算法设计与分析

授课教师：王若梅教授
邮箱：isswrm@mail.sysu.edu.cn

2015.09-2016.01

重要邮箱

PPT课件：
homework_algorithm@163.com
Password： homework

平时作业：
homework_sysu@163.com

作业要求

1. 学号+姓名
2. E-mail's subject: 算法分析作业之XXX
3. Algorithm analysis and notes
4. Main functions, ADT used in algorithm and parameters

授 课 安 排

- 课堂讲授：关于算法设计和分析的理论方法和实际应用；
- 学生实践：课下进行相关算法的设计和实现；
- 实践交流：以小组为单位进行实践交流，内容包括：
  1）具体问题的算法设计思路及其求解；
  2）专题问题的研究与理解。

考试：闭卷考试50%
实践交流及作业50%

参 考 书

1. 郑宗汉等, 算法设计与分析, 清华大学出版社
2. 王晓东等，计算机算法设计与分析，电子工业出版社
3. Thomas H.Cormen;Charles E.Leiserson;Ronald L.Rivest;Clifford Stein. Introduction to Algorithms, 2th Ed. The MIT Press, 2001, ISBN 978-0-262-33293-3. 影印版：《算法导论》（第二版），北京：高等教育出版社，2007，ISBN 978-7-040-11050-0. 中译版：潘金贵等译，《算法导论》（第2版），北京：机械工业出版社，2006，ISBN 7-111-18777-6
4. 郭嵩山、李志业、金涛、梁锋.《国际大学生程序设计竞赛例题解（一）数论、计算几何、搜索算法专集》. 电子工业出版社, 2006.5

第一章 算法分析介绍

一、 用计算机求解问题的步骤

- 问题分析： 准确完整的描述和理解问题
- 数学模型的建立
  最适合此问题的数学模型？是否存在可用借鉴的？
- ***算法***设计与选择
  建立求解问题的算法，判断最适合求解问题的算法是什么？

7

算法：

广义：在解决问题时，按照某种机械步骤一定可以得到问题结果（有解时给出问题的解，无解时给出无解的结论）的处理过程。

狭义：用计算机解决问题的方法和步骤的描述。

8

例1.1：给出求1+2+3+4+5的一个算法。

算法1　按照逐一相加的程序进行。

第一步　计算1+2, 得到3；

第二步　将第一步中的运算结果3与3相加, 得到6；

第三步　将第二步中的运算结果6与4相加, 得到10；

第四步　将第三步中的运算结果10与5相加, 得到15。

9

算法2　可以运用公式

$$1+2+3+\cdots+n=\frac{n(n+1)}{2}$$ 直接计算；

第一步　取n=5；

第二步　计算 $\frac{n(n+1)}{2}$

第三步　输出运算结果。

10

例1.2：三个牧师和三个野人过河，只有一条能装下两人的船，在河的任一边或者船上，若野人人数大于牧师人数，那么牧师就会有被吃掉的危险。你能不能找出一种安全的渡河算法呢？

第一步　两个野人先过河，一个野人回来；

第二步　再两个野人过河，一个野人回来；

第三步　两个牧师过河，一个野人和一个牧师回来；

第四步　两个牧师过河，一个野人回来；

第五步　两个野人过河，一个野人回来；

第六步　两个野人过河。

11

二、 学习算法设计的意义

- 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

20 世纪最伟大的科学技术发明---计算机；

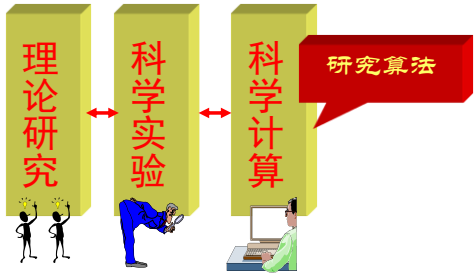
计算机是对人脑的模拟，它强化了人的思维；

没有软件的支持，超级计算机只是一堆废铁而已。

软件的核心就是算法 ！

程序＝数据结构＋算法

现代科学研究的三大支柱



21世纪信息社会的两个主要特征：

“计算机无处不在”

“数学无处不在”

