Skip lists are fascinating!

Jul 21

Skip lists are a fascinating data structure: very simple, and yet have the same asymptotic efficiency as much more complicated AVL trees and red-black trees. While many standard libraries for various programming languages provide a sorted set data structure, there are numerous problems that require more control over the internal data structure than a sorted set exposes. In this article, I will discuss the asymptotic efficiency of operations on skip lists, the ideas that make them work, and their interesting use cases. And, of course, I will give you the source code for a skip list in C#.

The time complexity of basic operations on a skip list is as follows:

Operation	*Time Complexity*
Insertion	O(log N)
Removal	O(log N)
Check if contains	O(log N)
Enumerate in order	O(N)

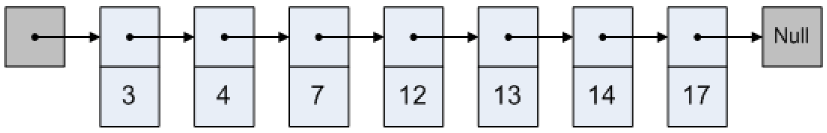
This makes skip list a very useful data structure. First, as mentioned earlier, skip list can be used as the underlying storage for a sorted set data structure. But, skip list can be directly used to implement some operations that are not efficient on a typical sorted set:

- Find the element in the set that is closest to some given value, in O(log N) time.
- Find the k-th largest element in the set, in O(log N) time. Requires a simple augmentation of the the skip list with partial counts.
- Count the number of elements in the set whose values fall into a given range, in O(log N) time. Also requires a simple augmentation of the skip list.

From a singly-linked list to a skip list

Sometimes the best way to understand how something works is to attempt to design it yourself. Let's try to go through that exercise with skip lists.

First, consider a regular sorted singly-linked list. Here is an example of one:



A sorted singly-linked list is not a terribly interesting data structure. The complexity of basic operations looks like this:

Operation	*Time Complexity*
Insertion	O(N)
Removal	O(N)
Check if contains	O(N)
Enumerate in order	O(N)

That is pretty unimpressive, actually. The only interesting value here is the O(N) in-order enumeration. For an insertion, removal or search, O(N) is about as bad as it gets. (There are more specialized use cases where sorted linked lists are appropriate, though.)

So, how can we make these operations on a sorted linked list faster? The main problem with a linked list is that it takes so long to get into its middle. That makes insertion, removal and search operations all O(N).

Well, here is an idea: let's consider a sorted multi-level list. We start out with a regular singly-linked list that connects nodes in-order. Then, we add a level-2 list that skips every other node. And a level-3 list that skips every other node in the level-2 list. And so forth, until we have a list that jumps somewhere

About

I am an engineer at a stealth-mode systems startup in downtown Mountain View, CA.

I can't say what we are building, but we are growing very rapidly and hiring software and hardware engineers. If that sounds interesting to you, message me at igoros@gmail.com.

Subscribe

617 readers BY FEEDBURNER

Feed

Twitter

Subscribe by email

Popular Posts

What really happens when you navigate to a URL
205 comments

Gallery of Processor Cache Effects
82 comments

7 tricks to simplify your programs with LINQ
45 comments

Fun with C# generics: down-casting to a generic type
29 comments

Fast and slow if-statements: branch prediction in modern processors
34 comments

Skip lists are fascinating!
45 comments

Volatile keyword in C# – memory model explained
29 comments

Efficient auto-complete with a ternary search tree
32 comments

Why computers represent signed integers using two's complement
17 comments

Use C# dynamic typing to conveniently access internals of an object
9 comments

Recent Posts

How RAID-6 dual parity calculation works

Is two to the power of infinity more than infinity?

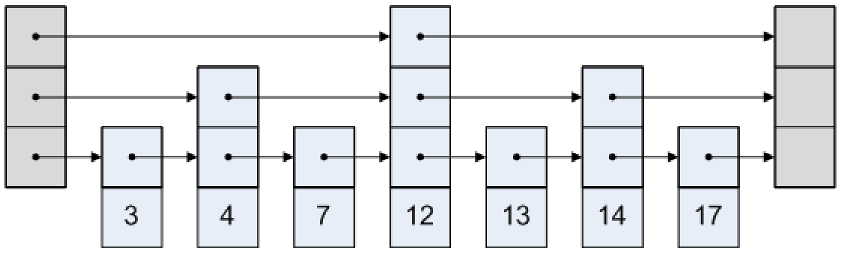
Why computers represent signed integers using two's complement

