Ordered Collection Overview

Skip List Analysis

Analysis

What is expected height (# of levels in tallest tower)?

How many times per level do we expect to move (or peak forward)?
Last time argued that expect $n \cdot p^i$ elements in level $i$. Find $i^*$ for which $n \cdot p^{i^*} = \frac{1}{P}$ for $P = \frac{1}{4}$, level when you only expect 4 elements to remain.

$n = \left( \frac{1}{P} \right)^{i+1}$

$log_{1/4} n = i + 1$

$i = log_{1/4} n - 1$
So at level $\log_{1/p} n - 1$ expect only $1/p$ elements to remain. # levels until this point $\log_{1/p} n$

Can prove $E[\text{height of tallest tower}] = \log_{1/p} n + \frac{1}{1-p} \left( p = \frac{1}{4} \log_4 n + \frac{4}{3} \right)$
Search Cost

Page 615 goes over more formally.

Intuitively,

Expect to look at \( \frac{1}{p} \) per level

\begin{align*}
\text{level } i & \quad \vdots \\
\text{look at towers at level } i-1
\end{align*}
So expect \( \log_{1/p} n + \frac{1}{1-p} \) levels

+ expect to consider \( 1/p \) elements per level

\[
\Rightarrow E[\text{search time}] \approx \frac{1}{p} \log_{1/p} n = \frac{1}{p \log_2 n}
\]

For \( p = 1/4 \), \( E[\text{search time}] \approx 2 \log_2 n \)
Space Usage

What is expected height of a tower

\[ E[h] = \sum_{i=0}^{\infty} i \cdot p^{i-1} (1-p) = \frac{1}{1-p} \]

- \( i-1 \) times
- continue last time
- occurs with prob \( p \)
- stop
- occurs prob \( 1-p \)
\# next/prev refs (dominates space usage)

\[ \sim N \cdot \frac{1}{1-P} \cdot 2 + 4 \left( \log_{\frac{1}{P}} n + \frac{1}{1-P} \right) \]

- \# towers
- expected height
- \# towers per level
- next + prev of each tower
- for head + tail

\[ = \frac{2n}{1-P} + 4 \log_{\frac{1}{P}} n + \frac{4}{1-P} \]
# next/prev refs in skip list for p = 1/4

\[ 2^{2/3} n + 2 \log_2 n + \frac{8}{3} \approx 2^{2/3} n \]

# left, right, parent refs in red-black tree

\[ 3n \]

# parent, child, next, prev refs in B^+ tree with t = 2

On average \( n^{n/2} \) leaves + \( \approx \frac{3n}{4} \) internal nodes

\[ \Rightarrow \approx 3(n/2) + 5(3n/4) = \frac{21n}{4} = 5.25n \]
**Skip List overview**

<table>
<thead>
<tr>
<th>As a function of $p$</th>
<th>$p = 1/2$</th>
<th>$p = 1/e$</th>
<th>$p = 1/4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>space usage</td>
<td>$2n/(1 - p) + 4 \log_{1/p} n$</td>
<td>$4n + 4 \log_2 n$</td>
<td>$\approx 3.2n + 2.8 \log_2 n$</td>
</tr>
<tr>
<td>search cost</td>
<td>$(\log_{1/p} n)/p$</td>
<td>$2 \log_2 n$</td>
<td>$\approx 1.88 \log_2 n$</td>
</tr>
</tbody>
</table>

The value of $p$ to minimize search cost is the one that balances space usage and search cost efficiently.
# Overview of Ordered Collection Data Structures

**Key**
- ☑ Excellent
- ○ Very Good
- ● Fair
  - Method does nothing

<table>
<thead>
<tr>
<th>Method</th>
<th>SortedArray</th>
<th>BinarySearchTree</th>
<th>RedBlackTree</th>
<th>SplayTree</th>
<th>Btree</th>
<th>B+Tree</th>
<th>SkipList</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(o)</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>addAll(c), per element</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>clear(), per element</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>contains(o)</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>ensureCapacity(x)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>getLocator(o)</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>iterator(), iteratorAtEnd()</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>max(), min()</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>predecessor(o)</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>remove(o), retainAll(c), per element</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>successor(o)</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>
### Key
- **⊲** Excellent
- **〇** Very Good
- **●** Fair
  - Method does nothing

<table>
<thead>
<tr>
<th>Method</th>
<th>SortedArray</th>
<th>BinarySearchTree</th>
<th>RedBlackTree</th>
<th>SplayTree</th>
<th>Btree</th>
<th>B+Tree</th>
<th>SkipList</th>
</tr>
</thead>
<tbody>
<tr>
<td>typical space</td>
<td>⊲</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>●</td>
<td>〇</td>
</tr>
<tr>
<td>randomized</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>●</td>
<td>〇</td>
</tr>
<tr>
<td>amortized approach (occasionally restructures)</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>●</td>
<td>〇</td>
<td>〇</td>
</tr>
<tr>
<td>fast access to recently accessed elements</td>
<td>〇</td>
<td>〇</td>
<td>●</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
</tr>
<tr>
<td>designed to minimize disk pages read</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
<td>●</td>
<td>〇</td>
<td>〇</td>
<td>〇</td>
</tr>
</tbody>
</table>

### Other Issues
- ADVANCE(): 〇
- GET(): 〇
- HASNEXT(), NEXT(): 〇
- REMOVE(): ●
- RETREAT(): 〇