Enumerating, Ranking and Unranking Binary Trees

J. M. PALLO

Département d'Informatique, Université de Dijon, BP 138, F21004 Dijon, France

We introduce weight sequences, which are sequences of positive integers characterising binary trees, in order to generate lexicographically binary trees as a list. Algorithms are developed to determine the position of a given weight sequence, and to generate the weight sequence of a given position. We also use weight sequences in order to study a transformation on binary trees. Furthermore, we make a link with term rewriting systems.

Received January 1984

1. INTRODUCTION

The set B of (ordered, oriented) binary trees is recursively

defined by the equation: $B = \Box + BB$, where O is an internal node and \Box is an external node. The size |T| of a tree T is the number of internal nodes. The weight \tilde{T} of a tree T is the number of external nodes: $\tilde{T} = |T| + 1$. The product of T and T' is the tree which we denote

by T T', the root of which admits T as left subtree and T' as right subtree. The external nodes of a tree T are numbered by a preorder traversal of T (visit the root, then traverse its subtrees from left to right). If B_n is the set of binary trees with n internal nodes then $b_n = \operatorname{card} B_n = \frac{1}{n+1} \binom{2n}{n}$: these are the well-known

This paper introduces a method of representing a binary tree of n internal nodes as a sequence of n integers, such that a bijection is induced between B_n and the set of these n-integer sequences. More precisely, given a binary tree T, each external node is the rightmost external node of some subtrees of T. Let w_i be the weight of the largest subtree which has external node i in the rightmost position. Then the sequence $(w_1, w_2, ..., w_n)$ characterises T. Such a sequence is called a weight sequence.

These sequences make it possible to generate all binary trees lexicographically with a most efficient algorithm, because they use a *global* information concerning their external nodes. Since Knott's paper⁶ was published, several papers have appeared dealing with the ordering and ranking of binary trees. Typically, the ordering involves showing the existence of a bijection between binary trees and integer sequences or permutations. 'Tree permutations' are used by Knott, ⁶ 'feasible sequences' by Ruskey and Hu, ¹² 'ballot sequences' by Rotem and Varol, ¹¹ and 'z-sequences' by Zaks. ¹⁴ Relations to existing algorithms are discussed. In Section 3 the ranking and unranking procedures are provided.

In Section 4 we study a transformation on B which is naturally linked with associativity and which always moves the internal nodes of a tree in the same direction. This transformation, called 'single left rotation' by Knuth,8 is used to restore balance when a new node has been inserted into a balanced tree. In Culik and Wood³ it is used to provide a metric on binary trees. In Lalonde and Des Rivières,¹0 it is called 'left-subordinate transformation' and used to transform a standardised tree representation of an arithmetic expression lacking operator precedence information into a tree containing it.

It can be applied after the parsing phase. It is also called 'A-transformation' in Bonnin and Pallo² in their own investigations of n-ary trees. Weight sequences provide a combinatorial characterization of this transformation, which induces a lattice structure on B_n . As regards term rewriting systems (Section 5), weight sequences are more suitable to handle this transformation on arithmetic expressions in infix notation.

2. THE WEIGHT SEQUENCES

Definition

Given a binary tree T, the weight sequence of T is the integer sequence $(w_T(1), w_T(2), ..., w_T(|T|))$, where $w_T(i)$ is the weight of the largest subtree of T, the last external node of which is the external node i of T.

If T =

then $w_T = (1, 2, 1, 1, 5, 1, 1, 3)$.

If the external node i is the first external node of a subtree of $T: w_T(i) = 1$.

If the external node i is the last external node of a subtree of $T: w_T(i) \ge 2$. For all $i: 1 \le w_T(i) \le i$.

Lemma 1

We can compute the weight sequence of the product T'' = T by using the weight sequences of the trees T and T':

$$w_{T''} = (w_T(1), ..., w_T(|T|), \tilde{T}, w_{T'}(1), ..., w_{T'}(|T'|))$$

Theorem 1

An integer sequence $(w_1, w_2, ..., w_n)$ is the weight sequence of a binary tree with n internal nodes iff for all i:

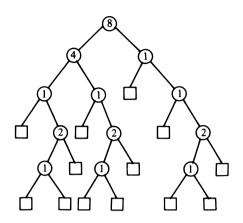
$$-1 \leqslant w_i \leqslant i$$
-for all $i' \in [i-w_i+1, i]: i-w_i \leqslant i'-w_{i'}.$

Proof

By induction on n by using Lemma 1.

