Appendix A

Mathematical Notation and Background

The design and analysis of algorithms requires familiarity with certain concepts from mathematics such as functions, sets, various summation formulas, and so forth. In this appendix we review some of these commonly used concepts and establish appropriate notation. Various other mathematical concepts are briefly discussed, including complex numbers, modular arithmetic, random walks, and eigenvectors.

**A.1 Basic Mathematical and Built-in Functions**

Certain functions occur frequently in the design and analysis of algorithms and are usually implemented as built-in functions in most high-level languages. Two of the most commonly used functions are the ceiling ⎡*x*⎤ and floor ⎣*x*⎦, defined to be the smallest integer greater than or equal to *x* and the largest integer smaller than or equal to *x*, respectively. For example, ⎡1.23⎤ = 2, and ⎣1.23⎦ = 1. For a positive *x*, ⎣*x*⎦ is obtained from *x* by truncating its decimal part. Note that *x* – 1 < ⎣*x*⎦ ≤ *x*, and *x* ≤ ⎡*x*⎤ < *x* + 1.

When ⎡*x*⎤ and ⎣*x*⎦ are invoked in our pseudocode as built-in functions, we often retain the mathematical notation for such invocations rather than using names such as **ceiling**(*x*) and **floor**(*x*), respectively. On the other hand, for functions such as , we usually use the name **sqrt**. The following is a list of names and definitions for some of the more commonly used built-in functions in our pseudocode.

**sqrt**(*x*) = , *x* a nonnegative real number

**abs** (*x*) = |*x*| =  *x* a real number.

*a* **mod** *b* = *a* – *b* ⎣*a*/*b*⎦, *a* and *b* integers, *b* ≠ 0.

**odd**(*n*) = **.true.** if, and only if, *n* **mod** 2 = 1, *n* an integer.

**even**(*n*) = **.true.** if, and only if, *n* **mod** 2 = 0, *n* an integer.

Note that for positive integers *a* and *b*, *a* **mod** *b* is the remainder when *a* is divided by *b*.

Another important function in algorithms is the logarithm function. Given a base *b* > 1, the logarithm of *x* to the base *b*, denoted by log*bx*, is defined to be the functional inverse of the exponential function *bx*. In other words, log*bx*, *x* > 0, is defined to be the power to which the base *b* must be raised to equal *x*, so that

 (A.1.1)

We illustrate the functional inverse relationship of 2*x* and log2*x* in Figure A.1. Note that their graphs are the reflections of one another about the line *y* = *x*. Since 2*x* grows very rapidly, it follows that log2*x* grows very slowly.



Graphs of *y* = log2*x*, *y* = *x*, and *y* = 2*x*

**Figure A.1**

The differences between logarithmic growth (*y* = log2*x*), linear growth (*y* = *x*), and exponential growth (*y* = 2*x*) are dramatic (see Figure A.2). The first few entries for *x* in Figure A.2 were obtained by doubling the previous value. Note that doubling the input to the function log2*x* only increases its output by one. On the other hand, doubling the input to the function 2*x* results in squaring its output. This is indeed a dramatic difference!



Table of values of *y* = *x*, *y* = log2*x*, and *y* = 2*x*

**Figure A.2**

The exponential function *bx* has the following fundamental properties:

 (A.1.2)

 (A.1.3)

The following two properties of log*bx* corresponding to (A.1.2) and (A.1.3) are immediately obtained by raising both sides to the power *b* and using (A.1.1).

 (A.1.4)

 (A.1.5)

The following useful formula allows us to change from one base to another:

 (A.1.6)

The most commonly used bases are 2, *e*, and 10. When the base is *e*, the logarithm is referred to as the *natural* logarithm and is denoted by ln *x*.

