第二章 递归与分治

2.1 算法基本工具

- 2.1.1 算法与数据结构
- 2.1.2 优化算法的数学模型
- 2.1.3 循环

2.1.1 算法与数据结构

I. 原始信息与处理结果的对应存储
II. 数组使信息有序化
III. 数组记录状态信息
IV. 大整数存储及运算

I. 原始信息与处理结果对应存储

每一个问题中的信息往往是多方面的，在算法中一般有输入信息、输出信息和信息加工处理过程中的中间信息。如何确定用数组进行信息存储，数组元素下标与信息如何对应等问题，很大程度上影响着算法的编写效率和运行效率。

下面的例子恰当地选择了用数组存储的信息，并把题目中的有关信息作为下标使用，使算法的实现过程大大简化。

【例2.1.1】某次选举，要从五个候选人（编号分别为1、2、3、4、5）中选一名厂长。请编程完成统计选票的工作。

- 算法设计：
- 1）虽然选票发放的数量一般是已知的，但收回的数量通常是无法预知的，所以算法采用随机循环，设计停止标志为“-1”。
- 2）统计过程的一般方法为：先为五个候选人各自设置五个“计数器”S1，S2，S3，S4，S5，然后根据录入数据通过多分支语句或嵌套条件语句决定为某个“计数器”累加1，这样做效率太低。
- 现在把五个“计数器”用数组代替，让选票中候选人的编号xp做下标，执行A[xp]=A[xp]+1就可方便地将选票内容累加到相应的“计数器”中。也就是说数组结构是存储统计结果的，而其下标正是原始信息。
- 3）考虑到算法的健壮性，要排除对1-5之外的数据进行统计。

【例2.1.2】编程统计身高（单位为厘米）。统计分150-154；155-159；160-164；165-169；170-174；175-179及低于150、高于179共八档次进行。

- 算法设计：
- 输入的身高可能在50-250之间，若用输入的身高数据直接作为数组下标进行统计，即使是用PASCAL语言可设置上、下标的下界，也要开辟200多个空间，而统计是分八个档次进行的，这样是完全没有必要的。
- 由于多数统计区间的大小是都固定为5，这样用“身高/5-29”做下标，则只需开辟8个元素的数组，对应八个统计档次，即可完成统计工作。

【例2.1.3】一次考试共考了语文、代数和外语三科。某小组共有九人，考后各科及格名单如下表，请编写算法找出三科全及格的学生的名单（学号）。

科目	及格学生学号
语文	1，9，6，8，4，3，7
代数	5，2，9，1，3，7
外语	8，1，6，7，3，5，4，9

方 法 一：

- 从语文及格名单中逐一抽出各及格学生学号，先在代数及格名单核对，若有该学号（说明代数也及格了），再在外语及格名单中继续查找，看该学号是否也在外语及格名单中。若仍在，说明该号属全及格学生的学号，否则就是至少有一科不及格的。若语文及格名单中就没有某生的号，显然该生根本不在比较之列，自然不属全及格学生 。
- 方法采用了枚举尝试的方法
- A，B，C三组分别存放语文、代数、外语及格名单，尝试范围为三重循环：
- I循环， 初值0， 终值6， 步长1
- J循环， 初值0， 终值5， 步长1
- K循环， 初值0， 终值7， 步长1
- 定解条件： A[I]=B[J]=C[K]
- 共尝试7*6*8=336次。

方法二 ：

- 用数组A的九个下标分量作为各号考生及格科目的计数器。将三科及格名单共存一个数组，当扫描完毕总及格名单后，凡下标计数器的值为3的就是全及格的，其余则至少有一科不及格的。
- 该方法同样也采用了枚举尝试的方法。


```
main( )
{int  a[10],i,xh;
for(i =1;i<=21;i=i+1)
  {input(xh);
   a[xh]=a[xh]+1;}
for(xh =1;xh<=9;xh=xh+1)
  if(a[xh] =3 )
    print(xh);
}
```


【例2.1.4】统计数字对的出现频率

算法说明：

输入N（2≤N≤100）个数字（在0与9之间），然后统计出这组数中相邻两数字组成的数字对出现的次数。例如：

输入：N=20 {表示要输入数的数目}

0 1 5 9 8 7 2 2 2 3 2 7 8 7 8 7 9 6 5 9

输出：（7，8）=2 （8，7）=3 {指（7，8）、（8，7）数字对出现次数分别为2次、3次

（7，2）=1 （2，7）=1

（2，2）=2

（2，3）=1 （3，2）=1

算 法 设 计：

- 其实并不是只有一维数组这样的数据结构可以在算法设计中有多采的应用，根据数据信息的特点，二维数组的使用同样可以使算法易于实现，此题就正是如此。
- 用10*10的二维数组（行、列下标均为0-9），存储统计结果，i行j列存储数字对（i，j）出现的次数，无需存储原始数据，用其做下标，在数据输入的同时就能统计出问题的结果。

II. 数组使信息有序化

- 当题目中的数据缺乏规律时，很难把重复的工作抽象成循环不变式来完成，但先用数组结构存储这些信息后，问题就迎刃而解。

【例2.1.5】编程将编号“翻译”成英文。例35706“翻译”成three-five-seven-zero-six。

- 算法设计1：
- 1） 编号一般位数较多，可按长整型输入和存储。
- 2） 将英文的“zero-nine”存储在数组中，对应下标为0-9。这样无数值规律可循的单词，通过下标就可以方便存取、访问了。
- 3） 通过求余、取整运算，可以取到编号的各个位数字。用这个数字作下标，正好能找到对应的英文数字。
- 4）考虑输出翻译结果是从高位到低位进行的，而取各位数字，比较简单的方法是从低位开始通过求余和整除运算逐步完成的。所以还要开辟一个数组来存储从低位到高位翻译好的结果，并记录编号的位数，最后倒着从高位到低位输出结果。

【例2.1.6 】一个顾客买了价值x元的商品，并将y元的钱交给售货员。售货员希望用张数最少的钱币找给顾客。

- 问题分析：无论买商品的价值x是多大，找的钱最多需要以下六种币值：50，20，10，5，2，1。
- 算法设计：
- 1）为了能达到找给顾客钱的张数最少，先尽量多地取大面额的币种，由大面额到小面额币种逐渐进行。
- 2）六种面额不是等到差数列，为了能构造出循环不变式，将六种币值存储在数组B。这样，六种币值就可表示为B[i]，i=1，2，3，4，5，6。为了能达到先尽量多地找大面额的币种，六种币值应该由大到小存储。
- 3）另外还要为统计六种面额的数量，同样为了能构造出循环不变式，设置一个有六个元素的累加器数组S，这样操作就可以通过循环顺利完成了。

