# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.



LAB REPORT on

# AI Lab Report

Submitted by

CHINMAYI (1BM21CS045)

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 "Internet of things lab" carried out by CHINMAYI (1BM21CS045), who is bonafide student of B. M. S. College of Engineering. It is in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2023. The Lab report has been approved as it satisfies the academic requirements in respect of a Artificial Intelligence lab - (22CS5PCAIN) work prescribed for the said degree.

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

`

## **Table of Contents**

Sl. No.	Title	Page No.
1.	Tic Tac Toe	1-7
2.	8 Puzzle Breadth First Search Algorithm	8-13
3.	8 Puzzle Iterative Deepening Search Algorithm	14-18
4.	8 Puzzle A* Search Algorithm	19-26
5.	Vacuum Cleaner	27-31
6.	Knowledge Base Entailment	32-34
7.	Knowledge Base Resolution	35-37
8.	Simulated Annealing	38-42
9.	Unification	43-50
10.	FOL to CNF	51-55
11.	Forward reasoning	52-61

### **Program 1: Tic Tac Toe**

Code:

```
tic=[]
import random
def board(tic):
  for i in range(0,9,3):
    print("+"+"-"*29+"+")
    print("|"+" "*9+"|"+" "*9+"|"+" "*9+"|")
    print("|"+" "*3,tic[0+i]," "*3+"|"+" "*3,tic[1+i]," "*3+"|"+"
"*3,tic[2+i]," "*3+"|")
    print("|"+" "*9+"|"+" "*9+"|"+" "*9+"|")
  print("+"+"-"*29+"+")
def update_comp():
  global tic,num
  for i in range(9):
    if tic[i]==i+1:
       num=i+1
       tic[num-1]='X'
       if winner(num-1)==False:
         #reverse the change
         tic[num-1]=num
       else:
         return
  for i in range(9):
    if tic[i]==i+1:
       num=i+1
       tic[num-1]='O'
       if winner(num-1)==True:
         tic[num-1]='X'
         return
       else:
         tic[num-1]=num
  num=random.randint(1,9)
  while num not in tic:
    num=random.randint(1,9)
  else:
```

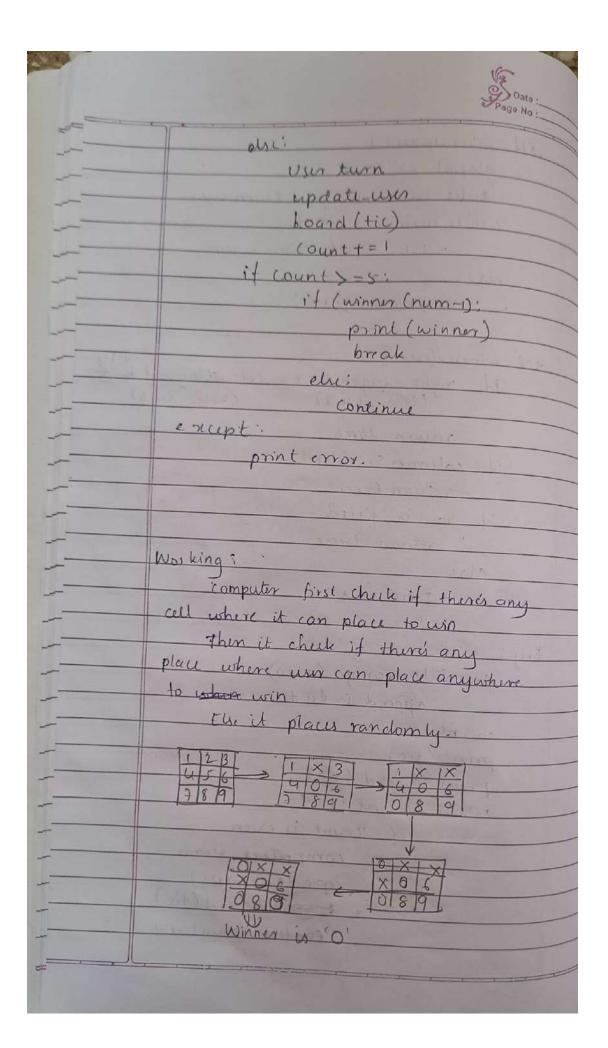
```
tic[num-1]='X'
```

```
def update user():
  global tic,num
  num=int(input("enter a number on the board :"))
  while num not in tic:
     num=int(input("enter a number on the board :"))
  else:
     tic[num-1]='O'
def winner(num):
  if tic[0]==tic[4] and tic[4]==tic[8] or tic[2]==tic[4] and tic[4]==tic[6]:
     return True
  if tic[num]==tic[num-3] and tic[num-3]==tic[num-6]:
     return True
  if tic[num//3*3] = tic[num//3*3+1] and
tic[num//3*3+1] == tic[num//3*3+2]:
     return True
  return False
try:
  for i in range(1,10):
     tic.append(i)
  count=0
  #print(tic)
  board(tic)
  while count!=9:
     if count%2==0:
       print("computer's turn :")
       update comp()
       board(tic)
       count+=1
     else:
       print("Your turn :")
       update user()
       board(tic)
       count+=1
```

```
if count>=5:
    if winner(num-1):
        print("winner is ",tic[num-1])
        break
    else:
        continue
except:
    print("\nerror\n")
```

```
78/12/23
     -lic-[]
     import random
         bound (tre)
                     tic [num-1] < 'X'
                     if mic[n-i] rotauinner
                             tic(hum-1) e'x'
             nume random. randin(1,9)
              fic Chum-1] etx
```

