Web Search Engine:

Advance Computing Concepts
Final Project (COMP8547)

Instructor: Dr Mahdi Firoozjaei

Team Roles

Arun Reddy Nalla **110088379**	**-**	**Hashing, Ranking**
Siddhartha Pitchika **110089009**	**-**	**Searching (BoyerMoore)**
Kalyan Venkatesh Poludasu **110089120**	**-**	**Spell Check (Edit Distan**
Veera Venkata Bharat Kumar Vayitla **110088432**	**-**	**Crawling, HTML to Text**

INTRODUCTION

FLOW DIAGRAM

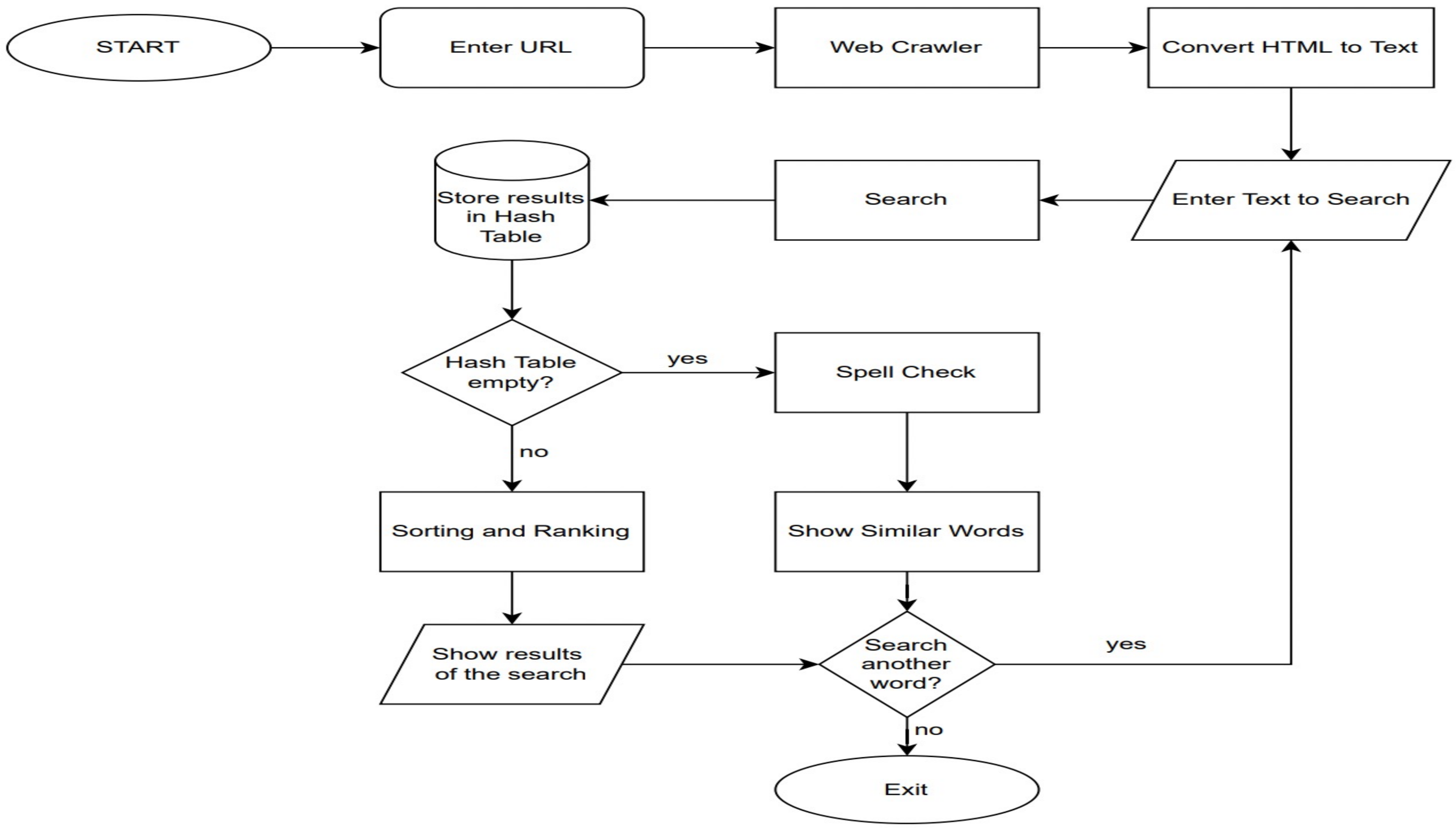
FEATURES

DEMO

Web Search Engine

An application created specifically to do web searches is known as a search engine. For any specific information specified in a text-based online search query, they conduct systematic searches of the World Wide Web and return us the weblink which matches the given information.

