Breaking the Binary Search Tree: A Tragicomic Tale of Random Insertions and Deletions

Author:
Alex Herrero

Professors:
Conrado Martínez
Amalia Duch
Salvador Roura

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Acknowledgment

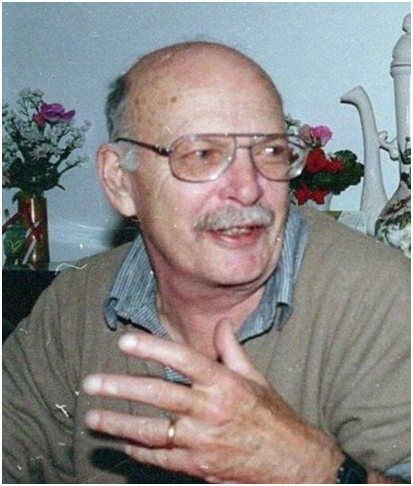
Special thanks to Conrado Martínez. His $\Theta(A(2^{n!}, m \log m))$[1] wisdom and advice have been helpful throughout this project.

[1]Where A is the Ackermann function. Not the inverse!!

Table of Contents

Introduction to BSTs

- He introduced the concept of BST on his paper: Thomas N. Hibbard. "Some Combinatorial Properties of Certain Trees With Applications to Searching and Sorting". In: *J. ACM* 9.1 (Jan. 1962), pp. 13–28. ISSN: 0004-5411. DOI: 10.1145/321105.321108. URL: https://doi.org/10.1145/321105.321108.



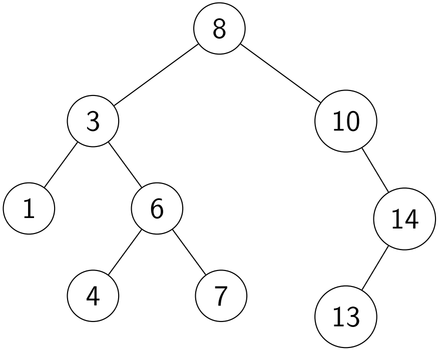
Thomas Hibbard (1929-2016)

- He introduced not only the concept of binary search trees (BSTs), but also the idea of randomness in BSTs.

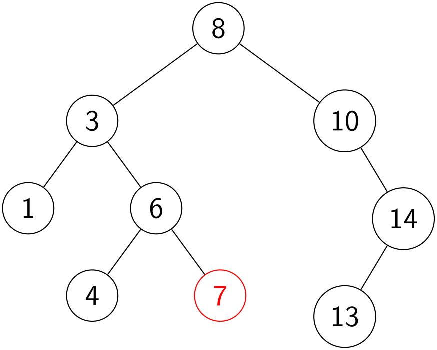
- He introduced not only the concept of binary search trees (BSTs), but also the idea of randomness in BSTs.
- Well-known concepts and algorithms for every computer scientist.

- He introduced not only the concept of binary search trees (BSTs), but also the idea of randomness in BSTs.
- Well-known concepts and algorithms for every computer scientist.
- We will take a look to Hibbard's deletion algorithm.