	Date :
14	usurupdu()
	global tic, num
	take num as input from user
L	while neum not in tic:
	· · · · · · · · · · · · · · · · · · ·
e	Lici Land
	Hicknum-1] = 10'
aled u	inner(num):
i-	right diagonal or left idiagon ( fill (2, u) (u,6)
	return toral
i.f	column is filled
	return true
ĵ.	row is filled
	return tirue
	else
	return false:
	and at analy man his arrange is a
try:	Then it there is there's are
	or i in range (1,10)
	append i to tic
	unt = 0
	int(tic)
p	rint (board (tic))
- 14	thile count! = 9
	if count is even
	computers turn
	upade-comp()
	boren board (tic)
	contint count ++



			i
1	2		-
4	5	6	
7	8	9	-
outer's	turn :		
1	2	3	1
4	5	6	
Х	8	9	-
r turn	nber on the	board :	-+ 2
			-+
1	0	3	1
4	5	6	1
x	8	9	1
outer's	turn :		
18	o		
4	5	X	
х	8	9	1
turn :	nber on the	board :	-+ 5
1	0	3	-
4	0	Х	1
х	8	9	-

computer's	turn :	
+		
1	0	3
4	0	X
х	Х	9
Your turn : enter a num	ber on the	board :9
1	0	
4	0	
Х	х	
computer's	turn :	
x		3
4	D	
x	х	0
Your turn : enter a num	ber on the	board :4
х	0	3
C	0	
x	Х	0
computer's	turn :	
x	0.	×

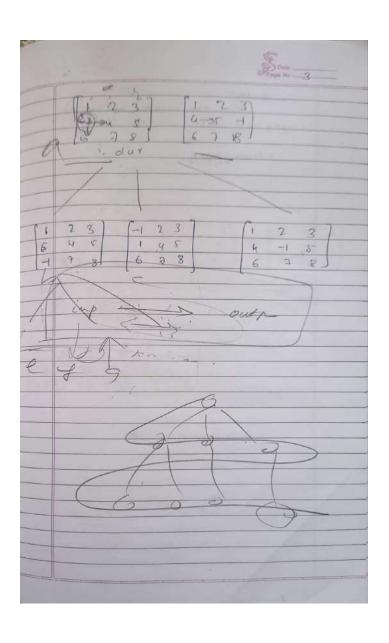
### Program 2:8 Puzzle Breadth First Search Algorithm

```
Code:
def bfs(src,target):
  queue=[]
  queue.append(src)
  exp=[]
  while len(queue)>0:
    source=queue.pop(0)
    #print("queue",queue)
    exp.append(source)
    print(source[0],",source[1],",source[2])
    print(source[3],'|',source[4],'|',source[5])
    print(source[6],'|',source[7],'|',source[8])
    print("----")
    if source==target:
       print("Success")
       return
    poss moves to do=[]
    poss moves to do=possible moves(source,exp)
    #print("possible moves",poss moves to do)
    for move in poss moves to do:
       if move not in exp and move not in queue:
        #print("move",move)
        queue.append(move)
def possible moves(state, visited states):
  b=state.index(0)
  #direction array
  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('l')
```

```
if b not in [2,5,8]:
    d.append('r')
  pos_moves_it_can=[]
  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 temp
src=[1,2,3,4,5,6,0,7,8]
target=[1,2,3,4,5,6,7,8,0]
bfs(src,target)
```

110	123 - M-25-11-2 Frace No. 5
Q	Solve 8 puzzle problem using bfs
	Algorithm
	Car beign to
	11 bfs function
	ble fore beautiful
	initialize que as collection of array by
	appending src to queue[]
	explored states.
	while queue is not empty
	soura ← quiu. poplo)
	cap. append (source) //stores emplored combin
	Ellation Albania State
	print = soura
	with sources of the new traverse out the
	if source is equal to target
	print source
	return (and shall man
	great of state uges
	pos-moves to do < possible movey source, exp
	The world the second
-10	for move in pos-moverta do
	if move is not in exp and queue
	quiu appind (move)
	0 00 00 11
15-	possible moves (state, vat visited-states)
	be index of empty cell
	initialize empty array of direction
	if to be not in the top row
-	d-append ('U')

	Able Ser L. Page No: 2
	if b not in bottom row
	d-append (d)
	if b not in first column
_	d. append (U)
	if b not in last column
	d. append (Y)
4	d[] will give possible direction's to slide
	numbers sold 17-
	for every direction in d
	possible moves it can appeal nd
	append state generated to
	possible_moverit=can-do[]
	BYACK - SHATE
	return moves it-can-do in possible-moves
The second	which is not already visited
j	gen (state, m, b)
	copy state to temp
	The state of the s
	if m is 'd'
	push move the number do below
	the empty cell up
	to the barry many of the
	it m is 'a'
	move the number above empty cell d
	if m is e
	move the number right to all des
	13 ) kon over Code



