Web Search Engine:

COMP8547 – Advanced Computing Concepts Final Project

Instructor: Dr. Mahdi Firoozjaei

This is where your searching begins

University of Windsor

Team Roles

Abhishek Patel

- 110068457

- Spell Checker
- Ranking

Jaydeep Dharamsey

- 110088507

- Searching
- Hashing

Raj Manoj Dedhia

- 110088375

- Crawling
- HTML to Text

Project Link... https://github.com/abhishekpatelmc/Uwin_ACC_Search_Engine

Click Here

University of Windsor

01 Introduction

What is a Web Search Engine?

02 Workflow Diagram

How does our Web Search Engine works?

03 Features Description

How this feature works?

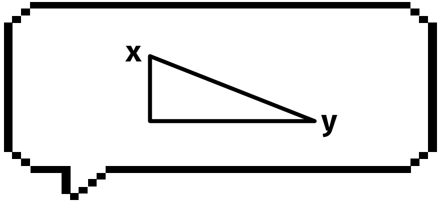
04 Demo

Let's have a look on our Web Search Engine now

05 Our Team

Our Project Team

University of Windsor



01

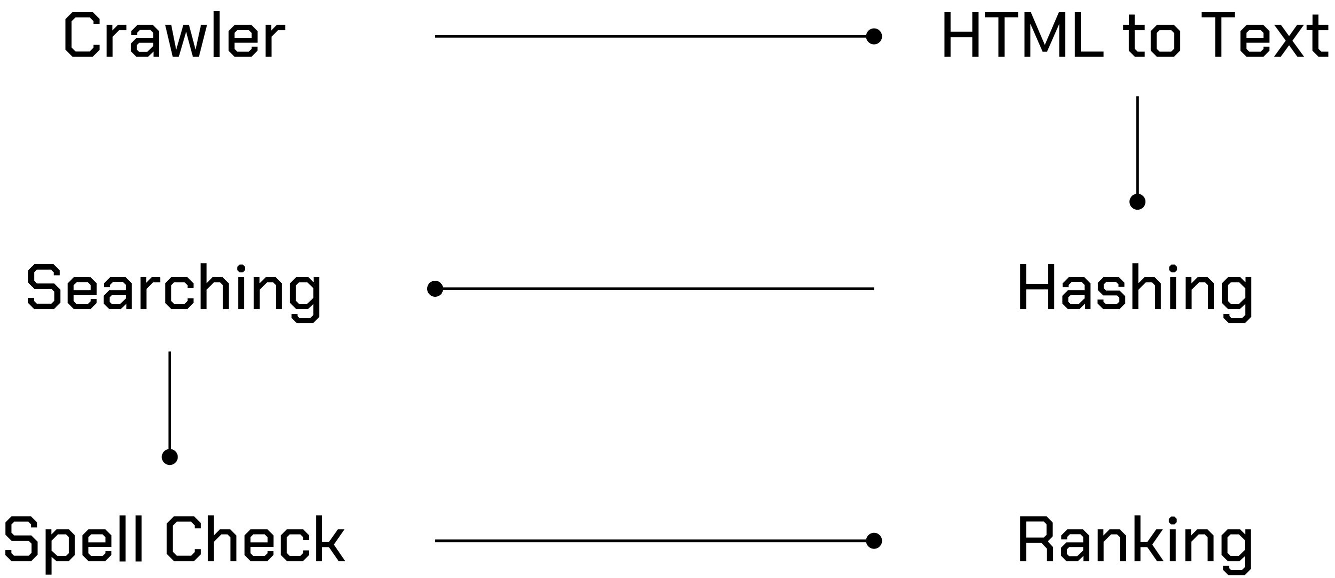
Introduction

What is Search Engine ?

An application created specifically to do web searches is known as a search engine.

Search engine will look at many web pages to find matches to the user's search inputs. It will return results ranked by relevancy and popularity by the search engine.

