CAS CS 112. Assignment 7

Due 11:59pm on Monday, December 12, 2011

In this assignment you will implement a program the solves the puzzle called "Doublets" proposed by Lewis Carroll in the English weekly *Vanity Fair*[1] (no relation to modern American *Vanity Fair*). The puzzle itself was announced in the paper as follows:

A NEW PUZZLE.

THE readers of *Vanity Fair* have during the last ten years shown so much interest in the Acrostics and Hard Cases which were first made the object of sustained competition for prizes in this journal, that it has been sought to invent for them an entirely new kind of puzzle, such as would interest them equally with those that have already been so successful. The subjoined letter from Mr. Lewis Carroll—a name already immortalised as that of the author of "Alice in Wonderland"—will explain itself, and will introduce a puzzle so entirely novel and withal as interesting, that the transmutation of the original into the final word of the Doublets may be expected to become an occupation to the full as amusing as the guessing of the Double Acrostics has already proved.

In order to enable readers to become acquainted with the new puzzle, preliminary Doublets will be given during the next three weeks—that is to say, in the present number of *Vanity Fair* and in those of the 5th and 12th April. A competition will then be opened—beginning with the Doublets published on the 19th April, and including all those published subsequently up to and including the number of the 26th July—for three prizes, consisting respectively of a Proof Album for the first and of Ordinary Albums for the second and third prizes.

The rule of scoring will be as follows:—A number of marks will be apportioned to each Doublet equal to the number of letters in the two words given. For example, in the instance given below of "Head" and "Tail," the number of possible marks to be gained would be eight; and this maximum will be gained by each one of those who make the chain with the least possible number of changes. If it be assumed that in this instance the chain cannot be completed with less than the four links given, then those who complete it with four links only will receive eight marks, while a mark will be deducted for every extra link used beyond four. Any competitor, therefore, using five links would score seven marks, any competitor using eight links would score four, and any using twelve links or more would score nothing. The marks gained by each competitor will be published each week.

In order to afford space for the Doublets the publication of Hard Cases will be discontinued from and after the 17th April.

[1] *Vanity Fair*, vol. XXI, no. 543, 29 March 1879, pp. 185–186

"DOUBLETS"—A VERBAL PUZZLE.

DEAR VANITY,—Just a year ago last Christmas, two young ladies—smarting under that sorest scourge of feminine humanity, the having "nothing to do"—besought me to send them "some riddles." But riddles I had none at hand, and therefore set myself to devise some other form of verbal torture which should serve the same purpose. The result of my meditations was a new kind of Puzzle—new at least to me—which, now that it has been fairly tested by a year's experience and commended by many friends, I offer to you, as a newly-gathered nut, to be cracked by the omnivorous teeth which have already masticated so many of your Double Acrostics.

The rules of the Puzzle are simple enough. Two words are proposed, of the same length; and the Puzzle consists in linking these together by interposing other words, each of which shall differ from the next word *in one letter only.* That is to say, one letter may be changed in one of the given words, then one letter in the word so obtained, and so on, till we arrive at the other given word. The letters must not be interchanged among themselves, but each must keep to its own place. As an example, the word "head" may be changed into "tail" by interposing the words "heal, teal, tell, tall." I call the two given words "a Doublet," the interposed words "Links," and the entire series "a Chain," of which I here append an example :—

H E A D
h e a l
t e a l
t e l l
t a l l
T A I L

It is, perhaps, needless to state that it is *de rigueur* that the links should be English words (including well-known names), such as might be used in good society.

The easiest "Doublets" are those in which the consonants in one word answer to consonants in the other, and the vowels to vowels; "head" and "tail" are a Doublet of this kind. Where

this is not the case, as in "head" and "bare," the first thing to be done is to transform one member of the Doublet into a word whose consonants and vowels shall answer to those in the other member (*e.g.*, "head, herd, here"), after which there is seldom much difficulty in completing the "Chain."

I am told that there is an American game involving a similar principle. I have never seen it, and can only say of its inventors, "*pereant qui ante nos nostra dixerunt!*"

LEWIS CARROLL.

DOUBLETS.

