Graph Analytics in R

My name is Rich,
and I'd like to tell you all
about network graphs.

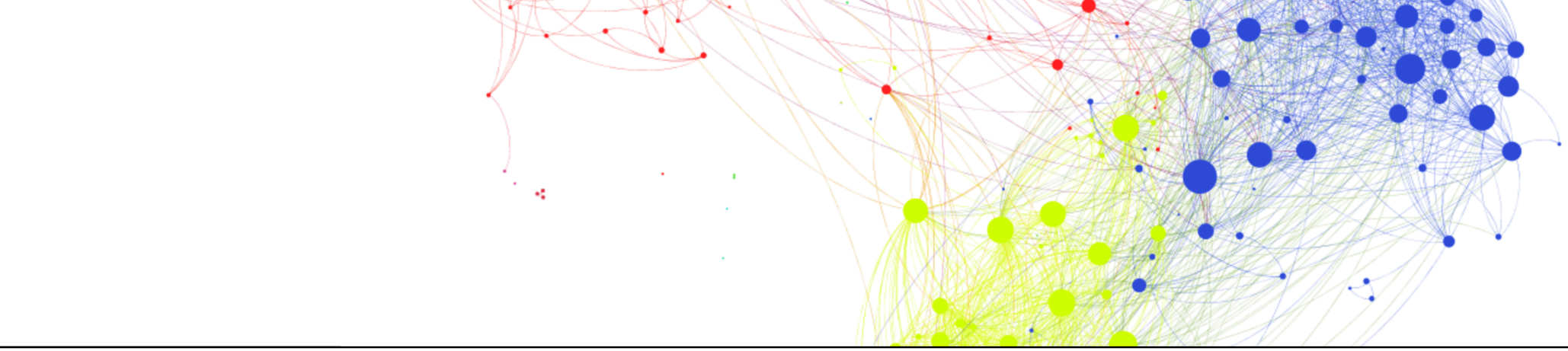
@riannone

riannone@me.com

rich-iannone

richard-iannone

April 25, 2017.

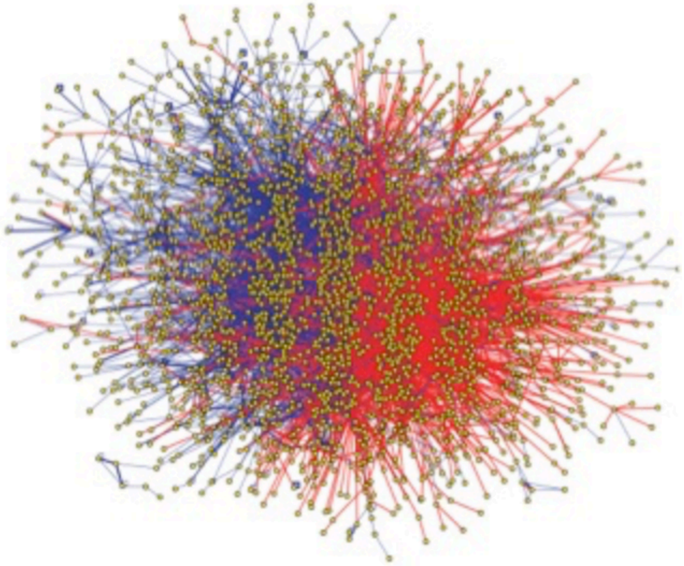


Why represent data as a graph instead of using tables?

— a graph representation can provide a different point of view

— resulting data models are much simpler and expressive than with relational databases

— can build sophisticated models that map closely to our problem domain



Most complex systems are graph-like,
check out this interactome!

SOME GRAPH BASICS

a *graph* is a *network*

a graph, informally, is:

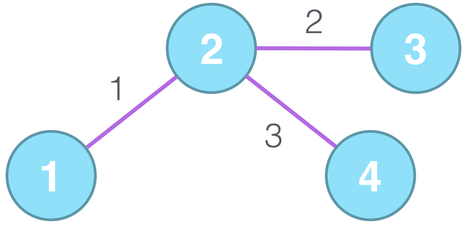
a set of nodes/vertices joined by a set of lines or arrows

a graph, more formally, is:

$G = (X, E)$

X is a set of vertices/nodes

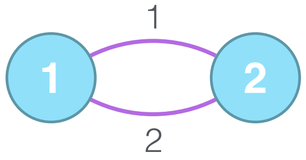
E is a set of vertex/node sets



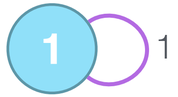
$X = \{x_1, x_2, x_3, x_4\}$

$E = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_2, x_4\}\}$

a *simple graph* doesn't have:

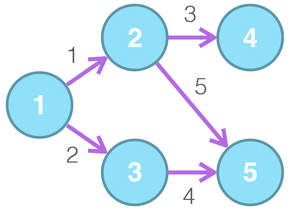


multiple edges

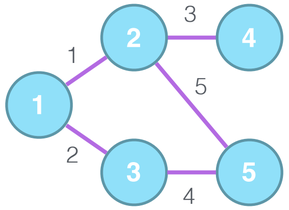


loops

in a *directed graph* the order of nodes in edge sets matters; not so for *undirected graph*s



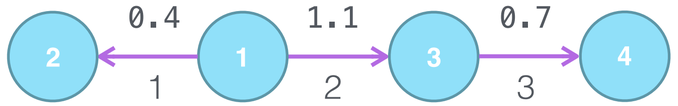
directed



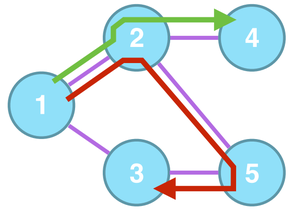
undirected

the *degree* for a node is the number of edges incident on it

a *weighted graph* has weights associated with each edge



a *walk* in a graph is movement between nodes



$\text{walk}(\rightarrow) = \{1, 2, 4\}$

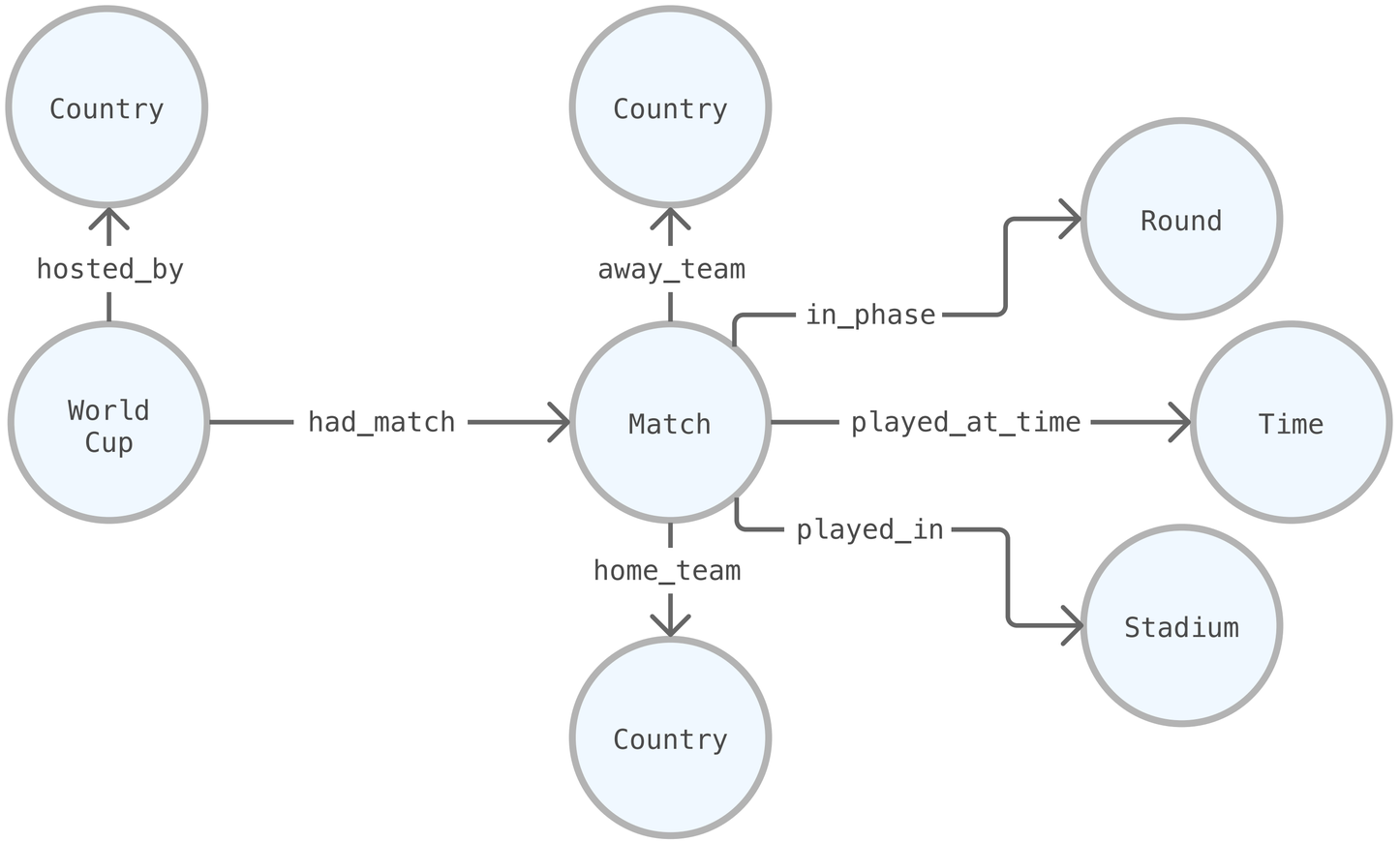
$\text{walk}(\rightarrow) = \{1, 2, 5, 3\}$

a graph is *connected* if one can walk from a node to any other node; else, *disconnected*

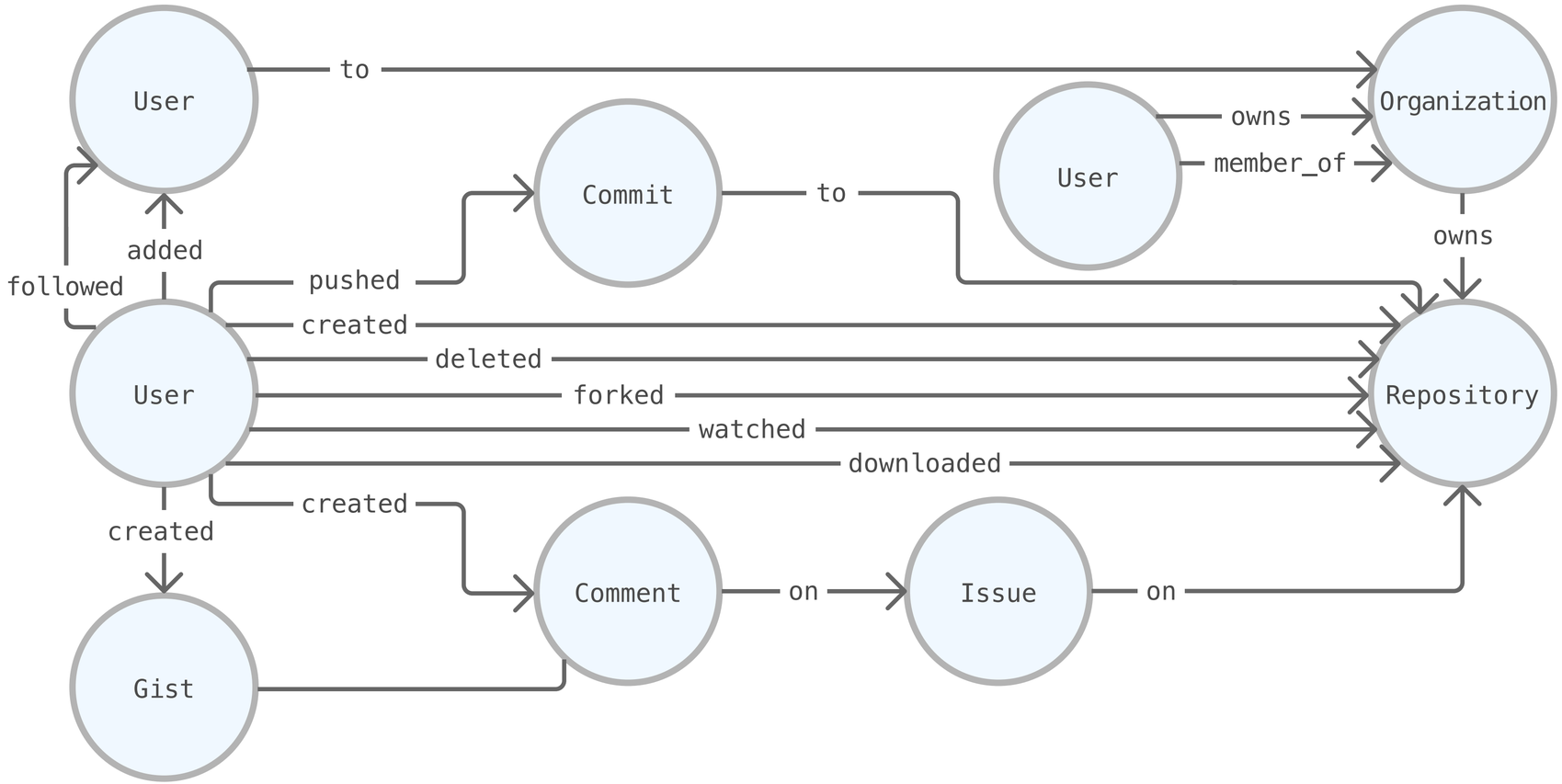
G_1 

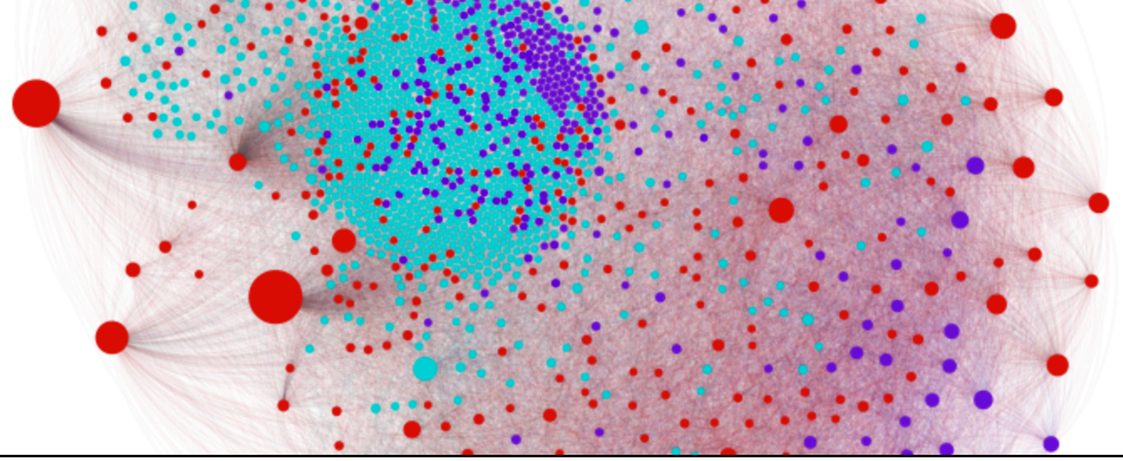
G_2 

AN EXAMPLE GRAPH: THE WORLD CUP



AN EXAMPLE GRAPH: SOFTWARE REPOSITORY





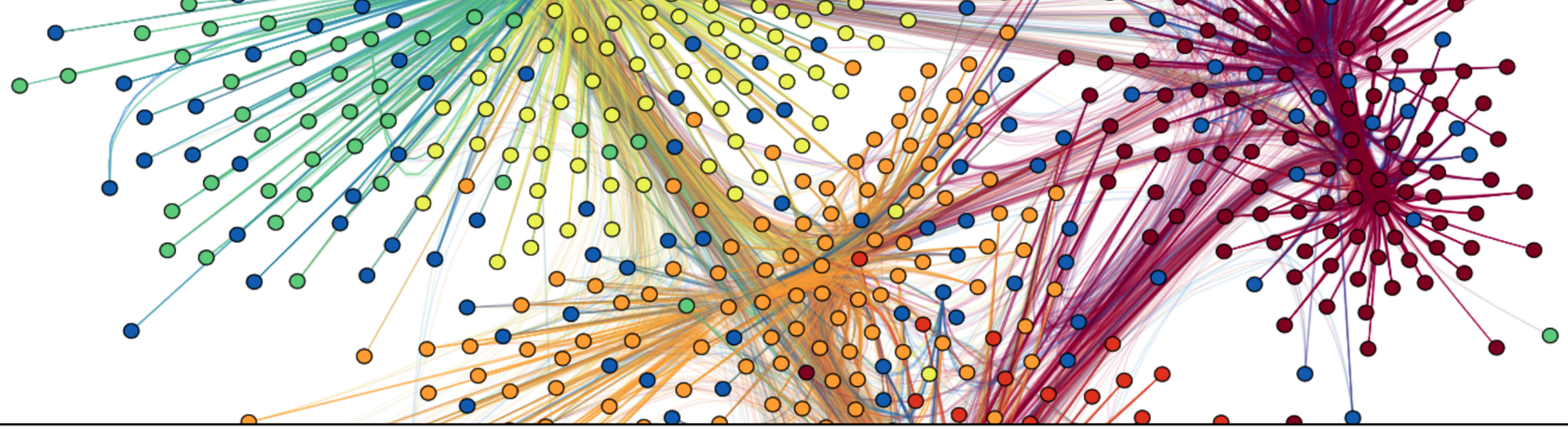
What tools are available for graph analytics?

