CSCI 230 Data Structures & Algorithms

Project 4 Catching Plagiarists[1]

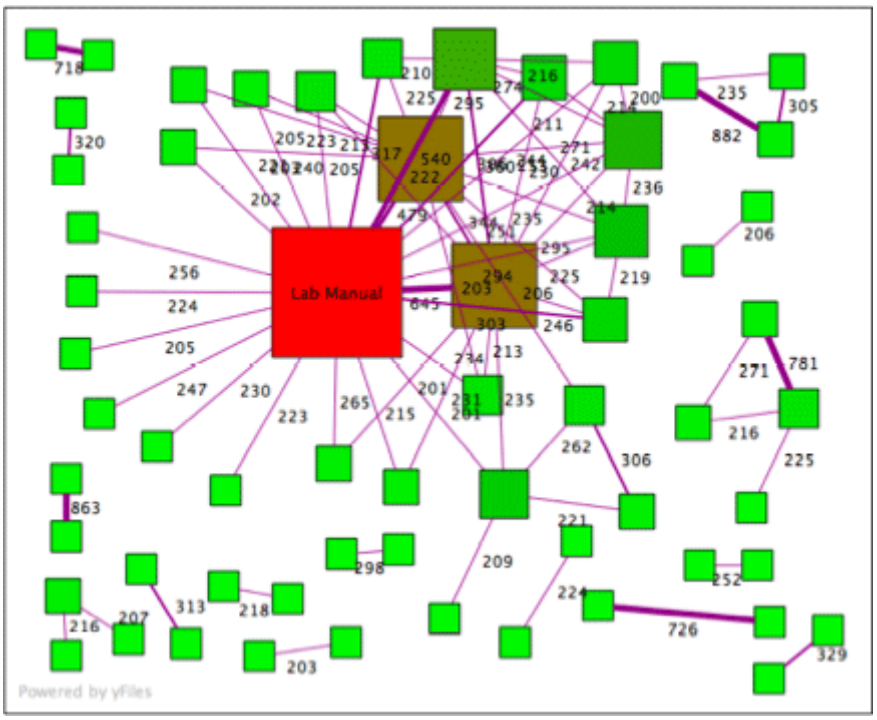
Crafton Hills College
Total Points: 100

Overview

This project presents a real problem that requires a software solution. Your goal is to try to (quickly) determine the similarities between documents in a large set to see if you can find out if plagiarism is going on within the group.

Below is an actual graph of lab reports submitted for Intro. Physics at a large University. This graph represents the data collected for about 800 lab reports. Each node in the graph represents some document. Each edge indicates the number of 6-word phrases shared between the documents it connects. To reduce "noise" a threshold of 200 common phrases has been set – so a document that shares fewer than 200 6-word phrases with all other documents is not shown. The "Lab Manual" is a sort of style-guide for the lab report and the two brown boxes are sample lab reports that were distributed. (Many people apparently "borrowed liberally" from these help materials). Particularly suspicious are clusters like the one in the top-right corner: those documents have an inordinate number of 6-word phrases in common with each other. It is likely that those people turned in essentially the same lab report or copied large portions from each other.

[1] Based on: http://people.ucls.uchicago.edu/~bfranke/nifty.bfranke/ and http://faculty.washington.edu/joelross/courses/archive/s13/cs261/hwk/7/



The Plagiarism Detection Program

Your task is very similar to the one described and shown above: find the common word sequences among documents in a closed set. Simply put, your input will be a set of plain-text documents, and a number n; your output will be some representation showing the number of n-word sequences each document has in common with every other document in the set.

Finally, you should identify "suspicious" groups of documents that share many common word-sequences among themselves but not with others. Your program should accept an additional input representing the detection threshold, which is the number of matches for a pair of documents above which plagiarism is flagged.

A sample program for reading a directory of documents and parsing the documents into n-word sequences is provided for your reference (`DocumentReader.java`).

Output

A nice product would be a simple console application that accepts 3 command-line arguments. For example:

```
./plagiarismCatcher path/to/files 6 200
```

which would churn and then produce a list (in order) of all the pairs of files in `path/to/docs` that shared more than 200 6-word sequences in common.

You can think of processing everything into an NxN matrix (where N is the number of total documents) with a number in each cell representing the number of "hits" between any pair of documents. For example: below is a small table showing the comparisons between 5 documents:

	A	B	C	D	E
A	-	4	50	700	0
B	-	-	0	0	5
C	-	-	-	50	0
D	-	-	-	-	0
E	-	-	-	-	-

From this table we can conclude that the writers of documents A, C and D share a high number of similar 6-word phrases. We can probably say A and D cheated with a high degree of certainty.

For a large set of documents, you may only want to print a matrix for those documents with a high number of hits above a certain threshold.

Printing an NxN matrix may be unmanageable for large sets. You could instead produce a list of documents ordered by number of hits. For example:

```
700:  A, D
50:   A, C
50:   C, D
5:    B, E
4:    A, B
```

THE DOCUMENTS

Multiple datasets are provided:

Three sets of documents (proj4-datasets.zip) for which your program must work are provided. The documents came from www.freeessays.cc, a repository of *really bad* high school and middle school essays on a variety of topics.

