Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

GÖKTUĞ ALİ AKIN
161044018

Course Assistant:AYŞE ŞERBETÇİ TURAN

1 INTRODUCTION

1.1 Problem Definition

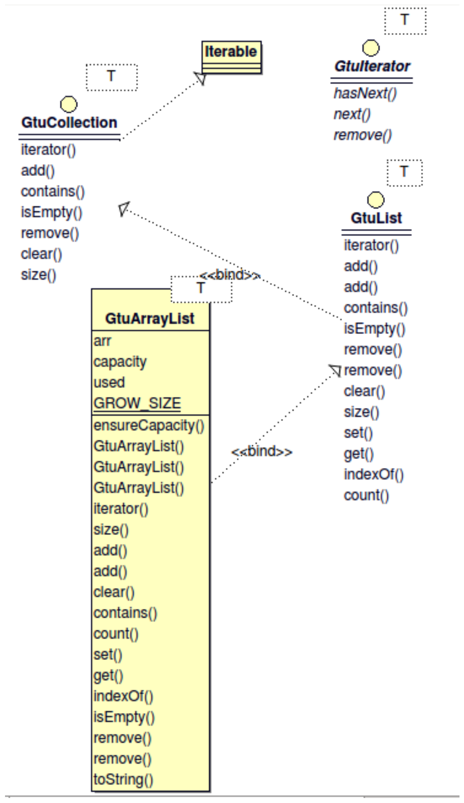
In this homework, the problem is design and implement a program that reads the input files which contains words and process these words with different topics which are about Natural Language Processing. For example, user of this program gets the bigrams and TFIDF values in these files.

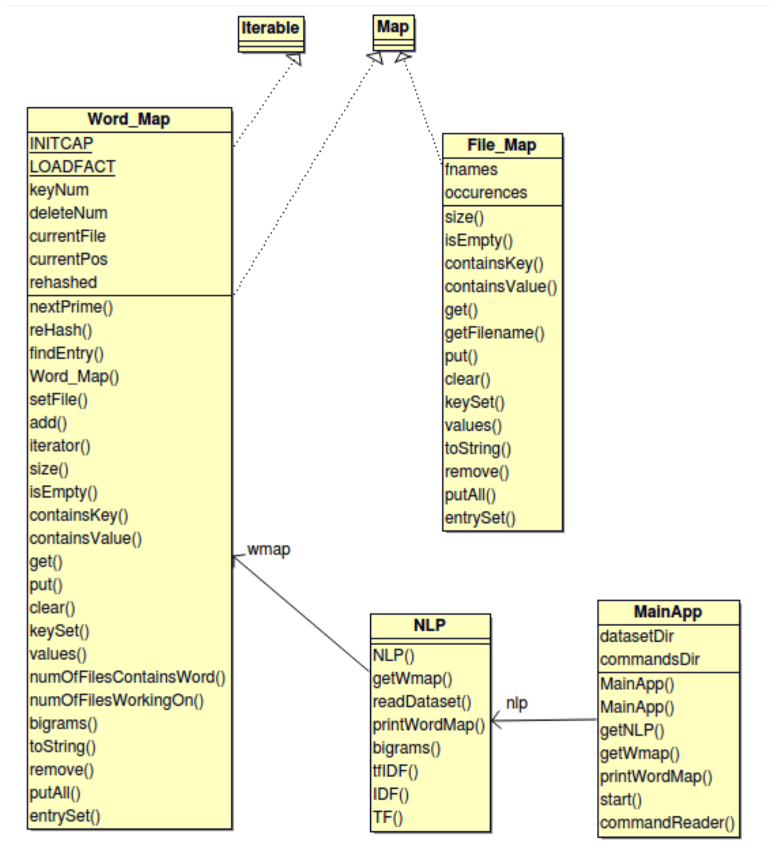
1.2 System Requirements

A user which has already installed Java her/his computer can run this program properly. In addition, this program process the words which are contained int txt files where the number of files is X. While X(number of files) is increasing, the run time of the program will decrease according to number of files which contained words will be processed. It is a bit exhausting for computer. There is no specific plug-in must be installed to run this program.

