

**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

<b>SL No</b>	<b>Name of Experiment</b>	<b>Page No</b>
1	Implement Tic –Tac –Toe Game	1-7
2	Implement 8 puzzle problem	8-12
3	Implement Iterative deepening search algorithm.	13-17
4	Implement A* search algorithm.	17-23
5	Implement vaccum cleaner agent.	23-27
6	Create a knowledge base using prepositional logic and show that the given query entails the knowledge base or not .	28-30
7	Create a knowledge base using prepositional logic and prove the given query using resolution	30-35
8	Implement unification in first order logic	36-41
9	Convert a given first order logic statement into Conjunctive Normal Form (CNF).	42-46
10	Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.	46-52

## 1. Implement Tic - Tac - Toe Game.

Date \_\_\_\_\_  
Page \_\_\_\_\_

```
if (xstate[choice] != 1 and zstate[choice] != 1):
```

```
    xstate[choice] = 1
```

```
    count += 1
```

```
else:
```

```
    print ("Already this space is occupied,  
try again")
```

```
    continue
```

```
win = checkwin(xstate, zstate)
```

```
if (win == -1):
```

```
    print ("Match over")
```

```
    pointboard(xstate, zstate)
```

```
    over = False
```

```
    break
```

```
if (count == 9):
```

```
    print ("Match over")
```

```
    print ("No space left")
```

```
    over = False
```

```
    break
```

```
    turn = (-turn)
```

Algorithm:

1) sum(a, b, c) :

↳ This function takes 3 inputs & return their sum

2) pointboard(xstate, zstate) :

↳ It prints tic-tac-toe board based on current state of X & Z

↳ It converts the state of each cell to 'X' if X has marked it, 'O' if Z has marked it

it or else ~~the~~ cell number.

### 3) checkwin(xstate, zstate) {

    ↳ It checks who won the game

    ↳ It checks all possible winning combinations

    ↳ If X wins, it prints & returns 1

    ↳ If Z wins, it prints & returns 0

    ↳ If there is no winner, it returns -1

4. The game runs in a loop until it is explicitly terminated.

    ↳ The game alternates turns between X & Z until there is a winner or a draw.

    ↳ Point-based function is called to print current state of board.

    ↳ The checkwin function is called after each move to check if there's winner.

    ↳ 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:

[1, 2, 3, 4, 5, 6, 7, 8, 9]

1	2	3
4	5	6
7	8	9

Computer's turn:

1	2	3
4	x	6
7	8	9

OUTPUT:

1	2	3
4	x	6
7	8	9

JAGADEESH(1BM21CS079)

rnd: 3

1	2	o
4	x	6
7	8	9

Computer's turn:

1	2	o
x	x	6
7	8	9

YOUR TURN:

Enter a number on the board: 6

1	2	o
x	x	o
7	8	9

Computer's turn:

1	2	o
x	x	o
7	8	x

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

Computer's turn:

o	o	o
x	x	o

X

Winner is X

## 2 .Solve 8 puzzle problems.

3.

8-puzzle:

Initialize the puzzle:

- 1) Create a Puzzle8 class with the initial state, goal state & possible moves

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

goal = [2, 1, 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 = Queue()

queue.put((state, initial\_state, [ ]))

until queue is empty over 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~~ tile not visited

mark it as visited

iv) get index of blank tile

v) Apply all moves & enqueue the resulting states & paths.

initial state: [1, 2, 3, 7, 5, 6, 4, 0]

queue: [(newstate-1, [move-1]), (newstate-2[move-2])]

