

# 离散数学及其应用 第5章 归纳与递归

---

## 5. Induction and Recursion

### 5.1 Mathematical Induction (MI)

#### 5.1.1 The First Principle of Mathematical Induction

#### 5.1.2 The Second Principle of Mathematical Induction

#### 5.1.3 Well-Ordering Property

### 5.2 Recursive Definition and Structural Induction

#### 5.2.1 Recursive Definition

#### 5.2.2 Structural Induction

#### 5.2.3 Generalized Induction

#### 5.2.4 Recursive Algorithms

## 5. Induction and Recursion

### 5.1 Mathematical Induction (MI)

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.

#### 5.1.1 The First Principle of Mathematical Induction

Written in logical expressions:

$\{P(1) \wedge \forall k [P(k) \rightarrow P(k+1)]\} \rightarrow \forall n P(n)$ , 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:

1. Inductive base: Establish  $P(1)$
2. Inductive step: Prove that  $P(k) \rightarrow P(k+1)$  for  $k \geq 1$
3. Conclusion: The inductive base and the inductive step together imply  $P(n) \forall n \geq 1$ .

The first principle of MI has a more general form:

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

1. Inductive base: Establish  $P(k)$
2. Inductive step: Prove that  $P(n) \rightarrow P(n+1)$  for  $n \geq k$
3. Conclusion: The inductive base and the inductive step together imply  $P(n) \forall n \geq k$ .

### 5.1.2 The Second Principle of Mathematical Induction

Also called Strong Induction, Complete Induction.

$$\{P(n_0) \wedge \forall k [k \geq n_0 \wedge P(n_0) \wedge P(n_0+1) \wedge \dots \wedge P(k) \rightarrow P(k+1)]\} \rightarrow \forall n P(n)$$

The procedure :

1. Inductive base: Establish  $P(n_0)$
2. Inductive step: Prove  $P(n_0) \wedge P(n_0+1) \wedge \dots \wedge P(k) \rightarrow P(k+1)$
3. Conclusion: The inductive base and the inductive step allow one to conclude that  $P(n) \forall n \geq n_0$

### 5.1.3 Well-Ordering Property

If every nonempty subset of a set has a least element, then we say the set is **well-ordered**. The set of all nonnegative integers and the set of all natural number are well-ordered.

 The validity of MI is based on well-ordering property.

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 according to the inductive base,  $m \neq 1$ ,  $m-1$  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

- Can be used to prove a conjecture once it has been made and is true.
- Proofs do not provide insights as to why theorems are true. You can prove a theorem by MI even if you do not have the slightest idea why it is true!
- Cannot be used to find new theorems.

📌 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!

## 5.2 Recursive Definition and Structural Induction

### 5.2.1 Recursive Definition

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

- Basis Step: Specify the value of the function at 0.
- Recursive Step: Give the rules for finding its value at an integer from its value at smaller integers.

📌 Recursively defined functions are well-defined.

Proof:

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

$P(0)$  is true.

Assume that  $P(n-1)$  is true. Then f is well-defined at  $n$ , since  $f(n)$  is given in terms of some  $f(n-1)$ .

Euclidean algorithm is recursive, so its complexity is a little hard to describe. The following theorem is a nice solution to this problem and an elegant application of recursively defined

functions.

### □ Lamé's Theorem 拉梅定理

Let  $a, b$  be positive integers with  $a \geq b$ . 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:

The Euclidean algorithm is represented as the following equations.

$$\begin{aligned} r_0 &= r_1 q_1 + r_2 & 0 \leq r_2 < r_1 \\ r_1 &= r_2 q_2 + r_3 & 0 \leq r_3 < r_2 \\ &\dots \\ r_{n-2} &= r_{n-1} q_{n-1} + r_n & 0 \leq r_n < r_{n-1} \\ r_{n-1} &= r_n q_n \end{aligned}$$

Let the number of decimal digits in  $b$  be  $k$ , then we need to prove  $n \leq 5k$ .

In the equations above,  $q_1, q_2, \dots, q_{n-1} \geq 1, q_n \geq 2, r_n < r_{n-1}$

This implies a Fibonacci sequence.

$$\begin{aligned} r_n &\geq 1 = f_2 \\ r_{n-1} &\geq 2r_n \geq 2f_2 = f_3 \\ r_{n-2} &\geq r_{n-1} + r_n \geq f_3 + f_2 = f_4 \\ r_{n-3} &\geq r_{n-2} + r_{n-1} \geq f_3 + f_4 = f_5 \\ &\dots \\ r_2 &\geq r_3 + r_4 \geq f_{n-1} + f_{n-2} = f_n \\ b = r_1 &\geq r_2 + r_3 \geq f_n + f_{n-1} = f_{n+1} \end{aligned}$$

In Fibonacci sequence, let  $a = (1+\sqrt{5})/2$ , then  $f_n > a^{n-2}$  whenever  $n \geq 3$ . So  $b > a^{n-1}$ .

