Enumeration and Succinct Encoding of AVL Trees

Jeremy Chizewer
University of Waterloo, Canada

Stephen Melczer
University of Waterloo, Canada

J. Ian Munro
University of Waterloo, Canada

Ava Pun
University of Waterloo, Canada

Abstract

We use a novel decomposition to create succinct data structures – supporting a wide range of operations on static trees in constant time – for a variety of tree classes, extending results of Munro, Nicholson, Benkner, and Wild. Motivated by the class of AVL trees, we further derive asymptotics for the information-theoretic lower bound on the number of bits needed to store tree classes whose generating functions satisfy certain functional equations. In particular, we prove that AVL trees require approximately 0.938 bits per node to encode.

2012 ACM Subject Classification Mathematics of computing → Enumeration

Keywords and phrases AVL trees, analytic combinatorics, succinct data structures, enumeration

Digital Object Identifier 10.4230/LIPIcs.AofA.2024.2

Funding *Stephen Melczer*: SM and JC partially funded by NSERC Discovery Grant RGPIN-2021-02382.
J. Ian Munro: JIM and AP partially funded by NSERC Discovery Grant RGPIN-2018-03972.

Acknowledgements The authors thank Andrew Odlyzko for discussions on the asymptotic behaviour of AVL trees and the growth constant α, and thank Sebastian Wild for alerting us to relevant references.

Introduction

AVL trees [1] (named for their discoverers, G. Adelson-Velsky and E. Landis) are a subclass of binary search trees with logarithmic height, a property they maintain with updates during insertions and deletions in logarithmic time. Indeed, AVL trees are the oldest class of binary search trees maintaining logarithmic height and are characterized by the key property that any pair of sibling subtrees differ in height by at most 1. In this paper, we examine the amount of storage needed to encode AVL trees with n nodes, a property intimately related to the number of AVL trees on n nodes. Odlyzko [13] gave a conjectural form for the number of AVL trees on n nodes in the 1980s, anticipating a forthcoming proof, but this proof did not appear in the literature.

If $\mathcal{C} = \bigsqcup_{n=0}^{\infty} \mathcal{C}_n$ is a family of objects, with $\mathcal{C}_n$ denoting the objects of size n in $\mathcal{C}$ then a representation of $\mathcal{C}$ is called *succinct* if it maps each object of $\mathcal{C}_n$ to a unique string of length $\log_2 |\mathcal{C}_n| + o(\log |\mathcal{C}_n|)$. A succinct representation is thus one whose space complexity asymptotically equals, up to lower-order terms, the information-theoretic lower bound. A *succinct data structure* [11, 12] for $\mathcal{C}$ is a succinct representation of $\mathcal{C}$ that supports a range of operations on $\mathcal{C}$ under reasonable time constraints.

35th International Conference on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms (AofA 2024).
Editors: Cécile Mailler and Sebastian Wild; Article No. 2; pp. 2:1–2:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

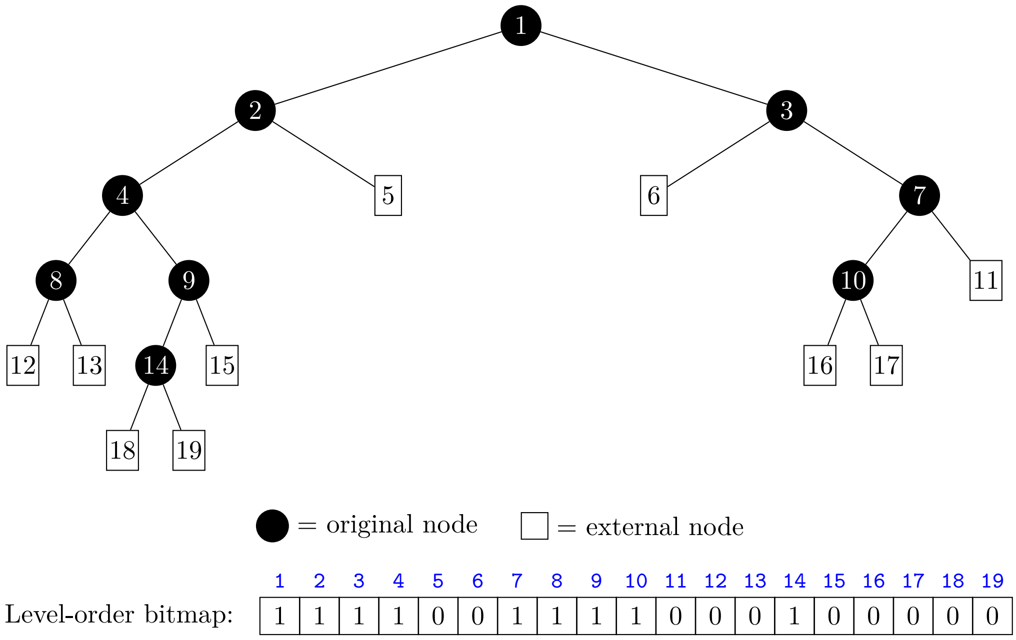

Figure 1 A binary tree and its level-order bitmap representation.

1 Representations of Trees

The theory of succinct data structures has a long history, much of it focused on representations of trees. We first describe some important classes of trees in this context, and then discuss our main results.

