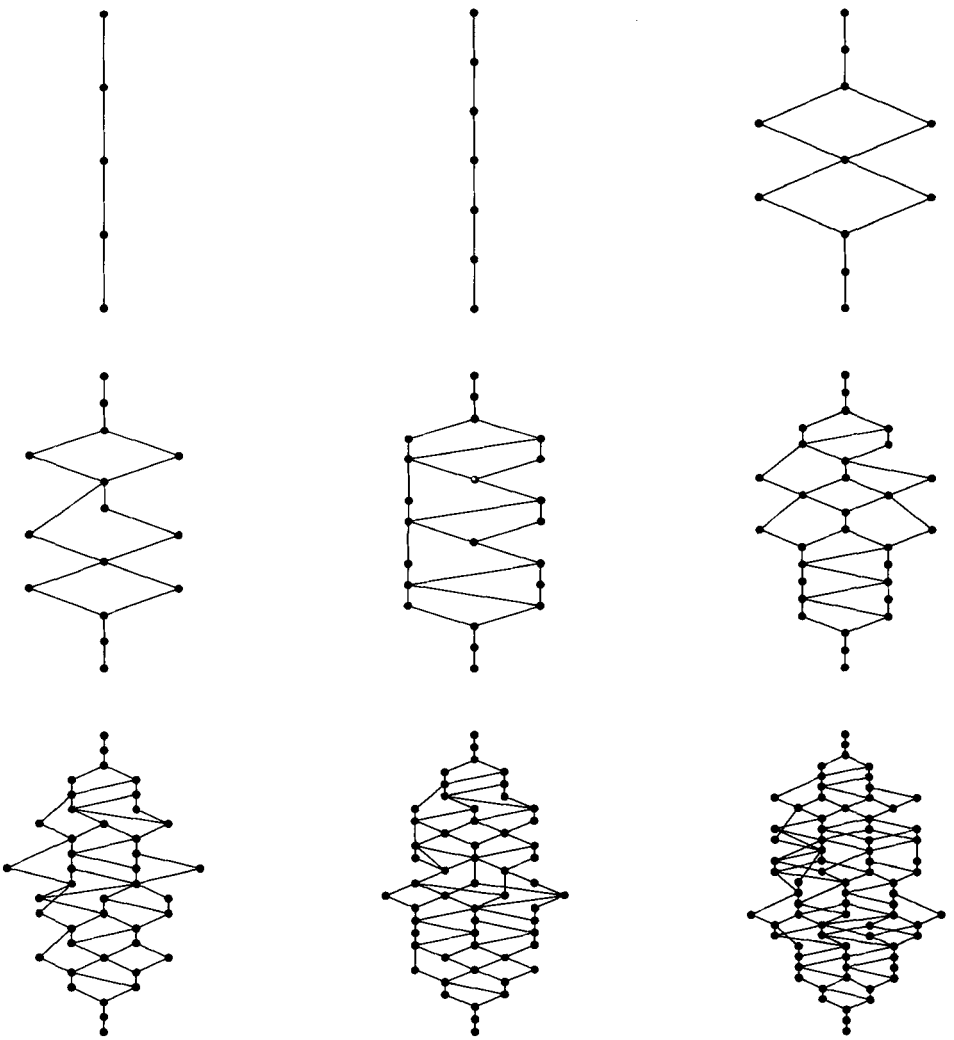
4. Partitions, Compositions, and Young Tableaux



https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

Permutations and subsets are one-dimensional structures, comprising the simple arrangement and selection of objects. This chapter considers more complicated combinatorial objects, some of which are inherently two-dimensional: partitions, compositions, Young tableaux, and set partitions.

A *set partition* is a partition of a set into subsets. A *partition* of an integer n is a set of positive integers that sum up to n. A *composition* of n represents n as the sum of nonnegative integers where order matters. Thus there are only three partitions of six into three parts but 28 compositions of six into three parts. *Young tableaux* are arrangements of n integers whose shape can be described by a partition of n, such that each row and column is sorted.

About the illustration overleaf:

Many interesting structures can be defined on the set of integer partitions of n. Here we display structures called *domination lattices* on integer partitions of n, for $n = 5, 6, \ldots, 13$. Intuitively, we have fine partitions at the bottom of a domination lattice, and these partitions become coarser as we travel upwards. Thus the bottom element in each of the pictures is $(1, 1, \ldots, 1)$ and the topmost element is (n). The command for producing this illustration is

```
ShowGraphArray[Partition[Table[DominationLattice[i], {i, 4, 12}], 3]]
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

4.1 Integer Partitions

Not content with having founded graph theory, Euler also initiated the study of integer partitions. An *integer partition* (in short, *partition*) of a positive integer n is a set of strictly positive integers that sum up to n. The following function `PartitionQ` tests if a given list is an integer partition.

```
PartitionQ[p_List] := (Min[p]>0) && Apply[And, Map[IntegerQ,p]]
PartitionQ[n_Integer, p_List] := (Apply[Plus, p] === n) && (Min[p]>0) && Apply[And, Map[IntegerQ,p]]
```

Identifying an Integer Partition

This loads the *Combinatorica* package.

```
In[1]:= <<DiscreteMath`Combinatorica`
```

$(2,3,4,1)$ is a partition of 10, even if it is not written in the conventional nonincreasing form.

```
In[2]:= PartitionQ[10, {2, 3, 4, 1}]
Out[2]= True
```

In the early twentieth century, the celebrated collaboration between Hardy and Ramanujan led to a remarkable formula for the number of partitions of n. Despite their apparent simplicity, the study of partitions has led to deep connections among many central areas in number theory [AO01]. We have already visited partitions in the context of the types of permutations (Section 3.1.2). We will revisit them in the study of Young tableaux in Section 4.4.

■ 4.1.1 Generating Partitions

The partitions of n can be efficiently constructed using a recursive definition, just like permutations and subsets. This computation is best done by solving a more general problem, that of constructing all partitions of n with largest part at most k. Below we define a function `Partitions` that takes arguments n and k and returns an enumeration of the partitions of n with a maximum element no greater than k. Since the elements of a partition are strictly positive, the largest element can be at most n, so `Partitions[n,n]` gives all the partitions of n.

Any partition of n with largest part at most k either contains a part of size k or it does not. The partitions that do can be constructed by prepending k to all partitions of $n-k$ with largest part at most k. The partitions that do not are all the partitions of n with largest part at most $k-1$. This recursive procedure is implemented in the function below.

```
Partitions[n_Integer] := Partitions[n,n]

Partitions[n_Integer,_] := {} /; (n<0)
Partitions[0,_] := { {} }
Partitions[n_Integer,1] := { Table[1,{n}] }
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

```
Partitions[_,0] := {}

Partitions[n_Integer, maxpart_Integer] :=
      Block[{$RecursionLimit = Infinity},
           Join[Map[(Prepend[#,maxpart])&, Partitions[n-maxpart,maxpart]],
               Partitions[n,maxpart-1]
           ]
      ]
```

Generating all Partitions of n

Here are the 11 partitions of 6. Observe that they are given in reverse lexicographic order.

```
In[3]:= Partitions[6]
Out[3]= {{6}, {5, 1}, {4, 2}, {4, 1, 1}, {3, 3},
   {3, 2, 1}, {3, 1, 1, 1}, {2, 2, 2}, {2, 2, 1, 1},
   {2, 1, 1, 1, 1}, {1, 1, 1, 1, 1, 1}}
```

Most of these partitions do not contain a part bigger than 3.

```
In[4]:= Partitions[6,3]
Out[4]= {{3, 3}, {3, 2, 1}, {3, 1, 1, 1}, {2, 2, 2},
   {2, 2, 1, 1}, {2, 1, 1, 1, 1}, {1, 1, 1, 1, 1, 1}}
```

This recursive procedure for enumerating partitions is a proof for the following identity on $p_{n,k}$, the number of partitions of n with maximum element at most k:

$$p_{n,k} = p_{n-k,k} + p_{n,k-1}, \text{ for } n \geq k > 0.$$

Letting $p_{n,0} = 0$ for all $n > 0$ and $p_{0,k} = 1$ for all $k \geq 0$ gives us a recurrence relation for $p_{n,k}$. The total number of partitions of n, p_n, is equal to $p_{n,n}$, and so this recurrence can be used to compute p_n as well. We will later see how to compute p_n more efficiently using a beautiful formula due to Euler.

```
NumberOfPartitions[n_Integer, k_Integer] := NumberOfPartitions2[n, k] /; ((n >= 0) && (k >= 0))

NumberOfPartitions2[n_Integer?Positive, 0] := 0
NumberOfPartitions2[0, k_Integer] := 1
NumberOfPartitions2[n_Integer?Positive, 1] := 1
NumberOfPartitions2[n_Integer?Positive, k_Integer?Positive] := NumberOfPartitions[n] /; (k >= n)

NumberOfPartitions2[n_Integer, k_Integer] :=
      Block[{$RecursionLimit = Infinity},
           NumberOfPartitions2[n, k] = NumberOfPartitions2[n, k-1] + NumberOfPartitions2[n-k, k]
      ]
```

Calculating the Number of Partitions

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

This is the table of values for $p_{n,k}$ for $n = 1,2,\ldots,8$ and $k \le n$. The entries along the diagonal give the total number of partitions of n. The second column shows that $p_{n,2} = \lfloor n/2 \rfloor + 1$.

```
In[5]:= Table[NumberOfPartitions[i, j], {i, 8}, {j, i}] // TableForm
Out[5]//TableForm= 1
                   1  2
                   1  2  3
                   1  3  4  5
                   1  3  5  6  7
                   1  4  7  9  10 11
                   1  4  8  11 13 14 15
                   1  5  10 15 18 20 21 22
```

The rules for generating the successor to a partition in reverse lexicographic order are fairly straightforward. Let $p = (p_1, p_2, \ldots, p_k)$ be a partition with $p_1 \ge p_2 \ge \cdots \ge p_k$. If the smallest part $p_k > 1$, then peel off one from it, thus increasing the number of parts by one. If not, find the largest j such that part $p_j > 1$ and replace parts $p_j, p_{j+1}, \ldots, p_k$ by

$$\left\lfloor \frac{\sum_{i=j}^{k} p_i}{p_j - 1} \right\rfloor = \left\lfloor \frac{p_j + (k-j)}{(p_j - 1)} \right\rfloor$$

copies of $p_j - 1$, with a last element containing any remainder. The wraparound condition occurs when the partition is all 1's.

```
NextPartition[p_List] := Join[Drop[p,-1],{Last[p]-1,1}]  /; (Last[p] > 1)

NextPartition[p_List] := {Apply[Plus,p]}  /; (Max[p] == 1)

NextPartition[p_List] := NPT[p];

NPT = Compile[{{p, _Integer, 1}},
            Module[{k = Length[p], q = Table[0, {Length[p] + 2}], j, m, r},
                  j = Position[p, 1][[1, 1]] - 1;
                  Do[q[[i]] = p[[i]], {i, j - 1}];
                  m = Quotient[p[[j]] + (k - j), p[[j]] - 1];
                  Do[q[[i]] = p[[j]] - 1, {i, j, j + m - 1}];
                  r = Mod[p[[j]] + (k - j), p[[j]] - 1];
                  q[[j + m]] = r;
                  DeleteCases[q, 0]
            ]
      ]
```

Constructing the Next Partition

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

Calling `NextPartition` repeatedly generates the complete set of partitions in reverse lexicographic order.

```
In[6]:= ( p=Table[1,{6}];
          Table[p=NextPartition[p],{NumberOfPartitions[6]}] )
Out[6]= {{6}, {5, 1}, {4, 2}, {4, 1, 1}, {3, 3},
   {3, 2, 1}, {3, 1, 1, 1}, {2, 2, 2}, {2, 2, 1, 1},
   {2, 1, 1, 1, 1}, {1, 1, 1, 1, 1, 1}}
```

Wilf defined "minimum change" for partitions as decreasing one part by 1 and increasing another part by 1. Here, a "part" of size 0 may increase to 1 and a part of size 1 may decrease to 0. He then posed the question: Is it possible to enumerate partitions in this Gray code order?

This is the binary relation that connects pairs of partitions that can be obtained from each other by incrementing one part and decrementing another.

