Implementation of the skip list data structure with it's update operations

Maha shakir Ibrahim
Iraqi university - college of education for Women

Abstract
A skip list data structure is really just a simulation of a binary search tree. Skip lists algorithm are simpler, faster and use less space. this data structure conceptually uses parallel sorted linked lists. Searching in a skip list is more difficult than searching in a regular sorted linked list. Because a skip list is a two dimensional data structure, it is implemented using a two dimensional network of nodes with four pointers. the implementation of the search, insert and delete operation taking a time of upto $\mathcal{O}(\log n)$. The skip list could be modified to implement the order statistic operations of RANK and SEARCH BY RANK while maintaining the same expected time.
Keywords: skip list , parallel linked list , randomized algorithm , rank.

تنفيذ هياكل البيانات لقائمة التخطي مع عمليات التحديث عليها

مها شاكر إبراهيم
الجامعة العراقية – كلية التربية للبنات

الخلاصة
ان قوائم التخطي هي من نوع هياكل البيانات التي تبدو مشابهة في طريقة تنفيذها للأشجار المتوازنة . خوارزمية عمل قوائم التخطي تكون أبسط و أسرع و تستخدم مساحة أقل. في قوائم التخطي تستخدم قوائم مترابطة و مرتبة و بشكل متواز. البحث في قوائم التخطي يكون أصعب من البحث في القوائم المترابطة المرتبة الاعتيادية. و لأن قوائم التخطي هي ان من نوع هياكل البيانات ثنائية البعدين, فهي تنفذ باستخدام شبكة ثنائية البعدين من العقد المكونة من أربعة مؤشرات. تنفيذ خوارزميات البحث , الادخال و الحذف تأخذ لوغارتيم(ن) من الوقت. كذلك من الممكن تغيير قوائم التخطي لتنفيذ عمليات تحديد المرتبة و البحث بالمرتبة مع المحافظة على نفس الوقت المستغرق.

1.Introduction

Skip lists were first described by William Pugh. He details how they work in Skip lists that is a probabilistic alternative to balanced trees, in Communications of the ACM, June 1990, and he conclude that Skip lists are a probabilistic data structure that seem 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.

The skip lists are implemented for a sorted list of items using a hierarchy of linked lists that connect increasingly sparse subsequences of the items, it is used in distributed applications (where the nodes represent physical computers, and pointers represent network connections) and for implementing highly scalable concurrent priority queues with less lock contention, or even without locking, and for implementing concurrent dictionaries.[1][2][3]

2. Skip list 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, as shown in figure(1) , for a set S of distinct elements, the skip list is a series of lists:$s_0, S_1, S_2, \ldots, S_h$ such that:

-Each list S_i contains the special keys $+\infty$ and $-\infty$.

-List S_0 contains the keys of S in increasing order.

-Each list is a subsequence of the previouse one. i.e,

$S_0 \supseteq S_1 \supseteq \ldots \supseteq S_h$

-List S_h contain only the two special keys. [1]

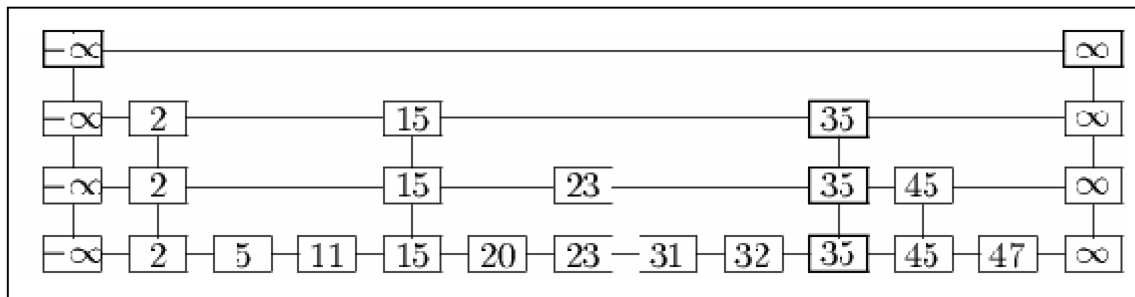
Example: for the following values:

$S_0 = \{2,5,11,15,20,23,31,32,35,45,47\}$

$S_1 = \{2,15,23,35,45\}$

$S_2 = \{2,15,35\}$

The corresponding skip lists will be just like in the following figure:



Figure(1) An example of a skip lists

3. Search operation in a skip list

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 ,as shown in the example of figure(2). Therefore, the total *expected* cost of a search is $\mathcal{O}(\log n)$.[1][3][4]

The search algorithm was implemented using the following operations:

Next(p): return the position following p on the same level.

