Skip Lists

Nota acerca del uso de Skip lists (fuente Coba Weel en Quora)

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**](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentSkipListSet.html)

**<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!

Una **Skip List** se construye por capas. La capa del fondo (la más baja) es una sencilla lista enlazada. Cada capa subsiguiente es como una "vía rápida" para la lista de la capa bajo esta. Un elemento de la capa *i* aparece en la capa *i*+1 con una probabilidad fija *p*.

En promedio, cada elemento aparece en 1/(1-*p*) listas, el elemento más alto (generalmente un elemento inicial colocado al principio de la lista por saltos) aparece en O(log(1/*p*) *n*) listas.

Para efectos de la complejidad, podemos garantizar que siempre va a ser log(n).

¿Cómo busca?

Borrar

Insertar