APPENDIX A

Induction

Let's explore the idea of induction a bit before getting into the formalities.

Example A.1 A simple inductive proof Suppose that $\mathcal{A}(n)$ is an assertion that depends on n. For example, take $\mathcal{A}(n)$ to be the assertion "$n! > 2^n$". In attempting to decide whether or not $\mathcal{A}(n)$ is true, we may first try to check it for some small values of n. In this example, $\mathcal{A}(1) =$ "$1! > 2^1$" is false, $\mathcal{A}(2) =$ "$2! > 2^2$" is false, and $\mathcal{A}(3) =$ "$3! > 2^3$" is false; but $\mathcal{A}(4) =$ "$4! > 2^4$" is true. We could go on like this, checking each value of n in turn, but this becomes quite tiring.

If we're smart, we can notice that to show that $\mathcal{A}(5)$ is true, we can take the true statement $\mathcal{A}(4) =$ "$4! > 2^4$" and multiply it by the inequality $5 > 2$ to get $5! > 2^5$. This proves that $\mathcal{A}(5)$ is true without doing all of the multiplications necessary to verify $\mathcal{A}(5)$ directly. Since $6 > 2$ and $\mathcal{A}(5)$ is true, we can use the same trick to verify that $\mathcal{A}(6)$ is true. Since $7 > 2$, we could use the fact that $\mathcal{A}(6)$ is true to show that $\mathcal{A}(7)$ is true. This, too, becomes tiring.

If we think about what we are doing in a little more generality, we see that if we have verified $\mathcal{A}(n-1)$ for some value of $n > 2$, we can combine this with $n > 2$ to verify $\mathcal{A}(n)$. This is a "generic" or "general" description of how the validity of $\mathcal{A}(n-1)$ can be transformed into the validity of $\mathcal{A}(n)$. Having verified that $\mathcal{A}(4)$ is true and given a valid generic argument for transforming the validity of $\mathcal{A}(n-1)$ into the validity of $\mathcal{A}(n)$, we claim that the statement "$n! > 2^n$ if $n > 3$" has been proved by *induction*. We hope you believe that this is a proof, because the alternative is to list the infinitude of cases, one for each n. □

Here is a formulation of induction:

Theorem A.1 Induction *Let $\mathcal{A}(m)$ be an assertion, the nature of which is dependent on the integer m. Suppose that we have proved $\mathcal{A}(n)$ for $n_0 \le n \le n_1$ and the statement*

> *"If $n > n_1$ and $\mathcal{A}(k)$ is true for all k such that $n_1 \le k < n$, then $\mathcal{A}(n)$ is true."*

Then $\mathcal{A}(m)$ is true for all $m \ge n_0$.

Definition A.1 *The statement "$\mathcal{A}(k)$ is true for all k such that $n_1 \le k < n$" is called the* **induction assumption** *or* **induction hypothesis** *and proving that this implies $\mathcal{A}(n)$ is called the* **inductive step**. *The values of n with $n_0 \le n \le n_1$ are called the* **base cases**.

In many situations, $n_0 = n_1$ and $n_1 = 0$ or 1; however, this is not always true. In fact, our example requires a different value of n_1. Before reading further, can you identify n_1 in Example A.1? Can you identify the induction assumption?

* * * Stop and think about this! * * *

In our example, we started by proving $\mathcal{A}(4) =$ "$4! > 2^4$", so $n_1 = 4$. The induction assumption is:

$$k! > 2^k \text{ is true for all } k \text{ such that } 4 \le k < n.$$

Remark. Sometimes induction is formulated differently. One difference is that people sometimes use $n+1$ in place of n. Thus the statement in the theorem would be

"If $n \ge n_1$ and $\mathcal{A}(k)$ is true for all $n_1 \le k \le n$, then $\mathcal{A}(n+1)$ is true."

Another difference is that some people always formulate it with $n_0 = 1$. Finally, some people restrict the definition of induction even further by allowing you to use only $\mathcal{A}(n)$ to prove $\mathcal{A}(n+1)$ (or, equivalently, only $\mathcal{A}(n-1)$ to prove $\mathcal{A}(n)$), rather than the full range of $\mathcal{A}(k)$ for $n_0 \le k \le n$. Putting these changes together, we obtain the following more restrictive formulation of induction that is found in some texts:

Corollary Restricted induction *Let $\mathcal{A}(m)$ be an assertion, the nature of which is dependent on the integer m. Suppose that we have proved $\mathcal{A}(1)$ and the statement*

"If $n > 1$ and $\mathcal{A}(n-1)$ is true then, $\mathcal{A}(n)$ is true."

Then $\mathcal{A}(m)$ is true for all $m \ge 1$.

Since this is a special case of Theorem A.1, we'll just simply use Theorem A.1. While there is never need to use a simpler form of induction than Theorem A.1, there is sometimes a need for a more general forms of induction. Discussion of these generalizations is beyond the scope of the text.

Example A.2 A summation We would like a formula for the sum of the first n integers. Let us write $S(n) = 1 + 2 + \ldots + n$ for the value of the sum. By a little calculation,

$$S(1) = 1, \quad S(2) = 3, \quad S(3) = 6, \quad S(4) = 10, \quad S(5) = 15, \quad S(6) = 21.$$

What is the general pattern? It turns out that $S(n) = \frac{n(n+1)}{2}$ is correct for $1 \le n \le 6$. Is it true in general? This is a perfect candidate for an induction proof with

$$n_0 = n_1 = 1 \qquad \text{and} \qquad \mathcal{A}(n): \quad \text{"}S(n) = \tfrac{n(n+1)}{2}\text{."} \tag{A.1}$$

Let's prove it. We have shown that $\mathcal{A}(1)$ is true. In this case we need only the restricted induction hypothesis; that is, we will prove the formula for $S(n)$ by using the formula for $S(n-1)$. Here it is (the inductive step):

$$\begin{aligned} S(n) &= 1 + 2 + \cdots + n = \Big(1 + 2 + \cdots + (n-1)\Big) + n \\ &= S(n-1) + n = \frac{(n-1)\big((n-1)+1\big)}{2} + n && \text{by } \mathcal{A}(n-1), \\ &= \frac{n(n+1)}{2} && \text{by algebra.} \end{aligned}$$

This completes the proof. □

Example A.3 Divisibility We will prove that for all integers $x > 1$ and all positive integers n, $x-1$ divides $x^n - 1$. In this case $n_0 = n_1 = 1$, $\mathcal{A}(n)$ is the statement that $x-1$ divides $x^n - 1$. Since $\mathcal{A}(1)$ states that $x-1$ divides $x-1$ it is obvious that $\mathcal{A}(1)$ is true. Now for the induction step. How can we rephrase $\mathcal{A}(n)$ so that it involves $\mathcal{A}(n-1)$? After a bit of thought, you may come up with

$$x^n - 1 \;=\; x(x^{n-1} - 1) + (x - 1). \tag{A.2}$$

Once this is discovered, the rest is easy:

- By $\mathcal{A}(n-1)$, $x^{n-1} - 1$ is divisible by $x - 1$, thus $x(x^{n-1} - 1)$ is divisible by $x - 1$.
- Also, $x - 1$ divides $x - 1$.
- Thus the sum in (A.2) is divisible by $x - 1$.

This completes the proof.

We could have used the same proof as this one for the statement that the polynomial $x - 1$ divides the polynomial $x^n - 1$ for all positive integers n. □

In the last example, the hardest part was figuring out how to use $\mathcal{A}(n-1)$ in the proof of $\mathcal{A}(n)$. This is quite common:

Observation *The difficult part of a proof by induction is often figuring out how to use the inductive hypothesis.*

Simple examples of inductive proofs may not make this clear. Another difficulty that can arise, as happened in Example A.2, may be that we are not even given a theorem but are asked to discover it.

Example A.4 An integral formula We want to prove the formula

$$\int_0^1 x^a (1-x)^b dx = \frac{a!\; b!}{(a+b+1)!}$$

for nonnegative integers a and b.

What should we induct on? We could choose a or b. Let's use b. Here it is with b replaced with n to look more like our usual formulation:

$$\int_0^1 x^a (1-x)^n dx \;=\; \frac{a!\; n!}{(a+n+1)!}. \tag{A.3}$$

This is our $\mathcal{A}(n)$. Also $n_0 = n_1 = 0$, the smallest nonnegative integer. You should verify that $\mathcal{A}(0)$ is true, thereby completing the first step in the inductive proof. (Remember that $0! = 1$.)

How can we use our inductive hypothesis to prove $\mathcal{A}(n)$? This is practically equivalent to asking how to manipulate the integral in (A.3) so that the power of $(1-x)$ is reduced. Finding the answer depends on a knowledge of techniques of integration. Someone who has mastered them will realize that integration by parts could be used to reduce the degree of $1-x$ in the integral. Let's do it. Integration by parts states that $\int u\, dv \;=\; uv - \int v\, du$. We set $u = (1-x)^n$ and $dv = x^a dx$. Then $du = -n(1-x)^{n-1}$, $v = \frac{x^{a+1}}{a+1}$ and

$$\begin{aligned}\int_0^1 x^a (1-x)^n dx \;&=\; \frac{(1-x)^n x^{a+1}}{a+1}\bigg|_0^1 + \frac{n}{a+1}\int_0^1 x^{a+1}(1-x)^{n-1} dx \\ &=\; \frac{n}{a+1}\int_0^1 x^{a+1}(1-x)^{n-1} dx.\end{aligned}$$

The last integral can be evaluated by $\mathcal{A}(n-1)$ and so

$$\int_0^1 x^a (1-x)^n dx = \frac{n}{a+1}\,\frac{(a+1)!\,(n-1)!}{((a+1)+(n-1)+1)!} \;=\; \frac{a!\,n!}{(a+n+1)!}.$$

This completes the inductive step, thereby proving (A.3). □

We now do a more combinatorial inductive proof. This requires a definition from the first chapter. If A is a set, let $|A|$ stand for the number of elements in A.

Definition A.2 Cartesian Product *If $C_1, \ldots, C_k$ are sets, the Cartesian product of the sets is written $C_1 \times \cdots \times C_k$ and consists of all k long lists $(x_1, \ldots, x_k)$ with $x_i \in C_i$ for $1 \le i \le k$.*

Example A.5 Size of the Cartesian product We want to prove

$$|C_1 \times \cdots \times C_n| \;=\; |C_1| \cdots |C_n|. \tag{A.4}$$

This is our $\mathcal{A}(n)$. Since this is trivially true for $n = 1$, it is reasonable to take $n_0 = n_1 = 1$. It turns out that this choice will make the inductive step more difficult. It is better to choose $n_1 = 2$. (Normally one would discover that part way through the proof. To simplify things a bit, we've "cheated" by telling you ahead of time.)