PYTHON
NetworkX, graph-tool, igraph, SNAP, boost, python-graph

R
graph, igraph, giRaph, diagram, network, RBGL, ggraph, dynamicGraph, DiagrammeR

DESKTOP APPLICATIONS
Cytoscape, BioFabric, Gephi, yEd, Tulip,
Graphviz, Tom Sawyer Perspectives, PGF/TikZ…

GRAPH DATABASES
Neo4J, OrientDB, AllegroGraph, DataStax, SAP HANA…



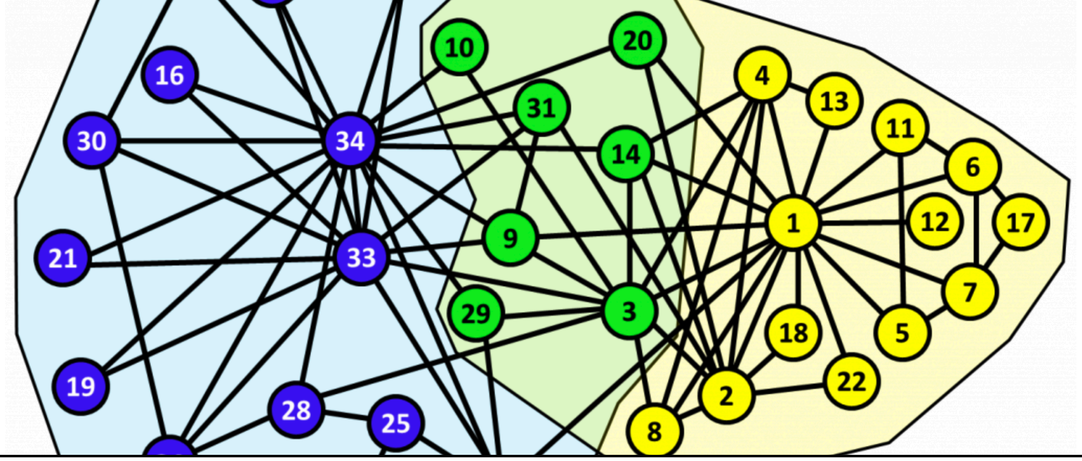
Here, we'll focus on the DiagrammeR package for R

Why does this package exist?

1 to make it simple to develop graphs and graph visualizations in R

2 to provide access to a useful toolset for creating, inspecting, modifying, and analyzing graphs

3 to allow R functions from other packages to modify graph data



Package Features:

180 functions for working with network graphs.

Many automatic and manual graph layout options.

Flexible traversal functions for moving around the graph.

Fully adopts the magrittr %>% operator for left-to-right function piping.

Modify graph data in-place using join, mutation, and recoding functions.

Import/Export from and to several graph and image formats.

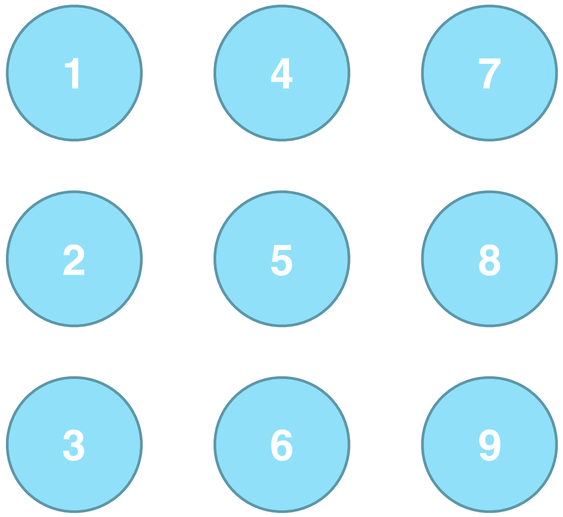
Automatically create highly-compressed graph backups.

Store multiple graphs in a single object to facilitate comparisons.

NODE/EDGE DATA FRAME PROCESSING	collections of nodes & edges
GRAPH OBJECT CREATION AND RENDERING	create and view the graph
GRAPH MODIFICATION	make changes to the graph
GRAPH TRAVERSALS	traverse and query the graph

NODE/EDGE DATA FRAME PROCESSING

collections of nodes & edges



a collection of nodes

a data frame

NODE/EDGE DATA FRAME PROCESSING

collections of nodes & edges

		id	label	type	[attrs]	[attrs]	[attrs]
1	1	...	...	...	...	...	...
2	2	...	...	...	...	...	...
3	3	...	...	...	...	...	...
4	4	...	...	...	...	...	...
5	5	...	...	...	...	...	...
6	6	...	...	...	...	...	...
7	7	...	...	...	...	...	...
8	8	...	...	...	...	...	...
9	9	...	...	...	...	...	...

a data frame for nodes: a node data frame

NODE/EDGE DATA FRAME PROCESSING

collections of nodes & edges

`id` integer-based, unique node ID values

`label` an optional label, exposed during views
(should be unique to make selections easier)

`type` an optional grouping value for nodes

`[attrs]` node attributes, might include:

`x`, `y`, `color`, `fillcolor`, `fontcolor`, `alpha`, `shape`, `peripheries`, `height`, `width`, `tooltip`, `fontname`, `fontsize`

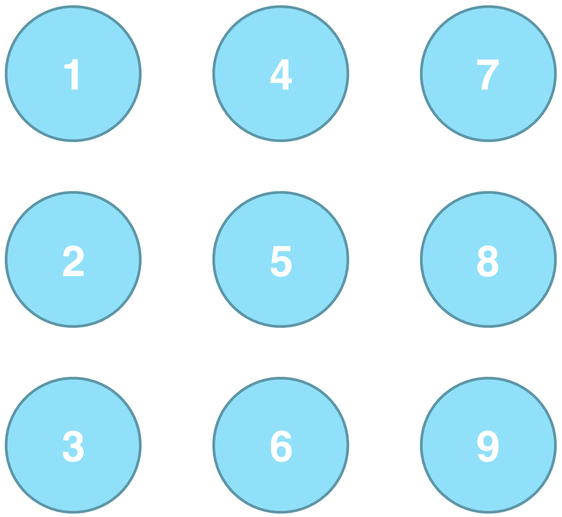
`[attrs]` attributes can include also columns of numerical or character-based data

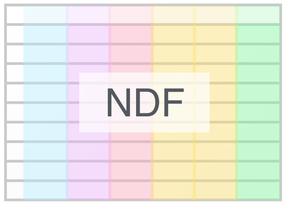
	id	label	type	color	width	value
1	1	1	X	blue	1.3	132.2
2	2	2	X	blue	0.7	98.3
3	3	3	X	red	1.3	182.4
4	4	4	X	red	0.8	112.3
5	5	5	Y	pink	1.1	35.7
6	6	6	Y	blue	0.9	126.2
7	7	7	Y	pink	1.8	252.7
8	8	8	Z	red	1.1	154.4
9	9	9	Z	pink	1.2	168.5

node data frame — NDF

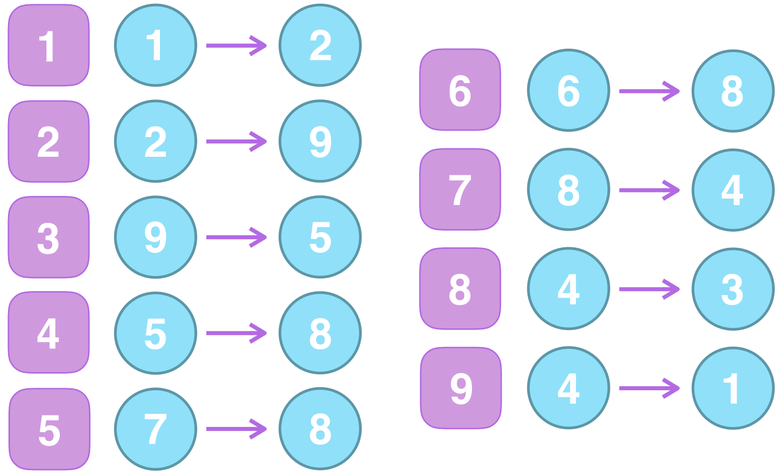
NODE/EDGE DATA FRAME PROCESSING

collections of nodes & edges





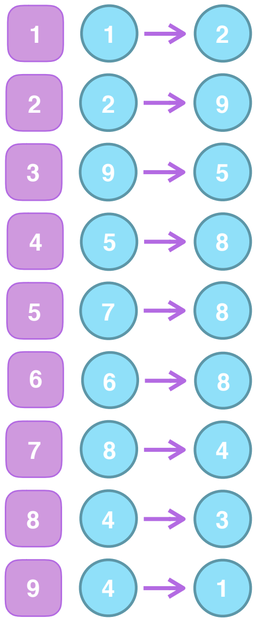
a collection of nodes — NDF



a collection of edges — EDF

NODE/EDGE DATA FRAME PROCESSING

collections of nodes & edges



	id	from	to	rel	[attrs]	[attrs]
1	1	...	...	...		
2	2	...	...	...		
3	3	...	...	...		
4	4	...	...	...		
5	5	...	...	...		
6	6	...	...	...		
7	7	...	...	...		
8	8	...	...	...		
9	9	...	...	...		

a data frame for edges: an edge data frame

NODE/EDGE DATA FRAME PROCESSING

collections of nodes & edges

`id` integer-based, unique edge ID values

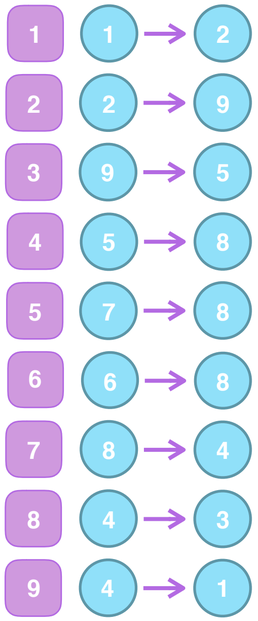
`from, to` the node ID values that define an edge between nodes

`rel` an optional grouping value for edges

`[attrs]` edge attributes, might include:

`color`, `alpha`, `penwidth`, `arrowhead`, `arrowtail`, `tooltip`, `fontname`, `fontsize`

`[attrs]` attributes can include also columns of numerical or character-based data

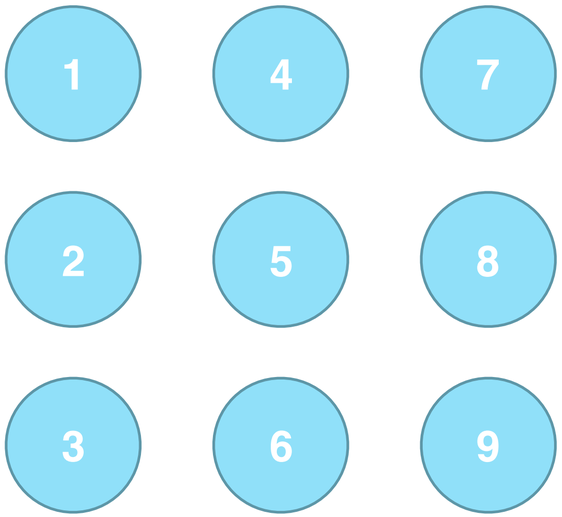


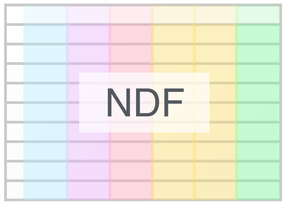
	id	from	to	rel	[attrs]	[attrs]
1	1	...	...	...		
2	2	...	...	...		
3	3	...	...	...		
4	4	...	...	...		
5	5	...	...	...		
6	6	...	...	...		
7	7	...	...	...		
8	8	...	...	...		
9	9	...	...	...		

edge data frame — EDF

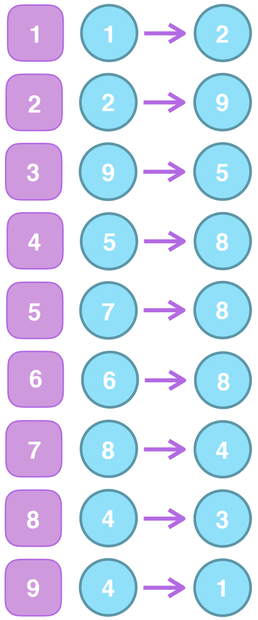
NODE/EDGE DATA FRAME PROCESSING

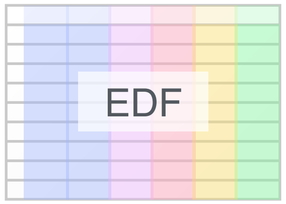
collections of nodes & edges





a collection of nodes — NDF





a collection of edges — EDF

NODE/EDGE DATA FRAME PROCESSING

collections of nodes & edges

GRAPH OBJECT CREATION AND RENDERING

create and view the graph

GRAPH MODIFICATION

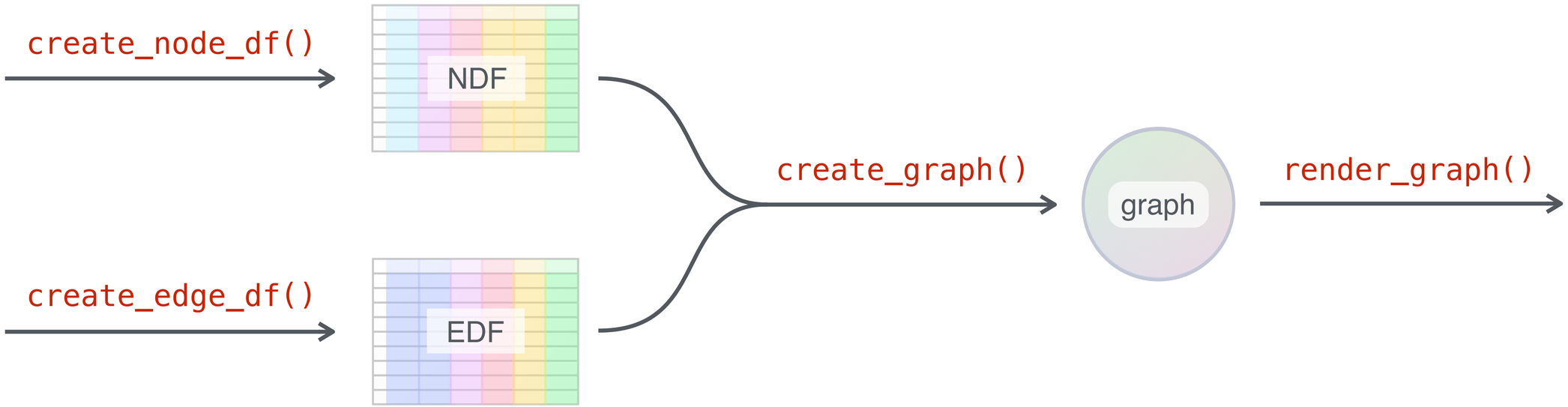
make changes to the graph

GRAPH TRAVERSALS

traverse and query the graph

GRAPH OBJECT CREATION AND RENDERING

create and view the graph



GRAPH OBJECT CREATION AND RENDERING

create and view the graph

R SCRIPT creating NDFs and EDFs

```
library(DiagrammeR)

ndf <-
  create_node_df(
    n = 4,
    label = c(“dzlw”, “kpsq”, “odje”, “jdmc”),
    type = c(“a”, “a”, “b”, “b”))
```

Create an NDF with `create_node_df()`. Specify the number of nodes with `n`. Give each node a `label` (unique) and a `type` (for categorization).

