# YET ANOTHER VARIATION ON BALANCED BINARY TREES

CHRISTOPHE RAFFALLI

*GAATI, University of French Polynesia*

Abstract. We give yet another approach to balanced binary trees. Here the constraints is that the size of a son if majored by twice the size of the other son plus a constant $\delta$. We give a relatively simple implementation that works for $\delta \in \{1, 2, 3\}$.

## 1. Introduction

Binary balanced tree is a very popular way of implementing sets or maps with good performance for all set operations. The AVL approach is to allow a maximum difference of height between the left and right son. This difference is 2 in the case of OCaml standard implementation.

To do this one stores the height of the tree at each node, but it seems more interesting to store the size of the tree which is useful to know in O(1). Many people have considered balanced tree where a maximum ratio of size between the two son is allowed [4, 3, 1]. For instance in Haskell, but they are a bit more complex and error prone (see [2] for the bugs).

We propose the following alternative definition, but we first give some notations:

**Notation 1.** *We denote by $|t|$ the size of a tree. We denote by $E$ the empty tree and by $N(l, v, r)$ a node with left son $l$, right son $r$ and value $v$.*

**Definition 2** ($\delta$-balanced)**.** *We consider usual binary search tree. We fix $\delta \in \{1, 2, 3\}$ and we say that a tree is $\delta$-balanced if for each node $N(l, v, r)$ in the tree we have:*

$$\begin{aligned} |l| &\leq 2|r| + \delta \\ |r| &\leq 2|l| + \delta \end{aligned}$$

One novelty of the paper is our balancing definition, which is affine and does not require a special case for empty son, the case $\delta = 1$ corresponds to $|l| + 1 \leq 2(|r| + 1)$ which was studied in the orginal work [4].

Indeed a constraint $|l| \leq K|r|$ implies $l = E$ when $r = E$. This is why article like [2] use a more complex inequality which is quadratic. To our knowledge a simple affine constraint was not fully studied.

The main constraint in all existing work, is that balancing should only be done using one simple rotation or one double rotation. This allows for very limited set of parameter in

*E-mail address*: christophe@raffalli.eu, christophe.raffalli@upf.pf.

the two coefficients used in the definition of balanced trees and in the decision to perform a simple or a double rotation.

In [2], it is established that $(3, 2)$ is the only valid integer parameter. However, even in the case of more complex operation like union of set, we do more rotations, and if we allow more rotations in general, which in practice will rarely occur, we have more freedom. In the code we propose, these extra rotations are seen because the $B$ function, use for addition and removal calls $J$ in case of rotation! Our implementation cover the parameter $(2, 1)$ when $\delta = 1$, which is not valid if we do not perform extra rotations. Those extra rotation are so rare that they mostly appear when building specific counter examples.

Another novelty is our analysis of the balancing via a very general and simple recursive function and its termination: this function preserves balancing by construction, provided that it terminates. Next, we deduce from the termination analysis an unrolled definition with 3 levels $J, B$ and $N$ very similar to the join, bal and create functions used by OCaml, except that $B$ calls $J$.

Our benchmark in OCaml shows very similar performance between our proposal with $\delta = 2$ and OCaml standard implementation. Globally we are even a bit faster, and allowing for a O(1) size function which is missing in OCaml. We also compared with F. Pottier's Baby library offering tree balances by size and we have overall similar performances.

## 2. Main function $J$

As usual all functions on set or maps can be written from a single function $J(l, v, r)$, that will build a tree with $v$ added provided $x_l < v < x_r$ for any $x_l \in l$ and any $x_r \in r$. The function $J$ we propose below only requires that $l$ and $r$ are $\delta$-balanced.

We first consider a very simple recursive $J$ function in figure 1.

There are two main things to prove: the termination of the tail calls at lines 6 and 13 and that it is legal to build a node at lines 8 and 15. The termination of the inner calls to $J$ is clear because the trees get smaller and therefore they are bounded by the tree height.

We first prove the second.