	S Data : Page Ma . L
-	if m = = r
	more the number mig left to
	empty cell left
-	2 (12)
	return temp

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

13

### **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 = \text{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")
```

Toploment iterative dispersing search algorithm  imital    2 3	21/2/2023	RA21-12-25 Date :
1 2 3	Implement	iterative deepening search
2 3	ange.	ricial Goal
- 4 6   2 3   - 2 3   4 6   4 6   7 5 8   7 5		1 2 2
1 2 3		1 1
1 2 3		7 5 8 7 7 8 -
1 2 3		
1-3 [123 [123] b26 46- 456 758 7-8		and tractical allowers his
1-3 [123 [123] b26 46- 456 758 7-8		to a mule
1-3 [123] [123] 126   46-   456 258   7-8	1 1 2 3	[ 2 3
1-3 [123 [123] 1-26   46-   456 1-58   7-58   7-8	4 4 6	7 4 6 4 6
1-3 [123] 126   46-   456 258   258   2-8	758	
1 - 3	(i proph	
1 - 3		one fratte
1 - 3		wither the part of the later of
1 - 3	7 1 X	
426 46- 456 758) 758) 7-8)	Tr. Chameir ages	Logist mi graphs sich
(a) was don't at her ind it	1 - 3 1 2 3	1.23
(258) (258) (25-8)  (a) which the set of the		456
(a) perhander of the company of the		(7-8)
(a) perhander of the company of the	L	
(c. 10) were dest of done ind it  bect is tota  (c. 10) vice moved of day is it it  bed his bec		del politice many trades
(6.10) were suit of sure is d. if  but in the control of sure is d. if  but by the bree  (8.50) for the fill for is d. if		b c - date index(a)
bet'n bbs  (bea) who mived at you is it it  but 'b' bbs  (see ) for this bill for it is it.	Land with paying	de F F T Entropent
bet'n his (66,0) who mived at far & d }! but 'h' his (87,0) for the Aid fil		
(65,07) who would night to d }!	(-10) Us	e duty tot door hild him
(J. F. o. ) for old []		bet i bbo
(J. F. O.) Var. Mil. Fil. Var. N. of Mil.	(65.2) also	moved at tox is it is bottom
Long Walls of the All All Land of the Land		hat 'b' bto
la nt 12 dans	(28.0)	on also not be a sell in
		ont I' long

gorithm	
id offs (puzzle, goal, get import intertools def dfs (route, depth) if depth is equal return  if route outs e  return route of the move in get m	to 0  qual to goal  ite.  roves (route[-i]):
if move not in nxt route	route:  dfs(route+ (movi)  depth-1)
if (nat-roo	urn nat-route
for depth in inter route ← dfs ( if route; return n	(puzzle], depth)
h ← state. index(0)	empty direction)
if b is not in first a  add 'u' to d  if b is not in bottom  add 'd' to d  if b is not in test co  add 's' to c	now (6,78)
	if dipth is equal return  if route of se e  return rou  for move in get m  if move not in  nut-route  if (nut-rou  return m  for depth in inter  route = dfs (  if route;  return m  possible-moves (state):  b = state index(o)  d = [] (intialized  if b is not in first made 'a' to d  if b is not in bottom  add 'd' to d  if b is not in teft co

```
if b not in right column (2,5,8)
     add '?' to d
    pos move = []
    for i in d
      pos moves append (generale (state, i, b))
    retion pos-moves
 det generale (state, m, b):
     temp = state.copy()
      if m is equal to d'
         swap (temp[b+3] and temp[b])
         m is equal to 'u'
         sup temp(b-3) and temp[b]
      if m is equal to'l'
         swap temp[b-1] & temp[b]
      if m is equal to r'
        swap temp (b+1) and temp b
    retur temp
initial = [1,2,3,0,4,6,7,5,8]
goal = [1,2,3, 4,5,6,7,8,0]
route @ (d-dfs(initial, goal, possible-moves)
if route
     print ("success")
elsc
```

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

return None

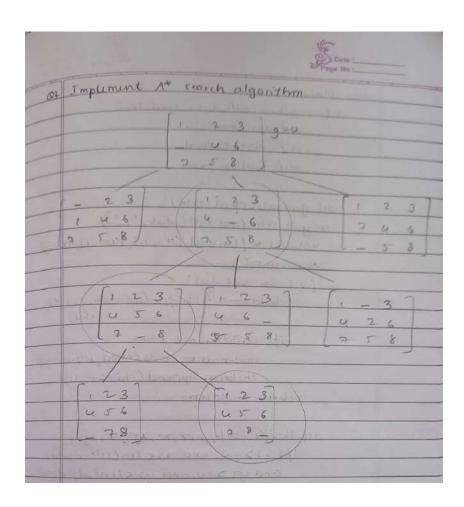
```
Code:
class Node:
  def init (self,data,level,fval):
     """ Initialize the node with the data, level of the node and the calculated fvalue """
     self.data = data
     self.level = level
     self.fval = fval
  def generate child(self):
    """ Generate child nodes from the given node by moving 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 \ge 0 and x2 < len(self.data) and y2 \ge 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:
```

