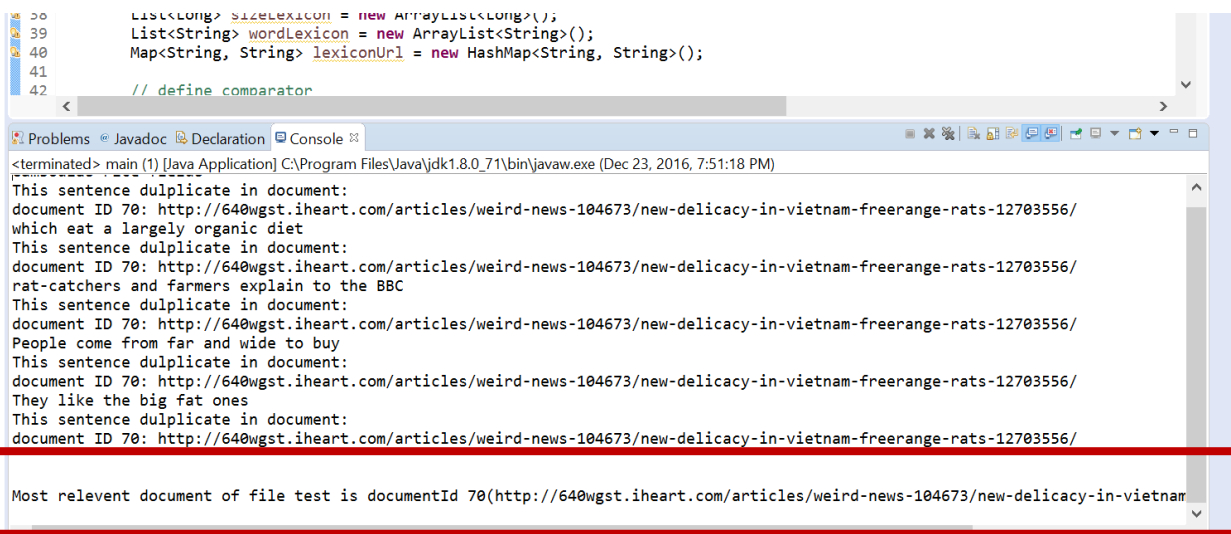


Project: Identifying and visualizing copied and plagiarized text

Result (How it works):

Given a file, call function `Query.getDulplicate(query, binaryFile, wordLexicon, offsetLexicon, sizeLexicon, lexiconUrl)`. The system will print out the relevant page of each sentence in that file, and finally print out the most relevant page of that file.



```
39 List<Long> sizeLexicon = new ArrayList<Long>();
40 List<String> wordLexicon = new ArrayList<String>();
41 Map<String, String> lexiconUrl = new HashMap<String, String>();
42 // define comparator

<terminated> main (1) [Java Application] C:\Program Files\Java\jdk1.8.0_71\bin\javaw.exe (Dec 23, 2016, 7:51:18 PM)
This sentence dulplicate in document:
document ID 70: http://640wgst.iheart.com/articles/weird-news-104673/new-delicacy-in-vietnam-freerange-rats-12703556/
which eat a largely organic diet
This sentence dulplicate in document:
document ID 70: http://640wgst.iheart.com/articles/weird-news-104673/new-delicacy-in-vietnam-freerange-rats-12703556/
rat-catchers and farmers explain to the BBC
This sentence dulplicate in document:
document ID 70: http://640wgst.iheart.com/articles/weird-news-104673/new-delicacy-in-vietnam-freerange-rats-12703556/
People come from far and wide to buy
This sentence dulplicate in document:
document ID 70: http://640wgst.iheart.com/articles/weird-news-104673/new-delicacy-in-vietnam-freerange-rats-12703556/
They like the big fat ones
This sentence dulplicate in document:
document ID 70: http://640wgst.iheart.com/articles/weird-news-104673/new-delicacy-in-vietnam-freerange-rats-12703556/

Most relevent document of file test is documentId 70(http://640wgst.iheart.com/articles/weird-news-104673/new-delicacy-in-vietnam-freerange-rats-12703556/)
```

Algorithm (How I did it):

This project is made of two main parts: parse and compress & query. After download all the files, parse each file, assign them document Id, get their URL, get postings of each document. Merge all the postings into a big file, change it into an inverted index.

