社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2006－AL－104　(5)
2006／1／20

整数分割の列挙

山中 克久[†] 川野 晋一郎[†] 菊地 洋右[‡] 中野 眞一[†]

概要　本文では，正の整数 n の整数分割を列挙するアルゴリズムを与える．この問題は，組み合わせ論において基本的な問題の１つであり，長い間，広く研究されてきた．これまで，整数分割を１つ当たり平均定数時間で列挙する方法しか知られていなかった．我々は，与えられた整数の整数分割を，最悪でも１つ当たり定数時間で重複なく列挙するアルゴリズムを提案する．また，条件付きの整数分割を定数時間で列挙するアルゴリズムをいくつか与える．

Constant Time Generation of Integer Partitions

Katsuhisa YAMANAKA[†], Shin-ichiro KAWANO[†],
Yosuke KIKUCHI[‡], and Shin-ichi NAKANO[†]

abstract In this paper we give a simple algorithm to generate all partitions of a positive integer n. The problem is one of basic problems in combinatorics, and have been extensively studied for a long time. Our algorithm generates each partition of a given integer in constant time for each without repetition, while known best algorithm generates each partition in constant time on "average". Also, we propose some algorithms to generate all partitions of an integer with some property in constant time.

1 Introduction

It is useful to have the complete list of objects for a particular class. One can use such a list to search for a counter-example to some conjecture, to find the best object among all candidates, or to experimentally measure an average performance of an algorithm over all possible inputs.

Many algorithms to generate all objects in a particular class, without repetition, are already known [B80, LN01, LR99, M98, N02, R78, W86]. Many excellent textbooks have been published on the subject [G93, KS98, K05, W89].

In this paper we consider the following generation problem. For a positive integer n, a partition of n is a sequence $a_1a_2\dots a_m$ of nonnegative integers $a_1 \geq a_2 \geq \cdots \geq a_m$ such that $n = a_1 + a_2 + \cdots + a_m$. Let $S(n)$ be the set of all partitions of n. For instance, for $n = 5$ there are seven such partitions: 5, 41, 311, 2111, 11111, 32, 221. Thus $|S(5)| = 7$.

Actually we have two representations for a partition. The representation above is called *the standard representation. The multiplicity representation* denotes a partition by each distinct integer with the number of occurrences. For example 22111 in the standard representation is $2\times2+1\times3$ in the multiplicity representation.

To generate all partitions of an integer is one of the basic problems in combinatorics, since that arise frequently in practice [A93, A76, F80, N71, Z98].

The number $|S(n)|$ of partitions of an integer n is given by the Hardy-Ramanujan-Rademacher asymptotic formula [A76, p.70].

$$|S(n)| \sim \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{\frac{2n}{3}}}$$

Many algorithms to generate all partitions of an integer have been proposed. See [A93, A76, F80, F81, M65, M70, N75, P79, R95, R77, S89, Z96, Z98]. Among them, the algorithms given in [R95, Z98] is the most efficient. These algorithms generate all partitions of an integer in standard representation in constant time for each "on average", without the output time. To design an algorithm to generate all partitions in standard representation in constant time for each (in worst case) is still open [Z98].

On the other hand, all integer partitions of an integer in multiplicity representation can be generated in constant time for each without the output time [A93, F81, N75].

In this paper we improve the best known results [R95, Z98] and solve the open problem. Our algorithm generates all partitions of an integer in standard representation in constant time for each in worst case.

Therefore, the derived sequence of partitions of an integer is a kind of combinatorial Gray code [J80, R93, S97, W89] for partitions of an integer. A Gray code [R00] is a cyclic sequence of all 2^k bitstrings of length k, such that each bitstring differs from the preceding one in a small number of bit entries. A combinatorial Gray code is a generalization of the Gray code.

In [Z98] two algorithms are proposed. They generate all partitions of an integer in lexicographic and anti-lexicographic order, respectively.

† 群馬大学大学院工学研究科電子情報工学専攻
Faculty of Engineering, Department of Computer Science, Gunma University
E-mail:{yamanaka, kawano, nakano}@msc.cs.gunma-u.ac.jp
‡ 独立行政法人科学技術振興機構今井量子計算機構プロジェクト
ERATO Quantum Computation and Information Project, JST
E-mail: kikuchi@qci.jst.go.jp

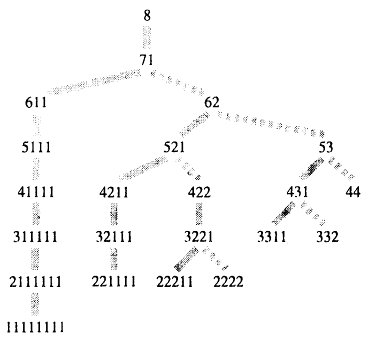

Figure 1: The family tree T_8.

On the other hand, our idea is quite different from theirs. The main idea of our algorithm is as follows. We first define a rooted tree (See Fig. 1) such that each vertex corresponds to a partition in $S(n)$, and each edge corresponds to a relation between two partitions. Then with traversing the tree we generate all partitions in $S(n)$. Note that our algorithm outputs each partition as the difference from the preceding one. With a similar technique we have already solved some generation problems for graphs [LN01, N01, N02, NU04], and set partitions [KN05]. This paper extends the technique for integer partitions.

