Chapter 5
Induction and Recursion

Chapter Summary

- ✓Mathematical Induction
- ✓Strong Induction
- ✓Well-Ordering
- ✓Recursive Definitions
- ✓Structural Induction
- ✓Recursive Algorithms
- ✓Program Correctness

5.1
Mathematical Induction

Introduction

Mathematical induction is used to prove propositions of the form $\forall n P(n)$, where the universe of discourse is the set of positive integers.

We will discuss:

- **How mathematical induction can be used?**
- **Ways to remember how mathematical induction works**
- **Why is mathematical induction valid?**
- **The good and bad of mathematical induction**

Climbing an Infinite Ladder

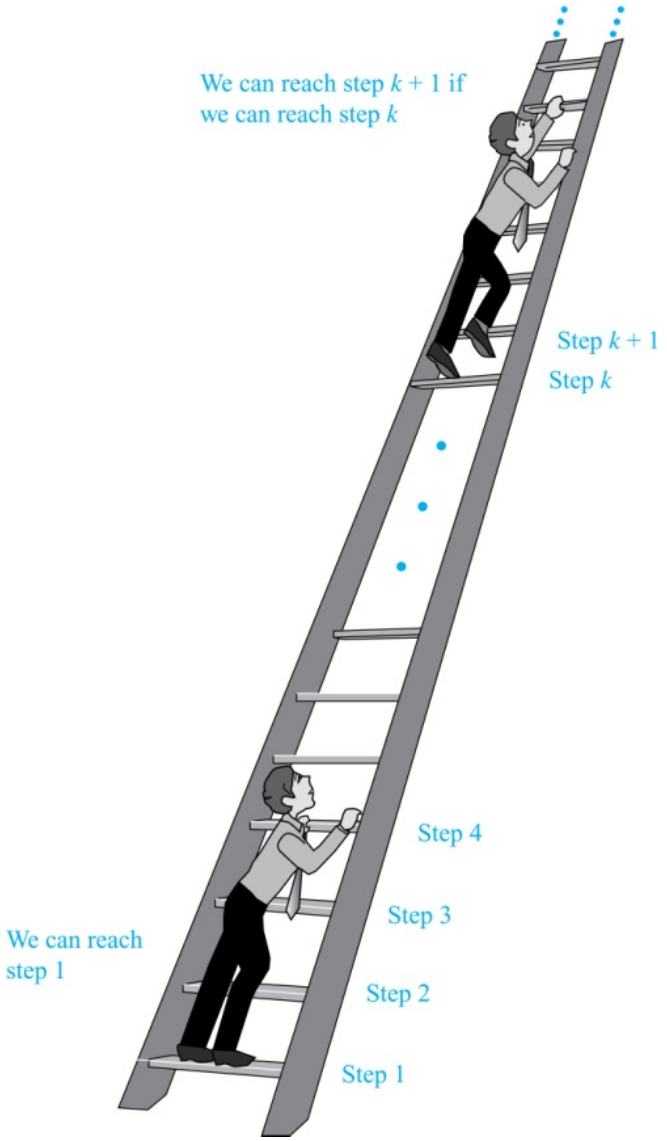
- An example motivates proof by mathematical induction

Suppose we have an infinite ladder:

1. **We can reach the first rung of the ladder.**
2. **If we can reach a particular rung of the ladder, then we can reach the next rung.**

Can we reach every rung?

- **From (1), we can reach the first rung.**
- **Then by applying (2), we can reach the second rung.**
- **Applying (2) again, the third rung.**
- **And so on.**
- **We can apply (2) any number of times to reach any particular rung, no matter how high up.**



Principle of Mathematical Induction

The (first) principle of Mathematical Induction

$$(P(1) \wedge \forall k(P(k) \rightarrow P(k+1))) \rightarrow \forall n\, P(n)$$

where the domain is the set of positive integers.

The principle has the following form.

$$P(1)$$

$$\underline{P(k) \rightarrow P(k+1)}$$

$$\therefore \forall n P(n)$$

The procedure for mathematical induction:

(1) Inductive base： Establish $P(1)$

(2) Inductive step： Prove that $P(k) \rightarrow P(k+1)$ for $k \geq 1$

Conclusion : The inductive base and the inductive step together imply $P(n)\ \forall\ n \geq 1$.

Climbing an Infinite Ladder Example:

•BASIS STEP: By (1), we can reach rung 1.

•INDUCTIVE STEP: Assume the inductive hypothesis that we can reach rung k. Then by (2), we can reach rung $k + 1$.

Hence, $P(k) \rightarrow P(k + 1)$ is true for all positive integers k. We can reach every rung on the ladder.

More general form

$$\forall n[n \geq \kappa \rightarrow P(n)]$$

The procedure :

(1) Inductive base: Establish $P(k)$

(2) Inductive step: Prove that $P(n) \rightarrow P(n+1)$ for $n \geq k$

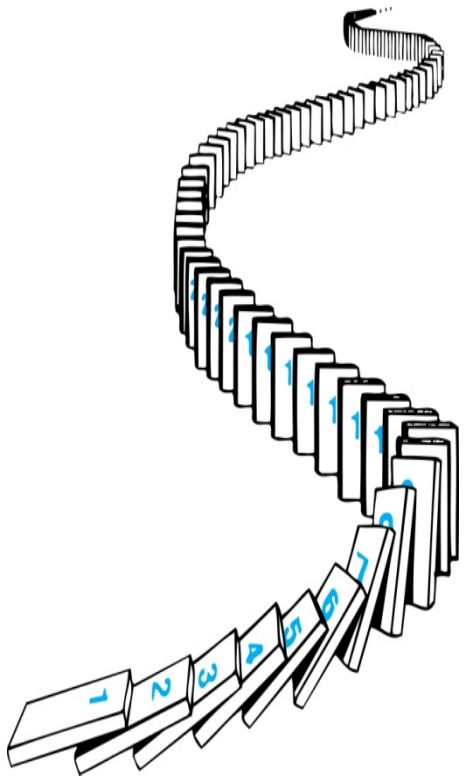
Conclusion: The inductive base and the inductive step together imply $P(n)$ $\forall$ $n \geq k$.

