CMSC 420: Skip Lists

Justin Wyss-Gallifent

October 20, 2022

1 Introduction .................................................. 2
2 Ideal Skip Lists ............................................... 2
3 Randomized Skip Lists ...................................... 3
4 Measurements ................................................ 4
  4.1 Important Note ........................................... 4
  4.2 Levels ...................................................... 4
  4.3 Storage ..................................................... 7
5 Search .......................................................... 8
  5.1 Algorithm .................................................. 8
  5.2 Time Complexity ......................................... 8
6 Insertion ....................................................... 9
  6.1 Algorithm .................................................. 9
  6.2 Time Complexity ......................................... 10
7 Deletion ....................................................... 10
  7.1 Algorithm .................................................. 10
  7.2 Time Complexity ......................................... 10
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 and values. Clearly it takes $O(n)$ 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 $O(n)$ in this case. But suppose we added a “fast lane” of pointers. To elaborate, think of the 0 in the head node above as the 0-lane, the really slow lane. We’ll add a fast lane of pointers called the 1-lane which skips every other node in the 0-lane:

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

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

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 the 3-lane and observe that it points to $\infty$. This is too large so we drop down to the 2-lane.

We check the pointer in the 3-lane 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 the 3-lane and observe that it points to $\infty$. This is too large and so we drop down to the 2-lane.

We check the next pointer in the 2-lane and observe that it points to 42. This is too large and so we drop down to the 1-lane.

We check the next pointer in the 1-lane 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 a 0-lane which links all nodes, a 1-lane which skips every other node in the 0-lane, a 2-lane which skips every other node in the 1-lane, and so on until adding more lanes 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 and we create a tail node which also reaches this enforced maximum level and has a value of $\infty$.

1. In the 0-lane we will have a regular linked list, starting with the head node, progressing through our keys, and ending with the tail node.

2. In the 1-lane we take each internal node in the 0-lane and say that there is a $p = 0.5$ probability that it is also in the 1-lane. 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 the 2-lane we take each internal node in the 1-lane and say that there is a $p = 0.5$ probability that it is also in the 1-lane. 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. We continue this until we “run out of” nodes. Note that in theory nodes could keep being included in higher and higher lanes albeit with lower and lower probability so we stop for sure when we hit our enforced maximum level.

Example 3.1. A randomized skip list might look like the following. The levels 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 $H \rightarrow 11 \rightarrow 30 \rightarrow 37$ and our path to 80 will be a really fast $H \rightarrow 42 \rightarrow 80$.

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. Having an enforced maximum level doesn’t change the $O$ bounds.

4.2 Levels

Theorem 4.2.1. Regarding levels in a skip list with $n$ nodes:

(a) The probability that at least one node reaches some level $L \geq 0$ (or beyond) equals:

$$ P = 1 - \left(1 - \frac{1}{2^L}\right)^n $$

(b) The probability that at least one node reaches some level $L \geq 0$ (or beyond) can be bounded:

$$ P \leq \frac{n}{2^L} $$
(c) The expected maximum level in a skip list with \( n \) nodes satisfies:

\[ E(ML(n)) = \Theta(\lg n) \]

(d) The expected maximum level in a skip list with \( n \) nodes satisfies:

\[ E(ML(n)) = 1 + \lg n \]

(e) The expected number of levels in a skip list with \( n \) nodes equals \( 2 + \lg n \).

**Proof.** We have:

(a) For any level \( L \), the probability that a single node reaches level \( L \) (and possibly beyond) is \( 1/2^L \) and so the probability that a single node does not reach level \( L \) (or beyond) is \( 1 - 1/2^L \).

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

\[ \left(1 - \frac{1}{2^L}\right)^n \]

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

\[ 1 - \left(1 - \frac{1}{2^L}\right)^n \]

(b) This is just a slightly easier-to-use bound than (a). Again since the probability that a single node reaches level \( L \) (or beyond) is \( 1/2^L \) it follows that the probability that any of the \( n \) nodes reach level \( L \) (or beyond) is at most the sum of this expression \( n \) times, or just \( n/2^L \).

