4 Recurrences

- **Difficulty of Algorithms Research**
  - **Model 建模**
  - **Specify 描述**
  - **Correctness 正确性**
    - **Verify 验证**
    - **Proof 证明**
  - **Complex 复杂度 （Efficiency 有效性）**
    - **Actual computing 实际可计算性**

Design 设计

Correctness Analysis 正确性分析

Computing Analysis 可计算性分析

- **Recurrence is a basic method to analyze algorithm**

Algorithms analysis

- **sum**

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

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
}		

4 Recurrences

Algorithms analysis

- **recursion**

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

		cost
MERGE-SORT(*A*, *p*, *r*)		***T*(*n*)**
1	**if *p* < *r***	
2	$q \leftarrow \lfloor (p+r)/2 \rfloor$	
3	**MERGE-SORT(*A*, *p*, *q*)**	***T*(*n*/2)**
4	**MERGE-SORT(*A*, *q*+1, *r*)**	***T*(*n*/2)**
5	**MERGE(*A*, *p*, *q*, *r*)**	***n***

4 Recurrences

Algorithms analysis

- **recursion**

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

```
f(n)
{
   if(n<=2)
      return 1;
   else
      return f(n-1)+f(n-2);
}
```

4 Recurrences

Algorithms analysis

- **recursion**

h(n) = h(0)*h(n-1) + h(1)*h(n-2) + ... + h(n-1)h(0)

$$h(n) = h(0)*h(n-1) + h(1)*h(n-2) + \dots + h(n-1)h(0)$$

(其中 n>=2) 另类递归式:

$$h(n) = ((4*n-2)/(n+1))*h(n-1)$$

该递推关系的解为:

$$h(n) = C(2n, n)/(n+1)$$

(n = 1, 2, 3,...)

E Zexal的二叉树（签到）

时间限制：1000ms 内存限制：65536kb

通过率：200/209 (95.69%) 正确率：200/596 (33.56%)

题目

知识点：树，数论，dp，递归（都可以做）

上学期我们学习了二叉树，也都知道3个结点的二叉树有5种， 现给你二叉树的结点个数n，要你输出不同形态二叉树的种数。

输入

第一个数为一个整数$n(n <= 30)$

输出

对于每组数据，输出一行，不同形态二叉树的种数。

输入样例

```
3
```

输出样例

```
5
```

4 Recurrences

A recurrence is an equation or inequality in terms of

- **one or more base cases, and**
- **itself, with smaller arguments.**

Examples:

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

Solution: $T(n)=n.$

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

Solution: $T(n)=n\lg n+n.$

(3) $$T(n)=\begin{cases}0 & \text{, if } n=2,\\ T(\sqrt{n})+1 & \text{, if } n>2.\end{cases}$$

Solution: $T(n)=\lg\lg n.$

(4) $$T(n)=\begin{cases}1 & \text{, if } n=1,\\ T(n/3)+T(2n/3)+n, & \text{if } n>1.\end{cases}$$

Solution: $T(n)=\Theta(n\lg n)$

4 Recurrences

How to obtain asymptotic "Θ" or "*O*" bounds on the recurrence solution?

- **Substitution method（置换法）: guesses a bound and then use mathematical induction to prove our guess correct.**
- **Iteration method（迭代法）: converts the recurrence into a summation and then relies on techniques for bounding summations to solve the recurrence.**
- **Recursion-tree method (a kind of iteration method)**
- **Master method（主方法，母函数法）: provides bounds for recurrences of the form $T(n) = aT(n/b)+f(n)$, where $a \geq 1$, $b > 1$, and $f(n)$ is a given function.**

Technicalities

In practice, we neglect certain technical details when we state and solve recurrences.（忽略技术细节）

1) **Assumption of integer arguments to functions**

- **Normally, $T(n)$ is only defined when n is an integer**
- **Example, the worst-case running time of MERGE-SORT**

$$T(n) = \begin{cases} \Theta(1) & \text{, if } n=1, \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{, if } n>1, \end{cases} \qquad (4.2)$$

Technicalities

2) Ignore **boundary conditions**

- Omit statements of the boundary conditions of recurrences, assume that **$T(n)$ is constant for small n**, that is **$T(n) = \Theta(1)$ for sufficiently small n**.

  （n 较小时，$T(n)$为常数）

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

- Example, state recurrence (4.1) as

  $T(n) = 2T(n/2) + \Theta(n)$, (Omit $T(1)=\Theta(1)$) (4.3)

  without explicitly giving values for small n.

