Package 'kit'

April 17, 2025

Type Package

Title Data Manipulation Functions Implemented in C

Version 0.0.20

Date 2025-04-17

Author Morgan Jacob [aut, cre, cph],
Sebastian Krantz [ctb]

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 2025-04-17 20:20:02 UTC

Contents

---

charToFact   *Convert Character Vector to Factor*

Description

Similar to `base::as.factor` but much faster and only for converting character vector to factor.

Usage

```
charToFact(x, decreasing=FALSE, addNA=TRUE,
          nThread=getOption("kit.nThread"))
```

Arguments

x	A vector of type character
decreasing	A boolean. Whether to order levels in decreasing order or not. Default is `FALSE`.
addNA	A boolean. Whether to include `NA` in levels of the output or not. Default is `TRUE`.
nThread	Number of thread to use.

Value

The character vector input as a factor. Please note that, unlike `as.factor`, NA levels are preserved by default, however this can be changed by setting argument `addNA` to `FALSE`.

Examples

```
x = c("b","A","B","a","\xe4","a")
Encoding(x) = "latin1"
identical(charToFact(x), as.factor(x))
identical(charToFact(c("a","b",NA,"a")), addNA(as.factor(c("a","b",NA,"a"))))
identical(charToFact(c("a","b",NA,"a"), addNA=FALSE), as.factor(c("a","b",NA,"a")))

# Benchmarks
# ----------
# x = sample(letters,3e7,TRUE)
# microbenchmark::microbenchmark(
#   kit=kit::charToFact(x,nThread = 1L),
#   base=as.factor(x),
#   times = 5L
# )
# Unit: milliseconds
# expr  min   lq   mean median   uq  max neval
# kit    188  190    196    194  200  208     5
# base 1402 1403   1455   1414 1420 1637     5
```

---

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. Function uniqLen is equivalent to base R
length(unique) or data.table::uniqueN.

Usage

```
fduplicated(x, fromLast = FALSE)
funique(x, fromLast = FALSE)
uniqLen(x)
```

Arguments

x   A vector, data.frame or matrix.

fromLast   A logical value to indicate whether the search should start from the end or be-
              ginning. Default is FALSE.

Value

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

Author(s)

Morgan Jacob

Examples

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

# Example 2: funique
funique(iris$Species)

# Example 3: uniqLen
uniqLen(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) 0.38 0.39  0.45    0.48 0.49  0.50     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))
#
# microbenchmark::microbenchmark(
#   kit=uniqLen(x),
#   data.table=uniqueN(x),
#   times = 5L, unit = "s"
```

```
# )
# Unit: seconds
#       expr  min    lq  mean  median    uq  max neval
# kit         0.17  0.17  0.17    0.17  0.17 0.17     5
# data.table 1.66  1.68  1.70    1.71  1.71 1.72     5 # (setDTthreads(1L))
# data.table 1.13  1.15  1.16    1.16  1.18 1.18     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 dimensions.
haystack	A matrix or vector to look into.
all	A logical value to indicate whether to return all occurrences (TRUE) or only the first one (FALSE). Default value is TRUE.
overlap	A logical value to indicate whether to allow the small matrix occurrences 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 behavior is similar to base::ifelse in the sense that it will promote or coerce yes and no to 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)
)
```

---

parallel-funs   *Parallel (Statistical) Functions*

Description

Vector-valued (statistical) functions operating in parallel over vectors passed as arguments, or a single list of vectors (such as a data frame). Similar to pmin and pmax, except that these functions

do not recycle vectors.

Usage

```
psum(..., na.rm = FALSE)
pprod(..., na.rm = FALSE)
pmean(..., na.rm = FALSE)
pfirst(...)  # (na.rm = TRUE)
plast(...)   # (na.rm = TRUE)
pall(..., na.rm = FALSE)
pallNA(...)
pallv(..., value)
pany(..., na.rm = FALSE)
panyNA(...)
panyv(..., value)
pcount(..., value)
pcountNA(...)
```

Arguments

`...`	suitable (atomic) vectors of the same length, or a single list of vectors (such as a `data.frame`). See Details on the allowed data types for each function, and Examples.
`na.rm`	A logical indicating whether missing values should be removed. Default value is `FALSE`, except for `pfirst` and `plast`.
`value`	A non `NULL` value of length 1.

Details

Functions `psum`, `pprod` work for integer, logical, double and complex types. `pmean` only supports integer, logical and double types. All 3 functions will error if used with factors.

`pfirst`/`plast` select the first/last non-missing value (or non-empty or `NULL` value for list-vectors). They accept all vector types with defined missing values + lists, but can only jointly handle integer and double types (not numeric and complex or character and factor). If factors are passed, they all need to have identical levels.

`pany` and `pall` are derived from base functions `all` and `any` and only allow logical inputs.

`pcount` counts the occurrence of `value`, and expects arguments of the same data type (except for `value = NA`). `pcountNA` is equivalent to `pcount` with `value = NA`, and they both allow `NA` counting in mixed-type data. `pcountNA` additionally supports list vectors and counts empty or `NULL` elements as `NA`.

Functions `panyv`/`pallv` are wrappers around `pcount`, and `panyNA`/`pallNA` are wrappers around `pcountNA`. They return a logical vector instead of the integer count.

None of these functions recycle vectors i.e. all input vectors need to have the same length. All functions support long vectors with up to `2^64-1` elements.

Value

psum/pprod/pmean return the sum, product or mean of all arguments. The value returned will be of the highest argument type (integer < double < complex). pprod only returns double or complex. pall[v/NA] and pany[v/NA] return a logical vector. pcount[NA] returns an integer vector. pfirst/plast return a vector of the same type as the inputs.

Author(s)

Morgan Jacob and Sebastian Krantz

See Also

Package 'collapse' provides column-wise and scalar-valued analogues to many of these functions.

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)

# Example 4: pfirst and plast
pfirst(x, y, z)
plast(x, y, z)

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

# Example 6: pany
pany(x, y, z, na.rm = FALSE)
pany(x, y, z, na.rm = TRUE)

# Example 7: pcount
pcount(x, y, z, value = TRUE)
pcountNA(x, y, z)
```

```
# Example 8: list/data.frame as an input
pprod(iris[,1:2])
psum(iris[,1:2])
pmean(iris[,1:2])

# 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
```

