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.

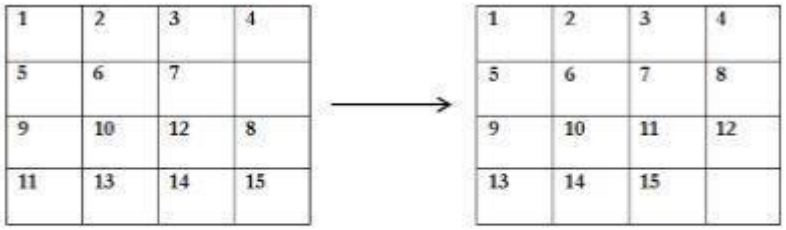


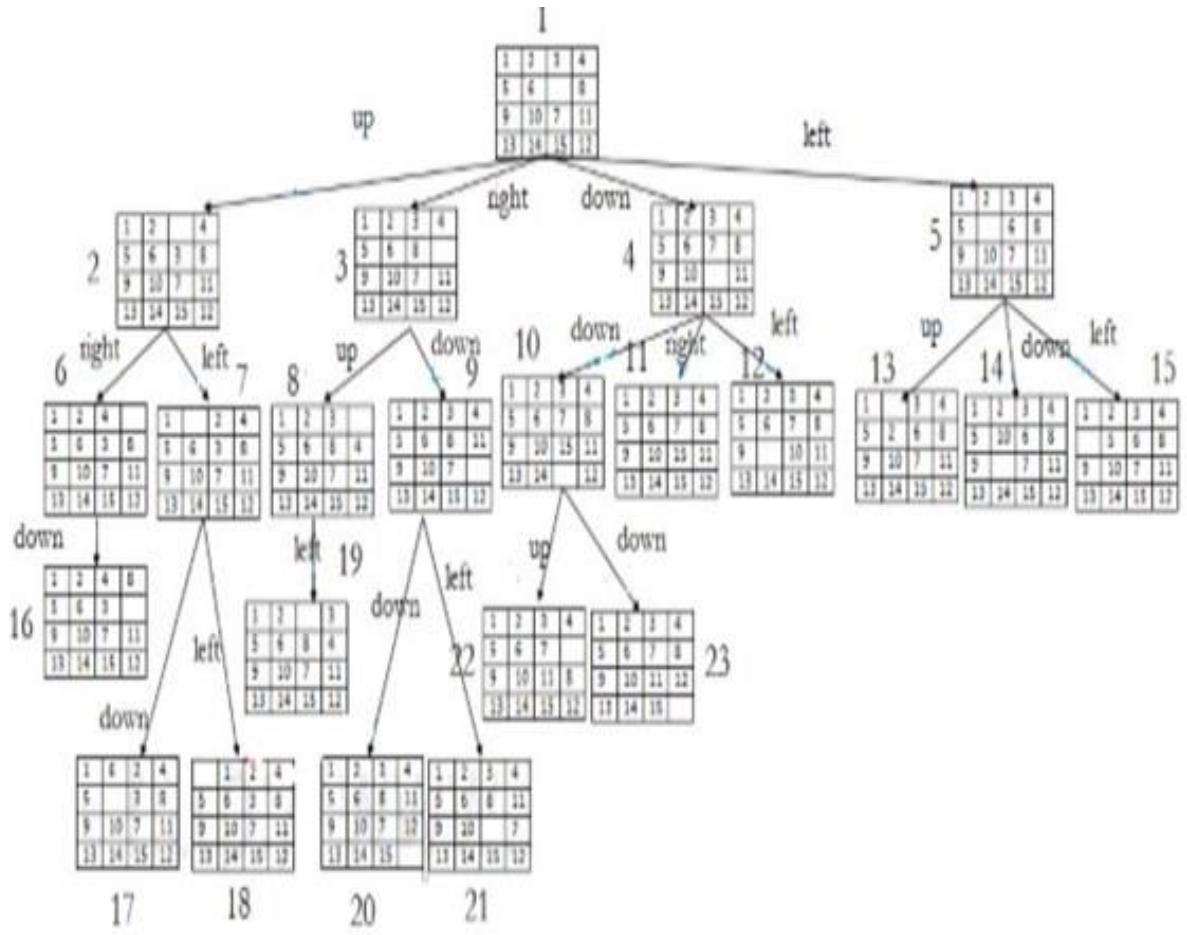
Figure 12

- 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:



Code :

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

```
#define N 8

// Function to check if a queen can be placed at board[row][col]
bool isSafe(int board[N][N], int row, int col) {
   int i, j;

   // Check this row on the left side
   for (i = 0; i < col; i++) {
      if (board[row][i]) {
         return false;
      }
   }

   // Check upper diagonal on left side
   for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
      if (board[i][j]) {
         return false;
      }
   }

   // Check lower diagonal on left side
   for (i = row, j = col; j >= 0 && i < N; i++, j--) {
      if (board[i][j]) {
         return false;
      }
   }

   return true;
}

// Recursive function to solve N-Queens problem using branch and bound
bool solveNQueensUtil(int board[N][N], int col) {
   // If all queens are placed then return true
   if (col >= N) {
      return true;
   }

   // Consider this column and try placing this queen in all rows one by one
   for (int i = 0; i < N; i++) {
```

```
        // Check if the queen can be placed on board[i][col]
        if (isSafe(board, i, col)) {
            // Place this queen in board[i][col]
            board[i][col] = 1;

            // Recur to place the rest of the queens
            if (solveNQueensUtil(board, col + 1)) {
                return true;
            }

            // If placing queen in board[i][col] doesn't lead to a solution then backtrack
            board[i][col] = 0; // Backtrack
        }
    }

    // If the queen cannot be placed in any row in this column, then return false
    return false;
}

// Function to solve N-Queens problem using branch and bound
void solveNQueens() {
    int board[N][N] = { {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0} };

    if (solveNQueensUtil(board, 0) == false) {
        printf("Solution does not exist");
        return;
    }

    // Print the solution
    printf("Solution:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
```

```
            printf(" %d ", board[i][j]);
        printf("\n");
    }
}

int main() {
    solveNQueens();
    return 0;
}
```

Output :

```
PS E:\Testing_Lang> cd "e:\Testing_Lang\" ; if ($?) { gcc test.c -o test } ; if ($?) { .\test }
Solution:
 1  0  0  0  0  0  0  0
 0  0  0  0  0  0  1  0
 0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  1
 0  1  0  0  0  0  0  0
 0  0  0  1  0  0  0  0
 0  0  0  0  0  1  0  0
 0  0  1  0  0  0  0  0
PS E:\Testing_Lang>
```

Conclusion: The 15 Puzzle problem has been implemented.