VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

ARTIFICIAL INTELLIGENCE

Submitted by

JAGADEESH A LATTI (1BM21CS079)

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
Oct 2023-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 **JAGADEESH A LATTI (1BM21CS079)**, 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 2022-23. The Lab report has been approved as it satisfies the academic requirements in respect of Artificial Intelligence Lab - **(22CS5PCAIN)**work prescribed for the said degree.

Swathi Sridharan
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Table of Contents

1.Implement Tic –Tac –Toe Game.

```
if(xstate[choice]!=1 and zstate[choice]!=1):
    zstate[choice]=1
    count+=1
else:
    print("Already this space is occupied,
          Try again")
    continue
Cwin=checkwin(xstate,zstate)

if(Cwin!=-1):
    print("Match over")
    printboard(xstate,z_state)
    turn=False
    break
if(count==9):
    print("Match over")
    print("No space left")
    turn=False
    break
turn=1-turn
```

Algorithm:

1) sum(a,b,c):
   - This function takes 3 inputs & returns their sum

2) printboard(xstate,zstate):
   a) It prints tic-tac-toe board based on current states of x & z.
   b) It converts the state of each cell to 'x' if x has marked it, 'O' if z has marked

it or else cell number.

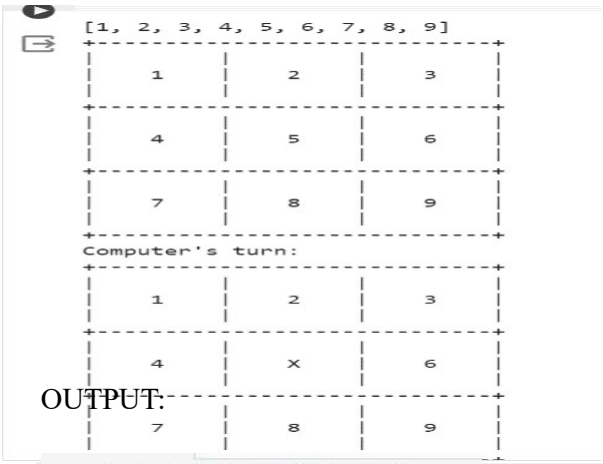
3) checkwin(xstate, zstate) {
   a) It checks who won the game
   b) It checks all possible winning combinations
   1) If x wins, it prints & returns 1
   2) if z wins, it prints & returns 0
   3) if there is no winner, it returns -1

4.
   a. The game runs in a loop until it is explicitly terminated.
   b) The game alternates turns between x & z until there is a winner or a draw.
   c) Printboard function is called to print current state of board.
   d) The checkwin function is called after each move to check if there's winner.
   e) Game ends when there is a winner or all spaces are filled.

Output:

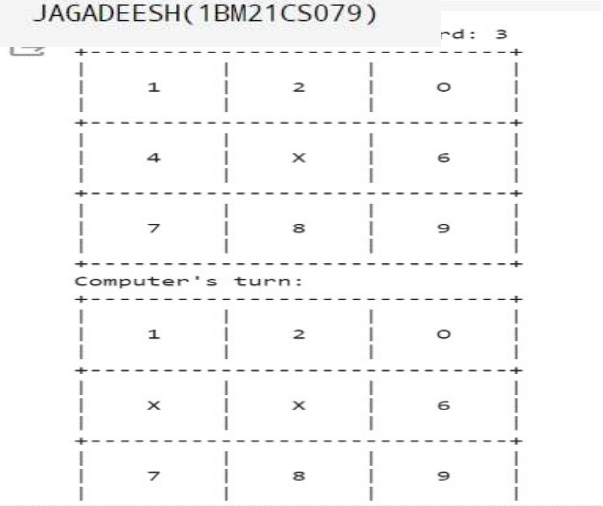
```
0 | 1 | 2
3 | 4 | 5
6 | 7 | 8

x's turn, Enter a number: 0
x | 1 | 2
3 | 4 | 5
6 | 7 | 8
```

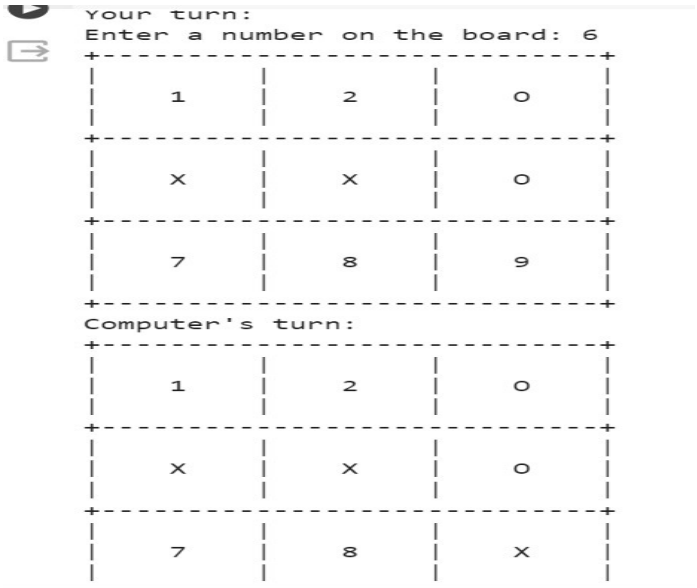
OUTPUT:



OUTPUT:

JAGADEESH(1BM21CS079)





```
Your turn:
Enter a number on the board: 1
+-------------------------------+
|               |               |
|       O       |       2       |       O       |
|               |               |
+-------------------------------+
|               |               |
|       X       |       X       |       O       |
|               |               |
+-------------------------------+
|               |               |
|       7       |       8       |       X       |
|               |               |
+-------------------------------+
Computer's turn:
+-------------------------------+
|               |               |
|       O       |       2       |       O       |
|               |               |
+-------------------------------+
|               |               |
|       X       |       X       |       O       |
|               |               |
+-------------------------------+
|               |               |
|       7       |       X       |       X       |
|               |               |
+-------------------------------+
```

```
Your turn:
Enter a number on the board: 2
+-------------------------------+
|               |               |
|       O       |       O       |       O       |
|               |               |
+-------------------------------+
|               |               |
|       X       |       X       |       O       |
|               |               |
+-------------------------------+
|               |               |
|       7       |       X       |       X       |
|               |               |
+-------------------------------+
Computer's turn:
+-------------------------------+
|               |               |
|       O       |       O       |       O       |
|               |               |
+-------------------------------+
|               |               |
|       X       |       X       |       O       |
|               |               |
+-------------------------------+
|               |               |
|       X       |       X       |       X       |
|               |               |
+-------------------------------+
Winner is X
```

2 .Solve 8 puzzle problems.

3. 8-puzzle:

Initialize the puzzle:

1) Create a Puzzle & class with the initial state, goal state, & possible moves

initial = [1, 2, 3, 4, 5, 6, 0, 7, 8]

goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]

moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]

2) define methods to print the current state, check if the puzzle is solved, get the index of blank tile & apply move.

Print current state:

for i in range(0, 9, 3):

print(state[i : i + 3])

Check if Puzzle is solved:

state == self.goal

Get index of the blank tile (represented by '0')

Apply move function:

Implement a method to apply a move to the current state, swapping the blank tile with adjacent tile.

BFS Algorithm

Initialize visited set to keep track of puzzle state to avoid revisiting.
visited = set()

Create an empty queue. It stores tuples where each tuple consists of puzzle state & corresponding path taken to reach state.
queue = deque()
queue.put((self.initial_state, [7]))

until queue is empty run the loop.
i) Deque a state & its path.
ii) Check if the state is the goal state.
if yes print solution path & break out of loop.

iii) if state is not visited
mark it as visited
iv) Get index of blank tile
v) Apply all move & execute the resulting state & path

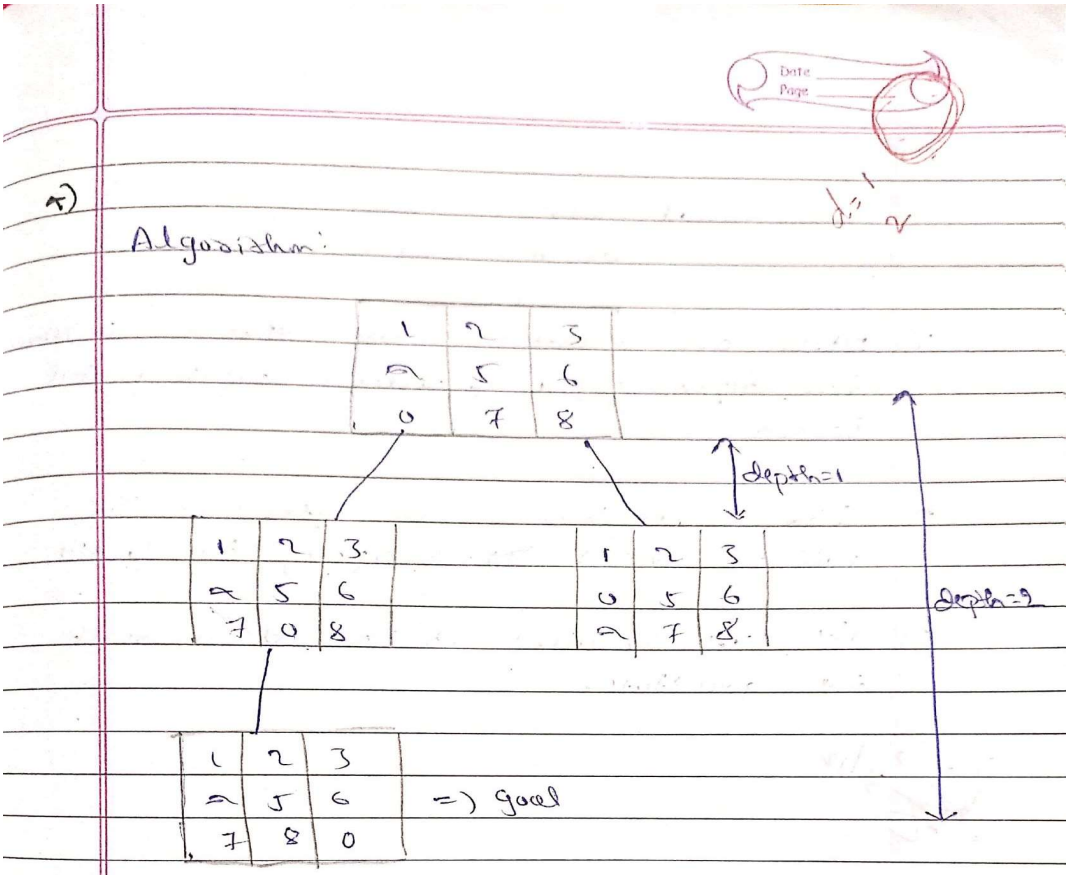
Initial state: [1, 2, 3, 4, 5, 6, 0, 7]
Queue: [(new state 1, [move 1]), (new state 2, [move 2])]

OUTPUT

```
JAGADEESH(1BM21CS079)
```

```
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
```

3. Implement Iterative deepening search algorithm.



Algorithm:

1) Initialize the initial state = [ ] & goal state for the 8-puzzle.
goal state: [1, 2, 3, 4, 5, 7, 8, 0]

2) set the depth=1 & expand the initial state
The depth_limited_search(depth) is performed
if node.state = goal
return node
else
for neighbour in get_neighbours(node.state)
child = puzzlenode(neighbour, node)
result = depth_limited_search(depth-1)

if result != Failure:
    return result

3) After one iteration where depth > 1, increase the depth by 1 & perform depth limited search.

