Announcements

- HW7 - graded, HW8 back soon?
- HW9 - due tomorrow
- Review Friday, test Monday
- HW10 posted, decoding checkpoint Monday before break
- No lab tomorrow
New problem: Data Storage

Ex.

<table>
<thead>
<tr>
<th>Locker #</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Dan</td>
</tr>
<tr>
<td>355</td>
<td>Kevin</td>
</tr>
<tr>
<td>101</td>
<td>Tracy</td>
</tr>
<tr>
<td>53</td>
<td>Nitish</td>
</tr>
<tr>
<td>201</td>
<td>David</td>
</tr>
</tbody>
</table>

We want to be able to retrieve a name quickly when given a locker number.

(\text{Let } n = \# \text{ of people } + m = \# \text{ of lockers} )
How could we store this?

1. Lists
   - Space: $O(n)$ - 1 node per person
   - find: $O(n)$
   - insert: $O(n)$
   - delete: $O(n)$

2. Vector
   - Space: $O(n)$
   - find: $O(n)$ (if sorted)
   - insert: $O(n)$ (sorted)
   - remove: $O(n)$
(3) Array:

- locker #: 0 1 2 3

- space: $O(m)$ → # lockers
- find: $O(1)$
- insert: $O(1)$
- delete: $O(1)$

(4) AVL trees - $O(\log n)$ for all ops
$O(n)$ site
Other examples

- Course # and schedule info
- Flight # and arrival info
- URL and html page
- Color and BMP file

Not always easy to figure out how to store and look up.
Dictionaries - associative arrays

A data structure which supports the following:

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

Note: Everything is based on keys!
Data Structures

First thing to note:
An array is a dictionary.

- key: index
- data: value in that spot

Other alternatives:
(see a few slides back)
Hashing

Assuming \( m \gg n \), an array is not very space efficient.

We would like to use \( O(n) \) space, not \( O(m) \).

But then the key needs to get smaller.

Additional challenge: non-numeric keys.
**Def:** 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 \(m = \# \text{ of keys}\).)

Then given \((k, e)\), we store \((k, e)\) in array spot \(A[h(k)]\).
Good hash functions:

- Are fast
- Don't have collisions: \( k = k' \) but \( h(k) = h(k') \)

key space (size \( m \)) \( \xrightarrow{(k,e)} \) \( h(k) \) \( \xrightarrow{O(1)} \)

\( m \gg N \)

\( h(k) = 0 \)
So we have a few steps.

1. Make key a number
2. Compress that number to $[0, N-1]$
3. Since not perfect, handle collisions somehow.
1. Take key and make it a number.
   (Remember! keys can be anything!)
   Ex: char, int, or short (all 32-bits)

   cast immediately to a #
Ex: long or float - 64 bits
(K needs to be 32 bits)
int hashCode (long x) {
    return int(unsigned long(x >> 32)) + int(x);
}
What about strings?

(Think ASCII.)

```
E r i n
```

```
69 + 114 + 105 + 110 = \text{int} \\
\text{truncate}
```

Goal: a single int.
But, in some cases, a strategy like this can backfire.

```plaintext
temp01 and temp10 and pm0te1
+e+m+p+0+1 = +e+m+p+1+0
```

go to same int

We want to avoid collisions between "similar" strings (or other types).
Next time: locality