Grandchildren-Weight-Balanced Binary Search Trees

Vincent Jugé
LIGM, Univ Gustave Eiffel & CNRS, Marne-la-Vallée, France
IRIF, Université Paris Cité & CNRS, France

Abstract

We revisit weight-balanced trees, also known as trees of bounded balance. Invented by Nievergelt and Reingold in 1972, these trees are obtained by assigning a weight to each node and requesting that the weight of each node should be quite larger than the weights of its children, the precise meaning of "quite larger" depending on a real-valued parameter γ. Blum and Mehlhorn then showed how to maintain them in a recursive (bottom-up) fashion when $2/11 \leqslant \gamma \leqslant 1-1/\sqrt{2}$, their algorithm requiring only an amortised constant number of tree rebalancing operations per update (insertion or deletion). Later, in 1993, Lai and Wood proposed a top-down procedure for updating these trees when $2/11 \leqslant \gamma \leqslant 1/4$.

Our contribution is two-fold. First, we strengthen the requirements of Nievergelt and Reingold, by also requesting that each node should have a substantially larger weight than its grandchildren, thereby obtaining what we call grandchildren-balanced trees. Grandchildren-balanced trees are not harder to maintain than weight-balanced trees, but enjoy a smaller node depth, both in the worst case (with a 6 % decrease) and on average (with a 1.6 % decrease). In particular, unlike standard weight-balanced trees, all grandchildren-balanced trees with n nodes are of height less than $2\log_2(n)$.

Second, we adapt the algorithm of Lai and Wood to all weight-balanced trees, i.e., to all parameter values γ such that $2/11 \leqslant \gamma \leqslant 1-1/\sqrt{2}$. More precisely, we adapt it to all grandchildren-balanced trees for which $1/4 < \gamma \leqslant 1-1/\sqrt{2}$. Finally, we show that, except in limit cases (where, for instance, $\gamma = 1-1/\sqrt{2}$), all these algorithms result in making a constant amortised number of tree rebalancing operations per tree update.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases Data structures, Balanced binary trees

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.40

Related Version *Full Version*: https://arxiv.org/abs/2410.08825 [12]

1 Introduction

Among the most fundamental data structures are search trees. Such trees are aimed at representing sets of pairwise comparable elements of size n while allowing basic operations such as membership test, insertion and deletion, which typically require a time linear in the depth of the tree. That is why various data structures such as height-balanced (AVL) trees [1], weight-balanced trees [17], B-trees [4], red-black trees [9], splay trees [19], relaxed k-trees [8] or weak AVL trees [10] were invented since the 1960s. All of them provide worst-case or amortized $\mathcal{O}(\log(n))$ complexities for membership test, insertion and deletion, and may use local modifications of the tree shape called *rotations*.

The most desirable features of such trees include their height, their internal and external path lengths, and the amortised number of rotations triggered by an insertion or deletion. Such statistics are presented in Table 1 below for families of binary trees.

Note that, although relaxed k-trees can be made arbitrarily short (given that each tree should be of height at least $\log_2(n)$), achieving these excellent guarantees on the height of the trees requires performing significantly more rotations, and thus considering other data structures may still be relevant. This is, for instance, why Haeupler, Tarjan and Sen proposed

Table 1 Asymptotic approximate worst-case height, internal/external path length and amortised number of rotations per update in binary trees with n nodes. Next to each worst-case bound are indicated references where this bound is proved.

Tree family	Worst-case		
	height	path length	am. rotations/update
AVL	$1.4404\log_2(n)$ [1]	$1.4404n\log_2(n)$ [13]	$\Theta(\log(n))$ [2]
Weight-balanced	$2\log_2(n)$ [17]	$1.1462n\log_2(n)$ [17]	$\Theta(1)$ [5]
Red-black	$2\log_2(n)$ [9]	$2n\log_2(n)$ [7]	$\Theta(1)$ [9]
Splay	n [19]	$n^2/2$ [19]	$\Theta(1)$ [19]
Relaxed k-	$(1+\varepsilon)\log_2(n)$ [8]	$(1+\varepsilon)n\log_2(n)$ [8]	$\mathcal{O}(1/\varepsilon)$ [8]
Weak AVL	$2\log_2(n)$ [9]	$2n\log_2(n)$ [7]	$\Theta(1)$ [10]
Grandchildren-balanced	$1.8798\log_2(n)$	$1.1271n\log_2(n)$	$\Theta(1)$

weak AVL trees in 2009: these are a variant of AVL trees, isomorphic to red-black trees, whose height can be worse than that of AVL trees (but never worse than that of red-black trees), but which require only a constant number of rotations per update, whereas standard AVL trees may require up to $\log(n)$ rotations per update.

In 1972, Nievergelt and Reingold invented *weight-balanced trees*, or *trees of bounded balance* [17]. This family of trees depends on a real parameter γ, and is denoted by $\mathsf{BB}[\gamma]$. It is based on the notion of *weight* of a node n, which is the integer $|n|$ defined as one plus the number of descendants of a node n; alternatively, $|n|$ is the number of empty sub-trees descending from n. Although they might be less efficient than other families of balanced trees in general, they are a reference data structure for order-statistic trees: they are relevant in contexts where we need to store the rank of elements in a dynamic set [6] (i.e., finding efficiently the k^{th} smallest element, or counting elements smaller than a given bound). They are also widely used in libraries of mainstream languages, e.g., Haskell set or map implementations [15, 16].

The family $\mathsf{BB}[\gamma]$ consists of those binary search trees in which, for all nodes u and v, we have $|u| \geqslant \gamma|v|$ whenever u is a child of v. Nievergelt and Reingold also gave an algorithm for dynamically maintaining $\mathsf{BB}[\gamma]$-trees of size n whenever $\gamma \leqslant \sqrt{2}-1$, which required only $\mathcal{O}(\log(n))$ element comparisons and pointer moves per modification.

Blum and Mehlhorn then found that this algorithm worked only when $2/11 \leqslant \gamma \leqslant \sqrt{2}-1$. They also proved that, in that case, it required only $\mathcal{O}(1)$ amortised pointer moves per modification [5]. Lai and Wood subsequently proposed an algorithm for maintaining such trees through a single top-down pass per modification whenever $2/11 \leqslant \gamma \leqslant 1/4$, or whenever updates were guaranteed to be non-redundant [14]. This limitation on the domain of γ when tackling possibly redundant updates is somewhat unfortunate, given that those values of γ for which $\mathsf{BB}[\gamma]$ have the best upper bounds on their height and path length are those for which γ approaches $\sqrt{2}-1$.

Top-down maintenance algorithms have two main advantages over bottom-up algorithms. First, bottom-up algorithms require maintaining a link from each node to its parent or storing the list of nodes we visited along a given branch on a stack, which makes them typically slower than top-down algorithms [3]. Second, they are also a strong bottleneck in concurrent or parallel settings: going down to a leaf and then back to the root must be an atomic operation (or requires maintaining children-to-parent links), since it might finish by changing the root [9, 10].

Thus, our goal here is two-fold. First, by making small changes to a long-known data structure, we improve the guarantees it offers in terms of tree height. Second, we aim at circumventing the limitations of the algorithm from Lai and Wood, by proposing a top-down updating algorithm that will be valid in all cases, and which we also extend to our enhanced data structure.

Contributions

We propose a new variant of weight-bounded trees, which we call *grandchildren-balanced trees*. This family of trees depends on two real parameters α and β, and is denoted by $\mathsf{GCB}[\alpha,\beta]$. It consists of those binary search trees in which, for all nodes u and v, we have $|u| \leqslant \alpha|v|$ whenever u is a child of v, and $|u| \leqslant \beta|v|$ whenever u is a grandchild of v. Grandchildren-balanced trees generalise weight-bounded trees, because $\mathsf{GCB}[\alpha,\alpha^2] = \mathsf{BB}[1-\alpha]$.

We prove that, for well-chosen values of α and β, the height and internal and external path lengths of grandchildren-balanced trees are smaller than those of weight-bounded trees; in particular, we break the $2\log_2(n)$ lower bound on the worst-case height of weight-balanced trees. These results are achieved by using Algorithm 11.

We also extend the algorithm of Lai and Wood, and propose an algorithm for maintaining grandchildren-balanced trees in a single top-down pass per modification, which requires only $\mathcal{O}(1)$ amortised pointer moves per modification; this is Algorithm 16. Our algorithm is valid for all relevant values of α and β, including those for which $\mathsf{GCB}[\alpha,\beta]$-trees enjoy the best upper bounds on their height and path length.

Most proofs are omitted in the body of this article. Those of Sections 2 and 3 can be found in the appendix; the other ones can be found in the complete version of this article [12].

2 Grandchildren-balanced trees

In this section, we describe a new family of binary search trees, called *grandchildren-balanced trees*, which depends on two real parameters α and β. We also describe the domain in which the pair (α,β) is meaningful and, for such pairs, we provide upper bounds on the height and internal and external path lengths of grandchildren-balanced trees.

Let us first recall the definition given in Section 1.

▶ **Definition 1.** *The* weight *of a binary tree $\mathcal{T}$, denoted by $|\mathcal{T}|$, is defined as the number of empty sub-trees of $\mathcal{T}$; alternatively, $|\mathcal{T}|-1$ is the number of nodes of $\mathcal{T}$. The* weight *of a node n of $\mathcal{T}$, also denoted by $|n|$, is defined as the weight of the sub-tree of $\mathcal{T}$ rooted at n.*

In what follows, it may be convenient to see each empty tree (of weight 1) as if it had two empty children of weight 1/2 each, themselves having two empty children of weight 1/4 each, and so on. In other words, we may see each empty tree as an infinite complete binary tree whose nodes of depth d have weight 2^{-d}.

Let n be a node of $\mathcal{T}$, let n_1 and n_2 be its children, and let n_{11}, n_{12}, n_{21} and n_{22} be its grandchildren. The child-balance *and the* grandchild-balance *(also called* C*-balance and* GC*-balance) of n in $\mathcal{T}$ are defined as the real numbers*

$$\mathsf{bal}_{\mathsf{C}}(\mathcal{T},n) = \frac{\max\{|n_1|,|n_2|\}}{|n|} \text{ and } \mathsf{bal}_{\mathsf{GC}}(\mathcal{T},n) = \frac{\max\{|n_{11}|,|n_{12}|,|n_{21}|,|n_{22}|\}}{|n|}.$$

When the context is clear, we may omit referring to $\mathcal{T}$, thereby simply writing $\mathsf{bal}_{\mathsf{C}}(n)$ and $\mathsf{bal}_{\mathsf{GC}}(n)$; alternatively, when n is the root of $\mathcal{T}$, its C*-balance and the* GC*-balance may also be directly denoted by $\mathsf{bal}_{\mathsf{C}}(\mathcal{T})$ and $\mathsf{bal}_{\mathsf{GC}}(\mathcal{T})$.*

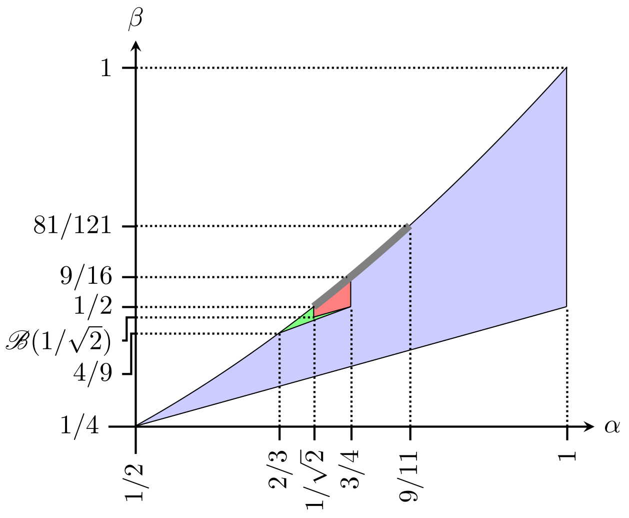

Figure 1 Two-dimensional domains $\mathcal{D}' \subseteq \mathcal{D}'' \subseteq \mathcal{D}$, and one-dimensional (thick, gray) curve $\mathcal{C}$. Whereas BB$[1-\alpha]$-trees correspond to choosing (α, β) on the curve $\mathcal{C}$, our GCB$[\alpha, \beta]$-trees can be created on the entire domain $\mathcal{D}'$.

