

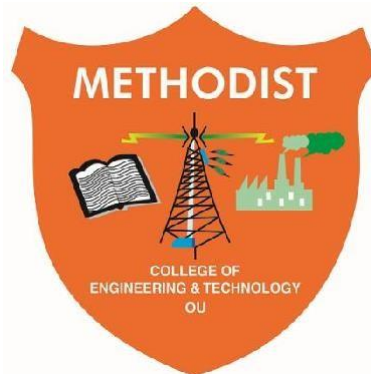


Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

(Affiliated to Osmania University & Approved by AICTE, New Delhi)



LABORATORY MANUAL

ARTIFICIAL INTELLIGENCE

LABORATORY

BE V Semester (AICTE Model Curriculum): 2022-23

NAME: _____

ROLL
NO: _____

BRANCH: _____ SEM: _____

**DEPARTMENT OF COMPUTER
SCIENCE &
ENGINEERING**

Empower youth- Architects of Future World

METHODIST

Estd:2008 COLLEGE OF ENGINEERING AND TECHNOLOGY

VISION

To produce ethical, socially conscious and innovative professionals who would contribute to sustainable technological development of the society.

MISSION

To impart quality engineering education with latest technological developments and interdisciplinary skills to make students succeed in professional practice.

To encourage research culture among faculty and students by establishing state of art laboratories and exposing them to modern industrial and organizational practices.

To inculcate humane qualities like environmental consciousness, leadership, social values, professional ethics and engage in independent and lifelong learning for sustainable contribution to the society.

METHODIST

Estd:2008 COLLEGE OF ENGINEERING AND TECHNOLOGY

**DEPARTMENT
OF
COMPUTER SCIENCE AND
ENGINEERING**

**LABORATORY MANUAL
ARTIFICIAL INTELLIGENCE
LABORATORY**

**Prepared
By**

**Ms. Sana Mateen
Associate Professor, Dept. of CSE**



METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND

VISION & MISSION

VISION

To become a leader in providing Computer Science & Engineering education with emphasis on knowledge and innovation.

MISSION

- To offer flexible programs of study with collaborations to suit industry needs.
- To provide quality education and training through novel pedagogical practices.
- To expedite high performance of excellence in teaching, research and innovations.
- To impart moral, ethical values and education with social responsibility.



METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND PROGRAM EDUCATIONAL OBJECTIVES

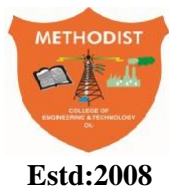
After 3-5 years of graduation, the graduates will be able to

PEO1: Apply technical concepts, Analyze, Synthesize data to Design and create novel products and solutions for the real life problems.

PEO2: Apply the knowledge of Computer Science Engineering to pursue higher education with due consideration to environment and society.

PEO3: Promote collaborative learning and spirit of team work through multidisciplinary projects

PEO4: Engage in life-long learning and develop entrepreneurial skills.



METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND PROGRAM OUTCOMES

Engineering graduates will be able to:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the Engineering community and with society at large, such as, being able to

comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

At the end of 4 years, Computer Science and Engineering graduates at MCET will be able to:

PSO1: Apply the knowledge of Computer Science and Engineering in various domains like networking and data mining to manage projects in multidisciplinary environments.

PSO2: Develop software applications with open-ended programming environments.

PSO3: Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms

Course Code	Course Title					Core /Elective	
PC 553 CS	Artificial Intelligence Lab					Core	
Prerequisite	Contact Hours per Week				CIE	SEE	Credits
	L	T	D	P			
-	-	-	-	2	25	50	1

Course Objectives

- To apply programming skills to formulate the solutions for computational problems.
- To study implementation first order predicate calculus using Prolog
- To familiarize with basic implementation of NLP with the help of Python libraries NLTK
- To understand python library scikit-learn for building machine learning models
- To enrich knowledge to select and apply relevant AI tools for the given problem

Course Outcomes

CO1: Design and develop solutions for informed and uninformed search problems in AI.

CO2: Demonstrate reasoning in first order logic using Prolog

CO3: Utilize advanced package like NLTK for implementing natural language processing.

CO4: Demonstrate and enrich knowledge to select and apply python libraries to synthesize information and develop supervised learning models

CO5: Develop a case study in multidisciplinary areas to demonstrate use of AI

List of Experiments to be performed:

1. Write basic programs on python
2. Write python program to implement List and tuples
3. Write a program to implement Water-Jug problem.
4. Write a program to implement Uninformed search techniques:
 - a. BFS
 - b. DFS
5. Write a program to implement Informed search techniques
 - a. Greedy Best first search
 - b. A* algorithm
6. Study of Prolog, its facts, and rules.
 - a. Write simple facts for the statements and querying it.
 - b. Write a program for Family-tree.
7. Write a python program to implement the methods of numpy.
8. Write a python program to create the following using pandas
 - a. create a dataframe from the dictionary
 - b. List the top and bottom 10 rows from the dataframe
 - c. Display the dimensions
 - d. Access the data at index 3
9. Write a program to train and validate the following classifiers for given data (scikit-learn):
 - a. Decision Tree
 - b. Multi-layer Feed Forward neural network
10. Text processing using NLTK
 - a. Remove stop words
 - b. Implement stemming
 - c. POS (Parts of Speech) tagging

In addition to the above programs, students should be encouraged to study implementations of one of the following

- Game bot (Tic Tac toe, 7 puzzle)
- Expert system (Simple Medical Diagnosis)
- Text classification
- Chat bot



METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Outcomes (CO's):

SUBJECT NAME :ARTIFICIAL INTELLIGENCE LAB

CODE :PC 552 CS

SEMESTER :V

CO No.	Course Outcome	Taxonomy Level
PC 553 CS.1	Design and develop solutions for informed and uninformed search problems in AI	Create
PC 553 CS.2	Demonstrate reasoning in first order logic using Prolog	Apply
PC 553 CS.3	Utilize advanced package like NLTK for implementing natural language processing	Apply
PC 553 CS.4	Demonstrate and enrich knowledge to select and apply python libraries to synthesize information and develop supervised learning models	Apply
PC 553 CS.5	Develop a case study in multidisciplinary areas to demonstrate use of AI	Create



Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the program / experiment details.
3. Student should enter into the laboratory with:
 - a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
 - b. Laboratory Record updated up to the last session experiments.
 - c. Formal dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviours with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out. If anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department

Principal



Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CODE OF CONDUCT FOR THE LABORATORY

- All students must observe the dress code while in the laboratory
- Footwear is NOT allowed
- Foods, drinks and smoking are NOT allowed
- All bags must be left at the indicated place
- The lab timetable must be strictly followed
- Be PUNCTUAL for your laboratory session
- All programs must be completed within the given time
- Noise must be kept to a minimum
- Workspace must be kept clean and tidy at all time
- All students are liable for any damage to system due to their own negligence
- Students are strictly PROHIBITED from taking out any items from the laboratory
- Report immediately to the lab programmer if any damages to equipment

BEFORE LEAVING LAB:

- Arrange all the equipment and chairs properly.
- Turn off / shut down the systems before leaving.
- Please check the laboratory notice board regularly for updates.

Lab In – charge



Estd:2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LIST OF EXPERIMENTS

Sl. No.	Name of the Experiment	Date of Experiment	Date of Submission	Page No	Faculty Signature
1.	Write basic programs on python				
2.	Write python program to implement List and tuples				
3.	Write a Program to implement Water Jug Problem.				
4.	Write a program to implement Uninformed search techniques: a. BFS b. DFS				
5.	Write a program to implement Informed search techniques a. Greedy Best first search b. A* algorithm				
6.	Study of Prolog, its facts, and rules. a. Write simple facts for the statements and querying it. b. Write a program for Family-tree				
7.	Write a python program to implement the methods of numpy.				
8.	Write a python program to create the following using pandas a. create a dataframe from the dictionary b. List the top and bottom 10 rows from the dataframe c. Display the dimensions d. Access the data at index 3				
9.	Write a program to train and validate the following classifiers for given data (scikit-learn): a. Decision Tree b. Multi-layer FeedForward neural network				

10.	Text processing using NLTK a. Remove stop words b. Implement stemming c. POS (Parts ofSpeech) tagging.				
-----	--	--	--	--	--

PROGRAM 1.**1.Basic programs of python**

```
[1] name=input("enter ur name")
```

```
enter ur name sana
```

```
[2] name
```

```
' sana '
```

```
[3] print(name)
```

```
sana
```

```
[4] def add(a,b):  
    print(f"addition of {a} and {b} is {a+b}")
```

```
[5] add(4,2)
```

```
addition of 4 and 2 is 6
```

```
[6] n1=int(input("enter the first number"))
```

```
enter the first number4
```

```
[7] n2=int(input("enter the second number"))
```

```
enter the second number2
```

```
add(n1,n2)
```

```
addition of 4 and 2 is 6
```

```
] print(f"addition of {n1} and {n2} is {n1+n2}")  
print(f"subtraction of {n1} and {n2} is {n1-n2}")  
print(f"multiplication of {n1} and {n2} is {n1*n2}")  
print(f"division of {n1} and {n2} is {n1/n2}")  
print(f"modulus of {n1} and {n2} is {n1%n2}")
```

```
addition of 4 and 2 is 6  
subtraction of 4 and 2 is 2  
multiplication of 4 and 2 is 8  
division of 4 and 2 is 2.0  
modulus of 4 and 2 is 0
```

