20.2 Distribution By Other Hash Functions

HashMaps/Tables have fast lookup times, but behind that "superpower" is a hash function.

CS61B - Hashing 2 - Video 2 - Distribution by Other Hash Functions (and Ch...

Distribution by other Hash Functions

Suppose our `hashCode()` implementation simply returns 0.

```
@Override
public int hashCode() {
    return 0;
}
```

› What distribution do we expect?

In order to get a more even distribution, what we can do is something to what we tried in the previous section where we utilize modulo. Let's say that we define the size of our Hash Table to have 4 buckets. This means it has 4 corresponding LinkedLists and 4 bucket indices labeled {0, 1, 2, 3}. The modding is not required in our `hashCode()` function as it is being done for us in the hash table to guarantee we can add to that bucket. As we said the hash code could really be any integer in the range of 4 billion unique values!

By using modulo, we ensure that our hashcode yields a number that can be represented as an index and clearly identifies which LinkedList to add to. Additionally, when adding a series of numbers at once, we see that we get an even distribution of numbers in our LinkedList yielding a constant lookup time.

```
@Override
public int hashCode() {
    return num;
}
```

This hash function should yield a much more even distribution! Objects with different `num` will now be more spread out across the buckets instead of all living in the 0th bucket. If our class does not explicitly override the `hashCode()` function, Java will use the default implementation, which returns the object's address in memory as its hash code!

Why Bother With Custom Hash Functions?

CS61B - Hashing 2 - Video 3 - Why Bother With Custom Hash Functions?

Let's discuss if the default hashCode function is a good hashCode function! It actually is a good spread as it relies on the fact that different objects will live in different places in the memory, and the memory address is effectively random. We will get a good distribution, since objects are basically assigned random indices to insert into the hash table.

This really raises an interesting question: why do we care about other custom hash functions when the default hashcode gets good spread? We'll read about this in the next section.

Previous
20.1 Hash Table Recap, Default Hash Function

Next
20.3 Contains & Duplicate Items

Last updated 1 year ago