Package 'adj'

May 7, 2026

Title Lightweight Adjacency Lists

Version 0.1.0

Description Provides an S3 class to represent graph adjacency lists using 'vctrs'. Allows for creation, subsetting, combining, and pretty printing of these lists. Adjacency lists can be easily converted to zero-indexed lists, which allows for easy passing of objects to low-level languages for processing.

Depends R (>= 3.5)

Imports rlang, cli, vctrs (>= 0.6.5)

Suggests methods, geos, Matrix, pillar, spelling, testthat (>= 3.0.0)

License MIT + file LICENSE

RoxygenNote 7.3.3

LazyData true

Language en-US

Encoding UTF-8

Config/build/compilation-database true

Config/testthat/edition 3

URL https://alarm-redist.org/adj/, https://github.com/alarm-redist/adj

NeedsCompilation yes

Author Christopher T. Kenny [aut, cre] (ORCID: <https://orcid.org/0000-0002-9386-6860>), Cory McCartan [aut] (ORCID: <https://orcid.org/0000-0002-6251-669X>)

Maintainer Christopher T. Kenny <ctkenny@proton.me>

Repository CRAN

Date/Publication 2026-02-20 10:30:15 UTC

Contents

`adj` *Create an adjacency list*

Description

Create an adjacency list from a list of vectors of adjacent node identifiers.

Usage

```
adj(
  ...,
  ids = NULL,
  duplicates = c("warn", "error", "allow", "remove"),
  self_loops = c("warn", "error", "allow", "remove")
)

as_adj(x)

is_adj(x)

adj_to_list(x, ids = NULL)
```

Arguments

`...`	Vectors or a single list of vectors. Vectors should be comprised either of (1-indexed) indices of adjacent nodes, or of unique identifiers, which must match to the provided `ids`. `NULL` can be used in place of a length-zero vector for nodes without neighbors.
`ids`	A vector of unique node identifiers. Each provided vector in `...` will be matched to these identifiers. If `NULL`, the identifiers are taken to be 1-indexed integers.
`duplicates`	Controls handling of duplicate neighbors. The value `"warn"` warns the user; `"error"` throws an error; `"allow"` allows duplicates, and `"remove"` removes duplicates silently and then sets the corresponding attribute to `"error"`.

self_loops — Controls handling of self-loops (nodes that are adjacent to themselves). The value "warn" warns the user; "error" throws an error; "allow" allows self-loops, and "remove" removes self-loops silently and then sets the corresponding attribute to "error".

x — An adj list

Details

Equality:

Equality for adj lists is evaluated elementwise. Two sets of neighbors are considered equal if they contain the same neighbors, regardless of order.

Number of nodes and edges:

The adj package is not focused on graph operations. The length() function will return the number of nodes. To compute the number of edges in an adjacency list a, use sum(lengths(a)), and divide by 2 for undirected graphs.

Value

An adj list

Examples

```
a1 = adj(list(c(2, 3), c(1, 3), c(1, 2)))
a2 = adj(list(c(3, 2), c(3, 1), c(2, 1)))
a1 == a2

adj(2:3, NULL, 4:5, integer(0), 1)
adj(1, 2, 3, self_loops = "remove")

adj(konigsberg$bridge_to, ids = konigsberg$area, duplicates = "allow")
adj(konigsberg$bridge_to, ids = konigsberg$area, duplicates = "remove")
```

---

adj_color *Find a coloring of an adjacency list*

Description

Greedily finds a coloring of an adjacency list, optionally grouped by a provided vector.

Usage

```
adj_color(x, groups = NULL, colors = 0, method = c("dsatur", "greedy"))
```

Arguments

x	An adj list
groups	An optional vector specifying the group membership for each node in x.
colors	Number of colors to use. If 0 (the default), uses as few colors as possible with this greedy algorithm.
method	Coloring method to use. "dsatur" uses the DSatur algorithm to try to minimize the number of colors. "greedy" traverses nodes in decreasing order of degree and may be appropriate when more colors are desired.

Value

An integer vector

References

Brélaz, Daniel (1979-04-01). "New methods to color the vertices of a graph". *Communications of the ACM*. 22 (4): 251–256.

Examples

```
a <- adj(konigsberg$bridge_to, ids = konigsberg$area, duplicates = "allow")
adj_color(a)
adj_color(a, colors = 3)
adj_color(a, groups = c("AD", "BC", "BC", "AD"))
```

---

adj_edges *Add and subtract edges from an adjacency list*

Description

Add and subtract edges from an adjacency list

Usage

```
adj_add_edges(x, v1, v2, ids = NULL)

adj_subtract_edges(x, v1, v2, ids = NULL)
```

Arguments

x	An adj list or object coercible to an adj list
v1	vector of vertex identifiers for the first vertex. Can be an integer index or a value to look up in ids, if that argument is provided. If more than one identifier is present, connects each to corresponding entry in v2.
v2	vector of vertex identifiers for the second vertex. Can be an integer index or a value to look up in ids, if that argument is provided. If more than one identifier is present, connects each to corresponding entry in v1.

ids — A vector of unique node identifiers. Each provided vector in v1 and v2 will be matched to these identifiers. If NULL, the identifiers are taken to be 1-indexed integers.

