T.Y.BSc

COMPUTER SCIENCE

INFORMATION RETRIEVAL

PRACTICALS

LIST OF PRACTICALS:

1. Write a program to demonstrate bitwise operation.
2. Implement Page Rank Algorithm.
3. Implement Dynamic programming algorithm for computing the edit distance between strings s1 and s2. (Hint. Levenshtein Distance)
4. Write a program to Compute Similarity between two text documents.
5. Write a program for Pre-processing of a Text Document: stop word removal.
6. Write a program for mining Twitter to identify tweets for a specific period and identify trends and named entities.
7. Write a program to implement simple web crawler.
8. Write a program to parse XML text, generate Web graph and compute topic specific page rank.
9. Bitwise operation:

a = 60 # 60 = 0011 1100

b = 13 # 13 = 0000 1101

c = 0

c = a & b; # 12 = 0000 1100

print("Line 1 - Value of c is ", c)

c = a | b; # 61 = 0011 1101

print ("Line 2 - Value of c is ", c)

c = a ^ b; # 49 = 0011 0001

print ("Line 3 - Value of c is ", c)

c = ~a; # -61 = 1100 0011

print ("Line 4 - Value of c is ", c)

c = a << 2; # 240 = 1111 0000

print ("Line 5 - Value of c is ", c)

c = a >> 2; # 15 = 0000 1111

print ("Line 6 - Value of c is ", c)

1. Page Rank Algorithm:

import numpy as np

from scipy.sparse import csc\_matrix

def float\_format(vector, decimal):

return np.round(vector.astype(float), decimals=decimal)

G = np.matrix([[1, 1, 0],

[1, 0, 1],

[0, 1, 0]])

n = len(G)

M = csc\_matrix(G, dtype=float)

rsums = np.array(M.sum(1))[:, 0]

ri, ci = M.nonzero()

M.data /= rsums[ri]

dp = 1 / n

E = np.zeros((n, n))

E[:] = dp

beta = 0.85

A = beta \* M + ((1 - beta) \* E)

r = np.full((n, 1), dp)

previous\_r = r

for it in range(30):

r = A @ r

if np.allclose(previous\_r, r, atol=1e-6):

break

previous\_r = r

print("Final PageRank:\n", float\_format(r, 3))

print("Sum of ranks:",np.sum(r))

1. Edit distance

import numpy as np

def levenshtein(s1,s2):

size\_x=len(s1)+1

size\_y=len(s2)+1

matrix=np.zeros((size\_x, size\_y))

for x in range(size\_x):

matrix[x,0]=x

for y in range(size\_y):

matrix[0,y]=y

for x in range(1,size\_x):

for y in range(1,size\_y):

if s1[x-1] == s2[y-1]:

matrix[x,y]=min(matrix[x-1,y]+ 1,matrix[x-1,y-1],matrix[x,y-1]+1)

else:

matrix[x,y]=min(matrix[x-1,y]+ 1,matrix[x-1,y-1]+1,matrix[x,y-1]+1)

print(matrix)

return(matrix[size\_x-1,size\_y-1])

levenshtein("Hello","hallo")

1. Compute similarity between 2 documents:

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

import numpy as np

import nltk

def process(file):

raw=open(file).read()

tokens=word\_tokenize(raw)

words=[w.lower() for w in tokens]

porter=nltk.PorterStemmer()

Stemmed\_tokens=[porter.stem(t) for t in words]

#removing stop words

stop\_words=set(stopwords.words('english'))

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

#count words

count=nltk.defaultdict(int)

for word in filtered\_tokens:

count[word]+=1

return count

def cos\_sim(a,b):

dot\_product=np.dot(a,b)

norm\_a=np.linalg.norm(a)

norm\_b=np.linalg.norm(b)

return dot\_product/(norm\_a \* norm\_b)

def getSimilarity(dict1,dict2):

all\_words\_list=[]

for key in dict1:

all\_words\_list.append(key)

for key in dict2:

all\_words\_list.append(key)

all\_words\_list\_size=len(all\_words\_list)

v1=np.zeros(all\_words\_list\_size,dtype=np.int)

v2=np.zeros(all\_words\_list\_size,dtype=np.int)

i=0

for (key) in all\_words\_list:

v1[i]=dict1.get(key,0)

v2[i]=dict2.get(key,0)

i=i+1

return cos\_sim(v1,v2)

if \_\_name\_\_=='\_\_main\_\_':

dict1=process("D:\TYCS\_41\Information Retrieval/text1.txt")

dict2=process("D:\TYCS\_41\Information Retrieval/text2.txt")

print("Similarity between two text documents",getSimilarity(dict1,dict2))

7. stop word removal:

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)

8. Web crawler:

import requests

from bs4 import BeautifulSoup

url=("www.amazon.in")

code=requests.get("https://" +url)

plain=code.text

s=BeautifulSoup(plain)

for link in s.find\_all("a"):

print(link.get("href"))

9. Twitter

import tweepy

consumer\_key='rCLpGlj086YIYl3xjz6dwNWTw'

consumer\_secret='8dDn10CO6k4HYhg2GIQepYiJXoW8aJ6W2UyvQew2cgupgX4uam'

access\_token='1104215432985305089-JzFqwAXhBBdAztqrKTkhFc3RGFLu6r'

access\_token\_secret='mSdxQ2uLCP0IWUoACCQp1IT8L6sM53RA7N12E5i6y5Oiq'

auth=tweepy.OAuthHandler(consumer\_key,consumer\_secret)

auth.set\_access\_token(access\_token,access\_token\_secret)

api=tweepy.API(auth)

public\_tweets=api.home\_timeline()

for tweet in public\_tweets:

print(tweet.text)

#name="modi"

#tweetCount=10

#results=api.user\_timeline(id=name,count=tweetCount)

#for tweet in results:

#print(tweet.text)

1. **XML parse tree**

import csv

import requests

import xml.etree.ElementTree as ET

def loadRSS():

url = 'http://www.hindustantimes.com/rss/topnews/rssfeed.xml'

resp = requests.get(url)

with open('topnewsfeed.xml', 'wb') as f:

f.write(resp.content)

def parseXML(xmlfile):

tree = ET.parse(xmlfile)

root = tree.getroot()

newsitems=[]

for item in root.findall('./channel/item'):

news = {}

for child in item:

if child.tag == '{http://search.yahoo.com/mrss/}content':

news['media']=child.attrib['url']

else:

news[child.tag]=child.text.encode('utf8')

newsitems.append(news)

return newsitems

def savetoCSV(newsitems, filename):

fields = ['guid', 'title', 'pubDate', 'description', 'link', 'media']

with open(filename, 'w') as csvfile:

writer = csv.DictWriter(csvfile, fieldnames=fields)

writer.writeheader()

writer.writerows(newsitems)

loadRSS()

newsitems = parseXML('topnewsfeed.xml')

savetoCSV(newsitems, 'topnews.csv')

def generate\_edges(graph):

edges=[]

for node in graph:

for neighbour in graph[node]:

edges.append((node,neighbour))

return edges