Skip Lists

Skip lists (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 $L_1$.

\[
\begin{align*}
L_2 & \quad 10 \quad \rightarrow \quad 53 \quad \rightarrow \quad 88 \\
L_1 & \quad 10 \quad \rightarrow \quad 18 \quad \rightarrow \quad 30 \quad \rightarrow \quad 46 \quad \rightarrow \quad 53 \quad \rightarrow \quad 67 \quad \rightarrow \quad 75 \quad \rightarrow \quad 81 \quad \rightarrow \quad 88 \quad \rightarrow \quad 93 \quad \rightarrow \quad 99
\end{align*}
\]

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

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

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

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

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 \( k\sqrt[3]{n} \). Setting \( k = \lg n \), we get the search time to be at most

\[
\lg n (n^{\frac{1}{\lg n}}) = \lg n (2^{\frac{\lg n}{\lg n}}) = 2 \lg n
\]

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.