Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

GÖKÇE NUR ERER
171044079

Course Assistant: Ayşe Şerbetçi Turan

1 INTRODUCTION

1.1 Problem Definition

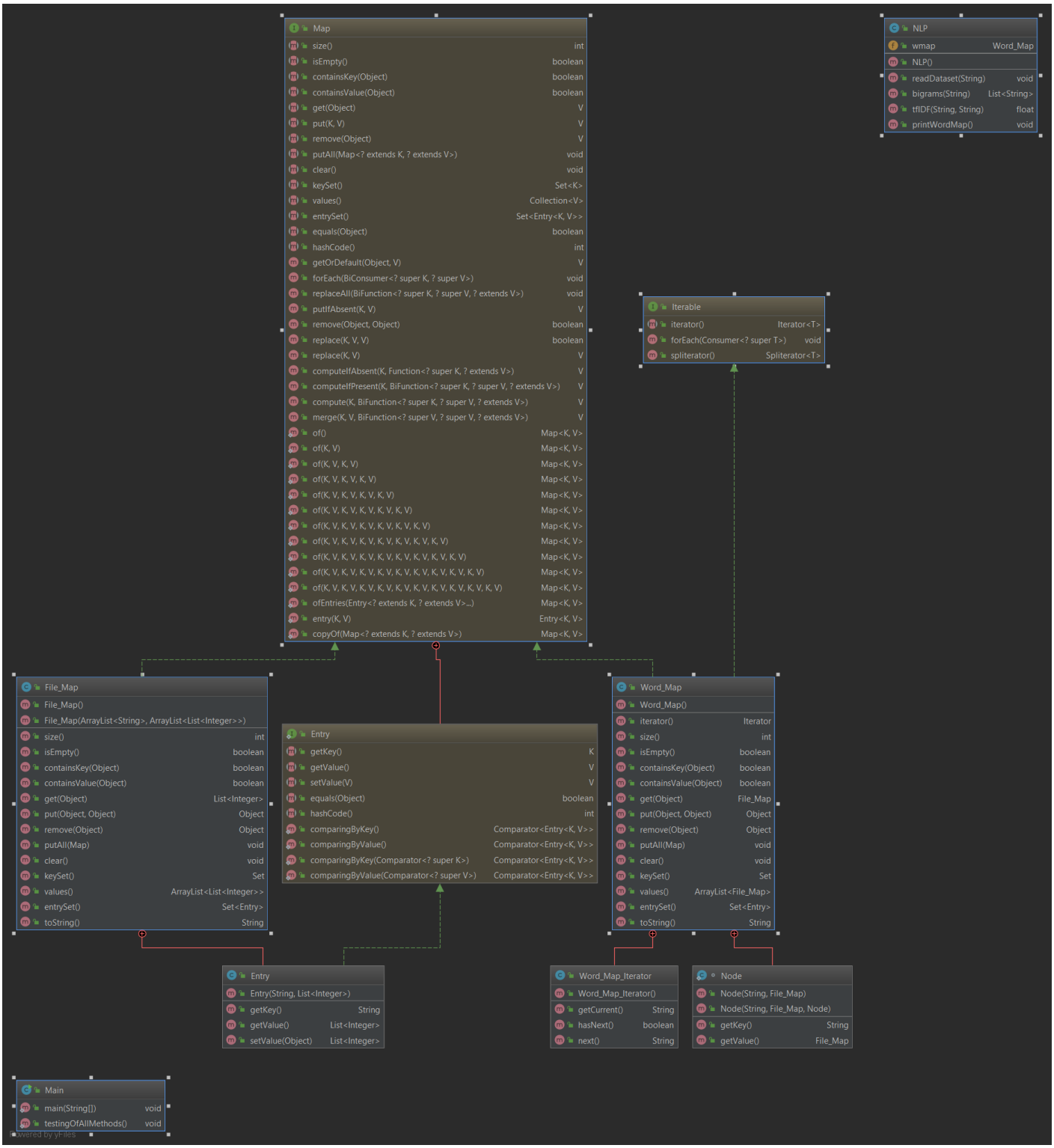
The objective of this project is to implement two hash maps to find the bigrams of a given word in the files and TFIDIF of some certain words. One of those hash maps (called word map) will implement a map that gets the words as its keys and the values of the map will correspond to the second map structure. The second map (called file map) will contain file names as keys and the word occurrence list in those files.

