Chapter 0

Introduction

Combinatorial problems and arguments have a long history in mathematics, but only in the last half of the 20th century did they become a coherent subject. The discipline was long viewed as a collection of isolated tricks, but now the methods are more systematic, and the connections and applications between combinatorics and other areas of mathematics (in both directions) are being studied.

In this book we explore some of these connections and many fundamental results of combinatorics. We do not assume any prior exposure to combinatorics, but we assume mathematical maturity and basic undergraduate mathematics, including elementary set theory, induction, equivalence relations, limits, calculus, linear algebra, etc.

One can classify mathematical problems by the type of question, the object being studied, the method used, etc. These various aspects make it hopeless to impose a linear order of development in the study of mathematics. We emphasize different aspects at different times.

Our questions are of three general types. Given constraints specified for an object, does it exist? If such objects exist, how many are there? With respect to some criterion, which one is the best? These are the problems of Existence, Enumeration, and Extremality. We emphasize enumerative problems in Part I and problems of existence and extremality in most of the rest of the text.

We study objects that are discrete structures of various types. The simplest is a set. More complicated structures arise by imposing constraints or relations on sets or families of sets. We study various arrangements in Part I and graphs in Part II. In Part III we study structures such as hypergraphs, partially ordered sets, combinatorial designs, and matroids.

Finally, we also study methods of combinatorics. Many techniques arise in conjunction with particular structures, but some are used in many contexts and are worthy of study in their own right. Our focus on techniques is clearest in Part IV, where we discuss the probabilistic method, algebraic methods, and connections with geometry, but many other methods appear in earlier parts.

In this brief introduction, we review definitions from elementary mathematics, introduce elementary concepts about graphs for use in Part I, introduce elementary notions of probability as background for questions throughout the text, describe some additional discrete structures, and mention the basic notions of complexity. This is background material to be consulted as needed.

SETS, FUNCTIONS, AND RELATIONS

Our most fundamental object is a set. We build other structures from sets and relations. We use $\mathbb{N}_0$ for the set of nonnegative integers and $\mathbb{N}$ for the set of positive integers, also called the natural numbers. We let $[n]$ (pronounced "bracket n") denote the set $\{1, \ldots, n\}$ of the first n natural numbers, with $[0] = \varnothing$. We take as given the number systems $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ (natural numbers, integers, rational numbers, real numbers, complex numbers) and their elementary arithmetic and order properties. Similarly we assume the elementary operations and notation of sets, such as membership, containment, union, intersection, complement, and difference. We write the difference of sets A and B as $A - B$, not $A \setminus B$.

A **function** f from a set A to a set B, written $f\colon A \to B$, assigns each $x \in A$ an element $f(x) \in B$. The set A is the **domain**; f is *defined on* A. The **image** of $x \in A$ is $f(x)$, and $\{f(x)\colon x \in A\}$ is the **image** of f. The function is **injective** if each $y \in B$ is the image of at most one element of A. It is **surjective** if each $y \in B$ is the image of at least one element of A. It is a **bijection** if it is injective and surjective, and then it provides a **one-to-one correspondence** between A and B.

A set S is **finite** if a bijection from S to $[n]$ exists for some $n \in \mathbb{N}_0$; the value n is then the **size** of S, written $|S|$. *Counting* a finite set means computing its size. Two sets (finite or infinite) have the same **cardinality** if there is a bijection from one to the other. For finite sets, "cardinality" is a synonym for "size". A set with the same cardinality as $\mathbb{N}$ is **countable**.

A **sequence** is a function with domain $\mathbb{N}$ (or $\mathbb{N}_0$); we write a_n for the image of n, the nth **term** of the sequence. Usually one letter (such as f) denotes a function; similarly we use $\langle a \rangle$ to denote the sequence with terms of the form a_n.

A **list** is a function defined on $[n]$ for some $n \in \mathbb{N}_0$; this is the finite analogue of a sequence. We write a list a of *length* n as an **n-tuple** $(a_1, \ldots, a_n)$. (Many authors use "sequence" for an n-tuple; we try to use "sequence" only for functions on $\mathbb{N}$.) A **binary n-tuple** or **$0,1$-list** is a list with entries in $\{0,1\}$. Similarly, a **$0,1$-matrix** or **binary matrix** has entries in $\{0,1\}$. A **ternary list** has entries in $\{0,1,2\}$. An **n-ary list** takes values from a specified set of size n. An **arithmetic progression** is a list of equally spaced integers.

