Lecture 28: HashMap & Collections

CS 62
Spring 2011
Kim Bruce & Kevin Coogan

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
  • if redefine equals then must redefine hashCode
    so x.equals(y) =>
    x.hashCode() == y.hashCode()

Important!!

• How important?
  • Eclipse include automated way of generating equals and hashcode methods under “Source” menu.
  • What if insert item into hash table and then change instance vble which affects hash code?
    • Like changing priority of elt in priority queue or key of elt in ordered structure!
What if get Hash Clashes?

- Home address of key K is H(K).
- Suppose have two keys K₁ ≠ K₂,
  - but H(K₁) = H(K₂), 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 \((\text{home} + j^2) \mod \text{TableSize}\) 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., \(\text{delta(Key)} = (\text{Key} \mod (\text{TableSize} - 2)) + 1\)
  - Should help with primary and secondary clustering.
  - Ex: Suppose \(H(n) = n \mod 5\). Then \(H(1) = H(6) = H(11)\).
    - However, \(\text{delta(1)} = 2\), \(\text{delta(6)} = 1\), and \(\text{delta(11)} = 3\).

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 $\alpha = \frac{\text{# elts in table}}{\text{size of table}}$
  - 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 for $\alpha = .9$

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Unsuccessful</th>
<th>Successful</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear rehashing</td>
<td>55</td>
<td>5.5</td>
</tr>
<tr>
<td>Double hashing</td>
<td>10</td>
<td>-4</td>
</tr>
<tr>
<td>External hashing</td>
<td>3</td>
<td>1.45</td>
</tr>
</tbody>
</table>

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

Performance

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Unsuccessful</th>
<th>Successful</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear rehashing</td>
<td>$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2}\right)$</td>
<td>$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)}\right)$</td>
</tr>
<tr>
<td>Double hashing</td>
<td>$\frac{1}{1-\alpha}$</td>
<td>$-(\frac{1}{\alpha}) \log(1-\alpha)$</td>
</tr>
<tr>
<td>External hashing</td>
<td>$\alpha + e^{-\alpha}$</td>
<td>$1 + \frac{1}{2} \alpha$</td>
</tr>
</tbody>
</table>

Fig. 15.12 The shape of the theoretical performance curves for various hashing techniques. (These graphs demonstrate theoretical predictions and not experimental results which are, of course, dependant on particular data and hashing functions.) Our hash table implementation uses linear probing.
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
  - HashTable(int initialCapacity, float loadFactor)
  - Default load factor is .75 if not specified, default capacity
  - 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