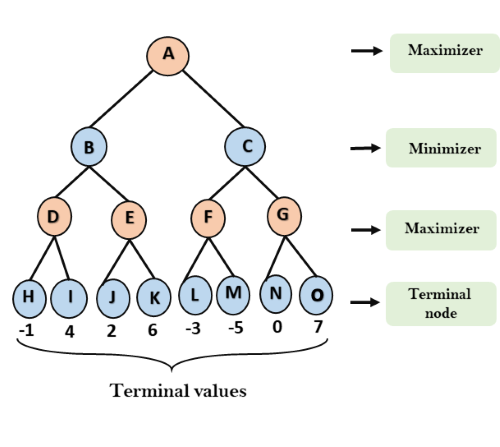
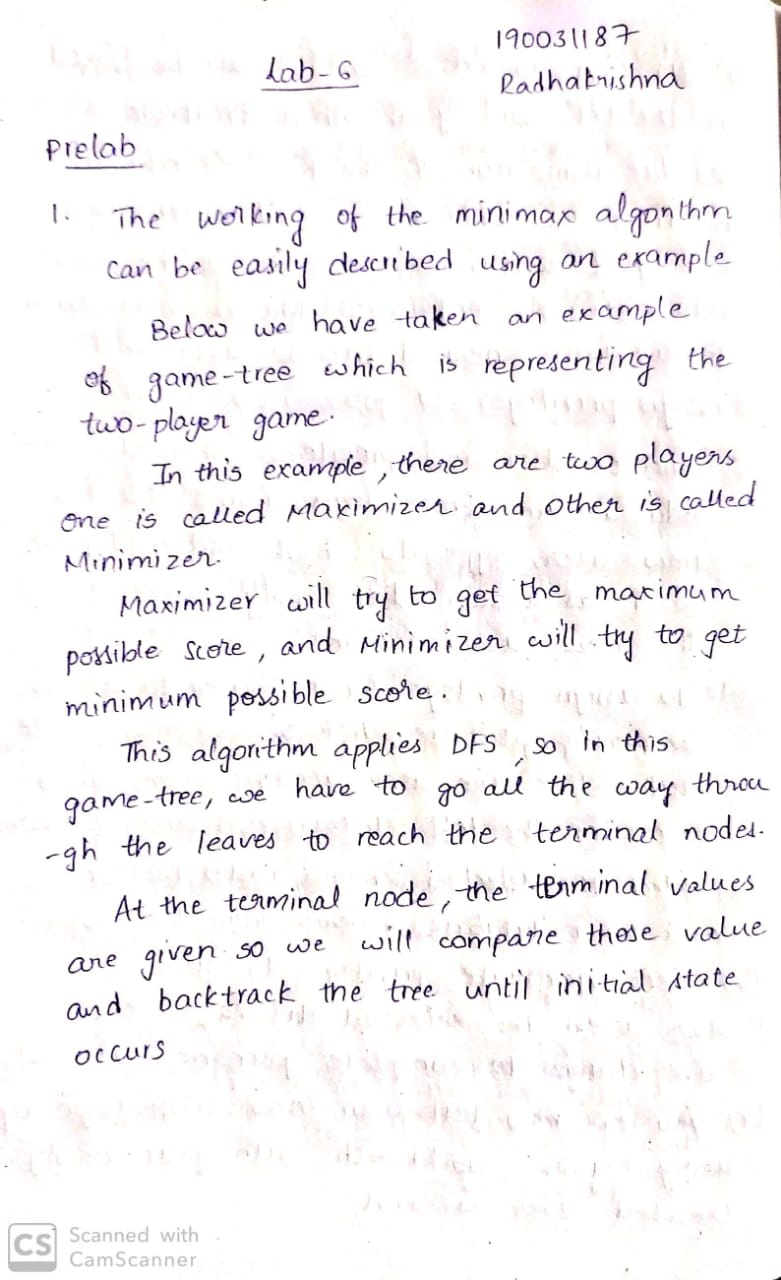
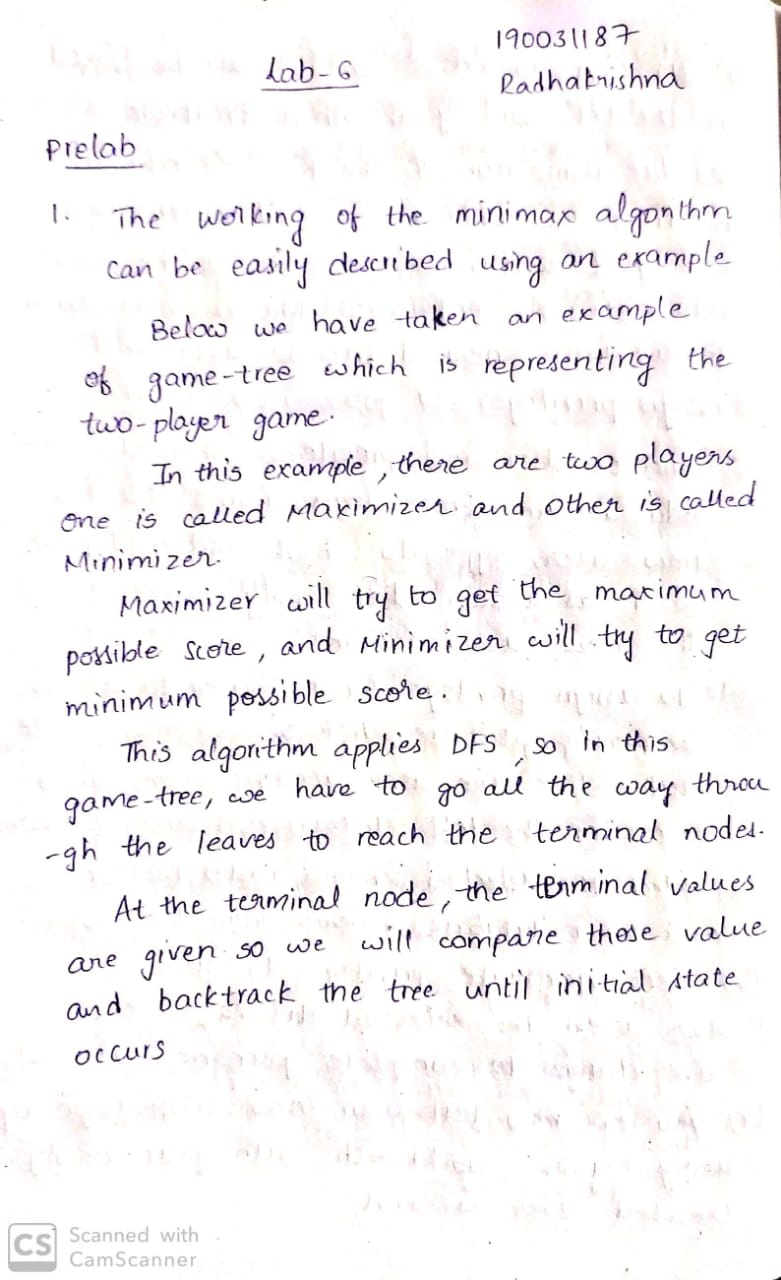
**Lab-6**