1.2 System Requirements

This project can run on an average computer which has Java components like (JVM, SDK, JDK...) and has an IntelliJ Idea IDE since it is and IntelliJ project.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams

Users are expected to choose the test results they request in this software. To see the results of the methods of the classes Word_Map and File_Map, user is expected to choose 0. To see the results of the NLP class methods user is expected to choose 1. The user is also expected to choose if they want to print the word map of the NLP instance. User is expected to choose 0 if they want to print the word map, to choose 1 if they don't want to print the word map.

2.3 Problem Solution Approach

Word_Map Class: This class implements a hashmap called Word_Map which hold a Node array to keep its entries. Node class is an inner class of Word_Map which contains keys and values of the word map and will be explained later. Word_Map class has the following methods and constructors:

Word_Map ():
creates a word map with a Node array with the capacity of 10.

boolean containsKey (Object Key):
checks if a certain key is in the word map. If the word map is empty, returns false. Complexity for this method is: O(n), n being the entry count. Because this method only traverses through the existing entries. Which makes the loop execute n times. Thus, complexity is O(n).

boolean containsValue(Object value):
checks if a certain value is in the word map. If the word map is empty, returns false. Complexity for this method is: O(n), n being the entry count. Because this method only traverses through the existing entries. Which makes the loop execute n times. Thus, complexity is O(n).

File_Map get (Object key):
gets the value of the given key.
Complexity for this method is: O (n), n being the word map's table length since to find the key, it is required to traverse in the array and check if the table entry's key matches

the given key. To do so a loop must traverse averagely n times. Thus, complexity is equal to O(n).

Object put (Object key, Object value):

Creates and puts the given entry denoted by given key and value in to the entry table of the word map. This method also uses rehash method (which will be explained further) if the size exceeds the multiplication of the current capacity and load factor. This method uses Java's Object class' own HashCode method to get the index of the newly added entry. To calculate the entry, the modulo of the current capacity is taken from the hashcode method. If the word map is empty, the entry is just added. If the word map is not empty and the key does not exist in the word map, method checks if the given index to the entry is empty in the word map, if it is empty it replaces the entry in the empty spot, if it is not empty then it finds the new index by linear probing (incrementing the index one by one until a free spot has been found). If the key already exist in the word map it simply just changes its file_map value. The complexity of this method is O(n),n being the table size because method simply traverses the table in the word map, if the word map is not empty. The worst case would be rehashing and then adding the element but it would be still a multiplicand of n which results in O(n).

void putAll (Map m):

for every entry of the map m, this method puts them in to the word map.

Complexity of this method is O(n), n being the map's entry count. Since all the entries have to be traversed to be added to the word map.

void clear ():

clears the whole array.

Complexity of this method is O(n), n being the word map's table size. Since it is required to traverse n times to assign all elements null.

Set keyset():

Returns a set of keys added in it.

Complexity of this method is O(n), n being the not null entry count. Since all keys of all entries have to be added they all have to be traverse.

ArrayList<File_Map> values ():

Returns an arraylist of values added in it.

Complexity of this method is O(n), n being the not null entry count. Since all values of all entries have to be added they all have to be traverse.

int size ():

returns the size of the word map.

Complexity for this method is O (1) since it just returns the size value kept in the word map.

boolean isEmpty ():

returns true if the word map is empty, else it returns false.

Complexity for this method is O (1), since it just compares the size variable with 0 which has an O (1) complexity.

void rehash():

resizes the word map's table and puts all the entries in to the new resized table.

