Distribution of Distinguishable Objects to Bins: Generating All Distributions
(Extended Abstract)

Muhammad Abdullah Adnan and Md. Saidur Rahman

Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology (BUET),
Dhaka-1000, Bangladesh.
{adnan,saidurrahman}@cse.buet.ac.bd

Abstract. In this paper we give an algorithm to generate all distributions of distinguishable objects to bins without repetition. Our algorithm generates each distribution in constant time. To the best of our knowledge, our algorithm is the first algorithm which generates each solution in $O(1)$ time in ordinary sense. As a byproduct of our algorithm, we get a new algorithm to enumerate all multiset partitions when the number of partitions is fixed and the partitions are numbered. In this case, our algorithm generates each multiset partition in constant time (in ordinary sense). Finally, we extend our algorithm for the case when the bins have priorities associated with them. Overall space complexity of our algorithm is $O(km)$, where there are m bins and the objects fall into k different classes.

Key words: Combinatorial Objects, Algorithm, Generating Problems, Multiset, Set Partitions.

1 Introduction

A well known counting problem in combinatorics is counting the number of ways objects can be distributed among bins [AU95,R00,AR06]. 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 fruits to distribute among them without cutting the fruits into parts. In how many ways the children receive fruits? The fruits or the objects, that we want to distribute, may be identical or of different kinds. Based on this criteria, the problem can be subdivided into two parts - identical case and non-identical case. Since the latter is more general, in this paper we will focus on the non-identical case and we call such objects as "distinguishable objects". Let there are m bins and n distinguishable objects where the objects fall into k different classes. Objects

M. Kaykobad and Md. Saidur Rahman (Eds.): WALCOM 2007, pp. 136–150, 2007.

within a class are identical with each other, but are distinguishable from those of other classes. Let n_j represent the number of objects in the jth class where $1 \leq j \leq k$. The paradigm problem is distributing different types of fruits to children. Suppose we have three apples, two pears and a banana to distribute to Kathy, Peter and Susan. Then $m = 3$, which is the number of children. There are $k = 3$ groups, with $n_1 = 3$, $n_2 = 2$, and $n_3 = 1$. Since there are 6 objects in total, so $n = 6$.

Now the question is - Can we count the number of solutions? For identical objects, the number of distributions for m bins and n identical objects is $\frac{(n+m-1)!}{n!(m-1)!}$ [AU95,R00,AR06]. We use this formula to solve the counting problem for distinguishable objects as follows. We first distribute the fruits of class 1 to all the bins. The number of such distributions with n_1 objects and m bins is $\frac{(n_1+m-1)!}{n_1!(m-1)!}$. Then we distribute the objects of second class and so on up to kth class. Thus the total number of distributions will be the product of all these solutions as in the following expression:

$$\frac{(n_1+m-1)!}{n_1!(m-1)!} . \frac{(n_2+m-1)!}{n_2!(m-1)!} \cdots \frac{(n_k+m-1)!}{n_k!(m-1)!}$$

Let $D(n, m, k)$ represents the set of all distributions of n objects to m bins where the objects fall into k different classes and each bin gets zero or more objects. For the previous example, we have $D(6, 3, 3)$ representing all distributions where the number of distribution is $\frac{5!}{3!2!} \cdot \frac{4!}{2!2!} \cdot \frac{3!}{1!2!} = 180$. Thus we count the number of distributions. However, in this paper we are not interested in counting the number of distributions, rather we are interested in generating all distributions. Generating all distributions has practical applications in channel allocation in computer networks, CPU scheduling, memory management, etc. [AR06,T02,T04]. In these days of automation, machines may also require to distribute objects among candidates optimally.

Generating the complete list of all solutions of such combinatorial problems has many useful applications. For example, 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 analyze an algorithm for its correctness or computational complexity. Early works in combinatorics focused on counting; because generating all objects requires huge computation. With the aid of fast computers it now has become feasible to list the objects in combinatorial classes. However, in order to generate entire list of objects from a class of moderate size, extremely efficient algorithms are required even with the fastest computers. Due to the reason mentioned above, recently many researchers have concentrated their attention for developing efficient algorithms to generate all objects of a particular class without repetitions [K06]. Examples of such exhaustive generation of combinatorial objects include generating all integer partition and set partitions, enumerating all binary trees, generating permutations and combinations, enumerating spanning trees, etc. [AR06,KN05,ZS98,NU03,YN04,FL79,NU04,NU05,BS94,S97,K06]. Generally, generating algorithms produce huge outputs, and the outputs dominate the running time of the generating algorithms. Therefore, many generating algorithms output solutions in an order such that each solution differs from the preceding one by a very small amount, and output each solution as the "difference" from the preceding one. Such orderings of solutions are known as *Gray codes* [S97,KN05,AR06].