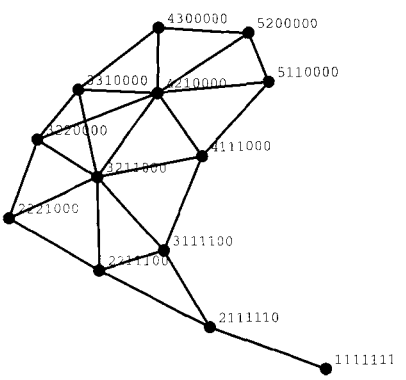
```
In[7]:= pr = Module[{t = #1 - #2}, Count[t, 1] == 1 &&
          Count[t, -1] == 1 && Count[t, 0] == Length[t]-2]&;
```

This relation assumes that the partitions contain the same number of elements, so we pad them with 0's.

```
In[8]:= p = Map[Join[#, Table[0, {7 - Length[#]}]] &, Partitions[7, 5]]
Out[8]= {{5, 2, 0, 0, 0, 0, 0}, {5, 1, 1, 0, 0, 0, 0},
   {4, 3, 0, 0, 0, 0, 0}, {4, 2, 1, 0, 0, 0, 0},
   {4, 1, 1, 1, 0, 0, 0}, {3, 3, 1, 0, 0, 0, 0},
   {3, 2, 2, 0, 0, 0, 0}, {3, 2, 1, 1, 0, 0, 0},
   {3, 1, 1, 1, 1, 0, 0}, {2, 2, 2, 1, 0, 0, 0},
   {2, 2, 1, 1, 1, 0, 0}, {2, 1, 1, 1, 1, 1, 0},
   {1, 1, 1, 1, 1, 1, 1}}
```

Here is the minimum change graph on partitions of 7 with maximum part at most 5. `SpringEmbedding` draws it with exactly one crossing. The graph is not regular at all, with vertex degrees varying from 1 to 7. The presence of a degree-1 vertex implies that the graph is not Hamiltonian. But does it have a Hamiltonian path?

```
In[9]:= ShowGraph[g75 = SpringEmbedding[MakeGraph[p, pr, Type->Undirected,
          VertexLabel -> True]], PlotRange->0.3];
```



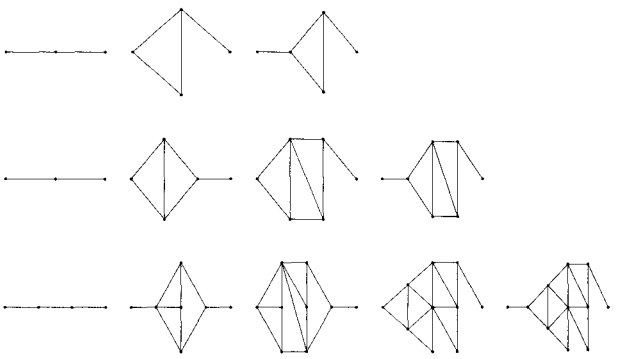
https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

The answer is affirmative, as this computation shows.

```
In[10]:= HamiltonianPath[g75]
Out[10]= {1, 2, 4, 3, 6, 7, 10, 8, 5, 9, 11, 12, 13}
```

These are the minimum change graphs on partitions of n with maximum part at most k, for varying n and k.

```
In[11]:= ShowGraphArray[gs = Table[RankedEmbedding[MakeGraph[
           pnk = Map[Join[#, Table[0, {n - Length[#]}]] &, Partitions[n, k]],
           pr, Type -> Undirected], {1}], {n, 4, 6}, {k, 2, n}]];
```



All of these graphs except the graph for $n = 6$ and $k = 4$ contain a Hamiltonian path from the lexicographically smallest partition to the lexicographically largest partition.

```
In[12]:= Map[(HamiltonianQ[AddEdge[#, {{1, V[#]}}]])&, Flatten[gs,1] ]
Out[12]= {True, True, True, True, True, True, True, True,
    True, False, True, True}
```

This shows that the graph for $n = 6$ and $k = 4$ also has a Hamiltonian path, though not one that starts at the lexicographically smallest and ends at the lexicographically largest partition.

```
In[13]:= HamiltonianPath[gs[[3, 3]]]
Out[13]= {9, 8, 5, 7, 6, 4, 3, 1, 2}
```

Savage [Sav89] showed that for all $n, k \geq 1$, except $(n, k) = (6, 4)$, the partitions of n with maximum part no greater than k can be enumerated in minimum change order starting at the lexicographically smallest partition and ending at the lexicographically largest. Partitions of 6 with maximum part at most 4 can also be enumerated in minimum change order, starting with the partition with all 1's and ending with the partition $(4, 1, 1)$.

■ 4.1.2 Generating Functions and Partitions

Some of the most elegant applications of generating functions in combinatorics appear in the context of partitions. Let $P(z) = \sum_{n=0}^{\infty} p(n)z^n$ be the generating function for the number of partitions of n. Euler noticed that $P(z)$ is equal to the product

$$(1 + z + z^2 + \cdots)(1 + z^2 + z^4 + \cdots)(1 + z^3 + z^6 + \cdots)\cdots,$$

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

where the ith bracket in the above product has the form $(1 + z^i + z^{2i} + z^{3i} + \cdots)$. As an example, let us calculate the coefficient of z^4 in the expansion. The z^4 term can be obtained in exactly five ways:

1. Pick z^4 from bracket 4 and 1 from every other bracket.
2. Pick z^3 from bracket 3, z from bracket 1, and 1 from every other bracket.
3. Pick z^4 from bracket 2 and 1 from every other bracket.
4. Pick z^2 from bracket 2, z^2 from bracket 1, and 1 from every other bracket.
5. Pick z^4 from bracket 1 and 1 from every other bracket.

Thus the coefficient of z^4 in the expansion of the above product is 5.

In general, the number of ways of obtaining z^n is equal to the number of ways of choosing $z^{1\cdot c_1}$ from bracket 1, $z^{2\cdot c_2}$ from bracket 2, $z^{3\cdot c_3}$ from bracket 3, and so on, such that $1\cdot c_1 + 2\cdot c_2 + 3\cdot c_3 + \cdots = n$. In other words, the coefficient of z^n in the expansion of the above product is the number of ways of picking nonnegative integers $c_1, c_2, c_3, \ldots$ such that $1 \cdot c_1 + 2 \cdot c_2 + 3 \cdot c_3 + \cdots = n$. Interpreting c_i as the number of copies of i that are chosen, we see this as $p(n)$, the number of ways of partitioning n. Rewriting the geometric series $(1 + z^i + z^{2i} + z^{3i} + \cdots)$ as $1/(1 - z^i)$, we get that

$$P(z) = \prod_{i=1}^{\infty} \frac{1}{(1 - z^i)}.$$

Mathematica contains considerable machinery for dealing with generating functions. This example uses the function `Series` to compute the first ten terms of the power series expansion of a function.

```
In[14]:= Product[Series[1/(1 - z^i), {z, 0, 10}], {i, 10}]

Out[14]= 1 + z + 2 z^2 + 3 z^3 + 5 z^4 + 7 z^5 + 11 z^6 +

      7      8      9      10      11
  15 z  + 22 z  + 30 z  + 42 z  + O[z]
```

The coefficients of the terms of this polynomial are indeed the number of partitions of n.

```
In[15]:= Table[NumberOfPartitions[n], {n, 0, 10}]
Out[15]= {1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42}
```

How is this generating function for the number of partitions useful? It can be mined for a series of beautiful identities involving partitions. Here we give a slick generating function-based proof that, for any positive integer n, the number of partitions of n with distinct parts equals the number of partitions of n with all odd parts.

The product $(1 + z)(1 + z^2)(1 + z^3)\cdots$ is the generating function for the number of partitions with distinct parts. The product

$$\frac{1}{(1 - z)}\frac{1}{(1 - z^3)}\frac{1}{(1 - z^5)}\cdots$$

is the generating function for the number of partitions with all odd parts. These two expressions are identical because

$$(1 + z)(1 + z^2)(1 + z^3)\cdots = \frac{(1 - z^2)}{(1 - z)}\frac{(1 - z^4)}{(1 - z^2)}\frac{(1 - z^6)}{(1 - z^3)}\cdots = \frac{1}{(1 - z)}\frac{1}{(1 - z^3)}\frac{1}{(1 - z^5)}\cdots.$$

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

Thus the coefficients of the terms are identical, establishing the equality. Here is an example.

This returns the partitions of 11 with distinct parts. There are 12 of these...

```
In[16]:= Select[Partitions[11], (Length[Union[#]] == Length[#] &)]
Out[16]= {{11}, {10, 1}, {9, 2}, {8, 3}, {8, 2, 1}, {7, 4},
    {7, 3, 1}, {6, 5}, {6, 4, 1}, {6, 3, 2}, {5, 4, 2},
    {5, 3, 2, 1}}
```

...and these are the partitions of 11 in which all parts are odd. There are 12 of these as well.

```
In[17]:= Select[Partitions[11], Apply[And,Map[Function[x,OddQ[x]],#]]&]
Out[17]= {{11}, {9, 1, 1}, {7, 3, 1}, {7, 1, 1, 1, 1},
    {5, 5, 1}, {5, 3, 3}, {5, 3, 1, 1, 1},
    {5, 1, 1, 1, 1, 1, 1}, {3, 3, 3, 1, 1},
    {3, 3, 1, 1, 1, 1, 1}, {3, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}}
```

Another important use of $P(z)$ is in deriving the remarkable identity

$$p(n) = p(n-1) + p(n-2) - p(n-5) - p(n-7) + \cdots.$$

Euler noticed that the reciprocal of the partition generating function satisfies the following beautiful identity:

$$\prod_{i=1}^{\infty}(1 - z^i) = 1 - z - z^2 + z^5 + z^7 - z^{12} - \cdots.$$

This is known as *Euler's Pentagonal Theorem*. The integers of the form $k(3k+1)/2$, that is, $0, 1, 2, 5, 7, 12, 15, \ldots$, are called *pentagonal numbers*. Thus Euler's Pentagonal Theorem can be written as

$$\prod_{i=1}^{\infty}(1 - z^i) = \sum_{k=-\infty}^{\infty}(-1)^k z^{(3k^2+k)/2}.$$

Here we expand the product of the first ten terms in Euler's Pentagonal Theorem. The resulting polynomial is correct in the coefficients of z^n for any n, $0 \le n \le 10$, because these terms will not be affected by any of the subsequent multiplications. Some of the remaining terms have coefficients that are not in the set $\{-1, 0, +1\}$, but these will eventually cancel out.

```
In[18]:= Expand[Product[(1 - z^i), {i, 10}]]
                                2    5    7    11    12    13    14
Out[18]= 1 - z - z  + z  + z  + z   - z   - z   - z   -

      15    18    19    20    21    22    25    26
   2 z   + z   + z   + z   + z   + 3 z   - z   - z   -

     27    28    29    30    33    34    35    36
   2 z   - 2 z   - z   - z   + 3 z   + z   + z   + z   +

    37    40    41    42    43    44    48    50    53
   z   - 2 z   - z   - z   - z   + z   + z   + z   - z   -

    54    55
   z   + z
```

Now we start with

$$\left(\prod_{i=1}^{\infty}\frac{1}{(1 - z^i)}\right) \times \left(\prod_{i=1}^{\infty}(1 - z^i)\right) = 1.$$

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

Using the fact that the first product is the generating function of the number of partitions and using Euler's Pentagonal Theorem for the second product, we get

$$\left(\sum_{n=0}^{\infty} p(n)z^n\right) \times (1 - z - z^2 + z^5 + z^7 - z^{12} - \cdots) = 1.$$

The coefficient of z^n on the left of the equation is

$$p(n) - p(n-1) - p(n-2) + p(n-5) + p(n-7) - \cdots,$$

while it is zero on the right. Equating coefficients on the two sides, we get

$$p(n) = p(n-1) + p(n-2) - p(n-5) - p(n-7) + \cdots.$$

In other words,

$$p(n) = \sum_{k=1}^{\infty} (-1)^{(k+1)} \left(p(n - k(3k+1)/2) + p(n + k(3k+1)/2)\right).$$

This is an infinite series, but it has only $O(n^{1/2})$ nonzero terms. Using this to compute $p(n)$ is much more efficient than using the recurrence for $p(n, n)$, so we use this identity to compute $p(n)$ in *Combinatorica*.

```
NumberOfPartitions[n_Integer] := NumberOfPartitions1[n]
NumberOfPartitions1[n_Integer] := 0  /; (n < 0)
NumberOfPartitions1[n_Integer] := 1  /; (n == 0)

NumberOfPartitions1[n_Integer] :=
      Block[{$RecursionLimit = Infinity, k},
           NumberOfPartitions1[n] =
           Sum[(-1)^(k+1) NumberOfPartitions1[n - k (3k-1)/2] +
              (-1)^(k+1) NumberOfPartitions1[n - k (3k+1)/2],
              {k, Ceiling[ (1+Sqrt[1.0 + 24n])/6 ], 1, -1}
           ]
      ]
```

Computing the Number of Integer Partitions

Over 80 years ago, Percy MacMahon computed the values of $p(n)$ for all n up to 200 using this identity. This table of values for $p(n)$ proved immensely useful for Hardy and Ramanujan, who were trying to check the accuracy of their formula for $p(n)$. Since then, much larger tables of $p(n)$ have appeared [GGM58].

```
In[19]:= NumberOfPartitions[200]
Out[19]= 3972999029388
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

We do not know how much time it took MacMahon to calculate $p(200)$, but with *Combinatorica* it takes just a few seconds to calculate $p(1000)$.

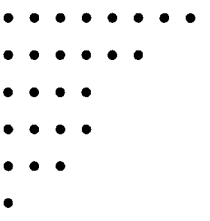
```
In[20]:= Timing[NumberOfPartitions[1000]]
Out[20]= {3.27 Second, 24061467864032622473692149727991}
```

■ 4.1.3 Ferrers Diagrams

Ferrers (or Ferrars [And76]) diagrams represent partitions as patterns of dots. They provide a useful tool for visualizing partitions, because moving the dots around provides a mechanism for proving bijections between classes of partitions. For example, the increment/decrement operation of the minimum change ordering on partitions is equivalent to moving a dot from one row to another. *Combinatorica* contains a function `FerrersDiagram` that produces a Ferrers' diagram.

Here is the Ferrers diagram for the partition $(8,6,4,4,3,1)$. The Ferrers diagram provides an easy way to prove that the number of partitions of n with largest part k is equal to the number of partitions of n with k parts, since transposing one gives an example of the other.

