Experiment No. : 8

Title: 15 puzzle problem using Branch and bound

Batch:B1 **Roll No.:16010420133** **Experiment No.: 8**

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:

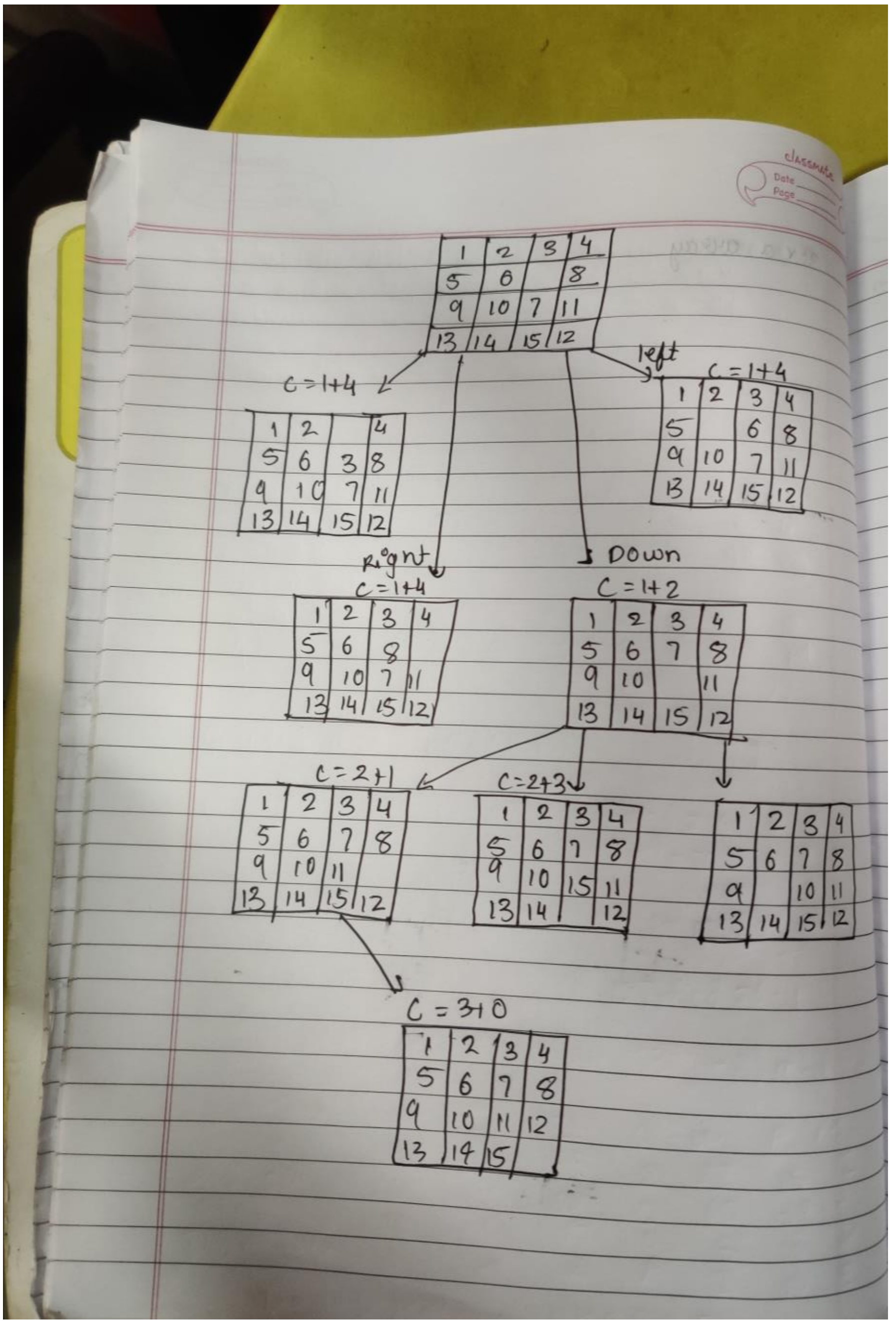
Problem Statement

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

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

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

Time complexity.
Best Case: $O(n^2)$
Worst Case:$O(n^3)$

- The algorithm presented uses an A* search to find the solution to the $(N^2 - 1)$-puzzle: arranging the numbers in order with a blank in the last location.
- The data structure used to efficiently solve the A* algorithm is a modified heap which is able to allow the user to update the priority in **O**(ln(n)) time: a index to each entry is stored in a hash table and when the priority is updated, the index allows the heap to, if necessary, percolate the object up.
- Hence 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;
}
```

```
int 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;
				}
			}


		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];
			}

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

```
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

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

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

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

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

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

Step 1:

Draw a table say 'T' with (n+1) number of rows and (w+1) number of columns. • Fill all the boxes of 0th row and 0th column with zeroes as shown

	0	1	2	3		W
0	0	0	0	0	......	0
1	0					
2	0					
	......					
n	0					

T-Table

Step 2:

Start filling the table row wise top to bottom from left to right.

Use the following formula

$T(i, j) = \max\{ T(i-1, j), value_i + T(i-1, j - weight_i) \}$

Here, T(i , j) = maximum value of the selected items if we can take items 1 to i and have weight restrictions of j.

- This step leads to completely filling the table.
- Then, value of the last box represents the maximum possible value that can be put into the knapsack.

Step 3:

To identify the items that must be put into the knapsack to obtain that maximum profit, • Consider the last column of the table.

- Start scanning the entries from bottom to top.
- On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
- After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.

Conclusion: (Based on the observations):

CO4: Understand Backtracking and Branch-and-bound algorithms.

Outcome:

Thus, we have studied about Backtracking and Branch and Bound and implemented the 15puzzle Problem using this**.**

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.