Intern. J. Computer Math., Vol. 70, pp. 319–332
Reprints available directly from the publisher
Photocopying permitted by license only

FAST ALGORITHMS FOR GENERATING INTEGER PARTITIONS

ANTOINE ZOGHBI[a] and IVAN STOJMENOVIĆ[b,*]

[a] *Bell Northern Research, P.O. Box 3511, Station C, Mail Stop 091, Ottawa, Ontario K1Y 4H7, Canada;*
[b] *Computer Science Department, SITE, University of Ottawa, Ottawa, Ontario K1N 6N5, Canada*

(Received 28 January 1998)

We present two new algorithms for generating integer partitions in the standard representation. They generate partitions in lexicographic and anti-lexicographic order, respectively. We prove that both algorithm generate partitions with constant average delay, exclusive of the output. These are the first known algorithms to produce partitions in the standard representation and with constant average delay. The performance of all known integer partition algorithms is measured and compared, separately for the standard and multiplicity representation. An empirical test shows that both new algorithms are several times faster than any of previously known algorithms for generating unrestricted integer partitions in the standard representation. Moreover, they are faster than any known algorithm for generating integer partition in the multiplicity representation (exclusive of the output).

Keywords: Combinatorial objects; lexicographic order; integer partitions

C. R. Categories: G.2.1. Combinatorial algorithms, integer partitions

1. INTRODUCTION

Given an integer n, it is possible to represent it as the sum of one or more positive integers a_i, *i.e.*, $n = x_1 + x_2 + \cdots + x_m$. This representation is called a partition if the order of the x_i is of no consequence. Thus two partitions of an integer n are distinct if they differ with respect to the x_i they contain.

*Corresponding author.

For example, there are seven distinct partitions of the integer 5:

$$5,\ 4+1,\ 3+2,\ 3+1+1,\ 2+2+1,\ 2+1+1+1,\ 1+1+1+1+1.$$

The partitions of an integer have been the subject of extensive study for over 300 years, since Leibniz asked Bernoulli if he had investigated $P(n)$, the number of partitions of an integer n. Details of the history and the state of the art as of 1920 can be found in Chapter 3 of [7]. Additional details and later results can be found in most combinatorics texts; in particular, see [14, 19, 29]. This interest is partly motivated by the important role played by partitions and compositions in many problems of combinatorics and algebra. In general, a list of all combinatorial objects of a given type might be used to search for a counter-example to some conjecture, or to test and analyze an algorithm for its correctness or computational complexity. For computational purposes one is often interested in generating all the partitions of an integer, or sometimes just those satisfying various restrictive conditions. Several such algorithms, dealing with both the unrestricted [3, 18, 20, 23, 25, 27, 30, 31] and restricted [3, 18, 24, 31, 41] cases, have appeared in the literature.

This paper is organized as follows. Section 2 gives definitions and relations between various kinds and representations of integer partitions. Section 3 surveys known algorithms for generating integer partitions. We describe two new algorithms for generating integer partitions in standard representation in Section 4. Their average delay property (exclusive of the output) is proved in Section 5. The major contribution of the paper is to present the first known algorithms to produce partitions in the standard representation and with constant average delay. Section 6 contains an empirical analysis of all known algorithms for generating integer partitions. The results show that both new algorithms are faster than any of the previously known algorithms, no matter what representation of the partitions is used (if the time to output partitions is not counted in). Some open problems are given in Conclusion section.

2. DEFINITIONS

Lexicographic order of combinatorial objects is defined as follows. If $A = (a_1, a_2, \ldots, a_{s'})$ and $B = (b_1, b_2, \ldots, b_{s''})$ are representations of objects, then A precedes B lexicographically if and only if, for some $j \geq 1$, $a_i = b_i$ when $i < j$, and a_j precedes b_j. For example, partitions of 5 in lexicographic order are: 11111, 2111, 221, 311, 32, 41, 5 (note that '+' sign is omitted).

Lexicographic order is desirable as it is the natural (dictionary) order, and can be easily characterized and traced manually. The anti-lexicographic order is the reverse of lexicographic one. The next interesting order is Gray code or minimal change order [35], and there exist other orders, mostly directly related to objects under consideration.

