Algorithms 2016: Mathematical Induction

(Based on [Manber 1989])

Yih-Kuen Tsay

1 Induction Principles

The Standard Induction Principle

- Let T be a theorem that includes a parameter n whose value can be any natural number.
- Here, natural numbers are positive integers, i.e., 1, 2, 3, ..., excluding 0 (sometimes we may include 0).
- To prove T, it suffices to prove the following two conditions:
 - -T holds for n=1. (Base case)
 - For every n > 1, if T holds for n 1, then T holds for n. (Inductive step)
- The assumption in the inductive step that T holds for n-1 is called the *induction hypothesis*.

A Starter

Theorem 1 (2.1). For all natural numbers x and n, $x^n - 1$ is divisible by x - 1.

Proof. (Suggestion: try to follow the structure of this proof when you present a proof by induction.) The proof is by induction on n.

```
Base case (n = 1): x - 1 is trivially divisible by x - 1.
```

Inductive step (n > 1): $x^n - 1 = x(x^{n-1} - 1) + (x - 1)$. $x^{n-1} - 1$ is divisible by x - 1 from the induction hypothesis and x - 1 is divisible by x - 1. Hence, $x^n - 1$ is divisible by x - 1.

Note: a is divisible by b if there exists an integer c such that $a = b \times c$.

Variants of Induction Principle

Theorem 2. If a statement P, with a parameter n, is true for n = 1, and if, for every $n \ge 1$, the truth of P for n implies its truth for n + 1, then P is true for all natural numbers.

Theorem 3 (Strong Induction). If a statement P, with a parameter n, is true for n = 1, and if, for every n > 1, the truth of P for all natural numbers < n implies its truth for n, then P is true for all natural numbers.

Theorem 4. If a statement P, with a parameter n, is true for n = 1 and for n = 2, and if, for every n > 2, the truth of P for n - 2 implies its truth for n, then P is true for all natural numbers.

2 Design by Induction

Design by Induction: First Glimpse

Problem 5. Given two sorted arrays A[1..m] and B[1..n] of positive integers, find their smallest common element; returns 0 if no common element is found.

- Assume the elements of each array are in ascending order.
- **Obvious solution**: take one element at a time from A and find out if it is also in B (or the other way around).
- How efficient is this solution?
- Can we do better?

Design by Induction: First Glimpse (cont.)

- There are m+n elements to begin with.
- Can we pick out one element such that either (1) it is the element we look for or (2) it can be ruled out from subsequent searches?
- In the second case, we are left with the same problem but with m+n-1 elements?
- Idea: compare the current first elements of A and B.
 - 1. If they are equal, then we are done.
 - 2. If not, the smaller one cannot be the smallest common element.

Design by Induction: First Glimpse (cont.)

Below is the complete solution:

```
 \begin{aligned} \textbf{Algorithm SCE}(A, m, B, n) : integer; \\ \textbf{begin} \\ & \textbf{if } m = 0 \textbf{ or } n = 0 \textbf{ then } SCE := 0; \\ & \textbf{if } A[1] = B[1] \textbf{ then} \\ & SCE := A[1]; \\ & \textbf{else if } A[1] < B[1] \textbf{ then} \\ & SCE := SCE(A[2..m], m - 1, B, n); \\ & \textbf{else } SCE := SCE(A, m, B[2..n], n - 1); \\ \textbf{end} \end{aligned}
```

Why Induction Works

- Computations carried out by a computer/machine can, in essence, be understood as mathematical functions.
- To solve practical problems with computers,
 - objects/things in a practical domain must be modeled as (mostly discrete) mathematical structures/sets, and
 - various manipulations of the objects become functions on the corresponding mathematical structures.
- Many mathematical structures are naturally defined by induction.
- Functions on inductive structures are also naturally defined by induction (recursion).

Recursively/Inductively-Defined Sets

- The natural numbers (including 0):
 - 1. Base case: 0 is a natural number.
 - 2. Inductive step: if n is a natural number, then n+1 is also a natural number.
- Binary trees:
 - 1. Base case: the empty tree is a binary tree.
 - 2. Inductive step: if L and R are binary trees, then a node with L and R as the left and the right children is also a binary tree.
- Nonempty binary trees:
 - 1. Base case: a single root node (without any child) is a binary tree.
 - 2. Inductive step: if L and R are binary trees, then a node with L as the left child and/or R as the right child is also a binary tree.

