VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

AI Lab Report

Submitted by

Kanjika Singh(1BM21CS086)

Under the Guidance of

Dr Asha G.R,
Assistant Professor,Department of CSE

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Nov-2023 to Feb-2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Artificial Intelligence**" carried out by **Kanjika Singh (1BM21CS086)** , who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24.

The Lab report has been approved as it satisfies the academic requirements in respect of **Artificial Intelligence (22CS5PC)** work prescribed for the said degree.

Dr. Asha G.R.	Dr. Jyothi Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, Kanjika Singh (1BM21CS086), student of 5th Semester, B.E, Department ofComputer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Artificial Intelligence**" has been carried out by me under the guidance of Prof. Sunayana S., Assistant Professor, Department of CSE, B. M. S. College ofEngineering, Bangalore during the academic semester November-2023-February-2024.

I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

Table of Contents

Program 1: Implement Tic Tac Toe

Code:

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

def insertLetter(letter, pos):
    board[pos] = letter

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

def printBoard(board):
    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 isWinner(bo, le):
    return (bo[7] == le and bo[8] == le and bo[9] == le) or (bo[4] == le
and
    bo[5] == le and bo[6] == le) or (bo[1] == le and bo[2] == le and
bo[3] == le) or (bo[1] == le and
    bo[4] == le and bo[7] == le) or (
    bo[2] == le and bo[5] == le and bo[8] == le) or (
    bo[3] == le and bo[6] == le and bo[9] == le) or (
    bo[1] == le and bo[5] == le and bo[9] == le) or (bo[3] ==
    le and bo[5] == le and bo[7] == le)

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
                    insertLetter('X', move)
                else:
                    print('Sorry, this space is occupied!')
            else:
                print('Please type a number within the range!')
        except:
            print('Please type a number!')

def compMove():
    possibleMoves = [x for x, letter in enumerate(board) if letter == ' '
and x
    != 0]
    move = 0
    for let in ['O', 'X']:
        for i in possibleMoves:
            boardCopy = board[:]
            boardCopy[i] = let
            if isWinner(boardCopy, let):
                move = i
                return move
    cornersOpen = []
    for i in possibleMoves:
        if i in [1, 3, 7, 9]:
            cornersOpen.append(i)
    if len(cornersOpen) > 0:
        move = selectRandom(cornersOpen)
        return move
    if 5 in possibleMoves:
        move = 5
        return move
    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 selectRandom(li):
    import random
    ln = len(li)
    r = random.randrange(0, ln)
    return li[r]

def isBoardFull(board):
    if board.count(' ') > 1:
```

```
        return False
    else:
        return True

def main():
    print('Welcome to Tic Tac Toe!')
    printBoard(board)
    while not (isBoardFull(board)):
        if not (isWinner(board, 'O')):
            playerMove()
            printBoard(board)
        else:
            print('Sorry, O\'s won this time!')
            break
    if not (isWinner(board, 'X')):
        move = compMove()
        if move == 0:
            print('Tie Game!')
        else:
            insertLetter('O', move)
            print('Computer placed an \'O\' in position', move, ':')
            printBoard(board)
    else:
        print('X\'s won this time! Good Job!')
    if isBoardFull(board):
        print('Tie Game!')

while True:
    answer = input('Do you want to play again? (Y/N)')
    if answer.lower() == 'y' or answer.lower() == 'yes':
        board = [' ' for x in range(10)]
        print('-----------------------------------')
        main()
    else:
        break
```

Observation:

Date : 17-11-2023 KA 17-11-23

Program 3 : Implement Tic Tac Toe

```
import random
board = [' ' for i in range(10)]

def insert(letter, pos):
    board[pos] = letter

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

def print(board):
    print(' | | ')
    print(' ' + board[1] + ' | ' + board[2] + ' | ' + board[3])
    print('-----------')
    print(' ' + board[4] + ' | ' + board[5] + ' | ' +
          board[6])
    print("-----------")
    print(' | | ')
    print(' ' + board[7] + ' | ' + board[8] + ' | ' +
          board[9])
    print(' | | ')
```

```
def is_winner(board, len):
    return(board[1] == len and board[2] == len and board[3] ==
                                                        len)
                    or
    (board[4] == len and board[5] = len and board[6] == len)
                    or
    (board[7] == len and board[8] == len and board[9] = len)
                    or
    (board[1] = =len and board[4] == len and board[7] = len)
                    or
    (board[2] = len and board[5] = len and board[8] = = len)
                    or
    (board[3] == len and board[6] = len and board[9] = = len)


def player1():
    run = True
    while run:
        move = input("Enter a position to place an 'X'(1-9)")
        try:
            move = int(move)
            if move > 0 and move < 10:
                if space_free(move):
                    run = False
                    insert('X', move)
                else:
                    print("Occupied")
                else:
```

Date: 17-11-2023

```
def CompMove():
    possible = [x for x, letter in enumerate(
        board) if letter == " " and x != 0]
    move

def compMove():
    run = True
    while run:
        move = random.randint(1,10)
        if (move > 0 and move <10):
            if spaceIsFree(move):
                run = False
                insertLetter('O', move)
            else:
                continue
        else:
            continue
```

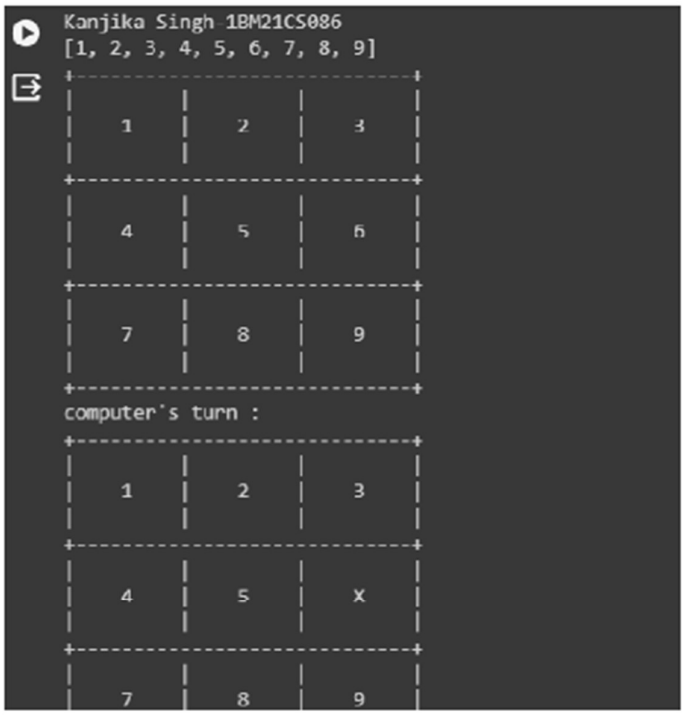
```
if not (board.count(' ') <10):
        playerMove()
        printBoard(board)
        if (isWinner(board, 'X')):
            print('You won')
            break
    else else:
            compMove()
            printBoard(board)
            if (isWinner(board,'O')):
                print("Computer won")
                break
    else:
        print("Tie this is")
```

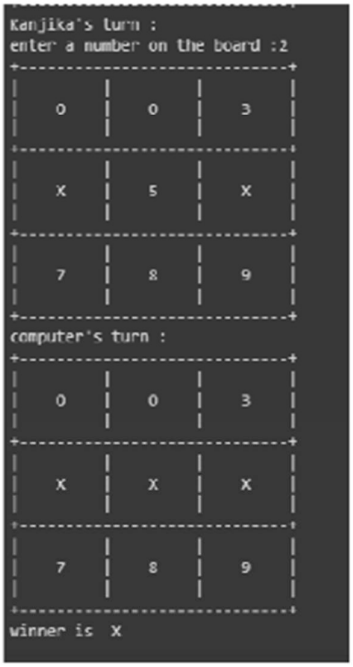
Date : 17-11-2023

Algorithm : Tic Tac Toe

- Create a 3×3 board consisting of empty space
- Create function insert() to insert a letter to the board and space_free() to check if position is free
- First allow player to play
  - If the board is free, insert X
  - Then check if move leads to the player to win or not
  - If the player does not wins, give computer the chance to play
- Continue till the board is empty.

Output:





Program 2 : 8 Puzzle Breadth First Search Algorithm

Code:

```
def bfs(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):
    #index of empty spot
    b = state.index(-1)

    #directions array
    d = []
    #Add all the possible directions

    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('l')
    if b not in [2,5,8]:
        d.append('r')
```

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

    # 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 i in d:
        pos_moves_it_can.append(gen(state,i,b))

    return [move_it_can for move_it_can in pos_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 new state with tested move to later check if "src == target"
    return temp
src = [1,2,3,-1,4,5,6,7,8]
target = [1,2,3,4,5,-1,6,7,8]
bfs(src, target)
```

Observation:

Date: 24/11/23

24-11-23

8 Puzzle Problem Using BFS

In the problem, the square with have N+1 tiles where N = 8, 15, 24 so on

N=8 means square will have 9 tiles (3 rows and 3 columns)

In the problem, initial state will be given and and we have do reach goal state.

