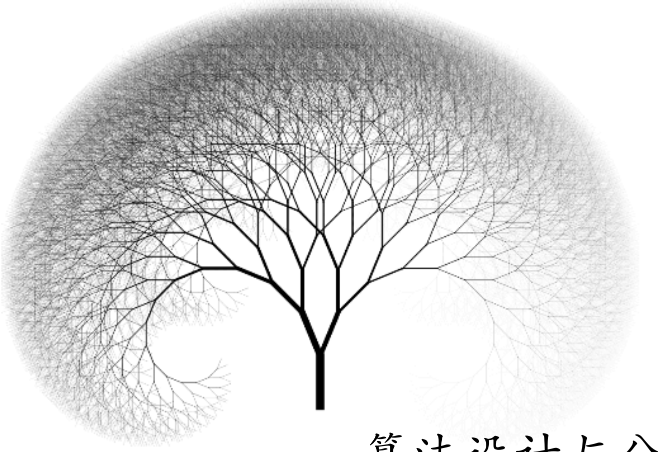


算法设计与分析

刘 安
苏州大学 计算机科学与技术学院
http://web.suda.edu.cn/anliu/

分治算法

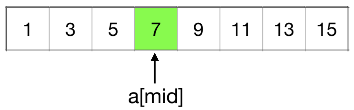
- 分解：将原问题分解为一组子问题，子问题与原问题类似，但是规模更小
- 解决：递归求解子问题，如果子问题足够小，停止递归，直接求解
- 合并：将子问题的解组合成原问题的解
- 回顾最大子数组问题
  - 分解：将原数组分解为两个大小相等的数组Left和Right
  - 解决：递归求解数组Left和Right的最大子数组
    - 如果数组只有一个元素，停止递归，直接求解
  - 合并：
    - $OPT(a[1..n]) = \max\{OPT(Left), OPT(Right), S(Cross)\}$
    - 计算跨越数组Left和Right的最大子数组

二分搜索

Binary Search

问题

- 输入：一个已排序数组$a[1..n]$（元素各不相同），一个元素x
- 输出：如果$x = a[j]$，返回j，否则返回-1
- 顺序搜索：$O(n)$
- 二分搜索
  - 分解：数组Left，中间元素$a[mid]$，数组Right
  - 解决：如果$a[mid] = x$，返回mid，否则递归求解数组Left或者数组Right
    - 如果数组为空，停止递归，直接求解（元素不存在）
  - 合并：不需要额外工作



时间复杂度分析

- 运行时间：$T(n) \le T(\lfloor n/2 \rfloor) + O(1)$，$T(1) = 1$
- 上式可以安全地简化为$T(n) \le T(n/2) + 1$
- $T(n) \le T(n/2) + 1 \le T(n/4) + 2 \le \cdots \le T(n/2^k) + k$
  - 令$n = 2^k$，那么$T(n) \le T(1) + \log n = 1 + \log n$
- $T(n) = O(\log n)$

```
int binary_search(const vector<int>& a, int x, int low, int high)
{
    if (low > high) return -1;
    int mid = (low + high) / 2;
    if (a[mid] == x) return mid;
    if (a[mid] > x)
        return binary_search(a, x, low, mid - 1);
    else
        return binary_search(a, x, mid + 1, high);
}
```


正确性分析

- 命题：如果$x \in a[low..high]$，算法返回j，其中$x = a[j]$, $low \le j \le high$，否则返回-1
- 对数组a的长度$n = high - low + 1$进行归纳
- 基本情况：$n = 0$
  - $low = high + 1$，此时算法返回-1，显然正确（空数组不包含x）
- 归纳假设：假设对所有长度小于$k \ge 1$的a的子数组，命题正确
- 归纳步骤：证明对长度为k的数组，命题正确
  - $a[mid] = x$：算法返回mid，显然正确
  - $a[mid] < x$：因为a有序，所以$x \in a[mid+1..high]$，根据归纳假设，子问题能够正确求解，所以命题正确
  - $a[mid] > x$：因为a有序，所以$x \in a[low..mid-1]$，根据归纳假设，子问题能够正确求解，所以命题正确

