

Computer Science 241

(Pair) Program 2 (50 points)

Due Monday, May 12th, 2014 at 10:00 PM

Read all of the instructions. Late work will not be accepted.

Overview

For the second programming assignment you will work on your own to create a speech recognition program. Specifically, it is capable of so-called “second pass” decoding, which takes as its input a special directed acyclic graph called a *lattice*. Each node in a lattice corresponds to a point in time during a sentence, and each edge corresponds to a word that might have been uttered between the two end-points. Each path from the special start node to the special end node corresponds to one possible *hypothesis* for the sentence that was uttered. Each edge is weighted by two scores: an acoustic model score (`amScore`) and a language model score (`lmScore`). You can combined these into a single score by taking a weighted sum, weighted by a parameter called the language model scale (`lmScale`). The combined score for an edge e is:

$$\text{combinedScore}(e) = \text{amScore}(e) + \text{lmScale} * \text{lmScore}(e)$$

The score of a path is just the sum of the scores of its edges. These scores represent negative log probabilities, so the lower the score the better. The shortest path, then, is the best path. Conveniently, there are very efficient algorithms for computing the shortest path through a directed acyclic graph. For example, even in a graph with 9.518×10^{86} unique paths (i.e. possible hypotheses), the best hypothesis can be found in less than two seconds on a standard laptop.

You will implement this functionality in the `Lattice` class. A skeleton `.java` file with several dummy methods have been provided to you to complete. The pre- and post-conditions for each method are specified; it will be your job to complete all of the methods according to those conditions. Additionally, `Lattice` relies upon two classes that have already been implemented for you: `Hypothesis`, for representing and performing operations on a speech recognition hypothesis, and `Edge`, which is used to store the information associates with each edge in the lattice. You are welcome to add as many `private` helper methods as you would like.

Pair Programming

As in Program 1, this is a pair programming assignment. You must work together with your assignment partner to design, develop and debug your work for Program 2. Recall that at any given moment when coding, one person (the “driver”) will be writing the code, while the other will be reviewing and offering suggestions (the “navigator”). You and your partner should switch roles frequently. To succeed, you must both be on the same page and both be engaged in the development of the program. No designing, development or debugging

can be done without both partners present. The goal of this approach is to increase your productivity and expose you to new approaches to developing code. Often the same bug that would take you three bleary-eyed hours to hunt down alone can be found within minutes by a second pair of eyes. 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.

I will assign your partner via Canvas. There is no lab, so you will need coordinate with your partner to find times when you can both be present. You will need to contact me ASAP if you cannot find sufficient overlap in your schedules to collaborate. **Let me stress again that no designing or programming is to be done without both partners present! If I determine this happened I will fail you and your partner for this assignment.**

Development and Testing

A program named Program2 (Program2.java) has been provided to you. **Do not modify Program2's code.** This program drives your Lattice class: it reads in test lattices, computes their best hypotheses, and reports various statistics of the lattices. It accomplishes this by generating new instances of type Lattice. Because Program2 interacts with your Lattice class, you must not change the method header for any of the public methods in Lattice.java (including, for example, adding new throws flags). You must supply this program with three arguments: the name of a “lattice list,”¹ an lmScale value for combining the acoustic and language model scores into a single combined score, and the name of an (**already created**) output directory, where two different representations of the lattice will be written. An example usage is:

```
C:> java.exe Program2 someLatticeListFile.txt 8.0 someOutputDirectory
```

or (in Linux):

```
$ java Program2 someLatticeListFile.txt 8.0 someOutputDirectory
```

I will use Program2 for grading; it is available for you to use during development. I will post limited examples of output by my code so that you can check that your formatting is correct. You should develop at least two new, simple, lattices and use these as test cases.

Visualizing Lattices

For each .lattice file in the input lattice list, two files will be created in someOutputDirectory: 1) another .lattice file that should be *identical* to the input .lattice file, 2) a .dot file that can be compiled into a pdf or svg image using the Graphviz dot program. At least for small lattices, the graphical representation can be a very helpful tool for understanding the data. To compile a .dot file into a pdf in Linux, use the following command

```
$ dot -Tpdf inputFileNamesHere.dot -o outputFileNamesHere.pdf
```

¹A lattice list simply contains a list of lattice files to process, along with their corresponding reference files. The file has one line per utterance. Each line begins with the name of a .lattice file, followed by a space, followed by the name of the corresponding .ref file.

Simply replace `pdf` with `svg` everywhere it occurs to compile to `svg` format. You can view `svg` files with the `inkscape` program. Be warned that for very large lattices, the `dot` command can take a prohibitively long time to run.

The Repository

You will develop this program under a Subversion version control repository. It does not matter which partner hosts the repository; I will check out both and grade the one that contains a `prog2` directory. The primary difference is that this time you will put your code in a new subdirectory named `prog2` (capitalization, spacing and spelling matter!), rather than `prog1`.

You *must* use this repository for the development and submission of your work (failure to have **8 or more** non-trivial revisions will result in **non-trivial penalties**). If your repository was not used for Program 1 but will be used for Program 2, verify that you are able to access and interact with it *as soon as possible* and contact me if there is a problem.

