Suraj Iyer
2021300045
SE Comps A, Batch C

DAA Experiment 8

Aim - To solve the 15 puzzle problem using branch and bound.

Theory - Branch and bound algorithms are used to find the optimal solution for combinatory, discrete, and general mathematical optimization problems. In general, given an NP-Hard problem, a branch and bound algorithm explores the entire search space of possible solutions and provides an optimal solution. A branch and bound algorithm consist of stepwise enumeration of possible candidate solutions by exploring the entire search space. With all the possible solutions, we first build a rooted decision tree. The root node represents the entire search space: each child node is a partial solution and part of the solution set. Before constructing the rooted decision tree, we set an upper and lower bound for a given problem based on the optimal solution. At each level, we need to make a decision about which node to include in the solution set. At each level, we explore the node with the best bound. In this way, we can find the best and optimal solution fast.

Now it is crucial to find a good upper and lower bound in such cases. We can find an upper bound by using any local optimization method or by picking any point in the search space. On the other hand, we can obtain a lower bound from convex relaxation or duality.

In general, we want to partition the solution set into smaller subsets of the solution. Then we construct a rooted decision tree, and finally, we choose the best possible subset (node) at each level to find the best possible solution set.

If the given problem is a discrete optimization problem, a branch and bound is a good choice. Discrete optimization is a subsection of optimization where the variables in the problem should belong to the discrete set. Examples of such problems are 0-1 Integer Programming or Network Flow problem.

Branch and bound work efficiently on the combinatorial optimization problems. Given an objective function for an optimization problem, combinatorial optimization is a process to find the maxima or minima for the objective function. The domain of the objective function should be discrete and large. Boolean Satisfiability, Integer Linear Programming are examples of the combinatorial optimization problems.

Algorithm -

Algorithm 1: Job Assignment Problem Using Branch And Bound

```
Data: Input cost matrix M[][]
Result: Assignment of jobs to each worker according to optimal
        cost
Function MinCost(M[][])
while True do
    E = LeastCost();
    if E is a leaf node then
        print();
        return;
    end
    for each child S of E do
        Add(S);
        S → parent = E;
    end
end
```

Code -

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

int m = 0, n = 4;

int cal(int temp[10][10], int t[10][10])
{
   int i, j, m = 0;
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
      {
         if (temp[i][j] != t[i][j])
            m++;
      }
   return m;
}

int check(int a[10][10], int t[10][10])
{
```

```
    int i, j, f = 1;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (a[i][j] != t[i][j])
                f = 0;
    return f;
}

void main()
{
    int p, i, j, n = 4, a[10][10], t[10][10], temp[10][10], r[10][10];
    int m = 0, x = 0, y = 0, d = 1000, dmin = 0, l = 0;

    printf("\nEnter the matrix to be solved,space with zero :\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &a[i][j]);

    printf("\nEnter the target matrix,space with zero :\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &t[i][j]);

    printf("\nEntered Matrix is :\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            printf("%d\t", a[i][j]);
        printf("\n");
    }

    printf("\nTarget Matrix is :\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            printf("%d\t", t[i][j]);
        printf("\n");
    }
```

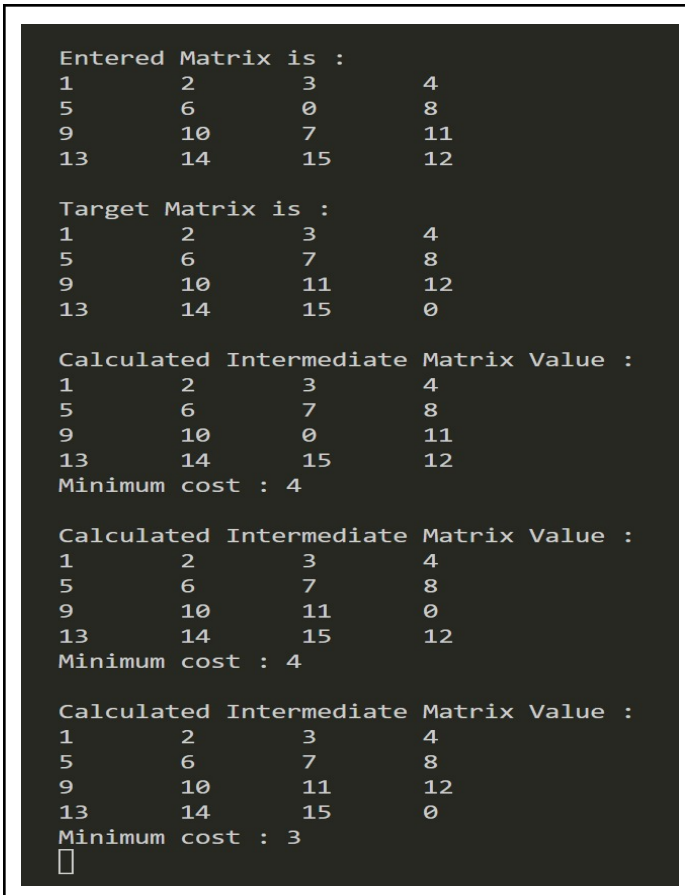
```
while (!(check(a, t)))
{
   l++;
   d = 1000;
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
      {
         if (a[i][j] == 0)
         {
            x = i;
            y = j;
         }
      }

   // To move upwards
   for (i = 0; i < n; i++)
      for (j = 0; j < n; 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 = l + m;
   if (dmin < d)
   {
      d = dmin;
      for (i = 0; i < n; i++)
         for (j = 0; j < n; j++)
            r[i][j] = temp[i][j];
   }

   // To move downwards
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         temp[i][j] = a[i][j];
```

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

// To move right side
for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
      temp[i][j] = a[i][j];
if (y != n - 1)
{
   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 < n; i++)
      for (j = 0; j < n; j++)
         r[i][j] = temp[i][j];
}

// To move left
for (i = 0; i < n; i++)
   for (j = 0; j < n; 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 < n; i++)
                for (j = 0; j < n; j++)
                    r[i][j] = temp[i][j];
        }

        printf("\nCalculated Intermediate Matrix Value :\n");
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
                printf("%d\t", r[i][j]);
            printf("\n");
        }
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
            {
                a[i][j] = r[i][j];
                temp[i][j] = 0;
            }
        printf("Minimum cost : %d\n", d);
    }
    getch();
}
```

Output -

```
Enter the matrix to be solved,space with zero :
1
2
3
4
5
6
0
8
9
10
7
11
13
14
15
12

Enter the target matrix,space with zero :
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0

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



Conclusion - I have implemented the 15 puzzle problem using the branch and bound approach and understood how it optimizes the solution.