第三章

Divide-and-Conquer技术

骆吉洲
计算机科学与工程系

提要

参考资料

《Introduction to Algorithms》
- 第33章: 33.3, 33.4

《网站资料》
- 第二章: 2.4, 2.5, 2.10

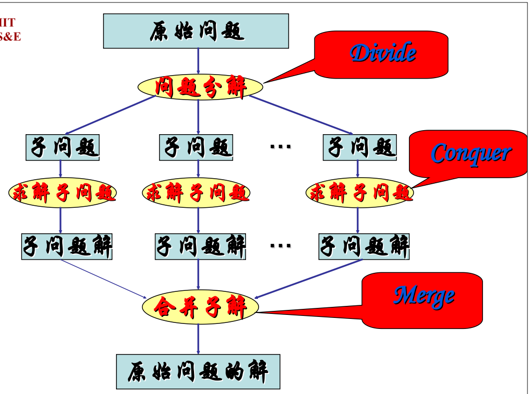
《网站资料》
- 第三章

3.1 Divide-and-Conquer原理

- Divide-and-Conquer算法的设计
- Divide-and-Conquer算法的分析

Divide-and-Conquer算法的设计

- 设计过程分为三个阶段
  - Divide: 整个问题划分为多个子问题
  - Conquer: 求解各子问题(递归调用正设计的算法)
  - Combine: 合并子问题的解, 形成原始问题的解



Divide-and-Conquer算法的分析

- 分析过程
  - 建立递归方程
  - 求解
- 递归方程的建立方法
  - 设输入大小为n, $T(n)$为时间复杂性
  - 当$n<c$, $T(n)=\theta(1)$

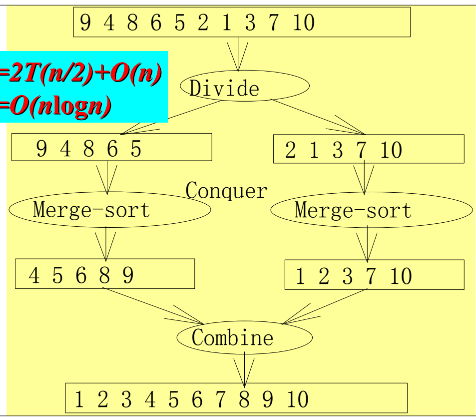
- Divide阶段的时间复杂性
  - 划分问题为a个子问题。
  - 每个子问题大小为n/b。
  - 划分时间可直接得到$=D(n)$
- Conquer阶段的时间复杂性
  - 递归调用
  - Conquer时间$= aT(n/b)$
- Combine阶段的时间复杂性
  - 时间可以直接得到$=C(n)$

- 总之
  - $T(n)=\theta(1)$ if $n<c$
  - $T(n)=aT(n/b)+D(n)+C(n)$ otherwise
- 求解递归方程$T(n)$
  - 使用第二章的方法

例1.

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

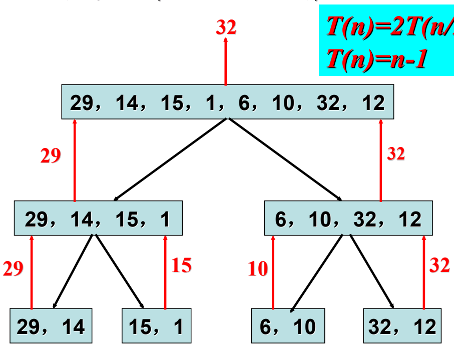
$T(n)=O(n\log n)$



例2. 求一个集合中的最大数算法

$T(n)=2T(n/2)+1$

$T(n)=n-1$



MaxMin问题

输入：数组$A[1,\ldots,n]$

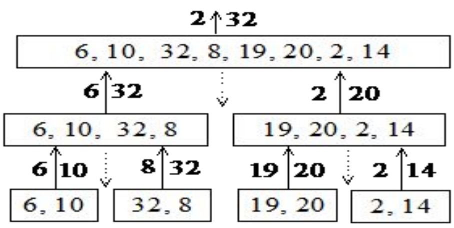
输出：A中的max和min

通常，直接扫面需要$2n-2$次比较操作

我们给出一个仅需$\lceil 3n/2-2\rceil$次比较操作的算法。

HIT CS&E

基本思想



HIT CS&E

算法

算法MaxMin(A)
输入: 数组$A[i,\ldots,j]$
输出:数组$A[i,\ldots,j]$中的max和min

1. If j-i+1 =1 Then 输出$A[i]$,$A[i]$,算法结束
2. If j-i+1 =2 Then
3. If $A[i]$< $A[j]$ Then输出$A[i]$,$A[j]$;算法结束
4. $k \leftarrow (j-i+1)/2$
5. $m_1,M_1 \leftarrow$MaxMin($A[i:k]$);
6. $m_2,M_2 \leftarrow$MaxMin($A[k+1:j]$);
7. $m \leftarrow \min(m_1,m_2)$;
8. $M \leftarrow \min(M_1,M_2)$;
9. 输出m,M

HIT CS&E

算法复杂性

$T(1)=0$
$T(2)=1$
$T(n)=2T(n/2)+2$
$=2^2T(n/2^2)+2^2+2$
$=\ldots$
$=2^{k-1}T(2)+2^{k-1}+2^{k-2}+\ldots+2^2+2$ $\qquad n=2^k$
$=2^{k-1}+2^k-1$
$=n/2+n-1$
$=3n/2-1$

HIT CS&E

3.2 整数乘法

HIT CS&E

问题定义

输入：n位二进制整数X和Y
输出：X和Y的乘积

通常，计算$X*Y$时间复杂性位$O(n^2)$，
我们给出一个复杂性为$O(n^{1.59})$的算法。

HIT CS&E

简单分治算法

X= | A (n/2位) | B (n/2位) | Y= | C (n/2位) | D (n/2位) |

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

算法

1. 划分产生A,B,C,D;
2. 计算n/2位乘法AC、AD、BC、BD;
3. 计算AD+BC;
4. AC左移n位，(AD+BC)左移n/2位;
5. 计算XY。

时间复杂性

$T(n)=4T(n/2)+\theta(n)$ $\qquad T(n)=\theta(n^2)$

算法的数学基础

$X=$ [A | B] (n/2位, n/2位)　　$Y=$ [C | D] (n/2位, n/2位)

$XY = (A2^{n/2} + B)(C2^{n/2} + D)$

$= AC2^{n} + (AD+BC)2^{n/2} + BD$

$AD+BC=(A-B)(C-D)+AC+BD$

算法

1. 划分产生A,B,C,D;
2. 计算A-B和C-D;
3. 计算$n/2$位乘法AC、BD、(A-B)(C-D);
4. 计算(A-B)(C-D)+AC+BD;
5. AC左移n位，((A-B)(C-D)+AC+BD)左移$n/2$位;
6. 计算XY

算法的分析

- 建立递归方程

  $T(n)=\theta(1)$　　if $n=1$

  $T(n)=3T(n/2)+O(n)$　　if $n>1$

- 使用Master定理

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

3.3 矩阵乘法

问题定义

输入：两个n× n矩阵A和B

输出：A和B的积

通常，计算XY的时间复杂性位$O(n^3)$，

我们给出一个复杂性为$O(n^{2.81})$的算法。

算法的数学基础

- 把C=AB中每个矩阵分成大小相同的4个子矩阵

  每个子矩阵都是一个$n/2 \times n/2$矩阵

- 于是

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

展开并整理等式的右边，即得到计算的方法

算法

- 计算$n/2 \times n/2$矩阵的10个加减和7个乘法

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

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

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

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

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

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

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

HIT CS&E

- 计算n/2×n/2矩阵的8个加减

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

HIT CS&E

算法复杂性分析

- 18个n/2×n/2矩阵加减法，每个需$O(n^2)$
- 7个n/2×n/2矩阵乘法
- 建立递归方程

$T(n)=O(1)$ n=2
$T(n)=7T(n/2) + O(n^2)$ n>2

- 使用Master定理求解T(n)

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

HIT CS&E

3.4 Finding the closest pair of points

HIT CS&E

问题定义

输入：Euclidean空间上的n个点的集合Q
输出：$P_1, P_2 \in Q$,
$Dis(P_1, P_2)=Min\{Dis(X, Y) \mid X, Y \in Q\}$

Dis(X, Y)是Euclidean距离：
如果$X=(x_1, x_2)$, $Y=(y_1, y_2)$，则

$$Dis(X,Y)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

HIT CS&E

一维空间算法

- 利用排序的算法
  - 算法
    - 把Q中的点排序
    - 通过排序集合的线性扫描找出最近点对
  - 时间复杂性
    - $T(n)=O(n\log n)$

HIT CS&E

一维空间算法(续)

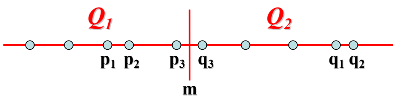
- Divide-and-conquer算法

Preprocessing:
1. 如果Q中仅包含2个点，则返回这个点对；
2. 求Q中点的中位数m。

Divide:

1. 用Q中点坐标中位数m把Q划分为两个大小相等的子集合

$Q_1 = \{x \in Q \mid x \le m\}$, $Q_2 = \{x \in Q \mid x > m\}$



Conquer:

1. 递归地在Q_1和Q_2中找出最接近点对 (p_1, p_2)和(q_1, q_2)

Merge:

2. 在(p_1, p_2)、(q_1, q_2)和某个(p_3, q_3)之间选择最接近点对(x, y)，其中p_3是Q_1中最大点，q_3是Q_2中最小点，(x, y)是Q中最接近点。

- 时间复杂性
  - Divide阶段需要$O(n)$时间
  - Conquer阶段需要$2T(n/2)$时间
  - Merge阶段需要$O(n)$时间
  - 递归方程

    $T(n) = O(1)$　　　$n = 2$

    $T(n) = 2T(n/2) + O(n)$　　　$n \ge 3$
  - 用Master定理求解$T(n)$

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

二维空间算法

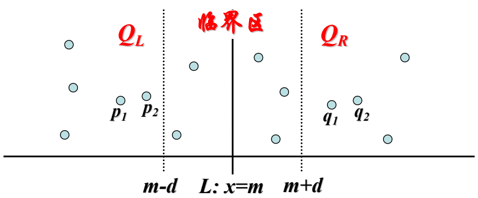
- Divide-and-conquer算法

Preprocessing:

1. 如果Q中仅包含一个点，则算法结束；
2. 把Q中点分别按x-坐标值和y-坐标值排序.