---

psort   *Parallel Sort*

Description

Similar to base::sort but just for character vector and partially using parallelism. It is currently experimental and might change in the future. Use with caution.

Usage

```
psort(x, decreasing=FALSE, na.last=NA,
      nThread=getOption("kit.nThread"),c.locale=TRUE)
```

Arguments

x	A vector of type character. If other, it will default to base::sort
na.last	For controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed.
decreasing	A boolean indicating where to sort the data in decreasing way. Default is FALSE.
nThread	Number of thread to use. Default value is 1L.
c.locale	A boolean, whether to use C Locale or R session locale. Default TRUE.

Value

Returns the input x in sorted order similar to base::sort but usually faster. If c.locale=FALSE, psort will return the same output as base::sort with method="quick", i.e. using R session locale. If c.locale=TRUE, psort will return the same output as base::sort with method="radix", i.e. using C locale. See example below.

Author(s)

Morgan Jacob

Examples

```
x = c("b","A","B","a","\xe4")
Encoding(x) = "latin1"
identical(psort(x, c.locale=FALSE), sort(x))
identical(psort(x, c.locale=TRUE), sort(x, method="radix"))

# Benchmarks
# ----------
# strings = as.character(as.hexmode(1:1000))
# x = sample(strings, 1e8, replace=TRUE)
# system.time({kit::psort(x, na.last = TRUE, nThread = 1L)})
#   user  system elapsed
# 2.833   0.434   3.277
# system.time({sort(x,method="radix",na.last = TRUE)})
#   user  system elapsed
# 5.597   0.559   6.176
# system.time({x[order(x,method="radix",na.last = TRUE)]})
#   user  system elapsed
# 5.561   0.563   6.143
```

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
```

---

shareData/getData/clearData

Share Data between R Sessions

Description

Experimental functions that enable the user to share a R object between 2 R sessions.

Usage

```
shareData(data, map_name, verbose=FALSE)
getData(map_name, verbose=FALSE)
clearData(x, verbose=FALSE)
```

Arguments

data	A R object like a vector or a data.frame.
map_name	A character. A name for the memory map location where to store the data.
x	An external pointer like the one returned by function shareData.
verbose	A logical value TRUE or FALSE to provide or not information to the user.

Value

shareData returns a external pointer. getData returns an R object stored in the memory location map_name. clearData returns TRUE or FALSE depending on whether the data have been cleared in memory.

Author(s)

Morgan Jacob

Examples

```
# In R session 1: share data in memory
# > x = shareData(mtcars,"share1")
#
# In R session 2: get data from session 1
# > getData("share1")
#
# In R session 1: clear data in memory
# > clearData(x)
```

---

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, index=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.
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.
index	A logical value (default TRUE) to indicate whether indexes or values of vec.

Value

integer vector of indices of the most extreme (according to `decreasing`) n values in vector vec. Please note that for large value of n, i.e. 1500 or 2000 (depending on the value of hasna), `topn` will default to base R function `order`.

Author(s)

Morgan Jacob

Examples

```
x = rnorm(1e4)

# 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
topn(x, 6L, decreasing=FALSE, index=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=kit::topn(x, 6L, decreasing=FALSE, index=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/nswitch   *Vectorised switch*

Description

vswitch/ nswitch 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"),
        checkEnc=TRUE)
nswitch(x, ..., default=NULL,
        nThread=getOption("kit.nThread"),
        checkEnc=TRUE)
```

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.
...	A sequence of values and outputs in the following order value1, output1, value2, output2, ..., valueN, outputN. Values value1, value2, ..., valueN must all have length1, same type and attributes. Each output may either share length with x or be length 1. Please see Examples section for further details.
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 attributes as items from outputs.
nThread	A integer for the number of threads to use with *openmp*. Default value is getOption("kit.nThread").
checkEnc	A logical value whether or not to check if x and values have comparable and consistent encoding. Default is TRUE.

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
a1 = 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.
b1 = nif(
  x==10L, 11L,
  x==20L, 21L,
  x==30L, 31L,
  x==40L, 41L,
  x==50L, 51L,
  default = NA_integer_
)
identical(a1, b1)

# nswitch can also be used as follows:
c1 = nswitch(x,
  10L, 11L,
  20L, 21L,
  30L, 31L,
  40L, 41L,
  50L, 51L,
  default = NA_integer_
)
identical(a1, c1)

# Example with list in 'outputs' argument
y = c(1, 0, NA_real_)
a2 = vswitch(
  x = y,
  values = c(1, 0),
  outputs = list(c(2, 3, 4), c(5, 6, 7)),
  default = 8
)

b2 = nif(
  y==1, c(2, 3, 4),
  y==0, c(5, 6, 7),
  default = 8
)
```

```
identical(a2, b2)

c2 = nswitch(y,
  1, c(2, 3, 4),
  0, c(5, 6, 7),
  default = 8
)

identical(a2, c2)

# 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