Package ‘oak’
April 22, 2019

Type Package
Title Trees Creation and Manipulation
Version 0.2.3
Date 2019-04-22
Description Functions and classes to create and manipulate trees and nodes.
License MIT + file LICENSE
LazyData TRUE
Depends R (>= 3.1.3)
Imports foreach, purrr, R6, rlang, rlist, rstudioapi, stats
Suggests knitr, testthat
URL https://github.com/paulponcet/oak
BugReports https://github.com/paulponcet/oak/issues
RoxygenNote 6.1.0
NeedsCompilation no
Author Paul Poncet [aut, cre]
Maintainer Paul Poncet <paulponcet@yahoo.fr>
Repository CRAN
Date/Publication 2019-04-22 20:50:42 UTC

R topics documented:

ancestors ........................................... 2
as.list.rtree ....................................... 3
as.node ........................................... 4
as.rtree ........................................... 5
chain .............................................. 6
children .......................................... 7
cut_leaves ....................................... 7
descendants ...................................... 8
The function `ancestors` returns the ancestors of a node in a given tree.

Usage

```r
ancestors(.node, .tree, include_node = FALSE)
```

## S3 method for class 'rtree'
ancestors(.node, .tree, include_node = FALSE)

Arguments

- `.node` node or character. The node or node label considered.
- `.tree` a tree.
- `include_node` logical. If FALSE (the default), `.node` is not part of the list returned.
as.list.rtree

Value

A (possibly empty) list of nodes.

Examples

### Rooted tree
(tr0 = c_(Bob, "Carl", "Daniel"))
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
(tr2 = r_("Alice", s = list(tr0, tr1)))
ancestors("Alice", tr2)
ancestors("Daniel", tr2, include_node = TRUE)

### Unrooted tree
(tr3 = r_(s = list(tr2, c_("Grand-Mother", "Father", "Son"))))
ancestors("Alice", tr3)
ancestors("Alice", tr3, include_node = TRUE)
ancestors("Daniel", tr3)
ancestors("Son", tr3)
ancestors("Son", tr3, include_node = TRUE)

---

### Conversion of a tree to a list

Description

This function converts a tree to a list.

Usage

```r
## S3 method for class 'rtree'
as.list(x, recursively = FALSE, ...)
```

Arguments

- `x` A tree.
- `recursively` logical. See below.
- `...` Additional arguments (not used).

Value

A list. If recursively=FALSE, this list is made up of the subtrees of `x`. If recursively=TRUE, these subtrees are themselves recursively converted to a list.
Examples

(tr0 = c_("Bob", "Carl", "Daniel"))
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
(tr2 = r_("Alice", s = list(tr0, tr1)))
as.list(tr2)
as.list(tr2, rec = TRUE)

## Unrooted tree
(tr3 = r_(s = list(tr2, c_("Grand-Mother", "Father", "Son"))))
as.list(tr3)
as.list(tr3, rec = TRUE)

as.node  Conversion to a node

Description

These methods convert an object to a node. A node is defined as an rtree object with no subtrees.

Usage

as.node(x, ...)

## S3 method for class 'character'
as.node(x, ...)

## S3 method for class 'tree'
as.node(x, ...)

Arguments

x  An object to be converted.
...	Additional parameters.

Value

A node.

Examples

## Rooted tree
(tr0 = c_("Bob", "Carl", "Daniel"))
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
(tr2 = r_("Alice", s = list(tr0, tr1)))
as.node(tr2) # the root of 'tr2'

## Unrooted tree
as.rtree

(tr3 = r_(s = list(tr2, c_("Grand-Mother", "Father", "Son"))))
## Not run:
as.node(tr3) # generates an error since 'tr3' is unrooted

## End(Not run)

---

as.rtree

Conversion to an 'rtree' object

Description

These methods convert an object to an rtree object.

Usage

as.rtree(x, ...)

## S3 method for class 'rtree'
as.rtree(x, ...)

## S3 method for class 'character'
as.rtree(x, ...)

## S3 method for class 'data.frame'
as.rtree(x, ...)

Arguments

x An object to be converted.
...
Additional parameters (not used).

Value

An rtree object.

Examples

## Rooted tree
df = data.frame(x = c("A", "A", "A", "A"),
y = c("B", "C", "C", "C"),
z = c("D", "E", "E", "F"),
stringsAsFactors = FALSE)

(as.rtree(df))

## Unrooted tree
df = data.frame(x = c("A", "A", "A", "X"),
y = c("B", "C", "C", "Y"),
...
z = c("D", "E", "E", "Z"),
stringsAsFactors = FALSE)

(as.rtree(df))

---

### chain

#### Chain constructor

**Description**

The function `chain` creates an 'rtree' object which is a chain, i.e. a totally ordered tree.

