# INF421, Lecture 3
# Graphs

Leo Liberti

LIX, École Polytechnique, France

# Course

- **Objective**: teach notions AND develop intelligence
- **Evaluation**: TP noté en salle info, Contrôle à la fin. Note: $\max(CC, \frac{3}{4}CC + \frac{1}{4}TP)$
- **Organization**: fri 31/8, 7/9, 14/9, 21/9, 28/9, 5/10, 12/10, 19/10, 26/10, amphi 1030-12 (Arago), TD 1330-1530, 1545-1745 (SI:30-34)
- **Books**:
  1. K. Mehlhorn & P. Sanders, *Algorithms and Data Structures*, Springer, 2008
  2. D. Knuth, *The Art of Computer Programming*, Addison-Wesley, 1997
  3. G. Dowek, *Les principes des langages de programmation*, Editions de l'X, 2008
  4. Ph. Baptiste & L. Maranget, *Programmation et Algorithmique*, Ecole Polytechnique (Polycopié), 2006
- **Website**: `www.enseignement.polytechnique.fr/informatique/INF421`
- **Blog**: `inf421.wordpress.com`
- **Contact**: `liberti@lix.polytechnique.fr` (e-mail subject: INF421)

# Lecture summary

- Graph definitions
- Operations on graphs
- Combinatorial problems on graphs
- Easy and hard problems
- Modelling problems for a generic solution method

# The minimal knowledge

- **Operations on graphs**: complement, line graph, contraction
- **Decision/optimization problems**: finding subgraphs with given properties
- **Easy problems**: solvable in polynomial time (**P**), e.g. minimum cost spanning tree, shortest paths, maximum matching
- **Hard problems**: efficient method for solving one would solve all of them (**NP**-hard), e.g. maximum clique, maximum stable set, vertex colouring
- **Mathematical Programming**: a generic model-and-solve approach

# Graph definitions

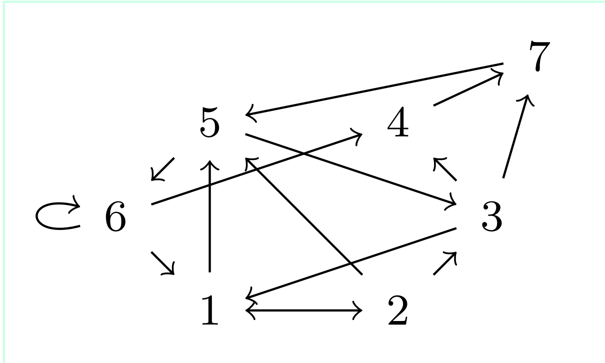
# Motivation

## The ultimate data structure

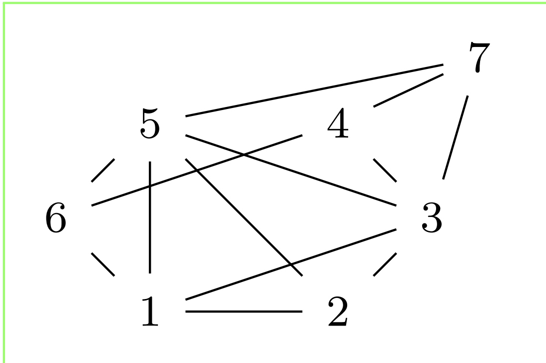
Most data structures can be represented by graphs

# Graphs and digraphs

- **Digraph** $G = (V, A)$: relation $A$ on set $V$
  - $V$: set of **nodes**
  - $A$: set of **arcs** $(u, v)$ with $u, v \in V$




- Graph $G = (V, E)$: symmetric relation $E$ on set $V$
  - $V$: set of **vertices**
  - $E$: set of **edges** $\{u, v\}$ with $u, v \in V$




- **Simple** (di)graphs: relation is *irreflexive*
  (I.e., $v$ *not related to itself for all* $v \in V$)

# Remarks

- Mainly, results for **undirected** graphs
- Many trivial extensions to digraphs
- **Warning**: not all trivial

Example

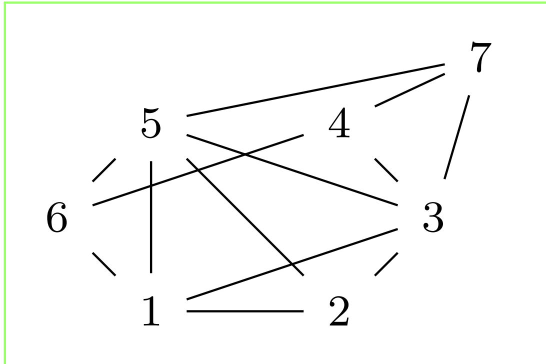
- $G$ a graph: $V(G)$ set of vertices, $E(G)$ set of edges
- **Extension to digraphs**:
  $V(G)$ set of nodes, $A(G)$ set of arcs

# Stars

**Stars**: vertices or edges adjacent to a given vertex

$\forall v \in V(G)$,

- if $G$ is undirected, **neighbourhood**

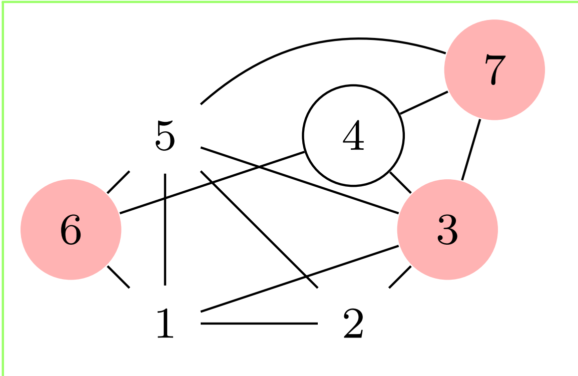


# Stars

**Stars**: vertices or edges adjacent to a given vertex

$\forall v \in V(G)$,

- if $G$ is undirected, **neighbourhood**
  - $N(v) = \{u \in V \mid \{u, v\} \in E(G)\}$

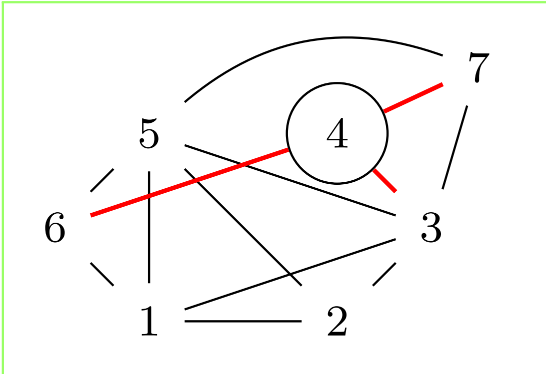


# Stars

**Stars**: vertices or edges adjacent to a given vertex

$\forall v \in V(G)$,

- if $G$ is undirected, **neighbourhood**
  - $N(v) = \{u \in V \mid \{u, v\} \in E(G)\}$
  - **Cutset** $\delta(v) = \{\{u, v\} \mid u \in N(v)\}$

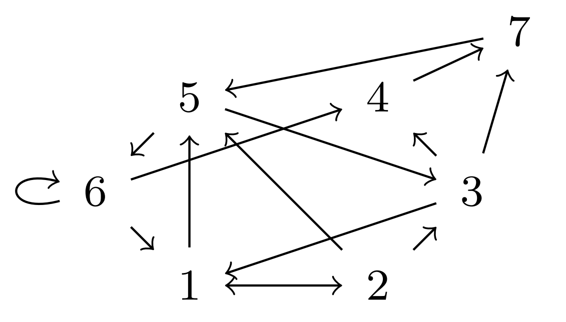


# Stars

**Stars**: vertices or edges adjacent to a given vertex

$\forall v \in V(G)$,

- if $G$ is undirected, **neighbourhood**
  - $N(v) = \{u \in V \mid \{u,v\} \in E(G)\}$
  - **Cutset** $\delta(v) = \{\{u,v\} \mid u \in N(v)\}$
- if $G$ is directed,

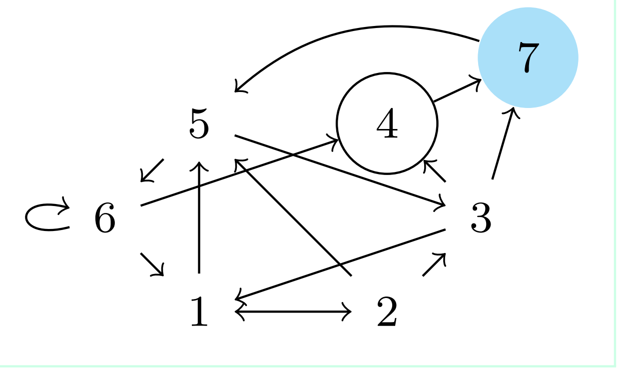


# Stars

**Stars**: vertices or edges adjacent to a given vertex

$\forall v \in V(G)$,

- if $G$ is undirected, **neighbourhood**
  - $N(v) = \{u \in V \mid \{u, v\} \in E(G)\}$
  - **Cutset** $\delta(v) = \{\{u, v\} \mid u \in N(v)\}$
- if $G$ is directed,
  - $N^+(v) = \{u \in V \mid (v, u) \in E(G)\}$

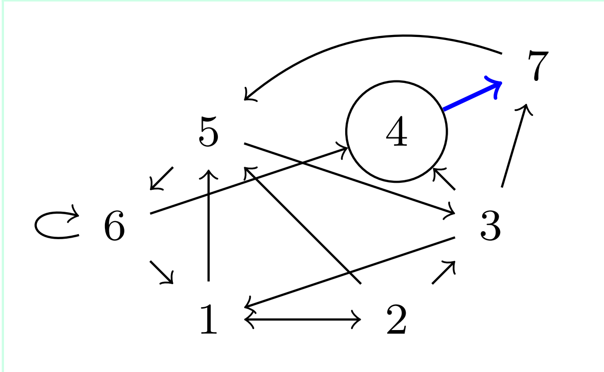


# Stars

**Stars**: vertices or edges adjacent to a given vertex

$\forall v \in V(G)$,

- if $G$ is undirected, **neighbourhood**
  - $N(v) = \{u \in V \mid \{u, v\} \in E(G)\}$
  - **Cutset** $\delta(v) = \{\{u, v\} \mid u \in N(v)\}$
- if $G$ is directed,
  - $N^+(v) = \{u \in V \mid (v, u) \in E(G)\}$
  - $\delta^+(v) = \{(v, u) \mid u \in N^+(v)\}$

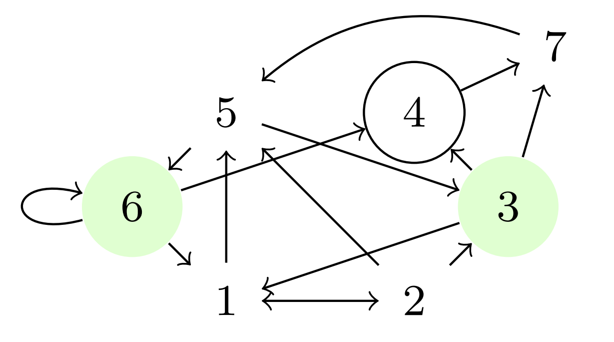


# Stars

**Stars**: vertices or edges adjacent to a given vertex

$\forall v \in V(G)$,

- if $G$ is undirected, **neighbourhood**
  - $N(v) = \{u \in V \mid \{u,v\} \in E(G)\}$
  - **Cutset** $\delta(v) = \{\{u,v\} \mid u \in N(v)\}$
- if $G$ is directed,
  - $N^+(v) = \{u \in V \mid (v,u) \in E(G)\}$
  - $\delta^+(v) = \{(v,u) \mid u \in N^+(v)\}$
  - $N^-(v) = \{u \in V \mid (u,v) \in E(G)\}$

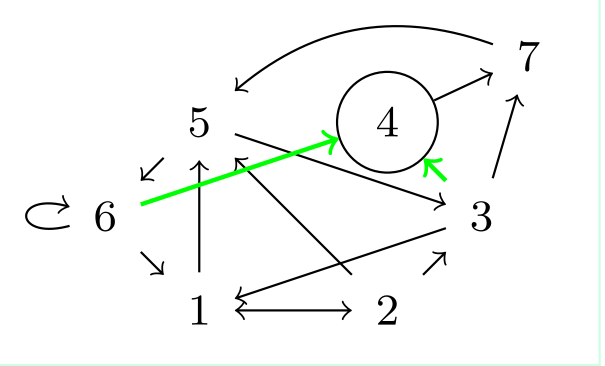


# Stars

**Stars**: vertices or edges adjacent to a given vertex

$\forall v \in V(G)$,

- if $G$ is undirected, **neighbourhood**
  - $N(v) = \{u \in V \mid \{u, v\} \in E(G)\}$
  - **Cutset** $\delta(v) = \{\{u, v\} \mid u \in N(v)\}$
- if $G$ is directed,
  - $N^+(v) = \{u \in V \mid (v, u) \in E(G)\}$
  - $\delta^+(v) = \{(v, u) \mid u \in N^+(v)\}$
  - $N^-(v) = \{u \in V \mid (u, v) \in E(G)\}$
  - $\delta^-(v) = \{(u, v) \mid u \in N^-(v)\}$



