Introduction to Algorithms

Ch.1 Introduction

He Huang
School of Computer Science and Technology
Soochow University

E-mail: huangh@suda.edu.cn

1

与MIT算法导论教材对应关系

教材 Chapter 1 ~ Chapter 3

Chapter 1. 算法在计算中的作用

Chapter 2. 算法基础

Chapter 3. 函数的渐进增长

Chapter 34. NP完全性理论(P/NP/NPC/NP-Hard)

2

Main Topics for this Chapter

- Some Basic Concepts 基本概念
- Asymptotic notations, and analysis 渐进时间表示及分析
- NP完全性理论 (区分并理解P/ NP/ NPC/ NP-Hard 几类问题)

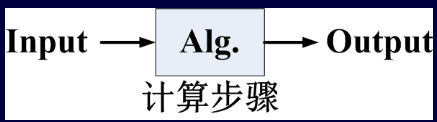
3

Chapter 1. Introduction

4

§ 1.1 算法(Algorithm)

- 非形式定义(Page 3 in the text book)：一个算法是任何一个良定义(well-defined)的计算过程，它接收某个值或值的集合作为输入，产生某个值或值的集合作为输出。因此，一个算法是一个计算步骤的序列，这些步骤将输入转化为输出。



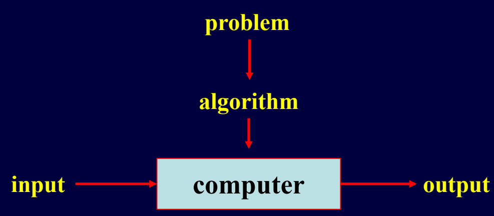
或者说，算法所描述的计算过程就是怎样达到所期望的I/O关系

5

§ 1.1 算法(Algorithm)

Another Def.

An ***algorithm*** is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.



6

例： 排序(Sorting)问题

- 问题描述
  - Input

    具有n个数的数列 $<a_1, a_2, \dots, a_n>$
  - Output

    上述序列的一个排列 $<a_1', a_2', \dots, a_n'>$ 满足关系：
    $a_1' \le a_2' \le \dots \le a_n'$
- 计算步骤
  - 如何达到上述关系

7

§ 1.1 算法(Algorithm)

- Instance (P3 text book)：一个问题的实例由计算该问题的一个解所需要的所有输入所组成。
- 正确性：若对每个输入实例，算法均终止于正确的输出，则称算法是正确的。

不正确的算法 { 对某些输入实例不停机 / 虽然停机，但不是所期望的答案

Note：一个不正确的算法，若其错误的概率是可控制的，有时也是有用的 (Chapter 31 大素数算法，错误率可控算法)。

- 算法的描述：可以用英语说明，可以是程序语言，但是只要能精确描述计算过程即可。

8

Algorithms

Algorithm ?= Program

- Algorithm. (webster.com)
  - -- A well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.

  --Broadly: a step-step procedure for solving a problem or accomplishing some end especially by a computer.

  --issues: correctness, efficiency (amount of work done and space used), storage (simplicity, clarity), optimality .etc.

§ 1.1 算法(Algorithm)——算法与程序的区别

- 算法

算法是指解决问题的一种方法或一个过程，是若干*指令*的*有穷序列*，满足性质：

(1)输入：有外部提供的量作为算法的输入

(2)输出：算法产生至少一个量作为输出。

(3)确定性：组成算法的每条指令是清晰，无歧义的。

(4)有限性：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

正确性是前提

10

§ 1.1 算法(Algorithm)——算法与程序的区别

- 程序

(1)程序是算法用某种程序设计语言的具体实现。

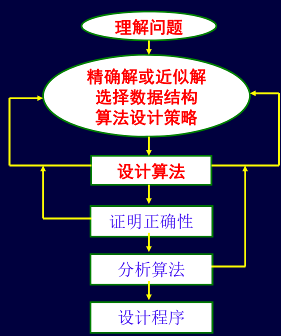
(2)程序可以不满足算法的性质(4)。

例如，操作系统是一个在无限循环中执行的程序，因而不是一个算法。

操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

11

§ 1.1 算法(Algorithm)
——问题求解(Problem Solving)



Some problems (P3-P4)

- Human Genome Project
  - 100,000 genes， sequences of the 3 billion chemical base pairs
- Internet
  - Finding good routes on which the data will travel
  - Search engine
- Electronic commerce
  - Public-key cryptography and digital signatures
- Manufacturing
  - Allocate scarce resources in the most beneficial way

Euclid's Algorithm

Problem: Find gcd(m,n), the greatest common divisor of two nonnegative, not both zero integers m and n

Examples: gcd(60,24) = 12, gcd(60,0) = 60, gcd(0,0) = ?

Euclid's algorithm is based on repeated application of equality

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

until the second number becomes 0, which makes the problem trivial.

