Comparing shapes of randomly generated binary search trees without balancing

Zino Onomiwo - zinoonomiwo@gmail.com
Joep van den Hoven - j.d.vandenhoven@students.uu.nl
Marijn Nijsse - marijn.nijsse@gmail.com
Ivo Gabe de Wolff - i.g.dewolff@uu.nl

November 7, 2016

Abstract

When binary search trees are generated from an uniformly shuffled list of numbers using only random inserts they become balanced. Interestingly enough not much is known about also performing random deletions. Using several different methods of generating binary search trees, such as for every random insertion also performing a random deletion, we can deduce if such trees end up balanced. This would mean that trees generated by random inputs and deletes do not necessarily need a balancing algorithm. By comparing the results of these different methods our research concludes that most trees likely become balanced, our research is however limited by the amount of nodes we can realistically add or remove.

1 Introduction

In this paper we discuss the results of our research about binary search trees. A binary search tree is a data structure, that can contain ordered data. A common data type is a key-value pair, with an ordering on the key. The structure supports some basic operations, insert, lookup and delete, which run in $\mathcal{O}(h)$, where h is the height of the tree. The height is defined as the number of edges on the longest simple downward path from the root to a leaf [3, appendix B.5.2]. A node in the tree has a value, a left child and a right child. Those children can either be nodes or leaves. The value of the node is larger than all values of all nodes under the left child (including the left child), and smaller than the values under the right child.

To illustrate the behavior of a tree, we will demonstrate the procedure for querying a tree. We compare the value in the root of the tree with the queried value. If they equal, we can return the root. Otherwise, we need to take a look at the left subtree (if the value in the root is larger than the queried value) or the right subtree. We start the same procedure for the root of this subtree, until we find a leaf. In that case, the queried value does not exist in the tree. This operations 'walks' to the bottom of the tree, so one can clearly see that the running time is limited by the height of the tree.

1.1 Balancing

The operations on a binary tree depend on the height of the tree. If the height of a tree is $\mathcal{O}(\log n)$, we call it a balanced tree. Note that this definition is actually about a 'family' of trees instead of a single tree, since the definition uses the asymptotic behavior of the height when n goes to infinity.

Many implementations of binary search trees use a balancing algorithm to ensure the tree is rebalanced afterwards. Our research is about trees that do not have such a balancing algorithm, and concerns trees generated by a sequence of insert and delete operations.

We want to find out whether a tree, generated by a random sequence of inserts and deletes, is balanced. To test the balance of a tree, we looked at:

- Height of the tree
- Average depth of the nodes
- Average depth of the leaves
- Variance of the depth of the leaves

The *height* of a balanced tree is always within $\mathcal{O}(\log n)$. The average *depth* of the nodes gives an indication for the running time for a random successful lookup. The average depth of the leaves indicates the time required for a random unsuccessful lookup, if all leaves have the same probability to be reached. We chose to measure these variables because the combination of all three gives a solid indication of the balance state and the running time of common operations.

In the following sections we will start by giving a more in depth description of our research and discussing the different variables which we measured for each tree. Further chapters describe and concisely display the results of our numerous tests. Finally we draw conclusions based on the findings derive from the tests.

2 Research

This research was performed by a group of four students as part of an assignment at the Utrecht University. Our interest in this project was sparked when we read a small section in Rivest R. et al (2009)[3] explaining the lack of knowledge on this subject, which read as follows:

> "Unfortunately, little is known about the average height of a binary search tree when both insertion and deletion are used to create it."

2.1 Research question

Based on our previous motivation we derived the following research question:

> Does a random order of inserts and deletes generate a balanced binary search tree, when the structure has no balancing algorithm?

2.2 Research description

In our research we will look at a binary search tree generated from random sequence of inserts and deletes. We are interested in seeing how *balanced* on average the resulting trees will be.

Since a balanced tree is defined by having a height of $\mathcal{O}(\log n)$, where n is the amount of nodes, looking at the height of a random tree will give a good indication of how balanced the tree is. When it comes to search algorithms, the height of a tree can give us an indication of what the maximum amount

of steps for a search might be. However, only checking the total height of the tree can give a false image of the balance, since the height can be much larger than the average depth. Such a tree will have a large height due to one or more long branches sticking out. Therefore it is necessary to look other properties of the trees as well.