Klingsberg [K82] gave an average constant time algorithm for sequential listing of the composition of an integer n into k parts. Using efficient tree traversal technique, Adnan and Rahman [AR06] improved the time complexity to constant time (in ordinary sense) and gave an efficient algorithm to generate all distribution of objects to bins when the objects are identical. They used efficient generation method based on the family tree structure of the distributions. However, in this paper we are interested in generating all distributions of distinguishable objects. This problem is more difficult than that of the identical case since the solution space is large. If we apply the algorithm for identical objects for distinguishable objects, there will be omission of distributions. Hence the algorithm for generating identical objects is not applicable for distinguishable objects.

The problem of generating all distribution of distinguishable objects can be viewed as generating multiset partitions when 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. Kawano and Nakano [KN06] gave an algorithm to generate multiset partition but the algorithm does not give solutions in constant time in ordinary sense. Using the algorithm [KN05] for set partition their algorithm [KN06] constructs a family tree for each type of element. Then they combine the solutions of each family tree to output each multiset partition. Since there are k family trees for k types of elements in the set, their algorithm takes $O(k)$ time to generate each solution. Their method is not applicable here since the partitions are fixed, ordered and numbered. Also we need to construct only one family tree for all solutions. Hence there is no need for recombination and our algorithm generates each solution in constant time for fixed, numbered and ordered partitions.

In this paper we give an algorithm to generate all distributions of n distinguishable objects to m bins where the objects fall into k different classes. Here, the number of bins is fixed and the bins are numbered and ordered. Our algorithm generates each distribution in constant time without repetition. The main feature of our algorithm is that we define a tree structure, that is parent-child relationships, among the distributions in $D(n, m, k)$ (see 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 relationship 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 $D(n, m, k)$ 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 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: (i) Relationship between left sibling and right sibling and (ii) 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 [S97,KN05,R00] for distributions. Our algorithm also generates the distributions *in place*, that means, the space complexity is linear. To the best of our knowledge, our algorithm is the first algorithm to generate all distribution in constant time per distribution in ordinary sense. We also extend our algorithm for the case when the bins have priorities associated with them. In this case, the bins are numbered in the order of priority. The sequence of generations maintain an order so that the successive generations maintain priority.

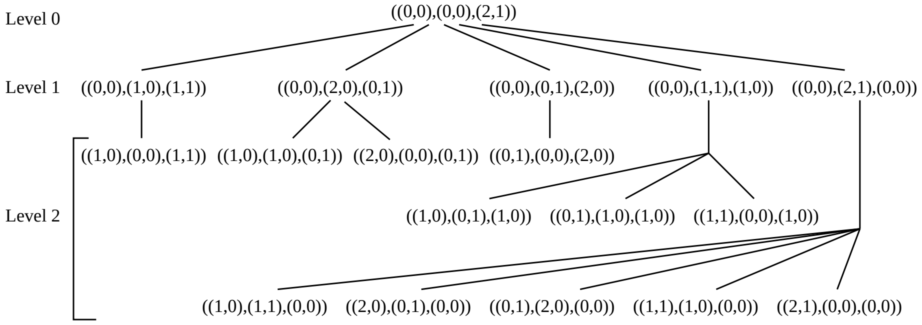


Fig. 1. The Family Tree $T_{3,3,2}$.

The rest of the paper is organized as follows. Section 2 gives some definitions. In Section 3, we define a tree structure among distributions in $D(n, m, k)$. In Section 4, we present the algorithm for generating all distributions of distinguishable objects to bins. Finally Section 5 is a conclusion.

2 Preliminaries

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

In mathematics and computer science, 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. Each vertex in a tree is either an internal vertex or a leaf. 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 $c_1, c_2, \dots, c_l$ 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. The *descendants* of a vertex v are those vertices that have v as an ancestor. A leaf in a family tree has no children.

For a positive integer n and $k < n$, *set partition* is the set of all partitions of $\{1, 2, \dots, n\}$ into k non-empty subsets. For instance, for $n = 4$ and $k = 2$ there are seven such partitions:

$$\{1,2,3\} \cup \{4\}, \{1,2,4\} \cup \{3\}, \{1,3,4\} \cup \{2\}, \{2,3,4\} \cup \{1\}, \{1,2\} \cup \{3,4\},$$
$$\{1,3\} \cup \{2,4\}, \{1,4\} \cup \{2,3\}.$$

A *simple set* is a collection of elements where all the elements are identical. A *multiset* is a collection of elements where all the elements are not identical. The elements of a multiset fall into different classes where the elements in the same class are identical but are distinguishable from those of other classes. For example, {1,1,2,3,1,3,2,2} is an example of multiset.

For positive integer k, let a be a sequence of positive integers $t_1, t_2, \dots, t_k$ where $t_j \geq 0$ for $1 \leq j \leq k$. We call a as *zero sequence* if $t_1 = t_2 = \cdots = t_k = 0$. That means all the integers in a zero sequence are 0. We call a as *nonzero sequence* if there exists an index j, $1 \leq j \leq k$, such that $t_j \neq 0$. That means a sequence is nonzero if at least one of the integers in the sequence is nonzero. Let b be another sequence of positive integers $u_1, u_2, \dots, u_k$ for $1 \leq j \leq k$. By addition of two sequences $a + b$ we mean the addition of corresponding elements $t_j + u_j$ where $1 \leq j \leq k$. Similarly, by subtraction of two sequences $a - b$ we mean the subtraction of corresponding elements $t_j - u_j$ where $1 \leq j \leq k$ and by equality of two sequences $a = b$ we mean the equality of corresponding elements $t_j = u_j$ where $1 \leq j \leq k$.