GRAPH OBJECT CREATION AND RENDERING

create and view the graph

R SCRIPT creating NDFs and EDFs

```
library(DiagrammeR)

ndf <-
  create_node_df(
    n = 4,
    label = c("dzlw", "kpsq", "odje", "jdmc"),
    type = c("a", "a", "b", "b"))

edf <-
  create_edge_df(
    from = c(1, 2, 3),
    to = c(4, 3, 1),
    rel = "a",
    length = c(50, 100, 250))
```

Create an NDF with `create_node_df()`. Specify the number of nodes with `n`. Give each node a `label` (unique) and a `type` (for categorization).

Create an EDF with `create_edge_df()`. The `from` and `to` vectors indicate the links between nodes:

`1->4 2->3 3->1`

All edges here have a `rel` value of `a`. Data for each edge is provided in the `length` edge attribute.

GRAPH OBJECT CREATION AND RENDERING

create and view the graph

R SCRIPT creating NDFs and EDFs

```
library(DiagrammeR)

ndf <-
  create_node_df(
    n = 4,
    label = c("dzlw", "kpsq", "odje", "jdmc"),
    type = c("a", "a", "b", "b"))

edf <-
  create_edge_df(
    from = c(1, 2, 3),
    to = c(4, 3, 1),
    rel = "a",
    length = c(50, 100, 250))

graph <-
  create_graph(
    nodes_df = ndf,
    edges_df = edf)
```

Create an NDF with `create_node_df()`. Specify the number of nodes with `n`. Give each node a `label` (unique) and a `type` (for categorization).

Create an EDF with `create_edge_df()`. The `from` and `to` vectors indicate the links between nodes:

`1->4 2->3 3->1`

All edges here have a `rel` value of `a`. Data for each edge is provided in the `length` edge attribute.

Create the graph object (`graph`) using the `create_graph()` function, supplying the `ndf` and `edf`.

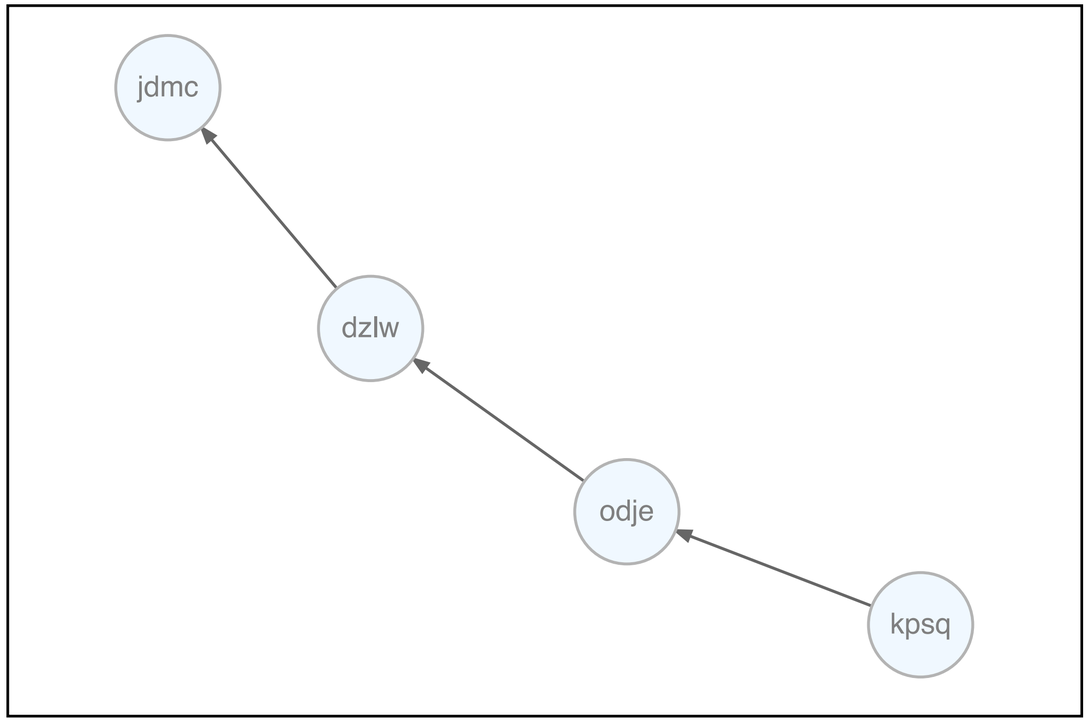
GRAPH OBJECT CREATION AND RENDERING

create and view the graph

R SCRIPT producing a graph diagram

```
ndf <-
  create_node_df(
    n = 4,
    label = c(“dzlw”, “kpsq”, “odje”, “jdmc”),
    type = c(“a”, “a”, “b”, “b”))

edf <-
  create_edge_df(
    from = c(1, 2, 3),
    to = c(4, 3, 1),
    rel = “a”,
    length = c(50, 100, 250))

graph <-
  create_graph(
    nodes_df = ndf,
    edges_df = edf)

render_graph(graph = graph)
```

GRAPH VIEWER



Render the graph object with the `render_graph()` function.

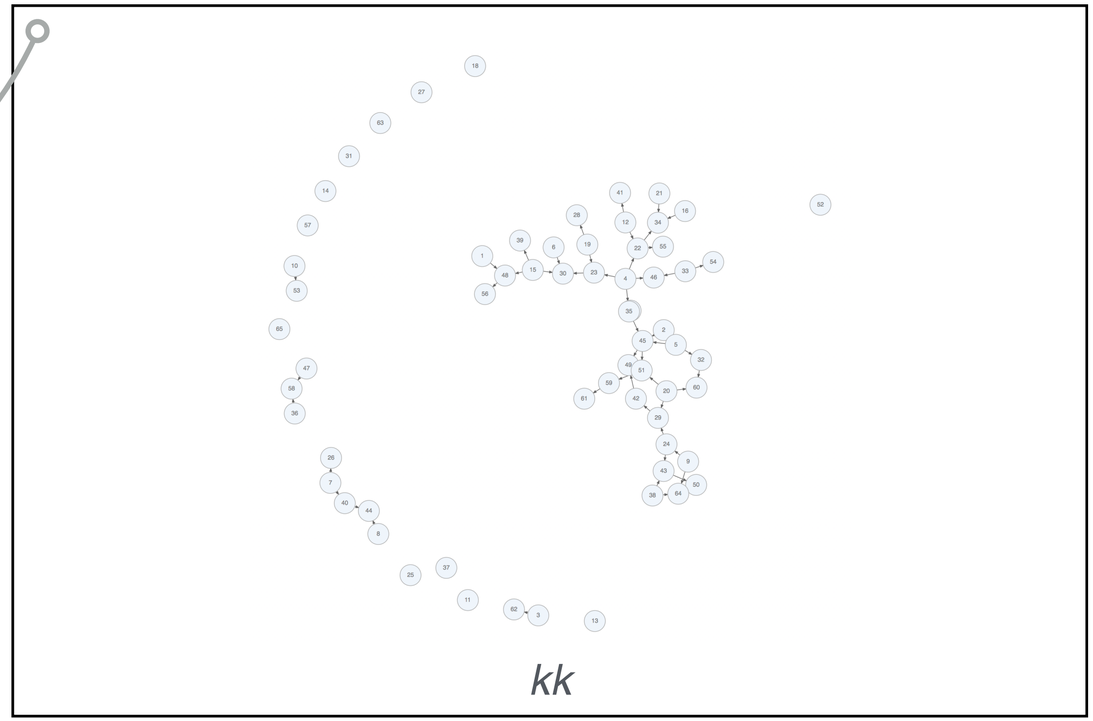
GRAPH OBJECT CREATION AND RENDERING

create and view the graph

R SCRIPT try different layouts with a larger graph

```
# Create a larger graph with `create_random_graph()`
larger_graph <-
  create_random_graph(
    n = 65, m = 50, set_seed = 23)

# Use the `kk` and `fr` layouts (force-directed)
render_graph(graph = larger_graph, layout = "kk")
```

GRAPH VIEWER



The different layouts provided in `render_graph()` allow for visualizations suitable for the graph at hand.

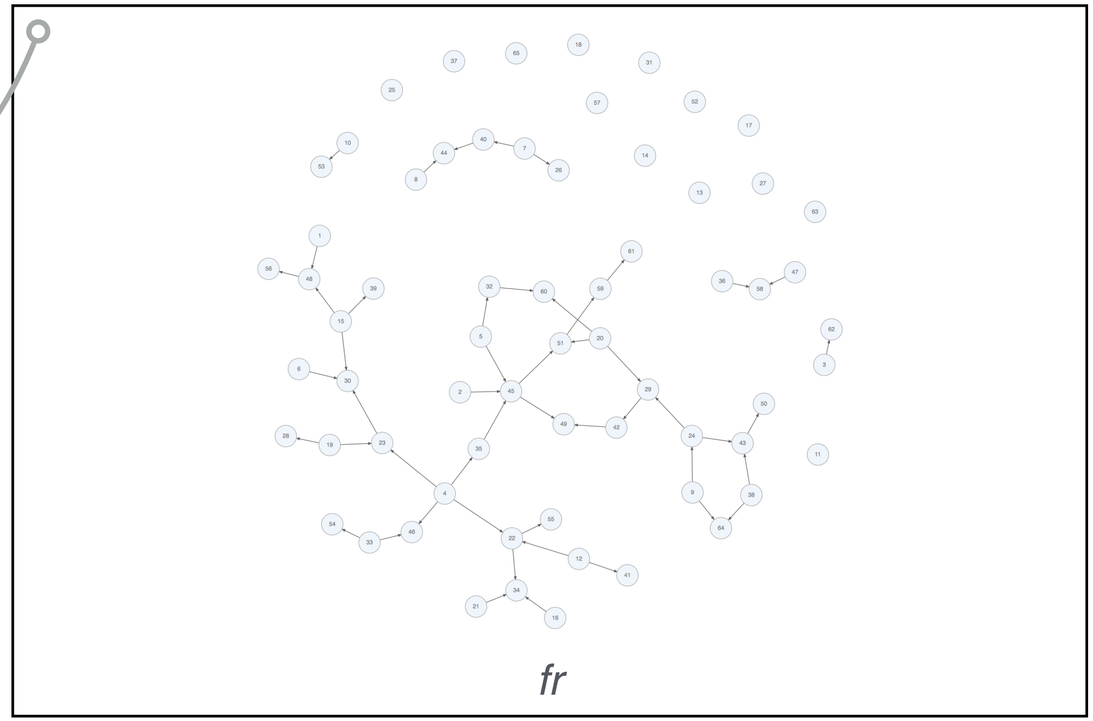
GRAPH OBJECT CREATION AND RENDERING

create and view the graph

R SCRIPT try different layouts with a larger graph

```
# Create a larger graph with `create_random_graph()`
larger_graph <-
  create_random_graph(
    n = 65, m = 50, set_seed = 23)

# Use the `kk` and `fr` layouts (force-directed)
render_graph(graph = larger_graph, layout = "kk")
render_graph(graph = larger_graph, layout = "fr")
```

GRAPH VIEWER



The different layouts provided in `render_graph()` allow for visualizations suitable for the graph at hand.

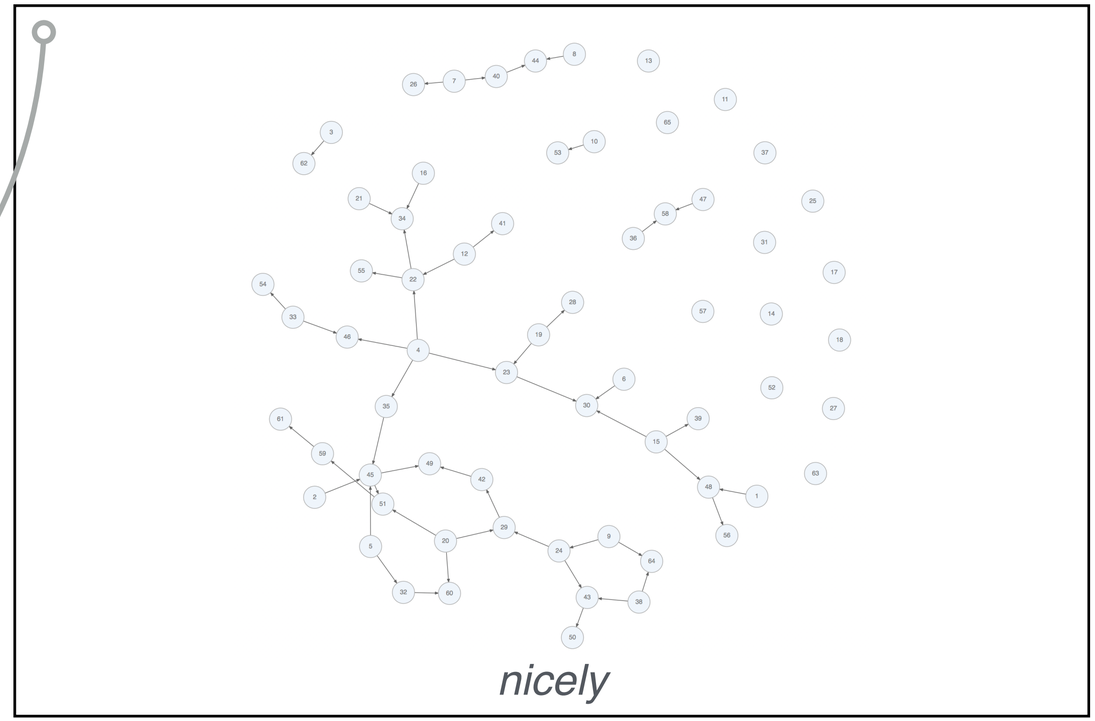
GRAPH OBJECT CREATION AND RENDERING

create and view the graph

R SCRIPT try different layouts with a larger graph

```
# Create a larger graph with `create_random_graph()`
larger_graph <-
  create_random_graph(
    n = 65, m = 50, set_seed = 23)

# Use the `kk` and `fr` layouts (force-directed)
render_graph(graph = larger_graph, layout = "kk")
render_graph(graph = larger_graph, layout = "fr")

# Use `nicely` to have the pkg choose a nice layout
render_graph(graph = larger_graph, layout = "nicely")
```

GRAPH VIEWER



nicely

The different layouts provided in `render_graph()` allow for visualizations suitable for the graph at hand.