Given real numbers α and β, we say that a node n of $\mathcal{T}$ is α-child balanced *(or α-C-balanced) when* bal$_{\mathsf{C}}(n) \leqslant \alpha$*;* β-grandchild-balanced *(or β-GC-balanced) when* bal$_{\mathsf{GC}}(n) \leqslant \beta$*; and* (α, β)-balanced *when n is both α-C-balanced and β-GC-balanced.*

Finally, we say that $\mathcal{T}$ is a GCB$_{x,y}[\alpha, \beta]$*-tree when its root is (x, y)-balanced and its other nodes are (α, β)-balanced. For the sake of concision, we simply say that $\mathcal{T}$ is a* GCB$_x[\alpha]$*-tree, a* GCB$[\alpha, \beta]$*-tree or a* GCB$[\alpha]$*-tree when $(y, \beta) = (1, 1)$, $(x, y) = (\alpha, \beta)$ or $(x, y, \beta) = (\alpha, 1, 1)$, respectively.*

Below, we will be mostly interested in the family of GCB$[\alpha, \beta]$-trees. Although it generalises weight-bounded trees invented by Nievergelt and Reingold, we had to shift away from their notation. Indeed, weight-bounded trees were parametrised with a real number γ, by requesting that $|u| \geqslant \gamma|v|$ whenever u is a child of v; however, being β-GC-balanced cannot be expressed by using constraints such as "$|u| \geqslant \gamma|v|$ whenever u is a grandchild of v". In particular, in [5, 17], the *root-balance* of the node n is simply defined as the real number $\rho(n) = |n_1|/|n|$; here, we set bal$_{\mathsf{C}}(n) = \max\{\rho(n), 1 - \rho(n)\}$.

The family of GCB$[\alpha, \beta]$-trees coincides with
- both families of GCB$[\alpha]$-trees and BB$[1-\alpha]$-trees when $\alpha^2 \leqslant \beta$;
- the empty set when $\alpha < 1/2$;
- the family of GCB$[1, \beta]$-trees when $1 \leqslant \alpha$;
- the family of GCB$[2\beta, \beta]$-trees when $\beta < \alpha/2$;
- the family of all binary trees when $\alpha = \beta = 1$.

Thus, we assume below that the pair (α, β) belongs to the domain

$$\mathcal{D} = \{(\alpha, \beta) \colon \alpha/2 \leqslant \beta \leqslant \alpha^2 \text{ and } 1/2 \leqslant \alpha \leqslant 1\} \setminus \{(1, 1)\},$$

represented in Figure 1 along with two other domains $\mathcal{D}'$ and $\mathcal{D}''$ at the end of Section 2.

We first obtain the following upper bound on the height of GCB$[\alpha, \beta]$-trees.

▶ **Theorem 2.** *When $(\alpha, \beta) \in \mathcal{D}$, each* GCB$[\alpha, \beta]$*-tree with $\mathbf{N}$ nodes is of height*

$$h \leqslant -2 \log_\beta(\mathbf{N} + 1).$$

Proof. Let $\mathcal{T}$ be a $\mathsf{GCB}[\alpha,\beta]$-tree with $\mathbf{N}$ nodes. Let h be its height, and let n be a node of $\mathcal{T}$ at depth h. An induction on k proves that $|n^{(k)}| \geqslant 2\beta^{-k/2}\alpha$ whenever $0 \leqslant k \leqslant h$, where $n^{(k)}$ denotes the kth ancestor of n, and is defined by n if $k = 0$, or as the parent of $n^{(k-1)}$ if $k \geqslant 1$. It follows that $\mathbf{N}+1 \geqslant |n^{(h)}| \geqslant 2\beta^{-h/2}\alpha \geqslant \beta^{-h/2}$, i.e., that $h \leqslant -2\log_\beta(\mathbf{N}+1)$. ◀

We also focus on the internal and external path lengths of $\mathsf{GCB}[\alpha,\beta]$-trees. The *internal* path length of a tree $\mathcal{T}$ is the sum of the lengths of the paths from the root of $\mathcal{T}$ to the nodes of $\mathcal{T}$; the *external* path length of $\mathcal{T}$ is the sum of the lengths of the paths from the root of $\mathcal{T}$ to the empty sub-trees of $\mathcal{T}$. These lengths are closely related to the average number of queries required to check membership in the set of labels of $\mathcal{T}$.

Indeed, let $\mathcal{T}$ be a binary search tree whose $\mathbf{N}$ nodes are labelled by elements of a linearly ordered set $\mathcal{E}$, and let λ_{int} and λ_{ext} be its internal and external path lengths. The labels of the nodes of $\mathcal{T}$ form a linearly ordered set $\mathcal{S} \subseteq \mathcal{E}$ of size $\mathbf{N}$, which splits $\mathcal{E}$ into $\mathbf{N}+1$ intervals. Then, the average number of queries used to find an element x is:

- $\lambda_{\mathsf{int}}/\mathbf{N}$ when x is chosen uniformly at random in $\mathcal{S}$;
- $\lambda_{\mathsf{ext}}/(\mathbf{N}+1)$ when x is chosen in $\mathcal{E} \setminus \mathcal{S}$, and the interval of $\mathcal{E} \setminus \mathcal{S}$ to which x belongs is chosen uniformly at random.

By construction, each node n increases the lengths λ_{int} and λ_{ext} by $|n| - 2$ and $|n|$, respectively. This means that λ_{ext} is the sum of the weights of the tree nodes, and that $\lambda_{\mathsf{int}} = \lambda_{\mathsf{ext}} - 2\mathbf{N}$.

By adapting the proof from [18], we obtain the following upper bound on the external path lengths of $\mathsf{GCB}[\alpha,\beta]$-trees.

▶ **Theorem 3.** *When $(\alpha,\beta) \in \mathcal{D}$, each $\mathsf{GCB}[\alpha,\beta]$-tree with $\mathbf{N}$ nodes is of external path length*

$$\lambda \leqslant (\mathbf{N}+1)\log_2(\mathbf{N}+1)/\Delta,$$

where $\Delta = (\mathsf{H}_2(\alpha) + \alpha\mathsf{H}_2(\beta/\alpha))/(1+\alpha)$, and $\mathsf{H}_2(x) = -x\log_2(x) - (1-x)\log_2(1-x)$ is Shannon's binary entropy.

In what follows, we will investigate more closely the family of $\mathsf{GCB}[\alpha,\beta]$-trees when (α,β) belongs to the more restricted domain

$$\mathcal{D}' = \{(\alpha,\beta)\colon 1/\sqrt{2} \leqslant \alpha < 3/4 \text{ and } \mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2\}, \text{ where } \mathscr{B}(\alpha) = \frac{\sqrt{1+4\alpha}-1}{2}.$$

In the limit cases where $\alpha = 1/\sqrt{2} \approx 0.7071$ and $\beta = \mathscr{B}(\alpha) \approx 0.4783$, Theorems 2 and 3 translate into the inequalities

$$h \leqslant 1.8798\log_2(\mathbf{N}+1) \text{ and } \lambda \leqslant 1.1271(\mathbf{N}+1)\log_2(\mathbf{N}+1),$$

thereby improving the inequalities

$$h \leqslant 2\log_2(\mathbf{N}+1) \text{ and } \lambda \leqslant 1.1462(\mathbf{N}+1)\log_2(\mathbf{N}+1)$$

obtained in the best case (i.e., in the limit case where $\alpha = 1/\sqrt{2}$ and $\beta = 1/2$) for $\mathsf{BB}[1-\alpha]$-trees.

On the domain $\mathcal{D}'$, or even on the larger domain $\mathcal{D}'' = \{(\alpha,\beta)\colon 2/3 \leqslant \beta/\alpha \leqslant \alpha < 3/4\}$, the upper bounds presented in Theorem 2 and 3 are (unsurprisingly) quite tight.

▶ **Proposition 4.** *Let (α,β) be a pair belonging to $\mathcal{D}''$, and let $\mathbf{N} \geqslant 0$ be an integer. There exists a $\mathsf{GCB}[\alpha,\beta]$-tree with $\mathbf{N}$ nodes, of height $h \geqslant -2\log_2(\mathbf{N}+1)/\log_2(\beta) - 7$ and external path length $\lambda \geqslant (\mathbf{N}+1)\log_2(\mathbf{N}+1)/\Delta - 4\mathbf{N}$.*

3 Bottom-up algorithm

In this section, we propose a simple algorithm for rebalancing a $\mathsf{GCB}[\alpha, \beta]$ tree $\mathcal{T}$ after having added a leaf to $\mathcal{T}$ or deleted a node from $\mathcal{T}$, when the pair (α, β) belongs to the domain $\mathcal{D}'$.

Rebalancing $\mathsf{GCB}[\alpha, \beta]$ trees might also be possible when $3/4 \leqslant \alpha \leqslant 1$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$ by using similar ideas, but doing so may require more complicated algorithms, in particular for dealing with base cases, which is not necessarily worth the effort since such parameters provide us with trees of larger height and path length.

In case of a deletion, as illustrated in the three cases of Figure 2, we can safely assume that the node to be deleted is a leaf s, by using Hibbard's deletion technique [11]. Indeed, if the targeted node a has just one child, since a is α-C-balanced and $\alpha < 3/4$, we know that $|a| \leqslant 3$, which means that this child is a leaf; thus, we can just focus on deleting that child and, *a posteriori*, replace the key of a with the key of this just-deleted child. Similarly, if the targeted node a has two children, we can first identify the successor of a as the deepest node b of the left branch stemming from the right child of a; b has no left child, which means that we can just focus on deleting it and, *a posteriori*, replace the key of a with the key of this just-deleted node b.

In both cases, let r be the root of $\mathcal{T}$, and let s be the node that must be inserted or deleted. The weights of the nodes from r to s need to be incremented by 1 in case of a non-redundant insertion (since s just replaced an empty tree, we can pretend that its weight was 1 before it was inserted), or decremented by 1 in case of a non-redundant deletion (since s was a leaf and is replaced by an empty tree, we can pretend that its new weight is 1). However, if the update is redundant, i.e., if we tried to insert a key that was already present in $\mathcal{T}$, or to delete an absent key, these weights will not be changed.

In practice, each tree node will contain two fields for storing explicitly its label and its weight. Moreover, in the description below, we simply say " if s lies in a sub-tree $\mathcal{T}'$ " as a place-holder for " determine, based on the key x that you want to insert or delete and on the key y stored in the root of the current tree, which sub-tree $\mathcal{T}'$ should contain s ".

Our algorithm is based on two algorithmic building blocks, called C-balancing (Algorithm 5) and GC-balancing (Algorithm 6). They consist in locally performing rotations in order to make a given node better balanced when needed, without damaging the balance of its parent and children too much.

In a nutshell, the idea of our bottom-up updating algorithm, which will be made more precise in Algorithm 11, is as follows:

1. We recursively update the sub-tree that needs to be updated.
2. Using the C-balancing algorithm ensures our root and its children are suitably child-balanced.
3. Using the GC-balancing algorithm then ensures our root is suitably child- and grandchild-balanced.