**A.2 Modular Arithmetic**

Modular arithmetic is often used in cryptography to compute a cryptographic key. In particular, it is used in Chapter 9 in the design of the RSA public-key cryptosystem. The operations of addition and multiplication for the set *Zn* = {0, …, *n* – 1} of integers (*residues*) modulo *n* are defined the same as over the integers, but the result *x* of each operation is reduced by replacing *x* with the remainder *r* when *x* is divided by *n.* We denote this remainder by *x mod n*. We write

*x* ≡ *y* (mod *n*)

if *x* – *y* is divisible by *n*; that is, if both *x* and *y* have the same remainder upon dividing by *n*. It is easily verified that *Zn* satisfies the following commutative ring properties:

*Addition is commutative, associative and every element has an inverse, so that Zn is a commutative (Abelian) group under addition:*

1. *x* + *y* ≡ *y* + *x* (mod *n*)
2. (*x* + *y*) + *z* ≡ *x* + (*y* + *z*) (mod *n*)
3. *x* + (– *x*) ≡ 0 (mod *n*)

*Multiplication is commutative and associative:*

1. *x* \* *y* ≡ *y* \* *x* (mod *n*)
2. (*x* \* *y*) + *z* ≡ *x* + (*y* + *z*) (mod *n*)

*Multiplication distributes over addition:*

1. *x*\*(*y* + *z*) ≡ *x*\**y* + *x*\**z* (mod *n*)

In the case when *n* is prime, *Zn* is also a commutative group under multiplication, so that it determines a field known as the Galois field of integers modulo *n*, denoted by GF(*n*). It is easily verified that the relation *R* on the set *Z* of all integers, given by *xRy* if and only if *x* ≡ *y* (mod *n*), is an equivalence relation, and an element *x* from *Zn* = {0, …, *n* – 1} can be identified with the equivalence class [*x*] = {…, *x* – 2*n*, *x* – *n, x*, *x* + *n*, *x* + 2*n*, …} of all integers *y* such that *x* ≡ *y* (mod *n*).

**A.3 Some Summation Formulas**

Summations involving arithmetic and geometric progressions occur frequently in the analysis of algorithms. Given real numbers *a*, *d*, and a positive integer *n*, the sum of the first *n* terms of the arithmetic progression with leading term *a* and difference *d* is given by

 (A.3.1)

Formula (A.3.1) is easily proved by adding the progression to a copy of itself, but where we add the *i*th term in the original progression to the (*n* – *i* + 1)st term in the copy. Then each of the resulting *n* summands equals (2*a* + (*n* – 1)*d*). In algorithm analysis, the formula (A.3.1) occurs most often with *a* = *d* = 1, in which case it reduces to

 (A.3.2)

Given a real number *x* and a positive integer *n*, the sum of the first *n* terms of the geometric progression 1, *x*, *x*2, . . . , *xp*, . . . is given by

 (A.3.3)

Formula (A.3.3) follows easily by multiplying both sides by *x* – 1 and simplifying the left-hand side. An important special case in the analysis of algorithms occurs when *x* = 2. Then (A.3.3) reduces to

 (A.3.4)

Another way of interpreting formula (A.3.4) is to note that the base-two expansion of the number 2*n* – 1 consists of a sequence of *n* ones.

A formula that is useful when analyzing the average behavior of algorithms is obtained by replacing *n* by *n* + 1 in (A.3.3) and then differentiating both sides of the equation, yielding

 (A.3.5)

For *x* a real number, –1 < *x* < 1, taking the limit of both sides of (A.3.3) and (A.3.5) as *n* approaches infinity yields the following two useful power series formulas.

 (A.3.6)

 (A.3.7)

Of course, (A.3.7) also can be obtained from (A.3.6) by differentiating both sides of the equation.

**A.4 Binomial Coefficients**

We now consider two combinatorial quantities that arise often in counting arguments and in probability computations. Suppose we have a finite set *S* having *n* elements. For any *k* between 0 and *n*, the number of ways to make an *ordered* choice of *k* elements from *S* is given by the product

 (A.4.1)

In particular, note that *n*! = *n(n)*. The following inequality will be useful

 A.4.2)

Let *C*(*n*, *k*) denote the number of ways to make an *unordered* choice of *k* elements from *S*. It then follows that

