Package ‘simplegraph’

December 10, 2015

Title Simple Graph Data Types and Basic Algorithms
Version 1.0.0
Author Gabor Csardi
Maintainer Gabor Csardi <gcsardi@mango-solutions.com>
Description Simple classic graph algorithms for simple graph classes.
Graphs may possess vertex and edge attributes. ‘simplegraph’ has no dependencies and it is written entirely in R, so it is easy to install.
License MIT + file LICENSE
LazyData true
URL https://github.com/mangothecat/simplegraph
BugReports https://github.com/mangothecat/simplegraph/issues
Suggests testthat
Imports utils
RoxygenNote 5.0.1
NeedsCompilation no
Repository CRAN
Date/Publication 2015-12-10 23:15:15

R topics documented:

adjacent_vertices ........................................ 2
bfs ........................................ 3
degree ........................................ 4
edges ........................................ 4
graph ........................................ 5
incident_edges ........................................ 7
is_loopy ........................................ 8
is_multigraph ........................................ 8
is_simple ........................................ 9
adjacent_vertices

is_weighted ................................................................. 10
order ................................................................. 11
predecessors ............................................................... 11
remove_loops .............................................................. 12
remove_multiple ............................................................ 13
sanitize .............................................................. 13
simplegraph ............................................................. 14
simplify ............................................................. 14
size ................................................................. 15
strength .............................................................. 15
topological_sort ............................................................ 16
transpose ............................................................. 17
vertex_ids ............................................................. 18
vertices ............................................................. 19

Index 21

adjacent_vertices

Adjacent vertices for all vertices in a graph

Description

A vertex is adjacent is it is either a successor, or a predecessor.

Usage

adjacent_vertices(graph)

Arguments

graph The graph.

Value

A named list of character vectors, the adjacent vertices for each vertex.

See Also

Other simple queries: edges, order, vertices

Examples

G <- graph(list(A = c("B", "C"), B = "C", C = "A"))
adjacent_vertices(G)
Breadth-first search of a graph

Usage

bfs(graph, from = vertex_ids(graph))

Arguments

graph Input graph.
from Character vector, which vertices to start the search from. By default all vertices are attempted.

Value

Character vector of the named of the visited vertices, in the order of their visit.

Examples

funcs <- graph(list(
  drop_internal = character(0),
  get_deps = c("get_description", "parse_deps", "%|%", "drop_internal"),
  get_description = "pkg_from_filename",
  parse_deps = "str_trim",
  cran_file = c("get_pkg_type", "r_minor_version", "cran_file"),
  download_urls = c("split_pkg_names_versions", "cran_file"),
  filename_from_url = character(0),
  get_pkg_type = character(0),
  pkg_download = c("dir_exists", "download_urls", "filename_from_url", "try_download"),
  r_minor_version = character(0),
  try_download = character(0),
  drop_missing_deps = character(0),
  install_order = character(0),
  restore = c("pkg_download", "drop_missing_deps", "install_order", "get_deps"),
  snap = character(0),
  `%|%` = character(0),
  data_frame = character(0),
  dir_exists = character(0),
  pkg_from_filename = character(0),
  split_pkg_names_versions = "data_frame",
  str_trim = character(0)
))
bfs(funcs)
### degree

**Degree of vertices**

**Description**

Degree of vertices

**Usage**

\[
\text{degree(graph, mode = c("out", "in", "total", "all"))}
\]

**Arguments**

- **graph**: Input graph.
- **mode**: Whether to calculate out-degree, in-degree, or the total degree.

**Value**

Named numeric vector of degrees.

**Examples**

```r
G <- graph(list(A = c("B", "C"), B = "C", C = "A"))
degree(G, mode = "out")
degree(G, mode = "in")
degree(G, mode = "total")
```

### edges

**Edges of a graph**

**Description**

Edges of a graph

**Usage**

\[
\text{edges(graph)}
\]

**Arguments**

- **graph**: The graph

**Value**

Data frame of edge data and metadata. The tail and head vertices are in the first two columns. The rest of the columns are metadata.
### Description

Graphs can be specified as adjacency lists or (two) data frames.

