本周教学内容

分治算法的设计思想与分析方法

改进分治算法的途径1

改进分治算法的途径2

芯片测试

幂乘算法

快速排序

分治算法的一般描述
分治算法的分析方法

分治算法的设计思想：
二分检索、二分归并排序、Hanoi塔

分治策略
的设计思想

分治策略的基本思想

分治策略（ Divide and Conquer ）

1. 将原始问题划分或者归结为规模较小的子问题
1. 递归或迭代求解每个子问题
2. 将子问题的解综合得到原问题的解

注意：

1. 子问题与原始问题性质完全一样
2. 子问题之间可彼此独立地求解
3. 递归停止时子问题可直接求解

二分检索

算法 Binary Search (T, l, r, x)

输入：数组 T，下标从 l 到 r；数 x

输出：j // 若x在T 中, j 为下标; 否则为 0

```
l←1;  r←n
while l≤r do
    m←⌊(l +r)/2⌋   // m为中间位置
    if T[m]=x then return m // x是中位数
    else if T[m]>x then r←m−1
         else l←m+1
return 0
```

二分检索算法设计思想

- 通过 x 与中位数的比较，将原问题归结为规模减半的子问题，如果 x 小于中位数，则子问题由小于 x 的数构成，否则子问题由大于 x 的数构成.
- 对子问题进行二分检索.
- 当子问题规模为 1 时，直接比较 x与 $T[m]$，若相等则返回 m，否则返回 0.

是否能够递归实现?

二分检索时间复杂度分析

二分检索问题最坏情况下时间复杂度

$$W(n) = W(\lfloor n/2 \rfloor)+1$$

$$W(1) = 1$$

可以解出

$$W(n)=\lfloor \log n \rfloor + 1$$

二分归并排序

算法 **Merge Sort** (A, p, r)

输入：数组 $A[p .. r]$

输出：元素按从小到大排序的数组 A

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor (p + r)/2 \rfloor$ 对半划分
3. **Merge Sort** (A, p, q) 子问题1
4. **Merge Sort** $(A, q+1, r)$ 子问题 2
5. **Merge** (A, p, q, r) 综合解

二分归并排序设计思想

- 划分将原问题归结为规模为 $n/2$ 的 2 个子问题
- 继续划分，将原问题归结为规模为 $n/4$ 的 4 个子问题. 继续…，当子问题规模为1 时，划分结束.
- 从规模 1到 $n/2$，陆续归并被排好序的两个子数组. 每归并一次，数组规模扩大一倍，直到原始数组.

二分归并排序时间复杂度分析

假设n为2的幂，二分归并排序最坏情况下时间复杂度

$$W(n) = 2W(n/2) + n-1$$

$$W(1) = 0$$

可以解出

$$W(n) = n\log n - n + 1$$

Hanoi塔的递归算法

算法 **Hanoi** (A, C, n) // n个盘子A到C

1. **if** n=1 **then move** (A, C) //1个盘子A到C
2. **else** **Hanoi** $(A, B, n-1)$
3. **move** (A, C)
4. **Hanoi** $(B, C, n-1)$

设 n个盘子的移动次数为 $T(n)$

$$T(n) = 2\,T(n-1) + 1,$$
$$T(1) = 1,$$
$$T(n)=2^n-1$$

算法设计思想

- 将原问题归结为规模为 ***n*-1** 的2个子问题.
- 继续归约，将原问题归结为规模为 ***n*-2** 的 **4** 个子问题. 继续...， 当子问题规模为**1** 时，归约过程截止.
- 从规模 **1**到 ***n*-1**，陆续组合两个子问题的解. 直到规模为***n***.

小结

通过几个例子展示分治算法的特点：

- 将原问题归约为规模小的子问题，子问题与原问题具有相同的性质.
- 子问题规模足够小时可直接求解.
- 算法可以递归也可以迭代实现.
- 算法的分析方法：递推方程.

分治算法的一般描述和分析方法

分治算法的一般性描述

分治算法 **Divide-and-Conquer(P)**

1. **if** $|P| \le c$ **then** $S(P)$
2. **divide** P **into** $P_1, P_2, \ldots, P_k$
3. **for** $i \leftarrow 1$ **to** k
4. $\quad y_i \leftarrow$ **Divide-and-Conquer(P_i)**
5. **Return Merge** $(y_1, y_2, \ldots, y_k)$

划分

求解子问题

综合解

设计要点

- 原问题可以划分或者归约为规模较小的子问题
  - 子问题与原问题具有相同的性质
  - 子问题的求解彼此独立
  - 划分时子问题的规模尽可能均衡
- 子问题规模足够小时可直接求解
- 子问题的解综合得到原问题的解
- 算法实现：递归或迭代

分治算法时间分析

时间复杂度函数的递推方程

$$W(n)=W(|P_1|)+W(|P_2|)+\dots+W(|P_k|)+f(n)$$
$$W(c)=C$$

- $P_1, P_2, \dots, P_k$ 为 划分后产生的 子问题
- $f(n)$为划分子问题以及将子问题的解综合得到原问题解的总工作量
- 规模为c的最小子问题的工作量为C

两类常见的递推方程

$$f(n)=\sum_{i=1}^{k}a_i f(n-i)+g(n) \qquad (1)$$

$$f(n)=af(\frac{n}{b})+d(n) \qquad (2)$$

例子：

Hanoi塔，　$W(n)=2W(n-1)+1$

二分检索，$W(n)=W(n/2)+1$

归并排序，$W(n)=2W(n/2)+n-1$

递推方程的求解

方程1 $$f(n)=\sum_{i=1}^{k}a_i f(n-i)+g(n)$$

方程2 $$f(n)=af(\frac{n}{b})+d(n)$$

求解方法

方程1：迭代法、递归树

方程2：迭代法、换元法、递归树、

主定理

方程2的解

方程 $$T(n) = aT(n/b) + d(n)$$

d(n)为常数

$$T(n) = \begin{cases} O(n^{\log_b a}) & a \neq 1 \\ O(\log n) & a = 1 \end{cases}$$

$d(n) = cn$

$$T(n) = \begin{cases} O(n) & a < b \\ O(n\log n) & a = b \\ O(n^{\log_b a}) & a > b \end{cases}$$

小结

- 分治算法的一般描述

  划分或归约为彼此独立的子问题
  分别求解每个子问题
  给出递归或迭代计算的终止条件
  如何由子问题的解得到原问题解

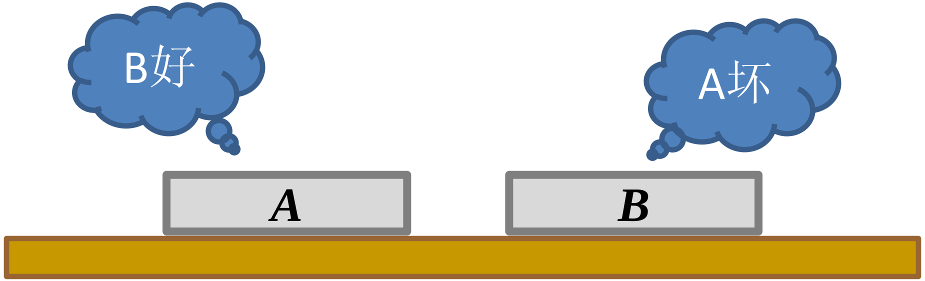
- 分治算法的分析方法

  求解时间复杂度的递推方程
  常用的递推方程的解

芯片测试

一次测试过程

测试方法：将2片芯片（*A*和*B*）置于测试台上，互相进行测试，测试报告是“好”或“坏”，只取其一.



假设：好芯片的报告一定是正确的，坏芯片的报告是不确定的（可能会出错）

测试结果分析

A 报告	B 报告	结论
B是好的	A是好的	A, B 都好或 A, B 都坏
B是好的	A是坏的	至少一片是坏的
B是坏的	A是好的	至少一片是坏的
B是坏的	A是坏的	至少一片是坏的

问题

输入：

n **片芯片，其中好芯片至少比坏芯片多 1片.**

问题：

设计一种测试方法，通过测试从 ***n*** **片芯片中挑出 1 片好芯片.**

要求：使用最少的测试次数.

判定芯片A 的好坏

问题：给定芯片A，判定A的好坏

方法：用其他 n-1片芯片对 A 测试 .

n=7：好芯片数 ≥ 4.

A 好，6个报告中至少 3 个报“好”

A 坏，6个报告中至少 4 个报“坏”

n是奇数：好芯片数 ≥ (n+1)/2.

A 好，至少有 (n–1)/2个报“好”

A 坏，至少有 (n+1)/2个报告“坏”

结论：至少一半报“好”，A是好芯片，

超过一半报“坏”，A是坏芯片.

判定芯片*A* 的好坏

n=8：好芯片数 ≥ 5.

A 好，7个报告中至少 4 个报 "好"

A 坏，7个报告中至少 5 个报 "坏"

n是偶数：好芯片数 ≥ $n/2+1$.

A 好，至少有 $n/2$个报告 "好"

A 坏，至少有 $n/2+1$个报告 "坏"

结论：n-1 份报告中，

至少一半报 "好"，则 *A* 为好芯片

超过一半报 "坏"，则 *A*为坏芯片

蛮力算法

测试方法：任取 1片测试，如果是好芯片，测试结束；如果是坏芯片，抛弃，再从剩下芯片中任取 1片测试，直到得到 1片好芯片.

时间估计：

第1片坏芯片，最多测试 n-2次，

第2片坏芯片，最多测试 n-3次，

…

总计 $\Theta(n^2)$

分治算法设计思想

假设 n 为偶数，将 n 片芯片两两一组做测试淘汰，剩下芯片构成子问题，进入下一轮分组淘汰.

淘汰规则：

"好, 好" $\Rightarrow$ 任留 1片，进入下轮

其他情况 $\Rightarrow$ 全部抛弃

递归截止条件：$n \leq 3$

3 片芯片，1 次测试可得到好芯片.

1 或 2 片芯片，不再需要测试.

分治算法的正确性

命题1 当 n 是偶数时，在上述淘汰规则下，经过一轮淘汰，剩下的好芯片比坏芯片至少多1片.

