A Fast Algorithm for Generating Set Partitions

M. C. ER
Department of Computer Science, The University of Western Australia, Nedlands, WA6009, Australia

A recursive algorithm for generating all partitions of the set \{1, 2, ..., n\} is presented. The average time complexity per partition generated of this algorithm is \(O(n^3)\). This algorithm runs faster than the previously fastest algorithm, whose average time complexity per partition is \(O(4^n)\). An empirical test confirms that this is indeed the case.

Received January 1987

1. INTRODUCTION

Let \(Z = \{1, 2, 3, ..., n\}\). A partition of the set \(Z\) consists of \(k\) classes \(\pi_1, \pi_2, ..., \pi_k\) such that \(\pi_i \cap \pi_j = \emptyset\) if \(i \neq j\), \(\pi_i \cup \pi_j \cup ... \cup \pi_k = Z\), and \(\pi_i \neq \emptyset\) for \(1 \leq i \leq k\). In this paper we consider the problem of generating all partitions of the set \(Z\).

This problem was considered by Nijenhuis and Wilf, who gave a generating algorithm. Another generating algorithm was presented by Kaye, which has the property that a successive partition differs from its predecessor by one element in a class. Most recently, Semb's constructed a set-partition algorithm which is faster than all the above-mentioned algorithms.

The purpose of this paper is to derive an efficient algorithm for generating all partitions of the set \(Z\), which is faster than Semb's algorithm as confirmed by empirical results. The complexity of this algorithm will also be established.

2. GENERATING ALGORITHM

Let \(C = c_1, c_2, c_3, ..., c_n\) be a codeword of a partition of the set \(Z\), such that \(c_i = j\) if \(i\) is in \(\pi_j\). A listing of all partitions of the set \(Z\), when \(n = 4\), and their codewords is presented in Fig. 1.

From the listing of codewords as shown in Fig. 1, it is apparent that \(1 \leq c_i \leq i\). In other words, \(i \in Z\) need not be placed in a class \(\pi_j\), such that \(j > i\). Furthermore, we observe that the codewords of all partitions of the set \(\{1, 2, 3, 4\}\) and their codewords.

<table>
<thead>
<tr>
<th>Partitions</th>
<th>Codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1234)</td>
<td>1111</td>
</tr>
<tr>
<td>(123)(4)</td>
<td>1112</td>
</tr>
<tr>
<td>(124)(3)</td>
<td>1121</td>
</tr>
<tr>
<td>(12)(34)</td>
<td>1122</td>
</tr>
<tr>
<td>(12)(3)(4)</td>
<td>1123</td>
</tr>
<tr>
<td>(13)(24)</td>
<td>1211</td>
</tr>
<tr>
<td>(13)(2)(4)</td>
<td>1212</td>
</tr>
<tr>
<td>(14)(23)</td>
<td>1221</td>
</tr>
<tr>
<td>(1)(234)</td>
<td>1222</td>
</tr>
<tr>
<td>(1)(23)(4)</td>
<td>1223</td>
</tr>
<tr>
<td>(14)(2)(3)</td>
<td>1231</td>
</tr>
<tr>
<td>(1)(2)(34)</td>
<td>1232</td>
</tr>
<tr>
<td>(1)(2)(3)(4)</td>
<td>1233</td>
</tr>
</tbody>
</table>

Figure 1. A listing of all partitions of the set \(\{1, 2, 3, 4\}\) and their codewords.

Figure 2. A recursive algorithm for generating all partitions of the set \(\{1, 2, ..., n\}\) via their corresponding codewords.

```
procedure SetPartitions(n: integer);
    var c: codeword;
    procedure SP(m, p: integer);
    var i: integer;
    begin
        if p > n then PrintPartition(c)
        else begin
            for i := 1 to m do begin
                c[p] := i;
                SP(m, p + 1)
            end;
            c[p] := m + 1;
            SP(m + 1, p + 1)
        end;
    end (SP);
    begin
        SP(0, 1)
    end (SetPartitions);
end
```

\{1, 2, ..., n\} can be obtained from the codewords of all partitions of the set \(\{1, 2, ..., n-1\}\) by appending \(c_n\) to the respective codewords. The range of values that \(c_n\) may assume is \(1, ..., \max(c_1, c_2, ..., c_{n-1}) + 1\).