How Mathematical Induction Works?

- ***If the nth domino falls over the (n+1)st must fall over so pushing the first one down means all must fall down.***

Consider an infinite sequence of dominoes, labeled 1,2,3, …, where each domino is standing.

Let $P(n)$ be the proposition that the nth domino is knocked over.



We know that the first domino is knocked down, i.e., $P(1)$ is true .

We also know that if whenever the kth domino is knocked over, it knocks over the $(k + 1)$st domino, i.e, $P(k) \rightarrow P(k + 1)$ is true for all positive integers k.

Hence, all dominos are knocked over.

$P(n)$ is true for all positive integers n.

Why Is Mathematical Induction Valid?

The validity of mathematical induction follows from the well-ordering property .

A set S is *well ordered* if every subset has a least element.

For example,

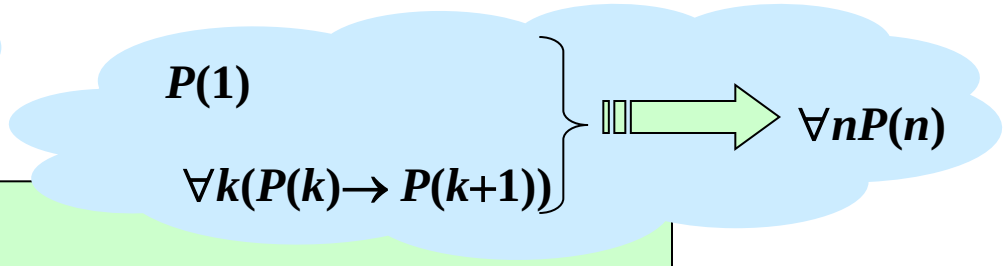
1) **N is well ordered (under the ≤ relation)**
2) **Z is not well ordered under the ≤ relation (Z has no smallest element).**
3) **[0, 1] is not well ordered since (0,1) does not have a least element.**

The well-ordering property

Every nonempty set of nonnegative integers has a least element.

The validity of Mathematical Induction

Why mathematical induction is valid?



Proof:

Assume that there is at least one positive integer for which $P(n)$ is false.

S: the set of positive integer for which $P(n)$ is false.

Then S is nonempty.

By the well-ordering property, S has a least element, which will be denoted by m.

Then $m \neq 1$, m -1 is a positive integer. m-1 is not in S. So $P(m - 1)$ is true.

Since the implication $P(k) \rightarrow P(k+1)$ is also true, $P(m)$ must be true.

The Good and Bad of Mathematical Induction

◆**The good**

- can be used to prove a conjecture once it is has been made and is true.

◆**The bad**

- Proofs do not provide insights as to why theorems are true
- Cannot be used to find new theorems

You can prove a theorem by M.I. even if you do not have the slightest idea why it is true!

Examples of Proofs by Mathematical Induction

- **Proving summation formular**
  - show that $1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$ for all nonnegative integers n.
- **Proving a variety of inequalities that hold for all positive integers greater than a particular positive integer**
  - show that $n<2^n$ for all positive integers n.
  - show that $2^n<n!$ for every integers n with $n\geq 4$.
- **Proving results about sets**
  - The Number of Subsets of a Finite Set
  - The generalization of one of De Morgan's laws: $\overline{\cap_{j=1}^{n} A_j}=\cup_{j=1}^{n} \overline{A_j}$
- **Proving results about algorithms**

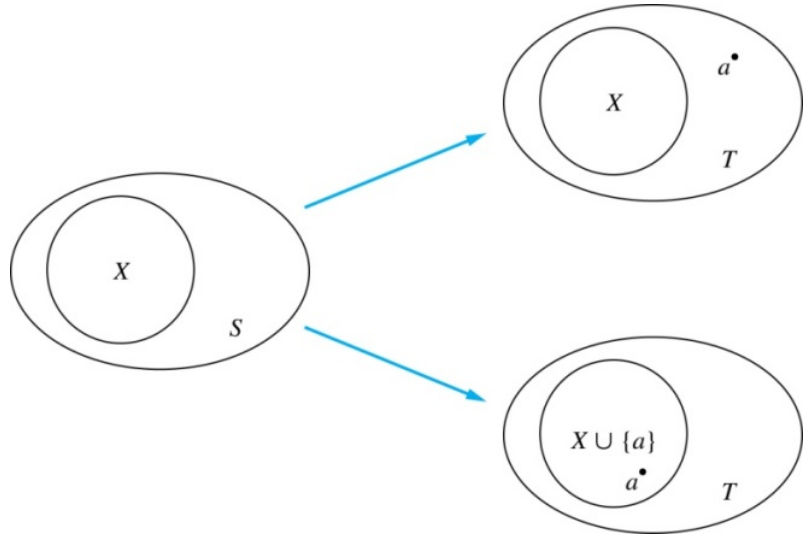
  For example, M.I. can be used to prove that some greedy algorithms always yields an optimal solution.
- **Creative uses of mathematical induction**

Number of Subsets of a Finite Set

〖Example 1〗 Use mathematical induction to show that if S is a finite set with n elements, where n is a nonnegative integer, then S has 2^n subsets.

