Generating partitions.

A partition of a positive integer N is a sequence of positive integers of which the sum is equal to N.

If $(p_0, p_1, \ldots, p_{k-1})$ and $(q_0, q_1, \ldots, q_{\ell-1})$ are two different partitions of N, and if m is the smallest index such that $p_m \neq q_m$, then $(p_0, p_1, \ldots, p_{k-1})$ is said to precede $(q_0, q_1, \ldots, q_{\ell-1})$ in the lexicographic order if and only if $p_m < q_m$.

a) Write a program that can, if so desired, generate all partitions of N in the lexicographic order.

b) Write a program that can generate all ascending partitions of N, i.e. all partitions $(p_0, p_1, \ldots, p_{k-1})$ of N such that $p_0 \leq p_1 \leq \cdots \leq p_{k-1}$, in the lexicographic order.

c) Write a program that, given L and N such that $1 \leq L \leq N$, generates all ascending partitions of N which have length L, i.e. which consist of L elements, in the lexicographic order.

*

Before tackling one of the specific problems a) or b) or c) we shall first focus our attention on the more general problem of generating a set of sequences in (lexicographic) order. We shall do so in some detail because, firstly, the problem is of a frequently recurring type and, secondly, the reasoning is not quite trivial to the novice.

+

Let V be a finite set of sequences over a well-ordered alphabet, such as the integers. Then we are able to impose on V the lexicographic ordering, which is a total ordering. Let Xf and Xl be the (lexicographically) first and last sequence in V, respectively, and let $S(X)$ be the (lexicographic) successor of a sequence X, $X \neq Xl$, in V; then a program that generates all sequences of V in (lexicographic) order is

```
Pr0:        X := Xf ; print(X);
            do X ≠ Xl → X := S(X); print(X) od.
```

If it is possible to find a sequence X such that $X \neq Xl \wedge S(X) = Xf$, then program Pr0 can be "simplified" a little bit into

```
Pr1:        "X : (X ≠ Xl ∧ S(X) = Xf)";
            do X ≠ Xl → X := S(X); print(X) od,
```

which is sometimes to be preferred over Pr0.

Another transcription of Pr0 is possible if a sequence Xll can be found such that $Xll = S(Xl) \wedge Xll \neq Xf$:

```
Pr2:        X := Xf;
            do X ≠ Xll → print(X); X := S(X) od.
```

The programs Pr1 and Pr2 are "simplifications" over Pr0 in the sense that the statements to print a sequence are textually more isolated.

The proof that these programs indeed generate all sequences of V in (the lexicographic) order is left to the reader who feels urged to do so.

<u>Note</u>: In the above we have not used that V consists of sequences, let alone that they are ordered lexicographically. Indeed, the programs given are adequate to generate any finite, totally ordered set.
If, however, it is about defining a total ordering on sequences there is only few opportunity to invent something else than the lexicographic ordering. (End of Note.)

+

For two finite sequences $X = (x_0, x_1, \ldots, x_{k-1})$ and $Y = (y_0, y_1, \ldots, y_{l-1})$ we take as our universal definition of lexicographic ordering

$$(X < Y) \; = \; (x_m < y_m) \; ,$$

where m equals the smallest index such that $x_m \neq y_m$. We observe that, for $X \neq Y$, such a value of m always exists.
<u>Note</u>: One might argue that this definition doesn't apply to the sequences "America" and "American", which indeed it doesn't. But this can always be remedied by postfixing each sequence with some strange character as, for instance, \$.
(End of Note.)

