Package ‘partitions’

February 20, 2015

Type Package
Title Additive partitions of integers
Version 1.9-15
Date 2009-04-28
Author Robin K. S. Hankin
Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>
Imports gmp, polynom
Description Additive partitions of integers. Enumerates the partitions, unequal partitions, and restricted partitions of an integer; the three corresponding partition functions are also given. Set partitions are now included.
License GPL
NeedsCompilation yes
Repository CRAN
Date/Publication 2013-12-08 20:35:28

R topics documented:

partitions-package .................................................. 2
as.matrix.partition ............................................... 3
bin ................................................................. 3
conjugate ......................................................... 4
nextpart ........................................................... 5
P ................................................................. 7
parts .............................................................. 9
perms ........................................................... 11
print.partition .................................................. 12
setparts ......................................................... 13
summary.partition ............................................. 15

Index 17
partitions-package

Integer partitions

Description

Routines to enumerate all partitions of an integer; includes restricted and unequal partitions.

Details

This package comprises eight functions: \( P() \), \( Q() \), \( R() \), and \( S() \) give the number of partitions, unequal partitions, restricted partitions, and block partitions of an integer.

Functions \( \text{parts()} \), \( \text{diffparts()} \), \( \text{restrictedparts()} \), and \( \text{blockparts()} \) enumerate these partitions.

Function \( \text{conjugate()} \) gives the conjugate of a partition and function \( \text{durfee()} \) gives the size of the Durfee square.

NB the emphasis in this package is terse, efficient C code. This means that there is a minimum of argument checking. For example, function \( \text{conjugate()} \) assumes that the partition is in standard form (ie nonincreasing); supplying a vector in nonstandard form will result in garbage being returned silently. Note that a block partition is not necessarily in standard form.

Author(s)

Robin K. S. Hankin

References


Examples

\[
\text{parts(5)}\\
\text{diffparts(9)}\\
\text{restrictedparts(15,10)}\\
\text{P}(10,\text{give=TRUE})\\
\text{Q}(10,\text{give=TRUE})\\
\text{R}(5,10)
\]
as.matrix.partition

Description

Coerce partitions to matrices and vice versa

Usage

## S3 method for class 'partition'
as.matrix(x, ...)
as.partition(x, ...)

Arguments

x
Object to be coerced
...
Further arguments

Author(s)

Robin K. S. Hankin

Examples

as.matrix(parts(5))

bin

Sundry binary functionality

Description

Utilities to convert things to binary

Usage

tobin(n, len, check=TRUE)
todec(bin)
comptobin(comp, check=TRUE)
bintocomp(bin, use.C=TRUE, check=TRUE)
Arguments

- **n**: Integer, to be converted to binary by function `tobin()`
- **len**: Length of the binary vector returned by function `tobin()`
- **bin**: Binary: a vector of 0s and 1s
- **comp**: A composition
- **check**: Boolean, with default TRUE meaning to perform various checks
- **use.C**: Boolean, with default TRUE meaning to use C

Details

These functions are not really intended for the end user; they are used in `nextcomposition()`.

- Function `tobin()` converts integer `n` to a binary string of length `len`
- Function `todec()` converts a binary string to decimal, so `todec(tobin(n,i)) == n`, provided `i` is big enough
- Function `comptobin()` converts a composition to binary
- Function `bintocomp()` converts a binary string to a composition

Author(s)

Robin K. S. Hankin

References


Examples

```r
> tobin(10,5)
[1] 10101
> todec(tobin(10,5))
[1] 10
> comptobin(c(1,1,4))
[1] 1 1 4
> bintocomp(c(1,1,0,0,1,1,1,1))
[1] 1 1 0 0 1 1 1 1
```

Description

Given a partition, provide its conjugate or Durfee square

Usage

```r
conjugate(x)
durfee(x)
```
nextpart

Arguments

- `x` Either a vector describing a partition, in standard form (ie nonincreasing); or a matrix whose columns are partitions in standard form

Details

Conjugation is described in Andrews, and (eg) Hardy and Wright.

Essentially, `conjugate()` carries out R idiom `rev(cumsum(table(factor(a[a>0], levels=max(a):1))))`, but faster.

