Lecture 0 Introduction

Lecturer: Han Lin

Something about this course

课程成绩：平时成绩（作业+课堂提问）30% + 期末机考70%

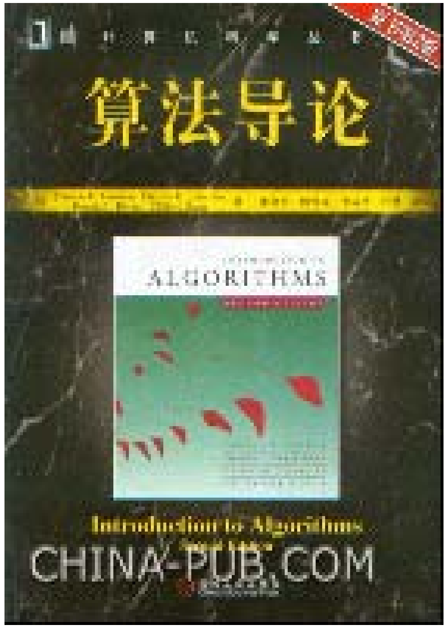
平时作业：https://leetcode.com/problemset/algorithms/

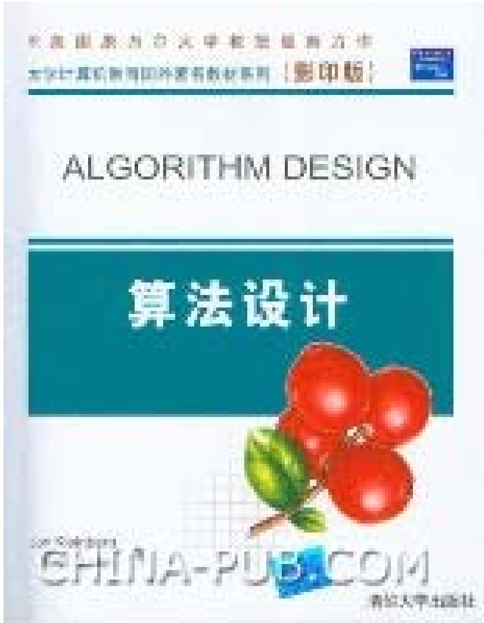
每周在这上面自选题目做，做完把程序和简要题解发到自己的博客上.