A listing of combinatorial objects is said to be in *gray code order* if each successive combinatorial objects in the listing differs by a constant amount. For example, 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.

For positive integers n, m and k, let $A \in D(n, m, k)$ be a distribution of n objects to m bins where the objects fall into k classes. Let n_j represent the number of objects in the jth class where $1 \leq j \leq k$. Clearly, $n_1 + n_2 + \cdots + n_k = n$ since every object must be in a class. The bins are ordered and numbered as $B_1, B_2, \dots, B_m$. Each bin contains objects of different classes. We order the different types of objects in a bin so that we can keep track of objects of different classes. Let a_i be a sequence of positive integers $(t_{i1}, t_{i2}, \dots, t_{ik})$ where t_{ij} represents the number of objects of jth type in ith bin B_i, for $1 \leq i \leq m$, $1 \leq j \leq k$. Then we can represent each $A \in D(n, m, k)$ by a unique sequence

$(a_1, a_2, \ldots, a_m)$. We call each a_i an inner sequence of A. The sequence for A is unique for each distribution because the bins are ordered and numbered and also the objects of different types are ordered. For example, the sequence $((0,0),(2,1))$ represents there are 2 bins because there are 2 sequence of sequences of integers and 3 objects which is sum of all the integers in the sequence and there are 2 classes of objects where 2 objects are from class 1 and 1 object from class 2. Also the second bin contains 3 objects i.e. 2 objects from class 1 and 1 object from class 2 and the first bin is empty (see Figure 2).

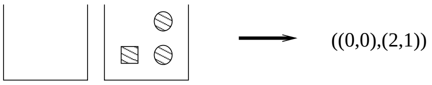


Fig. 2. Representation of a distribution of 3 objects to 2 bins where the objects fall into two classes and 2 objects from class 1 and 1 object from class 2.

For each such sequence of sequences in $D(n,m,k)$, we have the following equations:

$$\sum_{i=1}^{m}\sum_{j=1}^{k} t_{ij} = n, \tag{1}$$

$$\sum_{i=1}^{m} t_{ij} = n_j, \qquad for\ 1 \leq j \leq k,\ and \tag{2}$$

$$\sum_{j=1}^{k} n_j = n. \tag{3}$$

Equation 1 describes that the sum of all the integers in the sequence of sequences in equal to the total number of objects. This holds because the number of objects are fixed and every object is distributed somewhere in some bin. In Equation 2, we describe that every object of same kind are present in the sequence. Since every object must be in a class Equation 3 holds.

In the following sections we give an algorithm to generate all distributions of distinguishable objects to bins. For that purpose we define a unique parent-child relationship among the distributions in $D(n,m,k)$ 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 $D(3,3,2)$. Once such a parent-child relationship is established, we can generate all the distributions in $D(n,m,k)$ 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 $D(n, m, k)$ 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

In this section we define a tree structure $T_{n,m,k}$ among the distributions in $D(n, m, k)$.

For positive integers n, m and k, let $A \in D(n, m, k)$ be a distribution of n objects to $B_1, B_2, \dots, B_m$ bins where the objects fall into k classes. Let n_j represent the number of objects in the jth class where $1 \leq j \leq k$. From Equation 3, $n_1 + n_2 + \cdots + n_k = n$. For each $A \in D(n, m, k)$, we define a unique sequence of sequences of positive integers $(a_1, a_2, \dots, a_m)$, where a_i represents a sequence of integers $(t_{i1}, t_{i2}, \dots, t_{ik})$ where t_{ij} represents the number of objects of jth type in ith bin B_i, for $1 \leq i \leq m$, $1 \leq j \leq k$.

Now we define the family tree $T_{n,m,k}$ as follows. Each node in $T_{n,m,k}$ represents a distribution in $D(n, m, k)$. If there are m bins then there are m levels in $T_{n,m,k}$. A node is in level i, $0 \leq i < m$ in $T_{n,m,k}$ if $t_{lj} = 0$ for $1 \leq j \leq k$, $1 \leq l < (m - i)$ and $a_{(m-i)}$ is nonzero sequence. So, a node at level $m - 1$ has no leftmost inner zero sequence before leftmost inner nonzero sequence. As the level increases the number of leftmost inner zero sequence decreases and vice versa. Since the family tree 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 $T_{n,m,k}$ if $t_{lj} = 0$ for $1 \leq j \leq k$, $1 \leq l < (m)$ and $t_{mj} \neq 0$ for $1 \leq j \leq k$. We also have from Equation 1 that $\sum_{i=1}^{m} \sum_{j=1}^{k} t_{ij} = n$. Substituting the values for t_{lj} for $1 \leq j \leq k$, $1 \leq l < (m)$ we find that $\sum_{j=1}^{k} t_{mj} = n$. By using Equation 2 and Equation 3, we get $t_{mj} = n_j$ where $1 \leq j \leq k$. Thus we can say that there can be exactly one such node which is our root. So, the sequence for root is $((0, \dots, 0), (0, \dots, 0), \dots, (0, \dots, 0), (n_1, n_2, \dots, n_k))$. In other words, we can say that the number of leftmost inner zero sequence before any inner nonzero sequence in root is greater than any other sequence for any distribution in $D(n, m, k)$.

