KERALYEEA SAMAJAM DOMBIVLI'S MODEL COLLEGE

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 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).
6. Implement a basic IR system using Lucene.
7. Write a program for Pre-processing of a Text Document: stop word removal.
8. Write a program for mining Twitter to identify tweets for a specific period and identify trends and named entities.
9. Write a program to implement simple web crawler.
10. Write a program to parse XML text, generate Web graph and compute topic specific page rank.

1. 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)
```

2. Page Rank Algorithm:

```
# import some stuff
import numpy as np
from scipy.sparse import csc_matrix

from fractions import Fraction

# keep it clean and tidy
def float_format(vector, decimal):
    return np.round((vector).astype(np.float), decimals=decimal)

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

n=len(G)
#print(n)
# transform G into markov matrix A
```

```
M = csc_matrix(G,dtype=np.float)
rsums = np.array(M.sum(1))[:,0]
ri, ci = M.nonzero()
M.data /= rsums[ri]



# WWW matrix
# we have 3 webpages and probability of landing to each one is
1/3
#(default Probability)
#n=len(M)
dp = Fraction(1,n)

E = np.zeros((3,3))
E[:] = dp


# taxation
beta = 0.85

# WWW matrix
A = beta * M + ((1-beta) * E)

# initial vector

r = np.matrix([dp, dp, dp])
r = np.transpose(r)

previous_r = r
for it in range(1,30):
    r = A * r
    #check if converged
    if (previous_r==r).all():
        break
    previous_r = r

print ("Final:\n", float_format(r,3))
print( "sum", np.sum(r))
```

3. 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")
```

4. 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='8dDn10CO6k4HYhg2GIQepYiJXoW8aJ6W2UyvQew2cgupgX4
uam'
access_token='1104215432985305089-JzFqwAXhBBdAztqrKTkhFc3RGFLu6r
'
access_token_secret='mSdxQ2uLCP0IWUoACCQp1IT8L6sM53RA7N12E5i6y5O
iq'
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)
```

10 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
```