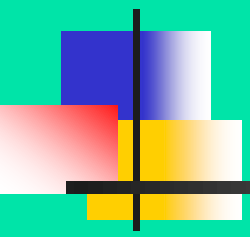


# 算法设计与分析

主讲：凌应标　博士/副教授

*issLYB@mail.sysu.edu.cn*

# 教材与参考书

- 陈慧南等编著，《算法设计与分析》，电子工业出版社，2012年7月
- 郑宗汉等编著，《算法设计与分析》，清华大学出版社，2004年11月
- 王晓东编著，《计算机算法设计与分析》，电子工业出版社，2001年1月
- Michael Sipser著，《计算理论导引》，机械工业出版社，1996年10月

# 课程要求

- 实验作业：50%
- 考试：50%

# 教学内容与目标

- 算法设计方法（**重点**）
  - 常规方法:递归与分治、动态规则、贪心策略、回溯法、分枝界限法（**熟练掌握和运用**）
  - 高级方法:概率算法、近似算法和自然算法，如遗传算法、模拟退火算法、禁忌算法、粒群算法与蚁群算法等（**补充**）
- 算法分析方法
  - 确定性算法的性能分析：递归方程与递归不等式求解（**基本掌握和运用**）
  - 随机算法的性能分析（**初步掌握**）
- 算法理论 （**了解**）
  - 计算模型
  - NP完全性

# 第1章　算法概述

内容纲要

- 算法与程序
- 算法复杂性表示
- 算法复杂性分析
- 递归方程求解方法

# 1.1 算法概念

- 算法概念形成的历史

  源于希尔伯特(公理化几何、数理逻辑 )提出的判定问题，并在其后尝试定义有效计算性或者有效方法中成形。

  - 哥德尔　1930年提出的递归函数
  - 海布兰德　1934年提出的递归函数
  - 克莱尼　　1935年提出的递归函数
  - 邱奇　　1936年提出的λ演算
  - 波斯特　　1936年的Formulation 1
  - 图灵　　1937年提出的图灵机

# 算法（algorithm）定义

- 全国科学技术名词审定委员会审定公布
  - 中文名称：算法
  - 英文名称：**algorithm**
  - 定义：模型分析的一组可行的、确定的和有穷的规则。
- 算法的定义
  - 是指根据计算机可完成的指令编写的解题方案的准确而完整的描述，是解决问题的一系列清晰指令，算法代表着用系统的方法描述解决问题的策略机制。
  - 可以理解为由基本运算及规定的运算顺序所构成的完整的解题步骤。或者看成按照要求设计好的有限的确切的计算序列，并且这样的步骤和序列可以解决一类问题。
- 程序
  - 实现一定功能的计算机语言的指令序列.可以是算法用某中程序设计语言的具体实现.但程序不一定满足算法的条件.

# 算法的性质：五个重要的特征

算法可以使用**自然语言、伪代码、流程图**等多种不同的方法来描述。

1、**输入**：一个算法有0个或多个输入，以刻画运算对象的初始情况，所谓0个输入是指算法本身定出了初始条件；

2、**输出**：一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的；

3、**确切性**：算法的每一步骤必须有确切的定义；

4、**有穷性**：算法的有穷性是指算法必须能在执行有限个步骤之后终止；

5、**可行性**：算法中执行的任何计算步骤都是可以被分解为基本的可执行的操作步，即每个计算步都可以在有限时间内完成（也称之为有效性）。

# 关于基本的可执行的操作步

- 原则上来说，算法可以是人执行或计算机执行。
- 本课程中，算法的执行者一般指计算机。因此，执行的基本操作是计算机能执行的所有指令的集合，即计算机系统的指令系统。所以，基本可执行的操作步包括：
  - 基本运算和操作：**算术运算、逻辑运算、关系运算和数据传输**
  - 控制结构：分支、循环和子程序调用
- 这一概念也是很难描述，可计算理论课程另有介绍。

# 例1:求最大公约数问题与算法

- 记两个整数m和n(0<=m<n)的最大公约数为GCD(n,m)

[算法1-1] 展转相除法求GCD(n,m)

输入:两个整数m和n(0<=m<n)

输出: GCD(n,m)

```
int GCD(n,m) {
  if (m==0) return n;
  while(m>0) {
        c=n mod m;
        n=m;
        m=c;
   }
   return n;
}
```

# 例1:续1

[算法1-2] 递归描述的展转相除法求GCD(n,m)

输入:两个整数m和n(0<=m<n)

输出: GCD(n,m)

```
int rGCD(n,m) {
  if (m==0) return n;
  return rGCD(m,(n mod m));
}
```

# 例1:续2

[算法1-3] 穷举法求GCD(n,m)

输入:两个整数m和n(0<=m<n)

输出: GCD(n,m)

```
int bGCD(n,m) {
    if (m==0) return n;
    t=m;
   while(m%t||n%t) t--;
   return t;
  }
```

# 例1:续3

- 一个简单的问题可以设计出多种不同的算法
  - 欧几里德算法
  - 连续检测法
- 不同的算法差别很大
  - 算法思想上
  - 描述形式上
- 算法怎么设计
- 算法怎么评价
- ……

# 1.1 为什么要学习算法

- **算法**是计算机科学的基础,更是程序的基石.具有良好的算法基础才能成为训练有素的软件人才
  - 必须知道来自不同计算领域的重要算法
  - 必须学会设计新算法,确认其正确性和分析其效率
- **设计算法**也是各个应用领域和技术人员使用计算机求解他们各自专业领域的问题所必备的能力
- **算法**是计算机科学知识训练的工具----D.E.Knuth
- **算法**是计算机科学的核心----David Harel

# 1.2 算法复杂性分析

- **算法复杂性**是运行算法程序所需要的计算机资源的定性或定量度量.
- **计算机资源**
  - CPU时间(时间复杂性)
  - 存储器空间(空间复杂性)
- **记号**
  - 问题规模N、问题输入I和算法A
  - 时间复杂性T=T(N,I,A),简写T=T(N,I)
  - 空间复杂性S=S(N,I,A) ,简写S=S(N,I)

# 复杂性的数学表达

- 抽象计算机
  - 指令: $k$种,$O_1, O_2, \ldots, O_k$
  - 指令执行时间:$t_1, t_2, \ldots, t_k$
- 时间复杂性$T=T(N,I,A)$ ,简记为$T=T(N,I)$
  - 对给定的$N$和$I$,算法$A$执行指令$O_i$的次数

    $e_i$， $i=1, 2, \ldots, k_k$
  - $T=T(N,I)$表达式

$$T(N,I)=\sum_{i=1}^{k} t_i e_i(N,I)$$

# 常用的三种时间复杂性

- 最坏时间复杂性

$$T_{\max}(N,I)=\max_{I\in D_n}T(N,I)=\sum_{i=1}^{k}t_i e_i(N,I^*)$$

- 最好时间复杂性

$$T_{\min}(N,I)=\min_{I\in D_n}T(N,I)=\sum_{i=1}^{k}t_i e_i(N,\tilde{I})$$

- 平均时间复杂性

$$T_{avg}(N,I)=\sum_{I\in D_N}P(I)_i(N,I)$$

# 复杂性的简化

- **复杂性的渐近表达式**

设时间复杂性$T(N)$,如果存在$\tilde{T}(N)$,有

$$\lim_{N\to\infty}\frac{(T(N)-\tilde{T}(N))}{T(N)}\to 0$$

则称$\tilde{T}(N)$是$T(N)$的**渐近表达式**

- **例**

设$T(N)=3N^2+4N\log N+7$和$\tilde{T}(N)=3N^2$,则

$$\lim_{N\to\infty}\frac{(T(N)-\tilde{T}(N))}{T(N)}=\lim_{N\to\infty}\frac{4N\log N+7}{3N^2+4N\log N+7}=0$$

