CSCI241 Spring 2020: Lab 7
Due Sunday, May 24th at 9:59pm

Overview

In this lab, you'll learn a little more about hash functions, including properties they must have, properties that are nice to have, and how hashing is done in Java.

Goals

This lab presents new material that was not discussed in lecture but you may be asked about on quizzes and/or exams. Here are the goals for this lab:

- Know what properties a function must have in order to be a **valid** hash function, and why.
- Know what properties are **nice** to have in a hash function, and why.
- Understand the contract of the java `Object` class's `equals` and `hashCode` methods.

1 Definitions

For purposes of this lab, a hash function is a function that maps a value to an integer. Note that we've excluded the modulus from this definition: someone (e.g., you) implementing hash table can take the integer returned by a hash function and calculate the modulus with their table size. This definition matches the way Java defines its `hashCode` method.

2 Hash Functions: Necessary Properties

To be a **valid** hash function, the hash function should satisfy the following properties:

1. Calling the hash function on the same value twice must return the same value.
2. Calling the hash function on two equal values must return the same value.
3. It is **not** necessarily the case that calling the hash function on two different objects returns **different** hash values. In other words, hash **collisions** are possible.

The reason for the first necessary property is simple: if you use a hash value to decide where (e.g., in which bucket) to store a value, then you need to be able to get back there again to retrieve it.

The second property is important for maintaining a well-defined set or mapping. In a Set, duplicate values are not allowed; in a Map, duplicate keys are not allowed. If a value is added to a HashSet, you need the hash function to send you to the correct bucket where its already-existing duplicate would exist, if it does, or you risk adding a duplicate value; similar reasoning applies to HashMaps with duplicate keys.

It is not reasonable to insist that different values have different hash values. There are only about 4.3 billion `int` values, and many types may have more (indeed, infinite) possible values. For example, there are about 8 billion strings containing 7 lower-case letters. Clearly, we can't assign a unique integer to every possible String.

3 Hash Functions: Nice Properties

In addition to the necessary properties above, it is **nice** if a hash function also has the following properties:

1. Fast (e.g., constant time).
2. As much as possible, different values have different hash values.

Speed is important because all of the runtime benefits of hashing hinge on the efficiency of finding the correct bucket where a value must be stored.

Although we can't count on it, a nice well-behaved hash function does a good job of spreading values out among many buckets, thereby minimizing the number of collisions. As we've seen in lecture, collisions are the cause of all the worst-case inefficiencies of hash tables. Fewer collisions means better practical runtime.

4 `hashCode` and `equals`

Hashing in java abides by the above principles. The Object class, from which all reference types inherit, has methods `equals` and `hashCode`. The `equals` method is Java's way of determining what is meant by "the same value"; the `hashCode` method implements a hash function for any type.

Read the specification for the `equals` method, then read the spec for the `hashCode` method. Notice that the `hashCode`'s contract closely resembles the rules outlined in Section 1.

The key thing to keep in mind when writing classes in Java is that **if your class overrides `equals`**, then there's a good chance that your `hashCode` method (still the version inherited from `Object`) violates the contract by allowing two values that are equal according to `equals` to hash to different values. If this is the case, your class needs to also override `hashCode` in order to avoid breaking the contract of the `hashCode` method's contract.

5 Questions

Instead of writing code for this lab, you will answer some written questions. Download `lab7_questions.txt` from the Lab 7 Canvas assignment and fill in the space below each question with your answer. Upload your submission to Canvas.