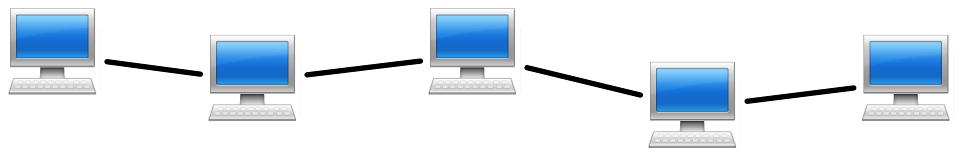
- **Weeks 1–2: informal introduction**
  - network = path 
- **Week 3: graph theory**
- **Weeks 4–7: models of computing**
  - what can be computed (efficiently)?
- **Weeks 8–11: lower bounds**
  - what cannot be computed (efficiently)?
- **Week 12: recap**

Recap of weeks 1–2

– Colouring paths

Model of computing:
Send, receive, update

- **All nodes in parallel:**
  - send messages to their neighbours
  - receive messages from neighbours
  - update their state
- **Stopping state = final output**
  - can send/receive, but not update any more

Example:
Colouring paths

- **2-colouring paths:**
  - possible in time $O(n)$
  - not possible in time $o(n)$

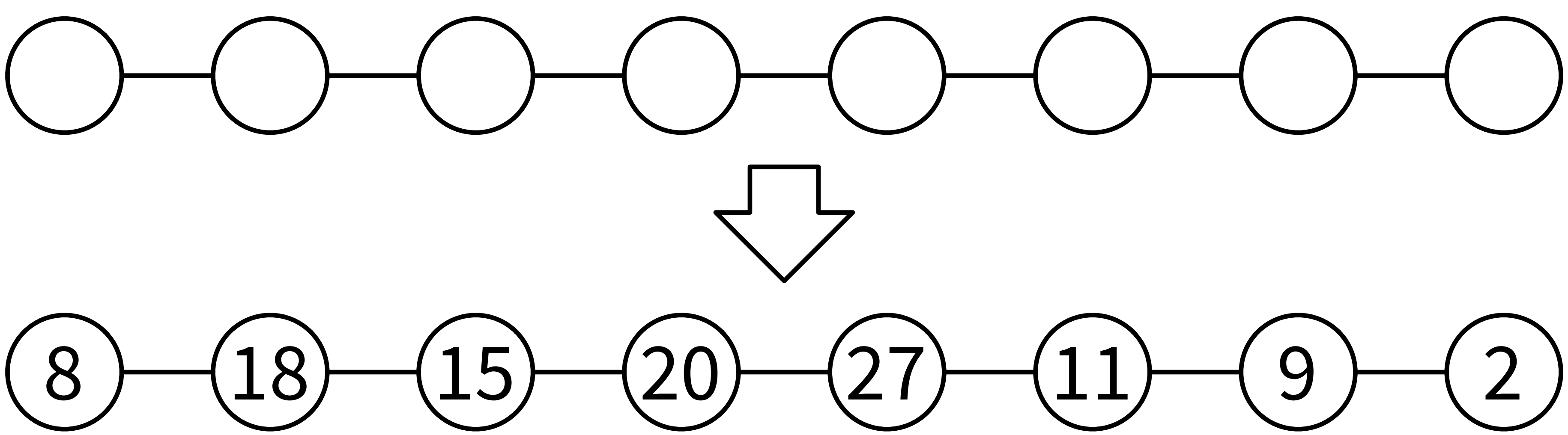
Assuming some unique identifiers

- **3-colouring paths:**
  - possible in time $O(\log^* n)$
  - not possible in time $o(\log^* n)$

Assuming "small" unique identifiers

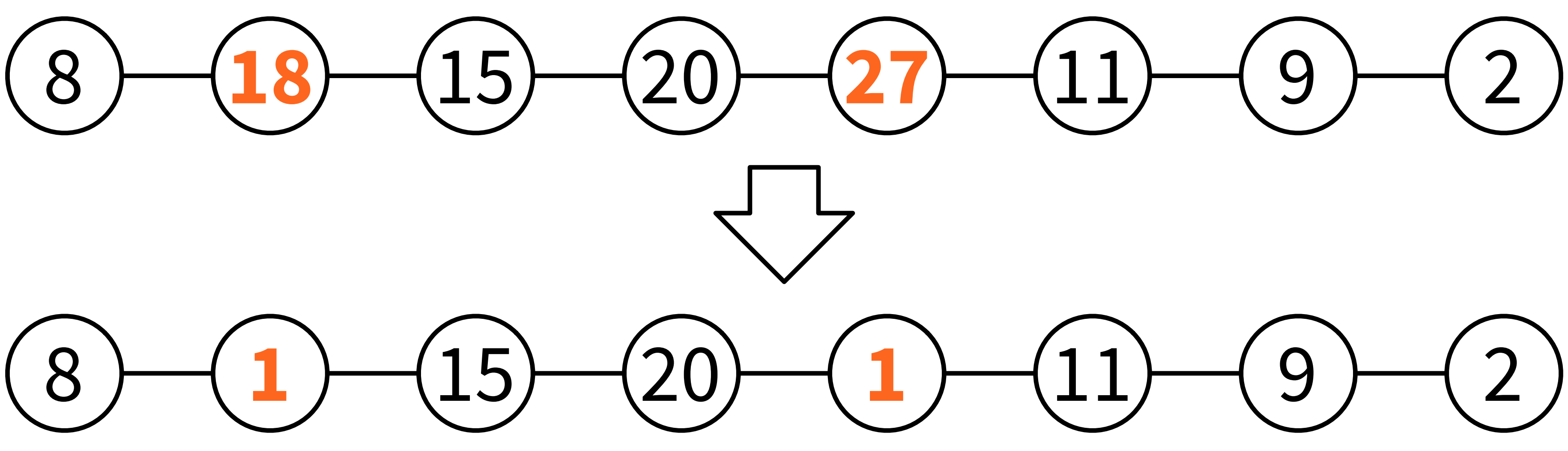
Algorithm design techniques

- **Symmetry breaking:** use e.g. unique identifiers or randomness to break symmetry



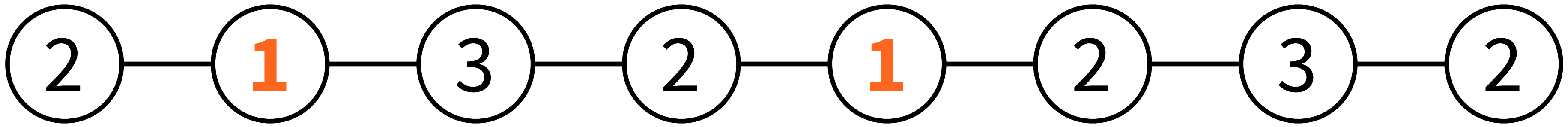
Algorithm design techniques

- **Independence:** non-adjacent nodes can act simultaneously in parallel without conflicts



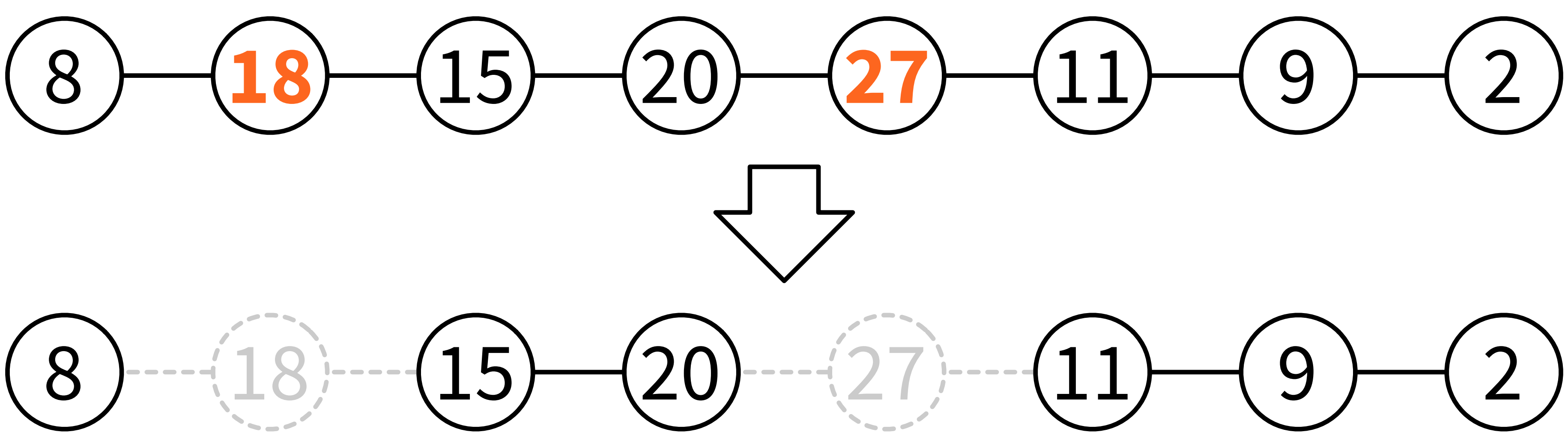
Algorithm design techniques

- **Independence:** non-adjacent nodes can act simultaneously in parallel without conflicts
- **Colouring → independence:** each colour class is an independent set



Algorithm design techniques

- **Divide and conquer:** split in smaller subproblems, solve recursively (in parallel)

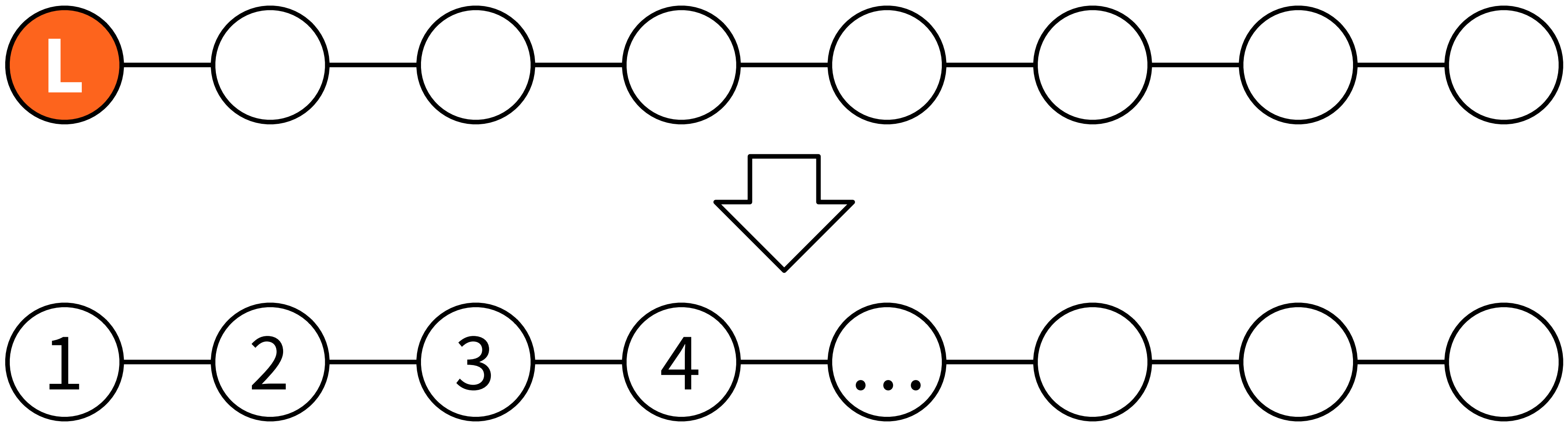


Algorithm design techniques

- **Composition and reductions:**
    - use “subroutines”
    - prove that a solution to problem X can be used to find a solution to problem Y
    - example: colourings $\leftrightarrow$ independent sets

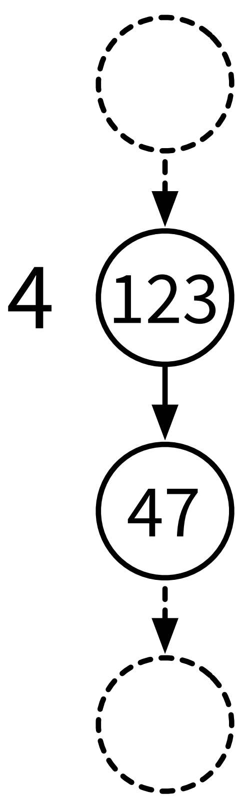
Algorithm design techniques

- **Simulate sequential algorithms:** elect **leader**, process nodes one by one



Algorithm design techniques

- **Fast colour reduction**



$\boldsymbol{c_0} = \mathbf{123} = \mathbf{01111011_2}$ (my colour)

$\boldsymbol{c_1} = \mathbf{47} = \mathbf{00101111_2}$ (successor's colour)

