Lecture: Skip List

- Skip list an attractive alternative to balanced search trees
- Idealized skip lists
- Implementing a skip list
- Search, Insert and Delete Operations
- Analysis

These notes are hand-written, unedited and sketchy. They are primarily used for, and based on my lectures.

If you find any bug, impreciseness, or a rare poor-/mis-interpretation of facts, please let me know. I will be grateful for any additional comments you have that are intended to make the quality of the notes better.

Please note that I will provide my hand-written lecture notes only for a subset of my lectures, not for all lectures. Therefore, it is your responsibility to attend all the lectures, take notes regularly, and ask me and/or the TAs if you have any questions.

Thank you!
--Chittu
**Skip List**

- Red-black trees guarantee $O(\log n)$ performance for:
  - **INSERT**, **DELETE**, **SEARCH** etc. Algorithms.
- However, they are not easy (quick!) to implement.
- Can we manage to get $O(\log n)$ performance from a linked list type data structure, with high probability?
- With high probability (whp): An event $E_x$ occurs whp, if $\forall x \geq 1, \exists c, E_x, \text{ s.t. } P_x[E_x] \geq 1 - \frac{c}{x^2} = 1 - O(1/n^2)$.

- **Skip list**: Bill Pugh, 1989
  - Simple, dynamic and efficient data structure
  - Randomized
  - $O(\log n)$ time operation whp.

**Building a Skip List from Scratch:**

- Start with a sorted linked list of $n$ elements (nodes).
- **SEARCH** for an element takes $\Theta(n)$ time. Can this be improved?

![Diagram of Skip List](attachment:image.png)

**SEARCH** $(22)$: follow path.

- **Question**: Which elements should be promoted to $S_1$?
- **Answer**: Goal is to minimize the worst-case performance
  - Evenly space the nodes in $S_k$.
  - But how many nodes? i.e. $|S_k| = ?$
Analysis: $|S_1| = \?$

- $\#\text{nodes searched} = \#\text{nodes searched in } S_1 + \#\text{nodes searched in } S_0$
  
  $= |S_1| + \frac{|S_0|}{|S_1|}$

  minimized when $|S_1| = \frac{|S_0|}{|S_1|}$. Then $|S_1| = \sqrt{|S_0|} = \sqrt{n}$,

  and the min. search cost $= |S_1| + \frac{|S_0|}{|S_1|} = \sqrt{n} + \frac{n}{\sqrt{n}} = 2\sqrt{n}$

### Diagram:

- $S_1$ with $\sqrt{n}$ nodes
- $S_0$ with $n$ nodes

---

Do you see the pattern here?

<table>
<thead>
<tr>
<th># Sorted Lists</th>
<th># Nodes Searched</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$2\sqrt{n}$</td>
</tr>
<tr>
<td>3</td>
<td>$3\sqrt{n}$</td>
</tr>
<tr>
<td>4</td>
<td>$4\sqrt{n}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>k</td>
<td>$k\sqrt{n}$</td>
</tr>
<tr>
<td>$\lg n$</td>
<td>$\lg n \sqrt{n} = 2\lg n = \Theta(\lg n)$</td>
</tr>
</tbody>
</table>

**Bottom Line:** With $\Theta(\lg n)$ sorted lists we get $\Theta(\lg n)$ performance.

This is an **Ideal Skip List**.
Observations:

- $S_0$ contains all the elements in SORTED order.
- For $S_i, i > 0$, $S_i \subseteq S_{i-1}$
- $\forall i > 0$, $S_i$ are SORTED.
- Tower holds copies of the same element.
- Horizontal and vertical links are as shown.
Searching Skip List:

\[
\text{SEARCH}(S, x), \quad x = 22
\]

\[
\begin{align*}
S_5 & \rightarrow 10 \\
S_4 & \rightarrow 12 \\
S_3 & \rightarrow 12 \rightarrow 16 \\
S_2 & \rightarrow 12 \rightarrow 16 \rightarrow 18 \\
S_1 & \rightarrow 10 \rightarrow 12 \rightarrow 16 \rightarrow 18 \rightarrow 20 \rightarrow 22 \rightarrow 24 \rightarrow 26 \\
S_0 & \rightarrow 10 \rightarrow 12 \rightarrow 14 \rightarrow 16 \rightarrow 18 \rightarrow 20 \rightarrow 22 \rightarrow 24 \rightarrow 26 \\
\end{align*}
\]

**SKIP-SEARCH** *(S, x)*

1. \( p \leftarrow S \)  
2. while \( p \cdot \text{below} \neq \text{NULL} \)  
3. \( p \leftarrow p \cdot \text{below} \) \quad // go down  
4. while \( x \geq p \cdot \text{next-key} \)  
5. \( p \leftarrow p \cdot \text{next} \) \quad // go right  
6. return \( p \)

Time: we'll analyze it afterwards.

Delete an element from the Skip List:

**SKIP-DELETE** *(S, x)*

1. Locate \( x \) using **SKIP-SEARCH**  
2. If \( x \) is found, then delete the tower for \( x \), and fix the pointers/references.

Note: Might have to reduce the level by 1.

Time: Same as **SKIP-SEARCH**.
Insertion:

Goal: To maintain roughly idealized skip list structure.

How? Use randomization during INSERT operations.

Observation: SkipList adds an element $x$ to the bottom list $S_0$.

Idea: Promote $x$ up some $i$ levels.

How? Flip a fair coin. If HEADS, then promote $x$ to next level up, and flip again, do the same, until the first TAILS.

Therefore, if $S_0$ has $n$ elements,
then $S_1$ has $n/2$ elements,
$S_2$ has $n/4$ elements,
$S_3$ has $n/8$ elements, etc.

**SKIP-INSERT($S$, $x$)**

1. **SKIP-SEARCH($S$, $x$)** to locate the position for $x$ in the list $S_0$
2. Insert $x$ in $S_0$
3. While the outcome of a fair coin flip is HEADS, promote $x$ to the next level, by possibly creating a new level. Also add pointers to neighbors.

**Note:** Add sentinel (marker) nodes $-\infty$ and $+\infty$ each time a new level is created.

**Time:** Same as **SKIP-SEARCH**.
Height of a Skip List:

Claim: With high probability (w.h.p. $1-O(1/n^k)$) $n$-element skip list has $O(\lg n)$ levels.

Proof:

- $h = \text{height of a skiplist } S \text{ with } n \text{ elements}$.
- $\Pr[\text{an element has tower height } i \geq 1]$
  $= \Pr[\text{getting } i \text{ consecutive heads during coin flips}]$
  $= \frac{1}{2^i}$
- $\therefore \Pr[\text{level i has at least one position}] \leq \frac{1}{2^i}$
  $1 \ 2 \ 3 \ \ldots \ n$

By union bound:

$\Pr[\bigcup_{i=1}^{n} E_i] \leq \sum_{i=1}^{n} \Pr[E_i]

\therefore \Pr[S \text{ has height } i=1] \leq \frac{n}{2^i}$

\begin{tabular}{|c|c|}
\hline
0 & So, $\Pr[S \text{ has height } i=h] \leq \frac{n}{2^i}$
\hline
\end{tabular}

Let $X = \# \text{coin flips}$

\begin{tabular}{|c|c|}
\hline
1 & $H \ H \ T \ T$
\hline
2 & $x = 4$
\hline
3 & $\Pr[X = k] = \frac{1}{2^k}$
\hline
4 & $\Pr[X > k] = \frac{1}{2^k}$
\hline
\end{tabular}

Example:

1. If $h > 3\lg n$, then $\Pr[h > 3\lg n] \leq \frac{3\lg n}{2^n} = \frac{n}{2^n} = \frac{1}{n^2}$

   $\Rightarrow \Pr[h \leq 3\lg n] \geq 1 - \frac{1}{n^2}$.

2. If $h > c\lg n$, then $\Pr[h > c\lg n] \leq \frac{c\lg n}{2^n} = \frac{n}{2^n} = \frac{1}{n^{c-1}}$

   $\Rightarrow \Pr[h \leq c\lg n] \geq 1 - \frac{1}{n^{c-1}}$, $\alpha = c - 1$.

\begin{tabular}{|c|}
\hline
0 & w.h.p.
\hline
\end{tabular}

$\therefore$ w.h.p. the height $h$ of $S$ is $O(\lg n)$. \qed
Analysis of **Skip-Search**:

- **Backward analysis**:
  - Trace the search path backward!
  - At a level \( i \), represented by \( S_i \), if the **Skip-Search** passes through the node \( z \), then looking backward, the following recurrence can be setup.

- Let \( T(i) \) = expected cost of climbing \( i \) levels.

- With probability \( p \), the search goes to left, i.e., stays at the same level \( i \), and
  - with probability \( 1-p \), the search goes one level up.

- At level 0 (i.e., \( S_0 \)), \( T(0) = 0 \).

- **Recurrence**:
  \[
  T(i) = p \cdot (T(i-1) + 1) + (1-p) \cdot (T(i) + 1)
  \]
  \[
  T(0) = 0.
  \]

- **Solving the recurrence**:
  \[
  T(i) = p \cdot (T(i-1) + 1) + (1-p) \cdot (T(i) + 1)
  \]
  \[
  \Rightarrow T(i) - (1-p) \cdot T(i) = p \cdot T(i-1) + 1
  \]
  \[
  \Rightarrow T(i) = T(i-1) + \frac{1}{p}
  \]

  Unfolding the recurrence, it is easy to see that
  \[
  T(i) = \frac{i}{p}.
  \]

- The expected height of a skip list is \( O(\log n) \).

- \( \therefore \) the expected cost of **Skip-Search** is
  \[
  T(i) \bigg|_{i=0} = O(\log n) = O(\log n) \]
  \[
  \begin{align*}
  &\quad \text{[can use}\quad p=\frac{1}{2}] \\
  \end{align*}
  \]
Space Complexity of Skip List:

- Expected number of nodes at level $i = \frac{n}{2^i}$

$\Rightarrow$ total expected number of nodes in $S$

$$\sum_{i=0}^{h} \frac{n}{2^i} = n \sum_{i=0}^{h} \frac{1}{2^i} = n \left[ \frac{1-\left(\frac{1}{2}\right)^{h+1}}{1-\frac{1}{2}} \right] = 2n \left[ 1 - \frac{1}{2^{h+1}} \right] \leq 2n,$$

$\forall h \geq 0.$

$\Rightarrow$ the expected requirement of $S$ is $O(n)$.

Note: About $2n$ space!

- Do you see a trivial (and loose) upper bound on space: $O(n \log n)$?
  - $n$ towers with expected height of each is $O(\log n)$.