Lecture 30: Maps & Hashing Continued

Fall 2016

Kim Bruce & Peter Mawhorter

This week

- Lab 10: introduction to C (input/output)
  - Come prepared
- Assignment 10: priority queue in C
  - Implement a heap data structure
  - Review heap algorithms from book
  - Read Bailey's heap implementation before starting

Naïve Version

```
public class Map<K, V> {
  protected V[] entries;

  public V get(K key) {
    int index = key.hashCode() % entries.length;
    return entries[index];
  }
  public void put(K key, V value) {
    int index = key.hashCode() % entries.length;
    entries[index] = value;
  }
}
```

Warning: This code is simplified!

Hash Collisions

- `k1.hashCode() == k2.hashCode()` but `k1 != k2`
  - May also be caused by the modulus operation
- This is inevitable (e.g., the birthday paradox)
- A "good" hash function rarely collides

Avoiding Collisions

Two main strategies:

1. Open addressing (rehashing):
   - Linear probing (use the next open space)
   - Cuckoo hashing
2. Bucketing (wide cells):
   - Separate chaining

Linear Probing

- If we collide, check next entry until one is empty
- Deletion is complicated
- Can only hold `entries.length` items
- Resizing the table requires rehashing everything

Separate Chaining

- Turn each entry into a linked list (or array, etc.)
- On collision add to the bucket
- Searching list is fast if lists are small
- Deletion is simple
- Can hold more than `entries.length` items easily
- In practice slower than linear probing

Cuckoo Hashing

- Create two tables `table1` and `table2`
- Create an alternate hash function

  `int altHash(Object o)`
  - Can't use `hashCode`…
- On collision kick out an item and take its spot
- Homeless item tries to get into the other table
  - Repeat as necessary

Cuckoo Hashing

- Lookup is worst-case O(1) (only two places)
- Deletion is easy
- Can cause an infinite loop (!)
  - Unlikely if hash functions are random
- Fast in practice

Load Factor

- Performance depends on *load factor*
- Load factor is $\alpha = \frac{n}{N}$ where n = items in table and
  N = size of table
- Higher load factor → more collisions → slow
- Usually want to ensure $\alpha < 0.75$
- Generally $\alpha > 0.75$ means resize the table (& rehash everything)

We'll analyze hash table performance in lab next week

Hashing and Equality

```
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);}
}
```

Respecting Equality

```
public class Point {
  public int x, y;

  public boolean equals(Object other) {
    ...
  }

  public int hashCode() {
    return (31 * this.x) ^ this.y;
  }
}
```

See: http://www.javamex.com/tutorials/collections/hash_f unction_guidelines.shtml