Binary Search Trees

Let $\mathcal{B}$ be the class of rooted binary trees, so that the number $|\mathcal{B}_n|$ of objects in $\mathcal{B}$ of size n is the nth Catalan number $b_n = \frac{1}{n+1}\binom{2n}{n}$. The class $\mathcal{B}$ lends itself well to storing ordered data in a structure called a *binary search tree.* The general idea is that for each node in the tree, the data stored in its left subtree will be smaller than the data at that node, and the data stored in the right subtree will be larger. To retrieve elements, one can recursively navigate through the tree by comparing the desired element to the current node, and moving to the left or right subtree if the element is respectively smaller or larger than the current node. As a result, it is desirable to efficiently support the navigation operations of moving to parent or child nodes in whatever representation is used.

A naive representation of $\mathcal{B}$ gives each node a label (using roughly $\log_2 n$ space) and stores the labels of each node's children and parent. The resulting data structure supports operations like finding node siblings in constant time, but is *not* succinct as it uses $\Theta(n \log n)$ bits while the information-theoretic lower bound is only $\log_2(b_n) = 2n + o(n)$. Somewhat conversely, a naive space-optimal representation of $\mathcal{B}$ is obtained by listing the objects of $\mathcal{B}_n$ in any canonical order and referencing a tree by its position $\{1, \dots, b_n\}$ in the order, but asking for information like the children or parents of a node in a specific tree is then expensive as it requires building parts of the tree.

Practical succinct representations of binary trees supporting efficient navigation date back to Jacobson [6], who encoded a tree by storing the binary string of length $2n+1$ obtained by adding *external* vertices so that every node has exactly two children, then taking a level-order traversal of the tree and recording a 1 for each original *internal* node encountered and a 0 for each external node encountered (see Figure 1). If each node is labelled by its position in a level-order traversal then, for instance, the children of the node labelled x in the tree encoded by the string σ have labels $2\,\mathrm{rank}_x(\sigma)$ and $2\,\mathrm{rank}_x(\sigma)+1$, where $\mathrm{rank}_x(\sigma)$ is the number of ones in σ up to (and including) the position x. By storing $o(n)$ bits, the rank operation (and similar supporting operations used to retrieve information about the trees) can be implemented in $O(1)$ time. Jacobson's results allow finding a parent or child using $O(\log_2 n)$ bit inspections; Clark [2] and Munro [8] improved this to $O(1)$ inspections of $\log_2 n$ bit words.

AVL Trees

Because the time taken to access elements in a binary search tree typically depends on the height of the tree, many data structures *balance* their trees as new data is added. The balance operation requires rearranging the tree while preserving the underlying property that, for each node, the elements in the left subtree are smaller and the elements in the right subtree are larger. One of the most popular balanced tree structures – for theoretical study and practical application – are *AVL trees* [1]. Roughly speaking, AVL trees have balancing rules that force the subtrees rooted at the children of any node differ in height by at most one. Throughout this paper we let $\mathcal{A}$ denote the class of AVL trees, so that $\mathcal{A}_n$ consists of all binary trees on n vertices such that the subtrees of any vertex differ in height by at most one (including empty subtrees).

Due to the way they are constructed, AVL trees have mainly been enumerated under height restrictions, and enumeration by number of vertices (which is crucial for determining space-efficient representations, but not as important for other applications) is less studied. A 1984 paper [13] of Odlyzko describes the behaviour of a family of trees whose generating functions satisfy certain equations. It ends by stating that the generating function of AVL trees "appears not to satisfy any simple functional equation, but by an intensive study... it can be shown" that $|\mathcal{A}_n| \sim n^{-1}\alpha^{-n}u(\log n)$ where $\alpha = 0.5219\dots$ is "a certain constant" and u is a periodic function, referencing for details a paper that was planned to be published but was never written.[1]

Efficiently Representing Tree Classes

Let B be a function satisfying $B(n) = \Theta(\log n)$. In [9] the authors give a method to construct a succinct encoding, and corresponding data structure, for any class of binary trees $\mathcal{T}$ satisfying the following four conditions.

1. *Fringe-hereditary*: For any tree $\tau \in \mathcal{T}$ and node $v \in \tau$ the *fringe subtree* $\tau[v]$, which consists of v and all of its descendants in τ, also belongs to $\mathcal{T}$.
2. *Worst-case B-fringe dominated*: Most nodes in members of $\mathcal{T}$ do not generate large fringe subtrees, in the sense that

   $$\Big|\{v \in \tau : \big|\tau[v]\big| \;\geq\; B(n)\}\Big| = o(n/\log B(n))$$

   for every binary tree τ in the subset $\mathcal{T}_n \subset \mathcal{T}$ containing the members of $\mathcal{T}$ with n nodes, where $|\tau|$ denotes the number of nodes in τ.

[1] The current authors thank Andrew Odlyzko for discussions on the asymptotic behaviour of AVL trees and the growth constant α.