 (A.4.3)

since *n(k)* stood for the number of ordered choices of *k* elements, and there are *k*! different orderings of these *k* elements. Hence, using (A.4.1) and (A.4.3), we have

 (A.4.4)

We refer to *C*(*n*, *k*) as “*n* choose *k*.” We often use the alternate notation  for *C*(*n*, *k*). As an illustration, consider the set *S* = {*a*, *b*, *c*, *d*}. There are twelve ways to choose an ordered subset of two elements from *S*, namely,



but only 12/2!=6 ways to choose an unordered two-element subset, namely,



Thus, 4(2) = 12, and 

The number  is also called a *binomial coefficient* because of its appearance in the binomial expansion:

 (A.4.5)

Taking the partial derivative of both sides of (A.4.5) with respect to *x* and then multiplying by *x* yields the following useful identity

 (A.4.6)

The binomial coefficients give us yet another interpretation of the formula

1 + 2 + . . . + *n* = *n*(*n* + 1)/2. Combining (A.3.2) and (A.4.4) yields

 (A.4.7)

A direct verification of (A.4.7) can be given as follows. Let {1, 2, . . . , *n* + 1} be the (*n* + 1)-element set. Then there are *n* ways to choose a two-element subset containing 1, *n* –1 ways to choose a two-element subset containing 2 but not 1, *n* –2 ways to choose a two-element subset containing 3 but not 1 and 2, and so forth.

The binomial expansion (A.4.5) has an important generalization (due to Newton) to the case where *a* = 1 and |*x*| < 1 and *n* is an arbitrary real number *c*.

 (A.4.8)

where the generalized binomial coefficient is defined by

 (A.4.9)

A problem arises when attempting to implement an algorithm for *C*(*n*,*k*) based directly on (A.4.4), namely, the rapid growth of the factorial function *k*!. For example, many compilers store integer variables using four bytes of storage, so that the largest integer that can be stored without overflow is 231 – 1 = 2,147,483,648. This value is already exceeded by 13!. One way around this is to calculate *C*(*n*,*k*) as the product of the *k* fractions . This avoids integer overflow, but has the disadvantage of using real arithmetic and introducing round-off errors if we simply compute and multiply the above fractions. However, if at the *i*th stage we multiply the previous result by the numerator (*n* – *i*) and then divide by the denominator *i* + 1, *i* = 1, …, *k*-1, we avoid round-off errors. In applications where you desire a table of all the binomial coefficients *C*(*j*,*k*) for *j* = 1, …, *n*, *k* = 0, …., *j*, it is best to use Pascal’s recurrence relation (2.2.4):

 (A.4.10)

To verify (A.23), consider a set of *n* elements *S* = {*s*1, *s*2, . . . , *sn*}. The initial conditions in (A.23) are true since the only subset having zero elements is the empty set and the only subset having *n* elements is *S* itself. To prove the recurrence relation, consider the element *s*1. The subsets of *S* having *k* elements fall into two disjoint classes: those that contain *s*1 and those that don’t. If *s*1 is in a subset of size *k*, then the remaining *k* – 1 elements of the subset must be chosen from the *n* – 1 elements in the subset *S*\{*s*1}. The number of such choices is *C*(*n* – 1, *k* – 1). On the other hand, if *s*1 is not in a subset of size *k*, then all *k* elements of this subset must be chosen from *S*\{*s*1}. The number of such choices is *C*(*n* – 1,*k*). Thus, the total number of ways to choose a subset of *k* elements is the sum of *C*(*n* – 1,*k* – 1) and *C*(*n* – 1,*k*), which establishes the recurrence relation (A.4.10).

In Figure A.3, we show the famous *Pascal’s Triangle*, where each successive row in the triangle is obtained from the row above by using the recurrence (A.4.10). The initial conditions yield the 1s forming two sides of the (infinite) triangle. The famous philosopher and mathematician Blaise Pascal exploited various properties of this triangle in a paper appearing in 1654. However, the triangle itself was already known to the Chinese as early as the eleventh century.