The weight sequence of a binary tree can be obtained by an inorder traversal (traverse the left subtree, visit the root, traverse the right subtree) after labelling each internal node with the weight of its left subtree.

If T =



then
$$w_T = (1, 1, 2, 4, 1, 1, 2, 8, 1, 1, 1, 2)$$

The following algorithm enables us, starting from a weight sequence, to compute the next one in the lexicographic order. In order to obtain all the n-weight sequences, we need only begin with (1, 1, ..., 1).

Algorithm TREE GENERATION

Begin with $(w_1, w_2, ..., w_n) = (1, 1, ..., 1)$ While $i = max \{k | w_k < k\}$ exists do: $j \leftarrow i - w_i$ $w_i \leftarrow i - j + w_j$ For m = i + 1 to n do $w_m \leftarrow 1$ enddo.

List of the weight sequences of the 42 binary trees with five internal nodes in lexicographic order

nve meema i	iodes ili iexicogi	apine order.					
11111	11211	12111					
11112	11212	12112					
11113	11214	12113					
11114	11215	12115					
11115	11231	12121					
11121	11234	12123					
11123	11235	12125					
11124	11241	12141					
11125	11245	12145					
11131	11311	12311					
11134	11312	12312					
11135	11315	12315					
11141	11341	12341					
11145	11345	12345					
In order to generate this list, the algorithm ha							

In order to generate this list, the algorithm had to use 64 comparisons of integers, i.e. an average number of comparisons equal to 64/42 = 1.52.

The work made by the previous algorithm is proportional to the number of comparisons made with a view to finding $i = \max\{k \mid w_k < k\}$. Let $t_{n,j}$ denote the number of *n*-weight sequences that require *j* comparisons in the generating algorithm, i.e. the number of n-weight sequences w such that $w = (w_1, ..., w_{n-j+1}, n-j+2, ..., n)$ and $w_{n-j+1} < n-j+1$. Therefore $t_{n,j}$ is also the number of (n-j+1) weight sequences which do not end in n-j+1. The number of (n-j+1) weight sequences

ending in n-j+1 being equal to b_{n-j} , we deduce that $t_{n,j}=b_{n-j+1}-b_{n-j}$. The total number of comparisons in order to generate the *n*-weight sequences is

$$\sum_{j=1}^{n} j t_{n,j} = \sum_{j=1}^{n} j (b_{n-j+1} - b_{n-j}) = \sum_{k=1}^{n} b_{k}.$$

The average number of comparisons per sequence is equal

$$\frac{1}{b_n} \sum_{k=1}^n b_k$$
, which tends to $4/3 = 1.33$ as *n* increases.

This algorithm is the most efficient one to generate lexicographically all the n-node binary trees by making use of *n*-integer sequences such that $1 \le s_i \le n$ for all *i*. Indeed, the average number of comparisons per n-feasible sequences¹² and per n-z sequences¹⁴ tends respectively to 3 and 4/3 as *n* increases. However, n-z sequences verify $1 \le z_i \le 2n$ for all i.

3. RANKING AND UNRANKING

Definition

The height $h(1 \le h \le n)$ of an *n*-weight sequence w is the integer $h = \max\{i \mid w_i = 1\}$. Let $\sigma_{n,h}$ denote the number of n-weight sequences of height h.

Lemma 2

To obtain the trees corresponding to the *n*-weight

sequences of height h, we just have to substitute \Box for the external node h of trees corresponding to the (n-1) weight sequences of height $\geq h-1$.

Corollary

We have

$$\sigma_{n,1} = b_{n-1}, \, \sigma_{n,n} = 1, \, \sigma_{n,h} = \sum_{j=h-1}^{n-1} \sigma_{n-1,j} \quad \text{for} \quad 2 \leqslant h \leqslant n-1.$$

These recurrence relations allow us to build the table:

		1	2	3	4	5	6	7	8
	1	1							
	2	1	1						
	3	2	2	1					
	4	5	5	3	1				
	5	14	14	9	4	1			
	6	42	42	28	14	5	1		
•	7	132	132	90	48	20	6	1	
;	3	429	429	297	165	75	27	7	1

Explicitly, we have

$$\sigma_{n,h} = (h/2n - h) {2n - h \choose n - h}$$
 for $h \ge 2$.

