VMap

a versioned Map implementation

May 2011

R. Mark Volkmann
mark@ociweb.com

Origin of the Idea

- Driving home from Columbia after visiting son
- Wife sleeping so can't play radio
- Thinking about functional programming
- Thinking about hash tables, a.k.a. maps ← invented in 1953 by Luhn and independently by Amdahl, Boehme, Rochester, Samuel

The Problem

- Thinking about how they conflict
  - FP avoids mutable things
  - maps are mutable
- Want to be able to
  - write functions that take maps as parameters and return a new map that is a modified version of the one passed in
- Is there a way to implement an efficient, immutable hash table?

Ignorance Is Bliss

- Really into Clojure a little over two years ago
- Forgot that Clojure already solved this problem
  - `PersistentHashMap` and `PersistentHashSet`
  - uses wide tries (up to 32 children)
  - from Wikipedia, a trie is "an ordered tree data structure that is used to store an associative array"
  - based on the paper "Ideal Hash Trees" by Phil Bagwell
- So my solution
  - is nothing like the Clojure solution
  - is a variation on the typical hash table implementation

Sets From Maps

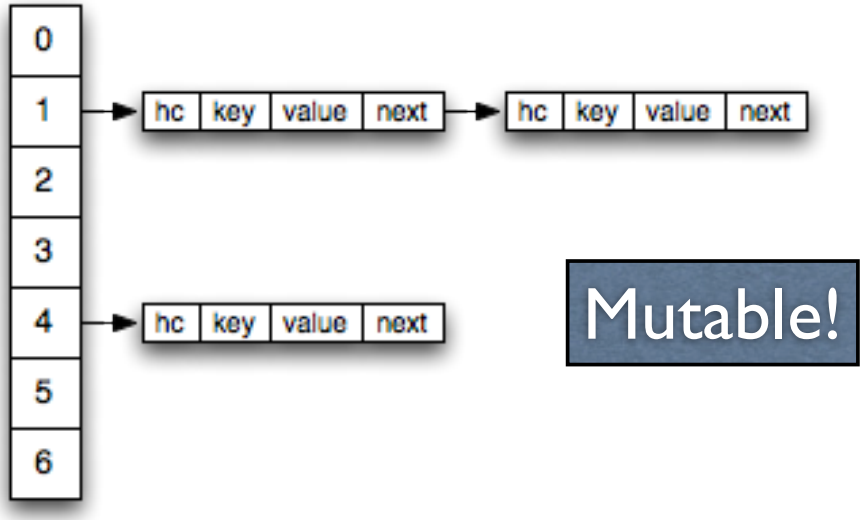
- Can implement sets from maps
- Values are booleans
- Slight optimization by
  - using boolean primitives instead of Boolean objects
  - ignoring values

Simple Approach

- Methods that normally mutate return a mutated copy instead of changing the original
  - add/put
  - delete/remove
- Stupid idea!
  - slow
  - uses too much memory
- Is there a way that one "version" of a map can share memory with another version?

Typical Map Implementation

- Buckets
  - array of chains
- Chains
  - linked list of entry objects
- Entry objects
  - hold hash code of key, reference to key, reference to value and reference to next entry
- Key objects
  - implement a "hashCode" method
    - used to locate the correct bucket
  - implement an "equals" method
    - used in chain searches to find the entry object for a given key



Steps to Add Key/Value

- Compute hash code of key
- Mod hash code by # of buckets
  to select bucket
- Walk entries in chain
  searching for an entry with an equal key
- If found, change value of existing entry
- Otherwise add new entry

In Code

- Ruby
  - `map[key] = value`
- Java
  - `map.put(key, value);`
- Objective-C
  - `[map setObject: value forKey: key];`

Steps to Find Value of a Key

- Compute hash code of key
- Mod hash code by # of buckets
  to select bucket
- Walk entries in chain
  searching for an entry with an equal key
- If found, return value of existing entry
- Otherwise return null

