Package 'hashmap'

August 29, 2016

Type Package

Title The Faster Hash Map

Version 0.1.0

Date 2016-04-01

URL https://github.com/nathan-russell/hashmap

BugReports https://github.com/nathan-russell/hashmap/issues

Description Provides a hash table class for fast
key-value storage of atomic vector types.
Internally, hashmap makes extensive use of Rcpp, boost::variant,
and boost::unordered_map to achieve high performance, type-safety,
and versatility, while maintaining compliance with the C++98 standard.

License MIT + file LICENSE

LazyData Yes

ByteCompile TRUE

Imports Rcpp (>= 0.12.4)

LinkingTo Rcpp, BH

Suggests devtools, microbenchmark, testthat

Depends methods

RcppModules Hashmap

Collate 'hashmap.R' 'classes.R' 'Hashmap-class.R' 'zzz.R'

NeedsCompilation yes

Author Nathan Russell [aut, cre]

Maintainer Nathan Russell <russell.nr2012@gmail.com>

Repository CRAN

Date/Publication 2016-04-09 01:15:32

R topics documented:

`hashmap` *Atomic vector hash map*

Description

Create a new `Hashmap` instance

Usage

```
hashmap(keys, values, ...)
```

Arguments

`keys`	an atomic vector representing lookup keys
`values`	an atomic vector of values associated with `keys` in a pair-wise manner
`...`	other arguments passed to `new` when constructing the `Hashmap` instance

Details

The following atomic vector types are currently supported for `keys`:

- `integer`
- `numeric`
- `character`
- `Date`
- `POSIXct`

The following atomic vector types are currently supported for `values`:

- `logical`
- `integer`
- `numeric`
- `character`
- `complex`
- `Date`
- `POSIXct`

Value

a Hashmap object

See Also

Hashmap-class for a more detailed discussion of available methods

Examples

```
x <- replicate(10e3,
    paste0(sample(letters, 12, TRUE),
           collapse = "")
)
y <- rnorm(length(x))
z <- sample(x, 100)

H <- hashmap(x, y)

all.equal(y[match(z, x)], H[[z]])

## Not run:
microbenchmark::microbenchmark(
    "R" = y[match(z, x)],
    "H" = H[[z]],
    times = 500L
)

## End(Not run)
```

Hashmap-class *Internal hash map class*

Description

A C++ class providing hash map functionality for atomic vectors

Details

A Hashmap object (H) resulting from a call to hashmap(keys, values) provides the following methods accessable via $method_name:

- keys(): returns the keys of H.
- values(): returns the values of H.
- cache_keys(): caches an internal vector with the hash table's current keys resulting in very low overhead calls to keys(). For larger hash tables this has a significant effect. However, any calls to modifying functions (clear, insert, etc.) will invalidate the cached state.

- `cache_values()`: caches an internal vector with the hash table's current values resulting in very low overhead calls to `values()`. For larger hash tables this has a significant effect. However, any calls to modifying functions (`clear`, `insert`, etc.) will invalidate the cached state.
- `keys_cached()`: returns `TRUE` if the hash table's keys are currently cached, and `FALSE` otherwise.
- `values_cached()`: returns `TRUE` if the hash table's values are currently cached, and `FALSE` otherwise.
- `erase(remove_keys)`: deletes entries for elements that exist in the hash table, and ignores elements that do not.
- `clear()`: deletes all keys and values from `H`.
- `data()`: returns a named vector of `values` using the `keys` of `H` as names.
- `empty()`: returns `TRUE` if `H` is empty (e.g. immediately following a call to `clear`), else returns `FALSE`.
- `find(lookup_keys)`: returns the `values` associated with `lookup_keys` for existing key elements, and `NA` otherwise.
- `has_key(lookup_key)`: returns `TRUE` if `lookup_key` exists as a key in `H` and `FALSE` if it does not.
- `has_keys(lookup_keys)`: vectorized equivalent of `has_key`.
- `rehash(n_buckets)`: for the internal hash table, sets the number of buckets to at least `n` and the load factor to less than the max load factor.
- `bucket_count()`: returns the current number of buckets in the internal hash table.
- `hash_value(keys)`: compute hash values for the vector `keys` using the hash table's internal hash function. Note that `keys` need not exist in the hash table, but it must have the same type as the hash table's keys. This can be useful for investigating the efficacy of the object's hash function.
- `renew(new_keys, new_values)`: deletes current `keys` and `values`, and reinitialize `H` with `new_keys` and `new_values`, where `new_keys` and `new_values` are allowed to be different SEXP types than the original `keys` and `values`.
- `insert(more_keys, more_values)`: adds more key-value pairs to `H`, where existing key elements (`intersect(H$keys(), more_keys)`) will be updated with the corresponding elements in `more_values`, and non-existing key elements `setdiff(H$keys(), more_keys)` will be inserted with the corresponding elements in `more_values`.
- `size()`: returns the size (number of key-value pairs) of (held by) `H`.

Additionally, the following two convenience methods which do not require the use of `$`:

- `` `[[` ``: equivalent to `find(lookup_keys)`.
- `` `[[<-` ``: equivalent to `insert(more_keys, more_values)`.

Examples

```
x <- replicate(10e3,
    paste0(sample(letters, 12, TRUE),
           collapse = "")
)
y <- rnorm(length(x))
z <- sample(x, 100)

H <- hashmap(x, y)

H$empty()       #[1] FALSE
H$size()        #[1] 10000

## necessarily
any(duplicated(H$keys()))        #[1] FALSE

all.equal(H[[z]], H$find(z))  #[1] TRUE

## hash map ordering is random
all.equal(
    sort(H[[x]]),
    sort(H$values()))  #[1] TRUE

## a named vector
head(H$data())

## redundant, but TRUE
all.equal(
    H[[names(head(H$data()))]],
    unname(head(H$data())))

## setting values
H2 <- hashmap(H$keys(), H$values())

all.equal(
    sort(H[[H2$keys()]]),
    sort(H2[[H$keys()]]))   #[1] TRUE

H$insert("A", round(pi, 5))

H2[["A"]] <- round(pi, 5)

## still true
all.equal(
    sort(H[[H2$keys()]]),
    sort(H2[[H$keys()]]))

## changing SEXPTYPE of key or value must be explicit
H3 <- hashmap(c("A", "B", "C"), c(1, 2, 3))

H3$size()     #[1] 3
```

```
H3$clear()
H3$size()     #[1] 0

## not allowed
class(try(H3[["D"]] <- "text", silent = TRUE)) #[1] "try-error"

## okay
H3$renew("D", "text")
H3$size()     #[1] 1
```

Rcpp_Hashmap-class *Hashmap internal class*

Description

Hashmap internal class

Index