CSCI 241 Data Structures
Assignment 1 (Automated Word Discovery)
Total Point: 35 (15% of course grade)
Due: Thursday, Jan 26, 2017 (10:00 pm)
Pair Team Programming
Late work will not be accepted

Overview

For the first programming assignment, you will work with your selected or assigned partner to create a program that is able to automatically discover words, in a crude approximation to human language acquisition. Its concept of what makes a word is somewhat limited: it thinks that words are sequences of characters or character sequences that co-occur frequently. From the natural language processing community, you will be using a quantity known as the 'bigram product", which was introduced to learn *multiword units* (e.g. learn that "San Jose" or "New York" really function like a single word). When it learns from a Java textbook, using a particular configuration of the program thresholds, it may discover words such as **a, abbreviated, ability, able, above, abstract, abstraction, abstractions,** etc.

You will implement this functionality in the **Wordifier** class, a skeleton **.java** file with several dummy methods that will be provided to you. The pre- and post-conditions for each method will be specified; it will be your job to complete all of the empty methods according to these conditions. You are free to add as many additional private helper methods as you would like. Now that you have a sense of what you will be doing, let's take a look at how you will be working on the assignment.

Pair Team Programming[1]

Pair programming is a software development technique where two programmers work together in front of one keyboard. One partner types code while the other is suggesting and/or reviewing every line of code as it is being typed. The person typing is called the driver. The person reviewing and/or suggesting code is called the observer or the navigator. The two programmers should switch roles

[1] These guidelines are based on versions developed by Perry Fizzano, Brian Hutchinson, Moushumi Sharmin, and Shameem Ahmed

frequently (e.g. every 20 to 30 minutes). For this to be a successful technique the team needs to start with a good program design so that they are on the same page when it is finally time to start typing on the computer. **In the true pair-programming model, no designing or programming is done without both partners are present!** Pair-programming has been shown to increase productivity in industry and may well increase yours, but there are additional reasons it is being used in this class. First, it is a means to increase collaboration, which is something department graduates now working in industry report that they wish they had more experience with. Second, working in pairs is a good teaching tool. Inevitably, in each pairing, the partners will have different styles and abilities (for instance, one person may be better at seeing the big picture while the other is better at finding detailed bugs or one person might like to code on paper first while the other likes to type it in and try it out). Because of that, you will have to learn to adjust to another person's style and ideally, you will meet each other half way when there are differences in approach. It is important that each person completely understands the program and so both parties need to be assertive. Be sure to explain your ideas carefully and ask questions when you are confused. In addition, it is crucial that you be patient! There is plenty of time allotted to complete this assignment as long as you proceed at a steady pace. Ask for help from me or the department tutors if you need it.

In team programming, tasks are divided into sub-tasks, team members are assigned to different sub-tasks, and they combine the separately produced code to create the final solution. In many environments, team members are not required to know the details of other member's code. However, they do need to understand the bigger picture, how the implementation is addressing the programming problem.

In our class, we will use a mixed approach, where each team is encouraged to follow the true pair-programming model whenever possible. If for any reason your team needs to divide responsibility, team members will be responsible for understanding the details of every code developed by him/her and the other member. This approach relaxes the requirements of pair programming by allowing you to work independently on some aspects of the assignment if needed, but you are still responsible for understanding every line of code produced and the rationale for using that strategy.

You will have a partner selected by yourself. Because there is no lab, you will need to coordinate with your partner to find times when you can both be present. You will need to contact me ASAP if there is any reason you will not be able to collaborate. Let me stress again that both

partners need to understand every detail of their implementation. **If I determine that one of you do not understand certain parts of the code, you and your partner will receive no credit for this assignment.**

Development and Testing

A program named Program1 (**Program1.java**) will be provided to you. **Do not modify Program1's code**. This program will drive your *Wordifier* class, will call the methods to learn the words, and will call the methods to evaluate them. It will accomplish this by calling public methods from your *Wordifier* class. Because it will call your methods, you must not change the method header for any of the public methods. You must supply this program with four arguments:

- The name of the input text file
- A minimum count threshold (used when learning which tokens to merge)
- A probability threshold (also used when deciding which tokens to merge) and
- The name of a dictionary file that it will use to evaluate the words you have learned.

Here is an example:

C:> java.exe Program1 javaTextBook.txt 5 0.03 dictionary.txt

or (in Linux):

$ java Program1 javaTextBook.txt 5 0.03 dictionary.txt

The grader and I will use Program1 for grading; it is available for you to use during development. The results produced by our completed version on a specific set of test files will be available to you, but not until close to the deadline. It is important that you develop your own test cases to confirm the correctness of your code. Once our cases are released, be sure to compare the output of your code against these files to make sure that your formatting is identical (and that your output is correct). You should test your class on at least two new text files, (name them **test1.txt** and **test2.txt** and include them with your submission).

Grading

- **Submitting your work:** When the clock strikes 10 PM on the due date, we will check out the latest submitted version of your assignment from Canvas. (**Do not forget to submit your latest working version before the due date**!) Your submission should include, at the least:

1. Wordifier.java
2. Your write-up
3. Your two new test input file you have created (less than 5MB per file, please)
    - Name your new test files test1.txt, test2.txt, etc.
4. Any other source code needed to compile your program/class.

