**Experiment-02**

**--------------------------------------------------------------------**

**Name:** Atharva Paliwal

**Roll No:** 40

**Date:** 11/08/2021

**--------------------------------------------------------------------**

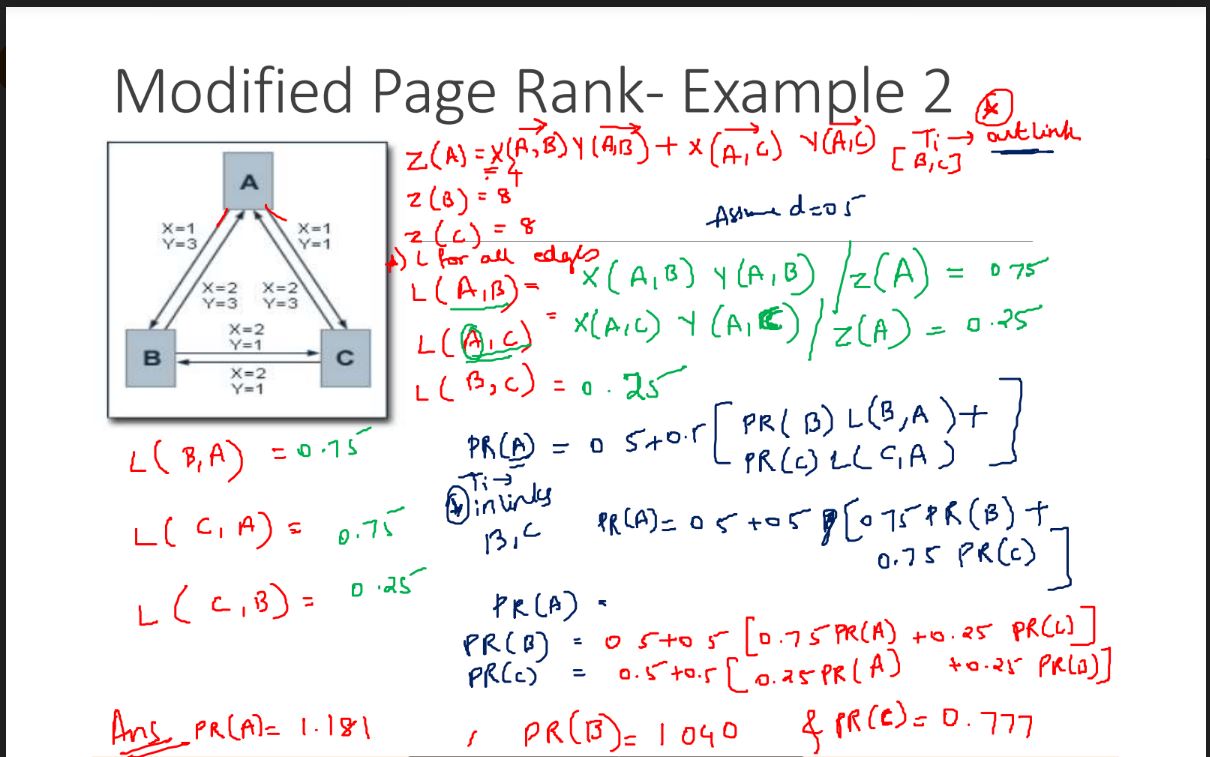
**Aim:** Write a program to:

A. Compute the Page Rank of all the webpages in an undirected web graph using the modified page rank method.

B. Integrate the page rank code with inverted index (practical 1) and now the results must be ranked according to page rank instead of term frequency.

**Details:**

* Accept graph as input from the user. By accepting names of web pages and one by one entering the edges.
* Ask the user to enter the information about the link location (upper half / lower half) and if it is emphasized. Accordingly, find the value of X and Y (weights) for each edge.
* Write function to create a weighted graph and add edges in a graph.
* Decide the data structure to be used for implementing graph. Also display graph if possible. [networkx can be used in python]
* Using the formulas of modified page rank find the equation for each web page in the web graph.
* Compute the page rank using the iterative approach for page rank computation.
* Display the page ranks and web pages ranked according to the page rank.
* Integrate the page rank code with inverted index (practical 1) and now the results must be ranked according to page rank instead of term frequency.
* Consider only 4 to 5 web pages in the web graph and text files for inverted index creation.

