Package ‘partitions’

October 23, 2021

Type Package

Title Additive Partitions of Integers

Version 1.10-4

Depends R (>= 3.6.0)

Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>

Imports gmp, polynom, sets, mathjaxr

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 and now compositions are included.

Suggests testthat

License GPL

URL https://github.com/RobinHankin/partitions

BugReports https://github.com/RobinHankin/partitions/issues

RdMacros mathjaxr

R topics documented:

partitions-package ................................................. 2
as.matrix.partition ............................................. 3
bin ................................................................. 3
conjugate .......................................................... 4
nextpart ........................................................... 6
P ................................................................. 7
parts ............................................................... 9
perms .............................................................. 13
print.partition .................................................. 14
setparts ......................................................... 15
summary.partition ............................................ 17

Index 19
partitions-package

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

\[
\begin{align*}
\text{parts}(5) \\
\text{diffparts}(9) \\
\text{restrictedparts}(15,10) \\
P(10,\text{give}=\text{TRUE}) \\
Q(10,\text{give}=\text{TRUE}) \\
R(5,10)
\end{align*}
\]
as.matrix.partition  Coerce partitions to matrices and vice versa

Description
Coercion to and from partitions

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 \( \text{len} \)
- Function `todec()` converts a binary string to decimal, so \( \text{todec(tobin(n,i))} = n \), 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)
todec(tobin(10,5))
comptobin(c(1,1,4))
bintocomp(c(1,1,0,0,1,1,1,1))
```

---

### conjugate

*Conjugate partitions and Durfee squares*

**Description**

Given a partition, provide its conjugate or Durfee square

**Usage**

```r
  conjugate(x, sorted = TRUE)
durfee(x, sorted = TRUE)
durfee_sorted(x)
```

**Arguments**

- `x` Either a vector describing a partition or a matrix whose columns are partitions.
- `sorted` A logical indicating whether the data is already in standard form. That is to say, are the data within each column sorted in decreasing order?
**Details**

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

The *conjugate* of a partition may be calculated by taking its Ferrers diagram and considering the partition defined by columns instead of rows. This may be visualised by flipping the Ferrers diagram about the leading diagonal.

Essentially, `conjugate()` carries out R idiom

```r
rev(cumsum(table(factor(a[a>0],levels=max(a):1))))
```

but is 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

```r
function(a){sum(a>=1:length(a))}
```

but is 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 non-increasing, you must use the `sorted = FALSE` flag. Otherwise, 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**

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

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

durfee(10:1)

# A partition in nonstandard form --- use `sorted = FALSE`
x <- parts(5)[sample(5),]
durfee(x, sorted = FALSE)
conjugate(x, sorted = FALSE)

# Suppose one wanted partitions of 8 with no part larger than 3:
conjugate(restrictedparts(8,3))

# (restrictedparts(8,3) splits 8 into at most 3 parts;
# so no part of the conjugate partition is larger than 3).
```
nextpart

Next partition

Description

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

Usage

nextpart(part, check=TRUE)
islastpart(part)
firstpart(n)
nextdiffpart(part, check=TRUE)
islastdiffpart(part)
firstdiffpart(n)
nextrestrictedpart(part, check=TRUE)
islastrestrictedpart(part)
firstrestrictedpart(n, m, include.zero=TRUE)
nextblockpart(part, f, n=sum(part), include.fewer=FALSE, check=TRUE)
islastblockpart(part, f, n=NULL, include.fewer=FALSE)
firstblockpart(f, n=NULL, include.fewer=FALSE)
nextcomposition(comp, restricted, include.zero=TRUE, check=TRUE)
islastcomposition(comp, restricted, include.zero=TRUE)
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 compo-
sitions 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

# Do the optimization in scrabble vignette, one partition at a time:
# (but with a smaller letter bag)
scrabble <- c(a=9 , b=2 , c=2 , 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, give = FALSE)
Q(n, give = FALSE)
R(m, n, include.zero = FALSE)
S(f, n = NULL, include.fewer = 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,give=TRUE)
Q(10,give=TRUE)
R(10,20,include.zero=FALSE)
R(10,20,include.zero=TRUE)
S(1:4,5)
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; and 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.fewer=FALSE)
multiset(v, n=length(v))
msset(v)
multinomial(v)
allbinom(n,k)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Integer to be partitioned. In function blockparts(), the default of NULL means to return all partitions of any size</td>
</tr>
<tr>
<td>m</td>
<td>In functions restrictedparts() and compositions(), the order of the partition</td>
</tr>
<tr>
<td>include.zero</td>
<td>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)</td>
</tr>
<tr>
<td>include.fewer</td>
<td>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</td>
</tr>
<tr>
<td>decreasing</td>
<td>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 to decreasing to FALSE has the effect of making conjugate() return garbage</td>
</tr>
<tr>
<td>f</td>
<td>In function blockparts(), a vector of strictly positive integers that gives the maximal number of blocks; see details</td>
</tr>
<tr>
<td>v</td>
<td>In function multiset(), an integer vector representing a multiset. Argument n is the size of the sample to be taken</td>
</tr>
<tr>
<td>k</td>
<td>In function allbinom(), the size of the set to be chosen; arguments match those of choose()</td>
</tr>
</tbody>
</table>
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 \( c(rep(0,m-1),n) \).