**Usage**

```r
chain(...)  
## S3 method for class 'rtree'
chain(...)  
## S3 method for class 'list'
chain(...)  
## S3 method for class 'numeric'
chain(...)  
## S3 method for class 'character'
chain(...)  
c_(...)  
```

**Arguments**

... characters or nodes.

**Value**

A chain.
**children**

*Children of a node*

**Description**

The function `children` returns the children of a node in a given tree.

**Usage**

```
children(.node, .tree, degree = 1L)
```

**Arguments**

- `.node` node or character. The node or node label considered.
- `.tree` A tree.
- `degree` integer. Currently not used.

**Value**

A (possibly empty) list of nodes.

---

**cut_leaves**

*Cut the leaves of a tree*

**Description**

The function `cut_leaves` cuts the leaves in `.tree`.

**Usage**

```
cut_leaves(.tree)
```

**Arguments**

- `.tree` A tree.

**Value**

A tree.
### descendants

**Descendants of a node**

**Description**

The function `descendants` returns the descendants of a node in a given tree.

**Usage**

```r
descendants(.node, .tree, include_node = FALSE)
```

#### S3 method for class 'rtree'

```r
descendants(.node, .tree, include_node = FALSE)
```

**Arguments**

- `.node`: node or character. The node or node label considered.
- `.tree`: A tree.
- `include_node`: logical. Currently not used.

**Value**

A (possibly empty) list of nodes.

### flatten

**Flatten a tree**

**Description**

The function `flatten` returns all the nodes that compose a given tree.

**Usage**

```r
flatten(.tree)
```

#### S3 method for class 'rtree'

```r
flatten(.tree)
```

**Arguments**

- `.tree`: A tree to be flattened.

**Value**

A (possibly empty) list of nodes.
**Examples**

```
## Rooted tree
(tr0 = c_("Bob", "Carl", "Daniel"))
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
(tr2 = r_("Alice", s = list(tr0, tr1)))
flatten(tr2)

## Unrooted tree
(tr3 = r_(s = list(tr2, c_("Grand-Mother", "Father", "Son"))))
flatten(tr3)
```

---

**Description**

This function returns the height of a tree, defined as the number of nodes of the longest subchain. For an empty tree, the height is conventionally equal to 0.

**Usage**

```r
height(.tree)
```

```
## S3 method for class 'rtree'
height(.tree)
```

```r
height(x) <- value
```

**Arguments**

- `.tree`, `x` A tree.
- `value` integer. Height to assign to the tree `x` (this calls the function `prune`).

**Value**

An integer.

**Examples**

```
## Rooted tree
(tr0 = c_("Bob", "Carl", "Daniel"))
(tr1 = r_("Dimitri", s = list(c_("Enoc"), c_("Ferdinand")))
(tr1 = r_("Caroline", s = list(tr1)))
(tr1 = r_("Bill", s = list(tr1)))
(tr2 = r_("Alice", s = list(tr0, tr1)))
height(empty_tree())
height(tr0)
```
is.binary_tree

Test if a tree is a binary tree

Description

The function `is.binary_tree` tests if a tree is binary.

Usage

`is.binary_tree(x)`

Arguments

`x` A tree to be tested.

Value

A logical.

Examples

```r
## FALSE
is.binary_tree(empty_tree())

## FALSE
tr = r_(s = list(r_("toto"), r_("tata")))
is.binary_tree(tr) # unrooted tree

## TRUE
tr = r_("titi", s = list(r_("toto"), r_("tata")))
is.binary_tree(tr)
```
is.chain

Test if a tree is a chain

Description

The function is.chain returns TRUE if x is a chain, FALSE otherwise.

Usage

is.chain(x)

## S3 method for class 'rtree'

is.chain(x)

Arguments

x  
A tree to be tested.

Value

A logical.

Examples

## FALSE
is.chain(empty_tree())

## TRUE
(tr0 = c_('Bob', 'Carl', 'Daniel'))
is.chain(tr0)

## FALSE
(tr1 = r_(s = list(tr0)))
is.chain(tr1)

## FALSE
(tr = r_("titi", s = list(r_("toto"), r_("tata"))))
is.chain(tr)
**is.node**

*Test if an object is a node*

**Description**

The function `is.node` returns TRUE if `x` is a node, FALSE otherwise. A 'node' is just an `rtree` object with no subtrees.

**Usage**

`is.node(x)`

**Arguments**

`x` An object to be tested.

**Value**

A logical.

