Discrete Mathematics

Chapter 7: Advanced Counting Techniques

Department of Mathematics
The FPT university

Chapter 7: Introduction

Topics covered:

Chapter 7: Introduction

Topics covered:

7.1 Recurrence Relations

Chapter 7: Introduction

Topics covered:

Chapter 7: Introduction

Topics covered:

7.1 Recurrence Relations

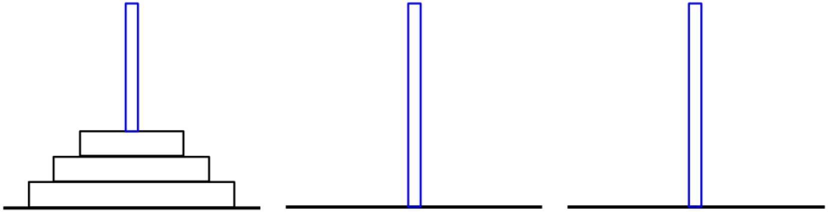
Example 1.(Compound Interest) A person deposited \$10,000 in a saving account at the rate of 11% a year with interest compounded annually.

Example 1.(Compound Interest) A person deposited $10,000 in a saving account at the rate of 11% a year with interest compounded annually. How much will be in the account after 30 years?

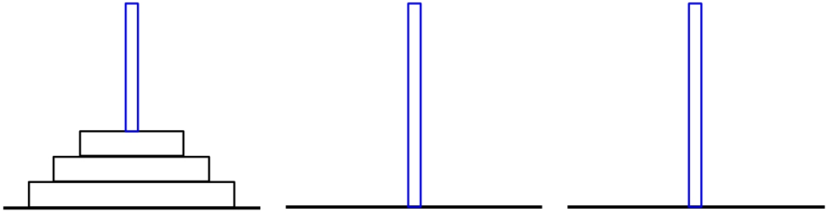
Example 1.(Compound Interest) A person deposited \$10,000 in a saving account at the rate of 11% a year with interest compounded annually. How much will be in the account after 30 years?

Example 2. A young pair of rabits (one of each sex) is placed on an island. A pair of rabbits does not breed untill they are 2 month old. After they are 2 month old, each month they produce a pair. Find the recurrence relation for the number of pairs of rabbits after n months (that is, at the end of the nth month).

Example 2. (The tower of Hanoi)



Example 2. (The tower of Hanoi)



64 disks are placed on the first of three pegs in order of size (as shown in the picture). A disk is allowed to move from one peg to another as long as a disk is never placed on a disk of smaller size. Find the least number of moves required to move all disks to another peg.

Example 3. How many bit strings of length 10 that do not have 2 consecutive 0s?

7.3 Divide-and-Conquer Algorithms and Recurrence Relations

Recall the Merge sort algorithm.

Procedure mergesort $(L = a_1, a_2, \dots, a_n)$
if $n > 1$ **then**
  $m := \lfloor n/2 \rfloor$
  $L_1 = a_1, a_2, \dots, a_m$
  $L_2 = a_{m+1}, a_{m+2}, \dots, a_n$
  $L := merge(mergesort(L_1), mergersort(L_2))$
Print (L)

7.3 Divide-and-Conquer Algorithms and Recurrence Relations

Recall the Merge sort algorithm.

Procedure mergesort $(L = a_1, a_2, \ldots, a_n)$
if $n > 1$ **then**
    $m := \lfloor n/2 \rfloor$
    $L_1 = a_1, a_2, \ldots, a_m$
    $L_2 = a_{m+1}, a_{m+2}, \ldots, a_n$
    $L := merge(mergesort(L_1), mergersort(L_2))$
Print (L)

Let $f(n)$ be the number of comparisons used in the algorithm. Then $f(1) = 1$ and $f(n) = 2f(n/2) + n$.

Let $f(n)$ be a function on the set of integers.

Let $f(n)$ be a function on the set of integers. A divide-and-conquer recurrence relation for f has the form

Let $f(n)$ be a function on the set of integers. A divide-and-conquer recurrence relation for f has the form

$$f(n) = af(n/b) + g(n),$$

Let $f(n)$ be a function on the set of integers. A divide-and-conquer recurrence relation for f has the form

$$f(n) = af(n/b) + g(n),$$

where $g(n)$ is some function and a, b are real numbers.