Value

An adj list

Examples

```
a <- adj(c(2, 3), 1, 1)
adj_add_edges(a, 2, 3)
adj_subtract_edges(a, 1, 2)
```

---

adj_from_shp *Create an* adj *list from a set of spatial polygons*

Description

Requires that the geos package be installed.

Usage

```
adj_from_shp(shp)
```

Arguments

shp — An object convertible to geos geometries representing polygons, such as an sf object, well-known text strings, or geos geometries.

Value

An adj list

Examples

```
shp <- c(
 "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
 "POLYGON ((0 1, 1 1, 1 2, 0 2, 0 1))",
 "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))",
 "POLYGON ((1 1, 2 1, 2 2, 1 2, 1 1))"
)

adj_from_shp(shp)
```

adj_indexing *Indexing operations on adjacency lists*

Description

adj overrides the default [and c() methods to allow for filtering, reordering, and concatenating adjacency lists while ensuring that indices remain internally consistent.

Usage

```
## S3 method for class 'adj'
x[i, ...]

## S3 method for class 'adj'
c(...)
```

Arguments

x	An adjacency list of class adj
i	Indexing vector
...	For c(), adjacency lists to concatenate. Ignored for [.

Details

When duplicate indices are present in the adjacency list, indexing is performed by slicing the adjacency matrix, which is slower and requires more memory. For large adjacency lists, slicing with duplicates will error for this reason; set options(adj.max_matrix_slice = Inf) to allow it, but be aware of the possible memory usage implications.

Value

A reindexed adjacency list for [, and a concatenated adjacency list for c().

Examples

```
a <- adj(c(2, 3), c(1, 3), c(1, 2))
a[1:2]
all(sample(a) == a) # any permutation yields the same graph

a <- adj(konigsberg$bridge_to, ids = konigsberg$area, duplicates = "remove")
c(a, a) # concatenates graphs with no connecting edges
```

adj_laplacian *Compute the Laplacian matrix of an adjacency list*

Description

The Laplacian matrix of a graph is defined as `L = D - A`, where `D` is the degree matrix (a diagonal matrix where `D[i, i]` is the degree of node `i`) and `A` is the adjacency matrix.

Usage

```
adj_laplacian(x, sparse = TRUE)
```

Arguments

`x`	An `adj` list
`sparse`	Whether to return a sparse matrix (of class `dgCMatrix`) or a dense matrix. Requires the `Matrix` package for sparse output.

Value

A matrix representing the Laplacian of the graph.

Examples

```
a <- adj(konigsberg$bridge_to, ids = konigsberg$area, duplicates = "allow")
L <- adj_laplacian(a, sparse = FALSE)
L

# count spanning trees (any minor of the Laplacian)
det(L[-1, -1])
```

adj_matrix *Convert adjacency lists to and from adjacency matrices*

Description

Adjacency lists can be converted to adjacency matrices and vice versa without loss.

Usage

```
adj_from_matrix(
  x,
  duplicates = c("warn", "error", "allow", "remove"),
  self_loops = c("warn", "error", "allow", "remove")
)

## S3 method for class 'adj'
as.matrix(x, sparse = FALSE, ...)
```

Arguments

x	An adjacency list or matrix
duplicates	Controls handling of duplicate neighbors. The value "warn" warns the user; "error" throws an error; "allow" allows duplicates, and "remove" removes duplicates silently and then sets the corresponding attribute to "error".
self_loops	Controls handling of self-loops (nodes that are adjacent to themselves). The value "warn" warns the user; "error" throws an error; "allow" allows self-loops, and "remove" removes self-loops silently and then sets the corresponding attribute to "error".
sparse	If TRUE, return a sparse matrix, which is often preferable for computation. See Matrix::sparseMatrix for details on this class.
...	Ignored.

Value

adj_from_matrix() returns an adj list; as.matrix() returns a matrix.

Examples

```
adj_from_matrix(1 - diag(3))

a = adj(konigsberg$bridge_to, ids = konigsberg$area, duplicates = "allow")
mat = as.matrix(a)
all(a == adj_from_matrix(mat, duplicates = "allow")) # TRUE
```

---

adj_quotient *Quotient an adjacency list by a vector*

Description

Computes the quotient graph of a given adjacency list by a provided grouping vector. Nodes in the same groups are merged into single nodes in the quotient graph. The resulting multi-edges and self-loops are handled according to the specified parameters.