Solution: Let $P(n)$ be the proposition that a set with n elements has 2^n subsets.

- Basis Step: $P(0)$ is true, because the empty set has only itself as a subset and $2^0 = 1$.
- Inductive Step: Assume $P(k)$ is true for an arbitrary nonnegative integer k.
  - Let T be a set with $k + 1$ elements. Then $T = S \cup \{a\}$, where $a \in T$ and $S = T - \{a\}$. Hence $|T| = k+1$.
  - For each subset X of S, there are exactly two subsets of T, i.e., X and $X \cup \{a\}$.
  - By the inductive hypothesis S has 2^k subsets. Since there are two subsets of T for each subset of S, the number of subsets of T is $2 \cdot 2^k = 2^{k+1}$.



Creative Uses of Mathematical Induction

〖**Example 2**〗 **Odd Pie Fights** An odd number of people stand in a yard at mutually distinct distances. At the same time each person throws a pie at their nearest neighbor, hitting this person. Use mathematical induction to show that there is at least one survivor, that is, at least one person who is not hit by a pie.

Solution: Let $P(n)$ be the statement that there is a survivor whenever $2n + 1$ people stand in a yard at distinct mutual distances and each person throws a pie at their nearest neighbor.

- Basis Step: When $n = 1$, there are $2n + 1 = 3$ people in the pie fight.
- Inductive Step: assume that $P(k)$ is true for an arbitrary odd integer k with $k \geq 3$. That is, assume that there is at least one survivor whenever $2k + 1$ people stand in a yard at distinct mutual distances and each throws a pie at their nearest neighbor. We must show that if the inductive hypothesis $P(k)$ is true, then $P(k + 1)$, the statement that there is at least one survivor whenever $2(k + 1) + 1 = 2k + 3$ people stand in a yard at distinct mutual distances and each throws a pie at their nearest neighbor, is also true.

Let A and B be the closest pair of people in this group of $2k + 3$ people. When each person throws a pie at the nearest person, A and B throw pies at each other. We have two cases to consider:

(i) when someone else throws a pie at either A or B

(ii) when no one else throws a pie at either A or B

Guidelines: Mathematical Induction Proofs

Template for Proofs by Mathematical Induction

1. **Express the statement that is to be proved in the form "for all $n \geq b$, $P(n)$" for a fixed integer b.**
2. **Write out the words "Basis Step." Then show that $P(b)$ is true, taking care that the correct value of b is used. This completes the first part of the proof.**
3. **Write out the words "Inductive Step".**
4. **State, and clearly identify, the inductive hypothesis, in the form "assume that $P(k)$ is true for an arbitrary fixed integer $k \geq b$."**
5. **State what needs to be proved under the assumption that the inductive hypothesis is true. That is, write out what $P(k + 1)$ says.**
6. **Prove the statement $P(k + 1)$ making use the assumption $P(k)$. Be sure that your proof is valid for all integers k with $k \geq b$, taking care that the proof works for small values of k, including $k = b$.**
7. **Clearly identify the conclusion of the inductive step, such as by saying "this completes the inductive step."**
8. **After completing the basis step and the inductive step, state the conclusion, namely, by mathematical induction, $P(n)$ is true for all integers n with $n \geq b$.**

5.2

Strong Induction and Well-ordering

Strong Induction

The Second Principle of Mathematical Induction
(Strong Induction, complete induction)

$$(P(n_0) \wedge \forall k\ (\ k>=n_0 \wedge P(n_0) \wedge P(n_0+1) \wedge \ldots \wedge P(k) \rightarrow P(k+1))) \rightarrow \forall n\ P(n)$$

The procedure :

(1) Inductive base: Establish $P(n_0)$

(3) Inductive step: Prove $P(n_0) \wedge P(n_0+1) \wedge \ldots \wedge P(k) \rightarrow P(k+1)$

Conclusion: The inductive base and the inductive step allow one to conclude that $P(n)$ $\forall n \geq n_0$

Examples of Proof Using Strong Induction

〖Example 1〗 Show that if n is an integer greater than 1, then n can be written uniquely as a prime or the product of two or more primes.

Solution:

Let $P(n)$ be n can be written uniquely as a prime or the product of two or more primes

(1) Inductive base

$P(2)$ is true.

(2) Inductive step

Assume that $P(k)$ is true for all positive integers k with $k \leq n$. Show that $P(k+1)$ is true under the assumption.

Proof by cases.

- **$k+1$ is prime**

  $P(k+1)$ is true.

- **$k+1$ is composite**

  $k+1=a\times b,\ 2\leq a, b< n+1$

Note:

1. The validity of both mathematical induction and strong induction follow from the well-ordering property.
2. In fact, mathematical induction, strong induction, and well-ordering are all equivalent principles.

Questions:

How to establish?

Another Strong Induction Proof Example

Some terms:

- **polygon**
- **side, vertex**
- **a polygon is simple**
- **Every simple polygon divides the plane into two regions: its interior, its exterior.**
- **convex, nonconvex**
- **diagonal, interior diagonal**
- **triangulation**

【LEMMA 1】Every simple polygon with at least four sides has an interior diagonal.

【Theorem 1】A simple polygon with n sides, where n is an integer with $n \geq 3$, can be triangulated into n-2 triangles.

