

**DEPARTMENT OF
SCHOOL OF COMPUTING
College of Engineering and Technology
SRM Institute of Science and Technology**



MINI PROJECT REPORT

ODD Semester, 2023-2024

Lab code & Sub Name : 21CSC201J & Data Structures and Algorithms

Year & Semester : II and III

Project Title : Pathfinding in a Maze Using Dijkstra's Algorithm

Lab Supervisor : **Dr. V. Pandimurugan**

Team Members : 1. Muhil (Reg.No:RA2211030010041)
2.Yashwanth (Reg.No:RA2211030010044)
3. Rishi (Reg.No:RA2211030010027)
4.Muhammad Fayaz(Reg.No:RA2211030010010)

TABLE OF CONTENTS

<i>C.NO.</i>	<i>TITLE</i>	<i>PAGE NO.</i>
1.	Problem Definition	3
2.	Problem Explanation with Diagram and Example	3
3.	Design Techniques Used - Dijkstra's Algorithm	4
4.	Algorithm for the Problem	4
5.	Explanation of Algorithm with Example	5
6.	Complexity Analysis	6
7.	Approach/methodology/programs	6
8.	Output	9
9.	Conclusion	10
10.	References	10

PATHFINDING IN A MAZE USING DIJKSTRA'S ALGORITHM

PROBLEM DEFINITION:

PROBLEM EXPLANATION WITH DIAGRAM AND EXAMPLE:

Problem Explanation:

Develop a program to find the shortest path through a maze represented as a grid. The maze contains open paths ('1') and obstacles ('0'). Using Dijkstra's algorithm, the program should determine the shortest path from a given starting point to a goal point while moving in four directions (up, down, left, and right). The program should handle various maze sizes, provide user input for maze configuration and points, and report the length of the shortest path or indicate when no valid path exists. This problem addresses the need for efficient maze pathfinding with practical applications in robotics, gaming, and navigation systems.

Diagram:



Example:

Consider a 5x5 maze, where '1' represents an open path, and '0' represents an obstacle. The start point 'S' is located at (0, 0), and the goal point 'G' is at (4, 4). The challenge is to find the shortest path from 'S' to 'G' through the maze while navigating around obstacles.

DESIGN TECHNIQUES USED:

Greedy Method:

Dijkstra's algorithm, employed in this project, falls under the category of the "Greedy Method" in algorithmic design techniques. The Greedy Method involves making locally optimal choices at each step, aiming to find a global optimum. Specifically, Dijkstra's algorithm in maze pathfinding selects the node with the smallest tentative distance from the start point to explore first. This process continues until the algorithm reaches the goal or examines all possible paths.

Dijkstra's algorithm demonstrates the Greedy Method by consistently choosing the most promising path, ensuring the shortest path from the start to the goal within a maze. Its local optimization strategy at each step contributes to the overall efficiency of the algorithm, making it a suitable choice for maze pathfinding.

Dijkstra's Algorithm:

Dijkstra's algorithm is a graph-based approach used to find the shortest path between nodes in a weighted graph. In the context of the maze problem, the maze grid is transformed into a graph, where each cell represents a node, and edges represent possible movements between adjacent cells. Dijkstra's algorithm iteratively explores the node with the smallest known distance from the start and updates the distances to its neighboring nodes until the goal node is reached. This algorithm guarantees the shortest path in a maze, making it a suitable choice for pathfinding.

ALGORITHM FOR THE PROBLEM:

1. Create a grid representation of the maze, where each cell is a node, and the edges represent possible movements between adjacent cells.
2. Initialize an array to store the distance to each cell, setting the distance to the start point as 0 and all other distances as infinity.
3. Create a priority queue to manage the nodes with their distances, and add the start node with a distance of 0.
4. While the priority queue is not empty:
 - a. Extract the node with the smallest distance from the priority queue.
 - b. If the extracted node is the goal node, the shortest path has been found. Terminate the algorithm.
 - c. Otherwise, for each neighboring node:
 - i. Calculate the distance to the neighboring node through the current node.
 - ii. If the calculated distance is less than the previously stored distance to the neighboring node, update the distance and add the neighboring node to the priority queue.
5. If the priority queue becomes empty without reaching the goal node, there is no valid path in the maze.