# Stars

**Stars**: vertices or edges adjacent to a given vertex

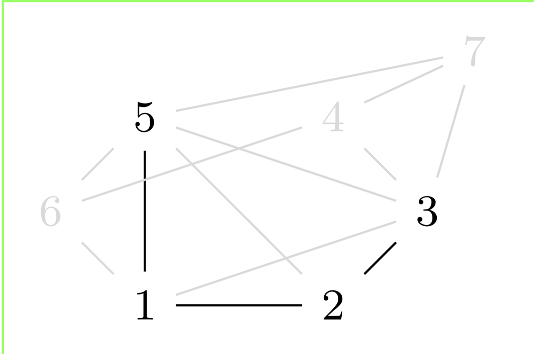
$\forall v \in V(G)$,

- if $G$ is undirected, **neighbourhood**
  - $N(v) = \{u \in V \mid \{u, v\} \in E(G)\}$
  - **Cutset** $\delta(v) = \{\{u, v\} \mid u \in N(v)\}$
- if $G$ is directed,
  - $N^+(v) = \{u \in V \mid (v, u) \in E(G)\}$
  - $\delta^+(v) = \{(v, u) \mid u \in N^+(v)\}$
  - $N^-(v) = \{u \in V \mid (u, v) \in E(G)\}$
  - $\delta^-(v) = \{(u, v) \mid u \in N^-(v)\}$
- $|N(v)|$ =**degree**, $|N^+(v)|$ =**outdegree**, $|N^-(v)|$ =**indegree** of $v$
- If $v$ in both $G, H$ write $N_G(v)$ and $N_H(v)$

  *(similarly for other star notation)*

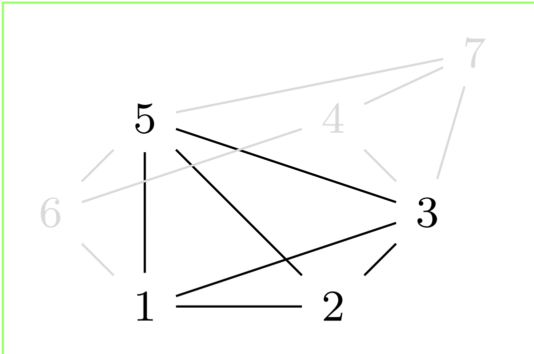
# Subgraphs

- **Subgraph** $H = (U, F)$ of $G = (V, E)$ if $H$ a graph s.t. $U \subseteq V \wedge F \subseteq E$




- **Spanning** subgraph $H = (U, F)$ of $G = (V, E)$: $U = V$
- Subgraph $H = (U, F)$ of $G = (V, E)$ **induced** by $U$:

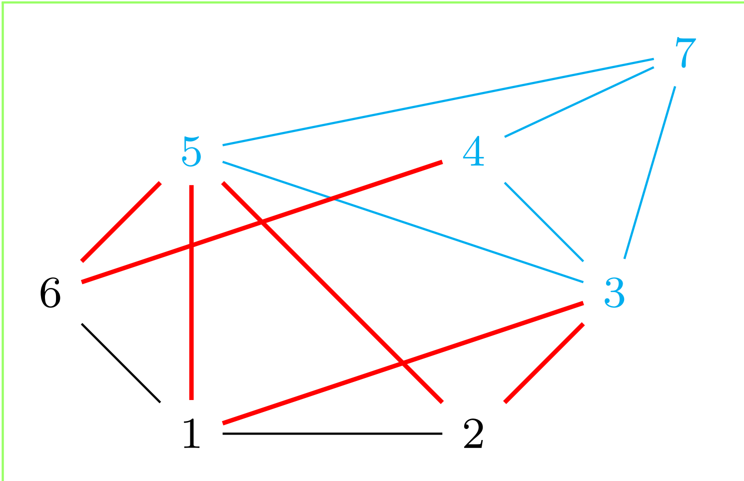
$$\forall u, v \in U \ (\{u, v\} \in E \rightarrow \{u, v\} \in F)$$




*Induced subgraph notation*: $H = G[U]$

# Cutsets

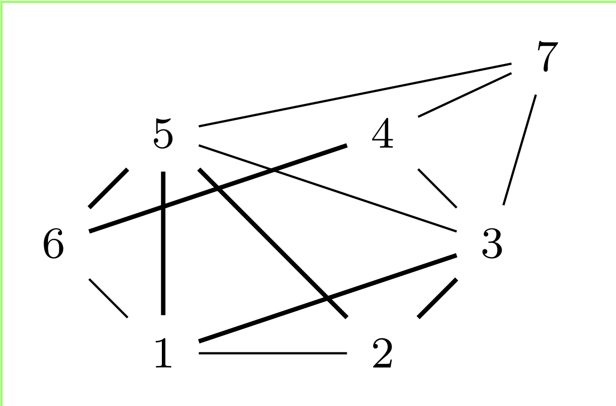
- $H = (U, F)$ a subgraph of $G = (V, E)$
- **Cutset**: $\delta(H) = \left( \bigcup_{u \in U} \delta(u) \right) \smallsetminus F$
  edge set "separating" $U$ and $V \smallsetminus U$
- E.g. $U = \{1, 2, 6\}$ and $H = G[U]$, then $\delta(H)$ shown in red



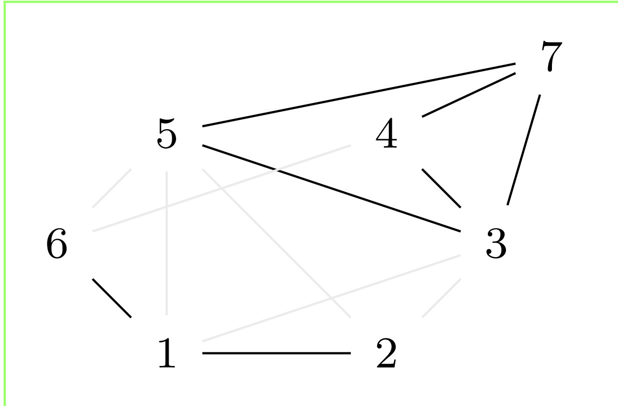

- Similar definitions for **directed cutsets**
  Thm.
- If $G$ is undirected then $\forall\, U \subseteq V(G) \quad \delta(U) = \delta(V \smallsetminus U)$

# Connectedness

- **Connected**: $\nexists$ empty nontrivial cutsets




*Connected*




*Not connected:* $\delta(\{1,2,6\}) = \varnothing$

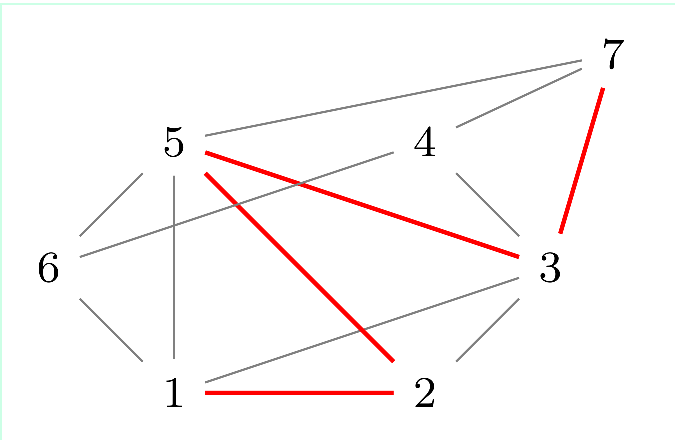
- **Connected component**: maximal connected subgraph

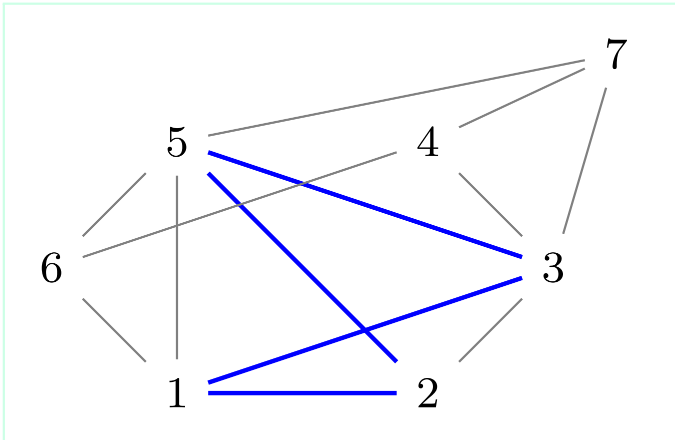
Most algorithms assume connected graphs

If not, apply alg. to each connected component

# Paths and cycles

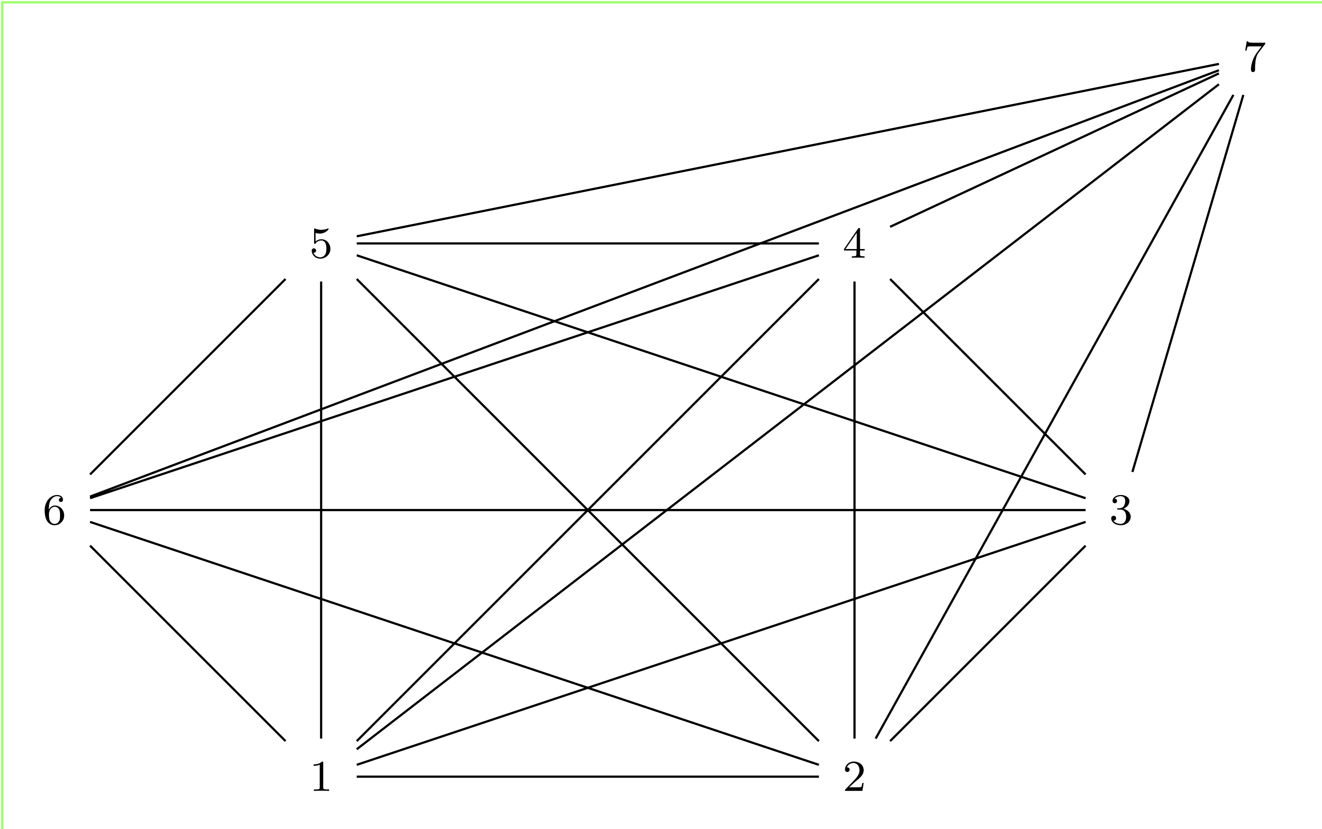
- $G$ a graph and $u, v \in V(G)$
- A **simple path $P$ from $u$ to $v$** in $G$: connected subgraph of $G$ s.t.:
  1. $\forall w \in V(P)\ (w \neq u \wedge w \neq v \rightarrow |N(w)| = 2)$
  2. if $u \neq v$ then $|N(u)| = |N(v)| = 1$
  3. if $u = v$ then $|N(u)| = |N(v)| = 2$
- Notation: path from $u$ to $v$: $P : u \rightarrow v$
- In $P : u \rightarrow v$, **$u, v$ are endpoints**
- A **simple cycle** is a simple path with equal endpoints
- Mostly, say paths/cycles to mean *simple* ones



# Complete graph

- **Complete graph** or $n$**-clique** $K_n$ on $n$ vertices:
  all possible edges