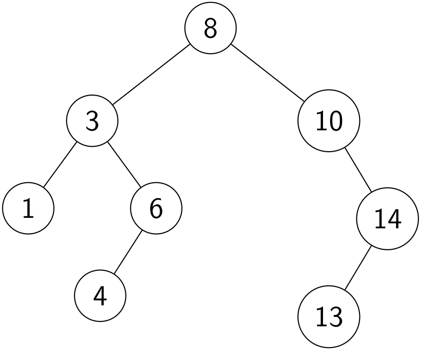
Leaf case



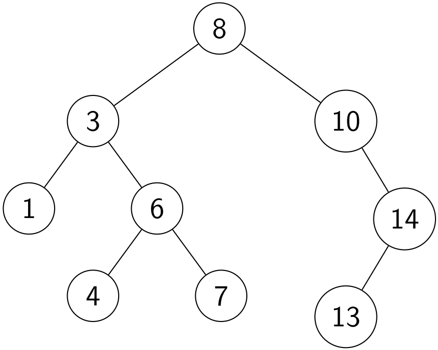
Leaf case



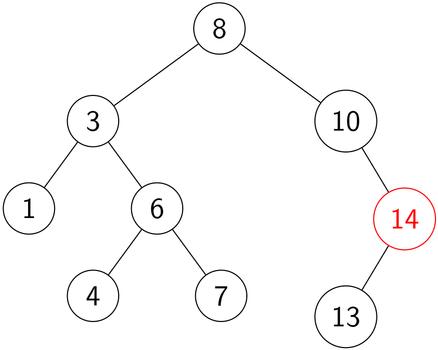
Leaf case



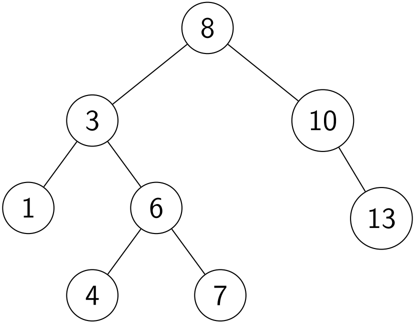
Only one subtree case



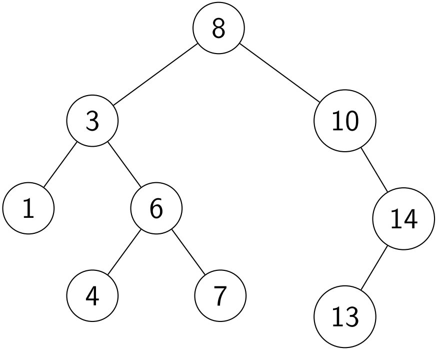
Only one subtree case



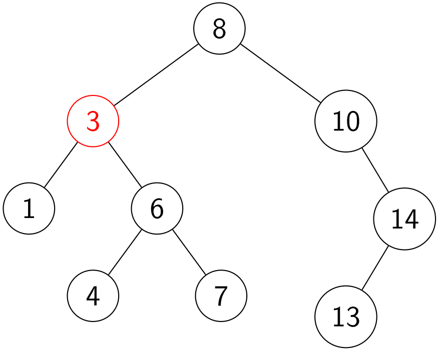
Only one subtree case



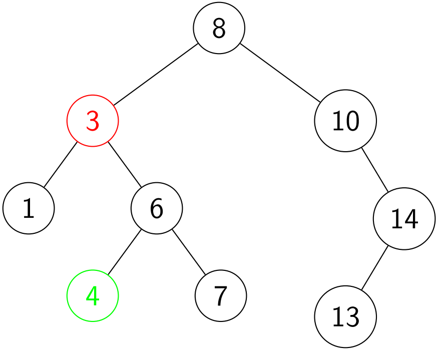
Two subtree case



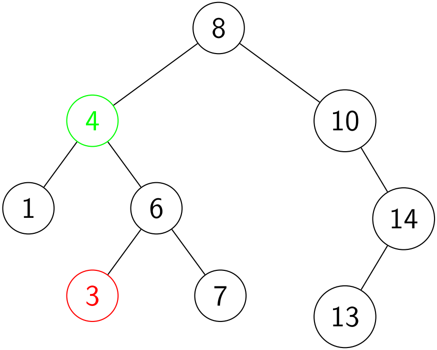
Two subtree case



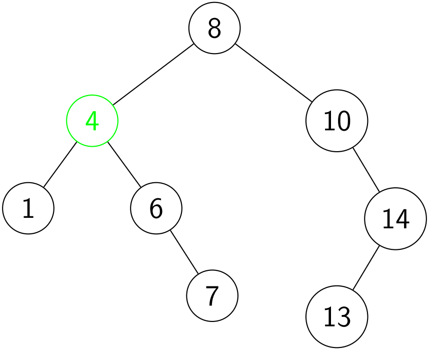
Two subtree case



Two subtree case



Two subtree case



function $\textsc{Delete}(T, x)$
  if $T.val < x$ **then**
    $T.right \leftarrow \textsc{Delete}(T.right, x)$
  else if $T.val > x$ **then**
    $T.left \leftarrow \textsc{Delete}(T.left, x)$
  else
    if $T.right = \text{null}$ **then**
      return $T.left$
    else
      $T.val \leftarrow \textsc{minValue}(T.right)$
      $T.right \leftarrow \textsc{Delete}(T.right, T.val)$
    end if
  end if
  return T
end function

Hibbard's paper was remarkable in that it contained one of the first formal theorems about algorithms:

Hibbard's paper was remarkable in that it contained one of the first formal theorems about algorithms:

Hibbard's Theorem (1962)

If $n+1$ items are inserted into an initially empty binary tree, in random order, and if one of those items (selected at random) is deleted, the probability that the resulting binary tree has a given shape is the same as the probability that this tree shape would be obtained by inserting n items into an initially empty tree, in random order.

Hibbard's paper was remarkable in that it contained one of the first formal theorems about algorithms:

Hibbard's Theorem (1962)

If $n+1$ items are inserted into an initially empty binary tree, in random order, and if one of those items (selected at random) is deleted, the probability that the resulting binary tree has a given shape is the same as the probability that this tree shape would be obtained by inserting n items into an initially empty tree, in random order.

It is reasonable to think that Hibbard's deletion algorithm preserves the randomness of BSTs...

Hibbard's paper was remarkable in that it contained one of the first formal theorems about algorithms:

Hibbard's Theorem (1962)

If $n + 1$ items are inserted into an initially empty binary tree, in random order, and if one of those items (selected at random) is deleted, the probability that the resulting binary tree has a given shape is the same as the probability that this tree shape would be obtained by inserting n items into an initially empty tree, in random order.

It is reasonable to think that Hibbard's deletion algorithm preserves the randomness of BSTs... And it was believed for more than 10 years!

Donald Knuth

The I^*D_r property might seem to be all that one needs to guarantee insensitivity to *any* number of deletions, when they are intermixed with insertions in any order. At least, many people (including the present author when writing the first edition of *The Art of Computer Programming Vol:3*) believed this[a].

[a] Donald Ervin Knuth. "Deletions that preserve randomness". In: *IEEE Transactions on Software Engineering* 5 (1977), pp. 351–359.