In Code

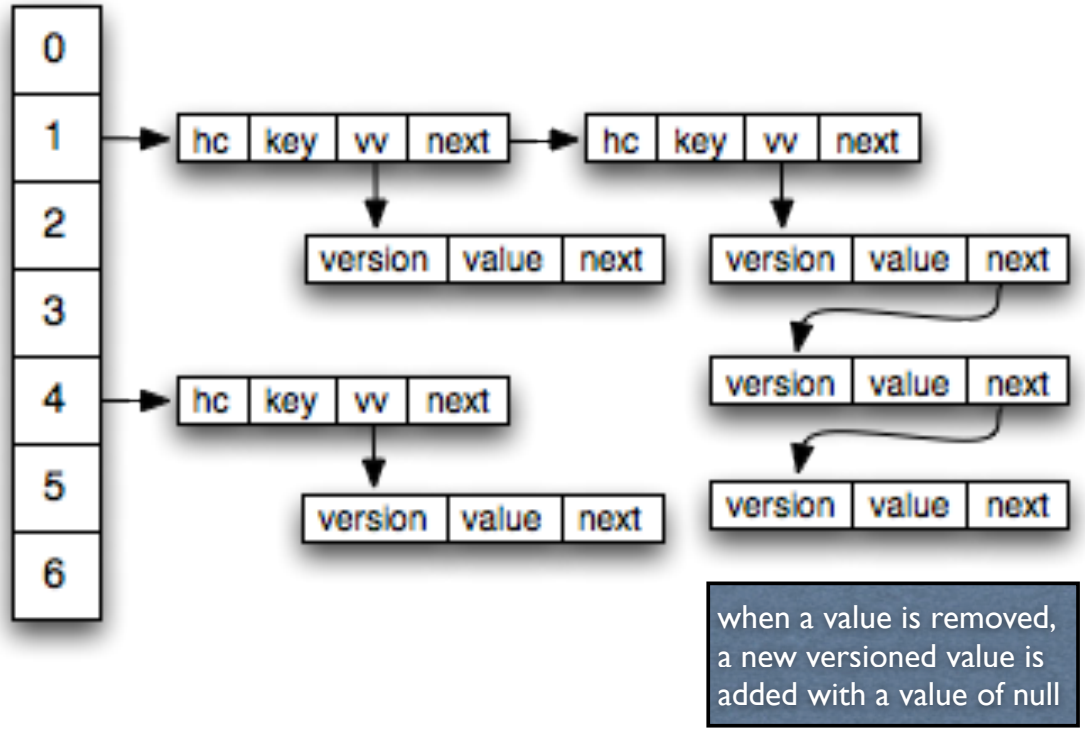
- Ruby
  - `value = map[key]`
- Java
  - `value = map.get(key);`
- Objective-C
  - `value = [map objectForKey: key];`

Rehashing

- As entries are added,
  average length of chains grows
- Lookups take longer due to
  sequential search of chains
- Rehashing fixes this by creating more buckets
  and redistributing entries into new, shorter chains
  - reason why hash codes are saved - avoids recomputing
  - new number of buckets is typically old*2 + 1
  - mod of hash code and new number of buckets locates new chain

How To Share Versions?

- Change entry objects to hold a chain of versioned values!
  - newest values at beginning since more likely to be retrieved



New VMap Instances

- Modifying a VMap creates a new instances
- But each new VMap instance can share an "internal map"
  - picture on previous slide
- Each VMap instance stores
  - version number
  - reference to internal map
  - size - to avoid recomputing each time it is requested

Rehashing Strategy

- Automatically rehash when entry count / bucket count > 0.75
  - means that on average 75% of the buckets contain one entry in their chain
    - of course many will contain more than one entry and many won't contain any entries
- Tries to avoid having many chains containing more than one entry
  - those require sequential searching

Version Limit

- Currently the type of version numbers is **`int`**
- Can't create more than **`2^31 - 1`** versions
  - 2,147,483,647 > 2 billion
- Need more than 2 billion versions?
- Could use **`long`** instead
  - 2^63 - 1 = 9,223,372,036,854,775,807 > 9 quintillion
- Other map implementations don't have a limit
- Is this a deal breaker?

Initial Implementation

- In Java because I felt most confident in getting it right there
  - adding support for generics greatly increased code complexity
- But most Java developers don't care about immutability
  - need to port to functional languages
- Lots of unit tests
- Performance tests compare to
  - Java `HashMap` and `HashSet`
  - Clojure `PersistentHashMap` and `PersistentHashSet`

Simple Example

```
VMap<String, Integer> map0 = new VHashMap<String, Integer>();
VMap<String, Integer> map1 = map0.put("foo", 1);
VMap<String, Integer> map2 = map1.put("bar", 2);
VMap<String, Integer> map3 = map2.delete("foo");
VMap<String, Integer> map4 = map3.put("bar", 3);
VMap<String, Integer> map5 = map4.put("foo", 4);
```

map0 is empty

map0 is still empty.
map1 only contains the key "foo".