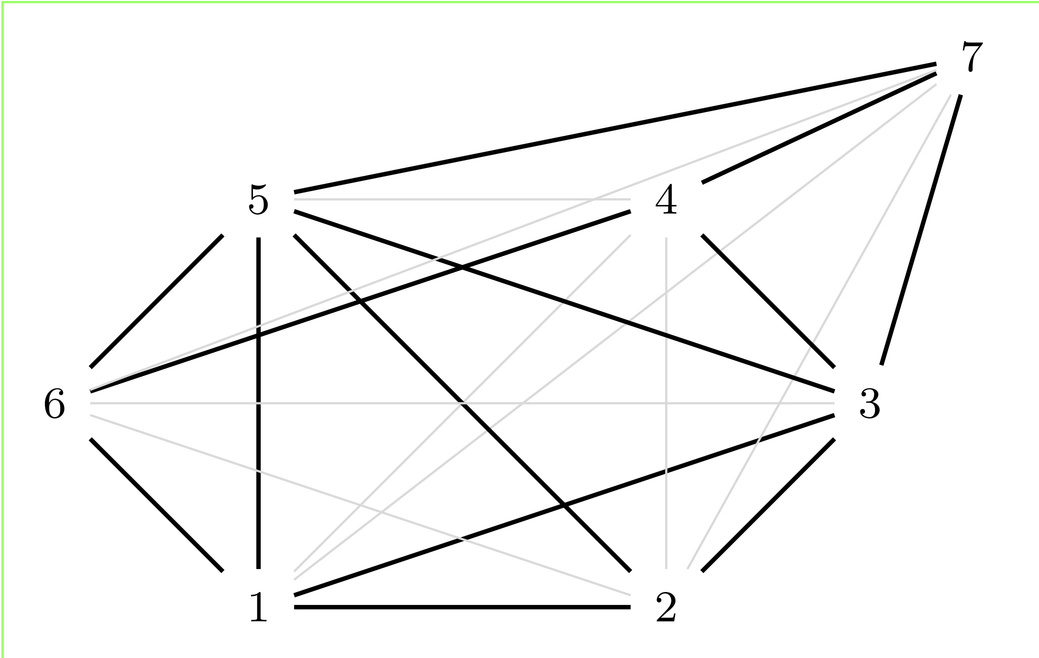



- Clique on vertex set $U$: denote by $K(U)$

# Complement graph

- Graph $G = (V, F)$ with $n$ vertices
- **Complement** of $G$: $\bar{G} = (V, E(K_n) \smallsetminus F)$

$G$

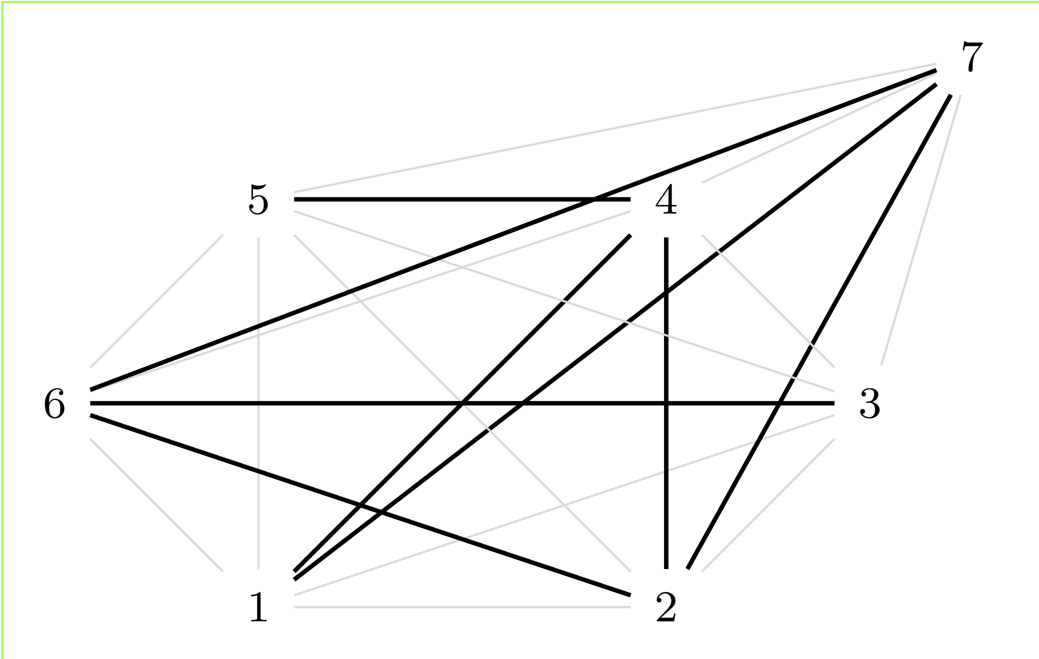



- $\overline{K_n}$: **empty graph** on $n$ vertices

# Complement graph

- Graph $G = (V, F)$ with $n$ vertices
- **Complement** of $G$: $\bar{G} = (V, E(K_n) \smallsetminus F)$

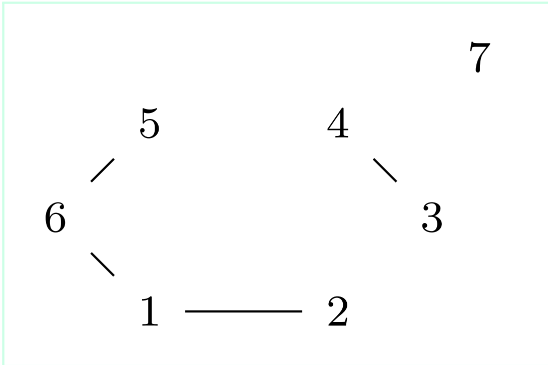
$\bar{G}$



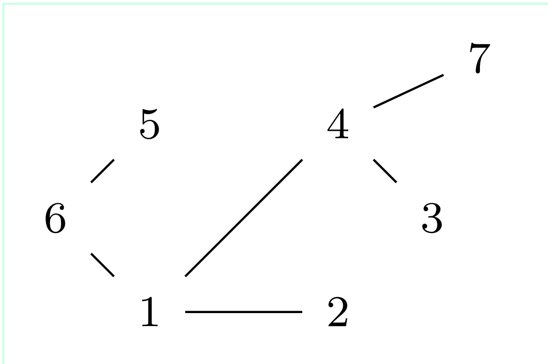

- $\overline{K_n}$: **empty graph** on $n$ vertices

# Forests and trees

- **Forest**: graph with no cycles



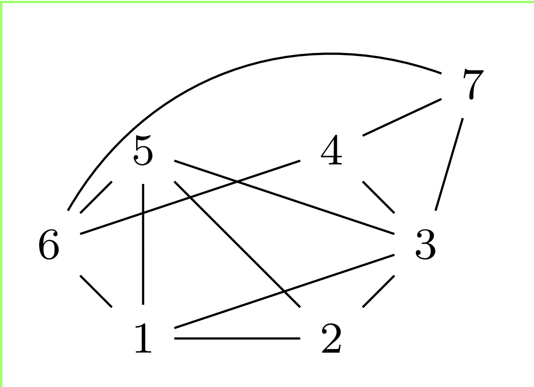

- **Tree**: connected forest



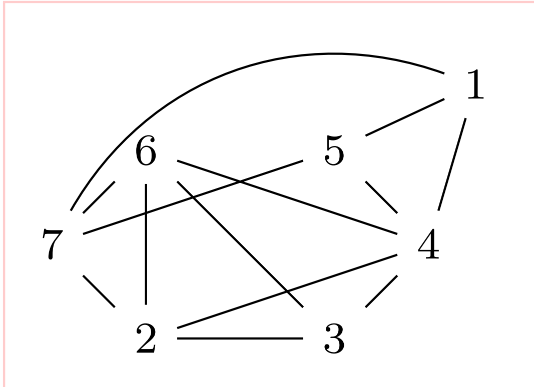

- Spanning tree subgraph of a graph $G$: **spanning tree** of $G$

# Graph isomorphism

- $|V| = n$, $S_n$ symmetric group of order $n$
  $\pi \in S_n$ permutes $V$, get new graph $H = \pi G = (\pi V, \pi E)$

$G =$ 


$\pi=(1,2,3,4,5,6,7) \longrightarrow$



$= H$

- $\exists \pi \in S_n \; (G = \pi H) \Rightarrow G, H$ **isomorphic**, $\pi$ **graph isomorphism**
- If $(\pi G = G)$, then $\pi$ is an **automorphism** of $G$

**Automorphism group** of $G$ is Aut$(G) = \langle (1,5), (4,7) \rangle \cong C_2 \times C_2$

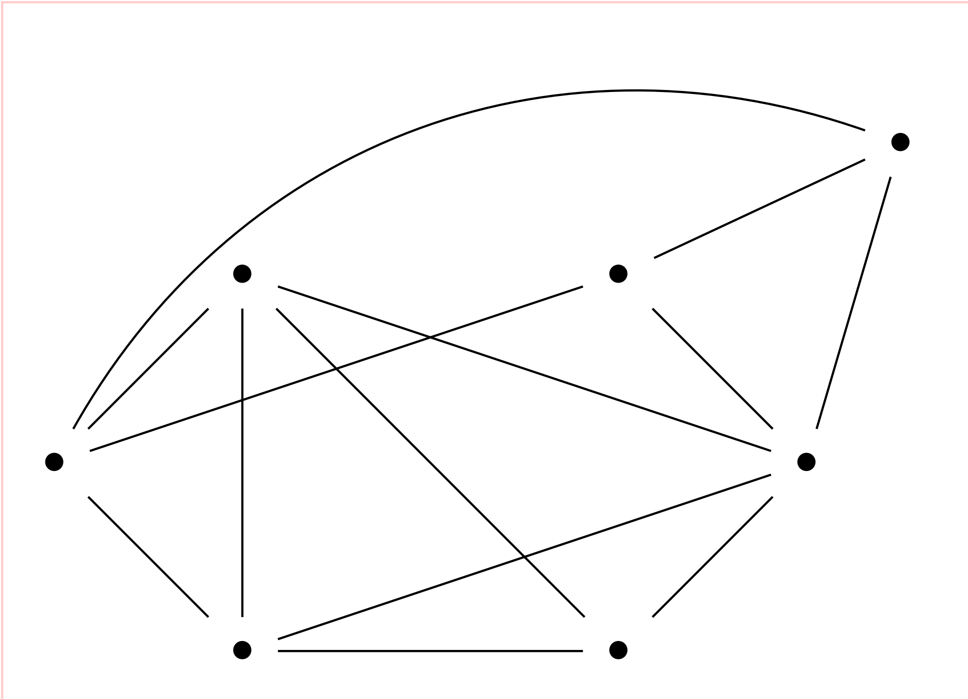
$N(1) = \{2,3,5,6\}$, $N(2) = \{1,3,5\}$
$N(3) = \{1,2,4,5,7\}$, $N(4) = \{3,6,7\}$
$N(5) = \{1,2,3,6\}$, $N(6) = \{1,4,5,7\}$
$N(7) = \{3,4,6\}$

$=$

$N(5) = \{2,3,1,6\}$, $N(2) = \{5,3,1\}$
$N(3) = \{5,2,7,1,4\}$, $N(7) = \{3,6,4\}$
$N(1) = \{5,2,3,6\}$, $N(6) = \{5,7,1,4\}$
$N(4) = \{3,7,6\}$

# Graphs modulo symmetry

- Symmetries act on vertex labels
- Ignore labels: *equivalence classes of graphs modulo symmetry*



- Unlabelled graphs

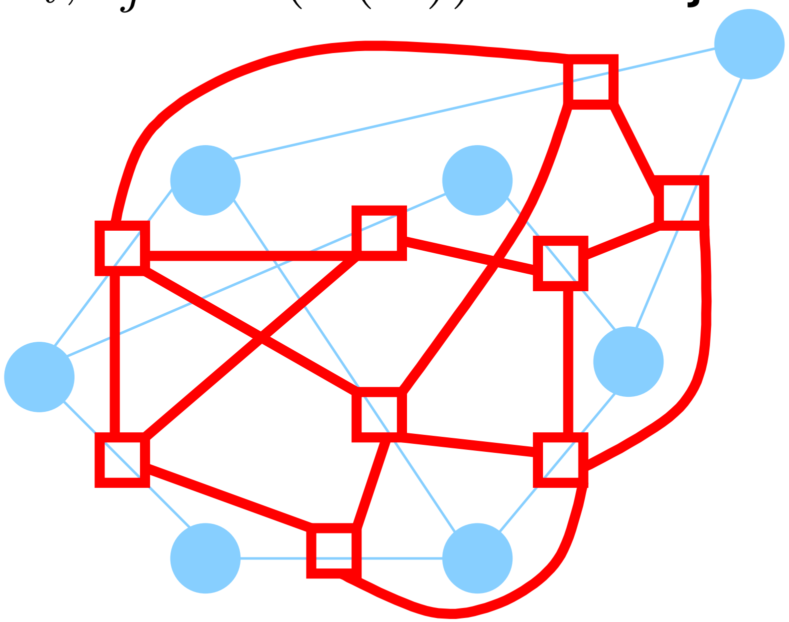
# Line graphs

- Graph $G = (V, E)$ with $E = \{e_1, \ldots, e_m\}$
- **Line graph** of $G$:

$$L(G) = (E, \{\{e_i, e_j\} \mid e_i \cap e_j \neq \varnothing\})$$

**Vertex of $L(G)$ $\Leftrightarrow$ edge of $G$**

- $e_i, e_j \in V(L(G))$ are adjacent $\Leftrightarrow$ $\exists v \in V$ s.t. $e_i, e_j \in \delta_G(v)$