证 设A, B都好的芯片 i 组, A与B一好一坏 j 组，A与B 都坏的 k 组. 淘汰后好芯片至少 i片，坏芯片至多 k片 .

$2i + 2j + 2k = n$ 初始芯片总数

$2i + j > 2k + j$ 初始 好芯片多于坏芯片

$\Rightarrow$ $i > k$

*n*为奇数时的特殊处理

当 *n* 是奇数时，可能出问题

输入：好 好 好 好 坏 坏 坏

分组：好 好 好 好 坏 坏 坏

淘汰后：好 好 坏 坏

处理办法：当 *n* 为奇数时，增加一轮对轮空芯片的单独测试.

如果该芯片为好芯片，算法结束；
如果是坏芯片，则淘汰该芯片.

伪码描述

```
算法 Test(n)
k ← n
while k >3 do
    将芯片分成⌊k/2⌋组  // 轮空处理
    for i =1 to ⌊k/2⌋ do
        if 2片好 then 则任取1片留下
        else 2片同时丢掉
    k←剩下的芯片数
if k =3 then
    任取2片芯片测试
    if 1好1坏 then 取没测的芯片
    else 任取1片被测芯片
if k =2 or 1 then 任取1片
```

分组淘汰

递归结束

时间复杂度分析

设输入规模为n

每轮淘汰后，芯片数至少减半

测试次数(含轮空处理)：$O(n)$

时间复杂度：

$$W(n) = W(n/2) + O(n)$$

$$W(3) = 1,\ W(2) = W(1) = 0$$

解得 $W(n) = O(n)$

小结

- 芯片测试的分治算法

  如何保证子问题与原问题性质相同:
  增加额外处理

  额外处理的工作量不改变函数的阶

  时间复杂度为$O(n)$

快速排序

基本思想

- 用首元素 x 作划分标准，将输入数组 A划分成不超过 x 的元素构成的数组 A_L，大于 x 的元素构成的数组 A_R. 其中 A_L, A_R从左到右存放在数组 A 的位置.
- 递归地对子问题 A_L和 A_R 进行排序，直到子问题规模为 1 时停止.

伪码

算法 **Quicksort** (A, p, r)

输入：数组$A[p..r]$

输出：排好序的数组 A

1. **if** $p < r$
2. **then** $q \leftarrow$ **Partition** (A, p, r)
3. $A[p] \leftrightarrow A[q]$
4. **Quicksort** (A, p, q-1)
5. **Quicksort** (A, q+1$, r$)

初始置 p=1, r=n，然后调用上述算法

划分过程

```
Partition (A, p, r)
x ← A[p]
i ← p
j ← r + 1
while true do
      repeat  j ← j –1
      until  A[ j ]≤ x  // 不超过首元素的
      repeat  i ← i +1
      until  A[ i ]>x  // 比首元素大的
      if  i < j
      then  A[ i ] ↔ A[ j ]
      else  return j
```

划分实例

27 99 0 8 13 64 86 16 7 10 88 25 90

i *j*

27 25 0 8 13 64 86 16 7 10 88 99 90

i *j*

27 25 0 8 13 10 86 16 7 64 88 99 90

i *j*

27 25 0 8 13 10 7 16 86 64 88 99 90

j *i*

16 25 0 8 13 10 7 27 86 64 88 99 90

时间复杂度

最坏情况: $W(n) = W(n-1)+n-1$

$$W(1) = 0$$

$$W(n) = n(n-1)/2$$

最好划分: $T(n) = 2\,T(n/2)+n-1$

$$T(1) = 0$$

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

均衡划分的时间复杂度

均衡划分：子问题的规模比不变

例如为 1:9

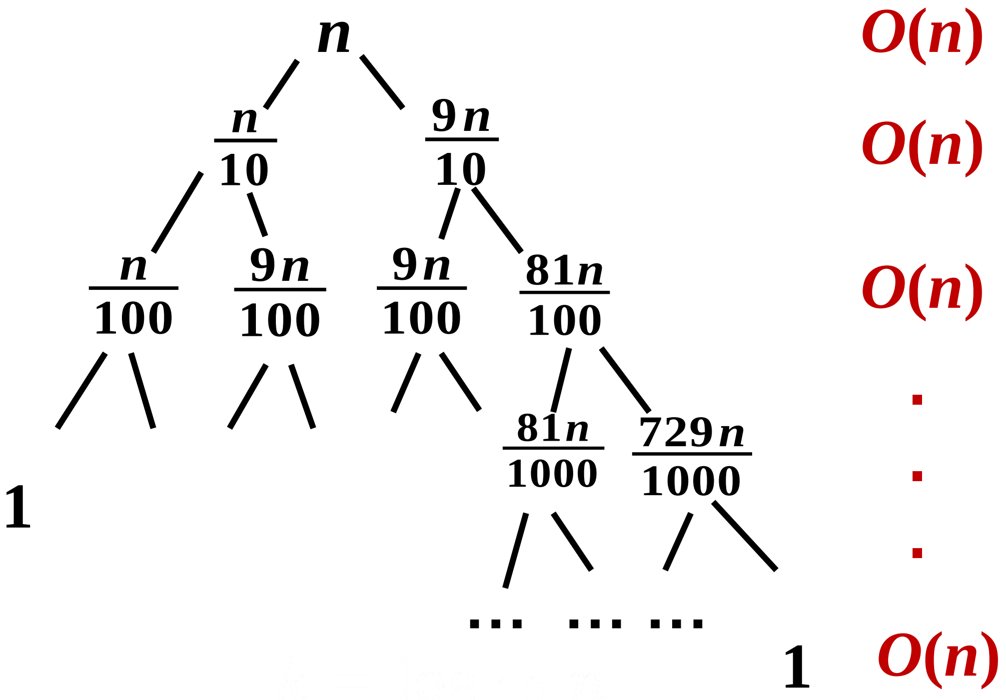
$$T(n) = T(n/10) + T(9n/10) + n$$

$$T(1) = 0$$

根据递归树，时间复杂度

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

递归树



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

平均时间复杂度

首元素排好序后处在 $1, 2, \ldots, n$

各种情况概率都为 $1/n$

首元素在位置 1：　　$T(0), T(n-1)$

首元素在位置 2：　　$T(1), T(n-2)$

....

首元素在位置 n-1：　$T(n-2), T(1)$

首元素在位置 n：　　$T(n-1), T(0)$

子问题工作量 $2[T(1)+T(2)+\ldots+T(n-1)]$

划分工作量　$n-1$

平均时间复杂度

$$T(n)=\frac{1}{n}\sum_{k=1}^{n-1}(T(k)+T(n-k))+n-1$$

$$\boxed{\begin{aligned}&T(n)=\frac{2}{n}\sum_{k=1}^{n-1}T(k)+n-1\\&T(1)=0\end{aligned}}$$

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

首元素划分后每个位置概率相等

小结

快速排序算法

- 分治策略
- 子问题划分是由首元素决定
- 最坏情况下时间$O(n^2)$
- 平均情况下时间为$O(n\log n)$

幂乘算法及应用

幂乘问题

输入： a为给定实数，n为自然数

输出： a^n

传统算法： 顺序相乘

$$a^n = (\ldots(((a\ a)a)a)\ldots)a$$

乘法次数： $\Theta(n)$

分治算法——划分

n为偶数 $\underbrace{a\,.\,.\,.\,.\,.\,.\,.\,a}_{n/2个} \mid \underbrace{a\,.\,.\,.\,.\,.\,.\,.\,a}_{n/2个}$

n为奇数 $\underbrace{a\,.\,.\,.\,.\,.\,.\,.\,a}_{(n-1)/2个} \mid \underbrace{a\,.\,.\,.\,.\,.\,.\,.\,a}_{(n-1)/2个} \mid a$

$$a^n = \begin{cases} a^{n/2} \times a^{n/2} & n \text{ 为偶数} \\ a^{(n-1)/2} \times a^{(n-1)/2} \times a & n \text{ 为奇数} \end{cases}$$

分治算法分析

以乘法作为基本运算

- 子问题规模：不超过$n/2$
- 两个规模近似 $n/2$的子问题完全一样，只要计算1次

$$W(n) = W(n/2) + \Theta(1)$$

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

幂乘算法的应用

Fibonacci数列： 1, 1, 2, 3, 5, 8, 13, 21, …

增加 F_0=0，得到数列

0, 1, 1, 2, 3, 5, 8, 13, 21, …

问题: 已知 F_0=0, F_1=1, 给定n, 计算 F_n.

通常算法：从 F_0, F_1, … 开始，根据递推公式

$$\boxed{F_n = F_{n-1} + F_{n-2}}$$

陆续相加可得 F_n， 时间复杂度为$\Theta(n)$

Fibonacci数的性质

定理1 设 $\{F_n\}$为 Fibonacci 数构成的数列，那么

$$\boxed{\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n}$$

归纳证明

n=1，左边= $\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ =右边

Fibonacci数的性质(续)

假设对任意正整数 n，命题成立，即

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

那么 $$\begin{bmatrix} \boxed{F_{n+2}} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix} = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n+1}$$

归纳假设代入

算法

令矩阵 $M=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$, 用乘幂算法计算 M^n

时间复杂度：

- 矩阵乘法次数 $T(n)=\Theta(\log n)$
- 每次矩阵乘法需要做 8 次元素相乘
- 总计元素相乘次数为 $\Theta(\log n)$

小结

- 分治算法的例子——幂乘算法
- 幂乘算法的应用

  计算Fibonacci数

  通常算法$O(n)$，分治算法为$O(\log n)$

改进分治算法的途径1：减少子问题数

减少子问题个数的依据

分治算法的时间复杂度方程

$$W(n) = aW(n/b) + d(n)$$

a：子问题数，n/b：子问题规模，

$d(n)$：划分与综合工作量.

当 a 较大, b较小, $d(n)$不大时,方程的解：

$$\underline{W(n) = \Theta(n^{\log_b a})}$$

减少a是降低函数$W(n)$的阶的途径.

利用子问题的依赖关系，使某些子问题的解通过组合其他子问题的解而得到.

