

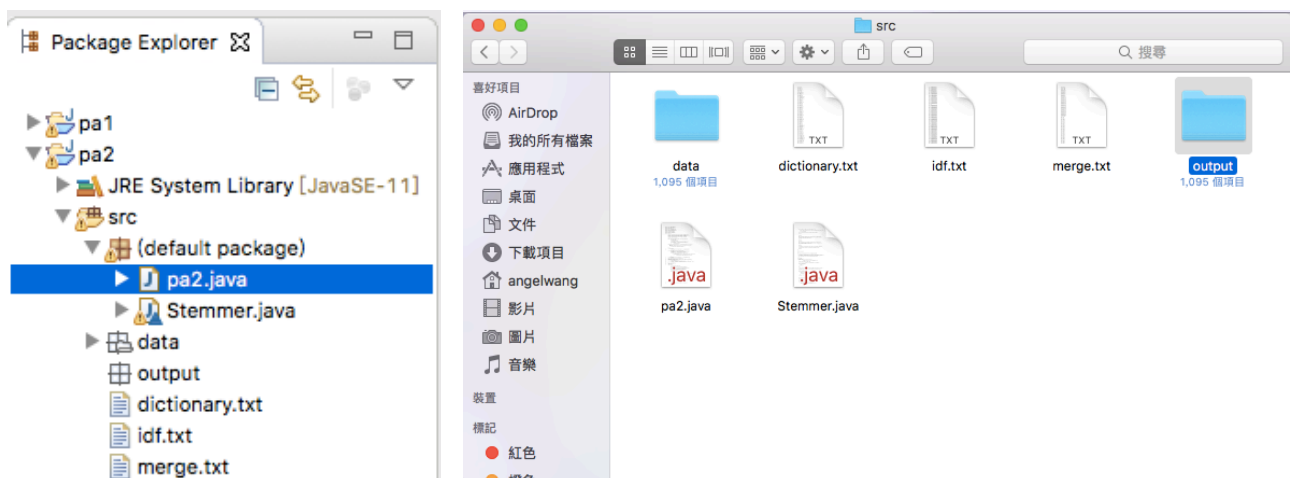
1. 執行環境：Eclipse 2019-06

2. 程式語言：Java (version: 1.8.0_101)

3. 執行方式：（* 有提供助教整個 pa2 java project）

- (1) 新增名為 pa2 的 java project，在 src 下建立新的 package 及 pa2 class (pa2.java)
- (2) 在 src 之下新增名為“data”的資料夾，將下載的 1095 個文件複製進去，供後續讀取檔案用
- (3) 在 src 之下新增名為“output”的資料夾，供第2題輸出各篇文章的 t_index 及 tf-idf unit vector 用
- (4) 在 default package 新增一個名為“Stemmer”的 class，貼入 Porter's Stemmer 的 code（來源：<https://tartarus.org/martin/PorterStemmer/java.txt>）
- (5) 為了讀取及寫入 txt 檔案，有 import java.io 的部分套件（同 PA1）
- (6) 執行 pa2.java 後會在 src 下建立並寫入 dictionary.txt，截圖中的 idf.txt 及 merge.txt 只是在寫作業時檢查用，已將 code 註解掉
- (7) 執行 pa2.java 後會在 output 的資料夾中建立 1095 個 txt 檔，並將 output 結果寫入（如右下圖所示），輸出檔案至 output 可能會跑比較久，還請助教耐心等待！

* 因為 pa2.java 要在 project 中執行，所以 data 和 output 的路徑設定為 src/data 及 src/output



4. 作業處理邏輯說明：

(1) Construct a dictionary based on the terms extracted from the given documents.