Pascal’s Triangle generated by *C*(*n*,*k*) = *C*(*n* – 1,*k* – 1) + *C*(*n* – 1,*k*), **init. cond.** *C*(*n*,0) = *C*(*n*,*n*) = 1

**Figure A.3**

**A.5 Sets**

Various definitions and notation from the elementary theory of sets are needed in our discussion of asymptotic behavior given in Chapter 3. These notions from set theory are also required in our discussion of elementary probability theory given in Chapter 6 and Appendix E. In addition to utilizing sets in the mathematical analysis of algorithms, there are many important algorithms that implement union and find operations for disjoint sets.

Whenever we talk about a set, we assume that the set is contained in a certain *universal set U*. The set *U* is always clear from context (all edges in a graph, all real-valued functions defined on the nonnegative integers, all outcomes of an experiment, etc.). The set of real numbers and the set of nonnegative integers occur often, and we denote them by R and N, respectively.

Given a set *A*, we indicate that *x* is a member of (belongs to) *A* by *x* ∈ *A*. We write

*x* ∉ *A* when *x* is not a member of (does not belong to) *A*. The set having no members, called the *empty set*, is denoted by ∅. Given two sets *A* and *B*, we say that *A* is a *subset* of *B*, denoted by *A* ⊆ *B*, if every member of *A* is also a member of *A.* We also say that *A* *is contained in A.* If *A* ⊆ *B* and *A* ≠ *B*, then we say that *A* is a *proper* subset of *B*, denoted by *A* ⊂ *A.* We also say that *A* is *strictly* contained in *A.* The *union* of *A* and *B*, denoted by

*A* ∪ *B*, is the set of all elements that are members of either *A* or *B* (or both). The *intersection* of *A* and *B*, denoted by *A* ∩ *B*, is the set of all elements that are members of both *A* and *A.* The sets *A* and *B* are called *disjoint*, if *A* ∩ *B* = ∅. The *difference* between *A* and *B*, denoted by *A*\*B*, is the set of all members of *A* that are not members of *A.* The *complement* of *A*, denoted by *Ac*, is the set of all members of the universal set *U* that are not members of *A*, that is *Ac* = *U*\*A*. The *Cartesian product A* × *B* is the set of all ordered pairs {(*a*,*b*): *a* ∈ *A*, *b* ∈ *B*}.

Figure A.4 gives *Venn diagrams* illustrating subset, union, intersection, difference, and complement.



Venn diagrams for set operations

**Figure A.4**

**A.6 Complex Numbers**

Often when solving problems whose inputs and outputs are real numbers, it is useful in intermediate stages to utilize the complex numbers. For example, in Chapter 5 we develop a fast algorithm for computing the symbolic product of two input polynomials with *real* coefficients (so that the output product polynomial also has *real* coefficients) by using the Fast Fourier Transform (FFT) to evaluate the polynomials at the set of *n*th roots of unity. We now quickly review the necessary theory of complex numbers required by the FFT and similar algorithms.

The use of counting numbers is part of the earliest written historical record. However, the use of negative numbers occurs much later. There are understandable reasons for the relatively late appearance of negative numbers, but their introduction apparently caused little controversy. This was not the case, however, when numbers such as  were introduced. Mathematicians wanted to solve simple equations such as *x*2 + 1 = 0, which had no solutions in the real numbers. Thus, the so-called *complex numbers* of the form *a* + *ib* were introduced, where *i* =  stood for a number whose square was –1. To make the introduction of *i* more acceptable, it was called an *imaginary number*. More generally, *ib* is referred to as the *imaginary part* of the complex number *a* + *ib* and *a* is called the *real part*.