例1：整数位乘问题

输入：X, Y 是 n 位二进制数， $n = 2^k$

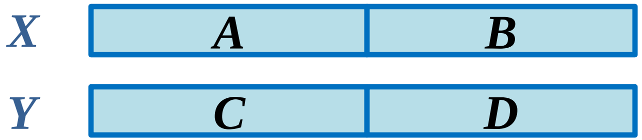
输出：XY

普通乘法：需要$O(n^2)$次位乘运算

简单划分：令

$$X = A2^{n/2} + B, \quad Y = C2^{n/2} + D.$$

$$XY = \underline{AC}\, 2^n + (\underline{AD} + \underline{BC})\, 2^{n/2} + \underline{BD}$$



$$W(n) = 4W(n/2) + O(n) \Rightarrow W(n) = O(n^2)$$

减少子问题个数

子问题间的依赖关系：代数变换

$$AD+BC = \underline{(A-B)(D-C)} + \underline{AC} + \underline{BD}$$

算法复杂度

$$W(n) = 3\,W(n/2) + cn$$

$$W(1) = 1$$

方程的解

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

例2：矩阵相乘问题

输入：A, B 为 n 阶矩阵，$n = 2^k$

输出：$C = AB$

通常矩阵乘法：

C 中有 n^2 个元素

每个元素需要做 n 次乘法

以元素相乘为基本运算

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

简单分治算法

分治法　将矩阵分块，得

$$\begin{pmatrix} \boxed{A_{11} \quad A_{12}} \\ A_{21} \quad A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} \boxed{C_{11}} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

其中

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

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

递推方程　$W(n) = 8\ W(n/2) + cn^2$

$W(1) = 1$

解　$W(n) = O(n^3)$.

Strassen 矩阵乘法

变换方法：

设计 $M_1, M_2, \ldots, M_7$，对应7个子问题

$$M_1 = A_{11}(B_{12} - B_{22})$$

$$M_2 = (A_{11} + A_{12})B_{22}$$

$$M_3 = (A_{21} + A_{22})B_{11}$$

$$M_4 = A_{22}(B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$M_6 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$M_7 = (A_{11} - A_{21})(B_{11} + B_{12})$$

Strassen 矩阵乘法(续)

利用中间矩阵，得到结果矩阵

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

时间复杂度函数:

$$W(n) = 7\,W(n/2) + 18(n/2)^2$$
$$W(1) = 1$$

解 $W(n) = O(n^{\log 7}) = O(n^{2.8075})$

矩阵乘法的研究及应用

矩阵乘法问题的难度：

- **Coppersmith–Winograd算法：$O(n^{2.376})$**
  目前为止最好的上界
- 目前为止最好的下界是： $\Omega(n^2)$

应用：

- 科学计算、图像处理、数据挖掘等
- 回归、聚类、主成分分析、决策树等挖掘算法常涉及大规模矩阵运算

改进途径小结

- 适用于：子问题个数多，划分和综合工作量不太大，时间复杂度函数
  $$W(n) = \Theta(n^{\log_b a})$$
- 利用子问题依赖关系，用某些子问题解的代数表达式表示另一些子问题的解，减少独立计算子问题个数.
- 综合解的工作量可能会增加，但增加的工作量不影响 $W(n)$的阶.

改进分治算法的途径2：增加预处理

例子：平面点对问题

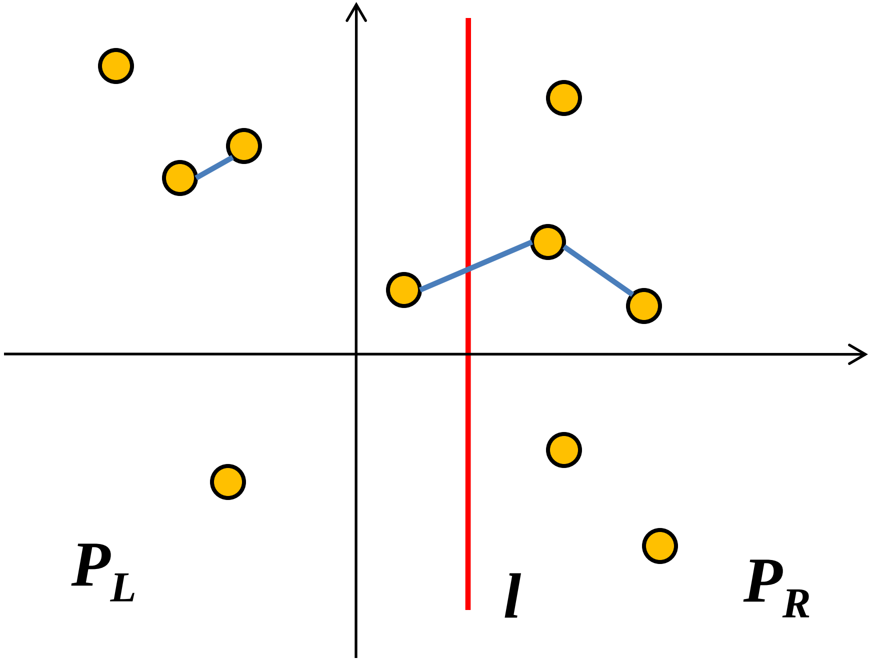
输入: 平面点集 P 中有n 个点, $n>1$
输出: P中的两个点，其距离最小

蛮力算法:
$C(n,2)$个点对,计算最小距离，$O(n^2)$

分治策略: P 划为大小相等的P_L和P_R

1. 分别计算 P_L、P_R中最近点对
2. 计算 P_L与P_R中各一个点的最近点对
3. 上述情况下的最近点对是解

划分实例：n=10



算法伪码

MinDistance (P, X, Y)

输入: 点集P, X和Y为横、纵坐标数组

输出: 最近的两个点及距离

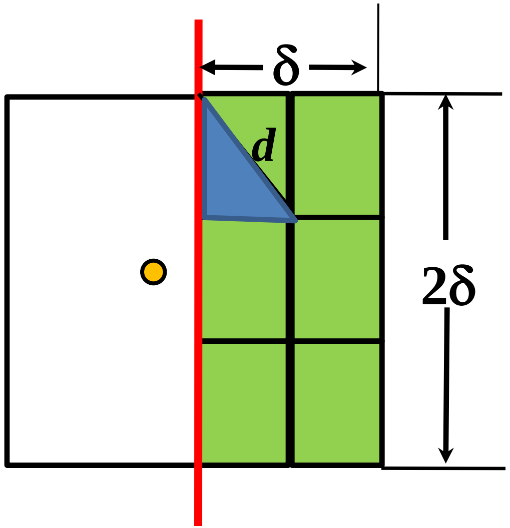
1. 若$|P|\le 3$，直接计算其最小距离
2. 排序X,Y
3. 做中垂线 l 将P划分为P_L和P_R
4. **MinDidtance** (P_L,X_L,Y_L)
5. **MinDistance** (P_R,X_R,Y_R)
6. $\delta=\min(\delta_L,\delta_R)$//$\delta_L,\delta_R$为子问题的距离
7. 检查距 l 不超过δ两侧各1个点的距离．若小于δ，修改 δ为这个值

跨边界处理

$$d=\sqrt{(\delta/2)^2+(2\delta/3)^2}$$

$$=\sqrt{\delta^2/4+4\delta^2/9}$$

$$=\sqrt{25\delta^2/36}=5\delta/6$$



右边每个小方格至多1个点，每个点至多比较对面的6个点，检查1个点是常数时间，$O(n)$ 个点需要$O(n)$时间

算法分析

步1 递归边界处理：　$O(1)$

步2 排序：　$O(n\log n)$

步3 划分：　$O(1)$

步4-5子问题：　$2T(n/2)$

步6确定δ:　$O(1)$

步7检查跨边界点对：　$O(n)$

$$T(n)=2T(n/2)+O(n\log n)$$
$$T(n)=O(1),\ n\le 3$$

递归树求解　$T(n)=O(n\log^2 n)$

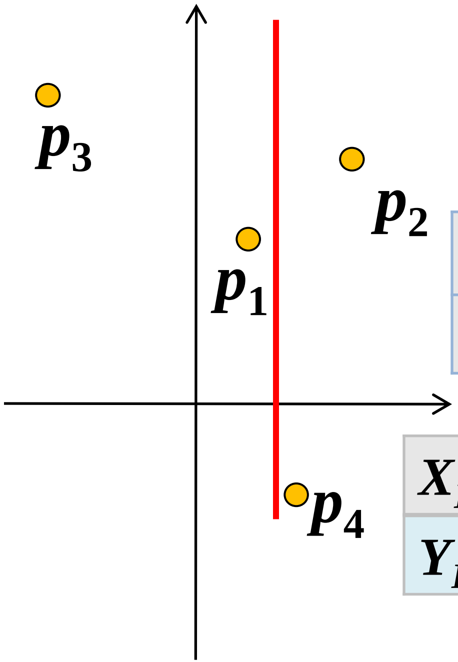
增加预处理

原算法：
在每次划分时对子问题数组重新排序

改进算法：
1. 在递归前对 X,Y 排序，作为预处理
2. 划分时对排序的数组 X,Y 进行拆分，
   得到针对子问题 P_L 的数组 X_L,Y_L 及
   针对子问题 P_R 的数组 X_R,Y_R

原问题规模为 n，拆分的时间为 $O(n)$

实例：递归中的拆分



P	1	2	3	4
X	0.5	2	-2	1
Y	2	3	4	-1

输入

X	-2(3)	0.5(1)	1(4)	2(2)
Y	-1(4)	2(1)	3(2)	4(3)

预处理

X_L	2(3)	0.5(1)
Y_L	2(1)	4(3)

X_R	1(4)	2(2)
Y_R	-1(4)	3(2)

拆分后

改进算法时间复杂度

$W(n)$为算法时间复杂度

递归过程:$T(n)$，预处理:$O(nlogn)$

$$W(n) = T(n) + O(nlogn)$$
$$T(n) = 2T(n/2) + O(n)$$
$$T(n) = O(1) \qquad n \le 3$$

