1.BFS

graph={

'5':['3','7'],

'3':['2','4'],

'7':['8'],

'2':[],

'4':['8'],

'8':[]

}

visited=[]

queue=[]

def BFS(visited, graph, node):

visited.append(node)

queue.append(node)

while queue:

m=queue.pop(0)

print(m , end=" ")

for neihghour in graph[m]:

if neihghour not in visited:

visited.append(neihghour)

queue.append(neihghour)

print("following is the breadth-first serch")

BFS(visited, graph,'5')

2.DFS

graph={

'5':['3','7'],

'3':['2','4'],

'7':['8'],

'2':[],

'4':['8'],

'8':[]

}

visited=set()

def DFS(visited, graph, node):

if node not in visited:

print(node)

visited.add(node)

for neighbour in graph[node]:

DFS(visited, graph, neighbour)

DFS(visited,graph,'5')

print("following is the depth-first serch")

DFS(visited, graph,'5')

3.Nqueens

def is\_safe(board,row,col,n):

for c in range(col,-1,-1):

if board[row][c]== 1:

return False

i=row

j=col

while i>=0 and j>=0:

if board[i][j]==1:

return False

i-=1

j-=1

i=row

j=col

while i<n and col>=0:

if board [i][j]==1:

return False

i+=1

j-=1

return True

def nQueens(board,col,n):

if col>=n:

return True

for i in range (n):

if is\_safe(board,i,col,n):

board[i][col]=1

if nQueens(board,col+1,n):

return True

board[i][col]=0

return False

n=int(input('enter size of board:'))

board=[[0 for j in range (n)] for i in range (n)]

if nQueens(board,0,n)==True:

for i in range(n):

for j in range (n):

print(board[i][j], end='')

print()

else:

print("not possible")

4.Backwards

database = ["Croaks", "Eat Flies", "Chirps", "Sings"]

knowbase = ["Frog", "Canary"]

color = ["Green", "Yellow"]

def main():

print("\*----- Backward Chaining -----\*")

print("\nX is \n1. Frog \n2. Canary\nSelect One: ", end='')

x = int(input())

print("")

if x == 1:

print("Chance of eating flies ", end='')

elif x == 2:

print("Chance of chirping ", end='')

else:

print("\n----- Invalid Option Selected")

return

print(f"\nX is {knowbase[x-1]}")

print("\n1. Green \n2. Yellow")

k = int(input())

if (k == 1 and x == 1) or (k == 2 and x == 2):

print(f"Yes, it is in {color[k-1]} color and will {database[x-1]}.")

else:

print("\n--- Invalid Knowledge Database")

if \_\_name\_\_ == "\_\_main\_\_":

main()

5.Forward

database = ["Croaks", "Eat Flies", "Chirps", "Sings"]

knowbase = ["Frog", "Canary", "Green", "Yellow"]

def main():

print("\*----- Forward Chaining -----\*")

print("\nX is \n1. Croaks \n2. Eat Flies \n3. Chirps \n4. Sings \nSelect One: ", end='')

x = int(input())

print("")

if x in [1, 2]:

print("Chance of Frog ", end='')

elif x in [3, 4]:

print("Chance of Canary ", end='')

else:

print("\n------ Invalid option selected -----", end='')

return

print(f"\nX is {database[x-1]}")

print("\nColor is 1. Green 2. Yellow \nSelect option: ", end='')

k = int(input())

if (k == 1 and x in [1, 2]) or (k == 2 and x in [3, 4]):

print(f"Yes, it is a {knowbase[x//2]} and color is {knowbase[k+1]}")

else:

print("\n--- Invalid Knowledge Database", end='')

if \_\_name\_\_ == "\_\_main\_\_":

main()

6.Alpha beta

MAX, MIN = 1000, -1000

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

if depth == 3:

return values[nodeIndex]

if maximizingPlayer:

best = MIN

for i in range(0, 2):

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

best = max(best, val)

alpha = max(alpha, best)

if beta <= alpha:

break

return best

else:

best = MAX

for i in range(0, 2):

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

best = min(best, val)

beta = min(beta, best)

if beta <= alpha:

break

return best

if \_\_name\_\_ == "\_\_main\_\_":

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

print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))

graph={

    '5':['3','7'],

    '3':['2','4'],

    '7':['8'],

    '2':[],

    '4':['8'],

    '8':[]

