R User Group Singapore Meetup

Network Analysis with tidygraph and ggraph in R

Naman Agrawal
National University of Singapore

2024-11-29

Table of Contents

Introduction to Network Analysis and Graphs

What is Network Analysis?

Network analysis is a method used to study and understand the relationships and interactions between entities within a system.

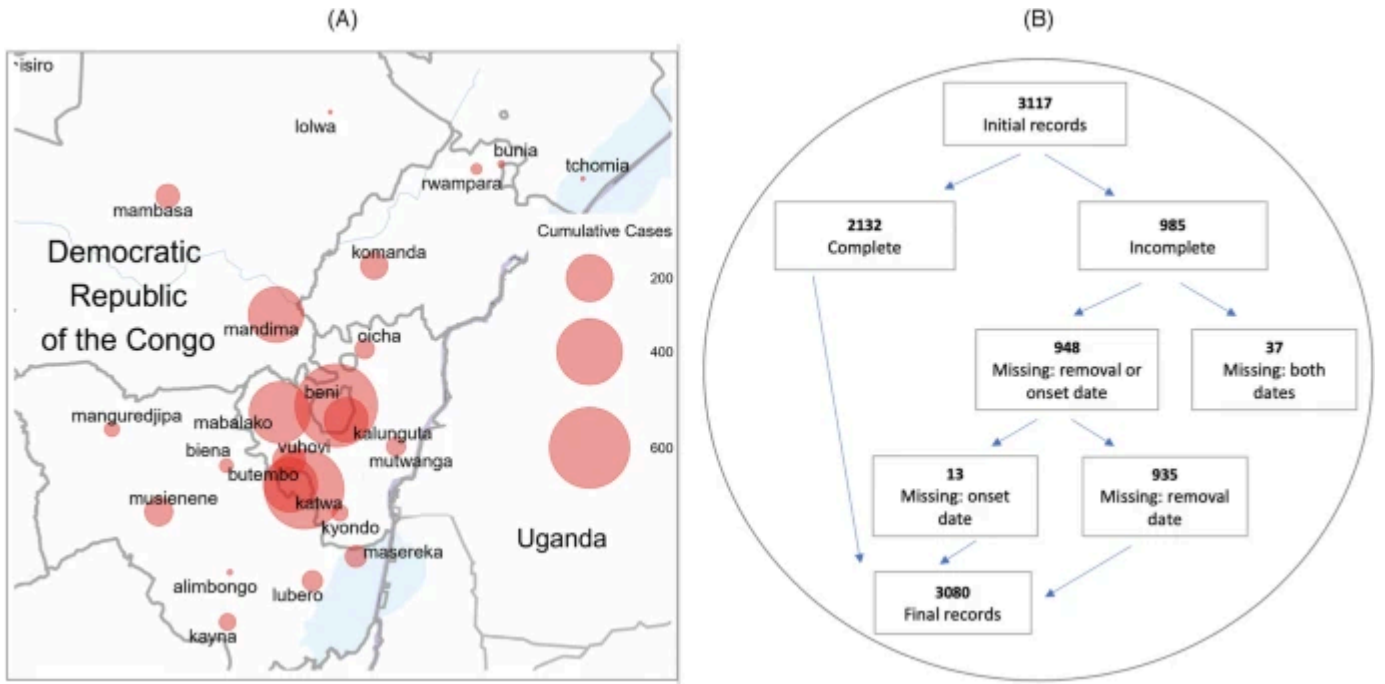
- **Examples:**
  - Understanding social networks (e.g., identifying influencers on social media).
  - Analyzing biological systems such as protein interactions.

How Network Analysis Differs from Tabular Data Analysis

Aspect	Tabular Data	Network Data
Focus	Observations	Relationships
Data Structure	Rows and Columns	Nodes and Edges
Visualization	Scatter plots, histograms	Graph diagrams (node-link)

Importance of Network Analysis?

- Helps uncover hidden patterns.
- Optimizes decision-making in complex systems.
- Identifies influential entities and critical connections.



https://www.nature.com/articles/s41598-022-09564-4

Introduction to Graphs

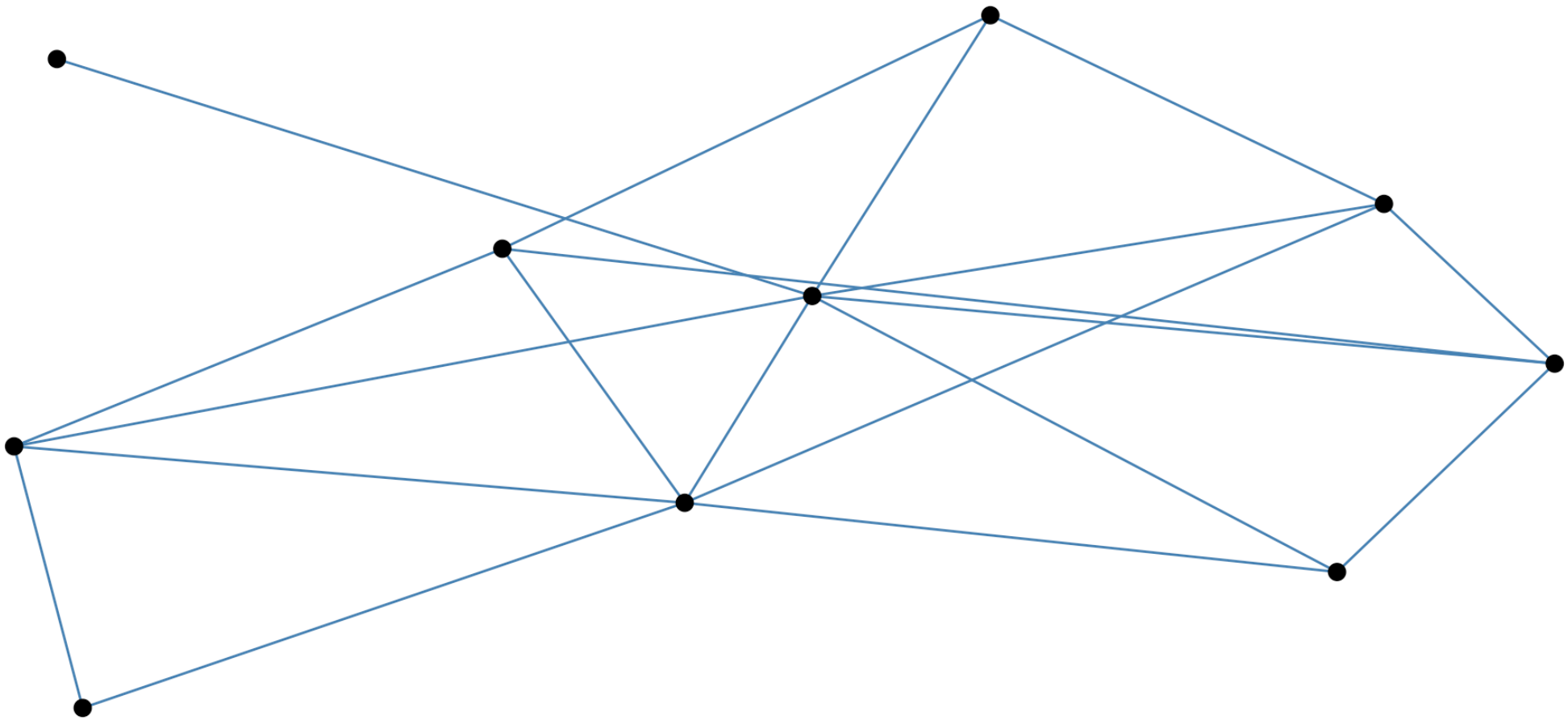
A graph is a mathematical structure used to model pairwise relationships between entities.

Components of a Graph

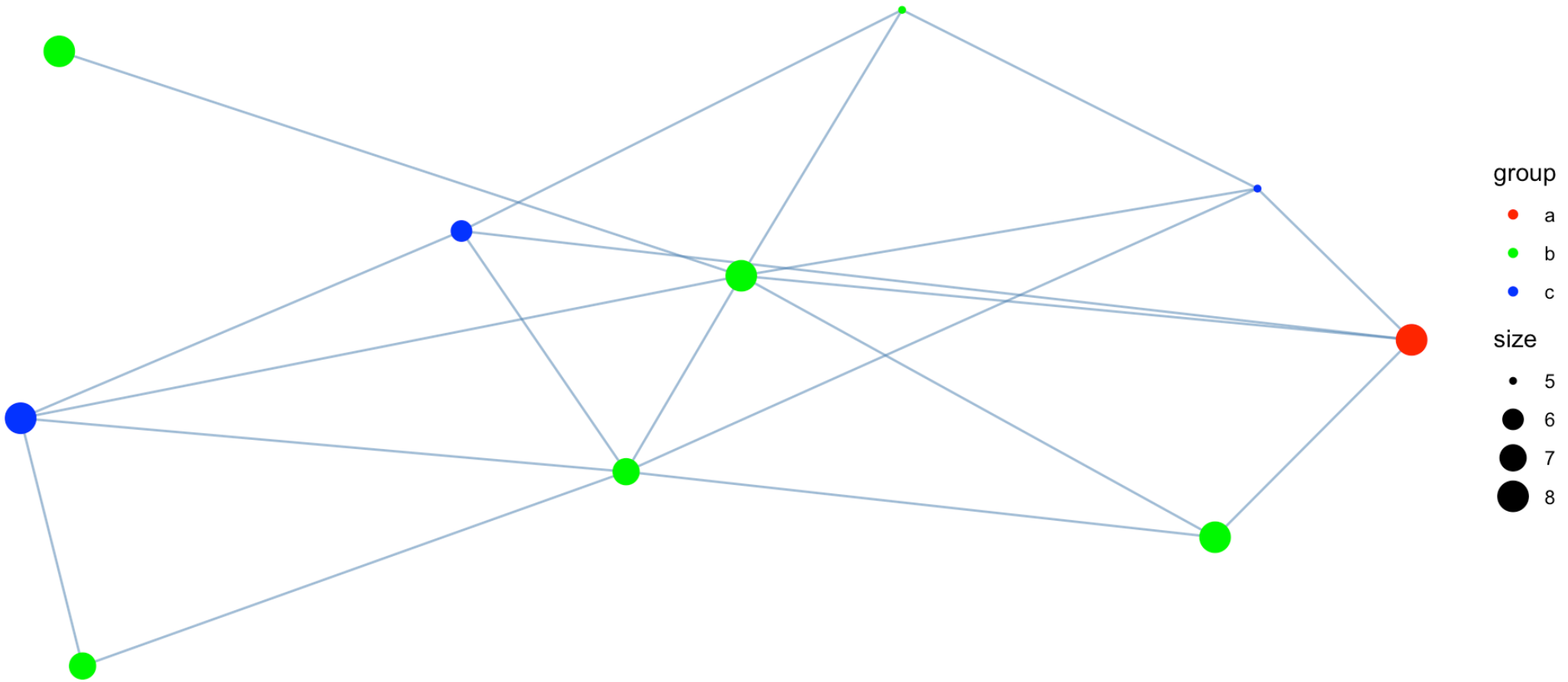
1. **Nodes (Vertices)**: Represent entities.
2. **Edges (Links)**: Represent relationships between nodes.

A graph may also have **attributes**, which are dditional data about nodes or edges (e.g., weights, labels).

Introduction to Graphs



Introduction to Graphs



Introduction to Graphs

Graph Types:

- **Directed**: Relationships have a direction (e.g., A → B).
- **Undirected**: Relationships are bidirectional.
- **Weighted**: Edges have weights (e.g., cost, distance).
- **Unweighted**: Edges do not carry weights.

Ways to Represent Graphs

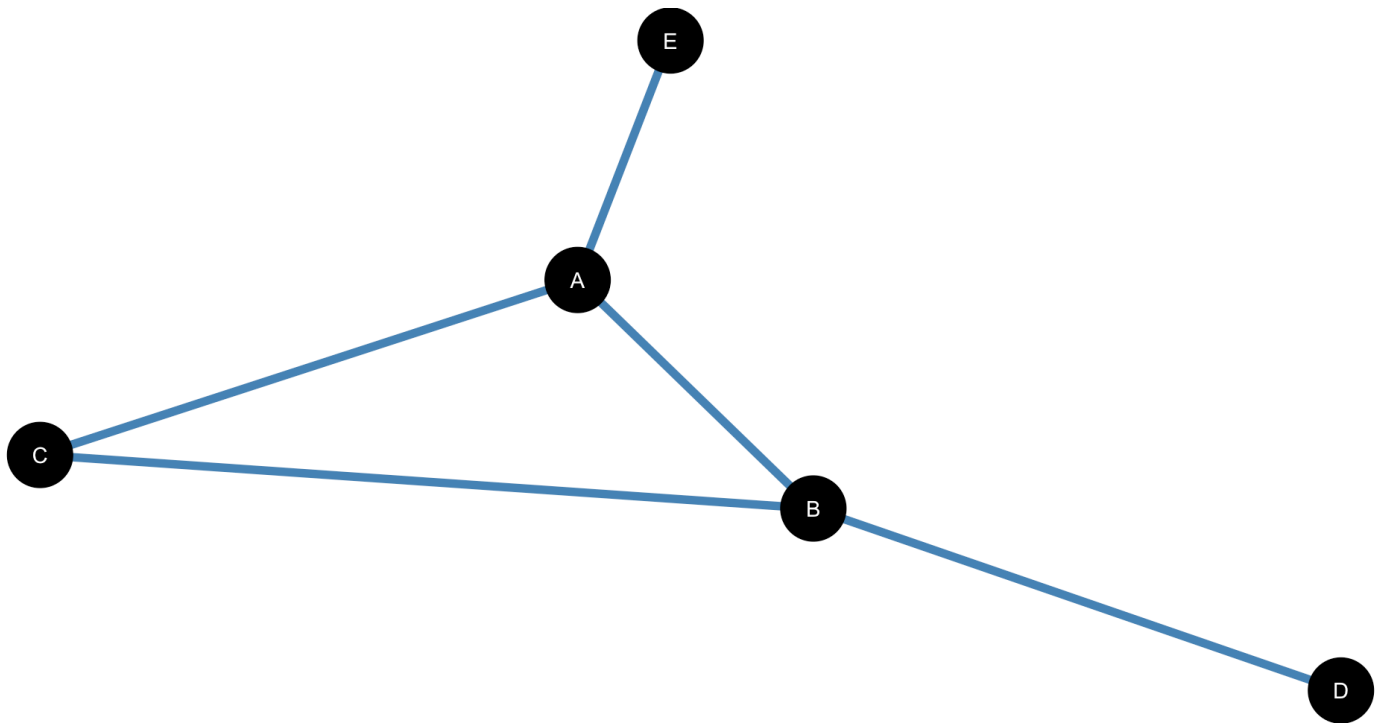
Graph Representations

1. **Edge List**: A list of node pairs connected by edges.
2. **Adjacency Matrix**: Rows and columns represent nodes; cell values indicate edges.
3. **Adjacency List**: A collection of lists where each list corresponds to a node and contains the nodes it is connected to via edges.

and many more …

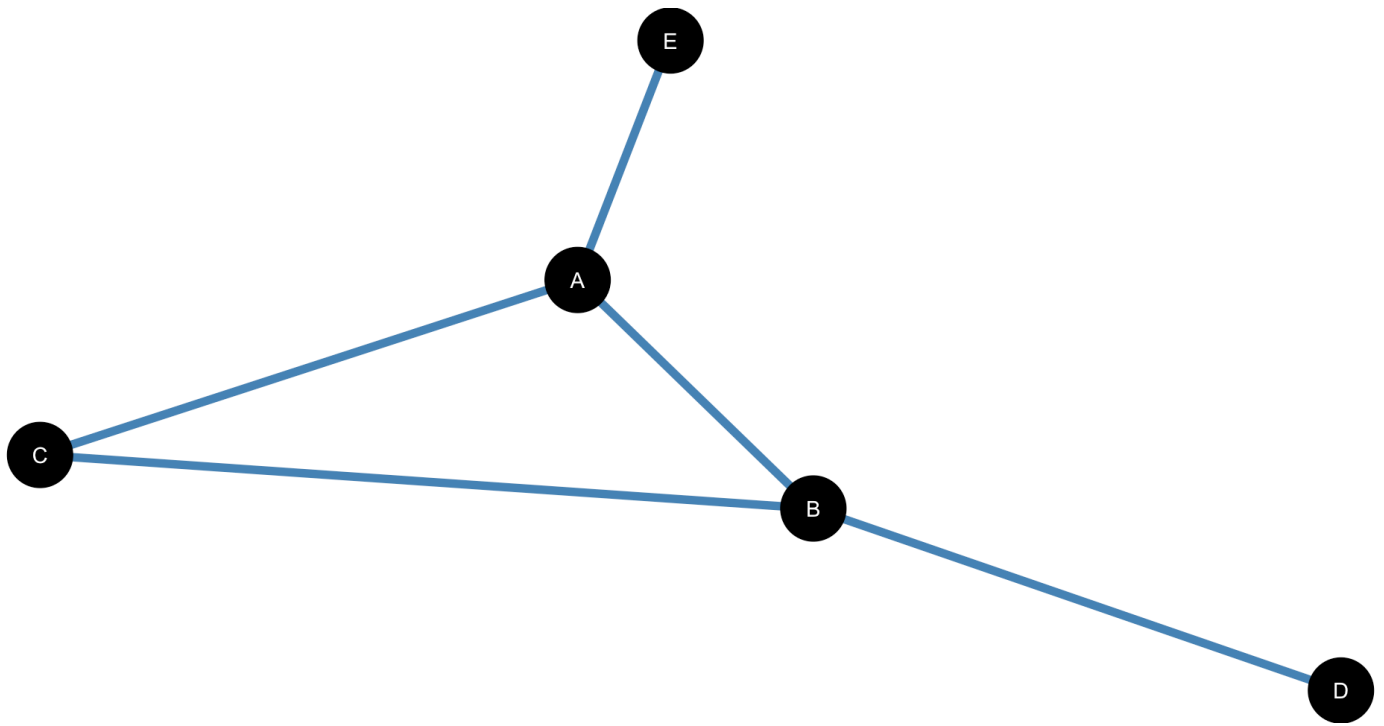
Representations: Edge List

```
1  (A, B)
2  (A, C)
3  (B, D)
4  (B, C)
5  (A, E)
```



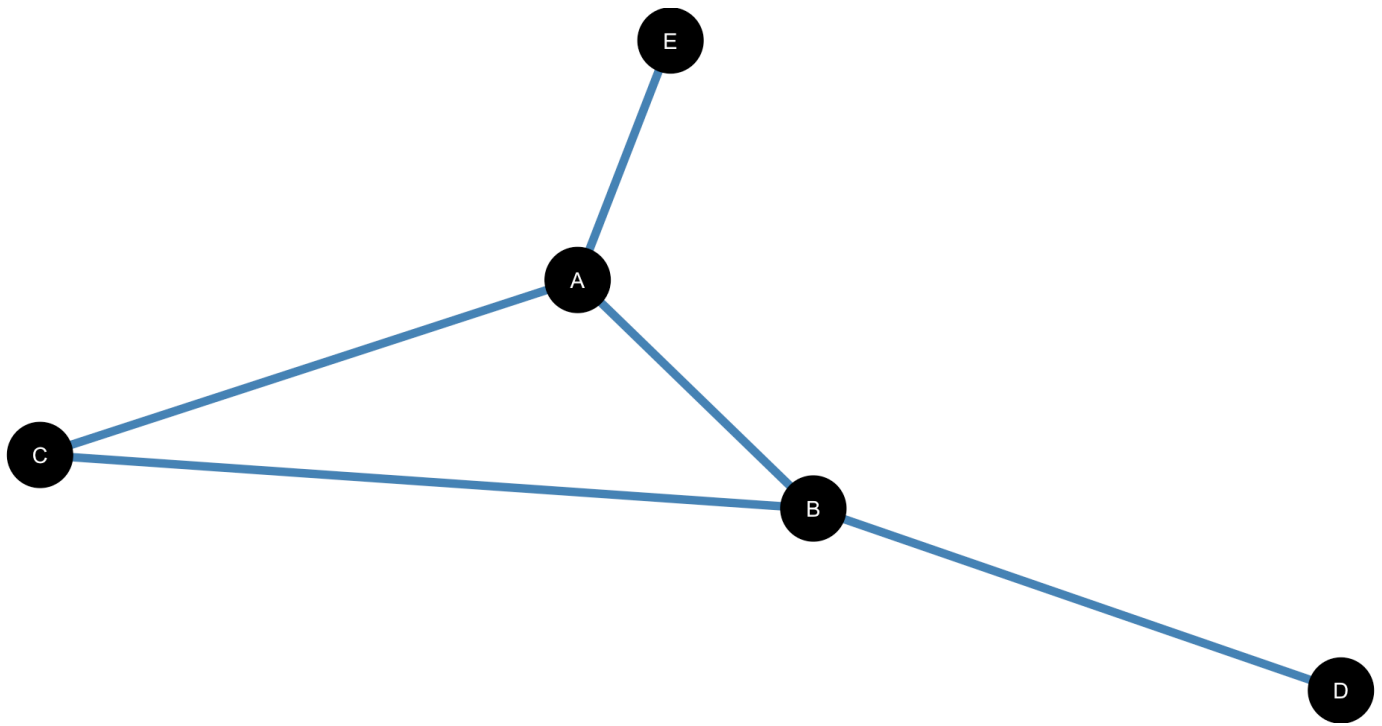
Representations: Adjacency Matrix

	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	1	0
C	1	1	0	0	0
D	0	1	0	0	0
E	1	0	0	0	0



