NAME:	Kunal Bhatia
UID:	2021300010
SUBJECT:	DAA
EXPERIMENT NO:	8
DATE OF PERFORMANCE:	12/04/23
DATE OF SUBMISSION:	19/04/23
AIM:	**Experiment based on branch and bound strategy (15 puzzle problem)**
Theory:	Branch and bound (BB, B&amp;B, or BnB) is a method for solving optimization problems by breaking them down into smaller sub-problems and using a bounding function to eliminate sub-problems that cannot contain the optimal solution. It is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as mathematical optimization. A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. The algorithm explores branches of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm.

The algorithm depends on efficient estimation of the lower and upper bounds of regions/branches of the search space.

If no bounds are available, the algorithm degenerates to an exhaustive search.

The method was first proposed by Ailsa Land and Alison Doig whilst carrying out research at the London School of

Economics sponsored by British Petroleum in 1960 for discrete programming, and has become the most commonly

used tool for solving NP-hard optimization problems. The name &quot;branch and bound&quot; first occurred in the work of Little

et al. on the traveling salesman problem.

The 15 puzzle (also called Gem Puzzle, Boss Puzzle, Game of Fifteen, Mystic Square and many others) is a sliding puzzle

having 15 square tiles numbered 1–15 in a frame that is 4 tile positions high and 4 positions wide (for a total of 16

positions), leaving one unoccupied position. Tiles in the same row or column of the open position can be moved by

sliding them horizontally or vertically, respectively. The goal of the puzzle is to place the tiles in numerical order.

Named for the number of tiles in the frame, the 15 puzzle may also be called a 16 puzzle, alluding to its total tile

capacity. Similar names are used for different sized variants of the 15 puzzle, such as the 8 puzzle which has 8 tiles in a

3×3 frame.

The n puzzle is a classical problem for modelling algorithms involving heuristics. Commonly used heuristics for this

problem include counting the number of misplaced tiles and finding the sum of the taxicab distances between each

block and its position in the goal configuration.[1] Note that both are admissible. That is, they never overestimate the

number of moves left, which ensures optimality for certain search algorithms such as A*.

Algorithm:

1. Define N as the size of the puzzle matrix.

2. Define a struct Node to represent each state of the puzzle.

3. Define a function printMat to print the puzzle matrix.

4. Define a function newChild to generate a new state of the puzzle by swapping two elements.

5. Define row and col arrays to represent possible moves in the puzzle.

6. Define a function checkCost to calculate the number of misplaced tiles in the puzzle.

7. Define a function isSafe to check if a move is valid within the puzzle boundaries.

8. Define a function print to print the solution path.

9. Define a compare function to compare Nodes based on their cost and level.

10. Define a function solve to find the solution for the puzzle.

11. Initialize the priority queue pq with the root node of the puzzle.

12. While pq is not empty:

a. Pop the node with the minimum cost from pq.

b. If the cost of the popped node is 0, print the solution path and return.

c. Generate child nodes by swapping elements in the puzzle matrix and calculate their cost.

d. Push the child nodes into pq.

13. End the loop.

14. Define the initial and final state of the puzzle matrices.

15. Call the solve function with the initial matrix, starting position (x, y), and the final matrix.

Program:

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

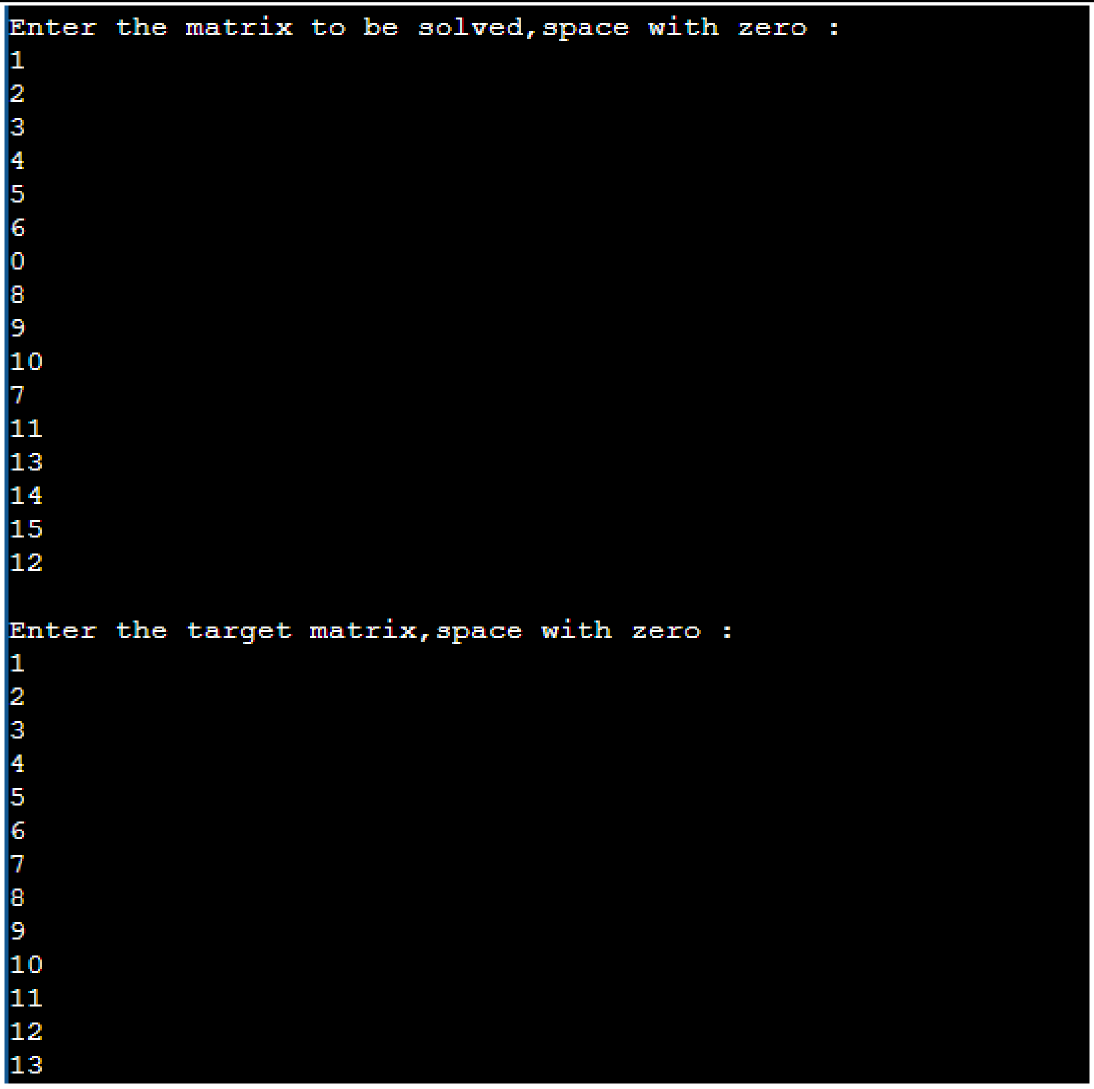
```c
int m=0,n=4;


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");
```

