Graph Theory and Analytics in R

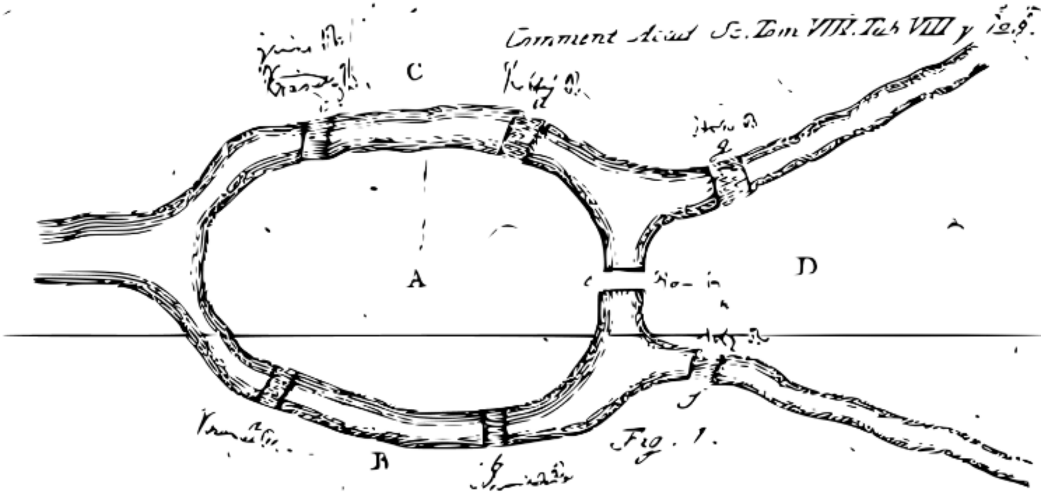
David Li

The goals of today

- Introduction to Graph Theory
  - Vertex
  - Edge
  - Degree
- Graph Analytics in R with Spark
  - Create graphframes
  - Run functions to get answers
- Apply the graph analytics in real word problems
  - Social media network analysis
    - Facebook/WeChat/Linkedin/YouTube
  - Investment banking analysis
    - Peer selection/Pitching/M&A

Introduction to Graph Theory

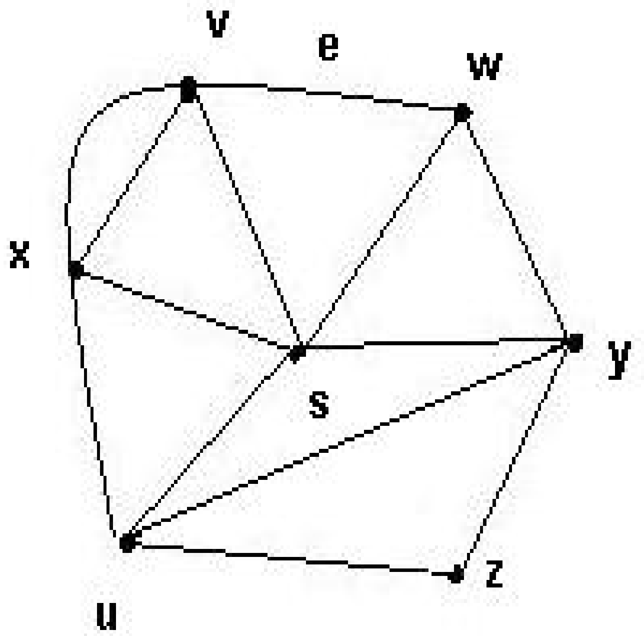
The Seven Bridges of Königsberg



- 1736: **Leonhard Euler**
  - Basel, 1707-St. Petersburg, 1786
  - He wrote *A solution to a problem concerning the geometry of a place.* First paper in graph theory.
- Problem of the Königsberg bridges:
  - Starting and ending at the same point, is it possible to cross all seven bridges just once and return to the starting point?

