EVALUATIVE ASSIGNMENT-7

Submitted by :

101803069 Samridhi Mangla

BE Third Year

Branch-COE-4

Submitted to:

MS.Anupam Garg

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

TIET,Patiala

In [1]:

```
# ques1) Solve the Tower of Hanoi problem using Breadth First Search

import sys
import copy
open=[] #empty list is created
closed=[] #empty list is created

def dequeue(): #dequeue function is used to remove head node from open list and add it to
closed
    global open
    global closed
    closed=closed+[open[0]]
    elem=open[0]
    del open[0]
    return elem

def enqueue(s): #enqueue function is used to add unexplored states in open list
    global open
    global closed

    if s not in open and s not in closed:
        open=open+[s]



def states(initial):
    for i in range(3):

        if len(initial[i])>0: #if length of list is greater than 0 then only remove disk
            elmen=initial[i][0] #store the remove disk value in variable elmen

            for j in range(3): #this loop is used to produce all possible combinations fo
r all rods
                if i!=j:
                    temp=copy.deepcopy(initial) #creating deepcopy of initial sate
                    if len(initial[j])==0:#if rod is empty then insert the elmen
                        temp[j].append(elmen)
                        del temp[i][0]
                        enqueue(temp)
                    elif len(initial[j])>0: #if rod is not empty then check if already e
xisting disk in that rod is greater than the elmen
                        if initial[j][0]>elmen:
                            temp[j].insert(0,elmen)
                            del temp[i][0]
                            enqueue(temp)



def towerofhannoi(initial,final):
    curr=copy.deepcopy(initial) #creating deepcopy of initial

    global closed
    while(True): #loop until we dont get the result
        if(curr==final):
            print(closed)
            return
        states(curr) #states function is called
        curr=dequeue() #dequeue function is called



def main():
    initial=[[1,2,3],[],[]] #creating initial state using list
    final=[[],[],[1,2,3]]    # goal state
    global open
    global closed
    closed=closed+[initial] #add initial state to closed list
    towerofhannoi(initial,final) #function calling
if __name__=="__main__":
    main()
```

```
[[[1, 2, 3], [], []], [[2, 3], [1], []], [[2, 3], [], [1]], [[3], [1], [2]], [[3], [2], [
1]], [[1, 3], [], [2]], [[3], [], [1, 2]], [[1, 3], [2], []], [[3], [1, 2], []], [[], [3]
, [1, 2]], [[], [1, 2], [3]], [[1], [3], [2]], [[], [1, 3], [2]], [[1], [2], [3]], [[], [
2], [1, 3]], [[1], [2, 3], []], [[2], [1, 3], []], [[1], [], [2, 3]], [[2], [], [1, 3]],
[[], [1, 2, 3], []], [[], [2, 3], [1]], [[1, 2], [3], []], [[2], [3], [1]], [[], [1], [2,
3]], [[], [], [1, 2, 3]]]
```

In [1]:

```
"""ques 2) Solve the 8-puzzle problem initial and final states given below and H(n) as
Manhattan distance of the initial as compared to the goal to be considered as the
heuristic function. Apply Hill climbing searching algorithm."""

#STEEP HILL CLIMBING




import sys
import copy
open=[]  #empty list is created
closed=[]  #empty list is created

def find_pos(s):  #function is used to calculate the position of empty tile

    for i in range(3):
        for j in range(3):
            if s[i][j] == 0:
                return([i,j])


def up(s,pos): #function is used to generate successor of initial state by moving empty t
ile upwards

    i = pos[0]
    j = pos[1]

    if i > 0:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i-1][j]
        temp[i-1][j] = 0
        return (temp)
    else:
        return (s)


def down(s,pos): #function is used to generate successor of initial state by moving empty
tile downwards

    i = pos[0]
    j = pos[1]

    if i < 2:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i+1][j]
        temp[i+1][j] = 0
        return (temp)
    else:
        return (s)


def right(s,pos): #function is used to generate successor of initial state by moving empt
y tile rightwards

    i = pos[0]
    j = pos[1]

    if j < 2:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i][j+1]
        temp[i][j+1] = 0
        return (temp)
    else:
        return (s)


def left(s,pos): #function is used to generate successor of initial state by moving empty tile leftwards

    i = pos[0]
    j = pos[1]

    if j > 0:
        temp = copy.deepcopy(s)
        temp[i][j] = temp[i][j-1]
        temp[i][j-1] = 0
        return (temp)
    else:
        return (s)




def heurestic(s,f): # heurestic function as Manhattan distance
    sum=0
    for i in range(3):
        for j in range(3):
            if(s[i][j]!=0):
                for r in range(3):
                    for t in range(3):
                        if s[i][j]==f[r][t]:
                            if i==r and j!=t:
                                if(j>t):
                                    sum=sum+(j-t)
                                else:
                                    sum=sum+(t-j)
                            elif j==t and i!=r:
                                if(i>r):
                                    sum=sum+(i-r)
                                else:
                                    sum=sum+(r-i)


    return sum



def states(s,g): #this function is used to produce all successors of s and then returning the one having lowest heurestic value
    temp=copy.deepcopy(s) #creating deepcopy of s
    sum1=heurestic(temp,g)
    q=[]                  #creating empty list
    pos = find_pos(temp)
    new = up(temp,pos)
    sum=heurestic(new,g)
    q=q+[[sum,new]]
    new=down(temp,pos)
    sum=heurestic(new,g)
    q=q+[[sum,new]]
    new=left(temp,pos)
    sum=heurestic(new,g)
    q=q+[[sum,new]]
    new=right(temp,pos)
    sum=heurestic(new,g)
    q=q+[[sum,new]]
    q.sort() #to get state having lowest heurestic value

    elmen=q[0][1]
    if(q[0][0]<sum1):
        return elmen
    else:
        return temp




def dequeue():#head node of open list is deleted and added in closed list
    global open
    global closed
    elmen=open[0]
    closed=closed+[open[0]]
    del open[0]
    return elmen


def puzzle(s,g):
    global open
    global closed
    curr=copy.deepcopy(s)
    if(s==g):
        return


    while(True):

        if(curr==g):
            print("FOUND!!")
            print(closed )
            return

        new=states(curr,g)
        if(new!=curr):
            open=open+[new] #adding the new state in open list



            curr=dequeue()
        else:
            print("not found")
            return


def main():
    s = [[1,2,3],[4,0,5],[7,8,6]] #initial state
    g = [[1,2,3],[4,5,6],[7,8,0]]  #final state
    global open
    global closed
    open=open
    closed=closed +[s]



    puzzle(s,g)

if __name__ == "__main__":
    main()
```

```
FOUND!!
[[[1, 2, 3], [4, 0, 5], [7, 8, 6]], [[1, 2, 3], [4, 5, 0], [7, 8, 6]], [[1, 2, 3], [4, 5,
6], [7, 8, 0]]]
```

In [0]: