Experiment No. 11
15 puzzle problem
Date of Performance:
Date of Submission:

Experiment No. 11

Title: 15 Puzzle

Aim: To study and implement 15 puzzle problem

Objective: To introduce Backtracking and Branch-Bound methods

Theory:

The 15 puzzle problem is invented by sam loyd in 1878.

- In this problem there are 15 tiles, which are numbered from 0 – 15.
- The objective of this problem is to transform the arrangement of tiles from initial arrangement to a goal arrangement.
- The initial and goal arrangement is shown by following figure.

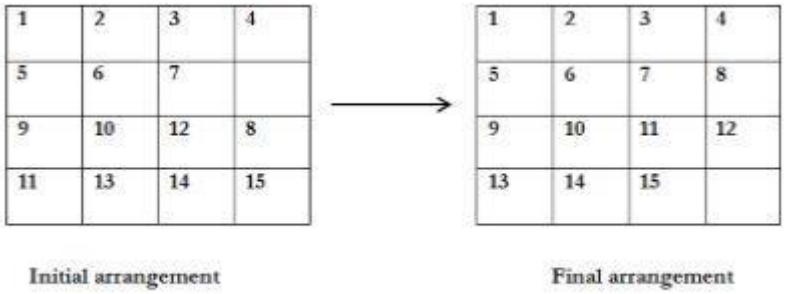
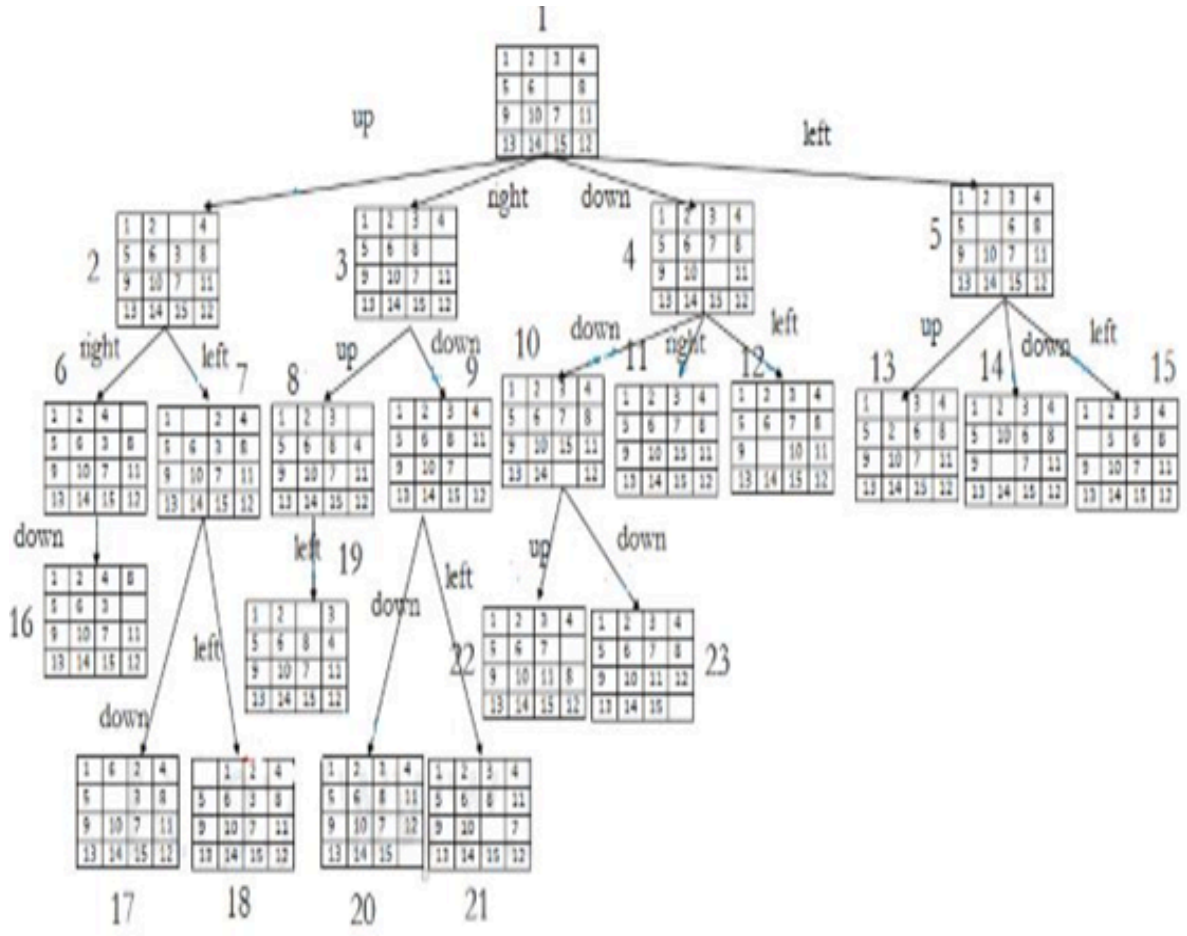


Figure 12

- There is always an empty slot in the initial arrangement.