▼ function definition

```
def funcname(parameterlist):
```


```
    #statements
```


```
funcname(parameterlist)#function call
```

```
[ ] def arithmetic(n1,n2):  
    print(f"addition of {n1} and {n2} is {n1+n2}")  
    print(f"subtraction of {n1} and {n2} is {n1-n2}")  
    print(f"multiplication of {n1} and {n2} is {n1*n2}")  
    print(f"division of {n1} and {n2} is {n1/n2}")  
    print(f"modulus of {n1} and {n2} is {n1%n2}")
```

```
[ ] #function call  
    arithmetic(4,2)
```

```
addition of 4 and 2 is 6  
subtraction of 4 and 2 is 2  
multiplication of 4 and 2 is 8  
division of 4 and 2 is 2.0  
modulus of 4 and 2 is 0
```

```
 n1=int(input('enter first number:'))  
n2=int(input("enter second number:"))
```

```
 enter first number:3  
enter second number:2
```

```
[ ] #function call  
    arithmetic(n1,n2)
```

```
addition of 3 and 2 is 5  
subtraction of 3 and 2 is 1  
multiplication of 3 and 2 is 6  
division of 3 and 2 is 1.5  
modulus of 3 and 2 is 1
```

```
# Python program to find the factorial of a number provided by the user
# using recursion

def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        # recursive call to the function
        return (x * factorial(x-1))

# change the value for a different result
num = 4

# to take input from the user
# num = int(input("Enter a number: "))

# call the factorial function
result = factorial(num)
print("The factorial of", num, "is", result)
```

The factorial of 4 is 24

PROGRAM 2.

A list is an ordered data structure with elements separated by a comma and enclosed within square brackets.

```
[ ]
```

```
[ ] list=[1,2,"apple"]
    print(type(list))
```

```
<class 'list'>
```

```
[ ] list2=[3,4,6]
    #adds the list2 at the end of the first list
    list.append(list2)
    print(list)
    #displays the index of the given element in the list
    print(list.index('apple'))
```

```
[1, 2, 'apple', [3, 4, 6]]
2
```

```
[ ] #removes the last element that has been added to the list
    print(list.pop())
```

```
[3, 4, 6]
```

```
[ ] list.append(2)
    #counts the number of occurrence of the given element
    print(list.count(2))
```

2

```
[ ] fruits=['apple','banana','mango']

    print(fruits)
```

['apple', 'banana', 'mango']

```
[ ] #appends orange to the fruits list
    fruits.append('orange')
```

```
[ ] fruits
```

['apple', 'banana', 'mango', 'orange']

```
[ ] flavor=['vanilla','butter scotch']
    #append flavor list to fruits list
    fruits.append(flavor)
    print(fruits)
```

['apple', 'banana', 'mango', 'orange', ['vanilla', 'butter scotch']]

```
[ ] #reverse the order of elements in the list
    fruits.reverse()
```

```
[ ] fruits
```

[['vanilla', 'butter scotch'], 'orange', 'mango', 'banana', 'apple']

```
[ ] #copies the fruits list into merged
    merged=fruits.copy()
    print(type(merged))
```

<class 'list'>

```
[ ] print(merged)
```

[['vanilla', 'butter scotch'], 'orange', 'mango', 'banana', 'apple']

```
[ ] #remove the specified element from the list
    merged.remove('apple')
```

```
[ ] print(merged)
```

[['vanilla', 'butter scotch'], 'orange', 'mango', 'banana']

```
[ ] flavor
```

['vanilla', 'butter scotch']

```
[ ] #sorts the list element
    flavor.sort()
```

```
[ ] flavor
```

['butter scotch', 'vanilla']

```
[ ] #displays the index of the specified element
    flavor.index('vanilla')
```

1

```
[ ] merged
```

['butter scotch', 'vanilla', 'orange', 'mango', 'banana']

```
[ ] #adds the elements of flavor list to merged list
merged.extend(flavor)
```

merged

```
[ ] [['butter scotch', 'vanilla'],
      'orange',
      'mango',
      'banana',
      'butter scotch',
      'vanilla']
```

Tuples

names=(1,"apple",2,"banana")

```
[ ] names[1]="berry"
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-2-d3120716bab> in <module>
----> 1 names[1]="berry"
```

TypeError: 'tuple' object does not support item assignment

[SEARCH STACK OVERFLOW](#)

```
[ ] #count() counts the number of times the element appears
print(names.count(1))
print(names.count("apple"))
```

```
1
1
```

```
[ ] print(names.index("banana"))#prints the index for a given element
```

```
3
```

```
[ ] thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
```

```
x = thistuple.index(8)
```

```
print(x)
```

```
3
```

PROGRAM 3.**3. Write a Program to implement Water Jug Problem.**

Given two jugs with the maximum capacity of m and n liters respectively. The jugs don't have markings on them which can help us to measure smaller quantities. The task is to measure d liters of water using these two jugs. Hence our goal is to reach from initial state (m, n) to final state (0, d) or (d, 0).

List of Operations you can PERFORM.

1. Fill Jug A Completely
2. Fill Jug B Completely
3. Empty Jug A Completely
4. Empty Jug B Completely
5. Pour From Jug A till Jug B filled Completely or A becomes empty
6. Pour From Jug B till Jug A filled Completely or B becomes empty
7. Pour all From Jug B to Jug A
8. Pour all From Jug A to Jug B

Program

```
a=int(input("Enter Jug A Capacity: "));
b=int(input("Enter Jug B Capacity: "));
ai=int(input("Initially Water in Jug A: "));
bi=int(input("Initially Water in Jug B: "));
af=int(input("Final State of Jug A: "));
bf=int(input("Final State of Jug B: "));
print("List Of Operations You can Do:\n");
print("1.Fill Jug A Completely\n");
print("2.Fill Jug B Completely\n");
print("3.Empty Jug A Completely\n");
print("4.Empty Jug B Completely\n");
print("5.Pour From Jug A till Jug B filled Completely or A becomes empty\n");
print("6.Pour From Jug B till Jug A filled Completely or B becomes empty\n");
print("7.Pour all From Jug B to Jug A\n");
print("8.Pour all From Jug A to Jug B\n");
while ((ai!=af or bi!=bf)):
    op=int(input("Enter the Operation: "));
    if(op==1):
        ai=a;
    elif(op==2):
        bi=b;
    elif(op==3):
        ai=0;
    elif(op==4):
        bi=0;
    elif(op==5):
        if(b-bi>ai):
            bi=ai+bi;
            ai=0;
        else:
            ai=ai-(b-bi);
            bi=b;
    elif(op==6):
        if(a-ai>bi):
            ai=a-bi;
            bi=0;
        else:
            ai=0;
            bi=bi-ai;
    elif(op==7):
        ai=bi;
        bi=0;
    elif(op==8):
        ai=0;
        bi=ai;
```

```
    ai=ai+bi;
    bi=0;
else:
    bi=bi-(a-ai);
    ai=a;
elif(op==7):
```

```
    ai=ai+bi;
    bi=0;
elif(op==8):
    bi=bi+ai;
    ai=0;
print(ai,bi);
```

Output:

```
C:\Users\RDUSER\AILAB>python waterjugg.py
Enter Jug A Capacity: 4
Enter Jug B Capacity: 3
Initially Water in Jug A: 0
Initially Water in Jug B: 0
Final State of Jug A: 0
Final State of Jug B: 2
List Of Operations You can Do:

1.Fill Jug A Completely
2.Fill Jug B Completely
3.Empty Jug A Completely
4.Empty Jug B Completely
5.Pour From Jug A till Jug B filled Completely or A becomes empty
6.Pour From Jug B till Jug A filled Completely or B becomes empty
7.Pour all From Jug B to Jug A
8.Pour all From Jug A to Jug B

Enter the Operation: 1
4 0
Enter the Operation: 5
1 3
Enter the Operation: 3
0 3
Enter the Operation: 6
3 0
Enter the Operation: 2
3 3
Enter the Operation: 6
4 2
Enter the Operation: 3
0 2
```

4. Write a program to implement Uninformed search techniques: a. BFS b. DFS

a.BFS

Breadth-First Search algorithm is a graph traversing technique, where you select a random initial node (source or root node) and start traversing the graph layerwise in such a way that all the nodes and their respective children nodes are visited and explored.

As breadth-first search is the process of traversing each node of the graph, a standard BFS algorithm traverses each vertex of the graph into two parts: 1) Visited 2) Not Visited. So, the purpose of the algorithm is to visit all the vertex while avoiding cycles.

