Graph Theory

Discrete Math, Spring 2025

Konstantin Chukharev

Graph Theory

- Graphs & digraphs
- Paths & connectivity
- Trees & spanning trees
- Bipartite graphs
- Matchings & Hall's theorem
- Planarity & coloring
- Network flows

Languages & Computation

- Alphabets & formal languages
- Regular expressions
- Finite automata (DFA, NFA)
- Pumping lemma
- Context-free grammars
- Pushdown automata
- Turing machines
- Decidability & complexity

Combinatorics & Recurrences

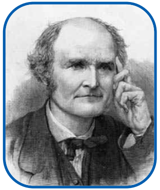
- Counting principles
- Permutations & combinations
- Inclusion–exclusion
- Partitions & Stirling numbers
- Generating functions
- Recurrence relations
- Asymptotic analysis

Graph Theory

"The origins of graph theory are humble, even frivolous."

— Norman L. Biggs

Leonhard Euler



Arthur Cayley

William Rowan Hamilton

Karl Menger

Philip Hall

Why Graph Theory?

Graphs are *everywhere* — they model relationships, connections, and structures.

Real-world applications:

- Social networks (friendships)
- Computer networks (routers)
- Transportation (roads, flights)
- Biology (protein interactions)
- Chemistry (molecular bonds)

Computer science applications:

- Data structures (linked lists, trees)
- Algorithms (shortest paths, flows)
- Compilers (dependency graphs)
- Databases (query optimization)
- AI (neural networks, knowledge graphs)

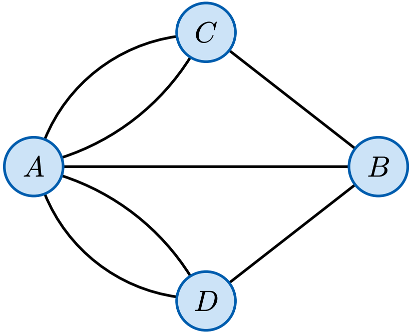
The power of abstraction: By stripping away irrelevant details, graphs let us see the *structure* of a problem. The same algorithm that finds the shortest route between cities also finds the fastest path in a game tree or the most efficient way to schedule tasks.

The Seven Bridges of Königsberg

In 1736, Leonhard Euler solved a famous puzzle:

Can one walk through the city of Königsberg, crossing each of its seven bridges exactly once?

Euler proved this is *impossible* — and in doing so, invented graph theory.



Historical note: This problem marks the birth of *topology* and *graph theory* as mathematical disciplines.

Basic Definitions

What is a Graph?

Graphs as models: Graphs are *mathematical abstractions* for modeling relationships, connections, and structures. Different kinds of relationships lead to different types of graphs.

Definition 1 (Abstract Approach): A *graph* is fundamentally a triple $G = (V, E, F)$, where:

- $V = \{v_1, v_2, ...\}$ is a finite set of *abstract vertices* (unique objects)
- $E = \{e_1, e_2, ...\}$ is a finite set of *abstract edges* (connections)
- F is a collection of *functions* that capture the graph's structure and semantics

The power of abstraction: Vertices and edges are just *labels* — the functions F define *all* the meaning:

- For *undirected* graphs: $F = \left\{\text{ends} : E \to \binom{V}{2}\right\}$ maps each edge to its two endpoints
- For *directed* graphs: $F = \{\text{begin} : E \to V, \text{end} : E \to V\}$ specify source and target
- For *weighted* graphs: add $\text{weight} : E \to \mathbb{R}$

What is a Graph? [2]

- For *hypergraphs:* $\text{incidence} : E \to 2^V$ maps edges to *subsets* of vertices
- For *vertex-labeled* graphs: add $\text{label} : V \to \Sigma$ for some alphabet Σ

Notation:

- $V(G)$ denotes the vertex set of graph G
- $E(G)$ denotes the edge set of graph G
- $|V(G)|$ is the *order* of G (number of vertices)
- $|E(G)|$ is the *size* of G (number of edges)

Bonus: This abstract approach handles *multigraphs* (parallel edges) and *loops* naturally — multiple edges in E can map to the same endpoint pair, and a loop edge maps to a singleton set $\{v\}$ or has $\text{begin}(e) = \text{end}(e) = v$.

Structural Representation (Alternative Approach)

Definition 2 (Structural Approach): Instead of abstract edges + functions, we can *encode structure directly* into the edge definition:

- *Undirected:* $E \subseteq \binom{V}{2}$ (unordered pairs $\{u, v\}$)
- *Directed:* $E \subseteq V \times V$ (ordered pairs (u, v))
- *Weighted:* $E \subseteq V \times V \times \mathbb{R}$ (triples (u, v, w))
- *Loops:* Include singletons $\{v\}$ in E or allow (v, v)

Trade-offs:

- *Pros:* Simpler for basic graphs; closer to programming impl (edge lists, adjacency matrices)
- *Cons:* Less flexible; need ad-hoc extensions for weighted graphs, hypergraphs, attributes; mixing structure with semantics

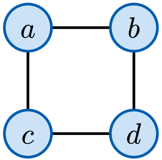
In practice: For this course, we'll mostly use the *structural representation* for simplicity, but keep the *abstract view* in mind — it explains why we can freely add weights, directions, labels, *etc.*

Undirected vs Directed Graphs

Definition 3 (Undirected Graph): In an *undirected graph*, edges are *unordered pairs*:

$$E \subseteq \binom{V}{2} = \{\{u, v\} \mid u, v \in V, u \neq v\}$$

The edge $\{u, v\}$ connects u and v symmetrically.



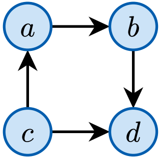
Undirected

Models: Mutual relationships (friendships, two-way roads, chemical bonds)

Definition 4 (Directed Graph): In a *directed graph* (digraph), edges are *ordered pairs*:

$$E \subseteq V \times V$$

The edge (u, v) goes *from* u *to* v.



Directed

Models: One-way relationships (follows, one-way streets, dependencies, function calls)

Simple Graphs, Multigraphs, and Pseudographs

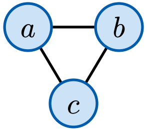
Definition 5:

- A *simple graph* has no *loops* (edges from a vertex to itself) and no *multi-edges* (multiple edges between the same pair of vertices).
- A *multigraph* allows *multi-edges* but no loops.
- A *pseudograph* allows both loops and multi-edges.