To begin with, we need to prove $\mathcal{A}(2)$. How can this be done? This is the difficult part. Let's postpone it for now and do the inductive step.

We claim that

the sets $C_1 \times \cdots \times C_n$ and $(C_1 \times \cdots \times C_{n-1}) \times C_n$ have the same number of elements.

Why is this? They just differ by pairs of parentheses: Suppose $x_1 \in C_1$, $x_2 \in C_2$, ... and $x_n \in C_n$. Then

$$(x_1, \ldots, x_n) \;\in\; C_1 \times \cdots \times C_n$$

and

$$((x_1, \ldots, x_{n-1}), x_n) \;\in\; (C_1 \times \cdots \times C_{n-1}) \times C_n$$

just differ by pairs of parentheses. Thus

$$\begin{aligned} |C_1 \times \cdots \times C_n| \;&=\; |(C_1 \times \cdots \times C_{n-1}) \times C_n| && \text{by the above,} \\ &=\; |C_1 \times \cdots \times C_{n-1}| \cdot |C_n| && \text{by } \mathcal{A}(2), \\ &=\; \Big(|C_1| \cdots |C_{n-1}|\Big)|C_n| && \text{by } \mathcal{A}(n-1). \end{aligned}$$

This completes the inductive step. Note that this is different from our previous inductive proofs in that we used both $\mathcal{A}(n-1)$ and $\mathcal{A}(2)$ in the inductive step.

We must still prove $\mathcal{A}(2)$. Let $C_1 = \{y_1, \ldots, y_k\}$, where $k = |C_1|$. Then

$$C_1 \times C_2 = (\{y_1\} \times C_2) \cup \cdots \cup (\{y_k\} \times C_2).$$

Since all of the sets in the union are disjoint, the number of elements in the union is the sum of the number of elements in each of the sets separately. Thus

$$|C_1 \times C_2| = \big|\{y_1\} \times C_2\big| + \cdots + \big|\{y_k\} \times C_2\big|.$$

You should be able to see that $|\{y_i\} \times C_2| = |C_2|$. Since the sum has $k = |C_1|$ terms, all of which equal $|C_2|$, we finally have $|C_1 \times C_2| = |C_1| \cdot |C_2|$. □

Our final example requires more than $\mathcal{A}(2)$ and $\mathcal{A}(n-1)$ to prove $\mathcal{A}(n)$.

Example A.6 Every integer is a product of primes A positive integer $n > 1$ is called a *prime* if its only divisors are 1 and n. The first few primes are 2, 3, 5, 7, 11, 13, 17, 19, 23. We now prove that every integer $n > 1$ is a product of primes, where we consider a single prime p to be a product of one prime, itself. We shall prove $\mathcal{A}(n)$:

"Every integer $n \geq 2$ is a product of primes."

We start with $n_0 = n_1 = 2$, which is a prime and hence a product of primes. Assume the induction hypothesis and consider $\mathcal{A}(n)$. If n is a prime, then it is a product of primes (itself). Otherwise, n has a divisor s with $1 < s < n$ and so $n = st$ where $1 < t < n$. By the induction hypotheses $\mathcal{A}(s)$ and $\mathcal{A}(t)$, s and t are each a product of primes, hence $n = st$ is a product of primes. This completes the proof of $\mathcal{A}(n)$ □

In all except Example A.5, we used algebra or calculus manipulations to arrange $\mathcal{A}(n)$ so that we could apply the inductive hypothesis. In Example A.5, we used a set theoretic argument: We found that the elements in the set $C_1 \times \cdots \times C_n$ could be put into one to one correspondence with the elements in the set $(C_1 \times \cdots \times C_{n-1}) \times C_n$. This sort of set-theoretic argument is more common in combinatorial inductive proofs than it is in noncombinatorial ones.

Exercises

In these exercises, indicate clearly

(i) what n_0, n_1 and $\mathcal{A}(n)$ are,

(ii) what the inductive step is and

(iii) where the inductive hypothesis is used.

A.1. Prove that the sum of the first n odd numbers is n^2.

A.2. Prove that $\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$ for $k \geq 0$.

A.3. Conjecture and prove the general formula of which the following are special cases:

$$1 - 4 + 9 - 16 = -(1 + 2 + 3 + 4)$$
$$1 - 4 + 9 - 16 + 25 = 1 + 2 + 3 + 4 + 5.$$

*A.4. Conjecture and prove the general formula of which the following are special cases:

$$\frac{x}{1+x} + \frac{2x^2}{1+x^2} = \frac{x}{1-x} - \frac{4x^4}{1-x^4}$$
$$\frac{x}{1+x} + \frac{2x^2}{1+x^2} + \frac{4x^4}{1+x^4} = \frac{x}{1-x} - \frac{8x^8}{1-x^8}.$$

A.5. Some calculus texts omit the proof of $(x^n)' = nx^{n-1}$ or slide over the inductive nature of the proof. Give a proper inductive proof for $n \geq 1$ using $x' = 1$ and the formula for the derivative of a product.

A.6. Conjecture and prove a formula for $\int_0^\infty x^n e^{-x} dx$ for $n \geq 0$. (Your answer should not include integrals.)

A.7. What is wrong with the following inductive proof that all people have the same sex? Let $\mathcal{A}(n)$ be "In any group of n people, all people are of the same sex." This is obviously true for $n = 1$. For $n > 1$, label the people $P_1, \ldots, P_n$. By the induction assumption, $P_1, \ldots, P_{n-1}$ are all of the same sex and $P_2, \ldots, P_n$ are all of the same sex. Since P_2 belongs to both groups, the sex of the two groups is the same and so we are done.

A.8. We will show by induction that $1 + 2 + \cdots + n = (2n + 1)^2/8$. By the inductive hypothesis, $1 + 2 + \cdots + (n - 1) = (2n - 1)^2/8$. Adding n to both sides and using $n + (2n - 1)^2/8 = (2n + 1)/8$, we obtain the formula. What is wrong with this proof?

A.9. Imagine drawing n distinct straight lines so as to divide the plane into regions in some fashion. Prove that the regions can be assigned numbers 0 and 1 so that if two regions share a line segment in their boundaries, then they are numbered differently.

APPENDIX B

Rates of Growth and Analysis of Algorithms

Suppose we have an algorithm and someone asks us "How good is it?" To answer that question, we need to know what they mean. They might mean "Is it correct?" or "Is it understandable?" or "Is it easy to program?" We won't deal with any of these.

They also might mean "How fast is it?" or "How much space does it need?" These two questions can be studied by similar methods, so we'll just focus on speed. Even now, the question is not precise enough. Does the person mean "How fast is it on this particular problem and this particular machine using this particular code and this particular compiler?" We could answer this simply by running the program! Unfortunately, that doesn't tell us what would happen with other machines or with other problems that the algorithm is designed to handle.

We would like to answer a question such as "How fast is Algorithm 1 for finding a spanning tree?" in such a way that we can compare that answer to "How fast is Algorithm 2 for finding a spanning tree?" and obtain something that is not machine or problem dependent. At first, this may sound like an impossible goal. To some extent it is; however, quite a bit can be said.

How do we achieve machine independence? We think in terms of simple machine operations such as multiplication, fetching from memory and so on. If one algorithm uses fewer of these than another, it should be faster. Those of you familiar with computer instruction timing will object that different basic machine operations take different amounts of time. That's true, but the times are not wildly different. Thus, if one algorithm uses *a lot fewer* operations than another, it should be faster. It should be clear from this that we can be a bit sloppy about what we call an operation; for example, we might call something like $x = a + b$ one operation. On the other hand, we can't be so sloppy that we call $x = a_1 + \cdots + a_n$ one operation if n is something that can be arbitrarily large.

Suppose we have an algorithm for a class of problems. If we program the algorithm in some language and use some compiler to produce code that we run on some machine, then there is a function $f(n)$ that measures the average (or worst, if you prefer) running time for the program on problems

of size n. We want to study how $f(n)$ grows with n, but we want to express our answers in a manner that is independent of the language, compiler, and computer. Mathematicians have introduced notation that is quite useful for studying rates of growth in this manner. We'll study the notation in this appendix.

B.1 The Basic Functions

Example B.1 Let's look at how long it takes to find the maximum of a list of n integers where we know nothing about the order they are in or how big the integers are. Let $a_1, \ldots, a_n$ be the list of integers. Here's our algorithm for finding the maximum.

$max = a_1$
`For` $i = 2, \ldots, n$
    `If` $a_i > max$`, then` $max = a_i$`.`
    `End for`
`Return` max

Being sloppy, we could say that the entire comparison and replacement in the "`If`" takes an operation and so does the stepping of the index i. Since this is done $n-1$ times, we get $2n-2$ operations. There are some setup and return operations, say s, giving a total of $2n-2+s$ operations. Since all this is rather sloppy all we can really say is that for large n and actual code on an actual machine, the procedure will take about Cn "ticks" of the machine's clock. Since we can't determine C by our methods, it will be helpful to have a notation that ignores it. We use $\Theta(f(n))$ to designate any function that behaves like a constant times $f(n)$ for arbitrarily large n. Thus we would say that the "`If`" takes time $\Theta(n)$ and the setup and return takes time $\Theta(1)$. Thus the total time is $\Theta(n) + \Theta(1)$. Since n is much bigger than 1 for large n, the total time is $\Theta(n)$. □

We need to define Θ more precisely and list its most important properties. We will also find it useful to define O, read "big oh."

Definition B.1 Notation for Θ and O *Let f and g be functions from the positive integers to the real numbers.*

- *We say that $g(n)$ is $O(f(n))$ if there exists a positive constant B such that $|g(n)| \leq B|f(n)|$ for all sufficiently large n. In this case we say that g grows no faster than f or, equivalently, that f grows at least as fast as g.*
- *We say that $g(n)$ is $O^+(f(n))$ if $g(n)$ is $O(f(n))$ and $g(n) \geq 0$ for sufficiently large n.*
- *We say that $g(n)$ is $\Theta(f(n))$ if (i) $g(n)$ is $O(f(n))$ and (ii) $f(n)$ is $O(g(n))$. In this case we say that f and g grow at the same rate.*
- *We say that $g(n)$ is $\Theta^+(f(n))$ if $g(n)$ is $\Theta(f(n))$ and $g(n) \geq 0$ for sufficiently large n.*

Remarks **1**. The phrase "$\mathcal{S}(n)$ is true for all sufficiently large n" means that there is some integer N such that $\mathcal{S}(n)$ is true whenever $n \geq N$.

