GRAPH THEORY [1]

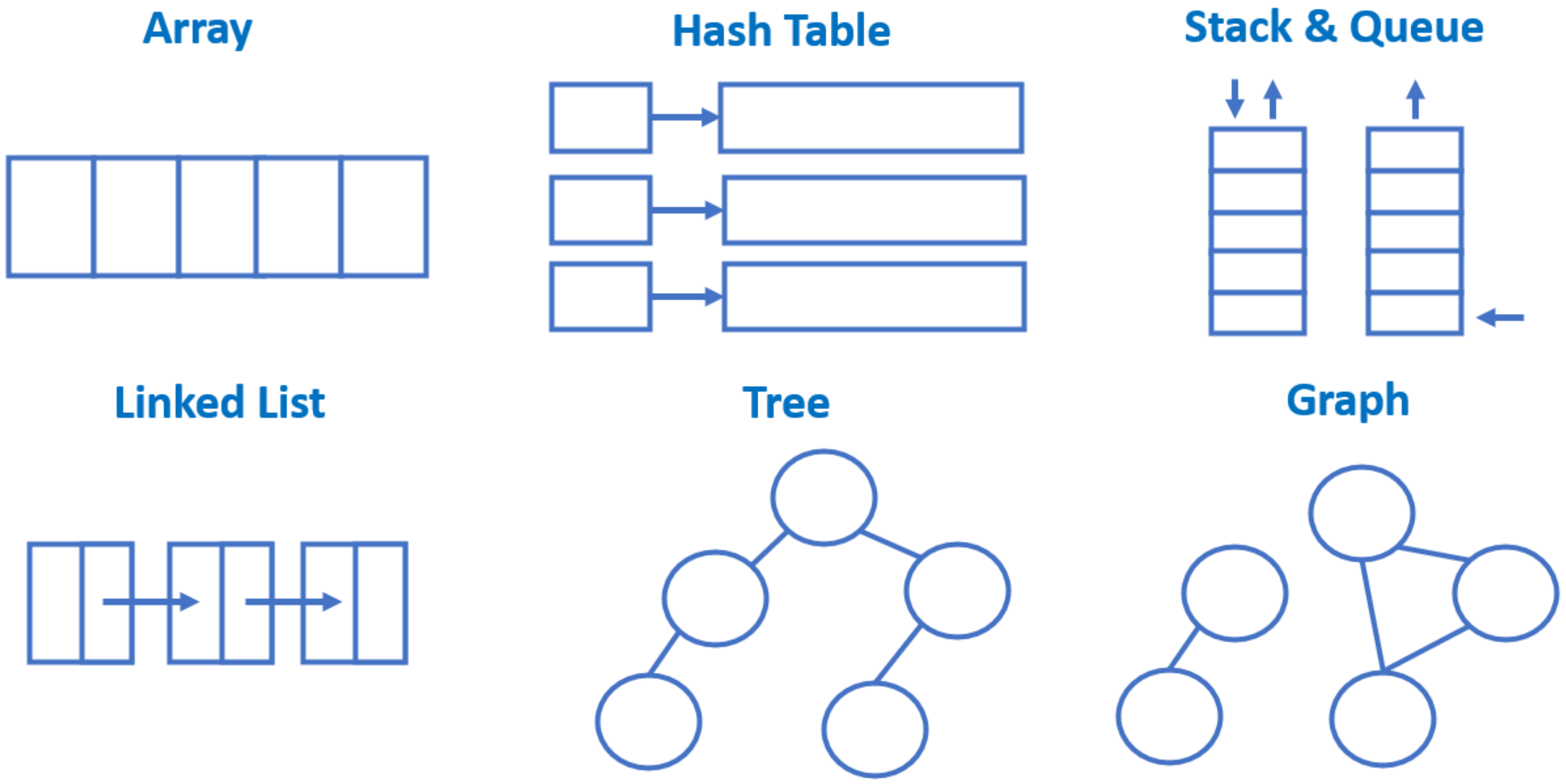
Introduction

Documents are here:

https://www-l2ti.univ-paris13.fr/~viennet/ens/2024-USTH-Graphs

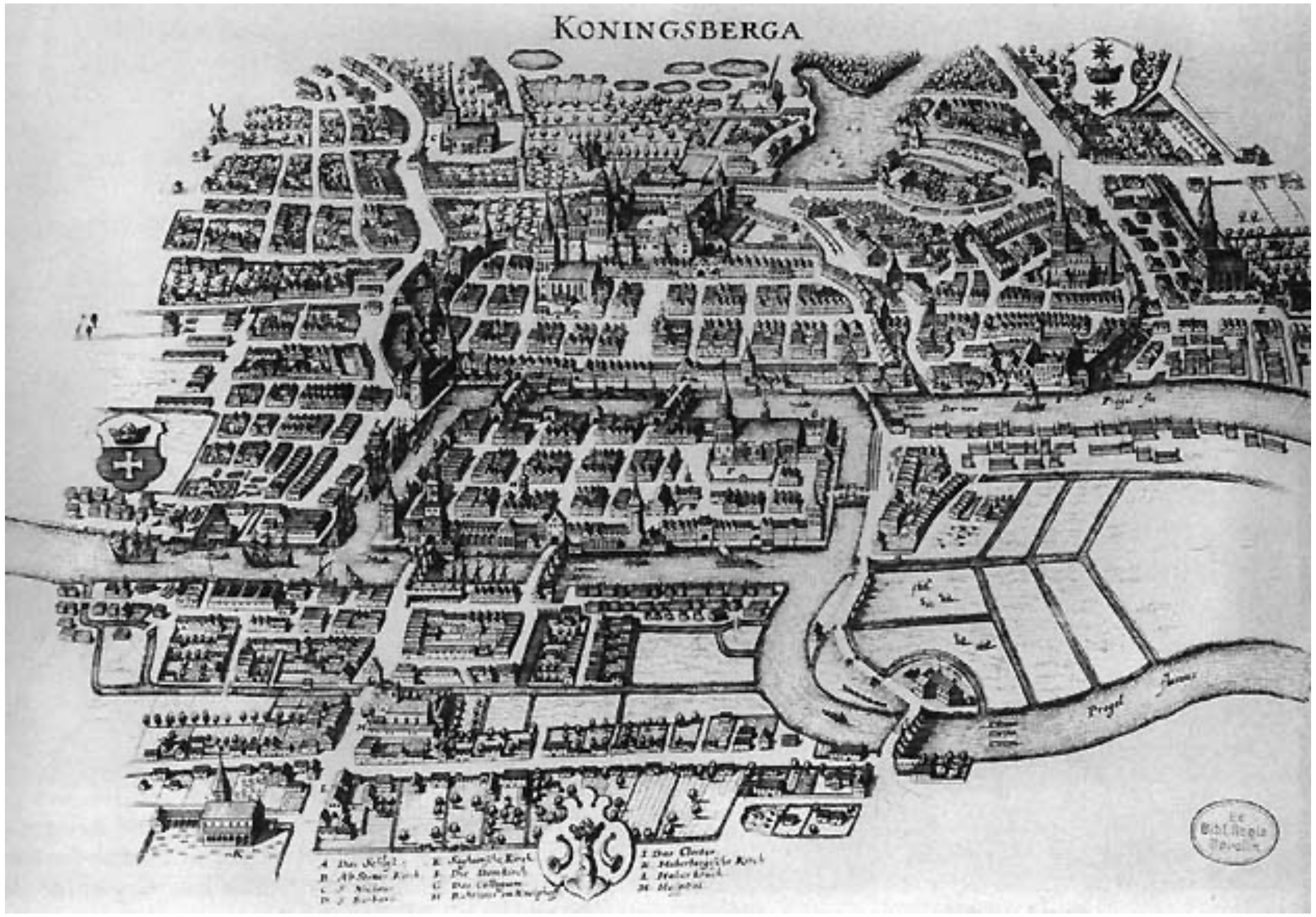
Emmanuel Viennet
emmanuel.viennet@univ-paris13.fr

