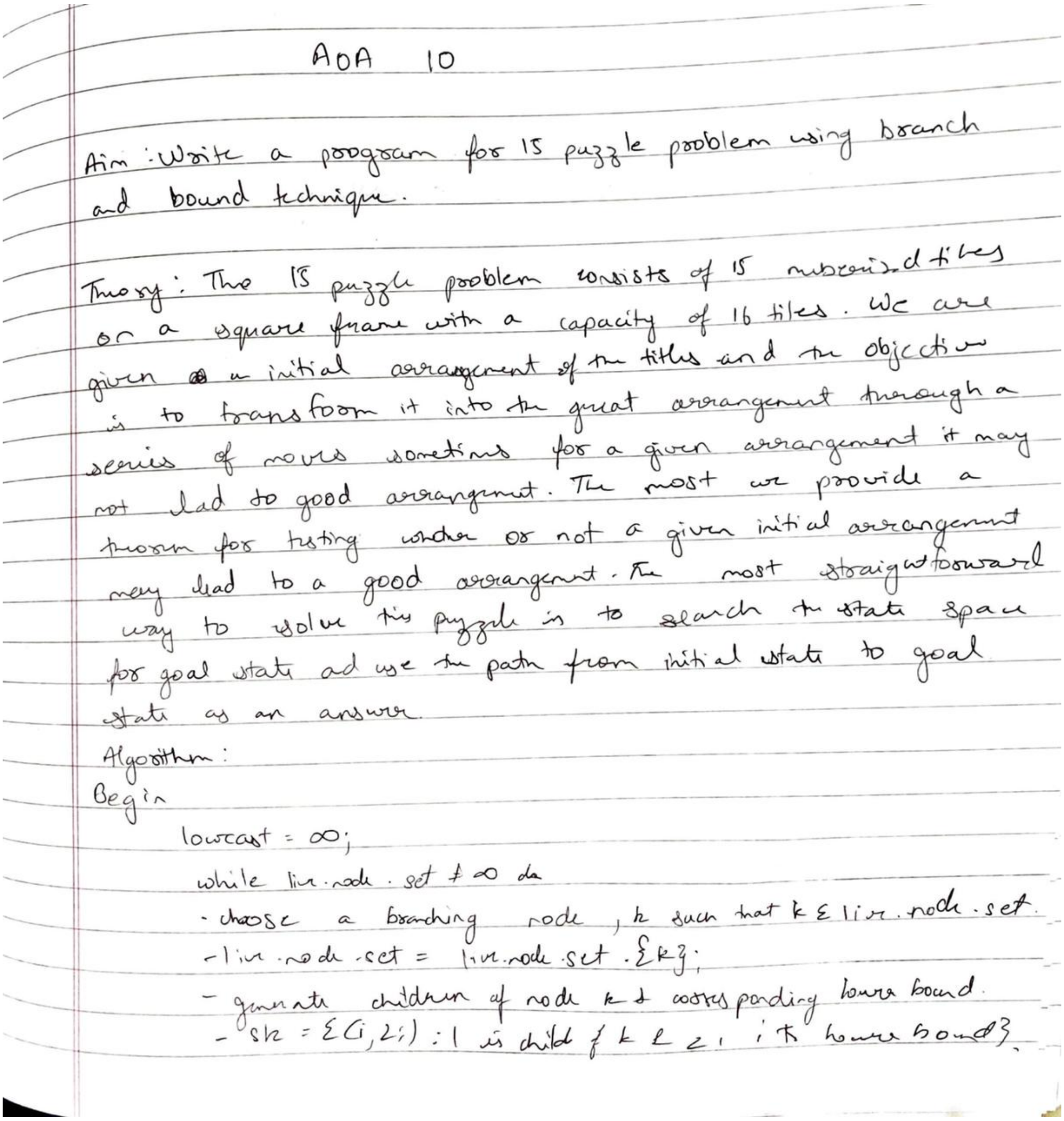
Experiment no: 10

Aim: Write a program for 15-puzzle problem with branch and bound technique.

