



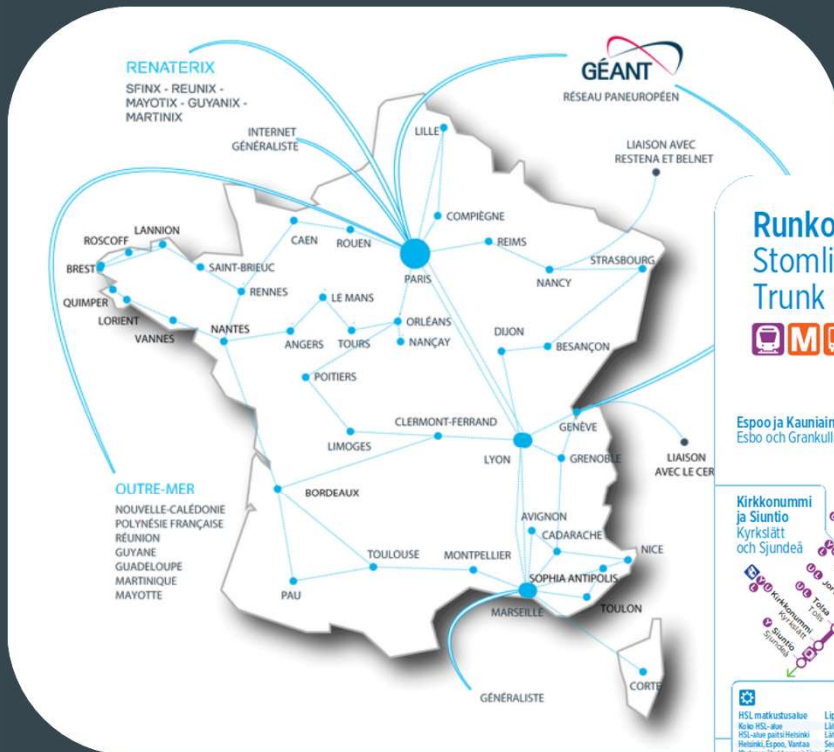
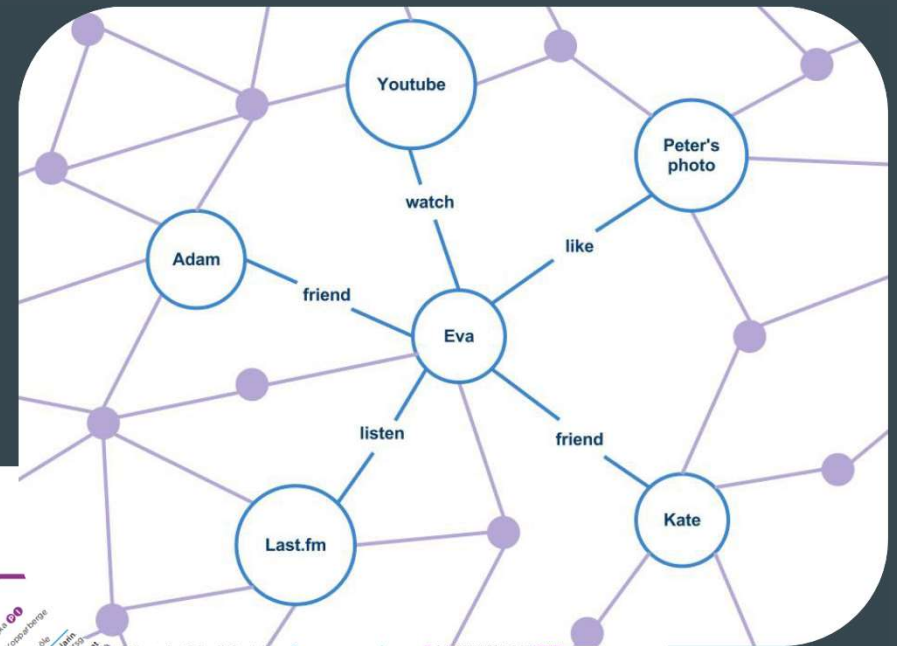
# Graph Theory 1

...