```
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()



	Page No:
, ,	Class Node
	del init (self, data, level, [val):
_	self.dala e data
	colf. level - level
	self-fual = fual
	def generate-chid (self):
	n; y = self.find (self.dat, '_')
	val-list e [[x,y-1], [x,y+1], [x-1,y], [x+1,
	children a[]
	for i in val-list;
	child - self sahuffle (self data, x,y, it)
	if child is not None
	chid-node ← Node (child, self-leve)
	children. append (child=node)
	return children
	def shuffle (self, puz, x1, xy1, x2, y2).
	if Exz>=0 and xx len (self-data
	and yr > =0 and yr < len (self. da
	temp-puze[]
	pr temp-puze self-copy(puz)
	temp = temp-puz (712) (y2)
	temp-
	swap temp-puz(x2,y2) and
	temp-pur (n, y)
	return temp puz
	else:
	neturn none

	S Deta :
7	A STATE OF THE PARTY OF THE PAR
1	class Purcle
1	def init- (self size)
1	selfine size
	self-open < ()
	self-closed = []
	Court below - June
	def accept (self)
	puz =[]
	for (i in range(0, self:n):
	tempe input () (plit (" ")
	puz.append (temp)
	return pur
	(an) rang
	def h(self, stort, goal)
	tempeo
	for 1 in range (0, self. n)
ļ	for j in range (0, xeldin)
	if startfillj! =goalfill & startli
ļ	!='-!
	temp < temp+1
	return temp
ļ	Janua L
	def f(self, start, goal)
	tempt
	return self. h (start. data, goal) +start le
	( mestang a should able )
	foliana Alexa Ish
	about a med to return po [12]
	Carlo National Control of the Contro

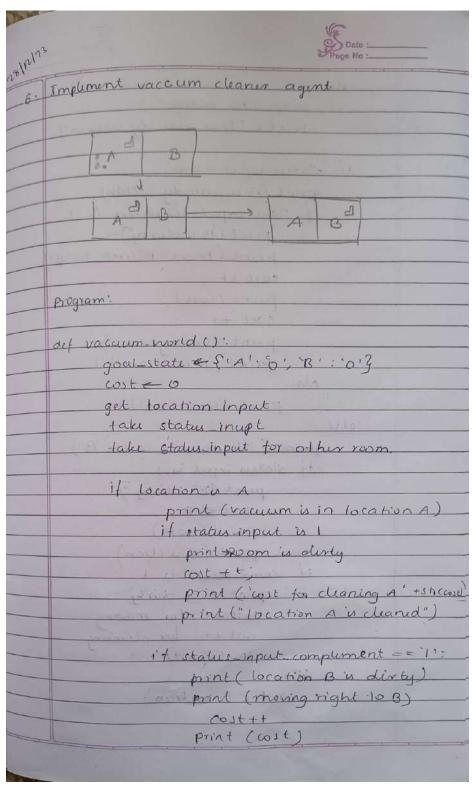
	Page No:
, ,	def process (self)
	print ("Enter start state "Tr")
	start = self-accepts)
	of print (" Enter goal state " ")
	goal = self.accept()
	(1- bidily
	start = Node (start, 0,0)
	Start (val = self. ((start, goal)
	(1 - aug - u )
	self. open. append (start)
	3 20 more of an expension
	while True:
	cur & self open [o]
	print ("")
	print (1')
	print ("\\'/\n")
	for i in curedata
	for y in 1
II dyesta l	print(), end = " )
1-12	print(" ")
	Hamilton Amilton
	if Gulf. h (curr. data, goal)==0)
	brigk
	for i in cur.genochild()
	i fual = self.f(i,goal)
	self.open.append()
	self-closed appen (cum)
	del selfiepen[9]
	self-open-vortley=lambda
	n: x. fval)

```
Enter the start state matrix
1 2 3
4 5 6
_ 7 8
Enter the goal state matrix
1 2 3
4 5 6
78_
\'\
|
|
1 2 3
4 5 6
_ 7 8
 ٧٠,
1 2 3
4 5 6
7 _ 8
 ۱,۱
1 2 3
4 5 6
78_
```

#### **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()
```



Soata Page No :
alse
print ("no action")
print ("B is already cleaned")
if status-input = = '01
print (n is already clian)
if status in put complement is 1
print (B is dista)
print (more RIGHT to BI
print (cost)
(USE ++
print (sucking cost)
print ('B is cleaned')
else (Bis cleaned)
print ('B is clean')
else many
print (" Vaccium is in las O:)
if status input is 1
print (B'is disty)
Cost ++ 3
print (cost)
print (Bisclean)
if complement is 1
print A is dirry
lost ++ for moving
cost ++ for cleaning
print (cost)
alse
printloc B ischan)
 The last of the la