The early apologies made for *i* =  are somewhat amusing nowadays, since the complex numbers *a* + *ib* can be formally introduced simply as pairs (*a*, *b*) of real numbers (so that *a* + *ib* corresponds to (*a*, *b*)), together with arithmetic operations of addition, subtraction, multiplication (denoted by juxtaposition), and reciprocation defined by

Definitions

**I.** (*a*, *b*) ± (*c*, *d*) = (*a* ± *c*, *b* ± *d*),

**II.** (*a*, *b*)(*c*, *d*) = (*ac* – *bd*, *ad* + *bc*),

**III.** (*a*, *b*)–1 = (*a*/(*a*2 + *b*2), – *b*/(*a*2 + *b*2)).

The right-hand sides of I, II, and III use the usual arithmetic operations on real numbers, and the left-hand sides define the corresponding operation (using the same symbols) on the complex numbers. Using III, division is defined as (*a*, *b*)/(*c*, *d*) = (*a*, *b*)(*c*, *d*)–1. The definitions II and III seem somewhat unmotivated, but they arise naturally as explained shortly. Note that the real number *a* can be identified with the complex number as (*a*, 0). The rules I, II and III when restricted to the pairs (*a*, 0) as *a* varies over the real numbers coincide with the ordinary arithmetic rules for real numbers. Thus we have naturally extended the real numbers to the larger set of complex numbers. Moreover, (0, 1)2 = (–1, 0), so that we have indeed found , which we denote by *i*. Of course, –*i* = (0, –1) also squares to –1, so that we now have two complex numbers whose square is –1, whereas the reals have none.

Note that under our identification of *b* with (*b*, 0) (and using *i* to denote (0, 1)), we can think of *ib* as the product of (0, 1) with (*b*, 0), that is, *ib* corresponds to (0, *b*). In the same way, the addition (*a*, 0) + (0, *b*) can be written as *a* + *ib*, and the multiplication (*c*, 0)(*a*, *b*) = (*ca*, *cb*) can be viewed as *c*(*a* + *ib*) = *ca* + *icb*. With similar identifications, we see how definition II arises naturally from the distributive and commutative laws



Definition III for reciprocals takes its motivation from



Just as we consider the real numbers as corresponding to points on a straight line, we think of the complex numbers as corresponding to points in a plane. Thus, we identify the complex numbers *z* = *x* + *iy* with points (two-dimensional vectors) (*x*, *y*) in the *xy* plane, with the reals *x* identified with the points (*x*, 0) on the *x* axis. The *modulus of z* = *x* + *iy*, denoted by |*z*|, is defined to be the distance from (*x*, *y*) to the origin (0, 0), that is,



Note that the modulus of a point on the *x* axis corresponds to the absolute value of the point.

Using the analogy with polar coordinates in the plane, given the complex number

*z* = *x* + *iy*, it is useful to write *z* as

 (A.6.1)

where  is a point on the unit circle. Hence,  can be written as cos(*θ*) + *i* sin(*θ*) for an appropriate angle *θ* between the vector determined by *z* and the *x* axis (see Figure A.5), so that (A.6.1) becomes

 (A.6.2)

In the representation (A.6.2), the angle *θ* is referred to as an *argument* of *z*. If 0 ≤ *θ* < 2π, then *θ* is called the *principal argument* of *z*.



Modulus |*z*| and principle argument *θ* of a sample complex number *z*

**Figure A.5**

Using the exponential function *ez*, we rewrite (A.6.2) as follows. In analogy with the Taylor’s expansion for *ex* for real *x*, the Taylor’s expansion for *ez* for complex *z* is given by

 (A.6.3)

Substituting *z* = *i*θ into Formula (A.6.3) yields

 (A.6.4)

Setting *r* = |*z*| and substituting (A.6.4) into (A.6.2) yields

 (A.6.5)

Remark

Substituting *θ* = *π* into (A.6.5) yields Euler’s formula



a remarkable formula indeed, since it relates the fundamental mathematical constants *e*, *i*, π, and –1 together in such a wonderfully simple way.

The expression (A.6.5) for *z* yields a nice geometric interpretation (see Figure A.6) for the product of two complex numbers *z*1 = *r*1*ei*θ1, *z*2 = *r*2*ei*θ2

 (A.6.6)



Geometry of the multiplication of complex numbers

**Figure A.6**

From the generalization of (A.6.6) to the product of *n* complex numbers, we have the following multiplication rule for obtaining the modulus and argument of the product of *n* complex numbers.

