南京大学
NANJING UNIVERSITY

算法设计与分析基础《Introduction to the Design and Analysis of Algorithms》

分治法

南京大学软件学院
李传艺
lcy@nju.edu.cn
费彝民楼917

目录

- 分治法回顾
  - 概念
  - 合并排序
  - 快速排序
- 折半查找
- 二叉树遍历及其相关特性
- 大整数乘法和Strassen矩阵乘法
- 分治解最近对问题
- 分治解凸包问题

分治法回顾（1）

- 最著名的通用算法设计技术
  - 很多有效的算法是分治算法的特殊实现：map-reduce、分布式数据库
- 具体方案流程
  - 将问题实例划分为同一个问题的几个较小实例，最好拥有相同规模
  - 对每一个较小规模的实例进行求解
  - 如果需要则以某种方式合并这些小问题的解得到原问题的解
- 通用分治递推式
  - $T_s(n) = a * T\left(\frac{n}{b}\right) + f(n)$
  - 如果$f(n) \in \Theta(n^d)$，其中d≥ 0，则(对O、Ω同样成立)：
  - $$T(n) \in \begin{cases} \Theta(n^d) & \text{当} a < b^d \\ \Theta(n^d log_b n) & \text{当} a = b^d \\ \Theta(n^{log_b a}) & \text{当} a > b^d \end{cases}$$

设 $a \geq 1$ 和 $b > 1$ 为常数，设 f(n)为一函数，T(n)由递归式

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

其中$\frac{n}{b}$指$\left\lfloor\frac{n}{b}\right\rfloor$和$\left\lceil\frac{n}{b}\right\rceil$，可以证明，略去上下去整不会对结果造成影响。那么 T(n)可能有如下的渐进界

(1)若 $f(n) < n^{\log_b^a}$，且是多项式的小于。即

$\exists\, \varepsilon > 0$，有 $f(n) = O\left(n^{\log_b^a - \varepsilon}\right)$，则 $T(n) = \Theta\left(n^{\log_b^a}\right)$

(2)若 $f(n) = n^{\log_b^a}$，则 $T(n) = \Theta\left(n^{\log_b^a} \log n\right)$

(3)若 $f(n) > n^{\log_b^a}$，且是多项式的大于。即

$\exists\, \varepsilon > 0$，有 $f(n) = \Omega\left(n^{\log_b^a + \varepsilon}\right)$，且对$\forall\, c < 1$ 与所有足够大的 n，有 $af\left(\frac{n}{b}\right) \leq cf(n)$，则 $T(n) = \Theta\left(f(n)\right)$

分治法回顾（2）

- 合并排序
  - 将列表分为两个大小最接近的部分
  - 递归地拆分，拆分时不保证顺序正确
  - 返回上一层递归时将每两个小部分合并为有序的部分
  - 继续返回给上一层递归，直到合并为完整列表

  - 算法复杂度：最坏情况$\Theta(nlog_2n)$——什么是最坏情况？

- 快速排序
  - 将列表参照某一个元素的值分为两个部分
  - 每个部分整体之间的位置就确定了
  - 递归拆分每一个小部分
  - 拆分完成后每一个部分内部顺序是正确的，且所有小部分之间的位置也是确定的
  - 不需要返回上一层递归进行合并

  - 算法复杂度：最好情况是$\Theta(nlog_2n)$——什么是最好情况？最坏情况是$\Theta(n^2)$

螺母和螺钉问题

- N个直径各不相同的螺母和N个直径各不相同的螺钉，但是螺钉和螺母是一一对应的
  - 如何快速找到它们的对应关系？
    - 螺母排序
    - 螺钉排序
  - 如果不可以使用螺母与螺母比较，也不能使用螺钉和螺钉比较，如何找到对应关系？

折半查找（1）

- 顺序查找
  - 遍历列表，比较键值与列表元素的大小；直到找到或者列表结束

- 折半查找
  - 对于有序列表的卓越查找方案
  - 通过对比查找键值与列表中间元素的大小，确定下一次查找的位置

$$A[0] \ldots\ldots A[m-1] \qquad K \updownarrow A[m] \qquad A[m+1] \ldots\ldots A[n-1]$$

如果K<A[m]，查这里　　如果K>A[m]，查这里

- 例K=70

下标	0	1	2	3	4	5	6	7	8	9	10	11	12
值	3	14	27	31	39	42	55	70	74	81	85	93	98
迭代1	l						m						r
迭代2								l		m			r
迭代3								lm	r				

折半查找（2）

- 递归方案

```
Algorithm   BinarySearch(A[l,…,r],K)
            //折半查找的递归方案
            m ← ⌊(l+r)/2⌋
            if   K == A[m]
                    return  m
            else if K < A[m]
                    return BinarySearch(A[l,…,m-1],K)
            else return BinarySearch(A[m+1,…,r],K)
```

- 非递归方案
  - 使用临时变量维护搜索范围**头尾**
  - 使用**循环**替代递归
  - 改返回为**更新**头尾

- 算法复杂度
  - 计算比较的次数：三路比较，通过一次比较得到是大于、等于还是小于
  - 最坏情况和最好情况是什么？

折半查找（3）

- 最坏情况
  - 查找列表中不存在的键值
  - 存在于列表中的某些键值：特点是什么？

0	1	2	3	4	5	6	7	8	9	10	11	12
3	14	27	31	39	42	55	70	74	81	85	93	98
	4		4		4			4		4		4
3		2		3		1	3		2		3	

  - 当n>1时，有比较次数的递推关系$C_{worst}(n) = C_{worst}(\lfloor n/2 \rfloor) + 1$，$C_{worst}(1) = 1$
  - $C_{worst}(n) = \lfloor log_2 n \rfloor + 1 = \lceil log_2(n+1) \rceil$

为什么要下取整？

- 最好情况

- 平均情况
  - 能够查找到：每一个位置的概率是相同的 → 约$log_2 n$ - 1
  - 查不到：最多次的查找 → 约$log_2(n+1)$

二叉树中的分治思想

- 定义
  - 要么为空，要么由一个根和两棵称为T_L和T_R的不相交二叉树构成
    - 根，左子树，右子树
  - 扩展
    - 完全二叉树：如果有h层，1~h-1层布满节点，h层节点从左向右分布
    - 满二叉树：除了叶子节点其它节点都有左右子节点，且叶子节点都在最底层
    - 二叉查找树：排序好二叉树，如果左子树不空，则左子树值小于根节点值；如果右子树不空，则右子树值都大于根节点；左右子树分别是二叉查找树。
    - 平衡二叉树：是一棵左右子树高度差绝对值不大于1的二叉查找树

- 二叉树和分治思想
  - 对二叉树的操作变为对左右子树分别操作
    - 计算二叉树高度
    - 二叉树遍历（查找、插入、删除等）
  - 不是所有二叉树的操作都体现分治思想
    - 二叉查找树的查找、插入和删除等
    - 类似折半查找

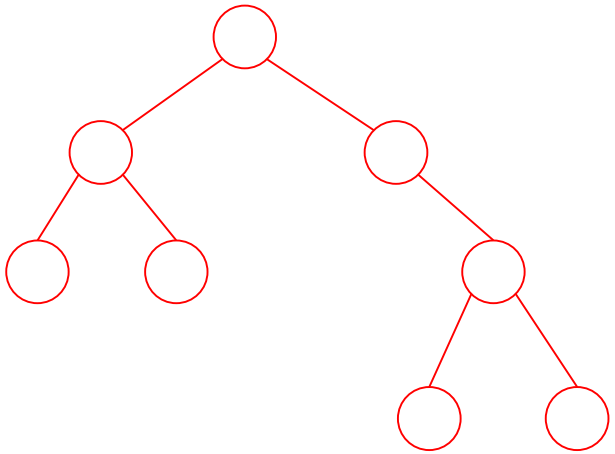
计算二叉树高度

- 递归算法（空树高度是-1）

```
Algorithm   Height (T)
            //递归计算二叉树高度
            if T == ∅
                return -1
            else
                return max{Height(T_L), Height(T_R)} + 1
```

