Package 'grr'

October 13, 2022

Title Alternative Implementations of Base R Functions

Version 0.9.5

Author Craig Varrichio <`canthony427@gmail.com`>

Maintainer Craig Varrichio <`canthony427@gmail.com`>

Description Alternative implementations of some base R functions, including sort, order, and match. Functions are simplified but can be faster or have other advantages.

Depends R (>= 3.0.0)

License GPL-3

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-08-26 20:35:38

R topics documented:

convertBase *Convert string representations of numbers in any base to any other base.*

Description

Convert string representations of numbers in any base to any other base.

Usage

```
convertBase(x, base1 = 10, base2 = 10)
```

Arguments

x	a vector of integers or strings to be converted
base1	the base of x
base2	the base of the output

See Also

as.octmode

as.hexmode

strtoi

Examples

```

identical(convertBase(1234,base2=8),as.character(as.octmode(1234)))

convertBase(17771,base1=8,base2=30)
convertBase(17771,base1=8,base2=10)
convertBase(8185,base1=10,base2=30)
```

extract *Extract/return parts of objects*

Description

Alternative to built-in `Extract` or `[`. Allows for extraction operations that are ambivalent to the data type of the object. For example, `extract(x,i)` will work on lists, vectors, data frames, matrices, etc.

Usage

```
extract(x, i = NULL, j = NULL)
```

Arguments

`x`	object from which to extract elements
`i, j`	indices specifying elements to extract. Can be `numeric`, `character`, or `logical` vectors.

Details

Extraction is 2-100x faster on data frames than with the built in operation - but does not preserve row names.

Examples

```
#Typically about twice as fast on normal subselections
orders<-data.frame(orderNum=1:1e5,
 sku=sample(1e3, 1e5, TRUE),
 customer=sample(1e4,1e5,TRUE))
a<-sample(1e5,1e4)
system.time(b<-orders[a,])
system.time(c<-extract(orders,a))
rownames(b)<-NULL
rownames(c)<-NULL
identical(b,c)

#Speedup increases to 50-100x with oversampling
a<-sample(1e5,1e6,TRUE)
system.time(b<-orders[a,])
system.time(c<-extract(orders,a))
rownames(b)<-NULL
rownames(c)<-NULL
identical(b,c)

#Can create function calls that work for multiple data types
alist<-as.list(1:50)
avector<-1:50
extract(alist,1:5)
extract(avector,1:5)
extract(orders,1:5)#'

## Not run:
orders<-data.frame(orderNum=as.character(sample(1e5, 1e6, TRUE)),
 sku=sample(1e3, 1e6, TRUE),
 customer=sample(1e4,1e6,TRUE))
system.time(a<-sample(1e6,1e7,TRUE))
system.time(b<-orders[a,])
system.time(c<-extract(orders,a))

## End(Not run)
```

`grr` *Alternative Implementations of Base R Functions*

Description

Alternative implementations of some base R functions, including sort, order, and match. Functions are simplified but can be faster or have other advantages. See the documentation of individual functions for details and benchmarks.

Details

Note that these functions cannot be considered drop-in replacements for the functions in base `R`. They do not implement all the same parameters and do not work for all data types. Utilize these with caution in specialized applications that require them.

`matches` *Value Matching*

Description

Returns a lookup table or list of the positions of ALL matches of its first argument in its second and vice versa. Similar to `match`, though that function only returns the first match.

Usage

```
matches(x, y, all.x = TRUE, all.y = TRUE, list = FALSE, indexes = TRUE,
  nomatch = NA)
```

Arguments

`x`	vector. The values to be matched. Long vectors are not currently supported.
`y`	vector. The values to be matched. Long vectors are not currently supported.
`all.x`	logical; if `TRUE`, then each value in `x` will be included even if it has no matching values in `y`
`all.y`	logical; if `TRUE`, then each value in `y` will be included even if it has no matching values in `x`
`list`	logical. If `TRUE`, the result will be returned as a list of vectors, each vector being the matching values in y. If `FALSE`, result is returned as a data frame with repeated values for each match.
`indexes`	logical. Whether to return the indices of the matches or the actual values.
`nomatch`	the value to be returned in the case when no match is found. If not provided and `indexes=TRUE`, items with no match will be represented as `NA`. If set to `NULL`, items with no match will be set to an index value of `length+1`. If indexes=FALSE, they will default to `NA`.

