算法基础

主讲人： 庄连生

Email: { *lszhuang@ustc.edu.cn* }

Spring 2018，**USTC**

University of Science and Technology of China

第三讲 递归式

内容提要：

- 代换法
- 迭代法
- 递归树法
- 主方法

- 对算法运行时间的分析，一般都是通过每条指令指定的代价 * 指令执行次数来计算的。比如，插入算法的运行时间分析。

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

INSERTION-SORT(*A*)	cost	times
for(*j* = 2; *j* <=*length*[*A*]; *j*++)	c_1	n
{ *key* = *A*[*j*]	c_2	n-1
// Insert *A*[*j*] into the sorted sequence *A*[1 .. *j*-1]	0	n-1
i = *j*-1	c_4	n-1
while(*i* > 0 && *A*[*i*] > *key*)	c_5	$\sum_{j=2}^{n} t_j$
{ *A*[*i*+1] = *A*[*i*]	c_6	$\sum_{j=2}^{n}(t_j - 1)$
i = *i*-1	c_7	$\sum_{j=2}^{n}(t_j - 1)$
}		
A[*i*+1] = *key*	c_8	n-1
}		

- 当一个算法包含对其自身的递归调用时，其运行时间通常可以用递归式来表示。
- 递归式是一组等式或不等式，它所描述的函数是用在更小的输入下该函数的值来定义的。如，归并排序：

$$T(n)=\begin{cases}1 & \text{, if } n=1\\ 2T(n/2)+n & \text{, if } n>1\end{cases} \qquad (4.1)$$

	cost
MERGE-SORT(*A*, *p*, *r*)	***T*(*n*)**
if *p* < *r*	
Then *q* ←	
MERGE-SORT(*A*, *p*, *q*)	***T*(*n*/2)**
MERGE-SORT(*A*, *q*+1, *r*)	***T*(*n*/2)**
MERGE(*A*, *p*, *q*, *r*)	***n***

- 在表达和求解递归式时一般都会忽略一些技术性细节：

$$T(n)=\begin{cases}1 & \text{, if } n=1\\ 2T(n/2)+n & \text{, if } n>1\end{cases} \qquad (4.1)$$

1）假设函数自变量为整数，忽略上取整和下取整。

$$T(n)=\begin{cases}\Theta(1) & \text{if} n=1\\ T(\lceil n/2\rceil)+T(\lfloor n/2\rfloor)+\Theta(n) & \text{if} n>1\end{cases}$$

2）忽略递归式的边界条件，并假设对于小的n值，T(n)是常量。

$$T(n)=2T(n/2)+\Theta(n)$$

初始值只会影响常数因子，并不会影响函数增长的阶。

递归表达式求解方法：

1）**代换法**：先猜有某个解存在，用数学归纳法证明猜测的正确性；

2）**迭代法**：把递归式转化为求和表达式，然后求和式的界；

3）**递归树法**：直观地表达了迭代法

4）**主方法**：给出了求解 $T(n) = aT(n/b)+f(n)$ 这种形式递归式的简单方法。

第三讲 递归式

内容提要：

- 代换法
- 迭代法
- 递归树法
- 主方法

代换法

代换法求解递归式要点：

1）猜测解的形式

2）用数学归纳法找出使解真正有效的常数

Example: $T(n)=\begin{cases}1 & \text{, if } n=1,\\ 2T(n/2)+n, & \text{if } n>1.\end{cases}$

(1) Guess: $T(n)=n\lg n+n$

(2) Induction

Basic: $n=1 \Rightarrow n\lg n+n=1=T(n)$

Inductive step: Inductive hypothesis is that $T(k)=k\lg k+k$ for all $k<n$.

Use the inductive hypothesis of $T(n/2)$ to prove $T(n)$

$$T(n) = 2T(n/2)+n$$

$$= 2((n/2)\lg(n/2)+(n/2))+n \quad \text{(by inductive hypothesis)}$$

$$= n\lg(n/2)+n+n \ = \ n(\lg n-\lg 2)+2n = n\lg n+n.$$

代换法

- 代换法求解递归式要点：

1）猜测解的形式

2）用数学归纳法找出使解真正有效的常数

Example: $T(n)=\begin{cases}1, & \text{if } n=1,\\ 2T(n/2)+n, & \text{if } n>1.\end{cases}$

(1) Guess: $T(n)=n\lg n+n$

(2) Induction

Basic: $n=1\Rightarrow n\lg n+n=1=T(n)$

Inductive step: Inductive hypothesis is that $T(k)=k\lg k+k$ for all $k<n$.