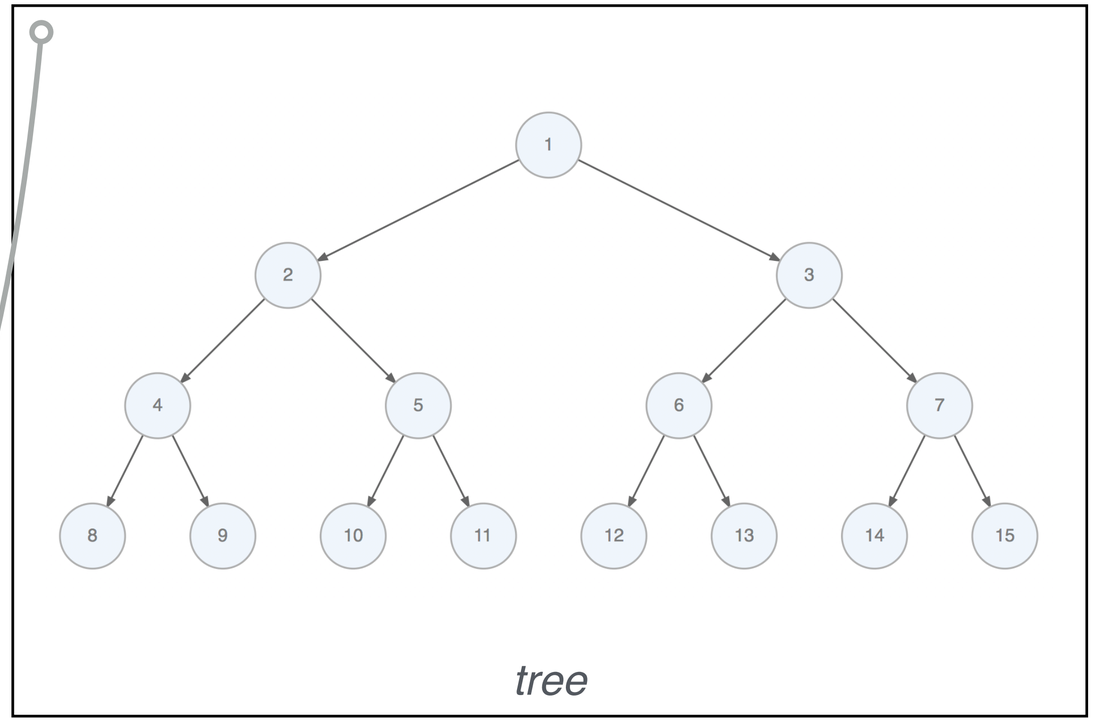
GRAPH OBJECT CREATION AND RENDERING

create and view the graph

R SCRIPT try different layouts with a larger graph

```
# Create a larger graph with `create_random_graph()`
larger_graph <-
  create_random_graph(
    n = 65, m = 50, set_seed = 23)

# Use the `kk` and `fr` layouts (force-directed)
render_graph(graph = larger_graph, layout = "kk")
render_graph(graph = larger_graph, layout = "fr")

# Use `nicely` to have the pkg choose a nice layout
render_graph(graph = larger_graph, layout = "nicely")

# Create a tree graph
tree_graph <-
  add_balanced_tree(create_graph(), k = 2, h = 3)

# Use the `tree` layout (hierarchical)
render_graph(graph = tree_graph, layout = "tree")
```

GRAPH VIEWER



The different layouts provided in `render_graph()` allow for visualizations suitable for the graph at hand.

NODE/EDGE DATA FRAME PROCESSING

collections of nodes & edges

GRAPH OBJECT CREATION AND RENDERING

create and view the graph

GRAPH MODIFICATION

make changes to the graph

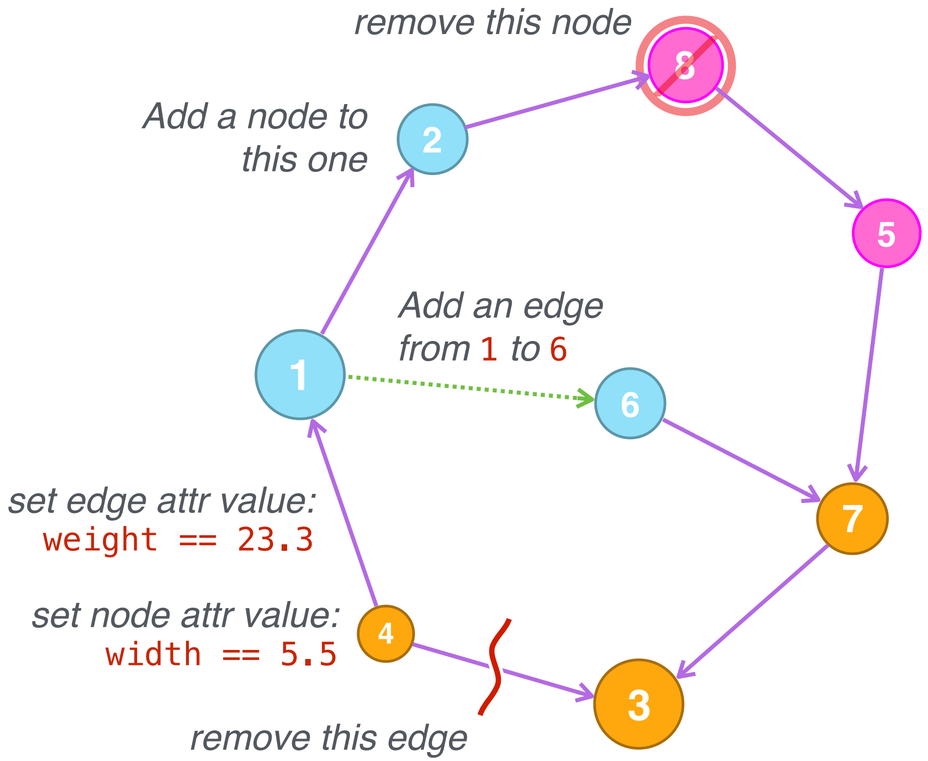
GRAPH TRAVERSALS

traverse and query the graph

GRAPH MODIFICATION

make changes to the graph

So, we have this graph right here…



Let's modify it a bit.

We can use some of these functions:

`add_node()`

`add_edge()`

`delete_node()`

`delete_edge()`

`set_node_attrs()`

`set_edge_attrs()`

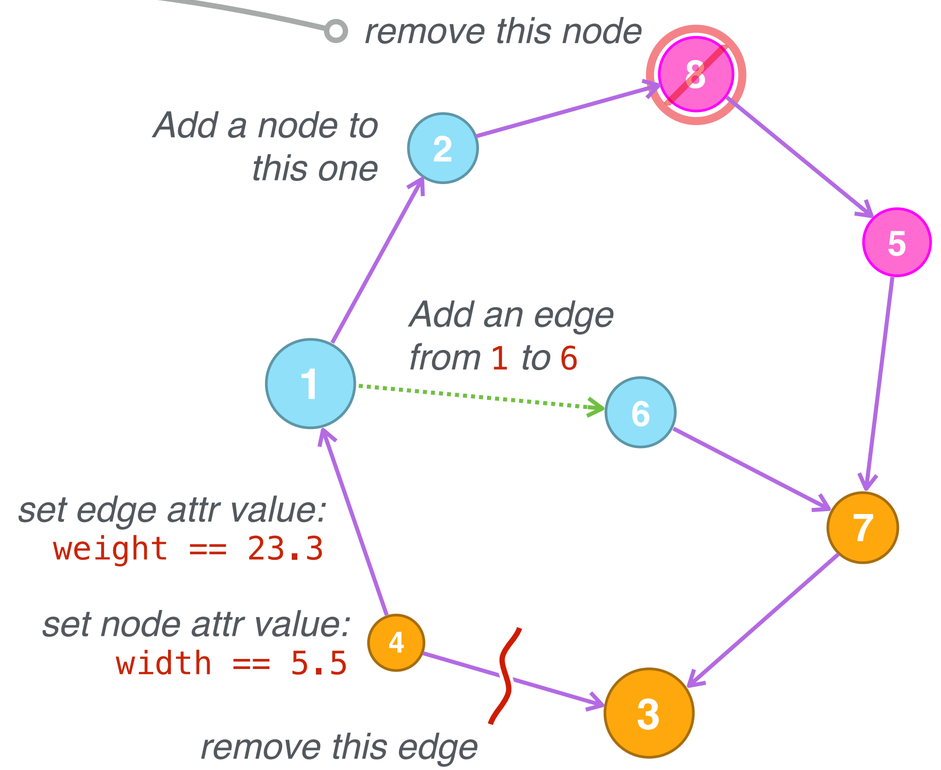
GRAPH MODIFICATION

make changes to the graph

R SCRIPT

modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)
```

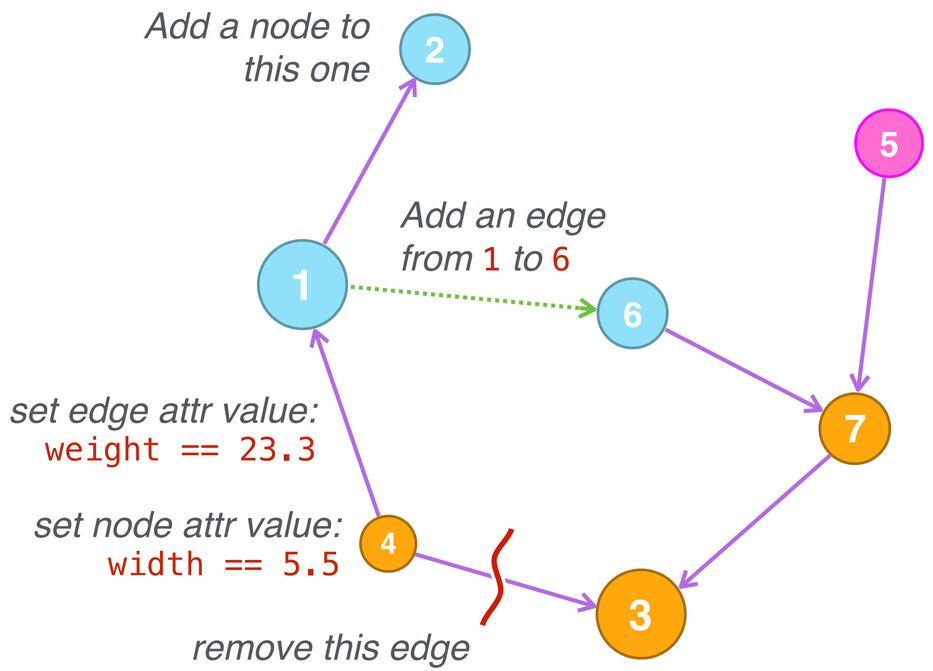


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)
```

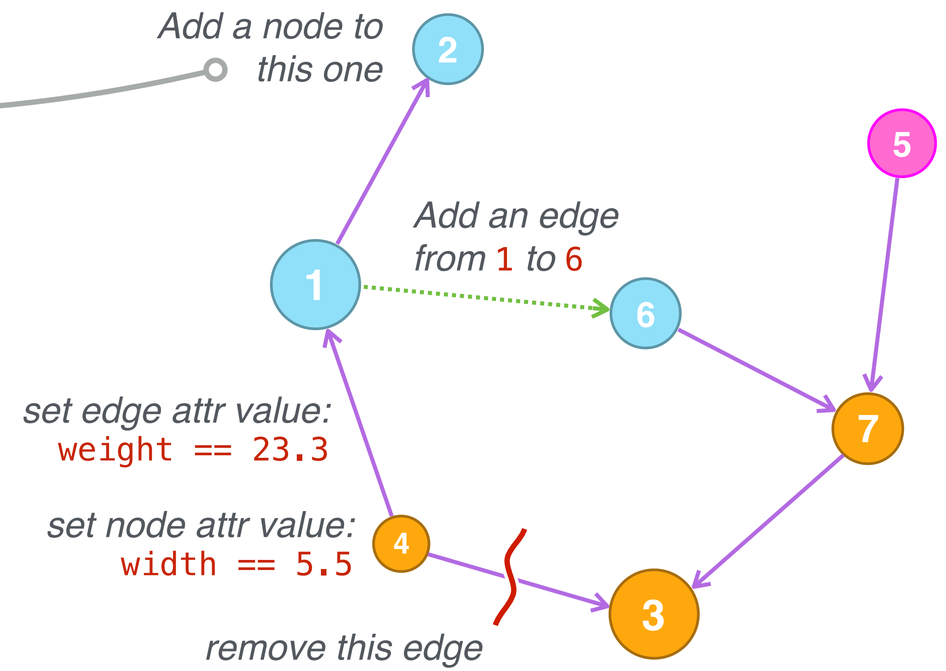


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)

# [2] Adding a node
graph <- graph %>% add_node(to = 2)
```

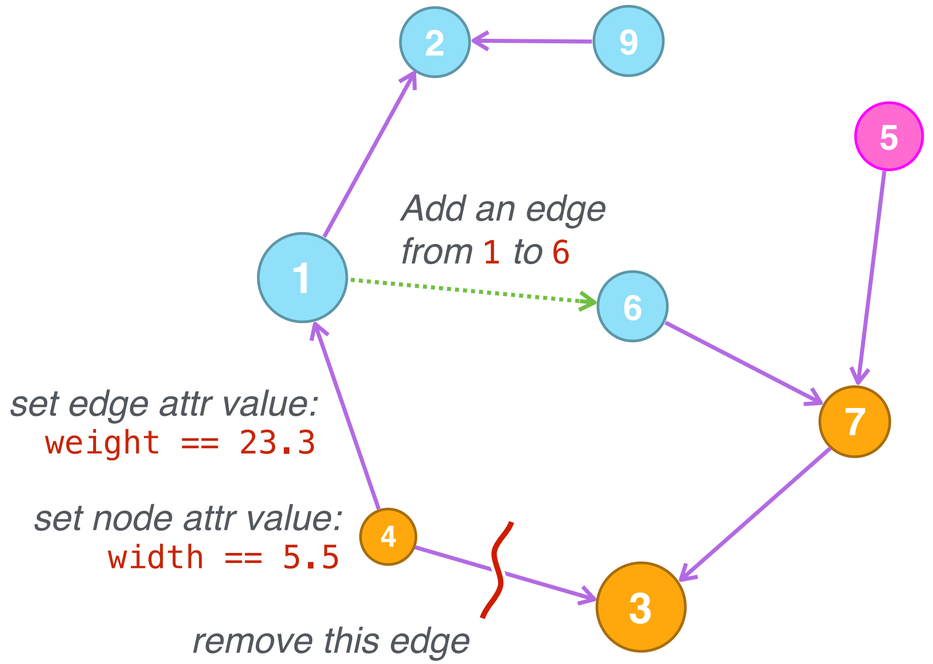


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)

# [2] Adding a node
graph <- graph %>% add_node(to = 2)
```

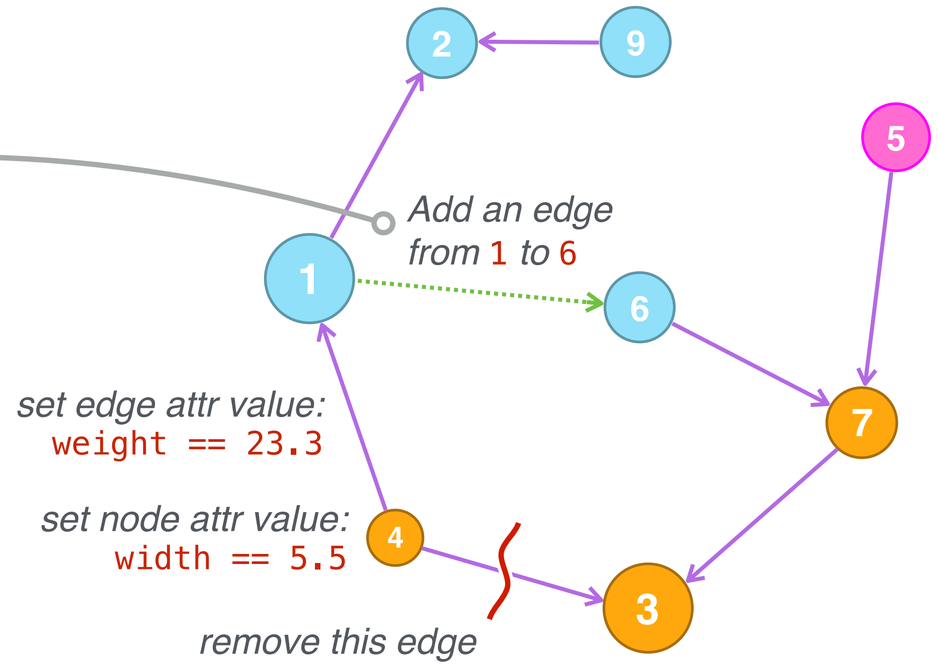


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)

