19.3 "Valid" & "Good" Hashcodes

Professor Hug's Lecture on Valid/Good Hashcodes

Valid Hashcodes!

You may see this term in discussions and potentially on exams. What exactly makes a hash code "valid"? There are two properties:

1. ***Deterministic***: The `hashCode()` function of two objects A and B who are equal to each other (`A.equals(B) == true`) have the same hashcode. *This also means the hash function cannot rely on attributes of the object that are not reflected in the* `.equals()` *method.*
2. ***Consistent***: The `hashCode()` function returns the same integer every time it is called on the same instance of an object. This means the `hashCode()` function must be independent of time/stopwatches, random number generators, or any methods that

would not give us a consistent `hashCode()` across multiple `hashCode()` function calls on the same object instance.

Note that there are no requirements that state that unequal objects should have different hash function values.

One could argue that these two requirements are in fact the same requirement. We can restate the requirement of consistency. Imagine we make a pointer named `A` to an object `O` at 12:00 pm and a pointer named `B` to this same object `O` at 1:00 pm. We know that the hash code should return the same integer for both objects, due to the consistency requirement. However, how do we formally define our statement "this same object `O`" above? Technically, the only reason we consider `B` to be pointing to the same thing as `A` is because of the `.equals()` method! This is starting to sound a lot like the determinism requirement.

Good Hashcodes!

You'll probably see this term a lot as well. But what makes a hashcode "good"? There are a few properties that can make a good `hashCode()`:

1. The `hashCode()` function must be valid.
2. The `hashCode()` function values should be spread as uniformly as possible over the set of all integers.
3. The `hashCode()` function should be relatively quick to compute [ideally O(1) constant time mathematical operations]

Now let's think more specifically about the impact of the hashing function. In general, we assume most hash functions will be "relatively quick". Why do we make this assumption? Given how intrinsic the hashing function is to our data structure, the runtime of this function will have a significant effect on the overall runtime of our data structure. This means we want our hash code to be "easily" computable (ideally constant time), so that we may maintain the O(1) runtime characteristic that makes hashing so special and efficient!

Previous
19.2 Hash Code

Next
19.4 Handling Collisions: Linear Probing and External Chaining

Last updated 1 year ago