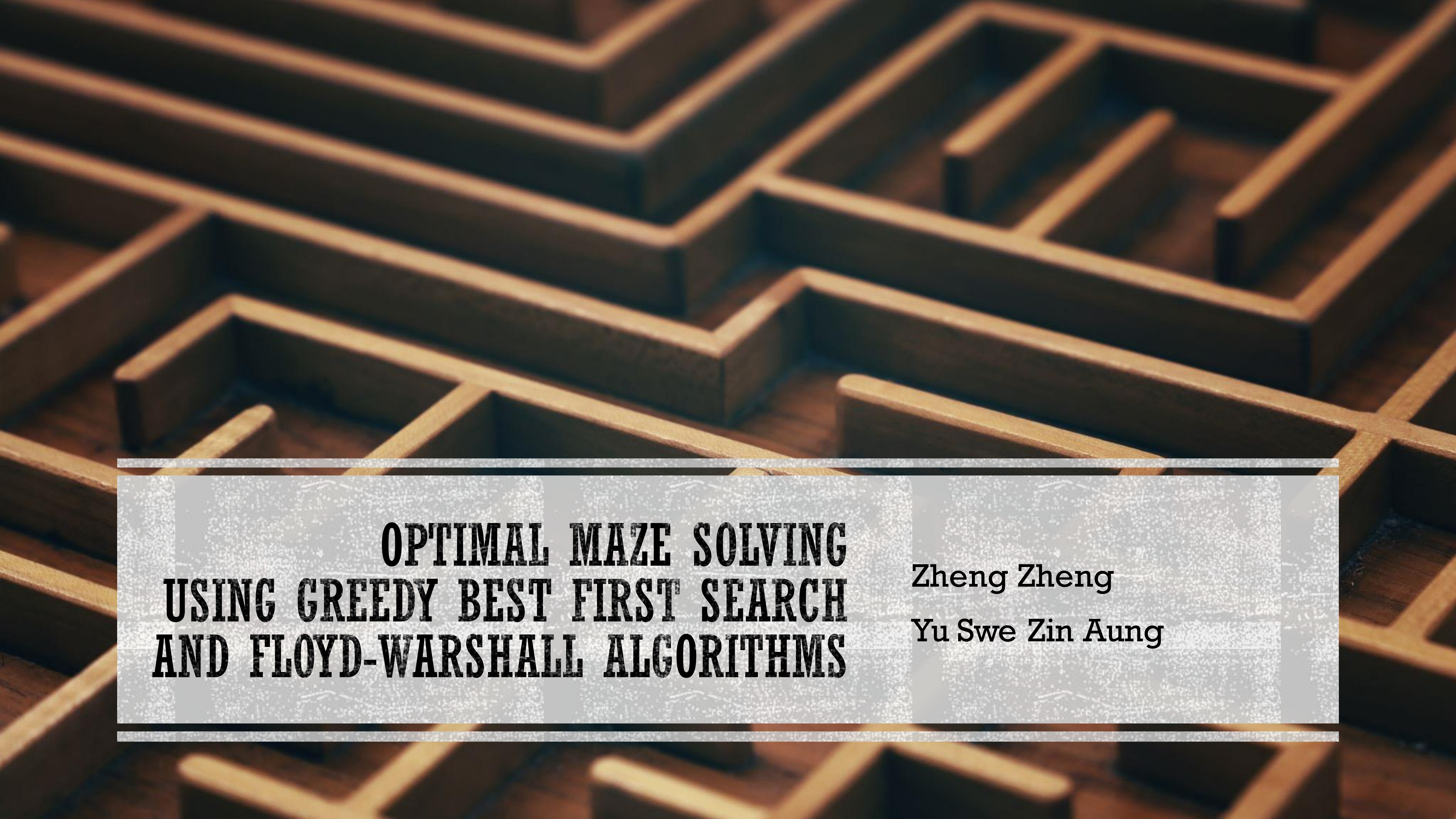
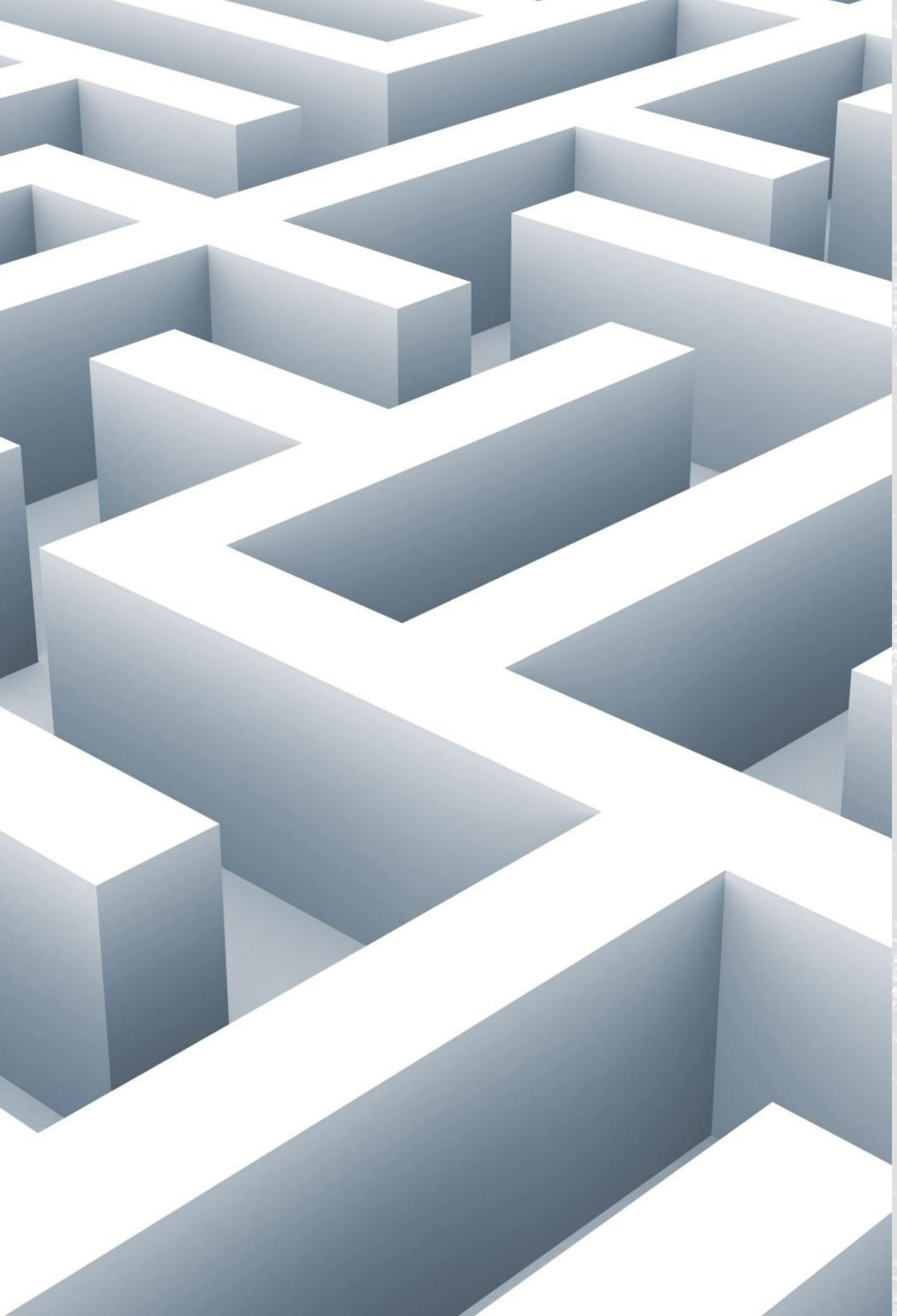


