CAS CS 112 – Spring 2012, Assignment 5

Due at 10:00 pm on Thursday, April 12

Anagram Solver

In this assignment you will implement a program which prints out all anagrams of a specified string. Two strings are anagrams of one another if by rearranging letters of one of the strings you can obtain another. For example, the string "toxic" is an anagram of "ioxct". For the purpose of this assignment, we are only interested in those anagrams of a given string which appear in the dictionary. The dictionary you will use is the Tournament Word List (TWL), a list of 178,691 words used by Scrabble players.

Algorithm and Implementation

Since we will be performing multiple anagram queries, our first step is to load all of the words in the TWL into an appropriate data structure. A primary requirement is that one must be able to efficiently search this data structure to look for anagrams of a given word. A clever trick that we will use to facilitate this is to work with *alphagrams* of words, not just the words themselves. The alphagram of a word (or any group of letters) consists of those letters arranged in alphabetical order, and is the way expert Scrabble players sort the tiles on their rack. So, the alphagram for the string "toxic" is "ciotx", similarly the alphagram for both "star" and "rats" is "arst".

Now what you will do is store words and their alphagrams into a hash table. For each word, you will compute its alphagram. Treat the alphagram of a word as the key of the <Key, Value> pairs stored in the hash table. For the value that is stored in the hash table, you will have to be a little bit creative: since every alphagram may be associated with many words, you need to reflect this in your design – you ideally want to retrieve all anagrams with one call to *get*(). Whichever way you decide to go, you should define a new Anagram class that contains all the relevant information about a group of Anagrams: the common alphagram, the words that share that alphagram, plus a hashCode and anything else you might need. Hashing on alphagrams guarantees that all words which are anagrams of one another are stored together, since those words have common alphagrams. You should feel free to use the methods described in class and in the text for appropriate hash functions for hashing strings (but please cite any source which you use).

Once the entire TWL has been stored in a hash table, it is time to solve anagram queries. Prompt the user to entire a string, and output all of the anagrams of that string, or a message if none are found. To find the anagrams, you will have to compute the alphagram of the input string, hash it, and scan the corresponding bucket to find matching words. Note that not everything in the bucket is necessarily a match: you will have to check whether keys match.

Submit three files:

1. Anagram.java – an Anagram class as described above.
2. HashTable.java – hash table with separate chaining (we strongly suggest using the textbook code without modification).
3. Client.java – a main program which loads words from the dictionary into the hashtable and then answers anagram queries.

Additional Details

You may fix a size for your hash table – for efficient searching, I recommend that the final hash table you submit contain around 100,000 buckets. (For debugging your code, I suggest you work with a much smaller practice dictionary, perhaps 10 words, and a much smaller hash table, perhaps 8 buckets).

If you design this correctly, you should be able to re-use the hash table implementation from the book without modification, and have a very thin Client. Take as much as time as needed to whiteboard the design as elegantly as possible, and then code it up in a short session.