4) Now get_neighbours will generate the possible moves by swapping the '0' tile

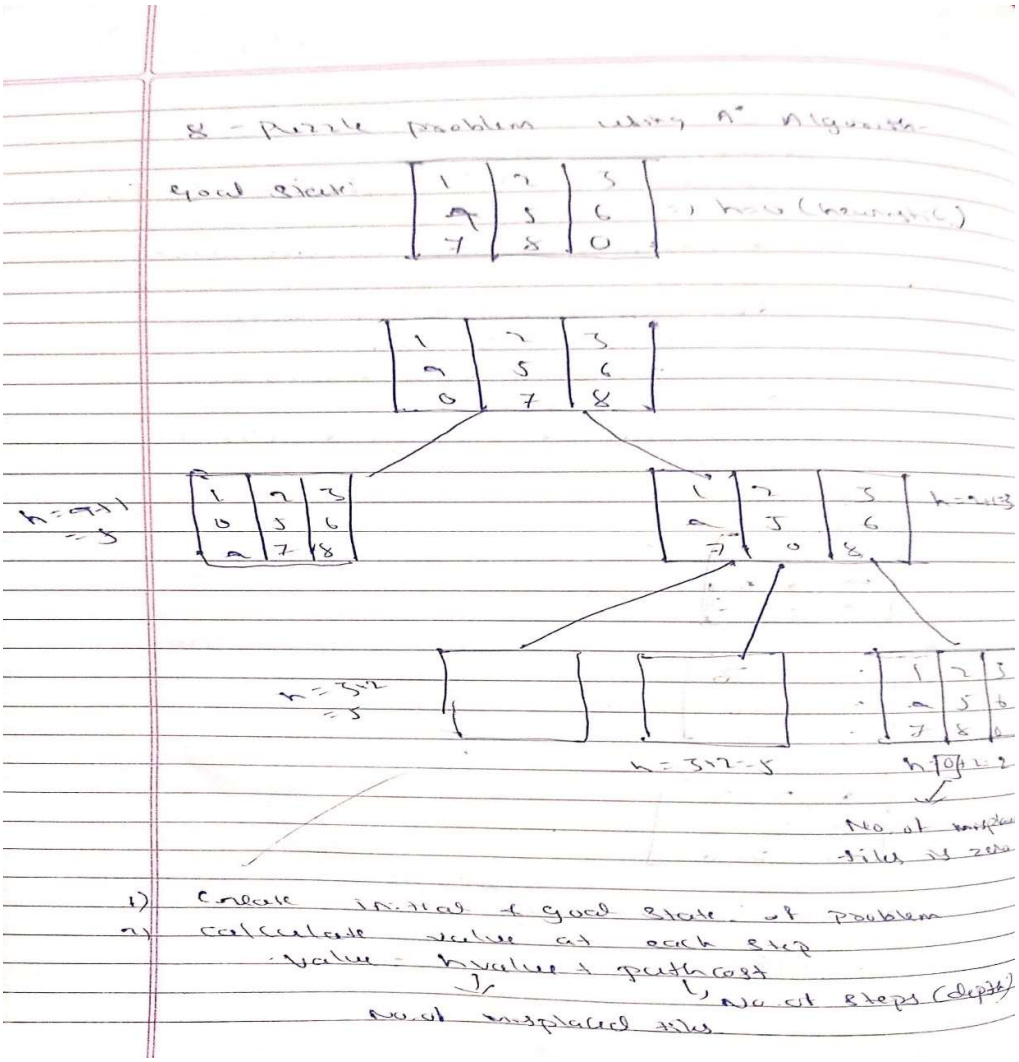
5) The path traversed is printed to reach the goal state.

8/12

OUTPUT

```
input
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 finished with exit code 0
Press ENTER to exit console.
```

4. Implement A* search algorithm.



3) Go to direction where h value is less.

4) All nods stored in open list (keep track of all nods).