Flow Diagram

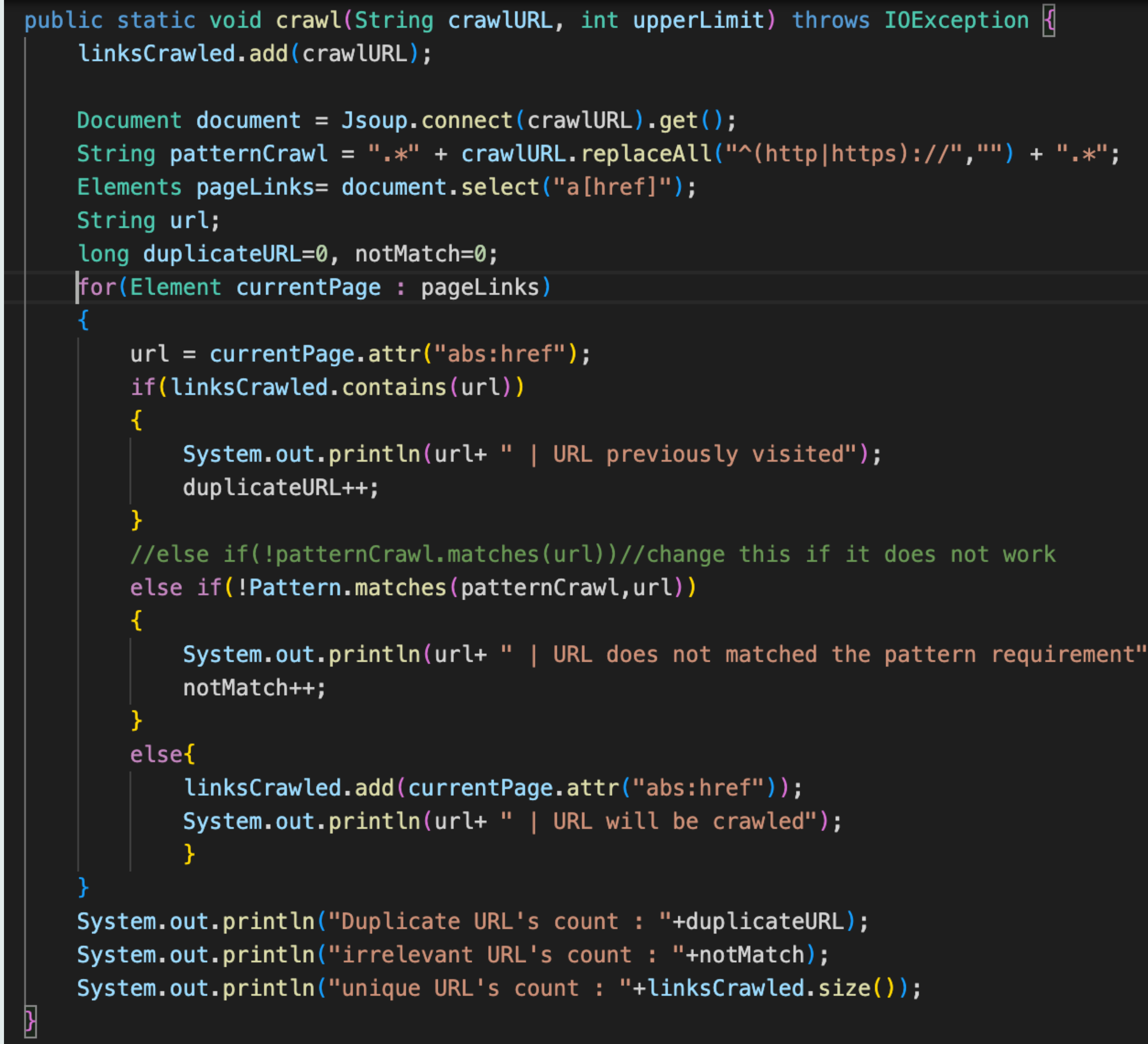


Crawler

- Discovers and scans HTML from URL
- In our project using href attribute crawls the websites
- Crawls and adds them to HashSet if given pattern matched
- Jsoup – Library used

Crawler

Crawls URLs and adds them to Hash Set and prints the count of already visited, unique links and irrelevant



