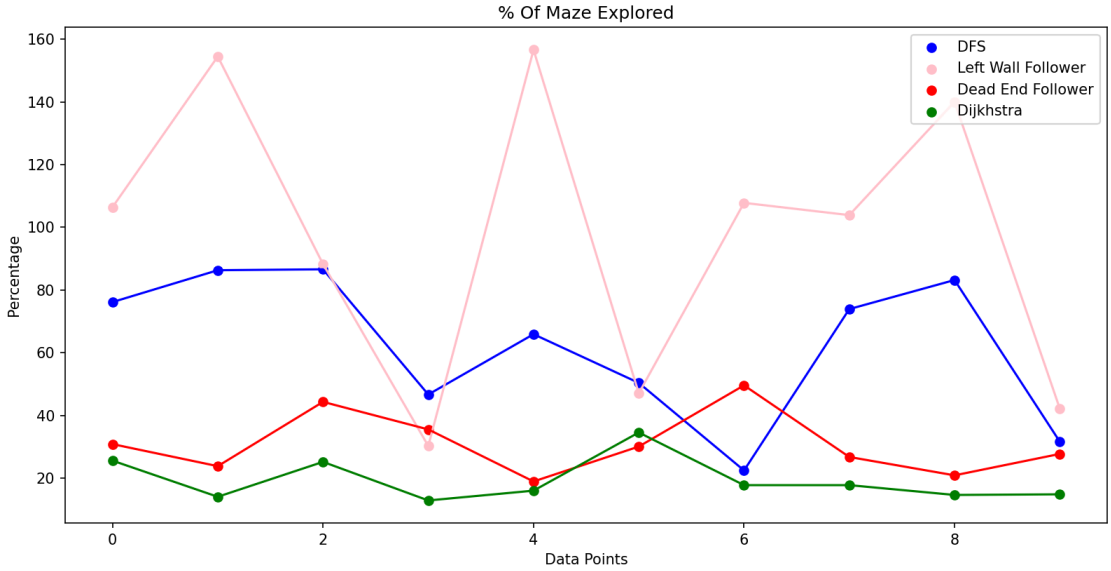
2. LENGTH OF SOLUTION PATH: As per the the number of cells in the solution path is concerned, the binary tree algorithm generates a maze with significantly more cells in the solution path as compared to Prim's Algorithm, where only 7.01% of cells comprise the solution path. The former's mean percentage is around 21.33, i.e. 1/5th of cells in the maze are part of the solution path.



Method- DFS algorithm was used to measure the number of cells in the length of the solution path of a 32x32 maze generated by Binary tree and Prim's algorithm. This was done 100 times, and the length of the solution path was expressed in terms of the percentage of maze cells in the solution path. The maze generated by both algorithms only contain a single solution; therefore, any maze-solving algorithm can be used to compare the length of the path.

II. MAZE SOLVING ALGORITHMS.

PERCENTAGE OF MAZE EXPLORED: The Left Wall Follower algorithm consistently attains a mean percentage of 99.57%, indicating its tendency to explore a substantial portion of the maze. Conversely, the Dead End and Dijkstra algorithms exhibit mean percentages of 32.70% and 19.92%, respectively, suggesting more efficient exploration with lower averages. The Dead End algorithm displays a moderately distributed exploration pattern, while Dijkstra's algorithm also showcases a relatively moderate distribution. The Depth-First Search (DFS) algorithm, with a mean of 63.99%, demonstrates a wider distribution, implying variable exploration efficiency across instances



Method: The number of cells visited while each algorithm solves a 32x32 maze was noted several times. These were expressed in terms of the percentage of maze explored before reaching the end.

CONCLUSION

In conclusion, For maze generation favouring longer solution paths with fewer turns and dead ends, Binary Tree algorithm is preferred over Prim's Algorithm. Prim's Algorithm tends to produce more complex mazes with a higher percentage of dead-end cells. This complexity, coupled with the algorithm's tendency to create shorter and more direct solution paths, makes Prim's Algorithm well-suited for scenarios demanding intricate and challenging mazes.
In maze solving, the Left Wall Follower algorithm proves to be the least efficient as it explores almost the entire maze when finding a solution. Dijkstra's Algorithm emerges as the most efficient, guaranteeing optimal paths but exploring less extensively. Depth-First Search (DFS) exhibits unpredictability with moderate exploration efficiency. The choice between algorithms hinges on the trade-off between exploration efficiency and the guarantee of optimal paths aligning with specific application requirements.

REFERENCES

1. Mazes for programmers- Jmis Buck.
2. https://weblog.jamisbuck.org/under-the-hood/
3. https://en.wikipedia.org/wiki/Maze-solving_algorithm