所以,称$3N^2$是$3N^2+4N\log N+7$的**渐近表达式**

# 上界

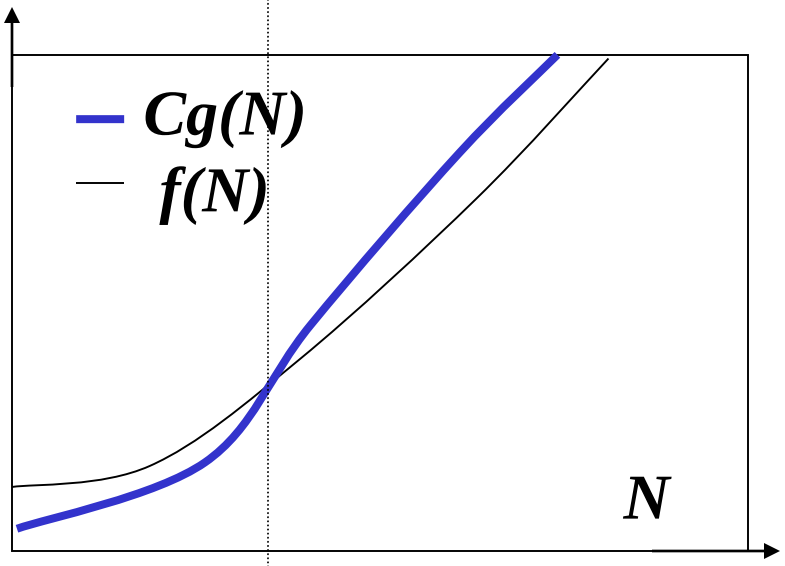
- **上界: *f(N)=O(g(N))***

  如果存在正的常数C和自然数$N_0$ ,使得当$N \geq N_0$时有

  $$f(N) \leq Cg(N)$$

  则称 ***f(N)*** 有上界,且***g(N)***是它的一个上界, 记为

  $$f(N)=O(g(N))$$




- **例**

  1. $3N = O(N)$
  2. $104N^2+568N+456 = O(N^2)$
  3. $8993N^2 = O(N^3)$
  4. 反例: $3N^3 \neq O(N^2)$

- **上界越低则评估越准确,结果就越有价值.**

# 下界

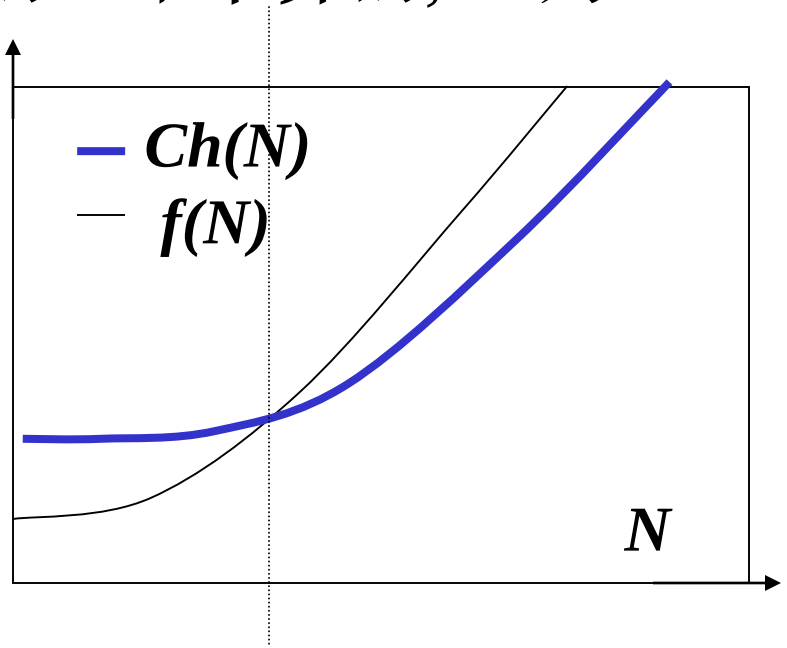
- 下界: $f(N)= \Omega(h(N))$

如果存在正的常数C和自然数$N_0$ ,使得当$N \geq N_0$时有

$$f(N) \geq Ch(N)$$

则称 $f(N)$ 有下界,且$h(N)$是它的一个下界的, 记为

$$f(N)=\Omega(h(N))$$




- 例

1. $3N= \Omega(1)$
2. $4N^2= \Omega(N^2)= \Omega(N)$
3. 反例: $3N^2 \neq \Omega(N^3)$

- 下界越高,则评估越准确

# 同阶与低阶

- $f(N)=\theta(h(N))$

  如果 $f(N)=O(h(N))$且$f(N)=\Omega(h(N))$

- $f(N)=o(h(N))$

  如果对任意给定的正常数$\varepsilon$ ,都存在正整数$N_0$ ,使得当$N\geq N_0$时有

  $f(N)/h(N)<\varepsilon$ 即 $\lim_{N\to\infty} f(N)/h(N)=0$

  则称 $f(N)$比$h(N)$低阶, 记为

  $f(N)=o(h(N))$

# 算法分析方法

- 例：顺序搜索算法

```
template<class Type>
int seqSearch(Type *a, int n, Type k)
{
    for(int i=0;i<n;i++)
        if (a[i]==k) return i;
    return -1;
}
```