Prev(p): return the position preceding p on the same level.

Below(p): return the position below p.

Above(p): return the position above p.

The SEARCH(x) algorithm will be as the following:

*Start at the first position of the top list.

*At the current position p, compare x with y (y $\leftarrow$ key(next(p)).

- If x=y the key found.
- If x>y : we scan forward.
  P$\leftarrow$next(p).
- If X<y : we drop down.

P ⟵ below(p).

*If we try to drop down past the bottom list we return key not found.

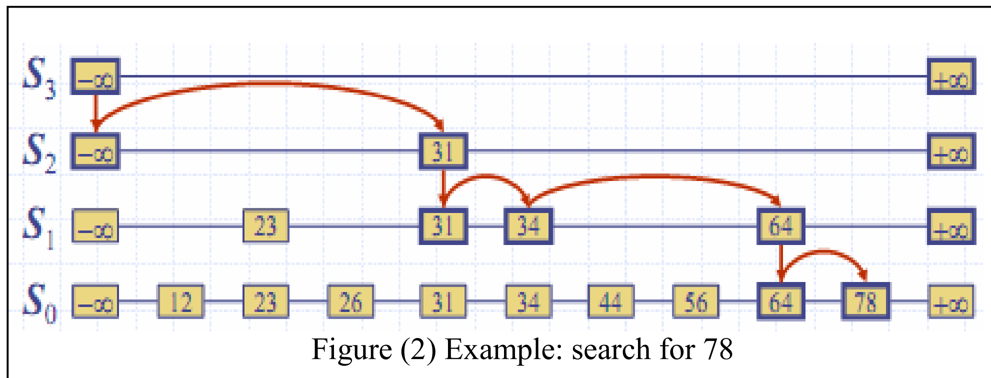


Figure (2) Example: search for 78

4. Insertion operation in the skip list

To insert an element in the skip list, a randomized algorithm was used ,such that the skip list makes random choices in arranging the entries in such a way that search and update times are o(log n) on average, where n is the number of entries . The main advantage of using randomization in data structures and algorithm design is that the structures and methods that result are usually simple and efficient. So the skip list was a simple randomized data structure.[3]

4.1. A randomized algorithm

A randomized algorithm performs coin tosses (i.e, uses random bits) to control its execution.

It contains statements of the type:

b ⟵ random( )
if b=0
do A…
else {b=1}
do B…

its running time depends on the outcomes of the coin tosses.[4]

4.2.Details of INSERTION algorithm

To insert a key k begin by performing a search(k) operation. This gives us the position p of the bottom level entry with the largest key less than or equal to k, the key k was inserted immediately after position p. after inserting the new entry at the bottom level , a randomized algorithm was performed by flipping a coin until it comes up tails, when the flip comes up head promot k to the next level up and flip again, until it comes up tail then stop. If the number of flips equal the height of the lists(h) then a new list with special keys must be added. An example about the insertion operation was shown in figure (3). the total expected running time of the insertion algorithm on a skip list with n entries is $\mathcal{O}(\log n)$[4][5]

The INSERT(k) algorithm will be as the following:

flipcoin ⟵ random(2)