### Usage

```r
graph(x, ...)```

### Arguments

- `x`  
  A data frame, or a named list of character vectors. See details below.

- `...`  
  Additional arguments, see details below.
Details

If the first argument is a data frame, then it is interpreted as vertex data, and a second data frame
must be supplied as edge data. The first column of the vertex data must contain (character) vertex
ids. The first two columns of the edge data frame must contain the directed edges of the graph,
in the order of tail and head, as characters referring to the nodes ids. Other columns are kept as
metadata.

If the first argument is not a data frame, but a list, then it is interpreted as an adjacency list. It must
be named, and the names will be used as vertex ids. Each list element must be a character vector
containing the successors of each vertex.

Value

A graph object.

Examples

```r
defs <- graph(list(
  drop_internal = character(0),
  get_deps = c("get_description", "parse_deps",
    "%|%", "drop_internal"),
  get_description = "pkg_from_filename",
  parse_deps = "str_trim",
  cran_file = c("get_pkg_type", "r_minor_version", "cran_file"),
  download_urls = c("split_pkg_names_versions", "cran_file"),
  filename_from_url = character(0),
  get_pkg_type = character(0),
  pkg_download = c("dir_exists", "download_urls",
    "filename_from_url", "try_download"),
  r_minor_version = character(0),
  try_download = character(0),
  drop_missing_deps = character(0),
  install_order = character(0),
  restore = c("pkg_download", "drop_missing_deps",
    "install_order", "get_deps"),
  snap = character(0),
  "%|%" = character(0),
  data_frame = character(0),
  dir_exists = character(0),
  pkg_from_filename = character(0),
  split_pkg_names_versions = "data_frame",
  str_trim = character(0)
))
defs

vertices <- data.frame(
  stringsAsFactors = FALSE,
  name = c("Tom Hanks", "Cate Blanchett", "Matt Damon", "Kate Winslet",
    "Saving Private Ryan", "Contagion", "The Talented Mr. Ripley"),
  what = c("actor", "actor", "actor", "actor", "movie", "movie", "movie"),
    NA, NA, NA),
  NA, NA, NA),
```
incident_edges


gender = c("M", "F", "M", "F", NA, NA, NA),
)
edges <- data.frame(
    stringsAsFactors = FALSE,
    actor = c("Tom Hanks", "Cate Blanchett", "Matt Damon", "Matt Damon",
              "Kate Winslet"),
    movie = c("Saving Private Ryan", "The Talented Mr. Ripley",
              "Saving Private Ryan", "The Talented Mr. Ripley", "Contagion")
)
actors <- graph(vertices, edges)
actors

incident_edges  Incident edges

Description
Incident edges

Usage
incident_edges(graph, mode = c("out", "in", "all", "total"))

Arguments

  graph Input graph.
  mode Whether to use out edges, in edges or all edges.

Value
A list of data frames, each a set of edges.

Examples

G <- graph(list(A = c("B", "C"), B = "C", C = "A"))
incident_edges(G, mode = "out")
incident_edges(G, mode = "in")
incident_edges(G, mode = "all")
is_loopy

Is this a loopy graph?

Description
A loopy graph has at least one loop edge: an edge from a vertex to itself.

Usage

is_loopy(graph)

Arguments

graph The input graph.

Value
Logical scalar.

See Also
Other multigraphs: is_multigraph, is_simple, remove_loops, remove_multiple, simplify

Examples

G <- graph(list(A = c("A", "B", "B"), B = c("A", "C"), C = "A"))
is_loopy(G)

G2 <- simplify(G)
is_loopy(G2)

is_multigraph

Is this a multigraph?

Description
A multigraph has at least one pair or multiple edges, edges connecting the same (ordered) pair of vertices.

Usage

is_multigraph(graph)

Arguments

graph Input graph.
is_simple

Value
Logical scalar.

See Also
Other multigraphs: is_loopy, is_simple, remove_loops, remove_multiple, simplify

Examples

```r
G <- graph(list(A = c("A", "B", "B"), B = c("A", "C"), C = "A"))
is_multigraph(G)

G2 <- simplify(G)
is_multigraph(G2)
```

Description
A simple graph contains no loop and multiple edges.

Usage

```r
is_simple(graph)
```

Arguments

graph The input graph.

Value
Logical scalar.

See Also
Other multigraphs: is_loopy, is_multigraph, remove_loops, remove_multiple, simplify

Examples

```r
G <- graph(list(A = c("A", "B", "B"), B = c("A", "C"), C = "A"))
is_simple(G)

G2 <- simplify(G)
is_simple(G2)
```
is_weighted | Is the graph weighted?

Description

Is the graph weighted?

Usage

is_weighted(graph)

Arguments

graph | The graph.

Examples

```r
G <- graph(
  data.frame(
    stringsAsFactors = FALSE,
    id = c("a", "b", "c", "d"),
  ),
  data.frame(
    stringsAsFactors = FALSE,
    from = c("a", "a", "b", "b", "c"),
    to = c("b", "d", "d", "c", "a"),
    weight = c(1, 2, 1, 3, 2)
  )
)

is_weighted(G)