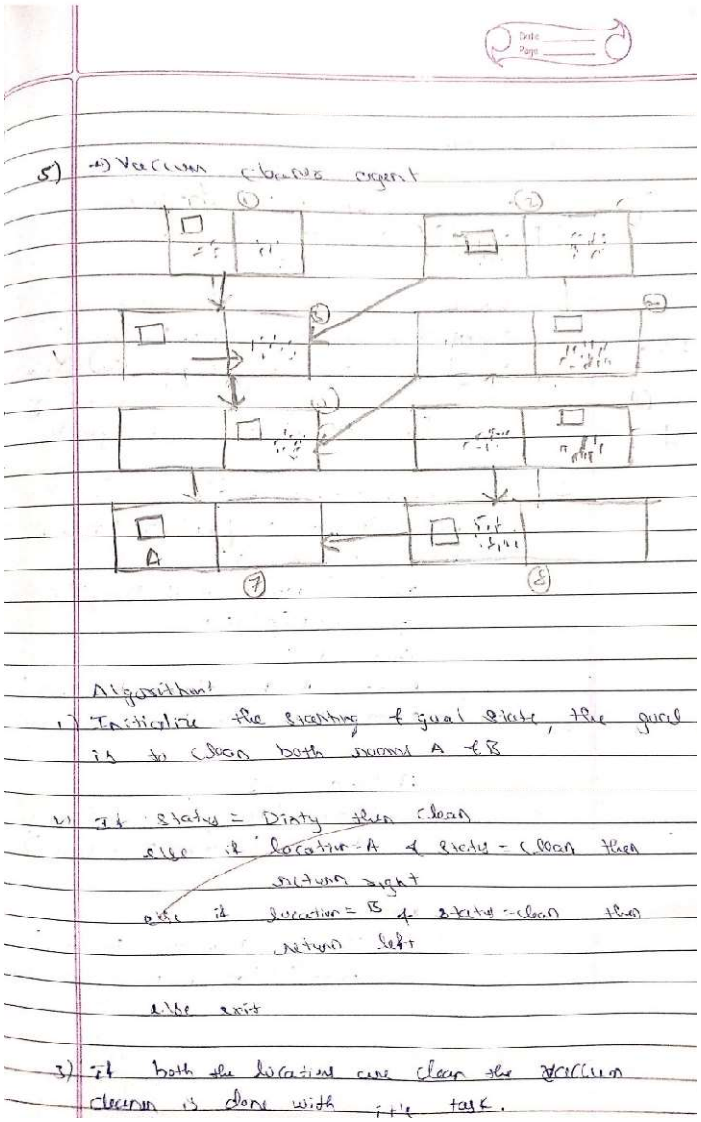
5) Explore nodes stored in closed list.

6) Goal is when h value is 0, implies that we have reached final state.

OUTPUT

```
Success! 8 puzzle problem solved
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]]
```

5. Implement vaccum cleaner agent.

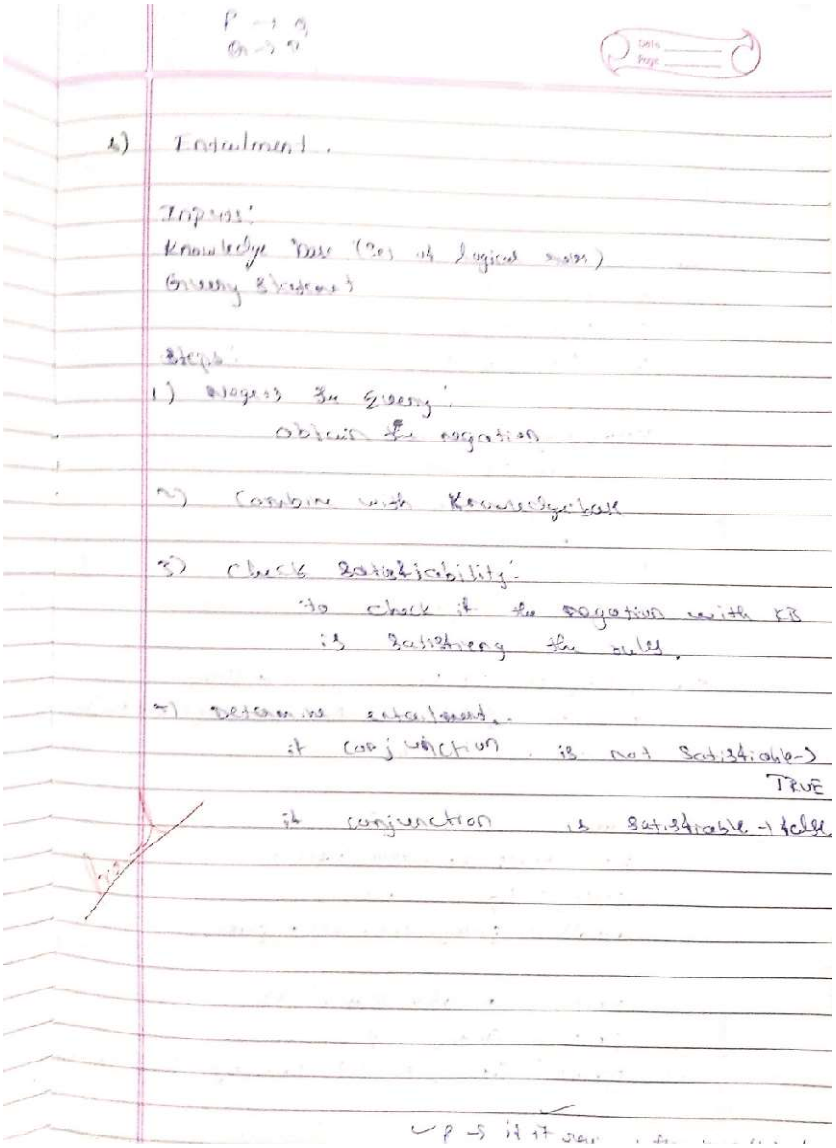


OUTPUT:

```
JAGADEESH(1BM21CS079)
```

```
Enter Location of Vacuumb
Enter status of b1
Enter status of other room1
Initial Location Condition{'A': '0', 'B': '0'}
Vacuum is placed in location B
Location B is Dirty.
COST for CLEANING 1
Location B has been Cleaned.
Location A is Dirty.
Moving LEFT to the Location A.
COST for moving LEFT2
COST for SUCK 3
Location A has been Cleaned.
GOAL STATE:
{'A': '0', 'B': '0'}
Performance Measurement: 3
```