Proof:

Let $T(n)$ be the statement that simple polygon with n sides can be triangulated into n-2 triangles

(1) Inductive base

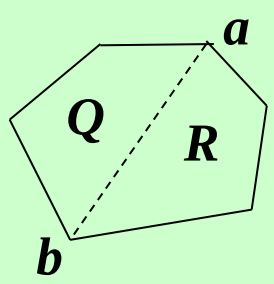
$T(3)$ is true.

(2) Inductive step

Assume that $T(j)$ is true for all integers j with $3 \leq j \leq k$. We must show $T(k+1)$ is true, that is that every simple polygon with $k+1$ sides can be triangulated into k-1 triangles.

Suppose that we have a simple polygon P with $k+1$ sides.

By Lemma 1, P has an interior diagonal ab. ab splits P into two simple polygon Q, with s $(3 \leq s \leq k)$ sides, and R, with t $(3 \leq t \leq k)$ sides.



Proofs by the Well-ordering property

〖Example 2〗 Use the well-ordering property to prove the division algorithm.

Recall that the division algorithm states that if a be an integer and d a positive integer,Then there are unique integers q and r, with $0 \le r < d$ such that $a = dq + r$.

Solution:

Let S be the set of nonnegative integers of the form $a-dq$, where q is an integer. This set in nonempty.

By the well-ordering property, S has a least element $r=a-dq_0$.

The integer r is nonnegative. It is also the case that $r<d$. If it were not , then there would be a smaller nonnegative element in S, namely, $r=a-d(q_0+1)$.

5.3

Recursive Definition and Structural Induction

Why Recursive definition?

Sometimes it is difficult to express the members of an object or numerical sequence explicitly.

For example,

The Fibonacci sequence:

$\{f_n\} = 0,1,1,2,3,5,8,13,21,34,55,\ldots$

Recursion **is a principle closely related to mathematical induction.**

In a *recursive definition*, an object is defined in terms of itself.

We can recursively define *sequences*, *functions* and *sets*.

Recursively defined functions

***Recursively defined functions*, with the set of nonnegative integers as its domain:**

- **Basis Step: Specify the value of the function at zero.**
- **Recursive Step: Give the rules for finding its value at an integer from its value at smaller integers.**

〖Example 1〗Give a recursive definition of factorial function.

Solution:

$$f(n) = n!=1 \times 2 \times \ldots \times (n-1) \times n$$

$$f(0) = 1$$

$$f(n) = nf(n-1)$$

Are recursively defined functions well-defined?

Recursively defined functions are well-defined.

Let $P(n)$ be the statement "f is well-defined at n .

(1) $P(0)$ is true.

(2) Assume that $P(n-1)$ is true. Then f is well-defined at n.

Since $f(n)$ is given in terms of some $f(n-1)$.

The Recursive definition of the Fibonacci numbers and relative proofs

〖Example 2〗Give a recursive definition of the Fibonacci numbers

$\{f_n\} = 0,1,1,2,3,5,8,13,21,34,55,\ldots$

Note:

The Fibonacci number f_n can be thought either as the nth term of the sequence of Fibonacci numbers $f_0, f_1, \ldots$or as the value at the integer n of a function $f(n)$.

Solution:

The Fibonacci numbers can be defined

$$f_0 = 0$$

$$f_1 = 1$$

$$f_n = f_{n-1} + f_{n-2} \text{ for } n=2,3,4,\ldots$$

〖Example 3〗 Show that

$$f_n > \alpha^{n-2} \quad \textit{where} \quad \alpha = \frac{1+\sqrt{5}}{2} \quad \textit{whenever} \quad n \geq 3.$$

Proof:

(1) Inductive base

$$f_3 = 2 > \alpha = \frac{1+\sqrt{5}}{2} \qquad f_4 = 3 > \alpha^2 = (\frac{1+\sqrt{5}}{2})^2$$

(2) Inductive step

Assume that *P*(*k*) is true, namely, that $f_k > \alpha^{k-2}$ **for all integers *k* with** $3 \leq k \leq n$ **.**

We must show that *P*(*k*+1) is true, that is $f_{k+1} > \alpha^{k-1}$ **.**

Since α **is the solution of** $x^2 - x - 1 = 0$ **, it follows that** $\alpha^2 = \alpha + 1$

Therefore,

$$\alpha^{k-1} = \alpha^2 \cdot \alpha^{k-3} = (\alpha+1) \cdot \alpha^{k-3} = \alpha^{k-2} + \alpha^{k-3}$$

By the inductive hypothesis, if $n \geq 5$ **, it follows that**

$$f_{k-1} > \alpha^{k-3}, f_k > \alpha^{k-2}$$

Therefore, $f_{k+1} = f_{k-1} + f_k > \alpha^{k-3} + \alpha^{k-2} = \alpha^{k-1}$

The Complexity of Euclidean algorithm

Euclidean algorithm **can used to find the greatest common divisor of the positive integers a and b, where a≥b.**

Let $a=bq+r$, where a, b, q, and r are integers. Then gcd (a, b)=gcd (r, b)

$a = r_0, b = r_1$

$r_0 = r_1 q_1 + r_2 \qquad 0 \le r_2 < r_1$

$r_1 = r_2 q_2 + r_3 \qquad 0 \le r_3 < r_2$

…

$r_{n-2} = r_{n-1} q_{n-1} + r_n \qquad 0 \le r_n < r_{n-1}$

$r_{n-1} = r_n q_n$

gcd (a, b)=gcd $(r_{n-1}, r_n)= r_n$

The complexity of Euclidean algorithm?