---

**is.root**

*Test if a node is a root of a tree*

**Description**

The function `is.root` returns TRUE if `.node` is a root of `.tree`, FALSE otherwise.

**Usage**

`is.root(.node, .tree)`

**Arguments**

`.node` A node or node label to be tested.

`.tree` A tree

**Value**

A logical.
is.rooted

Test if a tree has a root

Description
The function \texttt{is.rooted} returns \texttt{TRUE} if \texttt{.tree} is a rooted tree, \texttt{FALSE} otherwise.

Usage
\texttt{is.rooted(.tree)}

Arguments
\texttt{.tree} \hspace{1cm} A tree to be tested.

Value
A logical.

is.tree

Test if an object is a tree

Description
The function \texttt{is.tree} returns \texttt{TRUE} if \texttt{x} is a tree, \texttt{FALSE} otherwise.

Usage
\texttt{is.tree(x)}
\texttt{is.rtree(x)}

Arguments
\texttt{x} \hspace{1cm} An object to be tested.

Value
A logical.

Examples
\begin{verbatim}
## FALSE
is.tree(empty_tree())
\end{verbatim}
**label**  
*Labels of nodes and trees*

### Description
Get the label of a node or the labels of all nodes of a tree.

### Usage

```r
label(Nnode)
label(x) <- value
```

```r
## S3 method for class 'rtree'
labels(object, ...)
```

### Arguments

- `.node`, `x`  
  A node.

- `value`  
  Character. New label to be applied to the node.

- `...`  
  Additional arguments (not used).

- `.tree`, `object`  
  A tree.

### Examples

```r
# Rooted tree
(tr0 = c_("Bob", "Carl", "Daniel"))
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
(tr2 = r_("Alice", s = list(tr0, tr1)))
labels(tr0)
labels(tr1)
labels(tr2)

# Unrooted tree
(tr3 = r_(s = list(tr2, c_("Grand-Mother", "Father", "Son"))))
labels(tr3)
```

---

**leaves**  
*Leaves of a tree*

### Description
The function `leaves` returns the leaves of a tree.
Usage

leaves(.tree)

## S3 method for class 'rtree'
leaves(.tree)

is_leafnode(.node, .tree)

Arguments

.tree A tree.

.node A node of .tree.

Value

A (possibly empty) list of nodes.

Examples

## Chains
(tr0 = c_("Bob", "Carl", "Daniel"))
leaves(tr0)
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
leaves(tr1)

## Rooted tree
(tr2 = r_("Alice", s = list(tr0, tr1)))
leaves(tr2)

## Unrooted tree
(tr3 = r_(s = list(tr2, c_("Grand-Mother", "Father", "Son"))))
leaves(tr3)

---

Nodes Class

<table>
<thead>
<tr>
<th>Description</th>
<th>Nodes class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>Nodes</td>
</tr>
<tr>
<td>Format</td>
<td>An object of class R6ClassGenerator of length 24.</td>
</tr>
</tbody>
</table>
Fields

labels character. A vector of labels for each node. The labels must be part of the tree’s labels.

tree rtree. The tree where the nodes belong to.

... Additional parameters (currently not used).

Methods

get_labels() Get labels of the nodes
get_tree() Get the tree attached to the nodes
set_tree(tree) Set the tree
keep(i)

See Also

Tree and Node in this package.

---

parent Parent of a node

Description

The function parent returns the parent of a node in a given tree. If the node is not found in the tree or has no parent, the empty tree is returned.

Usage

parent(.node, .tree, degree = 1L)

## S3 method for class 'rtree'
parent(.node, .tree, degree = 1L)

has.parent(.node, .tree)

Arguments

.node node or character. The node or node label considered.
.tree A tree.
.degree integer. Currently not used.

Value

A node.
**print.rtree**

**Description**

This function prints the labels of the nodes of a tree, and displays the connections between these nodes.

**Usage**

```r
## S3 method for class 'rtree'
print(x, at = NULL, level = 1L, ...)
```

**Arguments**

- `x` A tree.
- `at` character. If not `NULL`, the name of an attribute of the nodes of `x` to be printed next to the node labels.
- `level` integer. This argument is used internally by the function, should not be used directly.
- `...` Additional parameters (not used).

**Value**

`x` is return invisibly.

**See Also**

For examples using the `at` argument, see `tree_apply`.

---

**Examples**

```r
## Rooted tree
(tr0 = c_("Bob", "Carl", "Daniel"))
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
(tr2 = r_("Alice", s = list(tr0, tr1)))

## Unrooted tree
(tr3 = r_(s = list(tr2, c_("Son", "Father", "Grand-Mother"))))
parent("Alice", tr3)
parent("Bob", tr3)
parent("any node", tr3)
```
prune  
\textit{Prune a tree}

\textbf{Description}

The function \texttt{prune} removes nodes in a tree whose height is greater than a given threshold.

