IR PRACTICALS

IR

Information retrieval (IR) is the process of retrieving relevant information from a large collection of data or documents. The goal of IR is to provide the user with the most relevant information that matches their query or information need.

IR typically involves the following steps:

1. Document collection: Gathering a large collection of data or documents related to the topic of interest.
2. Preprocessing: Cleaning, formatting, and transforming the documents into a standardized format for efficient storage and retrieval.
3. Indexing: Creating an index of the words or terms in the documents, along with their location, to speed up the retrieval process.
4. Query processing: Analyzing the user's query or search terms and selecting the most relevant documents from the index.
5. Ranking: Assigning a score or ranking to each document based on its relevance to the query, using techniques such as vector space model or probabilistic models.
6. Presentation: Displaying the results to the user in a meaningful and organized way, such as a list of ranked documents or a summary of the most relevant information.

IR is used in various applications, such as search engines, recommender systems, and information filtering. It is an important field in natural language processing and information science, as it helps users efficiently find the information they need from large volumes of data.

Practical 1

Bitwise Operation AND OR XOR NEGATE LEFTSHIFT RIGHTSHIFT

Theory:

Bitwise operations are operations that are performed on individual bits of binary numbers. These operations are often used in low-level programming, such as in embedded systems or device drivers.

The following are the bitwise operators:

1. AND (&): The AND operator compares the corresponding bits of two operands and returns a 1 if both bits are 1, otherwise it returns a 0.
2. XOR (^): The XOR operator compares the corresponding bits of two operands and returns a 1 if the bits are different, otherwise it returns a 0.
3. OR (|): The OR operator compares the corresponding bits of two operands and returns a 1 if at least one of the bits is 1, otherwise it returns a 0.
4. NEGATE (~): The NEGATE operator inverts all the bits of an operand, i.e., it changes 1s to 0s and 0s to 1s.
5. LEFTSHIFT (<<): The LEFTSHIFT operator shifts the bits of an operand to the left by a specified number of positions. The bits that are shifted out of the left end are lost, and 0s are shifted in from the right end.

6. RIGHTSHIFT (>>): The RIGHTSHIFT operator shifts the bits of an operand to the right by a specified number of positions. The bits that are shifted out of the right end are lost, and 0s are shifted in from the left end.

Code:

```
from tkinter import *

class MyWindow:

    def __init__(self, win):

        self.lbl1=Label(win, text='ENTER FIRST NUMBER:',font='Helvetica 10 bold',bg="black",fg="white")

        self.lbl2=Label(win, text='ENTER SECOND NUMBER:',font='Helvetica 10 bold',bg="black",fg="white")

        self.lbl3=Label(win, text='RESULT:',font='Helvetica 10 bold',bg="black",fg="white")

        self.t1=Entry(bd=3)

        self.t2=Entry()

        self.t3=Entry()

        self.btn1 = Button(win, text='AND')

        self.btn2=Button(win, text='OR')

        self.btn3=Button(win,text="Xor")

        self.btn4=Button(win,text="NEGATE")

        self.btn5=Button(win,text="LEFTSHIFT")

        self.btn6=Button(win,text="RIGHTSHIFT")

        self.lbl1.place(x=100, y=50)

        self.t1.place(x=290, y=50)

        self.lbl2.place(x=100, y=100)

        self.t2.place(x=290, y=100)

        self.b1=Button(win, text='AND',font='Helvetica 10 bold', command=self.And)

        self.b2=Button(win, text='OR',font='Helvetica 10 bold')

        self.b2.bind('<Button-1>', self.Or)

        self.b3=Button(win,text="Xor",font='Helvetica 10 bold')

        self.b3.bind('<Button-1>',self.Xor)

        self.b4=Button(win,text="NEGATE",font='Helvetica 10 bold')

        self.b4.bind('<Button-1>',self.Neg)

        self.b5=Button(win, text='LEFTSHIFT',font='Helvetica 10 bold')

        self.b5.bind('<Button-1>',self.LSH)

        self.b6=Button(win, text='RIGHTSHIFT',font='Helvetica 10 bold')

        self.b6.bind('<Button-1>',self.RSH)

        self.b1.place(x=100, y=150)

        self.b2.place(x=200, y=150)
```

```
        self.b3.place(x=300,y=150)
        self.b4.place(x=400,y=150)
        self.b5.place(x=500, y=150)
        self.b6.place(x=600, y=150)
        self.lbl3.place(x=100, y=200)
        self.t3.place(x=200, y=200)
    def And(self):
        self.t3.delete(0, 'end')
        num1=int(self.t1.get())
        num2=int(self.t2.get())
        result=num1&num2
        self.t3.insert(END, str(result))
    def Or(self, event):
        self.t3.delete(0, 'end')
        num1=int(self.t1.get())
        num2=int(self.t2.get())
        result=num1|num2
        self.t3.insert(END, str(result))
    def Xor(self, event):
        self.t3.delete(0, 'end')
        num1=int(self.t1.get())
        num2=int(self.t2.get())
        result=num1^num2
        self.t3.insert(END, str(result))
    def Neg(self, event):
        self.t3.delete(0, 'end')
        num1=int(self.t1.get())
        num2=int(self.t2.get())
        result=~num1
        self.t3.insert(END, str(result))
    def LSH(self, event):
        self.t3.delete(0, 'end')
        num1=int(self.t1.get())
        num2=int(self.t2.get())
        result=num1<<2
        self.t3.insert(END, str(result))
```

```
    def RSH(self, event):
        self.t3.delete(0, 'end')
        num1=int(self.t1.get())
        num2=int(self.t2.get())
        result=num1>>2
        self.t3.insert(END, str(result))
window=Tk()
mywin=MyWindow(window)
window.title('Tkinter Arthematic operations')
window.geometry("700x500+10+10")
window.config(bg="light green")
window.mainloop()
```