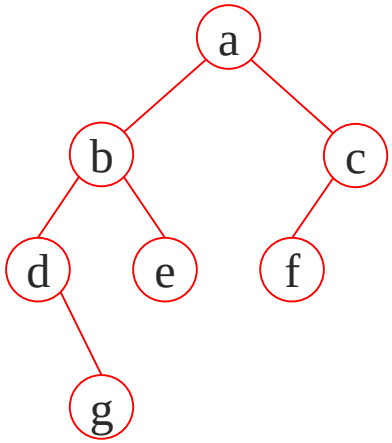
- 算法复杂度
  - 二叉树节点数规模表示为n(T)
  - 判断为空的次数: C(n(T)) 2n+1
  - Max比较的次数: A(n(T)) n
  - 加法的次数: A(n(T))

空树要判断一次
但是不用比较和加法



二叉树遍历

- 前序遍历
  - 根→左子树→右子树
  - a, b, d, g, e, c, f
- 中序遍历
  - 左子树→根→右子树
  - d, g, b, e, a, f, c
- 后序遍历
  - 左子树→右子树→根
  - g, d, e, b, f, c, a



二叉树节点间最大距离

- 可能的情况
  - 左子树中节点间最大距离
  - 右子树中节点间最大剧烈
  - 左子树中节点到右子树中节点的最大距离

大整数乘法（1）

- 需求和算法
  - 密码技术中，对超过100位的十进制整数进行乘法运算
  - 不能使用整型表示
  - 可以使用笔算的逻辑计算——类似大整数加法——乘法次数为n^2

- 是否存在复杂度小于n^2的算法？
- 乘法结果与“数位上的数字之间的关系”？

- 例23*14

$$\begin{array}{r} 23 \\ \times\ 14 \\ \hline 92 \\ 23 \\ \hline =322 \end{array}$$

共2×2次乘法
4次

$23=2\times10^1+3\times10^0$ 且 $14=1\times10^1+4\times10^0$

相乘：$23\times14=(2\times10^1+3\times10^0)\times(1\times10^1+4\times10^0)$

$=(2\times1)\times10^2+(2\times4+3\times1)\times10^1+(3\times4)\times10^0$

4次乘法

$=(2+3)\times(4+1)-(2\times1)-(3\times4)$

3次乘法、2次加法、2次减法.

大数乘法（2）

- 对于任意两个两位整数的例子

$$a = a_1a_0, b = b_1b_0$$

$$c = a \times b = c_2 10^2 + c_1 10^1 + c_0$$

$$\text{则} c_2 = a_1 \times b_1, c_0 = a_0 \times b_0$$

$$c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$$

- 如果是n位整数？
  - 分治思想：拆分为两个位数相同或接近的大整数
  - 假设n为偶数，$a = a_1a_0$其中$a = a_1 \times 10^{\frac{n}{2}} + a_0$，$b = b_1b_0$其中$b = b_1 \times 10^{\frac{n}{2}} + b_0$
  - 相同的结论
  - $c_2 = a_1 \times b_1, c_0 = a_0 \times b_0$，$c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$

$a_{11} \times 10^{\frac{n}{4}} + a_{12}$　　$a_{01} \times 10^{\frac{n}{4}} + a_{02}$

$b_{11} \times 10^{\frac{n}{4}} + b_{12}$　　$b_{01} \times 10^{\frac{n}{4}} + b_{02}$

