第五章 分治策略

苏 明

分治策略

- “分而治之”

✓ 把一个问题分解成几个部分

✓ 递归解决规模比较小的类似问题

✓ 把子问题的解合成完整的解

分治策略

- 经常使用的办法：

- 把一个规模为n的问题分解成两个规模$\frac{1}{2}$n的问题；
- 分别递归解决两个部分的问题；
- 在线性时间内把这两个部分问题的解合成一个完整的解。

归并排序算法

- 问题：给定n个元素，按照递增的顺序对其进行排序。
- **排序应用场景：**
- *直接应用*：

  目录文件排序；

  电话簿排序；

  Google上输出按照PageRank排序；

  间接应用：

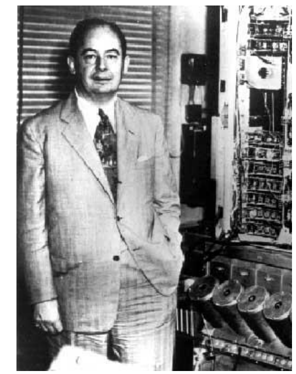
  区间调度；

  最小生成树问题；

  数据压缩；

归并排序

- 归并排序
  - Divide array into two halves.
  - Recursively sort each half.
  - Merge two halves to make sorted whole.



Jon von Neumann (1945)

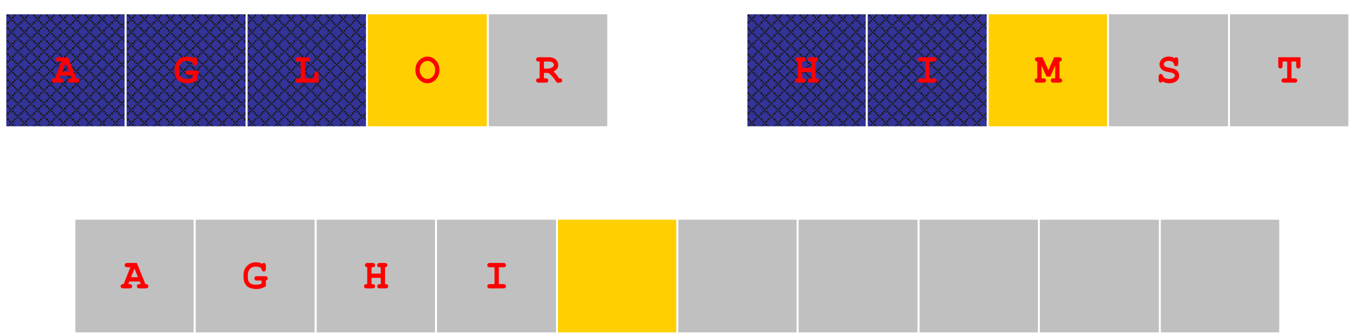
A	L	G	O	R	I	T	H	M	S

A	L	G	O	R	I	T	H	M	S	divide	O(1)
A	G	L	O	R	H	I	M	S	T	sort	2T(n/2)

A	G	H	I	L	M	O	R	S	T	merge	O(n)

归并排序

- 合并：把两个排好序的部分合成一个完整的排序
- 效率分析：采用一个临时数组；需要O(n)次比较



归并排序

- 对于归并排序算法，设T(n)表示该算法在规模为n的输入实例上最坏的运行（比较）时间。
- 开始假设n是2的整数次幂
- 可以得到递归公式如下：

递推关系

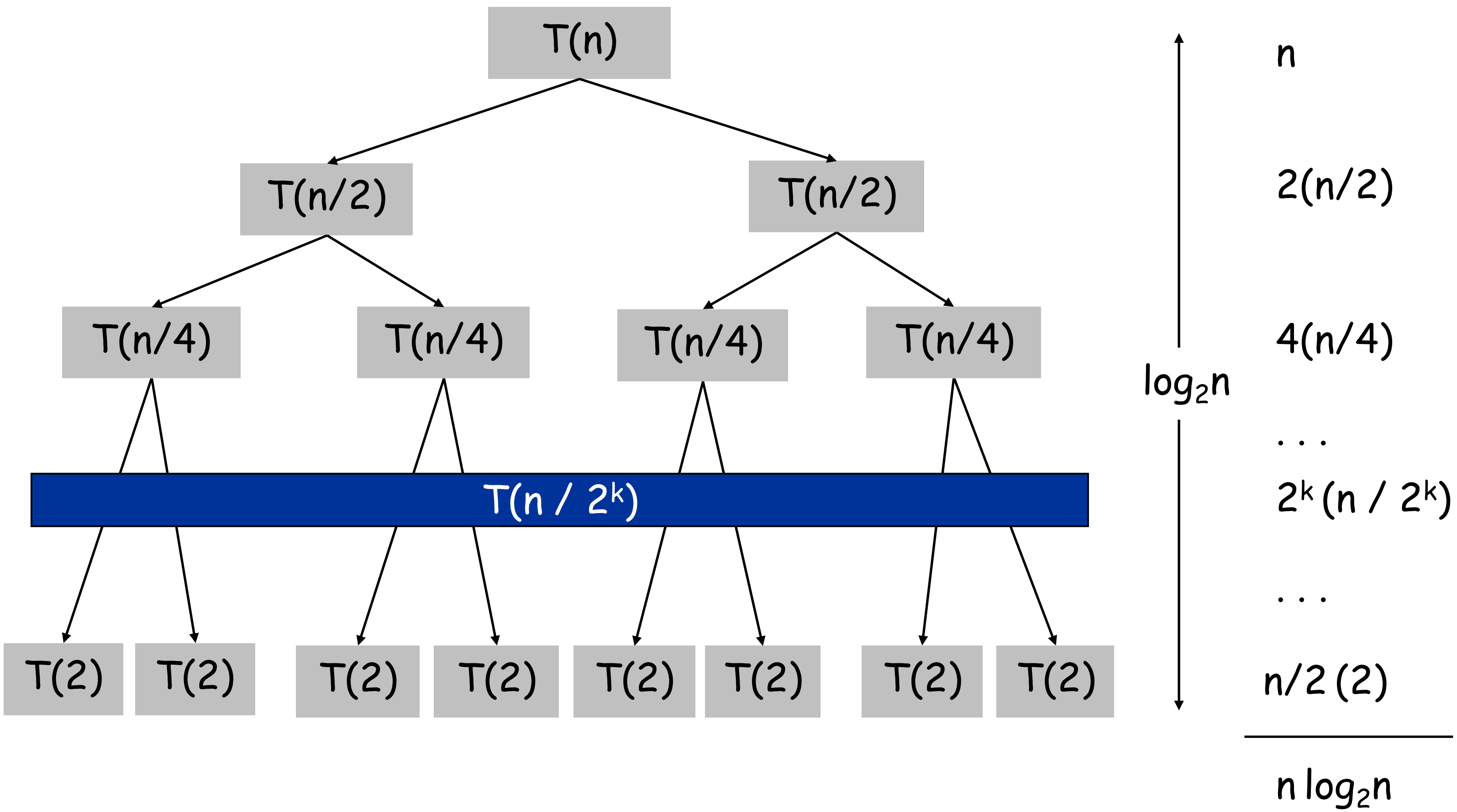
- 命题5.1 对某个常数c,

$$T(n) \le \begin{cases} 2T(n/2) + cn, n > 2 \\ c, n = 2 \end{cases}$$

- 或者

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

递推关系



T(n)
T(n/2)
T(n/2)
T(n/4)
T(n/4)
T(n/4)
T(n/4)
T(n / 2^k)
T(2)
T(2)
T(2)
T(2)
T(2)
T(2)
T(2)
T(2)
log₂n
n
2(n/2)
4(n/4)
. . .
2^k (n / 2^k)
. . .
n/2 (2)
n log₂n

递推关系

- 为了严格证明这个命题，先考虑最特殊情形

- 命题：如果T(n)满足如下关系，且n是2的整数次幂，那么$T(n) = n \log_2 n$.

$$
\mathrm{T}(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}
$$

递推关系

- 证明：对于n>1,可以得到

$$
\begin{aligned}
\frac{T(n)}{n} &= \frac{2T(n/2)}{n} + 1 \\
&= \frac{T(n/2)}{n/2} + 1 \\
&= \frac{T(n/4)}{n/4} + 1 + 1 \\
&\cdots \\
&= \frac{T(n/n)}{n/n} + \underbrace{1 + \cdots + 1}_{\log_2 n} \\
&= \log_2 n
\end{aligned}
$$

递推关系

- 证明： 用归纳法也可以：

✓ n=1 命题显然成立；

✓ 假设T(n) = n $\log_2$ n成立($n=2^k$)，那么

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n\log_2 n + 2n \\ &= 2n\left(\log_2(2n)-1\right) + 2n \\ &= 2n\log_2(2n) \end{aligned}$$

所以命题对于$n=2^{k+1}$成立。

递推关系

- 回到一般情形

- 命题：如果T(n)满足如下关系，那么T(n) $\leq n \lceil \lg n \rceil$.

$$T(n) \leq \begin{cases} 0 & \text{if } n=1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

