Artificial Intelligence Lab Report

Submitted by

SHASHANKA G (1BM20CS147)

Batch: C3

Course: Artificial Intelligence
Course Code: 20CS5PCAIP
Sem & Section: 5th & C

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B. M. S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)
BENGALURU-560019

2022-2023

Table of Contents

Date: 9/11/22

Program 1: Implement Tic –Tac –Toe Game.

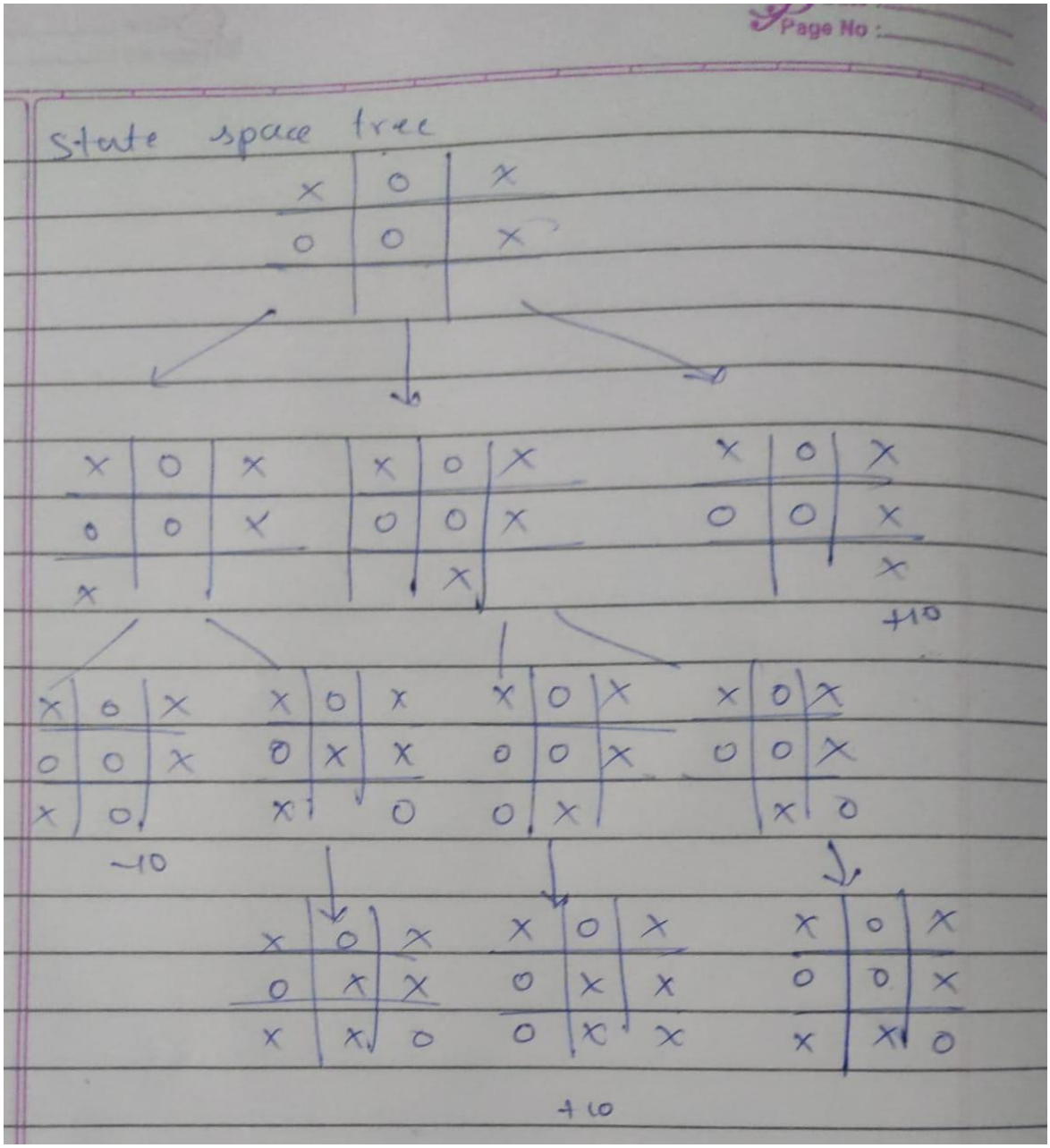
Algorithm:

Algorithm

```
int minimax (state, level) {
    if (state is a solution)   // base case
        return states value

    if (level is max) {
        while (state has more children)
            Minimax (child, level+1)
            if value returned is > best so
                far, store it & return it
    } else {   // level is a min value
        while (state has more children)
            Minimax (child, level+1)
            if value returned is < best so
                far, store it
    }
    return minimum returned value

    }
}
```

Space state Tree

State space tree



Code

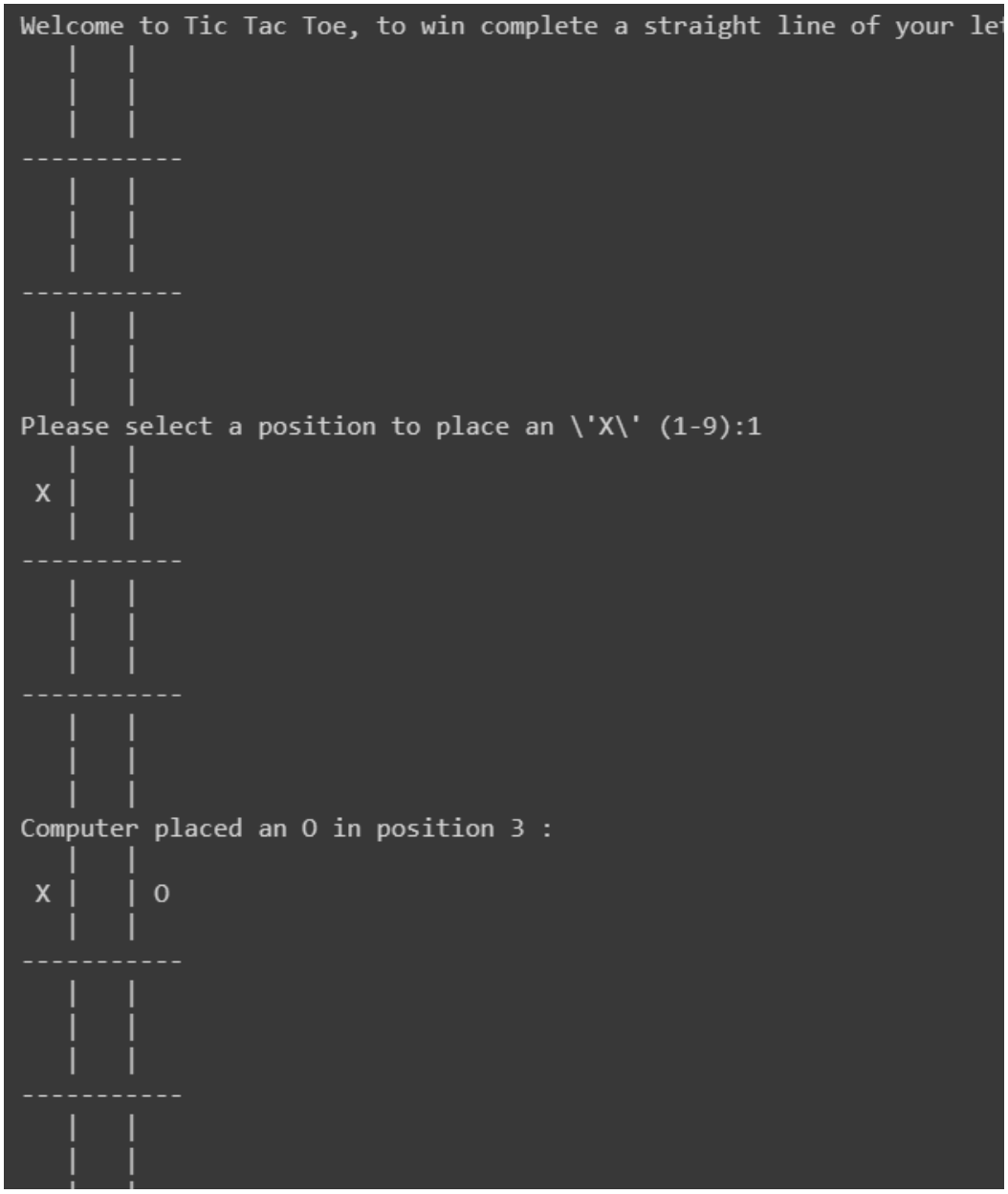
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

#Tic Tac Toe Game Python

board = [' ' for x in range(10)]


def insertBoard(letter, pos):
    global board
    board[pos] = letter


def spaceIsFree(pos):
    return board[pos] == ' '

def isWinner(bo, le):
    # Given a board and a player’s letter, this function returns True if
that player has won.
    # We use bo instead of board and le instead of letter so we don’t
have to type as much.
    return ((bo[7] == le and bo[8] == le and bo[9] == le) or # across the
top
    (bo[4] == le and bo[5] == le and bo[6] == le) or # across the middle
    (bo[1] == le and bo[2] == le and bo[3] == le) or # across the bottom
    (bo[7] == le and bo[4] == le and bo[1] == le) or # down the left side
    (bo[8] == le and bo[5] == le and bo[2] == le) or # down the middle
    (bo[9] == le and bo[6] == le and bo[3] == le) or # down the right
side
    (bo[7] == le and bo[5] == le and bo[3] == le) or # diagonal
    (bo[9] == le and bo[5] == le and bo[1] == le)) # diagonal


