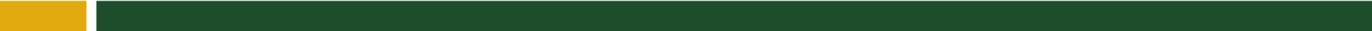


Induction and Recursion

CPE348

Theresa Migler-VonDollen

5.1 Mathematical Induction



Induction

Mathematical induction is used to prove statements of the form:
 $P(n)$ is true for all positive integers.

- Equivalently: If the domain is all positive integers: $\forall nP(n)$.

Two common descriptions:

- Ladder
- Dominoes

Induction

Induction: $(\forall n \in \mathbb{N}), P(n)$ is true

- 1 (Base Case) Show that $P(1)$ is true.
- 2 (Inductive Hypothesis) Suppose that $P(k)$ is true (for some k).
- 3 (Inductive Step) Show that $P(k + 1)$ is true.
- 4 (Conclusion) By steps 1 and 2 and the PMI, $P(n)$ is true for all \mathbb{N} .

Note: We do NOT assume that $P(k)$ is true for all $k \in \mathbb{N}$. Rather we assume that $P(k)$ is true and use that to show that $P(k + 1)$ is true.

Induction

Examples:

Theorem

$$1 \cdot 2 + 2 \cdot 3 + \cdots + n \cdot (n + 1) = \frac{n(n+1)(n+2)}{3}$$

Theorem

$$1 + \frac{1}{4} + \frac{1}{9} + \cdots + \frac{1}{n^2} < 2 - \frac{1}{n} \text{ for all } n \in \mathbb{N}, n > 1.$$

The base case should be $n = 2$.

Theorem

$$3^n < n! \quad \forall n \in \mathbb{N}, n > 6.$$

Theorem

$$6 \mid n^3 - n \quad (\exists c \in \mathbb{Z} \text{ such that } n^3 - n = 6 \cdot c) \text{ for all } n \in \mathbb{N}, n > 0.$$

Induction

Examples:

Theorem

If S is a set with n elements, then S has 2^n subsets.

Theorem

If A_1, A_2, \dots, A_n and B are sets, then

$$(A_1 \cap A_2 \cap A_3 \cap \dots \cap A_n) \cup B = (A_1 \cup B) \cap (A_2 \cup B) \cap \dots \cap (A_n \cup B)$$

Theorem

Let a_1, a_2, \dots, a_n be positive real numbers. The arithmetic mean of these numbers is defined by $A = \frac{a_1 + a_2 + \dots + a_n}{n}$. The geometric mean is defined to be $G = (a_1 a_2 \dots a_n)^{1/n}$.

$$A \geq G$$

5.2 Strong Induction and Well Ordering

Strong Induction

Strong Induction: $(\forall n \in \mathbb{N}), P(n)$ is true

- 1 (Base Case) Show that $P(1)$ is true.
- 2 (Inductive Hypothesis) Suppose, for all natural numbers, k , that $P(1), P(2), \dots, P(k)$ are true.
- 3 (Inductive Step) Show that $P(k + 1)$ is true.
- 4 (Conclusion) By steps 1 and 2 and the PMI, $P(n)$ is true for all \mathbb{N} .

This is equivalent to (regular-weak) induction and the Well Ordering Property.

Strong Induction

Examples:

Theorem

A postage of n cents can be formed using just 4-cent and 7-cent stamps for all values of $n \geq 18$.

- The bases cases are 18, 19, 20, and 21.

Theorem

Every positive integer n can be written as a sum of distinct powers of two.

- For the inductive step, consider the case of when $k + 1$ is even and when $k + 1$ is odd. When $k + 1$ is even, note that $\frac{k+1}{2} \in \mathbb{Z}$.

The Well-Ordering Property

The Well-Ordering Property

Every nonempty set of nonnegative integers has a least element.