- Basic operation: division
- n dividions have been used to find r_n

***n*=?**

For example, find gcd (662,414)

$662=414\times1+248$

$414=248\times1+166$

$248=166\times1+82$

$166=82\times2+2$

$82=2\times41$

hence, gcd (662,414)=2

$n=5 \leq 5k$

LAME'S Theorem

LAME'S Theorem **Let *a* , *b* be positive integers with *a*≥*b*. Then the number of divisions used by the Euclidean algorithm to find gcd (*a*, *b*) is less than or equal to five times the number of decimal digits in *b*.**

Proof:

$n \le 5k$

From the above equations,

$q_1, q_2, \ldots, q_{n-1} \ge 1$ $\quad q_n \ge 2 \quad \because r_n < r_{n-1}$

This implies

$$r_n \ge 1 = f_2$$

$$r_{n-1} \ge 2r_n \ge 2f_2 = f_3$$

$$r_{n-2} \ge r_{n-1} + r_n \ge f_3 + f_2 = f_4$$

$$r_{n-3} \ge r_{n-2} + r_{n-1} \ge f_3 + f_4 = f_5$$

$$r_0 = r_1 q_1 + r_2 \qquad 0 \le r_2 < r_1$$

$$r_1 = r_2 q_2 + r_3 \qquad 0 \le r_3 < r_2$$

$$\ldots$$

$$r_{n-2} = r_{n-1} q_{n-1} + r_n \qquad 0 \le r_n < r_{n-1}$$

$$r_{n-1} = r_n q_n$$

…

$$r_2 \geq r_3 + r_4 \geq f_{n-1} + f_{n-2} = f_n$$

$$b = r_1 \geq r_2 + r_3 \geq f_n + f_{n-1} = f_{n+1}$$

Q $f_{n+1} > \alpha^{n-1}$ for $n > 2$

$$\therefore b > \alpha^{n-1}$$

$$\log_{10} b > (n-1)\log_{10} \alpha > \frac{n-1}{5}$$

$$n - 1 < 5 \cdot \log_{10} b < 5k$$

$$\therefore n \leq 5k$$

$$r_0 = r_1\, q_1 + r_2 \qquad 0 \leq r_2 < r_1$$

$$r_1 = r_2\, q_2 + r_3 \qquad 0 \leq r_3 < r_2$$

…

$$r_{n-2} = r_{n-1}\, q_{n-1} + r_n \qquad 0 \leq r_n < r_{n-1}$$

$$r_{n-1} = r_n\, q_n$$

Recursively defined sets

- **Sets can be defined recursively.**
  - **Basis Step: Specify an initial collection of elements.**
  - **Recursive Step: Give the rules for constructing elements of the set from other elements already in the set.**

- Sometimes the recursive definition has an ***exclusion rule***, which specifies that the set contains nothing other than those elements specified in the basis step and generated by applications of the rules in the recursive step.
- We will always assume that the exclusion rule holds, even if it is not explicitly mentioned.

Recursively defined sets

〖Example 4〗 Subset of Integers S:

Basis Step: $3 \in S$

Recursive Step: if $x \in S$ and $y \in S$, then $x+y \in S$

Initially 3 is in *S*, then 3 + 3 = 6, then 3 + 6 = 9, etc.

〖Example 5〗 The natural numbers N.

Basis Step : $0 \in$ N.

Recursive Step : If *n* is in N, then *n* + 1 is in N.

Initially 0 is in *S*, then 0 + 1 = 1, then 1 + 1 = 2, etc.

Strings

Definition: The set Σ^* of *strings* over the alphabet Σ:

BASIS STEP: $\lambda \in \Sigma^*$ (λ is the empty string)

RECURSIVE STEP: If w is in Σ^* and x is in Σ, then $wx \in \Sigma^*$.

Example 6: If $\Sigma = \{0,1\}$, the strings in in Σ^* are the set of all bit strings, λ,0,1, 00,01,10, 11, etc.

Example 7: If $\Sigma = \{a,b\}$, show that *aab* is in Σ^*.

Since $\lambda \in \Sigma^*$ and $a \in \Sigma$, $a \in \Sigma^*$.

Since $a \in \Sigma^*$ and $a \in \Sigma$, $aa \in \Sigma^*$.

Since $aa \in \Sigma^*$ and $b \in \Sigma$, $aab \in \Sigma^*$.

String Concatenation

Definition: Two strings can be combined via the operation of *concatenation*. Let Σ be a set of symbols and Σ^* be the set of strings formed from the symbols in Σ. We can define the concatenation of two strings, denoted by $\cdot$, recursively as follows.

BASIS STEP: If $w \in \Sigma^*$, then $w \cdot \lambda = w$.

RECURSIVE STEP: If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$.

Often $w_1 \cdot w_2$ is written as $w_1 w_2$.

If $w_1 =$ *abra* and $w_2 =$ *cadabra*, the concatenation $w_1 w_2 =$ *abracadabra*.

◆ **Another important use of Recursive definitions to define well-formed formulae of various type.**

〖Example 8〗 Well-formed formulae for compound proposition.

Definition: The set of *well-formed formulae* in propositional logic involving **T**, **F**, propositional variables, and operators from the set $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.

Solution：：

Basis Step: *T*, *F*, and *p*, where *p* is a propositional variable, are well-formed formulae.

Recursive Step: $(\neg p)$, $(p \vee q)$, $(p \wedge q)$, $(p \rightarrow q)$, $(p \leftrightarrow q)$ are well-formed formulae if *p* and *q* are well-formed formulae.