def playerMove():
    run = True
    while run:
        move = input('Please select a position to place an \\\'X\\\' (1-
9):')
        try:
            move  = int(move)
            if move > 0 and move < 10:
                if spaceIsFree(move):
```

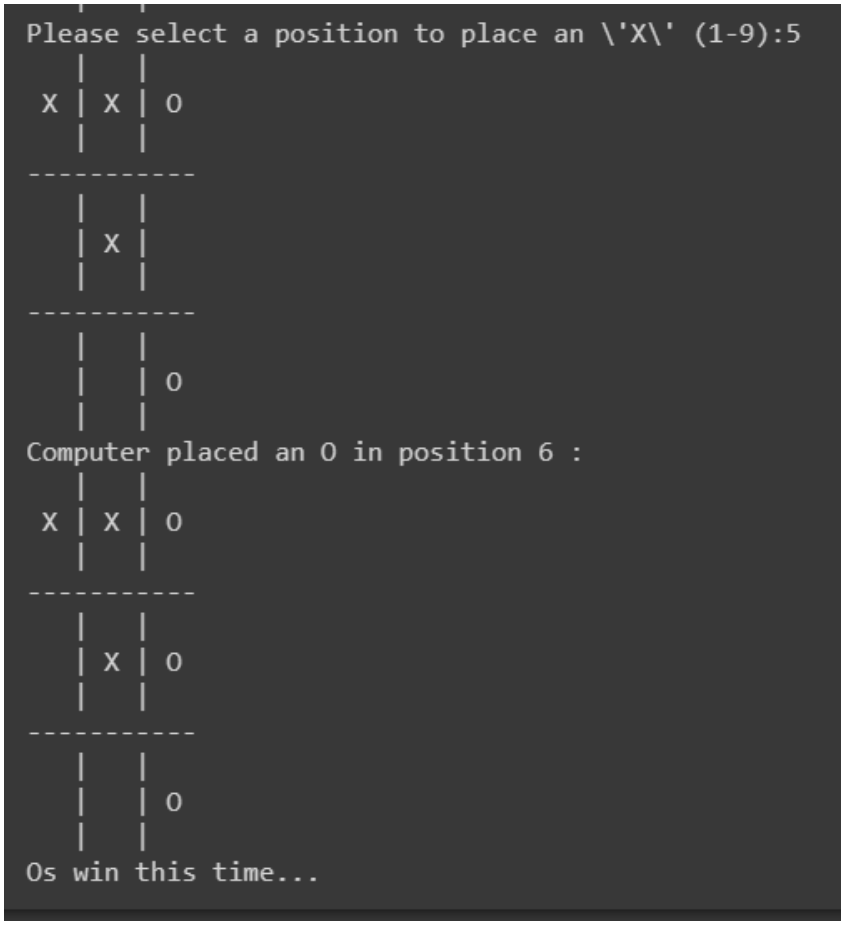
```
                    run = False
                    insertBoard('X', move)
                else:
                    print('This postion is already occupied!')
            else:
                print('Please type a number within the range!')
        except:
            print('Please type a number!')


def selectRandom(li):
    import random
    ln = len(li)
    r = random.randrange(0, ln)
    return li[r]


def compMove():
    possibleMoves = [x for x, letter in enumerate(board) if letter == ' '
and x != 0]
    move = 0

    #Check for possible winning move to take or to block opponents
winning move
    for let in ['O','X']:
        for i in possibleMoves:
            boardCopy = board[:]
            boardCopy[i] = let
            if isWinner(boardCopy, let):
                move = i
                return move


    #Try to take one of the corners
    cornersOpen = []
    for i in possibleMoves:
        if i in [1,3,7,9]:
            cornersOpen.append(i)
    if len(cornersOpen) > 0:
        move = selectRandom(cornersOpen)
        return move

    #Try to take the center
    if 5 in possibleMoves:
        move = 5
        return move

    #Take any edge
    edgesOpen = []
```

```
    for i in possibleMoves:
        if i in [2,4,6,8]:
            edgesOpen.append(i)

    if len(edgesOpen) > 0:
        move = selectRandom(edgesOpen)

    return move


def isBoardFull(board):
    if board.count(' ') > 1:
        return False
    else:
        return True

def printBoard():
    # "board" is a list of 10 strings representing the board (ignore
index 0)
    print('   |   |')
    print(' ' + board[1] + ' | ' + board[2] + ' | ' + board[3])
    print('   |   |')
    print('-----------')
    print('   |   |')
    print(' ' + board[4] + ' | ' + board[5] + ' | ' + board[6])
    print('   |   |')
    print('-----------')
    print('   |   |')
    print(' ' + board[7] + ' | ' + board[8] + ' | ' + board[9])
    print('   |   |')


def main():
    #Main game loop
    print('Welcome to Tic Tac Toe, to win complete a straight line of
your letter (Diagonal, Horizontal, Vertical). The board has positions 1-9
starting at the top left.')
    printBoard()

    while not(isBoardFull(board)):
        if not(isWinner(board, 'O')):
            playerMove()
            printBoard()
        else:
            print('Os win this time...')
            break


        if not(isWinner(board, 'X')):
```

```
            move = compMove()
            if move == 0:
                print('Game is a Tie! No more spaces left to move.')
            else:
                insertBoard('O', move)
                print('Computer placed an O in position', move, ':')
                printBoard()
        else:
            print('X wins, good job!')
            break


    if isBoardFull(board):
        print('Game is a tie! No more spaces left to move.')
```

Output:

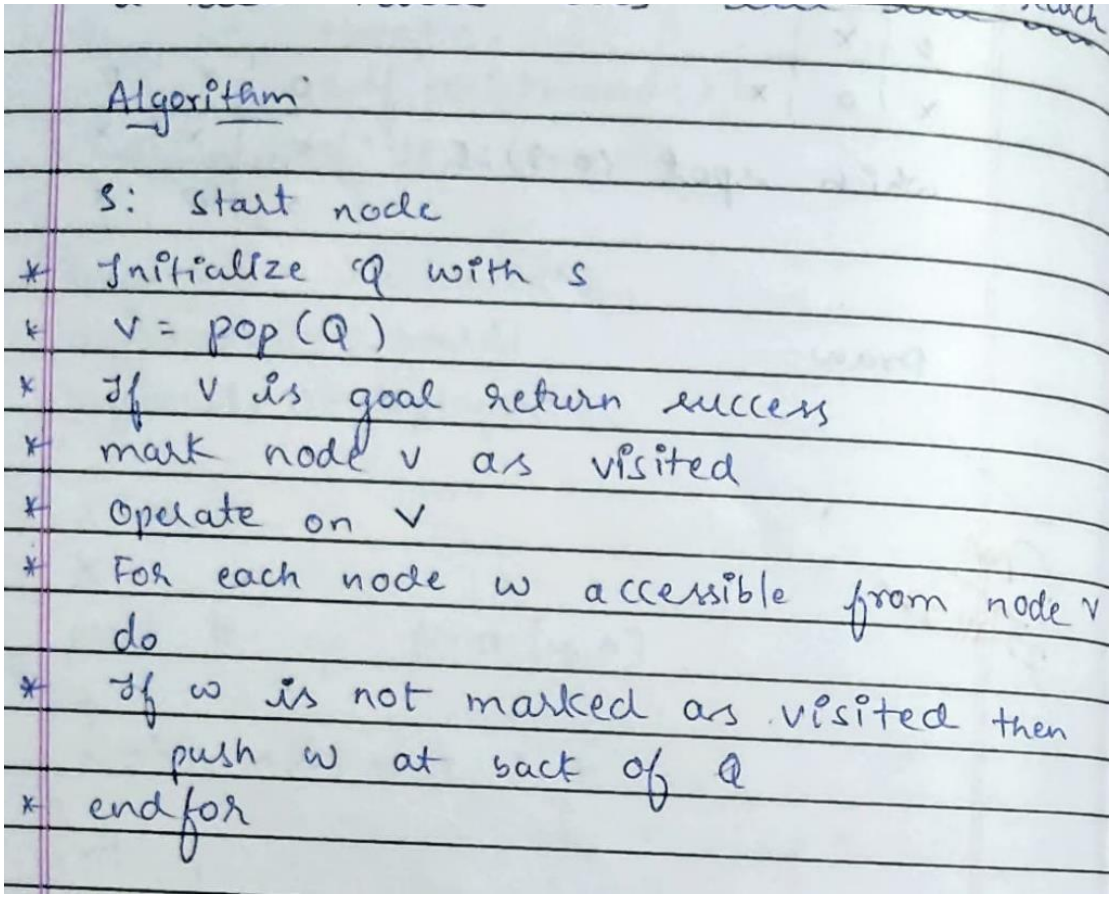

Welcome to Tic Tac Toe, to win complete a straight line of your le
Please select a position to place an \'X\' (1-9):1
Computer placed an O in position 3 :


Please select a position to place an \'X\' (1-9):5
X | X | O
| X |
Computer placed an O in position 6 :
X | X | O
| X | O
| | O
Os win this time...

Date: 16/11/22

Program 2: Solve 8 puzzle problem.

Algorithm:

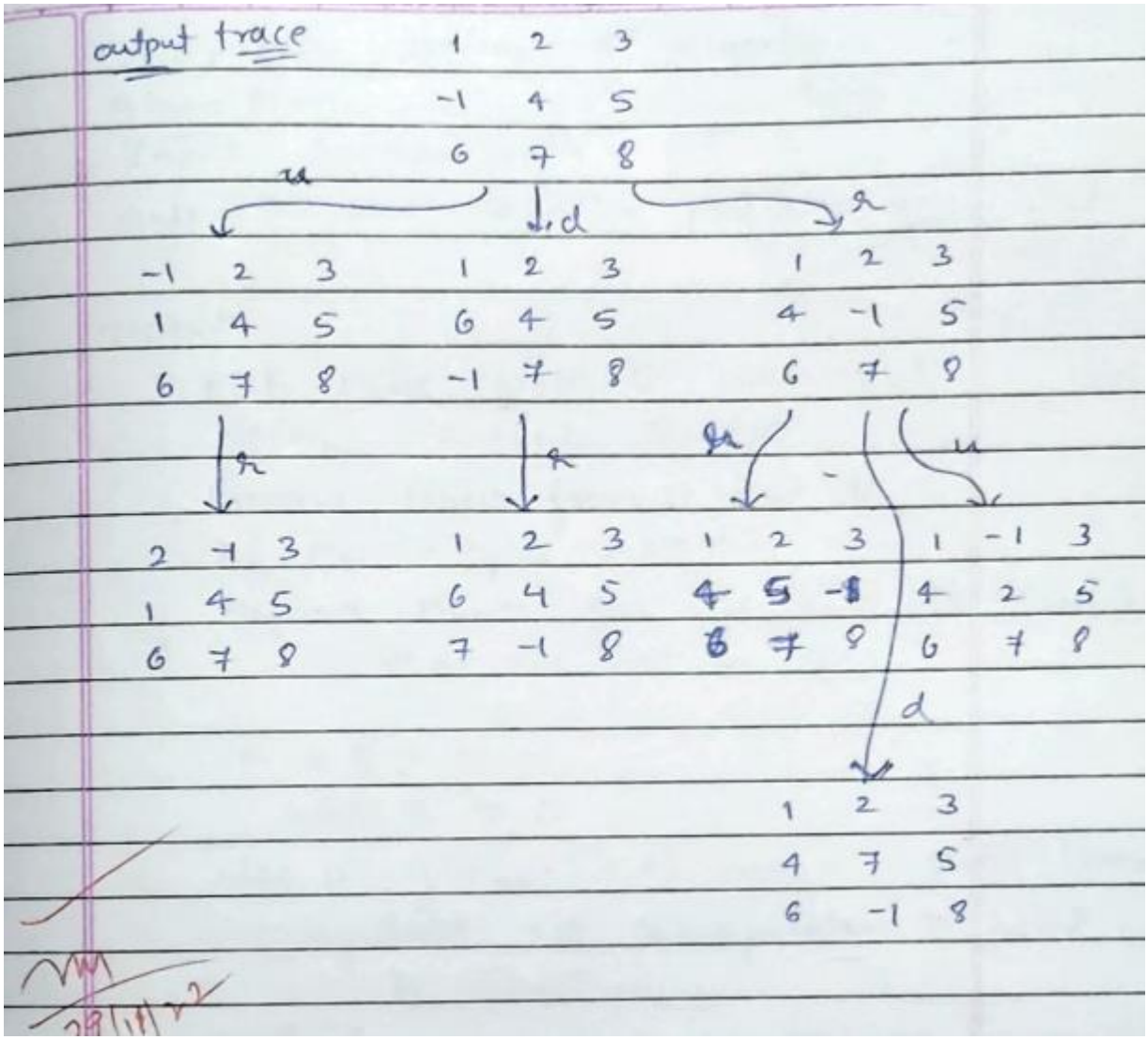


Algorithm

s: start node

* Initialize Q with s
* v = pop(Q)
* If v is goal return success
* mark node v as visited
* Operate on v
* For each node w accessible from node v do
* If w is not marked as visited then push w at back of Q
* endfor

State Space Tree:



Code:

```
def bsf(src,target):
   queue = []
   queue.append(src)

   exp = []

   while(len(queue)>0):
      source = queue.pop(0)
      exp.append(source)

      print(source)

      if source == target:
         print("Success")
         return
```

```
        poss_moves_to_do = []
        poss_moves_to_do = possible_moves(source,exp)

        for move in poss_moves_to_do:
            if move not in exp and move not in queue:
                queue.append(move)

def possible_moves(state,visited_states):
    b = state.index(-1)

    d = []
    if b not in [0,1,2]:
        d.append('u')
    if b not in [6,7,8]:
        d.append('d')
    if b not in [0,3,6]:
        d.append('1')
    if b not in [2,5,8]:
        d.append('r')

    poss_moves_it_can = []

    for i in d:
        poss_moves_it_can.append(gen(state,i,b))

    return [move_it_can for move_it_can in poss_moves_it_can if move_it_can not in
visited_states]

def gen(state,m,b):
    temp = state.copy()

    if m=='d':
        temp[b+3],temp[b] = temp[b],temp[b+3]
    if m=='u':
        temp[b-3],temp[b] = temp[b],temp[b-3]
    if m=='l':
        temp[b-1],temp[b] = temp[b],temp[b-1]
    if m=='r':
        temp[b+1],temp[b] = temp[b],temp[b+1]

    return temp
```

src = [1,2,3,-1,4,5,6,7,8]
target = [1,2,3,4,5,-1,6,7,8]
bsf(src,target)

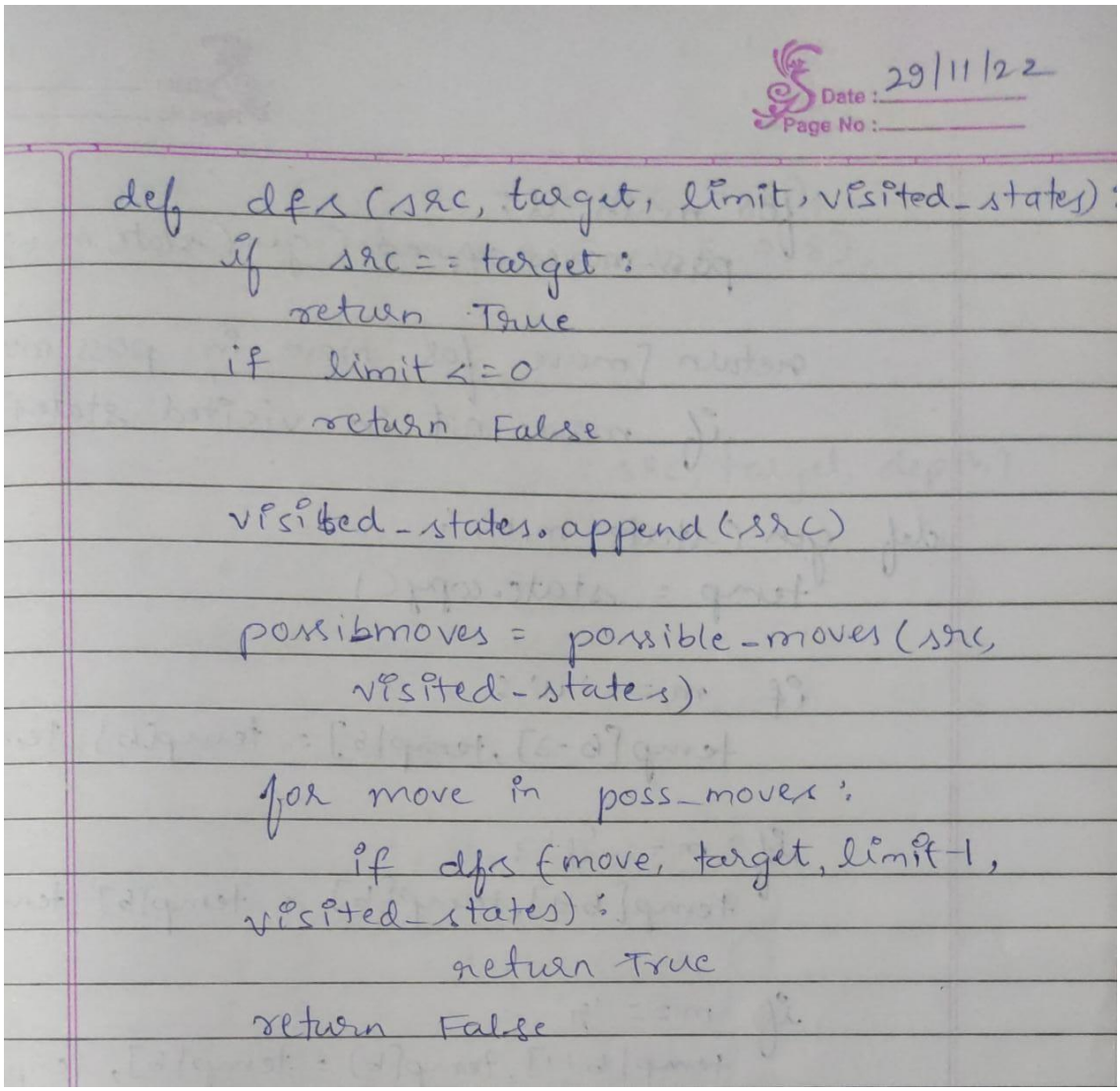
Output:

```
OUTPUT

C:\Shashanka G\puzzle8>python 8puzzle.py
[1, 2, 3, -1, 4, 5, 6, 7, 8]
[-1, 2, 3, 1, 4, 5, 6, 7, 8]
[1, 2, 3, 6, 4, 5, -1, 7, 8]
[1, 2, 3, 4, -1, 5, 6, 7, 8]
[2, -1, 3, 1, 4, 5, 6, 7, 8]
[1, 2, 3, 6, 4, 5, 7, -1, 8]
[1, -1, 3, 4, 2, 5, 6, 7, 8]
[1, 2, 3, 4, 7, 5, 6, -1, 8]
[1, 2, 3, 4, 5, -1, 6, 7, 8]
Success

C:\Shashanka G\puzzle8>
```

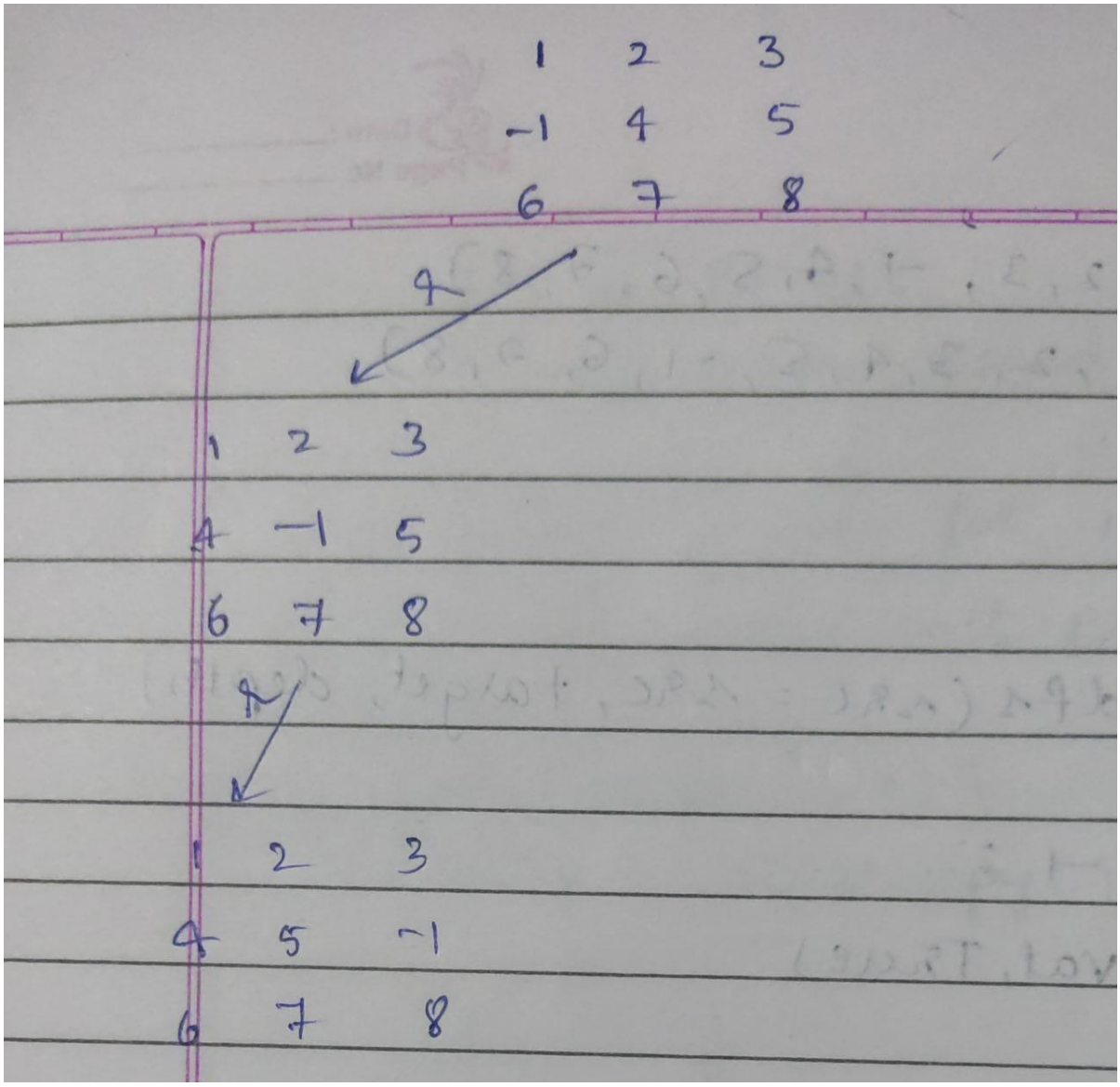
Date: 23/11/22

Program 3: Implement Iterative deepening search algorithm.

Algorithm:



State Space Tree:



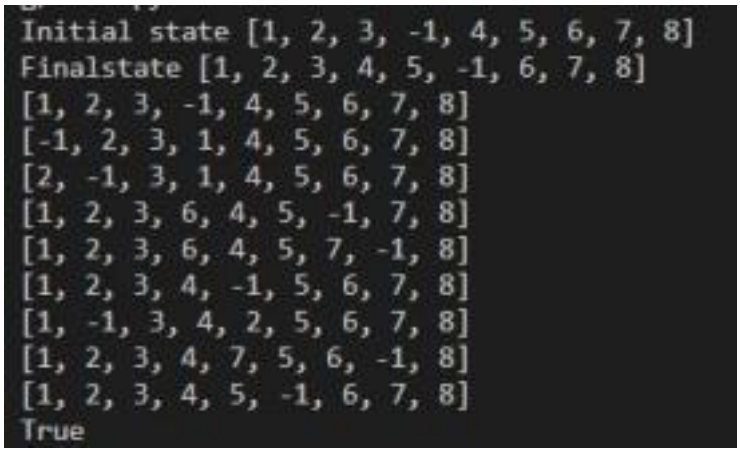
Code:

```
def dfs(src,target,limit,visited_states):
    print(visited_states)
    if src == target:
        return True
    if limit<=0:
        return False
    visited_states.append(src)
    poss_moves = possible_moves(src,visited_states)
```

```
    for move in poss_moves:
        if dfs(move,target,limit-1,visited_states):
            return True
    return False

def possible_moves(state,visited_states):
    b = state.index(-1)
    d = []
    if b not in [2,5,8]:
        d.append('r')
    if b-3 in range(9):
        d.append('u')
    if b not in [0,3,6]:
        d.append('l')
    if b+3 in range(9):
        d.append('d')
    pos_moves = []
    for m in d:
        pos_moves.append(gen(state,m,b))
    return [move for move in pos_moves if move not in visited_states]

def gen(state,m,b):
    temp = state.copy()
    if m == 'u':
        temp[b-3],temp[b] = temp[b],temp[b-3]
    if m == 'l':
        temp[b-1],temp[b] = temp[b],temp[b-1]
    if m == 'r':
        temp[b+1],temp[b] = temp[b],temp[b+1]
    if m == 'd':
        temp[b+3],temp[b] = temp[b], temp[b+3]
    return temp

def IDdfs(src,target,depth):
    visited_states = []
```

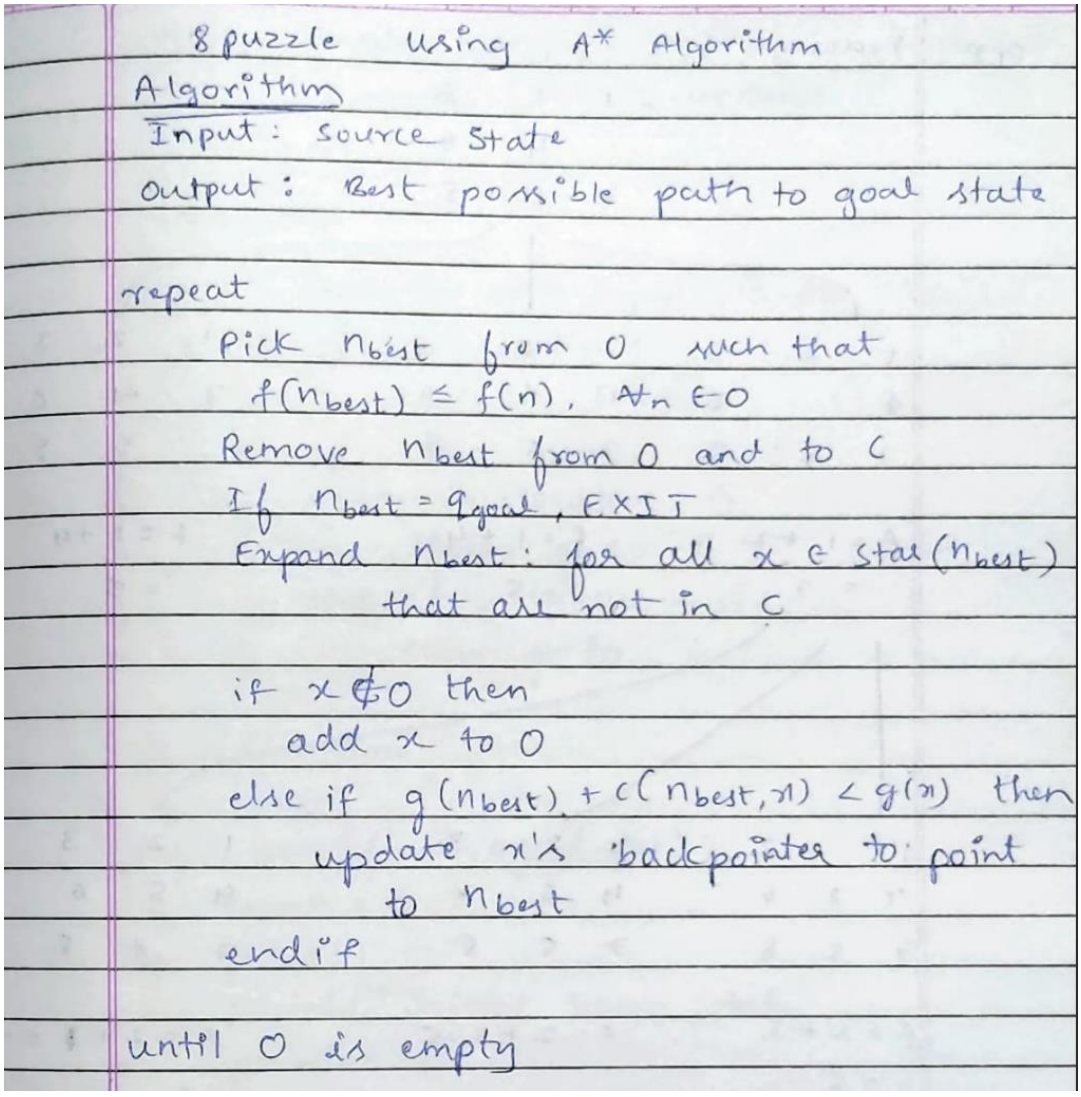
```
    for i in range(1,depth+1):
        if dfs(src,target,i,visited_states):
            return i
    return -1

src = [1,2,3,-1,4,5,6,7,8]
target = [1,2,3,4,5,-1,6,7,8]
depth = 25
val = IDdfs(src=src,target=target,depth=depth)
if val != -1:
    print(val,True)
else:
    print(False)
```

Output:



Date: 30/11/22

Program 4: Implement A* search algorithm



8 puzzle using A* Algorithm

Algorithm

Input : Source State

output : Best possible path to goal state

repeat

Pick n_{best} from O such that $f(n_{best}) \leq f(n)$, $\forall n \in O$

Remove n_{best} from O and to C

If $n_{best} = q_{goal}$, EXIT

Expand n_{best} : for all $x \in Star(n_{best})$ that are not in C

if $x \notin O$ then

add x to O

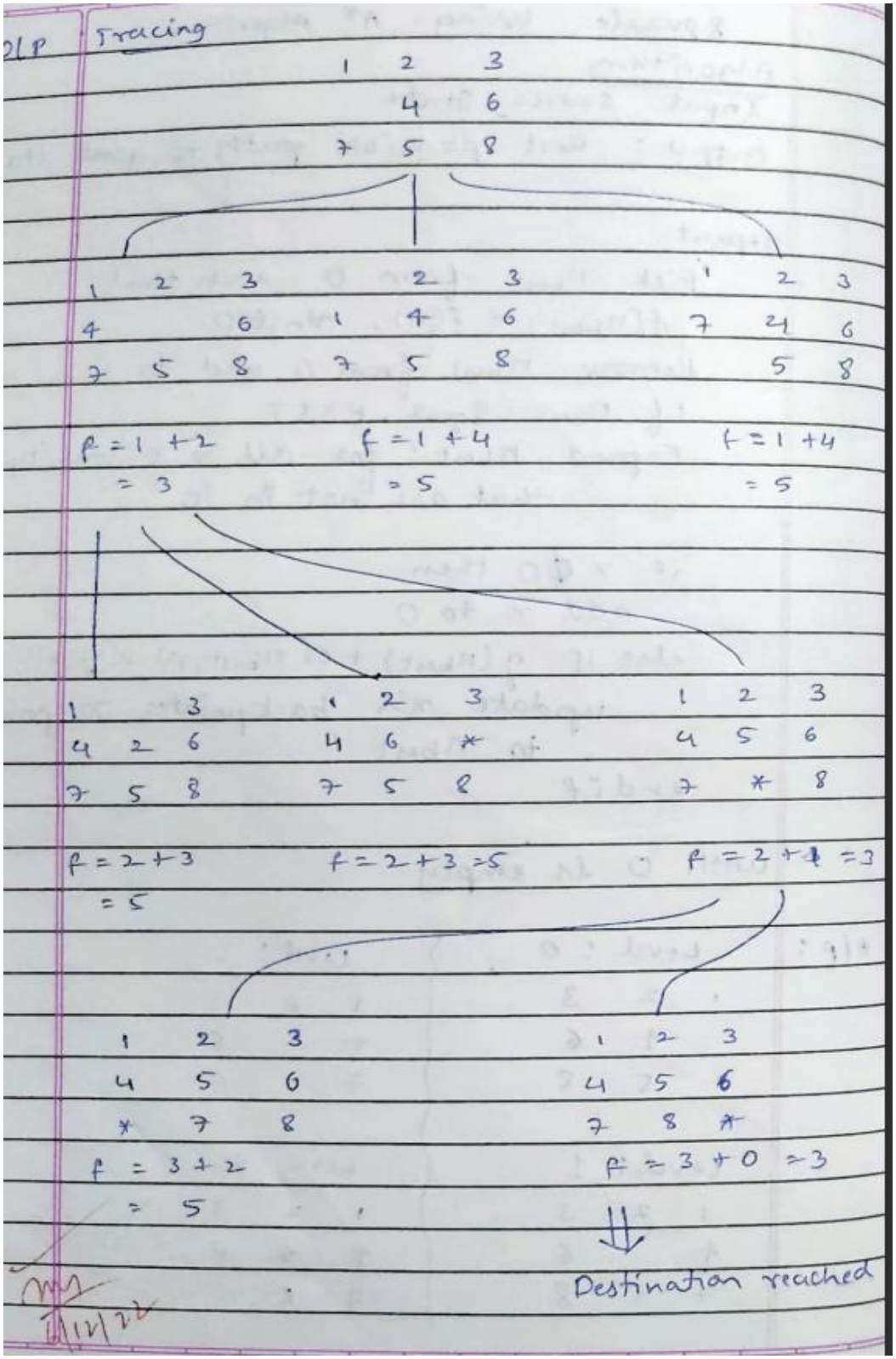
else if $g(n_{best}) + c(n_{best}, x) < g(x)$ then

update x's backpointer to point to n_{best}

endif

until O is empty

State Space Tree:



Code:

```
def print_b(src):
    state = src.copy()
    state[state.index(-1)] = ' '
    print(
        f"""
{state[0]} {state[1]} {state[2]}
{state[3]} {state[4]} {state[5]}
{state[6]} {state[7]} {state[8]}
        """
    )


def h(state, target):
    count=0
    for i in range(9):
```

```
        if state[i] != target[i]:
            count=count+1
    return count


def astar(state,target):# Add inputs if more are required
    states = [src]
    g = 0
    visited_states =[]
    while len(states):
        print(f"Level: {g}")
        moves = []
        for state in states:
            visited_states.append(state)
            print_b(state)
            if state == target:
                print("Success")
```

```
            return

        moves += [move for move in possible_moves(state, visited_states) if move not in moves]

    costs = [ h(move, target) for move in moves]

    states = [moves[i] for i in range(len(moves)) if costs[i] == min(costs)]

    g += 1

print("Fail")




def possible_moves(state, visited_state):  # Add inputs if more are required

    # Find index of empty spot and assign it to b

    b = state.index(-1)



    # 'd' for down, 'u' for up, 'r' for right, 'l' for left - directions array

    d = []



    # Add all possible direction into directions array - Hint using if statements
```

```
if b - 3 in range(9):
    d.append('u')
if b not in [0, 3, 6]:
    d.append('l')
if b not in [2, 5, 8]:
    d.append('r')
if b + 3 in range(9):
    d.append('d')



# If direction is possible then add state to move
pos_moves = []



# for all possible directions find the state if that move is played
### Jump to gen function to generate all possible moves in the given directions
for m in d:
    pos_moves.append(gen(state, m, b))
```

```
    # return all possible moves only if the move not in visited_states
    return [move for move in pos_moves if move not in visited_state]




def gen(state, m, b):
    temp = state.copy()



    # if move is to slide empty spot to the left and so on
    if m == 'u': temp[b - 3], temp[b] = temp[b], temp[b - 3]
    if m == 'l': temp[b - 1], temp[b] = temp[b], temp[b - 1]
    if m == 'r': temp[b + 1], temp[b] = temp[b], temp[b + 1]
    if m == 'd': temp[b + 3], temp[b] = temp[b], temp[b + 3]



    # return new state with tested move to later check if "src == target"
    return temp
```

Test 1

src = [1,2,3,-1,4,6,7,5,8]

target = [1,2,3,4,5,6,7,8,-1]

astar(src, target)

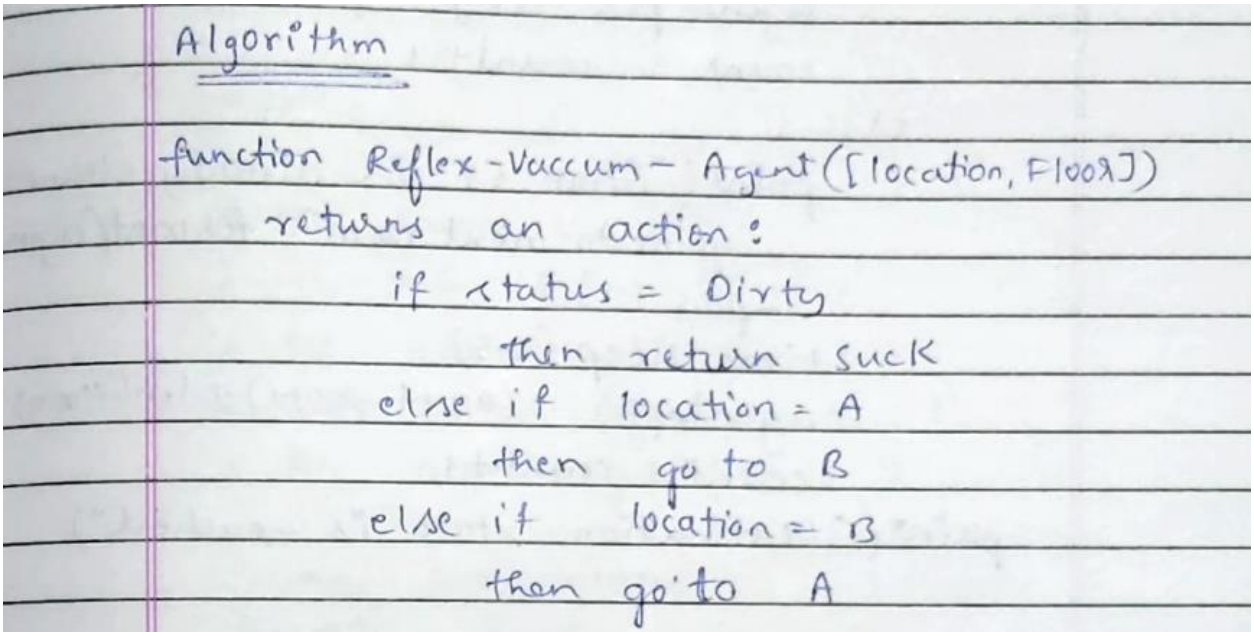
Output:

```
D:\Shashanka G\puzzle8_astart>python 8puzzle.py
Level: 0

1 2 3
  4 6
7 5 8

Level: 1

1 2 3
4   6
7 5 8

Level: 2

1 2 3
4 5 6
7   8

Level: 3

1 2 3
4 5 6
7 8

Success

D:\Shashanka G\puzzle8_astart>
```

Date: 7/12/22

Program 5: Implement vacuum cleaner agent.

Algorithm:



CODE

```
import time

def clean(floor,agent_pos):
    count = 0
    print()
    print("Ïnitial room status")
    print_floor(floor)
    agent_pos = (agent_pos-1) % len(floor)
    while count<len(floor):
        if all([x==0 for x in floor]):
            print()
            print("All rooms are cleaned")
            print()
            break
         # if room is dirty
        if floor[agent_pos] == 1:
            print()
            print("Room {} is dirty. Agent is cleaning now".format(agent_pos+1))
            time.sleep(2.5)
            floor[agent_pos] = 0
```

```
            print_floor(floor)
            agent_pos = (agent_pos+1) % len(floor)
            count = count+1
        # If room is cleaned
        else:
            print("Room {} is already cleaned, going to next room".format(agent_pos+1))
            time.sleep(0.5)
            agent_pos = (agent_pos+1)%len(floor)
            count = count+1
    print("Destination state is reached")

def print_floor(floor):
    status_map = {0:"CLEAN",1:"DIRTY"}
    for i in range(len(floor)):
        print("Room {}".format(i+1),end='  ')
    print()
    for i in range(len(floor)):
        print(status_map[floor[i]],end='\t')
    print()

def main():
    #Entre initial condition
    floor = [None,None]
    print("Enter room status\nDirty-->1 Clean-->0")
    floor[0] = int(input("Enter status of room 1:"))
    floor[1] = int(input("Enter status of room 2:"))
    agent_pos = int(input("Enter agent position:"))
    clean(floor,agent_pos)

main()
```

Output:

```
D:\Shashanka G\vaccume_cleaner>python vaccume_cleaner.py
Enter room status
Dirty-->1 Clean-->0
Enter status of room 1:1
Enter status of room 2:1
Enter agent position:1

Ïnitial room status
Room 1  Room 2
DIRTY   DIRTY

Room 1 is dirty. Agent is cleaning now
Room 1  Room 2
CLEAN   DIRTY

Room 2 is dirty. Agent is cleaning now
Room 1  Room 2
CLEAN   CLEAN
Destination state is reached



D:\Shashanka G\vaccume_cleaner>python vaccume_cleaner.py
Enter room status
Dirty-->1 Clean-->0
Enter status of room 1:1
Enter status of room 2:0
Enter agent position:2

Ïnitial room status
Room 1  Room 2
DIRTY   CLEAN
Room 2 is already cleaned, going to next room

Room 1 is dirty. Agent is cleaning now
Room 1  Room 2
CLEAN   CLEAN
Destination state is reached
```

Date: 28/12/22

Program 6: Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.

Algorithm and ouput

Date :
Page No :

Algorithm

function entailment (KB, α) returns true or false

inputs KB, the knowledge base

symbols ← a list of preposition symbols

function checkall (KB, α, symbol, model)
returns true or false

if empty (symbols) then
if PLTRUE? (KB, model)
then return true
else do
p ← First (symbols)
rest ← REST (symbols)
return checkall (KB, α, rest, model)
and
checkall (KB, α, rest, model)

O/P: Enter rule: (p∨q) ∧ (¬p∨q)
Enter query: p∧q

Knowledge base does not entail query

Code:

T = True

F = False

combinations = [

(T,T,T),

(T,T,F),

(T,F,T),