Graphs are a data structure

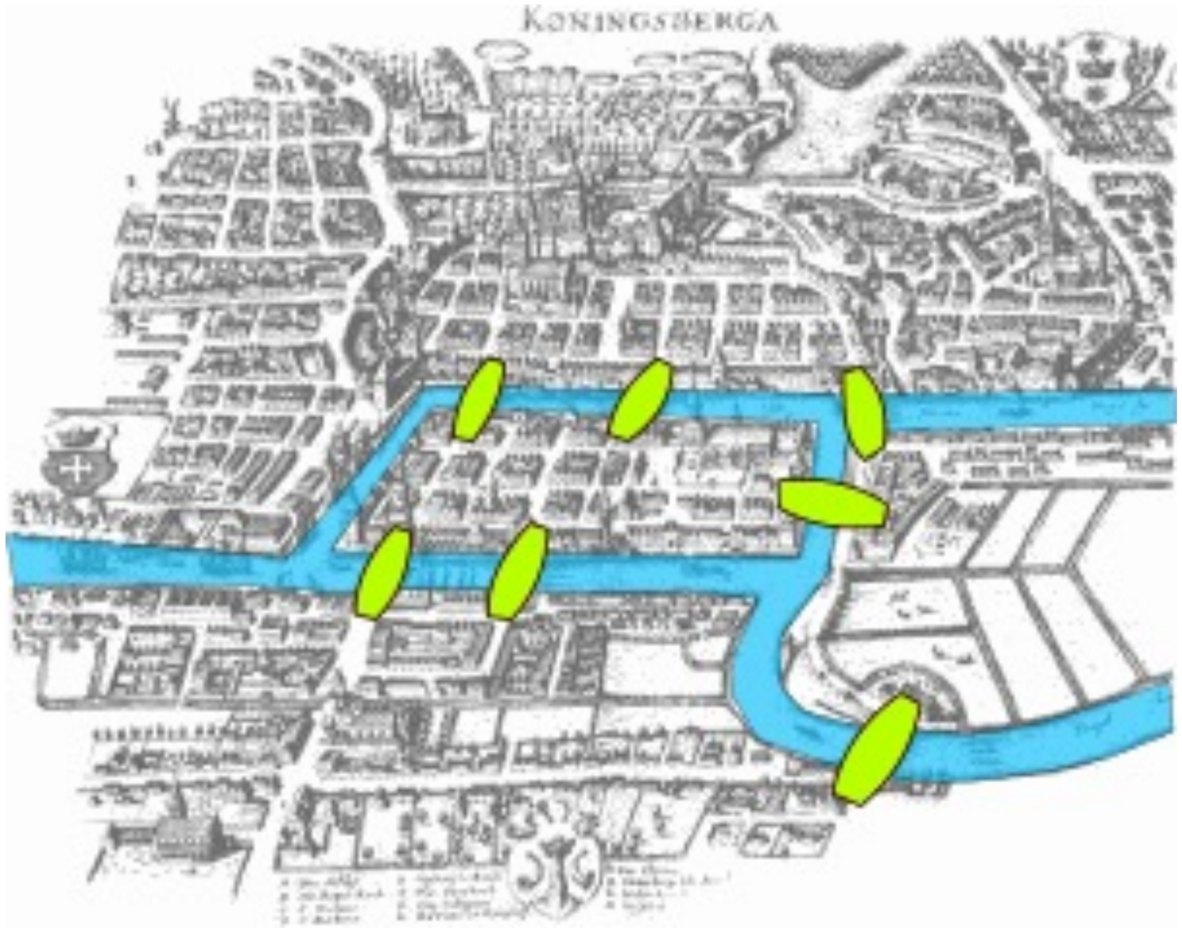


https://medium.com/codex/a-dummys-guide-to-linked-lists-part-1-44469f35f65a

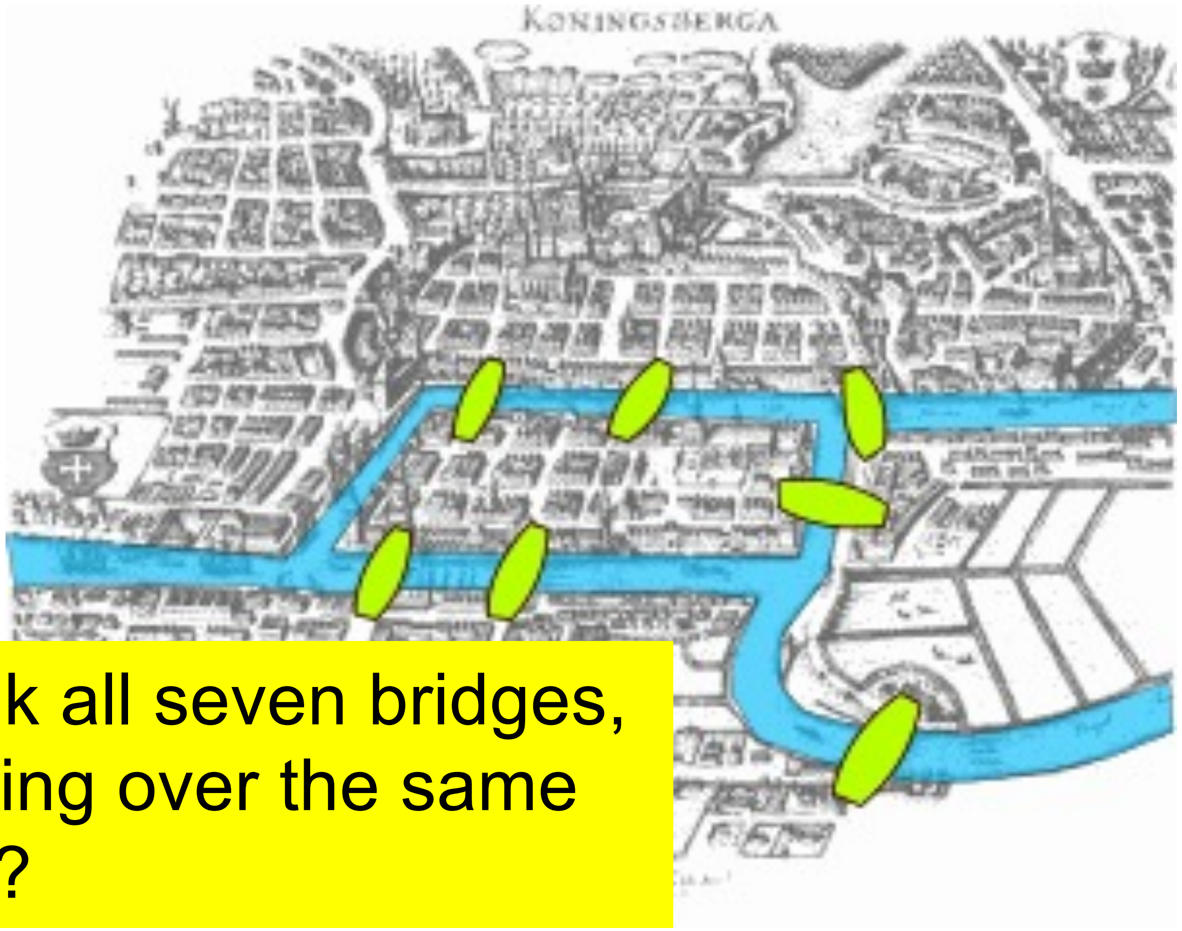
Seven Bridges of Königsberg



Seven Bridges of Königsberg

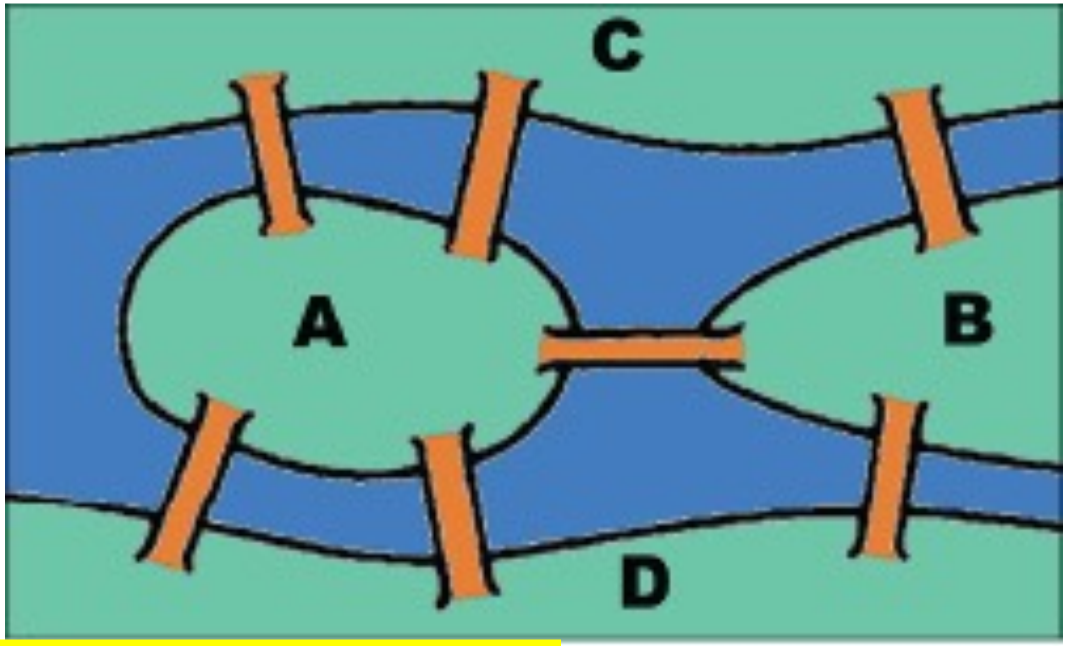


Seven Bridges of Königsberg



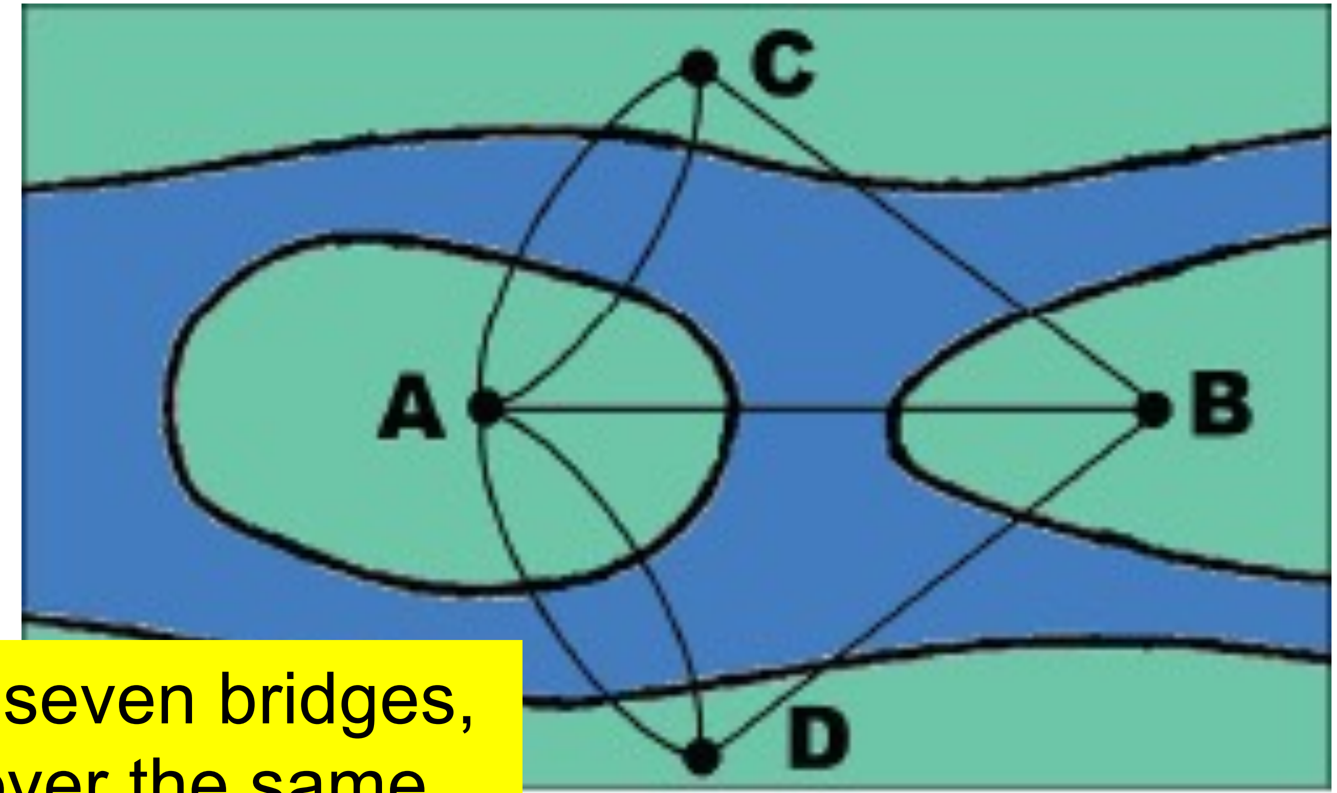
Can you walk all seven bridges, without walking over the same bridge twice?