BFS starts from a node, then it checks all the nodes at distance one from the beginning node, then it checks all the nodes at distance two, and so on. So as to recollect the nodes to be visited, BFS uses a queue.

The steps of the algorithm work as follow:

1. Start by putting any one of the graph's vertices at the back of the queue.
2. Now take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add those which are not within the visited list to the rear of the queue.
4. Keep continuing steps two and three till the queue is empty.

BFS pseudocode

The pseudocode for BFS in python goes as below:

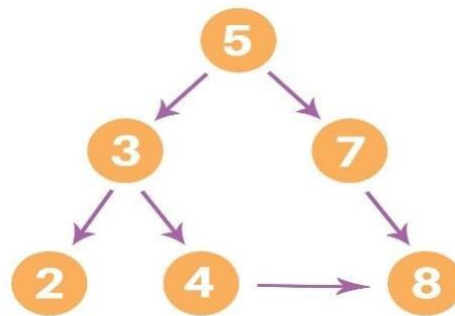
create a queue Q

mark v as visited and put v into Q

while Q is non-empty

 remove the head u of Q

 mark and enqueue all (unvisited) neighbors of u



Program:

```
graph = {  
    '5': ['3','7'],  
    '3': ['2', '4'],  
    '7': ['8'],  
    '2': [],  
    '4': ['8'],  
    '8': []  
}
```

```
visited = [] # List for visited nodes.
```

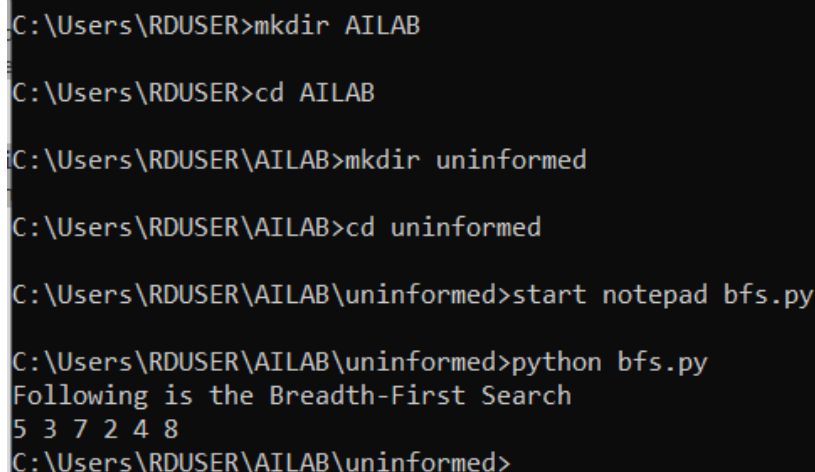
```
queue = [] #Initialize a queue
```

```
def bfs(visited, graph, node): #function for BFS
    visited.append(node)
    queue.append(node)

    while queue:          # Creating loop to visit each node
        m = queue.pop(0)
        print (m, end = " ")

        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')  # function calling
```

Output:

```
C:\Users\RDUSER>mkdir AILAB
C:\Users\RDUSER>cd AILAB
C:\Users\RDUSER\AILAB>mkdir uninformed
C:\Users\RDUSER\AILAB>cd uninformed
C:\Users\RDUSER\AILAB\uninformed>start notepad bfs.py
C:\Users\RDUSER\AILAB\uninformed>python bfs.py
Following is the Breadth-First Search
5 3 7 2 4 8
C:\Users\RDUSER\AILAB\uninformed>
```

DFS

The recursive method of the Depth-First Search algorithm is implemented using stack. A standard Depth-First Search implementation puts every vertex of the graph into one in all 2 categories: 1) Visited 2) Not Visited. The only purpose of this algorithm is to visit all the vertex of the graph avoiding cycles.

The DSF algorithm follows as:

1. We will start by putting any one of the graph's vertex on top of the stack.
2. After that take the top item of the stack and add it to the visited list of the vertex.
3. Next, create a list of that adjacent node of the vertex. Add the ones which aren't in the visited list of vertexes to the top of the stack.
4. Lastly, keep repeating steps 2 and 3 until the stack is empty.

DFS pseudocode

The pseudocode for Depth-First Search in python goes as below: In the init() function, notice that we run the DFS function on every node because many times, a graph may contain two different disconnected part and therefore to make sure that we have visited every vertex, we can also run the DFS algorithm at every node.

```
DFS(G, u)
```

```
    u.visited = true
```

```
    for each v ∈ G.Adj[u]
```

```
        if v.visited == false
```

```
            DFS(G,v)
```

```
init() {
```

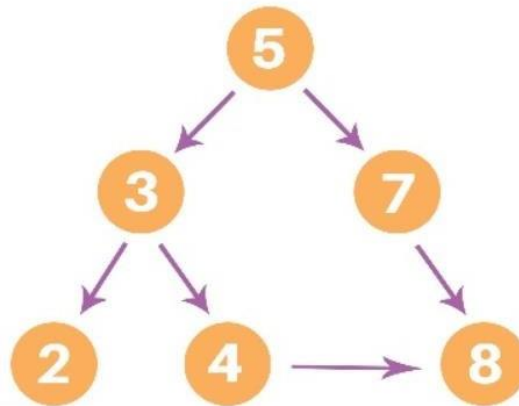
```
    For each u ∈ G
```

```
        u.visited = false
```

```
    For each u ∈ G
```

```
        DFS(G, u)
```

```
}
```

**Program:**

```
# Using a Python dictionary to act as an adjacency list
```

```
graph = {  
    '5': ['3','7'],  
    '3': ['2', '4'],  
    '7': ['8'],  
    '2': [],  
    '4': ['8'],  
    '8': []  
}
```

```
visited = set() # Set to keep track of visited nodes of graph.
```

```
def dfs(visited, graph, node): #function for dfs
```

```
    if node not in visited:
```

```
        print (node)
```

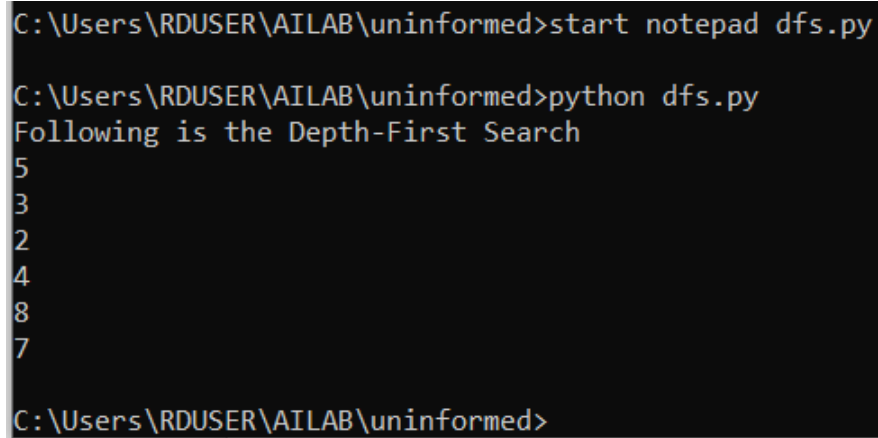
```
        visited.add(node)
```

```
        for neighbour in graph[node]:
```

```
            dfs(visited, graph, neighbour)
```

```
# Code
```

```
print("Following is the Depth-First Search")  
dfs(visited, graph, '5')
```

Output

```
C:\Users\RDUSER\AILAB\uninformed>start notepad dfs.py  
  
C:\Users\RDUSER\AILAB\uninformed>python dfs.py  
Following is the Depth-First Search  
5  
3  
2  
4  
8  
7  
  
C:\Users\RDUSER\AILAB\uninformed>
```

PROGRAM 5.**5. Write a program to implement Informed search techniques a. Greedy Best first search b. A* algorithm**

