Counting Partitions

M. D. Sheppeard

November 2019

Abstract

In this lecture we count the number of integer partitions $P(n)$ using an elementary algorithm based on the combinatorics of trees, coded using free apps on the iPad.

Integer partitions appear throughout mathematics. We will count them using the combinatorics of trees and cubes, fundamental to categorical methods in physics.

Here an ordinal $n$ takes a value in $\mathbb{N}^+ = \{1, 2, 3, 4, \cdots\}$ and $P(n)$ denotes the number of partitions of $n$ into any number of parts $r \geq 1$. We write

$$n = \lambda_1 + \lambda_2 + \cdots + \lambda_r,$$

where it is assumed that $\lambda_i \geq \lambda_{i+1}$ for any $i$, and each part is at least 1, so that an expression like $6 = 3 + 2 + 1$ is unique. Now $P(3) = 3$ counts the list $3, 2 + 1, 1 + 1 + 1$.

These ordinary partitions are commutative, because we do not differentiate between $3 + 2 + 1$ and $3 + 1 + 2$. But noncommutative partitions are far easier to count than commutative ones. Figure 1 shows how any noncommutative partition of $n$ is represented by either a rooted tree with two levels, or a set of $n - 1$ signs corresponding to the areas between the leaves on the tree. There are clearly $2^{n-1}$ such partitions. Beginning with this set of signs, for any given $n$, we construct an algorithm for computing $P(n)$.

First, observe that the integer partitions only see how the $+$ signs are partitioned within a string of signs. For example, in figure 1 for $P(4) = 5$, the partition $4 = 2 + 2$ corresponds to the string $+-+$, which we now think of as the two part string of strings $(+,+)$. A single $+$ sign in such a string denotes the number 2, a string $++$ denotes 3 and so on, so that $+-+--$ stands for the partition $6 = 3 + 2 + 1$, as does any other string of the form $(++,+)$. We count $P(n)$ by counting these sign strings. Let $k$ be the number of entries in a string of strings, so that for $(++,+)$ we have $k = 2$. For each value of $k$ up to the maximum, we first generate a list of perversities of length $k$, so that $(++,+)$ will become the numerical word 12. The maximum $k$ is simply the floor of $n/2$. A perversity (for homology [2]) is, by definition, a finite list $(x_1, x_2, \cdots, x_k)$ of ordinals such that $x_i \leq x_{i+1}$ for all $1 \leq i \leq k - 1$. In other words, a perversity is a partition. For example, each row in the list of matrices
in figure 4 corresponds to a perversity for the calculation of \( n = 6 \). At \( k = 1 \), we have up to 5 plus signs in the string. At \( k = 2 \), the maximum digit value is 3, because the space between the two parts is a minus sign. In general, the maximum digit is

\[ rr = n - 1 - 2(k - 1). \]

This set of perversities \( \mathcal{P}(n) \) is then truncated as follows. Let \( s \) equal the sum of digits in a perversity. In the calculation of \( P(n) \), we require that the maximum value of \( s \) be \( n - k \), because we only have \( n - 1 \) signs and we need \( k - 1 \) spaces. Let \( N(n, k) \) be the number of elements of \( \mathcal{P}(n) \) satisfying this condition. Then

\[
P(n) = 1 + \sum_{k=1}^{\text{floor}(n/2)} N(n, k).
\]

Although \( N(n, k) \) is itself made up of partitions, the computer may evaluate it with a blind search.

Let us look at the code. Using the MathStudio app on the iPad, the brute force code of figure 2 generates any given set of perversities, for \( rr \) letters in the alphabet and words of length \( ll \). It begins by generating all possible words in the letters \( \{1, 2, \cdots, rr\} \), and then selecting only those that are perversities. A sample output is given in figure 3 for the ten points on the tetractys triangle. The total number of perversities for a given pair of parameters is always a tetrahedral number. Considering the numerals \( \{1, 2, 3\} \) as letters \( X, Y \) and \( Z \) in an alphabet, the usual integer simplex coordinates are given by the number of occurrences of each letter in a word, so that for instance \( 111 \mapsto XXX \mapsto 300 \), and all coordinates have entries summing to 3.

Once a library of perversities is obtained, as shown in figure 4, we count \( P(n) \) by properly truncating the set. Although the computational complexity of this algorithm grows as expected with \( n \), it only involves very elementary logic operations on lists of ordinals.
Figure 2: Building perversities and simplices

References


Figure 3: Tetractys simplex on 10 points for \( rr = 3 \) and \( ll = 3 \)

Figure 4: Counting \( P(6) \) by summing allowed perversities