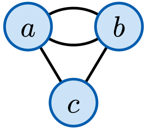
Abstract view: In the function-based approach, these distinctions are natural:

- *Simple:* the "ends" function is *injective* (different edges $\longrightarrow$ different endpoint pairs)
- *Multigraph:* "ends" can be non-injective; multiple edges map to the same $\{u, v\}$
- *Loops:* "ends" can map an edge to a singleton $\{v\}$ (or $\text{begin}(e) = \text{end}(e)$)

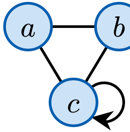
Note: Unless otherwise stated, "graph" means *simple undirected graph* in this course.



Simple



Multigraph



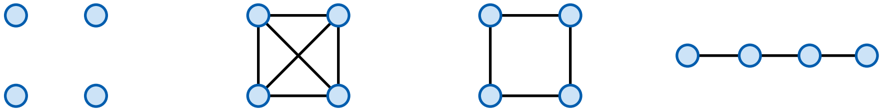
Pseudograph

Special Graphs

Definition 6:

- *Null graph*: no vertices ($V = \emptyset$)
- *Trivial graph*: single vertex, no edges ($|V| = 1$, $E = \emptyset$)
- *Empty graph* $\overline{K_n}$: n vertices, no edges
- *Complete graph* K_n: n vertices, all pairs connected
- *Cycle* C_n: n vertices in a cycle
- *Path* P_n: n vertices in a line

Example:



$\overline{K_4}$ (empty) K_4 (complete) C_4 (cycle) P_4 (path)

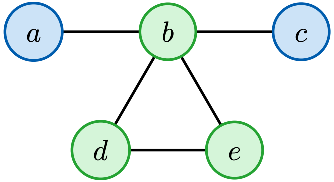
Theorem 1: The complete graph K_n has exactly $\binom{n}{2} = \frac{n(n-1)}{2}$ edges.

Adjacency and Incidence

Definition 7:

- Two vertices u and v are *adjacent* if there is an edge between them: $\{u, v\} \in E$.
- An edge e is *incident* to vertex v if v is an endpoint of e.
- The *neighborhood* of v is $N(v) = \{u \in V \mid \{u, v\} \in E\}$.

Example:



- a and b are *adjacent*
- a and c are *not adjacent*
- Edge $\{a, b\}$ is *incident* to a and b
- $N(b) = \{a, c, d, e\}$

Degree of a Vertex

Definition 8: The *degree* of a vertex v, denoted $\deg(v)$, is the number of edges incident to v.

- $\delta(G) = \min_{v \in V} \deg(v)$ is the *minimum degree*
- $\Delta(G) = \max_{v \in V} \deg(v)$ is the *maximum degree*

Theorem 2 (Handshaking Lemma): For any graph $G = \langle V, E \rangle$:

$$\sum_{v \in V} \deg(v) = 2 \; |E|$$

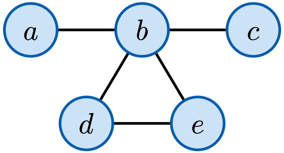
Proof: Each edge contributes exactly 2 to the sum of degrees (once for each endpoint). □

Corollary: The number of vertices with odd degree is always *even*.

Degree Sequences

Definition 9: The *degree sequence* of a graph is the list of vertex degrees in non-increasing order.

Example:



Degrees: $\deg(a) = 1$, $\deg(b) = 4$, $\deg(c) = 1$, $\deg(d) = 2$, $\deg(e) = 2$

Degree sequence: $(4, 2, 2, 1, 1)$

Question: Given a sequence of integers, can we determine if it's the degree sequence of some graph?

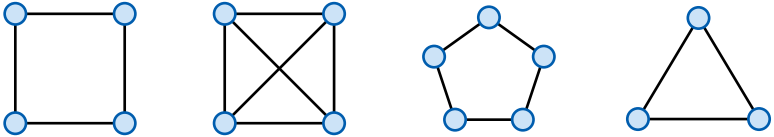
This is the *graph realization problem*.

Regular Graphs

Definition 10: A graph is *r-regular* if every vertex has degree r:

$$\forall v \in V : \deg(v) = r$$

Example:



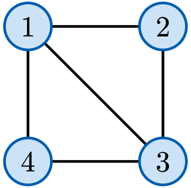
2-regular (cycle C_4) | **3-regular** (complete K_4) | **2-regular** (cycle C_5) | **2-regular** (complete K_3)

Graph Representations: Adjacency Matrix

Definition 11: The *adjacency matrix* A of a graph G with n vertices is an $n \times n$ matrix where:

$$A_{ij} = \begin{cases} 1 \text{ if } \{v_i, v_j\} \in E \\ 0 \text{ otherwise} \end{cases}$$

Example:



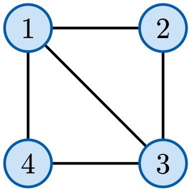
$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Properties: For undirected graphs, A is *symmetric*. The diagonal is all zeros for simple graphs.

Graph Representations: Adjacency List

Definition 12: The *adjacency list* representation stores, for each vertex v, a list of its neighbors $N(v)$.

Example:



Vertex	Neighbors
1	$2, 3, 4$
2	$1, 3$
3	$1, 2, 4$
4	$1, 3$

Space complexity: Adjacency matrix uses $O(n^2)$, adjacency list uses $O(n + m)$ where $m = |E|$.

Subgraphs

Definition 13: A graph $H = \langle V', E' \rangle$ is a *subgraph* of $G = \langle V, E \rangle$, denoted $H \subseteq G$, if

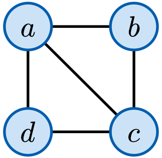
$$V' \subseteq V \quad \text{and} \quad E' \subseteq E$$

Definition 14:

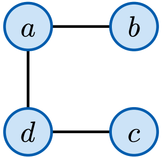
- A *spanning subgraph* includes all vertices: $V' = V$.
- An *induced subgraph* $G[S]$ on vertex set $S \subseteq V$ includes all edges between vertices in S:

$$E' = \{\{u, v\} \in E \mid u, v \in S\}$$

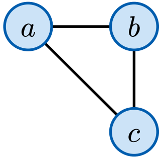
Example:



Original G



Spanning subgraph



Induced $G[\{a, b, c\}]$

Graph Isomorphism

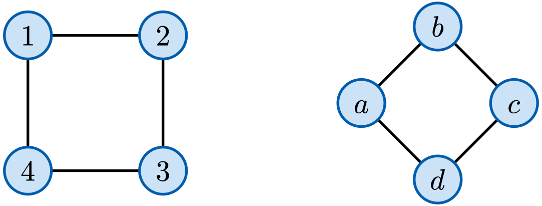
Definition 15: Graphs $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ are *isomorphic*, written $G_1 \simeq G_2$, if there exists a bijection $\varphi : V_1 \to V_2$ that *preserves adjacency*:

$$\{u, v\} \in E_1 \quad \Longleftrightarrow \quad \{\varphi(u), \varphi(v)\} \in E_2$$

Intuition: Isomorphic graphs are “the same graph” with different vertex labels. They have identical structure.

