Announcements

- Program due Saturday - checkpoint Thursday
- Give a "study sheet"
- Review session late next week
- Final is in 2 weeks
Last time: Huffman trees/codes
(Hash Tables)

Data Storage

<table>
<thead>
<tr>
<th>Locker #</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>109</td>
<td>Erin</td>
</tr>
<tr>
<td>65</td>
<td>Kerim</td>
</tr>
<tr>
<td>350</td>
<td>David</td>
</tr>
<tr>
<td>54</td>
<td>Mary</td>
</tr>
<tr>
<td>210</td>
<td>Austin</td>
</tr>
</tbody>
</table>

We want to be able to retrieve a name quickly given a locker #.
\[ n = \# \text{ of people} \]
\[ m = \# \text{ of lockers} \]

How could we store this?
(\& how much space/time would it take?)

**List**: Each node gets 2 data fields
- insert: \(O(1)\)
- find: \(O(n)\)
- space: \(O(n)\)

**Array/Vector**: \(m\) size, if locker is being used
- store name in that array spot
- space: \(O(m)\)
- find: \(O(1)\)
- insert: \(O(1)\)

**AVL tree**: Size: \(O(n)\)
- insert: \(O(\log n)\)
- find: \(O(\log n)\)
Other examples:

- Course # + schedule info
- Flight # + arrival info
- URL + HTML page
- Color + BMP

Not always easy to figure out how to store and lookup.
Dictionaries:

A structure which supports the following:

```cpp
void insert (keyType &k, dataType &d);
dataType find (keyType &k);
void remove (keyType &k);
```

Everything is based on keys!

(not always easy to "compare" keys)

→ (Notice: an array is a dictionary)
Hashing

An array is not very space efficient. We would like to take the key and make it smaller.

A hash function $h$ maps each key in our dictionary to an integer in the range $[0, N-1]$. 

($N$ should be much smaller than the # of keys.)

Then we store $(k, v)$ in $A[h(k)]$.
Good hash functions:
- Are fast (goal: O(1) time)
- Don't have collisions.

Collisions are unavoidable.

N < key space
First: map key to a number

Say we want keys to fit in an int.

What can we do for int, char, and short types?

Now what about long or float?

\[ a \rightarrow 32 \]

\[ a + b \rightarrow \text{ simplest way to hash} \]
int hashCode (long x) {
    return int (unsigned long (x >> 32) + int (x)) % N;
}