Complexity for this method is $O(n^2)$, n being the old table's size, since the loop in the method traverses n times, the new resized table will have the size of 2*n +1 so the put method will work in O(2*n+1) which is O(n). Since for every element put method is invoked the complexity is $O(n^2)$.

Word_Map_Iterator iterator():

Returns an iterator of type Word_Map_Iterator which is mentioned below.

Inner Class: Node Class

Node class is an inner class of Word_Map which has a String key and a File_Map value to represent the entries in the Word_Map. This class has the following constructors:

Node(String key,File_Map value,Node nextEntry):

Creates an entry with the given key, value and the next entry.

Inner Class: Word_Map_Iterator

This class implements the Iterator interface and it represents the iterator of the

Word_Map class. It iterates through the Node entries of the Word_Map's table.
It has the following methods and constructors:

Word_Map_Iterator(): returns the first added entry in the word map. (NOTE: first added element changes when rehashing happens).

String getCurrent() :

Returns the current node's key.

boolean hasNext():

Checks if the iterator has a next element to go.

String next():

Returns the current node's key and advances the current node to the next node.

File_Map Class: This class implements a map called File_Map which holds two arraylists one of them holding the file names and the other one holding the occurrence lists of the word which is file map's corresponding word map's key. This class also has an inner class called Entry and it will be explained below.

File_Map() :

creates a file map with an empty arraylist of file names and an empty arraylist of occurrence lists.

File_Map(ArrayList<String> file_names,ArrayList<List<Integer>> occurences) :

creates a file map and initializes its arraylists one of the with the file_names arraylist and the other one with the occurrences arraylist.

int size ():

returns the size of the file map.

Complexity for this method is O (1) since it just returns the size of the file name arraylist.

boolean isEmpty ():

returns true if the file map is empty, else it returns false.

Complexity for this method is O (1), since it just compares the size of the file name arraylist with 0 which has an O (1) complexity.

boolean containsKey(Object key):

returns true if the key is in the file map. Complexity of this method is O(n) since the indexOf operation of the arraylist class has the complexity of O(n).

boolean containsValue(Object value):

returns true if the given value is in the file map. The value is considered as an integer in this case. Complexity of this method is O(n*m), n being the size of the occurrences list, m being each individual list in the occurrences list's size. Since this loop will execute for all the n lists which have m sizes.

List<Integer> get(Object key):

Returns the occurrence list of the corresponding key. Complexity of this method is O(n) since indexOf and get methods of Java's ArrayList class is O(n) and they are added in complexities since they are one after another. So complexity would be O(2*n) which is O(n).

Object put(Object key, Object value):

Adds the given key to file names arraylist and adds the given value to the occurrence arraylist. If the key does not exist it creates a new arraylist for occurrence, adds the value in it and adds to the file map's occurrences arraylist, key is also newly added to the file names arraylist. If the key exists the value is simply added to the corresponding occurrence arraylist of the key.

The complexity of this method is O(n) since adding to a normal arraylist has the complexity of O(n).

Object remove(Object key):

Removes the entry corresponding to the key. Removes both key and value from their file names and occurrence arraylist.

Complexity of this function is O(n) since Java's ArrayList class' remove has O(n) complexity.

void putAll(Map m):

puts all the entries in the map m, into the file map.

Complexity of this method is O(n*m), n being the number of keys, m being each individual list in the occurrences list's size. Since this loop will execute for all the n lists which have m sizes.

Void clear():

Clears the arraylists. Complexity of this method is O(n) since clear method of the Java's ArrayList class has the complexity of O(n) and for both arraylists it is called. So the complexity is O(n+n) = O(2n) which is O(n).

Set keySet():

Returns all keys as a set.

ArrayList<List<Integer>> values():

Returns the occurrences arraylist. Complexity of this method is O(1) since it just returns the arraylist itself.