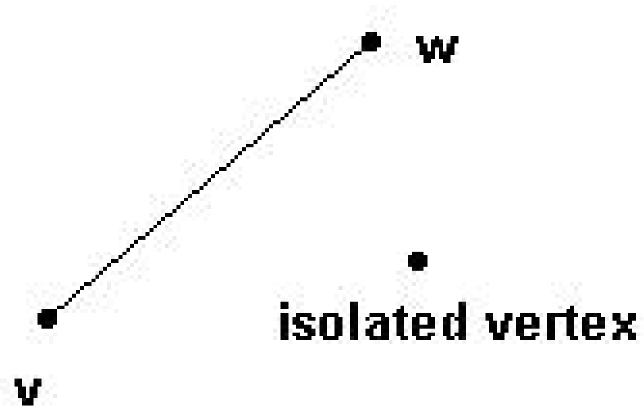
Basic Concepts of Graph

- What is a graph G?
- It is a pair G = (V, E), where
  - V = V(G) = set of vertices
  - E = E(G) = set of edges
- **Example:**
  - V = {s, u, v, w, x, y, z}
  - E = {(x,s), $(x,v)_1$, $(x,v)_2$, (x,u), (v,w), (s,v), (s,u), (s,w), (s,y), (w,y), (u,y), (u,z),(y,z)}



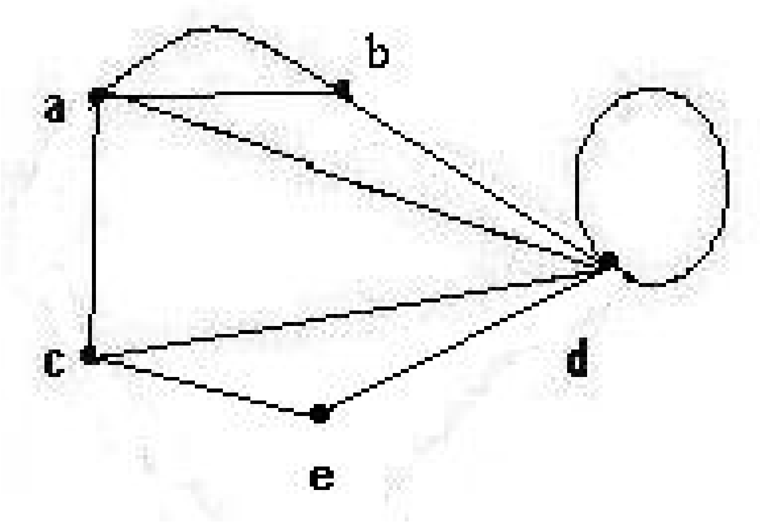
Edges

- An edge may be labeled by a pair of vertices, for instance e = (v,w).
- e is said to be *incident* on v and w.
- Isolated vertex = a vertex without incident edges.



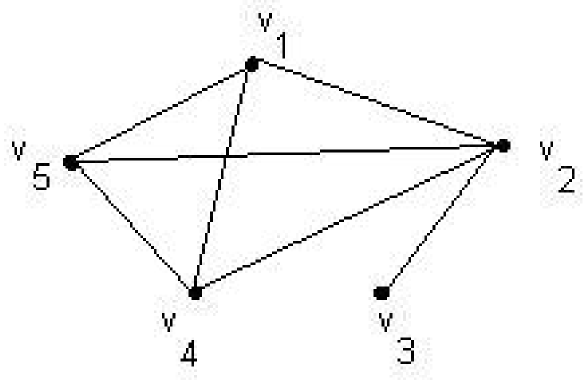
Special edges

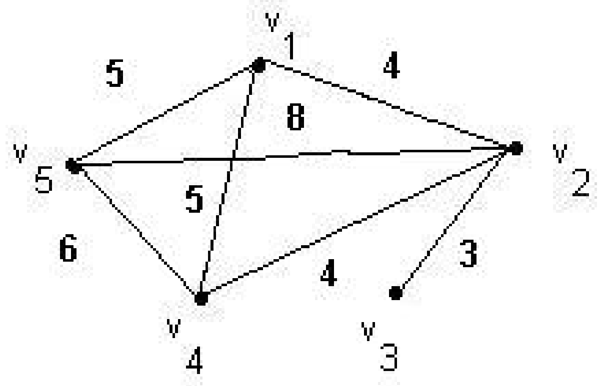
- Parallel edges
  - Two or more edges joining a pair of vertices
    - in the example, **a** and **b** are joined by two parallel edges
- Loops
  - An edge that starts and ends at the same vertex
    - In the example, vertex **d** has a loop