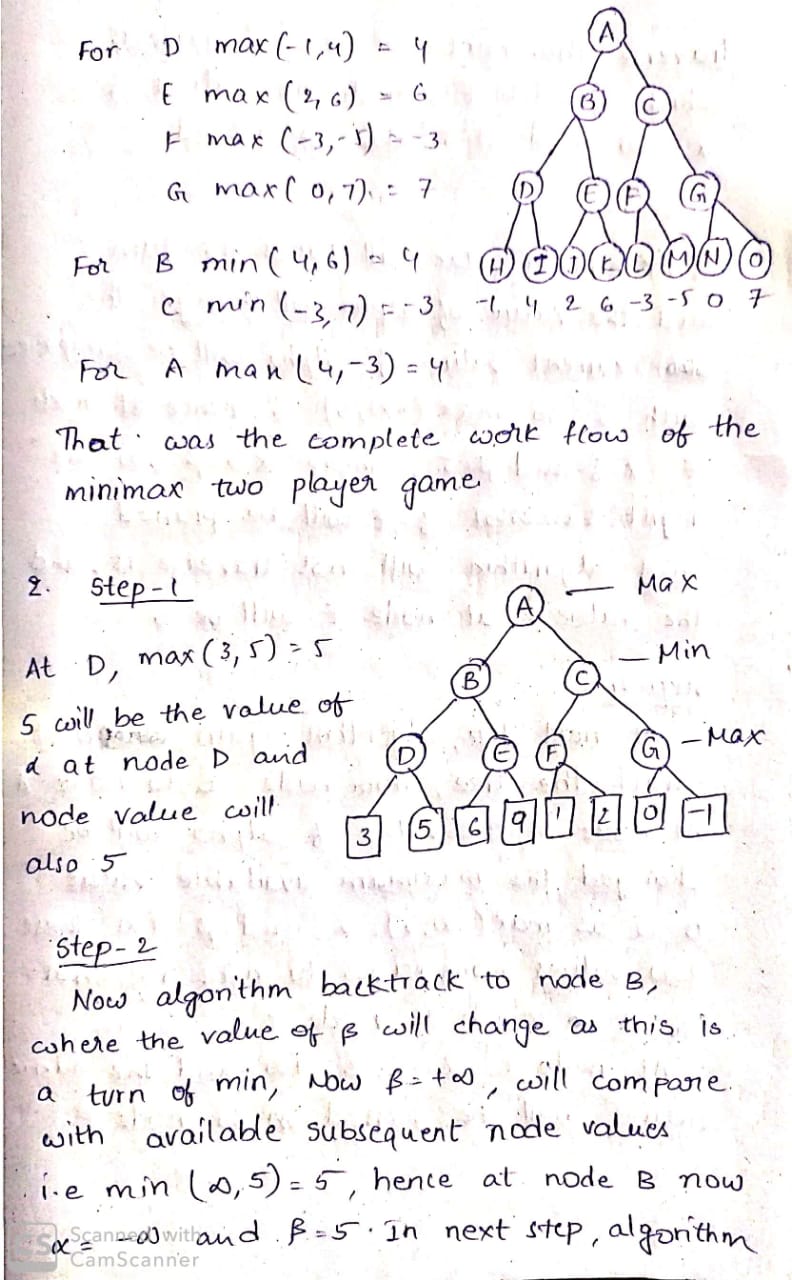
**PRELAB:**

1. Write the complete workflow of the minimax two player game.





