Skip List

call a tower \( t \)

t.next[ ]

tower holds: array of refs to next/prev towers, ref to associated data

abstract view

\[
\begin{align*}
L_0: & \quad a \quad a \quad b \quad c \quad r \quad s \quad r \quad t \quad t \\
L_1: & \quad b \quad r \quad t \\
L_2: & \quad \text{element} \quad \text{element} \quad \text{element} \quad \text{element} \quad \text{element} \quad \text{element} \quad \text{element} \quad \text{element} \quad \text{element} \\
\text{head} & \quad \text{element} \quad \text{element} \\
\text{tail} & \quad \text{element} \quad \text{element} \\
\end{align*}
\]
L₂: r
L₁: b r t
L₀: a a b c r s t t

Show as

How do you find min? head.next[0]
max? tail.prev[0]

How do you move to next/prev element from a given tower? ptr.next[0] ptr.prev[0]
How do you search for an element?
```java
Tower<E> findFirstOccurrence(E target) {
    Tower<E> left = head;
    Tower<E> right = tail;
    int level = height - 1;
    while (level >= 0) {
        Tower<E> next = left.next(level);
        if (right == next) {
            level--;
            //no
        } else {
            if (comp.compare(target, next.element) > 0) {
                left = next;
                right = next;
                level--;
            } else {
                level--;
            }
        }
    }
    return right;
}
```
How can you find predecessor/successor of an element (e.g., «d»)?

Use level 0 to navigate in sorted order.
**Insertion**

let \( p \) be 

prod of a 

continuing 

Ex. \( p=\frac{1}{4} \)

Find position to insert using search

Then randomly pick height of new tower as # biased coin flips until head obtained
Once the position is found and height is selected, just splice new tower into list for each level (already know where from search).

Also update height (occasionally must resize head/tail).
Deletion

height = 3

Search to find + then splice out of each level it is in (just like in a linked list).

Also update height (+ occasionally must resize head/tail).
Analysis

(expected)

(What is height (and of levels in tallest tower)?)

X

(How many times per level do we expect to move (or peak forward)?)
Let $n$ be \# elements (towers).

Claim: expect $n \cdot p^i$ elements in level $i$ list

\[
\begin{align*}
\hat{c} = 0 & \quad n \quad \hfill n \\
\hat{c} = 1 & \quad n \cdot p \quad \hfill n/4 \\
\hat{c} = 2 & \quad n \cdot p^2 \quad \hfill n/16 \\
\end{align*}
\]

For $p = \frac{1}{4}$

Prove by induction
When is \( n \cdot p^i = \frac{1}{p} \)

\[
n = \left(\frac{1}{p}\right)^{i+1}
\]

\[
\log_{\frac{1}{p}} n = i + 1
\]

\[
i = \log_{\frac{1}{p}} n - 1 \geq \log_{\frac{1}{p}} n
\]

when \( p = \frac{1}{4} \)

only 4 items expected at level \( i \).
So at level $\log_{1/p} n - 1$ expect only $1/p$ elements to remain.

Levels until this point $\log_{1/p} n$:

$0, 1, 2, \ldots, \log_{1/p} n - 1$

Can prove:

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

Page 615 goes over more formally.

Intuitively,

- prob $P$ it reaches level $i$
- \{ Expect to look at $1/p$ per level \}

level $i$

- look at towers at level $i-1$
So expect \( \log_{1/p} n + \frac{1}{1-p} \) levels

+ expect to consider \( \frac{1}{p} \) elements per level

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

For \( p = \frac{1}{4} \), \( E[\text{search time}] \approx 2 \log_2 n \)
$p \log \frac{n}{p}$

$\frac{1}{4} \frac{1}{3} \phi^{1/2}$
# next/prev refs (dominates space usage)

Expected height of a tower

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

\[
\frac{2}{1-p} \cdot n \quad (p=\frac{1}{4}, \frac{8}{3} n)
\]
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>

\[ \text{value of } p \text{ to minimize search cost} \]
Relationship between Skip List + B⁺-tree

1. Show how elements are grouped in B⁺ tree vs skiplist

2. What trade-offs are there between skiplist + B⁺ tree in terms of cache behavior
# Overview of Ordered Collection Data Structures

<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>1</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>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>contains(o)</td>
<td>1</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>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max(), min()</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>predecessor(o)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove(o), retainAll(c), per element</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>successor(o)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td>SortedArray</td>
<td>BinarySearchTree</td>
<td>RedBlackTree</td>
<td>SplayTree</td>
<td>Btree</td>
<td>B+Tree</td>
<td>SkipList</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>------------------</td>
<td>--------------</td>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>Excellent</td>
<td>⊡</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>Very Good</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>Fair</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>Method does nothing</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
</tbody>
</table>

**Method**

<table>
<thead>
<tr>
<th></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>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>randomized</td>
<td>⊡</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>amortized approach (occasionally restructures)</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>fast access to recently accessed elements</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>designed to minimize disk pages read</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
</tbody>
</table>

**Locator Methods**

<table>
<thead>
<tr>
<th></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>advance()</td>
<td>⊡</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>get()</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>hasNext(), next()</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>remove()</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
<tr>
<td>retreat()</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td>◯</td>
<td></td>
<td>◯</td>
<td>◯</td>
</tr>
</tbody>
</table>