# [2] Adding a node
graph <- graph %>% add_node(to = 2)

# [3] Adding an edge
graph <- graph %>% add_edge(from = 1, to = 6)
```

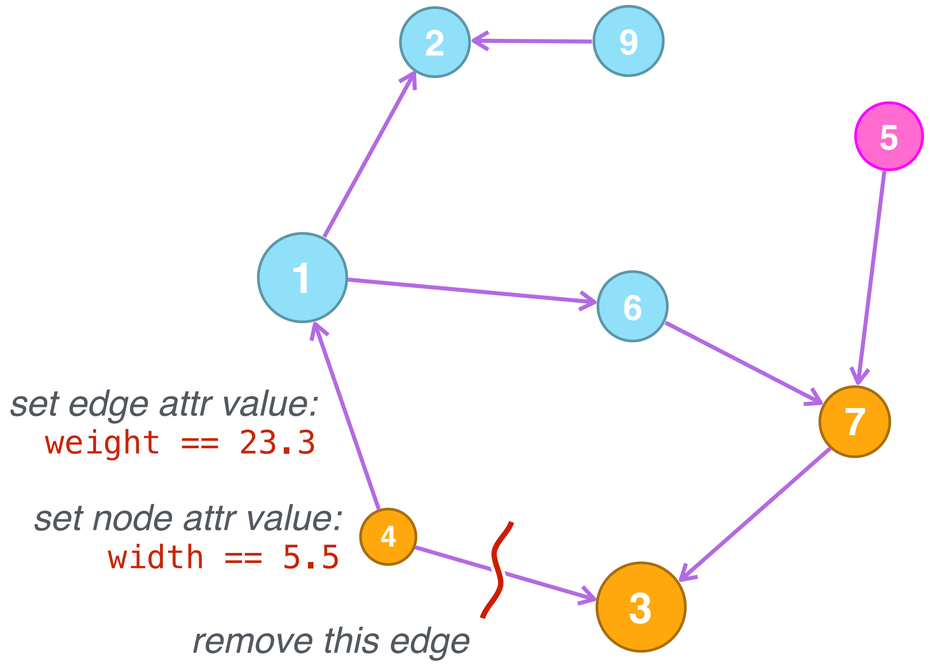


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)

# [2] Adding a node
graph <- graph %>% add_node(to = 2)

# [3] Adding an edge
graph <- graph %>% add_edge(from = 1, to = 6)
```

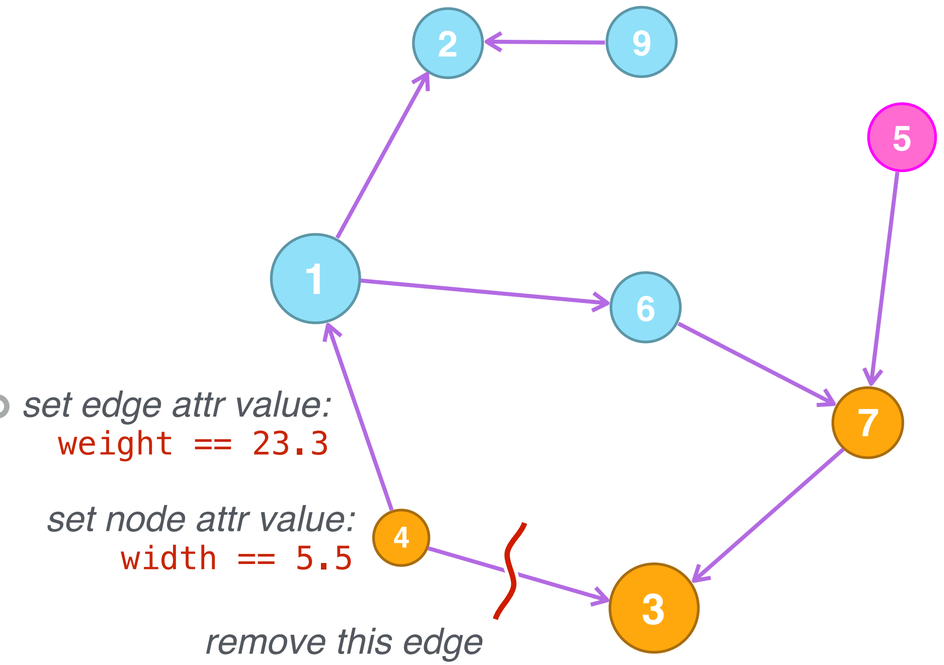


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)

# [2] Adding a node
graph <- graph %>% add_node(to = 2)

# [3] Adding an edge
graph <- graph %>% add_edge(from = 1, to = 6)

# [4] Setting an edge attribute value
graph <- graph %>%
  set_edge_attrs("weight", 23.3, from = 4, to = 1)
```

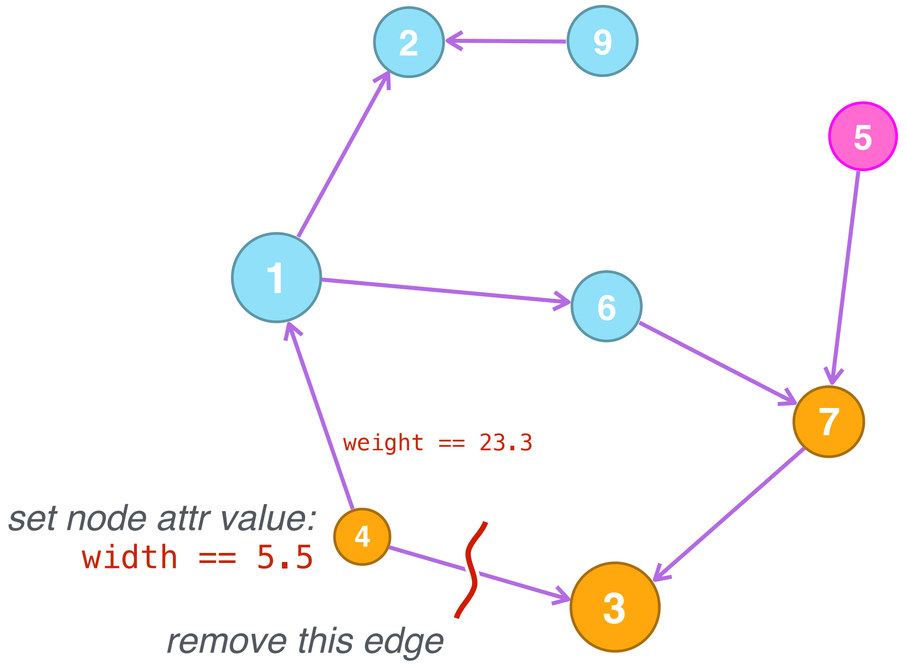


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```r
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)

# [2] Adding a node
graph <- graph %>% add_node(to = 2)

# [3] Adding an edge
graph <- graph %>% add_edge(from = 1, to = 6)

# [4] Setting an edge attribute value
graph <- graph %>%
  set_edge_attrs("weight", 23.3, from = 4, to = 1)
```

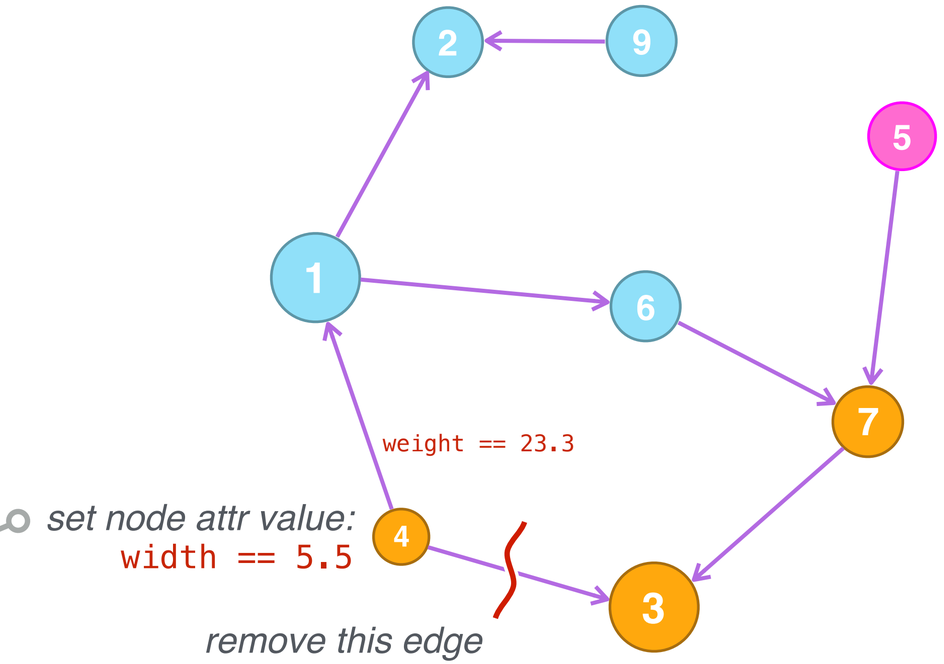


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)

# [2] Adding a node
graph <- graph %>% add_node(to = 2)

# [3] Adding an edge
graph <- graph %>% add_edge(from = 1, to = 6)

# [4] Setting an edge attribute value
graph <- graph %>%
  set_edge_attrs("weight", 23.3, from = 4, to = 1)

# [5] Setting a node attribute value
graph <- graph %>%
  set_node_attrs("width", 5.5, nodes = 4)
```

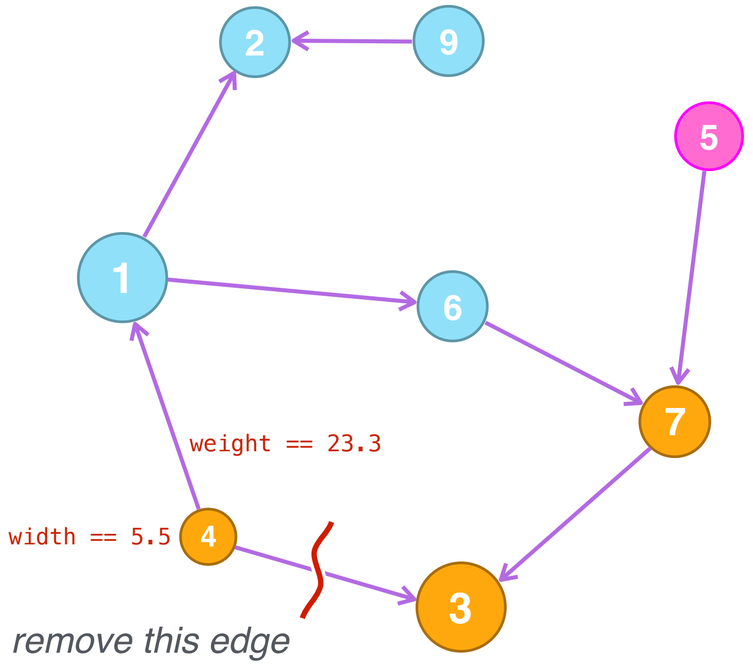


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)

# [2] Adding a node
graph <- graph %>% add_node(to = 2)

# [3] Adding an edge
graph <- graph %>% add_edge(from = 1, to = 6)

# [4] Setting an edge attribute value
graph <- graph %>%
  set_edge_attrs("weight", 23.3, from = 4, to = 1)

# [5] Setting a node attribute value
graph <- graph %>%
  set_node_attrs("width", 5.5, nodes = 4)
```

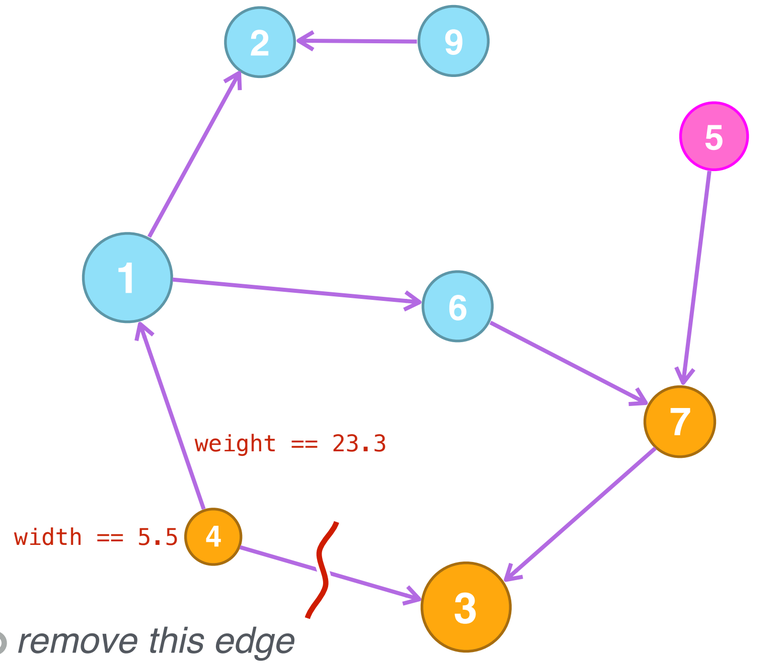


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)

# [2] Adding a node
graph <- graph %>% add_node(to = 2)

# [3] Adding an edge
graph <- graph %>% add_edge(from = 1, to = 6)

# [4] Setting an edge attribute value
graph <- graph %>%
  set_edge_attrs("weight", 23.3, from = 4, to = 1)

# [5] Setting a node attribute value
graph <- graph %>%
  set_node_attrs("width", 5.5, nodes = 4)

# [6] Removing an edge
graph <- graph %>% delete_edge(from = 4, to = 3)
```

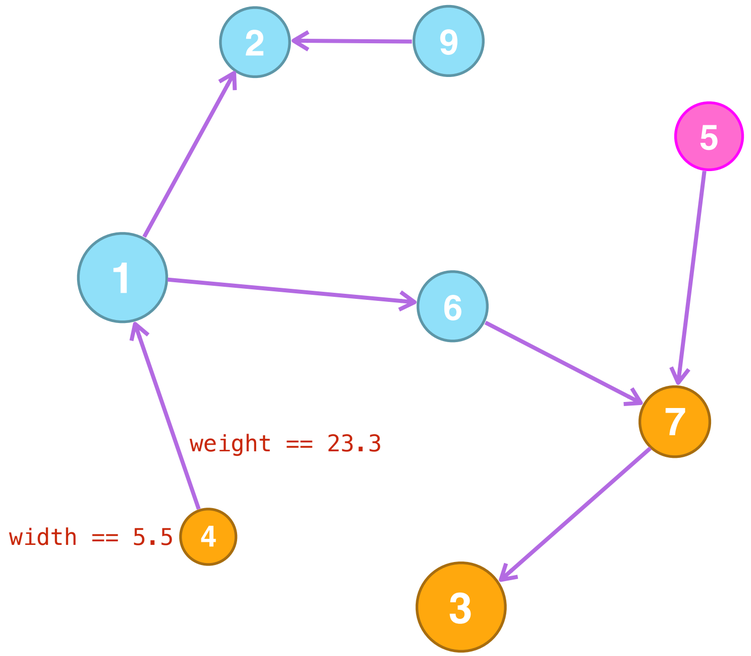


GRAPH MODIFICATION

make changes to the graph

R SCRIPT modifying our graph

```
# [1] Removing a node
graph <- graph %>% delete_node(node = 8)

