For Lab 4, you implemented an algorithm to find all the anagrams of a word and to find the word with the most anagrams in the English language. The algorithm was time consuming; for a vocabulary size $v$, it took $O(v \log v)$ to build the B-Tree to store the words, and then to find the anagram of a word $w$ of length $n$ it took $O(n! \log v)$, since we had to test all the permutations of the characters of $w$. Finding the word with the most anagrams took $O(n! v \log v)$.

For this lab we will solve the same problem using a much more efficient data structure. We will write a program that will find all the anagrams of every word in the English language and store them in a hash table in time $O(v)$. Given a word, its anagrams can be found in time $O(1)$ by accessing the table and the word with the most anagrams can be found in time $O(v)$ by a table traversal.

We will exploit the observation that word $w_i$ is an anagram of word $w_j$ if and only if the sorted characters of $w_i$ and $w_j$ are identical. For example, if $w_i = "stealer"$ and $w_j = "relates"$, then $\text{sorted}(w_i) = \text{sorted}(w_j) = "aeelrst"$, so $w_i$ is an anagram of $w_j$ (and vice versa).

To store the anagrams, we will use a hash table with chaining using the words with sorted characters as keys. Each node in the hash table will contain, in addition to the key, a reference to a list of anagrams. See the powerpoint presentation in the class website for additional explanation and an example.

Your task consists of writing a program (or programs) to do the following:

1. Read the words in the English dictionary and store them in a hash table with chaining, organizing them by groups of anagrams, as discussed above.
2. Write a method that prompts the user to type a word and then prints all the anagrams of that word.
3. Write a method that finds the word with the most anagrams in the vocabulary. Notice that this can be done by traversing the hash table; you don’t need to read the dictionary file again.
4. Repeat questions 1 to 3, but now use a hash table with linear probing. In this case, your $\text{SortedStringNode}$ (as described in the slides) does not have a $\text{next}$ field (but it still contains the $\text{anagrams}$ field, a reference to the list of anagrams). If a sorted string hashes to a location in the table that is already taken, you simply search sequentially for the next available location (wrapping around at the end the of the array).

Compare the times it takes to find the word with the most anagrams with both of your implementations, experimenting with various table sizes. As usual, write a report describing your work as specified in the syllabus.