博客平台推荐http://blog.csdn.net/

请学委收集每位同学的网址，周末前发给我

推荐书籍

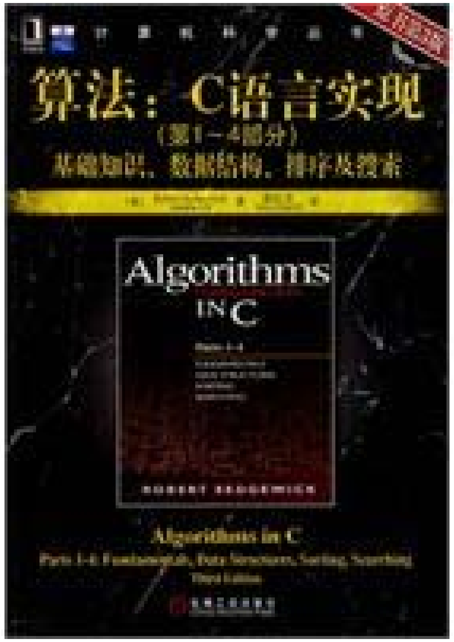

算法导论
ALGORITHMS
Introduction to Algorithms
CHINA-PUB.COM

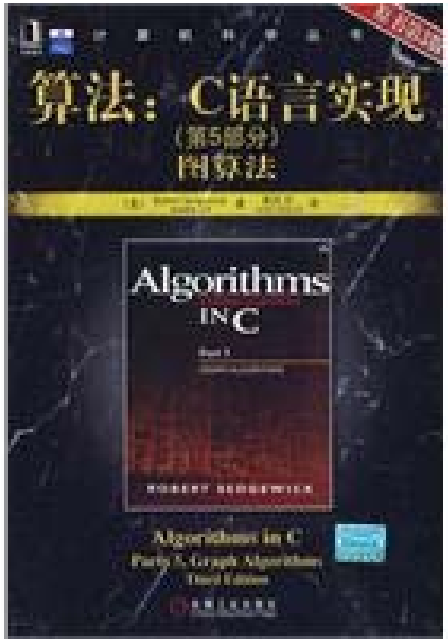

ALGORITHM DESIGN
算法设计
CHINA-PUB.COM


算法概论
Algorithms

教材

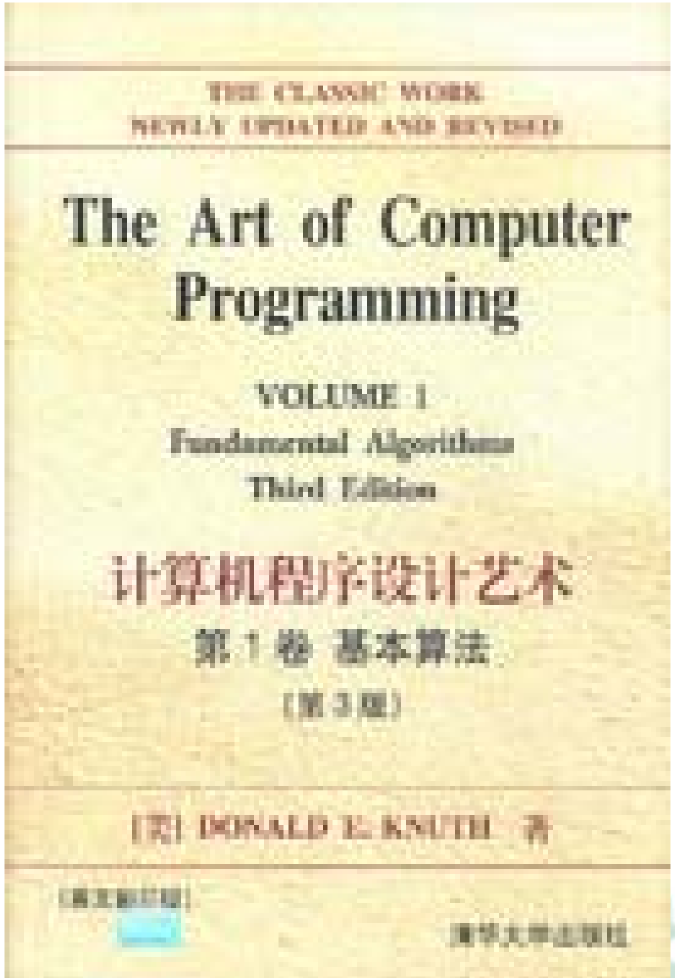
推荐书籍





特点：有较多源代码，适合学习算法的同时提高编程能力

高深秘笈



Donald Knuth
Turing Award (1974)

中山大學

开课第一问

- 什么是算法？

Informally, an *algorithm* is any well-defined computational procedure that takes some value, or set of values, as *input* and produces some value, or set of values, as *output*.

——《算法导论》

Knuth Said…

Knuth (1968, 1973) has given a list of five properties that are widely accepted as requirements for an algorithm:

Finiteness: "An algorithm must always terminate after a finite number of steps ... a *very* finite number, a reasonable number"

Definiteness: "Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case"

Input: "...quantities which are given to it initially before the algorithm begins. These inputs are taken from specified sets of objects"

Output: "...quantities which have a specified relation to the inputs"

Effectiveness: "... all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time by a man using paper and pencil"

Finally …

The word algorithm does not have a generally accepted definition.

From wikipedia

So, next question.

为什么学习算法？

原因之一：广泛的应用

- 网络与通信
- 人工智能
- 金融科技
- 信息安全
- 生物信息学
- 其他工业与商业领域
- 等等

为什么学习算法？

原因之二：强大的力量

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

为什么学习算法？

原因之三：Just for fun!