Core Modules



Crawler

HTML to Text

Hashing

Searching

Spell Check

Ranking

University of Windsor

Hello

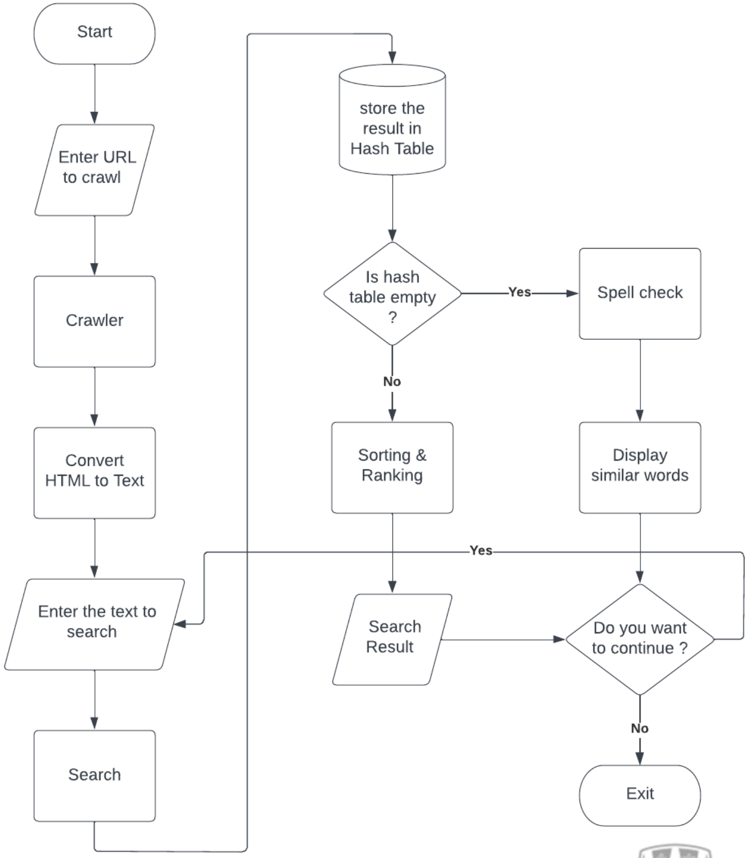
Hi

02

Workflow

200 !!FOUND

FLOW CHART



Start

Enter URL to crawl

Crawler

Convert HTML to Text

Enter the text to search

Search

store the result in Hash Table

Is hash table empty ?

Yes

No

Spell check

Sorting & Ranking

Display similar words

Search Result

Do you want to continue ?