Divide:

1. 计算Q中各点x-坐标的中位数m;
2. 用垂线$L: x=m$把Q划分成两个大小相等的子集合Q_L和Q_R，Q_L中点在L左边，Q_R中点在L右边.



Divide:

1. 递归地在Q_L、Q_R中找出最接近点对: $(p_1, p_2) \in Q_L$, $(q_1, q_2) \in Q_R$
2. $d = \min\{Dis(p_1, p_2), Dis(q_1, q_2)\}$;

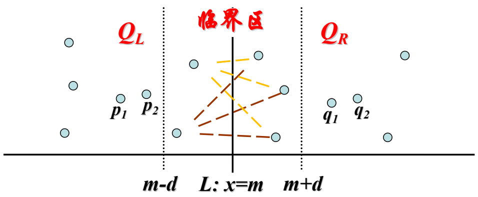
Merge:

1. 在临界区查找距离小于d的点对(p_l, q_r), $p_l \in Q_L$, $q_r \in Q_R$;
2. 如果找到，则(p_l, q_r)是Q中最接近点对，否则(p_1, p_2)和(q_1, q_2) 中距离最小者为Q中最接近点对.

关键是(p_l, q_r)的搜索方法及其搜索时间

(p_l, q_r)的直接搜索方法：

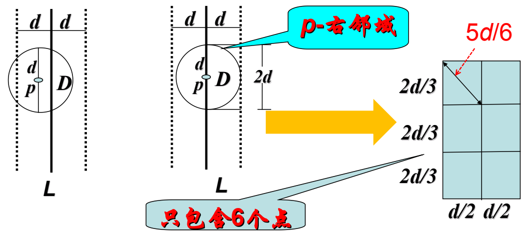
- $\forall p \in Q_L, \forall q \in Q_R$
  如果$dis(p, q)<d$，则更新d并记录(p, q)



- 合并的开销$\Theta(n^2)$
- $T(n)=2T(n/2)+\Theta(n^2)$
- $T(n)=\Theta(n^2)$

(p_l, q_r)的搜索方法：

- 如果(p, q)是最接近点对而且$p\in Q_L, q\in Q_R$，则$dis(p, q)<d$，(p, q)只能在下图的区域D.
- 若p在分割线L上，包含(p, q)的区域D最大，嵌于$d\times 2d$的矩形(p-右邻域)中，如下图所示.



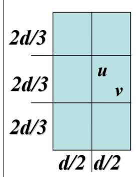
- 对于任意p，我们只需在p-右邻域中点q，最多6个.
- 算法
  1. 把临界区中所有点集合投影到分割线L上；
  2. 对于左临界区的每个点p，考察p-右临界区的每个点(这样的点共有6个) q，如果$Dis(p, q)<d$，则令$d=Dis(p, q)$;
  3. 如果d发生过变化，与最后的d对应的点对即为 (p_l, q_r)，否则不存在(p_l, q_r).

- 时间复杂性
  - Divide阶段需要$O(n)$时间
  - Conquer阶段需要$2T(n/2)$时间
  - Merge阶段需要$O(n)$时间
  - 递归方程
    $T(n)=O(1)$　　　　$n=2$
    $T(n)=2T(n/2)+O(n)$　　$n\geq 3$
  - 用Master定理求解$T(n)$
    $T(n)=O(n\log n)$

- 正确性分析

定理1. 对于左临界区中的每个点p，p-右邻域中仅包含6个点。

证明：把p-右邻域划分为6个$(d/2)\times(2d/3)$的矩形。若p-右邻域中点数大于6，由鸽巢原理，至少有一个矩形中有两个点，设为u、v.



$(x_u-x_v)^2+(y_u-y_v)^2\leq (d/2)^2+(2d/3)^2=25d^2/36$

即$Dis(u, v)\leq 5d/6<d$，与d的定义矛盾。

3.5 Finding the convex hull

HIT CS&E

问题定义

输入：平面上的n个点的集合Q

输出：CH(Q): Q的convex hull

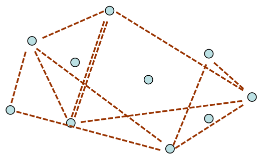
Q的convex hull是一个凸多边形P，Q的点或者在P上或者在P内

凸多边形P是具有如下性质多边形：连接P内任意两点的边都在P内

HIT CS&E

蛮力算法

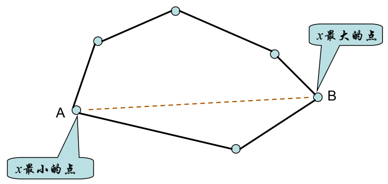
命题1:给定平面点集S，如果$P,P_i,P_j,P_k \in S$是四个不同的点，且P位于三角形$DP_iP_jP_k$的内部或边界上，则P不是S的凸包顶点



HIT CS&E

蛮力算法处理剩余点

- A是横坐标最小的点
- B是横坐标最大的点
- AB上方的点按横坐标递减排序得S_U
- AB下方的点按横坐标递增排序S_L
- 顺序输出A,S_L,B,S_U



HIT CS&E

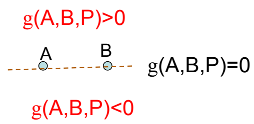
蛮力算法基本操作

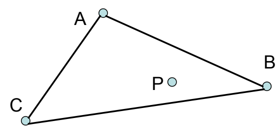
直线AB划分平面

g(A,B,P)=0
g(A,B,P)<0
g(A,B,P)>0

P∈ΔABC

g(A,B,P)· g(A,B,C) ≥0
g(A,C,P)· g(A,C,B) ≥0
g(B,C,P)· g(B,C,A) ≥0





- **算法BruteForceCH(Q)**

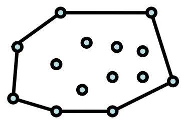
1. For ∀A,B,C,D∈Q Do /*4层循环*/
2. If 其中一点位于其他三点构成的三角形内 Then
3. 从Q中删除该点
4. A←Q中横坐标最大的点;
5. B←Q中横坐标最小的点;
6. S_L ←{P | P∈ Q且g(A,B,P)<0};
7. S_U ←{P | P∈ Q且g(A,B,P)>0};
8. 排序S_L,S_U
9. 输出A,S_L,B,逆序S_U;

$T(n)=\theta(n^4)$

HIT CS&E

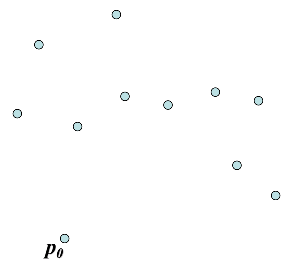
Graham-Scan算法

- **基本思想**
  - 当沿着Convex hull逆时针漫游时，总是向左转
  - 在极坐标系下按照极角大小排列，然后逆时针方向漫游点集，去除非Convex hull顶点(非左转点)。



