Arnold L. Rosenberg, Denis Trystram

Understand Mathematics, Understand Computing

Discrete Mathematics that All Computing Students Should Know

February 1, 2018

Springer

Understand Mathematics, Understand Computing

Discrete Mathematics that All Computing Students Should Know

Arnold L. Rosenberg
University of Massachusetts
Amherst, MA 01003, USA
rsnbrg@cs.umass.edu

Denis Trystram
University of Grenoble
Grenoble, FRANCE
denis.trystram@imag.fr

Contents

Chapter 1
INTRODUCTION

1.1 Overview

In the early days of computing, all aspects of the field were considered the domain of the "techies"—the engineers and scientists and mathematicians who designed the early computers and figured out how to use them to solve a range of problems that is expanding even to this day. Back then, one expected every computer-oriented professional to have a mastery of many mathematical topics. Times—and the field of computing—have changed: classical computing curricula, which have traditional led to a BS degree within a school of science or engineering have been joined by curricula that lead to a BA degree or a degree in IT or in business or .... Within this new world, many aspiring computer professionals arguably need little knowledge of mathematics. However, many argue that the computing field suffers for this lack of mathematics, which has weakened the ability of many practitioners to design and perform experiments, to analyze their results, and to formulate well-reasoned conclusions based on these results. This situation has denied computer science the popular confidence enjoyed by other empirical disciplines; indeed, many would question the math-deprived practitioners' ability to reason rigorously about basic computational phenomena. Yet others, however, argue that such "scientific" acumen is not needed by practitioners in many of the newer segments of the computing field.

The preceding discussion is important to computing educators because we serve a large population of students, with quite diverse needs and aspirations. We strive to keep this diversity in mind within this essay. As we discuss a range of mathematical topics for possible inclusion as prerequisites for the undergraduate study of computing, we try to indicate why the selected topics are needed. Readers can then evaluate which topics are needed for students enrolled in their computing program.

1.2 The Elements of Rigorous Reasoning

1.2.1 Basic Reasoning

Distinguishing name from object. A fundamental stumbling block in the road to cogent reasoning arises from the inability to distinguish names from the objects they denote. A prime example within the world of computing resides in the inability to distinguish a function (which can be viewed as an infinite set of argument-value pairs) from a program, which can be viewed as a name for the function. **Note** that the often-used view of a function as a *rule* for assigning values to arguments should be avoided, because it suggests – *erroneously* – that an implementable such rule always exists!

Quantitative reasoning. Students should understand the foundational distinction between "growing without bound" and being innite. Within this theme, they should appreciate situations such as the following. Every integer, and every polynomial with integer coefficients, is finite, but there are infinitely many integers and infinitely many polynomials. Students should be able to verify (cogently but not necessarily via any particular formalism) assertions such as the following.

- Let us be given polynomials $p(x)$ of degree a and $q(x)$ of degree $b > a$, where a, b need not be integers. There must exist a constant $X_{p,q}$ (i.e., a constant that depends on the properties of polynomials p and q) such that for all $x > X_{p,q}$, $p(x) < q(x)$.
  Thus, polynomials having bigger degrees eventually *majorize*—i.e., have larger values than—polynomials having smaller degrees.
- Continuing with polynomial q of degree b: For any real number $c > 1$, there exists a constant $Y_{c;q}$ (i.e., a constant that depends on the properties of polynomial q and constant c) such that for all $x > Y_{c;q}$, $c^x > q(x)$.
  Thus, exponential functions eventually *majorize* polynomials.

1.2.1.1 The Elements of Formal Reasoning

induction

proof by contradiction

1.2.1.2 The Elements of Empirical Reasoning

Empirical reasoning does not convey the certitude that formal reasoning does.

1.3 Our Approach to Mathematical Preliminaries

"If your only tool is a hammer . . . "

We now review a broad range of mathematical concepts that are central to the study and practice of CS/CE. As we develop these concepts, we shall repeatedly observe instances of the following "self-evident truth" (which is what "axiom" means).

> **The conceptual axiom**. *One's ability to think deeply about a complicated concept is always enhanced by having more than one way to think about the concept.*

Chapter 2
SETS, BOOLEAN ALGEBRA, AND LOGIC

2.1 Sets

Sets are probably the most basic object of mathematical discourse. We assume, therefore, that *the reader knows what a set is and recognizes that some sets are finite, while others are infinite*. Sample finite sets are, for example, the set of words in this essay or the set of characters in any JAVA program. Some familiar infinite sets are:

- the set of *nonnegative integers*;
- the set of *positive integers*;
- the set of *all integers*;
- the set of nonnegative *rational numbers*—which are quotients of integers;
- the set of nonnegative *real numbers*—which can be viewed computationally as the set of numbers that admit infinite decimal expansions,
- the set of nonnegative *complex numbers*—which can be viewed as ordered pairs of real numbers,
- the set of *all* finite-length binary strings.

When discussing computer-related matters, one often calls each 0 and 1 that occurs in a binary string a *bit* (for *binary digit*), which leads to the term *"bit string"* as a synonym of "binary string." Despite this assumption, we begin the essay by reviewing some basic concepts concerning sets and operations thereon.

For any finite set S, we denote by $|S|$ the *cardinality* of S, which is the number of elements in S. Finite sets having three special cardinalities are singled out with special names. If $|S| = 0$, then we call S the *empty set* and denote it $\emptyset$. (The empty set is often a limiting case of set-defined entities.) If $|S| = 1$, then we call S a *singleton*; and if $|S| = 2$, then we call S a *doubleton*. In many set-related domains, the sets of interest will be subsets of some fixed "universal" set U.

> We use the term "universal" as in "universe of discourse," not in the self-referencing sense of a set that contains all other sets, a construct (discussed by Bertrand Russell) which leads to mind-bending paradoxes.

