

Package ‘arrangements’

July 22, 2025

Type Package

Title Fast Generators and Iterators for Permutations, Combinations,
Integer Partitions and Compositions

Version 1.1.9

Date 2020-09-12

Description Fast generators and iterators for permutations, combinations,
integer partitions and compositions. The arrangements are in
lexicographical order and generated iteratively in a memory efficient manner.
It has been demonstrated that 'arrangements' outperforms most existing
packages of similar kind. Benchmarks could be found at
<<https://randy3k.github.io/arrangements/articles/benchmark.html>>.

License MIT + file LICENSE

URL <https://github.com/randy3k/arrangements>

Depends R (>= 3.4.0)

Imports gmp, methods, R6

Suggests foreach, knitr, rmarkdown, testthat

ByteCompile yes

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 6.1.1

SystemRequirements gmp (>= 4.2.3)

Author Randy Lai [aut, cre]

Maintainer Randy Lai <randy.cs.lai@gmail.com>

Repository CRAN

Date/Publication 2020-09-13 14:00:07 UTC

Contents

arrangements-package	2
Combinations	3
combinations	4
Compositions	5
compositions	7
ncombinations	8
ncompositions	9
npartitions	10
npermutations	11
Partitions	12
partitions	13
Permutations	14
permutations	16
Index	18

arrangements-package	<i>arrangements: Fast Generators and Iterators for Permutations, Combinations, Integer Partitions and Compositions</i>
----------------------	--

Description

Fast generators and iterators for permutations, combinations, integer partitions and compositions. The arrangements are in lexicographical order and generated iteratively in a memory efficient manner. It has been demonstrated that 'arrangements' outperforms most existing packages of similar kind. Benchmarks could be found at <<https://randy3k.github.io/arrangements/articles/benchmark.html>>.

Author(s)

Maintainer: Randy Lai <randy.cs.lai@gmail.com>

See Also

- Useful links:
- <https://randy3k.github.io/arrangements>

Combinations

*Combinations iterator***Description**

This function returns a [Combinations](#) iterator for iterating combinations of k items from n items. The iterator allows users to fetch the next combination(s) via the `getnext()` method.

Usage

Combinations

```
icombinations(x = NULL, k = NULL, n = NULL, v = NULL,
              freq = NULL, replace = FALSE, skip = NULL)
```

Arguments

<code>x</code>	an integer or a vector, will be treated as n if integer; otherwise, will be treated as v . Should not be specified together with n and v .
<code>k</code>	an integer, the number of items drawn, defaults to n if <code>freq</code> is <code>NULL</code> else <code>sum(freq)</code>
<code>n</code>	an integer, the total number of items, its value may be implicitly deduced from <code>length(v)</code> or <code>length(freq)</code>
<code>v</code>	a vector to be drawn, defaults to <code>1:n</code> .
<code>freq</code>	an integer vector of item repeat frequencies
<code>replace</code>	an logical to draw items with replacement
<code>skip</code>	the number of combinations skipped

Format

An object of class `R6ClassGenerator` of length 25.

Details

The `Combinations` class can be initialized by using the convenient wrapper `icombinations` or

```
Combinations$new(n, k, v = NULL, freq = NULL, replace = FALSE)
```

```
getnext(d = 1L, layout = NULL, drop = NULL)
collect(layout = "row")
reset()
```

d number of fetched arrangements

layout if "row", "column" or "list" is specified, the returned value would be a "row-major" matrix, a "column-major" matrix or a list respectively

drop vectorize a matrix or unlist a list

See Also

[combinations](#) for generating all combinations and [ncombinations](#) to calculate number of combinations

Examples

```
icomb <- icombinations(5, 2)
icomb$getNext()
icomb$getNext(2)
icomb$getNext(layout = "column", drop = FALSE)
# collect remaining combinations
icomb$collect()

library(foreach)
foreach(x = icombinations(5, 2), .combine=c) %do% {
  sum(x)
}
```

combinations	<i>Combinations generator</i>
--------------	-------------------------------

Description

This function generates all the combinations of selecting k items from n items. The results are in lexicographical order.

