

On generating B-trees with constant average delay and in lexicographic order

Mounir Belbaraka, Ivan Stojmenović *

Computer Science Department, University of Ottawa, 150 Louis Pasteur / Priv., Ottawa, Ontario, Canada K1N 9B4

(Communicated by S.G. Akl)

(Received 1 October 1992)

(Revised 1 October 1993)

Abstract

Gupta, Lee and Wong described algorithms for generating 2–3 trees and B-trees and left as open problems whether algorithms exist that generate them in lexicographic order, and whether it is possible to generate 2–3 trees or B-trees in constant average delay, exclusive of the output. In this note we modify the B-tree representation and show that the order of generating 2–3 trees and B-trees in their paper is lexicographic under the new representation. We also prove that their algorithm for generating B-trees has a constant average delay property. This is the first algorithm for which such a property is proved, showing that B-trees can be generated with constant average delay.

Key words: B-tree; Lexicographic order; Generating combinatorial objects; Design of algorithms

1. Introduction

A variety of balanced-tree schemes have been proposed for the organization of information so as to guarantee worst-case logarithmic search times. One such scheme, called 2–3 trees, was introduced by Hopcroft (see, for example, [1,10]). A 2–3 tree is a tree in which each internal (nonleaf) node has 2 or 3 children, and every path from the root to a leaf is of the same length. Internal nodes contain 1 or 2 keys depending upon whether they have 2 or 3 children respectively. Since we are interested only in the structure of the tree, the keys values are immaterial.

A B-tree of order m is a tree satisfying the following properties [3]: (1) All leaves are on the same level; (2) the root has q descendants, $2 \leq q \leq m$; and (3) other internal nodes have p descendants, $\lceil m/2 \rceil \leq p \leq m$. The 2–3 tree is a B-tree of order 3.

The production of available lists or catalogs of all the combinatorial configurations of a certain type is often useful. For example, a list of all shapes of trees of a given type might be used to search for a counter-example to some conjecture, or to test or analyze an algorithm for its correctness or computational complexity. The generation of such lists of all shapes of trees of some specified kind is therefore of some interest.

There exist more than forty sequential algorithms for generating binary and t -ary trees (see,

* Corresponding author.

for example, [15,16,18,21,2] and references therewith). Such algorithms also exist for generating AVL trees [12], 2–3 trees [6], B-trees [7], non-regular trees [5], unordered trees [14], free trees [20], trees with n nodes and m leaves [13], trees with nodes of any degree [17], and trees with bounded height [11].

Typically, a one-to-one correspondence is established between a class of trees and certain integer sequences. It is then shown how these sequences can be generated in order (usually lexicographic) and how given a sequence its position in this ordering can be determined (ranking) and vice-versa (unranking).

There are some desirable properties of algorithms that generate any kind of combinatorial objects. We mention two of them.

Sequence $A = (a_1, a_2, \dots, a_n)$ precedes sequence $B = (b_1, b_2, \dots, b_n)$ in lexicographic order if and only if, for some $j \geq 1$, $a_i = b_i$ when $i < j$, and $a_j < b_j$. It is desirable to generate objects in lexicographic order because the order is natural and easy to follow.

Let $P(n)$ be the number of combinatorial objects in question, each of them having the size $O(n)$. This means that the total output size is $O(nP(n))$. However, in some applications the objects which are generated do not need to be printed out, for they merely serve as the source of information for other procedures that work on combinatorial objects and check some criteria which may be verified without always looking at the whole new object. It makes sense then to consider generating combinatorial object without outputting them. Optimal algorithms in this sense work in $O(P(n))$ time, i.e. in constant time per object. Most algorithms that generate the next object from current one, for various kinds of combinatorial objects, have the property of having a constant average delay (exclusive of the output time); for example [3,15,8,18,19,21].