In standard representation, a partition of n is given by a sequence $x_1 \dots x_m$, where $x_1 \geq x_2 \geq \cdots \geq x_m$, and $x_1 + x_2 + \cdots + x_m = n$. In the sequel x will denote an arbitrary partition and m will denote the number of parts of x (m is not fixed). It is sometimes more convenient to use a multiplicity representation for partitions in terms of a list of the distinct parts of the partition and their respective multiplicities. Let $y_1 > \cdots > y_d$ be all distinct parts in a partitions, and $c_1, \dots, c_d$ their respective (positive) multiplicities. Clearly $c_1 y_1 + \cdots + c_d y_d = n$. The choice between the standard and multiplicity representation depends on the application. For example, the representation of B-trees in [12] requires the standard representation of integer partitions. The standard representation is used to generate an integer partition at random [25, 26].

We refer to the case of partitions without any limitations as being unrestricted partitions. Let restricted partitions be those partitions for which $x_1 \leq U$ is satisfied, *i.e.*, partitions whose largest part is no greater than U. Let $RP(n, U)$ be the number of restricted partitions of n whose largest part is no larger than U. Restricted partitions can be generated using an algorithm to generate unrestricted ones in lexicographic order and stopping the algorithm when the first part becomes greater than U (or starting with first partition $y_1 = k$, $c_1 = \lfloor n/y_1 \rfloor$, $y_2 = n - c_1 y_1$, $c_2 = 1$ if $y_2 > 0$, $c_2 = 0$ otherwise, in case of anti-lexicographic order).

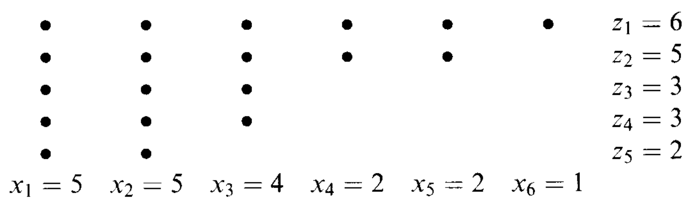
The number of unrestricted partitions $P(n)$ of n can be determined using the following recurrence relation: $RP(n, U) = RP(n-U, U) + RP(n, U-1)$ $(n \geq U \geq 1)$, and boundary condition $RP(n, 1) = RP(1, 1) = RP(n, 0) = RP(0, 1) = 1$ and $P(n) = RP(n, n)$.

Doubly restricted partitions contain parts of size between L and U, *i.e.*, $L \leq x_i \leq U$ for $i = 1, 2, \dots, m$. Multiply restricted partitions of n is a common name for various kinds of special partitions. Examples are partitions with prescribed part sizes (they have parts which are selected from an array v_i, $i = 1, 2, \dots, r$). Tournament scores are studied in [24] while graph degree sequences are studied in [16].

The case of partitions whose largest part is exactly U is given in [25] as a special case of their general method for listing, ranking and unranking combinatorial objects. Note that the case of partitions of n whose largest part is no greater than U is, by adding one more part of size

U, equivalent to the case of partitions of $n + U$ with largest part exactly U.

Using the Ferrers graph (*cf.* [27]) a one-to-one correspondence between partitions of n into m parts and partitions of n whose largest part is m is established. Let $z_1 \ldots z_m$ be a partition into m nonincreasing parts, $z_1 \geq \cdots \geq z_m$ and $x_1 \ldots x_k$, $x_1 \geq \cdots \geq x_k$, be a partition of n into any number of nonincreasing parts (*i.e.*, k varies) with largest part $x_1 = m$. The following Ferrers graph illustrates the relationship between the two kinds of partitions.



The following relation follows from the Ferrers graph: $z_i = \max \{j \mid x_j \geq i\}$. Consider now the corresponding multiplicity representation of partitions with largest part m: $c_1, \ldots, c_d$ are the multiplicities of $y_1, \ldots, y_d$, where $m = y_1 > \cdots > y_d$, and $c_1 y_1 + \cdots + c_d y_d = n$. For the partitions into m parts, let $e_1, \ldots, e_d$ be the multiplicities of $w_1, \ldots, w_d$ (clearly the two sequences have the same number d of different parts), where $w_1 > \cdots > w_d$, $e_1 w_1 + \cdots + e_d w_d = n$, and $e_1 + \cdots + e_d = m$. Then it easily follows that $w_{m-i+1} = c_1 + c_2 + \cdots + c_i$, and $e_{m-i+1} = y_i - y_{i+1}$ where $y_{d+1} = 0$. The sums for w_i can be easily maintained during the execution of a program for generating partitions of n with largest part m in the multiplicity representation.

The delay between two partitions is the time required to generate the new partition from the existing one. Delay is constant if the time is constant, assuming that the time to ouput partitions is not counted. Obviously, a procedure for generating next partition from current one has constant delay if it is loop-free and recursion-free. The average delay is the ratio of the total time to generate all partitions and the total number of partitions. An algorithm has constant average delay property if the ratio is less than a constant for any n, again exclusive of the output time.