Usage

```
combinations(x = NULL, k = NULL, n = NULL, v = NULL, freq = NULL,
  replace = FALSE, layout = NULL, nitem = -1L, skip = NULL,
  index = NULL, nsample = NULL, drop = NULL)
```

Arguments

x	an integer or a vector, will be treated as n if integer; otherwise, will be treated as v. Should not be specified together with n and v.
k	an integer, the number of items drawn, defaults to n if freq is NULL else sum(freq)
n	an integer, the total number of items, its value may be implicitly deduced from length(v) or length(freq)
v	a vector to be drawn, defaults to 1:n.
freq	an integer vector of item repeat frequencies
replace	an logical to draw items with replacement
layout	if "row", "column" or "list" is specified, the returned value would be a "row-major" matrix, a "column-major" matrix or a list respectively
nitem	number of combinations required, usually used with skip

skip	the number of combinations skipped
index	a vector of indices of the desired combinations
nsample	sampling random combinations
drop	vectorize a matrix or unlist a list

See Also

[icombinations](#) for iterating combinations and [ncombinations](#) to calculate number of combinations

Examples

```
# choose 2 from 4
combinations(4, 2)
combinations(LETTERS[1:3], k = 2)

# multiset with frequencies c(2, 3)
combinations(k = 3, freq = c(2, 3))

# with replacement
combinations(4, 2, replace = TRUE)

# column major
combinations(4, 2, layout = "column")

# list output
combinations(4, 2, layout = "list")

# specific range of combinations
combinations(4, 2, nitem = 2, skip = 3)

# specific combinations
combinations(4, 2, index = c(3, 5))

# random combinations
combinations(4, 2, nsample = 3)

# zero sized combinations
dim(combinations(5, 0))
dim(combinations(5, 6))
dim(combinations(0, 0))
dim(combinations(0, 1))
```

Description

This function returns a [Compositions](#) iterator for iterating compositions of a non-negative integer n into k parts or parts of any sizes. The iterator allows users to fetch the next partition(s) via the `getnext()` method.

Usage

```
Compositions
```

```
icompositions(n, k = NULL, descending = FALSE, skip = NULL)
```

Arguments

<code>n</code>	an non-negative integer to be partitioned
<code>k</code>	number of parts
<code>descending</code>	an logical to use reversed lexicographical order
<code>skip</code>	the number of compositions skipped

Format

An object of class `R6ClassGenerator` of length 25.

Details

The `Compositions` class can be initialized by using the convenient wrapper `icompositions` or

```
Compositions$new(n, k = NULL, descending = FALSE)
```

```
getnext(d = 1L, layout = NULL, drop = NULL)
collect(layout = "row")
reset()
```

d number of fetched arrangements

layout if "row", "column" or "list" is specified, the returned value would be a "row-major" matrix, a "column-major" matrix or a list respectively

drop vectorize a matrix or unlist a list

See Also

[compositions](#) for generating all compositions and [ncompositions](#) to calculate number of compositions

Examples

```

ipart <- icompositions(4)
ipart$getNext()
ipart$getNext(2)
ipart$getNext(layout = "column", drop = FALSE)
# collect remaining compositions
ipart$collect()

library(foreach)
foreach(x = icompositions(6, 2), .combine=c) %do% {
  prod(x)
}

```

compositions

*Compositions generator***Description**

This function generates the compositions of a non-negative integer n into k parts or parts of any sizes. The results are in lexicographical or reversed lexicographical order.

Usage

```

compositions(n, k = NULL, descending = FALSE, layout = NULL,
  nitem = -1L, skip = NULL, index = NULL, nsample = NULL,
  drop = NULL)

```

Arguments

<code>n</code>	an non-negative integer to be partitioned
<code>k</code>	number of parts
<code>descending</code>	an logical to use reversed lexicographical order
<code>layout</code>	if "row", "column" or "list" is specified, the returned value would be a "row-major" matrix, a "column-major" matrix or a list respectively
<code>nitem</code>	number of compositions required, usually used with <code>skip</code>
<code>skip</code>	the number of compositions skipped
<code>index</code>	a vector of indices of the desired compositions
<code>nsample</code>	sampling random compositions
<code>drop</code>	vectorize a matrix or unlist a list

See Also

[icompositions](#) for iterating compositions and [ncompositions](#) to calculate number of compositions