Suppose

Initial state

1	2	3
	4	6
7	5	8

Goal state

1	2	3
4	5	6
7	8	

Rules

- Empty space can move in 4 directions
  1) Up 2) Down 3) Right 4) Left
- It cannot move diagonally
- It can only take one step at a time

o	x	o
x	#	x
o	x	o

Tiles at o — no. of possible moves=2
Tiles at x — no. of possible moves=3
Tile at # — no. of possible moves=4

Using Breadth first search, which is uninformed search. This problem is solved by non heuristic approach.

Complexity $O(b^d)$ where
b — branching factor
d depth

Date : 24/11/23

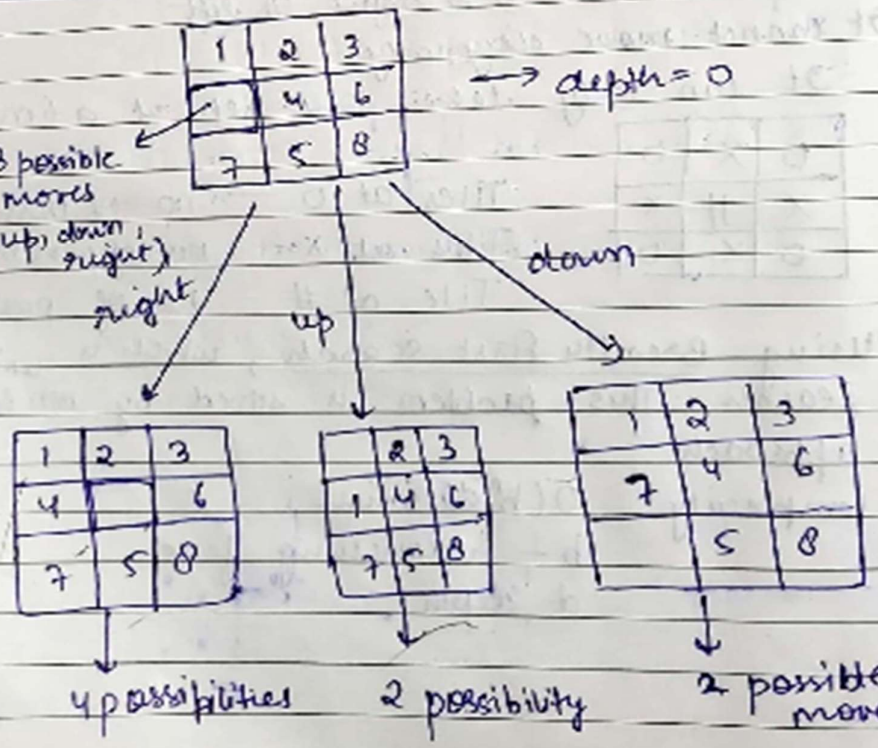
For 8 puzzle

Branching factor b = all possible moves of empty tile at each position / no. of tiles

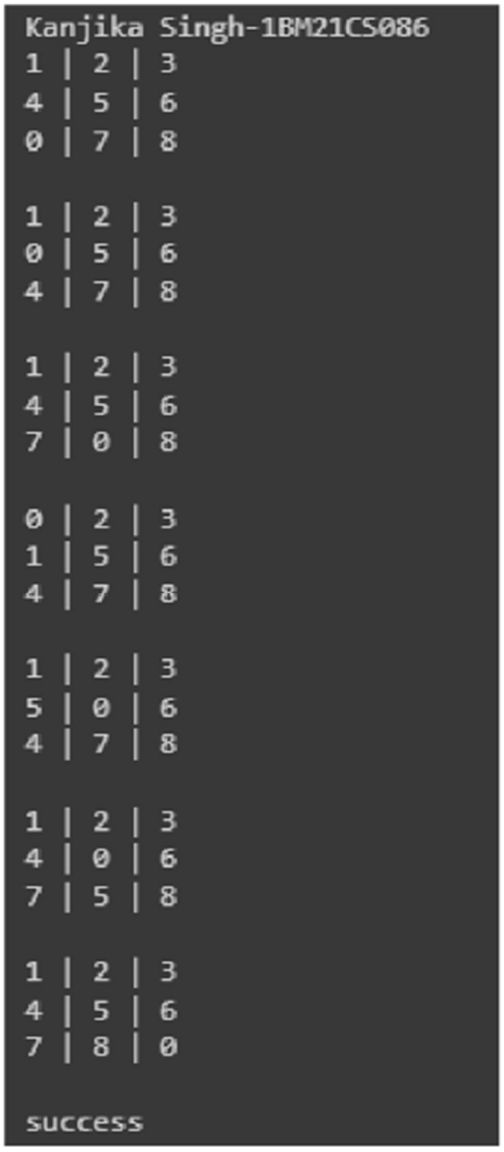
$= \frac{24}{9} = 2.67 \sim 3$

depth factor = initially 0
As we explore the nodes, depth factor increases
Every time, we check all possible moves of tiles branch it and consider every case and finally the goal state

For example:



Output:


Kanjika Singh-1BM21CS086
1 | 2 | 3
4 | 5 | 6
0 | 7 | 8
1 | 2 | 3
0 | 5 | 6
4 | 7 | 8
1 | 2 | 3
4 | 5 | 6
7 | 0 | 8
0 | 2 | 3
1 | 5 | 6
4 | 7 | 8
1 | 2 | 3
5 | 0 | 6
4 | 7 | 8
1 | 2 | 3
4 | 0 | 6
7 | 5 | 8
1 | 2 | 3
4 | 5 | 6
7 | 8 | 0
success

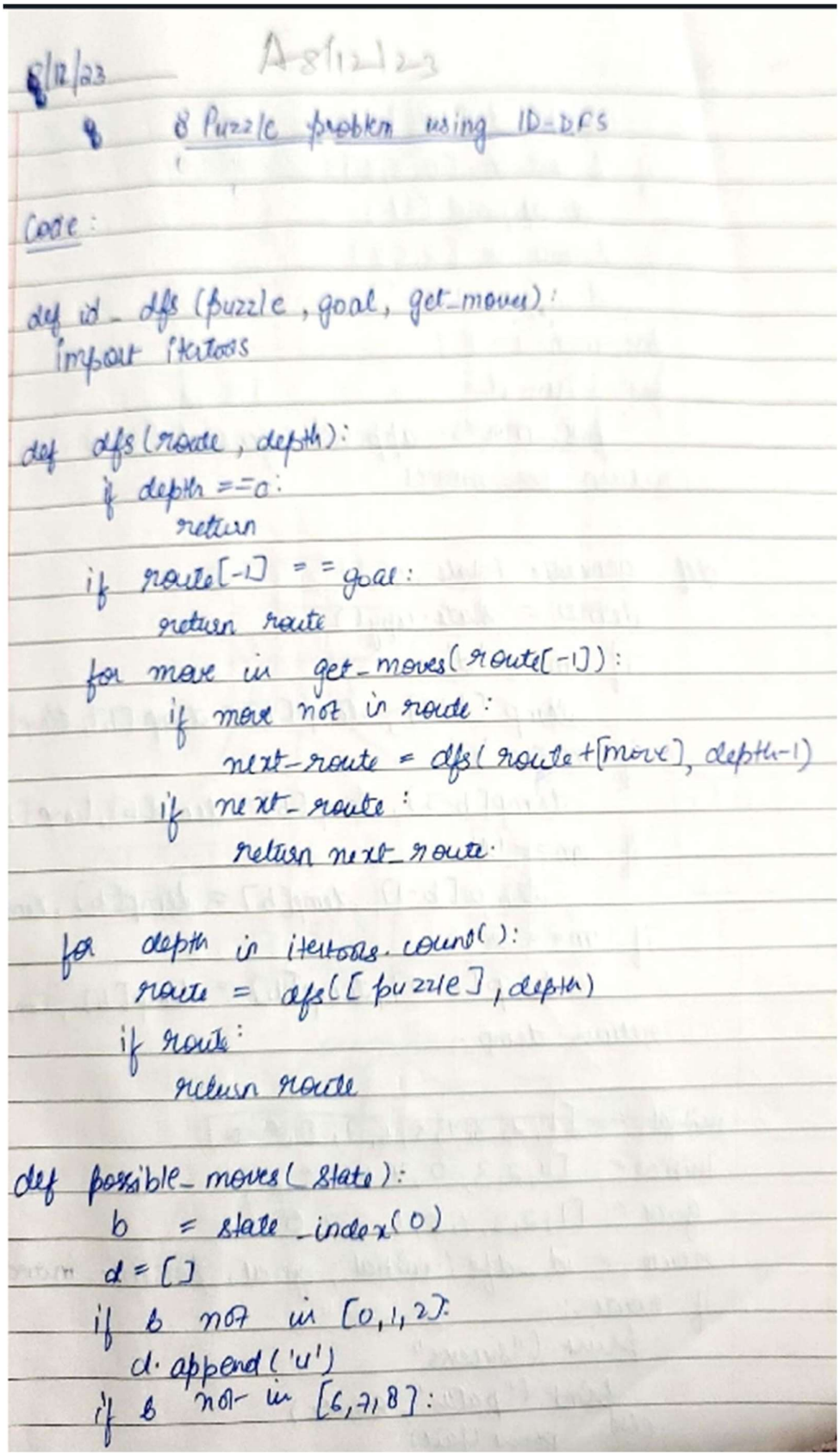
Program 3 : 8 Puzzle Iterative Deepening Search Algorithm