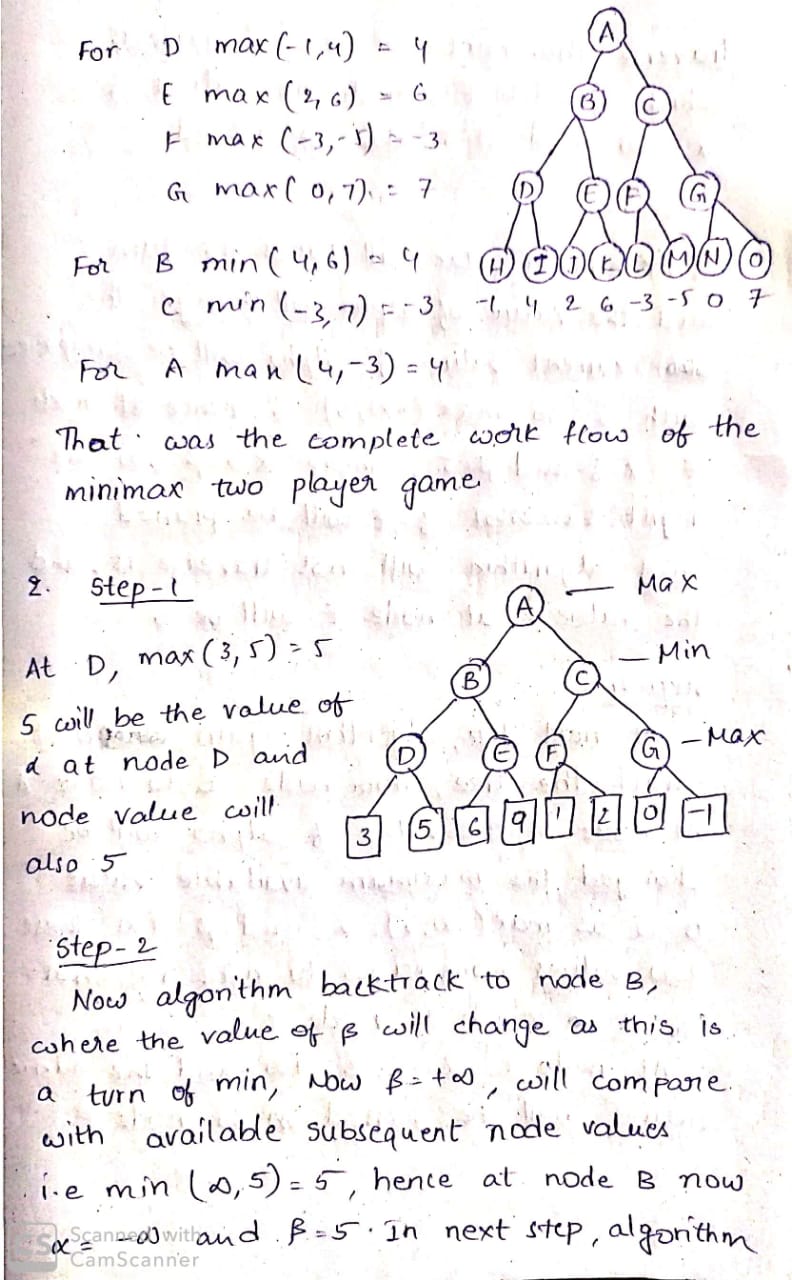


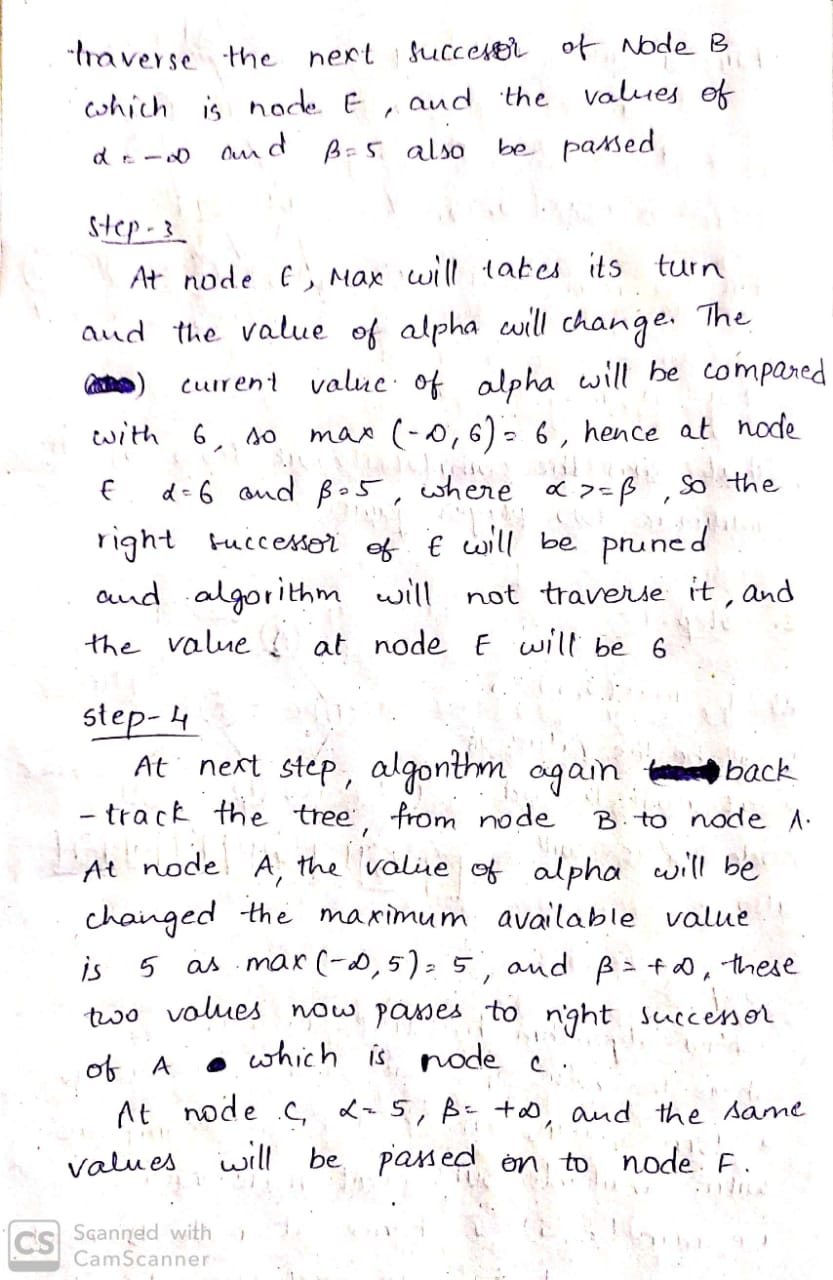