3. ALGORITHMS FOR GENERATING INTEGER PARTITIONS

In this section we briefly describe all known algorithms for generating integer partitions. Algorithms are divided according to the kind of partitions generated (unrestricted partitions, restricted partitions, doubly restricted partitions, and multiply restricted partitions), representation (standard representation, multiplicity representation, and combined representation), and the order of generating partitions (lexicographic order, anti-lexicographic order, Gray code order, and part order, in which algorithm actually ge- nerates all partitions containing exactly m parts, where m varies from 1 to n).

Clearly an algorithm for generating doubly restricted partitions can be used to generate restricted or unrestricted partitions. Also, an algorithm which generates restricted partitions can be used to produce unrestricted ones.

Each algorithm is given name by the first letters of its authors. These abbreviations are used later in comparison tables.

In anti-lexicographic order, a partition is derived from the previous one by subtracting 1 from the rightmost part greater than 1, and distributing the remainder as quickly as possible. For example, the partitions following $9+7+6+1+1+1+1+1+1$ is $9+7+5+5+2$. First such algorithm is reported by Stockmal [37] (algorithm S), and it generates restricted partitions. In algorithm S, each partition is represented by the integers $c[1]$ through $c[U]$, where $c[j]$ is the number of parts of the partition equal to the integer j. This combined representation contains zeros and does not have constant delay property.

In standard representation and anti-lexicographic order, the next partition is determined from current one $x_1 x_2 \ldots x_m$ in the following way. Let h be the number of parts of x greater than 1, *i.e.*, $x_i > 1$ for $1 \leq i \leq h$, and $x_i = 1$ for $h < i \leq m$. If $x_m > 1$ (or $h = m$) then the next partition is $x_1, x_2, \ldots, x_{m-1}, x_m - 1, 1$. Otherwise (*i.e.*, $h < m$), the next partition is obtained by replacing $x_h, x_{h+1} = 1, \ldots, x_m = 1$ with $(x_h - 1)$, $(x_h - 1), \ldots, (x_h - 1)$, d, containing c elements, where $0 < d \leq x_h - 1$ and $(x_h - 1)\ (c-1) + d = x_h + m - h$. Based on the general idea, several algorithms were developed: Algorithms A [3], M1 [23], PW1 [27], PW2 [27]. The delay between the generation of two consecutive partitions in the algorithm is $\Theta(n)$ in the worst case (even exclusive of the output).

The same strategy can be used to generate partitions in the multiplicity representation and anti-lexicographic order. The computation of the next partition from current one affects at most two smallest different parts, and creates at most two new different parts. It is possible to perform the update in constant delay per partition (exclusive of the output). Algorithms based on the description are the following: AS [2], FL3 [11], and NW [25].

All known algorithms for generating partitions in lexicographic order use the multiplicity representation. If $c_d > 1$ then one of parts y_d is increased by

1 and $y_d(c_d - 1) - 1$ parts of size 1 are added. Otherwise one of parts y_{d-1} is increased by 1 and $y_{d-1}(c_{d-1} - 1) + y_d - 1$ parts of size 1 are added. In both cases the part which is increased by 1 may happen to be same as the previous part(s), in which case the multiplicities are corrected. For example, the next partition for $5 \times 3 + 4 \times 3 + 2 \times 3$ is $5 \times 3 + 4 \times 3 + 3 \times 1 + 1 \times 5$ while the next partition for $5 \times 3 + 4 \times 3 + 2 \times 1$ is $5 \times 4 + 1 \times 9$. This method is due to Ehrlich (*cf.* [30]) and is referred to as Algorithm E [30]. Algorithm FL1 [9] also belongs to this group. Since the changes are done only on last few parts, the method has constant delay property.

Algorithms M2 [20] and W [41] make use of a procedure that maps an integer between 0 to $P(n) - 1$ into an integer partition. Using the map, which is a bijection (or one to one), it is possible to generate all partitions in various orders. For example, if the mapping is applied from 0 to $P(n) - 1$ one gets partitions in lexicographic order while the application from $P(n) - 1$ to 0 gives anti-lexicographic order. The method is not effective since the mapping uses very large counters $O(P(n))$.

There exist several algorithms that generate partitions of n into exactly m parts. Algorithm GLW [12] generate restricted partitions in the multiplicity representation, in lexicographic order. Algorithm RJ1 [31] generates doubly restricted partitions while Algorithm RJ2 [31] generates multiply restricted partitions (Algorithm RJ2 allows also to limit the number of occurrences of each part), both in anti-lexicographic order and in standard representation.