Graphs, trees, and origins of humanity

Fast and slow if-statements: branch prediction in modern processors

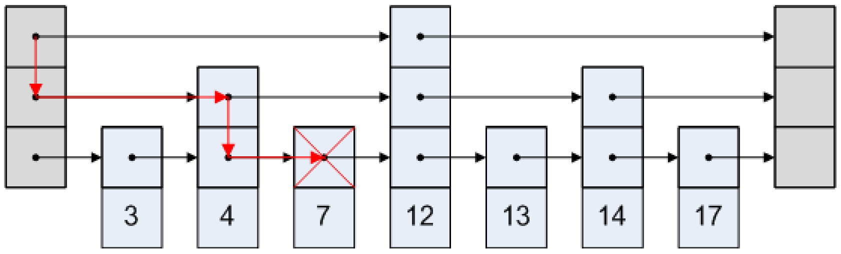
Recent Comments

Punci on What really happens when you navigate to a URL past the middle element. Our previous list now looks like this:



Now, checking whether a particular element is in the set only takes O(log N). The search algorithm is a lot like binary search. We first look in the top-most list and move to the right, making sure that we don't jump too far. For example, if we are searching for number 8, we will not take the level-3 link from the head node, because we would end up too far right: all the way at 12! If we can't move further right on a particular level, we drop to the next lower level, which has shorter jumps.

Search for value 8 would proceed like this:

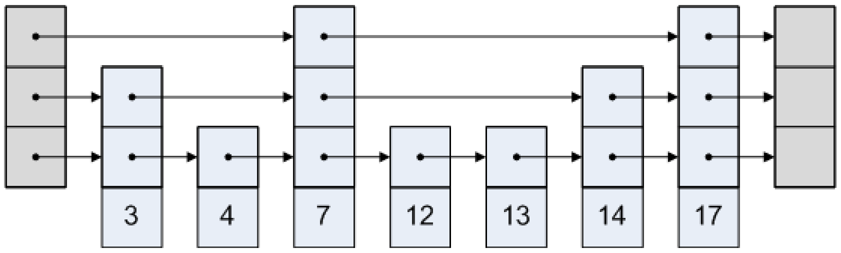


Since we landed on a 7, but we were looking for an 8, that means that 8 is not in the set.

The O(log N) search time is very nice. But, there is a problem: how do we implement insertions and removals efficiently, but in a way so that they maintain the structure of the multi-level list? This turns out to be quite a problem. AVL and red-black trees resolve it by tricky rebalancing operations.

Skip lists take an entirely different approach: a probabilistic one. Instead of ensuring that the level-2 list skips every other node, a skip list is designed in a way that the level-2 list skips one node on average. In some places, it may skip two nodes, and in other places, it may not skip any nodes. But overall, the structure of a skip list is very similar to the structure of a sorted multi-level list.

Here is an example of what a skip list may look like:



A skip list looks a bit like a slightly garbled sorted multi-level list. A skip list has many of the nice properties of a sorted multi-level list, such as O(log N) search times, but also allows simple O(log N) insertions and deletions.

Implementation of skip lists

So, given this description of a skip list, would you know how to implement it? It is not very hard, and it may be worth it to spend a couple minutes thinking about it.

- Insertion: decide how many lists will this node be a part of. With a probability of 1/2, make the node a part of the lowest-level list only. With 1/4 probability, the node will be a part of the lowest two lists. With 1/8 probability, the node will be a part of three lists. And so forth. Insert the node at the appropriate position in the lists that it is a part of.
- Deletion: remove the node from all sorted lists that it is a part of.
- Check if contains: we can use the O(log N) algorithm similar described above on multi-level lists.

And, all of these operations are pretty simple to implement in O(log N) time!

