Lecture 28: HashMap & Collections

CS 62
Spring 2015
Kim Bruce & America Chambers

Map<K,V>

- Collection of associations between a key and associated value, e.g. name & phone number
  - Though doesn't use Bailey's Association class
- As usual lots of implementations
- Also called dictionaries after example
  - Look up table!

Hash Functions

- Want H: EltType → Subscripts, where
  - H(elt) can be computed quickly
  - if e1 != e2 then H(e1) != H(e2)
    - H is one-to-one
  - Usually difficult to achieve
  - Looked at examples Wednesday
- if redefine equals then must redefine hashCode so x.equals(y) =>
  x.hashCode() == y.hashCode()

What if get Hash Clashes?

- Home address of key K is H(K).
- Suppose have two keys $K_1 \neq K_2$,
  - but $H(K_1) = H(K_2)$, i.e., have same home address
- What happens when insert both into hash table?
  - Note original key and value must both be stored!!
- Two ways out:
  1. Rehash as needed to find an empty slot (open addressing)
  2. External chaining

Quadratic Probing

- Use (home + j^2) % TableSize on jth rehash
  - Helps with secondary clustering, but not primary
  - Can result in case where don't try all slots
    - E.g., TableSize = 5, and start with H = 1. Rehashings give 2, 0, 0, 2, 1, 2, 0, 0, ...
    - The slots 3 and 4 will never be examined to see if they have room.

Double Hashing

- Use second hash function on key to determine delta for next try.
  - E.g., delta(Key) = (Key % (TableSize - 2)) + 1
  - Should help with primary and secondary clustering.
  - Ex: Spose H(n) = n % 5. Then H(1) = H(6) = H(11).
    - However, delta(1) = 2, delta(6) = 1, and delta (11) = 3.

External Chaining

- Each slot in table holds unlimited # elts
  - Each slot is list -- implemented as desire
  - For good performance, list should be short
    - so no need for balanced binary search tree -- waste of time
- Advantages
  - Deleting simple
  - # elts in table can be > # slots
  - Avoids problems of secondary clustering
  - Primary clustering?

Analysis

- Behavior of the hash clash strategies depends on the *load factor* of the table.
- Load factor α = # elts in table/size of table
  - ranges between 0 and 1 with open addressing
  - can be > 1 with external chaining.
- Higher the load factor, the more likely your are to have clashes.

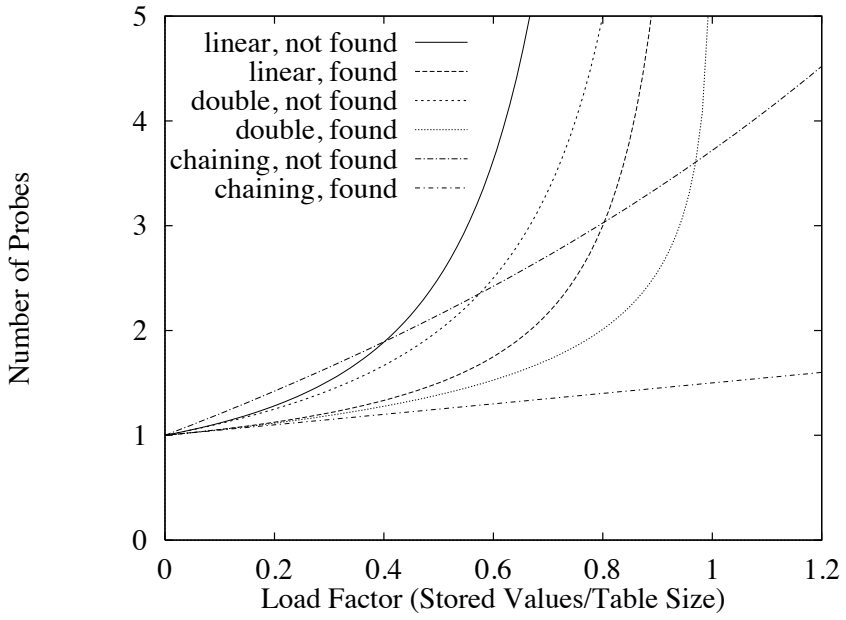
Performance

Strategy	*Unsuccessful*	*Successful*
Linear rehashing	$1/2\ (1+ 1/(1-\alpha)^2)$	$1/2\ (1+ 1/(1-\alpha))$
Double hashing	$1/(1-\alpha)$	$-(1/\alpha) \log(1-\alpha)$
External chaining	$\alpha + e^{-\alpha}$	$1 + 1/2\ \alpha$

Entries represent number of compares needed to find elt or demonstrate not there.

Performance for $\alpha = .9$

Strategy	*Unsuccessful*	*Successful*
Linear rehashing	55	5.5
Double hashing	10	~4
External hashing	3	1.45



Space requirements

- Open addressing: TableSize + n*objectsize
- External chaining: TableSize +n*(objectsize+1)
- Rule of thumb:
  - Small elts, small load factor -- use open addressing
  - Large elts, large load factor -- use external chaining

Using Hashcodes in Java

- HashMap and HashTable both implement Map
  - HashTable has all ops synchronized!
  - HashMap allows null keys and values - HT doesn't
  - HashSet is hashtable based implementation of sets.

HashMap<K,V>

- HashMap constructor
  - HashMap(int initialCapacity, float loadFactor)
  - Default load factor is .75 if not specified, default capacity 11.
  - If loadFactor exceeded then create larger table and rehash all old values -- expensive!
- Implementation seems to use external chaining

Capacity

- Don't want to set capacity too high as wastes space, though resizing expensive.
- Iterators through table require space proportional to capacity and current size.

Collections Framework

- Java library implementations of most useful general data structures.
- Description at http://docs.oracle.com/javase/6/docs/technotes/guides/collections/reference.html
- Includes concurrent implementations of data structures