Skip lists

Report

By. Soumadip Biswas

Roll: 10IT60R12

SIT, IIT Kharagpur.

Abstract:

[Skip list is a concept, basically a data structure designed intended for a better and efficient access to data than balanced trees stored in a sorted fashion. Basically it is demanded as a better approach to linked lists, these are multiple linked list stored in a parallel fashion. This document is to generate a brief report on the behaviour of the data structure, its evolution and its verities.]

INDEX

Introduction:

Skip lists were invented in 1990 by William Pugh. He details how they work in *Skip lists: a probabilistic alternative to balanced trees* in Communications of the ACM, June 1990, 33(6) 668-676.

To quote the inventor:

> *"Skip lists are a probabilistic data structure that seems likely to supplant balanced trees as the implementation method of choice for many applications. Skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster and use less space."*

A skip-list is a linked list sorted by keys. Each node is assigned a random height, up to some maximum. The number of nodes at any height decreases exponentially with that height. A skip-list node has one successor at each level of height. For example, a node of height 3 has three next pointers, one to the next node of height 1, and another to the next node at height 2, and so on.

Skip List Structure:

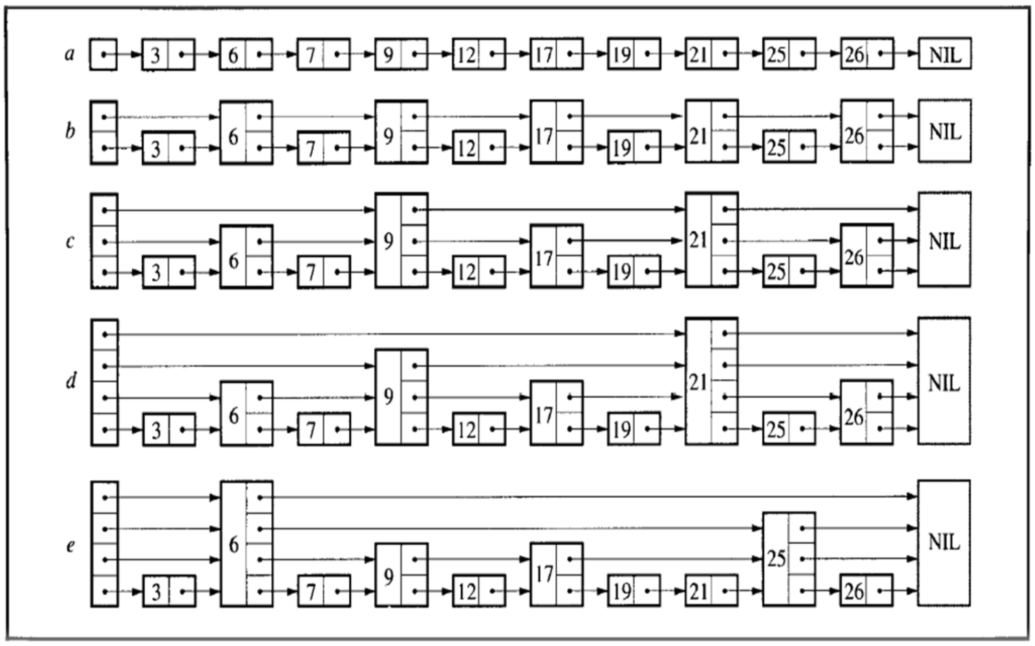
Skip lists are made up of a series of nodes connected one after the other. Each node contains a key/value pair as well as one or more references, or pointers, to nodes further along in the list. The number of references each node contains is determined randomly. This gives skip lists their probabilistic nature, and the number of references a node contains is called its node level.

Each node will have at least one node reference, and the first reference will always point to the next node in the list. In this way, skip lists are very much like linked lists. However, any additional node references can skip one or more intermediate nodes and point to nodes later in the list. This is where skip lists get their name.

Two nodes that are always present in a skip list are the header node and the NIL node. The header node marks the beginning of the list and the NIL node marks the end of the list. The NIL node is unique in that when a skip list is created, it is given a key greater than any legal key that will be inserted into the skip list. This is important to how skip lists algorithms work.

Skip lists have three more important properties: maximum level, current overall level, and probability. Maximum level is the highest level a node in a skip list may have. In other words, maximum level is the maximum number of references a skip list node may have to other nodes. The current overall level is the value of the node level with the highest level in the skip list. Probability is a value used in the algorithm for randomly determining the level for each node.