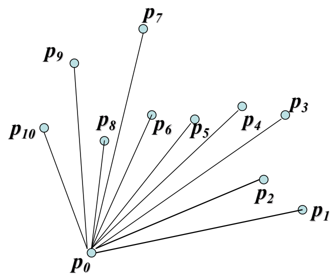
HIT CS&E

第1步



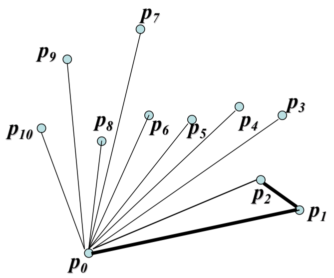
HIT CS&E

第2步



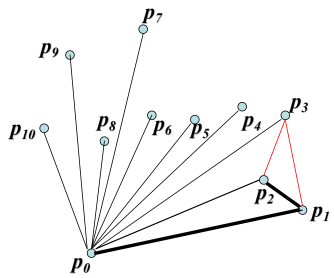
HIT CS&E

第3步



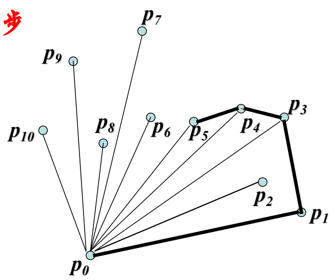
HIT CS&E

第3步



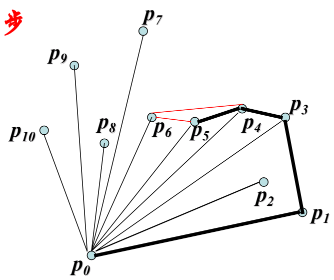
HIT CS&E

第4-7步



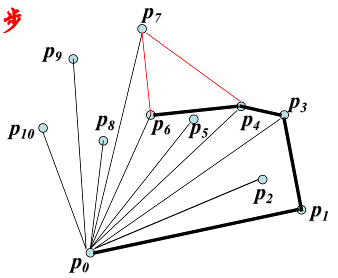
HIT CS&E

第4-7步



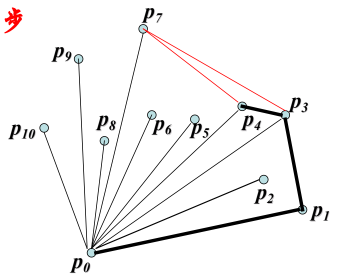
HIT
CS&E

第4-7步



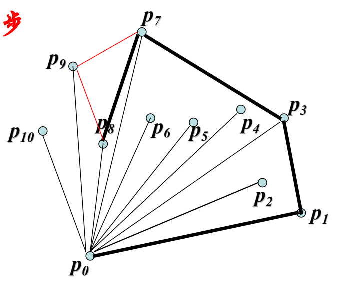
HIT
CS&E

第4-7步



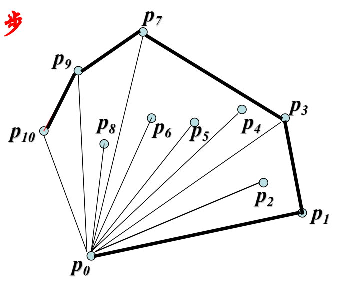
HIT
CS&E

第4-7步



HIT
CS&E

第4-7步



- **算法Graham-Scan(Q)**

/* 栈S从底到顶存储按逆时针方向排列的CH(Q)顶点 */
1. 求Q中y-坐标值最小的点p_0;
2. 按照与p_0极角(逆时针方向)大小排序Q中其余点，结果为$<p_1, p_2, \cdots, p_m>$;
3. Push(p_0, S); Push(p_1, S); Push(p_2, S);
4. FOR i=3 TO m DO
5. While Next-to-top(S)、Top(S)和p_i形成非左移动 Do
6. Pop(S);
7. Push(p_i, S);
8. Rerurn S.

HIT
CS&E

- **时间复杂性**
  - 第1步需要$O(1)$时间
  - 第2步需要$O(n\log n)$时间
  - 第3步需要$O(1)$时间
  - 第4-7步需要$O(n)$时间
  - 总时间复杂性$T(n)= O(n\log n)$

- **正确性分析**

  定理. 设n个二维点的集合Q是Graham-Scan算法的输入，$|Q|\geq 3$，算法结束时，栈S中自底到顶存储CH(Q)的顶点（按照逆时针顺序）.

  证明: 用归纳法证明：在第i次(i始于3) for循环结束时，栈S中自底到顶存储CH(Q_i)的顶点(按照逆时针顺序), $Q_i=\{p_0, p_1, \ldots, p_i\}$.

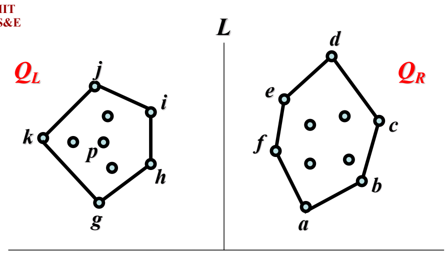
Divide-and-conquer算法

Preprocess: (时间复杂性为$O(1)$)

1. 如果|Q|<3, 算法停止;
2. 如果|Q|=3, 按照逆时针方向输出CH(Q)的顶点;

Divide:(时间复杂性为$O(n)$)

1. 选择一个垂直于x-轴的直线把Q划分为基本相等的两个集合Q_L和Q_R, Q_L在Q_R的左边;



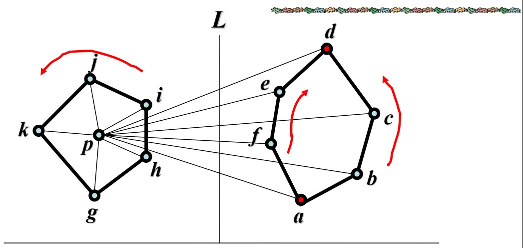
Conquer: (时间复杂性为$2T(n/2)$)

1. 递归地为Q_L和Q_R构造CH(Q_L)和CH(Q_R);

Merge:

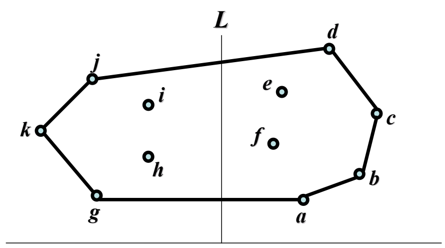
我们先通过一个例子来看Merge的思想

Merge实例



3个序列：<g, h, i, j, k>, <a, b, c, d>, <f, e>

合并以后：<g, h, a, b, f, c, e, d, i, j, k>



Graham-Scan

Merge:(时间复杂性为$O(n)$)

1. 找一个Q_L的内点p;
2. 在CH(Q_R)中找与p的极角最大和最小顶点u和v;
3. 构造如下三个点序列:
   (1) 按逆时针方向排列的CH(Q_L)的所有顶点,
   (2) 按逆时针方向排列的CH(Q_R)从u到v的顶点,
   (3) 按顺时针方向排列的CH(Q_R)从u到v的顶点;
4. 合并上述三个序列;
5. 在合并的序列上应用Graham-Scan.

时间复杂性

- Preprocessing阶段
  – $O(1)$
- Divide阶段
  – $O(n)$
- Conquer阶段
  – $2T(n/2)$
- Merge阶段
  – $O(n)$

时间复杂性

- 总的时间复杂性
  $T(n)=2T(n/2)+O(n)$
- 使用Master定理
  $T(n)= O(n\log n)$

3.6 剪枝搜索（Prune and search）

- 剪除与问题求解无关的数据,
- 剪除输入规模的αn个数据, 0<α<1
  – 剪枝的代价记为$P(n)$
- 对剩下的数据递归调用
  – $T(n)=T((1-\alpha)n)+P(n)$
- 利用第二章的技术分析算法复杂性

在有序数组中查找元素x

$A[1],A[2],....,A[k-1],\ A[k],A[k+1]......,A[n]$

x

- 将数组分为三个部分，$A[1:k-1]$,$A[k]$,$A[k+1:n]$
- 通过比较x=?$A[k]$,删除其中两个部分
- 为使任何情况下均至少删除一半以上的元素
  – 取$k=n/2$
- $T(n)=T(n/2)+1$　　　　$T(n)=O(\log n)$

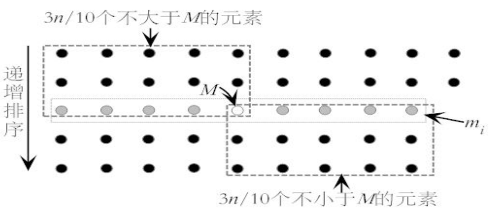
中位数线性时间选择

–本节讨论如何在$O(n)$时间内从n个不同的数中选取第i大的元素

–中位数问题也就解决了，因为选取中位数即选择第$n/2$-大的元素

Input: n个(不同)数构成的集合X,整数i,其中$1\leq i\leq n$

Output: $x\in X$使得X中恰有i-1个元素小于x



•选取M,将输入数据划分为三个部分$A[j]<M$, $A[j]=M$，$A[j]>M$

•判定目标元素在哪个部分

—若$|A[j]<M|\ >i$, 在第一个部分，对第一部分递归调用

—若$|A[j]<M|+|A[j]=x|\ >i$, 在第二个部分，输出第二部分任意元素

—否则，在第三个部分，对第三部分递归调用

•如何选择M，才能确保至少删掉常数因子个数据

算法Select(A,i)

Input: 数组$A[1:n]$, $1\le i\le n$

Output: $A[1:n]$中的第i-大的数

```
for j←1 to n/5
    InsertSort(A[(j-1)*5+1 : (j-1)*5+5]);
    swap(A[j], A[[(j-1)*5+3]);
x ←Select(A[1: n/5], n/10 );
k ←partition(A[1:n], x);
if      k=i then   return x;
else if  k>i then   retrun Select(A[1:k-1],i);
else                retrun Select(A[k+1:n],i-k);
```

$$T(n)\le\begin{cases}\Theta(1) & \text{if } n\le C\\ T\lfloor n/5\rfloor+T(7n/10+6)+O(n) & \text{if } n>C\end{cases}$$

HIT CS&E

补充材料

- 剪枝搜索的更多实例（3.7,3.8）
- 基于比较的排序算法（3.9）
- 线性时间排序算法（3.10）
- 排序算法时间复杂度下界（3. 11）

HIT CS&E

3.7 二元线性规划

HIT CS&E

问题定义

min $ax+by$

s.t. $a_1x+b_1y\ge c_1$

$a_2x+b_2y\ge c_2$

$a_3x+b_3y\ge c_3$

……

$a_nx+b_ny\ge c_n$

输入：实数a,b，实数数组$A[1:n]$,$B[1:n]$和$C[1:n]$

输出：x^*,y^*使得 $A[i]x^*+B[i]y^*\ge C[i]$对$i=1,2,\ldots,n$成立且ax^*+by^*达到最大值

HIT CS&E

问题的特殊形式

min y

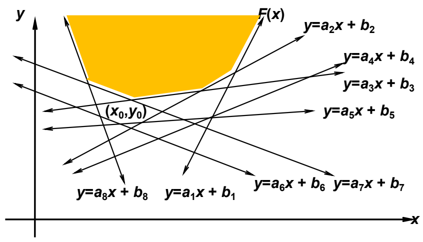
s.t. $y\ge a_1x+b_1$

$y\ge a_2x+b_2$

$y\ge a_3x+b_3$

……

$y\ge a_nx+b_n$



边界函数$F(x)$:

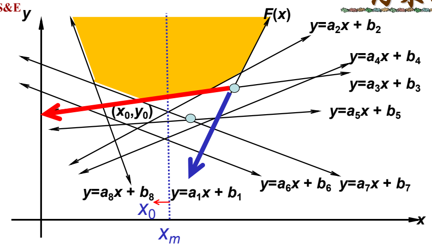
$F(x)=\max\limits_{1\le i\le n}\{a_ix+b_i\}$

最优解x_0:

$y_0=F(x_0)=\min\limits_{-\infty<x<\infty}F(x)$

HIT CS&E

约束的剪除



给定一个x_m,且已知$x_0<x_m$

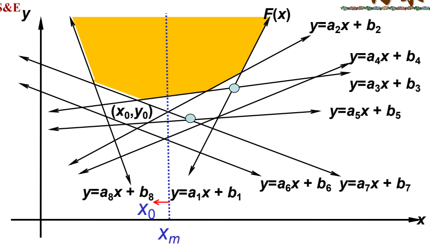
你能判断哪些约束有用，哪些约束无用吗?