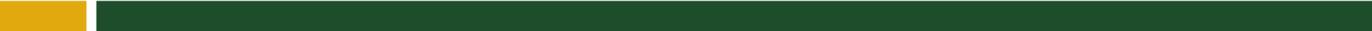
Examples:

In a round-robin tournament every player plays every other player exactly once and each match has exactly one winner and one loser.

We say that players p_1, p_2, \dots, p_m form a *cycle* if p_1 beats p_2 , p_2 beats p_3 , \dots , and p_m beats p_1 .

If there is a cycle of length $m \geq 3$ among the players, there must be a cycle of three of these players.

5.3 Recursive Definitions and Structural Induction



Recursively Defined Functions

Definition

We can define a function (on the set of nonnegative integers) using a **recursive definition** by defining the following two steps:

- 1 *Basis Step*: Specify the value of the function at zero (or one...)
- 2 *Recursive Step*: Give the rule for finding its value at an integer from its values at smaller integers.

Example:

Give a recursive definition for $n!$.

- Why is the basis step necessary?

Recursively Defined Functions

Examples:

- Suppose that f is defined recursively as follows:
 - $f(0) = 3$ and $f(n + 1) = 2f(n) + 3$
 - What is the value of $f(5)$?
- Let $a \in \mathbb{R}$, $a \neq 0$ and $n \in \mathbb{Z}^+$.
 - Give a recursive definition for $g(n) = a^n$.
- State the recursive definition of the Fibonacci sequence.
 - Do we only need one base case?

Definition

A **well defined** function is a function such that for every positive integer, the value of the function at this integer is determined in an unambiguous way.

Recursively defined functions are well defined.

Recursively Defined Sets and Structures

We can define sets recursively just as we define functions.

Example:

- Let S be the set defined as follows:
 - Basis Step: $7 \in S$
 - Recursive Step: If $x \in S$ and $y \in S$, then $x + y \in S$.
 - What are the elements of S ?

- We can define the set of *rooted trees* recursively:
 - Basis Step: A single vertex, r , is a rooted tree.
 - Recursive Step: Suppose that T_1, T_2, \dots, T_n are disjoint rooted trees with roots r_1, r_2, \dots, r_n respectively. Then the graph formed by starting with a root r (which is not already in one of the trees), and adding an edge from r to each of the vertices r_1, r_2, \dots, r_n is also a rooted tree.

Recursively Defined Sets and Structures

Definition

A **full binary tree** is a tree in which every vertex other than the leaves has exactly two children.

We can define full binary trees recursively.

- Basis Step: There is a full binary tree consisting of a single vertex, r .
- Recursive Step: If T_1 and T_2 are disjoint full binary trees, there is a full disjoint binary tree 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 .

Theorems About Full Binary Trees

Theorem

Let T be a nonempty full binary tree. Then:

- If T has i internal vertices, then the number of leaves is $i + 1$.*
- If T has a total of n vertices, then the number of internal vertices is $i = (n - 1)/2$.*
- If T has ℓ leaves, then the total number of vertices is $n = 2\ell - 1$*

Proofs on board.

5.4 Recursive Algorithms



Recursive Algorithms

Definition

An algorithm is called **recursive** if it solves a problem by reducing it to an instance of the same problem with smaller input.

Example:

Give a recursive algorithm for computing $n!$.

factorial(n)

Input: A nonnegative integer, n .

Output: $n!$

1: **if** $n == 0$ **then**

return 1

2: **else**

return $n \cdot \text{factorial}(n - 1)$

Recursive Algorithms

- Give a recursive algorithm for computing a^n ($a \in \mathbb{R}, a \neq 0$).
- Give a recursive algorithm for computing the n^{th} Fibonacci number.

Prove that the algorithms are correct.

Recursive Binary Search

BinarySearch(A, x, \min, \max)

Input: An array of sorted numbers A , values x , \min , and \max .

Output: The position of x in A , or a statement that x is not in A .

