Package ‘rindex’

August 31, 2018

Type Package
Title Indexing for R
Version 0.12.1
Date 2012-08-24
Author Jens Oehlschlägel <Jens.Oehlschlaegel@truecluster.com>
Maintainer Jens Oehlschlägel <Jens.Oehlschlaegel@truecluster.com>
Depends R (>= 2.6.0), regtest
Description Index structures allow quickly accessing elements from large collections. While btree are optimized for disk databases and ttree for ram databases we use hybrid static indexing which is quite optimal for R.
License file LICENSE
Encoding latin1
Repository CRAN
Date/Publication 2018-08-31 17:40:10 UTC
NeedsCompilation yes
License_restricts_use yes

R topics documented:

index ................................................................. 2
indexBytes .......................................................... 8
indexDemoClose ...................................................... 10
indexEQ ............................................................ 11
indexFind .......................................................... 12
indexFindInterval .................................................. 13
indexInit .......................................................... 14
indexMatch ........................................................ 14
match ............................................................... 15
Operators.index ..................................................... 16
sort.index .......................................................... 17
strncmp ............................................................. 18
Indexing vectors

Description

Indexing allows to extract small fractions of large vectors very quickly.

Usage

```r
index(x, uni = NULL, batch = NULL, verbose = FALSE)
## S3 method for class 'index'
c(...)
indexAddTree(obj, batch = NULL)
indexDelTree(obj)
indexAutobatch(n, batch = 64)
```

Arguments

- `x`: a vector (currently only character)
- `uni`: set to TRUE or FALSE to save checking for duplicates (default NULL for checking)
- `batch`: set to preferred batch size to influence RAM – speed – trade-off (default NULL for `indexAutobatch`)
- `verbose`: set to TRUE to report timing of index creation (default FALSE)
- `n`: number of elements to build a tree on
- `obj`: an object of class `index`
- `...`: objects of class `index`

Details

Basic functions

- `index`: creating, modifying and removing indices
- `c.index`: concatenate index objects (currently not tuned)
- `indexAutobatch`: calculates the optimal index resolution (size of a leaf) given number of elements and desired batch size
- `indexAddTree`: build or rebuild tree (`calloc`)
- `indexDelTree`: remove tree (`Free`)
- `rm`: removing index object removes tree at next garbage collection `gc`

Index information

- `indexNodes`: information, printing and retrieving all values
- `indexBytes`: returns index size in bytes
- `print.index`: prints index info and optionally tree
- `str.index`: removes class and calls `NextMethod("str")`
- `length.index`: identical to `length` of original vector
- `names.index`: currently forbidden
names<-.index currently forbidden

**Basic access**

- `sort.index` identical to `sort` of original vector, but much faster
- `order.index` identical to `order` of original vector, but much faster
- `[]` `index[]` returns original vector, subsetting works identical to susetting original vector [ via `NextMethod()`]
- `[[` `index[[` returns original vector, subsetting works identical to susetting original vector

**is.na.index** identical to `is.na` of original vector, but much faster

**Low level search**

- `indexFind` low level search functions return positions in index order (sorted)
- `indexFindlike` finding values in index that begin like search value (character indices only)

**Mid level search**

- `indexFindInterval` finding a sequence of exact or approximate values
- `indexMatch` finding positions of vector of search values

**High level search**

- `indexEQ` high level search functions return positions in original order (unsorted)
- `indexNE` index NotEqual value
- `indexLT` index LowerThan value
- `indexGT` index GreaterThan value
- `indexLE` index LowerEqual value
- `indexGE` index GreaterEqual value

**High level operators**

- `==.index` high level operators return logical vectors in original order (unsorted)
- `!=.index` index NotEqual value
- `<.index` index LowerThan value
- `>.index` index GreaterThan value
- `<=.index` index LowerEqual value
- `>=.index` index GreaterEqual value
- `match` high level matching
- `match.index` use this to match in an index instead of `match`
- `match.rindex` use this to match in a rindex instead of `match`