EXPLANATION OF ALGORITHM WITH EXAMPLE:

Let's apply Dijkstra's algorithm to the previously mentioned example. The grid, again, is as follows:

```
...
S 1 1 1 1
1 0 1 0 1
1 1 1 1 1
1 0 1 0 1
1 1 1 1 G
...
```

Step-by-Step Execution:

1. Initialize distances: Set the distance of the start node 'S' to 0 and all other nodes to infinity.
2. Create a priority queue and add the start node 'S' with distance 0.
3. While the priority queue is not empty:
 - a. Extract the node with the smallest distance. Initially, 'S' is extracted.
 - b. Explore its neighboring nodes, considering the four possible movements (up, down, left, right).
 - c. Update distances based on the neighboring nodes' positions and values (1 or 0):
 - From 'S,' we can reach the node (0, 1) with a distance of 1, (1, 0) with a distance of 1, and (1, 1) with a distance of 1.
 - Add these nodes to the priority queue for further exploration.
4. Continue this process until the goal node 'G' is reached or the priority queue becomes empty.
5. The algorithm terminates with the shortest path from 'S' to 'G' determined.

Shortest Path:

For the provided maze, Dijkstra's algorithm will find the shortest path from 'S' to 'G,' which, in this case, is the path following the '1's in the grid:

```
S 1 1 1 1
1 0 1 0 1
1 1 1 1 1
1 0 1 0 1
1 1 1 1 G
```

The algorithm calculates the length of this path as 12, which represents the number of '1's traversed from 'S' to 'G.'

COMPLEXITY ANALYSIS:

The time complexity of Dijkstra's algorithm depends on the number of nodes (cells) and edges in the graph. In the context of maze solving:

- Number of nodes: In an NxM maze, there are N*M cells, so the number of nodes is N*M.
- Number of edges: Each node can have up to 4 edges (up, down, left, right).

The worst-case time complexity is $O(N*M)$, making it an efficient algorithm for maze pathfinding. Additionally, the space complexity is $O(N*M)$ due to the storage of distances.

APPROACH/METHODOLOGY/PROGRAMS:

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
#include <string.h>

#define ROWS 5
#define COLS 5

// Function to print the grid
void printGrid(int grid[ROWS][COLS]) {
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%2d ", grid[i][j]);
        }
        printf("\n");
    }
}
```

```

// Function to check if a cell is valid
bool isValid(int row, int col) {
    return (row >= 0) && (row < ROWS) && (col >= 0) && (col < COLS);
}

// Dijkstra's algorithm to find the shortest path
void dijkstra(int grid[ROWS][COLS], int startX, int startY, int goalX, int goalY) {
    int dist[ROWS][COLS];
    bool visited[ROWS][COLS];

    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            dist[i][j] = INT_MAX;
            visited[i][j] = false;
        }
    }

    dist[startX][startY] = 0;

    for (int count = 0; count < ROWS * COLS - 1; count++) {
        int minDist = INT_MAX;
        int minRow = -1;
        int minCol = -1;

        for (int i = 0; i < ROWS; i++) {
            for (int j = 0; j < COLS; j++) {
                if (!visited[i][j] && dist[i][j] < minDist) {
                    minDist = dist[i][j];
                    minRow = i;
                    minCol = j;
                }
            }
        }

        if (minRow == -1 || minCol == -1) {
            break; // No valid path
        }

        visited[minRow][minCol] = true;

        int dr[] = {-1, 0, 1, 0};
        int dc[] = {0, 1, 0, -1};

        for (int k = 0; k < 4; k++) {
            int newRow = minRow + dr[k];
            int newCol = minCol + dc[k];

            if (isValid(newRow, newCol) && !visited[newRow][newCol] &&

```

```

        grid[newRow][newCol] && dist[minRow][minCol] + grid[newRow][newCol] <
dist[newRow][newCol]) {
            dist[newRow][newCol] = dist[minRow][minCol] + grid[newRow][newCol];
        }
    }
}

if (dist[goalX][goalY] == INT_MAX) {
    printf("No valid path from (%d, %d) to (%d, %d)\n", startX, startY, goalX, goalY);
} else {
    printf("Shortest path from (%d, %d) to (%d, %d) has a length of %d\n", startX,
startY, goalX, goalY, dist[goalX][goalY]);
}
}

int main() {
    int grid[ROWS][COLS] = {
        {1, 1, 1, 1, 1},
        {1, 0, 1, 0, 1},
        {1, 1, 1, 1, 1},
        {1, 0, 1, 0, 1},
        {1, 1, 1, 1, 1}
    };

    int startX, startY, goalX, goalY;
    char input[20];

    printf("Grid:\n");
    printGrid(grid);