OPTIMAL MAZE SOLVING USING GREEDY BEST FIRST SEARCH AND FLOYD-WARSHALL ALGORITHMS

Zheng Zheng

Yu Swe Zin Aung

CONTENT

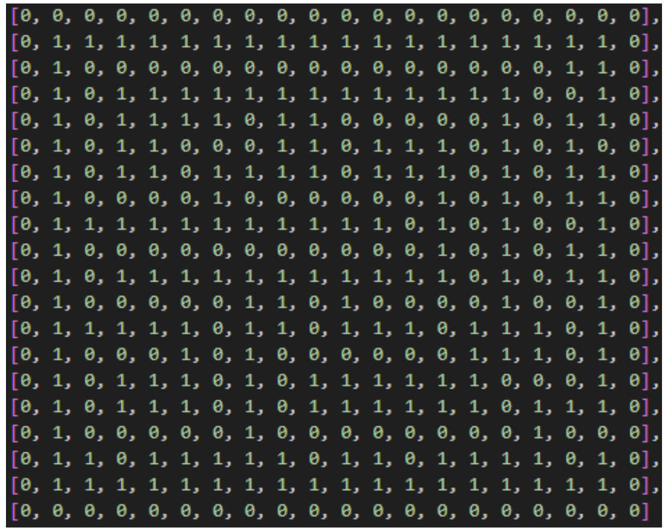


INTRODUCTION

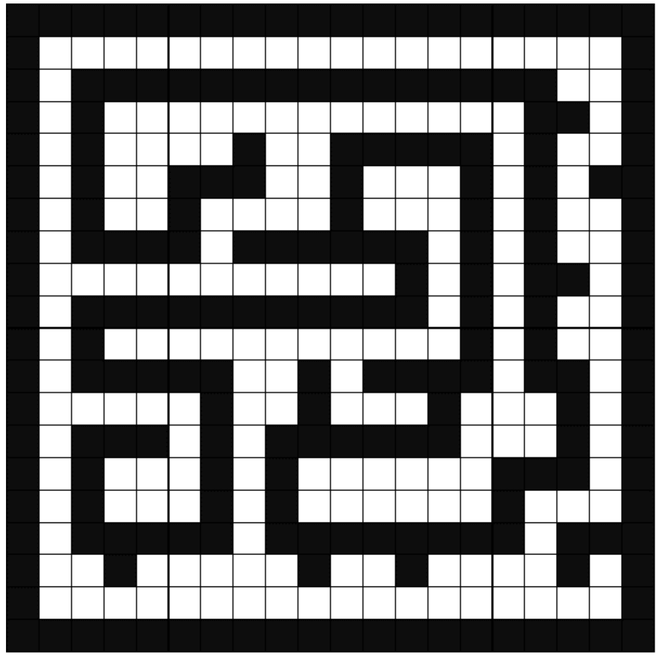
- Addressing maze-solving challenges with innovation
- Fusion of Greedy Best First Search and Floyd-Warshall algorithms
- Capitalizing on Greedy Best First Search heuristic
- Harnessing Floyd-Warshall for all-pairs shortest path
- Targeting efficient and effective maze-solving solution
- Significance of Maze Solving in Computer Science
  - Reflects real-world challenges and pathfinding domain
  - Applications in robotics, navigation systems, video games, and more
  - Mazes as a foundation for algorithm design and evaluation