# [2] Adding a node
graph <- graph %>% add_node(to = 2)

# [3] Adding an edge
graph <- graph %>% add_edge(from = 1, to = 6)

# [4] Setting an edge attribute value
graph <- graph %>%
  set_edge_attrs("weight", 23.3, from = 4, to = 1)

# [5] Setting a node attribute value
graph <- graph %>%
  set_node_attrs("width", 5.5, nodes = 4)

# [6] Removing an edge
graph <- graph %>% delete_edge(from = 4, to = 3)
```



GRAPH MODIFICATION

make changes to the graph

Sometimes we need to get information from the modified graph

— we can verify a that a modification occurred and we can validate it

— we can use an inspection function as part of another function that checks for a graph condition

```
if (node_count(graph) > 5) {
  graph <- [modification function]
}
```

— we can use the output of an inspection as an input for another graph function

```
graph <-
  create_graph() %>%
    add_path(5) %>%
    add_node(to = get_node_ids(.))
```

NODE/EDGE DATA FRAME PROCESSING	collections of nodes & edges
GRAPH OBJECT CREATION AND RENDERING	create and view the graph
GRAPH MODIFICATION	make changes to the graph
GRAPH TRAVERSALS	traverse and query the graph

GRAPH TRAVERSALS

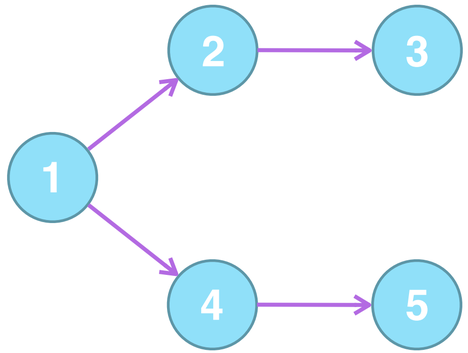
traverse and query the graph

Sometimes we need to traverse the the graph to perform queries on it

— we can do this with a combination of selections and traversals

---

Let's traverse out from node 1
by two steps…



TRAVERSALS

1. select node 1
2. traverse outward from 1 to other nodes
3. traverse outward again to another set of nodes

GRAPH TRAVERSALS

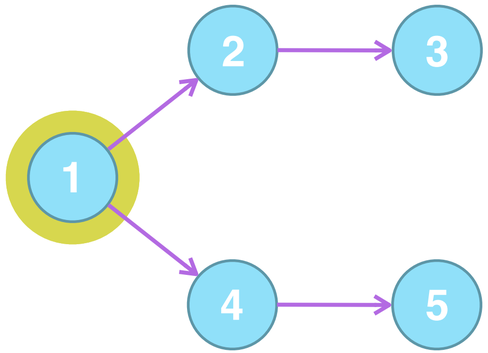
traverse and query the graph

Sometimes we need to traverse the the graph to perform queries on it

— we can do this with a combination of selections and traversals

---

Let's traverse out from node 1
by two steps…



TRAVERSALS

- **1** select node 1
  2 traverse outward from 1 to other nodes
  3 traverse outward again to another set of nodes

GRAPH TRAVERSALS

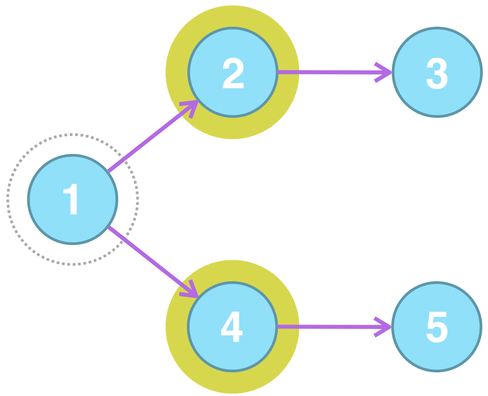
traverse and query the graph

Sometimes we need to traverse the the graph to perform queries on it

— we can do this with a combination of selections and traversals

---

Let's traverse out from node 1
by two steps…



TRAVERSALS

1. select node 1
2. traverse outward from 1 to other nodes
3. traverse outward again to another set of nodes

GRAPH TRAVERSALS

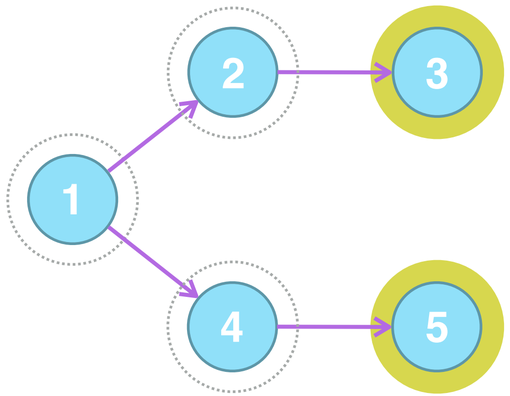
traverse and query the graph

Sometimes we need to traverse the the graph to perform queries on it

— we can do this with a combination of selections and traversals

---

Let's traverse out from node 1
by two steps…



TRAVERSALS

1. select node 1
2. traverse outward from 1 to other nodes
3. traverse outward again to another set of nodes

GRAPH TRAVERSALS

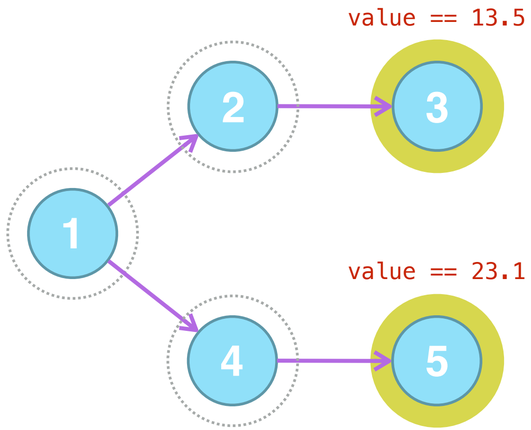
traverse and query the graph

Sometimes we need to traverse the the graph to perform queries on it

— we can do this with a combination of selections and traversals

---

Let's traverse out from node 1
by two steps…



`max(c(13.5, 23.1)) == 23.1`

TRAVERSALS

1. select node 1
2. traverse outward from 1 to other nodes
3. traverse outward again to another set of nodes

QUERY

for 3 and 5, get the maximum value from the `value` node attribute

GRAPH TRAVERSALS

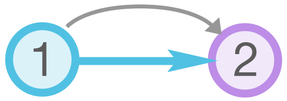
traverse and query the graph

Sometimes we need to traverse the the graph to perform queries on it
— we can do this with a combination of selections and traversals

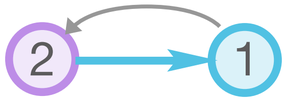
---

For traversals we can use these functions:

NODE-TO-NODE

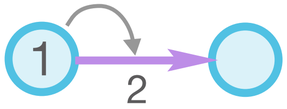


`trav_out()`

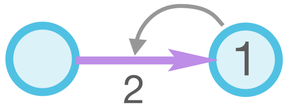


`trav_in()`

NODE-TO-EDGE

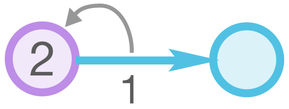


`trav_out_edge()`

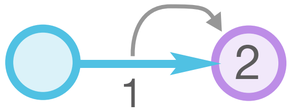


`trav_in_edge()`

EDGE-TO-NODE



`trav_out_node()`



`trav_in_node()`

We also need functions to get our initial selection, so use:

`select_nodes()`

`select_edges()`

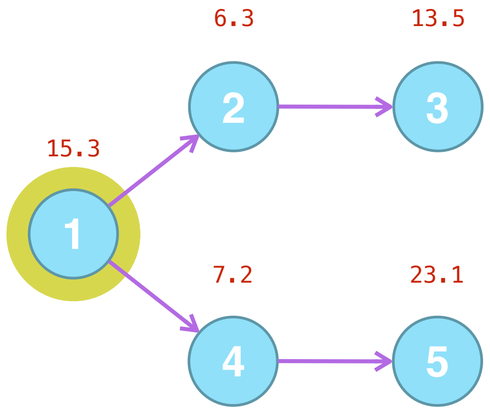
GRAPH TRAVERSALS

traverse and query the graph

R SCRIPT

traversing our graph

```
# [1] Selecting node with ID `1`
graph <- graph %>% select_nodes(nodes = 1)
```



TRAVERSALS

1. select node `1` with `select_nodes()`
2. traverse outward other nodes with `trav_out()`
3. traverse outward again with `trav_out()`

QUERY

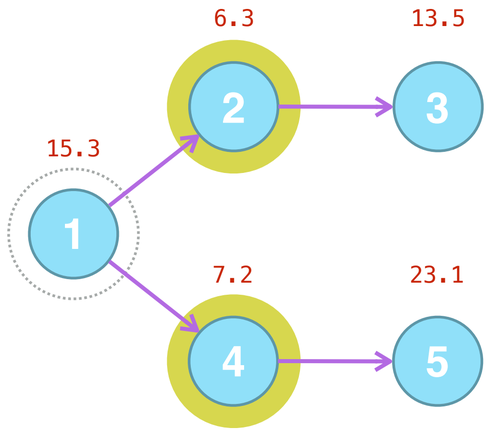
with new selection (from traversals) get max value of the `value` node attribute with `max()`

GRAPH TRAVERSALS

traverse and query the graph

R SCRIPT traversing our graph

```
# [1] Selecting node with ID `1`
graph <- graph %>% select_nodes(nodes = 1)

# [2] Traverse outward to other nodes.
graph <- graph %>% trav_out()
```



TRAVERSALS

1. select node `1` with `select_nodes()`
2. traverse outward other nodes with `trav_out()`
3. traverse outward again with `trav_out()`

QUERY

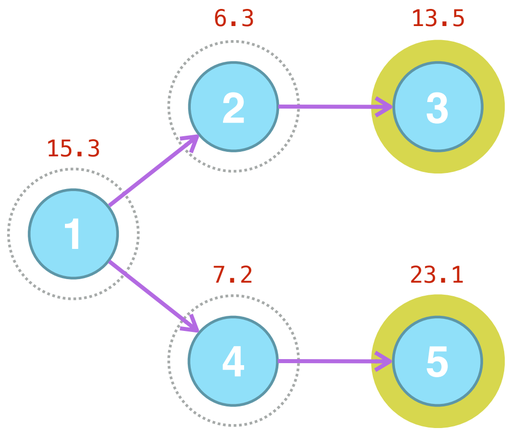
with new selection (from traversals) get max value of the `value` node attribute with `max()`

GRAPH TRAVERSALS

traverse and query the graph

R SCRIPT traversing our graph

```
# [1] Selecting node with ID `1`
graph <- graph %>% select_nodes(nodes = 1)

# [2] Traverse outward to other nodes.
graph <- graph %>% trav_out()

# [3] Traverse outward again by nodes.
graph <- graph %>% trav_out()
```



TRAVERSALS

1. select node `1` with `select_nodes()`
2. traverse outward other nodes with `trav_out()`
3. traverse outward again with `trav_out()`

QUERY

with new selection (from traversals) get max value of the `value` node attribute with `max()`

GRAPH TRAVERSALS

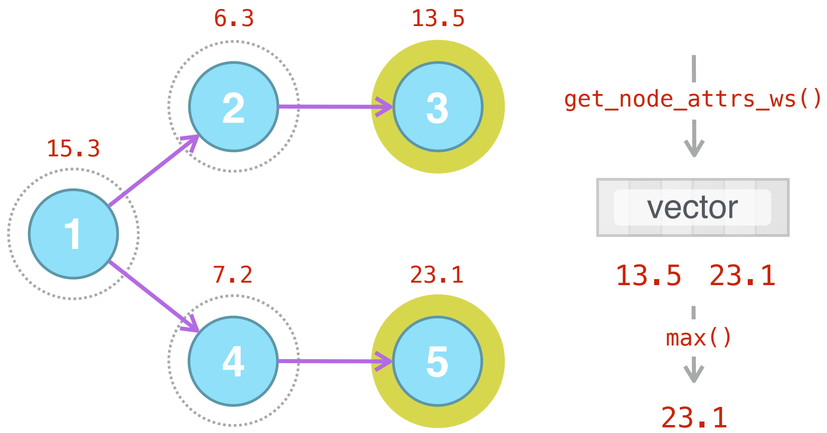
traverse and query the graph

R SCRIPT traversing our graph

```
# [1] Selecting node with ID `1`
graph <- graph %>% select_nodes(nodes = 1)

# [2] Traverse outward to other nodes.
graph <- graph %>% trav_out()

# [3] Traverse outward again by nodes.
graph <- graph %>% trav_out()

# Query: Get maximum of `value` from traversed nodes
graph %>%
  get_node_attrs_ws(node_attr = "value") %>%
  max(na.rm = TRUE)
```

R CONSOLE