```
if status input complement - - 1"
            print (loca is didy)
             increase cost for moving sept
             print (cost)
         print (A in decen)
    print (goal-state)
    print (cost)
vaccicim-world ()
example output:
o indicates clean and I indicates dirty
Enter location of vacum A
Enter status of A 1
Enter status of other room o
vacuum is placed in Location A
location A is dirly
Cost of cleaning A 1
Location A has been cleaned
No action 1
location B is already clean
GOAL state
  {x1:00 B1:014
Reformance Measure 1.
```

vacuum\_world()

O indicates clean and 1 indicates dirty
Enter Location of VacuumA
Enter status of A1
Enter status of other room0
Vacuum is placed in Location A
Location A is Dirty.
Cost for CLEANING A 1
Location A has been Cleaned.
No action1
Location B is already clean.
GOAL STATE:
{'A': '0', 'B': '0'}
Performance Measurement: 1

### 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)
```

11100	Some Maria
.00	logic and show that the given quary entally the knowledge base or not.
	Entailment means
	if and only if it is true on all worlds where I'B is true
	rg:
	$p \rightarrow q$
	quiny P
	Code:
	from sympy import symbols, And, Not, implies
	p = symbol ('p')
	q <- symbol ('q')  r <- symbol ('r')
	Traplies (p,q), #P->9
	Implies $(q_1r)$ , # $q \rightarrow r$ $1/(ot(r))$ # $\sim r$
	return knowledge base

	S Date :
det	query entails (knowledge base, query): entailment e- satisfiable (And Knowledge base,
_	Not (quing))
	return not entailment
	name = "main = ";
	kb = create knowledge base ()
	rult = query contails (16, query)
	print (kb)
	print ("Overy entails knowledge base", result
Outp	owledge Base: ~ & (Implies (p, a)) & Templies (q, s)
Qu	64,0
00	eny entails (chowledge base: False.
	mint throught against

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) \text{ for rule in kb})
     f = q(c)
     print(s, f)
     if s != f and s != F alse:
        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)
```

"///"	
	and prove the given query wing resolution
	Resolution combines the clauses containing complementary literals and generate a new clause
	Codi:
	det tell (ub, rule) kb-append (rule)
	Combinations = Enith table containing  3 * variables as array of type
	det eisk (kb,q):  for cin combinations
	f =q(c)  print(s,f)
	if & s! = f and s! = False return 'Doesn't entail'
	ch + [] His initialized
	rule str = input ("Enter rule as "sombda x"
	71 = Eval (rule str.)
-	full (kh, ri)

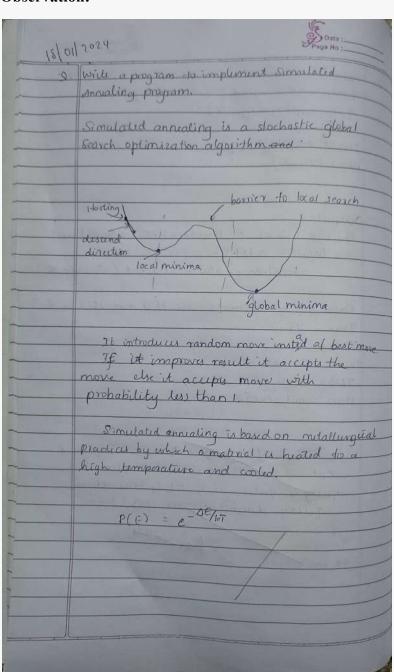
```
24214
 q = eval (quay str)
result e-ask (kp, g)
point (result)
Enter o rule as on lambda x . vio) orfoxily and
 ( To To and x [1])
        lambda 2: 260 or x(1) and (260) and x(1)
Enter query: dambola n: x[3] and x[1] or x[2]
True True
True True
      False
Does not entail
logic : thick if kb and query are equal it
   not if kb is true this implies
  query is false when 106 is true thus
    query doesn't entail 10B
```

## **Program 8. Simulated Annealing**

Code: import random import math class Solution: def \_\_init\_\_(self, CVRMSE, configuration): self.CVRMSE = CVRMSEself.config = configuration # Function prototype def gen\_rand\_sol(): a = [1, 2, 3, 4, 5]return Solution(-1.0, a) # global variables T = 1Tmin = 0.0001alpha = 0.9 $num_iterations = 100$ M = 5N = 5source array = [['X' for in range(N)] for in range(M)]temp = [] mini = Solution(float('inf'), temp) current\_sol = gen\_rand\_sol() def neighbor(current sol): return current\_sol

```
def cost(input configuration):
  return -1.0
# Mapping from [0, M*N] --> [0, M]x[0, N]
defindex to points(index):
  return [index % M, index // M]
# Returns minimum value based on optimization
while T > Tmin:
  for in range(num iterations):
    # Reassigns global minimum accordingly
    if current sol.CVRMSE < mini.CVRMSE:
       mini = current sol
    new sol = neighbor(current sol)
    ap = math.exp((current_sol.CVRMSE - new_sol.CVRMSE) / T)
    if ap > random.random():
       current sol = new sol
  T *= alpha # Decreases T, cooling phase
print(mini.CVRMSE, "\n")
for i in range(M):
  for j in range(N):
    source array[i][j] = 'X'
# Displays
for obj in mini.config:
  coord = index_to_points(obj)
  source_array[coord[0]][coord[1]] = '-'
```