递推关系

- 采用部分替换的方法

  在这种替换中，先猜想出解的形式，但是开始的时候常数项以及相关的参数不确定；在随后的推导中可以确定这些参数的值。有点类似于“待定系数法”。

其他递推关系

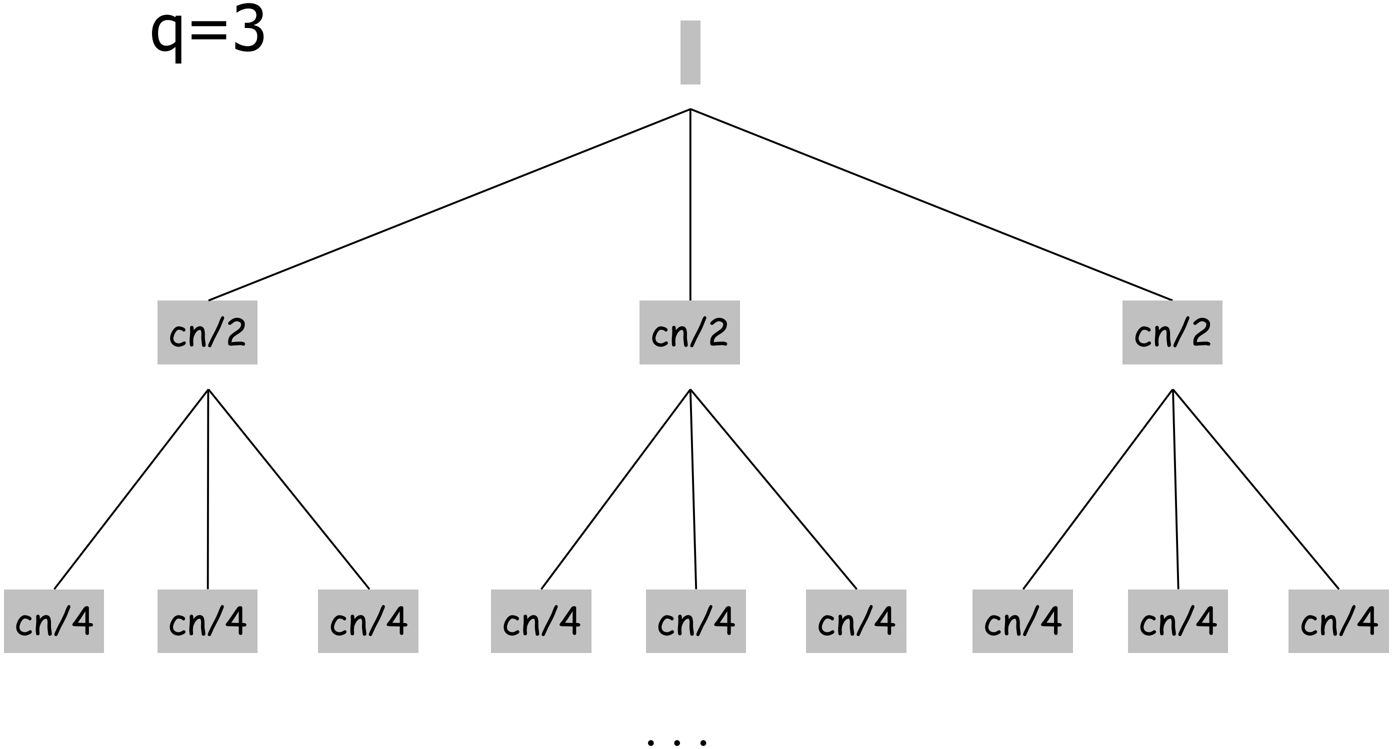
- 命题5.3 对某个常数c，

$$T(n) \le \begin{cases} qT(n/2) + cn, n > 2 \\ c, n = 2 \end{cases}$$

q>2, q=1, q=2 递推关系的性质不同

其他递推关系

q=3



cn

(3/2) cn

(9/4)cn

. . .

其他递推关系

类似的根据上图可以得到：

$$T(n) \le \sum_{j=0}^{\log_2 n-1} \left(\frac{q}{2}\right)^j cn = cn \sum_{j=0}^{\log_2 n-1} \left(\frac{q}{2}\right)^j = cn\left(\frac{r^{\log_2 n}-1}{r-1}\right)$$

$$\le cn\left(\frac{r^{\log_2 n}}{r-1}\right) = \frac{c}{r-1} n \cdot r^{\log_2 n} = \frac{c}{r-1} n \cdot n^{\log_2 r} = \frac{c}{r-1} n^{\log_2 q} = O(n^{\log_2 q})$$

定理5.4 任何满足5.3并具有q>2的函数T(.)是 $O(n^{\log_2 q})$ 有界的。

其他递推关系

- 定理5.5 任何满足5.3式并具有q=1的函数T(.) (T(n)<=T(n/2)+O(n))是O(n)有界的。
- 证明：

$$T(n) \le \sum_{j=0}^{\log_2 n-1} \frac{cn}{2^j} = cn \sum_{j=0}^{\log_2 n-1} \frac{1}{2^j} \le 2cn = O(n)$$

其他递推关系

- 命题5.6 对某个常数c,

$$T(n) \le \begin{cases} 2T(n/2)+cn^2, n>2 \\ c, n=2 \end{cases}$$

T(n)的复杂度应该是？

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

5.3 计数逆序

- 问题提出
- Web网站使用一种称为*协同过滤*的技术，试图使用这种技术把你(对于书，电影，餐馆等)的爱好与其他人表现的爱好进行匹配，一旦web网站识别某些人与你有“类似”的口味（基于评价各种事物的比较），它就推荐一些相关的新事物。

计数逆序

- 相似度量?
- 考虑**逆序**的个数
- 两个指标$i<j$构成一个逆序，如果$a_i>a_j$.

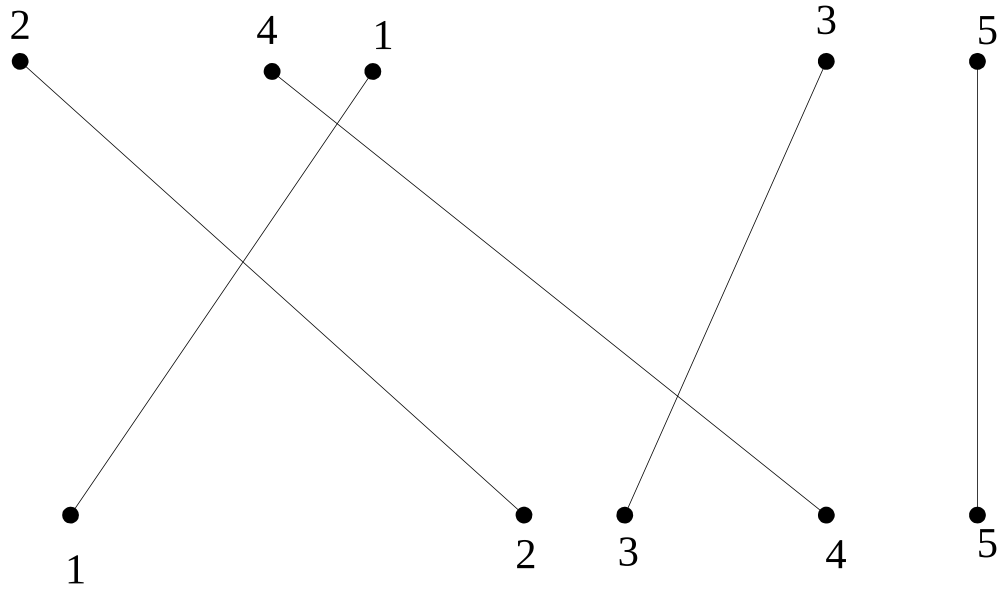
Songs

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

<u>Inversions</u>

3-2, 4-2

计数逆序



逆序数：3---(2,1),(4,1),(4,3)

计数逆序

- 穷举（Brute force）： $\Theta(n^2)$.
- 排序： $\Theta(n\lg n)$
- ？分治策略

---可同时进行逆序计数的工作

例子

- 如下给定的序列进行排序，计数逆序

1	5	4	8	10	2	6	9	12	11	3	7

Divide: O(1).

1	5	4	8	10	2

6	9	12	11	3	7

蓝色部分的逆序数？绿色部分的逆序数？

例子

1	5	4	8	10	2	6	9	12	11	3	7

Divide: O(1).

1	5	4	8	10	2

6	9	12	11	3	7

Conquer: 2T(n / 2)

5 blue-blue inversions

5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

例子

- 分治策略
  - 划分：把序列一分为二；
  - 分别递归求出左右各自部分的逆序数；
  - 组合: 计数左右部份之间的逆序数, 返回总的逆序数.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: O(1).

1	5	4	8	10	2

6	9	12	11	3	7

Conquer: 2T(n / 2)

5 blue-blue inversions

8 green-green inversions

9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Total = 5 + 8 + 9 = 22.

Efficient Combine: ???

例子

- 组合：计数蓝-绿逆序，假设左右部分已经排序，合并这两个部分为完整的排序。

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0

Count: O(n)

2	3	7	10	11	14	16	17	18	19	23	25

Merge: O(n)

$$T(n) \le T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + O(n) \quad \Rightarrow \quad \mathrm{T}(n) = O(n \log n)$$

算法

Merge-and-Count(A,B)

维护一个Current指针指向每个表，初始化指向首元素

维护一个变量Count用于逆序的计数，初始为0

While两个表都不空

  令a_i与b_j是由Current指针指向的元素

  把这两个表中较小的元素加到输出表中

  If b_j是较小的元素 then

    把Count加上在A中剩余的元素数

  Endif

  把较小元素选出的表中的Current指针前移

Endwhile

一旦一个表为空，把另一个表剩余的所有元素加到输出中

返回Count和合并后的表

算法

Sort-and-Count(L)

If这个表中有一个元素，then

 没有逆序

Else

 把这个表划分成两半，

 A包含前$\lceil n/2 \rceil$个元素

 B包含剩下的$\lfloor n/2 \rfloor$个元素

 (rA,A)=Sort-and-Count(A)

 (rB,B)=Sort-and-Count(B)

 (r,L)=Merge-and-Count(A,B)

Endif

返回r=rA+rB+r以及排好序的表L

算法分析

- 定理5.7 Sort-and-Count算法正确对输入表排序并且计数逆序个数；它对具有n个元素的表运行在O(nlogn)时间。

5.4 最邻近点对

- 问题：给定平面上的n个点，找最邻近的一对点
- 计算几何领域的基础
- 20世纪70年代， M. I. Shamos, D. Hoey, 关心**是否能找到一个渐进的比平方阶更快的算法**

最邻近点对

- 应用领域
  - Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
  - Special case of nearest neighbor, Euclidean MST, Voronoi.