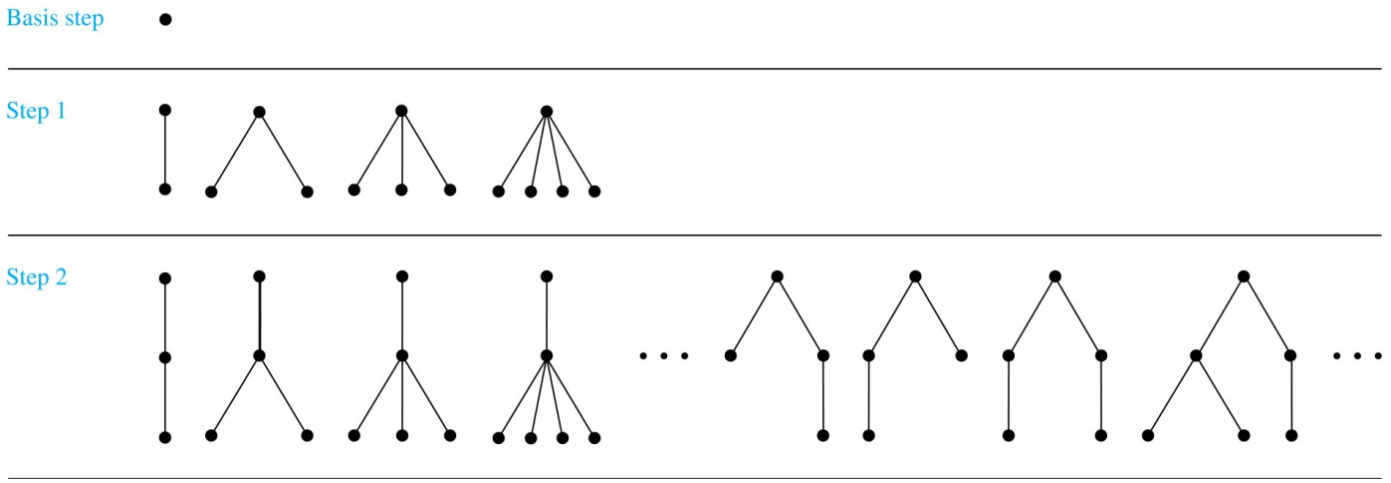
Examples: $((p \vee q) \rightarrow (q \wedge \mathrm{F}))$ is a well-formed formula.

$pq \wedge$ is not a well formed formula.

- **The set of rooted trees, extended binary trees and full binary trees can be defined recursively.**

For example, building up rooted trees.

BASIS STEP: A single vertex r is a rooted tree.
RECURSIVE STEP: Suppose that $T_1, T_2, \ldots, T_n$ are disjoint rooted trees with roots $r_1, r_2, \ldots, r_n$, respectively. Then the graph formed by starting with a root r, which is not in any of the rooted trees $T_1, T_2, \ldots, T_n$, and adding an edge from r to each of the vertices $r_1, r_2, \ldots, r_n$, is also a rooted tree.

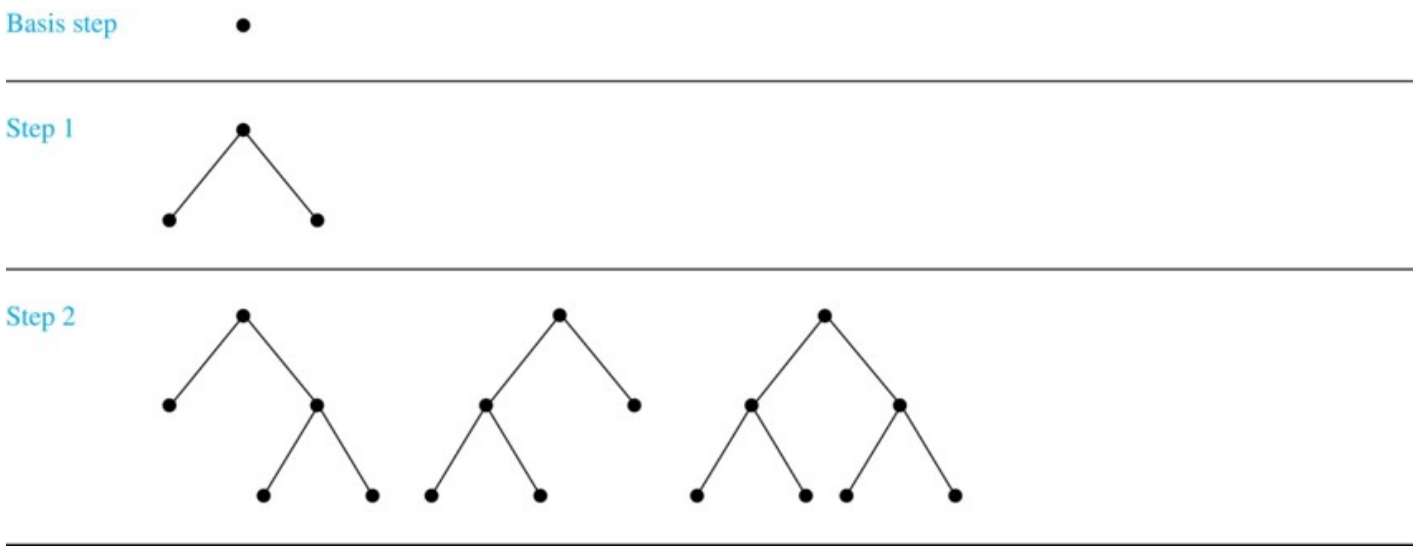


◆ Full binary trees

Definition: The set of *full binary trees* can be defined recursively by these steps.

