Tutorial for Middle Term Exam

Hengfeng Wei (魏恒峰)

2011 年 10 月 21 日

1 说明

该lecture note中的很多内容没有来得及讲,抱歉. 希望大家能够按照我们在课堂上讲解的思路,对照着lecture note 中的表格将算法整理出来,通过对比来了解各种算法问题的解结构.

还有,试卷中有些题目会有提示(hint). 需要注意的是,该提示只是说明如果你依照此思路来解题,会比较有保证,但是并不表示必须使用此方法.如果你有更简单的方法,完全可以无视该提示(当然要满足题目所要求的算法复杂度.).

note 中的EX 表示对应的练习题,一般以红色字样标出.

note 中比较重要的知识点亦以红色标出.

祝各位考试顺利.

若有疑问,可与我联系: hengxin0912@gmail.com 或者$qq: 245552163$

更正:

关于Master Theorem case 2 的general 版本,原lecture note上有误. 现已改正(可搜索"更正".). 为清晰起见,保留了原来的行文错误. 课堂上所讲的例题$T(n) = 2T(\frac{n}{2}) + n \log n$ 没有错误.

2 Mathematical Background

2.1 Summation

$\int$, 与Σ

处理Σ的常见技巧:

1. "Perturbation method"求$S_n = \sum_{0 \le k \le n} a_k$:

$$S_n+a_{n+1} = \sum_{0\le k\le n+1} a_k = a_0+\sum_{1\le k\le n+1} a_k = a_0+\sum_{1\le k+1\le n+1} a_{k+1} = a_0+\sum_{0\le k\le n} a_{k+1}.$$

EX: $S_n = \sum_{0\le k\le n} k2^k$

Sol:

$$S_n+(n+1)2^{n+1} = \sum_{0\le k\le n}(k+1)2^{k+1} = \sum_{0\le k\le n} k2^{k+1}+\sum_{0\le k\le n} 2^{k+1} = 2S_n+(2^{n+2}-2)$$

$$S_n = \sum_{0\le k\le n} k2^k = (n-1)2^{n+1}+2.$$

2. "Commutative law (change of variables)"

   EX: $\sum_{1\le i\le j\le k\le n} 1$

3. 还有一个非$\sum$相关的数学题, 即P_{61}1.5. 根据题目的提示,容易求解.

2.2 Mathematical Induction

数学归纳法从有限来推论无穷,其基本原理可概括为: 1) 当$n=1$的时候,该命题成立; 2) 假设当$n=k$时,该命题成立. 那么当$n=k+1$时,该命题也是正确的. 两者缺一不可.

在具体应用数学归纳法时,为保证思路清晰,可采用如下步骤:

Overview: 对. . . 进行归纳.

Basis:

I.H.:

I.Step.: 注意,最好显式地指明应用了归纳假设(I.H.)的地方.

数学归纳法的变种:

1. Basis 可能不止一条($F(n+2)=F(n+1)+F(n)$); 也可能不从1开始($2^n>n^2(n\ge 5)$).

2. 强数学归纳法.

   EX: Proving the correctness of Multiply

数学归纳法的关键点是从$n = k$前进到$n = k + 1$,而其应用场所一般是已知某恒等式,然后去证明. 数学归纳法的思考方式帮助我们适当地"退",退到一个简单的但是又不失重要性不失结构的原始问题,先解决这个简单的问题,然后利用数学归纳法前进. 这也就是通常的递归思想.

EX: (留作思考) "5顶帽子,3红2蓝.三人闭眼,戴帽.睁开眼睛后,让其猜测自己戴的是什么颜色的帽子.三人相互看了看,踌躇一会,然后异口同声地说自己头上戴的是红色的帽子.为什么?"

3 Recursive Relation and Asymptotic Order

3.1 Recursive Relation

常见的递归函数有三类(第一类为重点):

- $T(n) = aT(\frac{n}{b}) + d(n)$
- $a_nT(n) = b_nT_{n-1} + c_n$ (特例: $T_n = 2T_{n-1} + 1$)
- $ax(n) + bx(n-1) + cx(n-2) = f(n)$ (特例: $f(n) = 0$)

对每一类递归式都有三个问题需要回答:

1. 如何求解该类递归式?
2. 有哪些常见的递归式,与之相对应的典型算法有哪些?
3. 递归式对我们设计更高效的算法有何指导意义?

关于$T(n) = aT(\frac{n}{b}) + f(n)$:

1. 心中要有Recursion tree 的样子, 要能在不必画出Recursion tree 的情况下给出total cost 的公式:

$$T(n) = n^{\log_b a} + \sum_{0 \le j \le \log_b n - 1} a^j f(\frac{n}{b^j}).$$

   (a) 第一项, 表示$f(n) = 0$, 是子问题所需时间,假设divide 和combine 的代价为0. $a \uparrow$, more subproblems to solve, exp $\uparrow$;
   $b \uparrow$, smaller subproblem to solve, exp $\downarrow$.

   (b) 第二项, 表示divide and combine 的代价, 受$f(n), a^j, \frac{n}{b^j}$ 影响.

(c) total cost 取决于这两项之间的大小关系,这也是Master Theorem 所要表达的内容.

i. 叶子决定.

$$f(n) = O(n^{\log_b a-\epsilon}) \rightarrow T(n) = \Theta(n^{\log_b a})$$

ii. 平手.

$$f(n) = \Theta(n^{\log_b a}\log^{c+1} n) \rightarrow T(n) = \Theta(n^{\log_b a} \log n)$$

该式有误,现更正为:

$$f(n) = \Theta(n^{\log_b a}\log^{c} n) \rightarrow T(n) = \Theta(n^{\log_b a} \log^{c+1} n), (c \geq 0))$$

iii. divide and combine 决定

$$f(n) = \Omega(n^{\log_b a+\epsilon}) \rightarrow T(n) = \Theta(f(n))$$

(d) Q: 如何improve your alg?

A:

i. 叶子> divide and combine:

Don't Do: finding a faster way to combine subproblems

Should Do: find a way to divide a problem into fewer or smaller subproblems

ii. divide and combine > 叶子:

Should Do: decrease the $f(n)$.

如何求解递归式?

1. Master Theorem

$$T(n) = 7T(\frac{n}{2}) + n^2$$

2. When EX: Master Theorem Fails:

(a) 回归Recursion tree!

$$T(1) = 1; T(n) = 2T(\frac{n}{2}) + n \log n$$

第一项: $O(n)$;

第二项:

$$\sum_{0\leq j\leq k-1} 2^j 2^{k-j} \log(2^{k-j}) = 2^k \sum(k-j) = 2^{k-1}k(k+1).(k = \log n)$$

(b) 变量代换$m = \log n$.

$$T(n) = 2T(\sqrt{n}) + \log n$$

$m = \log n, n = 2^m$

(c) 大胆忽略那些你认为次要的项,套用Master Theorem,给出一个猜测.然后看看能否用数学归纳法给出证明.

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n$$

如果题目中要求证明的话,一般是使用数学归纳法来证明.(讲解数学归纳法)

关于$a_n T(n) = b_n T_{n-1} + c_n$:

1. 求解. “summation factor”

$$S_n = \frac{a_{n-1} \dots a_1}{b_n \dots b_2}$$

EX:

$$T_n = 2T_{n-1} + 1$$

quicksort 递归式求解.

$$T_0 = 0; T_n = n + 1 + \frac{2}{n} \sum_{k=0}^{n-1} T_k$$

Getting rid of the division and $\sum$.

$$nT_n = n^2 + n + 2 \sum_{0 \le k \le n-1} T_k, (n-1)T_{n-1} = (n-1)^2 + (n-1) + 2 \sum_{0 \le k \le n-2} T_k$$

(关键: 用“相减法”处理这里的$\sum$) 两式相减.

$$nT_n = (n+1)C_{n-1} + 2n.$$

关于$ax(n) + bx(n-1) + cx(n-2) = f(n)(or f(n) = 0)$:

1. 如何求解.
特征方程法. (linear second-order recurrences with constant coefficients)

Complexity	Recursion Relation	Typical Algorithm
$O(\log n)$	$T(n) = T(\frac{n}{2}) + O(1)$ or $T(n) = T(\frac{3n}{4}) + O(1)$	binary search
$O(n)$	$T(n) = 2T(\frac{n}{2}) + O(1)$ or $T(n) = T(\frac{n}{2}) + O(n)$ or $T(n) = T(\frac{3n}{4}) + O(n)$ or $T(n) = T(n-1) + O(1)$	parallel addition, selection
$O(n \log n)$	$T(n) = 2T(\frac{n}{2}) + O(n)$	merge sort, quick sort (best case)
$O(n^2)$	$T(n) = 4T(\frac{n}{2}) + O(n)$ or $T(n) = T(n-1) + O(n)$	integer multiplication (simple edition), matrix multiplication (simple edition), insertion sort, quick sort (worst case)
$O(n^{(\log_2 3)})$ or $O(n^{(\log_2 7)})$	$T(n) = 3T(\frac{n}{2}) + O(n)$ or $T(n) = 7T(\frac{n}{2}) + O(n^2)$	integer multiplication, matrix multiplication (advanced edition)
$O(2^n)$	$T(n) = 2T(n-1) + O(1)$	Hanoi tower

Table 1: 常见递归式,算法,复杂度总结

1. $O(\log n)$:

   EX: "Given a sorted array of distinct integers $A[1, \ldots, n]$, you want to find out whether there is an index i for which $A[i] = i$."

   EX(5.18) P_{245}: "E_1, E_2 each with n keys sorted in ascending order. Devise an $O(\log n)$ algorithm to find the nth smallest of the $2n$ distinct keys."

2. $O(n)$:

   EX: "Max Sum Subsequence" (与Array 相关的算法问题)

   关于MSS问题,先简单回顾$O(n^3)$, $O(n^2)$算法,稍为重点讲解$O(n \log n)$算法,重点讲解$O(n)$算法.重点在于算法的设计过程以及recursion relation在其中所起的指导作用.

   在数组上设计$O(n)$算法,可以参考该题的思想. 即假设$A(0, \ldots, i-1)$已经解决,如何来求解$A(0, \ldots, i)$. 得到这两个问题之间的关系之后,就可以设计一个循环版本的算法.而该算法实际上便是递归,其递归式为$T(n) = T(n-1) + O(1)$.在设计$O(n)$算法时, 可以考虑该递归式.

3. $O(n \log n)$: 先想想能不能用到各种经典的排序算法的思想.

   EX: "Bolts and Nuts", "Counting the Number of Inverses"

   再考虑$T(n) = 2T(\frac{n}{2}) + O(n)$.

4. "Aha! Insight!" 属于此类的算法需要一些灵感.

   EX: "Swapping Array Elements" $O(n)$的算法有两种.

   一种是利用$BA = (A^T B^T)^T$. T表示数组逆序.

   一种是利用群论的知识.实际上是个排列问题,而每个排列可以写成多个不相交的轮换之积. 轮换中的元素与循环群相关,而轮换的次数与该子群的陪集相关(为$gcd(n, m)$).

EX: 建议大家对照着渐近阶谱和我们上面分析的不同复杂度所对应的递归式和问题结构将你所接触过的算法整理一下,熟悉该思考方法.

3.2 Asymptotic Order

常用谱:

$$1 \prec \log\log n \prec \log n \prec n^{\epsilon} \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n} \prec c^{c^{n}} (0 < \epsilon < 1 < c)$$

渐近阶的比较方法:

1. 求导法;

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0 (a > 1); \lim_{n\to\infty} \frac{\log^b n}{n^a} = 0 (a > 0).$$

   下面结论比较重要: 任何底大于1的指数函数比任何多项式函数增长得更快; 任何正的多项式函数都比多项对数函数增长得快.

2. 取log法+ 逐级比较法;

   EX: Comparing the Asymptotic Behavior of $f(n) = n^{\log n}$ and $g(n) = (\log n)^n$

   该问题比较重要,多次出现在试卷中

   取log, 要求比较

$$(\log n)^2, n \log\log n$$

   由上面的结论,我们有

$$(\log n)^2 \prec n \log c \prec n \log\log n$$

3. 记住这些结论:

$$\log(n!) = \Theta(n \log n);$$

$$\sum_{i=1}^{n} \frac{1}{i} = \Theta(\ln n)$$

$$(\frac{n}{e})^n \sqrt{2\pi n} < n! < (\frac{n}{e})^n \sqrt{2\pi n}(1 + \frac{1}{11n})$$

4 Sorting

Q: 为什么研究这些经典算法?

A: 至少三点理由;

1. 这些算法是其它算法的基础或者提供预处理步骤.

   EX: 如Anagram.

2. 有些新问题的本质与这些经典问题相同.在解决新问题时,可以想一想它与之前学习的哪一个算法比较相像.

   EX: “Bolts and Nuts”, “Counting the Number of Inverses” (经典算法提供了一种解题的框架)

3. 它们提供了达到相应复杂度的算法样板.

关于Quicksort:

1. 每次pivot都被放在了最终的位置上.

2. pivot的选择. (随机, median!!!) quicksort的partition思想用于求median, 而求取的median 又用来做partition!

3. Quicksort的算法分析.

$$T_0 = 0; T_n = n + 1 + \frac{2}{n} \sum_{k=0}^{n-1} T_k$$

   Getting rid of the division and $\sum$.

$$nT_n = n^2 + n + 2 \sum_{0 \le k \le n-1} T_k, (n-1)T_{n-1} = (n-1)^2 + (n-1) + 2 \sum_{0 \le k \le n-2} T_k$$

   两式相减.

$$nT_n = (n+1)C_{n-1} + 2n.$$

4. Quicksort在n比较小时速度不如其它排序算法.
   EX: ($P_{244}, 5.14$) "sorting 5 elements with 7 comparisons".

5 Selection

讲解selection 算法背后的recursion relation.

EX(5.18) P_{245}: "E_1, E_2 each with n keys sorted in ascending order. Devise an $O(\log n)$ algorithm to find the nth smallest of the $2n$ distinct keys."

6 Searching

重要的数据结构Hashing 与Disjoint Set (Union-Find)

EX(6.19) P_{304}: Hash space complexity

EX(6.23) P_{304}: Sequence of operations (使得形成的树尽可能高即可)

7 Analysis Technology

decision tree; adversary argument; amortized analysis

习题:

1. EX(4.33) P_{213}: decision tree worst case

   该题解答见文件solution4-33.pdf

2. EX: ($P_{244}, 5.14$) " find the median of 5 elements with 6 comparisons"
   EX: ($P_{244}, 5.14$) "sorting 5 elements with 7 comparisons".

   此两题解答见文件sorting and median.pdf

3. EX:Amortized analysis for insertion cost collection of array
   amortized analysis 通常有三种方法:

   (a) 聚集分析(aggregate analysis).
       代价为1的建立新数组操作,每次进行一次;
       代价为2^1的合并数组操作,每2^1进行一次;

...
代价为2^i的合并数组操作,每2^i进行一次; 故总代价为:

$$\sum_{0\le i\le\lfloor\log n\rfloor}\lfloor 2^i\frac{n}{2^i}\rfloor < \sum_{0\le i\le\lfloor\log n\rfloor}\lfloor n\rfloor = n\lg n$$

(b) 记账法(accounting method)
建立新数组的代价$\log n$.

(c) 势函数法(potential method)

amortized analysis 是比较高级的算法分析技巧,需要一定经验.大家需要仔细体会课堂上讲解过的push-pop 和binary count 问题.

amortized analysis 区分理解decision tree, adversary argument, amortized analysis:

1. decision tree 和adversary argument 得到的是lower bound; amortized analysis 得到的是upper bound: it is the average performance of each operation in the worst case

2. average case analysis rely on probabilistic assumptions about the input. 比如,quicksort的average case性能很好,就意味着我们对input的分布做了假设. 在不幸运的情况下,性能会变坏; amortized analysis 得到的是upper bound, 与input无关.

8 For Exam

- 使用数学归纳法时要规范.具体步骤见本lecture note.
- 题目后面括号内的提示仅仅是提示,不是必须,不是评判对错的标准.
- Open mind.write carefully.
- 算法设计题,不要直接给出代码.应以介绍算法的设计原理为主.代码(包括伪码)是次要的,甚至可以完全没有.算法设计后,要给出算法复杂度的简要分析, 比如给出递归式并求解,以证明你设计的算法满足复杂度的要求.