Hash Tables

A table maps each element to a specific cell based on a key.
A hash function compresses a range of keys into a smaller range.
A hash table is a table that uses a hash function.

For example, if key range is 0 - 4800, hash table length is 49, and hash function is key/100 (integer division), then an object with key 34 would go into cell 0, or an object with key 340 would go into cell 3.

Problem: what if two elements map to the same location in the table? This is a collision.

2 Solutions: Chaining or open addressing.

Chaining: use other data structure to store multiple elements in a cell (ex: linked list, array list, tree) or use overflow area in table.
Linked

Array

overflow

Open addressing: use another position in table. Choose next position through various methods.

linear probing - look for next available space

<table>
<thead>
<tr>
<th></th>
<th>add 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>add 15</td>
</tr>
<tr>
<td>15</td>
<td>add 20</td>
</tr>
<tr>
<td>20</td>
<td>add 18</td>
</tr>
</tbody>
</table>

quadratic probing - add \(-1^{i-1} ((i+1)/2)^2\) to the hash code, where \(i\) is incremented every time there is a collision.

double hashing - hash the hash code if there is a collision.
hashing functions -
  division (or modulo) - \( 1234 \% 100 \Rightarrow 34 \)
  shift - \( 1234 \Rightarrow 2341 \)
  extraction - \( 1234 \Rightarrow 23 \)
  flip - \( 1234 \Rightarrow 4321 \)
  fold - \( 1234 \Rightarrow 12 + 34 \Rightarrow 46 \) (can use other ops)

Analysis

<table>
<thead>
<tr>
<th>Hash</th>
<th>AVL</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>find</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>remove</td>
<td>( O(n) )</td>
</tr>
</tbody>
</table>

This is not better than an AVL unless there is ...

A perfect hash function; hash function that maps all data to hash table with no empty cells and no collisions.

In practice a perfect hashing function is not needed, just a good one.