Source code

Here is a sample C# implementation for a skip list of integers:

```
class IntSkipList
{
    private class Node
    {
```

vietnam cheap flights on What really happens when you navigate to a URL
チアシードの食べ方 on What really happens when you navigate to a URL
imgur on What really happens when you navigate to a URL
Tabatha on How to write a self-printing program
published here on What really happens when you navigate to a URL
赤ちゃんが欲しい on What really happens when you navigate to a URL
honeymoon packages in kerala on What really happens when you navigate to a URL

Archives
[+]2014 (1)
[+]2011 (1)
[+]2010 (8)
[+]2009 (12)
[+]2008 (17)
[+]2007 (4)

```
    public Node[] Next { get; private set; }
    public int Value { get; private set; }

    public Node(int value, int level)
    {
        Value = value;
        Next = new Node[level];
    }
}

private Node _head = new Node(0, 33); // The max. number of levels is 33
private Random _rand = new Random();
private int _levels = 1;

/// <summary>
/// Inserts a value into the skip list.
/// </summary>
public void Insert(int value)
{
    // Determine the level of the new node. Generate a random number R. The numb
    // 1-bits before we encounter the first 0-bit is the level of the node. Sinc
    // 32-bit, the level can be at most 32.
    int level = 0;
    for (int R = _rand.Next(); (R & 1) == 1; R >>= 1)
    {
        level++;
        if (level == _levels) { _levels++; break; }
    }

    // Insert this node into the skip list
    Node newNode = new Node(value, level + 1);
    Node cur = _head;
    for (int i = _levels - 1; i >= 0; i--)
    {
        for (; cur.Next[i] != null; cur = cur.Next[i])
        {
            if (cur.Next[i].Value > value) break;
        }

        if (i <= level) { newNode.Next[i] = cur.Next[i]; cur.Next[i] = newNode;
    }
}

/// <summary>
/// Returns whether a particular value already exists in the skip list
/// </summary>
public bool Contains(int value)
{
    Node cur = _head;
    for (int i = _levels - 1; i >= 0; i--)
    {
        for (; cur.Next[i] != null; cur = cur.Next[i])
        {
            if (cur.Next[i].Value > value) break;
            if (cur.Next[i].Value == value) return true;
        }
    }
    return false;
}

/// <summary>
/// Attempts to remove one occurence of a particular value from the skip list. R
/// whether the value was found in the skip list.
/// </summary>
public bool Remove(int value)
{
    Node cur = _head;

    bool found = false;
    for (int i = _levels - 1; i >= 0; i--)
    {
        for (; cur.Next[i] != null; cur = cur.Next[i])
        {
            if (cur.Next[i].Value == value)
            {
                found = true;
                cur.Next[i] = cur.Next[i].Next[i];
                break;
            }

            if (cur.Next[i].Value > value) break;
        }
    }

    return found;
}
```

```
        /// <summary>
        /// Produces an enumerator that iterates over elements in the skip list in order
        /// </summary>
        public IEnumerable<int> Enumerate()
        {
            Node cur = _head.Next[0];
            while (cur != null)
            {
                yield return cur.Value;
                cur = cur.Next[0];
            }
        }
    }
```

Possible improvements

- Obviously, a more useful implementation would be generic, so that we can store values other than integers.
- Nodes can be structs instead of classes. This significantly reduces the number of heap allocations. But, we cannot use the null value anymore to represent the tail, which adds a bit of extra code.
- The skip list could be made associative, so that each node stores a key/value pair.
- The implementation I gave above is a multiset. It is pretty simple to change the skip list so that it implements a set instead.

Related:

- Original paper on skip lists [umd.edu]
- Skip list [wikipedia.com]
- Quicksort killer [igoro.com]
- Programming job interview challenge [igoro.com]

Tags: Algorithms

Posted by Igor Ostrovsky Algorithms Subscribe to RSS feed

45 Comments to "Skip lists are fascinating!"

Justin Etheredge says:
July 21, 2008 at 11:21 am

Man, that is really interesting! I'm going to have to dig into that a bit more later. Oh, and nice blog!