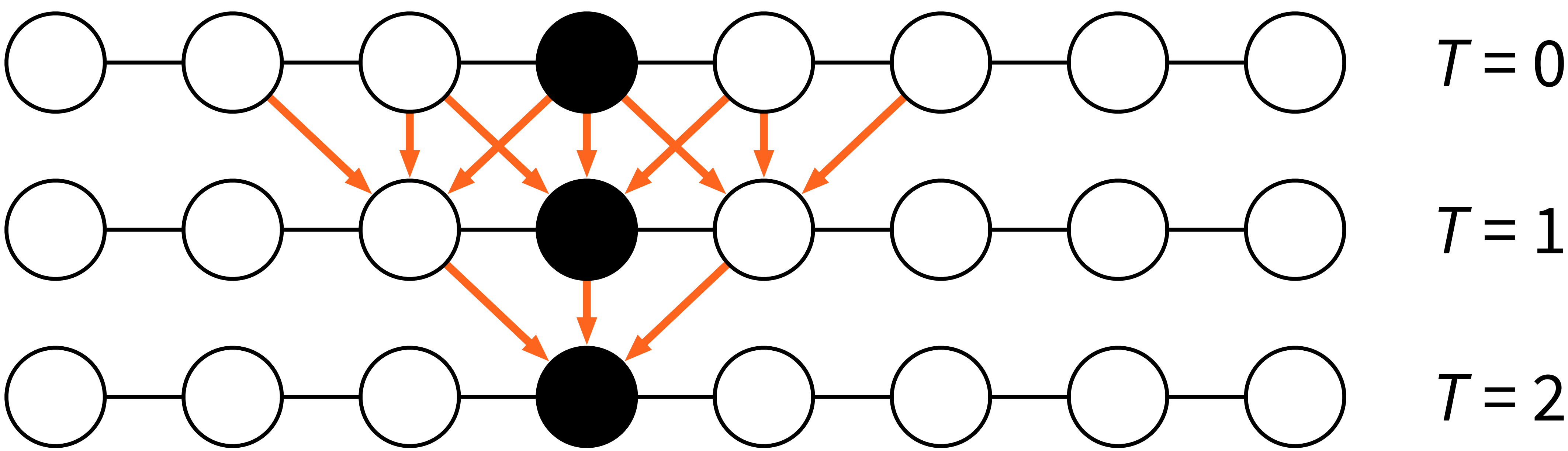
$\boldsymbol{i} = \mathbf{2}$ (bits numbered 0, 1, 2, … from right)

$\boldsymbol{b} = \mathbf{0}$ (in my colour bit number i was 0)

$\boldsymbol{c} = \mathbf{2 \cdot 2 + 0 = 4}$ (my new colour)

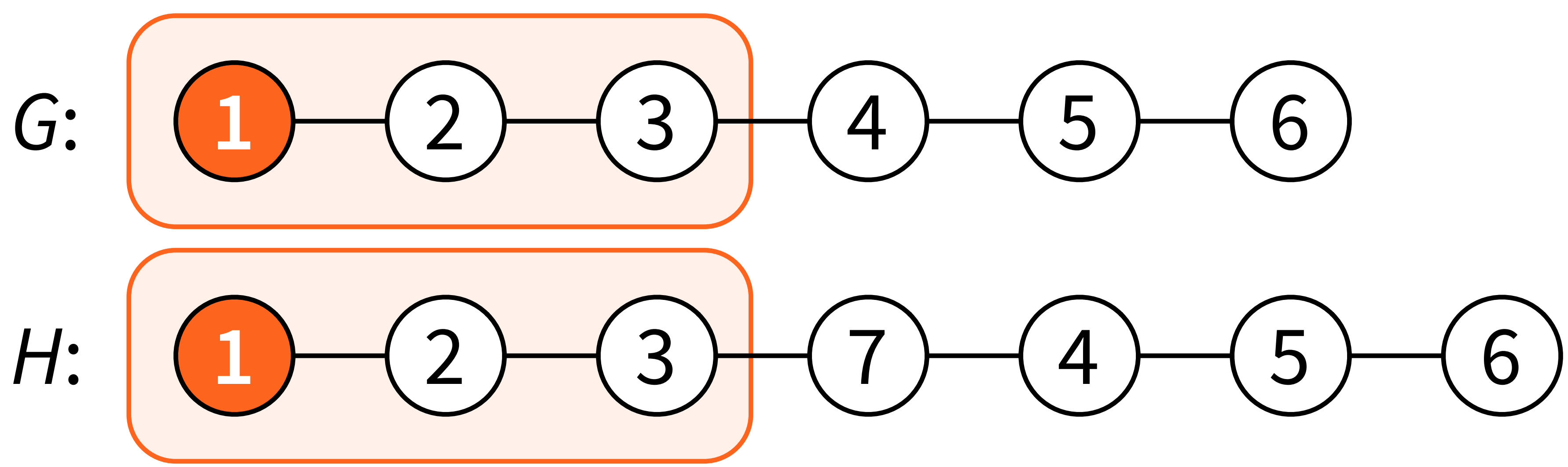
Proving lower bounds: **Locality**

- **State at time *T* only depends on initial information within distance *T***



Proving lower bounds: **Locality**

- **Same *T*-neighbourhood, same output after *T* rounds**



Example:
Colouring paths

- **2-colouring paths:**
  - possible in time $O(n)$
  - not possible in time $o(n)$
- **3-colouring paths:**
  - possible in time $O(\log^* n)$
  - not possible in time $o(\log^* n)$

Richard Cole and Uzi Vishkin (1986)

Nathan Linial (1992)

Week 3

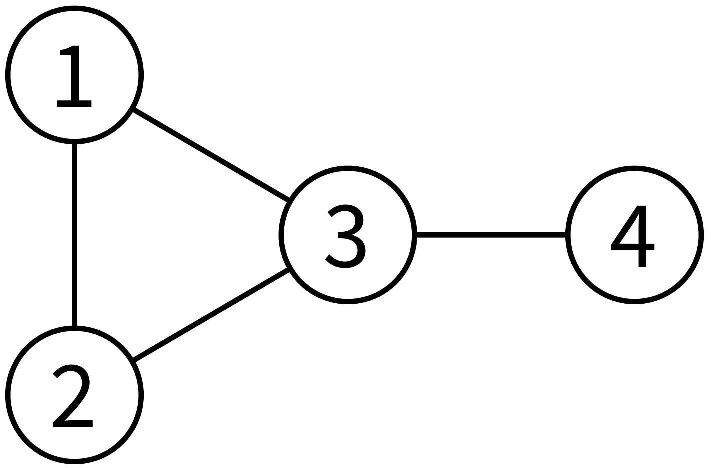
– Graph-theoretic foundations

Graph $G = (V, E)$

V **= set of nodes** (finite, non-empty)
E **= set of edges** (unordered pairs of nodes)

$G = (V, E)$
$V = \{1, 2, 3, 4\}$
$E = \{ \{1,2\}, \{1,3\}, \{2,3\}, \{3,4\} \}$

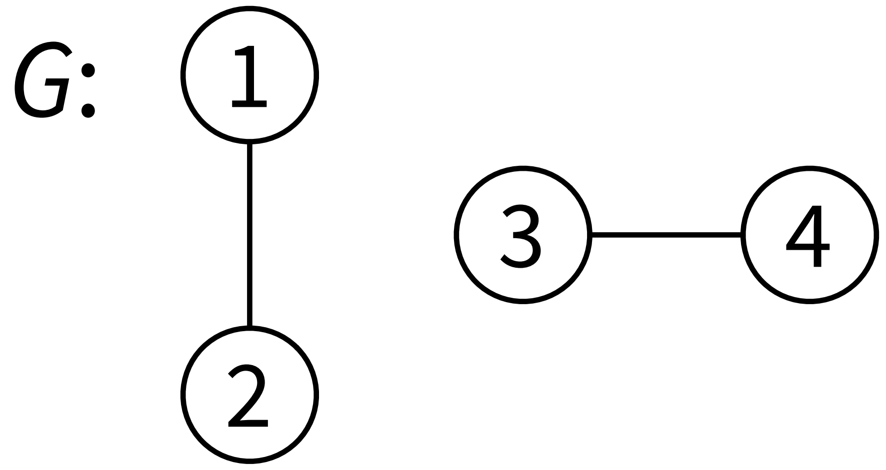
G:



Graph $G = (V, E)$

***V* = set of nodes** (finite, non-empty)
***E* = set of edges** (unordered pairs of nodes)

$G = (V, E)$
$V = \{1, 2, 3, 4\}$
$E = \{ \{1,2\}, \{3,4\} \}$

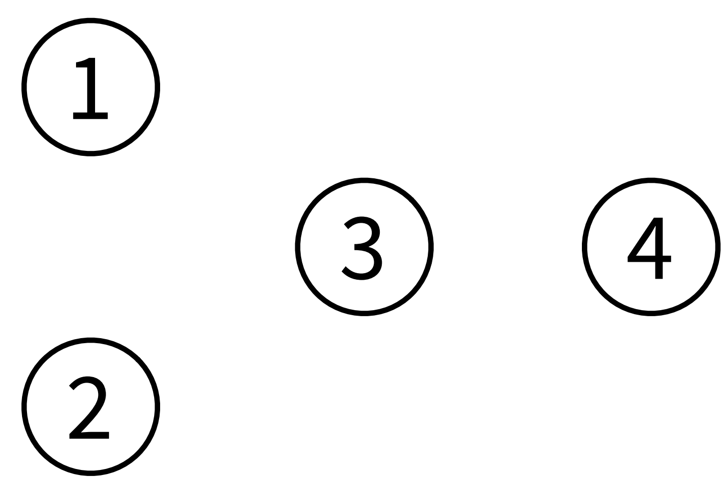


Graph $G = (V, E)$

***V* = set of nodes** (finite, non-empty)
***E* = set of edges** (unordered pairs of nodes)

$G = (V, E)$
$V = \{1, 2, 3, 4\}$
$E = \varnothing$

G:



Graph $G = (V, E)$

***V* = set of nodes** (a.k.a. "vertices")
***E* = set of edges**

Usually nodes are denoted with u, v
(if more nodes needed: s, t, u, v, u', v', v_1, v_2, etc.),
edges are denoted with e, e', e_1, e_2, etc.

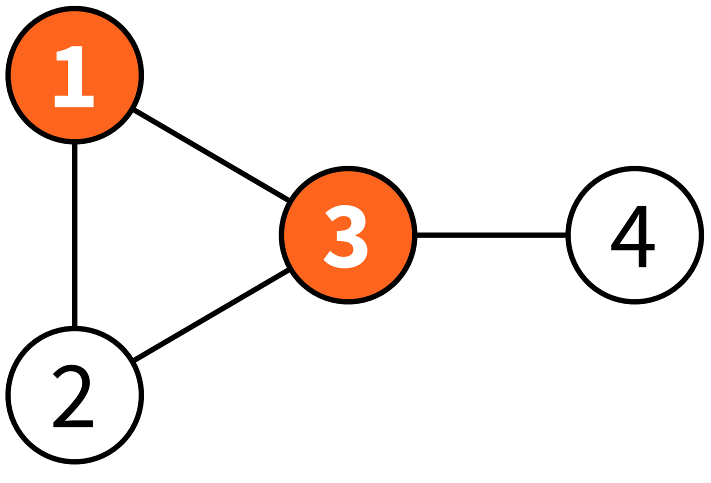
Convention: $n = |V|$, $m = |E|$

Graph $G = (V, E)$

u **and** ***v*** **are "adjacent nodes"**
= nodes ***u*** **and** ***v*** **are "neighbours"**
= there is an edge {*u*, *v*}

$G = (V, E)$
$V = \{\mathbf{1}, 2, \mathbf{3}, 4\}$
$E = \{ \{1,2\}, \mathbf{\{1,3\}}, \{2,3\}, \{3,4\} \}$

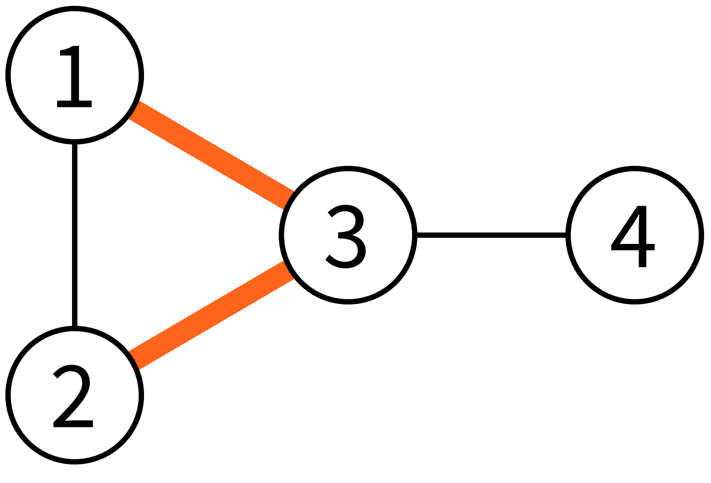
G:



Graph $G = (V, E)$

e_1 and e_2 are "adjacent edges"
= they share an endpoint
= their intersection is non-empty

$G = (V, E)$
$V = \{1, 2, 3, 4\}$
$E = \{\ \{1,2\}, \mathbf{\{1,3\}}, \mathbf{\{2,3\}}, \{3,4\}\ \}$