*Proof.* **Legacy of $N$ at line 8 and 15**

We treat the case of line 8, the other case is symmetrical. By hypothesis and using the tests, we have

$$\begin{array}{rcl} |l| & = & |l_l| + |r_l| + 1 > 2|r| + \delta \\ |r_l| - \delta & < & |l_l| \leq 2|l_r| + 1 \end{array}$$

Thus

$$\begin{array}{rcl} 2|r| + \delta & < & |l_l| + |r_l| + 1 \\ 2|r| + \delta & \leq & |l_l| + |r_l| \\ 2|r| & \leq & |l_l| + |r_l| - \delta \\ & < & 2|l_l| \\ |r| & < & |l_l| \end{array} \tag{2.1}$$

```
1:  function J(l, v, r)
2:      if |l| > 2|r| + δ then
3:          N(l_l, v_l, r_l) ← l
4:          if |l_l| ≤ |r_l| − δ then
5:              N(l_rl, v_rl, r_rl) ← r_l
6:              return J(J(l_l, v_l, l_rl), v_rl, J(r_rl, v, r))
7:          else
8:              return N(l_l, v_l, J(r_l, v, r))
9:      else if |r| > 2|l| + δ then
10:         N(l_r, v_r, r_r) ← r
11:         if |r_r| ≤ |l_r| − δ then
12:             N(l_lr, v_lr, r_lr) ← l_r
13:             return J(J(l, v, l_lr), v_lr, J(r_lr, v_l, r_l))
14:         else
15:             return N(J(l, v, l_r), v_r, r_r)
16:     else
17:         return N(l, v, r)
```

FIGURE 1. Function $J$

Let us denote $r' = J(r_l, v, r)$, we have

$$\begin{array}{rcl} |r'| & = & |r_l| + |r| + 1 \\ |r'| & < & |r_l| + |l_l| + 1 \text{ by 2.1} \\ & \leq & |r_l| + |l_l| \\ & < & 2|l_l| + \delta \\ & \leq & 2|l_l| + \delta \end{array} \tag{2.2}$$

We also have

$$\begin{array}{rcl} |l_l| & \leq & 2|r_l| + \delta \\ & \leq & 2|r'| + \delta \end{array} \tag{2.3}$$

The equations 2.2 and 2.3 are what is needed to be allowed to build the node $N(l_l, v_l, r') = N(l_l, v_l, J(r_l, v, r))$ □

For the termination of $J$ we prove the following stronger result:

**Lemma 3.** *There are at most 3 tail rec calls (counting the original call) at lines 6 and 13, and the third tail rec call can call $N$ to build a node. This means that in place of one function, we could have $J$ calling $B$ (a copy of $J$) for the tail call and $B$ calling $N$ for its tail call. We give $J$ and $B$ below in figure 2.*

*Proof.* Again, we treat only the first case. By hypothesis and using the tests, we have

$$\begin{array}{rcl} |l| & = & |l_l| + |r_l| + 1 > 2|r| + \delta \\ |l_l| + \delta & \leq & |r_l| \leq 2|l_l| + \delta \end{array}$$

$$\begin{array}{rcl} |r_l| & = & |l_{rl}| + |r_{rl}| + 1 \\ |l_{rl}| & \leq & 2|r_{rl}| + \delta \\ |r_{rl}| & \leq & 2|l_{rl}| + \delta \end{array}$$

We give a first inequality for $|l_l|$:

$$\begin{array}{rcl} 2|l_l| & \leq & |l_l| + |r_l| - \delta \\ & \leq & |l| - \delta - 1 \\ |l_l| & \leq & \frac{|l|}{2} - \frac{\delta + 1}{2} \end{array} \tag{2.4}$$

Next an inequality for $|r|$:

$$\begin{array}{rcl} 2|r| + \delta & < & |l_l| + |r_l| + 1 \\ 2|r| + \delta & \leq & |l_l| + |r_l| \\ 2|r| & \leq & |l_l| + |r_l| - \delta \\ 2|r| & \leq & |l_l| + 2|l_l| + \delta - \delta \\ |r| & \leq & \frac{3}{2}|l_l| \end{array} \tag{2.5}$$

Let us define the two sub-trees: $l' = J(l_l, v_l, l_{rl})$ and $r' = J(r_{rl}, v, r)$. We have

$$\begin{array}{rcl} |l'| & = & |l_l| + |l_{rl}| + 1 \\ |r'| & = & |r_{rl}| + |r| + 1 \end{array}$$