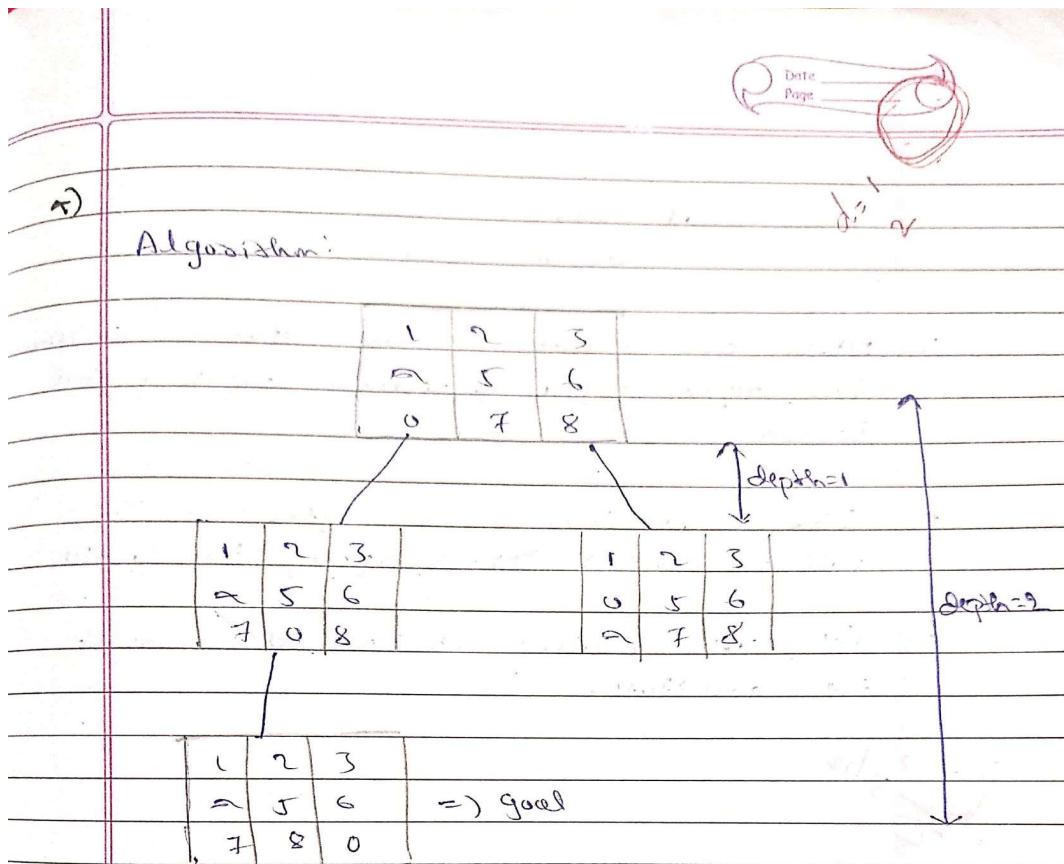
for all

## 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, 6, 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 = puzzle-node(neighbour, node)

result = depth-limited-search(depth-1)

if search == True;  
return result

- 3) After one iteration where depth > 1, increment the depth by 1 & perform depth limited search.
- 4) Here get\_neighbors will generate the possible moves by swapping the '0' tile.
- 5) The path travelled is pointed to reach the goal state.