Graph Isomorphism [2]

Example:



Both graphs are isomorphic to C_4. The bijection $\varphi : 1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 4 \mapsto d$ preserves adjacency.

Computational mystery: Graph isomorphism is in NP but *not known* to be NP-complete or in P.

In 2015, Babai showed it's in *quasipolynomial time* — a major breakthrough, but the exact complexity remains open.

Summary: Graph Basics

Core concepts:

- A *graph* $G = (V, E)$ is a pair of vertices and edges connecting them
- *Directed* vs *undirected*; *simple* graphs vs *multigraphs* vs *pseudographs*
- *Degree* $\deg(v)$ counts edges incident to v; Handshaking Lemma: $\sum \deg(v) = 2|E|$
- *Special graphs:* Complete K_n, cycle C_n, path P_n, bipartite $K_{m,n}$, hypercube Q_n

Coming up: Paths, connectivity, trees, bipartite graphs, matchings, Eulerian and Hamiltonian cycles, planarity, and coloring.

Graph representations:

- *Adjacency matrix:* $n \times n$ matrix, good for dense graphs, $O(n^2)$ space
- *Adjacency list:* list of neighbors per vertex, good for sparse graphs, $O(n + m)$ space

Structural concepts:

- *Subgraph:* subset of vertices/edges; *induced subgraph:* includes all edges between chosen vertices
- *Graph isomorphism:* bijection preserving adjacency — graphs are "the same" up to relabeling

Paths and Connectivity

Walks, Trails, and Paths

Definition 16: A *walk* in a graph is an alternating sequence of vertices and edges:

$$v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$$

where each edge $e_i = \{v_{i-1}, v_i\}$.

- A *trail* is a walk with *distinct edges*.
- A *path* is a walk with *distinct vertices* (hence distinct edges).

Type	**Vertices repeat?**	**Edges repeat?**	**Closed version**
Walk	Yes ✓	Yes ✓	Closed walk
Trail	Yes ✓	No ✗	Circuit
Path	No ✗	No ✗	Cycle

Note: A walk/trail/path is *closed* if it starts and ends at the same vertex.

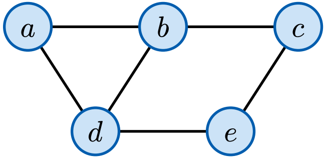
Length and Distance

Definition 17: The *length* of a walk (trail, path) is the number of edges in it.

Definition 18: The *distance* $\operatorname{dist}(u, v)$ between vertices u and v is the length of the shortest path from u to v.

If no path exists, we write $\operatorname{dist}(u, v) = \infty$.

Example:



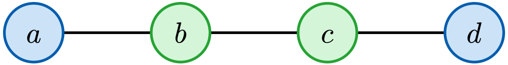
- $\operatorname{dist}(a, b) = 1$
- $\operatorname{dist}(a, c) = 2$
- $\operatorname{dist}(a, e) = 2$
- Path a-b-c has length 2
- Trail a-d-b-c-e-d has length 5

Eccentricity, Radius, and Diameter

Definition 19:

- *Eccentricity* of vertex v: $\text{ecc}(v) = \max_{u \in V} \text{dist}(v, u)$
- *Radius* of graph: $\text{rad}(G) = \min_{v \in V} \text{ecc}(v)$
- *Diameter* of graph: $\text{diam}(G) = \max_{v \in V} \text{ecc}(v)$
- *Center* of graph: $\text{center}(G) = \{v \in V \mid \text{ecc}(v) = \text{rad}(G)\}$

Example:



Path graph P_4:

- $\text{ecc}(a) = \text{ecc}(d) = 3$
- $\text{ecc}(b) = \text{ecc}(c) = 2$
- $\text{rad}(G) = 2$, $\text{diam}(G) = 3$
- $\text{center}(G) = \{b, c\}$

Theorem 3: For any connected graph G: $\text{rad}(G) \leq \text{diam}(G) \leq 2 \cdot \text{rad}(G)$

Connectivity

Definition 20: Two vertices u and v in an undirected graph G are *connected* if G contains a path from u to v. Otherwise, they are *disconnected*.

Definition 21: A graph G is *connected* if every pair of vertices in G is connected (*i.e.*, there exists a path between any two vertices).

A graph that is not connected is called *disconnected*.

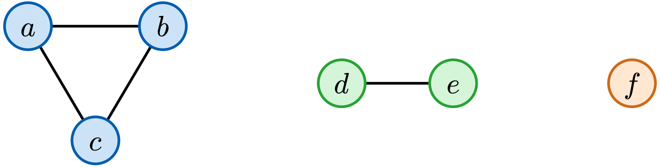
Note:

- A graph with a single vertex is connected (vacuously).
- An edgeless graph with two or more vertices is disconnected.

Connected Components

Definition 22: A *connected component* of G is a maximal connected subgraph.

Example:



This graph has 3 connected components: $\{a, b, c\}$, $\{d, e\}$, and $\{f\}$.

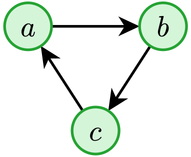
Key insight: "Being in the same connected component" is an *equivalence relation* on vertices.

Connectivity in Directed Graphs

Definition 23: A directed graph G is:

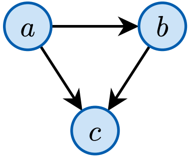
- **Weakly connected** if replacing all directed edges with undirected produces a connected graph.
- **Unilaterally connected** (or *semiconnected*) if for every pair of vertices u, v, there is a directed path from u to v *or* from v to u (or both).
- **Strongly connected** if for every pair of vertices u, v, there is a directed path from u to v *and* from v to u.

Example:



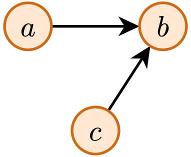
Strongly connected

$a \to b \to c \to a$



Unilaterally connected

$a \to b$, $a \to c$, $b \to c$



Weakly connected

No path $a \not\rightsquigarrow c$

Strongly Connected Components

Definition 24: A *strongly connected component* (SCC) of a digraph is a maximal strongly connected subgraph.

Condensation graph: If we contract each SCC to a single vertex, the result is a DAG (directed acyclic graph). This is called the *condensation* of G.

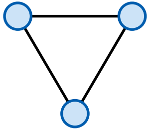
Algorithms: SCCs can be found in $O(n + m)$ time using Kosaraju's algorithm or Tarjan's algorithm (both based on DFS).

Girth

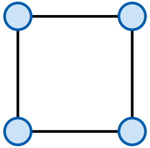
Definition 25: The *girth* of a graph G is the length of the shortest cycle in G.

If G has no cycles (is acyclic), we say $\text{girth}(G) = \infty$.

Example:



$\text{girth}(K_3) = 3$



$\text{girth}(C_4) = 4$

$\text{girth}(P_4) = \infty$

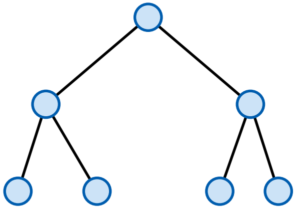
Trees and Forests

Trees: Definition

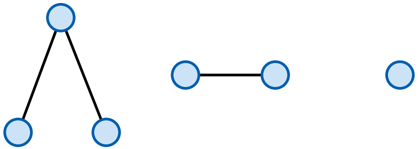
Definition 26: A *tree* is a connected acyclic graph.

A *forest* is an acyclic graph (a disjoint union of trees).

Example:



A tree



A forest (3 trees)

Characterizations of Trees

Theorem 4: For a graph G with n vertices, the following are equivalent:

1. G is a tree (connected and acyclic)
2. G is connected with exactly $n - 1$ edges
3. G is acyclic with exactly $n - 1$ edges
4. Any two vertices are connected by a *unique path*
5. G is *minimally connected*: removing any edge disconnects it
6. G is *maximally acyclic*: adding any edge creates a cycle

Why trees matter? Trees appear everywhere — file systems, parse trees, decision trees, spanning trees for network design. Their simple structure makes them amenable to recursive algorithms.

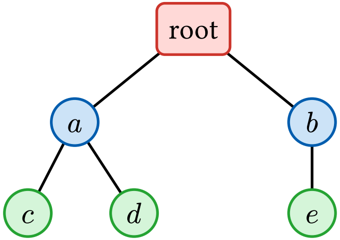
Rooted Trees

Definition 27: A *rooted tree* is a tree with one designated vertex called the *root*.

In a rooted tree:

- The *parent* of v is the neighbor of v on the path to the root
- The *children* of v are the other neighbors of v
- A *leaf* is a vertex with no children
- An *internal vertex* has at least one child

Example:



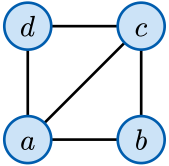
- Root has children a, b
- Leaves: c, d, e
- Internal vertices: root, a, b

Spanning Trees

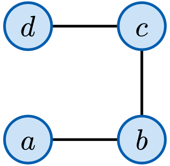
Definition 28: A *spanning tree* of a connected graph G is a spanning subgraph that is a tree.

Theorem 5: Every connected graph has at least one spanning tree.

Example:



Original graph



A spanning tree

Application: Finding minimum spanning trees (MST) is fundamental in network design.

Cayley's Formula

Theorem 6 (Cayley's Formula): The number of *labeled* trees on n vertices is exactly n^{n-2}.

Example:

- $n = 2$: $2^0 = 1$ tree (just one edge)
- $n = 3$: $3^1 = 3$ trees (three ways to pick the center)
- $n = 4$: $4^2 = 16$ trees
- $n = 5$: $5^3 = 125$ trees

Cayley's formula has many beautiful proofs. The most constructive uses *Prüfer sequences* — a bijection between labeled trees on $[n]$ and sequences in $[n]^{n-2}$.

Why n^{n-2}? Each of the $n - 2$ positions in a Prüfer sequence can be any of n vertices. The encoding is reversible, establishing the bijection.

Prüfer Sequences

Definition 29: A *Prüfer sequence* is a unique encoding of a labeled tree on n vertices as a sequence of $n - 2$ labels.

Encoding algorithm:

1. Find the leaf with the smallest label
2. Add its neighbor's label to the sequence
3. Remove the leaf from the tree
4. Repeat until 2 vertices remain

Example: Tree: 1-3-4-2, 3-5

Encoding: Remove 1 (neighbor 3), remove 2 (neighbor 4), remove 5 (neighbor 3).

Prüfer sequence: $(3, 4, 3)$

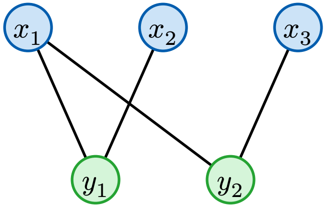
Bipartite Graphs

Definition of Bipartite Graphs

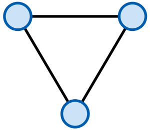
Definition 30: A graph $G = \langle V, E \rangle$ is *bipartite* if its vertices can be partitioned into two disjoint sets $V = X \sqcup Y$ such that every edge connects a vertex in X to a vertex in Y.

We write $G = \langle X \cup Y, E \rangle$ or $G = (X, Y, E)$.

Example:



Bipartite



Not bipartite
(contains triangle)

Characterization of Bipartite Graphs

Theorem 7: A graph is bipartite if and only if it contains no odd-length cycles.

Proof *(Sketch)*: (⇒) In a bipartite graph, any walk alternates between X and Y, so every cycle has even length.

(⇐) If no odd cycles exist, 2-color by BFS: pick any vertex, color it blue, color all neighbors green, color their neighbors blue, *etc.* No conflicts arise. □

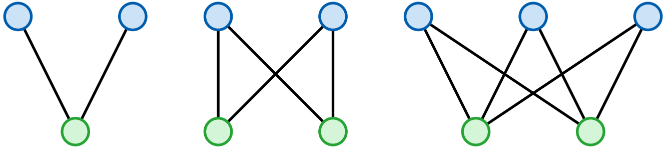
Bipartiteness can be checked in $O(n + m)$ time using BFS/DFS.
This is one of the few natural graph properties that admits efficient recognition.

Note: Checking if a graph is *3-colorable* is NP-complete, yet *2-colorable* (bipartite) is linear time!

Complete Bipartite Graphs

Definition 31: The *complete bipartite graph* $K_{m,n}$ has parts of sizes m and n, with every vertex in one part adjacent to every vertex in the other.

Example:



$K_{2,1}$ $K_{2,2}$ $K_{3,2}$

Note: $K_{m,n}$ has $m + n$ vertices and $m \cdot n$ edges.

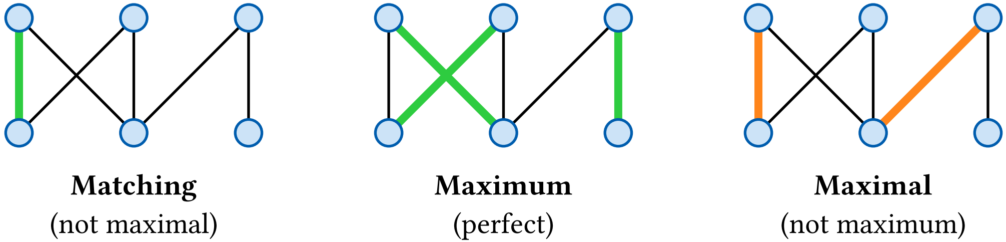
Matchings and Covers

Matchings

Definition 32: A *matching* $M \subseteq E$ is a set of pairwise non-adjacent edges (no two edges share a vertex).

Definition 33:
- A matching is *maximal* if no edge can be added to it.
- A matching is *maximum* if it has the largest possible size.
- A *perfect matching* covers all vertices.

Example:



Hall’s Marriage Theorem

Definition 34: Let $G = \langle X \cup Y, E \rangle$ be a bipartite graph. For a subset $S \subseteq X$, define the *neighborhood* of S:

$$N(S) = \{y \in Y \mid \exists x \in S : \{x, y\} \in E\}$$

Theorem 8 (Hall’s Marriage Theorem (Hall, 1935)): A bipartite graph $G = \langle X \cup Y, E \rangle$ has a matching that *saturates* X (*i.e.*, every vertex in X is matched) if and only if:

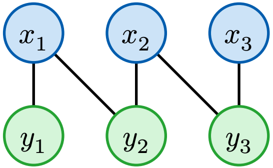
$$\forall S \subseteq X : |N(S)| \geq |S|$$

This is called **Hall’s condition** or the *marriage condition*.

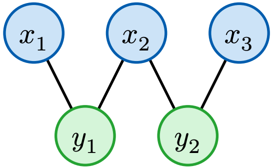
Examples: Hall's Condition

Why "Marriage"? Think of X as people seeking partners and Y as potential partners. Each person in X knows some people in Y (edges). Can everyone in X find a distinct partner? Only if no group of k people collectively knows fewer than k partners.

Example:



Satisfies Hall's Condition
Every subset S has $|N(S)| \geq |S|$.
Perfect matching exists.



Violates Hall's Condition
$S = \{x_1, x_2, x_3\}$ has $N(S) = \{y_1, y_2\}$.
Since $|N(S)| = 2 < 3 = |S|$, no matching saturates X.

Proof of Hall's Theorem

We prove both directions.

Direction (⇒): If a matching saturating X exists, then Hall's condition holds.

Proof: Let M be a matching that saturates X. For any $S \subseteq X$:

- Each vertex in S is matched to a distinct vertex in Y (by definition of matching).
- Let $M(S)$ be the set of partners of S under M. Then $|M(S)| = |S|$.
- Since every partner is a neighbor, $M(S) \subseteq N(S)$.
- Therefore: $|N(S)| \geq |M(S)| = |S|$. □

Direction (⇐): If Hall's condition holds, then a matching saturating X exists.

This is the interesting direction. We use **strong induction** on $n = |X|$.

Proof (Sufficiency): Base & Strategy

Base Case ($n = 1$): If $X = \{x\}$, Hall's condition gives $|N(\{x\})| \geq 1$, so x has a neighbor y. The edge $\{x, y\}$ is a matching.

Inductive Hypothesis: Assume the theorem holds for all bipartite graphs with $|X| < n$.

Inductive Step: Consider G with $|X| = n \geq 2$. We split into two cases:

- **Case 1:** Every proper subset S has *surplus* neighbors: $|N(S)| \geq |S| + 1$.
- **Case 2:** Some proper subset S is *tight*: $|N(S)| = |S|$.

Proof: Case 1 (Surplus)

Case 1: For all $\varnothing \neq S \subsetneq X$, we have $|N(S)| \geq |S| + 1$.

Strategy: Match an arbitrary edge, then use induction on the smaller graph.

1. Pick any edge $\{x, y\} \in E$ (exists because $X \neq \varnothing$ and Hall's condition ensures connectivity).
2. Remove both endpoints: let $G' = G - \{x, y\}$ and $X' = X \setminus x$.
3. **Verify Hall's condition in G':** Let $S' \subseteq X'$ be arbitrary.
   - In G, we have $|N_{G(S')}| \geq |S'| + 1$ (since $S' \subsetneq X$).
   - Removing y from Y reduces $|N(S')|$ by at most 1.
   - So $|N_{\{G'\}}(S')| \geq |N_{G(S')}| - 1 \geq (|S'| + 1) - 1 = |S'|$.
4. By induction, G' has a matching M' saturating X'.
5. Then $M = M' \cup \{\{x, y\}\}$ saturates X.

Proof: Case 2 (Tight Subset)

Case 2: There exists $\emptyset \neq S_0 \subsetneq X$ such that $|N(S_0)| = |S_0|$.

Strategy: Match S_0 independently, then match the rest.

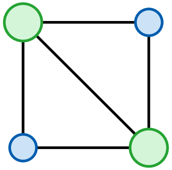
1. **Match S_0:** The induced subgraph $G[S_0 \cup N(S_0)]$ satisfies Hall's condition (inherited from G). Since $|S_0| < n$, by induction there exists a matching M_1 saturating S_0.
2. **Match the remainder:** Let $G' = G - S_0 - N(S_0)$ and $X' = X \setminus S_0$. We verify Hall's condition for G'. Let $A \subseteq X'$ be arbitrary.
   - In G: $|N_{G(A \cup S_0)}| \geq |A \cup S_0| = |A| + |S_0|$ (Hall's condition).
   - But $N_{G(A \cup S_0)} = N_{G(A)} \cup N_{G(S_0)} = N_{G(A)} \cup N(S_0)$ (disjoint by construction).
   - So $|N_{G(A)}| + |N(S_0)| \geq |A| + |S_0|$.
   - Since $|N(S_0)| = |S_0|$, we get $|N_{G(A)}| \geq |A|$.
   - In G', the neighbors of A are $N_{\{G'\}}(A) = N_{G(A)} \setminus N(S_0)$, but vertices in $N_{G(A)}$ were not in $N(S_0)$ (otherwise contradiction). So $|N_{\{G'\}}(A)| = |N_{G(A)}| \geq |A|$.
3. By induction, G' has a matching M_2 saturating X'.
4. Then $M = M_1 \cup M_2$ saturates X.

Vertex and Edge Covers

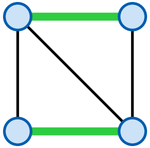
Definition 35: A *vertex cover* $R \subseteq V$ is a set of vertices such that every edge has at least one endpoint in R.

Definition 36: An *edge cover* $F \subseteq E$ is a set of edges such that every vertex is incident to at least one edge in F.