Structural Induction

- Structural induction is a generalization of mathematical induction on the natural numbers.
- It is used to prove that some proposition P(x) holds for all x of some sort of recursively/inductively defined structure such as binary trees.
- Proof by structural induction:
 - 1. Base case: the proposition holds for all the minimal structures.
 - 2. Inductive step: if the proposition holds for the immediate substructures of a certain structure S, then it also holds for S.

3 Proofs by Induction

Another Simple Example

Theorem 6 (2.4). If n is a natural number and 1+x>0, then $(1+x)^n \ge 1+nx$.

• Below are the key steps:

$$(1+x)^{n+1} = (1+x)(1+x)^n$$
 {induction hypothesis and $1+x > 0$ }
 $\geq (1+x)(1+nx)$
= $1+(n+1)x+nx^2$
 $\geq 1+(n+1)x$

• The main point here is that we should be clear about how conditions listed in the theorem are used.

3.1 Proving vs. Computing

Proving vs. Computing

Theorem 7 (2.2). $1+2+\cdots+n=\frac{n(n+1)}{2}$.

• This can be easily proven by induction.

• Key steps:
$$1 + 2 + \dots + n + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{n^2 + n + 2n + 2}{2} = \frac{n^2 + 3n + 2}{2} = \frac{(n+1)(n+2)}{2} = \frac{(n+1)((n+1)+1)}{2}$$
.

- Induction seems to be useful only if we already know the sum.
- What if we are asked to compute the sum of a series?
- Let's try $8 + 13 + 18 + 23 + \cdots + (3 + 5n)$.

Proving vs. Computing (cont.)

- Idea: guess and then verify by an inductive proof!
- The sum should be of the form $an^2 + bn + c$.
- By checking n = 1, 2, and 3, we get $\frac{5}{2}n^2 + \frac{11}{2}n$.
- Verify this for all n, i.e., the following theorem, by induction.

Theorem 8 (2.3). $8+13+18+23+\cdots+(3+5n)=\frac{5}{2}n^2+\frac{11}{2}n$.

3.2 Counting Regions

Counting Regions

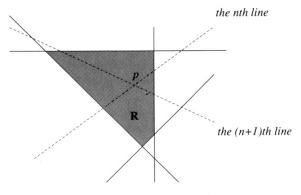


Figure 2.1 n+1 lines in general position.

Source: [Manber 1989].

Counting Regions (cont.)

Theorem 9 (2.5). The number of regions in the plane formed by n lines in general position is $\frac{n(n+1)}{2} + 1$.

A set of lines are in **general position** if (1) no two lines are parallel and (2) no three lines intersect at a common point.

- We observe that $\frac{n(n+1)}{2} = 1 + 2 + \cdots + n$.
- So, it suffices to prove the following:

Lemma 10. Adding one more line (the n-th line) to n-1 lines in general position in the plane increases the number of regions by n.

3.3 A Summation Problem

A Summation Problem

$$\begin{array}{rcl}
1 & = & 1 \\
3+5 & = & 8 \\
7+9+11 & = & 27 \\
13+15+17+19 & = & 64 \\
21+23+25+27+29 & = & 125
\end{array}$$

Theorem 11. The sum of row n in the triangle is n^3 .

Examine the difference between rows i+1 and i ...

Lemma 12. The last number in row n + 1 is $n^2 + 3n + 1$.

A Simple Inequality

Theorem 13 (2.7). $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n} < 1$, for all $n \ge 1$.

- There are at least two ways to select n terms from n+1 terms.
 - 1. $\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^n}\right) + \frac{1}{2^{n+1}}$.
 - 2. $\frac{1}{2} + (\frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^n} + \frac{1}{2^{n+1}}).$
- The second one leads to a successful inductive proof:

$$\frac{1}{2} + \left(\frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^n} + \frac{1}{2^{n+1}}\right)$$

$$= \frac{1}{2} + \frac{1}{2}\left(\frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^{n-1}} + \frac{1}{2^n}\right)$$

$$< \frac{1}{2} + \frac{1}{2}$$

$$= 1$$

3.4 Euler's Formula

Euler's Formula

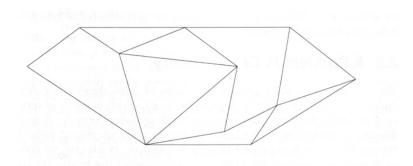


Figure 2.2 A planar map with 11 vertices, 19 edges, and 10 faces.