Special graphs

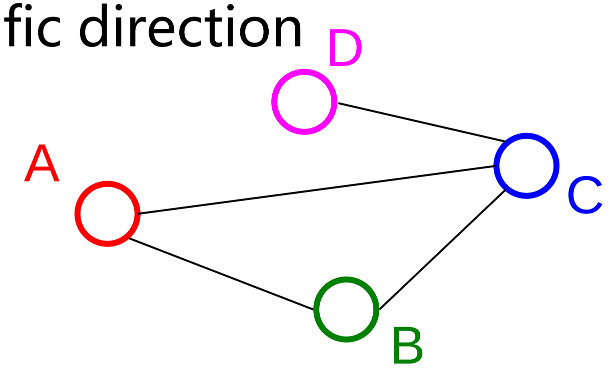
- Simple graph
  - A graph without loops or parallel edges.
- Weighted graph
  - A graph where each edge is assigned a numerical label or “weight”.
  - Examples:
    - Shipping price between cities,
    - ping round trip time between IPs,
    - exchange rate between currencies





Undirected Graphs and Degree

In undirected graphs, edges have no specific direction
- Edges are always "two-way "
  - Such as "friends" relationship

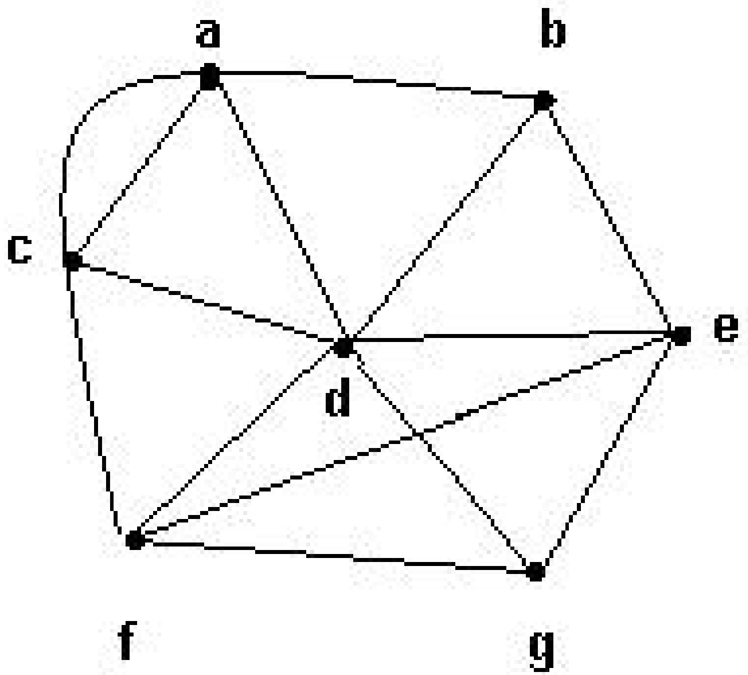


Thus, $(u, v) \in E$ implies $(v, u) \in E$.
- Only one of these edges needs to be in the set
- The other is implicit, so normalize how you check for it

Degree of a vertex: number of edges containing that vertex
- Put another way: the number of adjacent vertices

Degree of a vertex in undirected graphs

- The *degree* of a vertex v, denoted by $\delta(v)$, is the number of edges incident on v
- Example:
  - $\delta(a) = 4$, $\delta(b) = 3$,
  - $\delta(c) = 4$, $\delta(d) = 6$,
  - $\delta(e) = 4$, $\delta(f) = 4$,
  - $\delta(g) = 3$.



The Properties of Degree in undirected graphs

- The degree of a vertex deg(v) is a number of edges that have vertex v as an endpoint. Loop edge gives vertex a degree of 2
- In any graph the sum of degrees of all vertices equals twice the number of edges
- The total degree of a graph is even
- In any graph there are even number of vertices of odd degree