**Property:** the degree $|N_{L(G)}(e)|$ of a vertex $e = \{u, v\}$ of $L(G)$ is $|N_G(u)| + |N_G(v)| - 2$.

**Property:** $L(G)$ can be constructed from $G$ in polynomial time (how?)

# Operations on graphs

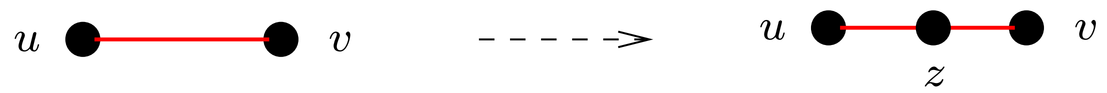
# Addition and removal

- Add a vertex $v$:
  update $V \leftarrow V \cup \{v\}$
- Add an edge $e = \{u, v\}$:
  add vertices $u, v$, update $E \leftarrow E \cup \{e\}$
- Remove an edge $e = \{u, v\}$:
  update $E \leftarrow E \smallsetminus \{e\}$
- Remove a vertex $v$:
  update $V \leftarrow V \smallsetminus \{v\}$ and $E \leftarrow E \smallsetminus \delta(v)$
- Operations on sets of vertices/edges:
  apply operation to each set element

# Subdivision and contraction

- Subdivide an edge $e = \{u, v\}$:

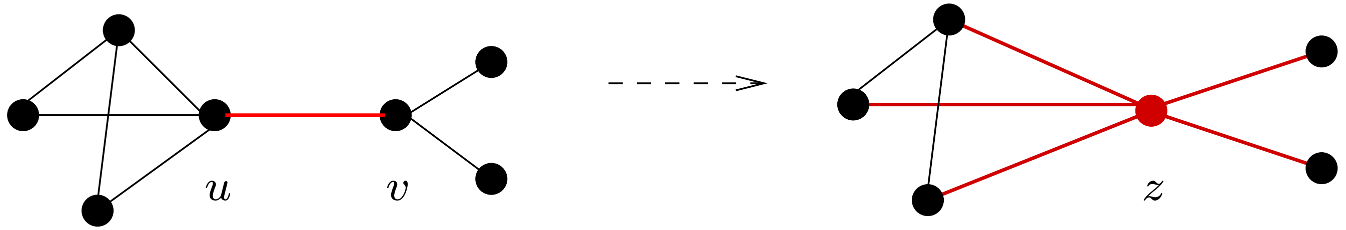
remove $e$, let $z \notin V$, add edges $\{u, z\}$ and $\{z, v\}$




- Contract an edge $e = \{u, v\}$:

`contract`$(G,e)$:

Let $N(e) = (N(u) \cup N(v)) \smallsetminus \{u, v\}$
Let $z$ be a vertex $\notin V$;
Add vertex $z$;
**for** $v \in N(e)$ **do**
    Add edge $\{v, z\}$;
**end for**
Remove edge $e$;



# Subgraph contraction

- Let $G = (V, E)$, $U \subseteq V$ and $H = G[U]$
- **Contraction** $G/U$: "$G$ modulo $H$"

```
contract(G,U):
1: Let z be a new vertex ∉ V
2: Add vertex z
3: for {u, v} ∈ δ(H) (assume WLOG u ∈ U, v ∈ V ∖ U) do
4:     Add edge {v, z}
5:     Remove edge {u, v}
6: end for
7: Remove G[U]
8: return G;
```

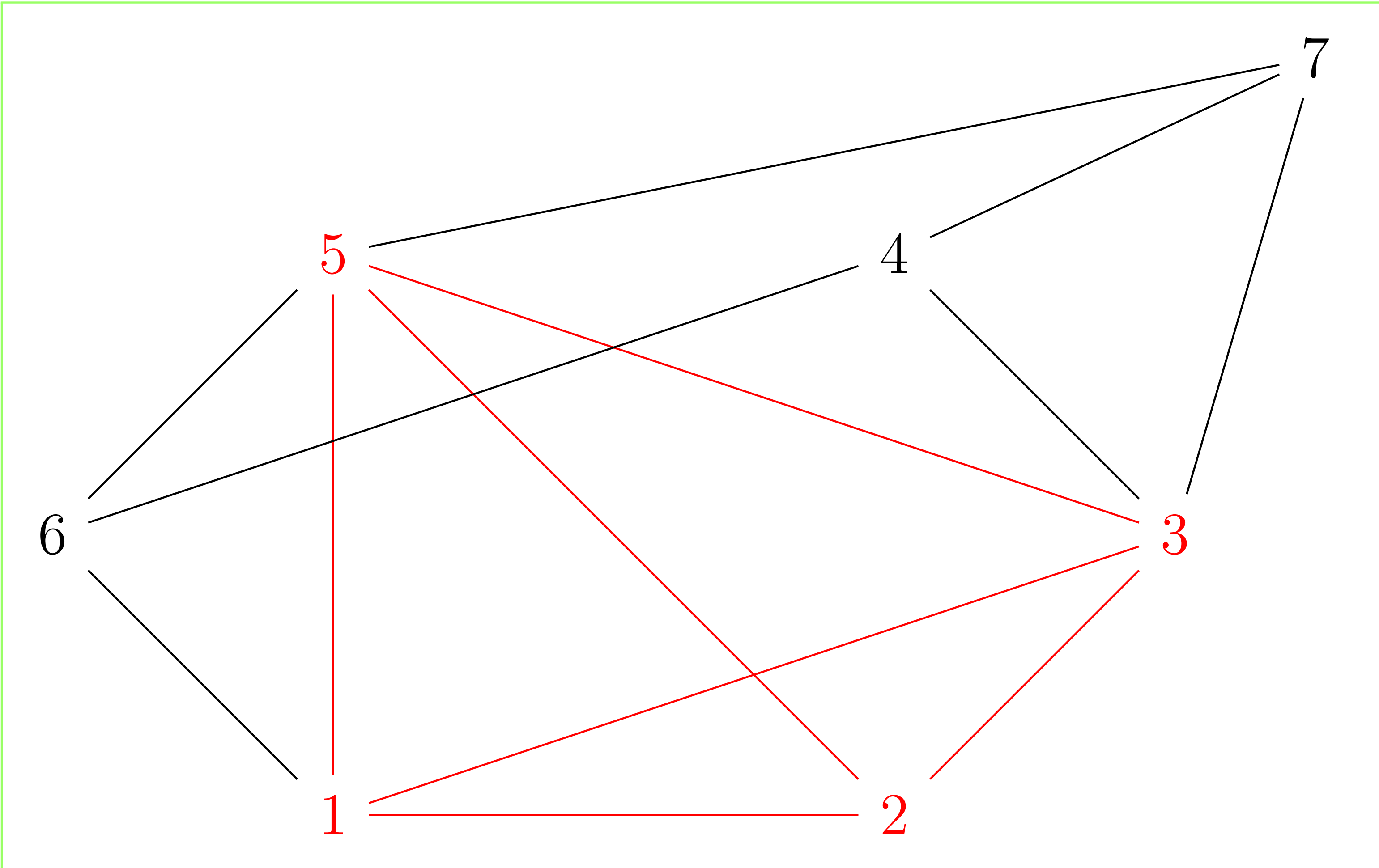
- At termination, subgraph $H$ replaced by single vertex $z$
- $G/U$ is formally defined to be $\texttt{contract}(G, U)$

Thm.

