Skip Lists

Skip lists are a data structure that can be used in place of balanced trees. Skip lists use probabilistic balancing rather than strictly enforced balancing and as a result the algorithms for insertion and deletion in skip lists are much simpler and significantly faster than equivalent algorithms for balanced trees.

Skip lists are balanced by consulting a random number generator. Although skip lists have bad worst-case performance, no input sequence consistently produces the worst-case performance (much like quicksort when the pivot element is chosen randomly).

Skip List Structure

Each element is represented by a node, the level of which is chosen randoml when the node is inserted without regard for the number of elements in the data structure. A $leveli$ node has i forward pointers, indexed 1 through i. There is no need to store the level of a node in the node. Levels are capped at some appropriate constant $MaxLevel$. The $levelofalist$ is the maximum level currently in the list (or 1 if the list if empty). The *header* of a list has forward pointers at levels one through $MaxLevel$. The forward pointers of the header at levels higher than the current maximum level of the list point to NIL.

Skip List Algorithms

Skip list operations are analogous to that of a binary tree. They include: **search**, **insert**, and **delete**. Note that skip lists are easily extendable to support operations like “find the minimum key” or “find the next key”.

Initialization

An element NIL is allocated and given a key greater than any legal key. All levels of all skip lists are terminated with NIL. A new list has level 1 and all forward pointers of the list’s header point to NIL.

Search Algorithm

Search works by traversing forward pointers that do not overshoot the node containing the element being searched for. When no more progress can be made at the current level of forward pointers, the search moves down to the next level. When we can make no more progress at level 1, we must be in front of the node that contains the desired element (if it is in the list).

At what level should the search be started? William's analysis suggests that ideally we should start at level L where we expect $log_{1/p}n$ where n is the number of elements in the list and p is the fraction of nodes in level i that also have level $i + 1$ pointers. Starting a search at the maximum level in the list does not add more than a small constant to the expected search time.

Insertion and Deletion Algorithms

To insert or delete a node, we simply search and splice. A vector *update* is maintained so that when the search is complete, $update[i]$ contains a pointer to the rightmost node of level i. The new node is of a random level. If the insertion generates a node with a greater level than the previous maximum, both $Maxlevel$ and the appropriate portions of the update vector are updated. After each deletion, we check if we have deleted the maximum element of the list and if so, decrease the maximum level of the list.

Determining $MaxLevel$

Since we can safely cap levels at $L(n)$, we should choose $MaxLevel = L(N)$ where N is an upper bound on the number of elements in a skip list. If $p = 1/2$, using $MaxLevel = 16$ is appropriate for skip lists containing containing up to 2^{16}.

Skip List implementation

My implementation of a skip list is available under my account on githb (**alabid**).

References

Skip Lists: A Probabilistic Alternative to Balanced Trees by William Pugh