Remco says:
July 21, 2008 at 4:13 pm

here you go:

public class SkipList : ICollection where T : IComparable
{
private struct SkipListNode
{
private readonly N item;
private readonly SkipListNode?[] next;

public SkipListNode(N item, int depth)
{
this.item = item;
next = new SkipListNode?[depth];
}

public N Item
{
get { return item; }
}

public SkipListNode?[] Next
{

```
get { return next; }
}
}

private int count = 0;
private int depth = 1;
private SkipListNode head = new SkipListNode(default(T), 33);

public bool Contains(T item)
{
SkipListNode cur = head;

for (int level = depth – 1; level >= 0; level–)
{
for (; cur.Next[level] != null; cur = cur.Next[level].Value)
{
if (cur.Next[level].Value.Item.CompareTo(item) > 0)
{
break;
}

if (cur.Next[level].Value.Item.CompareTo(item) == 0)
{
return true;
}
}
}

return false;
}

public bool Remove(T item)
{
SkipListNode cur = head;

bool found = false;

for (int level = depth – 1; level >= 0; level–)
{
for (; cur.Next[level] != null; cur = cur.Next[level].Value)
{
if (cur.Next[level].Value.Item.CompareTo(item) == 0)
{
found = true;
cur.Next[level] = cur.Next[level].Value.Next[level];
count–;
break;
}

if (cur.Next[level].Value.Item.CompareTo(item) > 0)
{
break;
}
}
}

return found;
}

public void Add(T item)
{
// Determine the new depth of this new node. Retrieve the hashcode for value. The number
of
// 1-bits before we encounter the first 0-bit is the level of the node. Since hashcode is
// 32-bit, the level can be at most 32.
int nodeLevel = 0;

for (int hash = item.GetHashCode(); (hash & 1) == 1; hash >>= 1)
{
nodeLevel++;

if (nodeLevel == depth)
{
```

```
depth++;
break;
}
}

// Insert this node into the skip list
SkipListNode newNode = new SkipListNode(item, nodeLevel + 1);
SkipListNode cur = head;

for (int level = depth – 1; level >= 0; level–)
{
for (; cur.Next[level] != null; cur = cur.Next[level].Value)
{
if (cur.Next[level].Value.Item.CompareTo(item) > 0)
{
break;
}
}

if (level <= nodeLevel)
{
newNode.Next[level] = cur.Next[level];
cur.Next[level] = newNode;
count++;
}
}
}

public void Clear()
{
head = new SkipListNode(default(T), 33);
count = 0;
}

public void CopyTo(T[] array, int arrayIndex)
{
throw new NotImplementedException();
}

public int Count
{
get { return count; }
}

public bool IsReadOnly
{
get { return false; }
}

private IEnumerator GetTheEnumerator()
{
SkipListNode? cur = head.Next[0];

while (cur != null)
{
yield return cur.Value.Item;
cur = cur.Value.Next[0];
}
}

public IEnumerator GetEnumerator()
{
return GetTheEnumerator();
}

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
{
return GetTheEnumerator();
}

}
```

Reflective Perspective - Chris Alcock » The Morning Brew #141 says:
July 22, 2008 at 2:17 am

[…] Skip lists are fascinating! – Igor Ostrovsky takes a look at a list data structure known as Skip Lists – interesting reading […]

Remco says:
July 22, 2008 at 6:52 am

Minor update: count++ in method Add needs to be moved two lines below.

Remco says:
July 22, 2008 at 7:18 am

BTW: The initial filling for a skiplist is nicely distributed, but i'm curious to find out what happens when many Add / Remove operations are used. Will the list remain still as efficient? What will happen to the depth used?

And my implementation relies on GetHashCode in combination with IComparable, i don't know if that's a correct assumption to make. It works for ints, since GetHashCode returns the same int value. If i recall correct, there is no such restraint for the values returned by GetHashCode. I think it will be better to avoid the problem, by not relying on IComparable at all.

I've made a different version of this list, which implements a Set (only unique items are added in the list, no duplicates). This works lightning fast.

Remco says:
July 22, 2008 at 7:41 am

BTW: IMHO the durations for linked lists are on average O(N/2), not O(N)

In the best case scenario the item is in front of the list, in the worst case scenario the last item is retrieved, on average the desired item is located somewhere in the middle of the list.

Remco says:
July 22, 2008 at 8:05 am

I've tested skiplist, list and hashtable, here are the results:

Testing SkipList vs List vs Hashtable, using 100000 items

[SkipList]
– Insertions : 455378
– Contains : 1005696
– Remove : 1480769

[List]
– Insertions : 8798
– Contains : 146389620
– Remove : 171897388

[HashTable]
– Insertions : 106101
– Contains : 160336
– Remove : 228002

AlexR says:
July 22, 2008 at 11:10 am

I don't see difference between skip list and binary tree without rebalancing operations. Is skip list more effective or not?