大整数乘法（3）

- 算法复杂度：乘法次数M(n)
  - 假设$n = 2^k$
  - $c_2 = a_1 \times b_1, c_0 = a_0 \times b_0$，$c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$三次乘法
  - 当n>1，$M(n) = 3M\left(\frac{n}{2}\right)$，$M(1) = 1$
  - 通用分治递推式？
  - $a = 3, b = 2, d = ?$
  - $M(n) = 3M\left(\frac{n}{2}\right) = 3M\left(2^{k-1}\right) = 3 * 3M\left(2^{k-2}\right) = \cdots = 3^k M(1) = 3^{log_2 n} = n^{log_2 3}$约$n^{1.58}$
- 从大于600位的整数开始，分治算法的性能超越笔算算法的性能
- 问题：如果在分治过程中出现奇数位怎么办？

Strassen矩阵乘法（1）

- 两个两阶矩阵的乘积计算需要8次乘法

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} a_{00}\times b_{00}+a_{01}\times b_{10} & a_{00}\times b_{01}+a_{01}\times b_{11} \\ a_{10}\times b_{00}+a_{11}\times b_{10} & a_{10}\times b_{01}+a_{11}\times b_{11} \end{bmatrix}$$

8次乘法

4次加法

- Strassen发现计算矩阵乘法可以只使用7次乘法

$$m_1=(a_{00}+a_{11})\times(b_{00}+b_{11})=a_{00}\times b_{00}+a_{00}\times b_{11}+a_{11}\times b_{00}+a_{11}\times b_{11}$$
$$m_2=(a_{10}+a_{11})\times b_{00}=a_{10}\times b_{00}+a_{11}\times b_{00}$$
$$m_3=a_{00}\times(b_{01}-b_{11})=a_{00}\times b_{01}-a_{00}\times b_{11}$$
$$m_4=a_{11}\times(b_{10}-b_{00})=a_{11}\times b_{10}-a_{11}\times b_{00}$$
$$m_5=(a_{00}+a_{01})\times b_{11}=a_{00}\times b_{11}+a_{01}\times b_{11}$$
$$m_6=(a_{10}-a_{00})\times(b_{00}+b_{01})=a_{10}\times b_{00}+a_{10}\times b_{01}-a_{00}\times b_{00}-a_{00}\times b_{01}$$
$$m_7=(a_{01}-a_{11})\times(b_{10}+b_{11})=a_{01}\times b_{10}+a_{01}\times b_{11}-a_{11}\times b_{10}-a_{11}\times b_{11}$$

$$\begin{bmatrix} m_1+m_4-m_5+m_7 & m_3+m_5 \\ m_2+m_4 & m_1+m_3-m_2+m_6 \end{bmatrix}$$

7次乘法

18次加减法

Strassen矩阵乘法（2）

- 其重要性体现在矩阵规模趋近无穷大时
  - 假设$n = 2^k$，如果不满足，则补0
  - 第一次把两个矩阵分别变为4个n/2阶矩阵相乘

$$A=\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \quad B=\begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} \quad C=\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$$

$$C=A\times B \Rightarrow \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}=\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}\times\begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$$C_{00}=A_{00}\times B_{00}+A_{01}\times B_{10}$$

or

$$M_1+M_4-M_5+M_7$$ 由A、B计算

$$M_1=(A_{00}+A_{11})\times(B_{00}+B_{11})$$

$A_{00}+A_{11}$：$\frac{n}{2}$；$B_{00}+B_{11}$：$\frac{n}{2}$；转换为两个$\frac{n}{2}$阶乘法

Strassen矩阵乘法（3）

- 算法复杂度：乘法计算的次数M(n)
  - 如果规模是$n = 2^k$
  - 普通算法的乘法次数为：n^3
  - Strassen算法的乘法次数递推式：

$$n > 1, M(n) = 7M\left(\frac{n}{2}\right);\ M(1) = 1$$

$$M(n) = 7^{log_2 n} = n^{log_2 7} \approx n^{2.807}$$

  - 只考虑乘法次数，有提高；
  - 加上加减法次数A(n)呢？
  - 递推公式

矩阵元素的个数

$n>1$ 时 $A(n)=7A(n/2)+18(n/2)^2$, $A(1)=0$

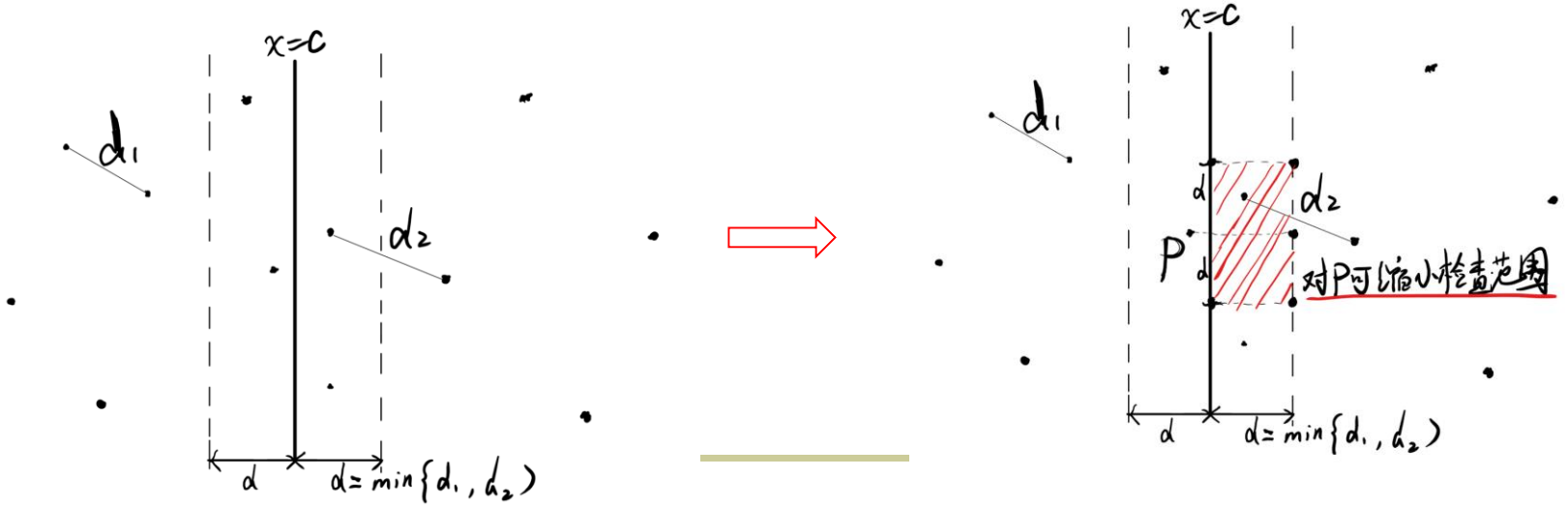
$\Rightarrow a=7,\ b=2,\ d=2$

$\Rightarrow a>b^d \Rightarrow A(n)\in\Theta(n^{\log_2 7})$ 与乘法相同

问题：如何推理出加法计算次数的闭合公式

最近对问题（1）

- 问题定义
  - $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$是平面上n个点构成的集合S，假设$n = 2^k$
  - 记得蛮力法吗？
  - 分治思想：
    - 使用一个坐标维度将点**分**为两个大小接近的集合：需要先排序
    - 分别计算两个集合中的最近对
    - 然后**合并**
    - 对小集合中的点递归使用分治思想

