2.4　分 治 法

分治法的基本思想是将一个规模为n的问题，分解为k个规模较小的子问题，这些子问题互相独立且与原问题相同。递归的求解这些子问题，然后将各个子问题的解合并得到原问题的解。

分治算法框架-1

- 算法设计思想：
  - 将整个问题分解成若干个小问题后分而治之。
  - 如果分解得到的子问题相对来说还太大，则可反复使用分治策略将这些子问题分成更小的同类型子问题，直至产生出方便求解的子问题，必要时逐步合并这些子问题的解，从而得到问题的解。
- 分治法的基本步骤在每一层递归上都有三个步骤：
  - 1)分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题；
  - 2)解决：若子问题规模较小而容易被解决则直接解，否则再继续分解为更小的子问题，直到容易解决；
  - 3)合并：将已求解的各个子问题的解，逐步合并为原问题的解。

分治算法框架-2

- 有时问题分解后，不必求解所有的子问题，也就不必作第三步的操作。比如折半查找，在判别出问题的解在某一个子问题中后，其它的子问题就不必求解了，问题的解就是最后(最小)的子问题的解。分治法的这类应用，又称为“减治法”。
- 多数问题需要所有子问题的解，并由子问题的解，使用恰当的方法合并成为整个问题的解，比如归并排序，就是不断将子问题中已排好序的解合并成较大规模的有序子集。

分治算法框架-3

适合用分治法策略的问题：

- 当求解一个输入规模为n且取值又相当大的问题时，用蛮力策略效率一般得不到保证。若问题能满足以下几个条件，就能用分治法来提高解决问题的效率。
  - 1)能将这n个数据分解成k个不同子集合，且得到k个子集合是可以独立求解的子问题，其中1＜k≤n；
  - 2)分解所得到的子问题与原问题具有相似的结构，便于利用递归或循环机制；
  - 3)求出这些子问题的解之后，就可推解出原问题的解；

2.4.2.1 分治法的适用条件

分治法所能解决的问题一般具有以下几个特征：

- 该问题的规模缩小到一定的程度就可以容易地解决；
- 该问题可以分解为若干个规模较小的相同问题，即该问题具有**最优子结构性质**
- 利用该问题分解出的子问题的解可以合并为该问题的解；
- 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

这条特征涉及到分治法的效率，如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然也可用分治法，但一般用**动态规划**较好。

2.5.2.2 分治法的基本步骤

划分步

治理步

组合步

```
divide-and-conquer(P)
 {
  if ( | P | <= n0) adhoc(P);   //解决小规模的问题
  divide P into smaller subinstances P1,P2,..., Pk;  //分解问题
  for (i=1;i<=k;i++)
   yi=divide-and-conquer(Pi);  //递归的解各子问题
  return merge(y1,..., yk);  //将各子问题的解合并为原问题的解
 }
```

人们从大量实践中发现，在用分治法设计算法时，最好使子问题的规模大致相同。即将一个问题分成大小相等的k个子问题的处理方法是行之有效的。这种使子问题规模大致相等的做法是出自一种**平衡(balancing)子问题**的思想，它几乎总是比子问题规模不等的做法要好。

2.5.3 实例分析与设计

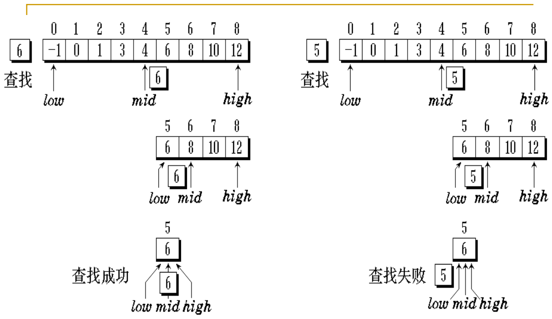
【例2.5.1】二分搜索技术

划分：

给定已按升序排好序的n个元素$A[0:n-1]$，现要在这n个元素中找出一特定元素x。

分析：
- ✓ 该问题的规模缩小到一定的程度就可以容易地解决；
- ✓ 该问题可以分解为若干个规模较小的相同问题；
- ✓ 分解出的子问题的解可以合并为原问题的解；
- ✓ 分解出的各个子问题是相互独立的。