计数

Counting Occurrences

问题

- 输入：一个已排序数组$a[1..n]$，一个元素x
- 输出：元素x的出现次数
- 直接使用二分搜索
  - 通过二分搜索可以在$O(\log n)$时间内找到元素x所在的块
  - 然后向左（右）扫描找到块的左（右）边界
  - 时间复杂度：$O(\log n + s)$，其中s是块的长度
    - 最坏情况下是$\Theta(n)$
- 能否做的更好？比如$O(\log n)$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	3	3	3	3	3	5	5	7	9	11	11	11	13	15	15

改进二分搜索

- 关键：找到块的右边界（和左边界）
- 分解：将$a[l..h]$分成$a[l..m-1]$，$a[m]$和$a[m+1..h]$
- 解决：
  - 基本情况：如果$l > h$，返回-1
  - 如果$a[m] = x$，并且$m = n$或者$a[m+1] > x$，返回m
  - 如果$a[m] > x$，那么递归求解数组Left，否则，递归求解数组Right
    - $a[m] < x$，或者$a[m] = x$ 且（$m \neq n$ 且 $a[m+1] = x$）
- 合并：不需要额外工作

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	3	3	3	3	3	5	5	7	9	11	11	11	13	15	15

正确性分析

- 命题：如果$x \in a[l..h]$，那么算法返回最后一个x所在的位置，否则返回-1
- 对数组a的长度$n = h - l + 1$进行归纳
- 基本情况：$n = 0$
  - $l = h + 1$，此时算法返回-1，显然正确（空数组不包含x）
- 归纳假设：假设对所有长度小于$k \geq 1$的a的子数组，命题正确
- 归纳步骤：证明对长度为k的数组，命题正确
  - $a[m] = x \wedge (m = n \vee a[m+1] > x)$：算法返回$m$，显然正确
  - $a[m] > x$：因为a有序，所以如果$x \in a[l..h]$，那么$x \in a[l..m-1]$，根据归纳假设，子问题能够正确求解（同时正确求解$x \notin a[l..h]$的情况）
  - 否则
    - $a[m] < x$：证明与$a[m] > x$情况类似
    - $a[m] = x \wedge m \neq n \wedge a[m+1] = x$：显然有$x \in a[m+1..h]$，根据归纳假设，该子问题可以正确求解

计数算法

- 时间复杂度：$O(\log n)$
- 正确性：基于算法first和算法last的正确性易证
  - first：找到左边界
  - last：找到右边界

```
int count(const vector<int>& a, int x)
{
    int i = first(a, x, 0, a.size() - 1, a.size());
    if (i == -1) return -1;
    int j = last(a, x, i, a.size() - 1, a.size());
    return j - i + 1;
}
```

* C++中lower_bound和upper_bound函数的实现

选择

Selection

问题

- 寻找中位数
  - 输入：数组$a[1..n]$
  - 输出：$a[1],\cdots,a[n]$的中位数
- 使用归纳求解问题时，增强归纳假设有时候反而更有效
- 选择
  - 输入：数组$a[1..n]$，整数k
  - 输出：a中第k小的数
- 基于排序的算法：$O(n\log n)$

3	36	5	21	8	13	11	20	5	4	1

1	3	4	5	5	8	11	13	20	21	36

递归关系

- 回顾快速排序
  - 划分：选择一个主元v，然后重新排列数组
    - L：小于v的元素；M：等于v的元素；R：大于v的元素
- $k=4$：第4小的数必然是5（$3<k<3+2$）
- $k=1$：第1小的数必然在L中，且必然是L中第1小的数（$k\le 3$）
- $k=8$：第8小的数必然在R中，且必然是R中第3小的数（$k>3+2$）
- $selection(a,k)=\begin{cases} selection(L,k) & \text{if } k\le|L| \\ v & \text{if } |L|<k\le|L|+|M| \\ selection(R,k-|L|-|M|) & \text{if } k>|L|+|M| \end{cases}$

3	36	5	21	8	13	11	20	5	4	1

$v=5$

3	4	1	5	5	36	21	8	13	11	20

L M R

递归求解的时间复杂度

- 划分：时间复杂度$O(n)$，空间复杂度$O(1)$
- 如果能够选择一个主元v，使得$|L|=|R|$，那么
  - $T(n)=T(n/2)+n$
  - $\Rightarrow T(n)=O(n)$
- 随机选择一个元素作为主元
  - 最好情况：$O(n)$
  - 最坏情况：$T(n)=T(n-1)+n$
    - $\Rightarrow T(n)=O(n^2)$
  - 平均情况：$O(n)$（证明参考算法导论第三版9.2节）
- 有没有别的方法来选择主元，使得最坏情况下$O(n)$呢？

