Skip Lists

March 5, 2018

Contents

1 Introduction

Going back to the past, we always want to be able to support an abstract data type called dictionary. This can be seen as the following problem

Input: A set of n records, each identified by one or more key fields

Problem: Build and maintain a data structure to efficiently locate, insert, or delete the record associated with any query key.

Basically based in this input and problem, we are looking for data structures that allow us to have the best performance in a dictionary. However, there is a precautionary note from "The Algorithm Design Manual" by Steven Skiena:

- *"In practice, it is more important to avoid using a bad data structure than to identify the single best option available."*

Some people will say that this is not essential, but when confronted with million of records as in many of the massive databases of today, it is essential to have the best data structure you can have.

2 What Kind of Data Structures for a Dictionary?

There are several types of data structures that can be used in a dictionary. For example, we can use

1. Ordered and Unordered Arrays.
2. Binary Search Trees.
3. Hash Tables.

As we can noticed, there are some drawbacks to each of data structures:

1. Arrays require a dynamic table support. They require long enough free memory segments. In addition, we have complexities:
   (a) Insertion: Unordered $O(1)$, Ordered $O(n)$.
   (b) Search: Unordered $O(n)$, Ordered $O(\log n)$.
2. Binary Search Trees require to have a well balanced tree such that the height is bounded by $O(\log n)$, where n is the number of elements stored in the data structure.
3. Hash Tables require a dynamic table implementation once the load α crosses a certain threshold, in addition to a new hash function.

Thus, we require a new data structure for supporting dictionary.

3 Skip Lists

There is new data structure called skip list which is a good fit for the properties we are looking for:

- Insertion complexity $O(\log n)$.
- Search complexity $O(\log n)$.
- No fixed-size structure as with binary search trees.

3.1 Starting from Scratch

Imagine that you only require to have searches. A first approximation to it is the use of a link list for it ($\Theta(n)$ search complexity). Then, using this How do we speed up searches?

- Use two link list, one a subsequence of the other.

Imagine the two lists as a road system (Fig. 1).

1. The Bottom is the normal road system.
2. The Top is the high way system.

We use the top one to as a traffic diversion for "express" stops.

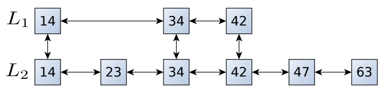

Figure 1: The two list idea

- **To Search first search in the top one (L_1) as far as possible, then go down and search in the bottom one (L_2).**

The Cost for a search is :

$$len\,(L_1) + \frac{len\,(L_2)}{len\,(L_1)} = len\,(L_1) + \frac{n}{len\,(L_1)}$$

This is minimized when:

$$len(LL1) = \frac{n}{len(LL1)} \Rightarrow len(LL1) = \sqrt{n}$$

Thus, search cost is equal to $2\sqrt{n}$. You can do more, you actually can put a high way to the high way. Then, we get a search cost of $3\sqrt[3]{n}$. In the general case, after all we can pile high ways on top of each other, $k\sqrt[k]{n}$. For $k = \lg n$, we have $\lg n \sqrt[\lg n]{n} = \lg n \cdot n^{\frac{1}{\lg n}} = 2\lg n$ because

- $y = \frac{1}{\lg n} \Rightarrow n = 2^{1/y} \Rightarrow \left(2^{1/y}\right)^y = 2$

Thus, we have the following definition.

Definition

A skip list for a set S of distinct $(key, element)$ items is a series of lists $S_0, S_1, ..., S_h$ such that:

- Each list S_i contains keys $-\infty$ and ∞.
- List S_0 contains all the keys in nondecreasing order.
- Each list is a subsequence of the previous one, $S_0 \supseteq S_1 \supseteq ... \supseteq S_h$.
- List S_h contains only the keys $-\infty$ and ∞.

4 The Basic Operations for the Skip Lists

Here, we add the probabilistic part of the skip list to minimize bad searches by using the insertions on it.

4.1 Insertions

In order to insert an element (x, o), we do the following

- Flip a fair coin. If head, add element to the next level up and repeat until tail. Then, add the last empty level.

Thus, in average we have

- We have probability 1/2 the element go up one level.
- We have probability 1/4 the element go up two levels.
- We have probability 1/8 the element go up three levels.
- etc...

4.2 Deletions

To remove an entry with key x from a skip list, we proceed as follows:

- We search for x in the skip list and find the positions $p_0, p_1, \ldots, p_i$ of the items with key x, where position p_j is in list S_j.
- We remove positions $p_0, p_1, \ldots, p_i$ from the lists $S_0, S_1, ..., S_i$.
- We remove all but one list containing only the two special keys.

5 Space Used by the Skip List

First, we prove how much space is used by the skip list.

Theorem The expected space usage of a skip list with n items is $O(n)$.

Proof: We use the following two basic probabilistic facts:

1. The probability of getting i consecutive heads when flipping a coin is $\frac{1}{2^i}$.
2. If each of n entries is present in a set with probability p, the expected size of the set is np. Remember $X = X_1 + X_2 + ... + X_n$ where X_i is an indicator function for event A_i = the i element is present in the set. Thus:

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} Pr\{A_i\} = \sum_{i=1}^{n} p = np$$

Consider a skip list with n entries

- By Fact 1, we insert an entry in list S_i with probability $\frac{1}{2^i}$.
- By Fact 2, the expected size of list S_i is $\frac{n}{2^i}$.
- The expected number of nodes used by the skip list is:

$$\sum_{i=0}^{h} \frac{n}{2^i} = n \sum_{i=0}^{h} \frac{1}{2^i} < 2n$$

Thus, the expected space usage of a skip list with n items is $O(n)$.

6 Height of the Skip List

The running time of the search an insertion algorithms is affected by the height h of the skip list: For this:

- We only need to show with high probability, a skip list with n items has height $O(\log n)$
- Fact 3: We can view the level, $l(x) = \max\{j | x \in S_j\}$, of the elements in the skip list S as the random variables with geometric distribution $X_1, X_2, X_3, ..., X_n$ going from the top to the bottom of the Skip List.Then,

$$Pr[X_k > i] \leq (1-p)^i \Rightarrow Pr\left\{\max_k X_k > i\right\} \leq n(1-p)^i.$$

Given that $Pr\{\max_k X_k > i\} = Pr\{(X_1 > i) \vee (X_2 > i) \vee .... \vee (X_n > i)\} = \sum_{k=1}^{n} Pr\{(X_k > i)\}$

  - Where the $\max_k X_k$ represent the list with the one entry apart from the special keys.

- REMEMBER!!! We are talking about a fair coin, thus $p = \frac{1}{2}$.

Theorem A skip list with n entries has height at most $3 \log n$ with probability at least $1 - \frac{1}{n^2}$

Proof:

Consider a skip list with n entires. By Fact 3, the probability that list S_i has at least one item is at most $\frac{n}{2^i}$. By picking $i = 3 \log n$, we have that the probability that $S_{3 \log n}$ has at least one entry is at most:

$$\frac{n}{2^{3 \log n}} = \frac{n}{n^3} = \frac{1}{n^2}.$$

Therefore, the probability that the height $h = 3 \log n$ of the skip list is $1 - \frac{1}{n^2}$.

7 Search and Update Times

The search time in a skip list is proportional to the number of drop-down steps, plus 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.

Theorem A search in a skip list takes $O(\log n)$ expected time.

Proof:

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. By Fact 4, in each list the expected number of scan-forward steps is 2. Thus, the expected number of scan-forward steps is $O(\log n)$.

The analysis of insertion and deletion gives similar results.