SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Integer Partitions

av

Teo Johannesson

2021 - No K47

Integer Partitions

Teo Johannesson

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Paul Vaderlind

2021

Integer Partitions

Teo Johannesson

November 10, 2021

Abstract

This thesis looks into some selected elements in the theory of integer partitions. The goal of the thesis is to explore methods to select partitions at random. Before these can be developed some preliminary theory is introduced. Ferrers diagrams are presented, a graphical representation for integer partitions. Then the theory of generating functions related to integer partitions is expanded on, which are useful for proving partition identities. Euler's pentagonal number theorem is presented and proven bijectively. This theorem is then used to derive a recurrence relation for the partition function. Other properties of the partition function $p(n)$ are explored and the partition function $p(n, k)$ is introduced. Different ways of ordering partitions are considered. Finally, the problem of selecting integer partitions at random is studied and algorithms for generating random partitions are developed. Difficulties in selecting partitions at random are tied to the elusive nature of the partition function $p(n)$. A better understanding of algorithms generating random partitions could give insight into $p(n)$.

Contents

1 Introduction

1.1 Integer partitions

A partition of a positive integer n is a sequence of positive integers $p_1, p_2, ..., p_k$ whose sum is n.

$$n = p_1 + p_2 + ... + p_k.$$

The summands are often referred to as parts. Two partitions with the same parts in different order are considered to be the same partition. For example, $3 + 2$ is a partition of 5 and is the same partition as $2 + 3$. All seven partitions of 5 are:

5,
$4 + 1$,
$3 + 2$,
$3 + 1 + 1$,
$2 + 2 + 1$,
$2 + 1 + 1 + 1$,
$1 + 1 + 1 + 1 + 1$.

1.2 The partition function

The number of partitions of a positive integer n is given by the partition function $p(n)$. There is no known simple formula for the partition function but there are asymptomatic formulas that can accurately compute $p(n)$[3]. Let $p(0) = 1$, then the first values of the partition function $p(n)$ are

$p(0) = 1$,
$p(1) = 1$,
$p(2) = 2$,
$p(3) = 3$,
$p(4) = 5$,
$p(5) = 7$,
$p(6) = 11$,
$p(7) = 15$.

The number of partitions of n increases rapidly as n increases. For example, $p(100) = 190\,569\,292$. How quickly it grows and other properties of the partition function will be studied more in-depth in section 5.

1.3 Partition identities

Sometimes it's interesting to consider partitions under various restrictions. A well know theorem proved by Euler, referred to as Euler's identity, says that the number of partitions with only odd parts equals the number of partitions with parts that are all distinct. For example, of the seven partitions of 5 there are three partitions with odd parts,

$$5, \quad 3 + 1 + 1, \quad 1 + 1 + 1 + 1 + 1,$$

and three partitions with distinct parts,

$$5, \quad 4+1, \quad 3+2.$$

Statements of this kind are called partition identities and the partition identity above can be written as

$$p(n \mid \text{odd parts}) = p(n \mid \text{distinct parts}).$$

Partition identities can be proved by constructing a bijection between the partitions in the different sets. A bijection proving Euler's identity is able to take a partition with odd parts and transform it into a unique partition with distinct parts, and the reverse. They can also be proved with generating functions, which will be introduced in section 3.

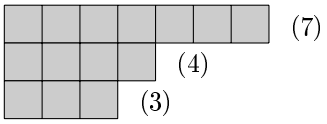
To create a bijection proving Euler's identity, let π be a partition with only odd parts, not necessarily distinct. For parts that occur more than once, merge these two by two. Repeat this process until all parts are distinct. The process will terminate at latest when only one part remains. For example, the partition $5+3+3+3+1+1+1+1$ would undergo the transformation

$$5+3+3+3+1+1+1+1 \ \rightarrow \ 6+5+3+2+2 \ \rightarrow \ 6+5+4+3.$$

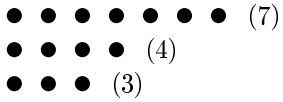
New parts created in this process by merging parts of odd size O_i in π take the form $2^k O_i$, $k \geq 0$. Hence, new parts created from distinct odd parts in π don't merge. It follows that parts in the new partition corresponding to O_i in π can be split in half successively to retrieve the same number of O_i that occurred in π. This proves that every partition with odd parts transforms into a unique partition with distinct parts. With the reverse process, and by a similar argument, it can be proved that any partition with distinct parts transforms into a unique partition with odd parts. This shows that it is a bijection.

2 Ferrers diagrams

A common way to represent partitions is with Ferrers diagrams (or Ferrers boards, sometimes also called Young diagrams). Many properties of partitions are more easily understood with the use of Ferrers diagrams. The partition $7+4+3$ can be represented with the diagram:



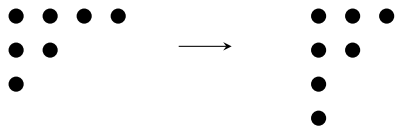
Sometimes using dots in the diagram is more convenient.



The rules for constructing Ferrers diagrams are intuitive, but can be put concretely. For each part in the partition, put an amount of squares (or dots) equal

to the size of the part on each row, left-justified. Then order the rows descending by size.

Conjugation is a transformation where rows in a Ferrers diagram are exchanged for columns:



The partition obtained is called the conjugate of the partition. This transformation is a bijection, so it immediately provides a proof for the partition identity

$$p(n \mid m \text{ parts}) = p(n \mid \text{largest part is } m).$$

In words it can be stated as: The number of partitions with m parts is equal to the number of partitions whose largest part is m.

3 Generating functions

The fundamental properties of multiplying powers together can be used to answer questions about integer partitions. Suppose we want to find the number of partitions of 7 with one odd part and one even part. It is possible to find the answer by enumerating all partitions of 7 and then select those satisfying the condition. Another way is to make use of the equation

$$(q^1 + q^3 + q^5)(q^2 + q^4 + q^6).$$

A partition of 7 with one odd part and one even part has to take one part from $\{1, 3, 5\}$ and the other from $\{2, 4, 6\}$. So the answer can be found by evaluating the product above and examining the coefficient of q^7.

The idea can be extended further. Let $S = \{r_1, r_2, ..., r_m\}$ be some finite set of integers. Then all partitions with distinct parts from S are present as exponents when evaluating the expression

$$(1 + q^{r_1})(1 + q^{r_2}) \ldots (1 + q^{r_m}) = \prod_{r \in S} (1 + q^r). \tag{1}$$

To see this, consider any integer r_i in S. A partition with distinct parts from S either includes r_i or it does not, which corresponds to selecting either 1 or q^{r_i} from the factor $(1 + q^{r_i})$.

The number of partitions for any positive integer n with distinct parts from S is determined by the coefficient of q^n. This can be expressed with

$$\sum_{n \geq 0} p(n \mid \text{distinct parts in } S) q^n = \prod_{r \in S} (1 + q^r),$$

and is called the generating function for partitions into distinct parts in S.