(Initially $\min = 0$ and $\max = n - 1$)

1: **if** $\max < \min$ **then**

return " $x \notin A$ "

2: **else**

3: **if** $x < A[\lfloor (\max + \min)/2 \rfloor]$ **then**

return BinarySearch($A, x, \min, \lfloor (\max + \min)/2 \rfloor - 1$)

4: **else if** $x > A[\lfloor (\max + \min)/2 \rfloor]$ **then**

return BinarySearch($A, x, \lfloor (\max + \min)/2 \rfloor + 1, \max$)

5: **else**

return $\lfloor (\max + \min)/2 \rfloor$

Binary Search - Correctness

Is it correct?

Lemma

If x is an element in the list, BinarySearch will return the correct position of x .

Proof.

By induction, in class. □

Lemma

If x is not an element of the list, BinarySearch will return that there is no such element in the list.

We must also guarantee that BinarySearch terminates.

Recurrence Relations - Binary Search

What is the running time of BinarySearch?

$$T(n) = T(n/2) + O(1)$$

- We will see more about how to solve recurrence relations in Chapter 7.

$$= O(1 + 1 + \dots + 1) = O(\log_2(n))$$

Merge Sort

Sorting

Input: A list of numbers, $a_1, a_2, a_3, \dots, a_n$.

Goal: Return a list of the same numbers sorted in increasing order.

MergeSort($A[0, \dots, n - 1]$)

Input: A list of unsorted numbers $A[0, \dots, n - 1]$

Output: The same list, sorted in increasing order

1: **if** $n \leq 1$ **then**

return A

2: **else**

return Merge(MergeSort($A[0, \dots, \lfloor n/2 \rfloor]$), MergeSort($A[\lfloor n/2 \rfloor + 1, \dots, n - 1]$))

Merge Sort

Sorting

Input: A sequence of numbers, $a_1, a_2, a_3, \dots, a_n$.

Goal: Return a list of the same numbers sorted in increasing order.

Merge($x[0, \dots, k - 1], y[0, \dots, \ell - 1]$)

Input: Two sorted lists, $x[0, \dots, k - 1]$ and $y[0, \dots, \ell - 1]$

Output: One sorted list that contains all elements of both lists.

- 1: **if** $x = \emptyset$ **then return** y
 - 2: **if** $y = \emptyset$ **then return** x
 - 3: **if** $x[0] \leq y[0]$ **then**
 return $x[0] \circ \text{Merge}(x[1, \dots, k - 1], y[0, \dots, \ell - 1])$
 - 4: **else**
 return $y[0] \circ \text{Merge}(x[0, \dots, k - 1], y[1, \dots, \ell - 1])$
-

Merge Sort - Correctness

Theorem

Merge *correctly merges two sorted lists.*

Proof.

We will proceed by induction on the total size of the lists being merged. (We will prove that Merge correctly merges two sorted lists of total size n .)

□ Base Case: ($n = 1$)

This will only occur if either x or y is empty, and the other list has exactly 1 element.

▣ Merge correctly merges the empty list with any other sorted list.

□ Inductive Hypothesis: Suppose that Merge correctly merges two sorted lists of total size equal to n .

Merge Sort - Correctness

Theorem

Merge correctly merges two sorted lists.

Proof (Cont.)

- Inductive Step: Consider two sorted lists with total size $n + 1$.
 - In steps 3 and 4 of the algorithm, Merge correctly places the smallest element at the beginning of the list.
 - Merge then concatenates that element with the Merge of the remaining elements of the two lists.
 - The total size of the remaining two lists is n .
 - By the Inductive Hypothesis, Merge correctly merges the remainder.
- Conclusion: Therefore, by PMI, Merge correctly merges two sorted lists.



Merge Sort - Running Time

- What is the running time?
 - ▣ $T(n) = 2T(n/2) + O(n) = O(n \log n)$