Example:



Vertex cover $\{a, c\}$



Edge cover $\{\{a, b\}, \{c, d\}\}$

König's Theorem

Theorem 9 (König's Theorem): In a bipartite graph:

$$\nu(G) = \tau(G)$$

where $\nu(G)$ is the size of a *maximum matching* and $\tau(G)$ is the size of a *minimum vertex cover*.

Key insight: This equality does *not* hold for general graphs! In a triangle K_3: $\nu = 1$ but $\tau = 2$.

Connection: König's theorem follows from the LP duality of matching and vertex cover. It also follows from the Max-Flow Min-Cut theorem on the associated network.

König's Theorem [2]

Theorem 10: In any graph without isolated vertices:

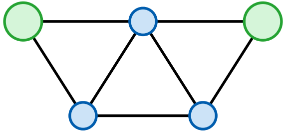
$$|\text{minimum vertex cover}| + |\text{maximum stable set}| = |V|$$

$$|\text{minimum edge cover}| + |\text{maximum matching}| = |V|$$

Stable Sets (Independent Sets)

Definition 37: A *stable set* (or *independent set*) $S \subseteq V$ is a set of pairwise non-adjacent vertices.

Example:



The green vertices $\{a, c\}$ form a stable set — no edges between them.

Complement relationship: S is a stable set in $G \iff S$ is a clique in $\overline{G}$.

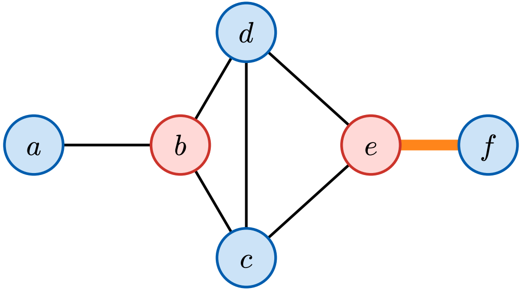
Connectivity Theory

Cut Vertices and Bridges

Definition 38: A *cut vertex* (or *articulation point*) is a vertex whose removal increases the number of connected components.

Definition 39: A *bridge* (or *cut edge*) is an edge whose removal increases the number of connected components.

Example:



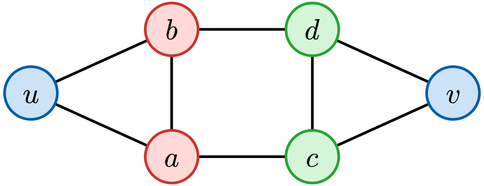
- Cut vertices: b, e
- Bridge: edge $\{e, f\}$

Separators and Cuts

Definition 40: For vertices $u, v \in V$, a *u-v separator* (or *u-v vertex cut*) is a set $S \subseteq V \setminus \{u, v\}$ such that u and v are in different components of $G - S$.

Definition 41: A *u-v edge cut* is a set $F \subseteq E$ such that u and v are in different components of $G - F$.

Example:



$S = \{a, b\}$ is a u-v separator. $S' = \{c, d\}$ is also a u-v separator.

Vertex and Edge Connectivity

Definition 42: The *vertex connectivity* $\kappa(G)$ is the minimum size of a vertex set S whose removal disconnects G or makes it trivial (single vertex).

Equivalently: $\kappa(G) = \min_{u,v}\{\text{minimum } u\text{-}v \text{ separator size}\}$ over all non-adjacent u, v.

Definition 43: The *edge connectivity* $\lambda(G)$ is the minimum size of an edge set F whose removal disconnects G.

Equivalently: $\lambda(G) = \min_{u,v}\{\text{minimum } u\text{-}v \text{ edge cut size}\}$ over all $u \neq v$.

For complete graphs K_n: we define $\kappa(K_n) = n - 1$ (need to remove all but one vertex).

k-Connectivity

Definition 44: A graph G is ***k*-vertex-connected** (or simply *k-connected*) if $\kappa(G) \geq k$.

Equivalently: G has more than k vertices, and $G - S$ is connected for every set S with $|S| < k$.

Definition 45: A graph G is ***k*-edge-connected** if $\lambda(G) \geq k$.

Equivalently: $G - F$ is connected for every edge set F with $|F| < k$.

Example:

- K_n is $(n-1)$-connected (both vertex and edge).
- C_n (cycle) is 2-connected and 2-edge-connected.
- A tree with $n \geq 2$ vertices has $\kappa = \lambda = 1$ (every edge is a bridge).

Whitney's Inequality

Theorem 11 (Whitney's Inequality): For any graph G:

$$\kappa(G) \leq \lambda(G) \leq \delta(G)$$

where $\delta(G)$ is the minimum degree.

Proof:

- $\lambda(G) \leq \delta(G)$: Removing all edges incident to a minimum-degree vertex disconnects it.
- $\kappa(G) \leq \lambda(G)$: Given a minimum edge cut F, for each edge in F pick one endpoint on the "smaller side". This gives a vertex separator of size $\leq |F|$. □

When are they equal? For k-regular graphs with high girth, often $\kappa = \lambda = k$. For example, the Petersen graph has $\kappa = \lambda = \delta = 3$.

Menger's Theorem

Theorem 12 (Menger's Theorem (Vertex Form)): Let u, v be non-adjacent vertices in G. Then:

$$\max\{\text{number of internally vertex-disjoint } u\text{-}v \text{ paths}\} = \min\{|S| : S \text{ is a } u\text{-}v \text{ separator}\}$$

Theorem 13 (Menger's Theorem (Edge Form)): For any distinct vertices u, v in G:

$$\max\{\text{number of edge-disjoint } u\text{-}v \text{ paths}\} = \min\{|F| : F \text{ is a } u\text{-}v \text{ edge cut}\}$$

Menger's theorem is equivalent to the Max-Flow Min-Cut theorem with unit capacities.

The *"flow"* (disjoint paths) and *"cut"* (separators) are *dual* notions.

Menger's Theorem: Corollaries

Theorem 14 (Global Vertex Connectivity): A graph G is k-connected if and only if every pair of distinct vertices is connected by at least k internally vertex-disjoint paths.

Theorem 15 (Global Edge Connectivity): A graph G is k-edge-connected if and only if every pair of distinct vertices is connected by at least k edge-disjoint paths.

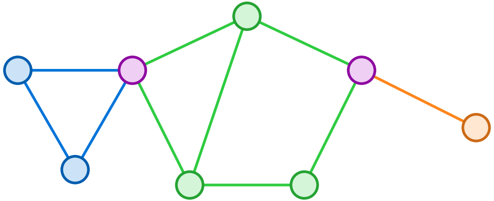
Intuition: High connectivity means many "independent routes" between any two vertices. Failure of a few vertices/edges cannot disconnect the graph.

