CSC 207 **Assignment 6: Doublets Search** **Spring 2011**

Assigned: Tuesday 8 March 2011

Due: Friday 18 March 2011

Topics: Polymorphism, Collections, Algorithms, Analysis

Collaboration: This homework assignment may be completed individually or in pairs.

Submission: Follow the instructions for submitting programs via P-Web and handing in a printed copy. Be sure to generate unit tests and a statement justifying their sufficiency.

In this assignment, you'll continue working with the support structures from the previous assignment to implement a search for a chain of links between doublets and experiment with properties of the various chain managers.

You can find an implementation binary for the previous assignment on the MathLAN in `~weinman/courses/CSC207/code/doublets`. The file `doublets.jar` contains `Links`, `Chain`, `ChainManager`, `StackChainManager`, and `QueueChainManager`. Documentation for these may be found in the subdirectory `doc/`. It also contains a version of the interactive program you are to write (see usage below). The subdirectory `data/` contains pre-processed `Links` objects you can use (see below).

Doublet Chain

With all of the pieces in place from the previous assignment, you are now ready to *find* a chain of links between doublets. But how? A sketch of the algorithm is as follows.

To get the process started, the initial chain is quite simple: it should contain the start word. This is your current (first) candidate chain.

What should you do with a candidate? First, see whether the last word in the chain is the ending doublet. If it is, the candidate chain is in fact a solution. Otherwise, you'd need to get the allowable linking words of the candidate chain's last word. (This is what you built in the first part of the assignment.)

For each of these possible linking words, you'd tell the chain manager that you'd (later) like to consider a new chain (built on the candidate) that has the linking word at the end. Of course, that's only if the linking word is not already in the candidate chain.

Once you've done that, you'd ask the chain manager for the next chain to consider and repeat the process. If the chain manager says there's nothing left, you know there is no chain between the doublets.

Create a class called `Doublets` whose constructor takes a `Links` object. Add a member method `getChain` that takes the two doublet strings (starting and ending) and a `ChainManager`, returning a solution `Chain` or `null` if there is none. This method should implement the algorithm sketched above.

Interactive Program

Optional Prequel

Because creating the links is a high-cost startup operation, you may want to modify `Links` so that it is serializable. Then you may write the processed word list to disk and load the object directly on the fly. The given implementation in `doublets.jar` does this, e.g.,

```
java -Xmx1024M -cp doublets.jar Links english-words.10.txt english-words.10.links
```

and several files are already available (`english-words.*.links`) for you to load:

```
Links links = (Links)(new ObjectInputStream (new FileInputStream(fileName))).readObject();
```

Program

Use all of these pieces to create an interactive program (`main` in `Doublets`) that allows you to specify the word file at start up. It should repeatedly prompt the user for a doublet and provide the chain solution or an appropriate message if there is none. Report the solution length, the number of candidate chains examined, and the maximum size of the candidate pool. For example (only include the `-cp` argument if you intend to run my version):

```
$ java -Xmx1024M -cp doublets.jar Doublets english-words.35.links
Welcome to Doublets, a game of "verbal torture."
Enter starting word: flour
Enter ending word: bread
Enter chain manager (s: stack, q: queue): s

Chain: [flour, floor, flood, blood, bloom, gloom, groom, broom, brood, broad, bread]
Length: 11
Candidates: 16
Max size: 6

Enter starting word: wet
Enter ending word: dry
Enter chain manager (s: stack, q: queue): q

Chain: [wet, set, sat, say, day, dry]
Length: 6
Candidates: 82651
Max size: 847047

Enter starting word: oat
Enter ending word: rye
The word "oat" is not valid. Please try again.

Enter starting word: owner
Enter ending word: bribe
Enter chain manager (s: stack, q: queue): s

No doublet chain exists from owner to bribe.

Enter starting word: ^C
$
```

Analysis

Write a short paper examining and explaining the behavior of your program (or the provided implementation) on this problem. The audience for your writing is your peers in this class. While your essay should include all the hallmarks of good writing (e.g., paragraph cohesion and unity, transitions, clear language), you should be sure to incorporate answers to the following questions along the way:

- Assume every word has B candidate plays, and a doublet's chain has L links. What is the big-O space complexity (memory requirements) of a search using a stack? a queue? Report some examples that demonstrate this empirically.
- Can you empirically characterize/compare the time complexity? That is, how do the number of candidates examined compare between the stack-based and queue-based searches? Report examples.
- For a given doublet, how does the solution quality (e.g., chain length) compare between the two candidate managers? Report examples.

Practical Notes

The Java option `-cp` is short for "class path." Adding a Java ARchive (jar) file to the classpath allows all the classes in the archive to be used. To use both your own files *and* class files from the jar file, include the current working directory ahead of the jar file:

```
java -cp .:doublets.jar ...
```

It is certainly possible to use my `Links` class with the pre-processed files, but otherwise use all your own files.

The Java option `-Xmx` allows one to specify the maximum heap size available to the program. You will likely need to increase this from Java's paltry default in order to do actual searches with the given links files.

The given pre-processed link files contain real data and are very large. You are encouraged to create your own small word lists as you begin working on and testing your project.

Epilogue

Human nature being what it is, the allowable plays for Vanity Fair's published puzzle competitions required some constraining. Several months later Carroll published an addendum:

> The following Glossary is intended to contain all well-known English words ... which may be used in good Society, and which can serve as Links. It is not intended to be used as a source from which words may be obtained, but only as a test of their being admissible.
>
> That such a Glossary is needed may best be proved by quoting the following passage from 'Vanity Fair' of May 17, 1879, premising that all the strange words, here used, had actually occurred in Chains sent in by competitors:—
>
> "CHOKER humbly presents his compliments to the four thousand three hundred and seventeen (or thereabouts) indignant Doubleteers who has so strongly shent him, and pre to being stoaked in the spate of their wrath, asks for a fiver of minutes for reflection. CHOKER is in a state of complete pye. He feels that there must be a stent to the admission of spick words. He is quite unable to sweal the chaffy spelt, to sile the pory cole, or to swill a spate from a piny ait to the song of the spink. Frils and the mystic Gole are strangers in his sheal: the chanceful Gord hath never brought him gold, nor ever did a cate become his ain. The Doubleteers will no doubt spank him sore, with slick quotations and wild words of yore, will pour upon his head whole steres of steens and poods of spiles points downwards. But he trusts that those alone who habitually use such words as these in Good Society and whose discourse is universally there understood, will be the first to cast a stean at him."
>
> As the chief object being aimed at has been to furnish a puzzle which shall be an amusing mental occupation at *all* times, whether a dictionary is at hand or not, it has been sought to include in this Glossary only such words as most educated people carry in their memories. If any doubt should arise as to whether any word that suggests itself is an admissable one, it may be settled by referring to the Glossary.

Fortunately, your program has a fixed list of words from which to draw, so that you will not have to "swill a spate from a piny ait to the song of the spink."