Let rank $(w_1, w_2, ..., w_n)$ denote the position of the *n*-weight sequence $(w_1, w_2, ..., w_n)$ in the lexicographic ordering of all these *n*-weight sequences.

Theorem

Given an n-weight sequence w, we have the relation:

rank
$$(w_1, ..., w_n) = \sum_{j=n+1}^n \sigma_{n,j} + \text{rank}(\overline{w}_1, ..., \overline{w}_{n-1}),$$

where h is the height of w

 \overline{w} is the (n-1) weight sequence obtained from w by the following:

$$\begin{array}{ll} \text{for} & 1 \leqslant j \leqslant h-1 \colon \ \overline{w}_j = w_j = 1 \\ \text{for} & h \leqslant j \leqslant n-1 \colon \begin{cases} \overline{w}_j = w_{j+1} - 1 & \text{if} \quad j - w_j + 1 \leqslant h \\ \overline{w}_j = w_{j+1} & \text{otherwise}. \end{cases}$$

The previous theorem enables us to compute the rank of a weight sequence as follows:

if
$$T =$$

then
$$w_T = (1, 2, 3, 1, 1, 6)$$

rank $(1, 2, 3, 1, 1, 6) = \sum_{j=2}^{6} \sigma_{6,j} + \text{rank } (1, 2, 1, 1, 5)$
rank $(1, 2, 1, 1, 5) = \sum_{j=2}^{5} \sigma_{5,j} + \text{rank } (1, 1, 1, 1, 4)$
rank $(1, 1, 1, 1, 4) = \sum_{j=4}^{3} \sigma_{4,j} + \text{rank } (1, 1, 3)$
rank $(1, 1, 3) = \sum_{j=3}^{3} \sigma_{3,j} + \text{rank } (1, 2)$
rank $(1, 2) = \sum_{j=2}^{2} \sigma_{2,j} + \text{rank } (1)$
Thus rank $(1, 2, 3, 1, 1, 6) = (42 + 28 + 14 + 5 + 1) + (14 + 9 + 4 + 1) + 1 + 1 + 1 = 122$.

The following algorithm enables us to compute the *n*-weight sequence of rank r ($1 \le r \le b_n$):

Algorithm UNRANKING

For m = n - 1 to 1 step - 1 **do**: find the integer k_m such that

$$\sum_{j=k_m+1}^n \sigma_{m+1,\,j}+1\leqslant r\leqslant \sum_{j=k_m}^n \sigma_{m+1,\,j}$$

$$r\leftarrow r-\sum_{j=k_m+1}^n \sigma_{m+1,\,j}$$
 enddo.
$$w_1\leftarrow 1$$

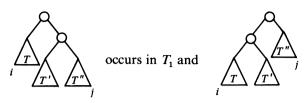
$$\begin{array}{lll} w_1 \leftarrow 1 \\ \textbf{For} & m=1 & \textbf{to} \\ \textbf{do} \colon \textbf{if} \ k_m = m+1 \\ & \textbf{then} w_{m+1} \leftarrow 1 \\ & \textbf{else for j} = \textbf{m to} \ k_m \ \textbf{step} -1 \\ & \textbf{do if} \ j - w_j + 1 \leqslant k_m \\ & \textbf{then} \ w_{j+1} \leftarrow w_j + 1 \\ & \textbf{else} \ w_{j+1} \leftarrow w_j \\ & \textbf{endif} \\ & \textbf{enddo} \\ & \textbf{endif} \\ & w_{k_m} \leftarrow 1 \\ & \textbf{enddo}. \end{array}$$

This algorithm makes it possible to generate random *n*-node binary trees by using a random integer $r \in [1, b_n]$.

4. THE LATTICE OF BINARY TREES

Definition

For two trees T_1 and T_2 of *n* nodes, $T_1 \rightarrow T_2$ if between external nodes i and j



occurs in T_2 , where T, T', T'' are subtrees of T_1 and T_2 . Let \rightarrow denote the reflexive transitive closure of \rightarrow .

For the trees of five external nodes, we get the diagram shown as Figure 1.

