Experiment No. : 8

Title: 15 puzzle problem using Branch and bound

Batch: A2 **Roll No.:16010421063** **Experiment No.:**

Aim: To Implement 8/15 puzzle problem using Branch and bound.

Algorithm of 15 puzzle problem using Branch and bound:

Working of 15 puzzle problem using Branch and bound:

Problem Statement
Find the following 15 puzzle problem using branch and bound technique and show each steps in detail using state space tree.

Also verify your answer by simulating steps of same question on following link.
http://www.sfu.ca/~jtmulhol/math302/puzzles-15.html

Solution

Derivation of 15 puzzle problem using Branch and bound:

Time complexity Analysis

Program(s) of 15 puzzle problem using Branch and bound:

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

bool check(vector<vector<int>> &a, vector<vector<int>> &t)
{
   for (int i = 0; i < 4; i++)
```

```
    {
        for (int j = 0; j < 4; j++)
        {
            if (a[i][j] != t[i][j])
                return false;
        }
    }
    return true;
}

int cal(vector<vector<int>> &a, vector<vector<int>> &t)
{
    int m = 0;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (a[i][j] != t[i][j])
                m++;
        }
    }
    return m;
}

int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    vector<vector<int>> a(4, vector<int>(4));
    vector<vector<int>> t = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 0}};
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            cin >> a[i][j];
```

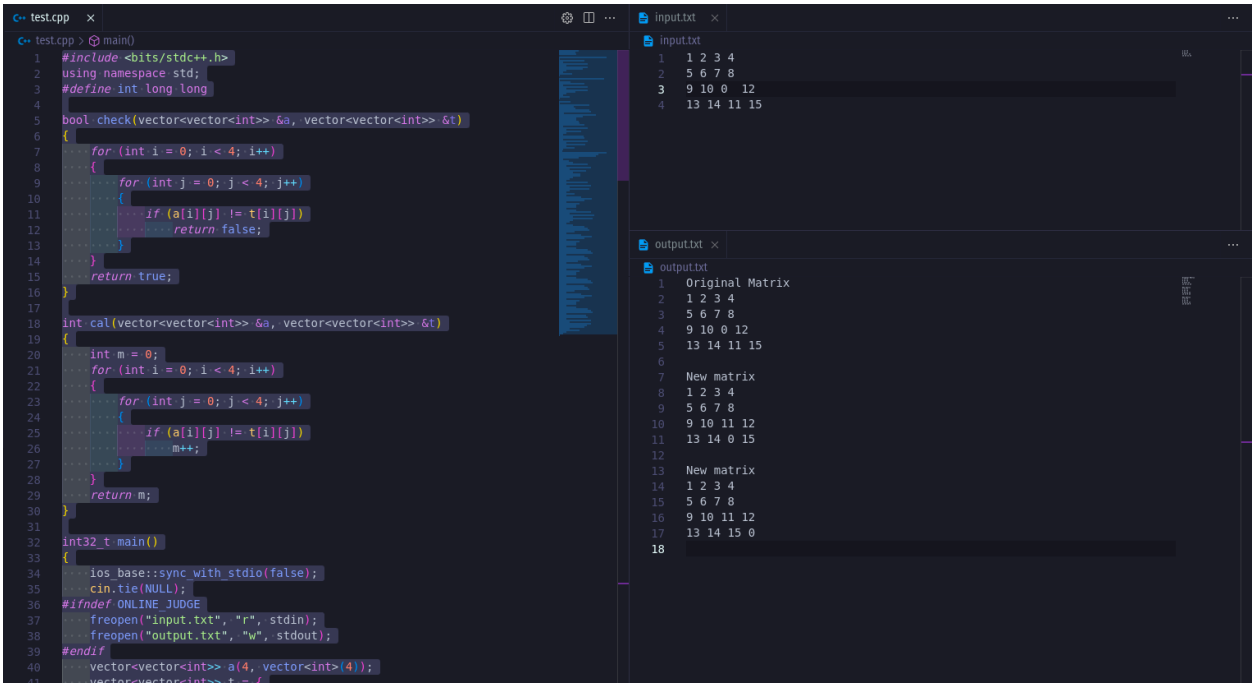
```
    }
}
cout<<"Original Matrix\n";
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < 4; j++)
    {
        cout<< a[i][j]<<" ";
    }
    cout<<"\n";
}

int p, l = 0, d = 1000, x = 0, y = 0, m = 0, dmin = 0;
vector<vector<int>> temp(4, vector<int>(4));
vector<vector<int>> r(4, vector<int>(4));
while (!check(a, t))
{
    l++;
    d = 1000;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (a[i][j] == 0)
            {
                x = i;
                y = j;
            }
        }
    }
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            temp[i][j] = a[i][j];
        }
    }
    if (x != 0)
    {
        p = temp[x][y];
        temp[x][y] = temp[x - 1][y];
        temp[x - 1][y] = p;
    }
```

