Today: Skip lists
- building from scratch
- insert algorithm
- analysis: time & space

Review:

<table>
<thead>
<tr>
<th>dynamic set data structure</th>
<th>Insert/ Delete</th>
<th>Exact Search</th>
<th>Predecessor/ Successor</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>linked list</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>= 'only comparison hashable comparison'</td>
</tr>
<tr>
<td>sorted array</td>
<td>O(n)</td>
<td>O(lg n)</td>
<td>O(lg n)</td>
<td></td>
</tr>
<tr>
<td>hash table</td>
<td>O(1) exp.</td>
<td>O(1) exp.</td>
<td>O(n)</td>
<td></td>
</tr>
<tr>
<td>BST</td>
<td>O(lg n)</td>
<td>O(lg n)</td>
<td>w.h.p. if uniform inserts O(lg n)</td>
<td></td>
</tr>
<tr>
<td>balanced search tree</td>
<td>focus this week</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Motivation:
- Predecessor/Successor
  - e.g. nearest match in library search
- comparisons only
- augmentation to store data about subparts of data structure
Skip lists [Pugh 1989]
- a simple, efficient dynamic search structure you'll never forget
- randomized
- $O(\log n)$ time/op. w.h.p.

Experiment in 2005: implement skip lists
$\approx 10$ minutes for linked list
$\approx 30$ minutes for skip list $\{\text{not bad}\}$
[\approx 60$ minutes debugging $]

Starting from scratch:
- simplest data structure: (sorted) linked list
- worst-case search: $O(n)$
- how to improve?

Example:
\begin{align*}
\text{1:} & \quad 14, 23, 34, 42, 50, 59, 66, 72, 79, 86, 96, 103, 110, 116, \ldots \\
\text{L}_1: & \quad 14 \leftrightarrow 34 \leftrightarrow 42 \leftrightarrow 50 \leftrightarrow 59 \leftrightarrow 66 \leftrightarrow 72 \leftrightarrow 79 \leftrightarrow 86 \leftrightarrow 96 \leftrightarrow \ldots \\
\text{L}_2: & \quad 14 \leftrightarrow 23 \leftrightarrow 34 \leftrightarrow 42 \leftrightarrow 50 \leftrightarrow 59 \leftrightarrow 66 \leftrightarrow 72 \leftrightarrow 79 \leftrightarrow 86 \leftrightarrow 96 \leftrightarrow \ldots \\
\end{align*}

(New York City 7th Ave. Subway line)
Idea: 2 sorted linked lists
- each element in one or both lists
- all elements in bottom list
- first element in both lists
- link between copies in both lists

Search(x):
- walk right in top list L₁ until going right would go too far
- walk down to bottom list L₂
- walk right in L₂ until x or successor found

Which elements should go in top list L₁?
- in subway: "popular stations"
- here want to minimize worst-case performance
- best to evenly space nodes in L₁
- but how many should be in L₁?

Analysis: search cost \( \approx |L₁| + \frac{|L₂|}{|L₁|} \)
- minimized (up to constant factors)
  when \( |L₁| = |L₂|/|L₁| \)
  i.e. \( |L₁|^2 = |L₂| = n \) (everyone is in L₂)
  i.e. \( |L₁| = \sqrt{n} \)
  \( \Rightarrow \) search cost \( \approx \sqrt{n} + n/\sqrt{n} = 2\sqrt{n} \)
Structure:

More lists:
- 2 sorted lists $\Rightarrow 2\sqrt{n}$
- 3 sorted lists $\Rightarrow 3\cdot3\sqrt{n}$
- $k$ sorted lists $\Rightarrow k\cdot k\sqrt{n}$
- $\log n$ sorted lists $\Rightarrow \log n \cdot \sqrt[2\log n]{n} = \log n \cdot 2^{\frac{\log n}{2\log n}} = 2^{\frac{1}{2}} = 2$