由于$a_1x+b_1< a_3x+b_3$在$x_0<x_m$恒成立

约束$y\geq a_1x+b_1$对求解问题无用

HIT CS&E

约束的剪除



给定一个x_m,且已知$x_0<x_m$

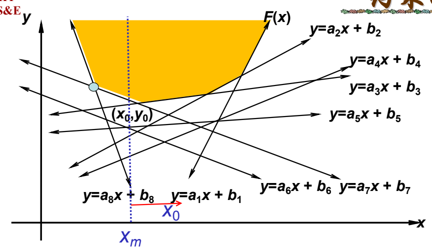
对右侧的每个交点，均能剪除斜率大的约束

例如：$y\geq a_1x+b_1$可剪除

$y\geq a_5x+b_5$可剪除

HIT CS&E

约束的剪除



同理：给定一个x_m,且已知$x_m < x_0$

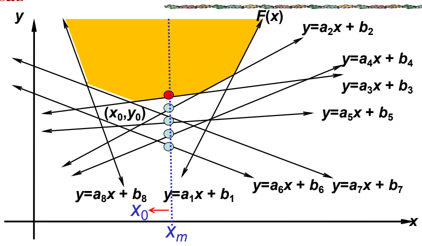
对左侧的每个交点，均能剪除斜率小的约束

例如：$y\geq a_8x+b_8$可剪除

$y\geq a_6x+b_6$可剪除

HIT CS&E

已知x_m判断$x_0<x_m$或$x_0>x_m$



令 $y_m=F(x_m)=\max\limits_{1\leq i\leq n}\{a_ix_m+b_i\}$

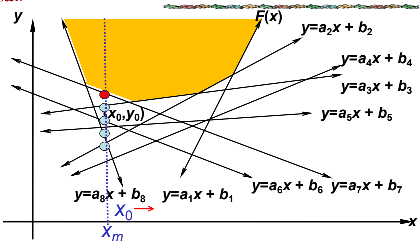
情形1：$y_m=a_kx_m+b_k$仅在一个约束上取到

若$a_k>0$,则$x_0<x_m$

若$a_k<0$,则$x_0>x_m$

HIT CS&E

已知x_m判断$x_0<x_m$或$x_0>x_m$



令 $y_m=F(x_m)=\max\limits_{1\leq i\leq n}\{a_ix_m+b_i\}$

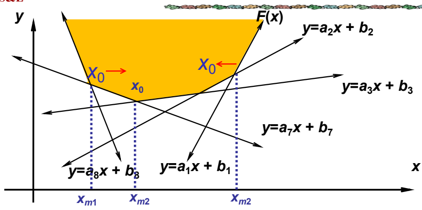
情形1：$y_m=a_kx_m+b_k$仅在一个约束上取到

若$a_k>0$,则$x_0<x_m$

若$a_k<0$,则$x_0>x_m$

HIT CS&E

已知x_m判断$x_0<x_m$或$x_0>x_m$



令 $y_m=F(x_m)=\max\limits_{1\leq i\leq n}\{a_ix_m+b_i\}$

情形2：y_m在多个约束上取到，令$g_{min}=\min\{a_k \mid y_m=a_kx_m+b_k\}$

$g_{max}=\max\{a_k \mid y_m=a_kx_m+b_k\}$

(1) 若$g_{min}<0$, $g_{max}<0$，则$x_0>x_m$

(2) 若$g_{min}<0$, $g_{max}>0$，则$x_0=x_m$

(3) 若$g_{min}>0$, $g_{max}>0$，则$x_0<x_m$

HIT CS&E

确定x_m

一旦确定了x_m

- 可以判定最优解$x_0<x_m$还是$x_0>x_m$
- 如果$x_0<x_m$,则右侧每个交点可剪除一个约束
- 如果$x_0>x_m$,则右侧每个交点可剪除一个约束

问题：如何选择x_m，总能剪除一定比例约束?

如果确定了x_m,

- 使其左侧和右侧均至少有$n/2$个交点
- 如果$x_0<x_m$,则可剪除$n/4$个约束
- 如果$x_0>x_m$,则可剪除$n/4$个约束

HIT CS&E

算法

算法SpecTwoLinear($A[1:n]$,$B[1:n]$)

输入: 实数数组$A[1:n]$,$B[1:n]$

输出： x_0,y_0使$y_0\geq A[i]x_0+B[i]$ ($i=1,\dots,n$)且ax_0+by_0达到最大值

1. If $n=2$ Then 输出 $y=A[1]x+B[1]$和$y=A[2]x+B[2]$的交点,结束
2. 求$y=A[i]x+B[i]$和$y=A[i+1]x+B[i+1]$的交点(x_i,y_i)，$i=1,3,5,\dots$
3. 求交点横坐标的中位数x_m
4. $y_m=F(x_m)=\max\{A[i]x_m+B[i] \mid 1\leq i\leq n\}$
   $g_{min}=\min\{A[k] \mid y_m=A[k]x_m+B[k]\}$
   $g_{max}=\max\{A[k] \mid y_m=A[k]x_m+B[k]\}$
5. If $g_{min}\cdot g_{max}<0$ Then 输出x_m,y_m，算法结束
6. Else If $g_{min}<0$, $g_{max}<0$ Then /*$x_0>x_m$*/
   利用 满足$x_i<x_m$的每个交点，剪除一个斜率较小的约束
7. Else /*$g_{min}>0$, $g_{max}>0$， $x_0<x_m$*/
   利用 满足$x_i>x_m$的每个交点，剪除一个斜率较大的约束
8. SpecTwoLinear($A[1:3n/4]$,$B[1:3n/4]$) /*剪除后剩下的约束*/

HIT CS&E

时间复杂性

$T(n)=\theta(1)$

$T(n)=T(3n/4)+\theta(n)$

$T(n)=\theta(n)$

HIT CS&E

一般的二元线性规划

min $ax+by$

s.t. $a_1x+b_1y\geq c_1$

$a_2x+b_2y\geq c_2$

……

$a_nx+b_ny\geq c_n$

$L\leq x\leq U$

$$\xrightarrow[y'=ax+by]{x'=x}$$

min y'

s.t.

$[a_1-ab_1/b]x'+(b_1/b)y'\geq c_1$

……

$[a_n-ab_n/b]x'+(b_n/b)y'\geq c_n$

$L\leq x'\leq U$

min y'

s.t.

$(b_1/b)y'\geq [ab_1/b-a_1]x'+c_1$

……

$(b_n/b)y'\geq [ab_n/b-a_n]x'+c_n$

$L\leq x'\leq U$

min y'

s.t.

$y'\geq [a-a_ib/b_i]x'+c_ib/b_i$ $i\in I$

$y'\leq [a-a_jb/b_j]x'+c_jb/b_j$ $j\in J$

$L\leq x'\leq U$

HIT CS&E

问题转换

min y

s.t. $y\geq a_ix+b_i$ $i\in I$

$y\leq a_jx+b_j$ $j\in J$

$L\leq x\leq U$

⇩

min $F_1(x)$

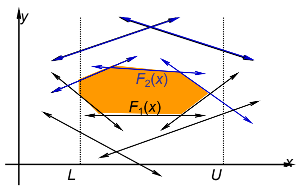
s.t. $F_1(x)\leq F_2(x)$

$L\leq x\leq U$

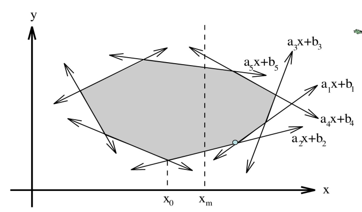
令 $F_1(x)=\max\{a_ix+b_i,\ i\in I\}$

$F_2(x)=\min\{a_jx+b_j,\ j\in J\}$

$F(x)=F_1(x)-F_2(x)$



约束的剪除



若已知 $x_0<x_m$, 则a_1x+b_1可被剪除

$a_1x+b_1<a_2x+b_2$ 在 $x<x_m$时恒成立

定义

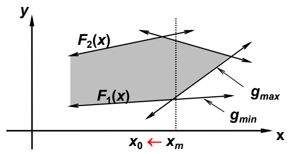
- $g_{min}=\min\{a_i \mid i\in I, a_ix_m+b_i=F_1(x_m)\}$, 最小斜率
- $g_{max}=\max\{a_i \mid i\in I, a_ix_m+b_i=F_1(x_m)\}$, 最大斜率
- $h_{min}=\min\{a_i \mid i\in J, a_ix_m+b_i=F_2(x_m)\}$, 最小斜率
- $h_{max}=\max\{a_i \mid i\in J, a_ix_m+b_i=F_2(x_m)\}$, 最大斜率

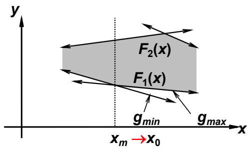
已知x_m判断$x_0<x_m$或$x_0>x_m$

情形1：x_m是可行解，即$F(x_m)=F_1(x_m)-F_2(x_m)\le 0$

(1)如果$g_{min}>0, g_{max}>0$，
则$x_0<x_m$

(2)如果$g_{min}<0, g_{max}<0$，
则$x_0>x_m$





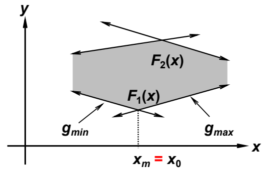
已知x_m判断$x_0<x_m$或$x_0>x_m$

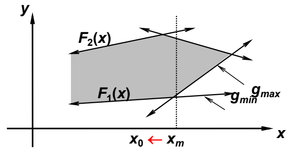
情形1：x_m是可行解，即$F(x_m)=F_1(x_m)-F_2(x_m)\le 0$

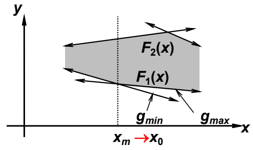
(1)如果$g_{min}>0, g_{max}>0$，
则$x_0<x_m$

(2)如果$g_{min}<0, g_{max}<0$，
则$x_0>x_m$

(3)如果$g_{min}<0, g_{max}>0$，
则$x_0=x_m$







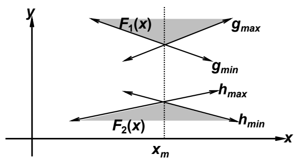
已知x_m判断$x_0<x_m$或$x_0>x_m$

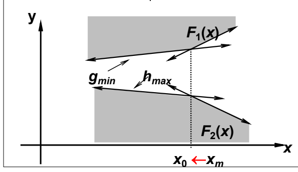
情形2：x_m不是可行解，即$F(x_m)=F_1(x_m)-F_2(x_m)>0$

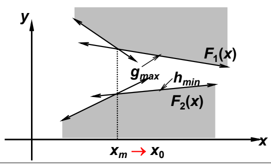
(1)如果$h_{max}<g_{min}$，
则$x_0<x_m$

(2)如果$g_{max}<h_{min}$，
则$x_0>x_m$

(3)如果$g_{min}\le h_{max}$且$g_{max}\ge h_{min}$
则问题无解







二元线性规划剪枝算法

算法 TwoLinear($A[1:n]$,$B[1:n]$,n_1)

输入：实数数组$A[1:n]$,$B[1:n]$，n_1　　/*$i\in I$ ($i\le I$) $i\in J$ ($i>n_1$)*/

输出：x_0,y_0使$y_0\ge(\le)A[i]x_0+B[i]$ ($i\le n_1$,$i>n_1$)且ax_0+by_0达到最大值

1. If |I|=2 且|J|=2 Then 用蛮力法求解,算法结束
2. 将I中约束两两配对，不同的对无公共约束
3. 将J中约束两两配对，不同的对无公共约束
4. 求$n/2$对约束的交点横坐标的**中值数x_m**
5. 如果对应优化解，则输出x_m,y_m，算法结束
6. 如果由x_m知问题无解，则算法结束
7. 根据最优解在x_m的左侧还是右侧，剪除每对约束中的一个约束
   共剪除$n/4$个约束得到$A[1:3n/4]$, $B[1:3n/4]$, n'
8. TwoLinear($A[1:3n/4]$,$B[1:3n/4]$,n');

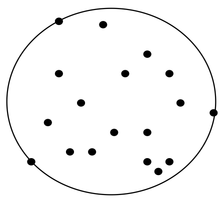
$T(n)=T(3n/4)+\theta(n)$
$\quad = \theta(n)$

3.8　1圆心问题

问题定义

输入：平面上n个点

输出：一个半径最小的圆使其包含所有输入点



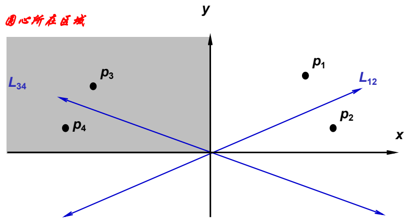
剪枝策略

L_{12}：p_1,p_2的垂直平分线

L_{34}：p_3,p_4的垂直平分线

L_{12},L_{34}交点到p_1p_2,p_3p_4的距离决定圆心所在区域

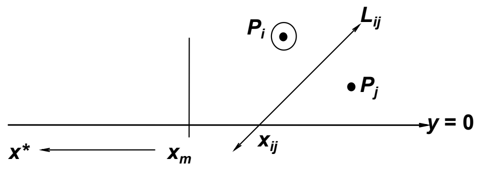
如果p_1到区域的距离小于p_2到区域的距离，p_1可删除



受限1-圆心问题

输入：平面上n个点和一条直线$y=y'$

输出：圆心位于$y=y'$上的半径最小的圆包含所有点



受限1-圆心算法

算法Constraint1Center(P[1:n],y')

输入：平面上n个点和一条直线$y=y'$

输出：圆心在$y=y'$上的半径最小的圆包含所有点

1. If $n \leq 2$ Then 用蛮力法求解圆心，算法结束
2. 输入点配对$(p_1,p_2),(p_3,p_4),\ldots,(p_{n-1},p_n)$;如果$n$是奇数，则最后一个点对为$(p_n,p_1)$
3. 计算(p_i,p_j)中垂线与y=y'的交点横坐标$x_{i,i+1}$ (i =1,3,..,n/2)
4. 计算$x_{i,i+1}$ (i =1,3,..,n/2)的中位数x_m
5. 计算距离(x_m,y')最远的输入点(x_j,y_j)。

   /*$x_j<x_m$,则圆心在x_m左侧,即$x^*<x_m$; 否则,$x^*>x_m$*/
6. If $x^*<x_m$ Then对$x_{i,i+1}>x_m$的(p_i,p_{i+1})剪除距(x_m,y')较近点

   Else 对$x_{i,i+1}<x_m$的(p_i,p_{i+1})剪除距(x_m,y')较近点
7. 对剩余输入点和直线$y=y'$递归调用算法

算法的复杂度

$T(n) = \theta(1)$

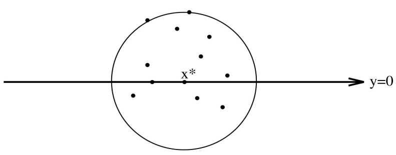
$T(n) = T(3n/4) + \theta(n)$

由此解得

$T(n) = \theta(n)$

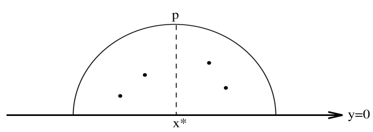
一般情况的处理

- 用受限1圆心算法，可以计算出直线y=0上的圆心$(x^*,0)$.
- 而且，用受限1圆心算法还可以
  - 令(x_s, y_s)表示最优解的圆心.
  - 我们可以判定 $y_s>0$, $y_s<0$ 还是 $y_s=0$.
  - 类似地，我们可以判定$x_s>0$, $x_s<0$ 还是 $x_s=0$



y_s的符号

- 令$(x^*,0)$是直线y=0上的最小圆圆心
- I是距离$(x^*,0)$最远的输入点构成的集合
- 情形1：$|I|=1$，$I=\{p\}$，则y_s与y_p符号相同



HIT CS&E

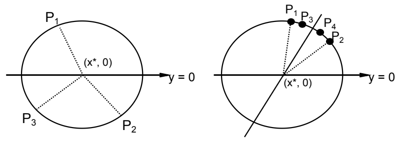
• 情形2: $|I|>1$

找出I中输入点张成的最小圆弧

圆弧端点记为为 $p_1=(x_1,y_1), p_2=(x_2,y_2)$

(a)如果圆弧>180°, 则 $y_s=0$

(b)如果圆弧≤180°, 则 y_s与 $(y_1+y_2)/2$同符号

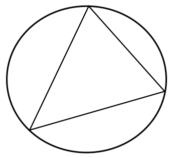


(a) (b)

HIT CS&E

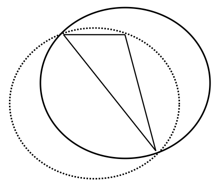
最优性判定

• 锐角三角形:



最优圆.

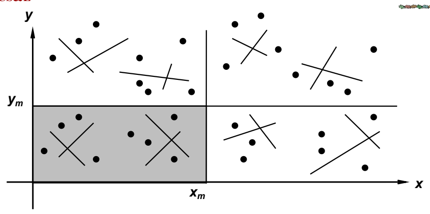
• 钝角三角形:



非最优圆.

HIT CS&E

例子



$x_s>x_m, y_s>y_m$,

阴影区域$n/8$个点对，每对可剪除一个点$n/4$

共可剪除n/16个点

其他情况类似

算法

算法OneCenter(S)

输入: 含n个点的平面点集S = {p_1, p_2, …, p_n}

输出: 覆盖S的最小圆圆心.

1. If $|S|\le 16$ Then 用蛮力法求解得到圆心，算法结束.
2. 将n点配对(p_1, p_2), (p_3, p_4), …,(p_{n-1}, p_n).
   计算(p_{i-1}, p_i)垂直平分线$L_{i/2}$及其斜率$s_{i/2}$, i=2,4, …,n
3. 计算{s_k| k=1,2,…,n/2} 的中位数s_m
4. 旋转坐标轴使得x-轴与直线$y=s_m x$重合
   $I^+ = \{L_i | s_i > 0\}$ $I^- = \{L_i | s_i < 0\}$ /* $|I^+|\approx|I^-|\approx n/4$ */
5. 构造直线对(L_i,l_i),$L_i\in I^+$,$l_i\in I^-$， i=1,…,n/4, 无公共直线
   计算L_i和l_i的交点(a_i,b_i), 计算b_1,…,$b_{n/4}$的中位数y^*
6. (x',y^*) ←Constraint1Center(S[1:n],y^*);
7. 如果(x',y^*) 是优化解，返回，算法终止;
8. 否则，记录$y_s<y^*$还是$y>y^*$

---

9. 计算$a_1,a_2,\ldots,a_{n/4}$的中位数x^*
10. (x^*,y') ←Constraint1Center(S[1:n],$x=x^*$);
11. 如果(x^*,y') 是优化解，返回，算法终止;
12. 否则，记录$x_s<x^*$还是$x>x^*$
13. 根据四种情况删除S中n/16个点
    情形1: $x_s<x^*$且$y_s<y^*$
    对每个满足$a_i>x^*$ 且$b_i>y^*$的交点(a_i, b_i)，设它是$L_i\in I^+$和$l_i\in I^-$的交点而l_i是(p_j,p_k)的中垂线，则从S中删除 p_j和p_k中距离 (x^*, y^*)更近的顶点.
    情形2: $x_s<x^*$且$y_s>y^*$, 类似地处理
    情形3: $x_s>x^*$且$y_s>y^*$, 类似地处理
    情形4: $x_s>x^*$且$y_s<y^*$, 类似地处理
14. 输出OneCenter(S) /*递归调用*/

HIT CS&E

时间复杂性

$T(n) = \theta(1)$ $n<16$

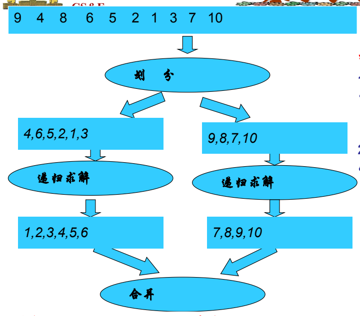
$T(n) = T(15n/16) + \theta(n)$ $n\ge 16$

由此解得

$T(n) = \theta(n)$

3.9基于分治思想的排序算法

HIT CS&E



划分的策略

1. 选择一个位置将数组 划分成两个部分mergesort

2. 选择一个划分标准x根据元素与x的大小关系来划分
quicksort

合并策略

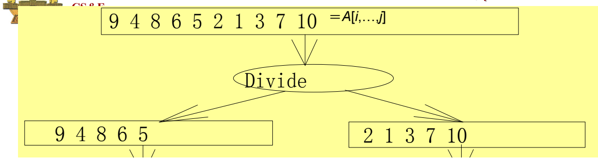
不同的划分策略对应不同的合并策略

由于分治思想涉及到递归调用，需要关心子问题的最一般形式

在排序问题中，子问题一般形式就是将$A[i,\ldots,j]$中的元素排序

3.9.1 Merge-sort算法

HIT CS&E



Divide：

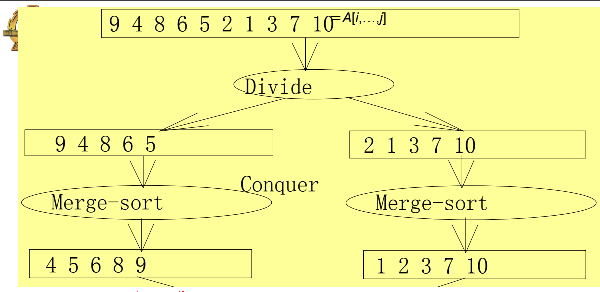
本质上仅需产生划分位置k

第一个子问题是$A[i,\ldots,k]$

第二个子问题是$A[k+1,\ldots,j]$

为使得两个问题的大小大致相当，k可以如下产生

$k=(i+j)/2$



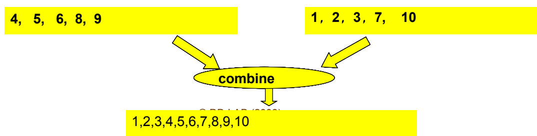
Conquer: 递归求解就是算法的递归调用

Mergesort(A,i,k); //求解第一个子问题

Mergesort(A,k+1,j); //求解第二个子问题

combine: 将两个有序序列合并成一个有序序列

```
1. l←i;    h ←k+1;   t=i                  //设置指针
2. While l≤k &  h< j   Do
      IF  A[l] < A[h]  THEN  B[t] ← A[l]; l ← l+1; t ← t+1;
      ELSE B[t] ← A[h]; h ← h+1; t ← t+1;
3. IF  l<k  THEH                            //第一个子问题有剩余元素
       For  v ← l  To  k  Do
            B[t] ← A[v]; t ← t+1;
4. IF  h<j  THEN                            //第二个子问题有剩余元素
       For  v ← h  To  j  Do
            B[t] ← A[v]; t ← t+1;
```



Mergesort算法

MergeSort(*A,i,j*)

Input: $A[i,\ldots,j]$

Output:排序后的$A[i,\ldots,j]$

```
1.    k ←(i+j)/2;
2.     MergeSort(A,i,k);
3.    MergeSort(A,k+1,j);
4.   l←i;    h ←k+1;   t=i                 //设置指针
5.   While l≤k &  h< j   Do
6.      IF  A[l] < A[h]  THEN  B[t] ← A[l]; l ← l+1; t ← t+1;
7.      ELSE B[t] ← A[h]; h ← h+1; t ← t+1;
8.   IF  l<k  THEH                         //第一个子问题有剩余元素
9.         For  v ← l  To  k  Do
10.              B[t] ← A[v]; t ← t+1;
11.  IF  h<j  THEN                         //第二个子问题有剩余元素
12.        For  v ← h  To  j  Do
13.              B[t] ← A[v]; t ← t+1;
14.  For  v ← i  To  j  Do                 //将归并后的数据复制到A中
15.        A[v] ← B[v];
```

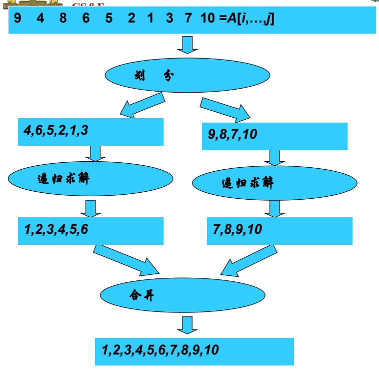
$T(n)=2T(n/2)+O(n)$

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

HIT CS&E

3.9.2 Quick Sort

Partition sort



基本思想

Divide:

确定一个划分标准x，将数组中小于x的元素放到数组的前面部分，大于x的元素放到数组的后面部分，并返回一个划分k;

Conquer:

递归调用算法将A[i,…,k]和A[k+1,…,j]求解。

Combine:

无操作

PartitionSort算法框架

PartitionSort(*A*,*i*,*j*)
Input: *A*[*i*,…,*j*], *x*
Output: 排序后的*A*[*i*,…,*j*]

1. 选择划分元素*x*;
2. *k*=partition(*A*,*i*,*j*,*x*); //用x完成划分
3. partitionSort(*A*,*i*,*k*); //递归求解子问题
4. partitionSort(*A*,*k*+1,*j*);

//无需额外的合并操作

Divide

9 4 8 6 5 2 1 3 7 10 =A[i,…,j]   x=7
9 4 8 6 5 2 1 3 7 10 =A[i,…,j]
3 4 8 6 5 2 1 9 7 10 =A[i,…,j]
3 4 1 6 5 2 8 9 7 10 =A[i,…,j]
3 4 1 6 5 2 8 9 7 10 =A[i,…,j]
3 4 1 6 5 2=A[i,…,high]   8 9 7 10=A[high+1,…,j]

指针low从低区中找出应放到x之后的第一个元素
指针high从高区中找出应放到x之前的第一个元素
如果确实应该交换low和high标记的两个元素的位置，则交换
否则划分位置就是high

Divide过程的算法描述

Partition(*A*,*i*,*j*,*x*)
Input: *A*[*i*,…,*j*], *x*
Output: 划分位置k使得A[i,…,k]中的元素均小于x且A[k+1,…,j]中的元素均大于等于x

1. *low*←*i*; *high* ←*j*;
2. While(*low*< *high*) Do
3. swap(*A*[*low*], *A*[*high*]);
4. While(*A*[*low*] < *x*) Do
5. *low*←*low*+1;
6. While(*A*[*low*] < *x*) Do
7. *high*←*high*-1;
8. return(*high*)

PartitionSort算法

PartitionSort(*A*,*i*,*j*)
Input: *A*[*i*,…,*j*], *x*
Output: 排序后的*A*[*i*,…,*j*]

1. *x*←*A*[*i*]; //以确定的策略选择x
2. *k*=partition(*A*,*i*,*j*,*x*); //用x完成划分
3. partitionSort(*A*,*i*,*k*); //递归求解子问题
4. partitionSort(*A*,*k*+1,*j*);

Partition(*A*,*i*,*j*,*x*)

1. *low*←*i*; *high* ←*j*;
2. While(*low*< *high*) Do
3. swap(*A*[*low*], *A*[*high*]);
4. While(*A*[*low*] < *x*) Do
5. *low*←*low*+1;
6. While(*A*[*low*] < *x*) Do
7. *high*←*high*-1;
8. return(*high*)

时间复杂度分析

- 最好时间复杂度
  - 每次划分都很均匀
  - $T(n)=2T(n/2)+n$
  - $T(1)=1$
  - $T(n)=O(n \log n)$
- 最坏时间复杂度
  - 每次划分均产生1个元素和n-1个元素
  - $T(n)=T(1)+T(n-1)+n$
  - $T(1)=1$
  - $T(n)=O(n^2)$

HIT CS&E

QuickSort算法

```
QuickSort(A,i,j)
Input:   A[i,…,j], x
Output: 排序后的A[i,…,j]
    temp ←rand(i,j);              //产生i，j之间的随机数
    x←A[temp];                    //以确定的策略选择x
    k=partition(A,i,j,x);         //用x完成划分
    QuickSort(A,i,k);             //递归求解子问题
    QuickSort(A,k+1,j);
```

```
Partition(A,i,j,x)
    low←i ;  high ←j;
    While( low< high ) Do
        swap(A[low], A[high]);
        While( A[low] < x )  Do
            low←low+1;
        While( A[low] < x )  Do
            high←high-1;
    return(high)
```


HIT CS&E

算法性能的分析

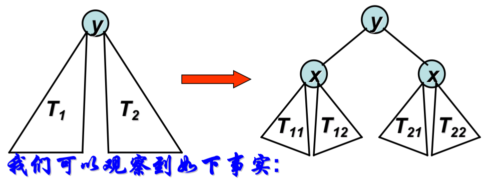
- 基本概念
  - $S_{(i)}$表示S中阶为i的元素
    例如，$S_{(1)}$和$S_{(n)}$分别是最小和最大元素
  - 随机变量X_{ij}定义如下：
    X_{ij}=1如果$S_{(i)}$和$S_{(j)}$在运行中被比较，否则为0
  - X_{ij}是$S_{(i)}$和$S_{(j)}$的比较次数
  - 算法的比较次数为 $\sum_{i=1}^{n}\sum_{j>i} X_{ij}$
  - 算法的平均复杂性为 $E[\sum_{i=1}^{n}\sum_{j>i} X_{ij}]=\sum_{i=1}^{n}\sum_{j>i} E[X_{ij}]$

HIT CS&E

- 计算$E[X_{ij}]$
  - 设p_{ij}为$S_{(i)}$和$S_{(j)}$在运行中比较的概率，则
    $E[X_{ij}]=p_{ij}\times 1+(1-p_{ij})\times 0=p_{ij}$

关键问题成为求解p_{ij}

- 求解P_{ij}
  - 我们可以用树表示算法的计算过程



  - 我们可以观察到如下事实：
    - 一个子树的根必须与其子树的所有节点比较
    - 不同子树中的节点不可能比较
    - 任意两个节点至多比较一次

HIT CS&E

- 当$S_{(i)}$, $S_{(i+1)}$, …, $S_{(j)}$在同一子树时，$S_{(i)}$和$S_{(j)}$才可能比较
- 由随机算法的特点，$S_{(i)}$, $S_{(i+1)}$, …, $S_{(j)}$在同一子树的概率为1
- 只有$S_{(i)}$或$S_{(j)}$被选为划分点时，$S_{(i)}$和$S_{(j)}$才可能比较
- $S_{(i)}$, $S_{(i+1)}$, …, $S_{(j)}$等可能地被选为划分点，所以$S_{(i)}$和$S_{(j)}$
  进行比较的概率是：2/(j-i+1)，即

$p_{ij}=2/(j-i+1)$

HIT CS&E

- 现在我们有

$$\sum_{i=1}^{n}\sum_{j>i} E[X_{ij}]=\sum_{i=1}^{n}\sum_{j>i} p_{ij}=\sum_{i=1}^{n}\sum_{j>i}\frac{2}{j-i+1}$$

$$\le\sum_{i=1}^{n}\sum_{k=1}^{n-i+1}\frac{2}{k}\le 2\sum_{i=1}^{n}\sum_{k=1}^{n}\frac{1}{k}=2nH_n=O(n\log n)$$

定理．随机排序算法的期望时间复杂性为O(nlogn)

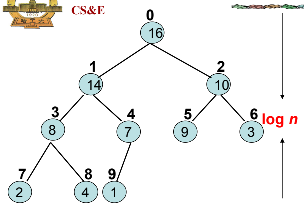
3.9.3 堆排序

堆

堆排序

堆排序算法的时间复杂度分析

堆



•二叉树

•任意节点的值均大(小)于其所有后代

•如果第i号节点缺孩子，则j(>i)无孩子

堆的数组实现

0	1	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1

Len(A) 数组大小

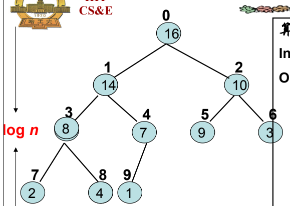
HSize 堆大小

Parent(i) = $\lfloor(i-1)/2\rfloor$

Left(i) $=2i+1$

Right(i) $=2i+2$

堆性质的维护



问题

- 在堆节点i，堆性质不成立
- 如何调整以维护堆性质

算法Max_Heapify(A,i)

Input：堆A,位置i,堆性质在i不成立

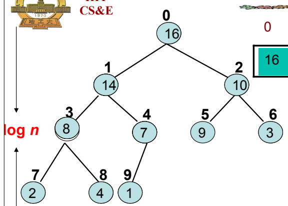
Output:满足堆性质的堆A

1. l←*Left*(i), r←*Right*(i);
2. if l≤*HSize*(A) && A[l]>A[i] then
   max←l;
   else *max*←i;
3. if r≤*HSize*(A) && A[r]>A[*max*]
   max←r;
4. if *max*≠i then
   swap(A,i,max);
   Max_Heapify(A, max);

所需比较操作的次数为i到叶子层的最长路径长度的2倍

时间复杂度为O(log n)

堆的建立



0	1	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1

算法Build_max_heap(A)

Input: 数组A

Output: 堆A

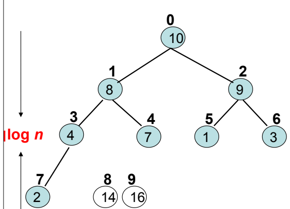
1. HSize←Len(A);
2. For i ←$\lfloor$Len(A)/2$\rfloor$-1 to 0
3. Max_Heapify(A,i);

时间复杂性

$$\sum_{h=0}^{\lfloor\log n\rfloor}\left\lceil\frac{n}{2^{h+1}}\right\rceil 2h \le \sum_{h=0}^{\lfloor\log n\rfloor}\left(\frac{n}{2^{h+1}}+1\right)2h = n\sum_{h=0}^{\lfloor\log n\rfloor}\frac{h}{2^h}+\sum_{h=0}^{\lfloor\log n\rfloor}2h = O(n)$$

堆排序

0	1	2	3	4	5	6	7	8	9
10	8	9	4	7	1	3	2	14	16



算法Heap_sort(A)

Input: 数组A

Output: 排序后的A

1. Build_max_heap(A);
2. For i ←Len(A)-1 to 1
3. Swap(A,1,i);
4. HSize=HSize-1;
5. Max_Heapify(A,1);

时间复杂性 O(n log n)

第1步 O(n)次比较操作

第2步循环 n-1遍 每遍至多log n次比较操作

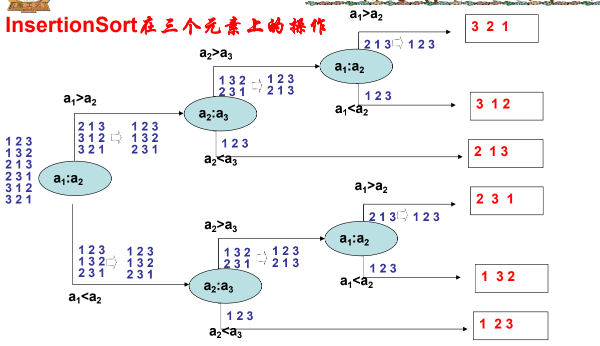
3.9.4 基于比较操作的排序算法的时间复杂度下界

Motivation

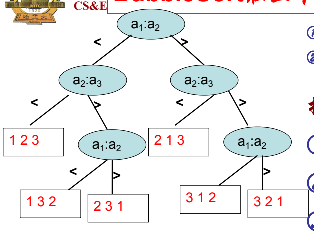
- 排序操作是一个基本操作，也是很多应用中面临的公共问题
- 有些算法(MergeSort, HeapSort, QuickSort)的时间复杂度为 $O(n\log n)$，有的算法(InsertionSort, BubbleSort,SelectionSort)的时间复杂度为$O(n^2)$
- 从渐近的角度讲，$O(n\log n)$是不是最优的排序时间？
- 或者说，能否找到 基于比较的复杂度为$O(n)$的排序算法？
- 多种尝试的失败，促使人们证明不存在基于比较的复杂度更低的排序算法 。

排序算法与决策树

InsertionSort在三个元素上的操作



BubbleSort在三个元素上的决策树



① 略去了元素交换操作

② 仅给出了二叉树节点和边

排序算法与二叉决策树

① 内节点：比较操作

② 叶节点：输入数据的大小关系

③ 边：<>分别标记左、右孩子

④ 算法在特定输入上的运行对应从根到叶的一条路径

⑤ 树的深度对应算法的最坏时间复杂度

⑥ 事实上，任意基于比较的排序算法均对应这样一棵决策树

寻找n个元素的最优排序算法

等价于

寻找$n!$种排列为所有叶节点的最优决策树

关于二叉树，我们知道

① 在叶子数量固定的所有二叉树中，平衡二叉树的深度最小

② 叶子数量为X的平衡二叉树的深度为$\lceil \log X \rceil$

基于比较的排序算法的时间复杂度下界为$\lceil \log n! \rceil$

注意: $\log n! = \Theta(n \log n)$

What lower bound tells us?

- First, it reassures us that widely used sorting algorithms are asymptotically optimal. Thus, one should not needlessly search for an $O(n)$ time algorithms (in the comparison-based class).
- Second, decision tree proof is one of the few non-trivial lower-bound proofs in computer science.
- Finally, knowing a lower bound for sorting also allows us to get lower bounds on other problems. Using a technique called **reduction**, any problem whose solution can indirectly lead to sorting must also have a lower bound of $\Omega(n \log n)$.

1. Straightforward application of decision tree method does not always give the best lower bound.
2. [**Closest Pair Problem:**] How many possible answers (or leaves) are there? At most $\binom{n}{2}$. This only gives a lower bound of $\Omega(\log n)$, which is very weak. Using more sophisticated methods, one can show a lower bound of $\Omega(n \log n)$.
3. [**Searching for a key in a sorted array:**] Number of leaves is $n+1$. Lower bound on the height of the decision tree is $\Omega(\log n)$. Thus, binary search is optimal.

3.10 线性时间排序算法

- 基于比较的排序算法的时间复杂度下界为 O(*n*log*n*)
- 要突破这一下界——不能再基于比较
- 本节介绍三个线性时间排序算法

3.10.1 Counting Sort

- 排序小范围内的整数，线性时间复杂度
- 假设所有输入数据介于 0..*k*之间
- 使用辅助数组 C[0..k]，C[*i*]是原始输入中小于等于*i* ($0 \leqslant i \leqslant k$)数据的个数
- 由C[]和原始输入，可以确定排序结果
- 当$k = O(n)$时,算法复杂度为 $\Theta(n)$.
- Counting sort 是稳定的，它保持相等的关键字值在排序前后的顺序

算法CountingSort(*A*,*B*,*k*)

输入：数组*A*[0:*n*-1]，$0 \le A[i] \le k$

输出：将A[]中数据排序后存入数组B[]

```
for i←0 to k
    C[i] ←0;
for  j←0 to Len(A)-1
    C[A[j]] ←C[A[j]]+1    //C[k]是A中k的个数
for i←1 to k
    C[i] ← C[i] +C[i-1] ; //C[k]是A中≤k的个数
for j←Len(A)-1 to 0
     B[C[A[j]]-1] ← A[j];
     C[A[j]] ←C[A[j]]-1;
```

时间复杂度$O(n+k)=O(n)$ if $k=O(n)$

例

	0	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3	4

	0	1	2	3	4	5
C	2	0	2	3	1	1

⇨

	0	1	2	3	4	5
C	0	2	2	4	7	8

	0	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	4	5

- 为什么不能总用counting sort来完成排序？
  - 因为其复杂性取决于输入元素的范围*k*
- 能用CountSort来排序32位的整数吗？为什么？
  - Answer: no, *k* too large (2^{32} = 4,294,967,296)

3.10.2 Radix Sort

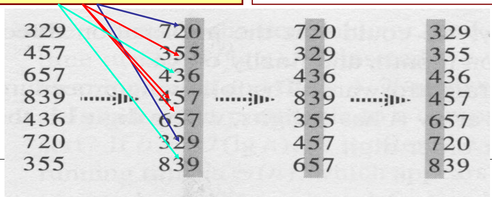
- 每个数字均由一些数值位构成
  - 每个数值位的取值均是有限的.
  - 在每个位上均可用CountingSort排序

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

- 直观上，我们可以先排序最高位，再排序次最高位…
  - Problem:最高位排序后，必须将输入数据依据最高位的取值分组，每组内再按次最高位排序…..,分组太多，太难伺候….
- 关键思想:先排序低位

```
RadixSort(A, d)
  for i=1 to d
      StableSort(A) on digit i
```

排序高位时，须保持低位的序



HIT CS&E

Radix Sort的正确性

- 对d（stableSort执行遍数）做归纳法:
  - d=1时，算法显然正确
  - 假设$d<i$时算法能给出正确的排序
  - 往证$d=i$时，算法能给出正确的排序
    - 如果两个数的第 i位不同,则第i位上大小关系即为这两个数的大小关系（低位的大小无关）
    - 如果两个数的第 i位相同，则这两个数的低位数字已经按大小排序了。由于排序第i位时使用了稳定排序，故排序第i位后这两个数的先后次序即其低位的大小顺序

HIT CS&E

Radix Sort的时间复杂性

- CountingSort在排序 n个界于1..k之间的元素 .
  - 时间开销为: $O(n + k)$
- 对于d位的 n个数（每个位介于1…k之间）
  - RadixSort排序每个位即调用一次CountingSort，其时间开销为$O(n+k)$
  - 因此总时间开销为 $O(dn+dk)$
- 若 d 是常数且 $k=O(n)$, 时间复杂度为$O(n)$

HIT CS&E

用 Radix Sort排序大整数

- Problem: 排序 1000,000个 64-位二进制整数
  - Use 8-bit radix.
  - Each counting sort on 8-bit numbers ranges from 1 to 128.
  - Can be sorted in 64/8=8 passes by counting sort.
  - $O(8 (n + 28))$.

HIT CS&E

- 一般而言,基于CountingSort的基数排序
  - 快
  - 渐进快 (i.e., $O(n)$)
  - 易于编码实现
  - 一个不错的选择
- 能用基数排序来排序浮点数?

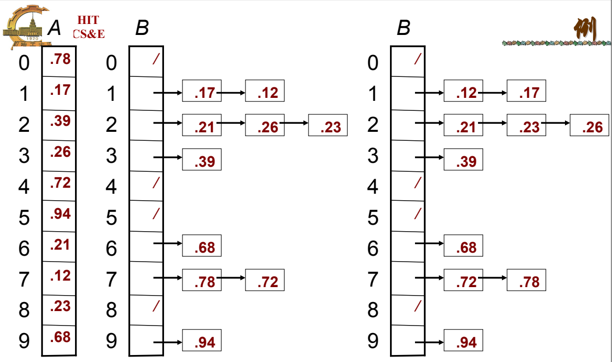
HIT CS&E

3.10.3 Bucket Sort

- 基本思想
  - 假设所有输入值均匀等可能地取自[0,1);
  - 初始化n个空桶，编号介于0到n-1之间;
  - 扫描输入，将数值$A[i]$放入编号为$\lfloor nA[i] \rfloor$的桶中;
  - 将各个桶内的数据各自排序
  - 依编号递增顺序输出各个桶内的数据
- 需要一系列桶,需要排序的值变换为桶的索引
  - 不需要比较操作

HIT CS&E

例



直观上，只要每个桶的数据不多，总时间可以是线性

HIT CS&E

Bucket Sort算法

算法BucketSort(A)

Input: 数组$A[0:n-1]$，$0\le A[i]<1$

Output: 排序后的数组A

1. for $j \leftarrow 0$ to m-1 do   // 初始化 m个桶
2. $B[j] \leftarrow$ NULL;
3. for $i \leftarrow 0$ to n-1 do
4.   将元素$A[i]$插入桶 $B[\lfloor nA[i] \rfloor]$中   //链表维
5. for $i \leftarrow 0$ to n-1 do
6.   用InsertionSort排序桶$B[i]$内的数据
7. 依编号递增顺序将各个桶内的数据回填到A中

HIT CS&E

时间复杂度分析

InsertionSort的时间复杂度为$O(n^2)$

$T(n)=\Theta(n)+\sum_{i=0,\ldots,n-1}O(n_i^2)$

n_i是落入$B[i]$中的数据的个数

对于含有n个数据的输入

- 具体的实例不一样，n_i的取值就不一样
- 如果将输入看成随机的，则n_i也是随机的
- $T(n)$也是随机的

由数学期望的线性性质和函数阶的性质得到

$E[T(n)]=\Theta(n)+\sum_{i=0,\ldots,n-1}O(E[n_i^2])$

–下面证明$E[n_i^2]$=2-1/n ,从而$E[T(n)]=\Theta(n)$

对于任意的i

令 X_{ij}=1   如果算法运行时，$A[j]$落入$B[i]$

X_{ij}=0   如果算法运行时，$A[j]$未落入$B[i]$

显然，由于$A[j]$均匀取值于[0,1), X_{ij}是随机变量

由桶的划分方式，知道 $P_r(X_{ij}=1) = 1/n$ $P_r(X_{ij}=0) = 1-1/n$

$E(X^2_{ij})=1/n$ $E(X_{ij}X_{ik})=1/n^2$ $(j\neq k)$

$$E(n_i^2) = E[(\sum_{j=0}^{n-1} X_{ij})^2]$$

$$= \sum_{j=0}^{n-1} E(X_{ij}^2) + \sum_{j=0}^{n-1}\sum_{\substack{0\le k\le n-1\\ k\ne j}} E(X_{ij}X_{ik})$$

$$= 1 + \frac{n-1}{n} = 2 - \frac{1}{n}$$