Next, consider three sequences X, Y and Z say, satisfying

$$X \prec Y \not\prec Z.$$

Let m be the smallest index such that $x_m \neq y_m$, and let n be the smallest index such that $x_n \neq z_n$. Then we can infer

$$m \geq n,$$

which we shall prove.
From $X \prec Y$ it follows

$$(\forall i: 0 \leq i < m: \quad x_i = y_i) \quad \wedge \quad x_m < y_m,$$

and from $X \prec Z$

$$(\forall i: 0 \leq i < n: \quad x_i = z_i) \quad \wedge \quad x_n < z_n.$$

If $m < n$ would hold we would have

$$(\forall i: 0 \leq i < m: \quad y_i = z_i) \quad \wedge \quad x_m = z_m < y_m,$$

or $\quad Z \prec Y$,
contrary to the assumption $Y \not\prec Z$. □

The above property tells us that among

all lexicographic successors of X there is no sequence which shares with X an initial segment longer than the initial segment X shares with its immediate successor $S(X)$. Therefore we are interested in the maximum value of m such that there exists a sequence $Y: Y \succ X$ in V, which has the form

$$Y = (x_0, x_1, \ldots, x_{m-1}, y_m, \ldots, y_{l-1}) \quad , \quad l > m.$$

Then $Y = S(X)$ if the sequence $(y_m, \ldots, y_{l-1})$ is chosen subject to the conditions

i) $y_m > x_m$, y_m minimal
ii) Y belongs to V
iii) $(y_m, \ldots, y_{l-1})$ is lexicographically minimal.

Such a choice is always possible thanks to the definition of m.

*

Now, let us return to the specific problems posed.

ad a) In the case where V equals the set of all partitions of a given positive integer N, it is absolutely obvious that

$$Xf = (1, 1, \ldots, 1) \quad , \quad \text{a sequence of } N \text{ ones,}$$
$$Xl = (N), \quad \text{a sequence of one } N.$$

So let us concentrate on the construction of $S(X)$, where X is a partition of N,

distinct from Xl.
If $X = (x_0, x_1, \ldots, x_{k-1})$, we observe

- $k \geq 2$
- x_{k-1} follows uniquely from $(x_0, x_1, \ldots, x_{k-2})$, so that X and $S(X)$ can not have $(x_0, x_1, \ldots, x_{k-2})$ in common as initial segment
- $(x_0, x_1, \ldots, x_{k-3}, x_{k-2} + x_{k-1})$ is a partition of N exceeding X in the lexicographic order.

Hence, choosing $m = k-2$,

$$S(X) = (x_0, x_1, \ldots, x_{m-1}, y_m, \ldots, y_{l-1})$$

in which $S(X)$ satisfies i) through iii)
Since both X and $S(X)$ are partitions of N, the sequence $(y_m, \ldots, y_{l-1})$ must be a partition of $N - (x_{k-2} + x_{k-1})$, in which case ii) is fulfilled. If it were only for iii), a sequence of $N - (x_{k-2} + x_{k-1})$ ones would be an appropriate choice for $(y_m, \ldots, y_{l-1})$, but in combination with i) we had better take

$$(y_m, \ldots, y_{l-1}) = (x_{k-2} + 1, 1, \ldots, 1), \text{ i.e.}$$

a sequence starting with the number $x_{k-2} + 1$ and ending with $x_{k-1} - 1$ ones.

Thus $S(X)$ is defined on all integer sequences with at least two elements, offering us the possibility to use Pr1 instead of Pr0, because

$$(0, N) \neq Xl \quad \wedge \quad S((0, N)) = Xf \ .$$

Taking an integer array x and an integer variable k to represent the sequence X,

$$X = (x(0), x(1), \ldots, x(k-1)),$$

the program becomes

```
$x(0) := 0;$  $x(1) := N;$  $k := 2;$  $\{X \neq Xl \wedge S(X) = Xf\}$
do $\neg(x(0) = N \wedge k = 1) \rightarrow \{X \neq Xl$, hence $k \geq 2\}$
    $r := x(k-1) - 1;$  $k := k-1;$  $\{k \geq 1\}$ $\{r \geq 0\}$
    $x(k-1) := x(k-1) + 1;$
    $\{(\Sigma i: 0 \leq i < k: x(i)) + r = N\}$
    do $r \neq 0 \rightarrow x(k) := 1;$ $r := r-1;$ $k := k+1$ od;
    $\{(\Sigma i: 0 \leq i < k: x(i)) = N$, hence
     permission to print$\}$
od.
```

Note: Optimizations like the replacement of the guard of the outer repetitive construct by $x(0) \neq N$ or by $k \neq 1$ are left as exercises.
(End of Note.)