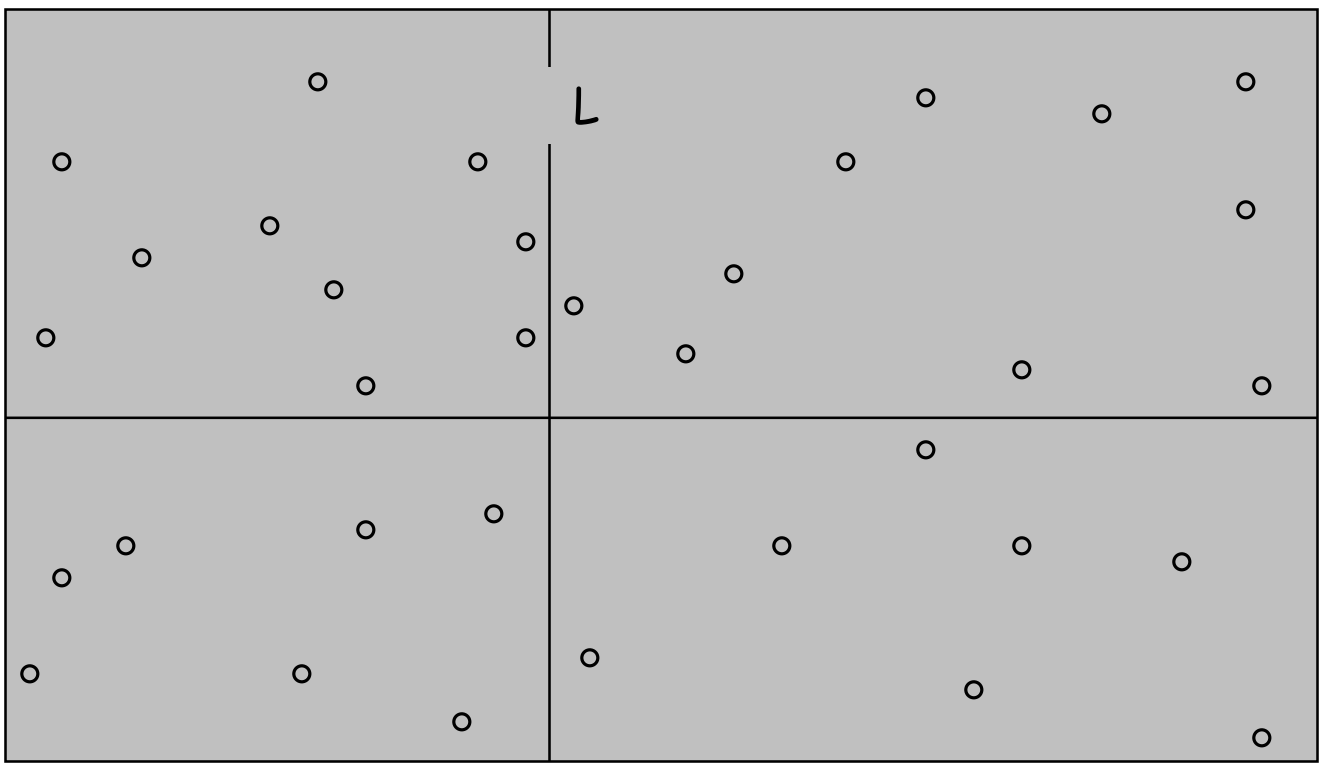
最邻近点对

可行性

- 特殊情形： 如果是所有的点在一个维度的直线上： O(nlogn)
- 为了使得后面的描述相对简单，假设这些点的x坐标都不相同

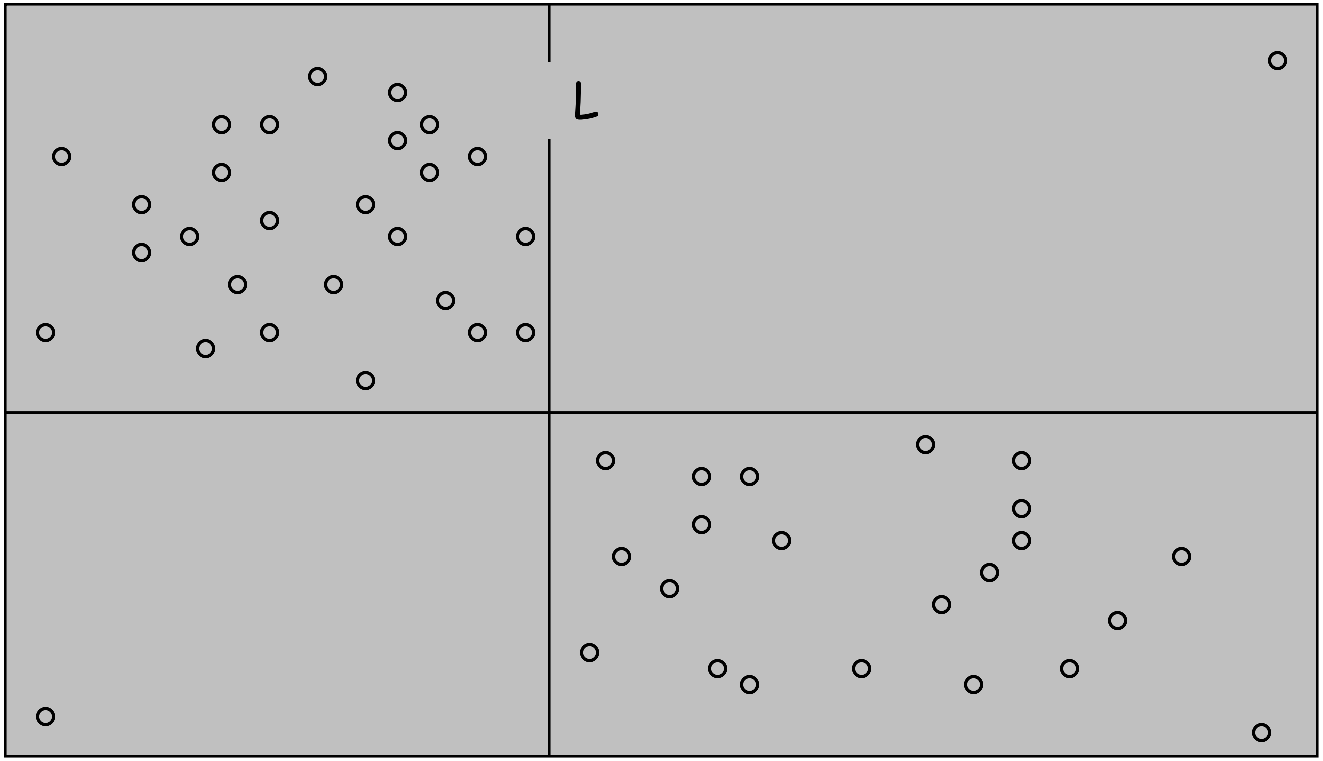
例子

- 考虑一种划分：把一块平面区域分成**四**个部分



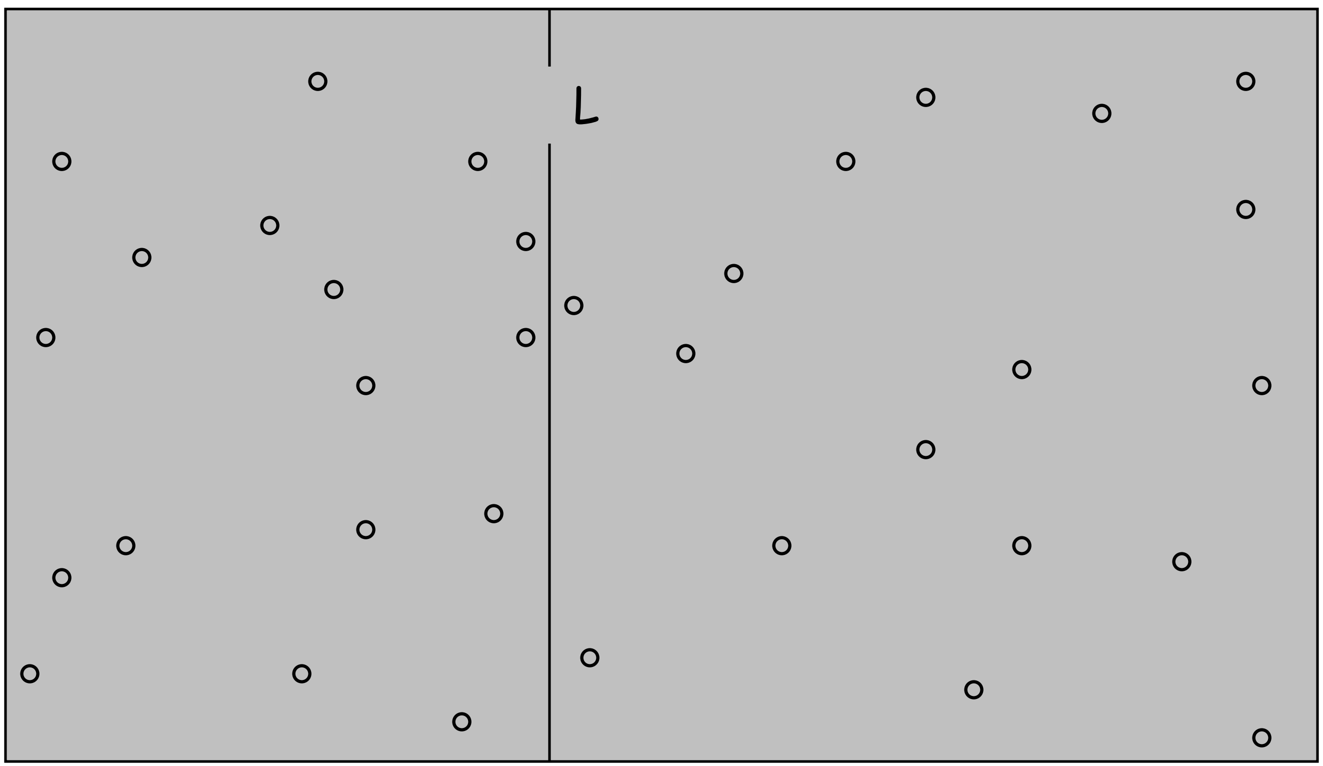
例子

- 这样的划分可能会带来的问题：不均匀



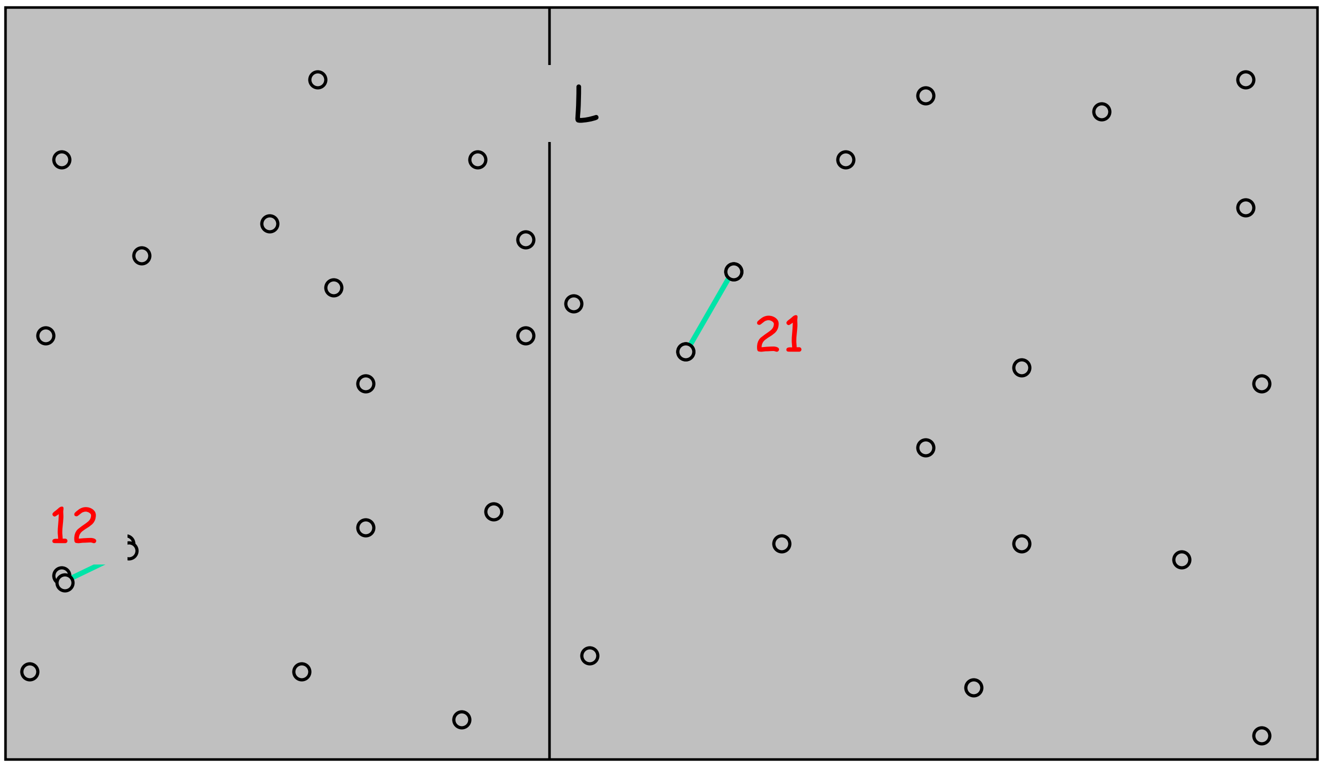
分治算法

- 划分的时候：用一条垂直线把点集分成相等的两部分