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

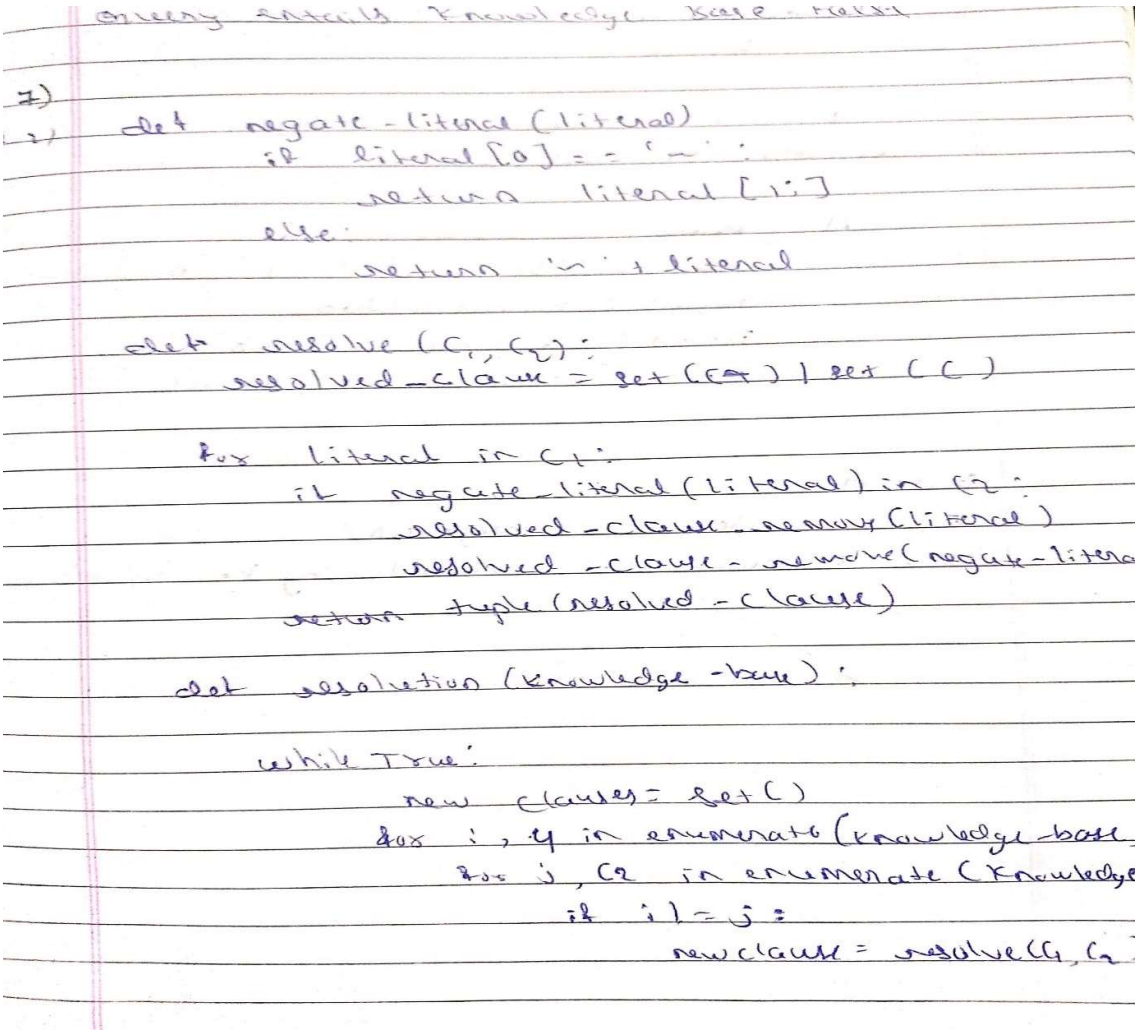


OUTPUT:

```
JAGADEESH(1BM21CS079)
```

```
Enter the knowledge base: (p^q)v(~pvq)
Enter the query: pvq
[True, True, True] :kb= True :q= True
[True, True, False] :kb= True :q= True
[True, False, True] :kb= False :q= True
[True, False, False] :kb= False :q= True
[False, True, True] :kb= True :q= True
[False, True, False] :kb= True :q= True
[False, False, True] :kb= True :q= False
Doesn't entail!!
```

7. Create a knowledge base using prepositional logic and prove the given query using resolution



```
7)
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)
            resolved_clause.remove(negate_litera
            return tuple(resolved_clause)

def resolution(knowledge_base):

    while True:
        new_clauses = set()
        for i, C1 in enumerate(knowledge_base
            for j, C2 in enumerate(knowledge
                if i != j:
                    new_clause = resolve(C1, C2
```

```
        if len(new_clause)>0 and new_clause not
                                    knowledge_base
            new_clause.add(new_clause)

    if not new_clause
        break

knowledge_base1 = new_clause
return knowledge_base

if __name__ == "__main__":
    kb = Clause('p','q'), ('~p','r'), ('~q','~r')
    result = resolution(kb)
    print("original kb", kb)
    print("resolved KB", result)
```

Output:

Enter statement = Negative

The statement not entailed by knowledge-based

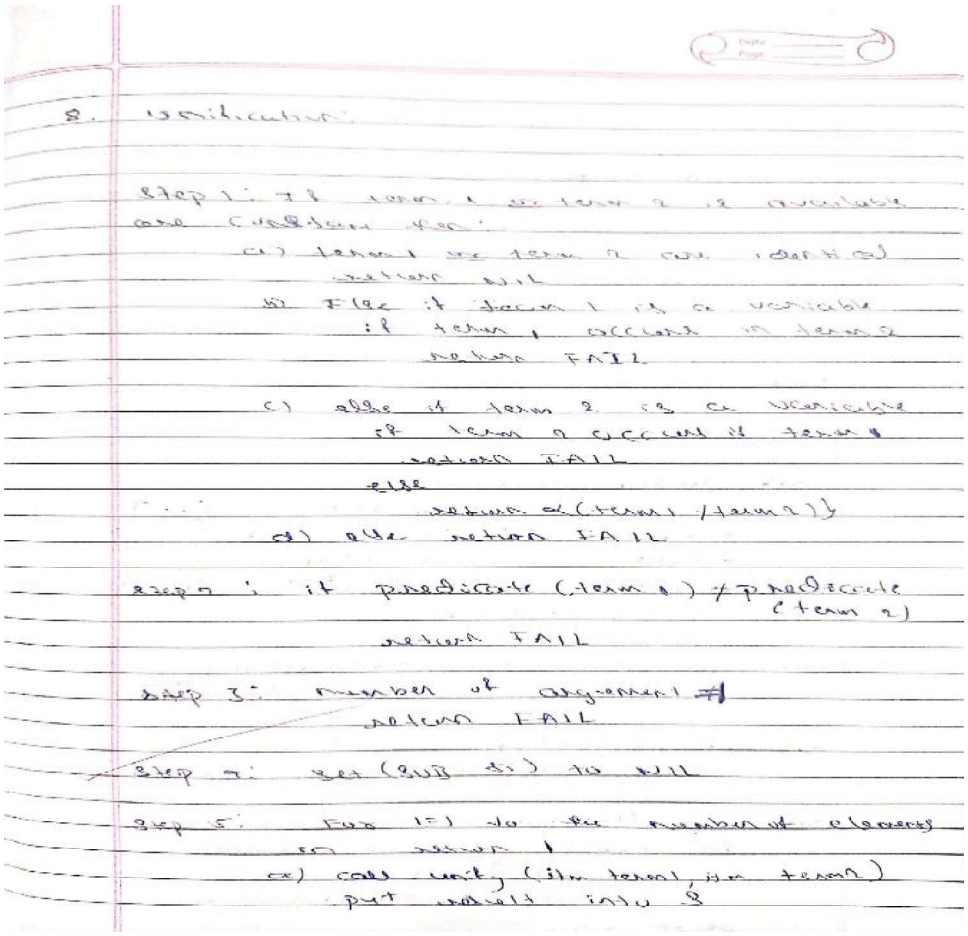
OUTPUT:

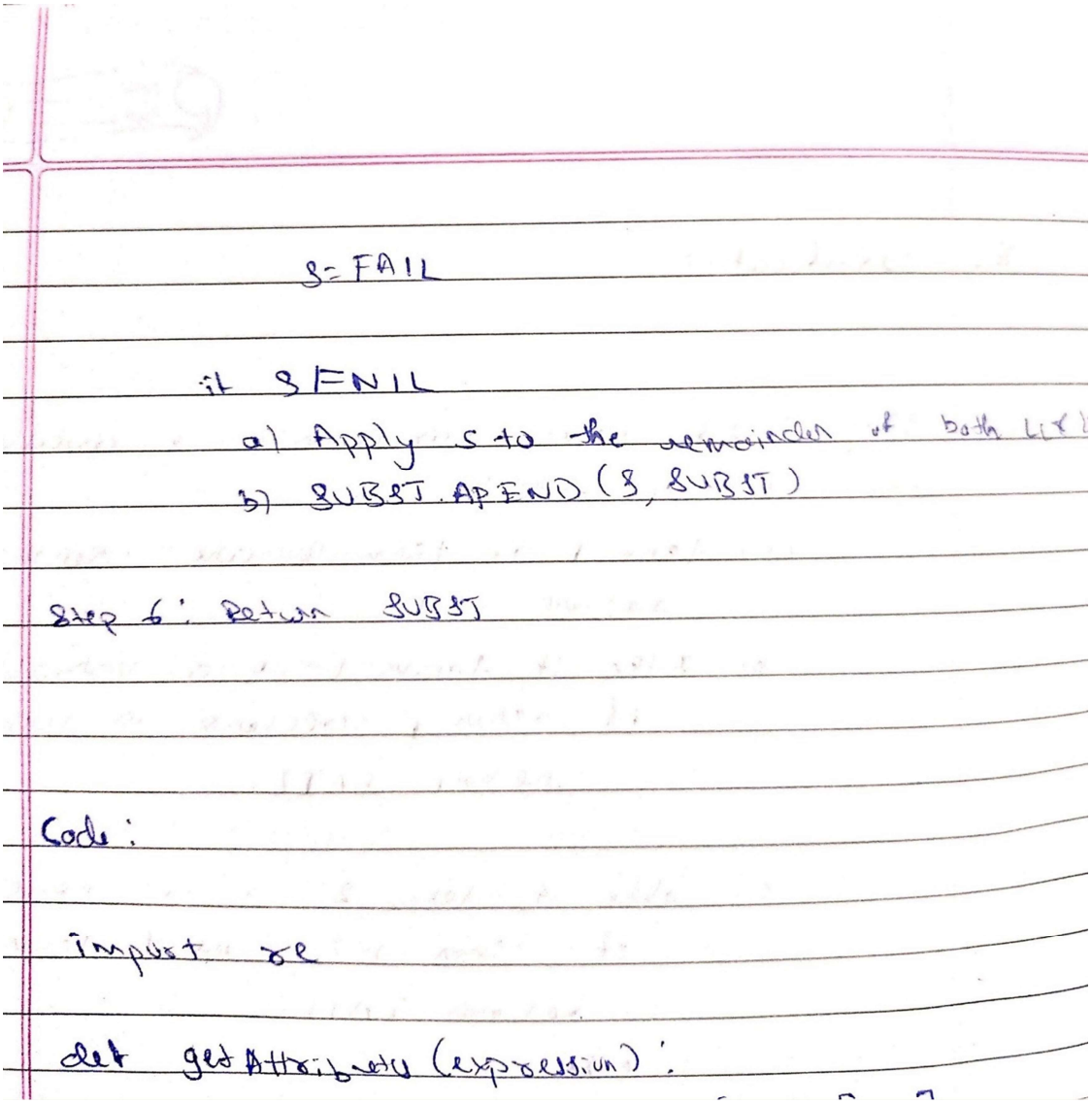
```
JAGADEESH(1BM21CS079)

Enter the clauses separated by a space: p v ~q ~r v p ~q
Enter the query: ~p
Trying to prove (p)^(v)^(~q)^(~r)^(v)^(p)^(~q)^(~p) by contradiction....
Knowledge Base entails the query, proved by resolution
```

```
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.
```

```
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.
```