Introduction to graphs

# A world of graphs

*Social  
interactions*



## Renater network



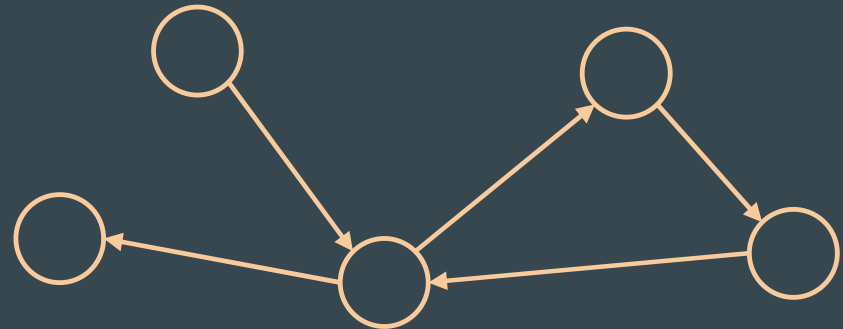
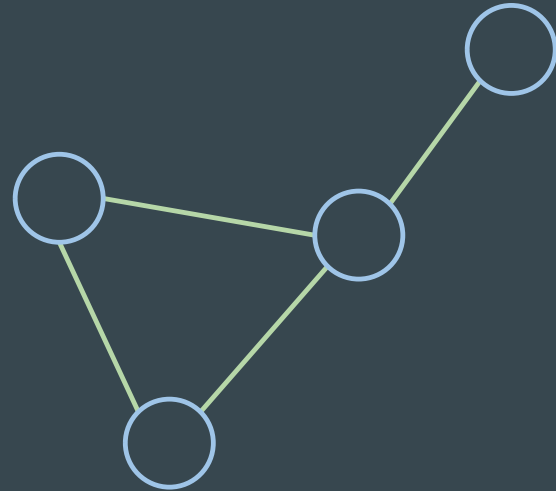
*Helsinki  
trains  
map*

# So what's a graph?

It's a tuple  $(V, E)$ , where

- $V$  is a set of vertices
- $E$  is a set of edges

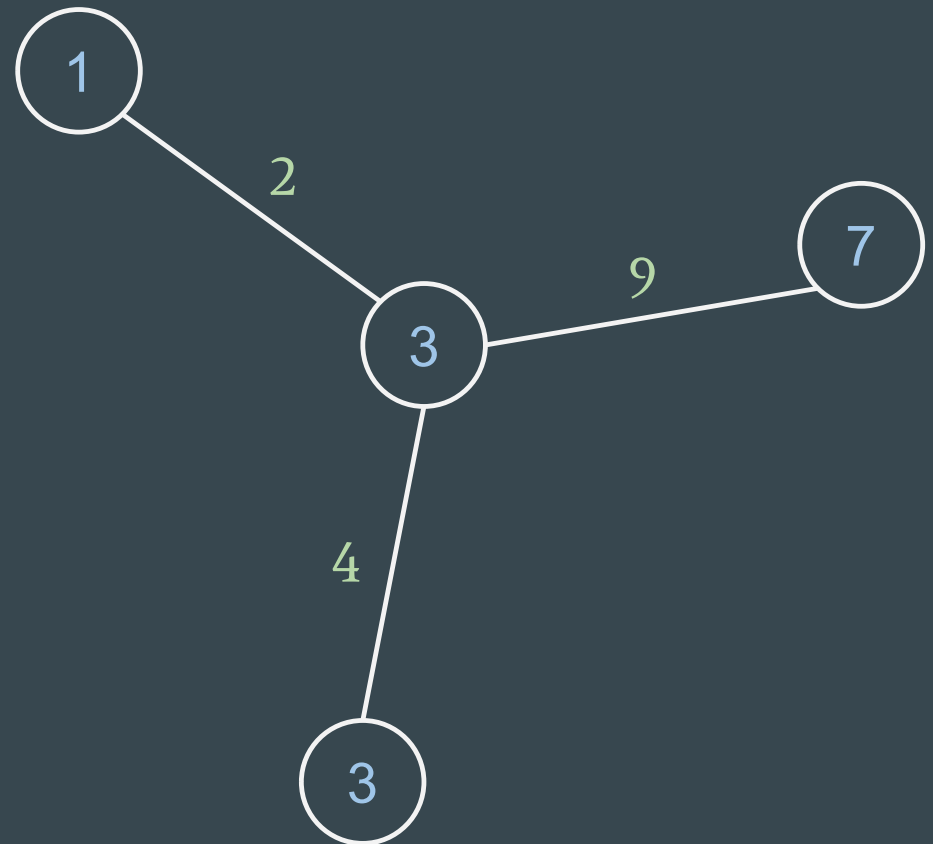
If the edges have an orientation, the graph is **directed**, otherwise it is undirected