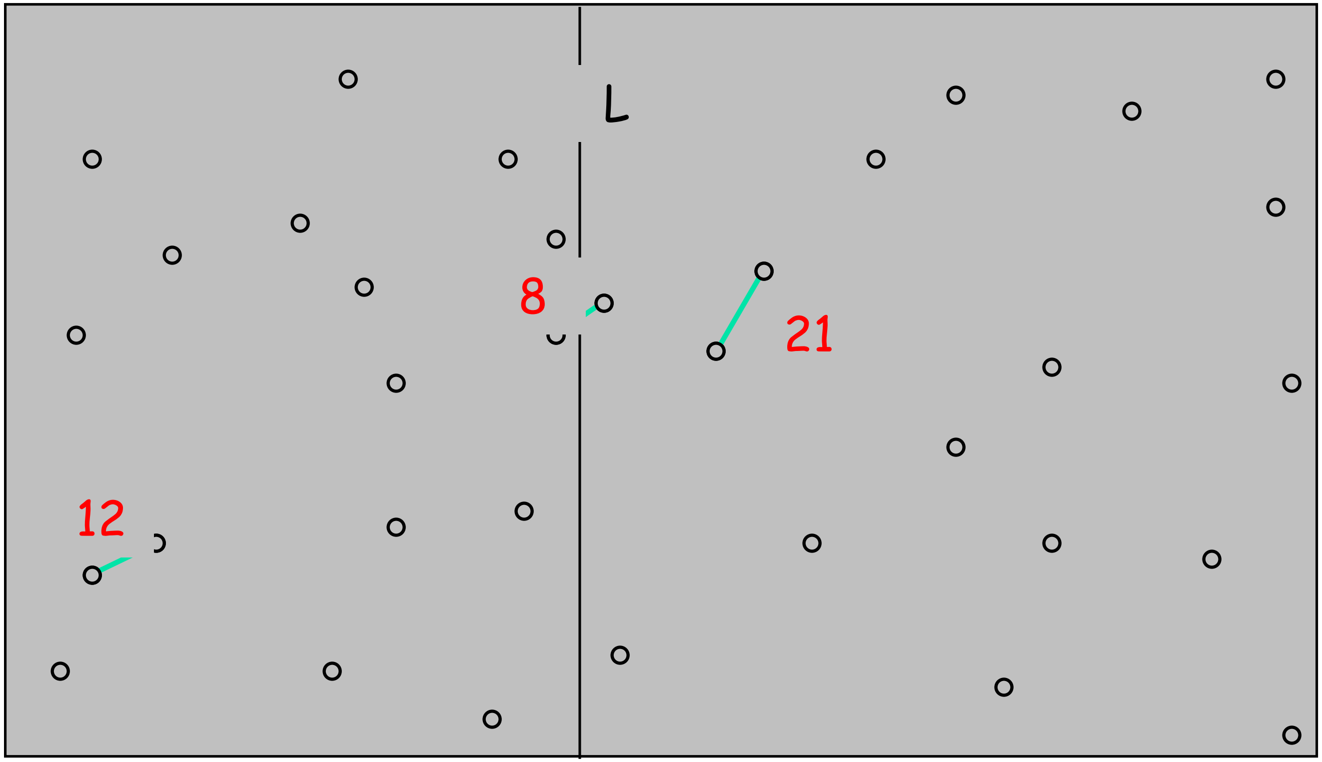
分治算法

- 在左右半部分寻找各自最邻近的点对



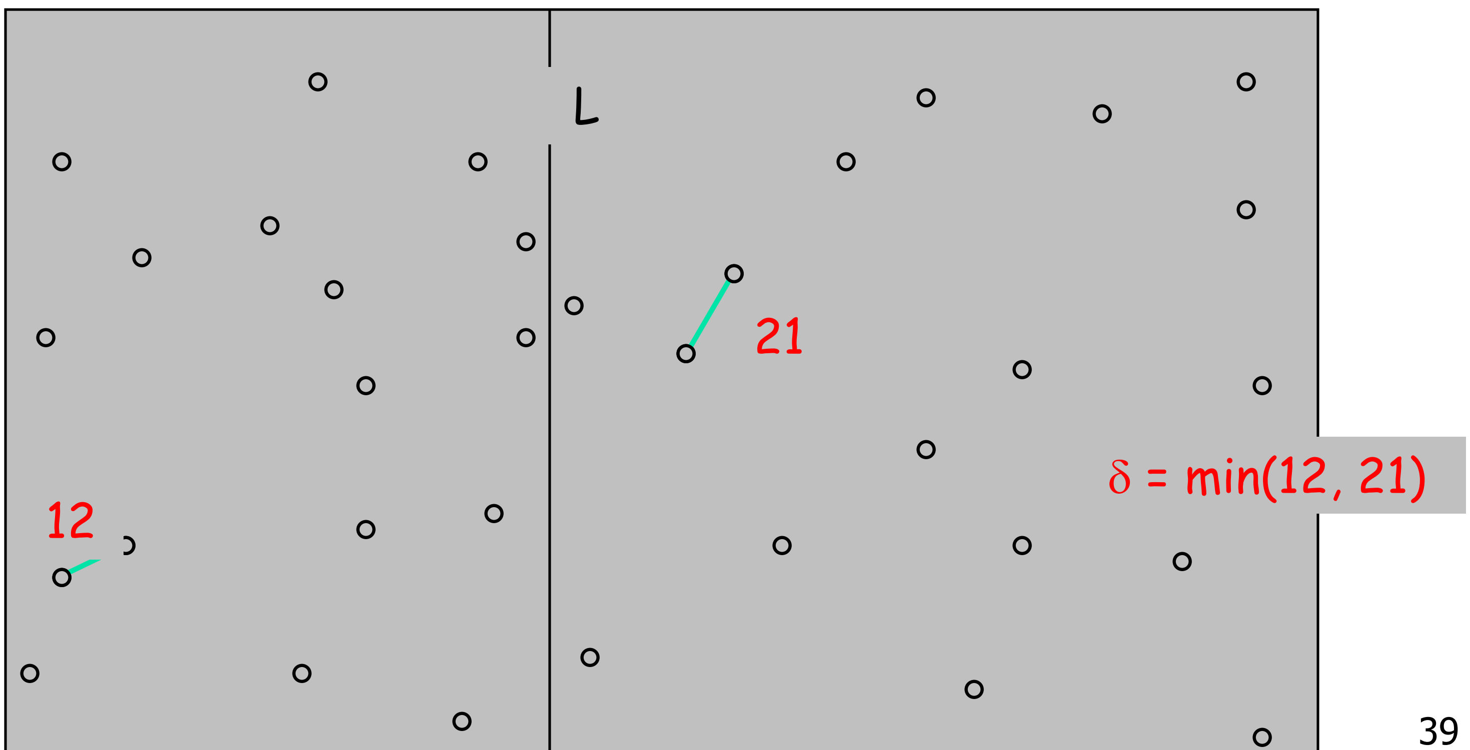
分治算法

- 组合：在“交界”的边中寻找最近的点对
- 最后返回上面3个解中的最优解



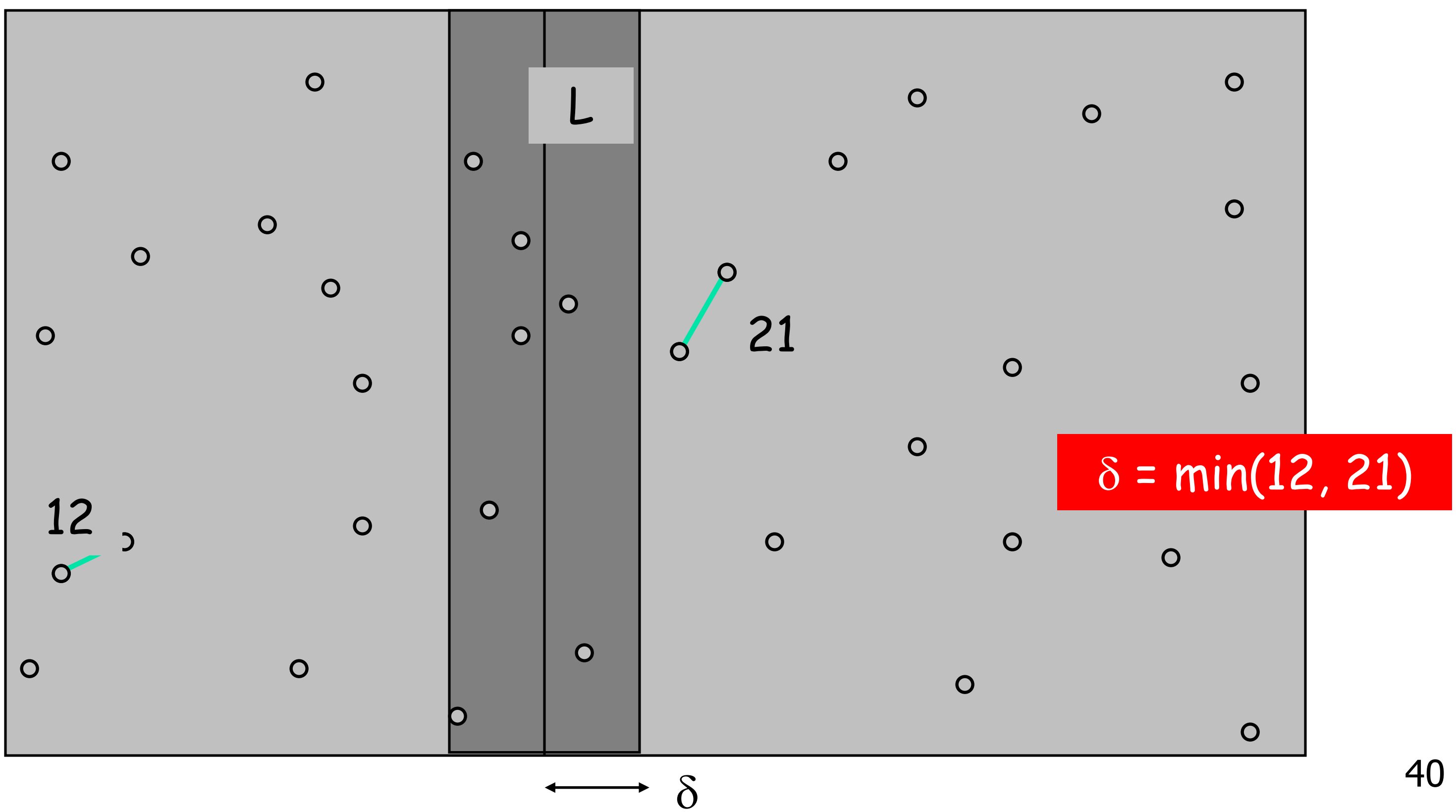
分治算法

- 问题的关键在于：寻找“交界”的边中比 $\delta = \min(\delta_L, \delta_R)$ 还要近的点对。



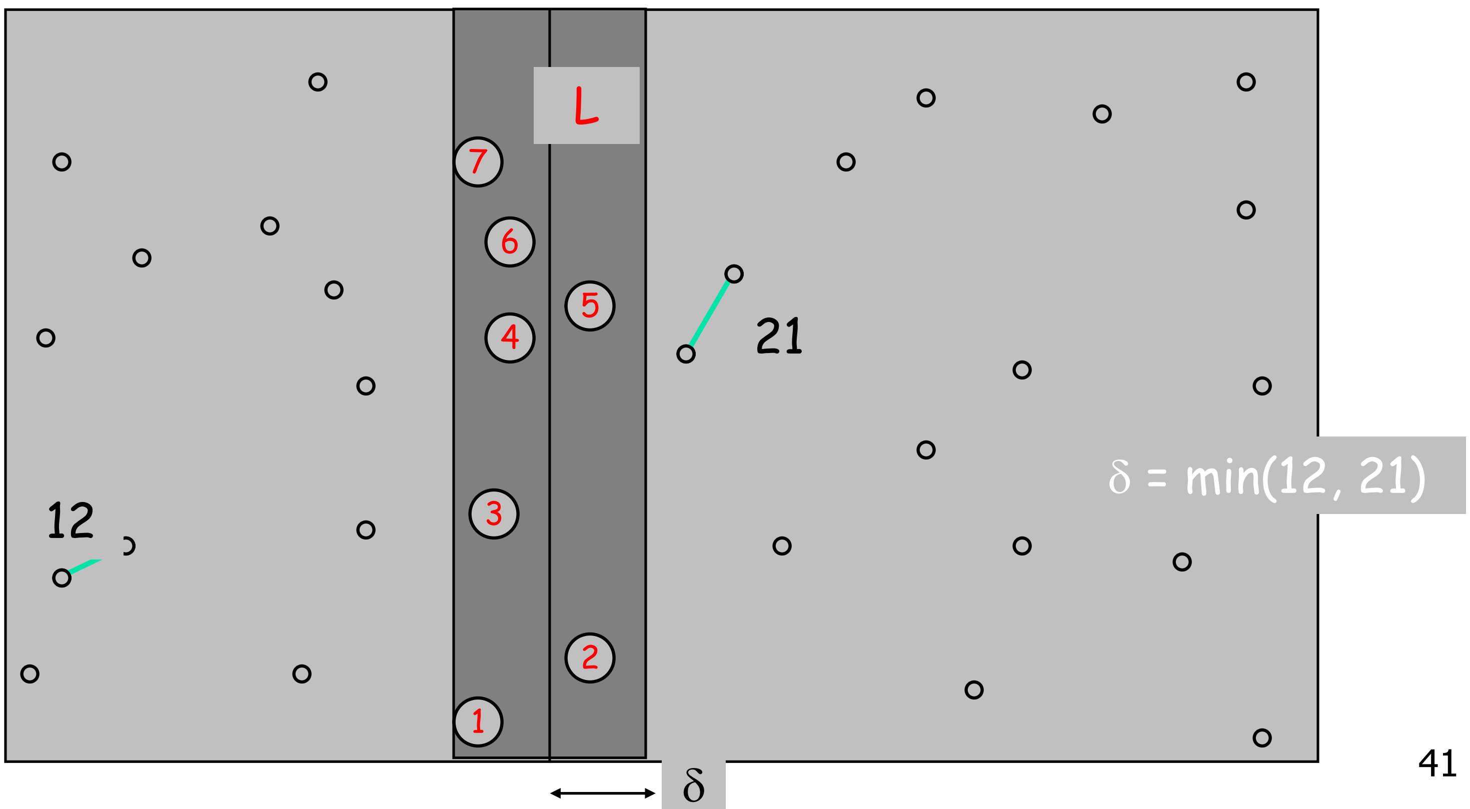
分治算法