```
# Displays optimal location
for i in range(M):
   row = ""
   for j in range(N):
      row += source_array[i][j] + " "
   print(row)
```



	Date :
import random	
import math	
class Solution:	
def init _ Cself cv	RMSE, condiquential
sold. CURMSE &	-CVRMSE
self-config e	
	The state of the s
del gen-rand sol():	
a=[1, 2,3,4,5]	
return Solution (-1.	() (a)
and the same of the state of	
T=1	paralla se e
Inin = 0.0001	
alpha = 0. q	LV 2 Several Editions
rum_iterations = 100	
M=5 : (M)	Sept 1 A
N=5 Trans	
source-away = [[X' for	- in range (N)7
for -in	range(M)]
temp = []	
mini = Totadian (float	('in/'), temp)
current so 1 = gen rand-	
9	
led neighbour (current	:(102
return current	
> (A horoatta)	
if cost (input work	auration)
return -1.0	y ar arating.
7 -170	
	ALL ALLEGE

	Program No.
ael ind	is to points (India):
78	turn (index open, index//m)
while T	> Tmin
for.	in range Cour iterations):
	of current sol CURMIE (mini CURMIE)
	TELEVIA - CALLEGIAND ASS.
1	new sol neighbour (current sol)
	ap = motherp((current solcvRMSE -
	f ap > random random ().
	current sol = new_sol
- T *	culpha
	A Property of the Control of the Con
print (n	rini. CVRMSE, " \h")
- for i in	range(M);
for	Jin range (N):
	Source array [i] [j]='x'
for obj	in mini config:
Cus	ord = indix-to-points (obj)
200	ru cmay [coord[o]][coord[i]]
for i in	range(M)
	in range (N)
	row + = source orray[i](j]+"
	t
	The second secon

### **Program 9: 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 is Variable(exp1):
     if checkOccurs(exp1, exp2):
       return False
     else:
       return [(exp2, exp1)]
  if is Variable(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 remaining Substitution:
     return False
  initialSubstitution.extend(remainingSubstitution)
  return initialSubstitution
exp1 = "knows(X)"
exp2 = "knows(Richard)"
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
exp1 = "knows(A,x)"
exp2 = "knows(y,mother(y))"
```

```
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
```

Unification is the process of making 2 different logical atomic expressions ident by finding a substitution.  It takes two literals as input and make indentical using substitution.  Unify algorithm is used for writication take 2 atomic sentences and returns a unification is a key component of all first order inference algo  P(ry)  P(a, f(z))  Talx f(z) ly  import re  def getAttributes (expression):  expression = expression, split ("c") [1:]  expression = expression [: -1]	1/2/24	Sonta -
different logical atomic expressions identify finding a substitution.  It takes two literals as input and make indentical using substitution.  Unify algorithm is used for unification take 2 atomic sendences and returns a unifier for those sendences.  Unification is a key component of all first order inference algo  P(ny)  P(a, f(z))  =) Fa/x f(z)/y  import re  olet getAltributus (expression):  expression = expression, split ("c") [1:]  expression = "(" join (expression)  expression = expression [: -1]  expression = expression [: -1]  expression = expression [: -1]  expression = expression [: -1]		
Unify algorithm is used for unification take 2 atomic sordences and returns a unifer for 1 hose sentences.  Unification is a key component of all first order inference algo  P(1,14)  P(a, f(z))  -) [a/x f(z)/y]  import re  olet getAttributes (expression):  expression = expression, split ("(") [1:]  expression = "(" join (expression)  expression = expression [:-]  expression = expression [:-]  expression = expression ("(") (") (") (")  expression = expression (")		different logical atomic expressions ident
import re  olet getAttributus (expression):  expression = "(" join (expression)  expression = expression [: -1]  expression = expression [: -1]  expression = resplit ("(") ( ! ( ! ( ! ( ! ) ) ) )  expression = resplit ("(") ! ( ! ( ! ( ! ) ) ) )  expression = resplit ("(") ! ( ! ( ! ) ) ( ! ( ! ) ) )		It takes two literals as input and make
import re  olet getAltributus (expression):  expression = "(" join (expression)  expression = expression [: -1]  expression = expression [: -1]  expression = resplit ("(") ( ! ( ! ( ! ( ! ) ) ) ( ! ( ! ( ! ) ) ) ( ! ( !		take 2 atomic sentences and returns a unific
import re  olet getAltributus (expression):  expression = expression, split ("(")[1:]  expression = "(" .join (expression)  expression = expression [:-1]  expression = expression [:-1]  expression = expression ("(?!<)) (?!)		Unification is a key component of all first order inference algo
expression = expression ("") [1:]  expression = "(" join (expression)  expression = expression [:-1]  expression = expression ("() () () () () ()  expression = expression ()		
expression = expression, split ("(") [1:]  expression = "(" .join (expression)  expression = expression [:-1]  expression = expression [:-1]  expression = expression ("(?) < (.) (?) (.)  expression		
expression = "(" join (expression)  expression = expression [:-1]  expression = resplit ("(?!< (.) (9!.))  expression)	4	
expression eresplit ("(?!<(.) (9!.))		expression = "(" join (expression)
		expression eresplit ("(?! <c.), (?!.)<="" td=""></c.),>