Compress this index using variable bytes compress. Compress one posting into many chunks. In the first 8 bits, there is the last document id of that chunk. Then,

there is the size of the whole chunk in the next 8 bits. Using variable byte method, put 128 (or less) documents Id, their frequency and their positons into the chunk.

Last Id (8 bits)	Size (8 bits)	128 Document Id	128 frequency	Position of all 128 documents
------------------	---------------	--------------------	---------------	-------------------------------------

Query: When get the query file, split the whole content into many phrases. For one phrase, get each word and find its inverted list. Using DAAT method find document ID contains same word. (If using TAAT, the computer will be out of memory.) First, find all the document id of each word, check the smallest Id of all inverted lists. Find the next document Id of that list. If all the inverted lists have the same id, check the position of that id. If the position is continuous, save it to the result, and keep finding others. After finishing all the sentences, we can find document id of plagiarized text. The system will print out the url of coped page of each sentence. Finally, the system will print out the most relevant page of the paper.

Main classes

1. Part one (Parse): Parse, Posting, Merge, Inverted Index, BinaryFile
 - a. Parse: parse the file and separate pages.
Function: `public int[] Parse.separate_page (String fileName, String destinationPath, String urlLexicon)`
 - i. function: parse a wet file, separate each page in it and get its url, save url and document id into a file.
 - ii. input:
 - filename: name of wet file to be parsed.
 - destinationPath: a dir to save posting of each page
 - urlLexicon: name of url lexicon file

- iii. return: int array of size 2, first element is the average length of all the pages, second element is number of all the documents

b. Posting: get posting of each page file

Function: public static void getPosting(String text, int did, String sortPath)

- i. function: read in a page file, get each word and its frequency of this page file
- ii. input:
 - text: content of each page file
 - did: document id of each page file
 - sortPath: directory that save the posting file

c. Merge: merge all the posting file into a big posting file.

Function: public void Merge(String sourcePath, String destinationFile, final Comparator<String> cmp,int flag)

- i. function: merge all the postings to one big posting
- ii. input:
 - sourcePath: name of folder that contains all the page postings
 - destinationFile: name of one big posting
 - cmp: comparator

d. Inverted Index: create inverted index from the posting file

Function: public void CreateInvertedIndex(String path, String destination, String lexiconPath, Comparator<String> cmp)

- i. function: create inverted index from a posting file
- ii. input:
 - path: name of the posting file
 - destination: name of the inverted index
 - lexiconPath: file to store word lexicon (not need)
 - cmp: comparator
- e. BinaryFile: : class used to store each posting files, create it for merging each page's posting.

2. Part two (Compress & Query): ByteUtils, Compress, VariableByteCode, Query

- a. ByteUtils: convert type Long to Byte array and vice versa.

Function: public static byte[] longToBytes(long l)

- i. function: convert long to a byte array with fix size of 8.
- ii. input: a Long type number
- iii. return: a Byte array with fix size of 8

Function: public static long bytesToLong(byte[] b)

- i. function: convert a byte array with fix size of 8 to a type Long number
- ii. input: a Byte array with fix size of 8
- iii. return: a Long type number

- b. Compress: compress an inverted index into a binary file. First, set buffer size equals 10M, read 10M inverted index, separate line, and get each word and its document id and frequency. Get the document id and frequency part, compress each 128 id and frequency in one block until all the id and frequency are compressed. Then get the size of that blocks and compute offset according to the size, put word, size, and frequency in to lexicon.

Function: public static void compress(String fileName, String metaLexicon, String binaryFile, int numberInTrunk)

- i. function: compress inverted index, get binary file to store information and lexicon file to search
- ii. input:
 - filename: name of inverted index
 - metaLexicon: name of lexicon file
 - binaryFile: name of binary file
 - numberInTrunk: 128

Function: public static long compress_one_word(int numberInTrunk, long offsetInBinaryFile, String c, String binaryFile, String metaFile)

- i. function: : save inverted index of one word into binary file and create its trunk information, return size of the binary file that contains information of that word.
- ii. input:
 - filename: name of inverted index
 - metaLexicon: name of lexicon file
 - binaryFile: name of binary file
 - numberInTrunk: 128

Function: public static long trunk_Compress_and_Metadata(int[] number, int[] freq, String fileName)