In this paper we first give a simple algorithm to generate all partitions of an integer n. Our first algorithm generates each partition in constant time. By slightly modifying the algorithm, we also give four more algorithms to generate all partitions of an integer with some property. Given an integer k, our second algorithm generates all partitions into at most k parts, and our third algorithm generates all partitions into exactly k parts. Each problem corresponds to one of the twelvefold way, that is a $2 \times 2 \times 3$ array of basic combinatorial problems [K05, St97]. We also give two more algorithms. If $n = 1$, then generating all partitions with/without some property is trivial. Hence we assume $n > 1$ in this paper.

The rest of the paper is organized as follows. Section 2 shows a tree structure among partitions of an integer. Section 3 presents our first algorithm. The algorithm generates each partition in $S(n)$ in $O(1)$ time on average. In section 4 we improve the algorithm so that it generates each partition in $O(1)$ time in worst case. By slightly modifying the algorithm we give four more algorithms to generate all partitions of an integer with some property in Section 5 - 8. Finally Section 9 is a conclusion.

Due to space limitation, several details and figures are omitted and can be found in full draft available at [Y04].

2 The Family Tree

In this section we define a tree structure among partitions in $S(n)$.

We define a partition $A \in S(n)$ of a positive integer n as follows. Given a positive integer n, *a partition* of n is an integers sequence $A = a_1a_2 \dots a_m$ for some $m \geq 1$ such that $a_1 \geq a_2 \geq \cdots \geq a_m > 0$ and $n = a_1 + a_2 + \cdots + a_m$. Each integer is called as *a part* of A, and they appear in nonincreasing order. If $m = 1$ then $A = n$ holds. We call it *the root partition* of n.

Then we define *the parent partition* $P(A)$ for each partition A in $S(n)$ except for the root partition as follows. Let $A = a_1a_2 \dots a_m$ be a partition in $S(n)$, and assume that A is not the root partition. We have the following two cases.

Case 1: $a_m = 1$.

We define $P(A) = (a_1 + 1)a_2 \dots a_{m-1}$ by removing a_m and adding one to a_1. Note that the number of parts of $P(A)$ is one less than that of A.

Case 2: $a_m > 1$.

We define $P(A) = (a_1+1)a_2 \dots (a_m - 1)$ by subtracting one from a_m and adding one to a_1. Note that the number of parts of $P(A)$ is equal to that of A.

A is called *a child partition* of $P(A)$. Note that A has the unique parent partition $P(A)$, on the other hand $P(A)$ has at most two child partitions, say Case 1 child and Case 2 child. We have the following lemma.

Lemma 2.1 *If $A \in S(n)$ and A is not the root partition, then $P(A) \in S(n)$.*

By the lemma above, given a partition A in $S(n)$, where A is not the root partition, repeatedly finding the parent partition of the derived partition produces the unique sequence $A, P(A), P(P(A)), \dots$ of partitions in $S(n)$, which eventually ends with the root partition. By merging these sequences we have *the family tree* of $S(n)$, denoted by T_n, such that the vertices of T_n correspond to the partitions in $S(n)$, and each edge corresponds to each relation between some A and $P(A)$. For instance, T_8 is shown in Fig. 1, where each solid line corresponds to the relation with Case 1, and each dashed line corresponds to Case 2.

3 Algorithm

In this section we give an algorithm to construct T_n and generate all partitions in $S(n)$.

If all child partitions of a given partition in $S(n)$ can be generated, then T_n can be constructed in a recursive manner, and all partitions in $S(n)$ can be generated. How can we generate all child partitions of a given partition?

Let $A = a_1a_2 \dots a_m (0 < m \leq n)$ be a partition in $S(n)$. We are going to give a method to compute all child partitions of A.

We now need some definitions. Let $A[m]$ be a partition derived from A by subtracting one from a_1 and adding one to a_m. Intuitively $A[m]$ is the possible Case

2 child of A. Also, let $A[m+1]$ be a partition derived from A by subtracting one from a_1 and appending a new part $a_{m+1} = 1$. Intuitively $A[m+1]$ is the possible Case 1 child of A. Thus, $A[m] = (a_1 - 1)a_2 \dots a_{m-1}(a_m + 1)$ and $A[m+1] = (a_1 - 1)a_2 \dots a_m a_{m+1}$.

If A is the root partition $A_r = n$ then we can observe that A_r has only one child partition $A[m+1] = (n-1)1$. Otherwise we have the following three cases.

Case 1: $a_1 = a_2$.

In this case, $a_1 < a_2$ holds both in $A[m]$ and $A[m+1]$. This means that neither $A[m]$ nor $A[m+1]$ is a partition in $S(n)$. Therefore A has no child partition.

Case 2: $a_1 > a_2$ and $a_{m-1} = a_m$.

In this case $a_{m-1} < a_m$ holds in $A[m]$, so $A[m]$ is not a partition, and A has no Case 2 child. On the other hand $A[m+1]$ is a partition in $S(n)$, and $A[m+1]$ is the Case 1 child of A.

Case 3: $a_1 > a_2$ and $a_{m-1} > a_m$.

We have two subcases.

Case 3(a): $m = 2$ and $a_{m-1} - a_m = 1$.