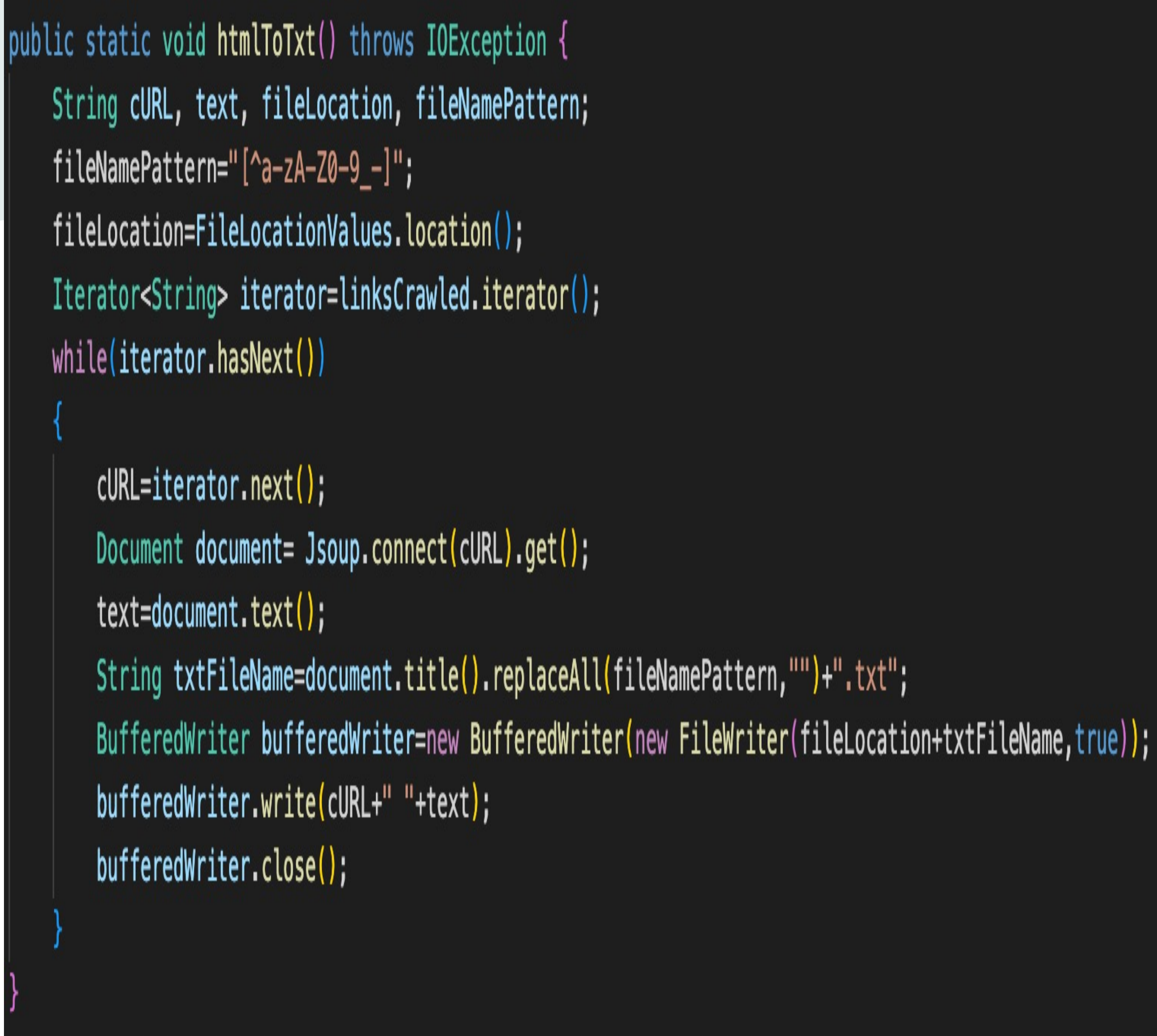
```java
public static void crawl(String crawlURL, int upperLimit) throws IOException {
    linksCrawled.add(crawlURL);

    Document document = Jsoup.connect(crawlURL).get();
    String patternCrawl = ".*" + crawlURL.replaceAll("^(http|https)://","") + ".*";
    Elements pageLinks= document.select("a[href]");
    String url;
    long duplicateURL=0, notMatch=0;
    for(Element currentPage : pageLinks)
    {
        url = currentPage.attr("abs:href");
        if(linksCrawled.contains(url))
        {
            System.out.println(url+ " | URL previously visited");
            duplicateURL++;
        }
        //else if(!patternCrawl.matches(url))//change this if it does not work
        else if(!Pattern.matches(patternCrawl,url))
        {
            System.out.println(url+ " | URL does not matched the pattern requirement"
            notMatch++;
        }
        else{
            linksCrawled.add(currentPage.attr("abs:href"));
            System.out.println(url+ " | URL will be crawled");
            }
    }
    System.out.println("Duplicate URL's count : "+duplicateURL);
    System.out.println("irrelevant URL's count : "+notMatch);
    System.out.println("unique URL's count : "+linksCrawled.size());
}
```

HTML to Text

- Now the HTML file will be converted Text file
- Gets the URLs from the hash set then it connects and loads the HTML file

HTML to Text

Now we will Iterate through the links in the Hash Set and connects to that URLs and get the HTML file and converts it to TEXT and saved to the Folder



```java
public static void htmlToTxt() throws IOException {
    String cURL, text, fileLocation, fileNamePattern;
    fileNamePattern="[^a-zA-Z0-9_-]";
    fileLocation=FileLocationValues.location();
    Iterator<String> iterator=linksCrawled.iterator();
    while(iterator.hasNext())
    {
        cURL=iterator.next();
        Document document= Jsoup.connect(cURL).get();
        text=document.text();
        String txtFileName=document.title().replaceAll(fileNamePattern,"")+".txt";
        BufferedWriter bufferedWriter=new BufferedWriter(new FileWriter(fileLocation+txtFileName,true));
        bufferedWriter.write(cURL+" "+text);
        bufferedWriter.close();
    }
}
```