Seven Bridges of Königsberg



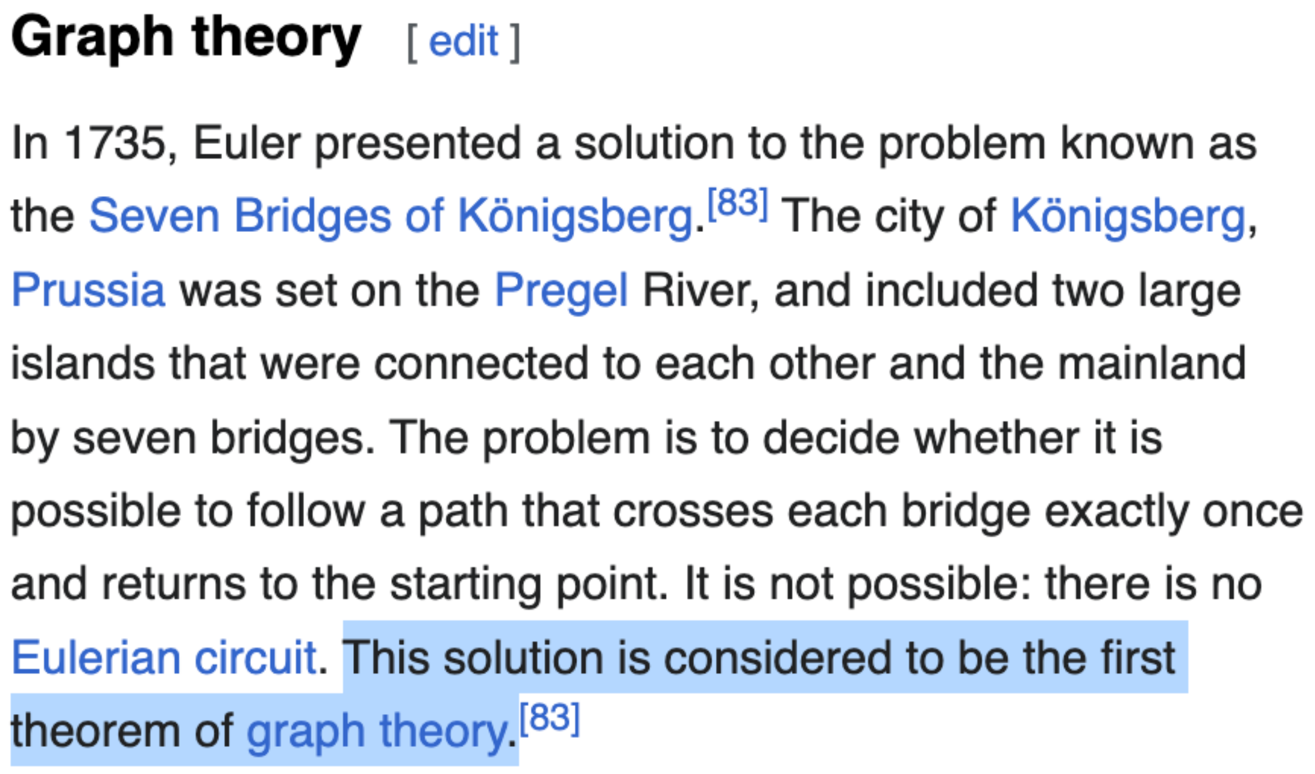
Can you walk all seven bridges, without walking over the same bridge twice?

Seven Bridges of Königsberg



Can you walk all seven bridges, without walking over the same bridge twice?

Leonhard Euler 1707-1783, German

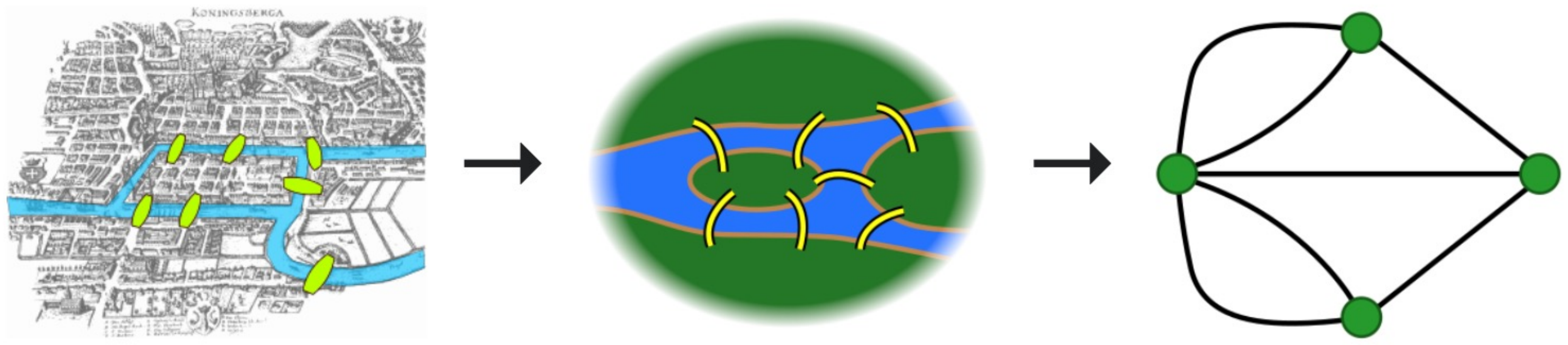


Graph theory [edit]

In 1735, Euler presented a solution to the problem known as the Seven Bridges of Königsberg.[83] The city of Königsberg, Prussia was set on the Pregel River, and included two large islands that were connected to each other and the mainland by seven bridges. The problem is to decide whether it is possible to follow a path that crosses each bridge exactly once and returns to the starting point. It is not possible: there is no Eulerian circuit. This solution is considered to be the first theorem of graph theory.[83]

https://en.wikipedia.org/wiki/Leonhard_Euler#Graph_theory

Seven Bridges of Königsberg



Can you walk all seven bridges, without walking over the same bridge twice?

Seven Bridges of Königsberg

Can you walk all seven bridges,
without walking over the same bridge twice?

You'd have to *start* from a node, *enter* another node AND *exit* it by another edge, …, and *terminate* on the starting node OR on another one.

So:
- intermediate nodes have necessarily an even number of edges (enter/exit x N)
- if ending at the starting node, this node must also have an **even number of edges**
- OR, if ending on another node, both START and END must have an ODD number of edges.

Seven Bridges of Königsberg

Can you walk all seven bridges,
without walking over the same bridge twice?

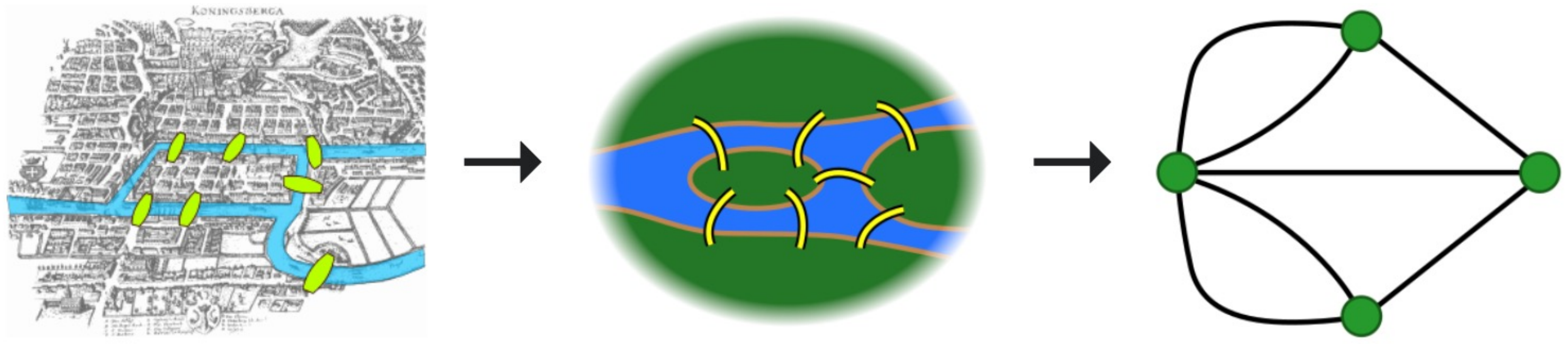
Euler's argument shows that a necessary condition for the walk of the desired form is that the graph
- be **connected**
- and have **exactly zero or two nodes of odd degree**.

This condition turns out also to be sufficient—a result stated by Euler and later proved by Carl Hierholzer.

Such a walk is now called an *Eulerian path* or *Euler walk* in his honor.

Further, if there are nodes of odd degree, then any Eulerian path will start at one of them and end at the other.

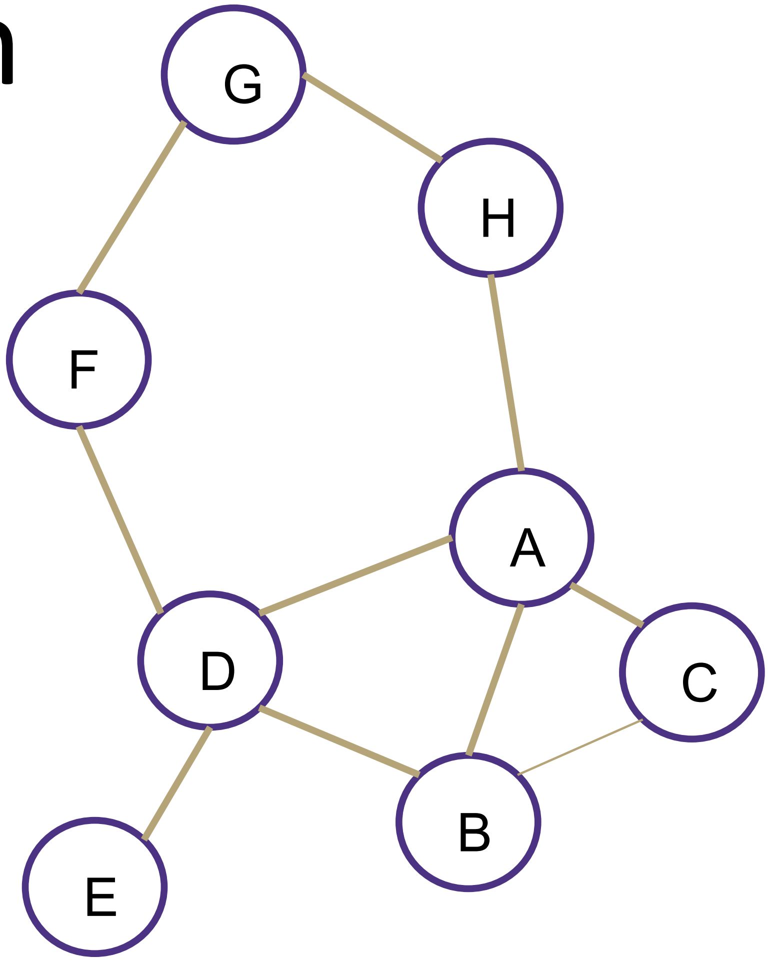
Seven Bridges of Königsberg



Can you walk all seven bridges, without walking over the same bridge twice?