Code:

```
# 8 Puzzle problem using Iterative deepening depth first search algorithm

def id_dfs(puzzle, goal, get_moves):
    import itertools
#get_moves -> possible_moves
    def dfs(route, depth):
        if depth == 0:
            return
        if route[-1] == goal:
            return route
        for move in get_moves(route[-1]):
            if move not in route:
                next_route = dfs(route + [move], depth - 1)
                if next_route:
                    return next_route

    for depth in itertools.count():
        route = dfs([puzzle], depth)
        if route:
            return route


def possible_moves(state):
    b = state.index(0)  # ) indicates White space -> so b has index of it.
    d = []  # direction

    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('l')
    if b not in [2, 5, 8]:
        d.append('r')

    pos_moves = []
    for i in d:
        pos_moves.append(generate(state, i, b))
    return pos_moves


def generate(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


# calling ID-DFS
initial = [1, 2, 3, 0, 4, 6, 7, 5, 8]
goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]

route = id_dfs(initial, goal, possible_moves)

if route:
    print("Success!! It is possible to solve 8 Puzzle problem")
    print("Path:", route)
else:
    print("Failed to find a solution")
```

Observation :



8/12/23 A-8/12/23

8 Puzzle problem using ID-DFS

Code:

```
def id_dfs(puzzle, goal, get_moves):
  import itertools

def dfs(route, depth):
    if depth == 0:
        return
    if route[-1] == goal:
        return route
    for move in get_moves(route[-1]):
        if move not in route:
            next_route = dfs(route+[move], depth-1)
        if next_route:
            return next_route

  for depth in itertools.count():
      route = dfs([puzzle], depth)
      if route:
          return route

def possible_moves(state):
    b = state.index(0)
    d = []
    if b not in [0,1,2]:
      d.append('u')
    if b not in [6,7,8]:
```

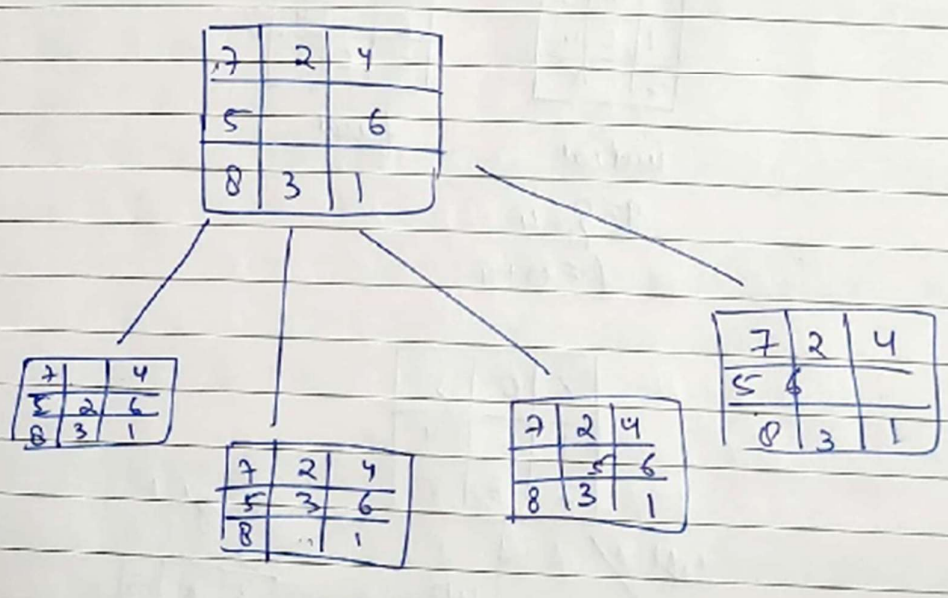
ID_DFS Combination of DFS and BFS
- DFS in BFS manner.

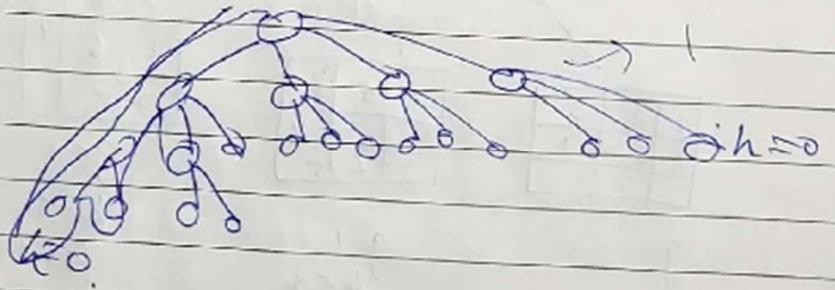
7	2	4
5		6
8	3	1

initial

	1	2
3	4	5
6	7	8.

Goal





Output:

```
Kanjika singh-1BM21CS086
Success!! It is possible to solve 8 Puzzle problem
Path: [[1, 2, 3, 0, 4, 6, 7, 5, 8], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
```

Program 4 : 8 Puzzle A* Search Algorithm

Code:

```
class Node:
    def __init__(self,data,level,fval):
        """ Initialize the node with the data, level of the node and the calculated fvalue """
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        """ Generate child nodes from the given node by moving the blank space
            either in the four directions {up,down,left,right} """
        x,y = self.find(self.data,'_')
        """ val_list contains position values for moving the blank space in either of
            the 4 directions [up,down,left,right] respectively. """
        val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data,x,y,i[0],i[1])
            if child is not None:
                child_node = Node(child,self.level+1,0)
                children.append(child_node)
        return children

    def shuffle(self,puz,x1,y1,x2,y2):
        """ Move the blank space in the given direction and if the position value are out
            of limits the return None """
        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
            temp_puz = []
            temp_puz = self.copy(puz)
            temp = temp_puz[x2][y2]
            temp_puz[x2][y2] = temp_puz[x1][y1]
            temp_puz[x1][y1] = temp
            return temp_puz
        else:
            return None


    def copy(self,root):
        """ Copy function to create a similar matrix of the given node"""
        temp = []
        for i in root:
            t = []
            for j in i:
                t.append(j)
```

```
            temp.append(t)
        return temp

    def find(self,puz,x):
        """ Specifically used to find the position of the blank space """
        for i in range(0,len(self.data)):
            for j in range(0,len(self.data)):
                if puz[i][j] == x:
                    return i,j


class Puzzle:
    def __init__(self,size):
        """ Initialize the puzzle size by the specified size,open and closed lists to empty """
        self.n = size
        self.open = []
        self.closed = []

    def accept(self):
        """ Accepts the puzzle from the user """
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz

    def f(self,start,goal):
        """ Heuristic Function to calculate hueristic value f(x) = h(x) + g(x) """
        return self.h(start.data,goal)+start.level

    def h(self,start,goal):
        """ Calculates the different between the given puzzles """
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp


    def process(self):
        """ Accept Start and Goal Puzzle state"""
        print("Enter the start state matrix \n")
        start = self.accept()
        print("Enter the goal state matrix \n")
        goal = self.accept()
```

```
        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        """ Put the start node in the open list"""
        self.open.append(start)
        print("\n\n")
        while True:
            cur = self.open[0]
            print("")
            print("  | ")
            print("  | ")
            print(" \\\'/ \n")
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")
            """ If the difference between current and goal node is 0 we have reached the goal
node"""
            if(self.h(cur.data,goal) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i,goal)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]

            """ sort the opne list based on f value """
            self.open.sort(key = lambda x:x.fval,reverse=False)


puz = Puzzle(3)
puz.process()
```