Searching

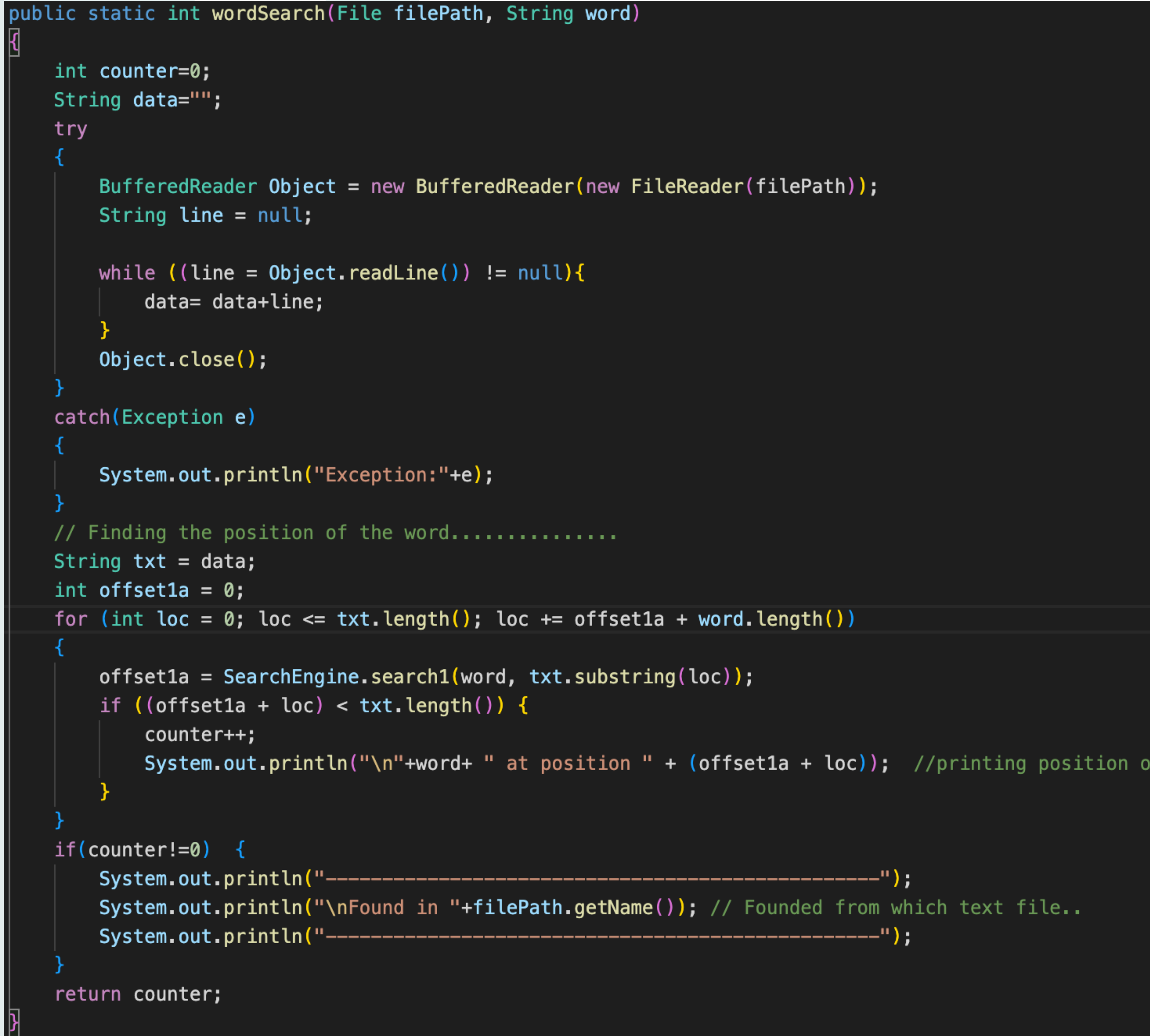
Considers the input from the user and searches the input from the text file.

For searching in the input text file, we are using Boyer Moore algorithm.