So  $k = \lg b > (n-1) \lg a > (n-1) / 5$ . So  $n \leq 5k$ .

### Recursively defined sets

- 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.

**String**, the set  $\Sigma^*$  of strings over the alphabet  $\Sigma$

- Basis Step:  $\lambda \in \Sigma^*$  ( $\lambda$  is the empty string)
- Recursive Step: If  $w$  is in  $\Sigma^*$  and  $x$  is in  $\Sigma$ , then  $wx \in \Sigma^*$ .

Here " $wx$ " is **string concatenation**. Let  $\Sigma$  be a set of symbols and  $\Sigma^*$  be the set of strings formed from the symbols in  $\Sigma$ . We can define the concatenation of two strings, denoted by " $\cdot$ ", recursively as follows. (Often  $w_1 \cdot w_2$  is written as  $w_1 w_2$ .)

- 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  $w_1 \cdot (w_2 \cdot x) = (w_1 \cdot w_2) \cdot x$ .

🍎e.g.

Another important use of recursive definitions to define well-formed formulae of various type.

**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.

🍎e.g.

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

Building up **rooted trees**:

- Basis Step: A single vertex  $r$  is a rooted tree.

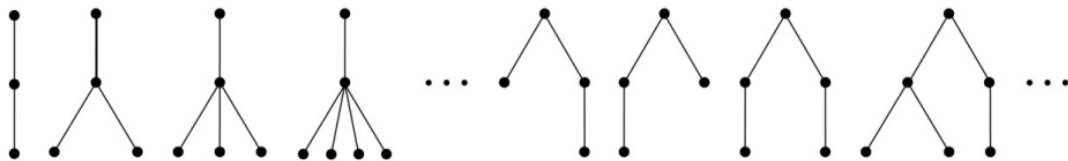
- Recursive Step: Suppose that  $T_1, T_2, \dots, T_n$  are disjoint rooted trees with roots  $r_1, r_2, \dots, r_n$ , respectively. Then the graph formed by starting with a root  $r$ , which is not in any of the rooted trees  $T_1, T_2, \dots, T_n$ , and adding an edge from  $r$  to each of the vertices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

Basis step    •

Step 1



Step 2



Building up **full binary trees** :

- 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  $T_2$ .

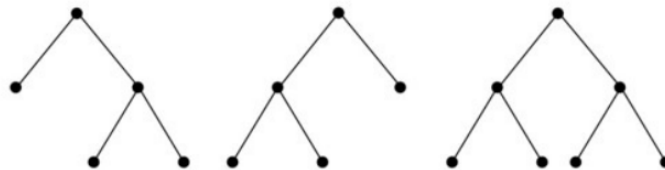
Basis step



Step 1



Step 2



## 5.2.2 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 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.

🔴e.g.

Show that every well-formed formula for compound propositions 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  $(\neg p)$ ,  $(p \vee q)$ ,  $(p \wedge q)$ ,  $(p \rightarrow q)$ ,  $(p \leftrightarrow q)$ , and they all contain equal numbers of left and right parentheses.

🔴e.g. Structural Induction and Binary Trees

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

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

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

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,  $n(T) = 1 \leq 2^{0+1} - 1 = 1$ .
- **RECURSIVE STEP:** Assume  $n(T_1) \leq 2^{h(T_1)+1} - 1$  and also  $n(T_2) \leq 2^{h(T_2)+1} - 1$  whenever  $T_1$  and  $T_2$  are full binary trees.

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

### 5.2.3 Generalized Induction

**Generalized induction** is used to prove results about sets other than the integers that have the well-ordering property. For example, consider an ordering on  $\mathbb{N} \times \mathbb{N}$ , 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.

● e.g.

**Example 11:** Suppose that  $a_{m,n}$  is defined for  $(m,n) \in \mathbb{N} \times \mathbb{N}$

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

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

**Solution:** Use generalized induction.

**BASIS STEP:**  $a_{0,0} = 0 + (0 \cdot 1)/2$

**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**  $\mathbb{N} \times \mathbb{N}$ .

**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**

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




## 5.2.4 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.

**Recursion** Successively reducing the computation to the evaluation of the function on a smaller integer.

**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.

 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.