中国科学技术大学 计算机科学与技术系
University of Science and Technology of China
Department of Computer Science and Technology

算法设计与分析

Design and Analysis of Algorithms

主讲人　徐云

Fall 2018, USTC

第1章(补充) 递归与分治法

1.1 递归设计技术

1.2 二分查找

1.3 大整数乘法

1.4 Strassen矩阵乘法

1.5 导线和开关

1.1 递归设计技术

- 递归的概念和种类
- 递归方法的三种应用
- 一个简单示例：n!
- 递归算法的非递归实现
- 递归算法设计举例

递归的概念和种类

- 递归的定义

  若一个对象部分地包含它自己，或用它自己给自己定义，则称这个对象是递归的；若一个过程直接地或间接地调用自己，则称这个过程是递归的过程。

- 递归有两种

  直接递归：自己调用自己

  间接递归：A调用B，B调用A

递归方法的三种应用

- 以下三个方面常用到递归方法

①递归定义

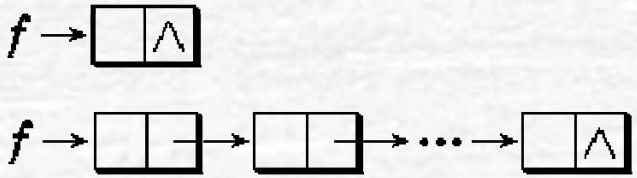
如，自然数定义：

-1是自然数；-一个自然数加1(后继)仍是自然数；

注：“1是自然数”是递归的临界条件

②递归的数据结构

如，单链表节点是递归扩展的



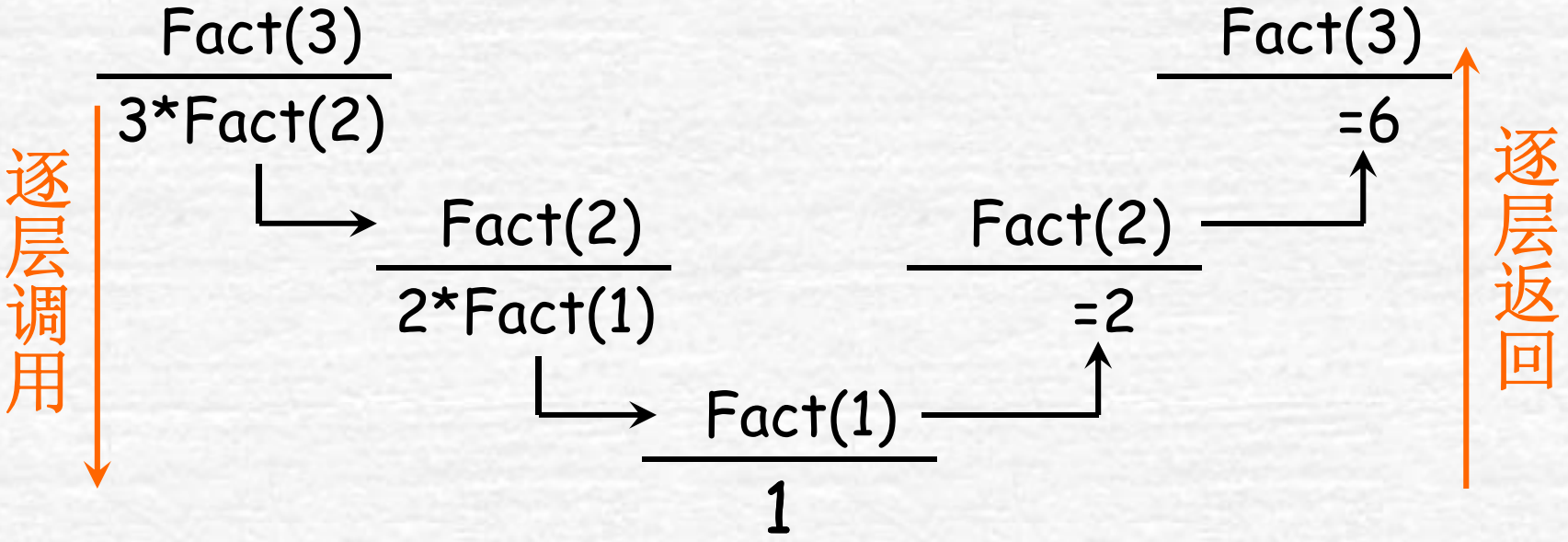
③问题的递归解法

如，汉诺塔问题的直观解法

一个简单示例：n!