- A small set containing about 25 documents (~34,000 words). This set is appropriate for testing that things work.
- A medium set containing about 75 documents (~90,000 words). This set can help test your program's scalability.
- A large set containing over 1300 documents (~1.6 million words). This set will test that your program can work at massive scale, and will make data structure type and algorithm speed important!

Each set contains one plagiarized document called catchmeifyoucan.txt. The medium and small sets of documents are just subsets of large big document set.

Your program should be able to process all of the documents in a given folder/directory.

Additional datasets are provided online at https://drive.google.com/drive/folders/0Bx5tTdyTEHTvVzBJNld4MDRoelk?usp=sharing. Try your best to make your program work for as many of these datasets as possible.

To get full credits, your program must work correctly for all three datasets in proj4-datasets.zip *and* three additional datasets provided online (docs50.zip, docs100.zip, docs200.zip, docs300.zip, docs400.zip).

STRATEGY

How are you going to do this? **The strategy is entirely up to you.** The straightforward matrix solution (comparing each six-word sequence, say, to all other six-word sequences) gives an $O(n^2)$ solution – where n is the total number of all words in all documents. For a large set of documents n^2 grows very large, very fast. It will work though – it will just take a while. For perspective, if the 25-document set takes 10 seconds to process this way, the 1300-document set will take over 6 hours…if you can actually hold the necessary data in memory which you probably can't.

Instead, you'll want to come up with a more clever solution that can be more efficient. You'll need to consider both the run-time efficiency (i.e., the big-O for your algorithm), and the space efficiency (what can you fit into memory?). Think about the data structures and algorithms we've discussed.

One way to gain ultimate control over the processing is to write your own specialized data structures. However, you're free to use anything in the Java API.

You should think about the strategy on your own, but if you need some inspiration, the following ideas might help:

- Strings can take up a lot of memory: 1 byte for each character, times the number of characters, and since each word shows up in multiple n-grams... ints on the other hand always take up only a fixed number of bytes. Converting the Strings into ints might help with space.
- You basically need to be able to search each document for a phrase that is in another document. Think about what data structures and algorithms led to fast searching.
- You'll need to keep track of how many "matches" you get between each document, but you don't necessarily need to store which n-grams match. What kind of data structure might let you track this information?

OTHER REQUIREMENTS

1. Code should be well-documented, including all classes and class members. Follow the style guidelines for the Javadoc tool (http://www.oracle.com/technetwork/articles/java/index-137868.html).
2. Record the running time of your program on the three provided data sets. You will need to show the results in your project report.
3. A report containing the following items:
    - Your name
    - Program Design Explanation
        - Describe the data structure (s) and algorithm(s) you used in your solution. Please be detailed--explain to me how your program works!
        - Why did you choose those data structure(s) and algorithm(s)?
        - How did you deal with the trade-off between space and time efficiency in implementing your solution?
    - Results of your program running on the provided data sets. For each data set, include
        - The outputs when detecting for pairs of files that shared more than 200 6-word sequences in common
        - Running times
            - You should run your program on different data sets on the same machine. Provide a short specification of the machine (processor model & speed, amount of RAM, hard drive capacity)
            - Show the results of your project in terms of running times on the three provided data sets and the (three minimum) additional test data sets that you used (you can show a chart/graph/table)
    - A brief discussion of your project experience
        - Did you enjoy this project? What problems did you encounter?
        - What did you get out from the project?
        - How did you find the project (too easy, easy, just right, difficult, too difficult)?
        - What type of help/references did you use in your project (e.g. book, web sites, classmates, tutors)? List their names.
        - If you worked with a partner, describe the roles of each group member.
4. Your source code and report must be submitted properly on Canvas. (IMPORTANT: if I am not able to read your files and/or run your program, your project will not be graded. It is your responsibility to make sure all files are submitted correctly before the deadline.)

If you work in a group, only one group member submits the project (code, report, etc.).

GRADING CRITERIA

1. Satisfaction of project requirements (75 points)
    - Calculates and reports the number of matching n-grams above a given threshold for a given document set (30 points)
    - Works with reasonable efficiency for the small, medium, large document sets in proj4-datasets.zip (20 points)
    - Works with reasonable efficiency for three additional datasets provided (15 points)
    - Demonstrates good program design (modularity, object-oriented design, data structures, algorithms) (10 points)
2. Well-written Report (15 points)
3. Documentation and coding style (10 points)
    - Provide sufficient documentation in the source code. **Each class and method should be documented clearly.**
    - Your name should be present in each user-defined class.
    - Use descriptive identifiers.
    - Use proper spacing and indentation (refer to the textbook's program style).

Need some giggles while you are taking a short break from this project?

Listen to this humorous song (Tom Lehrer – "Lobachevsky"):

https://www.youtube.com/watch?v=gXlfXirQF3A