```
    (T,F,F),

    (F,T,T),

    (F,T,F),

    (F,F,T),

    (F,F,F)]

variable = {"p":0,'q':1,"r":2}

kb=''
```

```
q=''

priority={'~':3,'^':2,'v':1}

def input_rules():

    global kb,q

    kb = input("Knowlwdge base:")

    q=input("query:")
```

```
def entailment():

    global kb,q

    print(''*10+"Truth table reference"+""*10)

    print('kb  α')

    print("-"*10)

    for combination in combinations:

        s = evaluatePostfix(toPostfix(kb),combination)
```

```
        f = evaluatePostfix(toPostfix(kb),combination)




        print(s,f)




        if s is True and f is False:




            return False




    return True




def isOperand(c):




    return c.isalpha() and c!='v'
```

```
def isLeftParanthesis(c):
    return ')' == c

def isRightParanthesis(c):
    return ')' == c

def isEmpty(stack):
```

```
    return len(stack)==0

def peek(stack):

    return stack[-1]

def hasLessOrEqualPriority(c1,c2):

    try:

        return priority[c1]<=priority[c2]
```

```
    except KeyError:

        return False

def toPostfix(infix):

    stack = []

    postfix = ''

    for c in infix:

        if isOperand(c):

            postfix += c
```

```
else:


    if isLeftParanthesis(c):


        stack.append(c)


    elif isRightParanthesis(c):


        operator = stack.pop()


        while not isLeftParanthesis(operator):


            postfix += operator


            operator = stack.pop()
```

```
        else:

            while (not isEmpty(stack)) and hasLessOrEqualPriority(c, peek(stack)):

                postfix += stack.pop()

            stack.append(c)

while (not isEmpty(stack)):

    postfix += stack.pop()

return postfix
```

```
def evaluatePostfix(exp, comb):

    stack = []

    for i in exp:

        if isOperand(i):

            stack.append(comb[variable[i]])

        elif i == '~':

            val1 = stack.pop()

            stack.append(not val1)
```

```
        else:

            val1 = stack.pop()

            val2 = stack.pop()

            stack.append(_eval(i,val2,val1))

    return stack.pop()

def _eval(i, val1, val2):

    if i == '^':
```

```
        return val2 and val1


    return val2 or val1


input_rules()


ans = entailment()


if ans:


    print("The Knowledge Base entails query")


    print(" KB |= α ")
```

```
else:
    print("The Knowledge Base does not entail query")
print("\n")
```

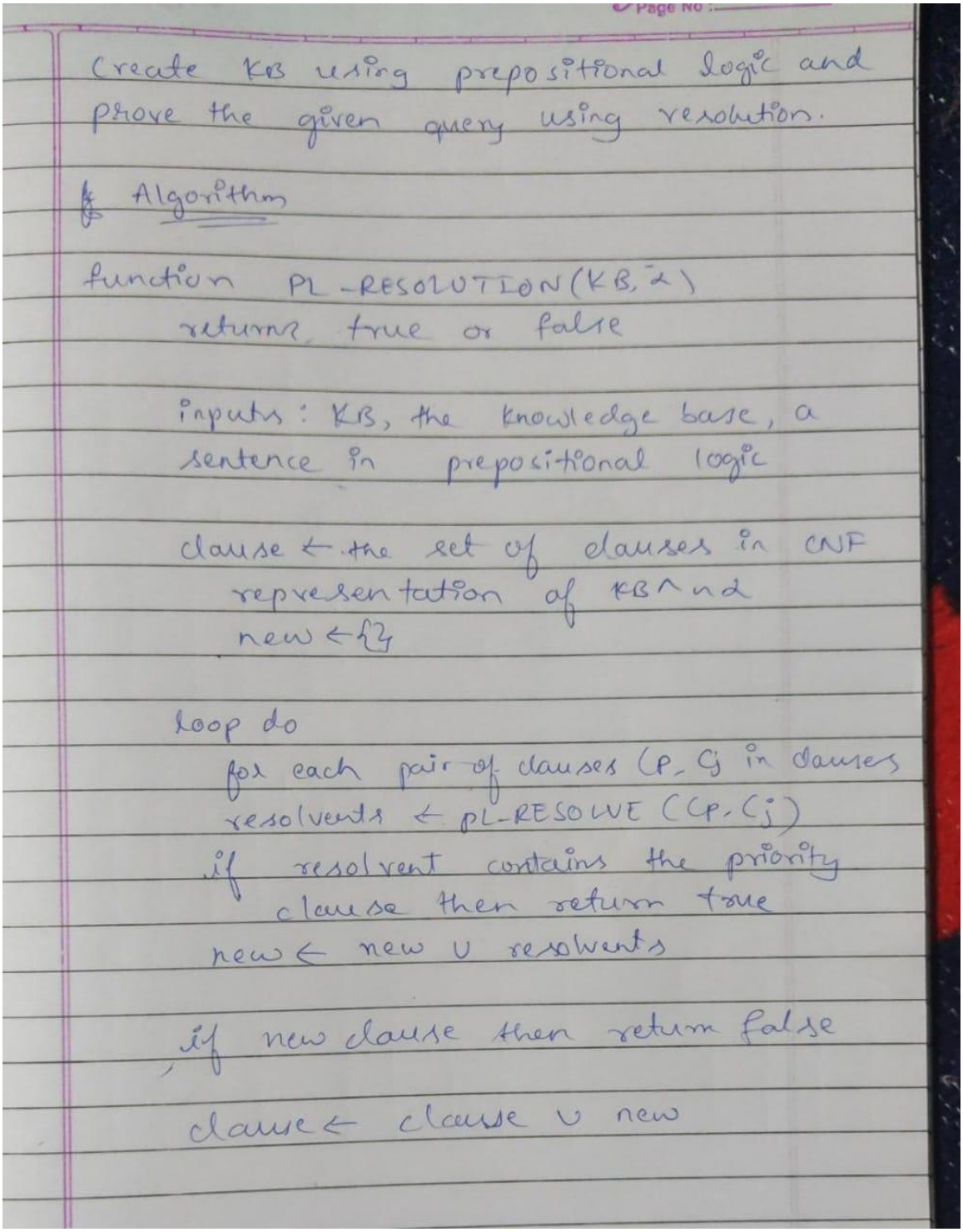

Output:

```
D:\Shashanka G\27-12-2022>python main.py
Knowlwdge base:p^q^r
query:pvq
Truth table reference
kb  α
----------
True True
False False
False False
False False
False False
False False
False False
False False
The Knowledge Base entails query
 KB |= α
```

Date: 3/1/23

Program 7: Create a knowledge base using propositional logic and prove the given query using resolution

Algorithm:



Page No :

Create KB using propositional logic and prove the given query using resolution.

& Algorithm

function PL-RESOLUTION(KB, α)
returns true or false

inputs: KB, the knowledge base, a sentence in propositional logic

clause ← the set of clauses in CNF representation of KB ^ ¬α
new ← {}

loop do
for each pair of clauses (Ci, Cj) in clauses
resolvents ← PL-RESOLVE(Ci, Cj)
if resolvent contains the priority clause then return true
new ← new ∪ resolvents

if new clause then return false

clause ← clause ∪ new

Code:

```
def disjunctify(clauses):
    disjuncts = []
    for clause in clauses:
        disjuncts.append(tuple(clause.split('v')))
    return disjuncts

def getResolvant(ci, cj, di, dj):
    resolvant = list(ci) + list(cj)
    resolvant.remove(di)
    resolvant.remove(dj)
    return tuple(resolvant)

def resolve(ci, cj):
    for di in ci:
        for dj in cj:
            if di == '~' + dj or dj == '~' + di:
                return getResolvant(ci, cj, di, dj)


def checkResolution(clauses, query):
    clauses += [query if query.startswith('~') else '~' + query]
    proposition = '^'.join(['(' + clause + ')' for clause in clauses])
    print(f'Trying to prove {proposition} by contradiction....')

    clauses = disjunctify(clauses)
    resolved = False
    new = set()

    while not resolved:
        n = len(clauses)
        pairs = [(clauses[i], clauses[j]) for i in range(n) for j in range(i + 1, n)]
        for (ci, cj) in pairs:
            resolvant = resolve(ci, cj)
            if not resolvant:
                resolved = True
                break
            new = new.union(set(resolvents))
        if new.issubset(set(clauses)):
            break
        for clause in new:
            if clause not in clauses:
                clauses.append(clause)
```

```
    if resolved:
        print('Knowledge Base entails the query, proved by resolution')
    else:
        print("Knowledge Base doesn't entail the query, no empty set produced after resolution")

#Test Case 1
clauses = input('Enter the clauses ').split()
query = input('Enter the query: ')
checkResolution(clauses, query)
```

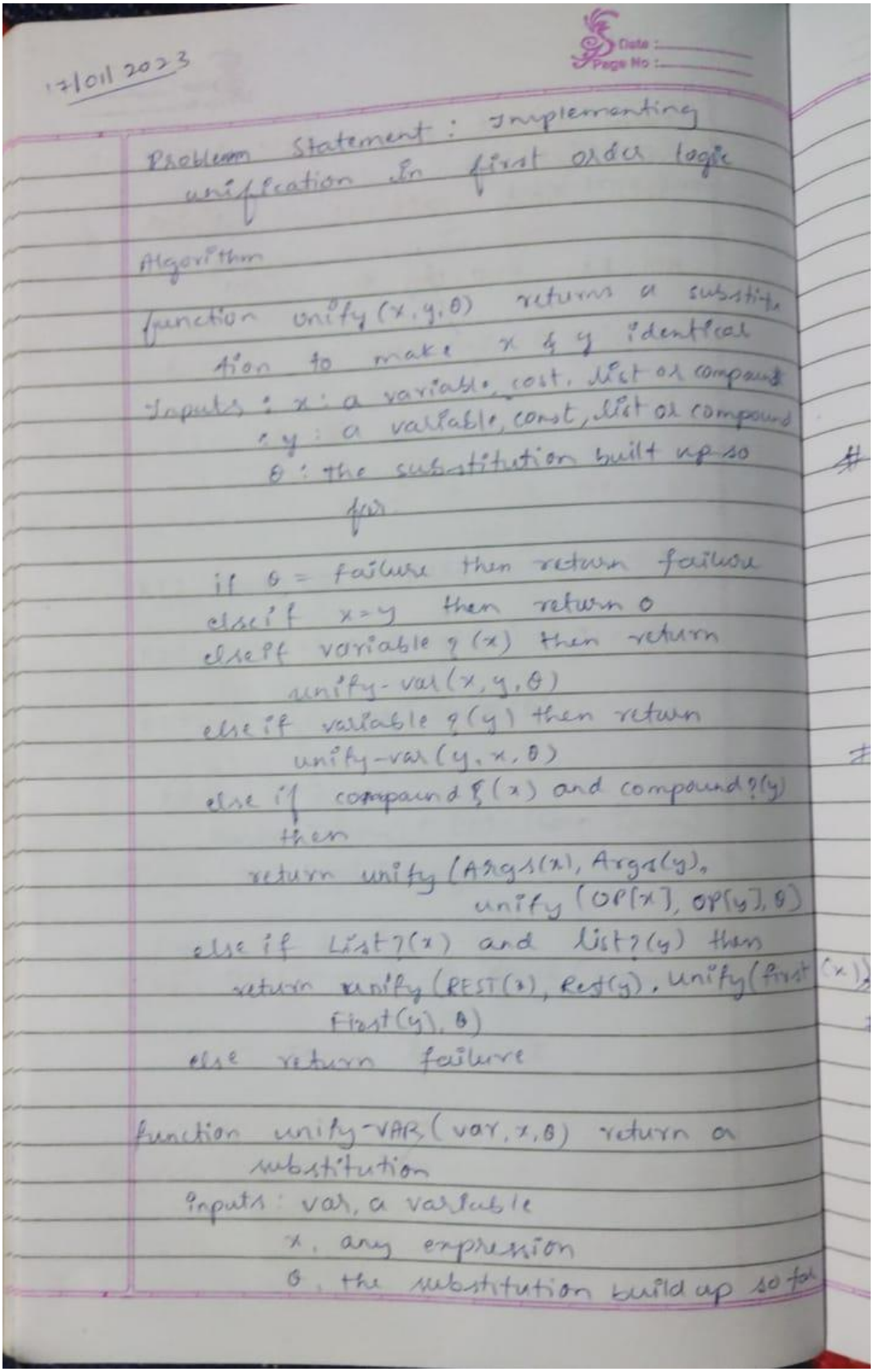
Output:

```
Enter the clauses ~Qv~PvR ~Q^P Q
Enter the query: Q
Trying to prove (~Qv~PvR)^(~Q^P)^(Q)^(~Q) by contradiction....
Knowledge Base entails the query, proved by resolution
```

Date: 10/1/23

Program 8: Implement unification in first order logic.

Algorithm:



17/01/2023

Problem Statement: Implementing unification in first order logic

Algorithm

function unify(x, y, θ) returns a substitution to make x & y identical

Inputs: x: a variable, const, list or compound
y: a variable, const, list or compound
θ: the substitution built up so far

if θ = failure then return failure
elseif x = y then return θ
elseif variable?(x) then return unify-var(x, y, θ)
else if variable?(y) then return unify-var(y, x, θ)
else if compound?(x) and compound?(y) then
return unify(Args1(x), Args2(y), unify(OP[x], OP[y], θ)
else if List?(x) and list?(y) then
return unify(REST(x), Rest(y), unify(first(x), First(y), θ)
else return failure

function unify-VAR(var, x, θ) return a substitution
Inputs: var, a variable
x, any expression
θ, the substitution build up so far

if {var/val} ∈ θ then return unify(val, x, θ)
else if {x/val} ∈ θ then return unify(var, val, θ)
else if occur-check?(var, x) then return failure
else return add {var/x} to θ

o/p:

#1 exps1 = "pmof(c,p)" # c-country, p-person
exps2 = "pmof(India, Modi)"
subs = unify(exp1, exp2)
print('substitutions')
o
substitutions:
(India/c, modi/p)

#2 exp1 = "pmof(c, p)"
exp2 = "pmof(c, p)"
subs = unify(exp1, exp2)

substitutions:
[] # since no need to change

#3 exp1 = "King(x)"
exp2 = "King(John)"

substitutions:
[John/x]

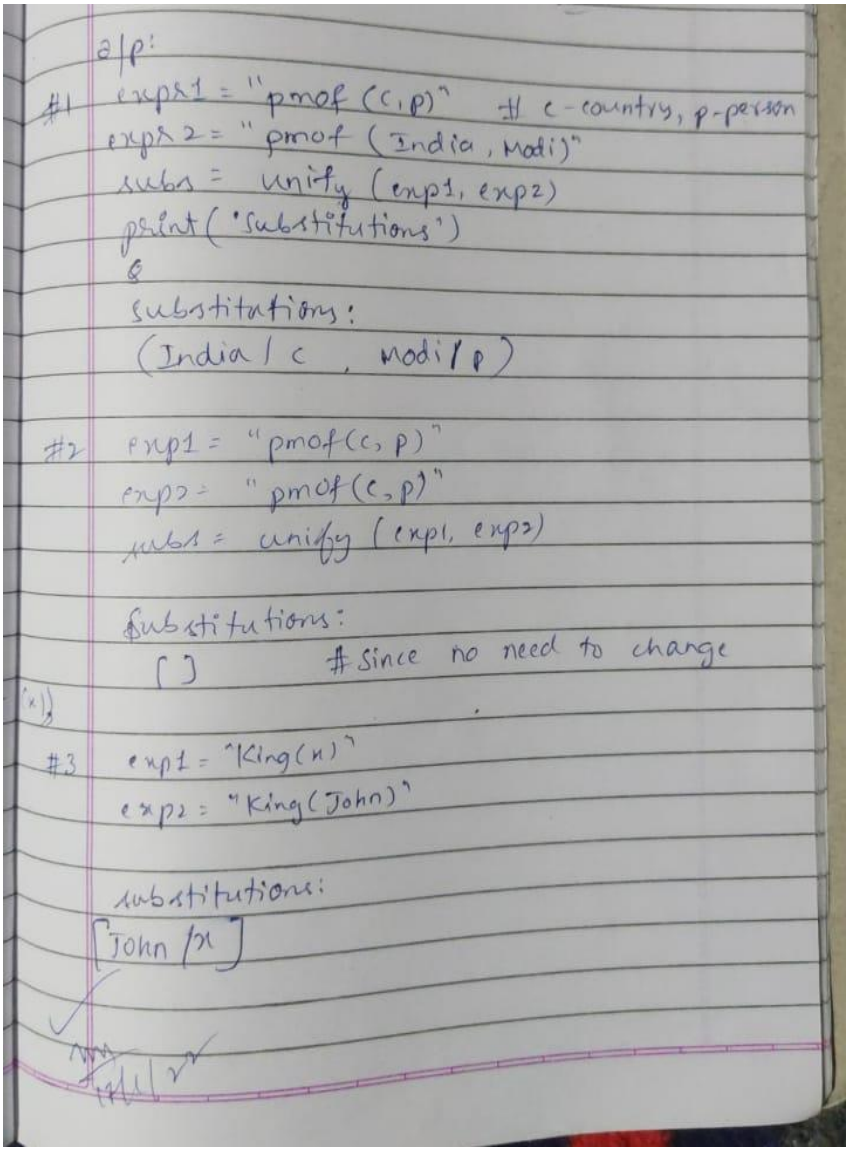
Code

```
import re

def getAttributes(expr):
    expr = expr.split("(")[1:]
    expr = "(".join(expr)
    expr = expr[:-1]
    expr = re.split("(?<!.),(?!.)", expr)
    return expr

def getInitialPredicate(expr):
    return expr.split("(")[0]

def isConstant(char):
    return char.isupper() and len(char) == 1

def isVariable(char):
    return char.islower() and len(char) == 1

def replaceAttributes(expr, old, new):
    attr = getAttributes(expr)
    for index, val in enumerate(attr):
        if val == old:
            attr[index] = new
    predicate = getInitialPredicate(expr)
    return predicate + "(" + ",".join(attr) + ")"

def apply(expr, subs):
    for sub in subs:
        new, old = sub  #substitution is a tuple of 2 values (new, old)
        expr = replaceAttributes(expr, old, new)
    return expr

def checkOccurs(var, expr):
    if expr.find(var) == -1:
        return False
    return True

def getFirstPart(expr):
    attr = getAttributes(expr)
    return attr[0]

def getRemainingPart(expr):
```

```
    predicate = getInitialPredicate(expr)
    attr = getAttributes(expr)
    newExpr = predicate + "(" + ",".join(attr[1:]) + ")"
    return newExpr

def unify(exp1, exp2):
    if exp1 == exp2:
        return []

    if isConstant(exp1) and isConstant(exp2):
        if exp1 != exp2:
            return False

    if isConstant(exp1):
        return [(exp1, exp2)]

    if isConstant(exp2):
        return [(exp2, exp1)]

    if isVariable(exp1):
        if checkOccurs(exp1, exp2):
            return False
        else:
            return [(exp2, exp1)]

    if isVariable(exp2):
        if checkOccurs(exp2, exp1):
            return False
        else:
            return [(exp1, exp2)]

    if getInitialPredicate(exp1) != getInitialPredicate(exp2):
        print("Cannot be unified")
        return False

    attributeCount1 = len(getAttributes(exp1))
    attributeCount2 = len(getAttributes(exp2))
    if attributeCount1 != attributeCount2:
        return False

    head1 = getFirstPart(exp1)
    head2 = getFirstPart(exp2)
    initialSub = unify(head1, head2)
    if not initialSub:
        return False
```

```
    if attributeCount1 == 1:
        return initialSub

    tail1 = getRemainingPart(exp1)
    tail2 = getRemainingPart(exp2)

    if initialSub != []:
        tail1 = apply(tail1, initialSub)
        tail2 = apply(tail2, initialSub)

    remainingSub = unify(tail1, tail2)
    if not remainingSub:
        return False

    initialSub.extend(remainingSub)
    res = []
    for tup in initialSub:
      st = ' / '.join(tup)
      res.append(st)

    return res

#Test Case 1
exp1 = "pmof(c,p)"  # c--->country   p--->person
exp2 = "pmof(c,p)"
subs = unify(exp1, exp2)
print("Substitutions:")
print(subs)
```

Output:



o/p:

#1
```
expr1 = "pmof (c,p)"   # c-country, p-person
expr2 = "pmof (India, Modi)"
subs = unify (exp1, exp2)
print('substitutions')
```

substitutions:

(India / c , Modi / p)

#2
```
exp1 = "pmof(c, p)"
exp2 = "pmof(c, p)"
subs = unify (exp1, exp2)
```

substitutions:

[ ] # since no need to change

#3
```
exp1 = "King(x)"
exp2 = "King(John)"
```

substitutions:

[John / x]

Date: 10/1/23

Program 9: Convert given first order logic statement into Conjunctive Normal Form (CNF).

Algorithm:

FOL TO CNF Algorithm

&FOL to CNF

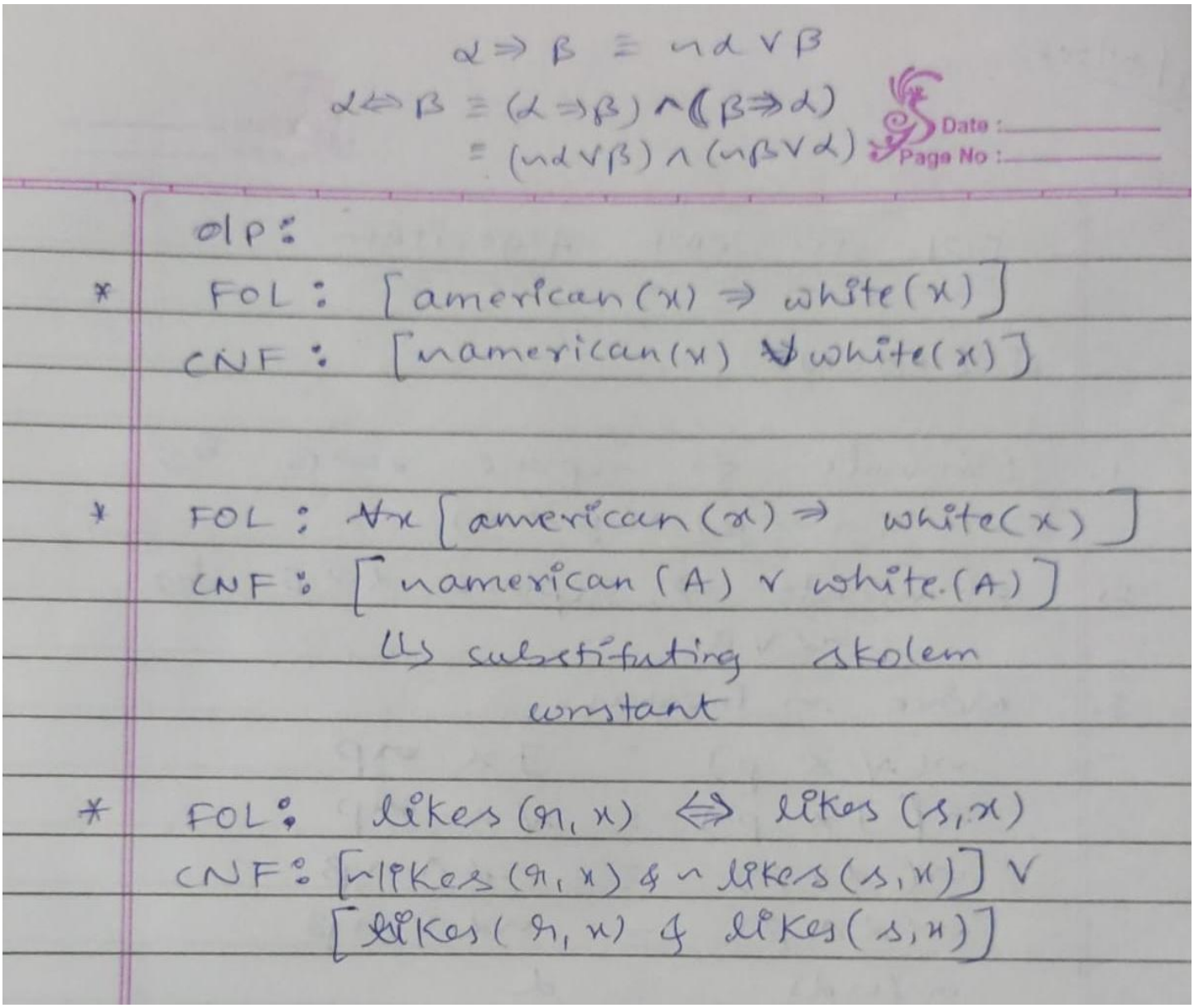
1. Eliminate $\Leftrightarrow$, replace $\alpha \Leftrightarrow \beta$ by
   $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
2. Eliminate $\Rightarrow$, replace $\alpha \Rightarrow \beta$ by
   $\neg\alpha \vee \beta$
3. Move $\neg$ inwards
   $\neg(\forall x\ p) \equiv \exists x \neg p$
   $\neg[\exists x\ p) \equiv \forall x \neg p$
   $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
   $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$
   $\neg(\neg\alpha) \equiv \alpha$
4. Standardize variables apart by them : each quantifier should use different variable.
5. Skolemization : Each existential variable is replaced by skolem constant or skolem function of the enclosing universally quantified variables.
   eg : $\exists x$ Rich(x) becomes Rich(G1)
6. Drop universal quantifiers
   eg : $\forall x$ Person becomes Person(x)
7) Distribute $\wedge$ over $\vee$:
   $(\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$

Code:

```
def getAttributes(string):
    expr = '
'
    matches = re.findall(expr, string)
    return [m for m in str(matches) if m.isalpha()]

def getPredicates(string):
    expr = '[a-z~]+
'
    return re.findall(expr, string)

def DeMorgan(sentence):
    string = ''.join(list(sentence).copy())
    string = string.replace('~~','')
    flag = '[' in string
    string = string.replace('~[','')
    string = string.strip(']')
    for predicate in getPredicates(string):
        string = string.replace(predicate, f'~{predicate}')
    s = list(string)
    for i, c in enumerate(string):
        if c == '|':
            s[i] = '&'
        elif c == '&':
            s[i] = '|'
    string = ''.join(s)
    string = string.replace('~~','')
    return f'[{string}]' if flag else string

def Skolemization(sentence):
    SKOLEM_CONSTANTS = [f'{chr(c)}' for c in range(ord('A'), ord('Z')+1)]
    statement = ''.join(list(sentence).copy())
    matches = re.findall('[∀∃].', statement)
    for match in matches[::-1]:
        statement = statement.replace(match, '')
        statements = re.findall('
]', statement)
        for s in statements:
            statement = statement.replace(s, s[1:-1])
        for predicate in getPredicates(statement):
            attributes = getAttributes(predicate)
            if ''.join(attributes).islower():
                statement = statement.replace(match[1],SKOLEM_CONSTANTS.pop(0))
            else:
                aL = [a for a in attributes if a.islower()]
                aU = [a for a in attributes if not a.islower()][0]
                statement = statement.replace(aU, f'{SKOLEM_CONSTANTS.pop(0)}({aL[0] if len(aL) else match[1]})')
    return statement
```

```
import re

def fol_to_cnf(fol):

    statement = fol.replace("<=>", "_")
    while '_' in statement:
        i = statement.index('_')
        new_statement = '[' + statement[:i] + '=>' + statement[i+1:] + ']&['+ statement[i+1:] + '=>' + statement[:i] + ']'
        statement = new_statement
    statement = statement.replace("=>", "-")
    expr = '
'
    statements = re.findall(expr, statement)
    for i, s in enumerate(statements):
        if '[' in s and ']' not in s:
            statements[i] += ']'
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
    while '-' in statement:
        i = statement.index('-')
        br = statement.index('[') if '[' in statement else 0
        new_statement = '~' + statement[br:i] + '|' + statement[i+1:]
        statement = statement[:br] + new_statement if br > 0 else new_statement
    while '~∀' in statement:
        i = statement.index('~∀')
        statement = list(statement)
        statement[i], statement[i+1], statement[i+2] = '∃', statement[i+2], '~'
        statement = ''.join(statement)
    while '~∃' in statement:
        i = statement.index('~∃')
        s = list(statement)
        s[i], s[i+1], s[i+2] = '∀', s[i+2], '~'
        statement = ''.join(s)
    statement = statement.replace('~[∀','[~∀')
    statement = statement.replace('~[∃','[~∃')
    expr = '(~[∀|∃].)'
    statements = re.findall(expr, statement)
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
    expr = '~
'
    statements = re.findall(expr, statement)
    for s in statements:
        statement = statement.replace(s, DeMorgan(s))
    return statement

#Interactive Test Cases
n = int(input())
while n:
  statement = input("Enter FOL statement: ")
  print(f"FOL converted to CNF: {Skolemization(fol_to_cnf(statement))} \n\n")
  n -= 1
```

Output


α ⇒ β ≡ ¬α ∨ β
α ⇔ β ≡ (α ⇒ β) ∧ (β ⇒ α)
≡ (¬α ∨ β) ∧ (¬β ∨ α)
Date :
Page No :
o/p:
* FOL : [american(x) ⇒ white(x)]
CNF : [¬american(x) ∨ white(x)]
* FOL : ∀x [american(x) ⇒ white(x)]
CNF : [¬american(A) ∨ white(A)]
By substituting skolem constant
* FOL : likes(a1, x) ⇔ likes(s, x)
CNF : [¬likes(a1, x) ∨ ¬likes(s, x)] ∨
[likes(a1, x) ∨ likes(s, x)]

Date: 10/1/23

Program 10: Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

Algorithm:

Forward Reasoning Algorithm.

function FOL-FC-ASK(KB, α) returns a
a substitution or false
repeat until new is empty
new ← { }

for each sentence q in KB do
(p1 ∧ ... ∧ pn ⇒ q) ← Standardize-Apart(s)

for each θ such that (p1 ∧ ... ∧ pn)θ
= (p'1 ∧ ... ∧ p'n)θ for some
p'1 ... p'n in KB

q' ← Subst(θ, q)

if q' is not a renaming of a
sentence already in KB or new then
do

add q' to new
φ ← Unify(q', α)
if φ is not fail then return φ

add new to KB

return false.

Code:

```
import re
def isVariable(x):
    return len(x) == 1 and x.islower() and x.isalpha()

def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr, string)
    return matches

def getPredicates(string):
    expr = '([a-z~]+)\([^&|]+\)'
    return re.findall(expr, string)

class Fact:
    def __init__(self, expression):
        self.expression = expression
        predicate, params = self.splitExpression(expression)
        self.predicate = predicate
        self.params = params
        self.result = any(self.getConstants())

    def splitExpression(self, expression):
        predicate = getPredicates(expression)[0]
        params = getAttributes(expression)[0].strip('()').split(',')
        return [predicate, params]

    def getResult(self):
        return self.result

    def getConstants(self):
        return [None if isVariable(c) else c for c in self.params]

    def getVariables(self):
        return [v if isVariable(v) else None for v in self.params]

    def substitute(self, constants):
        c = constants.copy()
        f = f"{self.predicate}({','.join([constants.pop(0) if isVariable(p) else p for p in self.params])})"
        return Fact(f)

class Implication:
    def __init__(self, expression):
        self.expression = expression
        l = expression.split('=>')
        self.lhs = [Fact(f) for f in l[0].split('&')]
        self.rhs = Fact(l[1])

    def evaluate(self, facts):
        constants = {}
```

```
        new_lhs = []
        for fact in facts:
            for val in self.lhs:
                if val.predicate == fact.predicate:
                    for i, v in enumerate(val.getVariables()):
                        if v:
                            constants[v] = fact.getConstants()[i]
                    new_lhs.append(fact)
        predicate, attributes = getPredicates(self.rhs.expression)[0],
str(getAttributes(self.rhs.expression)[0])
        for key in constants:
            if constants[key]:
                attributes = attributes.replace(key, constants[key])
        expr = f'{predicate}{attributes}'
        return Fact(expr) if len(new_lhs) and all([f.getResult() for f in new_lhs]) else None

class KB:
    def __init__(self):
        self.facts = set()
        self.implications = set()

    def tell(self, e):
        if '=>' in e:
            self.implications.add(Implication(e))
        else:
            self.facts.add(Fact(e))
        for i in self.implications:
            res = i.evaluate(self.facts)
            if res:
                self.facts.add(res)

    def query(self, e):
        facts = set([f.expression for f in self.facts])
        i = 1
        print(f'Querying {e}:')
        for f in facts:
            if Fact(f).predicate == Fact(e).predicate:
                print(f'\t{i}. {f}')
                i += 1

    def display(self):
        print("All facts: ")
        for i, f in enumerate(set([f.expression for f in self.facts])):
            print(f'\t{i+1}. {f}')

#Test Case 1
kb = KB()
kb.tell('missile(x)=>weapon(x)')
kb.tell('missile(M1)')
kb.tell('enemy(x,America)=>hostile(x)')
kb.tell('american(West)')
kb.tell('enemy(Nono,America)')
```

```
kb.tell('owns(Nono,M1)')
kb.tell('missile(x)&owns(Nono,x)=>sells(West,x,Nono)')
kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')
kb.query('criminal(x)')
kb.display()

#Test Case 2
kb_ = KB()
kb_.tell('king(x)&greedy(x)=>evil(x)')
kb_.tell('king(John)')
kb_.tell('greedy(John)')
kb_.tell('king(Richard)')
kb_.query('evil(x)')
```

Output: