RESEARCH & PROJECT SUBMISSIONS

Program:

Course Code: CSE323

Course Name: Programming With Data Structures

Examination Committee

Prof. Hosam Fahmy

Dr. Islam El-Maddah

Ain Shams University

Faculty of Engineering

Spring Semester – 2020

Student Personal Information

Student Name :Mohamed amr ahmed taha

Student Codes:
1601247

Plagiarism Statement

I certify that this assignment / port is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student Name: **Mohamed amr ahmed taha** **Date: 30/5/2020**

Submission Contents

01 *Background*

First Topic

An indexing framework in which the terms point to documents to which the terms have a place.

An inverted index is an index into a collection of archives of the words within the archives.

Each index entry gives the word and a list of records, conceivably with areas inside the reports, where the word happens.

The inverted index data structure is a central component of a normal search engine indexing algorithm. The objective of a search engine execution is to optimize the speed of the query: get the records where word X occurs. Once a forward index is created, which stores records of words per record, it is next inverted to create an inverted index. Querying the forward index would require successive iteration through each document and to each word to confirm a matching document.

The time, memory, and processing resources to perform such a query are not practical. Rather than posting the words per record within the forward index, the inverted index data structure is created, which records the documents per word. With the inverted index, the query can presently be settled by jumping to the word ID (through arbitrary access) in the inverted index.

Random access is for the most part considered as being speedier than consecutive access.

An index structure where each key value (term) is associated with a list of objects identifiers (documents). The list contains objects that contain the given key value.

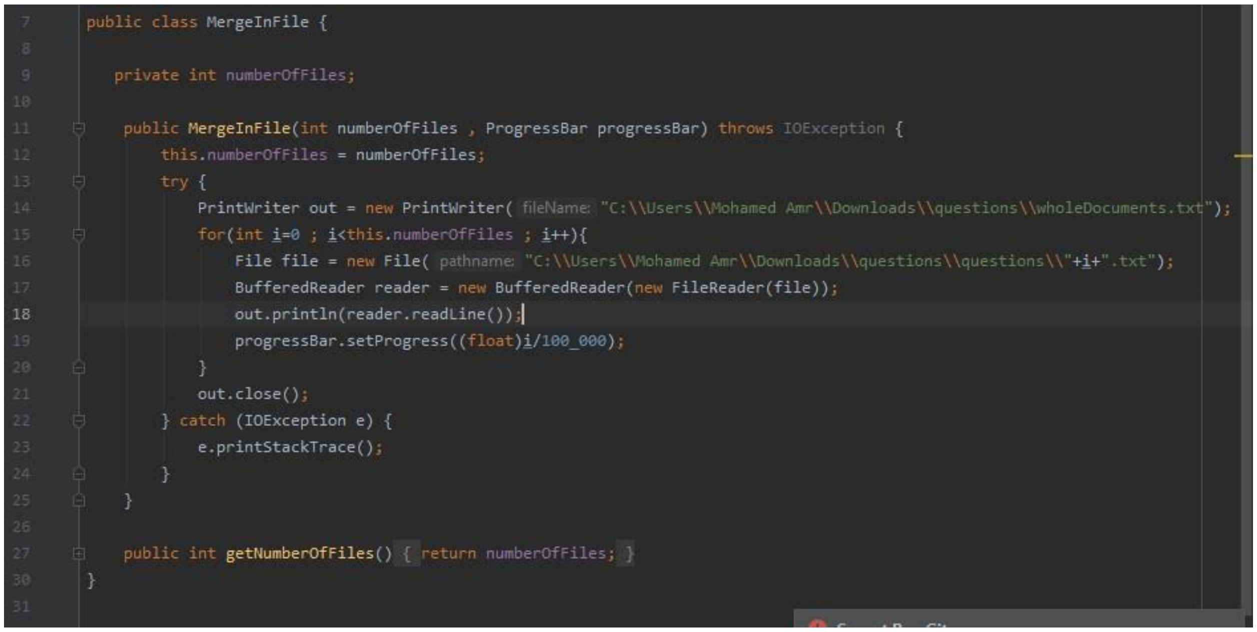
02 *Implementation*

Second Topic

Our Project is aiming to build an inverted index to make searching in a large dataset faster than using linear search algorithm, so we will show how to build an Inverted Index in the following lines…

Dataset

- The program has a large dataset consists of 100-000 document to search in, so the first thing we do is merging all texts inside these documents in one document by making every document text in a separate line. Why did we do this?
    - It saves us time in printing documents content that contains the query word.



```java
public class MergeInFile {

    private int numberOfFiles;

     public MergeInFile(int numberOfFiles , ProgressBar progressBar) throws IOException {
         this.numberOfFiles = numberOfFiles;
         try {
             PrintWriter out = new PrintWriter( fileName: "C:\\Users\\Mohamed Amr\\Downloads\\questions\\wholeDocuments.txt");
             for(int i=0 ; i<this.numberOfFiles ; i++){
                 File file = new File( pathname: "C:\\Users\\Mohamed Amr\\Downloads\\questions\\questions\\"+i+".txt");
                 BufferedReader reader = new BufferedReader(new FileReader(file));
                 out.println(reader.readLine());
                 progressBar.setProgress((float)i/100_000);
             }
             out.close();
         } catch (IOException e) {
             e.printStackTrace();
         }
     }

     public int getNumberOfFiles() { return numberOfFiles; }
}
```

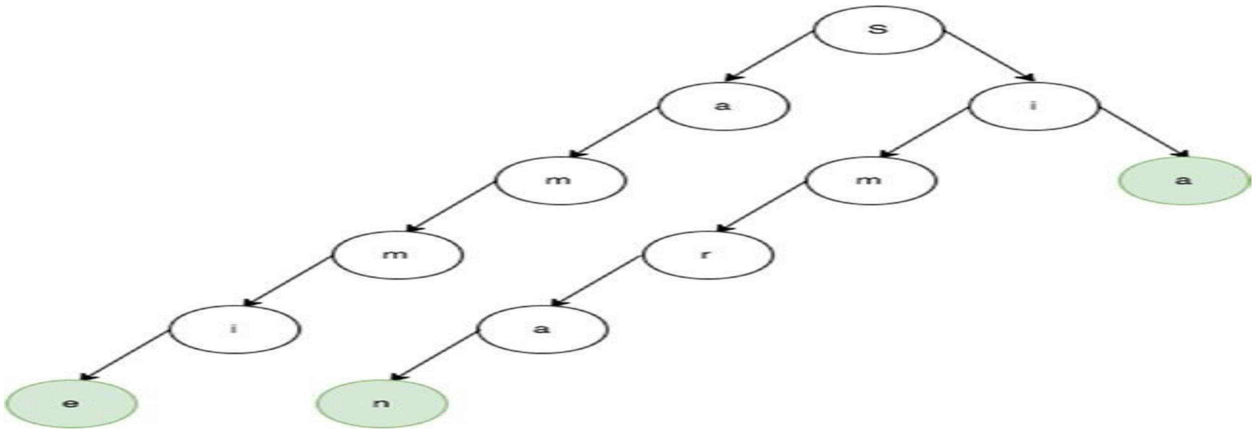
- Implementation of merging datasets:
    - We make a class called MergeInFile which has an attribute called numberOfFiles that specifies number of documents you want to merge.
    - we set the number of documents to merge in a file in the class constructor then the documents will be merged to the specified file.

Fetching data and Trie building

Before we dive in the explanation of this section first, we have to introduce our used data structure.

What is Trie?

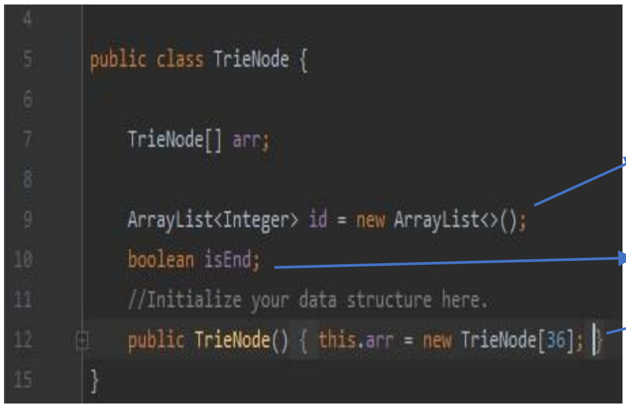
A Trie is special sort of tree used commonly for searching strings and matching on stored strings. At each level, nodes can branch off to create complete words. For instance, below figure shows a trie of the words: Sammie, Simran, Sia, Sam. Each ending node includes a Boolean flag: isCompleted. This indicates that the word ends in this path. For instance, m in Sam has endOfWord set to true. Nodes with endOfWord set to true are shaded in Figure.



How our Trie is Implemented

We make two classes for implementation:

1. TrieNode which contains:



```
public class TrieNode {

    TrieNode[] arr;

    ArrayList<Integer> id = new ArrayList<>();
    boolean isEnd;
    //Initialize your data structure here.
    public TrieNode() { this.arr = new TrieNode[36]; }
}
```

 ArrayList of id, which contains the documents ids that we found our words in

 Boolean isEnd.

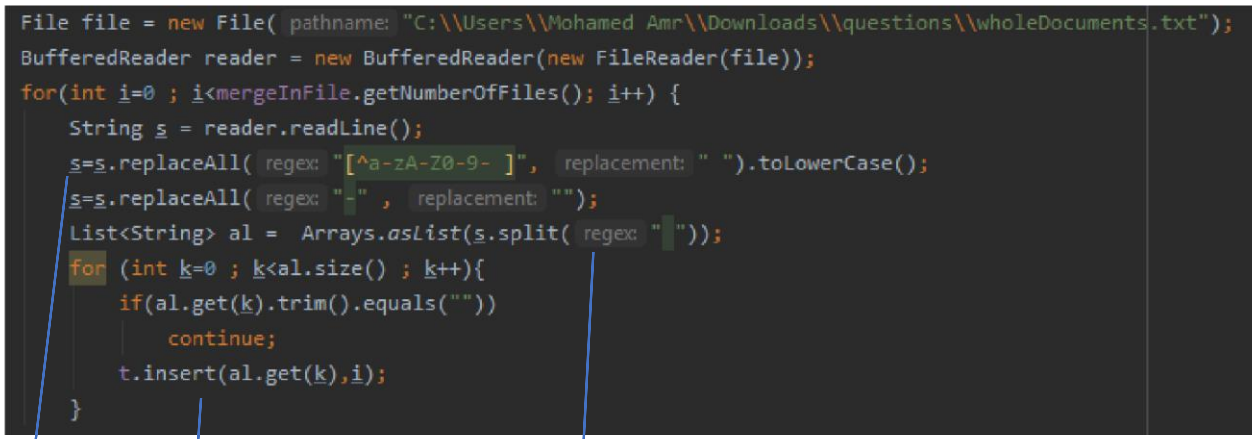
 Array of TrieNodes, which represents the characters.

2. Trie which contains:

- insert() method, which inserts a word in the trie.
    - Each character of the input key is inserted as an individual Trie node. Note that the children are an array of pointers (or references) to next level trie nodes. The key character acts as an index into the array children. In case the input key is new or an extension of the existing key, we have to construct non-existing nodes of the key, and mark end of the word for the last node. If the input key is a prefix of the existing key in Trie, we check the last node of the key as the end of a word. The key length determines Trie depth.

- If the word to be inserted is already in the trie, we have two cases:
    - ➢ The word is repeated in the same document. Thus, we compare the id of the given word to the last id in the array of the word that is in the trie.
    - ➢ The word is repeated in another document. Thus, we store the id of the given word in the id array of the word that is in the trie. Search() method, which returns true if the word found in the trie, and returns false if the word not found.

- searchNode() method, which returns reference to the word to add id and so on.
- GET_id() method, which returns reference to an ArrayList that contains the ids.



```
File file = new File( pathname: "C:\\Users\\Mohamed Amr\\Downloads\\questions\\wholeDocuments.txt");
BufferedReader reader = new BufferedReader(new FileReader(file));
for(int i=0 ; i<mergeInFile.getNumberOfFiles(); i++) {
    String s = reader.readLine();
    s=s.replaceAll( regex: "[^a-zA-Z0-9- ]", replacement: " ").toLowerCase();
    s=s.replaceAll( regex: "-" , replacement: "");
    List<String> al =  Arrays.asList(s.split( regex: " "));
    for (int k=0 ; k<al.size() ; k++){
        if(al.get(k).trim().equals(""))
            continue;
        t.insert(al.get(k),i);
    }
```

We take every line in the generated document as string then remove all special characters and leave only the numbers and alphabets.

➢ As our Trie implementation supports storing alphabets and numbers as well.

Then we split it into words which are stored in an array.

We take every word and insert it in the Trie by using insert() function in Trie class to build the inverted index.

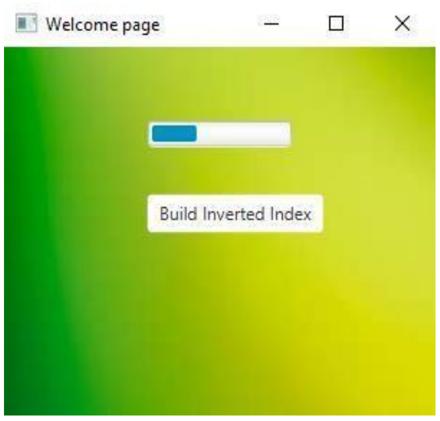
Searching

- Searching for a key is similar to insert operation, however, we only compare the characters and move down. The search can terminate due to the end of a string or lack of key in the trie.
- Using searchNode() (in Trie Class) method to get the word pointer to use it to get the list of document ids.
- In the former case, if the isCompleted field of the last node is true, then the key exists in the trie. In the second case, the search terminates without examining all the characters of the key, since the key is not present in the trie.

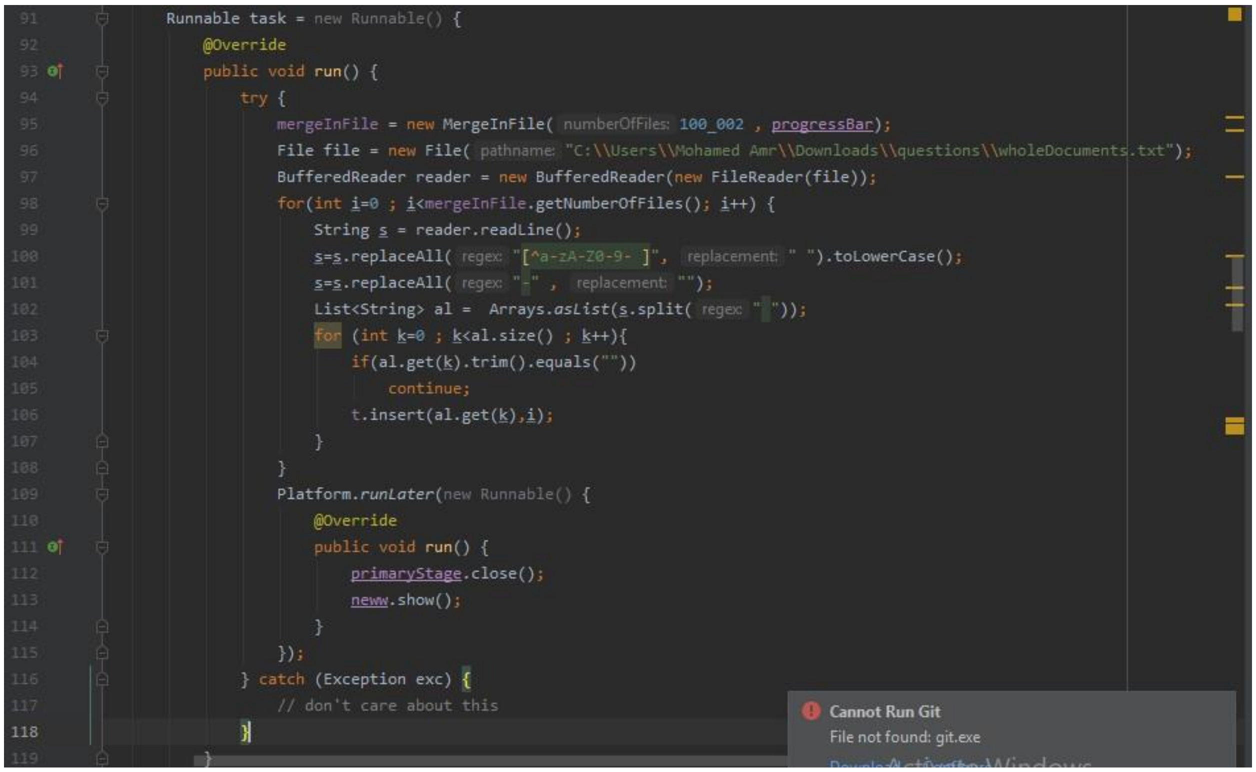
Generating output

- After searching for a certain word, we call GET_id() which returns an arrayList containing all the ids of files that have the word, then we use this arrayList to represents ids in a window in the GUI .
- To print all the content of the files that contain the word, we click "Show Documents" button to do so, which we will explain later in GUI section.

GUI



- When we click on "Build Inverted Index" button, the inverted index is built once through running the code mentioned in 'Dataset' and 'Fetching data and Trie building' sections.
- The progress bar shows success ratio of building inverted index

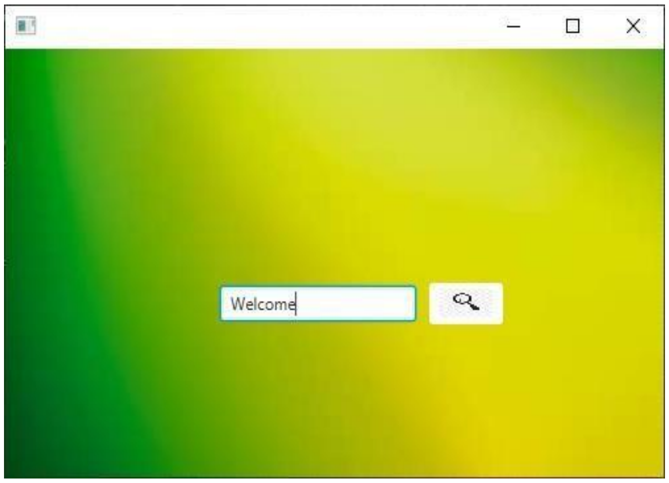


```
    Runnable task = new Runnable() {
        @Override
        public void run() {
            try {
                mergeInFile = new MergeInFile( numberOfFiles: 100_002 , progressBar);
                File file = new File( pathname: "C:\\Users\\Mohamed Amr\\Downloads\\questions\\wholeDocuments.txt");
                BufferedReader reader = new BufferedReader(new FileReader(file));
                for(int i=0 ; i<mergeInFile.getNumberOfFiles(); i++) {
                    String s = reader.readLine();
                    s=s.replaceAll( regex: "[^a-zA-Z0-9- ]",  replacement: " ").toLowerCase();
                    s=s.replaceAll( regex: "-" ,  replacement: "");
                    List<String> al =  Arrays.asList(s.split( regex: " "));
                    for (int k=0 ; k<al.size() ; k++){
                        if(al.get(k).trim().equals(""))
                            continue;
                        t.insert(al.get(k),i);
                    }
                }
                Platform.runLater(new Runnable() {
                    @Override
                    public void run() {
                        primaryStage.close();
                        neww.show();
                    }
                });
            } catch (Exception exc) {
                // don't care about this
            }
        }
```

```
Cannot Run Git
File not found: git.exe
```

As we see above we use Runnable to make the GUI interactive within the large time it is taken to building the inverted index.

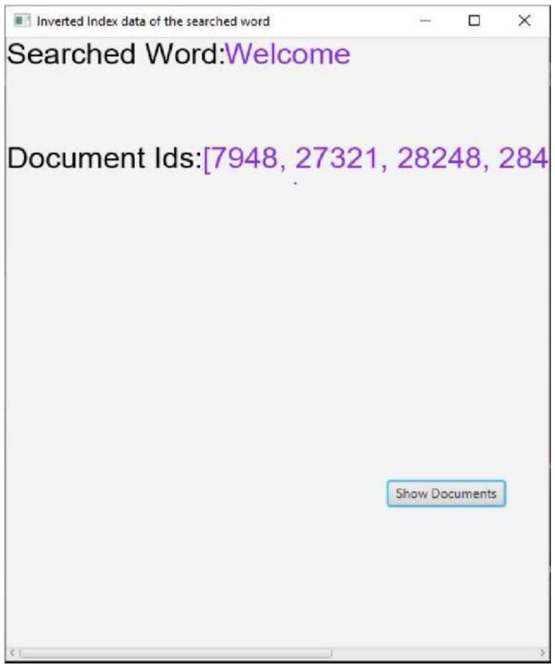
Platform.runLater is used to call UI thread to open new window after building the inverted index which is the search window.



- The text field accepts the words to be searched provided that it does not include any space characters.

- By clicking the ‘search’ icon, the code mentioned in ‘Searching’ section will be executed.

  This window represents:

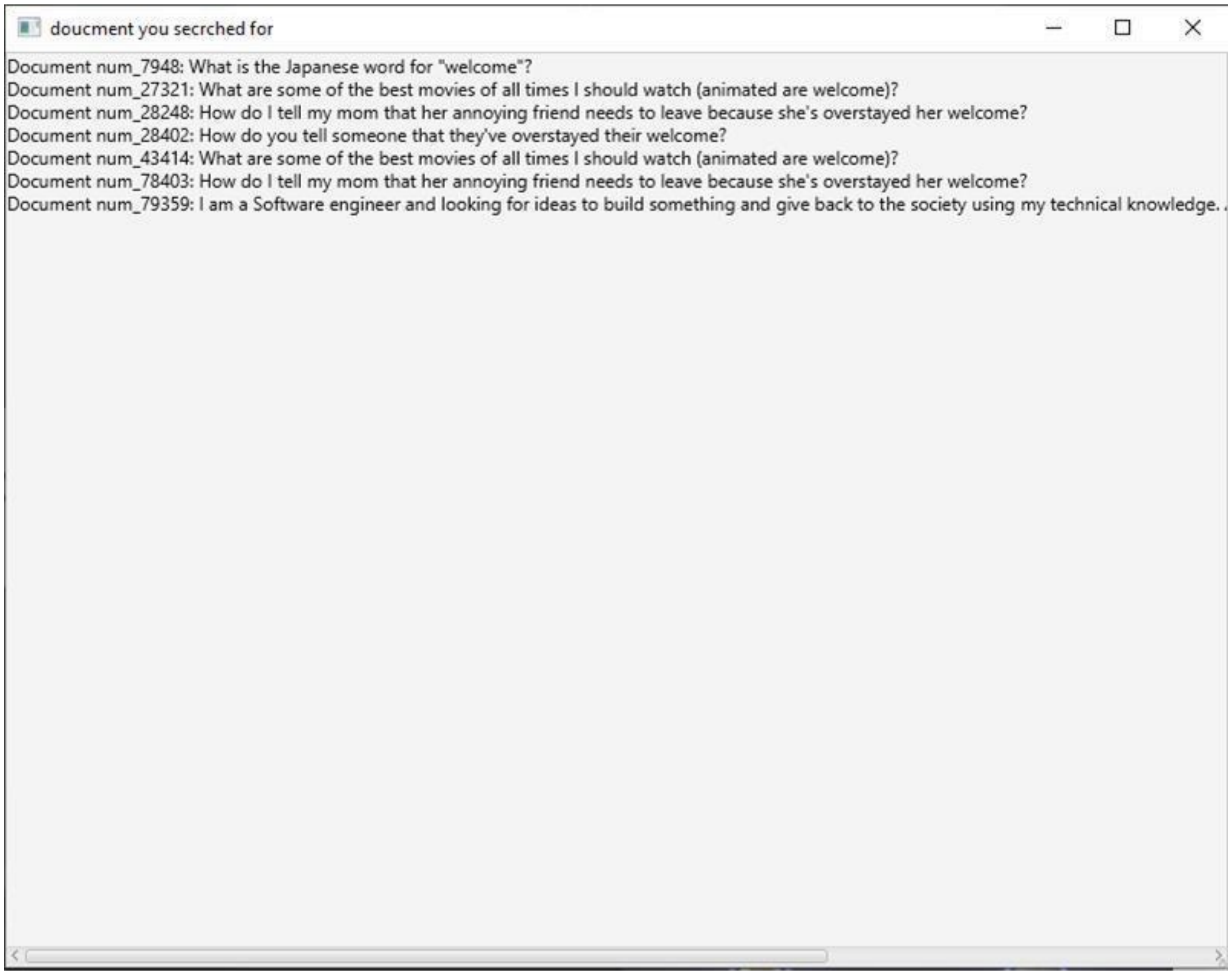


Inverted Index data of the searched word

Searched Word:Welcome

Document Ids:[7948, 27321, 28248, 284

- the word we searched for.
- Ids list representing the files containing that word.
- ‘Show Documents’ button whose functionality is explained in the next figure.



```
Document num_7948: What is the Japanese word for "welcome"?
Document num_27321: What are some of the best movies of all times I should watch (animated are welcome)?
Document num_28248: How do I tell my mom that her annoying friend needs to leave because she's overstayed her welcome?
Document num_28402: How do you tell someone that they've overstayed their welcome?
Document num_43414: What are some of the best movies of all times I should watch (animated are welcome)?
Document num_78403: How do I tell my mom that her annoying friend needs to leave because she's overstayed her welcome?
Document num_79359: I am a Software engineer and looking for ideas to build something and give back to the society using my technical knowledge.
```

- By clicking on ‘Show Documents’ button, all the files containing the word we searched for will be shown in another window of the GUI.

03 Complexity of Operation

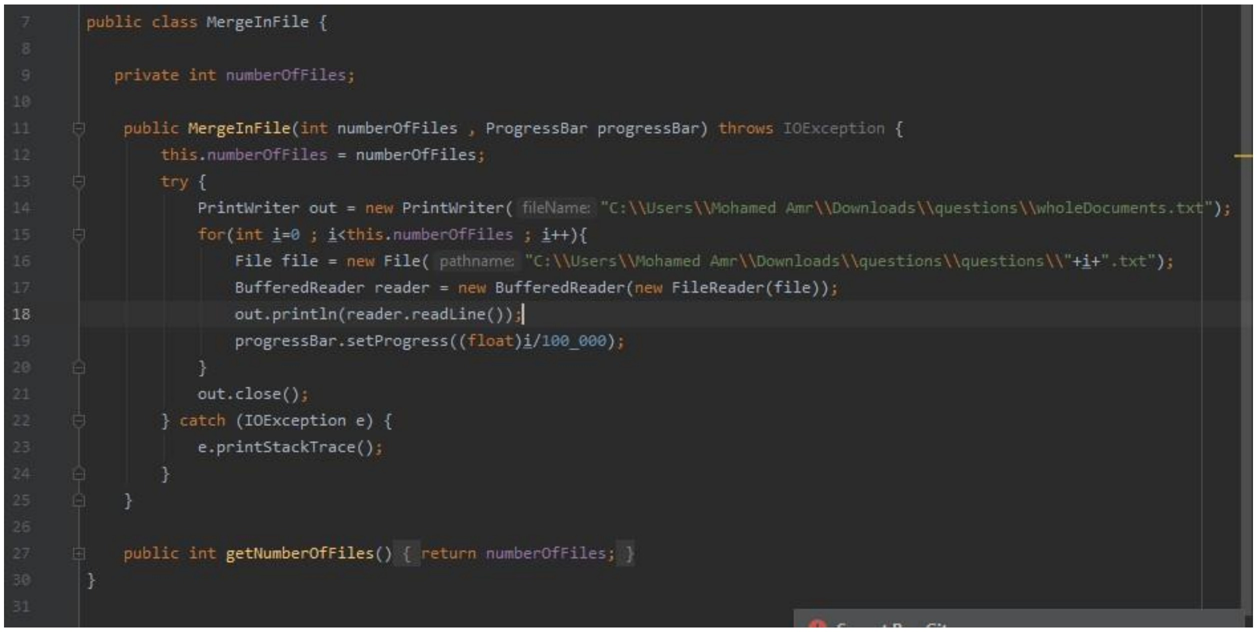
Third Topic

Talking about our algorithm complexity, we will divide our main program into two sections:

- Building the inverted index, which, by definition, has to be more complex than any part of our project.
- Searching and Generating output, which has to be as fast as possible.

Building The Inverted Index

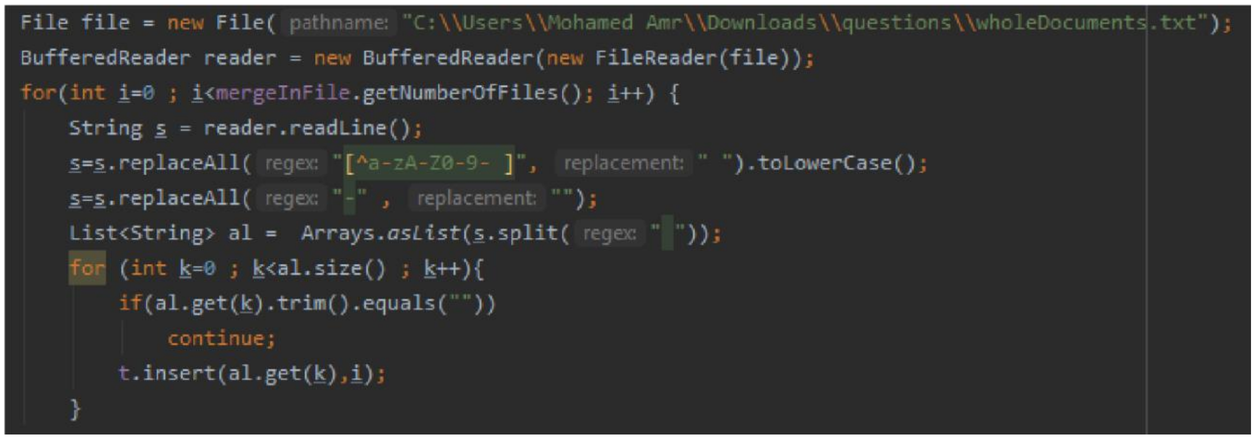
We can divide this operation into 3 steps:



```
public class MergeInFile {

   private int numberOfFiles;

    public MergeInFile(int numberOfFiles , ProgressBar progressBar) throws IOException {
        this.numberOfFiles = numberOfFiles;
        try {
            PrintWriter out = new PrintWriter( fileName: "C:\\Users\\Mohamed Amr\\Downloads\\questions\\wholeDocuments.txt");
            for(int i=0 ; i<this.numberOfFiles ; i++){
                File file = new File( pathname: "C:\\Users\\Mohamed Amr\\Downloads\\questions\\questions\\"+i+".txt");
                BufferedReader reader = new BufferedReader(new FileReader(file));
                out.println(reader.readLine());
                progressBar.setProgress((float)i/100_000);
            }
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public int getNumberOfFiles() { return numberOfFiles; }
}
```

- Creating ‘whole Documents’ file which contains the content all data files.

Complexity: O(N) where N is the number of files.



```
File file = new File( pathname: "C:\\Users\\Mohamed Amr\\Downloads\\questions\\wholeDocuments.txt");
BufferedReader reader = new BufferedReader(new FileReader(file));
for(int i=0 ; i<mergeInFile.getNumberOfFiles(); i++) {
    String s = reader.readLine();
    s=s.replaceAll( regex: "[^a-zA-Z0-9- ]", replacement: " ").toLowerCase();
    s=s.replaceAll( regex: "-" , replacement: "");
    List<String> al =  Arrays.asList(s.split( regex: " "));
    for (int k=0 ; k<al.size() ; k++){
        if(al.get(k).trim().equals(""))
            continue;
        t.insert(al.get(k),i);
    }
```

- Fetching data from that large file. Complexity: O(N*W)

where N is the number of files, and W is the number of words in each file.

- Storing a word in the Trie. Complexity: O(L) where L

is the length of the word.