- It was believed for more than a decade that Hibbard's algorithm preserved randomness on BSTs

- It was believed for more than a decade that Hibbard's algorithm preserved randomness on BSTs
- 1975: Gary Don Knott. *Deletion in binary storage trees.* Stanford University, 1975

Knott Paradox

Although Hibbard's theorem establishes that n+1 random insertions followed by a random deletion produce a tree whose shape has the distribution of n random insertions, it does not follow that a subsequent random insertion yields a tree whose shape has the distribution of n+1 random insertions

We will follow the explanaition from Arne T Jonassen and Donald E Knuth. "A trivial algorithm whose analysis isn't". In: *Journal of computer and system sciences* 16.3 (1978), pp. 301–322 for a BST of size $n = 3$

All BSTs for $x < y < z$

(x, y, z) $\quad$ (x, z, y) $\quad$ $(y, x, z) = (y, z, x)$ $\quad$ (z, x, y) $\quad$ (z, y, x)

```
x          x                                 z        z
 \          \                               /        /
  y          z           y                 x        y
   \        /           / \                 \      /
    z      y           x   z                 y    x
```

RR $\quad$ RL $\quad$ B $\quad$ LR $\quad$ LL

Permutation	Delete x	Delete y	Delete z
(x, y, z)	R	R	R
(x, z, y)	R	R	R
$(y, z, x) = (y, x, z)$	R	L	L
(z, x, y)	L	L	R
(z, y, x)	L	L	L

Permutation	Delete x	Delete y	Delete z
(x, y, z)	R	R	R
(x, z, y)	R	R	R
$(y, z, x) = (y, x, z)$	R	L	L
(z, x, y)	L	L	R
(z, y, x)	L	L	L

$$\mathbb{P}[L] = \frac{9}{18} = \frac{1}{2}$$

$$\mathbb{P}[R] = \frac{9}{18} = \frac{1}{2}$$

Now another random insertion w comes to the BST. Then we have four possible cases:

Now another random insertion w comes to the BST. Then we have four possible cases:

- $w < x < y < z$

Now another random insertion w comes to the BST. Then we have four possible cases:

- $w < x < y < z$
- $x < w < y < z$

Now another random insertion w comes to the BST. Then we have four possible cases:

- $w < x < y < z$
- $x < w < y < z$
- $x < y < w < z$

Now another random insertion w comes to the BST. Then we have four possible cases:

- $w < x < y < z$
- $x < w < y < z$
- $x < y < w < z$
- $x < y < z < w$

Now another random insertion w comes to the BST. Then we have four possible cases:

- $w < x < y < z$
- $x < w < y < z$
- $x < y < w < z$
- $x < y < z < w$

18 previous cases and 4 possibilities for w give us a total of 72 cases.

Inserting w

$w < x < y < z$

Permutation	Delete x	Delete y	Delete z
(x, y, z)	B	B	B
(x, z, y)	B	B	B
$(y, z, x) = (y, x, z)$	B	LL	LL
(z, x, y)	LL	LL	B
(z, y, x)	LL	LL	LL

Inserting w

$x < w < y < z$

Permutation	Delete x	Delete y	Delete z
(x, y, z)	B	RL	RL
(x, z, y)	B	RL	RL
$(y, z, x) = (y, x, z)$	B	LR	LR
(z, x, y)	LL	LR	RL
(z, y, x)	LL	LR	LR

Inserting w

$x < y < w < z$

Permutation	Delete x	Delete y	Delete z
(x, y, z)	RL	RL	RR
(x, z, y)	RL	RL	RR
$(y, z, x) = (y, x, z)$	RL	LR	B
(z, x, y)	LR	LR	RR
(z, y, x)	LR	LR	B

Inserting w

$x < y < z < w$

Permutation	Delete x	Delete y	Delete z
(x, y, z)	RR	RR	RR
(x, z, y)	RR	RR	RR
$(y, z, x) = (y, x, z)$	RR	B	B
(z, x, y)	B	B	RR
(z, y, x)	B	B	B

Probabilities

$$\mathbb{P}[LL] = \frac{11}{72} \qquad \mathbb{P}[LR] = \frac{13}{72}$$

$$\mathbb{P}[RL] = \frac{11}{72} \qquad \mathbb{P}[RR] = \frac{12}{72}$$

$$\mathbb{P}[B] = \frac{25}{72}$$

Know another random deletion comes. Let us analyze the probability of having and L shape:

Probabilities

$$\mathbb{P}[LL] = \frac{11}{72} \qquad \mathbb{P}[LR] = \frac{13}{72}$$

$$\mathbb{P}[RL] = \frac{11}{72} \qquad \mathbb{P}[RR] = \frac{12}{72}$$

$$\mathbb{P}[B] = \frac{25}{72}$$

Know another random deletion comes. Let us analyze the probability of having and L shape:

Probability L Shape

The probability of having an L shape after a random deletion is: $\mathbb{P}[L] = \mathbb{P}[LL] + \frac{2}{3}\mathbb{P}[LR] + \frac{2}{3}\mathbb{P}[B] = \frac{11}{72} + \frac{2}{3} \cdot \frac{13}{72} + \frac{2}{3} \cdot \frac{25}{72} = \frac{109}{216} > \frac{1}{2}!!$

Probabilities

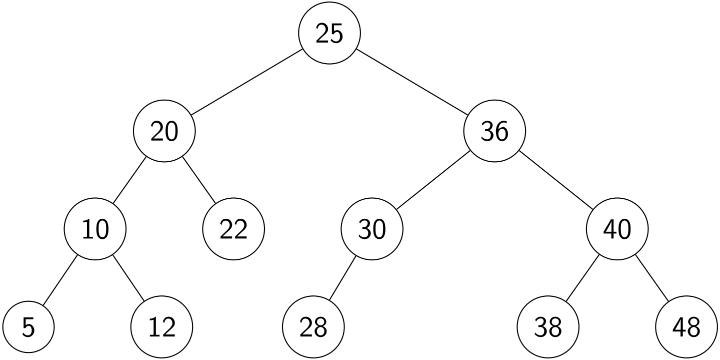
$$\mathbb{P}[LL] = \frac{11}{72} \qquad \mathbb{P}[LR] = \frac{13}{72}$$
$$\mathbb{P}[RL] = \frac{11}{72} \qquad \mathbb{P}[RR] = \frac{12}{72}$$
$$\mathbb{P}[B] = \frac{25}{72}$$