- n阶乘的定义：$n!=\begin{cases}1 & n=0\\ n*(n-1)! & n>0\end{cases}$

- 以求3！为例的计算过程



递归算法的非递归实现

- 示例：n!的递归和非递归算法

```
Fact1(n)  //递归程序
{  if n=0 return 1;
   else return n*fact1(n-1);
   }
```

```
Fact2(int n)  //非递归程序
{  p=1;
   for i←1 to n do  p←p*i;
   return p;
   }
```

- Remark：

(1)递归算法易设计和分析，但执行效率较低，常要转化为非递归程序；

(2)递归算法的非递归实现通常有三种实现方法：

①利用栈消除递归　　②利用迭代法消除递归

③末尾递归消除法

递归算法的非递归实现

- 思考题：

Fibonacci数递归和非递归算法

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n \geq 2 \end{cases}$$

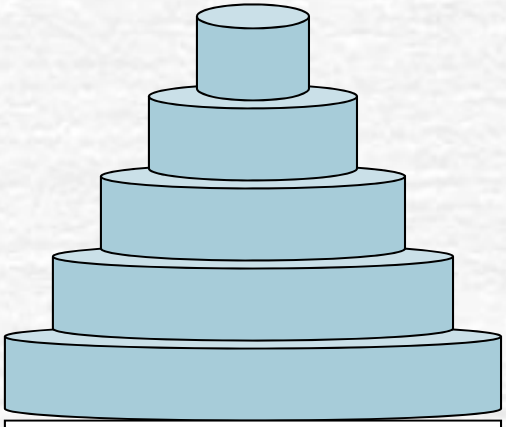
```
Fibonacci(n)
{   //递归算法
   if  n=0 or  n=1 then
       return n;
   else
       return  Fibonacci(n-1) + Fibonacci(n-2) ;
}
```

```
Fibonacci(n)
{   //非递归算法
   if n=0 or n=1  then
       return n;
   s1 = 0; s2 = 1;
   for i←2 to n do
   {
       sum←s1 + s2;
       s1←s2;
       s2←sum;
   }
   return sum;
}
```

递归算法设计举例：汉诺塔（1）

- 汉诺塔（Hanoi Tower）问题：这是一个流传很久的游戏。

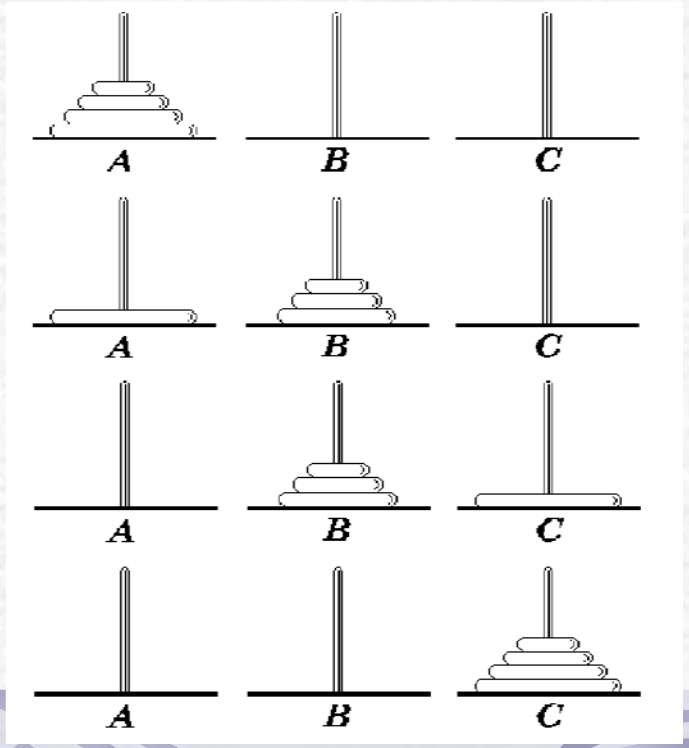
  1. 有三根杆子A,B,C。A杆上有n只碟子
  2. 每次移动一块碟子,小的只能叠在大的上面
  3. 把所有碟子从A杆经C杆全部移到B杆上.