Given a universal set U and a *subset* $S \subseteq U$ (the notation meaning that every element of S—if there are any—is also an element of U), we observe the inequalities

$$\emptyset \subseteq S \subseteq U.$$

It is often useful to have a convenient term and notation for *the set of all subsets of a set S*. This bigger set—it contains $2^{|S|}$ elements when S is finite—is denoted by $\mathscr{P}(S)$ and is called the *power set* of S.[1] You should satisfy yourself that the biggest and smallest elements of $\mathscr{P}(S)$ are, respectively, the set S itself and the empty set $\emptyset$.

Given two sets S and T, we denote by:

- $S \times T$ the *direct product* of S and T, which is the set of all ordered pairs whose first coordinate contains an element of S and whose second coordinate contains an element of T.
- $S \cap T$ the *intersection* of S and T, which is the set of elements that occur in *both* S and T.
- $S \cup T$ the *union* of S and T, which is the set of elements that occur in S, or in T, *or in both*. (Because of the "or both" qualifier, this operation is sometimes called *inclusive union*.)
- $S \setminus T$ is the *difference* of S and T, which is the set of elements that occur in S but not in T. (Particularly in the United States, one often finds "$S - T$" instead of "$S \setminus T$.")

When studying a context that subsumes a universal set U that contains all other sets, we include also the operation

- $\overline{T} = U \setminus T$, the *complement* of T (relative to the universal set U).
  For instance, the set of odd positive integers is the complement of the set of even positive integers, relative to the set of all positive integers.

We note a number of basic identities involving sets and operations on them.

- $S \setminus T = S \cap \overline{T}$,
- If $S \subseteq T$, then
  1. $S \setminus T = \emptyset$,
  2. $S \cap T = S$,
  3. $S \cup T = T$.

Note, in particular, that[2]

$$[S = T] \text{ iff } \Big[[S \subseteq T] \text{ and } [T \subseteq S]\Big] \text{ iff } \Big[(S \setminus T) \cup (T \setminus S) = \emptyset\Big].$$

The operations union, intersection, and complementation—and operations formed from them, such as set difference—are usually called the *Boolean (set) operations*,

[1] The name "power set" arises from the relative cardinalities of S and $\mathscr{P}(S)$ for finite S.

[2] "iff" abbreviates the common mathematical phrase, "if and only if."

(named for the 19th-century English mathematician George Boole. There are several important identities involving the Boolean set operations. Among the most frequently invoked are the two "laws" attributed to the 19th-century French mathematician Auguste De Morgan:

$$\text{For all sets } S \text{ and } T\text{:} \quad \begin{cases} \overline{S \cup T} = \overline{S} \cap \overline{T}, \\ \\ \overline{S \cap T} = \overline{S} \cup \overline{T}. \end{cases} \tag{2.1}$$

While we have focused thus far on Boolean operations on *sets*, there are "logical" analogues of these operations for logical sentences and their logical "truth values" 0 and 1:

- The logical analogue of complementation is (logical) not, which we denote by an overline;[3] e.g., $[\overline{0} = 1]$, and $[\overline{1} = 0]$.
- The logical analogue of union is (logical) or, which is also called *disjunction* or *logical sum*. Texts often denote "or" in expressions by "$\vee$"; e.g., $[X \vee Y = 1]$ iff $[X = 1]$ or $[Y = 1]$ or both.
- The logical analogue of intersection is (logical) and, which is also called *conjunction* or *logical product*. Texts often denote "and" in expressions by "$\wedge$"; e.g., $[X \wedge Y = 1]$ iff both $[X = 1]$ and $[Y = 1]$

We end this section with a set-theoretic definition that occurs in many contexts. Let $\mathscr{C}$ be any (finite or infinite) collection of sets, and let S and T be two elements of $\mathscr{C}$. (Note that $\mathscr{C}$ is a set whose elements are sets.) Think, e.g., of the concrete example of set intersection. We say that $\mathscr{C}$ is *closed* under intersection if whenever sets S and T (which could be the same set) both belong to $\mathscr{C}$, the set $S \cap T$ also belongs to $\mathscr{C}$. By De Morgan's laws, $\mathscr{C}$'s closure under union implies also its closure under intersection.

2.2 Binary Relations

2.2.1 The Formal Notion of Binary Relation

Given sets S and T, a *relation on S and T* (in that order) is any subset

$$R \subseteq S \times T.$$

When $S = T$, we often call R a *binary relation on (the set) S* ("*binary*" because there are *two* sets being related). Relations are so common that we use them in every aspect of our lives without even noticing them. The relations "equal," "less than," and "greater than or equal to" are simple examples of binary relations on the

[3] Context will always make it clear when we are talking about set complementation and when we are talking about logical not.

integers. These same three relations apply also to other familiar number systems such as the rational and real numbers; only "equal," though, holds (in the natural way) for the complex numbers. Some subset of the three relations "is a parent of," "is a child of," and "is a sibling of" probably are binary relations on (the set of people constituting) your family. To mention just one relation with distinct sets S and T, the relation "A is taking course X" is a relation on

$$(\text{the set of all students}) \times (\text{the set of all courses}).$$

By convention, with a binary relation $R \subseteq S \times T$, we often write "sRt" in place of the more conservative "$\langle s,t \rangle \in R$." For instance, in "real life," we write "$5 < 7$" rather than the strange-looking (but formally correct) "$\langle 5,7 \rangle \in <$."