- （1）$T_{max}(n) = \max\{ T(I) \mid size(I)=n \} = O(n)$
- （2）$T_{min}(n) = \min\{ T(I) \mid size(I)=n \} = O(1)$
- （3）在平均情况下，假设：
- (a) 搜索成功的概率为$p$（$0 \le p \le 1$）；
- (b) 在数组的每个位置$i$（$0 \le i < n$）搜索成功的概率相同，均为$p/n$。

$$T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$$

$$= \left(1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + 3 \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}\right) + n \cdot \left(1 - p\right)$$

$$= \frac{p}{n}\sum_{i=1}^{n} i + n\left(1 - p\right) = \frac{p(n+1)}{2} + n(1-p)$$

# 算法分析的基本法则

- 非递归算法：
- （1）for / while 循环
- 循环体内计算时间*循环次数；
- （2）嵌套循环
- 循环体内计算时间*所有循环次数；
- （3）顺序语句
- 各语句计算时间相加；
- （4）if-else语句
- if语句计算时间和else语句计算时间的较大者。

```
template<class Type>
void insertion_sort(Type *a, int n)
{
    Type key;                                  //  cost      times
    for (int i = 1; i < n; i++){               //   c1        n
        key=a[i];                              //   c2        n-1
        int j=i-1;                             //   c3        n-1
        while( j>=0 && a[j]>key ){             //   c4        sum of ti
          a[j+1]=a[j];                         //   c5        sum of (ti-1)
          j--;                                 //   c6        sum og (ti-1)
          }
       a[j+1]=key;                             //   c7        n-1
       }
}
```

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{i=1}^{n-1}t_i+c_5\sum_{i=1}^{n-1}(t_i-1)+c_6\sum_{i=1}^{n-1}(t_i-1)+c_7(n-1)$$

- 在最好情况下，$t_i=1$, for $1 \le i < n$;

$$\begin{aligned}T_{\min}(n)&=c_1n+c_2(n-1)+c_3(n-1)+c_4(n-1)+c_7(n-1)\\&=(c_1+c_2+c_3+c_4+c_7)n-(c_2+c_3+c_4+c_7)=O(n)\end{aligned}$$

- 在最坏情况下，$t_i \le i+1$, for $1 \le i < n$;

$$\sum_{i=1}^{n-1}(i+1)=\frac{n(n+1)}{2}-1 \qquad \sum_{i=1}^{n-1}i=\frac{n(n-1)}{2}$$

$$\begin{aligned}T_{\max}(n)&\le c_1n+c_2(n-1)+c_3(n-1)+\\&c_4\left(\frac{n(n+1)}{2}-1\right)+c_5\left(\frac{n(n-1)}{2}\right)+c_6\left(\frac{n(n-1)}{2}\right)+c_7(n-1)\\&=\frac{c_4+c_5+c_6}{2}n^2+\left(c_1+c_2+c_3+\frac{c_4-c_5-c_6}{2}+c_7\right)n-(c_2+c_3+c_4+c_7)\\&=O(n^2)\end{aligned}$$

- 对于输入数据$a[i]=n-i, i=0,1,\ldots,n-1$，算法insertion_sort 达到其最坏情形。因此，

$$T_{\max}(n) \geq \frac{c_4+c_5+c_6}{2}n^2 + \left(c_1+c_2+c_3+\frac{c_4-c_5-c_6}{2}+c_7\right)n-(c_2+c_3+c_4+c_7)$$

$$=\Omega(n^2)$$

- 由此可见，$T_{\max}(n)=\Theta(n^2)$

# 最优算法

- 问题的计算时间下界为$\Omega(f(n))$，则计算时间复杂性为$O(f(n))$的算法是最优算法。
- 例如，排序问题的计算时间下界为$\Omega(n\log n)$，计算时间复杂性为$O(n\log n)$的排序算法是最优算法。
- 堆排序算法是最优算法。

# 递归算法复杂性分析

```
int factorial(int n)
{
    if (n == 0) return 1;
    return n*factorial(n-1);
 }
```

$$T(n)=\begin{cases}0 & n=0\\ T(n-1)+1 & n>0\end{cases}$$

$$T(n)=T(n-1)+1$$

$$T(n-1)=T(n-2)+1$$

……

$$T(2)=T(2-1)+1$$

$$T(1)=1$$

所以

$$T(n)=n$$

# 递归方程求解方法

- 递推法(迭代法)
  - 递推常系数递归方程
  - 递推变系数递归方程
- 替换法(猜测加归纳证明)
- 换元
- 生成函数法
- 特征方程法
  - K阶常系数齐次递归方程
  - K阶常系数非齐次递归方程
- 通用递归方程法
- 积分法

# 递推法1-简单递推

扩展递推式,将其转换为一个和式,然后计算该和式.