Observation :

Date : 8/12/2023

8 Puzzle Problem using A* Algorithm

Finds most cost effective path to reach the final state from initial state

$f(n) = g(n) + h(n)$

$g(n) \rightarrow$ depth of node

$h(n) \rightarrow$ no. of misplaced tiles

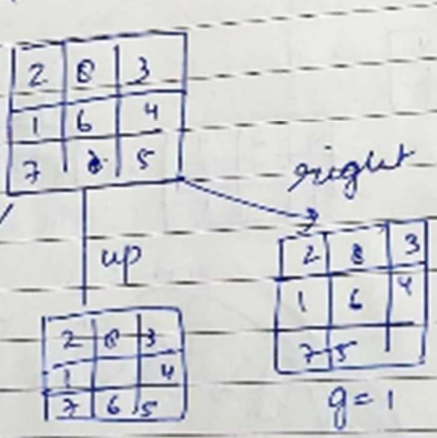
Suppose

initial

final

$g=0$ $h=4$

$f=0+4$



$g=1$ $h=5$ $f=1+5=6$

$g=1$ $h=3$ $f=1+3=4$

$g=1$ $h=5$ $f=1+5=6$

```
def f(self, start, goal):
    return self.h(start.data, goal) + start.level

def h(self, start, goal):
    temp = 0
    for i in range(0, self.n):
        for j in range(0, self.n):
            if start[i][j] != goal[i][j] and
                start[i][j] != '-':
                temp += 1
    return temp

def process(self):
    print("Enter the start state matrix \n")
    start = self.accept()
    print("Enter goal state \n")
    goal = self.accept()
    start = Node(start, 0, 0)
    start.fval = self.f(start, goal)
    self.open.append(start)
    print("\n")
    while True:
        cur = self.open[0]
        print("")
        print(" |")
        print(" | ")
        print(" \\\'/ \n")
        for i in cur.data:
            for j in i:
```

Date :

```
            print(j, end=" ")
        print("")
    if(self.h(cur.data, goal) ==0):
        break
for i in cur.generate_child():
    i.fval = self.f(i, goal)
    self.open.append(child)
    self.closed.append(cur)
del self.open[0]

self.open.sort(key = lambda x: x.fval,
                reverse = False)


puz = Puzzle(3)
puz.process()
```

Output

Enter start matrix

1 2 3
- 4 6
7 5 8

Enter the goal state

1 2 3
4 5 6
7 8 -

1
↓

1 2 3
- 4 6
7 5 8

Output:

```
Kanjika Singh- 1BM21CS086
Enter the start state matrix

1 2 3
4 5 6
8 7 _
Enter the goal state matrix

1 2 3
4 5 6
7 8 _
```

```
   |
   |
  \'/
...

1 2 3
4 5 6
8 _ 7

   |
   |
  \'/

1 2 3
_ 5 6
4 8 7

   |
   |
  \'/

1 2 3
4 5 6
8 _ 7
```

Program 5 : Vacuum Cleaner

Code:

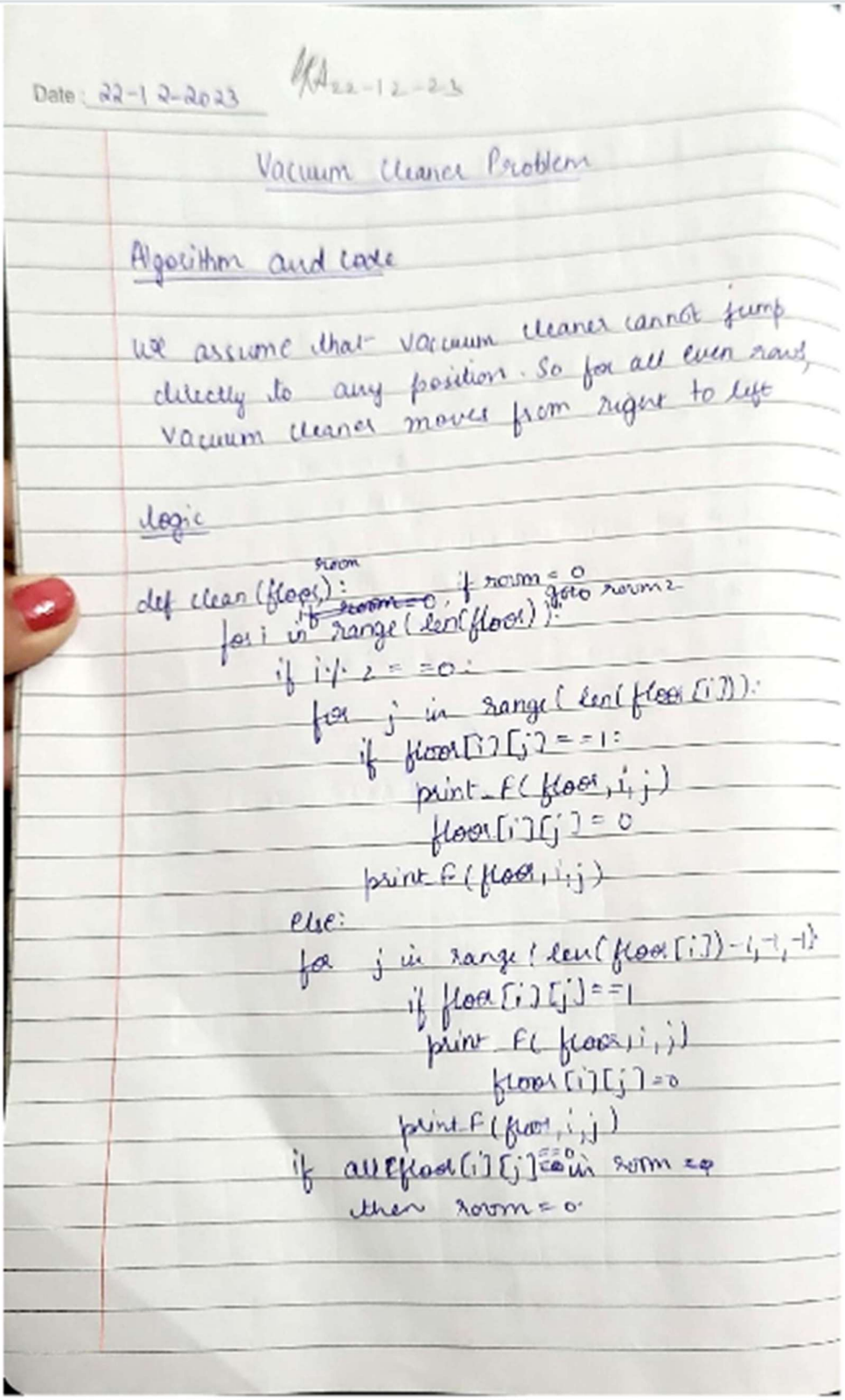
```
def clean_room(floor, room_row, room_col):
    if floor[room_row][room_col] == 1:
        print(f"Cleaning Room at ({room_row + 1}, {room_col + 1}) (Room was dirty)")
        floor[room_row][room_col] = 0
        print("Room is now clean.")
    else:
        print(f"Room at ({room_row + 1}, {room_col + 1}) is already clean.")

def main():
    rows = 2
    cols = 2
    floor = [[0, 0], [0, 0]]  # Initialize a 2x2 floor with clean rooms

    for i in range(rows):
        for j in range(cols):
            status = int(input(f"Enter clean status for Room at ({i + 1}, {j + 1}) (1 for dirty,
0 for clean): "))
            floor[i][j] = status

    for i in range(rows):
        for j in range(cols):
            clean_room(floor, i, j)

    print("Returning to Room at (1, 1) to check if it has become dirty again:")
    clean_room(floor, 0, 0)  # Checking Room at (1, 1) after cleaning all rooms

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

Four rooms:

```
def clean_room(room_name, is_dirty):
    if is_dirty:
        print(f"Cleaning {room_name} (Room was dirty)")
        print(f"{room_name} is now clean.")
        return 0  # Updated status after cleaning
    else:
        print(f"{room_name} is already clean.")
        return 0  # Status remains clean

def main():
    rooms = ["Room 1", "Room 2"]
    room_statuses = []
```

```
    for room in rooms:
        status = int(input(f"Enter clean status for {room} (1 for dirty, 0 for clean): "))
        room_statuses.append((room, status))
    print(room_statuses)

    for i, (room, status) in enumerate(room_statuses):
        room_statuses[i] = (room,clean_room(room, status)) # Update status after cleaning

    print(f"Returning to {rooms[0]} to check if it has become dirty again:")
    room_statuses[0]=status = (rooms[0],clean_room(rooms[0], room_statuses[0][1])) # Checking
Room 1 after cleaning all rooms

    print(f"{rooms[0]} is {'dirty' if room_statuses[0][1] else 'clean'} after checking.")

