Wuhan University of technology

School of computer science and technology

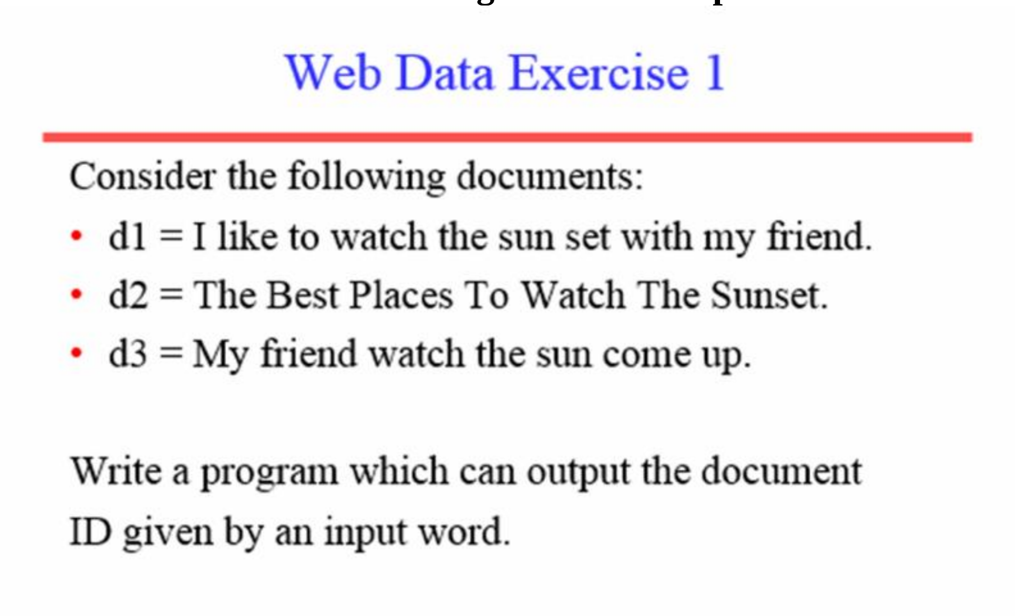
WEB DATA MANAGEMENT

01/07/2015

LAB REPORT

Submitted by: OUMOUSS EL MEHDI
Student No: 2014Y90100063
TEL No: 13125029535
Email: oumoussmehdi@hotmail.com
Submitted to: Prof. Dr. LI LIN

Web Data Management Lab Reports



Web Data Exercise 1

Consider the following documents:

- d1 = I like to watch the sun set with my friend.
- d2 = The Best Places To Watch The Sunset.
- d3 = My friend watch the sun come up.

Write a program which can output the document ID given by an input word.

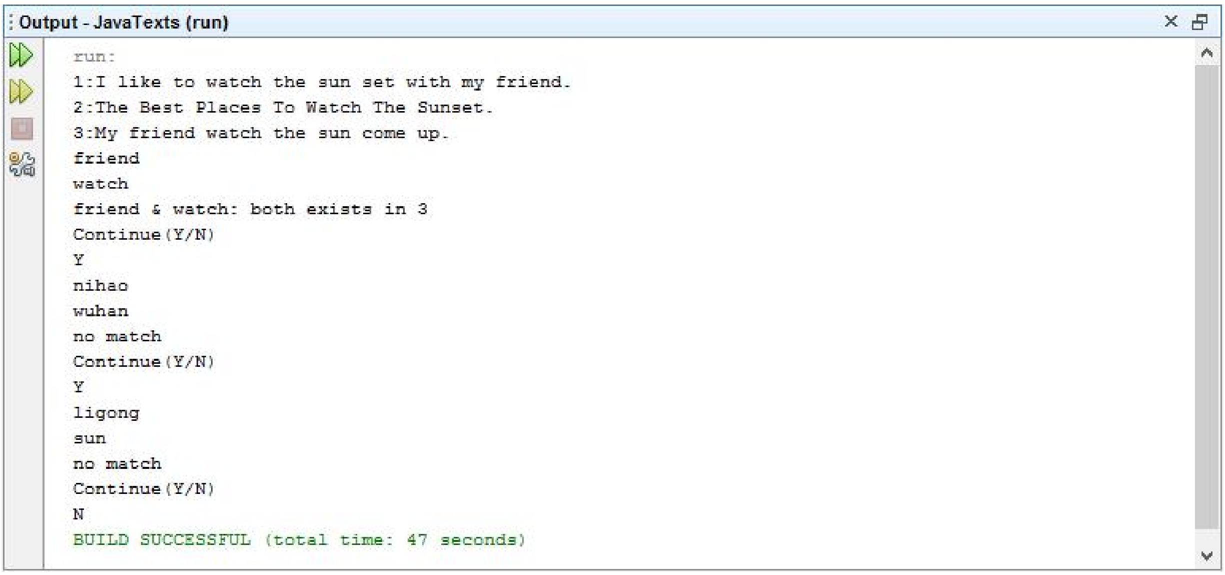
ALGORITHM:

The idea behind the code below, is to use a map where we are going to affect an id (key) to each document (value). Then we should iterate through the map. Each time we will get the value which contain the document (String) and try to find a match between the text in document and our input which is in the form of two words.