III. 数组记录状态信息

问题提出：

有的问题会限定在现有数据中，每个数据只能被使用一次，怎么样表示一个数据“使用过”还是没有“使用过”？

一个朴素的想法是：用数组存储已使用过的数据，然后每处理一个新数据就与前面的数据逐一比较看是否重复。这样做，当数据量大时，判断工作的效率就会越来越低。

【例2.1.7 】求X，使X^2为一个各位数字互不相同的九位数。

- 算法分析：只能用枚举法尝试完成此题。由X^2为一个九位数，估算X应在10000-32000之间。
- 算法设计：
- 1） 用一个10 个元素的状态数组p，记录数字0-9在X^2中出现的情况。数组元素都赋初值为1，表示数字0-9没有被使用过。
- 2） 对尝试的每一个数x，求x*x,并取其各个位数字，数字作为数组的下标，若对应元素为1，则该数字第一次出现，将对应的元素赋为0，表示该数字已出现一次。否则，若对应元素为0，则说明有重复数字，结束这次尝试。
- 3） 容易理解当状态数组p中有9个元素为0时，就找到了问题的解。但这样判定有解，需要扫描一遍数组p。为避免这个步骤，设置一个计数器k，在取x*x各个位数字的过程中记录不同的数字的个数，当k=9时就找到了问题的解。

【例2.1.8 】游戏问题：12个小朋友手拉手站成一个圆圈，从某一个小朋友开始报数，报到7的那个小朋友退到圈外，然后他的下一位重新报“1”。这样继续下去，直到最后只剩下一个小朋友，求解这个小朋友原来站在什么位置上呢？

算法设计：

这个问题初看起来很复杂，又是手拉手，又是站成圈，报数到7时退出圈……似乎很难入手。

细细分析起来，首先是如何表示状态的问题。开辟12个元素的数组，记录12个小朋友的状态，开始时将12个元素的数组值均赋为1，表示大家都在圈内。这样小朋友报数就用累加数组元素的值来模拟，累加到7时，该元素所代表的小朋友退到圈外，将相应元素的值改赋为0，这样再累加到该元素时和不会改变，又模拟了已出圈外的状态。

为了算法的通用性，算法允许对游戏的总人数、报数的起点、退出圈外的报数点任意输入。其中n表示做游戏的总人数，k表示开始及状态数组的下标变量，m表示退出圈外的报数点，即报m的人出队，p表示已退出圈外的人数。

IV. 大整数的存储及运算

【例2.1.9】 编程求当N<=100时，N！的准确值。

- 问题分析：
  - 例如：
  - 9！=362880
  - 100！ = 93 326215 443944 152681 699263

856266 700490 715968 264381 621468 592963

895217 599993 229915 608914 463976 156578

286253 697920 827223 758251 185210 916864

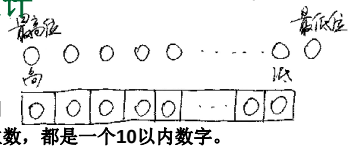
000000 000000 000000 000000

问题分析:

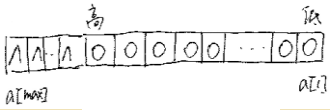
- 计算机存储数据是按类型分配空间的。在PC上：
  - 整型：2个字节16位，则整型数据的范围为-32768—32767；
  - 长整型：4个字节32位，则长整型数据的范围为 -2147483648—2147483647；
  - 实型：4个字节32位，但非精确存储数据，只有六位精度，数据的范围±(3.4e-38~3.4e+38)；
  - 双精度型：8个字节64位的存储空间，数据的范围±(1.7e-308~1.7e+308)，其精确位数是17位。
- 这些类型无法存储100！这样的“大整数”。
- 需要使用更复杂、更有针对性的数据结构。

算法设计

基于存储的考虑



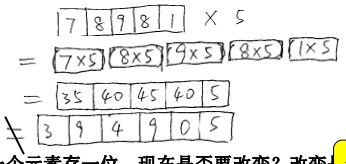
- 每一位数，都是一个10以内数字。
- 数组是有头有尾的：a[1]~a[n]。高位、低位谁头谁尾？
- 低位固定，而高位不定。最低位为a[1]，且在高端要为问题最大存储数据留够空位。



算法设计

基于功能的考虑

- 100！=1*2*3*……*99*100。
- 按此方法计算，最困难的操作是：
  大整数*乘数，其中乘数<=100。

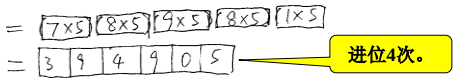


- 原来一个元素存一位，现在是否要改变？改变
- 问题出在哪里？
- 当低位元素计算后的值超过9时，需向高位元素进位。

如何进位？

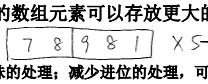
算法设计

基于性能的考虑



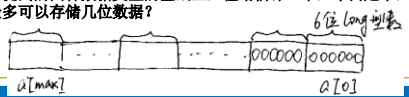
进位4次。

- int a[n]中的数组元素可以存放更大的数。如，每个存3位。



进位几次？

- 进位需要特殊的处理；减少进位的处理，可以提高效率。
- 每个元素处理的位数越多，进位次数越少。
- 在前面提到的四种数据类型的基础上，粗略估计一下，本问题中，每个元素最多可以存储几位数据？



2.1.2 优化算法的数学模型

说到数学建模，有好多人感到不知所云，下面我们看一个很简单的例子：

【例2.1.10】已知有五个数，求前四个数与第五个数分别相乘后的最大当数。给出两个算法分别如下：

```
max1(int a,b,c,d,e)
{  int x ;
   a=a*e;
   b=b*e;
   c=c*e;
   d=d*e;
   if( a>b)
      x=a;
   else
      x=b;
   if (c>x)
      x=c;
   if (d>x)
      x=d;
   print(x);
   }
```

```
max2(int a,b,c,d,e)
{ int x
   if (a>b)
      x=a;
   else
      x=b;
      if (c>x)
      x=c;
   if (d>x)
      x=d;
      x=x*e;
   print(x);
   }
```