We first give the wanted inequality for $|r'|$:

$$\begin{array}{rcl} |r'| & = & |r_{rl}| + |r| + 1 \\ & \leq & \frac{3}{2}|l_l| + |r_{rl}| + 1 \text{ by 2.5} \\ & \leq & \frac{3}{2}|l_l| + 2|l_{rl}| + \delta + 1 \\ & \leq & 2|l_l| + 2|l_{rl}| + \delta + 1 \\ & \leq & 2|l'| - 2 + \delta + 1 \\ & \leq & 2|l'| + \delta - 1 \\ & \leq & 2|l'| + \delta \end{array} \tag{2.6}$$

We give a first inequality for $|l'|$:

$$\begin{array}{rcl} |l'| & = & |l_l| + |l_{rl}| + 1 \\ & \leq & |r_l| - \delta + 2|r_{rl}| + \delta + 1 \\ & \leq & |r_l| + 2|r_{rl}| + 1 \\ & \leq & |l_{rl}| + |r_{rl}| + 1 + 2|r_{rl}| + 1 \\ & \leq & |l_{rl}| + 3|r_{rl}| + 2 \\ & \leq & 5|r_{rl}| + 2 + \delta \\ & \leq & 5|r'| - 5 + 2 + \delta \\ & \leq & 5|r'| - 3 + \delta \end{array} \tag{2.7}$$

This is not sufficient, but in the second tail rec call we now have a new hypothesis: $|l| \leq 5|r| - 3 + \delta$ which from 2.4 gives:

$$\begin{aligned} |l_l| &\leq \frac{5}{2}|r| - \frac{3}{2} + \frac{\delta}{2} - \frac{\delta+1}{2} \\ &\leq \frac{5}{2}|r| - 2 \end{aligned} \tag{2.8}$$

Using it we redo the majoration of $|l'|$ in the case of a second tail rec call:

$$\begin{aligned} |l'| &= |l_l| + |l_{rl}| + 1 \\ &\leq \frac{5}{2}|r| - 2 + 2|r_{rl}| + \delta + 1 \\ &\leq \frac{5}{2}|r| + 2|r_{rl}| + \delta - 1 \\ &\leq \frac{5}{2}|r'| - \frac{5}{2} + \delta - 1 \\ &\leq \frac{5}{2}|r'| - \frac{7}{2} + \delta \end{aligned} \tag{2.9}$$

This is still not sufficient, but in the third tail rec call we now have a stronger hypothesis: $|l| \leq \frac{5}{2}|r| - \frac{7}{2} + \delta$ which from 2.4 gives:

$$\begin{aligned} |l_l| &\leq \frac{5}{4}|r| - \frac{7}{4} + \frac{\delta}{2} - \frac{\delta+1}{2} \\ &\leq \frac{5}{4}|r| - \frac{9}{4} \end{aligned} \tag{2.10}$$

We redo a third and last time the majoration of $|l'|$ corresponding to a third tail rec call:

$$\begin{aligned} |l'| &= |l_l| + |l_{rl}| + 1 \\ &\leq \frac{5}{4}|r| - \frac{9}{4} + 2|r_{rl}| + \delta + 1 \\ &\leq \frac{5}{4}|r| + 2|r_{rl}| + \delta - \frac{5}{4} \\ &\leq 2|r'| - 2 + \delta - \frac{5}{4} \\ &\leq 2|r'| - \frac{13}{4} + 2\delta \\ &\leq 2|r'| + \delta \text{ because } \delta \leq 3 \end{aligned} \tag{2.11}$$

This together with 2.6 allows calling $N$ at the third tail rec call in $J$ (counting the first call). □

We can better analyse when we can call $N$ for the tail rec call in $J$, which gives us a correctness assumption for the function $B$ below:

**Lemma 4.** *We may call $N$ in $J$, in place of the tail rec calls, if $|l| \leq 4|r| + \delta + 3$ when $|l| > 2|r| + \delta$ and if $|r| \leq 4|l| + \delta + 3$ when $|r| > 2|l| + \delta$.*

