For Lab 3, you wrote code to generate all the anagrams of a given word. Your program generated all the permutations of the characters in the original word and then for each permutation it determined if it was a word in the English language. In order to check whether a word was part of the English language or not, you used a HashSet called `englishWordsSet`. This HashSet was populated with the contents of `words.txt`, which you downloaded from the following GitHub repository: https://github.com/dwyl/english-words/

In the previous lab, when a character appeared more than once in a word, the program would print repeated words as output. To prevent this from happening, once a word is printed, you should remove it from your table. Thus the original code would be modified as follows:

```java
public static void printAnagrams(String prefix, String word, HashSet<String> englishWordsSet){
    if(word.length() <= 1) {
        String str = prefix + word;
        if (englishWordsSet.contains(str)){
            System.out.println(str);
            englishWordsSet.remove(str);
        }
    }
    else {
        for(int i = 0; i < word.length(); i++) {
            String cur = word.substring(i, i + 1);
            String before = word.substring(0, i); // letters before cur
            String after = word.substring(i + 1); // letters after cur
            printAnagrams(prefix + cur, before + after,englishWordsSet);
        }
    }
}
```

In this lab, you will modify the method `printAnagrams`, so that it uses your own implementation of a hash table instead of Java’s HashSet.

To solve collisions, you are required to implement both techniques covered in class: chaining and linear probing. At the beginning of every run, your program must ask the user which technique must be used. You must write multiple hash functions, and test them with the two types of collision-solving methods. Compare the running times of both of your own implementations and the HashSet provided in Java. Can you come up with a faster implementation than the one built-in?

Also, write methods to determine the average number of comparisons required to perform a successful retrieve operation for every implementation. If you used a balanced search tree, you would need, on average, $\log_2(354,984) \approx 18$ comparisons, so for your hash table to be successful you’d need significantly fewer than that (in the ideal case, you’d need exactly 1 comparison per access).

As usual, write a report describing your results.

Hint 1: The choice of table size and hashing function has a large impact in the number of collisions and thus running time.

Hint 2: There are 354,984 English words in `words.txt`

Hint 3: You can think of a word in the English language as a base-26 number.