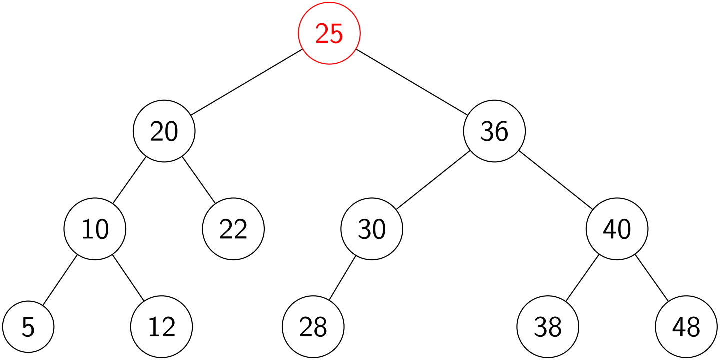
Know another random deletion comes. Let us analyze the probability of having and L shape:

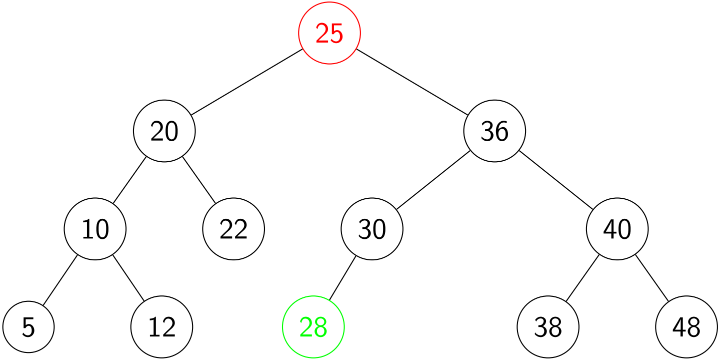
Probability L Shape

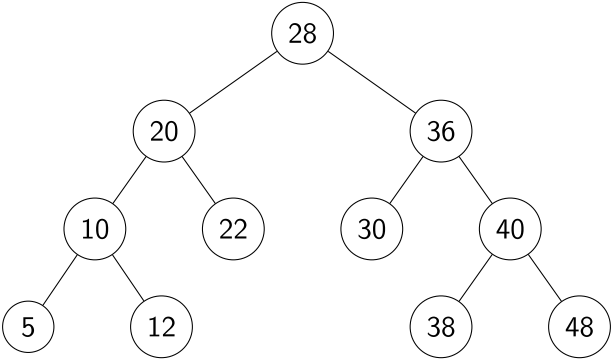
The probability of having an L shape after a random deletion is:
$\mathbb{P}[L] = \mathbb{P}[LL] + \frac{2}{3}\mathbb{P}[LR] + \frac{2}{3}\mathbb{P}[B] = \frac{11}{72} + \frac{2}{3} \cdot \frac{13}{72} + \frac{2}{3} \cdot \frac{25}{72} = \frac{109}{216} > \frac{1}{2}$!!

Actually it makes sense!


25
20
36
10
22
30
40
5
12
28
38
48


25
20
36
10
22
30
40
5
12
28
38
48


25
20
36
10
22
30
40
5
12
28
38
48



Did we change the probability of inserting a random element into one subtree? Somehow, yes...

Donald Knuth

The shape of the tree is random after deletions, but the relative distribution of values in a given tree shape may change, and it turns out that the first random insertion, after a deletion actually destroys the randomness property on shapes. This startling fact, first observed by Gary Knott in 1972, must be seen to be believed[a]

[a] Donald E Knuth. *The Art of Computer Programming: Sorting and Searching, volume 3*. Addison-Wesley Professional, 1998.

Knott was the first to notice that Hibbard's generalization was wrong.

Knott was the first to notice that Hibbard's generalization was wrong.
In his thesis he also gave some empirical data summarizing the results of simulation experiments, where BSTs randomly constructed by $I^n(ID)^m$. Leading to the following conjecture:

Knott's conjecture

Empirical evidence suggests strongly that the path length tends to decrease after repeated deletions and insertions, so the departure from randomness seems to be in the right direction; a theoretical explanation for this behavior is still lacking[a]

[a] Donald E Knuth. *The Art of Computer Programming: Sorting and Searching, volume 3*. Addison-Wesley Professional, 1998.

- Jeffrey L. Eppinger. "An empirical study of insertion and deletion in binary search trees". In: *Commun. ACM* 26.9 (Sept. 1983), pp. 663–669. ISSN: 0001-0782. DOI: 10.1145/358172.358183. URL: https://doi.org/10.1145/358172.358183.
- A landmark in experimental algorithmic literature

Jeffrey Eppinger (1960)

Recall:

- The expected number of comparisons used when searching for an element in a binary tree is proportional to the tree's path length

[2] Donald E Knuth. *The Art of Computer Programming: Sorting and Searching, volume 3*. Addison-Wesley Professional, 1998.