分析：很显然此问题分解出的子问题相互独立，即在$A[i]$的前面或后面查找x是独立的子问题，因此满足分治法的第四个适用条件。



查找成功的例子　　　　查找失败的例子

【例2.5.2】二进制大整数的乘法

请设计一个有效的算法，可以进行两个n 位二进制大整数的乘法运算。

划分：

	$n/2$ 位	$n/2$ 位
$X =$	a	b
$Y =$	c	d

$X = a\ 2^{n/2} + b \qquad Y = c\ 2^{n/2} + d$

$XY = ac\ 2^n + (ad+bc)\ 2^{n/2} + bd$

	$n/2$ 位	$n/2$ 位
X =	a	b
Y =	c	d

$X = a\ 2^{n/2} + b \qquad Y = c\ 2^{n/2} + d$

$XY = ac\ 2^n + (ad+bc)\ 2^{n/2} + bd$

4次$n/2$位整数的乘法以及3次不超过$2n$位的整数加法。

复杂度分析

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

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

$XY = ac\ 2^n + (ad+bc)\ 2^{n/2} + bd$

为了降低时间复杂度，必须减少乘法的次数。

1. $XY = ac\ 2^n + ((a-b)(d-c)+ac+bd)\ 2^{n/2} + bd$
2. $XY = ac\ 2^n + ((a+b)(d+c)-ac-bd)\ 2^{n/2} + bd$

复杂度分析

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

$T(n)=O(n^{\log 3}) =O(n^{1.59})$ ✓较大的改进

大整数的乘法

◆一般的分治方法：O(n²) ✖效率太低

◆分治法: O(n^1.59) ✔较大的改进

◆更快的方法??

➢如果将大整数分成更多段，用更复杂的方式把它们组合起来，将有可能得到更优的算法。

➢最终的，这个思想导致了**快速傅利叶变换**(Fast Fourier Transform)的产生。该方法也可以看作是一个复杂的分治算法。

【例2.5.3】多项式乘积的分治方法

计算两个n阶多项式的乘法：

$$p(x)=a_0+a_1x+a_2x^2+a_3x^3+\cdots+a_nx^n$$

$$q(x)=b_0+b_1x+b_2x^2+b_3x^3+\cdots+b_nx^n$$

采用一般的方法计算，需要（n+1)²次乘法运算和n(n+1)次加法运算。

多项式的划分原理

为了减少两个多项式乘法中乘法运算的次数，考虑把一个多项式划分成两个多项式。

$$p(x)=p_0(x)+p_1(x)x^{n/2}$$
$$q(x)=q_0(x)+q_1(x)x^{n/2} \qquad (4.1)$$

则有：

$$p(x)q(x)=p_0(x)q_0(x)+(p_0(x)q_1(x)+p_1(x)q_0(x))x^{n/2}+p_1(x)q_1(x)x^n$$

$$p(x)q(x)=p_0(x)q_0(x)+(p_0(x)q_1(x)+p_1(x)q_0(x))x^{n/2}+p_1(x)q_1(x)x^n$$

而：

$$(p_0(x)+p_1(x))(q_0(x)+q_1(x))=p_0(x)q_0(x)+p_1(x)q_1(x)+p_0(x)q_1(x)+p_1(x)q_0(x)$$

故：

$$p_0(x)q_1(x)+p_1(x)q_0(x)=(p_0(x)+p_1(x))(q_0(x)+q_1(x))-p_0(x)q_0(x)+p_1(x)q_1(x)$$

$$r_0(x)=p_0(x)q_0(x)$$
$$r_1(x)=p_1(x)q_1(x)$$
$$r_2(x)=(p_0(x)+p_1(x))(q_0(x)+q_1(x))$$

4个多项式乘法

$$p(x)q(x)=p_0(x)q_0(x)+(p_0(x)q_1(x)+p_1(x)q_0(x))x^{n/2}+p_1(x)q_1(x)x^n$$

$$p(x)q(x)=r_0(x)+((r_2(x)-r_0(x)-r_1(x))x^{n/2}+r_1(x)x^n$$

3个多项式乘法

$$r_0(x)=p_0(x)q_0(x)$$
$$r_1(x)=p_1(x)q_1(x)$$
$$r_2(x)=(p_0(x)+p_1(x))(q_0(x)+q_1(x))$$