21世纪信息社会对科技人才的要求：

--会“用数学”解决实际问题。

--会用计算机进行科学计算。

与其他课程的关系

数 据 结 构 | *算法设计与分析*

高级程序设计语言（C, C++）

与数据结构的区别：

- 考虑问题的角度：数据结构关心不同的数据结构在解题中的作用和效率；算法关心不同设计技术的适用性和效率。
- 考虑问题的高度：数据结构关心的是解具体问题，算法不仅如此，它提供一种解决问题的通用方法。

学习目标：“用计算机求解问题”

一般计算机对现实问题无能为力，需要人类对问题抽象化、形式化后才能去机械的执行。

- 问题分析：准确、完整地理解和描述问题
- 数学模型建立
- 算法设计与选择：创造性的活动
- 算法表示：思想的表示形式
- 算法分析：算法时空特性分析
- 算法实现
- 程序调试：测试
- 结果整理文档编制

三、算法及其表示方法

算法：是对特定问题求解步骤的一种描述。算法是指令的有限序列，其中每一条指令表示一个或多个操作。

1. 算法的表示

- 自然语言，流程图，程序设计语言、伪代码，

19

例1.3：欧几里德算法—自然语言描述算法

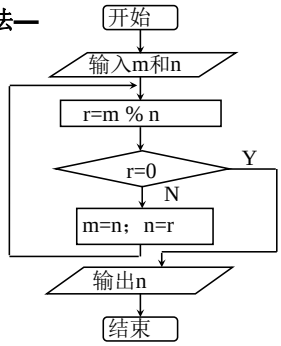
自然语言

① 输入m和n；
② 求m除以n的余数r；
③ 若r等于0，则n为最大公约数，算法结束；否则执行第④步；
④ 将n的值放在m中，将r的值放在n中；
⑤ 重新执行第②步。

20

例1.3欧几里德算法—流程图描述算法

流程图



21

例1.3 欧几里德算法—程序设计语言描述算法

程序设计语言

```
#include <iostream.h>
int CommonFactor(int m, int n)
{
   int r=m % n;
   while (r!=0)
   {
     m=n;
     n=r;
     r=m % n;
   }
   return n;
}
void main( )
{
  cout<<CommonFactor(63, 54)<<endl;
}
```

22

例1.3 欧几里德算法--伪代码

伪代码

1. r = m % n;
2. 循环直到 r 等于0
   2.1 m = n;
   2.2 n = r;
   2.3 r = m % n;
3. 输出 n ;

上述伪代码再具体一些，用C++的函数来描述-----》程序。

23

3. 算法基本技巧举例

- a. 算术运算的妙用
  - 例1.5 开灯问题
- b. 巧用“标志量”
  - 例1.6 判定输入n个数据互不相等
  - 例1.7 冒泡排序
  - 例1.8 三角形判断问题
  - 例1.9 最小公倍数
- c. 信息数字化
  - 例1.10 警察抓小偷
  - 例1.11 竞赛预测
- d. 学会找规律
  - 例1.12 数组移位

24

a. 算术运算的妙用-例1.5开灯 *问题*

- **算术运算：加减乘除。**

例1.5 开灯问题

从前，有从1到*n*依次编号的*n*个同学和*n* 盏灯。

1号同学将所有的灯都关掉；

2号同学将编号为2的倍数的灯都打开；

3号同学则将编号为3的倍数的灯作相反处理（该号灯如打开的，则关掉；如关闭的，则打开）；

以后的同学都将自己编号的倍数的灯，作相反处理。

问：经n个同学操作后，哪些灯是打开的？

a. 算术运算的妙用-例1.5问题 *分析/建模*

- **问题分析：**
- **1）用数组表示某种状态，这里定义有n个元素的a数组，它的每个下标变量a[i]视为一灯，i表示其编号。a[i]=1表示灯处于打开状态，a[i]=0表示灯处于关闭状态。**

2）实现将第i 灯作相反处理的操作：

- 条件语句if表示：
  if a[i] == 1 a[i] = 0；
  if a[i] == 0 a[i] = 1；
- 通过算术运算更简练、逼真：
  a[i]=1-a[i]。