Another way to measure the balance is to look at the average depth of all the nodes in a tree. The *average depth* of the nodes tells us if the nodes are densely clustered in a certain part of the tree, which is the case for a balanced tree, where the nodes will be clustered in the bottom few layers of the tree. Additionally the average depth of a tree is also an indication of the duration of an average search query will take for this tree.

Our last method of measuring the balance is to check the variance of the depths of all leaves. The *variance* of the depths of all leaves indicates how much each leaf differs in depth from the other leaves in the tree. Measuring this variance in a generated tree gives an insight in the possible imbalance of the tree, where a low variance means a more balanced tree.

For randomly generated binary search trees, it is known that for the height h and amount of nodes n, when n approaches infinity, $\frac{h}{\log n}$ goes to 4.31107 [1]. As we start out with a rather small n and approaching an infinitely large n it becomes beyond the realm of possibilities for us to research. The value of n is limited in our experiments, so it would not be a fare to compare this with the measured values for our generated trees.

Instead, we will compare the generated trees with a base-case, namely a tree generated with only insert operations.

3 Related work

Eppinger, J. (1983) [2] has done early research on the effects of performing insertions and deletions on the length of unbalanced binary search trees. Eppinger found that using a large number of insertions and an asymmetric deletion algorithm yields an increased internal path length, which will eventually become worse than the internal path length of a random tree. Using symmetric deletions the opposite occurs and the internal path length actually decreases. This effectively means that different types of deletion algorithms used on large random trees should yield either faster or slower average search speeds in the resulting trees.

In other work done by Martinez, C. and Roura, S. (1998) [4], they presented randomized algorithms for insertions and deletions that always produce random trees. This is in contrast with the earlier work done e.g. Eppinger, J. (1998) [2], who stated that repeatedly inserting and deleting nodes

in a random tree yields a tree that is no longer random. The trees end up random for Martinez and Roura, since they vary their algorithms for insertions and deletions based upon the size of a given tree.

Vinod P. et al [5] have described insertion and deletion methods that cause a tree to have a size of at most n / 2 without using a complex rebalancing algorithm. Even large insertions and asymmetric deletions can be handled by their solution. Thus, the relatively balanced state, of a tree can be preserved with relative ease, causing it to be more like a random tree. However, a random tree would not take any action should either of those operations occur, which could upset the balance of a tree. Our research only considered the random operations, which tend to be symmetric for the most part. Looking into the overall effect of such asymmetric operations would be an interesting topic for further research. However, we still provide insight into what happens to purely random trees.

4 Description of tests

In order to find an answer to our research question we developed a Haskell program that generated a large amount of random trees with both inserts and deletes. In our tests we generate trees using three different methods of handling deletes compared to data gathered from trees generated with inserts only.

We used two adjustable parameters, n and k, parameter n dictating the number of elements in the resulting tree and the parameter k dictating the number of deletes. So, during the creation of the tree, $n + k$ elements are inserted, of which k are deleted. We could then vary these numbers to see if it had a noteworthy influence.

4.1 Test program

The program generates the trees, with varying parameters n and k, using different methods to create a sequence of inserts and deletes (which we call *generators*). For all generated trees, we measure the height, average node depth, average leaf depth and leaf depth variance. Each setting is run eight times, and the result is averaged. The program exports its data to csv-files. The source code can be found at `https://github.com/ivogabe/random-binary-tree`.

4.2 Generators

As mentioned before, we used three different ways to construct a tree using inserts and deletes, alongside a fourth method which did not do any deletions at all as a base-case. Every deletion method uses symmetric deletions. We call a strategy to construct the tree based on a sequence of inserts and deletes a *generator*.

The first generator, *inserts-then-deletes*, was rather elementary, all that took place was the $n + k$ random inserts followed by k deletions. The second generator, *inserts-and-deletes*, intermingles the inserts and deletes. This generator creates a random list of $n + 2k$ numbers, where $2k$ of them are pairs. The first occurrence of a number refers to an insertion, the second to a deletion.

The last and probably most complicated generator was *lifetime*. It starts by creating a list of $n+k$ elements, which corresponds to the insert operations. k of these elements are given a so-called lifetime. Such an element was deleted after l insertions.

