Experiment No : 8

Name: Shawn Louis

Roll No: 31

Batch: B

Problem Statement:

To implement 15 Puzzle Problem

Using Branch & Bound Strategy

Objective:

- To be able to implement a problem using branch & bound strategy

Expected Outcome:

- **Ability to explain the problem statement**
- **Ability to build a puzzle using the specified strategy**
- **Ability to differentiate between brute force technique and branch and bound for the given problem.**

Theory:

Problem Definition :

The 15 puzzle problem is invented by sam loyd in 1878.

- In this problem there are 15 tiles, which are numbered from 0 – 15.
- The objective of this problem is to transform the arrangement of tiles from initial arrangement to a goal arrangement.
- The initial and goal arrangement is shown by following figure.

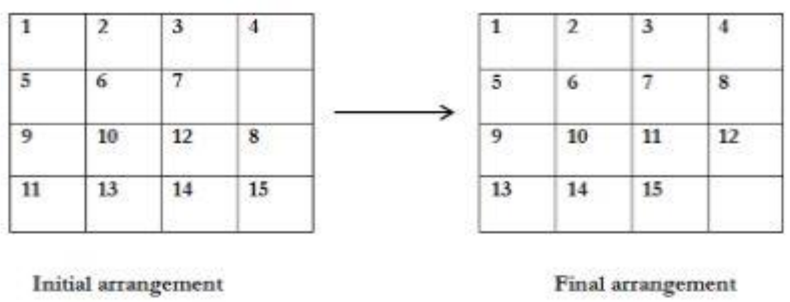

Figure 12

- There is always an empty slot in the initial arrangement.
- The legal moves are the moves in which the tiles adjacent to ES are moved to either left, right, up or down.
- Each move creates a new arrangement in a tile.
- These arrangements are called as states of the puzzle.
- The initial arrangement is called as initial state and goal arrangement is called as goal state.

- The state space tree for 15 puzzle is very large because there can be 16! Different arrangements.
- A partial state space tree can be shown in figure.
- In state space tree, the nodes are numbered as per the level.
- Each next move is generated based on empty slot positions.
- Edges are label according to the direction in which the empty space moves.
- The root node becomes the E – node.
- The child node 2, 3, 4 and 5 of this E – node get generated.
- Out of which node 4 becomes an E – node. For this node the live nodes 10, 11, 12 gets generated.
- Then the node 10 becomes the E – node for which the child nodes 22 and 23 gets generated.
- Finally we get a goal state at node 23.
- We can decide which node to become an E – node based on estimation formula.

Estimation formula:

C (x) = f (x) + G (x)

Where, f (x) is length of path from root to node x.

G (x) is number of non-blank tiles which are not in their goal position for node x.

C (x) denotes the lower bound cost of node x.

State Space Tree:

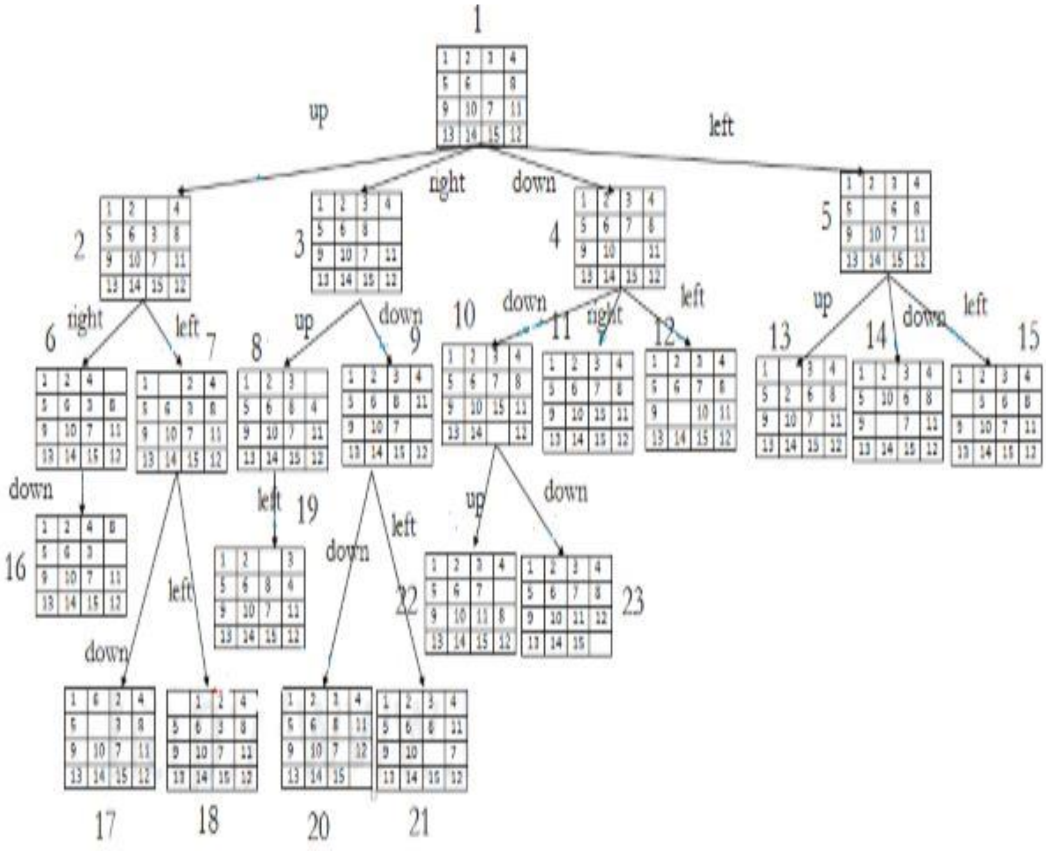

Figure 13

Algorithm:

We assume that moving one tile in any direction will have 1 unit cost. Keeping that in mind, we define a cost function for the 15-puzzle algorithm as below:

```
   c(x) = f(x) + h(x) where
   f(x) is the length of the path from root to x
        (the number of moves so far) and
   h(x) is the number of non-blank tiles not in
        their goal position (the number of mis-
        -placed tiles). There are at least h(x)
        moves to transform state x to a goal state
```

Program 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;
	clrscr();
	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 Snapshot:

```
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
```

Application	The numbers in the rows, columns and both main diagonals all add up to 30. Such an arrangement is called a Magic Square. There are 7040 different magic squares, but only 3520 of them can be solved with the blank in the lower right corner - as shown at the top of the page. The one shown above is the **only one which is solvable within 35 moves** and there are no magic squares which are solvable with fewer moves.
Outcome:	Successfully solved 15 puzzle problem using branch and bound strategy in C.