此用法也称为“乒乓开关”。简化逻辑表达式、减少条件判断

a. 算术运算的妙用-例1.5-*算法设计*

- **int a[1000];**

输入n的数值；

关闭所有灯，即a[1]~a[n]置为0；

第2个学生->第n个学生(学生i)进行操作：

操作对象：数组a，灯编号含因数i，即a[i*k] ；

操作： a[i*k] = 1 - a[i*k] ；

输出灯的开关状态。

b. 巧用“标志量”-例1.6-*问题分析*

- **例1.6 判定输入*n*个数据互不相等。**

问题分析/算法设计：

- 完全比较一遍需要*n*-1 + *n*-2 + *n*-3 +……+次比较。
- 双重循环；1=*n**(*n*-1)/2
- 通过引入标志量记录数据是否有重复的情况，相当于整合每趟循环中的比较结果。

b. 巧用“标志量”-例1.6-*算法设计*

- **建立一个充分大的数组；**

标志量flag=1；

输入*n*个数，保存在数组的前*n*个元素；

从第1个—>第*n*-1个(每个元素*i*)操作

与第*i*+1—>第*n*元素(每个元素*j*)比较，若相等则标志量置0，循环中断；

若flag=1，则无重复；反之，有重复。

例1.7冒泡排序

❖ 排序过程

初始关键字	第一趟排序	第二趟排序	第三趟排序	第四趟排序	第五趟排序	第六趟排序
49	38	38	38	38	13	13
38	49	49	49	13	27	27
65	65	65	13	27	38	38
97	76	13	27	49	49	
76	13	27	49	49		
13	27	49	65			
27	49	76				
49	97					

```
Void BubbleSort(SqList &L) {
  i = n ;
  wh k = 1 ;
     for ( j = 1; j < i ; j + +)
        if ( r[ j +1] < r[ j ]) {
           r[ j ] ←→ r[ j + 1 ] ;
           k = j ; //交换的位置
        }
     i = k ;
  } // while
}
```

冒泡排序算法

例1.8 输入3个数值，判断以它们为边长是否能构成三角形。如能构成，则判断属于哪种特殊三角形：等边、等腰或直角 .

- 问题分析：可能的输出情况
- 1）不构成三角形
- 2）构成等边三角形
- 3）构成等腰三角形
- 4）构成直角三角形
- 5）三角形不属于2), 3), 4)三种情况
- 其中3),4)可能同时输出，而其它几种同时输入就不合理了。
- 情况1）与其它情况互斥容易区分
- 情况5）是在三角形不属于2), 3), 4)三种情况时的输出。

例1.8 输入3个数值，判断以它们为边长是否能构成三角形。如能构成，则判断属于哪种特殊三角形：等边、等腰或直角

- 算法设计：算法中需要避免情况5）与情况2），3），4）之一同时输出。设置一标志量变量flag，当数据能构成三角形时就置flag为0表示情况5），一旦测试出数据属于2），3），4）中的一种，就置flag=1表示构成了特殊三角形，最后就不必输出“构成一般三角形”了；若flag最后仍为0，则输出“构成一般三角形”。

```
Main()
{  int a,b,c,flag;
 input(a,b,c);
 if (a>=b+c or b>=a+c or c>=a+b)
    printf("构不成三角形");
 else {
   flag=0;
   if (a*a=b*b+c*c or b*b=a*a+c*c or c*c=a*a+b*b) {
     printf("构成直角三角形");
     flag=1;
   }
   if (a=b and b=c) {
     printf("构成等边三角形");
     flag=1;
   }
   else if (a==b or b==c or c==a){
     printf("构成等腰三角形");
     flag=1;
   }
   if (flag==0)
     printf("构成三角形");
 }
}
```


例1.9 编写算法，求任意3个数的最小公倍数

- 算法设计：
- 1）用短除法求3个已知数的最小公倍数的过程就是求它们的因数之积，这个因数可能是3个数共有的、2个数共有或1个数独有的3种情况。
- 2）为了避免因数重复计算，每次都需要除掉3个整数中已找到的因数（即用因数去除含有它的整数）。因此需要记录i具体是哪一个数的因数，要对那个数进行整除。
- 例如：2是2，5，6中2，6两个数的因数，因此要用2，6去除以2得到新的一组数1，5，3再进行重复求解。

c. 信息数字化-例1.10 警察抓小偷 *问题*

- **一些表面上看是非数值的问题，但经过进行数字化后，就可方便进行算法设计。**

- 例1.10 警察抓小偷

  警察局抓了a，b，c，d四名偷窃嫌疑犯，其中只有一人是小偷。审问中

  a说：“我不是小偷。”

  b说：“c是小偷。”

  c说：“小偷肯定是d。”

  d说：“c在冤枉人。”