\textbf{Usage}

\begin{verbatim}
prune(.tree, max_height = 1L)
## S3 method for class 'rtree'
prune(.tree, max_height = 1L)
\end{verbatim}

\textbf{Arguments}

- \texttt{.tree}  
  A tree to be pruned.
- \texttt{max_height}  
  integer. The height imposed to the new tree.

\textbf{Value}

The pruned tree.

rev.rtree  
\textit{Reverse a chain}

\textbf{Description}

This function reverses the order of the nodes in a chain.

\textbf{Usage}

\begin{verbatim}
## S3 method for class 'rtree'
rev(x)
\end{verbatim}

\textbf{Arguments}

- \texttt{x}  
  A chain to be reversed.

\textbf{Value}

The reversed chain.

\textbf{Examples}

\begin{verbatim}
(tr0 = c("Bob", "Carl", "Daniel"))
(rev(tr0))
\end{verbatim}
**root**  
*Root(s) of a tree*

**Description**

Find the root of a tree (or the multiple roots for a non-rooted tree)

**Usage**

```
root(.tree)
```

```r
## S3 method for class 'rtree'
root(.tree)
```

```
roots(.tree)
```

```r
## S3 method for class 'rtree'
roots(.tree)
```

**Arguments**

- `.tree`  
  A tree.

**Value**

- `root` returns a node, the root of the tree if it exists; if `.tree` is not rooted, an error is thrown.
- `roots` returns a list of nodes, the roots of the tree.

---

**rtree**  
*Tree constructor*

**Description**

The function `rtree` creates an `rtree` (recursive tree) object.

**Usage**

```
rtree(label = NULL, subtrees = list(), ...)
```

```
r_(label = NULL, subtrees = list(), ...)
```

```
empty_tree()
```

```
is_empty(x)
```
## Default S3 method:
is_empty(x)

## S3 method for class 'rtree'
is_empty(x)

### Arguments

- **label**: character. The label of the root of the tree created. If label=NULL, the tree created is unrooted. If label=NULL and subtrees=list(), the tree created is the empty tree.
- **subtrees**: A (possibly empty) list of rtree objects. These rtrees must be rooted, otherwise an error is thrown.
- **...**: Additional arguments to be passed as attributes to each node of the tree.
- **x**: A tree.

### Value

An rtree object.

### Examples

#### Chains
```
(tr0 = c_("Bob", "Carl", "Daniel"))
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
```

#### Rooted tree
```
(tr2 = r_("Alice", s = list(tr0, tr1)))
```

#### Unrooted tree
```
(tr3 = r_(s = list(tr2, c_("Grand-Mother", "Father", "Son"))))
```

---

### Description

The function siblings returns the siblings of a node in a given tree.

### Usage

```
siblings(.node, .tree, include_node = FALSE)
```

## S3 method for class 'rtree'
siblings(.node, .tree, include_node = FALSE)
subtrees

Arguments

.node node or character. The node or label of the node considered.
.tree A tree.
.include_node logical. If FALSE (the default), the node .node is not included to the list of siblings.

Value

A (possibly empty) list of nodes.

Examples

## Rooted tree
(tr0 = c_("Bob", "Carl", "Daniel"))
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
(tr2 = r_("Alice", s = list(tr0, tr1)))
siblings("Bob", tr2)

## Unrooted tree
(tr3 = r_(s = list(tr2, c_("Grand-Mother", "Father", "Son"))))
siblings("Alice", tr3) # note that in 'tr3', Alice and Grand-Mother are not siblings

---

subtrees Subtrees of a tree

Description

The function `subtrees` returns the list of subtrees of the root of a given tree.

Usage

```r
subtrees(.tree)
```

### S3 method for class 'rtree'
```r
subtrees(.tree)
```

```r
subtrees(x) <- value
```

Arguments

.tree, x A tree.
.value Subtrees to be assigned to x.

Value

A (possibly empty) list of trees.
### Examples

#### Rooted tree

```r
tr0 = c_("Bob", "Carl", "Daniel")
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
(tr2 = r_("Alice", s = list(tr0, tr1)))
(subtrees(tr2))
```

#### Unrooted tree

```r
tr3 = r_(s = list(tr2, c_("Son", "Father", "Grand-Mother"))))
(subtrees(tr3))
```

---

**take_branch**

*Take a branch of a tree*

#### Description

The function `take_branch` looks for the subtree of `.tree` whose root is the node identified by `.node`.