OBJECTIVE

- Thorough examination of Greedy Best First Search strengths, weaknesses, and optimal applications in maze-solving.
- In-depth investigation into Floyd-Warshall algorithm's unique characteristics and its relevance to maze problems.
- Identification of performance and accuracy improvement opportunities within each algorithm.
- Development and implementation of tailored optimization methods for maze-solving scenarios.



```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
[0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0],
[0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0],
[0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0],
[0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0],
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0],
[0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0],
[0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0],
[0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0],
[0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0],
[0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0],
[0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0],
[0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```



BINARY MAZE

- Matrix Representation: The maze is a 2D matrix, where each cell can have a value of 0 (representing walls or obstacles) or 1 (representing open paths).
- Maze Image: The image matches the matrix's dimensions and visually represents the maze's layout.
- Binary Data Plotting: The binary data from the matrix determines the maze layout in the image.

IMPLEMENTING BINARY MAZE

- **Maze Representation**
  - 2D binary matrix defines maze layout.
  - Cells: Blocked (0) or Unblocked (1).
  - Blocked cells represent obstacles, unblocked cells form paths.
- **Pathfinding Algorithms**
  - Greedy Best First Search and Floyd-Warshall algorithms.
  - Finding shortest path between two points in the maze.
- **Graphical User Interface (GUI)**
  - Utilizing Pygame library.
  - Maze visualization and user interaction.
  - Setting start and end points by clicking cells.
- **Path Visualization**
  - Algorithm calculates shortest path.
  - Visualizes path as a red line connecting cells.
- **User Interaction**
  - Triggering algorithm by setting start and end points.
- **Real-time Visualization**
  - GUI updates in real-time.
  - Consistent display of pathfinding progress.
- **Termination**
  - Pygame handles event handling and window management.
  - Application can be closed or exited

GREEDY BEST FIRST SEARCH ALGORITHM

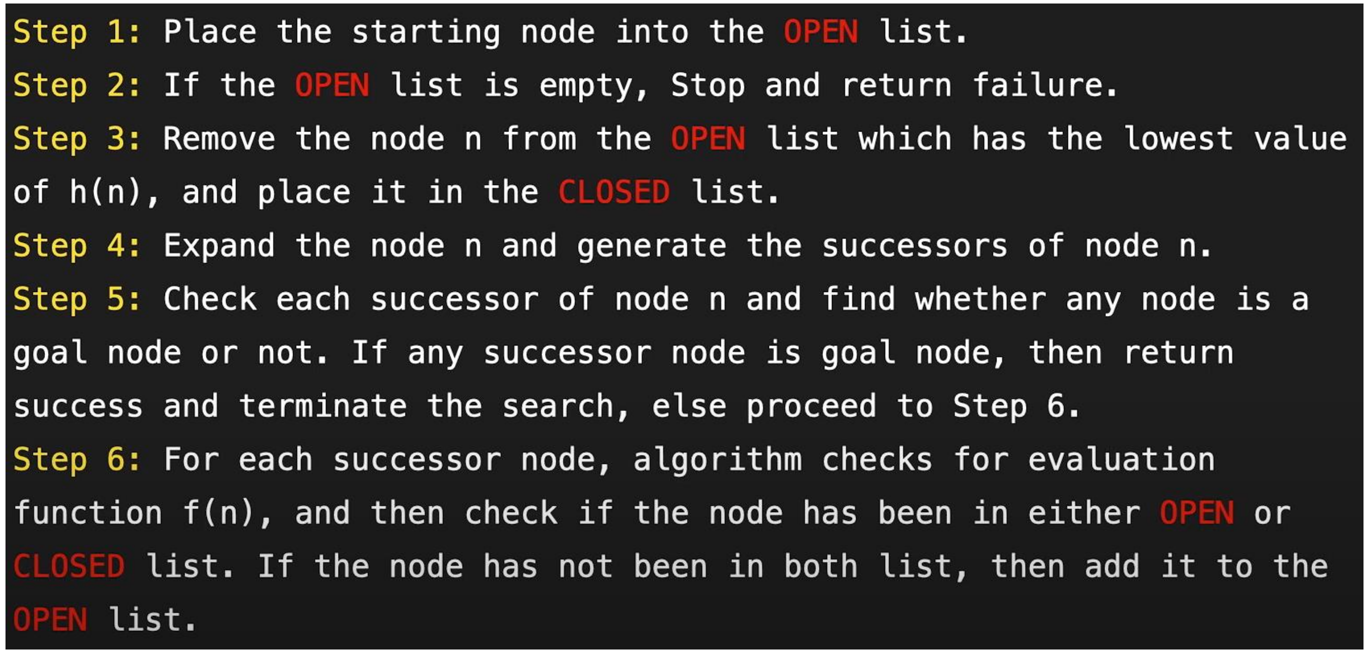
- A variant of Best First Search that uses a heuristic function to prioritize exploration based on node value
- The heuristic function, h(x), evaluates nodes based on proximity to target node, favoring lower cost options
- The greedy algorithm that prioritizes the most promising choice at each step
- **Efficiency:**
  - GBFS is more efficient than uninformed search algorithms (e.g., BFS, DFS).
  - Uses heuristic function to guide search towards promising paths.
  - Focuses on nodes closer to the goal, reducing search space.

GREEDY BEST FIRST SEARCH ALGORITHM (CONT.)

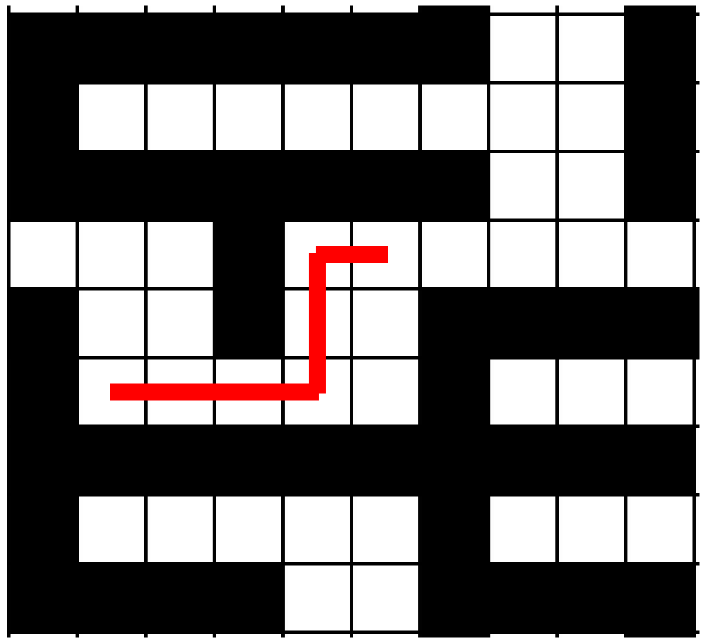
- **Greedy Approach:**
  - GBFS employs a greedy strategy in node selection.
  - Chooses node with best heuristic evaluation.
  - Can lead to faster solutions with accurate heuristic estimates.
- **Optimality under Certain Conditions:**
  - Guaranteed optimal solution when specific conditions met.
  - Heuristic function must be admissible and consistent.
  - Admissibility: Heuristic never overestimates actual goal cost.
  - Consistency: Ensures optimal search progression.

GREEDY BEST FIRST SEARCH ALGORITHM (CONT.)

- **Time Complexity: Worst case time complexity: O(bm)**
- **Space Complexity: Worst case space complexity: O(bm)**
  - b: Branching factor (average number of successors per state)
  - m: Maximum depth of the search space
- **GBFS Incompleteness:**
  - **Greedy Best-First Search is incomplete.**
  - **Even in finite state spaces, GBFS may fail to find a solution.**
  - **GBFS prioritizes nodes based on heuristic values.**
  - **If the heuristic is not well-informed, it can lead the algorithm astray.**
  - **May get stuck in loops or overlook optimal paths.**
  - **The algorithm doesn't backtrack, which can lead to incompleteness.**