In this case $a_{m-1} < a_m$ holds in $A[m]$, so $A[m]$ is not a partition, and A has no Case 2 child. On the other hand $A[m+1]$ is a partition in $S(n)$, and $A[m+1]$ is the Case 1 child of A.

Case 3(b): Otherwise.

In this case $A[m]$ is the Case 2 child of A, and $A[m+1]$ is the Case 1 child of A.

For instance, for $A = 431$ in $S(8)$, $A[3] = 332$ and $A[4] = 3311$, and they are the two child partitions of A. Based on the case analysis above, now we give an algorithm to generate all partitions.

```
    Procedure find-all-children($A = a_1 a_2 \cdots a_m$)
    { $A$ is the current partition.}
    begin
01    Output $A$ {Output the difference from the
                  preceding partition.}
02    if $a_1 > a_2$ then
03    begin
04      find-all-children($A[m+1]$)
        {Case 2, 3(a) and 3(b)}
05      if $a_{m-1} > a_m$ and ($m > 2$ or $a_{m-1} - a_m > 1$)
        then
06        find-all-children($A[m]$) {Case 3(b)}
07    end
    end
```

```
    Algorithm find-all-partitions($n$)
    begin
01    Output the root partition $A_r = n$
02    find-all-children($A = (n-1)1$)
    end
```

Theorem 3.1 *The algorithm uses $O(n)$ space and runs in $O(|S(n)|)$ time.*

Thus, the algorithm generates each partition in $O(1)$ time "on average". In the next section we improve the algorithm to generate each partition in $O(1)$ time "in worst case".

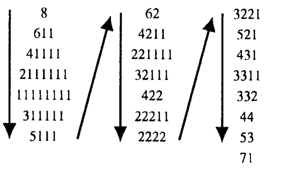

Figure 2: A combinatorial Gray code for $S(8)$.

4 Modification

The algorithm in Section 3 generates all partitions in $S(n)$ in $O(|S(n)|)$ time. Thus the algorithm generates each partition in $O(1)$ time "on average". However, after generating a partition corresponding to the last vertex in a large subtree of T_n, we have to merely return from the deep recursive call without outputting any partition. This may take much time. Therefore, each partition cannot be generated in $O(1)$ time in worst case.

However, a simple modification [NU04] improves the algorithm to generate each partition in $O(1)$ time. The algorithm is as follows.

```
    Procedure find-all-children2($A$, depth)
    { $A$ is the current partition, and depth is the depth
    of the recursive call.}
    begin
01  if depth is even
02  then Output $A$
    {before outputting its child partitions.}
03  Generate child partitions by the method in
    Section 3, and recursively call find-all
    -children2 for each child partition.
04  if depth is odd
05  then Output $A$
    {after outputting its child partitions.}
    end
```

One can observe that the algorithm generates all partitions so that each partition can be obtained from the preceding one by tracing at most three edges of T_n. Note that if A corresponds to a vertex v in T_n with odd depth, then we may need to trace three edges to generate the next partition. Otherwise we need to trace at most two edges to generate the next partition. Note that each partition is similar to the preceding one, since it can be obtained with at most three operations. See Fig. 2. Thus, we can regard the derived sequence of the partitions as a combinatorial Gray code [J80, R93, S97, W89] for partitions.

5 Partitions into at most k parts

In this section we give a method to generate all partitions into at most k parts.

Let $S_{\leq k}(n)$ be the set of all partitions into at most k

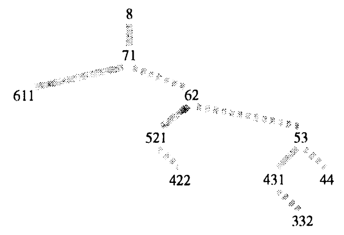

Figure 3: The family tree T_8^3.

parts. Since each part is positive, $n \geq k$ holds. Clearly all such partitions are in T_n but with other partitions. By removing other partitions in T_n, the family tree T_n^k of $S_{\leq k}(n)$ can be defined. Note that, after the removing, T_n remains to be connected (since the number of parts of $P(A)$ is always less than or equal to A), and T_n^k has the same root partition as T_n. For instance T_8^3 is shown in Fig. 3.

Now, given a partition A in $S_{\leq k}(n)$, we can generate all child partitions of A in $S_{\leq k}(n)$ as follows.

```
    Procedure find-all-children3($A = a_1a_2\cdots a_m$, $k$)
    { $A$ is the current partition.}
    begin
01    Output $A$ {Output the difference from the
                 preceding partition.}
02    if $a_1 > a_2$ then
03    begin
04      if $m < k$ then
05        find-all-children3($A[m+1]$)
06        {Case 2, 3(a) and 3(b)}
06      if $a_{m-1} > a_m$ and ($m > 2$ or $a_{m-1} - a_m > 1$)
        then
08        find-all-children3($A[m]$) {Case 3(b)}
09    end
    end
```

The above algorithm can be also modified so as to generate each partition in constant time in worst case, as we have shown in Section 4.

We have the following theorem.

Theorem 5.1 *One can generate all partitions of an integer n into at most k parts in $O(1)$ time for each.*

6 Partitions into exactly k parts

In this section we give an algorithm to generate all partitions into exactly k parts.

Let $S_{=k}(n)$ be the set of all partitions of n into exactly k parts, where n and k are positive integers and $n \geq k$ holds. There is the following simple correspondence between $S_{=k}(n)$ and $S_{\leq k}(n-k)$.

If $k = n$, then $S_{=n}(n)$ contains exactly one partition. In this case generating all partitions in $S_{=n}(n)$ is trivial. Hence we assume $n > k$ in this section.

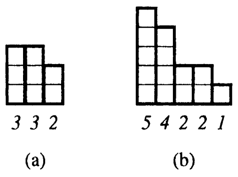

Figure 4: Ferrers diagrams for partitions 332 and 54221.

A partition in $S_{\leq k}(n-k)$ can be computed from a partition in $S_{=k}(n)$ as follows. Let A be a partition in $S_{=k}(n)$, and B be a partition obtained from A by subtracting one from each part of A. Since A has exactly k parts, B has at most k parts. Hence, B is in $S_{\leq k}(n-k)$.

Conversely, a partition in $S_{=k}(n)$ can be computed from a partition in $S_{\leq k}(n-k)$ as follows. Let B be a partition in $S_{\leq k}(n-k)$. Let b be the number of parts of B. Let A be a partition obtained from B by adding one to each part of B and then appending $k-b$ new parts with value 1. Since A has exactly k parts, A is in $S_{=k}(n)$.

Thus, there is a bijection between $S_{=k}(n)$ and $S_{\leq k}(n-k)$, and by slightly modifying the algorithm in Section 5, we can also generate all partitions in $S_{=k}(n)$.

We have the following theorem.

Theorem 6.1 *One can generate all partitions of an integer n into exactly k parts in $O(1)$ time for each.*

7 Restricted partitions

In this section we give two methods to generate all "restricted" partitions.

Given two positive integers n and h ($n \geq h$), *a restricted partition* is a partition of an integer n whose largest part is no greater than h. Let $R(n,h)$ be the set of all restricted partitions.

Our first method is based on a correspondence between $R(n,h)$ and $S_{\leq h}(n)$, where $S_{\leq h}(n)$ is the set of all partitions into at most h parts.

To show the correspondence, a diagrammatic representation of partitions is very useful. In this representation the parts are arranged in order, with the largest part first, and each part is represented by a column of the appropriate number of boxes. For example, Fig. 4 shows the diagrams of two partitions 332 and 54221. Such a diagram is called *a Ferrers diagram.*

Now, a partition B in $S_{\leq h}(n)$ can be computed from a given partition A in $R(n,h)$ as follows.

First the number of boxes on the lowest row in the Ferrers diagram of A is defined as the first part of B. Next the number of boxes on the second lowest row in the Ferrers diagram of A is defined as the second part of B. Similarly we can define every part of B. Thus a partition B can be computed from A. This correspondence is illustrated in Fig. 5. Now we show

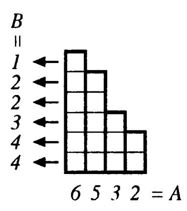

Figure 5: Illustration for converting.

that B is in $S_{\leq h}(n)$. The number of parts of B is at most h, because the largest part of A is no greater than h. Also, the sum of all parts of B is equal to n. Thus B is in $S_{\leq h}(n)$.

Conversely, by the method above, a partition A in $R(n,h)$ can be computed from a given partition B in $S_{\leq h}(n)$.

Thus, there is a bijection between $R(n,h)$ and $S_{\leq h}(n)$. Therefore, with a modification we can regard the algorithm in Section 5 also as the generating algorithm of $R(n,h)$.

Now we give our second method. The method is based on the family tree. We define a new family tree among partitions in $R(n,h)$. Again with traversing the family tree, all partitions in $R(n,h)$ can be generated, as shown in Section 3.

Let $A = a_1a_2\ldots a_m$ be a partition in $R(n,h)$. A satisfies $h \geq a_1 \geq a_2 \geq \cdots \geq a_m > 0$ and $n = a_1 + a_2 + \cdots + a_m$. We define, as the root partition of $R(n,h)$, the partition with the minimum number of parts in $R(n,h)$, as follows. If n is a multiple of h, then the root partition consists of $\frac{n}{h}$ parts with value h. Otherwise, the root partition consists of $\lfloor\frac{n}{h}\rfloor$ parts with value h and one more part with value $n - h\lfloor\frac{n}{h}\rfloor$.

We define the parent partition $P(A)$ for each partition A in $R(n,h)$ except for the root partition as follows. Let $A = a_1a_2\ldots a_ia_{i+1}\ldots a_m$ be a partition in $R(n,h)$. Assume that A is not the root partition, and $h = a_1 = a_2 = \cdots = a_i > a_{i+1}$ holds for some i. Note that, since A is not the root partition, A has at least two parts with value less than h, and $i+1 < m$ holds. Note that $i = 0$ implies A has no part with value h.

We have the following two cases.

Case 1: $a_m = 1$.

We define $P(A) = a_1a_2\ldots a_i(a_{i+1}+1)\ldots a_{m-1}$ by removing a_m and adding one to a_{i+1}.

Case 2: $a_m > 1$.

We define $P(A) = a_1a_2\ldots a_i(a_{i+1}+1)\ldots(a_m - 1)$ by subtracting one from a_m and adding one to a_{i+1}.

A is called a child partition of $P(A)$. Note that, in both cases, $P(A) \neq A$ holds since the added part a_{i+1} is different from the removing/subtracting part a_m. Thus we have the following lemma.

