Programming Assignment 3:
You Can Address Me As Eagle One

Due date: October 30 at the start of lecture.

Overview

In this lab, you will implement the Set ADT as a hash table with open addressing.

Hash Table Design

You will implement a HashSet class using notes from lecture and features appropriate to the language of your choice. General implementation details:

1. The table will use open addressing for collisions. The size of the table will **always be a power of 2**, and you will use the probing function $f(i) = \frac{i^2+i}{2}$.
2. Implement the *add*, *remove*, and *find* ADT operations.
   (a) Add: given a value, insert the value the hash table if it is not already present.
      i. **Prior to doing the insertion**, you must check to see if the **load level** (alpha) of the set is above `0.8`. If it is, you must first **resize the entire table**: create a new array that is twice as long as currently, then for each entry currently in the set, **rehash that entry** into its new location in the new array. (This index will change because m will have changed.)

   (b) Remove: given a value, remove the value from the hash table. You must replace the entry in the table with a special "NIL" / "deleted entry" value.

   (c) Find: given a value, return true iff the value exists in the set. "Finding" (and the "if not already present" stipulation for *add*) requires two comparisons: first, compute $h(k)$ for the value and go to that table location; compare the hashtable entry that you find there for *equality* with the parameter you are "find"ing. If they are equal, then return true! Otherwise, *repeatedly* use the **probing function** to find the next location where the value should be, until you either:
      i. find a hashtable entry that is equal to the parameter.
      ii. find an empty entry (**not** NIL), in which case you know the value is not in the set.
      iii. fail m times total – since our probing function has a period m (**see Group Assignment 3**), this indicates that the value is not in the set.
3. Implement these auxiliary methods:
   (a) `int count()`: return the number of values in the table, **not** the length of the table.
   (b) `double loadFactor()`: return the current load factor of the table.
4. The class constructor should take a parameter specifying how large the table's array should be. However, you cannot trust the user to construct the table with a power of 2 size, so **your constructor must "round" the given size to the smallest power of 2 that is greater than or equal to the requested size**.
5. In order to support the concept of "NIL", your hashtable cannot be an array of the same type of data you are storing. (For example, if you are making a HashSet of strings, the hashtable array cannot be a `String[]` variable.) This is because most languages don't have a way of distinguishing "null" (no value) from "NIL" (a value used to exist but was removed), so we need to fake it in our implementation. You'll want to make a private helper class called `Entry` that has two fields: one to store the value being placed into the set, and a boolean flag for whether that entry has been removed. You will examine these fields when performing the add and remove routines described prior.

There are two starter files for this assignment on BeachBoard, one for Java and one for C++. You can ask for help translating them if you are using a different language, but those two would be most appropriate for this assignment.

Program

Once your hash table is implemented (**and tested**), use your class in the following program:

1. Download the file `trump_speech.txt` from BeachBoard's Lab content. This file contains a speech given by President Donald Trump during his 2016 presidential campaign.
2. Construct a HashSet with an initial size of **50**.
3. Open the `trump_speech` file as a **text file** and repeatedly:
   (a) Read one word from the file.
   (b) **Remove** all non-alphanumeric characters from the word. A non-alphanumeric character is any character other than the lowercase and uppercase English letters, and the numerals 0 through 9.
   (c) Add the modified word to the hash set.
4. Once you have read all words from the file, **output** the total number of distinct words in the hash set.

Getting Started

I have uploaded starter files for Java and C++ to BeachBoard. You are welcome to use these files to begin your assignment. They comply with the standards outlined above, except they are missing a few functions that you will have to fill in.

Deliverables

Turn in:

1. A printed copy of your program's code.
2. The output from your program.