Today: Hashing (Data Structures Chapter 7).

Next topic: Sorting (Data Structures Chapter 8).

Hash functions

- To do this, must have way to convert key to bucket number: a hash function.
- Example:
  - \( N = 200 \) data items.
  - keys are longs, evenly spread over the range \( 0..2^{63} - 1 \).
  - Want to keep maximum search to \( L = 2 \) items.
  - Use hash function \( h(K) = K \% M \), where \( M = N/L = 100 \) is the number of buckets: \( 0 \leq h(K) < M \).
  - So 100232, 433, and 10002332482 go into different buckets, but 10, 400210, and 210 all go into the same bucket.

External chaining

- Array of \( M \) buckets.
- Each bucket is a list of data items.
  - Not all buckets have same length, but average is \( N/M = L \), the load factor.
  - To work well, hash function must avoid collisions: keys that "hash" to equal values.
Open Addressing

- Idea: Put one data item in each bucket.
- When there is a collision, and bucket is full, just use another.
- Various ways to do this:
  - Linear probes: If there is a collision at \( h(K) \), try \( h(K) + m \), \( h(K) + 2m \), etc. (wrap around at end).
  - Quadratic probes: \( h(K) + m \), \( h(K) + m^2 \), . . .
  - Double hashing: \( h(K) + h'(K) \), \( h(K) + 2h'(K) \), etc.
- Example: \( h(K) = K \% M \), with \( M = 10 \), linear probes with \( m = 1 \).
  - Add 1, 2, 11, 3, 102, 9, 18, 108, 309 to empty table.

- Things can get slow, even when table is far from full.
- Lots of literature on this technique, but
- Personally, I just settle for external chaining.

Filling the Table

- To get (likely to be) constant-time lookup, need to keep #buckets within constant factor of #items.
- So resize table when load factor gets higher than some limit.
- In general, must re-hash all table items.
- Still, this operation constant time per item,
- So by doubling table size each time, get constant amortized time for insertion and lookup
- (Assuming, that is, that our hash function is good).

Hash Functions: Strings

- For String, "s_0 s_1 \cdots s_{n-1}" want function that takes all characters and their positions into account.
- What’s wrong with \( s_0 + s_1 + \ldots + s_{n-1} \)?
- For strings, Java uses
  \[
  h(s) = s_0 \cdot 31^{n-1} + s_1 \cdot 31^{n-2} + \ldots + s_{n-1}
  \]
  computed modulo \( 2^{32} \) as in Java int arithmetic.
- To convert to a table index in \( 0..N-1 \), compute \( h(s) \% N \) (but don’t use table size that is multiple of 31!)
- Not as hard to compute as you might think; don’t even need multiplication!
  \[
  \text{int } r; r = 0; \\
  \text{for (int } i = 0; i < s \text{.length (); } i += 1) \\
  \text{r = (r << 5) - r + s \text{.charAt (i);} \\
  \]

Hash Functions: Other Data Structures I

- Lists (ArrayList, LinkedList, etc.) are analogous to strings: e.g., Java uses
  \[
  \text{hashCode = 1; Iterator i = list.iterator();} \\
  \text{while (i.hasNext()) {} \}
  \text{Object obj = i.next();} \\
  \text{hashCode =} \\
  \text{31*hashCode} \\
  \text{+ (obj==null ? 0 : obj.hashCode());} \\
  \]
- Can limit time spent computing hash function by not looking at entire list. For example: look only at first few items (if dealing with a List or SortedSet).
- Causes more collisions, but does not cause equal things to go to different buckets.
Hash Functions: Other Data Structures II

- Recursively defined data structures ⇒ recursively defined hash functions.
- For example, on a binary tree, one can use something like
  \[
  \text{hash}(T): \\
  \text{if } (T == \text{null}) \\
  \quad \text{return 0;} \\
  \text{else return } \text{someHashFunction}(T.\text{label}()) + 255 * \text{hash}(T.\text{left}()) + 255*255 * \text{hash}(T.\text{right}());
  \]
- Can use address of object ("hash on identity") if distinct (\(!=\)) objects are never considered equal.
- But careful! Won’t work for Strings, because \(\text{.equals}\) Strings could be in different buckets:
  ```java
  String H = "Hello", S1 = H + " world!", S2 = "Hello, world!";
  Here S1.equals(S2), but S1 != S2.
  ```

Characteristics

- Assuming good hash function, add, lookup, deletion take \(\Theta(1)\) time, amortized.
- Good for cases where one looks up equal keys.
- Usually bad for range queries: "Give me every name between Martin and Napoli." [Why?]
- But sometimes OK, if hash function is monotonic (i.e., when key \(k_1 > k_2\), then \(h(k_1) \geq h(k_2)\)). For example,
  - Items are time-stamped records; key is the time.
  - Hashing function is to have one bucket for every hour.
- Hashing is probably not a good idea for small sets that you rapidly create and discard [why?]

What Java Provides

- In class \text{Object}, is function \text{hashCode()}.
- By default, returns address of this, or something similar.
- Can override it for your particular type.
- For reasons given on last slide, is overridden for type \text{String}, as well as many types in the Java library, like all kinds of \text{List}.
- The types \text{Hashtable}, \text{HashSet}, and \text{HashMap} use \text{hashCode} to give you fast look-up of objects.
  ```java
  HashMap<\text{KeyType},\text{ValueType}> \text{map} = \\
  \text{new } \text{HashMap<\text{KeyType},\text{ValueType}>} (\text{approximate size, load factor});
  \text{map.put (key, value);} // Map KEY -> VALUE. \\
  \text{... map.get (someKey)} \quad // VALUE last mapped to by SOMEKEY. \\
  \text{... map.containsKey (someKey)} \quad // IS SOMEKEY mapped? \\
  \text{... map.keySet ()} // All keys in MAP (a Set)
  ```

Comparing Search Structures

Here, \(N\) is \#items, \(k\) is \#answers to query.

<table>
<thead>
<tr>
<th>Function</th>
<th>Unordered</th>
<th>Sorted</th>
<th>Bushy Search</th>
<th>&quot;Good&quot; Hash</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{find}</td>
<td>(\Theta(N))</td>
<td>(\Theta(\lg N))</td>
<td>(\Theta(\lg N))</td>
<td>(\Theta(1))</td>
<td>(\Theta(N))</td>
</tr>
<tr>
<td>\text{add}</td>
<td>(\Theta(1))</td>
<td>(\Theta(N))</td>
<td>(\Theta(\lg N))</td>
<td>(\Theta(1))</td>
<td>(\Theta(\lg N))</td>
</tr>
<tr>
<td>\text{range query}</td>
<td>(\Theta(N))</td>
<td>(\Theta(k + \lg N))</td>
<td>(\Theta(k + \lg N))</td>
<td>(\Theta(N))</td>
<td>(\Theta(1))</td>
</tr>
<tr>
<td>\text{find largest}</td>
<td>(\Theta(N))</td>
<td>(\Theta(1))</td>
<td>(\Theta(\lg N))</td>
<td>(\Theta(1))</td>
<td>(\Theta(1))</td>
</tr>
<tr>
<td>\text{remove largest}</td>
<td>(\Theta(N))</td>
<td>(\Theta(1))</td>
<td>(\Theta(\lg N))</td>
<td>(\Theta(N))</td>
<td>(\Theta(\lg N))</td>
</tr>
</tbody>
</table>