Discrete Mathematics

Unit 5: Induction, Recursion and Complexity

Syedur Rahman

Lecturer, CSE Department

North South University

syedur.rahman@wolfson.oxon.org

Acknowledgements

- These lecture notes contain some material from the following sources:
  - [Rosen]: *Discrete Mathematics and Its Applications* by K. Rosen, 5th Edition, Tata McGraw-Hill Edition.
  - [ICM]: *Introduction to Computer Mathematics* by C. Runciman, 2003

Proof by Weak Natural Induction

A proof by mathematical induction that $P(n)$ is true for every positive integer n consists of two steps:

BASIC STEP: The proposition $P(1)$ is shown to be true.

INDUCTIVE STEP: The implication $P(k) \Rightarrow P(k+1)$ is shown to be true for every positive integer k.

$P(k)$ is called the inductive hypothesis for the proof.

This can be expressed as a rule of inference as:

$$[P(1) \wedge \forall k\, P(k) \Rightarrow P(k+1)] \Rightarrow \forall n\, P(n)$$

An example of natural induction

You are required to prove that $1+2+3+\ldots+n = n(n+1)/2$
We say $P(k)$ is true iff $1+2+3+\ldots+n = k(k+1)/2$

BASIC STEP: $P(1)$ is true since $1 = 1(1+1)/2$
INDUCTIVE STEP: If $P(k)$ is true then
$(1)+(2)+(3)+\ldots+(k)+(k+1) = k(k+1)/2 + (k+1)$
$= (k^2+k)/2 + (k+1)$
$= (k^2 + k + 2k + 2)/2$
$= (k+1)(k+2)/2$
So we have established that if $P(k)$ is true then
$(1)+(2)+(3)+\ldots+(k)+(k+1) = (k+1)(k+2)/2$ i.e. $P(k+1)$ is true.

From the base case $P(1)$ is true. From the inductive step $P(k) \Rightarrow P(k+1)$ i.e. $P(1) \Rightarrow P(2)$ and then $P(2) \Rightarrow P(3)$ and so on. So $P(k)$ is true for all positive integers k. Therefore we have proved the relationship using induction.

Proof by Strong Natural Induction

A proof by strong natural induction that $P(n)$ is true for every positive integer n consists of two steps:

BASIC STEP: The proposition $P(1)$ is shown to be true.

INDUCTIVE STEP: It is shown that $[P(1) \wedge P(2) \wedge P(3) \wedge \ldots \wedge P(k)] \Rightarrow P(k+1)$ is true for every positive integer k.

The two forms of natural induction are equivalent, i.e. each can be shown to be a valid proof technique assuming the other.

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.

An example algorithm for computing a^n:

Procedure *power*(*a*: non-zero real number, *n*: nonnegative integer)

If $n = 0$ **then** *power*(*a*, *n*) := 1

Else *power*(*a*, *n*) := *a* x *power*(*a*, *n*-1)

Recursive Definitions

Some functions may be defined recursively, where there is one (or more) base case and the values of the function are dependent on the previous values using recursion.

For example: The factorial function

$f(0) = 1$ BASE DEFINITION

$f(n) = n * f(n\text{-}1)$ RECURSIVE DEFINITION

Another example: The Fibonacci sequence

$f(0) = 0$

$f(1) = 1$ BASE DEFINITIONS

$f(n) = f(n\text{-}1) + f(n\text{-}2)$ RECURSIVE DEFINITION

$$F(n) := \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

Complexity, Big Omega, Big Theta

Big-O (upper bound): Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $O(g(x))$ if there are constants C and k such that: $|f(x)| \leq C|g(x)|$ whenever $x>k$. We say that $f(x)$ grows no faster than $g(x)$.

E.g. $x^2 + x + 1$ is $O(x^2)$, can be proved if C=3 and k=2

$7x^2$ is $O(x^3)$, can be proved if C=1 and k=7

x^2 is $O(x^2 + x + 1)$, can be proved if C=1 and k=1

Big-Omega (lower bound): Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Omega(g(x))$ if there are constants C and k such that: $|f(x)| \geq C|g(x)|$ whenever $x>k$. Generally if $f(x)$ is $\Omega(g(x))$ then $g(x)$ is $O(f(x))$. We say that $f(x)$ grows no faster than $g(x)$.

Big-Theta (upper and lower bound): Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Theta(g(x))$, if and only if $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$. We say that $f(x)$ is of order $g(x)$.

Commonly Used Terminology for Complexity of Algorithms

Complexity	Terminology
$O(1)$	Constant Complexity
$O(\log n)$	Logarithmic Complexity
$O(n)$	Linear Complexity
$O(n \log n)$	$n \log n$ Complexity
$O(n^k)$	Polynomial Complexity
$O(k^n)$, where $k>1$	Exponential Complexity
$O(n!)$	Factorial Complexity

This list is in order of increasing complexity.
n is a variable whereas *k* is a constant.