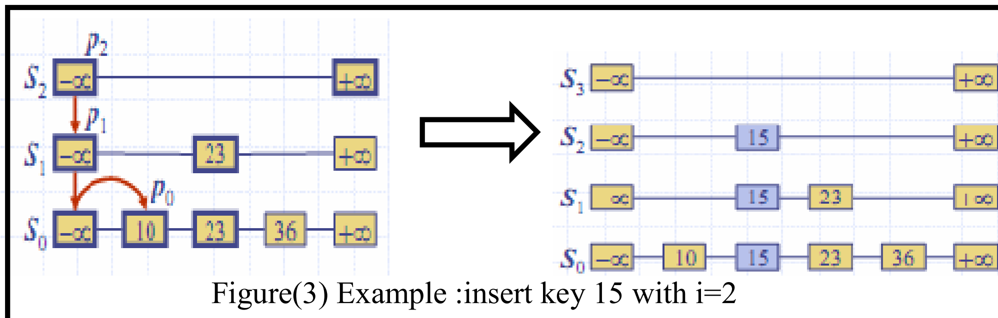
While(flipcoin=1)

i⟵i+1

If i>= h , add new lists $s_{h+1},\ldots,s_{i+1}$,each containing only the two special keys.

Search(k) in the skip list and find the positions $p_0,p_1,\ldots p_i$ in each list $s_0,s_1,\ldots s_i$

For j⟵0,…,i insert k into list s_j after position p_j.



Figure(3) Example :insert key 15 with i=2

5. Removal operation in the skip list

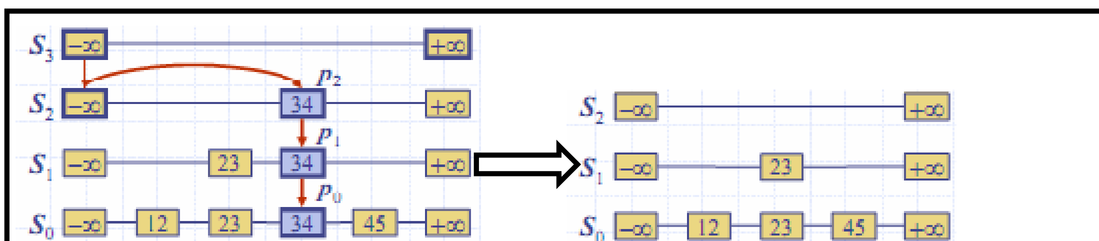
Like the search and insertion algorithms, the removal algorithm for a skip list is quite simple. In fact, it is even easier than the insertion algorithm . that is, to perform a remove(k) operation, we begin by executing search(k) operation . if the key k not found we return null**.** otherwise remove the position with the value k and all the positions above it, which are easily accessed by using above operations. The removal operation was shown in figure(4). The expected running time of the removal algorithm is $O(\log n)$, **[5][6].**

The algorithm for remove(k) will be as follow:

*Search(k) and find positions $P_0,P_1,\ldots,P_i$
Where P_j is in list S_j

*Remove positions $P_0,P_1,\ldots,P_i$ from the lists $S_0,S_1,\ldots,S_i$

*Remove all but on list containing only the two special keys.



Figure(4) Example : remove key 34

6. Space usage

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

Fact1: the probability of getting I consecutive heads when flipping a coin is $1/2^i$

Fact2: if each of n entries is present in a set with probability p,the expected size of the set is np.[1][4]

And thus for a skip list with n entries:

By fact1: an entry was inserted in list s_i with probability $1/2^i$

By fact2: the expected number of nodes used by the skip list is

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

thus the expected space usage of a skip list with n items is o(n).

7. Search and update time

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 $\mathcal{O}(\log n)$ with heigh probability . To analyze the scan-forward steps, we use yet another probabilistic fact:

The expected number of coin tosses required in order to get tail is 2.

When 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 the last fact in each list the expected number of scan-forward steps is $\mathcal{O}(\log n)$ expected time . the analysis of insertion and deletion gives similar result.[1][2]

8. Order Statistics Operations in Skip Lists

Here we implement the order statistics operations RANK and SEARCH-BY-RANK in a skip list. that is for a node x in a skip list L, RANK(x,L) gives the rank of x among the elements in the list. SEARCH-BY-RANK(K,L) is the inverse of RANK . it returns the k-th element in the skip list L. if no such node exist it return nill. The top level of the skip list denoted as level 1. And the skip list L must has a variable L.depth that stores the number of levels in L.[7]

8.1.The Modification of the Search, Insert and Delete Operation

The skip list implementation needs to be modified so that the RANK and SEARCH-BY-RANK can be implemented with $\mathcal{O}(\log n)$ time complexity.