  现在已经知道四个人中三人说的是真话，一人说的是假话，问到底谁是小偷？

c. 信息数字化-例1.10 - *问题分析*

- **问题分析：可通过循环，每次假设一名嫌疑犯为小偷，代入问题系统，检验是否只有一句假话。**
- 这种让所有可能解都进行检验，以求得真解的方法称为“枚举尝试法”，也是“蛮力法”、“暴力法”。
- 为方便设计程序，将a,b,c,d将四个人进行编号，号码分别为1，2，3，4。

c. 信息数字化-例1.10 - *算法设计*

- 算法设计：用变量x存放小偷的编号，则x的取值范围从1取到4，就假设了他们中的某人是小偷的所有情况。四个人所说的话就可以分别写成：
  - a说的话：x<>1
  - b说的话：x=3
  - c说的话：x=4
  - d说的话：x<>4或not(x=4)
- 注意:在x的枚举过程中，当这四个逻辑式的值相加等于3时，即表示“四个人中三人说的是真话，一人说的是假话”。

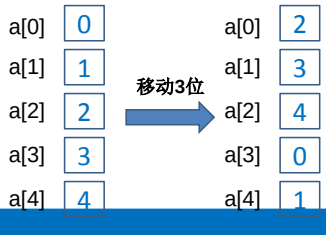
  if ((x!=1)+(x==3)+(x==4)+(x!=4)==3)

例1.11 3位老师对某次学生竞赛进行了预测，他们的预测如下：
甲说：学生A得第一名，学生B得第三名。
乙说：学生C得第一名，学生D得第四名。
丙说：学生D得第二名，学生A得第三名。
竞赛结果表明，它们各说对了一半，说错了一半，并且无名次并列，试编程输入a,b,c,d各自的名次

- 问题分析：用1,2,3,4分别代表学生a,b,c,d获得的名次，问题就可以利用三重循环把所有的情况枚举出来。
- 算法设计：
- 1）用a,b,c,d代表4个同学，其存储的值代表他们的名次。计算3重循环，d=10-a-b-c;
- 2）问题的已知内容可以表达为：
- (a==1)+(b==3)==1
- (c==1)+(d==4)==1
- (d==2)+(a==3)==1

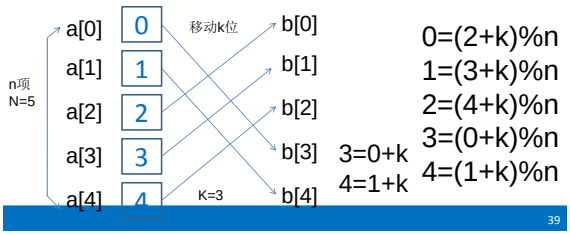
d 学会找规律-例1.12 数组移位

- **例1.12 数组中有n个数据，要将它们顺序循环向后移k位，即前面的元素向后移k位，后面的元素则循环向前移k位，例：0、1、2、3、4循环移3位后为：2、3、4、0、1。考虑到n会很大，不允许用2*n以上个空间来完成此题。**

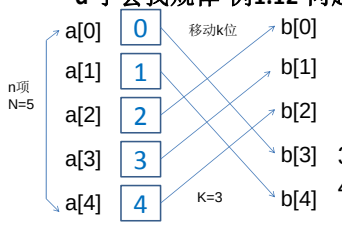


d 学会找规律-例1.12-问题分析(思路1)

- **问题分析1：若题目中没有关于存储空间的限制，我们可以方便地开辟两个一维数组，一个存储原始数据，另一个存储移动后的数据。**
- 开始部分的元素从前向后移，容易确定；但数组中后k个元素是从后向前移，如何确定？