```
In[21]:= FerrersDiagram[{8,6,4,4,3,1}]
```



`TransposePartition` exchanges rows for columns in a partition, thus *transposing* it.

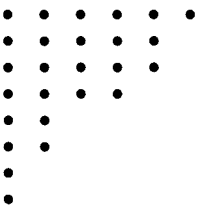
```
TransposePartition[{}] := {}

TransposePartition[p_List] :=
      Module[{s=Select[p,(#>0)&], i, row, r},
            row = Length[s];
            Table [r = row; While [s[[row]]<=i, row--]; r, {i,First[s]}]
      ]
```

Transposing a Partition

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

This Ferrers diagram is a transpose of the previous one. The first row has six dots corresponding to the six dots in the first column of the previous Ferrers diagram. Rex Dwyer observed [Sav89] that if a pair of partitions can be obtained from each other by minimum change (i.e., increment one part and decrement another), then their transposes can also be obtained from each other by minimum change. Since partitions of n with maximum element at most k can be listed in Gray code order, this means that partitions of n with at most k parts can also be listed in Gray code order.

```
In[22]:= FerrersDiagram[TransposePartition[{8,6,4,4,3,1}]];
```



The Ferrers diagram can be generalized by replacing the underlying square grid by other regular patterns in the plane, with interesting results [Pro89].

■ 4.1.4 Random Partitions

A formula for counting the number of different structures of a given type usually leads to an algorithm for constructing random instances of the structure. For each of the possible ways to create the first element of a structure, the enumeration formula can be used to give the number of ways to complete it. Using these counts, we can randomly select the first part in such a way that the complete structures are uniformly distributed.

As we have seen, the number of partitions $p_{n,k}$ of n with largest part at most k is given by the recurrence

$$p_{n,k} = p_{n-k,k} + p_{n,k-1}.$$

The largest part l in a random partition can be found by selecting a random number x such that $1 \le x \le p_{n,n}$, with l determined by $p_{n,l-1} < x \le p_{n,l}$.

Unfortunately, this function requires tabulating values of a two-dimensional function, which can be expensive for large values of n. Nijenhuis and Wilf [NW78] provide the following more efficient algorithm, which randomly selects the magnitude of a part d and its multiplicity j from n, recursing to find a random partition of $n - dj$.

```
RandomPartition[n_Integer?Positive] :=
  Module[{mult = Table[0, {n}], j, d, r=n, z},
    While[ (r > 0),
      d = 1;  j = 0;
      z = Random[] r PartitionsP[r];
      While [z >= 0, j++; If [r-j*d < 0, {j=1; d++;}]; z -= j*PartitionsP[r-j*d]];
      r -= j d; mult[[j]] += d;
    ];
```

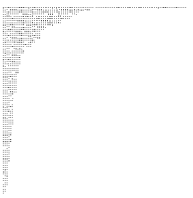
https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

```
  Reverse[Flatten[Table[Table[j, {mult[[j]]}], {j, Length[mult]}]]]
]
```

Constructing Random Partitions

The rectangle defined by the Ferrers diagram of a random partition for any value of $n > 3$ can be expected to contain more empty space than dots, even though exchanging the roles of dots and the empty positions defines another Ferrers diagram.

```
In[23]:= FerrersDiagram[RandomPartition[1000]];
```



Repeating an experiment from [NW78] illustrates that each partition is equally likely to occur with this implementation.

```
In[24]:= Distribution[ Table[RandomPartition[6], {880}] ]
Out[24]= {82, 69, 69, 82, 91, 81, 78, 97, 76, 64, 91}
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

4.2 Compositions

A *composition* of n is a particular arrangement of nonnegative integers that sum up to n. Compositions are perhaps a little less pleasing than partitions, since order matters and zero elements can be included. There is an infinite number of compositions of an integer, unless the number of parts is bounded. Compositions are easier to generate and count than partitions, however.

4.2.1 Random Compositions

The number of compositions of n into k parts equals $\binom{n+k-1}{n}$. This follows from the observation that a composition of n can be represented by inserting $(k-1)$ dividers into a row of n dots such that the numbers of dots between consecutive dividers equals the parts in the composition.

```
NumberOfCompositions[n_,k_] := Binomial[ n+k-1, n ]
```

Counting Compositions

This observation can be used to generate a random composition by using a random $(k-1)$-subset to select the dividers.

```
RandomComposition[n_Integer,k_Integer] :=
      Map[
            (#[[2]] - #[[1]] - 1)&,
            Partition[Join[{0},RandomKSubset[Range[n+k-1],k-1],{n+k}], 2, 1]
      ]
```

Generating Random Compositions

There are 28 compositions of six into three parts.

```
In[25]:= NumberOfCompositions[6,3]
Out[25]= 28
```

The quality of the random compositions reflects the quality of the random k-subsets generator.

```
In[26]:= Distribution[ Table[RandomComposition[6,3], {1000}] ]
Out[26]= {33, 34, 26, 44, 29, 38, 30, 34, 29, 36, 32, 37,
    31, 40, 41, 38, 38, 34, 38, 52, 39, 34, 30, 38, 30, 33,
    46, 36}
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

■ 4.2.2 Generating Compositions

This bijection between compositions and k-subsets also provides a method for listing all the compositions of an integer. By using k-subsets generated in lexicographic order, the resulting compositions will also be lexicographically ordered.

```
Compositions[n_Integer,k_Integer] :=
      Map[
            (Map[(#[[2]]-#[[1]]-1)&, Partition[Join[{0},#,{n+k}],2,1] ])&,
            KSubsets[Range[n+k-1],k-1]
      ]
```

Constructing All Compositions

Compositions of six into three parts are generated in lexicographic order. Going from one composition to the next typically involves incrementing one part and decrementing another. This is not always true, however, as in the transition from $(0,6,0)$ to $(1,0,5)$. Is it possible to enumerate compositions in Gray code order?

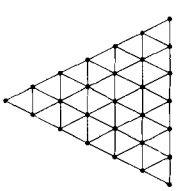
```
In[27]:= Compositions[6,3]
Out[27]= {{0, 0, 6}, {0, 1, 5}, {0, 2, 4}, {0, 3, 3},
    {0, 4, 2}, {0, 5, 1}, {0, 6, 0}, {1, 0, 5}, {1, 1, 4},
    {1, 2, 3}, {1, 3, 2}, {1, 4, 1}, {1, 5, 0}, {2, 0, 4},
    {2, 1, 3}, {2, 2, 2}, {2, 3, 1}, {2, 4, 0}, {3, 0, 3},
    {3, 1, 2}, {3, 2, 1}, {3, 3, 0}, {4, 0, 2}, {4, 1, 1},
    {4, 2, 0}, {5, 0, 1}, {5, 1, 0}, {6, 0, 0}}
```

We define the binary relation that connects pairs of compositions that can be obtained from each other using one increment and one decrement operation.

```
In[28]:= cr = Module[{diff = #1 - #2}, Count[diff, 1] == 1 &&
          Count[diff, -1] == 1 && Count[diff, 0] == Length[diff]-2]&;
```

Remarkably enough, the minimum change graph on compositions is a grid of triangles. This is a good example of how *Combinatorica*'s tools for visualizing graphs can lead to insights about the underlying combinatorial objects. It is easy to trace many distinct Hamiltonian paths in this grid, each of which correspond to a Gray code enumeration of compositions. Klingsberg [Kli82] has shown that, like permutations and combinations, compositions also can be sequenced in a minimum change order.

```
In[29]:= ShowGraph[g = RankedEmbedding[MakeGraph[Compositions[6, 3],
          cr, Type -> Undirected], {1}]];
```



Here is a Hamiltonian cycle that is found fairly quickly.

```
In[30]:= Timing[ HamiltonianCycle[g] ]
Out[30]= {1.33 Second, {1, 2, 3, 4, 5, 6, 7, 13, 12, 11,
     10, 9, 15, 16, 17, 18, 22, 21, 20, 24, 25, 27, 28, 26,
     23, 19, 14, 8, 1}}
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

The next composition can be constructed directly, instead of converting a composition to a k-subset using `NextKSubset` and then back to a composition.

```
NextComposition[l_List] :=
      Append[Table[0,{Length[l]-1}], Apply[Plus, l]] /; First[l]==Apply[Plus,l]

NextComposition[l_List] := NC[l]
NC = Compile[{{l, _Integer, 1}},
          Module[{n = Apply[Plus, l], nl = l, t = Length[l]},
                While[l[[t]] == 0, t--];
                nl[[t-1]]++;
                Do[nl[[i]] = 0, {i, t, Length[l]}];
                nl[[Length[l]]] = Apply[Plus, Take[l, -(Length[l] - t + 1)]] - 1; nl
          ]
    ]
```

Constructing the Next Composition

Here are the same compositions constructed earlier, although in a different order. Specifically, they are the reversals of the lexicographically sequenced compositions.

```
In[31]:= NestList[NextComposition, {0,0,6}, 28]
Out[31]= {{0, 0, 6}, {0, 1, 5}, {0, 2, 4}, {0, 3, 3},
    {0, 4, 2}, {0, 5, 1}, {0, 6, 0}, {1, 0, 5}, {1, 1, 4},
    {1, 2, 3}, {1, 3, 2}, {1, 4, 1}, {1, 5, 0}, {2, 0, 4},
    {2, 1, 3}, {2, 2, 2}, {2, 3, 1}, {2, 4, 0}, {3, 0, 3},
    {3, 1, 2}, {3, 2, 1}, {3, 3, 0}, {4, 0, 2}, {4, 1, 1},
    {4, 2, 0}, {5, 0, 1}, {5, 1, 0}, {6, 0, 0}, {0, 0, 6}}
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

4.3 Set Partitions

A *set partition* is just that, a partition of the elements of a set into subsets. In this section, we study algorithms for enumerating, ranking, unranking, and generating random instances of set partitions. Set partitions and the combinatorial numbers that count them show up in a variety of unexpected places and have been the focus of much research.

Here are the set partitions of $\{a,b,c,d\}$. The order in which subsets are listed in each set partition does not matter. The order in which elements are listed in each subset also does not matter. A set partition with k subsets can be thought of as a distribution of distinguishable objects into k indistinguishable boxes.

```
In[32]:= SetPartitions[{a, b, c, d}]
Out[32]= {{{a, b, c, d}}, {{a}, {b, c, d}},
    {{a, b}, {c, d}}, {{a, c, d}, {b}}, {{a, b, c}, {d}},
    {{a, d}, {b, c}}, {{a, b, d}, {c}}, {{a, c}, {b, d}},
    {{a}, {b}, {c, d}}, {{a}, {b, c}, {d}},
    {{a}, {b, d}, {c}}, {{a, b}, {c}, {d}},
    {{a, c}, {b}, {d}}, {{a, d}, {b}, {c}},
    {{a}, {b}, {c}, {d}}}
```

We define the following canonical way of listing a set partition. Assuming a total order on the input set X, write each subset in increasing order with the subsets themselves arranged in increasing order of their minimum elements.

Here the input is assumed to be a set partition of $\{a,b,c,d,e\}$. The default order on these symbols is used to transform the given set partition into canonical form.

```
In[33]:= ToCanonicalSetPartition[{{b}, {c, e}, {d, a}}]
Out[33]= {{a, d}, {b}, {c, e}}
```

Here the underlying order on the set is explicitly given as $e < d < c < b < a$. This is used to produce a canonical ordering of the set partition that is different from the one constructed above.

```
In[34]:= ToCanonicalSetPartition[{{b}, {c, e}, {d, a}}, {e, d, c, b, a}]
Out[34]= {{e, c}, {d, a}, {b}}
```

Testing whether a given list is a set partition is a fundamental operation.

```
SetPartitionQ[sp_] := (ListQ[sp]) && (Depth[sp] > 2) && SetPartitionQ[sp, Apply[Union, sp]]

SetPartitionQ[sp_, s_List] := (ListQ[sp]) && (Depth[sp] > 2) &&
                        (Apply[And, Map[ListQ, sp]]) && (Sort[Flatten[sp, 1]] === Sort[s])
```

Testing if a List Is a Set Partition

Yes, this is a set partition of $\{a,b,c,d,e,f\}$,...