PRACTICAL 2

Page rank algorithm(JAVA)

Theory : PageRank is a ranking algorithm used by search engines to rank web pages in their search engine results. It was developed by Larry Page and Sergey Brin, the founders of Google, while they were studying at Stanford University.

PageRank is based on the idea that a web page's importance is determined by the number and quality of the links pointing to it. The more high-quality links a page has, the more important it is considered to be. The algorithm works by assigning each web page a PageRank score, which is calculated based on the number and quality of the links pointing to it.

The PageRank score is represented as a decimal number between 0 and 1. A score of 0 means that the page has no incoming links, while a score of 1 means that the page has a large number of high-quality incoming links.

The algorithm works by analyzing the link structure of the web and calculating the PageRank scores for each page based on the links pointing to it. The algorithm takes into account the number of links pointing to a page, the quality of the pages linking to it, and the relevance of the linking pages to the content of the target page.

PageRank is just one of many ranking algorithms used by search engines. While it was once a major factor in determining search engine rankings, it is now just one of many factors used by search engines to determine the relevance and importance of web pages.

Code:

```
import java.util.*;
import java.io.*;
public class PageRank {
 public int path[][] = new int[10][10];
 public double pagerank[] = new double[10];
 public void calc(double totalNodes) {
  double InitialPageRank;
  double OutgoingLinks = 0;
  double DampingFactor = 0.85;
  double TempPageRank[] = new double[10];
  int ExternalNodeNumber;
  int InternalNodeNumber;
  int k = 1; // For Traversing
  int ITERATION_STEP = 1;
  InitialPageRank = 1 / totalNodes;
  System.out.printf(" Total Number of Nodes :" + totalNodes + "\t Initial PageRank  of All
Nodes :" + InitialPageRank + "\n");
  for (k = 1; k <= totalNodes; k++) {
   this.pagerank[k] = InitialPageRank;
  }
  System.out.printf("\n Initial PageRank Values , 0th Step \n");
  for (k = 1; k <= totalNodes; k++) {
   System.out.printf(" Page Rank of " + k + " is :\t" + this.pagerank[k] + "\n");
  }
  while (ITERATION_STEP <= 2)
  {
   for (k = 1; k <= totalNodes; k++) {
    TempPageRank[k] = this.pagerank[k];
```

```
   this.pagerank[k] = 0;
  }
  for (InternalNodeNumber = 1; InternalNodeNumber <= totalNodes;
InternalNodeNumber++) {
   for (ExternalNodeNumber = 1; ExternalNodeNumber <= totalNodes;
ExternalNodeNumber++) {
    if (this.path[ExternalNodeNumber][InternalNodeNumber] == 1) {
     k = 1;
     OutgoingLinks = 0;
     while (k <= totalNodes) {
      if (this.path[ExternalNodeNumber][k] == 1) {
       OutgoingLinks = OutgoingLinks + 1; // Counter for Outgoing Links
      }
      k = k + 1;
     }
     this.pagerank[InternalNodeNumber] += TempPageRank[ExternalNodeNumber] * (1 /
OutgoingLinks);
    }
   }
  }
  System.out.printf("\n After " + ITERATION_STEP + "th Step \n");
  for (k = 1; k <= totalNodes; k++)
   System.out.printf(" Page Rank of " + k + " is :\t" + this.pagerank[k] + "\n");
  ITERATION_STEP = ITERATION_STEP + 1;
 }
 for (k = 1; k <= totalNodes; k++) {
  this.pagerank[k] = (1 - DampingFactor) + DampingFactor * this.pagerank[k];
 }
 System.out.printf("\n Final Page Rank : \n");
 for (k = 1; k <= totalNodes; k++) {
```

```
  System.out.printf(" Page Rank of " + k + " is :\t" + this.pagerank[k] + "\n");
 }
}
public static void main(String args[]) {
 int nodes, i, j, cost;
 Scanner in = new Scanner(System.in);
 System.out.println("Enter the Number of WebPages \n");
 nodes = in .nextInt();
 PageRank p = new PageRank();
 System.out.println("Enter the Adjacency Matrix with 1->PATH & 0->NO PATH Between two
WebPages: \n");
 for (i = 1; i <= nodes; i++)
  for (j = 1; j <= nodes; j++) {
   p.path[i][j] = in .nextInt();
   if (j == i)
    p.path[i][j] = 0;
  }
 p.calc(nodes);

}
}
```