递归算法设计举例：汉诺塔（2）

- 递归求解：

1. 若只有一只碟子，直接将它从A杆移到B杆；
2. 把n-1只碟子从A杆经B杆移动到C杆，将A杆上第n只碟子移到B杆；然后再将n-1只碟子从C杆经A杆移到B杆。



递归算法设计举例：汉诺塔（3）

- 递归算法

```
Hanoi( n, A, B, C)
{ //将从小到大的n个圆盘从A移到B
   if  n>0  then
   {  Hanoi(n-1, A, C, B);  //将从小到大的n-1个圆盘从A移到C
      Move(n, A, B);        //将第n大的圆盘从A移到B
      Hanoi(n-1, C, B, A);  //将从小到大的n-1个圆盘从C移到B
    }
 }
```

- 算法分析
  - 正确性：用归纳法证明；
  - 时间复杂性：$T(n)=1+2T(n-1) \Rightarrow O(2^n)$
  - 算法的最优性：为最优的

递归算法设计举例：再论Fib数（1）

- 基于Fibonacci数递归定义的算法：$T(n)=\Omega((3/2)^n)$
- 改进的算法：利用公式

$$F_n=\begin{cases}0 & n=0\\ 1 & n=1\\ \left(F_{\lceil n/2\rceil}\right)^2+\left(F_{\lceil n/2\rceil-1}\right)^2 & n\geq 2\text{且为奇数}\\ \left(F_{\lceil n/2\rceil}\right)^2+2F_{\lceil n/2\rceil}F_{\lceil n/2\rceil-1} & n\geq 2\text{且为偶数}\end{cases}$$

- 算法：

```
Fibonacci(int n)
{   if n=0 or n=1 then
        return n;
    else {
        a ← Fibonacci((n+1)/2);
        b ← Fibonacci((n+1)/2-1);
        if n mod 2 = 0  then
            return a*(a+2*b);
        else
            return a*a+b*b;
      }
}
```

递归算法设计举例：再论Fib数（2）

- 时间分析：

$$T(n)=\begin{cases} O(1) & n\le 1 \\ 2T(n/2)+O(1) & n>1 \end{cases}$$

应用主方法：

$$f(n)=1<n^{1-\varepsilon}=n^{\log_2^2-\varepsilon} \qquad (\text{这里}\varepsilon=1/2)$$

由$case\,1$: $\quad T(n)=\theta(n)$

递归算法设计举例：生成全排列（1）

- 问题描述：编写一个算法，就地生成字符数组S[1..n]的所有排列，要求算法终止时S[1..n]保持初始状态。就地生成表示不使用S以外的数组。
- 问题分析
  - 分解：

    将原问题分解为n个子问题，

    子问题1：生成n-1个元素s[1],…,s[n-1]的全排列后接s[n];

    子问题2：生成n-1个元素s[1],…,s[n-2],s[n]的全排列后接s[n-1];

    … … …

    子问题n：生成n-1个元素s[2],…,s[n]的全排列后接s[1];
  - 递归求解：

    子问题与原问题的性质相同，只不过规模为n-1。设原问题的求解算法为permute(s,n)，则子问题就是递归调用permute(s,n-1)。临界条件：一个元素的排列就是该元素本身。

递归算法设计举例：生成全排列（2）

- 算法

```
Permute( s[], n )
{ //n个元素s[1..n], 输出所有的排列
   if n=1 then
      print(s);      //输出s的一个排列
   else {
      Permute(s, n-1); //第一个子问题求解，不需要交换s[n]
      for i←n-1 downto 1 do
      {  swap(s[i], s[n]);   //交换s[i], s[n]
         Permute(s, n-1);
         swap(s[i], s[n]);   //交换s[i], s[n], 使s恢复原状
       }
    }
 }
```

递归算法设计举例：生成全排列（3）

- 时间分析

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

$$\Rightarrow \quad T(n)=O(n!)$$

注：该算法是最优的

递归算法设计举例：集合划分（1）

- 问题描述：设有n个元素的集合$S=\{a_1,a_2,\ldots,a_n\}$，要求将S划分为k个子集$S_1,S_2,\ldots,S_k$，且满足：

  ①$S_i\neq\varphi$；　②$S_i\cap S_j=\varphi$ $i\neq j$；　③$S=S_1\cup S_2\cup\ldots\cup S_k$