```
In[35]:= SetPartitionQ[{{a, c}, {d, e}, {f}, {b}}, {a, b, c, d, e, f}]
Out[35]= True
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

...but it is not a set partition of $\{a, b, c, d\}$.

```
In[36]:= SetPartitionQ[{{a, c}, {d, e}, {f}, {b}}, {a, b, c, d}]
Out[36]= False
```

This is not a set partition since c appears in two subsets.

```
In[37]:= SetPartitionQ[{{a, c}, {d, c}, {f}, {b}}, {a, b, c, d, e, f}]
Out[37]= False
```

Since no second argument is given here, the function simply tests to see if the subsets are disjoint.

```
In[38]:= SetPartitionQ[{{a, c}, {1, 3}}]
Out[38]= True
```

■ 4.3.1 Generating Set Partitions

A subset in a set partition is usually called a *part* or a *block*. Set partitions of $\{1, 2, \ldots, n\}$ with exactly k blocks can be generated recursively by noting that every set partition either contains the element 1 by itself as a singleton set or 1 as a block with some other elements. Set partitions of the first type can be generated by constructing all set partitions with $(k-1)$ blocks of $\{2, 3, \ldots, n\}$ and then inserting the set $\{1\}$ into each set partition. Set partitions of the second type can be constructed by generating all set partitions with k blocks of $\{2, 3, \ldots, n\}$ and then inserting the element 1 into each subset in each partition. The function `KSetPartitions` implements this idea.

```
KSetPartitions[{}, 0] := {{}}
KSetPartitions[s_List, 0] := {}
KSetPartitions[s_List, k_Integer] := {} /; (k > Length[s])
KSetPartitions[s_List, k_Integer] := {Map[{#} &, s]} /; (k === Length[s])
KSetPartitions[s_List, k_Integer] :=
      Block[{$RecursionLimit = Infinity},
           Join[Map[Prepend[#, {First[s]}] &, KSetPartitions[Rest[s], k - 1]],
               Flatten[
                  Map[Table[Prepend[Delete[#, j], Prepend[#[[j]], s[[1]]]],
                         {j, Length[#]}
                     ]&,
                     KSetPartitions[Rest[s], k]
                  ], 1
               ]
           ]
      ] /; (k > 0) && (k < Length[s])

KSetPartitions[0, 0] := {{}}
KSetPartitions[0, k_Integer?Positive] := {}
KSetPartitions[n_Integer?Positive, 0] := {}
KSetPartitions[n_Integer?Positive, k_Integer?Positive] := KSetPartitions[Range[n], k]
```

Constructing Set Partitions with k Blocks

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

Here are the seven set partitions of $\{a,b,c,d\}$ with two blocks. Only one has a as a singleton set.

```
In[39]:= KSetPartitions[{a, b, c, d}, 2]
Out[39]= {{{a}, {b, c, d}}, {{a, b}, {c, d}},
    {{a, c, d}, {b}}, {{a, b, c}, {d}}, {{a, d}, {b, c}},
    {{a, b, d}, {c}}, {{a, c}, {b, d}}}
```

The number of set partitions of n elements into two blocks is $2^{n-1}-1$. Why? Because the block with element 1 can contain any *proper* (noncomplete) subset of the other $n-1$ elements.

```
In[40]:= Map[Length, Table[ KSetPartitions[i,2], {i,1,10}] ]
Out[40]= {0, 1, 3, 7, 15, 31, 63, 127, 255, 511}
```

This is the set of vertex labels for a graph whose vertices are 2-block set partitions of $\{1,2,3,4,5\}$. Since there are only two blocks, specifying one block completely specifies the entire set partition. This is the shortcut we take here.

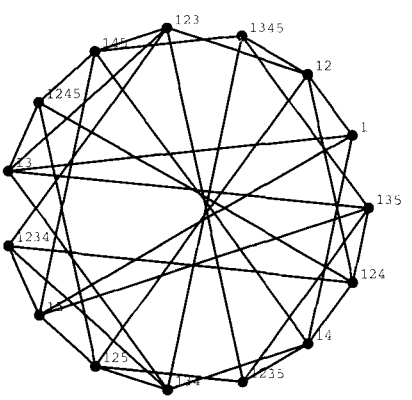
```
In[41]:= 152 = Map[StringJoin[Map[ToString, #[[1]]]] &, sp52 =
          KSetPartitions[5, 2]]
Out[41]= {1, 12, 1345, 123, 145, 1245, 13, 1234, 15, 125,
    134, 1235, 14, 124, 135}
```

Here we build the relation that connects pairs of 2-block set partitions of $\{1,2,3,4,5\}$ that can be obtained by moving an element from one block to another.

```
In[42]:= spr = (MemberQ[{1, 4}, Sum[Abs[Position[#1, i][[1, 1]] -
          Position[#2, i][[1, 1]]], {i, 5}]]) &;
```

The vertices of this graph are 2-block set partitions of $\{1,2,3,4,5\}$. The vertices are labeled with the block containing 1. A pair of set partitions (x,y) is connected by an edge if y can be obtained from x by moving an element to a different block. For example, the four neighbors of $\{\{1\},\{2,3,4,5\}\}$ are obtained by deleting an element from the second subset and inserting it into the first. This operation corresponds to a "minimum change" in going from one 2-block set partition to another.

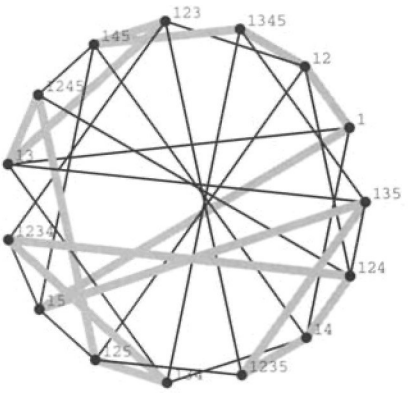
```
In[43]:= ShowGraph[g=SetVertexLabels[MakeGraph[KSetPartitions[5, 2], spr,
          Type->Undirected], 152], PlotRange->0.09];
```



https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

This Hamiltonian cycle tells us that 2-block set partitions of $\{1,2,3,4,5\}$ can be listed in Gray code order. However, `KSetPartitions` does not do so, instead listing set partitions in the order in which they appear on the circle above. So each "missing" edge between a pair of consecutive vertices on the circle corresponds to a set partition from which the next set partition was not obtained minimally. In Section 4.3.4, we explore further the problem of enumerating set partitions in Gray code order.

```
In[44]:= ShowGraph[Highlight[g, {Partition[HamiltonianCycle[g],2,1]},
                   HighlightedEdgeColors -> {Gray}], PlotRange -> 0.1];
```



The set of all set partitions is generated by constructing those with k blocks, for every $1 \le k \le n$.

```
SetPartitions[{}] := {{}}
SetPartitions[s_List] := Flatten[Table[KSetPartitions[s, i], {i, Length[s]}], 1]

SetPartitions[0] := {{}}
SetPartitions[n_Integer?Positive] := SetPartitions[Range[n]]
```

Constructing All Set Partitions

This is the set of all set partitions of the set $\{a,b,c,d\}$. Each set partition is in canonical order.

```
In[45]:= SetPartitions[{a, b, c, d}]
Out[45]= {{{a, b, c, d}}, {{a}, {b, c, d}},
    {{a, b}, {c, d}}, {{a, c, d}, {b}}, {{a, b, c}, {d}},
    {{a, d}, {b, c}}, {{a, b, d}, {c}}, {{a, c}, {b, d}},
    {{a}, {b}, {c, d}}, {{a}, {b, c}, {d}},
    {{a}, {b, d}, {c}}, {{a, b}, {c}, {d}},
    {{a, c}, {b}, {d}}, {{a, d}, {b}, {c}},
    {{a}, {b}, {c}, {d}}}
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

■ 4.3.2 Stirling and Bell Numbers

The number of set partitions of $\{1,2,\ldots,n\}$ having k blocks is a fundamental combinatorial number called the *Stirling number of the second kind*. In Section 3.1.4, we encountered the other kind of combinatorial numbers that James Stirling lent his name to. We use $\left\{ {n \atop k} \right\}$ to denote the number of set partitions of $\{1,2,\ldots,n\}$ having k blocks. Our notation of $\binom{n}{k}$, $\left[{n \atop k} \right]$, and $\left\{ {n \atop k} \right\}$ for binomial numbers, Stirling numbers of the first kind, and Stirling numbers of the second kind, respectively, follows Knuth's plea in favor of this notation [Knu92].

The recursive procedure mentioned in the previous section for generating set partitions of n elements with k blocks gives a proof of the following "Pascal triangle"–like identity:

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

for all integers $k, n > 0$, where $\left\{ {n \atop 0} \right\} = \left\{ {0 \atop k} \right\} = 0$ for integers $n > 0$ and $k > 0$ and $\left\{ {0 \atop 0} \right\} = 1$.

This recurrence can be verified for the entries in this table showing $\left\{ {n \atop k} \right\}$ for all $n = 1,2,\ldots,10$ and relevant k. Several patterns are evident from this table. For example, the sequence of next-to-last elements by row, $1,3,6,10,15,\ldots$, is the familiar "n choose 2" sequence, suggesting that $\left\{ {n \atop n-1} \right\} = \binom{n}{2}$.

```
In[46]:= Table[StirlingSecond[n, k], {n, 10}, {k, 1, n}] // ColumnForm
Out[46]=

{1}
{1, 1}
{1, 3, 1}
{1, 7, 6, 1}
{1, 15, 25, 10, 1}
{1, 31, 90, 65, 15, 1}
{1, 63, 301, 350, 140, 21, 1}
{1, 127, 966, 1701, 1050, 266, 28, 1}
{1, 255, 3025, 7770, 6951, 2646, 462, 36, 1}
{1, 511, 9330, 34105, 42525, 22827, 5880, 750, 45, 1}
```

Rather than compute Stirling numbers of the second kind using the above recurrence, we use the following beautiful identity that relates these Stirling numbers to binomial coefficients:

$$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{i=1}^{k} (-1)^{(k-i)} \binom{k}{i} i^n.$$

```
StirlingSecond[n_Integer,0] := If [n == 0, 1, 0]
StirlingSecond[0,k_Integer] := If [k == 0, 1, 0]

StirlingSecond[n_Integer?Positive, k_Integer?Positive] :=
       Sum [ (-1)^(k-i)*Binomial [k, i]*(i^n), {i, 1, k}]/k!
```

Computing Stirling Numbers of the Second Kind

There are many ways of partitioning a 100-element set into blocks of 50, in fact, more than 10^{100} of them!

```
In[47]:= StirlingSecond[100, 50]
Out[47]= 43098323700936634042151430154725869594352028961434\
    06139124417411312803190588537831455982616599920139 00
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

Here is a timing comparison of the two ways of computing Stirling numbers of the second kind. Using the recurrence to compute $\left\{ {200 \atop 100} \right\}$ is much slower than the sum involving binomial numbers.

```
In[48]:= <<extraCode/RecursiveStirling2;
         {Timing[RecursiveStirling2[200, 100];], Timing[ Stirling2[200,
         100];]}
Out[49]= {{6.51 Second, Null}, {0. Second, Null}}
```

Mathematica has a built-in function StirlingS2 to compute the Stirling numbers of the second kind. Surprisingly, the built-in function seems slower than our implementation.

```
In[50]:= Table[ Timing[StirlingS2[100 i, 50 i];]/
         Timing[StirlingSecond[100 i, 50 i];], {i, 2, 5}]
Out[50]= {{2.66667, 1}, {2.8, 1}, {2.88889, 1},
    {2.92857, 1}}
```

The same test, run again, tells a different story! This time the *Mathematica* built-in function is consistently faster. The cause of this unreliable behavior is the dynamic programming *Mathematica* uses to implement the recurrence for Stirling numbers of the second kind. Results from the earlier timing example are still around in memory and available for quick lookup.

```
In[51]:= Table[ Timing[StirlingS2[100 i, 50 i];]/
         Timing[StirlingSecond[100 i, 50 i];], {i, 2, 5}]
Out[51]= {{0., 1}, {0.4, 1}, {0.555556, 1}, {0.714286, 1}}
```

The total number of set partitions of $\{1, 2, \ldots, n\}$ is the nth *Bell number*, denoted B_n, another fundamental combinatorial number. Bell numbers are named after E. T. Bell, who is better known for his riveting accounts of the lives of famous mathematicians in *Men of Mathematics* [Bel86] than for his contributions to combinatorics. Clearly, $B_n = \sum_{k=1}^{n} \left\{ {n \atop k} \right\}$, and hence this formula can be used to compute them. However, an alternative recurrence provides a faster way of computing the Bell numbers. A simple bijection establishes the identity. A set partition of $\{1, 2, \ldots, n\}$ can be constructed by choosing the subset containing 1 first and then prepending the chosen subset to set partitions of the rest of the elements. $\binom{n-1}{k}$ is the number of ways of choosing a $(k+1)$-subset containing 1, and B_{n-k-1} is the number of set partitions of the rest of the elements. Summed over all possible k and simplified using the symmetry of binomial numbers, we get:

$$B_n = \sum_{k=0}^{n-1} \binom{n-1}{k} B_{n-(k+1)} = \sum_{k=0}^{n-1} \binom{n-1}{k} B_k.$$

```
BellB[n_Integer] := BellB1[n]
BellB1[0] := 1
BellB1[n_]:= Block[{$RecursionLimit = Infinity}, BellB1[n] = Sum [Binomial[n-1, k]*BellB1[k], {k, 0, n-1}]]
```

Computing the Bell Number

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

B_{300} can be computed about twice as fast by using the recurrence relation instead of the sum of Stirling numbers.

```
In[52]:= {Timing[ N[BellB[300]] ],
          Timing[N[ Sum[StirlingSecond[300, k], {k, 300}] ]]}

Out[52]= {{10.67 Second, 9.59371716083927 10^453 },
    {17.49 Second, 9.59371716083927 10^453 }}
```

Dynamic programming has been used to speed up the computation of the Bell numbers. The biggest payoff comes when Bell numbers from old computations are still around in memory. This is why B_{310} takes virtually no time after B_{300} has been computed.

```
In[53]:= Timing[N[ BellB[310] ]]

Out[53]= {0.96 Second, 3.315493362145864 10^472 }
```

■ 4.3.3 Ranking, Unranking, and Random Set Partitions

We now turn to the problems of ranking and unranking set partitions. The function `SetPartitions` lists set partitions with one block first, followed by set partitions with two blocks, and so on, ending with the set partition in which every block is a singleton. Thus the problem of ranking and unranking set partitions (as enumerated by the function `SetPartitions`) reduces to the problem of ranking and unranking set partitions with a fixed number of blocks.