Yes

No

Exit

University of Windsor

03

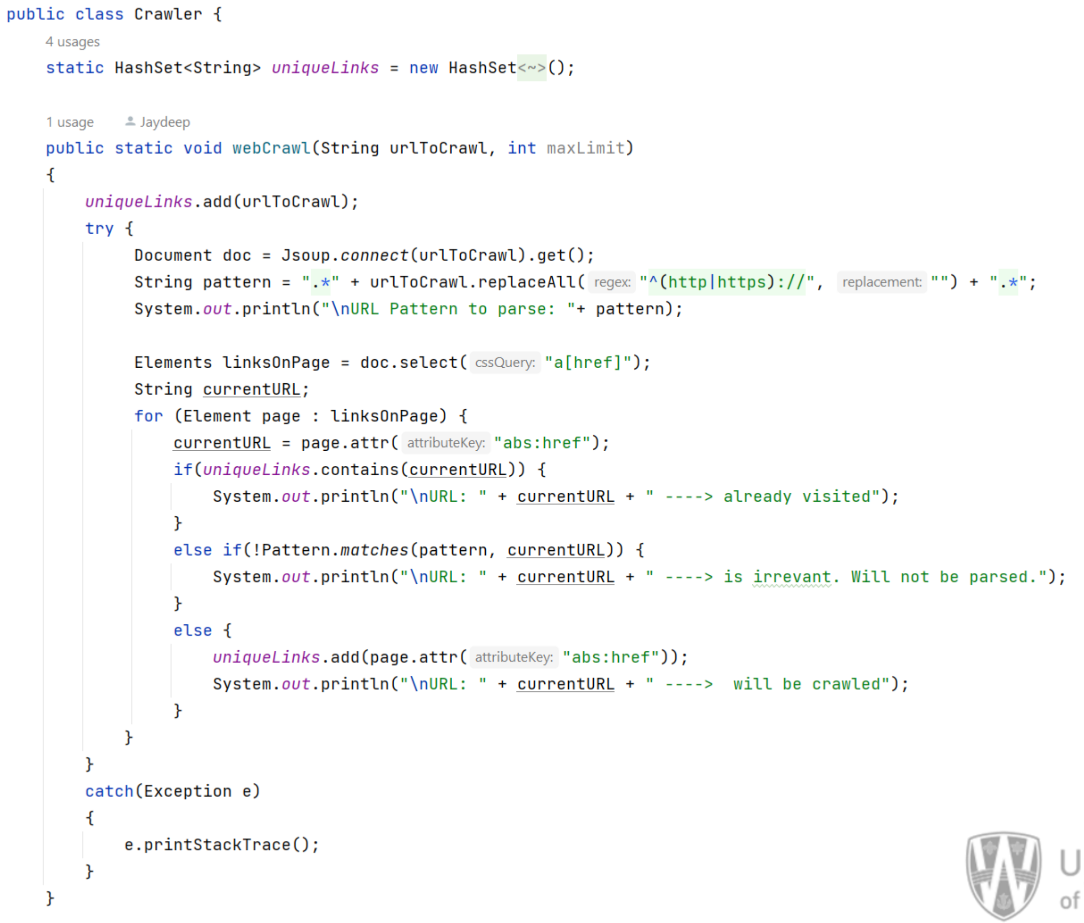
Features

Crawler

- Parsing HTML from a URL
- Library used - Jsoup
- Crawls a given webpage for all the hrefs
- Adds them to a Hashset after crawling

Crawler

Crawling n number of URLs and adding the valid URLs to the HashSet

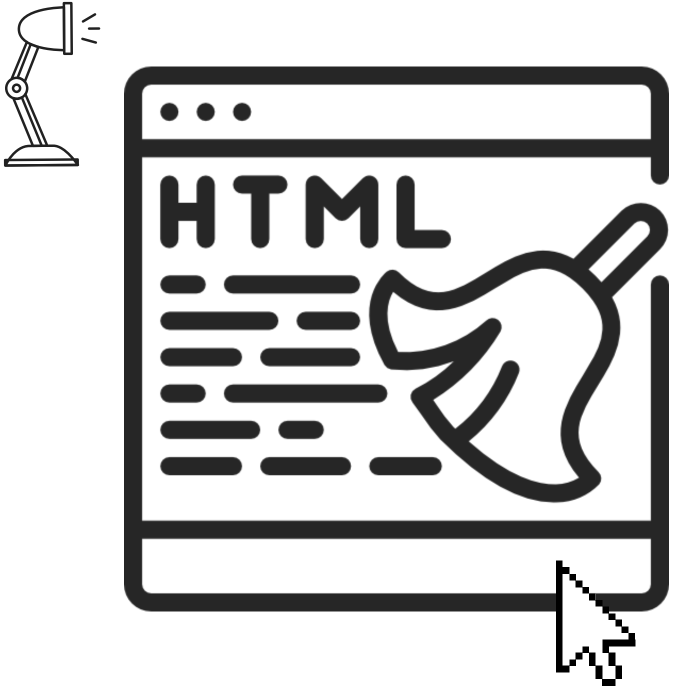


```
public class Crawler {
    4 usages
    static HashSet<String> uniqueLinks = new HashSet<~>();

    1 usage   Jaydeep
    public static void webCrawl(String urlToCrawl, int maxLimit)
    {
        uniqueLinks.add(urlToCrawl);
        try {
            Document doc = Jsoup.connect(urlToCrawl).get();
            String pattern = ".*" + urlToCrawl.replaceAll( regex: "^(http|https)://",  replacement: "") + ".*";
            System.out.println("\nURL Pattern to parse: "+ pattern);

            Elements linksOnPage = doc.select( cssQuery: "a[href]");
            String currentURL;
            for (Element page : linksOnPage) {
                currentURL = page.attr( attributeKey: "abs:href");
                if(uniqueLinks.contains(currentURL)) {
                    System.out.println("\nURL: " + currentURL + " ----> already visited");
                }
                else if(!Pattern.matches(pattern, currentURL)) {
                    System.out.println("\nURL: " + currentURL + " ----> is irrevant. Will not be parsed.");
                }
                else {
                    uniqueLinks.add(page.attr( attributeKey: "abs:href"));
                    System.out.println("\nURL: " + currentURL + " ---->  will be crawled");
                }
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
```