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

Observation:



Date: 22-12-2023

22-12-23

Vacuum Cleaner Problem

Algorithm and code

We assume that vacuum cleaner cannot jump directly to any position. So for all even rows, vacuum cleaner moves from right to left

logic

```
def clean(floor, room):
    if room = 0 goto room2
    for i in range(len(floor)):
        if i%2 == 0:
            for j in range(len(floor[i])):
                if floor[i][j] == 1:
                    print_f(floor, i, j)
                    floor[i][j] = 0
            print_f(floor, i, j)
        else:
            for j in range(len(floor[i])-1, -1, -1):
                if floor[i][j] == 1
                    print_f(floor, i, j)
                    floor[i][j] = 0
            print_f(floor, i, j)
    if all floor[i][j] == 0 in room =
        then room = 0.
```

Date:

```
def print_f(floor, row, col) :
    print("The floor matrix is belows")
        for r in range(len(floor)):
            for c in range(len(floor[r])):
                if r == row and c == col:
                    print(f" > {floor[r][c]} <", end=")
                else:
                    print(f" {floor[r][c]} ", end=' ')
                print

def main():
    floor = [ ]
    r = int(input("Enter number of room"),
    m = int(input("Enter rows"))
    p = int(input(" Enter no. of rows"))
    print("Enter clean status for each ~~cell~~ row)
    for i in range(m):
        f = list(map(int, input().split(" ")))
        floor.append(f)
    print()
    clean(floor, room)
```

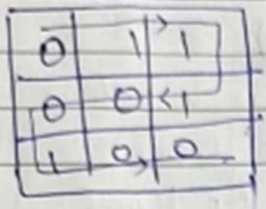
Two rooms

logic

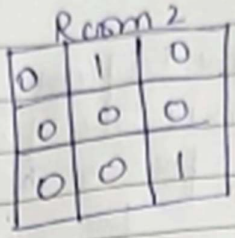
- Input for rooms 2 we take
- Initially, start from room 1 and inspect every grid
- If room is clean, {room1} = 0, goto room 2

Date :

Room 1



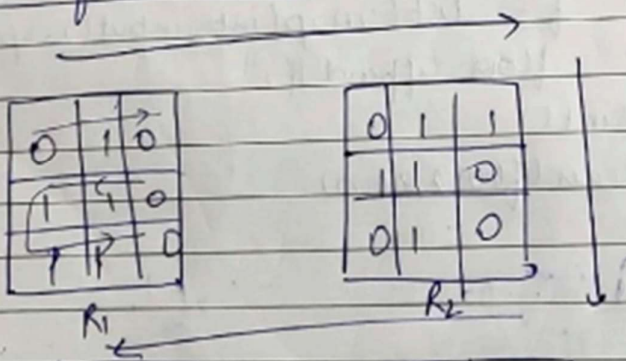
Room 2

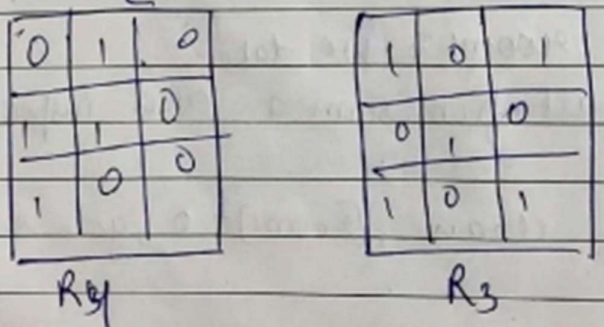


Room 1 → dirty → clean
check if room2 is completely clean (all grids)
if clean,
move to room2
If room2 → dirty
clean it by calling 'clean' function
If completely clean return and exit

Room1: status clean → move to Room2

for four rooms





Output:

```
Kanjika Singh-1BM21CS086
Enter clean status for Room 1 (1 for dirty, 0 for clean): 1
Enter clean status for Room 2 (1 for dirty, 0 for clean): 1
Cleaning Room 1 (Room was dirty)
Room 1 is now clean.
Cleaning Room 2 (Room was dirty)
Room 2 is now clean.
Returning to Room 1 to check if it has become dirty again:
Room 1 is already clean.
Room 1 is clean after checking.
```

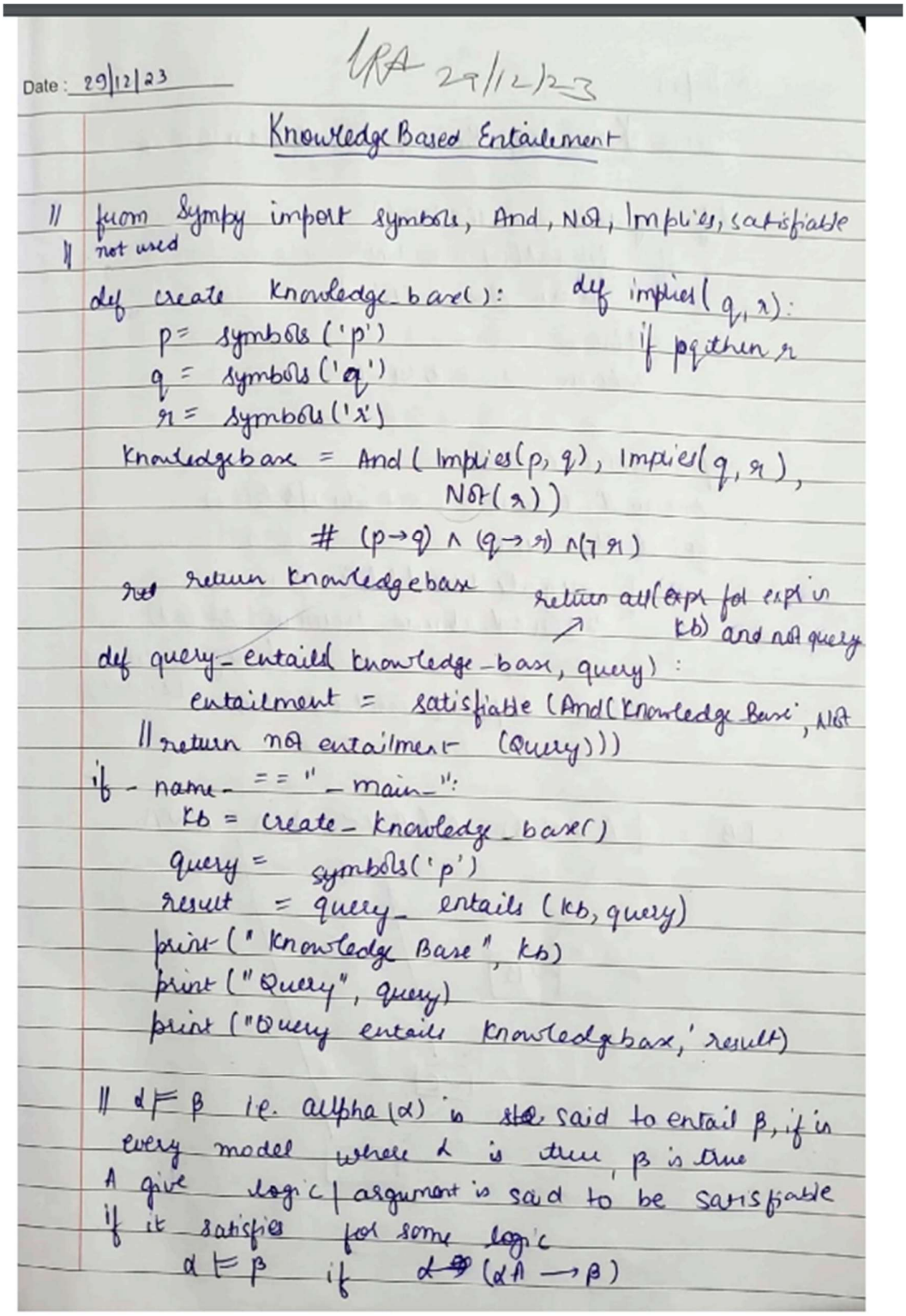
```
Kanjika Singh-1BM21CS086
Enter clean status for Room at (1, 1) (1 for dirty, 0 for clean): 1
Enter clean status for Room at (1, 2) (1 for dirty, 0 for clean): 0
Enter clean status for Room at (2, 1) (1 for dirty, 0 for clean): 1
Enter clean status for Room at (2, 2) (1 for dirty, 0 for clean): 0
Cleaning Room at (1, 1) (Room was dirty)
Room is now clean.
Room at (1, 2) is already clean.
Cleaning Room at (2, 1) (Room was dirty)
Room is now clean.
Room at (2, 2) is already clean.
Returning to Room at (1, 1) to check if it has become dirty again:
Room at (1, 1) is already clean.
```