  Functions `parts()` and `restrictedparts()` overlap in functionality. Note, however, that they can return identical partitions but in a different order: `parts(6)` and `restrictedparts(6,6)` for example. 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 \( \sum_{i=1}^{n} a_i = n \) 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 (ie a position with zero capacity).

  If \( n \) takes its default value of \texttt{NULL} \( \), then the restriction \( \sum_{i=1}^{n} a_i = n \) 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.

- With a single argument, `compositions(n)` returns all \( 2^{n-1} \) ways of partitioning an integer; thus \( 4+1+1 \) is distinct from \( 1+4+1 \) or \( 1+1+4 \).

  With two arguments, `compositions(n,m)` returns all nonnegative solutions to \( x_1 + \cdots + x_m = n \).

  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.

- Function `multiset()` returns all ways of ordering a multiset (`mset()` is a low-level helper function).

- Function `multinomial(v)` returns all ways of partitioning a set into distinguishable boxes of capacities \( v[1],v[2],\ldots,v[n] \). The number of columns is given by the multinomial coefficient \( \left( \sum v_i \right) \).

- Function `allbinom(n,k)` is provided for convenience; it enumerates the ways of choosing \( k \) objects from \( n \).

```r
> parts(7)
[1,]  7 6 5 5 4 4 3 3 3 3 3 2 2 2 1
[2,]  0 1 2 1 3 2 1 3 2 2 1 2 1 1
[3,]  0 0 0 1 0 1 1 2 1 1 2 1 1 1
[4,]  0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
[5,]  0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
[6,]  0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
[7,]  0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

> P(7)
[1] 15

> diffparts(9)
[1,]  9 8 7 6 6 5 5 4
[2,]  0 1 2 3 2 4 3 3
[3,]  0 0 0 0 1 0 1 2

> Q(9)
```
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].
If you have a *minimum* number of balls in each block, a construction like

```r
x <- c(1,1,2,1)  # min
y <- c(2,3,4,5)  # max
sweep(blockparts(y-x,7-sum(x)),1,x,"+")
```

can be helpful (that is, subtract off the minimum number of balls and add them back again at the end).

**Author(s)**
Robin K. S. Hankin

**References**
- D. Knuth, 2004. The art of computer programming, pre-fascicle 2B “Generating all permutations”

**See Also**
nextpart

**Examples**
```r
parts(5)
diffparts(10)
matplot(t(diffparts(27)),type='l',lty=1)

restrictedparts(9,4)
restrictedparts(9,4,FALSE)
restrictedparts(9,4,decreasing=TRUE)

blockparts(1:4)
blockparts(1:4,3)
blockparts(1:4,3,include.fewer=TRUE)

blockparts(c(4,3,3,2),5)  # Knuth's example, pre-fascicle 3a, p16

compositions(4)          # not the same as parts(4)
compositions(4,4)

# With 10 blocks, enumerate all partitions with maxima of 1:5 and minima # of c(0,1,1,2,1):
```
perms

```r
a <- c(0,1,1,2,1)
sweep(blockparts(1:5-a,10-sum(a)),1,a,"+")

#Knuth's example:
multiset(c(1,2,2,3))
multiset(rep(4*1:3,1:3),3)
```

### perms

**Enumerate the permutations of a vector**

#### Description

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

Function `perms()` returns all permutations in lexicographic order; function `plainperms()` returns all permutations by repeatedly exchanging adjacent pairs.

#### Usage

```r
perms(n)
plainperms(n)
```

#### Arguments

- **n**  
  Integer argument; permutations of \( 1:n \) returned

#### Note

Comments in the C code; algorithm lifted from ‘fasc2b.pdf’.

#### Author(s)

D. E. Knuth; C and R transliteration by Robin K. S. Hankin

#### References


#### See Also

- `parts`
print.partition

Print methods for partition objects and equivalence objects

Description

A print method for partition objects, summary partition objects, and equivalence classes. Includes various configurable options

Usage

## S3 method for class 'partition'
print(x, mat = getOption("matrixlike"), h = getOption("horiz"), ...)

## S3 method for class 'summary.partition'
print(x, ...)

## S3 method for class 'equivalence'
print(x, sep = getOption("separator"), ...)

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

Author(s)

Robin K. S. Hankin
Examples

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

---

**Description**

Enumeration of set partitions

**Usage**

```r
setparts(x)
listParts(x, do.set=FALSE)
vec_to_set(vec)
vec_to_eq(vec)
```

**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.
- **do.set**: Boolean, with TRUE meaning to return the set partitions in terms of sets (as per sets package) and default FALSE meaning to present the result in terms of equivalence classes.
- **vec**: An integer vector representing a set partition.