University of Windsor

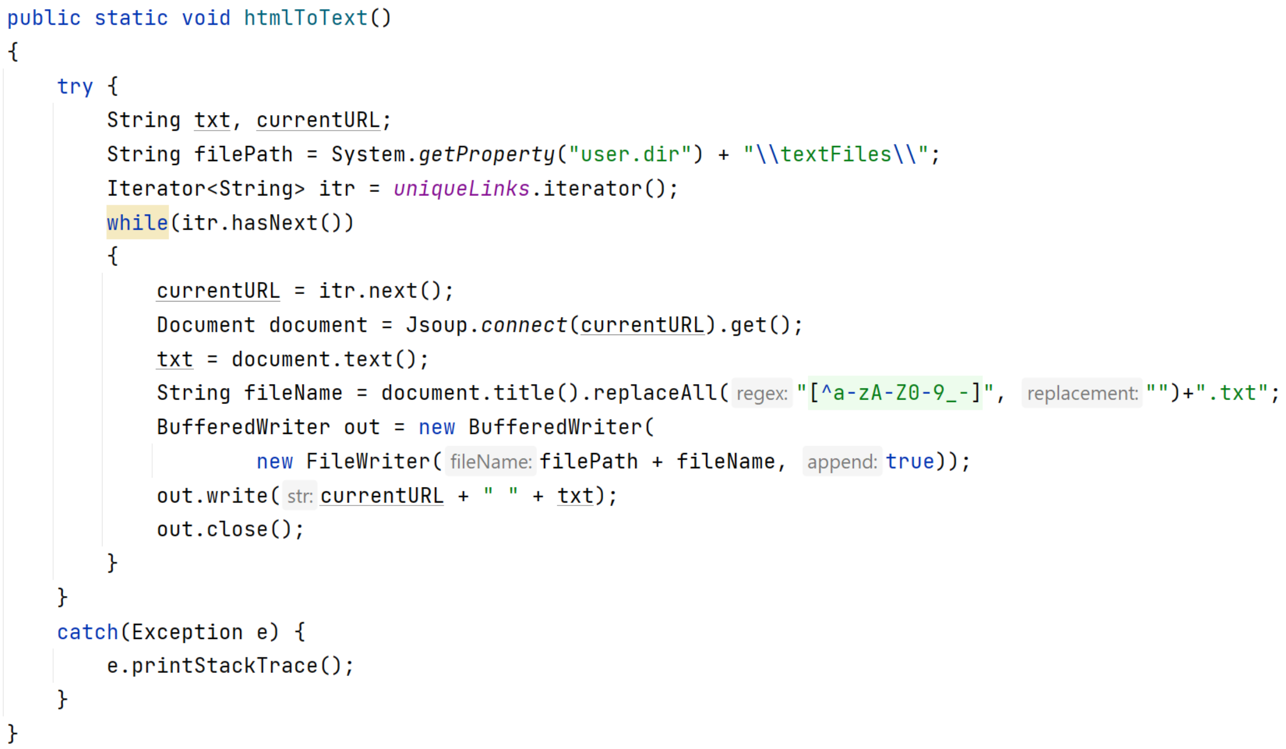
HTML to Text

- Connect to each URL from the Hashset and load the HTML page
- Convert the HTML files to txt files for easy parsing



HTML to Text

Connecting to the URLs and saving the txt file in the "textFiles" folder

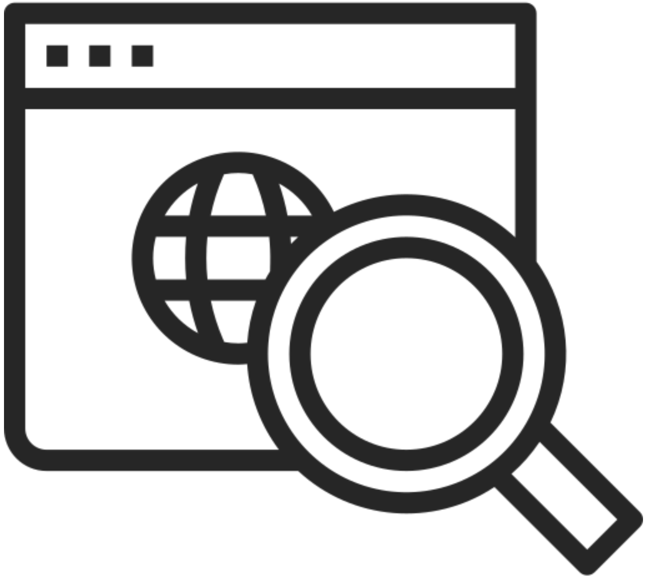


```
public static void htmlToText()
{
    try {
        String txt, currentURL;
        String filePath = System.getProperty("user.dir") + "\\textFiles\\";
        Iterator<String> itr = uniqueLinks.iterator();
        while(itr.hasNext())
        {
            currentURL = itr.next();
            Document document = Jsoup.connect(currentURL).get();
            txt = document.text();
            String fileName = document.title().replaceAll( regex: "[^a-zA-Z0-9_-]",  replacement: "")+".txt";
            BufferedWriter out = new BufferedWriter(
                    new FileWriter( fileName: filePath + fileName,  append: true));
            out.write( str: currentURL + " " + txt);
            out.close();
        }
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
```