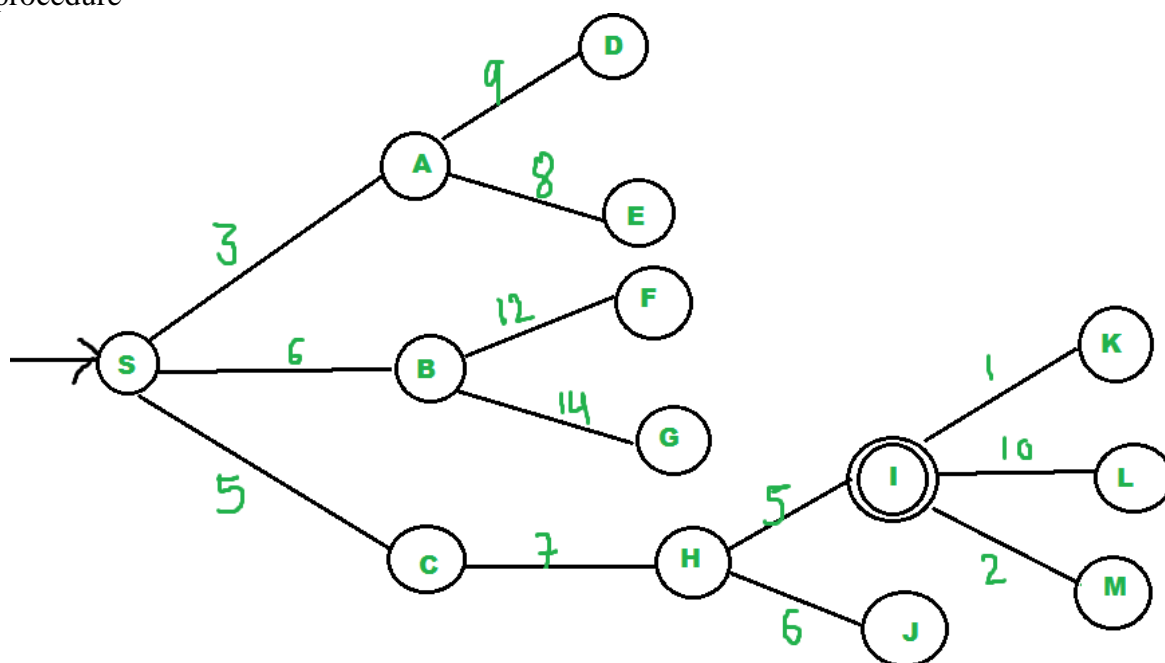
The greedy best-first search algorithm tries to explore the node that is closest to the goal. This algorithm evaluates nodes by using the heuristic function $h(n)$, that is, the evaluation function is equal to the heuristic function, $f(n) = h(n)$. This equivalency is what makes the search algorithm 'greedy.'

// Pseudocode for Best First Search

Best-First-Search(Graph g, Node start)

- Create an empty
PriorityQueue
PriorityQueue pq;
- Insert
"start" in
pq.
pq.insert(
start)
- Until PriorityQueue is empty
u = PriorityQueue.DeleteMin
If u is the goal
Exit
Else
Foreach neighbor v of u
If v "Unvisited"
Mark v "Visited"
pq.insert(v)
Mark u "Examined"

End procedure



- We start from source "S" and search for goal "I" using given costs and Best First search.
- pq initially contains S
 - We remove s from and process unvisited neighbors of S to pq.
 - pq now contains {A, C, B} (C is put before B because C has lesser cost)
- We remove A from pq and process unvisited neighbors of A to pq.

- pq now contains {C, B, E, D}
- We remove C from pq and process unvisited neighbors of C to pq.
 - pq now contains {B, H, E, D}
- We remove B from pq and process unvisited neighbors of B to pq.
 - pq now contains {H, E, D, F, G}
- We remove H from pq.
- Since our goal "I" is a neighbor of H, we return.

a. Greedy Best first search

```
from queue import PriorityQueue
```

```
v = 14
```

```
graph = [[] for i in range(v)]
```

```
# Function For Implementing Best First Search
```

```
# Gives output path having lowest cost
```

```
def best_first_search(actual_Src, target, n):
```

```
    visited = [False] * n
```

```
    pq = PriorityQueue()
```

```
    pq.put((0, actual_Src))
```

```
    visited[actual_Src] = True
```

```
    while pq.empty() == False:
```

```
        u = pq.get()[1]
```

```
        # Displaying the path having lowest cost
```

```
        print(u, end=" ")
```

```
        if u == target:
```

```
            break
```

```
        for v, c in graph[u]:
```

```
            if visited[v] == False:
```

```
                visited[v] = True
```

```
                pq.put((c, v))
```

```
    print()
```

```
# Function for adding edges to graph
```

```
def addedge(x, y, cost):
```

```
    graph[x].append((y, cost))
```

```
    graph[y].append((x, cost))
```

```
# The nodes shown in above example(by alphabets) are
```

```
# implemented using integers addedge(x,y,cost);
```

```
adddedge(0, 1, 3)
```

```
adddedge(0, 2, 6)
```

```
adddedge(0, 3, 5)
```

```
adddedge(1, 4, 9)
```

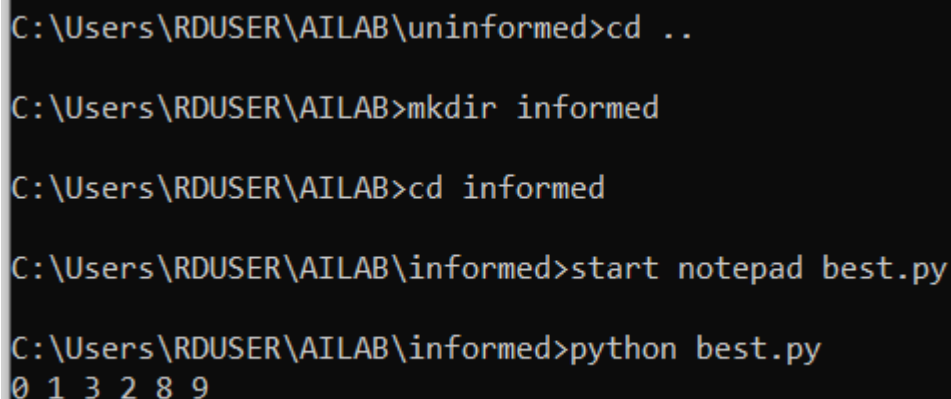
```
adddedge(1, 5, 8)
```

```
adddedge(2, 6, 12)
```

```
adddedge(2, 7, 14)
```

```
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)
```

```
source = 0
target = 9
best_first_search(source, target, v)
```

Output:

```
C:\Users\RDUSER\AILAB\uninformed>cd ..
C:\Users\RDUSER\AILAB>mkdir informed
C:\Users\RDUSER\AILAB>cd informed
C:\Users\RDUSER\AILAB\informed>start notepad best.py
C:\Users\RDUSER\AILAB\informed>python best.py
0 1 3 2 8 9
```

b. A* algorithm

An informed search strategy-one that uses problem-specific knowledge-can find solutions more efficiently. A key component of these algorithms is a heuristic function $h(n)$
 $h(n)$ = estimated cost of the cheapest path from node n to a goal node.

Admissible /heuristic never over estimated i.e. $h(n) \leq$ Actual cost. For example, Distance between two nodes(cities) \Rightarrow straight line distance and for 8-puzzel problem- Admissible heuristic can be number of misplaced tiles $h(n)= 8$.

A* Search technique

It is informed search technique. It uses additional information beyond problem formulation and tree. Search is based on Evaluation function $f(n)$. Evaluation function is based on both heuristic function $h(n)$ and $g(n)$.

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

It uses two queues for its implementation: open, close Queue. Open queue is a priority queue which is arranged in ascending order of $f(n)$

Algorithm:

1. Create a single member queue comprising of Root node
2. If FIRST member of queue is goal then goto step 5
3. If first member of queue is not goal then remove it from queue and add to close queue.
4. Consider its children if any, and add them to queue in ascending order of evaluation function $f(n)$.
5. If queue is not empty then goto step 2.
6. If queue is empty then goto step 6
7. Print 'success' and stop
8. Print 'failure' and stop.

Performance Comparison:

- Completeness: yes
 - Optimality: yes
-

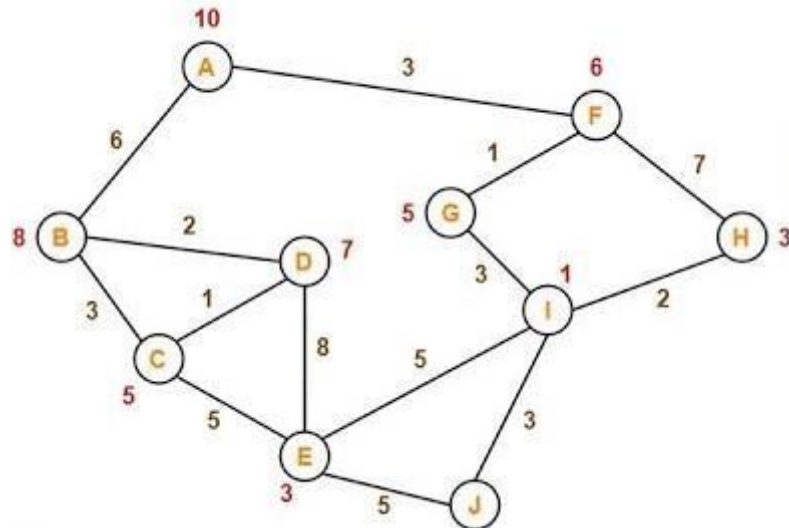
Limitation:

- It generate same node again and again
- Large Memory is required

Observation

Although A* generate many nodes it never uses those nodes for which $f(n) > c^*$ where c^* is optimum cost.