操作 / 算法	乘法	赋值	条件判断
Max1	4	7	3
Max2	1	4	3

以上两个算法基于的数学模型是不同的，一个算法先积再求最大值，另一个算法先求最大值再求积，求从上表可以看出，后一个算法的效率明显要高于前一个算法。

数学建模就是把现实世界中的实际问题加以提炼，抽象为数学模型，求出模型的解，验证模型的合理性，并用该数学模型所提供的解答来解释现实问题，我们把数学知识的这一应用过程称为数学建模。

- 数学建模的基本方法

从分析问题的几种简单的、特殊的情况中，发现一般规律或作出某种猜想，从而找到解决问题的途径。这种研究问题方法叫做归纳法。即归纳法是从简单到复杂，由个别到一般的一种研究方法。

杨辉三角形的应用

【例2. 1. 11】求n次二项式各项的系数：已知二项式的展 开式为：

$$(a+b)^n=C_n^0a^n+C_n^1a^{n-1}b+C_n^2a^{n-2}b^2+\cdots\cdots+C_n^nb^n$$

问题分析：若只用的数学组合数学的知识，直接建模

$$C_n^k=\frac{n!}{k!(n-k)!}$$

k=0，1，2，3……n。

用这个公式去计算，n+1个系数，即使你考虑到了前后系数之间的数值关系：算法中也要有大量的乘法和除法运算，效率是很低的。

数学知识是各阶多项式的系数呈杨辉三角形的规律

$(a+b)^0$ 1

$(a+b)^1$ 1 1

$(a+b)^2$ 1 2 1

$(a+b)^3$ 1 3 3 1

$(a+b)^4$ 1 4 6 4 1

$(a+b)^5$ ……

则求n次二项式的系数的数学模型就是求n阶杨辉三角形。

算法设计要点： 除了首尾两项系数为1外，当n>1时，$(a+b)^n$的中间各项系数是$(a+b)^{n-1}$的相应两项系数之和，如果把$(a+b)^n$的n+1的系数列为数组c，则除了c(1)、c(n+1)恒为1外，设$(a+b)^n$的系数为c(i)，$(a+b)^{n-1}$ 的系数设为c'(i)。则有：

c(i)=c'(i)+c'(i-1)

而当n=1时，只有两个系数 c(1) 和 c(2) (值都为1)。不难看出，对任何n，$(a+b)^n$的二项式系数可由$(a+b)^{n-1}$的系数求得，直到n=1时，两个系数有确定值，故可写成递归子算法。

公倍数的应用

【例2.1.12】编程完成下面给“余”猜数的游戏：
你心里先想好一个1~100之间的整数x，将它分别除以3、5和7并得到三个余数。你把这三个余数告诉计算机，计算机能马上猜出你心中的这个数。游戏过程如下：

please think of a number between 1 and 100
your number divided by 3 has a remainder of? 1
your number divided by 5 has a remainder of? 0
your number divided by 7 has a remainder of? 5
let me think a moment...
your number was 40

问题分析：算法的关键是：找出余数与求解数之间的关系，也就是建立问题的数学模型。

数学模型：

1）不难理解当s=u+3*v+3*w时，s除以3的余数与u除以3的余数是一样的。

2）对s=cu+3*v+3*w，当c除以3余数为1的数时，s除以3的余数与u除以3的余数也是一样的。证明如下：

c 除以 3余数为1，记c=3*k+1，则s=u+3*k*u+3*v+3*w，由1)的结论，上述结论正确。

记a，b，c分别为所猜数据d除以3，5，7后的余数，则d=70*a+21*b+15*c。

为问题的数学模型，其中70称作a的系数，21称作b的系数，15称作c的系数。

由以上数学常识，a、b、c的系数必须满足：

1） b、c的系数能被3整除，且a的系数被3整除余1；这样d除以3的余数与a相同。

2） a、c的系数能被5整除，且b的系数被5整除余1；这样d除以5的余数与b相同。

3) a、b的系数能被7整除，且c的系数被7整除余1；这样d除以7的余数与c相同。

由此可见：

c的系数是3和5的最公倍数且被7整除余1，正好是15；

a的系数是7和5的最公倍数且被3整除余1，最小只能是70；

b的系数是7和3的最公倍数且被5整除余1，正好是21。

递推关系求解方程

【例2.1.13】核反应堆中有α 和β 两种粒子，每秒钟内一个α 粒子可以反应产生3个β 粒子，而一个β 粒子可以反应产生1个α 粒子和2个β 粒子。若在t=0时刻的反应堆中只有一个α 粒子，求在t时刻的反应堆中α 粒子和β 粒子数。

数学模型1：本题中共涉及两个变量，设在i时刻α 粒子数为n_i，β 粒子数为m_i，则有：$n_0=1, m_0=0, n_i=m_{i-1}$，$m_i=3n_{i-1}+2m_{i-1}$

算法设计1：本题方便转化为求数列N和M的第t项，可用递推的方法求得n_t和m_t，此模型的算法如下：

数学模型2：设在t时刻的α 粒子数为f（t），β 粒子数为g(t)，依题可知：

$$\begin{cases} g(t)=3f(t-1)+2g(t-1) & (1) \\ f(t)=g(t-1) & (2) \\ g(0)=0, f(0)=1 \end{cases}$$

下面求解这个递归函数的非递归形式

由（2）得$f(t-1)=g(t-2)$ （3）

将（3）代入（1）得

$g(t)=3g(t-2)+2g(t-1)$ （t≥2） （4）

$g(0)=0, g(1)=3$

（4）式的特征根方程为：

$x^2-2x-3=0$

其特征根为x1=3, x2= -1

所以该式的递推关系的通解为

$g(t)=C_1\cdot 3^t+C_2\cdot(-1)^t$

代入初值g(0)=0,g(1)=3得

$C_1+C_2=0$

$3C_1—C_2=3$

解此方程组 $C_1=\frac{3}{4},C_2=-\frac{3}{4}$

所以该递推关系的解为

$\therefore g(t)=\frac{3}{4}\cdot 3^t-\frac{3}{4}\cdot(-1)^t$

$f(t)=g(t-1)=\frac{3}{4}\cdot 3^{t-1}-\frac{3}{4}\cdot(-1)^{t-1}$

即

$f(t)=\frac{3^t}{4}+\frac{3}{4}\cdot(-1)^t$

2.1.3 循环

- 设计算法重复处理大量数据的思路：以不变应万变；
  - 所谓“不变”是指循环体内运算的表现形式是不变的，而每次具体的执行内容却是不尽相同的。在循环体内用不变的运算表现形式去描述各种相似的重复运算。