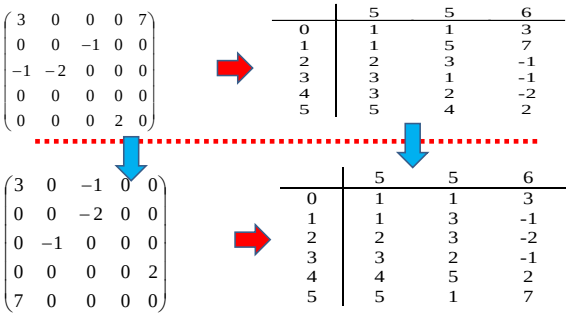


d 学会找规律-例1.12-问题分析(思路1)




```
void alg1(){
    int a[100],b[100],i,n,k;
    scanf("%d,%d",&n,&k);
    for (i=0;i<n;i=i+1)
        scanf("%d",&a[i]);
    for (i=0;i<n;i=i+1)
        b[(k+i) % n] = a[i];
    for (i=0;i<n;i=i+1)
        printf("%d,",b[i]);
    printf("\n");
}
```


例题1.13 稀疏矩阵的转置（三元组存储）



下面以矩阵的转置为例，说明在这种压缩存储结构上如何实现矩阵的运算。

一个$m\times n$的矩阵A，它的转置B是一个$n\times m$的矩阵，且a[i][j]=b[j][i]，$0\leqq i\leqq m$，$0\leqq j\leqq n$，即A的行是B的列，A的列是B的行。

由于A的列是B的行，因此，按a.data的列序转置，所得到的转置矩阵B的三元组表b.data必定是按行优先存放的。

转置算法

```
Void TransMatrix(TSMatrix M,TSMatrix T)
//M 和T 是矩阵的三元组表示, T 是转置矩阵
{T.mu=M.nu; T.nu=M.mu;T.tu=M.tu;
 if (T.tu)
  {  q=1;
     for (col=1;col<=M.nu;col++)
       for (p=1;p<=M.tu;p++)
        {  if (M.data[p].j==col)
             {  T.data[q].i=M.data[p].j;
                T.data[q].j=M.data[p].i;
                T.data[q].e=M.data[p].e;
                q=q+1;  }
          }
  }
}
```

表示三元组T的下标

按M的列值由小到大查找

分析这个算法，主要的工作是在p和col的两个循环中完成的，故算法的时间复杂度为$O(n*t)$，即矩阵的列数和非零元的个数的乘积成正比。

为了预先确定矩阵M中的每一列的第一个非零元素在数组B中应有的位置，需要先求得矩阵M中的每一列中非零元素的个数。因为：矩阵M中第一列的第一个非零元素在数组B中应有的位置等于前一列第一个非零元素的位置加上前列非零元素的个数。

为此，需要设置两个一维数组num[1..n]和cpot[1..n]

num[1..n]：统计M中每列非零元素的个数，num[col]的值可以由A的第二列求得。

cpot[1..n]：由递推关系得出M中的每列第一个非零元素在B中的位置。

算法通过cpot数组建立位置对应关系：

cpot[1]=1

cpot[col]=cpot[col-1]+num[col-1]

2<=cpl<=a.n

四、 算法分析

1. 算法分析的目的

通过对算法分析，在把算法变成程序实际运行前，就知道为完成一项任务所设计的算法的好坏，从而运行好算法，改进差算法，避免无益的人力和物力浪费。

47

2.算法复杂性

时间复杂性和空间复杂性

为什么要考虑时间复杂性？

1.有些计算机需要用户提供程序运行时间的上限，一旦达到这个上限，程序将被强制结束。

2.正在开发的程序可能需要提供一个满意的实时响应。

为什么要考虑空间复杂性？

1. 多用户系统中运行时，需指明分配给该程序的内存大小。
2. 可提前知道是否有足够可用的内存来运行该程序。
3. 一个问题可能有若干个内存需求各不相同的解决方案，从中择取。
4. 利用空间复杂性来估算一个程序所能解决问题的最大规模。