【例2.5.4】 Strassen矩阵乘法

A和B的乘积矩阵C中的元素C[i,j]定义为: $C[i][j]=\sum_{k=1}^{n}A[i][k]B[k][j]$

若依此定义来计算A和B的乘积矩阵C，则每计算C的一个元素C[i][j]，需要做n次乘法和n-1次加法。因此，算出矩阵C的 个元素所需的计算时间为O(n³)

◆分治法:

使用与上例类似的技术，将矩阵A，B和C中每一矩阵都分块成4个大小相等的子矩阵。由此可将方程C=AB重写为:

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

由此可得:

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

复杂度分析

$$T(n) = \begin{cases} O(1) & n = 2 \\ 8T(n/2) + O(n^2) & n > 2 \end{cases}$$

$T(n)=O(n^3)$

为了降低时间复杂度，必须减少乘法的次数。

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

$$M_1 = A_{11}(B_{12} - B_{22})$$
$$M_2 = (A_{11} + A_{12})B_{22}$$
$$M_3 = (A_{21} + A_{22})B_{11}$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_6 = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$M_7 = (A_{11} - A_{21})(B_{11} + B_{12})$$

➡

$$C_{11} = M_5 + M_4 - M_2 + M_6$$
$$C_{12} = M_1 + M_2$$
$$C_{21} = M_3 + M_4$$
$$C_{22} = M_5 + M_1 - M_3 - M_7$$

复杂度分析

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

$T(n)=O(n^{\log 7}) =O(n^{2.81})$ ✓ 较大的改进

◆传统方法：$O(n^3)$

◆分治法: $O(n^{2.81})$

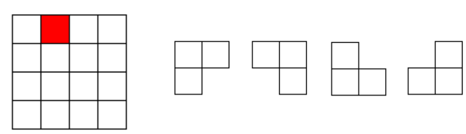
◆更快的方法??

➢Hopcroft和Kerr已经证明(1971)，计算2个 2 × 2 矩阵的乘积，7次乘法是必要的。因此，要想进一步改进矩阵乘法的时间复杂性，就不能再基于计算2×2矩阵的7次乘法这样的方法了。或许应当研究 3 × 3 或 5 × 5 矩阵的更好算法。

➢在Strassen之后又有许多算法改进了矩阵乘法的计算时间复杂性。目前最好的计算时间上界是 **$O(n^{2.376})$**

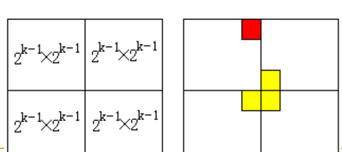
【例2.5.5】 棋盘覆盖

在一个$2^k \times 2^k$ 个方格组成的棋盘中，恰有一个方格与其它方格不同，称该方格为一特殊方格，且称该棋盘为一特殊棋盘。在棋盘覆盖问题中，要用图示的4种不同形态的L型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格，且任何2个L型骨牌不得重叠覆盖。



划分:

当k>0时，将$2^k \times 2^k$棋盘分割为4个$2^{k-1} \times 2^{k-1}$ 子棋盘(a)所示。特殊方格必位于4个较小子棋盘之一中，其余3个子棋盘中无特殊方格。为了将这3个无特殊方格的子棋盘转化为特殊棋盘，可以用一个L型骨牌覆盖这3个较小棋盘的会合处，如 (b)所示，从而将原问题转化为4个较小规模的棋盘覆盖问题。递归地使用这种分割，直至棋盘简化为棋盘1×1。



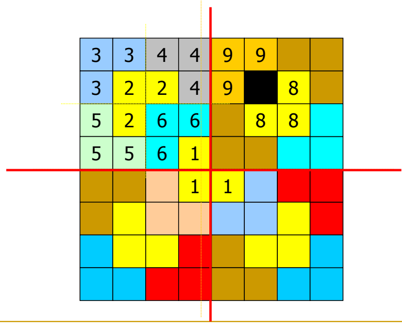
(a) (b)

复杂度分析

$$T(k) = \begin{cases} O(1) & k = 0 \\ 4T(k-1) + O(1) & k > 0 \end{cases}$$

$T(n)=O(4^k)$ 渐进意义下的最优算法

- $2^3 \times 2^3$棋盘的部分结果



【例2.5.6】循环赛日程表

- 设有n＝2^k个运动员要进行网球循环赛。现要设计一个满足以下要求的比赛日程表：
  (1)每个选手必须与其他n-1个选手各赛一次；
  (2)每个选手一天只能赛一次；
  (3)循环赛一共进行n-1天。