This is the same general idea as the original algorithm from Nievergelt and Reingold [5, 17], adapted to take grandchild balances into account: one should first make our tree child-balanced, and only then can one make it grandchild-balanced too.

One difficulty is that, when we perform a rotation to make our root child- or grandchild-balanced, this rotation should be worth its cost: we expect that, for the next few updates (a quantity that can be made precise, and will be studied in Section 5), no update will be required at all. This is what will make the amortised number of rotations per update a constant.

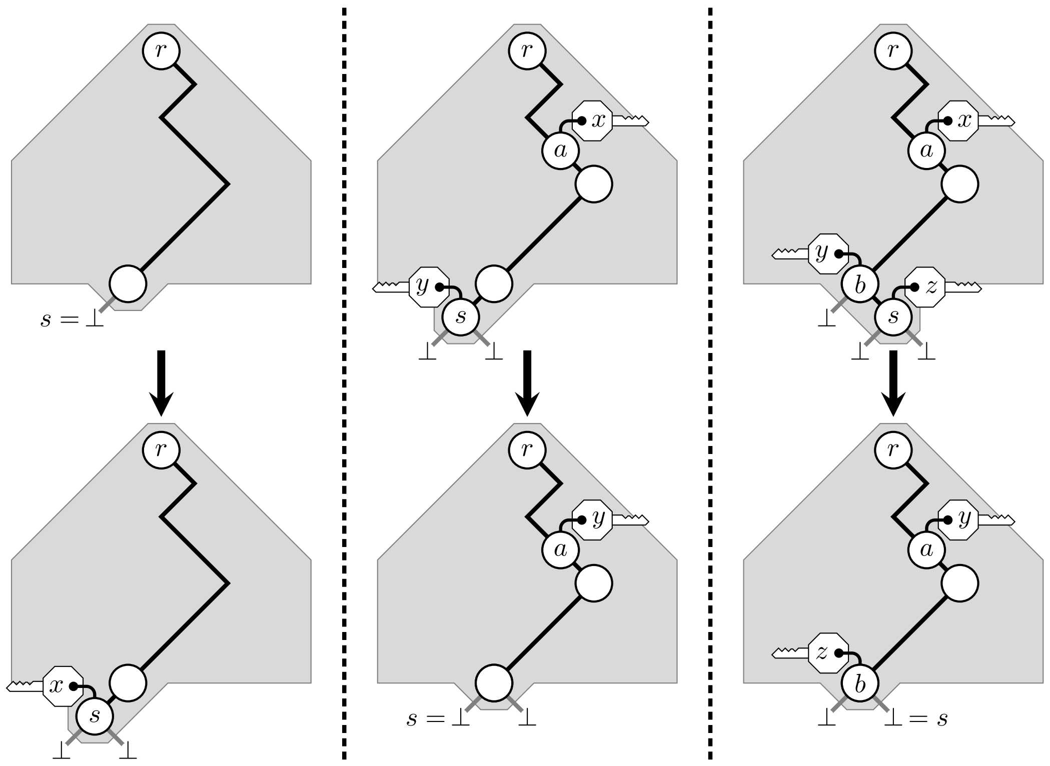

Figure 2 Inserting or deleting a key x: either an empty sub-tree grows into a leaf or a leaf shrinks to an empty sub-tree. When we wish to delete an internal node, we just focus on deleting a leaf descending from that node, and then swap node keys.

In what follows, $\mathcal{T}$ denotes a binary tree that may need to be rebalanced. Its root is denoted by n and, for every node x, the left and right children of x are denoted by x_1 and x_2, respectively. Furthermore, we consider that nodes *move* when rotations are performed. For instance, when either rotation represented in Figure 3 is performed, the node n, which used to be the root of the tree, becomes the right child of the root of the tree resulting from the rotation.

Below, we also say that a node whose children list changed during the rotation was *affected* by the rotation. We will always make sure that, when a simple or double rotation is performed on a tree $\mathcal{T}$, the resulting tree $\mathcal{T}'$ satisfies the inequality $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$. That way, if $\mathcal{T}$ was in fact rooted at some child of a node x, neither the C-balance nor the GC-balance of x, or of any unaffected node, will increase as a result of the rotation.

▶ **Algorithm 5** (C-balancing). *Given real numbers u, v, w and x, the $\mathsf{C}(u, v, w, x)$-balancing algorithm operates as follows on the binary tree $\mathcal{T}$ it receives as input:*

1. *if $|n_1| > u|n| + w$ and $|n_{11}| \geqslant (1 - v)|n| - x$, perform the simple rotation of Figure 3;*
2. *if $|n_2| > u|n| + w$ and $|n_{22}| \geqslant (1 - v)|n| - x$, perform the mirror image of that rotation;*
3. *if $|n_1| > u|n| + w$ and $|n_{11}| < (1 - v)|n| - x$, perform the double rotation of Figure 3;*
4. *if $|n_2| > u|n| + w$ and $|n_{22}| < (1 - v)|n| - x$, perform the mirror image of that rotation;*
5. *if $\max\{|n_1|, |n_2|\} \leqslant u|n| + w$, do not modify $\mathcal{T}$.*

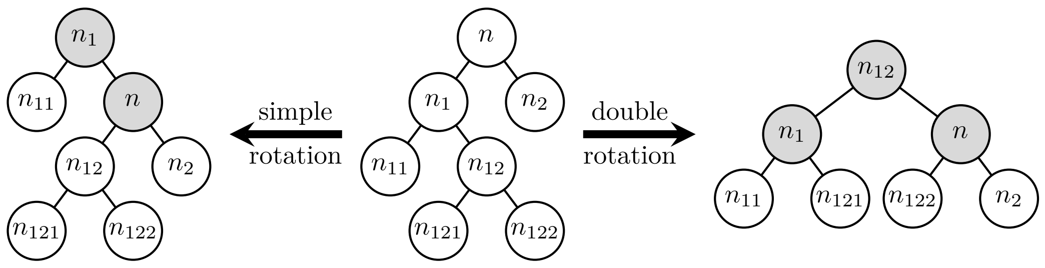

Figure 3 Performing a simple or double rotation. Affected nodes are coloured in grey.

▶ **Algorithm 6** (GC-balancing). *Given real numbers y and z, the* $\mathsf{GC}(y,z)$*-balancing algorithm operates as follows on the binary tree $\mathcal{T}$ it receives as input:*

1. *if $|n_{11}| > y|n| + z$, perform the simple rotation shown in Figure 3;*
2. *if $|n_{22}| > y|n| + z$, perform the mirror image of that rotation;*
3. *if $|n_{12}| > y|n| + z$, perform the double rotation shown in Figure 3;*
4. *if $|n_{21}| > y|n| + z$, perform the mirror image of that rotation;*
5. *if $\max\{|n_{11}|, |n_{12}|, |n_{21}|, |n_{22}|\} \leqslant y|n| + z$, do not modify $\mathcal{T}$.*

Parameters w, x and z can be seen as terms governing some error margin, that we will set equal to zero in Section 3 and will be non-zero in Section 4. Indeed, in the former case, we develop bottom-up updating algorithms, and thus have a perfect knowledge of $\mathcal{T}$. By contrast, in the latter case, we develop top-down algorithms and thus cannot yet know where the leaf s should be inserted or deleted: depending on the answer, the tree $\mathcal{T}$ may have different shapes, and we cannot anticipate which will be chosen.

The idea of the C-balancing algorithm is to either check that our tree root is u-C-balanced or to transform it into a v-C-balanced node; in the latter case, each affected node will also be made v-C-balanced, which will improve the C-balance of our tree root by at least $u - v$ without damaging the balances of other nodes too much. If we perform a simple rotation, n_{11} will be a new root child, and it will definitely not be too large, but its new sibling, of weight $|n| - |n_{11}|$, might prevent our new root from being v-C-balanced; this undesirable case should occur when $|n_{11}| < (1 - v)|n|$, which is why, in that case, we perform a double rotation instead of a simple one. More precisely, here is a result that can be stated about the C-balancing algorithm.

▶ **Lemma 7.** *Let α be a real number such that $1/\sqrt{2} \leqslant \alpha < 3/4$. Then, let $\alpha^\bullet = 19/24$ and*

$$\hat{\alpha} = \frac{1 + \alpha - \sqrt{(1-\alpha)(5-\alpha)}}{2(2\alpha - 1)}.$$

We have $1/\sqrt{2} \leqslant \hat{\alpha} \leqslant \alpha$. Moreover, when $\hat{\alpha} \leqslant u \leqslant \alpha$, the five cases in which the $\mathsf{C}(u, \hat{\alpha}, 0, 0)$*-balancing algorithm consists are pairwise incompatible, and those rotations they trigger can always be performed; the algorithm itself transforms each* $\mathsf{GCB}_{\alpha^\bullet}[\alpha]$*-tree $\mathcal{T}$ whose root is not u-C-balanced into a* $\mathsf{GCB}[\alpha]$*-tree $\mathcal{T}'$ such that* $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$ *and whose affected nodes are $\hat{\alpha}$-C-balanced.*

Proof. Below, we will use the following (in)equalities, which are all easy to check with any computer algebra system (whereas some are quite tedious to check by hand) whenever $1/\sqrt{2} \leqslant \alpha < 3/4$. Except the first two inequalities, each of them is labelled and later reused to prove another inequality with the same label; equality (2.1 + 2.5) is used to prove both subsequent inequalities (2.1) and (2.5).

$$
\begin{array}{llll}
1/\sqrt{2} \leqslant \hat{\alpha}; & & \hat{\alpha} \leqslant \alpha; & \\
\alpha\alpha^\bullet < \hat{\alpha}; & (1.1) & (1-\hat{\alpha})(\alpha^\bullet - 1 + \hat{\alpha}) < \hat{\alpha}(1-\alpha^\bullet); & (1.2) \\
(1-\hat{\alpha})^2 < \hat{\alpha}^2(1-\alpha); & (1.3) & (1-\hat{\alpha})^2 = \hat{\alpha}(1-\alpha)(2\hat{\alpha}-1); & (2.1+2.5) \\
(1-\hat{\alpha})\alpha^2 < \hat{\alpha}(1-\alpha); & (2.2) & (1-\hat{\alpha})(1-\alpha) + \alpha\alpha^\bullet < \hat{\alpha}; & (2.3) \\
(1-\hat{\alpha})\alpha^2\alpha^\bullet < \hat{\alpha}(1-\alpha^\bullet); & (2.4) & 1 + (\alpha^2-1)\hat{\alpha} < \hat{\alpha}. & (2.6)
\end{array}
$$

This already proves the inequality $1/\sqrt{2} \leqslant \hat{\alpha} \leqslant \alpha$ of Lemma 7.

Then, let $|m|$ denote the weight of a node m in the tree $\mathcal{T}$, and let $|m|'$ denote its weight in $\mathcal{T}'$. When these weights are equal, we will prefer using the notation $|m|$ even when considering the weight of m in $\mathcal{T}'$. The root of $\mathcal{T}$ is denoted by n, its left and right children are denoted by n_1 and n_2, and so on. Hence, when $u \geqslant \hat{\alpha}$, we have $2u|n| \geqslant \sqrt{2}|n| \geqslant |n_1|+|n_2|$, which makes the inequalities $|n_1| > u|n|$ and $|n_2| > u|n|$ incompatible.