No !
- connected, but
- 4 nodes with odd degree

Graph: Formal Definition



A **graph** is defined by a pair of sets G = (V, E) where…

- V is a set of **vertices (or nodes)**
  - A vertex or "node" is a data entity

V = { A, B, C, D, E, F, G, H }

- E is a set of **edges (or "links")**
  - An edge is a connection between two vertices

E = { (A, B), (A, C), (A, D), (A, H),
(C, B), (B, D), (D, E), (D, F), (F, G), (G, H)}

Graph elements: edges

Edges are also called arcs or links

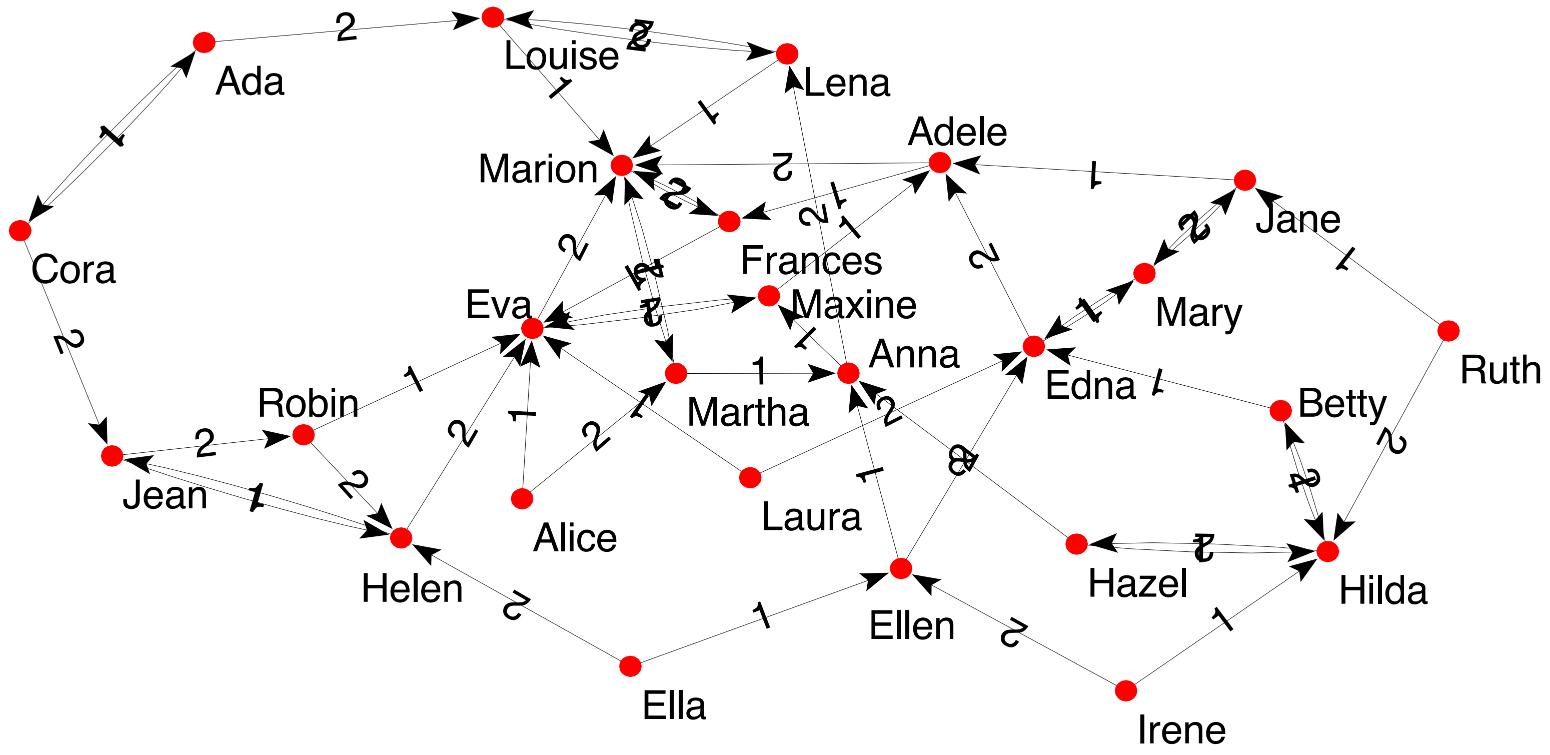
- **Directed**
  - A -> B
    - A likes B, A gave a gift to B, A is B’s child
- **Undirected**
  - A < – > B or A – B
    - A and B like each other
    - A and B are siblings
    - A and B are co-authors

Graph elements: edge’s attributes

- Examples
  - Weight (e.g. frequency of communication)
  - Ranking (best friend, second best friend…)
  - Type (friend, relative, co-worker)
  - Properties depending on the structure of the rest of the graph: e.g. betweenness

Example of directed graph

girls' school dormitory dining-table partners, 1st and 2nd choices
(Moreno, *The sociometry reader*, 1960)



Graphs Vocabulary

Graph Direction

- **Undirected graph** – edges have no direction and are two-way

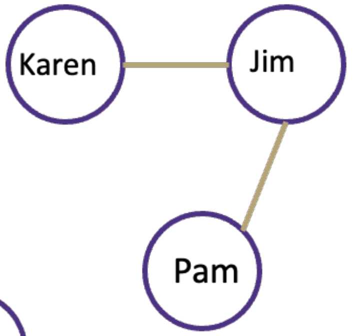
  V = { Karen, Jim, Pam }

  E = { (Jim, Pam), (Jim, Karen) } *inferred (Karen, Jim) and (Pam, Jim)*
- **Directed graphs** – edges have direction and are thus one-way

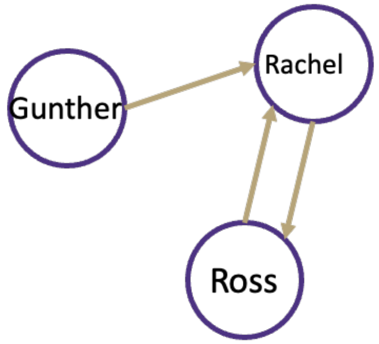
  V = { Gunther, Rachel, Ross }

  E = { (Gunther, Rachel), (Rachel, Ross), (Ross, Rachel) }

Undirected Graph:



Directed Graph:



Degree of a Vertex

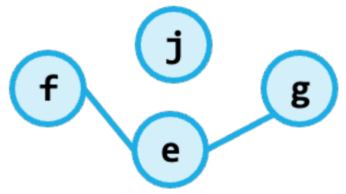
- **Degree** – the number of edges connected to that vertex

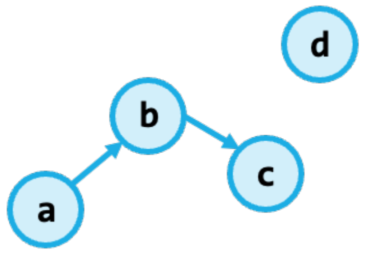
  Karen : 1, Jim : 1, Pam : 1
- **In-degree** – the number of directed edges that point to a vertex

  Gunther : 0, Rachel : 2, Ross : 1
- **Out-degree** – the number of directed edges that start at a vertex

  Gunther : 1, Rachel : 1, Ross : 1

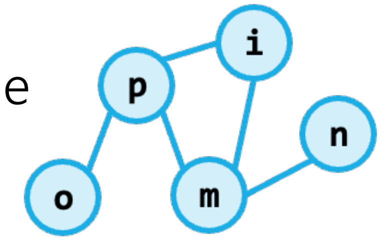
More Graph Terminology

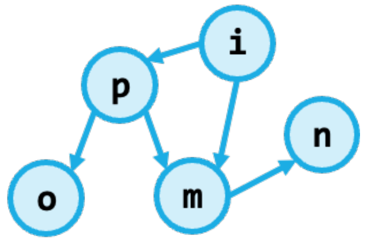




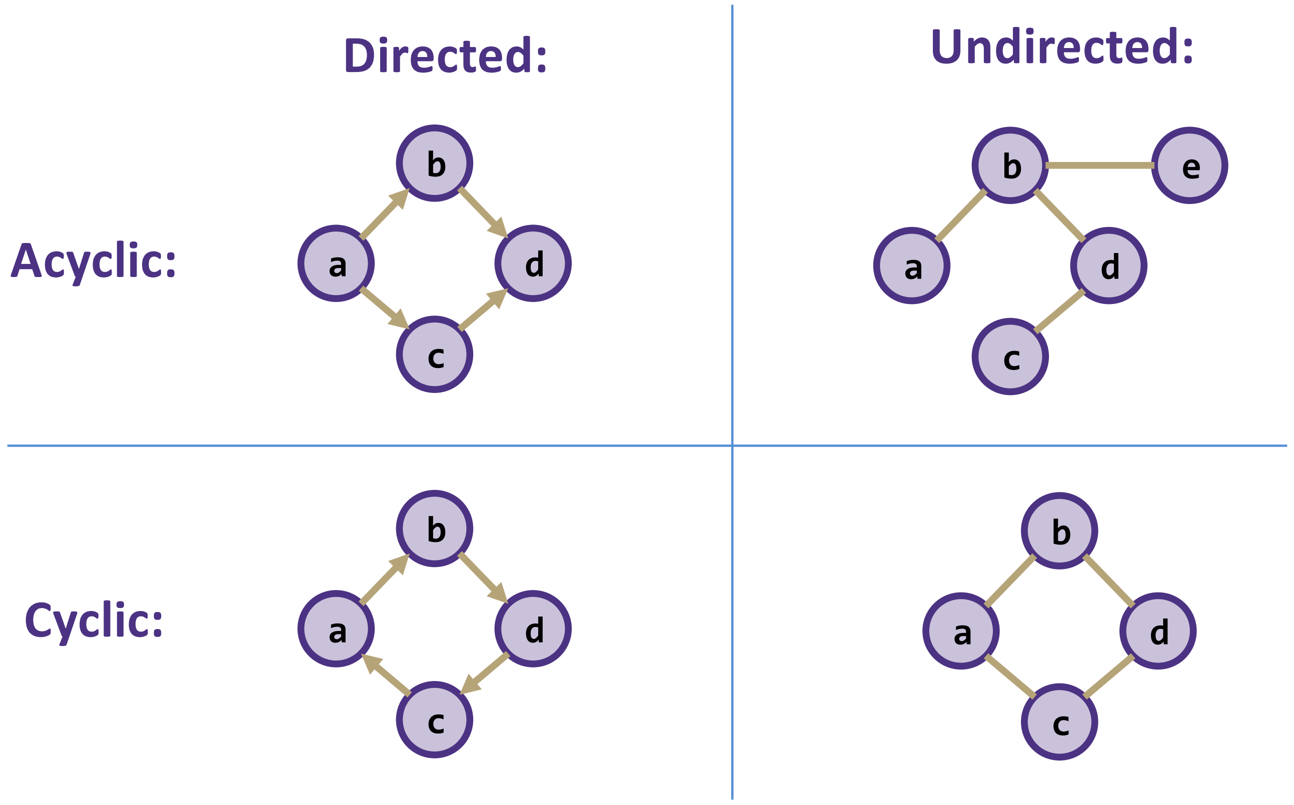
Two vertices are connected if there is a path between them
- If all the vertices are connected, we say the graph is connected

A path is a sequence of vertices connected by edges
- A simple path is a path without repeated vertices
- A cycle is a path whose first and last vertices are the same
    - A graph with a cycle is cyclic