PRACTICAL 3:

DYANAMIC LEVENSHTEIN DISTANCE BTWN S1 AND S2

Theory :

Levenshtein distance, also known as edit distance, is a metric used to measure the difference between two sequences of characters. It is defined as the minimum number of single-character insertions, deletions, and substitutions required to transform one string into another.

The dynamic Levenshtein algorithm is a method of calculating the Levenshtein distance between two strings using dynamic programming. In this method, a matrix is used to store the distances between all possible prefixes of the two strings. The matrix is then used to calculate the distance

between the full strings by building up the distances from the prefixes. The dynamic programming approach allows for efficient calculation of the Levenshtein distance between two strings, even for long strings.

Code :

```
package levenshteindistancedp;

import java.util.*;

class LevenshteinDistanceDP {

static int compute_Levenshtein_distanceDP(String str1,String str2)

{

int[][] dp = new int[str1.length() + 1][str2.length() + 1];

for (int i = 0; i <= str1.length(); i++)

{

for (int j = 0; j <= str2.length(); j++) {

if (i == 0) {

dp[i][j] = j;

}

else if (j == 0) {

dp[i][j] = i;

}

else {

dp[i][j] = minm_edits(dp[i - 1][j - 1] + NumOfReplacement(str1.charAt(i - 1),str2.charAt(j - 1)),

dp[i - 1][j] + 1,

dp[i][j - 1] + 1);

}

}

}

return dp[str1.length()][str2.length()];

}

static int NumOfReplacement(char c1, char c2)

{

return c1 == c2 ? 0 : 1;
```

```
}
static int minm_edits(int... nums)
{
return Arrays.stream(nums).min().orElse(
Integer.MAX_VALUE);
}
public static void main(String args[])
{
String s1 = "glomax";
String s2 = "folmax";
System.out.println(compute_Levenshtein_distanceDP(s1, s2));
}
}
```

PRACTICAL 4:

RECURSIVE LEVENSHTEIN DISTANCE BTWN S1 AND S2

Theory :

Levenshtein distance, also known as edit distance, is a metric used to measure the difference between two sequences of characters. It is defined as the minimum number of single-character insertions, deletions, and substitutions required to transform one string into another.

The recursive Levenshtein algorithm is a recursive method of calculating the Levenshtein distance between two strings. In this method, the distance between two strings is calculated recursively by breaking the problem down into smaller subproblems. The recursion stops when the strings are reduced to empty strings or single characters, at which point the Levenshtein distance is calculated using simple operations.

