

# Package ‘triebeard’

May 8, 2026

**Type** Package

**Title** 'Radix' Trees in 'Rcpp'

**Version** 0.4.1

**Author** Os Keyes [aut, cre], Drew Schmidt [aut], Yuuki Takano [cph]

**Maintainer** Os 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

**LinkingTo** Rcpp

**Encoding** UTF-8

**Imports** Rcpp

**RoxygenNote** 7.1.2

**Suggests** knitr, rmarkdown, testthat

**VignetteBuilder** knitr

**URL** <https://github.com/Ironholds/triebeard/>

**BugReports** <https://github.com/Ironholds/triebeard/issues>

**Date** 2023-03-04

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2023-03-04 23:30:07 UTC

## Contents

alter . . . . .	2
getters . . . . .	3
greedy_match . . . . .	3

longest_match . . . . .	4
prefix_match . . . . .	5
trie . . . . .	6
triebeard . . . . .	6

<b>Index</b>	<b>7</b>
--------------	----------

---

alter	<i>Add or remove trie entries</i>
-------	-----------------------------------

---

## 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 <a href="#">trie</a>
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	<i>Trie Getters</i>
---------	---------------------

---

**Description**

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

**Usage**

```
get_keys(trie)
```

```
get_values(trie)
```

**Arguments**

`trie` A trie object, created with [trie](#).

**Value**

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

---

greedy_match	<i>Greedily match against a tree</i>
--------------	--------------------------------------

---

**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(trie, to_match, include_keys = FALSE)
```

**Arguments**

`trie` a trie object, created with [trie](#)

`to_match` a character vector containing the strings to check against the trie's keys.

`include_keys` a logical value indicating whether to include the keys in the returned results or not. If TRUE (*not* the default) the returned object will be a list of `data.frames`, rather than of vectors.

**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. In the case that `include_keys` is TRUE, the matching keys will also be included

**See Also**

[longest\\_match](#) and [prefix\\_match](#) for longest and prefix matching, respectively.

**Examples**

```
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	<i>Find the longest match in a trie</i>
---------------	---

---

**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**

```
longest_match(trie, to_match, include_keys = FALSE)
```

**Arguments**

trie	a trie object, created with <a href="#">trie</a>
to_match	a character vector containing the strings to match against the trie's keys.
include_keys	a logical value indicating whether to include the keys in the returned results or not. If TRUE ( <i>not</i> the default) the returned object will be a data.frame, rather than a vector.

**See Also**

[prefix\\_match](#) and [greedy\\_match](#) for prefix and greedy matching, respectively.

**Examples**

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

---

prefix_match	<i>Find the prefix matches in a trie</i>
--------------	--

---

## 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, include_keys = FALSE)
```

## Arguments

trie	a trie object, created with <a href="#">trie</a>
to_match	a character vector containing the strings to check against the trie's keys.
include_keys	a logical value indicating whether to include the keys in the returned results or not. If TRUE ( <i>not</i> the default) the returned object will be a list of data.frames, rather than of vector.

## 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")
```

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**

<code>keys</code>	a character vector containing the keys for the trie.
<code>values</code>	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**

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

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

---

triebeard*Radix trees in Rcpp*

---

**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, 6

`trie_add`, 6

`trie_add (alter)`, 2

`trie_remove`, 6

`trie_remove (alter)`, 2

`triebeard`, 6

`triebeard-package (triebeard)`, 6