In [6] ranking and unranking procedures for 2–3 trees with n leaves (hence, $n - 1$ keys) were presented. Unranking procedure is then used to generate all 2–3 trees one by one, by simply unranking 1st, 2nd, 3rd, 4th, ... 2–3 trees until all of them are encountered. This takes $O(n)$ time between any two 2–3 trees. Since decoding (con-

structing a 2–3 structure from its array representation) can be done in $O(n)$ time, and the output size is also $O(n)$, this is claimed to be optimal in [6]. However, [6] does not take into account that the rank of a 2–3 tree can be a very large integer since the number of 2–3 trees is exponential in n . The representation of the rank of a tree may require $O(n)$ space and that much time for any arithmetic with it, causing an $O(n^2)$ delay in producing 2–3 trees rather than linear as claimed.

In [6] an open problem is posed, to see whether the next 2–3 tree can be generated from current one with constant average delay (exclusive of the output time). The algorithm [6] has a linear time average delay (in fact even higher, if dealing with large integers is counted). Also, [6] defined an order of 2–3 sequences, and states that “we could have chosen to order these sequences lexicographically, but we do not know of efficient algorithms for generation, ranking and unranking, given such an ordering.” In this paper we show, on the contrary, that their order becomes lexicographic, if a suitable sequence representation is used.

In [7] same authors considered more general problem of generating B-trees. They presented generating algorithm based on backtrack search instead of unranking, and obtained an algorithm that produces B-trees in time proportional to the output size. The same problems with lexicographic order and constant average delay remained unsolved in this paper. In [9] Kelsen presented $O(n)$ time algorithms for ranking and unranking B-trees on n leaves after a preprocessing step that required $O(n^2)$ time. The unranking algorithm may be used to generate B-trees with $O(n)$ delay per B-tree.

The contribution of this paper is to introduce sequence representation of B-trees under which the order in [6,7] is lexicographic, and to prove that the algorithm [7] has constant average delay property.

2. B-tree representation, encoding and ordering

As in [7], we shall represent a B-tree with n leaves (or $n - 1$ keys) by a sequence $a_1 a_2 \dots a_k$

that is obtained as follows: Starting with lowest level of internal nodes, list the number of children of each node, level by level ending with the root. Within the same level the listing is done from left to right.

If the sequence $a_1 \dots a_k$ represents a B-tree of order m with n leaves, then it must have the following properties [7] (for notational convenience, $x(y)$ means x_y):

- (1) $a_1 + a_2 + \dots + a_k = n + k - 1$,
- (2) $a_i \in \{\lceil m/2 \rceil, \lceil m/2 \rceil + 1, \dots, m\}$, $i = 1, 2, \dots, k - 1$, $2 \leq a_k \leq m$,
- (3) if $k > 1$ then there exist l_0, l_1, \dots, l_r such that $0 = l_0 < l_1 < \dots < l_{r-1} = k - 1$, $l_r = k$, $a(1) + \dots + a(l_1) = n$ (defines l_1), and $l_i - l_{i-1} = a(l_i + 1) + \dots + a(l_{i+1})$ for $i = 1, 2, \dots, r - 1$ (defines $l_i + 1$).

Sequences satisfying above properties are called *a-sequences* (determined by n), and their subsequences are *a-subsequences*. In [7] it is shown that a sequences determined by n are in one to one correspondence to B-trees with n leaves. Similarly, the sequence l_0, l_1, \dots, l_r is called *l-sequence*.

Property (3) says that every a-sequence should be partitionable into levels such that the number of nodes on a particular level is equal to the sum of the number of children of nodes at the next higher level, and the highest level has exactly one node. In fact, in the sequence $l_0 l_1 \dots l_r$, l_1 is the number of internal nodes at the lowest level (level 1), l_2 is the total number of internal nodes at the lowest two levels, l_3 is the total number of internal nodes at the lowest three levels and so on. Level i is composed from subsequence $a(l_{i-1} + 1), \dots, a(l_i)$, and the number $u_i = l_i - l_{i-1}$ will be referred to as its size. Clearly $u_r = 1$ and $u_{r-1} = a_k$. Note that the size of given level i is determined by $\lceil u_{i-1}/m \rceil \leq u_i \leq \lfloor u_{i-1}/\lceil m/2 \rceil \rfloor$.