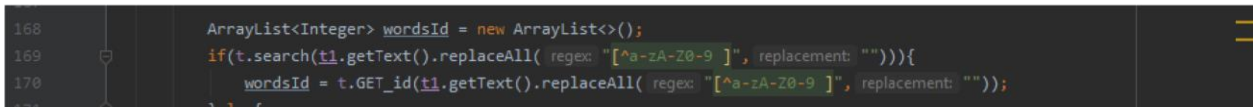
Thus, Building the inverted index complexity is as follows:

Creating ‘whole Documents: O(N)

Fetching data and storing in trie: O(N*W*L)

NOTICE: This part is going to be executed only once.

Searching and Generating output

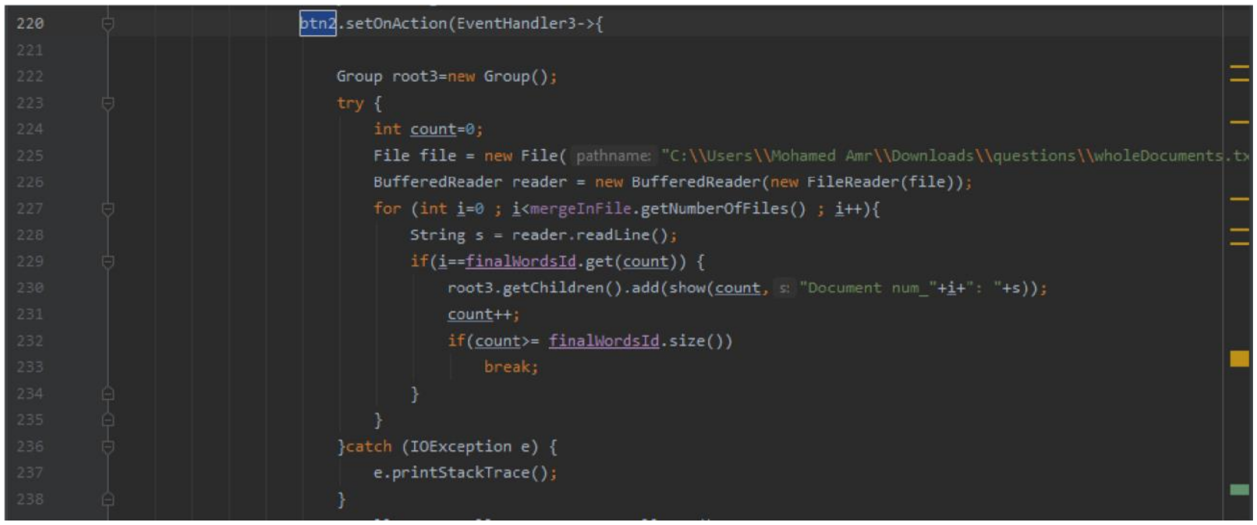


```
ArrayList<Integer> wordsId = new ArrayList<>();
if(t.search(t1.getText().replaceAll( regex: "[^a-zA-Z0-9 ]", replacement: ""))){
    wordsId = t.GET_id(t1.getText().replaceAll( regex: "[^a-zA-Z0-9 ]", replacement: ""));
```

We can divide this operation into 2 steps:

- Finding the word in the trie:

Complexity: O(L), where L is the length of the word.



```
                    btn2.setOnAction(EventHandler3->{

                        Group root3=new Group();
                        try {
                            int count=0;
                            File file = new File( pathname: "C:\\Users\\Mohamed Amr\\Downloads\\questions\\wholeDocuments.tx
                            BufferedReader reader = new BufferedReader(new FileReader(file));
                            for (int i=0 ; i<mergeInFile.getNumberOfFiles() ; i++){
                                String s = reader.readLine();
                                if(i==finalWordsId.get(count)) {
                                    root3.getChildren().add(show(count, s: "Document num_"+i+": "+s));
                                    count++;
                                    if(count>= finalWordsId.size())
                                        break;
                                }
                            }
                        }catch (IOException e) {
                            e.printStackTrace();
                        }
```

- Printing all the files containing that word:

  Complexity: O(N), where N is the total number of files (data set).

Note: if the last file contains the searched word is file number 100 then it will loop 100 times so it depends on the index of the last file.

Links

Inverted Index Source Code Link on GitHub
https://github.com/gittatohubato/InvertedIndex-GUI Video
Link on Google Drive

https://drive.google.com/file/d/1Y3Qoa7AgS0_0SjFKWXqPKpZs3x9x-XW2/view?usp=sharing

References

[1] JavaScript Data Structures and Algorithms an Introduction to Understanding and Implementing Core Data Structure and Algorithm Fundamentals Page 303

[2] (Yosifovich, Ionescu, Russinovich and Solomon, n.d.)