Proving that the desired rotations can indeed be performed amounts to showing that n and n_1 are actual tree nodes (instead of spurious empty nodes of weight $1/2$ or less) in case 1, and that n_{12} is also a tree node in case 3; cases 2 and 4 will be treated symmetrically. In cases 1 and 3, since $|n_1| > u|n| > |n|/2$, the node n is an actual tree node, and cannot be a leaf, which means, as desired, that n and n_1 are tree nodes; furthermore, $|n_1| > u(|n_1|+1) > (|n_1|+1)/\sqrt{2}$, i.e., $|n_1| > 1 + \sqrt{2} > 2$, and therefore $|n_1| \geqslant 3$ and $|n| \geqslant 4$. Then, in case 3, we also have $|n_{12}| = |n_1| - |n_{11}| > u|n| - (1-\hat{\alpha})|n| \geqslant (\sqrt{2}-1)|n| > 1$, which means that n_{12} is also a tree node.

It remains to prove that each affected node is $\hat{\alpha}$-C-balanced: if the root of $\mathcal{T}$ was not u-C-balanced, we will have $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \hat{\alpha} \leqslant u \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$.

When $\alpha^\bullet|n| \geqslant |n_1| > u|n| \geqslant \hat{\alpha}|n|$ and $|n_{11}| \geqslant (1-\hat{\alpha})|n|$, a simple rotation is performed, and

$$
\begin{aligned}
|n_{11}| &\leqslant \alpha|n_1| \leqslant \alpha\alpha^\bullet|n| \leqslant \hat{\alpha}|n| = \hat{\alpha}|n_1|'; && (1.1)\\
(1-\hat{\alpha})|n_{12}| &= (1-\hat{\alpha})(|n_1| - |n_{11}|) \leqslant (1-\hat{\alpha})(\alpha^\bullet|n| - (1-\hat{\alpha})|n|) \\
&\leqslant \hat{\alpha}(1-\alpha^\bullet)|n| \leqslant \hat{\alpha}(|n| - |n_1|) = \hat{\alpha}|n_2|; && (1.2)\\
(1-\hat{\alpha})|n_2| &= (1-\hat{\alpha})(|n| - |n_1|) \leqslant (1-\hat{\alpha})(1-\hat{\alpha})|n| \\
&\leqslant \hat{\alpha}(1-\alpha)\hat{\alpha}|n| \leqslant \hat{\alpha}(1-\alpha)|n_1| \leqslant \hat{\alpha}(|n_1| - |n_{11}|) = \hat{\alpha}|n_{12}|; && (1.3)\\
|n|' &= |n| - |n_{11}| \leqslant |n| - (1-\hat{\alpha})|n| = \hat{\alpha}|n| = \hat{\alpha}|n_1|', && (1.4)
\end{aligned}
$$

which means precisely that n and n_1 are $\hat{\alpha}$-C-balanced in $\mathcal{T}'$.

Similarly, when $\alpha^\bullet|n| \geqslant |n_1| > u|n| \geqslant \hat{\alpha}|n|$ and $|n_{11}| < (1-\hat{\alpha})|n|$, a double rotation is performed, and

$$
\begin{aligned}
(1-\hat{\alpha})|n_{11}| &\leqslant (1-\hat{\alpha})^2|n| = \hat{\alpha}(1-\alpha)(\hat{\alpha}|n| - (1-\hat{\alpha})|n|) \\
&\leqslant \hat{\alpha}(1-\alpha)(|n_1| - |n_{11}|) = \hat{\alpha}(1-\alpha)|n_{12}| \\
&\leqslant \hat{\alpha}(|n_{12}| - |n_{122}|) = \hat{\alpha}|n_{121}|; && (2.1)\\
(1-\hat{\alpha})|n_{121}| &\leqslant (1-\hat{\alpha})\alpha|n_{12}| \leqslant (1-\hat{\alpha})\alpha^2|n_1| \\
&\leqslant \hat{\alpha}(1-\alpha)|n_1| \leqslant \hat{\alpha}(|n_1| - |n_{12}|) = \hat{\alpha}|n_{11}|; && (2.2)\\
|n_1|' &= |n_{11}| + |n_{121}| \leqslant |n_{11}| + \alpha|n_{12}| = (1-\alpha)|n_{11}| + \alpha|n_1| \\
&\leqslant (1-\alpha)(1-\hat{\alpha})|n| + \alpha\alpha^\bullet|n| \leqslant \hat{\alpha}|n| = \hat{\alpha}|n_{12}|'; && (2.3)\\
(1-\hat{\alpha})|n_{122}| &\leqslant (1-\hat{\alpha})\alpha|n_{12}| \leqslant (1-\hat{\alpha})\alpha^2|n_1| \leqslant (1-\hat{\alpha})\alpha^2\alpha^\bullet|n| \\
&\leqslant \hat{\alpha}(1-\alpha^\bullet)|n| = \hat{\alpha}|n_2|; && (2.4)
\end{aligned}
$$

$$\begin{aligned}(1-\hat{\alpha})|n_2| = (1-\hat{\alpha})(|n|-|n_1|) &\leqslant (1-\hat{\alpha})^2|n| = \hat{\alpha}(1-\alpha)(\hat{\alpha}|n|-(1-\hat{\alpha})|n|) \\ &\leqslant \hat{\alpha}(1-\alpha)(|n_1|-|n_{11}|) = \hat{\alpha}(1-\alpha)|n_{12}| \\ &\leqslant \hat{\alpha}(|n_{12}|-|n_{121}|) = \hat{\alpha}|n_{122}|; \qquad (2.5) \\ |n|' = |n|-|n_1|+|n_{122}| &\leqslant |n|-|n_1|+\alpha|n_{12}| \\ &\leqslant |n|+(\alpha^2-1)|n_1| \leqslant |n|+(\alpha^2-1)\hat{\alpha}|n| \leqslant \hat{\alpha}|n| = \hat{\alpha}|n_{12}|', \qquad (2.6)\end{aligned}$$

which means precisely that n, n_1 and n_{12} are $\hat{\alpha}$-C-balanced in $\mathcal{T}'$.

Finally, the cases where $|n_2| > u|n|$ are symmetrical, and therefore the conclusion of Lemma 7 is valid in these cases too. ◀

Similarly, the GC-balancing algorithm aims at making our tree root y-GC-balanced. If our tree is too unbalanced, we will either promote the responsible grandchild to being a child (if this grandchild was n_{11} or n_{22}) with a simple rotation, or split it in two (if this grandchild was n_{12} or n_{21}) with a double rotation. For adequate values of the parameters u, v and y, each node affected by the GC-balancing algorithm will be $(v, \hat{y})$-balanced for some real number $\hat{y} \leqslant y$ (that does not need to be given as parameter), thus avoiding any damage to the GC-balance of affected nodes, and improving that of our tree root by at least $y - \hat{y}$. Hence, here is a result that can be stated about the GC-balancing algorithm; its proof is similar to that of Lemma 7, and can be found in appendix.

▶ **Lemma 8.** *Let α and β be real numbers such that $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$. Then, let*

$$\hat{\alpha} = \frac{1-2\alpha+6\alpha^2-\sqrt{(1-\alpha)(5-\alpha)}}{5(2\alpha-1)}, \ \hat{\beta} = \frac{1+\alpha-\sqrt{(1-\alpha)^2+4\alpha(\alpha^2-\beta)}}{2(1-\alpha^2+\beta)}\alpha,$$

as well as $\delta = \min\{\hat{\alpha}, \hat{\beta}/\alpha\}$.

We have $1/4 < \hat{\beta} \leqslant \beta$. Moreover, when $\hat{\alpha} \leqslant u \leqslant \alpha$ and $\hat{\beta} \leqslant y \leqslant \beta$, if we apply the $\mathsf{GC}(y,0)$-balancing algorithm on a $\mathsf{GCB}_{\hat{\alpha},1}[\alpha,\beta]$-tree $\mathcal{T}$, cases 1 to 5 are pairwise incompatible, and those rotations they trigger can always be performed; if the root of $\mathcal{T}$ is not y-GC-balanced, the algorithm transforms $\mathcal{T}$ into a $\mathsf{GCB}_{u,y}[\alpha,\beta]$-tree $\mathcal{T}'$ such that $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$ and whose affected nodes are $(\delta, \hat{\beta})$-balanced.

Finally, these building blocks can be combined to ensure that a $\mathsf{GCB}[\alpha,\beta]$-tree whose root is suddenly slightly out of line will be rebalanced.

▶ **Algorithm 9** (CGC-balancing)**.** *Given real numbers u, v, y and $\hat{y}$, the $\mathsf{CGC}(u,v,y,\hat{y})$-balancing algorithm executes the following operations on the tree $\mathcal{T}$ it receives as input:*

1. *the $\mathsf{C}(u,v,0,0)$-balancing algorithm transforms $\mathcal{T}$ into a tree $\mathcal{T}^{(1)}$;*
2. *if $\mathcal{T} \neq \mathcal{T}^{(1)}$,*
   - *the $\mathsf{GC}(\hat{y},0)$-balancing algorithm is applied (in parallel) to the left and right sub-trees of $\mathcal{T}^{(1)}$, which transforms $\mathcal{T}^{(1)}$ into a tree $\mathcal{T}^{(2)}$;*
   - *then, the $\mathsf{GC}(\hat{y},0)$-balancing algorithm is applied to $\mathcal{T}^{(2)}$ itself;*
3. *otherwise, the $\mathsf{GC}(y,0)$-balancing algorithm is applied to $\mathcal{T}$.*

▶ **Proposition 10.** *Let α and β be real numbers such that $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$; we recall that $\mathscr{B}(\alpha) = (\sqrt{1+4\alpha}-1)/2$. Then, let $\alpha^\bullet = 19/24$,*

$$\hat{\alpha} = \frac{1-2\alpha+6\alpha^2-\sqrt{(1-\alpha)(5-\alpha)}}{5(2\alpha-1)} \text{ and } \hat{\beta} = \frac{1+\alpha-\sqrt{(1-\alpha)^2+4\alpha(\alpha^2-\beta)}}{2(1-\alpha^2+\beta)}\alpha.$$

When $\hat{\alpha} \leqslant u \leqslant \alpha$ and $\hat{\beta} \leqslant y \leqslant \beta$, the $\mathsf{CGC}(u, \hat{\alpha}, y, \hat{\beta})$*-balancing algorithm, when applied to a* $\mathsf{GCB}_{\alpha^\bullet,1}[\alpha, \beta]$*-tree* $\mathcal{T}$ *whose root is not* (u, y)*-balanced, transforms* $\mathcal{T}$ *into a* $\mathsf{GCB}_{\hat{\alpha},\hat{\beta}}[\alpha, \beta]$*-tree* $\mathcal{T}'$ *such that* $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$ *and whose affected nodes are all* $(\hat{\alpha}, \hat{\beta})$*-balanced.*

Proof. If the root of $\mathcal{T}$ is (u, y)-balanced, the CGC-balancing algorithm does nothing. If this root is u-C-balanced but not y-GC-balanced, the $\mathsf{GC}(y, 0)$-balancing algorithm is called; Lemma 8 ensures that every affected node will be $(\hat{\alpha}, \hat{\beta})$-balanced and that $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$.

Finally, if the root of $\mathcal{T}$ it is not u-C-balanced, we will call the $\mathsf{C}(u, \hat{\alpha}, 0, 0)$-algorithm, obtaining a $\mathsf{GCB}[\alpha, \beta]$-tree $\mathcal{T}^{(1)}$, whose root will be denoted by r. Nodes affected by this call are r and one or two of its children r_1 and r_2. The $\mathsf{GC}(\hat{\beta}, 0)$-balancing algorithm is then applied to the $\mathsf{GCB}_{\alpha,1}[\alpha, \beta]$-trees rooted at r_1 and r_2; as a result, $\mathcal{T}^{(2)}$ is a $\mathsf{GCB}_{\alpha,1}[\alpha, \beta]$-tree rooted at r, to which the $\mathsf{GC}(\hat{\beta}, 0)$-balancing algorithm is applied once more. No tree to which the C- and GC-balancing algorithms were applied saw its C-balance increase, and thus $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$.