Suppose we want to compute the rank of a set partition S of a set X whose first element is x. Let S_x denote the block in S containing x. The function `KSetPartitions` first enumerates all set partitions of X in which x appears in a singleton set. These set partitions are enumerated according to the order in which the $(k-1)$-block set partitions of $X - \{x\}$ are enumerated. This means that if $S_x = \{x\}$, then the rank of S equals the rank of $S - \{x\}$ in the enumeration of set partitions of $X - \{x\}$. Following the set partitions in which x appears by itself, `KSetPartitions` enumerates set partitions in which x appears in blocks of size 2 or more. To construct these set partitions, the function `KSetPartitions` first enumerates the k-block set partitions of $X - \{x\}$. Let this sequence be $S_1, S_2, S_3, \ldots$. The desired sequence is constructed by inserting x into each of the k blocks of S_1 in turn – this gives rise to k set partitions – followed by inserting x into each of the k blocks of S_2, and so on. From this recursive construction of set partitions, the following recurrence for the rank of set partitions follows. In the following we use X' for $X - \{x\}$; S' for the set partition of X', in canonical form, obtained by removing x from S; and j for the position in S' of the subset $S_x - \{x\}$:

$$\text{Rank}\,(S, X) = \begin{cases} \text{Rank}\,(S', X') & \text{if } S_x = \{x\} \\ \left\{ {n-1 \atop k-1} \right\} + k\,\text{Rank}\,(S', X') + (j-1) & \text{otherwise.} \end{cases}$$

Since `SetPartitions` lists partitions in increasing order of number of blocks, the rank of S in the list of all set partitions of X (not just the ones with k blocks) is $\sum_{i=1}^{k-1} \left\{ {n \atop i} \right\}$ plus the rank of S in the list of k-block set partitions. In the following we present code for `RankKSetPartition` and `RankSetPartition`, functions for ranking set partitions within k-block set partitions and within all set partitions, respectively.

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

```
RankSetPartition [sp_?SetPartitionQ] :=
      Module[{n = Length[s = Sort[Flatten[sp, 1]]], k = Length[sp]},
            Sum[StirlingSecond[n, i], {i, 1, k-1}] + RankKSetPartition [sp, s]
      ]

RankSetPartition [sp_List, s_List] :=
      Module[{n = Length[s], k = Length[sp]},
            Sum[StirlingSecond[n, i], {i, 1, k-1}] + RankKSetPartition [sp, s]
      ] /; SetPartitionQ[sp, s]

RankKSetPartition[sp_?SetPartitionQ] :=
       Module[{s = Sort[Flatten[sp, 1]]},
             RankKSetPartition1[ToCanonicalSetPartition[sp, s], s]
       ]

RankKSetPartition[sp_List, s_List] :=
       RankKSetPartition1[ToCanonicalSetPartition[sp, s], s] /; SetPartitionQ[sp, s]
```

Ranking a Set Partition

The set partitions produced by SetPartitions are ranked. As expected, this produces the natural numbers.

```
In[54]:= Map[RankSetPartition, SetPartitions[{a, b, c, d}]]
Out[54]= {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
```

The rank of sp in the list of all eight-element set partitions is larger than its rank in the list of such set partitions with exactly four blocks.

```
In[55]:= sp = {{a, f}, {b}, {g, c, d}, {e, h}};
         {RankKSetPartition[sp], RankSetPartition[sp]}
Out[56]= {616, 1710}
```

The difference between the two ranks in the above example is exactly the number of set partitions of $\{a,b,c,d,e,f,g,h\}$ with one, two, or three blocks.

```
In[57]:= {%[[2]]-%[[1]], Sum[StirlingSecond[8,i], {i,1,3}]}
Out[57]= {1094, 1094}
```

We also provide the inverse functions UnrankKSetPartition and UnrankSetPartition. Since UnrankSetPartition can be easily derived from UnrankKSetPartition, we focus on the latter first. This can be constructed from the recurrence relation for the rank of a k-block set partition. Suppose we are given a set X with n elements; an integer k, $1 \le k \le n$; and an integer r, $0 \le r < \left\{ {n \atop k} \right\}$ and are asked to find the k-block set partition of X with rank r in the list of k-block set partitions of X generated by KSetPartitions. The recurrence for set partition rank tells us there are two cases. If $r < \left\{ {n-1 \atop k-1} \right\}$, then the given set partition contains the first element of X as a singleton, so we can recursively unrank r to get a $(k-1)$-block set partition of the rest of X. If $r \ge \left\{ {n-1 \atop k-1} \right\}$, then we know that the first element of X occurs in a set of size 2 or more. The position j of this set is the remainder when $r - \left\{ {n-1 \atop k-1} \right\}$ is divided by k, and the quotient from this division is the rank of the rest of the set

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

partition. The algorithm continues recursively. We find the "position" j of this set, recursively find the rest of set partition, and insert the first element of X into the jth set.

```
UnrankKSetPartition[r_Integer, {}, 0] := {}

UnrankKSetPartition[0, set_List, k_Integer?Positive] :=
      Append[Table[{set[[i]]}, {i, 1, k-1}],
            Take[set, -(Length[set]-k+1)]
      ] /; (k <= Length[set])

UnrankKSetPartition[r_Integer?Positive, set_List, k_Integer?Positive] :=
      Block[{n = Length[set], t, j, $RecursionLimit = Infinity},
            If[r < StirlingSecond[n-1, k-1],
              Prepend[UnrankKSetPartition[r, Rest[set], k-1],
                    {First[set]}
              ],
              t = r - StirlingSecond[n-1, k-1];
              j = 1 + Mod[t, k];
              tempSP = UnrankKSetPartition[Quotient[t, k], Rest[set], k];
              Prepend[Delete[tempSP, j], Prepend[tempSP[[j]], First[set]]]
            ]
      ] /; (k <= Length[set])

UnrankKSetPartition[r_Integer, n_Integer, k_Integer] :=
      UnrankKSetPartition[r, Range[n], k] /; (k <= n) && (k >= 0)
```

Unranking a Set Partition with k Blocks

This gives the ten-block set partition of $\{1, 2, \ldots, 20\}$ that has 100,076 set partitions before it in the list of ten-block set partitions generated by `KSetPartitions`.

```
In[58]:= UnrankKSetPartition[100076, Range[20], 10]
Out[58]= {{1}, {2}, {3}, {4}, {5}, {6}, {7},
    {8, 11, 14, 15, 17}, {9, 10, 18, 19, 20}, {12, 13, 16}}
```

As required, the rank and unrank functions are inverses of each other.

```
In[59]:= RankKSetPartition[%, Range[20]]
Out[59]= 100076
```

The rank of a set partition with k blocks in the list of all set partitions is $\sum_{i=1}^{k-1} \left\{ {n \atop i} \right\}$ plus the rank of the set partition in the list of all set partitions with k blocks. Given just the rank r in the list of all set partitions, we first need to find k, the number of blocks in the set partition. This is simply the largest k such that $\sum_{i=1}^{k-1} \left\{ {n \atop i} \right\} \leq r$. Once k is found, then $r - \sum_{i=1}^{k-1} \left\{ {n \atop i} \right\}$ is unranked using `UnrankKSetPartition`.

```
UnrankSetPartition[0, set_List] := {set}

UnrankSetPartition[r_Integer?Positive, set_List] :=
      Block[{n = Length[set], k = 0, sum = 0, $RecursionLimit = Infinity},
            While[sum <= r, k++; sum = sum + StirlingSecond[n, k]];
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

```
        UnrankKSetPartition[r - (sum - StirlingSecond[n, k]), set, k]
    ] /; (r < BellB[ Length[set] ])

UnrankSetPartition[0, 0] = {{}}
UnrankSetPartition[r_Integer, n_Integer?Positive] := UnrankSetPartition[r, Range[n]] /; (r >= 0)
```

Unranking a Set Partition

This gives the set partition of $\{1,2,\ldots,20\}$ that has 100,076 set partitions before it in the list of all set partitions generated by the function `SetPartitions`.

```
In[60]:= UnrankSetPartition[100076, Range[20]]
Out[60]= {{1, 2, 3, 5, 6, 8, 11, 17},
    {4, 7, 9, 10, 12, 13, 14, 15, 16, 18, 19, 20}}
```

Generating a random set partition is easy using `UnrankKSetPartition`, so we provide functions `RandomKSetPartitions` and `RandomSetPartitions` to do this.

```
RandomKSetPartition [{}, 0] := {}
RandomKSetPartition [set_List, k_Integer?Positive] :=
      UnrankKSetPartition [
        Random[Integer, {0, StirlingSecond[Length[set], k]-1}], set, k
      ] /; ((Length[set] > 0) && (k <= Length[set]))

RandomKSetPartition [0, 0] := {}
RandomKSetPartition [n_Integer?Positive, k_Integer?Positive] := RandomKSetPartition [Range[n], k] /; (k <= n)
RandomSetPartition[{}] := {}
RandomSetPartition [set_List] :=
      UnrankSetPartition [Random[Integer, {0, BellB[Length[set]]-1}], set] /; (Length[set] > 0)

RandomSetPartition [n_Integer] := RandomSetPartition [ Range[n] ]
```

Generating a Random Set Partition

A randomly chosen set partition of $\{1,2,\ldots,10\}$ and a randomly chosen set partition of $\{1,2,\ldots,10\}$ having four blocks.

```
In[61]:= {RandomSetPartition[10],
          RandomKSetPartition[10, 4]} // ColumnForm
Out[61]= {{1, 3, 6, 8, 10}, {2, 4, 7}, {5}, {9}}
         {{1, 2, 6}, {3, 5, 8, 9}, {4}, {7, 10}}
```

This shows the distribution of the number of blocks in 5000 randomly chosen set partitions of $\{1,2,\ldots,10\}$. The distribution increases first and then decreases.

```
In[62]:= Distribution[Map[Length, Table[RandomSetPartition[10], {5000}]]]
Out[62]= {25, 427, 1466, 1823, 958, 265, 35, 1}
```

This, modulo the rare terms, reflects the actual distribution of $\left\{ {n \atop k} \right\}$.

```
In[63]:= Table[StirlingSecond[10,i], {i,1,10}]
Out[63]= {1, 511, 9330, 34105, 42525, 22827, 5880, 750, 45,
    1}
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

■ 4.3.4 Set Partitions and Restricted Growth Functions

There is a neat bijection between set partitions and certain structures called restricted growth functions. A *restricted growth function* (in short, RGF) is a function f with domain and range equal to $\{1,2,\ldots,n\}$ satisfying the conditions

$$f(1) = 1 \qquad \text{and} \qquad f(i) \leq \max_{1 \leq j < i}\{f(j)\} + 1 \text{ for } 1 < i \leq n.$$

The latter condition "restricts" the growth of the function by requiring that $f(i)$ be at most one more than the maximum of the previous function values. We will usually write an RGF f as the sequence $(f(1),f(2),\ldots,f(n))$.

These are the RGFs on $\{1,2,3,4\}$ listed in lexicographic order. The lexicographic successor of $(1,1,1,2)$ is not $(1,1,1,3)$ because the latter sequence is not an RGF. For the last element to have value 3, at least one of the first three elements has to have value 2.

```
In[64]:= RGFs[4]
Out[64]= {{1, 1, 1, 1}, {1, 1, 1, 2}, {1, 1, 2, 1},
    {1, 1, 2, 2}, {1, 1, 2, 3}, {1, 2, 1, 1}, {1, 2, 1, 2},
    {1, 2, 1, 3}, {1, 2, 2, 1}, {1, 2, 2, 2}, {1, 2, 2, 3},
    {1, 2, 3, 1}, {1, 2, 3, 2}, {1, 2, 3, 3}, {1, 2, 3, 4}}
```

To see the bijection, take a set partition S of $\{1,2,\ldots,n\}$ in canonical form. We construct a function f on $\{1,2,\ldots,n\}$ by setting $f(i) = j$ if element i is in block j in S. This implies that $f(1) = 1$ because element 1 appears in block 1 in a set partition written in canonical form. Furthermore, if elements 1 through $(i-1)$ occur in blocks 1 through $(j-1)$, then element i can occur only in blocks 1 through j. This corresponds to the constraint that restricts the growth of the function. One can start with an RGF f on $\{1,2,\ldots,n\}$ and construct a set partition of $\{1,2,\ldots,n\}$ by throwing element i in block j, if $f(i) = j$. It is easy to check that the set partition thus generated is in canonical form.

