Lecture 25: Maps & Dictionaries

CS 62
Spring 2018
Alexandra Papoutsaki & William Devanny
Map ADT

• Collection of associations between a key and associated value
• 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.
• Cannot contain duplicate keys; at most one value per key (matches the mathematical concept).
• Also known as “dictionaries”, “symbol tables” or “associative arrays”.

Interface

```java
public interface Map<K, V> {
    int size();
    V get(Object key);
    V put(K key, V value);
    V remove(Object key);
}
```

- **size**: number of (key, value) pairs in map
- **put**: a new (key, value) pair in map. Return value replaced if key already exists or null.
- **get**: returns the corresponding value (or null) given a key
  - To distinguish null (no pair with such key was found) from null ((key, null) pair), use `containsKey`
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();
}
## Map Implementations

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>get</th>
<th>put</th>
<th>remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>$O(n)$</td>
<td>$O(n)$</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>

Last row is array where keys are subscripts

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)
  • Suppose use SS#’s as subscripts to table of students
Hashing

Map data of arbitrary size (keys) to data of fixed size (indices)
HashMaps

• Array-like implementations of maps that provide $O(1)$ lookup
• Components:
  • Hash table: array of “buckets”
  • Hash function: to compute index of bucket
• Value returned by hash function: hash code, hash value, or hash
• Typically, number of keys is larger than table size
• Ideally, hash function will assign each key to a unique bucket
• In practice, non-perfect hash functions which cause collisions
• Value returned is called hash code, hash value, or hash
Perfect Hashing

int hash(Object o);

• Should be $O(1)$.
• Should return an integer.
• The integers for our $n$ keys should be $0 \ldots n-1$.
• Must be a unique integer for every object.
  • That is, it should be injective.
• Given hash, just use an array where: $\text{items}[H(key)] = \text{value}$
• So important that hash\text{Code} function built-in to Java classes.
Hash Functions

• Look for reasonable function that scatters elements through array randomly so won’t bump into each other.
• Lose any ordering on keys
• Ideal is to find in time $O(1)$.
• We want to:
  • Find good hashing functions
  • Figure out what to do if 2 elements are sent to same location

• “A given hash function must always be tried on real data in order to find out whether it is effective or not.”
Actual Hashing

• Unique integer for an Object?
  Its address in memory.

• Numbers in 0 ... n-1?
  Take the modulus by n

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

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

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

• Call `obj.hashCode` instead of `hash(obj)`

• Let each map object do the modulus \( n \) is different

```java
public int hashCode() {
    return addr(this);
}
```
Handling and Equality

```java
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?