Details

This behavior can be imitated by using joins to create lookup tables, but `matches` is simpler and faster: usually faster than the best joins in other packages and thousands of times faster than the built in `merge`.

`all.x/all.y` correspond to the four types of database joins in the following way:

left `all.x=TRUE`, `all.y=FALSE`

right `all.x=FALSE`, `all.y=TRUE`

inner `all.x=FALSE`, `all.y=FALSE`

full `all.x=TRUE`, `all.y=TRUE`

Note that `NA` values will match other `NA` values.

Examples

```
one<-as.integer(1:10000)
two<-as.integer(sample(1:10000,1e3,TRUE))
system.time(a<-lapply(one, function (x) which(two %in% x)))
system.time(b<-matches(one,two,all.y=FALSE,list=TRUE))

#Only retain items from one with a match in two
b<-matches(one,two,all.x=FALSE,all.y=FALSE,list=TRUE)
length(b)==length(unique(two))

one<-round(runif(1e3),3)
two<-round(runif(1e3),3)
system.time(a<-lapply(one, function (x) which(two %in% x)))
system.time(b<-matches(one,two,all.y=FALSE,list=TRUE))

one<-as.character(1:1e5)
two<-as.character(sample(1:1e5,1e5,TRUE))
system.time(b<-matches(one,two,list=FALSE))
system.time(c<-merge(data.frame(key=one),data.frame(key=two),all=TRUE))

## Not run:
one<-as.integer(1:1000000)
two<-as.integer(sample(1:1000000,1e5,TRUE))
system.time(b<-matches(one,two,indexes=FALSE))
if(requireNamespace("dplyr",quietly=TRUE))
 system.time(c<-dplyr::full_join(data.frame(key=one),data.frame(key=two)))
if(require(data.table,quietly=TRUE))
 system.time(d<-merge(data.table(data.frame(key=one))
             ,data.table(data.frame(key=two))
             ,by='key',all=TRUE,allow.cartesian=TRUE))

one<-as.character(1:1000000)
two<-as.character(sample(1:1000000,1e5,TRUE))
system.time(a<-merge(one,two)) #Times out
system.time(b<-matches(one,two,indexes=FALSE))
if(requireNamespace("dplyr",quietly=TRUE))
```

```
 system.time(c<-dplyr::full_join(data.frame(key=one),data.frame(key=two)))#'
if(require(data.table,quietly=TRUE))
{
 system.time(d<-merge(data.table(data.frame(key=one))
             ,data.table(data.frame(key=two))
             ,by='key',all=TRUE,allow.cartesian=TRUE))
 identical(b[,1],as.character(d$key))
}

## End(Not run)
```

`order2` *Ordering vectors*

Description

Simplified implementation of `order`. For large vectors, typically is about 3x faster for numbers and 20x faster for characters.

Usage

```
order2(x)
```

Arguments

`x`	a vector of class numeric, integer, character, factor, or logical. Long vectors are not supported.

Examples

```
chars<-as.character(sample(1e3,1e4,TRUE))
system.time(a<-order(chars))
system.time(b<-order2(chars))
identical(chars[a],chars[b])

ints<-as.integer(sample(1e3,1e4,TRUE))
system.time(a<-order(ints))
system.time(b<-order2(ints))
identical(ints[a],ints[b])

nums<-runif(1e4)
system.time(a<-order(nums))
system.time(b<-order2(nums))
identical(nums[a],nums[b])

logs<-as.logical(sample(0:1,1e6,TRUE))
system.time(a<-order(logs))
system.time(b<-order2(logs))
identical(logs[a],logs[b])
```

```
facts<-as.factor(as.character(sample(1e3,1e4,TRUE)))
system.time(a<-order(facts))
system.time(b<-order2(facts))
identical(facts[a],facts[b])

#How are special values like NA and Inf handled?
#For numerics, values sort intuitively, with the important note that NA and
#NaN will come after all real numbers but before Inf.
(function (x) x[order2(x)])(c(1,2,NA,NaN,Inf,-Inf))
#For characters, values sort correctly with NA at the end.
(function (x) x[order2(x)])(c('C','B',NA,'A'))
#For factors, values sort correctly with NA at the end.
(function (x) x[order2(x)])(as.factor(c('C','B',NA,'A')))


#Ordering a data frame using order2
df<-data.frame(one=as.character(1:4e5),
   two=sample(1:1e5,4e5,TRUE),
   three=sample(letters,4e5,TRUE),stringsAsFactors=FALSE)
system.time(a<-df[order(df$one),])
system.time(b<-df[order2(df$one),])
system.time(a<-df[order(df$two),])
system.time(b<-df[order2(df$two),])

## Not run:
chars<-as.character(sample(1e5,1e6,TRUE))
system.time(a<-order(chars))
system.time(b<-order2(chars))

ints<-as.integer(sample(1e5,1e6,TRUE))
system.time(result<-order(ints))
system.time(result<-order2(ints))

nums<-runif(1e6)
system.time(result<-order(nums))
system.time(result<-order2(nums))

logs<-as.logical(sample(0:1,1e7,TRUE))
system.time(result<-order(logs))
system.time(result<-order2(logs))

facts<-as.factor(as.character(sample(1e5,1e6,TRUE)))
system.time(a<-order(facts))
system.time(b<-order2(facts))
identical(facts[a],facts[b])



## End(Not run)
```