The “Durfee square” of a partition is defined on page 281 of Hardy and Wright. It is the largest square of nodes contained in the partition’s Ferrers graph. Function `durfee()` returns the length of the side of the Durfee square, which Andrews denotes $d(\lambda)$. It is equivalent to R idiom `function(a){sum(a>=1:length(a))}`, but faster.

Value

Returns either a partition in standard form, or a matrix whose columns are partitions in standard form.

Note

If argument `x` is not nonincreasing, all bets are off: these functions will not work and will silently return garbage. Caveat emptor! (output from `blockparts()` is not necessarily non-increasing)

Author(s)

Robin K. S. Hankin

Examples

```
parts(5)
conjugate(parts(5))

restrictedparts(6,4)
conjugate(restrictedparts(6,4))

durfee(10:1)
```

Description

Given a partition, return the “next” one; or determine whether it is the last one.
Usage

```r
nextpart(part, check=TRUE)
```  
islastpart(part)

```r
firstpart(n)
```  
nextdiffpart(part, check=TRUE)

```r
islastdiffpart(part)
```  
firstdiffpart(n)

```r
nextrestrictedpart(part, check=TRUE)
```  
islastrestrictedpart(part)

```r
firstrestrictedpart(n, m, include.zero=TRUE)
```  
nextblockpart(part, f, n=sum(part), include.fewer=FALSE, check=TRUE)

```r
islastblockpart(part, f, n=NULL , include.fewer=FALSE)
```  
firstblockpart( f, n=NULL , include.fewer=FALSE)

```r
nextcomposition(comp, restricted, include.zero=TRUE, check=TRUE)
```  
islastcomposition(comp, restricted, include.zero=TRUE)

```r
firstcomposition(n, m=NULL , include.zero=TRUE)
```  
Arguments

- `part, comp` A partition or composition
- `check` Boolean, with default TRUE meaning to carry out various safety checks; the `next()` functions use C calls which might crash the session with some inputs
- `f, n, include.fewer, m, include.zero` Other arguments as per the vectorized version
- `restricted` In function `nextcomposition()` and `islastcomposition()`, Boolean, with TRUE meaning to consider compositions of fixed length [eg, to iterate through the columns of `compositions(6,3)`], and FALSE meaning to consider compositions of any length [eg to iterate through the columns of `compositions(6)`]

Details

These functions are intended to enumerate partitions one at a time, eliminating the need to store a huge matrix. This is useful for optimization over large domains and makes it possible to investigate larger partitions than is possible with the vectorized codes.

The idea is to use a `first...()` function to generate the first partition, then iterate using a `next...()` function, stopping when the `islast...()` function returns TRUE.

An example is given below, in which the “scrabble” problem is solved; note the small size of the sample space. More examples are given in the tests/aab.R file.

Note

Functions `nextpart()` and `nextdiffpart()` require a vector of the right length: they require and return a partition padded with zeros. Functions `nextrestrictedpart()` and `nextblockpart()` work with partitions of the specified length. Function `nextcomposition()` truncates any zeros at the end of the composition. This behaviour is inherited from the C code.

In functions `nextcomposition()` and `firstcomposition()`, argument `include.zero` is ignored if `restricted` is FALSE.
I must say that the performance of these functions is terrible; they are much much slower than their vectorized equivalents. The magnitude of the difference is much larger than I expected. Heigh ho. Frankly you would better off working directly in C.

Author(s)
Robin K. S. Hankin

See Also
parts

Examples

```r
# Do the optimization in scrabble vignette, one partition at a time:
# (but with a smaller letter bag)
scrabble <- c(a=9, b=2, c=4, d=4, e=12, f=2, g=3)

f <- function(a){prod(choose(scrabble,a))/choose(sum(scrabble),7)}
bestsofar <- 0
a <- firstblockpart(scrabble,7)
while(!islastpart(a)){
  jj <- f(a)
  if(jj>bestsofar){
    bestsofar <- jj
    bestpart <- a
  }
  a <- nextblockpart(a,scrabble)
}
```

---

\( P \)

**Number of partitions of an integer**

Description

Given an integer, \( P() \) returns the number of additive partitions, \( Q() \) returns the number of unequal partitions, and \( R() \) returns the number of restricted partitions. Function \( S() \) returns the number of block partitions.