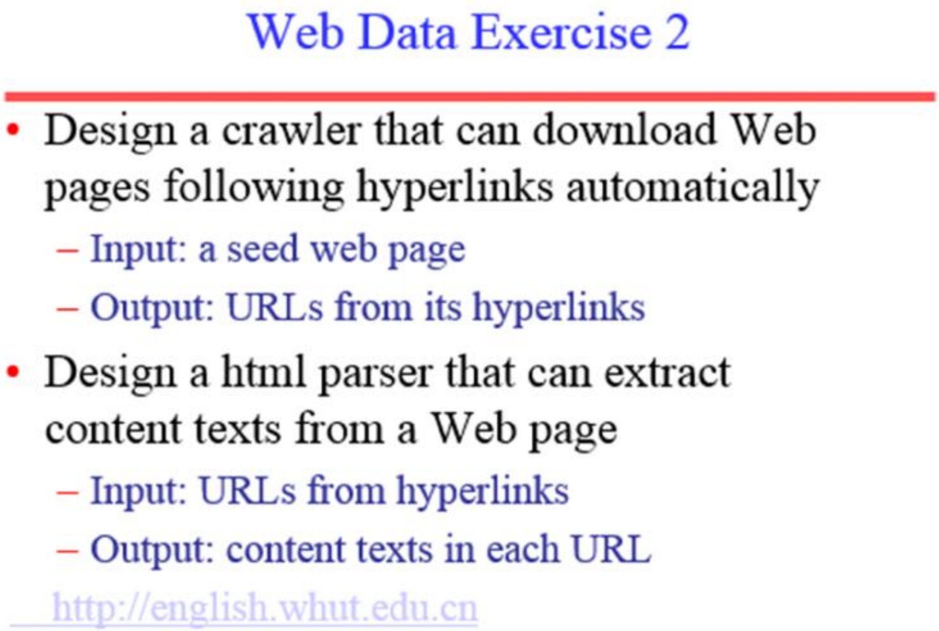
SOURCE CODE:

```java
package exercise1;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Scanner;

public class Exercise1 {

    static HashMap<Integer,String> documents = new HashMap();

    public static void main(String[] args) {

        //Documents:
        String d1 = "I like to watch the sun set with my friend.";
        String d2 = "The Best Places To Watch The Sunset.";
        String d3 = "My friend watch the sun come up.";

        //we use a map to affect an id(key) to each document
        documents.put(1, d1);
        documents.put(2, d2);
        documents.put(3, d3);
        
        //printing the document ( key : value (text) )
        for(Map.Entry<Integer, String> doc : documents.entrySet())
        {
            System.out.println(doc.getKey() + ":" + doc.getValue());
        }
        
        String a="";
        do{
            //a scanner to read our input (2 words)
            Scanner sc = new Scanner(System.in);
            String inputWord1 = sc.nextLine();
            String inputWord2 = sc.nextLine();
            
             //the method that will try to find the words in the documents
            toFind(documents,inputWord1,inputWord2);
            
            System.out.println("Continue(Y/N)");
            a = sc.next();
            }while(a.equalsIgnoreCase("Y"));
        }
    
    static void toFind(HashMap documents, String inputWord1,String inputWord2) {
        boolean b = false; // a flag that will determine if there is a match or not
        
        for (Iterator it = documents.entrySet().iterator(); it.hasNext();) {
            
            Map.Entry<Integer, String> doc = (Map.Entry<Integer, String>) it.next();
            String[] words = doc.getValue().split(" ");
            
            outerloop:
            for (String w1 : words) {
                if (w1.equalsIgnoreCase(inputWord1)) {
                    for (String w2 : words)
                        if(w2.equalsIgnoreCase(inputWord2)){
                            System.out.println(inputWord1 + " & " + inputWord2
                                    + ": both exists in " + doc.getKey());
                            //if we find the two words than just break the outerloop
                            b=true;
                            break outerloop;
                        }
                }
            }
        }
         if(b==false) System.out.println("no match");
    }
    
}
```

<u>**OUTPUT:**</u>



```
run:
1:I like to watch the sun set with my friend.
2:The Best Places To Watch The Sunset.
3:My friend watch the sun come up.
friend
watch
friend & watch: both exists in 3
Continue(Y/N)
Y
nihao
wuhan
no match
Continue(Y/N)
Y
ligong
sun
no match
Continue(Y/N)
N
BUILD SUCCESSFUL (total time: 47 seconds)
```



Web Data Exercise 2

- Design a crawler that can download Web pages following hyperlinks automatically
  - Input: a seed web page
  - Output: URLs from its hyperlinks
- Design a html parser that can extract content texts from a Web page
  - Input: URLs from hyperlinks
  - Output: content texts in each URL

http://english.whut.edu.cn

ALGORITHM:

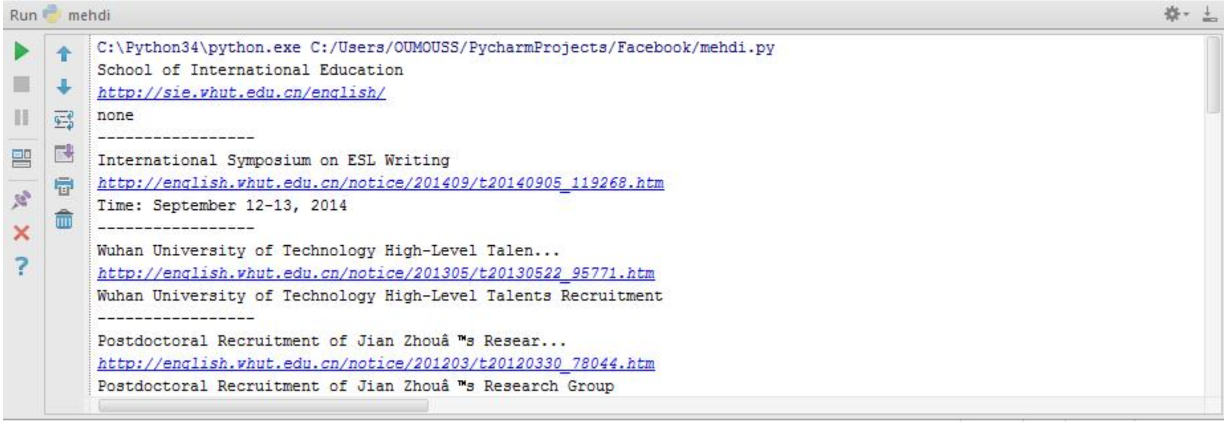
In this exercise, we are going to crawl the web site : http://english.whut.edu.cn . in order to get all the urls available in the page but specifically in this propose solution the urls contained in a `<a>…</a>` tag with a `'target'` equals to `'_blank'.`
After this, we are going to use each one of the urls retrieved and access its html code and extract the first paragraph available (text).

Note: This time we will use Python, personally I feel that Python is easier than Java when it comes to web crawling. The library that helps to pull data out of html and xml files in Python is called Beautiful Soup, in Java it is called JSoup.

