Package ‘kit’

July 21, 2020

Type Package

Title Data Manipulation Functions Implemented in C

Version 0.0.4

Date 2020-07-21

Author Morgan Jacob [aut, cre, cph]

Maintainer Morgan Jacob <morgan.emailbox@gmail.com>

Description Basic functions, implemented in C, for large data manipulation. Fast vectorised ifelse()/nested if()/switch() functions, psum()/pprod() functions equivalent to pmin()/pmax() plus others which are missing from base R. Most of these functions are callable at C level.

License GPL-3

Depends R (>= 3.1.0)

Encoding UTF-8

BugReports https://github.com/2005m/kit/issues

NeedsCompilation yes

ByteCompile TRUE

Repository CRAN

Date/Publication 2020-07-21 06:50:03 UTC

R topics documented:

`count` *count, countNA and countOccur*

Description

Simple functions to count the number of times an element occurs.

Usage

```
count(x, value)
countNA(x)
countOccur(x)
```

Arguments

`x` A vector or list for `countNA`. A vector for `count` and a vector or `data.frame` for `countOccur`.

`value` An element to look for. Must be non `NULL`, of length 1 and same type as `x`.

Value

For a vector `countNA` will return the total number of `NA` value. For a list, `countNA` will return a list with the number of `NA` in each item of the list. This is a major difference with `sum(is.na(x))` which will return the aggregated number of `NA`. Also, please note that every item of a list can be of different type and `countNA` will take them into account whether they are of type logical (`NA`), integer (`NA_integer_`), double (`NA_real_`), complex (`NA_complex_`) or character (`NA_character_`). As opposed to `countNA`, `count` does not support list type and requires `x` and `value` to be of the same type. Function `countOccur` takes vectors or data.frame as inputs and returns a `data.frame` with the number of times each value in the vector occurs or number of times each row in a `data.frame` occurs.

Author(s)

Morgan Jacob

See Also

`pcount`

Examples

```
x = c(1, 3, NA, 5)
count(x, 3)

countNA(x)
countNA(as.list(x))

countOccur(x)
```

```
# Benchmarks countNA
# ------------------
# x = sample(c(TRUE,NA,FALSE),1e8,TRUE) # 382 Mb
# microbenchmark::microbenchmark(
#   countNA(x),
#   sum(is.na(x)),
#   times=5L
# )
# Unit: milliseconds
#          expr   min    lq   mean  median    uq   max neval
# countNA(x)     98.7  99.2  101.2   100.1 101.4 106.4     5
# sum(is.na(x)) 405.4 441.3  478.9   461.1 523.9 562.6     5
#
# Benchmarks countOccur
# ---------------------
# x = rnorm(1e6)
# y = data.table::data.table(x)
# microbenchmark::microbenchmark(
#   kit= countOccur(x),
#   data.table = y[, .N, keyby = x],
#   table(x),
#   times = 10L
# )
# Unit: milliseconds
# expr        min         lq     mean    median      uq     max neval
# kit          62.26   63.88    89.29     75.49   95.17  162.40    10
# data.table  189.17  194.08   235.30    227.43  263.74  337.74    10 # setDTthreads(1L)
# data.table  140.15  143.91   190.04    182.85  234.48  261.43    10 # setDTthreads(2L)
# table(x)   3560.77 3705.06  3843.47   3807.12 4048.40 4104.11    10
```

`fduplicated/funique` *Fast duplicated and unique*

Description

Similar to base R functions `duplicated` and `unique`, `fduplicated` and `funique` are slightly faster for vectors and much faster for `data.frame`.

Usage

```
fduplicated(x)
funique(x)
```

Arguments

`x`	A vector or a data.frame.

Value

Function `fduplicated` returns a logical vector and `funique` returns a vector of the same type as `x` without the duplicated value.