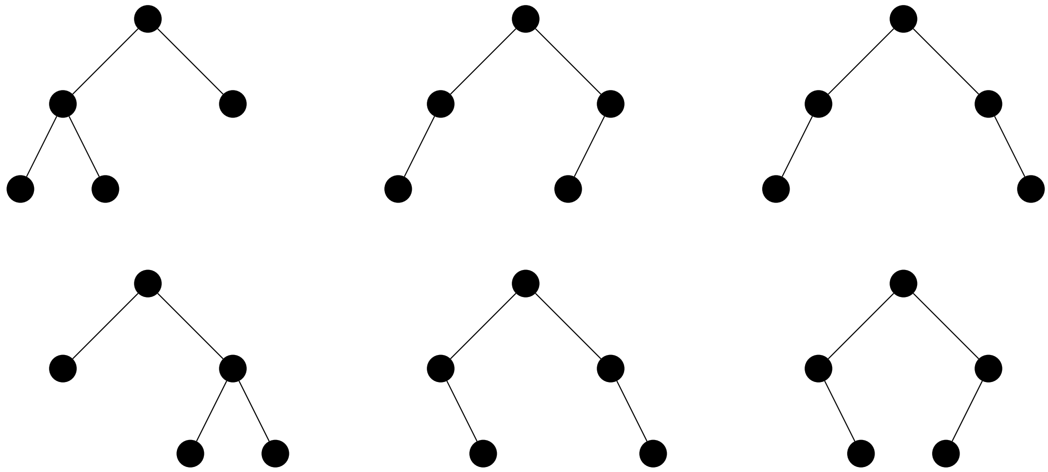

Figure 2 The six types of AVL trees with $n = 5$ nodes.

3. *Log-linear*: There is a constant $c > 0$ and a function $\vartheta(n) = o(n)$ such that

$$\log |\mathcal{T}_n| = cn + \vartheta(n). \tag{1}$$

4. *B-heavy twigged*: If v is a node of any $\tau \in \mathcal{T}$ with $|\tau[v]| \geq B(n)$, and $\tau_\ell[v]$ and $\tau_r[v]$ are the left and right subtrees of v in τ, then $|\tau_\ell[v]|, |\tau_r[v]| = \omega(1)$.

We present a new construction that gives a succinct encoding for all classes of trees satisfying only the first three conditions. By using constant time rank and select operations already supported by a succinct encoding for binary trees, we can also eliminate the use of so-called "portal nodes" and thus relax the second condition to the following.

2′. *Worst-case weakly fringe dominated:* Most nodes in members of $\mathcal{T}$ do not generate large fringe subtrees, in the sense that there is a $B'(n)$ satisfying $B'(n) = d \log n + o(\log n)$ for some $d < 1$ such that

$$\Big|\{v \in \tau : \big|\tau[v]\big| \;\geq\; B'(n)\}\Big| = o(n) \tag{2}$$

for every binary tree $\tau \in \mathcal{T}_n$.

Adopting terminology similar to that of [9], we call a class of binary trees *weakly tame* if it is fringe-hereditary, worst-case weakly fringe dominated, and log-linear.

▶ **Theorem 1.** *If $\mathcal{T}$ is a weakly tame class of binary trees then there exists a succinct encoding for $\mathcal{T}$ that supports the operations in Table 1 in $O(1)$ time using the $(\log n)$-bit word RAM model.*

▶ Remark 2. We support operations on static trees, leaving extensions to trees with updates (such as in [10]) to future work.

Proof. See Section 2. ◀

▶ **Corollary 3.** *There exists a succinct encoding for AVL trees that supports the operations in Table 1 in $O(1)$ time using the $(\log n)$-bit word RAM model.*

Proof. AVL trees are weakly tame (see [9, Example F.2]) so the result follows immediately from Theorem 1. ◀

▶ Remark 4. In [9] the log-linearity of AVL trees is inferred from the stated exponential growth of a_n in Odlyzko [13]. This growth is proven in Theorem 6 below.

A minor modification of the arguments in [9] show that *Left-Leaning AVL (LLAVL) Trees*, which are AVL trees with the added restriction that at every node the height of the left subtree is at least the height of the right subtree, are also weakly tame, giving the following.

▶ **Corollary 5.** *There exists a succinct encoding for LLAVL trees that supports the operations in Table 1 in* $O(1)$ *time using the* $(\log n)$*-bit word RAM model.*

Table 1 Operations discussed in [5, 9] which can be done in $O(1)$ time in the $(\log n)$-bit word RAM model in a succinct encoding of a binary tree.

`parent`(v)	the parent of v, same as `anc`$(v, 1)$
`degree`(v)	the number of children of v
`left_child`(v)	the left child of node v
`right_child`(v)	the right child of node v
`depth`(v)	the depth of v, i.e., the number of edges between the root and v
`anc`(v, i)	the ancestor of node v at depth `depth`$(v) - i$
`nbdesc`(v)	the number of descendants of v
`height`(v)	the height of the subtree rooted at node v
`LCA`(v, u)	the lowest common ancestor of nodes u and v
`leftmost_leaf`(v)	the leftmost leaf descendant of v
`rightmost_leaf`(v)	the rightmost leaf descendant of v
`level_leftmost`(ℓ)	the leftmost node on level ℓ
`level_rightmost`(ℓ)	the rightmost node on level ℓ
`level_pred`(v)	the node immediately to the left of v on the same level
`level_succ`(v)	the node immediately to the right of v on the same level
`node_rank`$_X(v)$	the position of v in the X-order, $X \in \{$`PRE, POST, IN`$\}$, i.e., in a preorder, postorder, or inorder traversal of the tree
`node_select`$_X(i)$	the ith node in the X-order, $X \in \{$`PRE, POST, IN`$\}$
`leaf_rank`(v)	the number of leaves before and including v in preorder
`leaf_select`(i)	the ith leaf in preorder