Software required: C

Theory:



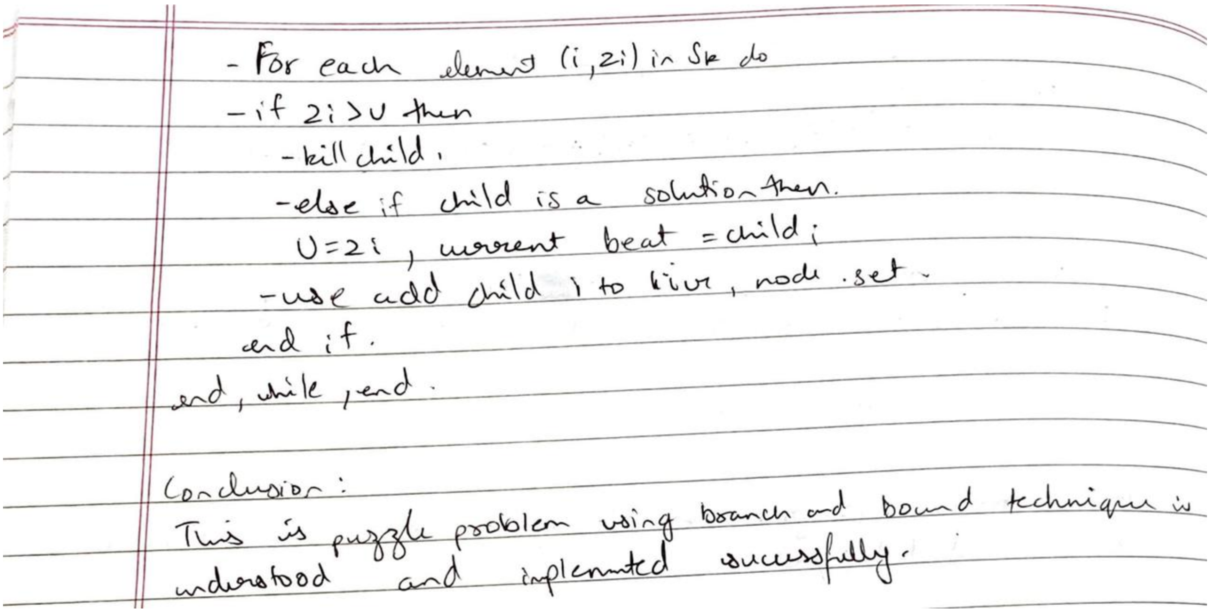
AOA 10

Aim : Write a program for 15 puzzle problem using branch and bound technique.

Theory : The 15 puzzle problem consists of 15 numbered tiles on a square frame with a capacity of 16 tiles. We are given a initial arrangement of the titles and the objective is to transform it into the great arrangement through a series of moves sometimes for a given arrangement it may not lead to good arrangement. The most we provide a theorem for testing wether or not a given initial arrangement may lead to a good arrangement. The most straigwtforward way to solve this puzzle is to search the state space for goal state ad use the path from initial state to goal state as an answer.

Algorithm :

```
Begin
        lowcast = $\infty$;
        while live.node.set $\neq \infty$ do
        - choose a branching node, k such that $k \in$ live.node.set.
        - live.node.set = live.node.set . $\{k\}$;
        - generate children of node k & corresponding lower bound.
        - sk = $\{(i, z_i)$ : i is child of k & $z_i$ its lower bound$\}$.
```



```
- For each element (i, zi) in Sk do
- if zi > U then
    - kill child.
    - else if child is a solution then.
      U=zi , current beat =child;
   -use add child i to live, node .set.
  end if.
end, while, end.
```

Conclusion:

This is puzzle problem using branch and bound technique is understood and implemented successfully.