In contrast to a set, the order of elements in a list matters, and elements in lists may repeat. A **multiset** differs from a set by allowing repeated elements, but order remains unimportant. We can specify a multiset by specifying the set of distinct elements and their multiplicities. Since the order is unimportant but repetition is allowed, some authors refer to multisets as "unordered lists".

A **permutation** of a finite set S is a bijection from S to itself. Since a function on $[n]$ is a list, we may view a permutation σ of $[n]$ as a function from $[n]$ to $[n]$ or as a listing of $[n]$ in some order $a_1, \ldots, a_n$, with a_i denoting $\sigma(i)$. The latter is the **word form** of the permutation. Both viewpoints will be useful.

We often discuss sets whose elements are also sets. To avoid confusion, we use **class** and **family** as synonyms for "set". Instead of saying "a set in a set of sets", we say "a member of a family of sets".

The **cartesian product** of sets S and T is the set $S \times T$ of ordered pairs $\{(s,t)\colon s \in S, t \in T\}$. A **(binary) relation** between S and T is a subset of the cartesian product $S \times T$. When $S = T$, we call this a *relation on* S. We say that the pairs in a relation *satisfy* the relation.

When S is a family of sets, the **containment relation** on S is the set of pairs $(A, B) \in S \times S$ such that $A \subseteq B$. The pair (A, B) satisfies the **disjointness relation** when $A \cap B = \varnothing$. As a relation, disjointness is a property of pairs. Hence "a family of disjoint sets" technically has no meaning; nevertheless, by convention the word "disjoint" means "pairwise disjoint" when applied to sets in a family.

An **equivalence relation** on S is a relation R with these properties:

1) **reflexive** — $(x, x) \in R$ for all $x \in S$.
2) **symmetric** — $(x, y) \in R$ if and only if $(y, x) \in R$.
3) **transitive** — if $(x, y), (y, z) \in R$, then $(x, z) \in R$.

Containment is reflexive and transitive but not symmetric and hence not an equivalence relation. "Having the same cardinality" defines an equivalence relation, since the identity function is a bijection from a set to itself, the inverse of a bijection is a bijection, and the composition of bijections is a bijection.

An **equivalence class** of an equivalence relation R on S is a maximal subset T of S such that all pairs of elements of T satisfy R. Elements x and y are in the same equivalence class if and only if $(x, y) \in R$, and each element of S belongs to exactly one such class. Hence the equivalence classes of an equivalence relation on S form a partition of S, where a **partition** of a set S is a family of disjoint nonempty sets whose union is S. The sets in a partition are its **blocks**. Conversely, given a partition of S, putting $(x, y) \in R$ if x and y are in the same block yields an equivalence relation R on S. Hence partitions of S and equivalence relations on S are essentially the same notion.

0.1. Example. *Examples of equivalence relations.*

Congruence modulo n. Integers x and y are *congruent modulo* n if $x - y$ is a multiple of n. The equivalence classes are the subsets of $\mathbb{Z}$ having a fixed remainder upon division by n. These are the **congruence classes** modulo n, and the family of congruence classes modulo n is denoted by $\mathbb{Z}_n$.

Orbits under a permutation. Viewing a permutation σ of $[n]$ as a bijection from $[n]$ to $[n]$, we use σ^k to denote the bijection obtained by applying σ successively k times. The relation R on $[n]$ that puts $(i, j) \in R$ if $\sigma^k(i) = j$ for some $k \in \mathbb{N}_0$ is reflexive and transitive. Since bijections have inverses, also R is symmetric. The equivalence class containing i in this equivalence relation is the set of elements obtained as σ is repeatedly applied, called its **orbit**. ■

0.2. Example. When S is a family of subsets of a "ground set" X, the **incidence relation** between X and S is the relation R consisting of the ordered pairs $(x, A) \in X \times S$ such that $x \in A$. When $X = \{x_1, \ldots, x_n\}$ and $S = \{S_1, \ldots, S_m\}$, we encode R as a 0, 1-matrix with position (i, j) being 1 if $x_i \in S_j$ and 0 otherwise. This is the **incidence matrix** for R (with respect to the given indexing of X and S). The jth column of the incidence matrix is the **incidence vector** of the set S_j; this 0, 1-vector records for each element of X whether it belongs to S_j. ■

GRAPHS

Many natural relations are symmetric and irreflexive, where **irreflexive** means that for each x the pair (x, x) does *not* satisfy the relation (disjointness on a family of nonempty sets, for example). Such relations are modeled by "graphs".

