Practical: 04

Aim: Write a program to implement Heuristic(Steepest Ascent)Search for 8 puzzle game problem.

Theory :

In artificial intelligence, a **heuristic** is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.

A **heuristic function**, also called simply a **heuristic**, is a function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow. For example, it may approximate the exact solution.

Steepest-Ascent Hill climbing: It first examines all the neighboring nodes and then selects the node closest to the solution state as of next node.

Heuristic is a technique designed to solve a problem quickly, when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution.

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

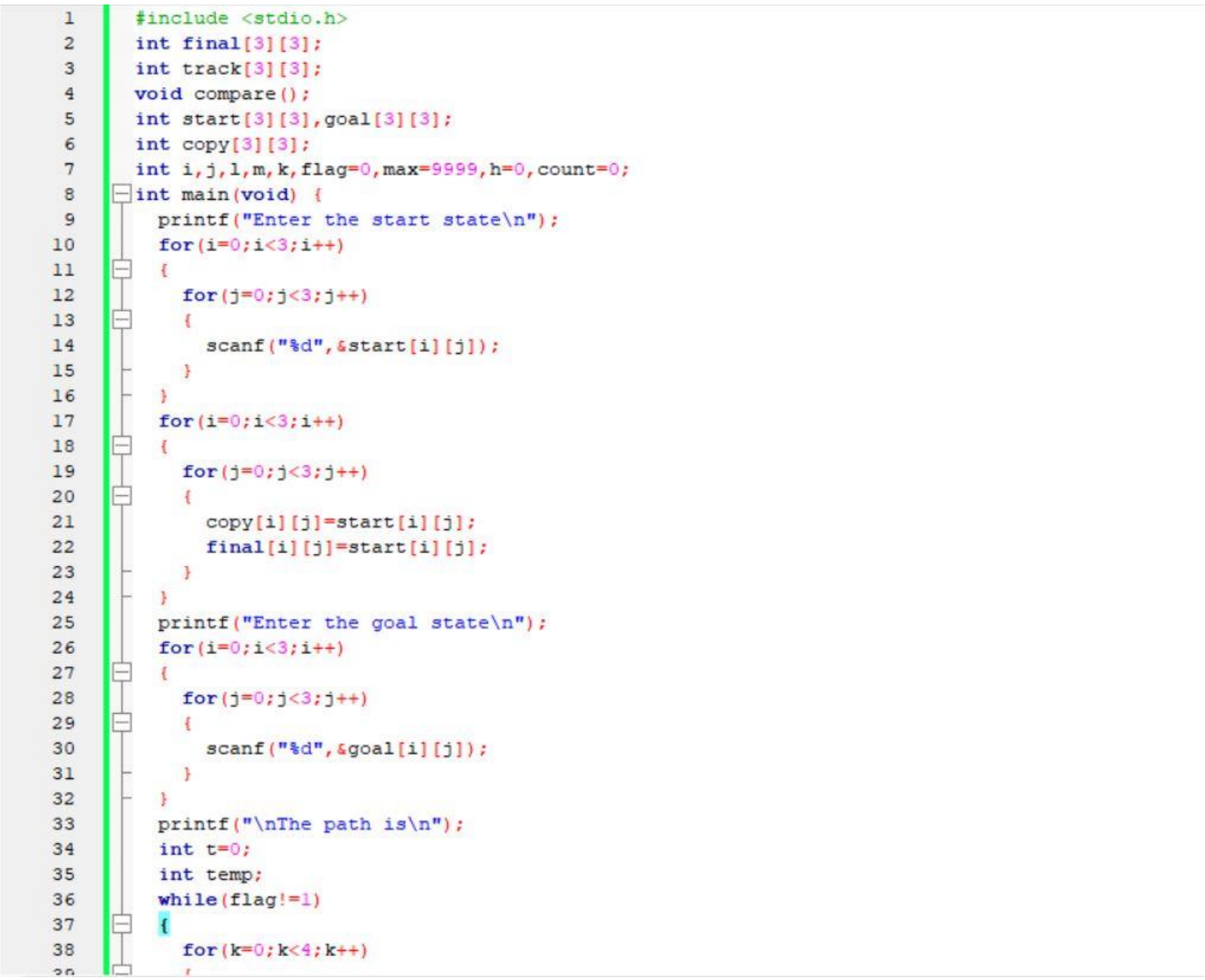
The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