Sum of the degrees of a graph

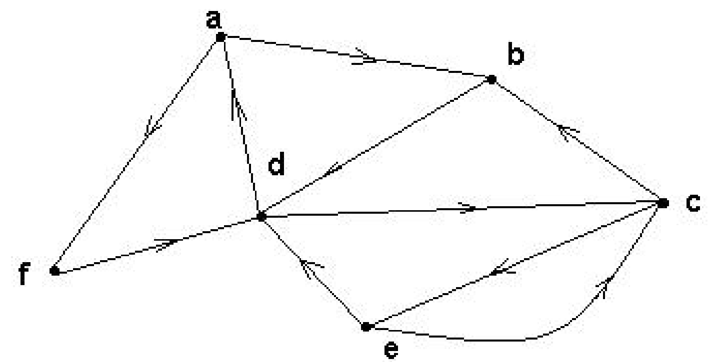
If G is a graph with m edges and n vertices $v_1, v_2, \ldots, v_n$, then

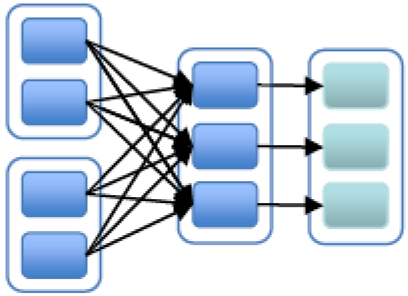
$$\sum_{i=1}^{n} \delta(v_i) = 2m$$

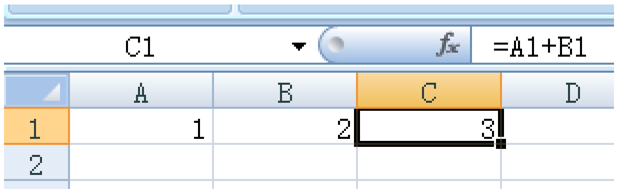
In particular, the sum of the degrees of all the vertices of a graph is even.

Directed graphs (digraphs)

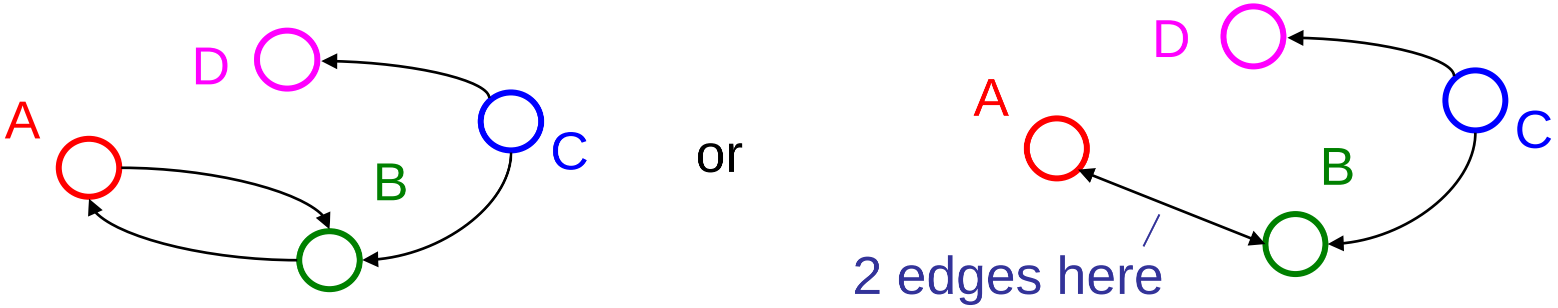
- G is a *directed graph* or *digraph*
  - if each edge has been associated with an ordered pair of vertices, i.e. each edge has a direction
  - Examples:
    - Spark tasks dependency graph
    - Excel Spreadsheet formulas
    - Family trees







Directed Graphs



In directed graphs (or digraphs), edges have direction

Thus, $(u, v) \in E$ does not imply $(v, u) \in E$.

- –such as "follow" in Facebook
- –Or "love"

Directed Graphs

Let $(u, v) \in E$ mean $u \rightarrow v$

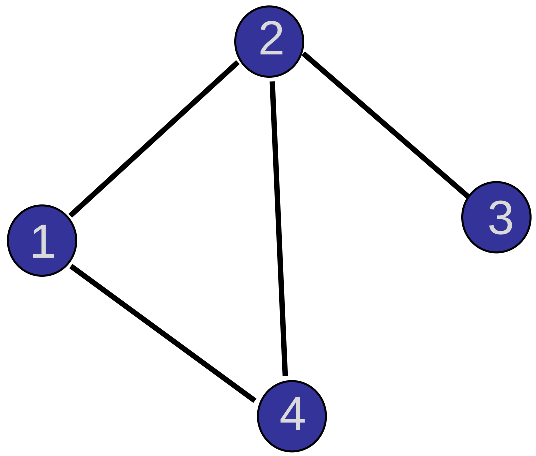
- Call u the source and v the destination