The following operation on relations occurs in many guises, in almost all mathematical theories. Let P and P' be binary relations on a set S. The *composition* of P and P' (in that order) is the relation

$$P'' \overset{\text{def}}{=} \Big\{ \langle s,u \rangle \in S \times S \mid (\exists t \in S) \Big[[sPt] \text{ and } [tP'u] \Big] \Big\}.$$

(Note how we have used both of our notational conventions for relations here. Note also our use of a common "shorthand" compound symbol "$\overset{\text{def}}{=}$": The sentence "$X \overset{\text{def}}{=} Y$" should be read "$X$ is, by definition, Y.")

There are two special classes of binary relations that are so important that we must single them out immediately, in the next two subsections.

2.2.2 Equivalence Relations

A binary relation R on a set S is an *equivalence relation* if it enjoys the following three properties:

1. R is *reflexive:* for all $s \in S$, we have sRs.
2. R is *symmetric:* for all $s, s' \in S$, we have sRs' whenever $s'Rs$.
3. R is *transitive:* for all $s, s', s'' \in S$, whenever we have sRs' and $s'Rs''$, we also have sRs''.

Sample familiar equivalence relations are:

- The equality relation, $=$, on a set S which relates each $s \in S$ with itself but with no other element of S.
- The relations $\equiv_{12}$ and $\equiv_{24}$ on integers, where[4]
  1. $n_1 \equiv_{12} n_2$ if and only if $|n_1 - n_2|$ is divisible by 12.
  2. $n_1 \equiv_{24} n_2$ if and only if $|n_1 - n_2|$ is divisible by 24.

[4] As usual, $|x|$ is the *absolute value*, or, *magnitude* of the number x. That is, if $x \geq 0$, then $|x| = x$; if $x < 0$, then $|x| = -x$.

We use relation $\equiv_{12}$ (without formally knowing it) whenever we tell time using a 12-hour clock and relation $\equiv_{24}$ whenever we tell time using a 24-hour clock.

Closely related to the notion of an equivalence relation on a set S is the notion of a *partition* of S. A partition of S is a nonempty collection of subsets $S_1, S_2, \ldots$ of S that are

1. *mutually exclusive:* for distinct indices i and j, $S_i \cap S_j = \emptyset$;
2. *collectively exhaustive:* $S_1 \cup S_2 \cup \cdots = S$.

We call each set S_i a *block* of the partition.

One verifies easily that *a partition of a set S and an equivalence relation on S are just two ways of looking at the same concept*.

The *index* of the equivalence relation R is its number of classes—which can be finite or infinite.

Let[5] $\equiv_1$ and $\equiv_2$ be two equivalence relations on a set S. We say that the relation $\equiv_1$ *is a refinement of* (or, *refines*) the relation $\equiv_2$ just when each block of $\equiv_1$ is a subset of some block of $\equiv_2$. A couple of basic facts:

- The equality relation, $=$, on S refines every equivalence relation on S. (In this sense, it is the finest equivalence relation on S.)
- Say that the equivalence relation $\equiv_1$ refines the equivalence relation $\equiv_2$ and that $\equiv_2$ has finite index I_2. Then either $\equiv_1$ also has finite index $I_1 \geq I_2$, or $\equiv_1$ has infinite index.

2.2.3 Functions

One learns early in school that a function from a set A to a set B is a rule that assigns a unique value from B to every value from A. Simple examples illustrate that this notion of function is more restrictive than necessary. Think, e.g., of the operation *division* on integers. We learn that division, like multiplication, is a function that assigns a number to a given pair of numbers. Yet we are warned almost immediately not to "divide by 0": The quotient upon division by 0 is "undefined." So, division is not quite a function as envisioned our initial definition of the notion. Indeed, in contrast to an expression such as "$4 \div 2$," which should lead to the result 2 in any programming environment,[6] expressions such as "$4 \div 0$" will lead to wildly different results in different programming environments. Since "wildly different" is anathema in any mathematical setting, we deal with situations such as just described by broadening the definition of "function" in a way that behaves like our initial simple definition under "well-behaved" circumstances and that extends the notion in an intellectually consistent way under "ill-behaved" circumstances. Let us begin to get formal.

[5] Conforming to common usage, we typically use the symbol $\equiv$, possibly with an embellishing subscript, to denote an equivalence relation.

[6] We are, of course, ignoring demons such as round-off error.

A *(partial) function from set S to set T* is a relation $F \subseteq S \times T$ that is *single-valued;* i.e., for each $s \in S$, there is *at most* one $t \in T$ such that sFt. We traditionally write "$F : S \to T$" as shorthand for the assertion, "F is a function from the set S to the set T"; we also traditionally write "$F(s) = t$" for the more conservative "sFt." (The single-valuedness of F makes the nonconservative notation safe.) We often call the set S the *source (set)* and T the *target (set)* for function F. When there is always a (perforce, unique) $t \in T$ for each $s \in S$, then we call F a *total* function.

You may be surprisedf to encounter functions that are not total, because most of the functions you deal with daily are *total*. Our mathematical ancestors had to do some fancy footwork in order to make your world so neat. Their choreography took two complementary forms.

1. They expanded the target set T on numerous occasions. As just two instances:
   - They appended both 0 and the negative integers to the preexisting positive integers[7] in order to make subtraction a total function.
   - They appended the rationals to the preexisting integers in order to make division (by nonzero numbers!) a total function.

   The irrational algebraic numbers, the nonalgebraic real numbers, and the nonreal complex numbers were similarly appended, in turn, to our number system in order to make certain (more complicated) functions total.
2. They adapted the function. In programming languages, in particular, true undefinedness is anathema, so such languages typically have ways of making functions total, via devices such as "integer division" (so that odd integers can be "divided by 2") as well as various ploys for accommodating "division by 0."