```
Step 1: Place the starting node into the OPEN list.
Step 2: If the OPEN list is empty, Stop and return failure.
Step 3: Remove the node n from the OPEN list which has the lowest value
of h(n), and place it in the CLOSED list.
Step 4: Expand the node n and generate the successors of node n.
Step 5: Check each successor of node n and find whether any node is a
goal node or not. If any successor node is goal node, then return
success and terminate the search, else proceed to Step 6.
Step 6: For each successor node, algorithm checks for evaluation
function f(n), and then check if the node has been in either OPEN or
CLOSED list. If the node has not been in both list, then add it to the
OPEN list.
```



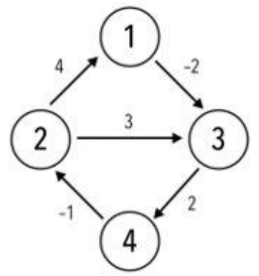
PSEUDOCODE OF GREEDY BEST FIRST SEARCH

FLOYD-WARSHALL ALGORITHM – OVER VIEW

A PIVOTAL ALGORITHM IN GRAPH THEORY FOR IDENTIFYING SHORTEST PATHS WITHIN A WEIGHTED GRAPH

Problem

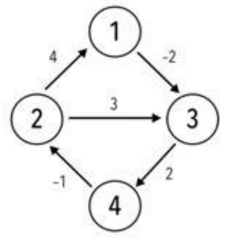
Given a directed graph G = (V,E) with vertex set V = {1, 2, ..., n}, we might wish to determine the shortest path from i to j for all vertex pairs i, j in V.

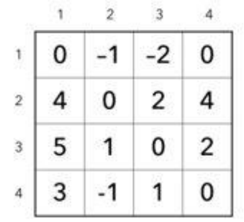


Input*

A weighted, directed graph: G = (V, E)

with a weight function that maps edges to real-valued weights: w = E -> R





	1	2	3	4
1	0	-1	-2	0
2	4	0	2	4
3	5	1	0	2
4	3	-1	1	0

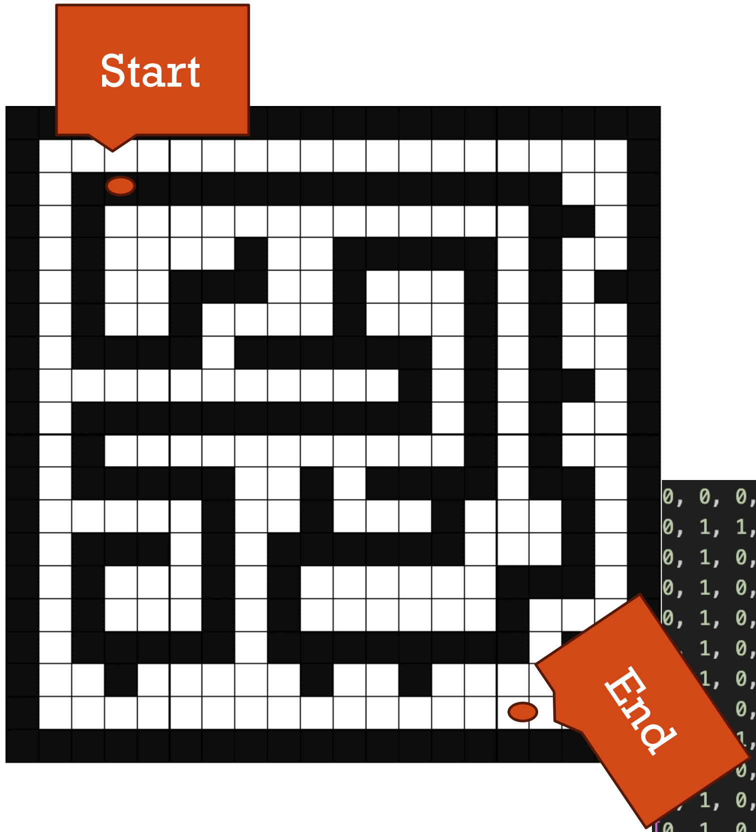
Output

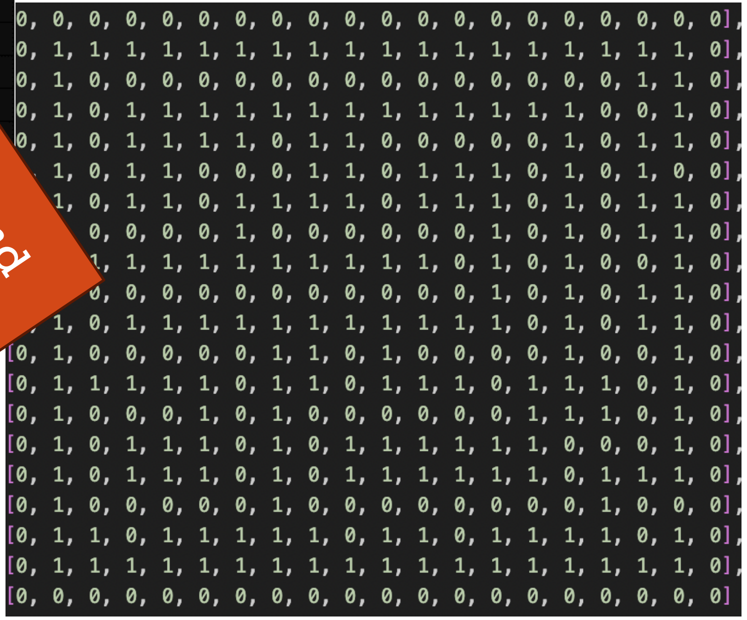
For every pair of vertices u, v in V, find a shortest (least-weight) path from u to v, where the weight of a path is the sum of the weights of its constituent edges

Tabular form

FLOYD-WARSHALL ALGORITHM – PROBLEM DEFINITION

MAZE SOLVING PROBLEM