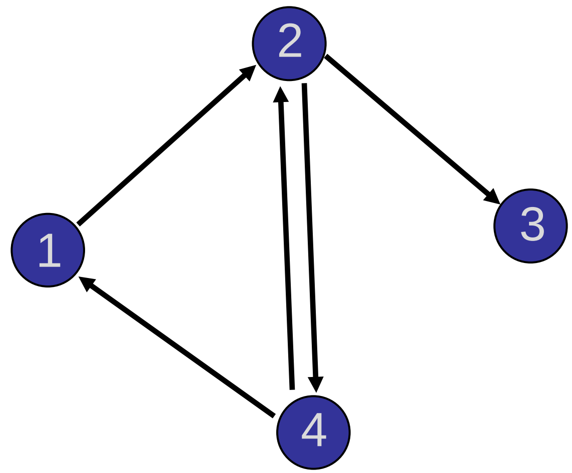
Degree in Directed Graphs

- In-Degree of a vertex: number of in-bound edges (edges where the vertex is the destination)
- Out-Degree of a vertex: number of out-bound edges (edges where the vertex is the source)

Degree in undirected graph and directed graph

- Degree of a vertex in an undirected graph is the number of edges incident on it.
- In a directed graph, the out degree of a vertex is the number of edges leaving it and the in degree is the number of edges entering it





The *degree* of vertex 2 is 3

The *in degree* of vertex 2 is 2 and the *in degree* of vertex 4 is 1

What does “Degree” mean in real world?

- Social media network
  - Popularity
  - Influence
  - Small-world effect
  - Marketing opportunities
- Search engine
  - Result ranking
  - Accuracy
- Investment Banking
  - Market analysis
  - Peer company analysis

Paths and Circuits

- A walk in a graph is an alternating sequence of adjacent vertices and edges
- A path is a walk that does not contain a repeated edge
- Simple path is a path that does not contain a repeated vertex
- A closed walk is a walk that starts and ends at the same vertex
- A circuit is a closed walk that does not contain a repeated edge
- A simple circuit is a circuit which does not have a repeated vertex except for the first and last

Shortest-path algorithm

- We say "a path exists from v_0 to v_n" if there is a list of vertices $[v_0, v_1, \ldots, v_n]$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < n$.

- Shortest path
  - The shortest path length between two vertices *i* and *j* is the number of edges comprising the shortest path (or a shortest path) between *i* and *j*.

- Due to Edsger W. Dijkstra, Dutch computer scientist born in 1930
- Dijkstra's algorithm finds the length of the shortest path from a single vertex to any other vertex in a connected weighted graph.
- For a simple, connected, weighted graph with n vertices, Dijkstra's algorithms has worst-case run time $\Theta(n^2)$.

Shortest-path in real world

- Flight ticket with minimum stops
- Amazon Logistics
- GPS direction
- The flow of data in a network
- Social engineering

Graph Analytics in R with Spark

Install libraries

```
install.packages("ggraph")
install.packages("igraph")
install.packages("graphframes")
```

```
library(sparklyr)
library(dplyr)
library(ggraph)
library(igraph)
library(graphframes)
```

- we use the highschool dataset from the ggraph package, which tracks friendship among high school boys.
- igraph is a library collection for creating and manipulating graphs and analyzing networks
- graphframes is a package for 'Apache Spark' that provides a DataFrame-based API for working with graphs.