Program 6 : Knowledge Base Entailment

Code:

```
from sympy import symbols, And, Not, Implies, satisfiable

def create_knowledge_base():
    # Define propositional symbols
    p = symbols('p')
    q = symbols('q')
    r = symbols('r')

    # Define knowledge base using logical statements
    knowledge_base = And(
        Implies(p, q),        # If p then q
        Implies(q, r),        # If q then r
        Not(r)                # Not r
    )

    return knowledge_base

def query_entails(knowledge_base, query):
    # Check if the knowledge base entails the query
    entailment = satisfiable(And(knowledge_base, Not(query)))

    # If there is no satisfying assignment, then the query is entailed
    return not entailment

if __name__ == "__main__":
    # Create the knowledge base
    kb = create_knowledge_base()

    # Define a query
    query = symbols('p')

    # Check if the query entails the knowledge base
    result = query_entails(kb, query)

    # Display the results
    print("Knowledge Base:", kb)
    print("Query:", query)
    print("Query entails Knowledge Base:", result)
```

Observation:



Date: 29/12/23

URA 29/12/23

Knowledge Based Entailement

```
// from Sympy import symbols, And, Not, Implies, satisfiable
// not used

def create knowledge_base():
    p = symbols('p')
    q = symbols('q')
    r = symbols('r')
    knowledgebase = And(Implies(p, q), Implies(q, r),
                        Not(r))
                  # (p→q) ∧ (q→r) ∧(¬r)
    return knowledgebase

def implies(q, r):
    if p q then r

    return all(expr for expr in kb) and not query

def query_entaild knowledge_base, query):
    entailment = satisfiable (And(Knowledge Base, Not
    // return not entailment (Query)))
if - name_ == "_main_":
    kb = create_knowledge_base()
    query = symbols('p')
    result = query_ entails (kb, query)
    print ("Knowledge Base", kb)
    print ("Query", query)
    print ("Query entails knowledgebase,' result)
```

// $\alpha \models \beta$ i.e. alpha (α) is said to entail β, if in every model where α is true, β is true

A give logic / argument is said to be satisfiable if it satisfies for some logic

$\alpha \models \beta$ if $\alpha \rightarrow (\alpha \wedge \rightarrow \beta)$

Output:

```
Kanjika Singh-1BM21CS086
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: p
Query entails Knowledge Base: False
```

Program 7 : Knowledge Base Resolution

Code:

```
def tell(kb, rule):
    kb.append(rule)

combinations = [(True, True, True), (True, True, False),
                (True, False, True), (True, False, False),
                (False, True, True), (False, True, False),
                (False, False, True), (False, False, False)]


def ask(kb, q):
    for c in combinations:
        s = all(rule(c) for rule in kb)
        f = q(c)
        print(s, f)
        if s != f and s != False:
            return 'Does not entail'
    return 'Entails'


kb = []

# Get user input for Rule 1
rule_str = input("Enter Rule 1 as a lambda function (e.g., lambda x: x[0] or x[1] and (x[0] and
x[1]): ")
r1 = eval(rule_str)
tell(kb, r1)

# Get user input for Query
query_str = input("Enter Query as a lambda function (e.g., lambda x: x[0] and x[1] and (x[0] or
x[1]): ")
q = eval(query_str)

# Ask KB Query
result = ask(kb, q)
print(result)
```

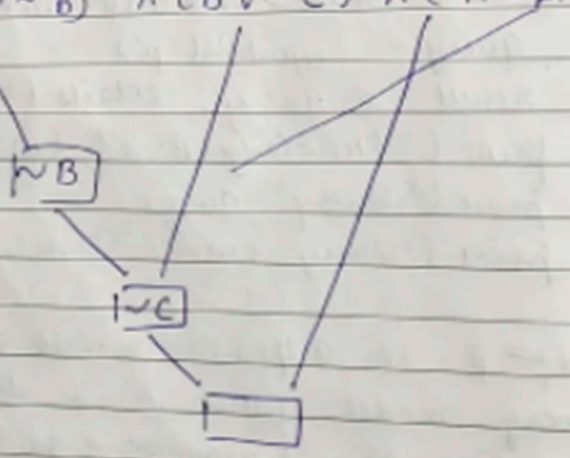
Observation:

Date : 29/12/23

Knowledge Based Resolution

```
def negate_literal(literal)
    if literal[0] == '~'
    return literal[1:]
    else
    return '~' + literal

def resolve(c1,c2)
    resolved_clause = set(c1)|set(c2)
    for literal in c1
        if negate_literal(literal) in c2:
        resolved_clause.remove(literal)
                                (negate_literal(literal))
    return tuple(resolved_clause)
```

KB : (A ∨ ~B) ∧ (B ∨ ~C) ∧ C ∧ ~A



Output:

```
Kanjika Singh-1BM21CS086

Step    |Clause |Derivation
------------------------------
 1.     | Rv~P  | Given.
 2.     | Rv~Q  | Given.
 3.     | ~RvP  | Given.
 4.     | ~RvQ  | Given.
 5.     | ~R    | Negated conclusion.
 6.     |       | Resolved Rv~P and ~RvP to Rv~R, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