```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0],
0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0],
  1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0],
  1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0],
     0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0],
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0],
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0],
  1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0],
[0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0],
[0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0],
[0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0],
[0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0],
[0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0],
[0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Objectives

- **The primary objective of this project is to develop an algorithm that efficiently and effectively solves a maze.**

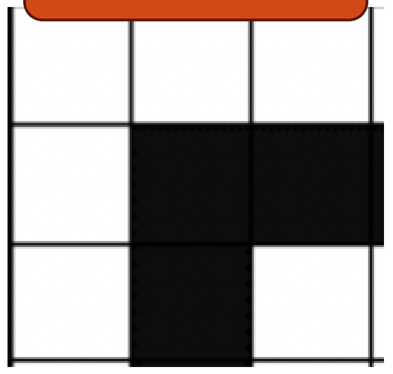
Challenges:

- **How to transfer Maze Matrix into Graph (Graph matrix with V and E)**
  - **Floyd-Warshall algorithm is mainly used for graph problem**
- **How to find the produce shortest path.**
  - **Floyd-Warshall algorithm is mainly used for generate the distance matrix.**

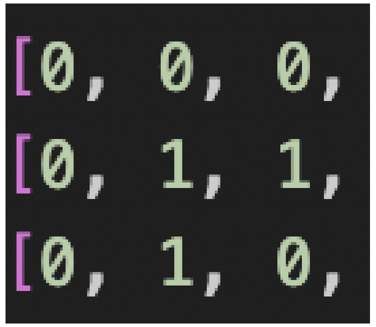
FLOYD-WARSHALL ALGORITHM – OUR APPROACH

MAZE TRANSFER THE MAZE MATRIX INTO A GRAPH (GENERATE VERTEX AND EDGE)