A **graph** G is a pair consisting of a set $V(G)$ of **vertices** and a set $E(G)$ of **edges**, where each edge is a set of two vertices. The **order** of G is $|V(G)|$, and its **size** is $|E(G)|$. Vertices u and v forming an edge are **adjacent**, and the **neighbors** of v are the vertices adjacent to it. Note that the adjacency relation is symmetric and irreflexive. The two vertices in an edge are its **endpoints**, and we say that the edge **joins** its endpoints.

An edge and its endpoints are **incident**. The **incidence relation** of a graph G is the set of pairs (v, e) in $V(G) \times E(G)$ such that v and e are incident to each other (v is an endpoint of e). When discussing graphs we drop set brackets for edges and write the edge $\{u, v\}$ as uv (or vu). To display a graph, we represent each vertex by a point in the plane and each edge by a curve whose endpoints are the points assigned to the vertices of the edge; this is a **drawing** of the graph.

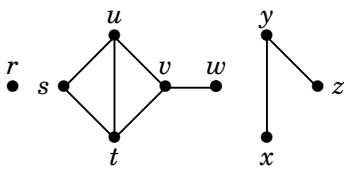
Structural properties of graphs do not depend on the names of the vertices. An **isomorphism** from a graph G to a graph H is a bijection $f\colon V(G) \to V(H)$ that preserves the adjacency relation: $uv \in E(G)$ if and only if $f(u)f(v) \in E(H)$. The resulting **isomorphism relation** is an equivalence relation on any set of graphs. Every graph isomorphic to G has the same structural properties as G, so we treat isomorphic graphs as "equal" when not given explicit vertex names.

Containment and union for graphs follow those notions for sets. A graph H is a **subgraph** of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Similarly, the **union** of graphs G and H is the graph $G \cup H$ with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$.

Many interesting enumerative questions can be asked about graphs. To permit discussion of such enumerative questions in Part I, we develop some elementary examples and properties of graphs here.

Many useful graphs are defined by their structural properties. A **path** is a graph whose vertices can be linearly ordered so that two vertices are adjacent if and only if they are consecutive in the ordering; the **endpoints** of the path are the first and last vertices in such an order. A **cycle** is a graph with an equal number of vertices and edges whose vertices can be placed around a circle so that two vertices are adjacent if and only if they appear consecutively along the circle.

A graph G is **connected** if for all $u, v \in V(G)$, it contains a path with endpoints u and v. A **component** of G is a maximal connected subgraph of G, where a **maximal** object of a given type is one that is not contained in another object of that type (**minimal** is defined similarly). The graph below has three components.



A u, v-**path** is a path with endpoints u and v. We say that u and v *are connected in* G if G contains a u, v-path. The **connection relation** on $V(G)$ is the set of pairs (u, v) such that G has a u, v-path; it is reflexive and symmetric. Our first proposition implies that it also is transitive and hence is an equivalence relation. The point is that although a u, v-path and a v, w-path together need not form a u, w-path, their union contains one. The equivalence classes of the connection relation are the vertex sets of the components of G.

The technique of **extremality** involves choosing an object that is extremal in some respect (a maximal connected subgraph, for example). This often requires finiteness, and our structures will be finite unless explicitly stated otherwise.

0.3. Proposition. If P is a u, v-path and P' is a v, w-path, then $P \cup P'$ contains a u, w-path.

Proof: We use extremality. Let x be the first vertex along P from u to v that also lies in P' (x exists, since both sets contain v). The union of the u, x-path in P and the x, w-path in P' is a u, w-path, since these subgraphs share only x. ■

A **spanning subgraph** of G is a subgraph of G with vertex set $V(G)$. A **tree** is a graph that is connected and contains no cycles. A **spanning tree** in G is a spanning subgraph of G that is a tree.

