Assignment No-3

Name :- Ayaan Shaikh

Roll No. :- C-42

Class :- T.Y . B.Tech.

Aim :- To implement Breadth First Search (BFS) and Depth First Search (DFS) algorithms to solve a given state-space problem and compare their time and space complexity empirically.

Problem Statement

Implement BFS and DFS to solve a given state-space problem (e.g., maze navigation or 8-puzzle). Compare time and space complexity empirically.

Description

Uninformed search algorithms explore the state space without using any domain-specific knowledge. Among these, **Breadth First Search (BFS)** and **Depth First Search (DFS)** are fundamental techniques used to traverse graphs and trees.

In this assignment, BFS and DFS are implemented to solve a **maze navigation problem**, which represents a state-space search problem. The performance of both algorithms is analyzed and compared based on time and space usage.

State-Space Problem: Maze Navigation

State Representation :-

- Each state represents the **current position (row, column)** of the agent in the maze.

Initial State :-

- Starting cell of the maze.

Goal State :-

- Destination cell of the maze.

Actions :-

- Move **Up**, **Down**, **Left**, or **Right** (if not blocked).

Maze Representation :-

- 0 → Free cell
- 1 → Obstacle

Maze Structure

```
maze = [
   [0, 1, 0, 0, 0],
   [0, 1, 0, 1, 0],
   [0, 0, 0, 1, 0],
   [1, 1, 0, 0, 0],
   [0, 0, 0, 1, 0]
]

start = (0, 0)
goal = (4, 4)
```

Algorithm 1: Breadth First Search (BFS)

Description :-

Breadth First Search explores the state space **level by level** using a queue. It is **complete** and guarantees the shortest path in an unweighted graph.

Python Implementation :-

```
fromcollectionsimport deque

defbfs(maze,start,goal):
    queue=deque([(start, [start])])
    visited=set([start])
    nodes_expanded= 0

    whilequeue:
        (x,y),path=queue.popleft()
        nodes_expanded += 1

        if(x,y)==goal:
            returnpath,nodes_expanded

        fordx,dyin[(-1,0),(1,0),(0,-1),(0,1)]:
            nx,ny=x+dx, y + dy
            if0<=nx<len(maze) and 0 <= ny < len(maze[0]):
                ifmaze[nx][ny] == 0 and (nx, ny) not in visited:
                    visited.add((nx, ny))
                    queue.append(((nx, ny), path + [(nx, ny)]))

     returnNone,nodes_expanded
```

Observation (BFS) :-

- BFS explores many nodes.
- Requires more memory due to queue storage.
- Always finds the shortest path.

Algorithm 2: Depth First Search (DFS)

Description :-

Depth First Search explores **as deep as possible** along one path before backtracking. It uses a stack or recursion and requires less memory than BFS.

Python Implementation :-

```
defdfs(maze,start,goal):
    stack=[(start,[start])]
    visited=set([start])
    nodes_expanded= 0

    whilestack:
        (x,y),path=stack.pop()
        nodes_expanded += 1

        if(x,y)==goal:
            returnpath,nodes_expanded

        fordx,dyin[(-1,0),(1,0),(0,-1),(0,1)]:
            nx,ny=x+dx, y + dy
            if0<=nx<len(maze) and 0 <= ny < len(maze[0]):
                ifmaze[nx][ny] == 0 and (nx, ny) not in visited:
                    visited.add((nx, ny))
                    stack.append(((nx, ny), path + [(nx, ny)]))

    returnNone,nodes_expanded
```

Observation (DFS) :-

- DFS uses less memory.
- May explore unnecessary deep paths.
- Does not guarantee shortest path.

Empirical Performance Comparison :-

```
bfs_path, bfs_nodes = bfs(maze, start, goal)
dfs_path, dfs_nodes = dfs(maze, start, goal)

print("BFS Nodes Expanded:", bfs_nodes)
print("DFS Nodes Expanded:", dfs_nodes)
```

Performance Comparison Table :-

Criteria	BFS	DFS
Data Structure	Queue	Stack
Nodes Expanded	Higher	Lower
Memory Usage	High	Low
Completeness	Yes	No
Optimal Solution	Yes	No
Execution Speed	Slower	Faster

Output Analysis :-

- BFS expands more nodes but guarantees the shortest path.
- DFS expands fewer nodes but may take a longer or sub-optimal route.
- BFS consumes more memory, while DFS is space-efficient.

Conclusion

This assignment successfully implements BFS and DFS algorithms to solve a maze navigation problem. BFS is complete and optimal but requires more memory. DFS is memory-efficient but does not guarantee the shortest path. The empirical comparison highlights the trade-off between time, space, and optimality in uninformed search algorithms.