Name: Shaan Agarwal
UID: 2021300001
Subject: Design And Analysis Of Algorithms Lab
Title: Experiment 08

Aim: To implement 15 puzzle problem using branch and bound strategy

Theory:

Branch and bound is one of the techniques used for problem solving. It is similar to the backtracking since it also uses the state space tree. It is used for solving the optimization problems and minimization problems. If we have given a maximization problem then we can convert it using the Branch and bound technique by simply converting the problem into a maximization problem.

PseudoCode:

```
BranchAndBound(Problem P):
    Initialize a priority queue Q
    Enqueue P into Q with initial lower bound L(P) = 0
    Initialize a variable best_solution to infinity

    while Q is not empty:
        Dequeue the highest priority item from Q as P
        if L(P) >= best_solution:
```

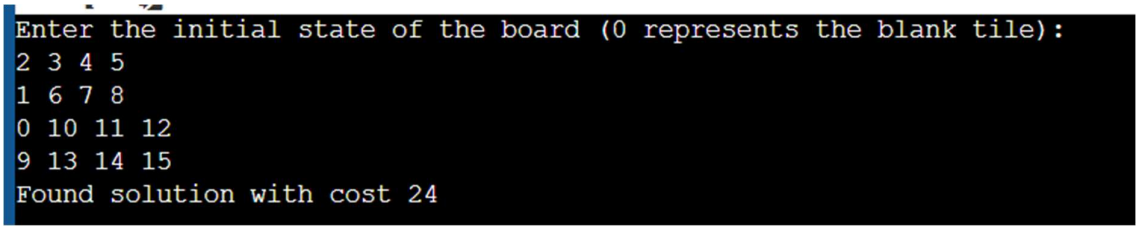
```
            continue

        if P is a complete solution:
            best_solution = P.cost

        for each child problem C of P:
            if C can't possibly lead to a better solution than the
best_solution:
                continue

            if C is a complete solution:
                best_solution = min(best_solution, C.cost)
            else:
                Compute a lower bound L(C)
                Enqueue C into Q with priority L(C)

    return best_solution
```

Program:

```
#include <bits/stdc++.h>
using namespace std;

const int INF = 1e9;

struct State {
    int board[4][4];
    int blank_row, blank_col;
    int cost, heuristic;

    State() {
        cost = 0;
        heuristic = 0;
        blank_row = 0;
        blank_col = 0;
        memset(board, 0, sizeof(board));
    }

    bool operator < (const State& other) const {
        return cost + heuristic > other.cost + other.heuristic;
    }
};

const int dx[] = {0, 0, 1, -1};
const int dy[] = {1, -1, 0, 0};

int manhattan_distance(int x1, int y1, int x2, int y2) {
    return abs(x1 - x2) + abs(y1 - y2);
}

int heuristic(const State& state) {
    int res = 0;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            if (state.board[i][j] == 0) continue;
            res += manhattan_distance(i, j, (state.board[i][j] - 1) / 4,
(state.board[i][j] - 1) % 4);
        }
    }
    return res;
}

int main() {
    State initial_state;
```

```
    cout << "Enter the initial state of the board (0 represents the blank
tile):\n";
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            cin >> initial_state.board[i][j];
            if (initial_state.board[i][j] == 0) {
                initial_state.blank_row = i;
                initial_state.blank_col = j;
            }
        }
    }

    priority_queue<State> q;
    q.push(initial_state);

    while (!q.empty()) {
        State current_state = q.top();
        q.pop();

        if (heuristic(current_state) == 0) {
            cout << "Found solution with cost " << current_state.cost << endl;
            break;
        }

        for (int k = 0; k < 4; k++) {
            int new_row = current_state.blank_row + dx[k];
            int new_col = current_state.blank_col + dy[k];

            if (new_row < 0 || new_row >= 4 || new_col < 0 || new_col >= 4)
continue;

            State new_state = current_state;
            swap(new_state.board[current_state.blank_row][current_state.blank_
col], new_state.board[new_row][new_col]);
            new_state.blank_row = new_row;
            new_state.blank_col = new_col;
            new_state.cost++;

            new_state.heuristic = heuristic(new_state);

            if (new_state.cost + new_state.heuristic >= INF) continue;

            q.push(new_state);
        }
    }

    return 0;
}
```

Output:



Conclusion: After performing the above experiment, I have understood the concept of branch and bound strategy and have applied the same for solving 15 puzzle problem.