Lemma 7.1 *If $A \in R(n,h)$ and A is not the root partition of $R(n,h)$, then $P(A) \in R(n,h)$.*

By the lemma above, similar to Section 3, we have the (new) family tree of $R(n,h)$, denoted by $T_{n,h}$. For instance, $T_{10,4}$ is shown in Fig. 6. Each vertex of $T_{10,4}$ corresponds to each partition in $R(10,4)$, and each edge of $T_{10,4}$ corresponds to each relation between some partition in $R(10,4)$ and its parent partition.

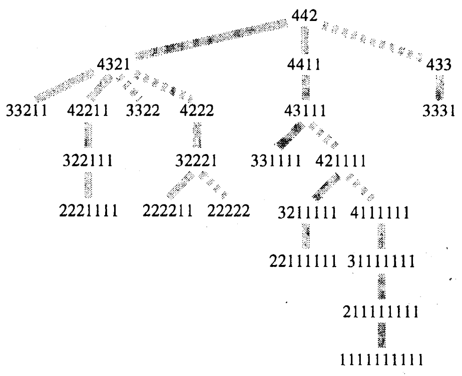

Figure 6: The family tree $T_{10,4}$.

Now we give an algorithm to construct $T_{n,h}$ and generate all partitions in $R(n,h)$. Given a partition $A = a_1a_2\ldots a_ia_{i+1}\ldots a_m$ in $R(n,h)$, all child partitions of A can be computed as follows.

We need some definitions. Let $A[i,m+1]$ be a partition derived from A by subtracting one from a_i and appending a new part $a_{m+1} = 1$. Let $A[i+1,m+1]$ be a partition derived from A by subtracting one from a_{i+1} and appending a new part $a_{m+1} = 1$. Let $A[i,m]$ be a partition derived from A by subtracting one from a_i and adding one to a_m. Let $A[i+1,m]$ be a partition derived from A by subtracting one from a_{i+1} and adding one to a_m. Thus, $A[i,m+1] = a_1a_2\ldots(a_i-1)a_{i+1}\ldots a_ma_{m+1}$, $A[i+1,m+1] = a_1a_2\ldots a_i(a_{i+1}-1)\ldots a_ma_{m+1}$, $A[i,m] = a_1a_2\ldots(a_i-1)a_{i+1}\ldots(a_m+1)$, and $A[i+1,m] = a_1a_2\ldots a_i(a_{i+1}-1)\ldots(a_m+1)$. It is clear that only those four are the candidates of child partitions of A. Note that if $i = 0$ then $A[i,m+1]$ and $A[i,m]$ are not defined.

Now we give a method to generate all child partitions of A.

If A is the root partition $A_r = hh\cdots hr$, where $r = n - h\lfloor\frac{n}{h}\rfloor$, then we can observe that there are the following four cases.

Case 1: $i = 0$.

Since $i = 0$, $A[i,m+1]$ and $A[i,m]$ are not defined. Also, since $A[i+1,m] = A_r$, it is not a child partition of A_r. On the other hand, $A[i+1,m+1]$ is a partition in $R(n,h)$. Thus, A_r has one child partition.

Case 2: $i > 0$ and $r = 0$.

In this case, since a_{i+1} is not defined as a part of A_r, $A[i+1,m+1]$ and $A[i+1,m]$ are not defined. Since $A[i,m] = A_r$, it is not a child partition of A_r. On the other hand, $A[i,m+1]$ is a partition in $R(n,h)$. Thus, A_r has one child partition.

Case 3: $i > 0$ and $r = 1$.

Since $a_{i+1} < a_{i+2}$ in $A[i{+}1, m{+}1]$, it is not a partition in $R(n, h)$. Also, since $A[i+1, m] = A_r$, it is not a partition in $R(n, h)$. We have two subcases.

Case 3(a): $a_i - a_{i+1} = 1$ $(a_i = 2)$.

Since $a_i < a_{i+1}$ in $A[i, m]$, it is not a partition in $R(n, h)$. On the other hand, $A[i, m+1]$ is a partition in $R(n, h)$. Thus, A_r has one child partition.

Case 3(b): $a_i - a_{i+1} > 1$.

$A[i, m{+}1]$ and $A[i, m]$ are partitions in $R(n, h)$. Thus, A_r has two child partitions.

Case 4: $i > 0$ and $r > 1$

We have two subcases.

Case 4(a): $a_i - a_{i+1} = 1$.

Since $a_i < a_{i+1} (= a_m)$ in $A[i, m]$, it is not a partition in $R(n, h)$. Also, since $A[i + 1, m] = A_r$, it is not a partition in $R(n, h)$. On the other hand, $A[i, m{+}1]$ and $A[i + 1, m + 1]$ are partitions in $R(n, h)$. Thus A_r has two child partitions.

Case 4(b): $a_i - a_{i+1} > 1$.

Since $A[i{+}1, m] = A_r$, it is not a partition in $R(n, h)$. On the other hand, $A[i, m + 1]$, $A[i + 1, m + 1]$ and $A[i, m]$ are partitions in $R(n, h)$. Thus A_r has three child partitions.

Note that the root partition A_r in $R(n, h)$ has at most three child partitions. For instance, $A_r = 442$ in $R(10, 4)$ has three child partitions $A[2, 4] = 4321$, $A[3, 4] = 4411$ and $A[2, 3] = 433$.