This could be done by adding an attribute span[x,i] for each node x at each level i. which indicates the number of elements spanned by the pointer at x to the next element at level i . specifically, if the rank of x is r_x and the element y following x at level i has rank r_y , the span[x,i] is r_y-r_x. if x is the end of the list, then span[x,i]=0.

Search operation: the search operation does not need to be changed.

Insert operation: the insert operation was made as before. In addition, for each element p_i=prev[x,i], calculate its rank r_i, which can be obtained by summing up the total span traversed up to p_i . for each level i, we update the values of span[x,i] and span[p_i,i] by the following formula in order:

$$\text{span}[x,i]=\text{span}[p_i,i]-(r_1-r_i) \quad \text{-------- (2)}$$

$$\text{span}[p_i,i]=r_1-r_i+1 \quad \text{-------------------- (3)}$$

The expected running time is the same as the original insert operation, which is $\mathcal{O}(\log n)$.**[6][7]**

Delete operation: the elements was deleted as befor. For each element p_i =prev[x,i] of x at each level i, we update the values span[p_i ,i], by the following formula:

$$span[p_i ,i]= span[p_i ,i]+ span[x ,i]+1 \text{-----------} (4)$$

The expected running time is the same as the original delete operation, which is $\mathcal{O}(\log n)$.

8.2 The Algorithm of RANK(X,L)

In the procedure RANK , the loop invariant is r=rank(y) , which hold just before entering the loop. Inside the loop, wherever r gets increased , the pointer y jumps ahead to the element with exactly the same distance. Therefore the invariant is preserved . if x is in the skip list , the procedure returns r when y=x, so it return the rank of x. otherwise, the procedure cannot find y and returns nil.the running time is the same as SEARCH, which is $\mathcal{O}(\log n)$ expected time.

The RANK(X,L) Algorithm will be as follow:

```
y=L
r=0
level=1
while level<=L.depth
    while key[next[y,level]] <= key[x]
        r=r+span[y,level]
        y=next[y,level]
   if y=x return r
   else level=level+1
return nill
```

8.3 The Algorithm for SEARCH-BY-RANK(k,L)

The procedure SEARCH-BY-RANK search for the element and keeps track of total span sum during the process. The invariant r+rank(y)= k and r>=0. The invariant is preserved in the loop because whenever r decreased, the pointer y jumps forward by the same amoun . the program exits the loop when r=0, so it follows that rank(y)=k. therefor the program returns y as the queried element. The analysis of the running time is the same as SEARCH, which is $\mathcal{O}(\log n)$ expected time. [7]

The SEARCH-BY-RANK(K,L) Algorithm will be as follow:

```
y=L
r=k
level=L
while r>0
    if level>L.depth return nil
    if span[y,level]<r
        r=r-span[y,level]
        y=next[y,level]
    else level=level+1
return y
```

9.Conclusions

The skip list implementation was so simple and efficient , it is just like any balanced tree but much simpler and faster. The algorithms for the skip list needs $\mathcal{O}(\log n)$ time. The implementation of the SEARCH and SEARCH-BY-RANK algorithms take the same time yet it is much simpler and more effective.

References:

[1] Michael T.Goodrich and Roberto Tamassia :Data structures and algorithms in java,fifth edition ,john and wiley sons,Inc.,2011.

[2] Wikipedia, the free encyclopedia.mht, skip list article.
http://en.wikipedia.org/wiki/Skip_lists

[3] Peter Brass: Advanced data structures , Cambridge university press, 2008.

[4] Erik Demaine ," introduction to algorithms" ,fall 2005.
http://videolectures.net/mit6046jf05_introduction_algorithms

[5] Julienne Walker, "skip list"
http://eternallyconfuzzled.com/tuts/datastructures/jsw_tut_skip.aspx

[6] Clifford A. Shaffer: A practical introduction to data structures and algorithm analysis,3'rd edition,January 2010.

[7] Erik D. Demaine and Charles E. Leiserson: "Introduction to Algorithms",2005.
http://ocw.mit.edu/.../lecture-12-skip-lists/6_046J2005L12.pdf