```
Kanjika Singh-1BM21CS086

Step    |Clause |Derivation
------------------------------
 1.     | PvQ   | Given.
 2.     | ~PvR  | Given.
 3.     | ~QvR  | Given.
 4.     | ~R    | Negated conclusion.
 5.     | QvR   | Resolved from PvQ and ~PvR.
 6.     | PvR   | Resolved from PvQ and ~QvR.
 7.     | ~P    | Resolved from ~PvR and ~R.
 8.     | ~Q    | Resolved from ~QvR and ~R.
 9.     | Q     | Resolved from ~R and QvR.
 10.    | P     | Resolved from ~R and PvR.
 11.    | R     | Resolved from QvR and ~Q.
 12.    |       | Resolved R and ~R to Rv~R, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

```
Kanjika Singh-1BM21CS086

Step    |Clause |Derivation
------------------------------
 1.     | PvQ   | Given.
 2.     | PvR   | Given.
 3.     | ~PvR  | Given.
 4.     | RvS   | Given.
 5.     | Rv~Q  | Given.
 6.     | ~Sv~Q | Given.
 7.     | ~R    | Negated conclusion.
 8.     | QvR   | Resolved from PvQ and ~PvR.
 9.     | Pv~S  | Resolved from PvQ and ~Sv~Q.
 10.    | P     | Resolved from PvR and ~R.
 11.    | ~P    | Resolved from ~PvR and ~R.
 12.    | Rv~S  | Resolved from ~PvR and Pv~S.
 13.    | R     | Resolved from ~PvR and P.
 14.    | S     | Resolved from RvS and ~R.
 15.    | ~Q    | Resolved from Rv~Q and ~R.
 16.    | Q     | Resolved from ~R and QvR.
 17.    | ~S    | Resolved from ~R and Rv~S.
 18.    |       | Resolved ~R and R to ~RvR, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

Program 8 : Unification

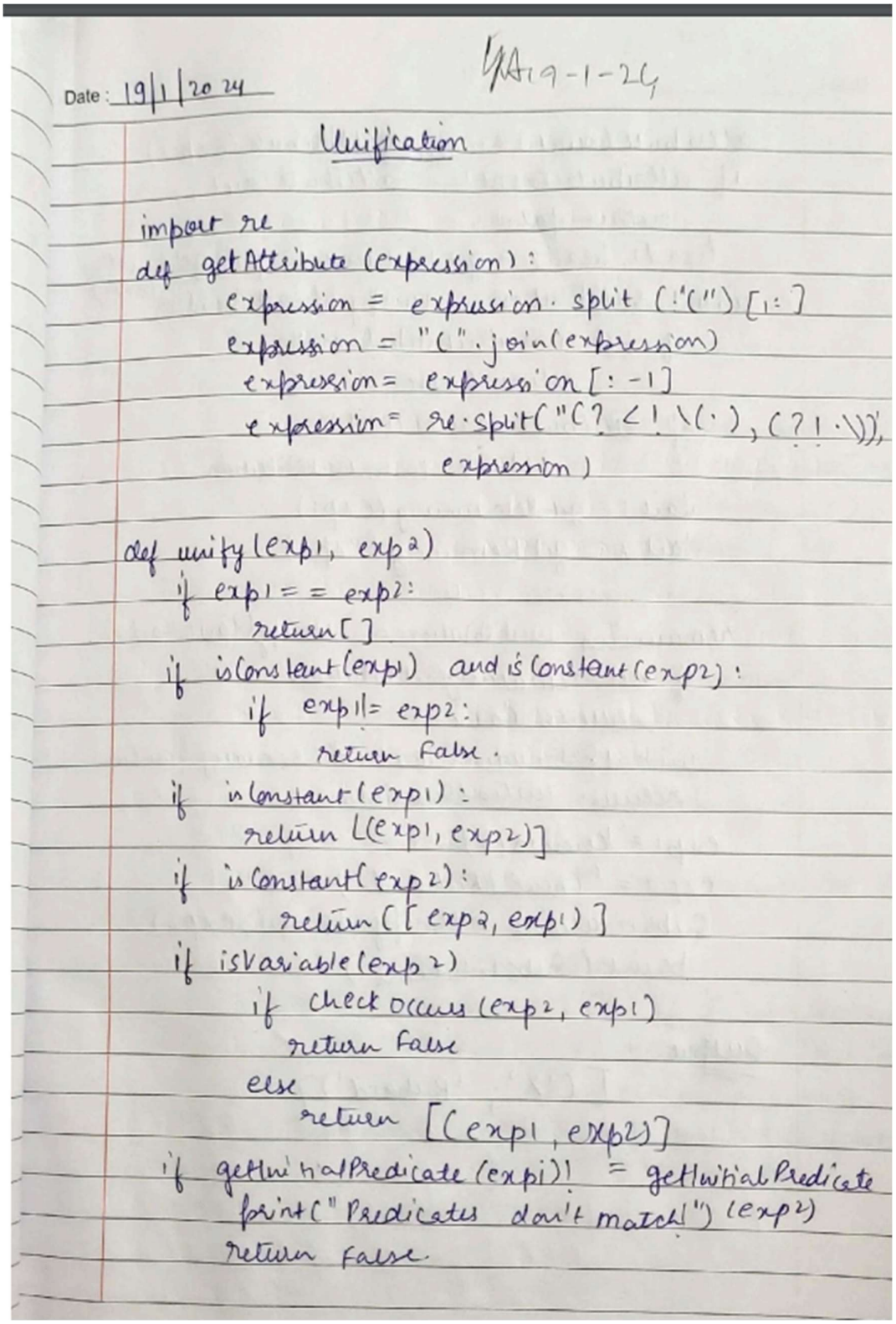
Code:

```
import re

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

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

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

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

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

def apply(exp, substitutions):
    for substitution in substitutions:
        new, old = substitution
        exp = replaceAttributes(exp, old, new)
    return exp
def checkOccurs(var, exp):
    if exp.find(var) == -1:
        return False
    return True


def getFirstPart(expression):
    attributes = getAttributes(expression)
    return attributes[0]


def getRemainingPart(expression):
```

```
    predicate = getInitialPredicate(expression)
    attributes = getAttributes(expression)
    newExpression = predicate + "(" + ",".join(attributes[1:]) + ")"
    return newExpression
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("Predicates do not match. Cannot be unified")
        return False

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

    head1 = getFirstPart(exp1)
    head2 = getFirstPart(exp2)
    initialSubstitution = unify(head1, head2)
    if not initialSubstitution:
        return False
    if attributeCount1 == 1:
        return initialSubstitution
```

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

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

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

    initialSubstitution.extend(remainingSubstitution)
    return initialSubstitution
exp1 = "knows(X)"
exp2 = "knows(Richard)"
substitutions = unify(exp1, exp2)
print("Kanjika Singh-1BM21CS086")
print("Substitutions:")
print(substitutions)
exp1 = "knows(A,x)"
exp2 = "knows(y,mother(y))"
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
```

Observation :



Date : 19/1/2024

19-1-24

Unification

```
import re
def getAttributes(expression):
    expression = expression.split("(")[1:]
    expression = "(".join(expression)
    expression = expression[:-1]
    expression = re.split("(?<!\(.),(?!.\))",
                          expression)

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(exp2)
        if checkOccurs(exp2, exp1)
            return False
        else
            return [(exp1, exp2)]
    if getInitialPredicate(expi)! = getInitialPredicate
        print("Predicates don't match!") (exp2)
        return False.
```

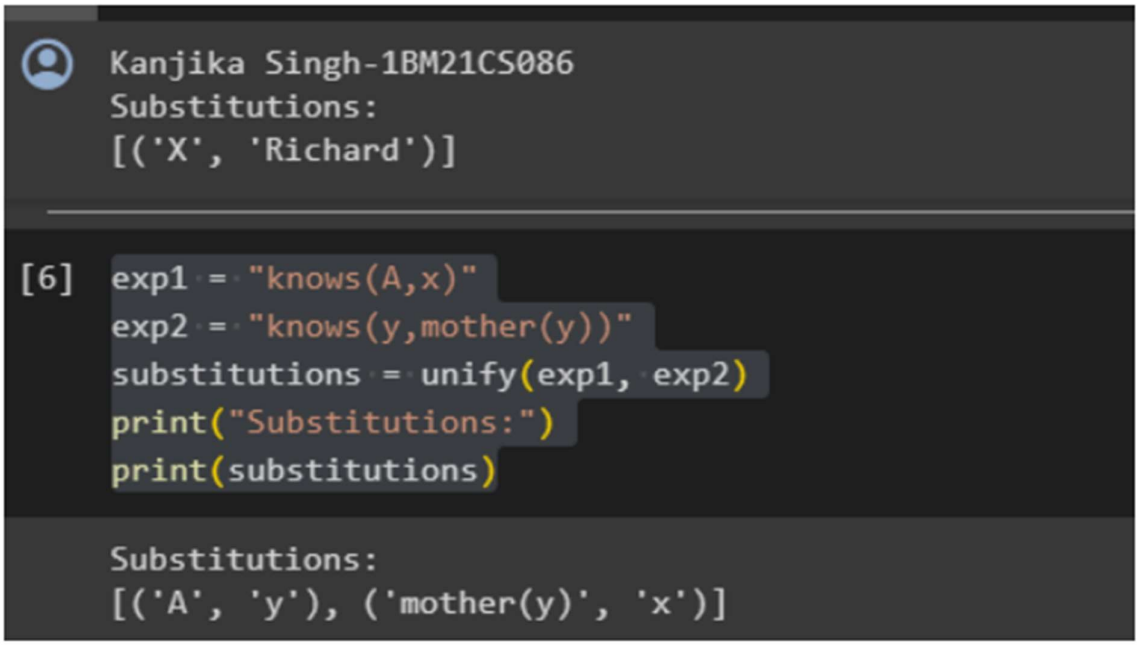
Date :

```
attribute count 1 = len(getAttribute(exp1))
if attribute count 1 = attribute count 2:
    return false
head1, head2 = get first Part (exp1), get first Part (exp2)
initial_substitution = unify (head1, head2)
    if not initialSubstitution:
        return false.
    if attribute count 1 == 1:
        return initialSubstitution
tail 1 = get Remaining (exp1)
tail 2 = get Remaining (exp2)

remaining substitution = unify (tail 1, tail 2)
    if not remainingSubstitution
        return false.
initial Substitution. extend (remaing Solution)
return initial Substitution
exp1 = 'Knows (x)'
exp2 = 'Knows (Richard")
Substitutions = unify (exp1, exp2)
print (Substitutions:)
```