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

(a) Child-Parent Relationship

It is convenient to consider the child-parent relationship before the parent-child relationship. Let $A \in D(n, m, k)$ be a sequence of sequences $(a_1, a_2, \dots, a_m)$ which is not a root sequence, where a_l represents a sequence of integers t_{lj} for $1 \leq j \leq k$, $1 \leq l \leq m$. The sequence A corresponds to a node of level i, $1 \leq i < m$. So, we have $t_{lj} = 0$ for $1 \leq j \leq k$, $1 \leq l < (m - i)$ and $a_{(m-i)}$ is a nonzero sequence. Let $A' \in D(n, m, k)$ be another sequence of sequences (p_1, p_2, $\dots$, p_{m-i}, p_{m-i+1}, $\dots$,p_m), $1 \leq i < m$ such that $p_1 = p_2 = \cdots = p_{m-i}$ are zero sequences and $p_{m-i+1} = a_{m-i} + a_{m-i+1}$ and $p_l = a_l$ for $m-i+1 < l \leq m$. Then A' is at level $i-1$ of $T_{n,m,k}$. We call the sequence A' as the *parent sequence* of A. Thus for each consecutive level we only deal with two sequences a_{m-i-1} and a_{m-i} and the rest of the sequences remain unchanged. The number of leftmost inner zero sequence increases in the parent sequence by applying child-parent relationship. For example, the solution $((1,1),(1,0))$, for $n=3$, $m=2$, $k=2$ and $n_1=2$, $n_2=1$, is a node of level 1 because a_1 is a nonzero sequence. It has a unique parent $((0,0),(2,1))$ as shown in Figure 3.

(b) Parent-Child Relationship

The parent-child relationship is just the reverse of child-parent relationship. Let $A \in D(n,m,k)$ be a sequence $(a_1, a_2, \ldots, a_m)$, where a_l represents a sequence of integers t_{lj} for $1 \leq j \leq k$, $1 \leq l \leq m$. The sequence A corresponds to a node of level i, $0 \leq i < m$. So, we have $t_{lj} = 0$ for $1 \leq j \leq k$, $1 \leq l < (m-i)$ and $a_{(m-i)}$ is a nonzero sequence. Let $A' \in D(n,m,k)$ be another sequence of sequences (c_1, c_2, …, c_{m-i-1}, c_{m-i}, …, c_m), $0 \leq i < m$ such that $c_1, c_2, \ldots, c_{m-i-2}$ are zero sequences, $c_{m-i-1} + c_{m-i} = a_{m-i}$, c_{m-i-1} is a nonzero sequence and $c_l = a_l$ for $m-i+1 \leq l \leq m$. Then A' is at level $i+1$ of $T_{n,m,k}$. We call the sequence A' as the *child sequence* of A. Like the child-parent relationship here we also deal with only two inner sequences a_{m-i-1} and a_{m-i} and the rest of the sequences remain unchanged. Hence from the child-parent relationship, one can observe that the number of children of A is equal to $(\prod_{j=1}^{k}(t_{(m-i)j}+1)) - 1$. The number of leftmost zero sequence decreases in the child sequence by applying parent-child relationship. For example, the solution $((0,0),(2,1))$, for $n=3$, $m=2$, $k=2$ and $n_1=2$, $n_2=1$, is a node of level 0 because a_1 is a zero sequence, a_2 is not a zero sequence. Here, $(\prod_{j=1}^{k}(t_{(m-i)j}+1)) - 1 = (2+1).(1+1) - 1 = 5$ so it has 5 children and the five children are ((1,0),(1,1)), ((2,0),(0,1)), ((0,1),(2,0)), ((1,1),(1,0)) and ((2,1),(0,0)) as shown in Figure 3.

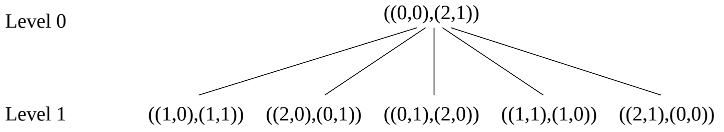


Fig. 3. The sequence $((0,0),(2,1))$ has five children.

From the above definitions we can construct the family tree $T_{n,m,k}$. We take the sequence $A_r = a_1, a_2, \ldots, a_m$ as root where $a_1, a_2, \ldots, a_{m-1}$ are zero sequences and $a_m = (n_1, n_2, \ldots, n_k)$ as we mentioned before. The family tree $T_{n,m,k}$ for the distributions in $D(n,m,k)$ is shown in Figure 1. Based on the above parent-child relationship, the following lemma proves that every distribution in $D(n,m,k)$ is present in $T_{n,m,k}$.