- a. 取出各篇文章的 tokens 並做前處理，建立名為 tokensArrayList 的 ArrayList<ArrayList<String>> 來儲存所有文章的 terms 及對應的 docID

各篇文章的前處理另外寫了一個 docTokens(int docID) 的 method，做法和 PA1 相同，只有更新了自定義的 stopwords list。

Code
<pre>public static void docTokens(int docID) throws Exception { ... }</pre>

用 for 迴圈將第1到第1095個文件帶入 docTokens(int docID) 的 method，將所有文章的 terms 及對應的 docID 都存到 tokensArrayList 中。

Code
<pre>18 public static void main(String[] args) throws Exception { 19 docSize = 1095; 20 /* 1. Dictionary */ 21 // (1) terms & docID 22 tokensArrayList = new ArrayList<ArrayList<String>>(); 23 for (int docID = 1; docID <= docSize; docID++) { 24 docTokens(docID); 25 } }</pre>

- b. import java.util.Collections 及 java.util.Comparator 套件來覆寫 sorting 方式，將 tokensArrayList 依 terms 的字母做排序

Code
<pre>26 Collections.sort(tokensArrayList, new Comparator<ArrayList<String>>() { 27 @Override 28 public int compare(ArrayList<String> o1, ArrayList<String> o2) { 29 return o1.get(0).compareTo(o2.get(0)); 30 } 31 });</pre>

- c. 同篇文章重複的 term 做合併

重新建立一個名為 mergedTokensList 的 ArrayList<ArrayList<String>>，用 for 迴圈將 tokensArrayList 每一個 element 取出來和前一個做比較，若兩者 term 和 docID 都不相同，就將這個 element 加進 mergedTokensList（移除同篇文章重複 term 的概念）。

Code
<pre>35 // (2) merge same term from the same document 36 mergedTokensList = new ArrayList<ArrayList<String>>(); 37 mergedTokensList.add(tokensArrayList.get(0)); 38 for(int i = 1; i < tokensArrayList.size(); i++) { 39 int pre = i-1; 40 if(!tokensArrayList.get(i).equals(tokensArrayList.get(pre))) { 41 mergedTokensList.add(tokensArrayList.get(i)); 42 } 43 }</pre>

d. 建立 document，記錄 t_index、term 和 df

重新建立一個名為 `irtm_dict` 的 `ArrayList<ArrayList<String>>` 來儲存 document 的內容。用 for 迴圈將 `mergedTokensList` 每一個 element 取出來和前一個做比較，如果該 element 的 term 和前一個不相同，就將該 term 加入 `irtm_dict`，並給予 `df=1`；如果 element 的 term 和前一個相同，表示同個 term 出現在不同文章，則 `df` 要加 1（用自建的變數 `count` 來計算），更新 `irtm_dict` 中該 term 的 `df`（因為 `mergedTokensList` 是有按照字母順序 sort 過的，所以同個字會排在一起，且 `irtm_dict` 中的 element 是一個一個加進去，該 term 在 `irtm_dict` 中的 index 會是當下的最後一個）。`t_index` 有另外建一個變數來記錄。

Code

```

57 // (3) t_index & term & df
58 irtm_dict = new ArrayList<ArrayList<String>>();
59 int t_index = 1;
60 ArrayList<String> firstEle = new ArrayList<String>();
61 firstEle.add(Integer.toString(t_index));
62 firstEle.add(mergedTokensList.get(0).get(0));
63 firstEle.add("1");
64 irtm_dict.add(firstEle);
65 for(int i = 1; i < mergedTokensList.size(); i++) {
66     int pre = i-1;
67     ArrayList<String> element = new ArrayList<String>();
68     if(!mergedTokensList.get(i).get(0).equals(mergedTokensList.get(pre).get(0))) {
69         t_index++;
70         element.add(Integer.toString(t_index));
71         element.add(mergedTokensList.get(i).get(0));
72         element.add("1");
73         irtm_dict.add(element);
74     } else {
75         int count = Integer.parseInt(irtm_dict.get(irtm_dict.size()-1).get(2));
76         count++;
77         irtm_dict.get(irtm_dict.size()-1).set(2, Integer.toString(count));
78     }
79 }