Grading

Submitting your work

When the clock strikes 10 PM on the due date, a script will automatically check out the latest committed version of your assignment. (**Do not forget to commit the work you want submitted before the due date!**) Your repository should have in it, at the least:

- `Lattice.java`
- Your write-up (named `writeup.txt`)
- At least new test input pairs you have created
 - Name your new test files `test1.lattice` (and `test1.ref`), `test2.lattice` (and `test2.ref`), etc.
 - Create a test list named `test.list` that contains all test lattices and refs
 - All of these test files should be in `prog2` itself, not a sub-directory
- Any other source code needed to compile your program / classes

Your repository need not and **should not contain your .class files**. Upon checking out your files, I will replace your version of `Program2.java`, `Edge.java` and `Hypothesis.java` with my original ones, compile all `.java` files, run `Program2` against a series of test lattices, analyze your code, and read your documentation.

Points

This assignment will be scored by taking the points earned and subtracting any deductions. You can earn up to 50 points.

Component	Points
Write Up & Test Cases	5
Lattice constructor	7
Lattice getUtteranceID	1
Lattice getNumNodes	1
Lattice getNumEdges	1
Lattice toString	3
Lattice decode	7
Lattice topologicalSort	7
Lattice countAllPaths	5
Lattice getLatticeDensity	2
Lattice writeAsDot	3
Lattice saveAsFile	1
Lattice uniqueWordsAtTime	3
Lattice printSortedHits	4
Total	50

You may also have deductions from your score for

- Poor code style (e.g. bad indentation, non-standard naming conventions)
- Inadequate versioning
- Errors compiling or running

Write-Up & Test Cases

In one or two pages, provide a write-up of your implementation. Submit your writeup as a **plaintext** file² named **writeup.txt**. Please include all of the following:

1. Your name
2. An acknowledgement 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.lattice` test? What did `test2.lattice` test?
6. Your observations on the effect of `lmScale` value on the kinds of sentences produced? For example, do the results differ systematically for low `lmScales` (≤ 3) vs high `lmScales` (≥ 20).
7. (Optional) Any other thoughts or comments you have on the assignment and its implementation.

²E.g. created by Notepad, or vim, or emacs.

Lattice Format

The provided lattices are in a very simple format. An example complete lattice file is listed below:

```
id CMU_20020319-1400_101_denise_1_0200626_0201468
start 0
end 6
numNodes 7
numEdges 7
node 0 0.00
node 1 0.00
node 2 0.00
node 3 0.19
node 4 0.19
node 5 0.62
node 6 0.83
edge 0 1 -silence- 0 78
edge 0 2 -silence- 0 38
edge 1 3 -silence- 2325 0
edge 2 4 -silence- 2325 0
edge 3 5 ok 7414 12
edge 4 5 okay 7414 7
edge 5 6 -silence- 862 0
```

Each line begins with one of the following keywords:

- **id.** The next token after this keyword will be the identifier for the utterance (a string).
- **start.** The next token after this keyword is the index of the special start node (an integer). This is usually 0, but need not be.
- **end.** The next token after this keyword is the index of the special end node (an integer). This is usually the node with the largest index in the graph, but need not be.
- **numNodes.** The next token after this keyword is the number of nodes in the graph (an integer).
- **numEdges.** The next token after this keyword is the number of edges in the graph (an integer).
- **node.** This defines one node in the graph. The first token after the keyword gives the node index (an integer); the second token after the keyword gives the time associated with the node (a double).
- **edge.** This defines one edge in the graph. The first token after the keyword gives the source node index (an integer) and the second token gives the destination node index (an integer). For example, 0 5 indicates that this is an edge from node 0 to node

5. The third token after the keyword is the word/label associated with the edge (a string). The fourth and fifth tokens are the acoustic and language model scores (both integers), respectively.

Dot Format

There are many kinds of graphs that can be described in dot format. Ours will be simple. The files you produce will consist of three parts:

1. It will always begin with this header:

```
digraph g {  
    rankdir="LR"
```

2. It will be followed with edge definitions, one per edge in the graph. These take the form:

```
srcNode -> destNode [label = "labelGoesHere"]
```

For example:

```
0 -> 1 [label = "-silence-"]
```

3. It will always end with a single right curly brace (to close the one opened in step 1):
}

An example complete dot file for a simple lattice with only seven edges is given below:

```
digraph g {  
    rankdir="LR"  
    0 -> 1 [label = "-silence-"]  
    0 -> 2 [label = "-silence-"]  
    1 -> 3 [label = "-silence-"]  
    2 -> 4 [label = "-silence-"]  
    3 -> 5 [label = "ok"]  
    4 -> 5 [label = "okay"]  
    5 -> 6 [label = "-silence-"]  
}
```

Academic Honesty

Aside from your assigned partner, you must not share code with any of 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 need help from another pair, step away from the computer and *discuss* strategies and approaches, not code specifics. I am available for help during office hours as are the department tutors. I am also available via email (do not wait until the last minute to email). If you participate in academic dishonesty, you will fail the course.