Experiment No. : 8

Title: 15 puzzle problem using Branch and bound

Batch: A2 **Roll No.:** 16010421073 **Experiment No.:** 8

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

Algorithm of 15 puzzle problem using Branch and bound:

Step 1: Check the position of the empty box
Step 2: Swap the position of the element neighbouring to the box. If the position is at the extreme corner then don't swap.
Step 3: Select the solution which has optimal cost i.e Minimum cost.
Step 4: Check that if the position is swap then don't swap the empty box with that same element.
Step 5: Continue till the puzzle gets solve.

Working of 15 puzzle problem using Branch and bound:

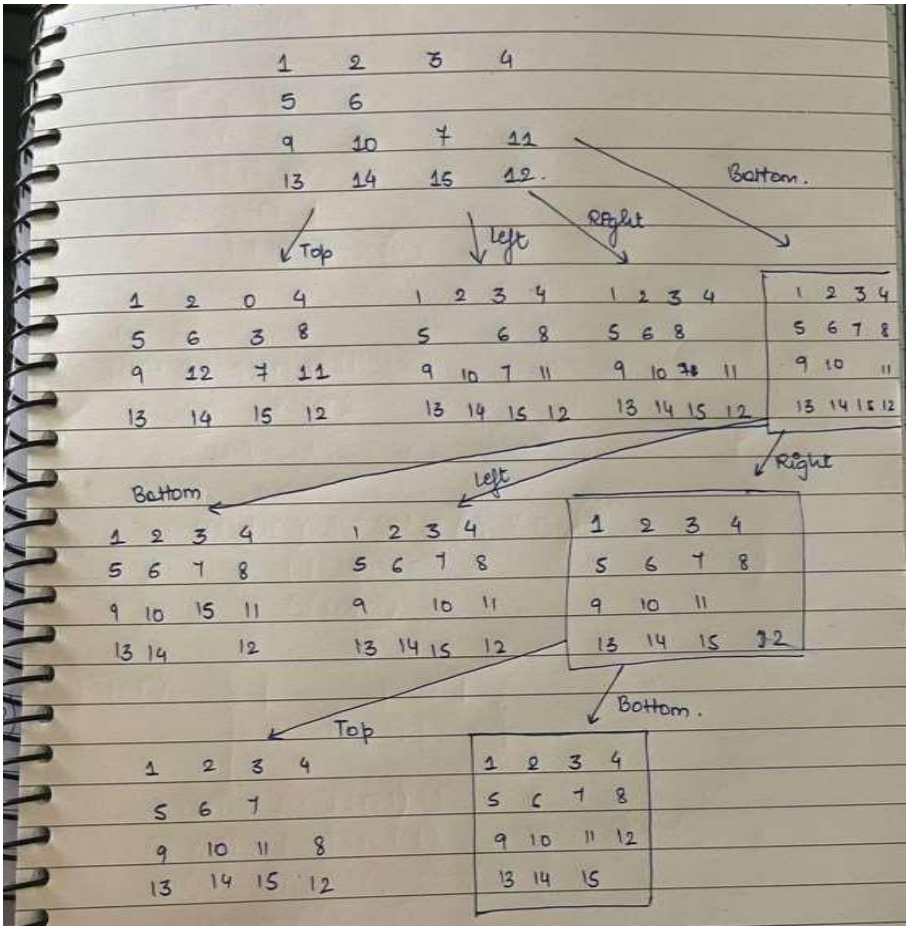
Firstly, we check the empty box having no element. Swap the empty element with all the neighbouring element and check the solution having lowest cost i.e optimal cost. Once if the position is swap with the element don't swap the empty box with that same element. If the empty box is on the extreme boundary then don't swap the element. Continue this process till all the elements are placed in correct order. This is the basic working of 15 puzzle problem using Branch and Bound method.

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

The time complexity of the algorithm is O(2^n),
where n is the level of the state space tree.

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

```
#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 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 :\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;
                        }
                }

        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];
```

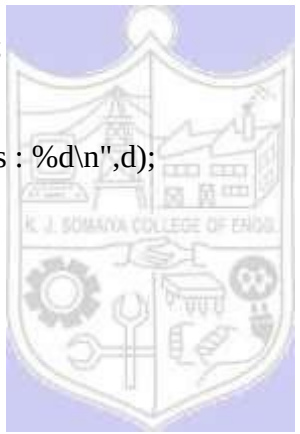
```
for(i=0;i < n;i++)
        for(j=0;j < n;j++)
                temp[i][j]=a[i][j];
if(x!=n-1)
{
        p=temp[x][y];
        temp[x][y]=temp[x+1][y];
        temp[x+1][y]=p;
}
m=cal(temp,t);
dmin=l+m;
if(dmin < d)
{
        d=dmin;
        for(i=0;i < n;i++)
                for(j=0;j < n;j++)
  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

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];
}


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("\nIntermediate Matrix :\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("Optimal cost is : %d\n",d);
        }
        getch();
}
```



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

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

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

Intermediate Matrix :
1       2       3       4
5       6       7       8
9       10      0       11
13      14      15      12
Optimal cost is : 4

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

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

Post Lab Questions:-

Explain how to solve the Knapsack problem using branch and bound.

Answer:

Given two integer arrays val[0..n-1] and wt[0..n-1] that represent values and weights associated with n items respectively. Find out the maximum value subset of val[] such that sumof the weights of this subset is smaller than or equal to Knapsack capacity W. Let us exploreall approaches for this problem.

1. A Greedy approach is to pick the items in decreasing order of value per unit weight. TheGreedy approach works only for fractional knapsack problem and may not produce correctresult for 0/1 knapsack.
2. We can use Dynamic Programming (DP) for 0/1 Knapsack problem. In DP, we use a 2Dtable of size n x W. The DP Solution doesn't work if item weights are not integers.
3. Since DP solution doesn't always work, a solution is to use Brute Force. With n items,there are 2^n solutions to be generated, check each to see if they satisfy the constraint, savemaximum solution that satisfies constraint. This solution can be expressed as tree.
4. We can use Backtracking to optimize the Brute Force solution. In the tree representation,we can do DFS of tree. If we reach a point where a solution no longeris feasible, there isno need to continue exploring. In the given example, backtrackingwould be much more effective if we had even more items or a smaller knapsack capacity.

Conclusion: (Based on the observations): 15 puzzle problem has been understood and implemented its solution using branch and bound successfully.

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.