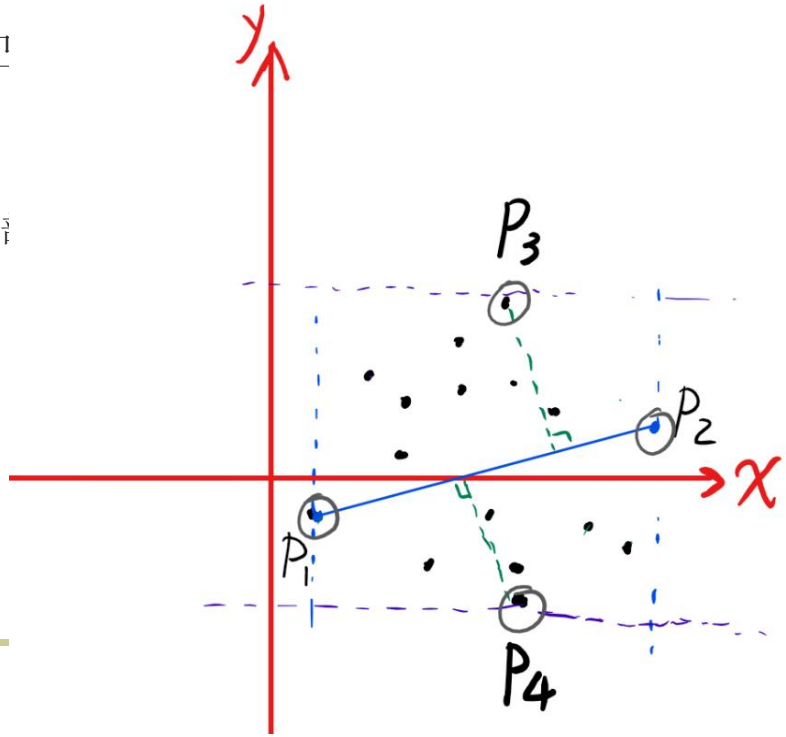


最近对问题（2）

- 算法复杂度：计算距离的次数，没计算一次也需要比较一次
  - 排序的复杂度
    - $\Theta(nlogn)$
  - 排序之后处理的递推公式
    - $T(n) = 2T\left(\frac{n}{2}\right) + M(n)$
    - M(n)是合并的时间；合并的具体逻辑？$O(n)$
    - $a = 2, b = 2, d = 1 \Rightarrow T(n) \in O(nlogn)$
  - 总体时间复杂度$O(nlogn)$
- 经过证明，最近对问题的最优效率是$\Omega(nlogn)$
- 扩展题：完成平面最近对问题的代码；输入点坐标列表，输出最近两个点。