8. Implement unification in first order logic :



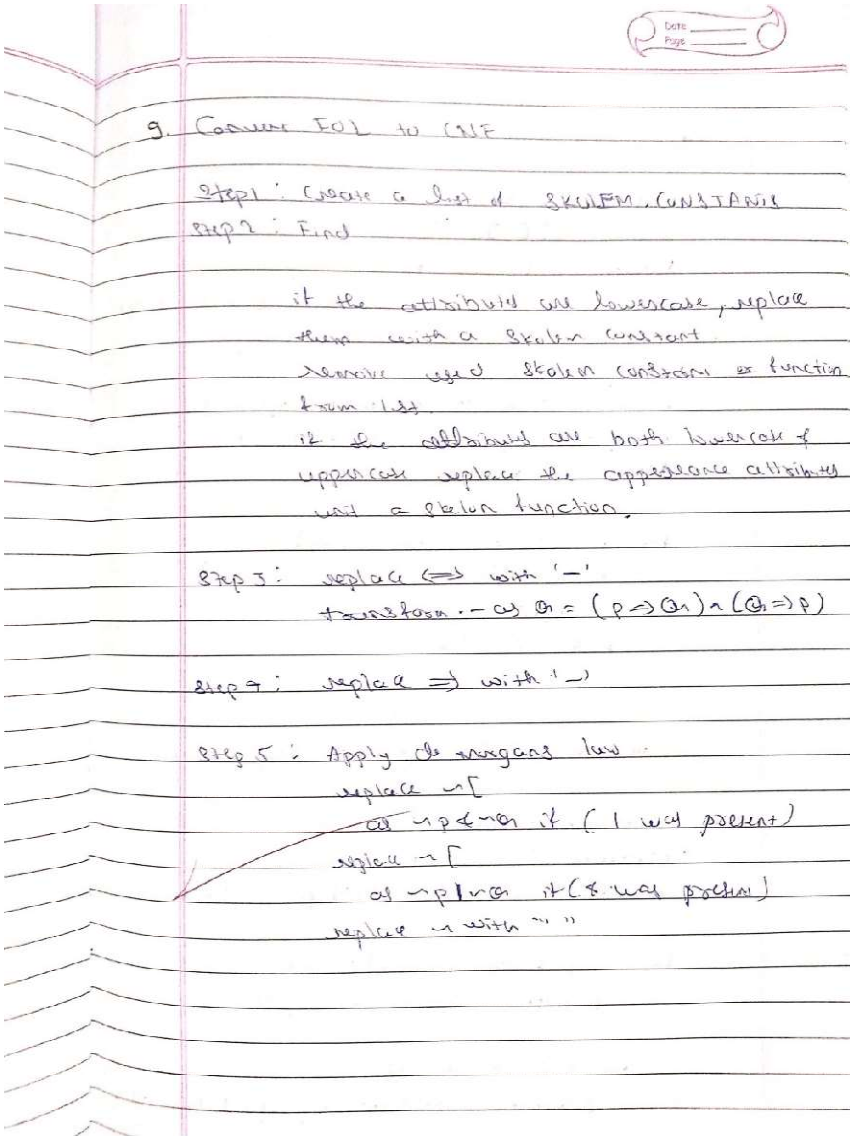


OUTPUT

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

```
Substitutions:
[('A', 'y'), ('mother(y)', 'x')]
```

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



9. Convert FOL to CNF

Step1 : Create a list of SKOLEM CONSTANTS

Step2 : Find

if the attributes are lowercase, replace them with a Skolem constant

recursive use of skolem constants or function term list.

if the attributes are both lowercase & uppercase replace the uppercase attribute with a skolem function.

Step3 : replace ⟺ with '→'

translation - $\alpha_1 \Leftrightarrow \alpha_2 = (p \Rightarrow \alpha_1) \wedge (\alpha_2 \Rightarrow p)$

Step4 : replace ⇒ with '→'

Step5 : Apply De morgans law

replace ~[

as ~ operator if (I was present)

replace ~[

as ~ plus it (& was present)