0.4. Proposition. Every connected graph contains a spanning tree.

Proof: Since every vertex is itself a tree, every connected graph G contains at least one tree. Let T be a maximal tree contained in G.

If $V(T) \neq V(G)$, then a path from a vertex outside $V(T)$ to a vertex in $V(T)$ yields an edge e with endpoints in $V(T)$ and $V(G) - V(T)$. Let T' be the subgraph of G obtained from T by adding edge e and its endpoint v outside T. By the transitivity of the connection relation, T' is connected.

Also every cycle in T' appears in T, since v is in only one edge in T'. Thus T' has no cycle and is a tree. This contradicts our extremal choice of T. We conclude that $V(T) = V(G)$, and T is a spanning tree. ■

The **degree** of a vertex in a graph is the number of edges incident to it. A **leaf** is a vertex of degree 1. A vertex of degree 0 is an **isolated vertex**.

A maximal path in a graph G is a path in G that is not a subgraph of another path in G. Thus every vertex of G adjacent to an endpoint of a maximal path in G must belong to the path.

0.5. Proposition. If all vertices in a finite graph G have degree at least 2, then G contains a cycle.

Proof: Since $V(G)$ is finite, G has a maximal path P. Let v be an endpoint of P. Since $d(v) \geq 2$, there is an edge vu not in P. Since P is maximal, u lies on P, and vu completes a cycle with the u, v-path in P. ■

0.6. Proposition. Every tree with at least two vertices has at least two leaves. Deleting a leaf from a tree yields a tree with one less vertex.

Proof: A connected graph with at least two vertices has no isolated vertices. A tree has no cycle, so the endpoints of a maximal path in a tree with at least two vertices have degree 1.

Given a leaf x in a tree G, obtain G' from G by deleting x and its incident edge. Since deleting a vertex creates no new subgraphs, G' has no cycles. Hence it suffices to show that G' is connected. For distinct vertices $u, v \in V(G')$, there a u, v-path P in G. Since internal vertices along a path have degree at least 2, P does not contain x. Hence P is also a u, v-path in G'. ■

0.7. Proposition. Every tree with n vertices has $n-1$ edges. Furthermore, every graph with $n-1$ edges that arises from n isolated vertices by iteratively adding an edge joining two components is a tree.

Proof: For the first statement, use induction on n. A 1-vertex tree has no edges. For $n > 1$, Proposition 0.6 provides a leaf whose deletion yields a tree with $n-1$ vertices. Since it has $n-2$ edges, the original tree has $n-1$ edges.

For the second statement, note first that each such edge addition creates no cycles. Such a cycle would contain the new edge uv and another u, v-path from the previous graph, which does not exist since u and v were in different components in that graph. Also, each such edge addition reduces the number of components by 1. Hence $n-1$ additions reduce the number of components to 1. Thus the resulting graph is acyclic and connected and is a tree. ■

We assume familiarity with these results in order to study counting problems about trees in Part I. We discuss the structure of graphs more fully in Part II.

DISCRETE PROBABILITY

Many enumerative questions are easily motivated using discrete probability. In such questions, there is a set U (the "universe") of possible outcomes of some process. These outcomes are assumed to be equally likely; this is the meaning of phrases like "a random element" and "chosen uniformly at random". The probability of a desired property is then defined to be $|A|/|U|$, where A is the subset of U consisting of all outcomes having the desired property.

In Part III we will also consider countable spaces, where an outcome may be any natural number. Here we review the definitions of probability spaces.

0.8. Definition. A **discrete probability space** is a finite or countable set S with a function $\mathbb{P}$ defined on the subsets of S (called **events**) such that
a) If $A \subseteq S$, then $0 \le \mathbb{P}(A) \le 1$,
b) $\mathbb{P}(S) = 1$, and
c) If $A_1, A_2, \ldots$ are pairwise disjoint subsets of S, then

$$\mathbb{P}\left(\bigcup A_i\right) = \sum_{i=1}^{\infty} \mathbb{P}(A_i).$$

0.9. Remark. For a finite probability space, assuming $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B)$ when $A \cap B = \varnothing$ and applying induction on k yields $\mathbb{P}(A) = \sum_{i=1}^{k} \mathbb{P}(B_i)$ when $B_1, \ldots, B_k$ is a partition of A. This follows from (c) above but does not imply it, so we require the more general condition.