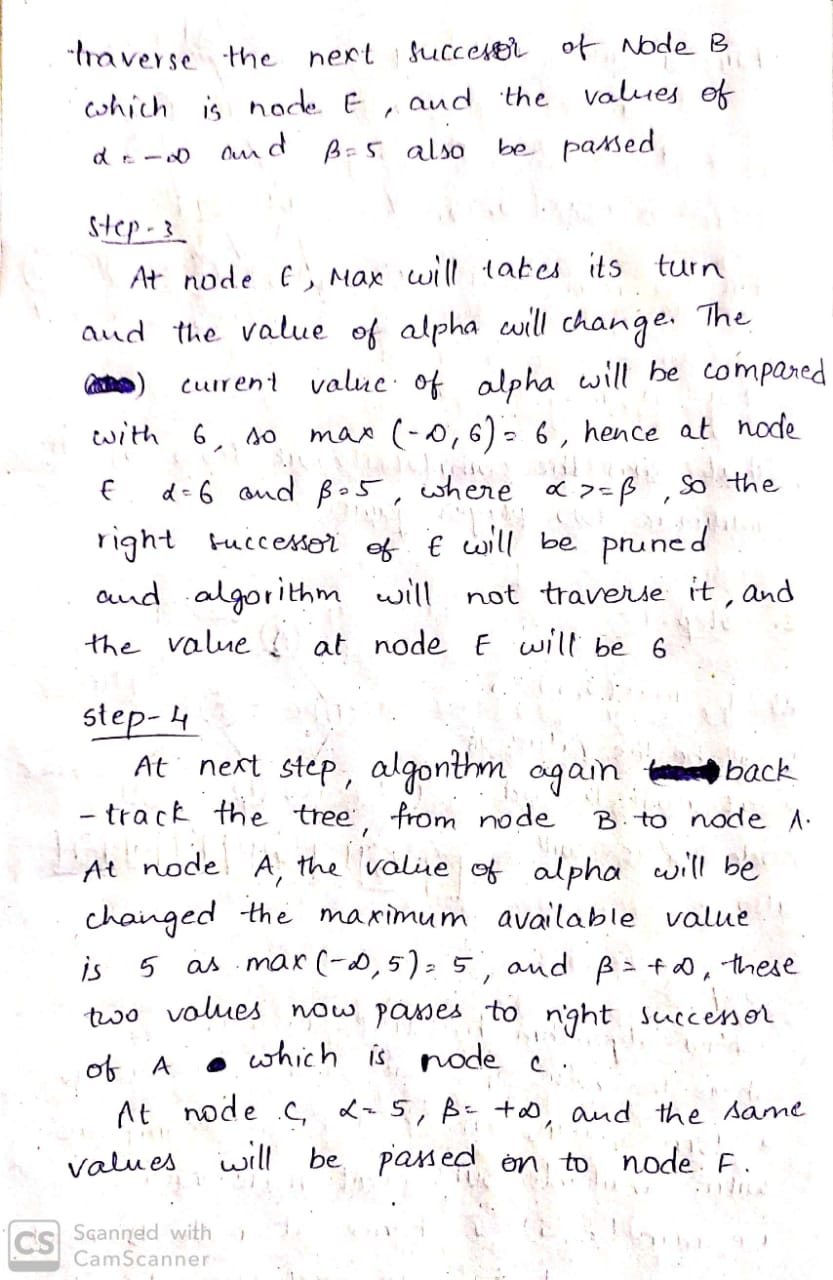
2. Write the complete workflow of the tree using alpha-beta Pruning.

