Package 'disordR'

December 10, 2022

Type Package

Title Non-Ordered Vectors

Version 0.0-9-4

Depends methods,Matrix (>= 1.3-3)

Imports digest

Suggests mvp,knitr,rmarkdown,testthat

VignetteBuilder knitr

Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>

Description Functionality for manipulating values of associative maps. Ordinary R vectors are unsuitable for working with values of associative maps because elements of an R vector may be accessed by reference to their location in the vector, but associative maps are stored in arbitrary order. However, when associating keys with values one needs both parts to be in 1-1 correspondence, so one cannot dispense with the order entirely. The 'disordR' package includes a single S4 class, disord. This class allows one to perform only those operations appropriate for manipulating values of associative maps and prevents any other operation (such as accessing an element at a particular location). A useful heuristic is that one is only allowed to access or modify a disord object using a python list comprehension. The idea is to prevent ill-defined operations on values (or keys) of associative maps, whose order is undefined or at best implementation-specific, while allowing and facilitating sensible operations. To cite the package in publications please use Hankin (2022) <doi:10.48550/ARXIV.2210.03856>.

License GPL (>= 2)

URL https://github.com/RobinHankin/disordR

BugReports https://github.com/RobinHankin/disordR/issues

NeedsCompilation no

Author Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Repository CRAN
Date/Publication 2022-12-10 14:10:02 UTC

R topics documented:

`Arith` *Arithmetic and logical operations*

Description

Arithmetic operations including low-level helper functions

Usage

```
disord_inverse(a)
disord_mod_disord(a,b)
disord_mod_numeric(a,b)
disord_negative(a)
disord_plus_disord(a,b)
disord_plus_numeric(a,b)
disord_power_disord(a,b)
disord_power_numeric(a,b)
numeric_power_disord(a,b)
disord_prod_disord(a,b)
disord_prod_numeric(a,b)
disord_logical_negate(x)
disord_arith_unary(e1,e2)
disord_arith_disord(e1,e2)
disord_arith_numeric(e1,e2)
numeric_arith_disord(e1,e2)
```

Arguments

`a,b,x`	at least one is a disord object
`e1,e2`	Formal arguments for S4 dispatch

Details

Basic low-level arithmetic operations, intended to be called from S4 dispatch.

These functions return a `disord` object or a regular vector as appropriate. Consistency is required. The hash is set to be that of the disord object if appropriate.

Value

Return a disord object or logical

Author(s)

Robin K. S. Hankin

Examples

```
a <- rdis()
a
a + 2*a
a > 5
a[a > 5] <- a[a > 5] + 100
a
```

---

c *Concatenation*

Description

Concatenation simply does not make sense for `disord` objects.

Value

Returns an error.

Note

I could not figure out how to stop idiom like "`c(1,rdis())`" from returning a result. Just don't use it, OK?

Author(s)

Robin K. S. Hankin

`consistent` *Check for consistency*

Description

The **disordR** package is designed to make permitted operations transparent and to prevent forbidden operations from being executed.

Function `consistent()` checks for matching hash codes of its arguments and returns a Boolean. It is called by function `check_matching_hash()` which either returns `TRUE` or reports an informative error message if not.

Usage

```
consistent(x,y)
x %~% y
check_matching_hash(e1,e2,use=NULL)
```

Arguments

`x,y,e1,e2`	Objects of class `disord`
`use`	optional object designed to give a more intelligible error message; typically `match.call()`

Details

Function `consistent()` checks that its arguments have the same hash code, and thus their elements can be paired up (e.g. added). Idiom `a %~% b` is equivalent to `consistent(a,b)`.

The package generally checks for consistency with function `check_matching_hash()` which provides some helpful diagnostics if `consistent()` finds a hash mismatch.

Value

Boolean or an error as appropriate

Author(s)

Robin K. S. Hankin

See Also

`disord`

Examples

```
# rdis() + rdis() # this would make check_matching_hash() report an error, if executed
```

disord *Functionality for* disord *objects*

Description

Allows arithmetic operators to be used for disord objects; the canonical application is coefficients of multivariate polynomials (as in the **mvp** package). The issue is that the storage order of disord objects is implementation-specific but the order (whatever it is) must be consistent between the list of keys and values in an associative array.

Usage

```
is.disord(x)
hash(x)
hashcal(x)
disord(v,h,drop=TRUE)
elements(x)
```

Arguments

x	Object of class disord
v	Vector of coefficients
h	Hash code
drop	Boolean, with default FALSE meaning to return a disord object and TRUE meaning to call drop() before returning

Details

The package provides a single S4 class, disord. A detailed vignette is provided that motivates the package.

Function hash() is the extractor function:

`` `hash` <- function(x){x@hash} ``

Compare hashcal() which is used to actually calculate the hash code for an object. Currently

`` `hashcal` <- function(x){digest::sha1(x)} ``

Function disord() takes a vector and returns a disord object; function elements() takes a disord and returns a vector. These are the central function of the package.

Checking for matching hash codes is done by function consistent(). This checks that its arguments have the same hash code, and thus their elements can be paired up (e.g. added). Idiom a %~% b is equivalent to consistent(a,b).

Value

Boolean, hash code, or object of class disord as appropriate.

Author(s)

Robin K. S. Hankin

Examples

```
(a <- rdis())
(b <- rdis())

a + 2*a + 2^a  # fine
# a + b # this would give an error if executed

a[a<0.5] <- 0       # round down; replacement works as expected

elements(a)
```

---

disord-class *Class* "disord"

Description

The disord class provides basic arithmetic and extract/replace methods for disord objects.

Objects from the Class

Objects can be created by calls of the form new("disord", ...), although functions disord() and (eventually) as.disord() are more user-friendly.

Slots

.Data: Object of class vector that specifies the elements

hash: Object of class character that specifies the hash code

Author(s)

Robin K. S. Hankin

Examples

```
showClass("disord")
```

drop *Drop redundant information*

Description

Coerce disord objects to vector when this makes sense

Usage

```
drop(x)
allsame(x)
```

Arguments

x	disord object

Details

If one has a disord object all of whose elements are identical, one usually wants to drop the disord attribute and coerce to a vector. This can be done without breaking disordR discipline. Function disord() takes a drop argument, defaulting to TRUE, which drops the disord class from its return value if all the elements are the same.

Similarly, function drop() takes a disord object and if all elements are identical it returns the elements in the form of a vector. Some extraction methods take a drop argument, which does the same thing if TRUE. This is only useful for disord objects created with disord(...,drop=FALSE)

The drop functionality is conceptually similar to the drop argument of base R's array extraction, as in

```
a <- matrix(1:30,5,6)
a[1,,drop=TRUE]
a[1,,drop=FALSE]
```

Function allsame() takes a vector and returns TRUE if all elements are identical.

Value

Function drop() returns either a vector or object of class disord as appropriate; allsame() returns a Boolean.

Author(s)

Robin K. S. Hankin

Examples

```
disord(c(3,3,3,3,3))             # default is drop=TRUE
disord(c(3,3,3,3,3),drop=FALSE)  # retains disord class

drop(disord(c(3,3,3,3),drop=FALSE))

## In extraction, argument drop discards disorderliness when possible:
a <- rdis()
a
a[] <- 6 # a becomes a vector
a
```

`extract` *Extraction and replacement methods for class* `"disord"`

Description

The `disord` class provides basic arithmetic and extract/replace methods for disord objects.

Class *index* is taken from the excellent **Matrix** package and is a `setClassUnion()` of classes `numeric`, `logical`, and `character`.

Methods

[`signature(x = "disord", i = "ANY", j = "ANY")`: ...

[`signature(x = "disord", i = "index", j = "index")`: ...

[`signature(x = "disord", i = "index", j = "missing")`: ...

[`signature(x = "disord", i = "missing", j = "index")`: ...

[`signature(x = "disord", i = "missing", j = "missing")`: ...

[`signature(x = "disord", i = "matrix", j = "missing")`: ...

[<- `signature(x = "disord", i = "index", j = "index")`: ...

[<- `signature(x = "disord", i = "index", j = "missing")`: ...

[<- `signature(x = "disord", i = "missing", j = "index")`: ...

[<- `signature(x = "disord", i = "matrix", j = "missing")`: ...

[<- `signature(x = "disord", i = "missing", j = "missing")`: ...

Arith `signature(e1 = "ANY", e2 = "disord")`: ...

Arith `signature(e1 = "disord", e2 = "ANY")`: ...

Arith `signature(e1 = "disord", e2 = "disord")`: ...

Arith `signature(e1 = "disord", e2 = "missing")`: ...

The extraction method takes a `drop` argument which if `TRUE`, returns the `drop()` of its value. Extraction, as in `x[i]`, is rarely useful. It is only defined if one extracts either all, or none, of the elements: anything else is undefined. Note that the hash code is unchanged if all elements are extracted (because the order might have changed) but unchanged if none are (because there is only one way to extract no elements).

Author(s)

Robin K. S. Hankin

See Also

drop

Examples

```
a <- disord(sample(9))
a
a[a>5]  # "give me all elements of a that exceed 5"

a[a<5] <- a[a<5] + 100  # "replace all elements of a that exceed 5 with their value plus 100"
a

## Following expressions would return an error if executed:
if(FALSE){
  a[1]
  a[1] <- 44
  a[1:2] <- a[3:4]
}

b <- disord(sample(9))
## Following expressions would also return an error if executed:
if(FALSE){
  a+b  # (not really an example of extraction)
  a[b>5]
  a[b>5] <- 100
  a[b>5] <- a[b>5] + 44
}
```

---

misc *Miscellaneous functions*

Description

This page documents various functions that work for disords, and I will add to these from time to time as I add new functions that make sense for disord objects. Functions like `sin()` and `abs()` work as expected: they take and return `disord` objects with the same hash as `x` (which means that idiom like `x + sin(x)` is accepted). However, there are a few functions that are a little more involved:

- `rev()` reverses its argument and returns a `disord` object with a reversed hash, which ensures that `rev(rev(x))==x` (and the two are consistent).
- `sort()` returns a vector of sorted elements (not a `disord`)
- length() returns the length of the data component of the object.

- `sapply(X,f)` returns a disord object which is the result of applying `f()` to each element of `X`.
- `match(x,table)` should behave as expected but note that if `table` is a `disord`, the result is not defined (because it is not known where the elements of `x` occur in `table`). Nevertheless `x %in% table` is defined and returns a `disord` object.

Some functionality is not yet implemented. Factors, lists, and named vectors do not behave entirely consistently in the package, and `paste()` gives inconsistent results when called with disords.

Arguments

`x`	Object of class `disord`

Value

Returns a disord

Author(s)

Robin K. S. Hankin

Examples

```
a <- disord(c(a=1,b=2,c=7))
a
names(a)
length(a)
sqrt(a)


# powers() and vars() in the mvp package return lists; see the vignette
# for more discussion.

l <- disord(list(3,6:9,1:10))
sapply(l,length)
```

---

rdis *Random disord objects*

Description

Returns a random disord object

Usage

```
rdis(n=9)
```

Arguments

`n`	Number of elements

Details

A simple `disord` object, intended as a quick "get you going" example

Value

A `disord` object.

Author(s)

Robin K. S. Hankin

Examples

```
rdis()
```

---

summary.disordR *Summaries of disord objects*

Description

A summary method for disord objects, and a print method for summaries.

Usage

```
## S3 method for class 'disord'
summary(object, ...)
## S3 method for class 'summary.disord'
print(x, ...)
```

Arguments

`object,x`	Object of class `disord`
`...`	Further arguments, currently ignored

Details

A `summary.disord` object is summary of a `disord` object `x`: a list with first element being the `hash(x)` and the second being `summary(elements(x))`. The print method is just a wrapper for this.

Author(s)

Robin K. S. Hankin

Examples

```
summary(rdis(1000))
```

Index