- 于是根据**平面几何的约束性质**，只需要考虑分界线附近的点集情况。



分治算法

- 把“窄带”之中的点集按照y坐标的大小排序编号



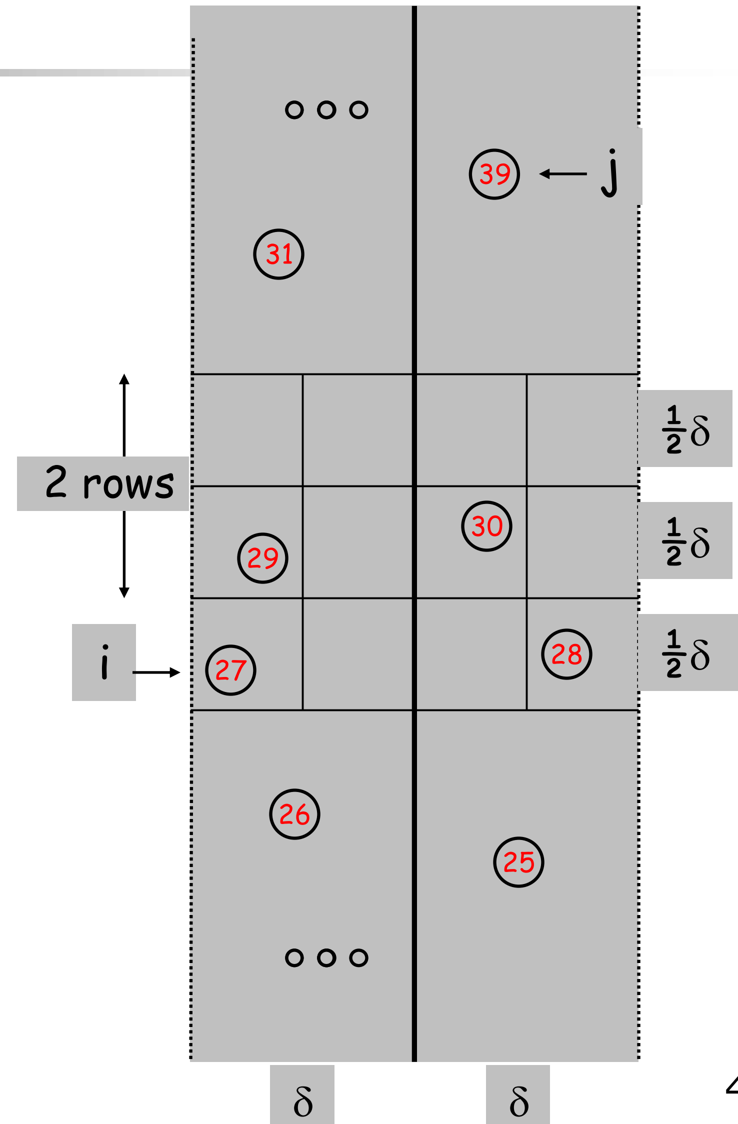
分治算法

- 定义 设s_i 是2δ窄带中按照y坐标排序的第i个点。
- 命题. 如果 $|i - j| \geq 12$, 那么s_i与s_j 之间的距离至少是δ.
- 证明.
  - 在$\frac{1}{2}\delta \times \frac{1}{2}\delta$的小盒子中，至多只有一个点.
  - s_i与s_j 之间至少可以间隔两行，距离$\geq 2(\frac{1}{2}\delta)$.