Here in the next picture there is a skip list has been shown which are nothing but parallel linked lists with additional pointers.

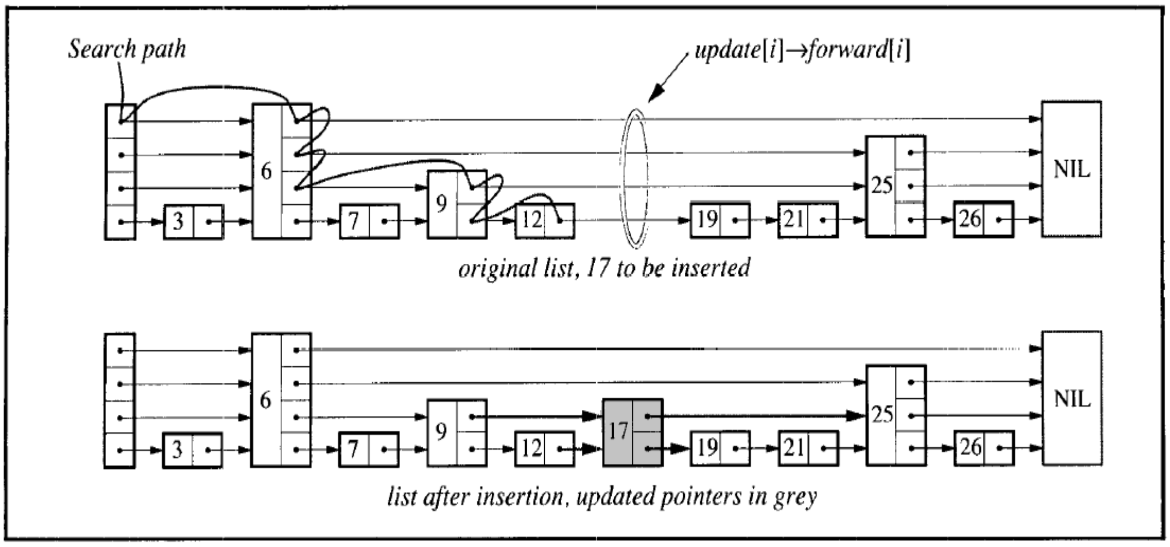


Linked Lists with Additional Pointers

<u>Operations on skip list</u>:

Search:

Searching for a key within a skip list begins with starting at header at the overall list level and moving forward in the list comparing node keys to the search key. If the node key is less than the search key, the search continues moving forward at the same level. If on the other hand, the node key is equal to or greater than the search key, the search drops down one level and continues forward. This process continues until the search key has been found if it is present in the skip list. If it is not, the search will either continue to the end of the list or until the first key with a value greater than the search key is found.


Search path
update[i]→forward[i]
6
9
3
7
12
19
21
25
26
NIL
original list, 17 to be inserted
17
list after insertion, updated pointers in grey

Complexity:

To search for an element with a given key

Find location in top list. Top list has O(1) elements with high probability. Location in this list defines a range of items in next list. Drop down a level and recurse