map0 is still empty.
map1 still only contains the key "foo".
map2 contains the keys "foo" and "bar".

map0 is still empty.
map1 still only contains the key "foo".
map2 still contains the keys "foo" and "bar".
map3 only contains the key "bar".

map0 is still empty.
map1 still only contains the key "foo".
map2 still contains the keys "foo" and "bar".
map3 still only contains the key "bar".
map4 contains the key "bar",
but with a different value than in map2.

map0 is still empty.
map1 still only contains the key "foo".
map2 still contains the keys "foo" and "bar".
map3 still only contains the key "bar".
map4 still contains the key "bar",
but with a different value than in map2.
map5 contains the keys "foo" and "bar",
but "foo" has a different value than in map1 and map2.

Multiple Values in a Version

```
// Using a set.
VSet<String> set1 =
    new VHashSet<String>("red", "orange", "yellow");
VSet<String> set2 = set.add("green", "blue", "purple");
```

set2 contains all six colors

```
// Using a map.
VMap<String, Integer> map =
    new VHashMap<String, Integer>();
map.put(
  new Pair<String, Integer>("foo", 1),
  new Pair<String, Integer>("bar", 2),
  new Pair<String, Integer>("baz", 3));
```

there is also a constructor that takes a variable number of `Pair` objects

Which Versioned Values?

- When performing a lookup
  in a particular VMap instance,
  which versioned values should be considered?
- A given VMap instance can use values
  from its ancestor maps
  - lower version number, but not necessarily all lower versions
  - version 6 may derive from version 4 which derives from 1 and 0
  - values are in order from newest to oldest version
  - sequential search, but don't need to search far in the common case
    - tend to want newer values more than older ones
    - tend to not have a large number of values for the same key

Recording Ancestors

- How does a VMap instance record its ancestors?
  - Java implementation using `java.util.BitSet`
  - version numbers are used as indexes
  - uses an array of longs that grows automatically as needed
- BitSets are large when many versions are created
  - 1 million versions => BitSet with 15,625 longs
- But VMap instances can share a BitSet
  - just need to be careful to only check versions of indexes that are <= instance version

Searching Versioned Values

```
VersionValue<V> vv = firstVV; // of an entry
while (vv != null) {
    if (vv.version == version.number) return vv;
    if (vv.version < version.number &&
        version.ancestors.get(vv.version)) {
        return vv;
    }
    vv = vv.next;
}
return null;
```

Thread Safe?

- Lots of synchronized methods
- Performance tests pay the cost for all those locks
- Could use a code review

Performance Tests ...

- Tested using full text of four classic books

Title	Words	Unique
Alice in Wonderland	26,388	3,153
Adventures of Tom Sawyer	70,040	8,761
A Tale of Two Cities	135,820	11,671
War and Peace	562,177	21,843

Map Performance Tests

- Data
  - 1st key is "firstKey"; 1st value is first word in book
  - 2nd key is 1st word in book; 2nd value is 2nd word in book
  - and so on