分治算法

注意到这里目的是为了说明不需要在窄带之间进行两两之间点的比较；只需要在一个常数阶的范围内比较就可以。

比如12还可以变成7，…



分治算法

```
Closest-Pair(p₁, …, pₙ) {
   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ₁ = Closest-Pair(left half)
   δ₂ = Closest-Pair(right half)
   δ  = min(δ₁, δ₂)

   Delete all points further than δ from separation line L

   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance between
   each point and next 11 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

Compute separation line L	$O(n \log n)$
Closest-Pair(left half), Closest-Pair(right half)	$2T(n / 2)$
Delete all points further than δ	$O(n)$
Sort remaining points	$O(n \log n)$
Scan points	$O(n)$

分治算法

- 运行时间：

$$T(n) \le 2T(n/2) + O(n \log n)$$

复杂度？

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

分治算法

- 是否存在O(n log n)的算法?

- 上述算法在窄带之中的排序消耗了O(nlogn); 所以希望在窄带中能够不必从头开始排序

- 通过预处理，可以在O(n)的时间内实现窄带中的排序

分治算法

- Closest-Pair(P)

  构造P_x与P_y（O(nlogn)）

  (po*,p1*)=Closest-Pair-Rec(P_x,P_y)

分治算法

Closest-Pair-Rec(Px,Py)

```
If |P|<=3 then
   进行简单穷举排序
End if

构造Qx,Qy,Rx,Ry(O(n)时间)
 （q0*,q1*）=Closest-Pair-Rec(Qx,Qy)
 （r0*,r1*）=Closest-Pair-Rec(Rx,Ry)

Delta=min(d(qo*,q1*),（r0*,r1*））
x*=集合Q中的点的最大的x坐标
L={(x,y):x=x*}
S=P中与L相距在Delta之内的点集
```

分治算法

构造Sy(O(n)时间)

For每个点Sy中的点s，计算从s到跟着Sy的15（12）个点中每个点的距离

令s,s'是达到其中最小距离的点对(O(n)时间)

```
If d(s,s')<Delta then
  Return (s,s')
Else if d(q0*,q1*)<d(r0*,r1*) then
  Return (q0*,q1*)
Else
  Return (r0*,r1*)
End if
```

分治算法

- 定理5.12 上面修改后的分治算法的运行时间是O(nlogn).
- 证明：

$$T(n) \le 2T(n/2) + O(n) \quad \Rightarrow \quad \mathrm{T}(n) = O(n \log n)$$

5.5 整数乘法

- 考虑两个整数运算的问题
- 加法：给定两个n位的整数a,b,求a+b
- O(n) bit operations

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

加法

- 乘法？给定两个n位的整数a,b,，求a*b

整数乘法

- 最直观的乘法
  - $\Theta(n^2)$ bit operations.