Subgraph contraction is equivalent to a sequence of edge contractions

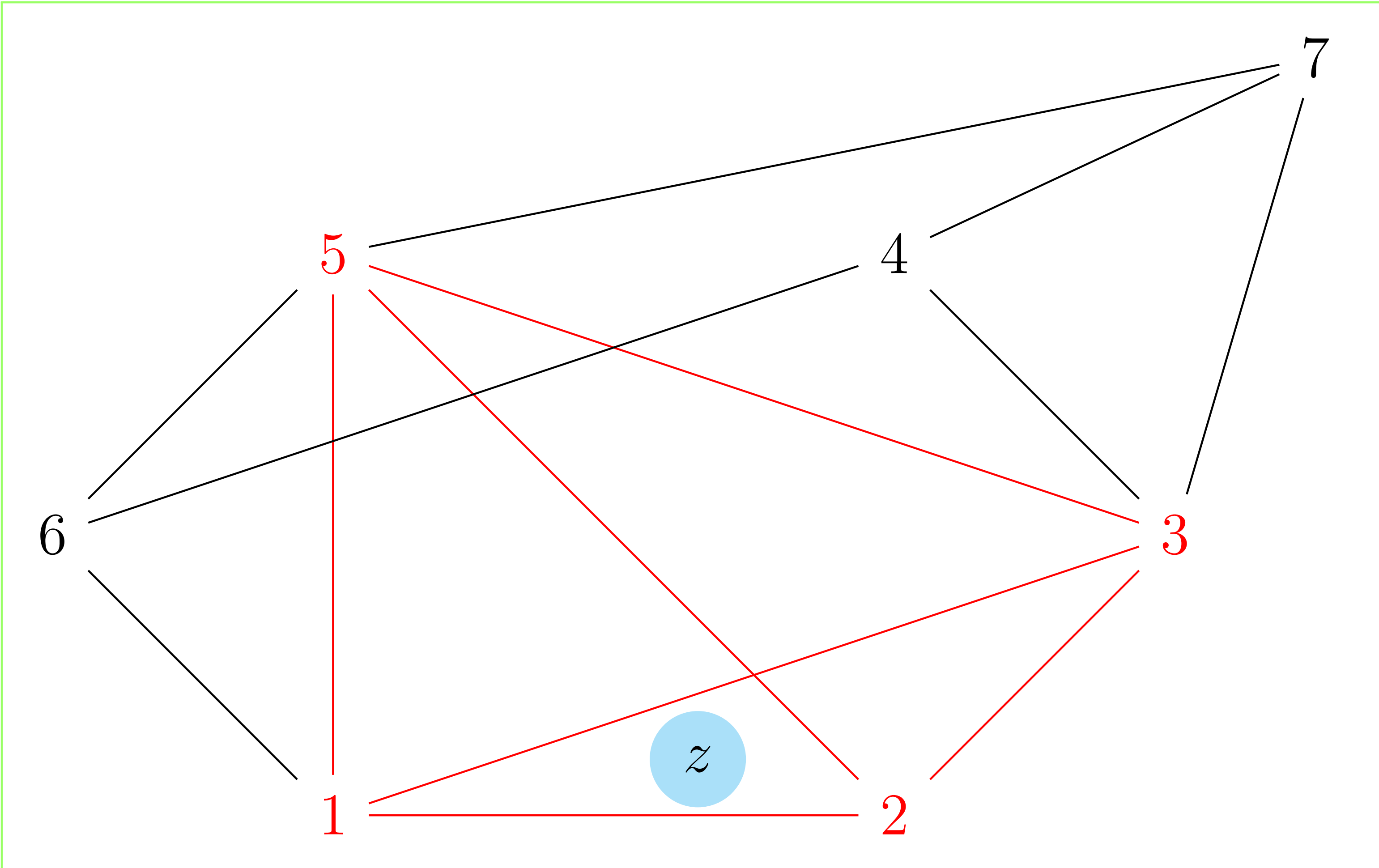
# Subgraph contraction algorithm

$U = \{1, 2, 3, 5\}$, $G[U]$ **in red**



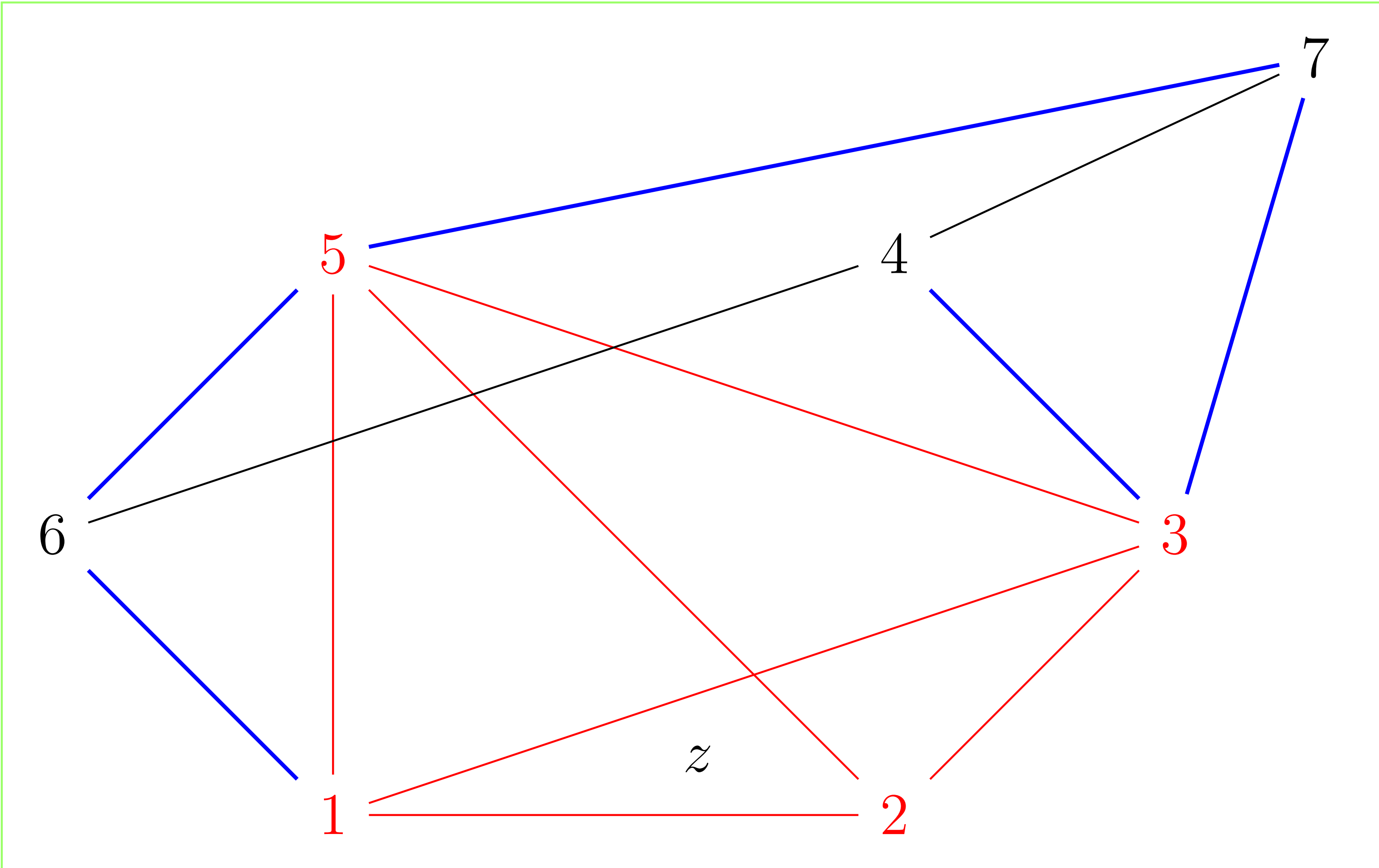
# Subgraph contraction algorithm

Add $z$



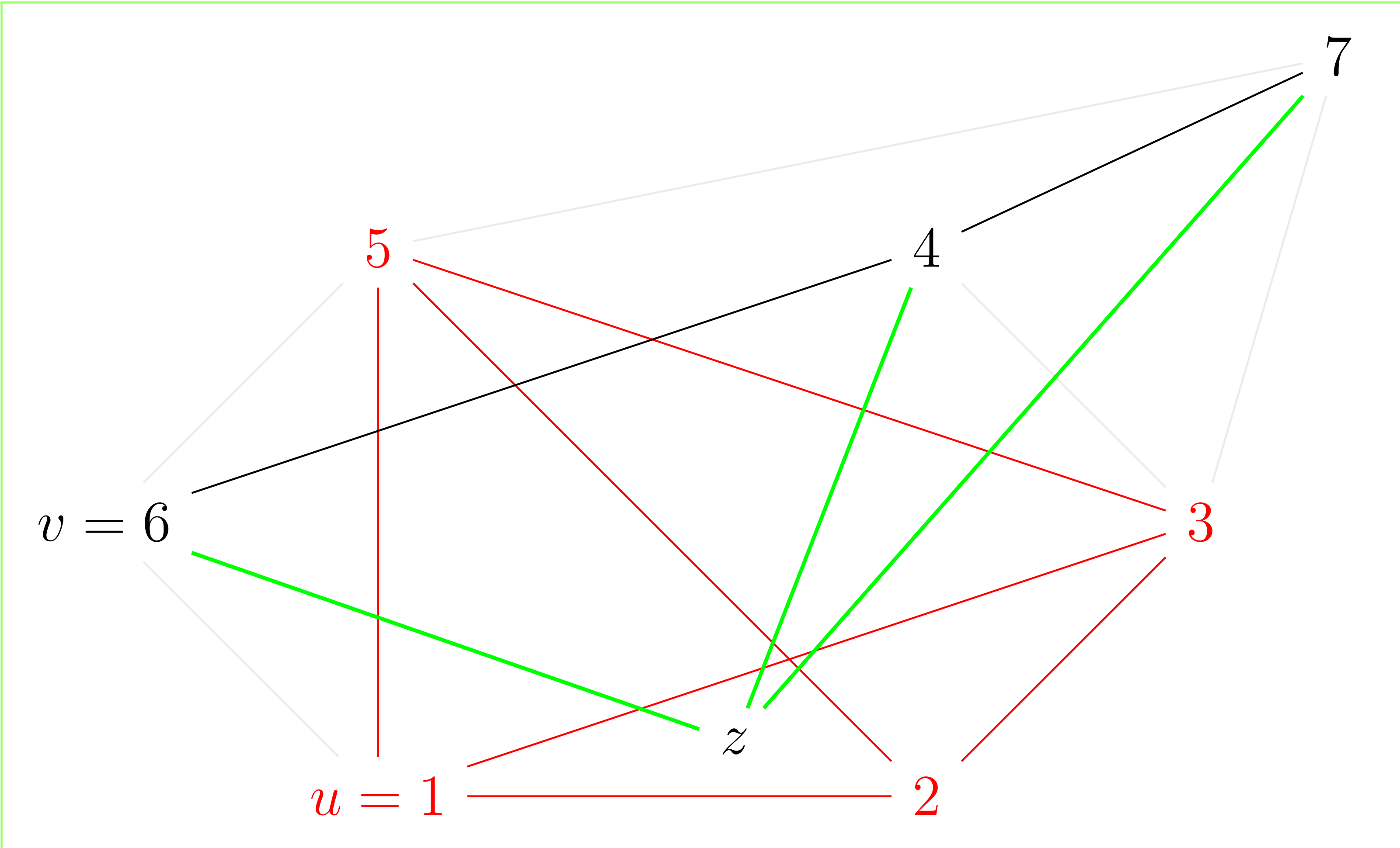
# Subgraph contraction algorithm

$\delta(G[U])$ **in blue** *(edges with just one endpoint in $U$)*



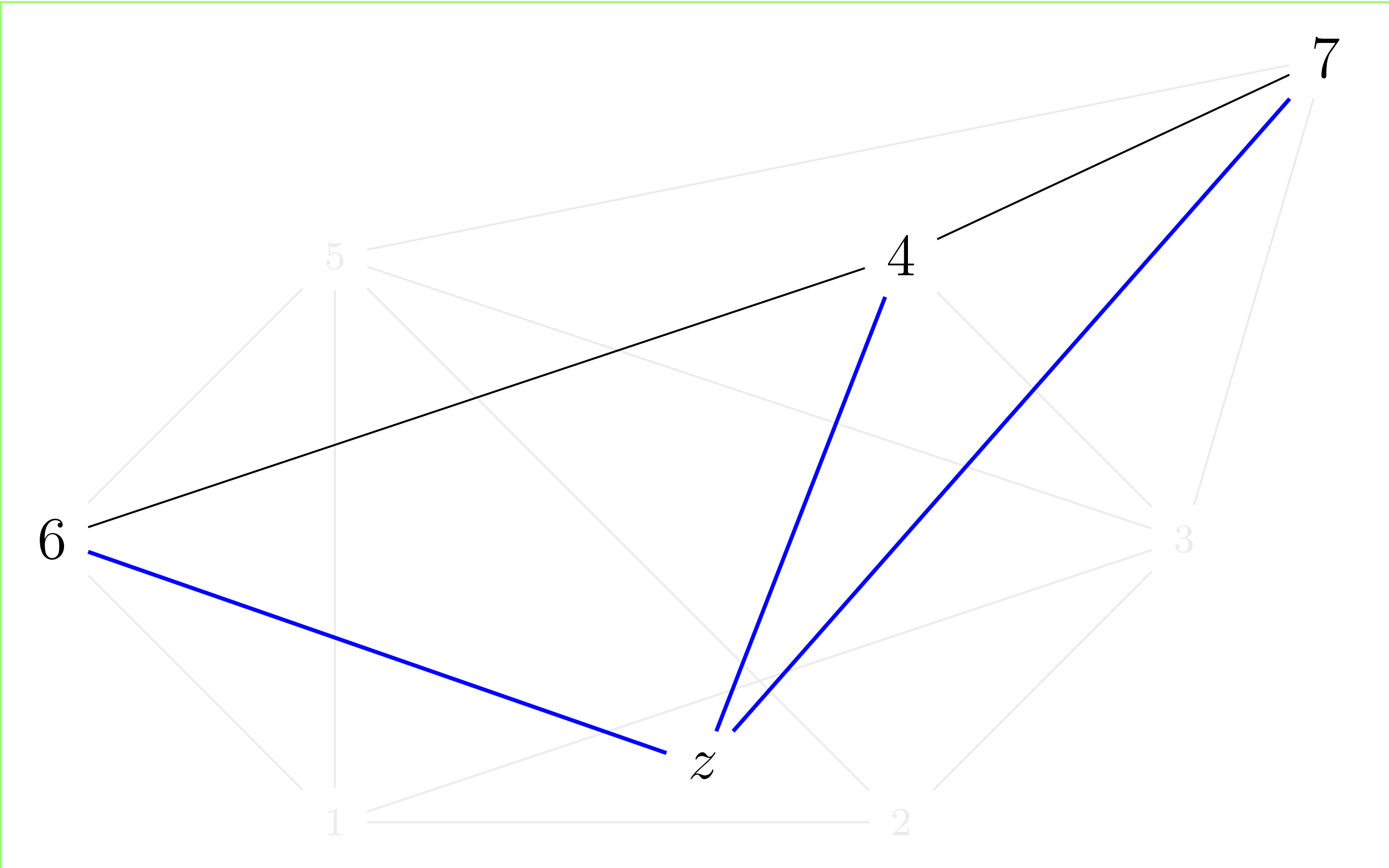
# Subgraph contraction algorithm

Add $\{v, z\}$ and remove $\{u, v\}$



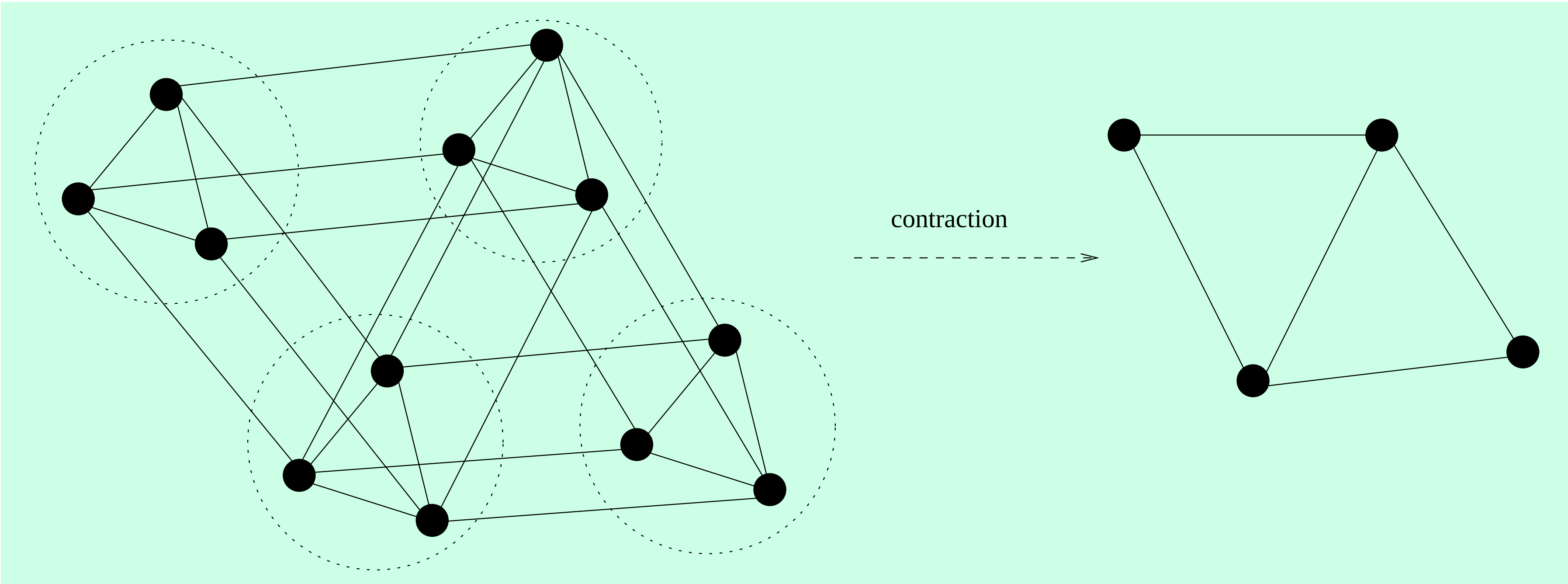
# Subgraph contraction algorithm

Remove $G[U]$ (end)



# Minors

- $F$ **minor** of $G$: $F$ isomorphic to a contracted $G$
- Useful to underline “essential structure”




*Contract some triangles*

# Combinatorial problems on graphs

# The subgraph problem

- **Decision problem**: YES/NO question parametrized over symbols representing the **instance** (i.e. the input)

# The subgraph problem

- **Decision problem**: YES/NO question parametrized over symbols representing the **instance** (i.e. the input)
- $\mathbb{G} =$all graphs, $P(G)$ valid sentence about some $G \in \mathbb{G}$

# The subgraph problem

- **Decision problem**: YES/NO question parametrized over symbols representing the **instance** (i.e. the input)
- $\mathbb{G}$ =all graphs, $P(G)$ valid sentence about some $G \in \mathbb{G}$
- A class of decision problems in graph theory:

  SUBGRAPH PROBLEM SCHEMA (SPS$_P$). Given a graph $G$, does it have a subgraph $H$ with property $P$?

