CHAPTER

5 Induction and Recursion

5.1 Mathematical Induction

Ch.5.1 Q.#3

3. Let $P(n)$ be the statement that $1^2 + 2^2 + \cdots + n^2 = n(n+1)(2n+1)/6$ for the positive integer n.

a) What is the statement $P(1)$?

$n=1$, $P(1)$

$1^2 = 1 \cdot 2 \cdot 3/6$

b) Show that $P(1)$ is true, completing the basis step of the proof.

$P(1) = 1 \cdot (1+1)(2(1)+1)/6$

$= 1 \cdot 2(3)/6 = 1$

c) What is the inductive hypothesis?

$$1^2 + 2^2 + \cdots + k^2 = \frac{k(k+1)(2k+1)}{6}$$

d) What do you need to prove in the inductive step?

To show for each $k \geq 1$ that $p(k) \Rightarrow p(k+1)$

$p(k+1)$

$$= 1^2 + 2^2 + \cdots + k^2 + (k+1)^2 = \frac{(k+1)(k+2)(2k+3)}{6}$$

e) Complete the inductive step, identifying where you use the inductive hypothesis.

$$(1^2 + 2^2 + \cdots + k^2) + (k+1)^2 = \frac{k(k+1)(2k+1)}{6} + (k+1)^2 \quad \text{(by the inductive hypothesis)}$$

$$= \frac{k+1}{6}(k(2k+1) + 6(k+1))$$

$$= \frac{k+1}{6}(2k^2 + k + 6k + 6)$$

$$= \frac{k+1}{6}(2k^2 + 7k + 6)$$

$$= \frac{k+1}{6}(2k+3)(k+2)$$

$$= \frac{(k+1)(k+2)(2k+3)}{6}$$

f) Explain why these steps show that this formula is true whenever n is a positive integer.

both the basis and inductive steps are completed.

By the principle of mathematical induction,

the statement is true for every positive integer n.

Mathematical Induction

In general, mathematical induction * can be used to prove statements that assert that $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function. A proof by mathematical induction has two parts, a **basis step**, where we show that $P(1)$ is true, and an **inductive step**, where we show that for all positive integers k, if $P(k)$ is true, then $P(k+1)$ is true.

PRINCIPLE OF MATHEMATICAL INDUCTION To prove that $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function, we complete two steps:

BASIS STEP: We verify that $P(1)$ is true.

INDUCTIVE STEP: We show that the conditional statement $P(k) \rightarrow P(k+1)$ is true for all positive integers k.

5.2 Strong Induction and Well-Ordering

Ch.5.2 Q# 3

3. Let $P(n)$ be the statement that a postage of n cents can be formed using just 3-cent stamps and 5-cent stamps. The parts of this exercise outline a strong induction proof that $P(n)$ is true for $n \geq 8$.
   a) Show that the statements $P(8)$, $P(9)$, and $P(10)$ are true, completing the basis step of the proof.
   b) What is the inductive hypothesis of the proof?
   c) What do you need to prove in the inductive step?
   d) Complete the inductive step for $k \geq 10$.
   e) Explain why these steps show that this statement is true whenever $n \geq 8$.

a) $P(8)$ is true.
because we can form 8 cents of postage with one 3-cent stamp and one 5-cent stamp

$P(9)$ is true.
because we can form 9 cents of postage with three 3-cent stamps.

$P(10)$ is true.
because we can form 10 cents of postage with two 5-cent stamps.

b) Inductive hypothesis
"By using 3-cent and 5-cent stamps we can form j cents postage for all j with $8 \leq j \leq k$. where we assume $k \geq 10$."

c) Inductive Step
Assuming the inductive hypothesis,
that we can form $k+1$ cents
postage using just 3-cent and 5-cent stamps.

INDUCTIVE STEP: We show that the conditional statement $[P(1) \wedge P(2) \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$ is true for all positive integers k.

Strong Induction

Before we illustrate how to use strong induction, we state this principle again.

STRONG INDUCTION To prove that $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function, we complete two steps:

BASIS STEP: We verify that the proposition $P(1)$ is true.