Igor Ostrovsky says:
July 22, 2008 at 12:12 pm

AlexR: A skip list is like a balanced tree. At least, it is like a balanced tree with an extremely high probability.

How does skip list do that? These two properties ensure it:
1. About 1/2 of nodes are in the lowest list only, about 1/4 nodes are in the lowest two lists, 1/8 in the lowest three, and so forth.
2. Nodes with different "heights" are mixed up about evenly.

After any sequence of insertions and removals, properties (1) and (2) still hold.

You can define these properties much more precisely, and use them to prove that insertion, removal and lookup are O(log N) on a skip list, with a very high probability.

Makes sense?

Igor Ostrovsky says:
July 22, 2008 at 12:34 pm

Remco:

Wow, seems like you looked at skip lists pretty in depth! Awesome, experimentation is a great way to learn. Here are responses to some of your points:

– The "distribution" of nodes: see answer to AlexR I just posted.

– GetHashCode: if two values are equal, then their hash codes are guaranteed to be equal. But, hash codes are not intended for less-than-greater-than comparison, and don't work that way in general.

– O(N) vs O(N/2): it is true that in a sorted list, a lookup will visit N/2 nodes on average. In the big-Oh notation, O(N) is the same as O(N/2), and it is customary to drop the constant factor of 1/2.

– Note that a skip list is an ordered data structure, so it can do a number of things efficiently that a hash table cannot: e.g. in-order enumeration, find the smallest element larger than X, etc.

– When running your benchmarks, make sure you are measuring a RELEASE build rather than a DEBUG build.

Hope that helps, and thanks for reading.

Remco says:
July 23, 2008 at 3:51 am

@Igor: Agreed. Benchmarking was done in release mode (debugging takes to long to run ;-)). The Hashtable is faster in inserts and removals, in-order iteration is not be supported by an hashtable (order is not gauranteed / used).

One last note on 'my' implementation: i think it's better (in general) to restore the randon number generation instead of using the hashcode to determine the depth of the node. Using hashcodes in case of integers will generate a nice distribution, but for all other types all bets are off 😉

Requiring where T : IComparable on the list is desirable.

(i just noticed that the type parameters in my sample are missing, just the better, this will prevent viewers from using it as-is, with all the known defects ;-))

Alex Miller says:
July 26, 2008 at 10:05 pm

Nice article! I wrote an article on skip lists a while back you might find interesting. It has links to a lecture available on iTunes from MIT that covers skip lists in some detail as well:

http://tech.puredanger.com/2007/10/03/skip-lists/

You should really also check out the ConcurrentSkipListMap included in Java 6:

http://java.sun.com/javase/6/d.....stMap.html

One of the big benefits of skip lists is that because they are built around linked lists, they can exhibit very localized locking. ConcurrentSkipListMap is a concurrent skip list based map and is written in lock free style (no synchronization, relies solely on CAS-like operations). This makes ConcurrentSkipListMap a concurrent sorted map with excellent performance. I'd recommend reading the source code as well as it is extremely well written and a great example of lock-free coding.

Weekly Web Nuggets #22 : Code Monkey Labs says:
July 28, 2008 at 11:00 am

[…] Skip Lists Are Fascinating: Igor Ostrovsky takes a look at the skip list, a simple, yet powerful data structure. […]

kim says:
August 8, 2008 at 1:16 am

I like it!!!!

Ion Sapoval says:
September 9, 2008 at 4:29 am

I like this type of data structure for it's simplicity and efficiency. Keep going with this kind of articles they're very useful. Thanks

Pablo says:
April 14, 2009 at 2:52 am

In fact all this information + sample code was posted on on MSDN a while ago.
Do the guys in MS use MSDN 😃 ?

Linked lists:
http://msdn.microsoft.com/en-u.....573(VS.80).aspx

Other data structures (queue, stack, BT, BST, etc.):
http://msdn.microsoft.com/en-u.....091(VS.80).aspx

Igor Ostrovsky says:
April 14, 2009 at 3:32 am

@Pablo: Ah, cool. I didn't know about the MSDN article.

My approach to explaining skip lists is significantly different from the explanation in the MSDN article, and there are differences in the implementation as well.

So, I wouldn't say that my post is redundant.

