Hashing
Part Two
Recap from Last Time
Hash Functions

- A **hash function** is a function that converts a large object (a genome, a string, a sequence of elements, etc.) into a smaller object (a shorter string, an integer, etc.)

- A hash function **must** be deterministic: given an input, it must always produce the same output.
  - *Why?*

- A hash function **should** try to produce different outputs for different inputs.
  - Not always possible if there are only finitely many possible outputs.
Overview of Our Approach

• To store key/value pairs efficiently, we will do the following:
  • Create a lot of **buckets** into which key/value pairs can be distributed.
  • Use a hash function to associate each possible key with a bucket.
  • To look up the value associated with a key:
    – Jump into the bucket containing that key.
    – Look at all the values in the bucket until you find the one associated with the key.
Building a Hash Table
Quick Announcements!
Apply to Section Lead!

http://cs198.stanford.edu
Casual CS Dinner

- Casual dinner for women studying computer science is next **Thursday, May 23** at **5:30PM** at the **Gates Patio**.
- Everyone is welcome!
- RSVP through link sent out Friday, or at **http://bit.ly/cscasualdinners**
YEAH Hours

- YEAH Hours (assignment review session) for Assignment 5 is **tomorrow**, May 21st in **Gates B12** from **5:30PM - 6:30PM**.
  - We will post notes on the course website.
} // End announcements
Hash Table Performance

• Suppose that we have \( n \) elements and \( b \) buckets.

• Assuming a good hash function, the expected time to look up an element is \( O(1 + n / b) \).

• The ratio \( n / b \) is called the \textbf{load factor}.

• Intuitively, this makes sense – if the elements are distributed evenly, you only need to look, on average, at \( n / b \) of them.
Hashing and Rehashing

Dumbledore  Harry  Lily

Draco  Hermione  Minerva

Ron  Hagrid  Snape
Hashing and Rehashing

Harry

Ron

Dumbledore

Hagrid

Voldemort

Snape

Draco

Minerva

Hermione

Lily

Ron

Hagrid

Snape
Hashing and Rehashing

Voldemort

Minerva  Lily  Dumbledore  Hagrid  Harry  Snape

Ron  Draco  Hermione
Hashing and Rehashing

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minerva</td>
<td>Lily</td>
<td>Dumbledore</td>
<td>Hagrid</td>
<td>Harry</td>
<td>Voldemort</td>
</tr>
<tr>
<td></td>
<td>Ron</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Draco</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Snape</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hermione</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hashing and Rehashing

- Idea: Track the number of buckets $b$ and the number of total elements $n$.
- When inserting, if $n/b$ exceeds some small constant (say, 2), double the number of buckets and redistribute the elements evenly.
- This makes $n/b \leq 2$, so the expected lookup time in a hash table is $O(1)$.
- On average, the lookup time is independent of the total number of elements in the table!
Coding it Up
The Final Analysis

- Expected time to do a lookup: \(O(1)\).
- Expected time to do an insertion:
  - Every \(n\) elements, must double the table size and rehash. Does \(O(n)\) work, but only every \(n\) iterations.
  - Then does \(O(1)\) expected work to do the insertion.
- *Amortized expected \(O(1)\) insertion!*
Next Time

• **Binary Search Trees**
  • How else might you store a large number of key/value pairs?
  • And why are our `Map` and `Set` stored in sorted order?