Use the inductive hypothesis of $T(n/2)$ to prove $T(n)$.

这个方法非常有效，但是只适合于求解一些比较容易猜测的情形！

代换法

- The substitution method can be used to establish either upper (O) or lower bounds (Ω) on a recurrence.
- example, determining an upper bound on the recurrence

$$T(n) = 2T(\lfloor n/2 \rfloor) + n, \qquad (4.4)$$

(1) Guessing that the solution is T(n) = O(n lg n).

(2) Proving T(n)≤cn lg n for a some constant c > 0.

- Assume that this bound holds for n/2 , that is, that $T(\lfloor n/2 \rfloor) \le c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$. Substituting into the recurrence yields

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \le 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n$$

$$\le cn\lg(n/2) + n = cn\lg n - cn\lg 2 + n = cn\lg n - cn + n \le cn\lg n,$$

where the last step holds as long as c ≥ 1.

代换法

- **Mathematical induction now requires us to show that our solution holds for the boundary conditions.**
- **Typically, the boundary conditions are suitable as base cases for the inductive proof.**

$$T(n) = 2T(\lfloor n/2 \rfloor) + n, \qquad (4.4)$$

$$T(n) = O(n \lg n) \quad , \ T(n) \le c\, n \lg n$$

- **This requirement can sometimes lead to problems.**
- **Assume that $T(1) = 1$ is the sole boundary condition of the recurrence. Then, we can't choose c large enough, since $T(1) \le c\ 1\ \lg 1 = 0$, which is at odds with $T(1)=1$. The case of our inductive proof fails to hold.**（递归结果与初始情况矛盾，即递归证明失败？）

代换法

$$T(n)=2T(\lfloor n/2 \rfloor)+n\ ;\quad T(n)=O(n\lg n)\quad ,\ T(n)\le c\ n\lg n$$

- **An inductive hypothesis inconsistent with specific boundary condition, How to overcome the difficulty?**
  （如何克服递归结果与边界条件不一致的问题？）
  - **asymptotic notation only requires us to prove $T(n)\le cn\lg n$ for $n\ge n_0$, where n0 is a constant.**
  - **to remove the difficult boundary condition $T(1) = 1$**
  - **Impose $T(2)$ and $T(3)$ as boundary conditions for the inductive proof.**
  - **From the recurrence, we derive $T(2) = 4$ and $T(3) = 5$.**
  - **The inductive proof that $T(n)\le cn\lg n$ can now be completed by choosing any $c\ge 2$ so that $T(2)\le c2\lg 2$ and $T(3)\le c\,3\lg 3$.**

代换法

做一个好的猜测：

- **Unfortunately, there is no general way to guess the correct solutions to recurrences.（猜想不是一种方法）**
- **Guessing a solution takes experience and, occasionally, creativity.**

(why we study the course? It’s a training for us to get experience, to catch occasion, to have creativity.)

- **Fortunately, though, there are some heuristics (recusion tress) that can help you become a good guesser.**

代换法

- **If a recurrence is similar to one you have seen before, then guessing a similar solution is reasonable. For example,**

$$T(n) = 2T(\lfloor n/2 \rfloor + 17) + n,$$

  - **which looks difficult because of the added “17”.**
  - **Intuitively, this additional term cannot substantially affect the solution to the recurrence.**（该附加项不会从本质上影响递归解）
  - **When *n* is large, the difference between *T*(*n*/2) and *T*(*n*/2 + 17) is not that large. Consequently, we make the guess that *T(n) = O(n lg n)*, which you can verify as correct by using the substitution method.**

一些细微的问题:

- **Sometimes, guess correctly, but somehow the math doesn't seem to work out in the induction.**

For example $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1.$

- **Guess the solution is $O(n)$, then try to show that $T(n) \le cn$ for an appropriate constant c. Substituting .. , then**

$$T(n) \le c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 = cn + 1,$$

which does not imply $T(n) \le cn$ for any choice of c.

代换法

一些细微的问题:

- Sometimes, guess correctly, but somehow the math doesn't seem to work out in the induction.

  For example $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1.$

  guess $T(n) = cn$, thne $T(n) \le c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 = cn + 1,$

  contradiction.

- Usually, it is that the inductive assumption isn't strong enough to prove the detailed bound. How to overcome?

  （递归假设条件不强）

- Revising the guess by subtracting a lower-order term often permits the math to go through. （减去低阶项）

代换法

一些细微的问题:

$$T(n) \le T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \qquad \text{Solution: } T(n) = O(n)$$

- **try a larger guess $T(n) = O(n2)$, which can work.**
- **But the guess that the solution is $T(n) = O(n)$ is correct.**
- **Intuitively, our guess is nearly right: we're only off by the constant 1, a lower-order term.**
- **Nevertheless, mathematical induction doesn't work!**
- **Subtracting a lower-order term from our previous guess. New guess is $T(n) \le cn - b$, where $b \ge 0$ is constant, then**

$$T(n) \le (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1 = cn - 2b + 1 \le cn - b \ ,$$

as long as $b \ge 1$. As before, the constant c must be chosen large enough to handle the boundary conditions.

代换法

一些细微的问题:

- Most people find the idea of subtracting a lower-order term counterintuitive. （违反直觉）
- After all, if the math doesn't work out, shouldn't we be increasing our guess?
- The key to understand this step is to remember that we are using mathematical induction: we can prove something stronger for a given value by assuming something stronger for smaller values. （假设更强的条件，可证明更强的结论）

避免陷阱：

- **It is easy to err in the use of asymptotic notation.**

  For example, in the recurrence (4.4)

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \qquad (4.4)$$

we can falsely prove $T(n) = O(n)$ by guessing $T(n) \le cn$ and then arguing

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \le 2(c\lfloor n/2 \rfloor) + n \\ &\le cn + n \\ &= O(n)\ , \qquad \Leftarrow wrong\,!!! \end{aligned}$$

since c is a constant. The error is that we haven't proved the exact form of the inductive hypothesis, that is, that $T(n) \le cn$.

代换法

改变变量:

- **algebraic manipulation（代数变换）: sometimes solute an unknown recurrence similar to one you have seen before.**

  Example, $T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n,$

  which looks difficult. Simplify the recurrence with a change of variables. For convenience, we shall not worry about rounding off values, such as $\sqrt{n}$, to be integers.

  Let $m = \lg n$, then $T(2^m) = 2T(2^{m/2})+m$.

  Thus rename $S(m)=T(2^m)$ => $S(m)=2S(m/2)+m$,

  which is very much like recurrence (4.4) and has the same solution: $S(m)=O(m \lg m)$. Changing back from $S(m)$ to $T(n)$, we obtain $T(n)=T(2^m)=S(m)=O(m \lg m)=O(\lg n \lg\lg n)$.

代换法

代换法求解步骤小结：

(1) 做一个好的猜测

(2) 证明一般情况成立

(3) 处理边界条件，可以进行边界扩展

注意事项：

(1) 对更小的值做更强的假设

(2) 避免陷阱

(3) 适当时进行变量替换

第三讲 递归式

内容提要：

- 代换法
- 迭代法
- 递归树法
- 主方法

迭代法

- **Substitution: It is difficult to come up with a good guess**
- **The iteration method**
  - **doesn't require us to guess the answer**
  - **may require more algebra**（迭代法对代数能力的要求较高）
  - **to expand (iterate) the recurrence and express it as a summation of terms, and the initial conditions**
  - **to evaluate summations.**（不断迭代展开为级数，并求和）

For example, $T(n) = 3T(\lfloor n/4 \rfloor) + n$

$$T(n) = n + 3T(\lfloor n/4 \rfloor)$$

$$= n + 3\ (\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor))$$

$$= n + 3(\lfloor n/4 \rfloor + 3(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor)))$$

$$= n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27T(\lfloor n/64 \rfloor),$$

迭代法

$$T(n) = n + 3T(\lfloor n/4 \rfloor) = n + 3\ (\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor))$$

$$= n + 3(\lfloor n/4 \rfloor + 3(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor)))$$

$$= n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27T(\lfloor n/64 \rfloor),$$