Recall:

- The expected number of comparisons used when searching for an element in a binary tree is proportional to the tree's path length
- $IPL = \sum_{v \in V(T)} d(root, v)$

[2] Donald E Knuth. *The Art of Computer Programming: Sorting and Searching, volume 3*. Addison-Wesley Professional, 1998.

Recall:

- The expected number of comparisons used when searching for an element in a binary tree is proportional to the tree's path length
- $IPL = \sum_{v \in V(T)} d(root, v)$
- For a random tree containing n nodes the expected IPL is denoted as I_n. The expected number of comparisons in a successful search is denoted as C_n
- Knuth[2] give the expected number of comparisons in a successful search, C_n, as $2(1 + \frac{1}{n})H_n - 3$
- By the relation $I_n = n(C_n - 1)$ one obtains the approximation $1.386 n \log n - 2.846 n$

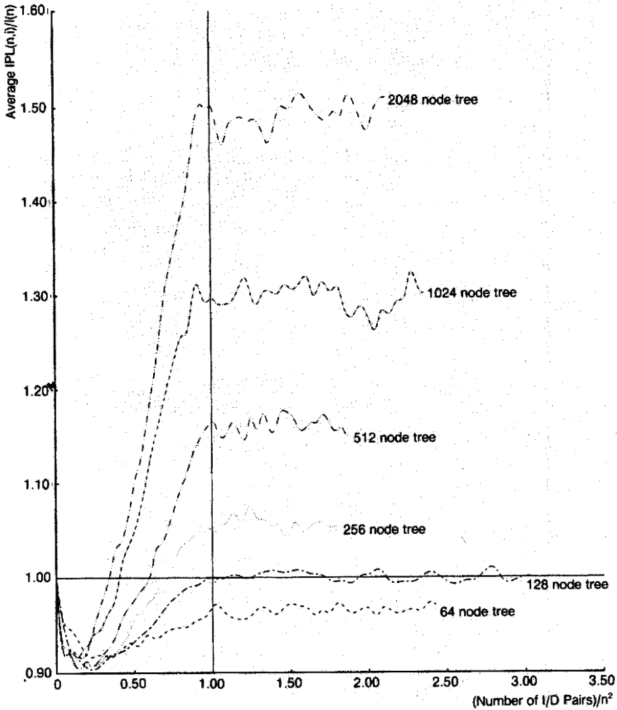
[2] Donald E Knuth. *The Art of Computer Programming: Sorting and Searching, volume 3*. Addison-Wesley Professional, 1998.

Recall:

- The expected number of comparisons used when searching for an element in a binary tree is proportional to the tree's path length
- $IPL = \sum_{v \in V(T)} d(root, v)$
- For a random tree containing n nodes the expected IPL is denoted as I_n. The expected number of comparisons in a successful search is denoted as C_n
- Knuth[2] give the expected number of comparisons in a successful search, C_n, as $2(1 + \frac{1}{n})H_n - 3$
- By the relation $I_n = n(C_n - 1)$ one obtains the approximation $1.386 n \log n - 2.846 n$
- A distribution of trees is said to be "better than random" when the expected IPL is less than I_n.

[2] Donald E Knuth. *The Art of Computer Programming: Sorting and Searching, volume 3*. Addison-Wesley Professional, 1998.

- Large samples of random BSTs of various sizes
- Based on Knott's experiments, extended with more insertions and deletions (a quadratic number in particular)

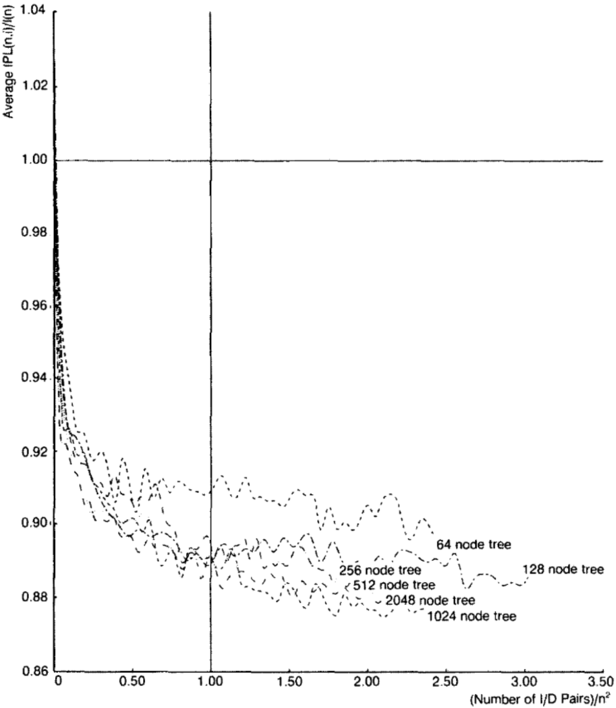


Average IPL(n,i)/I(n)
1.60
1.50
1.40
1.30
1.20
1.10
1.00
0.90
0
0.50
1.00
1.50
2.00
2.50
3.00
3.50
(Number of I/D Pairs)/n^2
2048 node tree
1024 node tree
512 node tree
256 node tree
128 node tree
64 node tree

- Hibbard's algorithm is asymmetric (always choose right subtree)