Source: [Manber 1989].

Euler's Formula (cont.)

Theorem 14 (2.8). The number of vertices (V), edges (E), and faces (F) in an arbitrary connected planar graph are related by the formula V + F = E + 2.

The proof is by induction on the number of faces.

Base case: graphs with only one face are trees . . .

Lemma 15. A tree with n vertices has n-1 edges.

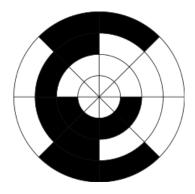
Inductive step: for a graph with more than one faces, there must be a cycle in the graph. Remove one edge from the cyle \dots

3.5 Gray Codes

Gray Codes

- A **Gray code** (after Frank Gray) for *n* objects is a binary-encoding scheme for naming the *n* objects such that the *n* names can be arranged in a *circular* list where *any two adjacent names*, *or code words*, *differ by only one bit*.
- Examples:
 - -00, 01, 11, 10
 - -000, 001, 011, 010, 110, 111, 101, 100
 - -000,001,011,111,101,100

A Gray Code in Picture



A rotary encoder using a 3-bit Gray code.

Source: Wikipedia.

Gray Codes (cont.)

Theorem 16 (2.10). There exist Gray codes of length $\frac{k}{2}$ for any positive even integer k.

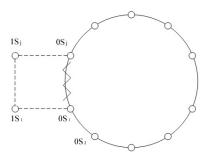


Figure 2.3 Constructing a Gray code of size 2k

Source: [Manber 1989] (adapted).

Note: j in the figure equals 2(k-1) and hence j+2 equals 2k.

Gray Codes (cont.)

Theorem 17 (2.10+). There exist Gray codes of length $\log_2 k$ for any positive integer k that is a power of 2.

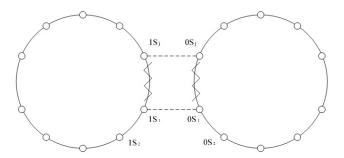


Figure 2.4 Constructing a Gray code from two smaller ones

Source: [Manber 1989] (adapted).

Gray Codes (cont.)

- $00, 01, 11, 10 \text{ (for } 2^2 \text{ objects)}$
- 000, 001, 011, 010 (add a 0)
- 100, 101, 111, 110 (add a 1)
- Combine the preceding two codes (read the second in reversed order): 000, 001, 011, 010, 110, 111, 101, 100 (for 2³ objects)

Gray Codes (cont.)

Theorem 18 (2.11–). There exist Gray codes of length $\lceil \log_2 k \rceil$ for any positive even integer k.

To generalize the result and ease the proof, we allow a Gray code to be *open* where the last name and the first name may differ by more than one bit.

Theorem 19 (2.11). There exist Gray codes of length $\lceil \log_2 k \rceil$ for any positive integer $k \geq 2$. The Gray codes for the even values of k are closed, and the Gray codes for odd values of k are open.

Gray Codes (cont.)

- 00, 01, 11 (open Gray code for 3 objects)
- 000, 001, 011 (add a 0)
- 100, 101, 111 (add a 1)
- Combine the preceding two codes (read the second in reversed order): 000, 001, 011, 111, 101, 100 (closed Gray code for 6 objects)

Gray Codes (cont.)

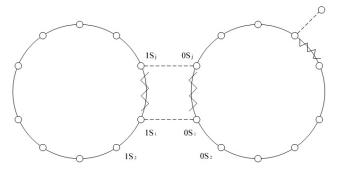


Figure 2.5 Constructing an open Gray code

Source: [Manber 1989] (adapted).

4 Reversed Induction

Arithmetic vs. Geometric Mean

Theorem 20 (2.13). If $x_1, x_2, ..., x_n$ are all positive numbers, then $(x_1 x_2 ... x_n)^{\frac{1}{n}} \le \frac{x_1 + x_2 + ... + x_n}{n}$.

First use the standard induction to prove the case of powers of 2 and then use the reversed induction principle below to prove for all natural numbers.

Theorem 21 (Reversed Induction Principle). If a statement P, with a parameter n, is true for an infinite subset of the natural numbers, and if, for every n > 1, the truth of P for n implies its truth for n - 1, then P is true for all natural numbers.

Arithmetic vs. Geometric Mean (cont.)