解得 $T(n) = O(nlogn)$

于是 $W(n) = O(nlogn)$

小结

- 依据

$$W(n) = aW(n/b) + f(n)$$

- 提高算法效率的方法：
  - 减少子问题个数a：

$$W(n)=O(n^{\log_b a})$$

  - 增加预处理，减少$f(n)$

本周教学内容

典型的分治算法

选择问题	信号平滑处理	计算几何
选第*k*小	快速傅立叶变换FFT算法	计算平面点集的凸包
选第二大	卷积计算	
选最大与最小	卷积及应用	
选最大		

选最大与最小

选择问题

输入：集合 L (含n个不等的实数)
输出：L中第 i 小元素

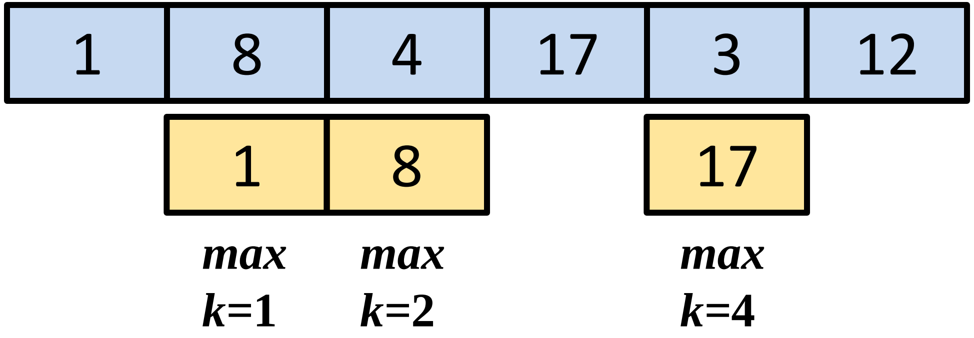
i=1, 称为最小元素

i=n，称为最大元素

位置处在中间的元素，称为中位元素
n为奇数，中位数唯一，$i = (n+1)/2$
n为偶数，可指定 $i = n/2+1$

选最大

算法：顺序比较



输出：$max = 17$，$k=4$

算法最坏情况下的时间$W(n)=n-1$

伪码

算法 Findmax

输入：*n* 个数的数组 *L*

输出：*max*, *k*

1. $max \leftarrow L[1]$
2. **for** $i \leftarrow 2$ **to** n **do**
3. 　　**if** $max < L[i]$
4. 　　**then** $max \leftarrow L[i]$
5. 　　　　$k \leftarrow i$
6. **return** max, k

选最大最小

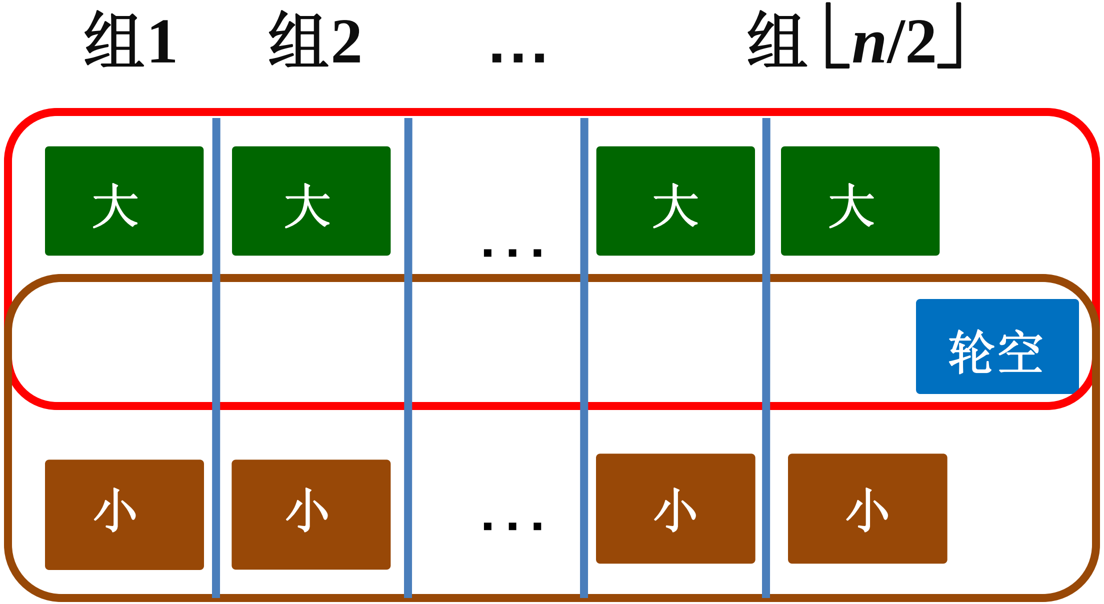
通常算法：

1. 顺序比较，先选最大 ***max***
2. 顺序比较，在剩余数组中选最小 ***min***，类似于选最大算法，但比较时保留较小的数

时间复杂性：

$$W(n) = n\text{-}1 + n\text{-}2 = 2n-3$$

分组算法



伪码

算法 FindMaxMin

输入：*n*个数的数组*L*

输出：*max*，*min*

1. 将 n 个元素两两一组分成$\lfloor n/2 \rfloor$组
2. 每组比较，得到$\lfloor n/2 \rfloor$个较小和$\lfloor n/2 \rfloor$个较大
3. 在$\lceil n/2 \rceil$个较大（含轮空元素）中找最大 max
4. 在$\lceil n/2 \rceil$个较小（含轮空元素）中找最小 min

最坏情况时间复杂度

行2 的组内比较： $\lfloor n/2 \rfloor$ 次

行3--4 求 *max* 和 *min* 比较：

至多 $2\lceil n/2 \rceil - 2$次

$$W(n) = \lfloor n/2 \rfloor + 2\lceil n/2 \rceil - 2$$
$$= n + \lceil n/2 \rceil - 2$$
$$= \lceil 3n/2 \rceil - 2$$

分治算法

1. 将数组 L 从中间划分为两个子数组 L_1 和 L_2
2. 递归地在 L_1 中求最大 max_1 和 min_1
3. 递归地在 L_2 中求最大 max_2 和 min_2
4. $max \leftarrow \mathrm{max}\{max_1, max_2\}$
5. $min \leftarrow \mathrm{min}\{min_1, min_2\}$

最坏情况时间复杂度

假设 $n = 2^k$,

$$W(n) = 2W(n/2) + 2$$

$$W(2) = 1$$

解 $W(2^k) = 2W(2^{k-1}) + 2$

$= 2[2W(2^{k-2}) + 2] + 2$

$= 2^2W(2^{k-2}) + 2^2+2 = \ \dots$

$= 2^{k-1} + 2^{k-1} + \dots + 2^2 + 2$

$= 3 \cdot 2^{k-1} - 2 = \boxed{3n/2 - 2}$

选择算法小结

选最大：顺序比较, 比较次数 $\boxed{n\text{-}1}$

选最大最小

- 选最大+ 选最小, 比较次数 2n-3
- 分组： 比较次数 $\boxed{\lceil 3n/2 \rceil - 2}$
- 分治：$n=2^k$，比较次数 $\boxed{3n/2\text{-}2}$

选第二大

选第二大

输入：n个数的数组 L

输出：第二大的数 *second*

通常算法：顺序比较

1．顺序比较找到最大 *max*

2．从剩下 $n-1$个数中找最大，就是第二大*second*

时间复杂度：

$$W(n) = n-1 + n-2 = 2n-3$$

提高效率的途径

- 成为第二大数的条件：仅在与最大数的比较中被淘汰.
- 要确定第二大数，必须知道最大数.
- 在确定最大数的过程中记录下被最大数直接淘汰的元素.
- 在上述范围（被最大数直接淘汰的数）内的最大数就是第二大数.
- 设计思想： 用空间换时间.

锦标赛算法

1. 两两分组比较，大者进入下一轮，直到剩下 1个元素 *max* 为止

2. 在每次比较中淘汰较小元素，将被淘汰元素记录在淘汰它的元素的链表上

3. 检查 *max* 的链表，从中找到最大元，即*second*

伪码

算法 FindSecond

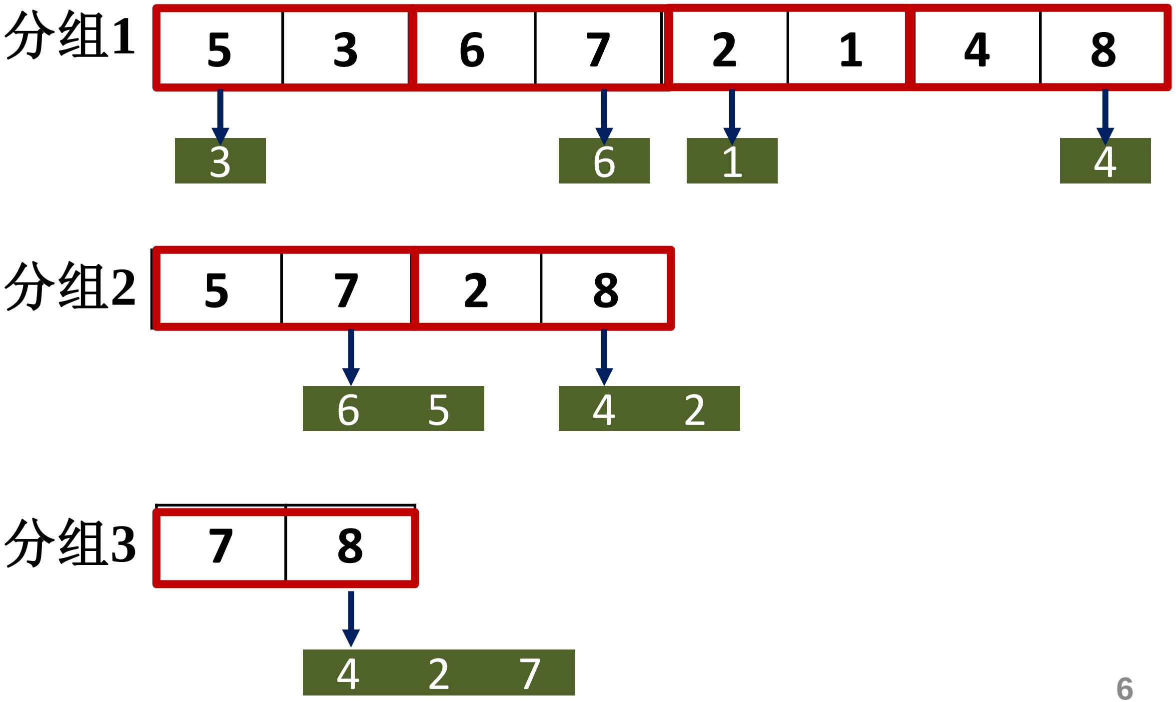
输入: *n*个数的数组 *L*, 输出: *second*

1. $k \leftarrow n$ // 参与淘汰的元素数
2. 将k个元素两两1组, 分成$\lfloor k/2 \rfloor$组
3. 每组的2个数比较，找到较大数
4. 将被淘汰数记入较大数的链表
5. if k 为奇数 then $k \leftarrow \lfloor k/2 \rfloor + 1$
6. else $k \leftarrow \lfloor k/2 \rfloor$
7. if $k>1$ then goto 2
8. $max \leftarrow$最大数
9. $second \leftarrow max$ 的链表中的最大

一轮淘汰

继续分组淘汰

实例



时间复杂度分析

命题1 设参与比较的有 t 个元素，经过 i 轮淘汰后元素数至多为 $\lceil t/2^i \rceil$.