Representations: Adjacency List

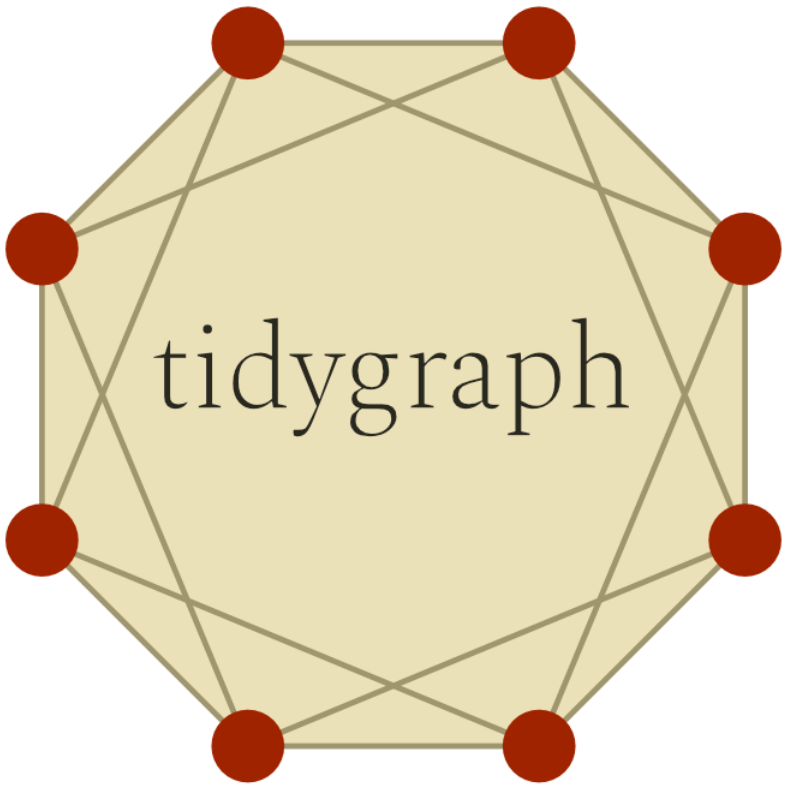
```
1  A: B, C, E
2  B: A, D, C
3  C: A, B
4  D: B
5  E: A
```



Basics of tidygraph

What are tidygraph and ggraph?

- tidygraph is used for graph manipulation (built upon igraph).
- ggraph is used for visualizing graphs created or manipulated using tidygraph.



https://www.data-imaginist.com/

How tidygraph works

tidygraph stores graph data using two tibbles:

- Nodes tibble: Contains node attributes.
- Edges tibble: Contains edge attributes.

```r
library(tidygraph)
# Create a graph
graph <- tbl_graph(
  nodes = tibble(name = c("A", "B", "C")),
  edges = tibble(from = c(1, 2, 3), to = c(2, 3, 1))
)
print(graph)
```

```
# A tbl_graph: 3 nodes and 3 edges
#
# A directed simple graph with 1 component
#
# Node Data: 3 × 1 (active)
  name
  <chr>
1 A
2 B
3 C
#
# Edge Data: 3 × 2
   from    to
  <int> <int>
1     1     2
2     2     3
3     3     1
```

How to Create a Graph?

Two ways:

1. `tbl_graph(nodes = ..., edges = ...)`: used when you have explicit data frames for nodes and edges.

- Nodes: A data frame where each row represents a node, and columns may include attributes (e.g., name, type).
- Edges: A data frame where each row represents a connection between nodes, typically with from and to columns to indicate relationships.

```
library(tidygraph)
nodes <- data.frame(name = c("A", "B", "C"), type = c("1", "1", "2"))
edges <- data.frame(from = c(1, 2), to = c(2, 3))
graph <- tbl_graph(nodes = nodes, edges = edges)
```

How to Create a Graph?

Two ways:

2. `as_tbl_graph()`: from common graph structures like igraph objects, adjacency matrices, or edge lists. Very general, works with a lot of data types like data frames or matrices:

```
library(tidygraph)
adj_matrix <- matrix(
  c(0, 1, 0,
    1, 0, 1,
    0, 1, 0), nrow = 3, byrow = TRUE,
  dimnames = list(c("A", "B", "C"), c("A", "B", "C"))
)
graph <- as_tbl_graph(adj_matrix)
```

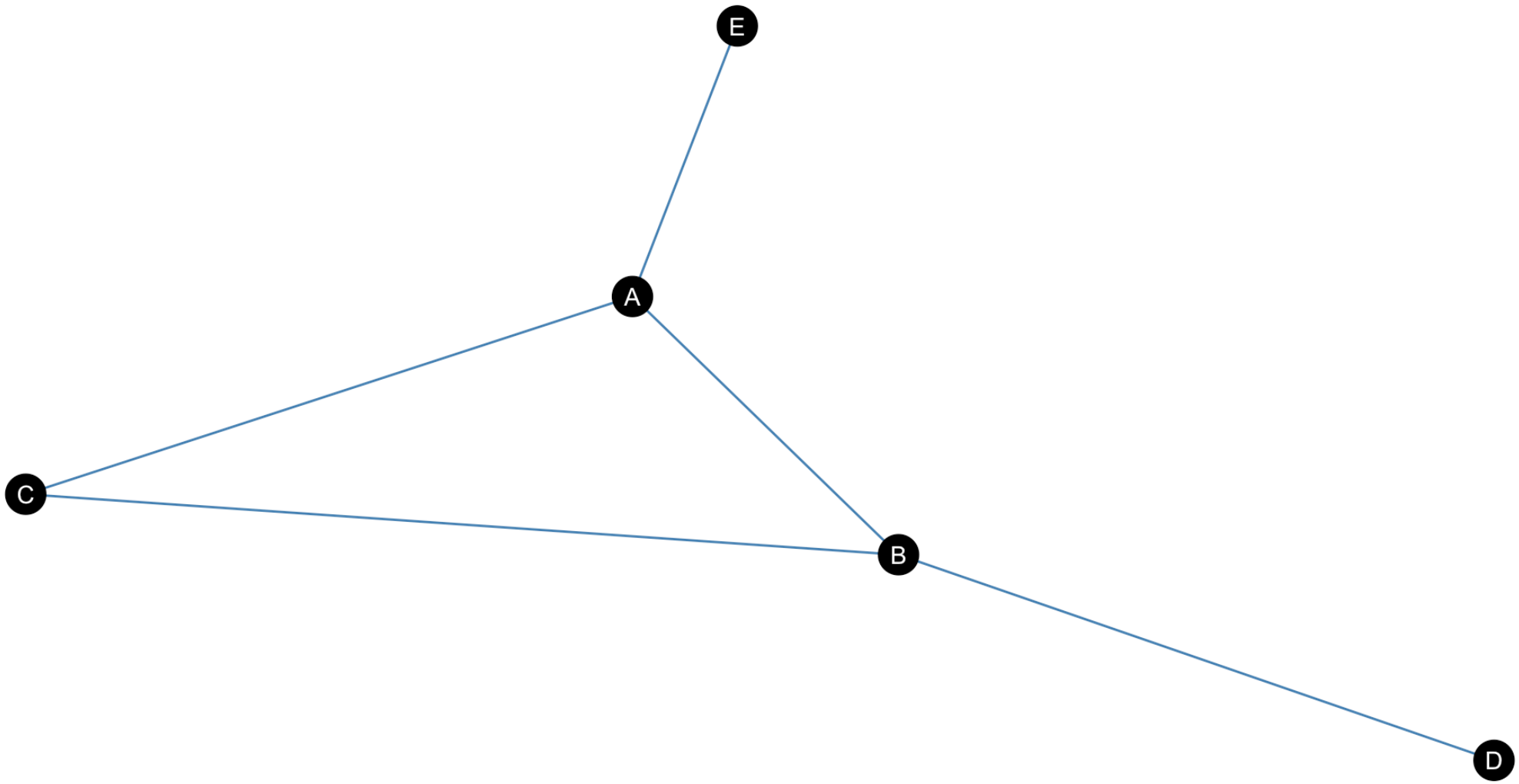
Key Operations in tidygraph

- `activate()`: Switch between node and edge context.
- `filter()`: Subset nodes or edges.
- `mutate()`: Add or modify attributes.

```
library(tidygraph)
edges <- data.frame(
  from = c("A", "A", "B", "B", "E"),
  to = c("B", "C", "D", "C", "A")
)
graph <- as_tbl_graph(edges, directed = FALSE)
print(graph)
```

```
# A tbl_graph: 5 nodes and 5 edges
#
# An undirected simple graph with 1 component
#
# Node Data: 5 × 1 (active)
  name
  <chr>
1 A
2 B
3 E
4 C
5 D
#
# Edge Data: 5 × 2
   from    to
  <int> <int>
1     1     2
2     1     4
3     2     5
# ℹ 2 more rows
```

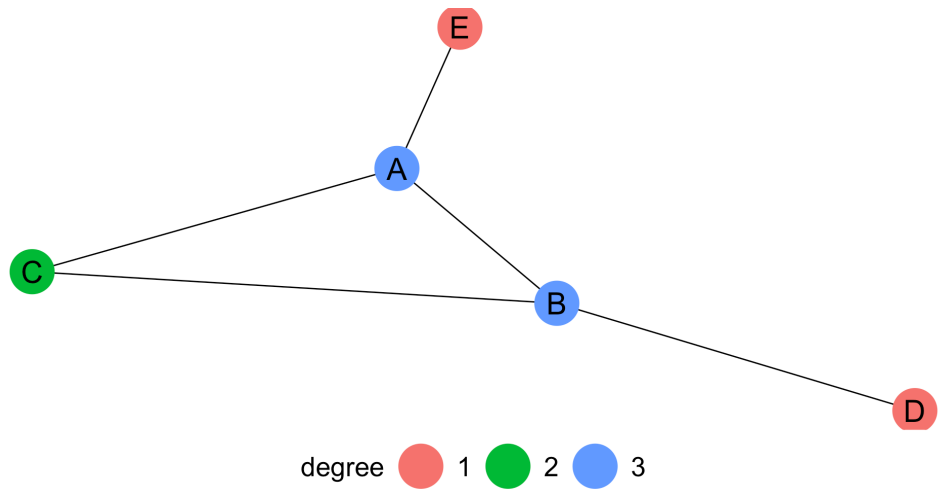
Key Operations in tidygraph



Key Operations in tidygraph

```r
# Activate nodes
g = graph %>%
  activate(nodes) %>%
  mutate(degree = as.factor(centrality_degree())) 
print(g)
```

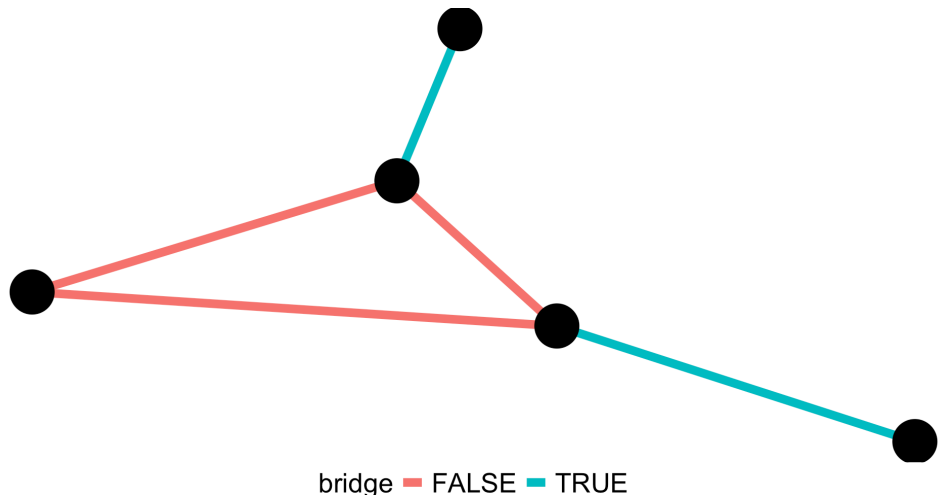
```
# A tbl_graph: 5 nodes and 5 edges
#
# An undirected simple graph with 1 component
#
# Node Data: 5 × 2 (active)
  name  degree
  <chr> <fct>
1 A     3
2 B     3
3 E     1
4 C     2
5 D     1
#
# Edge Data: 5 × 2
   from    to
  <int> <int>
1     1     2
2     1     4
3     2     5
# ℹ 2 more rows
```



Key Operations in tidygraph

```r
# Add a new attribute to edges
g = graph %>%
  activate(edges) %>%
  mutate(bridge = tidygraph::edge_is_bridge())
print(g)
```

```
# A tbl_graph: 5 nodes and 5 edges
#
# An undirected simple graph with 1 component
#
# Edge Data: 5 × 3 (active)
   from    to bridge
  <int> <int> <lgl>
1     1     2 FALSE
2     1     4 FALSE
3     2     5 TRUE
4     2     4 FALSE
5     1     3 TRUE
#
# Node Data: 5 × 1
  name
  <chr>
1 A
2 B
3 E
# ℹ 2 more rows
```



Key Operations in tidygraph

```
# Filter nodes with specific names
graph %>%
  activate(nodes) %>%
  filter(name != "B")
```

```
# A tbl_graph: 4 nodes and 2 edges
#
# An unrooted forest with 2 trees
#
# Node Data: 4 × 1 (active)
  name
  <chr>
1 A
2 E
3 C
4 D
#
# Edge Data: 2 × 2
   from    to
  <int> <int>
1     1     3
2     1     2
```

```
# Filter edges
graph %>%
  activate(edges) %>%
  filter(from == 1)
```

```
# A tbl_graph: 5 nodes and 3 edges
#
# An unrooted forest with 2 trees
#
# Edge Data: 3 × 2 (active)
   from    to
  <int> <int>
1     1     2
2     1     4
3     1     3
#
# Node Data: 5 × 1
  name
  <chr>
1 A
2 B
3 E
# ℹ 2 more rows
```

Visualization with ggraph

What is ggraph?

ggraph provides an easy way to visualize graphs using intuitive functions and multiple layout options.

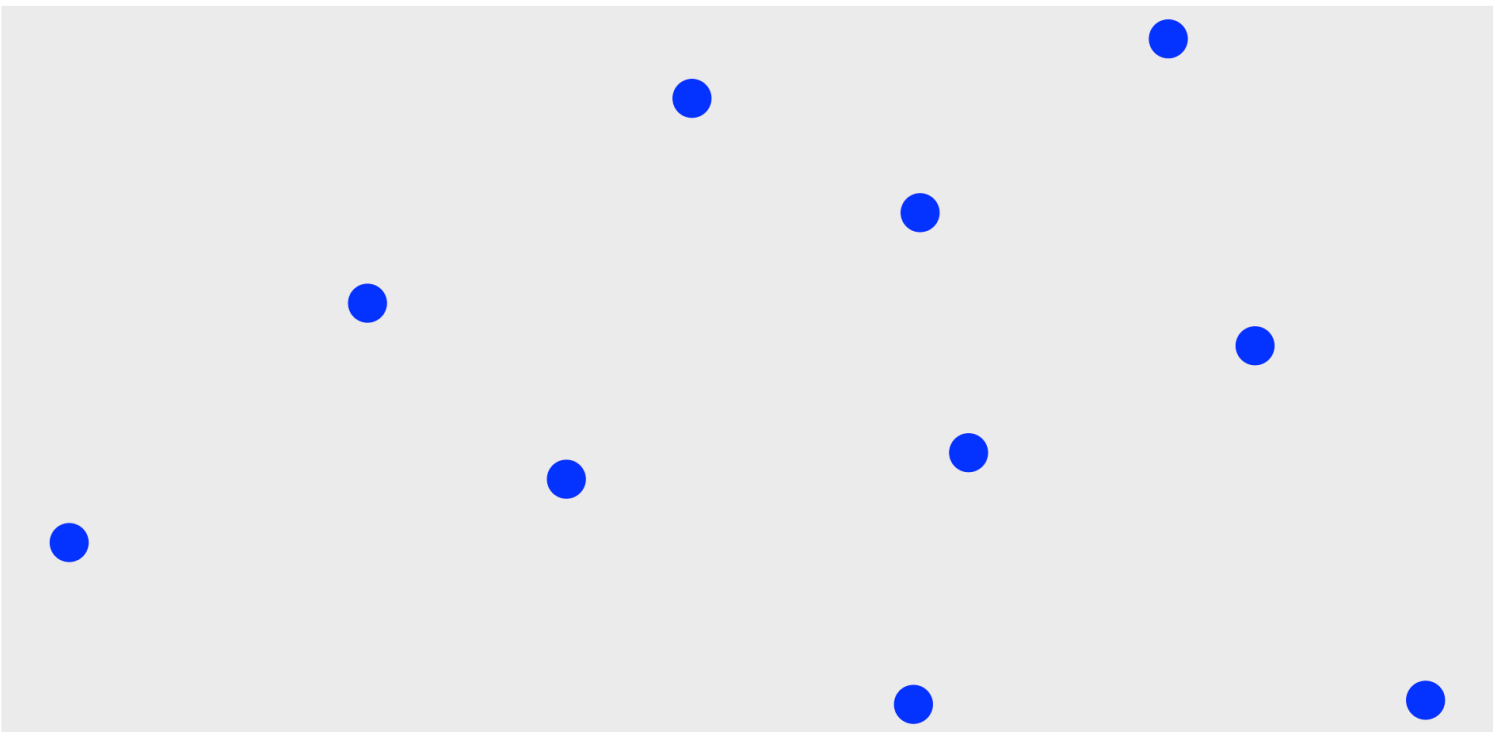
- Supports layouts like tree, circular, and force-directed.
- Seamlessly integrates with ggplot2.

Overview of Node Layers

- Node layers in ggraph are used to represent the entities of a graph visually.
- Common aesthetics include size (importance of nodes), color (group membership or categories), and shape (node type).
- The `geom_node_point()` function is the primary method to plot nodes.

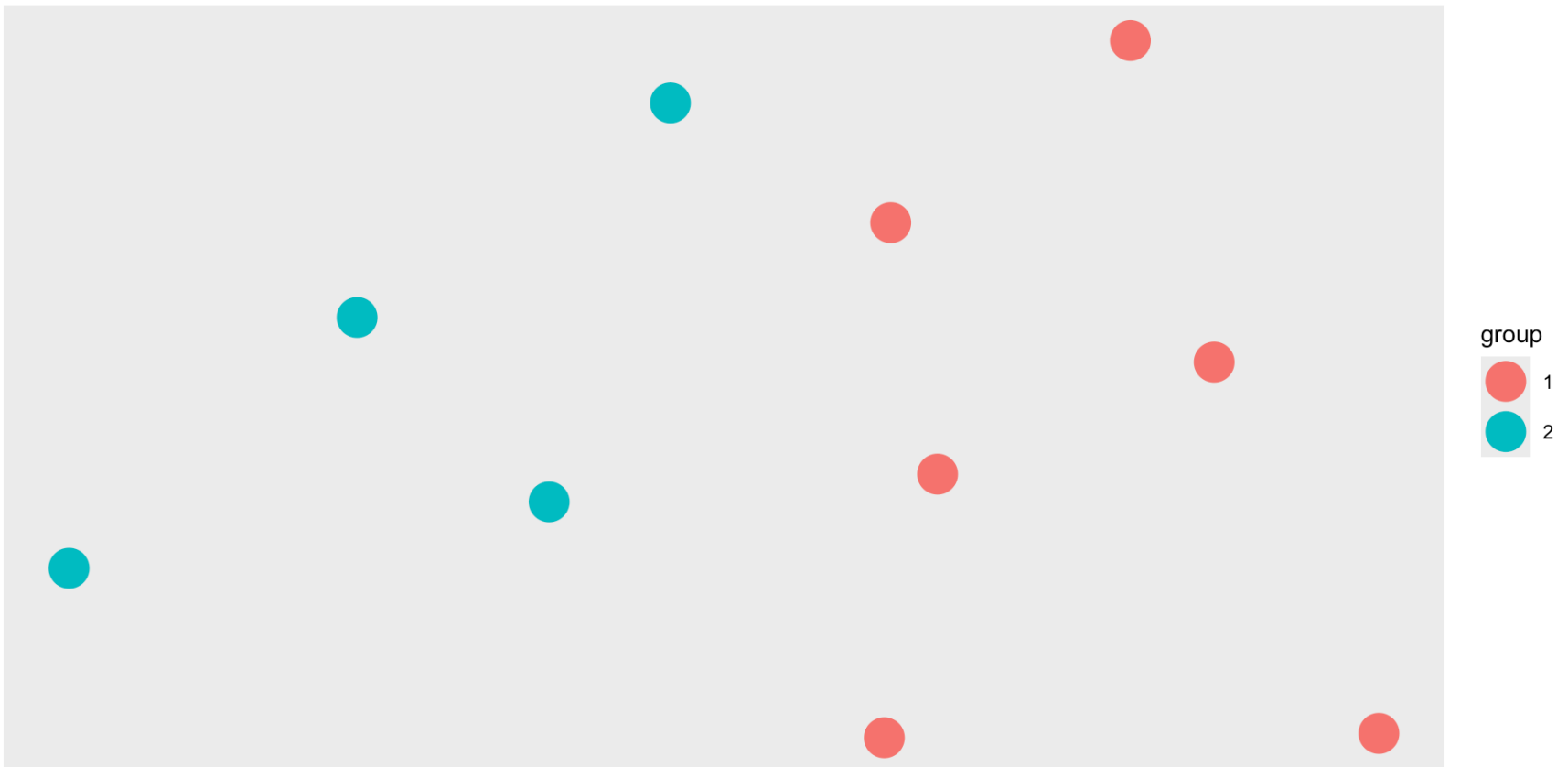
geom_node_point

```
1  library(ggraph)
2  set.seed(7)
3  graph <- sample_gnp(n = 10, p = 0.3)  # Random graph
4  graph = as_tbl_graph(graph, directed = FALSE)
5  ggraph(graph) +
6    geom_node_point(size = 8, color = 'blue')
```



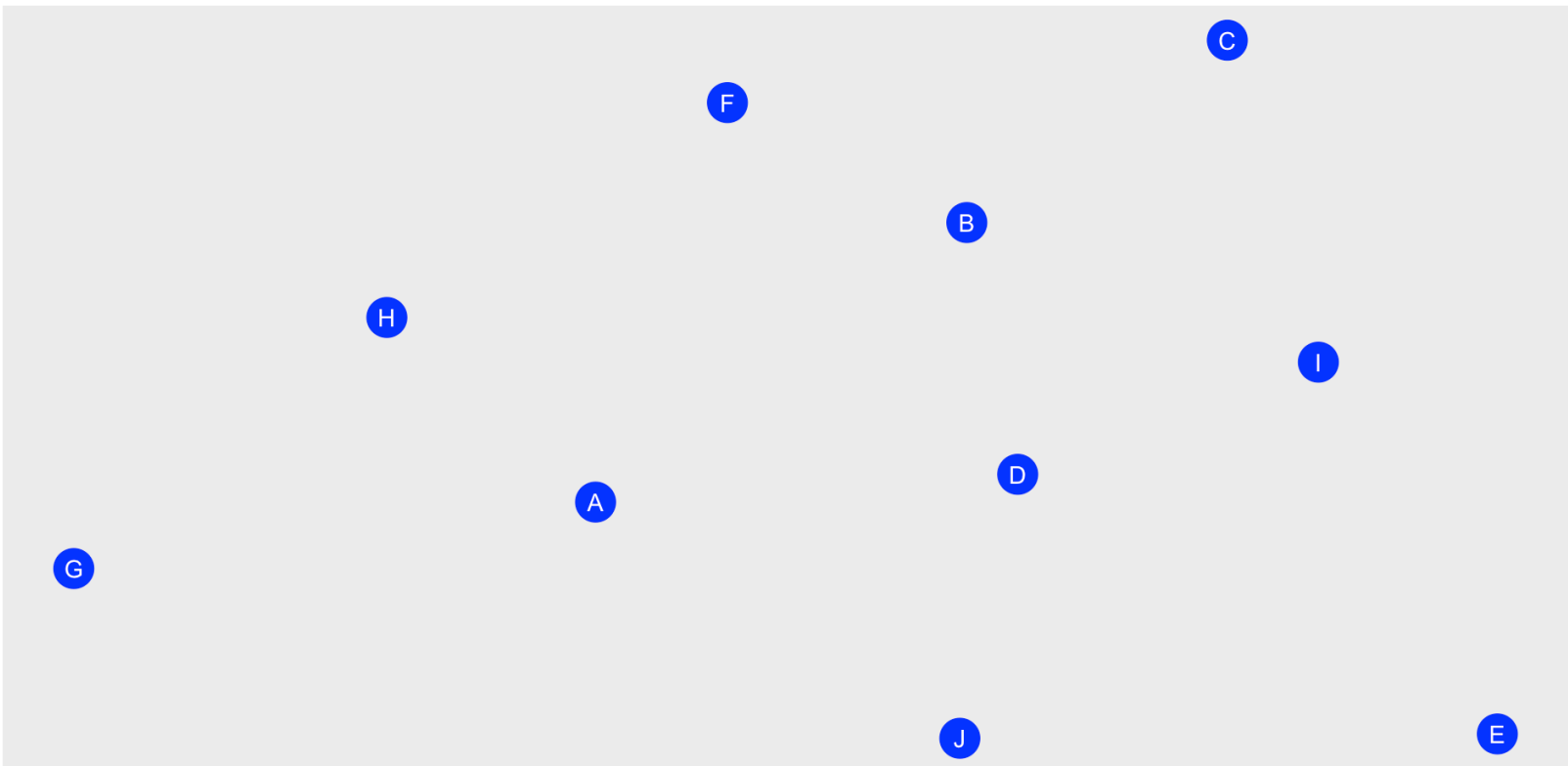
geom_node_point: aesthetics

```
gr <- graph %>% activate(nodes) %>%
  mutate(group = as.factor(group_louvain()))
ggraph(gr) +
  geom_node_point(aes(color = group), size = 8)
```



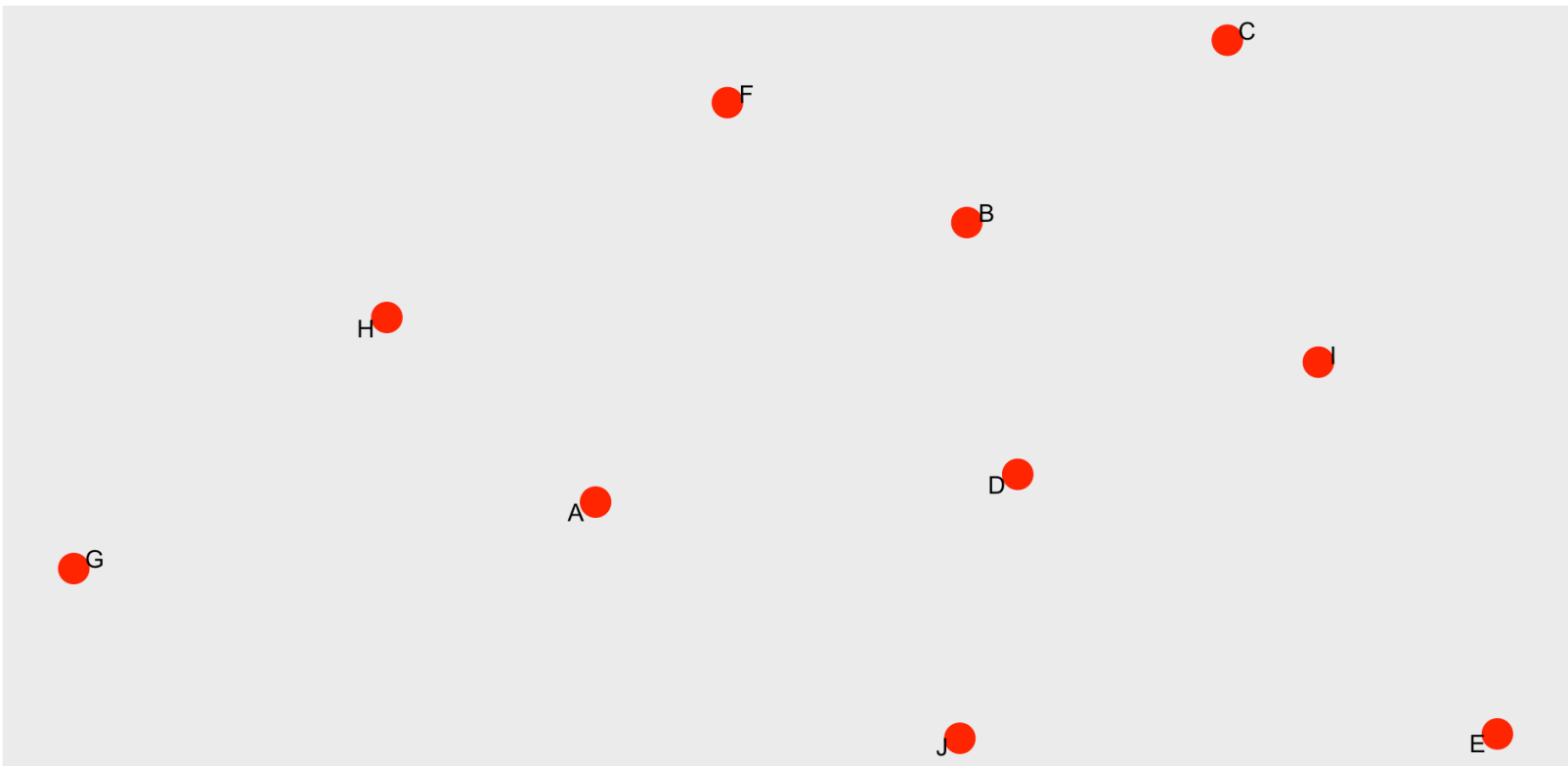
Adding Node Labels

```
1  gr = graph %>% activate(nodes) %>%
2    mutate(name = LETTERS[1:10])
3  ggraph(gr) +
4    geom_node_point(size = 8, color = 'blue') +
5    geom_node_text(aes(label = name), color = "white")
```



Adding Node Labels: repel

```
gr = graph %>% activate(nodes) %>%
  mutate(name = LETTERS[1:10])
ggraph(gr) +
  geom_node_point(size = 6, color = 'red') +
  geom_node_text(aes(label = name), color = "black", repel = T)
```

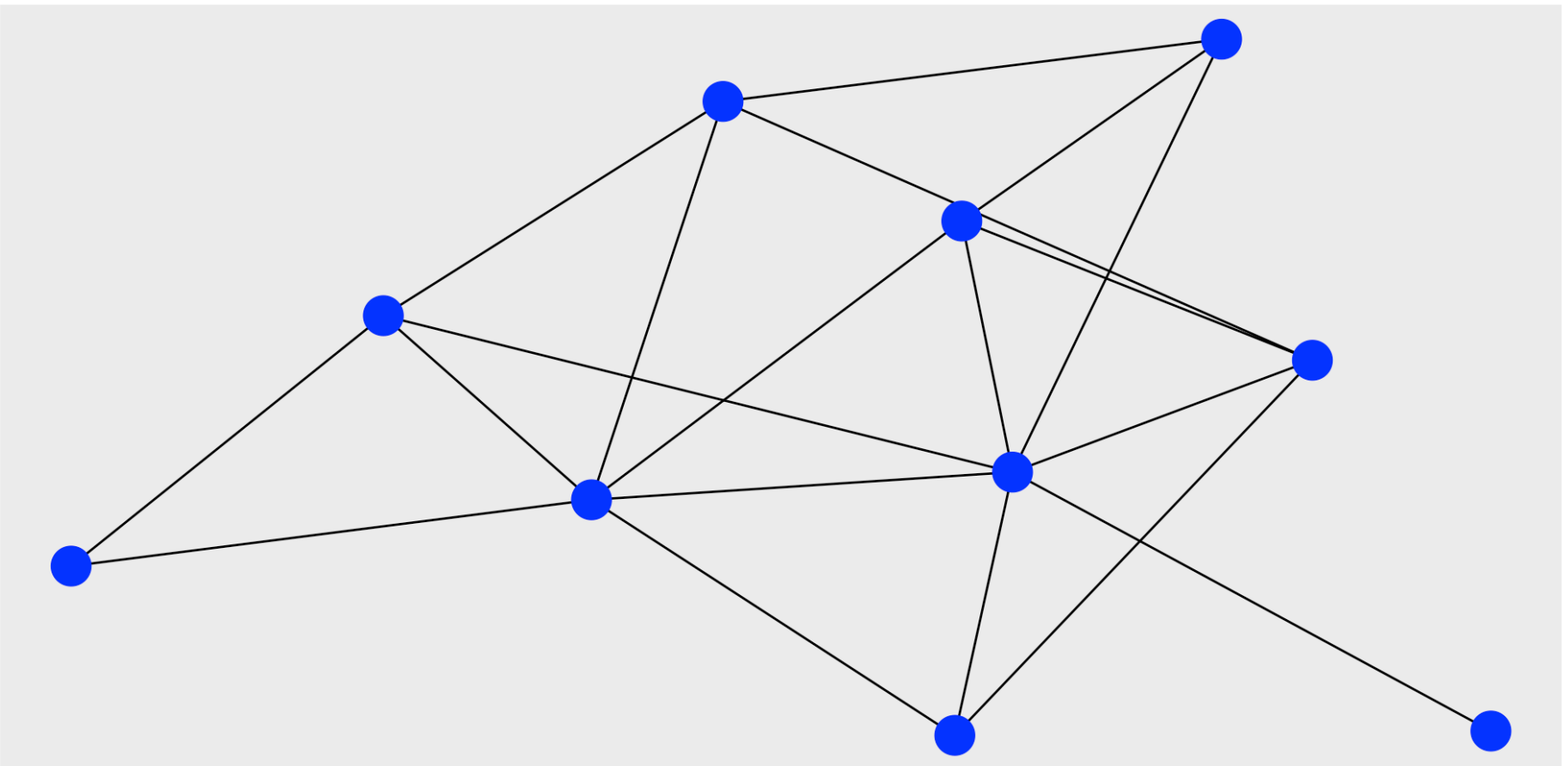


Overview of Edge Layers

- Edges represent the connections or relationships between nodes in a graph.
- Edge layers in ggraph can visualize various aspects of relationships, such as strength, direction, or type through aesthetics like width, color, and transparency.
- The geom_edge_link() function is the most commonly used layer for visualizing edges.

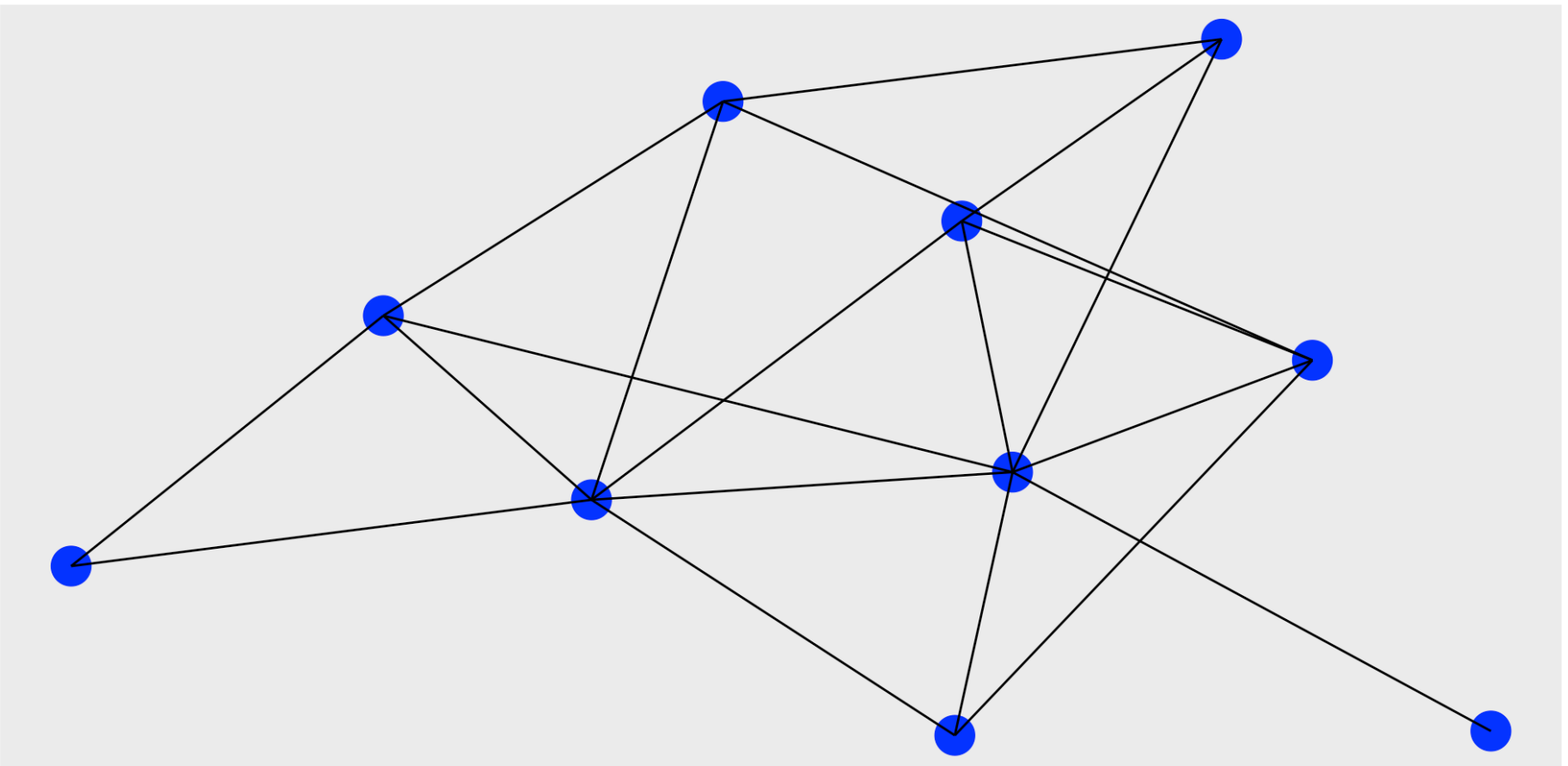
geom_edge_link

```
ggraph(graph) +
  geom_edge_link() +
  geom_node_point(size = 8, color = 'blue')
```



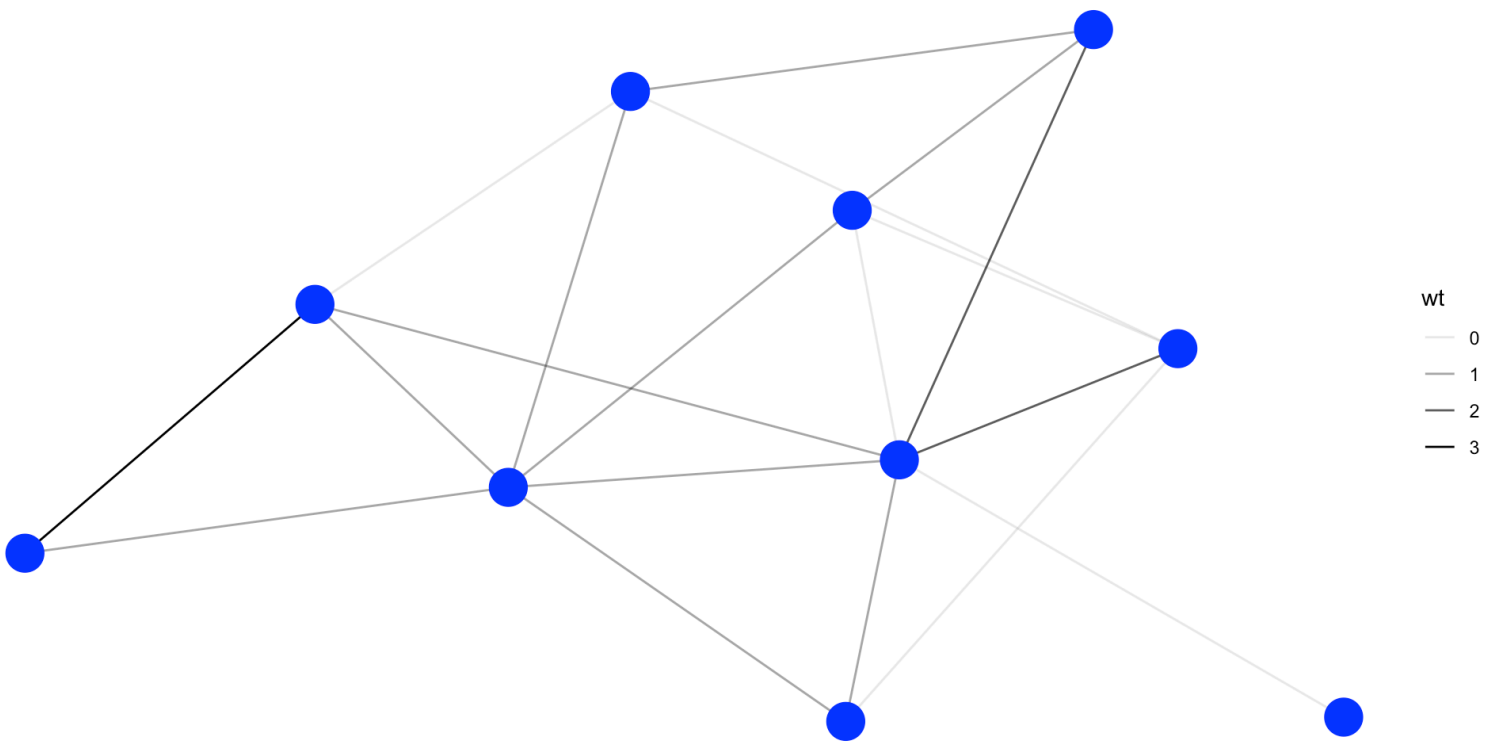
geom_edge_link: Order Matters

```
ggraph(graph) +
  geom_node_point(size = 8, color = 'blue') +
  geom_edge_link()
```



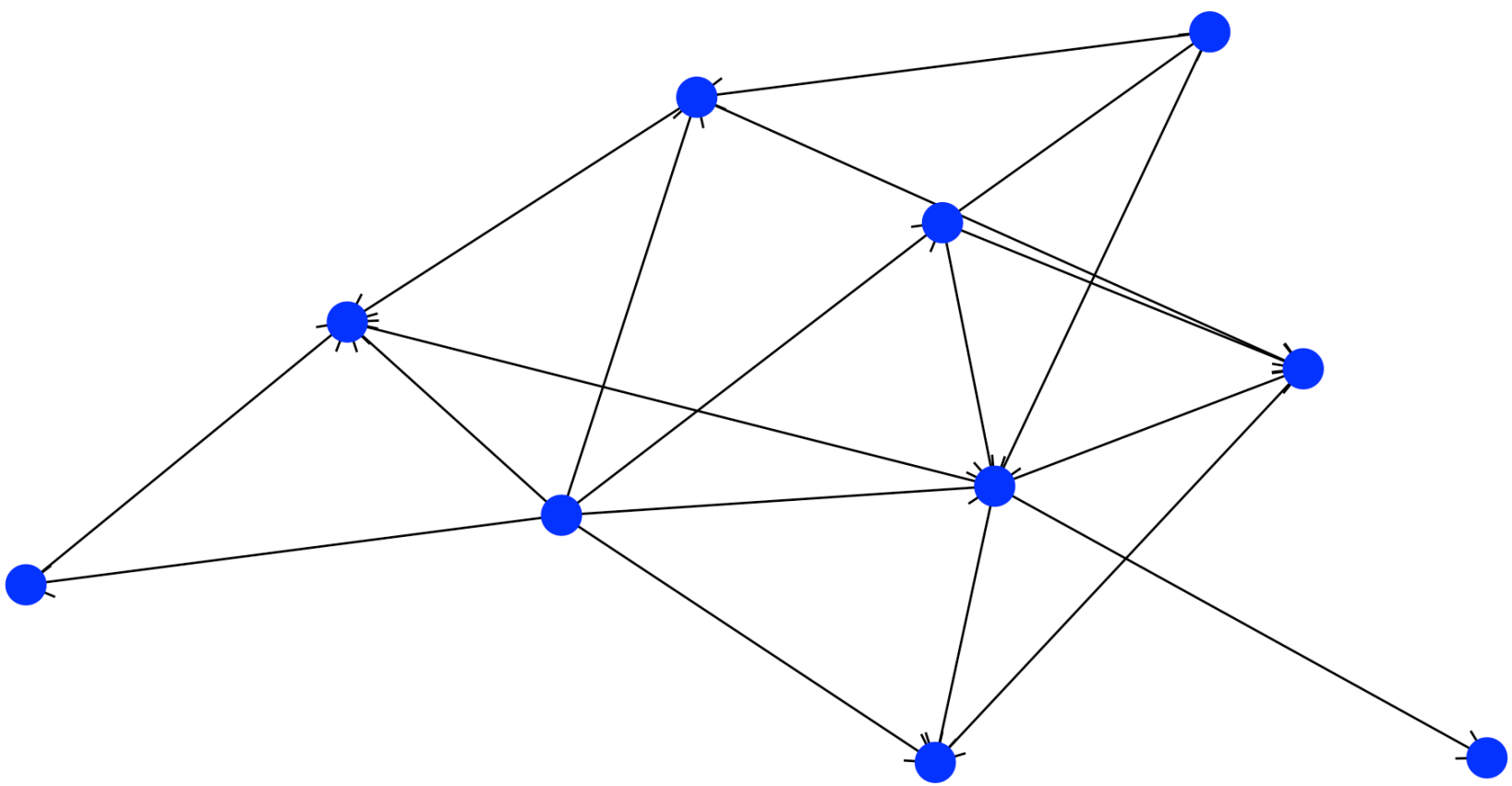
geom_edge_link: aesthetics

```
gr <- graph %>% activate(edges) %>%
  mutate(wt = rpois(n(), lambda = 1))
ggraph(gr) +
  geom_edge_link(aes(alpha = wt)) +
  geom_node_point(size = 8, color = 'blue') +
  theme_void()
```



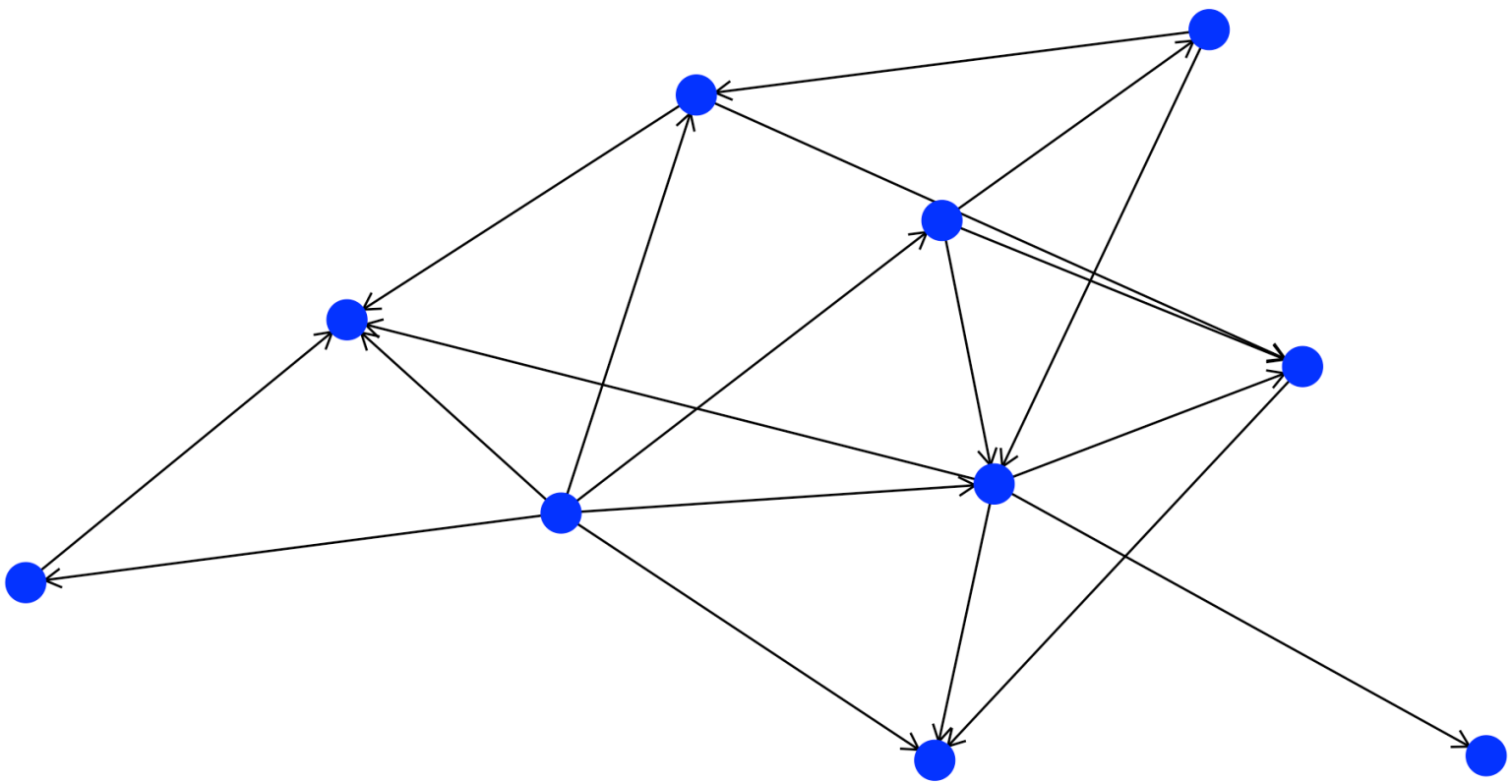
Edges in a Directed Graphs

```
ggraph(graph) +
  geom_edge_link(arrow = arrow(length = unit(5, 'mm'))) +
  geom_node_point(size = 8, color = 'blue') +
  theme_void()
```



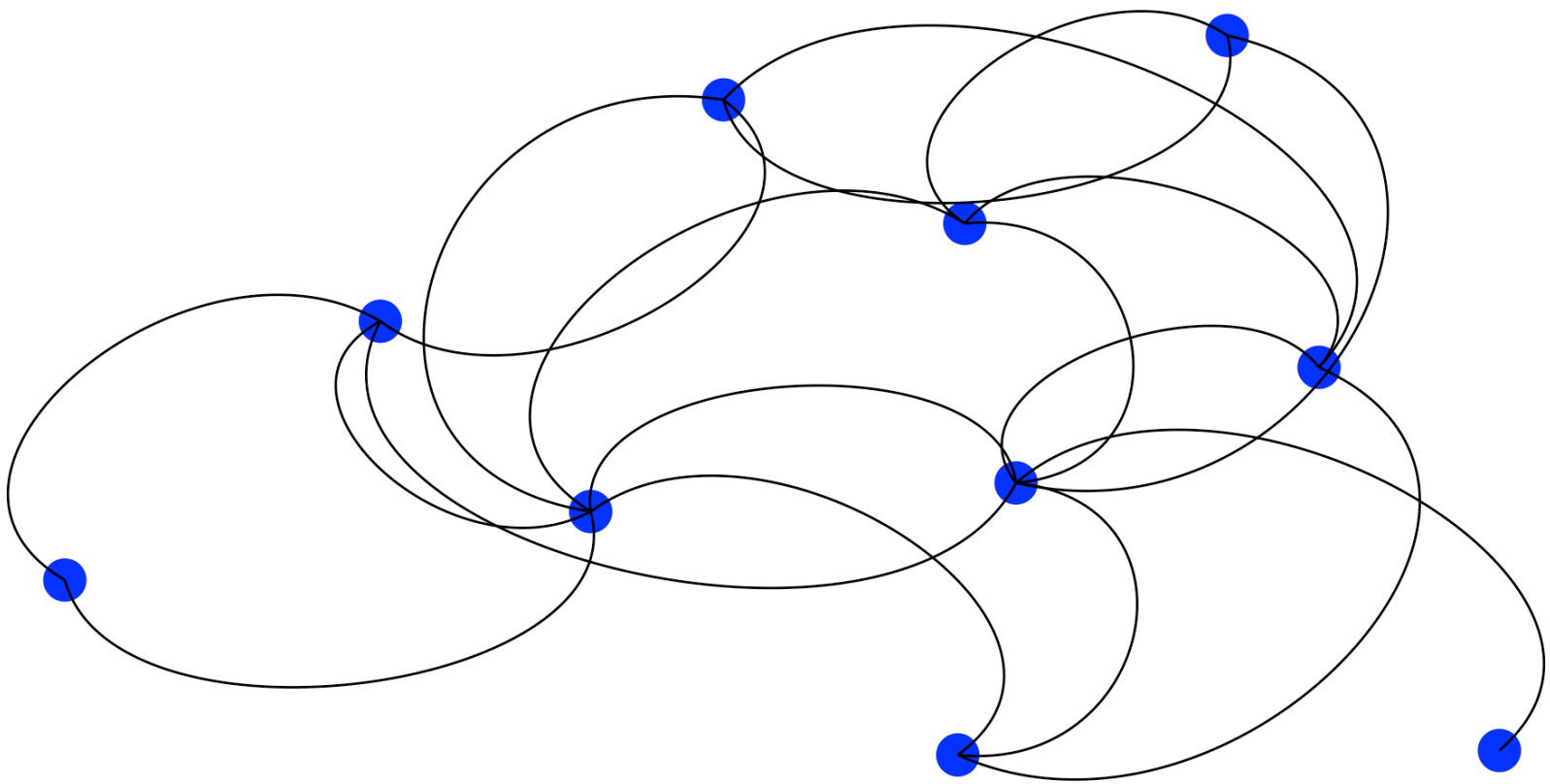
Edges in a Directed Graphs: end_cap

```
ggraph(graph) +
  geom_edge_link(arrow = arrow(length = unit(3, 'mm')),
                                 end_cap = circle(3, 'mm')) +
  geom_node_point(size = 8, color = 'blue') +
  theme_void()
```



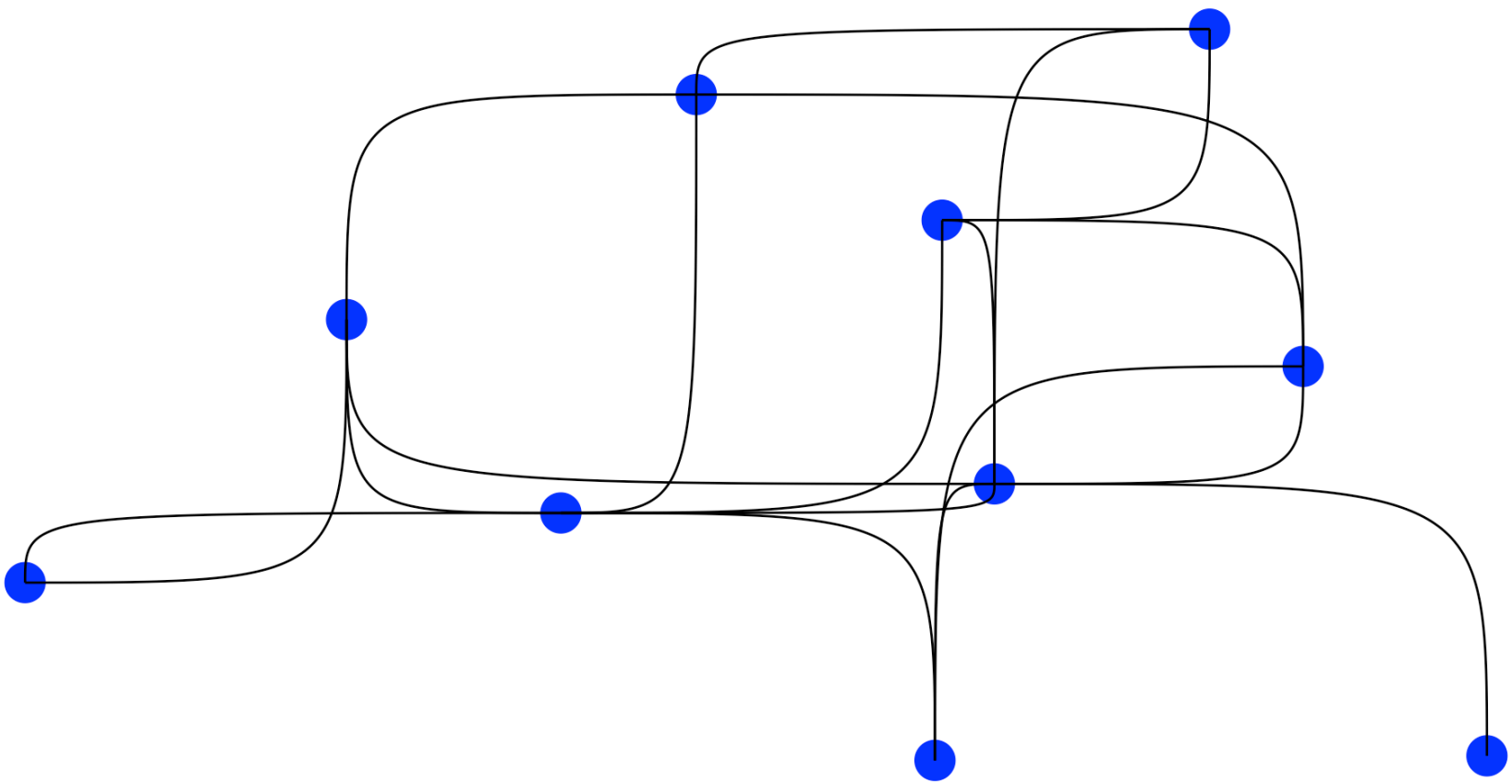
Other Kinds of Edges: arc

```
ggraph(graph) +
  geom_node_point(size = 8, color = 'blue') +
  geom_edge_arc() +
  theme_void()
```



Other Kinds of Edges: bend

```
ggraph(graph) +
  geom_node_point(size = 8, color = 'blue') +
  geom_edge_bend() +
  theme_void()
```



Overview of Layouts in ggraph

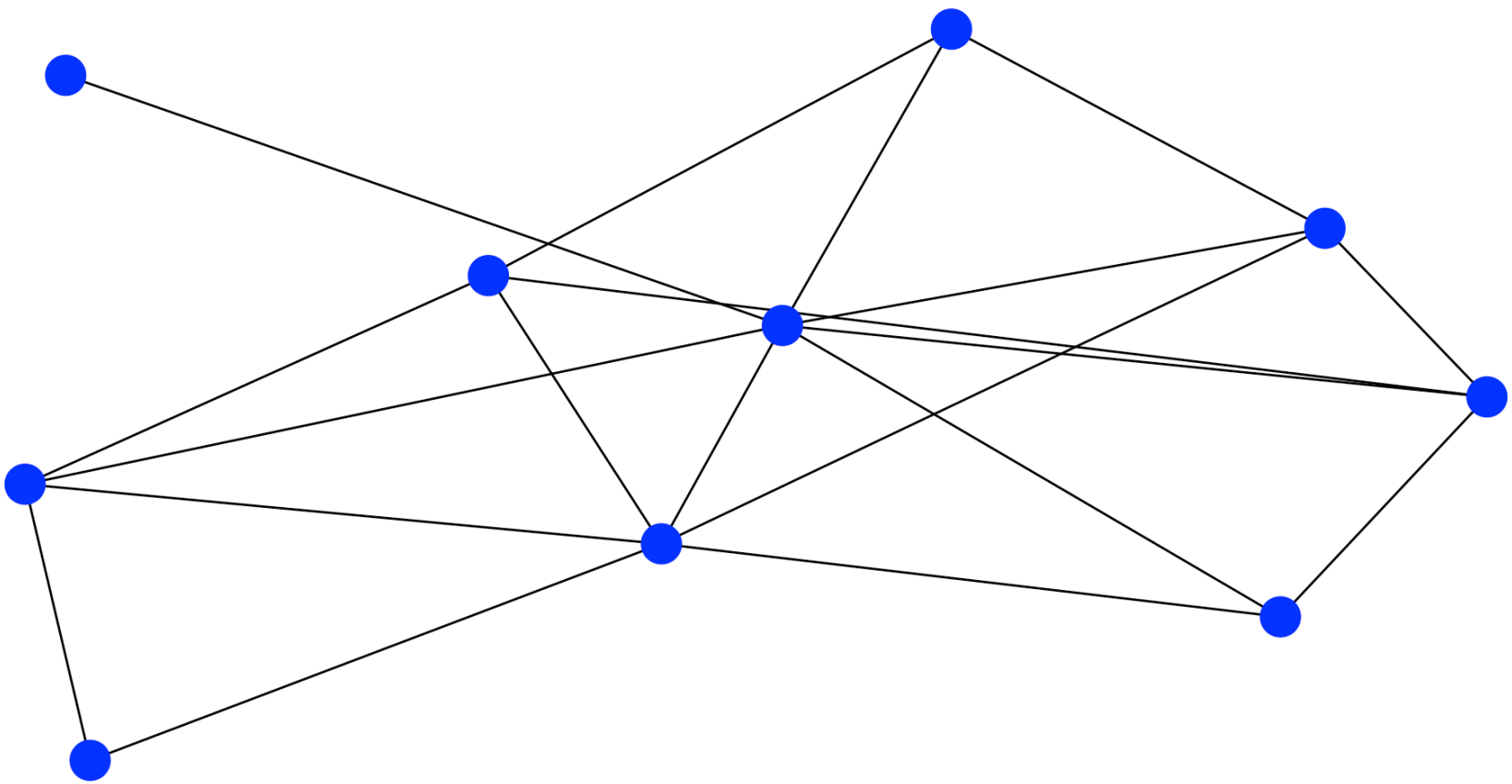
- Layouts determine the spatial arrangement of nodes and edges in a graph.
- Choosing the right layout is essential for clarity and aesthetics.
- ggraph supports a variety of layouts, including force-directed, circular, and hierarchical types.