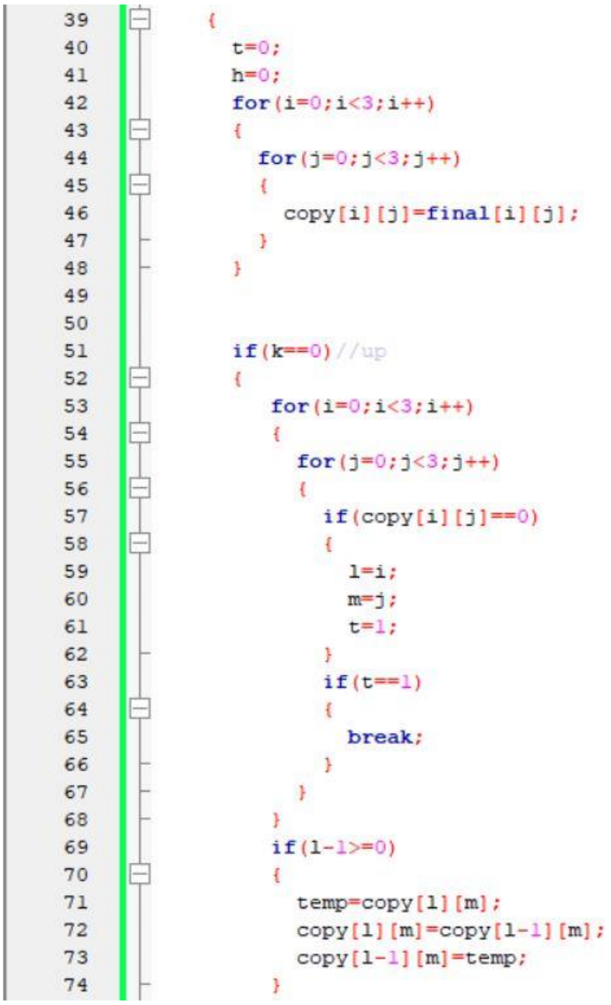
- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
    a. Let SUCC be a state such that any successor of the current state will be better than it.
    b. For each operator that applies to the current state:
        a. Apply the new operator and generate a new state.
        b. Evaluate the new state.
        c. If it is goal state, then return it and quit, else compare it to the SUCC.
        d. If it is better than SUCC, then set new state as SUCC.
        e. If the SUCC is better than the current state, then set current state to SUCC.

Step 5: Exit.