It is sometimes easier to think about the RGFs than the corresponding set partitions because RGFs have a linear structure. `SetPartitionToRGF` and `RGFToSetPartition` implement the bijection described above.

Generate the set partition corresponding to the RGF $(1,2,1,1,3)$ in canonical form.

```
In[65]:= RGFToSetPartition[{1, 2, 1, 1, 3}]
Out[65]= {{1, 3, 4}, {2}, {5}}
```

Inverting the above transformation gets us the original RGF back.

```
In[66]:= SetPartitionToRGF[%]
Out[66]= {1, 2, 1, 1, 3}
```

The number of blocks in the set partition equals the maximum value that the corresponding function takes.

```
In[67]:= {Length[p=RandomSetPartition[100]], Max[SetPartitionToRGF[p]]}
Out[67]= {26, 26}
```

This is the set of all 4-element RGFs constructed by generating the set partitions of $\{1,2,3,4\}$ and transforming these into RGFs. The resulting list is not in lexicographic order.

```
In[68]:= Map[SetPartitionToRGF, SetPartitions[4]]
Out[68]= {{1, 1, 1, 1}, {1, 2, 2, 2}, {1, 1, 2, 2},
    {1, 2, 1, 1}, {1, 1, 1, 2}, {1, 2, 2, 1}, {1, 1, 2, 1},
    {1, 2, 1, 2}, {1, 2, 3, 3}, {1, 2, 2, 3}, {1, 2, 3, 2},
    {1, 1, 2, 3}, {1, 2, 1, 3}, {1, 2, 3, 1}, {1, 2, 3, 4}}
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

Now we turn to the problem of generating RGFs. It turns out to be fairly easy to generate RGFs in lexicographic order. For an RGF f not equal to $(1,2,3,\ldots,n)$, let i be the largest value such that $f(i) < \max_{1\le j<i}\{f(j)\}+1$. In other words, i is the largest value such that $f(i)$ can be incremented without violating the RGF property. The lexicographic successor of f is the function obtained by incrementing $f(i)$ and setting $f(j) = 1$ for all larger j.

This is the list of RGFs on $\{1,2,3,4\}$ in lexicographic order.

```
In[69]:= RGFs[4]
Out[69]= {{1, 1, 1, 1}, {1, 1, 1, 2}, {1, 1, 2, 1},
    {1, 1, 2, 2}, {1, 1, 2, 3}, {1, 2, 1, 1}, {1, 2, 1, 2},
    {1, 2, 1, 3}, {1, 2, 2, 1}, {1, 2, 2, 2}, {1, 2, 2, 3},
    {1, 2, 3, 1}, {1, 2, 3, 2}, {1, 2, 3, 3}, {1, 2, 3, 4}}
```

These RGFs are now turned into set partitions. Going from an RGF to its lexicographic successor is equivalent to the following operation on set partitions. Let x be the maximum element in a nonsingleton block in the set partition. Move all elements larger than x into the first block and bump up x into the next block. Viewed in terms of insertions and deletions on set partitions, this is hardly a minimum change operation.

```
In[70]:= Map[RGFToSetPartition, %]
Out[70]= {{{1, 2, 3, 4}}, {{1, 2, 3}, {4}},
    {{1, 2, 4}, {3}}, {{1, 2}, {3, 4}}, {{1, 2}, {3}, {4}},
    {{1, 3, 4}, {2}}, {{1, 3}, {2, 4}}, {{1, 3}, {2}, {4}},
    {{1, 4}, {2, 3}}, {{1}, {2, 3, 4}}, {{1}, {2, 3}, {4}},
    {{1, 4}, {2}, {3}}, {{1}, {2, 4}, {3}},
    {{1}, {2}, {3, 4}}, {{1}, {2}, {3}, {4}}}
```

Generating RGFs is faster than generating set partitions.

```
In[71]:= Table[Timing[SetPartitions[i];]/Timing[RGFs[i];], {i, 7, 9}]
Out[71]= {{4., 1}, {3., 1}, {2.61458, 1}}
```

However, using RGFs to generate set partitions is slower than generating them directly because of the conversion cost.

```
In[72]:= Table[ Timing[Map[RGFToSetPartition, RGFs[i]];]/
          Timing[SetPartitions[i];], {i, 7, 9}]
Out[72]= {{13.3529, 1}, {19.5833, 1}, {24.5698, 1}}
```

RGFs provide an encoding for set partitions that makes it easy to ask Gray code enumeration questions. For example, is there an enumeration of RGFs such that each RGF differs from the previous in exactly one position and in that position by only 1? This "minimum change" corresponds to deleting one element from a block and inserting it into an adjacent block. We now pose this as a question of determining whether an appropriate graph has a Hamiltonian cycle.

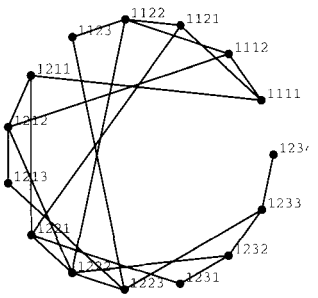
This is a binary relation on RGFs that connects pairs of RGFs that differ in exactly one position and in that position by exactly 1.

```
In[73]:= rgfr = (Count[#1-#2, 0] == Length[#1]-1) &&
                (Count[Abs[#1 - #2], 1] == 1)&;
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

This is the "minimum change" graph on RGFs constructed using the binary relation defined above. The graph contains a degree-1 vertex, implying that it has no Hamiltonian cycle; however, it may still have a Hamiltonian path, one of whose endpoints is necessarily the degree-1 vertex (1,2,3,4).

```
In[74]:= g = MakeGraph[RGFs[4], rgfr, Type->Undirected, VertexLabel->True];
         ShowGraph[g, PlotRange -> 0.15];
```



Here we see that this graph on the RGFs has no Hamiltonian path.

```
In[76]:= HamiltonianPath[g]
Out[76]= {}
```

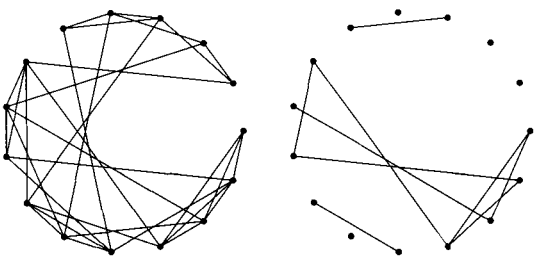
In fact, Ehrlich [Ehr73] shows that for infinitely many values of n, RGFs on $\{1,2,\ldots,n\}$ do not have a Gray code of this sort. The notion of minimum change can be relaxed a little, allowing for pairs of RGFs that differ in exactly one position, by at most 2, to be connected.

This relaxes the definition of "minimum change" a little.

```
In[77]:= rgfr = (Count[#1-#2, 0] == Length[#1] - 1) &&
                (Count[Abs[#1 - #2], 1] <= 2)&;
```

The resulting graph is a *supergraph* of the one constructed above. The added edges are revealed in the *difference* of the two graphs. The old graph did not have a Hamiltonian path, but the new graph...

```
In[78]:= g1 = MakeGraph[RGFs[4], rgfr, Type -> Undirected];
         ShowGraphArray[{g1, GraphDifference[g1,g]}];
```



...has more than 700 Hamiltonian cycles.

```
In[80]:= Length[HamiltonianCycle[g1, All]]
Out[80]= 748
```

Savage [Sav97] mentions that it is indeed possible to list RGFs in this relaxed Gray code order, even with some additional restrictions.

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

4.4 Young Tableaux

Many combinatorial problems are properly interpreted as two-dimensional structures. For example, suppose we are interested in counting the number of ways to arrange n left parentheses and n right parentheses so that they make balanced formulas. For $n = 3$, there are five distinct formulas: ((())), ()(()), (())(), (()()), and ()()(). Now consider a $2 \times n$ matrix containing the numbers $1, \ldots, 2n$, such that each row and column is in increasing order. There is a bijection between the balanced formulas and these matrices, by interpreting the $(1, i)$th entry in the matrix to be the position of the ith left parenthesis, and the $(2, j)$th entry to be the position of the jth right parenthesis. Since the columns are ordered, the left parenthesis always comes before the corresponding right parenthesis. This example illustrates the importance of ordered structures of ordered lists.

A *Young tableau of shape* $(n_1, n_2, \ldots, n_m)$, where $n_1 \geq n_2 \geq \cdots \geq n_m > 0$, is an arrangement of $n_1 + n_2 + \cdots + n_m$ distinct integers in an array of m rows with n_i elements in row i such that entries in each row and in each column are in increasing order. Below we provide a function `TableauQ` that tests if a given two-dimensional structure is a tableau. Even though our definition says that a tableau should contain distinct integers, `TableauQ` allows repetitions. Even though all the tableaux we will consider here will contain distinct integers, in other contexts it is useful to allow repetition [Knu70].

```
TableauQ[{}] = True
TableauQ[t_List] :=
      And [
            Apply[And, Map[(Apply[LessEqual,#])&, t] ],
            Apply[And, Map[(Apply[LessEqual,#])&, TransposeTableau[t]] ],
            Apply[GreaterEqual, ShapeOfTableau[t] ],
            Apply[GreaterEqual, Map[Length,TransposeTableau[t]] ]
      ]
TransposeTableau[tb_List] :=
      Module[{t=Select[tb,(Length[#]>=1)&],row},
            Table[
                  row = Map[First,t];
                  t = Map[ Rest, Select[t,(Length[#]>1)&] ];
                  row,
                  {Length[First[tb]]}
            ]
      ]
ShapeOfTableau[t_List] := Map[Length,t]
```

Identifying Young Tableaux

This is a tableau of shape $(3, 3, 1)$. Other same-shape tableaux containing 1 through 7 are also possible.

```
In[81]:= (t = {{1, 2, 5},{3, 4, 7}, {6}}) // TableForm
Out[81]//TableForm= 1   2   5
                    3   4   7
                    6
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

This test confirms that this is a tableau.

```
In[82]:= TableauQ[t]
Out[82]= True
```

Exchanging 6 and 7 gives a tableau of the same shape and with the same seven elements.

```
In[83]:= {t[[2, 3]], t[[3, 1]]} = {t[[3, 1]], t[[2, 3]]};
         TableauQ[t]
Out[84]= True
```

But this is not a tableau; note the first two elements of the second row.

```
In[85]:= TableauQ[{{1,2,5,9,10},{5,4,7,13},{4,8,12},{11}}]
Out[85]= False
```

Our interest in Young tableaux is combinatorial and follows Knuth [Knu73b], although their original application was algebraic. Young tableaux are also known as *plane partitions*. Another detailed treatment is [Sta71], which includes open problems.

■ 4.4.1 Insertion and Deletion

There is a remarkable bijection between permutations and ordered pairs of Young tableaux of the same shape. One of the implications of this is a bijection between involutions and Young tableaux. This bijection depends on two operations that are inverses of each other: an insertion operation and a deletion operation. These prove useful in several different contexts.

Here are the ten involutions of length 4.

```
In[86]:= Involutions[4]
Out[86]= {{2, 1, 4, 3}, {2, 1, 3, 4}, {3, 4, 1, 2},
    {3, 2, 1, 4}, {4, 3, 2, 1}, {4, 2, 3, 1}, {1, 3, 2, 4},
    {1, 4, 3, 2}, {1, 2, 4, 3}, {1, 2, 3, 4}}
```

Here are the ten tableaux that can be made with four distinct elements. These tableaux have different shapes, and the shapes are simply the partitions of 4. The function `Tableaux`, which we will discuss in Section 4.4.3, generates all tableaux of a given shape.

```
In[87]:= Map[Tableaux, Partitions[4]] // ColumnForm
Out[87]= {{{1, 2, 3, 4}}}
         {{{1, 3, 4}, {2}}, {{1, 2, 4}, {3}},

          {{1, 2, 3}, {4}}}
         {{{1, 3}, {2, 4}}, {{1, 2}, {3, 4}}}
         {{{1, 4}, {2}, {3}}, {{1, 3}, {2}, {4}},

          {{1, 2}, {3}, {4}}}
         {{{1}, {2}, {3}, {4}}}
```

Suppose we have a Young tableau T and an element x not in T. The insertion operation, on input T and x, returns a tableau T' that contains all the elements of T plus the extra element x. The shape of T' is identical to the shape of T except for one row, in which T' contains one more element than T. The insertion operation is implemented via the following algorithm. We start by attempting to insert x into the first row of T. If the element x is greater than the last element of the row, it can be placed at the end, resulting in the new tableau T'. If not, then we look for the least element y in the row that is larger than x. The element x takes y's place in the first row and "bumps" y down into the next row, where the insertion procedure starts anew with y. Eventually, either an element ends up on the end of a row or we run out of rows and the item must reside on a new row.

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

```
InsertIntoTableau[e_Integer, t_?TableauQ] := First[InsertIntoTableau[e, t, All]]

InsertIntoTableau[e_Integer,{}, All] := {{{e}}, 1}
InsertIntoTableau[e_Integer, t1_?TableauQ, All] :=
      Module[{item=e,row=0,col,t=t1},
            While [row < Length[t],
                  row++;
                  If [Last[t[[row]]] <= item,
                        AppendTo[t[[row]],item];
                        Return[{t, row}]
                  ];
                  col = Ceiling[ BinarySearch[t[[row]],item] ];
                  {item, t[[row,col]]} = {t[[row,col]], item};
            ];
            {Append[t, {item}], row+1}
      ]
```

Inserting a New Element into a Young Tableau

The new element 3 bumps the element 7 down to the next row. Element 7 finds a place at the end of the second row. The insertion changes the shape of the tableau from $(3,2,1)$ to $(3,3,1)$.