$v=5$

3	4	1	5	5	36	21	8	13	11	20

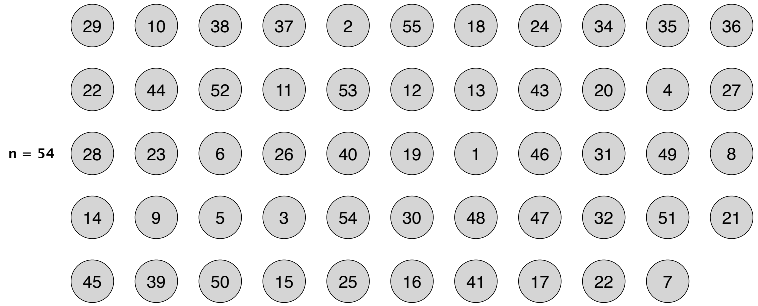
L M R

最坏情况为线性时间的选择算法

- 关键：找到一个合适的主元，使得$|L|$和$|R|$一定小于等于$\frac{7}{10}n$
  - 每次搜索范围最多是上一次的70%
- 挑战：在线性时间内找到这个主元
  - 聪明地选取$\frac{2}{10}n$个元素，然后递归地求解这些元素的中位数
- $T(n)\le T(\frac{7}{10}n)+T(\frac{2}{10}n)+O(n)$
  - 可以证明$T(n)=\Theta(n)$

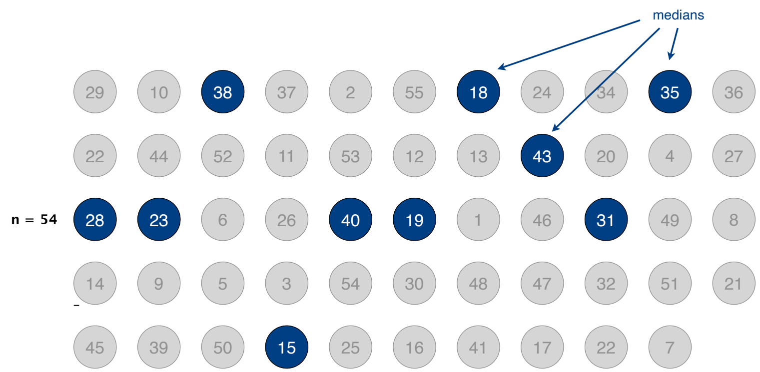
主元选择

- 将n个元素分成$\lfloor n/5 \rfloor$组，每组5个元组，可能还有额外的一组，由剩下的$n\%5$个元素组成



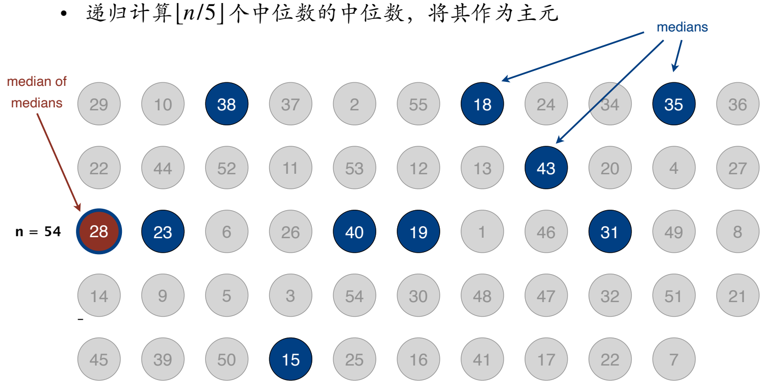
主元选择

- 将n个元素分成$\lfloor n/5 \rfloor$组，每组5个元组，可能还有额外的一组，由剩下的$n\%5$个元素组成
- 直接计算每一组的中位数（额外那一组除外）



主元选择

- 将n个元素分成$\lfloor n/5 \rfloor$组，每组5个元组，可能还有额外的一组，由剩下的$n\%5$个元素组成
- 直接计算每一组的中位数（额外那一组除外）
- 递归计算$\lfloor n/5 \rfloor$个中位数的中位数，将其作为主元



