# PRACTICAL NO:-01

**AIM:** WRITE A PROGRAM TO DEMONSTRATE BITWISE OPERATION.

**CODE:**

plays={"Anthony and Cleopatra":"Anthony is there, Brutus is Caeser is with Cleopatra mercy worser.",

"Julius Ceaser":"Anthony is there, Brutus is Caeser is but Calpurnia is.",

"The Tempest":"mercy worser","Hamlet":"Caeser and Brutus are present with mercy and worser",

"Othello":"Caeser is present with mercy and worser","Macbeth":"Anthony is there, Caeser, mercy."} words=["Anthony","Brutus","Caeser","Calpurnia","Cleopatra","mercy","worser"] vector\_matrix=[[0 for i in range(len(plays))] for j in range(len(words))]

text\_list=list((plays.values())) for i in range(len(words)):

for j in range(len(text\_list)):

if words[i] in text\_list[j]: vector\_matrix[i][j]=1

else:

vector\_matrix[i][j]=0

for i in vector\_matrix: print(i)

result=[]

string\_list=[]

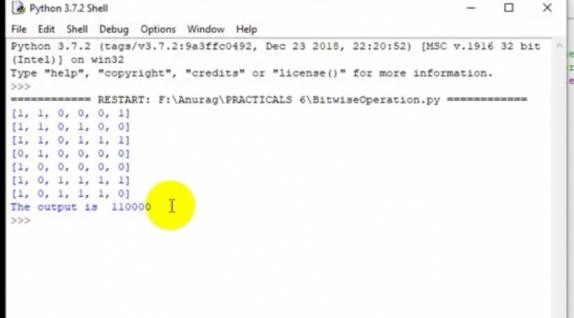
for vector in vector\_matrix: mystring = ""

for digit in vector: mystring += str(digit)

string\_list.append(int(mystring,2)) #print(string\_list)

print("The output is ",bin(string\_list[0]&string\_list[1]&(string\_list[2])).replace("0b",""))

**OUTPUT :**



# PRACTICAL NO:-02

**AIM:** IMPLEMENT PAGE RANK ALGORITHM.

**CODE:**

import numpy as np import scipy as sc import pandas as pd

from fractions import Fraction

def display\_format(my\_vector, my\_decimal): return np.round((my\_vector).astype(np.float),

decimals=my\_decimal)

my\_dp = Fraction(1,1) Mat = np.matrix([[0,0,1],

[Fraction(1,2),0,0],

[Fraction(1,2),1,0]]) Ex = np.zeros((3,3)) Ex[:] = my\_dp

Damp = 0.7

Al = Damp \* Mat + ((1-Damp) \* Ex)

r = np.matrix([my\_dp, my\_dp, my\_dp]) r = np.transpose(r)

previous\_r = r

for i in range(1,10): r = Al \* r

print (display\_format(r,3)) if (previous\_r==r).all():

break previous\_r = r

print ("Final:\n", display\_format(r,3)) print ("sum", np.sum(r))

Output [[0.333]

[0.217]

[0.45 ]]

[[0.415]

[0.217]

[0.368]]

[[0.358]

[0.245]

[0.397]]

[[0.378]

[0.225]

[0.397]]

[[0.378]

[0.232]

[0.39 ]]

[[0.373]

[0.232]

[0.395]]

[[0.376]

[0.231]

[0.393]]

[[0.375]

[0.232]

[0.393]]

[[0.375]

[0.231]

[0.394]]

[[0.375]

[0.231]

[0.393]]

[[0.375]

[0.231]

[0.393]]

# PRACTICAL NO:-03

**AIM:** IMPLEMENT DYNAMIC PROGRAMMING ALGORITHM FOR COMPUTING THE EDIT DISTANCE BETWEEN STRINGS S1 AND S2. (HINT. LEVENSHTEIN DISTANCE)

**CODE:**

def editDistance(str1, str2, m, n): if m == 0:

return n if n == 0:

return m

if str1[m-1]== str2[n-1]:

return editDistance(str1, str2, m-1, n-1)