Usage

- \( P(n, \text{give} = \text{FALSE}) \)
- \( Q(n, \text{give} = \text{FALSE}) \)
- \( R(m, n, \text{include.zero} = \text{FALSE}) \)
- \( S(f, n = \text{NULL}, \text{include.fewer} = \text{FALSE}) \)
Arguments

- **n**: Integer whose partition number is desired. In function \(S()\), the default of NULL means to return the number of partitions of any size.

- **m**: In function \(R()\), the order of the decomposition.

- **give**: Boolean, with default FALSE meaning to return just \(P(n)\) or \(Q(n)\) and TRUE meaning to return \(P(1:n)\) or \(Q(1:n)\) (this option takes no extra computation).

- **include.zero**: In restrictedparts(), Boolean with default FALSE meaning to count only partitions of \(n\) into exactly \(m\) parts; and TRUE meaning to include partitions of \(n\) into at most \(m\) parts (because parts of zero are included).

- **include.fewer**: In function blockparts(), Boolean with default FALSE meaning to return partitions into exactly \(n\) and TRUE meaning to return partitions into at most \(n\).

- **f**: In function \(S()\), the stack vector.

Details

Functions \(P()\) and \(Q()\) use Euler’s recursion formula. Function \(R()\) enumerates the partitions using Hindenburg’s method (see Andrews) and counts them until the recursion bottoms out.

Function \(S()\) finds the coefficient of \(x^n\) in the generating function \(\prod_{i=1}^{L} \sum_{j=0}^{f_i} x^j\), where \(L\) is the length of \(f\), using the polynom package.

All these functions return a double.

Note

Functions \(P()\) and \(Q()\) use unsigned long long integers, a type which is system-dependent. For me, \(P()\) works for \(n\) equal to or less than 416, and \(Q()\) works for \(n\) less than or equal to 792. YMMV; none of the methods test for overflow, so use with care!

Author(s)

Robin K. S. Hankin; \(S()\) is due to an anonymous JSS referee.

Examples

\[P(10, \text{give=TRUE})\]
\[Q(10, \text{give=TRUE})\]
\[R(10, 20, \text{include.zero=FALSE})\]
\[R(10, 20, \text{include.zero=TRUE})\]
\[S(1:4, 5)\]
**parts**

**Enumerate the partitions of an integer**

**Description**

Given an integer, return a matrix whose columns enumerate various partitions.

Function `parts()` returns the unrestricted partitions; function `diffparts()` returns the unequal partitions; function `restrictedparts()` returns the restricted partitions; function `blockparts()` returns the partitions subject to specified maxima; function `compositions()` returns all compositions of the argument.

**Usage**

```
parts(n)
diffparts(n)
restrictedparts(n, m, include.zero=TRUE, decreasing=TRUE)
blockparts(f, n=NULL, include.fewer=FALSE)
compositions(n, m=NULL, include.zero=TRUE)
```

**Arguments**

- `n` Integer to be partitioned. In function `blockparts()`, the default of `NULL` means to return all partitions of any size.
- `m` In functions `restrictedparts()` and `compositions()`, the order of the partition.
- `include.zero` In functions `restrictedparts()` and `compositions()`, Boolean with default `FALSE` meaning to include only partitions of `n` into *exactly* `m` parts; and `TRUE` meaning to include partitions of `n` into *at most* `m` parts (because zero parts are included).
- `include.fewer` In function `blockparts()`, Boolean with default `FALSE` meaning to return vectors whose sum is *exactly* `n` and `TRUE` meaning to return partitions whose sum is *at most* `n`.
- `decreasing` In restrictedparts(), Boolean with default `TRUE` meaning to return partitions whose parts are in decreasing order and `FALSE` meaning to return partitions in lexicographical order, as appearing in Hindenburg’s algorithm. Note that setting `decreasing` to `FALSE` has the effect of making `conjugate()` return garbage.
- `f` In function `blockparts()`, a vector of strictly positive integers that gives the maximal number of blocks; see details.

**Details**

- Function `parts()` uses the algorithm in Andrews. Function `diffparts()` uses a very similar algorithm that I have not seen elsewhere. These functions behave strangely if given an argument of zero.
• Function `restrictedparts()` uses the algorithm in Andrews, originally due to Hindenburg. For partitions into at most \( m \) parts, the same Hindenburg’s algorithm is used but with a start vector of \( c(\text{rep}(0, m-1), n) \). If \( m > n \), the partitions are padded with zeros.