MAZE

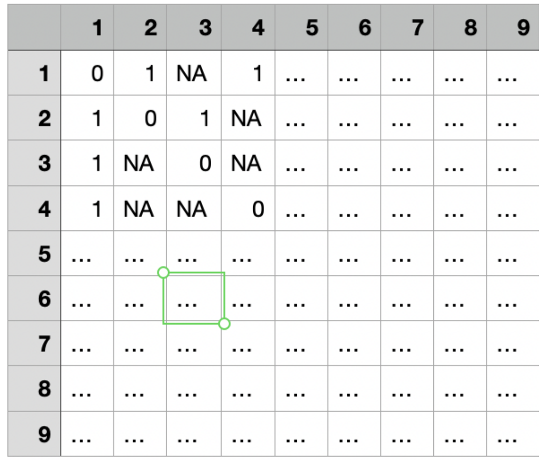


MAZE MATRIX

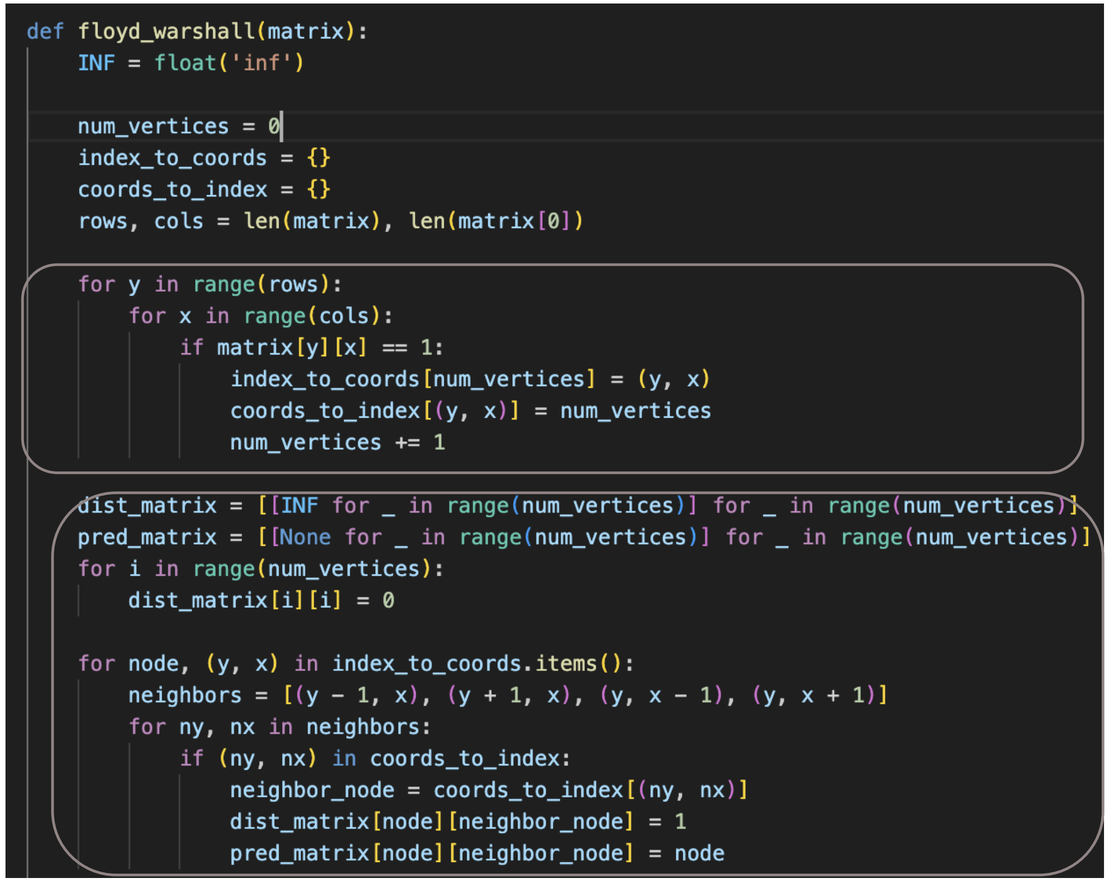


```
[0, 0, 0,
[0, 1, 1,
[0, 1, 0,
```

GRAPH MATRIX



	1	2	3	4	5	6	7	8	9
1	0	1	NA	1	...	...	...	...	...
2	1	0	1	NA	...	...	...	...	...
3	1	NA	0	NA	...	...	...	...	...
4	1	NA	NA	0	...	...	...	...	...
5	...	...	...	...	...	...	...	...	...
6	...	...	...	...	...	...	...	...	...
7	...	...	...	...	...	...	...	...	...
8	...	...	...	...	...	...	...	...	...
9	...	...	...	...	...	...	...	...	...

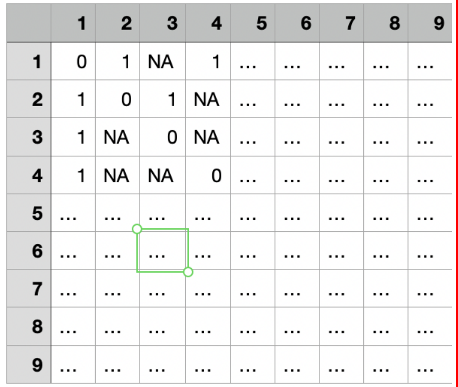


```python
def floyd_warshall(matrix):
    INF = float('inf')

    num_vertices = 0
    index_to_coords = {}
    coords_to_index = {}
    rows, cols = len(matrix), len(matrix[0])

    for y in range(rows):
        for x in range(cols):
            if matrix[y][x] == 1:
                index_to_coords[num_vertices] = (y, x)
                coords_to_index[(y, x)] = num_vertices
                num_vertices += 1

    dist_matrix = [[INF for _ in range(num_vertices)] for _ in range(num_vertices)]
    pred_matrix = [[None for _ in range(num_vertices)] for _ in range(num_vertices)]
    for i in range(num_vertices):
        dist_matrix[i][i] = 0

    for node, (y, x) in index_to_coords.items():
        neighbors = [(y - 1, x), (y + 1, x), (y, x - 1), (y, x + 1)]
        for ny, nx in neighbors:
            if (ny, nx) in coords_to_index:
                neighbor_node = coords_to_index[(ny, nx)]
                dist_matrix[node][neighbor_node] = 1
                pred_matrix[node][neighbor_node] = node
```

FLOYD-WARSHALL ALGORITHM – OUR APPROACH

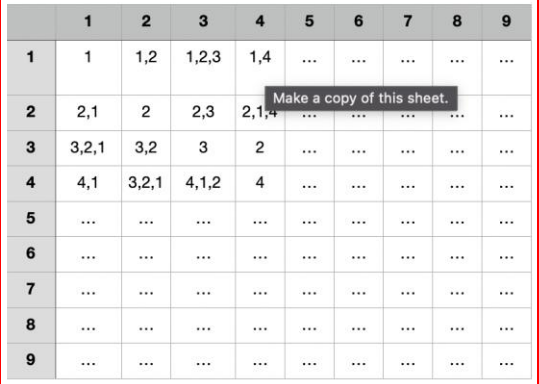
USE FLOYD WARSHALL TO FIND AND PRODUCE SHORTEST PATH

GRAPH MATRIX



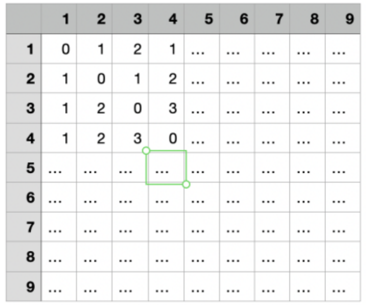
	1	2	3	4	5	6	7	8	9
1	0	1	NA	1	...	...	...	...	...
2	1	0	1	NA	...	...	...	...	...
3	1	NA	0	NA	...	...	...	...	...
4	1	NA	NA	0	...	...	...	...	...
5	...	...	...	...	...	...	...	...	...
6	...	...	...	...	...	...	...	...	...
7	...	...	...	...	...	...	...	...	...
8	...	...	...	...	...	...	...	...	...
9	...	...	...	...	...	...	...	...	...

GRAPH MATRIX


Make a copy of this sheet.

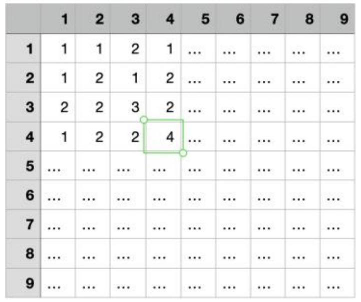
	1	2	3	4	5	6	7	8	9
1	1	1,2	1,2,3	1,4	...	...	...	...	...
2	2,1	2	2,3	2,1,4	...	...	...	...	...
3	3,2,1	3,2	3	2	...	...	...	...	...
4	4,1	3,2,1	4,1,2	4	...	...	...	...	...
5	...	...	...	...	...	...	...	...	...
6	...	...	...	...	...	...	...	...	...
7	...	...	...	...	...	...	...	...	...
8	...	...	...	...	...	...	...	...	...
9	...	...	...	...	...	...	...	...	...