基于中位数的中位数的选择算法

Mom-Select($a[i..j], k$)

if ($j - i + 1 < 50$)

return k^{th} smallest element of $a[i..j]$ via merge sort or brute force

Group $a[i..j]$ into $\lfloor n/5 \rfloor$ groups of 5 elements each (ignore leftovers)

$B \leftarrow$ median of each group of 5

$v \leftarrow$ Mom-Select($B, \lfloor n/10 \rfloor$)

$(L, M, R) \leftarrow$ Partition($a[i..j], v$)

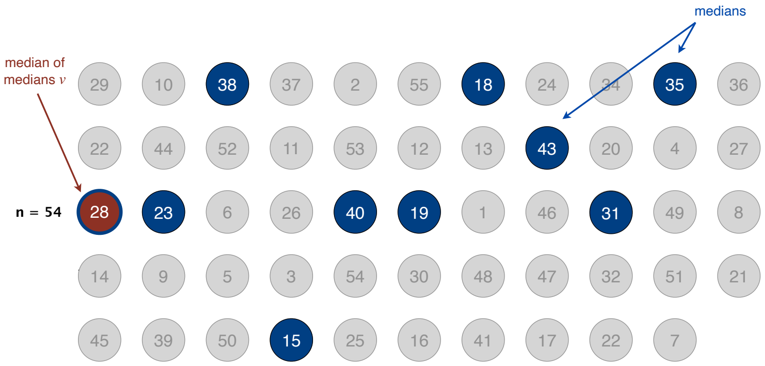
if ($k \leqslant |L|$) return Mom-Select(L, k)

else if ($k > |L| + |M|$) return Mom-Select($R, k - |L| - |M|$)

else return v

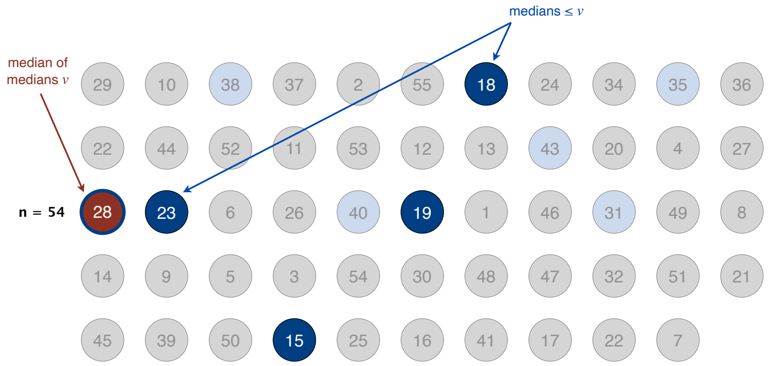
Mom-Select的运行时间

- 至少有一半中位数小于等于主元v



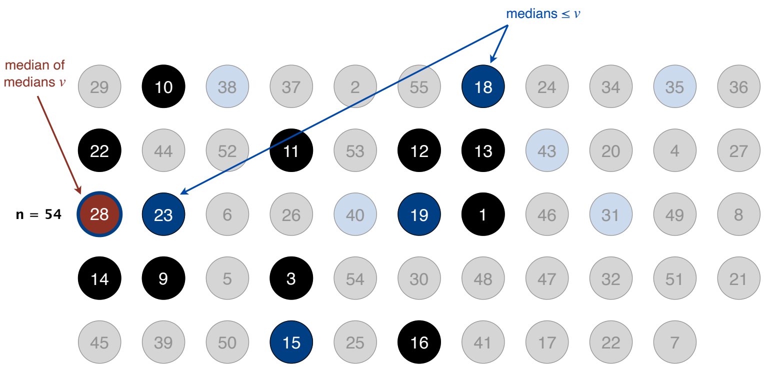
Mom-Select的运行时间

- 至少有一半中位数小于等于主元v
- 至少有$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$个中位数小于等于主元$v$