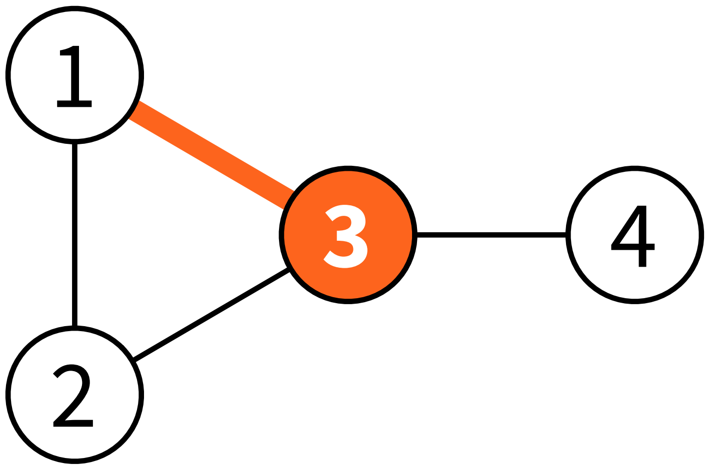
G: 

Graph $G = (V, E)$

Node v is "incident" to edge e
= v is an endpoint of e
= v is a member of e

$G = (V, E)$
$V = \{1, 2, \mathbf{3}, 4\}$
$E = \{ \{1,2\}, \mathbf{\{1,3\}}, \{2,3\}, \{3,4\} \}$

G:

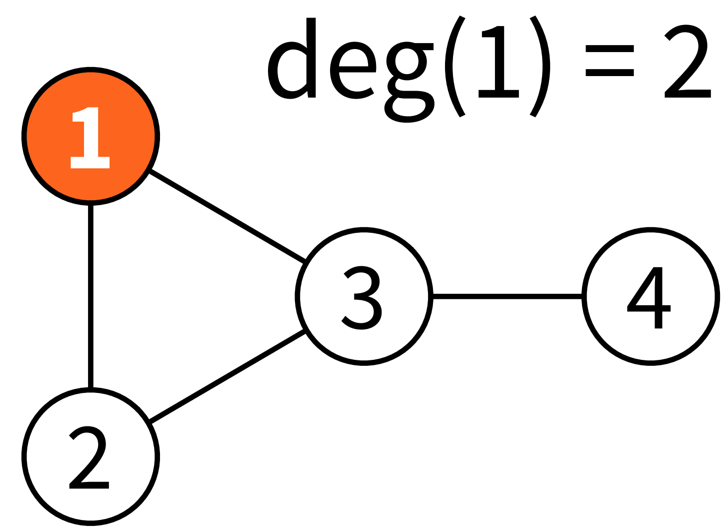


Graph $G = (V, E)$

Node of "degree" *k*
= node adjacent to *k* nodes
= node incident to *k* edges

$G = (V, E)$
$V = \{1, 2, 3, 4\}$
$E = \{ \{1,2\}, \{1,3\}, \{2,3\}, \{3,4\} \}$

G:

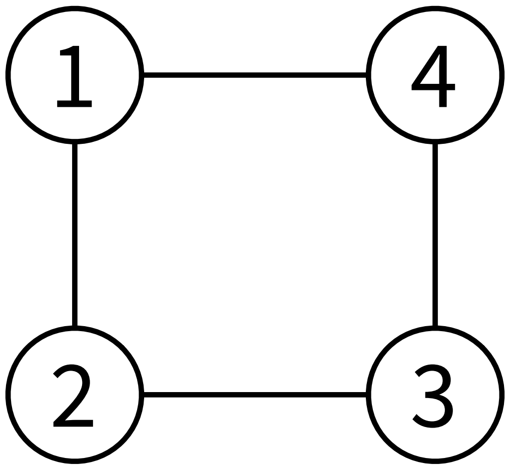


Graph $G = (V, E)$

"*k*-regular graph"
= all nodes have degree *k*
= all nodes have *k* neighbours

$G = (V, E)$
$V = \{1, 2, 3, 4\}$
$E = \{ \{1,2\}, \{2,3\}, \{3,4\}, \{1,4\} \}$

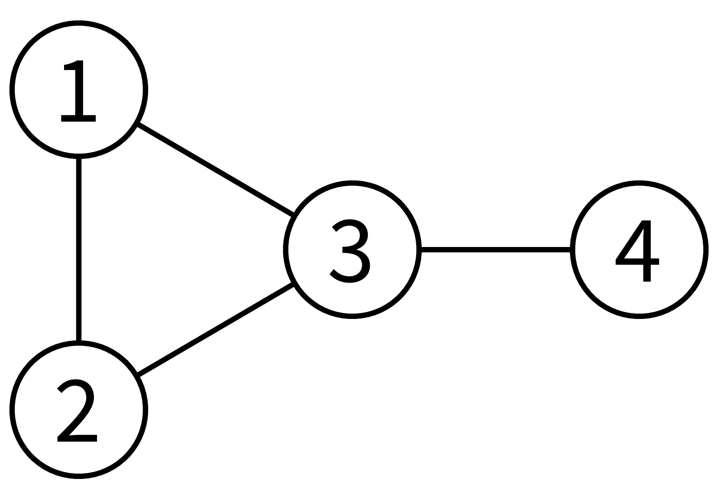
G:



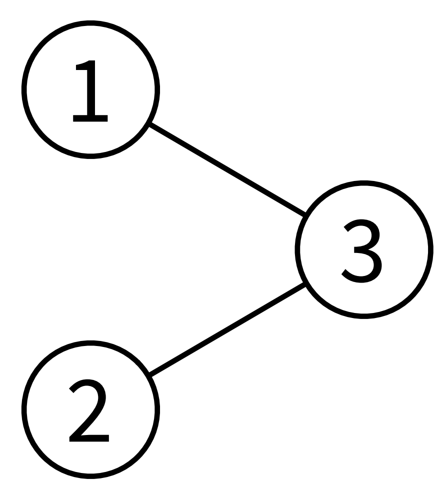
Subgraph

Graph *G'* = (*V'*, *E'*) is a "subgraph" of *G* = (*V*, *E*):
$V' \subseteq V$ and $E' \subseteq E$

G:

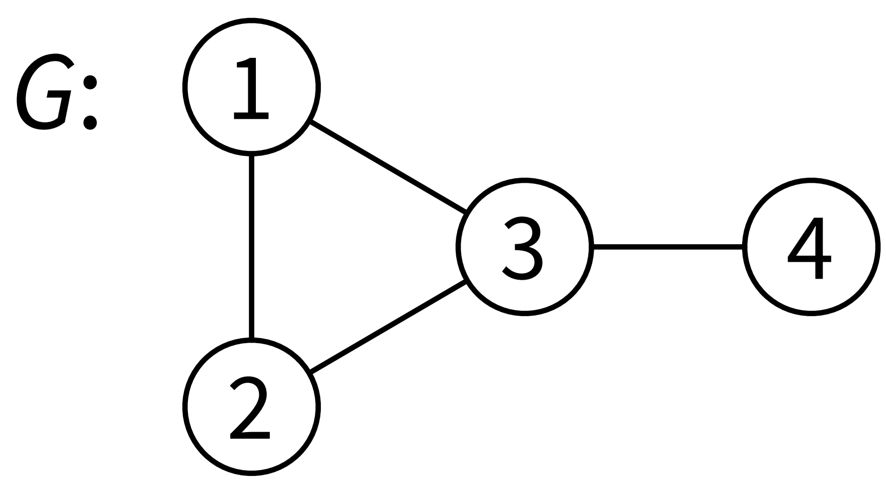


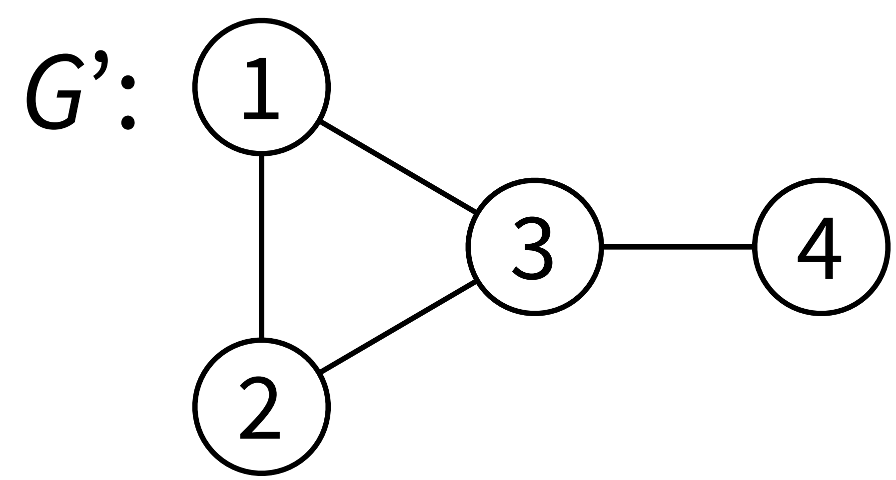
G':



Subgraph

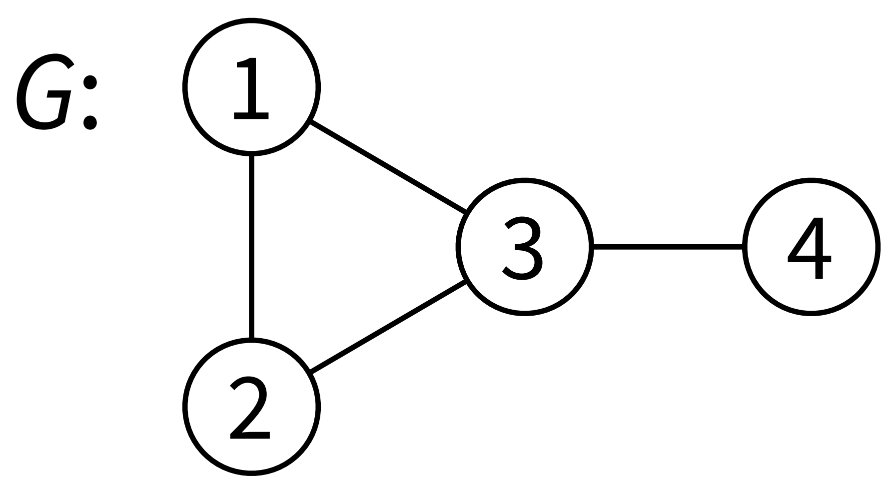
Graph $G' = (V', E')$ is a "subgraph" of $G = (V, E)$:
$V' \subseteq V$ and $E' \subseteq E$

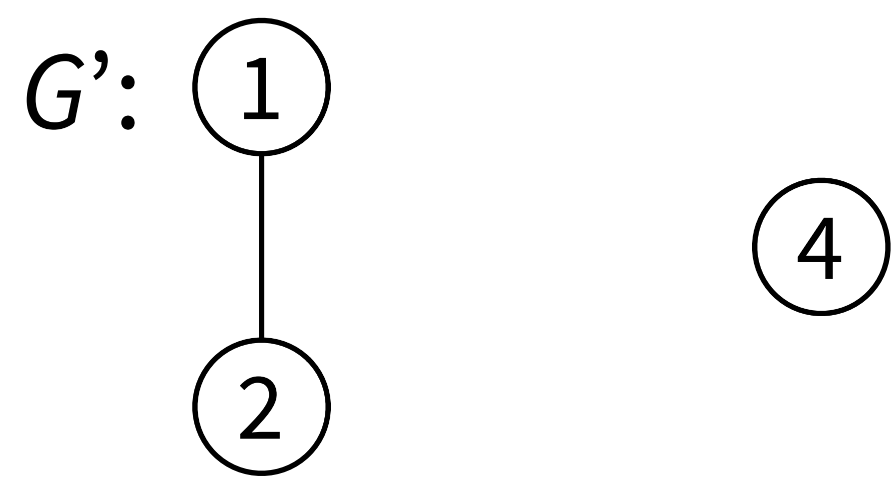




Subgraph

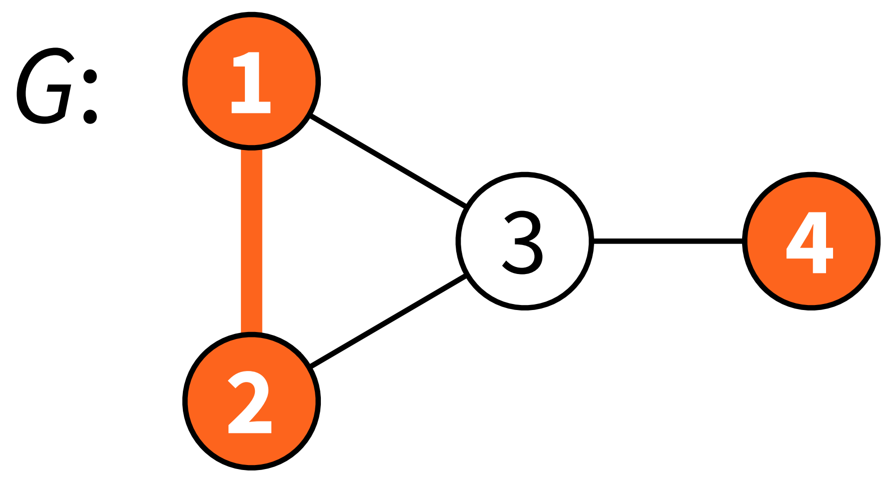
Graph *G'* = (*V'*, *E'*) is a "subgraph" of *G* = (*V*, *E*):
V' $\subseteq$ ***V*** **and *E'*** $\subseteq$ ***E***