Theorem 2

Given two binary trees T and T' of n nodes, $T \stackrel{*}{\rightarrow} T'$ iff $w_T(i) \leq w_{T'}(i)$ for all $i \in [1, n]$.

Proof

We prove easily by induction on n that the condition is necessary. Conversely, let us assume that $w_T \leq w_{T'}$ (i.e. $w_T(i) \le w_{T'}(i)$ for all i) and $T \ne T'$. Let i be the smallest integer such that $w_T(i) < w_{T'}(i)$. Therefore $w_T(i) < i$. Let $j = i - w_T(i) + 1 \ge 2$ and $k = j - w_T(j-1)$. There is then in T a subtree T_1 accepting k as first external node and j-1 as last external node, as well as a subtree T_2 accepting j as first external node and i as last external node. Tincludes the subtree $p \ge 1$, which, through

application of the single left rotation → becomes the of the tree T^1 . subtree

Let *l* be the largest integer such that $j = l - w_T(l) + 1$. The weight sequence of T^1 is the same as the weight sequence of T except in l because $w_T(l) = l - j + 1 < w_T 1(l) = l - k + 1$. By definition of $i: w_T(j-1) = w_{T'}(j-1) = j-k$. Thus $w_T(i) < w_{T'}(i)$ implies $w_{T'}(i) \ge i-k+1$. By the hypothesis: $w_T(l) = l - j + 1 \le w_{T'}(l)$ thus $w_{T'}(l) \ge l - i + 1$. Since $i \in [l-w_{T'}(l)+1, l]$, by applying theorem 1: $l-w_{T'}(l) \le i-w_{T'}(i) \le k-1$. Then $l-k+1=w_{T^1}(l) \le w_{T'}(l)$ and $w_{T^1} \le w_{T'}$. We have built a tree T^1 such that $w_T \le w_{T^1} \le w_{T'}$ and $T \to T^1$. By repeating this process, we shall find a finite sequence of trees T^i such that $w_T \leq w_{T^1} \leq w_{T^2} \leq \ldots \leq w_{T^n} = w_{T'}$ and $T \to T^1 \to T^2 \to \dots \to T^n = T'$. By transitivity: $T \stackrel{T}{\Rightarrow} T'$. Q.E.D.

Corollary

 $\stackrel{*}{\Rightarrow}$ is a partial ordering of B.

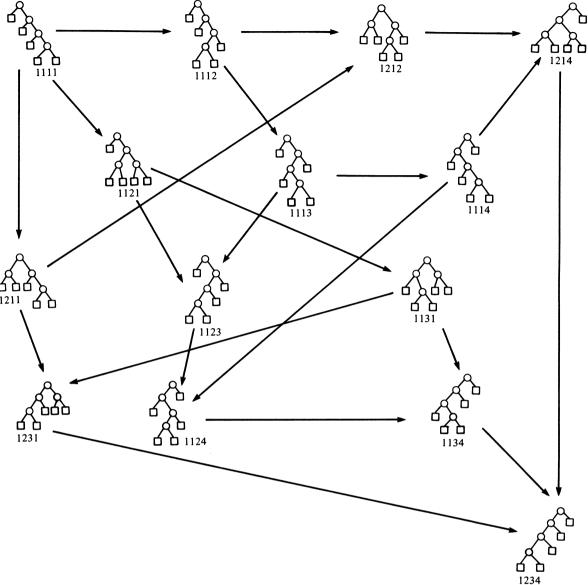


Figure 1

Proof

If $T \stackrel{*}{\Rightarrow} T'$ and $T' \stackrel{*}{\Rightarrow} T$, then $w_T \leqslant w_{T'}$ and $w_{T'} \leqslant w_T$ by theorem 2, thus $w_T = w_{T'}$. Therefore T = T' and $\stackrel{*}{\Rightarrow}$ is antisymmetric. Q.E.D.

Corollary

 $(B_n, \stackrel{*}{\rightarrow})$ is a lattice.