Force-directed Layouts

- Force-directed layouts, such as ‘fr’ (Fruchterman-Reingold) and ‘kk’ (Kamada-Kawai), simulate physical forces to position nodes.
- They are ideal for visualizing relationships dynamically.

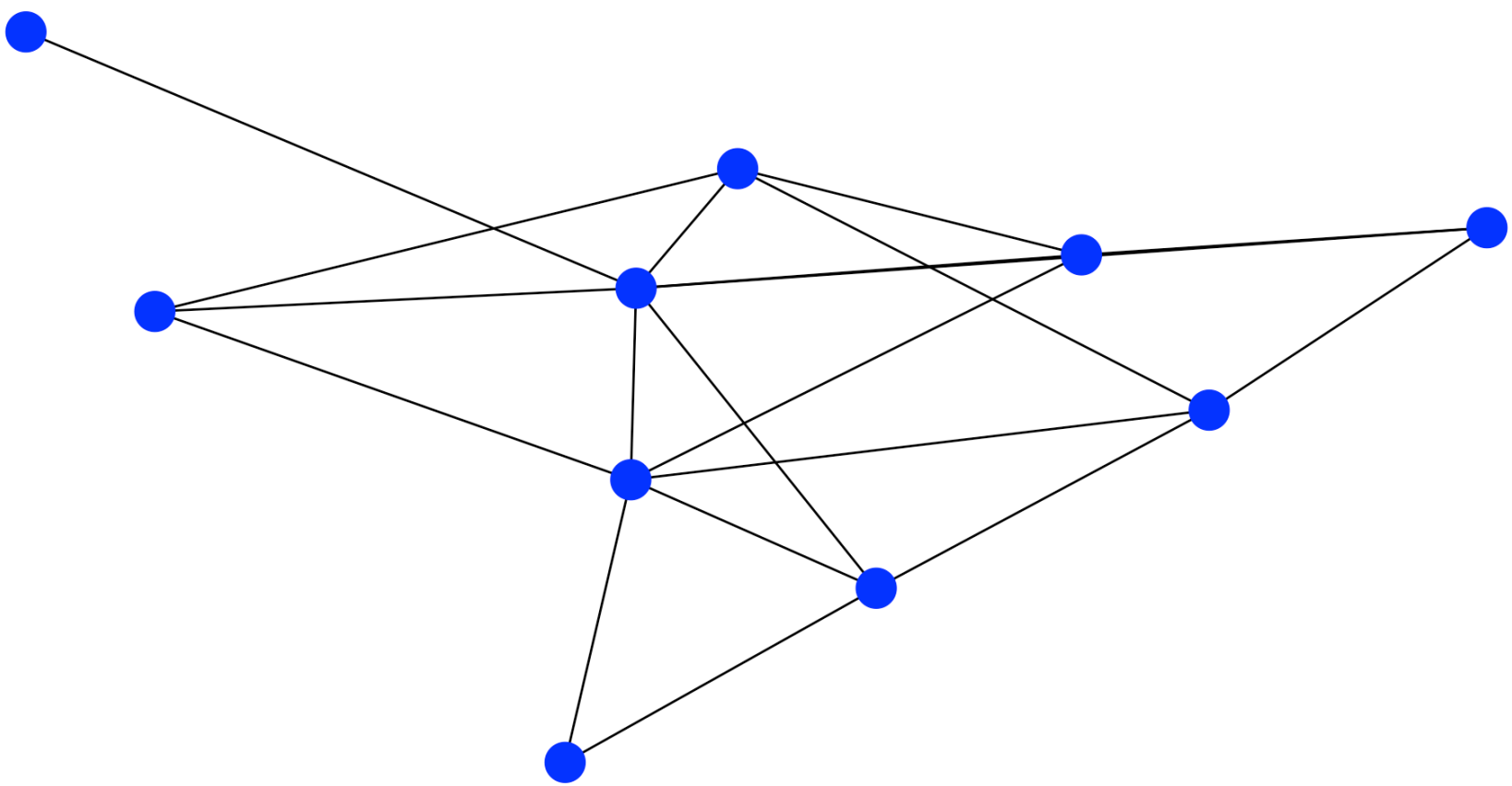
Kamada-Kawai Layout

```
ggraph(graph, layout = "kk") +
  geom_edge_link() +
  geom_node_point(size = 8, color = 'blue') +
  theme_void()
```



Fruchterman-Reingold Layout

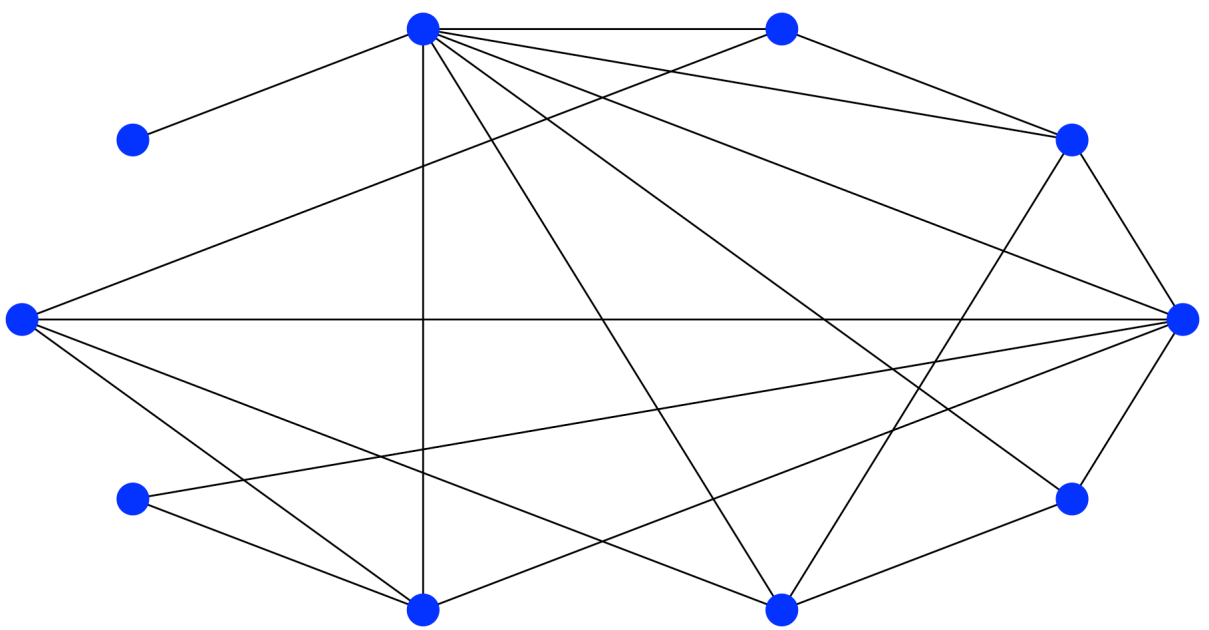
```
ggraph(graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point(size = 8, color = 'blue') +
  theme_void()
```



Circular Layouts

Circular layouts position nodes around a circle, making them suitable for cyclical or hierarchical relationships.

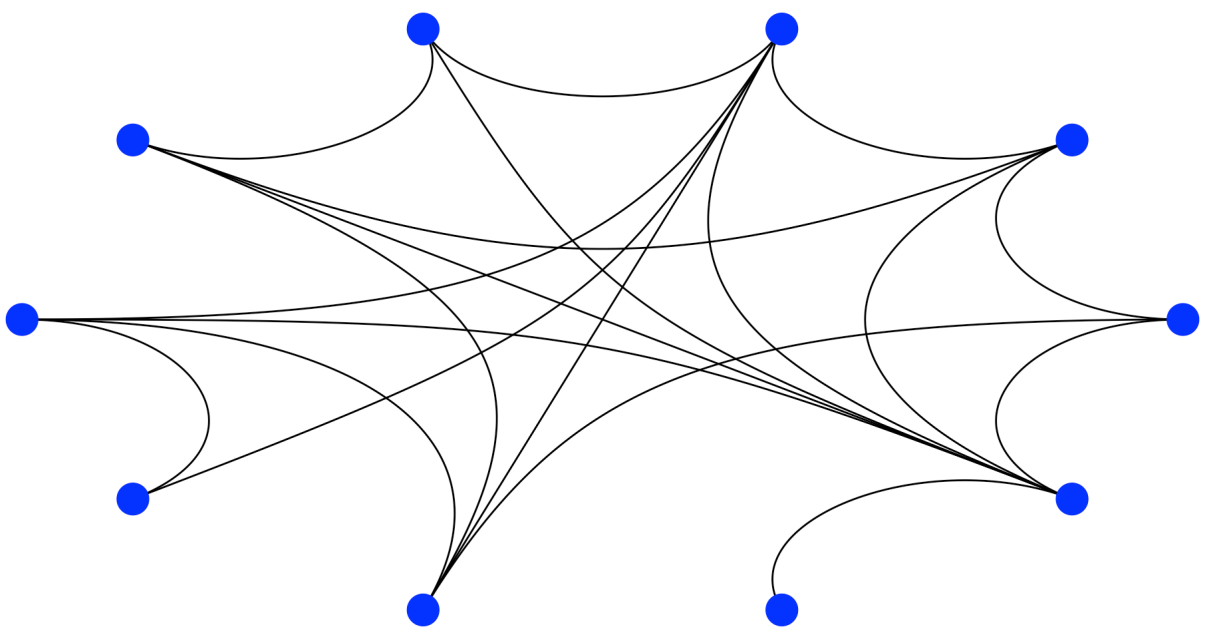
```
ggraph(graph, layout = "circle") +
  geom_edge_link() +
  geom_node_point(size = 8, color = 'blue') +
  theme_void()
```



Chord Diagrams

Used to compare similarities between groups of data or within a dataset.

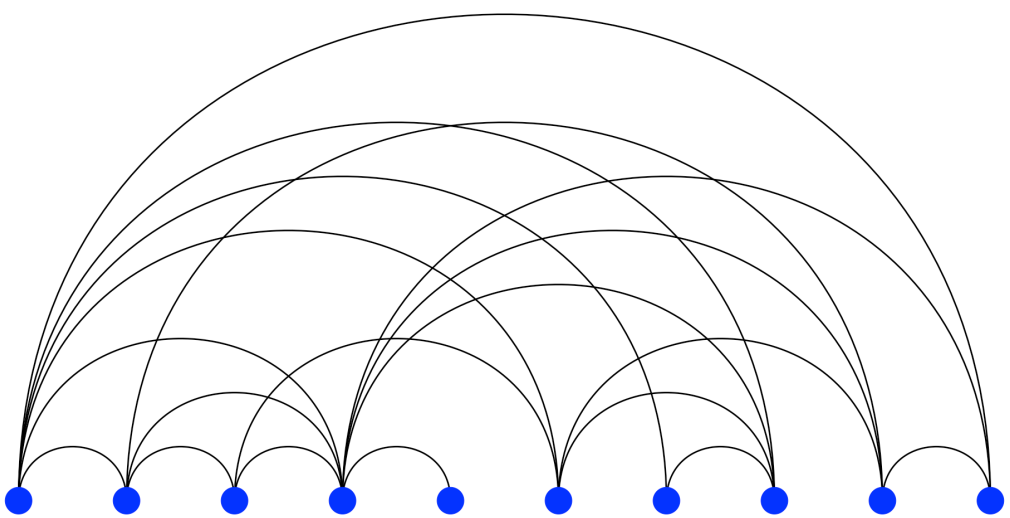
```
ggraph(graph, layout = "linear", circular = T) +
  geom_edge_arc() +
  geom_node_point(size = 8, color = 'blue') +
  theme_void()
```



Arc Diagrams

Nodes are placed along a single line (a one-dimensional axis) and arcs are used to show connections between those nodes, useful in finding the co-occurrence within the data.

```
ggraph(graph, layout = "linear") +
  geom_edge_arc() +
  geom_node_point(size = 8, color = 'blue') +
  theme_void()
```



An Example

nycflights Data

```r
library(nycflights13)
df = flights
head(df)
```

```
# A tibble: 6 × 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     1      517            515         2      830            819
2  2013     1     1      533            529         4      850            830
3  2013     1     1      542            540         2      923            850
4  2013     1     1      544            545        -1     1004           1022
5  2013     1     1      554            600        -6      812            837
6  2013     1     1      554            558        -4      740            728
# ℹ 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

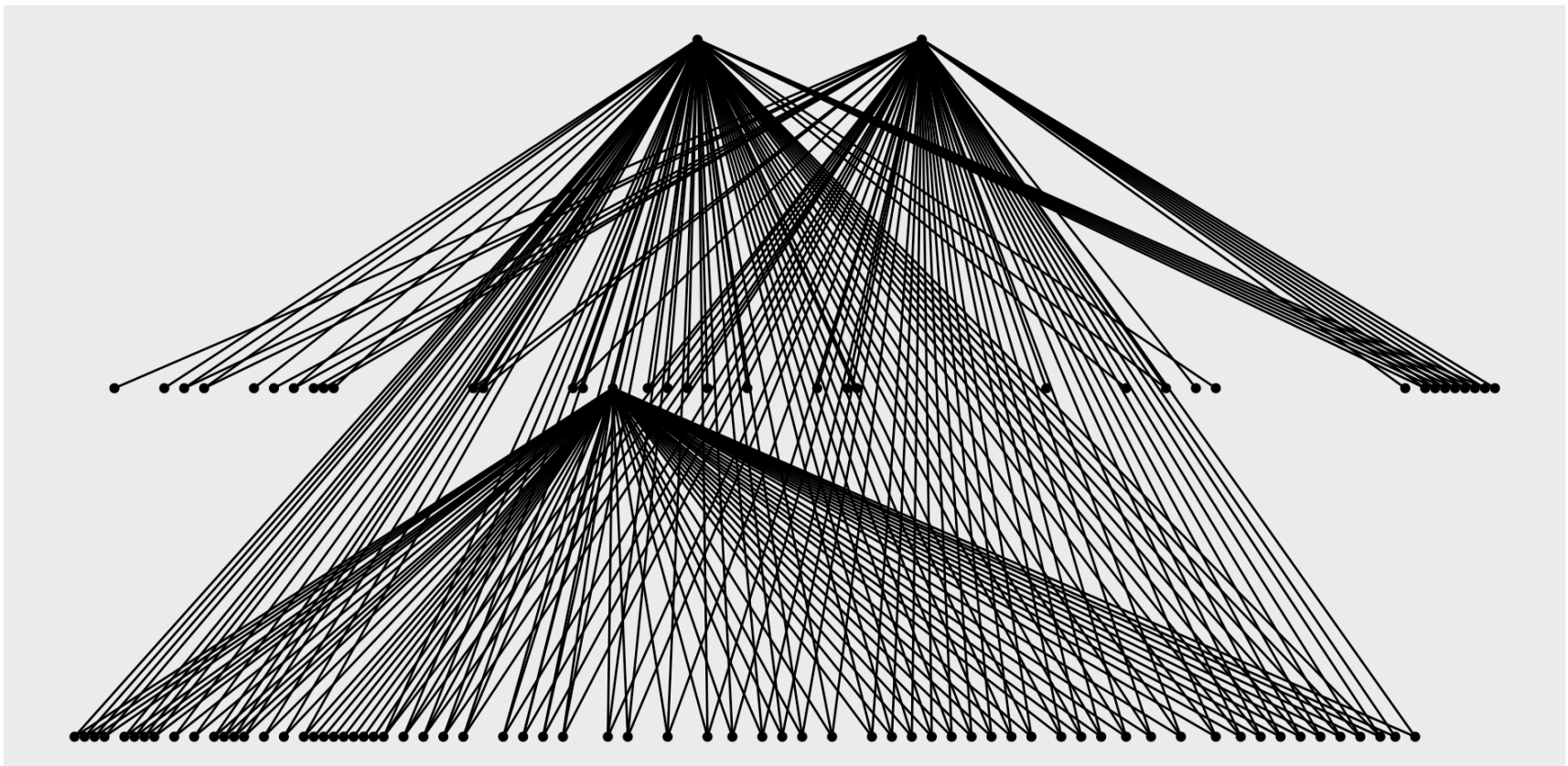
nycflights Data: Graph

```
1 gr = df %>% count(origin, dest) %>%
2   as_tbl_graph()
3 gr
```

```
# A tbl_graph: 107 nodes and 224 edges
#
# A directed acyclic simple graph with 1 component
#
# Node Data: 107 × 1 (active)
   name
   <chr>
 1 EWR
 2 JFK
 3 LGA
 4 ALB
 5 ANC
 6 ATL
 7 AUS
 8 AVL
```

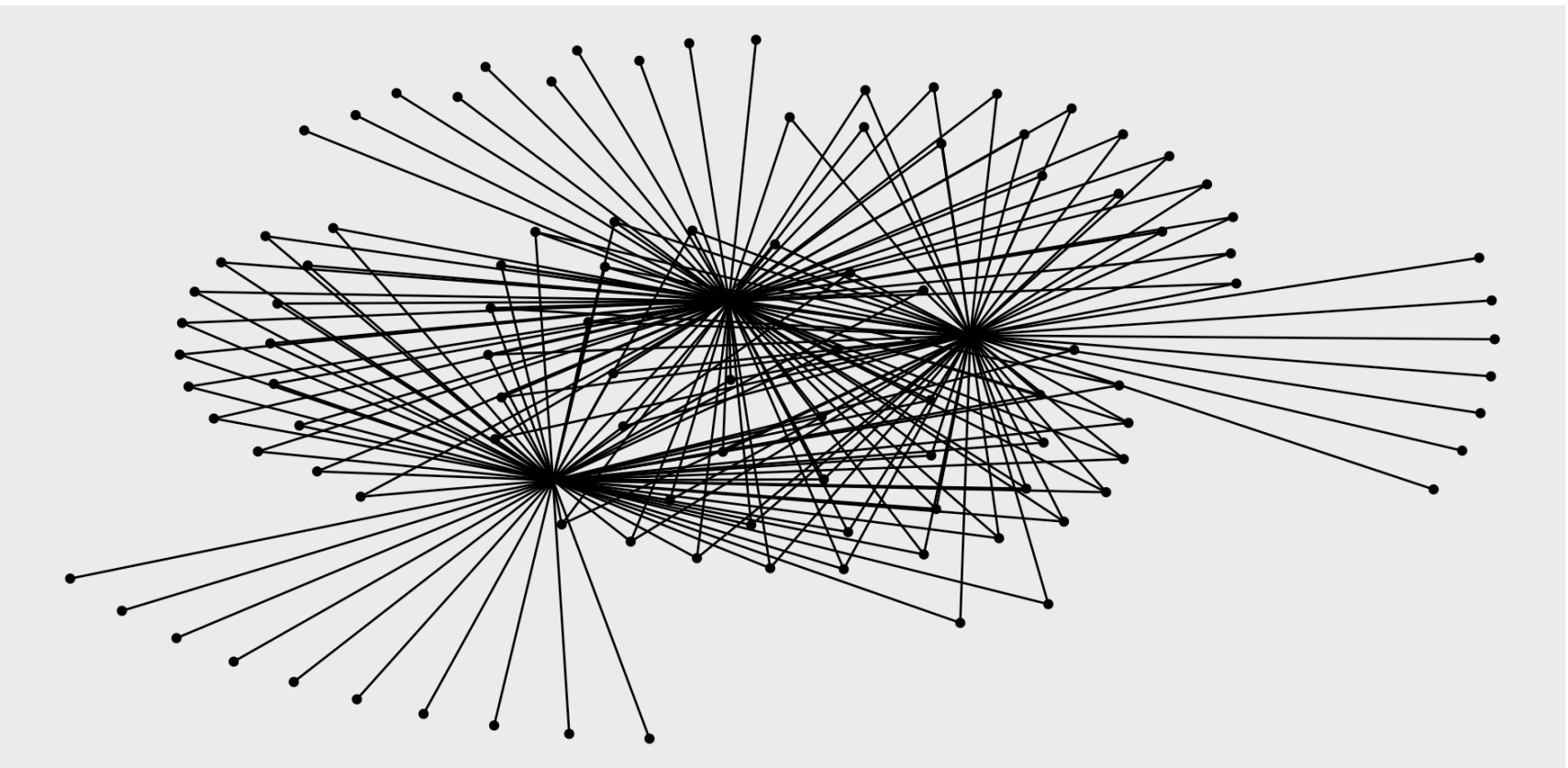
nycflights Data: Simple Visualization

