THE OHIO STATE UNIVERSITY

PROJECT 9: Tag Cloud Generator

Daniil Gofman

Ansh Pachauri

SW 2: Dev & Dsgn

Paolo Bucci

Yiyang Chen

Shivam Gupta

November 28, 2023

```
 1 import java.util.Comparator;
 2
 3 import components.map.Map;
 4 import components.map.Map1L;
 5 import components.set.Set;
 6 import components.set.Set1L;
 7 import components.simplereader.SimpleReader;
 8 import components.simplereader.SimpleReader1L;
 9 import components.simplewriter.SimpleWriter;
10 import components.simplewriter.SimpleWriter1L;
11 import components.sortingmachine.SortingMachine;
12 import components.sortingmachine.SortingMachine1L;
13 import components.utilities.FormatChecker;
14
15 /**
16  * Program to take a file of text, count each instance of every
   word, and
17  * generate a tag cloud with each word and corresponding size.
18  *
19  * @author Daniil Gofman
20  * @author Ansh Pachauri
21  *
22  */
23 public final class TagCloudGenerator {
24
25     /**
26      * No argument constructor--private to prevent
   instantiation.
27      */
28     private TagCloudGenerator() {
29     }
30
31     /**
32      * The maximum font size.
33      */
34     private static final int FONT_NUMBER = 37;
35     /**
36      * The maximum font size.
37      */
```

```
38     private static final int MIN_FONT_SIZE = 11;
39
40     /**
41      * Compare {@code String}s in alphabetical order.
42      */
43     private static class StringLT
44             implements Comparator<Map.Pair<String, Integer>> {
45         @Override
46         public int compare(Map.Pair<String, Integer> str1,
47                 Map.Pair<String, Integer> str2) {
48             //lower case and compare alphabetically
49             return
   str1.key().toLowerCase().compareTo(str2.key().toLowerCase());
50         }
51     }
52
53     /**
54      * Compare {@code Integer}i in decreasing order.
55      */
56     private static class IntegerLT
57             implements Comparator<Map.Pair<String, Integer>> {
58         @Override
59         public int compare(Map.Pair<String, Integer> int1,
60                 Map.Pair<String, Integer> int2) {
61             return int2.value().compareTo(int1.value());
62         }
63     }
64
65     /**
66      * Returns the first "word" (maximal length string of
   characters not in
67      * {@code separators}) or "separator string" (maximal
   length string of
68      * characters in {@code separators}) in the given {@code
   text} starting at
69      * the given {@code position}.
70      *
71      * @param text
72      *            the {@code String} from which to get the word
```

```
   or separator
 73     *            string
 74     * @param position
 75     *            the starting index
 76     * @param separators
 77     *            the {@code Set} of separator characters
 78     * @return the first word or separator string found in
   {@code text} starting
 79     *         * at index {@code position}
 80     * @requires 0 <= position < |text|
 81     * @ensures <pre>
 82     * nextWordOrSeparator =
 83     * text[position, position + |nextWordOrSeparator|) and
 84     * if entries(text[position, position + 1)) intersection
   separators = {} * then
 85     * entries(nextWordOrSeparator) intersection separators =
   {} and
 86     * (position + |nextWordOrSeparator| = |text| or
 87     * entries(text[position, position + |nextWordOrSeparator|
   + 1))
 88     * intersection separators /= {})
 89     * else
 90     * entries(nextWordOrSeparator) is subset of separators and
 91     * (position + |nextWordOrSeparator| = |text| or
 92     * entries(text[position, position + |nextWordOrSeparator|
   + 1))
 93     * is not subset of separators)
 94     * </pre>
 95     */
 96    private static String nextWordOrSeparator(String text, int
   position,
 97            Set<Character> separators) {
 98
 99        int end = position;
100        if (!separators.contains(text.charAt(position))) {
101            //find length of word
102            while (end < text.length()
103                    && !separators.contains(text.charAt(end)))
   {
```

```
104                 end++;
105             }
106         } else {
107             //find length of separator
108             while (end < text.length()
109                     && separators.contains(text.charAt(end))) {
110                 end++;
111             }
112         }
113         return text.substring(position, end);
114     }
115 
116     /**
117      * Generates the set of characters in the given {@code
    String} into the
118      * given {@code Set}.
119      *
120      * @param str
121      *            the given {@code String}
122      * @param strSet
123      *            the {@code Set} to be replaced
124      * @replaces strSet
125      * @ensures strSet = entries(str)
126      */
127     private static void generateElements(String str,
    Set<Character> strSet) {