```
                  1 1 0 1 0 1 0 1
                * 0 1 1 1 1 1 0 1
                -----------------
                1 1 0 1 0 1 0 1 0
              0 0 0 0 0 0 0 0 0
            1 1 0 1 0 1 0 1 0
          1 1 0 1 0 1 0 1 0
        1 1 0 1 0 1 0 1 0
      1 1 0 1 0 1 0 1 0
    1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
---------------------------------
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0
```

乘法

整数乘法

- 有没有更好的办法？
- 采用分治策略：每个整数分成高位，低位

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0
\end{aligned}
$$

算法有没有改进？

$$
\mathrm{T}(n) = \underbrace{4T\left(n/2\right)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow \mathrm{T}(n) = \Theta(n^2)
$$

整数乘法

- 改进的目标
- 能否把4次乘法的次数减少？
- 注意到

$$(x_1 + x_0)(y_1 + y_0) = x_1 y_1 + x_1 y_0 + x_0 y_1 + x_0 y_0$$

正好是上面四个乘法之和

整数乘法

$$
\begin{aligned}
x &= 2^{n/2}\cdot x_1 + x_0 \\
y &= 2^{n/2}\cdot y_1 + y_0 \\
xy &= 2^{n}\cdot x_1y_1 + 2^{n/2}\cdot\left(x_1y_0 + x_0y_1\right) + x_0y_0 \\
&= 2^{n}\cdot \underbrace{x_1y_1}_{A} + 2^{n/2}\cdot\left(\underbrace{(x_1+x_0)(y_1+y_0)}_{B} - \underbrace{x_1y_1}_{A} - \underbrace{x_0y_0}_{C}\right) + \underbrace{x_0y_0}_{C}
\end{aligned}
$$

所以可以把4个乘法变成3个！

整数乘法

- Recursive-Multiply(x,y)

令 $x = x_1 2^{n/2} + x_0, y = y_1 2^{n/2} + y_0$

计算x1+x0与y1+y0

p=Recursive-Multiply(x1+x0, y1+y0)

x1y1= Recursive-Multiply(x1,y1)

x0y0= Recursive-Multiply(x0,y0)

Return $x_1 y_1 2^n + (p - x_1 y_1 - x_0 y_0) 2^{n/2} + x_0 y_0$

分析算法

- 定理5.13 **[Karatsuba-Ofman, 1962]** Recursive-Multiply算法在两个n位因数上的运行时间是 $O(n^{\log_2 3}) = O(n^{1.59})$.
- 证明：

$$\mathrm{T}(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}}$$

$$\Rightarrow \mathrm{T}(n) = O(n^{\log_2 3}) = O(n^{1.585})$$

矩阵乘积

- 问题：给定两个n×n矩阵A,B,求C=AB.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \qquad \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

- 如果采用直接计算，那么复杂度为$\Theta(n^3)$ 次算术运算(arithmetic operations).
- 存不存在更有效的办法？

矩阵乘积

- 采用分治策略
  - 先把A,B分成$\frac{1}{2}$n×$\frac{1}{2}$n 的小块；
  - 分别递归计算8块$\frac{1}{2}$n×$\frac{1}{2}$n矩阵乘积；
  - 组合：把结果通过4次矩阵加法组合产生。

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

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$\mathrm{T}(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow \mathrm{T}(n) = \Theta(n^3)$$

矩阵乘积

- 问题的关键在于能否*减少乘法*的次数

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

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

7 次乘法
18 = 10 + 8 次加减法

矩阵乘积

- 矩阵快速乘积运算. (Strassen, 1969)
  - 先把A,B分成$\frac{1}{2}n \times \frac{1}{2}n$ 的小块;
  - 计算：通过10次加减法运算，生成14 个$\frac{1}{2}n \times \frac{1}{2}n$矩阵.
  - 分治：递归生成7个$\frac{1}{2}n \times \frac{1}{2}n$ 的矩阵乘积.
  - 组合：7个矩阵乘积通过8次矩阵的加减法生成需要的四个矩阵乘积

矩阵乘积

- 分析
  - 假设 n 是 2的整数次幂.
  - T(n) = # arithmetic operations.

$$\mathrm{T}(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \quad \Rightarrow \quad \mathrm{T}(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

注意到这是一个理论上的界，n 比较大的时候才能体现出Strassen算法的优势；比如，n ~ 2,500， Strassen算法在苹果机上的运行速度是常规算法的八十多倍

矩阵乘积

- 两个2×2分块矩阵乘法能否用7次向量乘法实现？
- **可以** [Strassen, 1969] $\Theta(n^{\log_2 7}) = O(n^{2.81})$
- 两个2×2分块矩阵乘法2×2矩阵能否用6次向量乘法实现？
- **不可能** [Hopcroft and Kerr, 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$
- 两个 3×3分块矩阵乘法能否用21次向量乘法实现？
- **不可能** $\Theta(n^{\log_3 21}) = O(n^{2.77})$
- 两个 70×70分块矩阵乘法能否用143,640次向量乘法实现?
- **可以！** [Pan, 1980] $\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$

矩阵乘积

- 复杂度最低记录：
  - December, 1979: $O(n^{2.521813})$.
  - January, 1980: $O(n^{2.521801})$.
  - 1987, Coppersmith-Winograd, $O(n^{2.376})$
  - 2020 年 10 月， Alman， Vassilevska Williams： $O(n^{2.3728596})$

- 猜想：对于任意的 $\varepsilon > 0$，存在 $O(n^{2+\varepsilon})$的优化算法.

- 类Strassen算法上每获得一点理论上的改进，实际上的可用性就会降低。 （需要n达到一定规模）

5.6 卷积与快速傅立叶变换

- 应用场景
  - signal processing, speech recognition, data compression, image processing.
  - DVD, JPEG, MP3, …
  - Numerical solutions to Poisson's equation

卷积

- 傅立叶变换在数字计算机出现以后，才引起了足够的重视

The FFT is one of the truly great computational developments of 20th century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT. - *Charles van Loan*

多项式系数表示

- 多项式 [系数表示法]

$$A(x)=a_0+a_1x+a_2x^2+\cdots+a_{n-1}x^{n-1}$$

$$B(x)=b_0+b_1x+b_2x^2+\cdots+b_{n-1}x^{n-1}$$

求值?

加法?

乘法?

多项式系数表示

- 求值： O(n) (采用Horner方法)

$$A(x) = a_0 + (x(a_1 + x(a_2 + \cdots + x(a_{n-2} + x(a_{n-1}))\cdots))$$

- 加法：(O(n) *arithmetic operations*)

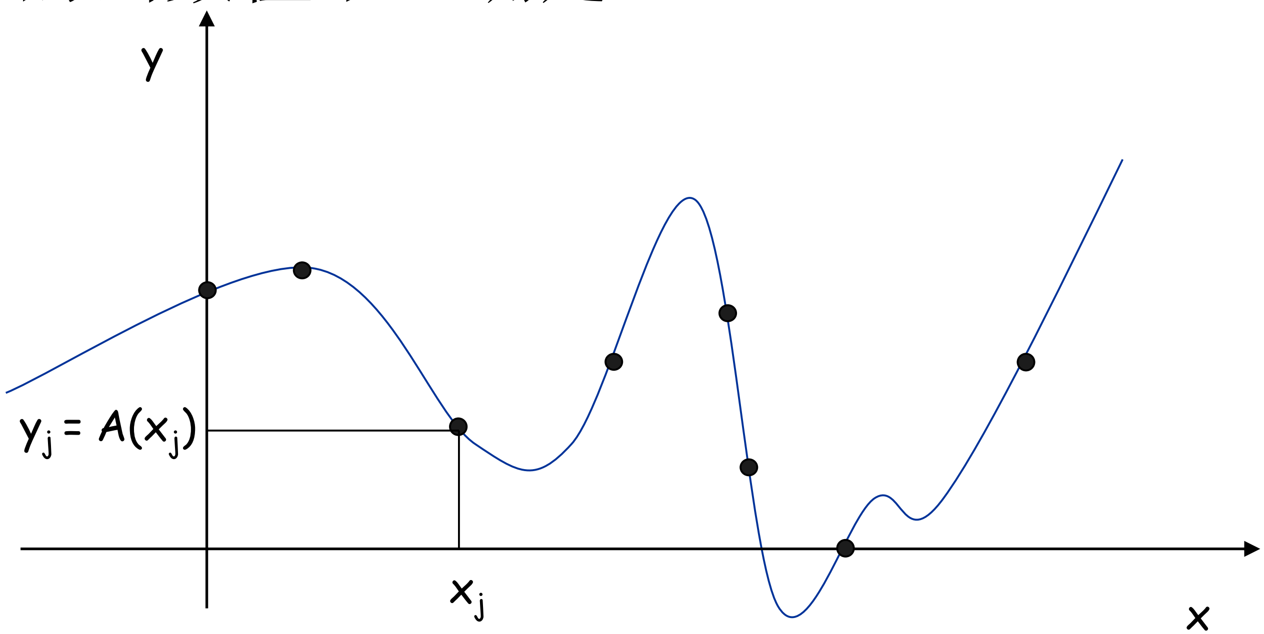
$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1}$$

- 乘法(卷积)： 采用直接算法O(n²).

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

多项式点值表示

- 代数学基本定理. [Gauss] n次复数多项式存在n个复数根.
- 推论. n-1次多项式A(x)能够被n个不同 的x处的函数值唯一确定.



多项式点值表示

- 多项式. [点值表示]

$$A(x):\ (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$$

$$B(x):\ (x_0,\ z_0), \ldots, (x_{n-1}, z_{n-1})$$

加法?

乘法?

求值?

多项式点值表示

- 加法： O(n) arithmetic operations.

$$A(x)+B(x): \quad (\mathrm{x}_0, y_0+z_0), \ldots, (\mathrm{x}_{\text{n-1}}, y_{n-1}+z_{n-1})$$

- 乘法： O(n), 需要 **2n-1** 个点

$$A(x)\times B(x): \quad (\mathrm{x}_0, y_0\times z_0), \ldots, (\mathrm{x}_{\text{2n-1}}, y_{2n-1}\times z_{2n-1})$$

- 求值： O(n^2) ,

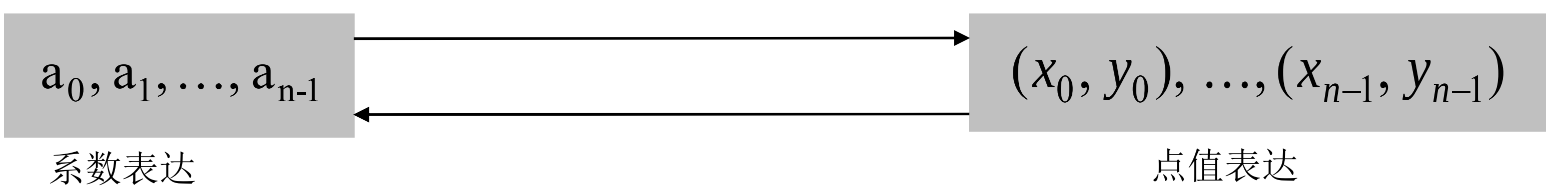
$$A(x)=\sum_{k=0}^{n-1} y_k \frac{\prod_{j\neq k}(x-x_j)}{\prod_{j\neq k}(x_k-x_j)}$$

两种表达之间的关系

- 多项式表达，点值表达

Representation	乘法	求值
系数表达	$O(n^2)$	$O(n)$
点值表达	$O(n)$	$O(n^2)$

- 希望能够找到一种新的表达方式，能够在这两种表达之间进行高效的转换



两种表达之间的关系

- **系数表达到点值的表达**

给定多项式$a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, 求它在n 个不同点 $x_0, \ldots, x_{n-1}$处的值

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$O(n^2)$ for matrix-vector multiply

Vandermonde matrix is invertible iff x_i distinct

两种表达之间的关系

- **点值表达到系数表达**

给定n个不同的 $x_0, \ldots, x_{n-1}$ 和其值 $y_0, \ldots, y_{n-1}$, 寻找满足条件的多项式$a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ 使得在这些点上符合函数值.

---涉及到Vandermonde矩阵求逆

如果对Vandermonde矩阵求逆，有人展开过这方面的研究，代价是$O(n^2)$.

分治策略

- 系数表达到点值表达
- 采用分治策略：
  把多项式表达分成**奇偶**部分
  - $A(x) \quad = \ a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$
  - $A_{even}(x) \ = \ a_0 + a_2x + a_4x^2 + a_6x^3.$
  - $A_{odd}(x) \ = \ a_1 + a_3x + a_5x^2 + a_7x^3.$
  - $\mathbf{A(\ x) = A_{even}(x^2) + x\, A_{odd}(x^2).}$
  - $\mathbf{A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2).}$

分治策略

- 尝试：考虑特殊点值x=±1.
  - $A(\ 1) = A_{even}(1) + 1\ A_{odd}(1)$.
  - $A(-1) = A_{even}(1) - 1\ A_{odd}(1)$.

如果计算两个次数小于等于$\frac{1}{2}$n 的多项式在一点(x=1)的值，那么可以确定一个次数小于等于n的多项式在两点的值(x=±1).

分治策略

- 另一次尝试：考虑特殊点值x=±1，±i.
  - $A(\ 1) = A_{even}(\ 1) + 1\ A_{odd}(\ 1)$.
  - $A(-1) = A_{even}(\ 1) - 1\ A_{odd}(\ 1)$.
  - $A(\ i) = A_{even}(-1) + i\ A_{odd}(-1)$.
  - $A(-i) = A_{even}(-1) - i\ A_{odd}(-1)$.

如果计算2个次数小于等于$\frac{1}{2}n$ 的多项式在两点(x=1，-1)的值，那么可以确定一个次数小于等于n的多项式在4点的值(x=±1, ±i).

分治策略

系数表达到点值表达：给定多项式$a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, 求它在n 个不同点$x_0, \ldots, x_{n-1}$处的值

- 关键之处：选择$x_k = \omega^k$，其中ω是n次单位根。

分治策略

$$
\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}
$$

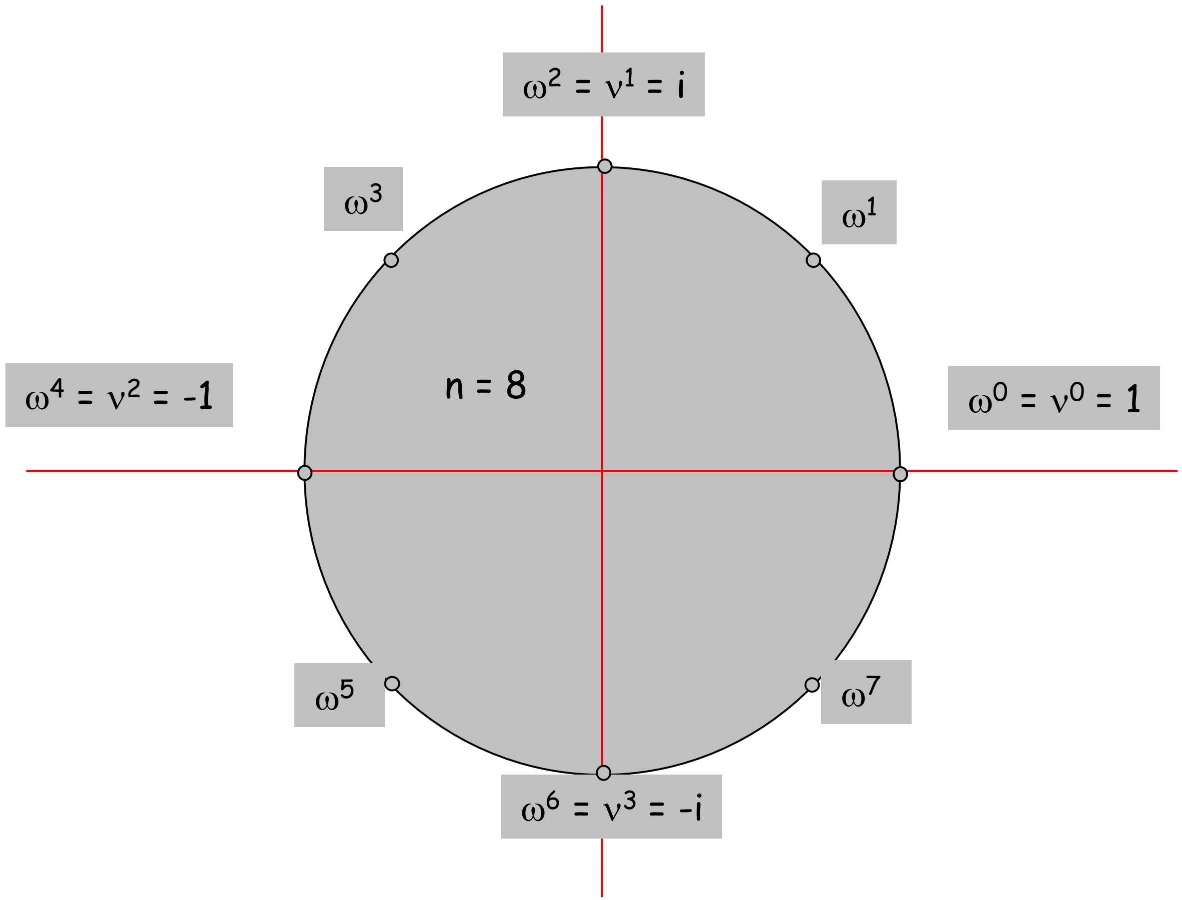
Discrete Fourier transform

Fourier matrix F_n

单位根的性质

- 定义：如果复数x满足$x^n = 1$，那么x被称为n次单位根.
- n次单位根可以表示为$\omega^0, \omega^1, \ldots, \omega^{n-1}$，其中 $\omega = e^{2\pi i / n}$.
- 如果n是偶数，那么n/2次单位根可以表示为$\nu^0, \nu^1, \ldots, \nu^{n/2-1}$，其中$\nu = e^{4\pi i / n}$.

单位根的性质


$\omega^2 = \nu^1 = i$
ω^3
ω^1
$\omega^4 = \nu^2 = -1$
n = 8
$\omega^0 = \nu^0 = 1$
ω^5
ω^7
$\omega^6 = \nu^3 = -i$

快速傅立叶变换

- 目标：求$A(x) = a_0 + \ldots + a_{n-1} x^{n-1}$ 在n次单位根$\omega^0, \omega^1, \ldots, \omega^{n-1}$处的值。
- 分治策略：
- 把多项式分成偶数部分和奇数部分
  - $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n/2-2} x^{(n-1)/2}$.
  - $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n/2-1} x^{(n-1)/2}$.
  - $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.

快速傅立叶变换

- 分别计算：计算多项式$A_{even}(x)$， $A_{odd}(x)$ 在 $\frac{1}{2}n^{th}$单位根$\nu^0, \nu^1, \ldots, \nu^{n/2-1}$处的值.
- 组合：
  - $A(\omega^k) = A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2$
  - $A(\omega^{k+n/2}) = A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2$

$\omega^{k+n/2} = -\omega^k$

$\nu^k = (\omega^k)^2 = (\omega^{k+n/2})^2$

快速傅立叶变换

- FFT算法