1 循环设计要点

- 循环控制-熟悉；
- 设计要点：
  - “自顶向下”的设计方法
  - 由具体到抽象设计循环结构
  - 注意算法的效率

- 【例2.1.14】找出1000以内所有完数。
- 如：28的因子为1、2、4、7，14，而28=1+2+4+7+14。因此28是“完数”。编算法找出1000之内的所有完数，并按下面格式输出其因子：28 it's factors are 1，2，4，7，14。

- 问题分析：
- 1、这里不是要质因数，所以找到因数后也无需将其从数据中“除掉”。
- 2、每个因数只记一次，如8的因数为1，2，4而不是1，2，2，2，4。(注：本题限定因数不包括这个数本身)

- 核心算法设计
  - for(i=0;i<n;i=i+1){
    判断i是否是完数;
    if是“完数”则按规则输出;
    }

判断是否是完数的方法是“不变”，被判断的数是“万变”。

自顶向下，逐步求精

- 判断i是否是完数
  - for(j=2;j<i;j=j+1)
    找i的因子，并累加;
    if 累加值等于i，则i是完数;

- 判断i是否是完数
  - s=1;
    for(j=2;j<i;j=j+1)
    if (i % j==0)
    s=s+j;
    if (s==i) i是“完数”;

- 【例2.1.5】编写算法：根据参数n打印具有下面规律的图形，如，当n=4时，图形如下：
  - 1
  - 5 2
  - 8 6 3
  - 10 9 7 4

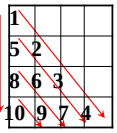
1
6 2
10 7 3
13 11 8 4
15 14 12 9 5
n=5

- 对于不太熟悉的问题，其“不变”不易抽象；

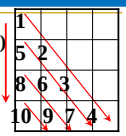
- 【例2.1.15】问题分析：
  - 容易发现图形中数据排列的规律。

```
1
5 2
8 6 3
10 9 7 4
```

  - 可不可以从1—最大数，通过循环，直接输出？
    - printf是按照从左至右、从上至下的顺序；若想直接输出，只有找出公式，循环计算得到序列：1-\n-5-2-\n-8-6-3-\n-10-9-7-4.
  - 另一种思路
    - 先用一个数组按此顺序存储数据，再正常输出；
    - 为数组赋值，也要用循环，如何循环？又要回到找规律公式的路上吗？
    - 斜着能循环吗？



斜行i取值(1~n)
列j取值(1~n+1-i)



- 用斜行、列描述新的循环方向。
- 这样可以描述循环。但数组元素的存取还是基于“行列”，能否简单转换？
- 关键问题：第i斜行、j列的数据对应于第几行第几列的元素？

a[i-1+j][j]

斜[1，1]—a[1,1]
斜[1，2]—a[2,2]
斜[1，3]—a[3,3]
斜[1，4]—a[4,4]

斜[2，1]—a[2,1]
斜[2，2]—a[3,2]
斜[2，3]—a[4,3]

斜[3，1]—a[3,1]
斜[3，2]—a[4,2]

斜[4，1]—a[4,1]

- 列号相同；
- 行号(显然行号与列号有关)
  - 第1斜行，对应行号1—n，行号与列号j同；
  - 第2斜行，对应行号2—n，行号比列号j大1；
  - 第3斜行，对应行号3—n，行号比列号j大2；

- 【例2.1.16】求：$1/1!-1/3!+1/5!-1/7!+\ldots+(-1)^{n+1}/(2n-1)!$

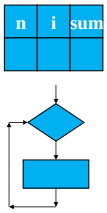
- 问题分析：此问题中既有累加又有累乘，累加的对象是累乘的结果。
  数学模型1：$S_n=S_{n-1}+(-1)^{n+1}/(2n-1)!$
  算法设计1：直接利用题目中累加项通式，构造出循环体不变式为：
  $S=S+(-1)^{n+1}/(2n-1)!$
  需要用二重循环来完成算法。

- 算法设计1：
  - 外层循环求累加S=S+A;
  - 内层循环求累乘A= $(-1)^{n+1}/(2n-1)!$ 。

2.2 递归设计

- 程序结构化设计的三种基本结构，顺序、选择、循环是不是必须的？
- 回答：在很多高级语言中，不是。可以抛弃谁呢？
- 递归能取代循环的作用 。
- 【例2.2.1】根据参数n，计算1+2+……+n。

```
void sum_loop(int n){
        int i,sum=0;
        for (i=1;i<=n;i++){
          sum = sum + i;
        }
        printf("\nsum = %d",sum);
}
```

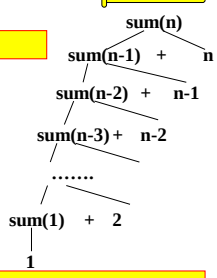


递归设计思路

- 根据参数n，计算1+2+……+n。

```
int sum_recursive(int n){
  int sum=0;
  if (n == 1) sum = 1;
  else
    sum = sum_recursive(n-1) + n;
  printf("\nsum = %d",sum);
  return sum;
}
```

递归树！

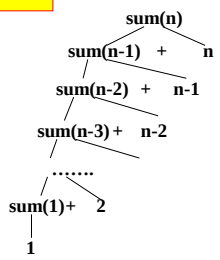


- 递归算法是一个模块(函数、过程)除了可调用其它模块(函数、过程)外，还可以直接或间接地调用自身的算法。直接/间接递归调用。

- 根据参数n，计算1+2+……+n。

- $T(n) = T(n-1)+O(1)$
  $= T(n-2)+O(1)+O(1)$
  $= \ldots\ldots\ldots$
  $= n*O(1)=O(n)$

```
int sum_recursive(int n){
  int sum=0;
  if (n == 1) sum = 1;
  else
    sum = sum_recursive(n-1) + n;
  printf("\nsum = %d",sum);
  return sum;
}
```



递归的定义

递归是算法设计中一种重要的方法，可以使许多程序结构简化，易理解，易证明。

递归定义的算法有两个部分：

递归基：直接定义最简单情况下的函数值；

递归步：通过较为简单情况下的函数值定义一般情况下的函数值。

应用条件与准则

- 适宜于用递归算法求解的条件是：
- （1）问题具有某种可借用的类同自身的子问题描述的性质；
- （2）某一有限步的子问题（也称作本原问题）有直接的解存在。

递归算法设计

递归算法既是一种有效的算法设计方法，也是一种有效的分析问题的方法。递归算法求解问题的基本思想是：对于一个较为复杂的问题，把原问题分解成若干个相对简单且类同的子问题，这样，原问题就可递推得到解。

