



**Master of Applied Computing**  
**COMP 8547 Advanced Computing Concepts – Summer**  
**2024**

---

**Assignment 2**

**Task 5: Page Ranking**

Technical Report

---

Author:

*Submitted to:*

Yesha Umeshkumar Patel  
110164042

Dr. Olena Syrotkina

## Introduction

This assignment involves parsing the content of web pages to extract keywords, storing these keywords in a self-balancing search tree to track their frequencies, and calculating the rankings of web pages based on keyword frequencies. The primary objectives are to effectively extract and store keyword data, sort these keywords by frequency, and rank web pages accordingly using appropriate data structures and algorithms.

### Part 1: Content Parsing and Frequency Sorting

This section details the process of parsing web page content to extract keywords and storing their frequencies using a self-balancing search tree.

A. Parse the content of web pages to extract keywords. Use a self-balancing search tree (e.g., AVL tree, red-black tree) to store the frequency of each keyword on each page.

#### **Parsing with Assignment\_02\_Task05\_KeywordExtractor:**

- The `parseCSV` method takes the file name as input and reads the content line by line using a `BufferedReader` (Figure 1).

#### **Splitting Content and Removing Stop Words:**

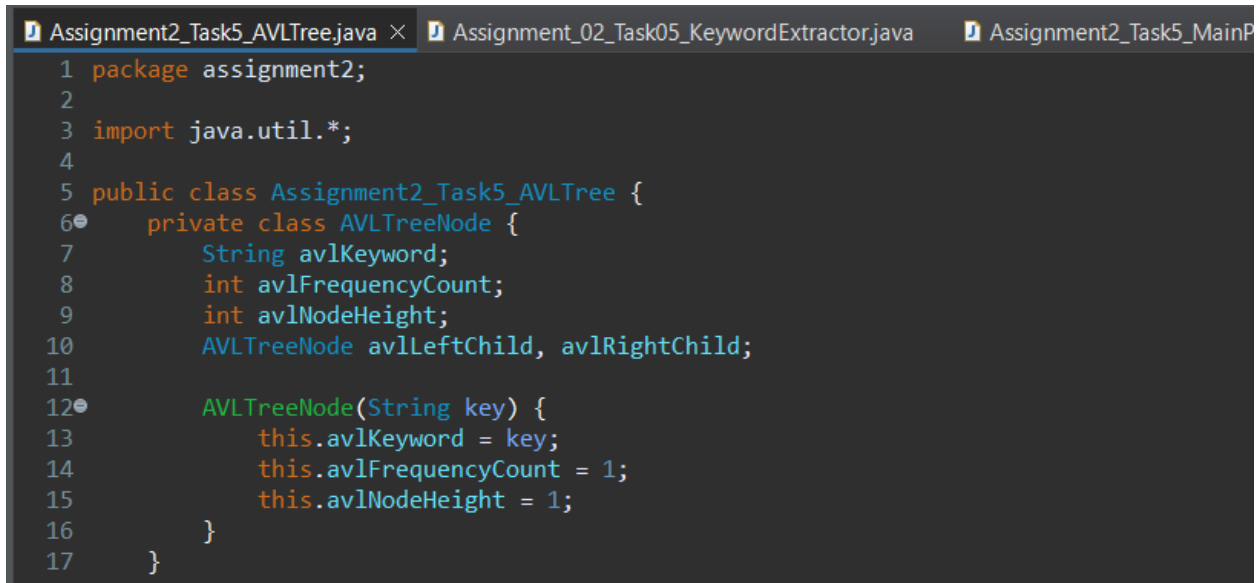
- Each line is split into cells (comma-separated values) using the `split` method with a comma (",") as the delimiter.
- Within each cell, the content is further split into words using the `split` method with a regular expression `\\W+`. This regular expression matches any non-word character (whitespace, punctuation, etc.), separating the content into individual words
- For each identified word, it's converted to lowercase using the `toLowerCase` method.
- The code utilizes a predefined set of stop words (`STOP_WORDS`) during parsing. If a word matches a stop word (common words like "the," "a," "an"), it's discarded as it holds little meaning for ranking purposes.