(c) Denote by \( E(ML(n)) \) the expected maximum level for a skip list with \( n \) nodes. For such a list if we go one level up we expect \( n/2 \) nodes and so:

\[ E(ML(n)) = E(ML(n/2)) + 1 \]

The Master Theorem tells us that \( E(ML(n)) = \Theta(\lg n) \).

(d) We can do better than (e) though. Consider \( E(ML(1)) \). There is a probability of \( 1/2^L \) that the single node reaches level \( L \) (or beyond) and hence the probability that a single node reaches exactly level \( L \) and no further equals:

\[ \frac{1}{2^L} - \frac{1}{2^{L+1}} = \frac{1}{2^{L+1}} \]

Thus:

\[ E(ML(1)) = \sum_{L=0}^{\infty} L \left(\frac{1}{2^{L+1}}\right) = \ldots = 1 \]
From there we can dig down with the recurrence relation. For simplicity assume $n$ is a power of 2 and then:

\[
E(ML(n)) = E(ML(n/2)) + 1 \\
= E(ML(n/4)) + 2 \\
= E(ML(n/8)) + 3 \\
= ... \\
= E(ML(n/n)) + \log n \\
= 1 + \log n
\]

(e) Part (e) follows immediately from (d) since the levels are 0, 1, 2, ..., $1 + \log n$.

\[ \Box \]

Note 4.2.1. The sum in there is a bit sneaky:

\[
\sum_{L=0}^{\infty} L \left( \frac{1}{2L+1} \right) = \sum_{L=1}^{\infty} L \left( \frac{1}{2L+1} \right) \\
= 2 \sum_{L=1}^{\infty} L \left( \frac{1}{2L+1} \right) - \sum_{L=1}^{\infty} L \left( \frac{1}{2L} \right) \\
= \sum_{L=1}^{\infty} L \left( \frac{1}{2L} \right) - \sum_{L=1}^{\infty} L \left( \frac{1}{2L+1} \right) \\
= \left[ 1 \left( \frac{1}{2} \right)^1 + 2 \left( \frac{1}{2} \right)^2 + 3 \left( \frac{1}{2} \right)^3 + ... \right] - \left[ 1 \left( \frac{1}{2} \right)^2 + 2 \left( \frac{1}{2} \right)^3 + 3 \left( \frac{1}{2} \right)^4 + ... \right] \\
= \left( \frac{1}{2} \right)^1 + \left( \frac{1}{2} \right)^2 + ... \\
= 1
\]

Example 4.1. Here are some consequential statistics for the case where we have 100 nodes:

(a) The probability that at least one node reaches level $L = 10$ (or beyond) equals:

\[
P = 1 - \left( 1 - \frac{1}{2^{10}} \right)^{100} \approx 0.09308265650895886
\]

(b) The probability that at least one node reaches level $L = 10$ (or beyond) can be bounded:

\[
P \leq \frac{100}{2^{10}} \approx 0.09765625
\]

(c) Not really specific.
(d) The expected maximum level in this skip list satisfies:

\[ E(ML(n)) = 1 + \lg 100 \approx 7.6438561897747 \]

(e) The expected number of levels in this skip list equals 2 + \lg n \approx 8.6438561897747.

Moreover it’s worth noting that even if we didn’t set an enforced maximum level
for our nodes we still have a very low probability of achieving a high number of
levels. For example with \( n = 100 \) nodes the probability of reaching level 15 is
only:

\[ P = 1 - \left(1 - \frac{1}{2^{10}}\right)^{100} \approx 0.0030471523581468984 \]

This feels somewhat intuitive given that the expected maximum level is about
7.6.

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 \( E(S(n)) = O(n) \).

**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 \( i = 0, 1, 2, \ldots \) 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 the sum:

\[ E(S(n)) = E\left(S\left(\sum_{i=0}^{\infty} \frac{1}{2^i}\right)\right) = 2n \]

\( \text{QED} \)

**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!

\( \text{QED} \)
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 fastest lane as long as possible and then dropping down when we would miss the target.

