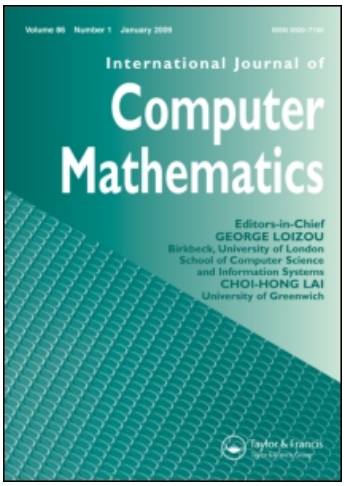

International Journal of Computer Mathematics
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713455451

Generating all distributions of objects to bins

Muhammad Abdullah Adnan [a]; Md. Saidur Rahman [a]
[a] Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh

First Published:March2009

To cite this Article Adnan, Muhammad Abdullah and Rahman, Md. Saidur(2009)'Generating all distributions of objects to bins',International Journal of Computer Mathematics,86:3,383 — 392

To link to this Article: DOI: 10.1080/00207160701639364

URL: http://dx.doi.org/10.1080/00207160701639364

International Journal of Computer Mathematics
Vol. 86, No. 3, March 2009, 383–392

Taylor & Francis
Taylor & Francis Group

Generating all distributions of objects to bins

Muhammad Abdullah Adnan* and Md. Saidur Rahman

Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh

(*Received 04 October 2006; revised version received 11 April 2007; second revision received 08 July 2007; accepted 08 August 2007*)

In this paper, we give an elegant algorithm to generate all distributions of n identical objects to m bins without repetition. The best known previous algorithm, due to Klingsberg, generates each distribution in constant time in average sense. Using a new technique of efficient tree traversal, in this paper we improve the time complexity to constant time (in ordinary sense). By modifying our algorithm, we can generate the distributions in anti-lexicographic order. Overall space complexity of our algorithm is $O(m \lg n)$, where m is the number of bins and n is the number of objects.

Keywords: combinatorial objects; algorithm; generating problems; integer partitions; set partitions

2000 AMS Subject Classifications: 05A15; 05A17; 05A18

1. Introduction

A well known counting problem in combinatorics is counting the number of ways objects can be distributed among bins [1,7]. The paradigm problem is counting the number of ways of distributing fruits to children. For example, Kathy, Peter and Susan are three children. We have four apples to distribute among them without cutting apples into parts. In how many ways the children receive apples? One can easily count the number of distributions for m bins and n objects, which is $(n+m-1)!/n!(m-1)!$ [1,7]. However, in this paper we are not interested in counting the number of distributions, rather we are interested in generating all distributions. It is useful to have the complete list of all such solutions. One can use such a list to search for a counter-example to some conjecture, to find best solution among all solutions or to test and analyse an algorithm for its correctness or computational complexity. Many algorithms to generate a particular class of objects without repetition, are already known [2–4,6,9,12,13].

Let, $D(n, m)$ represent the set of all distributions of n objects to m bins where each bin gets zero or more objects. For the previous example, we have $D(4, 3)$ representing all distributions. Now, let, (i, j, k) represent the situation in which Kathy receives i apples, Peter receives j and Susan receives k. The $6!/4!2! = 15$ possibilities are – (0,0,4), (0,1,3), (0,2,2), (0,3,1), (0,4,0), (1,0,3), (1,1,2), (1,2,1), (1,3,0), (4,0,0), (2,0,2), (2,1,1), (2,2,0), (3,0,1), (3,1,0).

Draw
n = 3
m = 2

*Corresponding author. Email: adnan@cse.buet.ac.bd

ISSN 0020-7160 print/ISSN 1029-0265 online

DOI: 10.1080/00207160701639364
http://www.informaworld.com

Applications

There are many applications of distribution of objects to bins. In these days of automation, machines may require to distribute objects among candidates optimally. Generating all distributions also has many applications in computer science [10,11]. In computer networks suppose there are several communication channels and several processes want to use the channels. We can think of communication channels as bins and the processes as our symbolic objects. To find out which distribution is better taking into account congestion, QoS, channel capacity and different factors, we may need to calculate these values for each distribution. Then we may choose the optimal one. In client-server broker distributed architecture, broker may need to dispatch requests among different servers in an efficient way.