- *Assume you already have sparklyr and dplyr*

Start from dataframe

```
head(highschool)
```

A data.frame: 6 × 3

	from	to	year
	<dbl>	<dbl>	<dbl>
1	1	14	1957
2	1	15	1957
3	1	21	1957
4	1	54	1957
5	1	55	1957
6	2	21	1957

- The vertices are the students and the edges describe pairs of students who happen to be friends in a particular year
- “from” and “to” are the student IDs
- “year” indicates the year when the two students were friends
- “friends” relationship is undirected edge, so we will build an undirected graph
- “friends” in our example doesn’t indicate weight, so the graph will be a simple undirected graph

The questions we want to answer

- From the dataframe, we can write R code to answer some basic questions
  - Who was the popular star in a particular year?
    - *in graph theory, find the vertices with <u>maximum degree</u> in a particular year*
  - How friendly was the atmosphere in a particular year?
    - *in graph theory, find the <u>average degree</u> in a particular year*

- The R code will have loops through the rows
  - slow
  - hard to reuse on other problems

Why need Spark?

- When the high school dataset is small, it can be processed in non-trivial R code (what's the time complexity?)

- Medium-size graph datasets can be very difficult to process

- Spark is well suited here to distribute the computation across a cluster of machines

- Spark supports processing graphs through the graphframes extension, which in turn uses the GraphX Spark component.

- GraphX is Apache Spark's API for graphs and graph-parallel computation. It's comparable in performance to the fastest specialized graph-processing systems and provides a growing library of graph algorithms.

Distributed dataframe in Spark

```
sc=spark_connect(master = "local")
highschool_tbl <- copy_to(sc, highschool, "highschool", overwrite = TRUE)
```

```
highschool_tbl
```

```
# Source: spark<highschool> [?? x 3]
    from    to  year
   <dbl> <dbl> <dbl>
 1     1    14  1957
 2     1    15  1957
 3     1    21  1957
 4     1    54  1957
 5     1    55  1957
 6     2    21  1957
 7     2    22  1957
 8     3     9  1957
 9     3    15  1957
10     4     5  1957
# ... with more rows
```

Same data, now distributed in Spark

Prepare data for graph

```
highschool_tbl <- copy_to(sc,
                          highschool,
                          "highschool",
                          overwrite = TRUE) %>%
  filter(year == 1957) %>%
  transmute(from = as.character(as.integer(from)),
            to = as.character(as.integer(to)))
```

- We are only interested in the year of 1957
- We only need the “from” and “to” to build the simple undirected graph

```
highschool_tbl
```

```
# Source: spark<?> [?? x 2]
   from  to
   <chr> <chr>
 1 1     14
 2 1     15
 3 1     21
 4 1     54
 5 1     55
 6 2     21
 7 2     22
 8 3     9
 9 3     15
10 4     5
# ... with more rows
```

Building graph step 1: vertices

```
from_tbl <- highschool_tbl %>%
    distinct(from) %>%
    transmute(id = from)
from_tbl
```

```
# Source: spark<?> [?? x 1]
   id
   <chr>
 1 1
 2 3
 3 4
 4 8
 5 12
 6 17
 7 20
 8 27
 9 29
10 31
# ... with more rows
```

```
to_tbl <- highschool_tbl %>%
    distinct(to) %>%
    transmute(id = to)
to_tbl
```

```
# Source: spark<?> [?? x 1]
   id
   <chr>
 1 43
 2 20
 3 17
 4 12
 5 8
 6 4
 7 27
 8 60
 9 65
10 68
# ... with more rows
```

```
vertices_tbl <- distinct(sdf_bind_rows(from_tbl, to_tbl))
vertices_tbl
```

```
# Source: spark<?> [?? x 1]
   id
   <chr>
 1 1
 2 3
 3 4
 4 8
 5 12
 6 17
 7 20
 8 27
 9 29
10 31
# ... with more rows
```

- Collect all unique IDs from “from” and “to” columns.
- They are vertices in our graph.
- All functions above run on Spark, so the performance will be good on big data

Building graph step 2: edges

```
edges_tbl <- highschool_tbl %>%
    transmute(src = from, dst = to)
edges_tbl
```

```
# Source: spark<?> [?? x 2]
   src   dst
   <chr> <chr>
 1 1     14
 2 1     15
 3 1     21
 4 1     54
 5 1     55
 6 2     21
 7 2     22
 8 3     9
 9 3     15
10 4     5
# ... with more rows
```

- Just rename the columns to “src” and “dst”
- Each row represents an edge
- Undirected, so it doesn’t matter which column is called “src”.