Moreover, each node m affected by the CGC-balancing algorithm is either affected by some call to the GC-balancing algorithm or the root of some tree that the GC-balancing algorithm did not modify; the latter case may only concern nodes r, r_1 and r_2. In both cases, m ends up being $(\hat{\alpha}, \hat{\beta})$-balanced, which completes the proof. ◀

Now is the time when we actually describe our update algorithm, which we will apply to a $\mathsf{GCB}[\alpha, \beta]$-tree in which a leaf s is to be inserted, or from which s is to be deleted. We have two regimes: the first one concerns trees with 11 nodes or less, which we can just restructure in order to make them as balanced as possible, and the second regime concerns trees with 12 nodes or more.

▶ **Algorithm 11** (Bottom-up update). *Let α, $\hat{\alpha}$ and β be real numbers given in Proposition 10. Our updating algorithm executes the following operations on the tree $\mathcal{T}$ it receives as input:*

- *if $\mathcal{T}$ contains 11 nodes or less, insert or delete s (if possible) and make the resulting tree a perfectly balanced tree, i.e., a tree in which the weights of two siblings differ by at most 1;*
- *if $\mathcal{T}$ contains 12 nodes or more,*
  1. *recursively rebalance the (left or right) sub-tree in which s lies, and then*
  2. *if the update turned out to be non-redundant, update the weight of the root of $\mathcal{T}$, and then apply the* $\mathsf{CGC}(\alpha, \hat{\alpha}, \beta, \hat{\beta})$*-balancing algorithm to $\mathcal{T}$.*

Due to the recursive flavour of this algorithm, the rebalancing operations are performed bottom-up. We will see in Section 4 how to perform these operations top-down.

▶ **Theorem 12.** *Let α and β be real numbers such that $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$. The tree obtained by using Algorithm 11 to insert a leaf in a* $\mathsf{GCB}[\alpha, \beta]$*-tree $\mathcal{T}$ or delete a node from $\mathcal{T}$ is also a* $\mathsf{GCB}[\alpha, \beta]$*-tree.*

Proof. Our proof is a variant of the proof of [5].

Let $\mathcal{T}'$ and $\mathcal{T}''$ be the trees obtained just after step 1 and step 2 of Algorithm 11, respectively. We will prove by induction on $|\mathcal{T}|$ that $\mathcal{T}''$ is a $\mathsf{GCB}[\alpha, \beta]$-tree. The result being correct by construction when $|\mathcal{T}| \leqslant 12$, we assume that $|\mathcal{T}| \geqslant 13$.

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the left and right sub-trees of $\mathcal{T}$, and let $t = |\mathcal{T}|$, $t_1 = |\mathcal{T}_1|$ and $t_2 = |\mathcal{T}_2|$. Similarly, let $\mathcal{T}_1'$ and $\mathcal{T}_2'$ be the left and right sub-trees of $\mathcal{T}'$, and let $t' = |\mathcal{T}'|$ $t_1' = |\mathcal{T}_1'|$ and $t_2' = |\mathcal{T}_2'|$. Without loss of generality, we assume that the update was non-redundant, and that the tree $\mathcal{T}$ was altered by adding s to $\mathcal{T}_1$ or by deleting s from $\mathcal{T}_2$. This means that either $(t', t_1', t_2') = (t+1, t_1+1, t_2)$ or $(t_1', t_2') = (t-1, t_1, t_2-1)$. In both cases, the induction hypothesis ensures that $\mathcal{T}_1'$ and $\mathcal{T}_2'$ are $\mathsf{GCB}[\alpha, \beta]$-trees.

We prove now that the root of $\mathcal{T}'$ is $\alpha^\bullet$-C-balanced, thereby allowing us to use Proposition 10 and completing the induction. Indeed, let $t = t_1 + t_2$ and $t' = t'_1 + t'_2$. Since $4t_1 \leqslant 4\alpha t < 3t$, we know that $4t_1 \leqslant 3t - 1$.

Thus, in case of an insertion, $t'_1 = t_1 + 1 \leqslant (3t + 3)/4 = 3t'/4 \leqslant \alpha^\bullet t'$ and $t'_2 = t_2 \leqslant \alpha t \leqslant \alpha^\bullet t'$.

Similarly, in case of a deletion, $t'_1 = t_1 \leqslant (3t-1)/4 = (3t'+2)/4 = \alpha^\bullet t' - (t'-12)/24 \leqslant \alpha^\bullet t'$, whereas $t'_2 = t_2 - 1 \leqslant \alpha t - 1 \leqslant \alpha t' \leqslant \alpha^\bullet t'$. ◀

4 Top-down algorithm

In this section, we propose a top-down algorithm for inserting an element into (or deleting an element from) a weight-balanced tree. This algorithm is valid whenever $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$, thereby completing the algorithm of Lai and Wood [14], which works only when $3/4 \leqslant \alpha \leqslant 9/11$ and $\beta = \alpha^2$. This new algorithm is inspired by theirs: we wish to perform a top-down restructuring pass while adjusting weight information. If the update is redundant, a second top-down pass will be needed to update this information, but no further restructuring will be needed.

More precisely, we aim at having a top-down algorithm that requires considering only a constant number of tree nodes at each step; this number may not depend on the parameters such as α and β. Moreover, we still wish, by using this algorithm, to perform only a constant number of rotations per update; this number *may* depend on α, β and other parameters. Consequently, the idea of the algorithm, which will be made more precise in Algorithm 16, is as follows:

1. If the tree is large enough, we may rebalance it to make sure it will remain balanced even if we recursively update one of its children. Like in Section 3, rotations performed in this phase should be so efficient that only an amortised constant number of rotations per update will be useful.
2. If the tree is small enough, we make it as balanced as possible, the notion of being top-down being void in this case.

A key object towards defining and proving the correctness of our algorithm is the notion of *robust* grandchildren-balanced trees.

▶ **Definition 13.** *Let n be a node of a binary tree $\mathcal{T}$, let n_1 and n_2 be its children, and let n_{11}, n_{12}, n_{21} and n_{22} be its grandchildren. Given real numbers α and β, we say that n is* robustly *α-C-balanced when*

$$\max\{|n_1|,|n_2|\} + 1 \leqslant \alpha(|n|+1) \text{ and } \max\{|n_1|,|n_2|\} \leqslant \alpha(|n|-1);$$

that n is robustly *β-GC-balanced when*

$$\max\{|n_{11}|,|n_{12}|,|n_{21}|,|n_{22}|\}+1 \leqslant \beta(|n|+1) \text{ and } \max\{|n_{11}|,|n_{12}|,|n_{21}|,|n_{22}|\} \leqslant \beta(|n|-1);$$

and that n is robustly *(α,β)-balanced when n is robustly α-C-balanced and robustly β-GC-balanced.*

Finally, we say that a tree $\mathcal{T}$ is a $\mathsf{RGCB}_{x,y}[\alpha,\beta]$-tree when its root is robustly (x,y)-balanced and its other nodes are (α,β)-balanced. For the sake of concision, we simply say that $\mathcal{T}$ is a $\mathsf{RGCB}[\alpha,\beta]$-tree or a $\mathsf{RGCB}[\alpha]$-tree when $(x,y) = (\alpha,\beta)$ or $(x,y,\beta) = (\alpha,1,1)$, respectively.

Inequalities $|n_i|+1 \leqslant \alpha(|n|+1)$ and $|n_i| \leqslant \alpha(|n|-1)$ ensure that, even if a leaf is inserted into or deleted from $\mathcal{T}$, the node n will remain α-C-balanced. Inequalities $|n_{ij}|+1 \leqslant \beta(|n|+1)$ and $|n_{ij}| \leqslant \beta(|n|-1)$ serve the same purpose, but for ensuring that n remains β-GC-balanced. Finally, each tree node is robustly $(1,1)$-balanced.

Setting $\mathsf{R}(t) = \min\{-t, t-1\}$ and observing that $\min\{t(|n|+1)-1, t(|n|-1)\} = t|n|+\mathsf{R}(t)$ for all real numbers t provides us with a more succinct characterisation of (α,β)-balanced nodes, which will require giving non-zero values to the parameters w, x and z: the node n is robustly (α,β)-balanced if and only if $\max\{|n_1|,|n_2|\} \leqslant \alpha|n| + R(\alpha)$ and $\max\{|n_{11}|,|n_{12}|,|n_{21}|,|n_{22}|\} \leqslant \beta|n| + \mathsf{R}(\beta)$.

We can now group several calls to the C- and GC-balancing algorithms into a so-called RCGC-balancing algorithm, whose aim is to make a tree root robustly balanced.

▶ **Algorithm 14** (RCGC-balancing). *Given real numbers u, v, y and $\hat{y}$, the $\mathsf{RCGC}(u,v,y,\hat{y})$-balancing algorithm executes the following operations, in this order, on the tree $\mathcal{T}$ it receives as input, thus transforming it into a tree $\mathcal{T}'$:*

1. *the $\mathsf{C}(u,v,\mathsf{R}(u),\mathsf{R}(v))$-balancing algorithm transforms $\mathcal{T}$ into a tree $\mathcal{T}^{(1)}$;*
2. *if $\mathcal{T} \neq \mathcal{T}^{(1)}$,*
   - *the $\mathsf{CGC}(v,v,\hat{y},\hat{y})$-balancing algorithm is applied (in parallel) to the left and right sub-trees of $\mathcal{T}^{(1)}$, which transforms $\mathcal{T}^{(1)}$ into a tree $\mathcal{T}^{(2)}$;*
   - *the $\mathsf{GC}(\hat{y},\mathsf{R}(\hat{y}))$-balancing algorithm transforms $\mathcal{T}^{(2)}$ into a tree $\mathcal{T}^{(3)}$;*
   - *the $\mathsf{CGC}(v,v,\hat{y},\hat{y})$-balancing algorithm is applied (in parallel) to the left and right sub-trees of $\mathcal{T}^{(3)}$, which yields the tree $\mathcal{T}'$;*
3. *otherwise, the $\mathsf{GC}(y,\mathsf{R}(y))$-balancing algorithm transforms $\mathcal{T}$ into a tree $\mathcal{T}^{(3)}$, and then,*
   - *if $\mathcal{T} \neq \mathcal{T}^{(3)}$, the $\mathsf{CGC}(v,v,\hat{y},\hat{y})$-balancing algorithm is applied (in parallel) to the left and right sub-trees of $\mathcal{T}^{(3)}$, which yields the tree $\mathcal{T}'$;*
   - *otherwise, $\mathcal{T}' = \mathcal{T}$.*

The idea is similar to that of the plain C- and GC-balancing algorithm, but we wish to transform a tree whose root is not robustly (u,y)-balanced into a tree $\mathcal{T}'$ whose root will be robustly $(v,\hat{y})$-balanced. Calling the $\mathsf{C}(u,v,\mathsf{R}(u),\mathsf{R}(v))$-, $\mathsf{GC}(\hat{y},\mathsf{R}(\hat{y}))$- and $\mathsf{GC}(y,\mathsf{R}(y))$-balancing algorithms achieves this goal, while possibly damaging the balance of the children of the root of $\mathcal{T}'$; we solve this issue by the CGC-balancing algorithm to the sub-trees rooted at these children.