# The subgraph problem

- **Decision problem**: YES/NO question parametrized over symbols representing the **instance** (i.e. the input)
- $\mathbb{G}$ =all graphs, $P(G)$ valid sentence about some $G \in \mathbb{G}$
- A class of decision problems in graph theory:

  SUBGRAPH PROBLEM SCHEMA (SPS$_P$). Given a graph $G$, does it have a subgraph $H$ with property $P$?
- In SPS$_P$, get a decision problem for each $P$

# The subgraph problem

- **Decision problem**: YES/NO question parametrized over symbols representing the **instance** (i.e. the input)
- $\mathbb{G}$ =all graphs, $P(G)$ valid sentence about some $G \in \mathbb{G}$
- A class of decision problems in graph theory:

  SUBGRAPH PROBLEM SCHEMA (SPS$_P$). Given a graph $G$, does it have a subgraph $H$ with property $P$?
- In SPS$_P$, get a decision problem for each $P$
- Decision problem $=$ set of all its instances

# The subgraph problem

- **Decision problem**: YES/NO question parametrized over symbols representing the **instance** (i.e. the input)
- $\mathbb{G}$ =all graphs, $P(G)$ valid sentence about some $G \in \mathbb{G}$
- A class of decision problems in graph theory:

  SUBGRAPH PROBLEM SCHEMA (SPS$_P$). Given a graph $G$, does it have a subgraph $H$ with property $P$?
- In SPS$_P$, get a decision problem for each $P$
- Decision problem $=$ set of all its instances
- Require solution YES or NO with **certificate** (proof that certifies the answer)

# The subgraph problem

- **Decision problem**: YES/NO question parametrized over symbols representing the **instance** (i.e. the input)
- $\mathbb{G}$ =all graphs, $P(G)$ valid sentence about some $G \in \mathbb{G}$
- A class of decision problems in graph theory:

  SUBGRAPH PROBLEM SCHEMA (SPS$_P$). Given a graph $G$, does it have a subgraph $H$ with property $P$?
- In SPS$_P$, get a decision problem for each $P$
- Decision problem $=$ set of all its instances
- Require solution YES or NO with **certificate** (proof that certifies the answer)
- E.g. if $P(H) \equiv (H \text{ is a cycle})$ the certificate is the cycle

# Complexity classes

- **P**: decision problems whose YES/NO certificates can be found in polynomial time (of the instance size)
- **E.g. Given $p, q, n \in \mathbb{Z}$, is $pq = n$?**
- **NP**: class of decision problems whose YES certificates can be verified in polynomial time
- E.g. Given graphs $G, H$, are they isomorphic?

# Algorithms, problems, classes

Consider worst-case complexity

- **Complexity of an algorithm**: asymptotic performance on $\infty$ly many instances parametrized by $n$, as $n \to \infty$
- Problem: $\infty$ly many instances
- Complexity of a problem: best algorithm for all instances in problem
- Problem class: all problems with similar complexity
- Complexity classes: classification of problems into "easy" and "hard"

# Graph optimization problems

- Given a decision problem, $\exists$ a corresponding **optimization problem**
- Consider scalar function $\mu : \mathbb{G} \to \mathbb{R}$
- E.g. $\mu$: number of vertices/edges
- Class of optimization problems on graphs:

  SUBGRAPH OPTIMIZATION PROBLEM SCHEMA (SOPS$_{P,\mu}$). Given a graph $G$, does it have a subgraph $H$ with property $P$ and min./max. $\mu$ value?

- Given a property $P$ and a function $\mu$, the set of instances of SOPS$_{P,\mu}$ is an **optimization problem**

# Easy problems

- **P** $=$ decision or optimization problems that can be solved in polynomial time $=$ "easy problems "
- MINIMUM SPANNING TREE (MST)

  To be seen in Lecture 4
- SHORTEST PATH PROBLEM (SPP) from a vertex $v$ to all other vertices

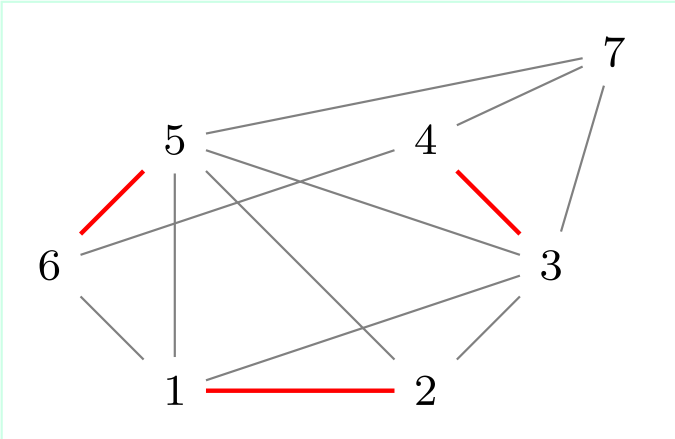
  To be seen in Lecture 9
- MAXIMUM MATCHING problem (MATCHING)

  Discussed in INF550

**Matching**: subgraph given by set of mutually non-adjacent edges

*A maximum matching* $M$*,*

$\mu(M) = |E(M)|$

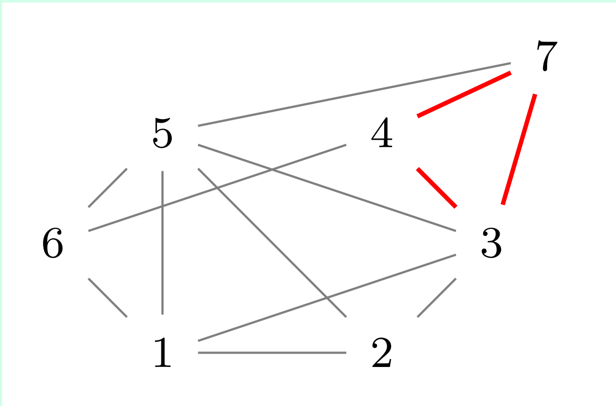


# Hard(er) problems

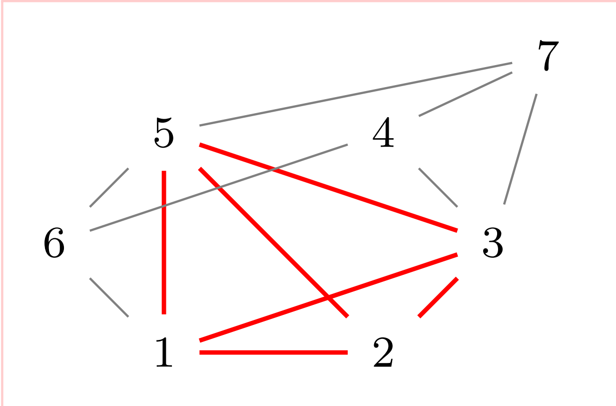
# Maximum clique

CLIQUE PROBLEM (CLIQUE). Given a graph $G$, what is the largest $n$ such that $G$ has $K_n$ as a subgraph?

- In CLIQUE, $P(H) \equiv [H = K(V(H))]$ and $\mu(H) = |V(H)|$




A clique in $G$




The largest clique in $G$

- Applications to social networks and bioinformatics

# Clique and NP-completeness

- Decision version of CLIQUE:

  $k$-CLIQUE PROBLEM ($k$-CLIQUE). Given a graph $G$ and an integer $k > 0$, does $G$ have $K_k$ as a subgraph?

- Consider the following result (which we won't prove) Thm.

  [Karp 1972] If CLIQUE $\in$ **P** then **P** $=$ **NP**

- Any decision problem for which such a result holds is called **NP**-complete
- It is not known whether **NP**-complete problems can be solved in polynomial time; the current guess is NO

# Solving NP-complete problems

- Decision problem $P$ is **NP**-complete $\equiv$ "$P$ is hard"

**Intuition**: if $P$ easy, every problem in **NP** is easy $\equiv$ all computer scientists to date are idiots — hopefully unlikely

# Solving NP-complete problems

- Decision problem $P$ is **NP**-complete $\equiv$ "$P$ is hard"

  **Intuition**: if $P$ easy, every problem in **NP** is easy $\equiv$ all computer scientists to date are idiots — hopefully unlikely

- Solving an **NP**-complete decision problem:
  - **exact** but <u>exponential-time</u> algorithms
  - **heuristic** algorithms

    **provide YES certificates, may not terminate on NO**

# Solving NP-complete problems

- Decision problem $P$ is **NP**-complete $\equiv$ "$P$ is hard"

  **Intuition**: if $P$ easy, every problem in **NP** is easy $\equiv$ all computer scientists to date are idiots — hopefully unlikely

- Solving an **NP**-complete decision problem:
  - **exact** but <u>exponential-time</u> algorithms
  - **heuristic** algorithms

    **provide YES certificates, may not terminate on NO**

- Optimization problem $P$ s.t. $P \in \mathbf{P} \to \mathbf{P} = \mathbf{NP}$:

  $P$ is **NP**-hard

# Solving NP-complete problems

- Decision problem $P$ is **NP**-complete $\equiv$ "$P$ is hard"

  **Intuition**: if $P$ easy, every problem in **NP** is easy $\equiv$ all computer scientists to date are idiots — hopefully unlikely

- Solving an **NP**-complete decision problem:
  - **exact** but exponential-time algorithms
  - **heuristic** algorithms

    **provide YES certificates, may not terminate on NO**

- Optimization problem $P$ s.t. $P \in$ **P** $\rightarrow$ **P** $=$ **NP**:

  $P$ is **NP**-hard

- $P$ **NP**-complete $\Leftrightarrow$ $P$ **NP**-hard $\wedge P \in$ **NP**

# Solving NP-complete problems

- Decision problem $P$ is **NP**-complete $\equiv$ "$P$ is hard"

  **Intuition**: if $P$ easy, every problem in **NP** is easy $\equiv$ all computer scientists to date are idiots — hopefully unlikely

- Solving an **NP**-complete decision problem:
  - **exact** but exponential-time algorithms
  - **heuristic** algorithms

    **provide YES certificates, may not terminate on NO**

- Optimization problem $P$ s.t. $P \in$ **P** $\rightarrow$ **P** $=$ **NP**:

  $P$ is **NP**-hard

- $P$ **NP**-complete $\Leftrightarrow$ $P$ **NP**-hard $\wedge P \in$ **NP**

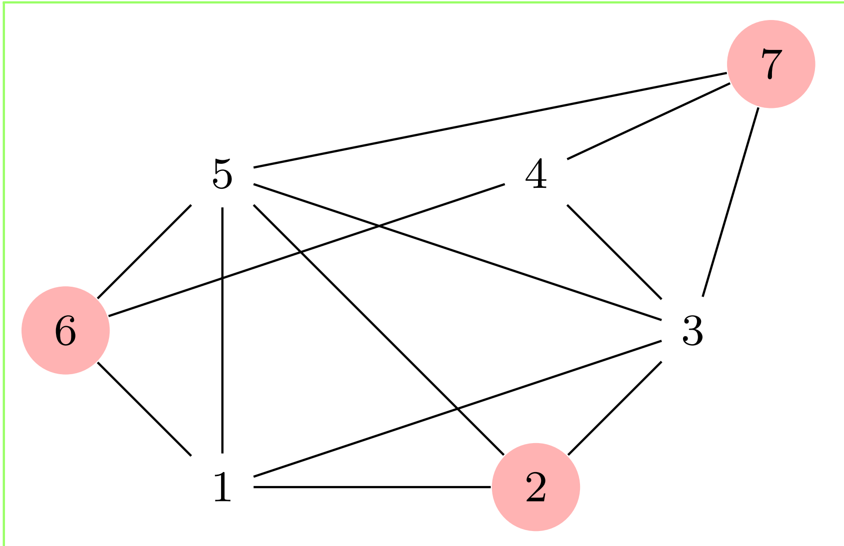
- $f$**-Approximation** algorithm: heuristic s.t.$\mu$-value of YES certificate no worse than $f(|G|)$ times optimal $\mu$ value

# Stables

- **Stable** (or **independent set**) in $G = (V, E)$: subset $U \subseteq V$ s.t. $\forall u, v \in U \ (\{u, v\} \notin E)$
  Thm.

  $U$ is a stable in $G$ if and only if $\overline{G[U]}$ is a clique

*a stable in $G$*




- Decision problem: $k$-STABLE

  Given $G$ and $k \in \mathbb{N}$, is there a stable $U \subseteq V(G)$ of size $k$?

- Optimization problem: STABLE

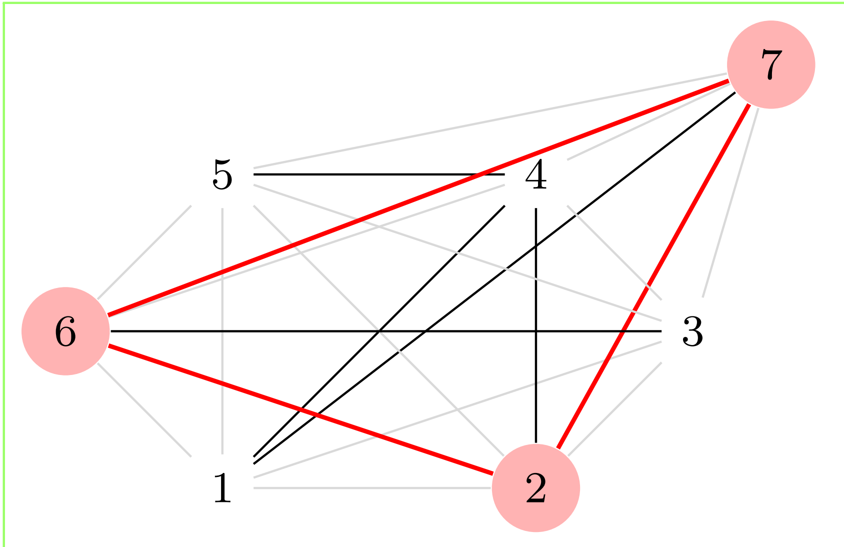
  Given $G$, find the stable of $G$ of maximum size

# Stables

- **Stable** (or **independent set**) in $G = (V, E)$: subset $U \subseteq V$ s.t. $\forall u, v \in U \ (\{u, v\} \notin E)$
  Thm.