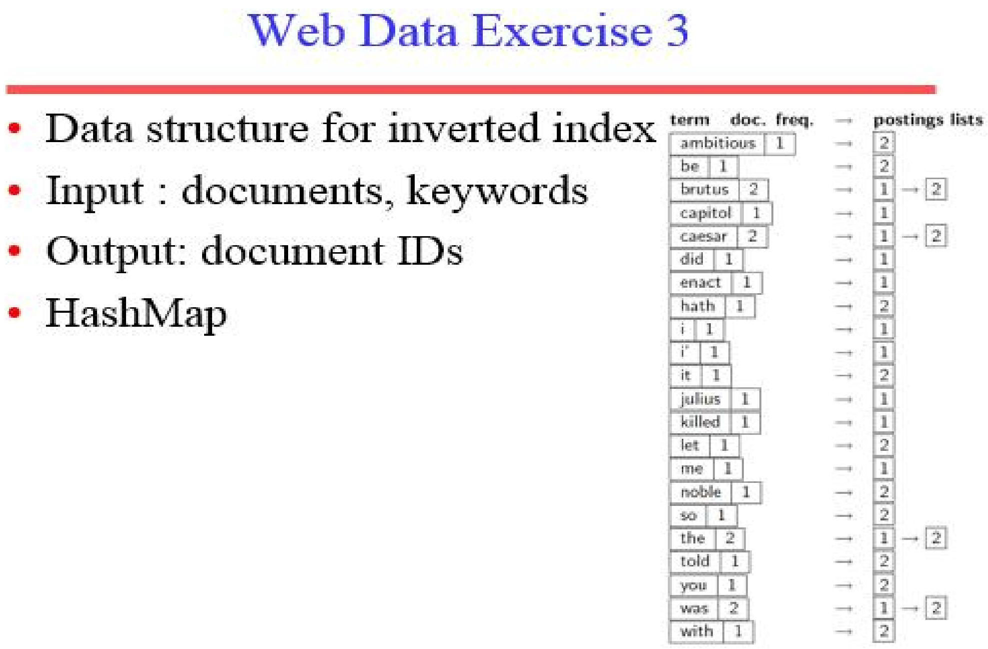
SOURCE CODE:

```
import requests
from bs4 import BeautifulSoup

#the website we are going to crawl
url = http://english.whut.edu.cn/

def web_spider():
        source_code = requests.get(url)

         #in plain_text we store the html page
        plain_text = source_code.text
        #we transform the text into a beautifulsoup object
        soup = BeautifulSoup(plain_text)
        #find all <a> tags, with target equals to _blank
        for link in soup.findAll('a',{'target':'_blank'}):
            #extract the href
            urlEnd = link.get('href')
            '''in this web page some links are in this form:
            <a href="./wn/201405/t20140526_116874.html" target="_blank"...</a>
            The href in no complete and starts with a point
            so we need to subtract the point and add the href at the end of the
            source web page'''
            if urlEnd.startswith("."):
                urlEnd = urlEnd[1:]
                href = "http://english.whut.edu.cn" + urlEnd
            else:
                #some links are just correct and do not need any correction
                href=urlEnd

            title = link.string
            print(title)
            print(href)
            get_text(href)
            print("-----------------")

#the function that will parse the html code and extract text
def get_text(item_url):
    source_code = requests.get(item_url)
    plain_text = source_code.text
    soup = BeautifulSoup(plain_text)

    #find the first paragraph
    p = soup.find('p')
    if(p):
        print(p.text)
    else:
        print('none')

#call
web_spider()
```

OUTPUT:



```
C:\Python34\python.exe C:/Users/OUMOUSS/PycharmProjects/Facebook/mehdi.py
School of International Education
http://sie.whut.edu.cn/english/
none
-----------------
International Symposium on ESL Writing
http://english.whut.edu.cn/notice/201409/t20140905_119268.htm
Time: September 12-13, 2014
-----------------
Wuhan University of Technology High-Level Talen...
http://english.whut.edu.cn/notice/201305/t20130522_95771.htm
Wuhan University of Technology High-Level Talents Recruitment
-----------------
Postdoctoral Recruitment of Jian Zhouâ ™s Resear...
http://english.whut.edu.cn/notice/201203/t20120330_78044.htm
Postdoctoral Recruitment of Jian Zhouâ ™s Research Group
```



Web Data Exercise 3

- Data structure for inverted index
- Input : documents, keywords
- Output: document IDs
- HashMap

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

INVERTED INDEX

An **inverted index** (also referred to as postings file or inverted file) is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents. The purpose of an inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database.
The inverted file may be the database file itself, rather than its index. It is the most popular data structure used in document retrieval systems. Used on a large scale for example in search engines.

[Wikipedia]

ALGORITHM:

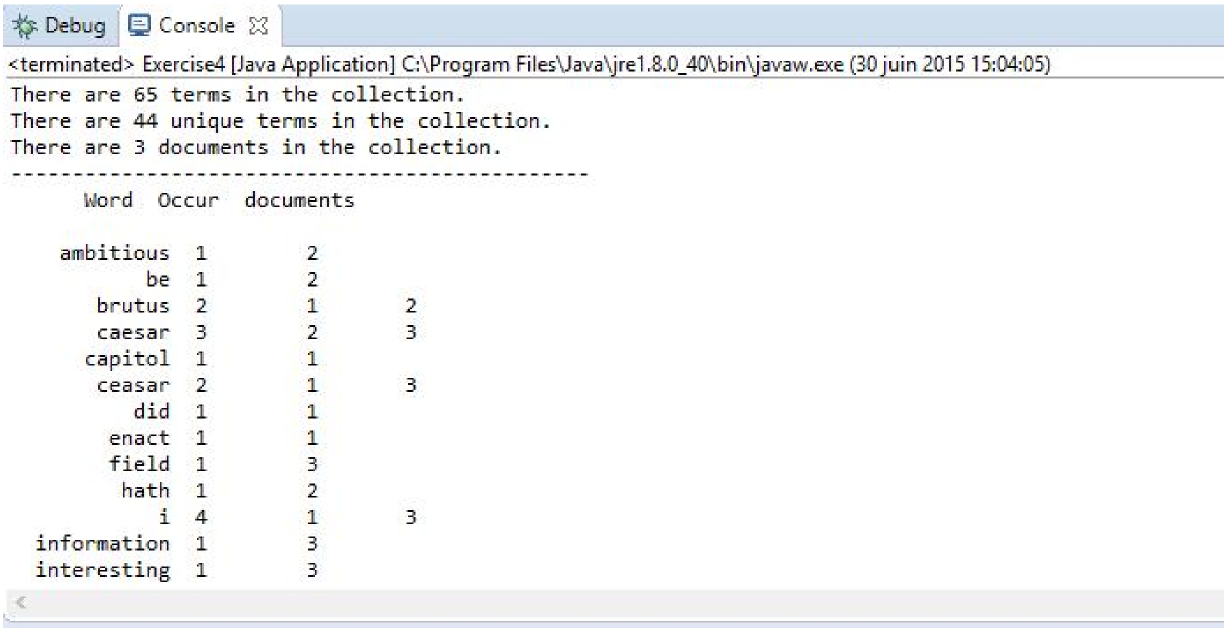
Here we are implementing the inverted index.