	Date :
del	get Initial Predicate (express);
	return exposer explit ("C") FO)
def	(clorstant (chax):
	return (har is supper() and len(char) ==
def	is Variable (char):
	return true if its lower and length is 1
	the sample of
det	replace Athibutes (enp, old, new).
	attributes e get Attributes (exp)
	for index val in enumente (actributes):
	if val equal to old
	attributer [index] < new
	predicate eget. Initial Predicate (exp)
	roturn predicate + "("+", ", join (attributes
	Congress topical some calledor
de f	apply (exp, substitutions):
	for substitution in substitutions;
	new, old e-situation
189	eap e-replace Attributes (enp, old, new)
	ntuin eap
def	chech Oceurs (var, exp);
	whick if var is present in a
	The Folge State of the State of
def	getFirstPart (expression)
	return the first attribute in the expression
4	A State of the Sta
det	got Remaining Part (expression)
	get attribut from intilial predicate
The same	get attribut from intilial predicate return predicate + "("+", ". jain attribulus [1:].

Page No :
Jef unify (exp!, exp?)
if both are same return []  if both are constant return false.
if explin constant
return [cexp1, exp1)]
return [cexp2, exp1)]
if expl is variable
if exprocurs in exp2
elic:
return (expr, expl)
if exp2 is variable
if exp2 occurs in exp1
else else
return true. [Enp2, exp2)
if initial predicates don't match
if length of attitute I are and the
return false
if not initial soln
return False
If attribude(aint1 = =1)  return initial solyn
Jail 2 & Get Remaining Part (expl)
C C C C C C C C C C C C C C C C C C C

	S Detw :
if initial soln ! = ["	
fail = apply (to	
+ail 2 = apply (+	ail 2, initial soln)
remaining sat - un	rify (tail , tail 2)
return False	
	IVA A LANGE
· initial soln-extend ()	remaining Substitution)
return initial Substitu	
ex	Admiral experience in a state of
expl="knows(X)"	43 a military 1 34 Mars
exp2 = "knows(Fichard)"	90 300000000000000000000000000000000000
print ("Substitutions;	<i>u</i> )
print (substitutions)	this prime later !
Substitutions:	an and a sage of
[('X', richard')]	1877 1 A 131
	A LOUIS WAY 1920 MAY
Mineral Andreas L	white a short way
	Sales for the late of the

```
exp1 = "knows(X)"
    exp2 = "knows(Richard)"
    substitutions = unify(exp1, exp2)
    print("Substitutions:")
    print(substitutions)
Substitutions:
    [('X', 'Richard')]
exp1 = "knows(A,x)"
```

```
exp2 = "knows(y,mother(y))"
    substitutions = unify(exp1, exp2)
    print("Substitutions:")
    print(substitutions)
Substitutions:
    [('A', 'y'), ('mother(y)', 'x')]
```

### **Program 10: FOL to CNF**

#### Code:

```
def getAttributes(string):
  expr = ' ([^{\wedge})] + ')'
  matches = re.findall(expr, string)
  return [m for m in str(matches) if m.isalpha()]
def getPredicates(string):
  expr = '[a-z\sim]+([A-Za-z,]+)'
  return re.findall(expr, string)
def DeMorgan(sentence):
  string = ".join(list(sentence).copy())
  string = string.replace('\sim\sim','')
  flag = '[' in string
  string = string.replace('\sim[','')
  string = string.strip(']')
  for predicate in getPredicates(string):
     string = string.replace(predicate, f' \sim \{predicate\}')
  s = list(string)
  for i, c in enumerate(string):
     if c == '|':
        s[i] = '\&'
     elif c == '&':
        s[i] = '|'
  string = ".join(s)
  string = string.replace('\sim\sim','')
  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('[\forall \exists].', 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 = \lceil \cdot \rceil + \text{statement}[i] + '=>' + \text{statement}[i+1:] + '] \& ['+ \text{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 = '\sim' + statement[br:i] + '|' + statement[i+1:]
     statement = statement[:br] + new statement if br > 0 else new statement
  while '~∀' in statement:
     i = statement.index('\sim \forall')
     statement = list(statement)
     statement[i], statement[i+1], statement[i+2] = '\exists', statement[i+2], '\sim'
```

```
statement = ".join(statement)
   while '\sim \exists' in statement:
      i = statement.index('\sim \exists')
      s = list(statement)
      s[i], s[i+1], s[i+2] = '\forall', s[i+2], '\sim'
     statement = ".join(s)
   statement = statement.replace('\sim[\forall','[\sim\forall')
   statement = statement.replace('\sim[∃','[\sim∃')
   expr = '(\sim [\forall |\exists].)'
   statements = re.findall(expr, statement)
   for s in statements:
      statement = statement.replace(s, fol to cnf(s))
   expr = ' \sim \backslash [[^{\wedge}]] + \backslash ]'
   statements = re.findall(expr, statement)
   for s in statements:
      statement = statement.replace(s, DeMorgan(s))
   return statement
print(Skolemization(fol_to_cnf("animal(y)<=>loves(x,y)")))
print(Skolemization(fol\_to\_cnf("\forall x[\forall y[animal(y)=>loves(x,y)]]=>[\exists z[loves(z,x)]]")))
print(fol to cnf("[american(x)\&weapon(y)\&sells(x,y,z)\&hostile(z)]=>criminal(x)"))
```