The problem of generating all distributions of n objects to m bins can be viewed as generating integer partitions of the integer n when there are m partitions, and the partitions are 'fixed', 'numbered' and 'ordered'. That means the number of partitions is fixed, the partitions are numbered and the assigned numbers to bins are not altered. Zoghbi and Stojmenovic [13] gave an algorithm to generate integer partitions with a specified order of generation (lexicographic and anti-lexicographic), but their partitions are not fixed, numbered and ordered. Moreover, their algorithm does not allow empty partitions. Kawano and Nakano [4] generated all set partitions where the number of partitions are fixed but the subsets are not numbered or ordered. They used efficient generation method based on the 'family tree structure' of the solutions. If we apply their method to this problem then we have to number the subsets. Then we have to permutate the numbers that we have assigned to the subsets. Since the objects in this problem are identical, permutating the assigned numbers leads to repetition when any two of the subsets contain same number of objects. Thus modifying their algorithm we cannot solve our problem of generating distributions.

Klingsberg [5] gave an algorithm for sequential listing of the composition of an integer n into k parts. The algorithm keeps pointers to the first and second non-zero elements in the sequence. Then by incrementing and decrementing the proper elements in the sequence their algorithm generates solutions. Their method is straightforward, but requires searching for the second non-zero element in the sequence, for the solutions having a non-zero as the first element. Hence, their algorithm cannot generate each solution in $O(1)$ time in ordinary sense, rather the cost per generation is constant averaging over all solutions in $D(n, m)$.

Klingsberg

In this paper we give a new algorithm to generate all distributions of n objects to m bins without repetition. Our algorithm generates each distribution in constant time without repetition. The main feature of our algorithm is that we define a tree structure representing parent–child relationships among the distributions in $D(n, m)$ as illustrated in Figure 1. In such a 'tree of distributions,' each node corresponds to a distribution of objects to bins and each node is generated from its parent in constant time. In our algorithm, we construct the tree structure among the distributions in such a way that the parent–child relation is unique, and hence there is no chance of producing duplicate distributions. Once such a parent–child relationship is established, one can generate all the distributions in $D(n, m)$ by traversing the tree using the relationship. But the problem of ordinary traversal is that after generating a distribution corresponding to the last vertex in the largest level in the tree, we have to merely return from the deep recursive

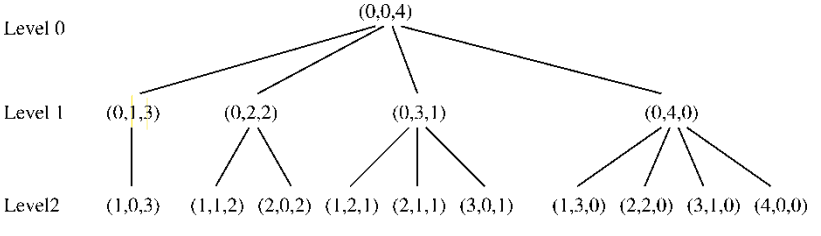

Figure 1. The family tree $T_{4,3}$ for four objects and three bins.

call without outputting any sequence and hence ordinary traversal generates each distribution in constant time 'on average'. To generate each distribution in $O(1)$ time (in ordinary sense), we define two additional types of relationships: (1) Relationship between left sibling and right sibling and (2) Leaf–ancestor relationship. Thus, our algorithm reduces many non-generation steps and outputs each distributions in constant time in ordinary sense (not in average sense). Our algorithm, generates a new distribution from an existing one by making a constant number of changes and outputs each distribution as the difference from the preceding one. Thus, we can regard the derived sequence of the outputs as a combinatorial Gray code [9,4,8] for distributions. Our algorithm also generates the distributions *in place*, that means, the space complexity is only $O(m)$. To the best of our knowledge, our algorithm is the first algorithm to generate all distributions in constant time per distribution in ordinary sense. Our algorithm also generates distributions with a specified order of generation. By using this algorithm we can generate integer partitions in anti-lexicographic order when the partitions are fixed and ordered.

The rest of the paper is organised as follows. Section 2 gives some definitions. In Section 3, we define a tree structure among distributions in $D(n, m)$. In Section 4, we present the improved tree traversal algorithm that generates each distribution in $O(1)$ time. Finally, Section 5 is a conclusion.

2. Preliminaries Safin

In this section we define some terms used in this paper.

Let G be a connected graph with n vertices. A *tree* is a connected graph without cycles. A *rooted tree* is a tree with one vertex r chosen as root. A *leaf* in a tree is a vertex of degree 1. A *family tree* is a rooted tree with parent–child relationship. The vertices of a rooted tree have *levels* associated with them. The root has the lowest level *i.e.* 0. The level for any other node is one more than its parent except root. Vertices with the same parent v are called *siblings*. The siblings may be ordered as $c_1, c_2, \ldots, c_l$ where l is the number of children of v. If the siblings are ordered then c_{i-1} is the *left sibling* of c_i for $1 < i \leq l$ and c_{i+1} is the *right sibling* of c_i for $1 \leq i < l$. The *ancestors* of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex and including the root itself. A leaf in a family tree has no children.