Using each of these methods a sizable amount of trees was generated and recorded. The results of this work can be found in the next chapter which contains the results of the many iterations we ran. We used the following parameters for the generators, with $n \in \{16, 24, 32, 48, 64, 96, 128, 192, 256, 384, 512, 768, 1024, 1536, 2048, 3072, 4096\}$:

- *inserts-then-deletes*: where k can be n, $10n$, $\frac{n^2}{8}$ or $\frac{n^3}{32}$
- *inserts-and-deletes*: where k can be n, $10n$, $\frac{n^2}{8}$ or $\frac{n^3}{32}$
- *lifetime*: where k can be n, $10n$, $\frac{n^2}{8}$ or $\frac{n^3}{32}$ and $l = \frac{n}{2}$

Tests with $k = \frac{n^2}{8}$ were restricted to $n \leq 2048$ and tests with $k = \frac{n^3}{32}$ were limited to $n \leq 512$.

5 Test results

We included the raw data including the calculation for the t-test as an attachment. The generator names are abbreviated: `itd` refers to *inserts-then-deletes*, `iad` to *inserts-and-deletes* and `life` refers to *lifetime*. The value for parameter k is abbreviated too, `1` means $k = n$, `10` means $k = 10n$, `n` means $k = \frac{n^2}{8}$ and `n2` means $k = \frac{n^3}{32}$. This section describes the most important results.

5.1 Height of the tree

We compared the heights of random trees generated with only inserts with trees generated with inserts and deletes. For all different tree generators, we paired the size of the generated tree with the base-case (a tree generated with only inserts), where both trees have the same number of nodes.

All tree generators except one generated a tree with a smaller height than the base-case. Only *inserts-and-deletes* with $k = \frac{n^2}{8}$ created a tree with a larger height. However, the height was not significantly larger (with $\alpha = 0.05$).

All other generators gave a lower height. This difference was significant ($\alpha = 0.05$) in only three cases:

- *inserts-and-deletes* with $k = 10n$
- *lifetime* with $k = 10n$
- *lifetime* with $k = \frac{n^2}{8}$

The first difference is not significant for $\alpha = 0.05$. The other two were significant for the higher confidence level.

5.2 Average depth

The average depth of the nodes was lower for all tree generators. The difference was significant only for the *lifetime* tests. The average depth of the leaves was in only case higher than the base-case, namely in *inserts-then-deletes* with $k = \frac{n^2}{8}$. All *lifetime* test yielded a significant lower average depth of the leaves, for $\alpha = 0.05$.

5.3 Leaf depth variance

Our last measurement is the variance in the depths of the leaves. For all generators, we found a lower variance than the base-case. This difference was significant in the *lifetime* generator, except for $k = n$.

5.4 Impact of deletes

All of the criteria we measured were not significant higher for trees generated with both inserts and deletes than inserts only. This shows that delete operations do not have a significant negative influence on the shape of the tree. The measurements were in some cases significant lower. In all of our measurements, *lifetime* yielded a significant lower value, for most values of k.

6 Conclusion & Discussion

In our experiments we created binary trees with inserts and deletes and compared them with a tree constructed using only inserts. We found that the delete operations do not have a significant negative influence on the shape of the tree. It does however have a significant positive influence in some cases.

This shows that for a random sequence of inserts and deletes, the resulting tree will probably be balanced. However, we tested only three ways to construct a tree (with varying parameters), and this will not model every application properly. Future recommendations would include testing more varying methods of deletions and preferably generate larger differences between trees, to possibly get more interesting results.

7 Reflection

We have been able to fully answer our research question about randomly generated search trees, with plenty of data to support this evidence as well. We managed to complete our research by following the research plan to the very last detail, without running into too much complications. Perhaps the one complication worth mentioning was that the data had to be generated, which was done by leaving the tree generation program running on a laptop for sometimes a long of amount of time (90 minutes). We tried to generate larger trees, but that caused trouble with the garbage collector, because it required using too much resources. Like any other complications which we might have had, this was nothing more than a minor hindrance.

References

[1] Luc Devroye. A note on the height of binary search trees. *Journal of the ACM (JACM)*, 33(3):489–498, 1986.

