Some observations on Skip Lists

Sandeep Sen
AT&T Bell Laboratories
Murray Hill, NJ 07974

Abstract

The Skip-list data-structure was proposed by Pugh [1] as an alternate to balanced binary search trees. This remarkably simple data-structure uses randomization and Pugh proved that the expected performance is comparable to the balanced trees. In this note we prove that the performance can be further guaranteed in the following sense - the probability of the search time or space complexity exceeding their expected values approaches 0 rapidly as the number of keys increases.

KEYWORDS: *analysis of algorithms, data-structures*

1 Introduction

Skip-list is a data structure introduced by Pugh [1] as an alternative to balanced binary search trees for handling dictionary operations on ordered lists. In Pugh [2], further applications are described. The underlying idea is to substitute complex book-keeping information used for maintaining balance conditions for binary trees by random sampling techniques. It has been shown by Pugh [1] that, given access to random bits, the *expected* search time in a skip-list of n elements is $O(\log n)$ [1] which compares very favourably with balanced binary trees. Moreover, the procedures for insertion and deletion are very simple which makes this data-structure a very attractive alternative to the balanced binary trees.

Since the search time is a stochastic variable (because of the use of randomization), it is of considerable interest to determine the bounds on the tails of its distribution. Often, it is crucial to know the behavior for any individual access rather than a chain of operations since it is more closely related to the real-time response. Pugh [1] derived the actual probability distribution functions for access time and verified experimentally that the probability of deviation from the expected behavior by any constant factor is vanishingly small. In this note, we present a simple proof for his observation and as a consequence of our main theorem, we show that the probability of deviating significantly from the mean behavior is vanishingly small for *any* individual access.

2 Review of Skip-lists

We briefly review the basic data-structure proposed by Pugh. This data-structure is maintained as a hierarchy of sorted linked-lists. The bottom-most level is the entire set of keys S. We denote the

[1] Note that all logarithms are to base 2 unless otherwise mentioned.

linked list at level i from the bottom as L_i and let $|L_i| = N_i$. By definition $L_0 = S$ and $|L_0| = n$. For all $0 \leq i$, $L_i \subset L_{i-1}$ and the topmost level, say level k has constant number of elements. Moreover, correspondences are maintained between common elements of lists L_i and L_{i-1}. For a key with value E, for each level i, we denote by T_i a tuple (l_i, r_i) such that $l_i \leq E \leq r_i$ and $l_i, r_i \in L_i$. We call this tuple *straddling pair* (of E) in level i.

The search begins from the topmost level L_k where T_k can be determined in constant time. If $l_k = E$ or $r_k = E$ then the search is successful else we recursively search among the elements $[l_k, r_k] \bigcap L_0$. Here $[l_k, r_k]$ denotes the closed interval bound by l_k and r_k. This is done by searching the elements of L_{k-1} which are bounded by l_k and r_k. Since both $l_k, r_k \in L_{k-1}$, the *descendence* from level k to $k-1$ is easily achieved in $O(1)$ time. In general, at any level i we determine the tuple T_i by walking through a portion of the list L_i. If l_i or r_i equals E then we are done else we repeat this procedure by *descending* to level $i-1$.

In other words, we refine the search progressively until we find an element in S equal to E or we terminate when we have determined (l_0, r_0). This procedure can also be viewed as searching in a tree that has variable degree (not necessarily two as in binary tree).

Of course, to be able to analyze this algorithm, one has to specify how the lists L_i are constructed and how they are dynamically maintained under deletions and additions. Very roughly, the idea is to have elements in i-th level point to approximately 2^i nodes ahead (in S) so that the number of levels is approximately $O(\log n)$. The time spent at each level i depends on $[l_{i+1}, r_{i+1}] \bigcap L_i$ and hence the objective is to keep this small. To achieve these conditions on-line, Pugh [1] uses the following elegant method. The nodes from the bottom-most layer (level 0) are chosen with probability p (for the purpose of our discussion we shall assume $p = 0.5$) to be in the first level. Subsequently at any level i, the nodes of level i are chosen to be in level $i+1$ independently with probability p and at any level we maintain a simple linked list where the elements are in sorted order. If $p = 0.5$, then it is not difficult to verify that for a list of size n, the *expected* number of elements in level i is approximately $n/2^i$ and are spaced about 2^i elements apart. The expected number of levels is clearly $O(\log n)$, (when we have just a trivial length list) and the expected space requirement is $O(n)$.