2 METHOD

2.1 Class Diagrams





2.2 Use Case Diagrams

There is no graphical user interface designed for this program. User can run this program giving dataset directory which contains all of the txt files that will be processed and a input txt file that contains commands (queries) such has bigram and tfidf. For input.txt file, there are two main commands. First is the **bigram** command. User can run this command by add a line to the input.txt file : ‘*bigram <word>’.* (word : word which will be searched for bigrams). Second command is the **tfidf** command. User can run this command by add a line to the input.txt file : ‘*tfidf <word> <filename>’.* (word : word which will be searched for TFIDF, filename which will be used in the TFIDF calculation.). User of this program, adds lines of commands into input.txt file and run this program.

Example input file :

bigram very

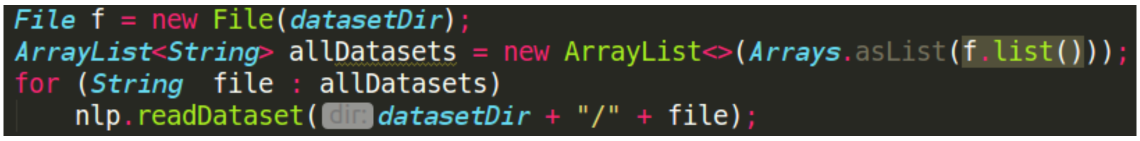
tfidf Coffe file1

bigram world

tfidf Brazil file2

2.3 Problem Solution Approach

Reading all file names with specified directory and read datasets



```
File f = new File(datasetDir);
ArrayList<String> allDatasets = new ArrayList<>(Arrays.asList(f.list()));
for (String  file : allDatasets)
    nlp.readDataset( dir: datasetDir + "/" + file);
```

To read all dataset file names, File class used for opening the file which contains dataset(txt) file which contains words. After opening the file, f.list() method used for reading all file names, and the are added into ArrayList. Finally, we have all file names in the arraylist.

To read all datasets, readDataset() method executed with every file name which are contained in the allDatasets ArrayList. By this approach, every dataset in the director readed.

Add all words into the word map with file information

With *readDataset()* method, file readed with given filename parameter. For every word readed during this method executing, this word added into word map by *Word_Map* add() method. This method accepts filename and position of the word. This position information will be added into File_Map.

Addition to the word map details

If a word is not contained in the word map, word will be succesfully added into the word map. A new File Map created for this word, occurence(position) information added to this File Map's value data. This is usual map adding.

If a word is contained in the word map, word will not added into the word map. *Put()* method of the word map, finds the file map of this word, adds the occurence(position) information into the file map's value data. So every word added into the word map, file maps are correctly updated.

Finding bigrams

To find the bigrams, *bigrams()* method of class Word_Map used. This method takes a parameter which described the word which will be used in the bigram finding process. For finding bigrams, I design a for loop which iterates over the word map. By this, all words are tried with given parameter word wheter it is a bigram or not. In this for loop, another for loop iterates over the file map of mainword(given by parameter) to try all files. Another nested for loop is designed to iterate over the occurence lists. So by this approach, every possibilty is tried brute-forcely. In addition, while iterating over the word map, iterator used(next entry linked structure). This is more effective than visiting the null cells in the table.

Calculating TFIDF

TFIDF = TF * IDF. There are two methods implemented to seperate the calculaton of TFIDF. TF value calculated by *TF()* method of World_Map class. This method iterates over the world map and find the correct file map. After finding the correct file map, I can access its occurence list with getter method. After accessing the occurence list, I can manipulate this data to calculate the Number of words appears in document and total number of terms in the document by iterating over the world map. IDF value calculated by two methods of World_Map class. Theese are *numOfFilesWorkingOn()* and *numOfFilesContainsWord()*. This methods, acces the file maps and manipulate the results to find the different number of files and different number of files which contains specific word. By knowing this values, IDF is calculated by *Math.log(numOfFilesWorkingOn() / numOfFilesContainsWord()).*