[2] Jeffrey L Eppinger. An empirical study of insertion and deletion in binary search trees. *Communications of the ACM*, 26(9):663–669, 1983.

[3] Cormen T. H., Leiserson C. E., Rivest R. L., and Stein C. *Introduction to Algorithms*, chapter Data Structures for Disjoint Sets, pages 568–582. The MIT Press, 2009.

[4] Conrado Martínez and Salvador Roura. Randomized binary search trees. *Journal of the ACM (JACM)*, 45(2):288–323, 1998.

[5] Prasad Vinod, Suri Pushpa, and Carsten Maple. Maintaining a random binary search tree dynamically. In *Tenth International Conference on Information Visualisation (IV'06)*, pages 483–488. IEEE, 2006.

Tree height

inserts	iad-1	itd-1	life-1	iad-10	itd-10	life-10	iad-n	itd-n	life-n	iad-n2	itd-n2	life-n2
5,875	5,875	5,750	5,875	6,125	6,000	5,625	5,250	5,375	5,875	6,125	5,250	5,375
7,500	7,125	7,375	7,000	6,125	6,750	6,250	6,500	6,750	7,000	6,875	7,125	7,250
8,500	8,250	7,750	8,250	7,625	8,125	7,125	8,125	8,125	7,875	8,750	8,125	7,625
9,625	9,625	8,875	9,000	9,250	8,625	8,375	8,000	10,250	8,500	9,250	9,000	8,625
10,875	10,375	9,750	9,625	10,750	10,375	9,750	10,125	10,375	9,750	11,000	11,750	9,875
12,000	11,000	10,750	11,625	11,250	11,125	11,125	11,500	11,500	11,250	12,000	12,250	12,125
13,750	13,000	12,500	12,125	12,375	13,000	12,125	13,500	13,750	12,250	12,875	12,625	12,625
13,000	15,250	14,375	14,125	14,375	14,875	12,250	13,250	15,375	12,250	13,625	14,625	14,000
15,250	14,875	15,750	15,875	15,000	15,375	14,875	16,000	15,750	15,250	15,750	15,375	15,625
16,875	16,375	16,000	16,625	16,500	16,500	16,000	17,000	16,875	15,250	18,625	16,500	17,125
18,375	18,000	19,125	18,000	18,250	17,500	16,125	19,625	17,250	16,375	17,750	17,875	17,625
20,125	20,125	19,750	19,750	19,375	20,750	18,000	20,500	20,750	18,500			
20,625	20,750	20,875	21,250	19,875	21,375	20,750	20,375	19,500	19,250			
23,375	22,125	22,375	22,250	21,875	22,750	20,625	22,000	22,500	20,750			
24,125	23,125	24,625	22,375	22,750	22,625	22,000	23,125	23,875	21,375			
24,500	25,125	24,875	24,500	24,625	24,500	24,250						
26,250	26,000	26,125	25,125	25,375	26,375	24,750						

Count	17	17	17	17	17	17	15	15	15	11	11	11
Average difference	-0,136	-0,161	-0,218	-0,176	-0,096	-0,446	-0,173	-0,190	-0,543	-0,135	-0,072	-0,364
Variance	0,082	0,110	0,050	0,113	0,096	0,131	0,146	0,089	0,061	0,111	0,153	0,057
Z-score	-1,947	-2,002	-4,017	-2,156	-1,280	-5,077	-1,756	-2,465	-8,507	-1,349	-0,615	-5,062
t_critical	2,47	2,47	2,47	2,47	2,47	2,47	2,51	2,51	2,51	2,63	2,63	2,63
Significant higher	no	no	no	no	no	no	no	no	no	no	no	no
Significant lower	no	no	yes	no	no	yes	no	no	yes	no	no	yes