Examples

```
# all compositions of 4
compositions(4)
# reversed lexicographical order
compositions(4, descending = TRUE)

# fixed number of parts
compositions(6, 3)
# reversed lexicographical order
compositions(6, 3, descending = TRUE)

# column major
compositions(4, layout = "column")
compositions(6, 3, layout = "column")

# list output
compositions(4, layout = "list")
compositions(6, 3, layout = "list")

# zero sized compositions
dim(compositions(0))
dim(compositions(5, 0))
dim(compositions(5, 6))
dim(compositions(0, 0))
dim(compositions(0, 1))
```

ncombinations	<i>Number of combinations</i>
---------------	-------------------------------

Description

Number of combinations

Usage

```
ncombinations(x = NULL, k = NULL, n = NULL, v = NULL,
  freq = NULL, replace = FALSE, bigz = FALSE)
```

Arguments

x	an integer or a vector, will be treated as n if integer; otherwise, will be treated as v. Should not be specified together with n and v.
k	an integer, the number of items drawn, defaults to n if freq is NULL else sum(freq)
n	an integer, the total number of items, its value may be implicitly deduced from length(v) or length(freq)
v	a vector to be drawn, defaults to 1:n.
freq	an integer vector of item repeat frequencies

replace an logical to draw items with replacement
 bigz an logical to use [gmp::bigz](#)

See Also

[combinations](#) for generating all combinations and [icombinations](#) for iterating combinations

Examples

```
ncombinations(5, 2)
ncombinations(LETTERS, k = 5)

# integer overflow
## Not run: ncombinations(40, 15)
ncombinations(40, 15, bigz = TRUE)

# number of combinations of `c("a", "b", "b")`
# they are `c("a", "b")` and `c("b", "b")`
ncombinations(k = 2, freq = c(1, 2))

# zero sized combinations
ncombinations(5, 0)
ncombinations(5, 6)
ncombinations(0, 1)
ncombinations(0, 0)
```

ncompositions	<i>Number of compositions</i>
---------------	-------------------------------

Description

Number of compositions

Usage

```
ncompositions(n, k = NULL, bigz = FALSE)
```

Arguments

n an non-negative integer to be partitioned
 k number of parts
 bigz an logical to use [gmp::bigz](#)

See Also

[compositions](#) for generating all compositions and [icompositions](#) for iterating compositions

Examples

```
# number of compositions of 10
ncompositions(10)
# number of compositions of 10 into 5 parts
ncompositions(10, 5)

# integer overflow
## Not run: ncompositions(160)
ncompositions(160, bigz = TRUE)

# zero sized compositions
ncompositions(0)
ncompositions(5, 0)
ncompositions(5, 6)
ncompositions(0, 0)
ncompositions(0, 1)
```

npartitions	<i>Number of partitions</i>
-------------	-----------------------------

Description

Number of partitions

Usage

```
npartitions(n, k = NULL, distinct = FALSE, bigz = FALSE)
```

Arguments

n	an non-negative integer to be partitioned
k	number of parts
distinct	an logical to restrict distinct values
bigz	an logical to use gmp::bigz

See Also

[partitions](#) for generating all partitions and [ipartitions](#) for iterating partitions

Examples

```
# number of partitions of 10
npartitions(10)
# number of partitions of 10 into 5 parts
npartitions(10, 5)

# integer overflow
## Not run: npartitions(160)
```

```

npartitions(160, bigz = TRUE)

# zero sized partitions
npartitions(0)
npartitions(5, 0)
npartitions(5, 6)
npartitions(0, 0)
npartitions(0, 1)

```

npermutations	<i>Number of permutations</i>
---------------	-------------------------------

Description

Number of permutations

Usage

```

npermutations(x = NULL, k = NULL, n = NULL, v = NULL,
  freq = NULL, replace = FALSE, bigz = FALSE)

```

Arguments

x	an integer or a vector, will be treated as n if integer; otherwise, will be treated as v. Should not be specified together with n and v.
k	an integer, the number of items drawn, defaults to n if freq is NULL else sum(freq)
n	an integer, the total number of items, its value may be implicitly deduced from length(v) or length(freq)
v	a vector to be drawn, defaults to 1:n.
freq	an integer vector of item repeat frequencies
replace	an logical to draw items with replacement
bigz	an logical to use gmp::bigz

See Also

[permutations](#) for generating all permutations and [ipermutations](#) for iterating permutations

Examples

```

npermutations(7)
npermutations(LETTERS[1:5])
npermutations(5, 2)
npermutations(LETTERS, k = 5)

# integer overflow
## Not run: npermutations(14, 10)
npermutations(14, 10, bigz = TRUE)

```

```
# number of permutations of `c("a", "b", "b")`
# they are `c("a", "b")`, `c("b", "b")` and `c("b", "b")`
npermutations(k = 2, freq = c(1, 2))

# zero sized partitions
npermutations(0)
npermutations(5, 0)
npermutations(5, 6)
npermutations(0, 1)
npermutations(0, 0)
```

Partitions

Partitions iterator

Description

This function returns a [Partitions](#) iterator for iterating partitions of an non-negative integer n into k parts or parts of any sizes. The iterator allows users to fetch the next partition(s) via the `getnext()` method.