There exists another solution for the case of partitions of n into exactly m parts in the standard representation. In [18, 30] algorithms are presented for generating unrestricted partitions in lexicographic order for each fixed m, but considering the parts of the partition in non-decreasing rather than non-increasing order. In fact, this algorithm was discovered by K. F. Hindenburg in 1778 (*cf.* [30]), and we refer to it as Algorithm H. To obtain the next partition from the current one, the elements are scanned from right to left, stopping at the rightmost x_i such that $x_m - x_i \geq 2$. Replace x_j by $x_i + 1$ for $j = i, i + 1, \ldots, m - 1$ and then replace x_m by the remainder, to get the sum n. For example, in the partition 11334, $i = 2$ and the next partition is 12225. If the multiplicity representation is used, the algorithm is loop free and works on the last indices only, thus having constant delay property. Note that the parts in Algorithm H can easily be reversely indexed to correspond to our conventional notation; however the order will be neither lexicographic nor anti-lexicographic. This algorithm is coded in the multiplicity representation in [44] (Algorithm Z).

When m varies from 1 to n, all mentioned algorithms generate all partitions of given kind.

We mention two more algorithms. Algorithm FL2 [9] generates unrestricted partitions in the multiplicity representation in so called *M*-order (defined in [9]) while Algorithm Sa [35] generates all partitions in a minimal change (or Gray code) order.

4. NEW ALGORITHMS FOR GENERATING PARTITIONS IN STANDARD REPRESENTATION

In this section we describe two new algorithms for generating integer partitions in standard representation and prove that they have constant average delay property. The first algorithm, named ZS1, generates partitions in anti-lexicographic order while the second, named ZS2, uses lexicographic order.

Recall that h is the index of the last part of partition which is greater than 1 while m is the number of parts. The major idea in Algorithm ZS1 is coming from the observation on the distribution of x_h. An empirical and theoretical study shows that $x_h = 2$ has growing frequency; it appears in 66% of cases for $n = 30$ and in 78% of partitions for $n = 90$ and appears to be increasing with n. Each partition of n containing a part of size 2 becomes, after deleting the part, a partition of $n-2$ (and *vice versa*). Therefore the number of partitions of n containing at least one part of size 2 is $P(n-2)$. By using asymptotic formulae [13] for $P(n)$, it is possible to proof that the ratio $P(n-2)/P(n)$ approaches 1 with increasing n. Thus almost all partitions contain at least one part of size 2. This special case is treated separately, and we will prove that it suffices to argue the constant average delay of Algorithm ZS1. Moreover, since more than 15 instructions in known algorithms which were used for all cases are replaced by 4 instructions in cases of at least one part of size 2 (which happens almost always), the speed up of about four times is expected even before experimental measurements. The case $x_h > 2$ is coded in a similar manner as earlier algorithm, except that assignments of parts which are supposed to receive value 1 is avoided by an initialization step that assigns 1 to each part and observation that inactive parts (these with index $> m$) are always left at value 1. The new algorithm is obtained when the above observation is applied to known algorithms, and can be coded as follows.

Algorithm ZS1

```
for $i \leftarrow 1$ to $n$ do $x_i \leftarrow 1$;
$x_1 \leftarrow n$; $m \leftarrow 1$; $h \leftarrow 1$; output $x_1$;
```

```
while $x_1 \neq 1$ do {
        if $x_h = 2$ then $\{m \leftarrow m + 1;\ x_h \leftarrow 1;\ h \leftarrow h - 1\}$
                else $\{r \leftarrow x_h - 1;\ t \leftarrow m - h + 1;\ x_h \leftarrow r$
                        while $t \geq r$ do $\{h \leftarrow h + 1;\ x_h \leftarrow r;\ t \leftarrow t - r\}$
                        if $t = 0$ then $m \leftarrow h$
                                else $\{m \leftarrow h + 1;$
                            if $t > 1$ then $\{h \leftarrow h + 1;\ x_h \leftarrow t\}\}$
        output $x_1, x_2, \ldots, x_m\}$
```