To insert an element, we first locate its position using the search strategy described previously. Note that a byproduct of the search algorithm are all the T_i's. At level 0, we choose it with probability p to be in level L_1. If it is selected, we insert it in the proper position (which can be trivially done from the knowledge of T_1), update the pointers and repeat this process from the present level. Deletion is very similar and it can be readily verified that deletion and insertion have the same asymptotic run time as the search operation. So we shall focus on this operation.

3 A tight analysis

To analyze the run-time of the search procedure, we look at it backwards, i.e., retrace the path from level 0. The search time is clearly the length of the path (number of links) traversed over all the levels. So one can count the number of links one traverses before climbing up a level. In other words the expected search time can be expressed in the following recurrence (from [1])

$$C(k) = (1-p)(1 + C(k)) + p(1 + C(k-1))$$

where C(k) is the expected cost for climbing k levels. From the boundary condition $C(0) = 0$, one readily obtains $C(k) = k/p$. For $k = O(\log n)$, this is $O(\log n)$. The recurrence captures the crux of the method in the following manner. At any node of a given level, we climb up if this node has been chosen to be in the next level or else we add one to the cost of the present level. The probability of this event (climbing up a level) is p which we consider to be a success event. Now the entire search procedure can be viewed in the following alternate manner. We are tossing a coin which turns up heads with probability p - how many times should we toss to come up with $O(\log n)$ heads ? Each head corresponds to the event of climbing up one level in the data structure and the total number of tosses is the cost of the search algorithm. We are done when we have climbed up $O(\log n)$ levels (there is some technicality about the number of levels being $O(\log n)$ but that will be addressed later). The number of heads obtained by tossing a coin N times is given by a Binomial random variable X with parameters N and p. The following inequalities due to Angluin and Valiant [4] known as Chernoff bounds [3] are often used to obtain tail-estimates of a binomially distributed random variable X.

$$Prob(X \leq (1-\epsilon)pN) \leq exp(-\epsilon^2 Np/3) \tag{1}$$

$$Prob(X \geq (1+\epsilon)pN) \leq exp(-\epsilon^2 Np/2) \tag{2}$$

Here $0 < \epsilon < 1$. For $N = 15 \log n$ and $p = 0.5$, $Prob(X \leq 1.5 \log n) \leq 1/n^2$ (using $\epsilon = 9/10$ in equation 1). Using appropriate constants, we can get rapidly decreasing probabilities of the form $Prob[X \leq c \log n] \leq 1/n^\alpha$ for $c, \alpha > 0$ and α increases with c. These constants can be fine tuned although we shall not bother with such an exercise here.

We thus state the following lemma.

Lemma 1 *The probability that access time for a fixed element in a skip-list data structure of length n exceeds $c \log n$ steps is less than $O(1/n^2)$ for an appropriate constant $c > 1$.*

Proof We compute the probability of obtaining fewer than k (the number of levels in the data-structure) heads when we toss a fair coin ($p = 1/2$) $c \log n$ times for some fixed constant $c > 1$. That is, we compute the probability that our search procedure exceeds $c \log n$ steps. Recall that each head is equivalent to climbing up one level and we are done when we have climbed k levels. To bound the number of levels, it is easy to see that the probability that any element of S appears in level i is at most $1/2^i$, i.e. it has turned up i consecutive heads. So the probability that any fixed element appears in level $3 \log n$ is at most $1/n^3$. The probability that $k > 3 \log n$ is the probability that at least one element of S appears in $L_{3 \log n}$. This is clearly at most n times the probability that any fixed element survives and hence probability of k exceeding $3 \log n$ is less than $1/n^2$.