SOURCE CODE:

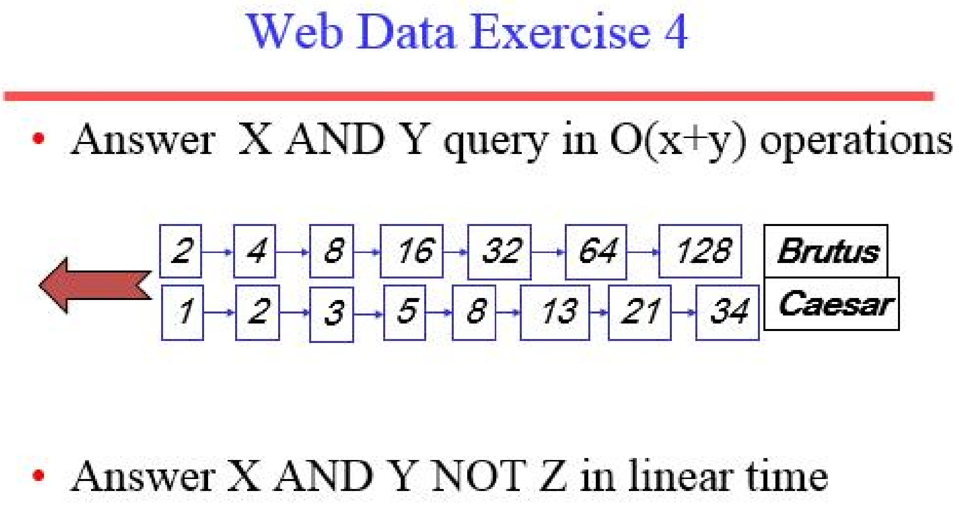
```
package exercise4;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.TreeMap;
import java.util.TreeSet;

public class InvertedIndex {
      //here is the main folder where we put the files to index
      static File path = new File("C:/Users/OUMOUSS/Desktop/Files");
      //paths is a table that contains all the files
      static String[] paths = path.list();
      //fileToId is a map where we affect to each file a unique id
      static Map<String,Integer> fileToId = new HashMap<String,Integer>();
      //wordToFrequency is the map that contains the frequence of each word
      static Map<String, Integer> wordToFrequency = new TreeMap<String, Integer>();
      //wordToFileId: word -> Postings (files ids)
      static Map<String,TreeSet<Integer>> wordToFileId = new
      TreeMap<String,TreeSet<Integer>>();
      // this method initiate fileToId
      public static void init() throws IOException {

             for (String p : paths) {
                   fileToId.put(p,fileToId.size()+1);
             }
             fileToId = sortByValue(fileToId);
             readWordFileId(wordToFrequency,wordToFileId);
             printI(wordToFrequency,wordToFileId);
             }
      //this is the method that will fill our wordToFrequency and wordToFileId
      private static void readWordFileId(Map<String, Integer> wordToFrequency,
                   Map<String, TreeSet<Integer>> wordToFileId) throws
FileNotFoundException {
             int total = 0;
             Scanner wordFile;
             String word; // A word read from the p
             Integer count; // The number of occurrences of the word
             int counter = 0;
             int docs = 0;


              for (String p : paths) {
                   wordFile = new Scanner(new File(path+"/"+p));

                   while (wordFile.hasNext()) {
                          word = wordFile.next();
                          word = word.toLowerCase();

                          // Get the current count of this word, add one, and then
store
                          // the new count:
                          count = getCount(word, wordToFrequency) + 1;
                          wordToFrequency.put(word, count);
                          total = total + count;
                          counter++;
                          docs = paths.length;

                          if (wordToFileId.containsKey(word)) { // The word has
occurred
                                (wordToFileId.get(word)).add(fileToId.get(p));
```

```
                } else { // No occurrences of this word

                    wordToFileId.put(word, new TreeSet<Integer>());
                    (wordToFileId.get(word)).add(fileToId.get(p));
                }
            }

        } // End of for loop
        System.out.println("There are " + total + " terms in the collection.");
        System.out.println("There are " + counter
                + " unique terms in the collection.");
        System.out.println("There are " + docs
                + " documents in the collection.");

    }

    //printI: print the inverted index structure
    private static void printI(Map<String, Integer> wordToFrequency,
            Map<String, TreeSet<Integer>> wordToFileId) {
        System.out.println("-----------------------------------------------");
        System.out.println("      Word  Occur  documents  ");

        for (String word : wordToFrequency.keySet()) {
            System.out.printf("\n %12s  %d \t", word,
wordToFrequency.get(word));
            Iterator it = (wordToFileId.get(word)).iterator();
            while(it.hasNext())
                System.out.print(it.next()+"\t");
        }

        System.out.println("\n -----------------------------------------------
");
    }

    public static int getCount(String word, Map<String, Integer> wordToFrequency) {
        if (wordToFrequency.containsKey(word)) {
            // The word has occurred before, so get its count from the map
            return wordToFrequency.get(word); // Auto-unboxed
        } else { // No occurrences of this word
            return 0;
        }
    }

    //this method helps sort a map by its value
    public static Map sortByValue(Map map) {
         List list = new LinkedList(map.entrySet());
         Collections.sort(list, new Comparator() {
              public int compare(Object o1, Object o2) {
                   return ((Comparable) ((Map.Entry) (o1)).getValue())
                  .compareTo(((Map.Entry) (o2)).getValue());
              }
         });

        Map result = new LinkedHashMap();
        for (Iterator it = list.iterator(); it.hasNext();) {
            Map.Entry entry = (Map.Entry)it.next();
            result.put(entry.getKey(), entry.getValue());
        }
        return result;
    }

    // return the postings of the word s
    public static TreeSet<Integer> getPosting(String s){
          return wordToFileId.get(s);
    }

}
```