Consider an example as below

**Program:**

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}          #store distance from starting node
    parents = {}    # parents contains an adjacency map of all nodes
    #distance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        #node with lowest f() is found
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                #nodes 'm' not in first and last set are added to first
                #n is set its parent
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                #for each node m,compare its distance from start i.e g(m)
                #from start through n node
            else:
```

```
        if g[m] > g[n] + weight:
            #update g(m)
            g[m] = g[n] + weight
            #change parent of m to n
            parents[m] = n
            #if m in closed set,remove and add to open
            if m in closed_set:
                closed_set.remove(m)
                open_set.add(m)
    if n == None:
        print('Path does not exist!')
        return None

    # if the current node is the stop_node
    # then we begin reconstructin the path from it to the start_node
    if n == stop_node:
        path = []
        while parents[n] != n:
            path.append(n)
            n = parents[n]
        path.append(start_node)
        path.reverse()
        print('Path found: {}'.format(path))
        return path
    # remove n from the open_list, and add it to closed_list
    # because all of his neighbors were inspected
    open_set.remove(n)
    closed_set.add(n)
    print('Path does not exist!')
    return None

#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
```

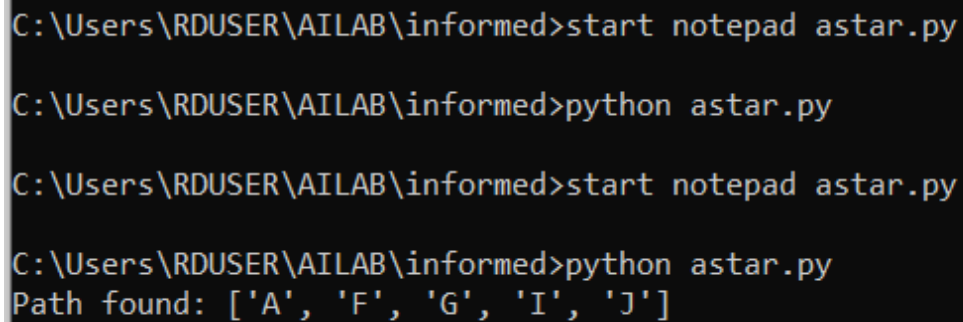
```
return H_dist[n]
```

```
#Describe your graph here
```

```
Graph_nodes = {  
    'A': [('B', 6), ('F', 3)],  
    'B': [('A', 6), ('C', 3), ('D', 2)],  
    'C': [('B', 3), ('D', 1), ('E', 5)],  
    'D': [('B', 2), ('C', 1), ('E', 8)],  
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],  
    'F': [('A', 3), ('G', 1), ('H', 7)],  
    'G': [('F', 1), ('I', 3)],  
    'H': [('F', 7), ('I', 2)],  
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],  
}
```

```
aStarAlgo('A', 'J')
```

Output:



```
C:\Users\RDUSER\AILAB\informed>start notepad astar.py  
  
C:\Users\RDUSER\AILAB\informed>python astar.py  
  
C:\Users\RDUSER\AILAB\informed>start notepad astar.py  
  
C:\Users\RDUSER\AILAB\informed>python astar.py  
Path found: ['A', 'F', 'G', 'I', 'J']
```

PROGRAM 6.**Study of Prolog, its facts, and rules.**

Prolog is a general purpose logic programming language associated with artificial intelligence and computational linguistics.

Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is declarative: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations. Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching databases, voice control systems, and filling templates.

Rules and facts:

Prolog programs describe relations, defined by means of clauses. Pure Prolog is restricted to Horn clauses. There are two types of clauses: facts and rules. A rule is of the form

Head:- Body.

and is read as "Head is true if Body is true". A rule's body consists of calls to predicates, which are called the rule's goals. The built-in predicate, `/2` (meaning a 2-arity operator with name,) denotes conjunction of goals, and `;/2` denotes disjunction. Conjunctions and disjunctions can only appear in the body, not in the head of a rule.

Clauses with empty bodies are called facts. An example of a fact is:

`cat (tom).`

which is equivalent to the rule:

`cat(tom) :- true.`

The built-in predicate `true/0` is always true.

Given the above fact, one can ask:

is tom a cat?

?- `cat(tom).`

Yes

what things are cats?

?- `cat(X).`

`X = tom`

Clauses with bodies are called rules. An example of a rule is:

`animal(X) :- cat(X).`

If we add that rule and ask what things are animals?

?- `animal(X).`

`X = tom`

Prolog must be able to handle arithmetic in order to be a useful general purpose programming language. However, arithmetic does not fit nicely into the logical scheme of things. That is, the concept of evaluating an arithmetic expression is in contrast to the straight pattern matching we have seen so far. For this reason, Prolog provides the built-in predicate 'is' that evaluates arithmetic expressions. Its syntax calls for the use of operators.

X is <arithmetic expression>

The variable X is set to the value of the arithmetic expression. On backtracking it is unassigned.

The variable X is set to the value of the arithmetic expression. On backtracking it is unassigned.

The arithmetic expression looks like an arithmetic expression in any other programming language.

Here is how to use Prolog as a calculator.

?- X is 2 + 2.

X = 4

?- X is 3 * 4 + 2.

X = 14

Parentheses clarify precedence.

?- X is 3 * (4 + 2).

X = 18

?- X is (8 / 4) / 2.

X = 1

In addition to 'is,' Prolog provides a number of operators that compare two numbers. These include 'greater than', 'less than', 'greater or equal than', and 'less or equal than.' They behave more logically, and succeed or fail according to whether the comparison is true or false. Notice the order of the symbols in the greater or equal than and less than or equal operators. They are specifically constructed not to look like an arrow, so that the use arrow symbols in programs is without confusion.

X > Y

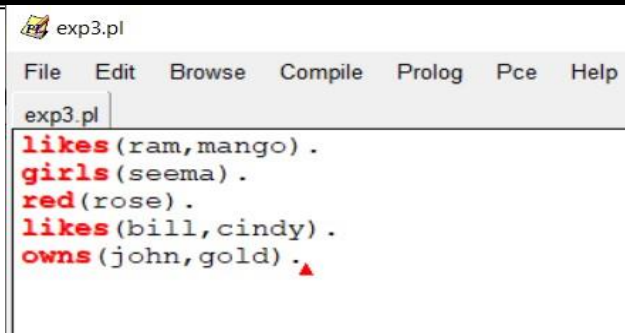
X < Y

X >= Y

X <= Y

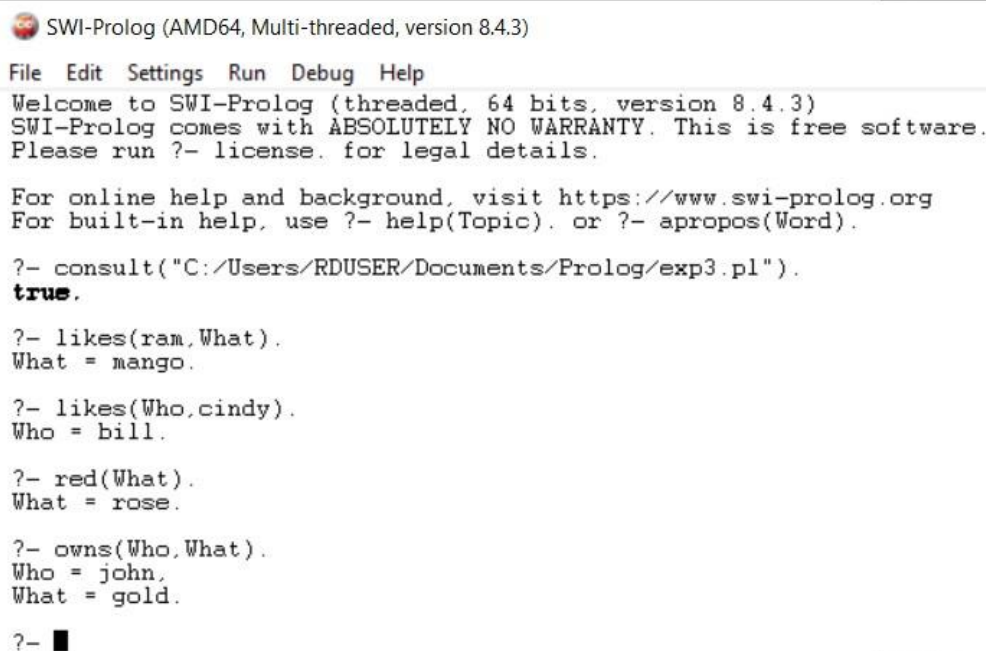
a. Write simple facts for the statements and querying it

- **Ram likes mango.**
 - **Seema is a girl.**
 - **Bill likes Cindy.**
 - **Rose is red.**
 - **John owns gold.**
-