Example: gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

14

Two descriptions of Euclid's algorithm

Step 1 If n = 0, return m and stop; otherwise go to Step 2

Step 2 Divide m by n and assign the value for the remainder to r

Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

Euclid(m,n)

while $n \neq 0$ do

  $r \leftarrow m \bmod n$

  $m \leftarrow n$

  $n \leftarrow r$

return m

15

Other methods for computing gcd(m,n)

Consecutive integer checking algorithm

Step 1 Assign the value of min{m,n} to t

Step 2 Divide m by t. If the remainder is 0, go to Step 3; otherwise, go to Step 4

Step 3 Divide n by t. If the remainder is 0, return t and stop; otherwise, go to Step 4

Step 4 Decrease t by 1 and go to Step 2

16

Other methods for gcd(m,n) [cont.]

Middle-school procedure

Step 1 Find the prime factorization of m

Step 2 Find the prime factorization of n

Step 3 Find all the common prime factors

Step 4 Compute the product of all the common prime factors and return it as gcd(m,n)

Is this an algorithm?

17

Sieve of Eratosthenes

Input: Integer $n \geq 2$

Output: List of primes less than or equal to n

for $p \leftarrow 2$ to n do $A[p] \leftarrow p$

for $p \leftarrow 2$ to $\lfloor \sqrt{n} \rfloor$ do

  if $A[p] \neq 0$ //p hasn't been previously eliminated from the list

   $j \leftarrow p * p$

   while $j \leq n$ do

    $A[j] \leftarrow 0$ //mark element as eliminated

    $j \leftarrow j + p$

Example: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

18

Importance of algorithms

- **Problem: sorting 10,000,000 integers**
  - Case 1: Computer A executes one billion instructions per second(1GHz), an algorithm taking time roughly equal to $2n^2$ to sort n integers.
  - Case 2: Computer B executes one hundred million instructions per second(100MHz), an algorithm taking time roughly equal to 50nlgn to sort n integers.
    - Case 1:

$$\frac{2\times(10^7)^2 \text{instructions}}{10^9 \text{instructions / second}} = 200000\text{seconds} \approx 55\text{hours}$$

    - Case 2:

$$\frac{50\times 10^7 \times \log 10^7 \text{instructions}}{10^8 \text{instructions / second}} = 105\text{seconds}$$

Importance of algorithms

Run time (nanoseconds)		$1.3\ N^3$	$10\ N^2$	$47\ N \log_2 N$	$48\ N$
Time to solve a problem of size	1000	1.3 seconds	10 msec	0.4 msec	0.048 msec
	10,000	22 minutes	1 second	6 msec	0.48 msec
	100,000	15 days	1.7 minutes	78 msec	4.8 msec
	million	41 years	2.8 hours	0.94 seconds	48 msec
	10 million	41 millennia	1.7 weeks	11 seconds	0.48 seconds
Max size problem solved in one	second	920	10,000	1 million	21 million
	minute	3,600	77,000	49 million	1.3 billion
	hour	14,000	600,000	2.4 billion	76 billion
	day	41,000	2.9 million	50 billion	1,800 billion
N multiplied by 10, time multiplied by		1,000	100	10+	10

§ 1.2 算法分析

- **伪代码(pseudo code)描述**

  我们课本采用伪代码描述算法（P9 介绍了伪代码描述算法的优势，简洁地表述算法本质，忽略细节)

- **伪代码的一些约定**

  课本P11

21

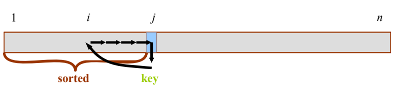
The problem of sorting

- **Input: sequence <a_1,a_2,…,a_n> of *n* natural numbers**
- **Output: permutation <a'_1, a'_2, …, a'_n> such that $a'_1 \le a'_2 \le \ldots \le a'_n$**
- **Example**
  - --Input: <5,2,4,6,1,3>
  - --Output: <1,2,3,4,5,6>

Insertion Sort

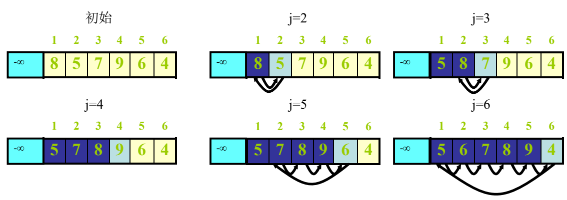
算法导论第二版的伪代码

```
INSERT-SORT (A)
for j ← 2 to length[A]
    do key ← A[j]
        ▷ Insert A[j] into the sorted sequence A[1..j - 1]
        i ← j – 1
        while i > 0 and A[i] > key
            do A[i + 1] ← A[i]   ▷ move item back
               i ← i – 1
        A[i + 1] ← key   ▷ find the insertion position
```



Insertion Sort

```
INSERT-SORT (A)
for j ← 2 to length[A]
    do key ← A[j]
        ▷ Insert A[j] into the sorted sequence A[1..j - 1]
        i ← j – 1
        while i > 0 and A[i] > key
            do A[i + 1] ← A[i]   ▷ move item back
               i ← i – 1
        A[i + 1] ← key   ▷ find the insertion position
```



Kinds of analysis

- **Worst-case: (usually)**

--T(n) = maximum time of algorithm on any input of size n.

