Package 'GeneralTree'

September 8, 2016

Type Package

Title General Tree Data Structure

Version 0.0.1

Description A general tree data structure implementation in R.

Depends R (>= 3.2.4), R6 (>= 2.1.2), utils (>= 2.3.0)

License Apache License (== 2.0)

Encoding UTF-8

LazyData true

Suggests testthat, covr, knitr, DiagrammeR, iterators, foreach, tools, rmarkdown, microbenchmark, doParallel

RoxygenNote 5.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Anton Bossenbroek [aut, cre]

Maintainer Anton Bossenbroek <anton.bossenbroek@me.com>

Repository CRAN

Date/Publication 2016-09-08 20:57:16

R topics documented:

`<-.GeneralTree` *Deep clone a General Tree.*

Description

Deep clone a General Tree.

Usage

```
`<-.GeneralTree`(x, value)
```

Arguments

`x`	The target to where the tree should be copied.
`value`	The general tree that should be cloned into.

Value

a clone of the tree.

`=.GeneralTree` *Deep clone a General Tree.*

Description

Deep clone a General Tree.

Usage

```
`=.GeneralTree`(x, value)
```

Arguments

`x`	The target to where the tree should be copied.
`value`	The general tree that should be cloned into.

Value

a clone of the tree.

as.data.frame.GeneralTree *Convert a GeneralTree to a data frame.*

Description

Convert a GeneralTree to a data frame.

Usage

```
## S3 method for class 'GeneralTree'
as.data.frame(x, row.names = NULL, optional = NULL,
  ...)
```

Arguments

x	GeneralTree to convert to a data frame.
row.names	Ignored.
optional	Ignored.
...	Ignored.

as.GeneralTree *Convert an object to a GeneralTree.*

Description

Convert an object to a GeneralTree.

Usage

```
as.GeneralTree(x, ...)
```

Arguments

x	The object that should be converted.
...	passed to underlying functions.

as.GeneralTree.data.frame

Convert a data frame to a GeneralTree.

Description

Convert a data frame to a GeneralTree.

Usage

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

Arguments

x	The data frame that should be converted to a tree.
...	id The column name of the column that holds the ids of each node. data The column name of the column that holds the data of each node. parent The column name of the column that holds the parent of each node, NA indicates a node is the root.

Examples

```
  test_tree_df <- data.frame(
      ID = c("root", "child1", "child2", "child3"),
      DATA = c("parent1", "data3.1", "data1.2", "data1.3"),
     PARENT = c(NA, "child3", "root", "root"), stringsAsFactors = FALSE)
as.GeneralTree(test_tree_df, id = "ID", data = "DATA", parent = "PARENT")
```

as.GeneralTree.expression

Convert a R parsed expression to a GeneralTree.

Description

Convert a R parsed expression to a GeneralTree.

Usage

```
## S3 method for class 'expression'
as.GeneralTree(x, ...)
```

Arguments

x	The expression that should be converted.
...	what = "token" fill the tree with tokens as the data field. what = "text" fill the tree with text as the data field.

Examples

```
p <- parse(text = "
                    tree <- GeneralTree$new(1, 'parent1')
                    tree$addNode(1, 2, 'child.1.2')
                    tree$addNode(2, 3, 'child.2.3')",
           keep.source = TRUE)
as.GeneralTree(p, what = "token")
as.GeneralTree(p, what = "text")
as.GeneralTree(p, what = c("text", "token"))
```

---

deep_clone *Deep clone a General tree*

Description

Deep clone a General tree

Usage

```
deep_clone(x)
```

Arguments

x	The object that should be converted.
...	passed to underlying functions.

---

deep_clone.GeneralTree *Deep clone a General Tree.*

Description

Deep clone a General Tree.

Usage

```
## S3 method for class 'GeneralTree'
deep_clone(x)
```

Arguments

x	The general tree that should be deep cloned.

Value

a clone of the tree.

---

deleteId *Delete a node with a given id.*

Description

Delete a node with a given id.

Usage

```
deleteId(self, id)
```

Arguments

self	The reference to the tree where the id should be searched.
id	The id that should be deleted.

GeneralTree *A tree that can have multiple childeren per parent.*

Description

This class allows to create a tree with multiple childs per node. The data as well as the id are left totally to the choice of the user and can even be different.

Usage

```
GeneralTree
```

Format

R6Class object.

Value

Object of R6Class with methods for creating a general tree.

Methods

`addNode(parent_id, id, data)` Add a new node to the tree. The new node will be a child of parent_id and have an id and data.

`searchData(id)` Search an node in the tree that has an id equal to `id`. This method returns the data associated with the node.

`searchNode(id)` Search an node in the tree that has an id equal to `id`. This method returns the node.

`searchBranch(id)` Search for a node in a particular branch of the tree. The function returns a node.

`getSiblingNodes()` Get all the siblings of this node in a list. The results will not include the node itself.

`getSiblingId()` Get all the sibling ids in a list. The results will not include the node itself.

`getSiblingData()` Get all the sibling data in a list. The results will not include the node itself.

`getChildNodes(recursive = FALSE)` Get the child nodes from the current branch. On default the function will only return one level deep. If `recursive` is set to `TRUE`, also childs in nested branches will be returned. The childs will all be returned in a list.