128         assert str != null : "Violations of: str is not null";
129         assert strSet != null : "Violation of: strSet is not
    null";
130 
131         for (int i = 0; i < str.length(); i++) {
132             char c = str.charAt(i);
133             if (!strSet.contains(c)) {
134                 strSet.add(c);
135             }
136         }
137     }
138 
139     /**
```

```
140     * Takes the file and makes a map with each key as the word
    in lower case
141     * and the value as the number of occurrences of that word.
142     *
143     * @param inFile
144     *            the SimpleReader file
145     * @ensures all words from the file will be in the map with
    the count of
146     *          each word
147     * @return Map<String, Integer> of words of the file and
    their counts
148     */
149    private static Map<String, Integer> fileToMap(SimpleReader
    inFile) {
150        assert inFile != null : "Violation of: inFile is not
    null";
151
152        Map<String, Integer> result = new Map1L<>();
153        // define separators for the words
154        final String separators = "'., ()-_?\"/!@#$
    %^&*\t1234567890:"
155                + ";[]{}+=~`><";
156        Set<Character> separatorSet = new Set1L<>();
157        generateElements(separators, separatorSet);
158
159        // read file, separate words and compile all words into
    the map
160        while (!inFile.atEOS()) {
161            String line = inFile.nextLine();
162            int i = 0;
163            while (i < line.length()) {
164                String word = nextWordOrSeparator(line, i,
    separatorSet)
165                        .toLowerCase();
166                boolean isWord = true;
167                for (int j = 0; j < word.length(); j++) {
168                    char c = word.charAt(j);
169                    if (separatorSet.contains(c)) {
170                        isWord = false;
```

```
171                     }
172                 }
173                 if (isWord) {
174                     //add word to map or increase the count
175                     if (result.hasKey(word)) {
176                         int count = result.value(word);
177                         result.replaceValue(word, count + 1);
178                     } else {
179                         result.add(word, 1);
180                     }
181                 }
182                 i += word.length();
183             }
184         }
185         return result;
186     }
187 
188     /**
189      * Takes a map and first sorts the it with sortingMachine
    by occurrences of
190      * each word in decreasing order. Then makes a second
    sortingMachine, and
191      * sorts the map alphabetically. numDisplay is the number
    of words to be put
192      * in first sortingMachine.
193      *
194      * @param map
195      *            map of all of the words and their counts
196      * @param numDisplay
197      *            the amount of words that will be in the first
    sortingMachine
198      * @requires map is not null
199      * @ensures all Map.Pairs is sorted in decreasing order by
    their values
200      * @return SortingMachine<Map.Pair<String, Integer>> of
    words and their
201      *         counts in alphabetical order
202      */
203     public static SortingMachine<Map.Pair<String, Integer>>
```

```
    mapToSMAlpha(
204             Map<String, Integer> map, Integer numDisplay) {
205         assert map != null : "Violation of: words is not null";
206 
207         Comparator<Map.Pair<String, Integer>> countSort = new
    IntegerLT();
208         Comparator<Map.Pair<String, Integer>> alphaSort = new
    StringLT();
209         SortingMachine<Map.Pair<String, Integer>> countSM = new
    SortingMachine1L<>(
210                 countSort);
211         SortingMachine<Map.Pair<String, Integer>> alphaSM = new
    SortingMachine1L<>(
212                 alphaSort);
213 
214         //make the sorting machine with the map
215         while (map.size() > 0) {
216             Map.Pair<String, Integer> temp = map.removeAny();
217             countSM.add(temp);
218         }
219         countSM.changeToExtractionMode();
220 
221         //make the sorting machine in alphabetical order
222         for (int i = 0; i < numDisplay; i++) {
223             if (countSM.size() != 0) {
224                 alphaSM.add(countSM.removeFirst());
225             }
226         }
227         alphaSM.changeToExtractionMode();
228 
229         return alphaSM;
230 
231     }
232 
233     /**
234      * Output the header of the file.
235      *
236      * @param out
237      *            the output file.
```

```
238     *
239     * @param inFile
240     *            the input file.
241     *
242     * @param numWords
243     *            number of words in the HTML file.
244     */
245    private static void header(SimpleWriter out, String inFile,
    int numWords) {