- The legal moves are the moves in which the tiles adjacent to ES are moved to either left, right, up or down.
- Each move creates a new arrangement in a tile.
- These arrangements are called as states of the puzzle.
- The initial arrangement is called as initial state and goal arrangement is called as goal state.
- The state space tree for 15 puzzle is very large because there can be 16! Different arrangements.
- A partial state space tree can be shown in figure.
- In state space tree, the nodes are numbered as per the level.
- Each next move is generated based on empty slot positions.
- Edges are label according to the direction in which the empty space moves.
- The root node becomes the E – node.
- The child node 2, 3, 4 and 5 of this E – node get generated.
- Out of which node 4 becomes an E – node. For this node the live nodes 10, 11, 12 gets generated.
- Then the node 10 becomes the E – node for which the child nodes 22 and 23 gets generated.
- Finally we get a goal state at node 23.
- We can decide which node to become an E – node based on estimation formula.

Example:



Implementation:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>


#define N 4
```

```
// Structure to represent a state of the puzzle
typedef struct {
    int puzzle[N][N]; // Configuration of the puzzle
    int x, y;         // Position of the blank tile
    char path[50];    // Path taken to reach this state
} State;

// Structure to represent a node in the BFS queue
typedef struct {
    State state;
    int distance; // Distance from the initial state
} Node;

// Function to swap two integers
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Function to check if a state is the goal state
bool isGoalState(State state) {
```

```
    int value = 1;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (state.puzzle[i][j] != value % (N * N)) {
                return false;
            }
            value++;
        }
    }
    return true;
}

// Function to print the path taken to reach the goal state
void printPath(State state) {
    printf("Path to the goal state:\n%s\n", state.path);
}

// Function to check if a move is valid
bool isValidMove(int x, int y) {
    return (x >= 0 && x < N && y >= 0 && y < N);
}

// Function to perform BFS to solve the 15 Puzzle problem
```

```
void solvePuzzle(State initialState) {
    // Define the possible moves (up, down, left, right)
    int dx[] = {-1, 1, 0, 0};
    int dy[] = {0, 0, -1, 1};
    char dir[] = {'U', 'D', 'L', 'R'};

    // Initialize the BFS queue
    Node queue[100000];
    int front = 0, rear = 0;
    queue[rear++] = (Node){initialState, 0};

    // Perform BFS
    while (front < rear) {
        Node currentNode = queue[front++];
        State currentState = currentNode.state;
        int currentDistance = currentNode.distance;

        // Check if current state is the goal state
        if (isGoalState(currentState)) {
            printPath(currentState);
            return;
        }
```

```
        // Explore all possible moves from the current state
        for (int i = 0; i < 4; i++) {
            int newX = currentState.x + dx[i];
            int newY = currentState.y + dy[i];

            // Check if the new position is valid
            if (isValidMove(newX, newY)) {
                // Create a new state by swapping the blank tile with the adjacent tile
                State newState = currentState;
                                    swap(&newState.puzzle[currentState.x][currentState.y],
&newState.puzzle[newX][newY]);
                newState.x = newX;
                newState.y = newY;
                // Update the path
                newState.path[currentDistance] = dir[i];
                newState.path[currentDistance + 1] = '\0';

                // Add the new state to the BFS queue
                queue[rear++] = (Node){newState, currentDistance + 1};
            }
        }
    }
```

```
    printf("No solution found.\n");
}

int main() {
    State initialState;
    int puzzle[N][N] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 0, 15}
    };

    // Find the position of the blank tile
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            initialState.puzzle[i][j] = puzzle[i][j];
            if (puzzle[i][j] == 0) {
                initialState.x = i;
                initialState.y = j;
            }
        }
    }
    initialState.path[0] = '\0';
```

```
    solvePuzzle(initialState);

    return 0;
}
```

Conclusion: The 15 Puzzle problem has been implemented.