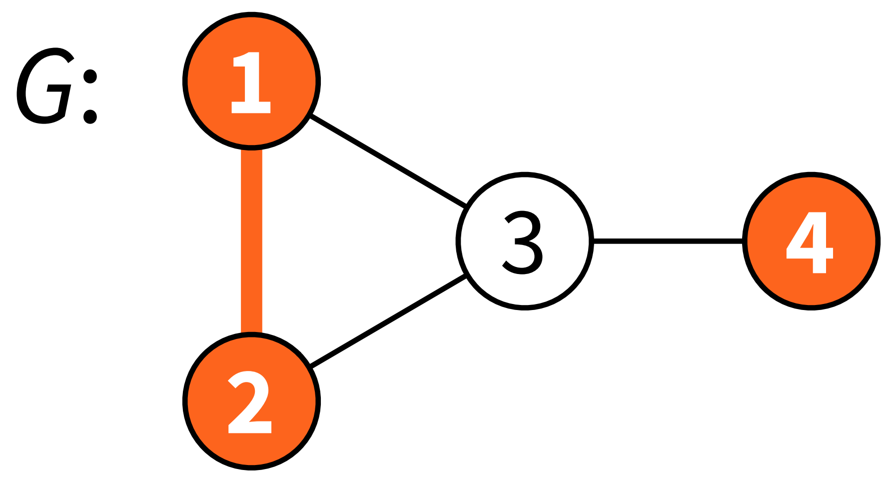
Subgraph

Graph $G' = (V', E')$ is a "subgraph" of $G = (V, E)$:
$V' \subseteq V$ and $E' \subseteq E$



Induced subgraph

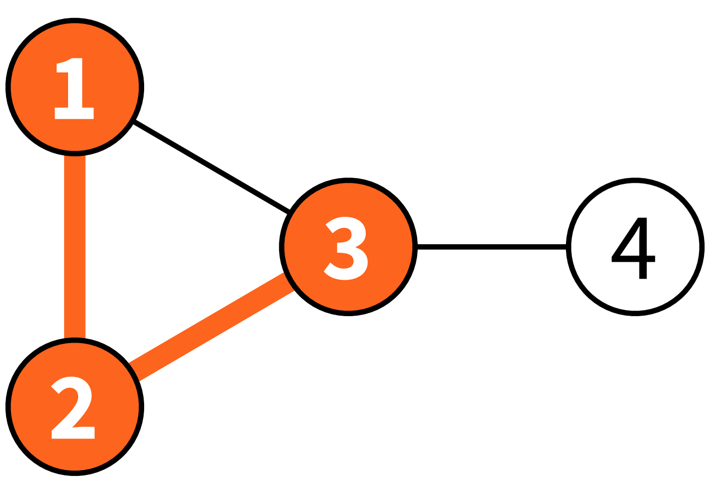
Subgraph "induced" by nodes *V'*
= all nodes of *V'* and all edges that connect them



Induced subgraph

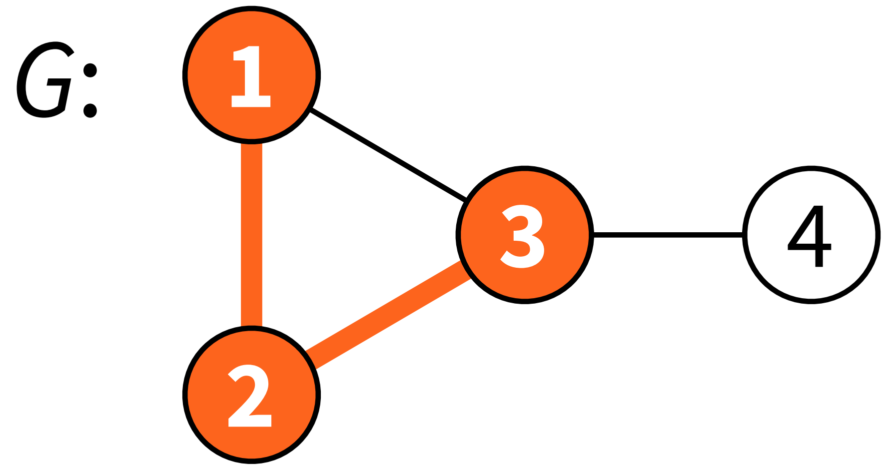
Subgraph "induced" by edges *E'*
= all edges of *E'* and all of their endpoints

G:



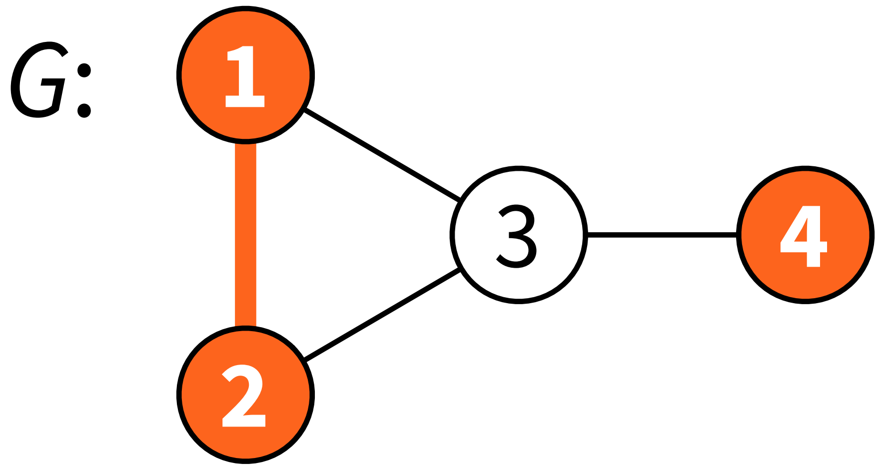
Induced subgraph

This is *not* a subgraph induced by any set of nodes — why?



Induced subgraph

This is *not* a subgraph induced by any set of edges — why?

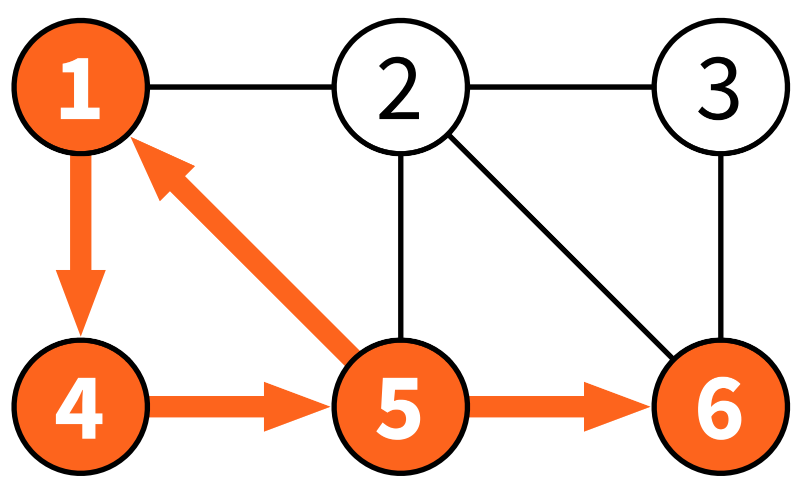


Walks, paths, and connectivity

"Walk" = alternating sequence of incident nodes and edges

$$w = (5, \{5,1\}, 1, \{1,4\}, 4, \{4,5\}, 5, \{5,6\}, 6)$$

G:



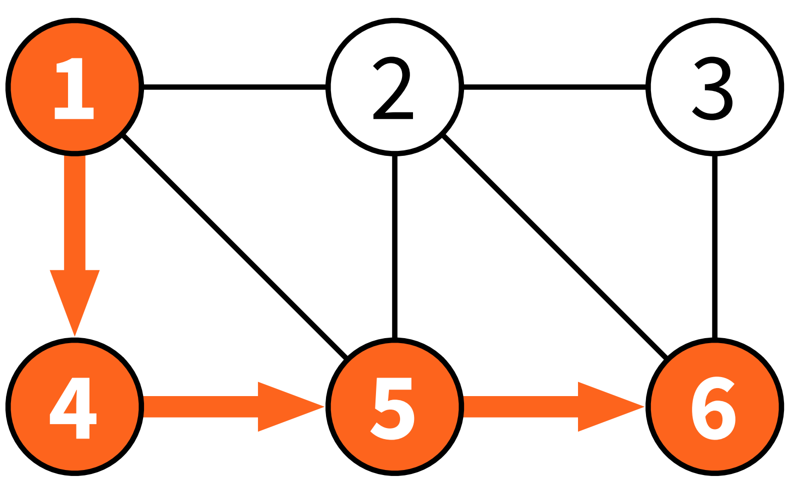
Walks, paths, and connectivity

"Path" = walk visiting each node at most once

"Length" of a path = number of *edges*

$w = (1, \{1,4\}, 4, \{4,5\}, 5, \{5,6\}, 6)$

G:

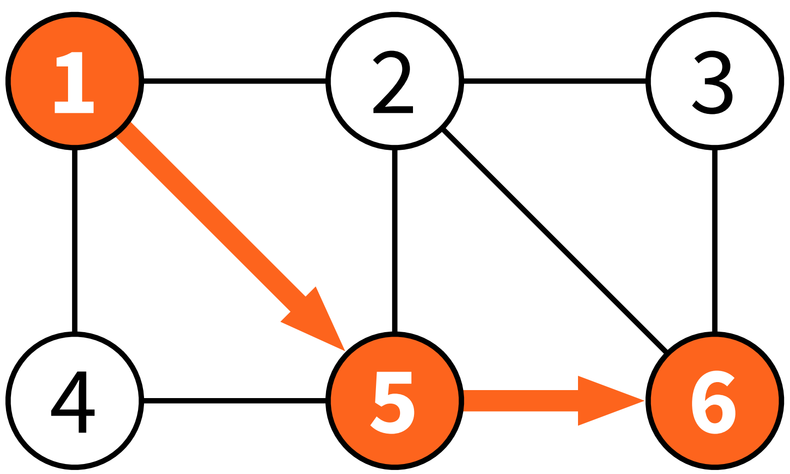


Walks, paths, and connectivity

“Distance” = length of a *shortest path*

$w = (1, \{1,5\}, 5, \{5,6\}, 6)$

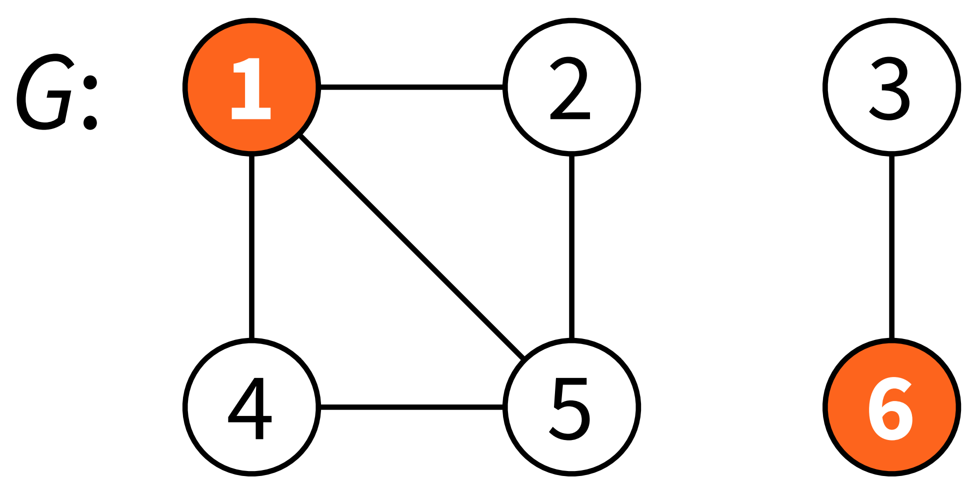
G:



dist(1, 6) = 2

Walks, paths, and connectivity

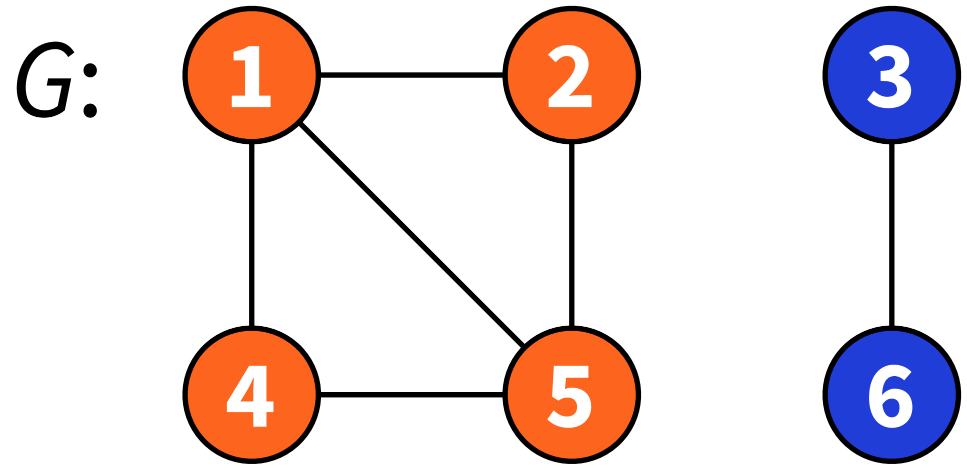
“Distance” = length of a *shortest path*
(infinite if no such path exists)