# So what's a graph?

Some graphs contain more information such as:

- values on vertices
- values on edges



# How to represent a graph?

Solution #1 : edge list

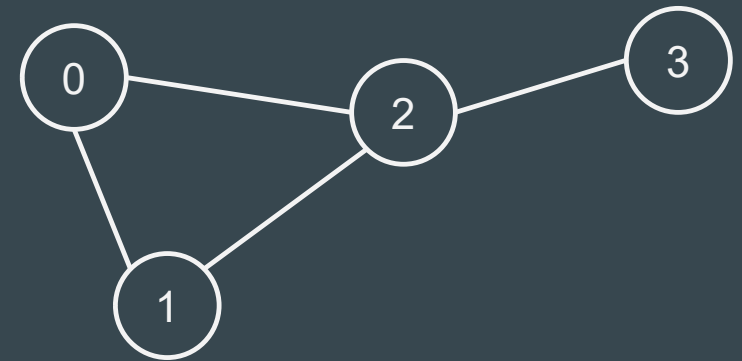
Let's label the vertices from 0 to  $n-1$

We need to find a data structure for the edges...

What about a list of couples?

Check if $i$ and $j$ are neighbors	$O( E )$
Find all neighbors of $i$	$O( E )$

→ awfully long for such simple operations



$[(0, 1), (0, 2), (1, 2), (2, 3)]$

# How to represent a graph?

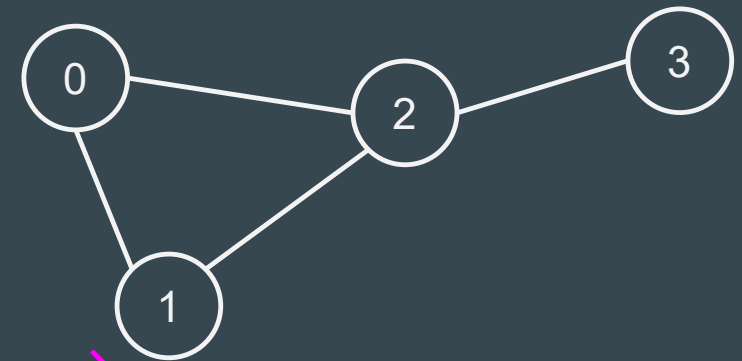
Solution #2 : adjacency matrix

Another idea is to store the relations as a matrix

→ boolean, or the value on the edge if the graph has some (and a dead value like None if there is no edge between two nodes)

Check if i and j are neighbors	$O(1)$
Find all neighbors of i	$O(n)$

→ but always takes  $O(n^2)$  in memory  
(not suitable for big graphs)