证 对 i 归纳. $i=1$,分 $\lfloor t/2 \rfloor$ 组，淘汰 $\lfloor t/2 \rfloor$ 个元素，进入下一轮元素数是

$$t - \lfloor t/2 \rfloor = \lceil t/2 \rceil$$

假设 i 轮分组淘汰后元素数至多为 $\lceil t/2^i \rceil$，那么 $i+1$ 轮分组淘汰后元素数为

$$\lceil \lceil t/2^i \rceil /2 \rceil = \lceil t/2^{i+1} \rceil$$

时间复杂度分析（续）

命题2 max 在第一阶段分组比较中总计进行了$\lceil \log n \rceil$次比较.

证　假设到产生 max 时总计进行 k 轮淘汰, 根据命题 1有$\lceil n/2^k \rceil=1$.

若 $n=2^d$，那么有

$$k = d = \log n = \lceil \log n \rceil$$

若 $2^d < n < 2^{d+1}$，那么

$$k = d + 1 = \lceil \log n \rceil$$

时间复杂度分析（续）

第一阶段元素数：n

比较次数：$\underline{n-1}$

淘汰了 $n-1$个元素

第二阶段：元素数$\lceil \log n \rceil$

比较次数：$\underline{\lceil \log n \rceil -1}$

淘汰元素数为$\lceil \log n \rceil -1$

时间复杂度是

$$W(n) = n - 1 + \lceil \log n \rceil - 1$$
$$= n + \lceil \log n \rceil - 2.$$

小结

求第二大算法

- 调用2次找最大：$2n-3$
- 锦标赛算法：$n+\lceil \log n \rceil-2$

主要的技术：用空间换时间

一般选择问题的算法设计

一般性选择问题

问题：选第 k 小.

输入：数组 S, S 的长度 n, 正整数 k,

$1\le k\le n$.

输出：第 k 小的数

实例 1

$S=\{3,4,8,2,5,9,18\}$，$k=4$，解：5

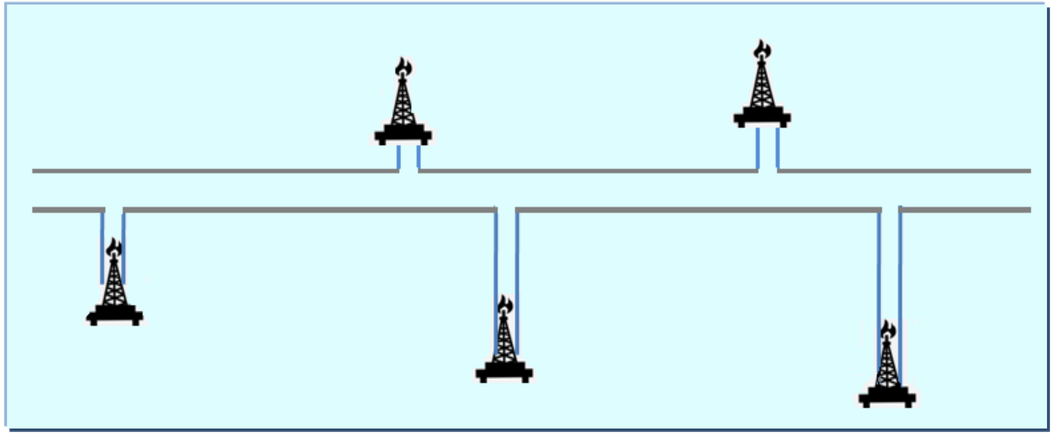
实例 2

统计数据的集合S，$|S|=n$，

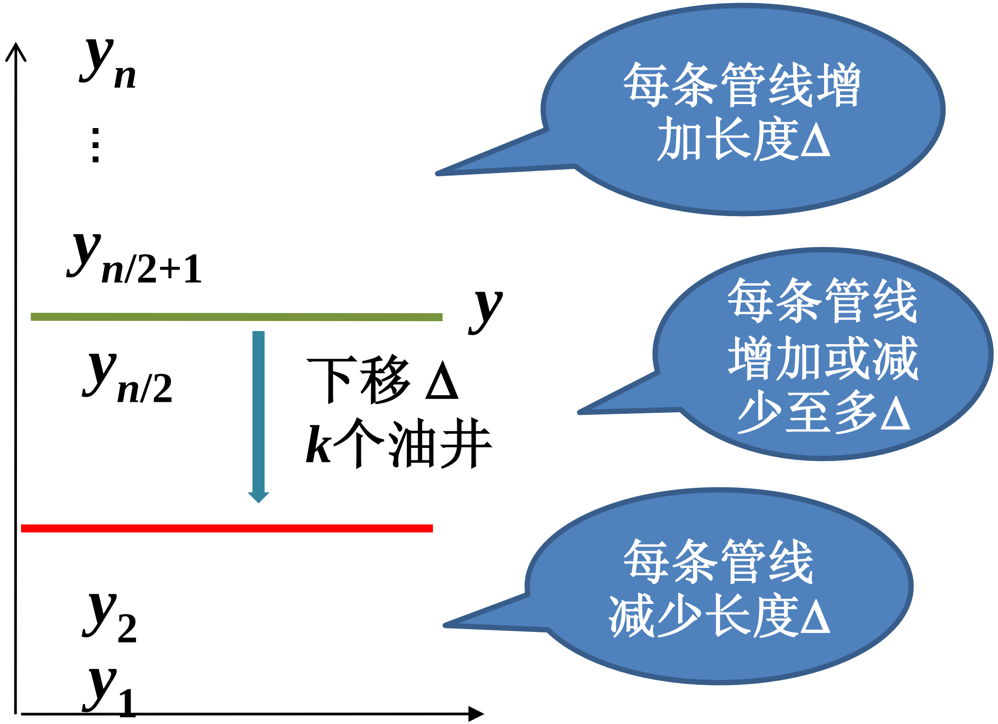
选中位数，$k=\lceil n/2\rceil$

一个应用：管道位置

问题：某区域有n口油井，需要修建输油管道. 根据设计要求，水平方向有一条主管道，每口油井修一条垂直方向的支管道通向主管道. 如何选择主管道的位置，以使得支管道长度的总和最小？



最优解：Y 坐标的中位数



下移后支管线总长度增加

简单的算法

算法一：

调用 k 次选最小算法

时间复杂度为 $O(kn)$

算法二：

先排序，然后输出第 k 小的数

时间复杂度为 $O(n\log n)$

分治算法

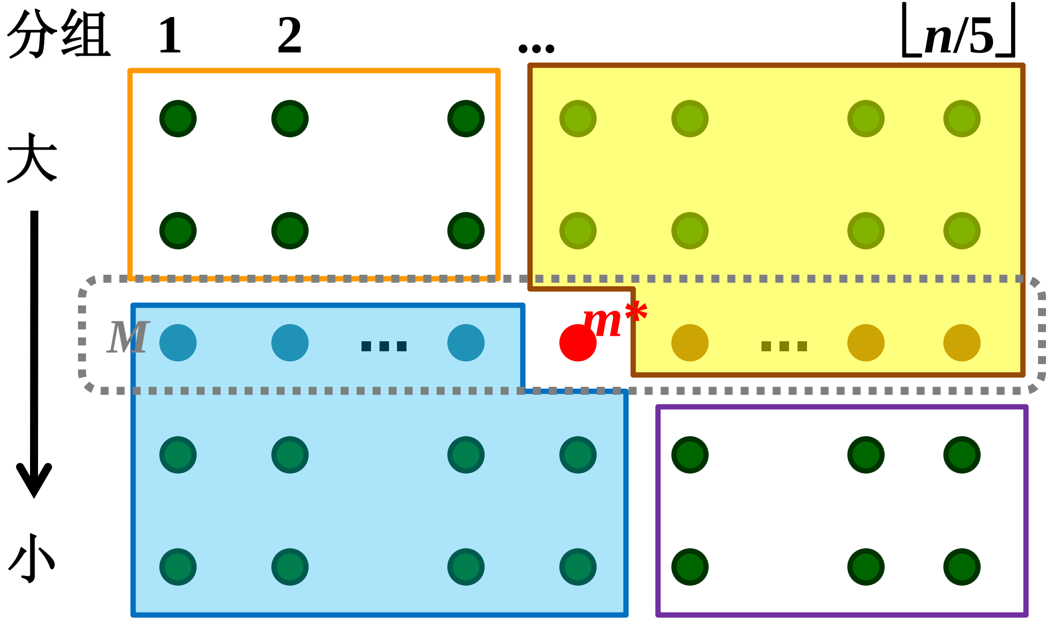
假设元素彼此不等，设计思想：

1. 用某个元素 m^*作为标准将 S 划分成 S_1 与 S_2，其中 S_1的元素小于 m^*，S_2 的元素大于等于 m^*.