For example, consider a-sequence 355333432 which corresponds to B-tree of order 5 with 25 leaves; see Fig. 1. The l-sequence is determined as follows: $l_1 = 7$ since $u_1 = 7$ first elements have sum 25 ($3 + 5 + 5 + 3 + 3 + 3 + 3$), $l_2 = 9$ since next $u_2 = 9 - 7 = 2$ elements have sum $u_1 = l_1 - l_0 = 7$ ($3 + 4$), and $l_3 = 10$ since $u_3 = 10 - 9 = 1$ and the remaining element is equal to $u_2 = l_2 - l_1 = 2$.

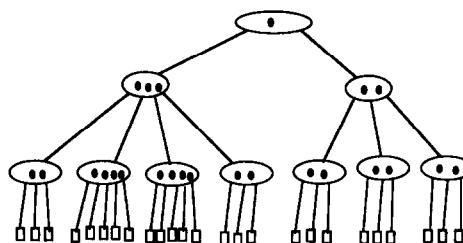


Fig. 1.

A normalized sequence $b(l_i + 1), \dots, b(l_{i+1})$ is associated with each a-subsequence of the form $a(l_i + 1), \dots, a(l_{i+1})$, for $j = 0, 1, \dots, r - 1$, such that b 's are permutations of a 's and are arranged in nonincreasing order, i.e. $b(l_i + 1) \geq \dots \geq b(l_{i+1})$.

In [6,7] an order of 2–3 and B-trees is defined and algorithms for generating trees in such order were given. The algorithm [7] runs in the following way.

The next B-tree is obtained from current one by a backtracking search (through levels $r - 1, r - 2, \dots, 1$), looking for the smallest possible index in current a-sequence that must be updated in order to produce the next a-sequence. When testing at level i , it is first checked whether there exist lexicographically higher a-subsequence having the same normalized sequence as the current a-subsequence. If so, the new a-subsequence is taken and the a-sequence is completed by lexicographically first possible subsequence. Otherwise, it is checked whether for the same l_i there exist lexicographically higher normalized sequence of the same length. If so, the corresponding first a-subsequence is constructed and a-sequence completed toward the next B-tree. Otherwise, l_i is increased by one, if possible, and the first a-sequence with given l_i constructed. Finally, if l_i has maximal possible value, the search backtracks to level $i - 1$.

[7] defines an order to correspond to above algorithms and apply a representation that does not lead to B-tree generation in lexicographic order. We point out that the order used in [6,7]

becomes lexicographic if B-trees are coded in the following way:

$$l_1 b(1) b(2) \dots b(l_1) a(1) a(2) \dots a(l_1) \\ l_2 b(l_1 + 1) \dots b(l_2) a(l_1 + 1) \dots a(l_2) \dots \\ l_r b(l_r) a(l_r).$$

We call this sequence a *B-tree sequence*. The ordering on B-trees is then easily defined as lexicographic order on corresponding B-tree sequences.

3. Constant average delay property

In this section we prove that the algorithm [7] generates B-trees in constant expected time, exclusive of the output time.

Let an (h, m) -composition be any integer sequence $b_1 \dots b_t$ such that $\lceil m/2 \rceil \leq b_i \leq m$ for each $i, 1 \leq i \leq t$, and $b_1 + \dots + b_t = h$, and let $C(h, m)$ be the number of such distinct (h, m) -compositions. In [7] normalized sequences are generated using a procedure that generates restricted integer partitions, and for an obtained integer partition all corresponding integer compositions (where the order of summands becomes important) are found. In our proof we only use the fact that the algorithm generates B-trees by generating such integer compositions, where the next composition can be determined from the current one in linear time. In this sense, our proof may be applied to some other algorithms for generating B-trees (for example, one which would avoid generating integer partitions and directly update integer compositions).

We use the following lemma in our proof.

Lemma. $C(h, m) \geq 5h^2/96m^2$ for $h \geq 4m$ and $m \geq 3$.

Proof. Let $p_1 d_1 + p_2 d_2 + \dots + p_s d_s = h$ be the normalized sequence for the (h, m) -composition $b_1 + \dots + b_t = h$, where $p_1 > p_2 > \dots > p_s$ are all distinct parts in given (h, m) -composition and d_1, \dots, d_s their multiplicities, $d_1 + d_2 + \dots + d_s = t$. The case $s = 1$ may occur only when $t = h/m$ ($b_1 = \dots = b_t = m$) or $t = h/\lceil m/2 \rceil$ ($b_1 = \dots$

$= b_t = \lceil m/2 \rceil$). Otherwise, for each $t, \lceil (h+1)/m \rceil \leq t \leq \lfloor (h-1)/\lceil m/2 \rceil \rfloor$, there exist at least two different parts ($s \geq 2$). We show that for each t in given interval there exist at least $t+1$ (h, m) -compositions, which can be obtained as follows. Starting from

$$p_1 + \dots + p_1 + p_2 + \dots + p_2 + \dots + p_s + \dots + p_s \\ = p_1 d_1 + p_2 d_2 + \dots + p_s d_s = h,$$

the last p_1 “walks” through the (h, m) -composition to generate $d_2 + \dots + d_s = t - d_1$ new (h, m) -compositions

$$p_1(d_1 - 1) + p_2 d_2 + \dots + p_{i-1} d_{i-1} + p_i d' \\ + p_1 + p_i(d_i - d') + p_{i+1} d_{i+1} + \dots + p_s d_s = h,$$

for $d' = 1, \dots, d_i$ and $i = 2, \dots, s$. Similarly, when p_s “walks” through the list, $d_1 + \dots + d_{s-1} = t - d_s$ (h, m) -compositions are produced. Thus there are at least $t - d_1 + t - d_s + 1 \geq t + 1$ (since $d_1 + d_s \leq t$) (h, m) -compositions for each t .

Therefore, the number of (h, m) -compositions of h is

$$C(h, m) \geq \left(\left\lceil \frac{h+1}{m} \right\rceil + 1 \right) + \left(\left\lceil \frac{h+1}{m} \right\rceil + 2 \right) \\ + \dots + \left(\left\lceil \frac{h-1}{\lceil m/2 \rceil} \right\rceil + 1 \right) \\ = \left(\left\lceil \frac{h-1}{\lceil m/2 \rceil} \right\rceil + \left\lceil \frac{h+1}{m} \right\rceil + 2 \right) \\ \times \left(\left\lceil \frac{h-1}{\lceil m/2 \rceil} \right\rceil - \left\lceil \frac{h+1}{m} \right\rceil + 1 \right)^{\frac{1}{2}}.$$

From $\lceil m/2 \rceil \leq (m+1)/2$ and $\lfloor x \rfloor > x - 1$ it follows that

$$\left\lceil \frac{h-1}{\lceil m/2 \rceil} \right\rceil \geq \left\lfloor \frac{2(h-1)}{m+1} \right\rfloor > \frac{2(h-1)}{m+1} - 1.$$

For $m \geq 3$ it easily follows that $2/(m+1) \geq 3/2m$. Thus

$$\left(\left\lceil \frac{h-1}{\lceil m/2 \rceil} \right\rceil + \left\lceil \frac{h+1}{m} \right\rceil + 2 \right) \\ \geq \frac{3(h-1)}{2m} - 1 + \frac{h+1}{m} + 2 \\ \geq \frac{3h-3+2h+2+2m}{2m} \geq \frac{5h}{2m}.$$

Consider now the second expression. From $\lceil (h+1)/m \rceil < (h+1)/m + 1$ and $\lfloor h-1/\lceil m/2 \rceil \rfloor > 3(h-1)/2m - 1$ it follows that

$$\left(\left\lfloor \frac{h-1}{\lceil m/2 \rceil} \right\rfloor - \left\lfloor \frac{h+1}{m} \right\rfloor + 1 \right) > \frac{3(h-1)}{2m} - 1 - \frac{h+1}{m} - 1 + 1.$$

We now prove that $3(h-1)/2m - (h+1)/m - 1 \geq h/24m$ for $h \geq 4m$ and $m \geq 3$. The condition is (after doing obvious transformations) equivalent to $h \geq (24/11)m + 60/11$. It easily follows from $h \geq 4m \geq (24/11)m + 60/11$ which is satisfied for $m \geq 3$.