```
# base graph
gr %>% ggraph() +
  geom_edge_link() +
  geom_node_point()
```



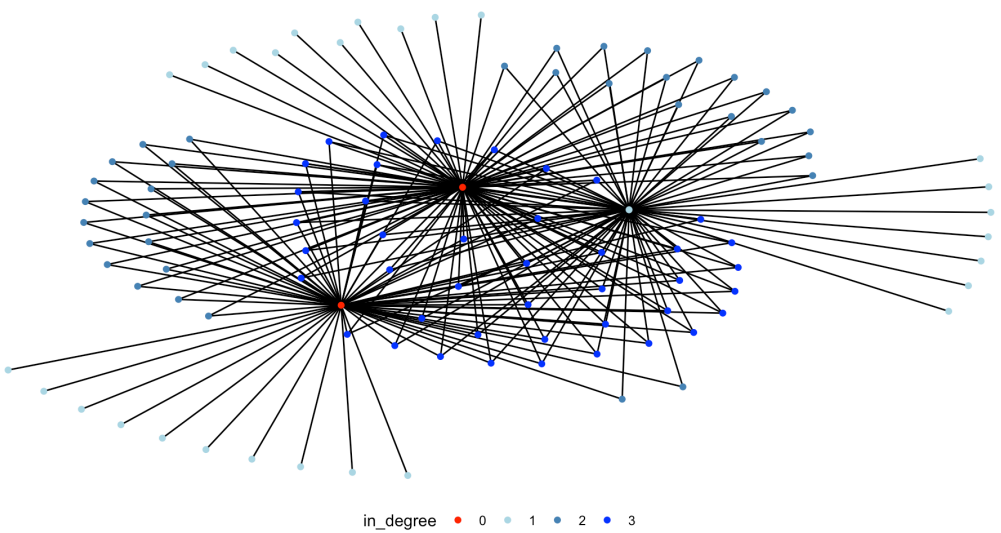
nycflights Data: Layout

```
# adjust layout
gr %>% ggraph(layout = "kk") +
  geom_edge_link() +
  geom_node_point()
```

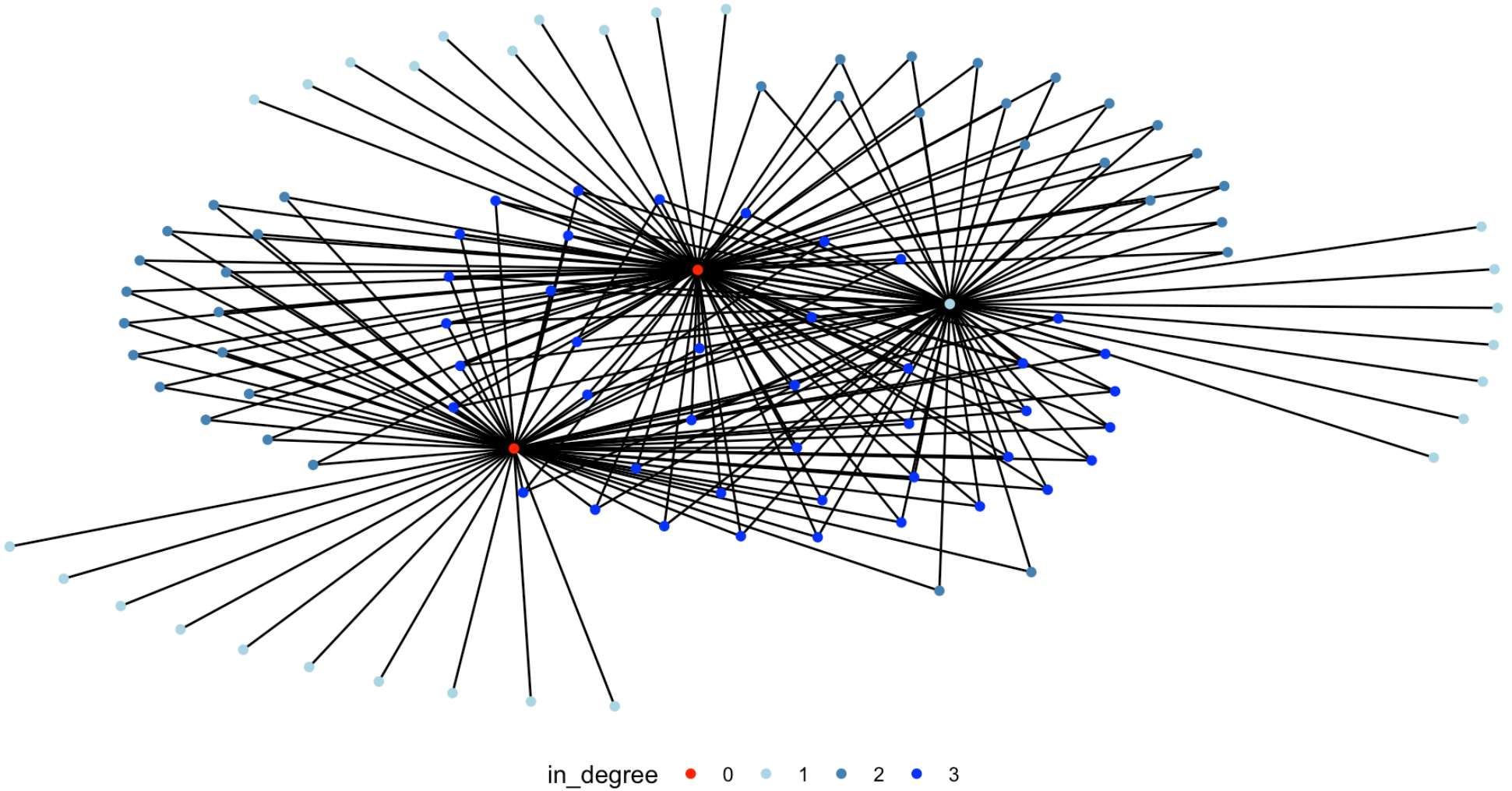


nycflights Data: In-Degree

```
# color nodes based on in-degree
gr %>%
  activate(nodes) %>%
  mutate(in_degree = as.factor(centrality_degree(mode = "in"))) %>%
  ggraph(layout = "kk") +
  geom_edge_link() +
  geom_node_point(aes(color = in_degree)) +
  scale_color_manual(values = c("0" = "red", "1" = "lightblue", "2" = "stee
  theme_void() +
  theme(legend.position = "bottom")
```

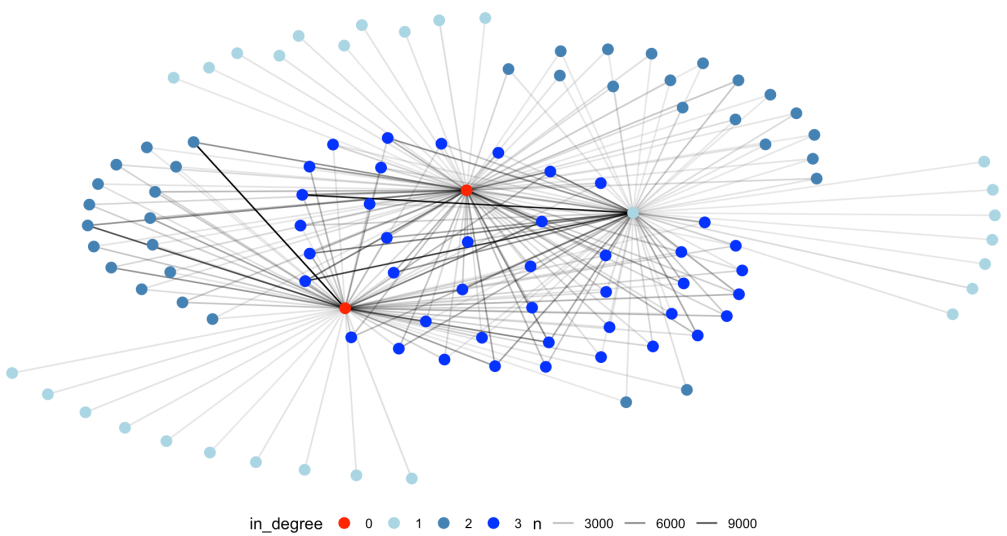


nycflights Data: In-Degree

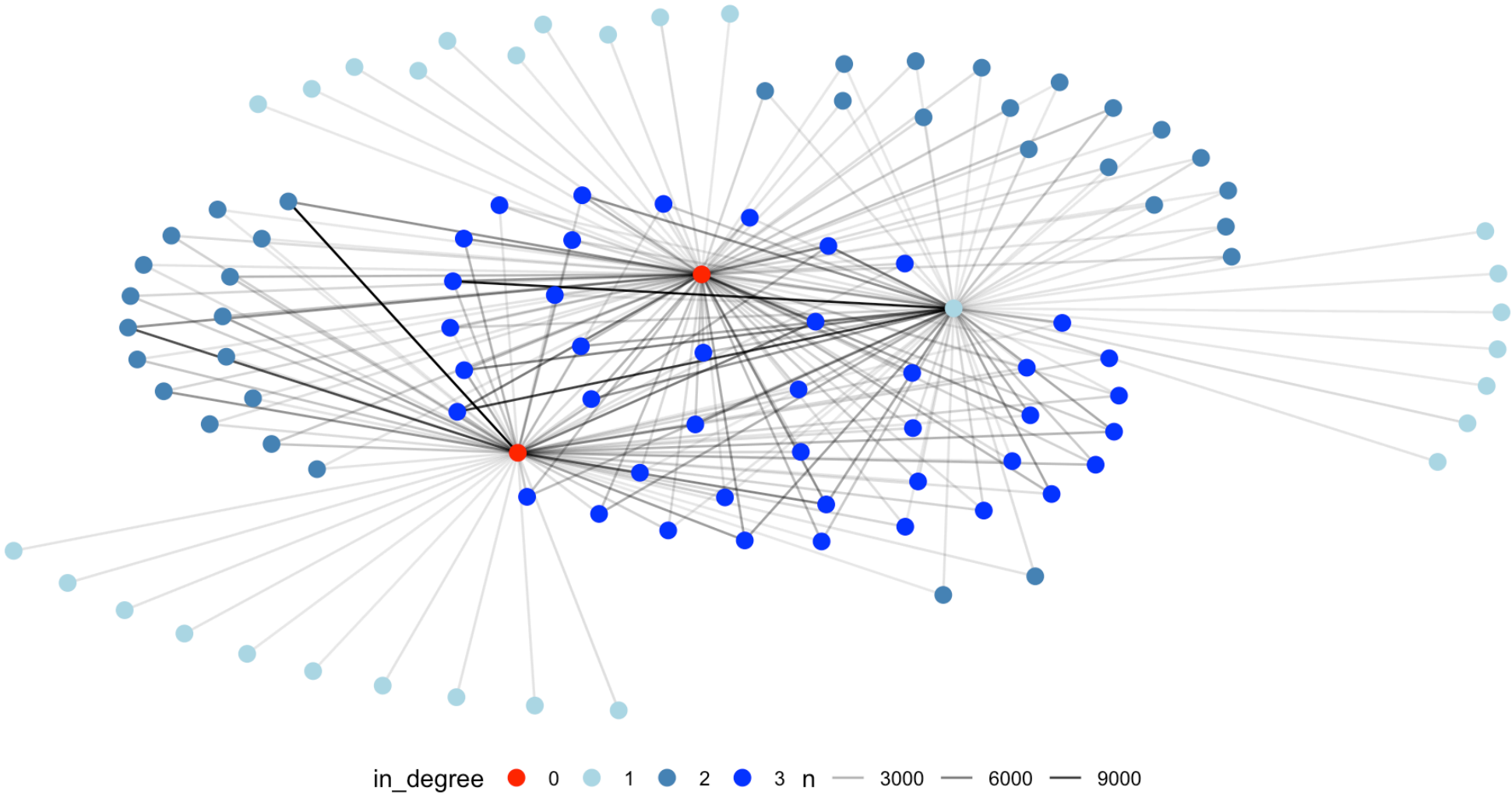


nycflights Data: Number of Flights

```
# add info on number of flights to edges + increase size
gr %>%
  activate(nodes) %>%
  mutate(in_degree = as.factor(centrality_degree(mode = "in"))) %>%
  ggraph(layout = "kk") +
  geom_edge_link(aes(alpha = n)) +
  geom_node_point(aes(color = in_degree), size = 3) +
  scale_color_manual(values = c("0" = "red", "1" = "lightblue", "2" = "stee
  theme_void() +
  theme(legend.position = "bottom")
```

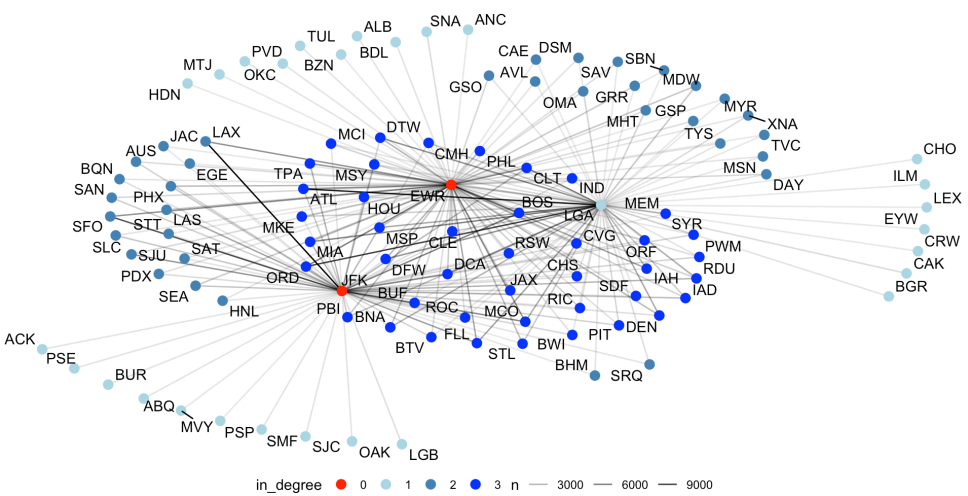


nycflights Data: Number of Flights

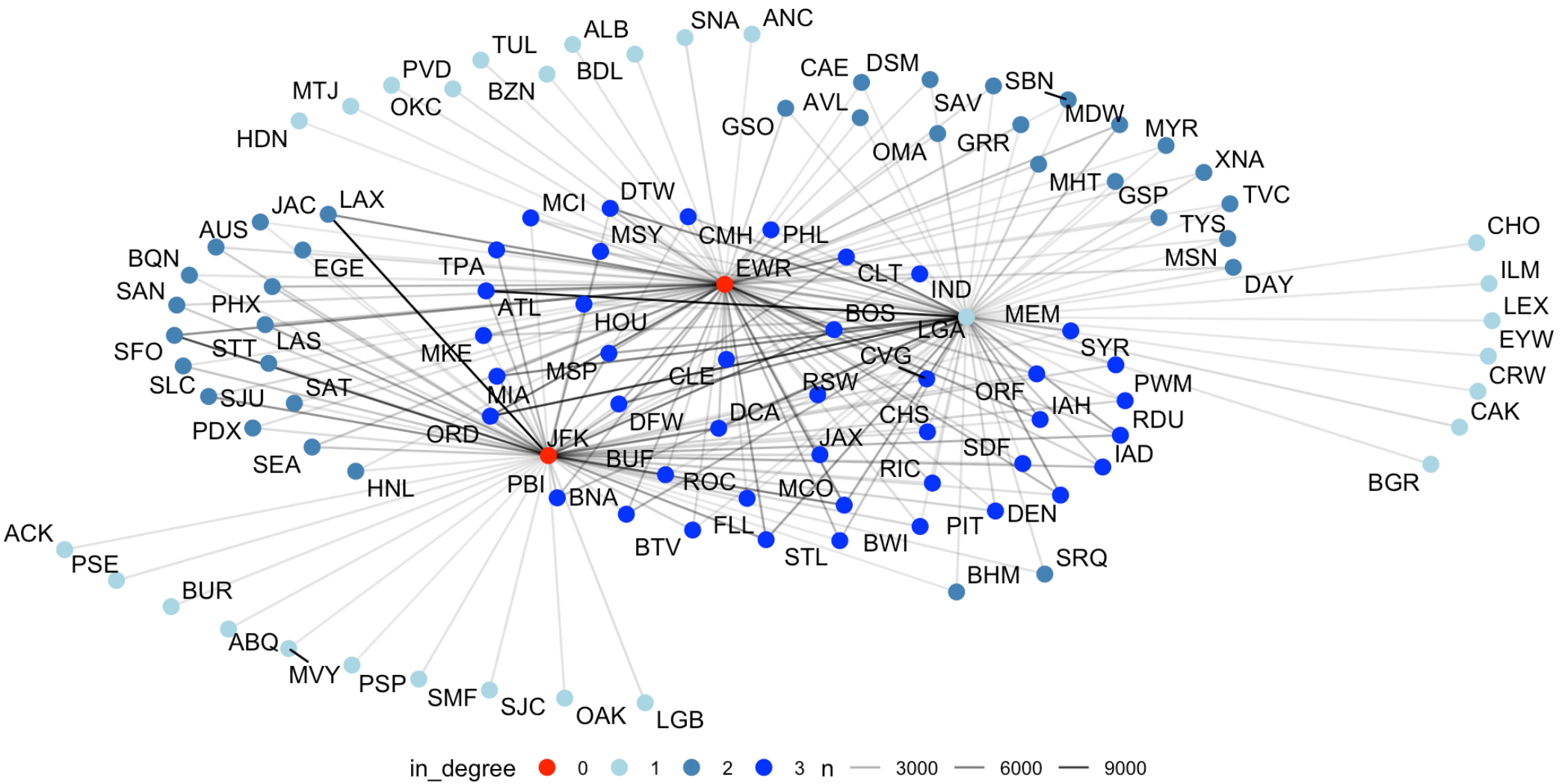


nycflights Data: Labels (1)

