第一章　算法

- 教学目标

  了解算法的基本概念、算法描述语言， 掌握几种常见算法的基本实现，并能分析算法的时间和空间复杂度。

- 学习要点

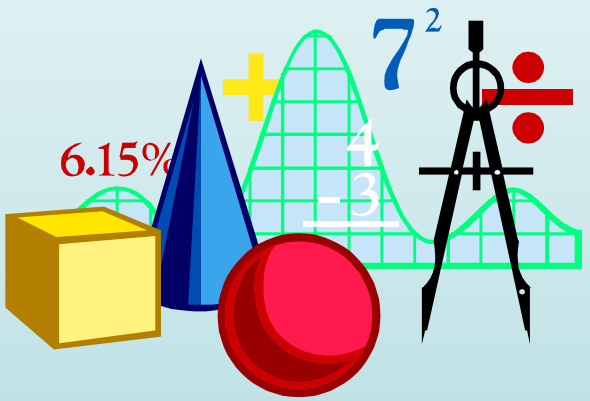
  (1) 掌握算法的基本概念和基本特征

  (2) 掌握常用的几种算法的思想，例如：列举法，递推法，递归法和减半递推。

  (3) 算法复杂度的概念和意义（时间复杂度与空间复杂度）。

算法

- 算法：解题方案的准确而完整地描述；
- 算法不等于程序；程序可作为算法的一种描述，编写程序时要受到计算机系统运行环境限制；
- 通常，程序的编制不可能优于算法的设计。



1.1 算法基本概念

算法的基本特征：能行性：

算法中描述的操作都是可通过已经实现的基本运算、执行有限次实现的；

包括两个方面：

1) 每个步骤都能实现。

例如不能分母为0等；

2) 执行结果能达到预期的目的。

例子： 单精度 （7位有效数字的加法）

$A=10^{12}$; B=1 ； $C=-10^{12}$

A+B+C ； A+C+B

1.1 算法基本概念

算法的基本特征

确定性：算法中的每一条指令必须有明确的含义，不能有二义性；

有穷性：一个算法必须总是在执行有穷步后结束，且每一步都可在合理的执行时间内完成；

拥有足够情报：算法在有足够的情报时才是有效的。

结论：

所谓算法，是一组严谨地定义运算顺序的规则，并且每一个规则都是有效的、明确的，此顺序将在有限的次数下终止。

1.1.2 算法的基本要素

要素1：对数据对象的运算和操作

（1）算术运算：主要包括加、减、乘、除等运算。

（2）逻辑运算：主要包括“与”、“或”、“非”等运算。

（3）关系运算：主要包括“大于”、“小于”、“等于”、“不等于”等运算。

（4）数据传输：主要包括赋值、输入、输出等操作。

计算机程序可以作为算法的一种描述，但程序编制时需要考虑很多细节问题（与算法无关），因此算法设计开始用别的描述工具（流程图等）。

1.1.2 算法的基本要素

要素2：算法的控制结构

- 算法中各操作之间的执行顺序称为算法的控制结构；
- 算法的控制结构给出了算法的基本框架;
- 算法一般都可以用顺序、选择、循环三种基本控制结构组合而成 。

算法的描述方式（常用的）：

算法描述

- 自然语言
- 流程图　特定的表示算法的图形符号
- 伪语言　包括程序设计语言的三大基本结构及自然语言的一种语言
- 类语言　类似高级语言的语言，例如，类PASCAL，类c语言

1.2 算法设计基本方法

- 列举法
- 归纳法
- 递推
- 递归
- 减半递推技术
- 回溯法

1.列举法

- 基本思想

根据提出的问题，列举所有可能的情况，并用问题中给定的条件检验哪些是需要的，哪些是不需要的。因此，列举法常用于解决“是否存在”或“有多少种可能”等类型的问题，例如求解不定方程的问题。

例题：设每只母鸡值3元，每只公鸡值2元，每只小鸡值0.5元。现要用100元钱买100只鸡，设计买鸡方案，这就是经典的求解百鸡问题。

```
int main（）
{int i,j,k;
for (i=0;i<=100;i++)
for (j=0;j<=100;j++)
for (k=0;k<=100;k++)
   {m=i+j+k; n=3*i+2*j+0.5*k;
     if( (m==100.0) &&(n==100))
        printf(“i=%d,j=%d,k=%d \n”,i,j,k);
    }
   return 0;
}
```

总循环次数为101^3＝1030301

```
int main（）
{int i,j,k;
for (i=0;i<=100;i++)
for (j=0;j<=100;j++)
for (k=0;k<=100;k++)
   {m=i+j+k; n=3*i+2*j+0.5*k;
     if( (m==100.0) &&(n==100))
      printf(“i=%d,j=%d,k=%d \n”,i,j,k);
    }
   return 0;
}
```

如何改进？

总循环次数为$101^3=1030301$

- ✓ 首先，考虑到母鸡为3元一只，因此，母鸡最多只能买33以，即算法中的外循环没有必要从0到100．而只需要从0到33就可以了。
- ✓ 其次，考虑到公鸡为2元一只，因此，公鸡最多只能头50只。又考虑到对公鸡的列举是在算法的第二层循环中，此时已经买了I只母鸡，且买一只母鸡的价钱相当于买1.5只公鸡。因此，由第一层循环已经确定买I只母鸡的前提下，公鸡最多只能买50-1.5I只，即第二层对J的循环只需从0到50-1.5I就可以了。
- ✓ 最后，考虑到买的总鸡数为100，而由第一层循环已确定买I只母鸡，由第二层循环已确定买J只公鸡，因此，买小鸡的数量只能是K=100-I-J，即第三层循环已经没有必要了。

列举法---改进后的算法（C）

```
int main（）
{int i,j,k;
for (i=0;i<=33;i++)
for (j=0;j<=50-1.5*i;j++)
   {k=100-i-j;
     if (3*i+2*j+0.5*k==100.0)
     printf(“i=%d,j=%d,k=%d \n”,i,j,k);;
    }
   return 0;
}
```

课堂练习

- 兔子和鸡问题：
- 共有动物35只 有脚90只
- 有多少兔子多少鸡？

2.归纳法

- 归纳法的基本思想

  通过列举少量的特殊情况，经过分析，最后找出一般的关系。

- 归纳是一种抽象

  即从特殊现象中找出一般关系。但由于在归纳的过程中不可能对所有的情况进行列举，因此，最后由归纳得到的结论还只是一种猜测，还需要对这种猜测加以必要的证明。

3.递推法

- 基本思想

  从已知的初始条件出发，逐次推出所要求的各中间结果和最后结果。其中初始条件或是问题本身已经给定，或是通过对问题的分析与化简而得到确定。

- 例题 $I_n = \int_0^1 \frac{x^n}{x+5} dx, \qquad n = 0,1,2,\cdots,20$

递推法例题分析

发现相邻两个积分之间存在以下关系:

$$I_n + 5I_{n-1} = \frac{1}{n}$$

这个关系就可以得到如下递推公式

$$I_n = \frac{1}{n} - 5I_{n-1}$$

只要知道I_{n-1} 就可以算出I_n，也就是说，只要知道了I_0，就可以通过这个递推公式计算出所有的积分值I_n（n=1，2，…，20）。

递推法例题分析

$$I_0 = \int_0^1 \frac{1}{x+5}\,dx = \ln 6 - \ln 5 = \ln(6/5) \approx 0.182322$$

$$\begin{cases} I_0 = 0.182322 \\ I_n = \dfrac{1}{n} - 5I_{n-1}, \quad n = 1,2,\cdots,20 \end{cases}$$

递推法

根据递推关系式可以改成另一个递推公式：

$$I_{n-1}=\frac{1}{5n}-\frac{1}{5}I_n$$

递推关系如下：

$$\begin{cases} I_{30}=1 \\ I_{n-1}=\dfrac{1}{5n}-\dfrac{1}{5}I_n, \quad n=30,29,\cdots,1 \end{cases}$$