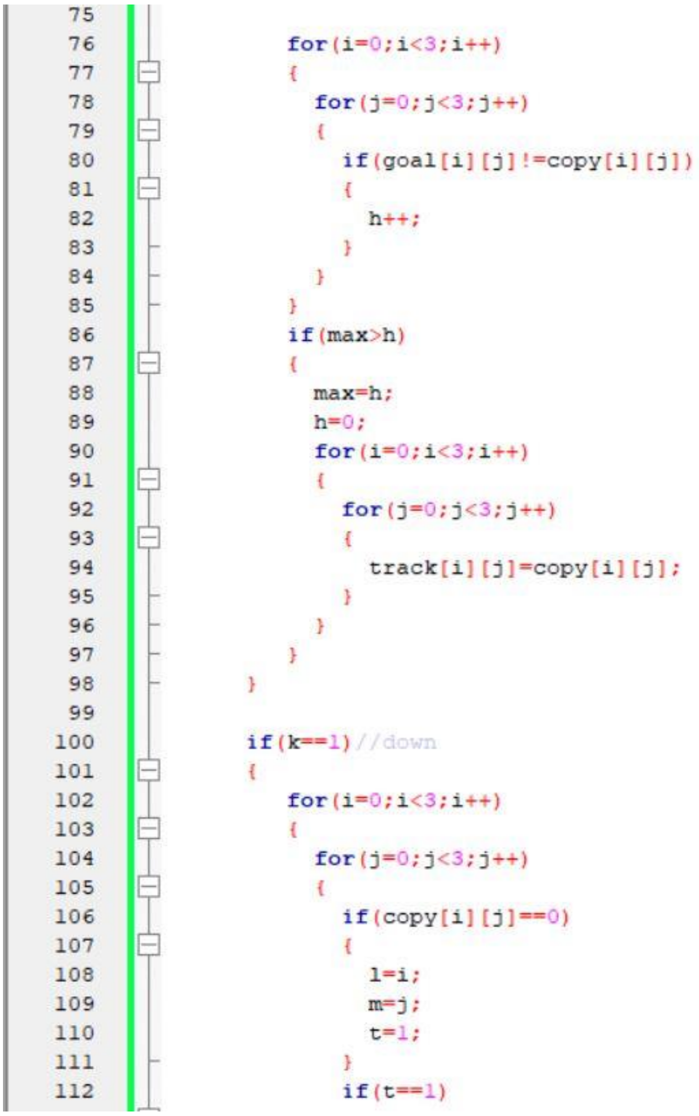
Program:



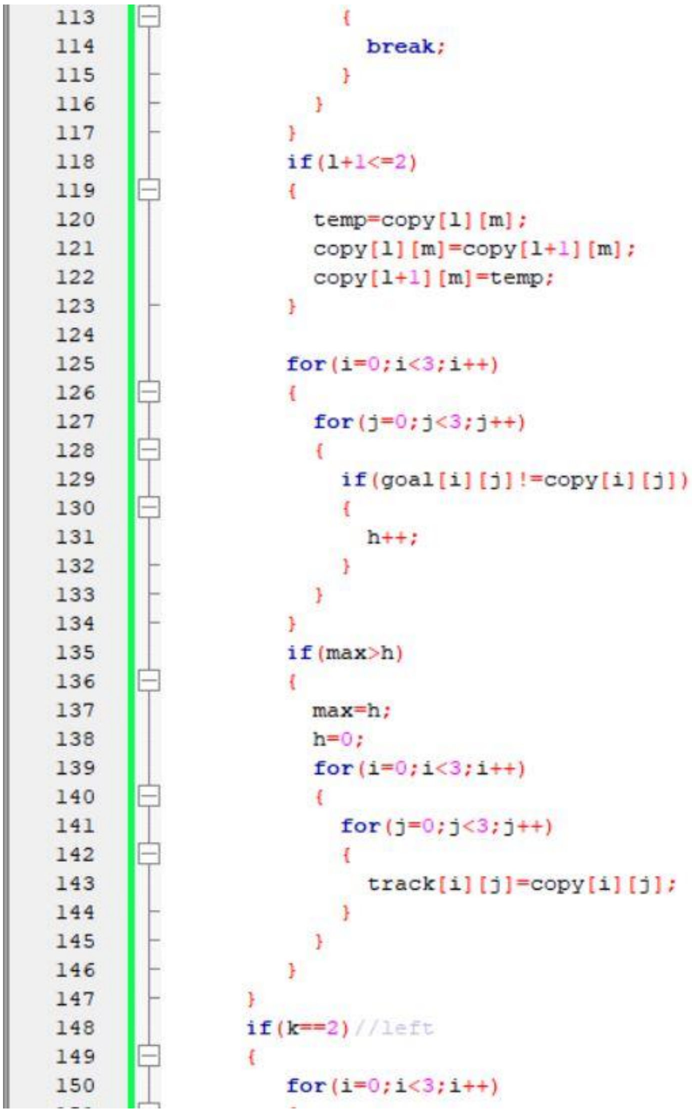
```c
#include <stdio.h>
int final[3][3];
int track[3][3];
void compare();
int start[3][3],goal[3][3];
int copy[3][3];
int i,j,l,m,k,flag=0,max=9999,h=0,count=0;
int main(void) {
  printf("Enter the start state\n");
  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
    {
      scanf("%d",&start[i][j]);
    }
  }
  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
    {
      copy[i][j]=start[i][j];
      final[i][j]=start[i][j];
    }
  }
  printf("Enter the goal state\n");
  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
    {
      scanf("%d",&goal[i][j]);
    }
  }
  printf("\nThe path is\n");
  int t=0;
  int temp;
  while(flag!=1)
  {
    for(k=0;k<4;k++)
```