```
# add labels: repel
gr %>%
  activate(nodes) %>%
  mutate(in_degree = as.factor(centrality_degree(mode = "in"))) %>%
  ggraph(layout = "kk") +
  geom_edge_link(aes(alpha = n)) +
  geom_node_point(aes(color = in_degree), size = 3) +
  scale_color_manual(values = c("0" = "red", "1" = "lightblue", "2" = "stee
  theme_void() +
  theme(legend.position = "bottom") +
  geom_node_text(aes(label = name), repel = T)
```

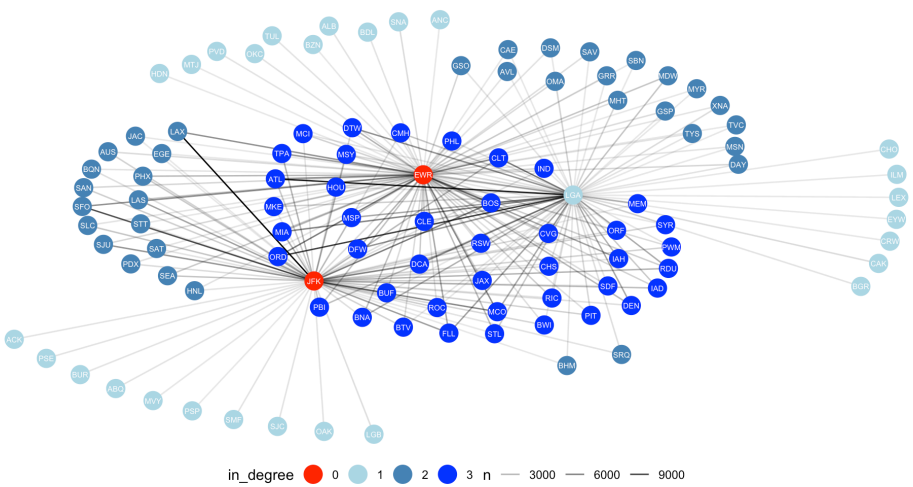


nycflights Data: Labels (1)

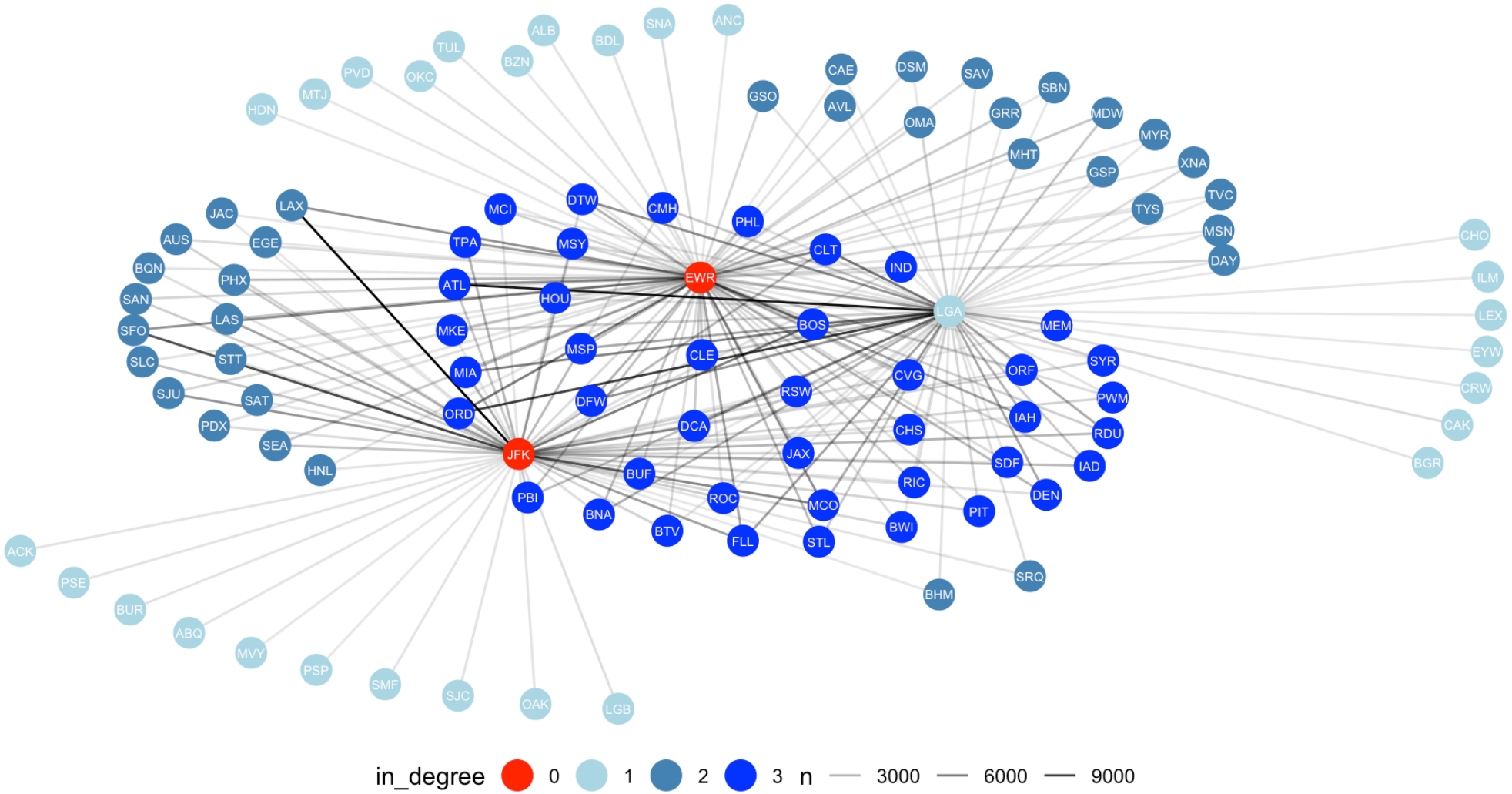


nycflights Data: Labels (2)

```
# add labels: inside points
gr %>%
  activate(nodes) %>%
  mutate(in_degree = as.factor(centrality_degree(mode = "in"))) %>%
  ggraph(layout = "kk") +
  geom_edge_link(aes(alpha = n)) +
  geom_node_point(aes(color = in_degree), size = 6) +
  scale_color_manual(values = c("0" = "red", "1" = "lightblue", "2" = "stee
  theme_void() +
  theme(legend.position = "bottom") +
  geom_node_text(aes(label = name), size = 2, color = "white")
```

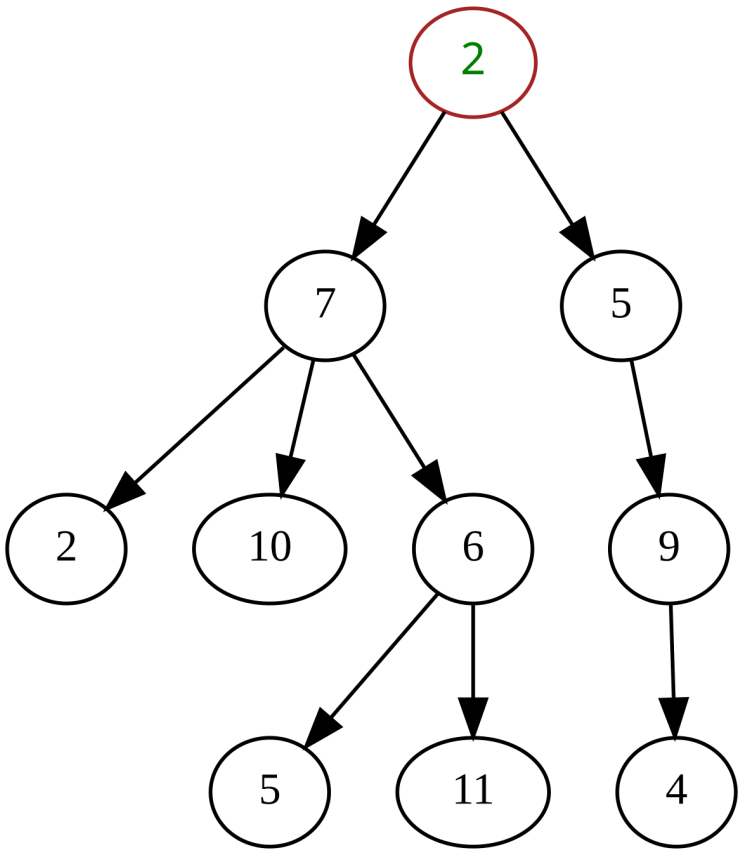


nycflights Data: Labels (2)



Trees in tidygraph

- A tree is a data structure used to represent hierarchical relationships.
- Nodes connected by edges, and each node in the tree contains data and is connected to other nodes in a parent-child relationship.



https://en.wikipedia.org/wiki/Tree_%2

Components of a Tree

- Node: Represents an element or data point in the tree.
- Edge: Represents the connection between two nodes.
- Root: The topmost node in a tree, with no parent. All other nodes are descendants of this root.
- Parent: A node that has one or more children.
- Child: A node that is a descendant of another node.
- Leaf: A node that does not have any children.
- Subtree: A tree formed by a node and its descendants.

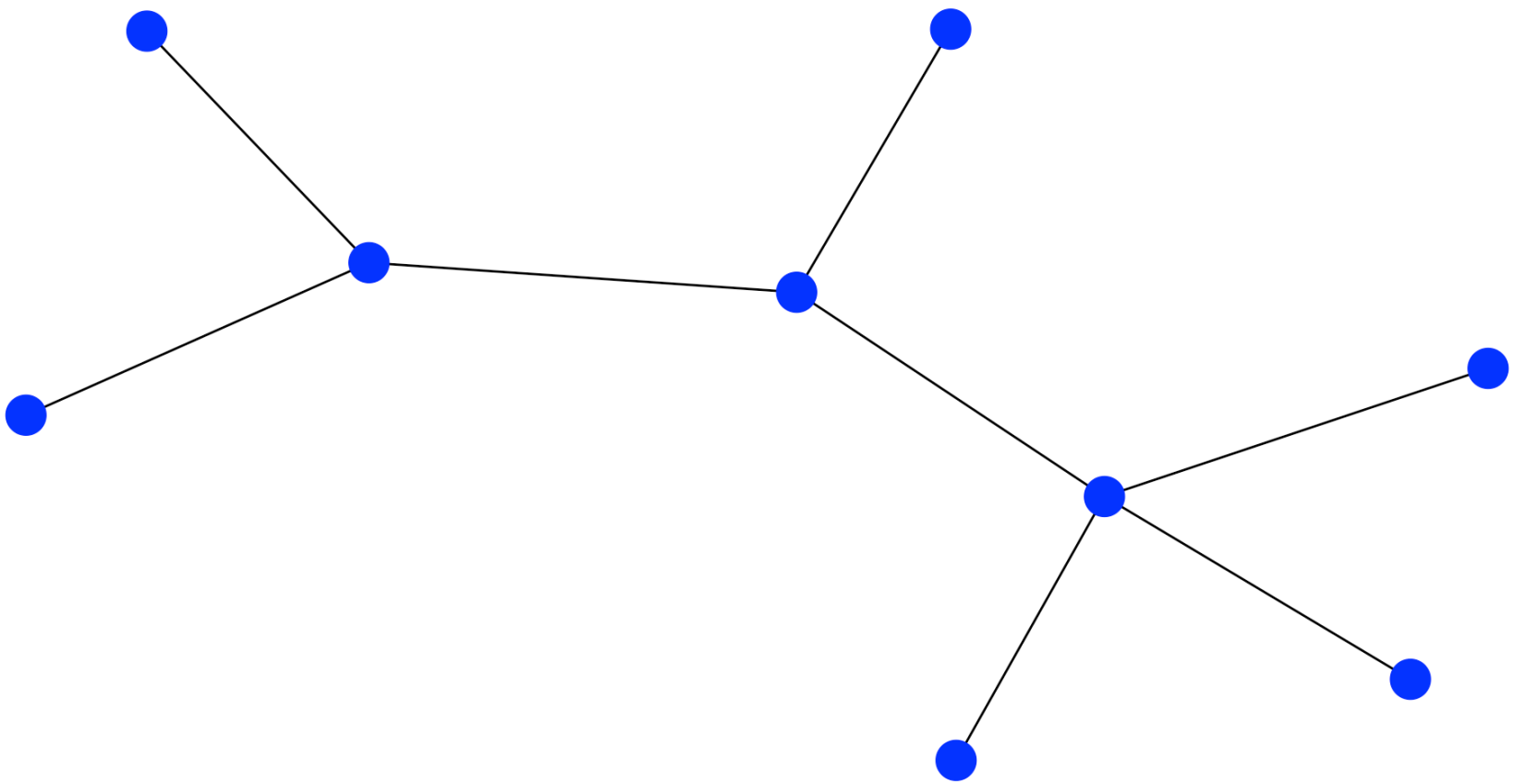
Trees in tidygraph

```r
# Create a sample hierarchical graph
graph <- data.frame(from = c("A", "A", "A", "B", "B", "C", "C", "C"),
                    to = c("B", "C", "I", "D", "E", "F", "G", "H"))
graph <- as_tbl_graph(graph)
print(graph)
```

```
# A tbl_graph: 9 nodes and 8 edges
#
# A rooted tree
#
# Node Data: 9 × 1 (active)
  name
  <chr>
1 A
2 B
3 C
4 I
5 D
6 E
7 F
8 G
```

Trees in tidygraph: kk layout

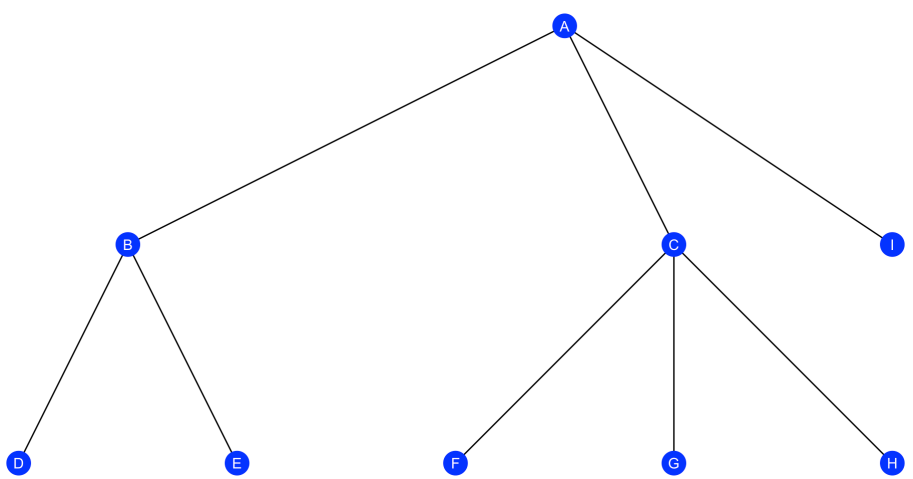
```
ggraph(graph, layout = "kk") +
  geom_edge_link() +
  geom_node_point(size = 8, color = 'blue') +
  theme_void()
```



Creating a Tree Layout

dendrogram layout arranges nodes and edges linearly in a hierarchical structure. The layout positions nodes based on their depth in the tree.

```
ggraph(graph, layout = 'tree') +
  geom_edge_link() +
  geom_node_point(size = 8, color = 'blue') +
  geom_node_text(aes(label = name), color = "white") +
  theme_void()
```



Creating a Dendrogram Layout

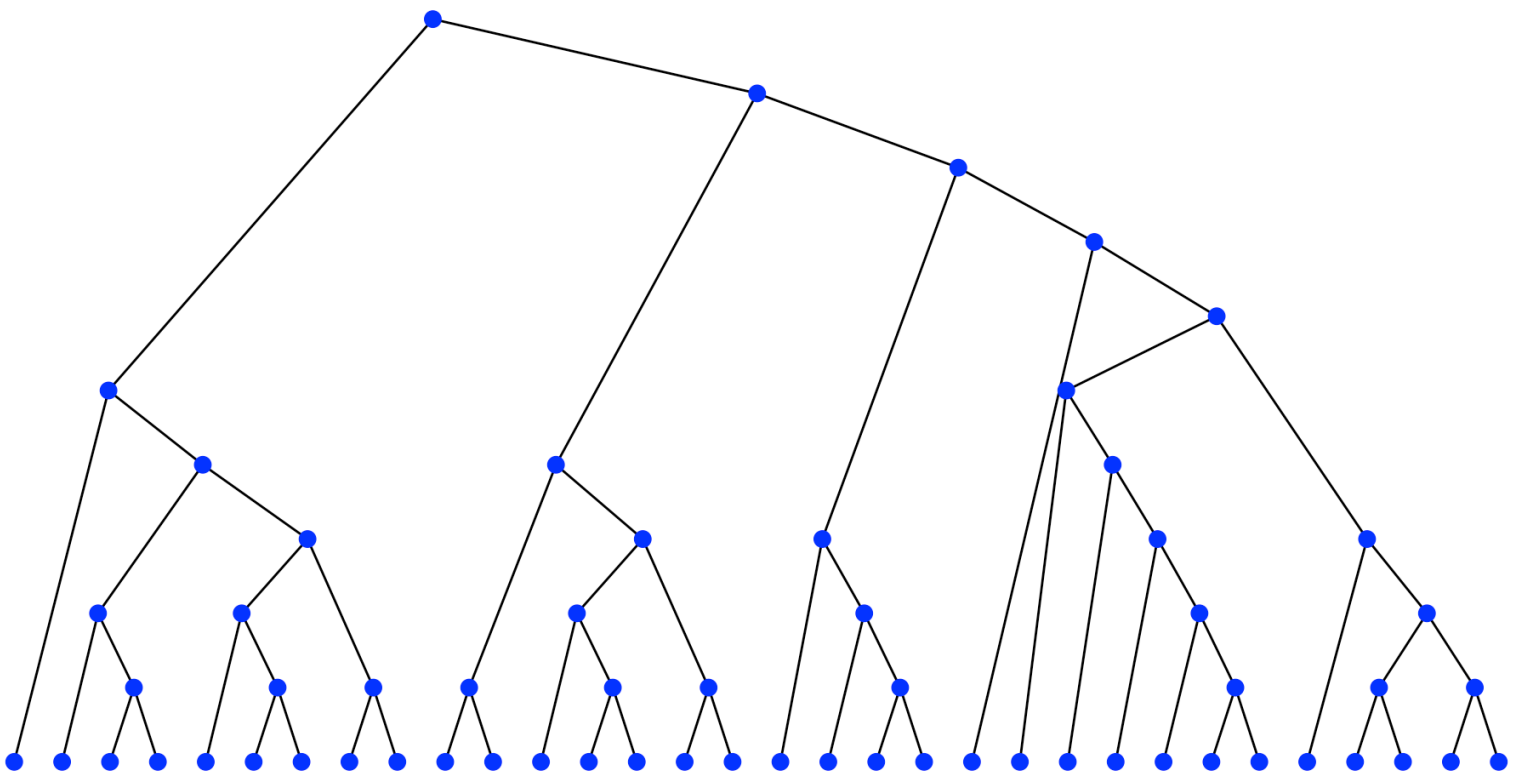
Dendrogram layouts are also very nice to show the output of hierarchical clustering

```
1 head(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

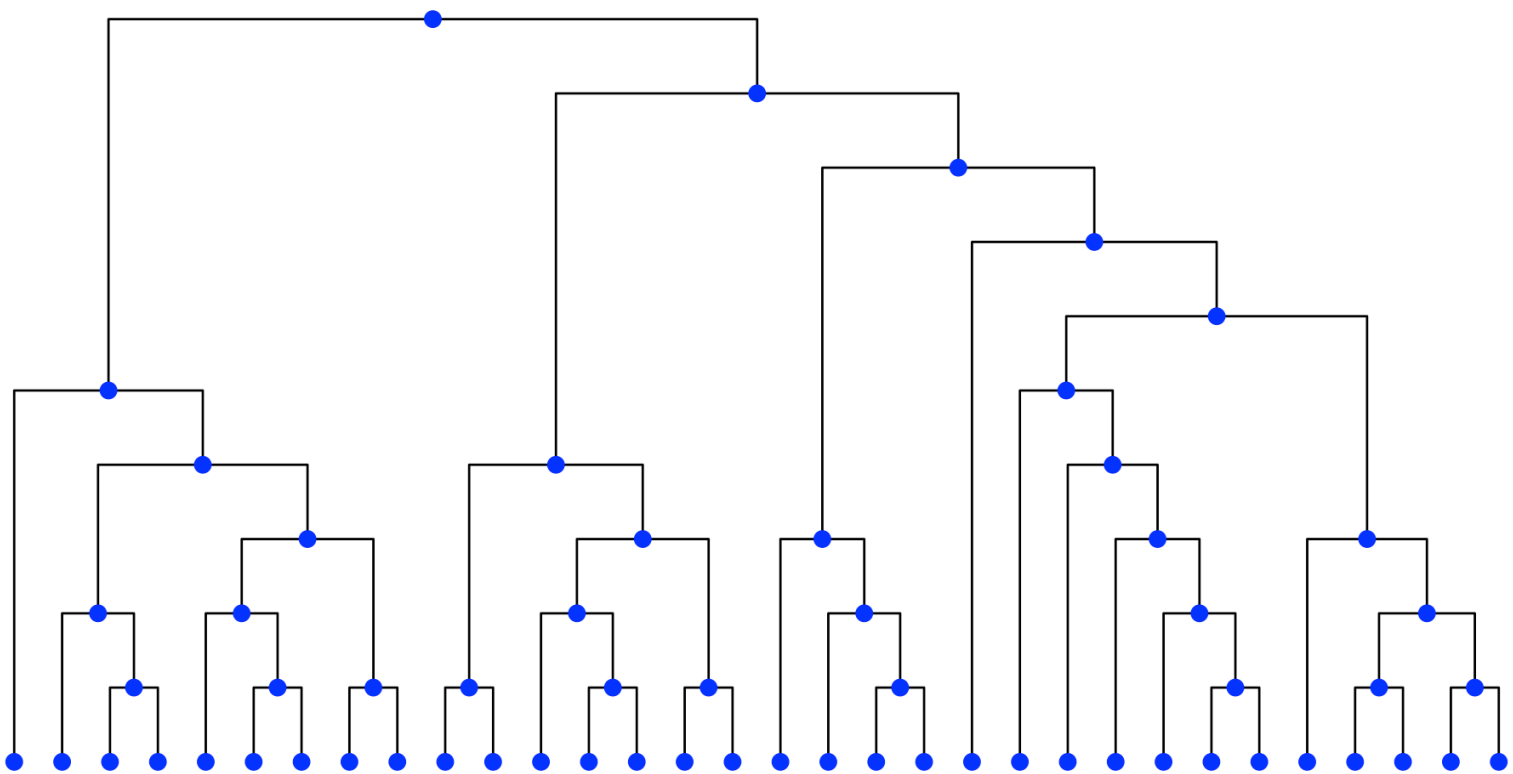
Creating a Dendrogram Layout

```
hclust_gr = as_tbl_graph(hclust(dist(mtcars)))
ggraph(hclust_gr, layout = 'dendrogram') +
  geom_edge_link() +
  geom_node_point(size = 3, color = 'blue') +
  theme_void()
```