▶ **Proposition 15.** *Let α, $\hat{\alpha}$, β and $\hat{\beta}$ be real numbers given in Proposition 10. The $\mathsf{RCGC}(\alpha,\hat{\alpha},\beta,\hat{\beta})$-balancing algorithm transforms each $\mathsf{GCB}_{\alpha,1}[\alpha,\beta]$-tree $\mathcal{T}$ with 30 nodes or more and whose root is not robustly (α,β)-balanced into a $\mathsf{RGCB}[\alpha,\beta]$-tree $\mathcal{T}'$ such that $\mathsf{bal}_{\mathsf{C}}(\mathcal{T}') \leqslant \mathsf{bal}_{\mathsf{C}}(\mathcal{T})$ and whose affected nodes are all $(\hat{\alpha},\hat{\beta})$-balanced.*

We finally describe our top-down updating algorithm, which we will apply to a $\mathsf{GCB}[\alpha,\beta]$-tree in which a leaf s is to be inserted, or from which s is to be deleted. Like the algorithm of Lai and Wood [14], this algorithm is not purely top-down, because (i) if we wanted to delete an internal node a, as illustrated in Figure 2, it requires maintaining a pointer to a, whose key will later be replaced by another node key, and (ii) if the update turned out to be redundant, a second top-down pass will be required to cancel every weight update we performed. Alternative representations of weight-balanced trees or $\mathsf{GCB}[\alpha,\beta]$-trees allow omitting this second pass: instead of storing the weight $|n|$ at each node n, we should store one of the weights $|n_1|$ or $|n_2|$ as well as a bit indicating which weight we stored.

▶ **Algorithm 16** (Top-down update). *Let α, $\hat{\alpha}$, β and $\hat{\beta}$ be real numbers given in Proposition 10. Our updating algorithm executes the following operations on the tree $\mathcal{T}$ it receives as input:*

- *if $\mathcal{T}$ contains 29 nodes or less, insert or delete s (if possible) and make the resulting tree a perfectly balanced tree, i.e., a tree in which the weights of two siblings differ by at most 1;*
- *if $\mathcal{T}$ contains 30 nodes or more,*
  1. *apply the $\mathsf{RCGC}(\alpha,\hat{\alpha},\beta,\hat{\beta})$-balancing algorithm to $\mathcal{T}$, then*
  2. *update the weight of the root of $\mathcal{T}$, and finally*
  3. *recursively update the (left or right) sub-tree of $\mathcal{T}$ that contains s.*

▶ **Theorem 17.** *Let α and β be real numbers such that $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$. The tree obtained by using Algorithm 16 to insert a leaf in a $\mathsf{GCB}[\alpha,\beta]$-tree $\mathcal{T}$ or delete a node from $\mathcal{T}$ is also a $\mathsf{GCB}[\alpha,\beta]$-tree.*

5 Complexity analysis

In this final section, we prove that Algorithms 11 and 16 proposed above perform an amortised constant number of rotations per attempted insertion or deletion.

The idea consists in evaluating the sums of child- and grandchild-balances of all tree nodes. Indeed, inserting or deleting a leaf s will slightly damage the child- and grandchild balances of the ancestors of s, which may increase the sum of these balances by no more than a constant. In the opposite direction, whenever the CGC- or RCGC-balancing algorithm changes a sub-tree $\mathcal{T}$, some node will see its child-balance decrease by approximately $\alpha - \hat{\alpha}$, or its grandchild-balance decrease by approximately $\beta - \hat{\beta}$; other node balances might be damaged in the process, but not to the point of exceeding $(\hat{\alpha}, \hat{\beta})$. Thus, not too many changes may be performed.

These ideas lead to the following result.

▶ **Theorem 18.** *Let α and β be real numbers such that $1/\sqrt{2} < \alpha < 3/4$ and $\mathscr{B}(\alpha) < \beta \leqslant \alpha^2$. Let η and ε be given by $\eta = \alpha - 1/\sqrt{2}$ and $\varepsilon = \min\{\beta - \mathscr{B}(\alpha), \alpha^2 - \beta\}$ if $\mathscr{B}(\alpha) < \beta < \alpha^2$, or $\varepsilon = 1$ if $\beta = \alpha^2$.*

Then, let $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k$ be $\mathsf{GCB}[\alpha,\beta]$-trees defined as follows: $\mathcal{T}_0$ is the empty tree and, for all $\ell \geqslant 0$, the tree $\mathcal{T}_{\ell+1}$ is obtained by inserting a leaf in $\mathcal{T}_\ell$ or deleting a leaf from $\mathcal{T}_\ell$, using either Algorithm 11 or 16 to do so. Gradually transforming $\mathcal{T}_0$ into $\mathcal{T}_k$ via such steps requires only $\mathcal{O}(k/\eta + k/\varepsilon)$ rotations.

Proof outline. Each node n is given a real-valued counter $\mathsf{c}(n)$ that receives the value 0 when n is created (i.e., inserted in a tree) or affected by a rotation, and is increased by $2/|n|$ when a descendant of n is about to be created or deleted.

Finally, let C be the sum of all these counters $\mathsf{c}(n)$, and let $\delta = \min\{\alpha - \hat{\alpha}, \beta - \hat{\beta}, 1/62\}/2$. One can prove that the sum C increases by no more than 16 when a leaf is inserted or deleted, and decreases by at least δ when the CGC- or RCGC-balancing algorithms trigger a rotation and the tree contains at least 32 nodes; if the tree contains fewer than 32 nodes, rotations do not make C increase anyway.

Since the sum C is initially zero, and terminates with a non-negative value, no more than $\mathcal{O}(k/\delta)$ rotations are performed on trees with 32 nodes or more, and no more than $\mathcal{O}(k)$ rotations can be performed on trees with fewer than 32 nodes. ◀

In particular, let us focus on some approach to the critical point $(\alpha_{\mathsf{c}}, \beta_{\mathsf{c}}) = (1/\sqrt{2}, \mathscr{B}(1/\sqrt{2}))$. When setting $\alpha = \alpha_{\mathsf{c}} + x$ and $\beta = \beta_{\mathsf{c}} + x$, and provided that $0 < x < 1/10$, the inequalities $1/\sqrt{2} < \alpha < 3/4$ and $\mathscr{B}(\alpha) < \beta < \alpha^2$ are valid. Moreover,

- Theorem 2 states that $\mathsf{GCB}[\alpha, \beta]$-trees with $\mathbf{N}$ nodes are of height

$$h \leqslant -2 \log_2(\mathbf{N}+1)/\log_2(\beta_\mathsf{c}) + 7 \log_2(\mathbf{N}+1)x;$$

- Theorem 3 states that $\mathsf{GCB}[\alpha, \beta]$-trees with $\mathbf{N}$ nodes are of external path length

$$\lambda \leqslant (\mathbf{N}+1)\log_2(\mathbf{N}+1)/\Delta_\mathsf{c} + 2(\mathbf{N}+1)\log_2(\mathbf{N}+1)x,$$

  where $\Delta_\mathsf{c} = (\mathsf{H}_2(\alpha_\mathsf{c}) + \alpha_\mathsf{c}\mathsf{H}_2(\beta_\mathsf{c}/\alpha_\mathsf{c}))/(1+\alpha_\mathsf{c})$;
- Theorem 18 states that inserting or deleting k leaves in $\mathsf{GCB}[\alpha, \beta]$-trees requires $\mathcal{O}(k/x)$ rotations. More precisely, digging in the constants hidden in the $\mathcal{O}$ notation yields a (crude) upper bound of less than $30(1 + 28/x)k$ rotations.

References

1 Georgii Maksimovich Adel'son-Velskii and Evgenii Mikhailovich Landis. An algorithm for organization of information. *Doklady Akademii Nauk*, 146(2):263–266, 1962.

2 Mahdi Amani, Kevin Lai, and Robert Tarjan. Amortized rotation cost in AVL trees. *Information Processing Letters*, 116(5):327–330, 2016. doi:10.1016/j.ipl.2015.12.009.

3 Lukas Barth and Dorothea Wagner. Engineering top-down weight-balanced trees. In *22nd Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 161–174. SIAM, 2020. doi:10.1137/1.9781611976007.13.

4 Rudolf Bayer and Edward McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972. doi:10.1007/BF00288683.

5 Norbert Blum and Kurt Mehlhorn. On the average number of rebalancing operations in weight-balanced trees. *Theoretical Computer Science*, 11:303–320, 1980. doi:10.1016/0304-3975(80)90018-3.

6 Gerth Stølting Brodal and Allan Grønlund Jørgensen. Data structures for range median queries. In *International Symposium on Algorithms and Computation*, pages 822–831. Springer, 2009. doi:10.1007/978-3-642-10631-6_83.

7 Helen Cameron and Derick Wood. A note on the path length of red-black trees. *Information Processing Letters*, 42(5):287–292, 1992. doi:10.1016/0020-0190(92)90038-W.

8 Rolf Fagerberg, Rune Jensen, and Kim Larsen. Search trees with relaxed balance and near-optimal height. In *7th International Workshop on Algorithms and Data Structures (WADS)*, volume 2125 of *Lecture Notes in Computer Science*, pages 414–425. Springer, 2001. doi:10.1007/3-540-44634-6_38.

9 Leonidas Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 8–21. IEEE Computer Society, 1978. doi:10.1109/SFCS.1978.3.

10 Bernhard Haeupler, Siddhartha Sen, and Robert Tarjan. Rank-balanced trees. *ACM Transactions on Algorithms (TALG)*, 11(4):1–26, 2015. doi:10.1145/2689412.

11 Thomas Hibbard. Some combinatorial properties of certain trees with applications to searching and sorting. *Journal of the ACM*, 9(1):13–28, 1962. doi:10.1145/321105.321108.

12 Vincent Jugé. Grand-children weight-balanced binary search trees, 2024. doi:10.48550/arXiv.2410.08825.

13 Rolf Klein and Derick Wood. A tight upper bound for the path length of AVL trees. *Theoretical Computer Science*, 72(2&3):251–264, 1990. doi:10.1016/0304-3975(90)90037-I.

14 Tony Lai and Derick Wood. A top-down updating algorithm for weight-balanced trees. *International Journal of Foundations of Computer Science*, 4(4):309–324, 1993. doi:10.1142/S0129054193000201.

15 Daan Leijen. Set implementation in Haskell 2010. hackage.haskell.org/package/containers-0.7/docs/Data-Set.html, 2002.

16 Daan Leijen and Andriy Palamarchuk. Map implementation in Haskell 2010. `hackage.haskell.org/package/containers-0.7/docs/Data-Map.html`, 2008.

17 Jürg Nievergelt and Edward Reingold. Binary search trees of bounded balance. In *4th Annual ACM Symposium on Theory of Computing (STOC)*, pages 137–142. ACM, 1972. `doi:10.1145/800152.804906`.

18 Jürg Nievergelt and C. K. Wong. Upper bounds for the total path length of binary trees. *Journal of the ACM*, 20(1):1–6, 1973. `doi:10.1145/321738.321739`.

19 Daniel Sleator and Robert Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985. `doi:10.1145/3828.3835`.

A Missing proofs of Sections 2 and 3

▶ **Theorem 3.** *When $(\alpha,\beta) \in \mathcal{D}$, each $\mathsf{GCB}[\alpha,\beta]$-tree with $\mathbf{N}$ nodes is of external path length*

$$\lambda \leqslant (\mathbf{N}+1)\log_2(\mathbf{N}+1)/\Delta,$$

where $\Delta = (\mathsf{H}_2(\alpha) + \alpha\mathsf{H}_2(\beta/\alpha))/(1+\alpha)$, and $\mathsf{H}_2(x) = -x\log_2(x) - (1-x)\log_2(1-x)$ is Shannon's binary entropy.