Multiplication Rule for *n* Complex Numbers

*Multiply the moduli and add the arguments.*

An important special case of the multiplication rule is a formula for the *n*th power of a complex number *z* = *rei*θ, known as DeMoivre’s formula:

 (A.6.7)

Given the complex number *z* ≠ 0, (A.6.7) yields a formula for the *n* different complex numbers whose *n*th powers equal *z* (that is, the *n*th roots of *z*):

 (A.6.8)

It is easily verified that the *n*th roots of *z* defined by (A.6.8) are all distinct. When *z* = 1, (A.6.8) yields the set of *n*th *roots of unity*. Note that these points all lie on the unit circle and are equally spaced, starting at 1 (see Figure A.7). Moreover, they are all powers of the single *n*th root *e*2π/*n*, which is a primitive *n*th root of unity. More generally, a primitive *n*th root of unity ω is a complex number such that *ωn* = 1, but *ωk* ≠ 1 for *k* = 1, . . . . , *n* – 1.



Eight roots of unity

**Figure A.7**

The following useful propositions are easily verified.

proposition A.6.1

Let *ω* be a primitive *n*th root of unity, and let *k* be an integer such that 0 < *k* < *n*. Then

 ⬜

proposition A.6.2

If *ω* is a primitive *n*th root of unity, then *ω*2 is a primitive (*n*/2)th root of unity. ⬜

proposition A.6.3

If *ω* is a primitive *n*th root of unity and *k* is a positive integer, then *ωk* = 1 if, and only if, *n* divides *k*. ⬜

A.7 Mathematical Induction

Mathematical induction is the most commonly used technique for establishing the correctness of an algorithm. Correctness proofs often proceed by establishing loop invariants with the aid of mathematical induction. Mathematical induction is also useful in both the analysis of algorithms and as an algorithm design tool.

Often we would like to establish the validity of a proposition or formula concerning the set of positive integers such as

 is true for all positive integers *n*.

This formula can be easily verified for any particular (small) *n* by directly computing and comparing both sides of the equation. For example, if *n* = 5, we get 12 + 22 + 32 + 42 + 52 = 1 + 4 + 9 + 16 + 25 = 55 = 5(6)(11)/6. The question is: How do we verify that the formula is true for *all* positive integers *n*? More generally, suppose we have a sequence of propositions *P*(1), *P*(2), . . . , *P*(*n*), . . . indexed by the positive integers and we wish to establish the truth of each one of these propositions. For example, the preceding formula is the sequence of propositions



Another example might be



Given such a sequence, we clearly can’t prove each proposition one at a time; that would take forever. Thus we use the Principle of Mathematical Induction, a very powerful method by which the truth of such a sequence of propositions *P*(1), *P*(2), . . . , *P*(*n*), . . . can be established.

A.7.1 Principle of Mathematical Induction

We first state the principle of mathematical induction in its usual form. We then state the principle in various alternative (basically equivalent) forms, which are often used in algorithm analysis.

Theorem A.7.1 Principle of Mathematical Induction

Suppose we have a sequence of propositions *P*(1), *P*(2), . . . , *P*(*n*), . . . for which the following two steps have been established:

**Basis step:** *P*(1) is true

**Induction** (**or Implication**) **step:** *if* *P*(*k*) is true for any given *k*,

*then* *P*(*k* + 1) must also be true.

Then *P*(*n*) is true for all positive integers *n*.

The validity of the Principle of Mathematical Induction can be seen as follows. Since *P*(1) is true, the induction step shows that *P*(2) is true. But the truth of *P*(2) in turn implies that *P*(3) is true, and so forth. The induction step allows this process to continue indefinitely. The truth of *P*(*n*) for all *n* rests ultimately on the following property of the positive integers: Every nonempty subset of the positive integers has a smallest element. The assumption that *P*(*k*) is false for some *k* then leads to a contradiction. Indeed, if the set {*k*: *P*(*k*) is false} is nonempty, then it has a smallest element *L*. But *l* > 1, so that *P*(*l* – 1) is true. Letting *n* = *l* – 1, our induction step implies that *P*(*n* + 1) = *P*(*l*) is true, a contradiction.

