Coursework 7 (DocDiff and Danube.org)

This coursework is worth 10%. The first part and second part are due on 22 November at 11pm; the third, more advanced part, is due on 21 December at 11pm. You are asked to implement Scala programs for measuring similarity in texts and for recommending movies according to a ratings list. Note the second part might include material you have not yet seen in the first two lectures.

Important:

- Make sure the files you submit can be processed by just calling `scala <<filename.scala>>` on the commandline.[1] Use the template files provided and do not make any changes to arguments of functions or to any types. You are free to implement any auxiliary function you might need.
- Do not leave any test cases running in your code because this might slow down your program! Comment test cases out before submission, otherwise you might hit a time-out.
- Do not use any mutable data structures in your submissions! They are not needed. This means you cannot create new `Arrays` or `ListBuffers`, for example.
- Do not use `return` in your code! It has a different meaning in Scala than in Java.
- Do not use `var`! This declares a mutable variable. Only use `val`!
- Do not use any parallel collections! No `.par` therefore! Our testing and marking infrastructure is not set up for it.

Also note that the running time of each part will be restricted to a maximum of 30 seconds on my laptop.

Disclaimer

It should be understood that the work you submit represents your **own** effort! You have not copied from anyone else. An exception is the Scala code I showed during the lectures or uploaded to KEATS, which you can freely use.

[1] All major OSes, including Windows, have a commandline. So there is no good reason to not download Scala, install it and run it on your own computer. Just do it!

Reference Implementation

Like the C++ assignments, the Scala assignments will work like this: you push your files to GitHub and receive (after sometimes a long delay) some automated feedback. In the end we take a snapshot of the submitted files and apply an automated marking script to them.

In addition, the Scala assignments come with a reference implementation in form of a `jar`-file. This allows you to run any test cases on your own computer. For example you can call Scala on the command line with the option `-cp docdiff.jar` and then query any function from the template file. Say you want to find out what the function `occurences` produces: for this you just need to prefix it with the object name `CW7a` (and `CW7b` respectively for `danube.jar`). If you want to find out what these functions produce for the list `List("a", "b", "b")`, you would type something like:

```
$ scala -cp docdiff.jar

scala> CW7a.occurences(List("a", "b", "b"))
...
```

Hints

Part 1 (4 Marks, file docdiff.scala)

It seems source code plagiarism—stealing someone else's code—is a serious problem at other universities.[2] Dedecting such plagiarism is time-consuming and disheartening. To aid the poor lecturers at other universities, let's implement a program that determines the similarity between two documents (be they code or English texts). A document will be represented as a list of strings.

Tasks

(1) Implement a function that cleans a string by finding all words in this string. For this use the regular expression `"\w+"` and the library function `findAllIn`. The function should return a list of strings.

[1 Mark]

(2) In order to compute the similarity between two documents, we associate each document with a `Map`. This Map represents the strings in a document and how many times these strings occur in a document. A simple (though slightly inefficient) method for counting the number of string-occurences in a document is as follows: remove all duplicates from the document; for each of these (unique) strings, count how many times they occur in the original document. Return a Map from strings to occurences. For example

```
occurences(List("a", "b", "b", "c", "d"))
```

produces `Map(a -> 1, b -> 2, c -> 1, d -> 1)` and

```
occurences(List("d", "b", "d", "b", "d"))
```

produces `Map(d -> 3, b -> 2)`. [1 Mark]

(3) You can think of the Maps calculated under (2) as efficient representations of sparse "vectors". In this subtask you need to implement the *product* of two vectors, sometimes also called *dot product*.[3]

For this implement a function that takes two documents (`List[String]`) as arguments. The function first calculates the (unique) strings in both. For each string, it multiplies the occurences in each document. If a string does not occur in one of the documents, then the product is zero. At the end you sum all products. For the two documents in (2) the dot product is 7:

[2] Surely, King's students, after all their instructions and warnings, would never commit such an offence.

[3] https://en.wikipedia.org/wiki/Dot_product

$$\underbrace{1 * 0}_{"a"} + \underbrace{2 * 2}_{"b"} + \underbrace{1 * 0}_{"c"} + \underbrace{1 * 3}_{"d"}$$

[1 Mark]

(4) Implement first a function that calculates the overlap between two documents, say d_1 and d_2, according to the formula

$$\texttt{overlap}(d_1, d_2) = \frac{d_1 \cdot d_2}{max(d_1^2, d_2^2)}$$

This function should return a `Double` between 0 and 1. The overlap between the lists in (2) is 0.5384615384615384.

Second implement a function that calculates the similarity of two strings, by first extracting the strings using the function from (1) and then calculating the overlap. [1 Mark]

You are creating Danube.org, which you hope will be the next big thing in online movie provider. You know that you can save money by anticipating what movies people will rent; you will pass these savings on to your users by offering a discount if they rent movies that Danube.org recommends. This assignment is meant to calculate

To do this, you offer an incentive for people to upload their lists of recommended books. From their lists, you can establish suggested pairs. A pair of books is a suggested pair if both books appear on one person's recommendation list. Of course, some suggested pairs are more popular than others. Also, any given book is paired with some books much more frequently than with others.