University of Windsor

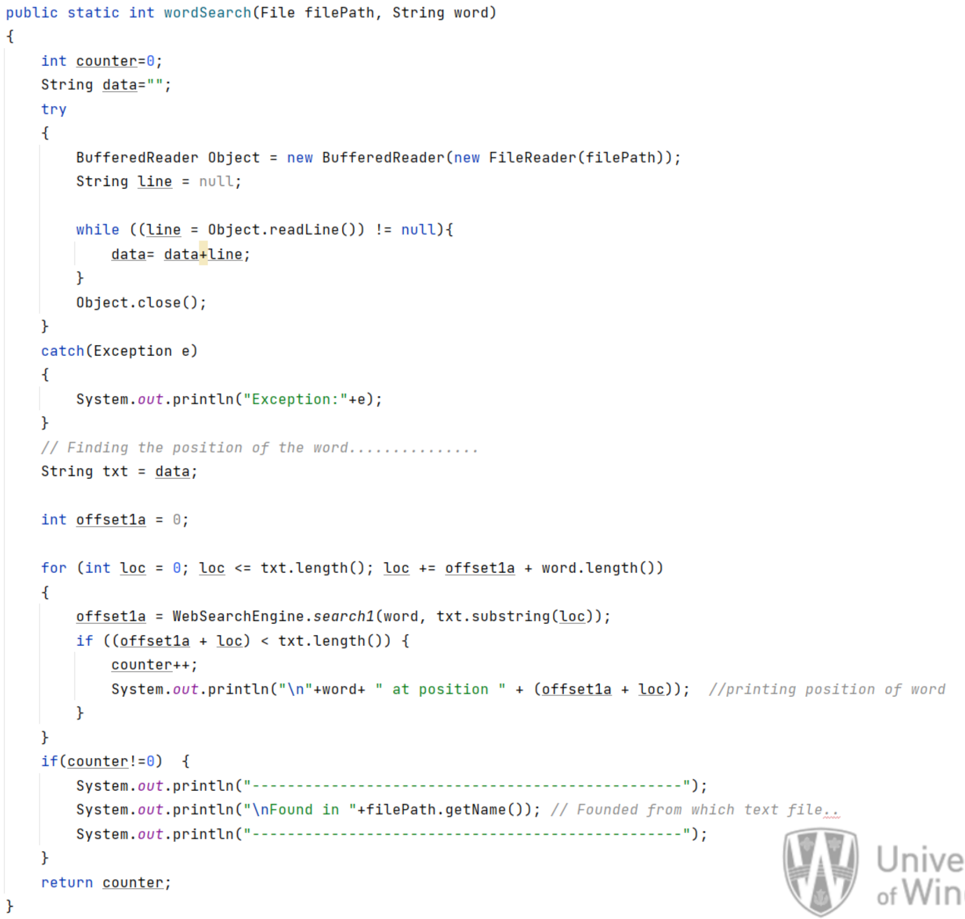
Searching

- Takes input from user to search
- Uses Boyer Moore algorithm to search from all the test files generated
- Counting instances of the text in each file and store it using hashing.



Searching

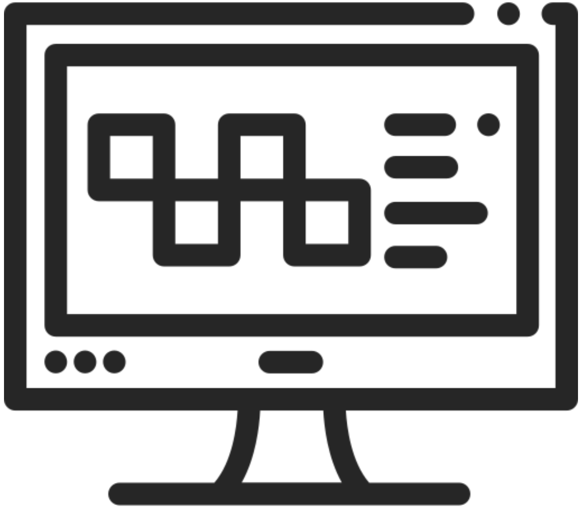
- Iterate words through all the files present.
- Consider each file as a set of characters and try to match using Boyer Moore Algorithm.



```
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 = WebSearchEngine.search1(word, txt.substring(loc));
        if ((offset1a + loc) < txt.length()) {
            counter++;
            System.out.println("\n"+word+ " at position " + (offset1a + loc));  //printing position of word
        }
    }
    if(counter!=0)  {
        System.out.println("---------------------------------------------------");
        System.out.println("\nFound in "+filePath.getName()); // Founded from which text file..
        System.out.println("---------------------------------------------------");
    }
    return counter;
}
```