- For all powers of 2, i.e., $n = 2^k$, $k \ge 1$: by induction on k.
- Base case: $(x_1x_2)^{\frac{1}{2}} \leq \frac{x_1+x_2}{2}$, squaring both sides
- Inductive step:

$$\begin{array}{ll} & (x_1x_2\cdots x_{2^{k+1}})^{\frac{1}{2^{k+1}}}\\ =& [(x_1x_2\cdots x_{2^{k+1}})^{\frac{1}{2^k}}]^{\frac{1}{2}}\\ =& [(x_1x_2\cdots x_{2^k})^{\frac{1}{2^k}}(x_{2^k+1}x_{2^k+2}\cdots x_{2^{k+1}})^{\frac{1}{2^k}}]^{\frac{1}{2}}\\ \leq& \frac{(x_1x_2\cdots x_{2^k})^{\frac{1}{2^k}}+(x_{2^k+1}x_{2^k+2}\cdots x_{2^{k+1}})^{\frac{1}{2^k}}}{2}, \text{ from the base case}\\ \leq& \frac{\frac{x_1+x_2+\cdots +x_{2^k}}{2^k}+\frac{x_{2^k+1}+x_{2^k+2}+\cdots +x_{2^{k+1}}}{2^k}}{2}, \text{ from the Ind. Hypo.}\\ =& \frac{x_1+x_2+\cdots +x_{2^{k+1}}}{2^{k+1}} \end{array}$$

Arithmetic vs. Geometric Mean (cont.)

- For all natural numbers: by reversed induction on n.
- Base case: the theorem holds for all powers of 2.
- Inductive step: observe that

$$\frac{x_1 + x_2 + \dots + x_{n-1}}{n-1} = \frac{x_1 + x_2 + \dots + x_{n-1} + \frac{x_1 + x_2 + \dots + x_{n-1}}{n-1}}{n}.$$

Arithmetic vs. Geometric Mean (cont.)

$$(x_1 x_2 \cdots x_{n-1} (\frac{x_1 + x_2 + \cdots + x_{n-1}}{n-1}))^{\frac{1}{n}} \leq \frac{x_1 + x_2 + \cdots + x_{n-1} + \frac{x_1 + x_2 + \cdots + x_{n-1}}{n-1}}{n}$$
 (from the Ind. Hypo.)
$$(x_1 x_2 \cdots x_{n-1} (\frac{x_1 + x_2 + \cdots + x_{n-1}}{n-1}))^{\frac{1}{n}} \leq \frac{x_1 + x_2 + \cdots + x_{n-1}}{n-1}$$

$$(x_1 x_2 \cdots x_{n-1} (\frac{x_1 + x_2 + \cdots + x_{n-1}}{n-1})) \leq (\frac{x_1 + x_2 + \cdots + x_{n-1}}{n-1})^n$$

$$(x_1 x_2 \cdots x_{n-1}) \leq (\frac{x_1 + x_2 + \cdots + x_{n-1}}{n-1})^{n-1}$$

$$(x_1 x_2 \cdots x_{n-1})^{\frac{1}{n-1}} \leq (\frac{x_1 + x_2 + \cdots + x_{n-1}}{n-1})$$

5 Loop Invariants

Loop Invariants

- An *invariant* at some point of a program is an assertion that holds whenever execution of the program reaches that point.
- Invariants are a bridge between the static text of a program and its dynamic computation.
- An invariant at the front of a while loop is called a *loop invariant* of the while loop.
- A loop invariant is formally established by induction.
 - Base case: the assertion holds right before the loop starts.
 - Inductive step: assuming the assertion holds before the *i*-th iteration ($i \ge 1$), it holds again after the iteration.

Number Conversion

```
\begin{aligned} \textbf{Algorithm Convert\_to\_Binary}(n);\\ \textbf{begin} \\ t &:= n;\\ k &:= 0;\\ \textbf{while } t > 0 \textbf{ do}\\ k &:= k + 1;\\ b[k] &:= t \bmod 2;\\ t &:= t \text{ div } 2;\\ \textbf{end} \end{aligned}
```

Number Conversion (cont.)

Theorem 22 (2.14). When Algorithm Convert_to_Binary terminates, the binary representation of n is stored in the array b.

Lemma 23. If m is the integer represented by the binary array b[1..k], then $n = t \cdot 2^k + m$ is a loop invariant of the while loop.

See separate handout for a detailed proof.