The algorithm counting the number of times the text appears in each file and storing it in a hash table

Searching

- The words will be iterated and searched in all the text files.
- Consider each file to be a set of characters that must be matched using the Boyer Moore Algorithm.



```
public static int wordSearch(File filePath, String word)
{

    int counter=0;
    String data="";
    try
    {
        BufferedReader Object = new BufferedReader(new FileReader(filePath));
        String line = null;

        while ((line = Object.readLine()) != null){
            data= data+line;
        }
        Object.close();
    }
    catch(Exception e)
    {
        System.out.println("Exception:"+e);
    }
    // Finding the position of the word..............
    String txt = data;
    int offset1a = 0;
    for (int loc = 0; loc <= txt.length(); loc += offset1a + word.length())
    {
        offset1a = SearchEngine.search1(word, txt.substring(loc));
        if ((offset1a + loc) < txt.length()) {
            counter++;
            System.out.println("\n"+word+ " at position " + (offset1a + loc));  //printing position o
        }
    }
    if(counter!=0)  {
        System.out.println("--------------------------------------------------");
        System.out.println("\nFound in "+filePath.getName()); // Founded from which text file..
        System.out.println("--------------------------------------------------");
    }
    return counter;
}
```

Hashing

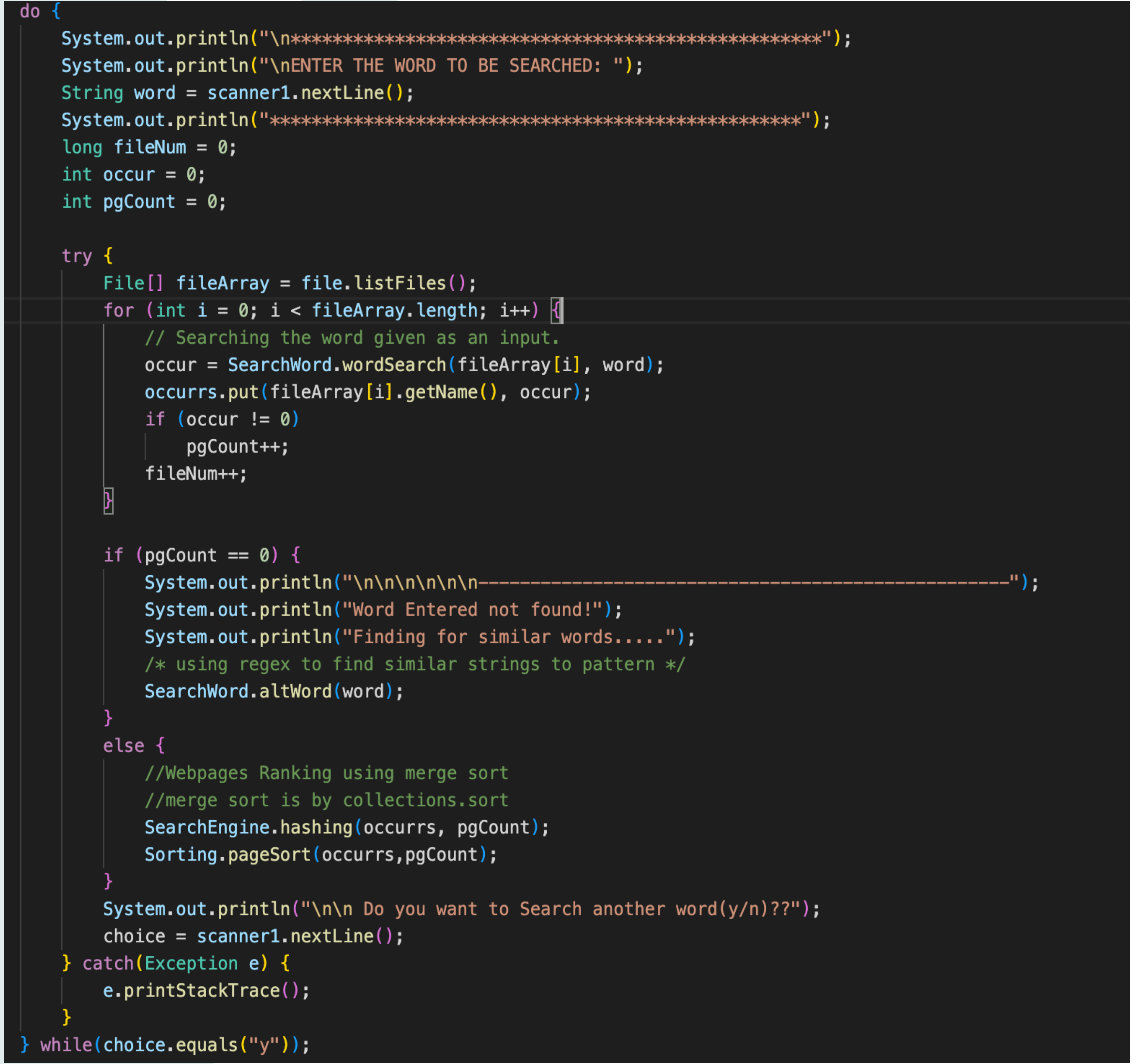
The results of a search are stored in a hash table.

File names are recorded as the key, while the number of occurrences in that file is saved as the value.

This hash table is also used for sorting and ranking.