The (20th-century) inventors of *Computation Theory* insisted on a theory of functions on nonnegative integers (or some transparent encoding thereof). The price for such "pureness" is that we must allow functions to be undefined on some arguments. Thus the theory renders such functions as "division by 2" and "taking square roots" as being *nontotal*: both are defined only on subsets of the positive integers (the even integers and the perfect squares, respectively).

Three special classes of functions merit explicit mention. For each, we give both a down-to-earth name and a more scholarly Latinate one.

A function $F : S \to T$ is:

1. *one-to-one* (or *injective*) if for each $t \in T$, there is at most one $s \in S$ such that $F(s) = t$;
   An injective function F is called an *injection*.
2. *onto* (or *surjective*) if for each $t \in T$, there is at least one $s \in S$ such that $F(s) = t$;
   A surjective function F is called a *surjection*.
3. *one-to-one, onto* (or *bijective*) if for each $t \in T$, there is precisely one $s \in S$ such that $F(s) = t$.
   A bijective function F is called a *bijection*.

[7] The great mathematician Leopold Kronecker said, "God made the integers, all else is the work of man"; Kronecker was referring, of course, to the *positive* integers.

2.3 The Elements of Mathematical Logic

2.3.1 The Fundamental Logical Connectives

2.3.2 Connecting Mathematical Logic with Logical Reasoning

Converse
  Contrapositive
  Proof by contradiction

Chapter 3
ARITHMETIC

3.1 Basic Numerical and Algebraic Concepts and Their Manipulations

constructs such as:

3.1.1 Powers and polynomials

Powers, including fractional ones; examples: x^2, $n^{3/2}$, $\sqrt{p}$. The student's familiarity should include facility with facts having to do with reciprocity and functional inverses, as in:

$$x^{-a} = \frac{1}{x^a}; \;\; (x^a)^{1/a} = x$$

Polynomials and their associated notions, such as degrees and coefficients, and computations therewith, including polynomial summation and multiplication (the latter being essential, e.g., when discussing a range of topics relating to, say, fault tolerance and encryption).

3.1.2 Exponentials and logarithms

This includes a discussion of bases and facility with the fact that exponential and logarithmic functions are inverse to one another, in the following sense. Given any integer $b > 1$ (for "base"), the *base-b logarithm* function $\log_b(\bullet)$ maps positive reals to reals and is defined by either of the following mutually inverse relations:

$$(\forall x > 0)[x \; = \; b^{\log_b x} \; = \; \log_b b^x]. \tag{3.1}$$

Taking logarithms is, thus, inverse to exponentiating. When $b = 2$, a particularly common special case within computation theory, we usually elide the base 2 and just write $\log x$, or, commonly, $\ln x$.

**HERE

The exponential and logarithmic functions within the context of information: The student should recognize and be able to reason about the following facts. If one has an alphabet A of a symbols and must provide distinct string-label "names" for n items, then at least one string-name must have length no shorter than $\lceil \log_a n \rceil$. Viewed contrapositively, the number of distinct strings of length k over alphabet A is a^k.

**H

3.1.3 Arithmetic and geometric sequences and series

The ability to sum – and perhaps approximate – simple series, including, *at least*, finite arithmetic series and both finite and infinite geometric series.
Arithmetic sequences and series.

$$\begin{array}{l}
\text{An } n\text{-term arithmetic sequence:} \\
\quad a,\ a+b,\ a+2b,\ a+3b,\ \ldots, a+(n-1)b \\
\\
\text{The corresponding arithmetic series:} \\
\quad a+(a+b)+(a+2b)+(a+3b)+\cdots+(a+(n-1)b) \\
\qquad = \ an+b\cdot(1+2+\cdots+n-1)
\end{array} \tag{3.2}$$

We can, thus, sum the arithmetic series in (3.2) by determining the sum of the first m positive integers; $m = n-1$ in (3.2). We accomplish this via a device known to Karl Friedrich Gauss as a pre-teen. Let S_m denote the desired sum.

$$\begin{array}{l}
\text{Write the sum } S_m \text{ "forward":} \\
\qquad S_m = 1 \ + \quad 2 \quad + \cdots + \ (m-1) \ + \ m \\
\text{Write } S_m \text{ in reverse:} \\
\qquad S_m = m \ + \ (m-1) \ + \cdots + \quad 2 \quad + \ 1
\end{array} \tag{3.3}$$

Now add the two versions of S_m in (3.3) *columnwise*. Because each column-sum equals $m+1$, we find that $2S_m = m(m+1)$, so that

$$S_m \ = \ 1+2+\cdots+(m-1)+m \ = \ \frac{1}{2}m(m+1) \ \stackrel{\text{def}}{=} \ \binom{m+1}{2}. \tag{3.4}$$

It follows that our original series in (3.2) sums as follows.

$$a+(a+b)+(a+2b)+(a+3b)+\cdots+(a+(n-1)b) \ = \ an+b\cdot\binom{n}{2}.$$

Geometric sequences and series.

$$\begin{array}{l} \text{An } n\text{-term geometric sequence:} \\ \quad a,\ ab,\ ab^2,\ \ldots, ab^{n-1} \\ \text{The corresponding geometric series:} \\ \quad a + ab + ab^2 + \cdots + ab^{n-1} \ = \ a(1 + b + b^2 + \cdots + b^{n-1}) \end{array} \tag{3.5}$$

Easily, we can sum the series in (3.5) by summing just the sub-series

$$S_b(n) \ \stackrel{\text{def}}{=} \ 1 + b + b^2 + \cdots + b^{n-1}. \tag{3.6}$$

We proceed as follows. Write $S_b(n)$ so that its terms are in *decreasing* order. We thereby isolate two cases.