MAZE MATRIX

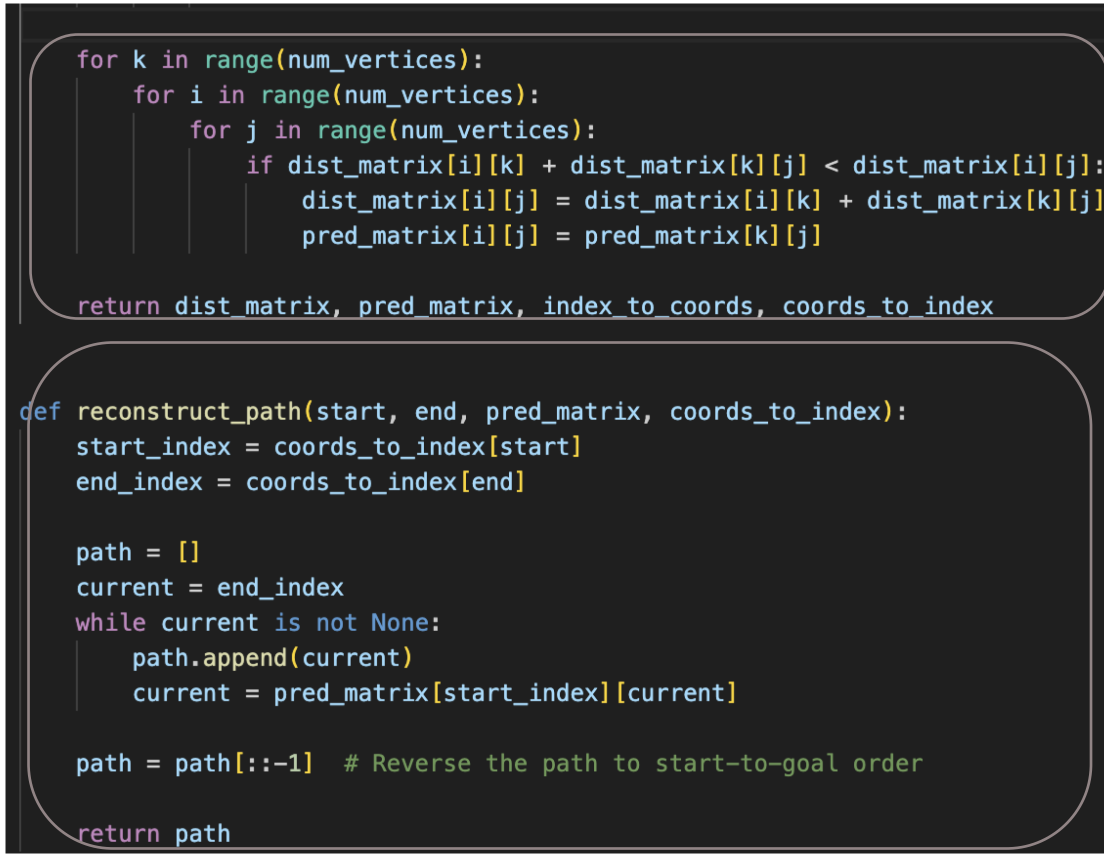


	1	2	3	4	5	6	7	8	9
1	0	1	2	1	...	...	...	...	...
2	1	0	1	2	...	...	...	...	...
3	1	2	0	3	...	...	...	...	...
4	1	2	3	0	...	...	...	...	...
5	...	...	...	...	...	...	...	...	...
6	...	...	...	...	...	...	...	...	...
7	...	...	...	...	...	...	...	...	...
8	...	...	...	...	...	...	...	...	...
9	...	...	...	...	...	...	...	...	...

MAZE MATRIX



	1	2	3	4	5	6	7	8	9
1	1	1	2	1	...	...	...	...	...
2	1	2	1	2	...	...	...	...	...
3	2	2	3	2	...	...	...	...	...
4	1	2	2	4	...	...	...	...	...
5	...	...	...	...	...	...	...	...	...
6	...	...	...	...	...	...	...	...	...
7	...	...	...	...	...	...	...	...	...
8	...	...	...	...	...	...	...	...	...
9	...	...	...	...	...	...	...	...	...



```
    for k in range(num_vertices):
        for i in range(num_vertices):
            for j in range(num_vertices):
                if dist_matrix[i][k] + dist_matrix[k][j] < dist_matrix[i][j]:
                    dist_matrix[i][j] = dist_matrix[i][k] + dist_matrix[k][j]
                    pred_matrix[i][j] = pred_matrix[k][j]

    return dist_matrix, pred_matrix, index_to_coords, coords_to_index


def reconstruct_path(start, end, pred_matrix, coords_to_index):
    start_index = coords_to_index[start]
    end_index = coords_to_index[end]

    path = []
    current = end_index
    while current is not None:
        path.append(current)
        current = pred_matrix[start_index][current]

    path = path[::-1]  # Reverse the path to start-to-goal order

    return path
```

PERFORMANCE COMPARISON

4.1 Time Complexity

Greedy Best First Search (GBFS)

The worst-case time complexity of GBFS is $O(V\log V)$.

Floyd-Warshall Algorithm

The time complexity of the Floyd-Warshall algorithm is $O(V3)$

4.2 Space Complexity

Greedy Best First Search (GBFS)

The primary data structures in GBFS are the priority_queue and visited set. Thus the space complexity is $O(V)$.