To characterize how much space is required by a succinct encoding, we derive an asymptotic bound on the number of AVL trees using techniques from *analytic combinatorics* [4, 7]. To this end, let $a_n = |\mathcal{A}_n|$ be the counting sequence of $\mathcal{A}$ and let $A(z) = \sum_{n \geq 0} a_n z^n$ be its associated generating function. The key to enumerating AVL trees is to let $A_h(z)$ be the generating function for the subclass of AVL trees with *height* h. The balance condition on subtrees implies that an AVL tree of height $h + 2$ is a root together with a subtree of height $h + 1$ and a subtree of height either $h + 1$ or h, giving rise to the recursive equation

$$A_{h+2}(z) = A_{h+1}(z)(A_{h+1}(z) + 2A_h(z)) \tag{3}$$

for all $h \geq 0$, where the factor of 2 indicates that the shorter subtree can be on the left or right side. This recursion, along with the initial conditions $A_0(z) = z$ (encoding the only AVL tree with height zero, which is a single vertex) and $A_1(z) = z^2$ (encoding the only AVL tree with height one, which is a root with two children) uniquely determines $A_h(z)$ for all h. Summing over all possible heights gives the generating function

$$A(z) = \sum_{h=0}^{\infty} A_h(z)$$

for AVL trees.

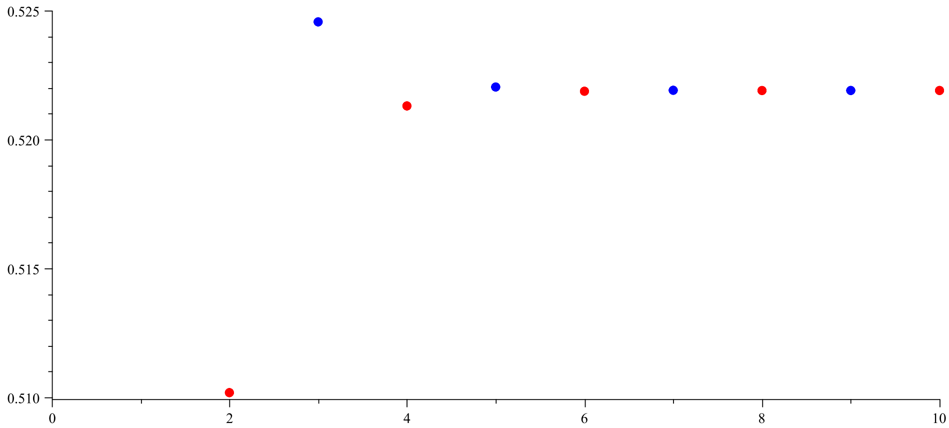

Figure 3 Values α_h converging to $\alpha = 0.5219\ldots$ monotonically from below among even h (red) and monotonically from above among odd h (blue).

Equation (3) implies that $A_h(z)$ is a non-constant polynomial with positive coefficients for all h, so the equation $A_h(z) = 1/3$ has a unique positive solution for all $h \in \mathbb{N}$ (see Figure 3 for values of these solutions). We prove the following.

▶ **Theorem 6.** *If α_h is the unique positive solution to $A_h(z) = 1/3$ then the limit*

$$\alpha = \lim_{h\to\infty} \alpha_h = 0.5219\ldots$$

exists. Furthermore,

$$\log_2(a_n) = \underbrace{n \log_2(\alpha^{-1})}_{n(0.938\ldots)} + \log\theta(n)$$

for a function θ growing at most sub-exponentially (meaning $\theta(n) = o(\kappa^n)$ for all $\kappa > 1$).

Proof. The result follows immediately from applying Theorem 13 below with $f(x_1, x_2) = x_1^2 + 2x_1x_2$, since the unique positive solution to $f(C, C) = C$ is $C = 1/3$. ◀

▶ Remark 7. A full proof of the claimed asymptotic behaviour $a_n \sim n^{-1}\alpha^{-n}u(\log n)$ in Odlyzko [13], which characterizes *sub-dominant* asymptotic terms for the bitsize, requires a more intense study of the recursion (3) and is outside the scope of this discussion. It is postponed to future work.

Our approach derives asymptotics for a family of generating functions satisfying recursive equations similar to (3). For instance, if $L_h(z)$ is the generating function for LLAVL trees with height h then

$$L_{h+2}(z) = L_{h+1}(z)(L_{h+1}(z) + L_h(z)) \qquad (4)$$

for all $h \geq 0$, as an LLAVL tree of height $h+2$ is a root together with a left subtree of height $h+1$ and a right subtree of height $h+1$ or h. Note that the only difference between this recurrence and the recursive equation (3) for AVL trees is the coefficient of $L_h(z)$, since there is now only one way to have an unbalanced pair of subtrees.

▶ **Theorem 8.** *If γ_h is the unique positive solution to $L_h(z) = 1/2$ then the limit*

$$\gamma = \lim_{h\to\infty} \gamma_h = 0.67418\ldots$$

is well-defined. Furthermore, the number ℓ_n of LLAVL trees on n nodes satisfies

$$\log_2(\ell_n) = \underbrace{n\log_2(\gamma^{-1})}_{n(0.568\ldots)} + \log\theta(n)$$

for a function θ growing at most sub-exponentially.