Author(s)

Morgan Jacob

Examples

```
# Example 1: fduplicated
fduplicated(iris$Species)

# Example 2: funique
funique(iris$Species)

# Benchmarks
# ----------
# x = sample(c(1:10,NA_integer_),1e8,TRUE) # 382 Mb
# microbenchmark::microbenchmark(
#   duplicated(x),
#   fduplicated(x),
#   times = 5L
# )
# Unit: seconds
#           expr  min   lq  mean  median   uq   max neval
# duplicated(x)  2.21 2.21  2.48    2.21 2.22  3.55     5
# fduplicated(x) 1.14 1.14  1.18    1.14 1.14  1.32     5
#
# vs data.table
# -------------
# df = iris[,5:1]
# for (i in 1:16) df = rbind(df, df)  # 338 Mb
# dt = data.table::as.data.table(df)
# microbenchmark::microbenchmark(
#   kit = funique(df),
#   data.table = unique(dt),
#   times = 5L
# )
# Unit: seconds
#       expr  min   lq  mean  median    uq  max neval
# kit        1.22 1.27  1.33    1.27  1.36 1.55     5
# data.table 6.20 6.24  6.43    6.33  6.46 6.93     5  (setDTthreads(1L))
# data.table 4.20 4.25  4.47    4.26  4.32 5.33     5  (setDTthreads(2L))
```

fpos *Find a matrix position inside a larger matrix*

Description

The function `fpos` returns the locations (row and column index) where a small matrix may be found in a larger matrix. The function also works with vectors.

Usage

```
  fpos(needle, haystack, all=TRUE, overlap=TRUE)
```

Arguments

`needle` A matrix or vector to search for in the larger matrix or vector `haystack`. Note that the `needle` dimensions (row and column size) must be smaller than the `haystack` diemsions.

`haystack` A matrix or vector to look into.

`all` A logical value to indicate whether to return all occurences (`TRUE`) or only the first one (`FALSE`). Default value is `TRUE`.

`overlap` A logical value to indicate whether to allow the small matrix occurences to overlap or not. Default value is `TRUE`.

Value

A two columns matrix that contains the position or index where the small matrix (needle) can be found in the larger matrix. The first column refers to rows and the second to columns. In case both the needle and haystack are vectors, a vector is returned.

Author(s)

Morgan Jacob

Examples

```
# Example 1: find a matrix inside a larger one
big_matrix = matrix(c(1:30), nrow = 10)
small_matrix = matrix(c(14, 15, 24, 25), nrow = 2)

fpos(small_matrix, big_matrix)

# Example 2: find a vector inside a larger one
fpos(14:15, 1:30)

# Example 3:
big_matrix = matrix(c(1:5), nrow = 10, ncol = 5)
small_matrix = matrix(c(2:3), nrow = 2, ncol = 2)

# return all occurences
fpos(small_matrix, big_matrix)

# return only the first
fpos(small_matrix, big_matrix, all = FALSE)
```

```
# return non overlapping occurences
fpos(small_matrix, big_matrix, overlap = FALSE)

# Benchmarks
# ----------
# x = matrix(1:5, nrow=1e4, ncol=5e3) # 191Mb
# microbenchmark::microbenchmark(
#  fpos=kit::fpos(1L, x),
#  which=which(x==1L, arr.ind=TRUE),
#  times=10L
# )
# Unit: milliseconds
#  expr  min  lq  mean median  uq  max neval
# fpos   202  206  220    221  231  241    10
# which  612  637  667    653  705  724    10
```

iif *Fast if else*

Description

iif is a faster and more robust replacement of ifelse. It is comparable to dplyr::if_else, hutils::if_else and data.table::fifelse. It returns a value with the same length as test filled with corresponding values from yes, no or eventually na, depending on test. It does not support S4 classes.

Usage

```
iif(test, yes, no, na=NULL, tprom=FALSE, nThread=getOption("kit.nThread"))
```

Arguments

test
: A logical vector.

yes, no
: Values to return depending on TRUE/FALSE element of test. They must be the same type and be either length 1 or the same length of test.

na
: Value to return if an element of test is missing. It must be the same type as yes/no and be either length 1 or the same length of test. Please note that NA is treated as logical value of length 1 as per the R documentation. NA_integer_, NA_real_, NA_complex_ and NA_character_ are equivalent to NA but for integer, double, complex and character. Default value for argument na is NULL and will automatically default to the equivalent NA type of argument yes.

tprom
: Argument to indicate whether type promotion of yes and no is allowed or not. Either FALSE or TRUE, default is FALSE to not allow type promotion.

nThread
: A integer for the number of threads to use with *openmp*. Default value is getOption("kit.nThread").