Usage

Partitions

```
ipartitions(n, k = NULL, distinct = FALSE, descending = FALSE,
  skip = NULL)
```

Arguments

<code>n</code>	an non-negative integer to be partitioned
<code>k</code>	number of parts
<code>distinct</code>	an logical to restrict distinct values
<code>descending</code>	an logical to use reversed lexicographical order
<code>skip</code>	the number of partitions skipped

Format

An object of class `R6ClassGenerator` of length 25.

Details

The `Partitions` class can be initialized by using the convenient wrapper `ipartitions` or

```
Partitions$new(n, k = NULL, descending = FALSE)
```

```

getnext(d = 1L, layout = NULL, drop = NULL)
collect(layout = "row")
reset()

```

d number of fetched arrangements

layout if "row", "column" or "list" is specified, the returned value would be a "row-major" matrix, a "column-major" matrix or a list respectively

drop vectorize a matrix or unlist a list

See Also

[partitions](#) for generating all partitions and [npartitions](#) to calculate number of partitions

Examples

```

ipart <- ipartitions(10)
ipart$getnext()
ipart$getnext(2)
ipart$getnext(layout = "column", drop = FALSE)
# collect remaining partitions
ipart$collect()

library(foreach)
foreach(x = ipartitions(6, 2), .combine=c) %do% {
  prod(x)
}

```

partitions

Partitions generator

Description

This function partitions an non-negative interger n into k parts or parts of any sizes. The results are in lexicographical or reversed lexicographical order.

Usage

```

partitions(n, k = NULL, distinct = FALSE, descending = FALSE,
  layout = NULL, nitem = -1L, skip = NULL, index = NULL,
  nsample = NULL, drop = NULL)

```

Arguments

n	an non-negative integer to be partitioned
k	number of parts
distinct	an logical to restrict distinct values
descending	an logical to use reversed lexicographical order

layout	if "row", "column" or "list" is specified, the returned value would be a "row-major" matrix, a "column-major" matrix or a list respectively
nitem	number of partitions required, usually used with skip
skip	the number of partitions skipped
index	a vector of indices of the desired partitions
nsample	sampling random partitions
drop	vectorize a matrix or unlist a list

See Also

[ipartitions](#) for iterating partitions and [npartitions](#) to calculate number of partitions

Examples

```
# all partitions of 6
partitions(6)
# reversed lexicographical order
partitions(6, descending = TRUE)

# fixed number of parts
partitions(10, 5)
# reversed lexicographical order
partitions(10, 5, descending = TRUE)

# column major
partitions(6, layout = "column")
partitions(6, 3, layout = "column")

# list output
partitions(6, layout = "list")
partitions(6, 3, layout = "list")

# zero sized partitions
dim(partitions(0))
dim(partitions(5, 0))
dim(partitions(5, 6))
dim(partitions(0, 0))
dim(partitions(0, 1))
```

Description

This function returns a [Permutations](#) iterator for iterating permutations of k items from n items. The iterator allows users to fetch the next permutation(s) via the `getnext()` method.

Usage

Permutations

```
ipermutations(x = NULL, k = NULL, n = NULL, v = NULL,
  freq = NULL, replace = FALSE, skip = NULL)
```

Arguments

x	an integer or a vector, will be treated as n if integer; otherwise, will be treated as v. Should not be specified together with n and v.
k	an integer, the number of items drawn, defaults to n if freq is NULL else sum(freq)
n	an integer, the total number of items, its value may be implicitly deduced from length(v) or length(freq)
v	a vector to be drawn, defaults to 1:n.
freq	an integer vector of item repeat frequencies
replace	an logical to draw items with replacement
skip	the number of combinations skipped

Format

An object of class R6ClassGenerator of length 25.