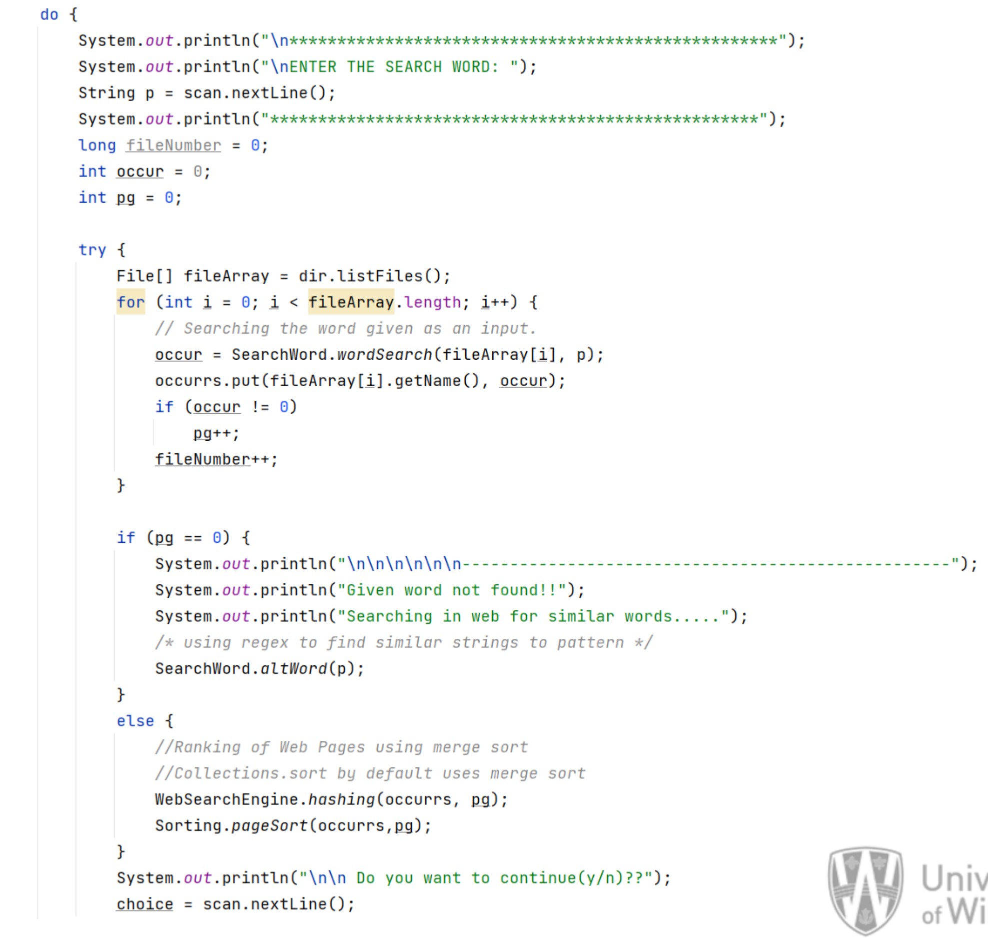
Hashing

- A hash table is used to store the results from searching.
- File names are stored as key and count of occurrences in that file are stored as value.
- Further this hash table is used for sorting and ranking.