From both approximations it follows that

$$C(h, m) \geq \frac{5h}{2m} \frac{h}{24m} \frac{1}{2} = \frac{5h^2}{96m^2}$$

for $h \geq 4m$ and $m \geq 3$. \square

Although a much stronger lemma can be derived (probably with more sophisticated arguments), this “weak” property is sufficient to prove the constant time average delay behavior of above algorithm.

Theorem. *The Gupta–Lee–Wong algorithm [7] has constant expected delay in producing next B-tree, excluding of output.*

Proof. Consider level i sequence $a(l_{i-1}+1) \dots a(l_i)$. The lexicographically next level sequence can be determined by a backtracking procedure that finds in a backward run an element that can be increased, and in a forward run finds the next sequence. The time for updating is linear in the number of elements of the sequence, i.e. it is bounded by cu_i , where c is a constant and $u_i = l_i - l_{i-1}$. However, the current level sequence can be completed in various ways to produce distinct B-tree sequences. During the production of these B-trees the algorithm for finding the next B-tree does not “arrive” in its search for the turning point to level i . For fixed sequence $a(l_{i-1}+1) \dots a(l_i)$, the sequence at the next level $a(l_i+1) \dots a(l_{i+1})$ can be any (u_i, m) -composition ($u_i = l_i$

$- l_{i-1}$) into parts with sizes between $\lceil m/2 \rceil$ and m , according to the definition of tree sequences. The number of (u_i, m) -compositions is at least $5u_i^2/96m^2$ whenever $u_i \geq 4m$ (from the Lemma). Therefore, cu_i operations are performed on level i when the turning point comes back to the level (even when the turning point continues its search to other levels), and in the meanwhile at least $5u_i^2/96m^2$ B-trees are produced that contribute zero operations (level i sequence intact). Thus the average number of operations per tree for level i is at most $cu_i/(5u_i^2/96m^2) = 96m^2c/5u_i$. The B-tree sequence $a_1 \dots a_k$ is partitioned always into levels l_1, \dots, l_r with sizes $u_i = l_i - l_{i-1}$, $1 \leq i \leq r$, and $u_i \geq \lceil u_i - 1/m \rceil \geq u_{i-1}/m$ is always satisfied, i.e. $u_i \leq m^{r-i}u_r + m^{r-i}$. Since $u_r = 1$ and $u_{r-1} = a_k \leq m < 4m$, the condition $u_i \geq 4m$ is not satisfied for $i = r-1$ and $i = r$. Let j be the maximal index such that $u_j \leq 4m$ (clearly $j \leq r-2$). The average number of operations for all levels $1, 2, \dots, j$ together is smaller than

$$\begin{aligned} & \left(\frac{1}{u_1} + \frac{1}{u_2} + \dots + \frac{1}{u_j} \right) \frac{96m^2c}{5} \\ & \leq \left(\frac{1}{m^{r-1}} + \frac{1}{m^{r-2}} + \dots + \frac{1}{m^{r-j}} \right) \frac{96m^2c}{5} \\ & < \frac{96m^2c}{5m^{r-j}} \frac{m}{m-1} < 40c, \end{aligned}$$

which is a constant.

The case which was not counted in is when $u_i < 4m$. Because each part is $\geq \lceil m/2 \rceil$, it means that the compositions of any such u_i has at most 7 parts ($t \leq 7$ in $b_1 + \dots + b_t = u_i$). Depending on m , there are between 1 and 3 parts in similar composition of 7 and probably one more part of size 2 or 3, i.e. at most 4 more elements in tree sequence. Therefore, the part of the sequence which is not considered in our operation count contains at most the last 11 elements. Since the algorithm is a backtrack search, for each new tree it takes at most constant additional time to pass through them in both backward and forward direction. Therefore the algorithm has overall constant time average delay. \square

5. Conclusion

Our proof of constant average delay property assumes only a kind of level sequence representation and their lexicographic order. More precisely, the proof requires merely that an algorithm generates all B-trees with given fixed a-subsequence (level sequence) in a block of consecutive sequences, such that the sequences in the block can be generated without any work performed on given a-subsequence except at the beginning and end of block when linear update time suffices. Therefore, proof applies, for instance, to an algorithm that generates B-trees in lexicographic order of a-sequences as the only part of B-tree representation (i.e. l-sequences and normalized sequences not being part of representation).

