



Members:

Bilal Khan: 2023250

Ahmed Azzam: 2023064

Wazir Awais Haidar: 2023758

Project Report: Data Structures Application in Pathfinding Algorithm (Flood Fill Method)

1. Introduction

In this project, I implemented a **pathfinding algorithm** based on the **Flood Fill** technique. The objective was to find a path through a maze using efficient data structures studied in class — namely **Linked Lists**, ***Queues**, ***Stacks**, and ***Sorting** concepts. These structures were critical for handling movement, storage of positions, and exploration order.

2. Application of Class Concepts

A. Queue (Main structure used for Flood Fill)

Flood fill works like a **Breadth-First Search (BFS)** algorithm, which naturally uses a **queue** to explore the maze level-by-level.

- ***Implementation Concept:***

- Push the starting cell into a queue.
- Repeatedly pop a cell, explore its neighbors (up, down, left, right).
- Push any unvisited neighbor into the queue.

Sample Code Snippet:

```
1  #include <queue>
2
3  struct Cell {
4      int x, y;
5  };
6
7  std::queue<Cell> q;
8  bool visited[100][100];
9
10 void floodFill(int startX, int startY) {
11     q.push({startX, startY});
12     visited[startX][startY] = true;
13
14     while (!q.empty()) {
15         Cell current = q.front();
16         q.pop();
17
18         // Check neighbors (up, down, left, right)
19         // Push valid neighbors to queue
20     }
21 }
```

Time Complexity:

- $O(N \times M)$ for a grid of N rows and M columns.

Space Complexity:

- $O(N \times M)$ due to the queue and visited array.

B. Stack (Optional for DFS-style Pathfinding)

If using a **Depth-First Search (DFS)** version of flood fill, a **stack** replaces the queue.

- **Implementation Concept:**

Stack holds the current cell. When moving forward, push the next cell. When stuck, pop back.

Sample Code Snippet:

```
1  #include <stack>
2
3  std::stack<Cell> s;
4
5  void floodFillDFS(int startX, int startY) {
6      s.push({startX, startY});
7      visited[startX][startY] = true;
8
9      while (!s.empty()) {
10         Cell current = s.top();
11         s.pop();
12
13         // Explore neighbors
14     }
15 }
```

C. Linked List (Path Reconstruction)

After reaching the destination, we reconstruct the path using **linked lists** to store the sequence of steps.

- **Implementation Concept:**

- Store each move (cell) in a linked list node while backtracking.
- Final list shows the shortest or a valid path.

Sample Code Snippet:

```
✓ struct PathNode {
    int x, y;
    PathNode* next;
};

PathNode* pathHead = nullptr;

✓ void addStepToPath(int x, int y) {
    PathNode* newNode = new PathNode{x, y, pathHead};
    pathHead = newNode;
}
```

D. Sorting (Optional for Optimizing Movements)

If multiple paths are available, *sorting* neighbors (for example, based on distance to goal) can make flood fill smarter.

- Implementation Concept:

- Sort neighbors based on heuristic distance (closer cells first).