递推法总结:
例题分析--改进前后

- 在老递推算法中，初值是近似的，即实际上存在一个误差；老递推算法递推过程中，当计算到I_{20}时，其误差是初值误差的 5^{20}倍。
- 在新递推算法中，虽然初值 I_{30}是近似的，而且误差可能很大。但在用新递推算法递推过程中，每递推计算一次，后一个计算值的误差是前一个计算值误差的**1/5**，因此，当计算到I_{20}时，其误差是初值I_{30}误差的$1/5^{10}$ 。

4.递归法

基本思想

（1）为了降低问题的复杂程度，总是将问题逐层分解，最后归结为一些最简单的问题。

（2）这种将问题逐层分解的过程，实际上并没有对问题进行求解，而只是当解决了最后那些最简单的问题后，再沿着原来分解的逆过程逐步进行综合，这就是递归的基本思想。

（3）因此，递归的基础也是归纳。

递归法举例说明

例：编写一个过程，对于输入的参数n，依次打印输　出自然数1到n。

非递归算法。

```
void wrt(int n)
{
 int k;
 for(k=1;k<=n;k++)
 printf("k=%d \n",k);
 return;
}
```

递归算法。

```
void wrt1(int n)
{
 if(n!=0)
  { wrt1(n-1);
   printf("n=%d \n",n);
  }
 return;
}
```

总结：

- 递归是构造计算机算法的一种基本方法。如果一个过程直接或间接地调用它自身，则称该过程是递归的，递归过程必须有一个递归终止条件，即存在“递归出口”。无条件的递归是毫无意义的。
- 递归分为直接递归与间接递归两种。如果一个算法P显式地调用自己则称为直接递归，例如算法1.4是一个直接递归的算法。如果算法P调用另一个算法Q，而算法Q又调用算法P，则称为间接递归调用。
- 递归过程能将一个复杂的问题归结为若干个较简单的问题，然后将这些较简单的每一个问题再归结为更简单的问题，这个过程可以一直做下去，直到最简单的问题为止。
- 递归与递推是既有区别又有联系的两个概念。递推是从已知的初始条件出发，逐次递推出最后所求的值。递归则是从需求的函数本身出发，逐次上溯调用其本身求解过程，直到递归的出口，然后再从里向外倒推回来，得到最终的值。一般说来，一个递推算法总可以转换为一个递归算法。

5.减半递推技术

- 举例说明：两个n阶矩阵相乘，通常需要作n^3次乘法。那么两个二阶矩阵相乘需要作8次乘法。

$$A=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad B=\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

其乘积矩阵C＝AB为：

$$C=\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}=\begin{bmatrix} a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} \\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22} \end{bmatrix}$$

减半递推技术举例

对于低阶的矩阵相乘问题，如二阶矩阵相乘，减少乘法次数是有可能的。令：

$$
\begin{cases}
x_1 = (a_{11} + a_{22})(b_{11} + b_{22}) \\
x_2 = (a_{21} + a_{22})b_{11} \\
x_3 = a_{11}(b_{12} - b_{22}) \\
x_4 = a_{22}(b_{21} - b_{11}) \\
x_5 = (a_{11} + a_{12})b_{22} \\
x_6 = (a_{21} - a_{11})(b_{11} + b_{12}) \\
x_7 = (a_{12} - a_{22})(b_{21} + b_{22})
\end{cases}
$$

减半递推技术举例

乘积矩阵C中的各元素可用以上7个量的线性组合来表示：

$$\begin{cases} c_{11} = x_1 + x_4 - x_5 + x_7 \\ c_{12} = x_3 + x_5 \\ c_{21} = x_2 + x_4 \\ c_{22} = x_1 + x_3 - x_2 + x_6 \end{cases}$$