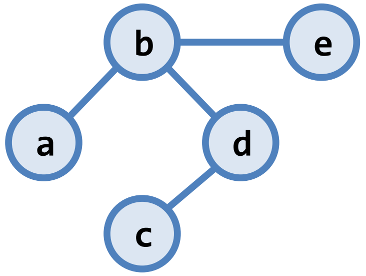


Directed vs Undirected; Acyclic vs Cyclic

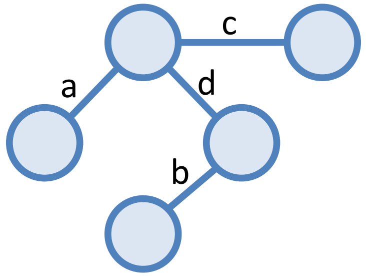


Labeled and Weighted Graphs

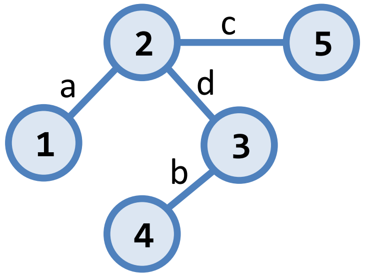
Vertex Labels



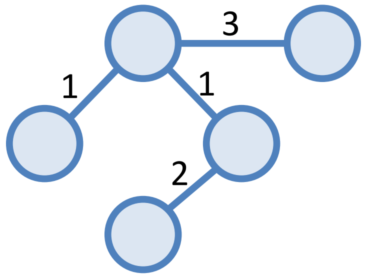
Edge Labels



Vertex & Edge Labels

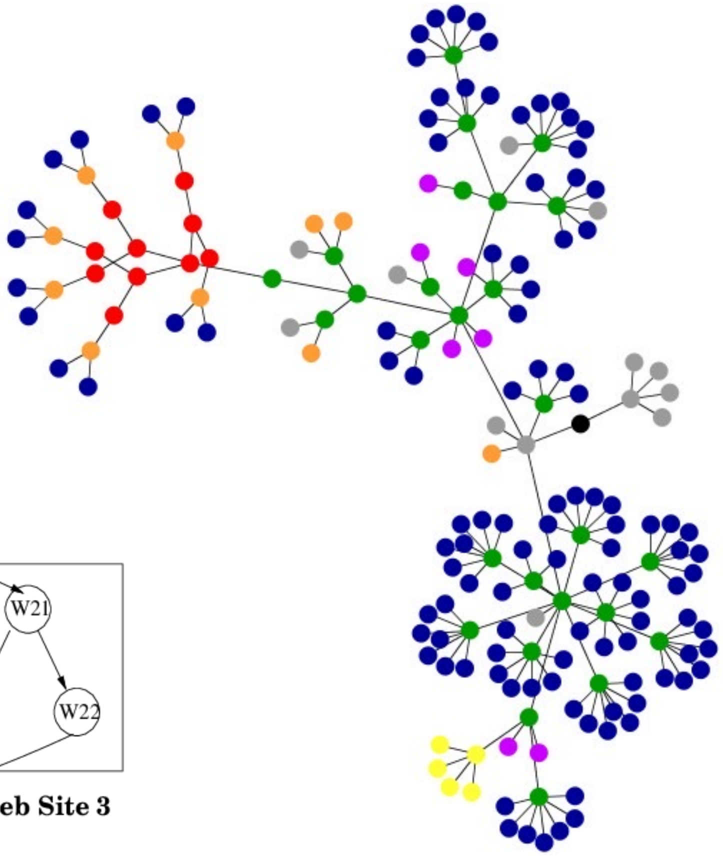


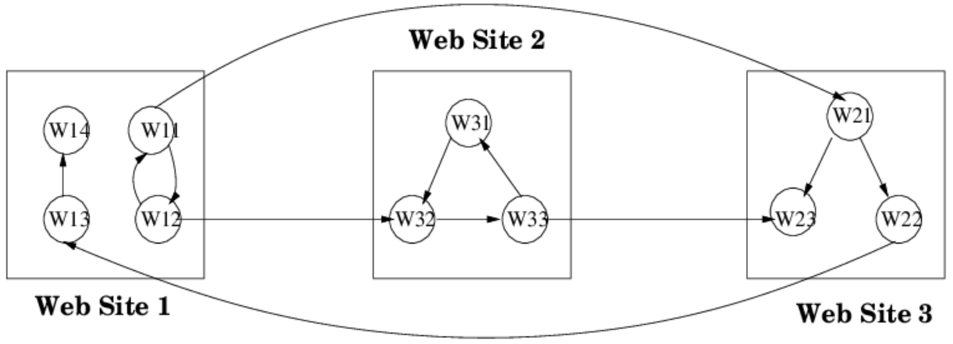
Numeric Edge Labels **(Edge Weights)**



Example: The Web

- **Vertices**: webpages.
- **Edges** from a to b if a has a hyperlink to b.
- Directed, since hyperlinks go in one direction

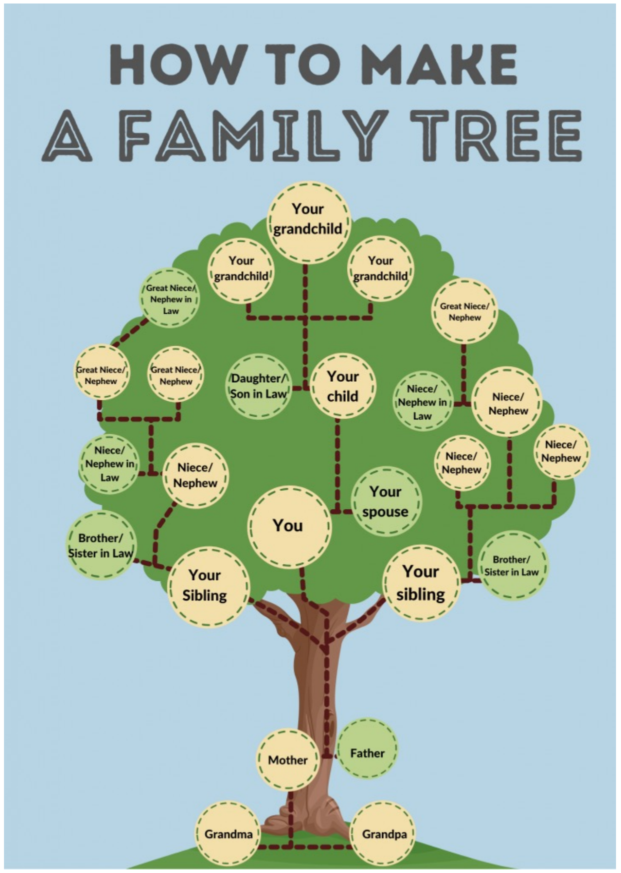




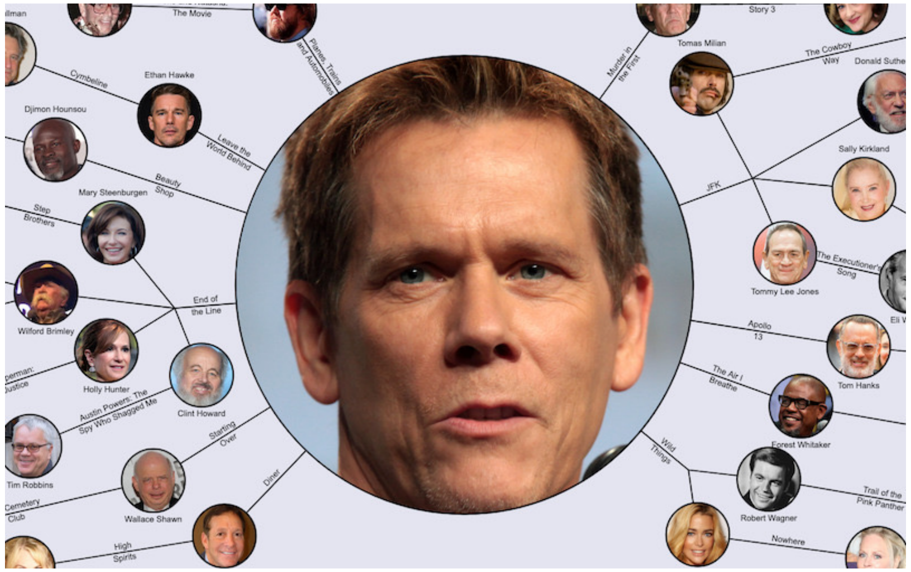
Example : Family Tree

- **Vertices**: people.
- **Edges**: relationships
- Undirected, bidirectional relationships (?)

By the way,
a **tree** *is* a **graph** !



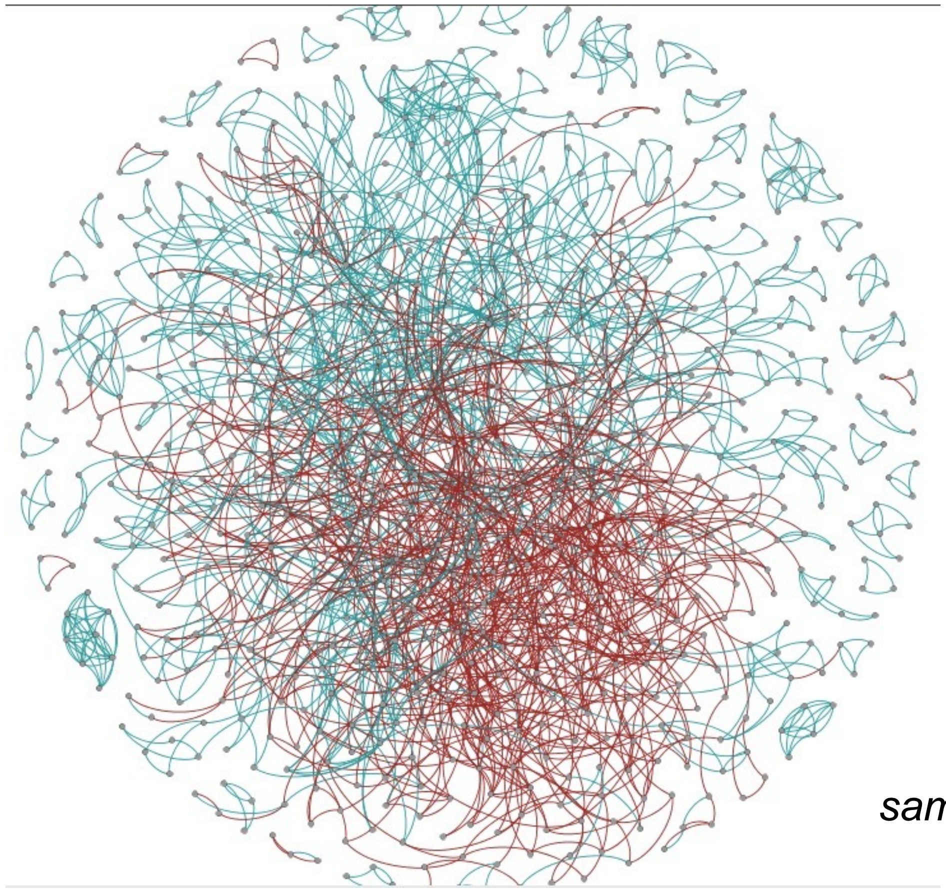
Example: 6 Degrees of Kevin Bacon



Six degrees of Kevin Bacon has been a popular way to measure affinity between actors. It links actors who have appeared in the same movie.

