CMSC 420: Skip Lists

Justin Wyss-Gallifent

April 10, 2025

1 Introduction

Skip lists were invented in 1990 by Bill Pugh, at UMD. These are a modification of a regular sorted linked list which provides logarithmic search, insertion and deletion, much like AVL trees. On the down side they require about twice as much storage.

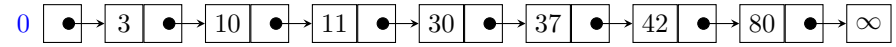
They do this by introducing an element of randomness to the sorted linked list construction, as we'll see.

2 Ideal Skip Lists

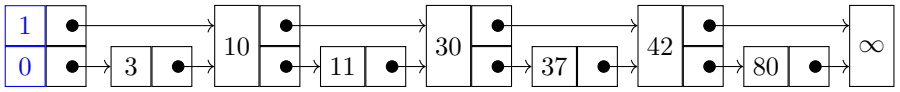
Suppose we have a sorted linked list with n keys. Clearly it takes $\mathcal{O}(n)$ time to find, insert and delete (insertion and deletion since we must find first). Is there a way to speed this up?

Let's start by hypothesizing a sorted linked list with just 8 elements. Suppose we also have a head node which is simply a pointer to the first element in the list and a tail node which is pointed to by the last element in the list.

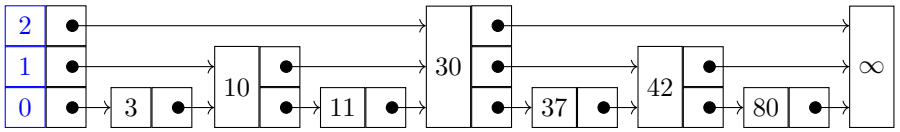
In the picture below ignore the 0 in the head node for now, don't treat it as a key in the list.



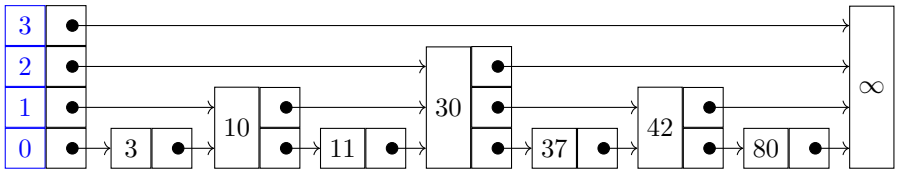
As mentioned at the start worst-case for find, insert, and delete is $\mathcal{O}(n)$ in this case. But suppose we added a "fast level" of pointers. To elaborate, think of the 0 in the head node above as level 0, the really slow level. We'll add a fast level of pointers at level 1 which skips every other node in level 0:



Great, so what if we added an extra fast level - level 2:



And last but not least the fastest level of all - level 3:



Okay, let's pause for a second to see why this might be useful. Suppose we are looking for the key 37.

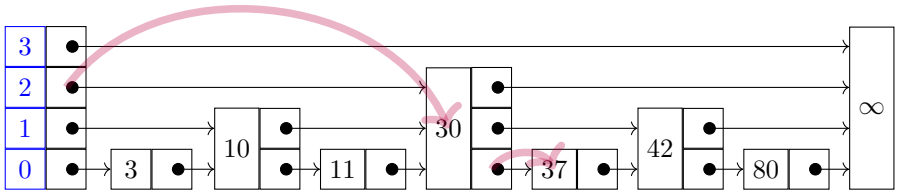
We start at the top pointer in level 3 and observe that it points to ∞. This is too large so we drop down to level 2.

We check the pointer in level 2 and observe that it points to 30. This is below our target of 37 so we follow the pointer to that node. We then check the next pointer in level 2 and observe that it points to ∞. This is too large and so we drop down to level 1.

We check the next pointer in level 1 and observe that it points to 42. This is too large and so we drop down to level 0.

We check the next pointer in level 0 and observe that it points to 37. This is our target and we're done!

Here is an illustration:



In an ideal skip list with n elements we would have level 0 which links all nodes, level 1 which skips every other node in level 0, level 2 which skips every other node in level 1, and so on until adding more levels adds nothing. Then we would find our target key using a generalization of the above approach.

