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

**Experiment No. 11**

**Title:** 15 Puzzle

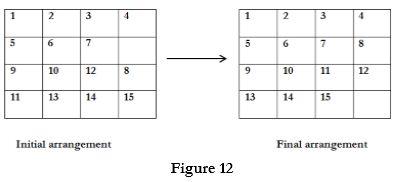
**Aim:** To study and implement 15 puzzle problem

**Objective:** To introduce Backtracking and Branch-Bound methods

**Theory:**

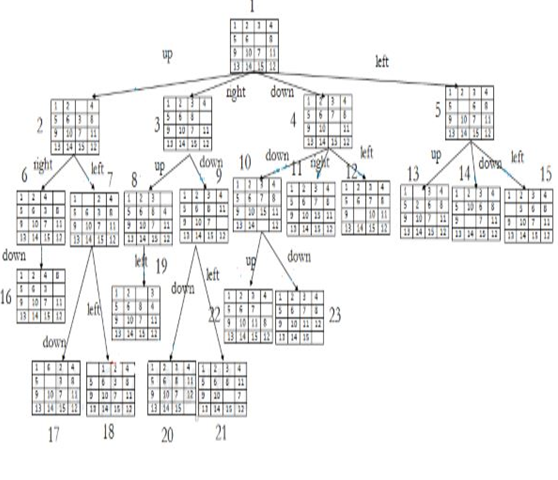
The 15 puzzle problem is invented by sam loyd in 1878.

* 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.



* 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>

#define N 4 // Size of the puzzle grid (N x N)

#define EMPTY\_TILE 0

// Function to print the current state of the puzzle

void printPuzzle(int puzzle[N][N]) {

int i,j;

    for (i = 0; i < N; i++) {

    for (j = 0; j < N; j++) {

            if (puzzle[i][j] == EMPTY\_TILE) {

                printf("   "); // Print empty tile

            } else {

                printf("%2d ", puzzle[i][j]);

            }

        }

        printf("\n");

    }

}

// Function to check if the puzzle is solved

int isSolved(int puzzle[N][N]) {

    int count = 1;

    int i,j;

    for (i = 0; i < N; i++) {

    for (j = 0; j < N; j++) {

        if (puzzle[i][j] != count && (i != N - 1 || j != N - 1)) {

                return 0; // Puzzle is not solved

            }

            count++;

        }

    }

    return 1; // Puzzle is solved

}

// Function to move the empty tile in the puzzle

void moveTile(int puzzle[N][N], int moveX, int moveY) {

    int emptyX, emptyY,i,j;

    // Find the position of the empty tile

    for (i = 0; i < N; i++) {

    for (j = 0; j < N; j++) {

            if (puzzle[i][j] == EMPTY\_TILE) {

                emptyX = i;

                emptyY = j;

                break;

            }

        }

    }

    // Swap the empty tile with the tile to be moved

    puzzle[emptyX][emptyY] = puzzle[emptyX + moveX][emptyY + moveY];

    puzzle[emptyX + moveX][emptyY + moveY] = EMPTY\_TILE;

}

int main() {

    int puzzle[N][N] = {

        {1, 2, 3, 4},

        {5, 6, 7, 8},

        {9, 10, 11, 12},

        {13, 14, 15, EMPTY\_TILE}

    };

    printf("Initial Puzzle State:\n");

    printPuzzle(puzzle);

    // Example move: Move the empty tile up

    moveTile(puzzle, -1, 0);

    printf("\nPuzzle State After Move:\n");

    printPuzzle(puzzle);

    // Check if the puzzle is solved

    if (isSolved(puzzle)) {

        printf("\nPuzzle Solved!\n");

    } else {

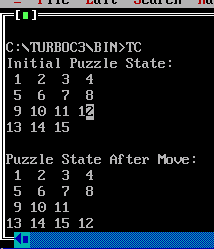
        printf("\nPuzzle Not Solved Yet.\n");

    }

    return 0;

}

**Output:**

****

**Conclusion:** The implementation of the 15 puzzle problem in C demonstrated the fundamental mechanics of puzzle manipulation and state checking. While the provided code offers a basic framework, further extensions could include implementing solving algorithms such as A\* search to find optimal solutions efficiently.