

**Gebze Technical University
Computer Engineering**

CSE 222 - 2019 Spring

HOMEWORK 6 REPORT

**GÖKHAN HAS
161044067**

Course Assistant: Ayşe Şerbetçi TURAN

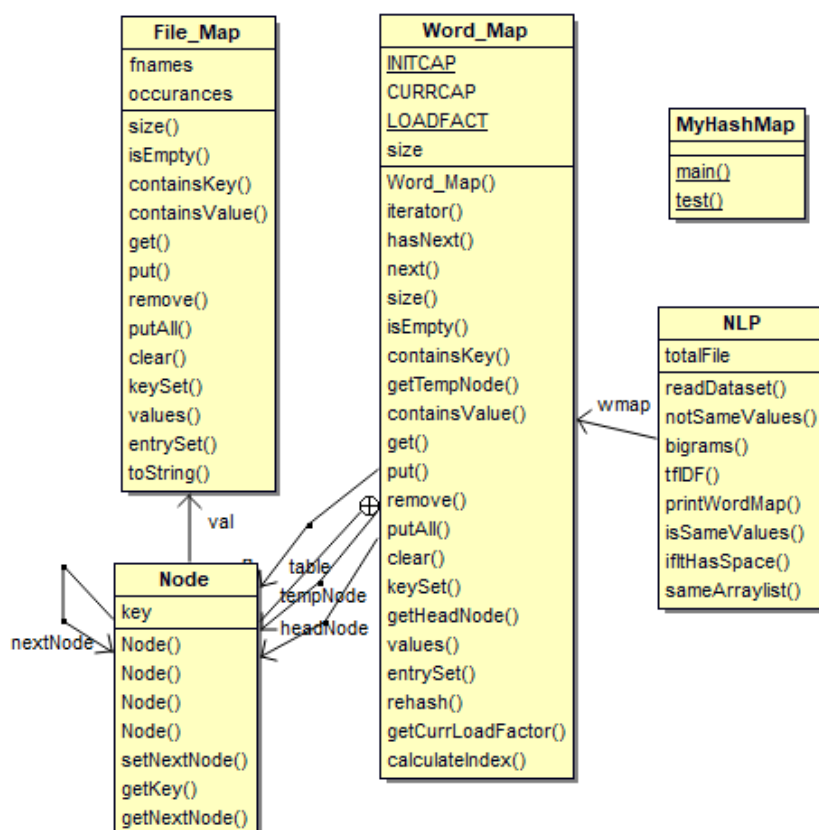
1 INTRODUCTION

1.1 Problem Definition

In this assignment, we've been given different files. We read the words in these files one by one and put them in wordMap class. The wordmap class takes the word as key and a filemap as the value. filemap is the name of the file as a key, and the value is the index of the word in the file. Then the calculation of bigram and TFIDF is requested. Input input will be given as input.txt and this will be printed on the desired output screen according to txt. When reading the file, the words should be fragmented as desired.

2 METHOD

2.1 Class Diagrams



2.2 Problem Solution Approach

In the File_Map class, two ArrayLists are given as data fields. It holds the names of the files read from the ArrayLists fnames. Occurances is keeping the index of the specified word in that file and keeping it in List <Integer>. This class, from the map interface to implement methods in the interface needs to be override. The size () method returns the size of fnames. Since fnames is an ArrayList, we can call direct fnames.size (). This process has $O(1)$ complexity. Again, the isEmpty () method has $O(1)$ complexity. Returns true if fnames is empty. The containsKey () method calls the ArrayList's contains. Therefore this process is $O(n)$. containsValue () is the same as $O(n)$, and occurrences return whether or not arrayList has the specified element. The get () method returns if there is that key in file_map. The time complexity is $O(1)$. Returns null if none. The put () method returns the new element to replace it if the element of the key is already present. If there is none, it creates and adds the new element, returns null, and $O(n)$ is because the contains () method is called. The remove () method is the same as $O(n)$. In the putAll () method I did using the linkedList. We get objects derived from a Map interface as a parameter. I have converted this object into a collection and arrayed with the toArray () method. Then I added the Put method. In this method it was $O(n)$. The clear () method cleans everything, $O(n)$. The values () method returns the values one by one. I used arrayList for the collection here. Because it was an arrayList in occurrences. For the entrySet () method, I created an object derived from the set interface and returned it by adding the keys to it. I had to do it carefully. So I chose the treemap.

The Word_Map class would behave like a hashMap. So I had to be very careful when implementing it. HashMaps also have to work in $O(1)$ to add and get methods. So we need to implement a method called table, when it is full. When rehash (), the rows are changed in table. I had to define it as a data field. I also defined a data field called headNode to use both in iterator and in some methods. It would make my job very easy. The iterator methods in Word_map work as they are known. next () object is returning. hasNext () is looking for the next element, it returns true if it exists. All these methods have $O(1)$ time complexity. Node class holds key, value and nextNode data fields. Methods $O(1)$. In the same way you () and isEmpty () methods for word_map $O(1)$. The hashCode event is triggered in the containsKey () method. The ease of using HashMap is starting here. In $O(1)$ time complexity, we define the containsKey () method. The containsValue () method is $O(n)$. Because we have to check if there is an object without knowing the key.

That's why we look at all the values. The `get ()` method in `HashMap` is $O(1)$. This gives us great convenience. The `put ()` method works in $O(1)$. Since we found index from `hasCode% table.length`, we can directly add that index to the value in that table. When the table is full, we have to do `rehash ()`. The `rehash ()` process works in $O(n)$, but the assignment of table size $3 * n + 1$ is 4 to 5 times `rehash ()`. So we can ignore this process. We don't need to implement the `remove ()` method. The `putAll ()` method retrieves objects derived from a map interface. It will have $O(n)$ because the assignment is done one by one. The `clear ()` method $O(1)$. I assign null to the table directly. The `values ()` method works in $O(n)$. Because we have to assign the values in the whole table to a collection. The `getCurrLoadFactor ()` method calculates the current `loadFactor`. Hence, $O(1)$. The `calculateIndex ()` method calculates index. Since collision is unlikely, the worst case is $O(n)$. But this process is too small to be ignored. In fact, this method is $O(1)$.

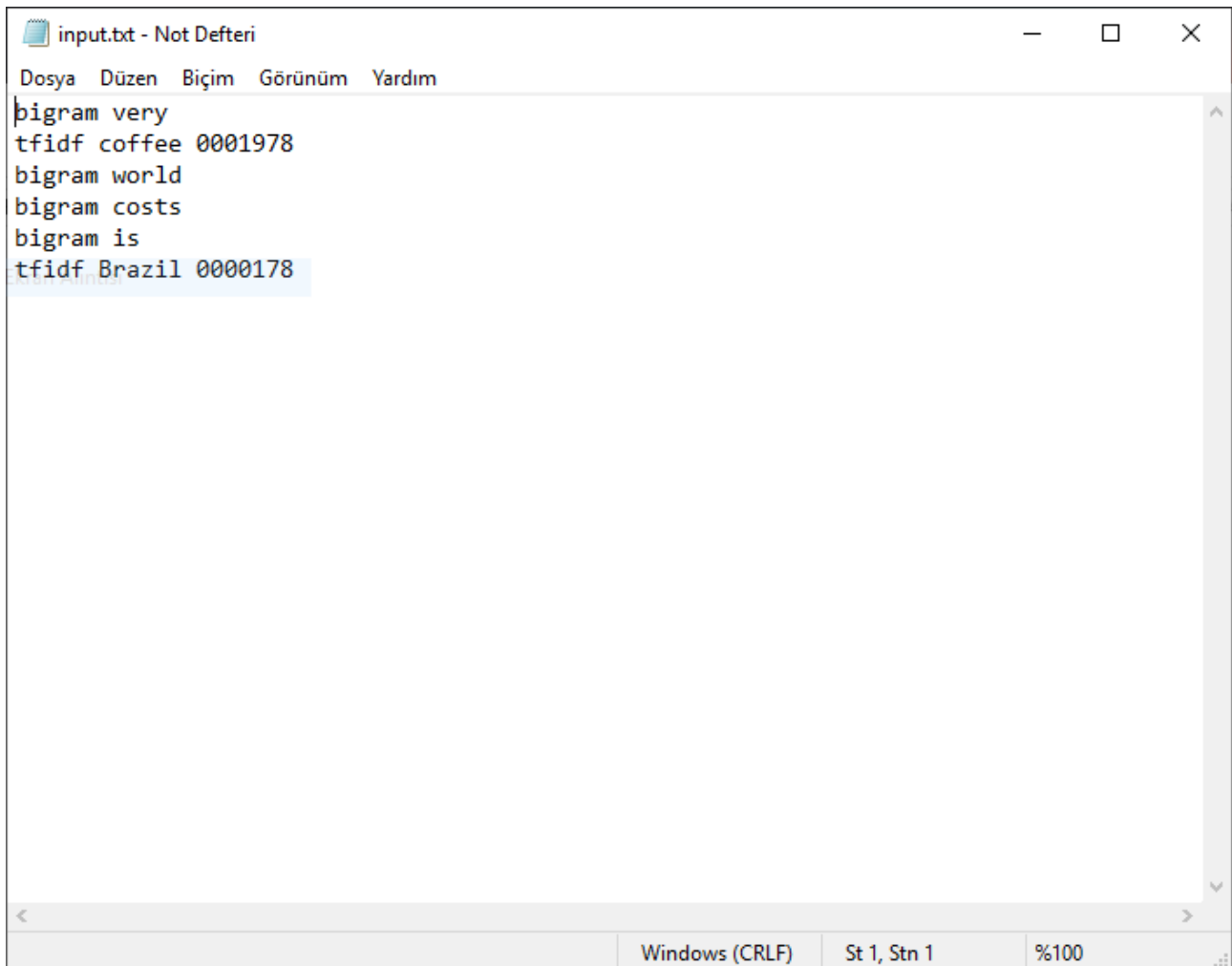
Finally we came to the `NLP` class where we will perform the main operations. In the `readDataSet ()` method, we read the files and assign them to `wordMap`. We're reading files from a folder. So we're reading all the files in that folder. Let n be the number of all files in the dataset folder. Then we read and shred words in a file. The total number of words in a file is m . Time complexity becomes $O(n * m)$. Because there are two loops and the loops rotate m and n times. I did not use any `FileMap` in the `NLP` class. I assigned direct `wmap`. When I got the same word, I added the new value to the old value because the `wmap's put ()` method returned the old value to me. Of course, I've checked the `wmap` for that word before.

For the `bigram ()` method, I decided to open the file again. Because in the `readDataSet ()` method I guess it will take more time if I assign it to a variable. I took the key and values from `wmap` and assigned them to variables. We only keep indexes in `wmap` in value. We do not hold what the word is in the value section. So I went to such a solution. I decided to return in `ArrayList` form because the method would return `List`. I returned the word and bigram by adding to `arrayList`. It was a little costly to open and close all files, but it would have been more costly if done in the other way. If we had a method to return the key values would not be needed to open the files again. I'm just reading the files that have just passed the word. You don't need to read all the files. If that word is mentioned in 5 files, only 5 files are read. If we say n the number of files passed by the word, the number of times in the specified files, we say m , time complexity $O(n * m)$ is happening.

As for the tFIDF () method, we get words and files. And we're calculating the tfidf of that word in that file. We get the key and value sent from wmap as a parameter. We are returning file_map as value and we get the size of the value in that file's name from within file_map. I have identified the TFt and IDf variables. TFt the number of times in that file / number of words in that file are doing. IDf is also specified in the pdf of the assignment. I am returning by multiplying these values. I'm opening the specified file. So this method only opens the file once. It is also used to find the total number of words in the file. If we say n the total number of words in the file, time complexity is $O(n)$.

3 RESULT

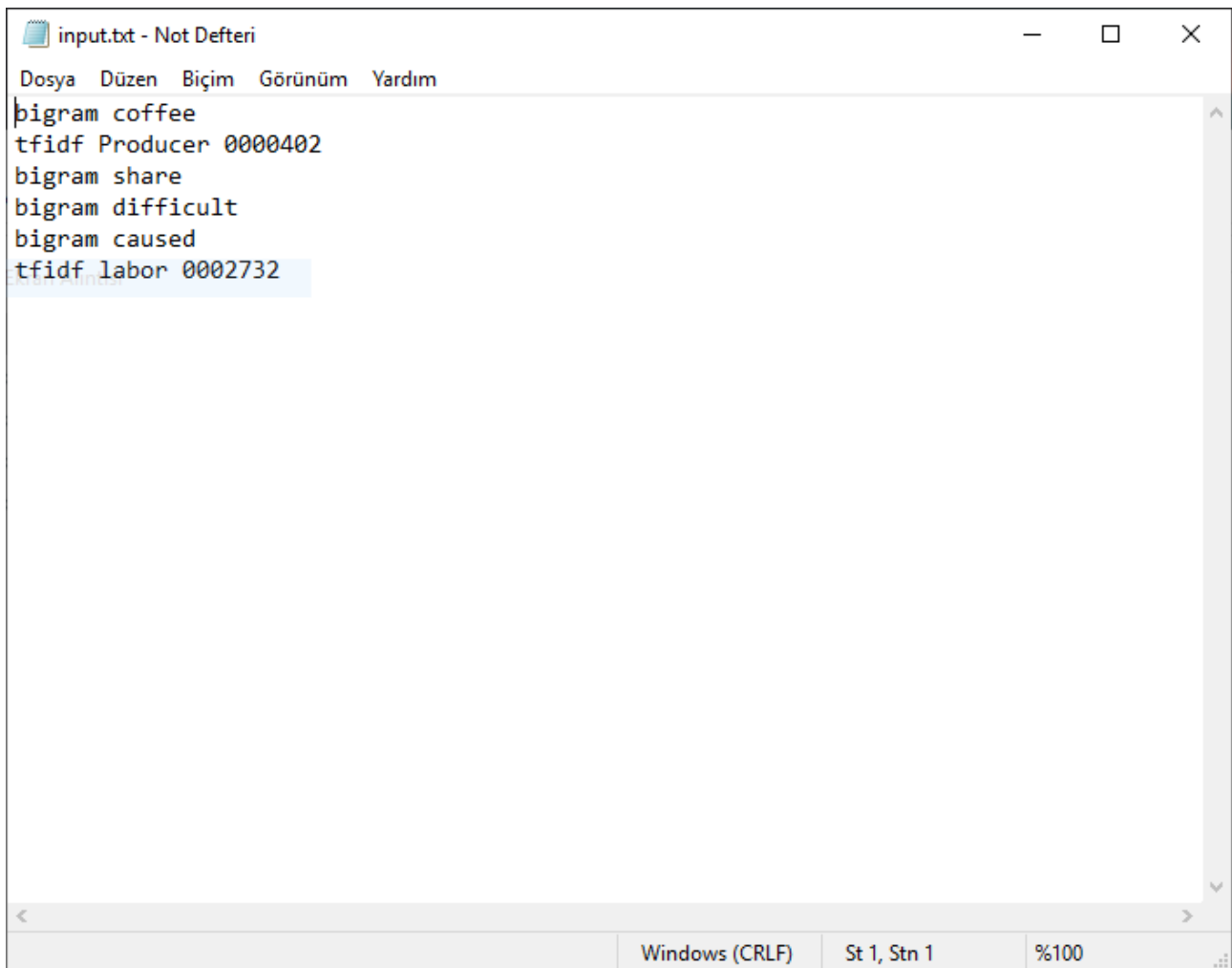
3.1 Test Cases



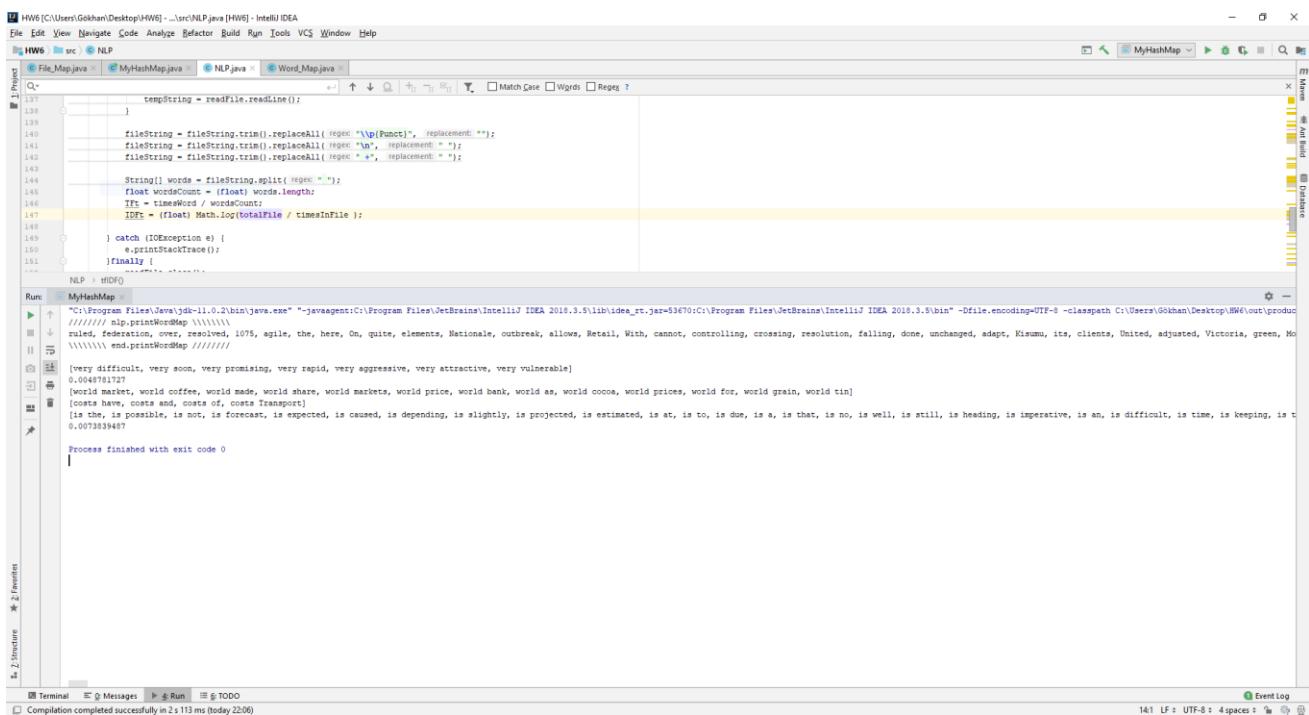
The screenshot shows a Notepad window titled "input.txt - Not Defteri". The menu bar includes "Dosya", "Düzen", "Biçim", "Görünüm", and "Yardım". The text content is as follows:

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

The last line, "tfidf Brazil 0000178", is highlighted in blue. The status bar at the bottom indicates "Windows (CRLF)", "St 1, Stn 1", and "%100".



3.2 Running Results



```
Run: MyHashMap
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.5\lib\idea_rt.jar=97110:C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.5\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Gökhan\Desktop\HW6\out\production\HW6\MyHashMap.jar
//////// nlp.printWordMap //////////
rules, federation, over, resolved, 1075, agile, the, here, on, quite, elements, Nationale, outbreak, allows, Retail, With, cannot, controlling, crossing, resolution, falling, done, unchanged, adapt, Kismu, its, clients, United, adjusted, Victoria, green, NY
//////// end-printWordMap //////////

[coffee export, coffee prices, coffee producer, coffee on, coffee market, coffee tea, coffee exports, coffee output, coffee production, coffee stocks, coffee exporting, coffee futures, coffee trade, coffee roasting, coffee dealers, coffee importer, coffee m
0.01994221
[share of, share and, share delegates, share looks, share to, share from, share under, share at, share the, share as, share Brazilian]
[difficult times, difficult to, difficult phase, difficult although, difficult for, difficult given]
[caused by, caused extensive, caused a, caused billions]
0.026661665

Process finished with exit code 0
```

Terminal Messages Run TODO

Compilation completed successfully in 1 s 692 ms (moments ago)

147:40 LF : UTF-8 : 4 spaces : Event Log

NOTE : I created a folder called Runing result and put the screenshots in that folder. You can look through that folder.