2. 如果 $k \le |S_1|$，则在 S_1中找第 k 小.
   如果 $k = |S_1|+1$，则 m^*是第 k 小
   如果 $k > |S_1|+1$，则在 S_2中找第 $k-|S_1|-1$小

算法效率取决于子问题规模，
如何通过m^*控制子问题规模？

*m**的选择与划分过程



A: 数需要与*m**比大小，*B*：数大于*m**

C: 数小于*m**，*D*:数需要与*m**比大小

实例：n=15, k=6

8	2	3	5	7	6	11	14	1	9	13	10	4	12	15

	8	14	15
	7	11	13
M	5	9	12
	3	6	10
	2	1	4

$m^* = 9$

	8	14	15	
A	7	11	13	B
	5	9	12	
C	3	6	10	D
	2	1	4	

8, 7, 10, 4 需要与9比较

归约为子问题

	8	14	15	
	7	11	13	
S_1	5	9	12	S_2
	3	6	10	
	2	1	4	

子问题

8 7 5 3 2 6 1 4

子问题规模 = 8，k=6

伪码

算法 Select (S, k)

输入：数组 S，正整数 k，

输出：S 中的第 k 小元素

1. 将S分5个一组, 共 $n_M = \lceil n/5 \rceil$组
2. 每组排序,中位数放到集合 M
3. $m^* \leftarrow \text{Select}(M, \lceil |M|/2 \rceil)$ //S分A,B,C,D
4. A,D元素小于m^*放 S_1,大于m^*放S_2
5. $S_1 \leftarrow S_1 \cup C$; $S_2 \leftarrow S_2 \cup B$ 划分
6. if $k = |S_1| + 1$ then 输出 m^*
7. else if $k \le |S_1|$ $\Leftarrow$ 递归计算子问题
8.   then Select (S_1, k)
9.   else Select (S_2, $k - |S_1| - 1$)

小结

选第 *k* 小的算法：

- 分治策略
- 确定*m**
- 用*m**划分数组归约为子问题
- 递归实现

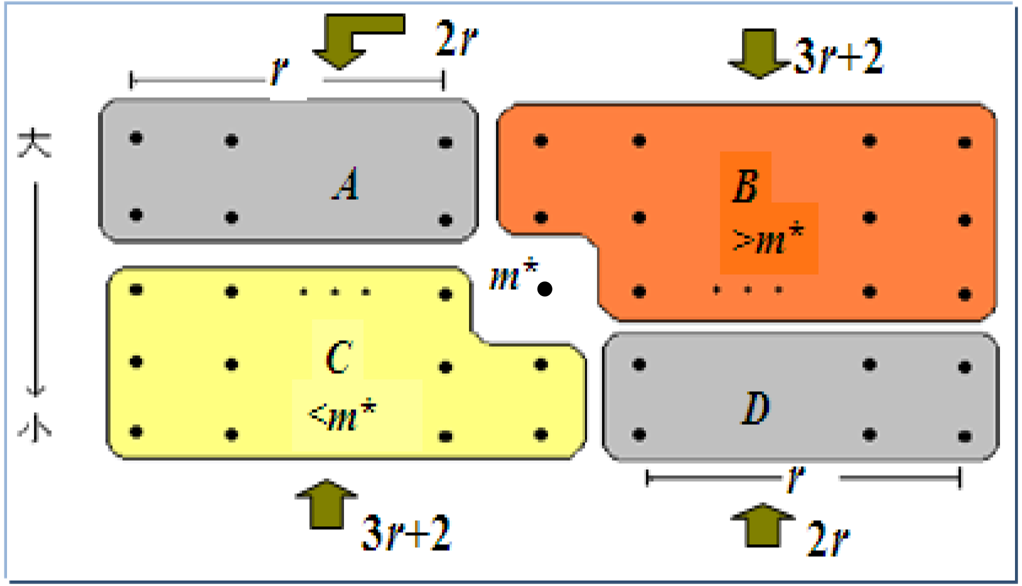
选择问题的算法分析

伪码

算法 Select (S, k)

1. 将S分5个一组, 共 $n_M = \lceil n/5 \rceil$组
2. 每组排序,中位数放到集合 M
3. $m^* \leftarrow$Select(M,$\lceil |M|/2 \rceil$) //S分A,B,C,D
4. A,D元素小于m^*放 S_1,大于m^*放S_2
5. $S_1 \leftarrow S_1 \cup C$; $S_2 \leftarrow S_2 \cup B$
6. if $k = |S_1| + 1$ then 输出 m^*
7. else if $k \leq |S_1|$
8.   then Select (S_1, k)
9.   else Select (S_2, $k - |S_1| - 1$)

用m*划分



$n = 5\ (2r+1)$, $|A| = |D| = 2r$

子问题规模至多： $2r+2r+3r+2 = 7r+2$

子问题规模估计

不妨设 $n = 5(2r+1)$， $|A|=|D|=2r$,

$$r = \frac{n/5-1}{2} = \frac{n}{10} - \frac{1}{2}$$

划分后子问题规模至多为

$$7r+2 = 7\left(\frac{n}{10} - \frac{1}{2}\right) + 2$$

$$= \frac{7n}{10} - \frac{3}{2} < \frac{7n}{10}$$

时间复杂度递推方程

算法工作量 $W(n)$

行2: $O(n)$ //每5个数找中位数,构成M

行3: $\underline{W(n/5)}$ // M 中找中位数 m^*

行4: $O(n)$ // 用m^*划分集合 S

行8-9: $\underline{W(7n/10)}$ //递归

$$W(n) \le W(n/5) + W(7n/10) + O(n)$$

递归树

$W(n)=W(n/5)+W(7n/10)+cn$

$$cn \qquad\qquad cn$$

$$\frac{cn}{5} \quad \frac{7cn}{10} \qquad\qquad 0.9cn$$

$$\frac{cn}{25} \quad \frac{7cn}{50} \quad \frac{7cn}{50} \quad \frac{49cn}{100} \qquad\qquad 0.81cn$$

……

$$W(n) \le cn\,(1+0.9+0.9^2+\ldots)=O(n)$$

讨论

分组时为什么5个元素一组？
3个一组或 7个一组行不行？

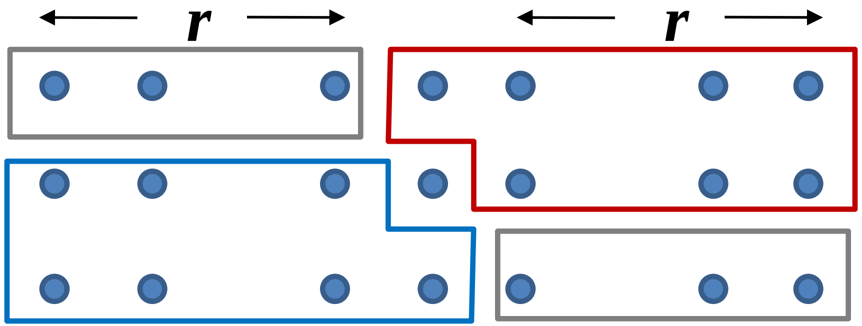
分析：递归调用

1. 求 m^*的工作量与 $|M| = n/t$ 相关，t 为每组元素数. t大, $|M|$小

2. 归约后子问题大小与分组元素数t 有关 . t 大，子问题规模大

3分组时的子问题规模

假设 t =3，3个一组：



$$n = 3(2r + 1)$$

$$r = (n/3 - 1)/2 = n/6 - 1/2$$

子问题规模最多为 $4r+1 = 4n/6-1$

算法的时间复杂度

算法的时间复杂度满足方程

$$W(n) = W(n/3) + W(4n/6) + cn$$

由递归树得 $W(n)=\Theta(n\log n)$

关键:

$|M|$与归约后子问题规模之和小于 n,

递归树每行的工作量构成公比小于 1 的等比级数, 算法复杂度才是$O(n)$.

小结

选第 k 小算法的时间分析

- 递推方程
- 分组时每组元素数的多少对时间复杂度的影响

卷积及其应用

向量计算

给定向量 $a = (a_0, a_1, \ldots, a_{n-1})$

$b = (b_0, b_1, \ldots, b_{n-1})$

向量和 $a+b = (a_0+b_0, a_1+b_1, \ldots, a_{n-1}+b_{n-1})$

内积 $a \cdot b = a_0b_0 + a_1b_1 + \ldots + a_{n-1}b_{n-1}$

卷积 $a*b = (c_0, c_1, \ldots, c_{2n-2})$，其中

$$c_k = \sum_{\substack{i+j=k \\ i,j<n}} a_i b_j, \quad k = 0,1,\ldots,2n-2$$

卷积的含义

在下述矩阵中，每个斜线的项之和恰好是卷积中的各个分量

$$
\begin{array}{c|ccccc}
 & ab_0 & ab_1 & \cdots & ab_{n-2} & ab_{n-1} \\
\hline
a_0 b & a_0b_0 & a_0b_1 & \cdots & a_0b_{n-2} & a_0b_{n-1} \\
a_1 b & a_1b_0 & a_1b_1 & \cdots & a_1b_{n-2} & a_1b_{n-1} \\
\vdots & \vdots & \vdots & & \vdots & \vdots \\
a_{n-2} b & a_{n-2}b_0 & a_{n-2}b_1 & \cdots & a_{n-2}b_{n-2} & a_{n-2}b_{n-1} \\
a_{n-1} b & a_{n-1}b_0 & a_{n-1}b_1 & \cdots & a_{n-1}b_{n-2} & a_{n-1}b_{n-1}
\end{array}
$$

c_0 c_1 c_{n-1}

c_{2n-3} c_{2n-2}

计算实例

向量 $a = (1, 2, 4, 3)$, $b = (4, 2, 8, 0)$

则 $a+b = (5, 4, 12, 3)$

$a \cdot b = (4, 4, 32, 0)$

$a * b = (4, 10, 28, 36, 38, 24, 0)$

	ab_0	ab_1	ab_2	ab_3
a_0b	1×4	1×2	1×8	1×0
a_1b	2×4	2×2	2×8	2×0
a_2b	4×4	4×2	4×8	4×0
a_3b	3×4	3×2	3×8	3×0

