Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

Yusuf Can Kan
161044007

Course Assistant: Ayşe Şerbetçi Turan

1 INTRODUCTION

1.1 Problem Definition

In this homework we want to achive calculating 2 basic NLP operations with using the HashMap logic. We want to retrieve the bigrams and we want to calculate TFIDF value. The bigram is the word set that has same words inside of all word sets. For example we want to calculate some word bigrams such as ‘coffee’. In that operation we can obtain word sets such that ;
{‘coffe table’ , ‘coffee plane’, ‘coffee is’, …}.
As we can see all the word sets has same word which is coffee.
The TIFDF value is used for obtaining a words importance in a single document. With that value we can district files with respect to words occurances. We can calculate TIFDF value such as like that;
TFIDF = TF*IDF.

TF is means Term Frequency. We can calculate TF like that;
TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF means Inverse Document Frequency. We can calculate IDF like that;
IDF(t) = log(Total number of documents / Number of documents with term t in it)

In this homework we have to 2 simple hashmap with linked each other. First hashmap has all the words inside of a some document sets. All the words is the key of the hashmap. For the placing key values we use linear probing. Besides that for the make simple operations build a linked list structure between the words. All the word make point to the other words so with that we don’t look the null areas and we can reduce the operations count.

First hashmaps has another hasmap structure for reaching the file name (for which file has the word) and index number(for where is the word inside the file). The second hashmap key is the file name and value is the list of index number.

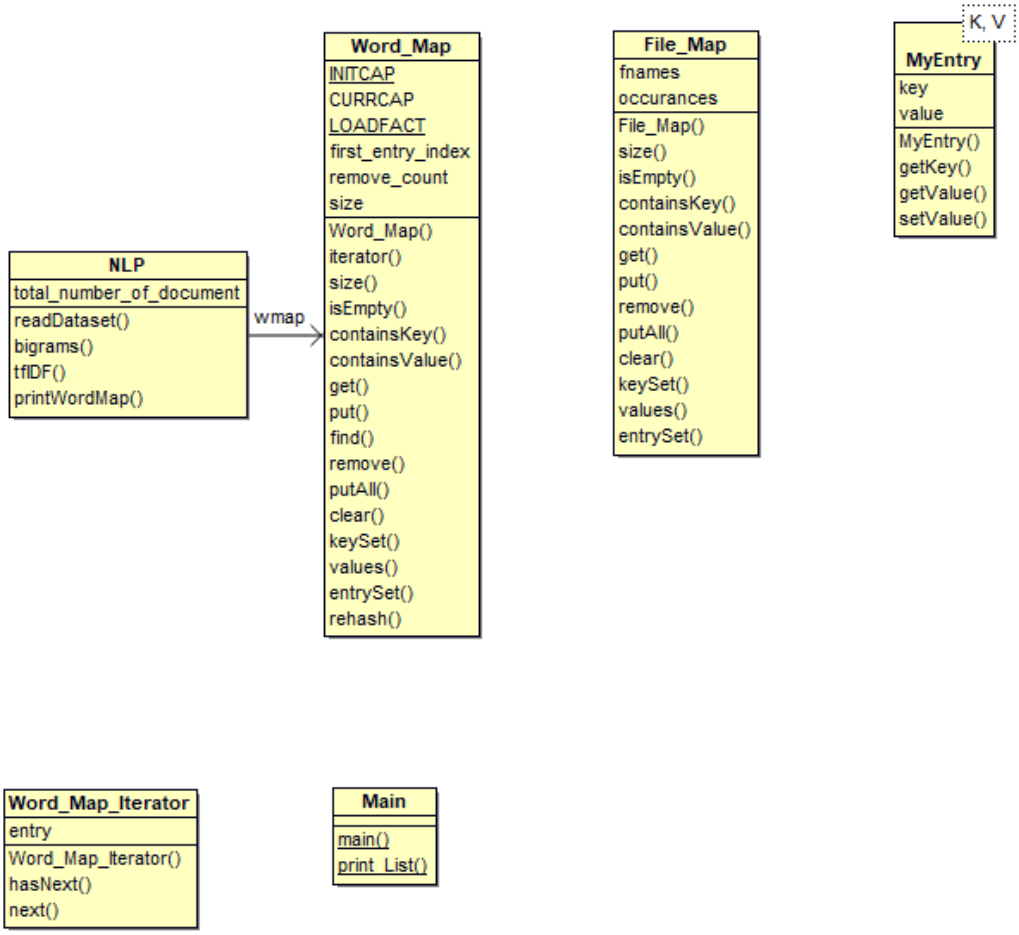
So for all that we can easily reach the word informations such that which files contains the words, which index inside the file,…