```

e. 輸出前面建立的 document

輸出方式和 PA1 相同，細節不贅述，路徑在此 java project 的 `src` 之下。

Code

```

83 // (4) Write document
84 File writhepath = new File("src/dictionary.txt");
85 writhepath.createNewFile();
86 BufferedWriter bw = new BufferedWriter(new FileWriter(writhepath));
87 String dictStr = "";
88 for(int i = 0; i < irtm_dict.size(); i++) {
89     dictStr = dictStr + irtm_dict.get(i).toString() + "\n";
90 }
91 bw.write(dictStr);
92 bw.flush();
93 bw.close();

```

(2) Transfer each document into a tf-idf unit vector.

a. 計算各個 term 的 idf

建立一個名為 `index_idf` 的 `ArrayList<ArrayList<String>>` 來儲存各個 term 的 index 及 idf。用 for 迴圈將 `irtm_dict` (有 `t_index`, `term`, `df`) 的每一個 element 取出，將 element 的 `t_index` 直接加入暫存的變數 `ArrayList<String> element` 中，再由 `irtm_dict` 中的 `df` 去計算出 `idf` 值後加入 `element`，最後再把 `element` 加入 `index_idf`，重複執行 `irtm_dict.size()` 次。

Code	
96	<code>/* 2. Transfer each document into a tf-idf unit vector */</code>
97	<code>// idf</code>
98	<code>index_idf = new ArrayList<ArrayList<String>>();</code>
99	<code>for(int i = 0; i < irtm_dict.size(); i++) {</code>
100	<code>ArrayList<String> element = new ArrayList<String>();</code>
101	<code>element.add(irtm_dict.get(i).get(0));</code>
102	<code>element.add(Double.toString(Math.log10(docSize/Integer.parseInt(irtm_dict.get(i).get(2)))));</code>
103	<code>index_idf.add(element);</code>
104	<code>}</code>

b. 計算 tf 值、tf-idf、tf-idf unit vector，並將各篇文章的 tf-idf unit vector 輸出至 output 資料夾

將單篇文章 tf、tf-idf、tf-idf unit vector 的計算寫成 method doc_tfidf(docID)，此 method 可以回傳整理好的 ArrayList<ArrayList<String>>，再由 method write_doc_tfidf(docID) 輸出成 txt 檔，重複執行 1095 次。

Code	
121	<code>// tf-idf</code>
122	<code>for (int docID = 1; docID <= docSize; docID++) {</code>
123	<code>write_doc_tfidf(docID, doc_tfidf(docID));</code>
124	<code>}</code>

以下說明 method doc_tfidf(int docID) 的處理邏輯：

首先建立了名為 term_f 的 ArrayList<ArrayList<String>> 來儲存單篇文章各個 term 的出現次數，做法和前面 dictionary 計算 df 方式類似。用 for 迴圈將最一開始的 tokensArrayList 每一個 element 取出來，如果該 element 的 docID 等於此 method 的 docID 參數，就接著和前一個 element 做比較，如果該 element 的 term 和 docID 與前一個不相同，就將該 term 加入 term_f，並給予 f=1；如果 element 的 term 和 docID 與前一個相同，表示這個 term 重複出現在同篇文章，則 f 要加 1（用自建的變數 count 來計算），更新 term_f 中該 term 的 f。因為 tokensArrayList 是有按照字母順序 sort 過的，所以同個字會排在一起，且 term_f 中的 element 是一個一個加進去，該 term 在 term_f 中的 index 會是當下的最後一個（size-1）。

Code	
185	<code>/* Method doc_tfidf(): calculate tf-idf for single document */</code>
186	<code>public static ArrayList<ArrayList<String>> doc_tfidf(int docID) {</code>
187	<code>// term_frequency</code>
188	<code>ArrayList<ArrayList<String>> term_f = new ArrayList<ArrayList<String>>();</code>
189	<code>ArrayList<String> element1 = new ArrayList<String>();</code>
190	<code>element1.add(tokensArrayList.get(0).get(0));</code>
191	<code>element1.add("1");</code>
192	<code>term_f.add(element1);</code>
193	<code>for(int i = 1; i < tokensArrayList.size(); i++) {</code>
194	<code>int pre = i-1;</code>
195	<code>ArrayList<String> element = new ArrayList<String>();</code>
196	<code>if(tokensArrayList.get(i).get(1).equals(Integer.toString(docID))) {</code>
197	<code>if(!tokensArrayList.get(i).equals(tokensArrayList.get(pre))) {</code>
198	<code>element.add(tokensArrayList.get(i).get(0));</code>
199	<code>element.add("1");</code>
200	<code>term_f.add(element);</code>
201	<code>} else{</code>
202	<code>int count = Integer.parseInt(term_f.get(term_f.size()-1).get(1));</code>
203	<code>count++;</code>
204	<code>term_f.get(term_f.size()-1).set(1, Integer.toString(count));</code>
205	<code>}</code>
206	<code>}</code>
207	<code>}</code>