Hash table

- Using a separate chaining mechanism, a hash table will save all the records from a search operation.
- Once 75 percent of the table is filled, the hash table's capacity will grow.



```java
do {
    System.out.println("\n***************************************************");
    System.out.println("\nENTER THE WORD TO BE SEARCHED: ");
    String word = scanner1.nextLine();
    System.out.println("***************************************************");
    long fileNum = 0;
    int occur = 0;
    int pgCount = 0;

    try {
        File[] fileArray = file.listFiles();
        for (int i = 0; i < fileArray.length; i++) {
            // Searching the word given as an input.
            occur = SearchWord.wordSearch(fileArray[i], word);
            occurrs.put(fileArray[i].getName(), occur);
            if (occur != 0)
                pgCount++;
            fileNum++;
        }

        if (pgCount == 0) {
            System.out.println("\n\n\n\n\n\n---------------------------------------------------");
            System.out.println("Word Entered not found!");
            System.out.println("Finding for similar words.....");
            /* using regex to find similar strings to pattern */
            SearchWord.altWord(word);
        }
        else {
            //Webpages Ranking using merge sort
            //merge sort is by collections.sort
            SearchEngine.hashing(occurrs, pgCount);
            Sorting.pageSort(occurrs,pgCount);
        }
        System.out.println("\n\n Do you want to Search another word(y/n)??");
        choice = scanner1.nextLine();
    } catch(Exception e) {
        e.printStackTrace();
    }
} while(choice.equals("y"));
```

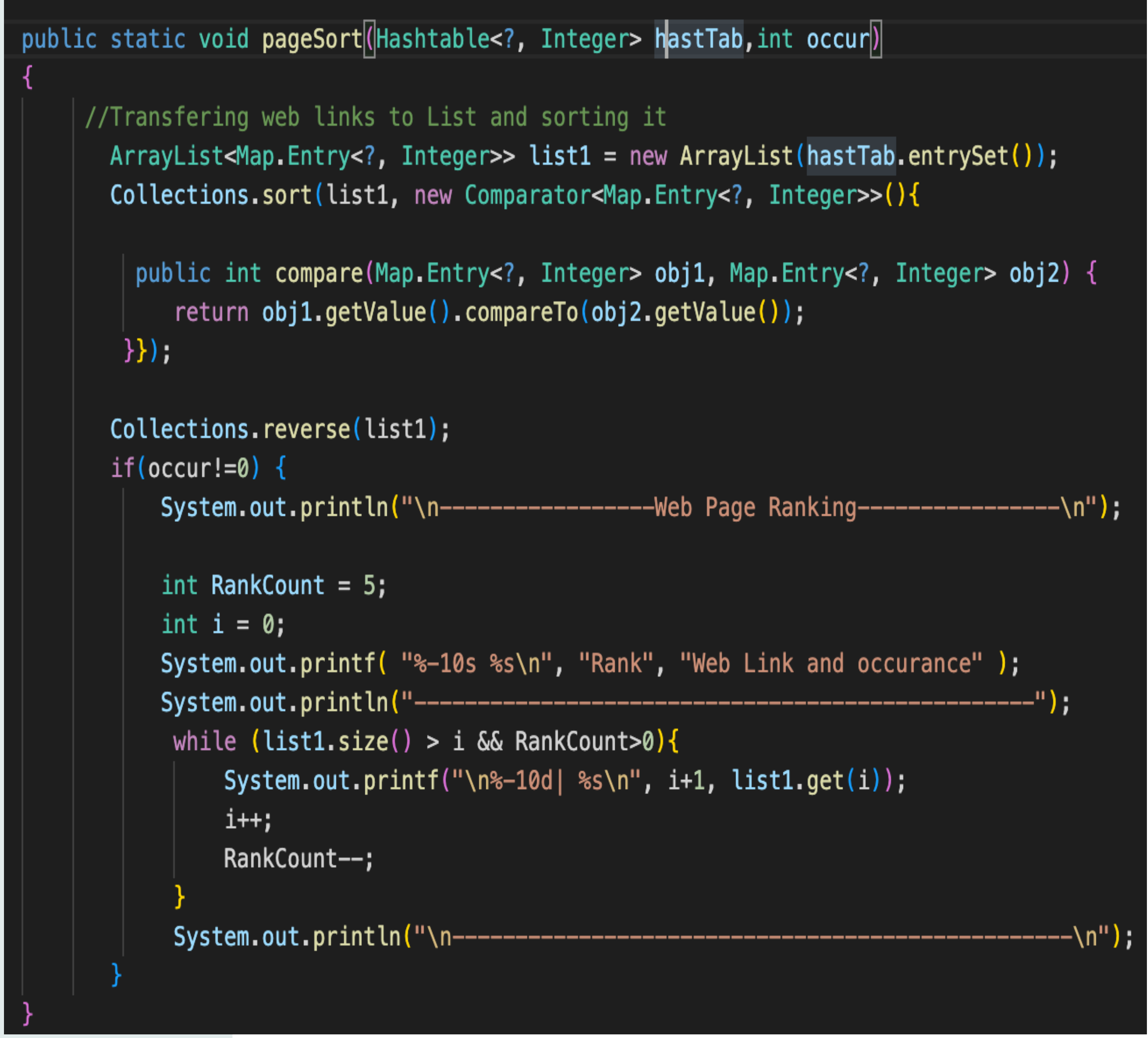
Ranking

The project shows the url of the webpages where the keyword appears the most frequently and sorts them by frequency.