Given an integer n, it is possible to represent it as the sum of one or more positive integers x_i, *i.e.* $n = x_1 + x_2 + \cdots + x_m$ for $1 \leq m \leq n$. This representation is called an *integer partition*. For a positive integer n and $k < n$, *set partition* is the set of all partitions of $\{1, 2, \ldots, n\}$ into k non-empty subsets.

For positive integers n and m, let $D(n, m)$ represents the set of all distributions of n objects to m bins where each bin gets zero or more objects. The bins are ordered and numbered as $B_1, B_2, \ldots, B_m$. For each distribution $A \in D(n, m)$, we define a unique sequence of non-negative integers $(a_1, a_2, \ldots, a_m)$, where a_i represents number of objects in ith bin B_i, for $1 \leq i \leq m$. The sequence for A is unique for each distribution because the bins are ordered and numbered. For example, $(0, 0, 4)$ represents three bins and four objects and the third bin contains four objects and the rest of the bins are empty. We can observe that for each sequence $a_1 + a_2 + \cdots + a_m = n$. This equality holds because the number of objects are fixed and we have to place every object to some bins.

Lexicographic order for distribution of objects is defined as follows. If $P = (p_1, p_2, \ldots, p_m)$ and $Q = (q_1, q_2, \ldots, q_m)$ are sequences for two distributions, then P precedes Q lexicographically if and only if, for some k, $p_k < q_k$ and $p_i = q_i$ for all $1 \leq i \leq (k-1)$. For example, the distributions of four objects to three bins in lexicographic order are: $(0, 0, 4)$, $(0, 1, 3)$, $(0, 2, 2)$, $(0, 3, 1)$, $(0, 4, 0)$ and so on. The *anti-lexicographic* order is the reverse of lexicographic one. The distributions in anti-lexicographic order are: $(4, 0, 0)$, $(3, 1, 0)$, $(3, 0, 1)$, $(2, 2, 0)$, $(2, 1, 1)$ and so on.

A listing of distributions is said to be in *gray code order* if each successive sequences for distributions in the listing differs by a minimum constant amount, *e.g.*, the swapping of elements or the flipping of a bit. In this paper, we establish such an ordering of all distribution of objects to bins so that each distribution can be generated by making constant amount of changes to the preceding distribution in the order.

In the following sections we give an algorithm to generate all distributions of identical objects to bins. For that purpose we define a unique parent–child relationship among the distributions in $D(n, m)$ so that the relationship among the distributions can be represented by a tree with a suitable distribution as the root. Figure 1 shows such a tree of distributions where each distribution in the tree is in $D(4, 3)$. Once such a parent–child relationship is established, we can generate all the distributions in $D(n, m)$ using the relationship. We do not need to build or store the entire tree of distributions at once, rather we generate each distribution in the order it appears in the tree structure.

In Section 3, we define a tree structure among distributions in $D(n, m)$ and in Section 4, we present our algorithm which generates each solution in $O(1)$ time in ordinary sense.

3. The family tree of distributions Safin

In this section we define a tree structure $T_{n,m}$ among the distributions in $D(n, m)$. Each node of $T_{n,m}$ represents a distribution $(a_1, a_2, \ldots, a_m) \in D(n, m)$ where a_k represents number of objects in kth bin, for $1 \leq k \leq m$. If there are m bins then there are m levels in $T_{n,m}$. A node is at level i in $T_{n,m}$ if $a_1 = a_2 = \cdots = a_{m-i-1} = 0$ and $a_{m-i} \neq 0$ for $0 \leq i < m$. As the level increases the number of leftmost 0 decreases and vice versa. Thus a node at level $m - 1$ has no leftmost 0 before leftmost non-zero integer, *i.e.* $a_1 \neq 0$. Since $T_{n,m}$ is a rooted tree we need a root and the root is a node at level 0. One can observe that a node is at level 0 in $T_{n,m}$ if $a_1 = a_2 = \cdots = a_{m-1} = 0$ and $a_m \neq 0$. In this case, $a_m = n$ and there can be exactly one such node. We thus take the sequence $(0, 0, \ldots, 0, n)$ as the root of $T_{n,m}$. Clearly, the number of leftmost 0 before any non-zero integer in root is greater than that of any other sequence for any distribution in $D(n, m)$.

