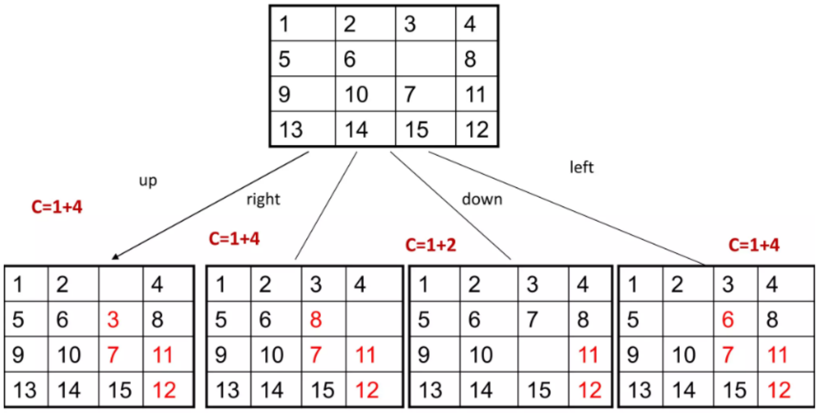
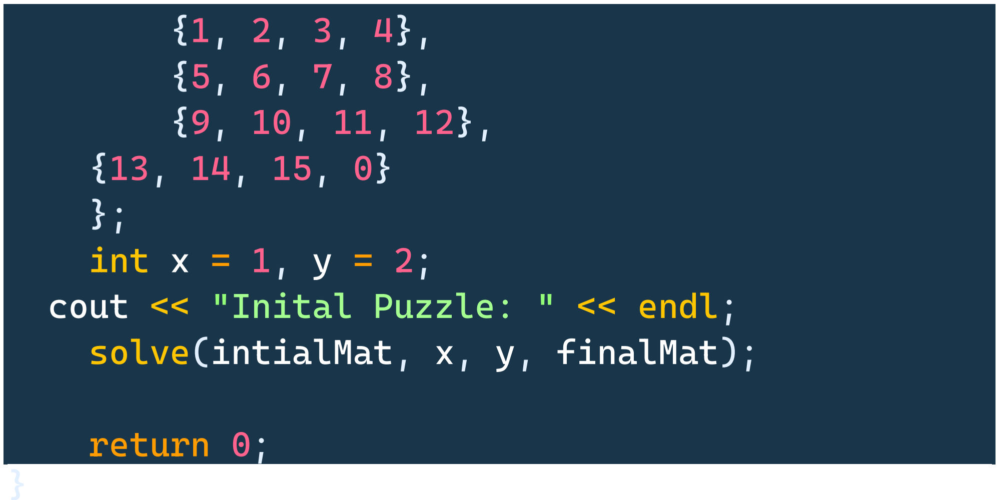
NAME:	Deepanshu Aggarwal
UID:	2021300002
BRANCH:	Computer engineering
BATCH:	A
SUBJECT:	DAA
EXPT NO:	8
AIM:	To Experiment based on branch and bound strategy (15 puzzle problem)
THEORY:	Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. Branch and Bound solve these problems relatively quickly. Given a 4×4 board with 16 tiles (every tile has one number from 1 to 15) and one empty space. The objective is to place the numbers on tiles to match the final configuration using the empty space. We can slide four adjacent (left, right, above, and below) tiles into the empty space. 

PROGRAM:

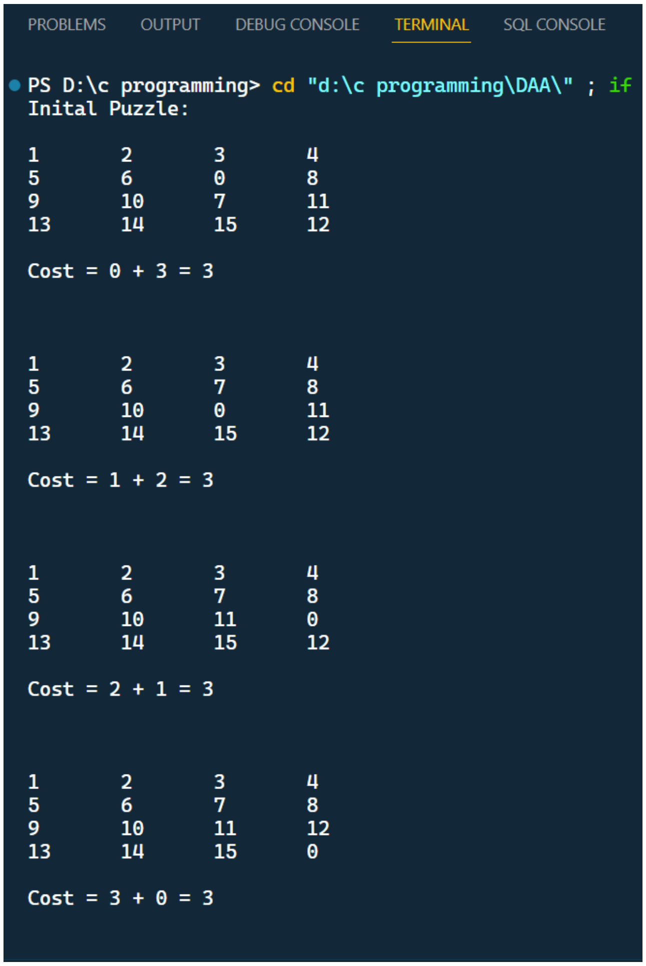
```
#include <bits/stdc++.h>
using namespace std;
#define N 4

struct Node
{
    Node* parent;
    int mat[N][N];
    int x, y;
    int cost;
    int level;
};

int printMat(int mat[N][N])
{
    printf("\n");
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf("%d\t", mat[i][j]);
        printf("\n");
    }
}

Node* newChild(int mat[N][N], int x, int y, int
newX,
            int newY, int level, Node* parent)
{
    Node* node = new Node;
    node->parent = parent;
    memcpy(node->mat, mat, sizeof node->mat);
    swap(node->mat[x][y], node-
>mat[newX][newY]);
    node->cost = INT_MAX;
    node->level = level;
    node->x = newX;
    node->y = newY;

    return node;
}

int row[] = { 1, 0, -1, 0 };
int col[] = { 0, -1, 0, 1 };

int checkCost(int intialMat[N][N], int
finalMat[N][N])
{
    int count = 0;
    for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        if (intialMat[i][j] && intialMat[i][j]
!= finalMat[i][j])
        count++;
    return count;
}

int isSafe(int x, int y)
{
    return (x >= 0 && x < N && y >= 0 && y < N);
}

void print(Node* rootNode)
{
    if (rootNode == NULL)
        return;
    print(rootNode->parent);
    printMat(rootNode->mat);
  cout << "\nCost = " << rootNode->level << " +
" << rootNode->cost  << " = " << rootNode->cost
+ rootNode->level << endl;
```

```
    printf("\n\n");
}

struct compare
{
    bool operator()(const Node* lhs, const Node*
rhs) const
    {
        return (lhs->cost + lhs->level) > (rhs-
>cost + rhs->level);
    }
};

void solve(int intialMat[N][N], int x, int y,
        int finalMat[N][N])
{

    priority_queue<Node*, std::vector<Node*>,
compare> pq;
    Node* rootNode = newChild(intialMat, x, y,
x, y, 0, NULL);
    rootNode->cost = checkCost(intialMat,
finalMat);

    pq.push(rootNode);

    while (!pq.empty())
    {

        Node* min = pq.top();
        pq.pop();

        if (min->cost == 0)
        {

            print(min);
            return;
        }

        for (int i = 0; i < 4; i++)
        {
            if (isSafe(min->x + row[i], min->y +
col[i]))
            {

                Node* child = newChild(min->mat,
min->x,
                            min->y, min->x +
row[i],
                            min->y + col[i],
                            min->level + 1,
min);
                child->cost = checkCost(child-
>mat, finalMat);
                pq.push(child);
            }
        }
    }
}

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



```
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
    {13, 14, 15, 0}
    };
    int x = 1, y = 2;
  cout << "Inital Puzzle: " << endl;
    solve(intialMat, x, y, finalMat);

    return 0;
}
```

RESULT:



```
PS D:\c programming> cd "d:\c programming\DAA\" ; if
Inital Puzzle:

1       2       3       4
5       6       0       8
9       10      7       11
13      14      15      12

Cost = 0 + 3 = 3



1       2       3       4
5       6       7       8
9       10      0       11
13      14      15      12

Cost = 1 + 2 = 3



1       2       3       4
5       6       7       8
9       10      11      0
13      14      15      12

Cost = 2 + 1 = 3



1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      0

Cost = 3 + 0 = 3
```

CONCLUSION:

Through this experiment, I learnt the concept of branch and bound algorithms and solved 15 puzzle problem using that approach.