Package ‘deduped’

November 29, 2025

Type Package

Title Making ``Deduplicated" Functions

Version 0.3.0

Description Contains one main function deduped() which speeds up slow, vectorized functions by only performing computations on the unique values of the input and expanding the results at the end.

License MIT + file LICENSE

Encoding UTF-8

Imports collapse, fastmatch,

RoxygenNote 7.3.3

Suggests purrr, testthat (>= 3.0.0), withr

Config/testthat/edition 3

URL https://github.com/orgadish/deduped

BugReports https://github.com/orgadish/deduped/issues

NeedsCompilation no

Author Or Gadish [aut, cre, cph]

Maintainer Or Gadish <orgadish@gmail.com>

Repository CRAN

Date/Publication 2025-11-29 07:30:02 UTC

Contents

deduped *Deduplicate a vectorized function to act on* unique *elements*

Description

Converts a vectorized function into one that only performs the computations on unique values in the first argument. The result is then expanded so that it is the same as if the computation was performed on all elements.

Usage

```
deduped(f)
```

Arguments

f	A function that accepts a vector or list as its first input.

Value

Deduplicated version of `f`.

Examples

```
x <- sample(LETTERS, 10)
x

large_x <- sample(rep(x, 10))
length(large_x)

slow_func <- function(x) {
  for (i in x) {
    Sys.sleep(0.001)
  }
  tolower(x)
}

system.time({
  y1 <- slow_func(large_x)
})

system.time({
  y2 <- deduped(slow_func)(large_x)
})

all(y1 == y2)
```

deduped_map *Apply a function to each* unique *element*

Description

DEPRECATED as of deduped 0.2.0.

Please use `deduped(lapply)()` or `deduped(purrr::map)()` instead.

Usage

```
deduped_map(.x, .f, ..., .progress = FALSE)
```

Arguments

`.x`	A list or atomic vector.
`.f`	A function, specified in one of the following ways: • A named function, e.g. `mean`. • An anonymous function, e.g. `\(x) x + 1` or `function(x) x + 1`. • A formula, e.g. `~ .x + 1`. Use `.x` to refer to the first argument. No longer recommended. • A string, integer, or list, e.g. `"idx"`, `1`, or `list("idx", 1)` which are shorthand for `\(x) pluck(x, "idx")`, `\(x) pluck(x, 1)`, and `\(x) pluck(x, "idx", 1)` respectively. Optionally supply `.default` to set a default value if the indexed element is `NULL` or does not exist. **[Experimental]** Wrap a function with `in_parallel()` to declare that it should be performed in parallel. See `in_parallel()` for more details. Use of `...` is not permitted in this context.

`...`	Additional arguments passed on to the mapped function.
	We now generally recommend against using `...` to pass additional (constant) arguments to `.f`. Instead use a shorthand anonymous function:
	`# Instead of`
	`x |> map(f, 1, 2, collapse = ",")`
	`# do:`
	`x |> map(\(x) f(x, 1, 2, collapse = ","))`
	This makes it easier to understand which arguments belong to which function and will tend to yield better error messages.

`.progress`	Whether to show a progress bar. Use `TRUE` to turn on a basic progress bar, use a string to give it a name, or see progress_bars for more details.

Value

A list whose length is the same as the length of the input, matching the output of `purrr::map()`.

See Also

deduped()

`with_deduped` *Deduplicate the first argument in an expression*

Description

This is a convenience wrapper for `deduped()` to allow it to be piped into an expression. It will recursively parse the first arguments of the expression call tree to find the bottom – when the first argument is not itself a function call.

- Without nesting: `f(x, ...) |> with_deduped()` is equivalent to `deduped(\(.z) f(.z, ...))(x)`.
- With nesting: `f(g(x, g2), f2) |> with_deduped()` is equivalent to `deduped(\(.z) f(g(.z, g2), f2))(x)`.

Usage

```
with_deduped(expr, env = parent.frame())
```

Arguments

`expr`	The expression to evaluate.
`env`	The environment within which to evaluate the expression. Can be modified when calling inside other functions.

Value

The result of evaluating the expression.

Examples

```
x <- sample(LETTERS, 10)
x

large_x <- sample(rep(x, 10))
length(large_x)

slow_func <- function(x) {
  for (i in x) {
    Sys.sleep(0.001)
  }
  tolower(x)
}

system.time({
  y1 <- slow_func(large_x)
})
```

```
system.time({
  y2 <- with_deduped(slow_func(large_x))

  # Can also use the R pipe (R >= 4.1.0) or magrittr pipe, for convenience.
  # slow_func(large_x) |> with_deduped()
})

all(y1 == y2)
```

Index