To construct $T_{n,m}$, we define two types of relationships: (a) Parent–child relationship and (b) Child–parent relationship among the distributions in $D(n, m)$, which are discussed in the following subsections.

3.1 *Parent–child relationship* Safin

Let $A \in D(n, m)$ be a sequence $(a_1, a_2, \ldots, a_m)$ and it corresponds to a node of level $i, 0 \leq i < m$ of $T_{n,m}$. So, we have $a_1 = a_2 = \cdots = a_{m-i-1} = 0$ and $a_{m-i} \neq 0$ for $0 \leq i < m$. The number of children it has is equal to a_{m-i}. The sequence of the children are defined in such a way that in order to generate a child from its parent we change only two integers in the parent sequence and the rest of the integers remain unchanged. The number of leftmost 0 decreases in the child sequence by applying parent–child relationship.

Let $C_j(A) \in D(n, m)$ be the sequence of jth child, $1 \leq j \leq a_{m-i}$ of A. Note that A is in level i of $T_{n,m}$ and $C_j(A)$ will be in level $i + 1$ of $T_{n,m}$. We define the sequence for $C_j(A)$ as $(c_1, c_2, \ldots, c_{m-i-1}, c_{m-i}, \ldots, c_m)$, where $0 \leq i < m$, $c_1 = c_2 = \cdots = c_{m-i-2} = 0$, $c_{m-i-1} = j$, $c_{m-i} = a_{m-i} - j$ and $c_k = a_k$ for $m - i + 1 \leq k \leq m$. Thus, we observe that $C_j(A)$ is a node of level $i + 1$, $0 \leq i < m - 1$ of $T_{n,m}$ and so $c_1 = c_2 = \cdots = c_{m-i-2} = 0$ and $c_{m-i-1} \neq 0$ for $0 \leq i < m - 1$. So, for each consecutive level we only deal with two numbers a_{m-i-1} and a_{m-i} and the rest of the integers remain unchanged. For example, the distribution $(0, 0, 4)$, for $n = 4$ and $m = 3$, is a node of level 0 because $a_1 = 0$, $a_2 = 0$ and $a_3 \neq 0$. Here, $a_{m-i} = 4$ so it has four children and the four children are shown in Figure 1.

3.2 *Child–parent relationship*

The child–parent relation is just the reverse of parent–child relation. Let, $A \in D(n, m)$ be a sequence $(a_1, a_2, \ldots, a_m)$ and it corresponds to a node of level i, $1 \leq i < m$ of $T_{n,m}$. So, we have $a_1 = a_2 = \cdots = a_{m-i-1} = 0$ and $a_{m-i} \neq 0$ for $0 < i < m$. We define a unique parent sequence of A at level $i-1$ of $T_{n,m}$. Like the parent–child relationship here we also deal with only two integers in the sequence. The number of leftmost 0 increases in the parent sequence by applying child–parent relationship.

Let $P(A) \in D(n, m)$ be the parent sequence of A. We define the sequence for $P(A)$ as $(p_1, p_2, \ldots, p_{m-i}, p_{m-i+1}, \ldots, p_m)$ where $1 \leq i < m$, $p_1 = p_2 = \cdots = p_{m-i} = 0$, $p_{m-i+1} = a_{m-i} + a_{m-i+1}$ and $p_j = a_j$ for $m - i + 1 < j \leq m$. Thus, we observe that $P(A)$ is a node of level $i-1$, $1 \leq i < m$ of $T_{n,m}$ and so $p_1 = p_2 = \cdots = p_{m-i} = 0$ and $p_{m-i+1} \neq 0$ for $1 \leq i < m$. For example, the distribution $(0, 3, 1)$, for $n = 4$ and $m = 3$, is a node of level 1 because $a_1 = 0$ and $a_2 \neq 0$. It has a unique parent $(0, 0, 4)$ as shown in Figure 1.

From the above definitions we can construct $T_{n,m}$. We take the sequence $A_r = (a_1, a_2, \ldots, a_m)$ as root where $a_1 = a_2 = \cdots = a_{m-1} = 0$ and $a_m = n$ as we mentioned before. The family tree $T_{4,3}$ for the distributions in $D(4, 3)$ is shown in Figure 1. Based on the above parent–child relationship, the following lemma proves that every distribution in $D(n, m)$ is present in $T_{n,m}$.