Suppose d copies of any part r from S is allowed. Then each factor in (1) take the form $(1 + q^r + q^{2r} + ... + q^{dr})$. This can be simplified by using the formula for

finite geometric series. For $x \neq 1$, the sum of the first $n+1$ terms of a geometric series is

$$1 + x + x^2 + ... + x^n = \sum_{k=0}^{n} x^k = \frac{1 - x^{n+1}}{1 - x}. \tag{2}$$

The generating function for partitions into parts from S with at most d copies of any part then becomes

$$\begin{aligned}
&\sum_{n \geq 0} p(n \mid \text{parts in } S, \text{ at most } d \text{ copies of any part}) q^n \\
&= \prod_{r \in S} (1 + q^r + q^{2r} + ... + q^{dr}) \\
&= \prod_{r \in S} \frac{1 - q^{(d+1)r}}{1 - q^r}.
\end{aligned}$$

For partitions with parts in S, where parts are allowed to occur any number of times, each factor in (1) then take the form $(1 + q^r + q^{2r} + ...)$. If the limit $n \to \infty$ in (2) is considered, then for $|x| < 1$ the sum becomes

$$1 + x + x^2 + ... = \sum_{k=0}^{\infty} x^k = \frac{1}{1 - x}.$$

With this, the generating function for partitions with parts in S becomes

$$\begin{aligned}
&\sum_{n \geq 0} p(n \mid \text{parts in } S) q^n \\
&= \prod_{r \in S} (1 + q^r + q^{2r} + ...) \\
&= \prod_{r \in S} \frac{1}{1 - q^r}.
\end{aligned}$$

In deriving these functions, some restrictions where made on q to enable the use of formulas for geometric series. Since q is used only to provide the machinery of exponents, this poses no problem.

Also important to note is that the set S considered above is finite. In deriving the formulas no reference is made to the amount of elements in S, so it's plausible that the formulas hold for infinite sets. This will be assumed to be true without proof. Thus, for any set of positive integers S, we have

$$\sum_{n \geq 0} p(n \mid \text{distinct parts in } S) q^n = \prod_{r \in S} (1 + q^r), \tag{3}$$

$$\sum_{n \geq 0} p(n \mid \text{parts in } S, \text{ at most } d \text{ copies of any part}) q^n = \prod_{r \in S} \frac{1 - q^{(d+1)r}}{1 - q^r}, \tag{4}$$

$$\sum_{n \geq 0} p(n \mid \text{parts in } S) q^n = \prod_{r \in S} \frac{1}{1 - q^r}. \tag{5}$$

3.1 Generating functions with two variables

Suppose an additional variable z is added to (1) such that

$$(1+zq^{r_1})(1+zq^{r_2})...(1+zq^{r_m}) = \prod_{r\in S}(1+zq^r).$$

When evaluating the above expression, the exponent of z will track the number of parts in the partition. For example, with $m=3$ it evaluates to

$$(1+zq^{r_1})(1+zq^{r_2})(1+zq^{r_3}) = 1+zq^{r_1}+zq^{r_2}+zq^{r_3}+z^2q^{r_1+r_2}+z^2q^{r_1+r_3}+z^2q^{r_2+r_3}+z^3q^{r_1+r_2+r_3}.$$

From this, an analog to (3) with two variables is

$$\sum_{n\geq 0}\sum_{m\geq 0} p(n \mid m \text{ distinct parts in } S)z^m q^n = \prod_{r\in S}(1+zq^r). \tag{6}$$

Similarly, (4) and (5) becomes

$$\sum_{n\geq 0}\sum_{m\geq 0} p(n \mid m \text{ parts in } S, \text{ at most } d \text{ copies of any part})z^m q^n = \prod_{r\in S}\frac{1-z^{d+1}q^{(d+1)r}}{1-zq^r},$$

$$\sum_{n\geq 0}\sum_{m\geq 0} p(n \mid m \text{ parts in } S)z^m q^n = \prod_{r\in S}\frac{1}{1-zq^r}.$$

4 Euler's pentagonal number theorem

Consider the infinite product $(1-x)(1-x^2)(1-x^3)...$. Expanding it for the first few terms gives

$$(1-x)(1-x^2)(1-x^3)... = 1-x-x^2+x^5+x^7-x^{12}-x^{15}+...$$

Euler deduced that the terms follow the law

$$\prod_{n=1}^{\infty}(1-x^n) = \sum_{k=-\infty}^{\infty}(-1)^k x^{k(3k-1)/2}, \tag{7}$$

and is known as the pentagonal number theorem[1].

4.1 Pentagonal numbers

The exponents of x on the right side in (7) are generalized pentagonal numbers. Pentagonal numbers are figurative numbers related to the number of dots in regular pentagons. The n'th pentagonal number p_n is the number of distinct dots in an array of n pentagons with sides ranging from 0 to $n-1$. Starting at a vertex, a dot is placed at each unit distance. The pentagons are overlaid so that they share one dot. Pentagons representing the first four pentagonal numbers are illustrated in Figure 1.

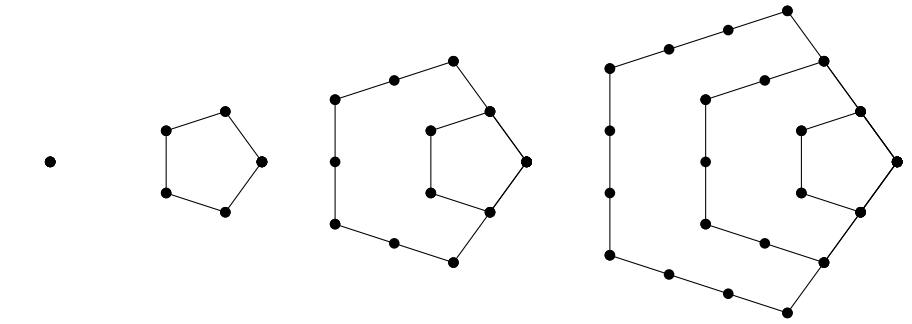


Figure 1: A visual representation of the first four pentagonal numbers.

The pentagonal numbers are given by the formula

$$p_n = \frac{3n^2 - n}{2}, \tag{8}$$

for $n \geq 1$. The first pentagonal numbers are then

$$1, 5, 12, 22, 35, 51, 70, 92, 117, 145, 176, 210, 247.$$

Generalized pentagonal numbers are obtained from (8) when n is any integer. Equivalently, they are given by the formula

$$p_n = \frac{3n^2 \pm n}{2},$$

for $n \geq 0$. The first generalized pentagonal numbers are then

$$0, 1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, 57, 70, 77.$$

4.2 Relation with partitions

The left side of (7) can be obtained from (6) by setting $z = -1$. This then establishes the identity

$$\prod_{n=1}^{\infty}(1-x^n) = \sum_{n\geq 0}\sum_{m\geq 0} p(n \mid m \text{ distinct parts})(-1)^m q^n$$
$$= \sum_{n\geq 0}(p(n \mid \text{even number of distinct parts}) - p(n \mid \text{odd number of distinct parts}))q^n.$$

For n fixed it means that

$$p(n \mid \text{even number of distinct parts}) - p(n \mid \text{odd number of distinct parts})$$
$$= \begin{cases} (-1)^k & \text{if } n = k(3k \pm 1)/2 \\ 0 & \text{otherwise.} \end{cases}$$

This can then be proved by first attempting to construct a bijection between

partitions of n with an even number of distinct parts, and partitions of n with an odd number of distinct parts. Then show that one partition is left over when n is equal to a generalized pentagonal number.

