

Package ‘intmap’

July 22, 2025

Type Package

Title Ordered Containers with Integer Keys

Version 1.0.0

Author Stéphane Laurent

Maintainer Stéphane Laurent <laurent_step@outlook.fr>

Description Provides a key-value store data structure. The keys are integers and the values can be any R object. This is like a list but indexed by a set of integers, not necessarily contiguous and possibly negative. The implementation uses a 'R6' class. These containers are not faster than lists but their usage can be more convenient for certain situations.

License GPL-3

URL <https://github.com/stla/intmap>

BugReports <https://github.com/stla/intmap/issues>

Imports maybe, methods, R6, Rcpp (>= 1.0.8)

LinkingTo BH, Rcpp

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-04-17 10:00:10 UTC

Contents

intmap	2
intmap-imports	11

Index	12
--------------	-----------

intmap	<i>R6 class representing an ordered map</i>
--------	---

Description

A map is given by keys and values.

Methods

Public methods:

- `intmap$new()`
- `intmap$print()`
- `intmap$size()`
- `intmap$keys()`
- `intmap$values()`
- `intmap$items()`
- `intmap$toList()`
- `intmap$at()`
- `intmap$get()`
- `intmap$index()`
- `intmap$extract()`
- `intmap$has_key()`
- `intmap$nth()`
- `intmap$insert()`
- `intmap$erase()`
- `intmap$merge()`
- `intmap$copy()`

Method `new()`: Creates a new intmap object.

Usage:

```
intmap$new(keys = NULL, values)
```

Arguments:

`keys` keys, an integer vector without NA value

`values` values, a list of R objects; keys and values must have the same length

Returns: An intmap object.

Examples:

```
intmap$new() # empty map
intmap$new(
  keys = c(4, -2),
  values = list(c(1, 2), c("a", "b", "c"))
)
# examples with duplicated keys:
```

```
intmap$new(  
  keys = c(1, 1, 5),  
  values = list(c(1, 2), c(3, 4), "x")  
)
```

Method print(): Show instance of an intmap object.

Usage:

```
intmap$print(...)
```

Arguments:

... ignored

Method size(): Size of the reference map.

Usage:

```
intmap$size()
```

Returns: An integer, the number of entries.

Examples:

```
imap <- intmap$new(  
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))  
)  
imap$size()
```

Method keys(): Get all keys.

Usage:

```
intmap$keys()
```

Returns: The keys, an integer vector.

Examples:

```
imap <- intmap$new(  
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))  
)  
imap$keys()
```

Method values(): Get all values.

Usage:

```
intmap$values()
```

Returns: The values, a list of R objects.

Examples:

```
imap <- intmap$new(  
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))  
)  
imap$values()
```

Method items(): Get all entries of the reference map.

Usage:

```
intmap$items()
```

Returns: The entries in a dataframe.

Examples:

```
imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$items()
```

Method `toList()`: Converts the map to a named list.

Usage:

```
intmap$toList()
```

Returns: A named list (the names are the keys).

Examples:

```
imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$toList()
```

Method `at()`: Returns the 'maybe' value corresponding to the given key.

Usage:

```
intmap$at(key)
```

Arguments:

`key` a key (integer)

Returns: A maybe value, either the value corresponding to the key as a 'Just' maybe value if the key is found, otherwise the 'Nothing' maybe value.

Examples:

```
imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$at(11)
from_just(imap$at(11))
imap$at(4)
```

Method `get()`: Get the value corresponding to the given key or a default value if this key is missing.

Usage:

```
intmap$get(key, default = NULL)
```

Arguments:

`key` a key (integer)

`default` a R object, the default value

Returns: Either the value corresponding to the key if the key is found, otherwise the default value.

Examples:

```
imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$get(11, default = 999)
imap$get(4, default = 999)
```

Method `index()`: Returns the index of the given key.

Usage:

```
intmap$index(key)
```

Arguments:

key a key (integer)

Returns: The index of the key, or NA if it is not found.

Examples:

```
imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$index(11)
imap$index(4)
```

Method `extract()`: Extract a submap from the reference map.

Usage:

```
intmap$extract(keys, inplace = FALSE, bydeleting = FALSE)
```

Arguments:

keys some keys, an integer vector; those which do not belong to the keys of the reference map will be ignored

inplace Boolean, whether to update the reference map or to return a new map

bydeleting Boolean, whether to construct the submap by deleting the keys which are not in keys or by starting from the empty submap and adding the entries

Returns: An intmap object if inplace=FALSE, otherwise the updated reference map, invisibly.

Examples:

```
imap <- intmap$new(
  keys = c(11, -2, 3), values = list(c("a", "b"), list(3, 4, 5), "X")
)
imap_copy <- imap$copy()
imap$extract(c(11, 3))
imap
imap$extract(c(11, 3), inplace = TRUE)
imap
imap_copy$extract(c(11, 3), bydeleting = TRUE)
imap_copy
imap_copy$extract(c(11, 3), inplace = TRUE, bydeleting = TRUE)
imap_copy
```

Method `has_key()`: Checks whether a key exists in the reference map.

Usage:

```
intmap$has_key(key)
```

Arguments:

key a key (integer)

Returns: A Boolean value.

Examples:

```
imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$has_key(11)
imap$has_key(1)
```

Method `nth()`: Returns the n-th entry of the reference map.

Usage:

```
intmap$nth(n, stop_if_too_large = TRUE)
```

Arguments:

n index, a positive integer

stop_if_too_large a Boolean value, whether to stop if n is too large, or to use maybe values

Returns: A list with the key and the value at index n if stop_if_too_large=TRUE and n is not too large, otherwise a maybe value: either this list wrapped in a 'Just' container, or 'Nothing'.