**Details**

A partition of a set $S = \{1, \ldots, n\}$ is a family of sets $T_i, \ldots, T_k$ satisfying:

- $i \neq j \rightarrow T_i \cap T_j = \emptyset$
- $\bigcup_{i=1}^{k} T_i = S$
- $T_i \neq \emptyset$ 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 `listParts()`.

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

- $(1234)$
- $(123)(4), (124)(3), (134)(2), (234)(1)$
- $(12)(34), (13)(24), (14)(23)$
- $(1)(2)(3)(4)$
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 2 2 2 2 2 3 3
  [2,] 2 2 3 1 1 2 2 3 2 3 2 2 3 1 1 1
  [3,] 3 2 2 3 2 1 1 1 3 2 2 2 2 2 1 2
  [4,] 2 3 2 3 2 3 2 2 1 1 2 2 1 1 2 1
  [5,] 1 1 1 2 2 3 2 3 2 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="")`.

Functions `vec_to_set()` and `vec_to_eq()` are low-level helper functions. These take an integer vector, typically a column of a matrix produced by `setparts()` and return their set representation.

**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.

**Note**

The `clue` package by Kurt Hornik contains functionality for partitions (specifically `cl_meet()` and `cl_join()`) which might be useful. Option `do.set` invokes functionality from the `sets` package by Meyer et al.

Note carefully that `setparts(c(2,1,1))` does not enumerate the ways of placing four numbered balls in three boxes of capacities 2,1,1. This is because there are two boxes of capacity 1, and swapping the balls between these boxes gives the same set partition (because sets are unordered). To do this, use `multinomial(c(a=2,b=1,c=1))`. See the `setparts` vignette for more details.

**Author(s)**

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

**References**


See Also
parts, print.partition

Examples

setparts(4)  # all partitions of a set of 4 elements
setparts(c(3,3,2))  # all partitions of a set of 8 elements
# into sets of sizes 3,3,2.

jj <- restrictedparts(5,3)
setparts(jj)  # partitions of a set of 5 elements into
# at most 3 sets
listParts(jj)  # induced equivalence classes
setparts(conjugate(jj))  # partitions of a set of 5 elements into
# sets not exceeding 3 elements
setparts(diffparts(5))  # partitions of a set of 5 elements into
# sets of different sizes

summary.partition Provides a summary of a partition

Description
Provides a summary of an object of class partition: usually the first and last few partitions (columns)

Usage
## S3 method for class 'partition'
summary(object, ...)

Arguments

object Partition
...
Further arguments; see details section below
Details

The ellipsis arguments are used to pass how many columns at the start and the end of the matrix are selected; this defaults to 10.

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 ommitted, 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

```r
summary(parts(7))
```

```r
summary(parts(11), 3)
```
Index

* **math**
  
  as.matrix.partition, 3
  conjugate, 4
  nextpart, 6
  p, 7
  parts, 9
  perms, 13
  print.partition, 14
  setparts, 15
  summary.partition, 17

* **package**
  
  partitions-package, 2

allbinom (parts), 9
as.matrix.partition, 3
as.partition (as.matrix.partition), 3

bin, 3
bintocomp (bin), 3
blockparts (parts), 9

c_allblockparts (parts), 9
c_alldiffparts (parts), 9
c_allparts (parts), 9
c_allperms (parts), 9
c_allrestrictedparts (parts), 9
c_bintocomp (parts), 9
c_comptobin (parts), 9
c_conjugate (parts), 9
c_durfee (parts), 9
c_multiset (parts), 9
c_nextblockpart (parts), 9
c_nextdiffpart (parts), 9
c_nextpart (parts), 9
c_nextrestrictedpart (parts), 9
c_numbdiffparts (parts), 9
c_numbparts (parts), 9
c_plainperms (parts), 9
c_tobin (parts), 9
c_wrap (parts), 9
compositions (parts), 9
comptobin (bin), 3
conjugate, 4
diffparts (parts), 9

Durfee (conjugate), 4
durfee (conjugate), 4
durfee_sorted (conjugate), 4
firstblockpart (nextpart), 6
firstcomposition (nextpart), 6
firstdiffpart (nextpart), 6
firstpart (nextpart), 6
firstrestrictedpart (nextpart), 6
islastblockpart (nextpart), 6
islastcomposition (nextpart), 6
islastdiffpart (nextpart), 6
islastpart (nextpart), 6
islastrestrictedpart (nextpart), 6
listParts (setparts), 15
mset (parts), 9
multinomial (parts), 9
multiset (parts), 9

nextblockpart (nextpart), 6
nextcomposition (nextpart), 6
nextdiffpart (nextpart), 6
nextpart, 6, 12
nextrestrictedpart (nextpart), 6
numbrestrictedparts_R (parts), 9

P, 7
partitions (partitions-package), 2
partitions-package, 2
parts, 7, 9, 13, 17
perms, 13
plainperms (perms), 13
print (print.partition), 14
print.partition, 14, 17

Q (P), 7

R (P), 7
restrictedparts (parts), 9

S (P), 7
setparts, 15
summary.partition, 17

tobin (bin), 3
todec (bin), 3

vec_to_eq (setparts), 15
vec_to_set (setparts), 15