适宜于用递归算法求解的问题的充分必要条件是：

（1）问题具有某种可借用的类同自身的子问题描述的性质；

（2）某一有限步的子问题（也称作本原问题）有直接的解存在。

当一个问题存在上述两个基本要素时，该问题的递归算法的设计方法是：

（1）把对原问题的求解设计成包含有对子问题求解的形式。

（2）设计递归出口。

递归算法分类

- 单路递归（一个递归过程中只有一个递归入口）
- 多路递归（一个递归过程中有多个入口）
- 间接递归（函数可通过其他函数间接调用自己）
- 迭代递归（每次递归调用都包含一次循环递归）

递归算法举例

【例2.2.2】阶乘函数

$$n!=n*(n-1)*(n-2)*.....*1$$

$$n!=n*(n-1)!$$

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

递归基

递归步

【例2.2.3】 Fibonacci数列

无穷数列1，1，2，3，5，8，13，21，34，55，……，称为Fibonacci数列。

$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

递归基

递归步

第n个Fibonacci数可递归地计算如下：

```
int fibonacci(int n)
 {
   if (n <= 1) return 1;
   return fibonacci(n-1)+fibonacci(n-2);
 }
```

【例2.2.4】欧几里德算法

- gcd (m, n) = gcd (n, m mod n)
- 输入 正整数m和n
  输出 m和n的最大公因子
  1. 如果n = 0, 计算停止返回m, m即为结果；否则继续2。
  2. 记r为m除以n的余数，即r=m mod n。
  3. 把n赋值给m，把r赋值给n，继续1。

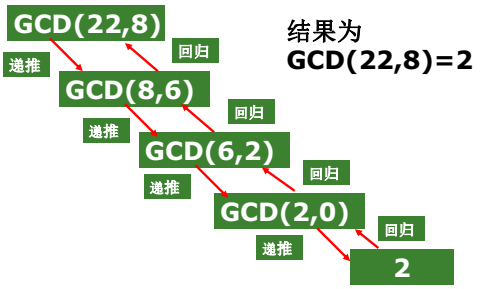
- 伪代码如下(循环)：

```
Euclid(m, n)
{
   while n<> 0
   {
        r = m % n;
        m = n;
        n = r;
    }
}
```

递归代码：

```
GCD(m,n) // 约定m>n //
{  if n=0 return(m)
   else return (GCD(n,m mod n))
}
```


例： GCD(22,8) = GCD(8,6) = GCD(6,2) = GCD(2,0) = 2;



【例2.2.5】 Ackerman函数

当一个函数及它的一个变量是由函数自身定义时，称这个函数是**双递归函数**。

Ackerman函数$A(n, m)$定义如下：

$$\begin{cases} A(1,0)=2 & \\ A(0,m)=1 & m\geq 0 \\ A(n,0)=n+2 & n\geq 2 \\ A(n,m)=A(A(n-1,m),m-1) & n,m\geq 1 \end{cases}$$

【例2.2.6】基于递归的插入排序

算法的基本思想：将待插入的关键字插入到已经排好序的序列中。

直接插入排序的算法

```
void InsertSort(SqList &L)
  {  for (int i=2;i<=L.length;++i)
        if (LT(L.r[i].key,L.r[i-1].key))
        { L.r[0]=L.r[i];  // L.r[0]为监视哨
            for (int j=i-1; LT(L.r[0].key,L.r[j].key); --j)
                L.r[j+1]=L.r[j];
                        L.r[j+1]=L.r[0];
        }
  }
```


假设要对n个元素的数组A进行排序，可以按如下步骤进行：

1）递归基：当n=1时，数组中只有一个元素，它已经是排序的；

2）递归步：如果前面k-1个元素已经排序，只要将第k个元素逐一与前面k-1个比较，把它插入适当的位置，即可完成k个元素的排序。

【例2.2.7】多项式求值问题

有如下多项式：

$$P_n(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$$

如果分别对每一项求值，需要$n(n+1)/2$个乘法，效率很低。

$$P_n(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$$
$$=((\cdots(((a_n)x+a_{n-1})x+a_{n-2})x\cdots)x+a_1)x+a_0$$

递归表示：

1）递归基：n=0,P_0=a_n

2）递归步：对任意的k,1≤k≤n，如果前面k-1步已经计算出P_{k-1}：

$$P_{k-1}=a_nx^{k-1}+a_{n-1}x^{k-2}+\cdots+a_{n-k+2}x+a_{n-k+1}$$

则有：$P_k=xP_{k-1}+a_{n-k}$

递归算法：

```
Float horner_pol(float x, float A[ ], int n)
{
  float p;
  if (n==0)
    p=A[0];
  else
    p=horner_pol(x, A,n-1)*x+A[n];
  return p;
}
```


时间复杂性分析

算法的时间复杂性由如下递归方程表示：

$$\begin{cases} f(0)=0 \\ f(n)=f(n-1)+1 \end{cases}$$

$$f(n)=O(n)$$

【例2.2.8】求数组主元素算法

A是具有n个元素的数组，x是A中的一个元素，若A中有一半以上的元素与x相同，则x是A的主元素。

算法分析：

方法一、每个元素都和其他元素进行比较，设计一个计数器进行相同元素个数的计数。若某个元素的计数器值大于n/2，则该元素是主元素。

时间性能？O(n*n)

方法二、对元素进行排序，然后计算相同元素的个数，从而找到主元素。O(nlogn)

方法三、随机法

方法四、寻找数组的中值元素，即数组中第n/2大的元素，因为主元素也必定是中值元素。然后从头到尾扫描数组，如果中值元素的个数大于n/2，则中值元素就是主元素。

因为数组主元素在数组中出现的次数大于n/2，所以有下面的结论：

结论1、移去数组中两个不同的元素后，如果原数组中有主元素，那么该主元素仍然是数组的主元素。

结论2、如果数组2K个元素中有K个元素相同（K<n/2），那么移去这2K个元素后，如果原来数组中有主元素，则该主元素仍然是新数组的主元素。

- **【例2.2.9】构造一个3×3的魔方：把数字1-9添入如图的表格中**

2	7	6
9	5	1
4	3	8

要求每横，竖，斜列之和均相等（如图是一种情况）。输出所有可能的魔方。

- 【例2.2.10】 RSA 加密算法是目前使用的最广泛的一种密钥体制，主要用于信息的加密、解密和数字签名。而其核心算法就是模幂算法，所以提高模幂算法的效率直接影响到加密算法的效率。模幂逻辑实现方法也比较多，已有模幂算法是将指数化为其对应二进制数来实现.