- 例 1: T(n)=2T(n-1)+1

$$T(n) = 2T(n-1)+1$$

$$= 2(2T(n-2)+1)+1 = 2^2 T(n-2)+2+1$$

$$= \ldots\ldots$$

$$= 2^{n-1}T(1)+2^{n-2}+\ldots+2^1+2^0$$

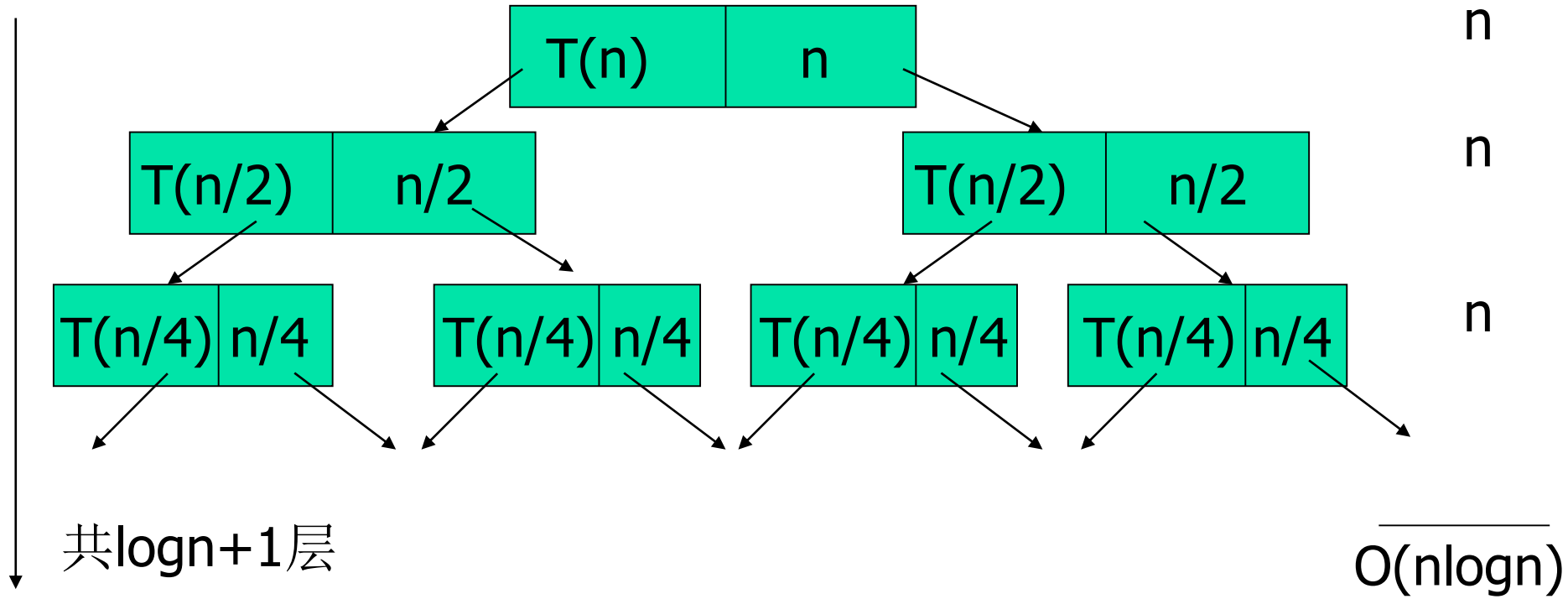
$$= 2^{n-1}+2^{n-2}+\ldots+2^1+2^0 = 2^n-1$$

练习1:扩展法求T(n)=3T(n-1)+1的解释式,T(0)=1

# 递推法1-简单递推

将递推式表示为一棵递归树,然后计算所有结点上表示的数之和.

- 例 2:T(n)=2T(n/2)+n




练习2: 用递归树求 T(n)=T(n/2)+T(2n/3)+n的解释式, T(0)=1

# 递推法2-递推求解变系数递归方程

扩展递推式,将其转换为一个较复杂的和式,然后计算该和式

- 例1：T(n)=2T(n-1)+n,T(0)=1

$$T(n) = 2T(n-1) + n$$

$$= 2(2T(n-2) + n - 1)) + n = 2^2 T(n-2) + 2(n-1) + n$$

$$= \ldots\ldots$$

$$= 2^n T(0) + 2^{n-1} \times 2 + \ldots + 2(n-1) + n$$

$$= 2^n \times 1 + 2^{n-1} \times 2 + \ldots + 2^1 \times (n-1) + 2^0 \times n$$

$$= 2^{n+1} - n - 2 \text{(请同学推导一下)}$$

练习3: 扩展法求T(n)=g(x)T(n-1)+h(n)的解释式, T(0)=1

# 替换法

试扩展几个n比较小的递推式求值,发现规律,然后猜测并归纳证明.