- **Vertices**: actors.
- **Edges**: movies (labeled)
- Undirected, a both actor would need to be in the movie for the edge to be added

Example: positive and negative weights



one person trusting/distrusting another

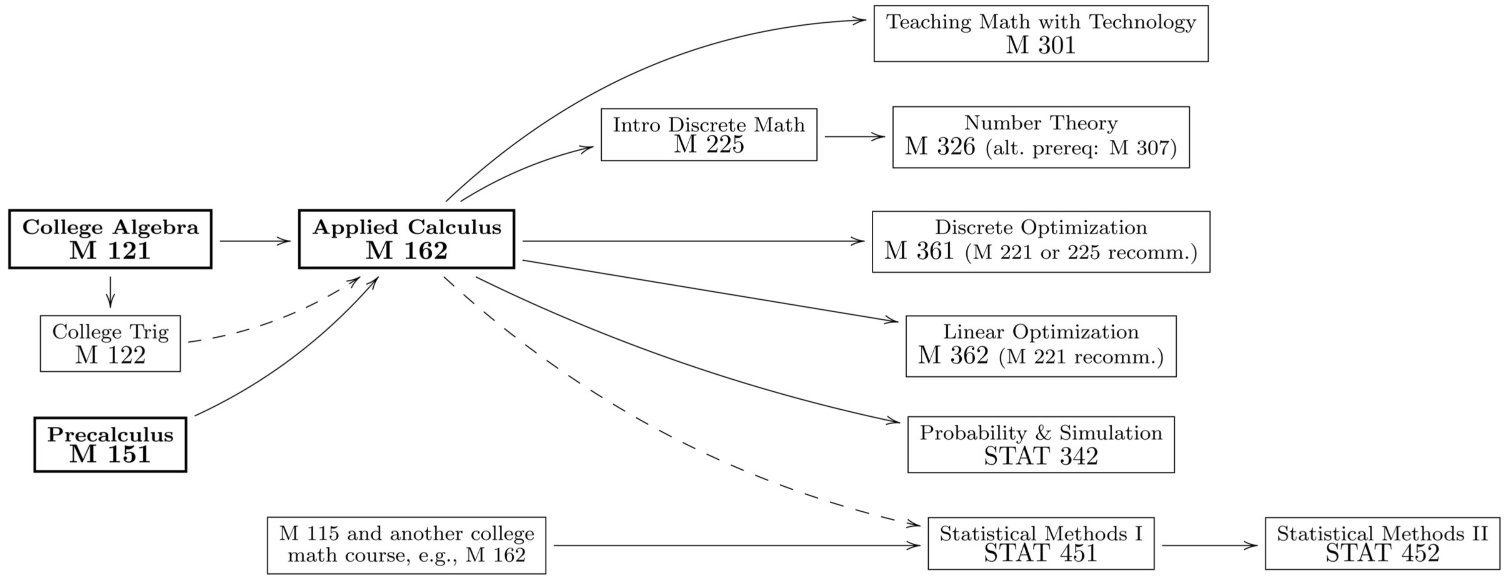
- **Vertices**: users (reviews' authors)
- **Edges**: trust
- Directed

[see https://networkrepository.com/epinions.php](https://networkrepository.com/epinions.php)

sample of positive & negative ratings from Epinions network

https://en.wikipedia.org/wiki/Epinions

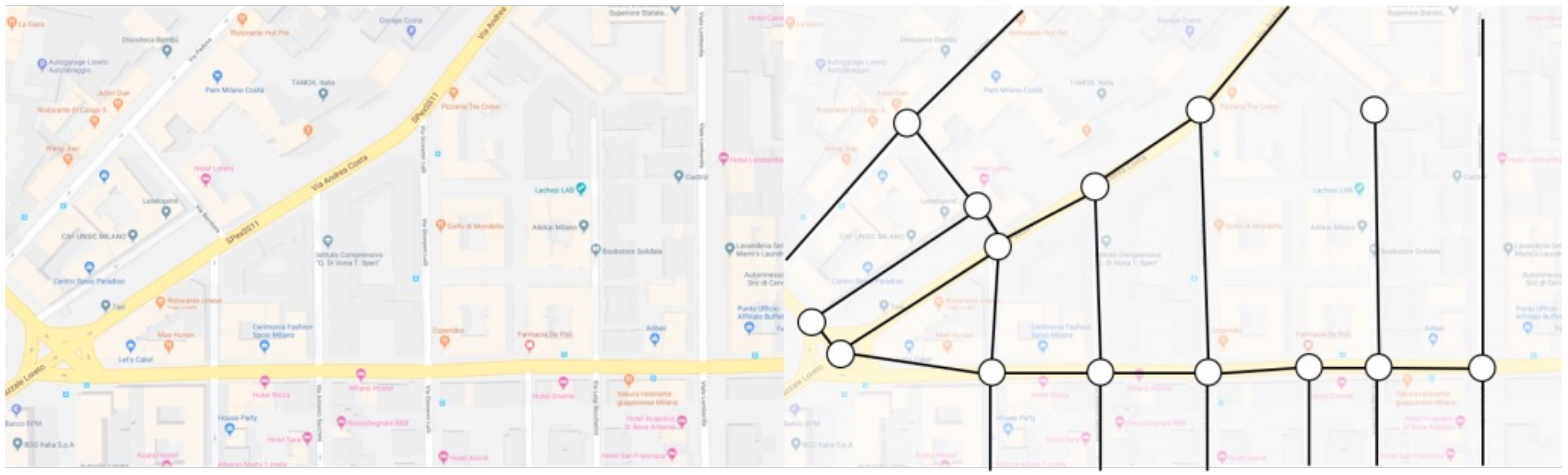
Example: Course Prequisites



- **Vertices**: courses
- **Edge**: from a to b if a is a prerequisite for b.
- Directed, since one course comes before the other

Example: map

- Ways to walk between campus buildings
  - **Vertices**: buildings.
  - **Edges**: A street name or walkway that connects 2 sites
  - Undirected, since each route can be walked both ways (?)



Representation of Graphs

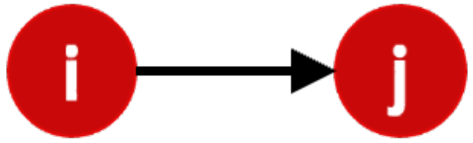
- Adjacency Matrix
- Edge List
- Adjacency List

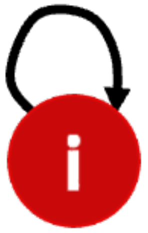
Representation of Graphs

Adjacency Matrix

Representing edges (who is adjacent to whom) as a matrix

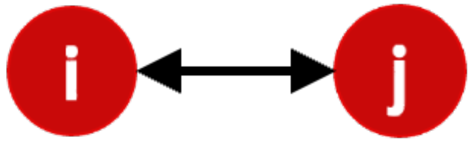
A_{ij} = 1 if node *i* has an edge to node *j*
= 0 if node *i* does not have an edge to j





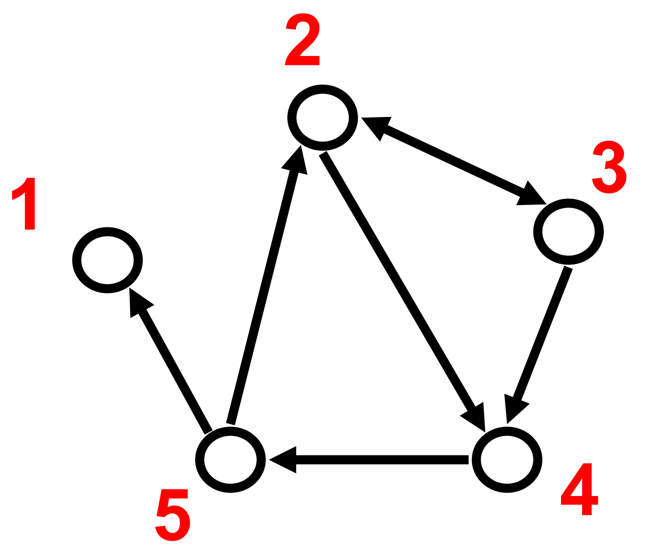
A_{ii} = 0 unless the network has self-loops

$A_{ij} = A_{ji}$ if the network is undirected,
or if *i* and *j* share a reciprocated edge



Representation of Graphs

Adjacency Matrix



$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Representation of Graphs

Adjacency Matrix

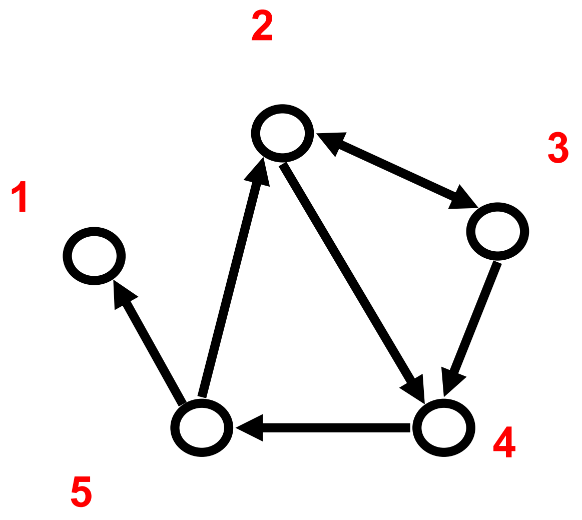
$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

- Pros:
  - Simple to implement
  - Easy and **fast** to tell if a pair (i,j) is an edge: simply check if A[i][j] is 1 or 0
- Cons:
  - No matter how few edges the graph has, the matrix takes $O(n^2)$ in **memory**

Representation of Graphs

Edge List

2, 3
2, 4
3, 2
3, 4
4, 5
5, 2
5, 1



Compact

Representation of Graphs

Adjacency List

For each node, list the neighbors:

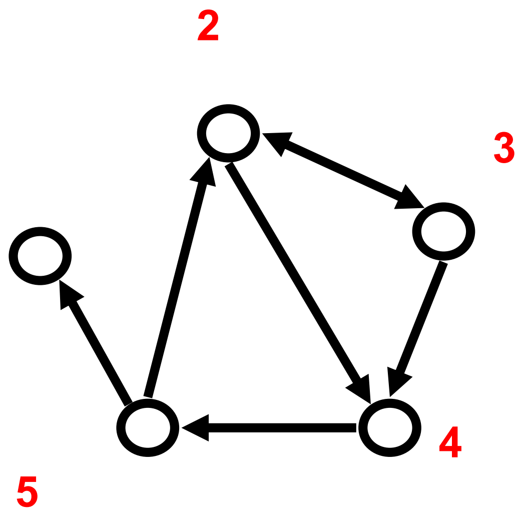
1:

2: 3 4

3: 2 4

4: 5

5: 1 2



Interesting for large *sparse* graphs

Representation of Graphs

Adjacency List : another example

L[0]: empty

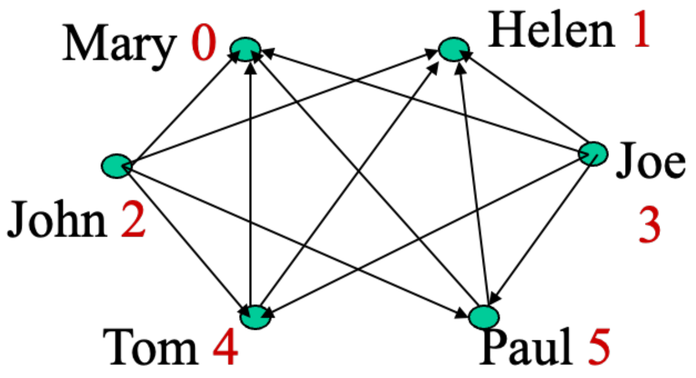
L[1]: empty

L[2]: 0, 1, 4, 5

L[3]: 0, 1, 4, 5

L[4]: 0, 1

L[5]: 0, 1



Representation of Graphs

Adjacency List :

- Pros:
  - Saves on space (memory): the representation takes as many memory words as there are nodes and edges

- Cons:
  - It takes O(n) time to determine if a pair of nodes (i,j) is an edge: one would have to search the linked list L[i], which takes time proportional to the length of L[i].

