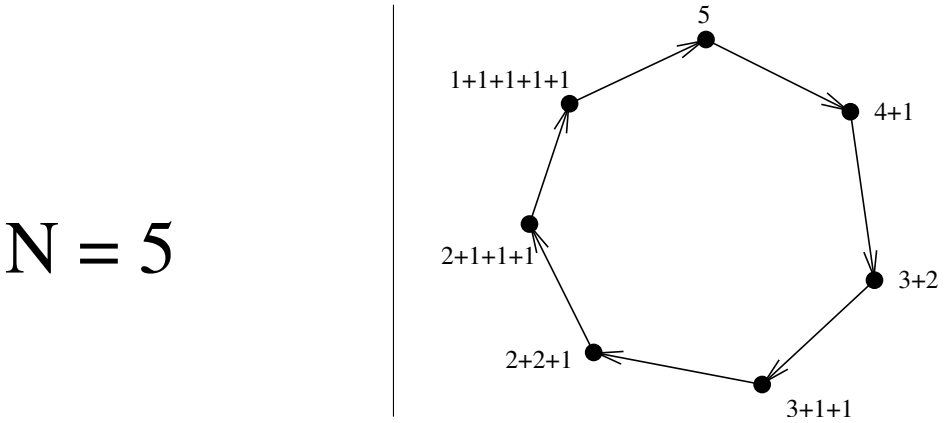

INPUT OUTPUT

14.6 Generating Partitions

Input description: An integer n.

Problem description: Generate (1) all, or (2) a random, or (3) the next integer or set partitions of length n.

Discussion: There are two different types of combinatorial objects denoted by the word "partition," namely integer partitions and set partitions. They are quite different beasts, but it is a good idea to make both a part of your vocabulary:

- *Integer partitions* are multisets of nonzero integers that add up exactly to n. For example, the seven distinct integer partitions of 5 are {5}, {4,1}, {3,2}, {3,1,1}, {2,2,1}, {2,1,1,1}, and {1,1,1,1,1}. An interesting application I encountered that required generating integer partitions was in a simulation of nuclear fission. When an atom is smashed, the nucleus of protons and neutrons is broken into a set of smaller clusters. The sum of the particles in the set of clusters must equal the original size of the nucleus. As such, the integer partitions of this original size represent all the possible ways to smash an atom.

- *Set partitions* divide the elements $1, \ldots, n$ into nonempty subsets. There are 15 distinct set partitions of $n = 4$: {1234}, {123,4}, {124,3}, {12,34}, {12,3,4}, {134,2}, {13,24}, {13,2,4}, {14,23}, {1,234}, {1,23,4}, {14,2,3}, {1,24,3}, {1,2,34}, and {1,2,3,4}. Several algorithm problems return set partitions as results, including *vertex/edge coloring* and *connected components*.

Although the number of integer partitions grows exponentially with n, they do so at a refreshingly slow rate. There are only 627 partitions of $n = 20$. It is even possible to enumerate all integer partitions of $n = 100$, since there there are only 190,569,292 of them.

The easiest way to generate integer partitions is to construct them in lexicographically decreasing order. The first partition is $\{n\}$ itself. The general rule is to subtract 1 from the smallest part that is > 1 and then collect all the 1's so as to match the new smallest part > 1. For example, the partition following $\{4, 3, 3, 3, 1, 1, 1, 1\}$ is $\{4, 3, 3, 2, 2, 2, 1\}$, since the five 1's left after $3 - 1 = 2$ becomes the smallest part are best packaged as 2,2,1. When the partition is all 1's, we have completed one pass through all the partitions.

This algorithm is sufficiently intricate to program that you should consider using one of the implementations below. In either case, test it to make sure that you get exactly 627 distinct partitions for $n = 20$.

Generating integer partitions uniformly at random is a trickier business than generating random permutations or subsets. This is because selecting the first (i.e. largest) element of the partition has a dramatic effect on the number of possible partitions that can be generated. Observe that no matter how large n is, there is only one partition of n whose largest part is 1. The number of partitions of n with largest part at most k is given by the recurrence

$$P_{n,k} = P_{n-k,k} + P_{n,k-1}$$

with the two boundary conditions $P_{x-y,x} = P_{x-y,x-y}$ and $P_{n,1} = 1$. This function can be used to select the largest part of your random partition with the correct probabilities and, by proceeding recursively, to eventually construct the entire random partition. Implementations are cited below.

Random partitions tend to have large numbers of fairly small parts, best visualized by a Ferrers diagram as in Figure 14.1. Each row of the diagram corresponds to one part of the partition, with the size of each part represented by that many dots.

Set partitions can be generated using techniques akin to integer partitions. Each set partition is encoded as a *restricted growth function*, $a_1, \ldots, a_n$, where $a_1 = 0$ and $a_i \leq 1 + \max(a_1, \ldots, a_i)$, for $i = 2, \ldots, n$. Each distinct digit identifies a subset, or *block*, of the partition, while the growth condition ensures that the blocks are sorted into a canonical order based on the smallest element in each block. For example, the restricted growth function $0, 1, 1, 2, 0, 3, 1$ defines the set partition $\{\{1, 5\}, \{2, 3, 7\}, \{4\}, \{6\}\}$.