- i. function: save content as binary file and return its metadata, return: size of this trunk. Do the difference of the document id and put them into a array list, call the function encode() to compress it and frequency array list into binary file.
- ii. input:
 - number: content needed to be saved as binary file
 - filename: binary file name;

- freq: frequency in id
- iii. return: chunk size

Function: `public static void decompress_chunk(String sourceFile, long offset, long size, List<Integer> result)`

- i. function: decompress each chunk. Read the binary file, jump to the offset position and read the data out
- ii. input:
 - sourceFile: name of binary file
 - offset: offset of the chunk
 - size: size of the chunk
 - result: the int number of the chunk data

Fuction: `public static long trunk_Compress_and_Metadata(int[] number, int[] freq ,String fileName)`

- i. function: save content as binary file and return its metadata, return: size of this trunk. Do the difference of the document id and put them into a array list, call the function `encode()` to compress it and frequency array list into binary file.
- ii. input:
 - number: content needed to be saved as binary file
 - filename: binary file name
 - freq: frequency in id

- c. VariableByteCode: using Variable byte code to convert int number to byte array

Function: `byte[] encode(List<Integer> list)`

- i. function: encode all the number in the array list called list. For every int number in that array list, call the function innerEncode.
- ii. input:
 - list: array list of int numbers
- iii. return:
 - byte array

Function: innerEncode(int num, List<Byte> resultList)

- i. function: compress one word. First, get the remainder of number divided by 128, change the remainder into byte array, and divide the quotient by 128 again. Put the byte array into an array list called resultList. Keep doing until all the number is compressed. Then change the first byte of the last 8 bytes from 0 to 1.
- ii. input:
 - num: int number needed to be converted
 - resultList: a byte array list to store byte result

- d. Query: search one phrase, get all the document id that contains this phrase, print out the most relevant document id and its url.

Function: public void read_in_memory(String metaLexicon, String urlLexicon, List<String> wordList, List<Long> offsetList, List<Long> sizeList, Map<String, String> lexiconUrl)

- i. function: read lexicon file into three array list, and read url lexicon file into hashmap
- ii. input:
 - n wordList, offsetList, sizeList: empty arraylist to store each word, its offset and size.

- LexiconUrl: hashmap to store each document id and its url;
- metaLexicon, urlLexicon: name of lexicon file;

Function: public static long[] chunkInfo(String binaryFile, long offset)

- i. function: get the information (last id, chunk size) of the chunk
- ii. input:
 - binaryFile: name of the compressed binary file
 - offset: the offset of the beginning of the chunk
- iii. return:
 - a long array, first element is last id of that chunk, second element is the size of chunk

Function: public static void getDuplicate(String query, String binaryFile, List<String> wordLexicon, List<Long> offsetLexicon, List<Long> sizeLexicon, Map<String, String> urlLexicon)

- i. function: When get the query file, split the whole content into many phrases. For one phrase, get each word and find its inverted list. Using DAAT method find document ID contains same word. (If using TAAT, the computer will be out of memory.) First, find all the document id of each word, check the smallest Id of all inverted lists. Find the next document Id of that list. If all the inverted lists have the same id, check the position of that id. If the position is continuous, save it to the result, and keep finding others. After finishing all the sentences, we can find document id of plagiarized text. The system will print out the url of copied page of each sentence. Finally, the system will print out the most relevant page of the paper.
- ii. input:
 - query: the phrase needed to be search
 - binaryFile: the name of the binaryFile

- wordLexicon, offsetLexicon, sizeLexicon: three lexicons that put into the memory before query
- urlLexicon: lexicon stored all the documents urls

Function: public static void

convertFullResultToldFreqPosition(List<Integer> result, List<Integer> id, List<Integer> freq, List<Integer> position)

- function: convert a full chunk data into document id, frequency and positions
- input:
 - result: the int list of original data
 - id, frequency, position: three lists that store each information

Function: public static void

convertNotFullResultToldFreqPosition(List<Integer> result, List<Integer> id, List<Integer> freq, List<Integer> position)

- function: convert a none full chunk data into document id, frequency and positions
- input:
 - result: the int list of original data
 - id, frequency, position: three lists that store each information

Function: public static int findPosition(List<Integer> id, List<Integer> freq, List<Integer> position, int keyId, List<Integer> keyPosition)

- function: find the position of a certain document id.
 - input:
 - id, freq, position: three lists that store each information
 - keyId: the key id
- keyPosition: a int list that store the position of that id