$c_2 = 4\times4 + 2\times2 +1\times8 = 28$

卷积与多项式乘法

多项式乘法：$C(x) = A(x)\,B(x)$

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

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

$$C(x) = \underline{a_0b_0 + (a_0b_1+a_1b_0)\,x + \ldots}$$

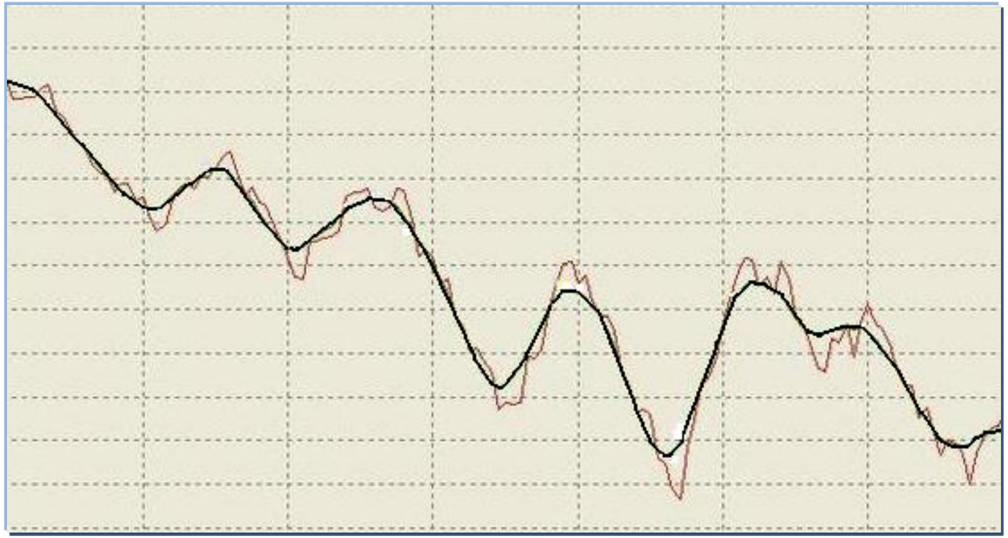
$$\underline{+\, a_{m-1}b_{n-1}\,x^{m+n-2}}$$

其中 x^k 的系数

$$c_k = \boxed{\sum_{\substack{i+j=k \\ i\in\{0,1,\ldots,m-1\} \\ j\in\{0,1,\ldots,n-1\}}} a_i b_j}, \quad k = 0, 1, \ldots, m+n-2$$

卷积应用：信号平滑处理

由于噪音干扰，对信号需要平滑处理

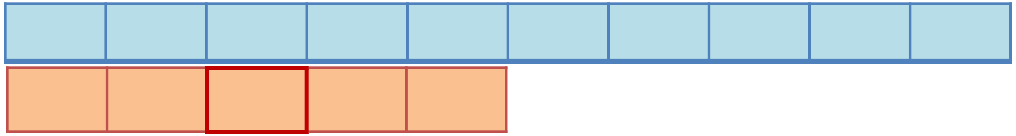


平滑处理

信号向量： $a=(a_0, a_1, \ldots, a_{m-1})$

$b=(b_{2k},b_{2k-1},\ldots,b_0) =(w_{-k}, \ldots, w_k)$

$$\boxed{a_i' = \sum_{s=-k}^{k} a_{i+s} b_{k-s} = \sum_{s=-k}^{k} a_{i+s} w_s}$$



把向量 b 看作$2k+1$长度窗口在a上移动计算 $a*b$, 得到 $(a_0', a_1',\ldots,a_{m-1}')$. 有少数项有误差.

实例

信号向量：　$a = (a_0, a_1, \ldots, a_8)$

步长：　$k = 2$

权值：　$w = (w_{-2}, w_{-1}, w_0, w_1, w_2)$

$= (b_4, b_3, b_2, b_1, b_0)$

$$a_i' = a_{i-2}b_4 + a_{i-1}b_3 + a_ib_2 + a_{i+1}b_1 + a_{i+2}b_0$$

下标之和为 $i + k$

$\boxed{w_{-2} \quad w_{-1} \quad w_0 \quad w_1 \quad w_2}$

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8

$$a_i' = a_{i-2}b_4 + a_{i-1}b_3 + a_ib_2 + a_{i+1}b_1 + a_{i+2}b_0$$

$a_0 b_0$	$a_0 b_1$	$a_0 b_2$	$a_0 b_3$	$a_0 b_4$	a_2'
$a_1 b_0$	$a_1 b_1$	$a_1 b_2$	$a_1 b_3$	$a_1 b_4$	a_3'
$a_2 b_0$	$a_2 b_1$	$a_2 b_2$	$a_2 b_3$	$a_2 b_4$	a_4'
$a_3 b_0$	$a_3 b_1$	$a_3 b_2$	$a_3 b_3$	$a_3 b_4$	a_5'
$a_4 b_0$	$a_4 b_1$	$a_4 b_2$	$a_4 b_3$	$a_4 b_4$	a_6'
$a_5 b_0$	$a_5 b_1$	$a_5 b_2$	$a_5 b_3$	$a_5 b_4$	
$a_6 b_0$	$a_6 b_1$	$a_6 b_2$	$a_6 b_3$	$a_6 b_4$	
$a_7 b_0$	$a_7 b_1$	$a_7 b_2$	$a_7 b_3$	$a_7 b_4$	
$a_8 b_0$	$a_8 b_1$	$a_8 b_2$	$a_8 b_3$	$a_8 b_4$	

小结

- 卷积的定义
- 卷积与多项式乘法的关系
- 卷积的应用——信号平滑处理

卷积计算

卷积计算：蛮力算法

向量 $a=(a_0,a_1,\ldots,a_{n-1})$和$b=(b_0,b_1,\ldots,b_{n-1})$

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

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

$$C(x)=A(x)B(x)$$

$$=a_0b_0+(a_0b_1+a_1b_0)\,x+\ldots+a_{n-1}b_{n-1}\,x^{2n-2}$$

$C(x)$的系数向量就是$a*b$.

卷积 $a*b$ 计算等价于多项式相乘

蛮力算法的时间：$O\,(n^2)$

计算2*n*-1次多项式*C*(*x*)

1. 选择值 $x_0, x_1, \ldots, x_{2n-1}$，
   求出 $A(x_j)$ 和 $B(x_j)$，
   $j = 0, 1, \ldots, 2n-1$

主要步骤:多项式求值

2. 对每个 j,计算$C(x_j)=A(x_j)B(x_j)$
3. 利用多项式插值方法，由$C(x)$ 在

   $$x = x_0, x_1, \ldots, x_{2n-1}$$

   的值求出多项式 $C(x)$的系数

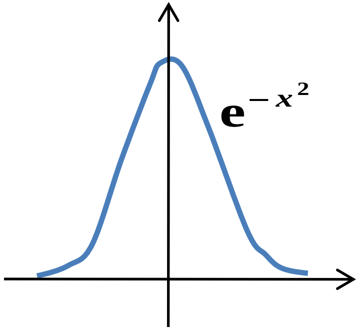
如何选择$x_0,x_1,\ldots,x_{2n-1}$的值？
高效多项式求值算法

高斯滤波的权值函数

高斯滤波的权值函数为

$$w_s = \frac{1}{z}\mathrm{e}^{-s^2}, \quad s = 0, \pm 1, \ldots, \pm k$$

$$w = (w_{-k}, \ldots, w_{-1}, w_0, w_1, \ldots, w_k)$$



其中 z 用于归一化处理，使所有的权值之和为1. 处理结果

$$a_i' = \sum_{s=-k}^{k} a_{i+s} w_s$$

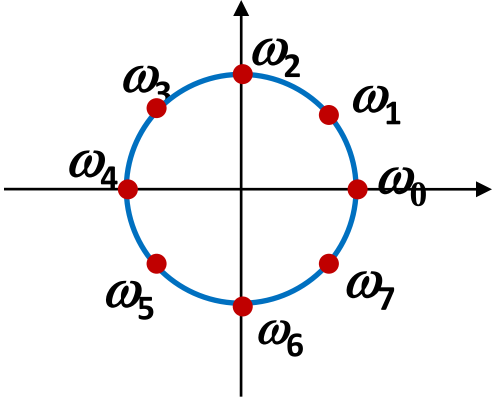
2*n*个数的选择:1的2*n*次根

$$\omega_j = \mathrm{e}^{\frac{2\pi j}{2n}\mathrm{i}} = \mathrm{e}^{\frac{\pi j}{n}\mathrm{i}}$$

$$= \cos\frac{\pi j}{n} + \mathrm{i}\sin\frac{\pi j}{n}$$

$j=0,1,\ldots,2n-1$， $\mathrm{i} = \sqrt{-1}$

n=4 的实例



$$\omega_0 = 1, \qquad \omega_1 = e^{\frac{\pi}{4}i} = \sqrt{2}/2 + \sqrt{2}/2 \cdot i,$$

$$\omega_2 = e^{\frac{\pi}{2}i} = i, \quad \omega_3 = e^{\frac{3\pi}{4}i} = -\sqrt{2}/2 + \sqrt{2}/2 \cdot i,$$

$$\omega_4 = e^{\pi i} = -1, \ \omega_5 = e^{\frac{5\pi}{4}i} = -\sqrt{2}/2 - \sqrt{2}/2 \cdot i,$$

$$\omega_6 = e^{\frac{3\pi}{2}i} = -i, \ \omega_7 = e^{\frac{7\pi}{4}i} = \sqrt{2}/2 - \sqrt{2/2} \cdot i$$

快速傅立叶变换FFT

1. 对 $x=1, \omega_1, \omega_2, \ldots, \omega_{2n-1}$, 分别计算 $A(x)$，$B(x)$

2. 利用步1的结果对每个 $x=1, \omega_1, \omega_2, \ldots, \omega_{2n-1}$, 计算 $C(x)$，得到 $C(1)=d_0, C(\omega_1)=d_1, \ldots, C(\omega_{2n-1})=d_{2n-1}$

3. 构造多项式

   $$D(x) = d_0+d_1x+d_2x^2+\ldots+d_{2n-1}x^{2n-1}$$

4. 对$x=1, \omega_1, \omega_2, \ldots, \omega_{2n-1}$, 计算$D(x)$, $D(1)$, $D(\omega_1)$, …, $D(\omega_{2n-1})$

快速傅立叶变换FFT(续)

可以证明：

$$D(1) = 2n\, c_0$$
$$D(\omega_1) = 2n\, c_{2n-1}$$
$$\cdots$$
$$D(\omega_{2n-1}) = 2nc_1$$

$$\Rightarrow$$

$$c_0 = D(1)/2n$$
$$c_{2n-1} = D(\omega_1)/2n$$
$$\cdots$$
$$c_1 = D(\omega_{2n-1})/2n$$

知道了$D(x)$的值，就能求$C(x)$的系数

算法的关键

令 $x = 1, \omega_1, \omega_2, \ldots, \omega_{2n-1}$,

- 步 1对 $2n$个x值分别求值多项式 $A(x), B(x)$
- 步2 做 $2n$次乘法
- 步3 对$2n$个x值求值多项式 $D(x)$

关键：一个对所有的 x 快速多项式求值算法

小结

卷积计算

- 蛮力算法$O(n^2)$
- 快速傅立叶变换FFT算法

  确定x的取值：1 的 $2n$ 次根

  关键步骤：多项式对x求值

如何设计多项式求值的快速算法？

快速傅立叶变换：FFT算法

多项式求值算法

给定多项式：

$$\boxed{A(x)=a_0+a_1x+\ldots+a_{n-1}x^{n-1}}$$

设 x 为1 的 $2n$ 次方根，对所有的 x 计算 $A(x)$ 的值.

