Assignment 4: Near Duplicate Detection

The purpose of this assignment was to perform near duplicate detection by means of computing Jaccard coefficient between documents.

Program Structure:

The solution consists of a single java file:

- **DuplicateDetection.java**: The class consists of functions to parse the directory consisting of all the documents and then find out the shingles in each document. It also labels the shingles as they are found out. The class consists of functions to find out the Jaccard coefficient as per the proposed method wherein 25 random a and b values are generated and then the Jaccard coefficients for the document pairs are computed depending upon the sketches prepared for the documents. The class also consists of a function to list the top 3 documents having the highest Jaccard coefficient with the first 10 documents. The solution explanation in the next section covers the flow of the program.

Solution Explanation:

In order to run the program, the user indicates the directory where the documents are located. Once the directory is provided, the **parseFileDirectory** function is called with the directory path. The function goes over all the files in the directory and calls the **parseFile** function with the attributes file path and file name.
The parseFile function reads the file's content and then breaks it into individual words using space (single or sequence) as the delimiter. The function then joins 3 subsequent words to form a shingle. These shingles are kept in a HashMap (**shingleLabel**) with the shingle as the key and an integer as the value. Each unique shingle is assigned a number (i.e. it is labelled) as and when it is encountered while sequentially reading the documents. Keeping it as a key in the hash map ensures that only a new shingle is labelled and older shingles are not. At the same time a TreeMap (**documentShingles**) is constructed which is a mapping from the document name to the list of unique shingles that it has. The list of shingles is an integer list holding the shingle's label as assigned in the shingleLabel map.
Once all the shingles have been determined (after iterating over all the files in the directory), we know how many shingles are there (**shingleCount**) and then the function **getClosestPrimeNumber** is invoked to determine the next largest prime number (**p**) after the number of shingles. Once that is determined, the function **generateABValues** is called to generate 25 (defined by **randomValueCount**) random a and b integer values where a lies in the range [1, p – 1] and b lies in the range [0, p – 1].
The **generateDocumentSketch** function is invoked next and it is here that for each document the labels of all the shingles in it are iterated over and the value $\mathbf{f_s(x)=(a_s x + b_s) \bmod p}$ is computed where s denotes the index for a and b values and x is the shingle label. For each s, the shingle with the smallest f_s value is determined and that is then added to the **documentSketch** which is a mapping (TreeMap) from the document name to the list of these shingles. Thus a documentSketch is 25 shingles representing the document.
Once we have the document sketches for all documents, the **generateDocumentPairs** function is called which iterates and compares each document with every other document by computing the Jaccard coefficient using the function **estimatedJaccardCoefficient**. The function (estimatedJaccardCoefficient) picks up the lists of shingle labels from the documentSketch of the two documents being compared and calculates the number of shingles which are common in both the documents. The count of these shingles is then divided by the number of shingles in these two lists i.e. 25 and returned. If this value is greater than 0.5 (defined by **thresholdVal**) then the document names are displayed along with the Jaccard coefficient value.
Finally, for the first 10 documents, the top 3 documents with the highest Jaccard coefficient are listed. The function **generateTopDocuments** is called for this purpose where the first parameter is the number of documents being read starting with the first document and the second parameter is the number of documents to be listed for that document in descending order of Jaccard coefficient value. The function estimatedJaccardCoefficient is used to compute the value and then the top 3 values are determined and the corresponding document along with the coefficient value is displayed on the screen. Please refer to the sample output at the end of this report showing the results.

Assumptions:

- The program runs once, enlists all the values for that run and exits. On a re-run, it begins with a new state and so results from the previous run may differ from the results of this run depending on the random a and b values.
- documentShingles only has the unique shingles for the document.
- A pair appears only once. If doc1 and doc2 appears as a pair then doc2 and doc1 would not appear.

Execution Instructions:

Unzip **a4.zip** to get Report.pdf (report file), Binary and Source folders. The Source folder consists of the src (consisting of the nearduplicatedetection package which contains DuplicateDetection.java) and bin (consisting of the nearduplicatedetection package which contains DuplicateDetection.class) folders.

Inside the Binary folder there is a file called DuplicateDetection.jar. Type the following command to run the system:

java -classpath **<path to DuplicateDetection.jar>** nearduplicatedetection.DuplicateDetection **<directory path on system where the files are kept>**

Note: Please use jdk 6. It is available on machines in the CSUGLAB

Sample Output:

The output is generated in the following order:

<File name 1> <File name 2>
Estimated Jaccard coefficient = <Jaccard coefficient value>
..
..
..