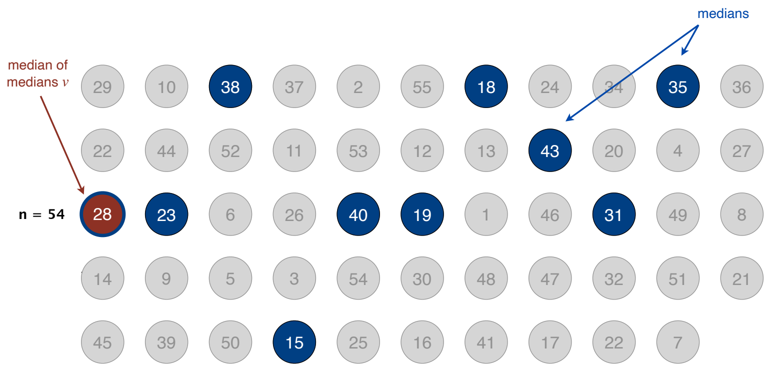
Mom-Select的运行时间

- 至少有一半中位数小于等于主元v
- 至少有$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$个中位数小于等于主元$v$
- 至少有$3\lfloor n/10 \rfloor$个元素小于等于主元v



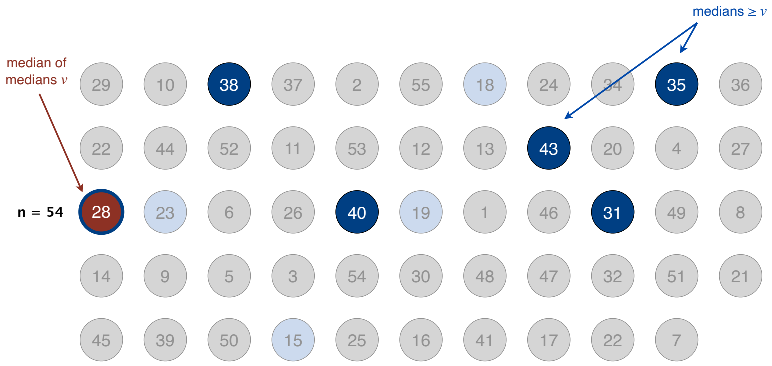
Mom-Select的运行时间

- 至少有一半中位数大于等于主元v



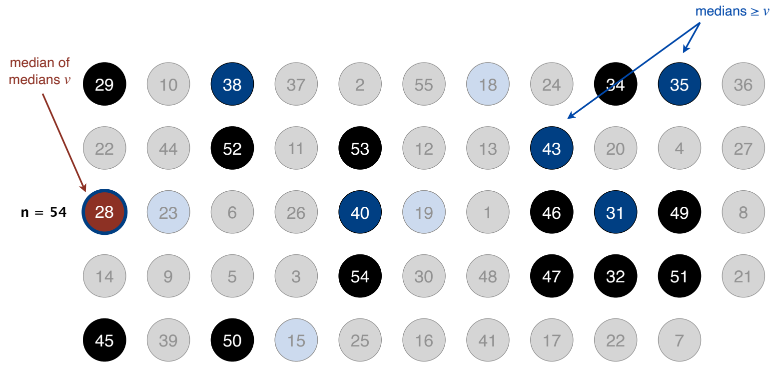
Mom-Select的运行时间

- 至少有一半中位数大于等于主元v
- 至少有$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$个中位数大于等于主元$v$



Mom-Select的运行时间

- 至少有一半中位数大于等于主元v
- 至少有$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$个中位数大于等于主元$v$
- 至少有$3\lfloor n/10 \rfloor$个元素大于等于主元v



Mom-Select的运行时间的递归式

- 从$\lfloor n/5 \rfloor$个中位数中递归地选择中位数，即主元v
- 至少有$3\lfloor n/10 \rfloor$个元素小于等于主元v
- 至少有$3\lfloor n/10 \rfloor$个元素大于等于主元v
- 从最多$n - 3\lfloor n/10 \rfloor$个元素中递归地选择
- 令$T(n)$是Mom-Select从n个元素中选择第k小元素所需的比较次数
  - $T(n) \le T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5 \cdot n$
    - 递归计算中位数的中位数
    - 递归选择
    - 划分：比较次数$\le n$
    - 计算5个元素的中位数：比较次数≤ 6

Mom-Select的运行时间

- $T(n) \le \begin{cases} 6n & \text{if } n < 50 \\ T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5 \cdot n & \text{if } n \ge 50 \end{cases}$
- 猜测：$T(n) \le 44n$
- 数学归纳法证明
  - 基本情况：当$n < 50$时使用归并排序，所以$T(n) \le 6n$
  - 当$n \ge 50$时，

$$\begin{aligned} T(n) &\le T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5 \cdot n \\ &\le 44(\lfloor n/5 \rfloor) + 44(n - 3\lfloor n/10 \rfloor) + 11/5 \cdot n \\ &\le 44(n/5) + 44n - 44(n/4) + 11/5 \cdot n \\ &= 44n \end{aligned}$$