```
exp3.pl
File Edit Browse Compile Prolog Pce Help
exp3.pl
likes(ram,mango) .
girls(seema) .
red(rose) .
likes(bill,cindy) .
owns(john,gold) .
```

Output:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult("C:/Users/RDUSER/Documents/Prolog/exp3.pl").
true.

?- likes(ram,What).
What = mango.

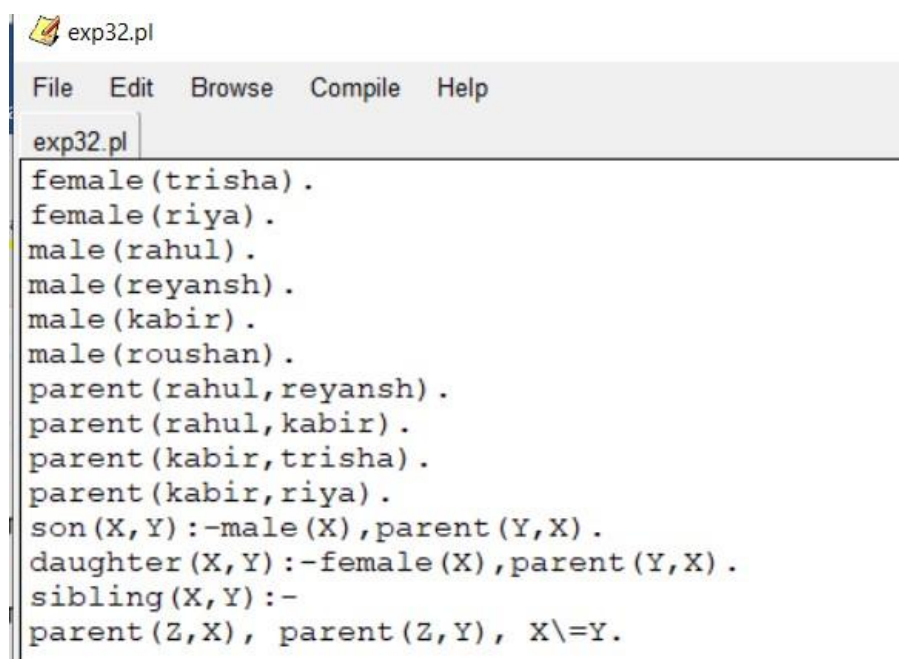
?- likes(Who,cindy).
Who = bill.

?- red(What).
What = rose.

?- owns(Who,What).
Who = john,
What = gold.

?-
```

b. Write a program for Family-tree.



```
exp32.pl
File Edit Browse Compile Help
exp32.pl
female(trisha) .
female(riya) .
male(rahul) .
male(reyansh) .
male(kabir) .
male(roushan) .
parent(rahul,reyansh) .
parent(rahul,kabir) .
parent(kabir,trisha) .
parent(kabir,riya) .
son(X,Y):-male(X),parent(Y,X) .
daughter(X,Y):-female(X),parent(Y,X) .
sibling(X,Y):-
parent(Z,X), parent(Z,Y), X\=Y.
```

Output:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
?- consult("C:/Users/RDUSER/Documents/Prolog/exp32.pl").
true.
?- son(reyansh,rahul).
true.
?- son(riya,reyansh).
false.
?- daughter(riya,kabir).
true.
?- daughter(roushan,kabir).
false.
?- sibling(reyansh,kabir).
true.
?-
```

PROGRAM 7.

Write a python program to implement the methods of numpy.

```
import numpy as np

[ ] pip install numpy

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (1.21.6)

[ ] arr = np.array([1, 2, 3, 4, 5])

[ ] print(arr)

[1 2 3 4 5]

[ ] print(type(arr))

<class 'numpy.ndarray'>

[ ] print(np.__version__)

1.21.6
```

Data types and attributes

[+ Code](#)[+ Text](#)

```
[ ] #numpy main datatype is ndarray
a1=np.array([1,2,3])
a1

array([1, 2, 3])

[ ] type(a1)

numpy.ndarray

[ ] a2=np.array([[1,2,3.3],[4,5,6.5]])

a3=np.array([[1,2,3],
            [4,5,6],
            [7,8,9]],
            [[10,11,12],
            [13,14,15],
            [16,17,18]])

[ ] a2.shape
```

```
(2, 3)
```

```
[ ] a2
```

```
array([[1. , 2. , 3.3],  
       [4. , 5. , 6.5]])
```

```
[ ] a3
```

```
array([[ 1,  2,  3],  
       [ 4,  5,  6],  
       [ 7,  8,  9]],  
      [[10, 11, 12],  
       [13, 14, 15],  
       [16, 17, 18]])
```

```
[ ] #shows number of rows and columns  
a1.shape
```

```
(3,)
```

```
[ ] a2.shape
```

```
(2, 3)
```

```
[ ]  
a3.shape
```

```
(2, 3, 3)
```

```
[ ] #number of dimensions in array  
a1.ndim,a2.ndim,a3.ndim
```

```
(1, 2, 3)
```

```
[ ] #data type  
a1.dtype,a2.dtype,a3.dtype
```

```
(dtype('int64'), dtype('float64'), dtype('int64'))
```

```
[ ] #displays the total number of elements  
a1.size,a2.size,a3.size
```

```
(3, 6, 18)
```

```
[ ] type(a1),type(a2),type(a3)
```

```
(numpy.ndarray, numpy.ndarray, numpy.ndarray)
```

PROGRAM 8.

8. Write a python program to create the following using pandas

- a. create a dataframe from the dictionary**
- b. List the top and bottom 10 rows from the dataframe**
- c. Display the dimensions**
- d. Access the data at index 3**

```
import pandas as pd
```

```
[ ] #2 data types in pandas
series=pd.Series(["BMW","Toyota","Honda"])
series
```

```
0      BMW
1    Toyota
2     Honda
dtype: object
```

```
[ ] #series--1 dimensional
```

```
[ ] colors=pd.Series(["Red","Blue","White"])
```

```
[ ] colors
```

```
0      Red
1     Blue
2    White
dtype: object
```

```
[ ] car_data=pd.DataFrame({"Car make":series,"color":colors})
    car_data
```

	Car make	color
0	BMW	Red
1	Toyota	Blue
2	Honda	White

```
[ ] #import data
    car_sales=pd.read_csv("car-sales.csv")
```

```
[ ] car_sales
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	\$4,000.00
1	Honda	Red	87899	4	\$5,000.00
2	Toyota	Blue	32549	3	\$7,000.00
3	BMW	Black	11179	5	\$22,000.00

```
car_sales.head(10)
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	\$4,000.00
1	Honda	Red	87899	4	\$5,000.00
2	Toyota	Blue	32549	3	\$7,000.00
3	BMW	Black	11179	5	\$22,000.00
4	Nissan	White	213095	4	\$3,500.00
5	Toyota	Green	99213	4	\$4,500.00
6	Honda	Blue	45698	4	\$7,500.00
7	Honda	Blue	54738	4	\$7,000.00
8	Toyota	White	60000	4	\$6,250.00
9	Nissan	White	31600	4	\$9,700.00



```
car_sales.tail(10)
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	\$4,000.00
1	Honda	Red	87899	4	\$5,000.00
2	Toyota	Blue	32549	3	\$7,000.00
3	BMW	Black	11179	5	\$22,000.00
4	Nissan	White	213095	4	\$3,500.00
5	Toyota	Green	99213	4	\$4,500.00
6	Honda	Blue	45698	4	\$7,500.00
7	Honda	Blue	54738	4	\$7,000.00
8	Toyota	White	60000	4	\$6,250.00
9	Nissan	White	31600	4	\$9,700.00



```
car_sales.loc[3]
```

```
Make          BMW
Colour        Black
Odometer (KM)  11179
Doors          5
Price         $22,000.00
Name: 3, dtype: object
```

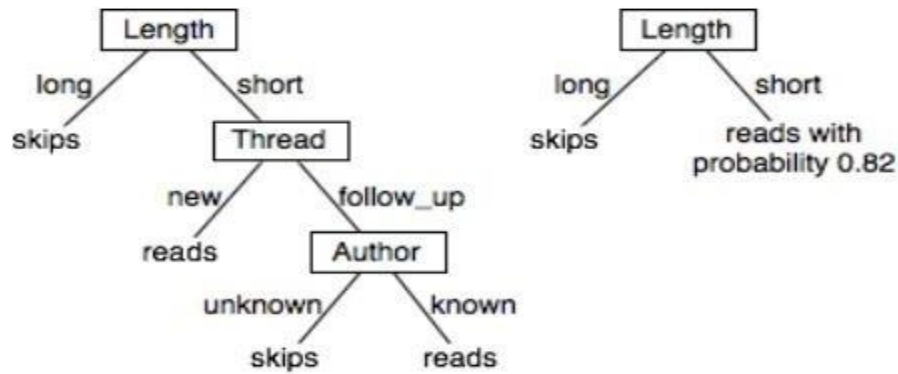
PROGRAM 9.

9. Write a program to train and validate the following classifiers for given data (scikit-learn): a. Decision Tree b. Multi-layer Feed Forward neural network

A decision tree is a simple representation for classifying examples. Decision tree learning is one of the most successful techniques for supervised classification learning. For this section, assume that all of the features have finite discrete domains, and there is a single target feature called the classification. Each element of the domain of the classification is called a class.

A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labelled with an input feature. The arcs coming from a node labeled with a feature are labeled with each of the possible values of the feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes.