We now describe the method for generating partitions in lexicographic order and standard representation of partitions. Each partition of n containing two parts of size 1 (*i.e.*, $m - h > 1$) becomes, after deleting these parts, a partition of $n - 2$ (and *vice versa*). Therefore the number of integer partitions containing at least two parts of size 1 is $P(n-2)$, as in the case of previous algorithm. The coding in this case is made simpler, in fact with constant delay, by replacing first two parts of size 1 by one part of size 2. The position h of last part > 1 is always maintained. Otherwise, to find the next partition in lexicographic order, an algorithm will do a backward search to find the first part that can be increased. The last part x_m cannot be increased. The next to last part x_{m-1} can be increased only if $x_{m-2} > x_{m-1}$. The element which will be increased is x_j where $x_{j-1} > x_j$ and $x_j = x_{j+1} = \cdots = x_{m-1}$. The j-th part becomes $x_j + 1$, h receives value j, and appropriate number of parts equal to 1 is added to complete the sum to n. For example, in the partition $5 + 5 + 5 + 4 + 4 + 4 + 1$ the leftmost 4 is increased, and the next partition is $5 + 5 + 5 + 5 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1$. The following is a code of appropriate Algorithm ZS2.

Algorithm ZS2

```
for $i \leftarrow 1$ to $n$ do $x_i \leftarrow 1$; output $x_i,\ i = 1, 2, \ldots, n$;
$x_0 \leftarrow -1;\ x_1 \leftarrow 2;\ h \leftarrow 1;\ m \leftarrow n-1$; output $x_i,\ i = 1, 2, \ldots, m$;
while $x_1 \neq n$ do {
        if $m - h > 1$ then $\{h \leftarrow h + 1;\ x_h \leftarrow 2;\ m \leftarrow m - 1\}$
                else $\{j \leftarrow m - 2;$
                        while $x_j = x_{m-1}$ do $\{x_j \leftarrow 1;\ j \leftarrow j - 1\}$;
                        $h \leftarrow j + 1;\ x_h \leftarrow x_{m-1} + 1$;
```

```
                    $r \leftarrow x_m + x_{m-1}(m-h-1)$; $x_m \leftarrow 1$;
                    **if** $m-h > 1$ **then** $x_{m-1} \leftarrow 1$;
                    $m \leftarrow h+r-1$;
    output $x_1, x_2, \ldots, x_m$}
```

5. CONSTANT AVERAGE DELAY PROPERTY OF NEW ALGORITHMS

The output size of each of $P(n)$ partitions is $O(n)$. This means that the total output size is $O(nP(n))$. However, in some applications the objects which are generated do not need to be printed out, for they merely serve as the source of information for other procedures that work on combinatorial objects and check some criteria which may be verified without always looking at the whole new object. It makes sense then to consider generating combinatorial object without outputing them. Optimal algorithms in this sense work in $O(P(n))$ time, *i.e.*, in constant time per object. Algorithms that generate the next object from current one with constant average delay (exclusive of the output time) exist for various kinds of combinatorial objects [6, 4, 33, 15, 39, 40, 43]. To the best of our knowledge, none of existing algorithms for generating integer partitions in standard representation is known to have constant average delay property. In this section we prove that Algorithms ZS1 and ZS2 have constant delay property. We need the following lemma in our proof.

LEMMA 1 *$RP(n, U) \geq n^2/12$ for $U \geq 3$.*

Proof Since $RP(n, U) \geq RP(n, 3)$ for $U > 3$, it is sufficient to prove $RP(n, 3) \geq n^2/12$. In the multiplicity representation, the partitions in $RP(n, 3)$ are of the following kind: $n = 3c_1 + 2c_2 + 1c_3$ (*i.e.*, $y_1 = 3$, $y_2 = 2, y_3 = 1$). The number of such partitions is equal to the number of solutions of the above equation. Clearly, $0 \leq c_1 \leq \lfloor n/3 \rfloor$. For each c_1 in the interval we solve the equation $2c_2 + c_3 = n - 3c_1$. This equation has unique solution c_3 for each c_2, $0 \leq c_2 \leq \lfloor (n-3c_1)/2 \rfloor$. Therefore, for fixed c_1, the number of solutions is $\lfloor (n-3c_1)/2 \rfloor + 1 \geq (n-3c_1)/2$. Taking all values of c_1 into account, the number of solutions is $\geq n/2 + (n-3)/2 + (n-6)/2 + \cdots + (n - 3\lfloor n/3 \rfloor)/2 = (\lfloor n/3 \rfloor + 1)n/2 - \lfloor n/3 \rfloor(\lfloor n/3 \rfloor + 1)3/4 = (\lfloor n/3 \rfloor + 1)(n/2 - \lfloor n/3 \rfloor 3/4) \geq (\lfloor n/3 \rfloor + 1)(n/2 - n/4) \geq n^2/12$. ■