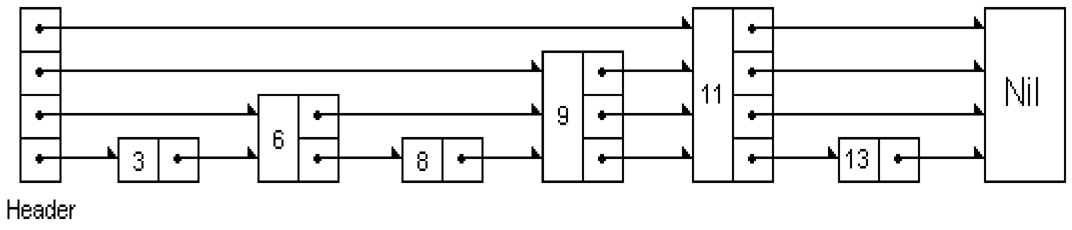
O(1) time per level on average

O(log *n*) levels with high probability

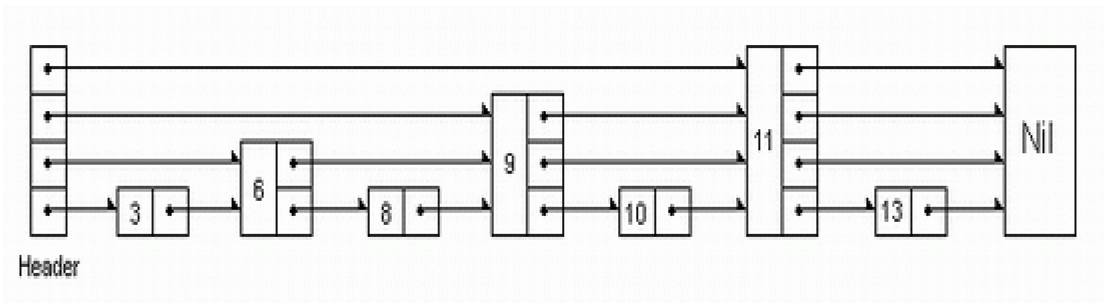
Total time: O(log *n*)

Insert:

Insertion begins with a search for the place in the skip list to insert the new key/value pair. The search algorithm is used with one change: an array of nodes is added to keep track of the places in the skip list where the search dropped down one level. This is done because the pointers in those nodes will need to be rearranged when the new node is inserted into the skip list.



Skip List Before Inserting



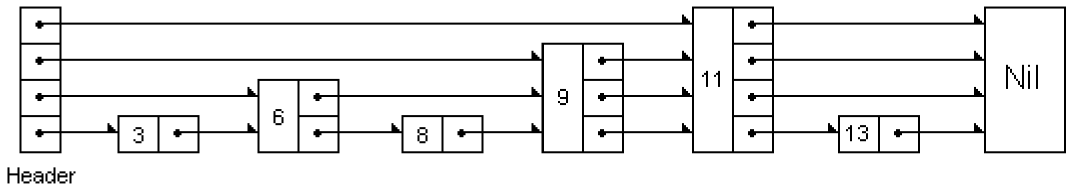
Skip List After Inserting of node 10

Complexity:

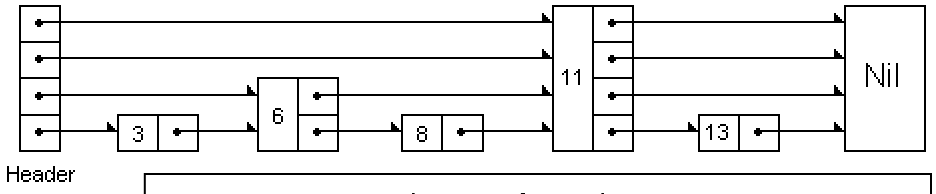
Do a search for that key
Insert element in bottom-level list
With probability *p*, recurse to insert in next level
Expected number of lists = 1+ p + p^2 + … = 1/ (1-p) = O(1)
if *p* is constant
Total time = Search + O(1) = O(log *n*) expected
Skip list delete: O(1)

Deletion:

Deletion uses the same search algorithm as insertion; it keeps track of each place in the list in which the search dropped down one level. If the key to be deleted is found, the node containing the key is removed.



Skip List Before Deletion



Skip List After Deletion

Complexity:

For delete it is simple. Here first we need to find the position that is search and then delete the node which is a constant time operation.

That is the complexity would be: search time + O(1)

Which is nothing but O(log n).

COMPLEXITY ANALYSIS:

Space

The space used by a skip list depends on the random bits used by each invocation of the insertion algorithm. We use the following two basic probabilistic facts:

Fact 1: The probability of getting $\boldsymbol{i}$ consecutive heads when flipping a coin is $1/2^{\boldsymbol{i}}$

Fact 2: If each of $\boldsymbol{n}$ entries is present in a set with probability $\boldsymbol{p}$, the expected size of the set is $\boldsymbol{np}$**.**

Consider a skip list with $\boldsymbol{n}$ entries

By Fact 1, we insert an entry in list $\boldsymbol{S_i}$ with probability $1/2^{\boldsymbol{i}}$

By Fact 2, the expected size of list $\boldsymbol{S_i}$ is $\boldsymbol{n}/2^{\boldsymbol{i}}$

The expected number of nodes used by the skip list is

Thus, the expected space usage of a skip list with $\boldsymbol{n}$ items is $\boldsymbol{O}(\boldsymbol{n})$

Height

The running time of the search an insertion algorithms is affected by the height h of the skip list. We show that with high probability, a skip list with n items has height $O(\log n)$. We use the following additional probabilistic fact:

Fact 3: If each of n events has probability p, the probability that at least one event occurs is at most np **.**

Consider a skip list with n entries

By Fact 1, we insert an entry in list S_i with probability $1/2^i$.

