Package 'multicool'

October 21, 2026

Type Package

Title Permutations of Multisets in Cool-Lex Order

Version 1.0.2

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. Multinomial coefficients, Stirling numbers of the second kind and Bell numbers are computed exactly using the 'gmp' package. 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, and the algorithm is described in Kelleher and O'Sullivan (2009) <doi:10.48550/arXiv.0909.2331>. 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)

Imports gmp

LinkingTo Rcpp

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author James Curran [aut, cre],
Aaron Williams [aut],
Jerome Kelleher [aut]

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

Repository CRAN

Date/Publication 2026-10-21 13:40:02 UTC

Contents

allPerm 2
Bell 3
genComp 4
initMC 6
multinom 7
nextPerm 8
Stirling2 10

Index **12**

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

Description

This function will return all permutations of a multiset

Usage

```
allPerm(mcObj, nThreads = 1)
```

Arguments

`mcObj`	an object of class mc - usually generated by `initMC`
`nThreads`	the number of threads to use. Using more than one only helps for large results (at least 100,000 permutations; smaller results are always generated with one thread), and only for integer, double and logical multisets, and multisets of other types that are mapped from integer codes, such as Dates. Character and factor multisets always use one thread. The gain is limited by memory bandwidth, so it is usually well short of `nThreads` times faster.

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. `nextPerm` can step through the permutations in batches instead, without holding them all in memory.

Author(s)

James M. Curran

See Also

initMC, multinom, nextPerm

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)

## 362,880 permutations, using two threads
m = initMC(1:9)
p = allPerm(m, nThreads = 2)
dim(p)
```

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, exact = FALSE)

B(n, exact = FALSE)
```

Arguments

n	A vector of one or more non-zero positive integers
exact	if TRUE the exact results are returned as big integers of class bigz from the **gmp** package.

Value

A vector of Bell numbers. As with Stirling2, they are computed exactly and returned as doubles by default, or as exact bigz values if exact is TRUE.

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)

## the exact value of B(50)
Bell(50, exact = TRUE)
```

---

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 a positive integer. If the latter form is used then only those partions of length less than or equal to len are returned. If len is greater than or equal to n then all partitions are returned.
addZeros	If true then each partition is padded with zeros to length len, or to length n if len is TRUE.

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 exceeptionally 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. Archived copy: https://web.archive.org/web/20260423220351/https://jeromekelleher.net/generating-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, with at least two elements and no missing values

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`	- a multiset of integer codes `1, ..., k`, one code for each unique element of `x`
`elements`	- the unique elements of `x`, so that `elements[set]` recovers the multiset being permuted
`length`	- the length of the multiset
`plain`	- `TRUE` if `elements` is a plain integer, double, logical or character vector, which the C++ code can write directly
`mc`	- an external pointer to the internal C++ object that holds the position of `nextPerm`. Users should not use this unless they really know what they are doing. It does not survive saving and reloading the object, so call `initMC` again in a new session

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

## a real-valued example

m3 = initMC(c(1.5, 1.5, 2.5))
allPerm(m3)
```

`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 $n = \sum_{i=1}^{k} n_i$.

Usage

```
multinom(x, counts = FALSE, useDouble = FALSE, exact = 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`	kept for backwards compatibility, and no longer has any effect. The coefficient is always computed exactly, and returned as a double unless `exact` is `TRUE`.
`exact`	if `TRUE` the exact result is returned as a big integer of class `bigz` from the **gmp** package.

Value

The multinomial coefficient for the given multiset, or given set of multiplicities. By default this is a double: the exact value rounded to double precision, so it is exact up to 2^{53} and correct to about 15 significant digits beyond that. If `exact` is `TRUE` it is an exact `bigz`.

Author(s)

James M. Curran

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 a large coefficient. 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
## exact = TRUE gives this. The default double precision result only
## keeps the first 15 or so significant digits.
## Thanks to Lev Dashevskiy for calling my attention to this.
x = c(12,11,8,8,6,5)
multinom(x, counts = TRUE)
multinom(x, counts = TRUE, exact = TRUE)
```

`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, n = 1)
```

Arguments

mcObj	an S3 object of class mc which must be created with initMC
n	the number of permutations to return. Asking for several at a time is much faster than calling nextPerm repeatedly, while still only holding n permutations in memory.

Value

If n is 1, either a vector with the next permutation of the multiset or FALSE when all permutations have been returned. Because a permutation always has at least two elements, isFALSE(nextPerm(m)) detects the end even when the multiset is logical.

If n is greater than 1, a matrix whose rows are the next n permutations, with fewer rows when fewer than n are left, or FALSE when all permutations have been returned.

Calling allPerm does not change the position of nextPerm.

Author(s)

James M. Curran

See Also

initMC, allPerm

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

## process all 3,628,800 permutations of 1:10 in batches of 100,000,
## without ever holding them all in memory
m3 = initMC(1:10)
nFixed = 0
while(!isFALSE(p <- nextPerm(m3, 100000))){
  nFixed = nFixed + sum(p[, 1] == 1)
}
nFixed ## the permutations that start with 1, i.e. 9!
```

`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, exact = FALSE)

S2(n, k, exact = FALSE)
```

Arguments

`n`	A vector of one or more non-negative integers
`k`	A vector of one or more non-negative integers
`exact`	if `TRUE` the exact results are returned as big integers of class `bigz` from the **gmp** package.

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 `k` have different lengths then `expand.grid` is used to construct a vector of (n, k) pairs

Value

A vector of Stirling numbers of the second kind. They are always computed exactly. By default they are returned as doubles, so they are exact up to 2^{53} and correct to about 15 significant digits beyond that. If `exact` is `TRUE` they are exact `bigz` values.

Author(s)

James Curran

References

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

See Also

`as.bigz`

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)

## exact = TRUE gives exact values, even when they are very large
S2(100, 50)
S2(100, 50, exact = TRUE)
```

Index

∗ **combinations**
multinom, 7
∗ **multinomial**
multinom, 7
∗ **partitions**
Bell, 3
genComp, 4
Stirling2, 10
∗ **permutations**
allPerm, 2
multinom, 7

allPerm, 2, *9*
as.bigz, *10*

B *(*Bell*)*, 3
Bell, 3

genComp, 4

initMC, *3*, 6, *9*

multinom, *3*, 7

nextPerm, *2, 3*, 8

S2 *(*Stirling2*)*, 10
Stirling2, *4*, 10