THEOREM 1 *Algorithms ZS1 and ZS2 generate unrestricted integer partitions in standard representation with constant average dealy, exclusive of the output.*

Proof Consider part $x_i \geq 3$ in the current partition. It received its value after a backtracking search (starting from last part) was performed to find an index $j \leq i$, called the turning point, that should change its value by 1 (increase/decrease for lexicographic/anti-lexicographic order) and to update values x_i for $j \leq i$. The time to perform both backtracking searches is $O(r_j)$ where $r_j = n - x_1 - x_2 - \cdots - x_j$ is the remainder to distribute after first j parts are fixed. We decide to charge the cost of the backtrack search evenly to all "swept" parts, such that each of them receives constant $O(1)$ time. Part x_i will be changed only after a similar backtracking step "swept" over i-th part or recognized i-th part as the turning point (note that i-th part is the turning point in at least one of the two backtracking step). There are $RP(r_i, x_i)$ such partitions which all keep x_j intact. For $x_i \geq 3$ the number of such partitions, according to Lemma 1, is $\geq r_i^2/12$. Therefore the average number of operations that are performed by such part i during the "run" of $RP(r_i, x_i)$, including the change of its value, is $O(1)/RP(r_i, x_i) \leq O(1)/r_i^2 = O(1/r_i^2) < q_i/r_i^2$ where q_i is a constant. Thus the average number of operations for all parts of size ≥ 3 is $\leq q_1/r_1^2 + q_2/r_2^2 + \cdots + q_s/r_s^2 \leq q(1/r_1^2 + \cdots + 1/r_s^2) < q(1/n^2 + 1/(n-1)^2 + \cdots + 1/1^2) < 2q$ (the last inequality can be obtained easily by applying integral operation on the last sum), which is a constant. The case which was not counted in is when $x_i \leq 2$. However, in this case both Algorithms ZS1 and ZS2 perform constant number of steps altogether on all such parts. Therefore the algorithm has overall constant time average delay. ■

6. EMPIRICAL ANALYSIS

In this section we present the results of the performance evaluation of known integer partition generation methods. To the best of our knowledge, this is the first comparison of algorithms for generating integer partitions. Similar analysis exist for permutations [36], combinations [1] and set partitions [8]. The comparison includes our newly proposed algorithms. All algorithms are divided into two groups according to representation-multiplicity or standard. Major comparison is done for unrestricted partitions (algorithms that generate restricted or doubly restricted partitions are also compared here for $L = 1$ and $U = n$; they are compared separately for other bounds of L and U in [44] and obtained results preserve the relative order as shown here).

All algorithms were coded in C language (coding in PASCAL was also done but the comparison gave similar results as in C language). Algorithms that were originally written in old-fashioned way (for example, using go to instruction), like Algorithms M1 and M2, are re-coded in C language fashion and compared in modified form (the modified form was faster by about 10% than the original one in all such cases). The algorithms were implemented on PC-286, Sun, and NeXT workstations in the laboratory of Computer Science Department, University of Ottawa. The actual CPU times and the average number of (arithmetic, logical and assignment) instructions per partition of the algorithms are summarized in tables below. CPU times are measured (averaged over three runs) when algorithms are run without printing out partitions. A separate measurement of the number of instructions rather than running time is also performed, but the results are essentially the same as these obtained by comparing CPU times. More details of all measurements are reported in [44].

The results show clearly that both Algorithms ZS1 and ZS2 are superior to all other known algorithms (M1, RJ1, PW1, H, RJ2, PW2, M2, W) that generate partitions in the standard representation. While their speed was comparable to each other, each of them was at least four times faster on any of three machines when partitions of 75 were generated. Moreover, both algorithms are even faster than any algorithm for generating integer partitions in the multiplicity representation.

Among algorithms that generate partitions in the multiplicity representation (as defined), Algorithm FL1 was fastest on all three machines, and for $n = 75$ was between above 10% and 100% faster than other Algorithms E, FL2, AS, FL3, NW while GLW proved inefficient compared with other algorithms. Algorithm S was faster than Algorithm FL1 and is included in the group since its combined representation is closer to multiplicity than to standard representation; it becomes inefficient if its output is transfered to the multiplicity representation.

Several algorithms were not implemented because of either their apparent inefficiency (for example, Algorithms [35, 38], or certain multiple restrictions which are the only ones of their kind (Algorithms from [16] and [24]), or generating more general objects than partitions [32].