CODE:

```
package levenshteindistancerecursive;
import java.util.*;
class LevenshteinDistanceRecursive {
static int compute_Levenshtein_distance(String str1, String str2)
{
if (str1.isEmpty())
```

```
{
return str2.length();
}
if (str2.isEmpty())
{
return str1.length();
}
int replace = compute_Levenshtein_distance( str1.substring(1), str2.substring(1)) +
NumOfReplacement(str1.charAt(0),str2.charAt(0));
int insert = compute_Levenshtein_distance(
str1, str2.substring(1))+ 1;
int delete = compute_Levenshtein_distance(
str1.substring(1), str2)+ 1;
return minm_edits(replace, insert, delete);
}
static int NumOfReplacement(char c1, char c2)
{
return c1 == c2 ? 0 : 1;
}
static int minm_edits(int... nums)
{
return Arrays.stream(nums).min().orElse(
Integer.MAX_VALUE);
}
public static void main(String args[])
{
String s1 = "glomax";
String s2 = "folmax";
System.out.println(compute_Levenshtein_distance(s1, s2));
}
}
```

PRACTICAL 5:

Map reduce to count occurrences of each character

Theory : MapReduce is a programming model and framework used to process and analyze large datasets in a distributed computing environment. It was developed by Google and has become popular in the field of big data analytics.

The MapReduce programming model consists of two phases: the map phase and the reduce phase. In the map phase, the input data is divided into smaller chunks and processed by multiple map functions in parallel. Each map function takes a set of key-value pairs as input and outputs an intermediate set of key-value pairs.

In the reduce phase, the intermediate key-value pairs generated by the map functions are grouped by key and processed by multiple reduce functions in parallel. Each reduce function takes a key and a set of values as input and outputs a final set of key-value pairs.

The MapReduce framework provides a way to distribute the data and processing across multiple nodes in a cluster, allowing for large datasets to be processed quickly and efficiently. The framework automatically handles data partitioning, scheduling, and fault tolerance, making it easy to develop and scale data processing applications.

Code:

```
Text = """MapReduce is a processing technique and a program modelfor distributed computing based on java.

The MapReduce algorithm contains twoimportant tasks, namely Map and Reduce.

Map takes a set of data and converts it into another set of data, where individualelements are broken down into tuples(key/value pairs).

Secondly, reduce task, which takes the output from a map as an inputand combines those data tuples into a smaller set of tuples.

As the sequence of thename 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 inputdata is in the form of file or directory and is stored in theHadoop 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 -The stage is the combination of the Shuffle stage and the Reduce stage.

The Reduced'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."""

for char in '-.,\n':
    Text = Text.replace(char,' ')

Text = Text.lower()

word_list = Text.split()
```

```
from collections import Counter

Counter(word_list).most_common()

d={}

for word in word_list:

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

word_freq = []

for key, value in d.items():

   word_freq.append((value,key))

word_freq.sort(reverse=True)

print(word_freq)
```

PRACTICAL 7

STOP WORD REMOVAL

Theory :

In information retrieval, stop words are common words that are often removed from search queries and documents because they do not provide much information about the content of the document or query. Examples of stop words include "the", "a", "an", "and", "or", "but", "of", "to", "in", and "for".

There are several methods for removing stop words from text in information retrieval:

1. Dictionary-based approach: Stop words are identified using a pre-defined list of stop words, and any occurrence of those words in the text is removed.
2. Frequency-based approach: Stop words are identified based on their frequency of occurrence in the text. Words that occur frequently but do not provide much meaning are removed.
3. Part of speech (POS)-based approach: Stop words are identified based on their part of speech, such as articles, prepositions, and conjunctions. These parts of speech are often considered less important for information retrieval.
4. Machine learning-based approach: A machine learning algorithm is trained to identify stop words based on their frequency of occurrence and their relationship to other words in the text.

Code:

```
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.lower() 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)
```

PRACTICAL 8

Using twitter to identify tweets and date/period

Theory :

Pandas is an open-source Python library for data manipulation and analysis. It provides data structures and functions to work with structured data, such as tables or spreadsheets, and is particularly useful for working with tabular data.

Some of the key features of Pandas include:

- Data structures: Pandas provides two main data structures - Series (one-dimensional) and DataFrame (two-dimensional) - to store and manipulate data.
- Data cleaning and preparation: Pandas has functions for handling missing data, removing duplicates, and transforming data.
- Data analysis and manipulation: Pandas provides functions for filtering, grouping, sorting, and aggregating data. It also supports merging and joining multiple datasets.
- Time series analysis: Pandas has built-in support for time series data, allowing for easy manipulation and analysis of time-based data.