- Hibbard's algorithm is asymmetric (always choose right subtree)
- A symmetric version of this algorithm was trivially implemented by Eppinger

```
function SYMMETRIC DELETE(T, x)
    if T.val < x then
        T.right ← SYMMETRIC DELETE(T.right, x)
    else if T.val > x then
        T.left ← SYMMETRIC DELETE(T.left, x)
    else
        if T.right = null then
            return T.left
        else
            if FLIPCOIN() = Head then
                T.val ← MINVALUE(T.right)
                T.right ← SYMMETRIC DELETE(T.right, T.val)
            else
                T.val ← MAXVALUE(T.left)
                T.left ← SYMMETRIC DELETE(T.left, T.val)
            end if
        end if
    end if
    return T
end function
```


Average IPL(n,i)/I(n)
1.04
1.02
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0
0.50
1.00
1.50
2.00
2.50
3.00
3.50
(Number of I/D Pairs)/n²
64 node tree
128 node tree
256 node tree
512 node tree
2048 node tree
1024 node tree

Comparison of Deletions

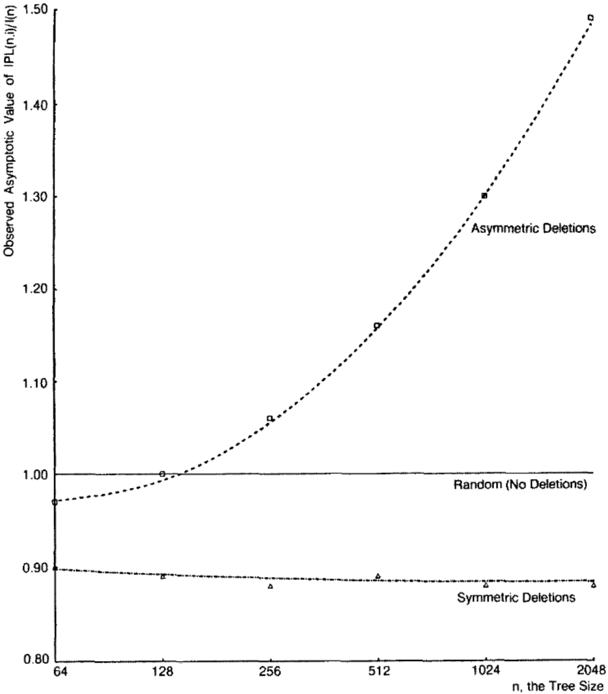
Asymmetric Deletions

n	Samples	$\overline{IPL_n}$	Variance
64	6000	0.97	0.01652
128	6800	1.00	0.01340
256	2300	1.06	0.00985
512	1200	1.16	0.00970
1024	750	1.30	0.01013
2048	5340	1.49	0.00771

Symmetric Deletions

n	Samples	$\overline{IPL_n}$	Variance
64	6000	0.905	0.01654
128	6800	0.890	0.00916
256	2300	0.888	0.00615
512	1200	0.890	0.00347
1024	750	0.881	0.00235
2048	5340	0.883	0.00269

Data obtained after a quadratic number of insertions/deletions.



Observed Asymptotic Value of IPL(n,i)/I(n)
1.50
1.40
1.30
1.20
1.10
1.00
0.90
0.80
Asymmetric Deletions
Random (No Deletions)
Symmetric Deletions
64
128
256
512
1024
2048
n, the Tree Size

Asymmetric Deletion

A least-square multiple regression weighted by the inverse of the variance yields to the following approximation:

Asymmetric Deletion

A least-square multiple regression weighted by the inverse of the variance yields to the following approximation:

$$\lim_{i\to\infty} \frac{\overline{IPL_{n,i}}}{I_n} \approx 0.0202 \log^2 n - 0.241 \log n + 1.69$$

Substituting $I_n \approx 1.386n \log n - 2.846n$ we obtain:

Asymmetric Deletion

A least-square multiple regression weighted by the inverse of the variance yields to the following approximation:

$$\lim_{i\to\infty} \frac{\overline{IPL_{n,i}}}{I_n} \approx 0.0202 \log^2 n - 0.241 \log n + 1.69$$

Substituting $I_n \approx 1.386 n \log n - 2.846 n$ we obtain:

$$\lim_{i\to\infty} \overline{IPL_{n,i}} \approx 0.0280 n \log^3 n - 0.392 n \log^2 n + 3.03 n \log n - 4.81 n$$

Asymmetric Deletion

A least-square multiple regression weighted by the inverse of the variance yields to the following approximation:

$$\lim_{i\to\infty} \frac{\overline{IPL_{n,i}}}{I_n} \approx 0.0202 \log^2 n - 0.241 \log n + 1.69$$

Substituting $I_n \approx 1.386n \log n - 2.846n$ we obtain:

$$\lim_{i\to\infty} \overline{IPL_{n,i}} \approx 0.0280n \log^3 n - 0.392n \log^2 n + 3.03n \log n - 4.81n$$

Expected Internal Path Length

The expected IPL of a tree performing the asymmetric deletion algorithm is, experimentally, $\Theta(n \log^3 n)$

Symmetric Deletion

Symmetric Deletions?