As our first illustration of the use of mathematical induction, we now establish that the formula 12 + 22 + . . . + *n*2 = *n*(*n* + 1)(2*n* + 1)/6 is true for all *n*. We proceed as follows.

**Basis step:** 

**Induction step:** Assume that *P*(*k*) is true for a given *k*, so that 12 + 22 + . . . + *k*2 = *k*(*k* + 1)(2*k* + 1)/6. We must show that it would follow that *P*(*k* + 1) is true, namely, that 12 + 22 + . . . + (*k* + 1)2 = (*k* + 1)(*k* + 2)(2*k* + 3)/6. We have



and therefore *P*(*k* + 1) is true.

Thus, we have proved the induction step, and by the Principle of Mathematical Induction, the proposition 12 + 22 + . . . + *n*2 = *n*(*n* + 1)(2*n* + 1)/6 is true for all positive integers *n*.

A.7.2 Variations of the Principle of Mathematical Induction

The following three variations of the principle of mathematical induction are frequently encountered in the analysis of algorithms. In practice, a combination of these variants might be used.

1. Often the sequence of propositions starts with an index different from 1, such as 0. Then the basis step starts with this initial index. The induction step remains the same, and the two steps together establish the truth of the propositions *P*(*n*) for all *n* greater than or equal to this initial index.

2. Sometimes the propositions are only finite in number, *P*(1), . . . , *P*(*l*). Then the induction step is modified to require that *k* < 1. Of course, the conclusion then drawn is that *P*(1), . . . , *P*(*l*) are all true if the basis and induction steps are valid.

3. The Principle of Mathematical Induction can also be stated in the following so-called *strong form*, where the induction step is as follows:

**Induction step (strong form):** For any positive integer *k*, *if P*(*j*) is true for *all* positive integers *j* ≤ *k*, *then P*(*k* + 1) must also be true.

is quadratic.

The strong form of induction is very useful in establishing the correctness of recursive algorithms, since for an input of size n the recursive calls often involve smaller input sizes than *n* – 1. For example, to prove that an inorder traversal of a binary search tree on *n* nodes visits the keys in increasing order, which is the basis of the algorithm *TreeSort* given in Chapter 4, the strong form of induction is needed since the left and right subtrees can have any number of nodes between 0 and *n* – 1.

**A.8 Walks in Graphs and Digraphs**

A *walk* *W* of length *p* in a graph *G* from *u* to *v* or *u-v* walk is an alternating sequence of vertices and edges *v*0, *e*1,*v*1,*e*2,,*v*2,..., *e*p,*v*p, where *v*0 = *u* and *v*p = *v*, such that each *e*i  = {*v*i,*v*i+1} is an edge of *G*, *i* = 0, 1, ..., *p*. Vertices *v*0 and *v*p are the initial and terminal vertices of *W*, respectively, and vertices *v*1, ..., *v*p are the internal vertices of *W*. A *u-v* walk in a directed graph *D* is defined analogously except that *e*i  = (*v*i,*v*i+1) is a directed edge.

The number *wij*(*k*) of walks from *v*i to *v*j of length *k* can be computed using the adjacency matrix as follows. We will assume that the node set of the digraph *D* is *V* = {0,1, …, *n* – 1}.

**Theorem A.8.1.** The number *wij*(*k*) of walks from *i* to *j* of length *k* in a digraph *D* equals the *ij*th entry of the *k*th power of the adjacency matrix *A* of *D*, that is

*wij*(*k*) = *Ak*(*i*,*j*), *i*, *j* = 0, …, *n* – 1.

**Proof** We prove Theorem A.8.1 using mathematical induction on *k*. The Theorem is true for *k =* 0since the only walks of length 0 are the trivial walks with no internal vertices whose initial and terminal vertices are the same, establishing the basis step. Now assume the Theorem is true for walks of length *k* – 1, that is, *wij*(*k* – 1) = *Ak*-1(*i*,*j*). It follows from the definition of a walk that





Thus, by induction hypothesis we have

.