There exist a straightforward decoding procedure that generates the B-tree data structure (meaning that the parent–children links are established) from given B-tree sequence. Since only the part of sequence that is updated needs to update its parent–children links in the procedure, the algorithm [7] can be completed to produce B-tree data structures with constant average delay.

The problem that can be considered is to find an algorithm that generates B-trees in Gray code order (i.e. minimal change order). Alternatively, it is interesting to derive an algorithm that generates B-trees (or merely corresponding sequences) which will have constant time delay in the worst case. It also remains an open problem to generate a parallel algorithm for generating B-trees, which will satisfy some desirable properties. Such algorithms exist for the case of binary and t -ary trees [2].

Acknowledgement

We appreciate careful reading and comments received by two referees that has improved the presentation of the paper.

References

- [1] A.V. Aho, J.E. Hopcroft and J.D. Ulman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1976).
- [2] S.G. Akl and I. Stojmenović, Generating binary trees in parallel, in: *Proc. Allerton Conf. on Communications, Control and Computing* (1992), to appear.
- [3] R. Bayer and E. McCreight, Organization and maintenance of large ordered indices, *Acta Inform.* **1** (1972) 173–189.
- [4] T. Beyer and S.M. Hedetniemi, Constant time generation of rooted trees, *SIAM J. Comput.* **9** (4) (1980) 706–712.
- [5] M.C. Er, A simple algorithm for generating non-regular trees in lexicographic order, *Comput. J.* **31** (1988) 61–64.
- [6] U.I. Gupta, D.T. Lee and C.K. Wong, Ranking and unranking of 2-3 trees, *SIAM J. Comput.* **11** (1982) 582–590.
- [7] U.I. Gupta, D.T. Lee and C.K. Wong, Ranking and unranking of B-trees, *J. Algebra* **4** (1983) 51–60.
- [8] T. Hikita, Listing and counting subtrees of equal size of a binary tree, *Inform. Process. Lett.* **17** (1983) 225–229.
- [9] P. Kelsen, Ranking and unranking of trees using regular reductions, Computer Science M.Sc. Thesis, University of Illinois at Urbana-Champaign, 1989.
- [10] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching* (Addison-Wesley, Reading, Ma, 1973).
- [11] C.C. Lee, D.T. Lee and C.K. Wong, Generating binary trees of bounded height, *Acta Inform.* **23** (1986) 529–544.
- [12] L. Li, Ranking and unranking of AVL trees, *SIAM J. Comput.* **15** (1986) 1025–1035.
- [13] J.M. Pallo, Generating trees with n nodes and m leaves, *Internat. J. Comput. Math.* **21** (1987) 133–144.
- [14] J.M. Pallo, Lexicographic generation of binary unordered trees, *Pattern Recognition Lett.* **10** (1989) 217–221.
- [15] F. Ruskey, Generating t -ary trees lexicographically, *SIAM J. Comput.* **7** (1978) 492–509.
- [16] F. Ruskey and T.C. Hu, Generating binary trees lexicographically, *SIAM J. Comput.* **6** (1977) 745–758.
- [17] W. Skarbek, Generating ordered trees, *Theoret. Comput. Sci.* **57** (1988) 153–159.
- [18] D.R. van Baronaigien, A Loopless algorithm for generating binary tree sequences, *Inform. Process. Lett.* **39** (1991) 189–194.
- [19] S.G. Williamson, *On the Ordering, Ranking, and Random Generation of Basic Combinatorial Sets*, Lecture Notes in Mathematics **579** (Springer, Berlin, 1976).
- [20] R.A. Wright, B. Richmond, A. Odlyzko and B.D. McKay, Constant time generation of free trees, *SIAM J. Comput.* **15** (1986) 540–548.
- [21] S. Zaks, Generation and ranking of k -ary trees, *Inform. Process. Lett.* **14** (1982) 44–48.