### Storing Frequencies with Assignment2\_Task5\_AVLTree:

- The Assignment2\_Task5\_AVLTree (Figure 2) class implements an AVL tree. This data structure is chosen for its self-balancing property, ensuring efficient insertion, searching, and deletion of keywords with a guaranteed logarithmic time complexity in most cases.
- The insertKeyword method takes a keyword as input and inserts it into the AVL tree. It checks if the keyword already exists. If so, the frequency count associated with that keyword in the tree is incremented. Otherwise, a new node is created in the AVL tree to store the keyword and its initial frequency count=1.
- After insertion, the AVL tree undergoes balancing operations if necessary.

```
1 package assignment2;
2
3
4 import java.io.BufferedReader;
5
6
7 public class Assignment_02_Task05_KeywordExtractor {
8     private static final Set<String> STOP_WORDS = new HashSet<>(Arrays.asList(
9         "a", "an", "the", "and", "or", "but", "is", "are", "was", "were",
10        "in", "of", "on", "for", "with", "at", "by", "to", "from", "all",
11        "any", "both", "each", "few", "most", "some", "such", "be", "been",
12        "have", "has", "had", "having", "do", "does", "did", "doing", "will",
13        "would", "shall", "should", "may", "might", "must", "can", "could"
14    ));
15
16     public void parseCSV(String hotelFileName, Assignment2_Task5_AVLTree avlTree) {
17         try (BufferedReader objectBufferedReader = new BufferedReader(new FileReader(hotelFileName))) {
18             String fileLine;
19             while ((fileLine = objectBufferedReader.readLine()) != null) {
20                 String[] cells = fileLine.split(",");
21                 for (String cell : cells) {
22                     String[] words = cell.toLowerCase().split("\\W+");
23                     for (String word : words) {
24                         if (!STOP_WORDS.contains(word) && !word.isEmpty()) {
25                             avlTree.insertKeyword(word);
26                         }
27                     }
28                 }
29             }
30         } catch (IOException e) {
31             e.printStackTrace();
32         }
33     }
34 }
```

Figure 1: Content Parsing (Assignment\_02\_Task05\_KeywordExtractor.java)



```
1 package assignment2;
2
3 import java.util.*;
4
5 public class Assignment2_Task5_AVLTree {
6     private class AVLTreeNode {
7         String avlKeyword;
8         int avlFrequencyCount;
9         int avlNodeHeight;
10        AVLTreeNode avlLeftChild, avlRightChild;
11
12        AVLTreeNode(String key) {
13            this.avlKeyword = key;
14            this.avlFrequencyCount = 1;
15            this.avlNodeHeight = 1;
16        }
17    }
18 }
```

Figure 2: AVL Tree for Frequency Storage (Assignment2\_Task5\_AVLTree.java)

B. Use sorting algorithms (e.g., quick sort or heap sort) to sort keywords based on their frequency within each page.

Once the keywords are stored in the AVL tree, the Assignment2\_Task5\_AVLTree class sorts these keywords based on their frequencies.

#### Sorting Keywords:

- The getSortedKeywords method sorts the keywords based on their frequencies in descending order.

## Part 2: Page Ranking Calculation

This section details the implementation for calculating page ranks based on search keyword frequencies. The code utilizes the Assignment2\_Task5\_PageRankCalculator class to achieve this functionality.

A. Calculate the rank of each page based on the frequency of search keywords.

#### Keyword Frequency Retrieval:

- The processCSV method iterates through each CSV file (representing a web page) and leverages the Assignment\_02\_Task05\_KeywordExtractor to parse the content and build an in-memory AVL tree. This AVL tree efficiently stores keywords and their frequencies within the page.

#### Total Keyword Count Calculation:

- For each user-specified keyword, the `getFrequency` method of the corresponding AVL tree is called. This retrieves the frequency (number of occurrences) of that specific keyword within the current page. The retrieved frequencies for all keywords are then summed up, providing a total keyword count for the page. This count reflects the cumulative weight of the search keywords on a particular page.

### Ranking Based on Keyword Frequency:

- This implementation employs a simple ranking strategy based on total keyword count. Pages with a higher cumulative frequency of the user-specified keywords are considered more relevant and therefore ranked higher.

