Package 'multicool'

May 9, 2026

Type Package

Title Permutations of Multisets in Cool-Lex Order

Version 1.0.1

Date 2024-02-05

Author James Curran, Aaron Williams, Jerome Kelleher, Dave Barber

Maintainer James Curran <j.curran@auckland.ac.nz>

Description A set of tools to permute multisets without loops or hash tables and to generate integer partitions. The permutation functions are based on C code from Aaron Williams. Cool-lex order is similar to colexicographical order. The algorithm is described in Williams, A. Loopless Generation of Multiset Permutations by Prefix Shifts. SODA 2009, Symposium on Discrete Algorithms, New York, United States. The permutation code is distributed without restrictions. The code for stable and efficient computation of multinomial coefficients comes from Dave Barber. The code can be download from <http://tamivox.org/dave/multinomial/index.html> and is distributed without conditions. The package also generates the integer partitions of a positive, non-zero integer n. The C++ code for this is based on Python code from Jerome Kelleher which can be found here <https://jeromekelleher.net/category/combinatorics.html>. The C++ code and Python code are distributed without conditions.

URL https://github.com/jmcurran/multicool

BugReports https://github.com/jmcurran/multicool/issues

Encoding UTF-8

License GPL-2

Depends methods, Rcpp (>= 0.11.2)

LinkingTo Rcpp

RcppModules Multicool

RoxygenNote 7.2.3

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-02-05 12:20:06 UTC

Contents

`allPerm` *Generate and return all permutations of a multiset*

Description

This function will return all permutations of a multiset

Usage

```
allPerm(mcObj)
```

Arguments

`mcObj`	an object of class mc - usually generated by `initMC`

Details

This function will return all permutations of a multiset. It makes no check to see if this is a sensible thing to do. Users are advised to check how many permutations are possible using the `multinom` function in this package.

Value

A matrix with each row being a different permutation of the multiset

Note

This function does not warn the user that the requested set of permutations may be very large. In addition, all working is handled entirely in memory, and so this may cause it to crash if the request is exceptionally large.

Author(s)

James M. Curran

See Also

`initMC`, `multinom`

Examples

```
## a small numeric example with 6 permuations
x = c(1,1,2,2)
m = initMC(x)
allPerm(m)

## a large character example - 60 possibilities
x = rep(letters[1:3], 3:1)
multinom(x) ## calculate the number of permutations
m = initMC(x)
allPerm(m)
```

`Bell` *Compute the Bell numbers*

Description

This function computes the Bell numbers, which is the summ of Stirling numbers of the second kind, $S(n, k)$, over $k = 1, \ldots, n$, i.e.

$$B_n = \sum_{k=1}^{n} S(n, k), n \geq 1$$

Usage

```
Bell(n)

B(n)
```

Arguments

`n`	A vector of one or more non-zero positive integers

Value

An vector of Bell numbers

Functions

- `B()`: Compute the Bell numbers

Author(s)

James Curran

References

https://en.wikipedia.org/wiki/Stirling_numbers_of_the_second_kind#Recurrence_relation

See Also

Stirling2

Examples

```
## returns B(6)
Bell(6)

## returns B(1), B(2), ..., B(6)
B(1:6)
```

---

genComp *Generate all, or a subset, of the integer partitions of an integer n.*

Description

This function will return either all, or a length restricted subset of the integer partitions of an integer n. The method works by considering compositions rather than partions, hence the name.

Usage

```
genComp(n, len = TRUE, addZeros = FALSE)
```

Arguments

n	A positive non-zero integer
len	Either logical TRUE, or an integer less than or equal to n. If the latter form is used then only those partions of length less than or equal to len are returned
addZeros	If true then the empty partitions are added to the list of partitions.

Details

This function will return all partions, or a subset, of an integer n. It makes no check to see if this is a sensible thing to do. It also does it in a lazy way in that in the restricted case it generates all partitions and then only returns those that satistfy the length constraint. Users are advised to check how many partitions are possible using partition number function which is implemented the P function in the **partions** package. Having said this P(50) is approximately 200 thousand, and P(100) around 190 million, so the function should work well for smallish n.

Value

A list with each list element representing an integer partition

Note

This function does not warn the user that the requested set of partitions may be very large. In addition, all working is handled entirely in memory, and so this may cause it to crash if the request is execeptionally large.

Author(s)

Jerome Kelleher (algorithm and Python version) and James M. Curran (C++ version/R interface)

References

Kelleher, J. (2005), Encoding Partitions As Ascending Compositions, PhD thesis, University College Cork.

Kelleher, J. and O'Sullivan, B. (2009), Generating All Partitions: A Comparison Of Two Encodings, https://arxiv.org/abs/0909.2331.

Kelleher, J. (2010) Generating Integer Partitions,https://jeromekelleher.net/tag/integer-partitions.html.

Examples

```
## a small numeric example with all 11 partitions of 6
genComp(6)

## a small example with the integer partitions of 6 of length 3 with empty partitions added
genComp(6, 3, TRUE)

## a larger example - 627 partions of 20, but restricted to those of length 3 or smaller
genComp(20, 3)
```

---

initMC *Initialise the permutation object*

Description

This function initialises the permutation object. It must be called before `nextPerm` can be called

Usage

```
initMC(x)
```

Arguments

`x`	a vector of integers, reals, logicals or characters

Value

a object of class `mc` which is a list containing elements

`mode`	- the mode of the original data in `x`, "integer", "double", or `mode(x)`
`set`	- either the multiset being permuted if `mode` is "integer" or a set of integers corresponding to the elements of the multiset

elements	- if mode is not "integer" then this contains the elements being permuted otherwise NULL
length	- the length of the multiset
mc	- a pointer to the internal C++ Multicool object. Users should not use this unless they really know what they are doing

Author(s)

James M. Curran

See Also

nextPerm

Examples

```
x = c(1,1,2,2)
m1 = initMC(x)
m1

## a non-integer example

x = rep(letters[1:4],c(2,1,2,2))
m2 = initMC(x)
m2
```

---

multinom *Calculate multinomial coefficients*

Description

This function calculates the number of permutations of a multiset, this being the multinomial coefficient. If a set X contains k unique elements $x_1, x_2, \ldots, x_k$ with associate counts (or multiplicities) of $n_1, n_2, \ldots, n_k$, then this function returns

$$\frac{n!}{n_1!n_2!\ldots n_k!}$$

where $n = \sum_{i=1} kn_i$.

Usage

```
multinom(x, counts = FALSE, useDouble = FALSE)
```

Arguments

x	Either a multiset (with one or more potentially non-unique elements), or if counts is TRUE a set of counts of the unique elements of X. If counts is FALSE and x is not numeric, then x will be coerced into an integer vector internally. If counts is TRUE then x must be a vector of integers that are greater than, or equal to zero.
counts	if counts is TRUE, then this means x is the set of counts $n_1, n_2, \ldots, n_k$ rather than the set itself
useDouble	if useDouble is TRUE then the computation will be done using double precision floating point arithmetic. This option was added because the internal code cannot handle integer overflow. The double precision code will may a result that is closer to the truth for large values, but this is not guaranteed. Ideally something like the GMP library should be used, but this is not a priority at this point in time.

Details

multinom depends on C++ code written by Dave Barber which can be found at http://tamivox.org/dave/multinomial/code.html. The code may require the STL algorithm library to be included in order to compile it.

Value

A single integer representing the multinomial coefficient for the given multiset, or given set of multiplicities.

Author(s)

James M. Curran, Dave Barber

References

http://tamivox.org/dave/multinomial/code.html

Examples

```
## An example with a multiset X = (a,a,a,b,b,c)
## There are 3 a s, 2 b s and 1 c, so the answer should be
## (3+2+1)!/(3!2!1!) = 6!/3!2!1! = 60
x = rep(letters[1:3],3:1)
multinom(x)

## in this example x is a vector of counts
## the answer should be the same as above as x = c(3,2,1)
x = rep(letters[1:3],3:1)
x = as.vector(table(x)) #coerce x into a vector of counts
multinom(x, counts = TRUE)


## An example of integer overflow. x is a vector of counts
```

```
## c(12,11,8,8,6,5). The true answer from Maple is
## 11,324,718,121,789,252,764,532,876,767,840,000
## The error in the integer based answer is obvious.
## The error using floating point is not, but from Maple is
## 0.705057123232160000e+10
## Thanks to Lev Dashevskiy for calling my attention to this.
## Not run: x = c(12,11,8,8,6,5)
multinom(x, counts = TRUE, useDouble = FALSE)
multinom(x, counts = TRUE, useDouble = TRUE)

## End(Not run)
```

---

nextPerm *Return the next permutation of the multiset*

Description

This function returns the next permuation of the multiset if there is one. `initMC` called before `nextPerm` can be called.

Usage

```
nextPerm(mcObj)
```

Arguments

`mcObj`	an S3 object of class `mc` which must be created with `initMC`

Value

either a vector with the next permutation of the multiset or `FALSE` when all permutations have been returned

Author(s)

James M. Curran

See Also

nextPerm

Examples

```
x = c(1,1,2,2)
m1 = initMC(x)

for(i in 1:6){
  cat(paste(paste(nextPerm(m1),collapse=","),"\n"))
```

```
}

## an example with letters
x = letters[1:4]
m2 = initMC(x)
nextPerm(m2)
nextPerm(m2)
## and so on
```

`Stirling2` *Compute Stirling numbers of the second kind*

Description

This function computes Stirling numbers of the second kind, $S(n, k)$, which count the number of ways of partitioning n distinct objects in to k non-empty sets.

Usage

```
Stirling2(n, k)

S2(n, k)
```

Arguments

`n`	A vector of one or more positive integers
`k`	A vector of one or more positive integers

Details

The implementation on this function is a simple recurrence relation which defines

$$S(n, k) = kS(n - 1, k), +S(n - 1, k - 1)$$

for $k > 0$ with the inital conditions $S(0, 0) = 1$ and $S(n, 0) = S(0, n) = 0$. If `n` and `n` have different lengths then `expand.grid` is used to construct a vector of (n, k) pairs

Value

An vector of Stirling numbers of the second kind

Functions

- `S2()`: Compute Stirling numbers of the second kind

Author(s)

James Curran

References

https://en.wikipedia.org/wiki/Stirling_numbers_of_the_second_kind#Recurrence_relation

Examples

```
## returns S(6, 3)
Stirling2(6, 3)

## returns S(6,1), S(6,2), ..., S(6,6)
S2(6, 1:6)

## returns S(6,1), S(5, 2), S(4, 3)
S2(6:4, 1:3)
```

Index