If A is not the root partition, then we have the following four cases.

Case 1: $i = 0$ and $a_{i+1} = a_{i+2}$.

Since $a_1 < a_2$ holds in both $A[i{+}1, m{+}1]$ and $A[i + 1, m]$, they are not partitions in $R(n, h)$. Also, since $i = 0$, $A[i, m + 1]$ and $A[i, m]$ are not defined. Thus A has no child partition.

Case 2: $i > 0$ and $a_{i+1} = a_{i+2}$.

Since $a_{i+1} < a_{i+2}$ holds in both $A[i + 1, m + 1]$ and $A[i + 1, m]$, they are not partitions in $R(n, h)$. We have two subcases.

Case 2(a): $a_{m-1} > a_m$.

$A[i, m{+}1]$ and $A[i, m]$ are partitions in $R(n, h)$. Thus A has two child partitions $A[i, m + 1]$ and $A[i, m]$.

Case 2(b): $a_{m-1} = a_m$.

Since $a_{m-1} < a_m$ holds in $A[i, m]$, it is not a partition in $R(n, h)$. On the other hand, $A[i, m{+}1]$ is a partition in $R(n, h)$. Thus A has one child partition $A[i, m + 1]$.

Case 3: $i = 0$ and $a_{i+1} > a_{i+2}$.

Omitted. See [Y04].

Case 4: $i > 0$ and $a_{i+1} > a_{i+2}$.

Omitted. See [Y04].

Based on the above cases, the algorithm is as follows.

```
Procedure
find-all-children4($A = a_1a_2 \ldots a_ia_{i+1} \ldots a_m$)
{$A$ is the current partition.}
begin
  Output $A$ {Output the difference from the
             preceding sequence.}
  if $i > 0$ and $a_{i+1} = a_{i+2}$ then   {Case2}
  begin
    find-all-children4($A[i, m + 1]$)
    {Case2(a) and (b)}
    if $a_{m-1} > a_m$ then
      find-all-children4($A[i, m]$)  {Case2(a)}
  end
  else if $i = 0$ and $a_{i+1} > a_{i+2}$ then   {Case3}
  begin
    find-all-children4($A[i + 1, m + 1]$)
    {Case3(a1), (a2) and (b)}
    if $a_{m-1} > a_m$
        and ($m > 2$ or $a_{i+1} - a_{i+2} > 1$) then
      find-all-children4($A[i{+}1, m]$)  {Case3(a2)}
  end
  else if $i > 0$ and $a_{i+1} > a_{i+2}$ then   {Case4}
  begin
    find-all-children4($A[i, m + 1]$)
    {Case4(a1), (a2) and (b)}
    find-all-children4($A[i + 1, m + 1]$)
    {Case4(a1), (a2) and (b)}
    if $a_{m-1} > a_m$ then
    begin
      find-all-children4($A[i, m]$)
      {Case4(a1) and (a2)}
      if $m - i > 2$ or $a_{i+1} - a_{i+2} > 1$ then
        find-all-children4($A[i + 1, m]$)
        {Case4(a2)}
    end
  end
end
```

```
Algorithm find-all-partitions4($n, h$)
begin
  Output the root partition $A_r = hh \ldots hr$
  if $i = 0$ then   {Case1}
    find-all-children4($A[i + 1, m + 1]$)
  else if $r = 0$ then   {Case2}
    find-all-children4($A[i, m + 1]$)
  else if $r = 1$ then   {Case3}
  begin
    find-all-children4($A[i, m + 1]$)
    {Case3(a) and (b)}
    if $a_i - a_{i+1} > 1$ then
      find-all-children4($A[i, m]$)   {Case3(b)}
  end
  else begin   {Case4}
    find-all-children4($A[i, m + 1]$)
    {Case4(a) and (b)}
    find-all-children4($A[i + 1, m + 1]$)
    {Case4(a) and (b)}
    if $a_i - a_{i+1} > 1$ then
      find-all-children4($A[i, m]$)  {Case 4(b)}
  end
end
```

By modifying the above algorithm as we have shown in Section 4, we have the following theorem.

Theorem 7.2 *One can generate all partitions in*

$R(n,h)$ in $O(1)$ time for each.

Let $R_{\leq k}(n,h)$ be the set of all partitions in $R(n,h)$ into at most k parts. Also, similar to Section 5, all partitions in $R_{\leq k}(n,h)$ can be generated. Note that, if $h \cdot k < n$, then there is no partition in $R_{\leq k}(n,h)$.

We have the following theorem.

Theorem 7.3 *One can generate all partitions in $R_{\leq k}(n,h)$ in $O(1)$ time for each.*

8 Doubly restricted partitions

Given three positive integers n, h, l ($n \geq h \geq l$), *a doubly restricted partition* is a partition of n whose largest part is no greater than h and smallest part is no less than l. Let $R(n,h,l)$ be the set of all doubly restricted partitions.

In this section we give a method to generate all partitions in $R(n,h,l)$.

Let $R_{=m}(n,h,l)$ be the set of all partitions in $R(n,h,l)$ into exactly m parts. First we show a method to generate all partitions in $R_{=m}(n,h,l)$. This method is used as a subroutine to generate all partitions in $R(n,h,l)$.