INDUCTIVE STEP: We show that the conditional statement $[P(1) \wedge P(2) \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$ is true for all positive integers k.

Strong induction is sometimes called the **second principle of mathematical induction** or **complete induction**. When the terminology "complete induction" is used, the principle of mathematical induction is called **incomplete induction**, a technical term that is a somewhat unfortunate choice because there is nothing incomplete about the principle of mathematical induction; after all, it is a valid proof technique.

STRONG INDUCTION AND THE INFINITE LADDER To better understand strong induction, consider the infinite ladder in Section 5.1. Strong induction tells us that we can reach all rungs if

1. we can reach the first rung, and
2. for every integer k, if we can reach all the first k rungs, then we can reach the $(k+1)$st rung.

That is, if $P(n)$ is the statement that we can reach the nth rung of the ladder, by strong induction we know that $P(n)$ is true for all positive integers n, because (1) tells us $P(1)$ is true, completing the basis step and (2) tells us that $P(1) \wedge P(2) \wedge \cdots \wedge P(k)$ implies $P(k+1)$, completing the inductive step.

d) Complete the inductive step for $k \geq 10$.

we want to form $k+1$ cents of postage.

Given: $k \geq 10$,

we know that $P(k-2)$ is true,

we can form $k-2$ cents of postage

As we put one more 3-cent stamp,

and then we can form $k-2+3$

$\Rightarrow k+1$ cents of postage.

e) Explain why these steps show that this statement is true whenever $n \geq 8$.

Both the basis step & the inductive step are completed, and by the principle of strong induction, the statement is true for every integer $n \geqslant 8$.

5.3 Recursive Definitions and Structural Induction

Ch. 5.3 Q# 7(a)

7. Give a recursive definition of the sequence $\{a_n\}$, $n = 1, 2, 3, \ldots$ if

a) $a_n = 6n$.

$a_0 = 0$ +6
$a_1 = 6$ +6
$a_2 = 12$ +6
$a_3 = 18$ +6
$a_4 = 24$
$\vdots$
$a_n = 6n$

This shows each term in this sequence is 6 greater than the preceding term.

Let $a_1 = 6$,

$\Rightarrow a_{n+1} = a_n + 6$, $\forall n \geq 1$.

Recursively Defined Functions

We use two steps to define a function with the set of nonnegative integers as its domain:

BASIS STEP: Specify the value of the function at zero.

RECURSIVE STEP: Give a rule for finding its value at an integer from its values at smaller integers.

Such a definition is called a **recursive** or **inductive definition**. Note that a function $f(n)$ from the set of nonnegative integers to the set of a real numbers is the same as a sequence $a_0, a_1, \ldots$ where a_i is a real number for every nonnegative integer i. So, defining a real-valued sequence $a_0, a_1, \ldots$ using a recurrence relation, as was done in Section 2.4, is the same as defining a function from the set of nonnegative integers to the set of real numbers.

DEFINITION 1 The set Σ^* of *strings* over the alphabet Σ is defined recursively by

BASIS STEP: $\lambda \in \Sigma^*$ (where λ is the empty string containing no symbols).

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

DEFINITION 2 Two strings can be combined via the operation of *concatenation*. Let Σ be a set of symbols and Σ^* the set of strings formed from symbols in Σ. We can define the concatenation of two strings, denoted by $\cdot$, recursively as follows.

BASIS STEP: If $w \in \Sigma^*$, then $w \cdot \lambda = w$, where λ is the empty string.

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

DEFINITION 3 The set of *rooted trees,* where a rooted tree consists of a set of vertices containing a distinguished vertex called the *root,* and edges connecting these vertices, can be defined recursively by these steps:

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.

DEFINITION 4 The set of *extended binary trees* can be defined recursively by these steps:

BASIS STEP: The empty set is an extended binary tree.

RECURSIVE STEP: If T_1 and T_2 are disjoint extended binary trees, there is an extended 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 when these trees are nonempty.