- 按此要求可将比赛日程表设计成有n行和n-1列的表，在表中第i行和第j列处填入第i个选手在第j天所遇到的选手。

- 如果只有两人参赛，比赛日程表？

1	2
2	1

注：第一列为参赛运动员编号，第二列为第一天与第一列运动员比赛的运动员编号。

- 若是四个人参赛，比赛日程表？

1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

分解为两个规模为2的子问题

按分治策略，将所有选手分为两组，n个选手的比赛日程表就可以通过为n/2个选手设计的比赛日程表来决定。递归地用一分为二的策略对选手进行分割，直到只剩下2个选手时，日程表制定就很简单。这时只要让这2个选手进行比赛即可。

1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

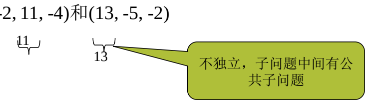
8个选手的比赛日程表

【例2.5.7】求数列的最大子段和

- **给定n个元素的整数数列(可能为负整数)a_1，a_2，…，a_n，求形如a_i，a_{i+1}，a_j，i，j=1，2，…，n，i<=j的子段，使其和为最大。**
- **例如，当(a1, a2, a3, a4, a5, a6)=(-2, 11, -4, 13, -5, -2)**

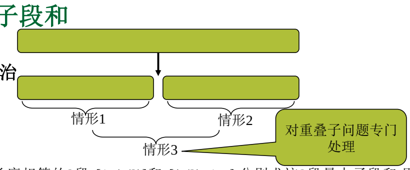
  max_sum = 20，best_i = 2，best_j = 4

- 若用一般的二分法解决该实例？
- 分解为两组(-2, 11, -4)和(13, -5, -2)



4 最大子段和

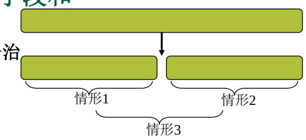
- “三”分治



将a[1..n]分为长度相等的2段a[1..(n/2)]和a[(n/2)+1..n],分别求这2段最大子段和,则a[1.n]最大子段和有3种情形:
1)a[1..n]的最大子段和与a[1..(n/2)]的最大子段和相同；
2)a[1..n]的最大子段和与a[(n/2)+1..n]的最大字段和相同;
3)a[1..n]的最大字段和为a[i..j],且1≤i≤(n/2),(n/2)+1≤j≤n。
情况1)和情况2)可分别递归求得。
对于情况3）,a[(n/2)]与a[(n/2)+1]一定在最大子段中。
因此,可以计算出a[i..(n/2)]最大值s1和a[(n/2)+1..j]最大值s2。则s1+s2即为情况3)时最优值。

4 最大子段和

- “三”分治



- 对分解得到的左右子段，求
  - 左右子段内部的最大子段和；
  - 左子段中满足以下条件的最大的子段和s1：以任何位置i开始，结尾处n/2结束；
  - 右子段中满足以下条件的最大的子段和s2：以起始位置n/2+1开始，任何位置j结束；
- 原问题最大子段和是左右子段内最大和与两个子段最大值求和s1+s2中的大者。

【例2.5.8】快速排序

- 快速排序方法的基本思想是任取待排序对象序列中的某个对象 (例如取第一个对象) 作为枢轴(pivot)，按照该对象的关键字大小，将整个对象序列划分为左右两个子序列：
  - 左侧子序列中所有对象的关键字都小于或等于枢轴对象的关键字
  - 右侧子序列中所有对象的关键字都大于枢轴对象的关键字
- 枢轴对象则排在这两个子序列中间(这也是该对象最终应安放的位置)。

实现：

```
void QuickSort (Type a[], int p, int r)
{
     if (p<r) {
       int q=Partition(a,p,r);
       QuickSort (a,p,q-1); //对左半段排序
       QuickSort (a,q+1,r); //对右半段排序
       }
}
```