Average node depth

inserts	iad-1	itd-1	life-1	iad-10	itd-10	life-10	iad-n	itd-n	life-n	iad-n2	itd-n2	life-n2
3,297	3,203	3,078	3,109	3,063	3,086	2,969	2,914	2,914	3,109	3,047	2,930	2,891
3,969	3,563	4,016	3,802	3,333	3,688	3,427	3,573	3,547	3,750	3,828	3,865	3,740
4,785	4,414	4,277	4,480	4,039	4,371	3,910	4,148	4,348	4,301	4,453	4,262	4,125
5,216	4,922	4,836	4,797	5,044	4,846	4,698	4,628	5,115	4,630	4,792	4,940	4,648
5,955	5,664	5,184	5,551	5,768	5,623	5,258	5,383	5,475	5,260	5,609	5,967	5,383
6,708	6,267	5,996	6,077	6,365	6,115	5,868	6,025	6,164	6,022	6,266	6,415	6,241
7,149	6,889	6,906	6,777	6,589	6,685	6,416	7,044	6,981	6,450	6,895	6,735	6,642
7,326	7,691	7,528	7,508	7,577	7,774	7,194	7,526	7,599	7,012	7,570	7,686	7,510
8,390	8,006	8,397	8,214	8,085	8,205	7,912	8,346	8,032	8,197	8,294	8,355	8,069
8,838	8,730	8,627	8,828	8,889	8,701	8,564	8,870	9,250	8,313	9,521	8,843	8,653
9,433	9,451	9,761	9,028	9,631	9,314	9,090	10,124	9,480	8,813	9,302	10,271	9,165
10,459	10,925	10,486	10,385	10,521	10,764	9,987	10,520	10,652	9,982			
11,055	10,800	11,115	10,837	10,761	11,394	10,948	10,875	10,622	10,314			
11,983	11,736	11,561	11,697	11,355	12,012	11,242	11,794	11,796	10,873			
12,254	12,152	12,089	12,026	12,470	12,158	11,430	12,448	11,996	11,651			
12,644	13,036	13,036	12,923	12,960	12,916	13,236						
13,687	13,393	13,515	13,408	13,704	13,858	13,407						
Count	17	17	17	17	17	17	15	15	15	11	11	11
Average difference	-0,136	-0,161	-0,218	-0,176	-0,096	-0,446	-0,173	-0,190	-0,543	-0,135	-0,072	-0,364
Variance	0,082	0,110	0,050	0,113	0,096	0,131	0,146	0,089	0,061	0,111	0,153	0,057
Z-score	-1,947	-2,002	-4,017	-2,156	-1,280	-5,077	-1,756	-2,465	-8,507	-1,349	-0,615	-5,062
t_critical	2,473	2,473	2,473	2,473	2,473	2,473	2,510	2,510	2,510	2,634	2,634	2,634
Significant higher	no	no	no	no	no	no	no	no	no	no	no	no
Significant lower	no	no	yes	no	no	yes	no	no	yes	no	no	yes

Average leaf depth

inserts	iad-1	itd-1	life-1	iad-10	itd-10	life-10	iad-n	itd-n	life-n	iad-n2	itd-n2	life-n2
4,819	4,897	4,779	4,809	4,765	4,787	4,676	4,625	4,625	4,809	4,750	4,640	4,603
5,587	5,340	5,775	5,570	5,120	5,460	5,210	5,350	5,325	5,520	5,595	5,630	5,510
6,445	6,220	6,087	6,284	5,856	6,178	5,731	5,962	6,155	6,110	6,258	6,072	5,939
6,968	6,781	6,696	6,658	6,901	6,707	6,561	6,492	6,969	6,495	6,653	6,798	6,513
7,744	7,546	7,073	7,435	7,648	7,506	7,146	7,269	7,360	7,148	7,492	7,844	7,269
8,551	8,182	7,914	7,994	8,278	8,031	7,787	7,942	8,080	7,939	8,180	8,329	8,156
9,024	8,820	8,837	8,709	8,522	8,617	8,351	8,974	8,912	8,385	8,826	8,668	8,575
9,240	9,641	9,479	9,459	9,528	9,723	9,146	9,477	9,549	8,965	9,520	9,635	9,460
10,317	9,967	10,357	10,175	10,046	10,165	9,874	10,305	9,993	10,157	10,254	10,315	10,030
10,787	10,702	10,600	10,800	10,860	10,673	10,537	10,842	11,221	10,286	11,492	10,815	10,625
11,392	11,428	11,738	11,007	11,608	11,292	11,068	12,101	11,458	10,791	11,280	12,247	11,143
12,429	12,908	12,470	12,369	12,505	12,747	11,972	12,504	12,636	11,967			
13,032	12,787	13,102	12,825	12,749	13,381	12,936	12,862	12,609	12,302			
13,966	13,727	13,552	13,688	13,346	14,003	13,233	13,785	13,787	12,865			
14,241	14,145	14,083	14,019	14,463	14,151	13,424	14,441	13,989	13,644			
14,635	15,031	15,032	14,919	14,955	14,911	15,231						
15,680	15,389	15,512	15,404	15,700	15,854	15,404						
Count	17	17	17	17	17	17	15	15	15	11	11	11
Average difference	-0,079	-0,104	-0,161	-0,118	-0,039	-0,386	-0,107	-0,125	-0,477	-0,052	0,011	-0,277
Variance	0,072	0,097	0,046	0,094	0,078	0,120	0,118	0,075	0,078	0,094	0,127	0,047
Z-score	-1,215	-1,381	-3,096	-1,588	-0,583	-4,600	-1,211	-1,766	-6,623	-0,563	0,101	-4,230
t_critical	2,473	2,473	2,473	2,473	2,473	2,473	2,510	2,510	2,510	2,634	2,634	2,634
Significant higher	no	no	no	no	no	no	no	no	no	no	no	no
Significant lower	no	no	yes	no	no	yes	no	no	yes	no	no	yes