Sorting algorthim is used for ranking webpages.

Ranking

Hash Table having URLs will be sorted with respect to the occurrences of the word searched on the web page and Printing the top five web links with occurrence count.

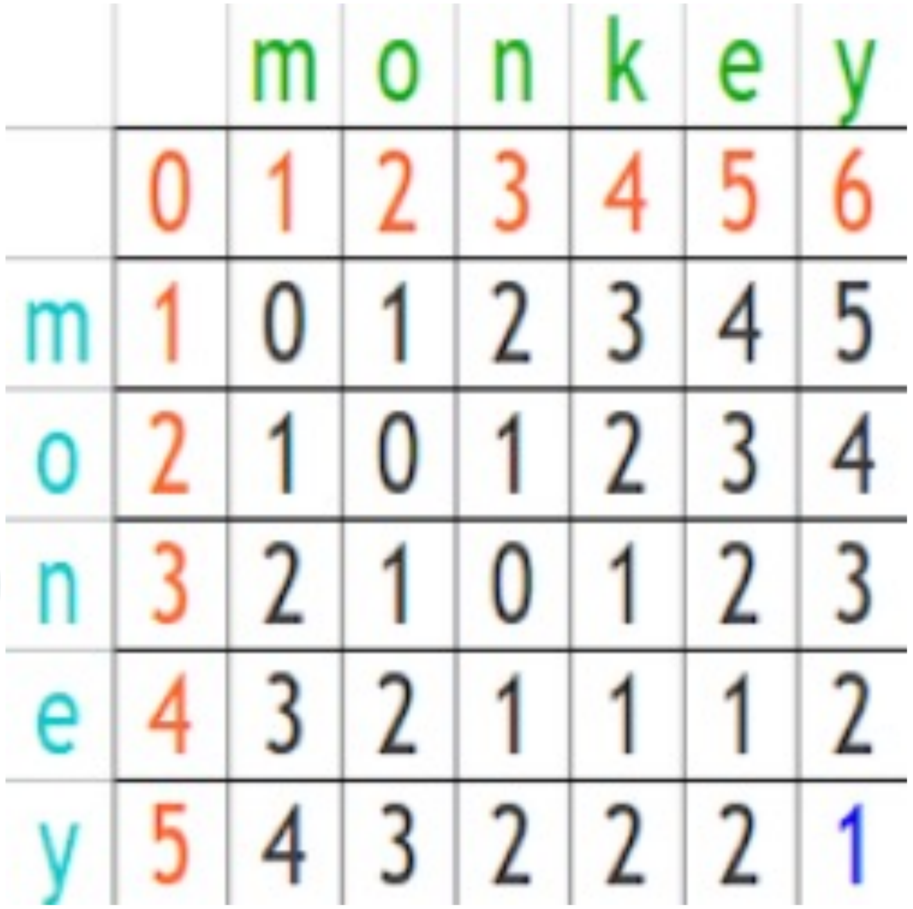


```java
public static void pageSort(Hashtable<?, Integer> hastTab,int occur)
{
        //Transfering web links to List and sorting it
          ArrayList<Map.Entry<?, Integer>> list1 = new ArrayList(hastTab.entrySet());
          Collections.sort(list1, new Comparator<Map.Entry<?, Integer>>(){

            public int compare(Map.Entry<?, Integer> obj1, Map.Entry<?, Integer> obj2) {
                return obj1.getValue().compareTo(obj2.getValue());
           }});

          Collections.reverse(list1);
          if(occur!=0) {
              System.out.println("\n---------------Web Page Ranking---------------\n");

              int RankCount = 5;
              int i = 0;
              System.out.printf( "%-10s %s\n", "Rank", "Web Link and occurance" );
              System.out.println("---------------------------------------------");
               while (list1.size() > i && RankCount>0){
                    System.out.printf("\n%-10d| %s\n", i+1, list1.get(i));
                    i++;
                    RankCount--;
               }
               System.out.println("\n---------------------------------------------\n");
          }
}
```

Edit Distance

- We get edit distance by calculating how many edit operations are required to change one word to another word.