Proof. The result follows by applying Theorem 13 below with $f(x_1, x_2) = x_1^2 + x_1x_2$, since the unique positive solution to $f(C, C) = C$ is $C = 1/2$. ◀

2 A New Succinct Encoding for Weakly Tame Classes

We now prove Theorem 1, first describing our encoding and then showing it has the stated properties.

2.1 Our encoding

Let $\mathcal{E}$ denote a succinct data structure representing all binary trees that supports the operations in Table 1, and denote the encoding of a binary tree τ in this data structure by $\mathcal{E}(\tau)$. We now fix a weakly tame class of binary trees $\mathcal{T}$ and, given a binary tree $\tau \in \mathcal{T}$ of size n, define the *upper tree*

$$\tau' = \Big\{v \in \tau : \big|\tau[p(v)]\big| \geq d\log n\Big\}$$

where $p(v)$ denotes the parent of a vertex v in the tree τ and d is a constant such that $B'(n) = d\log n + o(\log n)$ satisfies (2) in the definition of worst-case weakly fringe dominated.

Our succinct data structure for $\mathcal{T}$ is constructed as follows.

1. We simply copy the encodings $\mathcal{E}(\tau')$ for upper trees.
2. For every $1 \leq j < d\log n$ we write down a lookup table mapping the trees in $\mathcal{T}_j$ (with j nodes) to their corresponding $\mathcal{E}$ encoding. We can do this, for example, by enumerating the $\mathcal{T}_j$ in lexicographic order by the $\mathcal{E}$ encoding using integers of bitsize $\log|\mathcal{T}_j| = cj + o(j)$, where c is the constant in the definition of log-linearity (1).
3. For each leaf node $\ell \in \tau'$ the fringe subtree $\tau[\ell]$ has size $|\tau[\ell]| < d\log n$ by definition of τ'. We call these trees *lower trees*, and write them down using their encoding in a lookup table in `leaf_rank` order of their roots in τ', storing the root locations in an indexable dictionary.
4. Lastly, we store additional information in (fully) indexable dictionaries to support operations like `node_rank/select`, `level_succ/pred`, and `leaf_rank/select`. For instance, for `node_rank/select` we store a fully indexable dictionary that maps the `node_rank` for a node in τ' to the `node_rank` of the node in τ. The techniques to support the other operations are similar, and are analogous to constructions used in [5, 3].

2.2 Proof of Size and Operation Time Bounds

Navigation through the upper tree follows standard navigation using $\mathcal{E}$, which supports the desired operations in constant time. When a leaf node ℓ is reached in the upper tree, the operation $x =$ `leaf_rank`(ℓ) gives the index of the child tree in the indexable dictionary.

Then the operation `select`(x) gives the location of the string encoding the child tree. Finally, using the table mapping our encoding to the $\mathcal{E}$ encoding gives us the ability to perform all the navigation operations on the smaller tree. In order to perform the lookup using the mapping, it is necessary to know the size of the tree. This can be inferred from the space in memory allocated to the naming, which can be calculated by the operation `select`$(x+1)$ in the indexable dictionary to find the starting location of the next child tree. To navigate back to the upper tree from a child tree, we use the reverse operations of $y = \texttt{rank}(x)$ in the indexable dictionary followed by `select_leaf`(y) in the upper tree.

To get the `node_rank` of a node in τ' we use the fully indexable dictionary, and to get the `node_rank` of a node not in τ' we simply get the `node_rank` of the root of the child tree and the `node_rank` of the node within the child tree and perform the appropriate arithmetic depending on the desired rank order (`pre, post, in`). For `node_select`, if the node is in τ' then selecting using the indexable dictionary is sufficient. Otherwise, the node is in a child tree and the initial `node_select` will return the predecessor node in τ' which will be the root of the child tree when using `preorder` (the argument is similar for `postorder` and `inorder`). Using the rank of this root and appropriate arithmetic, we can then select the desired node in the child tree. Implementing the other operations is analogous. It is clear that all of these operations are supported in constant time, since they involve a constant number of calls to the constant-time operations in the existing data structures, and lookups using $(\log n)$-bit words.

Space Complexity

The space used by $\mathcal{E}(\tau')$ is $o(n)$ by the weakly tame property. The space used by the lookup tables is $O(n^d \log n) = o(n)$ by definition of τ' and d, and the space used by all of the encodings of the child trees is $cn + o(n)$ by log-linearity. Lastly, the space needed for the indexable dictionaries is $o(n)$ for each [3, Lemmas 1 and 2]. Summing these requirements shows that the total storage required is $cn + o(n)$ many bits, so the encoding is succinct. ◀

3 Asymptotics for a Family of Recursions

We derive the asymptotic behaviour of a family of generating functions which includes Theorem 6 as a special case. Let $\mathcal{F}$ be a combinatorial class decomposed into a disjoint union of finite subclasses $\mathcal{F} = \bigsqcup_{h=0}^{\infty} \mathcal{F}_h$ whose generating functions $F_h(z)$ are non-constant and satisfy a recursion

$$F_h(z) = f(F_{h-1}(z), F_{h-2}(z), \ldots, F_{h-c}(z)) \quad \text{for all} \quad h \geq c, \tag{5}$$

where c is a positive integer and f is a multivariate polynomial with non-negative coefficients.