2. Saying that something is $\Theta^+(f(n))$ gives an idea of *how big it is* for large values of n. Saying that something is $O^+(f(n))$ gives an idea of *an upper bound on how big it is* for all large values of n. (We said "idea of" because we don't know what the constants are.)

3. The notation O^+ and Θ^+ is *not standard*. We have introduced it because it is convenient when combining functions.

Theorem B.1 Some properties of Θ and O *We have*

(a) *$g(n)$ is $\Theta(f(n))$ if and only if there are positive constants A and B such that $A|f(n)| \leq |g(n)| \leq B|f(n)|$ for all sufficiently large n.*

(b) *If $g(n)$ is $\Theta^+(f(n))$, then $g(n)$ is $\Theta(f(n))$.*
If $g(n)$ is $O^+(f(n))$, then $g(n)$ is $O(f(n))$.
If $g(n)$ is $\Theta(f(n))$, then $g(n)$ is $O(f(n))$.

(c) *$f(n)$ is $\Theta(f(n))$ and $f(n)$ is $O(f(n))$.*

(d) *If $g(n)$ is $\Theta(f(n))$ and C and D are nonzero constants, then $Cg(n)$ is $\Theta(Df(n))$.*
If $g(n)$ is $O(f(n))$ and C and D are nonzero constants, then $Cg(n)$ is $O(Df(n))$.

(e) *If $g(n)$ is $\Theta(f(n))$, then $f(n)$ is $\Theta(g(n))$.*

(f) *If $g(n)$ is $\Theta(f(n))$ and $f(n)$ is $\Theta(h(n))$, then $g(n)$ is $\Theta(h(n))$.*
If $g(n)$ is $O(f(n))$ and $f(n)$ is $O(h(n))$, then $g(n)$ is $O(h(n))$.

(g) *If $g_1(n)$ is $\Theta(f_1(n))$ and $g_2(n)$ is $\Theta(f_2(n))$, then $g_1(n)g_2(n)$ is $\Theta(f_1(n)f_2(n))$ and, if in addition f_2 and g_2 are never zero, then $g_1(n)/g_2(n)$ is $\Theta(f_1(n)/f_2(n))$.*
If $g_1(n)$ is $O(f_1(n))$ and $g_2(n)$ is $O(f_2(n))$, then $g_1(n)g_2(n)$ is $O(f_1(n)f_2(n))$.

(h) *If $g_1(n)$ is $\Theta^+(f_1(n))$ and $g_2(n)$ is $\Theta^+(f_2(n))$, then $g_1(n)+g_2(n)$ is $\Theta^+\big(\max\big(|f_1(n)|, |f_2(n)|\big)\big)$.*
If $g_1(n)$ is $O(f_1(n))$ and $g_2(n)$ is $O(f_2(n))$, then $g_1(n) + g_2(n)$ is $O\big(\max\big(|f_1(n)|, |f_2(n)|\big)\big)$.

(i) *If $g(n)$ is $\Theta^+(f(n))$ and $h(n)$ is $O^+(f(n))$, then $g(n) + h(n)$ is $\Theta^+(f(n))$.*

Note that by Theorem 5.1 (p. 127) and properties (c), (e) and (f) above, the statement "$g(n)$ is $\Theta(f(n))$" defines an equivalence relation on the set of functions from the positive integers to the reals. Similarly, "$g(n)$ is $\Theta^+(f(n))$" defines an equivalence relation on the set of functions which are positive for sufficiently large n.

Proof: If you completely understand the definitions of O, O^+, Θ, and Θ^+, many parts of the theorem will be obvious to you. None of the parts is difficult to prove and so we leave most of the proofs as an exercise. To help you out, we'll do a couple of proofs.

We'll do (f) for Θ. By (a), there are nonzero constants A_i and B_i such that

$$A_1|f(n)| \le |g(n)| \ \le \ B_1|f(n)|$$

and

$$A_2|h(n)| \le |f(n)| \ \le \ B_2|h(n)|$$

for all sufficiently large n. It follows that

$$A_1A_2|h(n)| \ \le \ A_1|f(n)| \ \le \ |g(n)| \ \le \ B_1|f(n)| \ \le \ B_1B_2|h(n)|$$

for all sufficiently large n. With $A = A_1A_2$ and $B = B_1B_2$, it follows that $g(n)$ is $\Theta(h(n))$.

We'll do (h) for Θ^+. By (a), there are nonzero constants A_i and B_i such that

$$A_1|f_1(n)| \ \le \ g_1(n) \ \le \ B_1|f_1(n)| \quad \text{and} \quad A_2|f_2(n)| \ \le \ g_2(n) \ \le \ B_2|f_2(n)|$$

for sufficiently large n. Adding gives

$$A_1|f_1(n)| + A_2|f_2(n)| \ \le \ g_1(n) + g_2(n) \ \le \ B_1|f_1(n)| + B_2|f_2(n)| \tag{B.1}$$

We now do two things. First, let $A = \min(A_1, A_2)$ and note that

$$A_1|f_1(n)| + A_2|f_2(n)| \ \ge \ A\big(|f_1(n)| + |f_2(n)|\big) \ \ge \ A\max\big(|f_1(n)|, |f_2(n)|\big).$$

Second, let $B = 2\max(B_1, B_2)$ and note that

$$\begin{aligned} B_1|f_1(n)| + B_2|f_2(n)| \ &\le \ \frac{B\big(|f_1(n)| + |f_2(n)|\big)}{2} \\ &\le \ \frac{B\,2\max(|f_1(n)|, |f_2(n)|\big)}{2} \ = \ B\max\big(|f_1(n)|, |f_2(n)|\big). \end{aligned}$$

Using these two results in (B.1) gives

$$A\max\big(|f_1(n)|, |f_2(n)|\big) \ \le \ g_1(n) + g_2(n) \ \le \ B\max(|f_1(n)|, |f_2(n)|\big),$$

which proves that $g_1(n) + g_2(n)$ is $\Theta^+\max\big(|f_1(n)|, |f_2(n)|\big)$. □

Example B.2 Using the notation To illustrate these ideas, we'll consider three algorithms for evaluating a polynomial $p(x)$ of degree n at some point r; i.e., computing $p_0 + p_1r + \cdots + p_nr^n$. We are interested in how fast they are when n is large. Here are the procedures. You should convince yourself that they work.

```
Poly1(n, p, r)
      S = p_0
      For i = 1, ..., n     S = S + p_i*Pow(r, i).
      Return S
End

Pow(r, i)
      P = 1
      For j = 1, ..., i     P = P * r.
      Return P
End
```

```
Poly2(n, p, r)
      S = p_0
      P = 1
      For i = 1, ..., n
            P = P * r
            S = S + p_i * P
      End for
      Return S
End
```

```
Poly3(n, p, r)
      S = p_n
      /* Here i decreases from n to 1. */
      For i = n, ..., 2, 1      S = S * r + p_{i-1}
      Return S
End
```

Let $T_n(\texttt{Name})$ be the time required for the procedure `Name`. Let's analyze `Poly1`. The "`For`" loop in `Pow` is executed i times and so takes Ci operations for some constant $C > 0$. The setup and return in `Pow` takes some constant number of operations $D > 0$. Thus $T_n(\texttt{Pow}) = Ci + D$ operations. As a result, the ith iteration of the "`For`" loop in `Poly1` takes $Ci + E$ operations for some constants C and $E > D$. Adding this over $i = 1, 2, \ldots, n$, we see that the total time spent in the "`For`" loop is $\Theta^+(n^2)$ since $\sum_{i=1}^{n} i = n(n+1)/2$. (You should write out the details.) Since the rest of `Poly1` takes $\Theta^+(1)$ time, $T_n(\texttt{Poly1})$ is $\Theta^+(n^2)$ by Theorem B.1(h).

The amount of time spent in the "`For`" loop of `Poly2` is constant and the loop is executed n times. It follows that $T_n(\texttt{Poly2})$ is $\Theta^+(n)$. The same analysis applies to `Poly3`.

What can we conclude from this about the comparative speed of the algorithms? By Theorem B.1(a), Θ^+, there are positive reals A and B so that $An^2 \le T_n(\texttt{Poly1})$ and $T_n(\texttt{Poly2}) \le Bn$ for sufficiently large n. Thus $T_n(\texttt{Poly2})/T_n(\texttt{Poly1}) \le B/An$. As n gets larger, `Poly2` looks better and better compared to `Poly1`.

Unfortunately, the crudeness of Θ does not allow us to make any distinction between `Poly2` and `Poly3`. What we can say is that $T_n(\texttt{Poly2})$ is $\Theta^+(T_n(\texttt{Poly3}))$; i.e., $T_n(\texttt{Poly2})$ and $T_n(\texttt{Poly3})$ grow at the same rate. A more refined estimate can be obtained by counting the actual number of operations involved. You should compare the number of multiplications required and thereby obtain a more refined estimate. □

So far we've talked about how long an algorithm takes to run as if this were a simple, clear concept. In the next example we'll see that there's an important point that we've ignored.

Example B.3 What is average running time? Let's consider the problem of (a) deciding whether or not a simple graph[1] can be properly colored with four colors and, (b) if a proper coloring exists, producing one. (A proper coloring is a function $\lambda: V \to C$, the set of colors, such that, if $\{u, v\}$ is an edge, then $\lambda(u) \neq \lambda(v)$.) We may as well assume that $V = \underline{n}$ and that the colors are c_1, c_2, c_3 and c_4.

Here's a simple algorithm to determine a λ by using backtracking to go lexicographically through possible colorings $\lambda(1), \lambda(2), \ldots, \lambda(n)$.

1. Initialize: Set $v = 1$ and $\lambda(1) = c_1$.
2. Advance in decision tree: If $v = n$, stop with λ determined; otherwise, set $v = v + 1$ and $\lambda(v) = c_1$.
3. Test: If $\lambda(i) \neq \lambda(v)$ for all $i < v$ for which $\{i, v\} \in E$, go to Step 2.
4. Select next decision: Let j be such that $\lambda(v) = c_j$. If $j < 4$, set $\lambda(v) = c_{j+1}$ and go to Step 3.
5. Backtrack: If $v = 1$, stop with coloring impossible; otherwise, set $v = v - 1$ and go to Step 4.

How fast is this algorithm? Obviously it will depend on the graph. For example, if the subgraph induced by the first five vertices is the complete graph K_5 (i.e., all of the ten possible edges are present), then the algorithm stops after trying to color the first five vertices and discovering that there is no proper coloring. If the last five vertices induce K_5 and the remaining $n - 5$ vertices have no edges, then the algorithm makes $\Theta^+(4^n)$ assignments of the form $\lambda(v) = c_k$.

It's reasonable to talk about the average time the algorithm takes if we expect to give it lots of graphs to look at. Most n vertex graphs will have many sets of five vertices that induce K_5. (We won't prove this.) Thus, we should expect that the algorithm will stop quickly on most graphs. In fact, it can be proved that the average number of assignments of the form $\lambda(v) = c_k$ that are made is $\Theta^+(1)$. This means that the average running time of the algorithm is bounded for all n, which is quite good!

Now suppose you give this algorithm to a friend, telling him that on average the running time is practically independent of the number of vertices. He thanks you profusely for such a wonderful algorithm and puts it to work coloring randomly generated "planar" graphs. By the Four Color Theorem, every planar graph can be properly colored with four colors, so the algorithm must make at least n assignments of the form $\lambda(v) = c_k$. (Actually it will almost surely make *many, many* more.) Your friend soon comes back to you complaining bitterly.

What went wrong? In our previous discussion we were averaging over all simple graphs with n vertices. Your friend was interested in the average over all simple planar graphs with n vertices. These averages are *very* different! There is a moral here:

You must be VERY clear what you are averaging over.

Because situations like this do occur in real life, computer scientists are careful to specify what kind of running time they are talking about; either the average of the running time over some reasonable, clearly specified set of problems or the worst (longest) running time over all possibilities. □

[1] A "simple graph" is a set V, called vertices, and a set E of two element subsets of V, called edges. One thinks of an edge $\{u, v\}$ as joining the two vertices u and v.

Example B.4 Which algorithm is faster? Suppose we have two algorithms for a problem, one of which is $\Theta^+(n^2)$ and the other of which is $\Theta^+(n^2 \ln\ln n)$. Which is better?[2] It would seem that the first algorithm is better since n^2 grows slower than $n^2 \ln\ln n$. That's true if n is large enough. How large is large enough? In other words, what is the *crossover point*, the time when we should switch from one algorithm to the other in the interests of speed? To decide, we need to know more than just $\Theta^+(\ )$ because that notation omits constants. Suppose one algorithm has running time close to $3n^2$, and the other, close to $n^2 \ln\ln n$. The second algorithm is better as long as $3 > \ln\ln n$. Exponentiating twice, we get $n < e^{e^3}$, which is about 5×10^8. This is a large crossover point! On the other hand, suppose the first algorithm's running time is close to n^2. In that case, the second algorithm is better as long as $1 > \ln\ln n$, that is, $n < e^e$, which is about 15. A slight change in the constant caused a huge change in the crossover point!

If slight changes in constants matter this much in locating crossover points, what good is $\Theta^+(\ )$ notation? We've misled you! The crossover points are not that important. What matters is how much faster one algorithm is than another. If one algorithm has running time An^2 and the other has running time $Bn^2 \ln\ln n$, the ratio of their speeds is $(B/A) \ln\ln n$. This is fairly close to B/A for a large range of n because the function $\ln\ln n$ grows so slowly. In other words, the running time of the two algorithms differs by nearly a constant factor for practical values of n.

We're still faced with a problem when deciding between two algorithms since we don't know the constants. Suppose both algorithms are $\Theta^+(n^2)$. Which do we choose? If you want to be *certain* you have the faster algorithm, you'll either have to do some very careful analysis of the code or run the algorithms and time them. However, there is a rule of thumb that works pretty well: More complicated data structures lead to larger constants.

Let's summarize what we've learned in the last two paragraphs. Suppose we want to choose the faster of Algorithm A whose running time is $\Theta^+(a(n))$ and Algorithm B whose running time is $\Theta^+(b(n))$.)

- If possible, simplify $a(n)$ and $b(n)$ and ignore all slowly growing functions of n such as $\ln\ln n$. ("What about $\ln n$?" you ask. That's a borderline situation. It's usually better to keep it.) This gives two new functions $a^*(n)$ and $b^*(n)$.
- If $a^*(n) = \Theta^+(b^*(n))$, choose the algorithm with the simpler data structures; otherwise, choose the algorithm with the smaller function.

These rules are far from foolproof, but they provide some guidance. □

We now define two more notations that are useful in discussing rate of growth. The notation o is read "little oh". There is no standard convention for reading $\sim$.

Definition B.2 Notation for o and $\sim$ *Let f, g and h be functions from the positive integers to the real numbers.*

- *If $\lim_{n\to\infty} g(n)/f(n) = 1$, we say that $g(n) \sim f(n)$. In this case, we say that f and g are* **asymptotically equal**.
- *If $\lim_{n\to\infty} h(n)/f(n) = 0$, we say that $hf(n)$ is $o(g(n))$. In this case, we say that h grows slower than f or, equivalently, that f grows faster than h.*

[2] This situation actually occurs, see the discussion at the end of Example 6.3 (p. 152).

Asymptotic equality has many of the properties of equality. The two main exceptions are cancellation and exponentiation:

- You should verify that $n^2 + 1 \sim n^2 + n$ and cancelling the n^2 from both sides is bad because $1 \sim n$ is *false*.
- You should verify that exponentiation is bad by showing that $e^{n^2+1} \sim e^{n^2+n}$ is *false*. In fact, you should verify that e^{n^2+1} is $o(e^{n^2+n})$.

In the following theorem, we see that asymptotic equality defines an equivalence relation (d), allows multiplication and division (c, e), and allows addition of functions with the same sign (f). You should verify that Theorem B.1 says the same thing for $\Theta(\ )$.

Theorem B.2 Some properties of o and $\sim$ *We have*

(a) *If $g(n)$ is $o(f(n))$, then $g(n)$ is $O(f(n))$.*
If $f(n) \sim g(n)$, then $f(n)$ is $\Theta(g(n))$.

(b) *$f(n)$ is* **not** *$o(f(n))$.*

(c) *If $g(n)$ is $o(f(n))$ and C and D are nonzero constants, then $Cg(n)$ is $o(Df(n))$.*
If $g(n) \sim f(n)$ and C is a nonzero constant, then $Cg(n) \sim Cf(n)$.

(d) *"$g(n) \sim f(n)$" defines and equivalence relation.*

(e) *If $g_1(n)$ is $o(f_1(n))$ and $g_2(n)$ is $o(f_2(n))$, then $g_1(n)g_2(n)$ is $o(f_1(n)f_2(n))$.*
If $g_1(n) \sim f_1(n)$ and $g_2(n) \sim f_2(n)$, then $g_1(n)g_2(n) \sim f_1(n)f_2(n)$
and $g_1(n)/g_2(n) \sim f_1(n)/f_2(n)$.

(f) *If $g_1(n)$ is $o(f_1(n))$ and $g_2(n)$ is $o(f_2(n))$, then $g_1(n) + g_2(n)$ is $o(\max(f_1(n), f_2(n)))$.*
If $g_1(n) \sim f_1(n)$ and $g_2(n) \sim f_2(n)$ and $f_1(n)$ and $g_1(n)$ are nonnegative for all sufficiently large n, then $g_1(n) + g_2(n) \sim f_1(n) + f_2(n)$.

(g) *If $h(n)$ is $o(f(n))$ and $g(n) \sim f(n)$, then $g(n) + h(n) \sim f(n)$.*
If $h(n)$ is $o(f(n))$ and $g(n)$ is $\Theta(f(n))$, then $g(n) + h(n)$ is $\Theta(f(n))$.
If $h(n)$ is $o(f(n))$ and $g(n)$ is $O(f(n))$, then $g(n) + h(n)$ is $O(f(n))$.

(h) *If $g_1(n)$ is $o(f_1(n))$ and $g_2(n)$ is $O(f_2(n))$, then $g_1(n)g_2(n)$ is $o(f_1(n)f_2(n))$.*

The proof is left as an exercise. We'll see some applications in the next section.

Exercises

B.1.1. Prove the properties of $\Theta(\ )$, $\Theta^+(\ )$, $O(\ )$, and $O^+(\ )$ given in Theorem B.1.

B.1.2. Prove by example that (g) in Theorem B.1 does not hold for Θ.

B.1.3. Prove or disprove the statement:
"$g(n)$ is $O(f(n))$" defines an equivalence relation for functions from the positive integers to the nonnegative reals (as did the corresponding statement for Θ).

B.1.4. In each case, prove that $f(x)$ is $\Theta(g(x))$ using the definition of $\Theta(\ )$.

(a) $f(x) = x^3 + 5x^2 + 10$, $g(x) = 20x^3$.

(b) $f(x) = x^2 + 5x^2 + 10$, $g(x) = 200x^2$.

B.1.5. In each case, show that the given series has the indicated property.

(a) $\sum_{i=1}^{n} i^2$ is $\Theta(n^3)$.

(b) $\sum_{i=1}^{n} i^3$ is $\Theta(n^4)$.

(c) $\sum_{i=1}^{n} i^{1/2}$ is $\Theta(n^{3/2})$.

B.1.6. Show each of the following

(a) $\sum_{i=1}^{n} i^{-1}$ is $\Theta(\log_b(n))$ for any base $b > 1$.

(b) $\log_b(n!)$ is $O(n \log_b(n))$ for any base $b > 1$

B.1.7. We have three algorithms for solving a problem for graphs. Suppose algorithm A takes n^2 milliseconds to run on a graph with n vertices, algorithm B takes $100n$ milliseconds and algorithm C takes $100(2^{n/10} - 1)$ milliseconds.

(a) Compute the running times for the three algorithms with $n = 5$, 10, 30, 100 and 300. Which algorithm is fastest in each case? slowest?

(b) Which algorithm is fastest for all very large values of n? Which is slowest?

B.1.8. Prove Theorem B.2.