Lemma 1. *For any distribution $A \in D(n,m,k)$, there is a unique sequence of distributions that transforms A into the root A_r of $T_{n,m,k}$.*

Proof. Let $A \in D(n, m, k)$ be a sequence, where A is not the root sequence. We determine the level of A in the family tree $T_{n,m,k}$. Then by applying child-parent relationship, we find the parent sequence $P(A)$ of A. Now 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 $D(n, m, k)$ which eventually ends with the root sequence A_r of $T_{n,m,k}$. 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. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

Lemma 1 ensures that there can be no omission of distributions in the family tree $T_{n,m,k}$. Since there is a unique sequence of operations that transforms a distribution $A \in D(n, m, k)$ into the root A_r of $T_{n,m,k}$, by reversing the operations we can generate that particular distribution, staring from root. We now have to make sure that the family tree $T_{n,m,k}$ represents distributions without repetition. Based on the parent-child and child-parent relationships, the following lemma proves this property of $T_{n,m,k}$.

Lemma 2. *The family tree $T_{n,m,k}$ represents distributions in $D(n, m, k)$ without repetition.*

Proof. Given a sequence $A \in D(n, m, k)$, the children of A are defined in such a way that no other sequence in $D(n, m, k)$ can generate same child. Let $A, B \in D(n, m, k)$ be two different sequences at level i of $T_{n,m,k}$. For a contradiction, assume that A and B generate the same child C. Then C is a sequence of level $i + 1$ of $T_{n,m,k}$. The sequences for A, B and C are a_j, b_j and c_j for $1 \le j \le m$. Clearly, a_l=b_l for $1 \le l \le m - i - 1$ and the parent-child relationship yields $a_l = b_l = c_l$ for $m-i+1 \le l \le m$. Therefore $a_l = b_l$ for $l \ne m-i$ and $1 \le l \le m$. But we have $a_1 + a_2 + \cdots + a_m = b_1 + b_2 + \cdots + b_m$ by Equation 1. Then a_l must be equal to b_l, for $1 \le l \le m$. This implies that A and B are the same sequence, a contradiction. Hence every sequence has a single and unique parent. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

4 Generating Distributions

In this section, we give an algorithm to construct $T_{n,m,k}$ and generate all distributions.

One can use the parent-child relationships described in the previous section to construct the family tree $T_{n,m,k}$ and hence generate all the distributions in $D(n, m, k)$ by traversing the tree using the relationships. If we can generate all child sequences of a given sequence in $D(n, m, k)$, then in a recursive manner we can construct $T_{n,m,k}$ and generate all sequence in $D(n, m, k)$. We have the root sequence $A_r = ((0, \ldots, 0), (0, \ldots, 0), \ldots, (0, \ldots, 0), (n_1, n_2, \ldots, n_k))$. We get the child sequence A_c by using the parent to child relation discussed above.

```
Procedure Find-All-Child-Distributions($A = ( ( t_{11}, t_{12}, \ldots, t_{1k} ), ( t_{21}, t_{22}, \ldots, t_{2k} ), \ldots, ( t_{m1}, t_{m2}, \ldots, t_{mk} ) ), i$)
{$A$ is the current sequence, $i$ indicates the current level, $A_c$ is the child sequence }
begin
  Output $A$ {Output the difference from the previous distribution}
  for $i_k = 0$ to $t_{(m-i)k}$
    for $i_{k-1} = 0$ to $t_{(m-i)(k-1)}$
      ...
        for $i_1 = 0$ to $t_{(m-i)1}$
          Find-All-Child-Distributions( $A_c = ( ( t_{11}, t_{12}, \ldots, t_{1k} ), ( t_{21}, t_{22}, \ldots, t_{2k} ), \ldots, ( t_{(m-i-2)1}, t_{(m-i-2)2}, \ldots, t_{(m-i-2)k} ), ( i_1, i_2, \ldots, i_k ), ( t_{(m-i)1} - i_1, t_{(m-i)2} - i_2, \ldots, t_{(m-i)k} - i_k ), \ldots, ( t_{m1}, t_{m2}, \ldots, t_{mk} ) ), i + 1$);
end;
Algorithm Find-All-Distributions($n, m$)
begin
  Find-All-Child-Distributions( $A_r = ( (0,\ldots, 0), (0,\ldots, 0),\ldots, (0,\ldots, 0), (n_1, n_2,\ldots, n_k) ), 0$ );
end.
```

Lemma 1 and 2 ensure that Algorithm **Find-All-Distributions** generates all distributions without repetition. We now have the following lemma, whose proof is omitted in this extended abstract.

Lemma 3. *The algorithm* **Find-All-Distributions** *uses $O(mk)$ space and runs in $O(|D(n, m, k)|)$ time.*

The algorithm **Find-All-Distributions** generates all sequences in $D(n, m, k)$ in $O(|D(n, m, k)|)$ time. Thus the algorithm generates each sequence in $O(1)$ time "on average". However, after generating a sequence corresponding to the last vertex in the largest level in a large subtree of $T_{n,m,k}$, we have to merely return from the deep recursive call without outputting any sequence and hence we cannot generate each sequence in $O(1)$ time (in ordinary sense). To generate each distribution in $O(1)$ time (in ordinary sense), we introduce two additional types of relationships:

(i) Relationship between left sibling and right sibling and
(ii) Leaf-ancestor relationship.