單篇文章各個 term 的出現次數算出來後便可以計算 term frequency。先透過 for 迴圈將 term_f 中的所有次數相加總，再用另一個 for 迴圈將 term_f 中各個 term 的出現次數除以剛剛算出的加總值，將 term 和 tf 另外存在名為 term_tf 的 ArrayList<ArrayList<String>>。

Code

```

209 | // term_tf
210 | ArrayList<ArrayList<String>> term_tf = new ArrayList<ArrayList<String>>();
211 | int total_f = 0;
212 | for(int i = 0; i < term_f.size(); i++) {
213 |     total_f += Integer.parseInt(term_f.get(i).get(1));
214 | }
215 | for(int i = 0; i < term_f.size(); i++) {
216 |     ArrayList<String> element = new ArrayList<String>();
217 |     element.add(term_f.get(i).get(0));
218 |     element.add(Double.toString(Double.parseDouble(term_f.get(i).get(1))/total_f));
219 |     term_tf.add(element);
220 | }

```

接著建立一個名為 index_tf_idf 的 ArrayList<ArrayList<String>> 來儲存待會算出各個 term 的 tf-idf。首先用 for 迴圈將 term_tf 中各個 element 的 term 取出，透過第二層迴圈找出該 term 在 irtm_dict 中的 t_index。這裡建了一個名為 termIndex 的變數來記錄該 term 在 irtm_dict 所對應的 index，也就是 t_index - 1（Java 中 ArrayList 的 index 從 0 開始，但 t_index 是從 1 開始），由於 index 不可能等於 -1，所以將初始值設為 -1，若後續處理錯誤就可以發現錯誤。找出 termIndex 後便可將該 term 的 t_index 從 irtm_dict 取出，加入暫存的 ArrayList<String> element，再從 index_idf 中取出 t_index 所對應的 idf，去計算 tf-idf，最後將 element 加入 index_tf_idf 中，重複執行直到所有 term 都處理完。

Code

```

222 | // index_tf_idf
223 | ArrayList<ArrayList<String>> index_tf_idf = new ArrayList<ArrayList<String>>();
224 | for(int i = 0; i < term_tf.size(); i++) {
225 |     ArrayList<String> element = new ArrayList<String>();
226 |     Object term = term_tf.get(i).get(0);
227 |     int termIndex = -1;
228 |     for(int j = 0; j < irtm_dict.size(); j++) {
229 |         if(irtm_dict.get(j).get(1).equals(term)) {
230 |             termIndex = Integer.parseInt(irtm_dict.get(j).get(0)) - 1;
231 |         }
232 |     }
233 |     double idf = Double.parseDouble(index_idf.get(termIndex).get(1));
234 |     element.add(irtm_dict.get(termIndex).get(0));
235 |     element.add(Double.toString(Double.parseDouble(term_tf.get(i).get(1))*idf));
236 |     index_tf_idf.add(element);
237 | }

```

因為要將 tf-idf 轉換成 unit vector，所以先計算單篇文章的 vector length。

Code

```

239 | // vector length
240 | double sum = 0;
241 | for(int i = 0; i < index_tf_idf.size(); i++) {
242 |     sum += Math.pow((Double.parseDouble(index_tf_idf.get(i).get(1))),2);
243 | }
244 | double vector_length = Math.sqrt(sum);