You should not submit your .class files. Upon checking out your files, I will replace your version of Program1.java with my original one, compile all .java files, run *Program1* against a series of test documents, analyze your code, and read your write-up.

- **Points:** This assignment will be scored by taking the points earned and subtracting any deductions. You can earn up to 35 points:

Component	**Points**
Write Up & Test Cases	5
Contribution Summary	2
loadSentences	2
findNewWords	3
Resegment	4
computeCounts	3
convertCountsToProbabilities	4
getScores	4
getVocabulary	2
loadDictionary	2
printNumWordsDiscovered	4
Total	**35**

You may also have deductions from your score for Poor coding style (e.g. bad indentation, non-standard naming conventions) or errors on compiling or running.

- **Write-Up & Test Cases:** With your partner, in one or two pages, provide a write-up of your implementation. Please submit your write-up as a **plain text** file named **writeup.txt** (e.g. created by Notepad, or vim, or emacs). Your write-up should include the following points:
    1. Your names

2. An acknowledgment and discussion of any parts of the program that are not working. Failure to disclose obvious problems will result in additional penalties.
3. An acknowledgment and discussion of any parts of the program that appear to be inefficient (in either time or space complexity).
4. A discussion of the portions of the assignment that were most challenging. What about those portions was challenging?
5. A discussion on how you approached testing that your program was correct and asymptotically efficient. What did test1.txt test? What did test2.txt test?
6. What combination of input file, counts threshold and probability threshold led to discovering the largest number of unique words; which led to discovering the largest total number of words?

- **Contribution Summary:** In an email, provide a two-paragraph write-up of your and your partner's contribution to this assignment. Please submit your write-up as an **email**. Your write-up should include the following points:
    1. Your names
    2. One paragraph explaining your contribution to this assignment. Please include examples of methods you implemented, how you contributed to the design, bug fixing efforts, etc.
    3. One paragraph explaining your partners' contribution to this assignment. Please include examples of methods your partner implemented, design ideas, bug fixes, etc.
    4. Any collaboration problem experienced and how you approached the problem and solved it.

Details

- **The Bigram Product:** A bigram is a pair of text tokens; for example, "John Doe" is a bigram, as is "computer science" as is "t h". The bigram product (BP) is the product of two conditional probabilities:

$$BP(x_1, x_2) = P(x_1 \text{ preceding } x_2)P(x_2 \text{ following } x_1)$$

It can be rewritten, however, in an easier to use the form:

$$BP(x_1, x_2) = \frac{P(x_1, x_2)}{\sqrt{P_L(x_1)P_R(x_2)}}$$

Here, P(x1, x2) denotes an empirical joint probability; i.e., the number of times we saw the bigram x1x2 over the total number of bigrams in the data. $P_L(x)$ denotes an empirical unigram (one word) probability for words in the left position; i.e., the number of times we saw word x in the first (left) position over the total number of bigrams in the data. $P_R(x)$ denotes the empirical unigram probability for words in the second (right) position.

- **Program1's Behavior:** Program1.java drives the overall behavior. It does the following:
  1. Load the text data into a LinkedList representing the data
  2. Load the dictionary (provided) into a HashSet
  3. While not converged
     a. Count bigram occurrences in the data
     b. Convert the bigram counts into bigram joint probabilities and token unigram probabilities.
     c. Compute the bigram product scores for all observed bigrams using the probabilities
     d. Select a subset of the bigrams to "merge" into new tokens; picking ones that meet the count and probability thresholds
     e. If no bigrams were selected to merge, we have converged
     f. If 1+ bigrams were selected to merge, then resegment the data, where pairs of words selected in to be merged are merged in the data. For example, if the previous data consisted of the sequence "A B C D E F G H I", and two bigrams were selected to be merged (say, "B C" and "G H"), then resegment would produce data consisting of the sequence "A BC D E F GH I".
  4. Identify the vocabulary of the final segmentation (the set of all unique tokens in the data)
  5. Compare the vocabulary against the dictionary, printing the tokens that are actual dictionary words, and printing some percentages of how many words you got right. To do this last step, you need to understand the difference between tokens and types. The sentence "the dog chased the cat" has four types (unique words): {the, dog, chased, cat}, but it has five tokens: the, dog, chased, the, cat. The phrase "E I E I O" has three types ({E, I, O}) but five tokens.

Academic Honesty

To remind you: aside from your designated partner, you must not share code with your classmates: you must not look at others' code or show your classmates your code. You cannot take, in part or in whole, any code from any outside source, including the Internet, nor can you post your code to it. If you and your partner need help from another pair, all involved should step away from the computer and discuss strategies and approaches, not code specifics. I am available for help during office hours, as are department tutors, but you should attend these hours with your partner. I am also available via email (make sure you and your partner are included in the email and do not wait until the last minute to email). If you participate in academic dishonesty, you will fail this course.