上述方法计算两个二阶矩阵相乘只需要7次乘法就够了，比通常的方法减少了1次乘法。以此类推，最后可以归结为计算一阶矩阵相乘的问题，而一阶矩阵相乘只需要一次乘法。

减半递推技术
例题分析

根据以上分析，假设 $n=2^k$，且 $n=2^k$ 阶矩阵相乘所需要的乘法次数为**M（k）**，则有

$$M(k)=7M(k-1)=7^2M(k-2)=\cdots=7^kM(0)$$

因此有 $M(k)=7^k=7^{\log_2 n}=n^{\log_2 7}\approx n^{2.81}$

减半递推技术
总结：

- 对问题分而治之。工程上常用的分治法是减半递推技术。这个技术在快速算法的研究中有很重要的实用价值。所谓“减半”，是指将问题的规模减半，而问题的性质不变，所谓“递推”，是指重复“减半”的过程。

例题

设方程f(x)=0在区间[a,b]上有实根，且f(a)与f(b)异号。利用二分法求该方程在区间[a,b]上的一个实根。

分析：

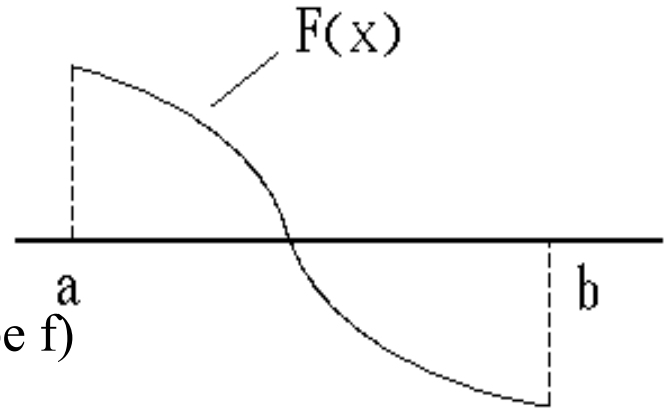
- 首先取给定区间的中点c=(a+b)/2。
- 然后判断f(c)是否为0。若f(c)=0，则说明c即为所求的根，求解过程结束；如果f(c) ≠0 ，则根据以下原则将原区间减半：

  若f(a)f(c)<0，则取原区间的前半部分；

  若f(b)f(c)<0 ，则取原区间的后半部分。
- 最后判断减半后的区间长度是否已经很小。

  若|a-b|<ε，则过程结束，取（a+b）/2为根的近似值；否则，则重复上述的减半过程。

```
double fun1(double x)
{
        return x*x-2;
}
typedef double (*FuncType)(double );
double root(double a, double b, double eps, FuncType f)
{
    double f0=f(a),f1, c;
   do
     {
        c=(a+b)/2; f1=(*f)(c);
        if (f1==0)
        { return c;}
        if (f0*f1>0) a=c;
        else  b=c;
     }
     while(fabs(a-b)<=eps);
     c=(a+b)/2;
     return(c);
}
```



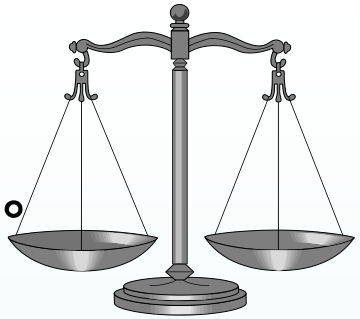
6.回溯法

基本思想

1. 在工程上，有些实际问题却很难归纳出一组简单的递推公式或直观的求解步骤，并且也不能进行无限的列举。
2. 一种有效的方法是“试”。通过对问题的分析，找出一个解决问题的线索；然后沿着这个线索逐步试探。
3. 对于每一步试探，若试探成功，就得到问题的解；若试探失败，就逐步回退，换别的路线再进行试探。这种方法称为回溯法。

例：n皇后问题（n=4）(参考教材P9)

