Package 'rle'

May 28, 2025

Version 0.10.0

Date 2025-05-28

Title Common Functions for Run-Length Encoded Vectors

Description Common 'base' and 'stats' methods for 'rle' objects, aiming to make it possible to treat them transparently as vectors.

Depends R (>= 3.5)

Imports methods

Copyright file inst/COPYRIGHT

BugReports https://github.com/statnet/rle/issues

License GPL-3

RoxygenNote 7.3.2.9000

Encoding UTF-8

Suggests covr

NeedsCompilation yes

Author Pavel N. Krivitsky [aut, cre] (ORCID: <https://orcid.org/0000-0002-9101-3362>)

Maintainer Pavel N. Krivitsky <pavel@statnet.org>

Repository CRAN

Date/Publication 2025-05-28 14:10:02 UTC

Contents

`rle-package` *rle: Common Functions for Run-Length Encoded Vectors*

Description

Common `base` and `stats` methods for `rle` objects, aiming to make it possible to treat them transparently as vectors.

History

This package grew out of the needs of the `ergm` package for a run-length encoded representation of extremely long vectors with a small number of contiguous runs, and these functions were originally implemented in the `statnet.common` package.

It has been split out into its own package to enable others to use this functionality without installing any unnecessary dependencies and to facilitate contributions under a simplified license.

What works and what doesn't

The long-run aim of this package is to make it possible to treat `rle` objects transparently as unnamed vectors. As of this writing, the biggest unimplemented feature are:

- It is possible to use the indexing (`[` and `[[`) operators to extract by positive numeric indices but not by logical or negative numeric indices, and the implementation is far from optimal. It is not possible to replace individual elements of the vector represented by an `rle` object. See Extract.rle for more details.
- Method `rep.rle` currently has limited functionality.

Author(s)

Maintainer: Pavel N. Krivitsky <pavel@statnet.org> (ORCID)

See Also

Useful links:

- Report bugs at https://github.com/statnet/rle/issues

`as.rle` *Coerce to* `rle` *if not already an* `rle` *object*

Description

Coerce to `rle` if not already an `rle` object

Usage

```
as.rle(x)

## S3 method for class 'rle'
as.rle(x)

## Default S3 method:
as.rle(x)
```

Arguments

`x`	the object to be coerced.

`compress` *A generic function for compressing a data structure.*

Description

A generic function for compressing a data structure.

Usage

```
compress(x, ...)
```

Arguments

`x`	the object to be compressed.
`...`	additional arguments to methods.

compress.rle *Compress the* rle *object by merging adjacent runs*

Description

Compress the rle object by merging adjacent runs

Usage

```
## S3 method for class 'rle'
compress(x, ...)
```

Arguments

x	an rle object.
...	additional objects; if given, all arguments are concatenated.

Note

Since rle stores run lengths as integers, compress.rle will not merge runs that add up to lengths greater than what can be represented by a 32-bit signed integer (2147483647).

Examples

```
x <- rle(as.logical(rbinom(10,1,.7)))
y <- rle(as.logical(rbinom(10,1,.3)))

stopifnot(identical(rle(inverse.rle(x)&inverse.rle(y)),compress(x&y)))

big <- structure(list(lengths=as.integer(rep(.Machine$integer.max/4,6)),
                      values=rep(TRUE,6)), class="rle")

stopifnot(all(aggregate(as.numeric(lengths)~values,
                        data=as.data.frame(unclass(big)),FUN=sum)
              ==
              aggregate(as.numeric(lengths)~values,
                        data=as.data.frame(unclass(compress(big))),
                        FUN=sum)))
```

Extract.rle *Indexing Methods for* rle *Objects*

Description

These methods provide indexing functionality for rle objects on the scale of the original scale (the elements of the vector that was compressed) where possible.

Usage

```
## S3 method for class 'rle'
x[i, ..., unclass = getOption("rle.unclass_index") %||% FALSE]

## S3 replacement method for class 'rle'
x[i, ..., unclass = getOption("rle.unclass_index") %||% FALSE] <- value

## S3 method for class 'rle'
x[[i, ..., unclass = getOption("rle.unclass_index") %||% FALSE]]

## S3 replacement method for class 'rle'
x[[i, ..., unclass = getOption("rle.unclass_index") %||% FALSE]] <- value

## S3 method for class 'rle'
x$name

## S3 replacement method for class 'rle'
x$name <- value
```

Arguments

x, i, name, value, ...	Arguments to indexing operators. See Extract documentation in the **base** package.
unclass	Logical: whether to process the arguments as if for an ordinary list; default other than FALSE can be set with options(rle.unclass_index=...).

Details

At this time, the **rle** following form of indexing are supported:

operation	index	effect
[	numeric >= 0	as vector
[	numeric < 0	no
[	logical	no
[	character	on rle
[<-	numeric >= 0	no
[<-	numeric < 0	no

[<-	logical	no
[<-	character	on rle
[[	numeric	as vector
[[<-	numeric	no
[[	character	on rle
[[<-	character	on rle
$	character	on rle
$<-	character	on rle

Generally, character indexes will access the underlying elements of the rle object, $lengths and $values.

Value

For character indices, the corresponding sublists or elements of the rle object; for numeric indices, for [[the element at the specified position and for [an rle containing the elements at the specified position(s).

Note

Some of these methods and inputs produce an error in order to future-proof code that depends on the rle package by preventing their use.

See Also

index_to_run()

Examples

```
# Indexing by character or by $ works, including sub-indexing.
x <- rle(1:5)
x["values"]] <- 2:6
x
x$values[2:3] <- 7:8
x

# From example(rle):
z <- c(TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE, TRUE, TRUE)
rle(z)
rle(z)[3:5] # Extract a sub-rle
rle(z)[[4]] # Extract an element

stopifnot(identical(inverse.rle(rle(z)[3:5]), z[3:5]))
# Fractional:
stopifnot(identical(inverse.rle(rle(z)[3.5]), z[3.5]))
# Zero:
stopifnot(identical(inverse.rle(rle(z)[0]), z[0]))
# Out of range:
stopifnot(identical(inverse.rle(rle(z)[20]), z[20]))
# A mix:
strange <- c(20, 3:5, 0, NA, 1:2)
```

```
stopifnot(identical(inverse.rle(rle(z)[strange]), z[strange]))
```

---

index_to_run *Map an element in a vector represented by an* rle *to its run*

Description

Map an element in a vector represented by an rle to its run

Usage

```
index_to_run(x, i, ...)

## S3 method for class 'rle'
index_to_run(x, i, ...)
```

Arguments

x	an rle object.
i	a numeric vector of indices to map; fractional values are rounded down.
...	additional arguments to methods.

Value

An integer vector. Negative values of i and 0 are always mapped to 0. Indexes above the range represented by x are mapped to the number of runs + 1.

Note

This function is generic for future-proofing.

Examples

```
# From example(rle):
z <- c(TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, TRUE, TRUE, TRUE)
rle(z)

stopifnot(identical(
  index_to_run(rle(z), (-1):10),
  c(0L, 0L, 1L, 1L, 2L, 2L, 3L, 4L, 5L, 5L, 5L, 6L)
))
```

Math.rle *Mathematical functions for* rle *Objects*

Description

Mathematical functions that work independently elementwise on vectors described in Math are implemented for rle objects. See Details for list of exceptions.

Usage

```
## S3 method for class 'rle'
Math(x, ...)
```

Arguments

x	An rle object.
...	Additional arguments.

Details

Supported functions include all elements of the S3 Math group excluding the "cumulative" ones, which are not supported at this time and will raise an error. As of this writing, functions supported include (from R help) abs, sign, sqrt, floor, ceiling, trunc, round, signif, exp, log, expm1, log1p, cos, sin, tan, cospi, sinpi, tanpi, acos, asin, atan, cosh, sinh, tanh, acosh, asinh, atanh, lgamma, gamma, digamma, and trigamma.

Functions cumsum, cumprod, cummax, and cummin are not supported at this time and will raise an error.

Value

In every supported case, the call should result in an rle that would have resulted had the call been applied to the original (uncompressed) vector, then compressed using rle. (At no point in the calculation is the uncompressed vector actually constructed, of course.)

By default, the functions do not merge adjacent runs with the same value. This must be done explicitly with compress.rle.

Examples

```
x <- rle(sample(runif(2), 10, c(.7,.3), replace=TRUE))

stopifnot(isTRUE(all.equal(sin(inverse.rle(x)),inverse.rle(sin(x)))))
stopifnot(inherits(try(cumprod(x)), "try-error"))
```

Ops.rle *Unary and Binary Operations for* rle *Objects*

Description

Unary and binary Arithmetic and Logic operators (with exceptions given below) are implemented between two rle objects and between an rle object and a scalar.

Usage

```
## S3 method for class 'rle'
Ops(e1, e2)
```

Arguments

e1, e2	Arguments to unary (e1) and binary (e1 and e2) operators.

Details

Supported operations include all elements of the Ops group, as well as xor. Within the Arithmetic and Logic operators, this includes (taken from the R help): +, -, *, /, ^, < , >, <=, >=, !=, ==, %%, %/%, &, |, !, and xor; but excludes non-vector logical functions and operators such as isTRUE and &&.

Value

In every supported case, the operation should result in an rle that would have resulted had the operation been applied to the original (uncompressed) vectors, then compressed using rle, with the proviso that if the resulting function creates adjacent runs of the same value, they are *not* merged. This must be done explicitly with compress.rle. (At no point in the calculation are the uncompressed vectors actually constructed, of course.)

An operation between an rle and a zero-length object produces an empty rle.

Examples

```
x <- rle(as.logical(rbinom(10,1,.7)))
y <- rle(as.logical(rbinom(10,1,.3)))

stopifnot(isTRUE(all.equal((!inverse.rle(x)),inverse.rle(!x))))

stopifnot(isTRUE(all.equal((inverse.rle(x)|inverse.rle(y)),inverse.rle(x|y))))

stopifnot(isTRUE(all.equal((inverse.rle(x)&inverse.rle(y)),inverse.rle(x&y))))

x <- rle(sample(c(-1,+1), 10, c(.7,.3), replace=TRUE))
y <- rle(sample(c(-1,+1), 10, c(.3,.7), replace=TRUE))

stopifnot(isTRUE(all.equal((inverse.rle(x)*inverse.rle(y)),inverse.rle(x*y))))
```

```
stopifnot(isTRUE(all.equal((2*inverse.rle(y)),inverse.rle(2*y))))
stopifnot(isTRUE(all.equal((inverse.rle(x)*2),inverse.rle(x*2))))

stopifnot(isTRUE(all.equal((inverse.rle(x)/inverse.rle(y)),inverse.rle(x/y))))
stopifnot(isTRUE(all.equal((2/inverse.rle(y)),inverse.rle(2/y))))
stopifnot(isTRUE(all.equal((inverse.rle(x)/2),inverse.rle(x/2))))

stopifnot(isTRUE(all.equal((-inverse.rle(y)),inverse.rle(-y))))
stopifnot(isTRUE(all.equal((inverse.rle(x)-inverse.rle(y)),inverse.rle(x-y))))

stopifnot(isTRUE(all.equal((inverse.rle(x)%/%inverse.rle(y)),inverse.rle(x%/%y))))

stopifnot(isTRUE(all.equal(inverse.rle(x)==inverse.rle(y),inverse.rle(x==y))))

stopifnot(isTRUE(all.equal((inverse.rle(x)>inverse.rle(y)),inverse.rle(x>y))))
```

`rep.rle` *A* `rep` *method for* `rle` *objects*

Description

A `rep` method for `rle` objects

Usage

```
## S3 method for class 'rle'
rep(
  x,
  ...,
  scale = c("element", "run"),
  doNotCompact = FALSE,
  doNotCompress = doNotCompact
)
```

Arguments

`x`	an `rle` object.
`...`	see documentation for `rep`.
`scale`	whether to replicate the elements of the RLE-compressed vector or the runs.
`doNotCompress, doNotCompact`	whether the method should call `compress.rle` the results before returning. Methods liable to produce very long output vectors, like `rep`, have this set `FALSE` by default. `doNotCompact` is an old name for this argument.

Note

The `rep` method for `rle` objects is very limited at this time. Even though the default setting is to replicate elements of the vector, only the run-replicating functionality is implemented at this time except for the simplest case (scalar `times` argument).

Examples

```
x <- rle(sample(c(-1,+1), 10, c(.7,.3), replace=TRUE))
y <- rpois(length(x$lengths), 2)

stopifnot(isTRUE(all.equal(rep(inverse.rle(x), rep(y, x$lengths)),
                           inverse.rle(rep(x, y, scale="run")))))

stopifnot(isTRUE(all.equal(rep(inverse.rle(x), max(y)),
                           inverse.rle(rep(x, max(y), scale="element")))))
```

---

rle-methods *Miscellaneous Common Methods for* rle *Objects*

Description

Miscellaneous Common Methods for rle Objects

Usage

```
## S3 method for class 'rle'
c(...)

## S3 method for class 'rle'
mean(x, na.rm = FALSE, ...)

## S3 method for class 'rle'
length(x)

## S3 method for class 'rle'
is.na(x)

## S3 method for class 'rle'
str(object, ...)
```

Arguments

...	For c, objects to be concatenated. The first object must be of class rle.
x, object	An rle object.
na.rm	Whether missing values are to be ignored (TRUE) or propagated (FALSE).

Note

The length method returns the length of the vector represented by the object, obtained by summing the lengths of individual runs. This can be overridden by setting options(rle.unclass_index = FALSE), which causes it to return the length of the underlying representation (usually 2) instead.

Examples

```
x <- rle(as.logical(rbinom(10,1,.7)))
y <- rle(as.logical(rbinom(10,1,.3)))

stopifnot(isTRUE(all.equal(c(inverse.rle(x),inverse.rle(y)),inverse.rle(c(x,y)))))


stopifnot(isTRUE(all.equal(mean(inverse.rle(x)),mean(x))))
stopifnot(isTRUE(all.equal(mean(inverse.rle(y)),mean(y))))


stopifnot(isTRUE(all.equal(length(inverse.rle(x)),length(x))))
stopifnot(isTRUE(all.equal(length(inverse.rle(y)),length(y))))

x$values[1] <- NA
y$values[1] <- NA
stopifnot(isTRUE(all.equal(is.na(inverse.rle(x)),inverse.rle(is.na(x)))))
stopifnot(isTRUE(all.equal(is.na(inverse.rle(y)),inverse.rle(is.na(y)))))


str(x)
```

`Summary.rle` *Summary methods for* `rle` *objects.*

Description

Summarisation functions for vectors described in Summary are implemented for `rle` objects.

Usage

```
## S3 method for class 'rle'
Summary(..., na.rm)
```

Arguments

`...`	`rle` objects or objects that can be coerced to `rle`.
`na.rm`	Whether the missing values should be ignored (`TRUE`) or propagated (`FALSE`).

Details

Supported functions include all elements of the S3 Summary group. As of this writing, functions supported include (from R help) `all`, `any`, `max`, `min`, `prod`, `range`, and `sum`.

Value

In every supported case, the call should produce the same result as what would have resulted had the call been applied to the original (uncompressed) vector. (At no point in the calculation is the uncompressed vector actually constructed, of course.) The exception is that if `values` are of class `integer`, the result will nonetheless always be upcast to `numeric` to avert overflows. This behaviour may change in the future.

Examples

```
x <- rle(as.logical(rbinom(20,1,.7)))
y <- rle(as.logical(rbinom(20,1,.3)))

stopifnot(isTRUE(all.equal(any(x, y),any(inverse.rle(x), inverse.rle(y)))))
stopifnot(isTRUE(all.equal(any(y),any(inverse.rle(y)))))

stopifnot(isTRUE(all.equal(sum(inverse.rle(x),inverse.rle(y)),sum(x,y))))
stopifnot(isTRUE(all.equal(sum(inverse.rle(y)),sum(y))))

y$values[2:3] <- NA
stopifnot(isTRUE(all.equal(sum(inverse.rle(y), na.rm=TRUE),sum(y, na.rm=TRUE))))
stopifnot(isTRUE(all.equal(sum(inverse.rle(y), na.rm=FALSE),sum(y, na.rm=FALSE))))
```

Index