0	1	1	0
1	0	1	0
1	1	0	1
0	0	1	0

Symmetry for a directed graph

# How to represent a graph?

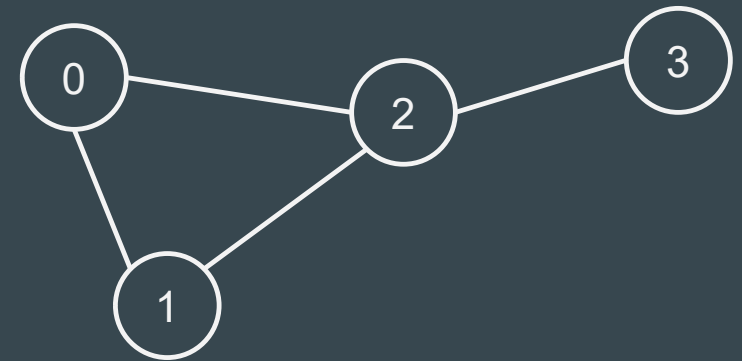
Solution #3 : Adjacency list

→ store all neighbors of each node

(there is a variant with sets, they take more memory but allow quicker adjacency check)

Check if i and j are neighbors	$O( E  / n)$ avg
Find all neighbors of i	$O( E  / n)$ avg

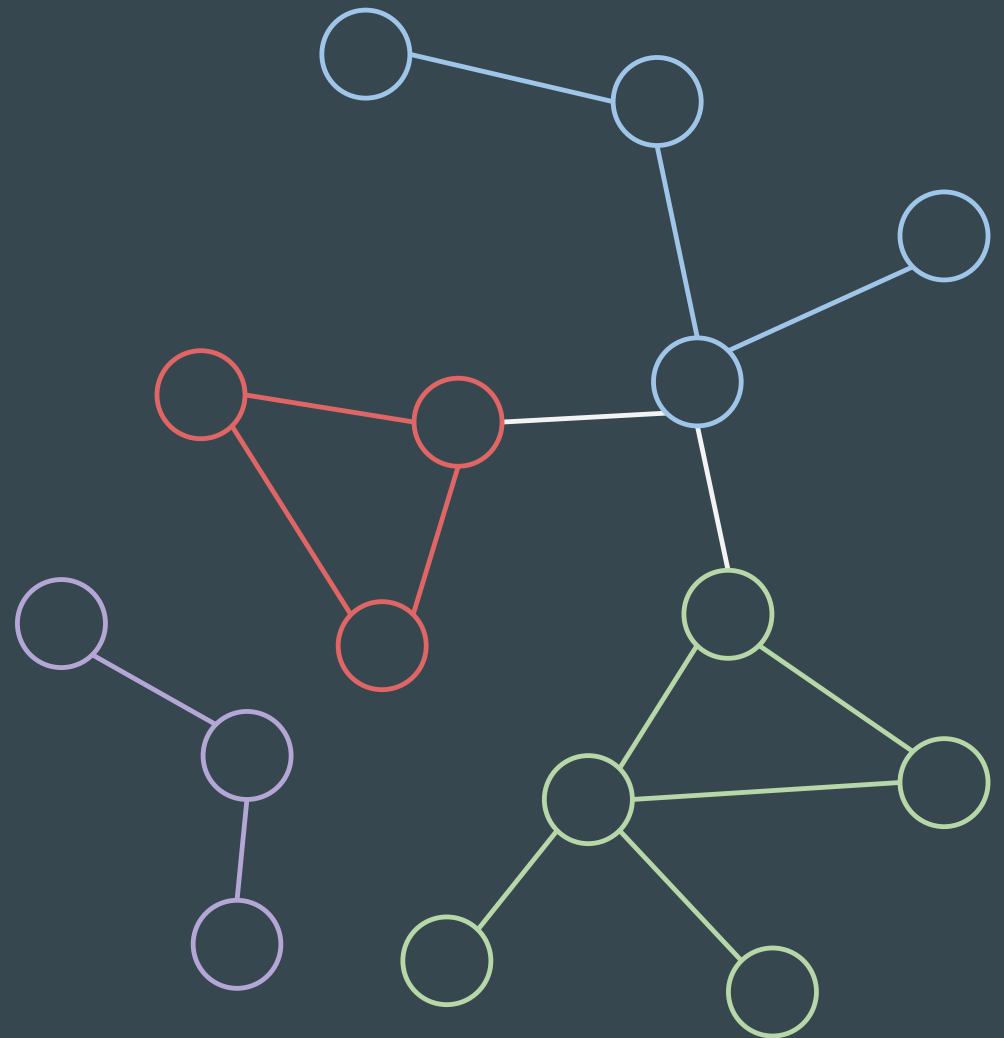
→ this representation is very often preferred as it is compact and efficient



```
{0: [1, 2],  
 1: [0, 2],  
 2: [0, 1, 3],  
 3: [2]}
```

