Lecture 24: Maps & Dictionaries

CS 62

Fall 2018

Alexandra Papoutsaki & William Devanny

Map ADT

- Collection of disjoint entries that are associations between a key and a value
- Store and retrieve value fast 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

```
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)` entries in map
- `put`: a new `(key,value)` entry in map. Return old value replaced if key already exists or `null` .
- `get`: returns the corresponding value (or `null`) given a key
  - To distinguish `null` (no entry with such key was found) from `null` (`(key,null)` entry), use `containsKey`
- `remove`: delete the entry with `key` and return corresponding value. Return `null` if no entry with such key exists

Interface

```
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();
}
```

Example

- `OfficeNumbers = {}`

- `put("YW", 111) → null`
  `OfficeNumbers = {("YW", 111) }`
- `put("EB", 221) → null`
  `OfficeNumbers = {("YW", 111), ("EB", 221) }`
- `put("KB", 112) → null`
  `OfficeNumbers = {("YW", 111), ("EB", 221), ("KB", 112) }`
- `put("YC", 223) → null`
  `OfficeNumbers = {("YW", 111), ("EB", 221), ("KB", 112), ("YC", 223) }`
- `get("KB") → 112`
  `OfficeNumbers = {("YW", 111), ("EB", 221), ("KB", 112), ("YC", 223) }`
- `get("AP") → null`
  `OfficeNumbers = {("YW", 111), ("EB", 221), ("KB", 112), ("YC", 223) }`
- `put("EB", 127) → 221`
  `OfficeNumbers = {("YW", 111), ("EB", 127), ("KB", 112), ("YC", 223) }`

Map Implementations

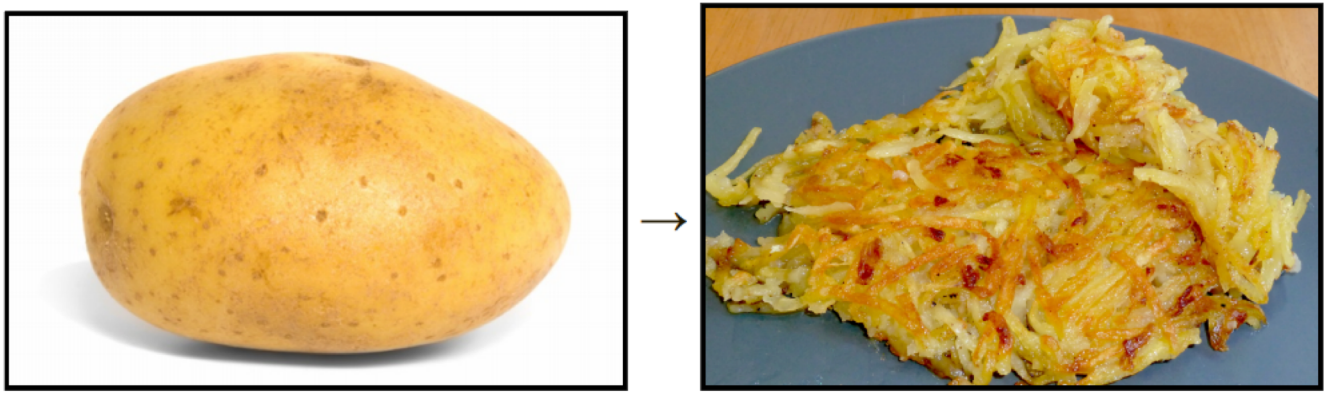
Data Structure	get	put	remove
List	$O(n)$	$O(n)$	$O(n)$
Array	$O(n)$	$O(n)$	$O(n)$
Sorted list	$O(logn)$	$O(n)$	$O(n)$
Balanced BST	$O(logn)$	$O(logn)$	$O(logn)$
Array["key range"]	$O(1)$	$O(1)$	$O(1)$

Last row is array where keys are subscripts

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)

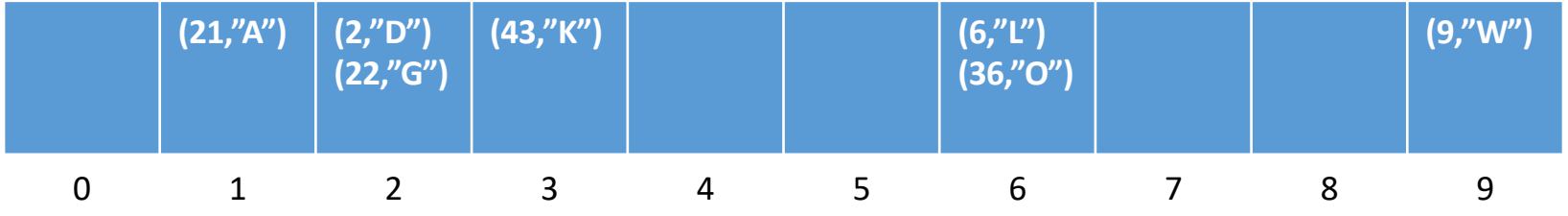
Hans Luhn, Nat Rochester, Gene Amdahl, Elaine McGraw, Arthur Samuel, 1953

HashMaps

- Array-like implementations of maps that provide $O(1)$ storing, deletion, and lookup of values given a key
- Components:
  - Hash table: array of N "buckets"
  - Hash function: to compute index of bucket, that is maps key to $0, \ldots, N-1$
- Value returned by hash function: hash code, hash value, or hash

Ex: 10 buckets, $h(k) = k\%10$

- (21,"A"), (2,"D"), (22,"G"), (43,"K"), (6,"L"), (36,"O"), (9,"W") }



Lookup: Given key k, compute $h(k)$, find value in entry stored in $h(k)$-indexed bucket

e.g., Lookup 21, $h(21) = 1$, return $(21, "A")$

Perfect Hashing

```
int hash(Object o);
```

- Should be $O(1)$.
- Should return an integer.
- The integers for N keys should be $0 \ldots N\text{-}1$.
- Must be a unique integer for every object.
  - That is, it should be bijective.
- equal keys should lead to equal hashes
  - E.g., `String s1 = "hello"`, `String s2 = "hello"`, if hash function is memory address of key, the hashrd of `s1` and `s2` would be different!
- So important that `hashCode` function built-in to Java classes.

Hash Functions

- Look for reasonable function that scatters elements through array randomly so they won't bump into each other
- Lose any ordering on keys
- Ideal is to find value 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."*

Handling and Equality

```
public class Employee {
    int         employeeId;
    String      name;
    Department dept;

    // other methods would be in here

    @Override
    public int hashCode() {
        int hash = 1;
        hash = hash * 17 + employeeId;
        hash = hash * 31 + name.hashCode();
        hash = hash * 13 + (dept == null ? 0 : dept.hashCode());
        return hash;
    }
}
```

https://en.wikipedia.org/wiki/Java_hashCode()

Problems

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