3 Randomized Skip Lists

This is all very quaint but the truth of the matter is that once a key is inserted or deleted the entire structure is ruined and we certainly don't want to have to rebuild all the pointers every time this happens.

Instead then we'll add an element of randomness to the procedure as follows. First we set an *enforced maximum level*. We create a head node which reaches this enforced maximum level and has no value (sometimes $-\infty$) and we create a tail node which also reaches this enforced maximum level and has a value of ∞.

Note that in all of what follows, as with our non-randomized case, the levels are 0-indexed starting at the bottom.

1. In level 0 we will have a regular linked list, starting with the head node, progressing through our keys, and ending with the tail node.
2. In level 1 we take each internal node which reaches level 0 (all of them) and say that there is a $p = 0.5$ probability that it also reaches level 1. We then form a linked list using these nodes, again starting with the head node, progressing through the nodes that do extend to level 1, and ending

with the tail node.

3. In level 2 we take each internal node which reaches level 1 (about half of them) and say that there is a $p = 0.5$ probability that it also reaches level 2. We then form a linked list using these nodes, again starting with the head node, progressing through the nodes that do extend to level 2, and ending with the tail node.
4. In level 3 we take each internal node which reaches level 2 (about quarter of them) and say that there is a $p = 0.5$ probability that it also reaches level 3. We then form a linked list using these nodes, again starting with the head node, progressing through the nodes that do extend to level 3, and ending with the tail node.
5. We continue this until we "run out of" nodes. Note that in theory nodes could keep being included in higher and higher levels albeit with lower and lower probability so we stop for sure when we hit our enforced maximum level.

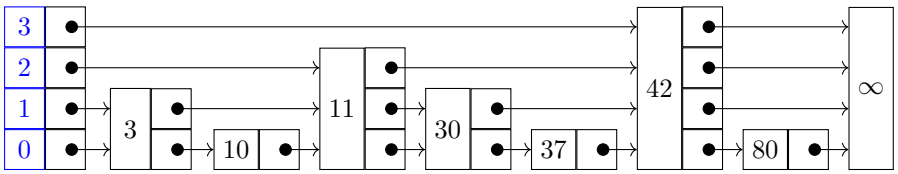
A few definitions. The first is repeated.

Definition 3.0.1. We have:

- The *(enforced) maximum level* is the upper limit that we set on how far each node could reach.
- The *top level for the skip list* is the topmost level that we actually reach.
- The *top level* for a given node is simply the largest level index it reaches.

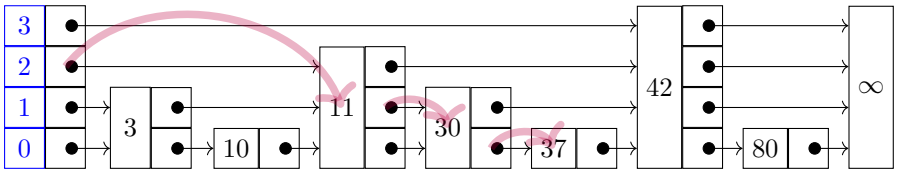
The latter two are of course less than or equal to the enforced maximum level.

Example 3.1. A randomized skip list might look like the following. The top levels of the nodes were generated by flipping a coin with an enforced maximum level of 3 (indexed 0 to 3):



So now if we're looking for 37 our path will be $H \longrightarrow 11 \longrightarrow 30 \longrightarrow 37$ and our path to 80 will be a really fast $H \longrightarrow 42 \longrightarrow 80$.

Here's the path for 37:



4 Measurements

4.1 Important Note

All of the analysis here is based upon having no enforced maximum level.

4.2 Levels and Node Counts

Theorem 4.2.1. Regarding levels in a skip list with probability p and with n nodes:

(a) For a given level $i \geq 0$, the probability that at least one node reaches level i or beyond equals:

$$1 - \left(1 - p^i\right)^n$$

Proof. For any level $i \geq 0$, the probability that a single node reaches level i (and possibly beyond) is p^i and so the probability that a single node does not reach level i (or beyond) is $1 - p^i$.