3. 如何进行算法分析？

•**事前分析**：就算法本身，通过对其执行性能的理论分析，得出关于算法特性——**时间和空间**的一个特征函数（O、Ω）——与计算机软硬件没有直接关系。

•**事后测试**：将算法编制成程序后放到计算机上运行，收集其执行时间和空间占用等统计资料，进行分析判断——直接与物理实现有关。

50

4、空间复杂性(Space Complexity)

- **程序p的空间复杂度指程序运行时所需的内存空间大小和实例特征的函数关系.**
- 程序运行时所需空间包括:
  - 指令空间：与实例特征无关的常数
  - 数据空间：
    - 常量和简单变量-实例无关
    - 复合变量(数组、链表、树和图等)--问题实例特征有关
  - 环境栈空间(函数调用)-是否递归?
    - 非递归：实例特征无关
    - 递归：实例特征相关

51

5、时间复杂性(time complexity)

- 时间复杂性指程序执行时所用的时间+编译时间（实例无关）
- 在解析地分析时间复杂度时,使用以下两种时间单位并计算：
  - 操作计数(operation count):算法的基本操作
  - (程序)步计数(step count):分析全部程序
- 要点:基本操作或程序步的执行时间必须是常数

52

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。

算法渐近复杂性分析中常用函数

- **（1）单调函数**
- 单调递增：$m \le n \Rightarrow f(m) \le f(n)$;
- 单调递减：$m \le n \Rightarrow f(m) \ge f(n)$;
- 严格单调递增：$m < n \Rightarrow f(m) < f(n)$;
- 严格单调递减：$m < n \Rightarrow f(m) > f(n)$.
- **（2）取整函数**
- $\lfloor x \rfloor$：不大于x的最大整数；
- $\lceil x \rceil$：不小于x的最小整数。

54

取整函数的若干性质

- $x-1<\lfloor x\rfloor\le x\le\lceil x\rceil<x+1$;
- $\lfloor n/2\rfloor+\lceil n/2\rceil=n$;
- 对于$n\ge 0$，$a,b>0$，有：
- $\lceil\lceil n/a\rceil/b\rceil=\lceil n/ab\rceil$;
- $\lfloor\lfloor n/a\rfloor/b\rfloor=\lfloor n/ab\rfloor$;
- $\lceil a/b\rceil\le(a+(b-1))/b$;
- $\lfloor a/b\rfloor\ge(a-(b-1))/b$;
- $f(x)=\lfloor x\rfloor$, $g(x)=\lceil x\rceil$为单调递增函数。

- **（3）多项式函数**
- $p(n)=a_0+a_1n+a_2n^2+\ldots+a_dn^d$； $a_d>0$;
- $p(n)=\Theta(n^d)$;
- $f(n)=O(n^k)\Leftrightarrow f(n)$多项式有界；
- $f(n)=O(1)\Leftrightarrow f(n)\le c$;

- **（4）指数函数**
- 对于正整数m,n和实数$a>0$:
- $a^0=1$;
- $a^1=a$;
- $a^{-1}=1/a$;
- $(a^m)^n=a^{mn}$;
- $(a^m)^n=(a^n)^m$;
- $a^ma^n=a^{m+n}$;

$$e^x=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots=\sum_{i=0}^{\infty}\frac{x^i}{i!}$$

- $e^x\ge 1+x$;
- $|x|\le 1\Rightarrow 1+x\le e^x\le 1+x+x^2$;
- $e^x=1+x+\Theta(x^2)$, as $x\to 0$;

$$\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n=e^x$$

- **（5）对数函数**
- $\log n=\log_2 n$;
- $\lg n=\log_{10} n$;
- $\ln n=\log_e n$;
- $\log^k n=(\log n)^k$;
- $\log\log n=\log(\log n)$;
- for a>0,b>0,c>0

$$a=b^{\log_b a}$$

$$\log_c(ab)=\log_c a+\log_c b$$

$$\log_b a^n=n\log_b a$$

$$\log_b a=\frac{\log_c a}{\log_c b}$$

$$\log_b(1/a)=-\log_b a$$

$$\log_b a=\frac{1}{\log_a b}$$

$$a^{\log_b c}=c^{\log_b a}$$

- $|x| \le 1 \Rightarrow \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots$.
- for $x > -1$, $\frac{x}{1+x} \le \ln(1+x) \le x$

