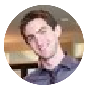
Why aren't skip lists used more often instead of balanced trees?

http://en.wikipedia.org/wiki/Skip_list

6 Answers


Mark Gordon, aka msg555, CS PhD at U. Michigan
Answered Nov 7, 2012

So these were the only reasons I could come up with. Skip lists, it seems to me, are just an alternative way of representing a search tree.

1) Worst case logarithmic running time. Hands down the most obvious reason. Why open yourself up to DOS attacks when the worst case solution is readily available and works well?

2) If you don't care about 1, then why not just use splay trees? Splay trees are simple to implement, automatically adapt to your query distribution, and are perhaps one of the most malleable data structures imaginable (for example, implementing Euler tour trees on top of them is a snap). Plus you'll make Danny Sleator happy.

4.1k Views · 6 Upvotes

Related Questions

More Answers Below

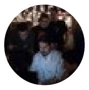
When do we prefer to use Skip List compared to balanced search tree (AVL etc)?

Why should I use skip lists over binary search trees?

What uses are there for Skip Lists?

What are the disadvantages of skip lists?

What is an example/use case where skip list has been used?


Edward Kmett, (|computer scientist〉 + |mathematician〉)/√2
Answered Aug 29, 2015

Binary trees are easily manipulated functionally with path-copying. When you modify a node in the tree, you can copy the path to the root just as easily as modifying the tree locally. This may turn O(1) space worth of edits into O(log n) space worth of path-copied nodes, but it doesn't affect your time bound.

Why am I rambling on about trees when you asked about skip-lists?

Well, you can twist around your thinking about a skip-list to see it as a form of binary tree with some extra links you don't use for the insert/delete/search operations thrown in.

I gave an answer on stackoverflow about functional skip-lists that showed this connection more clearly.

Anyways, if you care about functional data structures then skip-lists tend not to fit in unless you do something like that transformation, and then they are no better than any other tree with O(log n) height, except that viewed this way as a tree they have height O(log n) "with high probability" not just in expectation. This can be useful when analyzing using many of them in the execution of another algorithm.

5.9k Views · 7 Upvotes

Debasish Ghosh, Programmer at large
Updated Oct 6, 2012

As far as efficiency is concerned skip lists are comparable to balanced search trees and has a simpler implementation as well. However there are the following negatives (IMHO) that make them not as frequently used as trees (or even hashes):

- skip lists take more space than a balanced tree. I think a skip list takes 4 pointers per node on an average, which is twice as much as a tree.
- skip lists are not cache friendly because they don't optimize locality of reference i.e. related elements tend to stay far apart and not on the same page.
- lack of available implementations. There are far more optimized tree implementations than skip lists.

8.7k Views · 34 Upvotes

Mark Nelson, Engineer at Cisco Systems
Answered Nov 9, 2010

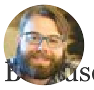
I personally think it is simply a matter of novelty. Balanced trees have been around since the dawn ages of computing, which means guys like me had at least a quick mention of something like an AVL tree back in the punchcard days.

Skip lists are pretty recent, and probably didn't start getting any real attention until perhaps 15 years ago. And while they are arguably more efficient than balanced trees, they aren't that much more efficient - they have the same asymptotic behavior.

Check back in 20 years and you might see a change.

3.1k Views · 7 Upvotes

Jaap Weel, Programmer. Economist.


Answered Oct 27, 2014

Because it's very rare that your finite maps need to be both ordered and thread-safe, and that's really the only case that skip lists are *particularly* good for.

There are basically four flavors of maps commonly encountered:

- **Single-threaded unordered maps** are most easily implemented as hash tables, especially if you have a good estimate of their size so you don't have to dynamically resize them a lot.
- **Thread-safe unordered maps** are also most easily implemented as hash tables, since it's easy to reduce lock contention simply by splitting the table into parts ("stripes") and locking those separately.
- **Single-threaded ordered maps** are best implemented using trees. They can be made extremely space-efficient, their cache performance can be optimized, etc.
- **Thread-safe ordered maps** are where skip lists really shine. You can even make them lock-free. Research in this stuff continues and there's new exotic trees coming out all the time, but skip lists have the great advantage that they are comparatively easy to understand.

Given all this, it shouldn't surprise you that in Java, the standard **Set<T>** implementations are **HashSet<T>** and **ConcurrentHashSet<T>**, but the standard **NavigableSet<T>**implementations are **TreeSet<T>** and... **ConcurrentSkipListSet** **<T>**! (The situation for Map<K,V> and NavigableMap<K,V> is the same.)

It's just that in practice, when I use finite maps, I find that they only need to be concurrent some of the time, and they only need to be ordered some of the time, and they need to be both concurrent and ordered... pretty much never.

But if ever one needs to be, I'll be sure to reach for a skip list!

3.6k Views · 29 Upvotes

Eugene Yarovoi, Tech Interviews and Competitive Programming Meetup organizer
Updated May 13, 2014

It's probably mostly because they're less known, with fewer available libraries. People may also be slightly afraid of randomized algorithms in some contexts.

There are at least some claims that skip lists can be better than trees for certain things. Some skip list implementations appear to perform very well under concurrency. This Stack Overflow answer has some interesting information about this: Skip List vs. Binary Tree .

I have not fully evaluated all the information in the thread, but my initial impression is that, while skip lists may outperform some types of balanced trees under heavy concurrent modifications, it is mostly trees whose balancing algorithms are based on tree rotations that have concurrency problems. Trees that split and merge nodes, like B trees or B+ trees, should be a lot better than rotation-based trees in this regard.

3.2k Views · 7 Upvotes