Examples:

```
imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$nth(2)
imap$nth(2, stop_if_too_large = FALSE)
imap$nth(9, stop_if_too_large = FALSE)
```

Method `insert()`: Insert a new entry in the reference map.

Usage:

```
intmap$insert(key, value, replace = FALSE)
```

Arguments:

key a key (integer)

value a value (R object)

replace Boolean, whether to replace the value if the key is already present

Returns: This updates the reference map and this returns a Boolean value: if replace=FALSE, this returns TRUE if the value has been inserted (i.e. the given key is new); similarly, if replace=TRUE, this returns TRUE if the given key is new (so FALSE means that the value of the existing key has been replaced).

Examples:

```

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$insert(3, c(6, 7)) # TRUE (insertion)
imap
imap$insert(11, c(8, 9)) # FALSE (no change)
imap
imap$insert(11, c(8, 9), replace = TRUE) # FALSE (replacement)
imap

```

Method `erase()`: Erase the entries of the reference map whose keys are the given ones.

Usage:

```
intmap$erase(keys)
```

Arguments:

keys some keys, an integer vector; those which do not belong to the keys of the reference map are ignored

Returns: The reference map, invisibly.

Examples:

```

imap <- intmap$new(
  keys = c(11, -2, 3), values = list(c("a", "b"), list(3, 4, 5), "X")
)
imap$erase(11)
imap
imap$erase(c(-2, 3))
imap

```

Method `merge()`: Merge the reference map with another map.

Usage:

```
intmap$merge(map)
```

Arguments:

map an intmap object

Returns: The updated reference map, invisibly. Keys of map that are also keys of the reference map are ignored, i.e. there is no replacement, only insertions.

Examples:

```

imap1 <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap2 <- intmap$new(
  keys = c(11, 3), values = list("X", "Z")
)
imap1$merge(imap2)
imap1

```

Method `copy()`: Copy the reference map.

Usage:

```
intmap$copy()
```

Returns: A copy of the reference map.

Examples:

```
imap <- intmap$new(
  keys = c(11, 3), values = list(TRUE, "Z")
)
true_copy <- imap$copy()
true_copy$erase(11)
imap
naive_copy <- imap
naive_copy$erase(11)
imap
```

Examples

```
## -----
## Method `intmap$new`
## -----

intmap$new() # empty map
intmap$new(
  keys = c(4, -2),
  values = list(c(1, 2), c("a", "b", "c"))
)
# examples with duplicated keys:
intmap$new(
  keys = c(1, 1, 5),
  values = list(c(1, 2), c(3, 4), "x")
)

## -----
## Method `intmap$size`
## -----

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$size()

## -----
## Method `intmap$keys`
## -----

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$keys()

## -----
## Method `intmap$values`
## -----
```

```

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$values()

## -----
## Method `intmap$items`
## -----

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$items()

## -----
## Method `intmap$toList`
## -----

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$toList()

## -----
## Method `intmap$at`
## -----

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$at(11)
from_just(imap$at(11))
imap$at(4)

## -----
## Method `intmap$get`
## -----

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$get(11, default = 999)
imap$get(4, default = 999)

## -----
## Method `intmap$index`
## -----

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$index(11)

```

```

imap$index(4)

## -----
## Method `intmap$extract`
## -----

imap <- intmap$new(
  keys = c(11, -2, 3), values = list(c("a", "b"), list(3, 4, 5), "X")
)
imap_copy <- imap$copy()
imap$extract(c(11, 3))
imap
imap$extract(c(11, 3), inplace = TRUE)
imap
imap_copy$extract(c(11, 3), bydeleting = TRUE)
imap_copy
imap_copy$extract(c(11, 3), inplace = TRUE, bydeleting = TRUE)
imap_copy

## -----
## Method `intmap$has_key`
## -----

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$has_key(11)
imap$has_key(1)

## -----
## Method `intmap$nth`
## -----

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$nth(2)
imap$nth(2, stop_if_too_large = FALSE)
imap$nth(9, stop_if_too_large = FALSE)

## -----
## Method `intmap$insert`
## -----

imap <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap$insert(3, c(6, 7)) # TRUE (insertion)
imap
imap$insert(11, c(8, 9)) # FALSE (no change)
imap
imap$insert(11, c(8, 9), replace = TRUE) # FALSE (replacement)
imap

```

```
## -----
## Method `intmap$erase`
## -----

imap <- intmap$new(
  keys = c(11, -2, 3), values = list(c("a", "b"), list(3, 4, 5), "X")
)
imap$erase(11)
imap
imap$erase(c(-2, 3))
imap

## -----
## Method `intmap$merge`
## -----

imap1 <- intmap$new(
  keys = c(11, -2), values = list(c("a", "b"), list(3, 4, 5))
)
imap2 <- intmap$new(
  keys = c(11, 3), values = list("X", "Z")
)
imap1$merge(imap2)
imap1

## -----
## Method `intmap$copy`
## -----

imap <- intmap$new(
  keys = c(11, 3), values = list(TRUE, "Z")
)
true_copy <- imap$copy()
true_copy$erase(11)
imap
naive_copy <- imap
naive_copy$erase(11)
imap
```

intmap-imports

Extract value from a 'Just' value

Description

The `from_just` function is imported from the **maybe** package. Follow the link to its documentation: [from_just](#). It has been imported for convenient use of the `intmap$at` method, which returns a 'Just' value.

Index

`from_just`, [11](#)

`from_just (intmap-imports)`, [11](#)

`intmap`, [2](#)

`intmap-imports`, [11](#)