1.4 算法的复杂度分析



算法评价的标准：时间复杂度和空间复杂度。

时间复杂度

指在计算机上运行该算法所花费的时间。常以基本运算的重复次数来衡量。用“O（数量级）”来表示，称为“阶”。常见的时间复杂度有：

$O(1)$，$O(\log n)$，$O(n)$，$O(n^2)$，$O(2^n)$

常数阶　对数阶　线性阶　平方阶　指数阶

空间复杂度

指算法在计算机上运行所占用的存储空间的大小。

（程序、输入、额外辅助）

平均性态

- 所谓平均性态分析，是指用各种特定输入下的基本运算次数的带权平均值来度量算法的工作量。
- 设**x**是所有可能输入中的某个特定输入，**p(x)**是**x**出现的概率（即输入为**x**的概率），**t(x)**是算法在输入为**x**时所执行的基本运算次数，则算法的平均性态定义为：

$$A(n) = \sum_{x \in D_n} p(x)t(x)$$

最坏情况复杂性

- 所谓最坏情况分析，是指在规模为n时，算法所执行的基本运算的最大次数。
- 它定义为：$$W(n)=\max_{x\in D_n}\{t(x)\}$$

例题

例：采用顺序搜索法，在长度为**n**的一维数组中查找值为**x**的元素。即从数组的第一个元素开始，逐个与被查值**x**进行比较。基本运算为**x**与数组元素的比较。

- 平均性态分析

设被查项**x**在数组中的概率为**q**

$$t_i = \begin{cases} i & (1 \le i \le n) \\ n & (i = n+1) \end{cases}$$

$$p_i = \begin{cases} q/n & (1 \le i \le n) \\ 1-q & (i = n+1) \end{cases}$$

$$A(n) = \sum_{i=1}^{n+1} p_i t_i = \sum_{i=1}^{n} (q/n)i + (1-q)n = (n+1)q/2 + (1-q)n$$

- 如果已知需要查找的**x**一定在数组中，此时**q＝1**，则**A(n)=（n＋I）／2。**
- 如果已知需要查找的**x**有一半的机会在数组中，此时**q＝1／2**；则

A(n)=[(n+1)/4]+n/2≈3n/4

最坏情况分析

- 在这个例子中，最坏情况发生在需要查找的**x**是数组中的最后一个元素，或**x**不在数组中的时候，此时显然有：

$$W(n) = \max\{t_i \mid 1 \le i \le n+1\} = n$$

例 说明下列各个程序段的时间复杂度。

① /*交换a和b的内容*/

```
t=a; a=b; b=t;
```

② /*求n以内所有2的幂次数的和，即$1+2^1+2^2+\ldots+2^k$，$2^k \le n$ */

```
sum=0;
for(i=1;i<=n;i*=2)  sum+=i;
```

说明：在程序段①中，三条语句的执行次数均为1，与规模n无关，故可知其时间复杂度为O(1)。

在程序段②中，执行次数最多的语句是循环体**sum+=i**，它执行的次数未知，显然不是**n**次，若设为**k**次，由于$2^k \leq n$，所以有$k \leq \log_2 n$故时间复杂度为$O(\log_2 n)$。

算法的最优性

1. 衡量算法的好坏主要依据算法的复杂度，特别是时间复杂度。通常总是在最坏的情况下分析算法的工作量。
2. 最优算法是指在解决一个问题时，如果在被研究的算法类中，没有一个算法比现有算法执行更少的基本运算，则称此算法是最优的。

总结：

算法的基本概念

算法的基本特征

算法的要素

算法描述语言

算法设计基本方法

算法复杂度分析

作业

- 给定三个整数 a,b,c 写出寻找其中数的算法，并分析平均和最坏情况下要作多少次比较？

作业

四位同学（编号为1,2,3,4）在晚上宿舍过生日聚会，突然停电了5分钟。来电后，发现生日蛋糕被一个同学偷吃了一口。

同学1说：“不是我”；

同学2说：“是3吃的”；

同学3说：“是4吃的”；

同学4说：“3胡说”。

已知这四位同学有三位同学说真话，一位同学说假话，请用列举法编程解决该问题，并写出运行结果。