1.2 System Requirements

My solution does not require spesific piece of hardware. It requires just a little memory and CPU but not too much.You can use my solution with every computer unless the computer mush have JVM. It works every machine which has JVM. It does not requires spesific operating system. You can use it on Linux, Windows or Mac operating system. My solution is so simple, you can use it almost everywhere.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams

This software can be uses with set of files which has words inside of it. The user can reach and store some words bigrams inside of all the documents easily and user can reach the importance of one word for the single document. The only thing which expected of the user is giving the folder which has a lot of word files inside of it and giving the readable file which contains operations and right inputs inside of it. Simple input file example is below;
bigram very
tfidf coffee 0001978
bigram world
bigram costs
bigram is
tfidf Brazil 0000178

The user supposed to use this software in this simple order. First user must give the path of the data set file, after that user must create the file which contains operations and right inputs, and after all the inputs user must starts the software and finally user can reach all the outputs.

Use case diagram is not required for this homework.

2.3 Problem Solution Approach

First i implemented the wordmap class and its structures. For that i used the linear probing algorithm and for that i implemented the proper put and get methods and rehash method. For every adding operation i created a find() method for calculating the given keys indexex. This method calculates the index and if that index is taken with another index it looks the other indexes and finds the most proper index. If the map already has that key, find() method returns that keys indexes. For the rehashing operations i holds the load factor value. With respect the load factor i resized the array size with 2 times. After that for the easy iterations i implement the simple iterator class. For reaching the words with easy way i created a linked list structure with using the simple inner node class. Inside of that node class there are 3 fiels which are key,value and next node. I hold the next node indexes inside the wordmap class so with that i can reach the head of the linked list easily.

After implementing word map and its structure i implemented the file map structure, inside of that hasmap we use arraylist structure so there is no need to rehashing operations. Every put operations exteds the data structure. Inside of that hasmap i stored the file name with string and i stored the index numbers with integer list structure. In this way we can reach index numbers inside of spesific file easily.

After implementing two main hashmap class i implemented the NLP class and i implemented the simple NLP operations with using the wordmap and filemap hashmaps.

I choose that desing because it is easy to use and easy to implement. This structure is the best structure for this aim. First you can reach the word which you want to make some operations with the linked list structure and after that you can look the file easily and you can reach the index. The below table is contains WordMap,FileMap and NLP classes methods time and space complexity with expressions.

WordMap Class		
Method Name:	**Time Complexity:**	**Space Complexity:**
iterator()	O(1)	O(1)
size()	O(1)	O(1)
isEmpty()	O(1)	O(1)
containsKey()	O(n) Which n is the number of element, word linked list has.	O(n) Which n is the number of element, word linked list has.
containsValue()	O(n) Which n is the number of element, word linked list has.	O(n) Which n is the number of element, word linked list has.
get()	O(n) Which n is the unproper index while looking inside the find method.	O(n) Which n is the unproper index looking inside the find method.
put()	O(n) Which n is the number of element, word linked list has.	O(n) Which n is the number of element, word linked list has.
find()	O(n) Which n is the unproper index which is looked while looking for an proper index.	O(n) Which n is the unproper index which is looked while looking for an proper index.
putAll()	O(n) Which n is the element number given map structure and element number of current hashmap.	O(n) Which n is the element number given map structure and element number of current hashmap.
clear()	O(n) Which n is the number of elements.	O(n) Which n is the number of elements.