Hash table

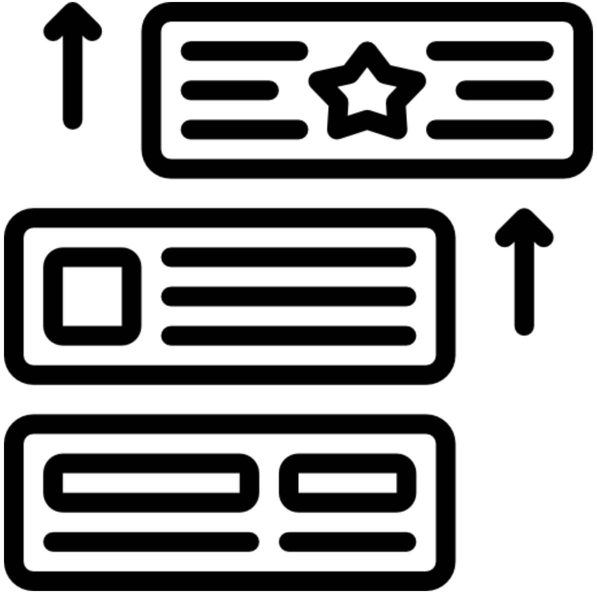
- Hash table will save all the records from search operation using separate chaining method.
- Hash table will increase its capacity once 75% of the table is filled.



```
do {
    System.out.println("\n***************************************************");
    System.out.println("\nENTER THE SEARCH WORD: ");
    String p = scan.nextLine();
    System.out.println("***************************************************");
    long fileNumber = 0;
    int occur = 0;
    int pg = 0;

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

        if (pg == 0) {
            System.out.println("\n\n\n\n\n\n---------------------------------------------------");
            System.out.println("Given word not found!!");
            System.out.println("Searching in web for similar words.....");
            /* using regex to find similar strings to pattern */
            SearchWord.altWord(p);
        }
        else {
            //Ranking of Web Pages using merge sort
            //Collections.sort by default uses merge sort
            WebSearchEngine.hashing(occurrs, pg);
            Sorting.pageSort(occurrs,pg);
        }
        System.out.println("\n\n Do you want to continue(y/n)??");
        choice = scan.nextLine();
```

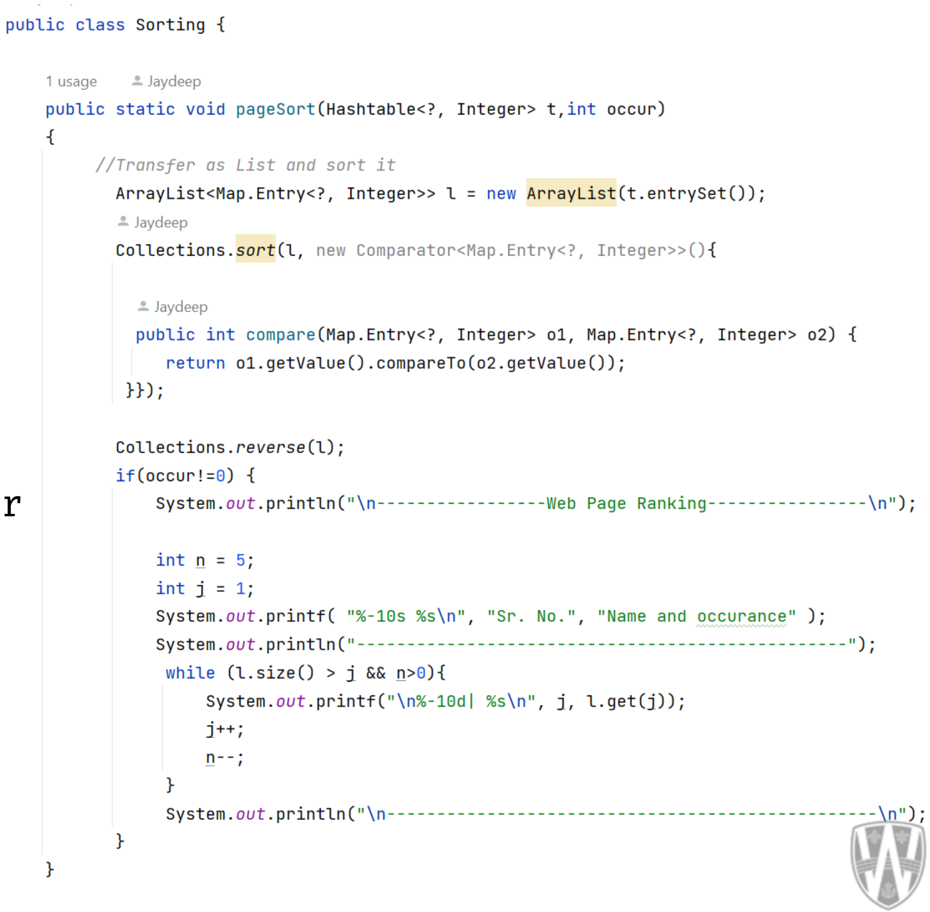
Ranking

- Ranking is the order in which the indexed results appear on the result page
- Sorting operation is performed to get the ranking of the results



Ranking - Sorting

- Sorting the hash table according to the number of occurrences.
- Displaying top 5 file names having highest value in the hash table.