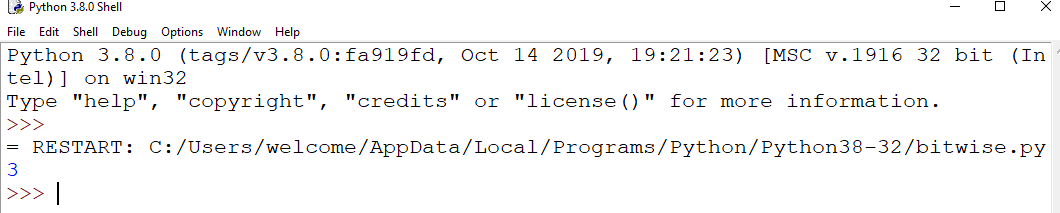
return 1 + min(editDistance(str1, str2, m, n-1), # Insert

editDistance(str1, str2, m-1, n), # Remove editDistance(str1, str2, m-1, n-1) # Replace )

str1 = "sunday" str2 = "saturday"

print (editDistance(str1, str2, len(str1), len(str2)) )

**OUTPUT:**



# PRACTICAL NO:-04

**AIM:** WRITE A PROGRAM TO COMPUTE SIMILARITY BETWEEN TWO TEXT DOCUMENTS.

**CODE:**

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize # X = input("Enter first string: ").lower()

# Y = input("Enter second string: ").lower() X =open('file1.txt','r').read()

Y =open('file2.txt','r').read() # tokenization

X\_list = word\_tokenize(X) Y\_list = word\_tokenize(Y)

# sw contains the list of stopwords sw = stopwords.words('english') l1 =[];l2 =[]

# remove stop words from string

X\_set = {w for w in X\_list if not w in sw} Y\_set = {w for w in Y\_list if not w in sw}

# form a set containing keywords of both strings rvector = X\_set.union(Y\_set)

for w in rvector:

if w in X\_set: l1.append(1) # create a vector else: l1.append(0)

if w in Y\_set: l2.append(1) else: l2.append(0)

c = 0

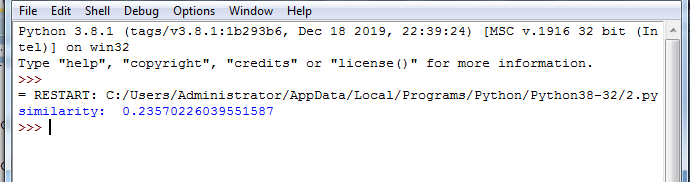
# cosine formula

for i in range(len(rvector)):

c+= l1[i]\*l2[i]

cosine = c / float((sum(l1)\*sum(l2))\*\*0.5) print("similarity: ", cosine)

**OUTPUT:**



# PRACTICAL NO:-05

**AIM:** WRITE A MAP-REDUCE PROGRAM TO COUNT THE NUMBER OF OCCURRENCES OF EACH ALPHABETIC CHARACTER IN THE GIVEN DATASET. THE COUNT FOR EACH LETTER SHOULD BE CASE-INSENSITIVE (I.E., INCLUDE BOTH UPPER-CASE AND LOWER-CASE VERSIONS OF THE LETTER; IGNORE NON-ALPHABETIC CHARACTERS).

**CODE:**

Text="""MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

Map stage − The map or mapper’s job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.

Reduce stage − This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer’s job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

"""

# Cleaning text and lower casing all words for char in '-.,\n':

Text=Text.replace(char,' ')

Text = Text.lower()# split returns a list of words delimited by sequences of whitespace (including tabs, newlines, etc, like re's \s)

word\_list = Text.split()

from collections import Counter Counter(word\_list).most\_common() # Initializing Dictionary

