Journal of Discrete Algorithms 14 (2012) 13–20

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Skip lift: A probabilistic alternative to red–black trees [☆]

Prosenjit Bose, Karim Douïeb [*], Pat Morin

School of Computer Science, Carleton University, Herzberg Building, 1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6 Canada

A R T I C L E I N F O

Article history:
Available online 8 December 2011

Keywords:
Data structure
Dictionary
Skip list
Finger search

A B S T R A C T

We present the *Skip lift*, a randomized dictionary data structure inspired by the skip list [Pugh'90, Comm. of the ACM]. Similar to the skip list, the skip lift has the finger search property: given a pointer to an arbitrary element f, searching for an element x takes expected $O(\log\delta)$ time where δ is the rank distance between the elements x and f. The skip lift uses nodes of $O(1)$ worst-case size (for a total of $O(n)$ worst-case space usage) and it is one of the few efficient dictionary data structures that performs an $O(1)$ worst-case number of structural changes (pointers/fields modifications) during an update operation. Given a pointer to the element to be removed from the skip lift the deletion operation takes $O(1)$ worst-case time.

1. Introduction

The dictionary problem is fundamental in computer science. It asks for a data structure in the pointer machine model that stores a totally ordered set S of n elements and supports the operations search, insert and delete. A large number of data structures optimally solve this problem in worst-case $O(\log n)$ time per operation. Some of them guarantee an $O(1)$ worst-case number of structural changes (pointers/fields modifications) after an insertion or a deletion operation [12,19,11,13,10,6]. Note that a structural change takes $O(1)$ time.

Typically the update operations that is insert and delete, are performed in two phases: first, search for the position where the update has to take place. Second, perform the actual update and restore the balance of the structure. When the position where the new element has to be inserted or deleted is already known then the first phase of an update could be avoided. In general the first phase is considered to be part of the search operation. A dictionary that guarantees an $O(1)$ worst-case number of structural changes per update does not necessary quickly perform the second phase of the update. Much research effort has been aimed at improving the worst-case time taken by the second phase of the update: Levcopoulos and Overmars [13] presented the first search tree that takes $O(1)$ worst-case time for this second phase of the update. Later Fleischer [10] simplified this result. Brodal et al. [6] additionally guaranteed that such structures can also have the finger search property in worst-case time. These structures however are quite complicated and not really practical.

On the other hand, most randomized dictionaries are simple, practical and achieve the same performance as the result of Brodal et al. [6] in the expected sense. In the worst-case though their performance is far from optimal. Here we develop a simple randomized dictionary, called a skip lift, inspired by the skip list [18], that improves the worst-case performance of the second phase of the update operations. Namely we obtain a structure that has the finger search property in expectation and performs an $O(1)$ worst-case number of structural changes per update. Given a pointer to the element to be removed from the skip lift, the deletion operation takes $O(1)$ worst-case time.

☆ Research partially supported by NSERC and MRI.

* Corresponding author.

E-mail addresses: jit@cg.scs.carleton.ca (P. Bose), karim@cg.scs.carleton.ca (K. Douïeb), morin@cg.scs.carleton.ca (P. Morin).
URL: http://cg.scs.carleton.ca (K. Douïeb).