B.1.9. Let $p(x)$ be a polynomial of degree k with positive leading coefficient and suppose that $a > 1$. Prove the following.

(a) $\Theta(p(n))$ is $\Theta(n^k)$.

(b) $O(p(n))$ is $O(n^k)$.

(c) $p(n) = o(a^n)$. (Also, what does this say about the speed of a polynomial time algorithm versus one which takes exponential time?)

(d) $O(a^{p(n)})$ is $O(a^{Cn^k})$ for some $C > 0$.

(e) Unless $p(x) = p_1 x^k + p_2$ for some p_1 and p_2, there is no C such that $a^{p(n)}$ is $\Theta(a^{Cn^k})$.

B.1.10. Suppose $1 < a < b$ and $f(n) \to +\infty$ as $n \to \infty$. Prove that

$$a^{f(n)} = o(b^{f(n)}) \qquad \text{and} \qquad a^{g(n)} = o(a^{f(n)+g(n)}),$$

for all functions $g(n)$.

B.1.11. Consider the following algorithm for determining if the distinct integers $a_1, a_2, \ldots, a_n$ are in increasing order.

```
For i = 2, ..., n
     If a_{i-1} > a_i, return ‘‘out of order.’’
End for
Return ‘‘in order.’’
```

(a) Discuss worst case running time.

(b) Discuss average running time for all permutations of $\underline{n}$.

(c) Discuss average running time for all permutations of $\underline{2n}$ such that $a_1 < a_2 < \cdots < a_n$.

B.2 Doing Arithmetic

If we try to use Theorems B.1 and B.2 in a series of calculations, the lack of arithmetic notation becomes awkward. You're already familiar with the usefulness of notation; for example, when one understands the notation, it is easier to understand and verify the statement

$$(ax - b)^2 = a^2x^2 - 2abx + b^2$$

than it is to understand and verify the statement

> The square of the difference between the product of a and x and b equals the square of a times the square of x decreased by twice the product of a, b and x and increased by the square of b.

If we simply interpret "is" in the theorems as an equality, we run into problems. For example, we would then say that since $n = O(n)$ and $n + 1 = O(n)$, we would have $n = n + 1$. How can we introduce arithmetic notation and avoid such problems? The key is to redefine Θ, O and o slightly using sets. Let Θ^*, O^* and o^* be our old definitions. Our new ones are:

- $\Theta(f(n)) = \{g(n) \mid g(n) \text{ is } \Theta^*(f(n))\}$,
- $\Theta^+(f(n)) = \{g(n) \mid g(n) \text{ is } \Theta^*(f(n)) \text{ and } g(n) \text{ is positive for large } n\}$,
- $O(f(n)) = \{g(n) \mid g(n) \text{ is } O^*(f(n))\}$,
- $O^+(f(n)) = \{g(n) \mid g(n) \text{ is } O^*(f(n)) \text{ and } g(n) \text{ is positive for large } n\}$,
- $o(f(n)) = \{g(n) \mid g(n) \text{ is } o^*(f(n))\}$.

If we replace "is" with "is in", Theorems B.1 and B.2 are still correct. For example, the last part Theorem B.1(b) becomes

> If $g(n) \in \Theta(f(n))$, then $g(n) \in O(f(n))$.

We want to make two other changes:

- Replace functions, numbers, and so on with sets so that we use $\subseteq$ instead of $\in$. For example, instead of $5n \in O(n)$, we say $\{5n\} \subseteq O(n)$.
- An arithmetic operation between sets is done element by element; for example,

$$A + B = \{a + b \mid a \in A \text{ and } b \in B\}.$$

Let's rewrite parts of Theorem B.1 using this notation:

(b) If $\{g(n)\} \subseteq \Theta^+(f(n))$, then $\{g(n)\} \subseteq \Theta(f(n))$.
If $\{g(n)\} \subseteq O^+(f(n))$, then $\{g(n)\} \subseteq O(f(n))$.
If $\{g(n)\} \subseteq \Theta(f(n))$, then $\{g(n)\} \subseteq O(f(n))$.

(f) If $\{g(n)\} \subseteq \Theta(f(n))$ and $\{f(n)\} \subseteq \Theta(h(n))$, then $\{g(n)\} \subseteq \Theta(h(n))$.
If $\{g(n)\} \subseteq O(f(n))$ and $\{f(n)\} \subseteq O(h(n))$, then $\{g(n)\} \subseteq O(h(n))$.

(i) If $\{g(n)\} \subseteq \Theta^+(f(n))$ and $\{h(n)\} \subseteq O^+(f(n))$, then $\{g(n) + h(n)\} \subseteq \Theta^+(f(n))$.

We leave it to you to translate other parts and to translate Theorem B.2.

In practice, people simplify the notation we've introduced by replacing things like $\{f(n)\}$ with $f(n)$, which is good since it makes these easier to read. They also replace $\subseteq$ with $=$, which is dangerous but is, unfortunately, the standard convention. We'll abide by these conventions, but will remind you of what we're doing by footnotes in the text.

Example B.5 Using the notation The statement $f(n) \sim g(n)$ is equivalent to the statement $f(n) = g(n)(1 + o(1))$ and also to $f(n) = g(n) + o(g(n))$. The first is because $f(n)/g(n) \to 1$ if and only if $f(n)/g(n) = 1 + o(1)$. The second follows from

$$g(n)(1+o(1)) = g(n)+g(n)o(1) = g(n)+o(g(n)) \quad \text{and} \quad g(n)+o(g(n)) = g(n)+g(n)o(1) = g(n)(1+o(1)).$$

Why do we need the second of these statements?

* * * Stop and think about this! * * *

Remember that $=$ really means $\subseteq$, so the first statement shows that $g(n)(1 + o(1)) \subseteq g(n) + o(g(n))$ and the second shows that $g(n) = o(g(n)) \subseteq g(n)(1 + o(1))$. Taken together, the two statements show that the sets $g(n)(1 + o(1))$ and $g(n) + o(g(n))$ are equal and so $f(n)$ is in one if and only if it is in the other.

We can include functions of sets: Suppose S is a subset of the domain of the function F, define $F(S) = \{f(s) \mid s \in S\}$. With this notation,

$$e^{o(1)} \;=\; 1 + o(1) \;=\; e^{o(1)} \qquad \text{and} \qquad e^{O(1)} \;=\; O^+(1);$$

however, $O^+(1) \neq e^{O(1)}$. Why is this?

* * * Stop and think about this! * * *

We have $e^{-n} \in O^+(1)$ but, since $n \notin O(1)$, $e^{-n} \notin e^{O(1)}$ □

Everything we've done so far is with functions from the positive integers to the reals and we've asked what happens as $n \to \infty$. We can have functions on other sets and ask what happens when we take a different limit. For example, the definition of a derivative can be written as

$$\frac{f(x+h) - f(x)}{h} \;\sim\; f'(x) \quad \text{as } h \to 0,$$

provided $f'(x) \neq 0$. Taylor's theorem with remainder can be written

$$f(x) \;=\; \sum_{k=0}^{n} \frac{f^{k)}(0)x^k}{k!} + O(x^{n+1}) \quad \text{as } x \to 0,$$

provided $f^{(n+1)}(x)$ is well behaved near $x = 0$. Of course, this is not as good as the form in your calculus book because it says nothing about how big the error term, $O(x^{n+1})$ is for a particular function $f(x)$.

B.3 NP-Complete Problems

Computer scientists talk about "polynomial time algorithms." What does this mean? Suppose that the algorithm can handle arbitrarily large problems and that it takes $\Theta(n)$ seconds on a problem of "size" n. Then we call it a linear time algorithm. More generally, if there is a (possibly quite large) integer k such that the worst case running time on a problem of "size" n is $O(n^k)$, then we say the algorithm is polynomial time.

You may have noticed the quotes around size and wondered why. It is necessary to specify what we mean by the size of a problem. Size is often interpreted as the number of bits required to specify the problem in binary form. You may object that this is imprecise since a problem can

be specified in many ways. This is true; however, the number of bits in one "reasonable" representation doesn't differ too much from the number of bits in another. We won't pursue this further.

If the worst case time for an algorithm is polynomial, theoretical computer scientists think of this as a good algorithm. (This is because polynomials grow relatively slowly; for example, exponential functions grow much faster.) The problem that the algorithm solves is called *tractable*.

Do there exist *intractable problems*; i.e., problems for which no polynomial time algorithm can ever be found? Yes, but we won't study them here. More interesting is the fact that there are a large number of practical problems for which

- no polynomial time algorithm is known and
- no one has been able prove that the problems are intractable.

We'll discuss this a bit. Consider the following problems.

- Coloring Problem: For any $c > 2$, devise an algorithm whose input can be any simple graph and whose output answers the question "Can the graph be properly colored in c colors?"
- Traveling Salesman Problem: For any B, devise an algorithm whose input can be any $n > 0$ and any edge labeling $\lambda: \mathcal{P}_2(\underline{n}) \to \mathbb{R}$ for K_n, the complete graph on n vertices. The algorithm must answer the question "Is there a cycle through all n vertices with cost B or less?" (The cost of a cycle is the sum of $\lambda(e)$ over all e in the cycle.)
- Language Recognition Problem: Devise an algorithm whose input is two finite sets S and T and an integer k. The elements of S and T are finite strings of zeroes and ones. The algorithm must answer the question "Does there exist a finite automaton with k states that accepts all strings in S and accepts none of the strings in T?"

No one knows if these problems are tractable, but it is known that, if one is tractable, then they all are. There are hundreds more problems that people are interested in which belong to this particular list in which all or none are tractable. These problems are called *NP-complete* Many people regard deciding if the NP-complete problems are tractable to be one of the foremost open problems in theoretical computer science.

The NP-complete problems have an interesting property which we now discuss. If the algorithm says "yes," then there must be a specific example that shows why this is so (an assignment of colors, a cycle, an automaton). There is no requirement that the algorithm actually produce such an example. Suppose we somehow obtain a coloring, a cycle or an automaton which is claimed to be such an example. Part of the definition of NP-complete requires that we be able to check the claim in polynomial time. Thus we can check a purported example quickly but, so far as is known, it may take a long time to determine if such an example exists. In other words, I can check your guesses quickly but I don't know how to tell you quickly if any examples exist.

There are problems like the NP-complete problems where no one knows how to do any checking in polynomial time. For example, modify the traveling salesman problem to ask for the minimum cost cycle. No one knows how to verify in polynomial time that a given cycle is actually the minimum cost cycle. If the modified traveling salesman problem is tractable, so is the one we presented above: You need only find the minimum cost cycle and compare its cost to B. Such problems are called *NP-hard* because they are at least as hard as NP-complete problems. A problem which is tractable if the NP-complete problems are tractable is called *NP-easy*.

Some problems are both NP-easy and NP-hard but may not be NP-complete. Why is this? NP-complete problems must ask a "yes/no" type of question and it must be possible to check a specific example in polynomial time as noted in the previous paragraph.

What can we do if we cannot find a good algorithm for a problem? There are three main types of partial algorithms:

1. Almost good: It is polynomial time for all but a very small subset of possible problems. (If we are interested in all graphs, our coloring algorithm in Example B.3 is almost good for any fixed c.)
2. Almost correct: It is polynomial time but in some rare cases does not find the correct answer. (If we are interested in all graphs and a fixed c, automatically reporting that a large graph can't be colored with c colors is almost correct—but it is rather useless.) In some situations, a fast almost correct algorithm can be useful.
3. Close: It is a polynomial time algorithm for a minimization problem and comes close to the true minimum. (There are useful close algorithms for approximating the minimum cycle in the Traveling Salesman Problem.)

Some of the algorithms make use of random number generators in interesting ways. Unfortunately, further discussion of these problems is beyond the scope of this text.

Exercises

B.3.1. The *chromatic number* $\chi(G)$ of a graph G is the least number of colors needed to properly color G. Using the fact that the problem of deciding whether a graph can be properly colored with c colors is NP-complete, prove the following.

(a) The problem of determining $\chi(G)$ is NP-hard.

(b) The problem of determining $\chi(G)$ is NP-easy.
Hint. You can color G with c colors if and only if $c \geq \chi(G)$.

B.3.2. The *bin packing problem* can be described as follows. Given a set S of positive integers and integers B and K, is there a partition of S into K blocks so that the sum of the integers in each block does not exceed B? This problem is known to be NP-complete.

(a) Prove that the following modified problem is NP-easy and NP-hard. Given a set S of positive integers and an integer B, what is the smallest K such that the answer to the bin packing problem is "yes?"

(b) Call the solution to the modified bin packing problem $K(S, B)$. Prove that

$$K(S, B) \ \geq \ \frac{1}{B} \sum_{s \in S} s.$$

(c) The "First Fit" algorithm obtains an upper bound on $K(S, B)$. We now describe it. Start with an infinite sequence of boxes (bins) $B_1, B_2, \ldots$. Each box can hold any number of integers as long as their sum doesn't exceed K. Let $s_1, s_2, \ldots$ be some ordering of S. If the s_i's are placed in the B_j's, the nonempty boxes form an ordered partition of S and so the number of them is an upper bound for $K(S, B)$. For $i = 1, 2, \ldots, |S|$, place s_i in the B_j with the lowest index such that it will not make the sum of the integers in B_j exceed K. Estimate the running time of the algorithm in terms of $|S|$, B and the number of bins actually used.

(d) Call the bound on K obtained by the First Fit algorithm $FF(S, B)$. Prove that $FF(S, B) < 2K(S, B) + 1$.
Hint. When First Fit is done, which bins can be at most half full?

Notes and References

Many texts discuss the notation for rate of growth. A particularly nice introduction is given in Chapter 9 of Graham, Knuth, and Patashnik [4].

Entire textbooks are devoted primarily to the analysis of algorithms. Examples include the books by Aho, Hopcroft and Ullman [1], Baase [2], Knuth [6], Manber [7], Papadimitiriou and Steiglitz [8], and Wilf [9]. There is an extensive journal literature on NP-completeness. The classic book by Garey and Johnson [3] discusses many of the examples. Other discussions can be found in the texts by Papadimitriou and Steiglitz [8] and by Wilf [9] and in the article by Hartmanis [5].

1. Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).
2. Sara Baase, *Computer Algorithms: Introduction to Design and Analysis*, 3rd ed., Addison-Wesley (1999).
3. Michael R. Garey and David S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W.H. Freeman (1979).
4. Ronald L. Graham, Donald E. Knuth, and Oren Patashnik, *Concrete Mathematics*, 2nd ed., Addison-Wesley, Reading (1994).
5. Juris Hartmanis, *Overview of computational complexity theory*, Proceedings of Symposia in Applied Mathematics *38* (1989), 1–17.
6. Donald E. Knuth, *The Art of Computer Programming. Vol. 1: Fundamental Algorithms,* 3rd ed.; *Vol. 2: Seminumerical Algorithms,* 3nd ed.; *Vol. 3: Sorting and Searching,* 3nd ed.; Addison-Wesley (1997, 1997, 1998).
7. Udi Manber, *Introduction to Algorithms: A Creative Approach*, Addison-Wesley (1989).
8. Christos H. Papadimitriou and Kenneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover (1998).
9. Herbert S. Wilf, *Algorithms and Complexity*, Prentice-Hall (1986).

APPENDIX C

Basic Probability

This appendix is a rapid introduction to the concepts from probability theory that are needed in the text. It is not intended to substitute for a basic course in probability theory, which we strongly recommend for anyone planning either to apply combinatorics (especially in computer science) or to delve into combinatorics for its own sake.

C.1 Probability Spaces and Random Variables

For simplicity, we will limit our attention to finite probability spaces and to real-valued random variables.

Definition C.1 Finite probability space *A* **finite probability space** *is a finite set S together with a function $\Pr : S \to \mathbb{R}$ such that*

$$0 \leq \Pr(s) \leq 1 \quad \text{for all } s \in S \qquad \text{and} \qquad \sum_{s \in S} \Pr(s) = 1.$$

We call the elements of S **elementary events** *and the subsets of S* **events***. For $T \subseteq S$, we define*

$$\Pr(T) = \sum_{t \in T} \Pr(t).$$

Note that $\Pr(\emptyset) = 0$, $\Pr(S) = 1$ and $\Pr(s) = \Pr(\{s\})$ for $s \in S$.

If $\mathcal{A}(s)$ is a statement that makes sense for $s \in S$, we define $\Pr(\mathcal{A}) = \Pr(T)$, were T is the set of all $t \in S$ for which $\mathcal{A}(t)$ is true.

One often has $\Pr(s) = 1/|S|$ for all $s \in S$. In this case, $\Pr(T) = |T|/|S|$, the fraction of elements in S that lie in T. In this case, we call $\Pr$ the **uniform distribution** *on S.*

The terminology "event" can sometimes be misleading. For example, if S consists of all $2^{|A|}$ subsets of $|A|$, an elementary event is a subset of A. Suppose Pr is the uniform distribution on S. If T is the event consisting of all subsets of size k, then $\Pr(T)$ is the fraction of subsets of size k. We say that $\Pr(T)$ is the probability that a subset of A chosen *uniformly at random* has size k. "Uniformly" is often omitted and we simply say that the probability a randomly chosen subset has size k is $|T|/|S| = \binom{|A|}{k}2^{-|A|}$. In statement notation,

$$\Pr(\text{a subset has size } k) \;=\; \binom{|A|}{k}2^{-|A|}.$$

The notion of the probability of a statement being true is neither more nor less general than the notion of the probability of a subset of the probability space S. To see this, note that

- with any statement $\mathcal{A}$ we can associate the set A of elements of S for which $\mathcal{A}$ is true while
- with any subset T of S we can associate the statement "$t \in T$."

Here are some simple, useful properties of Pr. You should be able to supply the proofs by writing all probabilities as sums of probabilities of elementary events and noticing which elementary events appear in which sums.

Theorem C.1 *Suppose $(S, \Pr)$ is a probability space and A, $A_1, \ldots, A_k$ and B are subsets of S.*

(a) *If $A \subseteq B$, then $0 \le \Pr(A) \le \Pr(B) \le 1$.*

(b) $\Pr(S \setminus A) + \Pr(A) = 1$. *One also writes $S - A$, A^c and A' for $S \setminus A$.*

(c) $\Pr(A \cup B) \;=\; \Pr(A) + \Pr(B) - \Pr(A \cap B)$.

(d) $\Pr(A_1 \cup \cdots \cup A_k) \;\le\; \Pr(A_1) + \cdots + \Pr(A_k)$.

We now define a "random variable." It is neither a variable nor random, rather, it is a function:

Definition C.2 Random variable *Given a probability space $(S, \Pr)$, a* **random variable** *on the space is a function $X : S \to \mathbb{R}$.*

People often use capital letters near the end of the alphabet to denote random variables. Why the name "random variable" for a function? The terminology arose historically before probability theory was put on a mathematical foundation. For example, suppose you toss a coin 10 times and let X be the number of heads. The value of X varies with the results of your tosses and it is random because your tosses are random. In probability theory terms, if the coin tosses are fair, we can define a probability space and a random variable as follows.

- S is the set of all 2^{10} possible 10-long head-tail sequences,
- $\Pr(s) = 1/|S| = 2^{-10} = 1/1024$,
- $X(s)$ equals the number of heads in the 10-long sequence s.

The probability that you get exactly four heads can be written as $\Pr(X = 4)$, which equals $\binom{10}{4}2^{-10}$ since there are $\binom{10}{4}$ 10-long head-tail sequences that contain exactly four heads.

Definition C.3 Independence *Let $(S, \Pr)$ be a probability space and let $\mathcal{X}$ be a set of random variables on $(S, \Pr)$. We say that the random variables in $\mathcal{X}$ are* **mutually independent** *if, for every subset $\{X_1, \ldots, X_k\}$ of $\mathcal{X}$ and all real numbers $x_1, \ldots, x_k$, we have*

$$\Pr(X_1 = x_1 \text{ and } \cdots \text{ and } X_k = x_k) \;=\; \Pr(X_1 = x_1) \cdots \Pr(X_k = x_k),$$

where the probabilities on the right are multiplied. We often abbreviate "mutual independence" to "independence."

Intuitively, the concept of independence means that knowing the values of some of the random variables gives no information about the values of others. For example, consider tossing a fair coin randomly ten times to produce a 10-long sequence of heads and tails. Define

$$X_i \;=\; \begin{cases} 1, & \text{if toss } i \text{ is heads,} \\ 0, & \text{if toss } i \text{ is tails.} \end{cases} \tag{C.1}$$

Then the set $\{X_1, \ldots, X_{10}\}$ of random variables are mutually independent.

We now look at "product spaces." They arise in natural ways and lead naturally to independence.