The CPU time in tables below are in seconds. The names of algorithms are defined earlier, and the orders of generations are abbreviated as follows: A (anti-lexicographic), L (lexicographic), P (part order), U (unranking, *i.e.*, mapping from integers to partitions) and N (none of them). The tables refer

TABLE I Generating unrestricted partitions of 75 in the standard representation

Alg.	*Order*	*PC*-286	*SUN*	*NeXT*
ZS1	A	110.0	9.1	19.4
ZS2	L	124.0	10.3	22.5
M1	A	517.0	45.6	116.2
PW1	A	800.0	61.2	123.2
RJI	P	652.0	67.9	130.3
H	N	892.0	75.1	145.4
RJ2	P	997.0	84.9	189.9
PW2	A	1164.0	254.3	223.1
A	A	3138.0	280.9	565.8
M2	U	8086.0	405.2	1371.5
W	U	8508.0	469.1	1621.4

TABLE II Generating unrestricted partitions of 75 in the multiplicity representation

Alg.	*Order*	*PC*-286	*SUN*	*NeXT*
S	L	188.0	13.3	24.9
FL1	L	176.0	14.4	42.8
E	L	191.0	20.8	48.5
FL2	N	245.0	17.0	50.3
FL3	A	214.0	28.3	60.5
AS	A	212.0	30.9	62.6
NW	A	326.0	37.3	80.0
Z	N	319.0	39.0	82.0
GLW	P	500.0	61.6	126.8

to partition of 75. In [44] running time are given for partitions of some other numbers (15, 30, 45 60, and 90) but the results do not differ significantly from the case of number 75.

7. CONCLUSION

We have shown in this paper that two new algorithms for generating integer partition in standard representation, described here, have constant average delay property. To the best of our knowledge, there are first such algorithms for which the property is proved. There exist well-known algorithms for generating partitions in the multiplicity representation. Although the change from multiplicity to the standard representation is routine, it is not a constant time operation and one cannot obtain constant average delay algorithm for generating integer partitions in standard representation using one algorithm that generates them in the multiplicity representation. There are applications which require the standard representation of integer partitions; for example, generating B-trees [4, 12]. Open problem is to find a constant time (worst case) delay algorithm for generating unrestricted integer partitions in standard representation, exclusive of the ouput. This means that there should be constant number of differences in parts in neighboring partitions. Algorithm [35] achieves later (with one minimal change difference) but fails to do than in constant time.

Another problem is to find the average number of parts in a partition. This would give precise total output size if partitions are to be printed out. In [2] two parallel algorithms that generate partitions on a linear array of $n\sqrt{n}$ processors in the standard (multiplicity) representations are given, and they run with constant delay. The costs of the algorithms (product of the number of processors used and time complexity) are $O(nP(n))$ and $O(\sqrt{n}P(n))$ (respectively) which may not be cost-optimal. An empirical study indicates that the average number of parts of a partition of n is $n^{2/3}$ (with relative error $<3\%$ for $n < 500$) while the number of different parts is $n^{4/9}$ (with relative error $<10\%$ for $30 \leq n \leq 120$).

Acknowledgements

This research is partially supported by NSERC.

References

[1] Akl, S. G. (1981). A comparison of combination generation methods, *ACM Trans. on Math. Software*, **7**(1), 42–45.

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

[3] Andrews, G. E. (1976). The theory of partitions, Addison-Wesley, Reading, Ma.

[4] Belbaraka, M. and Stojmenović, I. (1944). On generating B-trees with constant average delay and in lexicographic order, *Information Processing Letters*, **49**(1), 27–32.

[5] Berge, C. (1971). Principles of Combinatorics, Academic Press.

[6] Beyer, T. and Hedetniemi, S. M. (1980). Constant time generation of rooted trees, *SIAM J. Comput.*, **9**(4), 706–712.

[7] Dickson, L. E. (1971). History of the theory of numbers, Vol. II, Diophantine Analysis, Chelsea Publishing Co., New York.

[8] Djokic, B., Miyakawa, M., Sekiguchi, S., Semba, I. and Stojmenović, I. (1989). A fast algorithm for generating set partitions, *The Computer J.*, **32**(3), 281–282.

[9] Fenner, T. I. and Loizou, G. (1980). A binary tree representation and related algorithms for generating integer partitions, *The Computer J.*, **23**(4), 332–337.

[10] Fenner, T. I. and Loizou, G. (1983). Tree traversal related algorithms for generating integer partitions, *SIAM J. Computing*, **12**(3), 551–564.

[11] Fenner, T. I. and Loïzou, G. (1981). An analysis of two related loop-free algorithms for generating integer partitions, *Acta Informatica*, **16**, 237–252.