To classify an example, filter it down the tree, as follows. For each feature encountered in the tree, the arc corresponding to the value of the example for that feature is followed. When a leaf is reached, the classification corresponding to that leaf is returned.



Decision Tree

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn import tree
```

```
iris=load_iris()
```

```
print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```
#convert ndarray to dataframe
irisdata=iris.data
colnam=iris.feature_names
df=pd.DataFrame(data=irisdata,columns=colnam)
```

```
#export to csvfile
df.to_csv("iriss.csv",index=False)
```

Now, load the iris dataset

To see all the features in the dataset, use the print function

To see all the target names in the dataset-

```
[6] print(iris.target_names)

['setosa' 'versicolor' 'virginica']
```

Remove the Labels

```
# This is formatted as code
```

Now, we will remove the elements in the 0th, 50th, and 100th position. 0th element belongs to the Setosa species, 50th belongs Versicolor species and the 100th belongs to the Virginica species.

This will remove the labels for us to train our decision tree classifier better and check if it is able to classify the data well.

```
[7] #Spilitting the dataset 1 way
removed =[0,50,100]
new_target = np.delete(iris.target,removed)
new_data = np.delete(iris.data,removed, axis=0)
```

Train the Decision Tree Classifier

```
[8] #read from csvfile
dff=pd.read_csv("/content/iriss.csv")
```

```
[9] #assign data and column
X=dff
Y=iris.target
```

```
[10] #train test split
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,
        test_size = 0.4,
        random_state = 42)
```

```
[11] #train classifier
clf = tree.DecisionTreeClassifier(criterion="entropy") # defining decision tree classifier
clf=clf.fit(X_train,Y_train) # train data on new data and new target
prediction = clf.predict(X_test) # assign removed data as input
```

```
[12] from sklearn.metrics import accuracy_score
```

we check if our predicted labels match the original labels

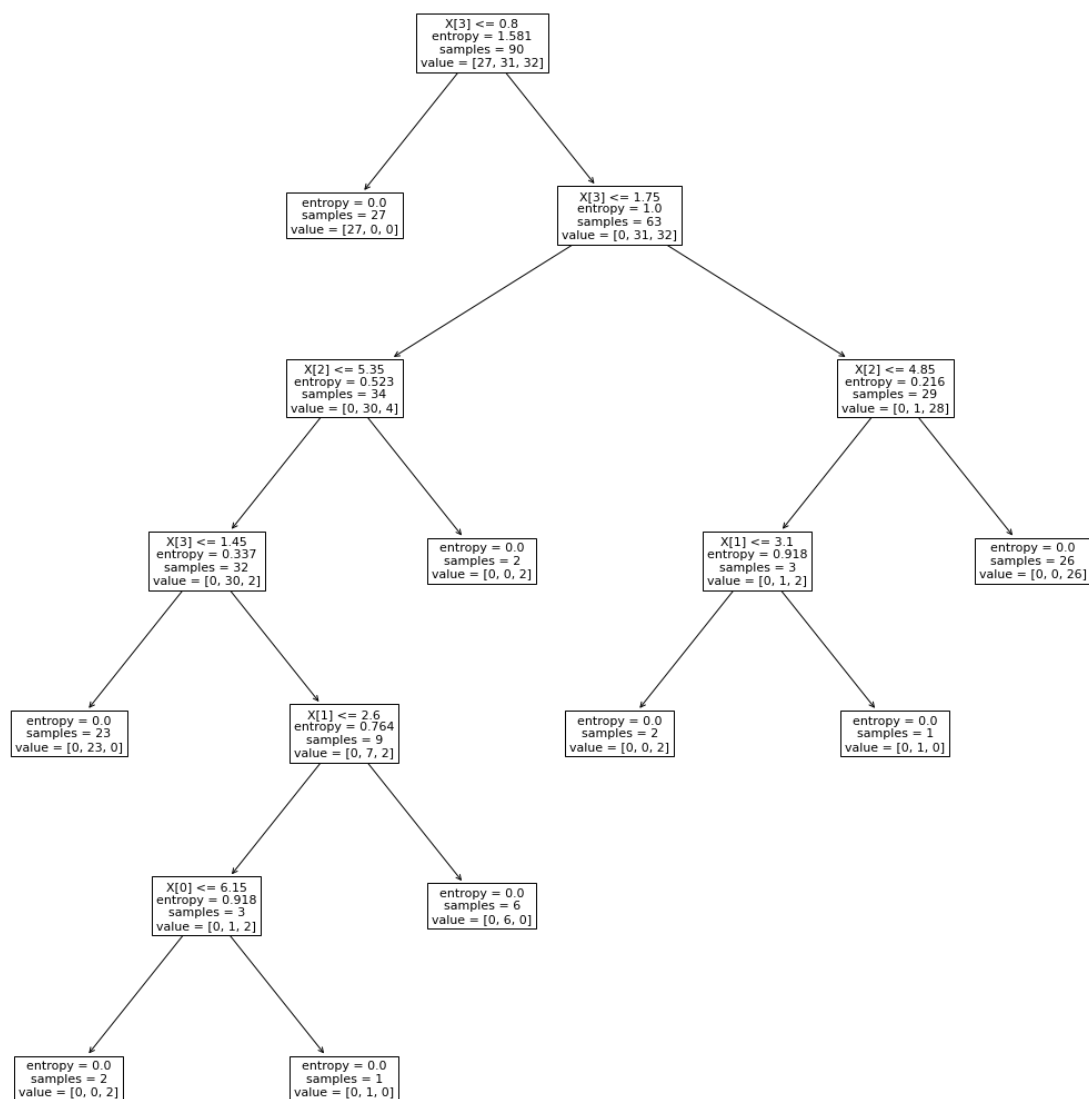
```
[13] #print("Original Labels",X_test)
      #print("Labels Predicted",prediction)

      print("Train data accuracy:",accuracy_score(y_true = Y_train, y_pred=clf.predict(X_train)))
      print("Test data accuracy:",accuracy_score(y_true = Y_test, y_pred=prediction))

      Train data accuracy: 1.0
      Test data accuracy: 0.9833333333333333
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(20,20))
tree.plot_tree(clf,fontsize=11)
```

```
[Text(0.4444444444444444, 0.9285714285714286, 'X[3] <= 0.8\nentropy = 1.581\nsamples = 90\nvalue = [27, 31, 32]',
Text(0.3333333333333333, 0.7857142857142857, 'entropy = 0.0\nsamples = 27\nvalue = [27, 0, 0]'),
Text(0.5555555555555556, 0.7857142857142857, 'X[3] <= 1.75\nentropy = 1.0\nsamples = 63\nvalue = [0, 31, 32]'),
Text(0.3333333333333333, 0.6428571428571429, 'X[2] <= 5.35\nentropy = 0.523\nsamples = 34\nvalue = [0, 30, 4]',
Text(0.2222222222222222, 0.5, 'X[3] <= 1.45\nentropy = 0.337\nsamples = 32\nvalue = [0, 30, 2]'),
Text(0.1111111111111111, 0.35714285714285715, 'entropy = 0.0\nsamples = 23\nvalue = [0, 23, 0]'),
Text(0.3333333333333333, 0.35714285714285715, 'X[1] <= 2.6\nentropy = 0.764\nsamples = 9\nvalue = [0, 7, 2]'),
Text(0.2222222222222222, 0.21428571428571427, 'X[0] <= 6.15\nentropy = 0.918\nsamples = 3\nvalue = [0, 1, 2]',
Text(0.1111111111111111, 0.07142857142857142, 'entropy = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
Text(0.3333333333333333, 0.07142857142857142, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
Text(0.4444444444444444, 0.21428571428571427, 'entropy = 0.0\nsamples = 6\nvalue = [0, 6, 0]'),
Text(0.4444444444444444, 0.5, 'entropy = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
Text(0.7777777777777778, 0.6428571428571429, 'X[2] <= 4.85\nentropy = 0.216\nsamples = 29\nvalue = [0, 1, 28]',
Text(0.6666666666666666, 0.5, 'X[1] <= 3.1\nentropy = 0.918\nsamples = 3\nvalue = [0, 1, 2]'),
Text(0.5555555555555556, 0.35714285714285715, 'entropy = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
Text(0.7777777777777778, 0.35714285714285715, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
Text(0.8888888888888888, 0.5, 'entropy = 0.0\nsamples = 26\nvalue = [0, 0, 26]')]
```



- **Multi-layer Feed Forward neural network**

a simple Multi Layer Network with two hidden layers and one output layer. Each and every link in the network will be associated with a weight. We will initialise those weights and create the architecture using the below snippet of code.

Importing required modules into our notebook

```
import numpy as np
import pandas as pd

# Load data
data=pd.read_csv('/content/HR_comma_sep.csv')

data.head()
```

	satisfaction_level	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Departments	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
1	0.80	0.86	5	262	6	0	1	0	sales	medium
2	0.11	0.88	7	272	4	0	1	0	sales	medium
3	0.72	0.87	5	223	5	0	1	0	sales	low
4	0.37	0.52	2	159	3	0	1	0	sales	low