- 例：T(n)=2T(n-1)+1,T(0)=1

$$T(3) = 7 = 2^3 - 1$$

$$T(4) = 15 = 2^4 - 1$$

$$T(5) = 31 = 2^5 - 1$$

……

$$T(n) = 2^n - 1 \text{(猜测)}$$

请同学们用归纳法证明结论.

# 换元法

- 对函数的定义域进行转换,并在新的定义域里,定义一个新的递归方程
- 把问题转换为对新的递归方程的求解.
- 然后再把所得的解转换回原方程的解.

# 换元法-例

- 例：T(n)=2T(n-1)+n/2-1,T(1)=1,其中$n=2^k$

把n表示成k的关系，原递归方程改写为:

$$\begin{cases} T(2^k) = 2T(2^{k-1}) + n/2 - 1 \\ T(2^0) = 1 \end{cases}$$

再令$g(k)=T(2^k)=T(n)$,原递归方程可改写为:

$$\begin{cases} g(k) = 2g(k-1) + 2^{k-1} - 1 \\ g(0) = 1 \end{cases}$$

容易得到

$$g(k) = \frac{1}{2} \times 2^k \times k + 1 = \frac{1}{2} n \log n + 1$$

# 生成函数法

- 把递归函数h(n)的各个取值构造函数G(x):
  $G(x)=h(1)x+h(2)x^2+h(3)x^{3+\cdots}=\sum_{k=1}^{\infty}h(k)x^k$
- **为了求出h(n),对G(x)进行演算,求出其解析表达式**
- 再对G(x)的解析表达式转换为对应的幂级数
- 对照幂级数的系数,获得h(n)的解析表达式

# 生成函数法-例1

- 递归函数h(n):T(n)=2T(n-1)+1,T(1)=1
- 构造生成函数G(x):

  $G(x)=h(1)x+h(2)x^2+h(3)x^{3+\cdots} = \sum_{k=1}^{\infty} h(k)x^k$

- **演算**

$$G(x)-2xG(x)=$$

$$h(1)x+h(2)x^2+h(3)x^3+\ldots$$

$$-2h(1)x^2-2h(2)x^3-2h(3)x^4-\ldots$$

$$=h(1)+(h(2)-2h(1))x+(h(3)-2h(2))x^2+(h(3)-2h(2))x^3+\ldots$$

$$=x+x^2+x^3+\ldots$$

$$=\frac{x}{1-x}$$

# 生成函数法-例1-续1

$$(1-2x)G(x)=\frac{x}{1-x}$$

$$G(x)=\frac{x}{(1-x)(1-2x)}$$

$$令G(x)=\frac{A}{1-x}-\frac{B}{1-2x}=\frac{A-2Ax+B-Bx}{(1-x)(1-2x)}$$

有:

$$\begin{cases}A+B=0\\-2A-B=1\end{cases}$$

得:$A=-1, B=1$

于是:

$$G(x)=\frac{1}{1-2x}-\frac{1}{1-x}$$

$$=(1+2x+2^2x^2+2^3x^3+\ldots)-(1+x+x^2+x^3+\ldots)$$

$$=(2-1)x+(2^2-1)x^2+(2^3-1)x^3+\ldots$$

$$=\sum_{k=1}^{\infty}(2^k-1)x^k$$

# 生成函数法-例1-续2

- 对照可以看:$h(n)=2^n-1$
- 关键在演算这一步
  - 根据递推式化简
  - 分解:待定系数法
  - 再展开

练习4:生成函数法求T(n)=T(n-1)+T(n-2)的解释式,
T(1)=T(2)=1

# 特征方程法

- 特征方程法
  - K阶常系数齐次递归方程
  - K阶常系数非齐次递归方程
- 例:

$$\begin{cases} f(n) = a_1 f(n-1) + a_2 f(n-2) + \ldots + a_k f(n-k) \\ f(i) = b_i \quad 0 \le i < k \end{cases}$$

