Package 'arrangements'

November 8, 2018

Type Package

Title Fast Generators and Iterators for Permutations, Combinations and Partitions

Version 1.1.5

Date 2018-11-07

Description Fast generators and iterators for permutations, combinations and partitions. The iterators allow users to generate arrangements in a memory efficient manner and the generated arrangements are in lexicographical (dictionary) order. Permutations and combinations can be drawn with/without replacement and support multisets. It has been demonstrated that 'arrangements' outperforms most of the existing packages of similar kind. Some benchmarks could be found at <https://randy3k.github.io/arrangements/articles/benchmark.html>.

URL https://randy3k.github.io/arrangements

License MIT + file LICENSE

Depends R (>= 3.4.0)

Imports methods, R6, gmp

Suggests foreach, testthat, knitr, rmarkdown

NeedsCompilation yes

SystemRequirements gmp (>= 4.2.3)

ByteCompile yes

RoxygenNote 6.1.0

Author Randy Lai [aut, cre]

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

Repository CRAN

Date/Publication 2018-11-08 15:20:06 UTC

R topics documented:

arrangements-package . . . 2
Combinations . . . 2
combinations . . . 4
ncombinations . . . 5
npartitions . . . 6
npermutations . . . 7
Partitions . . . 8
partitions . . . 10
Permutations . . . 11
permutations . . . 12

Index **15**

`arrangements-package` *arrangements: Fast Generators and Iterators for Permutations, Combinations and Partitions*

Description

Fast generators and iterators for permutations, combinations and partitions. The iterators allow users to generate arrangements in a memory efficient manner and the generated arrangements are in lexicographical (dictionary) order. Permutations and combinations can be drawn with/without replacement and support multisets. It has been demonstrated that 'arrangements' outperforms most of the existing packages of similar kind. Some benchmarks could be found at <https://randy3k.github.io/arrangements/articles/bencl

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, k = n, 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
k	an integer, the number of items drawn
n	an integer, the total number of items, its actual value may be determined by other variables
v	a vector to be drawn
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 24.

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` *Combinations generator*

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 = n, 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`
`k`	an integer, the number of items drawn
`n`	an integer, the total number of items, its actual value may be determined by other variables
`v`	a vector to be drawn
`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")

# specifc 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))
```

---

ncombinations *Number of combinations*

Description

Number of combinations

Usage

```
ncombinations(x = NULL, k = n, 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
k	an integer, the number of items drawn
n	an integer, the total number of items, its actual value may be determined by other variables
v	a vector to be drawn
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)
```

---

npartitions *Number of partitions*

Description

Number of partitions

Usage

```
npartitions(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

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 *Number of permutations*

Description

Number of permutations

Usage

```
npermutations(x = NULL, k = n, 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
k	an integer, the number of items drawn
n	an integer, the total number of items, its actual value may be determined by other variables
v	a vector to be drawn

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, descending = FALSE, skip = NULL)
```

Arguments

n	an non-negative integer to be partitioned
k	number of parts
descending	logical to use reversed lexicographical order
skip	the number of partitions skipped

Format

An object of class R6ClassGenerator of length 24.

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, 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
descending	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))
```

`Permutations` *Permutations iterator*

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, k = n, 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
k	an integer, the number of items drawn
n	an integer, the total number of items, its actual value may be determined by other variables
v	a vector to be drawn
`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 24.

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` *Permutations generator*

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 = n, 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
k	an integer, the number of items drawn
n	an integer, the total number of items, its actual value may be determined by other variables
v	a vector to be drawn
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 permutations required, usually used with skip
skip	the number of permutations skipped
index	a vector of indices of the desired permutations
nsample	sampling random permutations
drop	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")

# specifc 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

∗Topic **datasets**
Combinations, 2
Partitions, 8
Permutations, 11

arrangements *(*arrangements-package*)*, 2
arrangements-package, 2

Combinations, 2
combinations, *3*, 4, *6*

gmp::bigz, *6–8*

icombinations, *5, 6*
icombinations *(*Combinations*)*, 2
ipartitions, *7*, *10*
ipartitions *(*Partitions*)*, 8
ipermutations, *8*, *13*
ipermutations *(*Permutations*)*, 11

ncombinations, *3*, *5*, 5
npartitions, 6, *9, 10*
npermutations, 7, *12, 13*

Partitions, 8
partitions, *7*, *9*, 10
Permutations, 11
permutations, *8*, *12*, 12