1. Say first that $b > 1$. In this case, we write the series in the form

   $$S_b^{b>1}(n) \ = \ b^{n-1} + b^{n-2} + \cdots + b^2 + b + 1,$$

   and we note that

   $$S_b^{b>1}(n) \ = \ b^{n-1} \ + \ \frac{1}{b} \cdot S_b^{b>1}(n) \ - \ \frac{1}{b}.$$

   In other words, we have

   $$\left(1 - \frac{1}{b}\right) S_b^{b>1}(n) \ = \ b^{n-1} - \frac{1}{b},$$

   or equivalently,

   $$S_b^{b>1}(n) \ = \ \frac{b^n - 1}{b - 1}.$$

2. Alternatively, if $b < 1$, then we write the series in the form

   $$S_b^{b<1}(n) \ = \ 1 + b + b^2 + b^3 + \cdots + b^{n-1}.$$

   and we note that

   $$S_b^{b<1}(n) \ = \ 1 \ + \ b \cdot S_b^{b<1}(n) \ - \ b^n.$$

   In other words,

   $$(1 - b) S_b^{b<1}(n) \ = \ 1 \ - \ b^n$$

   or equivalently,

   $$S_b^{b>1}(n) \ = \ \frac{1 - b^n}{1 - b}.$$

Note that $S_b^{b>1}(n)$ and $S_b^{b<1}(n)$ actually have the same form. We have chosen to write them differently to stress their *approximate* values, which are useful in "back-of-the-envelope" calculations: For very large values of n, we have

$$S_b^{b>1}(n) \;\approx\; \frac{b^n}{b-1} \quad \text{while} \quad S_b^{b<1}(n) \;\approx\; \frac{1}{1-b}.$$

3.1.4 Linear recurrences

With the preceding list of topics, a student has access to what is called "The Master Theorem for Linear Recurrences" in the widely known Cormen-Leiserson-Rivest-Stein text on algorithm design and analysis. This level of mathematical preparation should be adequate for most early-undergrad courses on data structures and algorithms.

3.2 Elementary counting

within discrete frameworks, including introducing discrete probability/likelihood as a ratio:

$$\frac{\text{number of targeted events}}{\text{number of possible events}}$$

3.3 Congruences and Modular Arithmetic

3.4 Numbers and Numerals

3.4.1 Number vs. Numeral: Object vs. Name

3.4.2 Geometric series and positional number systems

The relation between simple geometric series and numeration within positional number systems – including changing bases in such systems.

3.5 Useful Nonalgebraic Notions

3.5.1 Nonalgebraic Notions Involving Numbers

Floors and ceilings. Given any real number x, we denote by $\lfloor x \rfloor$ the *floor* (or *integer part*) of x, which is the largest integer that that does not exceed x. Symmetrically, we denote by $\lceil x \rceil$ the *ceiling* of x, which is the smallest integer that is at least as large as x. For any nonnegative integer n,

$$\lfloor n \rfloor \ = \ \lceil n \rceil \ = \ n;$$

for any positive rational number $n + p/q$, where n, p, and q are positive integers and $p < q$,

$$\lfloor n + p/q \rfloor \ = \ n, \ \text{ and } \ \lceil n + p/q \rceil \ = \ n + 1.$$

Absolute values, or, magnitudes Given any real number x, positive or negative, we denote by $|x|$ the *absolute value* or *magnitude* of x. If $x \geq 0$, then $|x| = x$; if $x < 0$, then $|x| = -x$.

If the intended curriculum will approach more sophisticated application areas such as robotics or data science or information retrieval or data mining (of course, at levels consistent with the students' preparation), then one would do well to insist on familiarity with notions such as:

3.6 Advanced Topics

3.6.1 Measures of distance in tuple-spaces

including the following norms/metrics: L_1 (Manhattan, or, rook's-move distance), L_2 (Euclidean distance); L_∞ (King's-move distance).

3.6.2 Edit-distance: a measure of closeness in string spaces

Chapter 4
GRAPH-THEORETIC ENTITIES

4.1 Basic Graph-Theoretic Topics

Graphs provide one of the richest technical and conceptual frameworks in the world of computing. They provide concrete representations of manifold data structures, hence must be well understood in preparation for a "Data Structures and Algorithms" course. They embody tangible abstractions of relationships of all sorts, hence must be well understood in order to discuss entities as varied as web-search engines and social networks with precision and rigor. As with most of the topics we are discussing, graph-oriented concepts must be taught "in layers". All students must be conversant with the use of graphs to represent and reason about a variety of complicated relationships, but the degree of sophistication that an individual student requires depends both on the abilities of the student and the range of applications that will appear in the student's program. The fundamental concepts in this essay should be understood by all students—although each can be developed with more texture and nuance within the context of specific application domains.

Many developments in computing technology over recent decades have made it imperative that graphs no longer be viewed by students as the static objects introduced, e.g., as abstractions of data struntures. Applications ranging from databases to web-search engines to social networks demand an appreciation of graphs as dynamic objects. This change in perspective affects many aspects of the mathematical prerequisites for any CS/CE program.

4.1.1 Fundamental Notions

The basic components of graphs are *nodes/vertices* (one encounters both terms in the literature) and the *edges* that interconnect them. When a graph is *undirected*, an edge connotes some sort of sibling-like relationship among nodes of "equal" status. When a graph is *directed* (sometimes referred to as a *digraph*), an edge connotes an "unequal" relationship such as parenthood or priority or dependence; edges in directed graphs are often termed *arcs*. In many situations involving directed graphs, it is important to recognize The *dual* of a digraph G is a digraph obtained by reversing all of G's arcs. One sometimes encounters situations when arguments about, or operations on, a digraph G can be "translated" to arguments about, or operations on, G's dual with only clerical effort. A *subgraph* G' of a graph G is a graph whose nodes are a subset of G's and whose edges are a subset of G's that interconnect only nodes of G'. A *path* in an undirected graph is a sequence of nodes within which every adjacent pair is connected by an edge. A path is a *cycle* if all nodes in the sequence are distinct, except for the first and last, which are identical. Paths and cycles in directed graphs are defined similarly, except that every adjacent pair of nodes must be connected by an arc, and all arcs must "point in the same direction."