A recursive algorithm may be constructed based on these properties, and is shown in Fig. 2. Note that in the inner procedure \(SP(m, p)\), the parameter \(m = \max(c_1, c_2, ..., c_{p-1})\) and the parameter \(p\) indicates the current
digit of a codeword under consideration. The procedure
PrintPartition simply prints the content of a codeword,
converting it into the corresponding partition. As this
procedure is quite straightforward, it is omitted here.

In the interest of constructing an efficient algorithm
for generating set partitions, the inner procedure $SP$
may be fine tuned as shown in Fig. 3 by reducing the number
of procedure calls.

3. ANALYSIS OF ALGORITHM

To analyse the complexity of the generating algorithm,
we may trace the activation tree of the procedure $SP$
shown in Fig. 3. An example of the activation tree is
shown in Fig. 4, while generating the codewords of all
partitions of the set $\{1, 2, 3, 4\}$. From Fig. 4 we see that
each edge of the activation tree corresponds to an assignment of
a value to a digit of a codeword. The number of edges is a reasonable measure
of the work involved, and may be used to characterize
the complexity of the generating algorithm. The number
of edges at level $i$ is equal to the number of partitions of
the set $\{1, 2, ..., i\}$, which is given by the well-known Bell
number $B_i$. Let $A(n)$ be the total number of edges in the
activation tree for generating the codewords of all
partitions of the set $Z$. Clearly, we have

$$A(n) = \sum_{i=1}^{n} B_i$$

Since the total number of codewords generated is $B_n$, the
average time complexity ($t_{av}$) per codeword of the
procedure $SP$ is

$$t_{av} = A(n)/B_n < 1.6.$$  

We may summarise the result as follows.

**Theorem 1**

The average time complexity per partition of the
algorithm SetPartitions is $\Theta(1.6)$.

4. PERFORMANCE EVALUATION

To evaluate the actual performance of our algorithm and
compare it with that of Semba's algorithm, both
algorithms have been implemented in Pascal and compiled
under the Berkeley's Pascal compiler. The actual
running times, averaged over 3 runs, of both algorithms
on a VAX 11/750 computer running under Unix 4.2
BSD are summarised in Table 1. The result shows clearly
that our algorithm SetPartitions is superior to Semba's
algorithm in all cases.

<table>
<thead>
<tr>
<th>$n$</th>
<th>Semba's algorithm</th>
<th>SetPartitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.53</td>
<td>0.50</td>
</tr>
<tr>
<td>9</td>
<td>3.10</td>
<td>1.90</td>
</tr>
<tr>
<td>10</td>
<td>16.97</td>
<td>10.27</td>
</tr>
<tr>
<td>11</td>
<td>96.70</td>
<td>59.00</td>
</tr>
<tr>
<td>12</td>
<td>586.57</td>
<td>371.93</td>
</tr>
<tr>
<td>13</td>
<td>3778.43</td>
<td>2327.40</td>
</tr>
</tbody>
</table>

5. CONCLUDING REMARKS

We have succeeded in deriving an efficient algorithm
SetPartitions for generating all partitions of the set $\{1, 2, ...
..., n\}$. Its average time complexity per partition generated
is $\Theta(1.6)$. Compared with Semba's algorithm, whose
average time complexity per partition generated is $\Theta(4)$,
our algorithm is preferred. This is also confirmed by
empirical results.

Since Semba's algorithm was previously the best
algorithm, now our algorithm SetPartitions runs faster
than Semba's algorithm; therefore our algorithm is
currently the fastest algorithm for generating all
partitions of the set $\{1, 2, ..., n\}$. It is also of interest to note that the codewords generated are always in lexicographic order.

REFERENCES

1. R. A. Kaye, A Gray code for set partitions. Information
2. A. Nijenhuis and H. S. Wilf, Combinatorial Algorithms.
3. I. Semba, An efficient algorithm for generating all
partitions of the set $\{1, 2, ..., n\}$. Journal of Information