**Value**

An object of class ‘index’, i.e. a list with components

- `val` sorted vector of values
- `pos` integer vector of original positions of sorted values
- `n` number of values (including NAs)
- `nNA` number of NAs
- `batch` resolution of tree
- `uni` logical flagging the index as unique (TRUE) or non-unique (FALSE)
- `tree` external pointer to C-tree
Theory

Linear search has $O(n)$ time complexity. By using an index tree, search time can be reduced to $O(\log(n))$. An index that can be used with any R-vector $x$ of length $n$ needs to store the original values together with the original positions, i.e. `list(val=x, pos=order(x))`, thus requires – strongly simplified – $2 \times n$ RAM. If we store this information in a binary tree, each value pair `{val, pos}` is stored in its own node together with two pointers, the memory requirements are – strongly simplified – $4 \times n$ RAM. The b-tree stores more than one value and two pointers in one node and thus minimizes the number of nodes that need to be read (from disk). However, used in RAM, b-trees increase the search time because they impurify logarithmic search across nodes with linear search within nodes. By contrast, the t-tree is optimized for RAM: it stores only two pointers but many sorted values within each node: for branching only min and max values need to be searched, linear search is only required at the final leaf node. However, realizing a t-tree within R’s memory model requires additional overhead for the SEXPREC data structures. We avoid that by defining a static read only tree (and save implementation of insert and delete operations). The $b^*$tree and $t^*$tree versions of the mentioned indices reduce the size of the search nodes by storing the data itself in the leafnodes only. This leads to some redundant storage but speeds up search. Some implementations connect the leafnodes by extra pointers to speed up linear search. If we take this principle to the extreme, we can save these extra pointers and merge all leave nodes into one single big leaf. By doing so we loose the ability to update the index, but we gain a static read-only tree structure that supports very fast linear search as well as logarithmic search.

```
  / \
 /   / C-tree
/   /   R-val
\------------- R-pos
\-------------
```

We implement this efficiently by storing the sorted values vector together with the order positions as standard R (SEXPREC) objects and add a pure C tree that is built from pointered struct nodes. The leaf nodes do use integer addressing instead of pointers to identify the associated part of the SEXPREC vector (pointers can’t be used because the R garbage collector may move the vector). The topnode is linked into R using an external pointer. The tree itself can be removed explicitly from memory using `indexDeleteTree`. If the index object containing the external pointer is deleted, the tree will be freed from memory at the next garbage collection through its finalization method. If the index object does not contain a valid external pointer to the tree – e.g. when loading an index object from disk – the tree will be quickly transparently rebuild or can be build explicitly via `indexAddTree`.

Benefits

- much faster access to small fractions of large vectors (or few rownames of long data.frames)
- index allows and handles duplicated values
- index allows and handles NAs (unique version of index can still have more than one NA)
- index sort order is identical to (implemented via) `order` with `na.last=TRUE`
- sorted values and original positions are stored simply as standard R vectors
values in original sequence can be retrieved via \[ \text{from the index (original vector not required, thus no duplication of values)} \]

- several vector methods give expected results (\text{length, is.na, sort, order, [}, \text{Comparison operators)}
- the tree needs minimum \text{RAM due to lean C structs (index is close to 2*n)}
- the tree resolution can be user-defined (\text{RAM – speed – trade-off)}
- the tree will not consume disk space when the index is saved
- the tree is build up transparently when needed the first time (e.g. after loading the index from disk)
- the tree is build up very quickly because the main work – the sorting – needs not to be repeated
- the tree is removed transparently from the garbage collector when the index is deleted
- this type of tree should perfectly complement the new disk vectors (packages \text{ff} and \text{R.huge})