Because the nodes are independent, it follows that the probability that none of the nodes reaches level i (or beyond) equals:

$$\left(1 - p^i\right)^n$$

Hence the probability that at least one node does reach level i (or beyond) is:

$$1 - \left(1 - p^i\right)^n$$

$\mathcal{QED}$

Example 4.1. For $p = 0.5$ the probability that at least one node reaches level $L = 10$ (or beyond) equals:

$$1 - \left(1 - 0.5^{10}\right)^{100} \approx 0.09308 \approx 9.308\%$$

(b) For $i \geq 0$ denote by L_i the number of nodes in level i of a skip list. Then $E(L_i) = np^i$.

Proof. For each node there is a p^i probability it reaches level i. Since each node grows independently and there are n nodes we expect there to be np^i nodes in level i. $\mathcal{QED}$

(c) Let N be the number of levels in a skip list with n nodes. Then:

$$E(N) = \mathcal{O}(\lg n)$$

Proof. Here is the proof for $p = 0.5$. The proof needs some tweaking for other p but the final result still holds.

For $i \geq 0$ define I_i to be 1 if there are nodes in level i of the skip list and 0 otherwise. It follows then that:

$$N = \sum_{i=0}^{\infty} I_i$$

Our claim is then that $E(N) = \mathcal{O}(\lg n)$.
First observe that:

$$\begin{aligned}
E(N) &= E\left(\sum_{i=0}^{\infty} I_i\right) \\
&= \sum_{i=0}^{\infty} E(I_i) \\
&= \sum_{i=0}^{\lfloor \lg n \rfloor} E(I_i) + \sum_{i=\lfloor \lg n \rfloor + 1}^{\infty} E(I_i) \\
&\leq \sum_{i=0}^{\lfloor \lg n \rfloor} 1 + \sum_{i=\lfloor \lg n \rfloor + 1}^{\infty} E(L_i) \\
&\leq 1 + \lfloor \lg n \rfloor + \sum_{i=\lfloor \lg n \rfloor + 1}^{\infty} n(0.5)^i \\
&\leq 1 + \lg n + n(0.5)^{\lfloor \lg n \rfloor + 1} \sum_{i=0}^{\infty} (0.5)^i \\
&\leq 1 + \lg n + \frac{n}{2^{\lfloor \lg n \rfloor + 1}}(2)
\end{aligned}$$

Then observe that:

$$2^{\lfloor \lg n \rfloor + 1} \geq 2^{\lg n} = n$$

And thus:

$$E(NL) \leq 1 + \lg n + 1(2) = 3 + \lg n$$

$\mathcal{QED}$

4.3 Storage

Theorem 4.3.1. Denote by $S(n)$ the storage needed for a skip list with n entries. Then the expected storage is $E(S(n)) = \mathcal{O}(n)$.

Proof. As mentioned we are going to ignore the enforced maximum level here but we are also going to ignore the head and tail nodes. However you are encouraged to think about the fact that this doesn't affect the outcome.

First off, the keys take up $\Theta(n)$ space for n nodes so the real issue is how much space the pointers take up.

For each level $i = 0, 1, 2, ...$ there is a $1/2^i$ probability that each node reaches level i and hence we expect $n/2^i$ pointers at level i.

Consequently the storage needed is essentially the sum:

$$E(S(n)) = \sum_{i=0}^{\infty} \frac{n}{2^i} = 2n$$

$\mathcal{QED}$

Note 4.3.1. In the computation above we are using the fact that $E(x + y) = E(x) + E(y)$ for expected value calculations, a basic fact from probability.
Theorem 4.3.2. The worst-case storage without an enforced maximum level is infinite.

Proof. There is no bound on the number of pointers for any one node, let alone all n of them! $\mathcal{QED}$

5 Search

5.1 Algorithm

The search process is very easy. We start at the header node all the way on the left, at the highest level. We follow this level across until the final node before we overshoot our target. When we reach this node, we drop down a level and repeat.

Essentially we are staying in the topmost level as long as possible and then dropping down when we would miss the target.

5.2 Time Complexity

It turns out that the average case time complexity for search is $\mathcal{O}(\lg n)$.

The analysis of this is a bit sneaky. We will follow the search process in reverse, starting at the target node and working our way back to the top of the header node. As a precursor let's refine our description of the search procedure into *steps*. If we are at a certain location in the skip list (meaning a certain level of a certain node) then a step consists of either:

- Going up a level if we can; There is a $p = 0.5$ probability of this.
- Going left if we cannot; There is then also a $1 - p = 0.5$ probability of this.