```
In[88]:= InsertIntoTableau[3, {{1, 2, 7}, {3, 4}, {6}}]
Out[88]= {{1, 2, 3}, {3, 4, 7}, {6}}
```

We can use the insertion algorithm to construct a Young tableau associated with a permutation π, by starting with an empty tableau and inserting each element of the permutation into the tableau in the order $\pi(1), \pi(2), \ldots, \pi(n)$.

```
ConstructTableau[p_List] := Module[{T}, T[a_, b_] := InsertIntoTableau[b, a]; Fold[T, {}, p]]
```

Constructing Young Tableaux

Each permutation generates a tableau, which is not necessarily unique.

```
In[89]:= ConstructTableau[{6,4,9,5,7,1,2,8}]
Out[89]= {{1, 2, 7, 8}, {4, 5}, {6, 9}}
```

In particular, this permutation generates the same tableau as the previous one.

```
In[90]:= ConstructTableau[{6,4,9,5,7,8,1,2}]
Out[90]= {{1, 2, 7, 8}, {4, 5}, {6, 9}}
```

Deletion is the inverse of insertion, in the following sense. Suppose that x was inserted into a tableau T, resulting in a new tableau T'. Further suppose that in row r, T' contains one more element than T. Then the deletion operation with inputs T' and r returns the tableau T. The algorithm for the deletion operation is essentially the opposite of the insertion algorithm. The last element in row r, say a, gets kicked up to row $(r-1)$, replacing the largest element smaller than a in the row. The replaced element, say b, gets kicked up to row $(r-2)$, and so on, until an element gets kicked out of the first row.

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

```
DeleteFromTableau[t1_?TableauQ,r_Integer]:=
      Module [{t=t1, col, row, item=Last[t1[[r]]]},
            col = Length[t[[r]]];
            If[col == 1, t = Drop[t,-1], t[[r]] = Drop[t[[r]],-1]];
            Do [
                  While [t[[row,col]]<=item && Length[t[[row]]]>col, col++];
                  If [item < t[[row,col]], col--];
                  {item,t[[row,col]]} = {t[[row,col]],item},
                  {row,r-1,1,-1}
            ];
            t
      ]
```

Deleting an Element From a Young Tableau

Observe that this tableau is missing 3.

```
In[91]:= TableForm[ ConstructTableau[{6,4,9,5,7,1,2,8}] ]
Out[91]//TableForm= 1   2   7   8
                    4   5
                    6   9
```

Inserting a 3 into the tableau bumps 7 from the first row and thus adds an element to the second row.

```
In[92]:= TableForm[ InsertIntoTableau[3,%] ]
Out[92]//TableForm= 1   2   3   8
                    4   5   7
                    6   9
```

Deleting the last element from the second row restores the tableau to its initial state.

```
In[93]:= TableForm[ DeleteFromTableau[%,2] ]
Out[93]//TableForm= 1   2   7   8
                    4   5
                    6   9
```

■ 4.4.2 Permutations and Pairs of Tableaux

From any given n-permutation π, we can build a pair of tableaux (P, Q) of identical shape, containing elements 1 through n. Start with a pair (P_0, Q_0) of empty tableaux. In the generic ith step of the algorithm, we have the tableaux pair (P_{i-1}, Q_{i-1}). Insert element $\pi(i)$ into tableau P_{i-1} and call the resulting tableau P_i. Let r be the row in P_i that has one extra element as a result of the insertion. Construct Q_i from Q_{i-1} by adding element i at the end of row r. Note that i is larger than any other element in Q_i, and hence it is the largest element in its row. Furthermore, there are no elements immediately below i. This implies that Q_i continues to be a tableau after the insertion of i. In general, it can be easily verified that P_i and Q_i are tableaux of identical shapes with P_i containing elements $\pi(1), \pi(2), \ldots, \pi(i)$ and Q_i containing elements $1, 2, \ldots, i$. Denote by (P, Q) the tableaux pair (P_n, Q_n) that we end up with after all the insertions.

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

We saw in an earlier example that two different permutations π and π' can lead to the same tableau P. However, the order in which P is constructed in one case is different from the order in which it is constructed in the other case. So "annotating" P with the tableau Q that is a record of how P was created is enough to disambiguate between the two situations. More precisely, we can construct a unique n-permutation π, given a tableaux pair (P, Q) of identical shape and each containing the elements 1 through n. This construction essentially reverses the construction of (P, Q) from π. We start looking for the element n in Q and if we find it in a row r, then that means that the last insertion into P caused row r to expand. We then delete element n from Q and perform the deletion operation on P at row r. In a generic step of this algorithm, we look for the largest element in Q. If i is the largest element in Q and it appears in row r_i, we then delete i from Q and perform the deletion operation on P at row r_i.

These algorithms give a bijection between ordered pairs of Young tableaux (P, Q) of the same shapes and permutations [Knu70, Sch61]. This is the celebrated *Robinson-Schensted-Knuth correspondence.*

```
PermutationToTableaux[{}] := {{}, {}}

PermutationToTableaux[p_?PermutationQ] :=
     Module[{pt = {{p[[1]]}}, qt = {{1}}, r},
          Do[{pt, r} = InsertIntoTableau[p[[i]], pt, All];
            If[r <= Length[qt], AppendTo[qt[[r]], i], AppendTo[qt, {i}]],
            {i, 2, Length[p]}
          ];
          {pt, qt}
     ]

TableauxToPermutation[p1_?TableauQ,q1_?TableauQ] :=
      Module[{p=p1, q=q1, row, firstrow},
            Reverse[
                  Table[
                        firstrow = First[p];
                        row = Position[q, Max[q]] [[1,1]];
                        p = DeleteFromTableau[p,row];
                        q[[row]] = Drop[ q[[row]], -1];
                        If[ p == {},
                              First[firstrow],
                              First[Complement[firstrow,First[p]]]
                        ],
                        {Apply[Plus,ShapeOfTableau[p1]]}
                  ]
            ]
      ] /; ShapeOfTableau[p1] === ShapeOfTableau[q1]
```

The Robinson-Schensted-Knuth Correspondence

To illustrate this bijection, we start with a random 10-permutation...

```
In[94]:= p = RandomPermutation[10]
Out[94]= {1, 3, 6, 4, 7, 10, 8, 5, 9, 2}
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

...and construct the corresponding tableaux pair. Note that these two tableaux have the same shapes and contain the elements 1 through 10.

```
In[95]:= {P, Q} = PermutationToTableaux[p]
Out[95]= {{{1, 2, 4, 5, 8, 9}, {3, 7}, {6}, {10}},
    {{1, 2, 3, 5, 6, 9}, {4, 7}, {8}, {10}}}
```

This takes us from the tableaux pair back to the original permutation.

```
In[96]:= TableauxToPermutation[P, Q]
Out[96]= {1, 3, 6, 4, 7, 10, 8, 5, 9, 2}
```

An important feature of this bijection is that the permutation π corresponds to the tableaux pair (P, Q) if and only if π^{-1} corresponds to the tableaux pair (Q, P).

The two tableaux corresponding to p reappear, but they are flipped in order.

```
In[97]:= q = InversePermutation[p];
         PermutationToTableaux[q]
Out[98]= {{{1, 2, 3, 5, 6, 9}, {4, 7}, {8}, {10}},
    {{1, 2, 4, 5, 8, 9}, {3, 7}, {6}, {10}}}
```

Since cycles have length at most 2, this is an involution. See Section 3.2.1 for more details on involutions.

```
In[99]:= p = FromCycles[{ {1, 4}, {3}, {2, 5}, {6, 7}}]
Out[99]= {4, 5, 3, 1, 2, 7, 6}
```

The tableaux pair corresponding to an involution contains identical permutations. This follows from the fact that if $\pi = \pi^{-1}$, then the corresponding tableaux pairs (P, Q) and (Q, P) are identical, implying that $P = Q$.

```
In[100]:= PermutationToTableaux[p]
Out[100]= {{{1, 2, 6}, {3, 5, 7}, {4}},
    {{1, 2, 6}, {3, 5, 7}, {4}}}
```

■ 4.4.3 Generating Young Tableaux

Systematically constructing all the Young tableaux of a given shape is a more complex problem than building the other combinatorial objects we have seen. Part of the problem is defining a logical sequence for the two-dimensional structures. We follow the construction of [NW78] and start by defining functions `LastLexicographicTableau` and `FirstLexicographicTableau`.

We define the first lexicographic tableau to consist of *columns* of contiguous integers.

```
In[101]:= TableForm[FirstLexicographicTableau[{4,3,3,2}]]
Out[101]//TableForm= 1   5   9   12
                     2   6   10
                     3   7   11
                     4   8
```

The last lexicographic tableau consists of *rows* of contiguous integers.

```
In[102]:= TableForm[ LastLexicographicTableau[{4,3,3,2}] ]
Out[102]//TableForm= 1   2   3   4
                     5   6   7
                     8   9   10
                     11  12
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

In the lexicographically last tableau, all rows consist of a contiguous range of integers. Thus for each integer k, the subtableau defined by the elements $1,...,k$ has k in the last row of the subtableau t. In general, to construct the lexicographically next tableau, we identify the smallest k such that k is *not* in the last row of its subtableau. Say this subtableau of k elements has shape $\{s_1,...,s_m\}$. The next tableaux will be the lexicographically first tableau with k in the next rightmost corner of the subtableaux, with the elements of t that are greater than k appended to the corresponding rows of the new subtableau.

The list of tableaux of shape $\{2,2,1\}$ illustrates the amount of freedom available to tableau structures. The smallest element is always in the upper left-hand corner, but the largest element is free to be the rightmost position of the last row defined by all the *distinct* parts of the partition.

```
In[103]:= Tableaux[{2,2,1}]
Out[103]= {{{1, 4}, {2, 5}, {3}}, {{1, 3}, {2, 5}, {4}},
     {{1, 2}, {3, 5}, {4}}, {{1, 3}, {2, 4}, {5}},
     {{1, 2}, {3, 4}, {5}}}
```

By iterating through the different integer partitions as shapes, all tableaux of a particular size can be constructed.

```
In[104]:= Tableaux[3]
Out[104]= {{{1, 2, 3}}, {{1, 3}, {2}}, {{1, 2}, {3}},
     {{1}, {2}, {3}}}
```

■ 4.4.4 Counting Tableaux by Shape

Each position p within a Young tableau defines an L-shaped *hook*, consisting of p, all the elements below p, and all the elements to the right of p. The *hook length formula* gives the number of tableaux of a given shape as $n!$ divided by the product of the hook length of each position, where n is the number of positions in the tableau. A convincing argument that the formula works is as follows: Of the $n!$ ways to label a tableau of a given shape, only those where the minimum element in each hook is in the corner can be tableaux, so for each hook the probability that the tableau condition is satisfied is one over the hook length. Unfortunately, this argument is bogus because these probabilities are not independent, but correct proofs that the formula works appear in [FZ82, Knu73b, NW78].

```
NumberOfTableaux[{}] := 1
NumberOfTableaux[s_List] :=
      Module[{row,col,transpose=TransposePartition[s]},
            (Apply[Plus,s])! /
            Product [
                  (transpose[[col]]-row+s[[row]]-col+1),
                  {row,Length[s]}, {col,s[[row]]}
            ]
      ]

NumberOfTableaux[n_Integer] := Apply[Plus, Map[NumberOfTableaux, Partitions[n]]]
```

The Hook Length Formula

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

The hook length formula can be used to count the number of tableaux for any shape, which for small sizes can be confirmed by constructing and counting them.

```
In[105]:= {NumberOfTableaux[{3,2,1}], Length[Tableaux[{3,2,1}]]}
Out[105]= {16, 16}
```

Using the hook length formula over all partitions of n computes the number of tableaux on n elements.

```
In[106]:= NumberOfTableaux[10]
Out[106]= 9496
```

A *biparential heap* [MS80] is a triangular data structure where each element is greater than both its parents (shown to the left and above each element). Pascal's triangle is an example of a biparential heap.

```
In[107]:= RandomTableau[ Reverse[Range[8]] ] // TableForm
Out[107]//TableForm= 1   2   3   4   10  16  17  18
                     5   6   11  12  23  30  31
                     7   8   15  22  32  34
                     9   19  21  26  33
                     13  20  25  27
                     14  24  28
                     29  36
                     35
```

The structure of a biparential heap is determined by a Young tableau of the given shape. Here we compute the number of biparential heaps on 55 distinct elements.