BASIS STEP: There is a full binary tree consisting of only a single vertex r.

RECURSIVE STEP: If T_1 and T_2 are disjoint full binary trees, there is a full binary tree, denoted by $T_1 \cdot T_2$, consisting of a root r together with edges connecting the root to each of the roots of the left subtree T_1 and the right subtree T_2.



Induction and Recursively Defined Sets

Example 9: Show that the set S defined by specifying that $3 \in S$ and that if $x \in S$ and $y \in S$, then $x + y$ is in S, is the set of all positive integers that are multiples of 3.

Solution: Let A be the set of all positive integers divisible by 3. To prove that $A = S$, show that A is a subset of S and S is a subset of A.

$A \subseteq S$: Let $P(n)$ be the statement that $3n$ belongs to S.
BASIS STEP: $3 \cdot 1 = 3 \in S$, by the first part of recursive definition.
INDUCTIVE STEP: Assume $P(k)$ is true. By the second part of the recursive definition, if $3k \in S$, then since $3 \in S$, $3k + 3 = 3(k + 1) \in S$. Hence, $P(k + 1)$ is true.

$S \subseteq A$:
BASIS STEP: $3 \in S$ by the first part of recursive definition, and $3 = 3 \cdot 1$.
INDUCTIVE STEP: The second part of the recursive definition adds $x + y$ to S, if both x and y are in S. If x and y are both in A, then both x and y are divisible by 3. By part (i) of Theorem 1 of Section 4.1, it follows that $x + y$ is divisible by 3.

Structural Induction

To prove results about recursively defined sets, we can use

- ✓ **Mathematical induction**
- ✓ **Structural induction**

A proof by structural induction:

Basis Step: Show that the result holds for all elements specified in the basis step of the recursive definition to be in the set.

Recursive Step: Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

The validity of structural induction

The validity of structural induction follows from the principle of mathematical induction for the nonnegative integers.

$p(n)$: the result is true for all elements of the set that are generated by n or fewer applications of the rules in the recursive step of a recursive definition.

Basis Step: Show that $p(0)$ is true.

Recursive Step: if we assume $p(k)$ is true, it follows that $p(k+1)$ is true.

Examples of Proofs Using Structural Induction

〖Example 10〗Show that every well-formed formula for compound propositions, as defined in Example 5, contains an equal number of left and right parentheses.

Proof:

Basis Step: Show that the result is true for *T*, *F*, and *p*, whenever *p* is a propositional variable.

Recursive Step: Show that if the result is true for the compound propositions *p* and *q* , it is also true for ($\neg p$), ($p \vee q$), ($p \wedge q$), ($p \rightarrow q$), ($p \leftrightarrow q$).

Structural Induction and Binary Trees

Definition: The *height h(T)* of a full binary tree *T* is defined recursively as follows:

- **BASIS STEP: The height of a full binary tree *T* consisting of only a root *r* is *h(T)* = 0.**
- **RECURSIVE STEP: If T_1 and T_2 are full binary trees, then the full binary tree $T = T_1 \cdot T_2$ has height**

$$h(T) = 1 + \max\left(h(T_1), h(T_2)\right).$$

The number of vertices *n*(*T*) of a full binary tree *T* satisfies the following recursive formula:

- **BASIS STEP: The number of vertices of a full binary tree *T* consisting of only a root *r* is *n*(*T*) = 1.**
- **RECURSIVE STEP: If T_1 and T_2 are full binary trees, then the full binary tree $T = T_1 \cdot T_2$ has the number of vertices**

$$n(T) = 1 + n(T_1) + n(T_2).$$

Structural Induction and Binary Trees

Theorem: If T is a full binary tree, then $n(T) \le 2^{h(T)+1} - 1.$

Proof: Use structural induction.

- **BASIS STEP: The result holds for a full binary tree consisting only of a root,**

  $n(T) = 1$ and $h(T) = 0.$ **Hence,** $n(T) = 1 \le 2^{0+1} - 1 = 1.$

- **RECURSIVE STEP: Assume** $n(T_1) \le 2^{h(T_1)+1} - 1$ **and also** $n(T_2) \le 2^{h(T_2)+1} - 1$ **whenever T_1 and T_2 are full binary trees.**

$$
\begin{aligned}
n(T) &= 1 + n(T_1) + n(T_2) && \left(\textit{by recursive formula of } n(T)\right) \\
&\le 1 + \left(2^{h(T_1)+1} - 1\right) + \left(2^{h(T_2)+1} - 1\right) && \left(\textit{by inductive hypothesis}\right) \\
&\le 2 \cdot \max\left(2^{h(T_1)+1}, 2^{h(T_2)+1}\right) - 1 \\
&= 2 \cdot 2^{\max\left(h(T_1), h(T_2)+1\right)} - 1 && \left(\max\left(2^x, 2^y\right) = 2^{\max(x,y)}\right) \\
&= 2 \cdot 2^{h(t)} - 1 && \left(\textit{by recursive definition of } h(T)\right) \\
&= 2^{h(t)+1} - 1
\end{aligned}
$$

Generalized Induction

- *Generalized induction* is used to prove results about sets other than the integers that have the well-ordering property. (*explored in more detail in Chapter* 9)
- For example, consider an ordering on N × N, ordered pairs of nonnegative integers. Specify that (x_1, y_1) is less than or equal to (x_2, y_2) if either $x_1 < x_2$, or $x_1 = x_2$ and $y_1 < y_2$. This is called the *lexicographic ordering*.
- Strings are also commonly ordered by a *lexicographic ordering*.
- The next example uses generalized induction to prove a result about ordered pairs from N × N.