vector<string> v1

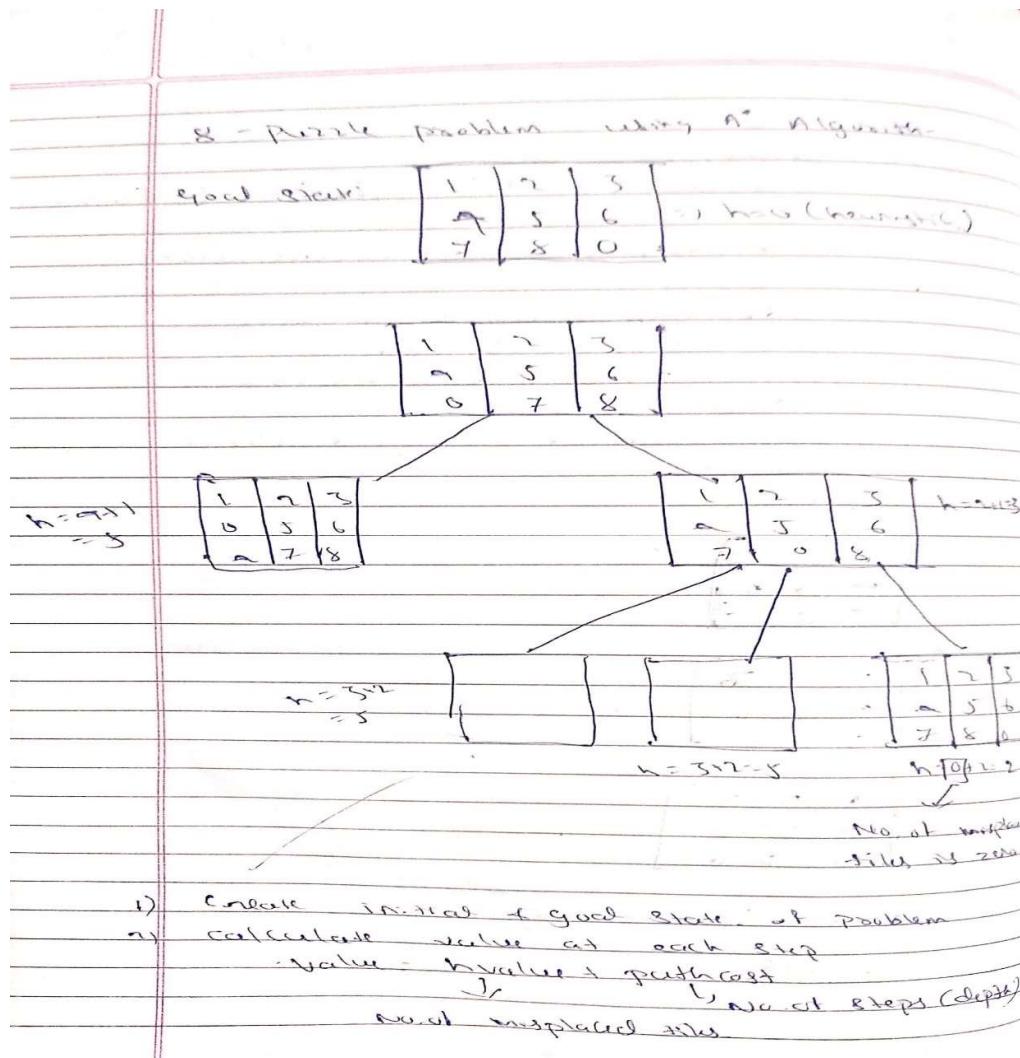
## OUTPUT

```

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 f-value is less.

a) All nodes stored in open list (keep track of cell nodes).

b) Explore nodes stored in closed list.

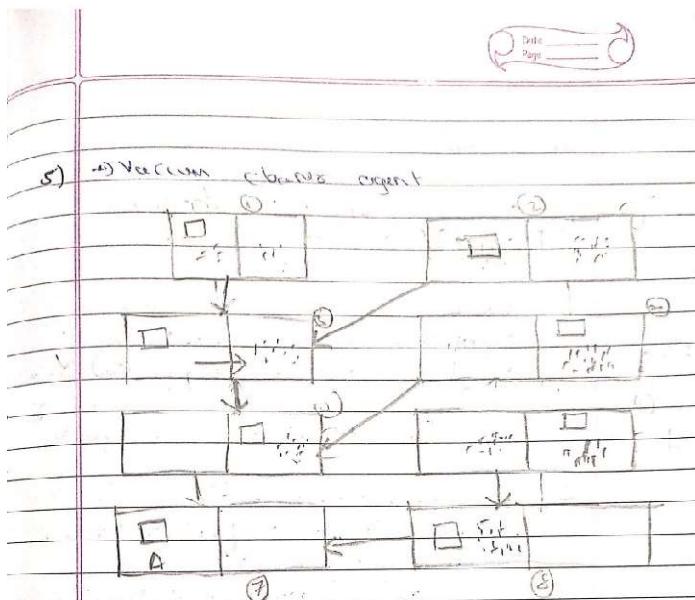
c) Goal is when f-value is 0, implies that we have reached final state.

(S)  
Goal

## 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.



- Algorithm
- i) Initialize the starting & goal state, the goal is to clean both around A & B
  - ii) if state = Dirty then clean  
else if location = A & dirty = clean then  
return right  
else if location = B & dirty = clean then  
return left
  - iii) exit
  - 4) if both the location are clean the vacuum cleaner is done with its task.

OUTPUT:

JAGADEESH(1BM21CS079)

```
vacuum>
→ 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 propositional logic and show that the given query entails the knowledge base or not .

$$P \rightarrow Q$$
$$\neg P \rightarrow \neg Q$$

100%  
true

### a) Entailment.

Input:

Knowledge Base (KB) in logical forms  
Grounded KB forms

Step:

i)  $\neg P \rightarrow \neg Q$  Given

obtain the negation

ii) Combine with Knowledge base

iii) Check satisfiability

to check if the conjunction with KB

is satisfiable the rules.

iv) Determine satisfiability

if conjunction is not satisfiable  $\rightarrow$  TRUE

if conjunction is satisfiable  $\rightarrow$  false

$\neg P \rightarrow \neg Q$  is it true or false

OUTPUT:

JAGADEESH(1BM21CS079)

Enter the knowledge base:  $(P \wedge Q) \vee (\neg P \vee Q)$   
Enter the query:  $P \vee Q$   
[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 propositional logic and prove the given query using resolution

every entails Knowledge base - maxwell

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\_literal(literal))  
 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\_base):  
 if i != j:  
 new\_clauses = resolve(C1, C2)

if len(new-clause) > 0 & new-clause not  
knowledge base  
new-clause.add(new-clause)

it is not new clause

break

knowledge-base = new-clause  
return knowledge-base

if clause == "original-":

kb = {('p', 'g'), ('¬p', 'δ'), ('¬g', '¬δ')}

result = resolution(kb)

point("original kb", kb)

point("resolved kb", result)

Output:

Enter statement = Negation

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  $(p) \wedge (v) \wedge (\neg q) \wedge (\neg r) \wedge (v) \wedge (p) \wedge (\neg q) \wedge (\neg p)$  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 :