Output

```
[('X', 'Richard')]
```

Output:


Kanjika Singh-1BM21CS086
Substitutions:
[('X', 'Richard')]
[6]
exp1 = "knows(A,x)"
exp2 = "knows(y,mother(y))"
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
Substitutions:
[('A', 'y'), ('mother(y)', 'x')]

Program 9 : FOL to CNF

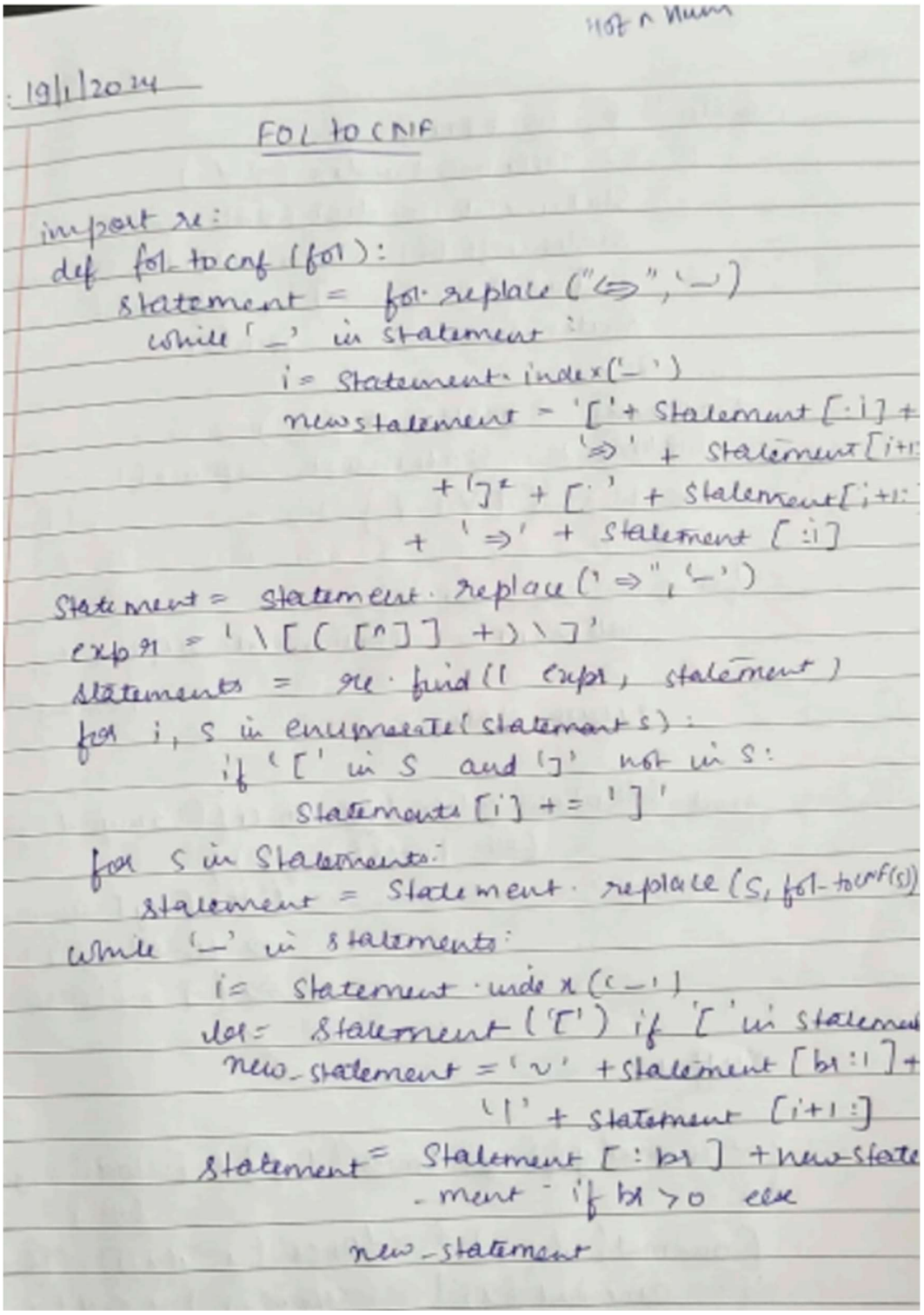
Code:

```
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~]+\([A-Za-z,]+\)'
    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
print("Kanjika Singh-1BM21CS086")
print(Skolemization(fol_to_cnf("animal(y)<=>loves(x,y)")))
print(Skolemization(fol_to_cnf("∀x[∀y[animal(y)=>loves(x,y)]]=>[∃z[loves(z,x)]]")))
print(fol_to_cnf("[american(x)&weapon(y)&sells(x,y,z)&hostile(z)]=>criminal(x)"))
```

Observation:



19/1/2024

FOL TO CNF

```
import re:
def fol_to_cnf (fol):
    statement = fol.replace("<=>", '_')
    while '_' in statement:
        i = statement.index('_')
        newstatement = '[' + statement[:i] +
                       '=>' + statement[i+1:
                     + ']' + '[' + statement[i+1:
                     + '=>' + statement[:i]
    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 statements:
        i = statement.index('_')
        brl = statement('[') if '[' in statemen
        new_statement = '~' + statement[br:i] +
                        '|' + statement[i+1:]
        statement = statement[:br] + new_state
                    ment if br > 0 else
                    new_statement
```

Date :

```
while '~∀' in statement:
    i = statement.index('~∀')
    statement = list(statement)
    statement[i], statement[i+1], statement
    [i+2] = '[', statement[i+2],
    statement = ''.join(s)

statement = list(statement)
statement = statement.replace('~[∃+', '[~∃')
expr = '([[∀|∃].?)'

for s in statements:
    statement = statement.replace(s,
                                  Demorgan(s))

return statement

print(Skolemization(fol_to_cnf('animal(y)
        ⇒ loves(x,y)'))
_____ : _____ ('∀x [∀y [animal(y)
                ⇒ loves(x,y)]]
                ⇒ (∃ z [loves(z,x)]]
```

Output:

[~animal(y) | ~~loves~~ loves (x,y)] & [~loves(x,y) | ani
(y)]

[animal(G(x)) & ~loves (x, G(x))] | [[loves(
[~ animal (x) | ~ weapon(y) | ~ sell [x,y,z)|
∨ hostile(z)]|
criminal(x)

Output:

```
Kanjika Singh-1BM21CS086
[~animal(y)|loves(x,y)]&[~loves(x,y)|animal(y)]
[animal(G(x))&~loves(x,G(x))]|[loves(F(x),x)]
[~american(x)|~weapon(y)|~sells(x,y,z)|~hostile(z)]|criminal(x)
```

Program 10 : Forward Reasoning

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}')
print("Kanjika Singh-1BM21CS086")
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()
print("Kanjika Singh-1BM21CS086")

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)')
```

Observation:

Date: 19/1/24

Forward Reasoning

```
def get Predictions(string):
    exps = '([a-z~]+)\([^&|]+\)'
    return re.findall(exps, string)

class Implication:
    def __init__(self, expression):
        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.get
                                          Vars()):
                        if v:
                            constant[v] = fact.getConstants
                                          ()[i]
                    new_lhs.append(fact)

class KB:
    def tell(self, e):
        if '=>' in e:
            self.implication.add(Implication(e))
```

Date :

```
def display(self):
    print("All facts:")
    for i, f in enumerate(set([f.expression
            for f in self.facts])):
        print(f'{i+1}. {f}')

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:

```
Kanjika Singh-1BM21CS086
Querying criminal(x):
        1. criminal(West)
All facts:
        1. hostile(Nono)
        2. weapon(M1)
        3. enemy(Nono,America)
        4. sells(West,M1,Nono)
        5. criminal(West)
        6. owns(Nono,M1)
        7. missile(M1)
        8. american(West)
```

```
Kanjika Singh-1BM21CS086
Querying evil(x):
        1. evil(John)
```