Usage

```
adj_quotient(
  x,
  groups,
  duplicates = c("remove", "allow", "error", "warn"),
  self_loops = c("remove", "allow", "error", "warn")
)

adj_quotient_int(
  x,
  groups,
```

```
  n_groups,
  duplicates = c("remove", "allow", "error", "warn"),
  self_loops = c("remove", "allow", "error", "warn")
)
```

Arguments

x	An adj list
groups	A vector specifying the group membership for each node in x. adj_quotient() will process this vector with vctrs::vec_group_id(); adj_quotient_int() expects an (1-indexed) integer vector.
duplicates	Controls handling of duplicate neighbors. The value "warn" warns the user; "error" throws an error; "allow" allows duplicates, and "remove" removes duplicates silently and then sets the corresponding attribute to "error".
self_loops	Controls handling of self-loops (nodes that are adjacent to themselves). The value "warn" warns the user; "error" throws an error; "allow" allows self-loops, and "remove" removes self-loops silently and then sets the corresponding attribute to "error".
n_groups	Number of unique groups.

Value

A new adj list.

Examples

```
a <- adj(konigsberg$bridge_to, ids = konigsberg$area, duplicates = "allow")
# merge two islands (A and D)
adj_quotient(a, c("AD", "B", "C", "AD"))
adj_quotient_int(a, c(1L, 2L, 3L, 1L), n_groups = 3L, self_loops = "allow")
```

---

adj_zero_index *Convert adjacency list to use zero-based indices*

Description

Subtracts 1 from each index in the adjacency list and returns a bare list of integer vectors, suitable for providing to C/C++ code that uses zero-based indexing.

Usage

```
adj_zero_index(x)
```

Arguments

x	An adjacency list.

Value

A list of integer vectors with zero-based indices.

Examples

```
a <- adj(konigsberg$bridge_to, ids = konigsberg$area, duplicates = "allow")
adj_zero_index(a)
```

`format.adj` *Format and print methods for adjacency lists*

Description

Adjacency lists are printed as sets of indices for each node.

Usage

```
## S3 method for class 'adj'
format(x, n = 3, ...)

## S3 method for class 'adj'
print(x, n = 3, ...)
```

Arguments

`x`	An `adj` list.
`n`	Maximum number of neighbors to show before truncating with an ellipsis.
`...`	Ignored.

Value

A character vector representing each entry in the adjacency list.

Examples

```
a = adj(konigsberg$bridge_to, ids = konigsberg$area, duplicates = "allow")
format(a)
print(a, n = 5)
```

konigsberg *The Seven Bridges of Königsberg*

Description

A dataset encoding the adjacency structure of seven bridges in Königsberg, Prussia, as described by Leonhard Euler in 1736.

Usage

```
konigsberg
```

Format

konigsberg**:**

A data frame with 4 rows and 4 columns:

area The four land areas, A-D, as described by Euler. Area 'A' corresponds to the central island of Kneiphof.

bridge_to A list column, where each entry is a character vector listing the areas directly connected by bridges to the area in that row.

x The longitude of the area center, for plotting.

y The latitude of the area center, for plotting.

References

Euler, Leonhard (1741). "Solutio problematis ad geometriam situs pertinentis". *Commentarii Academiae Scientiarum Petropolitanae*: 128–140.

plot.adj *Basic plotting for adjacency lists*

Description

Plots an adjacency list as a set of nodes and edges, with optional coordinate values for each node. Edge thickness is proportional to the number of edges between each pair of nodes. Self loops are represented with larger points.

Usage

```
## S3 method for class 'adj'
plot(x, y = NULL, edges = NULL, nodes = TRUE, xlab = NA, ylab = NA, ...)
```

Arguments

x	An adj list
y	Optional matrix of coordinates for each node. If NULL, nodes are plotted along the diagonal. Other types are accepted as long as they are convertible to a 2-column matrix with as.matrix(y)[, 1:2], which is run internally.
edges	Type of line to use when drawing edges. Passed to graphics::lines(). When y is NULL, defaults to "s" (step function); otherwise defaults to "l" for a straight line.
nodes	If TRUE, nodes are plotted as points; if FALSE, only edges are plotted.
xlab, ylab	Labels for the x- and y-axes.
...	Additional arguments passed on to the initial plot() of the nodes.

Value

NULL, invisibly.

Examples

```
plot(adj(2, c(1, 3), 2))
plot(adj(2, c(1, 2, 3), c(2, 2, 2), self_loops="allow", duplicates="allow"))

a <- adj(konigsberg$bridge_to, ids = konigsberg$area, duplicates = "allow")
plot(a, konigsberg[c("x", "y")])
```

---

t.adj *Transpose an adjacency list*

Description

Reverse the direction of edges in an adjacency list. For undirected graphs, this is a no-op.

Usage

```
## S3 method for class 'adj'
t(x)
```

Arguments

x	An adj list

Value

An adj list with edges reversed.

Examples

```
a <- adj(2, 3, 1)
all(t(a) == adj(3, 1, 2))
```

Index