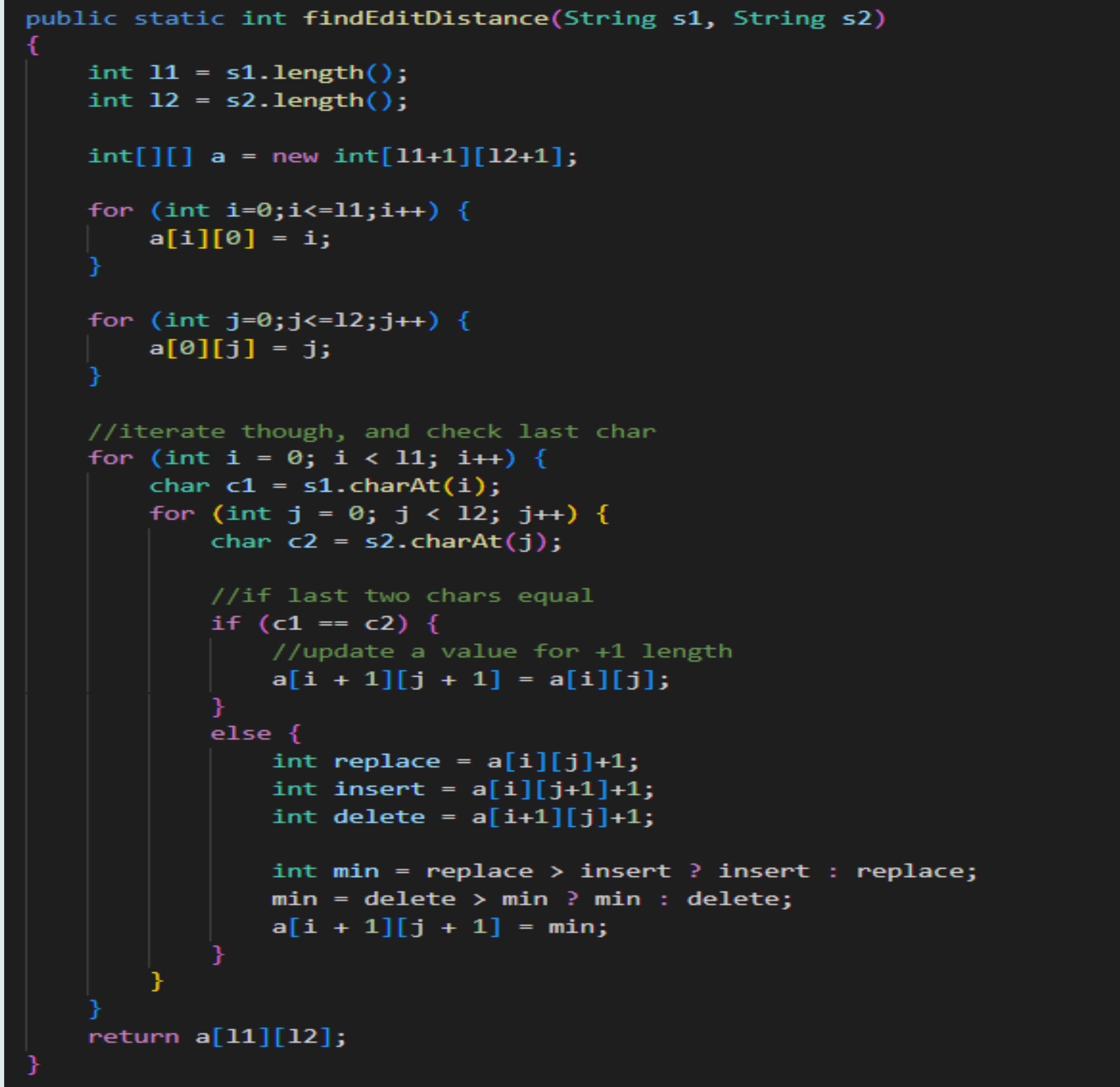
- Operations
    - Insertion
    - Deletion
    - Replacement



		m	o	n	k	e	y
	0	1	2	3	4	5	6
m	1	0	1	2	3	4	5
o	2	1	0	1	2	3	4
n	3	2	1	0	1	2	3
e	4	3	2	1	1	1	2
y	5	4	3	2	2	2	1

Edit Distance

- Tokenizing and storing all text files in a list
- The input is matched against each distinct string in the list, and the word with the smallest edit distance is given back as the output.



```java
public static int findEditDistance(String s1, String s2)
{
    int l1 = s1.length();
    int l2 = s2.length();

    int[][] a = new int[l1+1][l2+1];

    for (int i=0;i<=l1;i++) {
        a[i][0] = i;
    }

    for (int j=0;j<=l2;j++) {
        a[0][j] = j;
    }

    //iterate though, and check last char
    for (int i = 0; i < l1; i++) {
        char c1 = s1.charAt(i);
        for (int j = 0; j < l2; j++) {
            char c2 = s2.charAt(j);

            //if last two chars equal
            if (c1 == c2) {
                //update a value for +1 length
                a[i + 1][j + 1] = a[i][j];
            }
            else {
                int replace = a[i][j]+1;
                int insert = a[i][j+1]+1;
                int delete = a[i+1][j]+1;

                int min = replace > insert ? insert : replace;
                min = delete > min ? min : delete;
                a[i + 1][j + 1] = min;
            }
        }
    }
    return a[l1][l2];
}
```

DEMO

Thank You