Details

The Permutations class can be initialized by using the convenient wrapper ipermutations or

```
Permutations$new(n, k, v = NULL, freq = NULL, replace = FALSE)
```

```
getnext(d = 1L, layout = NULL, drop = NULL)
collect(layout = "row")
reset()
```

d number of fetched arrangements

layout if "row", "column" or "list" is specified, the returned value would be a "row-major" matrix, a "column-major" matrix or a list respectively

drop vectorize a matrix or unlist a list

See Also

[permutations](#) for generating all permutations and [npermutations](#) to calculate number of permutations

Examples

```

iperm <- ipermutations(5, 2)
iperm$getnext()
iperm$getnext(2)
iperm$getnext(layout = "column", drop = FALSE)
# collect remaining permutations
iperm$collect()

library(foreach)
foreach(x = ipermutations(5, 2), .combine=c) %do% {
  sum(x)
}

```

permutations	<i>Permutations generator</i>
--------------	-------------------------------

Description

This function generates all the permutations of selecting k items from n items. The results are in lexicographical order.

Usage

```

permutations(x = NULL, k = NULL, n = NULL, v = NULL, freq = NULL,
  replace = FALSE, layout = NULL, nitem = -1L, skip = NULL,
  index = NULL, nsample = NULL, drop = NULL)

```

Arguments

<code>x</code>	an integer or a vector, will be treated as n if integer; otherwise, will be treated as v . Should not be specified together with n and v .
<code>k</code>	an integer, the number of items drawn, defaults to n if <code>freq</code> is <code>NULL</code> else <code>sum(freq)</code>
<code>n</code>	an integer, the total number of items, its value may be implicitly deduced from <code>length(v)</code> or <code>length(freq)</code>
<code>v</code>	a vector to be drawn, defaults to <code>1:n</code> .
<code>freq</code>	an integer vector of item repeat frequencies
<code>replace</code>	an logical to draw items with replacement
<code>layout</code>	if "row", "column" or "list" is specified, the returned value would be a "row-major" matrix, a "column-major" matrix or a list respectively
<code>nitem</code>	number of permutations required, usually used with <code>skip</code>
<code>skip</code>	the number of permutations skipped
<code>index</code>	a vector of indices of the desired permutations
<code>nsample</code>	sampling random permutations
<code>drop</code>	vectorize a matrix or unlist a list

See Also

[ipermutations](#) for iterating permutations and [npermutations](#) to calculate number of permutations

Examples

```
permutations(3)
permutations(LETTERS[1:3])

# choose 2 from 4
permutations(4, 2)
permutations(LETTERS[1:3], k = 2)

# multiset with frequencies c(2, 3)
permutations(k = 3, freq = c(2, 3))

# with replacement
permutations(4, 2, replace = TRUE)

# column major
permutations(3, layout = "column")
permutations(4, 2, layout = "column")

# list output
permutations(3, layout = "list")
permutations(4, 2, layout = "list")

# specific range of permutations
permutations(4, 2, nitem = 2, skip = 3)

# specific permutations
permutations(4, 2, index = c(3, 5))

# random permutations
permutations(4, 2, nsample = 3)

# zero sized permutations
dim(permutations(0))
dim(permutations(5, 0))
dim(permutations(5, 6))
dim(permutations(0, 0))
dim(permutations(0, 1))
```

Index

* datasets

- Combinations, [3](#)
- Compositions, [5](#)
- Partitions, [12](#)
- Permutations, [14](#)

arrangements (arrangements-package), [2](#)
arrangements-package, [2](#)

Combinations, [3](#), [3](#)
combinations, [4](#), [4](#), [9](#)
Compositions, [5](#), [6](#)
compositions, [6](#), [7](#), [9](#)

gmp::bigz, [9–11](#)

icombinations, [5](#), [9](#)
icombinations (Combinations), [3](#)
icompositions, [7](#), [9](#)
icompositions (Compositions), [5](#)
ipartitions, [10](#), [14](#)
ipartitions (Partitions), [12](#)
ipermutations, [11](#), [17](#)
ipermutations (Permutations), [14](#)

ncombinations, [4](#), [5](#), [8](#)
ncompositions, [6](#), [7](#), [9](#)
npartitions, [10](#), [13](#), [14](#)
npermutations, [11](#), [15](#), [17](#)

Partitions, [12](#), [12](#)
partitions, [10](#), [13](#), [13](#)
Permutations, [14](#), [14](#)
permutations, [11](#), [15](#), [16](#)