• Function `blockparts()` enumerates the compositions of an integer subject to a maximum criterion: given vector \( y = (y_1, \ldots, y_n) \) all sets of \( a = (a_1, \ldots, a_n) \) satisfying \( \sum a_i = n \) subject to \( 0 \leq a_i \leq y_i \) for all \( i \) are given in lexicographical order. If argument \( y \) includes zero elements, these are treated consistently (i.e., a position with zero capacity).

If \( n \) takes its default value of `NULL`, then the restriction \( \sum a_i = n \) is relaxed (so that the numbers may sum to anything). Note that these solutions are not necessarily in standard form, so functions `durfee()` and `conjugate()` may fail.

• Function `compositions()` returns all \( 2^{n-1} \) ways of partitioning an integer; thus \( 4+1+1 \) is distinct from \( 1+4+1 \) or \( 1+1+4 \). This function is different from all the others in the package in that it is written in R; it is not clear that C would be any faster.

Note

These vectorized functions return a matrix whose columns are the partitions. If this matrix is too large, consider enumerating the partitions individually using the functionality documented in `nextpart.Rd`.

One commonly encountered idiom is `blockparts(rep(n, n), n)`, which is equivalent to `compositions(n, n)` [Sloane’s A001700).

Author(s)

Robin K. S. Hankin

References


See Also

`nextpart`
perms

Examples

parts(5)
diffparts(10)
restrictedparts(9,4)
restrictedparts(9,4,\text{FALSE})
restrictedparts(9,4,\text{decreasing}=\text{TRUE})

blockparts(1:4)
blockparts(1:4,3)
blockparts(1:4,3,\text{include.fewer}=\text{TRUE})

blockparts(c(4,3,3,2),5) \ # \text{Knuth's example, Fascicle 3a, p16}

compositions(3)

perms \hspace{1cm} \textit{Enumerate the permutations of a vector}

Description

Given an integer \(n\), return a matrix whose columns enumerate various permutations of \(1:n\).

Function \texttt{allperms()} returns all permutations in lexicographic order; function \texttt{plainperms()} returns all permutations by repeatedly exchanging adjacent pairs.

Usage

perms(n)
plainperms(n)

Arguments

\(n\) \hspace{1cm} \text{Integer argument; permutations of } 1:n \text{ returned}

Note

Comments in the \texttt{C} code; algorithm lifted from ‘fasc2b.pdf’.

Author(s)

D. E. Knuth; \texttt{C} and \texttt{R} transliteration by Robin K. S. Hankin

References

See Also

parts

Examples

summary(perms(5))

# Knuth's Figure 18:
matplot(t(apply(plainperms(4), 2, order)),
    type="l", lty=1, lwd=5, asp=1,
    frame=FALSE, axes=FALSE, ylab="", col=gray((1:5)/5))

Arguments

x          Object to be printed: an object of class either partition or summary.partition
mat        Boolean, with TRUE meaning to print like a matrix, and any other value meaning
to print without column names (which usually results in more compact appearance)

h          Boolean governing the orientation of the printed matrix, with TRUE meaning to
print with the rows being the partitions and any other value (the default) meaning
to print the transpose

sep        Character vector, with special value of NULL interpreted as a comma; see examples section

...        Further arguments provided for compatibility
setparts

Author(s)
Robin K. S. Hankin

Examples
print(parts(5))
summary(parts(7))
listParts(3)
options(separator="")
listParts(5)

setparts  Set partitions

Description
Enumeration of set partitions

Usage
setparts(x)
listParts(x)

Arguments
x  If a vector of length 1, the size of the set to be partitioned. If a vector of length
greater than 1, return all equivalence relations with equivalence classes with
sizes of the elements of x. If a matrix, return all equivalence classes with sizes
of the columns of x

Details
A partition of a set \( S = \{1, \ldots, n\} \) is a family of sets \( T_1, \ldots, T_k \) satisfying

- \( i \neq j \implies T_i \cap T_j = \emptyset \)
- \( \bigcup_{i=1}^{k} T_k = S \)
- \( T_i \neq \emptyset \) for \( i = 1, \ldots, k \)

The induced equivalence relation has \( i \sim j \) if and only if \( i \) and \( j \) belong to the same partition.
Equivalence classes may be listed using \texttt{listParts()}.