LEMMA 3.1 *For any distribution $A \in D(n, m)$, there is a unique sequence of distributions that transforms A into the root A_r of $T_{n,m}$.*

Proof Let $A \in D(n, m)$ be a sequence, where A is not the root sequence. By applying child–parent relationship, we find the parent sequence $P(A)$ of the sequence A. If $P(A)$ is the root sequence, then we stop. Otherwise, we apply the same procedure to $P(A)$ and find its parent $P(P(A))$. By continuously applying this process of finding the parent sequence of the derived sequence, we have the unique sequence $A, P(A), P(P(A)), \ldots$ of sequences in $D(n, m)$, which eventually ends with the root sequence A_r of $T_{n,m}$. We observe that $P(A)$ has at least one zero more than A in its sequence. Thus $A, P(A), P(P(A)), \ldots$ never lead to a cycle and the level of the derived sequence decreases which ends up with the level of root sequence A_r. ■

Lemma 3.1 ensures that there can be no omission of distributions in the family tree $T_{n,m}$. Since there is a unique sequence of operations that transforms a distribution $A \in D(n, m)$ into the root A_r of $T_{n,m}$, by reversing the operations we can generate that particular distribution, staring from root. We have the following lemma to show that distributions are generated without repetition.

LEMMA 3.2 *The family tree $T_{n,m}$ represents distributions in $D(n, m)$ without repetition.*

Proof Given a sequence $A \in D(n, m)$, the children of A are defined in such a way that no other sequence in $D(n, m)$ can generate any child of A. Assume for a contradiction that the two sequences $A, B \in D(n, m)$ are at level i of $T_{n,m}$ and generate same child C. So, C is a sequence of level $i+1$ of $T_{n,m}$. The sequences for A, B and C are a_j, b_j and c_j for $1 \leq j \leq m$. Clearly, $a_k = b_k = 0$ for $1 \leq k \leq m - i - 1$. According to parent–child relationship, we have $a_k = b_k = c_k$ for $m - i + 1 \leq k \leq m$ because only two integers in the sequence are changed and the rest of the integers remain unchanged. From the above two equations we have $a_k = b_k$ for $k \neq m - i$ and $1 \leq k \leq m$. Note that, $a_1 + a_2 + \cdots + a_m = n = b_1 + b_2 + \cdots + b_m$. Substituting the values for a_k and b_k for $k \neq m - i$ and $1 \leq k \leq m$ and simplifying yields $a_{m-i} = b_{m-i}$. So, $a_k = b_k$ for $1 \leq k \leq m$. This implies that A and B are same sequence. By contradiction, every sequence has a single and unique parent. ■

4. Efficient tree traversal

In this section, we present an efficient algorithm to traverse the family tree defined in the previous section and generate all distributions in $D(n, m)$.

One can use depth first traversal to traverse the family tree. But the problem of depth first traversal is that after generating a distribution corresponding to the last vertex in the largest level in the tree, we have to merely return from the deep recursive call without outputting any sequence and hence depth first traversal generates each distribution in constant time 'on average'. To generate each distribution in $O(1)$ time (in ordinary sense), we introduce two additional types of relationships: (1) Relationship between left sibling and right sibling and (2) Leaf–ancestor relationship, as described in Section 4.1 and 4.2. In Section 4.3 we present our efficient tree traversal algorithm.

4.1 *Relationship between left sibling and right sibling* Adnan

Let $A \in D(n, m)$ be a sequence $(a_1, a_2, \ldots, a_m)$ and it corresponds to a node of level i, $1 \leq i < m$ of $T_{n,m}$. So, we have $a_1 = a_2 = \cdots = a_{m-i-1} = 0$ and $a_{m-i} \neq 0$ for $1 \leq i < m$. We say the right sibling $A_s \in D(n, m)$ of node A exists if $a_{m-i+1} \neq 0$ at level i of $T_{n,m}$. Then we call the sequence A_s as the right sibling of A and A as the left sibling of A_s. The aim of defining the relationship between left sibling and right sibling is to generate A_s directly from A without generating their parent as an intermediate distribution. Hence, it improves efficiency by reducing non-generation steps.

We define the sequence for A_s as $(s_1, s_2, \ldots, s_{m-i}, s_{m-i+1}, \ldots, s_m)$, $1 \leq i < m$, where $s_1 = s_2 = \cdots = s_{m-i-1} = 0$, $s_{m-i} = a_{m-i} - 1$, $s_{m-i+1} = a_{m-i+1} + 1$ and $s_j = a_j$ for $m - i + 2 \leq j \leq m$. That means, to obtain A_s from A, we set $s_{m-i} = a_{m-i} - 1$ and $s_{m-i+1} = a_{m-i+1} + 1$ and the rest of the integers remain unchanged. Thus, A_s can be obtained from A in constant time. For example, in Figure 2 the distribution $(0, 0, 3, 1)$ is a node of level 1 and it has a right sibling $(0, 0, 4, 0)$.