$\mathrm{dist}(1, 6) = \infty$

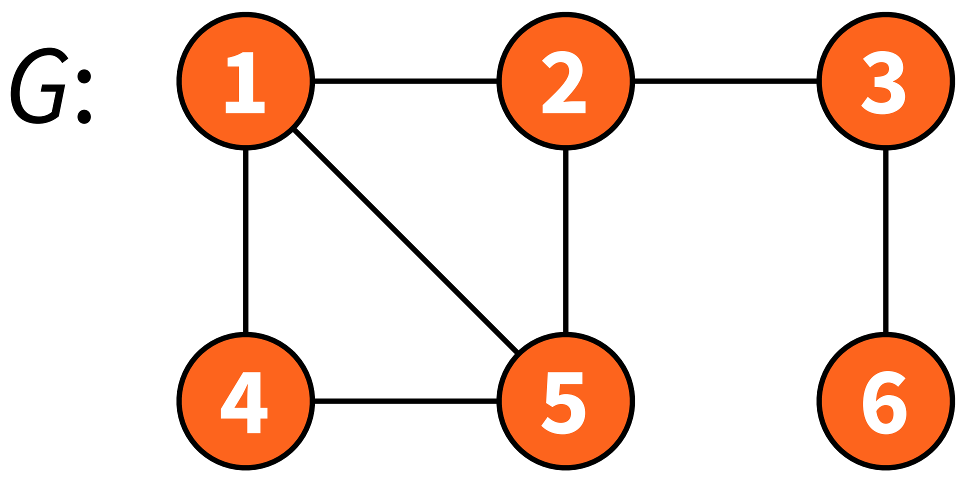
Walks, paths, and connectivity

“Connected component” *C*:
there is a path between any two nodes of *C*



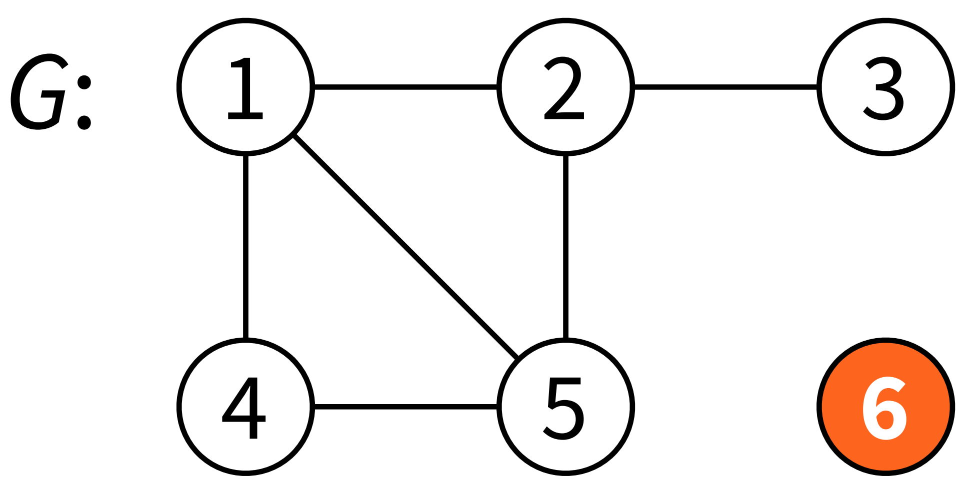
Walks, paths, and connectivity

Graph is "connected" if only 1 connected component



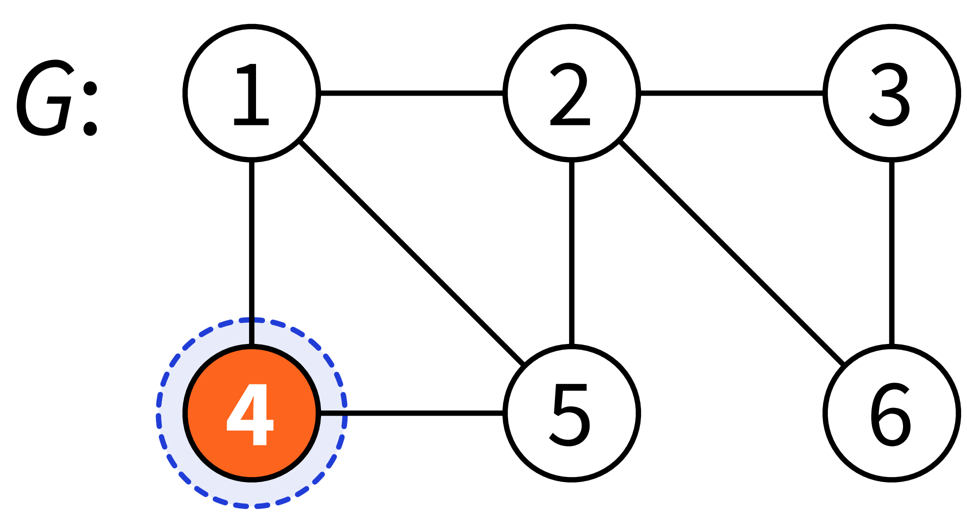
Walks, paths, and connectivity

“Isolated node” = node of degree 0



Walks, paths, and connectivity

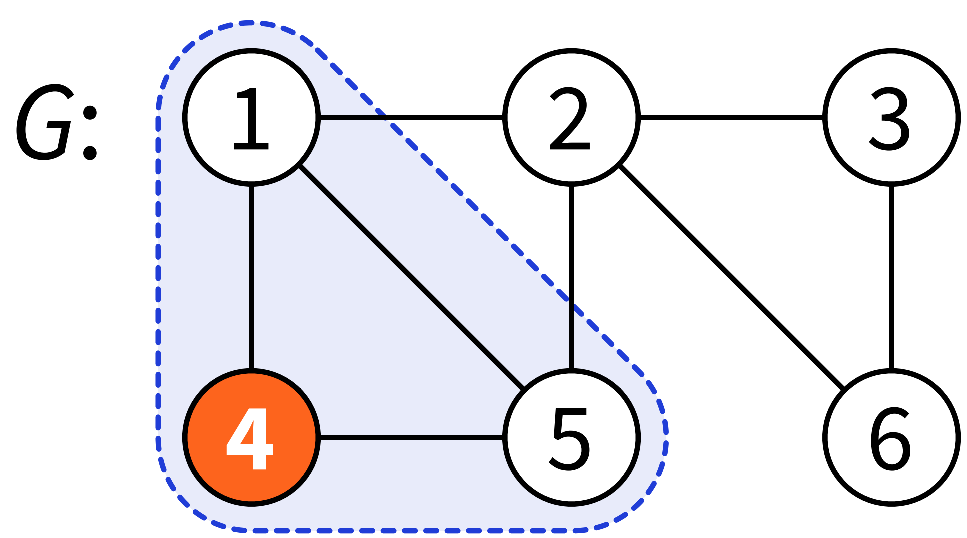
ball(v, r) = "radius-*r* neighbourhood of *v*"
= nodes at distance at most *r* from node *v*



ball(4, 0) = {4}
ball(4, 1) = {4, 1, 5}
ball(4, 2) = {4, 1, 5, 2}
ball(4, 3) = *V*

Walks, paths, and connectivity

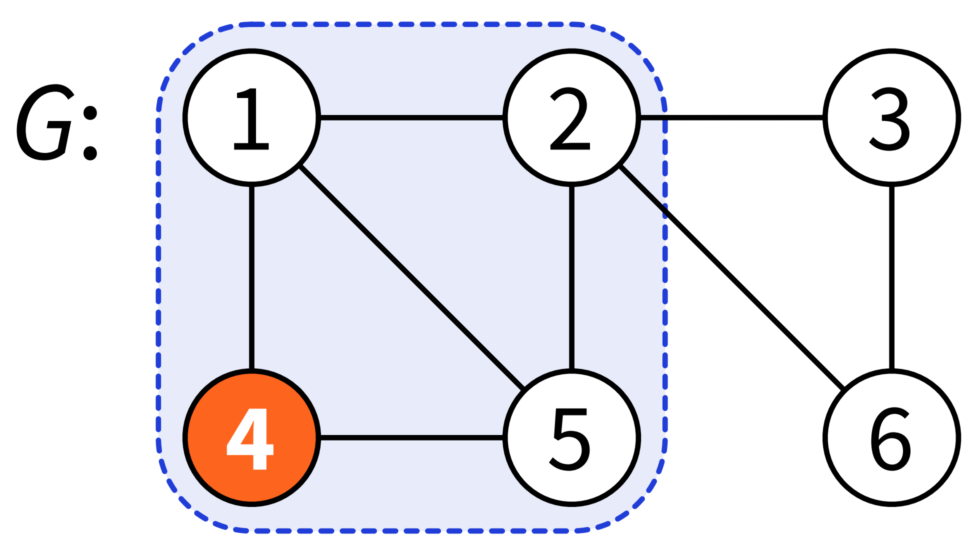
ball(v, r) = "radius-*r* neighbourhood of *v*"
= nodes at distance at most *r* from node *v*



ball(4, 0) = {4}
ball(4, 1) = {4, 1, 5}
ball(4, 2) = {4, 1, 5, 2}
ball(4, 3) = *V*

Walks, paths, and connectivity

ball(v, r) = “radius-*r* neighbourhood of *v*”
= nodes at distance at most *r* from node *v*

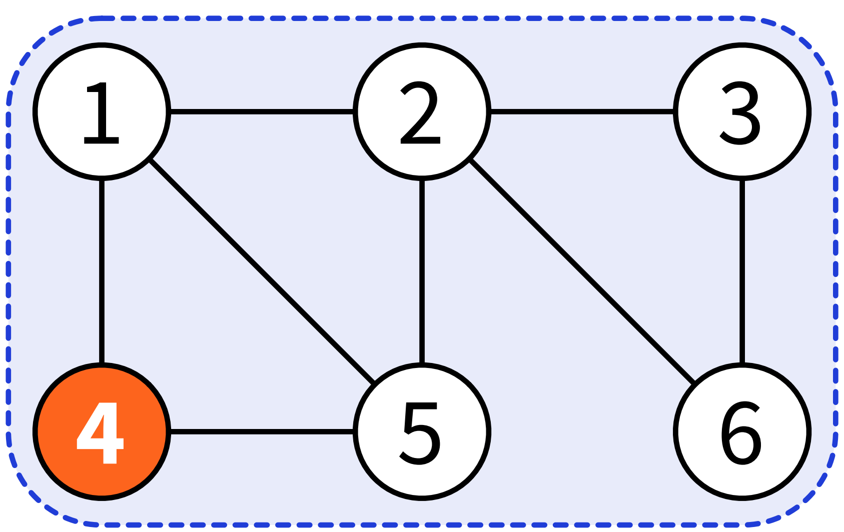


ball(4, 0) = {4}
ball(4, 1) = {4, 1, 5}
ball(4, 2) = {4, 1, 5, 2}
ball(4, 3) = *V*

Walks, paths, and connectivity

ball(v, r) = "radius-*r* neighbourhood of *v*"
= nodes at distance at most *r* from node *v*

G:

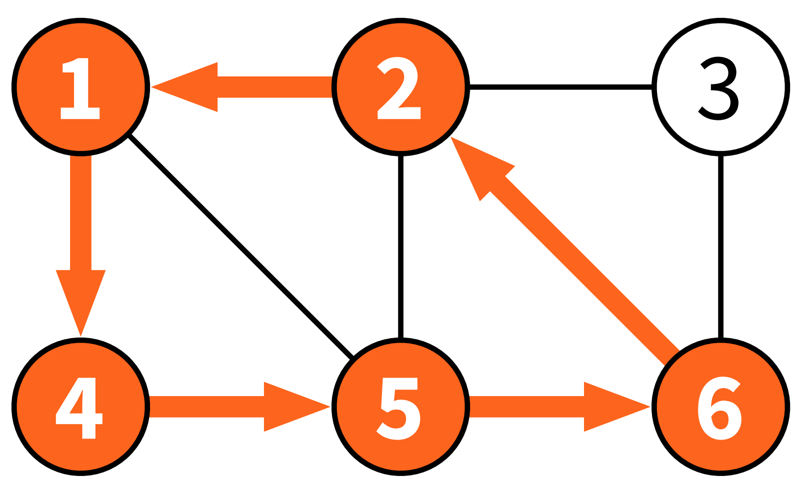


ball(4, 0) = {4}
ball(4, 1) = {4, 1, 5}
ball(4, 2) = {4, 1, 5, 2}
ball(4, 3) = *V*