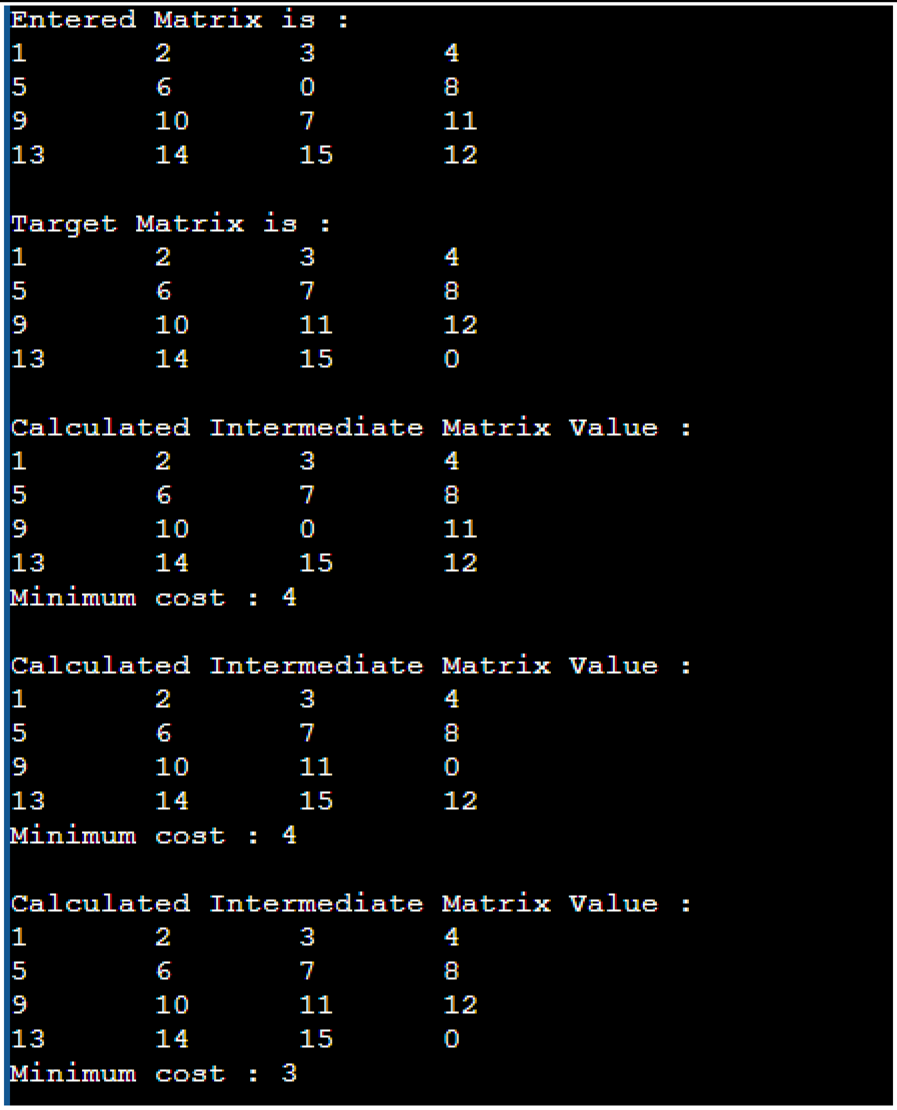
```
for(i=0;i < n;i++)
{
    for(j=0;j < n;j++)
        printf("%d\t",t[i][j]);
    printf("\n");
}

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);

  }

}
```

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
```

	
Conclusion:	I understood the technique and concept to solve 15 puzzle problem.

```
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
```