4.2 *Leaf–ancestor relationship* Adnan

To avoid returning from deep recursive call without outputting any sequence, we define leaf–ancestor relationship. After generating the sequence A_l of the last vertex in the largest level *i.e. rightmost leaf*, we do not return to parent. Instead, we return to the nearest ancestor A_a which has right sibling. By *rightmost leaf* we mean that leaf which has no right sibling. Thus this leaf-ancestor relation saves many non-generation steps. Furthermore, using leaf-ancestor relationship, the nearest ancestor can be generated from the leaf sequence by a simple swap operation between two integers in the sequence. The other integers in the sequence remain unchanged.

Let the sequence for a leaf $A_l \in D(n, m)$ be $(a_1, a_2, \ldots, a_m)$ and it corresponds to a node of level $m - 1$ of $T_{n,m}$. Then $a_1 \neq 0$. We say that the ancestor sequence $A_a \in D(n, m)$ of node A_l

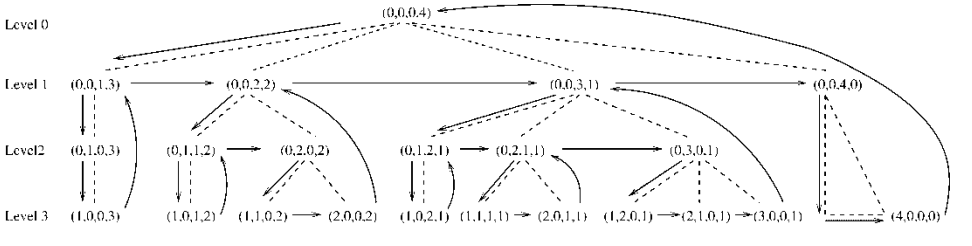

Figure 2. Efficient traversal of the family tree $T_{4,4}$.

exists if $a_2 = 0$ *i.e.* A has no right sibling. We define the ancestor sequence A_a of A_l at level $m-1-k$ if $a_2 = a_3 = \cdots = a_{k+1} = 0$ and $a_{k+2} \neq 0$. The sequence for nearest ancestor A_a is determined by the number of consecutive 0's after a_1 in the sequence for A_l. We denote the number of such 0's in the sequence A_l as k. This k will determine the level and sequence of the nearest ancestor A_a, which has right sibling.

We define the sequence for A_a as $(s_1, s_2, \ldots, s_k, s_{k+1}, \ldots, s_m)$, where $s_1 = s_2 = \cdots = s_k = 0$ and $s_{k+1} = a_1$ and $s_j = a_j$ for $k+1 < j \leq m$. In other words, to obtain A_a from A_l, we swap a_1 and a_{k+1} and the rest of the integers remain unchanged. One can observe that the sequence A_a is at level $m-1-k$ of $T_{n,m}$. For example, in Figure 2 the distribution (3,0,0,1) is a node of level 3 and it has a nearest ancestor (0,0,3,1), which is obtained by swapping first integer 3 and third integer 0. We have the following lemma on the nearest ancestor A_a of A_l.

LEMMA 4.1 *Let A_l be a leaf sequence of $T_{n,m}$ having no right sibling. Then A_l has a unique ancestor sequence A_a in $T_{n,m}$. Furthermore, either A_a has a right sibling in $T_{n,m}$ or A_a is the root A_r of $T_{n,m}$.*

Proof Let the sequence for $A_l \in D(n,m)$ be $(a_1, a_2, \ldots, a_m)$ and it corresponds to a node of level $m-1$ of $T_{n,m}$. Note that $a_1 \neq 0$ and $a_2 = 0$. We get the sequence for A_a by swapping a_1 and a_{k+1} where k is the number of consecutive 0's after a_1. Clearly A_a is an ancestor of A_l. Note that A_a is at level $m-1-k$. By Lemma 3.2, parent–child relationship is unique. Hence, by repeatedly applying child–parent relation on A_l, we will reach a unique ancestor at level $m-1-k$. For $k = m-1$, one can observe that we get the root sequence A_r by swapping a_1 and a_m. For $1 \leq k < m-1$, we get the unique ancestor sequence A_a, which has a right sibling. ■