*Proof.* We do only the first case. From $|l| \leq 4|r| + \delta + 3$, 2.4 gives:

$$\begin{aligned} |l_l| &\leq 2|r| + \frac{\delta+3}{2} - \frac{\delta+1}{2} \\ &\leq 2|r| + 1 \end{aligned} \tag{2.12}$$

We redo the majoration of $|l'|$:

$$\begin{aligned} |l'| &= |l_l| + |l_{rl}| + 1 \\ &\leq 2|r| + 1 + 2|r_{rl}| + \delta + 1 \\ &\leq 2|r| + 2|r_{rl}| + \delta + 2 \\ &\leq 2|r'| - 2 + \delta + 2 \\ &\leq 2|r'| + \delta \end{aligned} \tag{2.13}$$

This together with 2.6 allows calling $N$ in place of the tail rec call in $J$. □

**Lemma 5.** *The two inner calls to $J$: $J(l_l, v_l, l_{rl})$ line 6 and $J(r_{lr}, v_l, r_l)$ line 13 only need two tail rec calls (including the initial call) and we may call $N$ in the second call. This means that we may call $B$ here in the unrolled version to come.*

*Proof.* Again, we only prove the first case as the other is symmetrical. In the call, we have

$$\begin{aligned} |l_l| &\leq |r_l| - \delta \\ &\leq |l_{rl}| + |r_{rl}| + 1 - \delta \\ &\leq 3|l_{rl}| + 1 \\ &\leq 4|l_{rl}| + \delta + 3 \end{aligned} \tag{2.14}$$

We conclude using lemma 4. □

This allow us to unroll $J$ as in figure 2, calling $B$ the first tail recursive call to $J$.
We still have to justify the following:
We may call $B$ and not $J$ at lines 25 and 32.

*Proof.* We only treat the case of line 25. To be allowed to call $B$, we must have by lemma 4 $|r_l| \leq 4|r| + \delta + 3$ and $|r| \leq 4|r| + \delta + 3$.

The first is immediate, as if we called $B$, it means we have $|r_l| < |l| \leq 4|r| + \delta + 3$. For the second inequality, we have

$$\begin{aligned} 2|r| + \delta &< |l| \\ |r| &< \frac{1}{2}(|l| - \delta) \\ &< \frac{1}{2}(|l_l| + |r_l| + 1 - \delta) \\ &< \frac{1}{2}(3|r_l| + \delta + 1 - \delta) \\ &< \frac{1}{2}(3|r_l| + 1) \\ &\leq 4|r_l| + \delta + 3 \end{aligned} \tag{2.16}$$

```
1: function J(l, v, r)
2:     if |l| > 2|r| + δ then
3:         N(l_l, v_l, r_l) ← l
4:         if |l_l| ≤ |r_l| + δ then
5:             N(l_rl, v_rl, r_rl) ← r_l
6:             return B(B(l_l, v_l, l_rl), v_rl, J(r_rl, v, r))
7:         else
8:             return N(l_l, v_l, J(r_l, v, r))
9:     else if |r| > 2|l| + δ then
10:        N(l_r, v_r, r_r) ← r
11:        if |r_r| ≤ |l_r| + δ then
12:            N(l_lr, v_lr, r_lr) ← l_r
13:            return B(J(l, v, l_lr), v_lr, B(r_lr, v_l, r_l))
14:        else
15:            return N(J(l, v, l_r), v_r, r_r)
16:    else
17:        return N(l, v, r)
18: function B(l, v, r)
19:    if |l| > 2|r| + δ then
20:        N(l_l, v_l, r_l) ← l
21:        if |l_l| ≤ |r_l| + δ then
22:            N(l_rl, v_rl, r_rl) ← r_l
23:            return N(B(l_l, v_l, l_rl), v_rl, J(r_rl, v, r))
24:        else
25:            return N(l_l, v_l, B(r_l, v, r))
26:    else if |r| > 2|l| + δ then
27:        N(l_r, v_r, r_r) ← r
28:        if |r_r| ≤ |l_r| + δ then
29:            N(l_lr, v_lr, r_lr) ← l_r
30:            return N(J(l, v, l_lr), v_lr, B(r_lr, v_l, r_l))
31:        else
32:            return N(B(l, v, l_r), v_r, r_r)
33:    else
34:        return N(l, v, r)
```