- **How far must we iterate the recurrence?**
  - **The *i*th term in the series is** $3^i \lfloor n/4^i \rfloor$.
  - **The iteration halts when** $\lfloor n/4^i \rfloor = 1$. **By continuing the iteration until this point and using the bound** $\lfloor n/4^i \rfloor \le n/4^i$ **, we get a decreasing geometric series:**

$$T(n) \le n + 3n/4 + 9n/16 + 27n/64 + \cdots + 3^i T(n/4^i)$$

$$\le n\sum_{i-0}^{\infty}(3/4)^i + \Theta(n^{\log_4 3}) = 4n + o(n) = O(n).$$

$$(n/4^i = 1 \Rightarrow i = \log_4 n \Rightarrow 3^i = 3^{\log_4 n} = n^{\log_4 3})$$

迭代法

- The iteration method usually leads to lots of algebra. It can be a challenge. The key points:
  - the number of times the recurrence needs to be iterated to reach the boundary condition,（递归次数）
  - and the sum of the terms arising from each level of the iteration process.（级数求和）

- Sometimes, in the process of iterating a recurrence, you can guess the solution without working out all the math. Then, the iteration can be abandoned in favor of the substitution method, which usually requires less algebra.

（在展开递归式为迭代求和的过程中，有时只需要部分展开，然后根据其规律来猜想递归式的解，接着用代换法进行证明。）

迭代法

- **When a recurrence contains floor and ceiling functions, the math can become especially complicated.**

- **Often, it helps to assume that the recurrence is defined only on exact powers of a number.**

Example, $T(n) = 3T(\lfloor n/4 \rfloor) + n$

if we had assumed that $n = 4^k$ for some integer k, the floor functions could have been conveniently omitted.

第三讲 递归式

内容提要：

- 代换法
- 迭代法
- 递归树法
- 主方法

递归树法

- **Drawing out a recursion tree, is a straightforward way to devise a good guess, and to show the iteration method intuitively.**（画递归树可以从直观上表示迭代法，也有助于猜想递归式的解）
- **Recursion trees are particularly useful when the recurrence describes the running time of a divide-and-conquer algorithm.**
- 递归树中，每一个节点都代表着递归函数调用集合中一个子问题的代价。将树中每一层内的代价相加得到一个每层代价的集合，再将每层的代价相加得到递归式所有层次的总代价。

递归树法

- 使用递归树产生好的猜测时，通常需要容忍小量的“不良量”：
  - ✓ 1）floor, ceiling忽略；
  - ✓ 2）n经常假设为某个整数的幂次方；
- 例子见课本P41
- 关键：1）树的深度如何确定？

  2）每个节点的代价—>每层的代价—>总代价

第三讲 递归式

内容提要：

- 代换法
- 迭代法
- 递归树法
- 主方法

主方法

- solving recurrences of the form

$$T(n) = aT(n/b)+f(n), \qquad (4.5)$$

  where $a\geq 1$ and $b>1$, and $f(n)$ is asymptotically positive.

- The master method requires memorization of three cases, but then the solution of many recurrences can be determined quite easily, often without pencil and paper.
- n/b might not be an integer. Replacing each of $T(n/b)$ with either $T(\lfloor n/b \rfloor)$ or $T(\lceil n/b \rceil)$ doesn't affect the asymptotic behavior .
- Normally, it is convenient to omit the floor and ceiling functions when writing divide-and-conquer recurrences of this form.

主方法

- **Theorem 4.1 （主定理）**

  Let $a \geq 1$ and $b>1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

  $$T(n) = aT(n/b) + f(n),$$

  where we interpret n/b to mean either $\lfloor n/b \rfloor$ *or* $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows.

1. If $f(n) = O(n^{(\log_b a)-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{(\log_b a)+\varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

主方法

$$T(n) = aT(n/b) + f(n),$$

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right), & f(n) = O\left(n^{(\log_b a)-\varepsilon}\right) \\ \Theta\left(n^{\log_b a}\lg n\right), & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta(f(n)), & f(n) = \Omega\left(n^{(\log_b a)+\varepsilon}\right) \text{ and } af(n/b) \le cf(n) \text{ for large } n \end{cases} \quad \exists \varepsilon > 0, c < 1$$