$\log n$ lists: IDEAL SKIP LIST
- like a binary tree where $\text{key}(x) = \min(\text{subtree})$
  (actually a level-linked $B^+$-tree)
- bottom list contains all elements
- each list contains a subset of list below
- first element in all lists
- vertical linked list of all copies of an element

Search(72)
**Search**($x$):
- start at top left (beginning of first list)
- until we fall below bottom list:
  - if going right would go too far: go down
  - else: go right

**Skip list** maintains roughly this idealized structure using randomization during inserts
- idea: Insert($x$) always adds $x$ to bottom list, then promotes $x$ up some $i$ levels
- when should $i = 0$ / $i \geq 1$? $1/2$
- when should $i = 1$ / $i \geq 2$? $1/4$
- when should $i = 2$ / $i \geq 3$? $1/8$
  etc.

**Insert**($x$):
- Search($x$) to find where $x$ fits in bottom list
- insert $x$ in bottom list
- flip fair coin
- if heads: promote $x$ to next level up
  (possibly newly created)

**Detail**: add special $-\infty$ value to every list
  ⇒ first element is in every list
  (needed for Search)
**Exercise:** build a skip list with a real coin
- write H/T coin flip at each node

**Delete** \(x\): just remove \(x\) from all lists

Why are skip lists good?

**Warmup Lemma:** # levels in \(n\)-element skip list is \(O(\log n)\) w.h.p.

\[
\frac{c \log n}{n} \leq \text{prob. } 1 - \frac{1}{n^\alpha}
\]

**Proof:** failure probability (not \(\leq c \log n\) levels)

\[
= \Pr\{ \exists > c \log n \text{ levels} \}
\]

\[
= \Pr\{ \text{some element got promoted} > c \log n \text{ times} \}
\]

\[
\leq n \cdot \Pr\{ \text{element } x \text{ got promoted} > c \log n \text{ times} \}
\]

by Union Bound

\[
= n \cdot \left( \frac{1}{2} \right)^{c \log n}
\]

\[
= \frac{n}{n^c}
\]

\[
= \frac{1}{n^{c-1}}
\]

\[
= \frac{1}{n^\alpha} \quad \text{for } \alpha = c-1 \text{ i.e. } c = \alpha+1. \quad \square
\]
Theorem: any search in an n-element skip list costs $O(lg\ n)$ w.h.p.

Cool idea: analyze search backwards (up)
- search starts [ends] at node in bottom list
- at each node visited:
  - if node wasn't promoted higher (tails here) then we go [came from] left
  - if node was promoted higher (heads here) then we go [came from] up
- stop [start] when we reach top level or -oo

Proof of theorem:
- Search makes "up" & "left" moves, each with probability $\frac{1}{2}$
- number of "up" moves $\leq$ # levels
- $\leq c \cdot lg\ n$ w.h.p. (by Warmup Lemma)

$\Rightarrow$ w.h.p., number of moves $\leq$ number of coin flips before we get $\geq c \cdot lg\ n$ tails
- now use coin-flipping result from Lecture 4:

Claim: $\geq c \cdot lg\ n$ tails in $O(lg\ n)$ coin flips, w.h.p.
Space clearly $O(n \log n)$ by Warmup Lemma...

In fact:

**Theorem**: space used by $n$-element skip list is $O(n)$ with exponentially high prob. (& in expectation)

**Proof**: can think of skip-list construction as a big sequence of coin flips
- distance between consecutive heads = # times to promote next element
- space = # nodes = total # coin flips
- finished after $n$ heads

$\Rightarrow$ space = # coin flips until $\geq n$ heads

$= O(n)$ w. exp. h.p.

Also: $E[\text{space}] = E\left[ \sum_x \# \text{ lists containing } x \right]

= \sum_x E[\# \text{ lists containing } x]

\quad \text{(linearity of expectation)}

= \sum_x (1 + \frac{1}{2} + \frac{1}{4} + \cdots)

= \sum_x 2

= 2n \square$