  $U$ is a stable in $G$ if and only if $\overline{G[U]}$ is a clique

  $\overline{G[U]}$ *is a clique*




- Decision problem: $k$-STABLE

  Given $G$ and $k \in \mathbb{N}$, is there a stable $U \subseteq V(G)$ of size $k$?

- Optimization problem: STABLE

  Given $G$, find the stable of $G$ of maximum size

# NP-completeness of $k$-STABLE

**Thm.**

$k$-STABLE is **NP**-complete

**Proof**

Consider an instance $(G, k)$ of $k$-CLIQUE

The complement graph $\bar{G}$ can be obtained in polynomial time $(*)$

It is easy to show that $\bar{\bar{G}} = G \quad (**)$

By $(**)$ and previous thm.,

$(G, k)$ is a YES instance of $k$-CLIQUE iff $(\bar{G}, k)$ is a YES instance of $k$-STABLE

By $(*)$, if $k$-STABLE $\in$ **P** then $k$-CLIQUE $\in$ **P** (*transform to $k$-STABLE, solve, transform back*)

By **NP**-completeness of $k$-CLIQUE, $k$-STABLE $\in$ **P** implies **P** $=$ **NP**

Hence $k$-STABLE is **NP**-complete

- How to show that a problem $\mathcal{P}$ is **NP**-complete:
  - Take another **NP**-complete problem $\mathcal{Q}$ “similar” to $\mathcal{P}$
  - Reduce (in polytime) an instance of $\mathcal{Q}$ to an instance of $\mathcal{P}$
  - Show reduction preserves the YES/NO property

# Stable heuristic

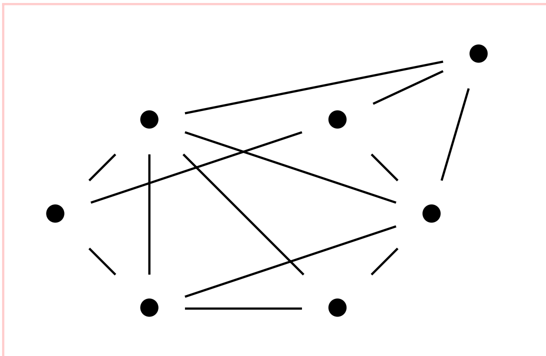
- The following **greedy** method will find a *maximal* stable

1: $U = \varnothing$;
2: order $V$ by increasing values of $|N(v)|$;
3: **while** $V \neq \varnothing$ **do**
4: $\quad v = \min V$;
5: $\quad U \leftarrow U \cup \{v\}$;
6: $\quad V \leftarrow V \smallsetminus (\{v\} \cup N(v))$
7: **end while**

- Worst-case: $O(n)$ (given by an empty graph)

*degree sequence*

$(3, 3, 3, 3, 4, 5, 5)$



- STABLE heuristic $\Rightarrow$ CLIQUE heuristic

# Stable heuristic

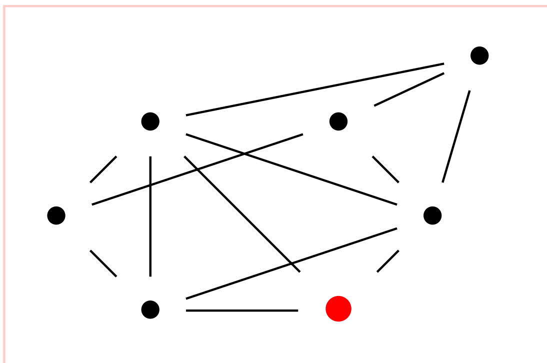
- The following **greedy** method will find a *maximal* stable

```
U = ∅;
order V by increasing values of |N(v)|;
while V ≠ ∅ do
   v = min V;
   U ← U ∪ {v};
   V ← V ∖ ({v} ∪ N(v))
end while
```

- Worst-case: $O(n)$ (given by an empty graph)

*select* $\min V$

*put it in* $U$



- STABLE heuristic $\Rightarrow$ CLIQUE heuristic

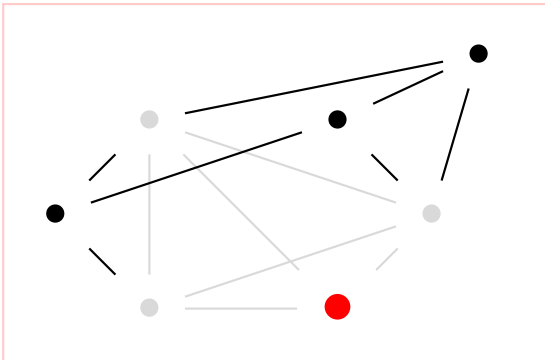
# Stable heuristic

- The following **greedy** method will find a *maximal* stable

```
U = ∅;
order V by increasing values of |N(v)|;
while V ≠ ∅ do
    v = min V;
    U ← U ∪ {v};
    V ← V ∖ ({v} ∪ N(v))
end while
```

- Worst-case: $O(n)$ (given by an empty graph)

*remove $v$ and its star from $V$*



- Stable heuristic $\Rightarrow$ Clique heuristic

# Stable heuristic

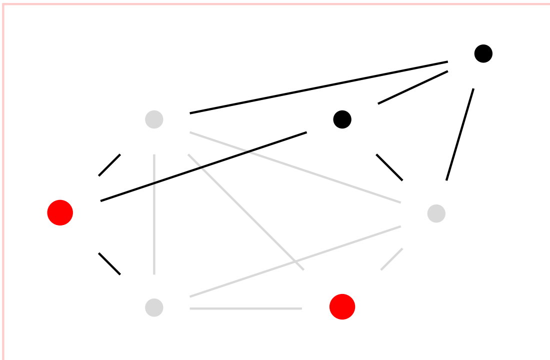
- The following **greedy** method will find a *maximal* stable

$U = \varnothing$;
order $V$ by increasing values of $|N(v)|$;
**while** $V \neq \varnothing$ **do**
  $v = \min V$;
  $U \leftarrow U \cup \{v\}$;
  $V \leftarrow V \smallsetminus (\{v\} \cup N(v))$
**end while**

- Worst-case: $O(n)$ (given by an empty graph)

*select* $\min V$

*put it in* $U$



- STABLE heuristic $\Rightarrow$ CLIQUE heuristic

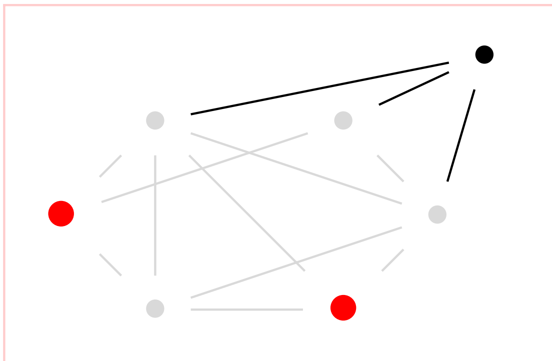
# Stable heuristic

- The following **greedy** method will find a *maximal* stable

1: $U = \varnothing$;
2: order $V$ by increasing values of $|N(v)|$;
3: **while** $V \neq \varnothing$ **do**
4: $\quad v = \min V$;
5: $\quad U \leftarrow U \cup \{v\}$;
6: $\quad V \leftarrow V \smallsetminus (\{v\} \cup N(v))$
7: **end while**

- Worst-case: $O(n)$ (given by an empty graph)

*remove $v$ and its star from $V$*



- STABLE heuristic $\Rightarrow$ CLIQUE heuristic

# Stable heuristic

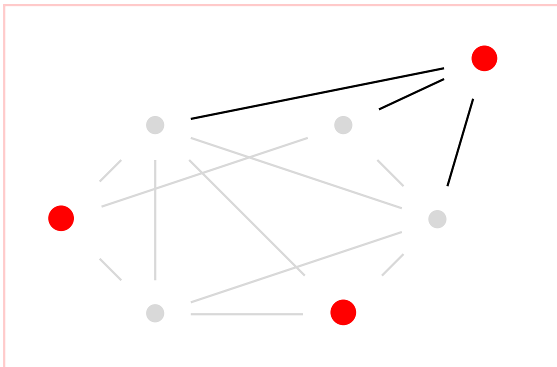
- The following **greedy** method will find a *maximal* stable

```
U = ∅;
order V by increasing values of |N(v)|;
while V ≠ ∅ do
    v = min V;
    U ← U ∪ {v};
    V ← V ∖ ({v} ∪ N(v))
end while
```

- Worst-case: $O(n)$ (given by an empty graph)

*select* $\min V$

*put it in* $U$



- Stable heuristic $\Rightarrow$ Clique heuristic

# Stable heuristic

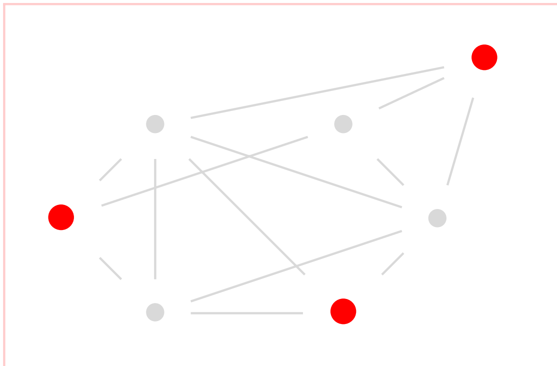
- The following **greedy** method will find a *maximal* stable

```
U = ∅;
order V by increasing values of |N(v)|;
while V ≠ ∅ do
   v = min V;
   U ← U ∪ {v};
   V ← V ∖ ({v} ∪ N(v))
end while
```

- Worst-case: $O(n)$ (given by an empty graph)

*remove $v$ and its star from $V$*

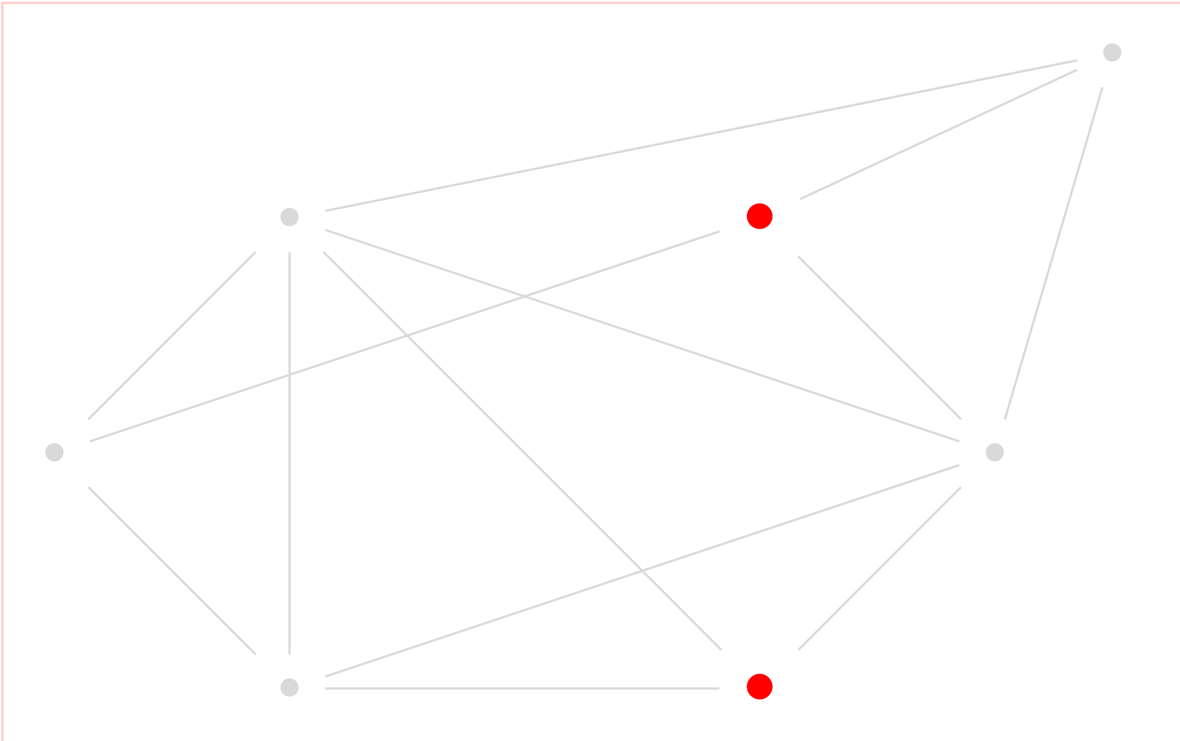
*stop: maximal stable*



- Stable heuristic $\Rightarrow$ Clique heuristic

# Heuristic fails

- Heuristic fails to find a maximum stable
- When choosing second element of $U$, take

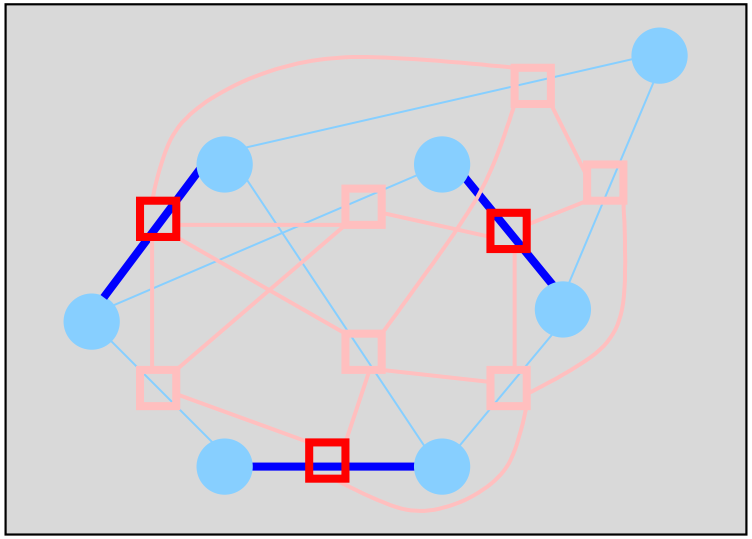


- Algorithm stops with a stable of cardinality 2

# Polynomial cases

- $P$ an **NP**-complete decision problem
- **Polynomial case**: $C \subseteq P$ s.t. $C \in$ **P**
- E.g. $\mathcal{L} = \{H \in \mathbb{G} \mid \exists G \in \mathbb{G}\ (H = L(G))\}$
- $\mathcal{L} =$ graphs that are line graphs of another graph

Proof



Thm.