By Fact 3, the probability that list S_i has at least one item is at most $n/2^i$.

Now

By picking i = klog n, we have that the probability that $S_{\text{klog } n}$ has at least one entry is at most.

$$n/2^{\text{klog } n} = n/n^{k} = 1/n^{k-1}$$

Thus a skip list with n entries has height at most 3log n with probability at least $1 - n^{-2}$

Time

The search time in a skip list is proportional to the number of drop-down steps and the number of scan-forward steps. The drop-down steps are bounded by the height of the skip list and thus are $O(\log n)$ with high probability. To analyze the scan-forward steps, we use yet another probabilistic fact:

Fact 4: The expected number of coin tosses required in order to get tails is 2.

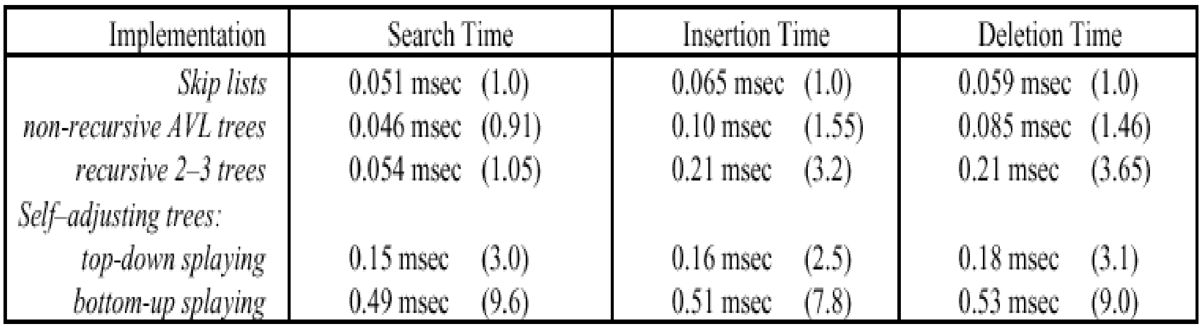
When we scan forward in a list, the destination key does not belong to a higher list. A scan-forward step is associated with a former coin toss that gave tails. So

By Fact 4, in each list the expected number of scan-forward steps is 2.

Thus, the expected number of scan-forward steps is $O(\log n)$. We conclude that a search in a skip list takes $O(\log n)$ expected time. The analysis of insertion and deletion gives similar results

A Survey:

The times in this table reflect the CPU time on a Sun-3/60 to perform an operation in a data structure containing 2 16 elements with integer keys. The values in parenthesis show the results relative to the skip list time. The times for insertion and deletion do not include the time for memory management.



Implementation	Search Time	Insertion Time	Deletion Time
Skip lists	0.051 msec (1.0)	0.065 msec (1.0)	0.059 msec (1.0)
non-recursive AVL trees	0.046 msec (0.91)	0.10 msec (1.55)	0.085 msec (1.46)
recursive 2–3 trees	0.054 msec (1.05)	0.21 msec (3.2)	0.21 msec (3.65)
Self–adjusting trees:			
top-down splaying	0.15 msec (3.0)	0.16 msec (2.5)	0.18 msec (3.1)
bottom-up splaying	0.49 msec (9.6)	0.51 msec (7.8)	0.53 msec (9.0)

Advantages:

- The memory requirements are less than the used for balanced trees
- Insertion and deletion don’t need to be balanced again.
- Insertion, deletion, search and union operations support.

Disadvantages:

- Each node in a Skip list has an array of forward pointers that link the remote nodes.
- The number of pointers ahead in a node its determined by a probabilistic function during insertion; the order its randomized.
- We search for an element by traversing forward pointers that do not overshoot the node containing the element being searched for.

Applications in some fields:

- Q-Map - The QMap class is a value-based template class that provides a dictionary
- Skip DB - It is an open-source database format using ordered key/value pairs

References:

http://www.lesliesanford.com/Programming/SkipList.shtml

A Skip List Cookbook by William Pugh, Institute for Advanced Computer Studies, July 1989.

ww3.algorithmdesign.net/handouts/**SkipLists**.pdf

http://www.csee.umbc.edu/courses/undergraduate/341/fall01/Lectures/SkipLists/skip_lists/skip_lists.html

www.itl.nist.gov/div897/sqg/dads/HTML/**skiplist**.html

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