Package ‘bsearchtools’

February 22, 2017

Type       Package
Title      Binary Search Tools
Version    0.0.61
Date       2017-02-22
Author     Marco Giuliano
Maintainer  Marco Giuliano <mgiuliano.mail@gmail.com>
Description Exposes the binary search functions of the C++ standard library (std::lower_bound, std::upper_bound) plus other convenience functions, allowing faster lookups on sorted vectors.
License    GPL (>= 2)
Imports    Rcpp (>= 0.12.4)
URL        https://github.com/digEmAll/bsearchtools
BugReports https://github.com/digEmAll/bsearchtools/issues
LinkingTo  Rcpp
NeedsCompilation yes
Repository  CRAN
Date/Publication 2017-02-22 23:26:43

R topics documented:

  bsearchtools-package ........................................... 2
  DFI .............................................................. 4
  DFI.coercion .................................................... 5
  DFI.getIndex .................................................... 6
  DFI.indexes ..................................................... 7
  DFI.subset ...................................................... 7
  DFI.unWrap ...................................................... 8
  indexesEqualTo ................................................ 9
  indexesInRange ............................................... 10
  indexesMerge .................................................. 11
**Description**

Exposes the binary search based functions of the C++ standard library (std::lower_bound, std::upper_bound) plus other convenience functions, allowing faster lookups on sorted vectors. It also includes a lightweight data.frame/matrix wrapper (DFI), which automatically creates indexes on the columns for faster lookups.

**Details**

- **Package:** bsearchtools
- **Type:** Package
- **Version:** 0.0.61
- **Date:** 2017-02-22
- **License:** GPL (>= 2)

This package allows to perform the most common binary search operations on sorted vectors (integer, numeric, bool and character vectors are supported). It exposes lower-bound/upper-bound functions working exactly like their the C++ standard library counterparts, and some convenience functions allowing efficient values and ranges lookups.

Note that these functions are especially designed to be used for non-vectorized operations (e.g. inside loops); for vectorized operations, the great data.table package already fulfills basically every R programmer needs.

**Author(s)**

Marco Giuliano

Maintainer: Marco Giuliano <mgiuliano.mail@gmail.com>

**References**

Project repository: https://github.com/digEmAll/bsearchtools/


**See Also**

sort, order, data.table
Examples

```
require(bsearchtools)

### get indexes of values in range
### search values in range [2,4]

# N.B. v must be sorted!
v1 <- sort(c(3,5,7,10,4,8,13,3,2))

indexesInRangeNumeric(v1, 2, 4)
# is identical to:
which(v1 >= 2 & v1 <= 4)

### What if vector is not sorted?
### (and we're going to perform a lot of lookups on it)

v2 <- c(3,5,7,10,4,8,13,3,2)

# we can create two intermediate vectors
ordIdxs <- order(v2)
sortedV2 <- v2[ordIdxs]

# then use them as follows:
ordIdxs[indexesInRangeNumeric(sortedV2, 2, 4)]

# this returns the same indexes:
# N.B.: 'which' returns ascending indexes while the previous line does not:
# sort the result if you want them ascending
which(v2 >= 2 & v2 <= 4)

### N.B. the previous code is basically what is performed by DFI objects under the hood
### check DFI function documentation for further information
DF <- data.frame(v2=v2)
DFI_obj <- DFI(DF)
indexes <- DFI.subset(DFI_obj, RG('v2', 2, 4), return.indexes=TRUE)

## Not run:
### big example to measure the performance difference
set.seed(123)  # for reproducibility
sortedValues <- sort(sample(1:1e4, 1e5, replace=TRUE))

# measure time difference doing same operation 500 times
tm1 <- system.time(for(i in 1:500) res2 <- which(sortedValues >= 7000 & sortedValues <= 7500))

print(paste("which" took,"tm1["elapsed"]"))
```

```
DFI

Create a data.frame (or matrix) with indexes

Description

Turn a data.frame (or matrix) object into a DFI object allowing faster lookups on indexed columns (indexed column to be intended as DB indexes).

Usage