算法1：对每个 x 做下述运算：

依次计算每个项 $a_i\, x^i$，$i=1, \ldots, n-1$

对 $a_i\, x^i$ $(i=0,1,\ldots,n-1)$求和.

蛮力算法的时间复杂度

$$T_1(n) = O(n^3)$$

改进的求值算法

算法2： 依次对 每个 x 做下述计算

$$A_1(x)=a_{n-1}$$

$$A_2(x)=a_{n-2}+xA_1(x)=a_{n-2}+a_{n-1}x$$

$$A_3(x)=a_{n-3}+xA_2(x)=a_{n-3}+a_{n-2}x+a_{n-1}x^2$$

$$\cdots$$

$$A_n(x)=a_0+x\,A_{n-1}(x)=A(x)$$

时间复杂度

$$T_2(n)=O(n^2)$$

偶系数与奇系数多项式

n=4

$$A(x)=a_0+a_1x+a_2x^2+a_3x^3$$

$$A_{\text{even}}(x) = a_0+a_2x$$

$$A_{\text{odd}}(x) = a_1+a_3x$$

$$\boxed{A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)}$$

$$= a_0+a_2x^2 + x(a_1+a_3x^2)$$

分治多项式求值算法

一般公式（n为偶数）

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

$$A_{\text{even}}(x)=a_0+a_2x+a_4x^2+\ldots+a_{n-2}x^{(n-2)/2}$$

$$A_{\text{odd}}(x)=a_1+a_3x+a_5x^2+\ldots+a_{n-1}\,x^{(n-2)/2}$$

$$\boxed{A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2)}$$

- x^2 也是1 的 $2n$ 次根
- 偶次数与奇次数多项式计算作为 $n/2$ 规模的子问题，然后利用子问题的解 $A_{\text{even}}(x^2)$ 与 $A_{\text{odd}}(x^2)$ 得到 $A(x)$

分治求值算法设计

算法 3:

1. 计算 1 的所有的 $2n$ 次根

   $1, \omega_1, \omega_2, \ldots, \omega_{2n-1}$

2. 分别计算$\boxed{A_{even}(x^2)}$与$\boxed{A_{odd}(x^2)}$

3. 利用步2 的结果计算 $A(x)$

   $A(x) = \boxed{A_{even}(x^2)} + x\boxed{A_{odd}(x^2)}$

注意：x^2不需要重新计算，所有根在单位圆间隔分布，隔一取一即可.

分治求值算法分析

$$T(n) = T_1(n) + f(n)$$

$f(n)=O(n)$ 是步 1 计算2n次根的时间

递归过程 $T_1(n) = 2T_1(n/2) + g(n)$

$T_1(1) = O(1)$,

$g(n) = O(n)$ 是对所有2n次根在步3组合解的时间

$$T_1(n)=O(n\log n)$$

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

FFT算法伪码

1. 求值$A(\omega_j)$和$B(\omega_j)$，$j=0,1,\ldots,2n-1$
2. 计算 $C(\omega_j)$，　　　　$j=0, 1, \ldots, 2n-1$
3. 构造多项式

$$D(x)=C(\omega_0)+C(\omega_1)x+\ldots+C(\omega_{2n-1})x^{2n-1}$$

4. 计算所有的 $D(\omega_j)$，$j=0,1,\ldots,2n-1$
5. 利用下式计算$C(x)$的系数 c_j，

$$D(\omega_j) = 2nc_{2n-j}$$

$$j = 0, 1, \ldots, 2n-1$$

FFT算法分析

步1: 求值 $A(\omega_j)$ 和 $B(\omega_j)$　$O(n\log n)$

步2: 计算所有的 $C(\omega_j)$　$O(n)$

步3:

步4: 计算所有的 $D(\omega_j)$　$O(n\log n)$

步5: 计算所有的 c_j　$O(n)$

算法时间为　$O(n\log n)$

小结

- 多项式求值算法

  蛮力算法：$O(n^3)$

  改进的求值算法：$O(n^2)$

  FFT算法：$O(n\log n)$

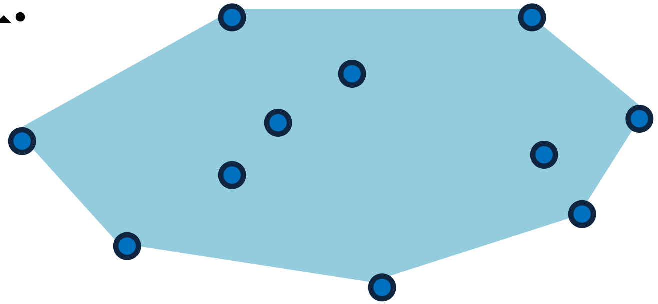
- FFT算法的设计与分析

平面点集的凸包

平面点集的凸包

问题（平面点集的凸包）

给定大量离散点的集合*Q*，求一个最小的凸多边形，使得*Q*中的点在该多边形内或者边上.

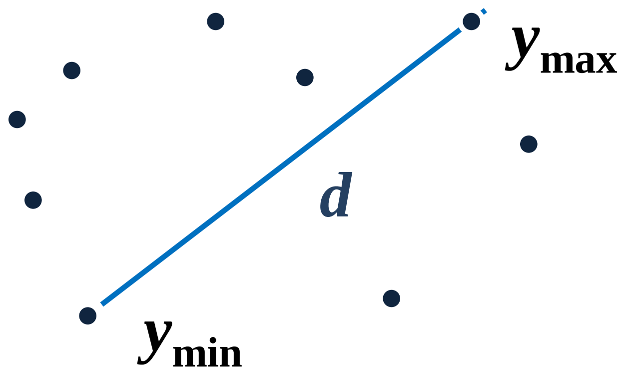


应用背景

图形处理中用于形状识别：字形识别、碰撞检测等

分治算法

1. 以 连接最大纵坐标点 y_{max} 和最小纵坐标点 y_{min} 的线段$d=\{y_{max},y_{min}\}$划分L 为左点集 L_{left} 和右点集 L_{right}



2. Deal (L_{left})；Deal (L_{right})

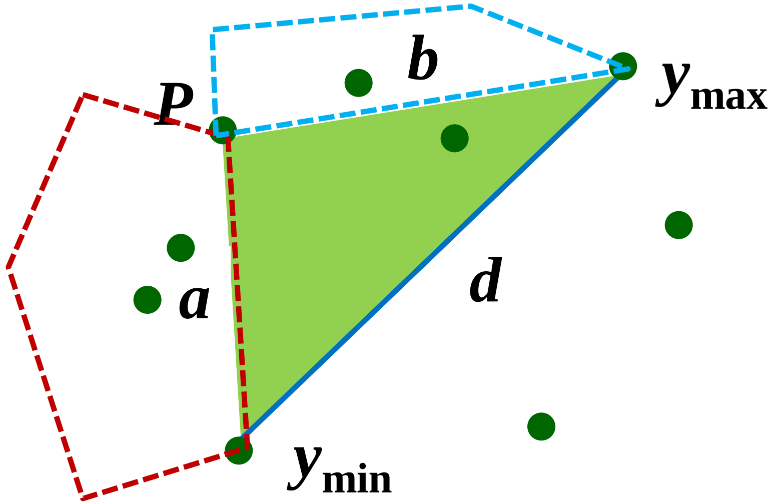
Deal (L_{left})

考虑 L_{left}：确定距d 最远的点P

在三角形内的点，删除；

a 外的点与 a 构成 L_{left} 的子问题；

b 外的点与 b 构成 L_{left} 的子问题 .



伪码

Deal (L_{left})

1. 以 d 和距离 d 最远点 P 构成三角形，P加入凸包，另外两条边分别记作 a 和 b
2. 检查 L_{left} 中其他点是否在三角形内；在则从 L中删除；否则根据在 a 或 b 边的外侧划分在两个子问题中
3. Deal (a)
4. Deal (b)

算法分析

- 初始用d划分 $O(n)$
- Deal 递归调用 $W(n)$
  - 找凸包顶点 P $O(n)$
  - 根据点的位置划分子问题 $O(n)$
- $$\boxed{\begin{aligned} &W(n) = W(n-1) + O(n) \\ &W(3) = O(1) \end{aligned}}$$

  最坏情况为$O(n^2)$

  $T(n) = O(n) + W(n) = O(n^2)$
- Graham扫描算法 $O\ (n\log n)$

小结：分治算法设计

- 将原问题归约为子问题
  直接划分注意尽量均衡
  通过计算归约为特殊的子问题
  子问题与原问题具有相同的性质
  子问题之间独立计算

- 算法实现：
  递归或迭代实现
  注意递归执行的边界

小结：分治算法的分析及改进

- 时间复杂度分析
  给出关于时间复杂度函数的递推方程和初值
  求解方程
- 提高效率的途径
  减少子问题个数
  预处理

重要的分治算法

- 检索算法：二分检索
- 排序算法：快速排序、二分归并排序
- 选择算法
- 快速傅立叶变换 FFT 算法
- 平面点集的凸包