keySet()	O(n) Which n is the number of elements inside the wordmap.	O(n) Which n is the number of elements inside the wordmap.
values()	O(n) Which n is the number of elements inside the wordmap.	O(n) Which n is the number of elements inside the wordmap.
rehash()	O(n) Which n is the number of elements inside the wordmap and number of elements which added the map while resizing(comes from put method).	O(n) Which n is the number of elements inside the wordmap and number of elements which added the map while resizing(comes from put method).

FileMap Class		
Method Name:	**Time Complexity:**	**Space Complexity:**
size()	O(1)	O(1)
isEmpty()	O(1)	O(1)
containsKey()	O(n) Which n is the number of element, string list has.	O(n) Which n is the number of element, string list has.
containsValue()	O(n) Which n is the number of element, value list has.	O(n) Which n is the number of element, value list has.
get()	O(n) Which is the number of element which key list has.	O(n) Which is the number of element which key list has.
put()	O(n) Which n is the number of element, key list has.	O(n) Which n is the number of element, key list has.

remove()	O(n) Which n is the number of element, key list has.	O(n) Which n is the number of element, key list has.
putAll()	O(n) Which n is the element number given map structure and element number of current hashmap.	O(n) Which n is the element number given map structure and element number of current hashmap.
clear()	O(n) Which n is the number of elements.(Comes from collection array list implementation.)	O(n) Which n is the number of elements.(Comes from collection array list implementation.)
keySet()	O(n) Which n is the number of elements inside the wordmap.	O(n) Which n is the number of elements inside the wordmap.
values()	O(1)	O(1)
entrySet()	O(n) Which n is the number of element key list has.	O(n) Which n is the number of element key list has. It comes from created entryset class.

NLP Class		
Method Name:	**Time Complexity:**	**Space Complexity:**
readDataSet()	O(a*b*c*d) = O(n) Which a is the file lists inside a input directory, b is line number for every file, c is word number for every line, d is the filemap class put method complexity which is O(n) which n is the number of element file has.	O(n) Takes same space with respect to loop size. So the complexity is O(n) with the same reason of time complexit.
bigrams()	O(k) + O(a*b*c*d) = O(n) Which k is the wordmap get method complexity, a is the file map entries number ,c is the file map key number, c is the file hashmap value list size.	O(n) It tooks the O(1) size between the loops. So we can obtain O(n) space complexity with the same reasof of time complexity.
tfIDF()	O(a*b) = O(n) Which a is the key number of word map class and b is the key number of file map class.	O(a*b) = O(n) Which a is the key number of word map class and b is the key number of file map class.
printWordMap()	O(n) Which n is the number of element word map has.	O(n) Which n is the number of element word map has.

3 RESULT

3.1 Test Cases

First I tested the some priority methods for general classes and after validifying all of that is working, I tested the general program execution.