FIGURE 2. Unrolled version of $J$, using $B$

□

Remark: we can not call $B$ at lines 23 and 30, because continuing the above reasoning gives:

$$\begin{aligned} |r| &< \frac{1}{2}(3|r_l|+1) \\ &< \frac{1}{2}(3(|l_{rl}|+|r_{rl}|+1)+1) \\ &< \frac{1}{2}(3(2|r_{rl}|+\delta+|r_{rl}|+1)+1) \\ &< \frac{1}{2}(9|r_{rl}|+3\delta+4) \\ &< \frac{9}{2}|r_{rl}|+\frac{3}{2}\delta+2 \end{aligned}$$

which is not sufficient to call $B$.

## 3. Further optimisation

A first optimisation is to provide two variations of each function $B$ and $J$ that are only legal when we know which son is larger. This is rather straightforward.

Another point is to call $B$ directly instead of $J$ in the case of small variations in size. This is an obvious corollary of lemma 4.

**Lemma 6.** *We may call $B$ in the case of a function adding or removing one element to a set.*

*Proof.* When there is an increase of size at most one in $l$, we have $|l| \leq 2|r| + \delta + 1$. When there is a decrease of size at most one in $r$, we have $|l| \leq 2|r| + \delta + 2$. In both cases, this allows for calling $B$ by lemma 4. Adding in $r$ or removing in $l$ is symmetrical. □

## 4. Experiments

We give here some tests comparing with OCaml's implementation and F. Pottier's Baby library. We not only give timings, but also information about the length of branches (minimum, maximum and average). We give the gain in percentage, negative meaning we are faster.

There is only one case which is not faster of similar (15% slower): removal of all elements in order after inserting them in order.

We see that the trees are a bit (not much) well balanced with our approach and the speed up is more important with smaller trees but decreases with bigger trees.

First with trees of size $2 \times 10^5$:

```
Adding 200000 consecutive integers in a set:
OCaml's set: : 0.07744s, branches: (min: 17, max: 19, avg: 17.69)
Proposal set: : 0.06695s, branches: (min: 17, max: 19, avg: 17.69)
Baby H Set: : 0.06744s
Baby W Set: : 0.07302s
Variation for ordered add: Ours: -13.55% Baby H: -12.91% Baby W: -5.70%
```

```
Checking mem on all elements of the above:
OCaml's set: : 0.01025s
Proposal set: : 0.01008s
Baby H Set: : 0.01041s
Baby W Set: : 0.01043s
Variation for check mem: Ours: -1.61% Baby H: 1.58% Baby W: 1.79%
Removing all elements of the above:
OCaml's set: : 0.00946s
Proposal set: : 0.01218s
Baby H Set: : 0.01039s
Baby W Set: : 0.01038s
Variation for ordered rm: Ours: 28.79% Baby H: 9.84% Baby W: 9.76%

Adding 200000 random integers in a set:
OCaml's set: : 0.14590s, branches: (min: 13, max: 23, avg: 18.30)
Proposal set: : 0.13377s, branches: (min: 14, max: 23, avg: 18.04)
Baby H Set: : 0.13593s
Baby W Set: : 0.13831s
Variation for random add: Ours: -8.31% Baby H: -6.83% Baby W: -5.20%
Checking mem on all elements of the above:
OCaml's set: : 0.01196s
Proposal set: : 0.01276s
Baby H Set: : 0.01309s
Baby W Set: : 0.01441s
Variation for check mem: Ours: 6.64% Baby H: 9.46% Baby W: 20.44%
Removing all elements of the above:
OCaml's set: : 0.01751s
Proposal set: : 0.01226s
Baby H Set: : 0.01371s
Baby W Set: : 0.01398s
Variation for random rm: Ours: -29.94% Baby H: -21.67% Baby W: -20.16%

Building set of ~200000 elements by random union:
OCaml's set: : 0.12909s, branches: (min: 11, max: 25, avg: 18.73)
Proposal set: : 0.12393s, branches: (min: 13, max: 23, avg: 18.10)
Baby H Set: : 0.10989s
Baby W Set: : 0.11887s
Variation for random union: Ours: -3.99% Baby H: -14.87% Baby W: -7.92%
Checking mem on all elements of the above:
OCaml's set: : 0.01125s
Proposal set: : 0.01089s
Baby H Set: : 0.01141s
Baby W Set: : 0.01156s
```

```
Variation for check mem: Ours: -3.21% Baby H: 1.37% Baby W: 2.73%
Removing all elements of the above:
OCaml's set: : 0.01652s
Proposal set: : 0.01137s
Baby H Set: : 0.01262s
Baby W Set: : 0.01274s
Variation for random union rm: Ours: -31.16% Baby H: -23.60% Baby W: -22.88%
```