More general definitions of probability space allow the probability function to be defined only on subsets of S with certain properties, but the simple definition above suffices for our purposes. On the rare occasions where we mention continuous probability spaces, we will be informal (that is, non-rigorous). For example, when choosing a point at random from a region in the plane, we adopt the intuitive notion that the probability it lies in a particular subregion is proportional to the area, with "regions" simple enough not to worry about measurability.

Immediate consequences of Definition 0.8 include
a) $\mathbb{P}(\varnothing) = 0$,
b) $\mathbb{P}(\overline{A}) = 1 - \mathbb{P}(A)$, and
c) $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$.
Furthermore, $\mathbb{P}(A) = \sum_{a \in A} \mathbb{P}(a)$, writing $\mathbb{P}(a)$ for $\mathbb{P}(\{a\})$ when $a \in S$. Elements of a probability space S are **sample points** or **outcomes** (of an "experiment"). ■

0.10. Definition. Events A and B in a probability space are **independent** if $\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$. For events A and B with $\mathbb{P}(B) \neq 0$, the **conditional probability** *of A given B*, written $\mathbb{P}(A \mid B)$, is defined to be $\mathbb{P}(A \cap B)/\mathbb{P}(B)$.

0.11. Remark. Saying that a probability space is generated by making choices "independently" means that the space is a cartesian product. The probability of an outcome is the product of the probabilities of its coordinates in the factors. For example, when flipping a coin n times independently, each flip has outcome head or tail, each with probability 1/2. The probability of a given list is 2^{-n}.

Making a choice **uniformly at random** means that the possible outcomes are equally likely. When flipping a coin, the flips are generated uniformly at random (probability 1/2 of each outcome) and independently.

Studying conditional probability has the effect of normalizing or restricting the space to the points within the event B. The intuitive idea is that the conditional probability (given that B occurs) is the fraction of B where A also occurs.

The probability of a joint event can be computed as a product of conditional probabilities. For example,

$$\mathbb{P}(\textstyle\bigcap_{k=1}^{n} A_i) = \mathbb{P}(A_n \mid \bigcap_{k=1}^{n-1} A_i)\, \mathbb{P}(A_{n-1} \mid \bigcap_{k=1}^{n-2} A_i) \cdots \mathbb{P}(A_2 \mid A_1)\, \mathbb{P}(A_1).$$ ■

In the last half of the 20th century, advanced techniques for studying probability spaces found many applications to difficult combinatorial problems. Combinatorial techniques show that a "good enough" object exists by constructing it, but probabilistic methods are nonconstructive. An object with a desired property must exist when the probability of that property is nonzero in an appropriate probability space. Similarly, one can show the existence of an object with a large value of a parameter X by showing that the expected value of X is that large when the objects are randomly generated.

0.12. Definition. A **random variable** on a discrete probability space S is a function $X\colon S \to \mathbb{R}$. It is **discrete** when the range is finite or countable, often $\mathbb{N}_0$. Let $X = k$ denote the event $\{a \in S\colon X(a) = k\}$, and write $\mathbb{P}(X = k)$ for its probability. The **expectation** or **expected value** $\mathbb{E}(X)$ of X is $\sum_{a \in S} X(a)\mathbb{P}(a)$, when this sum converges. When X is a discrete random variable, we write this as $\mathbb{E}(X) = \sum_{k=0}^{\infty} k \cdot \mathbb{P}(X = k)$. The **pigeonhole property** of the expectation is the statement that there is an element of the probability space for which the value of X is as large as (or as small as) $\mathbb{E}(X)$.

Using the pigeonhole property requires a value or bound for $\mathbb{E}(X)$. Often the computation applies the "linearity of expectation" to an expression for X as a sum of simpler random variables. We restrict our attention to sums of finitely

many random variables on discrete probability spaces. Analogous results hold in continuous probability spaces.

0.13. Lemma. (**Linearity of Expectation**) If X and $X_1, \dots, X_k$ are random variables on the same space such that $X = \sum X_i$, then $\mathbb{E}(X) = \sum \mathbb{E}(X_i)$. Also $\mathbb{E}(cX) = c\mathbb{E}(X)$ for any constant c.