Since there is a one-to-one equivalence between set partitions and restricted growth functions, we can use lexicographic order on the restricted growth functions to order the set partitions. Indeed, the fifteen partitions of $\{1, 2, 3, 4\}$ listed above are sequenced according to the lexicographic order of their restricted growth function (check it out).

We can use a similar counting strategy to generate random set partitions as we did with integer partitions. The Stirling numbers of the second kind $\{{n \atop k}\}$ count the

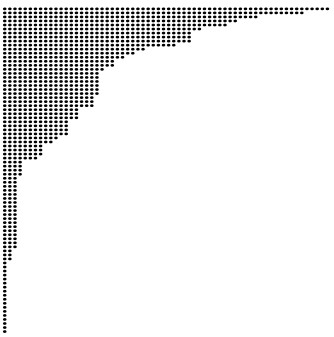

Figure 14.1: The Ferrers diagram of a random partition of $n = 1000$

number of partitions of $\{1, \ldots, n\}$ with exactly k blocks. They are computed using the recurrence

$$\left\{ {n \atop k} \right\} = \left\{ {n-1 \atop k-1} \right\} + k \left\{ {n-1 \atop k} \right\}$$

with the boundary conditions $\left\{ {n \atop n} \right\} = \left\{ {n \atop 1} \right\} = 1$. The reader is referred to the Notes and Implementations section for more details.

Implementations: Kreher and Stinson [KS99] generate both integer and set partitions in lexicographic order, including ranking/unranking functions. These implementations in C are available at *http://www.math.mtu.edu/~kreher/cages/Src.html*.

The *Combinatorial Object Server* (*http://theory.cs.uvic.ca/*) developed by Frank Ruskey of the University of Victoria is a unique resource for generating permutations, subsets, partitions, graphs, and other objects. An interactive interface enables you to specify which objects you would like returned. Implementations in C, Pascal, and Java are available for certain types of objects.

C++ routines for generating an astonishing variety of combinatorial objects, including integer partitions and compositions, are available in the combinatorics package at *http://www.jjj.de/fxt/*.

Nijenhuis and Wilf [NW78] is a venerable but still excellent source on generating combinatorial objects. They provide efficient Fortran implementations of algorithms to construct random and sequential integer partitions, set partitions, compositions, and Young tableaux. See Section 19.1.10 (page 661) for details.

Combinatorica [PS03] provides Mathematica implementations of algorithms to construct random and sequential integer partitions, compositions, strings, and

Young tableaux, as well as to count and manipulate these objects. See Section 19.1.9 (page 661).

Notes: The best reference on algorithms for generating both integer and set partitions is Knuth [Knu05b]. Good expositions include [KS99, NW78, Rus03, PS03]. Andrews [And98] is the primary reference on integer partitions and related topics, with [AE04] his more accessible introduction.

Integer and set partitions are both special cases of *multiset partitions*, or set partitions of nonnecessarily distinct numbers. In particular, the distinct set partitions on the multiset $\{1, 1, 1, \ldots, 1\}$ correspond exactly to integer partitions. Multiset partitions are discussed in Knuth [Knu05b].

The (long) history of combinatorial object generation is detailed by Knuth [Knu06]. Particularly interesting are connections between set partitions and a Japanese incense burning game, and the naming of all 52 set partitions for $n = 5$ with distinct chapters from the oldest novel known, *The Tale of Genji*.

Two related combinatorial objects are Young tableaux and integer compositions, although they are less likely to emerge in applications. Generation algorithms for both are presented in [NW78, Rus03, PS03].

Young tableaux are two-dimensional configurations of integers $\{1, \ldots, n\}$ where the number of elements in each row is defined by an integer partition of n. Further, the elements of each row and column are sorted in increasing order, and the rows are left-justified. This notion of shape captures a wide variety of structures as special cases. They have many interesting properties, including the existence of a bijection between pairs of tableaux and permutations.

Compositions represent the possible assignments of a set of n indistinguishable balls to k distinguishable boxes. For example, we can place three balls into two boxes as {3,0}, {2,1}, {1,2}, or {0,3}. Compositions are most easily constructed sequentially in lexicographic order. To construct them randomly, pick a random $(k-1)$-subset of $n+k-1$ items using the algorithm of Section 14.5 (page 452), and count the number of unselected items between the selected ones. For example, if $k = 5$ and $n = 10$, the $(5-1)$ subset {1,3,7,14} of $1, \ldots, (n+k-1) = 14$ defines the composition {0,1,3,6,0}, since there are no items to the left of element 1 nor right of element 14.

Related Problems: Generating permutations (see page 448), generating subsets (see page 452).