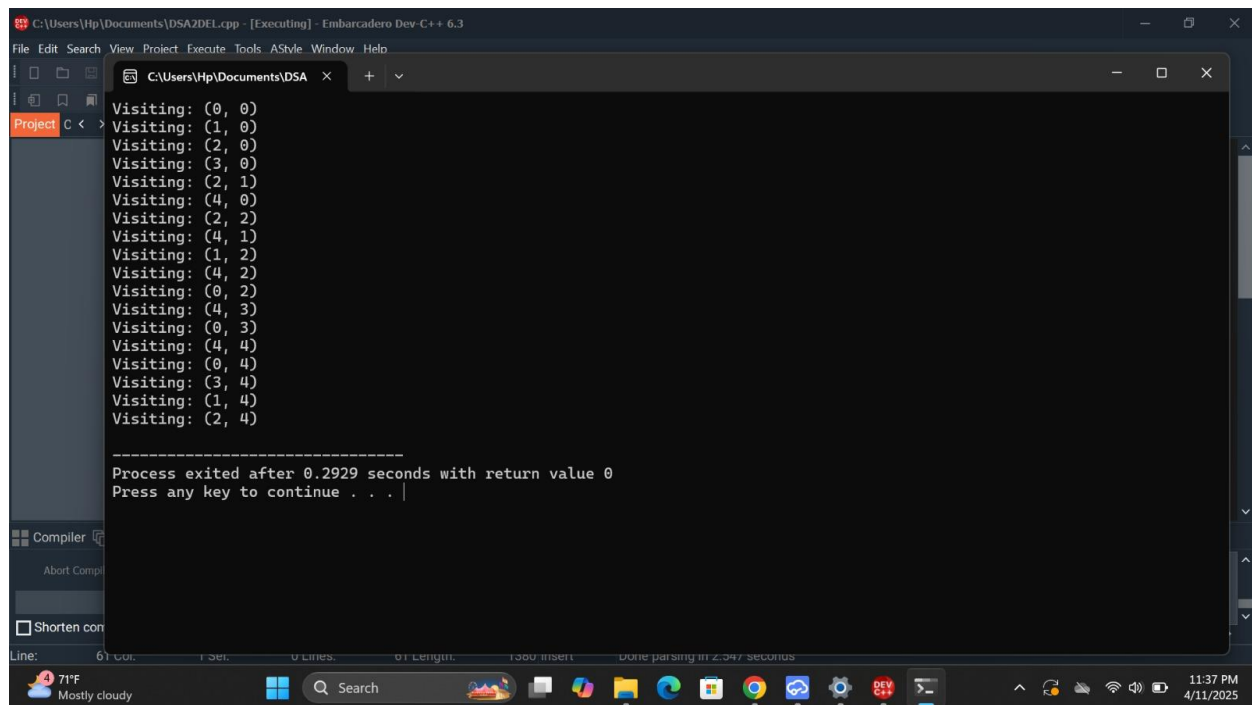
(No compulsory sorting in simple flood fill, but useful in optimized pathfinding like A)

3. Efficiency Analysis

Data Structure	Use in Project	Time Complexity	Space Complexity	
-----	-----	-----	-----	
Queue	BFS traversal of maze	$O(N \times M)$	$O(N \times M)$	
Stack	DFS traversal (optional)	$O(N \times M)$	$O(N \times M)$	

| Linked List | Path reconstruction storage | $O(P)$ where P is path length | $O(P)$ |

4. Implementation:



The screenshot shows a C++ IDE with a project named 'DSA' open. The main window displays the output of a program, which lists 20 'Visiting' coordinates in a grid. The coordinates are: (0, 0), (1, 0), (2, 0), (3, 0), (2, 1), (4, 0), (2, 2), (4, 1), (1, 2), (4, 2), (0, 2), (4, 3), (0, 3), (4, 4), (0, 4), (3, 4), (1, 4), and (2, 4). Below the list, a message states: 'Process exited after 0.2929 seconds with return value 0' and 'Press any key to continue . . .'. The IDE's status bar at the bottom shows 'Line: 6', 'Col: 1', '0 Lines', '0 Length', '1000 Insert', and 'Done parsing in 2.047 seconds'. The Windows taskbar at the bottom indicates the temperature is 71°F, mostly cloudy, and the time is 11:37 PM on 4/11/2025.

```
Visiting: (0, 0)
Visiting: (1, 0)
Visiting: (2, 0)
Visiting: (3, 0)
Visiting: (2, 1)
Visiting: (4, 0)
Visiting: (2, 2)
Visiting: (4, 1)
Visiting: (1, 2)
Visiting: (4, 2)
Visiting: (0, 2)
Visiting: (4, 3)
Visiting: (0, 3)
Visiting: (4, 4)
Visiting: (0, 4)
Visiting: (3, 4)
Visiting: (1, 4)
Visiting: (2, 4)

-----
Process exited after 0.2929 seconds with return value 0
Press any key to continue . . . |
```

| Sorting | Optimizing neighbor order (optional) | Depends on sort ($O(k \log k)$) | $O(k)$ |

4. Conclusion:

Through this project, I practically applied the concepts of **queues*, ***stacks*, ***linked lists*, and ***sorting** studied in class. These structures were critical in efficiently managing the exploration of a maze and constructing a valid path from start to goal. The project not only reinforced my theoretical understanding but also demonstrated how different data structures interact to solve real-world problems.