OUTPUT:



```
<terminated> Exercise4 [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (30 juin 2015 15:04:05)
There are 65 terms in the collection.
There are 44 unique terms in the collection.
There are 3 documents in the collection.
-----------------------------------------------
      Word  Occur  documents

    ambitious  1        2
           be  1        2
       brutus  2        1       2
       caesar  3        2       3
      capitol  1        1
       ceasar  2        1       3
          did  1        1
        enact  1        1
        field  1        3
         hath  1        2
            i  4        1       3
  information  1        3
  interesting  1        3
```



Web Data Exercise 4

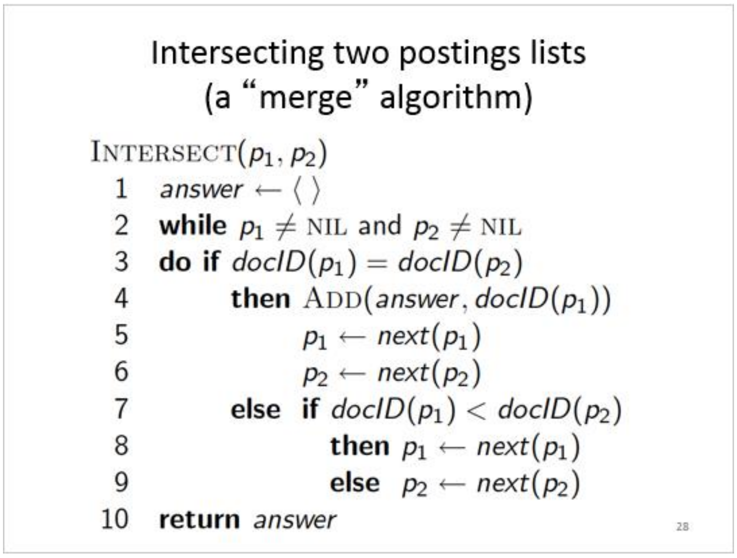
- Answer X AND Y query in O(x+y) operations

- Answer X AND Y NOT Z in linear time

ALGORITHM:

This exercise’s solution is in the same package as the last exercise’s solution. In other word, we are going to use the inverted index in order to answer X AND Y query (in linear time) using the implementation of the next algorithm.

The below code source includes comments.



Intersecting two postings lists
(a “merge” algorithm)

```
INTERSECT($p_1, p_2$)
  answer ← ⟨ ⟩
  while $p_1 \neq$ NIL and $p_2 \neq$ NIL
  do if $docID(p_1) = docID(p_2)$
        then ADD(answer, $docID(p_1)$)
             $p_1 \leftarrow next(p_1)$
             $p_2 \leftarrow next(p_2)$
        else if $docID(p_1) < docID(p_2)$
                then $p_1 \leftarrow next(p_1)$
                else $p_2 \leftarrow next(p_2)$
  return answer
```

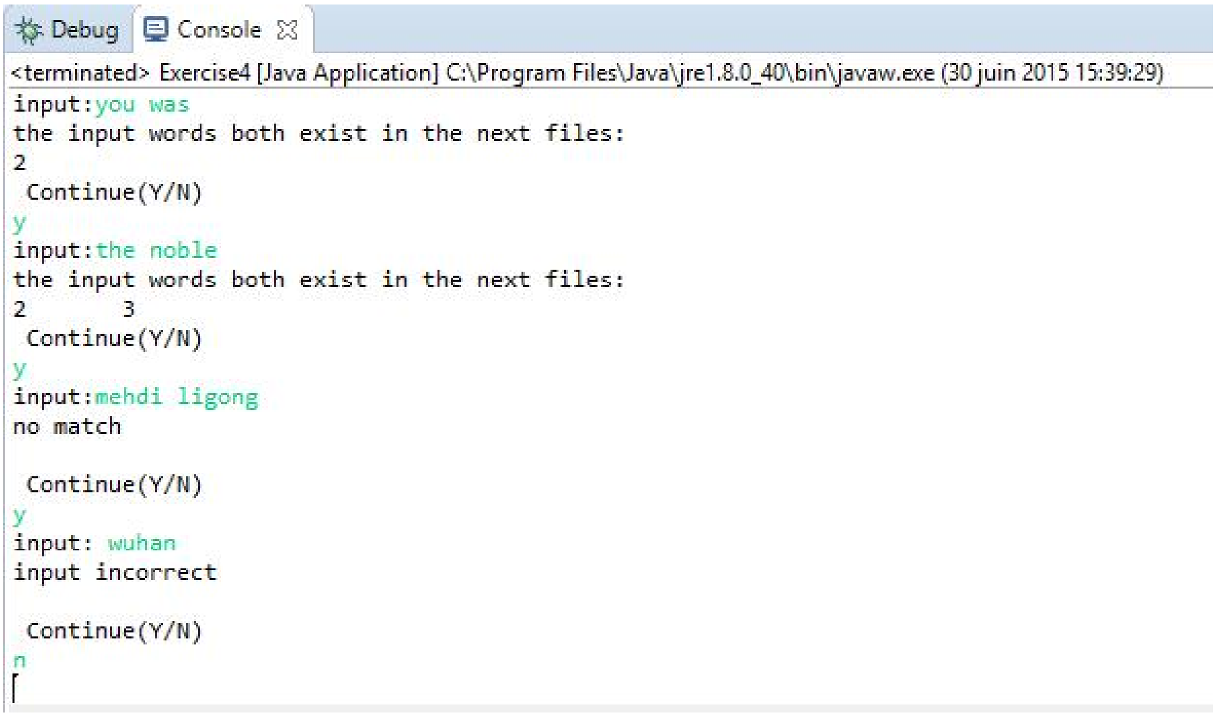
SOURCE CODE:

```
package exercise4;

import java.io.IOException;
import java.util.Iterator;
import java.util.Scanner;
import java.util.TreeSet;

public class Exercise4 {

    public static void main(String[] args) throws IOException {
        InvertedIndex i = new InvertedIndex();
        i.init();// initiate
        String a = "";
        do {
            // a scanner to read our input (2 words separated by space)
            Scanner sc = new Scanner(System.in);
            loop: try {
                System.out.print("input:");
                String inputWord = sc.nextLine();
                String[] parts = inputWord.split("\\s+");
                if (parts.length != 2)
                    throw new IllegalArgumentException(
                            "String not in correct format");
                String part1 = parts[0];
                String part2 = parts[1];
                if (part1.isEmpty() || part2.isEmpty()) {
                    throw new IllegalArgumentException();
                }

            // the method that will try to find the words in the documents
                toFind(i, part1, part2);
            } catch (IllegalArgumentException e) {
                System.out.println("input incorrect");
                break loop;
                // continue;
            }
            System.out.println("\n Continue(Y/N)");
            a = sc.next();
        } while (a.equalsIgnoreCase("Y"));
    }

    //toFind : search the two inputs in the invertedindex
    private static void toFind(InvertedIndex i, String inputWord1, String
inputWord2) {
        TreeSet<Integer> ts1 = i.getPosting(inputWord1);
        TreeSet<Integer> ts2 = i.getPosting(inputWord2);
        if (ts1 != null && ts2 != null) {
            TreeSet<Integer> answer = new TreeSet<Integer>();
            answer = intersection(ts1, ts2);
            if (!answer.isEmpty()) {
                System.out
                        .println("the input words both exist in the
next files:");
```

```
                printTree(answer);
            } else {
                System.out.println("no match");
            }

        } else {
            System.out.println("no match");
        }

    }

    //intersection is the implementation of the linear merge between two postings
    public static TreeSet<Integer> intersection(TreeSet<Integer>
t1,TreeSet<Integer> t2) {
        TreeSet<Integer> ts = new TreeSet<Integer>();
        Iterator<Integer> it1 = t1.iterator();
        Iterator<Integer> it2 = t2.iterator();
        int a = (int) it1.next();
        int b = (int) it2.next();
        while (it1 != null && it2 != null) {
            if (a == b) {
                ts.add(a);
                if (it1.hasNext()) {
                    a = (int) it1.next();
                } else {
                    it1 = null;
                }
                if (it2.hasNext()) {
                    b = (int) it2.next();
                } else {
                    it2 = null;
                }
            } else {
                if (a < b) {
                    if (it1.hasNext())
                        a = (int) it1.next();
                    else
                        it1 = null;
                } else {
                    if (it2.hasNext())
                        b = (int) it2.next();
                    else
                        it2 = null;
                }
            }
        }
        return ts;
    }

    //printTree: just print any treeSet given
    public static void printTree(TreeSet t) {
        Iterator it = t.iterator();
        while (it.hasNext())
            System.out.print(it.next() + "\t");
    }
```