d = {}

# counting number of times each word comes up in list of words (in dictionary) for word in word\_list:

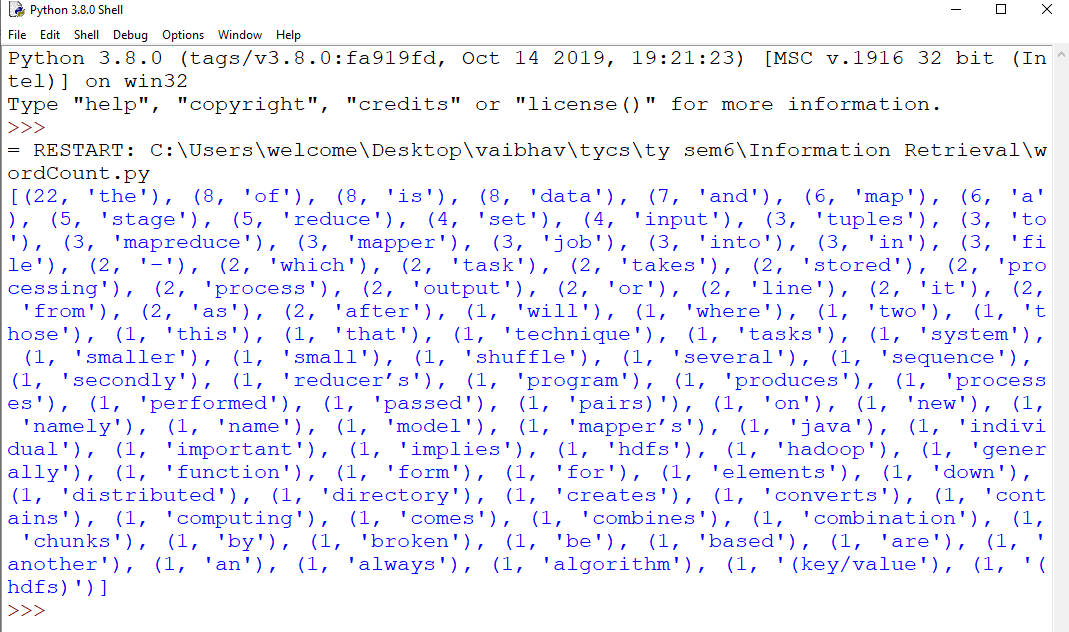
d[word] = d.get(word, 0) + 1

#reverse the key and values so they can be sorted using tuples. word\_freq = []

for key, value in d.items(): word\_freq.append((value, key))

word\_freq.sort(reverse=True) print(word\_freq)