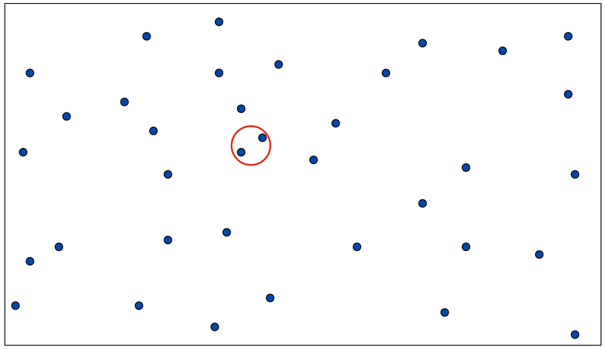
当$n \ge 50$时，$3\lfloor n/10 \rfloor \ge n/4$

最近点对

Closest Pair

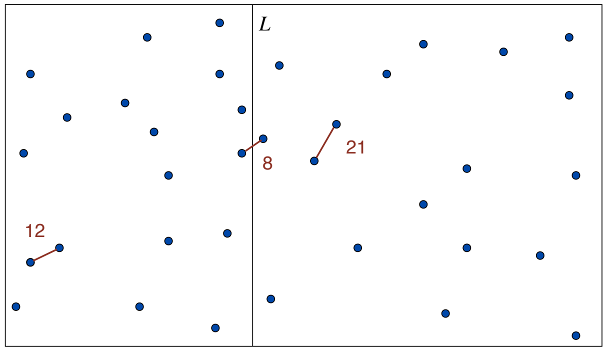
问题

- 给定二维平面上的n个点，找到距离最近的一对点
- 穷举法：$\Theta(n^2)$
- 一维情况：数轴上的n个点
  - 排序 + 扫描：$O(n \log n)$



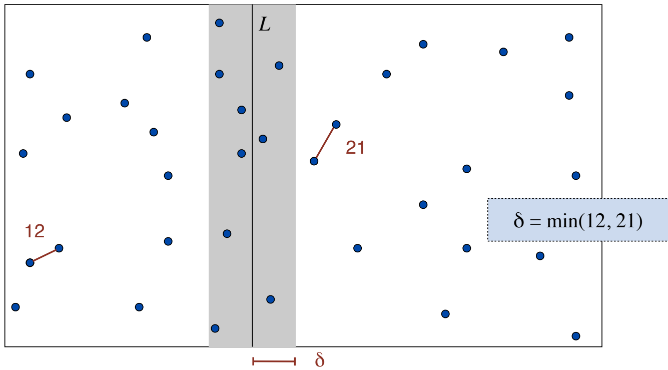
分治法求最近点对

- 分解：构造垂线L使得L的两侧各有$n/2$个点
- 解决：递归地求解每侧的最近点对
- 合并：找到最近的特殊点对，即一个点在L的左侧，另一个点在L的右侧
- 返回三者中的最近点对



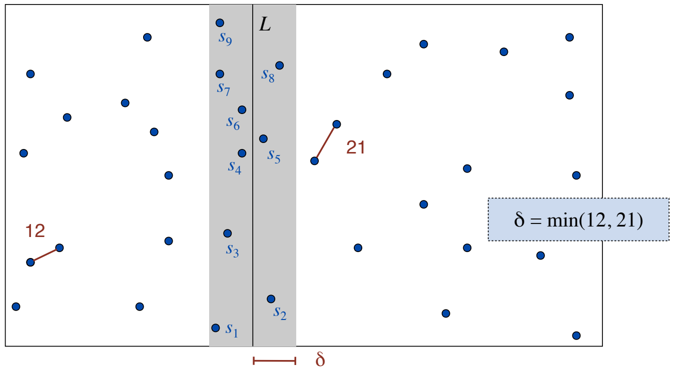
如何找到特殊的最近点对

- 假设L左侧的最近点对的距离是δ_1，L右侧的最近点对的距离是δ_2，令$\delta = \min\{\delta_1, \delta_2\}$
- 仅仅需要考虑距离L小于δ的点



