Package ‘orbweaver’

November 24, 2023

Title Fast and Efficient Graph Data Structures

Version 0.0.3

Description Empower your data analysis with 'orbweaver', an R package designed for effortless construction and analysis of graph data structures. With 'orbweaver', you can seamlessly build and manipulate graph structures, leveraging its high-performance methods for filtering, joining, and mutating data within the R environment. Drawing inspiration from the efficiency of the 'data.table' package, 'orbweaver' ensures that mutations and changes to the graph are performed in place, streamlining your workflow for optimal productivity.

URL https://github.com/ixpantia/orbweaver

BugReports https://github.com/ixpantia/orbweaver/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Config/rextendr/version 0.3.1

SystemRequirements Cargo (Rust's package manager), rustc

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author ixpantia, SRL [cph],
Andres Quintero [aut, cre],
The authors of the dependency Rust crates [ctb] (see inst/AUTHORS file for details)

Maintainer Andres Quintero <andres@ixpantia.com>

Repository CRAN

Date/Publication 2023-11-24 17:00:02 UTC

R topics documented:

`add_child` *Add Child*

Description

Adds a child to a node. If the child already exists, nothing happens.

If the parent or the child do not exist, they are created.

Note: This function modifies the graph in place. It does not return a new graph. This is done for performance and memory reasons. It works in a similar way to the `data.table` package.

Usage

```
add_child(graph, parent, child)
```

Arguments

`graph`	The graph to add the child to.
`parent`	The ID of the parent node.
`child`	The ID of the child node.

Value

A reference to the graph passed in.

`add_node` *Add Node*

Description

Adds a node to the graph. If the node already exists, nothing happens.

Note: This function modifies the graph in place. It does not return a new graph. This is done for performance and memory reasons. It works in a similar way to the `data.table` package.

Usage

```
add_node(graph, node)
```

Arguments

`graph`	The graph to add the node to.
`node`	The ID of the node to add.

Value

A reference to the graph passed in.

`as.list.AcyclicGraph` *Convert to List*

Description

Converts the graph to a list.

Usage

```
## S3 method for class 'AcyclicGraph'
as.list(x, ...)
```

Arguments

`x`	The graph to convert to a list.
`...`	Ignored.

Value

A list representation of the graph.

`as_graph` *As Graph*

Description

Attempts to convert the object to a graph.

Usage

```
as_graph(x, type, ...)
```

Arguments

`x`	The object to convert to a graph.
`type`	The type of graph to convert to. Currently only `acyclic` is supported.
`...`	Additional arguments passed to the method.

Value

A graph of the given type.

`as_graph.data.frame` *Data.frame as Graph*

Description

Converts a data.frame to a graph.

Usage

```
## S3 method for class 'data.frame'
as_graph(x, type, ...)
```

Arguments

`x`	The data.frame to convert to a graph.
`type`	The type of graph to convert to. Currently only `acyclic` is supported.
`...`	Ignored.

Value

A graph of the given type.

`clone_graph` *Clone Graph*

Description

Creates a copy of the graph.

Usage

```
clone_graph(graph)
```

Arguments

`graph`	The graph to clone.

Value

A new graph that is a copy of the original.

`find_all_paths` *Find all paths between two nodes*

Description

Finds all paths between two nodes.

Usage

```
find_all_paths(graph, from, to)
```

Arguments

`graph`	The graph to search in.
`from`	The ID of the node to start the search from.
`to`	The ID of the node to end the search at.

Value

A list of character vectors of the paths between the two nodes.

`find_least_common_parents` *Get Least Common Parents from an Acyclic Graph*

Description

Gets the least common parents of a set of nodes. This is the set of parents that are parents of all the nodes and that have been selected.

This is useful for example if you want to group by the set of parents of a set of nodes.

Usage

```
find_least_common_parents(graph, nodes)
```

Arguments

`graph`	The graph to get the least common parents from.
`nodes`	The nodes to get the least common parents of.

Value

A character vector of the least common parents of the nodes.

`find_leaves` *Get Leaves / Maximum Depth*

Description

Gets the leaves of the graph that descend from a node.

Usage

```
find_leaves(graph, node)
```

Arguments

`graph`	The graph to get the leaves from.
`node`	The ID of the node to get the leaves of.

Value

A character vector of the leaves of the node.

`find_roots` *Find Roots*

Description

Gets the roots of the graph.

Usage

```
find_roots(graph)
```

Arguments

`graph`	The graph to get the roots from.

Value

A character vector of the roots of the graph.

`get_children` *Get Children*

Description

Gets the children of a node.

Usage

```
get_children(graph, node)
```

Arguments

`graph`	The graph to get the children from.
`node`	The ID of the node to get the children of.

Value

A character vector of the children of the node.

`get_parents` *Get Parents*

Description

Gets the parents of a node.

Usage

```
get_parents(graph, node)
```

Arguments

`graph`	The graph to get the parents from.
`node`	The ID of the node to get the parents of.

Value

A character vector of the parents of the node.

`new_graph` *Initialize a New Graph*

Description

Initializes a new graph with the given type.

Usage

```
new_graph(type)
```

Arguments

`type`	The type of graph to create. Currently only `acyclic` is supported.

Value

A new graph of the given type.

`search_for_node` *Search for a node inside a graph by name*

Description

Searches for nodes with a name that matches the given string.

Usage

```
search_for_node(graph, node_id, case_sensitive = TRUE)
```

Arguments

`graph`	The graph to search in.
`node_id`	The string to search for.
`case_sensitive`	Whether the search should be case sensitive.

Value

A character vector of the nodes that match the search.

Index