Leaf depth variance

inserts	iad-1	itd-1	life-1	iad-10	itd-10	life-10	iad-n	itd-n	life-n	iad-n2	itd-n2	life-n2
1,993	2,368	2,191	2,212	2,367	2,251	1,855	1,466	1,816	2,231	2,387	1,514	1,603
3,310	2,458	3,311	2,569	1,510	2,465	1,830	2,030	2,150	2,503	2,641	2,954	2,785
4,005	3,631	2,924	3,852	2,815	3,507	2,122	3,000	3,326	3,170	4,254	3,367	2,798
4,334	3,784	3,367	3,385	4,039	3,333	2,672	2,602	4,538	2,375	3,397	3,736	2,964
5,226	4,540	3,321	3,790	4,475	4,963	3,466	3,802	4,318	3,476	4,842	6,077	4,041
5,451	4,251	3,764	4,498	5,220	4,171	3,888	4,251	4,063	4,260	5,964	6,003	5,286
6,863	5,858	5,128	4,554	4,807	5,477	4,481	5,891	6,198	4,150	4,978	4,893	5,265
5,169	6,879	6,051	6,205	6,382	7,282	4,129	5,226	7,003	3,748	6,025	6,723	6,004
7,330	6,323	6,690	7,653	6,786	6,451	5,632	7,288	6,392	6,562	7,035	7,437	6,526
6,882	6,864	6,474	7,205	7,113	6,578	5,803	7,122	8,390	5,334	10,774	7,082	6,892
8,461	8,267	9,281	6,878	8,607	7,326	6,350	10,621	7,525	5,559	7,532	9,197	7,130
9,084	10,238	9,505	9,704	9,623	10,115	7,153	10,741	10,494	7,337			
10,170	9,434	9,922	9,778	8,702	11,056	8,869	9,394	7,975	7,296			
12,428	9,983	10,204	9,857	9,039	11,472	8,978	10,220	11,076	7,483			
10,844	10,025	10,658	9,846	10,851	9,970	8,155	10,794	10,715	8,192			
10,581	11,151	11,384	11,234	11,350	11,050	11,733						
12,047	11,532	12,212	11,196	12,036	12,785	10,626						
Count	17	17	17	17	17	17	15	15	15	11	11	11
Average difference	-0,388	-0,458	-0,574	-0,497	-0,231	-1,555	-0,473	-0,371	-1,858	0,073	-0,004	-0,703
Variance	0,922	0,998	1,081	1,368	0,979	1,007	1,424	1,360	1,537	2,228	0,904	0,543
Z-score	-1,666	-1,891	-2,277	-1,754	-0,963	-6,390	-1,537	-1,234	-5,806	0,163	-0,013	-3,164
t_critical	2,473	2,473	2,473	2,473	2,473	2,473	2,510	2,510	2,510	2,634	2,634	2,634
Significant higher	no	no	no	no	no	no	no	no	no	no	no	no
Significant lower	no	no	no	no	no	yes	no	no	yes	no	no	yes