Getting formal. A *directed graph* (*digraph*, for short) $\mathscr{G}$ is given by a set of *nodes* $\mathscr{N}_\mathscr{G}$ and a set of *arcs* (or *directed edges*) $\mathscr{A}_\mathscr{G}$. Each arc has the form $(u \rightarrow v)$, where $u, v \in \mathscr{N}_\mathscr{G}$; we say that this arc goes *from u to v*. A *path* in $\mathscr{G}$ is a sequence of arcs that share adjacent endpoints, as in the following path from node u_1 to node u_n:

$$(u_1 \rightarrow u_2),\ (u_2 \rightarrow u_3),\ \ldots,\ (u_{n-2} \rightarrow u_{n-1}),\ (u_{n-1} \rightarrow u_n). \tag{4.1}$$

It is sometimes useful to endow the arcs of a digraph with labels from an alphabet Σ. When so endowed, the path (4.1) would be written

$$(u_1 \overset{\lambda_1}{\rightarrow} u_2),\ (u_2 \overset{\lambda_2}{\rightarrow} u_3),\ \ldots,\ (u_{n-2} \overset{\lambda_{n-2}}{\rightarrow} u_{n-1}),\ (u_{n-1} \overset{\lambda_{n-1}}{\rightarrow} u_n),$$

where the λ_i denote symbols from Σ. If $u_1 = u_n$, then we call the preceding path a *cycle*.

An *undirected graph* is obtained from a directed graph by removing the directionality of the arcs; the thus-beheaded arcs are called *edges*. Whereas we say:

the *arc* (u, v) goes *from* node u *to* node v

we say:

the undirected edge $\{u, v\}$ goes *between* nodes u and v

or, more simply:

the undirected edge $\{u, v\}$*connects* nodes u and v.

Undirected graphs are usually the default concept, in the following sense: *When $\mathscr{G}$ is described as a "graph," with no qualifier "directed" or "undirected," it is understood that $\mathscr{G}$ is an undirected graph.*

4.1.2 Graph evolution and decomposition

Any course on algorithms will discuss graphs, especially trees, that evolve over time. Such structures arise in the study of "classical" algorithmic topics such minimum-spanning-tree and branch-and-bound algorithms, as well as in the study of more modern topics such as social networks and internetworks. For the most part, the mathematics already appearing in this essay suffices for these studies: the students must assimilate new algorithmic notions, not new mathematics. That said, students will be challenged to utilize the mathematical devices that they have (hopefully) mastered in new, more sophisticated, ways. Instructors who wish to lead their students to even a casual understanding of emerging modalities and platforms for computing are thereby challenged to teach required mathematical preliminaries using exemplars that include these new modalities and exemplars.

A fundamental variety of relevant notions within the study of graphs reside in the notions known in various contexts via terms such as *graph separators* or *graph bisection*. The key idea that underlies these notions is that certain graph-theoretic structures interconnect their graphs' constituent nodes more or less densely — and the type and level of interconnectivity has important algorithmic consequences. In such situations, the student must understand how the phenomenon/a modeled by the graphs of interest are elucidated by the way a graph can be broken into subgraphs by excising nodes or edges. When discussing communication-related structures, for instance, graph are often used to model the individual pairwise communications that must occur in order to accomplish the desired overall communication (such as a broadcast). There is often a provable tradeoff between the number of such pairwise communications and the overall time for the completion of the overall task. As another example, when discussing social networks, one can pose questions such as: which node in a network is best to connect to (when joining the network) in order to best facilitate one's interactions or influence within the community. The latter topic leads, e.g., to the study of *power-law* networks, a topic that would not be studied in depth in any early course; indeed, the structure of these networks is not yet well understood even in advanced settings.

4.1.3 Trees

Within the world of graphs, trees occupy a place of honor. In their undirected form, the are defined by the proerty of containing no cycles. They then provide the minimal interconnection structure that provides a path connecting every pair of nodes. Of course, trees appear also in directed form. Among directed trees, a very important subclass are those that have a *root*, i.e., a node that has a directed path to every other node of the tree. In applications, rooted trees represent hierarchical organizations (as do families when viewed generationally).

More formally: *rooted trees* are a class of *acyclic* digraphs. Paths in trees that start at the root are often called *branches*. The *acyclicity* of a tree $\mathscr{T}$ means that for any branch of $\mathscr{T}$ of the form (4.1), we cannot have $u_1 = u_n$, for this would create a cycle. Each rooted tree $\mathscr{T}$ has a designated *root node* $u_n \in \mathscr{N}_{\mathscr{T}}$ that resides at the end of a branch (4.1) that starts at $r_{\mathscr{T}}$ (so $u_1 = r_{\mathscr{T}}$) is said to reside at *depth* $n-1$ in $\mathscr{T}$; by convention, $r_{\mathscr{T}}$ is said to reside at depth 0. $\mathscr{T}$'s root $r_{\mathscr{T}}$ has some number (possibly 0) of arcs that go from $r_{\mathscr{T}}$ to its *children,* each of which thus resides at depth 1 in $\mathscr{T}$; in turn, each child has some number of arcs (possibly 0) to its children, and so on. (Think of a family tree.) For each arc $(u \to v) \in A_{\mathscr{T}}$, we call u a *parent* of v, and v a *child* of u, in $\mathscr{T}$; clearly, the depth of each child is one greater than the depth of its parent. Every node of $\mathscr{T}$ except for $r_{\mathscr{T}}$ has precisely one parent; $r_{\mathscr{T}}$ has no parents. A childless node of a tree is a *leaf*. The transitive extensions of the parent and child relations are, respectively, the *ancestor* and *descendant* relations. The *degree* of a node v in a tree is the number of children that the node has, call it c_v. If every nonleaf node in a tree has the same degree c, then we call c the *degree of the tree*.