\text{Limitations}

- currently indexing is only available for character vectors (not logical, integer, real, complex)
- currently indices cannot be stored as columns of data frames
- functions that want to retrieve the index values in original order need to call \text{indexname[]} if the complete vector is needed
- building the index is rather slow (as slow as sorting in \text{R is, building the tree is very fast)}
- like all indices the index will reduce performance if a large fraction of the vector is accessed.
- not all methods have been maximally tuned via \text{C-code (e.g. c.index\text{).}}

\text{Open questions}

- Will \text{R core officially export C entry points to UTF-8 resistant strcmp (STRCOLL) and strncmp (nothing yet)?}
- Shall we make \text{order()} generic to allow dispatch of \text{order.index} like we have already dispatch of \text{sort.index}\text{?}
- Can we call the evaluator from \text{C in order to make }\text{<>comparisons independent of atomic mode (UTF-8 etc.) or will this kill too much performance?}
- Can we call the evaluator from \text{C for [-accessing the vector elements in order to generalize the index to ff or R.huge or will this kill too much performance?}

\text{Note}

For each \text{F00indexF00} related to the ‘index’ class a function \text{F00rindexF00} exists related to the ‘rindex’ class. The ‘rindex’ class is a pure \text{R-prototype and is kept for regression-testing using binregtest from package regtest.} The regression tests are in \text{dontshow sections in the examples of this help. If you run example(index) the regression tests will (unavoidably) trigger warnings.}

\text{Author(s)}

Jens Oehlschlägel
References


See Also

`order`, `sort`, `match`

Examples

```r
#library(rindex)
x <- sample(c(rep("c", 5), paste("d", 1:5, sep=""), c(letters[c(1,2,5,6)], NA)))

cat("\n")
cat("creating an index from atomic vector\n")
i <- index(x)
i
cat("creating an index by combining indices\n")
i2 <- c(i, i)
i2
cat("if the index (or index$tree) is removed, the C-tree is removed at the next garbage collection\n")
cat("the index tree can also removed and created explicitly\n")
i <- indexDelTree(i)
i
i <- indexAddTree(i, batch=3)
print(i, tree=TRUE)
indexNodes(i)
indexBytes(i)

cat("\n")
cat("extracting the original vector\n")
i[]
cat("subsetting works as expected\n")
i[1:3]
cat("accessing the sorted data is much faster\n")
sort(i)[1:3]
cat("accessing the ordering is also much faster (order.index is not dispatched since order is not yet generic)\n")
order.index(i)[1:3]

identical(is.na(i), is.na(x))
identical(length(i), length(x))

cat("\n")
cat("LOW LEVEL SEARCH returns position in SORTED VECTOR\n")
cat("low level search for position of lowest instance of value in the index\n")
indexFind(i, "c")
cat("low level search for position of highest instance of value in the index\n")
indexFind(i, "c", findLow=FALSE)
cat("low level search for position of lowest instance beginning like search value\n")
```
indexFindlike(i,"d")
cat("low level search for position of highest instance beginning like search value\n")
indexFindlike(i,"d",findlow=FALSE)

cat("\n")
cat("MID LEVEL SEARCH also returns position in SORTED VECTOR\n")
cat("mid level search for a set of values\n")
indexMatch(i,c("c","f"), findlow=TRUE)  # giving parameter findlow= suppresses the warning issued on non-unique
sort(i)[indexMatch(i,c("c","f"), findlow=TRUE)]
indexMatch(i,c("c","f"), findlow=TRUE, what="pos")
indexMatch(i,c("c","f"), findlow=TRUE, what="val")
indexMatch(i,c("c","f"), findlow=FALSE, what="pos")

cat("mid level search for interval of values\n")
indexFindInterval(i,"b","f")
cat("by default the searched endpoints are included\n")
sort(i)[indexFindInterval(i,"b","f")]  # but they can be excluded
sort(i)[indexFindInterval(i,"b","f",high.include=FALSE)]
cat("by default the searched endpoints need not to be present\n")
sort(i)[indexFindInterval(i,"a1","e1")]  # but this can be required
sort(i)[indexFindInterval(i,"a1","e1",low.exact=TRUE)]
cat("each of the searched endpoints can be defined via indexFindlike\n")
sort(i)[indexFindInterval(i,"c","d",FUN=indexFindlike)]

cat("\n")
cat("HIGH LEVEL SEARCH returns POSITION(s) IN ORIGINAL VECTOR but in SEQUENCE OF INDEX\n")
indexEQ(i,"d3")
indexNE(i,"d3")
indexLT(i,"d3")
indexLE(i,"d3")
indexGT(i,"d3")
indexGE(i,"d3")
cat("searching for several values returns a list\n")
indexEQ(i,c("b","c","z",NA))
indexEQ(i,c("b","c","z",NA), what="val")

cat("\n")
cat("HIGH LEVEL OPERATORS returns TRUE/FALSE AT ORIGINAL VECTOR POSITIONS\n")
i=="d3"  # i="d3"
i!="d3"  # i<"d3"
i<="d3"  # i>"d3"
i>="d3"  # i<"d3"

cat("HIGH LEVEL match.index and behave as expected\n")
match(c("b","c","z",NA), x)
match.index(c("b","c","z",NA), i)
## Not run:

```r
cat("function timefactor helps with timing\n")

n <- 1000000
x <- sample(1:n)
names(x) <- paste("a", x, sep="")
d <- data.frame(x=as.vector(x), row.names=names(x))

nsub <- 100
i <- sample(1:n, nsub)
ni <- names(x)[1]

ind <- index(names(x), verbose=TRUE)
ind

# test vectors
cat("character susetting is by magnitude slower than integer subsettting\n")
timefactor( x[ni] , x[i] , 10, 10000)
cat("character susetting is approx as slow as matching\n")
timefactor( x[ni] , x[match(ni,names(x))] , 10, 10)
cat("for small fractions of n indexing is much faster\n")
timefactor( x[ni] , x[indexMatch(ind,ni)] , 10, 100)

# test dataframes
cat("character susetting is by magnitude slower than integer subsettting\n")
timefactor( d[ni,] , d[i,] , 1, 100)
cat("obvious implementation problem (in R-2.5.1 susetting is much slower than via matching)\n")
timefactor( d[ni,] , d[match(ni,rownames(d))], , 1, 1)
cat("for small fractions of n indexing is much faster\n")
timefactor( d[ni,] , d[indexMatch(ind,ni)],[, 1, 10)
```

## End(Not run)

---

### indexBytes

**Index information**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some functions giving information about indices</td>
</tr>
</tbody>
</table>
Usage

indexNodes(obj)
indexBytes(obj)
#
S3 method for class 'index'
print(x, tree = FALSE, ...)
#
S3 method for class 'index'
str(object, ...)
#
S3 method for class 'index'
length(x)

Arguments

x an object of class ‘index’
obj an object of class ‘index’
object an object of class ‘index’
tree TRUE to print the tree (default FALSE)
... ignored or passed

Details

indexNodes returns number of tree nodes
indexBytes returns index size in bytes
print.index prints index info and optionally tree
str.index removes class and calls NextMethod("str")
length.index identical to length of original vector
names.index currently forbidden
names<-.index currently forbidden

Value

Functions indexNodes, indexBytes and length.index return an integer (number of nodes, number of bytes, length of vector).

Note

There are dummy functions names.index, names<-.index and [<-.index that catch non-supported use of these generics on index objects.

Author(s)

Jens Oehlschlägel
indexDemoClose

Demo functions for creating and removing external pointers

Description

Function indexDemoOpen creates an external pointer pointing to an integer vector of length ‘n’, indexDemoClose frees the memory linked to the pointer.

Usage

indexDemoOpen(n = 10)
indexDemoClose(extPtr)

Arguments

extPtr the external pointer returned by indexDemoOpen
n the length of the vector stored in C RAM

Details

If the returned pointer is removed finalization happens at the next gc(), if indexDemoClose is used on the pointer then finalization happens immediately, if the pointer is then removed, the finalizer is called again at the next gc() (but doesn’t finalize again).

