***Artificial Intelligence***

***CSL 411***

***Lab Journal 8***

****

**Student Name:Rida Sajjad**

**Enrolment No.01-134152-063**

**Class and Section BSCS-7B**

**Department of Computer Science**

**BAHRIA UNIVERSITY, ISLAMABAD**

**Task1:**

import math

def minmax(depth,nodeIndex,isMax,scores,h):

    if(depth==h):

        return scores[nodeIndex]

    elif(isMax):

        return max(minmax(depth+1,(nodeIndex\*2),False,scores,h),minmax(depth+1,(nodeIndex\*2)+1,False,scores,h))

    elif(isMax!=True):

        return min(minmax(depth+1,(nodeIndex\*2),True,scores,h),minmax(depth+1,(nodeIndex\*2)+1,True,scores,h))

scores=[3,5,2,9,12,5,23,24]

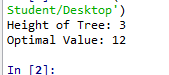
h=1+math.log(len(scores)/2)

print ('Height of Tree:', math.ceil(h))

result=minmax(0,0,True,scores,math.ceil(h))

print ("Optimal Value:", result)

**output:**



**Task2 &3:**

import math

def alphabeta(depth,nodeIndex,maximizingPlayer,values,alpha,beta):

if(depth==3):

return scores[nodeIndex]

elif(maximizingPlayer):

best=alpha

for i in range(0,2): val=alphabeta(depth+1,(nodeIndex\*2)+i,False,values,alpha,beta)

print(val)

alpha=max(val,alpha)

best=alpha

if (beta<=alpha):

break

return best

elif(maximizingPlayer!=True):

best=beta

for i in range(0,2):

val=alphabeta(depth+1,(nodeIndex\*2)+i,True,values,alpha,beta)

print(val)

beta=min(val,beta)

best=beta

if (beta<=alpha):

break

return best

def evaluateBoard(Board):

if(Board[0][0]=='X'):

if(Board[1][1]=='X'):

if(Board[2][2]=='X'):

return 10;

elif(Board[0][0]=='0'):

if(Board[1][1]=='0'):

if(Board[2][2]=='0'):

return 20;

elif(Board[0][0]=='0'):

if(Board[0][1]=='0'):

if(Board[0][2]=='0'):

return 20;

values=[3,5,6,9,1,2,0,-1]

print("Optimal value (AlphaBeta):",alphabeta(0,0,True,values,-1000,1000))

scores=[3,5,2,9,12,5,23,24]

h=1+math.log(len(scores)/2)

print ('Height of Tree:', math.ceil(h))

result=minmax(0,0,True,scores,math.ceil(h))

print ("Optimal Value:", result)

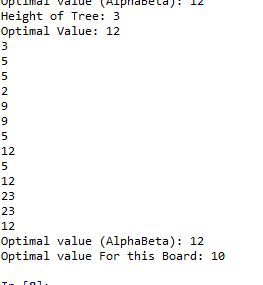
values=[3,5,6,9,1,2,0,-1]

print ('Optimal value (AlphaBeta):',alphabeta(0,0,True,values,-1000,1000))

Board=[['X','.','O'],['.','X','O'],['.','.','X']]

print ('Optimal value For this Board:',evaluateBoard(Board))

**output:**



return goalnode

elif depth >0:

if start==goalnode:

return goalnode

elif start !=goalnode:

for n in self.adj[start]:

found=self.dls(n,goalnode,depth-1)

if found ==goalnode:

return found

def IDDFS(self,start, goalnode, maxDepth):

for i in range(maxDepth):

if (self.dls(start, goalnode, i)):

print(start)

return True

return False

a=DWGraph()

a.add\_node('SR') a.add\_node('A') a.add\_node('B')

a.add\_node('C') a.add\_edge('SR','C',5) a.add\_edge('SR','B',5) a.add\_edge('C','B',5) a.add\_edge('SR','A',5) p=a.dfs('SR')

print(p)

print('cost is',cost)

pp=a.dls('SR','C',2)

print("goal node is",pp)

ppp=a.IDDFS('SR','A',4)

print("gpal node is",ppp)

**Result/Output:**