Observe that a step (interpreted forward) takes time $\Theta(1)$.
Theorem 5.2.1. The expected search time in a skip list with n nodes is $\mathcal{O}(\lg n)$.

Proof. Here is the proof for $p = 0.5$. The proof needs some tweaking for other p but the final result still holds.

We divide the search procedure into steps up (going up a level) and steps left (going left by reverse-following a pointer). If as earlier we denote by N the number of levels in the skip list then we know that $E(N) = \mathcal{O}(\lg n)$ and since the number of steps up is at most $N - 1$ we know that this will be $\mathcal{O}(\lg n)$ as well. Since each step is $\Theta(1)$ we know the steps up take time $\mathcal{O}(\lg n)$.

Now let's consider the number of steps left. If we denote by S_i the number of steps taken left in level i and since there is one final step back to the header node then we are trying to calculate the expected value:

$$E\left(1 + \sum_{i=0}^{\infty} S_i\right)$$

Observe the following facts, recalling that L_i denotes the number of nodes in level i:

- Since $S_i \leq L_i$ we know that $E(S_i) \leq E(L_i) = n(0.5)^i$.
- At a given node the probability of moving left exactly 0 steps is 0.5, exactly 1 step is 0.5^2, and so on. Thus the expected number of steps taken to the left in any given level is certainly less than the following which would require lists of infinite length:

$$E(S_i) \leq \sum_{i=0}^{\infty} i(0.5)^{i+1} = ... = 1$$

Thus we have:

$$\begin{aligned}
E\left(1 + \sum_{i=0}^{\infty} S_i\right) &= 1 + \sum_{i=0}^{\infty} E(S_i) \\
&= 1 + \sum_{i=0}^{\lfloor \lg n \rfloor} E(S_i) + \sum_{i=\lfloor \lg n \rfloor + 1}^{\infty} E(S_i) \\
&\leq 1 + \sum_{i=0}^{\lfloor \lg n \rfloor} 1 + \sum_{i=\lfloor \lg n \rfloor + 1}^{\infty} n(0.5)^i \\
&\leq 1 + 1 + \lg n + n(0.5)^{\lfloor \lg n \rfloor + 1} \sum_{i=0}^{\infty} (0.5)^i
\end{aligned}$$

The final part of this is handled exactly as with the earlier proof of $E(N)$ yielding:

$$E\left(1 + \sum_{i=0}^{\infty} S_i\right) \leq 4 + \lg n$$

Again since each step is $\Theta(1)$ we know the steps left take time $\mathcal{O}(\lg n)$.

Thus the total time taken is $\mathcal{O}(\lg n)$. $\mathcal{QED}$

Theorem 5.2.2. Search is best case $\mathcal{O}(1)$, average-case $\mathcal{O}(\lg n)$, and worst-case $\mathcal{O}(n)$.

Proof. We have just proven the average-case. The best case is if our target is at the end of the topmost pointer in the header node, thus taking one step to find. The worst case is if the skip list is a linked list and we are looking for the final element. $\mathcal{QED}$

6 Insertion Algorithm and Time Complexity

We need to develop an algorithm to insert a new key/node x. The algorithm proceeds as follows:

1. Figure out the top level of this node using the $p = 0.5$ approach.
2. We then search for x. Since x is not in the list we will overshoot it repeatedly down to and including level 0. For each pointer which overshoots it if that pointer is on some level i and if our new node exists at level i then we break that pointer up to point to our new node and from our new node to the original target.

In terms of time complexity:

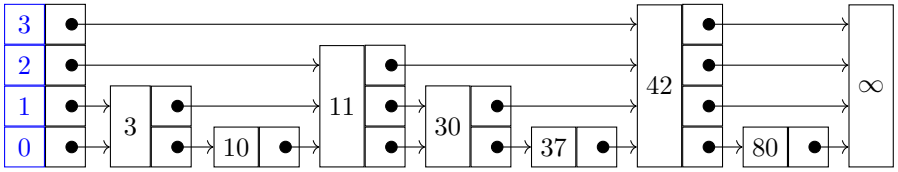
1. The expected top level is:

$$\sum_{i=0}^{\infty} i \cdot 0.5^i = ... = 2$$

   so this is certainly average-case $\mathcal{O}(1)$. Note that it is best-case $\mathcal{O}(1)$ and assuming we have a maximum enforced level it is worst-case $\mathcal{O}(1)$ too. Without a maximum enforced level it would be worst-case $\mathcal{O}(\infty)$.
2. Since search is average case $\mathcal{O}(\lg n)$ and since we are re-wiring at most N pointers at constant time apiece and since there are at most N pointers and $E(N) = \mathcal{O}(\lg n)$ we know that this is average-case $\mathcal{O}(\lg n)$. Best-case this would be $\mathcal{O}(1)$ and worst-case would be $\mathcal{O}(n)$ assuming a maximum enforced level.