Value

Function indexDemoOpen returns an external pointer, indexDemoClose returns the integer vector previously linked by the pointer.

Author(s)

Jens Oehlschlägel

References

R Development Core Team (2007). Writing R Extensions.

See Also

indexInit, indexDone, indexAddTree, indexDelTree
indexEQ

Examples

```r
ptr <- indexDemoOpen()
rm(ptr)
gc()

ptr <- indexDemoOpen()
indexDemoClose(ptr)
rm(ptr)
gc()
```

---

**indexEQ**  
*High level comparison function*

**Description**

Compare index against value

**Usage**

```r
indexEQ(obj, x, what = c("pos", "val", "ind"), ...)
indexNE(obj, x, what = c("pos", "val", "ind"), ...)
indexLT(obj, x, what = c("pos", "val", "ind"), ...)
indexLE(obj, x, what = c("pos", "val", "ind"), ...)
indexGT(obj, x, what = c("pos", "val", "ind"), ...)
indexGE(obj, x, what = c("pos", "val", "ind"), ...)
```

**Arguments**

- `obj` an object of class ‘index’
- `x` a scalar comparison value
- `what` on of c("ind", "pos", "val")
- `...` further arguments passed to `indexFindInterval`

**Details**

- `indexEQ` index EQual value
- `indexNE` index NotEqual value
- `indexLT` index LowerThan value
- `indexLE` index LowerEqual value
- `indexGT` index GreaterThan value
- `indexGE` index GreaterEqual value
**Value**

A vector of original positions (pos), index positions (ind) or values (val).

**Author(s)**

Jens Oehlschlägel

**See Also**

`index`, `indexFindInterval`, `==.index`

---

**Description**

Find position in index

**Usage**

```r
indexFind(obj, val, findlow = TRUE)
indexFindlike(obj, val, findlow = TRUE)
```

**Arguments**

- `obj` an object of class 'index'
- `val` search value
- `findlow` FALSE to find highest instance of value (default false)

**Details**

- `indexFind` finding exact values in index
- `indexFindlike` finding values in index that begin like search value (character indices only)

**Value**

An integer position of lowest or highest instance of search value

**Author(s)**

Jens Oehlschlägel
indexFindInterval

See Also

index, indexFindInterval, grep

indexFindInterval  Mid level search: interval

Description

Find index positions in interval of search values

Usage

indexFindInterval(obj, low = NULL, high = NULL, low.include = TRUE, high.include = TRUE, low.exact = FALSE, high.exact = FALSE, lowFUN = fun, highFUN = fun, FUN = indexFind)

Arguments

obj  an object of class ‘index’
low  low search value
high  high search value
low.include  FALSE to not include the lower search value (default TRUE)
high.include  FALSE to not include the upper search value (default TRUE)
low.exact  TRUE to require the the low search value is present (default FALSE)
high.exact  TRUE to require the the upper search value is present (default FALSE)
lowFUN  low level search function to identify lower index position (default FUN)
highFUN  low level search function to identify lower index position (default FUN)
FUN  low level search function to identify both index positions (default indexFind)

Details

indexFindInterval  finding a sequence of exact or approximate values

Value

An integer sequence of index positions

Author(s)

Jens Oehlschlägel
indexMatch

See Also

index, indexFind, indexMatch

indexInit

Load / unload rindex library

Description

Function indexInit loads the rindex shared library, indexDone unloads the rindex shared library.

Usage

indexInit()
indexDone()

details

You are responsible to free all memory using indexDelTree before calling indexDone.

Value

See dyn.load and dyn.unload

Author(s)

Jens Oehlschlägel

See Also

dyn.load, dyn.unload, indexDelTree

indexMatch

Mid level search: match set

Description

Find (lowest/highest) index positions for set of search values

Usage

indexMatch(obj, x, findlow = TRUE, what = c("ind", "pos", "val"))
**Arguments**

- **obj**: an object of class ‘index’
- **x**: a vector of search values
- **findlow**: FALSE to take highest instance (default TRUE)
- **what**: on of c("ind", "pos", "val")

**Details**

```
indexMatch  finding positions of vector of search values
```

**Value**

A vector of index positions (ind), original positions (pos) or values (val).