Walks, paths, and connectivity

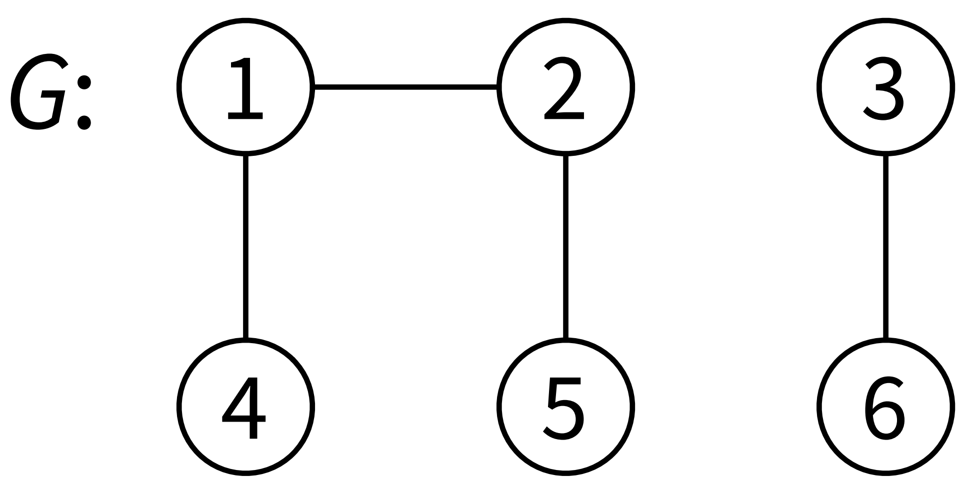
"Cycle" = closed walk that visits each node and each edge at most once (length ≥ 3)

G:



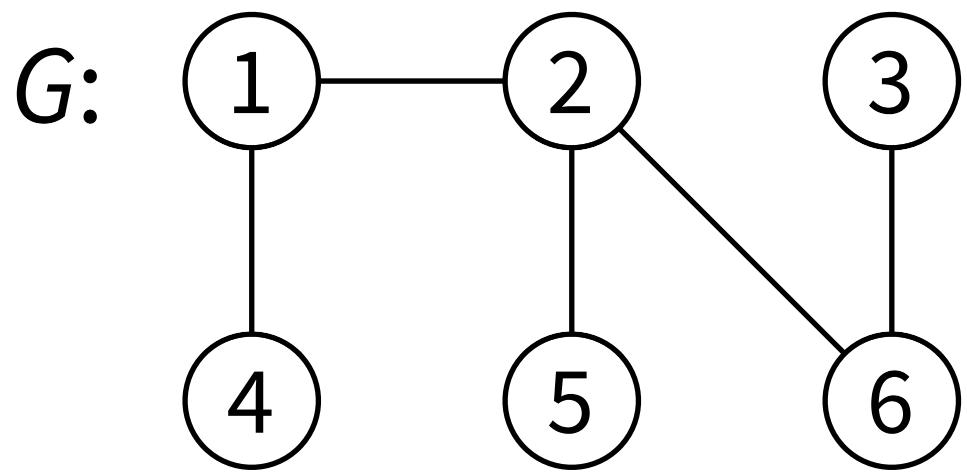
Walks, paths, and connectivity

"Acyclic graph" = graph without any cycles



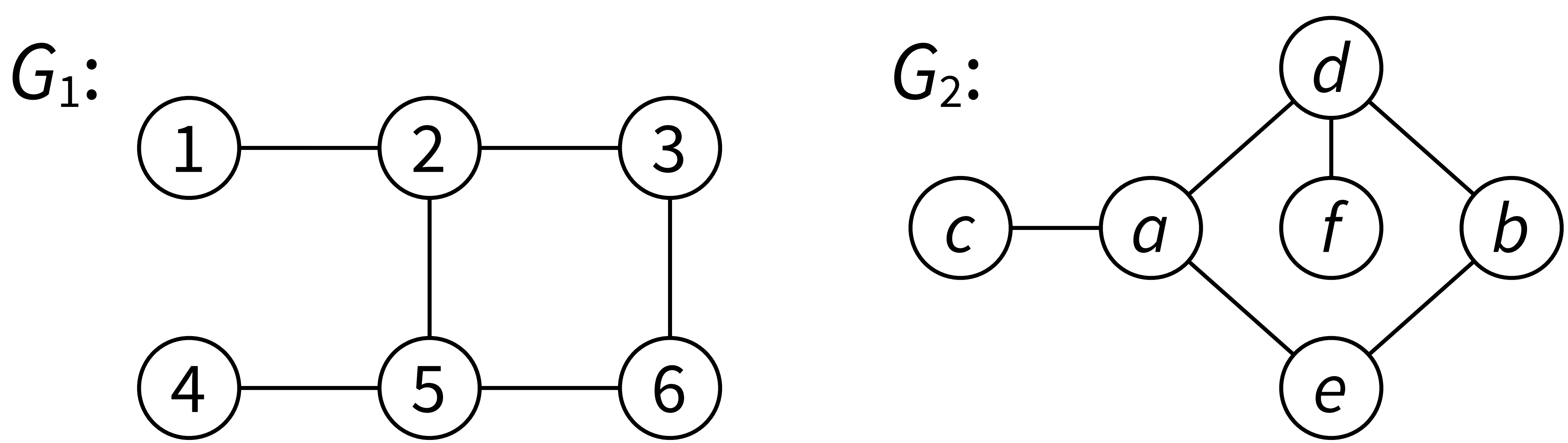
Walks, paths, and connectivity

"Tree" = connected acyclic graph
"Forest" = acyclic graph



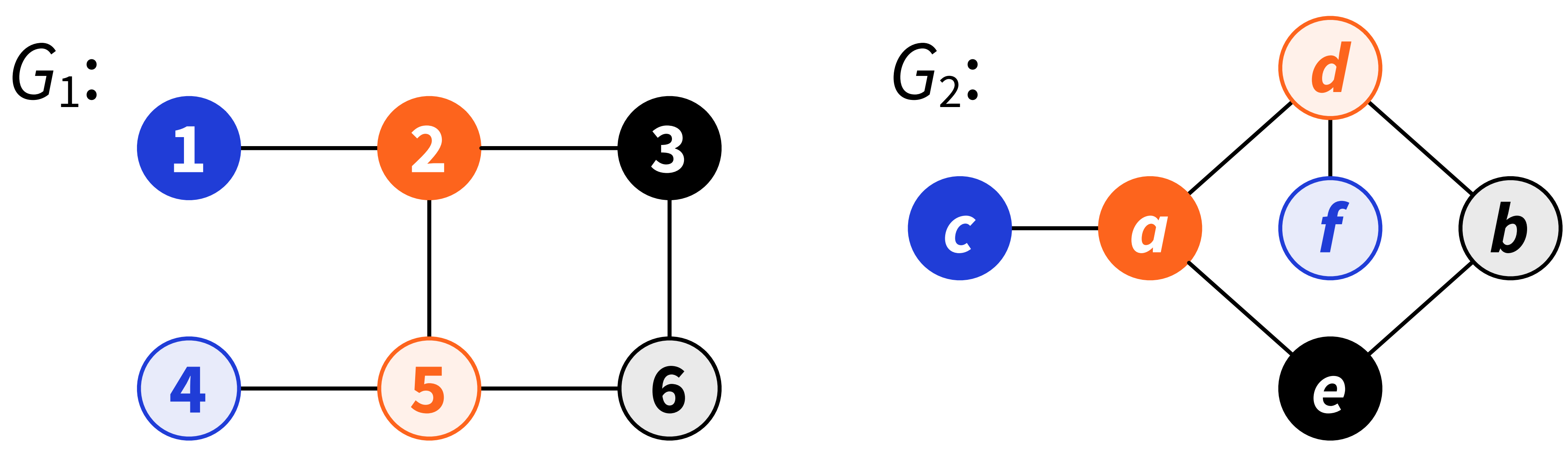
Isomorphism

"Isomorphism" from $G_1 = (V_1, E_1)$ to $G_2 = (V_2, E_2)$: bijection f: $V_1 \to V_2$ that preserves adjacency



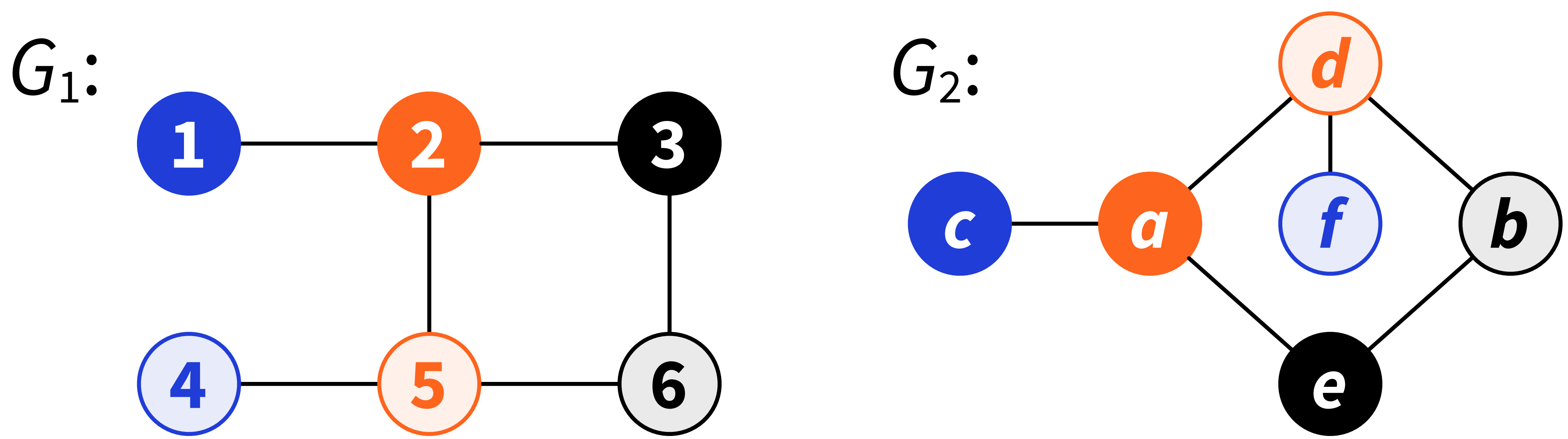
Isomorphism

"Isomorphism" from $G_1 = (V_1, E_1)$ to $G_2 = (V_2, E_2)$: bijection f: $V_1 \to V_2$ that preserves adjacency