- 递归算法步骤(以 xr mod p 为例) :
- (1) 若 r ＞2，判断 r 的奇偶性，若 r 为奇数则转(2) ，若 r 为偶数则转(3) 。若 r≤2，则转(4) ;
- (2) r 为奇数，r = r/2，m［i + +］= x，x = xmodp，转(1) ;
- (3) r 为偶数，r = r/2，x = xmod p，转(1)
- (4) 若 r 为 2，则计算(m［0］* m［1］* …*m［i］* x* x) mod p; 若 r 为 1，则计算 (m［0］* m［1］* …* m［i］* x) mod p;


```
int Mod2(int x, int r, int p) {
    if (r > 2) {
        if (r % 2 == 1) {
            r = r / 2;
            return (x * Mod2((x * x) % p, r,
p));
            }// r mod 2 = =1 情况下递归运算
        if (r % 2 == 0) {
            r = r / 2;
            return (Mod2((x * x) % p, r, p)) %
p;
        }// r mod 2 = =0 情况下递归运算
    }
    if (r == 2)
        return (x * x) % p;
    if (r == 1)
        return x % p;// 递归算法的两个出口

}
```


- 【例2.2.11】求实对称矩阵的特征值
- 设实对称矩阵如下：

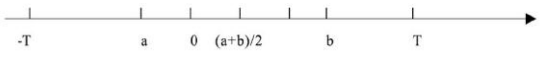
$$T=\begin{vmatrix} \delta_1 & \alpha_1 & \cdots & 0 \\ \alpha_1 & \delta_2 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \alpha_{n-1} \\ 0 & \cdots & \cdots & \delta_n \end{vmatrix}$$

用$P_x(\lambda)$表示矩阵T- λ的阶顺序主子式为：

$$p_n(\lambda)=\begin{vmatrix} \delta_1-\lambda & \alpha_1 & \cdots & 0 \\ \alpha_1 & \delta_2-\lambda & \cdots & \cdots \\ \cdots & \cdots & \cdots & \alpha_{n-1} \\ 0 & \cdots & \alpha_{n-1} & \delta_n-\lambda \end{vmatrix}$$

$$P_n(\lambda)=(\delta_n-\lambda)p_{n-1}(\lambda)-a_{n-1}^2 p_{n-2}(\lambda)$$

- 设a,b为二实数,大于a的特征值个数为$s_n(a)$，大于b的特征值个数为$s_n(b)$
- 若矩阵的第k个特征值λ_k（特征值按从大到小顺序排列）位于区间(a,b]中，而第k+1个特征值λ_{k+1}不在区间（a,b]中，则$s_n(a)=k,s_n(b)<k$
- 取区间中点(a+b)/2,计算$s_n((a+b)/2)$…



- 【例2.2.12】楼梯有n阶台阶，上楼可以一步上1阶，也可以一步上2阶，编一程序计算共有多少种不同的走法。例如，当n=3时，共有3种走法，即1+1+1，1+2，2+1。
  算法分析：设n阶台阶的走法数为f(n)，则：

$$F(n)=\begin{cases}1 & n=1\\ 2 & n=2\\ f(n-1)+f(n-2) & n>2\end{cases}$$

- 【例2.2.13】核反应堆中α、β两种粒子，每单位时间，1个α粒子分裂为3个β粒子，1个β粒子分裂为2个β粒子和1个α粒子，假设t=0时刻，反应堆中有1个α粒子，求在t时刻的反应堆中α、β粒子数？

【例2.2.14】排列问题

有n个元素，编号为1,2,…,n，用一个具有n个元素的数组A来存放所生成的排列，然后输出它们。

操作步骤：

（1）数组第一个元素为1，即排列的第一个元素为1，生成后面的n-1个元素的排列。

（2）数组第一个元素与第二个元素互换，使得排列的第一个元素为2，生成后面的n-1个元素的排列。

（3）如此继续，最后数组第一个元素与第n个元素互换，使得排列的第一个元素为n，生成后面的n-1个元素的排列。

在上面的第一步中，为了生成后面n-1个元素的排列，继续采用如下步骤：

（1）数组第二个元素为2，即排列的第二个元素为2，生成后面的n-2个元素的排列。

（2）数组第二个元素与第三个元素互换，使得排列的第二个元素为3，生成后面的n-2个元素的排列。

（3）如此继续，最后数组第二个元素与第n个元素互换，使得排列的第二个元素为n，生成后面的n-2个元素的排列。

这种步骤一直继续，当排列的前n-2个元素已经确定后，为生成后面2个元素的排列，可以采用如下步骤：

（1）数组第n-1个元素为n-1，即排列的第n-1个元素为n-1，生成后面的1个元素的排列，此时数组中的n个元素已经构成一个排列。

（2）数组第n-1个元素与第n个元素互换，使得排列的第n-1个元素为n，生成后面的n-2个元素的排列。

通过上述分析，设排列算法perm(A,k,n)表示生成后面k个元素的排列，有如下递归表示：

（1）递归基：k=1，只有一个元素，已构成一个排列。

（2）递归步：对任意的k,1≤k≤n，如果可由算法perm(A,k-1,n)完成数组后面k-1个元素的排列，为完成数组后面k个元素的排列Perm(A,k,n)，逐一对数组第n-k元素与数组中第n-k~n元素进行互换，每互换一次，就执行一次perm(A,k-1,n)操作，产生一个排列。

【例2.2.15】整数划分问题

将正整数n表示成一系列正整数之和：$n=n_1+n_2+\ldots+n_k$，

其中$n_1\geq n_2\geq\ldots\geq n_k\geq 1$，$k\geq 1$。

正整数n的这种表示称为正整数n的划分。求正整数n的不同划分个数。

例如正整数6有如下11种不同的划分：

6；

5+1；

4+2，4+1+1；

3+3，3+2+1，3+1+1+1；

2+2+2，2+2+1+1，2+1+1+1+1；

1+1+1+1+1+1。

前面的几个例子中，问题本身都具有比较明显的递归关系，因而容易用递归函数直接求解。

在本例中，如果设$p(n)$为正整数n的划分数，则难以找到递归关系，因此考虑增加一个自变量：将最大加数n_1不大于m的划分个数记作$q(n,m)$。可以建立$q(n,m)$的如下递归关系。