Proof. Let $\lambda(\mathcal{T})$ be the sum of the weights of the nodes of a tree $\mathcal{T}$: we prove by induction on $|\mathcal{T}|$ that $\lambda(\mathcal{T}) \leqslant |\mathcal{T}|\log_2(|\mathcal{T}|)/\Delta$.

When $|\mathcal{T}| = 1$, this inequality rewrites as $0 \leqslant 0$. When $|\mathcal{T}| = 2$, it rewrites as $\Delta \leqslant 1$, which follows from the fact that $\mathsf{H}_2(x) \leqslant 1$ whenever $x \in [0,1]$. Therefore, we assume that $|\mathcal{T}| \geqslant 3$.

Let n be the root of $\mathcal{T}$, and let $\mathcal{T}_1$ and $\mathcal{T}_2$ be its left and right sub-trees. Without loss of generality, we assume that $|\mathcal{T}_1| \geqslant |\mathcal{T}_2|$. Thus, $|\mathcal{T}_1| \geqslant |\mathcal{T}|/2 > 1$, and $\mathcal{T}_1$ is non-empty. Let n_1 be its root, and let $\mathcal{T}_{11}$ and $\mathcal{T}_{12}$ be the left and right sub-trees of n_1. Once again, without loss of generality, we assume that $|\mathcal{T}_{11}| \geqslant |\mathcal{T}_{12}|$.

Let $t = |\mathcal{T}|$, $x = |\mathcal{T}_1|/|\mathcal{T}|$ and $y = |\mathcal{T}_{11}|/|\mathcal{T}_1|$. The induction hypothesis states now that

$$\begin{aligned}\Delta\lambda(\mathcal{T})/t &= \Delta(x+1) + \Delta\lambda(\mathcal{T}_{11})/t + \Delta\lambda(\mathcal{T}_{12})/t + \Delta\lambda(\mathcal{T}_2)/t \\ &\leqslant \Delta(x+1) + xy\log_2(txy) + x(1-y)\log_2(tx(1-y)) + (1-x)\log_2(t(1-x)) \\ &\leqslant \Delta(x+1) + \log_2(t) - x\mathsf{H}_2(y) - \mathsf{H}_2(x),\end{aligned}$$

and it remains to prove that the quantity $\mathsf{F}_{\alpha,\beta}(x,y) = x\mathsf{H}_2(y) + \mathsf{H}_2(x) - \Delta(x+1)$ is non-negative.

First, let $\gamma = \beta/\alpha$. If $\gamma \leqslant x \leqslant \alpha$, since H_2 is decreasing on the interval $[1/2,1]$ and $1/2 \leqslant y \leqslant \beta/x$, we know that $\mathsf{F}_{\alpha,\beta}(x,y) \geqslant \mathsf{F}_{\alpha,\beta}(x,\beta/x)$. Moreover, the function $\mathsf{G}\colon x \mapsto \mathsf{F}_{\alpha,\beta}(x,\beta/x)$ is concave on $[\gamma,\alpha]$, because its second derivative is

$$\mathsf{G}''(x) = -\frac{1-\beta}{(x-\beta)(1-x)\ln(2)} < 0$$

whenever $\beta \leqslant \gamma < x < \alpha \leqslant 1$. Observing that $\mathsf{G}(\alpha) = \mathsf{F}_{\alpha,\beta}(\alpha,\gamma) = 0$ and that

$$\begin{aligned}\mathsf{G}(\gamma) = \mathsf{F}_{\alpha,\beta}(\gamma,\alpha) &= (\beta\mathsf{H}_2(\alpha) + \mathsf{H}_2(\gamma)) - (\gamma+1)(\mathsf{H}_2(\alpha) + \alpha\mathsf{H}_2(\gamma))/(\alpha+1) \\ &= (1-\beta)(\mathsf{H}_2(\gamma) - \mathsf{H}_2(\alpha))/(\alpha+1) \geqslant 0,\end{aligned}$$

we conclude that $\mathsf{F}_{\alpha,\beta}(x,y) \geqslant \mathsf{G}(x) \geqslant \min\{\mathsf{G}(\alpha),\mathsf{G}(\gamma)\} \geqslant 0$ whenever $\gamma \leqslant x \leqslant \alpha$. Finally, if $1/2 \leqslant x \leqslant \gamma$, and since $1/2 \leqslant y \leqslant \alpha$, we have $\mathsf{F}_{\alpha,\beta}(x,y) \geqslant \mathsf{F}_{\alpha,x}(x,y) \geqslant \mathsf{F}_{\alpha,x}(x,\alpha) \geqslant 0$. ◀

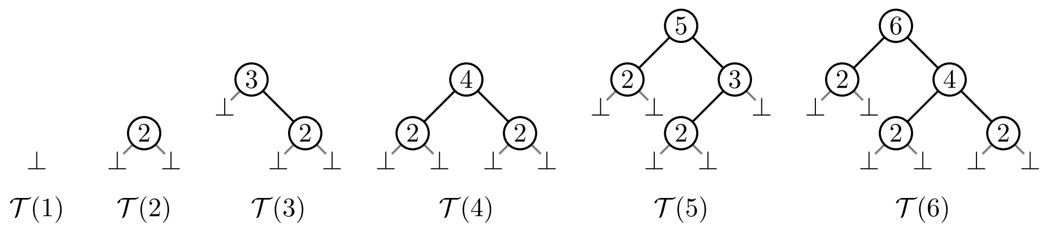

Figure 4 Trees $\mathcal{T}(1)$ to $\mathcal{T}(6)$. Nodes are labelled by their weight; empty trees are denoted by $\perp$.

▶ **Proposition 4.** *Let (α, β) be a pair belonging to $\mathcal{D}''$, and let $\mathbf{N} \geqslant 0$ be an integer. There exists a* $\mathsf{GCB}[\alpha, \beta]$*-tree with $\mathbf{N}$ nodes, of height $h \geqslant -2\log_2(\mathbf{N}+1)/\log_2(\beta) - 7$ and external path length $\lambda \geqslant (\mathbf{N}+1)\log_2(\mathbf{N}+1)/\Delta - 4\mathbf{N}$.*

Proof. Let $\gamma = \beta/\alpha$. For each integer $s \geqslant 1$, we define inductively a tree $\mathcal{T}(s)$ of weight s as follows:

- if $s \leqslant 2$, $\mathcal{T}(s)$ is the only tree of weight s;
- if $s \geqslant 3$, $\mathcal{T}(s)$ is the tree whose left child is $\mathcal{T}(s_1)$ and whose right child's children are $\mathcal{T}(s_{21})$ and $\mathcal{T}(s_{22})$, where $s_1 = s - \lfloor \alpha s \rfloor$, $s_{21} = \lfloor \gamma \lfloor \alpha s \rfloor \rfloor$ and $s_{22} = s - s_1 - s_{21}$.

We present in Figure 4 the trees $\mathcal{T}(s)$ obtained when $1 \leqslant s \leqslant 6$. Since $2/3 < \gamma \leqslant \alpha < 3/4$, these trees do not depend on the values of α and γ (or β).

We first prove by induction on s that $\mathcal{T}(s)$ is a $\mathsf{GCB}[\alpha, \beta]$-tree. This is visibly true when $s \leqslant 4$, hence we assume that $s \geqslant 5$. Let $s_2 = \lfloor \alpha s \rfloor$ be the weight of the right child of $\mathcal{T}(s)$: it suffices to prove that $s_1 \leqslant \gamma s$, $s_2 \leqslant \alpha s$, $s_{21} \leqslant \gamma s_2$ and $s_{22} \leqslant \gamma s_2$. The inequalities $s_2 \leqslant \alpha s$ and $s_{21} \leqslant \gamma s_2$ are immediate, and $s_2 \geqslant \lfloor 2s/3 \rfloor \geqslant 3$, thereby proving that:

- $s_1 \leqslant (1 - \alpha + 1/s)s \leqslant (1 - 2/3 + 1/5)s < 2s/3 < \gamma s$;
- $s_{22} \leqslant (1 - \gamma + 1/s_2)s_2 \leqslant (1 - 2/3 + 1/3)s_2 = 2s_2/3 < \gamma s_2$.

Now, let $h(s)$ be the height of $\mathcal{T}(s)$ and let $\lambda(s)$ be the sum of the weights of the nodes of $\mathcal{T}(s)$. We will prove inductively that $h(s) \geqslant h^+(s+\kappa)$ and $\lambda(s) \geqslant \lambda^+(s+\kappa) + v$ for all $s \geqslant 1$, where we set $h^+(x) = -2\log_\beta(x) - 7$, $\lambda^+(x) = x\log_2(x)/\Delta - 4x$, $\kappa = (\gamma+1)/(1-\beta)$ and $v = (1 + (\alpha+1)\kappa)/2$.

First, we observe, when $(\alpha, \beta) \in \mathcal{D}'$, that $4/5 \leqslant \mathsf{H}_2(3/4) \leqslant \Delta \leqslant \mathsf{H}_2(2/3) \leqslant \log_2(5e)/4$, $3 \leqslant \kappa \leqslant 4$ and $3 \leqslant v \leqslant 4$. We will use these inequalities several times below.

For example, $\kappa + 1 \leqslant 5 \leqslant (4/3)^6 \leqslant \beta^3$ and $\kappa + 2 \leqslant 6 \leqslant (4/3)^7 \leqslant \beta^{7/2}$, so that $h(s) = s - 2 \geqslant h^+(s+\kappa)$ when $s = 1$ or $s = 2$.

Then, if $s \geqslant 3$, observing that $s_{21} + \kappa \geqslant \beta s - \gamma - 1 = \beta(s+\kappa)$ and using the induction hypothesis proves that $h(s) \geqslant h(s_{21}) + 2 \geqslant h^+(s_{21}+\kappa) + 2 \geqslant 2 - 2\log_\beta(\beta(s+\kappa)) - 7 = h^+(s+\kappa)$.

Second, let $\lambda^-(s)$ be the smallest possible sum of the weights of the nodes of a binary tree with weight s. By construction, $\lambda(s) \geqslant \lambda^-(s)$, and λ^- shines as the non-decreasing convex function defined by $\lambda^-(1) = 0$ and $\lambda^-(s) = s + \lambda^-(\lfloor s/2 \rfloor) + \lambda^-(\lceil s/2 \rceil)$ for all $s \geqslant 2$. Consequently, observing that

$$\lambda^+(s+\kappa) + v \leqslant 5(s+\kappa)\log_2(s+\kappa)/4 - 4(s+\kappa) + v \leqslant 5(s+4)\log_2(s+4)/4 - 4(s+3) + 4$$

whenever $s \geqslant 1$ suffices to check (by computer) that $\lambda(s) \geqslant \lambda^-(s) \geqslant s\lfloor \log_2(s) \rfloor \geqslant \lambda^+(s+\kappa)$ for all integers $s \leqslant 26$.

Then, if $s \geqslant 27$, observe that $\alpha(1-\alpha)(s+\kappa) \geqslant 2/3 \times 1/4 \times 30 \geqslant 5$. It follows that:

- $s_1 + \kappa \geqslant (1-\alpha)s + \kappa \geqslant (1-\alpha)(s+\kappa) \geqslant 5$;
- $s_{21} + \kappa \geqslant \alpha\gamma s + \kappa - \gamma - 1 = \alpha\gamma(s+\kappa) \geqslant 5$;
- $s_{22} + \kappa \geqslant \alpha(1-\gamma)s + \kappa + \gamma - 1 \geqslant \alpha(1-\gamma)(s+\kappa) \geqslant 5$.