```

最後利用 for 迴圈將 index_tf_idf 中的各個 tf_idf 值除以剛才算出的 vector length，另外存成名為 index_tf_idf_n 的 ArrayList<ArrayList<String>>，並回傳給呼叫方。

Code
<pre> 246 // normalization 247 ArrayList<ArrayList<String>> index_tf_idf_n = new ArrayList<ArrayList<String>>(); 248 for(int i = 0; i < index_tf_idf.size(); i++) { 249 ArrayList<String> element = new ArrayList<String>(); 250 element.add(index_tf_idf.get(i).get(0)); 251 element.add(Double.toString(Double.parseDouble(index_tf_idf.get(i).get(1))/vector_length)); 252 index_tf_idf_n.add(element); 253 } 254 255 return index_tf_idf_n; 256 }</pre>

以下說明 method write_doc_tfidf(int docID, ArrayList<ArrayList<String>> n) 的處理邏輯：

輸出方式和 PA1 及前面都相同，比較特別的是檔名會隨著 docID 而改變，因為是輸出單篇文章的 method，所以需要有 docID 的參數值。此外，在這裡的輸出和所要輸出的內容寫在不同 method，所以也需要所要輸出之內容的參數，由於此 method 是為了輸出 doc_tfidf(int docID) 所回傳的 ArrayList<ArrayList<String>> 而設計，便將參數型態設為 ArrayList<ArrayList<String>>。

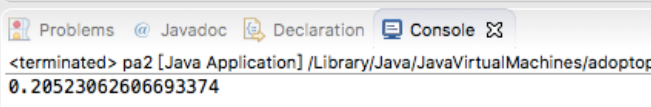
Code
<pre> 258 /* Method write_doc_tfidf(): write index & tf-idf to txt file for single document */ 259 public static void write_doc_tfidf(int docID, ArrayList<ArrayList<String>> n) throws Exception { 260 // output 261 File outputPath = new File(String.format("src/output/doc%d.txt", docID)); 262 outputPath.createNewFile(); 263 BufferedWriter tbw_output = new BufferedWriter(new FileWriter(outputPath)); 264 String output_result = ""; 265 for(int i = 0; i < n.size(); i++) { 266 output_result = output_result + n.get(i).toString() + "\n"; 267 } 268 tbw_output.write(output_result); 269 tbw_output.flush(); 270 tbw_output.close(); 271 }</pre>

(3) Write a function cosine(Docx, Docy) which loads the tf-idf vectors of documents x and y and returns their cosine similarity.

因為前面有寫了可以回傳單篇文章 tf-idf unit vector 的 method doc_tfidf(int docID)，因此不需要再另外讀取檔案，只要互叫該 method 即可。計算 similarity 是利用雙迴圈 for 找出 docx 和 docy 所有相同的 term (t_index 相同) 做內積，最後會回傳 similarity。

Code
<pre> 274 /* Method cosine(Docx, Docy) */ 275 public static double cosine(int docx, int docy) { 276 ArrayList<ArrayList<String>> x = doc_tfidf(docx); 277 ArrayList<ArrayList<String>> y = doc_tfidf(docy); 278 double similarity = 0; 279 for(int i = 0; i < x.size(); i++) { 280 for(int j = 0; j < y.size(); j++) { 281 if(x.get(i).get(0).equals(y.get(j).get(0))) { 282 double multi = Double.parseDouble(x.get(i).get(1)) * Double.parseDouble(y.get(j).get(1)); 283 similarity += multi; 284 } 285 } 286 } 287 return similarity; 288 }</pre>

若在 main 中呼叫此 method，給予參數 1 (doc1) 和 2 (doc2)，可以在 console 中印出 doc1 和 doc2 的 cosine similarity，約為 0.205，兩篇文章的相似度偏低。

Code
<pre>126 /* 3. doc1 and doc2 similarity */ 127 System.out.print(cosine(1,2)); 128</pre>  <p>The screenshot shows an IDE interface with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output: <code><terminated> pa2 [Java Application] /Library/Java/JavaVirtualMachines/adoptop...</code> followed by the value <code>0.20523062606693374</code>.</p>

2021.11.15