Euler provided a proof for the theorem, but did so using generating functions. It was only later that Legendre put the theorem in the context of integer partitions. Thereafter Fabian Franklin discovered a bijective proof[3] which is presented in the next section.

4.3 A bijective proof for the pentagonal number theorem

A bijection between partitions counted by $p(n \mid \text{even number of distinct parts})$ and $p(n \mid \text{odd number of distinct parts})$ could be constructed by defining an invertible transformation that takes a partition of n with an even number of distinct parts and transforms it into a partition of n with an odd number of distinct parts.

For any partition of n with distinct parts, define s to be the number of parts that differ by 1, starting with the largest part. Define m to be the size of the smallest part. For example, in the partition of $n = 23$ defined by $23 = 7 + 6 + 5 + 3 + 2$ we have $s = 3$ and $m = 2$. The number s can also be seen as the longest diagonal that can be drawn in a Ferrers diagram, starting from the top, illustrated in Figure 2.

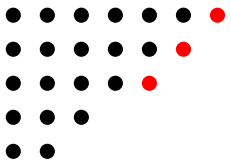


Figure 2: Ferrers diagram for the partition of 23 defined by $23 = 7+6+5+3+2$ with the diagonal corresponding to s in red.

Define a transformation for partitions of n with distinct parts by: If $m \leq s$, then put the smallest part of the partition on the diagonal. Otherwise, if $m > s$, then create a new smallest part with the diagonal. Figure 3 and Figure 4 illustrates the transformation.

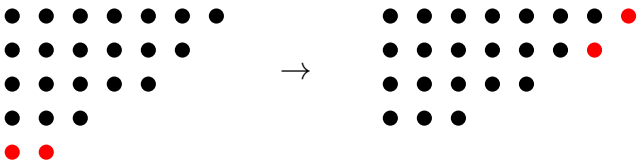


Figure 3: If $m \leq s$, then the transformation puts the smallest part on the diagonal.

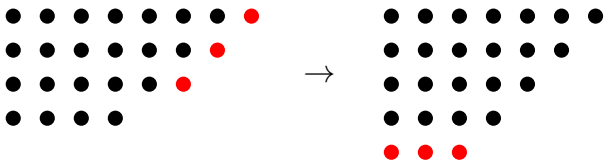


Figure 4: If $m > s$, then the transformation creates a new smallest part from the diagonal.

Transforming a partition this way either removes a part, or creates a new part, and therefore changes the parity of the number of parts. The parts in the new partition are distinct and the transformation is reversible; performing the transformation twice brings back the original partition. This means that every partition with an odd number of distinct parts is paired with a partition with an even number of distinct parts.

This holds true except for two special cases. When the diagonal intersects the smallest part, and the smallest part is equal to the diagonal or one dot larger, then the transformation is no longer reversible. For example, with the partition of $n = 12$ defined by $12 = 5+4+3$, the transformation fails to change the parity of the partition, illustrated in Figure 5.

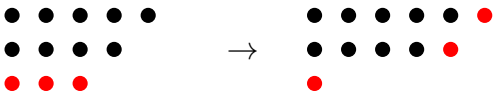


Figure 5: If the diagonal intersects the smallest part, and $s = m$, then the transformation fails to change the parity of the partition.

In the case when the smallest part and the diagonal intersect, and the smallest part is one dot larger than the diagonal, the transformation fails to create a partition with distinct parts, illustrated in Figure 6.

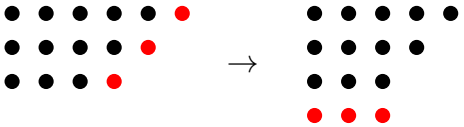


Figure 6: When the diagonal intersects the smallest part, and $s + 1 = m$, then the transformation fails to create a partition with distinct parts.

These special partitions can only be created for certain n. First, consider the case when $s = m$. Then

$$n = m + (m+1) + ... + (2m-1) = m^2 + \frac{m(m-1)}{2} = \frac{m(3m-1)}{2} = \frac{s(3s-1)}{2}.$$

In the case $s+1 = m$, when the smallest part is one dot larger than the diagonal, then

$$n = m + (m+1) + ... + (2m-2) = \frac{m(3m-1)}{2} - (2m-1) = \frac{(3m-2)(m-1)}{2}.$$

Replacing m with s, we get

$$\frac{(3m-2)(m-1)}{2} = \frac{(3(s+1)-2)((s+1)-1)}{2} = \frac{s(3s+1)}{2}.$$

For these n, where $n = \frac{k(3k\pm1)}{2}$ with $k \geq 0$, there's exactly one partition that can't be transformed, and this partition has k parts. For other n, every partition with an odd number of distinct parts can be paired with a partition with an even number of distinct parts. This proves that for any positive integer n we have

$$\begin{aligned} &p(n \mid \text{even number of distinct parts}) - p(n \mid \text{odd number of distinct parts}) \\ &= \begin{cases} (-1)^k & \text{if } n = k(3k \pm 1)/2 \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

5 Properties of the partition function $p(n)$

5.1 A recurrence relation for $p(n)$

While there's no simple formula for the partition function its values can be computed using recurrence relations. A recurrence relation for $p(n)$ can be obtained from the pentagonal number theorem.

For $k = 0$ the term on the right side in (7) is $(-1)^0 x^{0(3\cdot 0-1)/2} = 1$. The terms for negative k can be rewritten with

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

This shows the equality

$$\sum_{k=-\infty}^{\infty} (-1)^k x^{k(3k-1)/2} = 1 + \sum_{k=1}^{\infty} (-1)^k x^{k(3k-1)/2}(1+x^k). \tag{9}$$

In (5), let S be the set of positive integers, then

$$\sum_{n=0}^{\infty} p(n) q^n = \prod_{r=1}^{\infty} \frac{1}{1-q^r},$$

or equivalently,

$$\prod_{r=1}^{\infty} (1-q^r) \sum_{n=0}^{\infty} p(n) q^n = 1.$$

Together with (7) and (9) we have

$$(1 + \sum_{k=1}^{\infty} (-1)^k q^{k(3k-1)/2}(1+q^k)) \sum_{n=0}^{\infty} p(n) q^n = 1.$$

By comparing coefficients of q^n a recurrence formula for $p(n)$ can be obtained. First observe that $p(0)q^0 = 1$. Subtracting 1 from both sides and expanding the product gives

$$\sum_{n=1}^{\infty} p(n)q^n + \sum_{m=0}^{\infty} p(m)q^m \sum_{k=1}^{\infty} (-1)^k q^{k(3k-1)/2}(1+q^k) = 0.$$

This shows that for a given q^n, the factor $p(n)$ in the series to the left must equal the negative sum of the coefficients of q^n in the series to the right. The terms can be reordered to better reveal the coefficients.

$$\begin{aligned}
&\sum_{m=0}^{\infty} p(m)q^m \sum_{k=1}^{\infty} (-1)^k q^{k(3k-1)/2}(1+q^k) \\
&= \sum_{m=0}^{\infty}\sum_{k=1}^{\infty} (-1)^k p(m) q^{m+k(3k-1)/2} + \sum_{m=0}^{\infty}\sum_{k=1}^{\infty} (-1)^k p(m) q^{m+k(3k+1)/2} \\
&= \sum_{n=1}^{\infty}\sum_{k=1}^{\infty} (-1)^k p(n-k(3k-1)/2) q^n + \sum_{n=1}^{\infty}\sum_{k=1}^{\infty} (-1)^k p(n-k(3k+1)/2) q^n.
\end{aligned}$$

In the last step m is replaced with n by setting $n = m + k(3k \pm 1)/2$.

So for a fixed n, we have the recurrence relation

$$\begin{aligned}
p(n) &= -\sum_{k=1}^{\infty} (-1)^k p(n-k(3k-1)/2) - \sum_{k=1}^{\infty} (-1)^k p(n-k(3k+1)/2) \\
&= p(n-1) + p(n-2) - p(n-5) - p(n-7) + \ldots
\end{aligned}$$

5.2 Upper bounds for $p(n)$

It was mentioned in the beginning that the partition function $p(n)$ is increasing and grows rapidly. This section proves that $p(n)$ is strictly increasing for $n \geq 1$. It also puts an upper bound on $p(n)$ by proving that both the number of compositions and the sequence of Fibonacci numbers grows faster than $p(n)$.

5.2.1 $p(n)$ is increasing

For any positive integer n, a partition of $n-1$ can be transformed into a unique partition of n by adding a 1-part to it. The transformation can be reversed for any partition of n with at least one 1-part. This shows that

$$p(n \mid \text{at least one 1-part}) = p(n-1).$$

Every partition of n either has a 1-part or it doesn't, and for $n \geq 2$ there's at least one partition of n with no 1-part. For $n \geq 2$, then

$$p(n) = p(n \mid \text{at least one 1-part}) + p(n \mid \text{no 1-part}) > p(n-1). \tag{10}$$

By induction, this shows that $p(n)$ is strictly increasing for $n \geq 1$.

5.2.2 An upper bound on $p(n)$ with compositions

An upper bound for the partition function $p(n)$ can be obtained from the corresponding function counting compositions.

A composition of a positive integer n is a sequence of positive integers $p_1,\ p_2,\ ...,\ p_k$ whose sum is n. In contrast to partitions, different orderings of the integers yields different compositions. Denote the number of compositions of n by $c(n)$. Then, since every partition corresponds to a unique composition, we have $p(n) \leq c(n)$. For any integer $n \geq 3$ there's always at least one partition that corresponds to a composition with parts that can be ordered differently to obtain other compositions, so for $n \geq 3$ the relation is strict

$$p(n) < c(n).$$

The number of compositions $c(n)$ is much easier to compute than the number of partitions $p(n)$. Every composition can uniquely be written as sums of 1's separated by commas. For example, the composition $2 + 3$ of 5 can be written as

$$1 + 1, 1 + 1 + 1.$$

Between each 1 there's either a plus sign or a comma. So to determine a composition of n is equivalent to selecting ',' or '+', $n - 1$ times. This can be done in 2^{n-1} ways, hence $c(n) = 2^{n-1}$. An upper bound on $p(n)$ is then

$$p(n) < 2^{n-1},$$

for $n \geq 3$.

5.2.3 An upper bound on $p(n)$ with the Fibonacci numbers

Another upper bound for the partition function is given by the set of Fibonacci numbers. The Fibonacci numbers is the famous number sequence defined by:

$F_0 = 0,$
$F_1 = 1,$
and for $n \geq 2,$
$F_n = F_{n-1} + F_{n-2}.$

An expression for how $p(n)$ relates to $p(n-1)$ and $p(n-2)$ would enable a comparison with the Fibonacci numbers. From (10), we have

$$p(n) = p(n-1) + p(n \mid \text{no 1-part}). \tag{11}$$

How does $p(n \mid \text{no 1-part})$ relate to $p(n-2)$?
A partition π counted by $p(n \mid \text{no 1-part})$ can be transformed into a unique partition of $n-2$ by taking the smallest part of size k of π and split it into one 2-part and $(k-2)$ 1-parts, and then remove the 2-part. Note that $k \geq 2$ always holds. This shows that for $n \geq 2$

$$p(n-2) \geq p(n \mid \text{no 1-part}).$$

Together with (11) we have

$$p(n) \leq p(n-1) + p(n-2).$$

Now suppose $p(k-1) \leq F_k$ and $p(k-2) \leq F_{k-1}$ for some integer $k \geq 2$. Then

$$p(k) \leq p(k-1) + p(n-2) \leq F_k + F_{k-1} = F_{k+1}.$$

Since $p(0) \leq F_1$ and $p(1) \leq F_2$, this proves by induction that

$$p(n) \leq F_{n+1},$$

for any $n \geq 0$.

5.3 Asymptotic properties

How $p(n)$ behaves as n increases has been studied extensively. One of the greatest developments in the theory of integer partitions during the twentieth century was the development of the formula

$$p(n) = \frac{1}{\pi\sqrt{2}} \sum_{k=1}^{\infty} A_k(n)\sqrt{k}\Big[\frac{d}{dx}\frac{sinh(\frac{\pi}{k}\sqrt{\frac{2}{3}(x-\frac{1}{24})})}{\sqrt{x-\frac{1}{24}}}\Big]_{x=n}$$

It is presented here because it's interesting in itself and has no direct bearing on later parts. A more complete account of this formula can be found here[2].

Algorithms for generating partitions behave differently for different orders of n. The impact of the asymptotic properties of $p(n)$ and other functions on these is not obvious.

6 The partition function $p(n, k)$

Define $p(n, k)$ to be the number of partitions of n whose largest part is k. With $k = n$, $p(n, k)$ counts the number of partitions of n that has a part of size n, so $p(n, n) = 1$. With $k = 1$, $p(n, k)$ counts the number of partitions of n whose largest part is 1. There's only one way to partition an integer n into parts of 1's, so $p(n, 1) = 1$.

A partition of n with only 1's is the conjugate of a partition of n with a single part, so the partition identity related to conjugate partitions also shows that

$$p(n, n) = p(n, 1).$$

To find the number $p(n, k)$ for any positive integers n and k it would be ideal to have a formula for the function. It turns out that formulas for small k are possible to derive, but as k increases the formulas become increasingly complex[3].