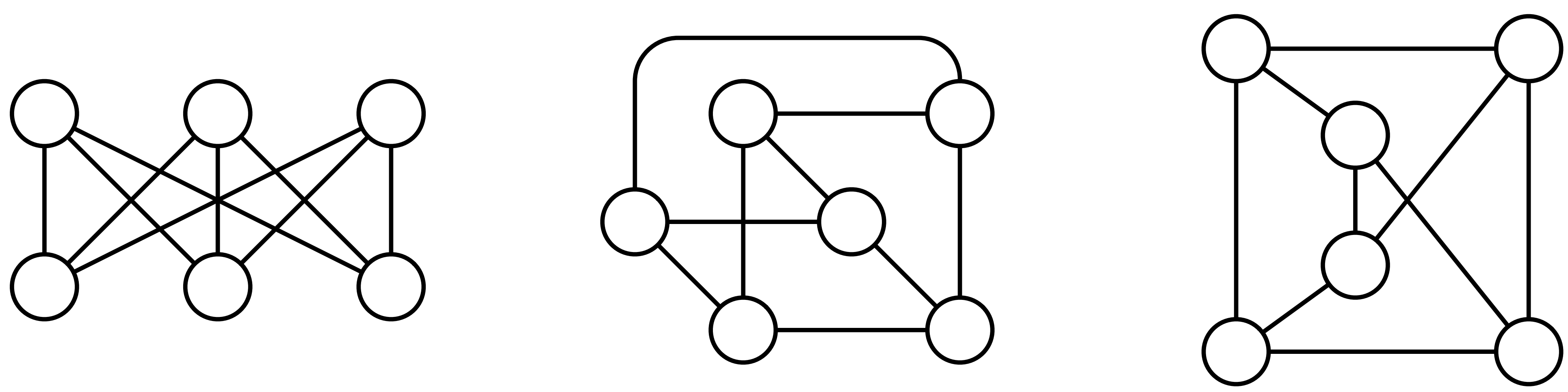
Isomorphism

Graphs are "isomorphic" if there exists an isomorphism form one to another



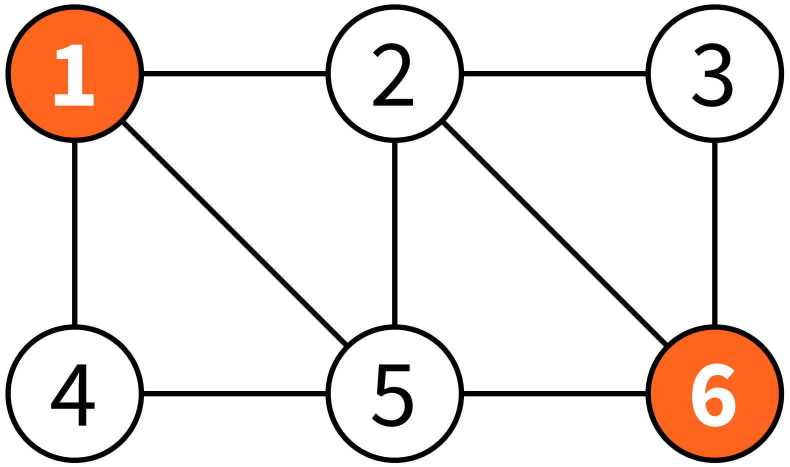
Isomorphism

Graphs are "isomorphic" if there exists an isomorphism form one to another



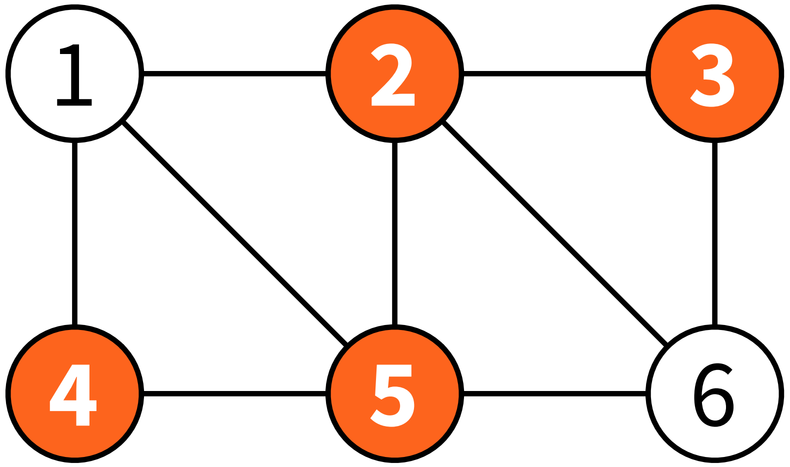
Graph problems

“Independent set”: non-adjacent nodes



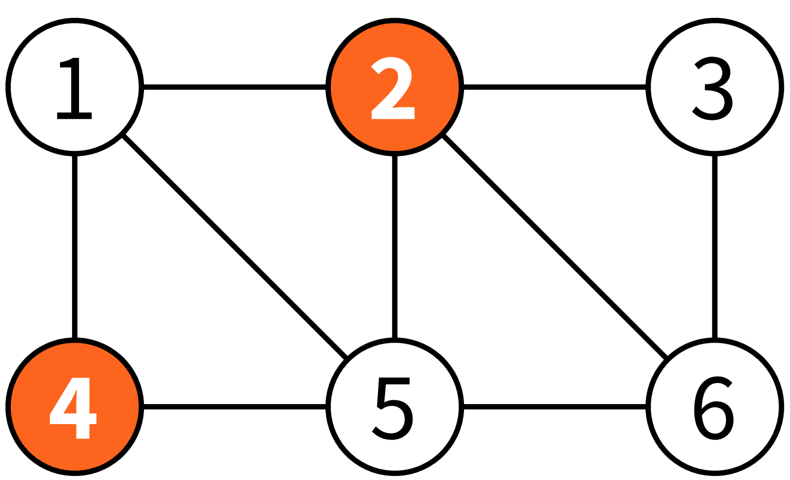
Graph problems

"Vertex cover": at least one endpoint of each edge (all edges are "covered" with these nodes)