This completes the induction step and the proof of Theorem A.4. █

Theorem A.8.1 can be generalized as follows to the case where we associate a real weight *pij* with each edge (*i,j*).

**Theorem A.8.2**

Let *D* be a digraph and let *p* be a weighting of the edges over the real numbers.Suppose *wij*(*k*) is the sum over all walks *W* of length *k* of the product of the *p*-weights on the edges of *W*. Then, *wij*(*k*)equals the *ij*th entry of the *k*th power of the *p*-weighted adjacency matrix *Ap* of *D*.

An important special case of Theorem A.8.2 is when the vertices of *D* represent states of a Markov chain and *pij* represents the transition probability from state *i* to state *j*. The probability *pij* depends only on being at state *si* and moving to state *sj* and not on the previous transition history. Further, there is an edge from *i* to *j* whenever *pij* > 0. To represent all possible Markov chains we allow the digraph to have a loop at node *i*, in which case *pii* represents the probability that the state remains the same in the next transition. Letting *pij*(*k*) denote the probability of moving from state *i* to state *j* after *k* transitions, it is easily verified that *pij*(*k*) = *wij*(*k*). Thus, by Theorem A.5, *pij*(*k*) can be computed as the *ij*th entry of the *kth* power of the matrix *Ap*.

A Markov chain can be interpreted as a random walk in the digraph *D*, or an idle surfer in the case when *D* is the web digraph *W* (see Chapter 17), where *pij* is the probability that a walker at state *i* will move to state *j* in the next step. An important special case is when the transition probabilities for any node *i* are all same, so that



where *d*out(*i*) denotes the out-degree of vertex *i*.

We say that the Markov chain is *irreducible* if *D* is strongly connected and that it is *aperiodic* if for any state *i* the greatest common divisor over all the numbers *k* such that *pij*(*k*) > 0 is 1; that is, gcd {*k* | *pij*(*k*) > 0} = 1. It is easily verified that the latter condition is equivalent to the condition that, for any *i*, there is a closed walk of length *k* containing *i* for all but a finite number of integers *k*.

**A.9 Eigenvalues and Eigenvectors**

Let *M* be an *n*-by-*n* matrix over the real numbers. An *eigenvalue* of a *M* is a real number *λ* such that

*MX* =*λX* (A.9.1)

for some nonzero column vector *X***.** The vector*X*iscalled an *eigenvector* associated with eigenvalue *λ*. There may be multiple linearly independent eigenvectors associated with the same eigenvalue. For example, if *M* is the identity matrix then every vector is an eigenvector for eigenvalue 1. Subtracting *λX* from both sides of Formula (A.9.1) , we obtain

(*M – λI*) *X =* 0*.*

If follows that the matrix *M – λI* is singular, so that its determinant is zero; that is

det(*M – λI*) = 0. (A.9.2)

All the eigenvalues of *M* are determined by Formula (A.9.2). Note that the eigenvalues of *M* are the same as its transpose *MT*. If *M* is the matrix *Ap* associated with a Markov chain then it can be shown that all the eigenvalues are less than or equal to 1. Further, since *ApJ* = *J*, where *J* is the column vector of all 1’s, 1 in an eigenvalue of *Ap*, so that 1 is the largest eigenvalue of *Ap*. It follows that 1 is the largest eigenvalue of *ApT*, and there is an eigenvector vector *X* (written as a column vector) such that

*ApTX = X.*

If the Markov chain associated with *Ap* is irreducible and aperiodic,then it can be shown that *X* is unique. We refer to this unique vector as the *principle eigenvector* or *stationary distribution* of *Ap.* Further *X* can be approximated by iterating the formula

*Xi* = *ApXi*-1, *i* = 1, 2, … ,

where the initial column vector *X*0 can be taken to be any vector whose entries sum to 1. If the Markov chain associated with the matrix *Ap* is irreducible and aperiodic then, for any column vector *X*0 of positive real numbers whose entries sum to 1, the limit of *Xi* as *i* goes to infinity is the principle eigenvector or stationary distribution *X* of *Ap.*