Another way of computing $p(n, k)$ is to find a recurrence relation. Consider a partition counted by $p(n, k)$. It has at least one part of size k, and this is the largest part. Either it has exactly one part of size k, or it has more than one part of size k. All partitions of n with exactly one part of size k are counted by $p(n-1, k-1)$. This can be seen by observing that a partition counted by $p(n-1, k-1)$ has at least one part of size $k-1$. Adding 1 to one of these parts results in a partition of n with exactly one part of size k. Partitions of n

with more than one part of size k are counted by $p(n-k,k)$. This leads to the recurrence relation

$$\begin{aligned} p(n,k) &= p(n \mid \text{exactly one part of size } k) + \\ &\quad p(n \mid \text{more than one part of size } k) \\ &= p(n-1,k-1) + p(n-k,k). \end{aligned} \tag{12}$$

7 Ordering partitions

A set can be ordered by a binary relation that indicates if one element precedes another. The set of positive integers are ordered by the relation 'less than'

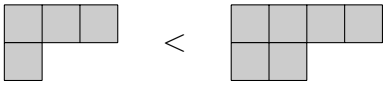
$$1 < 2 < 3 < ...$$

Since for any two different integers one is less than the other, and the relation is transitive, the set of positive integers with the relation 'less than' form a totally ordered set. The relation is called a total order.

A set and a relation where not all the elements in the set are ordered by the relation is called a partially ordered set. The relation is then called a partial order.

7.1 Partial orders

There are many different partial orderings of integer partitions. A relation called the usual order can be defined using Ferrers boards. A partition is said to precede another, denoted by '$<$', if its Ferrers board can be contained in the Ferrers board of the other. For example, the partition $3+1$ of 4 precedes the partition $4+2$ of 6 under the usual order, illustrated below.



The usual order is a partial order. In particular, no two different partitions of the same integer n are related.

A partial order can be visualized with its Hasse diagram. The Hasse diagram of a partially ordered set is constructed by placing the elements of the set in a plane, with greater elements above smaller elements. Then lines are drawn between elements that are directly related. That is, for a set S with elements $x, y \in S$, a line is drawn between them if $x < y$ and there's no element $z \in S$ such that $x < z < y$. This means that there's a path, going either up or down, from any element to all comparable elements. The Hasse diagram of the usual order for all partitions of integers less than or equal to four is depicted in Figure 7.

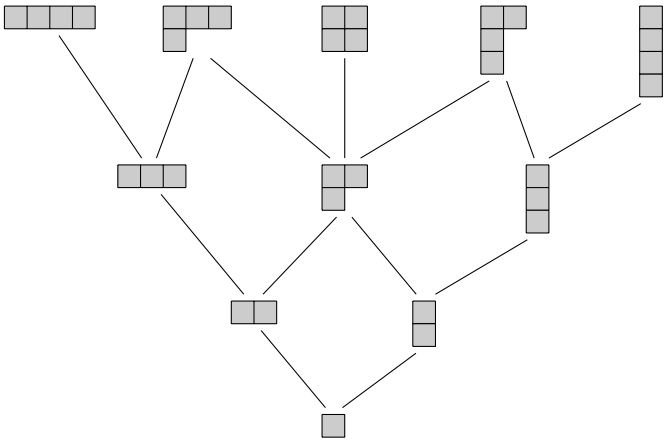


Figure 7: The Hasse diagram of the usual order for partitions of integers less than or equal to four.

7.2 Total orders

The total ordering of integers defined by 'less than' is intuitive. Not all sets can be ordered as easily. Are there total orders for integer partitions?

Define a relation as follows. Let π and λ be any two different integer partitions. If π is a partition of n and λ is a partition of m and $n < m$, then define $\pi < \lambda$. Otherwise, if $n = m$, compare the largest part in π with the largest part in λ. If they are equal, proceed to compare the next largest part and further, until the parts differ. Then if the part in λ is larger than the part being compared to in π, define $\pi < \lambda$. Otherwise define $\lambda < \pi$.

To show that the relation is a total order, let π and λ be two different integer partitions. If they are partitions of different integers, they are ordered as the corresponding integers under 'less than', which is a total order.

Otherwise, if they are partitions of the same integer, then they differ at least for some part. This guarantees that all differing partitions are related, but the relation must also be transitive.

Ordering the parts descending by size, let i be the index of the first part where they differ. Let μ be another integer partition and assume $\pi < \lambda$, $\lambda < \mu$. Further, let j be the index of the first part where λ and μ differ. There are three cases to consider. Either (1) $j < i$, (2) $j = i$ or (3) $j > i$. Denote part i of a partition π by π_i.

1. $j < i$. For all $k < i$, we have $\pi_k = \lambda_k$. Then $\pi_j = \lambda_j < \mu_j \implies \pi < \mu$.
2. $j = i$. Since $\pi_i < \lambda_i$, we have $\pi_j < \lambda_j < \mu_j \implies \pi < \mu$.
3. $j > i$. Then $\pi_i < \lambda_i = \mu_i \implies \pi < \mu$.

This shows that the relation is transitive. The total order defined above is called lexicographic order, sometimes also called dictionary order.

With a total order for a set, each element has a position, or an index, according

to the ordering. For the set of partitions and the lexicographic order, the index for a partition can be computed by counting the number of partitions with smaller parts. If partitions of smaller integers are not considered in the ordering, then the index of a partition π of n with parts $\{r_1,\ r_2,\ ...,\ r_m\}$ in descending order can be computed with the formula

$$i_\pi = f(n, r_1 - 1) + f(n - r_1, r_2 - 1) + f(n - (r_1 + r_2), r_3 - 1) + ... + f(r_m, r_m - 1),$$

where $f(n, k)$ is the number of partitions of n with parts less than or equal to k. Here, the partition of n with only 1's receives the index 0, and the partition of n with a single part receives the index $f(n, n-1) = p(n) - 1$. This works since partitions that precede π in the order have parts that are less than or equal to corresponding parts in $\{r_1,\ r_2,\ ...,\ r_m\}$. For example, the term $f(n, r_1 - 1)$ counts all partitions of n that precede π by having a smaller part at the first position.

8 Random partitions

Is it possible to generate partitions so that each partition has an equal chance of being generated? That is, for a given $n \geq 1$, is there a method to select a partition of n so that each partition has a priori probability $\frac{1}{p(n)}$ of being selected? One suggestion is to generate all partitions of n and then select one at random. However, considering how rapidly the partition function $p(n)$ grows, this is impractical for larger n.

8.1 Methods to generate partitions

There are many ways of generating partitions by random processes. One simple method is to "slice" n into smaller pieces.

Algorithm 1.

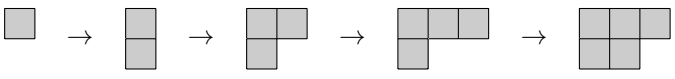
1. Starting with n, take a part $1 \leq m \leq n$ of n where m is selected uniformly at random from the interval $[1, n]$.

2. If $n - m > 1$, set $n = n - m$ and go to (1), otherwise stop.

Another method is to make use of Ferrers boards.

Algorithm 2.
Starting with an empty board, adding squares at inner corners will always result in a new Ferrers board. A Ferrers board for any integer n can then be constructed by adding n squares successively at randomly chosen inner corners.

An example for how a partition might be generated by algorithm 2 is illustrated below.



Are these methods selecting partitions at random?
The methods just suggested are able to generate all partitions for any integer

n, and do so unpredictably. Unfortunately, they do not generate all partitions with equal probability. To see this, we need only to consider any particular type of partition. For example, consider those partitions consisting of a single part.