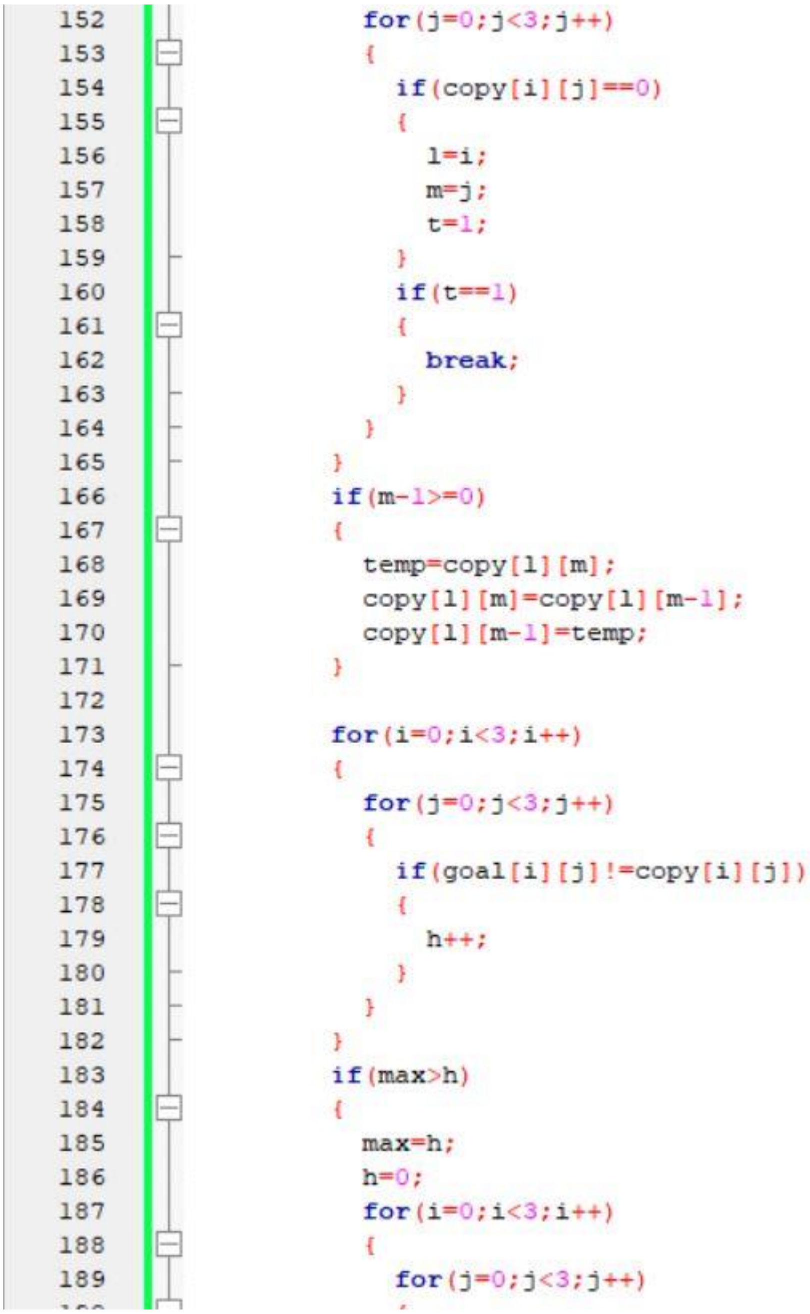
```
{
  t=0;
  h=0;
  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
    {
      copy[i][j]=final[i][j];
    }
  }


  if(k==0)//up
  {
     for(i=0;i<3;i++)
     {
       for(j=0;j<3;j++)
       {
         if(copy[i][j]==0)
         {
           l=i;
           m=j;
           t=1;
         }
         if(t==1)
         {
           break;
         }
       }
     }
     if(l-1>=0)
     {
       temp=copy[l][m];
       copy[l][m]=copy[l-1][m];
       copy[l-1][m]=temp;
     }
```