+

We end the discussion of this example with a somewhat detailed analysis of the program's so-called "time-complexity", i.e. the number of steps prescribed by the program when run on a sequential machine.
To begin with we count the number of partitions of N. An easy way to do this is to split the set of all partitions into disjoint classes, to count the number of partitions in each class, and to add these numbers. It is easy to classify

the partitions by looking at their last elements, for instance. The number of partitions of N that have z as their last element equals the number of partitions of N-z, obviously so. Therefore, when defining

$A(n)$ = number of partitions of n,

we have

$$A(N) = (\Sigma z: 1 \leq z \leq N: A(N-z))$$
$$= (\Sigma z: 0 \leq z < N: A(z)) \quad , \quad N \geq 1,$$

$$A(0) = 1 \quad .$$

We solve this recurrence scheme as follows:

$$A(N) = (\Sigma z: 0 \leq z < N: A(z))$$
$$= (\Sigma z: 0 \leq z < N-1: A(z)) + A(N-1)$$
$$= A(N-1) + A(N-1) \quad , \quad N \geq 2,$$
$$= 2 * A(N-1) \text{ , hence}$$

$$A(N) = 2^{N-1} \quad , \quad N \geq 1 \quad .$$

Thus, the repeatable statement of the outer repetitive construct is invoked 2^{N-1} times precisely.

For a partition whose last element equals z the inner repeatable statement is invoked z-1 times precisely. Therefore we have that

$$B(N) = (\Sigma z: 1 \leq z < N: (z-1) * A(N-z)) + (N-1)$$

is the total number of invocations of the inner repeatable statement, precisely. (The term $(N-1)$ originates from the partition $(0, N)$ which is dealt with separately because it is not captured by A.)

Substituting our result for A, we have to compute

$$B(N) = \left(\Sigma\, z : 1 \le z < N : (z-1) \cdot 2^{N-z-1}\right) + (N-1) .$$

It can be 'seen' rather instantaneously, or shown by mathematical induction, or derived by elementary calculus, that

$$B(N) = 2^{N-1} - 1 .$$

(Yours Truly derived it, checked it by mathematical induction, and then 'saw' why the answer was correct.)

We conclude that the inner repeatable statement is in toto even --slightly-- less often repeated than the outer one, so that the time-complexity of our program is the time-complexity of the outer repetitive construct, which is $\mathcal{O}(2^N)$.

So far for the treatment of example a)

*

ad b) In the case where V is the set of ascending partitions of N, i.e. the set of all partitions $(p_0, p_1, \dots, p_{k-1})$ of N satisfying $p_0 \le p_1 \le \dots \le p_{k-1}$, it is again absolutely obvious that

$$Xf = (1, 1, \dots, 1) \quad , \text{ a sequence of N ones}$$
$$Xl = (N) \quad , \text{ a sequence of one N.}$$

So, let us concentrate on the construction of S(X), where X is an ascending partition of N, distinct from Xl.
If $X = (x_0, x_1, \dots, x_{k-1})$, we observe

- $k \ge 2$
- x_{k-1} follows uniquely from $(x_0, x_1, \dots, x_{k-2})$ so that X and S(X) can not have $(x_0, x_1, \dots, x_{k-2})$ in common as initial segment
- $(x_0, x_1, \dots, x_{k-3}, x_{k-2} + x_{k-1})$ is an ascending partition of N exceeding X in the lexicographic order.

