Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

SUB: AIML
Branch: COMPS B

Name :	**Chetan Deepak Patil**
UID :	**2023301012**
Exp no :	**02**
Aim :	**Implement Tic Tac Toe and print the goal state using DFS. At what depth of Goal is found? Analyze DFS with respect to Completeness, Optimality, Space and time complexity.**
Thesis :	**Explanation:** 1. **DFS Function:** ○ Added an additional parameter `bestPath` to keep track of the optimal path leading to the minimum depth winning state. ○ When a winning state is found, it checks if the depth is less than the current `minDepth` and updates `minDepth` and `bestPath` accordingly. 2. **Main Function:** ○ Initializes `bestPath` to store the board state leading to the optimal solution. ○ Calls `dfs` and prints the minimum depth and the board configuration (`bestPath`) if a winning state is found. **Output:** ● The program will output the minimum depth required to win and print the board configuration that represents the optimal path to achieve this winning state.

Analysis:

Analysis of DFS in the Tic-Tac-Toe Problem

1. **Completeness:**
   - DFS is complete in this scenario because it will explore all possible game states systematically by examining every move from the initial state until it either finds a winning state or exhausts all possibilities. Since the Tic-Tac-Toe board is finite with a maximum of 9 cells, DFS will always find a solution if one exists.
2. **Optimality:**
   - DFS is not inherently optimal. It explores one path as deeply as possible before backtracking, which means it may find a solution that is not the shortest. In the context of finding a minimum depth winning state, the code includes a mechanism to track and update the minimum depth, effectively making it optimal for this specific problem by comparing depths whenever a winning state is found.
3. **Space Complexity:**
   - The space complexity is primarily influenced by the depth of the recursion stack and the space required to store the board states. For a 3x3 Tic-Tac-Toe board, the maximum depth of recursion is 9 (one move per cell).
   - Space complexity is O(N2)O(N^2)O(N2), where NNN is the size of the board (3 in this case), because we store board states and use a recursion stack.
4. **Time Complexity:**
   - The time complexity is determined by the number of possible board configurations. Each cell can have three states (empty, 'X', 'O'), leading to 3N23^{N^2}3N2 potential configurations, but many of these are invalid due to game rules (e.g., not all cells can be filled with 'X').
   - Thus, the effective time complexity can be approximated as O((N2)!)O((N^2)!)O((N2)!) for the worst-case scenario, where the factorial accounts for all permutations of moves and board configurations.

In practice, for small-sized boards like Tic-Tac-Toe, DFS can be feasible. However, for larger game spaces or more complex problems, other algorithms like BFS or heuristic-based approaches might be more suitable to ensure optimal solutions and manage complexity.

Code :	(see code below)

```
#include <stdio.h>

#include <stdbool.h>



#define SIZE 3

#define EMPTY ' '

#define PLAYER_X 'X'

#define PLAYER_O 'O'



// To print the boards

void printBoard(char board[SIZE][SIZE]) {

    for (int i = 0; i < SIZE; i++) {

        for (int j = 0; j < SIZE; j++) {

            printf("%c ", board[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}
```

```
// Checking winning state
bool isWinning(char board[SIZE][SIZE], char player) {
    for (int i = 0; i < SIZE; i++) {
        if ((board[i][0] == player && board[i][1] == player &&
board[i][2] == player) ||
            (board[0][i] == player && board[1][i] == player &&
board[2][i] == player)) {
            return true;
        }
    }
    return (board[0][0] == player && board[1][1] == player &&
board[2][2] == player) ||
           (board[0][2] == player && board[1][1] == player &&
board[2][0] == player);
}


// To perform DFS and find minimum depth for a win
bool dfs(char board[SIZE][SIZE], int depth, char currentPlayer, int
*minDepth, char minDepthBoard[SIZE][SIZE]) {
    if (isWinning(board, PLAYER_X) || isWinning(board, PLAYER_O)) {
        if (depth < *minDepth) {
            *minDepth = depth;
```

```
            for (int i = 0; i < SIZE; i++) {
                for (int j = 0; j < SIZE; j++) {
                    minDepthBoard[i][j] = board[i][j];
                }
            }
        }
        printBoard(board);
        return true;
    }

    if (depth == SIZE * SIZE) return false;

    bool foundSolution = false;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (board[i][j] == EMPTY) {
                board[i][j] = currentPlayer;
                if (dfs(board, depth + 1, (currentPlayer ==
PLAYER_X) ? PLAYER_O : PLAYER_X, minDepth, minDepthBoard)) {
                    foundSolution = true;
                }
```

```
                board[i][j] = EMPTY;
            }
        }
    }
    return foundSolution;
}


int main() {
    char board[SIZE][SIZE] = {{EMPTY, EMPTY, EMPTY}, {EMPTY, EMPTY,
EMPTY}, {EMPTY, EMPTY, EMPTY}};
    int minDepth = SIZE * SIZE;
    char minDepthBoard[SIZE][SIZE] = {{EMPTY, EMPTY, EMPTY},
{EMPTY, EMPTY, EMPTY}, {EMPTY, EMPTY, EMPTY}};
    printf("Goal States:\n");
    if (dfs(board, 0, PLAYER_X, &minDepth, minDepthBoard)) {
        printf("Minimum depth to win is: %d\n", minDepth);
        printf("Board state at minimum depth:\n");
        printBoard(minDepthBoard);
    } else {
        printf("No winning state found.\n");
    }
    return 0;
```

Output :

The output of the above code is as follows :

ie. It prints the all the possible winning states and Optimal state.

```
[Running] cd "c:\Users\Chetan\OneDrive\Desktop\5th_Sem\AIML\Expt_2\" && gcc tic.c -o
Minimum depth to win is: 5

[Done] exited with code=0 in 1.542 seconds

[Running] cd "c:\Users\Chetan\OneDrive\Desktop\5th_Sem\AIML\Expt_2\" && gcc tic.c -o
Goal States:
X O X
O X O
X

X O X
O X O
O X X

X O X
O X O
X X O

X O X
O X O
    X
```

```
O X X
O X O
X O X

X X O
O X O
X O X

    X
O X O
X O X

Minimum depth to win is: 5
Board state at minimum depth:
X O O
X
X


[Done] exited with code=0 in 0.8 seconds
```

Conclusion :	**Conclusion:** DFS provides a systematic approach to exploring all possible moves in a game, making it a useful tool for problem-solving in small-scale, finite scenarios like Tic-Tac-Toe, though it may not always yield the most optimal path.

	Learnings: The implementation demonstrates how Depth-First Search (DFS) can effectively explore game states in Tic-Tac-Toe to find optimal solutions, showcasing the importance of recursive backtracking and state management in algorithm design.