Details

In contrast to ifelse attributes are copied from yes to the output. This is useful when returning Date, factor or other classes. Like dplyr::if_else and hutils::if_else, the na argument is by default set to NULL. This argument is set to NA in data.table::fifelse. Similarly to dplyr::if_else and when tprom=FALSE, iif requires same type for arguments yes and no. This is not strictly the case for data.table::fifelse which will coerce integer to double. When tprom=TRUE, iif behaviour is similar to base::ifelse in the sense that it will promote or coerce yes and noto the "highest" used type. Note, however, that unlike base::ifelse attributes are still conserved.

Value

A vector of the same length as test and attributes as yes. Data values are taken from the values of yes and no, eventually na.

Author(s)

Morgan Jacob

See Also

nif vswitch

Examples

```
x = c(1:4, 3:2, 1:4)
iif(x > 2L, x, x - 1L)

# unlike ifelse, iif preserves attributes, taken from the 'yes' argument
dates = as.Date(c("2011-01-01","2011-01-02","2011-01-03","2011-01-04","2011-01-05"))
ifelse(dates == "2011-01-01", dates - 1, dates)
iif(dates == "2011-01-01", dates - 1, dates)
yes = factor(c("a","b","c"))
no = yes[1L]
ifelse(c(TRUE,FALSE,TRUE), yes, no)
iif(c(TRUE,FALSE,TRUE), yes, no)

# Example of using the 'na' argument
iif(test = c(-5L:5L < 0L, NA), yes = 1L, no = 0L, na = 2L)

# Example of using the 'tprom' argument
iif(test = c(-5L:5L < 0L, NA), yes = 1L, no = "0", na = 2L, tprom = TRUE)
```

---

nif *Nested if else*

Description

nif is a fast implementation of SQL CASE WHEN statement for R. Conceptually, nif is a nested version of iif (with smarter implementation than manual nesting). It is not the same but it is comparable to dplyr::case_when and data.table::fcase.

Usage

```
nif(..., default=NULL)
```

Arguments

`...` A sequence consisting of logical condition (`when`)-resulting value (`value`) *pairs* in the following order `when1,value1,when2,value2,...,whenN,valueN`. Logical conditions `when1,when2,...,whenN` must all have the same length, type and attributes. Each `value` may either share length with `when` or be length 1. Please see Examples section for further details.

`default` Default return value, `NULL` by default, for when all of the logical conditions `when1,when2,...,whenN` are `FALSE` or missing for some entries. Argument `default` can be a vector either of length 1 or length of logical conditions `when1,when2,...,whenN`. Note that argument 'default' must be named explicitly.

Details

Unlike `data.table::fcase`, the `default` argument is set to `NULL`. In addition, `nif` can be called by other packages at C level. Note that at C level, the function has an additional argument `SEXP md` which is either `TRUE` for lazy evaluation or `FALSE` for non lazy evaluation. This argument is not exposed to R users and is more for C users.

Value

Vector with the same length as the logical conditions (`when`) in `...`, filled with the corresponding values (`value`) from `...`, or eventually `default`. Attributes of output values `value1,value2,...valueN` in `...` are preserved.

Author(s)

Morgan Jacob

See Also

`iif` `vswitch`

Examples

```
x = 1:10
nif(
x < 5L, 1L,
x > 5L, 3L
)

nif(
x < 5L, 1L:10L,
x > 5L, 3L:12L
)

# Lazy evaluation example
```

```
nif(
x < 5L, 1L,
x >= 5L, 3L,
x == 5L, stop("provided value is an unexpected one!")
)

# nif preserves attributes, example with dates
nif(
x < 5L, as.Date("2019-10-11"),
x > 5L, as.Date("2019-10-14")
)

# nif example with factor; note the matching levels
nif(
x < 5L, factor("a", levels=letters[1:3]),
x > 5L, factor("b", levels=letters[1:3])
)

# Example of using the 'default' argument
nif(
x < 5L, 1L,
x > 5L, 3L,
default = 5L
)

nif(
x < 5L, 1L,
x > 5L, 3L,
default = rep(5L, 10L)
)
```

psum/pprod/pmean/pall/pany

Sum, Product, Mean and more

Description

Similar to pmin and pmax but for sum, product and mean. Only works for integer, double and complex types. These functions do not recycle vectors. pany and pall are derived from base functions all and any. Note that pmean only works with integer and double types.

Usage

```
psum(..., na.rm=FALSE)
pprod(..., na.rm=FALSE)
pall(..., na.rm=FALSE)
pany(..., na.rm=FALSE)
pmean(..., na.rm=FALSE)
pcount(..., value)
```

Arguments

`...` Numeric arguments of type integer, double complex. Logical vector for `pall` and `pany`.