```r
DFI(DF, indexes.col.names=colnames(DF))
as.DFI(DF, indexes.col.names=colnames(DF)) # exactly the same as DFI()
is.DFI(x)
```  

## S3 method for class 'DFI'

```r
print(x, ...)
```

Arguments

- **DF**: A data.frame or matrix object (must have column names defined).
- **indexes.col.names**: The column names for which we want to create the indexes. Only integer, numeric, logical and character are supported, so be careful since data.frame by default turns strings into factors (see data.frame stringsAsFactors argument)
- **x**: A DFI object.
- **...**: optional arguments passed to inner print methods of data.frame and matrix.

Details

Basically, `DFI()` function creates a wrapper of `DF`. This wrapper contains the original data.frame or matrix plus the necessary indexes data and the class of the wrapped object. These extra data will be used to perform faster lookups (in `DFI.subset` function) and can be extracted using the appropriate functions `DFI.unWrap`, `DFI.indexes`, `DFI.getIndex`.

Value

An object with class "DFI"

Note

Since version 0.0.47 DFI objects do not inherit from data.frame or matrix anymore, hence they cannot be modified/subsetted using data.frame/matrix standard operators. This has been changed since the column indexes are not recreated automatically and once the object is modified, DFI.subset could give wrong results without any warning. To use the standard replacement and subset operators, extract the original object first using `DFI.unWrap(DFIobj)`. 

## End(Not run)
DFI.coercion

See Also
DFI.subset DFI.unWrap DFI.indexes DFI.getIndex

Examples

```r
DF <- data.frame(Foo=c(3,5,7,1,5,8,7,10),
    Bar=c("A","B","B","C","B","B","C","A"),
    Baz=c(TRUE,FALSE),
    stringsAsFactors=FALSE)
DFIobj <- DFI(DF, c("Foo","Bar")) # create a DFI from DF with indexes on "Foo" and "Bar" columns
```

DFI.coercion  

Coerce a DFI object

Description

Coerce a DFI object to data.frame or matrix

Usage

```r
## S3 method for class 'DFI'
as.data.frame(x, ...)
## S3 method for class 'DFI'
as.matrix(x, ...)
```

Arguments

- `x`  
a DFI object
- `...`  
optional arguments passed to inner as.data.frame and as.matrix methods.

Value

A data.frame or matrix object

See Also

DFI

Examples

```r
## create a simple DFIobj
DF <- data.frame(Foo=c(3,5,7,1,5,8,7,10),
    Bar=c("A","B","B","C","B","B","C","A"),
    Baz=c(TRUE,FALSE),
    stringsAsFactors=FALSE)
DFIobj <- DFI(DF, c("Foo","Bar")) # create a DFI from DF with indexes on "Foo" and "Bar" columns
```
DFI.getIndex

Extract the index information of a DFI object

Description

Return the index data (i.e. ordered indexes, and sorted values) of an indexed column of a DFI object

Usage

DFI.getIndex(DFIobj, name)

Arguments

DFIobj a DFI object
name the name of the indexed column in the DFI object

Value

A list with two values:

- idxs the indexes used to sort the column values (as returned by `order(colValues, na.last=NA)`
- sorted the sorted values of the column (as returned by `colValues[idxs]`)

See Also

DFI

Examples

```r
### create a simple DFIobj
DF <- data.frame(Foo=c(3,5,7,1,5,8,7,10),
                 Bar=c("A","B","B","C","B","B","C","A"),
                 Baz=c(TRUE,FALSE),
                 stringsAsFactors=FALSE)
DFIobj <- DFI(DF, c("Foo","Bar")) # create a DFI from DF with indexes on "Foo" and "Bar" columns