There are exactly fifteen ways to partition a set of four elements:

\begin{align*}
(1234) \\
(123)(4), (124)(3), (134)(2), (234)(1) \\
(12)(34), (13)(24), (14)(23) \\
(1)(2)(3)(4)
\end{align*}
Note that (12)(3)(4) is the same partition as, for example, (3)(4)(21) as the equivalence relation is the same.

Consider partitions of a set \( S \) of five elements (named 1, 2, 3, 4, 5) with sizes 2,2,1. These may be enumerated as follows:

```r
> u <- c(2,2,1)
> setparts(u)

[1,] 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3
[2,] 2 2 3 1 1 2 2 3 2 2 3 1 1 1 1 1
[3,] 3 2 2 3 2 2 1 1 1 3 2 2 2 1 2 2
[4,] 2 3 2 3 3 2 3 2 1 1 1 2 2 1 1 1
[5,] 1 1 2 2 3 2 3 2 3 2 1 2 2
```

See how each column has two 1s, two 2s and one 3. This is because the first and second classes have size two, and the third has size one.

The first partition, \( x = c(1,2,3,2,1) \), is read “class 1 contains elements 1 and 5 (because the first and fifth element of \( x \) is 1); class 2 contains elements 2 and 4 (because the second and fourth element of \( x \) is 2); and class 3 contains element 3 (because the third element of \( x \) is 3)”. Formally, class \( i \) has elements which(\( x = u[i] \)).

You can change the print method by setting, eg. option(separator=" ").

**Value**

Returns a matrix each of whose columns show a set partition; an object of class "partition". Type ?print.partition to see how to change the options for printing.

**Author(s)**

Luke G. West (C++) and Robin K. S. Hankin (R); listParts() provided by Diana Tichy

**References**


**See Also**

`parts`, `print.partition`

**Examples**

```r
setparts(4)  # all partitions of a set of 4 elements
setparts(c(3,3,2))  # all partitions of a set of 8 elements
```
The function is designed to behave as expected: if there is an argument named “n”, then this is used. If there is no such argument, the first one is used.
Value

A summary object is a list, comprising three elements:

- **shortened** Boolean, with TRUE meaning that the middle section of the matrix is omitted, and FALSE meaning that the entire matrix is returned because n is too big.
- **n** Number of columns to return at the start and the end of the matrix.
- **out** Matrix returned: just the first and last n columns (if shortened is TRUE), or the whole matrix if not.

Author(s)

Robin K. S. Hankin

Examples

- `summary(parts(7))`
- `summary(parts(11),3)`
Index

*Topic **math**
  - as.matrix.partition, 3
  - conjugate, 4
  - nextpart, 5
  - P, 7
  - parts, 9
  - perms, 11
  - print.partition, 12
  - setparts, 13
  - summary.partition, 15

*Topic **package**
  - partitions-package, 2
  - as.matrix.partition, 3
  - as.partition(as.matrix.partition), 3
  - bin, 3
  - bintocomp(bin), 3
  - blockparts(parts), 9
  - compositions(parts), 9
  - comptobin(bin), 3
  - conjugate, 4
  - diffparts(parts), 9
  - Durfee(conjugate), 4
  - durfee(conjugate), 4
  - firstblockpart(nextpart), 5
  - firstcomposition(nextpart), 5
  - firstdiffpart(nextpart), 5
  - firstpart(nextpart), 5
  - firstrestrictedpart(nextpart), 5
  - islastblockpart(nextpart), 5
  - islastcomposition(nextpart), 5
  - islastdiffpart(nextpart), 5
  - islastpart(nextpart), 5
  - islastrestrictedpart(nextpart), 5
  - listParts(setparts), 13
  - nextblockpart(nextpart), 5
  - nextcomposition(nextpart), 5
  - nextdiffpart(nextpart), 5
  - nextpart, 5, 10
  - nextrestrictedpart(nextpart), 5
  - P, 7
  - partitions(partitions-package), 2
  - partitions-package, 2
  - parts, 7, 9, 12, 14
  - perms, 11
  - plainperms(perms), 11
  - print.equivalence(print.partition), 12
  - print.partition, 12, 14
  - print.summary.partition
    (print.partition), 12
  - Q(P), 7
  - R(P), 7
  - restrictedparts(parts), 9
  - S(P), 7
  - setparts, 13
  - summary.partition, 15
  - tobin(bin), 3
  - todec(bin), 3