    // Get the start point (row col)
    while (1) {
        printf("Enter the start point (row col): ");
        fgets(input, sizeof(input), stdin);
        if (sscanf(input, "%d%d", &startX, &startY) == 2 && isValid(startX, startY)) {
            break;
        } else {
            printf("Invalid input. Please enter valid coordinates.\n");
        }
    }

    // Get the goal point (row col)
    while (1) {
        printf("Enter the goal point (row col): ");
        fgets(input, sizeof(input), stdin);
        if (sscanf(input, "%d%d", &goalX, &goalY) == 2 && isValid(goalX, goalY)) {
            break;
        } else {
            printf("Invalid input. Please enter valid coordinates.\n");
        }
    }
}

```



```

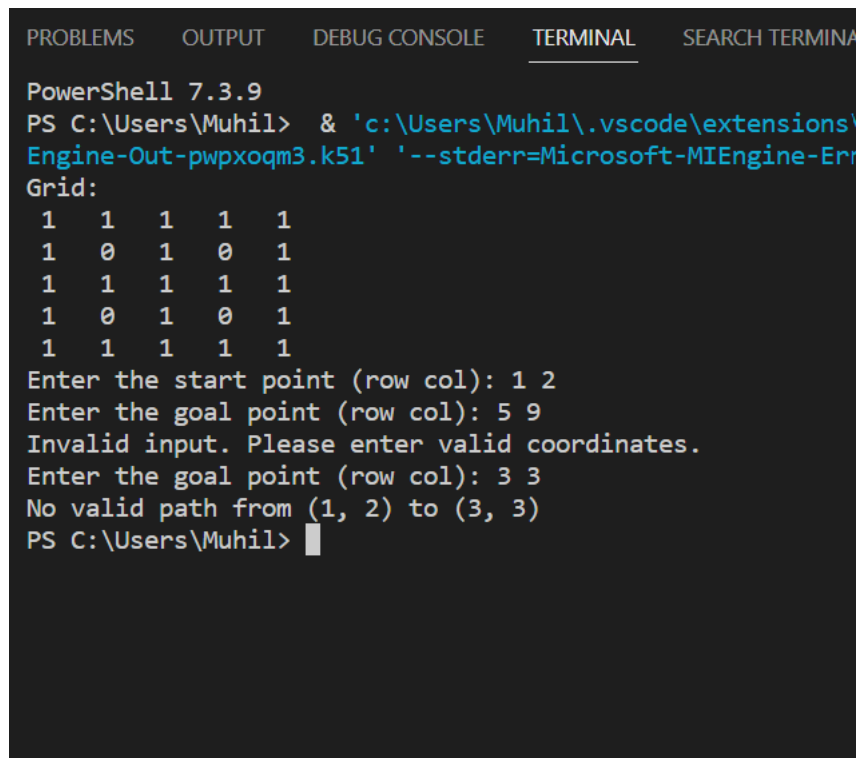
    }

    dijkstra(grid, startX, startY, goalX, goalY);

    return 0;
}

```

OUTPUT:



```

PowerShell 7.3.9
PS C:\Users\Muhil> & 'c:\Users\Muhil\.vscode\extensions\Microsoft-MIEngine-Err
Grid:
  1  1  1  1  1
  1  0  1  0  1
  1  1  1  1  1
  1  0  1  0  1
  1  1  1  1  1
Enter the start point (row col): 1 2
Enter the goal point (row col): 5 9
Invalid input. Please enter valid coordinates.
Enter the goal point (row col): 3 3
No valid path from (1, 2) to (3, 3)
PS C:\Users\Muhil>

```

CONCLUSIONS:

In this report, we explored the problem of pathfinding in a maze using Dijkstra's algorithm. We provided a detailed explanation of the algorithm, along with a step-by-step example of its application in a sample maze. Dijkstra's algorithm guarantees the shortest path, and its time complexity is efficient, making it a suitable choice for maze navigation.

Maze pathfinding is a practical problem with various real-world applications, including robotics, gaming, and navigation systems. By using Dijkstra's algorithm, we can ensure efficient and optimal routes to reach a goal within a maze.

REFERENCES:

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
2. Russell, S. J., & Norvig, P. (2009). Artificial Intelligence: A Modern Approach (3rd ed.). Pearson.
3. Millington, I., & Funge, J. (2009). Artificial Intelligence for Games. CRC Press.
4. Eberly, D. H. (2010). 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics. CRC Press.
5. Dijkstra, E.W. (1959). A Note on Two Problems in Connexion with Graphs. Numerische Mathematik, 1(1), 269-271..