Lemma 4.1 ensures that A_l has a unique ancestor A_a. As we see later, A_a plays an important role in our algorithm. Note that, we may need to return to ancestor A_a if current node is a leaf A_l and for a leaf sequence A_l we have $a_1 \neq 0$. A_a is obtained from A_l by swapping a_1 and a_{k+1} where k is the number of consecutive 0's after a_1. Now, to find out k we have to search the sequence A_l from a_1 to a_{k+1} such that $a_2 = a_3 = \cdots = a_{k+1} = 0$ and $a_1 \neq 0$, $a_{k+2} \neq 0$. We reduce the complexity of searching by keeping extra information as shown in Figure 3 (for simplicity we omit the separators). The information consists of the number of subsequences of consecutive 0's and the number of 0's in each subsequence after a_{m-i}, where i is the current level. For this we keep a stack of size $m/2$. The top of the stack determines the current k. Initially the stack is empty. As soon as we find a zero, when moving from parent to child or left sibling to right sibling, we push a 1 on the stack. We increment the top of the stack by one for consecutive 0's. We make a pop operation when we apply the leaf-ancestor relationship. The stack operations are shown in

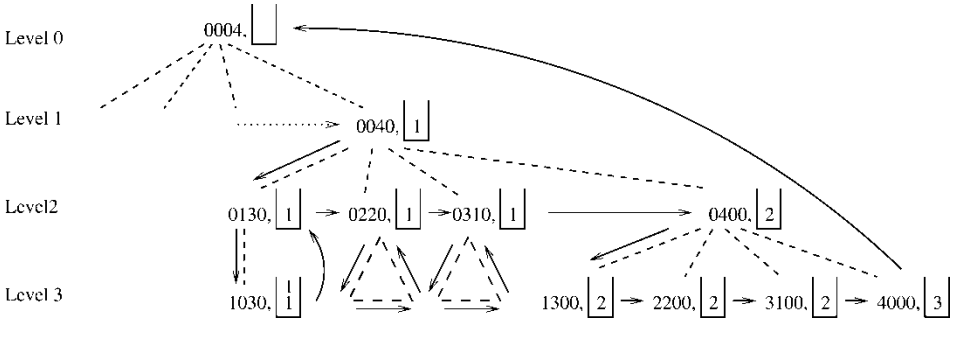

Figure 3. Efficient traversal of $T_{4,4}$ keeping extra information.

Figure 3. One can observe that there can be at most $m/2$ subsequences of consecutive 0's in a sequence of size m. Therefore, in worst case we need a stack of size $m/2$.

4.3 *The efficient algorithm* Adnan

In this section we present an efficient algorithm to generate all distributions in $D(n, m)$. We use three relationships in this algorithm, they are parent–child relationships, relationship between left sibling and right sibling and leaf–ancestor relationship. We start with the root sequence $A_r = (0, \ldots, 0, n)$. By applying parent–child relationship, we go from root down the family tree $T_{n,m}$ until we reach leaf at level $m-1$. Then, we apply the relationship between left sibling and right sibling to traverse horizontally until we reach a node, which has no right sibling. Then, by applying leaf–ancestor relationship, we return to that nearest ancestor which has sibling. Then we again apply the relationship between left sibling and right sibling. The sequence of applying relationships and generating distributions continues until we reach root. This algorithm thus reduces non-generation steps and generates each sequence in $O(1)$ time (in ordinary sense).

Procedure Find-All-Child-Distributions$(A = (a_1, a_2, \ldots, a_m), i, S)$
{ A is the current sequence, i indicates the current level, A_c is the child sequence, A_s is the right sibling sequence, A_a is the ancestor sequence and S indicates the current stack }
begin
  Output A {Output the difference from the previous distribution}
  if $a_1 = 0$ then
    begin
      {A has child}
      if $a_{m-i} - 1 = 0$ then
        if $a_{m-i+1} \neq 0$ then **Push**$(1, S)$;
        else Top(S)=**Top**$(S)+1$;
      Find-All-Child-Distributions$(A_c = (a_1, a_2, \ldots, a_{m-i-2}, 1, (a_{m-i} - 1), \ldots, a_m), i + 1, S)$;
    end
  else if $a_2 \neq 0$ then
    begin
      {A has right sibling}
      if $a_2 - 1 = 0$ then
        if $a_3 \neq 0$ then **Push**$(1, S)$;
        else Top(S)=**Top**$(S)+1$;
      Find-All-Child-Distributions$(A_s = ((a_1 + 1), (a_2 - 1), \ldots, a_m), i, S)$
    end
  else
    begin
      k=**Pop**(S);
      Swap(a_1, a_{k+1}); {Generate the ancestor A_a of A}
      if $k = m - 1$ then **return** ; {A_a is the root}
      else
      begin
        {A_a has right sibling}
        if $a_{k+2} - 1 = 0$ then
          if $a_{k+3} \neq 0$ or $k + 2 = m$ then **Push**$(1, S)$;
          else Top(S)=**Top**$(S)+1$;
        Find-All-Child-Distributions$(A_{as} = (a_1, a_2, \ldots, (a_{k+1}+1), (a_{k+2} - 1), \ldots, a_m), m-1-k, S)$;
      end
    end
end;