Proof: In a discrete probability space, each sample point contributes the same amount to each side of each of these equations. ■

One reason for the influence of probabilistic methods is that exact counts are both too difficult and unnecessary in large structures. Somehow the probabilistic methods capture the most important aspects or most dominant terms. Probabilistic methods are especially effective for extremal problems when it turns out that "most" instances are near the optimum.

OTHER DISCRETE STRUCTURES

In addition to subsets, permutations, and graphs, many other structures are used to model combinatorial problems. We briefly describe several studied in this book in order to suggest the scope of the text. The reader should treat this section lightly; precise definitions and examples will be given later.

0.14. Example. *Digraphs and multigraphs.* General binary relations are modeled using **directed graphs** (**digraphs**); these differ from graphs in that the edges are *ordered* pairs of vertices. In an edge *from x to y*, the first vertex is the **tail** and the second is the **head**. We illustrate directed graphs by drawings that use arrows (curves with direction) for the edges.

We sometimes allow edges in digraphs to be pairs of the form (x, x), called **loops**. For example, the **functional digraph** of a function $f\colon A \to A$ has vertex set A; its edges are the pairs $(x, f(x))$ for $x \in A$. The fixed points of f become loops in the functional digraph. A permutation is a bijection $\sigma\colon A \to A$; its functional digraph consists of disjoint (directed) cycles corresponding to its orbits.

We may modify a graph or digraph both by allowing loops (one-vertex edges) and by allowing more than one edge with the same endpoints. Here $E(G)$ becomes a multiset. The resulting model is a **multigraph** or **multidigraph**. ■

0.15. Example. *Order relations.* The containment relation on a family of sets is a reflexive relation that is transitive but not symmetric and hence is not an equivalence relation. It is a fundamental example of another important type of relation. A relation R is **antisymmetric** if $(x, y) \in R$ and $(y, x) \in R$ imply $x = y$. A relation is an **order relation** if it is reflexive, antisymmetric, and transitive.

Besides containment, other order relations include the divisibility relation on the set of positive integers and the componentwise order on $\mathbb{R}^n$, which contains the pair (u, v) when $u_i \le v_i$ for all i. Other examples arise in scheduling: events occupy some interval in time, and we say that A "precedes" B if A ends before B begins. This defines an order relation on any set of intervals. We study order relations in Part III. ■

0.16. Example. *Hypergraphs.* Another generalization of graphs allows edges of arbitrary size. A **hypergraph** G consists of a set $V(G)$ of vertices and a set $E(G)$ of edges, where an edge can be any subset of $V(G)$. A **k-uniform hypergraph** is a hypergraph whose edges all have size k; graphs are simply 2-uniform hypergraphs. A hypergraph is **regular** if every vertex is in the same number of edges. Since the edge set of a hypergraph with vertex set X is a family of subsets of X, hypergraphs can be studied from set-theoretic, order-theoretic, and graph-theoretic viewpoints. They will be helpful in Parts III and IV. ■

0.17. Example. *Designs and projective planes.* In Chapter 13 we study a special type of regular uniform hypergraph that has applications to design of experiments and to extremal combinatorial problems. A **block design** is a regular uniform hypergraph in which any two points (vertices) appear together in the same number of blocks (edges). Equivalently, any two rows in the incidence matrix for the membership relation of elements in blocks have the same dot product.

When any two points lie in exactly one common block, we may request also that any two blocks have exactly one common point. The resulting configurations are **projective planes**, in which the blocks are called **lines**. Projective planes can be obtained from finite fields and correspond to special families of latin squares. A **Latin square** of order n is an arrangement of symbols of n types in an n-by-n matrix so that each symbol appears exactly once in each row and in each column. A classical application is the assignment of types of fertilizers and seeds to regions in an agricultural plot to reduce the effect of soil differences. ■

0.18. Example. *Matroids.* In Chapter 11 we study another special structure. Matroids can be viewed as special families of subsets of a set. This interprets them as a special type of hypergraph. We postpone the precise definition and observe merely that the matroid context permits common generalizations of fundamental results in graph theory, linear algebra, and the theory of ordered sets. For example, matroids permit a natural generalization of the result that a spanning tree of minimum total weight in a connected graph with weighted edges can be found by iteratively including the cheapest edge that does not form a cycle with edges already chosen. Much of the elementary theory of dimension in linear algebra also arises as a special case of matroid properties. ■

