Package ‘triebeard’

August 29, 2016

Type Package
Title 'Radix' Trees in 'Rcpp'
Version 0.3.0
Author Oliver Keyes [aut, cre], Drew Schmidt [aut], Yuuki Takano [cph]
Maintainer Oliver Keyes <ironholds@gmail.com>
Description 'Radix trees', or 'tries', are key-value data structures optimised for efficient lookups, similar in purpose to hash tables. 'triebeard' provides an implementation of 'radix trees' for use in R programming and in developing packages with 'Rcpp'.
License MIT + file LICENSE
LazyData TRUE
LinkingTo Rcpp
Imports Rcpp
RoxygenNote 5.0.1
Suggests knitr, rmarkdown, testthat
VignetteBuilder knitr
URL https://github.com/Ironholds/triebeard/
BugReports https://github.com/Ironholds/triebeard/issues
Date 2016-08-03
NeedsCompilation yes
Repository CRAN
Date/Publication 2016-08-04 00:57:37

R topics documented:

alter ................................................................. 2
getters .................................................................. 3
greedy_match ....................................................... 3
Description
   `trie_add` and `trie_remove` allow you to add or remove entries from tries, respectively.

Usage
   `trie_add(trie, keys, values)
   trie_remove(trie, keys)

Arguments
   trie          a trie object created with `trie`
   keys          a character vector containing the keys of the entries to add (or remove). Entries with NA keys will not be added.
   values        an atomic vector, matching the type of the trie, containing the values of the entries to add. Entries with NA values will not be added.

Value
   nothing; the trie is modified in-place

See Also
   `trie` for creating tries in the first place.

Examples
   trie <- trie("foo", "bar")
   length(trie)
   
   trie_add(trie, "baz", "qux")
   length(trie)
   
   trie_remove(trie, "baz")
   length(trie)
getters

_Trie Getters_

**Description**

"Getters" for the data stored in a trie object. `get_keys` gets the keys, `get_values` gets the values.

**Usage**

- `get_keys(try)`
- `get_values(try)`

**Arguments**

- `trie` A trie object, created with `trie`.

**Value**

An atomic vector of keys or values stored in the trie.

---

**greedy_match**

_Greedily match against a tree_

**Description**

greedy_match accepts a trie and a character vector and returns the values associated with any key that is "greedily" (read: fuzzily) matched against one of the character vector entries.

**Usage**

- `greedy_match(try, to_match)`

**Arguments**

- `trie` a trie object, created with `trie`.
- `to_match` a character vector containing the strings to check against the trie’s keys.

**Value**

A list, the length of `to_match`, with each entry containing any trie values where the `to_match` element greedily matches the associated key. In the case that nothing was found, the entry will contain NA.
longest_match

See Also

longest_match and prefix_match for longest and prefix matching, respectively.

Examples

```r
trie <- trie(keys = c("afford", "affair", "available", "binary", "bind", "blind"),
             values = c("afford", "affair", "available", "binary", "bind", "blind"))
greedy_match(trie, c("avoid", "bring", "attack"))
```

longest_match  Find the longest match in a trie

Description

longest_match accepts a trie and a character vector and returns the value associated with whichever key had the longest match to each entry in the character vector. A trie of "binary" and "bind", for example, with an entry-to-compare of "binder", will match to "bind".

Usage

```r
longest_match(trie, to_match)
```

Arguments

- `trie`: a trie object, created with `trie`
- `to_match`: a character vector containing the strings to match against the trie's keys.

See Also

prefix_match and greedy_match for prefix and greedy matching, respectively.

Examples

```r
trie <- trie(keys = c("afford", "affair", "available", "binary", "bind", "blind"),
             values = c("afford", "affair", "available", "binary", "bind", "blind"))
longest_match(trie, "binder")
```
prefix_match

Find the prefix matches in a trie

Description

prefix_match accepts a trie and a character vector and returns the values associated with any key that has a particular character vector entry as a prefix (see the examples).

Usage

prefix_match(trie, to_match)

Arguments

trie a trie object, created with trie
to_match a character vector containing the strings to check against the trie’s keys.

Value

a list, the length of to_match, with each entry containing any trie values where the to_match element was a prefix of the associated key. In the case that nothing was found, the entry will contain NA.

See Also

longest_match and greedy_match for longest and greedy matching, respectively.

Examples

trie <- trie(keys = c("afford", "affair", "available", "binary", "bind", "blind"),
              values = c("afford", "affair", "available", "binary", "bind", "blind"))
prefix_match(trie, "aff")

create_trie

Create a Trie

Description

create_trie creates a trie (a key-value store optimised for matching) out of a provided character vector of keys, and a numeric, character, logical or integer vector of values (both the same length).

Usage

trie(keys, values)
Arguments

- **keys**: a character vector containing the keys for the trie.
- **values**: an atomic vector of any type, containing the values to pair with keys. Must be the same length as keys.

Value

- a `trie` object.

See Also

- `trie_add` and `trie_remove` for adding to and removing from tries after their creation, and `longest_match` and other match functions for matching values against the keys of a created trie.

Examples

```r
# An integer trie
int_trie <- trie(keys = "foo", values = 1)

# A string trie
str_trie <- trie(keys = "foo", values = "bar")
```

Description

This package provides access to Radix tree (or "trie") structures in Rcpp. At a later date it will hopefully provide them in R, too.
Index

alter, 2

get_keys (getters), 3
get_values (getters), 3
getters, 3

greedy_match, 3, 4, 5

longest_match, 4, 4, 5, 6

prefix_match, 4, 5

trie, 2-5, 5

trie_add, 6

trie_add (alter), 2

trie_remove, 6

trie_remove (alter), 2

triebeard, 6

triebeard-package (triebeard), 6