Pandas is widely used in data analysis and data science projects, particularly for tasks such as data cleaning, data wrangling, and exploratory data analysis. It is also often used in conjunction with other libraries in the Python data science ecosystem, such as NumPy, Matplotlib, and Scikit-learn.

Code :

```
import os

import pandas as pd

from datetime import date


today = date.today()

end_date = today

print(end_date)
```

```
search_term = 'Elon Musk'
from_date = '2022-02-01'

os.system(f"snscrape --since {from_date} twitter-search '{search_term} until:{end_date}' > result-tweets.txt")

if os.stat("result-tweets.txt").st_size == 0:
    counter = 0
else:
    df = pd.read_csv('result-tweets.txt')
    counter = df.size

print('Number Of Tweets : '+ str(counter))

max_results = 100
extracted_tweets = "snscrape --format '{content!r}'"+ f" --max-results {max_results} --since {from_date} twitter-search '{search_term} until:{end_date}' > extracted-tweets.txt"

os.system(extracted_tweets)

if os.stat("extracted-tweets.txt").st_size == 0:
    print('No Tweets found')
else:
    df = pd.read_csv('extracted-tweets.txt', names=['content'])
    for row in df['content'].iteritems():
        print(row)
```

PRACTICAL 8

Web crawler

Theory : A web crawler, also known as a spider or spiderbot, is an automated program or script that systematically browses the World Wide Web, typically for the purpose of indexing and gathering information about web pages. The primary function of a web crawler is to collect information from the web and make it available to search engines, which can then use that information to improve search results.

There are several different types of web crawlers:

1. General purpose crawlers: These crawlers are designed to crawl the entire web and collect information about every page they encounter. Examples of general purpose crawlers include Googlebot and Bingbot.
2. Focused crawlers: These crawlers are designed to crawl a specific subset of the web, such as a particular domain or set of websites. Focused crawlers are often used for research or monitoring specific topics or industries.
3. Incremental crawlers: These crawlers are designed to crawl only new or updated content since the last crawl, in order to keep search results up to date.
4. Distributed crawlers: These crawlers use multiple machines to crawl the web in parallel, in order to increase efficiency and speed.
5. Deep web crawlers: These crawlers are designed to access and index content that is not easily accessible through conventional search engines, such as dynamically generated web pages or password-protected content.
6. Extractor crawlers: These crawlers are designed to extract specific types of data from web pages, such as email addresses or product information

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

```
import re
from bs4 import BeautifulSoup as sp
from urllib.request import urlopen
Crawl = urlopen("https://en.wikipedia.org/wiki/Swami_Vivekananda")
bsobj = sp(Crawl.read(),'lxml')
for link in bsobj.find_all('a'):
    if 'href' in link.attrs:
        print(link.attrs['href'])
print('******')
```

```
for link in bsobj.find('div',{'id':'bodyContent'}).find_all('a',href = re.compile('^(/wiki/)((?!:).)*$')):
    if 'href' in link.attrs:
        print(link.attrs['href'])
```

PRACTICAL FOR SIMILARITY OF WORDS BTWN TWO FILES

Theory

In Information Retrieval (IR), similarity between two files is often measured by comparing their vector representations using similarity metrics such as cosine similarity.

To represent a file as a vector, we can use a bag-of-words (BoW) model, which counts the frequency of each word in the file and represents the file as a vector where each dimension corresponds to a unique word in the file collection.

Once we have the vector representations of the two files, we can calculate their cosine similarity. The cosine similarity measures the cosine of the angle between two vectors and ranges from -1 to 1, where a value of 1 indicates that the vectors are identical, and a value of -1 indicates that the vectors are completely dissimilar.

Therefore, in IR, similarity between two files is determined by how many common words they share and how often these words occur in each file. The more similar the files are, the closer their vector representations will be, and the higher their cosine similarity will be.

CODE :

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
X = open('file1.txt','r').read()
Y= open('file2.txt','r').read()
X_list = word_tokenize(X)
Y_list = word_tokenize(Y)
sw=stopwords.words('english')
l1=[]; l2=[]
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}
rvector = X_set.union(Y_set)
for w in rvector:
    if w in X_set:
        l1.append(1)
```

```
    else:
        l1.append(0)
    if w in Y_set:
        l2.append(1)
    else:
        l2.append(0)
c = 0
for i in range(len(rvector)):
    c+=l1[i]*l2[i]
cosine= c/float((sum(l1)*sum(l2))**0.5)
print("similarity : ",cosine)
```