  - The reason is that although changing the value of $T(1)$ changes the solution to the recurrence, the **order of growth is unchanged**.（改变边界值，可能改变递归式的解，但不改变解的函数增长率）

Technicalities

neglect certain technical details

1) Assumption of integer arguments to functions
2) Ignore boundary conditions
3) **Omit floors, ceilings**

- **These details don’t affect the asymptotic bounds of many recurrences encountered in the analysis of algorithms** （忽略细节不会影响算法的渐近分析）

4.1 The substitution method

Key points

1) **guessing the form of the solution;**

2) **using mathematical induction to show the solution works.**

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 $T(n)$

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&= 2((n/2)\lg(n/2) + (n/2)) + n \quad \text{(by inductive hypothesis)} \\
&= n\lg(n/2) + n + n \ = \ n(\lg n - \lg 2) + 2n = n \lg n + n.
\end{aligned}
$$

4.1 The substitution method

Key points

1) guessing the form of the solution;
2) using mathematical induction to show the solution works.

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

(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 $T(n)$.

- **The method is powerful, but it can be applied only in cases when it is easy to guess the form of the answer.**

4.1 The substitution method

- **The substitution method is powerful, but it can be applied only in cases when it is easy to guess the form of the answer.**

$$h(n) = h(0)*h(n-1) + h(1)*h(n-2) + \ldots + h(n-1)h(0)$$

(其中 n>=2) 另类递归式:

$$h(n) = ((4*n-2)/(n+1))*h(n-1)$$

$$h(n) = C(2n, n)/(n+1)$$

$$(n = 1, 2, 3, \ldots)$$

E Zexal的二叉树（签到）

时间限制：1000ms 内存限制：65536kb

通过率：200/209 (95.69%) 正确率：200/596 (33.56%)

题目

知识点：树，数论，dp，递归（都可以做）

上学期我们学习了二叉树，也都知道3个结点的二叉树有5种，现给你二叉树的结点个数n，要你输出不同形态二叉树的种数。

输入

第一个数为一个整数$n(n <= 30)$

输出

对于每组数据，输出一行，不同形态二叉树的种数。

输入样例

```
3
```

输出样例

```
5
```

4.1 The substitution method

- **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**

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

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

(2) Proving $T(n) \le cn \lg n$ 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 $c \ge 1$.

4.1 The substitution method

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

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

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

- **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？）**

(1)To extend boundary conditions

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

- **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 n_0 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**
    $T(2) = 2*T(1) + 2 = 4$, $T(3) = 2*T(1) + 3 = 5$.
  - **The inductive proof that $T(n) \le cn \lg n$ can now be completed by choosing any $c \ge 2$ so that**
    $T(2) = 4 \le c*2 \lg 2$ and $T(3) = 5 \le c\ 3 \lg 3$.

4.1.1 Making a good guess

- **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 (recursion tress) that can help you become a good guesser.**

4.1.1 Making a good guess

- **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.**

4.1.1 Making a good guess

- **Another way to make a good guess is to prove loose upper and lower bounds on the recurrence and then reduce the range of uncertainty.**

  （寻找松的上、下渐近界，范围缩小，逐步逼近）

  For example

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

  - **we might start with a lower bound of $T(n) = \Omega(n)$, since we have the term n in the recurrence,**
  - **and we can prove an initial upper bound $T(n) = O(n^2)$.**
  - **Then, we can gradually lower the upper bound and raise the lower bound until we converge on the correct, asymptotically tight solution of $T(n)=\Theta(n\lg n)$.**

4.1.2 Subtleties

- **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**

$$\begin{aligned} T(n) &= T(n/2) + T(n/2) + 1 \\ &\le c \cdot n/2 + c \cdot n/2 + 1 = cn + 1, \end{aligned}$$

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

4.1.2 Subtleties

- **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)

4.1.2 Subtleties

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

- **try a larger guess $T(n) = O(n^2)$, 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!**
- **(2)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.

4.1.2 Subtleties

- **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）**

4.1.3 (3)Avoiding pitfalls (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*)≤*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*)≤*cn*.

4.1.4 Changing variables

- **algebraic manipulation: 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 $T(n)=T(2^m)=S(m)=O(m \lg m)=O(\lg n \lg\lg n)$.

4.1 The substitution method: Notes

(1) To extend boundary conditions

(2) Subtracting a lower-order term

(3) Avoiding pitfalls

4.2 The iteration method

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

$$
\begin{aligned}
T(n) &= n + 3T(\lfloor n/4 \rfloor) \\
&= n + 3\ (\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor)) \\
&= n + 3(\lfloor n/4 \rfloor + 3(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor))) \\
&= n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27T(\lfloor n/64 \rfloor),
\end{aligned}
$$

4.2 The iteration method

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

- **How far must we iterate the recurrence?**
  - **The *i*th term in the series is** $3^i T(\lfloor n/4^i \rfloor)$ **.**
  - **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:**

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

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

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

4.2 The iteration method

- **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,**（迭代次数）
  - **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，接着用置换法进行证明。）

4.2 The iteration method

- **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.

* 4.3 The recursion-tree method

- **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.**

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