Floyd-Warshall Algorithm

The Floyd-Warshall algorithm requires two matrices (the dist_matrix and pred_matrix), each of size $V\times V$. Hence, the space complexity is $O(V2)$

4.3 Practical Considerations

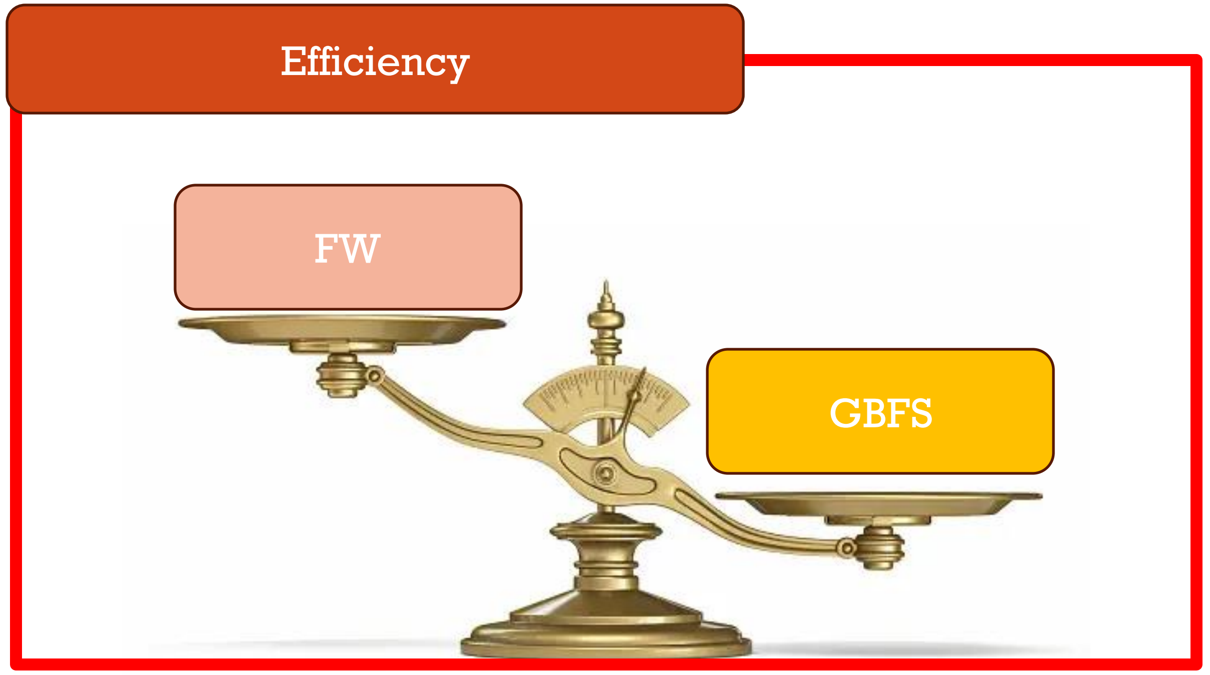
Greedy Best First Search (GBFS)

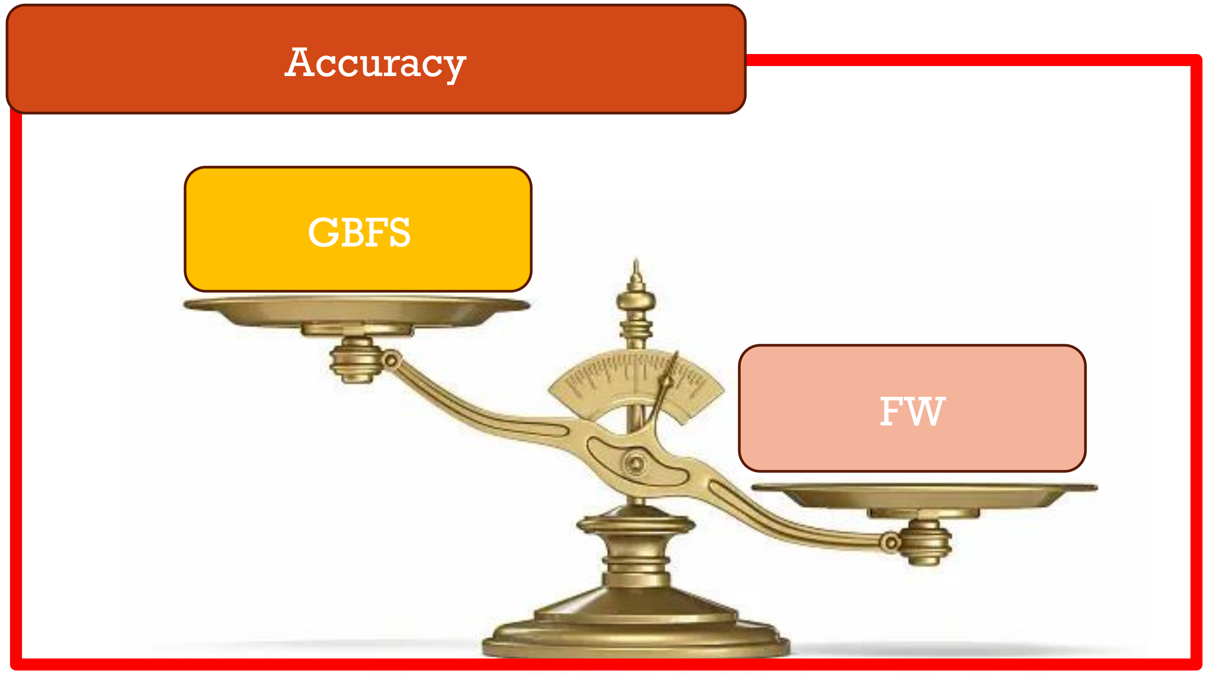
- **Optimality**: **GBFS does not guarantee the shortest path,**
- **Efficiency**: GBFS generally exhibits good performance, especially when the heuristic is well- chosen, as it is directed and avoids exploring unnecessary paths.
- **Use Case**: GBFS is particularly effective for large mazes where an approximate solution is satisfactory, and we want to find a path quickly.

Floyd-Warshall Algorithm

- **Optimality**: The Floyd-Warshall algorithm guarantees the shortest path between any two nodes, making it optimal for cases where the shortest path is required.
- **Efficiency**: Despite its cubic time complexity, Floyd-Warshall is highly predictable and can be efficient for small to moderately sized mazes. However, for very large mazes, its performance might become a bottleneck.
- **Use Case**: It is particularly useful when we need to solve multiple queries for shortest paths in the same maze, as it precomputes all pairs shortest paths.

CONCLUSION





THANK YOU

- Any Questions?