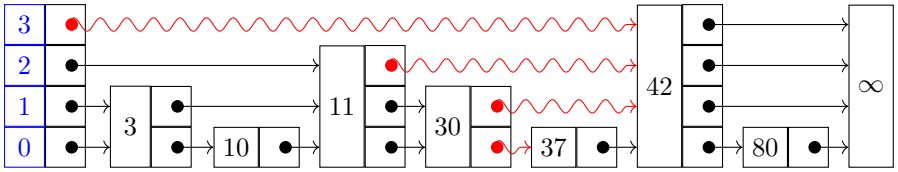
Thus overall we have best-case $\mathcal{O}(1)$, average-case $\mathcal{O}(\lg n)$, and worst-case $\mathcal{O}(n)$.

Example 6.1. Consider this skip list from earlier:

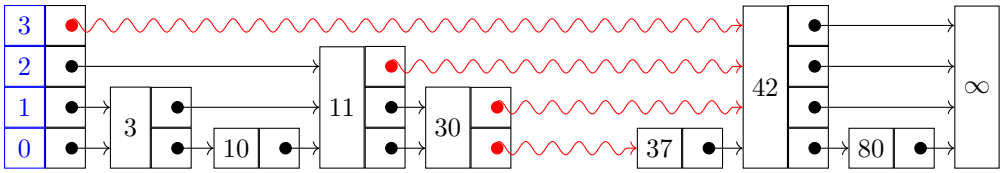


Suppose we want to insert 32 into this skip list and randomization tells us the top level should be 3.

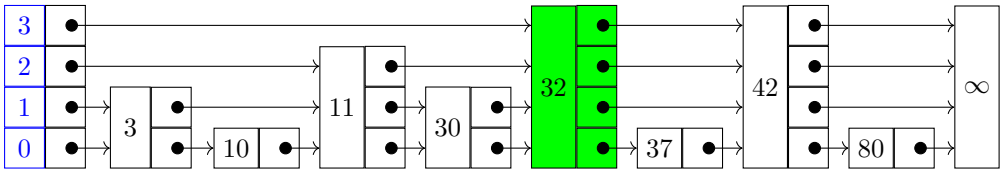
When we traverse looking for 32 we end up at 37. Here are the pointers which overshoot 32 and which exist on a level which 32 reaches:



We then make space for our new node:



And we wire up the pointers:



7 Deletion Algorithm and Time Complexity

We need to develop an algorithm to delete a key x. The algorithm proceeds as follows:

1. We search for x but we don't follow the pointers which point to it. Rather for each pointer we encounter which points to x we join that pointer with its subsequent pointer on the same level.

In terms of time complexity:

1. Since search is average case $\mathcal{O}(\lg n)$ and since we are joining at most N pointers at constant time apiece and since there are at most N pointers and $E(N) = \mathcal{O}(\lg n)$ we know that this is average-case $\mathcal{O}(\lg n)$. Best-case this would be $\mathcal{O}(1)$ and worst-case would be $\mathcal{O}(n)$ assuming a maximum enforced level.

Thus overall we have best-case $\mathcal{O}(1)$, average-case $\mathcal{O}(\lg n)$, and worst-case $\mathcal{O}(n)$.

Example 7.1. In the example above if we wished to delete 11 we would search for 11 but we don't follow the pointers which point to it. The pointers we encounter which point to 11 and which are then joined are:

1. $2 \longrightarrow 11$ is joined to $11 \longrightarrow 32$.
2. $3 \longrightarrow 11$ is joined to $11 \longrightarrow 30$.
3. $10 \longrightarrow 11$ is joined to $11 \longrightarrow 30$.