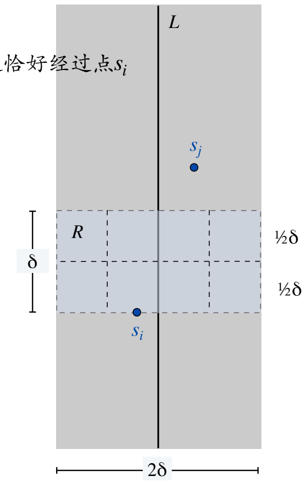
如何找到特殊的最近点对

- 将宽为2δ的带状区域内的点根据y坐标从小到大排序
- 对于每个点，只需计算它与7个点之间的距离
- $7n$次计算就可以找到特殊的最近点对



如何找到特殊的最近点对

- 令s_i是带状区域中y坐标第i小的点
- 如果$|j-i|>7$，那么$d(s_i,s_j)\geq\delta$
  - 考虑带状区域中一个$2\delta\times\delta$的矩形R，其下底边恰好经过点s_i
  - 令s_j是任意位于R上方的点，显然$d(s_i,s_j)\geq\delta$
  - 将矩形R分成8个小正方形
  - 每个小正方形中最多只有一个点
    - 小正方形的对角线长为$\delta/\sqrt{2}<\delta$
    - L两侧的最近点对的距离是δ
  - 除了s_i，R中最多只有7个点



最近点对的分治算法

Closest-Pair($p_1,\cdots,p_n$)

构造分割线将$p_1,\cdots,p_n$分成L和R两个部分，每部分均有$n/2$个点 $//O(n)$

$\delta_1\leftarrow$ Closest_Pair(L) $//T(n/2)$

$\delta_2\leftarrow$ Closest_Pair(R) $//T(n/2)$

$\delta\leftarrow\min\{\delta_1,\delta_2\}$

$A\leftarrow$ 所有距离分割线小于δ的点 $//O(n)$

将A中的点按照纵坐标排序 $//O(n\log n)$

对于A中每个点 $//O(n)$

计算其与后续7个点的距离，如果某个距离小于δ，更新δ

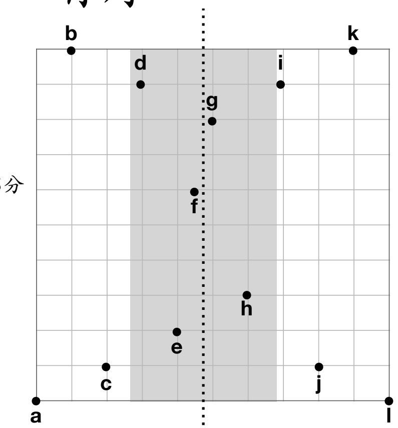
返回δ

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

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

保持输入有序

- 输入
  - P_x：n个点，按照横坐标排序
  - P_y：n个点，按照纵坐标排序
- 分解：将n个点分成L和R左右两个部分
- 变量
  - L_x：L中的点，按照横坐标排序
  - L_y：L中的点，按照纵坐标排序
  - R_x：R中的点，按照横坐标排序
  - R_y：R中的点，按照纵坐标排序
  - A_y：与垂线距离小于δ的点，按照纵坐标排序



$A_y=[e,h,f,g,d]$

$P_x=[a,b,c,d,e,f,g,h,i,j,k,l]$ $L_x=[a,b,c,d,e,f]$ $R_x=[g,h,i,j,k,l]$

$P_y=[a,l,c,j,e,h,f,g,d,i,b,k]$ $L_y=[a,c,e,f,d,b]$ $R_y=[l,j,h,g,i,k]$

最近点对的分治算法-优化版

Closest_Pair(P_x, P_y)

将P_x左半部分的点加入L_x，右半部分的点加入R_x $//O(n)$

对于P_y中的每个点，根据其横坐标将其加入L_y或者R_y $//O(n)$

$(l_1, l_2) \leftarrow$ Closest_Pair(L_x, L_y) $//T(n/2)$

$(r_1, r_2) \leftarrow$ Closest_Pair(R_x, R_y) $//T(n/2)$

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

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

$\delta \leftarrow \min\{d(l_1, l_2), d(r_1, r_2)\}$

对于P_y中的每个点，如果其距离分割线的距离小于δ，将其加入A_y $//O(n)$

对于A_y中的每个点 $//O(n)$

计算其与后续7个点的距离，如果某个距离小于δ，更新δ和最近点对

返回最近点对