Source Code:

```
#include<stdio.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;

}

}

//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;

}
```

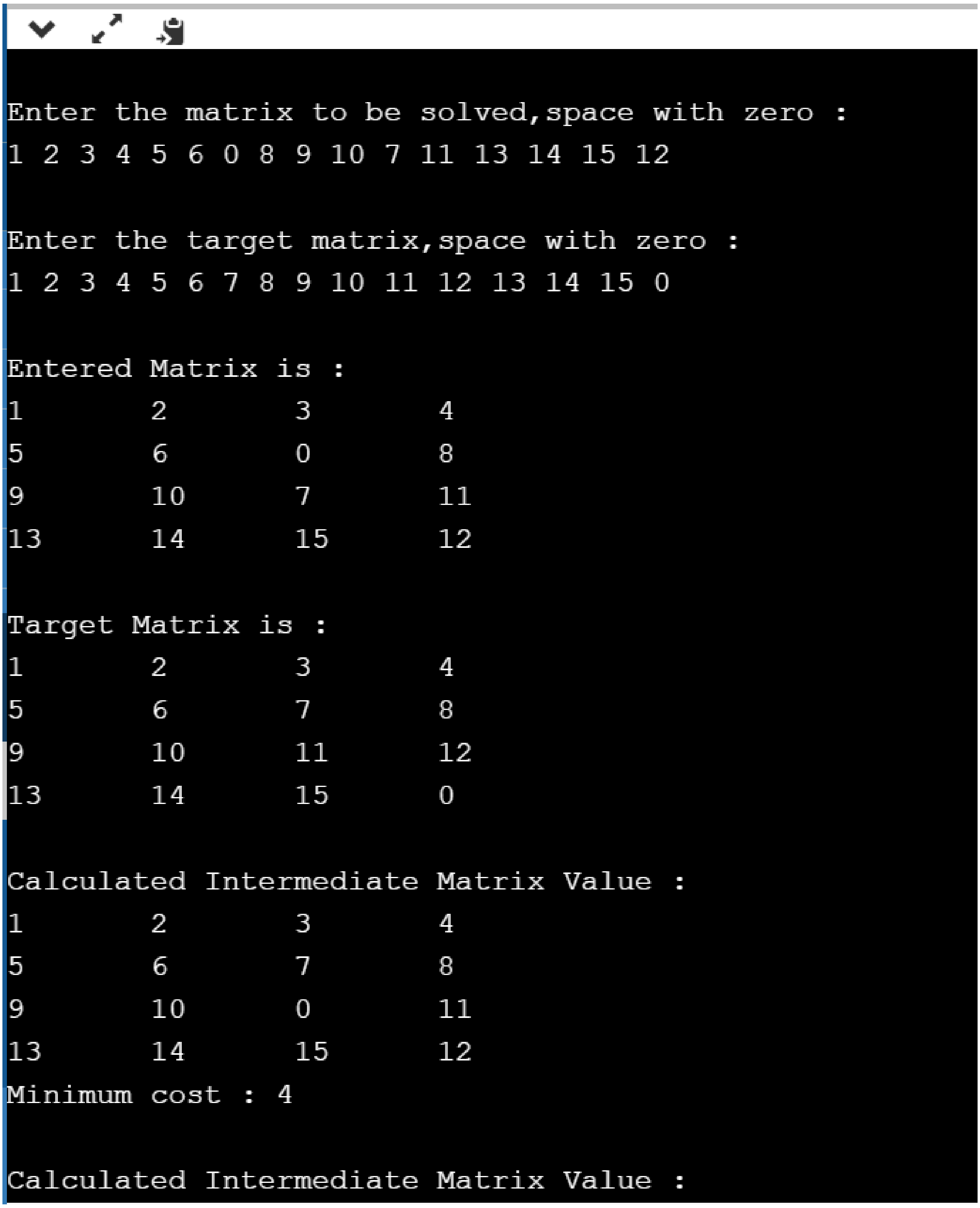
```
m=cal(temp,t);

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);
}
return 0;
}
```