Symmetric Deletion

Symmetric Deletions?

$$\lim_{i \to \infty} \frac{\overline{IPL_{n,i}}}{I_n} \approx 0.88$$

Or that

$$\lim_{i \to \infty} \overline{IPL_{n,i}} \approx 1.22 n \log n - 2.50 n$$

Symmetric Deletion

Symmetric Deletions?

$$\lim_{i\to\infty} \frac{\overline{IPL_{n,i}}}{I_n} \approx 0.88$$

Or that

$$\lim_{i\to\infty} \overline{IPL_{n,i}} \approx 1.22n\log n - 2.50n$$

Expected Internal Path Length

The expected IPL of a tree performing the symmetric deletion algorithm is, experimentally, $\Theta(n\log n)$. Since the perfect tree has IPL $\Omega(n\log n)$ we know that, experimentally, this result is optimum!

Explaining the Behaviour of Binary Search Trees Under Prolonged Updates: A Model and Simulations[3]

Abstract

In this paper we present an extensive study into the long-term behaviour of binary search trees subjected to updates using the usual deletion algorithms taught in introductory textbooks. We develop a model of the behaviour of such trees which **leads us to conjecture that the asymptotic average search path length is** $\Theta(N^{0.5})$. We present results of large simulations which strongly support this conjecture. **However, introducing a simple modification to ensure symmetry in the algorithms, the model predicts no such long-term deterioration. Simulations in fact indicate that asymptotically the average path length of such trees is less than the** $1.386 \ldots \log_2 N$ **average path length of trees generated from random insertion sequences**

[3] J. Culberson and J. I. Munro. "Explaining the behaviour of binary search trees under prolonged updates: a model and simulations". In: *Comput. J.* 32.1 (Feb. 1989), pp. 68–75. ISSN: 0010-4620. DOI: 10.1093/comjnl/32.1.68. URL: https://doi.org/10.1093/comjnl/32.1.68.

Analysis of the standard deletion algorithms in exact fit domain binary search trees[4]

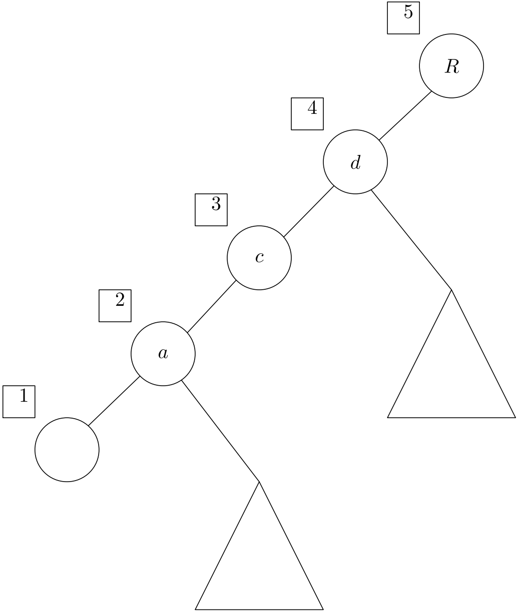
Abstract

It is well known that the expected search time in an N node binary search tree generated by a random sequence of insertions is $O(\log N)$. Little has been published about the asymptotic cost when insertions and deletions are made following the usual algorithms with no attempt to retain balance. **We show that after a sufficient number of updates, each consisting of choosing an element at random, removing it, and reinserting the same value, that the average search cost is $\Theta(N^{\frac{1}{2}})$**

[4] Joseph Culberson and J Ian Munro. "Analysis of the standard deletion algorithms in exact fit domain binary search trees". In: *Algorithmica* 5 (1990), pp. 295–311.

System of tagging a BST as follows:

- The smallest key in the tree receives a new tag whenever it is inserted
- Whenever a key is deleted, all the tags currently attached to it are moved to the next larger key, unless the deleted key is the largest, in which case its tags are discarded


5
R
4
d
3
c
2
a
1

Lemma 2

In an EFD the expected size of the interval containing the jth smallest key at the time it enters the interval is

$$E_j = \frac{2^{2j-2}}{\binom{2j-2}{j-1}} - 1 \approx \sqrt{\pi j}$$

Lemma 2

In an EFD the expected size of the interval containing the jth smallest key at the time it enters the interval is

$$E_j = \frac{2^{2j-2}}{\binom{2j-2}{j-1}} - 1 \approx \sqrt{\pi j}$$

Lemma 3

The expected size of the rth subtree on an EFD after sufficiently many updates is $O(\sqrt{N})$ for all r

Lemma 2

In an EFD the expected size of the interval containing the jth smallest key at the time it enters the interval is

$$E_j = \frac{2^{2j-2}}{\binom{2j-2}{j-1}} - 1 \approx \sqrt{\pi j}$$

Lemma 3

The expected size of the rth subtree on an EFD after sufficiently many updates is $O(\sqrt{N})$ for all r

Lemma 4

The expected number of nodes in the backbone of the EFD tree is $O(\sqrt{N})$ after sufficiently many updates

Theorem 1

The IPL of the EFD tree is $\Theta(N^{3/2})$

Deletions that preserve randomness[5]

Abstract

This paper discusses dynamic properties of data structures under insertions and deletions. It is shown that, in certain circumstances, the result of n random insertions and m random deletions will be equivalent to $n - m$ random insertions, *under various interpretations of the word random and under various constraints on the order of insertions and deletions.*

[5] Donald Ervin Knuth. "Deletions that preserve randomness". In: *IEEE Transactions on Software Engineering* 5 (1977), pp. 351–359.