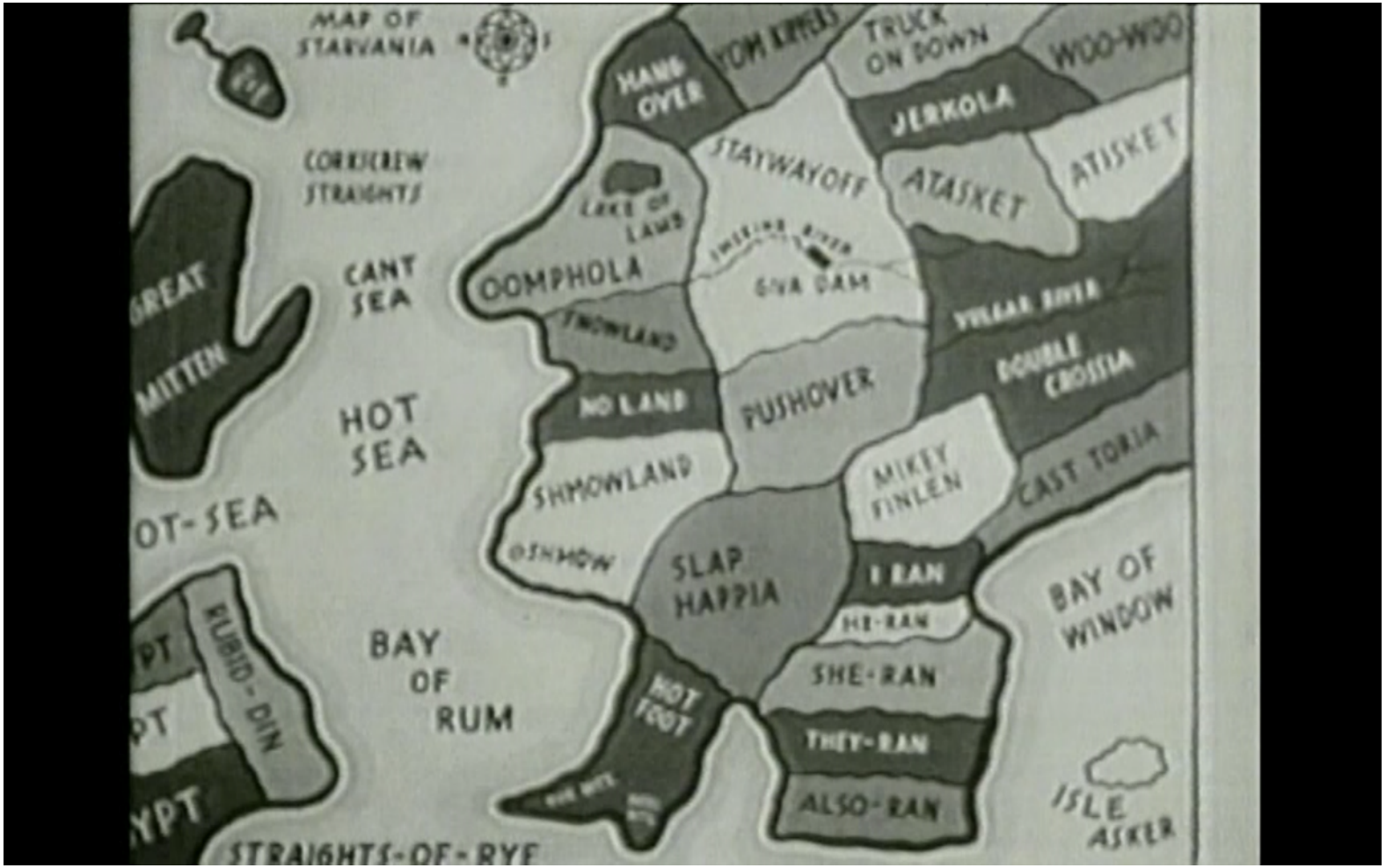
For "War and Peace" this results in 30,762 values for the key "the"!

- Steps
  - get list of key/value pairs (not included in timing)
  - create empty map
  - populate map from pairs
  - retrieve the value for each key (lookup) and verify
- Other details
  - perform a priming run for each map implementation
  - capture time of second run for each map implementation

Set Performance Tests

- Data
  - ordered collection of all words in book
    - not a set, so includes duplicates
- Steps
  - get ordered list of words (not included in timing)
  - create an empty set
  - add each of the words to the set one at a time
  - verify that the set contains each of the words (lookup)
- Other details
  - perform a priming run for each set implementation
  - capture time of second run for each set implementation

Study This Map!



Performance Results

numbers are in milliseconds;
1st is time to **load** data; 2nd is time to **lookup** values; 3rd is **total**

	Alice in Wonderland	Adventures of Tom Sawyer	A Tale of Two Cities	War and Peace
java.util.HashMap	2 1 3	6 1 7	14 4 18	89 6 95
clojure.lang.PersistentHashMap	25 3 28	46 2 48	251 3 254	658 9 667
com.ociweb.collection.VHashMap	10 5 15	24 8 32	51 5 56	271 17 288
java.util.HashSet	8 3 11	17 5 22	34 15 49	91 75 166
clojure.lang.PersistentHashSet	32 3 35	95 7 102	85 22 107	149 110 259
com.ociweb.collection.VHashSet	6 4 10	16 11 27	39 36 75	154 113 267

Conclusions

- Can't compete with java.util versions
  - but those are mutable
- Much faster loading times than Clojure versions
  - but advantage becomes less when number of entries gets very large
- Slower lookup times than Clojure versions
  - but that is a much smaller part of the total than load time
  - not good though since most apps will perform more lookups than loads
- Need to verify thread safety
- Maybe more optimizations can be made

What Do You Think?

- Has a similar idea already been evaluated?
- Is this idea worth pursuing further?
- Do you have ideas for further optimizations?
- What programming language communities would value a port of this?

- On GitHub - https://github.com/mvolkmann/VMap
- Send feedback to mark@ociweb.com
- Thanks for listening!