4.2 Hypergraphs: an advanced topic

When studying certain computing-related topics, one will need a generalization of graphs called *hypergraphs*. A hypergraph has nodes that are analogous to the nodes of a graph, but in place of edges a hypergraph has *hyperedges*. Each hyperedge is a set of nodes whose size is not restricted to 2; thus, hypergraphs represent internode relationships that are not “binary”. A simple example can be framed by describing *bus-connected* systems, such as occur in certain genres of digital circuits and certain message-passing systems. The underlying idea is that nodes that coreside in a hyperedge represent agents that “hear” all messages simultaneously, or equipotential points in a network. Because of their inherent complexity, hypergraphs as graph-theoretic objects are usually relegated to advanced courses, but specific concrete instantiations, exemplified by bus-oriented communication and digital circuits should be accessible even to early students.

Chapter 5
ASYMPTOTICS

Asymptotics can be viewed as a language and a system of reasoning that allow one to talk in a *qualitative* voice about *quantitative* topics. We thereby generalize to arbitrary growth functions terms such as “linear”, “quadratic”, “exponential”, and “logarithmic”.

Such a language and system are indispensable if one needs to reason about computational topics over a range of situations, such as a range (“all existing”?) computer architectures and software systems. As two simple examples: (1) Carry-ripple adders perform additions in time linear in the lengths n of the summands (measured in number of bits) no matter what these lengths are. (2) Comparison-based sorting algorithms can sort lists of n keys in time proportional to $n \log n$, but no faster—where the base of the logarithm depends on the characteristics of the computing platform. More precise versions of the preceding statements require specication of the number n and other details, possibly down to the clock speeds of the host computer’s circuitry.

5.1 The language of asymptotics

The language of asymptotics, which has its origins in the field of Number Theory in the late 19th century, builds on the following terminology, which is likely what one would cover in an early undergraduate course. More advanced aspects of the language would likely by beyond the needs of most students of computing, aside from specialists in advanced courses. The basics of the language build on three primitive notations and notions. Standard sources, such as any text on algorithm design and analysis, flesh out the following ideas.

- *The big-O notation*. The assertion $f(x) = O(g(x))$ says, intuitively, that the function f grows no faster than function g. It is, thus, the asymptotic analogue of “less than”.
  Formally: $f(x) = O(g(x))$
  means

$(\exists c > 0)(\exists x^{\#})(\forall x > x^{\#})[f(x) \leq c \cdot g(x)]$

- *The big-Ω notation*. The assertion $f(x) = \Omega(g(x))$ says, intuitively, that the function f grows at least as fast as function g. It is, thus, the asymptotic analogue of "greater than".
  Formally: $f(x) = \Omega(g(x))$
  means
  $(\exists c > 0)(\exists x^{\#})(\forall x > x^{\#})[f(x) \geq c \cdot g(x)]$

- *The big-Θ notation*. The assertion $f(n) = \Theta(g(n))$ says, intuitively, that the function f grows at the same rate as does function g. It is, thus, the asymptotic analogue of "equal to".
  Formally: $f(x) = \Theta(g(x))$
  means
  $(\exists c_1 > 0)(\exists c_2 > 0)(\exists x^{\#})(\forall x > x^{\#})[c_1 \cdot g(x) \leq f(x) \leq c_2 \cdot g(x)]$

One renders the preceding intuitive explanations precise by pointing out that the three specifies relations (a) take hold *eventually*, i.e., only for large arguments to the functions f and g, and (b) hold up to an unspecified constant of proportionality.

5.2 The "uncertainties" in asymptotic relationships

The formal definitions of all three of our asymptotic relationships are bracketed by two important quantifiers:

$$\text{“}(\exists c > 0)'' \quad \text{and “}(\forall x > x^{\#})''.$$

The former, *uncertain-size* quantifier, asserts that asymptotic notions describe functional behavior "in the large". Thus, in common with more common qualitative descriptors of quantitative growth such as linear, quadratic, cubic, quartic, exponential, logarithmic, etc., asymptotic relationships give no infomation about constants of proportionality. *We are not saying that constant factors do not matter! We are, rather, saying that we want to discuss growth patterns in the large.*

The latter, *uncertain-time* quantifier asserts that asymptotic relationships between functions are promised to hold only "eventually", i.e., "for sufficiently large values of the argument x". Therefore, in particular, asymptotic notions cannot be employed to discuss or analyze quantities that can never grow beyond a fixed finite value. The fact that all instances of a quantity throughout history have been below N is immaterial, as long as it is conceivable that an instance larger than N could appear at some time in the future.

These quantifiers in particular distinguishes claims of asymptotic relationship from the more familiar definite inequalities such as "$f(x) \leq g(x)$" or $f(x) \geq 7 \cdot g(x)$. In fact, it is often easier to think about our three asymptotic bounding assertions as establishing *envelopes* for $f(x)$:

- Say that $f(x) = O(g(x))$. If one draws the graphs of the functions $f(x)$ and $c \cdot g(x)$, then as one traces the graphs with increasing values of x, one eventually reaches a point $x^{\#}$ beyond which the graph of $f(x)$ never enters the territory *above* the graph of $c \cdot g(x)$.
- Say that $f(x) = \Omega(g(x))$. This situation is the up-down mirror image of the preceding one: just replace the highlighted "*above*" with "*below*."
- Say that $f(x) = \Theta(g(x))$. We now have a two-sided envelope: beyond $x^{\#}$, the graph of $f(x)$ never enters the territory *above* the graph of $c_1 \cdot g(x)$ and never enters the territory *below* the graph of $c_2 \cdot g(x)$.