Creating a Dendrogram Layout: Edges

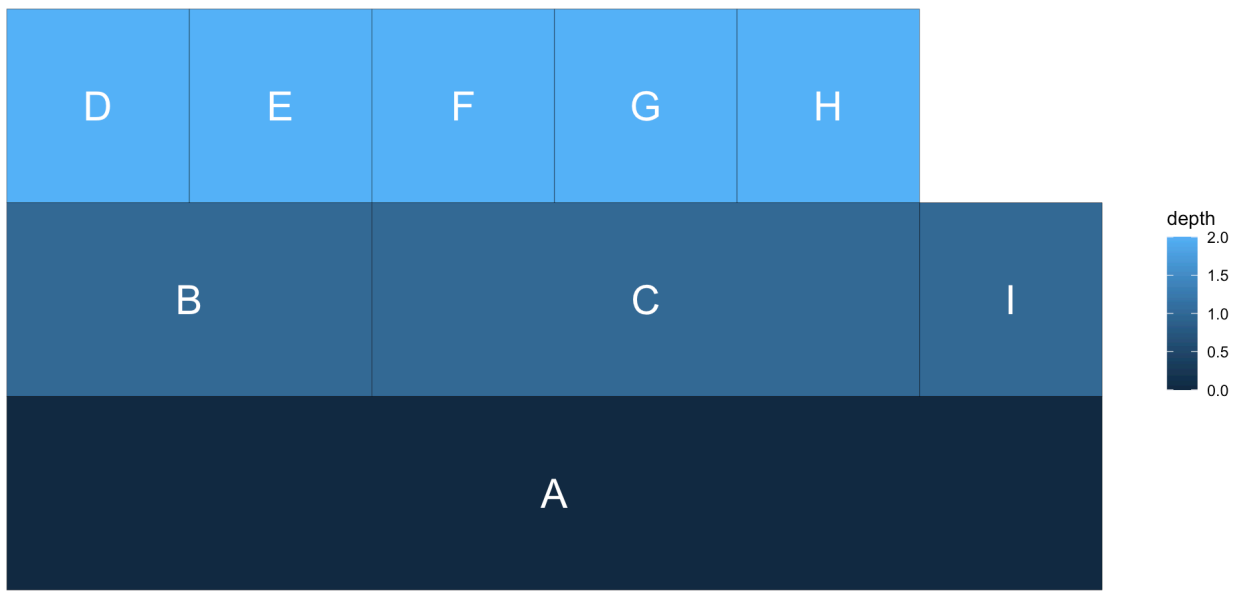
```
hclust_gr = as_tbl_graph(hclust(dist(mtcars)))
ggraph(hclust_gr, layout = 'dendrogram') +
  geom_edge_elbow() +
  geom_node_point(size = 3, color = 'blue') +
  theme_void()
```



Icicle Plots

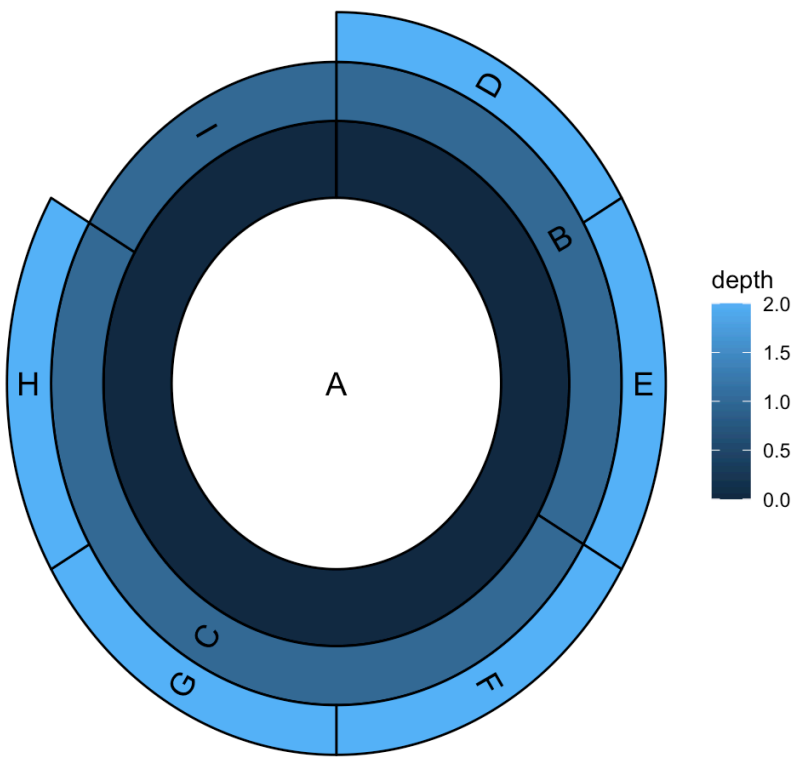
Nodes are represented as rectangles, with the area or position indicating relationships.

```
ggraph(graph, layout = 'partition') +
  geom_node_tile(aes(fill = depth)) +
  geom_node_text(aes(label = name), color = "white", size = 8) +
  theme_void()
```



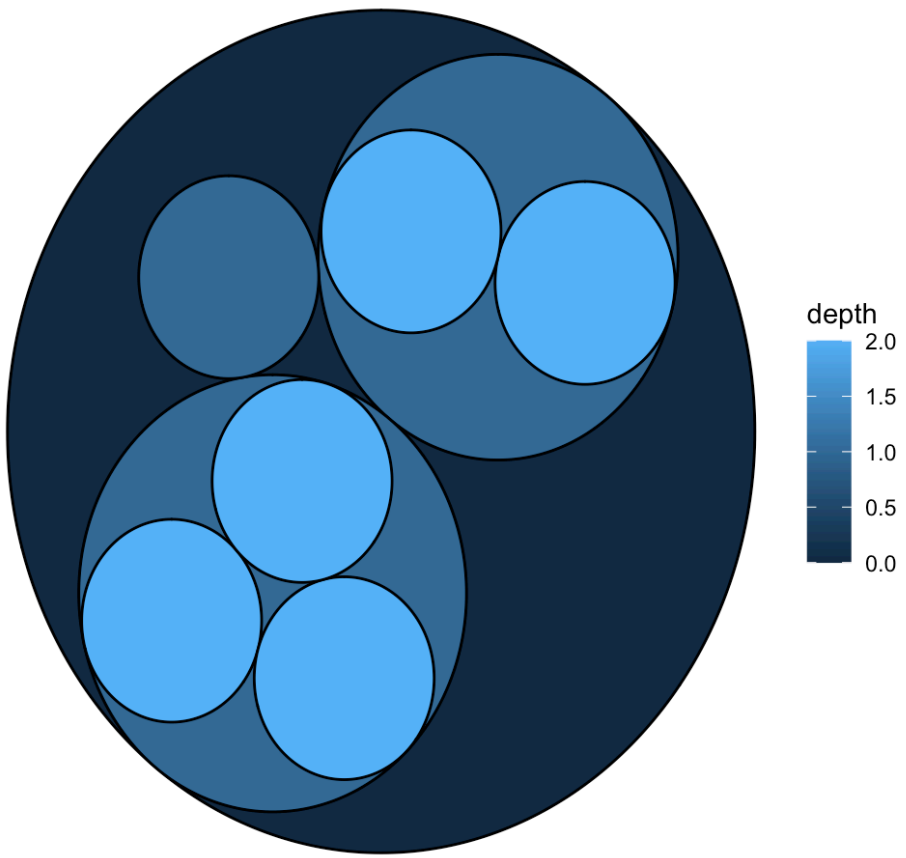
Sunburst Plots

```
ggraph(graph, layout = 'partition', circular = TRUE) +
  geom_node_arc_bar(aes(fill = depth)) +
  geom_node_text(aes(label = name, angle = node_angle(x, y)),
                 size = 5) +
  theme_void()
```



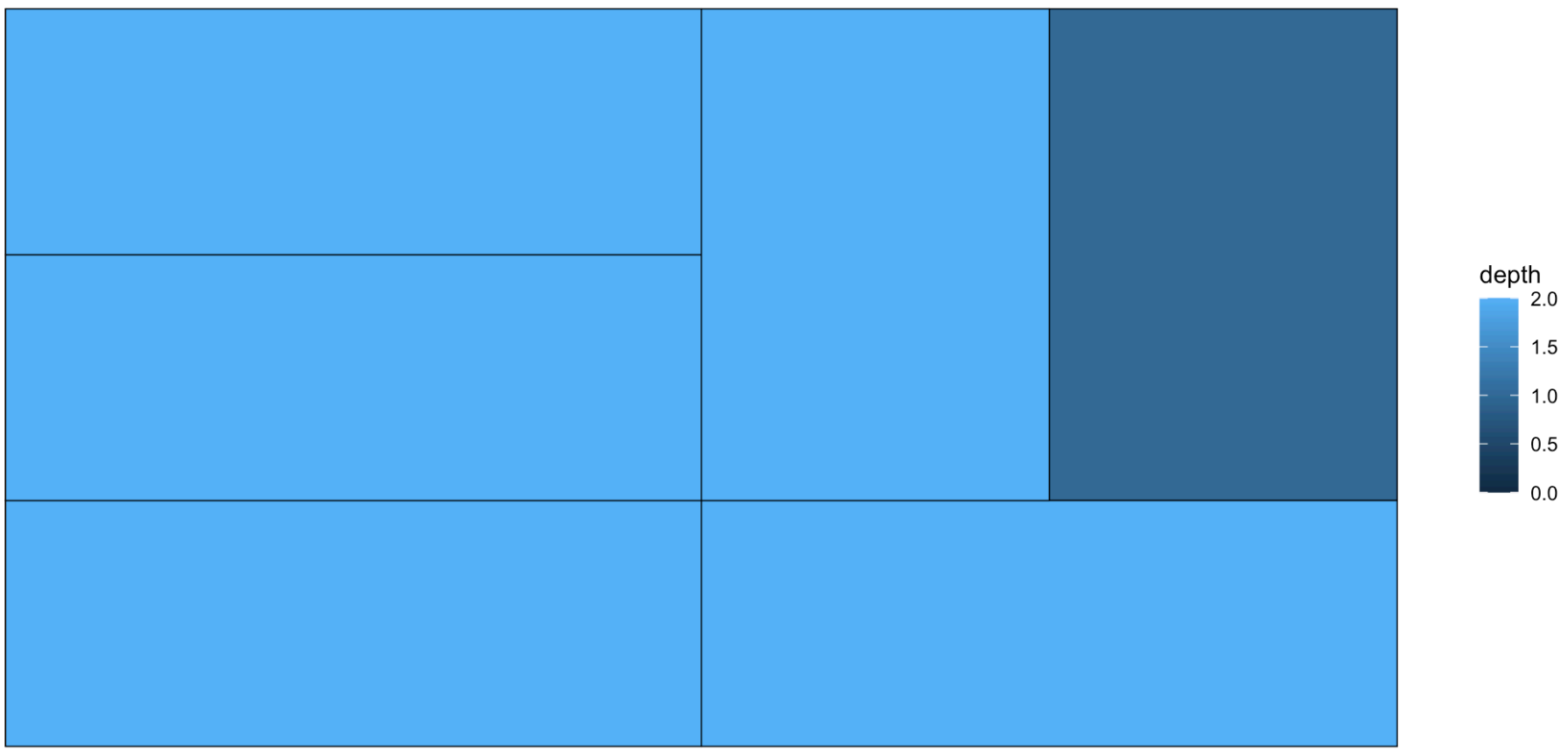
Circle Packs

```
ggraph(graph, layout = 'circlepack') +
  geom_node_circle(aes(fill = depth)) +
  theme_void()
```



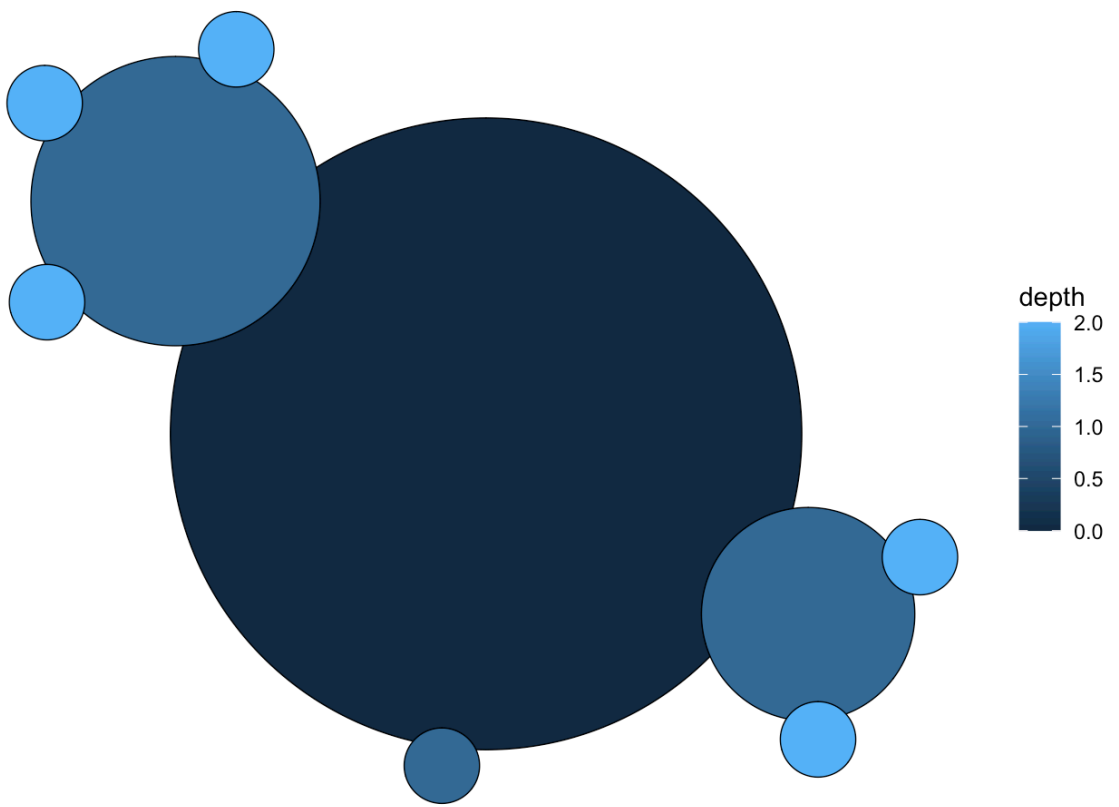
Tree Maps

```
ggraph(graph, layout = 'treemap') +
  geom_node_tile(aes(fill = depth), size = 0.25) +
  theme_void()
```

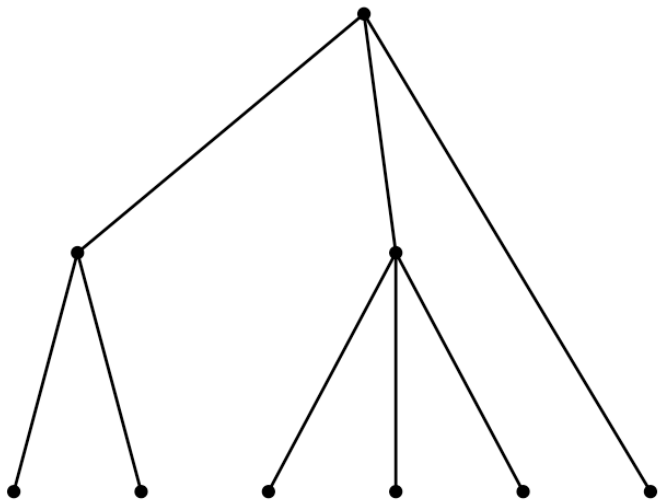


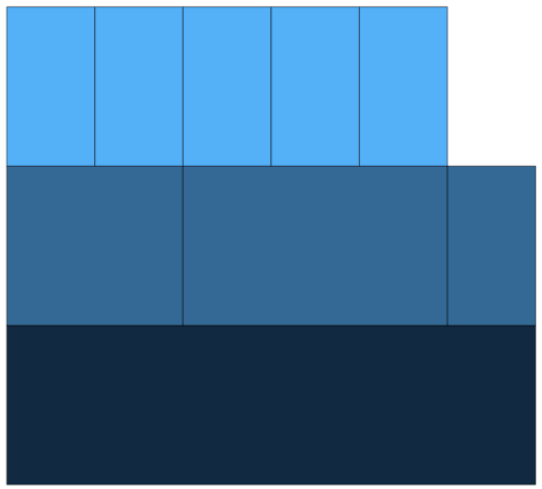
Cactustree

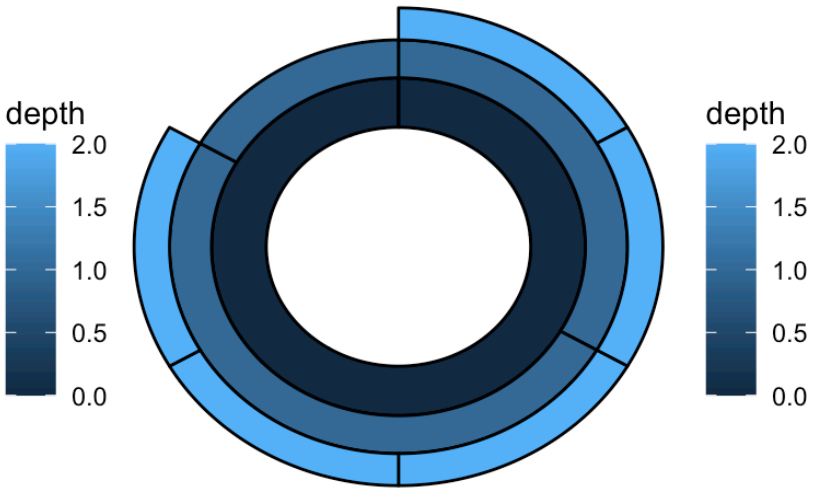
```
ggraph(graph, layout = 'cactustree') +
  geom_node_circle(aes(fill = depth), size = 0.25) +
  coord_fixed() +
  theme_void()
```

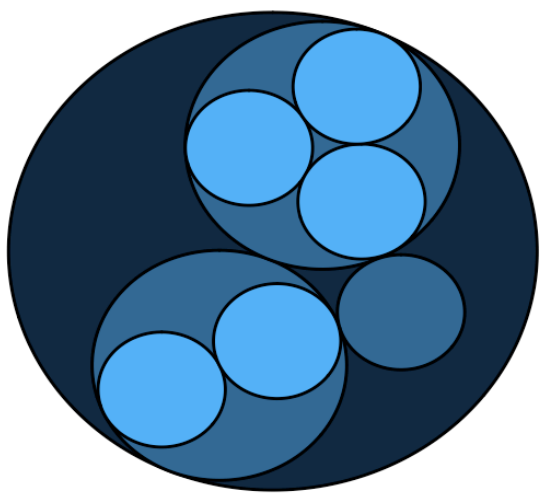


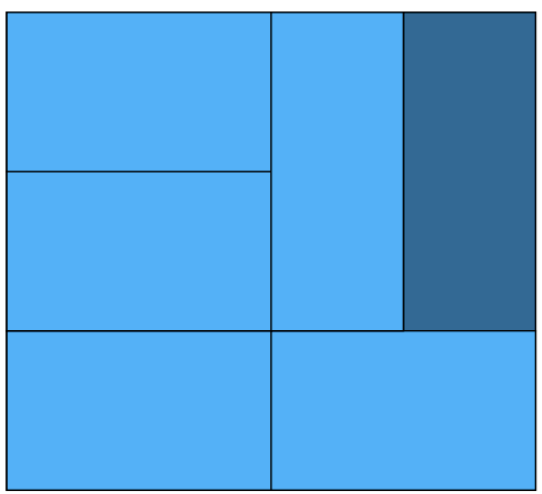
A Comparison

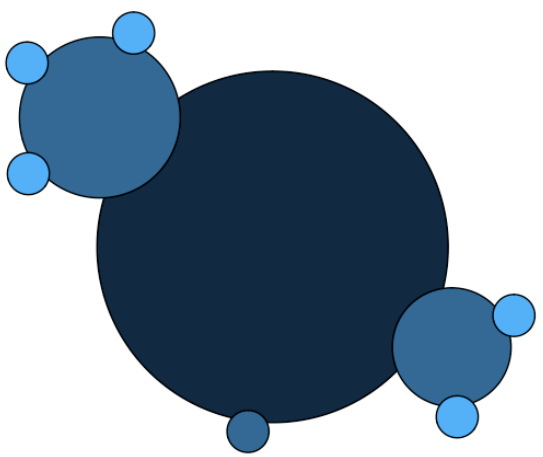












Thank You!

Email: naman.agr03@gmail.com