PROGRAMMING ASSIGNMENT 06: SIMILARITY DETECTOR

Estimated reading time: 10 minutes
Estimated time to complete: two to three hours (plus debugging time)
Prerequisites: Assignment 03
Starter code: similarity-detector-student.zip
Collaboration: **not permitted**

Overview

Many classes at UMass and other colleges and universities use one or more automated tools to alert instructors to possible plagiarism. These tools, such as turnitin and MossPlus, are used to evaluate the *similarity* of texts to one another and/or to large corpora of source material. Turnitin, for example, doesn't just check student assignments against one another – it can compare against assignments submitted in previous years (thus helping deter paper banks), or against all of Wikipedia, or against other digitized texts.

At its core, these services have to compare the similarity between texts (or program code). How do they work? In this assignment, you'll build a simplified similarity detection system and find out. Your system will convert input texts into *sets* representing those texts, and will compare those sets using the Jaccard index, a measure of the similarity between sets.

We've provided a large set of unit tests to help with automated testing, though you might also want to write a class with a `main` method for interactive testing. The Gradescope autograder includes a few more tests, but they exist primarily to verify you're not gaming the autograder. If your code can pass the tests we've provided, it is likely correct.

As before, we've disable the timeout code so you can use the debugger, but if your code gets stuck during testing, you might want to uncomment these two lines at the top of each test file:

```
@Rule
public Timeout globalTimeout = Timeout.seconds(10); // 10 seconds
```

Goals

- Translate written descriptions of behavior into code.
- Practice writing static methods.
- Practice interacting with the `Set` and `List` abstractions.
- Test code using unit tests.

Downloading and importing the starter code

As in previous assignments, download and save (but do not decompress) the provided archive file containing the starter code. Then import it into Eclipse in the same way; you should end up with a `similarity-detector-student` project in the "Project Explorer".

What to do

Complete the code in both `SetUtilities` and `SimilarityUtilities`.

`SetUtilities`

`SetUtilities` has four methods for you to complete. Reading the type signatures:

```
public static <E> Set<E> union(Set<E> s, Set<E> t)
```

you should understand that these are *generic methods*, that is, they are parameterized on type `<E>`. `union` will work on any two of same-typed `Set<E>`s. That's what the `<E>` before the return type of a `static` method indicates: this `static` method is generic on type `<E>`.

These four methods should be straightforward to implement. Get them done early, because you'll need them for the next part of the assignment.

`SimilarityUtilities`

There are six methods to complete here.

As in previous assignments, you'll find `String.split` helpful. But I don't expect you to know "regular expressions" at this point, so here are two you'll need:

- Passing `"\\n"` to `String.split` will split the original string on newlines (`'\n'`s). It will return an array of strings representing the split lines. You may still need to handle empty lines yourself, though.
- Passing `"\\W+"` (note: uppercase `W`) to `String.split` will split the original string on non-word characters. (`\W` means "a non-word character" and + means "one or more times in a row"). You may still need to handle empty words yourself, as above.

You may find other instance methods of `String`, such as `trim` and `toLowercase`, helpful.

In the first `lineSimilarity` method, your code should convert each text to a set using `trimmedLines`, then compare them using `jaccardIndex`. There is no trick here!

In the second `lineSimilarity` method, there is a `template` argument. The `template` is a string that you *don't* want to consider when comparing the similarity of two texts. For example, if I were checking your programming assignments for similarity, I would include the starter code as the template. To "ignore" the template, convert a text to a set and a template to a set, then use set difference to remove the template set from the text set. Do so for both input texts, then find their Jaccard index.

Then comes the `shingle` method. Shingling is a venerable technique for finding partial duplicates (see, for example, http://nlp.stanford.edu/IR-book/html/htmledition/near-duplicates-and-shingling-1.html) that creates a set of multiple overlapping pieces of a text for further comparison. The Java doc has an example, and the JUnit test cases have further examples; it should be straightforward to generalize from those examples. You're likely to need a pair of nested `for` loops.

Finally, implement the `shingleSimilarity` method, which parallels the second `lineSimilarity` method but requires shingling the words of the text, rather than the trimmed lines.

Submitting the assignment

When you have completed the changes to your code, you should export an archive file containing the entire Java project. To do this, follow the same steps as from Assignment 01 to produce a `.zip` file, and upload it to Gradescope.

Remember, you can resubmit the assignment as many times as you want, until the deadline. If it turns out you missed something and your code doesn't pass 100% of the tests, you can keep working until it does.