Hence, choosing $m = k-2$,

$$S(X) = (x_0, x_1, \dots, x_{m-1}, y_m, \dots, y_{l-1})$$

in which S(X) satisfies i) through iii).
Since both X and S(X) are ascending partitions of N, the sequence $(y_m, \dots, y_{l-1})$ must be an ascending partition of $N - (x_{k-2} + x_{k-1})$, satisfying $y_m \ge x_{m-1}$ if $m \ge 1$, in which case ii) is fulfilled. The requirement $y_m \ge x_{m-1}$ is implied as soon as i), $y_m > x_m$, is fulfilled. The minimal value

of y_m satisfying $y_m > x_m$ is $yy = x_m + 1$. In order to satisfy iii) as well, we have to choose $(y_m, \dots, y_{l-1}) = (yy, \dots, yy, z)$, where $yy \leq z < 2 * yy$, i.e. a sequence of zero or more numbers yy followed by the number z, such that $(\sum i: m \leq i < l: y_i) = N - (x_{k-2} + x_{k-1})$.

With this definition of $S(X)$ we observe that

$$(0, N) \neq Xl \quad \wedge \quad S((0,N)) = Xf ,$$

offering us the opportunity to use Pr1.

Representing the sequence X by

$$X = (x(0), x(1), \dots, x(k-1))$$

the program becomes:

```
x(0) := 0; x(1) := N; k := 2; {X ≠ Xl ∧ S(X) = Xf}
do ¬(x(0) = N ∧ k ≠ 1) → {X ≠ Xl, hence k ≥ 2}
      r := x(k-1) - 1; k := k-1; {k ≥ 1}{r ≥ 0}
      yy := x(k-1) + 1; k := k-1; {k ≥ 0}{yy ≥ 1}
      {(Σ i: 0 ≤ i < k: x(i)) + yy + r = N}{r ≥ 0}
      do r ≥ yy → x(k) := yy; r := r-yy; k := k+1 od;
      {0 ≤ r < yy}
      x(k) := yy + r; k := k+1;
      {(Σ i: 0 ≤ i < k: x(i)) = N, hence
       permission to print}
od.
```

+

If we wish to find the program's time-complexity, we have to count the number of ascending partitions of N. This, however, is so difficult that its undertaking falls outside the scope of our current interest. In fact it is even worse, since nobody has ever found an elementary formula to express this number.
In not too elementary books on combinatorial analysis or on (analytic) number theory it can be found that the number is of the order of magnitude

$$\frac{1}{4N\sqrt{3}}\, e^{\pi\sqrt{\frac{2}{3}N}}$$

for large N : an impressive result!
Hence, though less than 2^{N-1}, the number grows exponentially as N grows, i.e. it explodes.

So far for the treatment of example b)

*

ad c) In this case V is the set of all ascending partitions of N which have a prescribed length L, i.e. the set of all ascending partitions $(p_0, p_1, \ldots, p_{L-1})$ of N.

<u>Note</u>: As the reader may guess, the treatment of this example is not easier than the treatment of the previous one; for if it were easier, we would have dealt with this one first and we would

have solved the previous one by letting L take on all values 1 through N, which we didn't. (End of Note.)

It is absolutely obvious that

$$Xf = (1, 1, \ldots, 1, 1 + N - L).$$

The value of Xl is a little bit less apparent, and we concentrate on the construction of $S(X)$, $X \neq Xl$, first.
If $X = (x_0, x_1, \ldots, x_{L-1})$ then $S(X)$ has the form $(x_0, x_1, \ldots, x_{m-1}, yy, yy, \ldots, yy + r)$ in which

- $yy = x_m + 1 \quad \wedge \quad r \geq 0$
- $(L - m) * yy + r = (\Sigma i : m \leq i < L : x(i))$
- $0 \leq m < L$.