Proof

Let us prove that any pair of *n*-node binary trees T and T' has an infimum $T \wedge T'$. If $T'' \stackrel{*}{\Rightarrow} T$ and $T'' \stackrel{*}{\Rightarrow} T'$, then $w_{T''}(i) \leqslant \inf(w_T(i), w_{T'}(i))$ for all i. We just have to show that $w_i = \inf(w_T(i), w_{T'}(i))$ is a weight sequence, i.e. verifies conditions of Theorem 1. It is obvious that $1 \leqslant w_i \leqslant i$. If $i' \in [i - w_i + 1, i]$ then $i - w_{T'}(i) \leqslant i' - w_{T'}(i')$ and $i - w_{T'}(i) \leqslant i' - w_{T'}(i')$ from which we deduce $i - w_i \leqslant i' - w_i'$. Therefore $w_{T \wedge T'}(i) = \inf(w_T(i), w_{T'}(i))$ for all i.

Now let us define recursively the dual tree \overline{T} of a tree

$$T \text{ by } \Box = \Box \text{ and } \overrightarrow{T'} \overrightarrow{T''} = \overrightarrow{T''} \overrightarrow{T}'$$

Since $T \stackrel{*}{\Rightarrow} T'$ iff $\overline{T} \stackrel{*}{\Rightarrow} \overline{T}$, any pair of \underline{n} -node binary trees T and T' has a supremum $T \vee T' = \overline{T} \wedge \overline{T'}$. Remark that $w_{T \vee T'}(i) \neq \sup(w_T(i), w_{T'}(i))$ generally. Q.E.D.

5. APPLICATIONS TO TERM REWRITING SYSTEMS

The weight sequence of an arithmetic expression E in infix notation is obtained by scanning E from left to right and by associating to each operator the number of variable symbols of its left operand.

Let E = (((x+y)+(z+(y+t)))+((u+(v+x))+t)), an expression whose corresponding tree is the one given in Section 2. Then $w_E = (1, 2, 1, 1, 5, 1, 1, 3)$.

When associativity occurs in term rewriting systems, we orient the associativity axiom: $x.(y.z) \rightarrow (x.y).z$. Given two arithmetic expressions E and E' of same length, having the same and unique operator, and whose variables appear in the same order from left to right, Theorem 2 allows us to see immediately if $E \stackrel{*}{\Rightarrow} E'$ ($\stackrel{*}{\Rightarrow}$ is the rewriting relation), i.e. if one can go from E to E' by moving parentheses in the same direction. Furthermore, the proof of theorem 2 allows us to construct a path going from E to E' by the following algorithm.

Algorithm PATH CONSTRUCTION

while
$$i = min\{k \mid w_E(k) < w_{E'}(k)\}\ exists$$

do $j \leftarrow i - w_E(i) + 1\ k \leftarrow j - w_E(j-1)$
 $l = max\{m \ge i \mid j = m - w_E(m) + 1\}$
 $w_E(l) \leftarrow l - k + 1$

enddo.

This path has a minimal length which is bounded above by |E|.

Example

Let
$$E = (x + (y + ((z+s) + (t + (u + (v + q))))))$$
 and $E' = (((((x+y) + (z+s)) + t) + u) + (v + q))$ $w_E = (1, 1, 1, 2, 1, 1, 1)$ and $w_{E'} = (1, 2, 1, 4, 5, 6, 1)$. Consequently $E \stackrel{*}{\Rightarrow} E'$.

We construct below a minimal path joining E and E': E = (x + (y + ((z+s) + (t + (u + (v + q))))))

$$\begin{cases} s_E = (1, 1, 1, 2, 1, 1, 1) \\ s_{E'} = (1, 2, 1, 4, 5, 6, 1) \end{cases} i = 2 \quad j = 2 \quad k = 1 \quad l = 2$$

$$E^{1} = ((x+y)+((z+s)+(t+(u+(v+q)))))$$

$$\begin{cases} s_{E^{1}} = (1, 2, 1, 2, 1, 1, 1) \\ s_{E'} = (1, 2, 1, 4, 5, 6, 1) \end{cases} i = 4 \quad j = 3 \quad k = 1 \quad l = 4$$

$$E^{2} = (((x+y)+(z+s))+(t+(u+(v+q))))$$

$$\begin{cases} s_{E^{2}} = (1, 2, 1, 4, 1, 1, 1) \\ s_{E'} = (1, 2, 1, 4, 5, 6, 1) \end{cases} i = 5 \quad j = 5 \quad k = 1 \quad l = 5$$