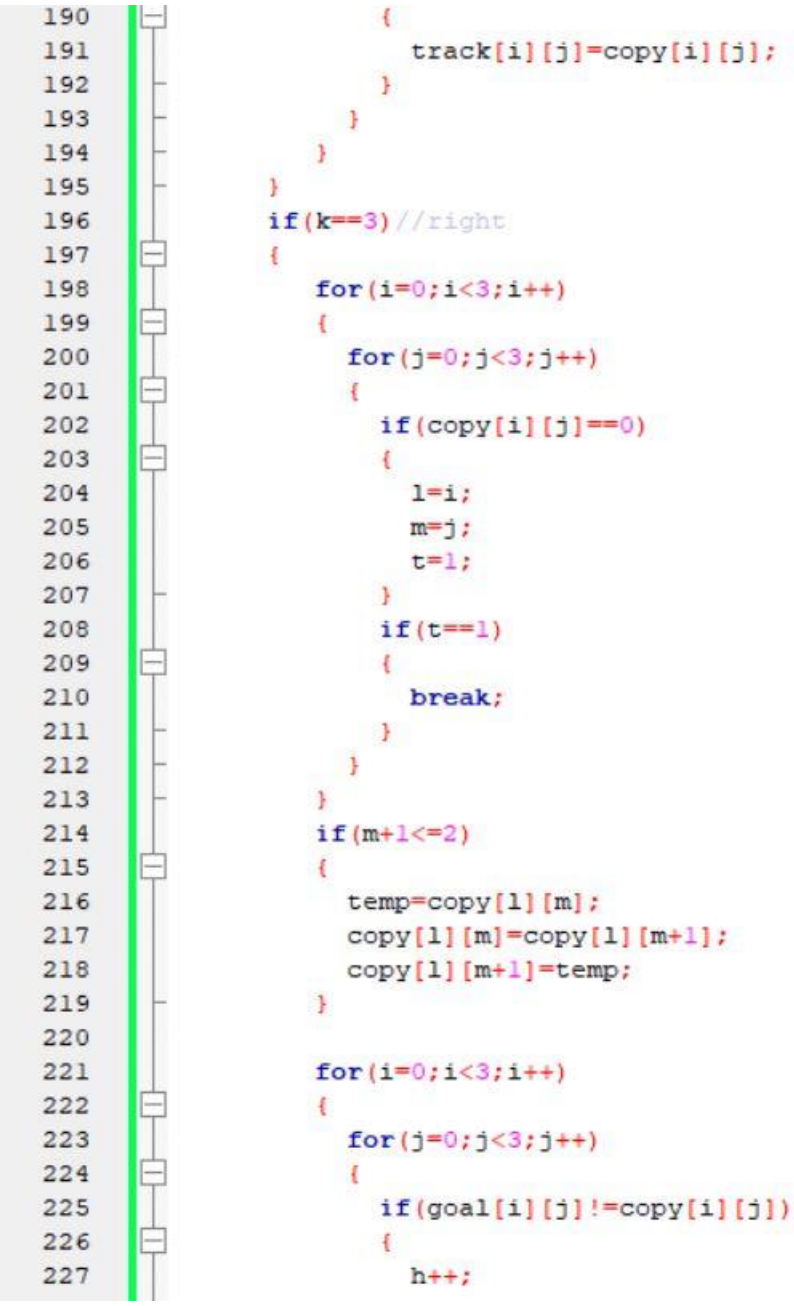
```

  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
    {
      if(goal[i][j]!=copy[i][j])
      {
        h++;
      }
    }
  }
  if(max>h)
  {
    max=h;
    h=0;
    for(i=0;i<3;i++)
    {
      for(j=0;j<3;j++)
      {
        track[i][j]=copy[i][j];
      }
    }
  }
}

if(k==1)//down
{
  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
    {
      if(copy[i][j]==0)
      {
        l=i;
        m=j;
        t=1;
      }
      if(t==1)
```