```
fft(n, a₀,a₁,…,aₙ₋₁) {
   if (n == 1) return a₀

   (e₀,e₁,…,e_{n/2-1}) ← FFT(n/2, a₀,a₂,a₄,…,aₙ₋₂)
   (d₀,d₁,…,d_{n/2-1}) ← FFT(n/2, a₁,a₃,a₅,…,aₙ₋₁)

   for k = 0 to n/2 - 1 {
      ωᵏ ← e^{2πik/n}
      yₖ     ← eₖ + ωᵏ dₖ
      y_{k+n/2} ← eₖ - ωᵏ dₖ
   }

   return (y₀,y₁,…,yₙ₋₁)
}
```

快速傅立叶变换

- 定理．FFT 算法能够在O(n log n) 步骤内计算n-1次多项式在每个n次单位根处的值(n是2的整数次幂)。

- 运行时间

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

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

快速傅立叶变换

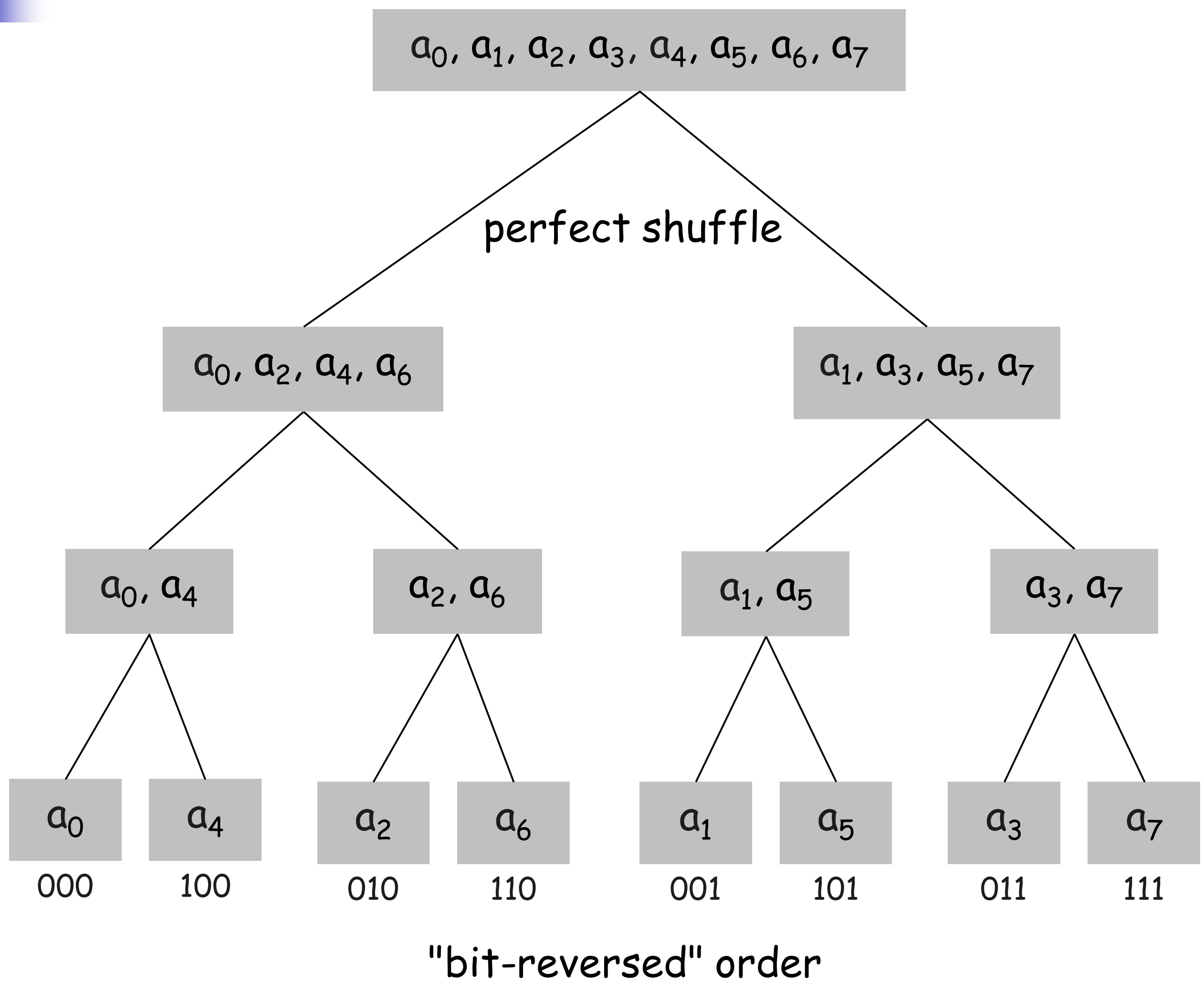
O(n log n)

$$a_0, a_1, \ldots, a_{n-1} \rightleftarrows (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$$

系数表达

点值表达

快速傅立叶变换



$a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7$
perfect shuffle
a_0, a_2, a_4, a_6
a_1, a_3, a_5, a_7
a_0, a_4
a_2, a_6
a_1, a_5
a_3, a_7
a_0
a_4
a_2
a_6
a_1
a_5
a_3
a_7
000
100
010
110
001
101
011
111
"bit-reversed" order

傅立叶变换逆变换

- 目标. 给定n-1次多项式在n个点$\omega^0, \omega^1, \ldots, \omega^{n-1}$处的点值$y_0, \ldots, y_{n-1}$, 给出满足条件的唯一的多项式 $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Inverse DFT

Fourier matrix inverse $(F_n)^{-1}$

傅立叶变换逆变换

- 命题：傅立叶变换逆变换矩阵形式如下

$$G_n = \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

傅立叶变换逆变换

- 求和引理：设ω 是n次单位根的**生成元**，那么 $\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} n & \text{if } k \equiv 0 \bmod n \\ 0 & \text{otherwise} \end{cases}$.

- 命题: $F_n \times G_n = I_n$

- 证明: $(F_n G_n)_{kk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj} \omega^{-jk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$

傅立叶变换逆变换

- 发现表达形式与前面的算法计算的对象(DFT)非常类似

- 如何计算傅立叶变换逆变换？

应用同样的算法；同时注意把
$\omega^{-1} = e^{-2\pi i/n}$ 看成是 n 次单位根的生成元.

IFFT快速算法

```
ifft(n, a₀,a₁,…,aₙ₋₁) {
   if (n == 1) return a₀

   (e₀,e₁,…,e_{n/2-1}) ← IFFT(n/2, a₀,a₂,a₄,…,aₙ₋₂)
   (d₀,d₁,…,d_{n/2-1}) ← IFFT(n/2, a₁,a₃,a₅,…,aₙ₋₁)

   for k = 0 to n/2 - 1 {
      ωᵏ ← e^{-2πik/n}
      yₖ       ← (eₖ + ωᵏ dₖ) / n
      y_{k+n/2} ← (eₖ - ωᵏ dₖ) / n
   }

   return (y₀,y₁,…,yₙ₋₁)
}
```

IFFT快速算法

- 定理：IFFT算法能够在O(n log n)步骤内，根据n次单位根的值，计算出n-1次多项式的系数.（n是2的整数次幂）

$$a_0, a_1, \ldots, a_{n-1} \quad \underset{O(n \log n)}{\overset{O(n \log n)}{\rightleftarrows}} \quad (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$$

系数表达　　　　点值表达

卷积

- 给定向量 $a=(a_0,a_1,\dots a_{n-1}),b=(b_0,b_1,\dots b_{n-1})$,把它们看成多项式$A(x)=a_0+a_1x+a_2x^2+\cdots+a_{n-1}x^{n-1}$, $B(x)=b_0+b_1x+b_2x^2+\cdots+b_{n-1}x^{n-1}$
  对于多项式$C(x)=A(x)B(x)$,可以知道 $c=(c_0,c_1,\dots c_{2n-2})$
  就是卷积a*b.

- 定理5.15 使用快速傅立叶变换，可以在O(n log n) 时间内计算初始向量a和b的卷积。

卷积

系数表达

$a_0, a_1, \ldots, a_{n-1}$
$b_0, b_1, \ldots, b_{n-1}$

FFT

O(n log n)

点值表达

$A(x_0), \ldots, A(x_{2n-1})$
$B(x_0), \ldots, B(x_{2n-1})$

point-value multiplication

O(n)

$C(x_0), C(x_1), \ldots, C(x_{2n-1})$

点值表达

inverse FFT

O(n log n)

系数表达

$c_0, c_1, \ldots, c_{2n-2}$

应用

- **计算两个n位整数的乘法:** $a = a_{n-1} \cdots a_1 a_0$
  以及 $b = b_{n-1} \cdots b_1 b_0$
  生成两个n次多项式: $p(x) = \sum_{j=0}^{n-1} a_j x^j$ $q(x) = \sum_{j=0}^{n-1} b_j x^j$
  注意到 a = p(10), b = q(10).
  求值 r(10) = a*b
- 采用 FFT： O(n log n).

推广

- 对于所有的正整数n，都存在复杂度为O(nlogn)的快速傅里叶变换算法
- MIT， Frigo and Johnson, 完成了优化的FFT变换的C库文件， http://www.fftw.org