1/2/2	Data:
100	Convert a given bist order logic statement into
	conjunctive normal form CNF is an opproach to express Foliar wind with
	ex A, AA2A AAn ex (AVB) A (CND)
	A CONTRACTOR OF THE PARTY OF TH
	def De Morgan (sentence)  replace (predicate, "~ {predicate}")  atro replace OR with AND
	join string list  (cplace ('nn' with '')  return f'[ Esting y ]' it flag else String
	def SICOLEMIZATION (SUNTURU):  the create a just of SICOLEM CONSTANTS
	det fol-to-cnf (fol):  statement = fol replace (" <=> , '-")  replace with > int in the statement  statement = statement replace (">" ,"-")  if I' is in S and 'I' not in S  add 'T'
	for sin statement:  Statement = statement. replace (s, Joi-to

```
convert A-DB into MAVB
   replace NV with I statement form N
    replace ~ I with V statement (part) ~
    explare returnists for
            CM with [N]
    (. [EIV] -) = Taxo
    statements - re-findall (expr, statements
    for sin statements.
        statement = statement replace(s, that Demogras)
    return statement
print (sho) emization (fol-to-ent ("animal (y))
                      (over(x,y")))
print (Sloolemization (JOL to CATCOUX (Vy
       [animal(y)=>love(x,y)]=>[72[loves(2,x)])
point (fol-to-engl " [american (x)4 weapon (y) &
    selle (riy, 2) & hostile (2) ] Somming (2)"))
[nanimal(y) /loves(xig)] & [Nlove(xig) lanimally)]
[animal (9(2)) 4 ~ loves (4,6(2))][loves (+(2),2)]
[ a american (1) 1 ~ weapon (4) 1 ~ sells (2,4,2) ]
    ~ hostile(2)] | criminal (2)
```

## **Program 11: Forward Reasoning**

```
Code:
import re
def isVariable(x):
  return len(x) == 1 and x.islower() and x.isalpha()
def getAttributes(string):
  expr = ' ([^{\wedge})] + )'
  matches = re.findall(expr, string)
  return matches
def getPredicates(string):
  expr = '([a-z\sim]+) \setminus ([^{\&}]+)'
  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]
```

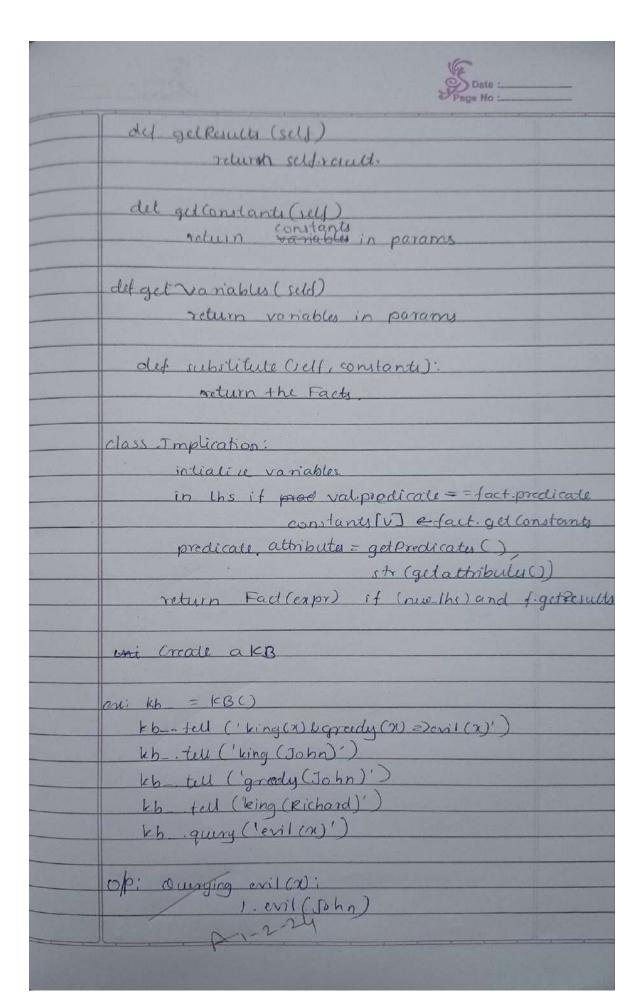
def getVariables(self):

```
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
     1 = expression.split('=>')
     self.lhs = [Fact(f) for f in 1[0].split('&')]
     self.rhs = Fact(1[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'\setminus 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\}')
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()
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)')

10)	Date :Pagu No :
11.10	(reate a ICB consisting of FOL statements and prove the given query using funct reasoning
	Forward masoning is a dop-down approach that Starts with available facts and uses
	derne conclusions and make decision.
	FI AND Decision en
	del and a later and a later a
	return len(a)=21 and x. islover() and x-is alpha
Consu	det getAttributes (strings) return attributes in the string
	def getPredicates (string) return all predicates in string
	class Fact:  initialize expression predicate params and result.
	def spliet Expression (self, expression)  sp return predicate and params of the expression.



```
print(f'\t{i+1}. {f}')
 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()
Querying criminal(x):

    criminal(West)

All facts:

 missile(M1)
```

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

```
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)')

Querying evil(x):
    1. evil(John)
```