246        assert out != null : "Violation of: out is not null";
247        assert out.isOpen() : "Violation of: out.is_open";
248        assert inFile != null : "Violation of: inFile is not
    null";
249        /*
250         * Output the index header HTML text.
251         */
252        out.println("<html>");
253        out.println("<head>");
254        out.println(
255                "<title>Top " + numWords + " words in " +
    inFile + "</title>");
256        out.println("<link href=\"http://web.cse.ohio-
    state.edu/software"
257                + "/2231/web-sw2/assignments/projects/tag-
    cloud-generator/data/"
258                + "tagcloud.css\" rel=\"stylesheet\"
    type=\"text/css\">");
259        out.println(
260                "<link href=\"tagcloud.css\" rel=\"stylesheet\"
    type=\"text/css\">");
261        out.println("</head>");
262        out.println("<body>");
263        out.println("<h2>Top " + numWords + " words in " +
    inFile + "</h2>");
264        out.println("<hr>");
265        out.println("<div class = \"cdiv\">");
266        out.println("<p class = \"cbox\">");
267    }
268
```

```
269     /**
270      * Output the ending tags in the generated HTML file.
271      *
272      * @param output
273      *            the output file.
274      */
275     private static void footer(SimpleWriter output) {
276         assert output != null : "Violation of: out is not
    null";
277         assert output.isOpen() : "Violation of: out.is_open";
278         //output the closing tags
279         output.println("</p>");
280         output.println("</div>");
281         output.println("</body>");
282         output.println("</html>");
283     }
284
285     /**
286      * Method to generate the main body of 'tag cloud'.
287      *
288      * @param output
289      * @param alphaSort
290      */
291     private static void generateList(SimpleWriter output,
292             SortingMachine<Map.Pair<String, Integer>>
    alphaSort) {
293         int countMax = 0;
294         int countMin = 100;
295         for (Map.Pair<String, Integer> i : alphaSort) {
296             if (i.value() > countMax) {
297                 countMax = i.value();
298             }
299             if (i.value() < countMin) {
300                 countMin = i.value();
301             }
302         }
303         int difference = countMax - countMin;
304         int interDifference = difference / FONT_NUMBER;
305         if (interDifference == 0) {
```

```
306             interDifference = 1;
307         }
308 
309         while (alphaSort.size() > 0) {
310             Map.Pair<String, Integer> temp =
    alphaSort.removeFirst();
311             int font = ((temp.value() - countMin) /
    interDifference)
312                     + MIN_FONT_SIZE;
313             output.println("<span style=\"cursor:default\"
    class=\"f" + font
314                     + "\"title = \"count:" + temp.value() +
    "\">" + temp.key()
315                     + "</span>");
316         }
317     }
318 
319     /**
320      * Main method.
321      *
322      * @param args
323      *             the command line arguments
324      */
325     public static void main(String[] args) {
326         SimpleReader in = new SimpleReader1L();
327         SimpleWriter out = new SimpleWriter1L();
328         out.print("Enter the name of the input text file: ");
329         String inputFile = in.nextLine();
330         SimpleReader input = new SimpleReader1L(inputFile);
331 
332         out.print("Enter the name of the output file: ");
333         String outputFile = in.nextLine();
334         SimpleWriter output = new SimpleWriter1L(outputFile);
335 
336         out.print(
337                 "Enter number of words to be included in the
    generated tag cloud: ");
338         String wordNumStr = in.nextLine();
339         while (!FormatChecker.canParseInt(wordNumStr)
```

```
340                 || !(Integer.parseInt(wordNumStr) > 0)) {
341             out.print("ERROR: Not Positive Integer \nPlease
    enter the "
342                     + "number of words that will be displayed
    (integer): ");
343             wordNumStr = in.nextLine();
344         }
345         int wordNum = Integer.parseInt(wordNumStr);
346 
347         // Output header of a html-file
348         header(output, inputFile, wordNum);
349         // Output body of a html-file
350         Map<String, Integer> map = fileToMap(input);
351 
352         SortingMachine<Map.Pair<String, Integer>> alphaSort =
    mapToSMAlpha(map,
353                 wordNum);
354 
355         generateList(output, alphaSort);
356         // Output footer of a html-file
357         footer(output);
358         out.println("Program completed");
359 
360         /*
361          * Close input and output streams
362          */
363         input.close();
364         output.close();
365         in.close();
366         out.close();
367     }
368 
369 }
370 
```