`na.rm` A logical indicating whether missing values should be removed. Default value is `FALSE`.

`value` A non `NULL` value of length 1. `pcount` will count how many times it occurs.

Value

Return the sum, product or mean of all numeric arguments. The value returned will be of the type of the highest argument types (integer < double < complex). For `pall` and `pany`, a logical vector is returned.

Author(s)

Morgan Jacob

Examples

```
x = c(1, 3, NA, 5)
y = c(2, NA, 4, 1)
z = c(3, 4, 4, 1)

# Example 1: psum
psum(x, y, z, na.rm = FALSE)
psum(x, y, z, na.rm = TRUE)

# Example 2: pprod
pprod(x, y, z, na.rm = FALSE)
pprod(x, y, z, na.rm = TRUE)

# Example 3: pmean
pmean(x, y, z, na.rm = FALSE)
pmean(x, y, z, na.rm = TRUE)

# Adjust x, y, and z to use in pall and pany
x = c(TRUE, FALSE, NA, FALSE)
y = c(TRUE, NA, TRUE, TRUE)
z = c(TRUE, TRUE, FALSE, NA)

# Example 4: pall
pall(x, y, z, na.rm = FALSE)
pall(x, y, z, na.rm = TRUE)

# Example 5: pany
pany(x, y, z, na.rm = FALSE)
pany(x, y, z, na.rm = TRUE)

# Example 6: pcount
pcount(x, y, z, value = TRUE)
```

```
# Benchmarks
# ----------
# n = 1e8L
# x = rnorm(n) # 763 Mb
# y = rnorm(n)
# z = rnorm(n)
#
# microbenchmark::microbenchmark(
#   kit=psum(x, y, z, na.rm = TRUE),
#   base=rowSums(do.call(cbind,list(x, y, z)), na.rm=TRUE),
#   times = 5L, unit = "s"
# )
# Unit: Second
# expr  min   lq mean median   uq  max neval
# kit  0.52 0.52 0.65   0.55 0.83 0.84     5
# base 2.16 2.27 2.34   2.35 2.43 2.49     5
#
# x = sample(c(TRUE, FALSE, NA), n, TRUE) # 382 Mb
# y = sample(c(TRUE, FALSE, NA), n, TRUE)
# z = sample(c(TRUE, FALSE, NA), n, TRUE)
#
# microbenchmark::microbenchmark(
#   kit=pany(x, y, z, na.rm = TRUE),
#   base=sapply(1:n, function(i) any(x[i],y[i],z[i],na.rm=TRUE)),
#   times = 5L
# )
# Unit: Second
# expr    min     lq   mean   median     uq    max neval
# kit    1.07   1.09   1.15     1.10   1.23   1.23     5
# base 111.31 112.02 112.78   112.97 113.55 114.03     5
```

setlevels *Set levels of a factor object*

Description

A function to set levels of a factor object.

Usage

```
setlevels(x, old=levels(x), new, skip_absent=FALSE)
```

Arguments

`x` A factor object.

`old` A character vector containing the factor levels to be changed. Default is levels of `x`.

`new` The new character vector containing the factor levels to be added.

`skip_absent` Skip items in `old` that are missing (i.e. absent) in ‘names(x)‘. Default `FALSE` halts with error if any are missing.

Value

Returns an invisible and modified factor object.

Author(s)

Morgan Jacob

Examples

```
x = factor(c("A", "A", "B", "B", "B", "C")) # factor vector with levels A B C
setlevels(x, new = c("X", "Y", "Z"))        # set factor levels to: X Y Z
setlevels(x, old = "X", new = "A")          # set factor levels X to A
```

topn *Top N values index*

Description

topn is used to get the indices of the few values of an input. This is an extension of which.max/which.min which provide *only* the first such index.

The output is the same as order(vec)[1:n], but internally optimized not to sort the irrelevant elements of the input (and therefore much faster, for small n relative to input size).

Usage

```
topn(vec, n=6L, decreasing=TRUE, hasna=TRUE)
```

Arguments

vec	A numeric vector of type numeric or integer. Other types are not supported yet.
n	A positive integer value greater or equal to 1. Maximum value is 1000.
decreasing	A logical value (default TRUE) to indicate whether to sort vec in decreasing or increasing order. Equivalent to argument decreasing in function base::order. Please note that unlike topn default value in base::order is FALSE.
hasna	A logical value (default TRUE) to indicate whether vec contains NA values.