G2 <- graph(
  data.frame(
    stringsAsFactors = FALSE,
    id = c("a", "b", "c", "d"),
  ),
  data.frame(
    stringsAsFactors = FALSE,
    from = c("a", "a", "b", "b", "c"),
    to = c("b", "d", "d", "c", "a")
  )
)

is_weighted(G2)
```
**order**

*Order of a graph*

**Description**

The order of the graph is the number of vertices.

**Usage**

\[ \text{order}(\text{graph}) \]

**Arguments**

- **graph**: The graph.

**Value**

Numeric scalar, the number of vertices.

**See Also**

Other simple queries: `adjacent_vertices`, `edges`, `vertices`

**Examples**

\[
\begin{align*}
G & \leftarrow \text{graph}(\text{list}(A = c(\text{"B"}, \text{"C"}), B = \text{"C"}, C = \text{"A"}))) \\
\text{order}(G)
\end{align*}
\]

---

**predecessors**

*Predecessors and successors*

**Description**

Predecessors and successors

**Usage**

- **predecessors**(graph)
- **successors**(graph)

**Arguments**

- **graph**: Input graph
Value

Named list of character vectors, the predecessors or the successors of each vertex.

Examples

```r
G <- graph(list(A = c("B", "C"), B = "C", C = "A"))
predecessors(G)
successors(G)
```

---

**remove_loops**  
Remove loop edges from a graph

Description

Remove loop edges from a graph

Usage

```r
remove_loops(graph)
```

Arguments

- **graph**  
  Input graph

Value

Graph, with loop edges removed.

See Also

Other multigraphs: `is_loopy`, `is_multigraph`, `is_simple`, `remove_multiple`, `simplify`

Examples

```r
G <- graph(list(A = c("A", "B", "B"), B = c("A", "C"), C = "A"))
is_loopy(G)
is_loopy(remove_loops(G))
```
remove_multiple

Remove multiple edges from a graph

Description
Remove multiple edges from a graph

Usage
remove_multiple(graph)

Arguments
graph Input graph.

Value
Graph, without the multiple edges. (More precisely, from each set of multiple edges, only one, the first one, is kept.)

See Also
Other multigraphs: is_loopy, is_multigraph, is_simple, remove_loops, simplify

Examples
G <- graph(list(A = c("A", "B", "B"), B = c("A", "C"), C = "A"))
is_multigraph(G)
is_multigraph(remove_multiple(G))

sanitize
Check the validity of a graph data structure

Description
This is mainly for internal checks, but occasionally it might also useful externally.

Usage
sanitize(x, ...)

Arguments
x Graph.
... Extra arguments are curently ignored.
**simplify**

**Examples**

```r
G <- graph(list(A = c("B", "C"), B = "C", C = "A"))
sanitize(G)

G <- c(G, list("this is not good" = c(1, 2, 3)))
try(sanitize(G))
```

---

**simplegraph**

*Simple Graph Data Types and Basic Algorithms*

**Description**

Simple classic graph algorithms for simple graph classes. Graphs may possess vertex and edge attributes. 'simplegraph' has no dependencies and it is writing entirely in R, so it is easy to install.

**simplify**

*Remove multiple and loop edges from a graph*

**Description**

Remove multiple and loop edges from a graph

**Usage**

```r
simplify(graph)
```

**Arguments**

- `graph`: Input graph.

**Value**

Another graph, with the multiple and loop edges removed.

**See Also**

Other multigraphs: `is_loopy, is_multigraph, is_simple, remove_loops, remove_multiple`

**Examples**

```r
G <- graph(list(A = c("A", "B", "B"), B = c("A", "C"), C = "A"))
is_simple(G)

G2 <- simplify(G)
is_simple(G2)
```
size

The size of the graph is the number of edges

Description

The size of the graph is the number of edges

Usage

size(graph)

Arguments

graph The graph.

Value

Numeric scalar, the number of edges.

Examples

G <- graph(list(A = c("B", "C"), B = "C", C = "A"))
size(G)

strength

Vertex strength: sum of weights of incident edges

Description

This is also called weighed degree.

Usage

strength(graph, mode = c("out", "in", "total", "all"))

Arguments

graph Input graph.
mode Whether to consider incoming (in), outgoing (out) or all (total) edges.

Details

For non-weighted graphs, the degree is returned as a fallback.

Value

Named numeric vector.
Examples

```r
G <- graph(
  data.frame(
    stringsAsFactors = FALSE,
    id = c("a", "b", "c", "d")
  ),
  data.frame(
    stringsAsFactors = FALSE,
    from = c("a", "a", "b", "b", "c"),
    to = c("b", "d", "d", "c", "a"),
    weight = c(1, 2, 1, 3, 2)
  )
)

strength(G)

G2 <- graph(
  data.frame(
    stringsAsFactors = FALSE,
    id = c("a", "b", "c", "d")
  ),
  data.frame(
    stringsAsFactors = FALSE,
    from = c("a", "a", "b", "b", "c"),
    to = c("b", "d", "d", "c", "a")
  )
)

strength(G2)
```

---

topological_sort  

Topological sorting of a graph

Description

Topological sorting of a graph

Usage

`topological_sort(graph)`

Arguments

graph  

Input graph.

Value

Character vector of vertex ids, in topological order.
Examples