**Note**

`indexMatch` warns if called on a non-unique index. This warning can be suppressed by giving parameter `findlow` explicitly.

**Author(s)**

Jens Oehlschlägel

**See Also**

`index`, `indexFind`, `indexFindInterval`, `match`
Value

Functions `match.index` and `match.rindex` return a vector of original positions (or the nomatch value NA).

Author(s)

Jens Oehlschlägel

See Also

`index`, `indexFind`, `indexMatch`, `match`

Operators.index  

*High level comparison operator*

Description

Compare index against value

Usage

```r
## S3 method for class 'index'
e1 == e2
## S3 method for class 'index'
e1 != e2
## S3 method for class 'index'
e1 < e2
## S3 method for class 'index'
e1 <= e2
## S3 method for class 'index'
e1 > e2
## S3 method for class 'index'
e1 >= e2
```

Arguments

- `e1` an object of class ‘index’
- `e2` a scalar comparison value

Details

- `==.index` index EQual value
- `!=.index` index NotEqual value
- `<.index` index LowerThan value
- `>.index` index GreaterThan value
- `<=.index` index LowerEqual value
- `>=.index` index GreaterEqual value
Value
A vector of logical values

Author(s)
Jens Oehlschlägel

See Also
index, indexFindInterval, indexEq

Description
Functions to extract from an index 1) the original vector, 2) the sorted vector, 3) the original positions (order) and 4) logical NAiness.

Usage

```r
## S3 method for class 'index'
x[i, ...]
## S3 method for class 'index'
sort(x, decreasing = FALSE, na.last = NA, ...)
order.index(..., na.last = TRUE, decreasing = FALSE)
## S3 method for class 'index'
is.na(x)
```

Arguments

- `x`: an object of class ‘index’
- `i`: subset information
- `...`: one object of class ‘index’ for `order.index`, otherwise not to be used
- `decreasing`: TRUE to sort decreasing (default FALSE)
- `na.last`: FALSE to sort NAs first (default TRUE)

Details

- `sort.index`: identical to `sort` of original vector, but much faster
- `order.index`: identical to `order` of original vector, but much faster
- `[.index`: `index[]` returns original vector, subsetting works identical to susetting original vector `[` (via `NextMethod`)
- `<=.index`: currently forbidden
- `is.na.index`: identical to `is.na` of original vector, but much faster
**Value**

Function [.index returns the original vector (or part of it), sort.index returns a sorted vector of values, order.index returns a vector of original integer positions and is.na.index returns a logical vector.

**Note**

There are dummy functions names.index, names<-.index and [<-.index that catch non-supported use of these generics on index objects. Note that for non-unique indices order.index(..., decreasing=TRUE) handles ties not identical to order(..., decreasing=TRUE).

**Author(s)**

Jens Oehlschlägel

**See Also**

index, sort, order, [

---

**strncmp**  
String comparison à la C

**Description**

Functions to compare two vectors of strings like the standard C functions do.

**Usage**

strncmp(a, b)
strncmp(a, b, n)

**Arguments**

- `a` character vector
- `b` character vector
- `n` number of characters to compare

**Value**

1 if `a>b`, -1 if `a<b`, 0 if `a==b`.