}

OUTPUT:



```
<terminated> Exercise4 [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (30 juin 2015 15:39:29)
input:you was
the input words both exist in the next files:
2
 Continue(Y/N)
y
input:the noble
the input words both exist in the next files:
2       3
 Continue(Y/N)
y
input:mehdi ligong
no match

 Continue(Y/N)
y
input: wuhan
input incorrect

 Continue(Y/N)
n
```

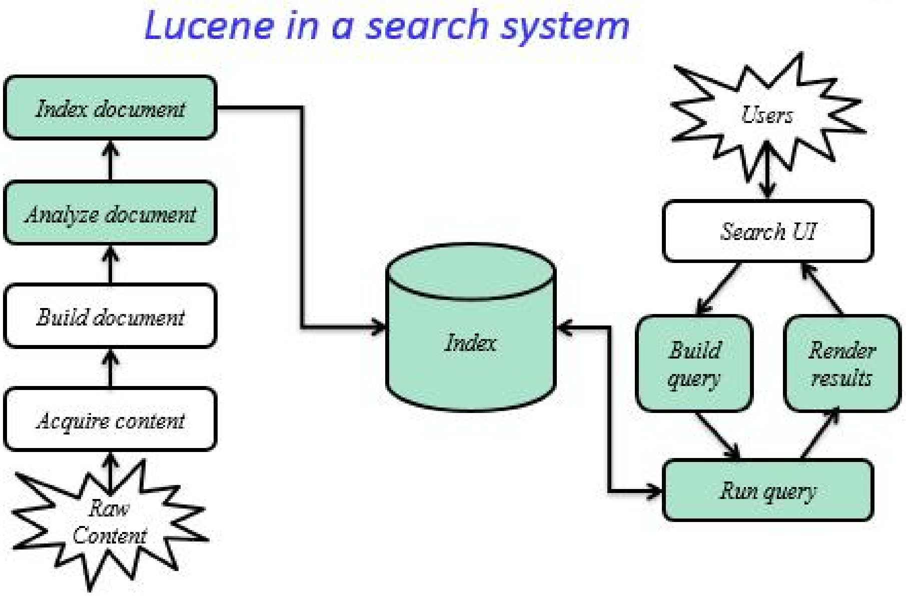
Note:

The input entry should be in the form of **two word separated by** space. Additionaly the input entry is sensitive to space. Because we used the input.split("\\s+") method; (e.g. wuhan and _wuhan are two different cases) This code can just output document ids where both words figure.

Improvement:

- We can manage so that entry input is no more space sensitive at the beginning of input.
- We can improve the system so that we are able to input indetermined number of words.

Web Data Exercise 5


Lucene in a search system

LUCENE:

Apache Lucene™ is a high-performance, full-featured text search engine library written entirely in Java by Doug Cutting. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

Apache Lucene is an open source project available for free download. It is supported by the Apache Software Foundation and is released under the Apache Software License.

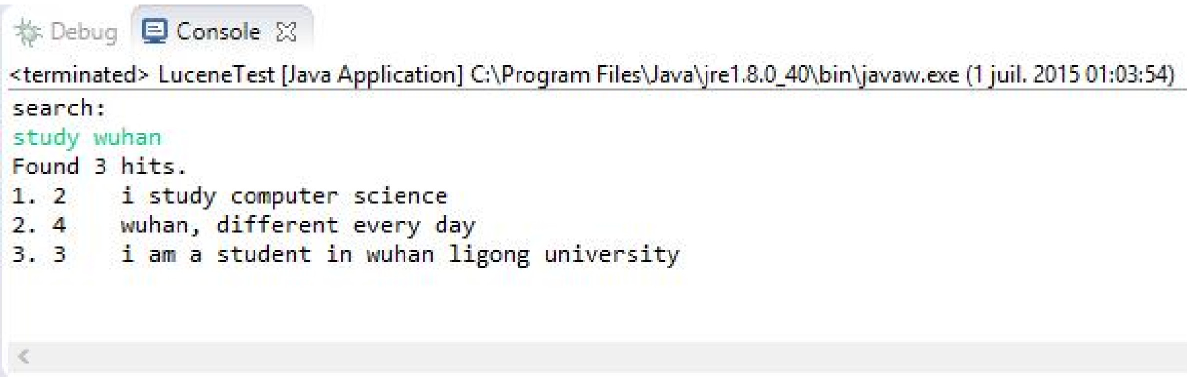
Lucene has been ported to other programming languages including Perl, C#, C++, Python,Ruby, and PHP.