Set<Entry> entrySet():

Returns a set of entries in the type of Entry which will be explained below. Complexity of this method is O(n), n being the file names arraylist size since the entry count will be equal to the file name count. So the loop will execute n times which results in O(n) complexity.

Inner Class: Entry: This class implements the Map interface's Entry interface to represent entries in file map as in keys and values. This class has the following constructors and methods:

Entry(String k,List<Integer> v):

Initializes the key as the given key and initializes the given value as the value.

String getKey():

Returns the key. Complexity of this method is O(1), since only the key variable is returned.

List<Integer> getValue():

Returns the value. Complexity of this method is O(1), since only the value variable is returned.

List<Integer> setValue():

Returns the old value. Complexity of this method is O(1), since only the value variable is returned and the rest of the operations in the method are assignments.

NLP Class: NLP class is a class that has a word map in it that reads a dataset directory and fills the word map to find the bigrams and the tfidf of some words. . NLP class has the following methods and constructors:

NLP():

Initializes the word map and initializes file count to be zero.

void readDataset(String dir):

reads the dataset in the given directory and fills up the word map. Complexity of this method is O(m*n), m being the word count and n being the file map's size.

List<String> bigrams(String word):

Finds the bigrams of a given word in the dataset. Complexity of this method is O(n*m*q), n being the word map's size, m being the file map's size and q being the each individual list's size in the file map. Since the loops execute in a nested way and they each iterate in the variables mentioned above, the complexity is their multiplication.

float tfIDF(String word, String filename):

This method finds the tfIDF of a given word and filename according to this formula:
TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).
IDF(t) = log(Total number of documents / Number of documents with term t in it)
TFIDF = TF*IDF
Complexity of this method is O(n), n being the word count in the file.

3 RESULT

3.1 Test Cases

Tested this program with the provided dataset directory, with the commands taken from the input.txt file. Both are provided in the project files.

3.2 Running Results

Only the NLP method results are provided in here, since the main goal was that. If each result of each method is requested, there is an option to view all the methods' testing in my program. It can be viewed from there. Also the printing of the word map is too long to add to this report so it can be viewed with another option in my program. So, it can be viewed from there too.

```
To see the method results of File_Map and Word_Map please enter
 0
To see the method results of NLP please enter 1
1
Calculating bigrams...
[very difficult, very promising, very rapid, very aggressive,
very attractive, very vulnerable, very soon]
Calculating TFIDF...
0.0048781727
Calculating bigrams...
[world markets, world market, world price, world grain, world
tin, world coffee, world bank, world as, world share, world
prices, world for, world cocoa, world made]
Calculating bigrams...
[costs have, costs and, costs of, costs Transport]
Calculating bigrams...
[is projected, is expected, is set, is ending, is getting, is
down, is only, is harvested, is trying, is to, is the, is
possible, is not, is a, is that, is no, is well, is still, is
forecast, is caused, is depending, is flowering, is heading, is
 imperative, is slightly, is estimated, is at, is likely, is
unchanged, is now, is insisting, is unfair, is are, is due, is
insufficient, is wrong, is unrealistic, is put, is currently,
is open, is scheduled, is concerned, is high, is committed, is
downward, is also, is sceptical, is how, is favourable, is
unlikely, is an, is trimming, is Muda, is improving, is one, is
 he, is searching, is going, is precisely, is great, is
beginning, is foreseeable, is affecting, is 112, is willing, is
 proposing, is fairly, is some, is planned, is very, is passed
, is difficult, is apparent, is after, is aimed, is time, is
keeping, is too, is defining, is sold, is uncertain, is meeting
, is sending, is more, is keen, is faced, is in, is basically,
is being, is showing, is helping, is it, is often, is why]
Calculating TFIDF...
0.0073839487
To see the result of the printWordMap() please enter 0, to
continue press 1:
1
Size of the word map in NLP:3836
```

- Main titles -> 16pt , 2 line break
- Subtitles -> 14pt, 1.5 line break
- Paragraph -> 12pt, 1.5 line break