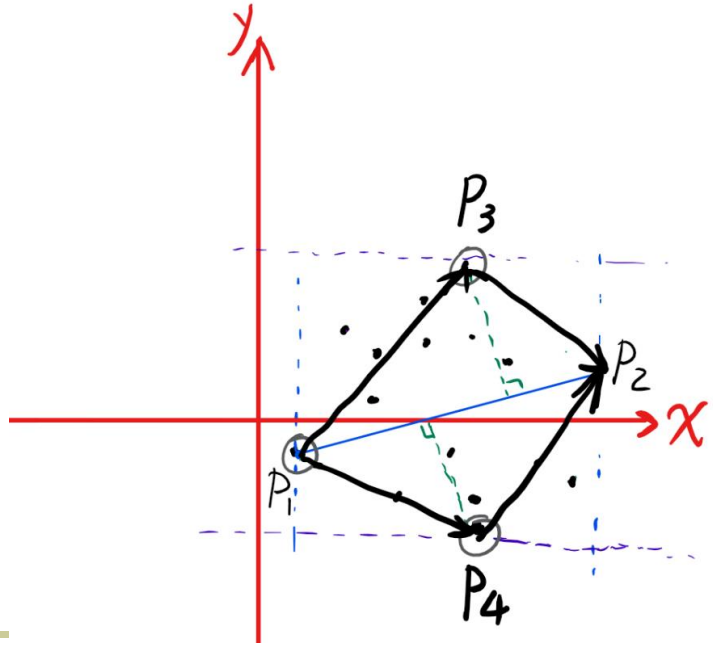
凸包问题（1）

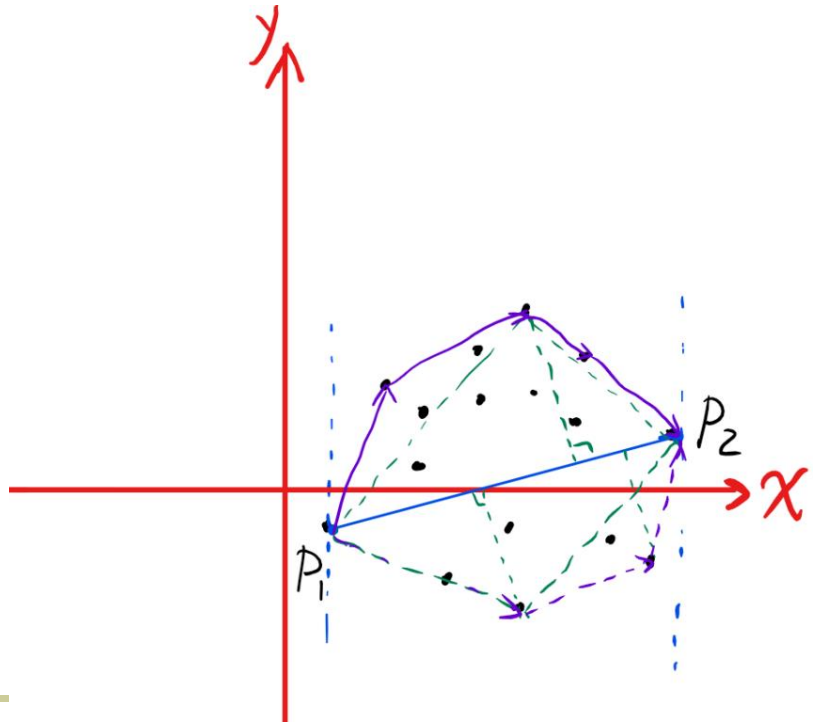
- 问题定义
  - 找到平面上n个给定点的凸包→找到能够完全包含给定n个点的以其中一些点位顶点的最小凸多边形
- 蛮力法？
  - 如果其它点在两个点连线的一边，则为多边形的一条边
- 问题回顾：设计一个线性效率的算法找到凸
- 分治法思想
  - 找到某种方式将点集合分为两个部分，且两个
  - 某一个维度上的最大值与最小值一定是顶点
  - 顶点的连线



凸包问题（2）

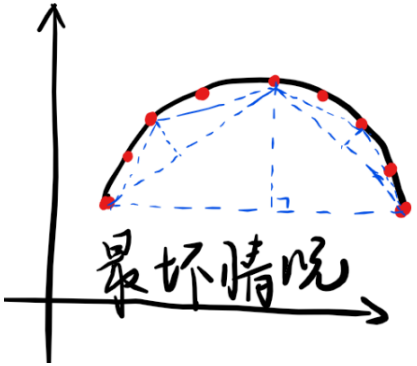
- 两个顶点的连线将点分为两个部分
- 两个部分可以用同样的方法处理
  - 找到距离直线距离最大的点
    - 一定是凸包顶点
    - 其它顶点一定位于有向线段的左侧





凸包问题（3）

- 算法复杂度
  - 最坏情况
    - 在一边，每次找到一个点
    - 检查点在左边和计算最大值的复杂度
      - $\Theta(n^2)$
  - 但是平均复杂度会好很多，为什么？
    - 比较平均地分成两个部分
    - 每个部分在三角形区域内的点不再考虑，点会减少很多



思考题

- 给定平面上有n个白点和n个黑点，任意三点不共线，试实际一个分治算法将每个白点与一个黑点相连，是所有连线互不相交。

- 有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M。返回频数最高的100个词

总结

- 分治法——分而治之
  - 将问题划分为多个子问题
  - 对每一个子问题求解
  - 将每个子问题的解合并

- 合并排序、快速排序
- 折半查找
- 二叉树
- 大整数乘法、Strassen矩阵乘法
- 最近对问题
- 凸包问题

- 大数据的分治处理

- 下节课：减治法的具体实现

谢 谢！