In algorithm 1, the probability of generating a partition of n with a single part is easy to compute. The only way to end up with only one part is to choose $m = n$ in the first loop, and the process immediately stops. The probability of this occurring is $\frac{1}{n}$, which for $n > 3$ is greater than $\frac{1}{p(n)}$.

In algorithm 2, the probability of generating a partition of n with a single part can be computed as follows. The only way to end up with only one part is to add squares at the top row at every step. The number of positions to add squares to determines the probability of this occurring. The first square must always be placed at the top row. Subsequent squares have two positions where they can be added: the top row, or the second row. The probability of adding all squares on the top row is then $\frac{1}{2^{n-1}}$.

In the section that looked into upper bounds for the partition function $p(n)$ it was concluded that $p(n) < 2^{n-1}$, for $n \geq 3$. For those n, partitions with a single part are generated with smaller probability than $\frac{1}{p(n)}$ using algorithm 2.

A proof says that something is but doesn't always say why that is. It isn't obvious why algorithm 2 doesn't generate different partitions with equal probability. To better see why, consider the Hasse diagram of the usual order, in Figure 7. Generating a partition of n with algorithm 2 is equivalent to taking a random path from the bottom node in the diagram to a node $n-1$ steps up. The path is selected at random but, since some nodes have more paths leading to them than others, the resulting partition is not.

Whether there's a simple algorithm to generate random partitions that doesn't require computationally expensive functions such as the partition function $p(n)$ or the related function $p(n,k)$ is an open question. Using these functions, the problem of generating random partitions becomes much easier to solve.

8.2 Methods to generate random partitions

8.2.1 An algorithm that uses $p(n,k)$

One way to select a partition of n uniformly at random is by first determining the largest part k of the partition and then select a partition of $n-k$, with parts less than or equal to k.

Let $p(n,k)$ denote the number of partitions of n with largest part equal to k. Then for any $k \in [1,n]$, the probability of selecting a partition of n with largest part k must equal $\frac{p(n,k)}{p(n)}$. For subsequent parts, only partitions with parts less than or equal to k are to be considered. So for example, having selected the largest part r, the next part $k \in [1,r]$ is selected according to the probabilities

$$Prob(k) = \frac{p(n,k)}{\sum_{i=1}^{r} p(n,i)}.$$

It is possible to compute the denominator in an alternative way. Any partition counted by $p(n,k)$ can be transformed to a unique partition of $n-k$ by removing a part of size k. The partition obtained has parts less than or equal to k which

shows that

$$p(n,k) = p(n-k \mid \text{parts less than or equal to } k).$$

This leads to the identity

$$\sum_{i=1}^{k} p(n,i) = p(n \mid \text{parts less than or equal to } k) = p(n+k,k).$$

With this an algorithm to generate random partitions can be constructed.

Algorithm 3.

1. Set $r = n$.
2. For $k \in [1, r]$, choose k with probability $\frac{p(n,k)}{p(n+r,r)}$.
3. If $n - k > 1$, set $n = n - k$ and $r = k$ and go to (2), otherwise stop.

In each loop, the selected k is added as a part to the partition being generated. The function $p(n,k)$ can be computed using the recurrence formula (12).

While the algorithm above works, it has the downside of requiring the computation of $p(n,k)$ for a variety of values of n and k. It is also cumbersome to store these values in a large matrix or in some other data structure, to prevent the same values from being evaluated multiple times. Next a more efficient algorithm is described that avoids using $p(n,k)$, developed by Nijenhuis and Wilf[6].

8.2.2 An algorithm that avoids using $p(n,k)$

Let $\sigma(n)$ denote the sum of the divisors of n. An identity of Euler asserts that

$$np(n) = \sum_{m<n} \sigma(n-m)p(m). \tag{13}$$

It can be derived from the generating function

$$\sum_{j=1}^{\infty} (1-x^j)^{-1} = \prod_{n=0}^{\infty} p(n)x^n$$

by first taking the logarithm on both sides and then differentiate with respect to x. Understanding the identity (2.1) combinatorially is useful when developing algorithms so a combinatorial proof is presented.

Let π denote a partition of some integer $m < n$ and let d be a divisor of $n - m$. With each pair (π, d) there's a corresponding partition of n. By appending $\frac{n-m}{d}$ copies of d to the partition π of m, a partition π' of n is obtained.

Suppose d copies of π' is made and that this is done for all partitions of m, for all $m \in [0, n[$ and all divisors d of $n - m$. If we can show that each partition of n is created n times during this process, this proves (2.1).
Consider any particular partition π' of n:

$$\pi' : n = \mu_1 r_1 + \mu_2 r_2 + ... + \mu_k r_k$$

where r_i are the distinct parts of π' and μ_i are their multiplicities. For any t, $1 \leq t \leq \mu_i$, $1 \leq i \leq k$, π' can be constructed by appending t copies of r_i to a partition of $n - tr_i$. Here r_i corresponds to d, and $n - tr_i$ to m above. Each value of t yields one copy of π' and doing this for all t in the range and copying each constructed partition r_i times we get:

$$\sum_{i=1}^{k} r_i \sum_{1}^{\mu_i} 1 = \sum_{i=1}^{k} r_i \mu_i = n,$$

copies of π', as required.

The identity (2.1) can be interpreted as a probability mass function of m. Since (2.1) can be rewritten as

$$1 = \frac{\sum_{m<n} \sigma(n-m)p(m)}{np(n)},$$

we have

$$Prob(m) = \frac{\sigma(n-m)p(m)}{np(n)}.$$

Once m is selected, d is selected according to the probabilities

$$Prob(d) = \frac{d}{\sigma(n-m)}.$$

This is sufficient to construct an algorithm but can be further simplified as follows. Let $p(k) = 0$ for $k < 0$, then

$$\begin{aligned} np(n) &= \sum_{m=1}^{\infty} \sigma(n-m)p(m) \\ &= \sum_{m=1}^{\infty} \sum_{d|m} dp(n-m) \\ &= \sum_{j=1}^{\infty} \sum_{d=1}^{\infty} dp(n-jd). \end{aligned}$$

From this, a function for selecting d and j is obtained.

$$Prob(d, j) = \frac{dp(n-jd)}{np(n)}.$$

Now for the algorithm.

Algorithm 4.

1. Set $P =$ empty partition and $m = n$.
2. For each d and j, $d \geq 1$, $j \geq 1$, $jd \leq m$, choose d and j with probability $\frac{dp(m-jd)}{mp(m)}$.
3. Append j copies of d to P.
4. Set $m = m - jd$.
5. If $m > 0$, go to (2), otherwise stop.

8.2.3 Algorithms that don't use partition functions

One of the goals of this thesis is to find a simple and efficient algorithm for generating random partitions. However, algorithms capable of generating random partitions seems to almost inevitably require values of partition functions to compute the necessary probabilities. This isn't the case if some implicit assumptions are alleviated. One assumption that has been made is that every run of the algorithm must predictably generate a partition. Allowing for runs to fail may decrease worst case performance but possibly increase performance in the average case.

A secondary goal is to explore different algorithms to learn more about integer partitions. Therefore, any algorithm capable of generating partitions at random is of interest. While we were unable to find an efficient algorithm of the sort described above, we did find a simple algorithm that doesn't rely on expensive functions. A brief interlude is made before describing the algorithm, that considers a way to select partitions at random.

In interpreting the problems associated with algorithm 2, we saw that it generates some partitions with greater frequency than others because there are varying number of paths to different partitions. Selecting a path at random would work if there were an equal number of paths to each partition. A solution to this is to consider only a single path to each partition. For example, in the Hasse diagram in Figure 7, one could consider only the leftmost path leading to each partition.

In generating partitions, this can be done by constructing a partition with greater parts first, or equivalently, from top to bottom in a Ferrers board. However, doing this one square at a time, the probabilities to either add a square to the current part or to a new part depends on the number of leftmost paths from the resulting partitions, which is obtained from the partition function $p(n, k)$.

While the above argument doesn't provide a way to avoid partition functions, it does present a viewpoint for selection at random. In section 5 we saw that a composition of a positive integer n can be determined by selecting ',' or '+', $n - 1$ times. In a Hasse diagram for compositions under the usual order, there are a varying number of paths to different compositions. Selecting '+' or ',' successively corresponds to selecting a path at random when only a single path to each composition is considered.

Constructing a composition this way can be described with the steps: Set current row to the top row and add a square to the current row. At each step either add a square to the current row ('+' is selected) or set the current row to the row below (',' is selected).

This algorithm can then be used to generate partitions. If a valid Ferrers board is generated, then a partition has been selected at random. Otherwise if an invalid Ferrers board is generated, start over. For generating partitions it isn't efficient, but can be improved upon and also shows that there are simple methods for generating random partitions. The algorithm can be summarized concretely.

Algorithm 5.

1. Start with an empty partition P and set the current row to the top row.
2. Add a square to the current row.
3. If the current row has more squares than the row above, then go to (1).
4. With probability 1/2, increase current row by 1.
5. If n squares have been added, then stop, otherwise go to (2).

For small n the average performance is similar to algorithms 3 and 4. Then for larger n the performance decreases, since the number of compositions grows more rapidly than $p(n)$.

There are two possibilities for improving the algorithm. First, the probabilities can be assigned differently to prevent invalid Ferrers boards from being generated. Using the partition function $p(n, k)$ this can be done exactly, but is costly. There might be approximate methods for doing this. A second possibility is to transform invalid Ferrers boards to valid partitions. However, this seems difficult, since it depends on the difference between the number of partitions $p(n)$ and the number of compositions.

8.3 Tests

Running a test for the algorithms described above, generating integer partitions of $n = 6$, the results in Table 1 were obtained. Each algorithm was executed 10^6 times and the relative frequency of each partition was recorded. With $p(6) = 11$, the expected frequency is approximately 0.0909 for selection that is uniform at random. The code is written in C++ and is available in appendix A.

Partition	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5
1+1+1+1+1+1	0.0014	0.0311	0.0908	0.0910	0.0907
2+1+1+1+1	0.0209	0.0845	0.0907	0.0910	0.0912
2+2+1+1	0.0626	0.1134	0.0909	0.0909	0.0907
2+2+2	0.0209	0.0602	0.0908	0.0909	0.0911
3+1+1+1	0.0556	0.1181	0.0909	0.0909	0.0911
3+2+1	0.1666	0.1851	0.0909	0.0909	0.0913
3+3	0.0555	0.0603	0.0909	0.0909	0.0905
4+1+1	0.1250	0.1180	0.0910	0.0907	0.0902
4+2	0.1249	0.1136	0.0909	0.0909	0.0908
5+1	0.2002	0.0846	0.0911	0.0911	0.0909
6	0.1666	0.0311	0.0910	0.0909	0.0916

Table 1: Relative frequencies of integer partitions of 6.

The relative frequencies for algorithm 3, 4 and 5 conform well to the expected frequency. The relative frequency for partitions of a single part, here a single part of 6, for algorithm 1 and algorithm 2 also agree with their expected values. For algorithm 1 the expected value is $\frac{1}{n} = \frac{1}{6} \approx 0.1667$ and for algorithm 2 it is $\frac{1}{2^{n-1}} = \frac{1}{2^5} \approx 0.313$. The results also suggest that algorithm 2 generates the conjugate of a partition with equal probability.

In doing the tests, once a partition has been generated its frequency needs to be updated. It is practical to have these frequencies in an array, and to index them with the partition. This can be achieved using the lexicographic order. Since it is a total order, it is a bijection between the indices and the partitions, meaning there's an inverse. After selecting a random index in the range $[1, p(n)]$, a partition can then be constructed using the inverse of the lexicographic order, and this would select a partition uniform at random. Unfortunately, since the lexicographic order uses $p(n, k)$ to generate indices, this wouldn't yield a more efficient algorithm. However, any total order would work so it could be a way to find a better algorithm.

9 Conclusion

We saw that the simple intuitive methods for generating partitions, in algorithm 1 and algorithm 2, fails to generate partitions at random. The methods that rely on partition functions to compute probabilities, in algorithm 3 and algorithm 4, successfully generates random partitions but are more costly to run. In algorithm 5 we saw that if algorithms are allowed to be unreliable in generating partitions, then there are simple solutions.

It is difficult to find simple algorithms that predictably generates partitions at random and that doesn't rely on partition functions. It might be possible to prove that there are no algorithms of this sort. Considering how closely the problem of selecting random partitions is tied to the number of partitions, this difficulty could be related to the problem of finding a simple formula to the partition function $p(n)$.

There are many directions left to explore. An interesting area that hasn't been studied in depth here is the distribution of partitions generated by algorithms that don't generate partitions with equal probability. It should also be noted that more efficient algorithms for generating random partitions have been developed, but have not been included here due to their complexity[4][5].

10 References

[1] George E. Andrews. Euler's pentagonal number theorem. *Mathematics Magazine*, 56(5):279–284, 1983.

[2] George E. Andrews. *The Theory of Partitions.* Cambridge University Press, 2003.

[3] George E. Andrews and Kimmo Eriksson. *Integer Partitions.* Cambridge University Press, 2004.

[4] Richard Arratia and Stephen Desalvo. Probabilistic divide-and-conquer: A new exact simulation method, with integer partitions as an example. *Combinatorics Probability and Computing*, 24(3):324–351, 2016.

[5] Bert Fristedt. The structure of random partitions of large integers. *Transactions of the American Mathematical Society*, 337(2):703–735, 1993.

[6] Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms*. University of Pennsylvania, 1978.

[7] Herbert S. Wilf. *Lectures on Integer Partitions*. University of Pennsylvania, 2000.

A Code for tests

```cpp
#include <iostream>
#include <vector>
#include <string>

int pentagonal(int n) {
  return (3 * n * n - n) / 2;
}

int generalized_pentagonal(int n) {
  int n2 = n / 2;
  return (3 * n2 * n2 + ((n%2)? 1: -1) * n2) / 2;
}

std::vector<int> generate_pn(int n) {
  std::vector<int> p(n + 1);
  p[0] = 1;

  if (n == 0)
    return p;

  p[1] = 1;

  for (int i = 2; i < n+1; i++) {
    p[i] = 0;
    for (int j = 0; j < n; j++) {
      int pent = generalized_pentagonal(j+2);
      if (i - pent < 0)
        break;
      p[i] += (((j / 2) % 2) ? -1 : 1) * p[i - pent];
    }
  }
  return p;
}

int pnk(int n, int k) {
  if (n < k || n < 0 || (n > 0 && k == 0))
    return 0;
  if (n == k || (n > 0 && k == 1) || (n == 0 && k == 0))
    return 1;
  return pnk(n - 1, k - 1) + pnk(n - k, k);
}

int pn_leqk(int n, int k) {
  if (k <= 0) return 0;
  return pnk(n, k) + pn_leqk(n, k - 1);
}

int index_mul(int n, const std::vector<int>& P) {
  int rank = 0, prev_parts = 0;
  for (int ri = P.size() - 1; ri >= 0; ri--) {
    for (int k = 0; k < P[ri]; k++) {
      rank += pn_leqk(n - prev_parts, ri - 1);
      prev_parts += ri;
    }
  }
  return rank;
}

const std::vector<int> partition_from_index(int n, int index) {
  std::vector<int> P(n + 1, 0);
```

```
  int N = n;
  n = pn_leqk(N, N) - index - 1;

  int k, p;
  while (N > 0) {
    for (k = 0; k < N; ++k) {
      p = pn_leqk(N, k);
      if (p > n) break;
    }
    P[k]++;
    N -= k;
    n -= pn_leqk(N, k - 1);
  }

  return P;
}

std::vector<int> algorithm1(int n) {
  std::vector<int> P(n + 1, 0);
  if (n == 0)
    return P;

  int r = n;
  while (r > 0) {
    int part = 1 + rand() % r;
    P[part]++;
    r -= part;
  }

  return P;
}

std::vector<int> algorithm2(int n) {
  std::vector<int> P(n + 1, 0);
  if (n == 0)
    return P;

  std::vector<int> parts(n,0);
  parts[0]++;
  std::vector<int> valid_rows;
  for (int i = 0; i < n-1; i++) {
    valid_rows.clear();
    valid_rows.emplace_back(0);
    for (unsigned int row = 1; row < parts.size(); row++) {
      if (parts[row] < parts[row - 1]) {
        valid_rows.emplace_back(row);
      }
    }
    int add_to = rand() % valid_rows.size();
    parts[valid_rows[add_to]]++;
  }

  for (int part : parts)
    P[part]++;
  return P;
}

std::vector<int> algorithm3(int n, int r) {
  std::vector<int> P(n + 1, 0);
  if (n == 0)
    return P;

```

```
  double sum = 0;
  double roll =  (double)rand() / (double)RAND_MAX;
  int k = 1;
  for (; k <= r; k++) {
    //sum += (double)pnk(n, k) / (double)pn_leqk(n, r);
    sum += (double)pnk(n, k) / (double)pnk(n + r, r);
    if (sum >= roll)
      break;
  }
  P[k]++;

  std::vector<int> sub = algorithm3(n - k, std::min(n, k));
  for (unsigned int i = 0; i < sub.size(); i++) {
    P[i] += sub[i];
  }

  return P;
}

std::vector<int> algorithm4(int n, const std::vector<int>& p) {
  std::vector<int> P(n + 1, 0);
  int m = n;
  do {
    double r = (double)rand() / (double)RAND_MAX;
    double sum = 0;
    int j, d;
    for (j = 1;; j++) {
      for (d = 1; j * d <= m; d++) {
        sum += ((double)d * (double)p[m - j * d]) / ((double)m * (
    double)p[m]);
        if (sum >= r) {
          break;
        }
      }
      if (sum >= r) {
        break;
      }
    }
    P[d] += j;
    m -= j * d;

  } while (m > 0);

  return P;
}

std::vector<int> algorithm5(int n) {
  std::vector<int> parts(n, 0);
  if (n == 0)
    return std::vector<int>(1, 0);

  int row = 0;
  parts[row] = 1;
  for (int i = 1, j = 0; i < n; i++, j++) {
    double r = (double)rand() / (double)RAND_MAX;
    if (r > 0.5)
      row++;

    if (row && parts[row] >= parts[row - 1]) {
      return std::vector<int>(0);
    }
    parts[row]++;
```

```
184   }
185   std::vector<int> P(n + 1, 0);
186   for (int part : parts)
187     P[part]++;
188   return P;
189 }
190 
191 int main() {
192   srand(time(NULL));
193 
194   int n = 6;
195   std::vector<int> p = generate_pn(n);
196 
197   bool print_pn = false;
198   if (print_pn) {
199     for (int i = 0; i < n; i++) {
200       std::cout << "p(" << i << ") = " << p[i] << std::endl;
201     }
202     std::cout << std::endl;
203   }
204   std::vector<int> frequencies(p[n]);
205   //int runs = 10;
206   int runs = 1000000;
207   //int runs = 10000000;
208 
209   int failed_runs = 0;
210   for (int k = 0; k < runs; k++) {
211     //std::vector<int> P = algorithm1(n);
212     //std::vector<int> P = algorithm2(n);
213     std::vector<int> P = algorithm3(n, n);
214     //std::vector<int> P = algorithm4(n, p);
215     //std::vector<int> P = algorithm5(n);
216 
217     if (P.size()) {
218       int sum = 0;
219       for (unsigned int i = 0; i < P.size(); i++) {
220         sum += i * P[i];
221       }
222       if (sum != n)
223         std::cerr << "invalid partition!\n";
224 
225       frequencies[index_mul(n, P)]++;
226     }
227     else {
228       k--;
229       failed_runs++;
230     }
231   }
232 
233   std::cout << "runs: " << runs << ", failed runs:" << failed_runs
      << std::endl;
234   std::cout << std::endl;
235   std::cout << "frequencies:\n";
236   for (unsigned int i = 0; i < frequencies.size(); i++) {
237     std::cout << i << ", " << (double)frequencies[i] / (double)runs
       << std::endl;
238   }
239   std::cout << std::endl;
240   std::cout << "1/p(n) = " << 1.0 / (double)p[n] << std::endl;
241 
242   bool print_samples = false;
243   if (print_samples) {
```

```
244     std::cout << std::endl;
245     std::cout << "samples:\n";
246     for (int k = 0; k < 10; k++) {
247       std::vector<int> P = algorithm4(n, p);
248       std::vector<int> parts;
249       for (unsigned int i = 1; i < P.size(); i++) {
250         for (int j = 0; j < P[i]; j++)
251           parts.push_back(i);
252       }
253       std::cout << n << " = ";
254       for (unsigned int i = 0; i < parts.size(); i++) {
255         std::cout << parts[parts.size() - 1 - i] << (i + 1 < parts.
    size() ? " + " : "");
256       }
257       std::cout << std::endl << "rank:" << index_mul(n, P) << std::
    endl;
258       std::cout << std::endl;
259     }
260   }
261   return 0;
262 }
```