```
data['Departments']
```

```
0      sales
1      sales
2      sales
3      sales
4      sales
...
14994  support
14995  support
14996  support
14997  support
14998  support
Name: Departments, Length: 14999, dtype: object
```

```
# Import LabelEncoder
from sklearn import preprocessing

# Creating labelEncoder
le = preprocessing.LabelEncoder()
print(le)

# Converting string labels into numbers.
data['salary']=le.fit_transform(data['salary'])
data['Departments']=le.fit_transform(data['Departments'])
```

```
LabelEncoder()
```

```
data['salary']
]

0      1
1      2
2      2
3      1
4      1
..
14994   1
14995   1
14996   1
14997   1
14998   1
Name: salary, Length: 14999, dtype: int64

]

# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42) # 70% training and 30% test

# Import MLPClassifier
from sklearn.neural_network import MLPClassifier

# Create model object
clf = MLPClassifier(hidden_layer_sizes=(6,5),
                    random_state=5,
                    verbose=True,
                    learning_rate_init=0.01)

# Fit data onto the model
clf.fit(X_train,y_train)

Iteration 1, loss = 0.61512605
Iteration 2, loss = 0.57545658
Iteration 3, loss = 0.55823146
Iteration 4, loss = 0.53011644
Iteration 5, loss = 0.50549749
Iteration 6, loss = 0.48004244
Iteration 7, loss = 0.47915513
Iteration 8, loss = 0.46239153
Iteration 9, loss = 0.47441120
Iteration 10, loss = 0.46241650
Iteration 11, loss = 0.45068143
Iteration 12, loss = 0.45071101
Iteration 13, loss = 0.45213613
Iteration 14, loss = 0.46049483
Iteration 15, loss = 0.45897398
Iteration 16, loss = 0.46170601
Iteration 17, loss = 0.45527116
Iteration 18, loss = 0.44996595
Iteration 19, loss = 0.44982305
Iteration 20, loss = 0.45384764
Iteration 21, loss = 0.46981282
Iteration 22, loss = 0.45010489
Iteration 23, loss = 0.46852413
Iteration 24, loss = 0.45242336

# Make prediction on test dataset
ypred=clf.predict(X_test)

# Import accuracy score
from sklearn.metrics import accuracy_score

# Calculate accuracy
accuracy_score(y_test,ypred)

0.9386666666666666
```

10. Text processing using NLTK

- **Remove stop words**
 - **Implement stemming**
 - **POS (Parts of Speech) tagging**
- Introduction:**

NLTK is a toolkit build for working with NLP in Python. It provides us various text processing libraries with a lot of test datasets. A variety of tasks can be performed using NLTK such as tokenizing, parse tree visualization, etc.

Text Processing steps :

- Tokenization
- Lower case conversion
- Stop Words removal
- Stemming
- Lemmatization
- Parse tree or Syntax Tree generation
- POS Tagging

Tokenization:

The breaking down of text into smaller units is called tokens. tokens are a small part of that text. If we have a sentence, the idea is to separate each word and build a vocabulary such that we can represent all words uniquely in a list. Numbers, words, etc.. all fall under tokens.

Lower case conversion:

We want our model to not get confused by seeing the same word with different cases like one starting with capital and one without and interpret both differently. So we convert all words into the lower case to avoid redundancy in the token list.

Stop Words removal:

When we use the features from a text to model, we will encounter a lot of noise. These are the stop words like the, he, her, etc... which don't help us and, just be removed before processing for cleaner processing inside the model. With NLTK we can see all the stop words available in the English language.

Stemming:

In our text we may find many words like playing, played, playfully, etc... which have a root word, play all of these convey the same meaning. So we can just extract the root word and remove the rest. Here the root word formed is called 'stem' and it is not necessarily that stem needs to exist and have a meaning. Just by committing the suffix and prefix, we generate the stems.

Lemmatization:

We want to extract the base form of the word here. The word extracted here is called Lemma and it is available in the dictionary. We have the WordNet corpus and the lemma generated will be available in this corpus. NLTK provides us with the WordNet Lemmatizer that makes use of the WordNet Database to lookup lemmas of words.

Stemming is much faster than lemmatization as it doesn't need to lookup in the dictionary and just follows the algorithm to generate the root words.

Parse tree or Syntax Tree generation :

We can define grammar and then use NLTK RegexpParser to extract all parts of speech from the sentence and draw functions to visualize it.

POS Tagging:

Part of Speech tagging is used in text processing to avoid confusion between two same words that have different meanings. With respect to the definition and context, we give each word a particular tag and process them. Two Steps are used here:

- Tokenize text (word_tokenize).
- Apply the pos_tag from NLTK to the above step.

Program

```
import nltk
```

```

nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('omw-1.4')
text="""Hello Mr. Smith, how are you doing today? The weather is great, and city is awesome.The sky is pinkish-blue. You shouldn't eat cardboard"""

```

```

import re
text = re.sub(r"[^a-zA-Z0-9]", " ", text.lower())
words = text.split()
print(words)

['hello', 'mr', 'smith', 'how', 'are', 'you', 'doing', 'today', 'the', 'weather', 'is', 'great', 'and', 'city', 'is', 'awesome', 'the', 'sky', 'is', 'pinkish', 'blue', 'you', 'shouldn', 't', 'eat', 'cardboard']

```

```

from nltk.corpus import stopwords

```

```

print(stopwords.words("english"))

```

```

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

```

```

words = [w for w in words if w not in stopwords.words("english")]

```

```

print(words)

```

```

['hello', 'mr', 'smith', 'today', 'weather', 'great', 'city', 'awesome', 'sky', 'pinkish', 'blue', 'eat', 'cardboard']

```

```

from nltk.stem.porter import PorterStemmer

```

```

# Reduce words to their stems

```

```

stemmed = [PorterStemmer().stem(w) for w in words]

```

```

print(stemmed)

```

```

['hello', 'mr', 'smith', 'today', 'weather', 'great', 'city', 'awesome', 'sky', 'pinkish', 'blue', 'eat', 'cardboard']

```

```

from nltk.stem.wordnet import WordNetLemmatizer

```

```

# Reduce words to their root form

```

```

lemmed = [WordNetLemmatizer().lemmatize(w) for w in words]

```

```

print(lemmed)

```

```

['hello', 'mr', 'smith', 'today', 'weather', 'great', 'city', 'awesome', 'sky', 'pinkish', 'blue', 'eat', 'cardboard'] #

```

```

Import required libraries

```

```

import nltk

```

```

nltk.download('punkt')

```

```

nltk.download('averaged_perceptron_tagger')

```

```

from nltk import pos_tag, word_tokenize, RegexpParser

```

```

# Example text

```

```

sample_text = "The quick brown fox jumps over the lazy dog"

```

```

# Find all parts of speech in above sentence

```

```

tagged = pos_tag(word_tokenize(sample_text))

```


#Extract all parts of speech from any text

chunker = RegexpParser("""

NP: {} #To extract Noun Phrases

P: {} #To extract Prepositions

V: {} #To extract Verbs

PP: {} #To extract Prepositional Phrases

VP: {} #To extract Verb Phrases """)

Print all parts of speech in above sentence

output = chunker.parse(tagged)

print("After Extracting", output)

```
[nltk_data] Downloading package punkt to
[nltk_data]   C:\Users\RDUSER\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]   C:\Users\RDUSER\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]   date!
```

After Extracting (S

The/DT

quick/JJ

brown/NN

fox/NN

jumps/VBZ

over/IN

the/DT

lazy/JJ

dog/NN)

output.draw()

import nltk

from nltk.corpus import stopwords

nltk.download('punkt')

nltk.download('averaged_perceptron_tagger')

from nltk.tokenize import word_tokenize, sent_tokenize

stop_words = set(stopwords.words('english'))

txt = "Natural language processing is an exciting area."

#"Huge budget have been allocated for this."

sent_tokenize is one of instances of

PunktSentenceTokenizer from the nltk.tokenize.punkt module

tokenized = sent_tokenize(txt)

for i in tokenized:

Word tokenizers is used to find the words

and punctuation in a string

wordsList = nltk.word_tokenize(i)

removing stop words from wordsList

wordsList = [w for w in wordsList if not w in stop_words]

Using a Tagger. Which is part-of-speech

tagger or POS-tagger.

tagged = nltk.pos_tag(wordsList)

print(tagged)


```
[('Natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'), ('exciting', 'JJ'), ('area', 'NN'), ('.', '.')]
```

```
[nltk_data] Downloading package punkt to  
[nltk_data]   C:\Users\RDUSER\AppData\Roaming\nltk_data...  
[nltk_data]   Package punkt is already up-to-date!  
[nltk_data] Downloading package averaged_perceptron_tagger to  
[nltk_data]   C:\Users\RDUSER\AppData\Roaming\nltk_data...  
[nltk_data]   Package averaged_perceptron_tagger is already up-to-  
[nltk_data]   date!
```