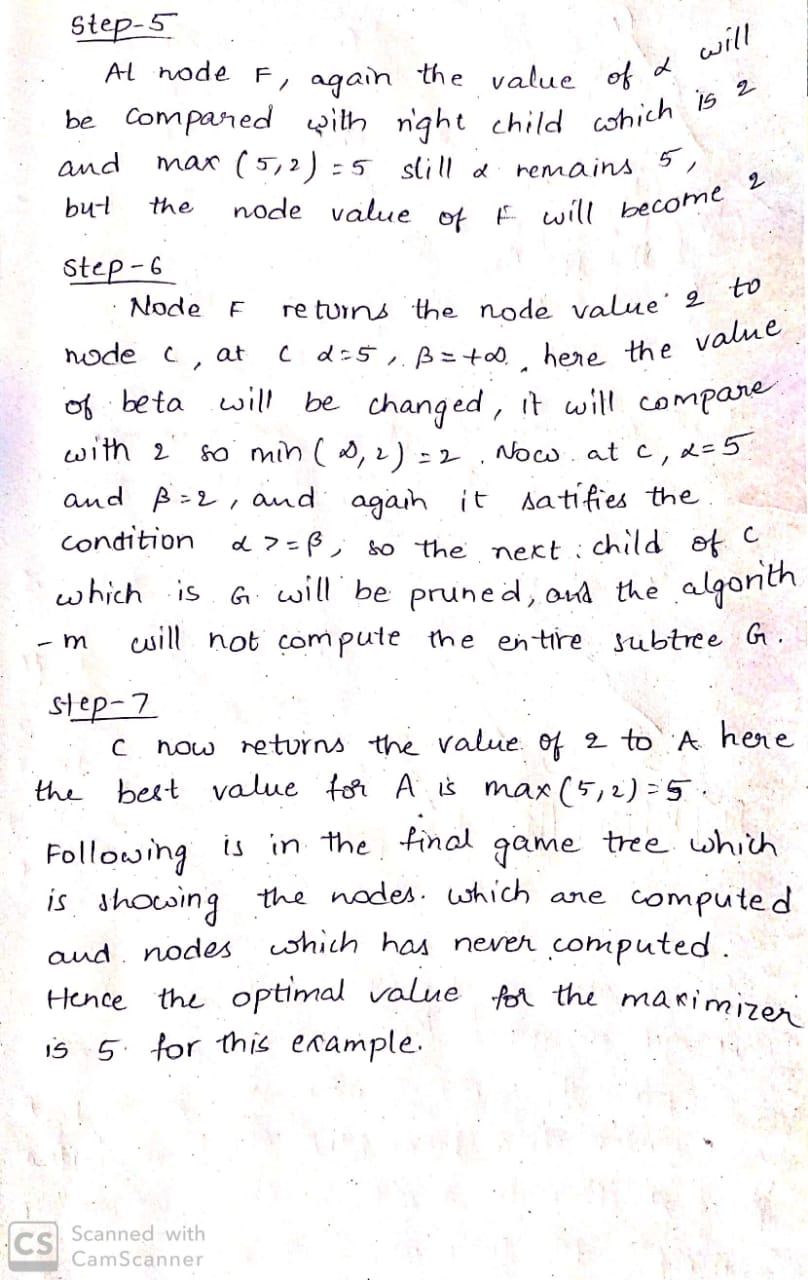
A picture containing drawing

Description automatically generated



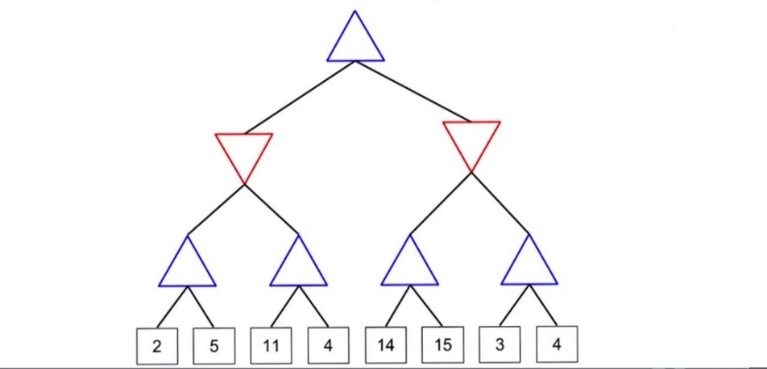






**INLAB:**

1. Write a python code to print the root node using minimax algorithm.



Output:

The Root Node value is: 5

**CODE**

import numpy as np

class Game\_Tree:

  succs=dict(A=dict(a1='B',a2='C'),

             B=dict(b1='D',b2='E'),

             C=dict(c1='F',c2='G'),

             D=dict(d1='D1',d2='D2'),

             E=dict(e1='E1',e2='E2'),

             F=dict(f1='F1',f2='F2'),

             G=dict(g1='G1',g2='G2'))

  utils=dict(D1=2,D2=5,E1=11,E2=4,F1=14,F2=15,G1=3,G2=4)

  def actions(self,state):

    return list(self.succs.get(state,{}).keys())

  def result(self,state,move):

    return self.succs[state][move]

  def utility(self,state,player):

    if player=='MAX':

      return self.utils[state]

    else:

      return -self.utils[state]

  def terminal\_test(self,state):

    return state not in ('A','B','C','D','E','F','G')

  def to\_move(self,state):

    return 'MIN' if state in 'BC' else 'MAX'

def minmax\_decision(state,game):

  player=game.to\_move(state)

  def max\_value(state):

    if game.terminal\_test(state):

      return game.utility(state,player)

    v=-np.inf

    for a in game.actions(state):

      v=max(v,min\_value(game.result(state,a)))

    return v

  def min\_value(state):

    if game.terminal\_test(state):

      return game.utility(state,player)

    v=np.inf

    for a in game.actions(state):

      v=min(v,max\_value(game.result(state,a)))

    return v

  v=-np.inf

  for a in game.actions(state):