Value

integer vector of indices of the most extreme (according to decreasing) n values in vector vec.

Author(s)

Morgan Jacob

Examples

```
x = rnorm(1e6)

# Example 1: index of top 6 negative values
topn(x, 6L, decreasing=FALSE)
order(x)[1:6]

# Example 2: index of top 6 positive values
topn(x, 6L, decreasing = TRUE)
order(x, decreasing=TRUE)[1:6]

# Example 3: top 6 negative values
x[topn(x, 6L, decreasing=FALSE)]
sort(x)[1:6]

# Benchmarks
# ----------
# x = rnorm(1e7) # 76Mb
# microbenchmark::microbenchmark(
#   topn=kit::topn(x, 6L),
#   order=order(x, decreasing=TRUE)[1:6],
#   times=10L
# )
# Unit: milliseconds
#  expr min   lq  mean median   uq  max neval
# topn   11   11    13     11   12   18    10
# order 563  565   587    566  602  661    10
#
# microbenchmark::microbenchmark(
#  topn=x[kit::topn(x, 6L, decreasing=FALSE)],
#  sort=sort(x, partial=1:6)[1:6],
#  times=10L
# )
# Unit: milliseconds
# expr min  lq  mean median   uq  max neval
# topn  11  11    11     11   12   12    10
# sort 167 175   197    178  205  303    10
```

`vswitch` *Vectorised switch*

Description

`vswitch` is a vectorised version of base function `switch`. This function can also be seen as a particular case of function `nif`, as shown in examples below, and should also be faster.

Usage

```
vswitch(x, values, outputs, default=NULL, nThread=getOption("kit.nThread"))
```

Arguments

`x` A vector or list.

`values` A vector or list with values from `x` to match. Note that `x` and `values` must have the same class and attributes.

`outputs` A list or vector with the outputs to return for every matching values. Each item of the list must be of length 1 or length of `x`. Note that if all list items are of length 1 then it might be simpler to use a vector.

`default` Values to return is no match. Must be a vector or list of length 1 or same length as `x`. Also, `default` must have the same type, class and aatributes as items from `outputs`.

`nThread` A integer for the number of threads to use with *openmp*. Default value is `getOption("kit.nThread")`.

Value

A vector or list of the same length as `x` with values from `outputs` items and from `default` if missing.

Author(s)

Morgan Jacob

See Also

`iif` `nif`

Examples

```
x = sample(c(10L, 20L, 30L, 40L, 50L, 60L), 3e2, replace=TRUE)

# The below example of 'vswitch' is
a = vswitch(
  x = x,
  values = c(10L,20L,30L,40L,50L),
  outputs = c(11L,21L,31L,41L,51L),
  default = NA_integer_
)

# equivalent to the following 'nif' example.
# However for large vectors 'vswitch' should be faster.
b = nif(
  x==10L, 11L,
  x==20L, 21L,
  x==30L, 31L,
  x==40L, 41L,
  x==50L, 51L,
  default = NA_integer_
)
identical(a, b)
```

```
# Example with list in 'outputs' argument
y = c(1, 0, NA_real_)
c = vswitch(
  x = y,
  values = c(1, 0),
  outputs = list(c(2, 3, 4), c(5, 6, 7)),
  default = 8
)

d = nif(
  y==1, c(2, 3, 4),
  y==0, c(5, 6, 7),
  default = 8
)

identical(c, d)

# Benchmarks
# ----------
# x = sample(1:100, 3e8, TRUE) # 1.1Gb
# microbenchmark::microbenchmark(
# nif=kit::nif(
#   x==10L,  0L,
#   x==20L, 10L,
#   x==30L, 20L,
#   default= 30L
#  ),
# vswitch=kit::vswitch(
#   x, c( 10L, 20L, 30L), list(0L, 10L, 20L), 30L
# ),
# times=10L
# )
# Unit: seconds
#    expr   min    lq  mean median   uq  max neval
# nif      4.27  4.37  4.43   4.42 4.52 4.53    10
# vswitch  1.08  1.09  1.20   1.10 1.43 1.44    10 # 1 thread
# vswitch  0.46  0.57  0.57   0.58 0.58 0.60    10 # 2 threads
```

Index