For the testing some classes priority methods i gave the below inputs;

```
Word_Map wm = new Word_Map();

File_Map fm1 = new File_Map();
List l1 = new ArrayList<Integer>();
l1.add(1);
l1.add(2);
l1.add(3);
fm1.put("yusuf1",l1);

File_Map fm2 = new File_Map();
List l2 = new ArrayList<Integer>();
l2.add(1);
l2.add(2);
l2.add(3);
fm2.put("yusuf1",l2);
fm2.put("yusuf2",l2);
fm2.put("yusuf3",l1);

File_Map fm3 = new File_Map();
List l3 = new ArrayList<Integer>();
l3.add(1);
l3.add(2);
l3.add(3);
fm3.put("asadafaga",l1);
fm3.put("yusuf1",l3);
fm3.put("yusuf3",l2);
fm3.put("yusuf12",l1);

wm.put("e1",fm1);
wm.put("e2",fm2);
wm.put("e3",fm3);
wm.put("e4",fm3);
wm.put("e5",fm3);
wm.put("e6",fm3);
```

```
System.out.println("wordmap containsKey test;");
if(wm.containsKey("e1")){
    System.out.println("WordMap contains e1 key.");
}
if(!wm.containsKey("e300")){
    System.out.println("WordMap does not contain e300 key.");
}

System.out.println("\n\nwordmap containsValue test;");
if(wm.containsValue(fm1)){
    System.out.println("WordMap contains fm1 value.");
}
if(!wm.containsValue(null)){
    System.out.println("WordMap does not contain null value.");
}

System.out.println("\n\nwordmap get() method test;");
if(fm3.equals(wm.get("e6")))
    System.out.println("Get method works!");



System.out.println("\n\nfilemap containsKey test;");
if(fm1.containsKey("yusuf1")){
    System.out.println("FileMap contains yusuf1 key.");
}
if(!fm1.containsKey("yusuf2")){
    System.out.println("FileMap does not contain yusuf2 key.");
}

System.out.println("\n\nfilemap containsValue test;");
if(fm1.containsValue(l1)){
```

```
    System.out.println("FileMap contains l1 value.");
}
if(!fm1.containsValue(null)){
    System.out.println("FileMap does not contain null value.");
}

System.out.println("\n\nFileMap get() method test;");
if(l3.equals(fm3.get("yusuf1")))
    System.out.println("Get method works!");
```

For the testing all program i gave the test input below;

```
bigram very
tfidf coffee 0001978
bigram world
bigram costs
bigram is
tfidf Brazil 0000178
```

3.2 Running Results

Classes some methods test outputs;

```
wordmap containsKey test;
WordMap contains e1 key.
WordMap does not contain e300 key.


wordmap containsValue test;
WordMap contains fm1 value.
WordMap does not contain null value.


wordmap get() method test;
Get method works!


filemap containsKey test;
FileMap contains yusuf1 key.
FileMap does not contain yusuf2 key.


filemap containsValue test;
FileMap contains l1 value.
FileMap does not contain null value.


FileMap get() method test;
Get method works!
```

General code test outputs;

```
[very vulnerable ,very attractive ,very rapid ,very promising ,very aggressive ,very difficult ,very soon]

0.004878173

[world markets ,world market ,world price ,world coffee ,world cocoa ,world share ,world made ,world prices ,world as ,
world tin ,world grain ,world for ,world bank]

[costs of ,costs have ,costs Transport ,costs and]

[is a ,is imperative ,is heading ,is the ,is expected ,is slightly ,is at ,is forecast ,is estimated ,
is projected ,is possible ,is not ,is wrong ,is insufficient ,is put ,is currently ,is unrealistic ,is scheduled ,is flowering ,
is foreseeable ,is precisely ,is great ,is beginning ,is ending ,is set ,is getting ,is due ,is still ,is likely ,
is no ,is unchanged ,is unlikely ,is showing ,is helping ,is being ,is often ,is why ,is it ,is searching ,
is also ,is high ,is now ,is down ,is meeting ,is to ,is proposing ,is willing ,is some ,is fairly ,
is downward ,is favourable ,is sceptical ,is how ,is depending ,is caused ,is Muda ,is improving ,is trimming ,is planned ,
is harvested ,is trying ,is sending ,is in ,is faced ,is basically ,is too ,is uncertain ,is keeping ,is time ,
is sold ,is defining ,is committed ,is affecting ,is going ,is 112 ,is difficult ,is well ,is that ,is he ,
is one ,is insisting ,is are ,is unfair ,is apparent ,is keen ,is more ,is an ,is open ,is concerned ,
is very ,is passed ,is after ,is aimed ,is only]

0.007383948
```