Generalized Induction

Example 11: Suppose that $a_{m,n}$ is defined for $(m,n) \in \mathbf{N} \times \mathbf{N}$ by $a_{0,0} = 0$ and

$$a_{m,n} = \begin{cases} a_{m-1,n} + 1 & \text{if } n = 0 \text{ and } m > 0 \\ a_{m,n-1} + n & \text{if } n > 0 \end{cases}.$$

Show that $a_{m,n} = m + n(n+1)/2$ is defined for all $(m,n) \in \mathbf{N} \times \mathbf{N}$.

Solution: Use generalized induction.

BASIS STEP: $a_{0,0} = 0 + (0.1)/2$

INDUCTIVE STEP: Assume that $a_{m',n'} = m' + n'(n'+1)/2$ **whenever (m',n') is less than (m,n) in the lexicographic ordering of $\mathbf{N} \times \mathbf{N}$.**

If $n = 0$, by the inductive hypothesis we can conclude

$$a_{m,n} = a_{m-1,n} + 1 = m - 1 + n(n+1)/2 + 1 = m + n(n+1)/2.$$

If n > 0, by the inductive hypothesis we can conclude

$$a_{m,n} = a_{m-1,n} + 1 = m + n(n-1)/2 + n = m + n(n+1)/2.$$

5.4
Recursive Algorithms

Section Summary

- ✓Recursive Algorithms
- ✓Proving Recursive Algorithms Correct
- ✓Recursion and Iteration

Recursive Algorithms

- An algorithm is called *recursive* if it solves a problem by reducing it to an instance of the same problem with smaller input.
- For the algorithm to terminate, the instance of the problem must eventually be reduced to some initial case for which the solution is known.

Recursive Factorial Algorithm

〖Example 1〗 Give a recursive algorithm for computing the factorial function *n*! .

$$n! = \text{factorial}\,(n) = \begin{cases} 1, & \text{if } n = 0 \\ n \cdot \text{factorial}\,(n-1), & \text{if } n > 0 \end{cases}$$

Algorithm 1 A Recursive Procedure for Factorials.

```
procedure factorial(n:nonnegative integer)
{
    if n=0 then
        factorial(n):=1
    else
       factorial(n) := n×factorial(n-1)
}
```

How does this algorithm work?

Recursive GCD Algorithm

〖Example 2〗 Give a recursive algorithm for computing the greatest common divisor of two nonnegative integers a and b with $a<b$.

Solution: Use the reduction

$$\gcd(a,b) = \gcd(b \bmod a, a)$$

and the condition $\gcd(0,b) = b$ when $b > 0$.

How does this algorithm work?

Algorithm 2 A Recursive Algorithm for Computing gcd(a,b).

procedure gcd(a,b: nonnegative integers with $a<b$)

{ if a=0 then

gcd(a,b):=b

else

gcd(a,b):= gcd(b mod a, a)}

Other Recursive Algorithms

- ✓ Recursive Exponentiation Algorithm for computing a^n
- ✓ Recursive Modular Exponentiation Algorithm for computing $b^n \bmod m$
- ✓ Recursive Binary Search Algorithm

Is this recursive algorithms correct?

Proving Recursive Algorithms Correct

- **Both mathematical and strong induction are useful techniques to show that recursive algorithms always produce the correct output.**

Example: Prove that the algorithm for computing the powers of real numbers is correct.

```
procedure power(a: nonzero real number, n: nonnegative integer)
if n = 0 then return 1
else return a· power (a, n − 1)
{output is a^n}
```

Solution: Use mathematical induction on the exponent n.

BASIS STEP: $a^0 = 1$ for every nonzero real number a, and $power(a,0) = 1$

INDUCTIVE STEP: The inductive hypothesis is that $power(a,k) = a^k$, for all $a \neq 0$. Assuming the inductive hypothesis, the algorithm correctly computes a^{k+1}, since

$$power(a,k+1) = a \cdot power(a,k) = a \cdot a^k = a^{k+1}.$$

Recursion and Iteration

- **Recursion**
  - Successively reducing the computation to the evaluation of the function an smaller integers
- **Iteration**
  - Start with the value of the function at one or more integers, the base cases, and successively apply the recursive definition to find the value of the function at successive large integers.

Recursion and Iteration

Algorithm 3 A Recursive Algorithm for Fibonacci Numbers.

```
procedure fibo(n: nonnegative integer)
if n = 0 then fibo(0) := 0
else if n = 1 then fibo(1) := 1
else fibo(n) := fibo(n – 1) + fibo(n – 2)
```

Algorithm 4 An Iterative Algorithm for Fibonacci Numbers.

```
procedure iterative_fibo(n: nonnegative integer)
if n = 0 then y := 0
else
begin
        x := 0
        y := 1
        for i := 1 to n-1
        begin
                z := x + y
                x : = y
                y := z
        end
end   {y is the n-th Fibonacci number}
```

Note:

- **For every recursive algorithm, there is an equivalent iterative algorithm.**
- **Recursive algorithms are often shorter, more elegant, and easier to understand than their iterative counterparts.**
- **However, iterative algorithms are usually more efficient in their use of space and time.**

Homework:

SE： P.331 43,61

P.344 31,41

P.358 10,12,31,43,50,59 (b)(d)

EE： P.352 43,61

P.365 31,41

P.379 10,12,33,45,52,61 (b)(d)