Given that $k \leq 3 \log n$ we choose a value of c, say c_0 (to be plugged into equation 1 of Chernoff bounds) such that the probability of obtaining fewer than $3 \log n$ heads in $c_0 \log n$ tosses is less than $1/n^2$. The search algorithm for a fixed key exceeds $c_0 \log n$ steps if one of the above events fail; either the number of levels exceeds $3 \log n$ or we get fewer than $3 \log n$ heads from $c_0 \log n$ tosses. This is clearly the summation of the failure probabilities of the individual events which is $O(1/n^2)$. □.

Theorem 1 *The probability that the access time for any arbitrary element in skip-list exceeds $O(\log n)$ is less than $1/n^\alpha$ for any fixed $\alpha > 0$.*

Proof: A list of n elements induces $n+1$ intervals. From the previous lemma, the probability P that the search time for a fixed element exceeding $c \log n$ is less than $1/n^2$. Note that all elements in a fixed interval $[l_0, r_0]$ follow the same path in the data-structure. It follows that for any interval the probability of the access time exceeding $O(\log n)$ is n times P. As mentioned before, the constants can be chosen appropriately to achieve this. □

It is possible to obtain even tighter bounds on the space requirement for a skip list of n elements. From Pugh [1] it is known that the expected space is $O(n)$. Moreover it is clear that it does not exceed $O(n \log n)$ with probability $1 - 1/n^2$ (no element survives more than $O(\log n)$ levels with this probability from the previous lemma).

From Chernoff bounds (equation 2), it follows that the probability that $N_i \geq 3/4 N_{i-1}$ is less than $e^{-\beta\sqrt{n}}$ for $N_{i-1} = O(\sqrt{n})$ for some fixed constant $\beta > 0$. Recall that $N_i = |L_i|$. We say that a level is *good* if $N_i \leq 3/4 N_{i-1}$ and *bad* otherwise. Let $\gamma = \frac{1}{\log(4/3)}$. So the probability that $N_j > \sqrt{n}$ for some $j > \gamma \log n$ is less than $e^{-\Omega(\sqrt{n})}$ by summing up the failure probabilities of the $O(\log n)$ *bad* events. The probability that any element of level j, survives another $O(\sqrt{n})$ levels is less than $1/2^{\sqrt{n}}$. i.e., the probability of obtaining $\sqrt{n}$ consecutive heads. Hence the probability that $k > j + \sqrt{n}$ is less than $1/2^{\Omega(\sqrt{n})}$; the reader can work out the exact constants hidden behind the Ω notation.

The space bound is then

$$\sum_{i=0}^{k} N_i \leq \sum_{i=0}^{j} N_i + \sum_{i=j+1}^{k} N_i$$

From our previous discussion the probability that the first term exceeds $\sum_{i=0} (3/4)^i n$ is less than $e^{-\Omega(\sqrt{n})}$ and the probability that the second term exceeds $\sqrt{n} \cdot \sqrt{n}$ $(= n)$ is less than $1/2^{O(\sqrt{n})}$. For the sum to exceed $O(n)$ one of the terms would have to exceed $4n$. So we state the following result:

Theorem 2 *There exists a constant α such that the probability of the space exceeding αn, is less than $1/2^{\Omega(\sqrt{n})}$.*

In conclusion, we note that there is considerable experimental evidence that skip-lists could be a strong alternative to balanced binary trees. The result in this note establishes that it is competitive in a certain worst-case sense also. It will be interesting to analyze the performance of the Randomized Search Trees of Aragon and Seidel [5] from the viewpoint of bounding the variance.

References

[1] W. Pugh, "Skip Lists: A Probabilistic Alternative to Balanced Trees," Communications of the ACM, Volume 33 Number 6, June 1990, pp. 668-676.

[2] W. Pugh, "A Skip List Cookbook," Technical Report, UMIACS-TR- 89-72.1, University of Maryland.

[3] Chernoff, H., "A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations," Annals of Math. Statistics 23, 1952, pp. 493-507.

[4] D. Angluin and L. Valiant, "Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings," *Journal of Computer and System Science*, 18, 1979.

[5] C. Aragon and R. Seidel, "Randomized Search Trees," *Proc. of the 30th Annual Symposium on Foundations of Computer Science*, 1989, pp. 540-545.