Definition C.4 Product space *Let $(S_1, \Pr_1), \ldots, (S_n, \Pr_n)$ be probability spaces. The product space*

$$(S, \Pr) \;=\; (S_1, \Pr_1) \times \cdots \times (S_n, Pr_n) \tag{C.2}$$

is defined by $S = S_1 \times \cdots \times S_n$, a Cartesian product, and

$$\Pr((a_1, \ldots, a_n)) \;=\; \Pr_1(a_1) \cdots \Pr_n(a_n) \quad \textit{for all} \quad (a_1, \ldots, a_n) \in S.$$

We may write $\Pr(a_1, \ldots, a_n)$ instead of $\Pr((a_1, \ldots, a_n))$.

As an example, consider tossing a fair coin randomly ten times. The probability space for a single toss is the set $\{H, T\}$, indicating heads and tails, with the uniform distribution. The product of ten copies of this space is the probability space for ten random tosses of a fair coin.

Theorem C.2 Independence in product spaces *Suppose a product space $(S, \Pr)$ is given by (C.2). Let $I_1, \ldots, I_m$ be pairwise disjoint subsets of $\underline{n}$; that is, $I_i \cap I_j = \emptyset$ whenever $i \neq j$. Suppose X_k is a random variable whose value on $(a_1, \ldots, a_n)$ depends only on the values of those a_i for which $i \in I_k$. Then $X_1, \ldots, X_m$ are mutually independent.*

We omit the proof.

Continuing with our coin toss example, Let $I_i = \{i\}$ and define X_i by (C.1). By the theorem, the X_i are mutually independent.

C.2 Expectation and Variance

Definition C.5 Expectation *Let $(S, \Pr)$ be a probability space The* **expectation of a random variable** *X is*

$$\mathbf{E}(X) \;=\; \sum_{s \in S} X(s) \Pr(s).$$

The expectation is also called the **mean**.

Let $R \subset \mathbb{R}$ be the set of values taken on by X. By collecting terms in the sum over S according to the value of $X(s)$ we can rewrite the definition as

$$\mathbf{E}(X) \;=\; \sum_{r \in R} r \Pr(X = r).$$

In this form, it should be clear the expected value of a random variable can be thought of as its average value.

Definition C.6 Variance *Let $(S, \Pr)$ be a probability space The* **variance of a random variable** *X is*

$$\mathbf{var}(X) \;=\; \sum_{s \in S} (X(s) - \mathbf{E}(X))^2 \Pr(s) \;=\; \sum_{r \in R} (r - \mathbf{E}(X))^2 \Pr(X = r),$$

where $R \subset \mathbb{R}$ is the set of values taken on by X.

The variance of a random variable measures how much it tends to deviate from its expected value. One might think that

$$\sum_{r \in R} |r - \mathbf{E}(X)| \Pr(X = r)$$

would be a better measure; however, there are computational and theoretical reasons for preferring the variance.

We often have a random variable X that takes on only the values 0 and 1. Let $p = \Pr(X = 1)$. You should be able to prove that $\mathbf{E}(X) = p$ and $\mathbf{var}(X) = p(1 - p)$.

The following theorem can be proved be algebraic manipulations of the definitions of mean and variance. Since this is an appendix, we omit the proof.

Theorem C.3 Properties of Mean and Variance *Let $X_1, \ldots, X_k$ and Y be random variables on a probability space $(S, \Pr)$.*

(a) *For real numbers a and b, $\mathbf{E}(aY + b) = a\,\mathbf{E}(Y) + b$ and $\mathbf{var}(aY + b) = a^2\,\mathbf{var}(Y)$.*

(b) $\mathbf{var}(Y) \;=\; \mathbf{E}\big((Y - \mathbf{E}(Y))^2\big) \;=\; \mathbf{E}(Y^2) - \big(\mathbf{E}(Y)\big)^2$.

(c) $\mathbf{E}(X_1 + \cdots + X_k) \;=\; \mathbf{E}(X_1) + \cdots + \mathbf{E}(X_k)$.

(d) *If the X_i are independent, then $\mathbf{var}(X_1 + \cdots + X_k) = \mathbf{var}(X_1) + \cdots + \mathbf{var}(X_k)$.*

Chebyshev's Theorem tells us that it is unlikely that the value of a random variable will be far from its mean, where the "unit of distance" is the square root of the variance. (The square root of $\mathbf{var}(X)$ is also called the *standard deviation* of X and is written σ_X.)

Theorem C.4 Chebyshev's inequality *If X is a random variable on a probability space and $t \geq 1$, then*

$$\Pr\Big(|X - \mathbf{E}(X)| > t\sqrt{\mathbf{var}(X)}\Big) \;<\; \frac{1}{t^2}. \tag{C.3}$$

For example, if a fair coin is tossed n times and X is the number of heads, then one can show that

$$\mathbf{E}(X) = n/2 \quad \text{and} \quad \mathbf{var}(X) = n/4. \tag{C.4}$$

Chebyshev's inequality tells us that X is not likely to be many multiples of $\sqrt{n}$ from $n/2$. Specifically, it says that

$$\Pr\Big(|X - n/2| > (t/2)\sqrt{n}\Big) \;<\; \frac{1}{t^2}.$$

Let's use Theorem C.3 to prove (C.4). Let Y_k be a random variable which is 1 if toss k lands heads and is 0 if it lands tails. By the definition of mean,

$$\mathbf{E}(Y_k) \;=\; 0\Pr(Y_k = 0) + 1\Pr(Y_k = 1) \;=\; 0 + 1/2 \;=\; 1/2$$

and, by the Theorem C.3(b) and the observation that $Y_k^2 = Y_k$,

$$\mathbf{var}(Y_k) \;=\; \mathbf{E}(Y_k^2) - \mathbf{E}(Y_k)^2 \;=\; \mathbf{E}(Y_k) - \mathbf{E}(Y_k)^2 \;=\; 1/2 - (1/2)^2 \;=\; 1/4.$$

Notice that $X = Y_1 + Y_2 + \cdots + Y_n$ and the Y_k are independent since the coin is tossed randomly. By Theorem C.3(c) and (d), we have (C.4).

APPENDIX D

Partial Fractions

We will discuss those aspects of partial fractions that are most relevant in enumeration. Although not necessary for our purposes, the theoretical background of the subject consists of two easily discussed parts, so we include it. The rest of this appendix is devoted to computational aspects of partial fractions.

Theory

The following result has many applications, one of which is to the theory of partial fractions.

Theorem D.1 Fundamental Theorem of Algebra *If $p(x)$ is a polynomial of degree n whose coefficients are complex numbers, then $p(x)$ can be written as a product of linear factors, each of which has coefficients which are complex numbers.*

We will not prove this.

In calculus classes, one usually uses a corollary of this theorem: If the coefficients of $p(x)$ are real numbers, then it is possible to factor $p(x)$ into a product of linear and quadratic factors, each of which has real coefficients. We will not use the corollary because it is usually more useful in combinatorics to write $p(x)$ as a product of linear factors.

By the Fundamental Theorem of Algebra, every polynomial $p(x)$ can be factored in the form

$$p(x) \;=\; Cx^n(1-\alpha_1 x)^{n_1}(1-\alpha_2 x)^{n_2}\cdots(1-\alpha_k x)^{n_k}, \tag{D.1}$$

where the α_i's are distinct nonzero complex numbers. Although this can always be done, it is, in general, *very difficult* to do. In (D.1), the α_i's are the reciprocals of the nonzero roots of $p(x)=0$ and n_i is the "multiplicity" of the root $1/\alpha_i$.

Suppose that $p(x)=p_1(x)p_2(x)\cdots p_k(x)$ and $q(x)$ are polynomials such that

- the degree of $q(x)$ is less than the degree of $p(x)$;
- none of the $p_i(x)$ is a constant;
- no pair $p_i(x)$ and $p_j(x)$ have a common root.

The Chinese Remainder Theorem for polynomials asserts that there exist unique polynomials $q_1(x),\ldots,q_k(x)$ (depending on $q(x)$ and the $p_i(x)$) such that

- the degree of $q_i(x)$ is less than the degree of $p_i(x)$ for all i;
- if the coefficients of $q(x)$ and the $p_i(x)$ are rational, then so are the coefficients of the $q_i(x)$;
- $\dfrac{q(x)}{p(x)} = \dfrac{q_1(x)}{p_1(x)} + \dfrac{q_2(x)}{p_2(x)} + \cdots + \dfrac{q_k(x)}{p_k(x)}$.

This is called a *partial fraction expansion* of $q(x)/p(x)$. For combinatorial applications, we take the $p_i(x)$'s to be of the form $(1-\alpha_i x)^{n_i}$.

Suppose we have been able to factor $p(x)$ as shown in (D.1). In our applications, we normally have $n = 0$, so we will assume this is the case. We can also easily remove the factor of C by dividing $q(x)$ by C. Let $p_i(x) = (1-\alpha_i x)^{n_i}$. With some work, we can obtain a partial fraction expansion for $q(x)/p(x)$. Finally, with a bit more work, we can rewrite $q_i(x)/(1-\alpha_i x)^{n_i}$ as

$$\frac{\beta_{i,1}}{1-\alpha_i x} + \frac{\beta_{i,2}}{(1-\alpha_i x)^2} + \cdots + \frac{\beta_{i,n_i}}{(1-\alpha_i x)^{n_i}},$$

where the $\beta_{i,j}$'s are complex numbers. (We will not prove this.)

Authors of calculus texts prefer a different partial fraction expansion for $q(x)/p(x)$. In the first place, as we already noted, they avoid complex numbers. This can be done by appropriately combining factors in (D.1). In the second place, they usually prefer that the highest degree term of each factor have coefficient 1, unlike combinatorialists who prefer that the constant term be 1.

Computations

The computational aspect of partial fractions has two parts. The first is the factoring of a polynomial $p(x)$ and the second is obtaining a partial fraction expansion.

In general, factoring is difficult. The polynomials we deal with can be factored by using the factoring methods of basic algebra, including the formula for the roots of a quadratic equation. The latter is used as follows:

$$Ax^2 + Bx + C \;=\; A(x-r_1)(x-r_2) \quad \text{where} \quad r_1, r_2 \;=\; \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}.$$

We will not review basic algebra methods. There is one unusual aspect to the sort of factoring we want to do in connection with generating functions. We want to factor $p(x)$ so that it is a product of a constant and factors of the form $1 - cx$. This can be done by factoring the polynomial $p(1/y)y^n$, where n is the degree of $p(x)$. The examples should make this clearer.

Example D.1 A factorization Factor the polynomial $p(x) = 1 - x - 4x^3$. Following the suggestion, we look at

$$r(y) \;=\; p(1/y)y^3 = y^3 - y^2 - 4.$$

Since $r(2) = 0$, $y - 2$ must be a factor of $r(y)$. Dividing it out we obtain $y^2 + y + 2$. By the quadratic formula, the zeroes of $y^2 + y + 2$ are $(-1 \pm \sqrt{-7}\,)/2$. Thus

$$r(y) \;=\; \Big(y-2\Big)\Big(y - \frac{-1+\sqrt{-7}}{2}\Big)\Big(y - \frac{-1-\sqrt{-7}}{2}\Big).$$

Since $p(x) = x^3 r(1/x)$, we finally have

$$p(x) \;=\; \Big(1-2x\Big)\Big(1 - \frac{-1+\sqrt{-7}}{2}x\Big)\Big(1 - \frac{-1-\sqrt{-7}}{2}x\Big). \quad \square$$

Example D.2 Partial fractions for a general quadratic We want to expand $q(x)/p(x)$ in partial fractions when $p(x)$ is a quadratic with distinct roots and $q(x)$ is of degree one or less. We will assume that $p(x)$ has been factored:

$$p(x) \;=\; (1-ax)(1-bx).$$

Since $p(x)$ has distinct roots, $a \neq b$.

Let us expand $1/p(x)$. We can write

$$\frac{1}{(1-ax)(1-bx)} \;=\; \frac{u}{1-ax} + \frac{v}{1-bx}, \tag{D.2}$$

where u and v are numbers that we must find. (If a and b are real, then u and v will be, too; however, if a or b is complex, then u and v could also be complex.) There are various ways we could find u and v. We'll show you two methods.

The straightforward method is to clear (D.2) of fractions and then equate powers of x. Here's what happens: Since

$$1 \;=\; u(1-bx) + v(1-ax) = (u+v) - (bu+av)x,$$

we have

$$1 \;=\; u+v \qquad \text{and} \qquad 0 \;=\; -(bu+av).$$

The solution to these equations is $u \;=\; a/(a-b)$ and $v \;=\; b/(b-a)$.

Another method for solving (D.2) is to multiply it by $1-ax$ and then choose x so that $1-ax=0$; i.e., $x = 1/a$. After the first step we have

$$\frac{1}{1-bx} \;=\; u + \frac{v(1-ax)}{1-bx}. \tag{D.3}$$

When we set $1-ax=0$, the last term in (D.3) disappears—that's why we chose that value for x. Substituting in (D.3), we get

$$u \;=\; \frac{1}{1-b/a} = \frac{a}{a-b}.$$

By the symmetry of (D.2), v is obtained simply by interchanging a and b.

We have shown by two methods that

$$\frac{1}{(1-ax)(1-bx)} \;=\; \frac{\frac{a}{a-b}}{1-ax} + \frac{\frac{b}{b-a}}{1-bx}. \tag{D.4}$$

By either of these methods, one can show that

$$\frac{x}{(1-ax)(1-bx)} \;=\; \frac{\frac{1}{a-b}}{1-ax} + \frac{\frac{1}{b-a}}{1-bx}. \tag{D.5}$$

We leave this as an exercise. You can save yourself quite a bit of work in partial fraction calculations if you use (D.4) and (D.5). □

Example D.3 A specific quadratic

Let's expand

$$\frac{1+3x}{(1+2x)(1-3x)}$$

by partial fractions. Using (D.4) and (D.5) with $a=-2$ and $b=3$, we easily obtain

$$\begin{aligned}\frac{1+3x}{(1+2x)(1-3x)} &= \frac{1}{(1+2x)(1-3x)}+\frac{3x}{(1+2x)(1-3x)}\\ &= \frac{2/5}{1+2x}+\frac{3/5}{1-3x}+\frac{-3/5}{1+2x}+\frac{3/5}{1-3x}\\ &= \frac{-1/5}{1+2x}+\frac{6/5}{1-3x}.\end{aligned} \tag{D.6}$$

To see how much effort has been saved, you are encouraged to derive this result without using (D.4) and (D.5). □

Example D.4 A factored cubic

Let's expand

$$\frac{1+3x}{(1-x)(1+2x)(1-3x)}$$

by partial fractions. We begin by factoring out $1-x$ and using (D.6). Next we use some algebra and then apply (D.4) twice. Here it is.

$$\begin{aligned}\frac{1+3x}{(1-x)(1+2x)(1-3x)} &= \frac{1}{1-x}\left(\frac{-1/5}{1+2x}+\frac{6/5}{1-3x}\right)\\ &= \frac{-1/5}{(1-x)(1+2x)}+\frac{6/5}{(1-x)(1-3x)}\\ &= \frac{(-1/5)(1/3)}{1-x}+\frac{(-1/5)(2/3)}{1+2x}+\frac{(6/5)(-1/2)}{1-x}+\frac{(6/5)(3/2)}{1-3x}\\ &= \frac{-2/3}{1-x}+\frac{-2/15}{1+2x}+\frac{9/5}{1-3x}.\end{aligned}$$

Notice how we were able to deal with the cubic denominator by iterating the method for dealing with a quadratic denominator. This will work in any situation as long as the denominator has no repeated factors. □

Example D.5 A squared quadratic

Let's expand

$$\frac{1+3x}{(1+2x)^2(1-3x)^2}. \tag{D.7}$$

Before tackling this problem, let's do the simpler case where the numerator is one:

$$\frac{1}{(1+2x)^2(1-3x)^2} = \left(\frac{1}{(1+2x)(1-3x)}\right)^2. \tag{D.8}$$

Using (D.4), this becomes

$$\left(\frac{2/5}{1+2x}+\frac{3/5}{1-3x}\right)^2,$$

which can be expanded to

$$\frac{4/25}{(1+2x)^2}+\frac{9/25}{(1-3x)^2}+\frac{12/25}{(1+2x)(1-3x)}.$$

The first two terms are already in standard partial fraction form and you should have no trouble expanding the last.

How does this help us with (D.7) since we still have a $3x$ left in the numerator? We can cheat a little bit and allow ourselves to write the expansion of (D.7) as simply $1+3x$ times the expansion of (D.8). This does not cause any problems in the applications of partial fractions that we are interested in other than slightly more complicated answers. □

Example D.6 Another problem How can we expand $1/(1-x)^2(1-2x)$ by partial fractions? Do any of our earlier tricks work? Yes, we simply write

$$\frac{1}{(1-x)^2(1-2x)} \;=\; \frac{1}{1-x}\left(\frac{1}{(1-x)(1-2x)}\right)$$

and continue as in Example D.4. □

As you should have seen by now, a bit of cleverness with partial fractions can save quite a bit of work. Another trick worth remembering is to use letters in place of complicated numbers. We conclude with an example which illustrates another trick.

***Example D.7 More tricks** Expand

$$\frac{x+6x^2}{1-x-4x^3} \tag{D.9}$$

in partial fractions.

We'll write the denominator as

$$(1-2x)(1+x+2x^2) \;=\; (1-2x)(1-cx)(1-dx)$$

where

$$c \;=\; \frac{-1+\sqrt{-7}}{2} \qquad \text{and} \qquad d \;=\; \frac{-1-\sqrt{-7}}{2},$$

a factorization found in Example D.1.

There are various ways we can attack this problem. Let's think about this.

- An obvious approach is to use the method of Example D.4. The $6x^2$ causes a bit of a problem in the numerator because the first step in expanding

  $$\frac{x+6x^2}{1+x+2x^2}$$

  is to divide so that the numerator is left as a lower degree than the denominator. If we take this approach, we should carry along c and d as much as possible rather than their rather messy values. Also, since they are zeros of y^2+y+2, we have the equations $c^2=-c-2$ and $d^2=-d-2$, which may help simplify some expressions. Also, $cd=2$ and $c+d=-1$. We leave this approach for you to carry out.
- We could use the previous idea after first removing the factor of $x+6x^2$ and then reintroducing it at the end. The justification for this is the last paragraph of Example D.5.
- Another approach is to write $x+6x^2 \;=\; x(1+6x)$. We can obtain partial fractions for $(1+6x)/(1+x+x^2)$ using (D.4) and (D.5). This result can be multiplied by $x/(1-2x)$ and expanded by (D.5).
- A different approach is to remove the $1-2x$ partial fraction term from (D.9), leaving a quadratic denominator. The resulting problem can then be done by our trusty formulas (D.4) and (D.5). Let's do this. We have

  $$\frac{x+6x^2}{(1-2x)(1+x+2x^2)} \;=\; \frac{u}{1-2x}+\frac{v}{1-cx}+\frac{w}{1-dx}. \tag{D.10}$$

Applying the trick of multiplying by $1-2x$ and setting $1-2x=0$, we have

$$u = \left.\frac{x+6x^2}{1+x+2x^2}\right|_{x=1/2} = 1.$$

Subtracting $u/(1-2x)$ from both sides of (D.10), we obtain

$$\begin{aligned}\frac{v}{1-cx}+\frac{w}{1-dx} &= \frac{x+6x^2}{(1-2x)(1+x+2x^2)}-\frac{1}{1-2x}\\ &= \frac{-1+4x^2}{(1-2x)(1+x+2x^2)}\\ &= \frac{-1-2x}{1+x+2x^2}.\end{aligned}$$

The last of these can now be expanded by (D.4) and (D.5). The cancellation of a factor of $1-2x$ from the numerator and denominator was not luck. It had to happen if our algebra was correct because we were removing the $1-2x$ partial fraction term. □

Since computing partial fractions can involve a lot of algebra, it is useful to have an algebra package do the computations. If that's not feasible, it's a good idea to check your calculations. This can be done in various ways

- Use a graphing calculator to plot the partial fraction expansion and the original fraction and see if they agree.
- Reverse the procedure: combine the partial fractions into one fraction and see if the result equals the original fraction.
- Compute several values to see if the value of the partial fraction expansion agrees with the value of the original fraction. For $p(x)/q(x)$, it suffices to compute $\max(\deg(q(x)),\ \deg(p(x))+1)$ values.

If you wish to practice working problems involving partial fractions, look in the partial fractions section of any complete calculus text.