Let $A = a_1 a_2 \ldots a_m$ be a partition in $R_{=m}(n,h,l)$. By subtracting l from every part of A, $A^{'} = (a_1 - l)(a_2 - l)\ldots(a_r - l)$ is derived, where r is the number of parts of A whose value is larger than l. Note that the number of parts of $A^{'}$ may be less than m. Clearly $A^{'}$ is in $R_{\leq m}(n-ml, h-l)$, where $R_{\leq m}(n-ml, h-l)$ is the set of all restricted partitions into at most m parts. Conversely, A can be derived by adding l to every part of $A^{'}$ and appending $(m-r)$ parts with value l to $A^{'}$. Thus, there is a bijection between $R_{=m}(n,h,l)$ and $R_{\leq m}(n-ml, h-l)$.

The algorithm in Theorem 7.3 can be modified to generate each partition in the multiplicity representation. The modified algorithm generates each partition in $R_{\leq m}(n-ml, h-l)$ in constant time. We have the following lemma.

Lemma 8.1 *One can generate all partitions in $R_{=m}(n,h,l)$ in $O(1)$ time for each.*

Next we show that, by repeatedly using the algorithm in Lemma 8.1 for $m = \lfloor \frac{n}{l} \rfloor, \lfloor \frac{n}{l} \rfloor - 1, \ldots, \lceil \frac{n}{h} \rceil$, all partitions in $R(n,h,l)$ can be generated in constant time. Note that, if $(h-l)\lfloor \frac{n}{l} \rfloor \leq n - l\lfloor \frac{n}{l} \rfloor$ then, $R(n,h,l)$ has no partition. The algorithm generates each partition in the multiplicity representation.

The algorithm in Lemma 8.1 first outputs the root partition of $R_{=m}(n,h,l)$, then outputs each partition in $R_{=m}(n,h,l)$ in constant time for each, and finally outputs the last child partition of the root partition. Actually, after that, the algorithm generates the root partition again in constant time, but does not output it again. So all we need to show is that, after the root partition A of $R_{=m}(n,h,l)$ is generated (again), the root partition B of $R_{=m-1}(n,h,l)$ can be generated in constant time.

Now we show that B can be computed from A in constant time. Let s and t be the remainders of $\frac{n-ml}{h-l}$ and $\frac{l}{h-l}$, respectively. Assume

$$A = \begin{cases} h \times m_h + (s+l) \times 1 + l \times m_l, & \text{if } s > 0 \\ h \times m_h + l \times m_l, & \text{if } s = 0, \end{cases}$$

where m_h and m_l are the number of parts with value h and l, respectively. B can be computed from A as follows. First we remove the last part with value l. Next, by using the removed part, we construct as many parts with value h as possible. Formally, B is as follows.

$$B = \begin{cases} h \times (m_h + \lfloor \frac{l}{h-l} \rfloor + 1) + (s^{'} + l) \times 1 \\ \quad +l \times (m_l - \lfloor \frac{l}{h-l} \rfloor - 2), \quad \text{if } \frac{s+t}{h-l} > 1 \\ h \times (m_h + \lfloor \frac{l}{h-l} \rfloor + 1) \\ \quad +l \times (m_l - \lfloor \frac{l}{h-l} \rfloor - 1), \quad \text{if } \frac{s+t}{h-l} = 1 \\ h \times (m_h + \lfloor \frac{l}{h-l} \rfloor) + (s^{'} + l) \times 1 \\ \quad +l \times (m_l - \lfloor \frac{l}{h-l} \rfloor - 2), \\ \qquad \text{if } 0 < \frac{s+t}{h-l} < 1 \text{ and } s = 0 \\ h \times (m_h + \lfloor \frac{l}{h-l} \rfloor) + (s^{'} + l) \times 1 \\ \quad +l \times (m_l - \lfloor \frac{l}{h-l} \rfloor - 1), \\ \qquad \text{if } 0 < \frac{s+t}{h-l} < 1 \text{ and } s > 0 \\ h \times (m_h + \lfloor \frac{l}{h-l} \rfloor) \\ \quad +l \times (m_l - \lfloor \frac{l}{h-l} \rfloor - 1), \quad \text{if } \frac{s+t}{h-l} = 0, \end{cases}$$

where $s^{'}$ is the remainder of $\frac{s+t}{h-l}$.

In the above method, if partitions are represented in standard representation, B can not be generated from A in constant time. Because at least $\lfloor \frac{l}{h-l} \rfloor$ parts of A may be changed to generate B. Since we use the multiplicity representation in this section, we can generate B from A in constant time.

We have the following theorem.

Theorem 8.2 *One can generate all partitions in $R(n,h,l)$ in $O(1)$ time for each (in the multiplicity representation).*

9 Conclusion

In this paper we have given five simple algorithms to generate all partitions with some property. We first define a family tree such that each vertex corresponds to each partition with the given property, then output each partition in constant time by traversing the family tree.

References

[A93] G. S. Akl and I. Stojmenović, *Parallel algorithms for generating integer partitions and compositions*, Journal of Combinatorial Mathematics and Combinatorial Computing, 13, (1993), pp.107–120.

[A76] G. Andrews, *The Theory of Partitions*, Encyclopedia of Mathematics and its Applications, Vol. 2. Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, (1976).

