Skip Lists

A skip list (due to Bill Pugh in 1990) is a randomized data structure that can be thought of as a generalization of sorted linked lists. They have most of the desirable properties of balanced binary search trees and the simplicity of linked lists. Skip lists support the standard operations for managing a dynamic data set – INSERT, DELETE, and SEARCH.

In a sorted linked list with \( n \) elements, searching an element takes \( \Theta(n) \) time. What if we had two-level sorted linked lists structure as shown in the figure below? The bottom list \( L_1 \) contains all the elements and another list \( L_2 \) contains only a subset of elements of \( L_1 \).

Note that to minimize the worst case search time, the nodes in \( L_2 \) should be evenly distributed, i.e., the number of nodes in \( L_1 \) that are “skipped” between any two consecutive nodes in \( L_2 \) should be the same.

The search time in such a 2-level structure is given by

\[
|L_2| + |L_1| = |L_2| + \frac{n}{|L_2|}
\]

To minimize the search time, the two terms in the above expression must be equal. Thus solving

\[
|L_2| = \frac{n}{|L_2|}
\]

we get \( |L_2| = \sqrt{n} \) and hence the total search time is at most \( 2\sqrt{n} \). Generalizing it a step further, consider a 3-level structure – list \( L_1 \) containing all the elements, list \( L_2 \) containing elements that are evenly distributed over elements in \( L_1 \), and list \( L_3 \) containing elements that are evenly distributed over elements in \( L_2 \). The search time in a 3-level structure is given by

\[
|L_3| + \frac{|L_2|}{|L_3|} + |L_1| + \frac{|L_2|}{|L_3|} + \frac{n}{|L_2|}
\]

The above expression is minimized when the three terms are equal. If we solve the following two equations:

\[
|L_3| = \frac{|L_2|}{|L_3|} \quad \text{and} \quad \frac{|L_2|}{|L_3|} = \frac{n}{|L_2|},
\]

we get that \( |L_3| = \sqrt[3]{n} \) and hence the search time is at most \( 3\sqrt[3]{n} \). Generalizing this to a \( k \)-level structure, we get the total search time to be at most \( k\sqrt[k]{n} \). Setting \( k = \log n \), we get the search time to be at most

\[
\log n \left( n^{\frac{1}{\log n}} \right) = \log n \left( 2^{\frac{\log n}{\log 2}} \right) = 2 \log n
\]
Thus, in a skip list with \( \lg n \) levels, the bottom list \( L_1 \) contains all of the \( n \) elements, \( L_2 \) contains \( n/2 \) elements (every other element of \( L_1 \)), \( L_3 \) contains \( n/4 \) elements, and so on. Note that this structure is like a perfectly balanced binary search tree. The insertion or deletion of a node could disrupt this structure, however just as in AVL trees, we do not need a perfectly balanced structure – a little imbalance still allows for fast search time. In skip lists the balancing is done using randomization – for each element in \( L_i \) we toss a fair coin to decide whether the element should also belong to \( L_{i+1} \) or not. In expectation, we would expect half the elements to also be part of \( L_{i+1} \). Note that the randomization in constructing this data structure does not arise because of any assumption on the distribution of the keys. The randomization only depends on a random number generator. Thus an adversary cannot pick a “bad” input for this data structure, i.e., a sequence of keys that will result in poor performance of this data structure.

The three operations \textsc{Search}, \textsc{Insert}, and \textsc{Delete} can be implemented as follows. Let \( \ell \) be the largest index of a linked list. We add a sentinel nodes \(-\infty\) to each list.

\begin{verbatim}
SEARCH(x):
    \( v_\ell \leftarrow \) element with key \(-\infty\) in \( L_\ell \).
    \textbf{for} i \:\textbf{from} \:\ell \:\textbf{down to} \:1 \:\textbf{do}
    \hspace{1cm} Follow the down-link from \( v_i \) to \( v_{i-1} \).
    \hspace{1cm} Follow the right-links starting from \( v_{i-1} \) until the key of an element is larger than \( x \).
    \hspace{1cm} \( v_{i-1} \leftarrow \) largest element in \( L_{i-1} \) that is at most \( x \).
    \textbf{return} \( v_1 \)
\end{verbatim}

The pseudo-code for the \textsc{Insert} operation is given below.

\begin{verbatim}
INSERT(x):
    \textbf{if} \( \text{SEARCH}(x) = x \) \textbf{then}
        \textbf{return}
    \( (v_\ell, v_{\ell-1}, \ldots, v_1) \leftarrow \) elements in \( L_\ell, L_{\ell-1}, \ldots, L_1 \) with key values at most \( x \)
    Flip a fair coin until we get tails. Let \( f \) be the number of coin flips.
    \textbf{for} i \:\textbf{from} \:1 \:\textbf{to} \:\min(\ell, f) \:\textbf{do}
        \hspace{1cm} Insert \( x \) in \( L_i \)
    \textbf{if} \ f > \ell \textbf{then}
        \hspace{1cm} Create lists \( L_{\ell+1}, L_{\ell+2}, \ldots, L_f \).
        \hspace{1cm} Add elements \( \{ -\infty, x \} \) to \( L_\ell, L_{\ell+1}, \ldots, L_f \).
        \hspace{1cm} Create \( L_{f+1} \) containing \( -\infty \).
        \ell \leftarrow \max(\ell, f + 1)
\end{verbatim}

The pseudo-code for the \textsc{Delete} operation is as follows.

\begin{verbatim}
Delete($x$):
if Search($x$) $\neq x$ then
    return
(v_\ell, v_{\ell-1}, \ldots, v_1) \leftarrow \text{elements in } L_\ell, L_{\ell-1}, \ldots, L_1 \text{ with key values at most } x
for i \leftarrow 1 \text{ to } \ell \text{ do}
    if $v_i = x$ then
        Delete $x$ in $L_i$
while $\ell \geq 2$ and $L_{\ell-1}$ contains only the element $-\infty$ do
    $\ell \leftarrow \ell - 1$

An example of a skip list on the set $\{30, 81, 10, 46, 53, 75, 99, 88, 93, 67, 18\}$ is as shown in the figure below. The search path for element with key 75 is highlighted.

Analysis

In the analysis below we will show that some events happen with a high probability (w.h.p). This means that the probability with which the event occurs is at least $1 - \frac{1}{n^c}$, for some constant $c \geq 1$.

Let $x_1, x_2, \ldots, x_n$ be the elements inserted into the skip list. Let $H_i$ denote the number of lists that element $x_i$ belongs to. Note that $H_i$ is exactly equal to the total number of flips of a fair coin until the first “success”. This is a geometric random variable with parameter $1/2$ and hence $E[H_i] = 2$. Let $H = \max\{H_1, H_2, \ldots, H_n\}$.

Theorem. The space requirement for a skip list with $n$ elements is $O(n)$ in expectation.

Proof. Let $S$ be the random variable denoting the total amount of space required by a skip list with $n$ elements. Taking into account the space overhead of the sentinel keys ($-\infty$), we have

$$S = O\left(\sum_{i=1}^{n} H_i\right)$$

$$E[S] = O\left(\sum_{i=1}^{n} E[H_i]\right)$$

$$= O(n)$$
Lemma 1 \( \Pr[H \geq h] \leq \frac{n}{2^{h-1}} \)

Proof We have

\[
\Pr[H \geq h] = \Pr[\bigcup_{i=1}^{n} H_i \geq h] \\
\leq \sum_{i=1}^{n} \Pr[H_i \geq h] \quad \text{(using the union-bound)} \\
= \frac{n}{2^{h-1}}
\]

Lemma 2 The expected number of levels in a skip list with \( n \) elements is at most \( \lceil \lg n \rceil + 3 \).

Proof The number of levels in a skiplist of \( n \) elements is \( H + 1 \). The second term is due to the list with the highest index that contains only the key \(-\infty\). Since \( H \) is a non-negative random variable, we have

\[
\mathbb{E}[H] = \sum_{h=1}^{\lfloor \lg n \rfloor} \Pr[H \geq h] \\
= \sum_{h=1}^{\lfloor \lg n \rfloor} \Pr[H \geq h] + \sum_{h=\lceil \lg n \rceil + 1}^{\infty} \Pr[H \geq h] \\
\leq \sum_{h=1}^{\lfloor \lg n \rfloor} 1 + \sum_{h=\lceil \lg n \rceil + 1}^{\infty} \frac{n}{2^{h-1}} \\
\leq \lceil \lg n \rceil + \sum_{h=\lceil \lg n \rceil + 1}^{\infty} \frac{n}{2^{h-1}} \\
= \lceil \lg n \rceil + \frac{n}{2^{\lceil \lg n \rceil + 1}} \sum_{i=0}^{\infty} \frac{1}{2^i} \\
= \lceil \lg n \rceil + 2
\]

Thus the expected number of levels is \( \mathbb{E}[H] + 1 \leq \lceil \lg n \rceil + 3 \).

Lemma 3 With a high probability, a skip list with \( n \) elements has \( O(\log n) \) levels.

Proof. We will show that w.h.p, the number of levels in a skip list with \( n \) elements is at most \( 2 \lg n + 1 \).

This is equivalent to showing that w.h.p, \( H \leq 2 \lg n \). From Lemma 1, we have

\[
\Pr[H \geq 2 \lg n + 1] \leq \frac{n}{2^{2\lg n + 1 - 1}} = \frac{1}{n}
\]