(i) Relationship Between Left Sibling and Right Sibling

Let $A \in D(n, m, k)$ be a sequence of sequences $(a_1, a_2, \ldots, a_m)$ which is not a root sequence, where a_l represents a sequence of integers t_{lj} for $1 \le j \le k$, $1 \le l \le m$. The sequence A corresponds to a node of level i, $0 \le i < m$. So, we have $t_{lj} = 0$ for $1 \le j \le k$, $1 \le l < (m - i)$ and $a_{(m-i)}$ is a nonzero sequence. We say the *right sibling sequence*, $A_s \in D(n, m, k)$ of this node A exists if $a_{(m-i+1)}$ is a nonzero sequence at level i. Then we call the sequence A *left sibling* of A_s.

We define the sequence for A_s as $s_1, s_2, \ldots, s_{m-i}, s_{m-i+1}, \ldots, s_m, 1 \le i < m$ where s_1, s_2, …, s_{m-i-1} are zero sequences and $s_j = a_j$ for $m - i + 2 \le j \le m$ and to find s_{m-i}, s_{m-i+1} we apply child-parent relationship and then parent-child relationship. Thus, we observe that A_s is a node of level i, $1 \le i < m$

and so $s_1, s_2, \ldots, s_{m-i-1}$ are zero sequences and s_{m-i} is nonzero sequence for $1 \leq i < m$. For example, the solution $((0,0),(1,0),(1,1))$, for $n = 3$, $m = 3$, $k = 2$ and $n_1 = 2$, $n_2 = 1$, is a node of level 1 because a_1 is a nonzero sequence. It has a unique right sibling $((0,0),(2,0),(0,1))$ as shown in Figure 4.

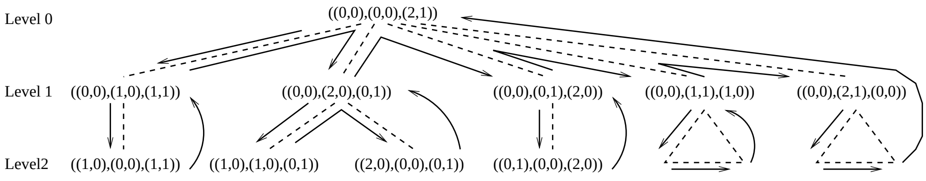


Fig. 4. Efficient Traversal of the family tree $T_{3,3,2}$.

(ii) Leaf-Ancestor Relationship

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. Another reason of defining leaf-ancestor relationship is that the nearest ancestor can be generated from the leaf sequence by just a simple swap operation between two inner sequences in the sequence. This is possible due to the data structure that we use for this case (as described in the following subsection). For the swap operation we just swap the pointers to sequences. The other inner sequences remain unchanged.

Let $A_l \in D(n,m,k)$ be a sequence of sequences $(a_1, a_2, \ldots, a_m)$ of leaf, where a_p represents a sequence of integers t_{pj} for $1 \leq j \leq k$, $1 \leq p \leq m$. The sequence A_l corresponds to a node of level $m-1$. So, we have $a_{(m-i)}$ is a nonzero sequence. We say that the *ancestor sequence* $A_a \in D(n,m,k)$ of this node A_l exists if a_2 is a zero sequence that is it has no right sibling. We define a unique ancestor sequence of A_l at level $m-1-q$ where $a_2, a_3, \ldots, a_{q+1}$ are zero sequence and a_{q+2} is a nonzero sequence. This means we want to skip the long sequence of inner zero sequence in the sequence for A_l. The nearest ancestor sequence is determined by the number of zero sequence in this sequence. We denote the number of inner zero sequence as q. This q will determine the level and sequence of the nearest ancestor A_a which has sibling.

We define the sequence for A_a as $s_1, s_2, \ldots, s_q, s_{q+1}, \ldots, s_m$, where s_1, s_2, $\ldots$,s_q are zero sequence and $s_{q+1} = a_1$ and $s_j = a_j$ for $q+1 < j \leq m$. In other words, we just swap the sequences a_1 and a_{q+1} in the sequence and the rest of the inner sequences remain unchanged. For example, in Figure 4 the solution ((2,0),(0,0),(0,1)), for $n = 3$, $m = 3$, $k = 2$ and $n_1 = 2$, $n_2 = 1$, is a node of level 2 because a_1 is a nonzero sequence. It has a unique ancestor $((0,0),(2,0),(0,1))$ which is obtained by swapping first and second sequences. We have the following lemma on uniqueness of the nearest ancestor A_a of A_l, whose proof is omitted in this extended abstract.