- **Comparing the function $f(n)$ with $n^{\log_b a}$. Intuitively, the solution is determined by the larger of the two functions.**
  - **Case 1, $n^{\log_b a}$ larger, then the solution is $T(n) = \Theta(n^{\log_b a})$.**
  - **Case 3, $f(n)$ larger, then the solution is $T(n) = \Theta(f(n))$.**
  - **Case 2, the two functions are the same size, we multiply by a logarithmic factor, and the solution is**

$$T(n) = \Theta(n^{\log_b a}\lg n) = \Theta(f(n)\lg n) \quad .$$

主方法

$$T(n) = aT(n/b) + f(n),$$

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right), & f(n) = O\left(n^{(\log_b a)-\varepsilon}\right) \\ \Theta\left(n^{\log_b a}\lg n\right), & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right), & f(n) = \Omega\left(n^{(\log_b a)+\varepsilon}\right) \text{ and } af(n/b) \le cf(n) \text{ for large } n \end{cases} \quad \exists \varepsilon > 0, c < 1$$

主方法特殊情况：

- **Polynomially**
  - **Case 1, $f(n)$ must be polynomially smaller than $n^{\log_b a}$.**
  - **Case 3, $f(n)$ must be polynomially larger than $n^{\log_b a}$.**
- **Gap** Example: $f(n) = n\lg n$, $n^{\log_b a} = n$
  - **There is a gap between cases 1 and 2 when $f(n)$ is smaller than $n^{\log_b a}$ but not polynomially smaller.**
  - **Similarly, there is a gap between cases 2 and 3 when $f(n)$ is larger than $n^{\log_b a}$ but not polynomially larger.**

主方法

$$T(n) = aT(n/b) + f(n),$$

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right), & f(n) = O\left(n^{(\log_b a)-\varepsilon}\right) \\ \Theta\left(n^{\log_b a} \lg n\right), & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right), & f(n) = \Omega\left(n^{(\log_b a)+\varepsilon}\right) \text{ and } af(n/b) \le cf(n) \text{ for large } n \end{cases} \quad \left.\right\} \exists \varepsilon > 0, c < 1$$

举例:

- **$T(n)=9T(n/3)+n$**

$$a = 9, b = 3, f(n) = n \quad \Rightarrow \quad n^{\log_b a} = n^{\log_3 9} = n^2 = \Theta(n^2)$$

$$\Rightarrow \quad f(n) = O(n^{\log_3 9-\varepsilon}), \text{where } \varepsilon = 1 \quad \Rightarrow \quad T(n) = \Theta(n^2)$$

- **$T(n)=T(2n/3)+1$**

$$a = 1, b = 3/2, f(n) = 1 \quad \Rightarrow \quad n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$$

$$\Rightarrow \quad f(n) = \Theta(n^{\log_3 a}) = \Theta(1) \quad \Rightarrow \quad T(n) = \Theta(\lg n)$$

主方法

$$T(n) = aT(n/b) + f(n),$$

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right), & f(n) = O\left(n^{(\log_b a)-\varepsilon}\right) \\ \Theta\left(n^{\log_b a} \lg n\right), & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right), & f(n) = \Omega\left(n^{(\log_b a)+\varepsilon}\right) \text{ and } af(n/b) \le cf(n) \text{ for large } n \end{cases} \Bigg\} \exists \varepsilon > 0, c < 1$$

举例： *T(n)=3T(n/4)+nlgn*

$$a = 3, b = 4, f(n) = n \lg n \quad \Rightarrow \quad n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$$

$$\Rightarrow \quad f(n) = \Omega(n^{(\log_4 3)+\varepsilon}), \text{where } \varepsilon \approx 0.2, \text{ and for sufficiently large } n,$$

$$af(n/b) = 3(n/4)\lg(n/4) \le (3/4)n\lg n = cf(n) \text{ for } c = 3/4$$

$$\Rightarrow \quad T(n) = \Theta(n \lg n)$$

主方法

$$T(n) = aT(n/b) + f(n),$$

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right), & f(n) = O\left(n^{(\log_b a)-\varepsilon}\right) \\ \Theta\left(n^{\log_b a}\lg n\right), & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta(f(n)), & f(n) = \Omega\left(n^{(\log_b a)+\varepsilon}\right) \text{ and } af(n/b) \le cf(n) \text{ for large } n \end{cases} \quad \begin{matrix} \varepsilon > 0 \\ c < 1 \end{matrix}$$

- ***T*(*n*)=2*T*(*n*/2)+*n*lg*n***

$$a = 2, b = 2, f(n) = n\lg n \quad \Rightarrow \quad n^{\log_b a} = n,$$

but $f(n)/n = \lg n$, which is asymptotically less than n^{ε} for any positive constant c, that is $f(n)$ is not polynomially larger than n. Consequently, the recurrence falls into the gap between case 2 and case 3.

谢谢！

Q & A