sample2 *A wrapper for `sample.int` and `extract` that makes it easy to quickly sample rows from any object, including Matrix and sparse matrix objects.*

Description

Row names are not preserved.

Usage

```
sample2(x, size, replace = FALSE, prob = NULL)
```

Arguments

`x`	object from which to extract elements
`size`	a positive number, the number of items to choose.
`replace`	Should sampling be with replacement?
`prob`	A vector of probability weights for obtaining the elements of the vector being sampled.

Examples

```
#Sampling from a list
l1<-as.list(1:1e6)
b<-sample2(l1,1e5)

#Sampling from a data frame
orders<-data.frame(orderNum=sample(1e5, 1e6, TRUE),
   sku=sample(1e3, 1e6, TRUE),
   customer=sample(1e4,1e6,TRUE),stringsAsFactors=FALSE)

a<-sample2(orders,250000)

#With oversampling sample2 can be much faster than the alternatives,
#with the caveat that it does not preserve row names.
system.time(a<-sample2(orders,2000000,TRUE))
system.time(b<-orders[sample.int(nrow(orders),2000000,TRUE),])
## Not run:

system.time(c<-dplyr::sample_n(orders,2000000,replace=TRUE))

#Can quickly sample for sparse matrices while preserving sparsity
sm<-rsparsematrix(20000000,10000,density=.0001)
sm2<-sample2(sm,1000000)

## End(Not run)
```

sort2 *Sorting vectors*

Description

Simplified implementation of `sort`. For large vectors, typically is about 2x faster for numbers and 20x faster for characters and factors.

Usage

```
sort2(x)
```

Arguments

`x`	a vector of class numeric, integer, character, factor, or logical. Long vectors are not supported.

Examples

```
chars<-as.character(sample(1e3,1e4,TRUE))
system.time(a<-sort(chars))
system.time(b<-sort2(chars))
identical(a,b)

ints<-as.integer(sample(1e3,1e4,TRUE))
system.time(a<-sort(ints))
system.time(b<-sort2(ints))
identical(a,b)

nums<-runif(1e4)
system.time(a<-sort(nums))
system.time(b<-sort2(nums))
identical(a,b)

logs<-as.logical(sample(0:1,1e6,TRUE))
system.time(result<-sort(logs))
system.time(result<-sort2(logs))

facts<-as.factor(as.character(sample(1e3,1e4,TRUE)))
system.time(a<-sort(facts))
system.time(b<-sort2(facts))
identical(a,b)

#How are special values like NA and Inf handled?
#For numerics, values sort intuitively, with the important note that NA and
#NaN will come after all real numbers but before Inf.
sort2(c(1,2,NA,NaN,Inf,-Inf))
#For characters, values sort correctly with NA at the end.
sort2(c('C','B',NA,'A'))
#For factors, values sort correctly with NA at the end
```

```
sort2(as.factor(c('C','B',NA,'A')))

## Not run:
chars<-as.character(sample(1e5,1e6,TRUE))
system.time(a<-sort(chars))
system.time(b<-sort2(chars))

ints<-as.integer(sample(1e5,1e6,TRUE))
system.time(result<-sort(ints))
system.time(result<-sort2(ints))

nums<-runif(1e6)
system.time(result<-sort(nums))
system.time(result<-sort2(nums))

logs<-as.logical(sample(0:1,1e7,TRUE))
system.time(result<-sort(logs))
system.time(result<-sort2(logs))

facts<-as.factor(as.character(sample(1e5,1e6,TRUE)))
system.time(a<-sort(facts))
system.time(b<-sort2(facts))

## End(Not run)
```

Index