Lemma 4. *Let A_l be a leaf sequence of $T_{n,m,k}$ having no right sibling. Then A_l has a unique ancestor sequence A_a in $T_{n,m,k}$. Furthermore, either A_a has a right sibling in $T_{n,m,k}$ or A_a is the root A_r of $T_{n,m,k}$.*

Lemma 4 ensures that A_l has a unique ancestor A_a. As we see later A_a plays an important role in our algorithm. Now we present the algorithm to generate all distributions in $D(n, m, k)$. We use three relations in this algorithm; they are parent-child relation, relation between left sibling and right sibling and leaf-ancestor relation. By applying parent-child relation, we go from root down the family tree $T_{n,m,k}$ 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 relation, we return to that nearest ancestor which has sibling. Then we again apply relation 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-Distributions2( $A = ( ( t_{11}, t_{12}, \ldots, t_{1k} ), ( t_{21}, t_{22}, \ldots, t_{2k} ), \ldots, ( t_{m1}, t_{m2}, \ldots, t_{mk} ) )$ , $i$)
{ $A$ is the current sequence }
begin
  Output $A$ {Output the difference from the previous distribution}
  if $A$ has child then
    begin
      Generate the first child $A_c$ of $A$;
      Find-All-Child-Distributions2($A_c$, $i + 1$);
    end
  else
    if $A$ has right sibling then
      begin
        Generate right sibling $A_s$;
        Find-All-Child-Distributions2($A_s$, $i$);
      end
    else
      begin
        Generate the ancestor $A_a$ of $A$ at level $i - q$ such that
        either $A_a$ has right sibling or $A_a$ is root;
        if $A_a$ is the root at level 0
          then done
        else
          begin
            Generate right sibling $A_{as}$ of $A_a$;
            Find-All-Child-Distributions2($A_{as}$, $i - q$);
          end
      end
```