}

visited=[]

queue=[]

def  BFS(visited, graph, node):

 visited.append(node)

 queue.append(node)

 while queue:

  m=queue.pop(0)

  print(m , end=" ")

  for neihghour in graph[m]:

   if neihghour not in visited:

    visited.append(neihghour)

    queue.append(neihghour)

print("following is the breadth-first serch")

BFS(visited, graph,'5')

graph={

    '5':['3','7'],

    '3':['2','4'],

    '7':['8'],

    '2':[],

    '4':['8'],

    '8':[]

}

visited=set()

def  DFS(visited, graph, node):

    if node not in visited:

        print(node)

        visited.add(node)

        for neighbour in graph[node]:

            DFS(visited, graph, neighbour)

            DFS(visited,graph,'5')

print("following is the depth-first serch")

DFS(visited, graph,'5')

def is\_safe(board,row,col,n):

    for c in range(col,-1,-1):

        if board[row][c]== 1:

            return False

    i=row

    j=col

    while i>=0 and j>=0:

        if board[i][j]==1:

            return False

        i-=1

        j-=1

    i=row

    j=col

    while i<n and col>=0:

        if board [i][j]==1:

            return False

        i+=1

        j-=1

    return True

def nQueens(board,col,n):

    if col>=n:

        return True

    for i in range (n):

        if is\_safe(board,i,col,n):

            board[i][col]=1

            if nQueens(board,col+1,n):

                return True

            board[i][col]=0

    return False

n=int(input('enter size of board:'))

board=[[0 for j in range (n)] for i in range (n)]

if nQueens(board,0,n)==True:

    for i in range(n):

        for j in range (n):

            print(board[i][j], end='')

        print()

else:

    print("not possible")

database = ["Croaks", "Eat Flies", "Chirps", "Sings"]

knowbase = ["Frog", "Canary"]

color = ["Green", "Yellow"]

def main():

    print("\*----- Backward Chaining -----\*")

    print("\nX is \n1. Frog \n2. Canary\nSelect One: ", end='')

    x = int(input())

    print("")

    if x == 1:

        print("Chance of eating flies ", end='')

    elif x == 2:

        print("Chance of chirping ", end='')

    else:

        print("\n----- Invalid Option Selected")

        return

    print(f"\nX is {knowbase[x-1]}")

    print("\n1. Green \n2. Yellow")

    k = int(input())

    if (k == 1 and x == 1) or (k == 2 and x == 2):

        print(f"Yes, it is in {color[k-1]} color and will {database[x-1]}.")

    else:

        print("\n--- Invalid Knowledge Database")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

database = ["Croaks", "Eat Flies", "Chirps", "Sings"]

knowbase = ["Frog", "Canary", "Green", "Yellow"]

def main():

    print("\*----- Forward Chaining -----\*")

    print("\nX is \n1. Croaks \n2. Eat Flies \n3. Chirps \n4. Sings \nSelect One: ", end='')

    x = int(input())

    print("")

    if x in [1, 2]:

        print("Chance of Frog ", end='')

    elif x in [3, 4]:

        print("Chance of Canary ", end='')

    else:

        print("\n------ Invalid option selected -----", end='')

        return

    print(f"\nX is {database[x-1]}")

    print("\nColor is 1. Green 2. Yellow \nSelect option: ", end='')

    k = int(input())

    if (k == 1 and x in [1, 2]) or (k == 2 and x in [3, 4]):

        print(f"Yes, it is a {knowbase[x//2]} and color is {knowbase[k+1]}")

    else:

        print("\n--- Invalid Knowledge Database", end='')

if \_\_name\_\_ == "\_\_main\_\_":

    main()

MAX, MIN = 1000, -1000

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

  if depth == 3:

     return values[nodeIndex]

  if maximizingPlayer:

     best = MIN

     for i in range(0, 2):

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

        best = max(best, val)

        alpha = max(alpha, best)

        if beta <= alpha:

           break

     return best

  else:

     best = MAX

     for i in range(0, 2):

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

         best = min(best, val)

         beta = min(beta, best)

         if beta <= alpha:

            break

     return best

if \_\_name\_\_ == "\_\_main\_\_":

 values = [3, 5, 6, 9, 1, 2, 0, -1]

 print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))