(1) $q(n,1)=1$, $n\geq 1$；

当最大加数n_1不大于1时，任何正整数n只有一种划分形式，即 $n=\overbrace{1+1+\cdots+1}^{n}$

(2) $q(n,m)=q(n,n)$, $m\geq n$;

最大加数n_1实际上不能大于n。因此，$q(1,m)=1$。

(3) $q(n,n)=1+q(n,n-1)$；

正整数n的划分由$n_1=n$的划分和$n_1\leq n-1$的划分组成。

(4) $q(n,m)=q(n,m-1)+q(n-m,m)$, $n>m>1$；

正整数n的最大加数n_1不大于m的划分由$n_1=m$的划分和$n_1\leq n-1$ 的划分组成。

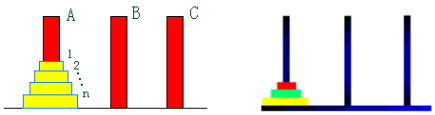
$$q(n,m)=\begin{cases}1 & n=1,m=1\\ q(n,n) & n<m\\ 1+q(n,n-1) & n=m\\ q(n,m-1)+q(n-m,m) & n>m>1\end{cases}$$

正整数n的划分数$p(n)=q(n,n)$。

【例2.2.16】 Hanoi塔问题

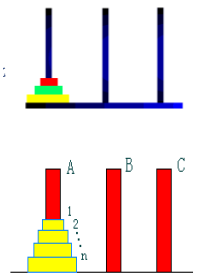
设a,b,c是3个塔座。开始时，在塔座a上有一叠共n个圆盘，这些圆盘自下而上，由大到小地叠在一起。各圆盘从小到大编号为1,2,…,n，现要求将塔座a上的这一叠圆盘移到塔座b上，并仍按同样顺序叠置。在移动圆盘时应遵守以下移动规则：

- 规则1：每次只能移动1个圆盘；
- 规则2：任何时刻都不允许将较大的圆盘压在较小的圆盘之上；
- 规则3：在满足移动规则1和2的前提下，可将圆盘移至a,b,c中任一塔座上。



Hanoi塔问题

```
void hanoi(int n, int a, int b, int c)
  {
    if (n > 0)
    {
      hanoi(n-1, a, c, b);
      move(a,b);
      hanoi(n-1, c, b, a);
    }
  }
```



递归与循环的比较

- ◆递归与循环都是解决“重复操作”的机制。
- ◆递归使一些复杂的问题处理起来简单明了。
- ◆就效率而言，递归算法的实现往往要比迭代算法耗费更多的时间（调用和返回均需要额外的时间）与存贮空间（用来保存不同次调用情况下变量的当前值的栈空间），也限制了递归的深度。
- ◆每个迭代算法原则上总可以转换成与它等价的递归算法；反之不然 。
- ◆递归的层次是可以控制的，而循环嵌套的层次只能是固定的，因此递归是比循环更灵活的重复操作的机制。

优点： 结构清晰，可读性强，而且容易用数学归纳法来证明算法的正确性，因此它为设计算法、调试程序带来很大方便。

递归小结

缺点： 递归算法的运行效率较低，无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。

2.3 递归方程的求解

- **算法的运行时间，存在着递归的关系，进行算法分析时就要建立相应的递归方程。**
- **目标：求解递归方程。**

2.3.1生成函数及其性质

- **生成函数及其性质**

算法的运行时间，随着递归深度的增加而增加。假定序列： $a_0, a_1, \ldots, a_k$

表示递归算法在不同递归深度时的运行时间，则序列中的每一个元素之间，存在着一定的递归关系。

一般希望了解当深度为$k=n$时，序列a_n的值。

如果可以借助一个“参数”z，来建立一个无穷级数的和：

$$G(z)=a_0+a_1z+a_2z^2\cdots+\cdots=\sum_{k=0}^{\infty}a_kz^k$$

然后通过对函数$G(z)$的一系列演算，得到序列 $a_0, a_1, \ldots, a_k$ 的一个通项表达式，则可以获得递归算法在深度为$k=n$时的运行时间。

定义：令 $a_0, a_1, a_2, \ldots$ 是一个实数序列，构造如下的函数：

$$G(z)=a_0+a_1z+a_2z^2\cdots+\cdots=\sum_{k=0}^{\infty}a_kz^k$$

则函数$G(z)$称为序列 $a_0, a_1, a_2, \ldots$ 的生成函数。

当序列 $a_0, a_1, a_2, \ldots$ 确定时，对应的生成函数只依赖于“参数”z。反之，当生成函数确定时，所对应的序列也被确定。

生成函数的性质：

1）加法

$$\alpha G(z)+\beta H(z)=\alpha\sum_{k=0}^{\infty}a_kz^k+\beta\sum_{k=0}^{\infty}b_kz^k=\sum_{k=0}^{\infty}(\alpha a_k+\beta b_k)z^k$$

2）移位

$$\alpha^m G(z)=\alpha\sum_{k=m}^{\infty}a_{k-m}z^k$$

3）乘法

$$G(z)H(z)=\sum_{k=0}^{\infty}a_kz^k\cdot\sum_{k=0}^{\infty}b_kz^k=\sum_{k=0}^{\infty}c_kz^k$$

$$c_n=\sum_{k=0}^{n}a_kb_{n-k}$$

4）Z变换：设 $G(z)=\sum_{k=0}^{\infty}a_kz^k$

是序列 $a_0, a_1, a_2, \ldots$ 的生成函数，则

$$G(cz)=a_0+a_1(cz)+a_2(cz)^2+a_3(cz)^3+\cdots$$
$$=a_0+ca_1z+c^2a_2z^2+c^3a_3z^3+\cdots$$