```
end;
Algorithm Find-All-Distributions2($n, m$)
begin
  Find-All-Child-Distributions2( $A_r$ = ( (0,. . ., 0), (0,. . ., 0), . . ., (0,. . .,0), ($n_1$,
$n_2$,. . ., $n_k$) ), 0 );
end.
```

The tree traversal according to the efficient algorithm is depicted in Figure 4. Now we describe the data structure that we use to represent a distribution in $D(n, m, k)$ that will help us to generate each distribution in constant time. 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 is a nonzero sequence. A_a is obtained from A_l by swapping a_1 and a_{q+1} where q is the number of consecutive zero sequence after a_1. Now, to find out q we have to search the sequence A_l from a_1 to a_{q+1} such that $a_2, a_3, \ldots, a_{q+1}$ are inner zero sequence and a_1, a_{q+2} are inner nonzero sequence. We reduce the complexity of searching by keeping extra information as shown in Figure 5. The information consists of the number of subsequences of consecutive inner zero sequence and the number of inner zero sequence 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 q. Initially the stack is empty. As soon as we find a zero sequence, 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 for consecutive zero sequence. We make a pop operation when we apply the leaf-ancestor relation. The stack operations are shown in Figure 5. One can observe that there can be at most $m/2$ subsequences of consecutive inner zero sequence in a sequence of size m. Therefore, in worst case we need a stack of size $m/2$.

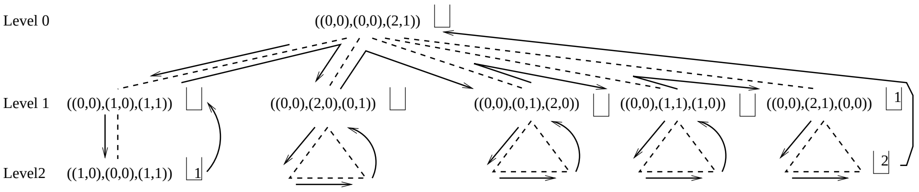


Fig. 5. Efficient Traversal of $T_{4,3}$ keeping extra information.

The operations that we use to generate distributions are addition, subtraction, increment, decrement and swap. The index of the two operands for these operations are known. So, we might think of keeping an array of integers. Since for distinguishable objects we deal with sequence of sequences, we may want to use array of array of integers that means two-dimensional array of integers. But note that for applying leaf-ancestor relationship we need to swap entire sequence of integers. By keeping 2D-array of integers it will take $O(k)$ time to swap such array of integer. This is not efficient. To do swap operation in constant time, we use a special data structure as shown in Figure 6. We keep an array of pointers for each bins pointing to an array of integers. The array of integers represents the inner sequence that is the sequence of different types of objects in a particular bin. The structure may be viewed as array of objects where each object is an array of integers in object-oriented sense. Thus by swapping the pointers we will be able to swap entire array in $O(1)$ time.

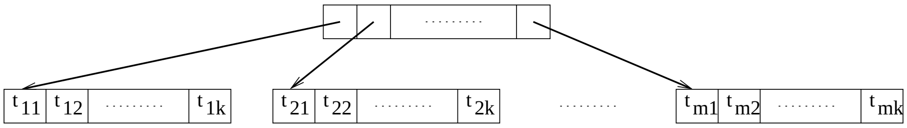
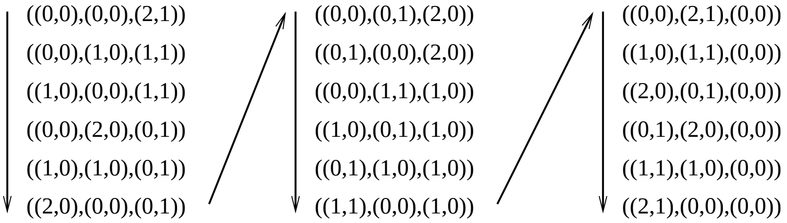


Fig. 6. Illustration of data structure that we use to represent a distribution for distinguishable objects.

Using the data structure mentioned above one can efficiently implement the algorithm **Find-All-Distributions2** and hence the following theorem holds. The detail is omitted.

Theorem 1. *The algorithm* **Find-All-Distributions2** *uses $O(mk)$ space and generates each distribution in $D(n, m, k)$ in constant time (in ordinary sense).*



((0,0),(0,0),(2,1))
((0,0),(1,0),(1,1))
((1,0),(0,0),(1,1))
((0,0),(2,0),(0,1))
((1,0),(1,0),(0,1))
((2,0),(0,0),(0,1))

((0,0),(0,1),(2,0))
((0,1),(0,0),(2,0))
((0,0),(1,1),(1,0))
((1,0),(0,1),(1,0))
((0,1),(1,0),(1,0))
((1,1),(0,0),(1,0))

((0,0),(2,1),(0,0))
((1,0),(1,1),(0,0))
((2,0),(0,1),(0,0))
((0,1),(2,0),(0,0))
((1,1),(1,0),(0,0))
((2,1),(0,0),(0,0))

Fig. 7. A Gray code for $D(3, 3, 2)$.

5 Conclusion

In this paper we give a simple elegant algorithm to generate all distributions in $D(n, m, k)$. The algorithm generates each distribution in constant time with linear space complexity. We also present an efficient tree traversal algorithm that generates each solution in $O(1)$ time. Note that each sequence is similar to the preceding one, since it can be obtained by at most two operations. Thus, we can regard the derived sequence of the sequences as a combinatorial Gray code [S97,KN05,R00] for distributions (see Figure 7). Our algorithm can also be extended for the case when the bins have priorities associated with them. In this case, the bins are numbered in the order of priority. The sequence of generations will maintain an order such that the bin with highest priority gets highest number of objects at first and then the priorities of the bins are decreased one by one. Thus the sequence of generations maintain an order so that the generations maintain priority. The main feature of our algorithms is that they are constant time solution which is a very important requirement for generation problems.

References

[AR06] M. A. Adnan and M. S. Rahman, *Distribution of objects to bins: generating all distributions*, Proc. of International Conference on Computer and Information Technology (ICCIT'06), 2006 (to appear).

[AU95] A. V. Aho and J. D. Ullman, *Foundation of Computer Science*, Computer Science Press, New York, 1995.

[BS94] M. Belbaraka and I. Stojmenovic, *On generating B-trees with constant average delay and in lexicographic order*, Information Processing Letters, 49, pp. 27-32, 1994.

[FL79] T. I. Fenner and G. Loizou, *A binary tree representation and related algorithms for generating integer partitions*, The Computer Journal, 23, pp. 332-337, 1979.

[K06] D. E. Knuth, *The Art of Computer Programming*, Vol.4, url: http://www.cs.utsa.edu/ wagner/knuth/, 2006.

[K82] P. Klingsberg, *A gray code for compositions*, Journal of Algorithms, 3, pp. 41-44, 1982.

[KN05] S. Kawano and S. Nakano, *Constant time generation of set partition*, IEICE Trans. Fundamentals, E88-A, 4, pp. 930-934, 2005.

[KN06] S. Kawano and S. Nakano, *Generating Multiset Partitions*, (on private communication), 2006.

[NU03] S. Nakano and T. Uno, *Efficient generation of rooted trees*, NII Technical Report, NII-2003-005E, July 2003.

[NU04] S. Nakano and T. Uno, *Constant time generation of trees with specified diameter*, Proc. of WG 2004, LNCS 3353, pp. 33-45, 2004.

[NU05] S. Nakano and T. Uno, *Generating colored trees*, Proc. of WG 2005, LNCS 3787, pp. 249-260, 2005.

[R00] K. H. Rosen, *Discrete Mathematics and Its Applications*, WCB/McGraw-Hill, Singapore, 2000.

[S97] C. Savage, *A survey of combinatorial gray codes*, SIAM Review, 39, pp. 605-629, 1997.

[T02] A. S. Tanenbaum, *Computer Networks*, Prentice Hall, Upper Saddle River, NJ, 2002.

[T04] A. S. Tanenbaum, *Modern Operating Systems*, Prentice Hall, Upper Saddle River, NJ, 2004.

[YN04] K. Yamanaka and S. Nakano, *Generating all realizers*, IEICE Trans. Inf. and Syst., J87-DI, 12, pp. 1043-1050, 2004.

[ZS98] A. Zoghbi and I. Stojmenovic, *Fast algorithm for generating integer partitions*, International Journal of Computer Mathematics, 70, pp. 319-332, 1998.