Building graph step 3: graph

```
graph <- gf_graphframe(vertices_tbl, edges_tbl)
graph
```

```
GraphFrame
Vertices:
  Database: spark_connection
  $ id <chr> "1", "3", "4", "8", "12", "17", "20", "27", "29", "31", "34", "3...
Edges:
  Database: spark_connection
  $ src <chr> "1", "1", "1", "1", "1", "2", "2", "3", "3", "4", "4", "4", "4"...
  $ dst <chr> "14", "15", "21", "54", "55", "21", "22", "9", "15", "5", "18",...
```

- Use vertices and edges to build a graph
- graphframe is stored in a distributed mode on Spark cluster machines

Ready to answer questions

- Who was the popular star in a particular year?
  - *in graph theory, find the vertices with <u>maximum degree</u> in a particular year*

```
gf_degrees(graph)
```

```
# Source: spark<?> [?? x 2]
   id    degree
   <chr>  <int>
 1 1          5
 2 3          2
 3 4          5
 4 43        12
 5 20        11
 6 17         5
 7 8          3
 8 12         5
 9 27         4
10 60         7
# ... with more rows
```

```
gf_degrees(graph) %>% arrange(desc(degree))
```

```
# Source:     spark<?> [?? x 2]
# Ordered by: desc(degree)
   id    degree
   <chr>  <int>
 1 71        16
 2 21        14
 3 22        14
 4 70        13
 5 54        13
 6 69        12
 7 43        12
 8 66        12
 9 52        11
10 67        11
# ... with more rows
```

Ready to answer questions

- How friendly was the atmosphere in a particular year?
  - *in graph theory, find the <u>average degree</u> in a particular year*

In the year of 1957

```
gf_degrees(graph) %>% summarise(friends = mean(degree, na.rm = TRUE))
```

```
# Source: spark<?> [?? x 1]
  friends
    <dbl>
1    6.94
```

In the year of 1958

```
gf_degrees(graph) %>% summarise(friends = mean(degree, na.rm = TRUE))
```

```
# Source: spark<?> [?? x 1]
  friends
    <dbl>
1    7.62
```

It shows the class of 1958 was more friendly than 1957!

Shortest-path question

- what are everybody's shortest distances from one student in a particular year?
  - *in graph theory, find the <u>shortest-path</u> from every vertex to a particular vertex in a particular year*

```
gf_shortest_paths(graph, 14) %>%
  filter(size(distances) > 0) %>%
  mutate(distance = explode(map_values(distances))) %>%
  select(id, distance)
```

```
# Source: spark<?> [?? x 2]
   id    distance
   <chr>    <int>
 1 17           2
 2 29           4
 3 34           6
 4 8            1
 5 13           3
 6 14           0
 7 22           6
 8 24           8
 9 28           6
10 38           7
# ... with more rows
```

```
filter(edges_tbl, dst==14)
```

```
# Source: spark<?> [?? x 2]
  src   dst
  <chr> <chr>
1 1     14
2 8     14
```

Shortest-path question

- Who has more influence in his friends circle?
  - *in graph theory, find the <u>average shortest-path</u> to a vertex in a particular year*

```
gf_shortest_paths(graph, 14) %>%
    filter(size(distances) > 0) %>%
    mutate(distance = explode(map_values(distances))) %>%
    select(id, distance) %>%
    summarise(influence = mean(distance, na.rm = TRUE))
```

```
# Source: spark<?> [?? x 1]
  influence
      <dbl>
1      4.96
```

```
gf_shortest_paths(graph, 33) %>%
    filter(size(distances) > 0) %>%
    mutate(distance = explode(map_values(distances))) %>%
    select(id, distance) %>%
    summarise(influence = mean(distance, na.rm = TRUE))
```

```
# Source: spark<?> [?? x 1]
  influence
      <dbl>
1      3.33
```

It shows that in the class of 1957, the student 33 has more influence (less average distance) than student 14 in their friends circle.

If I launch a marketing campaign, which student would be my target?

Summary

- Graphs are a powerful representation and have been studied deeply
- Many real world problems are fundamentally graph problems.
- R with Spark provides a high performance framework to solve all these types of problems.

- Read more
    - https://therinspark.com/extensions.html#graphs
    - https://spark.rstudio.com/graphframes/
    - https://github.com/rstudio/graphframes

- Exercise of Today
    - Follow the jupyter notebook to install libs and run the sample code