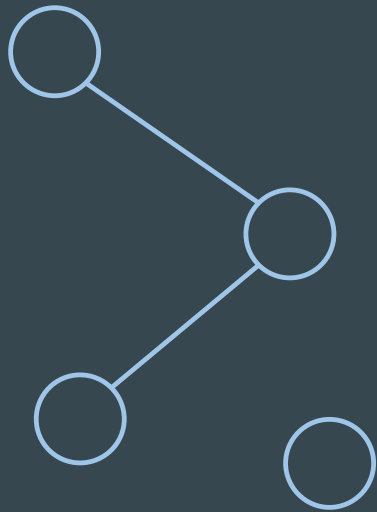
# Just a few terms

- a **path** is a sequence of 2-by-2 adjacent nodes and edges
- a **simple path** is a path that has no repeated vertices and edges
- a **cycle** is a path starting and ending at the same node
- a **connected component** is a subgraph in which there exists a path between any two nodes but not with any external node

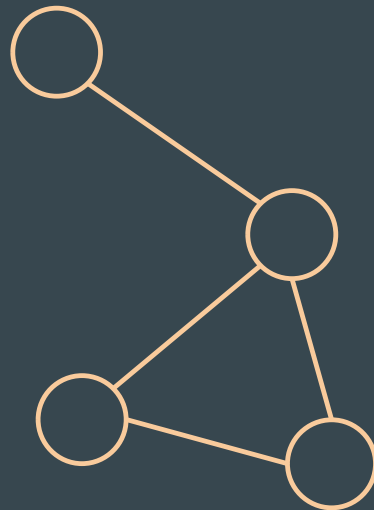




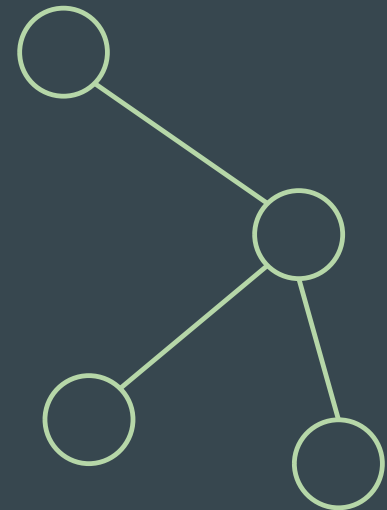
# Special graphs (undirected)



acyclic graph



connected graph



tree = connected + acyclic

# Note on directed graphs

- their matrix isn't symmetrical
- being acyclic doesn't mean that the undirected counterpart is acyclic too
- there are two notions of connection for directed graphs:
  - weakly connected = the corresponding undirected graph is connected
  - strongly connected = there is a path in each direction for each pair of vertices

# Some properties of trees...

- A tree has  $N-1$  edges for  $N$  vertices
- A tree is a connected & acyclic graph
- Remove any edge from the tree and it disconnects the graph
- In the general case vertice can have any degree
  - Some trees are more specific : binary trees, k-trees

Slides: Louis Sugy for INSAIgo, revised in  
2020 by Louis Gombert

*Helsinki trains map:*

HSL

*The Internet in 2015:*

The Opte Project

*Renater network:*

Renater