  求S的k划分数。

- 问题分析
  - 例如：S={1,2,3,4}，k=3，共有6种不同的划分，即划分数为6，这些划分为

    {1,2}∪{3}∪{4}；　{1,3}∪{2}∪{4}；
    {1,4}∪{2}∪{3}；

    {2,3}∪{1}∪{4}；　{2,4}∪{1}∪{3}；
    {3,4}∪{1}∪{2}；

递归算法设计举例：集合划分（2）

- 问题分析(cont.)
  - 递归关系的推导

设n个元素$a_1,a_2,\ldots,a_n$放入k个集合的划分数为s(n,k)，考虑a_n的二种情形：

Case 1: 设$\{a_n\}$恰为k个子集中的一个，此时划分数为

s(n-1,k-1)

Case 2: $\{a_n\}$不是k个子集中的一个，即a_n与其它元素构成一个子集。这时可以先将$\{a_1,a_2,\ldots,a_{n-1}\}$划分成k个子集，共有s(n-1,k)种划分，再将$a_n$加入到k个子集中的某个子集中去，共有k种方式。由乘法原理知：划分数有k*s(n-1,k)

=> s(n,k)=s(n-1,k-1)+k*s(n-1,k) n>k 且k>1

递归算法设计举例：集合划分（3）

- 问题分析(cont.)
  - 递归关系的推导

下面确定s(n,k)的边界条件：

(1)s(n,0)=0，即n个元素不放入任何一个集合中去；

(2)s(n,1)=1，s(n,n)=1

$$\Rightarrow \quad s(n,k)=\begin{cases} s(n-1,k-1)+k*s(n-1,k) & n>k\text{且}k>1 \\ 0 & k=0\text{或}n<k \\ 1 & k=1\text{或}n=k \end{cases}$$

递归算法设计举例：集合划分（4）

- 算法

```
S( n, k)
{
   if k=0 or n < k then
       return 0;
   else if k=1 or n=k then
       return 1;
   else return s(n-1,k-1)+k*s(n-1,k);
  }
```

- 时间分析？

第1章(补充) 递归与分治法

1.1 递归设计技术

1.2 二分查找

1.3 大整数乘法

1.4 Strassen矩阵乘法

1.5 导线和开关

1.2 二分查找

- 基本思想
- 递归算法
- 非递归算法

基本思想和非递归算法

- 基本思想

将有序序列（升序）等分为几乎相等的两部分，待查关键字与划分元比较。如果小于划分元，则递归处理左半部分；否则，递归处理右半部分。

- 非递归算法

```
BinarySearch1(L[ ], n, x)  //找到x返回下标值，找不到返回-1
{
    left←1, right←n; flag←0;  //flag为标志变量
    while (left≤right and flag=0)  do
    {
       mid ←⌊(left+right)/2⌋;
       if x=L[mid]  then flag←1;
       else if x < L[mid] then right←mid-1;
       else left←mid+1;
     }
    if flag=1 then return mid;
    else return -1;
}
```

递归算法

- 递归算法

```
BinarySearch2(L[ ], x, i, j)
{  //在有序表L[i..j]中查找x
    if i > j  then return -1;
    if  i=j then
       if  x=L[i]  then  return i;
       else return -1;
    else  {
       mid ← ⌊(i+j)/2⌋;
       if x=L[mid] then return mid;
       else if x < L[mid]  then
            return BinarySearch2(L, x, i, mid-1);
       else return BinarySearch2(L, x, mid+1, j);
    }
}
```