# K阶常系数齐次递归方程

- 递归方程形式

$$\begin{cases} f(n) = a_1 f(n-1) + a_2 f(n-2) + \ldots + a_k f(n-k) \\ f(i) = b_i \quad 0 \le i < k \end{cases}$$

- 特征方程及其根

$$x^k - a_1 x^{k-1} - a_2 x^{k-2} - \ldots - a_k = 0$$

其根:$q_1, q_2, \ldots, q_k$

- 构造递归方程解
  - 情况一:无重根

$$f(n) = c_1 q_1^n + c_2 q_2^n + \ldots + c_k q_k^n$$

  - 情况二:有重根, $q_1, \ldots, q_{i-1}, (q_i = q_{i+1} = \ldots = q_{i+r-1}), q_{i+r}, \ldots, q_k$

$$f(n) = c_1 q_1^n + \ldots + c_{i-1} q_{i-1}^n + (c_i + c_{i+1} n + \ldots + c_{i+r-1} n^{r-1}) q_i^n + \ldots + c_k q_k^n$$

# K阶常系数齐次递归方程-例

- 求解三阶常系数齐次递归方程形式

$$\begin{cases} f(n)=6f(n-1)-11f(n-2)+6f(n-3) \\ f(0)=0 \\ f(1)=2 \\ f(2)=10 \end{cases}$$

- 特征方程及其根

$$x^3-6x^2+11x-6=0$$

改写为: $x^3-3x^2-3x^2+9x+2x-6=0$

再进行因式分解,得:

(x-1)(x-2)(x-3)=0

则有特征根: q1=1,q2=2,q3=3

# K阶常系数齐次递归方程-例-续

- 构造递归方程解
  - 属于情况一:无重根,递归方程通解为

$$f(n) = c_1 q_1^n + c_2 q_2^n + c_3 q_3^n$$

$$= c_1 + c_2 2^n + c_3 3^n$$

再由初始条件得到:

$$\begin{cases} c_1 + c_2 + c_3 = 0 \\ c_1 + 2c_2 + 3c_3 = 2 \\ c_1 + 4c_2 + 9c_3 = 10 \end{cases}$$

解此联立方程,得:

$c_1 = 0,\ c_2 = -2,\ c_3 = 2$

于是递归方程的解为:

$$f(n) = 2(3^n - 2^n)$$

# K阶常系数非齐次递归方程

- 递归方程形式

$$\begin{cases} f(n) = a_1 f(n-1) + a_2 f(n-2) + \ldots + a_k f(n-k) + g(x) \\ f(i) = b_i \quad 0 \le i < k \end{cases}$$

- 递归方程通解形式

$$f(n) = \overline{f(n)} + f^*(n)$$

其中,$\overline{f(n)}$是对应齐次方程的通解,而

$f^*(n)$是原非次方程的特解.

- 寻找**特解**没有有效的办法,有几种特殊情况可以求特解.

# K阶常系数非齐次递归方程-关于特解

- 情况一:$g(x)$是n的m次多项式,即:

$$g(x) = b_1 n^m + b_2 n^{m-1} + \ldots + b_m n + b_{m+1}$$

其特解$f^*(n)$也是n的m次多项式:

$$f^*(x) = A_1 n^m + A_2 n^{m-1} + \ldots + A_m n + A_{m+1}$$

- 情况二:$g(x)$是形式如下的指数函数:

$$g(x) = (b_1 n^m + b_2 n^{m-1} + \ldots + b_m n + b_{m+1}) a^n$$

如果a不是特征方程的重根,特解$f^*(n)$为:

$$f^*(x) = (A_1 n^m + A_2 n^{m-1} + \ldots + A_m n + A_{m+1}) a^n$$

如果a是特征方程的r重根,特解$f^*(n)$为:

$$f^*(x) = (A_1 n^m + A_2 n^{m-1} + \ldots + A_r n + A_{r+1}) n^r a^n$$

# K阶常系数非齐次递归方程-例

- 求解如下递归方程形式

$$\begin{cases} f(n) = 7f(n-1) - 10f(n-2) + 4n^2 \\ f(0) = 1 \\ f(1) = 2 \end{cases}$$