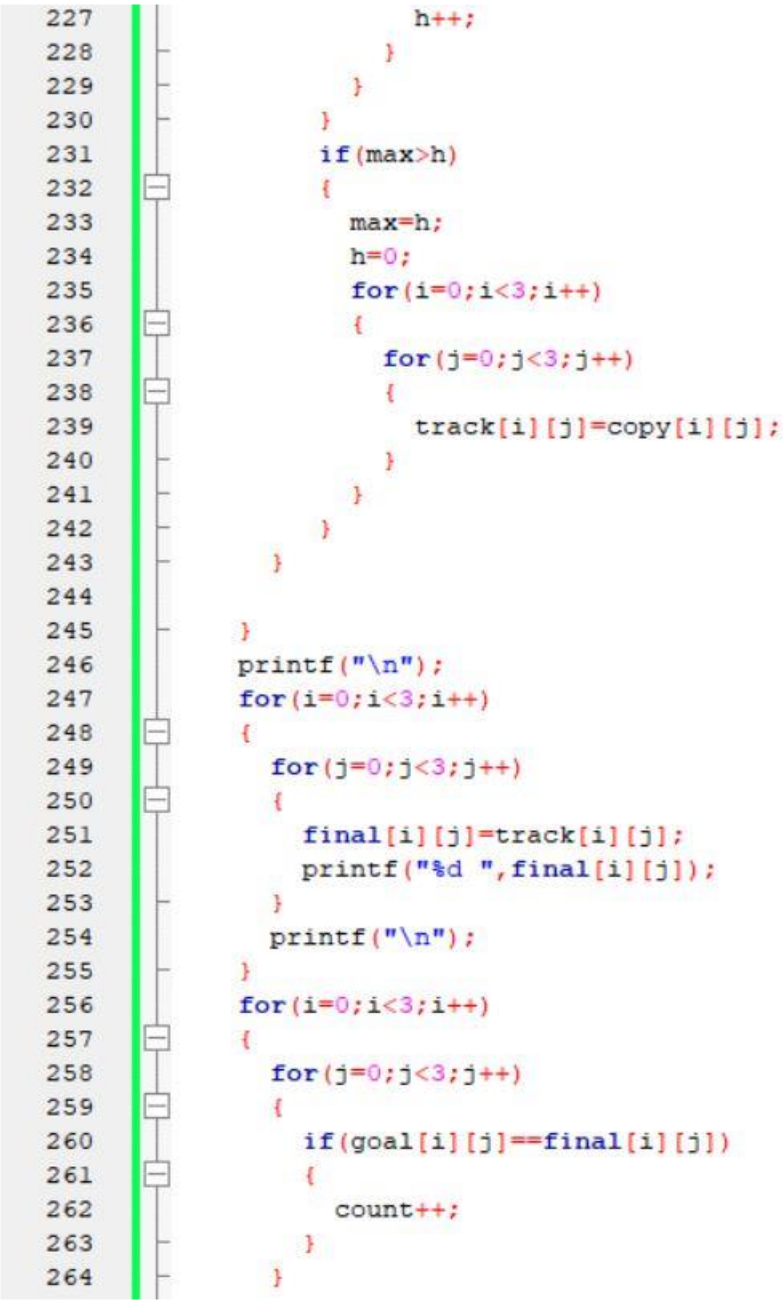
```
          {
            break;
          }
        }
      }
      if(l+1<=2)
      {
        temp=copy[l][m];
        copy[l][m]=copy[l+1][m];
        copy[l+1][m]=temp;
      }

      for(i=0;i<3;i++)
      {
        for(j=0;j<3;j++)
        {
          if(goal[i][j]!=copy[i][j])
          {
            h++;
          }
        }
      }
      if(max>h)
      {
        max=h;
        h=0;
        for(i=0;i<3;i++)
        {
          for(j=0;j<3;j++)
          {
            track[i][j]=copy[i][j];
          }
        }
      }
    }
    if(k==2)//left
    {
      for(i=0;i<3;i++)
```