#### Usage

```r
take_branch(.tree, .node)
```

#### Arguments

- `.tree` A tree.
- `.node` A node of `.tree`.

#### Value

A tree.
Tree

Tree Class

Description

Tree class
Node class; inherits from the classNodes.

Usage

Tree
Node

Format

An object of class R6ClassGenerator of length 24.

Fields

data  object to be converted to an rtree.
label  character. The label of the node; must be part of the tree’s labels.
tree  rtree. The tree where the node belongs to.
...  Additional parameters (currently not used).

Methods

get_tree()  Get the tree (as an rtree object)
ancestors_of(.node, include_node = FALSE)  Get the ancestors of the node
children_of(.node, degree = 1L)  Get the children of the node
descendants_of(.node, include_node = FALSE)  Get the descendants of the node
parent_of(.node, degree = 1L)  Get the parent of the node
siblings_of (.node, include_node = FALSE)  Get the siblings of the node

get_label()  Get the label of the node
ancestors(include_node = FALSE)  Get the ancestors of the node
children(degree = 1L)  Get the children of the node
descendants(include_node = FALSE)  Get the descendants of the node
parent(degree = 1L)  Get the parent of the node
siblings(include_node = FALSE)  Get the siblings of the node

See Also

Nodes and Node in this package.
Tree and Nodes in this package.
tree_apply

Apply a function to each node of a tree

Description

.tree_apply applies a function fun to each node of .tree and stores the results in the attribute at.

Usage

.tree_apply(.tree, ...)

### S3 method for class 'rtree'
.tree_apply(.tree, at, fun, ...)

Arguments

.tree A tree.

... Additional arguments to be passed to fun.

.at character. Name of the attribute to be created at each node of .tree that will contain the result of fun.

.fun function or character. A function taking two arguments .node and .tree (in this order), to be applied to each node of the tree.

Examples

### Rooted tree
(tr0 = c_("Bob", "Carl", "Daniel"))
(tr1 = c_("Bill", "Caroline", "Dimitri", "Enoc"))
(tr2 = r_("Alice", s = list(tr0, tr1)))

### Unrooted tree
(tr3 = r_(s = list(tr2, c_("Grand-Mother", "Father", "Son"))))

f = function(.node, .tree) nchar(label(.node))
tr4 = tree_apply(tr3, at = "value", fun = f)
print(tr4, at = "value")

g = function(.node, .tree) height(take_branch(.tree, .node))
tr5 = tree_apply(tr3, at = "height", fun = g)
print(tr5, at = "height")
Description

Update a tree with new subtrees

Usage

## S3 method for class 'rtree'
update(object, subtrees = NULL, ...)

Arguments

object A tree to be updated.
subtrees A list of trees.
... Additional arguments to be passed as attributes to each node of the tree.

%->%

Add a tree at the bottom of a chain

Description

The function %->% links the rooted tree e2 to the bottom of the chain e1.

Usage

e1 %->% e2

Arguments

e1 A chain.
e2 A rooted tree.

Value

A rooted tree.
Examples

## Chain
(tr0 = c_(Bob, Carl, Daniel))

## Rooted tree
(tr1 = c_(Bill, Caroline, Dimitri, Enoc))
(tr2 = c_(John, Thomas))
(tr3 = r_(Alice, s = list(tr1, tr2)))

## Linking both
tr0 \%->\% tr3
Index

*Topic **datasets**
  - Nodes, 15
  - Tree, 23
  - %->%, 25

ancestors, 2, 23
as.list.rtree, 3
as.node, 4
as.rtree, 5

c_(chain), 6
chain, 6
children, 7, 23
cut_leaves, 7
descendants, 8, 23
descendants<-(height), 9
empty_tree(rtree), 19
flatten, 8
has.parent(parent), 16
height, 9
height<-(height), 9

is.binary_tree, 10
is.chain, 11
is.node, 12
is.root, 12
is.rooted, 13
is.rtree(is.tree), 13
is.tree, 13
is_empty(rtree), 19
is_leafnode(leaves), 14

label, 14
label<-(label), 14
labels.rtree(label), 14
leaves, 14

Node, 16, 23

Node (Tree), 23
Nodes, 15, 23

parent, 16, 23
print.rtree, 17
prune, 9, 18
r_(rtree), 19
rev.rtree, 18
root, 19
roots(root), 19
rtree, 19, 23
siblings, 20, 23
subtrees, 21
subtrees<-(subtrees), 21
take_branch, 22
take_node(take_branch), 22
Tree, 16, 23, 23
tree_apply, 17, 24
update.rtree, 25