Package 'combiter'

October 12, 2022

Type Package

Title Combinatorics Iterators

Version 1.0.3

Description Provides iterators for combinations, permutations, subsets, and Cartesian product, which allow one to go through all elements without creating a huge set of all possible values.

License MIT + file LICENSE

Depends R (>= 3.1)

LazyData TRUE

LinkingTo Rcpp

Imports iterators, itertools, Rcpp

RoxygenNote 6.0.1

Suggests combinat, foreach, testthat

URL https://github.com/kota7/combiter

BugReports https://github.com/kota7/combiter/issues

NeedsCompilation yes

Author Kota Mori [aut, cre]

Maintainer Kota Mori <kmori05@gmail.com>

Repository CRAN

Date/Publication 2017-12-04 12:36:31 UTC

R topics documented:

`getFirst` *First Value of Iterator*

Description

`getFirst` is a generic function that returns the first value of iterators

Usage

```
getFirst(obj, ...)
```

Arguments

`obj`	an R object
`...`	additional arguments

Value

iterator value, format dependes on the objects

`getLast` *Last Value of Iterator*

Description

`getFirst` is a generic function that returns the last value of iterators

Usage

```
getLast(obj, ...)
```

Arguments

`obj`	an R object
`...`	additional arguments

Value

iterator value, format dependes on the objects

hasPrev *Does This Iterator Have A Previous Element*

Description

hasPrev is a generic function that indicates if the iterator has another element backward.

Usage

```
hasPrev(obj, ...)
```

Arguments

obj	an R object
...	additional arguments

Value

Logical value indicating whether the iterator has a previous element.

icartes *Cartesian Product Iterator*

Description

Create an iterator going through Cartesian product of several items.

Usage

```
icartes(nvec)

icartesv(...)
```

Arguments

nvec	integer vector of number of items
...	set of iterables (subsettable by [)

Details

- icartes iterates through all combinations of integers
- icartesv iterates through all combinations of general values

Value

iterator object

Examples

```
x <- icartes(c(3, 2, 4))
ct <- 0
while (hasNext(x))
{
  ct <- ct + 1
  i <- nextElem(x)
  cat(sprintf("%3d : %s\n", ct, paste0(i, collapse = " ")))
}

x <- icartesv(Month=c("Jan", "Feb", "Mar"),
              Loc=c("NY", "LA"),
              By=c("car", "plane", "bus"))
as.list(x)
```

`icomb` *Combination Iterator*

Description

Create an iterator for all combinations k integers out of 1 through n.

Usage

```
icomb(n, k)

icombv(values, k)
```

Arguments

`n`	positive integer
`k`	positive integer no greater than n
`values`	iterable (subsettable by `[`)

Details

- `icomb` iterates through integer vectors
- `icombv` iterates through general values

Value

iterator object

Examples

```
x <- icomb(5, 3)
ct <- 0
while (hasNext(x))
{
  ct <- ct + 1
  i <- nextElem(x)
  cat(sprintf("%3d : %s\n", ct, paste0(i, collapse = " ")))
}

as.list(icombv(c("A", "G", "C"), 2))
```

`iperm` *Permutation Iterator*

Description

Create an iterator for all permutations of size k of integers 1 to n.

Usage

```
iperm(n, k = n)

ipermv(values, k = length(values))
```

Arguments

`n`	positive integer
`k`	positive integer
`values`	iterable (subsettable by `[`)

Details

- `iperm` iterates through integer vectors
- `ipermv` iterates through general values

Value

iterator object

Examples

```
x <- iperm(3)
ct <- 0
while (hasNext(x))
{
  ct <- ct + 1
  i <- nextElem(x)
```

```
  cat(sprintf("%3d : %s\n", ct, paste0(i, collapse = " ")))
}


as.list(ipermv(c("R", "G", "B")))
```

---

isubset *Subset Iterator*

Description

Create an iterator for all subsets of integers 1 through n.

Usage

```
isubset(n)

isubsetv(values)
```

Arguments

n	positive integer
values	iterable (subsettable by [)

Details

- isubset iterates through integer vectors
- isubsetv iterates through general values

Value

iterator object

Examples

```
x <- isubset(3)
ct <- 0
while (hasNext(x))
{
  ct <- ct + 1
  i <- nextElem(x)
  cat(sprintf("%3d : %s\n", ct, paste0(i, collapse = " ")))
}

as.list(isubsetv(letters[1:4]))
```

prevElem *Get Previous Element of Iterator*

Description

`prevElem` is a generic funcion to move an iterator object one step backward.

Usage

```
prevElem(obj, ...)
```

Arguments

`obj`	an R object
`...`	additional arguments

Value

iterator value

recursiveiter *Factory of Iterators defined by Recursive Transition Functions*

Description

This is a constructor for custom iterator objects. It requires four functions, "next", "prev", "first", and "last", and additional parameters.

The state of the constructor is characterized by the variable `i`. The "next" and "prev" function must take `i` and the parameters and return the next and previous state variables respectively. The behavior where there is no more state left is arbitrary.

The "first" and "last" functions must take the additional parameters and return the initial and last state variables respectively.

The created object is an iterator of class `recursiveiter`, which inherits `abstractiter` and `iter`. It can be used with `foreach` and accepts `as.list` conversion.

Usage

```
recursiveiter(nextFunc, prevFunc, firstFunc, lastFunc, ...)
```

Arguments

`nextFunc, prevFunc`	Functions that take the iterator state and the parameters `...` and returns the next or previous state
`firstFunc, lastFunc`	Functions that take the parameters `...` and returns the first or last state of the iteration
`...`	additional parameters of the iterator

Value

iterator object

Examples

```
fibiter <- recursiveiter(
  nextFunc = function(i) if (length(i)==1 && i==0) 1 else
                         if (length(i)==1 && i==1) c(1,1) else
                         c(sum(i), i[1]),
  prevFunc = NULL, firstFunc = function() 0, lastFunc = function() Inf)
for (k in 1:20) cat(nextElem(fibiter)[1], "")
```

Index