Reading commands from the input file and encapsulation of the main program

To encapsulate a main app into simple prgoram, **MainApp** class is designed and implemented. This class has a constructor that thakes dataset folder directory and the input file name. In main, we can just create a mainapp with giving dataset folder and input file name and start the program with *start()* method. So, what MainApp class do to encapsulate and simplfy the program ? This class, has an inner class named **NLPJob** which describes the job (queries) which will be readed from the input txt file. This NLPJob class has data fields which are *commandName(String)*, *folderName(String)* and a *wordName(String)*. In this NLPJob class constructor, parameter Commands(String) parsed word by word, divide the whole command into two or three words. *Run()* method of this class, runs the correct command by looking up to its data fields which are described.

Another approach of the MainApp class is, when this class is created, it reads the command.txt file due to given commandsDir(String) parameter and instantiate a NLPJob object and add this object into the job ArrayList which contains jobs (queries) as a data field otf MainApp class. So, we have an ArrayList which contains waiting jobs(NLPJob). *Start()* method runs this all jobs by invoking their *run()* method declared in NLPJob class.

Time complexities of File Map class methods

containsKey() : O(N).

containsValue() : O(N).

isEmpty() : O(1)

size() : O(1)

getFilename() : O(N).

put() : O(N)

clear() : O(1).

keySet() : O(1).

values() : O(1).

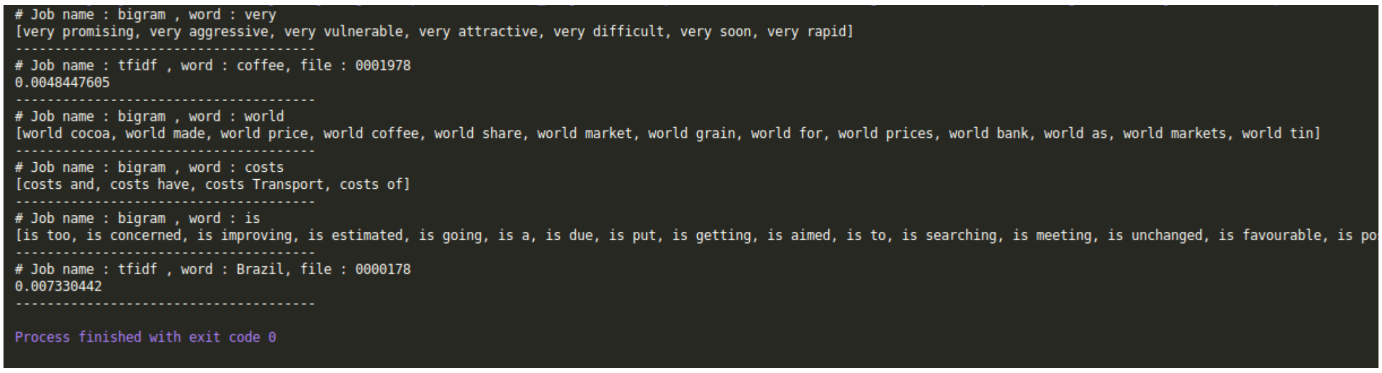
Get() : O(N).

3 RESULT

3.1 Test Cases

I tried this program with dataset folder which contains lots of txt files.

3.2 Running Results



```
# Job name : bigram , word : very
[very promising, very aggressive, very vulnerable, very attractive, very difficult, very soon, very rapid]
--------------------------------------
# Job name : tfidf , word : coffee, file : 0001978
0.0048447605
--------------------------------------
# Job name : bigram , word : world
[world cocoa, world made, world price, world coffee, world share, world market, world grain, world for, world prices, world bank, world as, world markets, world tin]
--------------------------------------
# Job name : bigram , word : costs
[costs and, costs have, costs Transport, costs of]
--------------------------------------
# Job name : bigram , word : is
[is too, is concerned, is improving, is estimated, is going, is a, is due, is put, is getting, is aimed, is to, is searching, is meeting, is unchanged, is favourable, is po
--------------------------------------
# Job name : tfidf , word : Brazil, file : 0000178
0.007330442
--------------------------------------

Process finished with exit code 0
```