### get the index data of 'Bar' column
DFI.getIndex(DFIobj,"Bar")
```
DFI.indexes

Get the indexes names of a DFI object

Description

Method to get the indexes names of a DFI object

Usage

DFI.indexes(DFIobj)

Arguments

DFIobj A DFI object

Value

A character vector containing the name of the indexed columns of the DFI object

Examples

### create a simple DFIobj
DF <- data.frame(Foo=c(3,5,7,1,5,8,7,10),
                 Bar=c("A","B","B","C","B","B","C","A"),
                 Baz=c(TRUE,FALSE),
                 stringsAsFactors=FALSE)
DFIobj <- DFI(DF, c("Foo","Bar")) # create a DFI from DF with indexes on "Foo" and "Bar" columns

### get the indexes names (returns c("Foo","Bar")
DFI.indexes(DFIobj)

DFI.subset

Subset a DFI object

Description

Function to subset a DFI object efficiently (using binary search) by creating complex filters on indexed columns. For details about column indexes, refer to DFI, for information about NA handling, refer to rowfilters.DFI.

Usage

DFI.subset(DFIobj, filter=NULL, return.indexes=FALSE,
           sort.indexes=TRUE, colfilter=NULL, drop=NULL)
Arguments

- **DFIobj**: a DFI object.
- **filter**: a filter object created by functions `EQ`, `RG`, `IN`, `AND`, `OR`, `NOT`.
- **return.indexes**: if TRUE, the row indexes satisfying the filter are returned instead of the DFI subset.
- **sort.indexes**: if FALSE the order of the rows or row.index returned will not be necessarily equal to the original order in the DFI object. If TRUE, subsetting will keep the original row/row.indexes order. FALSE usually gives a better performance.
- **colFilter**: if return.indexes==TRUE is ignored; otherwise, if not NULL, it will be passed as second argument of data.frame/matrix subset operator i.e. `[,colFilter]`.
- **drop**: if different from NULL is passed as drop argument of data.frame and matrix subset (ignored if return.indexes=TRUE).

Value

A subset of the data.frame or matrix wrapped by the DFI object, unless return.indexes==TRUE in which case an integer vector with the row indexes will be returned.

See Also

- `DFI`, `EQ`, `IN`, `RG`, `NOT`, `AND`, `OR`

Examples

```r
### create a simple DFIobj
DF <- data.frame(Foo=c(3,5,7,1,5,8,7,10),
                 Bar=c("A","B","B","C","B","B","C","A"),
                 Baz=rep(c(TRUE,FALSE),6),
                 stringsAsFactors=FALSE)
DFIobj <- DFI(DF, c("Foo","Bar"))  # create a DFI from DF with indexes on "Foo" and "Bar" columns
DFI.unwrap(DFIobj, filter=OR(EQ('Foo',5),EQ('Bar','B')))
```

---

**DFI.unwrap**

Unwrap a DFI object returning the original wrapped object

Description

Extract the original wrapped object (data.frame or matrix) inside a DFI object

Usage

`DFI.unwrap(DFIobj)`
indexesEqualTo

Arguments

DFIobj a DFI object

Value

A data.frame or matrix according to the class of the original object

See Also

DFI

Examples

### create a simple DFIobj
DF <- data.frame(Foo=c(3,5,7,1,5,8,7,10),
    Bar=c("A","B","C","B","B","C","A"),
    Baz=c(TRUE,FALSE),
    stringsAsFactors=FALSE)
DFIobj <- DFI(DF, c("Foo","Bar")) # create a DFI from DF with indexes on "Foo" and "Bar" columns

### get the inner data.frame
DFI.unWrap(DFIobj)

indexesEqualTo  Find indexes of a value using binary search

Description

Given a sorted vector, it returns the indexes of the vector elements equal to valueToSearch.

The functions suffixed with the vector type (indexInRangeNumeric,indexInRangeLogical etc.) can be used ONLY with the specified type, otherwise the vector is coerced, and they are (hopefully negligibly) faster then the generic indexesEqualTo function.

Usage

indexesEqualTo(sortedValues,valueToSearch,indexesRemap=NULL)
indexesEqualToNumeric(sortedValues,valueToSearch,indexesRemap=NULL)
indexesEqualToInteger(sortedValues,valueToSearch,indexesRemap=NULL)
indexesEqualToLogical(sortedValues,valueToSearch,indexesRemap=NULL)
indexesEqualToCharacter(sortedValues,valueToSearch,indexesRemap=NULL)

Arguments

sortedValues A sorted atomic vector of type numeric, integer, logical or character
valueToSearch The value to search in the vector
indexesRemap An integer vector to be used to remap the indexes returned by lookup on sorted-Values, or NULL (the default). Mostly used internally by DFI.
indexesInRange

Value

The indexes of the vector elements equal to valueToSearch.

Examples

indexesEqualTo(c(1,4,5,5,7,9),5) # returns c(3,4)
indexesEqualTo(c(1,4,5,5,7,9),10) # returns empty vector

indexesInRange

Find indexes in a range using binary search

Description

Given a sorted vector, it returns the indexes of the vector elements included in range \([lbInclusive,ubInclusive]\).

The functions suffixed with the vector type (indexInRangeNumeric,indexInRangeLogical etc.) can be used ONLY with the specified type, otherwise the vector is coerced, and they are (hopefully negligibly) faster then the generic indexInRange function.

Usage

indexesInRange(sortedValues,lbInclusive,ubInclusive,indexesRemap=NULL)
indexesInRangeNumeric(sortedValues,lbInclusive,ubInclusive,indexesRemap=NULL)
indexesInRangeInteger(sortedValues,lbInclusive,ubInclusive,indexesRemap=NULL)
indexesInRangeLogical(sortedValues,lbInclusive,ubInclusive,indexesRemap=NULL)
indexesInRangeCharacter(sortedValues,lbInclusive,ubInclusive,indexesRemap=NULL)

Arguments

- **sortedValues**: A sorted atomic vector of type numeric, integer, logical or character
- **lbInclusive**: The inclusive lower bound of the range
- **ubInclusive**: The inclusive upper bound of the range
- **indexesRemap**: An integer vector to be used to remap the indexes returned by lookup on sortedValues, or NULL (the default). Mostly used internally by DFI.

Value

The indexes of the vector elements included in range \([lbInclusive,ubInclusive]\).

Examples

indexesInRange(c(1,4,5,5,7,9),5) # returns c(3,4,5)
indexesInRange(c(1,4,5,5,7,9),10,11) # returns empty vector
indexesMerge

Intersection / union of list of indexes

Description

Functions to perform intersection or union of a list of integer vectors. This functions are used by DFL.subset for AND/OR filters.

Usage

intersectIndexesList(lst, sorted=TRUE)
unionIndexesList(lst, sorted=TRUE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lst</td>
<td>list of integer vectors on which intersection or union must be performed</td>
</tr>
<tr>
<td>sorted</td>
<td>logical value used to specify if the returned indexes should be sorted ascending (default TRUE)</td>
</tr>
</tbody>
</table>

Details

The returned vector is sorted ascending. intersectIndexesList is implemented in C++ and corresponds to `sort(unique(reduce(f=intertsect, x=lst)))` (without the sort function if sorted=FALSE). unionIndexesList is partially implemented in C++ and corresponds to `sort(unique(reduce(f=union, x=lst)))` (without the sort function if sorted=FALSE).

Value

A vector of integers.

Examples

intersectIndexesList(list(1:7,4:8,3:5))
unionIndexesList(list(1:7,4:8,3:5))

lb

Binary search based lower bound operation
Description

Returns the index pointing to the first element in the vector that is not less than (i.e. greater or equal to) valueToSearch. The behavior is the same as C++ std::lower_bound function, hence, if the vector is empty or valueToSearch is lower than the first element of the vector, it returns the first index (i.e. 1).

The functions suffixed with the vector type (lbNumeric, lbLogical etc.) can be used ONLY with the specified type, otherwise the vector is coerced, and they are (hopefully negligibly) faster then the generic lb function.

For information about NAs handling see details section.

Usage

lb(sortedValues, valueToSearch)
lbInteger(sortedValues, valueToSearch)
lbNumeric(sortedValues, valueToSearch)
lbLogical(sortedValues, valueToSearch)
lbCharacter(sortedValues, valueToSearch)

Arguments

sortedValues    A sorted atomic vector of type numeric, integer, logical or character.
valueToSearch   The value to search. If equal to NA, 1 is returned.

Details

lb* functions expect sortedValues to be a vector sorted ascending (duplicated values are allowed). Since the binary search functions rely on values comparison (using < operator) and NA cannot be compared by definition, if sortedValues vector contains NA, the result is unpredictable and NO warning is given. Hence remove them before calling these functions.

Value

The index pointing to the first element in the vector that is not less than (i.e. greater or equal to) valueToSearch.

References


Examples

lb(c(1,4,5,5,7,9),5) # returns 3
lb(c(1,4,5,5,7,9),-1) # returns 1
lb(numeric(),-1) # returns 1
Description

Functions for row filters creation in DF1 subset.
For information about NAs handling see details section.

Usage

RG(col, from, to)
IN(col, values)
EQ(col, val)
EQNA(col)
NOT(filter)
OR(...)
AND(...)

## S3 method for class 'DF1.FEXPR'
print(x,...)

## S3 method for class 'DF1.FEXPR'
toString(x,...)

## S3 method for class 'DF1.FEXPR'
as.character(x,...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>column name to be used in the filter condition (must be an indexed column).</td>
</tr>
<tr>
<td>from</td>
<td>inclusive lower-bound of the range (RG) filter condition.</td>
</tr>
<tr>
<td>to</td>
<td>inclusive upper-bound of the range (RG) filter condition.</td>
</tr>
<tr>
<td>values</td>
<td>valid values for the filter condition (used by IN).</td>
</tr>
<tr>
<td>val</td>
<td>valid value for the filter condition (used by EQ).</td>
</tr>
<tr>
<td>filter</td>
<td>filter condition to be negated (created with RG,IN,EQ,NOT,OR,AND).</td>
</tr>
<tr>
<td>...</td>
<td>one or more filters conditions to be put in AND or OR (created with RG,IN,EQ,NOT,OR,AND).</td>
</tr>
<tr>
<td>x</td>
<td>an object of class DF1.FEXPR</td>
</tr>
</tbody>
</table>

Details

Any filter function applied to an indexed column will filter out the NAs present in that column by default (except for EQNA). So, for example, the following filter: EQ("A",3) is actually equal to: !is.na(A) & A == 3. The functions print(filterExpr), toString(filterExpr) and as.character(filterExpr) return the string representation of the filter that you would use in a normal data.frame subset.
RG function accepts NA in from to arguments, and this "turns off" the part of the filter set to NA. So, for instance RG("A",NA,to) will return all values A <= to (but still filtering out the NA values).

EQ function accepts NA in val argument, and this simply "turns off" the filter on the column returning all the elements in the column (but still filtering out the NA values).

IN(colName,values) function is converted to OR(EQ(colName,values[1]),EQ(colName,values[2]),...) hence, if values contains NA, the filter will return all the elements in the column (but still filtering out the NA values).

EQNA(colName) function can be used to select the NAs in the column, which are excluded by the other operators.

Value

EQ,RG,IN,EQNA,NOT,AND,OR functions return an object inheriting from class 'DFI.FEXPR' to be used as row filter in DFI.subset function. print,string,as.character functions return the string representation of an object of class 'DFI.FEXPR'.

See Also

DFI.subset

Examples

# create the following filter: 18 <= Age <= 55 & Married == TRUE
filter <- AND(RG('Age',18,55),EQ('Married',TRUE))

# create the following filter: Age == 25 | Married == TRUE | Name == 'John'
filter <- OR(EQ('Age',25),EQ('Married',TRUE),EQ('Name','John'))

Description

Returns the index pointing to the first element in the vector that is greater than valueToSearch. The behavior is the same as C++ std::upper_bound function, hence, if the vector is empty it or if valueToSearch is greater than the last element of the vector, it returns length(sortedValues) + 1.

The functions suffixed with the vector type (ubNumeric,ubLogical etc.) can be used ONLY with the specified type, otherwise the vector is coerced, and they are (hopefully negligibly) faster than the generic ub function.

For information about NAs handling see details section.
Usage

\[
\begin{align*}
&\texttt{ub}(\text{sortedValues, valueToSearch}) \\
&\texttt{ubInteger}(\text{sortedValues, valueToSearch}) \\
&\texttt{ubNumeric}(\text{sortedValues, valueToSearch}) \\
&\texttt{ubLogical}(\text{sortedValues, valueToSearch}) \\
&\texttt{ubCharacter}(\text{sortedValues, valueToSearch})
\end{align*}
\]

Arguments

\[
\begin{align*}
\text{sortedValues} & \quad \text{A sorted atomic vector of type numeric, integer, logical or character} \\
\text{valueToSearch} & \quad \text{The value to search. If equal to NA, length(sortedValues)+1 is returned.}
\end{align*}
\]

Details

\texttt{ub} functions expect \texttt{sortedValues} to be a vector sorted ascending (duplicated values are allowed). Since the binary search functions rely on values comparison (using \texttt{<} operator) and NA cannot be compared by definition, if \texttt{sortedValues} vector contains NA, the result is unpredictable and NO warning is given. Hence remove them before calling these functions.

Value

The index pointing to the first element in the vector that is not less than (i.e. greater or equal to) \texttt{valueToSearch}.

References


Examples

\[
\begin{align*}
&\texttt{ub(c(1,4,5,5,7,9),5)} \ # \text{returns 5} \\
&\texttt{ub(c(1,4,5,5,7,9),10)} \ # \text{returns 7} \\
&\texttt{ub(numeric(),10)} \ # \text{returns 1}
\end{align*}
\]
Index

*Topic iteration
  bsearchtools-package, 2
*Topic manip
  bsearchtools-package, 2
*Topic package
  bsearchtools-package, 2
*Topic programming
  bsearchtools-package, 2

AND, 8
AND (rowfilters.DFI), 13
as.character.DFI.FEXPR
  (rowfilters.DFI), 13
as.data.frame.DFI (DFI.coercion), 5
as.DFI (DFI), 4
as.matrix.DFI (DFI.coercion), 5

bsearchtools (bsearchtools-package), 2
bsearchtools-package, 2
data.table, 2
DFI, 4, 5–9
DFI.coercion, 5
DFI.getIndex, 4, 5, 6
DFI.indexes, 4, 5, 7
DFI.subset, 4, 5, 7, 14
DFI.unWrap, 4, 5, 8

EQ, 8
EQ (rowfilters.DFI), 13
EQA (rowfilters.DFI), 13

IN, 8
IN (rowfilters.DFI), 13
indexesEqualTo, 9
indexesEqualToCharacter
  (indexesEqualTo), 9
indexesEqualToInteger (indexesEqualTo),
  9
indexesEqualToLogical (indexesEqualTo),
  9

indexesEqualToNumeric (indexesEqualTo), 9
indexesInRange, 10
indexesInRangeCharacter
  (indexesInRange), 10
indexesInRangeInteger (indexesInRange), 10
indexesInRangeLogical (indexesInRange), 10
indexesInRangeNumeric (indexesInRange), 10
indexesMerge, 11
intersectIndexesList (indexesMerge), 11
is.DFI (DFI), 4
lb, 11
lbCharacter (lb), 11
lbInteger (lb), 11
lbLogical (lb), 11
lbNumeric (lb), 11

NOT, 8
NOT (rowfilters.DFI), 13

OR, 8
OR (rowfilters.DFI), 13
order, 2

print.DFI (DFI), 4
print.DFI.FEXPR (rowfilters.DFI), 13

RG, 8
RG (rowfilters.DFI), 13
rowfilters.DFI, 7, 13

sort, 2

toString.DFI.FEXPR (rowfilters.DFI), 13

ub, 14
ubCharacter (ub), 14
INDEX

ubInteger (ub), 14
ubLogical (ub), 14
ubNumeric (ub), 14
unionIndexesList (indexesMerge), 11