5.2 Time Complexity
It turns out that the time complexity for search is $O(\lg n)$.

The analysis of this is a bit sneaky. We assume we’re at the target and consider tracing back to the head node. It’s easier to approach the problem this way because we progress as follows:

(i) Go to the highest level of the node we are currently at.
(ii) Go left to the previous node.
(iii) Go back to (i). Continue until we reach the head node.

Theorem 5.2.1. The expected number of nodes visited in a skip list with $n$ nodes is $O(\lg n)$.

Proof. Assume that the skip list has maximum level $L$. This could be because we enforced this level or because the nodes stopped growing there. For each $i$ let $E(V(i))$ equal the expected number of nodes visited in the top $i$ levels of a skip list. Let’s pretend that the skip list extends infinitely far left. This is of course unrealistic but will suffice.

At any given current node we have a 0.5 probability of moving up a level, meaning that we will visit the current node and then $E(V(i-1))$ more nodes, and we have a 0.5 probability of staying at the same level, meaning that we will visit the current node and then $E(V(i))$ more nodes.

This means we have:

$$E(V(i)) = 0.5(1 + E(V(i-1))) + 0.5(1 + E(V(i)))$$

We can rewrite this with basic algebra:

$$E(V(n)) = 0.5(1 + E(V(i-1))) + 0.5(1 + E(V(i)))$$
$$E(V(n)) = 1 + 0.5E(V(i-1)) + 0.5E(V(i))$$
$$0.5E(V(i)) = 1 + 0.5E(V(i-1))$$
$$E(V(i)) = E(V(i-1)) + 2$$
Along with the fact that $E(V(0)) = 0$ (in the top 0 levels there are 0 nodes) this can be expanded to show that $E(V(i)) = 2i$:

$$
E(V(i)) = E(V(i - 1)) + 2 \\
= E(V(i - 2)) + 4 \\
= E(V(i - 3)) + 6 \\
= \vdots \\
= E(V(i - i)) + 2i \\
= 0 + 2i
$$

Returning to our assumption that the list continues infinitely far left we now comment that this is not of course true, and so $E(V(i)) \leq 2i$. It follows then that for a skip list where the maximum level is $L$ and since we are starting our backwards journey at level 0 we expect to visit $E(V(L + 1)) \leq 2(L + 1)$ nodes. As we have shown that we expect $O(lg n)$ levels we also know that $E(V(L + 1)) = O(lg n)$.

QED

**Theorem 5.2.2.** Search is average case $O(lg n)$ and worst-case $O(n)$.

*Proof.* Assuming it takes constant time to visit/process one node and since on average we visit $O(lg n)$ nodes the first result follows immediately. The second result follows from the fact that there is a probability, albeit very low, that the skip list turns out to be a regular linked list.

QED

**Note 5.2.1.** It seems to be reasonable to argue that visiting/processing a node might take more time. After all, we need to figure out which level to follow and this requires scanning all of the levels until we find an appropriate one.

Consider though that we have set a maximum level which is constant and independent of $n$, so visiting/processing a node is in fact $O(n)$.

### 6 Insertion

**6.1 Algorithm**

Insertion is quite easy. We first insert the new node into the correct place in the 0-lane. Then there is a 0.5 probability that it is also included in the 1-lane. If it is, we insert it there and then continue; There is a 0.5 probability that it is also included in the 2-lane. And so on until we reach our predetermined maximum.
6.2 Time Complexity

Insertion time complexity is based upon finding the insertion point, hence this is $O(\lg n)$ average case and $O(n)$ worst-case.

Note that the actual insertion, once the location is found, is $\Theta(1)$, because we have placed a maximum on the number of levels.

7 Deletion

7.1 Algorithm

Deletion is even easier - we simply delete the node and patch up the linked lists that it was in.

7.2 Time Complexity

Deletion time complexity is based upon finding the node to be deleted, hence this is $O(\lg n)$ average case and $O(n)$ worst-case.

Note that the actual deletion, once the location is found, is $\Theta(1)$, because we have placed a maximum on the number of levels.