一个简单的例子：求斐波那契数列第n项的三个算法

斐波那契数列：

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0\,. \end{cases}$$

斐波那契数列是呈指数增长的，事实上，我们有

$$F_n \approx 2^{0.694n}$$

算法一

```
function fib1(n)
if n = 0:  return 0
if n = 1:  return 1
return fib1(n-1) + fib1(n-2)
```

运算次数为指数级！

算法二

```
function fib2(n)
if n = 0 return 0
create an array f[0...n]
f[0] = 0, f[1] = 1
for i = 2...n:
    f[i] = f[i-1] + f[i-2]
return f[n]
```

多项式时间复杂度的算法！

算法三

利用方阵的快速幂：

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}.$$

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

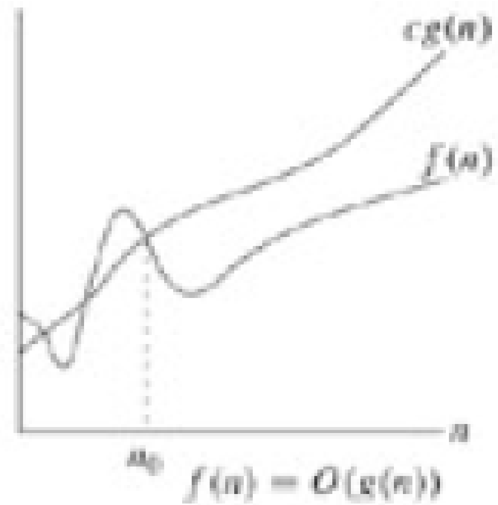
$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}.$$

关键：方阵的n次幂可以只做O(logn)次矩阵乘法

回顾：大O符号

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$.

例如：5 $n^2 \in O(n^2)$，只需取c=5, n0=1.



Just as $O(\cdot)$ is an analog of $\le$, we can also define analogs of $\ge$ and $=$ as follows:

$$f = \Omega(g) \text{ means } g = O(f)$$
$$f = \Theta(g) \text{ means } f = O(g) \text{ and } f = \Omega(g).$$

回顾：小o符号

$o(g(n)) = \{f(n)$: for any positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \leq f(n) < cg(n)$ for all $n \geq n_0\}$.

$f(n) \in o(g(n))$ 等价于

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 .$$

For example, $2n = o(n^2)$, but $2n^2 \neq o(n^2)$.

回顾：小ω符号

$\omega(g(n)) = \{f(n)$: for any positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \le cg(n) < f(n)$ for all $n \ge n_0\}$.

For example, $n^2/2 = \omega(n)$, but $n^2/2 \ne \omega(n^2)$. The relation $f(n) = \omega(g(n))$ implies that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty ,$$

类比

$$f(n) = O(g(n)) \approx a \le b,$$
$$f(n) = \Omega(g(n)) \approx a \ge b,$$
$$f(n) = \Theta(g(n)) \approx a = b,$$
$$f(n) = o(g(n)) \approx a < b,$$
$$f(n) = \omega(g(n)) \approx a > b.$$

We say that $f(n)$ is ***asymptotically smaller*** than $g(n)$ if $f(n) = o(g(n))$, and $f(n)$ is ***asymptotically larger*** than $g(n)$ if $f(n) = \omega(g(n))$.

比较函数的常用技巧

- 多项和中只取“分量”最重的
- 忽略常量因子
- 求极限
- 指数函数的增长快于幂函数，幂函数的增长快于对数函数

练习

下列每组中的两个函数谁“打败”谁？

	$f(n)$	$g(n)$
(a)	$n - 100$	$n - 200$
(b)	$n^{1/2}$	$n^{2/3}$
(c)	$100n + \log n$	$n + (\log n)^2$
(d)	$n \log n$	$10n \log 10n$
(e)	$\log 2n$	$\log 3n$
(f)	$10 \log n$	$\log(n^2)$
(g)	$n^{1.01}$	$n \log^2 n$

Reading

- 课本第0章
- 预习第2章

See you next time!