

C/C++

Tweet

Like 11

Share

G+



Permalink

Skip Lists in C++

By Bill Whitney, November 01, 1998

[2 Comments](#)

Skip lists are an efficient alternative to balanced trees, and rather easier to implement correctly.

If you're like me, you're always looking for an alternative data structure that not only performs admirably, but is easy to implement and understand as well. The skip list, described by William Pugh in "[Skip Lists: A Probabilistic Alternative to Balanced Trees](#)" [PDF] fits this description.

Skip lists are ordered key/object-based containers offering excellent performance with minimal processing overhead. Although skip lists resemble linked lists, they actually have more in common with balanced trees when it comes to performance. The most important difference is that skip lists require none of the periodic reorganization required with balanced trees — the searching, insertion, and removal algorithms are simpler to code, understand, and extend.

How Skip Lists Work

Figure 1 shows a skip list containing a product inventory. All data is contained in a node and identified by a key. Probably the most outstanding feature of the nodes is the variations in their heights. By filling a list with nodes of varying heights, the search algorithm can "see" past subsequent nodes when searching for a particular value. For example, given a node list in which 50% of the nodes are twice as high as the rest, search time is cut in half. The algorithm needs to look at only half of the values.

Figure 1 - Product Inventory

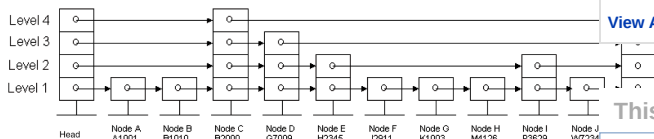


Figure 1: A sample skip list

As the height of the nodes increase, the visibility between nodes has the potential to be much greater, resulting in even shorter searches. In the ideal skip list, nodes of designated heights would be distributed throughout the list at pre-determined intervals. This would guarantee the maximum benefit from the nodes' look-ahead capability. To sustain performance, however, skip lists employ a probabilistic approach to assigning node height. This approach also removes the requirement for constant reorganization to maintain pre-determined intervals. I'll show the algorithm for probabilistic node height a little later. Consult the reference at the end of this article for a detailed mathematical analysis.

Looking back at Figure 1, the head and tail are actual nodes in the skip list, but they serve only as markers for the beginning and end; they contain no useful data. The head points to the first visible node at each height (only A and C in Figure 1). Any node for which there exists no subsequent node at its height (C, D, I and J) points to the "tail" node. Suppose a search is made for product code H2345 in the product inventory. Each search starts at the head of the skip list, comparing the sought-after key with the first node pointed to by the root node at the list's current height.

Before I go any further, I should explain the concept of "maximum height." The maximum height of a skip list defines the number of vertical pointers the skip list can maintain. I use the term "current

C/C++ Recent Articles

[Dr. Dobb's Archive](#)
[Jolt Awards 2015: Coding Tools](#)
[Building Node.js Projects in Visual Studio](#)
[Building Portable Games in C++](#)
[C# and .NET's Sudden Ubiquity](#)

Most Popular

[Stories](#) [Blogs](#)

[The C++14 Standard: What You Need to Know](#)
[A Simple and Efficient FFT Implementation in C++:](#)
[Part I](#)
[State Machine Design in C++](#)
[Lambdas in C++11](#)
[A Lightweight Logger for C++](#)

Your System Status

WE'RE SORRY!

You need to update your Flash Player.



IMPORTANT: After installing the required upgrade please reload this browser window to view the video player.

[View All Videos](#)

This month's Dr. Dobb's Journal



This month, Dr. Dobb's Journal is devoted to mobile programming. We introduce you to Apple's new Swift programming language, discuss the perils of being the third-most-popular mobile platform, revisit SQLite on Android, **and much more!**

[Download the latest issue today. >>](#)

Upcoming Events

[Live Events](#) [WebCasts](#)

[INsecurity - A Dark Reading Conference: Join Us](#)
[November 29-30 In The D.C. Area - INsecurity](#)
[Transform Business Processes with API-enabled](#)
[Integrations - Enterprise Connect Orlando](#)
[Registration is Open for Enterprise Connect](#)
[Conference and Expo - Enterprise Connect](#)

height" to refer to the tallest node that has been inserted into the skip list so far. Because node height is probabilistic, current height reaches maximum height as a skip list matures. This will become clearer when I present the code.