```
[1] 23.1
```



TRAVERSALS

1. select node `1` with `select_nodes()`
2. traverse outward other nodes with `trav_out()`
3. traverse outward again with `trav_out()`

QUERY

with new selection (from traversals) get max value of the `value` node attribute with `max()`

GRAPH TRAVERSALS

traverse and query the graph

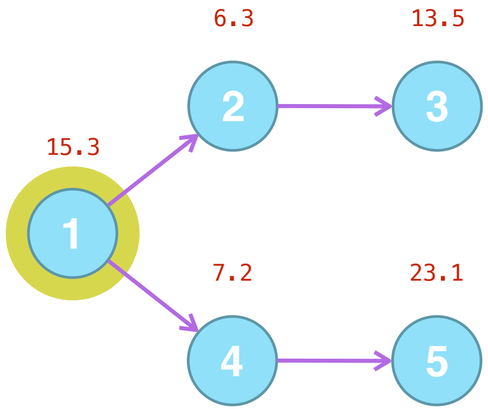
Traversals used with filtering conditions allow for more advanced queries:

— if we understand the graph model, we can use traversal motifs

— with adequate metadata (i.e., `type`, `rel`, and other data) we can target nodes

---

Let's traverse out from node 1 and include a traversal condition…



TRAVERSALS

- **1** select node 1
  2 traverse outward from 1 to other nodes where `value > 5`
  3 traverse outward again to nodes where `value > 15`

GRAPH TRAVERSALS

traverse and query the graph

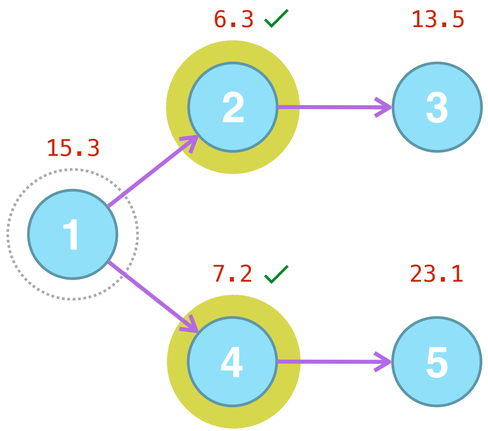
Traversals used with filtering conditions allow for more advanced queries:

— if we understand the graph model, we can use traversal motifs

— with adequate metadata (i.e., `type`, `rel`, and other data) we can target nodes

---

Let's traverse out from node 1 and include a traversal condition…



TRAVERSALS

1. select node 1
2. traverse outward from 1 to other nodes where `value > 5`
3. traverse outward again to nodes where `value > 15`

GRAPH TRAVERSALS

traverse and query the graph

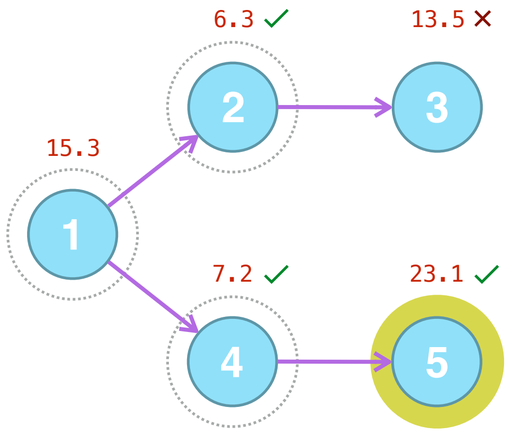
Traversals used with filtering conditions allow for more advanced queries:

— if we understand the graph model, we can use traversal motifs

— with adequate metadata (i.e., `type`, `rel`, and other data) we can target nodes

---

Let's traverse out from node 1 and include a traversal condition…



TRAVERSALS

1. select node 1
2. traverse outward from 1 to other nodes where `value > 5`
3. traverse outward again to nodes where `value > 15`

GRAPH TRAVERSALS

traverse and query the graph

Traversals used with filtering conditions allow for more advanced queries:

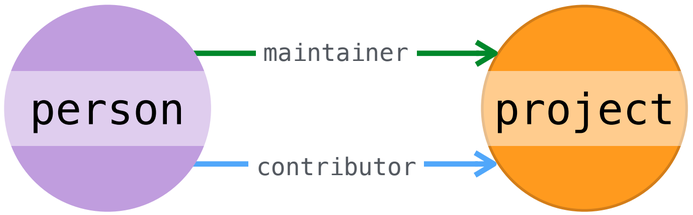
— if we understand the graph model, we can use traversal motifs

— with adequate metadata (i.e., `type`, `rel`, and other data) we can target nodes

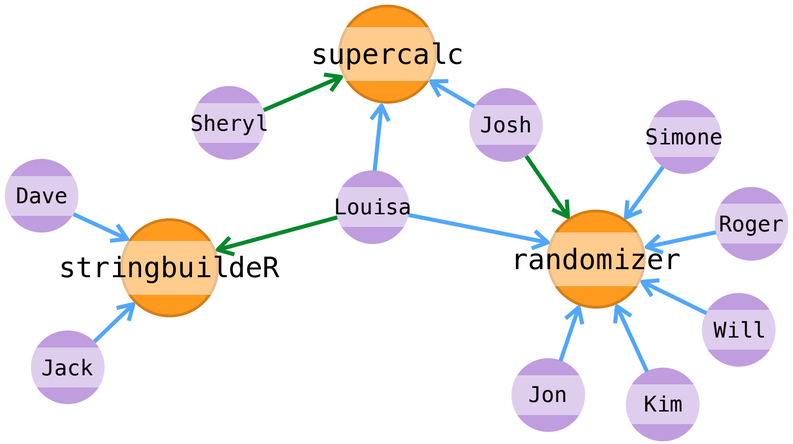
---

Using a (very simple) software repository graph…

GRAPH MODEL



THE GRAPH



GRAPH METADATA

nodes:

`type`, `label`, `name`, `age`, `email`, `join_date`, `follower_count`, `following_count`, `project`, `start_date`, `stars`, `language`

edges:

`rel`, `commits`

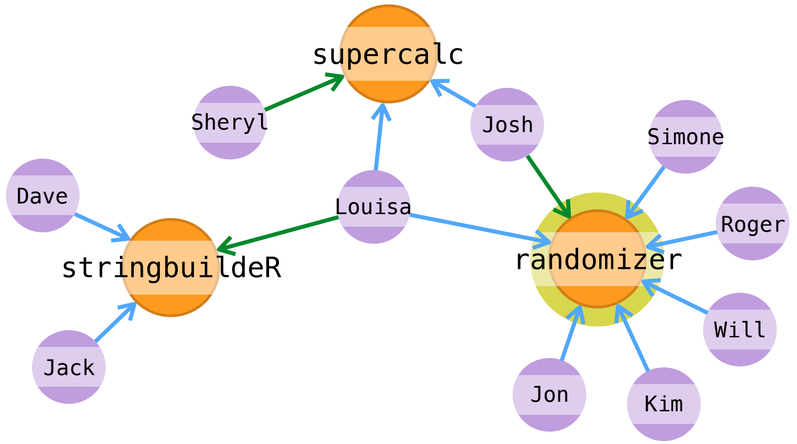
GRAPH TRAVERSALS

traverse and query the graph

R SCRIPT total commits to the randomizer project

```
# [1] Selecting node of the `randomizer` project
graph <- graph %>%
  select_nodes(conditions = "project == 'randomizer'")
```

THE GRAPH



GRAPH METADATA

nodes:

`type`, `label`, `name`, `age`, `email`, `join_date`, `follower_count`, `following_count`, `project`, `start_date`, `stars`, `language`

edges:

`rel`, `commits`

TRAVERSALS

1. select `randomizer` with `select_nodes()`
2. traverse to inward edges with `trav_in_edge()`

QUERY

with the new selection get sum of the `commits` edge attribute with `sum()`

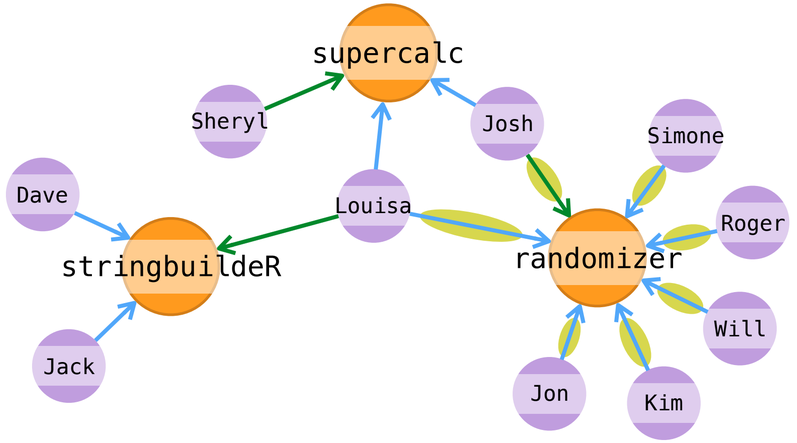
GRAPH TRAVERSALS

traverse and query the graph

R SCRIPT total commits to the randomizer project

```
# [1] Selecting node of the `randomizer` project
graph <- graph %>%
  select_nodes(conditions = "project == 'randomizer'")

# [2] Traverse to all inward edges
graph <- graph %>% trav_in_edge()
```

THE GRAPH



GRAPH METADATA

nodes:

`type`, `label`, `name`, `age`, `email`, `join_date`, `follower_count`, `following_count`, `project`, `start_date`, `stars`, `language`

edges:

`rel`, `commits`

TRAVERSALS

1. select `randomizer` with `select_nodes()`
2. traverse to inward edges with `trav_in_edge()`

QUERY

with the new selection get sum of the `commits` edge attribute with `sum()`

GRAPH TRAVERSALS

traverse and query the graph

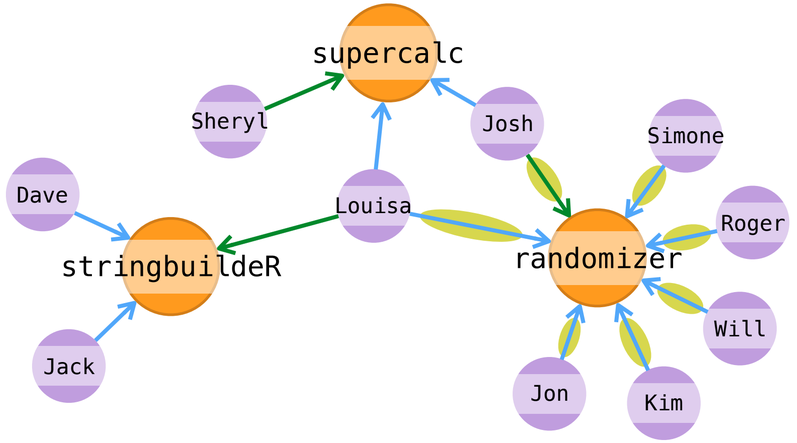
R SCRIPT total commits to the randomizer project

```
# [1] Selecting node of the `randomizer` project
graph <- graph %>%
  select_nodes(conditions = "project == 'randomizer'")

# [2] Traverse to all inward edges
graph <- graph %>% trav_in_edge()

# Query: Get sum of `commits` from traversed edges
graph %>%
  get_edge_attrs_ws(edge_attr = "commits") %>%
  sum()
```

R CONSOLE

```
[1] 3117
```

THE GRAPH



GRAPH METADATA

nodes:

`type`, `label`, `name`, `age`, `email`, `join_date`, `follower_count`, `following_count`, `project`, `start_date`, `stars`, `language`

edges:

`rel`, `commits`

TRAVERSALS

1. select `randomizer` with `select_nodes()`
2. traverse to inward edges with `trav_in_edge()`

QUERY

with the new selection get sum of the `commits` edge attribute with `sum()`

GRAPH TRAVERSALS

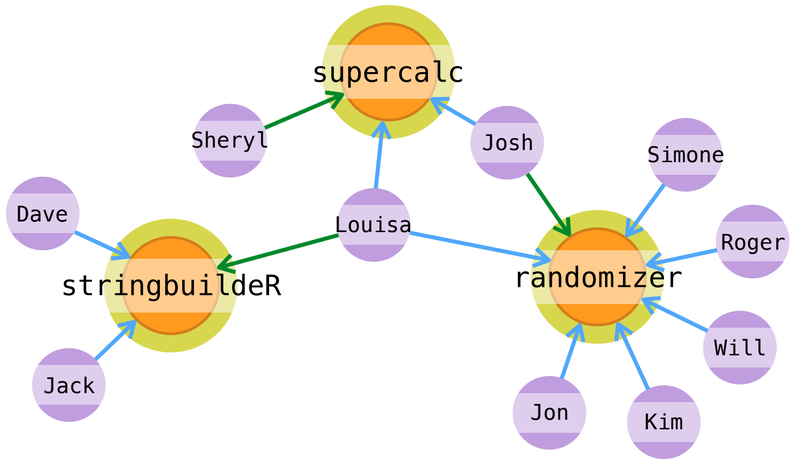
traverse and query the graph

R SCRIPT

average maintainer follower count

```
# [1] Selecting nodes of all software projects
graph <- graph %>%
  select_nodes(conditions = "type == 'project'")
```

THE GRAPH



GRAPH METADATA

nodes:

`type`, `label`, `name`, `age`, `email`, `join_date`, `follower_count`, `following_count`, `project`, `start_date`, `stars`, `language`

edges:

`rel`, `commits`

TRAVERSALS

1. select all software projects with `select_nodes()`
2. traverse to inward edges with `trav_in_edge()`
3. traverse to nodes via out-edges with `trav_out_node()`

QUERY

get the mean `follower_count` with `mean()`

GRAPH TRAVERSALS

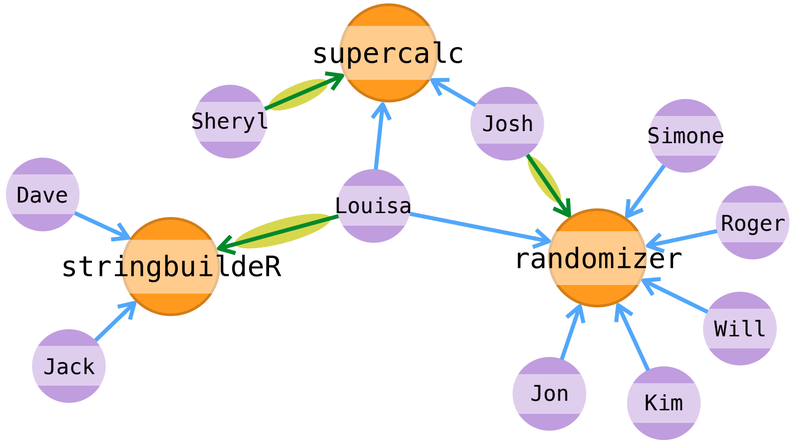
traverse and query the graph

R SCRIPT

average maintainer follower count

```
# [1] Selecting nodes of all software projects
graph <- graph %>%
  select_nodes(conditions = "type == 'project'")

# [2] Traverse to all inward edges
graph <- graph %>%
  trav_in_edge(conditions = "rel == 'maintainer'")
```

THE GRAPH



GRAPH METADATA

nodes:

`type`, `label`, `name`, `age`, `email`, `join_date`, `follower_count`, `following_count`, `project`, `start_date`, `stars`, `language`

edges:

`rel`, `commits`

TRAVERSALS

1. select all software projects with `select_nodes()`
2. traverse to inward edges with `trav_in_edge()`
3. traverse to nodes via out-edges with `trav_out_node()`

QUERY

get the mean `follower_count` with `mean()`

GRAPH TRAVERSALS

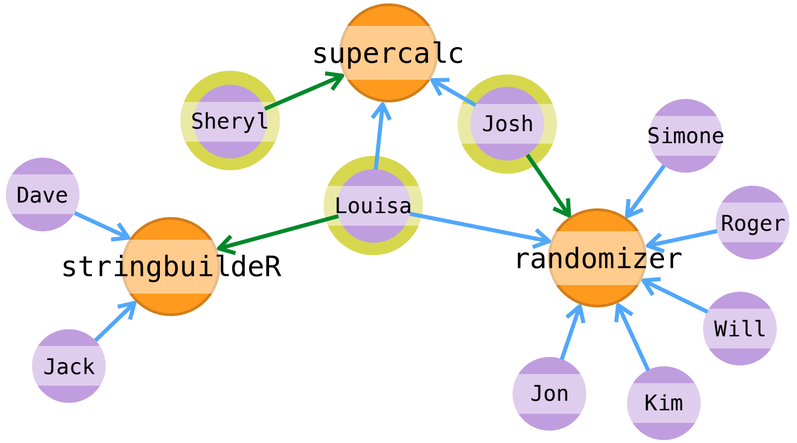
traverse and query the graph

R SCRIPT

average maintainer follower count

```
# [1] Selecting nodes of all software projects
graph <- graph %>%
  select_nodes(conditions = "type == 'project'")

# [2] Traverse to all inward edges
graph <- graph %>%
  trav_in_edge(conditions = "rel == 'maintainer'")

# [3] Traverse to all nodes via their outbound edges
graph <- graph %>% trav_out_node()
```

THE GRAPH



GRAPH METADATA

nodes:

`type`, `label`, `name`, `age`, `email`, `join_date`, `follower_count`, `following_count`, `project`, `start_date`, `stars`, `language`

edges:

`rel`, `commits`

TRAVERSALS

1. select all software projects with `select_nodes()`
2. traverse to inward edges with `trav_in_edge()`
3. traverse to nodes via out-edges with `trav_out_node()`

QUERY

get the mean `follower_count` with `mean()`

GRAPH TRAVERSALS

traverse and query the graph

R SCRIPT

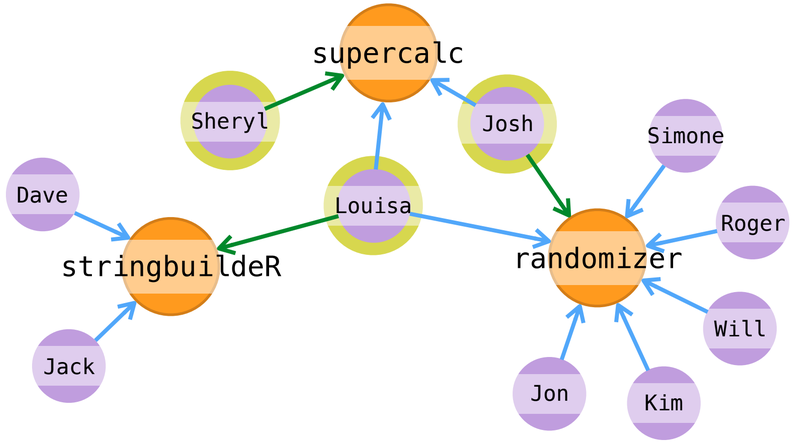
average maintainer follower count

```
# [1] Selecting nodes of all software projects
graph <- graph %>%
  select_nodes(conditions = “type == ‘project’”)

# [2] Traverse to all inward edges
graph <- graph %>%
  trav_in_edge(conditions = “rel == ‘maintainer’”)

# [3] Traverse to all nodes via their outbound edges
graph <- graph %>% trav_out_node()

# Query: Get mean `follower_count` from the nodes
graph %>%
  get_node_attrs_ws(node_attr = “follower_count”) %>%
  mean()
```

R CONSOLE