DEFINITION 5 The set of full binary trees can be defined recursively by these steps:

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

DEFINITION 6 We define the height $h(T)$ of a full binary tree T recursively.

BASIS STEP: The height of the 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 $h(T) = 1 + \max(h(T_1), h(T_2))$.

5.4 Recursive Algorithms

Ch. 5.4 Q# 9

9. Give a recursive algorithm for finding the sum of the first n odd positive integers.

Base case => when n = 1

odd positive integer = 2n - 1

procedure SumOfOdds (n : positive integer)
if n = 1 then return 1
else return SumOfOdds (n-1) + 2n - 1

ALGORITHM 1 A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n-1)$
{output is $n!$}

ALGORITHM 2 A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n-1)$
{output is a^n}

ALGORITHM 3 A Recursive Algorithm for Computing $\gcd(a, b)$.

procedure *gcd*(a, b: nonnegative integers with $a < b$)
if $a = 0$ **then return** b
else return $gcd(b \textbf{ mod } a, a)$
{output is $\gcd(a, b)$}

ALGORITHM 4 Recursive Modular Exponentiation.

procedure *mpower*(b, n, m: integers with $b > 0$ and $m \geq 2, n \geq 0$)
if $n = 0$ **then**
  return 1
else if n is even **then**
  return $mpower(b, n/2, m)^2$ **mod** m
else
  return $(mpower(b, \lfloor n/2 \rfloor, m)^2 \textbf{ mod } m \cdot b \textbf{ mod } m) \textbf{ mod } m$
{output is b^n **mod** m}

ALGORITHM 5 A Recursive Linear Search Algorithm.

procedure *search*(i, j, x: i, j, x integers, $1 \leq i \leq j \leq n$)
if $a_i = x$ **then**
  return i
else if $i = j$ **then**
  return 0
else
  return $search(i+1, j, x)$
{output is the location of x in $a_1, a_2, \ldots, a_n$ if it appears; otherwise it is 0}

ALGORITHM 6 A Recursive Binary Search Algorithm.

procedure *binary search*(i, j, x: i, j, x integers, $1 \leq i \leq j \leq n$)
$m := \lfloor (i+j)/2 \rfloor$
if $x = a_m$ **then**
  return m
else if ($x < a_m$ and $i < m$) **then**
  return *binary search*$(i, m-1, x)$
else if ($x > a_m$ and $j > m$) **then**
  return *binary search*$(m+1, j, x)$
else return 0
{output is location of x in $a_1, a_2, \ldots, a_n$ if it appears; otherwise it is 0}

ALGORITHM 7 A Recursive Algorithm for Fibonacci Numbers.

procedure *fibonacci*(n: nonnegative integer)
if $n = 0$ **then return** 0
else if $n = 1$ **then return** 1
else return $fibonacci(n-1) + fibonacci(n-2)$
{output is $fibonacci(n)$}

ALGORITHM 8 An Iterative Algorithm for Computing Fibonacci Numbers.

procedure *iterative fibonacci*(n: nonnegative integer)
if $n = 0$ **then return** 0
else
  $x := 0$
  $y := 1$
  for $i := 1$ to $n - 1$
    $z := x + y$
    $x := y$
    $y := z$
  return y
{output is the nth Fibonacci number}

ALGORITHM 9 A Recursive Merge Sort.

procedure *mergesort*($L = a_1, \ldots, a_n$)
if $n > 1$ **then**
  $m := \lfloor n/2 \rfloor$
  $L_1 := a_1, a_2, \ldots, a_m$
  $L_2 := a_{m+1}, a_{m+2}, \ldots, a_n$
  $L := merge(mergesort(L_1), mergesort(L_2))$
{L is now sorted into elements in nondecreasing order}

ALGORITHM 10 Merging Two Lists.

procedure *merge*(L_1, L_2: sorted lists)
$L :=$ empty list
while L_1 and L_2 are both nonempty
  remove smaller of first elements of L_1 and L_2 from its list; put it at the right end of L
  if this removal makes one list empty **then** remove all elements from the other list and append them to L
return L {L is the merged list with elements in increasing order}