- **Average-case: (sometimes)**

--T(n) = expected time of algorithm over all inputs of size n.
--Need assumption of statistics distribution of inputs.

- **Best-case: (bogus假象)**

--Cheat with a slow algorithm that works fast on some input.

Insertion sort analysis

- Worst case: Input reverse sorted

$$T(n)=\sum_{j=2..n}\Theta(j)=\Theta(n^2)$$

- Average case: All permutations equally likely

$$T(n)=\sum_{j=2..n}\Theta(j/2)=\Theta(n^2)$$

- Is insertion sort a fast sorting algorithm?

--Moderately so, for small n.
--Not at all, for large n.

§ 1.2 算法分析

- 目的

分析算法就是估计算法所需资源，选取有效的算法。(时间，空间，通信带宽等)

- 计算模型

单处理机，RAM (Random Access Machine, 随机存取机)模型，其中指令是顺序执行的，无并发操作

- 涉及的知识基础

离散组合数学、概率论、代数等(分析)

程序设计、数据结构(算法设计)

27

§ 1.2 算法分析(续)

- 时间分析

$$\text{算法耗费的时间与}\begin{cases}\text{输入实例的大小}\\ \text{实例的构成}\end{cases}\text{有关}$$

$$\text{例如：插入排序就如同打牌时整理纸牌}\begin{cases}\text{纸牌数目}\\ \text{输入本身就有序}\end{cases}$$

❖Input-Size

通常用整数表示，取决于被研究的问题

例：排序一个数组，Size的自然度量是项数

两数相乘，最好的度量是总的位数

有时，需用两个或多个整数表示输入规模，如图的顶点和边

28

§ 1.2 算法分析(续)

- 时间分析

❖运行时间

①用基本操作的数目(执行步数)来度量；(好处是算法分析独立于机器，即任何基本操作看作是单位时间)

②用更接近实际的计算机上实现的时间来度量；(如RAM模型，不同的指令具有不同的执行时间)

但两者相差一个常数因子。

❖最坏时间

最坏运行时间指Size为n时任何输入的最长运行时间。

29

§ 1.2 算法分析(续)

为何要分析算法的最坏运行时间(P15)?

①它是算法对于任何输入的运行时间的上界；

②对于某些算法，最坏情况常常发生，如在DB中搜索一个并不存在的记录；

③平均时间往往和最坏时间相当(常数因子不同, 也有反例哦！)

❖平均运行时间

常常假定一个给定Size的所有输入是等概率的。实际上这种可能并不一定成立，但可以用随机化算法强迫它成立。有时平均时间和最坏时间不是同数量级，算法选择依据是:

最好、最坏的概率较小时，尽量选择平均时间较小的算法。

30

Analysis algorithms

- We shall assume a generic one-processor, random-access machine (RAM) model of computation.
  - Instructions are executed one after another, with no concurrent operations.
  - Each time, an instruction of a program is executed as an atom operation. An instruction includes arithmetic operations, logical operations, data movement and control operations.
  - Each such instruction takes a constant amount of time.
  - RAM capacity is large enough.
- Under RAM model: count fundamental operations

Analysis of insertion sort

	INSERT-SORT (A)	cost	times
1	for j ← 2 to length[A]	c_1	n
2	do key ← A[j]	c_2	n-1
3	▷ Insert A[j] into the sorted sequence A[1..j - 1]	0	n-1
4	i ← j – 1	c_4	n-1
5	while i > 0 and A[i] > key	c_5	$\sum_{j=2}^{n} t_j$
6	do A[i + 1] ← A[i] ▷ move item back	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	i ← i – 1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	A[i + 1] ← key ▷ find the insertion position	c_8	n-1

t_j : the number of times the while loop test in line 5 is executed for the j value.

Analysis of insertion sort

- To compute $T(n)$, the running time of Insertion-sort, we sum the products of the cost and times columns, obtaining

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1)$$

$$+ c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

- The best-case if the array is already sorted,

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$
$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

--The running time is a linear function of n.

Analysis of insertion sort

- The worst-case results is the array is in reverse sorted order-that is, in decreasing order.

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n(n+1)/2 - 1) + c_6(n(n-1)/2)$$
$$c_7(n(n-1)/2) + c_8(n-1)$$
$$= (c_5/2 + c_6/2 + c_7/2)n^2$$
$$+ (c_1 + c_2 + c_4 + c_5/2 - c_6/2 - c_7/2 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

-- The running time is a quadratic function of n.

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j = n(n+1)/2 - 1$$

$$\sum_{j=2}^{n} (t_j - 1) = \sum_{j=2}^{n} (j-1) = n(n-1)/2$$

§ 1.3 为什么要研究算法

- 软件系统性能取决于算法的效率及快速的硬件，有时高效的算法更重要

35