CS 310: Hash Functions and Tables

Chris Kauffman

Week 6-1
Logistics

Reminder Career Fair
- October Wed 7 (Sci/Eng) and Thu 8 (Humanities)
- 11am-4pm Johnson Center
- Info for students here
- Workshops: list is here

HW 2
- Finishing up AdditiveList?
- Due in 9 days
- Test Cases available soon

Reading
- Weiss Ch 20: Hash Table
- Weiss Ch 6.7-8: Maps and Sets

Goals Today
- Hash Functions
- Collision Resolution
- Separate Chaining In Hash Tables
Every Object’s Doin’ it... but not well

Every object in java has a `hashCode()` method

▶ Why?
▶ How are hash codes computed by default?
▶ Official Docs

Override `hashCode()`

▶ For your own classes, override default `hashCode()`
▶ Compute hash based on the internal data of an object
▶ Return an integer "representing" the object
▶ Class is now "hashable"

Question
What is the *Hash Contract* to which all `hashCode()` implementations should adhere?
Hash Functions

Hash Contract

- If \( x \) equals \( y \) is true, then \( x.hashCode() == y.hashCode() \)
- Equal object → Same hash code
- If \( x \) equals \( y \) is false, hash codes may be different or the same

Goals of a Hash Function

- If \( x \) and \( y \) are equal, \textbf{must} have same hash code
- If \( x \) and \( y \) not equal, try to make \( x.hashCode() \) different from \( y.hashCode() \)
- Making hash codes different reduces collisions in hash tables
- Compute \( x.hashCode() \) as quickly as possible
First Example: String.hashCode()

From String javadocs for public int hashCode()

Returns a hash code for this string. The hash code for a String object is computed as

\[ s[0]*31^{(n-1)} + s[1]*31^{(n-2)} + \ldots + s[n-1] \]

using int arithmetic, where \( s[i] \) is the \( i \)th character of the string, \( n \) is the length of the string, and \(^\ldots\) indicates exponentiation.

Examples

Welcome to DrJava.

```java
> "a".hashCode()   > String s = "Hash!";
97                > s.hashCode()
> "b".hashCode()   69497011
98                > (31*31*31*31)\*'H' + (31*31*31)\*'a' +
> "ab".hashCode()  > (31*31)\*'s' + (31)\*'h' + '!' 69497011
3105
> "ba".hashCode()  69497011
3135
```
Consider `String.hashCode()`

The hash code of a string `s` is computed as

\[ s[0] \times 31^{(n-1)} + s[1] \times 31^{(n-2)} + \ldots + s[n-1] \]

using int arithmetic, where `s[i]` is the `i`th character of the string, `n` is the length of the string, and `\^` indicates exponentiation. (The hash value of the empty string is zero.)

Practice: Discuss the Following

- Is this what you expected for string?
- Is 31 special?
- **Write code** for `String`’s `hashCode()` method. In Java.
- Complexity of code?
- Optimizations?
- Alternative hash functions for strings?
Polynomial Hash Code Tricks

String uses a polynomial hash code

\[ a_0 X^{n-1} + a_1 X^{n-2} + a_2 X^{n-3} + \cdots + a_{n-1} X^0 \]

31 is \( X \) in the above

- 31 is not special
- Early java used 37 instead

A Trick

Can regroup a polynomial of any degree

Example of regrouping degree 3 polynomial

\[ a_0 X^3 + a_1 X^2 + a_2 X^1 + a_3 \]

regrouped becomes

\[ (((a_0)X + a_1)X + a_2)X + a_3 \]
Implementations

**Slow: Original**

\[ s[0] \cdot 31^{(n-1)} + s[1] \cdot 31^{(n-2)} + \ldots + s[n-1] \]

```java
public int hashCode() {
    char s[] = this.value;
    int h = 0, i;
    for(i=s.length-1; i>=0; i--){
        h += s[i] * ((int) Math.pow(31, i));
    }
    return h;
}
```

**Faster: Exploit Regrouping**

\[ \ldots((s[0]) \cdot 31 + s[1]) \cdot 31 + s[2]) \cdot 31 + \ldots) \]

```java
public int hashCode() {
    char s[] = this.value;
    int h = 0, i;
    for (i=0; i<s.length; i++){
        h = 31 * h + s[i];
    }
    return h;
}
```

Examine parens carefully in expression
The Full Implementation uses Caching

Compute once, save for later

class String{
    private char[] str; // Chars of string
    private int hash;   // Default to 0

    public int hashCode() {
        // Check if the hash has already been computed
        if(this.hash!=0 || this.str.length==0){
            return this.hash;
        }
        // Hasn’t been computed, compute and store
        for(int i=0; i < this.str.length; i++) {
            this.hash = 31 * this.hash + this.str[i];
        }
        return this.hash;
    }
}

Not exactly how java.util.String looks but it’s the general idea
public int hashCode()

Ideas for `hashCode()` implementation of the following things

<table>
<thead>
<tr>
<th>Fundamental Types</th>
<th>Container Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Integer</td>
<td>▶ Integer []</td>
</tr>
<tr>
<td>▶ Long</td>
<td>▶ Double []</td>
</tr>
<tr>
<td>▶ Character</td>
<td>▶ String []</td>
</tr>
<tr>
<td>▶ Boolean</td>
<td>▶ ArrayList&lt;T&gt;</td>
</tr>
<tr>
<td>▶ Float</td>
<td>▶ LinkedList&lt;T&gt;</td>
</tr>
</tbody>
</table>
| ▶ Double                     | ▶ class Flurb{
|                              |     int x;
|                              |     double y;
|                              |     String s;
|                              | } |