Displaying top 3 Jaccard Coefficient documents for first 10 documents

1.<Name of 1st document>
Top documents are:
1. <Name of document having highest Jaccard coefficient value with this document>
Jaccard Coefficient = <Jaccard coefficient value>
2. <Name of document having second highest Jaccard coefficient value with this document>
Jaccard Coefficient = <Jaccard coefficient value>
3. <Name of document having third highest Jaccard coefficient value with this document>
Jaccard Coefficient = <Jaccard coefficient value>
..
..
..
10.<Name of 10th document>
Top documents are:
1. <Name of document having highest Jaccard coefficient value with this document>
Jaccard Coefficient = <Jaccard coefficient value>
2. <Name of document having second highest Jaccard coefficient value with this document>
Jaccard Coefficient = <Jaccard coefficient value>
3. <Name of document having third highest Jaccard coefficient value with this document>
Jaccard Coefficient = <Jaccard coefficient value>

In order to run the program type out the command as described in the previous section. For eg. on my machine I type out the following command and get the result as follows (command is in bold):

java -classpath ~/Documents/4300\ -\ Information\ Retrieval/Assignment\ 4/a4/DuplicateDetection.jar nearduplicatedetection.DuplicateDetection ~/Documents/4300\ -\ Information\ Retrieval/Assignment\ 4/test/

```
file01.txt file75.txt
Estimated Jaccard coefficient = 0.96

file02.txt file76.txt
Estimated Jaccard coefficient = 0.88

file03.txt file87.txt
Estimated Jaccard coefficient = 0.96

file09.txt file64.txt
Estimated Jaccard coefficient = 0.84

file14.txt file88.txt
Estimated Jaccard coefficient = 0.84

file18.txt file99.txt
Estimated Jaccard coefficient = 0.96

file23.txt file51.txt
Estimated Jaccard coefficient = 0.96

file25.txt file40.txt
Estimated Jaccard coefficient = 0.96

file32.txt file52.txt
Estimated Jaccard coefficient = 0.96

file34.txt file63.txt
Estimated Jaccard coefficient = 0.96

Displaying top 3 Jaccard Coefficient documents for first 10 documents

1.file00.txt
Top documents are:
1.file66.txt
Jaccard Coefficient = 0.24
2.file70.txt
Jaccard Coefficient = 0.16
3.file72.txt
Jaccard Coefficient = 0.16

2.file01.txt
Top documents are:
1.file75.txt
Jaccard Coefficient = 0.96
2.file68.txt
Jaccard Coefficient = 0.2
3.file67.txt
Jaccard Coefficient = 0.16

3.file02.txt
Top documents are:
1.file76.txt
Jaccard Coefficient = 0.88
2.file81.txt
Jaccard Coefficient = 0.16
3.file08.txt
```

```
Jaccard Coefficient = 0.04

4.file03.txt
Top documents are:
1.file87.txt
Jaccard Coefficient = 0.96
2.file77.txt
Jaccard Coefficient = 0.16
3.file78.txt
Jaccard Coefficient = 0.12

5.file04.txt
Top documents are:
1.file71.txt
Jaccard Coefficient = 0.28
2.file68.txt
Jaccard Coefficient = 0.24
3.file70.txt
Jaccard Coefficient = 0.16

6.file05.txt
Top documents are:
1.file80.txt
Jaccard Coefficient = 0.24
2.file77.txt
Jaccard Coefficient = 0.2
3.file86.txt
Jaccard Coefficient = 0.2

7.file06.txt
Top documents are:
1.file85.txt
Jaccard Coefficient = 0.2
2.file77.txt
Jaccard Coefficient = 0.16
3.file79.txt
Jaccard Coefficient = 0.16

8.file07.txt
Top documents are:
1.file82.txt
Jaccard Coefficient = 0.16
2.file77.txt
Jaccard Coefficient = 0.08
3.file80.txt
Jaccard Coefficient = 0.08

9.file08.txt
Top documents are:
1.file80.txt
Jaccard Coefficient = 0.16
2.file81.txt
Jaccard Coefficient = 0.16
3.file84.txt
Jaccard Coefficient = 0.12

10.file09.txt
Top documents are:
1.file64.txt
Jaccard Coefficient = 0.84
2.file73.txt
Jaccard Coefficient = 0.2
3.file70.txt
Jaccard Coefficient = 0.16
```