复杂性分析：

快速排序算法的性能取决于划分的对称性。通过修改算法**partition**，可以设计出采用随机选择策略的快速排序算法。在快速排序算法的每一步中，当数组还没有被划分时，可以在a[p:r]中随机选出一个元素作为划分基准，这样可以使划分基准的选择是随机的，从而可以期望划分是较对称的。

$$\begin{cases} f(1)=0 & n=1 \\ f(n)=2f(n/2)+\Theta(n), & n>1 \end{cases}$$

最坏时间复杂度：O(n^2)
平均时间复杂度：O(nlogn)
辅助空间：O(n)或O(logn)

【例2.5.9】线性时间选择

给定线性序集中n个元素和一个整数k，$1\le k\le n$，要求找出这n个元素中第k小的元素。

算法的基本思想：在分治算法的递归调用的每一个划分里，放弃一个固定的部分，对其余元素进行递归。于是，问题的规模便以几何级数递减。

具体步骤如下：
1）当n<=n_0时，直接排序数组，第k个元素即为第k小的元素。其中n_0为某个阈值；否则转2）
2）把元素划分为p=n/5 组，每组5个元素，不足5个元素的组不予处理；
3）取每组中值元素，构成一个规模为p的数组M;
4）对M递归的执行该算法，得到一个中值的中值m;
5）把原数组划分成P,Q R三组，大于m的放P，等于m的放Q，小于m的放R;
6）如果│P│>k，对P进行递归算法，否则转7）；
7）如果│P│+│Q│ >=k，m就是要选择的元素，否则转8）；
8）对R进行递归算法。

例如：按递增顺序，找出下面29个元素的第18小的元素。

8,31,60,33,17,4,51,57,49,35,

11,43,37,3,13,52,6,19,25,32,

54,16,5,41,7,23,22,46,29

执行步骤：k=18

1）分组：（8,31,60,33,17），(4,51,57,49,35），（11,43,37,3,13），(52,6,19,25,32），（54,16,5,41,7），(23,22,46,29）（不予处理）

2）提取每组的中值元素构成中值元素数组（31,49,13,25,16）；

3）递归的求中值数组的中值，为m=25;

4）根据25，将原数组重新划分为

P=(8,17,4,11,3,13,6,19,16,5,7,23,22)

Q=(25)

R=(31,60,33,51,57,49,35,43,37,52,32,54,41,46,29)

5）由于｜P｜=13，｜Q｜=1，而k=18，所以放弃P,Q，k=k-13-1=4;对R进行递归求解；

R=(31,60,33,51,57,49,35,43,37,52,32,54,41,46,29)

6）将R划分为 (31,60,33,51,57），(49,35,43,37,52），（32,54,41,46,29)

7）取这三组的中值得到中值数组（51,43,41），中值为m=43

8）将R数组按中值分组：

P=(31,33,35,37,32,41,29)

Q=(43)

R=(60,51,57,49,52,54,46)

9）因为k=4，故放弃Q,R，递归求解P数组

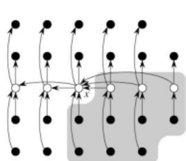
P=(31,33,35,37,32,41,29)

10）对P分组：（31,33,35,37,32），其中值元素为33;

11）根据33，将P数组重新分组：

P=(31,32,29), Q=(33), R=(35,37,41)

12）k=4，要取得值在P，Q中，排序P,Q中的数据，故要选择的数据为33。



取n_0=38:

复杂度分析

$$T(n) \le \begin{cases} C_1 & n < 38 \\ C_2 n + T(n/5) + T(3n/4) & n \ge 38 \end{cases}$$

$T(n)=\boldsymbol{O}(n)$

【例2.5.10】斐波那契数列

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

问题：如何计算斐波那契数列？

（1）最古老方法：递归

if(n=0 or 1) return n;

else 递归地计算F(n+1)和F(n+2)

时间复杂度为指数级（n的常数次方）。

（2）最简易改进：存储每个已计算出来的F(n)，求和：

F(n)=F(n-1)+F(n-2)。

时间复杂度：O(n)

但是计算第N个时，需要先计算所有前面的数。

斐波那契数列

（3）矩阵方法：数列有如下性质：

$$\begin{pmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

（So，变成2阶矩阵乘法问题）

上式用数学归纳法证明：

(1) 当n=1时：

$$\begin{pmatrix} F(2) & F(1) \\ F(1) & F(0) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

(2) 当n>1时：

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