是序列 $a_0, ca_1, c^2a_2, \ldots$ 的生成函数。有：

$$\frac{1}{1-cz}=1+cz+c^2z^2+c^3z^3+\cdots$$

所以，$\frac{1}{1-cz}$ 是序列 $1, c, c^2, c^3, \cdots$ 的生成函数。

当c=1时，有：$\frac{1}{1-z}=1+z+z^2+z^3+\cdots$

$$\frac{1}{1-cz}=1+cz+c^2z^2+c^3z^3+\cdots$$

当c=1时，有：$\frac{1}{1-z}=1+z+z^2+z^3+\cdots$

则，$\frac{1}{1-z}$ 是序列 $1,1,1,1,\cdots$ 的生成函数。

利用 $\frac{1}{2}(G(z)+G(-z))=a_0+a_2z^2+a_4z^4+\cdots$

可以去掉级数中的奇数项，同样，也可以利用：

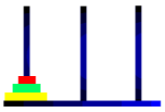
$$\frac{1}{2}(G(z)-G(-z))=a_1z+a_3z^3+a_5z^5+\cdots$$

可以去掉级数中的偶数项。

2.3.2 用生成函数求解递归方程

Hanoi塔问题

```
void hanoi(int n, int a, int b, int c)
 {
   if (n > 0)
   {
     hanoi(n-1, a, c, b);
     move(a,b);
     hanoi(n-1, c, b, a);
   }
 }
```



$$\begin{cases}h(1)=1\\h(n)=2h(n-1)+1\end{cases}$$

为了求解递归方程，用$h(n)$作为系数，构造一个生成函数如下：

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

为了求出$h(n)$的值，对$G(x)$进行演算，求出它的解析表达式，再把解析表达式转换成对应的幂级数，级数中x^n的系数，即为$h(n)$的值。

令：

$$G(x)-2xG(x)=h(1)x+h(2)x^2+h(3)x^3+\cdots-2h(1)x^2-2h(2)x^3-\cdots$$
$$=h(1)x+(h(2)-2h(1))x^2+(h(3)-2h(2))x^3+\cdots$$

根据公式（1）以及 $\frac{1}{1-z}=1+z+z^2+\cdots$

可得：$(1-2x)G(x)=x+x^2+x^3+\cdots$
$=\frac{x}{1-x}$

所以：$G(x)=\frac{x}{(1-x)(1-2x)}$

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

求得：$A=-1,\quad B=1.$

则：$G(x)=\frac{1}{(1-2x)}-\frac{1}{(1-x)}$
$=\sum_{k=1}^{\infty}(2^k-1)x^k$

$$h(n)=2^n-1$$

2.3.3 用特征方程求解递归方程

K阶常系数线性齐次递归方程为：

$$\begin{cases}f(n)=a_1f(n-1)+a_2f(n-2)+\cdots+a_kf(n-k)\\f(i)=b_i\end{cases} \qquad (1)$$

用 x^n 取代 $f(n)$：$x^n=a_1x^{n-1}+a_2x^{n-2}+\cdots+a_kx^{n-k}$

两边除以 x^{n-k}：$x^k=a_1x^{k-1}+a_2x^{k-2}+\cdots+a_k$

则：$x^k-a_1x^{k-1}-a_2x^{k-2}-\cdots-a_k=0$

为公式（1）的特征方程。

可以求出特征方程的根，得到递归方程的通解。再利用递归方程的初始条件，建立通解中的待定系数，从而得到递归方程的解。

求解过程中有两种情况分别进行讨论：

第一种情况，特征方程的k个根互不相同。这时令 $q_1,q_2,\cdots,q_k$

是特征方程的根，则公式（1）的通解为：

$$f(n)=c_1q_1^n+c_2q_2^n+\cdots+c_kq_k^n$$

第二种情况，特征方程的k个根中有r个重根 $q_i,q_{i+1},\cdots,q_{i+r-1}$

这时递归方程的通解为：

$$f(n)=c_1q_1^n+\cdots+c_{i-1}q_{i-1}^n+$$
$$(c_i+c_{i+1}n+\cdots+c_{i+r-1}n^{r-1})q_i^n+\cdots+c_kq_k^n$$

例：三阶常系数线性齐次递归方程如下：

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

特征方程为：$x^3-6x^2+11x-6=0$

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

特征根为：$q_1=1 \quad q_2=2 \quad q_3=3$

所以递归方程的通解为：

$$\begin{aligned} f(n) &= c_1q_1^n+c_2q_2^n+c_3q_3^n \\ &= c_1+c_2 2^n+c_3 3^n \end{aligned}$$

由初始条件有：$\begin{cases} f(n)=6f(n-1)-11f(n-2)+6f(n-3) \\ f(0)=0 \\ f(1)=2 \\ f(2)=10 \end{cases}$

$$f(0)=c_1+c_2+c_3=0$$
$$f(1)=c_1+2c_2+3c_3=2$$
$$f(2)=c_1+4c_2+9c_3=10$$

故：$c_1=0 \quad c_2=-2 \quad c_3=2 \quad f(n)=2(3^n-2^n)$

2.3.4 用递推方法求解递归方程

$$\begin{cases} f(n)=bf(n-1)+g(n) \\ f(0)=c \end{cases}$$

$$\begin{aligned} f(n) &= bf(n-1)+g(n) \\ &= b(bf(n-2)+g(n-1))+g(n) \\ &= b^2f(n-2)+bg(n-1)+g(n) \\ &\cdots \\ &= b^nf(0)+b^{n-1}g(1)+\cdots+bg(n-1)+g(n) \\ &= cb^n+\sum_{i=1}^{n}b^{n-i}g(i) \end{aligned}$$

汉诺塔问题：

$$\begin{cases} h(n)=2h(n-1)+1 \\ h(1)=1 \end{cases}$$

利用前面的递推定义：$b=2 \quad c=1 \quad g(n)=1$

$$\begin{aligned} h(n) &= cb^{n-1}+\sum_{i=1}^{n-1}b^{n-i-1}g(i) \\ &= 2^{n-1}+2^{n-2}+\cdots+2+1 \\ &= 2^n-1 \end{aligned}$$

作业：

1）用生成函数求解递归方程：

(1) $f(n)=2f(n-1)+1$, $f(1)=2$

(2) $f(n)=2f(n/2)+cn$, $f(1)=0$

2）解递归方程：三阶常系数线性齐次递归方程如下：

$$\begin{cases} f(n)=5f(n-1)-7f(n-2)+3f(n-3) \\ f(0)=1 \\ f(1)=2 \\ f(2)=7 \end{cases}$$

3）解递归方程：

$f(n)=f(n-2) \quad f(0)=5 \quad f(1)=-1$

$f(n)=f(n-1)+n^2 \quad f(0)=0$