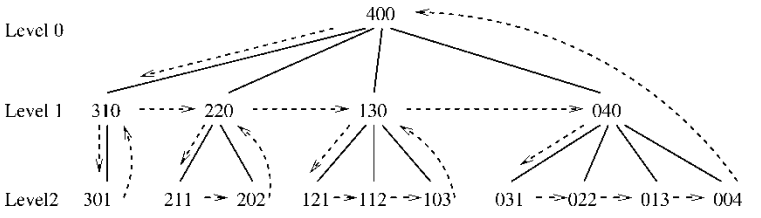

Figure 4. Illustration of generation of $D(4, 3)$ in anti-lexicographic order.

Algorithm Find-All-Distributions(n, m)
{A_r is the root sequence, S indicates the current stack }
begin
Find-All-Child-Distributions($A_r = (0, \ldots, 0, n), 0, S$);
end.

The tree traversal according to the efficient algorithm is depicted in Figure 2. For a sequence we need $O(m)$ space and additional $m/2$ space is required for stack manipulation. The value of each integer in the sequence can be at most n. Hence we need $\lg n$ bits to represent the number of objects in each bin. Thus, the memory requirement is $O(m \lg n)$ bits. One can observe that the algorithm generates all sequences such that each sequence in $T_{n,m}$ can be obtained from the preceding one by at most two operations. Thus, the algorithm generates each sequence in $O(1)$ time per sequence. Thus we have the following theorem.

THEOREM 4.2 *The algorithm* **Find-All-Distributions** *uses* $O(m \lg n)$ *space and generates each distribution in* $D(n, m)$ *in constant time* (*in ordinary sense*).

5. Conclusion

In this paper, we give a simple elegant algorithm to generate all distributions in $D(n, m)$. The algorithm generates each distribution in constant time with linear space complexity. We present a new efficient tree traversal algorithm that generates each distribution in $O(1)$ time in ordinary sense. Our algorithm also generates distributions with a specified order of generation. One can observe that in our algorithm, if we reverse the order of the bins then we will get the generation of distributions in anti-lexicographic order (see Figure 4).

Acknowledgement

We wish to thank the two anonymous referees for their valuable comments and suggestions for improving the presentation of the paper. We thank Shin-ichi Nakano for encouraging us to work in this area and for giving us valuable comments on the manuscript of the paper. We also thank Shin-ichiro Kawano for helpful discussions.

References

[1] A.V. Aho and J.D. Ullman, *Foundation of Computer Science*, Computer Science Press, New York, 1995.
[2] D. Avis and K. Fukuda, *Reverse search for enumeration*, Discrete Appl. Math. 65 (1996), pp. 21–46.
[3] T.I. Fenner and G. Loizou, *A binary tree representation and related algorithms for generating integer partitions*, Comp. J. 23 (1979), pp. 332–337.
[4] S. Kawano and S. Nakano, *Constant time generation of set partition*, IEICE Trans. Fundam. E88-A(4) (2005), pp. 930–934.
[5] P. Klingsberg, *A gray code for compositions*, J. Algorithms 3 (1982), pp. 41–44.

[6] S. Nakano and T. Uno, *Constant time generation of trees with specified diameter*, Proc. of WG 2004, LNCS 3353 (2004), pp. 33–45.
[7] A. Nijenhuis and H. Wilf, *Combinatorial Algorithms*, Academic Press, New York, 1978.
[8] K.H. Rosen, *Discrete Mathematics and Its Applications*, WCB/McGraw-Hill, Singapore, 2000.
[9] C. Savage, *A survey of combinatorial gray codes*, SIAM Rev. 39 (1997), pp. 605–629.
[10] A.S. Tanenbaum, *Computer Networks*, Prentice Hall, Upper Saddle River, New Jersey, 2002.
[11] ———, *Modern Operating Systems*, Prentice Hall, Upper Saddle River, New Jersey, 2004.
[12] K. Yamanaka, *et al.*, *Constant time generation of integer partitions*, IEICE Trans. Fundam. E-90-A (5) (2007), pp. 888–895.
[13] A. Zoghbi and I. Stojmenovic, *Fast algorithm for generating integer partitions*, Intern. J. Comput. Math. 70 (1998), pp. 319–332.