SOURCE CODE:

```
package lucene;

import java.io.IOException;
import java.util.Scanner;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.StringField;
import org.apache.lucene.document.TextField;
import org.apache.lucene.index.DirectoryReader;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.index.IndexWriterConfig;
import org.apache.lucene.queryparser.classic.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopScoreDocCollector;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.RAMDirectory;
import org.apache.lucene.util.Version;


public class LuceneTest
{
      public static void main(String[] args)
      {
            Scanner sc = new Scanner(System.in);
            try
            {
                  //     Specify the analyzer for tokenizing text.
                //The same analyzer should be used for indexing and searching
                  StandardAnalyzer analyzer = new
StandardAnalyzer(Version.LUCENE_4_10_2);

                  //     Code to create the index
                  Directory index = new RAMDirectory();

                  IndexWriterConfig config = new
IndexWriterConfig(Version.LUCENE_4_10_2, analyzer);

                  IndexWriter w = new IndexWriter(index, config);
                  addDoc(w, "hi my name is xiaomai.", "1");
                  addDoc(w, "i study computer science", "2");
                  addDoc(w, "i am a student in wuhan ligong university", "3");
                  addDoc(w, "wuhan, different every day", "4");
                  addDoc(w, "enjoy you studies", "5");
                  addDoc(w, "are you a student?", "6");
                  w.close();

                  //     Text to search
                  System.out.println("search:");
                  String s = sc.nextLine();
                  String querystr = args.length > 0 ? args[0] : s;

                  //     The \"sentence\" arg specifies the default field to use
when no field is explicitly specified in the query
                  Query q = new QueryParser(Version.LUCENE_4_10_2, "sentence",
analyzer).parse(querystr);

                  // Searching code
                  int hitsPerPage = 10;
                IndexReader reader = DirectoryReader.open(index);
                IndexSearcher searcher = new IndexSearcher(reader);
                TopScoreDocCollector collector =
TopScoreDocCollector.create(hitsPerPage, true);
                searcher.search(q, collector);
                ScoreDoc[] hits = collector.topDocs().scoreDocs;

                //Code to display the results of search
                System.out.println("Found " + hits.length + " hits.");
                for(int i=0;i<hits.length;++i)
```

```
                 {
                   int docId = hits[i].doc;
                   Document d = searcher.doc(docId);
                   System.out.println((i + 1) + ". " + d.get("id") + "\t" +
d.get("sentence"));
                 }

                 // reader can only be closed when there is no need to access the
documents any more
                 reader.close();
             }
             catch(Exception e)
             {
                   System.out.println(e.getMessage());
             }
      }
      private static void addDoc(IndexWriter w, String sentence, String id) throws
IOException
      {
               Document doc = new Document();
               // A text field will be tokenized
               doc.add(new TextField("sentence", sentence, Field.Store.YES));
               // We use a string field for id because we don\'t want it tokenized
               doc.add(new StringField("id", id, Field.Store.YES));
               w.addDocument(doc);
      }
}
```

OUTPUT:



```
<terminated> LuceneTest [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (1 juil. 2015 01:03:54)
search:
study wuhan
Found 3 hits.
1. 2    i study computer science
2. 4    wuhan, different every day
3. 3    i am a student in wuhan ligong university
```

EXERCISE: TF-IDF & COSINE SIMILARITY

TfIdfMain

```java
package com.mehdi.tfidf;

import java.io.FileNotFoundException;
import java.io.IOException;

//import javax.swing.text.html.parser.DocumentParser;


public class TfIdfMain {

    /**
     * Main method
     * @param args
     * @throws FileNotFoundException
     * @throws IOException
     */
    public static void main(String args[]) throws FileNotFoundException, IOException
    {
        DocumentParser dp = new DocumentParser();
        dp.parseFiles("C:/Users/OUMOUSS/Desktop/F");
        dp.tfIdfCalculator(); //calculates tfidf
        dp.getCosineSimilarity(); //calculated cosine similarity
        System.out.println(dp.tfidfDocsVector);
    }
}
```

DocumentParser

```java
package com.mehdi.tfidf;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class DocumentParser {

    //This variable will hold all terms of each document in an array.
    private List<String[]> termsDocsArray = new ArrayList<String[]>();
    private List<String> allTerms = new ArrayList<String>(); //to hold all terms
    List<double[]> tfidfDocsVector = new ArrayList<double[]>();

    /**
     * Method to read files and store in array.
     * @param filePath : source file path
     * @throws FileNotFoundException
     * @throws IOException
     */
    public void parseFiles(String filePath) throws FileNotFoundException, IOException
{
        File[] allfiles = new File(filePath).listFiles();
        BufferedReader in = null;
        for (File f : allfiles) {
            if (f.getName().endsWith(".txt")) {
                in = new BufferedReader(new FileReader(f));
                StringBuilder sb = new StringBuilder();
                String s = null;
                while ((s = in.readLine()) != null) {
                    sb.append(s);
                }
                String[] tokenizedTerms = sb.toString().replaceAll("[\\W&&[^\\s]]",
"").split("\\W+");   //to get individual terms
                for (String term : tokenizedTerms) {
                    if (!allTerms.contains(term)) {  //avoid duplicate entry
                        allTerms.add(term);
                    }
                }
                termsDocsArray.add(tokenizedTerms);
            }
        }

    }

    /**
     * Method to create termVector according to its tfidf score.
     */
    public void tfIdfCalculator() {
        double tf; //term frequency
        double idf; //inverse document frequency
        double tfidf; //term requency inverse document frequency
        for (String[] docTermsArray : termsDocsArray) {
            double[] tfidfvectors = new double[allTerms.size()];
            int count = 0;
            for (String terms : allTerms) {
                tf = new TfIdf().tfCalculator(docTermsArray, terms);
                idf = new TfIdf().idfCalculator(termsDocsArray, terms);
                tfidf = tf * idf;
                tfidfvectors[count] = tfidf;
                count++;
            }
            tfidfDocsVector.add(tfidfvectors);  //storing document vectors;
        }
    }

    /**
     * Method to calculate cosine similarity between all the documents.
     */
    public void getCosineSimilarity() {
        System.out.println("the matrix with cosine similarity values:");
        for (int i = 0; i < tfidfDocsVector.size(); i++) {
            for (int j = 0; j < tfidfDocsVector.size(); j++) {
                System.out.println("between " + i + " and " + j + "  =  "
                                   + new CosineSimilarity().cosineSimilarity
```

```
                                       (
                                         tfidfDocsVector.get(i),
                                         tfidfDocsVector.get(j)
                                       )
                                  );
            }
        }
    }
    public String toString(){
      //Iterator it = new termsDocsArray.iterator();
      for(Object o:termsDocsArray){
             String s = (String)o;
             for(int i=0;i<s.length();i++){
                   System.out.print(s);
             }
      }
             return null;
      //List<String[]> termsDocsArray
    }

}
```