**Author(s)**

Jens Oehlschlägel
strlen

See Also

substr, Comparison
Index

=! .index (Operators.index), 16
=! .rindex (Operators.index), 16
*Topic database
  index, 2
  indexBytes, 8
  indexEQ, 11
  indexFind, 12
  indexFindInterval, 13
  indexInit, 14
  indexMatch, 14
  match, 15
  Operators.index, 16
  sort.index, 17
*Topic misc
  index, 2
  indexBytes, 8
  indexDemoClose, 10
  indexEQ, 11
  indexFind, 12
  indexFindInterval, 13
  indexInit, 14
  indexMatch, 14
  match, 15
  Operators.index, 16
  sort.index, 17
  strncmp, 18
< .index, 3, 16
< .index (Operators.index), 16
< .rindex (Operators.index), 16
<= .index, 3, 16
<= .index (Operators.index), 16
<= .rindex (Operators.index), 16
== .index, 3, 12, 16
== .index (Operators.index), 16
== .rindex (Operators.index), 16
>.index, 3, 16
>.index (Operators.index), 16
>.rindex (Operators.index), 16
>= .index, 3, 16
>= .index (Operators.index), 16
>= .rindex (Operators.index), 16
[3, 17, 18
[.index, 3, 17
[.index (sort.index), 17
[.rindex (sort.index), 17
[<= .index, 3, 17
[<= .index (sort.index), 17
[<= .rindex (sort.index), 17

binregtest, 5

c.index (index), 2

c.rindex (index), 2

Comparison, 19

dyn.load, 14
dyn.unload, 14

gc, 2
grep, 13

index, 2, 10, 12–18
indexAddTree, 10
indexAddTree (index), 2
indexAutobatch (index), 2
indexBytes, 2, 8, 9
indexDelTree, 10, 14
indexDelTree (index), 2
indexDemoClose, 10
indexDemoOpen (indexDemoClose), 10
indexDone, 10
indexDone (indexInit), 14
indexEQ, 3, 11, 11, 17
indexFind, 3, 12, 12, 13–16
indexFindInterval, 3, 11–13, 13, 15, 17
indexFindlike, 3, 12
indexFindlike (indexFind), 12
indexGE, 3, 11
indexGE (indexEQ), 11
indexGT, 3, 11
INDEX

indexGT (indexEQ), 11
indexInit, 10, 14
indexLE, 3, 11
indexLE (indexEQ), 11
indexLT, 3, 11
indexLT (indexEQ), 11
indexMatch, 3, 14, 14, 15, 16
indexNE, 3, 11
indexNE (indexEQ), 11
indexNodes, 2, 9
indexNodes (indexBytes), 8
is.na, 3, 17
is.na.index, 3, 17
is.na.index (sort.index), 17
is.na.index (sort.index), 17
length, 2, 9, 10
length.index, 2, 9
length.index (indexBytes), 8
length.rindex (indexBytes), 8
match, 3, 6, 15, 15, 16
match.index, 3
match.rindex, 3
names.index, 2, 9
names.index (indexBytes), 8
names.rindex (indexBytes), 8
names<- .index, 3, 9
names<- .index (indexBytes), 8
names<- .rindex (indexBytes), 8
NextMethod, 3, 17
object.size, 10
Operators.index, 16
order, 3, 4, 6, 17, 18
order.index, 3, 17
order.index (sort.index), 17
order.rindex (sort.index), 17
print.index, 2, 9
print.index (indexBytes), 8
print.rindex (indexBytes), 8
rindex (index), 2
rindexAddTree (index), 2
rindexAutobatch (index), 2
rindexBytes (indexBytes), 8
rindexDeleteTree (index), 2
rindexEQ (indexEQ), 11
rindexFind (indexFind), 12
rindexFindInterval (indexFindInterval), 13
rindexFindLike (indexFind), 12
rindexGE (indexEQ), 11
rindexGT (indexEQ), 11
rindexLE (indexEQ), 11
rindexLT (indexEQ), 11
rindexMatch (indexMatch), 14
rindexNodes (indexEQ), 11
rindexNE (indexEQ), 11
rm, 2
sort, 3, 6, 17, 18
sort.index, 3, 17, 17
sort.rindex (sort.index), 17
str.index, 2, 9
str.index (indexBytes), 8
str.rindex (indexBytes), 8
strcmp (strncmp), 18
strncmp, 18
substr, 19