CSCI 241

Scott Wehrwein

Hashing non-integers
Hashing in Java

Goals

Know how to hash types other than integers (e.g., Strings)

Know how hashing is implemented in Java via `Object`'s `hashCode` method.

Hash function thus far:

```
h(x) = x % C
```

How do we hash values that aren't integers?

Can we hash values that aren't integers?

Sure! Here's a proof:

- If it's a value in memory, it's encoded in binary.
- Interpret the bits as an integer and hash as before.

Often do it some other way, but the theme is: find a way to convert your type to an integer, then mod.

Example: Multiple Integers

Hash a tuple of integers (a, b, c, d):

- h((a, b, c, d)) = (a + b + c + d) % N
- h((a, b, c, d)) = (ak^1 + bk^2+ ck^3+ dk^4) % N

for some constant k

Example: `Strings`

Convert each character to its integer character code (ASCII or unicode).

You now have a tuple of integers.

Java's `String` uses:

```
s[0]*31^(n-1) + s[1]*31^(n-2)+ … +s[n-1]
```

Hashing in Java

Scenario 1: You are using a class that someone else wrote.

- Object has a `hashCode` method.
  Unless overridden, this returns the object's address in memory.
- That class inherits from `Object`.
- You don't need to know how to hash it: just call its `hashCode` method.
- Detail: `hashCode` returns an integer. You'll need to mod it by your table's size.

Hashing in Java

Scenario 1: You are using a class that someone else wrote.

- Object has a `hashCode` method.
  Unless overridden, this returns the object's address in memory.
- That class inherits from `Object`.
- You don't need to know how to hash it: just call its **`hashCode`** method.
- Detail: **`hashCode`** returns an integer. You'll need to mod it by your table's size.

Hashing in Java

Scenario 2: You are writing a class.

- Object has a `hashCode` method.

  Unless overridden, this returns the object's address in memory.

- Your class inherits from `Object`.
- You may override `hashCode`
- You may **need** to override `hashCode` if you're overriding `equals`.

More on this in Lab