REFERENCES

- 1. T. Beyer and S. M. Hedetniemi, Constant time generation of rooted trees. *SIAM Journal on Computing* 9 (4), 706-712 (1980).
- A. Bonnin and J. M. Pallo, A-Transformation dans les arbres n-aires. Discrete Mathematics 45, 153-163 (1983).
- 3. K. Culik and D. Wood, A note on some tree similarity measures. *Information Processing Letters* 15, 39-42 (1982).
- M. C. Er, A note on generating well-formed parenthesis strings lexicographically. The Computer Journal 26 (3), 205-207 (1983).
- G. Huet and D. Oppen, Equations and rewrite rules: a survey. i Formal Languages Theory: Perspectives and Open Problems, pp. 349-405. Academic Press, London (1980).
- G. D. Knott, A numbering system for binary trees. Comm. ACM 20 (2), 113-115 (1977).
- 7. D. E. Knuth, *The Art of Computer Programming*, vol. 1: Fundamental Algorithms. Addison-Wesley, London (1968).
- 8. D. E. Knuth, The Art of Computer Programming, vol. 3: Sorting and Searching. Addison-Wesley, London (1973).

$$E^{3} = ((((x+y)+(z+s))+t)+(u+v+q)))$$

$$\begin{cases} s_{E^{3}} = (1, 2, 1, 4, 5, 1, 1) \\ s_{E'} = (1, 2, 1, 4, 5, 6, 1) \end{cases}$$

$$i = 6 \quad j = 6 \quad k = 1 \quad l = 6$$

$$E' = (((((x+y)+(z+s))+t)+u)+(v+q))$$

If an arithmetic expression E has x as variable k and x^{-1} as variable k+1 with $w_E(k) = w_E(k+1) = 1$, and if we want to apply the rewrite rule $x \cdot x^{-1} \to e$, the following algorithm allows us to compute the weight sequence of an expression E' such that $E \stackrel{*}{\to} E'$ for the associative rule and $w_{E'}(k) = 1$, $w_{E'}(k+1) \ge 2$.

$$\begin{aligned} & \textbf{while} \ w_{k+1} = 1 \\ & \textbf{do} \ l = \max \left\{ m \geqslant k+1 \, | \, m-w_m = k \right\} \\ & \underbrace{w_l \leftarrow w_l + 1} \end{aligned}$$

enddo.

The same method is applicable if we want to apply the rewrite rule $(x.y).y^{-1} \rightarrow x$ on an expression E which has x as variable k, y as variable k+1, y^{-1} as variable k+2, $w_E(k) = 1$ and $w_E(k+1) \le 2$. We obtain E' such that $E \stackrel{*}{\Rightarrow} E'$, $w_{E'}(k) = 1$, $w_{E'}(k+1) = 2$, $w_{E'}(k+2) \ge 3$.

Given an arithmetic expression E containing operators of different types, we define the sequence c_E in the following manner: $c_E(i)$ is the operator adjacent to the set of parentheses adjacent to the variable i (except the first and the last variable).

For example, if E = (((x+y)/(z+t))+x) then $c_E = (/,/,+)$.

If we can apply several oriented associativity rules in expressions having different operators, then $E \stackrel{*}{\Rightarrow} E'$ iff the variables of E and E' appear in the same order from left to right, $w_E \leq w_{E'}$ and $c_E = c_{E'}$.

- D. E. Knuth and P. Bendix, Simple word problems in universal algebras. In *Computational Problems in Abstract* Algebras, pp. 263-297. Pergamon Press, Oxford (1970).
- W. R. Lalonde and J. Des Rivières, Handling operator precedence in arithmetic expressions with tree transformations. ACM Transactions on Programming Languages and Systems 3 (1), 83-103 (1981).
- D. Rotem and Y. L. Varol, Generation of binary trees from ballot sequences. *Journal of the ACM* 25 (3), 396-404 (1978)
- F. Ruskey and T. C. Hu, Generating binary trees lexicographically. SIAM Journal on Computing 7 (4), 492-509 (1978).
- 13. M. Solomon and R. A. Finkel, A note on enumerating binary trees. *Journal of the ACM* 27 (1), 3-5 (1980).
- 14. S. Zaks, Lexicographic generation of ordered trees. *Theoretical Computer Science* 10, 63-82 (1980).