A maximum matching in $G$ is a stable in $L(G)$

- MATCHING $\in$ **P** and finding $L(G)$ is polytime $\Rightarrow$ STABLE$_{\mathcal{L}}$ $\in$ **P**

# Vertex colouring

- Decision problem

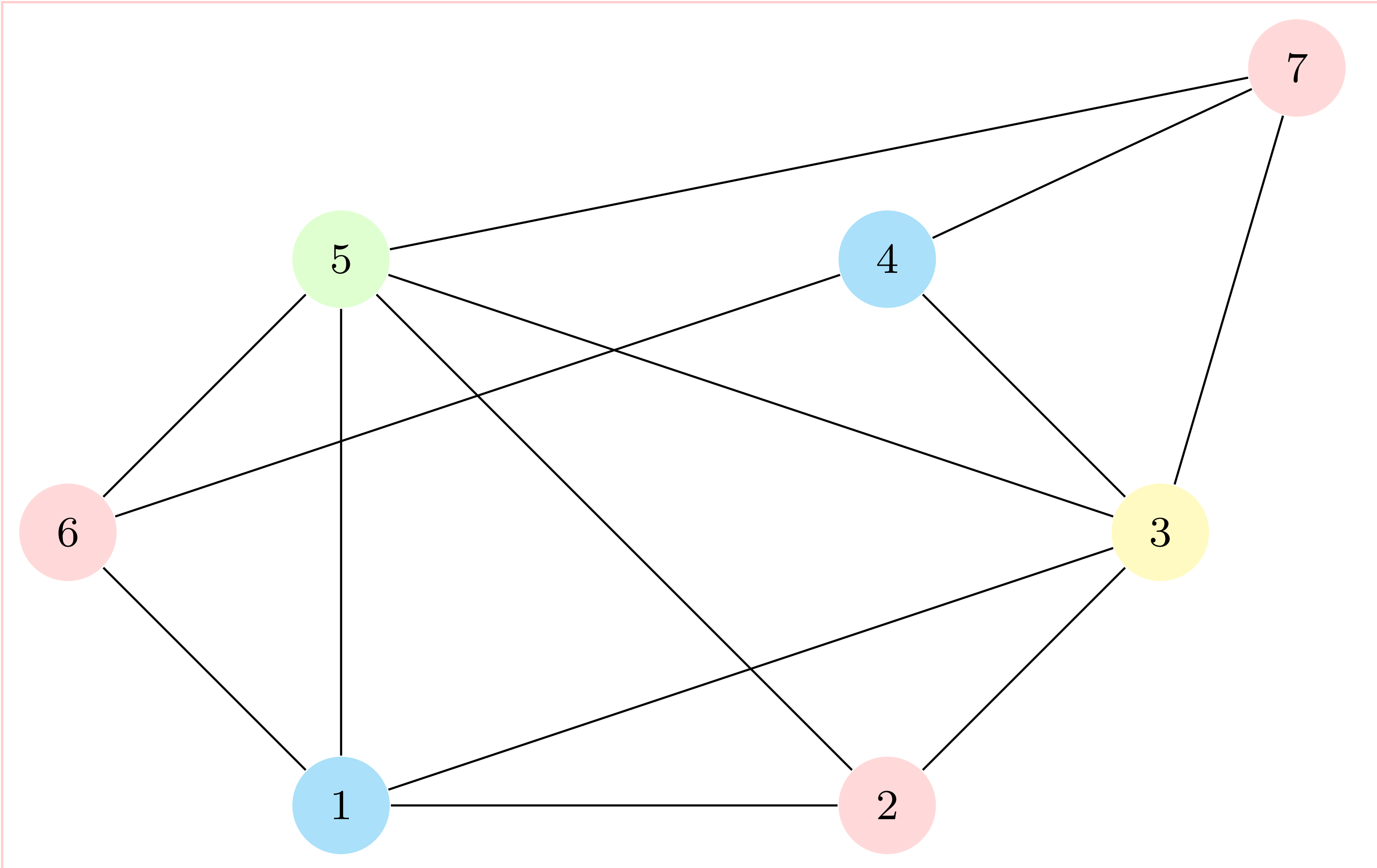
  VERTEX $k$-COLOURING PROBLEM ($k$-VCP). Given a graph $G = (V, E)$ and an integer $k > 0$, find a function $c : V \to \{1, \ldots, k\}$ such that $\forall \{u, v\} \in E \ (c(u) \neq c(v))$

- Optimization problem

  VERTEX COLOURING PROBLEM (VCP). Given a graph $G = (V, E)$, find the minimum $k \in \mathbb{N}$ such that there is a function $c : V \to \{1, \ldots, k\}$ with $\forall \{u, v\} \in E \ (c(u) \neq c(v))$

- Applications to scheduling and wireless networks
- In general, allocate resources to minimum number of classes without conflicts

# Vertex colouring example



# Vertex colouring heuristic

Thm.

Each color set $C_k = \{v \in V \mid c(v) = k\}$ is a stable

- Use stable set heuristic as a sub-step

```
k = 1;
U = V;
while U ≠ ∅ do
  C_k = maximalStable(G[U]);
  U ← U ∖ C_k;
  k ← k + 1;
end while
```

- Worst-case: $O(n)$ (given by an empty or complete graph)

# Model-and-solve

# Mathematical programming

- Take e.g. the STABLE problem
- Input (also called **parameters**):
  - set of vertices $V$
  - set of edges $E$
- Output: $x : V \to \{0,1\}$

$$\forall v \in V \quad x(v) = \begin{cases} 1 & \text{if } v \in \text{maximum stable} \\ 0 & \text{otherwise} \end{cases}$$

- We also write $x_v = x(v)$
- We'd like $x = (x_v \mid v \in V) \in \{0,1\}^{|V|}$ to be the **characteristic vector** of the maximum stable $S^*$
- $x_1, \ldots, x_{|V|}$ are also called **decision variables**

# Objective function

- If we take $x = (0,0,0,0,0,0,0)$, $S^* = \varnothing$ and $|S^*| = 0$ (minimum possible value)
- If we take $x = (1,1,1,1,1,1,1) = \mathbf{1}$, $|S^*| = |V| = 7$ has the maximum possible value
- Characteristic vector $x$ should satisfy the **objective function**

$$\max_x \sum_{v \in V} x_v$$

# Constraints

- Consider the solution $x = \mathbf{1}$
- $x$ certainly maximizes the objective
- ...but $S^* = V$ is not a stable!

  $x = 1$ is an **infeasible solution**

- The **feasible set** is the set of all vectors in $\{0,1\}^{|V|}$ which encode stable sets
- Defining property of a stable:

  Two adjacent vertices cannot both belong to the stable

- In other words,

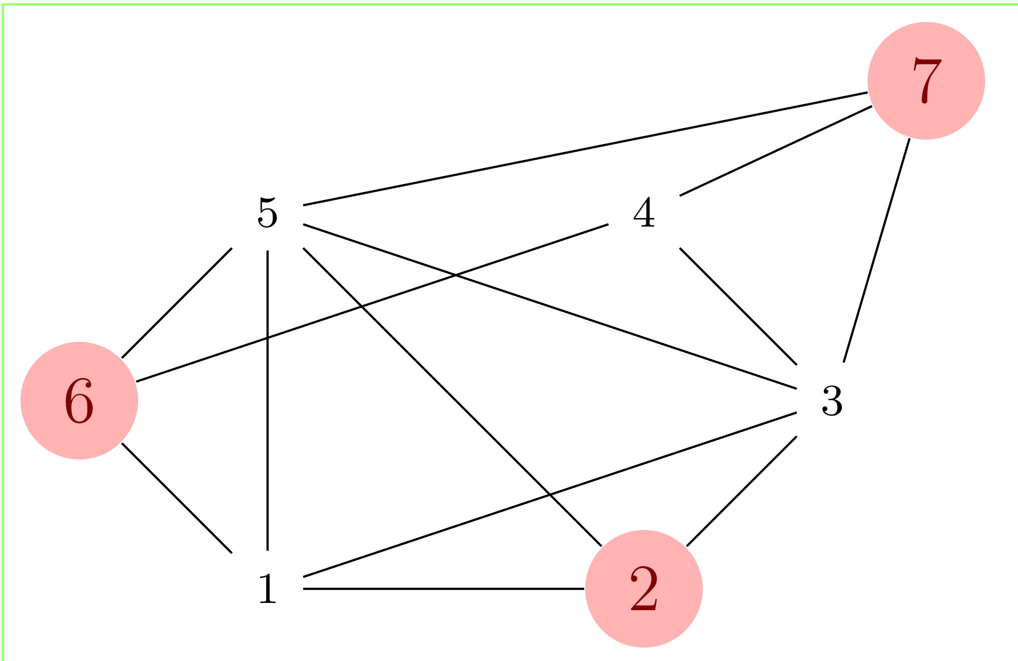
  *choose at most one vertex adjacent to each edge*

- Written formally,

$$\forall \{u,v\} \in E \quad x_u + x_v \leq 1$$

# Verify the constraints

- $x = (0,1,0,0,0,0,1,1)$ encodes $S^* = \{2,6,7\}$
- $x_u + x_v = 2$ only for $\{u,v\} \in F = \{\{2,6\},\{2,7\},\{6,7\}\}$
- Notice $F \cap E = \varnothing$
- Hence, $x_u + x_v \leq 1$ for all $\{u,v\} \in E$



# So what?

- OK, so the **Mathematical Programming** (MP) **formulation**

$$\begin{array}{rrcl} \max_x & \sum_{v\in V} x_v & & \\ \forall \{u,v\}\in E & x_u + x_v & \le & 1 \\ & x & \in & \{0,1\}^{|V|} \end{array}$$

  describes STABLE correctly
- As long as we can't solve it, why should we care?

# The magical method

- But WE CAN!
- Use generic MP solvers
- These algorithms can solve *ANY* MP formulation expressed with linear forms, or *prove* that there is no solution
- Based on Branch-and-Bound (BB)
- The YES certificate is the characteristic vector of a feasible solution
- The NO certificate is the whole BB tree, which implicitly (and intelligently) enumerates the feasible set
- YES certificate lengths are polynomial, NO certificates may have exponential length

# CLIQUE and MATCHING

- Clique (use complement graph):

$$\begin{array}{rrcl} \max_x & \sum_{v\in V} x_v & & \\ \forall \{u,v\} \not\in E, u \neq v & x_u + x_v & \leq & 1 \\ & x & \in & \{0,1\}^{|V|} \end{array}$$

- Matching:

$$\begin{array}{rrcl} \max_x & \sum_{\{u,v\}\in E} x_{uv} & & \\ \forall u \in V & \sum_{v\in N(u)} x_{uv} & \leq & 1 \\ & x & \in & \{0,1\}^{|E|} \end{array}$$

**Warning**: although MATCHING$\in$**P**, solving the MP formulation with BB is exponential-time

# How to

- Come see me, I'll give you a personal demo
- Go to `www.ampl.com` and download the AMPL software, student version
- AMPL is for modelling, i.e. writing MP formulations
- Still from `www.ampl.com`, you can download a student version of the ILOG CPLEX BB implementation

# And tomorrow?

**If you're interested in modelling problems as MPs**

- M1:
  - MAP557 (Optimization: Theory and Applications)
- M2:
  - MPRO (Master Parisien en Recherche Operationnelle)
    `http://uma.ensta-paristech.fr/mpro/`

# The end