Referring to Figure 1, the product sought (H2345) is located in node E. This sample skip list has a current height of 4, a maximum height of 4, and contains ten nodes labeled A through J. All searches start at the root node, from the level indicated by current height (4). Root node level 4 points to node C at level 4. Comparing the sought key (H2345) with node C's key (B2000), shows that the search is still below the value needed. Next, the search algorithm follows node C's level 4 pointer, which brings it to the list's tail. That's obviously too far, so the search drops down to node C's level 3 pointer and looks ahead to node D. Node D's key value is G7009, which is too low. Node D's level 3 pointer leads to the list's tail again so the search drops down to level 2 and looks ahead. The next node is E, and checking its key produces a match. The search algorithm had to traverse three nodes to locate the key value H2345 in node E. If it were looking for the value P3629 in node I (9th node in the list), the algorithm would have had to look at only four nodes.

The C++ Implementation

I've implemented the skip list in two template classes, `SkipList` and `SkipNode`. I also use a helper class called `RandomHeight`, which is based on Pugh's algorithm for generating probabilistically random node heights. `RandomHeight` is called upon by the `SkipList` `insert` method when new nodes are generated. It appears in Listing One (`RandomHeight.h`) and Listing Two (`RandomHeight.cpp`).

Listing One

```
1  #ifndef SKIP_LIST_RANDOM
2  #define SKIP_LIST_RANDOM
3  class RandomHeight
4  {
5  public:
6      RandomHeight(int maxLvl, float prob)
7          : ~RandomHeight() {}
8          int newLevel(void);
9
10 private:
11     int maxLevel;
12     float probability;
13 };
14 #endif //SKIP_LIST_RANDOM
15
16 /* End of File */
```

Listing Two

```
1  #include <stdlib.h>
2  #include "RandomHeight.h"
3
4  RandomHeight::RandomHeight
5      (int maxLvl, float prob)
6  {
7      randomize();
8      maxLevel = maxLvl;
9      probability = prob;
10 }
11
12 int RandomHeight::newLevel(void)
13 {
14     int tmpLvl = 1;
15     // Develop a random number between 1
16     // maxLvl (node height).
17     while ((random(2) < probability) &&
18           (tmpLvl < maxLevel))
19         tmpLvl++;
20
21     return tmpLvl;
22 }
23
24 //End of File
```

The `RandomHeight` constructor lays the ground rules for the scope of the random numbers to be generated. These rules consist of the maximum node height (`level`), and the percentage of nodes that should appear at level 1. When new nodes are created, a random node height is obtained from `RandomHeight`'s `newLevel` method. Given a height of 4 and a percentage of 0.5, for example, 50% of the generated values would produce level 1 nodes, 25% of the nodes would be level 2, 12.5% of the nodes would be level 3 nodes, and so on. The `random(2)` call produces a number between 0 and 1 using my compiler.

Orlando

Featured Reports

[What's this?](#)

Cloud Collaboration Tools: Big Hopes, Big Needs
 Return of the Silos
 Strategy: The Hybrid Enterprise Data Center
 SaaS and E-Discovery: Navigating Complex Waters
 Database Defenses

[More >>](#)


Featured Whitepapers

[What's this?](#)

The Role of the WAN in Your Hybrid Cloud
 Mid-Market Mayem: Cybercriminals Wreak Havoc
 Beyond Big Enterprises
 Overview: Cloud Operations Platform for AWS
 Case Study: Gilt
 Coding to standards and quality: supply-chain
 application development

[More >>](#)