▶ Remark 9. The elements of $\mathcal{F}_h$ are usually *not* the objects of $\mathcal{F}$ of size h (in our tree applications they contain trees of height h, not trees with h nodes). The fact that each $\mathcal{F}_h$ is finite implies that the $F_h(z)$ are polynomials with non-negative coefficients. The coefficient of z^n in $F_h(z)$ counts the number of objects of size n within the subclass indexed by h.

We assume that there exists a (necessarily unique) positive real solution C to the equation $C = f(C, C, \ldots, C)$, and for each $h \geq 0$ we let α_h be the unique positive real solution to $F_h(z) = C$. In order to rule out degenerate cases and cases where the counting sequence has periodic behaviour, we need another definition.

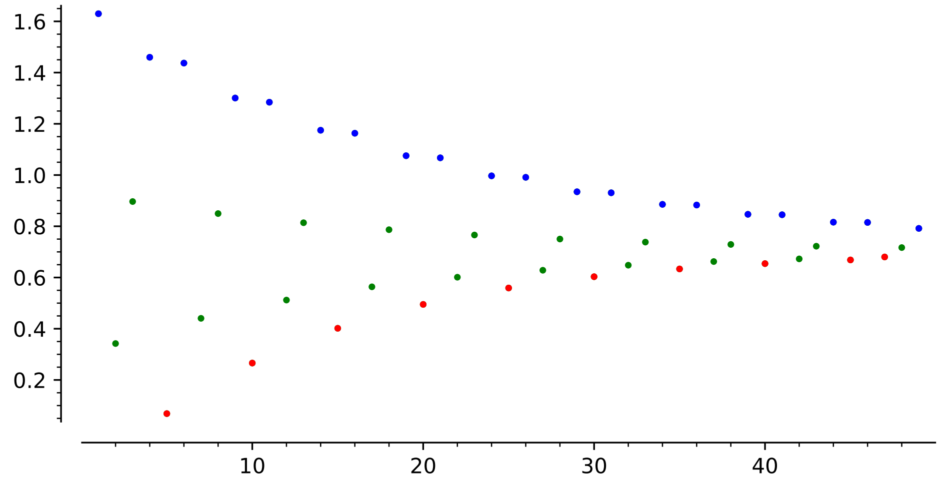

Figure 4 Values α_i converging with u_is shown in blue and ℓ_js shown in red.

▶ **Definition 10** (recursive-dependent)**.** *We call the polynomial f* recursive-dependent *if there exists a constant k (depending only on f) such that for any indices $i, j \geq c$ with $i \geq j + k$ there exists a sequence of applications of the recurrence* (5) *resulting in a polynomial P with $F_i = P(F_{\ell_1}, \dots, F_{\ell_m})$ for some $0 \leq \ell_1 < \cdots < \ell_m \leq i$ where $\frac{\partial P}{\partial F_j} \neq 0$.*

▶ **Example 11.** The polynomial $f(x, y) = y$ is not recursive-dependent because it leads to the recursion $F_h(z) = F_{h-2}(z)$, meaning that the values of F_h when h is even can be independent of those where h is odd.

▶ **Lemma 12.** *If f is recursive-dependent with non-negative coefficients and a positive fixed point then the limit $\alpha = \lim_{h\to\infty} \alpha_h$ exists.*

Proof. We start by defining two subsequences of α_h to give upper and lower bounds on its limit, then prove that these are equal. First, we let

- u_0 be the smallest index $j \in \{0, \dots, c-1\}$ such that $\alpha_j = \max\{\alpha_0, \dots, \alpha_{c-1}\}$

and for all $i \geq 0$ let

- u_{i+1} be the smallest index $j \in \{u_i + 1, \dots, u_i + c\}$ such that $\alpha_j = \max\{\alpha_{u_i+1}, \dots, \alpha_{u_i+c}\}$,

so that the u_i denote the indices of the maximum values of the α_h as h ranges over intervals of size at most c. Conversely, we let

- ℓ_0 be the index $j \in \{0, \dots, c-1\}$ such that $\alpha_j = \min\{\alpha_0, \dots, \alpha_{c-1}\}$

and for all $j \geq 0$ let

- ℓ_{i+1} be the index $j \in \{u_i + 1, \dots, u_i + c\}$ such that $\alpha_j = \min\{\alpha_{u_i+1}, \dots, \alpha_{u_i+c}\}$,

so that the ℓ_j denote the indices of the minimum values of the α_h as h ranges over intervals of size at most c.

We claim that the subsequence α_{u_i} is non-increasing. To establish this, we fix $i \geq 1$ and consider α_{u_i}. By definition, $\alpha_{u_i} \geq \alpha_{u_j}$ for all $j \in \{u_{i-1} + 1, \dots, u_{i-1} + c\}$. Thus, if $u_{i+1} \in \{u_{i-1} + 1, \dots, u_{i-1} + c\}$ then $\alpha_{u_i} \geq \alpha_{u_{i+1}}$ as claimed. If, on the other hand, $u_{i+1} > u_{i-1} + c$ then repeated application of the recursion (5) implies

$$\begin{aligned} F_{u_{i+1}}(\alpha_{u_i}) &= f\Big(F_{u_{i+1}-1}(\alpha_{u_i}), \dots, F_{u_{i+1}-c}(\alpha_{u_i})\Big) \\ &\ \vdots \\ &= Q\Big(F_{u_{i-1}+1}(\alpha_{u_i}), \dots, F_{u_{i-1}+c}(\alpha_{u_i})\Big), \end{aligned}$$