But also, if m is chosen as the maximal value for which these three conditions are satisfied then $S(X) = (x_0, x_1, \ldots, x_{m-1}, yy, yy, \ldots, yy + r)$. (The reader is requested to check this.)
Hence, as a corollary, we are interested in the maximum value of m: $0 \leq m < L$ such that

$$(L - m) * (x_m + 1) \leq (\Sigma i : m \leq i < L : x(i)) \quad . (0)$$

Now back to Xl. Its value is somewhat tedious, and so is the test $X \neq Xl$. So, let us seek to use Pr2 which doesn't mention Xl at all. If $X = Xl$ no value of m exists for which (0) holds and $0 \leq m < L$. But (0)

evidently holds if we are prepared to allow for $m = -1$ and if we define $x_{-1} = -1$.
Then our function S is defined on Xl as well, viz.

$$S((x_{-1}, Xl)) =$$
$$S((-1, Xl)) =$$
$$(0, 0, \ldots, 0, N-1) = Xll,$$

a sequence of length $L+1$.

Using Pr2, we obtain

```
x(-1) := -1;
m := 0; do $m \neq L \rightarrow$ x(m) := 1; m := m+1 od;
x(L-1) := x(L-1) + N-L;
{$X = Xf$}
do $x(-1) \neq 0 \rightarrow$ {$X \neq Xll$, hence
        permission to print}
    m := L-1; r := x(m);
    {$r = (\sum i : m \leq i < L : x(i))$}
    do $(L-m) * (x(m)+1) > r \rightarrow$
            m := m-1; r := r + x(m)
    od;
    yy := x(m) + 1;
    {$(L-m) * yy \leq r$}
    {$(\sum i : 0 \leq i < m : x(i)) + r = N$}
    do $m \neq L \rightarrow$
            x(m) := yy; m := m+1; r := r-yy
    od;
    x(L-1) := x(L-1) + r
od.
```

+

For some values of L, such as L = 1 or L = 2 or L = N, the number of ascending partitions of N of length L grows polynomially as N grows. From this we are not allowed to conclude that it therefore will be true for all values of L, in spite of the fame and the wide-spread use of the principle of Poor Man's Induction.
Since the number of ascending partitions of N with unrestricted length grows exponentially as N grows, i.e. it explodes, we must conclude that for some value of L the number of ascending partitions of length L explodes as well.

So far for the treatment of example c), and so far for partitions.

*

The examples dealt with above have served as programming exercises during such a long time that, in the author's mind, they have lost most of their appeal. Nevertheless, students -- being refreshed each year -- have always been struggling in solving the problems. One reason for their struggle was that they didn't deal with the problem of generating a finite set of things in isolation. They were thoroughly unfamiliar with general schemes like Pr0 or the derived versions Pr1 or Pr2. Another reason for their struggle was their inability to deal with the well-known notion of lexicographic succession <u>in precision</u>. These two observations can be taken as a justification for the enclosure of the above text; but there is more.

Programs of the above type make, when run on a machine, an exploding demand on time, i.e. we can see them terminate for some small value of the argument N and we will never see them terminate for N a little bit larger.
There is, however, a general interest in these kinds of programs, exhibited by for instance "Compulsive Programmers", Artificial Intelligentsia and by Commerce. The interest is almost always confined to those cases where N is just a little bit too large. What happens is that they, anxious to see the outcome of the programs, start to improve the efficiency. They do so by, firstly, looking for better algorithms (which is fine), secondly by drawing on a repertoire of coding tricks, thereby mangling the logical structure of the programs, and thirdly by tailoring the program texts to all the peculiarities of the local machinery, thereby making their designs unexplainable and unfit for general usage. If the operation was successful, N appears to be just a little bit too small.
The main problem is that they failed to broaden the insight in the nature of computer programming, on the contrary: they obscured it.
Moreover, trying to resist such temptations makes life easier, healthier, cheaper and it will not lead us less further too much. When faced with programs that make an exploding demand on time we had better be able to understand what that

means.

A final remark concerns the fact that partitions are often regarded as so-called "combinatorial objects" and that programs operating on such objects are often classified as "combinatorial algorithms". As the reader may have noticed we have not dropped those terms during the discussion.
It is felt that such classifications of programming problems is often useless, quite often misleading and sometimes dangerous. It is particularly dangerous in those cases where indices are called "keys", sequence-elements are called "employers", and where programs are called "Data Base"!

W.H.J. Feijen,
Sterksel,
March 24, 1981