Blocks (2-Connected Components)

Definition 46: A *block* of a graph G is a maximal connected subgraph with no cut vertex (*i.e.*, maximal 2-connected subgraph, or a bridge, or an isolated vertex).

Note:
- A 2-connected graph is its own single block.
- Every edge belongs to exactly one block.
- Blocks can share at most one vertex — and that vertex is a cut vertex.

Example:



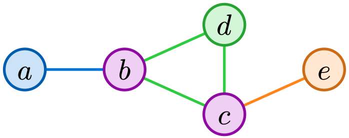
Three blocks: blue triangle, green pentagon, orange bridge. Purple = cut vertices.

Block-Cut Tree

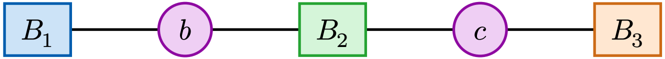
Definition 47: The *block-cut tree* (or *BC-tree*) of a connected graph G is a bipartite tree T where:

- One part contains a node for each *block* of G.
- The other part contains a node for each *cut vertex* of G.
- A block-node B is adjacent to a cut-vertex-node v iff $v \in B$.

Example:



Graph G



Block-Cut Tree

Applications: The block-cut tree decomposes G into 2-connected pieces. Many problems can be solved by dynamic programming on this tree.

Islands (2-Edge-Connected Components)

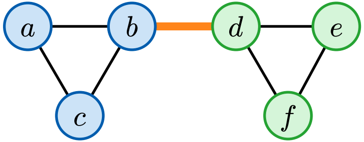
Definition 48: An *island* (or *2-edge-connected component*) is a maximal subgraph with no bridges.

Equivalently: vertices u and v are in the same island iff they lie on a common cycle.

Note:
- Islands are separated by bridges.
- Every vertex belongs to exactly one island.
- Unlike blocks, islands partition the vertex set (not just edges).

Blocks vs Islands

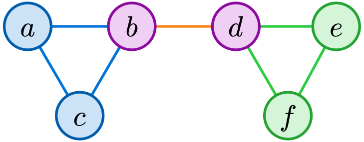
Example:



Islands

Blue and green are 2-edge-connected components.

Orange = bridge.



Blocks

Blue triangle, green triangle, orange bridge.

Purple = cut vertices b and d.

Key difference:

- **Blocks** = 2-vertex-connectivity: no cut vertices within a block.
- **Islands** = 2-edge-connectivity: no bridges within an island.

Blocks may share cut vertices; islands partition vertices.

Bridge Tree

Definition 49: The *bridge tree* (or *island tree*) of a connected graph G is obtained by contracting each island to a single vertex. The edges of this tree are exactly the bridges of G.

Analogy:

- Block-cut tree: decomposition by *cut vertices* into *blocks*.
- Bridge tree: decomposition by *bridges* into *islands*.

Theorem 16: A graph is 2-edge-connected iff its bridge tree is a single vertex (no bridges).

A graph is 2-vertex-connected iff its block-cut tree has a single block node.

Eulerian and Hamiltonian Graphs

Eulerian Paths and Circuits

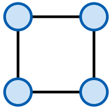
Definition 50:

- An *Eulerian trail* is a trail that visits every edge exactly once.
- An *Eulerian circuit* is a closed Eulerian trail.
- A graph is *Eulerian* if it has an Eulerian circuit.

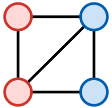
Theorem 17 (Euler's Theorem): A connected graph has an Eulerian circuit if and only if every vertex has *even degree*.

A connected graph has an Eulerian trail (but not circuit) if and only if exactly *two vertices* have odd degree.

Example:



Eulerian
(all degrees even)



Has Eulerian trail
(2 odd vertices)

Hamiltonian Paths and Cycles

Definition 51:
- A *Hamiltonian path* visits every vertex exactly once.
- A *Hamiltonian cycle* is a cycle that visits every vertex exactly once.
- A graph is *Hamiltonian* if it has a Hamiltonian cycle.

Warning: Unlike Eulerian graphs, there is *no simple characterization* of Hamiltonian graphs!

Determining if a graph is Hamiltonian is NP-complete.

Sufficient Conditions for Hamiltonicity

Theorem 18 (Ore's Theorem): If G has $n \geq 3$ vertices and for every pair of non-adjacent vertices u, v:

$$\deg(u) + \deg(v) \geq n$$

then G is Hamiltonian.

Theorem 19 (Dirac's Theorem): If G has $n \geq 3$ vertices and $\delta(G) \geq \frac{n}{2}$, then G is Hamiltonian.

Eulerian vs Hamiltonian: Summary

	Eulerian	**Hamiltonian**
Visits	Every *edge* once	Every *vertex* once
Characterization	Degree condition	NP-complete to decide
Algorithm	$O(m)$ – Hierholzer's	Exponential (backtracking)
Named after	Euler (1736)	Hamilton (1857)

Historical note: Hamilton sold a puzzle ("Icosian game") based on finding Hamiltonian cycles on a dodecahedron graph.

The dodecahedral graph has exactly 30 distinct Hamiltonian cycles.

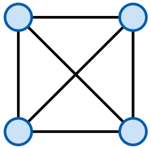
Planar Graphs

Planar Graphs: Definition

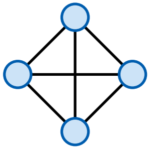
Definition 52: A graph is *planar* if it can be drawn in the plane without edge crossings.

A *plane graph* is a specific planar embedding (drawing) of a planar graph.

Example:



K_4 with crossings



K_4 planar embedding

K_4 is planar — it can be redrawn without crossings.

Faces and Euler's Formula

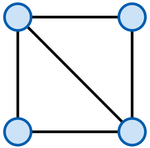
Definition 53: A *face* of a plane graph is a connected region bounded by edges. The unbounded region is the *outer face* (or *infinite face*).

Theorem 20 (Euler's Polyhedron Formula): For any connected plane graph with n vertices, m edges, and f faces:

$$n - m + f = 2$$

Deep insight: The quantity $n - m + f$ is called the *Euler characteristic*. It equals 2 for any surface homeomorphic to a sphere. For a torus, it equals 0. This connects graph theory to topology!

Example:



- Vertices: $n = 4$
- Edges: $m = 5$
- Faces: $f = 3$ (2 inner + 1 outer)

Check: $4 - 5 + 3 = 2$ ✓

Consequences of Euler's Formula

Theorem 21: For any simple planar graph with $n \geq 3$ vertices and m edges:

$$m \leq 3n - 6$$

Proof: Each face has ≥ 3 edges on its boundary, and each edge borders at most 2 faces. So $3f \leq 2m$, giving $f \leq \frac{2m}{3}$.

By Euler's formula: $2 = n - m + f \leq n - m + \frac{2m}{3} = n - \frac{m}{3}$. Therefore $m \leq 3n - 6$. □

Theorem 22: For any simple planar *bipartite* graph with $n \geq 3$ vertices:

$$m \leq 2n - 4$$

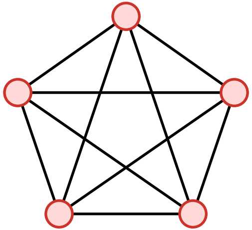
Corollary: K_5 (with 10 edges but $3 \cdot 5 - 6 = 9$) and $K_{3,3}$ (with 9 edges but $2 \cdot 6 - 4 = 8$) are *not* planar.

Kuratowski's Theorem

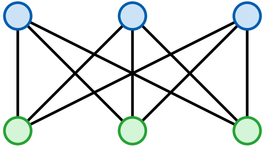
Theorem 23 (Kuratowski's Theorem): A graph is planar if and only if it contains no subdivision of K_5 or $K_{3,3}$ as a subgraph.

Definition 54: A *subdivision* of a graph G is obtained by replacing edges with paths.

Example:



K_5 (not planar)



$K_{3,3}$ (not planar)

Graph Coloring

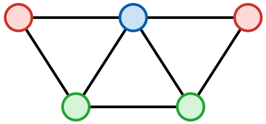
Vertex Coloring

Definition 55: A *(proper) vertex coloring* of a graph assigns colors to vertices such that adjacent vertices receive different colors.

Definition 56: A graph is *k-colorable* if it has a proper coloring using at most k colors.

The *chromatic number* $\chi(G)$ is the minimum k such that G is k-colorable.

Example:



This graph is 3-colorable. Is $\chi(G) = 3$?

Chromatic Number: Bounds

Theorem 24: For any graph G:

$$\omega(G) \leq \chi(G) \leq \Delta(G) + 1$$

where $\omega(G)$ is the *clique number* and $\Delta(G)$ is the maximum degree.

Proof *(Lower bound)*: A clique of size k needs k different colors. □

Theorem 25 (Brooks' Theorem): For any connected graph G that is not a complete graph or an odd cycle:

$$\chi(G) \leq \Delta(G)$$

Computing $\chi(G)$ is NP-hard, but checking 2-colorability is $\mathcal{O}(n + m)$.

The Four Color Theorem

Theorem 26 (Four Color Theorem): Every planar graph is 4-colorable: $\chi(G) \leq 4$ for all planar G.

A controversial proof:

- Conjectured in 1852 by Francis Guthrie
- Proved in 1976 by Appel and Haken *using a computer*
- First major theorem requiring computational verification
- Checked 1,500 "unavoidable" configurations
- Sparked debates: Is a computer-assisted proof a "real" proof?

The dual view: Coloring vertices of a planar graph = coloring regions of a map so adjacent regions differ. Every map needs at most 4 colors!n

Edge Coloring

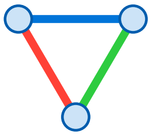
Definition 57: An *edge coloring* assigns colors to edges such that edges sharing a vertex receive different colors.

The *chromatic index* $\chi'(G)$ is the minimum number of colors needed.

Theorem 27 (Vizing's Theorem): For any simple graph G:

$$\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$$

Example:



Triangle K_3 needs 3 colors: $\chi'(K_3) = 3 = \Delta + 1$.

Cliques and Stable Sets

Cliques

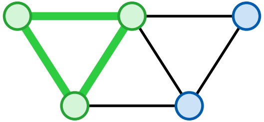
Definition 58: A *clique* is a subset of vertices $Q \subseteq V$ such that every pair of vertices in Q is adjacent.

Equivalently, Q induces a complete subgraph.

Definition 59:

- *Clique number* $\omega(G)$: size of the largest clique
- A clique is *maximal* if no vertex can be added
- A clique is *maximum* if it has the largest possible size

Example:



Maximum clique $\{a, b, c\}$ shown in green. $\omega(G) = 3$.

Ramsey Theory: A Taste

Theorem 28 (Ramsey's Theorem (simplified)): For any positive integers r and s, there exists a number $R(r, s)$ such that any 2-coloring of the edges of K_n (with $n \geq R(r, s)$) contains either a red K_r or a blue K_s.

Example: $R(3, 3) = 6$: Among any 6 people, there are either 3 mutual friends or 3 mutual strangers.

Warning: Computing Ramsey numbers is extremely hard. We know $R(3, 3) = 6$, $R(4, 4) = 18$, but $R(5, 5)$ is unknown!

Famous quote by Erdős: "Suppose aliens invade the earth and threaten to obliterate it in a year's time unless human beings can find $R(5, 5)$. We could marshal the world's best minds and fastest computers, and within a year we could probably calculate the value. If they digit $R(6, 6)$, we would have no choice but to launch a preemptive attack."

Summary and Connections

Graph Theory: Key Concepts

Structural concepts:

- Degree, adjacency, neighborhoods
- Paths, cycles, connectivity
- Trees, spanning trees, forests
- Bipartiteness (2-colorability)
- Planarity (Euler's formula)

Optimization problems:

- Matchings (Hall, König)
- Vertex/edge covers
- Graph coloring (χ, χ')
- Cliques and stable sets
- Connectivity (Menger)

Foundational theorems:

- Handshaking: $\sum \deg(v) = 2m$
- Euler's formula: $n - m + f = 2$
- Hall's marriage theorem (matchings $\leftrightarrow$ neighborhoods)
- Menger's theorem (paths $\leftrightarrow$ cuts)
- Four color theorem (planarity $\rightarrow$ 4-colorability)

What's Next: Flow Networks

Coming up: Network flows unify and generalize graph theory:

- Maximum bipartite matching = max flow in unit network
- Menger's theorem = max-flow min-cut with unit capacities
- Hall's condition = flow feasibility check
- König's theorem = LP duality for bipartite matching

Graph theory provides the foundation for:

- Algorithms (BFS, DFS, shortest paths, MST)
- Network design and optimization
- Formal language theory (automata are directed labeled graphs!)
- Combinatorics, counting, and probabilistic methods

Exercises

1. Prove that every tree with $n \geq 2$ vertices has at least 2 leaves.
2. Show that the Petersen graph is not planar.
3. Find the chromatic number of C_n for all $n \geq 3$.
4. Prove König's theorem using Hall's theorem.
5. For which values of n does K_n have an Eulerian circuit?
6. Find all graphs G with $\kappa(G) = \lambda(G) = \delta(G)$.
7. Prove that every 2-connected graph has no cut vertices.
8. Show that a graph is bipartite iff it has no odd cycles.