- 对应的齐次递归方程的特征方程为:

$$x^2 - 7x + 10 = 0$$

得到特征根为:q1=2,q2=5

对应齐次递归方程通解为: $\overline{f(n)} = c_1 2^n + c_2 5^n$

令该非齐次递归方程的特解为: $f^*(n) = A_1 n^2 + A_2 n + A_3$

代入原方程,得:

$$A_1 n^2 + A_2 n + A_3 - 7(A_1(n-1)^2 + A_2(n-1) + A_3))$$

$$+10(A_1(n-2)^2 + A_2(n-2) + A_3) = 4n^2$$

# K阶常系数非齐次递归方程-例-续

代入原方程,得:

$$A_1n^2+A_2n+A_3-7(A_1(n-1)^2+A_2(n-1)+A_3))$$

$$+10(A_1(n-2)^2+A_2(n-2)+A_3)=4n^2$$

化简,得:

$$4A_1n^2+(-26A_1+4A_2)n+33A_1-13A_2+4A_3)=4n^2$$

得联立方程:

$$\begin{cases}4A_1=4\\-26A_1+4A_2=0\\33A_1-13A_2+4A_3=0\end{cases}$$

求得:

$$A_1=1,A_2=\frac{13}{2},A_3=\frac{103}{8}$$

# K阶常系数非齐次递归方程-例-续

所以原方程的通解为:

$$f(n) = c_1 2^n + c_2 5^n + n^2 + \frac{13}{2}n + \frac{103}{8}$$

再代入初始条件,得联立方程:

$$\begin{cases} f(0) = c1 + c2 + \dfrac{103}{8} = 0 \\ f(1) = 2c1 + 5c2 + \dfrac{163}{8} = 2 \end{cases}$$

求得: $c_1 = \frac{41}{3}, c_2 = \frac{43}{24}$

最后得原方程的通解为:

$$f(n) = -\frac{41}{3}2^n + \frac{43}{24}5^n + n^2 + \frac{13}{2}n + \frac{103}{8}$$

# 通用递归方程法

- 常见形式如下的递归方程

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

其中$a>=1$和$b>1$是常数，$f(n)$是一渐近正函数，$n/b$指$\lceil n/b \rceil$或$\lfloor n/b \rfloor$

- 几种情况的通解

（1）若对于某个正常数$\varepsilon$,有 $f(n) = O(n^{\log_b a - \varepsilon})$ ,则 $T(n) = \theta(n^{\log_b a})$

（2）若 $f(n) = \theta(n^{\log_b a})$ ,则 $T(n) = \Theta(n^{\log_b a} \log n)$

（3）若对于某个正常数$\varepsilon$，有 $f(n) = \theta(n^{\log_b a + \varepsilon})$ ,且对某个正常数$c<1$和所有足够大的$n$,有$af(n/b)<=cf(n)$,则$T(n) = \Theta(f(n))$

# 通用递归方程法-例

- 例1 $T(n)=16T(n/4)+n$

  这里$a$=16和$b=4$是常数，$f(n)=n=O(n^{\log_b a-\varepsilon})=O(n^{2-\varepsilon},)$其中$\varepsilon=1$,
  符合第一种情况，所以

$$T(n)=\Theta(n^{\log_b a})=\Theta(n^2)$$

- 例2 $T(n)=T(3n/7)+1$

  这里$a$=1和$b=7/3$是常数，且

$$n^{\log_b a}=n^{\log_{7/3}1}=n^0=1,\ f(n)=1=\Theta(n^{\log_b a})$$

  符合第二种情况，所以 $T(n)=\Theta(n^{\log_b a}\log n)=\Theta(\log n)$

- 例3 $T(n)=3T(n/4)+n\log n$

  这里$a$=3,$b=4$，$n^{\log_b a}=n^{\log_4 3}=O(n^{0.793}),\ f(n)=n\log n=\Omega(n^{\log_4 3+\varepsilon})$
  其中$\varepsilon$约为0.2，且对正常数$c$=3/4,由于对足够大的$n$,有
  3($n$/4)log(n/4)<=(3/4)$n$log$n$,符合第三种情况，则

$$T(n)=\Theta(f(n))=\Theta(n\log n)$$

# 课后作业

(一)硬件厂商XYZ公司宣称他们最新研制的微处理器运行速度为其竞争对手ABC同类产品的10倍。对于计算复杂性分别为$n,n^2,n^3$和$n!$的各类算法，若用XYZ公司的计算机在1小时内能解输入规模为n的问题，那么用ABC算法公司的计算机在1小时内分别能解输入规模为多大的问题？

(二)证明：如果$g(N)=O(f(N))$，则 $O(f)+O(g)=O(f)$

(三)证明： $n!=\theta(n^n)$

(四)用k阶常系数齐次递归方程的方法求解菲波那奇序列的递归方程.