COMPLEXITY

The growth of combinatorics has been stimulated by computer science, which studies the computational aspects of discrete mathematics. We will comment occasionally on the computational complexity of problems. A simple measure of the performance of an algorithm is its worst-case running time, as a function of the size of the input. A problem is *efficiently solved* if it has a solution algorithm whose running time is bounded by a polynomial in the size of the input.

The *size* of the input is its length in bits in some encoding of the problem. For our purposes, natural parameters such as the order of a matrix or the number of vertices suffice to measure size. A polynomial in n is bounded by a polynomial in

n^2 or n^3, so the manner of encoding input is unimportant unless the problem has exponentially large numbers as input data.

Complexity considers asymptotic growth rates. The set of functions whose magnitude is bounded above by a constant multiple of f (for sufficiently large arguments) is called $O(f)$. Several pertinent sets of functions arise when comparing growth rates to f, as listed below.

$$o(f) = \{g\colon |g(x)|/|f(x)| \to 0\}$$
$$O(f) = \{g\colon \exists c, a \in \mathbb{R} \text{ such that } |g(x)| \le |cf(x)| \text{ for } x > a\}$$
$$\Omega(f) = \{g\colon \exists c, a \in \mathbb{R} \text{ such that } |g(x)| \ge |cf(x)| \text{ for } x > a\}$$
$$\omega(f) = \{g\colon |g(x)|/|f(x)| \to \infty\}$$
$$\Theta(f) = O(f) \cap \Omega(f)$$

Derbyshire [2003] attributed "Big Oh notation" to Landau [1909], but Landau [1909, p. 883] borrowed it from Bachmann [1894].

Properly speaking, $o(f)$, $O(f)$, $\Omega(f)$, $\omega(f)$, and $\Theta(f)$ are sets, and it is correct to write $g \in O(f)$ when describing the growth rate of g. Mathematicians and computer scientists routinely write $g(n) = O(f(n))$ to mean $g \in O(f)$ (see Knuth [1976a]). Some avoid this by writing "$g(n)$ is $O(f(n))$", treating $O(f)$ as an adjective. We can compute with members of $O(f)$ somewhat as we do with congruence classes, but we have no symbol like "$\equiv$" for doing this. Thus in this book we sometimes use expressions like $f(n) = n^2 + O(n^{3/2})$ (meaning $f(n) - n^2 \in O(n^{3/2})$), but where the grammar is appropriate we use the membership symbol. The expression $g(n) \sim f(n)$ means that g is **asymptotic** to f, which can be written as $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 1$ or $g(n) = f(n)(1 + o(1))$.

Complexity classes are studied using **decision problems** that have yes/no answers, such as "does the input graph have a spanning cycle?" Optimization problems (such as "what is the maximum length of a cycle in this graph?") can be solved using successive decision problems, such as "does this graph have a cycle of length at least k?", where k is part of the input. The class of decision problems solvable by a worst-case polynomial-time algorithm (polynomial in the size of the input) is called "P".

Many decision problems have no known polynomial-time solution algorithm but have a polynomial-time algorithm for verifying a YES answer. For example, existence of a spanning path can be verified by giving the order of vertices on such a path and checking that successive vertices are adjacent. When checking all possible permutations in parallel, each computation path is short. It is verifying a NO answer that is difficult.

A **deterministic algorithm** follows only one computation path on a given input. A **nondeterministic algorithm** follows multiple computation paths simultaneously. In the example above, given an input graph, such an algorithm checks all possible vertex orderings simultaneously to seek a spanning path; each ordering can be checked in polynomial time. A **nondeterministic polynomial-time** algorithm follows one computation path for each way of specifying a polynomial-length stream of bits, with each such computation running in polynomial time. The bit stream is not the input to the problem; the bits specify an option to consider, which in the example here is a vertex ordering.

A nondeterministic algorithm *solves* a decision problem if for every input I, the answer to the problem on I is YES if and only if the algorithm applied to I has