```
In[108]:= NumberOfTableaux[ Reverse[Range[10]] ]
Out[108]= 4426148608487407218364569920471040 0
```

Our study of Young tableaux was motivated by the number of well-formed formulas that can be made from n sets of parentheses, which we showed is the number of distinct tableaux of shape $\{n, n\}$. Since any balanced set of parentheses has a leftmost point $k+1$ at which the number of left and right parentheses are equal, peeling off the first left parenthesis and the $(k+1)$st right parenthesis leaves two balanced sets k and $n-1-k$ parentheses, which leads to the following recurrence:

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \binom{2n}{n}/(n+1).$$

This recurrence defines the *Catalan numbers*, which occur in a surprising number of problems in combinatorics, from the number of triangulations of a convex polygon to the number of paths across a lattice that do not rise above the main diagonal. This recurrence can be solved using generating functions to reveal the nice closed form

$$C_n = \frac{1}{(n+1)}\binom{2n}{n}.$$

The *Mathematica* add-on package `DiscreteMath`CombinatorialFunctions`` contains this definition of the function `CatalanNumber`.

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

This function is named `CatalanNumber` to avoid conflict with the built-in *Mathematica* function `Catalan`, which returns Catalan's constant.

```
In[109]:= Table[CatalanNumber[i], {i,2,20}]
Out[109]= {2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786,
    208012, 742900, 2674440, 9694845, 35357670, 129644790,
    477638700, 1767263190, 6564120420}
```

As we have seen, the Catalan numbers are a special case of the hook length formula.

```
In[110]:= Table[ NumberOfTableaux[{i,i}], {i,2,20}]
Out[110]= {2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786,
    208012, 742900, 2674440, 9694845, 35357670, 129644790,
    477638700, 1767263190, 6564120420}
```

■ 4.4.5 Random Tableaux

Constructing tableaux of a given shape randomly with a uniform distribution is another sticky problem solved in [NW78]. The key observation is that the largest element in the tableau must go in a rightmost corner of the shape, and that once the appropriate corner is selected the construction can proceed to position all smaller elements. Selecting a corner at random from the list of candidates will not give a uniform construction, since there are different numbers of ways to complete the tableau, depending on the shape left after the insertion. Using `NumberOfTableaux`, this can be computed for each of the possible shapes, and so a truly random corner selection can be made. `RandomTableau` implements this procedure.

Each of the 117,123,756,750 tableaux of this shape will be selected with equal likelihood.

```
In[111]:= TableForm[ RandomTableau[{6,5,5,4,3,2}] ]
Out[111]//TableForm= 1    2    3    6    9    15
                     4    8    10   18   21
                     5    12   17   22   23
                     7    13   19   25
                     11   16   24
                     14   20
```

Repeating the experiment in [NW78] illustrates that each of the 16 tableaux occurs with roughly equal frequency.

```
In[112]:= Distribution[ Table[RandomTableau[{3,2,1}], {1000}] ]
Out[112]= {75, 72, 63, 53, 66, 66, 58, 56, 54, 55, 64, 61,
    57, 56, 82, 62}
```

■ 4.4.6 Longest Increasing Subsequences

When we construct a tableau with the insertion algorithm, every element begins its life in some position in the first row, from which it may later be bumped. The elements that originally entered in the ith column are said to belong to the ith *class* of the tableau, and this class distinction leads to an interesting algorithm to find the longest increasing scattered subsequence of a permutation [Sch61]. `TableauClasses` partitions the elements of a permutation into classes.

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

```
TableauClasses[p_?PermutationQ] :=
      Module[{classes=Table[{},{Length[p]}],t={}},
            Scan [(t = InsertIntoTableau[#,t];
                  PrependTo[classes[[Position[First[t],#] [[1,1]] ]], #])&,
                  p
            ];
            Select[classes, (# != {})&]
      ]
```

Partitioning a Permutation into Classes

Note that $4,5,7,8$ form a longest increasing (scattered) subsequence of the following tableau, which contains the same number of columns.

```
In[113]:= TableauClasses[{6,4,9,5,7,1,2,8,3}]
Out[113]= {{1, 4, 6}, {2, 5, 9}, {3, 7}, {8}}
```

The reverse permutation has $3,5,9$ as a longest increasing subsequence, making it the longest decreasing subsequence of the original permutation.

```
In[114]:= TableauClasses[{3,8,2,1,7,5,9,4,6}]
Out[114]= {{1, 2, 3}, {4, 5, 7, 8}, {6, 9}}
```

This connection between the number of classes and the length of the longest scattered subsequence is not a coincidence. For an element of the permutation to enter the tableau in the kth column, there had to be elements creating $k-1$ columns before it. A column is created whenever the inserted element is greater than the last element of the first row, so clearly the elements that start new columns form an increasing subsequence. The fact that it is the *longest* increasing subsequence is a consequence of the first class corresponding to the left-to-right minima of the permutation and the kth class being the left-to-right minima of the permutation minus the elements of the first $k-1$ classes.

```
LongestIncreasingSubsequence[{}] := {}
LongestIncreasingSubsequence[p_?PermutationQ] :=
      Module[{c,x,xlast},
            c = TableauClasses[p];
            xlast = x = First[ Last[c] ];
            Append[Reverse[Map[(x = First[Intersection[#, Take[p, Position[p,x][[1,1]]]]])&,
                        Reverse[Drop[c,-1]]
                  ]
            ],
            xlast
      ]
```

Identifying the Longest Increasing Subsequence in a Permutation

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

The longest increasing *contiguous* subsequence can be found by selecting the largest of the runs in the permutation. Finding the largest scattered subsequence is a much harder problem.

```
In[115]:= First[ Sort[ Runs[p=RandomPermutation[50]], (Length[#1] >
          Length[#2])&] ]
Out[115]= {9, 26, 32, 39}
```

A pigeonhole result [ES35] states that any sequence of $n^2 + 1$ distinct integers must contain either an increasing or a decreasing scattered subsequence of length $n + 1$. Thus at least one of these sequences must be at least eight integers long.

```
In[116]:= {LongestIncreasingSubsequence[p],
          LongestIncreasingSubsequence[Reverse[p]]}
Out[116]= {{4, 6, 9, 11, 13, 15, 24, 31, 36, 38, 40, 44},
     {2, 3, 7, 14, 16, 27, 30, 32, 35, 45}}
```

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

4.5 Exercises

4.5.1 Thought Exercises

1. Write down the generating function for the number of partitions of n in which each part is at most 6.
2. Write down the generating function for the number of partitions of n in which each part is a multiple of 3.
3. Devise a *bijective* proof for the fact that for any positive integer n, the number of partitions of n with distinct parts equals the number of partitions of n with all odd parts. A bijective proof of an identity involves establishing a one-one correspondence between the set represented by the left-hand side and the set represented by the right-hand side.
4. Prove that the number of integer partitions with two parts $p_{n,2} = \lfloor n/2 \rfloor + 1$.
5. Prove that the size of the Durfee square remains unchanged in transposing an integer partition.
6. Explain why the minimum change graph on compositions is a grid of triangles.
7. Prove that the tableaux pair corresponding to an involution contains identical permutations.
8. Prove that $\left\{ {n \atop n-1} \right\} = \binom{n}{2}$.
9. Why is k not an argument for `UnrankKSetPartition`?
10. Consider the following attempt to generate a random RGF f on $\{1, 2, \ldots, n\}$: Set $f(1) = 1$; after generating $f(1), f(2), \ldots, f(i-1)$, we generate $f(i+1)$ by picking an integer in the set $\{f(i), f(i)+1, \ldots, n\}$, uniformly at random. Does this generate RGFs uniformly at random? If yes, prove it; if no, show two RGFs and prove that they are generated with distinct probabilities.

4.5.2 Programming Exercises

1. A recursive algorithm to generate a list of all set partitions of $\{1, 2, \ldots, n\}$ in minimum change order is as follows [Lay76]. Let π be any partition of $\{2, 3, \ldots, n\}$. The *children* of π are obtained by inserting 1 into one of the blocks of π or adjoining 1 to π as a singleton block. Suppose inductively we have a list L_{n-1} of set partitions of $\{2, 3, \ldots, n\}$ in minimum change order. Take the first set partition in L_{n-1} and list its children in natural order, that is, 1 into the first block, 1 into the second block, and so on. Then take the children on the second set partition in L_{n-1} and list them in reverse order, and repeat this alternation with each group of siblings.
   Implement this algorithm. Also, use the idea to devise an algorithm to list the set partitions of $\{1, 2, \ldots, n\}$ with k blocks, for a given k, in minimum change order.

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

2. Let us suppose that the identities

$$\left\{ {n \atop k} \right\} = \left\{ {n-1 \atop k-1} \right\} + k \left\{ {n-1 \atop k} \right\} \qquad \text{and} \qquad \left[{n \atop k} \right] = \left[{n-1 \atop k-1} \right] + (n-1) \left[{n-1 \atop k} \right]$$

are valid for *all* integers, not just nonnegative integers. Assume the standard boundary conditions: If $n = k = 0$, both numbers are 1; otherwise, if $n = 0$ or $k = 0$, both numbers are 0. This leads to unique solutions to the two recurrences, thereby giving us values for Stirling numbers with negative arguments. Extend the functions `StirlingFirst` and `StirlingSecond` so that they work for negative arguments as well.

Using this function, produce a table of $\left\{ {n \atop k} \right\}$ values for $k \in [-5 \ldots 5]$ and $n \in [-5 \ldots 5]$. Examine the table carefully and conclude a neat "duality" between Stirling numbers of the two kinds.

3. An alternate recurrence [NW78] for the number of integer partitions of n is

$$p_n = \frac{1}{n} \sum_{m=0}^{n-1} \sigma(n-m) p_m,$$

where $p_0 = 1$ and $\sigma(k)$ is the sum of the divisors of k. Does this recurrence give a better way to calculate p_n than `NumberOfPartitions`? You may use the built-in *Mathematica* function `DivisorSigma[1,k]` to compute $\sigma(k)$.

4. Design and implement an efficient algorithm for constructing all partitions of n with distinct parts. Compare it to using `Select` and `Partitions`.

5. Repeat the previous exercise, constructing all partitions of n into odd parts, even parts, distinct odd parts, and finally all *self-conjugate* partitions, meaning they are equal to their transpose.

6. Implement a function [Sav89] to generate the partitions of n in Gray code order, as defined in the chapter.

■ 4.5.3 Experimental Exercises

1. Experiment with `NumberOfTableaux` to determine the shape that, for a given number of elements, maximizes the number of Young tableaux over all partitions of n.

2. A partition p *contains* partition q if the Ferrers diagram of p contains the Ferrers diagram of q. For example, $\{3,3,2\}$ contains both $\{3,3,1\}$ and $\{3,2,2\}$ as well as many smaller partitions. *Young's lattice* Y_p [SW86] is the partial order of the partitions contained within p ordered by containment. Use `MakeGraph` to construct Young's lattice for an arbitrary partition p.

3. `ConstructTableau` takes a permutation to a Young tableau that is not necessarily unique. For the permutations of length n, which tableaux are the most frequently generated?

4. Experiment to determine the expected amount of vacant dots in the rectangle defined by the Ferrers diagram of a random partition of size n.

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press

5. It is easy to observe from various examples in this chapter that $p(n)$ grows quite slowly as compared to $n!$. How does $p(n)$ compare to 2^n? Experiment by comparing $p(n)$ with a variety of functions. Do you think $p(n)$ grows exponentially?

6. Examining the sequence

$$\left\{ {n \atop 1} \right\}, \left\{ {n \atop 2} \right\}, \ldots, \left\{ {n \atop n} \right\}$$

for a few values of n should convince the reader that elements in this sequence increase strictly starting at 1, reach a maximum value, and then decrease strictly down to 1. Strangely enough, this property is an unproven conjecture! Aigner [Aig79] proves that this sequence strictly increases until it reaches a plateau of at most 2 maxima and then strictly decreases. Write a function that takes a positive integer n and tests the uniqueness of the maximum value in $\{ \left\{ {n \atop k} \right\} \mid k \in [n] \}$. What is the largest value of n that you can reasonably test using this technique?

7. A result of Canfield [Can78b] provides a faster way of testing the conjecture described above. Let K_n be a positive integer such that

$$\left\{ {n \atop K_n} \right\} \geq \left\{ {n \atop k} \right\} \text{ for all } k \in \{1, 2, \ldots, n\}.$$

Then, for sufficiently large n, $K_n = \lfloor t \rfloor$ or $K_n = \lfloor t \rfloor + 1$, where t is the solution to the equation

$$\frac{(t+2)t\log(t+2)}{t+1} = n.$$

Write a function that takes a positive integer n and uses this technique to test the uniqueness of maxima conjecture for Stirling numbers of the second kind. What is the largest value of n for which this test can be performed in a reasonable amount of time?

https://doi.org/10.1017/CBO9781139164849.005 Published online by Cambridge University Press