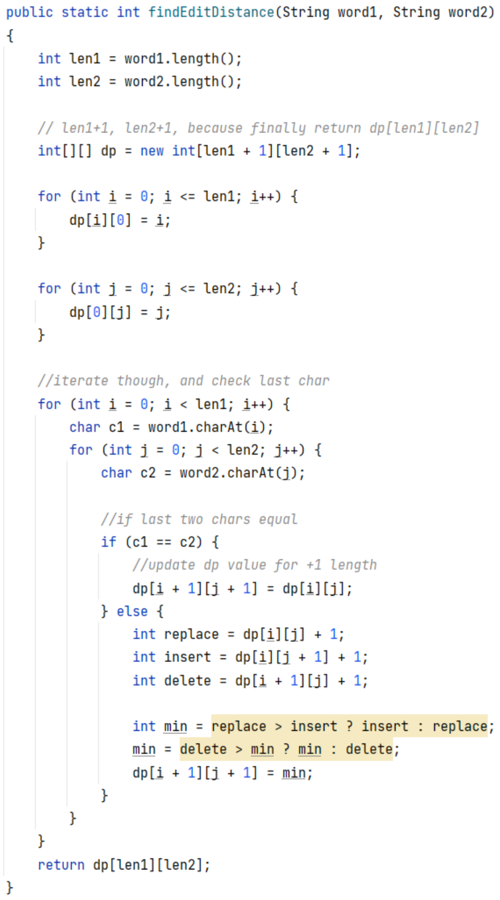
```
public class Sorting {

    1 usage   Jaydeep
    public static void pageSort(Hashtable<?, Integer> t,int occur)
    {
         //Transfer as List and sort it
           ArrayList<Map.Entry<?, Integer>> l = new ArrayList(t.entrySet());
           Jaydeep
           Collections.sort(l, new Comparator<Map.Entry<?, Integer>>(){

             Jaydeep
             public int compare(Map.Entry<?, Integer> o1, Map.Entry<?, Integer> o2) {
                return o1.getValue().compareTo(o2.getValue());
            }});

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

               int n = 5;
               int j = 1;
               System.out.printf( "%-10s %s\n", "Sr. No.", "Name and occurance" );
               System.out.println("-------------------------------------------------");
                while (l.size() > j && n>0){
                    System.out.printf("\n%-10d| %s\n", j, l.get(j));
                    j++;
                    n--;
                }
                System.out.println("\n-------------------------------------------------\n");
           }
    }
```

Edit Distance

- Edit distance is calculated by measuring how many operations are required to convert one string into the other.
- Operations :
  - Insertion
  - Deletion
  - Replacement

Edit Distance

- All text files are tokenized and stored in a list.
- For every unique string in the list the input is matched and the word with least edit distance is returned as output.



```java
public static int findEditDistance(String word1, String word2)
{
    int len1 = word1.length();
    int len2 = word2.length();

    // len1+1, len2+1, because finally return dp[len1][len2]
    int[][] dp = new int[len1 + 1][len2 + 1];

    for (int i = 0; i <= len1; i++) {
        dp[i][0] = i;
    }

    for (int j = 0; j <= len2; j++) {
        dp[0][j] = j;
    }

    //iterate though, and check last char
    for (int i = 0; i < len1; i++) {
        char c1 = word1.charAt(i);
        for (int j = 0; j < len2; j++) {
            char c2 = word2.charAt(j);

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

                int min = replace > insert ? insert : replace;
                min = delete > min ? min : delete;
                dp[i + 1][j + 1] = min;
            }
        }
    }
    return dp[len1][len2];
}
```

04

Demo

05 Our team

Abhishek Patel

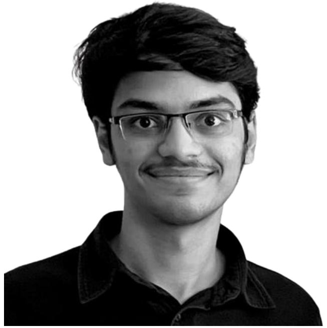
linkedin.com/in/abhishekpatelmc/

github.com/abhishekpatelmc

Jaydeep Dharamsey

linkedin.com/in/jaydeepdharamsey/

github.com/Jaydeep21



Raj Manoj Dedhia

linkedin.com/in/rajdedhia10/

github.com/rajdedhia10

University of Windsor

202 Request Accepted

Thank You

University of Windsor