```
[1] 281
```

THE GRAPH



GRAPH METADATA

nodes:

`type`, `label`, `name`, `age`, `email`, `join_date`, `follower_count`, `following_count`, `project`, `start_date`, `stars`, `language`

edges:

`rel`, `commits`

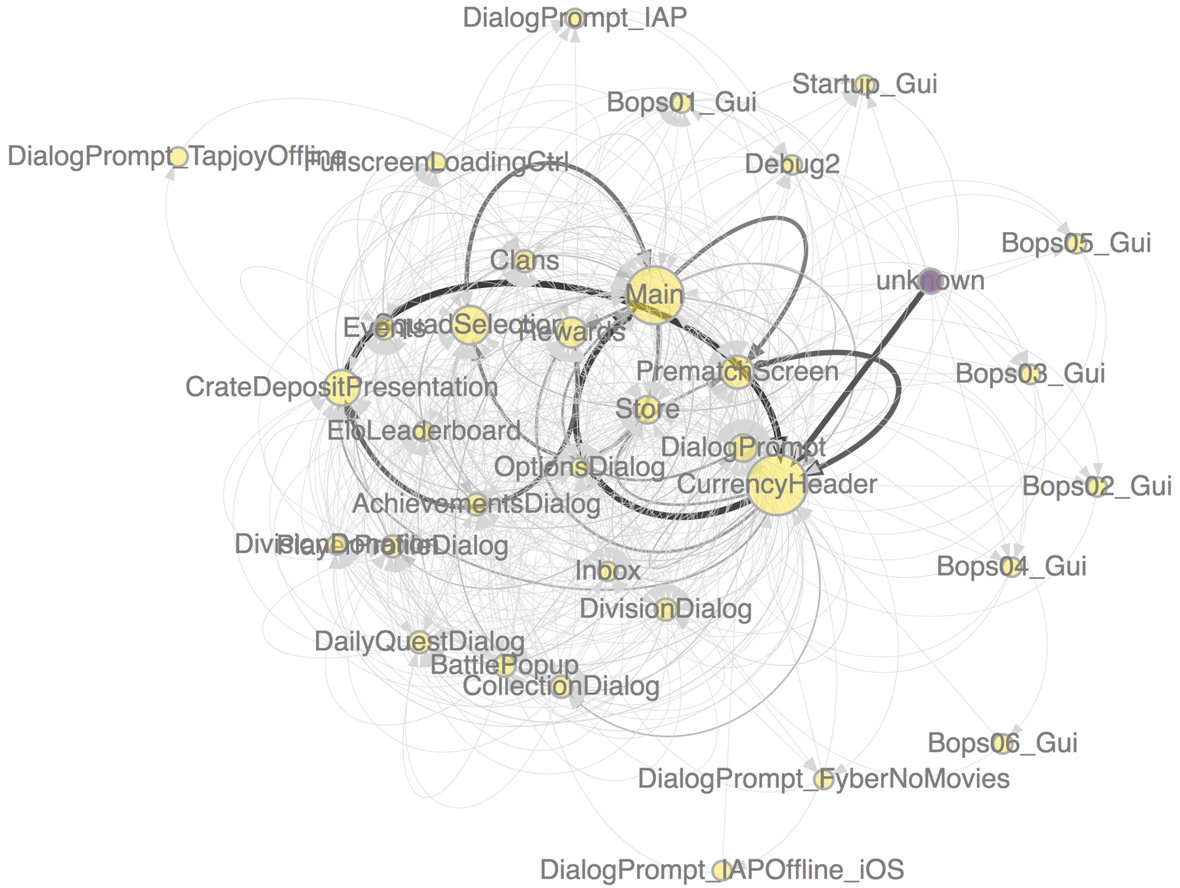
TRAVERSALS

1. select all software projects with `select_nodes()`
2. traverse to inward edges with `trav_in_edge()`
3. traverse to nodes via out-edges with `trav_out_node()`

QUERY

get the mean `follower_count` with `mean()`

Using actual game data, we've used graph analytics to better understand UI flow patterns.



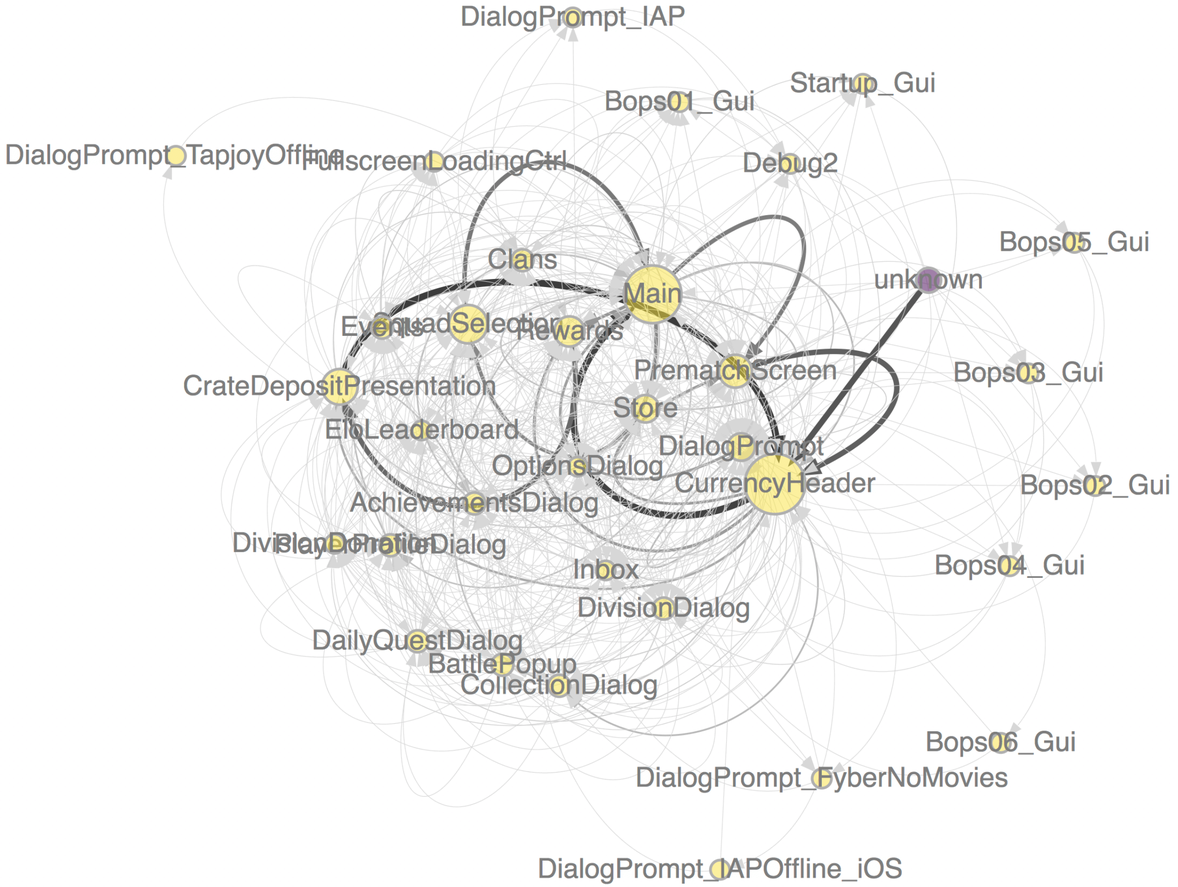
ALL UI TRANSITIONS

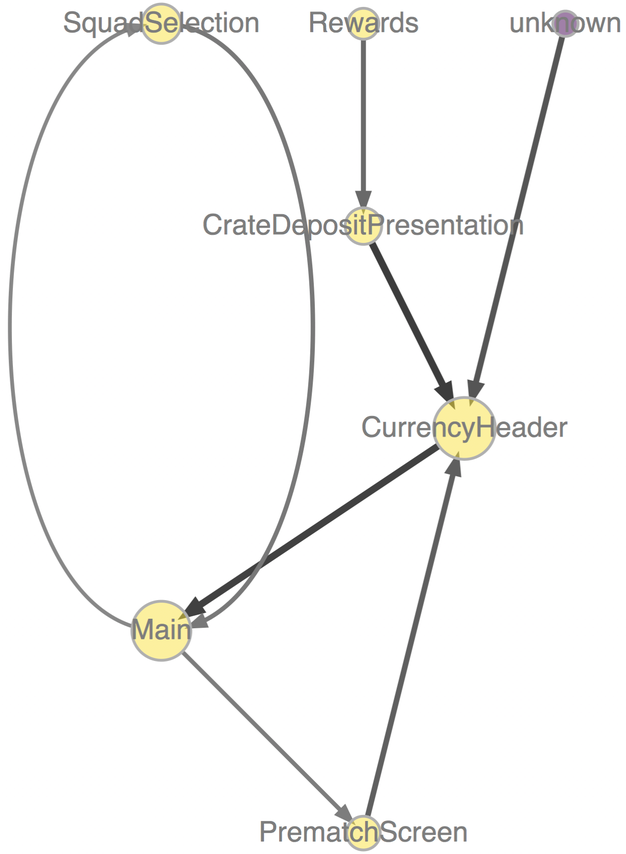
— node size is count of users in that view

— line thickness is transition count for that path

Using actual game data, we've used graph analytics to better understand UI flow patterns.

```
graph %>%
  select_edges("weight > 500000") %>%
  create_subgraph_ws() %>%
  render_graph(layout = "tree")
```





ALL UI TRANSITIONS

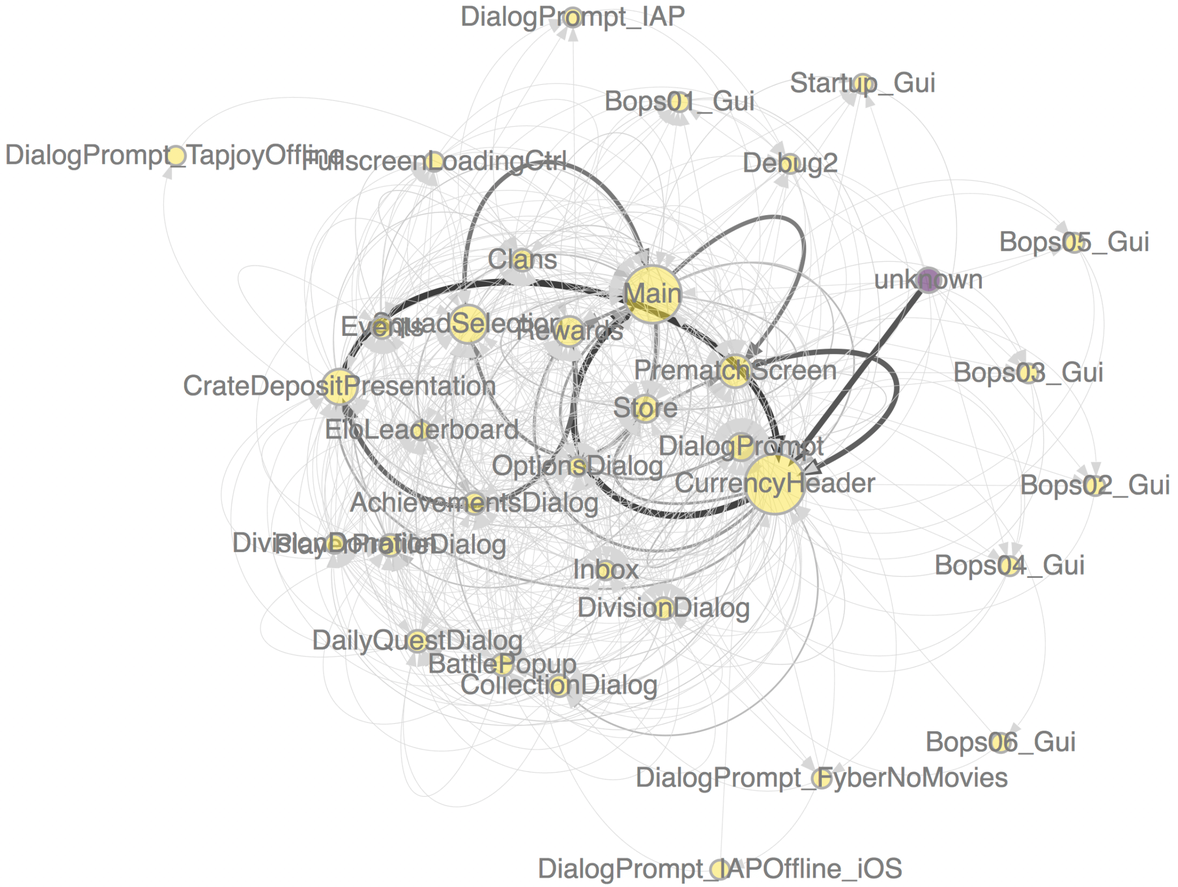
— node size is count of users in that view
— line thickness is transition count for that path

MAJOR UI TRANSITIONS

— this is a subgraph retaining only edges (and the associated nodes) of highest weight

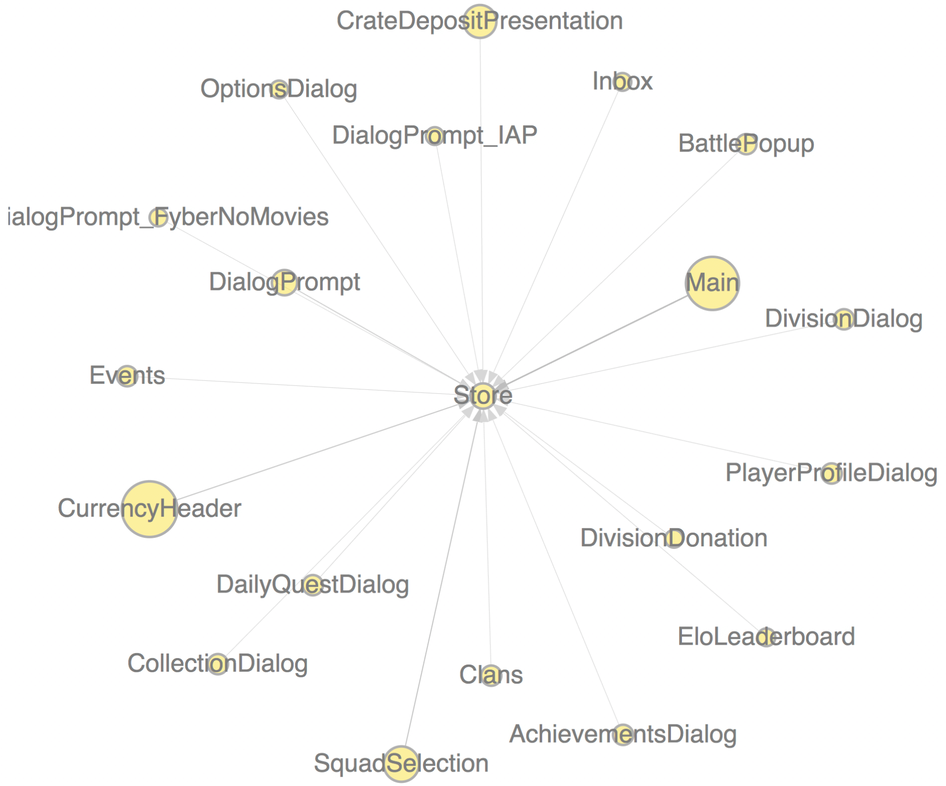
Using actual game data, we've used graph analytics to better understand UI flow patterns.

```
graph %>%
  select_nodes("label == 'Store'") %>%
  trav_in_edge("weight > 100") %>%
  create_subgraph_ws() %>%
  render_graph(layout = "kr")
```



ALL UI TRANSITIONS

— node size is count of users in that view
— line thickness is transition count for that path



TRANSITIONS TO STORE

— this is a subgraph showing the main UI pathways to the game's store screen

That’s it for now. I hope you found this useful.

If you’d like more information, visit the **DiagrammeR** Github page:

https://github.com/rich-iannone/DiagrammeR

If you’d like to talk more about this or contribute ideas/code, let’s talk!

Ask me questions…