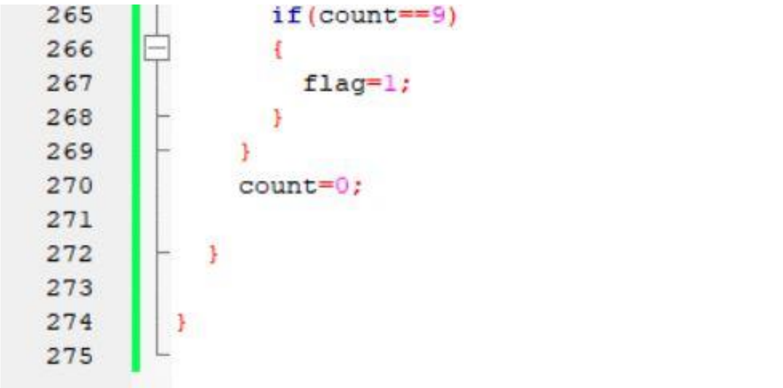
```
  for(j=0;j<3;j++)
  {
    if(copy[i][j]==0)
    {
      l=i;
      m=j;
      t=1;
    }
    if(t==1)
    {
      break;
    }
  }
}
if(m-1>=0)
{
  temp=copy[l][m];
  copy[l][m]=copy[l][m-1];
  copy[l][m-1]=temp;
}

for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  {
    if(goal[i][j]!=copy[i][j])
    {
      h++;
    }
  }
}
if(max>h)
{
  max=h;
  h=0;
  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
```



```
          {
            track[i][j]=copy[i][j];
          }
        }
      }
    }
    if(k==3)//right
    {
      for(i=0;i<3;i++)
      {
        for(j=0;j<3;j++)
        {
          if(copy[i][j]==0)
          {
            l=i;
            m=j;
            t=1;
          }
          if(t==1)
          {
            break;
          }
        }
      }
      if(m+1<=2)
      {
        temp=copy[l][m];
        copy[l][m]=copy[l][m+1];
        copy[l][m+1]=temp;
      }

      for(i=0;i<3;i++)
      {
        for(j=0;j<3;j++)
        {
          if(goal[i][j]!=copy[i][j])
          {
            h++;
```