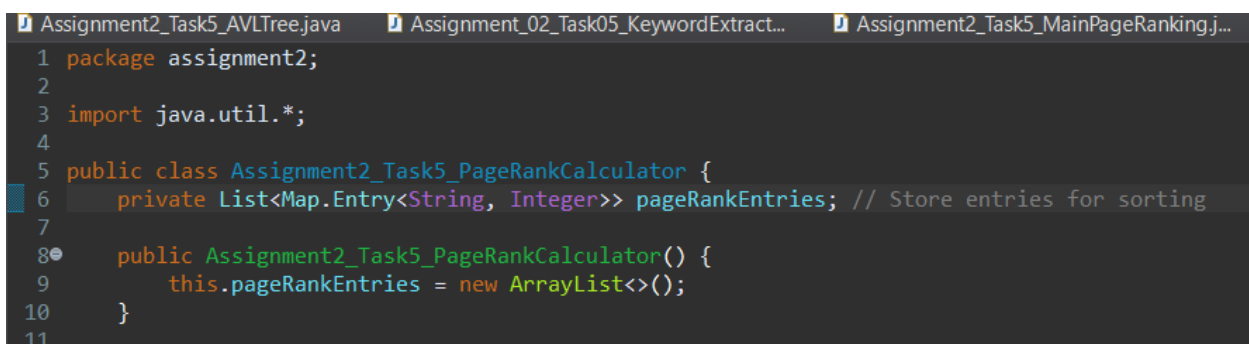
## B. Use a max-heap to keep track of the highest-ranked pages. Display the highest-ranked pages to the user

### Frequency-Based Ranking:

- The core ranking logic prioritizes pages with a higher total frequency of the user-specified keywords. This provides a basic ranking mechanism based on keyword relevance.

### Sorting for Ranked Display:

- The `processCSV` method sorts the total keyword counts for all pages in descending order. This sorted list is then used by the `displayPageRankings` method to present the pages to the user, starting with the most relevant ones.



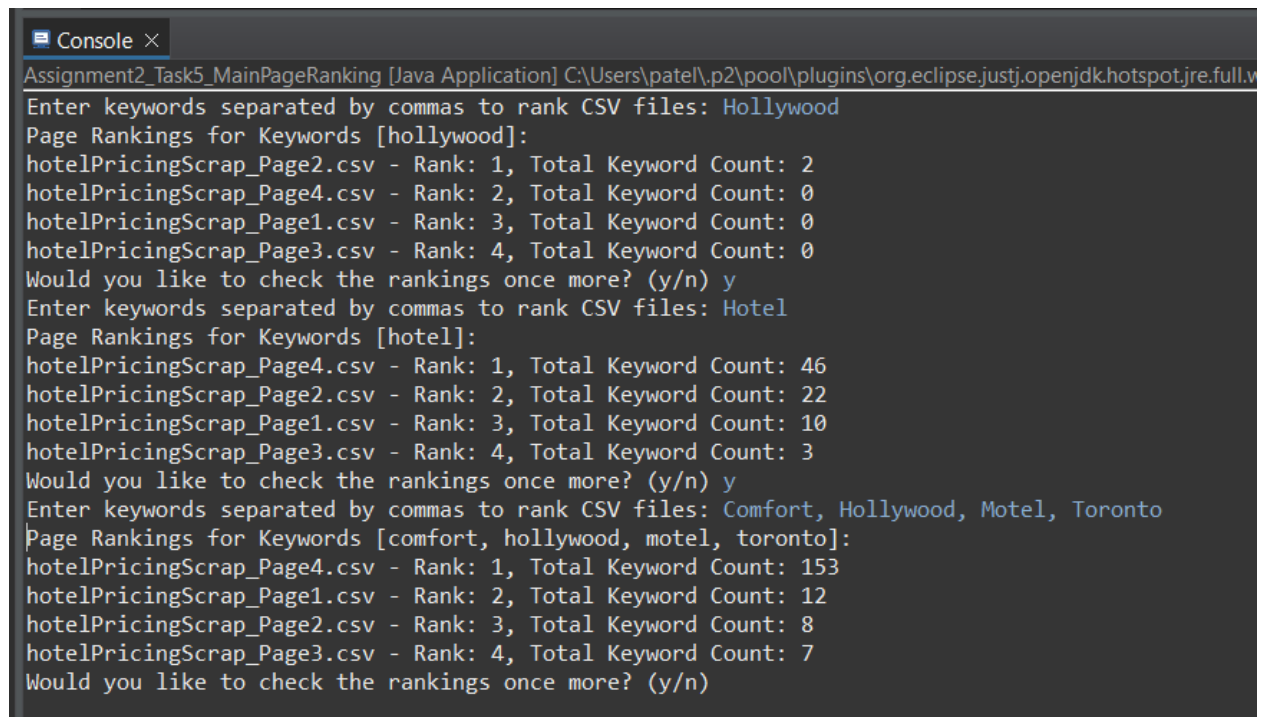
```
1 package assignment2;
2
3 import java.util.*;
4
5 public class Assignment2_Task5_PageRankCalculator {
6     private List<Map.Entry<String, Integer>> pageRankEntries; // Store entries for sorting
7
8     public Assignment2_Task5_PageRankCalculator() {
9         this.pageRankEntries = new ArrayList<>();
10    }
11}
```