Soumeya Tarfi says:
August 3, 2010 at 4:56 am

I read the article about skip lists on msdn and I read your article. I like them both but I have to say your article is an awesome quick read and gets to the point faster. I didn't know much about skip lists and I found your article pretty straight forward. Thanks!

bmakowsky handbags says:
December 12, 2010 at 11:31 pm

Outstanding article over again. Thanks!

Matthieu says:
January 5, 2011 at 10:52 am

@Igor: Actually, the factor of nodes from one level to another need not be fixed to 1/2. Other factors can be used, and I had read experiments that 1/4 turned out better in the general case. Now of course… I just can't find it back… but knowing that the factor can change is already something I guess.

Barry Kelly says:
March 14, 2011 at 12:20 am

```
for (int R = _rand.Next(); (R & 1) == 1; R >>= 1)
```

For C# on .NET, this may work fine, but in general, this is a bad way of extracting a 0/1 choice from a pseudo-random number generator (PRNG). Many PRNGs use linear congruential generators, which are just a multiply and an add, and have highly predictable low bits as a result (frequently just a repeating pattern with a short period, as short as 4 or 8).

Safer:

```
while (_rand.Next(2) == 0)
```

– or even simply reading the bits off the other end.

Elmer says:
March 14, 2011 at 1:20 am

Hello Igor,

Very nice blog. I happen to stumble on your blog when a fellow user in StackOverflow posted the link to your blog as part of his answer to my question.

Do you have ideas on where we can be able to apply Skip Lists? Or any idea on what program to create so that students can be able to grasp its concepts easily?

Cheers!

Igor Ostrovsky says:
March 14, 2011 at 2:29 am

Barry Kelly: Yeah, reading the bits in the opposite order sounds like a good idea, especially if you are concerned about the quality of the random number generator. Using _rand.Next(2) sounds reasonable too, although that solution may turn out slower – hard to say without trying it out and measuring.

links for 2011-03-14 « Blarney Fellow says:
March 14, 2011 at 7:24 pm

[…] Skip lists are fascinating! (tags: algorithm data-structure tree search) […]

Jingyi says:
March 28, 2011 at 1:37 pm

I guess I didn't get it. I don't see the O(lgN) for the insertion in this implementation. For example, inserting into the lowest level, it is already O(N) because you have to traverse the entire list. To me, the implementation for insert is N + N/2 + N/4 + … + N/pow(2, N) = 2N, which is O(N), am i right here? The same thing for Contains. Let's say, there is a value that is not in the list, when you search it, you have to traverse lgN list and the longest list is N. Can anybody explain that to me?

Jingyi says:
March 28, 2011 at 1:43 pm

Sorry, I guess I didn't understand the code completely. I was wrong, it didn't traverse the entire list for the lower levels.

Duc says:
April 10, 2011 at 11:28 am

Hello Elmer,

Here is an real life application for skiplist.

An application breaks a stream of media data (65KB) into 1000 segments of 65B (bytes – these bytes are numbered; hence ordered) for delivered to receivers (UDP traffics). It is very possible and very likely in wireless environment, these will be arriving out of order; however, the stream must be in order before it can be presented to the presentation application; Hence these segment must be assembled into an order list in kernel. In this scenario, skiplist is very good since the insert can be done in O(log n); balanced trees (trees) might not be so good.

Similar application can be used for ip fragmentation reassembling; however in this case, a simple linked list will do since the number of fragments belonging to an ip fragement stream is usually small (8 or less fragments).

Hope this helps.

Regards
DL

vinay polisetti says:
October 30, 2011 at 5:02 am

Nice article explaining the use cases of skip lists !!! find some more of them in http://cracktheinterview.org/

Thanks !!

Cheers !!

neetu says:
September 5, 2012 at 12:44 pm

good:)
now i want the difference between a hash table and skip lists..

putri says:
December 18, 2012 at 5:04 am

very nice blogs
this is nice article....can you give me example code of skip list in c++, i'am so need it to finish my task...

aabhas says:
April 6, 2013 at 7:59 am

Can u explain the last line of insert function Why is "if (i <= level) { newNode.Next[i] = cur.Next[i]; cur.Next[i] = newNode; }"
i can get ur.next[i]=newNOde but not newNode.Next[i]=cur.Next[i]

yacineb says:
June 28, 2013 at 4:16 am

In a skiplist , the probability that a node reaches a level k , follows a geometrical law with a parameter p=0.5.