Representation of Graphs

Sparse Adjacency Matrix

A graph is frequently *sparse*

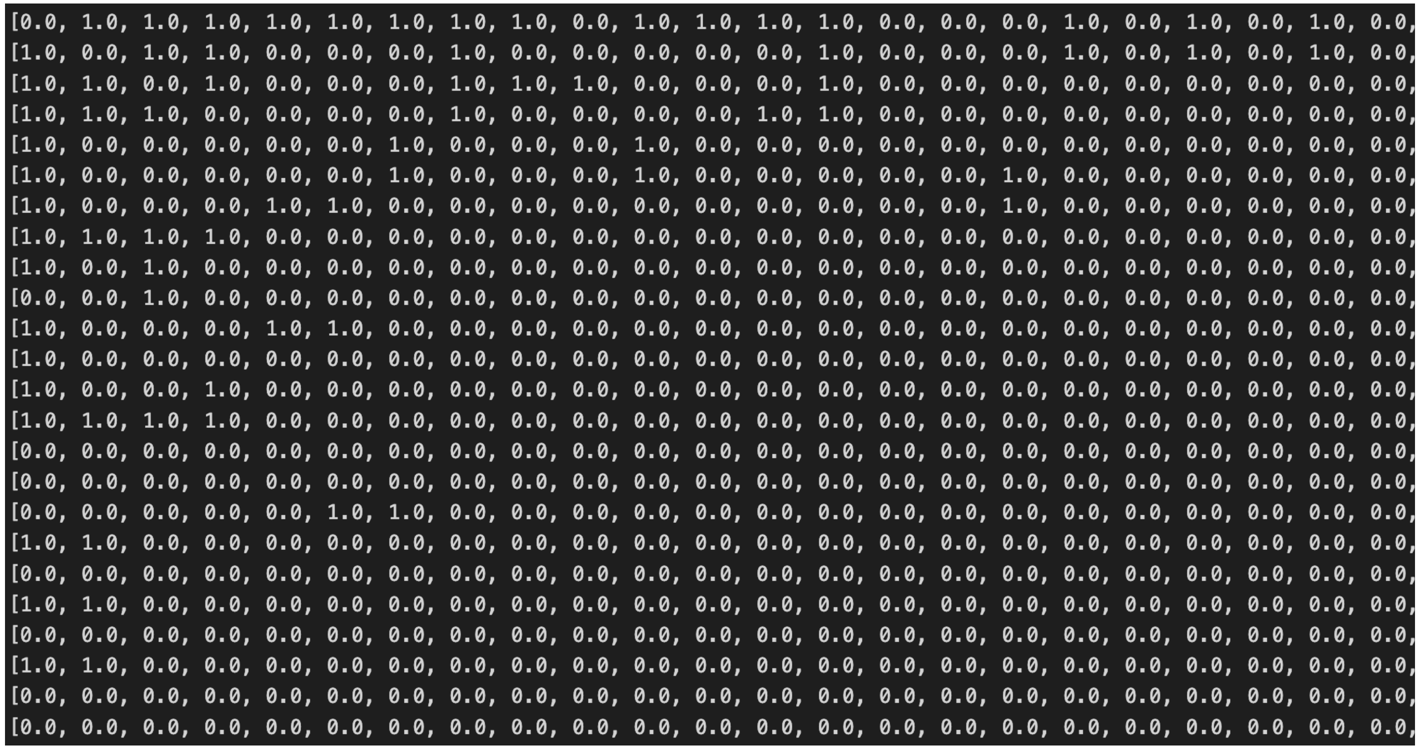
If we have N nodes, the maximum number of edges for an undirected graph is L = N(N-1)/2

But in a lot of graphs, we have L growing proportionally with N : L = d . N
where d is the average degree of the nodes.