- （6）阶乘函数

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

$$n! = 1 \cdot 2 \cdot 3 \cdots n$$

- Stirling's approximation

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n} \qquad \frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$$

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\log(n!) = \Theta(n \log n)$$

执行时间随*n*的增长而增长的情况

n	*n*!	*n*	*n*!	*n*	*n*!	*n*	*n*!
5	120μ s	9	362ms	13	1.72h	17	11.27year
6	720μ s	10	3.62s	14	24h	18	203year
7	5.04ms	11	39.9s	15	15day	19	3857year
8	40.3ms	12	479.0s	16	242day	20	77146year

1.多项式时间复杂度算法 一p类问题
2.指数时间复杂度算法 n! 或者2exp(n)一NP类问题

6. 算法分析实例

一个具体算法的时间复杂度和空间复杂度往往不独立，在算法设计中要在时间效率和空间效率之间折衷。

- **非递归算法分析**

a．仅依赖于问题规模的时间复杂度

有一类简单的问题，其操作具有普遍性，也就是说对所有的数据均等价地进行处理。

【例1.14】交换i和j的内容。

```
Temp=i;
i=j;
j=temp;
```

以上三条单个语句的频度均为1，该算法段的执行时间是一个与问题规模*n*无关的常数。算法的时间复杂度为常数阶，记作$T(n)=O(1)$。

如果算法的执行时间不随着问题规模n的增加而增长，即使算法中有上千条语句，其执行时间也不过是一个较大的常数。此类算法的时间复杂度是O(1)。

【例1.15】循环次数直接依赖规模n一变量计数之一。

```
(1) x=0;y=0;
(2) for(k=1;k<=n;k++)
(3)     x++;
(4) for(i=1;i<=n;i++)
(5)     for(j=1;j<=n;j++)
(6)       y++;
```

该算法段的时间复杂度为$T(n)=O(n^2)$。

当有若干个循环语句时，算法的时间复杂度是由嵌套层数最多的循环语句中最内层语句的频度f(n)决定的。

【例1.16】循环次数间接依赖规模n-变量计数之二。

```
(1) x=1;
(2) for(i=1; i<=n; i++)
(3)   for(j=1; j<=i; j++)
(4)     for(k=1; k<=j; k++)
(5)       x++;
```

该算法段中频度最大的语句是(5)，从内层循环向外层分析语句(5)的执行次数：

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\sum_{i=1}^{n}\sum_{j=1}^{i}j=\sum_{i=1}^{n}i(i+1)/2=[n(n+1)(2n+1)/6+n(n+1)/2]/2$$

则该算法段的时间复杂度为：$T(n)=O(n^3/6+\text{低次项})=O(n^3)$。

b. 算法的时间复杂度与输入实例的初始状态有关。

这类算法的时间复杂度的分析比较复杂，一般分最好情况（处理最少的情况），最坏情况（处理最多的情况）和平均情况分别进行讨论。

【例1.17】在数值A[0..n-1]中查找给定值K：

```
(1) i=n-1;
(2) while( i>=0 and A[i]<>k )
(3)     i=i-1;
(4) return i;
```

此算法的频度不仅与问题规模*n*有关，还与输入实例中*A*的各元素取值及*k*的取值有关：
1. 若*A*中没有与*k*相等的元素，则(2)频度*f*(*n*)=*n*(最坏情况)；
2. 若*A*最后一个元素等于*k*，则(2)频度*f*(*n*)是常数1(最好情况)；

在求平均情况时，一般地假设查找不同元素的概率P是相同的，则算法的平均复杂度为：

$$\sum_{i=n-1}^{0}Pi(n-i)=\frac{1}{n}(1+2+3+\Lambda+n)=\frac{n+1}{2}=O(n)$$

若对于查找不同元素的概率P不相同时，其算法复杂度就只能做近似分析，或在构造更好的算法或存储结构后，做较准确的分析。

小　结

- **理解什么是算法；算法特性；**
- **理解问题求解的步骤及算法在其中的重要地位；**
- **理解算法的计算复杂性概念；**
- **掌握算法复杂度分析的一般方法和数学表示；**
- **学会算法设计基本技巧；**
- **学会对算法进行分析。**