Let $f(n)$ be a function on the set of integers. A divide-and-conquer recurrence relation for f has the form

$$f(n) = af(n/b) + g(n),$$

where $g(n)$ is some function and a, b are real numbers.

Example. Let $f(n)$ be such that $f(1) = 2$ and $f(n) = f(n/3) + 1$.

Let $f(n)$ be a function on the set of integers. A divide-and-conquer recurrence relation for f has the form

$$f(n) = af(n/b) + g(n),$$

where $g(n)$ is some function and a, b are real numbers.

Example. Let $f(n)$ be such that $f(1) = 2$ and $f(n) = f(n/3) + 1$. Find $f(81)$, $f(3^k)$.

Let $f(n)$ be a function on the set of integers. A divide-and-conquer recurrence relation for f has the form

$$f(n) = af(n/b) + g(n),$$

where $g(n)$ is some function and a, b are real numbers.

Example. Let $f(n)$ be such that $f(1) = 2$ and $f(n) = f(n/3) + 1$. Find $f(81)$, $f(3^k)$.

Master Theorem

Let f be an increasing function that satisfies

$$f(n) = af(n/b) + cn^d,$$

for all $n = b^k$, where k is a positive integer, $a \geq 1$ and $b > 1$ be positive integers, and c, d are positive real numbers. then

$$f(n) = \begin{cases} O(n^d) & \text{if} \quad a < b^d, \\ O(n^d \log n) & \text{if} \quad a = b^d, \\ O(n^{\log_b a}) & \text{if} \quad a > b^d. \end{cases}$$

Fast Multiplication Algorithm

Fast Multiplication Algorithm

The conventional algorithm of multiplying two integers whose binary expansions has length n has complexity $O(n^2)$.

Fast Multiplication Algorithm

The conventional algorithm of multiplying two integers whose binary expansions has length n has complexity $O(n^2)$. The following algorithm has a better complexity.

Fast Multiplication Algorithm

The conventional algorithm of multiplying two integers whose binary expansions has length n has complexity $O(n^2)$. The following algorithm has a better complexity.

Let $a = (a_{2n-1}a_{2n-2}\ldots a_0)_2$ and $b = (b_{2n-1}b_{2n-2}\ldots b_0)_2$.

Fast Multiplication Algorithm

The conventional algorithm of multiplying two integers whose binary expansions has length n has complexity $O(n^2)$. The following algorithm has a better complexity.

Let $a = (a_{2n-1}a_{2n-2}\dots a_0)_2$ and $b = (b_{2n-1}b_{2n-2}\dots b_0)_2$.

Let $a = 2^n A_1 + A_0$ and $b = 2^n B_1 + B_0$. Then

$$ab = (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0.$$

Fast Multiplication Algorithm

The conventional algorithm of multiplying two integers whose binary expansions has length n has complexity $O(n^2)$. The following algorithm has a better complexity.

Let $a = (a_{2n-1}a_{2n-2}\ldots a_0)_2$ and $b = (b_{2n-1}b_{2n-2}\ldots b_0)_2$.

Let $a = 2^n A_1 + A_0$ and $b = 2^n B_1 + B_0$. Then

$$ab = (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0.$$

Let $f(n)$ be the total number of bit operations used in this algorithm for integers of length n then

$$f(2n) = 3f(n) + Cn,$$

where C is a constant.

Fast Multiplication Algorithm

The conventional algorithm of multiplying two integers whose binary expansions has length n has complexity $O(n^2)$. The following algorithm has a better complexity.

Let $a = (a_{2n-1}a_{2n-2}\dots a_0)_2$ and $b = (b_{2n-1}b_{2n-2}\dots b_0)_2$.

Let $a = 2^n A_1 + A_0$ and $b = 2^n B_1 + B_0$. Then

$$ab = (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0.$$

Let $f(n)$ be the total number of bit operations used in this algorithm for integers of length n then

$$f(2n) = 3f(n) + Cn,$$

where C is a constant.

Using the Master Theorem we obtain $f(n) \approx O(n^{1.6})$.