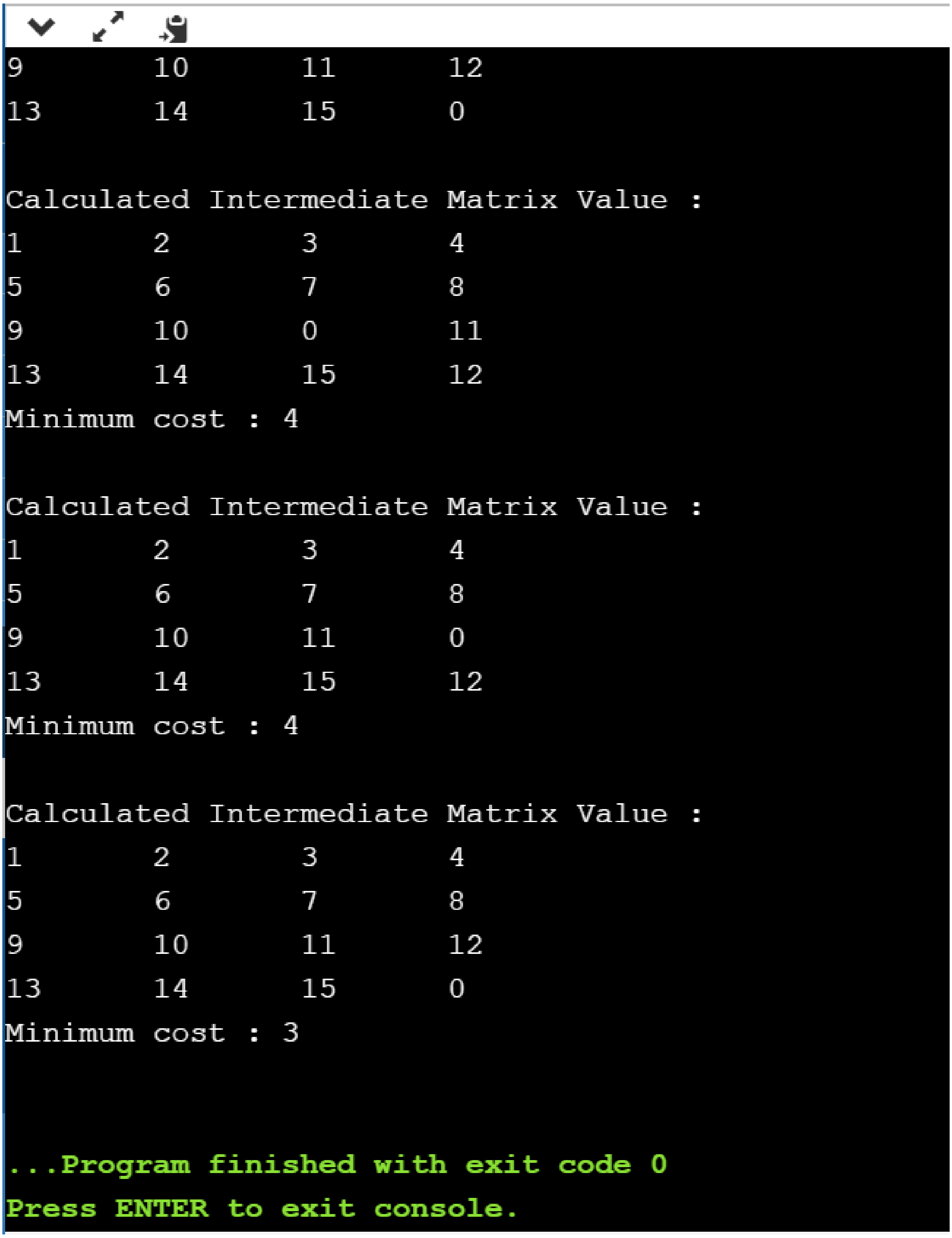
Analysis:

Algorithm	15-puzzle problem
Time Complexity	$O(n^2)$
Space Complexity	O(n)

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

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 :
```



```
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



...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion: 15-puzzle problem has been studied and implemented.