8. unification

Step 1: If both terms are variables or constants  
case 1: both are term 1 are identical  
return true

case 2: If term 1 is a variable  
if term 1 occurs in term 2  
return FAIL

case 3: If both are variables  
if term 1 occurs in term 2  
return FAIL

else

return ok(terms, terms)

case 4: else return FAIL

Step 2: if predicate(term 1) ≠ predicate(term 2)  
return FAIL

Step 3: member of argument #  
return FAIL

Step 4: set (arg# 1) to fail

Step 5: For i = 1 to the number of elements  
in argument 1  
case 1: if term 1 is a variable  
put variable into S

$s = \text{FAIL}$

if  $s \neq \text{NIL}$

- a) Apply  $s$  to the remainder of both lists
- b)  $\text{SUBST-APEND}(s, \text{SUBST})$

Step 6: Return  $\text{SUBST}$

Code:

```
import re
def get_attributes(expression):
```

## OUTPUT

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

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

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

Date \_\_\_\_\_  
Page \_\_\_\_\_

g. Convert FOL to CNF

Step 1: Create a list of SKOLEM CONSTANTS

Step 2: Find

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

Attribute used: Skolem constant or function  
format:  $\lambda$

if the attributes are both lowercase or uppercase, replace the uppercase attributes with a Skolem function.

Step 3: replace  $\Leftrightarrow$  with ' $\neg$ '  
transform.  $\neg \phi \Rightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$

Step 4: replace  $\Rightarrow$  with ' $\neg$ '

Step 5: Apply De Morgan law  
replace  $\neg\neg$   
as  $\neg\neg p \equiv p$  if ( $\neg$  was present)  
replace  $\neg\neg$   
as  $\neg\neg p \equiv p$  if ( $\neg$  was present)  
replace  $\neg\neg$  with " "

OUTPUT

JAGADEESH(1BM21CS079)



Enter FOL statement:  $x+y\_z*s$

FOL converted to CNF:  $[\neg x+y \mid z*s] \wedge [\neg z*s \mid x+y]$

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

10.  
3) Create the KB consisting for prove the given query  
using Forward reasoning.

Impress:  $\lambda$

def isAvailable( $x$ ):  
 return  $safe(x) \wedge \neg \text{isBusy}(x)$

at getAvailable(string):  
 expr = (( $\lambda x \exists y$ ) \ (\mathcal{C} \in \mathcal{A} \ \mathcal{C} = y))

return  $\text{setDiff}(\mathcal{C}, \mathcal{A})$

class Test:  
 def \_\_init\_\_(self, expression):  
 self.expression = expression  
 def evaluate(self, environment):  
 if self.expression == 'x':  
 return environment['x']  
 elif self.expression == 'y':  
 return environment['y']  
 else:  
 return self.expression[1:-1]  
 def \_\_str\_\_(self):  
 return str(self.expression)

def setDiff(left, right):  
 left = list(left)  
 right = list(right)  
 result = [x for x in left if x not in right]  
 return result

def evaluate(left, right):  
 environment = {}  
 environment['x'] = left  
 environment['y'] = right  
 test = Test(left)  
 result = test.evaluate(environment)  
 return result

A diagram of a single neuron. It features a central circular cell body labeled "cell body". Two short, branching lines extend from the left side of the cell body, labeled "dendrite". A long, thin line extends from the right side of the cell body, labeled "axon".

for Reg. in customs;  
it (was) very bad;

Overall feels (large) & feel (anatomical) (soft  
( E.E. gets you to ) & for & in mind )  
else work

class X IS

~~def init - (self):~~  
~~def self, self = self()~~  
~~self. implements = self()~~

did well (last, e').

It is  $\Rightarrow$  we  
test implication and implement it?

else:  
    verb, noun, and character)

then  $\tau$  is left-rightward.  
and evaluate left-hand  
at  $\tau\tilde{\tau}$ .  
such. Right-evaluates

Q. A. Gregory (left):  
Please "tell both!"  
There is, I am convinced, a lot of experience  
there & in research.  
Please check it out.

$KB = KB()$   
 $KB - tell ( 'Ring(x) \wedge greedy(x) \Rightarrow evil(x)' )$   
 $KB - tell ( 'King(John)' )$   
 $KB - tell ( 'greedy(John)' )$   
 $KB - tell ( 'King(Richard)' )$   
 $KB - query ( 'evil(x)' )$

**Output:**  
Querying  $evil(x)$   
 1.  $evil(John)$

Q  
 (Q) ~~query~~

## OUTPUT

JAGADEESH(1BM21CS079)

↳ Enter number of statements in Knowledge Base: 4  
 Elephant(x)  $\Rightarrow$  Mammal(x)  
 Lion(Mufasa)  
 Mammal(x)  $\Rightarrow$  Animal(x)  
 Animal(Simba)  
 Enter Query:  
 Mammal(x)  
 Querying Mammal(x):  
 All facts:  
 1. Lion(Mufasa)  
 2. Animal(Simba)