where Q is a multivariate polynomial with non-negative coefficients such that $Q(C,\dots,C) = C$. All the F_h are monotonically increasing as non-constant polynomials with non-negative coefficients, so $F_j(\alpha_{u_i}) \geq F_j(\alpha_{u_j}) = C$ for all $j \in \{u_{i-1}+1,\dots,u_{i-1}+c\}$ and

$$F_{u_{i+1}}(\alpha_{u_i}) \geq Q\Big(C,\dots,C\Big) = C.$$

Since $F_{u_{i+1}}$ is monotonically increasing and $F_{u_{i+1}}(\alpha_{u_{i+1}}) = C$, we once again see that $\alpha_{u_i} \geq \alpha_{u_{i+1}}$. As i was arbitrary, we have proven that α_{u_i} is non-increasing. The same argument, reversing inequalities, proves that the subsequence α_{ℓ_j} is non-decreasing.

As α_{ℓ_j} is non-decreasing and α_{u_i} is non-increasing, either $\alpha_{\ell_j} \leq \alpha_{u_i}$ for all $i, j \geq 0$ or $\alpha_{\ell_j} > \alpha_{u_i}$ for all sufficiently large i and j. The second case implies the existence of indices $a, b > 0$ such that $\alpha_{\ell_b} > \alpha_{u_a}$ but $\ell_b \in \{u_{a-1}+1,\dots,u_{a-1}+c\}$ so that u_a is not the maximum index of α_j in this range, giving a contradiction. Thus, $\alpha_{\ell_j} \leq \alpha_{u_i}$ for all $i, j \geq 0$ and the limits

$$\alpha_u = \lim_{i\to\infty} \alpha_{u_i} \quad \text{and} \quad \alpha_\ell = \lim_{j\to\infty} \alpha_{\ell_j}$$

exist. To prove that the limit of α_h exists as $h \to \infty$, it is now sufficient to prove that $\alpha_u = \alpha_\ell$.

Suppose toward contradiction that $\alpha_u \neq \alpha_\ell$, and define $a = \alpha_u - \alpha_\ell > 0$. For any $\epsilon > 0$, we pick i, j, k sufficiently large so that $\ell_j > u_i > \ell_k + c$ and $|\alpha_{u_i} - \alpha_u|, |\alpha_{\ell_j} - \alpha_\ell|, |\alpha_{\ell_k} - \alpha_\ell| < \epsilon$. Then by recursive-dependence we can recursively decompose F_{ℓ_j} in terms of F_{u_i}, and possibly some other terms $F_{h_1},\dots,F_{h_r}$ where each $|h_n - u_i| \leq c$, to get

$$C = F_{\ell_j}(\alpha_{\ell_j}) = P(F_{u_i}(\alpha_{\ell_j}), F_{h_1}(\alpha_{\ell_j}),\dots,F_{h_r}(\alpha_{\ell_j}))$$

where $P(F_{u_i}, F_{h_1},\dots,F_{h_r})$ is a polynomial with non-negative coefficients that depends on F_{u_i} and satisfies $P(C,\dots,C) = C$. Because P is monotonically increasing in each coordinate, and $\alpha_{\ell_k} + \epsilon > \alpha_\ell \geq \alpha_{\ell_j}$, we see that

$$C \leq P(F_{u_i}(\alpha_{\ell_k}+\epsilon), F_{h_1}(\alpha_{\ell_k}+\epsilon),\dots,F_{h_r}(\alpha_{\ell_k}+\epsilon)).$$

Furthermore, each $\alpha_{h_n} \geq \alpha_{\ell_k}$ so

$$\begin{aligned} C &\leq P(F_{u_i}(\alpha_{\ell_k}+\epsilon), F_{h_1}(\alpha_{h_1}+\epsilon),\dots,F_{h_r}(\alpha_{h_r}+\epsilon)) \\ &\leq P(F_{u_i}(\alpha_{\ell_k}+\epsilon), C+\mathrm{poly}(\epsilon),\dots,C+\mathrm{poly}(\epsilon)). \end{aligned}$$

Finally, $\alpha_{u_i} - a \geq \alpha_{\ell_k}$ so

$$C \leq P(F_{u_i}(\alpha_{u_i} - a + \epsilon), C+\mathrm{poly}(\epsilon),\dots,C+\mathrm{poly}(\epsilon)).$$

Because a is fixed, P is monotonically increasing in each variable, and $F_{u_i}(\alpha_{u_i}) = C$, taking $\epsilon \to 0$ shows that the right-hand side of this last inequality is strictly less than $P(C,\dots,C) = C$, a contradiction. Thus, $a = 0$ and the limit $\alpha = \alpha_u = \alpha_\ell$ exists. ◀

▶ **Theorem 13.** *If f is recursive-dependent with non-negative coefficients and a positive fixed point, then the number a_n of objects in $\mathcal{F}$ of size n satisfies*

$$a_n = \alpha^{-n}\,\theta(n),$$

where α is the limit described in Lemma 12 and $\theta(n)$ is a function growing at most sub-exponentially.