[12] Gupta, U. I., Lee, D. T. and Wong, C. K. (1983). Ranking and unranking of B-trees, *J. of Algorithms*, **4**, 51–60.

[13] Hardy, G. H. and Ramanujan, S. (1918). Asymptotic formulae in combinatorial analysis, *Proc. London Math. Soc.*, **17**, 237–252.

[14] Hall, M. (1967). Combinatorial Theory, Blaisdell, Waltham, Mass.

[15] Hikita, T. (1983). Listing and counting subtrees of equal size of a binary tree, *Inf. Proc. Lett.*, **17**, 225–229.

[16] James, K. R. and Riha, W. (1976). Algorithm for generating graphs of a given partition, *Computing*, **16**, 153–161.

[17] Knuth, D. E. (1968). The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison-Wesley, Reading, Ma.

[18] Lehmer, D. H. (1964). The machine tools of combinatorics, In: Applied Combinatorial Mathematics, Beckenbach (Ed.), Wiley, NY.

[19] Liu, C. L. (1968). Introduction to Combinatorial Mathematics, McGraw Hill.

[20] McKay, J. K. S. (1965). Partition generator, Alg. 263, *Comm. ACM*, **8**, 493.

[21] McKay, J. K. S. (1965). Number of restricted partitions of *n*, Alg. 262, *Comm. ACM*, **8**, 493.

[22] McKay, J. K. S. (1965). Map of partitions into integers, Alg. 264, *Comm. ACM*, **8**, 493.

[23] McKay, J. K. S. (1970). Partitions in natural order, Alg. 371, *Comm. ACM*, **13**, 52.

[24] Narayana, T. V., Mathsen, R. M. and Saranji, J. (1971). An algorithm for generating partitions and its applications, *J. Comb. Theory*, **11**, 54–61.

[25] Nijenhius, A. and Wilf, H. S. (1975). Combinatorial Algorithms, Academic Press, NY.

[26] Nijenhius, A. and Wilf, H. S. (1975). A method and two algorithms on the theory of partitions, *J. Comb. Theory A*, **18**, 219–222.

[27] Page, E. S. and Wilson, L. B. (1979). An Introduction to Computational Combinatorics, Cambridge Univ. Press.

[28] Read, R. C. (1980). A survey of graph generation techniques, *Lect. Notes in Math.*, **884**, 77–89.

[29] Riordan, J. (1958). An introduction to Combinatorial Analysis, John Wiley.

[30] Reingold, E. M., Nievergelt, J. and Deo, N. (1977). Combinatorial Algorithms, Prentice Hall, Englewood Cliffs, New Jersey.

[31] Riha, W. and James, K. R. (1976). Efficient algorithms for doubly and multiply restricted partitions, Algorithm 29, *Computing*, **16**, 163–168.

[32] Rubin, F. (1976). Partitions of integers, *ACM Transactions on Mathematical Software*, **2**(4), 364–374.

[33] Ruskey, F. (1978). Generating t-ary trees lexicographically, *SIAM J. Comput.*, **7**, 492–509.

[34] Sanchis, L. (1992). Counting and generating integer partitions in parallel, *Proc. Int. Conf. on Computing and Information ICCI'92*, pp. 54–57.

[35] Savage, C. D. (1989). Gray code sequences of partitions, *J. of Alg.*, **10**, 577–595.

[36] Sedgewick, R. (1977). Permutation generation methods, *Comp. Surv.*, **9**, 137–164.

[37] Stockmal, F. (1962). Generation of partitions in part-count form, Algorithm 95, *Comm. ACM*, **5**, 344.

[38] Tomasi, C. (1982). Two simple algorithms for the generation of partitions of an integer, *Alta Frequenza*, LI, **6**, 352–356.

[39] van Baronaigien, D. R. (1991). A loopless algorithm for generating binary tree sequences, *Information Processing Letters*, **39**, 189–194.

[40] Wells, M. B. (1971). Elements of Combinatorial Computing, Pergamon Press, Oxford.

[41] White, J. S. (1970). Restricted partition generator, Alg. 374, *Comm. ACM*, **13**, 120.

[42] White, J. S., Number of doubly restricted partitions, Alg. 373, *ibid.*

[43] Zaks, S. and Richards, D. (1979). Generating trees and other combinatorial objects lexicograpically, *SIAM J. on Comput.*, **8**(1), 73–81.

[44] Zoghbi, A. (1993). Algorithms for generating integer partitions, M. S. Thesis, University of Ottawa.