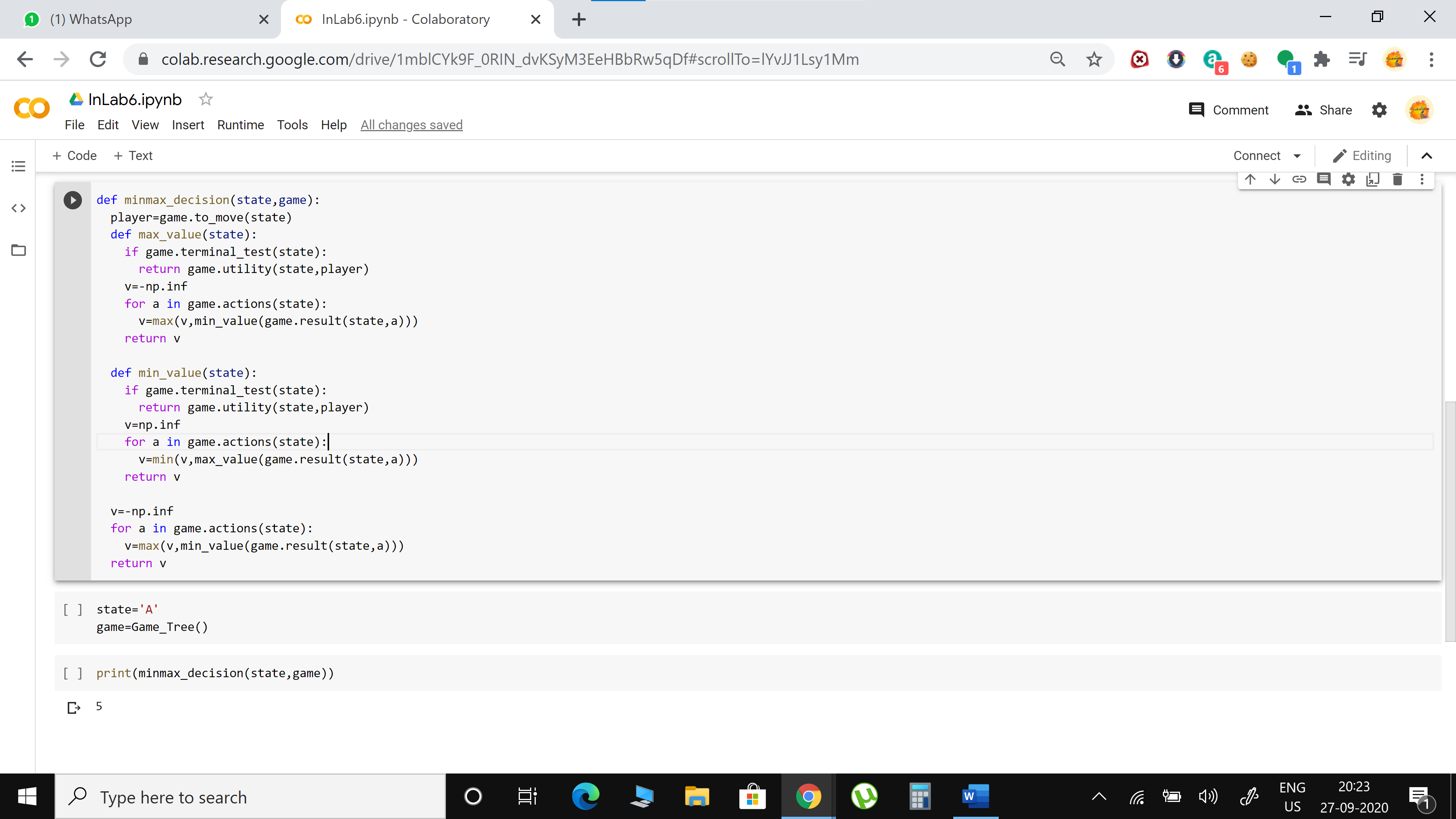
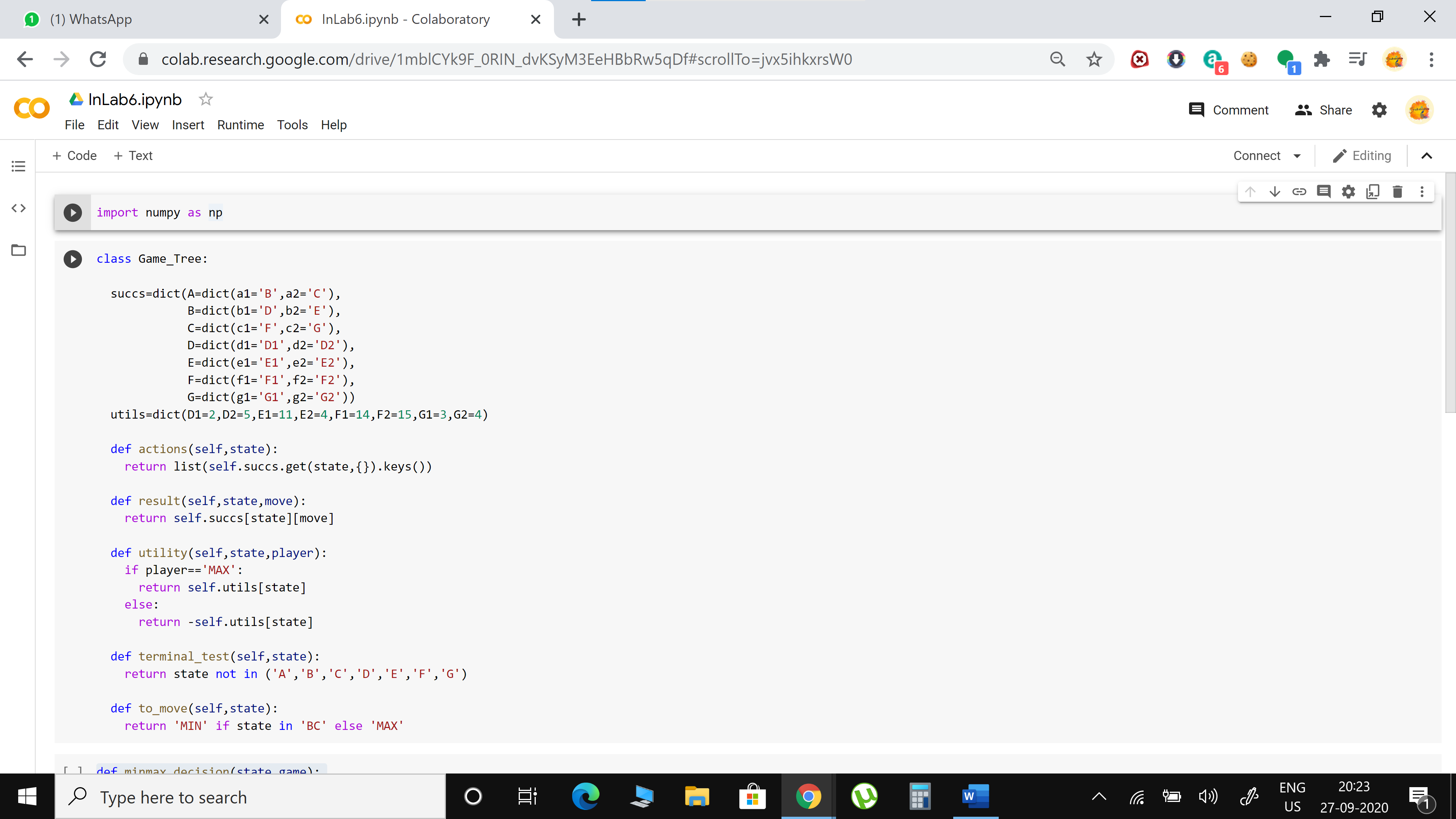
    v=max(v,min\_value(game.result(state,a)))