Proof. We prove that the generating function $F(z)$ is analytic for $|z| < \alpha$ by showing that the series $\sum_{h=0}^{\infty} F_h(z)$ converges for these values of z. Because $|F(z)| \to \infty$ as $z \to \alpha$, the point $z = \alpha$ is then a singularity of $F(z)$ of smallest modulus, and thus (by the root test for series convergence) the reciprocal of the exponential growth of a_n.

First, assume that there exists some $k \geq 0$ and $0 < \lambda < 1$ such that $F_h(z) < \lambda C$ for every $h \in \{k, k+1, \ldots, k+c-1\}$. Let A be the sum of the coefficients of all degree 1 terms of f. Since f has non-negative coefficients and a positive real fixed point, we must have $A < 1$. Let $g(x_1, \ldots, x_c)$ be the function created by removing all degree one terms from f. Observe that $C = AC + g(C, \ldots, C)$, and thus $g(\lambda C, \ldots, \lambda C) \leq \lambda^2 g(C, \ldots, C) = \lambda^2(1-A)C$, so that

$$f(\lambda C, \ldots, \lambda C) \leq A\lambda C + \lambda^2(1-A)C.$$

Algebraic manipulation shows that $A\lambda + \lambda^2(1-A) \leq \lambda$, and since f has non-negative coefficients we can conclude that for every $h \in \{k+c, k+1+c, \ldots, k+2c-1\}$ we have $F_h(z) \leq A\lambda C + \lambda^2(1-A)C$. Let $\lambda_0 = \lambda$ and define $\lambda_i = \lambda_{i-1}(A + \lambda_{i-1} - A\lambda_{i-1})$ for all $i \geq 1$. By the above argument we have

$$F_{ch+k}(z) \leq \lambda_h C,$$

so it remains to show that $\sum_{i=0}^{\infty} \lambda_i$ converges. We will show that $\lambda_i \leq \lambda(A + \lambda - A\lambda)^i$ by induction on i. The result holds by definition for $i = 1$. If the result holds for some $j \geq 1$ then

$$\begin{aligned}\lambda_{j+1} &= \lambda_j(A + \lambda_j - A\lambda_j) \\ &\leq \lambda(A + \lambda - A\lambda)^j(A + \lambda_j - A\lambda_j) \\ &\leq \lambda(A + \lambda - A\lambda)^{j+1},\end{aligned}$$

where the last inequality follows from the fact that $\lambda_j < \lambda$ since $A + \lambda - A\lambda < 1$. The sum $\sum_{i=0}^{\infty} \lambda(A + \lambda - A\lambda)^i$ converges as a geometric series, and thus $\sum_{h=0}^{\infty} F_h(z)$ converges.

It remains to show that if $|z| < \alpha$ then such a k and λ exist. For any $|z| < \alpha$ there is some N sufficiently large such $|z| < \alpha_n$ for all $n \geq N$. By the definition of α_n, and since the coefficients of F_n are all positive, we must have $F_n(z) < C$. Hence $F_n(z) < \lambda_n C$ for some $0 < \lambda_n < 1$. Taking $k = N$ and letting λ be the largest λ_n for $n \in \{N, N+1, \ldots, N+c-1\}$ proves our final claim. ◀

References

1 G. Adelson-Velsky and E. Landis. An algorithm for the organization of information. In *Proceedings of the USSR Academy of Sciences (in Russian)*, 1962.

2 Clark, David. *Compact PAT trees.* PhD thesis, University of Waterloo, 1997. URL: `http://hdl.handle.net/10012/64`.

3 Arash Farzan and J. Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, January 2014. `doi:10.1007/s00453-012-9664-0`.

4 Philippe Flajolet and Robert Sedgewick. *Analytic combinatorics.* Cambridge University Press, Cambridge, 2009.

5 Meng He, J. Ian Munro, and S. Srinivasa Rao. Succinct ordinal trees based on tree covering. In *Automata, Languages and Programming*, pages 509–520. 2007.

6 G. Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science*, pages 549–554, 1989.

7 Stephen Melczer. *An Invitation to Analytic Combinatorics: From One to Several Variables.* Texts and Monographs in Symbolic Computation. Springer International Publishing, 2021.

8 J. Ian Munro. Tables. In Vijay Chandru and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science, 16th Conference, Hyderabad, India, December 18-20, 1996, Proceedings*, volume 1180 of *Lecture Notes in Computer Science*, pages 37–42. Springer, 1996. doi:10.1007/3-540-62034-6_35.

9 J. Ian Munro, Patrick K. Nicholson, Louisa Seelbach Benkner, and Sebastian Wild. Hypersuccinct trees – new universal tree source codes for optimal compressed tree data structures and range minima. In *29th Annual European Symposium on Algorithm*, pages 70:1–70:18, 2021.

10 J. Ian Munro, Venkatesh Raman, and Adam J. Storm. Representing dynamic binary trees succinctly. In S. Rao Kosaraju, editor, *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*, pages 529–536. ACM/SIAM, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365526.

11 J. Ian Munro and S. Srinivasa Rao. Succinct representation of data structures. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035179.ch37.

12 Gonzalo Navarro. *Compact Data Structures: A Practical Approach*. Cambridge University Press, 2016.

13 A. M. Odlyzko. Some new methods and results in tree enumeration. In *Proceedings of the thirteenth Manitoba conference on numerical mathematics and computing (Winnipeg, Man., 1983)*, volume 42, pages 27–52, 1984.