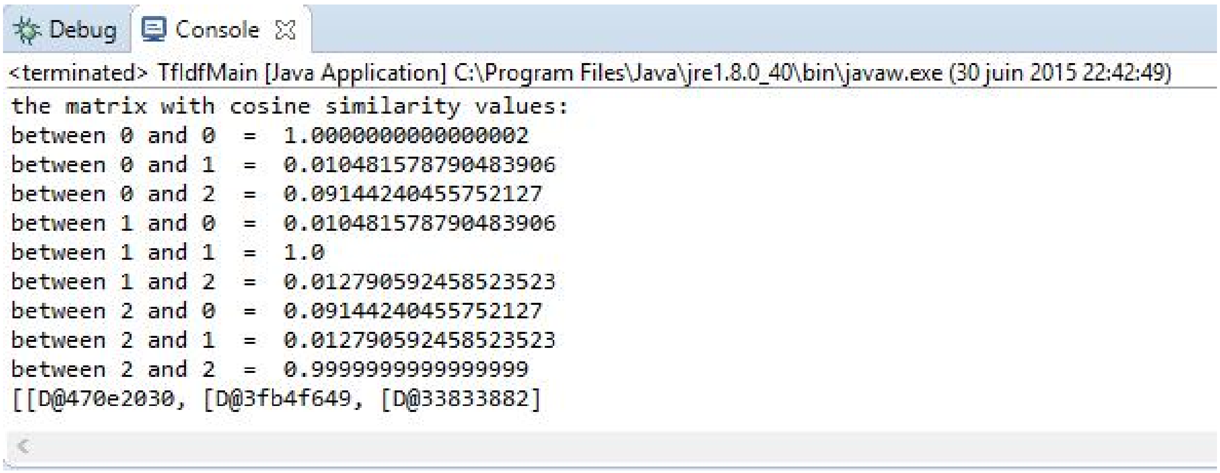
TfIdf

```
package com.mehdi.tfidf;

import java.util.List;


public class TfIdf {

    //<editor-fold defaultstate="collapsed" desc="TF Calculator">
    /**
     * Calculated the tf of term termToCheck
     * @param totalterms : Array of all the words under processing document
     * @param termToCheck : term of which tf is to be calculated.
     * @return tf(term frequency) of term termToCheck
     */
    public double tfCalculator(String[] totalterms, String termToCheck) {
        double count = 0;  //to count the overall occurrence of the term termToCheck
        for (String s : totalterms) {
            if (s.equalsIgnoreCase(termToCheck)) {
                count++;
            }
        }
        return count / totalterms.length;
    }

    /**
     * Calculated idf of term termToCheck
     * @param allTerms : all the terms of all the documents
     * @param termToCheck
     * @return idf(inverse document frequency) score
     */
    public double idfCalculator(List<String[]> allTerms, String termToCheck) {
        double count = 0;
        for (String[] ss : allTerms) {
            for (String s : ss) {
                if (s.equalsIgnoreCase(termToCheck)) {
                    count++;
                    break;
                }
            }
        }
        return Math.log(allTerms.size() / count);
    }
//</editor-fold>
}
//</editor-fold>
```

CosineSimilarity

```
package com.mehdi.tfidf;

public class CosineSimilarity {

    /**
     * Method to calculate cosine similarity between two documents.
     * @param docVector1 : document vector 1 (a)
     * @param docVector2 : document vector 2 (b)
     * @return
     */
    public double cosineSimilarity(double[] docVector1, double[] docVector2) {
        double dotProduct = 0.0;
        double magnitude1 = 0.0;
        double magnitude2 = 0.0;
        double cosineSimilarity = 0.0;

        for (int i = 0; i < docVector1.length; i++) //docVector1 and docVector2 must
be of same length
        {
            dotProduct += docVector1[i] * docVector2[i];  //a.b
            magnitude1 += Math.pow(docVector1[i], 2);  //(a^2)
            magnitude2 += Math.pow(docVector2[i], 2); //(b^2)
        }

        magnitude1 = Math.sqrt(magnitude1);//sqrt(a^2)
        magnitude2 = Math.sqrt(magnitude2);//sqrt(b^2)

        if (magnitude1 != 0.0 | magnitude2 != 0.0) {
            cosineSimilarity = dotProduct / (magnitude1 * magnitude2);
        } else {
            return 0.0;
        }
        return cosineSimilarity;
    }
}
```

OUTPUT:



```
<terminated> TfIdfMain [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (30 juin 2015 22:42:49)
the matrix with cosine similarity values:
between 0 and 0  =  1.0000000000000002
between 0 and 1  =  0.010481578790483906
between 0 and 2  =  0.09144240455752127
between 1 and 0  =  0.010481578790483906
between 1 and 1  =  1.0
between 1 and 2  =  0.012790592458523523
between 2 and 0  =  0.09144240455752127
between 2 and 1  =  0.012790592458523523
between 2 and 2  =  0.9999999999999999
[[D@470e2030, [D@3fb4f649, [D@33833882]
```