In this case, the adjacency matrix if full of ***zeroes***

Representation of Graphs

Sparse Adjacency Matrix



```
[0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0,
[1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0,
[1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

Representation of Graphs

Sparse Adjacency Matrix

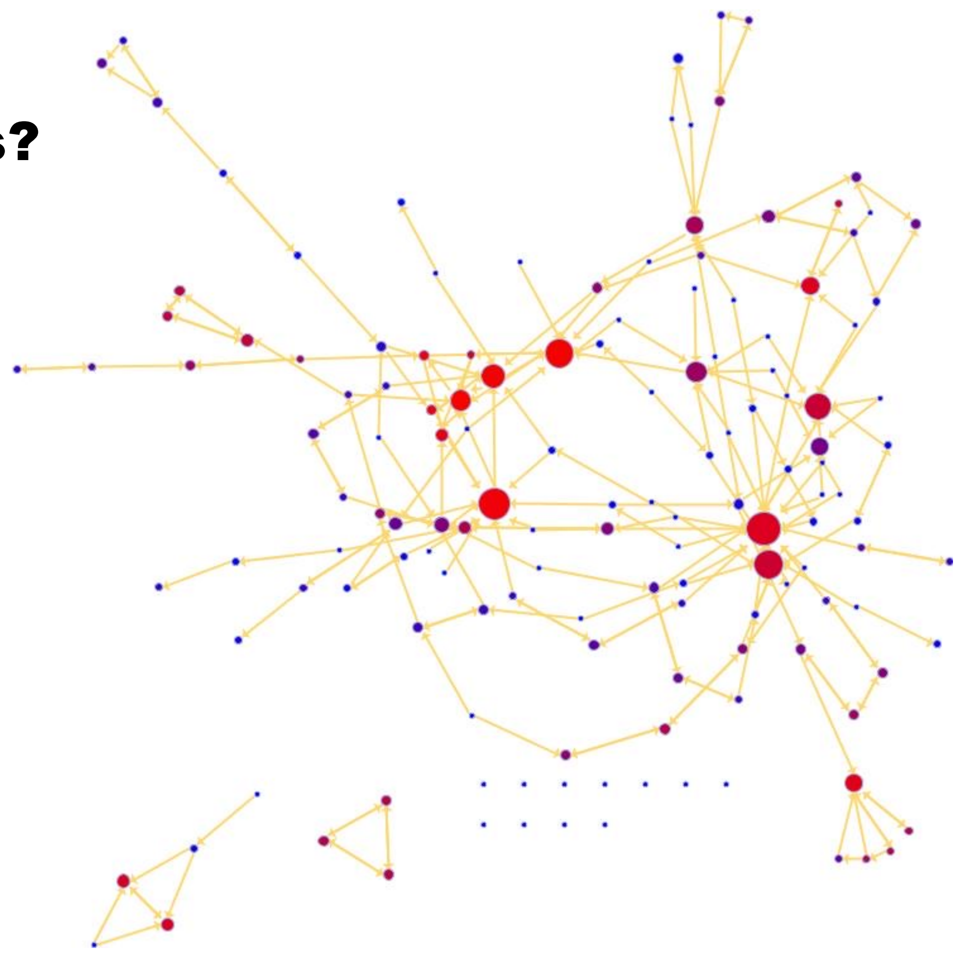
A sparse matrix is internally stored as linked list, but presents itself as a matrix for common operations

Sparse matrices are supported by most linear algebra packages (Python, Matlab, Scilab, …)

See eg https://docs.scipy.org/doc/scipy/reference/sparse.html

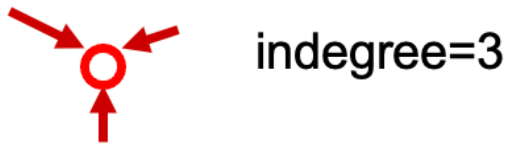
Node Degree

which node has the most edges?

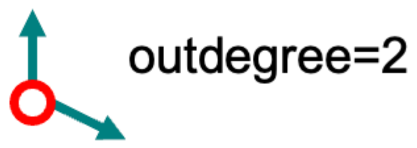


Node degree

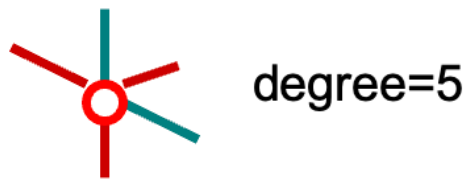
- **indegree**
  how many directed edges (arcs) are incident on a node



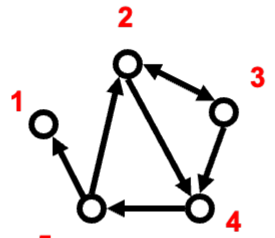
- **outdegree**
  how many directed edges (arcs) originate at a node



- **degree** (in or out)
  number of edges incident on a node



Computing the degrees



- Outdegree = $\sum_{j=1}^{n} A_{ij}$

A = $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$

example: outdegree for node 3 is 2, which we obtain by summing the number of non-zero entries in the 3rd *row*

$$\sum_{j=1}^{n} A_{3j}$$

- Indegree = $\sum_{i=1}^{n} A_{ij}$

A = $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$

example: the indegree for node 3 is 1, which we obtain by summing the number of non-zero entries in the 3rd *column*

$$\sum_{i=1}^{n} A_{i3}$$

Handshaking lemma

handshaking lemma states that, in every finite undirected graph, the number of vertices that touch an **odd number of edges** is **even**.

⇔ *every graph has an even number of **odd** nodes*

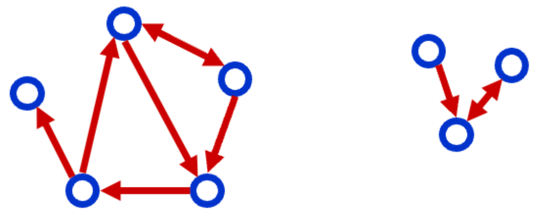
The degree sum formula states that

$$\sum_{v \in V} \deg v = 2|E|,$$

Degree sequence and degree distribution

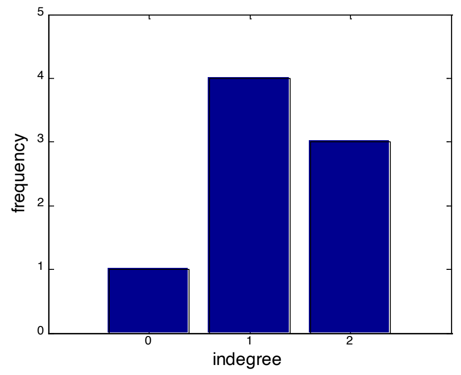
Degree sequence: An ordered list of the (in,out) degree of each node

- In-degree sequence:
  - [2, 2, 2, 1, 1, 1, 1, 0]
- Out-degree sequence:
  - [2, 2, 2, 2, 1, 1, 1, 0]
- (undirected) degree sequence:
  - [3, 3, 3, 2, 2, 1, 1, 1]

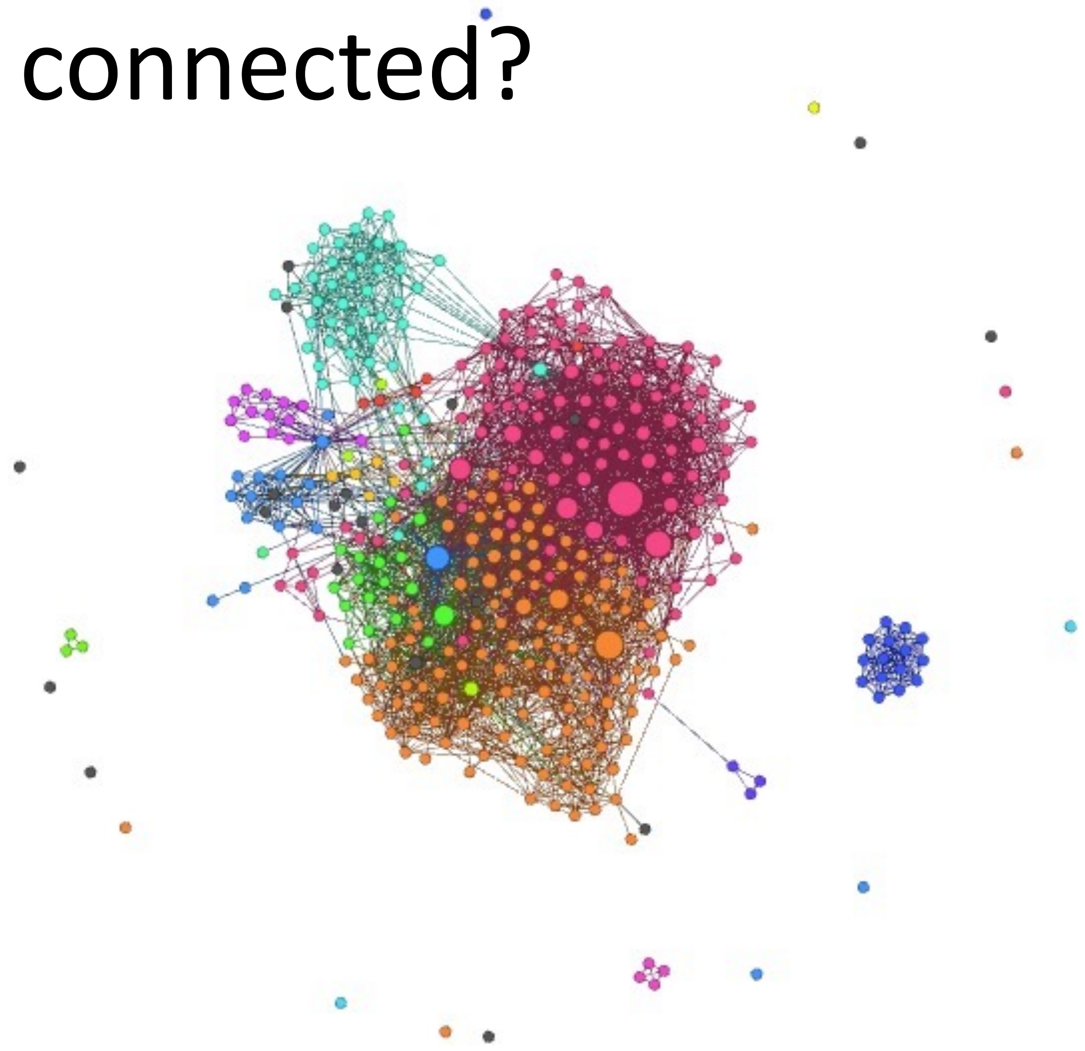


Degree distribution: A frequency count of the occurrence of each degree

- In-degree distribution:
  - [(2,3) (1,4) (0,1)]
- Out-degree distribution:
  - [(2,4) (1,3) (0,1)]
- (undirected) distribution:
  - [(3,3) (2,2) (1,3)]

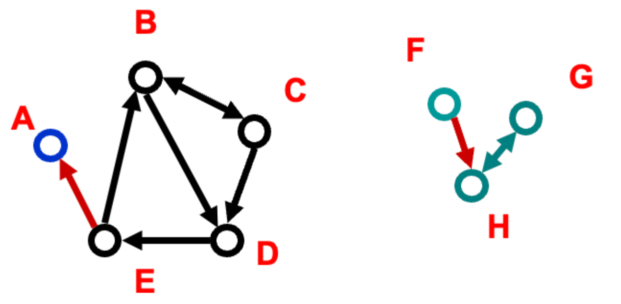


Is everything connected?



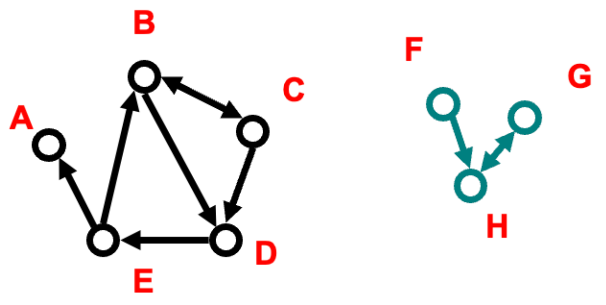
Connected Components

- Strongly connected components
  - Each node within the component can be reached from every other node in the component by following directed links



- Strongly connected components
  - B C D E
  - A
  - G H
  - F

- Weakly connected components: every node can be reached from every other node by following links in either direction

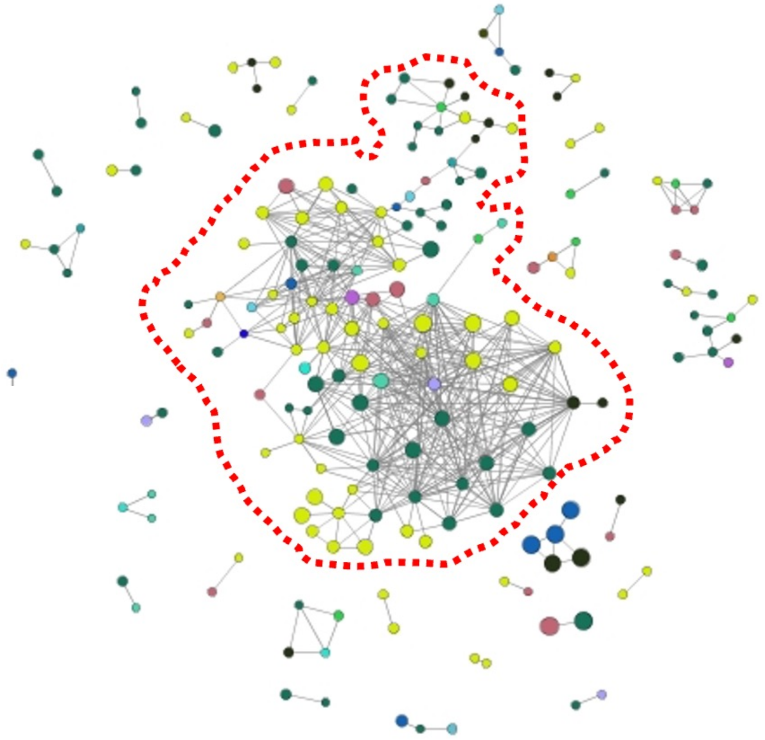


- Weakly connected components
  - A B C D E
  - G H F

- In undirected networks one talks simply about 'connected components'

Giant Component

- if the largest component encompasses a significant fraction of the graph, it is called the **giant component**



Graph problems

There are lots of interesting questions we can ask about a graph:

- What is the shortest route from S to T?
- What is the longest without cycles?
- Are there cycles?
- Is there a tour (cycle) you can take that only uses each node (station) exactly once?
- Is there a tour (cycle) that uses each edge exactly once?

Graph problems

Some well known graph problems and their common names:

- s-t Path. *Is there a path between vertices s and t?*
- Connectivity. *Is the graph connected?*
- Biconnectivity. *Is there a vertex whose removal disconnects the graph?*
- Shortest s-t Path. *What is the shortest path between vertices s and t?*
- Cycle Detection. *Does the graph contain any cycles?*
- Euler Tour. *Is there a cycle that uses every edge exactly once?*
- Hamilton Tour. *Is there a cycle that uses every vertex exactly once?*
- Planarity. *Can you draw the graph on paper with no crossing edges?*
- Isomorphism. *Are two graphs the same graph (in disguise)?*

Graph problems are among the most mathematically rich areas of CS theory!

Conclusion

Graph theory is a fundamental component of computer science with wide-ranging applications

- **Data Structures and Algorithms**: Graphs are essential in representing complex data structures like networks, which are central to various algorithms in computer science, such as those used in searching (like BFS and DFS), shortest path algorithms (like Dijkstra's and Bellman-Ford), and network flow algorithms.
- **Network Analysis**: Analyzing and optimizing computer networks, social networks, and web networks, including understanding the internet's topology, routing protocols, and analyzing social media interactions.
- **Problem Solving and Optimization**: Many complex computer science problems are modeled using graphs, including scheduling problems, resource allocation, and optimization problems (like the *Traveling Salesman Problem*), making graph theory a key tool for developing efficient solutions.
- **Database Theory**: Graphs are used in the modeling of databases, particularly in understanding relationships within network databases and for designing efficient data retrieval algorithms, including the use of graph databases in big data applications.
- **Artificial Intelligence and Machine Learning**: Graph theory plays a role in AI and ML, particularly in areas like semantic networks, neural networks, and in developing algorithms for clustering and pattern recognition, enhancing machine learning models' effectiveness in interpreting complex datasets.

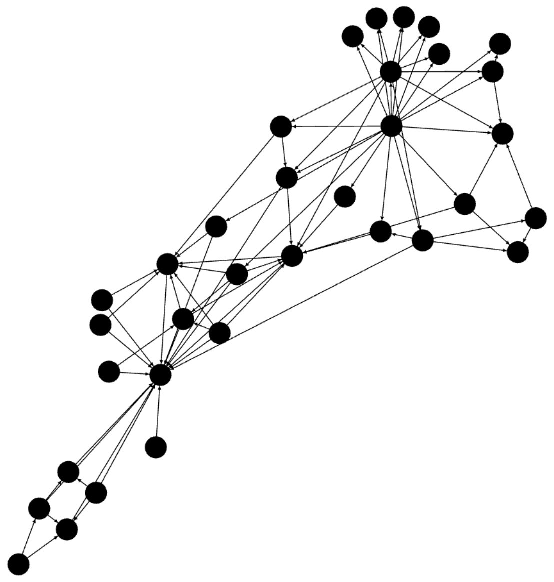
In this course…

We will study Graph Theory from the Compute Science point of view:

- describe basic algorithms : paths, connected components, flows…
- discuss the *complexity* of the algorithms
- implement some of them (in Python)
- show some useful software tools

Coming next:

1. Implement in Python a naïve Graph class using edge adjacency lists
2. Implement in Python a naïve Graph class using adjacency matrix
3. Load a real graph
4. Using network library
5. Presenting Gephi software



Documents are here:

https://www-l2ti.univ-paris13.fr/~viennet/ens/2024-USTH-Graphs