1570-8667/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.jda.2011.12.017

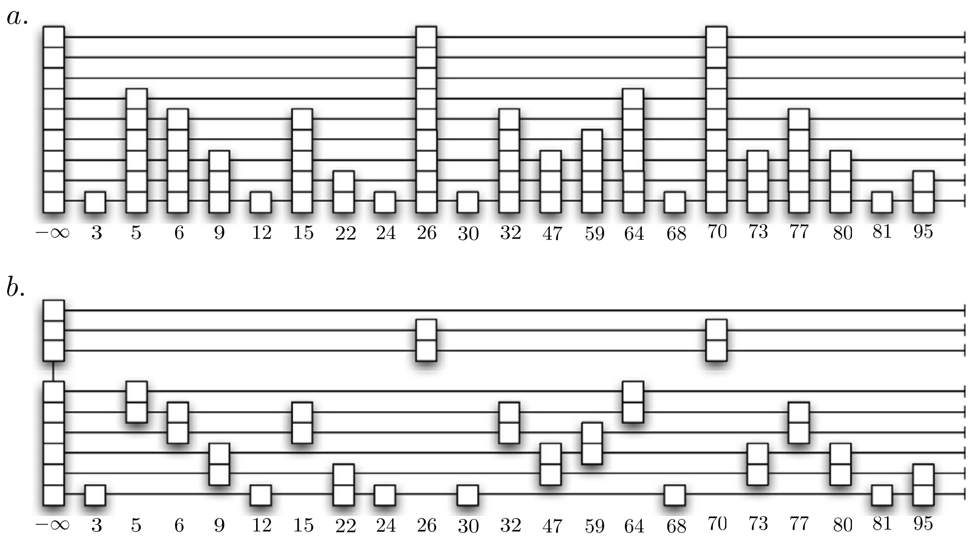

Fig. 1. a. Skip list, b. Skip lift.

In Section 1.1 we describe the original skip list dictionary. In Section 1.2 we mention some work related to the skip list dictionary. In Section 2 we introduce our new skip lift data structure. In Section 3 we show how to enhance the skip lift structure to allow a simple finger search. Finally, in Section 4, we give an overview of some classical randomized dictionary data structures. For each of them we briefly describe its construction and how the dictionary operations are performed. We show that, for these classical randomized dictionaries, in some situations $\Omega(n)$ structural changes are necessary to perform the update operations.

1.1. Skip list

The *skip list* of Pugh [18] was introduced as a probabilistic alternative to balanced trees. It is a dictionary data structure storing a totally ordered set S of n elements that supports insertion, deletion and search operations in $O(\log n)$ expected time. Additionally the expected number of structural changes (pointer modifications) performed on the skip list during an update is $O(1)$. A skip list is built in levels, the bottom level (level 1) is a sorted linked list of all elements in S. The higher levels of the skip list are build iteratively. Each level is a sublist of the previous one where each element of a level is copied to the level above with (independent) probability p. The copies of an element are linked between adjacent levels (see Fig. 1.a).

The *height* $h(s)$ of an element s is defined as the highest level where s appears. The height $H(\mathcal{L})$ of a skip list $\mathcal{L}$ is defined as $\max_{s\in\mathcal{L}} h(s)$ and the *depth* $d(s)$ of s is $H(\mathcal{L}) - h(s)$. The expected height of a skip list is by definition $O(\log_{1/p} n)$. Adjacent elements on the same level are connected by their left and right pointers. The copies of the same element from two adjacent levels are connected by their up and down pointers.

1.1.1. Search

To search for a given element x in a skip list we start from the highest level of the sentinel element which has a key value $-\infty$. We follow the right pointers on a same level until we are about to overshoot the element x that is until the element on the right has a key value strictly greater than x. Then we go down one level and we iterate the process until x is found or until we have reached the lowest level (in this case we know that x is not in S and we have found its predecessor).

1.1.2. Updates

To insert an element x in a skip list we first determine its height in the structure. Then we start a search for x in the list to find the position where x has to be inserted. During the search we update the pointers of the copies of the elements that are adjacent to a newly created copy of x.

The deletion of an element x from a skip list is straightforward given the insertion process. We first search for x and we delete one by one all its copies while updating the pointers of the copies of elements that are adjacent to a copy of x.

1.2. Related work

Precise analysis of the expected search cost in a skip list has been extensively studied, we refer to the thesis of Papadakis for more information [17]. Several variants of the skip list have been considered: Munro et al. [16] developed a deterministic version of the skip list, based on B-trees [3], that performs each dictionary operation in worst-case $O(\lg n)$ time. Under the