Announcements
- Program due tomorrow
- HW due next Monday
- Review next Friday at 10:30
  (check webpage Thursday in case of a room change)
- Final Monday at noon
- No conflict (so far)
Final comments on hashing (re-cap)

1. Convert key to 32-bit int, k
2. Compute \( h(k) \), a value between \( 0 \text{ to } N-1 \)
   - Goals:
     - Fast - \( O(1) \)
     - Minimize collisions
3. Deal with collisions. How?
   - Auxiliary data structure (list, tree, etc.)
   - Linear probing
   - Quadratic probing
Issues with Quadratic Probing:

- Can still cause "secondary" clustering
- N really must be prime for this to work

- Even with N prime, starts to fail when array gets half full

(Runtimes are essentially the same)

Even worse: this can just fail: the array has an empty spot, but this function can't get to it
Secondary Hashing

- Try $A[h(k)]$
- If full, try $A[h(k) + f(j)] \mod N$

where

$f(j) = j \cdot h'(k)$

with $h'$ a different hash function

Another hash fun

$h(k) + 2h'(k)$
$h(k)$
$h(k) + h'(k)$

quadratic, we had
$f(g) = j^2$
Load Factors

Separate chaining actually works as well as most others in practice although it does use more space.

Most of these methods only work well if \( \frac{n}{N} < 0.5 \).

(Even chaining starts to fail if \( \frac{n}{N} > 0.9 \))
Rehashing

Because we need $\frac{n}{N} < 0.5$, most hash code checks if the array has become more than half full.

If so, it stops and recomputes everything for a larger $N$, usually at least twice as big.

(Still not too bad in an amortized sense — think vectors.)
In practice

Hashing is the fastest thing!

Given a good load factor, this uses O(a small amount of space) and runs in O(1) time.

This is a good example when the guaranteed run times are very different from what we see in practice.
Treaps: a new binary tree data structure

- Nodes will contain both values and priorities

- A treap is a BST over the values and a heap over the priorities.

Use letters + int values priorities

<min heaps
Example:

letters form a BST
numbers give a mind leap
(letters are in alpha order)
Insert

Insert: \((S, 0)\)

Forget priority
Where should \(S\) go in BST?
Am I a min heap?
In heap, we "bubbled up". Will that work here?

No. If I swap of $R + S$ nodes, no longer a BST over values.

$R \leq S$
Rotations

$x \leq y$ are in correct BST order, with $x \leq y$, but priorities are wrong.

Fix:

Nice: this is just our pivot.
So: insert \((s, 0)\)
Treap again!
Downside: What can height be?
Can we force them to be balanced?
No: \((A, 0) \ (B, 3) \ (C, 11) \ (D, 20)\)

Each set of data has a unique treap.
Draw heap with \((A, 4), (C, 2), (X, 11), (M, 3), (Q, 1), (Z, 5)\)

Who is root?

(save tons of time)
Randomized treaps:

Alternative to AVL trees.

Each element will get a random priority.

Expected height of the treap will be $O(\log n)$.

Worst case is still $O(n)$. 
Code: How do we implement?

Inherit from Binary Search Tree.

- Priorities are in aux.
- Insert (use pivot to fix)
- Delete

(look for extra lecture notes)