The theorical maximum average level in the skip list is = int(log2(N))+1 where N is the maximum number of elements in the list ==> int(log2(int.MaxValue = 2^31 -1))+1 = 32

Anyway.. there is no need to go further than 32 because the probability that a node "reaches" the 32th level is already extremely low = (0.5)^32

There is a better way to get a random level for a node.
Look at this function :

```
private int GetRandomHeight() {
int level = 0;
while (_rand.NextDouble() < 0.5 && level < 32)
{
++level;
if (level == _levels) { ++_levels; break; }
}
return level;
}
```

Scott says:
July 6, 2013 at 6:03 am

Very useful article, thankyou.

An interesting way to generate a random level (at least in C) is to generate a random 32-bit number 'r' then use a count leading zeros function e.g. __builtin_clz(r). Assuming each bit has 0.5 probability of being a 1 or a 0 you'll get the required distribution of levels. Remember to cap the level at your chosen maximum though. Also, you can't use the stdlib rand() function without taking into account the value of RAND_MAX, on my machine it will never set the top bit. Count trailing zeros might be a more foolproof operation but the ARM architecture has no count trailing zeros instruction which is what I'm working on.

```
level = MIN( __builtin_clz( gen_rand_u32() ), MAX_LEVELS );
```

Horia says:
October 16, 2013 at 1:39 pm

very nice article,

how would you implement the partial counts in order to find the k-th largest element?

you cannot update the current partial counts at every insertion since that would modify the O(logN) insertion time, right?

Igor Ostrovsky says:
October 16, 2013 at 3:08 pm

Horia: for every arrow, you'd track how many elements it skips over.

Naveen says:
November 8, 2013 at 8:17 am

What is the real life application of skip list?

Vlad says:
February 28, 2014 at 2:20 pm

A cleaner more mathematical way to select levels is using a log base 1/2 of x, floored with a cast to int. I wrote a little article explaining it, if you are interested: http://echolot-1.github.io/skip-list-levels/

Pedro says:
March 2, 2014 at 8:13 pm

This is way better than MSDN Developer Network´s explanation, thanks a lot

David Piepgrass says:
May 26, 2014 at 1:47 pm

The implementation as given is very inefficient. The proposed optimization "Nodes can be structs instead of classes" doesn't work – despite Remco's version – because nodes point to each other and you can't make references to structs. What you could do, though, is store all your nodes in a single List:

```
partial class SkipList<T> {
. struct Node {
. . public int[] Next;
. . public T Value;
. }
. List<Node> _data; better yet: use InternalList<Node> from
. //
https://github.com/qwertie/Loyc/blob/master/Src/Loyc.Essentials/Collections/Implementati
. int _firstEmptyCell;
}
```

Then rather than a normal linked list, you use indexes as if they were pointers; Next[i] is the index of the next item. When "removing" an item, just clear its Node.Value to default(T) and add it to a singly-linked list of "empty cells" which will be used later when adding items (this implies that you can't easily reduce memory usage of the list, but this is how .NET collections normally work anyway)

Then, for less memory usage, eliminate the Node struct and the _data list and instead use this:

```
List<T> _values;
List<List> _next;
```

in this case _next[j][i] stores the value that used to be stored in _data[i].Next[j], such that _next[j].Count == _values.Count.

David Piepgrass says:
May 26, 2014 at 1:50 pm

(List<List> _next; should have said List<List<T>> — I neglected to convert one of the "<" signs to "&lt;")

Ivory says:
May 22, 2015 at 8:57 pm

So folks should find the origins in their eyes issues by removing their ranges.

That's since whenever you don lenses, your sight are better to become tired.

Pham The Trung says:
March 7, 2016 at 6:50 am

Your remove and contains method have bugs. You need to handle case when head.value == value

Pham The Trung says:
March 7, 2016 at 6:52 am

Oops, sorry, head doesn't have any value

Dale says:
April 25, 2016 at 8:01 am

Very nice explanation.

iq option Regolamentato says:
May 4, 2016 at 12:49 am

Having read this I thought it was very informative.
I appreciate you finding the time and energy to put this content together.
I once again find myself spending a significant amount of time both reading and posting comments. But so what, it was still worth it!

Leave a Reply

Name

Email (will not be published)

Website

You can use these tags: <a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>

Submit Comment

RSS