Moreover, the function λ^+ is increasing on the interval $(16^\Delta/e, +\infty)$, and $16^\Delta/e \leqslant 5$. In addition, the equality $\lambda^+(x) = (\alpha+1)x + \lambda^+((1-\alpha)x) + \lambda^+(\alpha\gamma x) + \lambda^+(\alpha(1-\gamma)x$ is valid for all $x \geqslant 1$. Thus, the induction hypothesis proves that

$$\begin{aligned}
\lambda(s) &\geqslant s + s_2 + \lambda(s_1) + \lambda(s_{21}) + \lambda(s_{22}) \\
&\geqslant s + s_2 + \lambda^+(s_1+\kappa) + \lambda^+(s_{21}+\kappa) + \lambda^+(s_{22}+\kappa) + 3v \\
&\geqslant (\alpha+1)s + \lambda^+((1-\alpha)(s+\kappa)) + \lambda^+(\alpha\gamma(s+\kappa)) + \lambda^+(\alpha(1-\gamma)(s+\kappa)) + (3v-1) \\
&\geqslant \lambda^+(s+\kappa) - (\alpha+1)\kappa + (3v-1) = \lambda^+(s+\kappa) + v.
\end{aligned}$$

Finally, we check by hand that $\lambda(s) \geqslant s\lfloor\log_2(s)\rfloor \geqslant 5s\log_2(s)/4 - 4s + 4 \geqslant s\log_2(s)/\Delta - 4s + 4$ when $1 \leqslant s \leqslant 8$, whereas, since λ^+ is increasing on $[5, +\infty)$,

$$\begin{aligned}
\lambda(s) &\geqslant \lambda^+(s+\kappa) + v \geqslant \lambda^+(s+3) + 3 = (s+3)\log_2(s+3)/\Delta - 4s - 9 \\
&\geqslant (s\log_2(s) + 3\log_2(s+3))/\Delta - 4s - 9 \\
&\geqslant s\log_2(s)/\Delta + 15\log_2(12)/4 - 4s - 9 \geqslant s\log_2(s)/\Delta - 4s + 4
\end{aligned}$$

when $s \geqslant 9$. ◀

▶ **Lemma 8.** *Let α and β be real numbers such that $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$. Then, let*

$$\hat{\alpha} = \frac{1 - 2\alpha + 6\alpha^2 - \sqrt{(1-\alpha)(5-\alpha)}}{5(2\alpha-1)},\ \hat{\beta} = \frac{1 + \alpha - \sqrt{(1-\alpha)^2 + 4\alpha(\alpha^2-\beta)}}{2(1-\alpha^2+\beta)}\alpha,$$

as well as $\delta = \min\{\hat{\alpha}, \hat{\beta}/\alpha\}$.

We have $1/4 < \hat{\beta} \leqslant \beta$. Moreover, when $\hat{\alpha} \leqslant u \leqslant \alpha$ and $\hat{\beta} \leqslant y \leqslant \beta$, if we apply the $\mathsf{GC}(y,0)$-balancing algorithm on a $\mathsf{GCB}_{\hat{\alpha},1}[\alpha,\beta]$-tree $\mathcal{T}$, cases 1 to 5 are pairwise incompatible, and those rotations they trigger can always be performed; if the root of $\mathcal{T}$ is not y-GC-balanced, the algorithm transforms $\mathcal{T}$ into a $\mathsf{GCB}_{u,y}[\alpha,\beta]$-tree $\mathcal{T}'$ such that $\mathsf{bal}_\mathsf{C}(\mathcal{T}') \leqslant \mathsf{bal}_\mathsf{C}(\mathcal{T})$ and whose affected nodes are $(\delta,\hat{\beta})$-balanced.

Proof. Below, we will use the following inequalities, which are all easy to check with any computer algebra system (whereas some look too frightening to check by hand) whenever $1/\sqrt{2} \leqslant \alpha < 3/4$ and $\mathscr{B}(\alpha) \leqslant \beta \leqslant \alpha^2$. Except the first three inequalities, each of them is labelled and later reused to prove another inequality with the same label; the first two inequalities already prove that $1/4 < \hat{\beta} \leqslant \beta$, as stated in Lemma 8.

$1/4 < \hat{\beta}$;		$\hat{\beta} \leqslant \beta$;	
$\alpha < 2\hat{\beta}$;		$\alpha^2 < \delta$;	(3.1)
$(1-\delta)(\alpha-\hat{\beta}) < \delta(1-\alpha)$;	(3.2)	$(1-\delta)(1-\hat{\gamma}) \leqslant \delta\hat{\gamma}(1-\alpha)$;	(3.3)
$1-\hat{\beta} < \delta$;	(3.4)	$(1-\delta)(\alpha-\hat{\beta}) < \delta(\hat{\beta}-\alpha\beta)$;	(4.1)
$(1-\delta)\beta < \delta(1-\alpha)$;	(4.2)	$\alpha - \hat{\beta} + \alpha\beta < \delta$;	(4.3)
$(1-\delta)\alpha\beta < \delta(1-\alpha)$;	(4.4)	$1 \leqslant \delta(1+\beta)$;	(4.5[a])
$1 - \delta + (\delta + \beta\delta - 1)\alpha \leqslant \delta\hat{\beta}$;	(4.5[b])	$1 + (\beta-1)\hat{\gamma} < \delta$.	(4.6)

Then, let $|m|$ denote the weight of a node u in the tree $\mathcal{T}$, and let $|m|'$ denote its weight in $\mathcal{T}'$. When these weights are equal, we will prefer using the notation $|m|$ even when considering the weight of m in $\mathcal{T}'$. The root of $\mathcal{T}$ is denoted by n, its left and right children are denoted by n_1 and n_2, and so on. Finally, we set $\hat{\gamma} = \hat{\beta}/\alpha$, so that $\delta = \min\{\hat{\alpha}, \hat{\gamma}\}$.

Since $2\beta \geqslant \alpha$ and n is α-C-balanced, observing that $2\beta|n| \geqslant \alpha|n| \geqslant |n_1| = |n_{11}| + |n_{12}|$ proves that the inequalities $|n_{11}| > \beta|n|$ and $|n_{12}| > \beta|n|$ are incompatible. Similarly, the nodes n_1 and n_2 are α-C-balanced, and thus observing that $2\beta|n| \geqslant \alpha|n| = \alpha|n_1| + \alpha|n_2| \geqslant |n_{11}| + |n_{21}|$ proves that the inequalities $|n_{11}| > \beta|n|$ and $|n_{21}| > \beta|n|$ are incompatible. Finally, for symmetry reasons, all four inequalities $|n_{ij}| > \beta|n|$ are incompatible: this makes our description of the algorithm unambiguous, as announced.

Then, proving that the desired rotations can indeed be performed simply requires showing that, if some case 1 to 4 happens, the grandchild n_{ij} be a grandchild of n whose weight $|n_{ij}|$ is maximal is an actual tree node instead of an empty node of weight 1/2 or less. If this were the case, we would have $|n_i| = 2|n_{ij}| \leqslant 1$, and n_i would be either an empty node or a leaf; in both cases, we would have $|n| \geqslant 2|n_i|$, i.e., $|n| \geqslant 4|n_{ij}|$, contradicting the inequality $|n_{ij}| > y|n| \geqslant \hat{\beta}|n| > |n|/4$.

It remains to prove that each affected node m is δ-C-balanced; its children being α-C-balanced, it will then be $\hat{\beta}$-GC-balanced. Furthermore, if a rotation was triggered, n was not β-GC-balanced but its children were α-C-balanced, which proves that $\mathsf{bal}_{\mathsf{C}}(\mathcal{T}) > \beta/\alpha \geqslant \hat{\delta} \geqslant \mathsf{bal}_{\mathsf{C}}(\mathcal{T}')$.

When $|n_{11}| > w|n| \geqslant \hat{\beta}|n|$, a simple rotation is performed, and

$$\begin{aligned}
|n_{11}| &\leqslant \alpha|n_1| \leqslant \alpha^2|n| \leqslant \delta|n| = \delta|n_1|'; && (3.1)\\
(1-\delta)|n_{12}| &= (1-\delta)(|n_1| - |n_{11}|) \leqslant (1-\delta)(\alpha|n| - \hat{\beta}|n|) \\
&\leqslant \delta(1-\alpha)|n| \leqslant \delta(|n| - |n_1|) = \delta|n_2|; && (3.2)\\
(1-\delta)|n_2| &= (1-\delta)(|n| - |n_1|) \leqslant (1-\delta)(|n| - |n_{11}|/\alpha) \leqslant (1-\delta)(1-\hat{\gamma})|n| \\
&\leqslant \delta\hat{\gamma}(1-\alpha)|n| \leqslant \delta(1/\alpha - 1)|n_{11}| < \delta(|n_1| - |n_{11}|) = \delta|n_{12}|; && (3.3)\\
|n| &= |n| - |n_{11}| \leqslant (1-\hat{\beta})|n| \leqslant \delta|n| = \delta|n_1|', && (3.4)
\end{aligned}$$

which means precisely that n and n_1 are δ-C-balanced in $\mathcal{T}'$.

Similarly, when $|n_{12}| > w|n| \geqslant \hat{\beta}|n|$, a double rotation is performed, and

$$\begin{aligned}
(1-\delta)|n_{11}| &= (1-\delta)(|n_1| - |n_{12}|) \leqslant (1-\delta)(\alpha|n| - \hat{\beta}|n|) \\
&\leqslant \delta(\hat{\beta}|n| - \alpha\beta|n|) \leqslant \delta(|n_{12}| - \beta|n_1|) \leqslant \delta(|n_{12}| - |n_{122}|) = \delta|n_{121}|; && (4.1)\\
(1-\delta)|n_{121}| &\leqslant (1-\delta)\beta|n_1| \leqslant \delta(|n_1| - \alpha|n_1|) \leqslant \delta(|n_1| - |n_{12}|) = \delta|n_{11}|; && (4.2)\\
|n_1|' &= |n_1| - |n_{12}| + |n_{121}| \leqslant |n_1| - |n_{12}| + \beta|n_1| \\
&\leqslant (\alpha - \hat{\beta} + \alpha\beta)|n| \leqslant \delta|n| = \delta|n_{12}|'; && (4.3)\\
(1-\delta)|n_{122}| &\leqslant (1-\delta)\beta|n_1| \leqslant (1-\delta)\alpha\beta|n| \leqslant \delta(1-\alpha)|n| \leqslant \delta(|n| - |n_1|) = \delta|n_2|; && (4.4)\\
(1-\delta)|n_2| &\leqslant (1-\delta)|n_2| + \beta\delta|n_1| - \delta|n_{121}| = (1-\delta)|n| + (\delta + \beta\delta - 1)|n_1| - \delta|n_{121}| \\
&\leqslant (1-\delta)|n| + (\delta + \beta\delta - 1)\alpha|n| - \delta|n_{121}| && (4.5^{\mathrm{a}})\\
&\leqslant \delta\hat{\beta}|n| - \delta|n_{121}| \leqslant \delta|n_{12}| - \delta|n_{121}| = \delta|n_{122}|; && (4.5^{\mathrm{b}})\\
|n|' &= |n| - |n_1| + |n_{122}| \leqslant |n| + (\beta - 1)|n_1| \\
&\leqslant |n| + (\beta-1)|n_{12}|/\alpha \leqslant |n| + (\beta-1)\hat{\gamma}|n| \leqslant \delta|n| = \delta|n_{12}|', && (4.6)
\end{aligned}$$

which means precisely that n, n_1 and n_{12} are δ-C-balanced in $\mathcal{T}'$.

Finally, the cases where $|n_{21}| > w|n|$ or $|n_{22}| > w|n|$ are symmetrical, and therefore the conclusion of Lemma 8 is valid in these cases too. ◀