PES UNIVERSITY

Generic Programming

UE14CS337

GENERIC SKIP LISTS

Project Report

Team Members

Anush S. Kumar	Sushrith Arkal	Tejas S Kasetty
01FB14ECS037	01FB14ECS262	01FB14ECS267

Table of Contents

Table of Figures

Introduction

In computer science, a skip list is a data structure that allows fast search within an ordered sequence of elements. Fast search is made possible by maintaining a linked hierarchy of subsequences, with each successive subsequence skipping over fewer elements than the previous one. Searching starts in the sparsest subsequence until two consecutive elements have been found, one smaller and one larger than or equal to the element searched for. Via the linked hierarchy, these two elements link to elements of the next sparsest subsequence, where searching is continued until finally we are searching in the full sequence. The elements that are skipped over may be chosen probabilistically or deterministically, with the former being more common.

Description

A skip list is built in layers. The bottom layer is an ordinary ordered linked list. Each higher layer acts as an "express lane" for the lists below, where an element in layer i appears in layer i+1 with some fixed probability p (two commonly used values for p are 1/2 or 1/4). On average, each element appears in 1/(1-p) lists, and the tallest element (usually a special head element at the front of the skip list) in all the lists. The skip list contains ***log*** $_{1/p}$ ***n***, lists.

A search for a target element begins at the head element in the top list, and proceeds horizontally until the current element is greater than or equal to the target. If the current element is equal to the target, it has been found. If the current element is greater than the target, or the search reaches the end of the linked list, the procedure is repeated after returning to the previous element and dropping down vertically to the next lower list. The expected number of steps in each linked list is at most 1/p, which can be seen by tracing the search path backwards from the target until reaching an element that appears in the next higher list or reaching the beginning of the current list. Therefore, the total expected cost of a search is ***(log*** $_{1/p}$ ***n)/p***, which is ***O(log n)*** when p is a constant. By choosing different values of p, it is possible to trade search costs against storage costs.

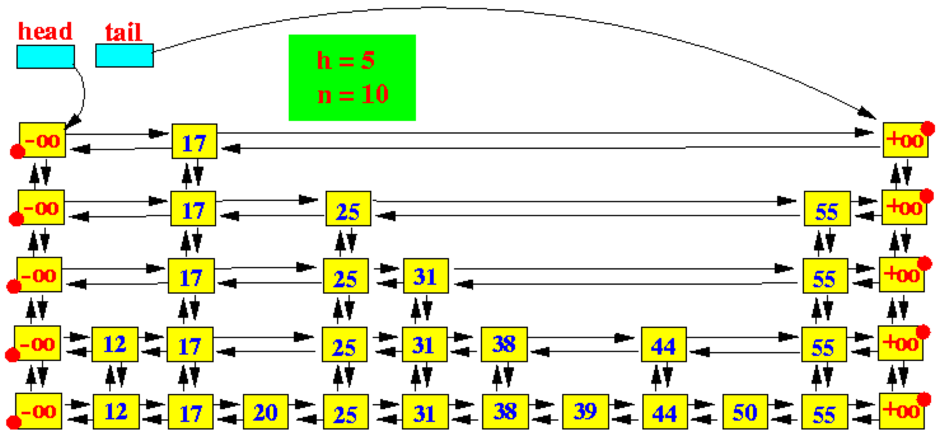

FIGURE I : A SCHEMATIC PICTURE OF THE SKIP LIST DATA STRUCTURE

Implementation details

The elements used for a skip list can contain more than one pointer since they can participate in more than one list.

Insertions and deletions are implemented much like the corresponding linked-list operations, except that "tall" elements must be inserted into or deleted from more than one linked list.

O(n) operations, which force us to visit every node in ascending order (such as printing the entire list), provide the opportunity to perform a behind-the-scenes derandomization of the level structure of the skip-list in an optimal way, bringing the skip list to ***O(log n)*** search time. (Choose the level of the i'th finite node to be 1 plus the number of times we can repeatedly divide i by 2 before it becomes odd. Also, i=0 for the negative infinity header as we have the usual special case of choosing the highest possible level for negative and/or positive infinite nodes.) However this also allows someone to know where all of the higher-than-level 1 nodes are and delete them.

Conclusion

A skip list does not provide the same absolute worst-case performance guarantees as more traditional balanced tree data structures, because it is always possible (though with very low probability) that the coin-flips used to build the skip list will produce a badly balanced structure. However, they work well in practice, and the randomized balancing scheme has been argued to be easier to implement than the deterministic balancing schemes used in balanced binary search trees. Skip lists are also useful in parallel computing, where insertions can be done in different parts of the skip list in parallel without any global rebalancing of the data structure. Such parallelism can be especially advantageous for resource discovery in an ad-hoc wireless network because a randomized skip list can be made robust to the loss of any single node.