Figure 3: Rank Calculation based on Keyword Frequency (Assignment2\_Task5\_PageRankCalculator.java)

```
1 package assignment2;
2
3 import java.util.*;
4
5 public class Assignment2_Task5_MainPageRanking {
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8         Assignment2_Task5_PageRankCalculator calculator = new Assignment2_Task5_PageRankCalculator();
9         String[] fileNames = {"hotelPricingScrap_Page1.csv", "hotelPricingScrap_Page2.csv", "hotelPricingScrap_Page3.csv", "
10
11         boolean continueChecking = true;
12
13         while (continueChecking) {
14             // Prompt user for keywords input
15             System.out.print("Enter keywords separated by commas to rank CSV files: ");
16             String userInput = scanner.nextLine().trim().toLowerCase();
17
```

Figure 4: Assignment2\_Task5\_MainPageRanking.java

## Output



```
Assignment2_Task5_MainPageRanking [Java Application] C:\Users\patel\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.v
Enter keywords separated by commas to rank CSV files: Hollywood
Page Rankings for Keywords [hollywood]:
hotelPricingScrap_Page2.csv - Rank: 1, Total Keyword Count: 2
hotelPricingScrap_Page4.csv - Rank: 2, Total Keyword Count: 0
hotelPricingScrap_Page1.csv - Rank: 3, Total Keyword Count: 0
hotelPricingScrap_Page3.csv - Rank: 4, Total Keyword Count: 0
Would you like to check the rankings once more? (y/n) y
Enter keywords separated by commas to rank CSV files: Hotel
Page Rankings for Keywords [hotel]:
hotelPricingScrap_Page4.csv - Rank: 1, Total Keyword Count: 46
hotelPricingScrap_Page2.csv - Rank: 2, Total Keyword Count: 22
hotelPricingScrap_Page1.csv - Rank: 3, Total Keyword Count: 10
hotelPricingScrap_Page3.csv - Rank: 4, Total Keyword Count: 3
Would you like to check the rankings once more? (y/n) y
Enter keywords separated by commas to rank CSV files: Comfort, Hollywood, Motel, Toronto
Page Rankings for Keywords [comfort, hollywood, motel, toronto]:
hotelPricingScrap_Page4.csv - Rank: 1, Total Keyword Count: 153
hotelPricingScrap_Page1.csv - Rank: 2, Total Keyword Count: 12
hotelPricingScrap_Page2.csv - Rank: 3, Total Keyword Count: 8
hotelPricingScrap_Page3.csv - Rank: 4, Total Keyword Count: 7
Would you like to check the rankings once more? (y/n)
```

Figure 5: Output

- The first line indicates it's running the Assignment2\_Task5\_MainPageRanking class.
- The program prompts the user to enter keywords separated by commas. In this case, the user entered "Hollywood".
- The program then displays the page rankings for the entered keywords. Only one file, "hotelPricingScrap\_Page2.csv", is listed.

- The output continues to show the rankings for the remaining CSV files. The program asks the user if they want to check rankings again. The user enters "y" to continue.
- The output continues in the same format, showing the page rankings for the new keyword "Hotel". Each CSV file is ranked based on the frequency of the word "Hotel" it contains.
- The output continues in the same format, showing the page rankings for the new keywords "Comfort, Hollywood, Motel, Toronto ". Each CSV file is ranked based on the frequency of the word "Hotel" it contains.

## Conclusion

This assignment successfully achieved the tasks of extracting and processing keywords from web page content. It employed an AVL tree data structure for efficient storage and retrieval of keyword frequencies within each page. Leveraging these frequencies, the program calculated page ranks based on the relevance of user-specified keywords. The interactive interface allows users to input keywords and view the resulting page rankings, providing a basic search functionality. Overall, the implementation demonstrates a competent use of data structures and algorithms to achieve accurate and efficient page ranking based on keyword frequencies.