In addition to allowing one to make familiar growth-rate comparisons such as "$n^{14} = O(n^{15})$" and "$1.001^n = \Omega(n^{1000})$," we can now also make assertions such as "$\sin x = \Theta(1)$," which are much clumsier to explain in words.

Beyond the big letters. There are "small"-letter analogues of the preceding "big"-letter asymptotic notations, but they are only rarely encountered in discourse about real computations (although they do arise in the analysis of algorithms).

5.3 Inescapable complications

The story we have told thus far is covered in many sources and courses. Two complications to the story are covered less faithfully, although lacking them, one cannot perform cogent asymptotic reasoning. Both complications involve the notion of *uniformity*.

1. *Multiple functions*. Say that we have four functions, f, g, h, k, and we know that both

$$f(n) = O(g(n)) \quad \text{and} \quad h(n) = O(k(n))$$

It is intuitive that

$$f(n) + h(n) = O(g(n) + k(n))$$

— but is it true?

In short, the answer is YES, but verifying that requires a bit of subtlety, because, absent hitherto undisclosed information, the proportionality constants $c_{f,g}$ and $N_{f,g}$ that witness the big-O relationship between functions f and g have no connection with the constants, $c_{h,k}, N_{h,k}$ that witness the analogous relationship between functions h and k. Therefore, in order to verify the posited relationship between functions $f + h$ and $g + k$, one much find witnessing constants $c_{f+h,g+k}$ and $N_{f+h,g+k}$. Of course, this task requires only elementary reasoning and manipulation — but it must be done!

2. *Multivariate functions*. Finally, we discuss the scenario that almost automatically accompanies the transition from a focus on sequential, single-agent computing to a focus on PDC. Within this broadened context, most functions that describe a system have one or more variables that describe the computing system — its number of processors or of agents or the sizes of its memory modules or the communication-radii of its transponders or . . . , in addition to the one or more variables that describe the data that the system is processing. Within such scenarios, every assertion of an asymptotic relationship, of the form

$$f(\mathbf{m};\mathbf{n}) = O(g(\mathbf{m};\mathbf{n}))$$

must explicitly specify the following information:

- which variables can grow without bound;
- among such unbounded variables, which participate in the posited asymptotic relation;
- for each participating unbounded variable x, what are the constants c_x and N_x that witness the posited asymptotic relationship(s).

Clearly the complexity of cogent asymptotic reasoning — hence also the complexity of teaching about such reasoning — gets much more complicated in the multivariate settings engendered by PDC. But, the benefits of being able to reason qualitatively about the quantitative aspects of computing increase at least commensurately!

Chapter 6
PROBABILITY AND STATISTICS

Elements of probability theory and statistics infuse every area of computing. The practicality of many algorithms that are experientially the most efficient for their target tasks depend on the *distribution* of inputs in "real" situations. Design methodologies for crucial complex circuits must acknowledge the *mean times to failure* of the critical components of the circuits. Sophisticated searching algorithms must take into account the relative *likelihoods* of finding one's goal in the various optional search directions. Analyzing and understanding large corpora of data requires a variety of methodologies that build on the concepts of *clustering* and/or *decomposition*.

A student needs at least an introduction to the foundations of probability and statistics to even understand, all the moreso to master, the terms highlighted in the preceding paragraph. We outline many of the key concepts that a student must be exposed to in the following subsections.

6.1 The Elements of Combinatorial Probability

Perhaps the easiest and most engaging way to introduce "probability via counting" is by calculating the comparative likelihoods of various deals in 5-card poker and of various rolls of a pair of dice. The arithmetic required for this discussion is elementary and the "application" to gambling of interest even to non-gamblers: "Why is such a deal in poker (say, a straight) worth more than another (say, three of a kind)?" One can also introduce in this setting concepts such as randomness, bias, etc., that are so important in the design of experiments and the analysis of their outcomes.

6.2 Toward a Basic Understanding of Statistics

Most students whose interest tend to the empirical will likely “do” statistics with the aid of apps, rather than by explicitly writing programs that perform the required calculations. That said, all students should understand the crucial notion of *random variable* and should be conversant with the most common statistical distributions. “Conversant” in this context should include complete understandings of the (low-numbered) moments of *at least* the *uniform* and *exponential* distributions. They should know how to compute, say, the means and variances of various distributions and, most importantly, they should *understand* the sense in which the variance of a distribution give *important* information that is not available from the mean. All of this is prerequisite to rigor in experimentation.

6.3 Beyond the Basics

As students are introduced to modern topics within computing, whether at the level of a Computing Literacy course or a post-core technical course, they will have to master a variety of more specialized topics that combine pieces of the elements we have discussed in this essay. While these topics are beyond the level of generality aimed at in this essay, some may be appropriate prerequisites to programs that have some specialized foci.

- Issues relating to *clustering* find application in applicatios as diverse as: *linear-algebraic computations*, *data mining*, *design and layout of digital circuitry*.
- Issues building on *graph separation/decomposition* are encountered when studying: *linear-algebraic computing*, *fault-tolerant design*, *load balancing*.
- Many issues relating to *fault/failure tolerance* and *data analytics* benefit from study using *random walks* (at least in one dimension).
- Many useful ideas regarding the *encoding and manipulation of data* can be gleaned from the elements of *information theory* and *computer arithmetic*.

The preceding list is really endless. Hopefully readers will be inspired by our few examples to compile a longer version that is appropriate for their particular environments.