CMPSCI 187, Spring 2015 Discussion #13: Hash Functions: Individual Handout

Marc Liberatore and John Ridgway

27 April 2015

Remember that every student gets a copy of this handout, but every pair still hands in one response, on a separate answer sheet provided. Do not hand in this paper.

In lecture tomorrow we will be discussing hash-tables. This data structure is crucially dependent on what is known as a *hash function*. A hash function is a function that takes some value as an argument, and returns an integer that is somehow related to that number. The one absolute requirement on a hash function is that two values that are to be considered equal must hash to the same value. There is also a desirable characteristic: that the hash function return distinct integers for each of the values it is likely to be used on.

In Java there is a method in `Object` (and thus inherited by every other class) called `hashCode()` which takes no arguments and returns an **`int`**. The following is a quote from the Java SE 7 API:

> The general contract of `hashCode` is:
>
> - Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
> - If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.
> - It is not required that if two objects are unequal according to the `equals(java.lang.Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

The default implementation of `hashCode` is to return a value derived from the actual address of the object, thus returning a different value for every object.

Question 1: Here is a possible hash function. Does it meet the absolute requirement? Does it have the desirable characteristic?

```java
public int hashCode() {
  return 1;
}
```

Question 2: For this question assume that we are writing a `hashCode()` method for a class similar to the `String` class. Its sole attribute `characters` is an array of **`char`** that contains the characters making up the string.

One possible hash function would be to simply add the characters together, as follows:

```java
public hashCode() {
  int result = 0;
  for (char c : characters) {
    result += c;
  }
  return result;
}
```

(and yes, this works; characters can be added and subtracted and converted to integers).

State a potential problem with this. *Hint: think about strings with the same letters in a different order.*

Question 3: Considering the method in the previous question: Do you think it would be a good idea to only consider the first five characters so as to reduce the amount of computation?

Question 4: As before, assume that we are writing a `hashCode()` method for a class similar to the `String` class. Suppose we know that the the input to this string will always be a URL representing a web site served by HTTP, located on the server `www.cs.umass.edu`. How might you improve the runtime of the previous hash function?

Question 5: As before, assume that we are writing a `hashCode()` method for a class similar to the `String` class. Further suppose that we've initialized a `Random` object as an attribute named `random` in this class.

```java
public hashCode() {
  int result = random.nextInt();
  for (char c : characters) {
    result = 31 * result + c;
  }
  return result;
}
```

Is this function more or less likely to return a unique value for each input than our previous attempt? And, is it a valid implementation of `hashCode`? Why or why not?