**Theory:** 

**Program:**

#PART-1

#Part 1.0 - GRAPH INPUT

n,e=map(int,input("Enter Number of Pages and Edge Connection : ").split())

adj=[[] for i in range(n+1)]

inLink = {}

for \_ in range(e):

u,v=map(int,input().split())

if (v in inLink):

inLink[v].append(u)

else:

inLink[v] = [u]

x=int(input("Enter 1 if Link is not Emphasized and 2 if Link is Emphasized : "))

y=int(input("Enter 1 if Link is in Lower Half and 3 if Link is in Upper Half : "))

w=[x,y]

adj[u].append([v,w])

# print(adj[1])

#Displaying incomming Links of Nodes

print('inLinks : ',inLink)

#Part 1.1 - VALUE OF Z

# Z = Sum(X(A,outLinks)\*Y(A,outLinks))

z=[]

for i in range(1,n+1):

curr\_z=0

for j in adj[i]:

curr\_z+=j[1][0]\*j[1][1]

z.append(curr\_z)

#Displaying Z values for all nodes

print("Z : ", z)

#Part1.2 - VALUE OF L

# L(A,B) = X(A,B)\*Y(A,B) / Z(A)

L={}

for i in range(1,n+1):

for j in adj[i]:

L[(i,j[0])]=j[1][0]\*j[1][1]/z[i-1]

#Displaying L values for all edges

print("L : ", L)

#Part 1.3 - VALUE OF PAGE RANK

#d=0.5

# PR(A) = 0.5 + 0.5\*(PR(inLinks)\*L(inLinks, A))

pageRank = [0]\*(n+1)

tempRank = [1]\*(n+1)

while pageRank != tempRank:

tempRank = pageRank[:]

for i in range(1, n+1):

temp = 0

for j in inLink[i]:

temp = temp + pageRank[j]\*L[(j,i)]

pageRank[i] = 0.5 + 0.5\*temp

print("Page Rank : ", pageRank[1:])

print()

print()

###################################################################

print("=======================PART-2==============================")

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

ps = PorterStemmer()

def readfile(filename,dict,cnt):

#reading filename

file = open(filename, encoding='utf8')

read = file.read() #extracting the file contents

file.seek(0) #pointer from where the data has to be written/read

line = 1 #line number

for word in read: #extracting each word from the file contents

if word == '\n': #if new line occurs increment the line

line += 1

array = []

for i in range(line):

array.append(file.readline()) #appending every line in an array

#1.REMOVE PUNCTUATIONS

punc = '''!()-[]{};:'"\, <>./?@#$%^&\*\_~''' #string of punctuations

#checking whether there is any punctuation present,if any then replacing it with a space

for ele in read:

if ele in punc:

read = read.replace(ele, " ")

read=read.lower() #doing all elements lowercase

#2.REMOVING STOPWORDS

for i in range(1):

text\_tokens = word\_tokenize(read) #tokenizing every word ex -> form becomes 'form'

tokens\_without\_sw = [word for word in text\_tokens if not word in stopwords.words()] #list of tokenized words without stopwords

#3. STORING IN DICTIONARY

for i in range(line):

check = array[i].lower()

for w in tokens\_without\_sw:

# print(dict\_for\_cnt)

if w in check:

item=ps.stem(w) #This line gives us the root word for the current word

if item not in dict:

dict[item] = [] #creating a list for storing count if the word is not present in dict

if item in dict:

dict[item].append(cnt) #appending the file number where the word is present

dict[item]=list(set(dict[item]))#using set so that if any word is present multiple times in a file

#then it is not written multiple times

#DRIVER CODE

dict={} #dictionary for storing the file numbers

for i in range(1,4):

readfile('text'+str(i)+'.txt',dict,i) #reading each file once

#Displaying all the words along with there occurence in which file

for word,list\_of\_files in dict.items():

print(word,' : ',list\_of\_files)

query = input("Enter String to be searched : ")

qword = query.split()

qwords=[]

for q in qword:

qroot=ps.stem(q) #getting the root word for the current word

if qroot not in qwords:

qwords.append(qroot) #appending the root words

#Displaying in which file the string is present

word\_list=[] #list for displaying part 3

for q in qword:

word=ps.stem(q) #taking out the root word for the current word

for doc\_id in dict[word]:

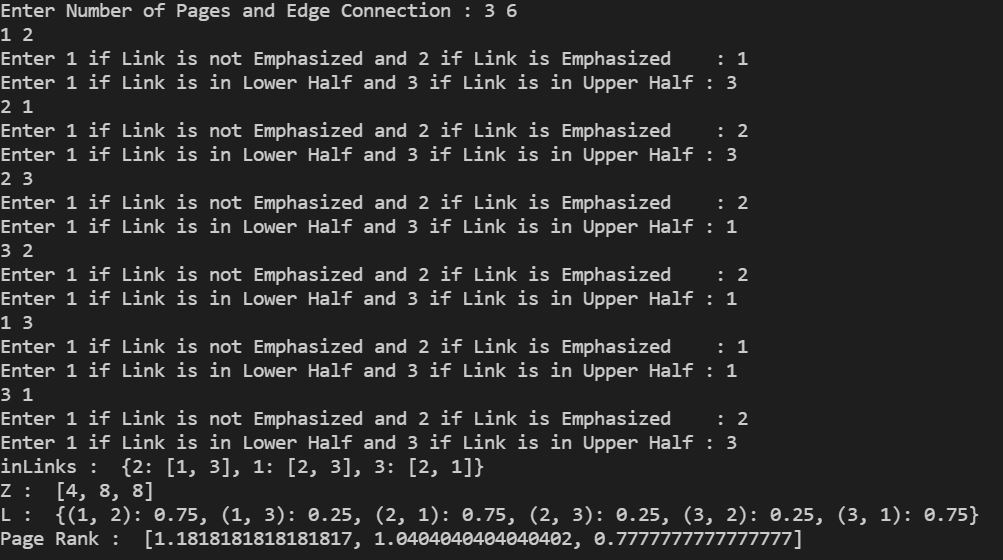
word\_list.append((pageRank[doc\_id],doc\_id,q)) #appending for every doc\_id the count the word appears in the doc

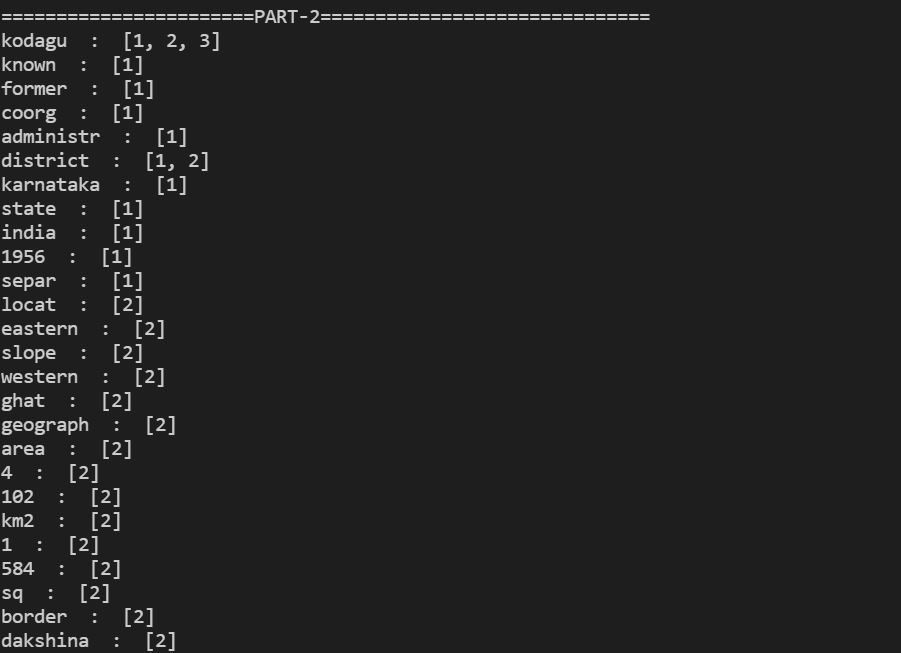
word\_list.sort(reverse=True)

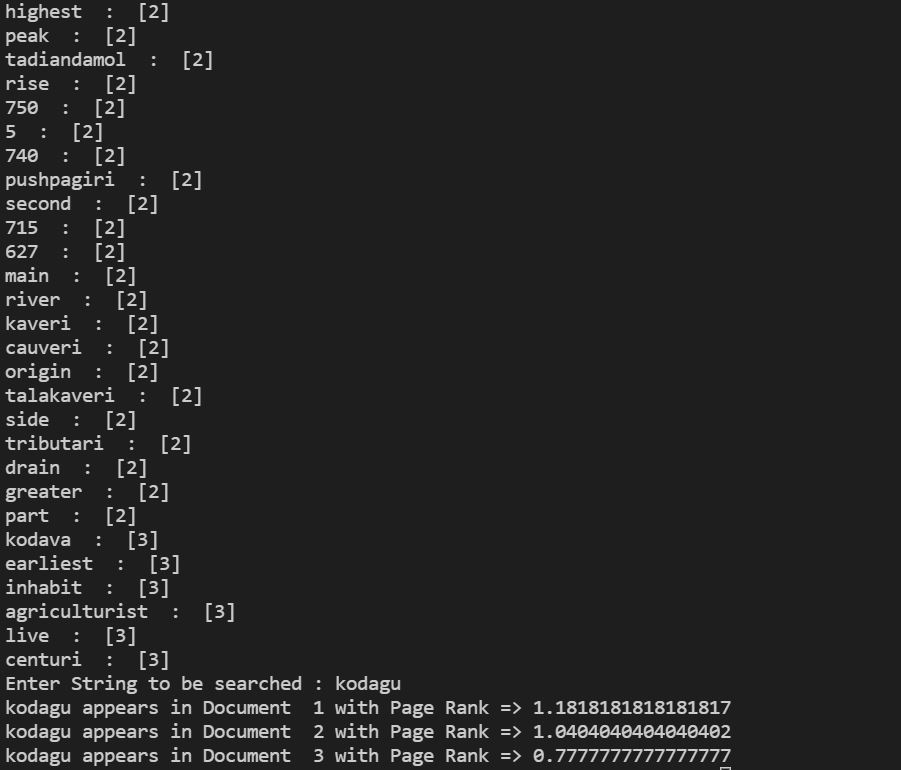
for word in word\_list:

print(word[2],'appears in Document ',word[1],'with Page Rank =>',word[0])

**Output:**







**\*\*\*END\*\*\***