[B80] T. Beyer and S. M. Hedetniemi, *Constant Time Generation of Rooted Trees*, SIAM J. Comput., 9, (1980), pp.706–712.

[F80] T. I. Fenner and G. Loizou, *A Binary Tree Representation and Related Algorithms for Generating Integer Partitions*, The Computer Journal, 23, (1980), pp.332–337.

[F81] T. I. Fenner and G. Loizou, *An Analysis of Two Related Loop-free Algorithms for Generating Integer Partitions*, Acta Informatica, 16, (1981), pp.237–252.

[G93] L. A. Goldberg, *Efficient Algorithms for Listing Combinatorial Structures*, Cambridge University Press, New York, (1993).

[J80] J. T. Joichi, D. E. White, and S. G. Williamson, *Combinatorial Gray Codes*, SIAM J. Comput., 9, (1980), pp.130–141.

[KN05] S. Kawano and S. Nakano, *Constant Time Generation of Set Partitions*, IEICE TRANS. FUNDAMENTALS, Vol. E88-A, no. 4, (2005), pp. 930–934.

[KS98] D. L. Kreher and D. R. Stinson, *Combinatorial Algorithms*, CRC Press, Boca Raton, (1998).

[K05] D. E. Knuth, *The Art of Computer Programmings*, vol. 4, Fascicle 3, Addison-Wesley, (2005).

[LN01] Z. Li and S. Nakano, *Efficient Generation of Plane Triangulations without Repetitions*, Proc. ICALP2001, LNCS 2076, (2001), pp.433–443.

[LR99] G. Li and F. Ruskey, *The Advantage of Forward Thinking in Generating Rooted and Free Trees*, Proc. 10th Annual ACM-SIAM Symp. on Discrete Algorithms, (1999), pp.939–940.

[M98] B. D. McKay, *Isomorph-free Exhaustive Generation*, J. of Algorithms, 26, (1998), pp.306–324.

[M65] J. K. S. McKay, *Partition Generator*, Algorithm 263, Communication of the ACM, 8, (1965), pp.493.

[M70] J. K. S. McKay, *Partitions in Natural Order*, Algorithm 371, Communication of the ACM, 13, (1970), pp.52.

[N01] S. Nakano, *Enumerating Floorplans with n Rooms*, Proc. ISAAC 2001, LNCS 2223, (2001), pp.104–115.

[N02] S. Nakano, *Efficient Generation of Plane Trees*, Information Processing Letters, 84, (2002), pp.167–172.

[NU04] S. Nakano and T. Uno, *Constant Time Generation of Trees with Specified Diameter*, Proc. WG2004, LNCS 3353, (2004), pp.33–45.

[N71] T. V. Narayana, R. M. Mathsen, and J. Sarangi, *An Algorithm for Generating Partitions and Its Applications*, Jounal of Combinatorial Theory (A), vol.11, (1971), pp.54–61.

[N75] A. Nijenhuis and H. S. Wilf, *Combinatorial Algorithms*, Academic Press, NY, (1975).

[P79] E. S. Page and L. B. Wilson, *An introduction to Computational Combinatorics*, Cambridge University Press, (1979).

[R95] D. Rasmussen, C. D. Savage, and D. B. West, *Gray Code Enumeration of Families of Integer Partition*, Journal of Combinatorial Theory (A), 70(2), (1995), pp.201–229.

[R77] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice Hall, Englewood Cliffs, New Jersey, (1977).

[R78] R. C. Read, *How to Avoid Isomorphism Search When Cataloguing Combinatorial Configurations*, Annals of Discrete Mathematics, 2, (1978), pp.107–120.

[R93] F. Ruskey, *Simple combinatorial Gray codes constructed by reversing sublists*, Proc. ISAAC93, LNCS 762, (1993), pp.201–208.

[R00] K. H. Rosen (Eds.), *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, Boca Raton, (2000).

[S89] C. Savage, *Gray Code Sequences of Partitions*, Journal of Algorithms, 10, (1989), pp.577–595.

[S97] C. Savage, *A Survey of Combinatorial Gray Codes*, SIAM Review, 39, (1997) pp. 605-629.

[St97] R. P. Stanley, *Enumerative Combinatorics*, Vol. 1, Cambridge University Press, (1997).

[W89] H. S. Wilf, *Combinatorial Algorithms : An Update*, SIAM, (1989).

[W86] R. A. Wright, B. Richmond, A. Odlyzko, and B. D. McKay, *Constant Time Generation of Free Trees*, SIAM J. Comput., 15, (1986), pp.540–548.

[Y04] K. Yamanaka, S. Kawano, Y. Kikuchi, and S. Nakano, *Constant time Generation of Integer Partitions*, `http://www.msc.cs.gunma-u.ac.jp/~yamanaka/Tech/intPartition.pdf`, (2005).

[Z96] M. Zito, I. Pu, M. Amos, and A. Gibbons, *RNC Algorithms for the Uniform Generation of Combinatorial Structures*, Proc. of the 7th ACM-SIAM Annual Symposium on Discrete Algorithm, (1996), pp.429–437.

[Z98] A. Zoghbi, I. Stojmenović, *Fast Algorithms for Generating Integer Partitions*, International Journal of Computer Mathematics, Vol. 70, (1998), pp.319–332.