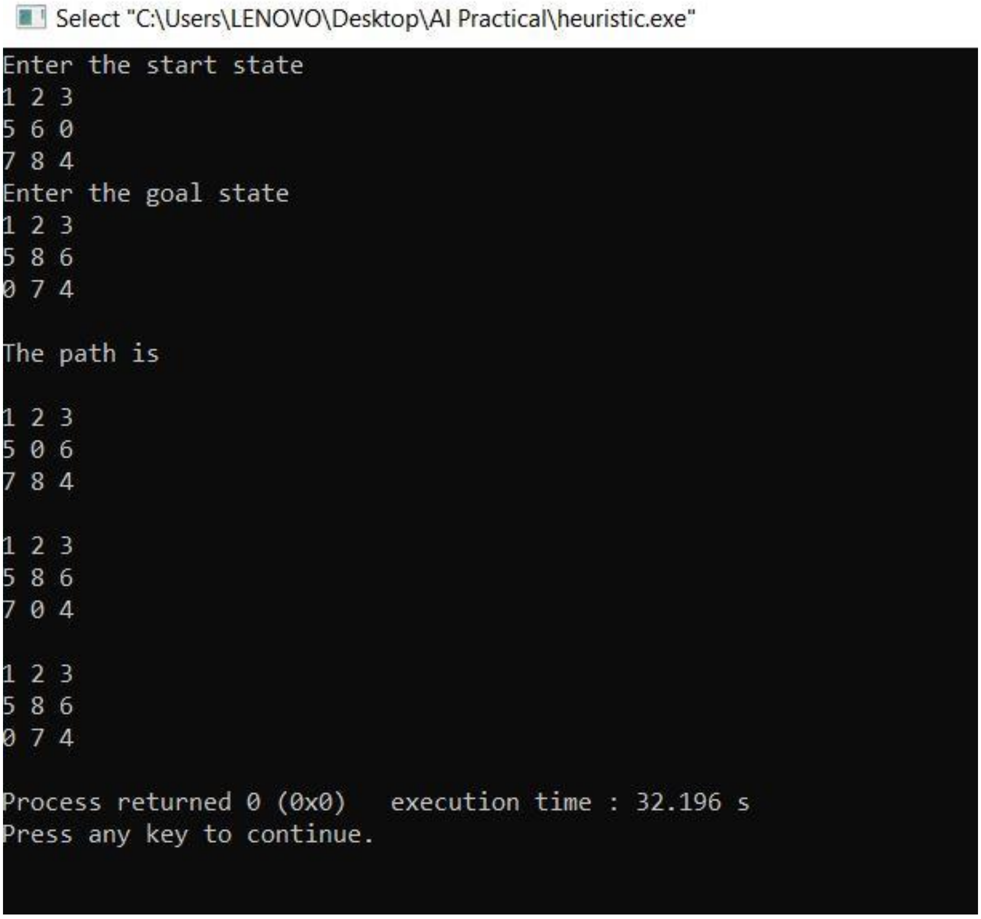


```
               h++;
             }
           }
         }
         if(max>h)
         {
           max=h;
           h=0;
           for(i=0;i<3;i++)
           {
             for(j=0;j<3;j++)
             {
               track[i][j]=copy[i][j];
             }
           }
         }
      }

    }
    printf("\n");
    for(i=0;i<3;i++)
    {
      for(j=0;j<3;j++)
      {
        final[i][j]=track[i][j];
        printf("%d ",final[i][j]);
      }
      printf("\n");
    }
    for(i=0;i<3;i++)
    {
      for(j=0;j<3;j++)
      {
        if(goal[i][j]==final[i][j])
        {
          count++;
        }
      }
```



```
      if(count==9)
      {
        flag=1;
      }
    }
    count=0;

  }

}

```

Output:



Select "C:\Users\LENOVO\Desktop\AI Practical\heuristic.exe"

```
Enter the start state
1 2 3
5 6 0
7 8 4
Enter the goal state
1 2 3
5 8 6
0 7 4

The path is

1 2 3
5 0 6
7 8 4

1 2 3
5 8 6
7 0 4

1 2 3
5 8 6
0 7 4

Process returned 0 (0x0)   execution time : 32.196 s
Press any key to continue.
```

Output: **:** Successfully Implemented Heuristic (Steepest Ascent) Search for 8 puzzle problem game problem.