  return v

state='A'

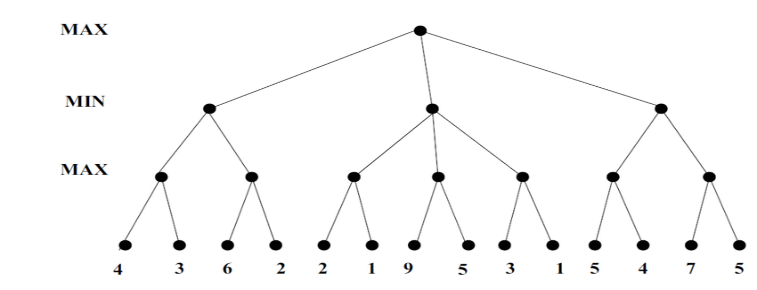
game=Game\_Tree()

print(minmax\_decision(state,game))



**POSTLAB:**

1. Write a python code to implement the given tree and print the alpha - beta values in the root node and also print the result and time pruned by using Alpha beta pruning.



Output:

(Alpha, beta): 5 15

Result: 5

Times pruned: 4

**CODE**

import numpy as np

class Game\_Tree:

  succs=dict(A=dict(a1='B',a2='C',a3='D'),

             B=dict(b1='E',b2='F'),

             C=dict(c1='G',c2='H',c3='I'),

             D=dict(d1='J',d2='K'),

             E=dict(e1='E1',e2='E2'),

             F=dict(f1='F1',f2='F2'),

             G=dict(g1='G1',g2='G2'),

             H=dict(h1='H1',h2='H2'),

             I=dict(i1='I1',i2='I2'),

             J=dict(j1='J1',j2='J2'),

             K=dict(k1='K1',k2='K2'))

  utils=dict(E1=4,E2=3,F1=6,F2=4,G1=2,G2=1,H1=9,H2=5,I1=3,I2=1,J1=5,J2=4,K1=7,K2=5)

  initial='A'

  def actions(self,state):

    return list(self.succs.get(state,{}).keys())

  def result(self,state,move):

    return self.succs[state][move]

  def utility(self,state,player):

    if player=='MAX':

      return self.utils[state]

    else:

      return -self.utils[state]

  def terminal\_test(self,state):

    return state not in('A','B','C','D','E','F','G','H','I','J','K')

  def to\_move(self,state):

    return 'MIN' if state in 'BCD' else 'MAX'

p=0

alpha=-np.inf

beta=np.inf

def alpha\_beta\_search(state,game):

  player=game.to\_move(state)

  #Functions used by alpha\_beta

  def max\_value(state,alpha,beta):

      global p

      if game.terminal\_test(state):

        return game.utility(state,player)

      v=-np.inf

      for a in game.actions(state):

        v=max(v,min\_value(game.result(state,a),alpha,beta))

        if v>=beta:

          p=p+1

          return v

        alpha=max(alpha,v)

      return v

  def min\_value(state,alpha,beta):

      global p

      if game.terminal\_test(state):

        return game.utility(state,player)

      v=np.inf

      for a in game.actions(state):

        v=min(v,max\_value(game.result(state,a),alpha,beta))

        if v<=alpha:

          p=p+1

          return v

        beta=min(beta,v)

      return (v)

  #Body of alpha\_beta\_search

  global alpha

  global beta

  best\_action=None

  for a in game.actions(state):

    v=min\_value(game.result(state,a),alpha,beta)

    if v > alpha:

      alpha=v

      best\_action=a

    else:

      beta=v

  return (v)

state='A'

game=Game\_Tree()

print("Result- ",alpha\_beta\_search(state,game))

print("Times pruned- ",p)

print("Alpha- ",alpha)

print("Beta- ",beta)