`getChildId(recursive = FALSE)` Get the ids from all the child nodes. If `recursive` is set to `TRUE`, also ids from childs in nested branches will be returned. The ids will all be returned in a list.

`getChildData(recursive = FALSE)` Get the data from all the child nodes. If `recursive` is set to `TRUE`, also data from childs in nested branches will be returned. The data will all be returned in a list.

`deleteId(id)` Delete a node with id equal to `id`. All child nodes will also be deleted.

`delete()` Delete the current node and all childs. Should not be called directly.

`iterator()` Get an iterator to iterate through the tree in a depth first search.

`nextElem()` Get the next element in a depth first search. Before using this function always create an iterator.

`toString(what = c("id", "data"), string_prepend = "")` Creates a string representation of the node. Note that id and data should work with paste to work correctly. All branches will also be returned to the string.

Active methods

`root` Returns the root of a node.

`left_child` Returns the left child of a node.

`siblings` Returns the left sibling of a node.

`id` Returns the id of a node.

`have_child` Returns `TRUE` if the node has childs and `FALSE` otherwise.

`have_siblings` Returns `TRUE` if the node has siblings and `FALSE` otherwise.

`is_last_sibling` Returns `TRUE` if the node is the last siblings and `FALSE` otherwise.

`have_private_siblings` Returns `TRUE` if the node has a private field siblings set and `FALSE` otherwise.

have_parent Returns TRUE if the node has a parent field set and FALSE otherwise.

data Returns the data of the node.

id Returns the id of the node.

is_root Returns TRUE if the node is the root and FALSE otherwise.

parent Return the parent of the node.

treeDepth Returns the depth of the tree.

branch_depth Returns the depth of the branch.

isSingletonTree Returns TRUE if the tree contains only a single element and FALSE otherwise.

Examples

```
# Create a tree
tree <- GeneralTree$new(0, "root")
tree$addNode(0, 1, "child.0.1")
tree$addNode(0, 2, "child.0.2")
tree$addNode(0, 3, "child.0.3")
tree$addNode(3, 4, "child.3.4")
tree$searchData(4)

#
# Print the tree
tree

#
# Example how to iterate through the tree in a depth first iteration.
i <- tree$iterator()
while (!is.null(i)) {
   i$setData(paste("id:", i$id, " : data", i$data))
   i <- tryCatch(i$nextElem(), error = function(e) NULL)
}

# An example with the foreach package.
require(iterators)
require(foreach)
itx <- iter(tree, by = "id")
numbers_in_tree <- foreach(i = itx, .combine = c) %do% c(i)

itx <- iter(tree, by = "data")
data_in_tree <- foreach(i = itx, .combine = c) %do% c(i)
```

---

generate_grViz *Create a DiagrammeR graph that represents the tree.*

Description

Create a DiagrammeR graph that represents the tree.

Usage

```
generate_grViz(obj, what = c("id", "data"), ...)
```

Arguments

obj	the tree that should be converted to a DiagrammeR graph.
what	select what to draw in the tree.
...	Additional arguments passed to create_nodes

See Also

create_nodes

---

getChildData *Get the data of the child nodes below the current node.*

Description

Get the data of the child nodes below the current node.

Usage

```
getChildData(self, recursive = FALSE)
```

Arguments

self	The node where to start.
recursive	Should the function be called on all child nodes too?

Value

the data associated with child nodes.

`getChildId` *Get the ids of the child nodes below the current node.*

Description

Get the ids of the child nodes below the current node.

Usage

```
getChildId(self, recursive = FALSE)
```

Arguments

`self`	The node where to start.
`recursive`	Should the function be called on all child nodes too?

Value

the ids associated with child nodes.

`getChildNodes` *Get all the child nodes below the current node.*

Description

Get all the child nodes below the current node.

Usage

```
getChildNodes(self, recursive = FALSE)
```

Arguments

`self`	The node where to start
`recursive`	Should the function be called on all child nodes too?

Value

a list of child nodes.

plot.GeneralTree *Plot a GeneralTree object.*

Description

Plot a GeneralTree object.

Usage

```
## S3 method for class 'GeneralTree'
plot(x, ...)
```

Arguments

x	tree to plot.
...	arguments passed to underlying functions.

print.GeneralTree *Print a GeneralTree object.*

Description

Print a GeneralTree object.

Usage

```
## S3 method for class 'GeneralTree'
print(x, ...)
```

Arguments

x	tree to print.
...	arguments passed to underlying functions.

`searchData` *Search for an id in starting at a point in the tree and return the data matching the id.*

Description

Search for an id in starting at a point in the tree and return the data matching the id.

Usage

```
searchData(self, id)
```

Arguments

`self`	the node where to start searching.
`id`	the id to look for.

Value

The data associated with an id.

`searchNode` *Search for an id in starting at a point in the tree and return the node matching the id.*

Description

Search for an id in starting at a point in the tree and return the node matching the id.

Usage

```
searchNode(self, id)
```

Arguments

`self`	the node where to start searching.
`id`	the id to look for.

Value

The data associated with an id.

Index