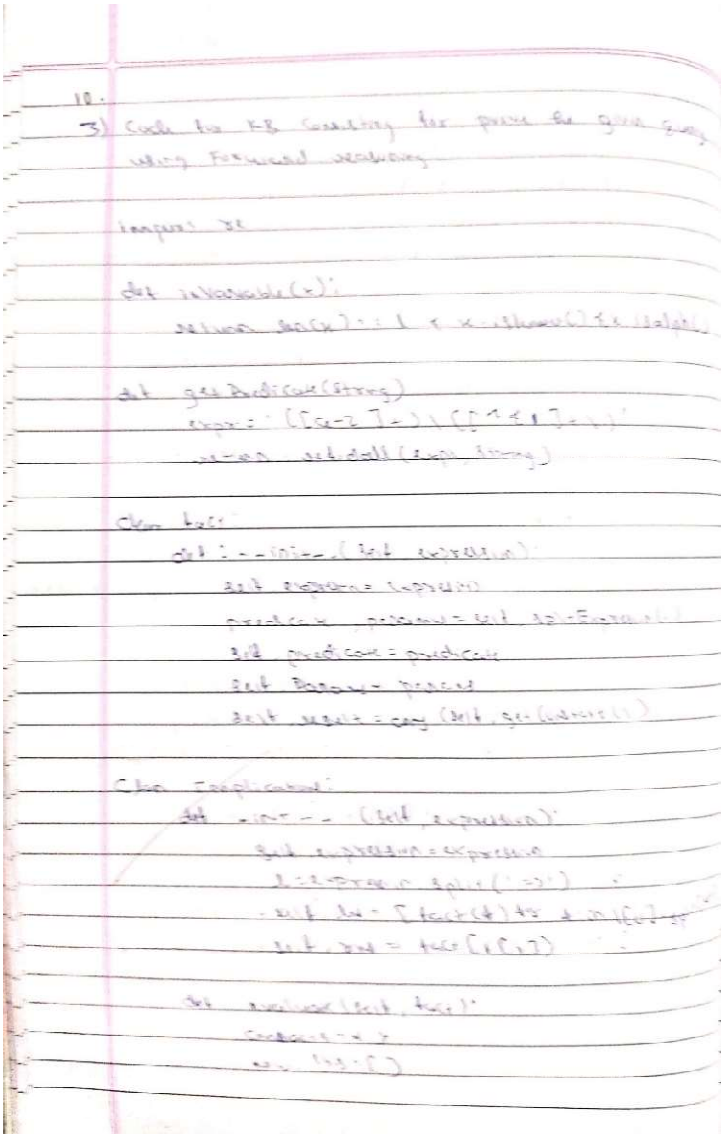
replace ~ with "~"

OUTPUT

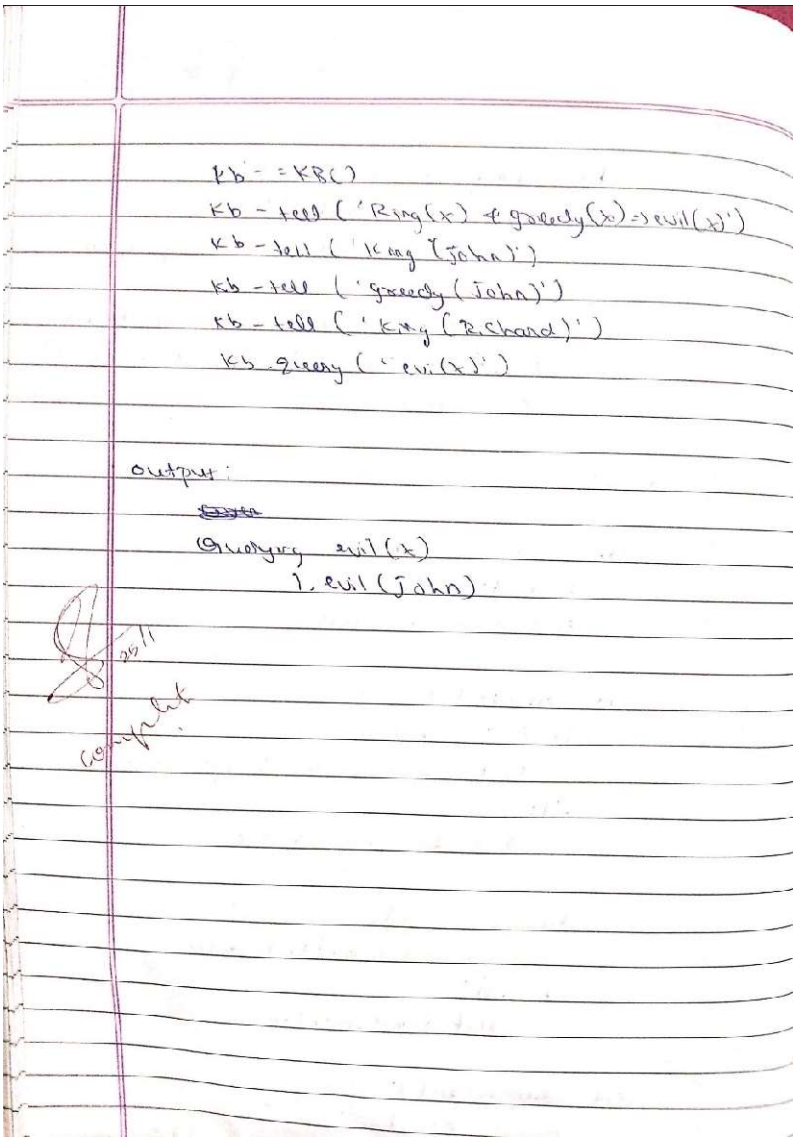
```
JAGADEESH(1BM21CS079)
```

```
Enter FOL statement: x+y_z*s
FOL converted to CNF: [~x+y|z*s]&[~z*s|x+y]
```

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



```
for key in customers:
    if customers[key]:
        attributes = attributes.replace

return Fact(expr) if len(new_line) ( all
    ( [f.get_result () for f in new])
    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 display(self):
        print("All facts: ")
        for i, f in enumerate(set([f.expression
                                  for f in self.facts])):
            print(f'{i+1}. {f}')
```



OUTPUT

JAGADEESH(1BM21CS079)

```
Enter number of statements in Knowledge Base: 4
Elephant(x) => Mammal(x)
Lion(Mufasa)
Mammal(x) => Animal(x)
Animal(Simba)
Enter Query:
Mammal(x)
Querying Mammal(x):
All facts:
        1. Lion(Mufasa)
        2. Animal(Simba)
```