- 时间分析

$$T(n)=\begin{cases} O(1) & n=1 \\ T(n/2)+O(1) & n>1 \end{cases}$$

$$\Rightarrow T(n)=O(\log n)$$

第1章(补充) 递归与分治法

1.1 递归设计技术

1.2 二分查找

1.3 大整数乘法

1.4 Strassen矩阵乘法

1.5 导线和开关

1.3 大整数乘法

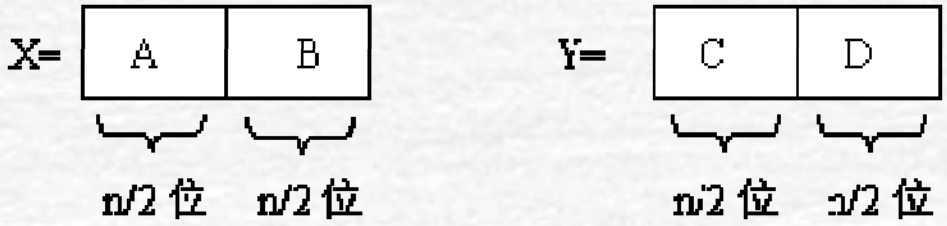
- 问题描述
- 普通递归乘法分析
- 改进的分治乘法

问题描述

- 处理很大的整数时，它无法在计算机硬件能直接表示的范围内进行处理。若用浮点数来表示它，则只能近似地表示它的大小，计算结果中的有效数字也受到限制。若要精确地表示大整数并在计算结果中要求精确地得到所有位数上的数字，就必须用软件的方法来实现大整数的算术运算。