- Abstract studies on deletion and insertion on any data structure

- Abstract studies on deletion and insertion on any data structure
- Generalization of Hibbard's theorem appears as *one-step deletion insensitivity* abbreviated I^*D_r

- Abstract studies on deletion and insertion on any data structure
- Generalization of Hibbard's theorem appears as *one-step deletion insensitivity* abbreviated I^*D_r
- Deletion insensitivity: I^*D, I^*D^*, I^*DI^*, $(I, D)^*$

- Abstract studies on deletion and insertion on any data structure
- Generalization of Hibbard's theorem appears as *one-step deletion insensitivity* abbreviated I^*D_r
- Deletion insensitivity: I^*D, I^*D^*, I^*DI^*, $(I, D)^*$
- *Under various constraints on the order of insertions*: In particular three different types of insertions
- *Under various constraints on the order of deletions*: In particular six different types of deletions

A trivial algorithm whose analysis isn't[6]

Abstract

Very few theoretical results have been obtained to date about the behavior of information retrieval algorithms under random *deletions*, as well as random insertions. The present paper offers a possible explanation for this dearth of results, by showing that one of the simplest such algorithms already requires a surprisingly intricate analysis. Even when **the data structure never contains more than three items at a time, it is shown that the performance of the standard tree search/insertion/deletion algorithm involves Bessel functions and the solution of bivariate integral equations**. A step-by-step expository analysis of this problem is given, and it is shown how the difficulties arise and can be surmounted.

[6] Arne T Jonassen and Donald E Knuth. "A trivial algorithm whose analysis isn't". In: *Journal of computer and system sciences* 16.3 (1978), pp. 301–322.

- https://doi.org/10.1016/0022-0000(78)90020-X

- `https://doi.org/10.1016/0022-0000(78)90020-X`
- "Random deletions do not always enhance the average path length; the pattern *IIIDIDIDI* leads to a better average search time than does the same patter followed by *DI*

- https://doi.org/10.1016/0022-0000(78)90020-X
- "Random deletions do not always enhance the average path length; the pattern *IIIDIDIDI* leads to a better average search time than does the same patter followed by *DI*
- With Knuth's modification on Hibbard's algorithm (considering a special case as one separate case) they obtained the following:

Last paragraph in Jonassen and Knuth article

(...) Since the values of c_n in the unmodified algorithm are *greater* than $1/3$, for $n \geq 1$, the average internal path length actually turns out to be worse when we use the "improved" algorithm. On the other hand, Knott's empirical data indicate that the modified algorithm does indeed lead to an improvement when the trees are larger.

Do we know a deletion algorithm that preserve randomness?

Do we know a deletion algorithm that preserve randomness?

- Conrado Martínez and Salvador Roura. "Randomized binary search trees". In: *Journal of the ACM (JACM)* 45.2 (1998), pp. 288–323
- Raimund Seidel and Cecilia R Aragon. "Randomized search trees". In: *Algorithmica* 16.4 (1996), pp. 464–497

Do you wanna know a little bit more about this Tragicomic Tale? Check:

Wolfgang Panny. “Deletions in random binary search trees: A story of errors”. In: *Journal of statistical planning and inference* 140.8 (2010), pp. 2335–2345

- Culberson, J. and J. I. Munro. "Explaining the behaviour of binary search trees under prolonged updates: a model and simulations". In: *Comput. J.* 32.1 (Feb. 1989), pp. 68–75. ISSN: 0010-4620. DOI: 10.1093/comjnl/32.1.68. URL: https://doi.org/10.1093/comjnl/32.1.68.
- Culberson, Joseph and J Ian Munro. "Analysis of the standard deletion algorithms in exact fit domain binary search trees". In: *Algorithmica* 5 (1990), pp. 295–311.
- Eppinger, Jeffrey L. "An empirical study of insertion and deletion in binary search trees". In: *Commun. ACM* 26.9 (Sept. 1983), pp. 663–669. ISSN: 0001-0782. DOI: 10.1145/358172.358183. URL: https://doi.org/10.1145/358172.358183.
- Hibbard, Thomas N. "Some Combinatorial Properties of Certain Trees With Applications to Searching and Sorting". In: *J. ACM* 9.1 (Jan. 1962), pp. 13–28. ISSN: 0004-5411. DOI:

10.1145/321105.321108. URL: https://doi.org/10.1145/321105.321108.

Jonassen, Arne T and Donald E Knuth. "A trivial algorithm whose analysis isn't". In: *Journal of computer and system sciences* 16.3 (1978), pp. 301–322.

Knott, Gary Don. *Deletion in binary storage trees.* Stanford University, 1975.

Knuth, Donald E. *The Art of Computer Programming: Sorting and Searching, volume 3.* Addison-Wesley Professional, 1998.

Knuth, Donald Ervin. "Deletions that preserve randomness". In: *IEEE Transactions on Software Engineering* 5 (1977), pp. 351–359.

Martínez, Conrado and Salvador Roura. "Randomized binary search trees". In: *Journal of the ACM (JACM)* 45.2 (1998), pp. 288–323.

Panny, Wolfgang. "Deletions in random binary search trees: A story of errors". In: *Journal of statistical planning and inference* 140.8 (2010), pp. 2335–2345.

Seidel, Raimund and Cecilia R Aragon. "Randomized search trees". In: *Algorithmica* 16.4 (1996), pp. 464–497.

Breaking the Binary Search Tree: A Tragicomic Tale of Random Insertions and Deletions

Author:
Alex Herrero

Professors:
Conrado Martínez
Amalia Duch
Salvador Roura

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona