A generating function for restricted partitions

Robin K. S. Hankin
Auckland University of Technology

Abstract

A generating function for restricted partitions (originally due, as far as I can tell, to Wilf (2000)) is presented and R idiom using the spray package given. The generating function approach is shown to be not particularly efficient compared to the direct enumeration used in the partitions package.

Keywords: Restricted partitions, generating function, R.

1. Introduction

The partitions package gives functionality for various integer partition enumeration problems including that of restricted partitions, function restrictedparts():

```r
> library("partitions")

> jj <- restrictedparts(7,3)

[1,] 7 6 5 4 5 4 3 3
[2,] 0 1 2 3 1 2 3 2
[3,] 0 0 0 0 1 1 1 2

> ncol(jj)

[1] 8
```

Here I will consider function \texttt{R()}, which calculates the size of the matrix required:

```r
> R(3,7,include.zero=TRUE)

[1] 8
```

Function \texttt{R()} is very basic; all it does is to go through all the restricted partitions, counting them one by one until the recursion bottoms out:

```c
unsigned int numrestrictedparts(int *x, const int m){
    unsigned int count=1;
```
A generating function for restricted partitions

while(c_nextrestrictedpart(x, &m)==0){
    count++;
} return count;

To implement a potentially more efficient method, we can use generating functions. Here we follow Wilf and, using his terminology, define an infinite polynomial \( P(x, y) \) as follows:

\[
P(x, y) = \prod_{r=0}^{\infty} \frac{1}{1 - x^r y}
\]  

(1)

Or, expanding:

\[
P(x, y) = (1 + y + y^2 + y^3 + \cdots)(1 + xy + x^2 y^2 + x^3 y^3 + \cdots)(1 + x^r y + x^{2r} y^2 + x^{3r} y^3 + \cdots) \cdots
\]  

(2)

The power of \( x \) counts the total of the chosen integers (the size of the partition), and the power of \( y \) counts the number of integers chosen (the length of the partition). Thus the number of partitions of \( k \) into at most \( n \) parts is the coefficient of \( x^k y^n \) in \( P(x, y) \).

In numerical work it is convenient and efficient to ignore terms with a power of \( x \) higher than \( n \) (sum of integers chosen exceeds \( n \)), or with power of \( y \) higher than \( k \) (number of integers chosen exceeds \( k \))

Taking \( R(3,7,\text{include.zero}=\text{TRUE}) \) as an example we would truncate equation (2) as follows:

\[
P(x, y) = \left(1 + y + y^2 + y^3\right) \left(1 + xy + x^2 y^2 + x^3 y^3\right) \cdots \left(1 + x^r y + x^{2r} y^2 + x^{3r} y^3\right) \cdots
\]  

(3)

and the coefficients of \( P(x, y) \) up to \( x^7 y^3 \) would correctly count the restricted partitions.

Note that we need consider only at most four terms in each bracket (powers of \( y \) above three being irrelevant) and we may stop the continued product at the \( x^7 \) term as further brackets contain only one and powers of \( x \) above the eighth.

The R implementation uses the \texttt{spray} package, in particular function \texttt{ooom(x)} which returns \( \frac{1}{1-x} \).

```r
> library("spray")
> R_gf <- function(k,n){ # version 1
+     x <- spray(cbind(1,0))
+     y <- spray(cbind(0,1))
+     P <- ooom(y,k) # term x^0; number of zeros chosen
+     for(i in seq_len(k)){ # starts at 1
+         P <- P*ooom(x^i*y,n)
+     }
+     return(value(P[k,n]))
+ }
```
Thus

\[ R_\text{gf}(7,3) \]

\[ \{1\} \ 8 \]

We can do slightly better in terms of efficiency by ruthlessly cutting out powers higher than needed:

\[\text{strip} \leftarrow \text{function}(P,k,n) \{ \ # \text{strips out powers higher than needed} \]
\[ + \ \text{ind} \leftarrow \text{index}(P) \]
\[ + \ \text{val} \leftarrow \text{value}(P) \]
\[ + \ \text{wanted} \leftarrow (\text{ind}[1,1] \leq k) \ \& \ (\text{ind}[1,2] \leq n) \]
\[ + \ \text{spray} (\text{ind}[\text{wanted},:], \text{val}[\text{wanted}]) \]
\[ \} \]

which is used here:

\[ R_\text{gf2} \leftarrow \text{function}(k,n,\text{give\_poly}=\text{FALSE}) \{ \]
\[ + \ x \leftarrow \text{spray}(\text{cbind}(x=1,y=0)) \]
\[ + \ y \leftarrow \text{spray}(\text{cbind}(x=0,y=1)) \]
\[ + \ P \leftarrow \text{oom}(y,k) \ # \text{term } x^0 \]
\[ + \ \text{for}(i \text{ in seq\_len}(k)) \{ \ # \text{starts at } 1 \]
\[ + \ \ P \leftarrow \text{strip}(P*\text{oom}(\text{spray}(\text{cbind}(i,0))*y, \ \text{min}(n,\text{ceiling}(k/i))),k,n) \]
\[ + \} \]
\[ + \ \text{if}(\text{give\_poly}) \{ \]
\[ + \ \text{return}(P) \]
\[ + \} \ \text{else} \{ \]
\[ + \ \text{return}(\text{value}(P[k,n])) \]
\[ + \} \]

then

\[ R_\text{gf2}(7,3) \]

\[ \{1\} \ 8 \]

2. Computational efficiency

We can test the computational efficiency of the generating function approach using larger values of \( k \) and \( n \):

\[ k \leftarrow 140 \]
\[ n \leftarrow 4 \]
A generating function for restricted partitions

```r
> system.time(jj1 <- R(n,k,include.zero=TRUE))

  user  system elapsed
  0     0       0

> system.time(jj2 <- R_gf2(k,n))

  user  system elapsed
  0.897  0.008   0.905

> jj1==jj2

[1] TRUE
```

So the generating function approach is not particularly efficient, at least not in this sort of use-case with the `spray` package. It might be better with the `skimpy` package; I don’t know. Of course, `R_gf2()` calculates the generating polynomial which gives very much more information than is returned. Perhaps this is why it is so slow compared to function `R()`, although it is surprising to see direct enumeration so heavily outperforming a generating function.

References


Affiliation:

Robin K. S. Hankin
Auckland University of Technology
AUT Tower
Wakefield Street
Auckland, New Zealand
E-mail: hankin.robin@gmail.com