Most Recent Premium Content

Digital Issues

2014

Dr. Dobb's Journal

November - [Mobile Development](#)
 August - [Web Development](#)
 May - [Testing](#)
 February - [Languages](#)

Dr. Dobb's Tech Digest

DevOps
 Open Source
 Windows and .NET programming
 The Design of Messaging Middleware and 10 Tips from
 Tech Writers
 Parallel Array Operations in Java 8 and Android on x86:
 Java Native Interface and the Android Native
 Development Kit

2013

January - [Mobile Development](#)
 February - [Parallel Programming](#)
 March - [Windows Programming](#)
 April - [Programming Languages](#)
 May - [Web Development](#)
 June - [Database Development](#)
 July - [Testing](#)
 August - [Debugging and Defect Management](#)
 September - [Version Control](#)
 October - [DevOps](#)
 November - [Really Big Data](#)
 December - [Design](#)

2012

January - [C & C++](#)
 February - [Parallel Programming](#)
 March - [Microsoft Technologies](#)
 April - [Mobile Development](#)
 May - [Database Programming](#)
 June - [Web Development](#)
 July - [Security](#)
 August - [ALM & Development Tools](#)
 September - [Cloud & Web Development](#)
 October - [JVM Languages](#)
 November - [Testing](#)
 December - [DevOps](#)

2011

[1](#) [2](#) [3](#) [4](#) [Next](#)

Related Reading

- [News](#)
- [Commentary](#)
- [Application Intelligence For Advanced Dummies](#)
- [AppGyver AppArchitect 2.0 Appears](#)
- [Applause Launches Mobile Beta Management](#)
- [XMind 6 Public Beta Now Available](#)
[More News»](#)
- [Slideshow](#)
- [Video](#)
- [Developer Reading List](#)
- [The Most Underused Compiler Switches in Visual C++](#)
- [C++ Reading List](#)
- [Jolt Awards 2013: The Best Programmer Libraries](#)
[More Slideshows»](#)
- [Most Popular](#)
- [RESTful Web Services: A Tutorial](#)
- [Developer Reading List: The Must-Have Books for JavaScript](#)
- [Hadoop: Writing and Running Your First Project](#)
- [Unit Testing with Python](#)
[More Popular»](#)

More Insights

White Papers

- [The Role of the WAN in Your Hybrid Cloud](#)
- [Driving Your Cloud Strategy with Private Network Solutions](#)

[More >>](#)

Reports

- [Cloud Collaboration Tools: Big Hopes, Big Needs](#)
- [State of Cloud 2011: Time for Process Maturation](#)

[More >>](#)

Webcasts

- [Catch the Security Breach Before It's Out of Reach](#)
- [Architecting Private and Hybrid Cloud Solutions: Best Practices Revealed](#)

[More >>](#)

INFO-LINK

[Login or Register to Comment](#)



[mskelton068](#)

Using a construct such as the new `unique_ptr` from C++0x would make ownership semantics clear, and would correctly handle deletion of client-created objects.



[Christopher Yeleighton](#)

The `SkipNode` keeps its data pointers from outside, keeps them and deletes them at its end of life. It does not make sense to me; you should not delete objects created by the client.



TECHNOLOGY GROUP

Black Hat
Content Marketing Institute
Content Marketing World
Dark Reading

Enterprise Connect
Fusion
GDC
Gamasutra

HDI
ICMI
InformationWeek
INsecurity

Interop ITX
Network Computing
No Jitter
VRDC

COMMUNITIES SERVED

Content Marketing
Enterprise IT
Enterprise Communications
Game Developers
Information Security
IT Services & Support

WORKING WITH US

Advertising Contacts
Event Calendar
Tech Marketing
Solutions
Contact Us
Licensing

[Terms of Service](#) | [Privacy Statement](#) | [Legal Entities](#) | Copyright © 2017 UBM, All rights reserved

- Dr. Dobb's Home
- Articles
- News
- Blogs
- Source Code
- Dobb's TV
- Webinars & Events
- About Us
- Contact Us
- Site Map
- Editorial Calendar