```r
funcs <- graph(list(
  drop_internal = character(0),
  get_deps = c("get_description", "parse_deps",
              "%|\%", "drop_internal"),
  get_description = "pkg_from_filename",
  parse_deps = "str_trim",
  cran_file = c("get_pkg_type", "r_minor_version", "cran_file"),
  download_urls = c("split_pkg_names_versions", "cran_file"),
  filename_from_url = character(0),
  get_pkg_type = character(0),
  pkg_download = c("dir_exists", "download_urls",
                  "filename_from_url", "try_download"),
  r_minor_version = character(0),
  try_download = character(0),
  drop_missing_deps = character(0),
  install_order = character(0),
  restore = c("pkg_download", "drop_missing_deps",
              "install_order", "get_deps"),
  snap = character(0),
  `\%|\%` = character(0),
  data_frame = character(0),
  dir_exists = character(0),
  pkg_from_filename = character(0),
  split_pkg_names_versions = "data_frame",
  str_trim = character(0)
))

topological_sort(remove_loops(funcs))
```

**Description**

The transposed graph have the same vertices, and the same number of edges, but all edge directions are opposite compared to the original graph.

**Usage**

```r
transpose(graph)
```

**Arguments**

- `graph` Input graph

**Value**

Transposed graph.
Examples

```r
funcs <- graph(list(
  drop_internal = character(0),
  get_deps = c("get_description", "parse_deps",
               "%%", "drop_internal"),
  get_description = "pkg_from_filename",
  parse_deps = "str_trim",
  cran_file = c("get_pkg_type", "r_minor_version", "cran_file"),
  download_urls = c("split_pkg_names_versions", "cran_file"),
  filename_from_url = character(0),
  get_pkg_type = character(0),
  pkg_download = c("dir_exists", "download_urls",
               "filename_from_url", "try_download"),
  r_minor_version = character(0),
  try_download = character(0),
  drop_missing_deps = character(0),
  install_order = character(0),
  restore = c("pkg_download", "drop_missing_deps",
               "install_order", "get_deps"),
  snap = character(0),
  `%%` = character(0),
  data_frame = character(0),
  dir_exists = character(0),
  pkg_from_filename = character(0),
  split_pkg_names_versions = "data_frame",
  str_trim = character(0)
 ))
edges(transpose(funcs))
```

---

<table>
<thead>
<tr>
<th>vertex_ids</th>
<th>Vertex ids of a graph</th>
</tr>
</thead>
</table>

Description

Vertex ids of a graph

Usage

`vertex_ids(graph)`

Arguments

- `graph` The graph.

Value

Character vector of vertex ids.
Examples

```r
G <- graph(list(A = c("B", "C"), B = "C", C = "A"))
vertex_ids(G)
```

---

<table>
<thead>
<tr>
<th>vertices</th>
<th>Vertices of a graph, with metadata</th>
</tr>
</thead>
</table>

Description

Vertices of a graph, with metadata

Usage

```r
vertices(graph)
```

Arguments

- `graph` The graph.

Value

Character vector of vertex names.

See Also

Other simple queries: `adjacent_vertices`, `edges`, `order`

Examples

```r
bridges <- graph(list(
  "Altstadt-Loebenicht" = c(
    "Kneiphof",
    "Kneiphof",
    "Lomse"
  ),
  "Kneiphof" = c(
    "Altstadt-Loebenicht",
    "Altstadt-Loebenicht",
    "Vorstadt-Haberberg",
    "Vorstadt-Haberberg",
    "Lomse"
  ),
  "Vorstadt-Haberberg" = c(
    "Kneiphof",
    "Kneiphof",
    "Lomse"
  ),
  "Lomse" = c(
    "Altstadt-Loebenicht",
```
Index

adjacent_vertices, 2, 5, 11, 19
bfs, 3
degree, 4
edges, 2, 4, 11, 19
graph, 5
incident_edges, 7
is_loopy, 8, 9, 12–14
is_multigraph, 8, 8, 9, 12–14
is_simple, 8, 9, 9, 12–14
is_weighted, 10
order, 2, 5, 11, 19
predecessors, 11
remove_loops, 8, 9, 12, 13, 14
remove_multiple, 8, 9, 12, 13, 14
sanitize, 13
simplegraph, 14
simplegraph-package (simplegraph), 14
simplify, 8, 9, 12, 13, 14
size, 15
strength, 15
successors (predecessors), 11
topological_sort, 16
transpose, 17
vertex_ids, 18
vertices, 2, 5, 11, 19