```
        m = cal(temp, t);
        dmin = 1 + m;
        if (dmin < d)
        {
            d = dmin;
            for (int i = 0; i < 4; i++)
            {
                for (int j = 0; j < 4; j++)
                {
                    r[i][j] = temp[i][j];
                }
            }
        }
        int i, j;
        for (i = 0; i < 4; i++)
            for (j = 0; j < 4; j++)
                temp[i][j] = a[i][j];
        if (x != 3)
        {
            p = temp[x][y];
            temp[x][y] = temp[x + 1][y];
            temp[x + 1][y] = p;
        }
        m = cal(temp, t);
        dmin = 1 + m;
        if (dmin < d)
        {
            d = dmin;
            for (i = 0; i < 4; i++)
                for (j = 0; j < 4; j++)
                    r[i][j] = temp[i][j];
        }
        for (i = 0; i < 4; i++)
            for (j = 0; j < 4; j++)
                temp[i][j] = a[i][j];
        if (y != 4 - 1)
        {
            p = temp[x][y];
            temp[x][y] = temp[x][y + 1];
            temp[x][y + 1] = p;
        }
        m = cal(temp, t);
        dmin = 1 + m;
```

```
        if (dmin < d)
        {
            d = dmin;
            for (i = 0; i < 4; i++)
                for (j = 0; j < 4; j++)
                    r[i][j] = temp[i][j];
        }
        for (i = 0; i < 4; i++)
            for (j = 0; j < 4; j++)
                temp[i][j] = a[i][j];
        if (y != 0)
        {
            p = temp[x][y];
            temp[x][y] = temp[x][y - 1];
            temp[x][y - 1] = p;
        }
        m = cal(temp, t);
        dmin = l + m;
        if (dmin < d)
        {
            d = dmin;
            for (i = 0; i < 4; i++)
                for (j = 0; j < 4; j++)
                    r[i][j] = temp[i][j];
        }
        cout << "\nNew matrix\n";
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                cout << r[i][j] << " ";
                a[i][j] = r[i][j];
                temp[i][j] = 0;
            }
            cout << "\n";
        }
    }
}
```

Output(o) of 15 puzzle problem using Branch and bound:



```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long

bool check(vector<vector<int>> &a, vector<vector<int>> &t)
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (a[i][j] != t[i][j])
                return false;
        }
    }
    return true;
}

int cal(vector<vector<int>> &a, vector<vector<int>> &t)
{
    int m = 0;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (a[i][j] != t[i][j])
                m++;
        }
    }
    return m;
}

int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    vector<vector<int>> a(4, vector<int>(4));
```

input.txt

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

output.txt

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

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

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

Post Lab Questions:- Explain how to solve the Knapsack problem using branch and bound.

The Knapsack problem involves selecting a set of items with given weights and values, such that the total weight of the selected items is less than or equal to a given limit, and the total value is maximized. Branch and bound is an algorithmic technique that can be used to solve this problem. The basic idea behind branch and bound is to divide the problem into smaller subproblems, solve each subproblem recursively, and keep track of the best solution found so far. At each step, the algorithm chooses a subset of the remaining items to include in the knapsack, and branches into two subproblems: one where the selected item is included, and one where it is not. The algorithm then recursively solves each subproblem, keeping track of the best solution found so far. The algorithm prunes branches that cannot possibly yield a better solution than the best solution found so far. This is done by calculating an upper bound on the value of the remaining items, and comparing it to the best solution found so far. If the upper bound is less than the best solution, the branch is pruned. This process continues until all subproblems have been solved, and the best solution found is returned.

Conclusion: (Based on the observations):
Implemented the 15 puzzle problem

Outcome:CO3 Implement Backtracking and Branch-and-bound algorithms

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.