普通递归乘法分析

设X,Y是n位二进制整数，分段表示如下：



即 $X=A2^{n/2}+B$， $Y=C2^{n/2}+D$ 则

$$XY=(A2^{n/2}+B)(C2^{n/2}+D)=AC2^{n}+(AD+BC)\,2^{n/2}+BD$$

计算成本：4次n/2位乘法，3次不超过n位加法，2次移位，所有加法和移位共计O(n)次运算。我们有

$$\begin{cases} T(1)=1 \\ T(n)=4T(n/2)+O(n) \end{cases}$$

=> $T(n)=O(n^2)$

改进的分治乘法（1）

- 1962年苏联数学家Karatsuba和Ofman提出

由$X=A2^{n/2}+B$，$Y=C2^{n/2}+D$ 则

$$XY=(A2^{n/2}+B)(C2^{n/2}+D)=AC2^{n}+(AD+BC)\,2^{n/2}+BD$$
$$=AC2^{n}+((A-B)(D-C)+AC+BD)2^{n/2}+BD$$

计算成本：3次n/2位乘法，6次不超过n位加减法，2次移位，所有加法和移位共计O(n)次运算。由此可得，

$$\begin{cases} T(1)=1 \\ T(n)=3T(n/2)+cn \end{cases}$$

=> $T(n)=O(n^{\log 3})=O(n^{1.59})$

注：如果将一个大整数分成3段或4段做乘法，计算复杂性会发生什么变化呢？是否优于分成2段的乘法？这个问题请大家自己考虑。

改进的分治乘法（2）

- 改进的分治算法举例

已知X=2368，y=3925，求XY

解：取基d=10，A=23，B=68，C=39，D=25，则

① AC=23×39=897

② BD=68×25=1700

③ (A-B)(D-C)+AC+BD

=(23-68)(25-39)+897+1700

=45×14+8897+1700

=3227

∴ $XY=1700+3227\times10^2+897\times10^4=9294400$

共做12次位乘，而普通乘法要做16次位乘。

注：大整数乘法在数据加密领域中有广泛应用，n值的位数通常在200位以上。

第1章(补充) 递归与分治法

1.1 递归设计技术

1.2 二分查找

1.3 大整数乘法

1.4 Strassen矩阵乘法

1.5 导线和开关

1.4 Strassen矩阵乘法

- 普通矩阵乘法
- Strassen矩阵乘法

普通矩阵乘法

设n×n阶矩阵A和B，计算C=A×B

$$c_{ij}=\sum_{k=1}^{n}a_{ik}b_{kj}\quad i,j=1,2,\cdots,n$$

- 求C=AB即对n^2个元素c_{ij}进行计算，故要作n^3次乘法。
- 相当长时间内没有人怀疑过是否可以用少于n^3次乘法来完成。

Strassen矩阵乘法（1）

- 以n=2为例的普通矩阵乘法

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

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

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

$$=\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

共需8次乘法

Strassen矩阵乘法（2）

- Strassen的分治算法(1969)
  - 将C=A×B写为2×2的分块矩阵：

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

  - Strassen提出的算法如下：

令 $P=(A_{11}+A_{22})(B_{11}+B_{22})$, $Q=(A_{21}+A_{22})B_{11}$

$R=A_{11}(B_{12}-B_{22})$, $S=A_{22}(B_{21}-B_{11})$

$T=(A_{11}+A_{12})B_{22}$, $U=(A_{21}-A_{11})(B_{11}+B_{12})$

$V=(A_{12}-A_{22})(B_{21}+B_{22})$

则 $C_{11}=P+S-T+V$, $C_{12}=R+T$

$C_{21}=Q+S$, $C_{22}=P+R-Q+U$

乘法次数从8次减为7次。

Strassen矩阵乘法（3）

- 时间分析：

用了7次对于n/2阶矩阵乘积的递归调用和18次n/2阶矩阵的加减运算。由此可知，该算法的所需的计算时间T(n)满足如下的递归方程：

$$T(n)=\begin{cases} O(1) & n=2 \\ 7T(n/2)+O(n^2) & n>2 \end{cases}$$

=> $T(n)=O(n^{\log 7})\approx O(n^{2.81})$

- Remark：

.有人提出了计算2×2阶矩阵乘法的36种不同方法，都只要做7次乘法；

. Hopcroft和Kerr(1971)已经证明，7次乘法是必要的；

.是否研究3×3或5×5矩阵的更好算法？

.目前最好的计算时间上界是$O(n^{2.367})$，而最好下界仍是$\Omega(n^2)$。

第1章(补充) 递归与分治法

1.1 递归设计技术

1.2 二分查找

1.3 大整数乘法

1.4 Strassen矩阵乘法

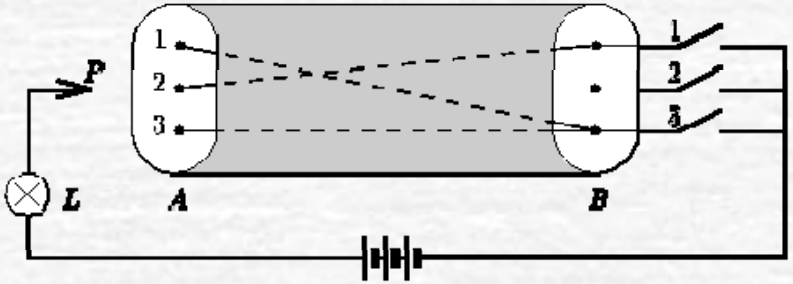
1.5 导线和开关

1.5 导线和开关

- 问题描述
- 穷举检测法
- 分治检测法

问题描述

如图所示，具有3根导线的电缆把A区和B区连接起来。在A区3根导线标以1，2，3；在B区导线1和3被连到开关3，导线2连到开关1。



一般地，电缆含m(1≤m≤90)根导线，在A区标以1,2,...,m。在B区有m个开关，标为1,2,..,m。每一根导线都被严格地连到这些开关中的某一个上；每一个开关上可以连有0根或多根导线。

算法应作出哪些测量来确定导线和开关的连接，使测量数最少，这些测量是探头的移动和开关的开闭数。

穷举检测法

- 算法描述

  将探头p移到导线i, i=1,2,…,m;

  顺序闭合开关j, j=1,2,…,m;

  如果灯L亮，则导线i与开关j相连；

  否则，导线i与开关j不相连；

  将开关j打开；

- 操作数分析

  探头移动m次，每次开关操作2m次，∴ $T(m)=O(m^2)$

- 操作举例（3根导线、3个开关）

  探头移动3次，开关操作（6+2+6）=14次

分治检测法（1）

- 算法描述

第1轮操作：//确定每根导线属于哪半区

上半区开关合上，下半区开关断开，探头遍历；

第2轮操作：//确定每根导线属于哪半半区

对每半区再分：上半半区开关合上，下半半区开关断开，探头遍历；

… …

第$\lfloor \log m \rfloor+1$轮操作：//确定每根导线属于哪个开关

每个开关合上，探头遍历；

- 操作数分析

每轮开关开闭2m次，∴ T(m)=O(mlogm)

分治检测法（2）

- 操作举例（3根导线、3个开关）

	探头移动次数	开关操作次数	合计
第1轮：	3	2	
第2轮：	2	2	
小计：	5	4	9
注：穷举法：	3	14	17

- 思考题

  写出分治检测法的形式化算法。

End of SChap1