1. Drive PIG into STY.
2. Raise FOUR to FIVE.
3. Make WHEAT into BREAD.

No answer can be acknowledged unless it be received at "Vanity Fair" Office by twelve o'clock at noon next Thursday.

During the first few weeks of the competition administered by *Vanity Fair*, there was much debate on its pages about which words are acceptable—for instance, whether words such as "spank" and "hell" were words that "might be used in good society." Ultimately, Lewis Carroll published a glossary of acceptable words. Unfortunately, I was unable to find a typed-up version of it; therefore, we will stick to using the same Scrabble word list as for assignment 6 (you already have it). This makes it easier to find links, as the word list is quite a bit larger than Carroll's dictionary (and, in particular, contains the words "spank" and "hell").

We highly recommend debugging on much smaller dictionaries that you write yourself. It's not a bad idea to start with a 2-word dictionary, in fact.

The approach you will take is as follows. First, construct an (undirected) graph of all the words in the dictionary, with an edge between two words if and only if they differ in exactly one character. Do so by reading in one word at a time, and figuring out all of its edges to the previously read words, and all of their edges to it.

After the graph is constructed, searching for the shortest chain given a doublet is a matter of finding the vertices corresponding to two words in the doublet, and then running a breadth-first-search. You will have to implement the breadth-first search method to do so; the method should return the stack of nodes representing the chain.

You will be graded not only on correctness, but also on efficiency of your solution. In fact, if you are not careful, your code will be too slow to be useful.

There are at least two possible approaches (feel free to use your own):

1. Have `WordGraph.Vertex` contain all the information that a Vertex logically contains. Specifically, have it store its word (as a `char` array), a linked list of its neighbors (which are themselves also of type `WordGraph.Vertex`), and perhaps even a `boolean` flag indicating whether it has been visited and a vertex indicating its predecessor in the breadth-first-search. Store a graph as an array of such vertices, in alphabetical order. In this approach, you don't use the number of the vertex much, since vertices point to other vertices directly as their neighbors.

   Use the fact that the words are already sorted in the dictionary: this makes it easy to search for neighbors during graph construction by using binary search; moreover, you know that the word you read in is alphabetically *after* all the words that have already been added to the graph,

which helps cut down the search.

If you store the visited flag inside each vertex, be sure to include proper clean-up at the end of your breadth-first search to clear the "visited" flag of every vertex that had it set, in order to prepare for the next breadth-first search.

In this approach, all your code would go into into a single file `WordGraph.java`.

2. Alternatively, you may use the more modular approach that the book advocates. In this approach, you can use directly (with proper attribution, of course!) a lot of code from the book — in fact, you would be able to reuse entire classes. Note that the book treats vertices as integers. Since your vertices are words, your `WordGraph` would contain (a) a graph whose vertices are integers, from the book; (b) an array that contains the word corresponding to each integer, an alphabetical order (easy to do, since the dictionary already comes in alphabetical order). In that case, `WordGraph.Vertex` needs to contain only the integer (however, its public methods, such as `toString`, would need to find information corresponding to that integer).

This is likely your most complicated programming project so far. We provide two files: `DoubletsGUI.java` (which you should not modify) and `WordGraph.java` (which is mostly a skeleton). Read and understand them before proceeding.

Optional, not for credit, only if you have time. When you add each word, you will have to perform a binary search for all of its one-letter-substituted variations. You can speed up that search if you search for the variation in alphabetical order, make your binary search method return the insertion point location whether or not the variation was found, and start off the search for the next variation at that location rather than at the beginning of the array. To make binary search do this and yet at the same time tell you whether the variation was found or not, the trick is to have it return the usual index of the variation if the variation was found, and the value $(-\texttt{insertion\_point} - 1)$ if it wasn't, where `insertion_point` is the location where you would insert the variation in the array if you were to place it there. In my implementation, this sped up graph construction by about a factor of 2. I provide a javadoc for this optional binary search in `BetterBinSearch.java`.

You can also speed up loading by having separate instances of the graph class for each word length, and adding each word to the appropriate instance.

You could also try to study the graph. Interesting questions to ask include the following. How many connected components, and of what sizes, at each length? What is the average degree? What is the maximum degree? How many words have no neighbors at all?