**OUTPUT:**



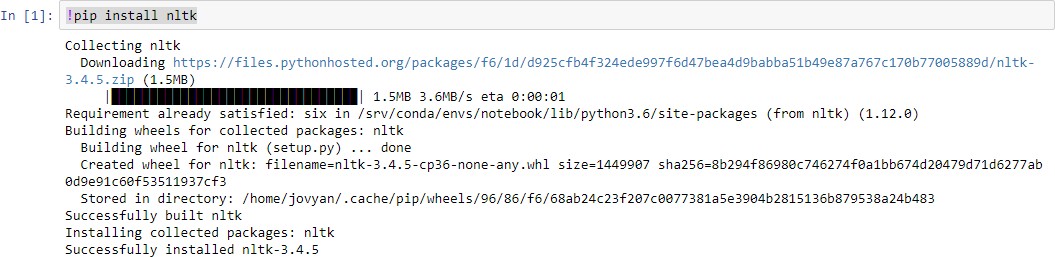
# PRACTICAL NO:-06

**AIM:** WRITE A PROGRAM FOR PRE-PROCESSING OF A TEXT DOCUMENT: STOP WORD REMOVAL.

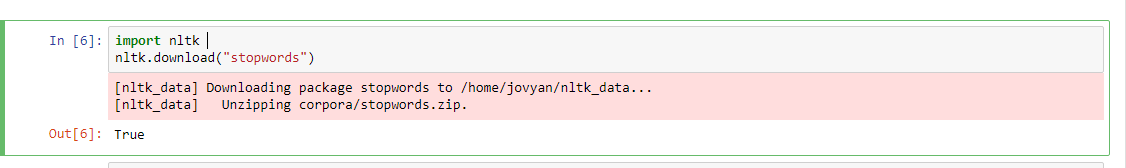
**CODE:**

1. Install nltk

!pip install nltk

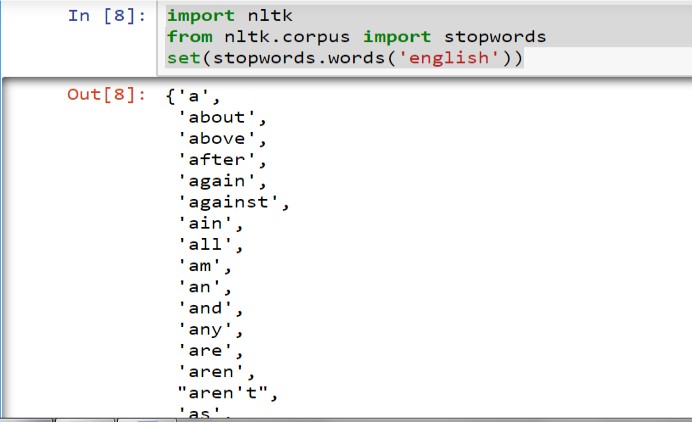


1. download stopwords in nltk import nltk nltk.download("stopwords")



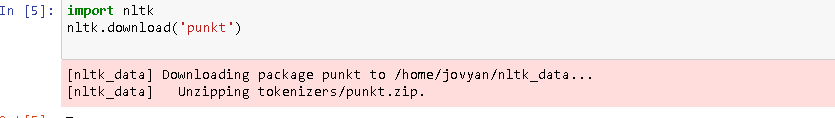
import nltk

from nltk.corpus import stopwords set(stopwords.words('english'))



now download punkt in nltk

import nltk nltk.download('punkt')



**4**. Stopwords coding

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

example\_sent="This is a sample sentence,showing off the stop words filtration." stop\_words=set(stopwords.words('english')) word\_tokens=word\_tokenize(example\_sent)

filtered\_sentence=[w for w in word\_tokens if not w in stop\_words] filtered\_sentence=[]

for w in word\_tokens:

if w not in stop\_words: filtered\_sentence.append(w)

print(word\_tokens) print(filtered\_sentence)

**OUTPUT:**



# PRACTICAL NO:-07

**AIM:** WRITE A PROGRAM TO IMPLEMENT SIMPLE WEB CRAWLER.

A Web Crawler is a program that navigates the Web and finds new or updated pages for indexing. The Crawler starts with seed websites or a wide range of popular URLs (also known as the frontier) and searches in depth and width for hyperlinks to extract.

A Web Crawler must be kind and robust. Kindness for a Crawler means that it respects the rules set by the robots.txt and avoids visiting a website too often. Robustness refers to the ability to avoid spider traps and other malicious behavior. Other good attributes for a Web Crawler is distributivity amongst multiple distributed machines, expandability, continuity and ability to prioritize based on page quality.

**Steps to create web crawler**

**The basic steps to write a Web Crawler are:**

* Pick a URL from the frontier
* Fetch the HTML code
* Parse the HTML to extract links to other URLs
* Check if you have already crawled the URLs and/or if you have seen the same content before

If not add it to the index For each extracted URL

Confirm that it agrees to be checked (robots.txt, crawling frequency)

**CODE:**

import requests

from bs4 import BeautifulSoup

URL = "<https://en.wikipedia.org/wiki/States_and_union_territories_of_India>" res = requests.get(URL).text

soup = BeautifulSoup(res,'lxml') states=[]

for items in soup.find('table', class\_='wikitable').find\_all('tr')[1::1]: data = items.find\_all(['th','td'])

#print(data[0].text)

states.append(data[0].text) print(states)

**OUTPUT:**



# PRACTICAL NO:-08

**AIM:** WRITE A PROGRAM TO PARSE XML TEXT, GENERATE WEB GRAPH AND COMPUTE TOPIC SPECIFIC PAGE RANK.

**CODE:**

**Xml file:**

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<root testAttr="testValue"> The Tree

<children>

<child name="Jack">First</child>

<child name="Rose">Second</child>

<child name="Blue Ivy"> Third

<grandchildren>

<data>One</data>

<data>Two</data>

<unique>Twins</unique>

</grandchildren>

</child>

<child name="Jane">Fourth</child>

</children>

</root>

import xml.etree.ElementTree as ET tree = ET.parse('items.xml')

root = tree.getroot() # all items data

print('Expertise Data:') for elem in root:

for subelem in elem: print(subelem.text)

**OUTPUT:**

Expertise Data: Expertise Data: First

Second Third