Lecture 29: Maps & Hashing
Fall 2016
Kim Bruce & Peter Mawhorter

Midterm #2
- In class next Friday, 11-31
- Will cover material through parallelism
- Questions will focus on material since midterm #1:
  - Array representations of trees
  - Heaps & Heapsort
  - Binary search trees
  - Balanced binary search trees
  - Parallelism
  - Concurrency

Hackathon!
- Announcement near the end of class today.

Darwin Contest Results
Section 1

<table>
<thead>
<tr>
<th>Student</th>
<th>Survivors:</th>
<th>Percentage:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dylan Keezell</td>
<td>60514 / 72800</td>
<td>83.1%</td>
</tr>
<tr>
<td>Julian DeGroot-Lutzner</td>
<td>42853 / 72800</td>
<td>58.9%</td>
</tr>
<tr>
<td>Rex Bodoia</td>
<td>35229 / 72800</td>
<td>48.4%</td>
</tr>
</tbody>
</table>

Honorable Mention:
Victor deFontnouvelle (47.5%), Brady DeMeritt (47.4%), Wentao Guo (45.6%)
Darwin Contest Results

Section 2

<table>
<thead>
<tr>
<th>Student</th>
<th>Survivors</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Somtochukwu Uzoegwu</td>
<td>89798 / 108800</td>
<td>82.5%</td>
</tr>
<tr>
<td>Dominic Frempong</td>
<td>89348 / 108800</td>
<td>82.1%</td>
</tr>
<tr>
<td>Claire Genre</td>
<td>65386 / 108800</td>
<td>60.1%</td>
</tr>
</tbody>
</table>

Honorable mention:
Victor Machado (59.5%), Eberto Ruiz (58.3%), Brook Solomon (57.1%)

Maps

- Store and retrieve data based on a key.
  - Store phone numbers by name.
  - Store word pair frequencies by first word.
  - Store account info by user ID.
- At most one value per key
  (matches the mathematical concept).
- Also known as “dictionaries” or “hash tables.”

Interface

```java
public interface Map<K, V> {
    int size();
    V get(Object key);
    V put(K key, V value);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    Set<K> keySet();
    Collection<V> values();
}
```

```java
public interface Map<K, V> {
    int size();
    V get(Object key);
    V put(K key, V value);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    Set<K> keySet();
    Collection<V> values();
}
```
### Implementations

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>get</th>
<th>set</th>
<th>remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>sorted list</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>balanced BST</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>array[“key range”]</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

[hp://bigocheatsheet.com/](http://bigocheatsheet.com/)

### Problem

Goal: Array-like performance for all keys

Problems:
- Keys are not integers
  (and there is no obvious way to convert them)
- Key range may be large or infinite
  (and keys may be sparse)

### Hashing

“Peter Mawhorter” → “pmawhorter”
“Kim Bruce” → “kim”

- What if keys aren’t integers?
- What if we don’t know the keys ahead of time?

**fixed array**

Can we do better than a balanced binary search tree?
Perfect Hashing

- Should be O(1).
- Should return an integer.
- The integers for our N keys should be 0 ... N-1.
- Must be a unique integer for every object.
  - That is, it should be bijective.

Given hash, just use an array where:

\[
\text{items}[H(\text{key})] = \text{value}
\]

Actual Hashing

- Unique integer for an object?
  - Its address in memory.
- Numbers in 0 ... N-1?
  - Take the modulus by N.

```
public int hash(Object o, int n) {
    return addr(o) % n;
}
```

✓ Should be O(1).
✓ Should return an integer.
✓ The integers for our N keys should be 0 ... N-1.
✗ Must be a unique integer for every object.
  (true in the limit as \( N \rightarrow \infty \))

```
public int hashCode() {
    return addr(this);
}
```

- Call obj.hashCode instead of hash(obj)
- Let each map object do the modulus (N is different)
public class Point {
    public int x, y;
    
    public boolean equals(Object other) {
        if (other instanceof Point) {
            return (this.x == other.x && this.y == other.y);
        }
        return false;
    }
    
    public int hashCode() { return addr(this); }
}

Problems

- What to do when results aren’t unique?
- What about objects with .equals?
- How can we get a good distribution of results?