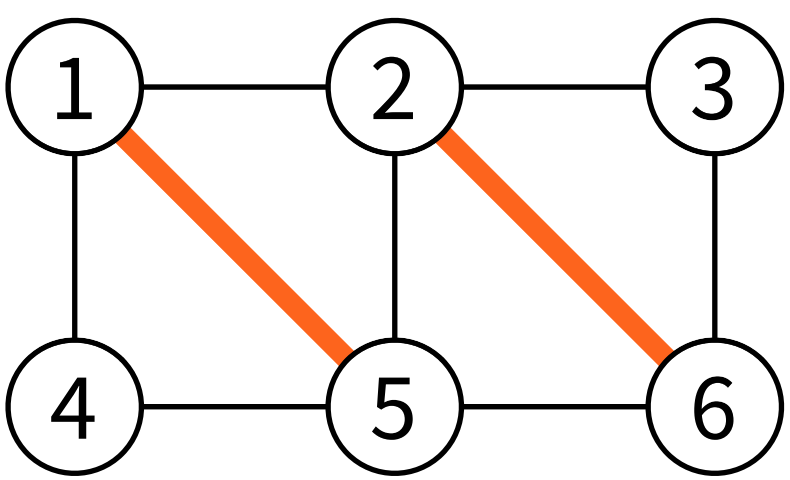
Graph problems

“Dominating set”: all other nodes have a neighbour in this set



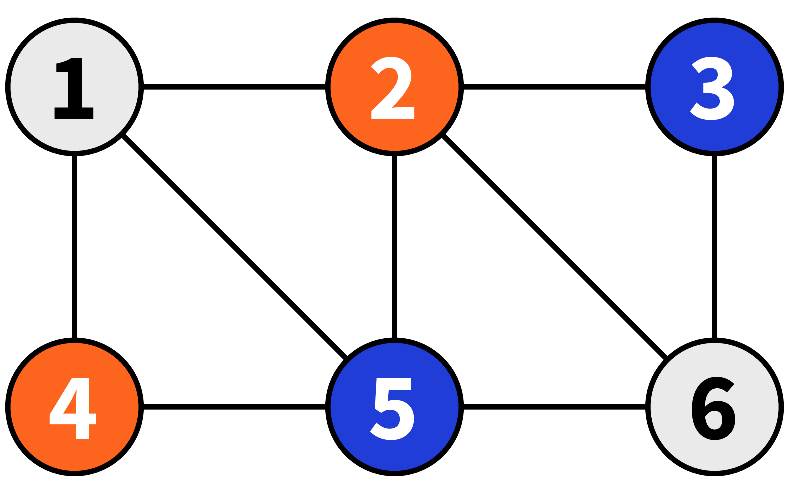
Graph problems

“Matching”: non-adjacent edges



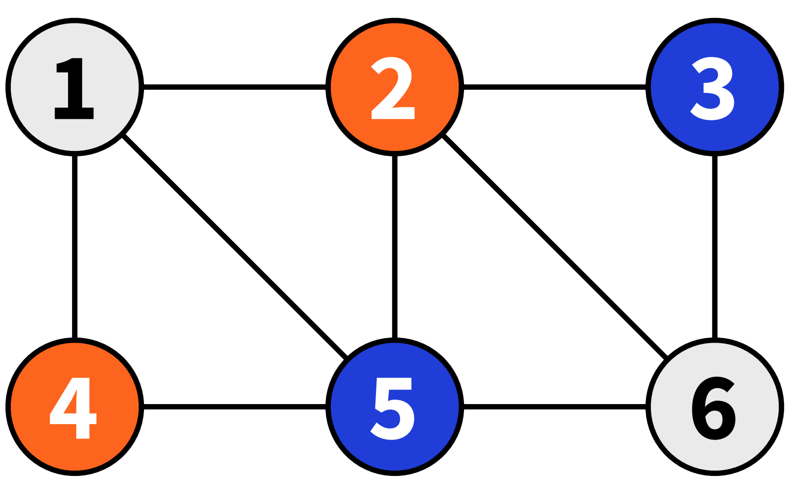
Graph problems

“Vertex colouring”:
adjacent nodes have different colours



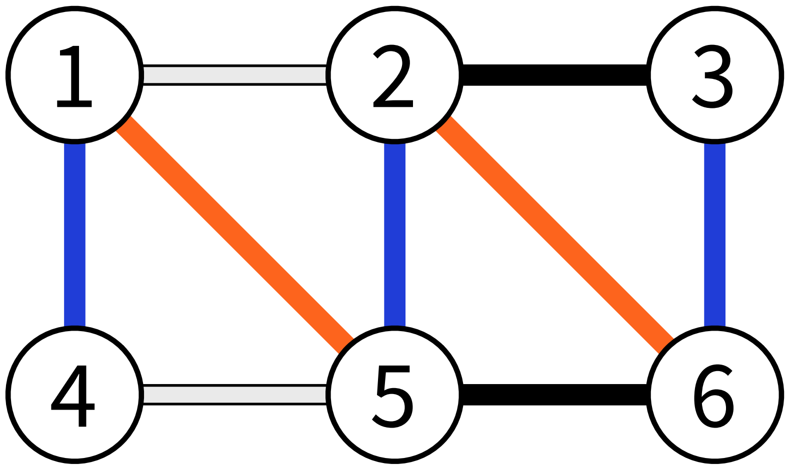
Graph problems

“Vertex colouring”:
each colour class is an independent set



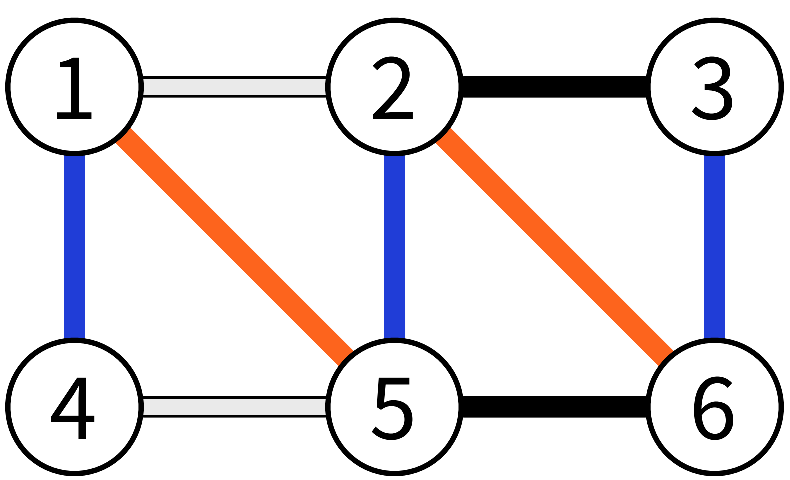
Graph problems

“Edge colouring”:
adjacent edges have different colours



Graph problems

“Edge colouring”:
each colour class is a matching

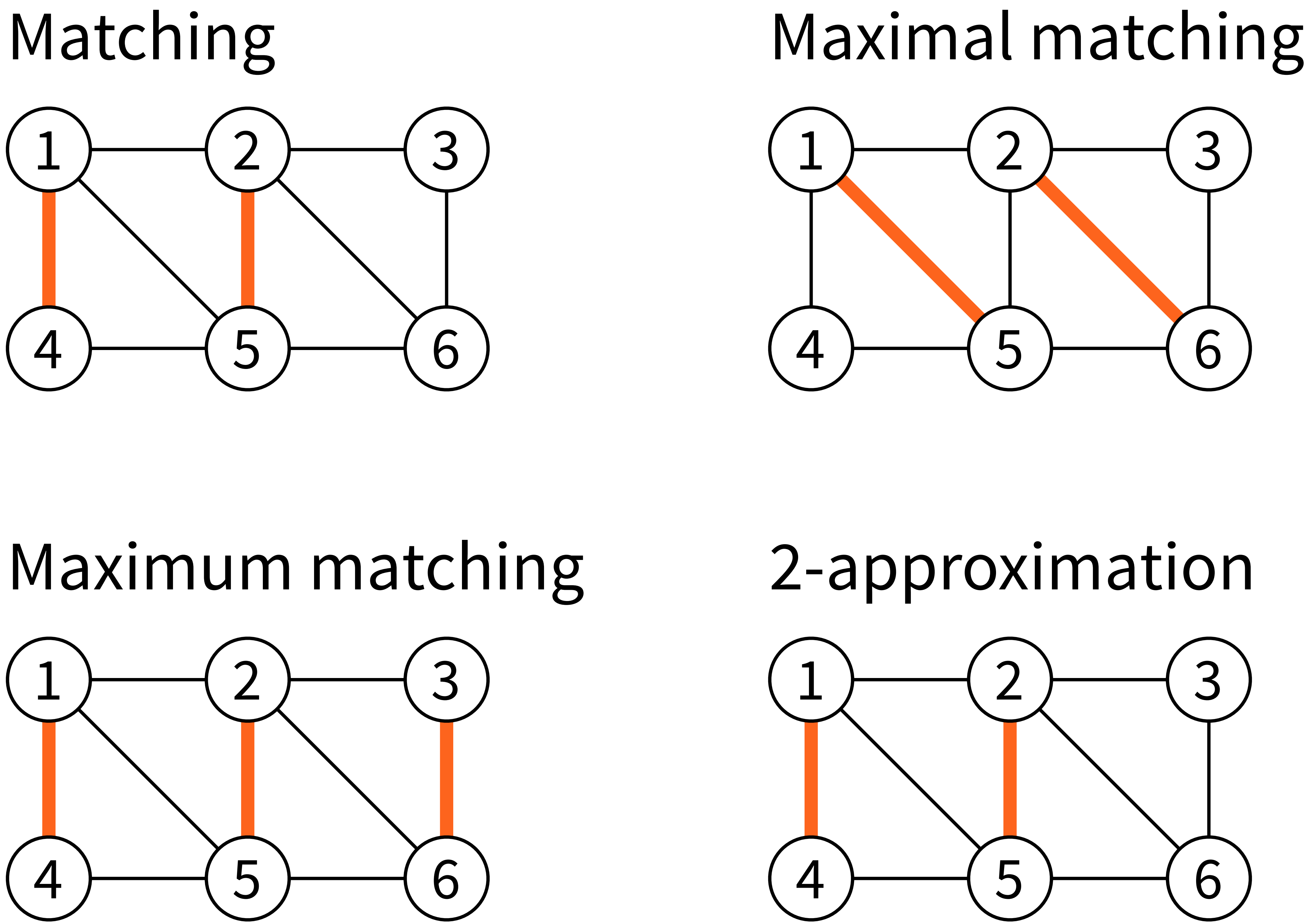


Graph problems

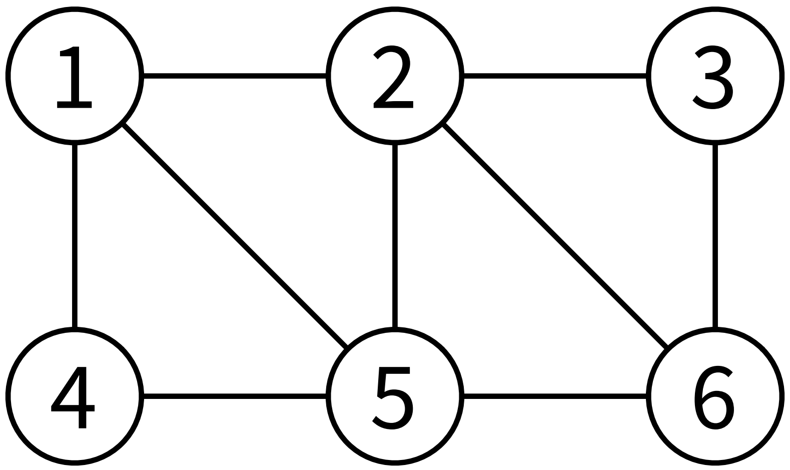
- **More definitions in the textbook:**
  - edge cover, edge dominating set
  - domatic partition, edge domatic partition
  - weak colouring
  - factorisation …

Maximisation problems

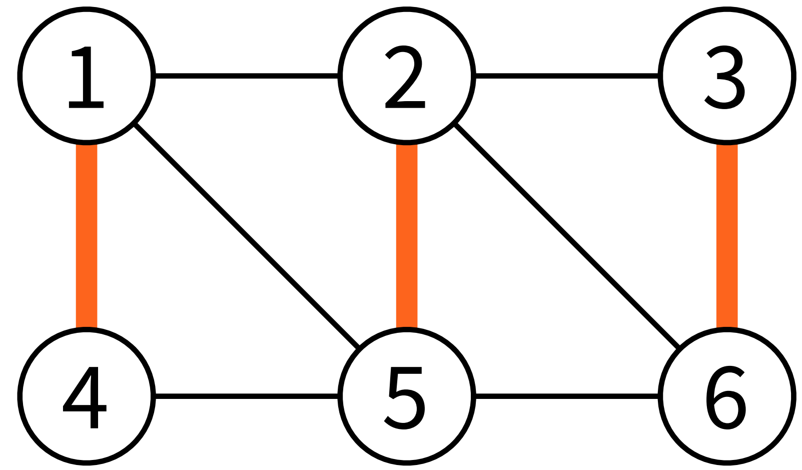
- **maximal** = cannot add anything
- **maximum** = largest possible size
- ***x*-approximation** =
  at least $1/x$ times maximum


Matching
1
2
3
4
5
6
Maximal matching
1
2
3
4
5
6
Maximum matching
1
2
3
4
5
6
2-approximation
1
2
3
4
5
6

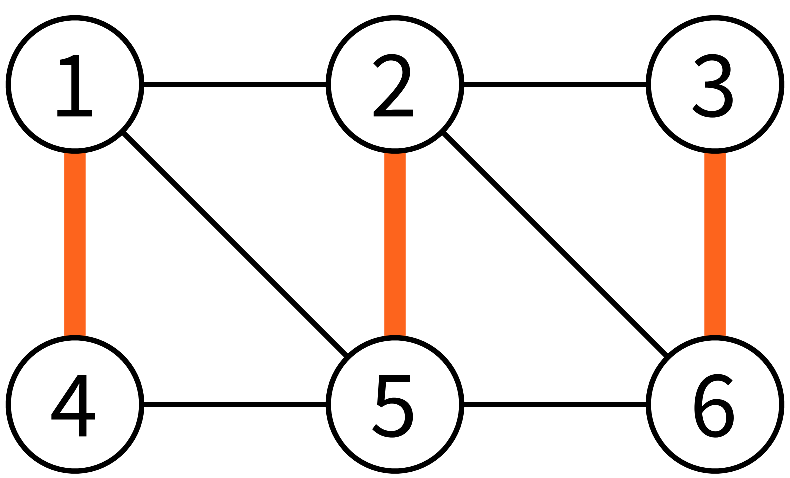
Matching



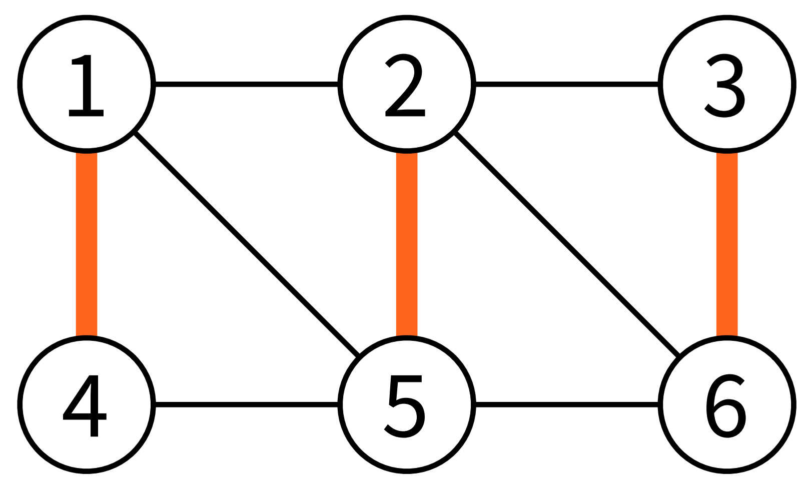
Maximal matching



Maximum matching



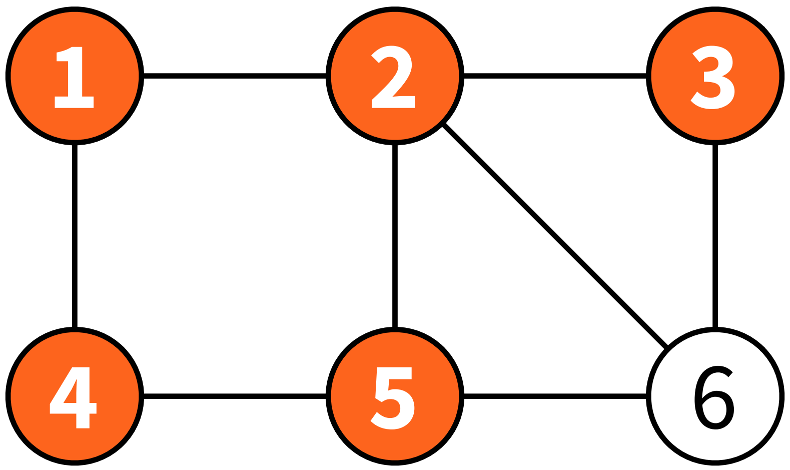
2-approximation



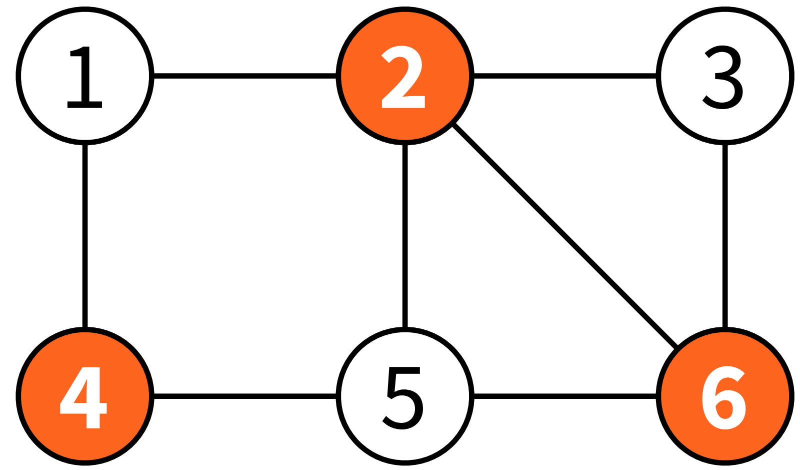
Minimisation problems

- **minimal** = cannot remove anything
- **minimum** = smallest possible size
- ***x*-approximation** =
  at most x times minimum

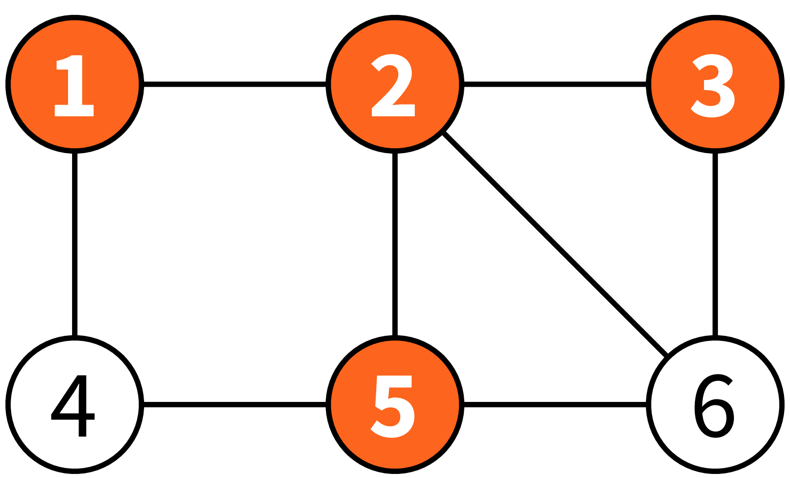
Vertex cover (VC)



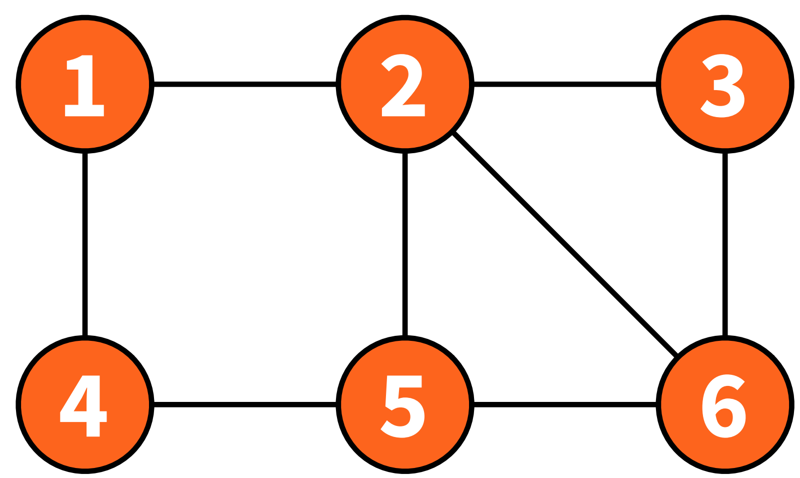
Minimum VC



Minimal VC



2-approximation

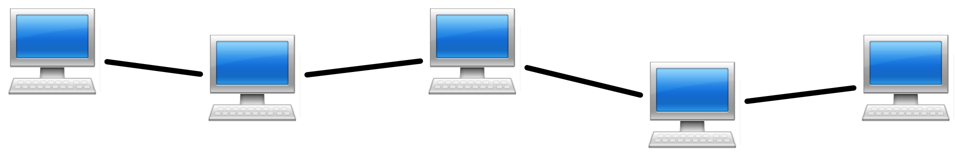


Approximation

- **Approximations are always feasible solutions!**
- **“2-approximation of minimum vertex cover”**
  - vertex cover
  - ≤ 2 times as large as minimum vertex cover

Graph theory and distributed algorithms

- **Network ≈ graph:** node ≈ computer, edge ≈ link
- **Graph theory used to:**
  - define: model of computing,
    what we want to solve, what we assume …
  - prove: correctness of algorithms,
    time complexity, impossibility results …

- **Weeks 1–2: informal introduction**
  - network = path 
- **Week 3: graph theory**
- **Weeks 4–7: models of computing**
  - what can be computed (efficiently)?
- **Weeks 8–11: lower bounds**
  - what cannot be computed (efficiently)?
- **Week 12: recap**