First with trees of size $10^6$:

```
Adding 1000000 consecutive integers in a set:
OCaml's set: : 1.33801s, branches: (min: 19, max: 21, avg: 19.95)
Proposal set: : 1.25426s, branches: (min: 19, max: 21, avg: 19.95)
Baby H Set: : 1.26545s
Baby W Set: : 1.38632s
Variation for ordered add: Ours: -6.26% Baby H: -5.42% Baby W: 3.61%
Checking mem on all elements of the above:
OCaml's set: : 0.05620s
Proposal set: : 0.05377s
Baby H Set: : 0.05386s
Baby W Set: : 0.05492s
Variation for check mem: Ours: -4.32% Baby H: -4.17% Baby W: -2.27%
Removing all elements of the above:
OCaml's set: : 0.05345s
Proposal set: : 0.06333s
Baby H Set: : 0.06149s
Baby W Set: : 0.05640s
Variation for ordered rm: Ours: 18.49% Baby H: 15.04% Baby W: 5.51%

Adding 1000000 random integers in a set:
OCaml's set: : 1.92168s, branches: (min: 15, max: 27, avg: 20.68)
Proposal set: : 1.86738s, branches: (min: 16, max: 26, avg: 20.40)
Baby H Set: : 1.91301s
Baby W Set: : 1.87831s
Variation for random add: Ours: -2.83% Baby H: -0.45% Baby W: -2.26%
Checking mem on all elements of the above:
OCaml's set: : 0.07864s
Proposal set: : 0.08043s
Baby H Set: : 0.08280s
Baby W Set: : 0.08337s
Variation for check mem: Ours: 2.27% Baby H: 5.29% Baby W: 6.01%
Removing all elements of the above:
OCaml's set: : 0.09937s
Proposal set: : 0.08205s
```

```
Baby H Set: : 0.08516s
Baby W Set: : 0.08483s
Variation for random rm: Ours: -17.43% Baby H: -14.30% Baby W: -14.63%

Building set of ~1000000 elements by random union:
OCaml's set: : 0.80437s, branches: (min: 14, max: 28, avg: 21.00)
Proposal set: : 0.77576s, branches: (min: 16, max: 25, avg: 20.42)
Baby H Set: : 0.76688s
Baby W Set: : 0.71225s
Variation for random union: Ours: -3.56% Baby H: -4.66% Baby W: -11.45%
Checking mem on all elements of the above:
OCaml's set: : 0.06261s
Proposal set: : 0.06475s
Baby H Set: : 0.05949s
Baby W Set: : 0.06009s
Variation for check mem: Ours: 3.43% Baby H: -4.98% Baby W: -4.02%
Removing all elements of the above:
OCaml's set: : 0.09015s
Proposal set: : 0.07256s
Baby H Set: : 0.07097s
Baby W Set: : 0.06945s
Variation for random union rm: Ours: -19.51% Baby H: -21.27% Baby W: -22.96%
```

## References


[1] SR Adams. An efficient functional implementatation of sets. *Report CSTR*, pages 92–10, 1992.

[2] Yoichi Hirai and Kazuhiko Yamamoto. Balancing weight-balanced trees. *J. Funct. Program.*, 21:287–307, 05 2011.

[3] J. Nievergelt and E. M. Reingold. Binary search trees of bounded balance. *SIAM Journal on Computing*, 2(1):33–43, 1973.

[4] Jürg Nievergelt and Edward Reingold. Binary search trees of bounded balance. volume 2, pages 137–142, 01 1972.