

程序设计实习（实验班-2023春）

距离度量及其计算：欧氏距离

授课教师：姜少峰

助教：张宇博 楼家宁

Email: shaofeng.jiang@pku.edu.cn

欧氏距离

- 是最常见的向量的距离度量

- $\forall x, y \in \mathbb{R}^d, \|x - y\|_2 = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$

- $\|\cdot\|_2$ 是 ℓ_2 -norm (2-范数)

- 欧氏空间有三角形不等式: $\forall x, y, z \in \mathbb{R}^d, \|x - y\| + \|y - z\| \geq \|x - z\|$

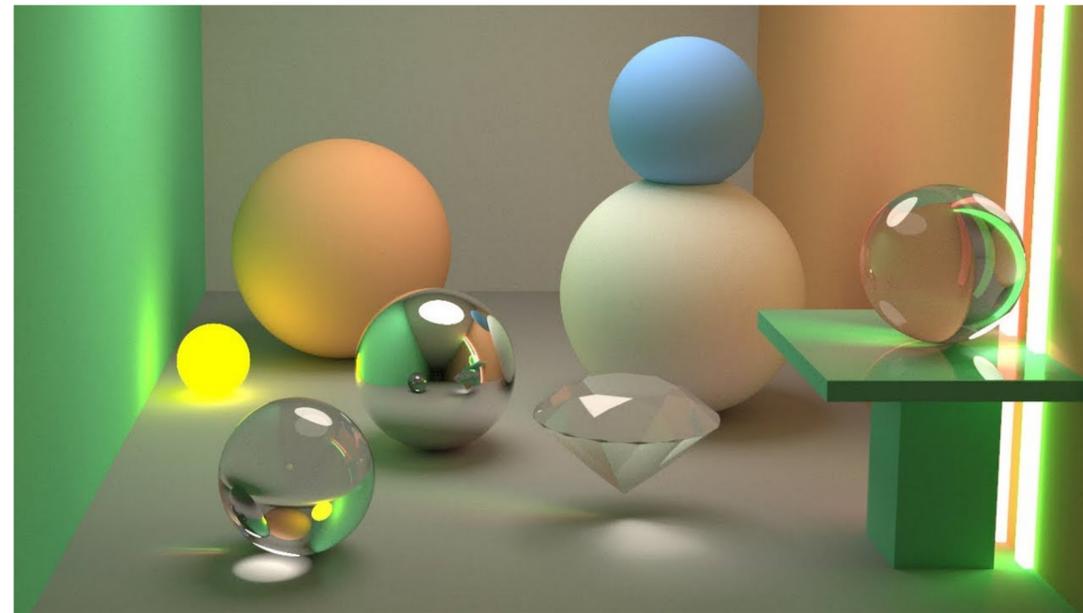
为何重要?

- 我们生存在3维欧氏空间里，距离自然定义为欧氏距离
- computer graphics, computer vision等也主要在欧氏空间内研究

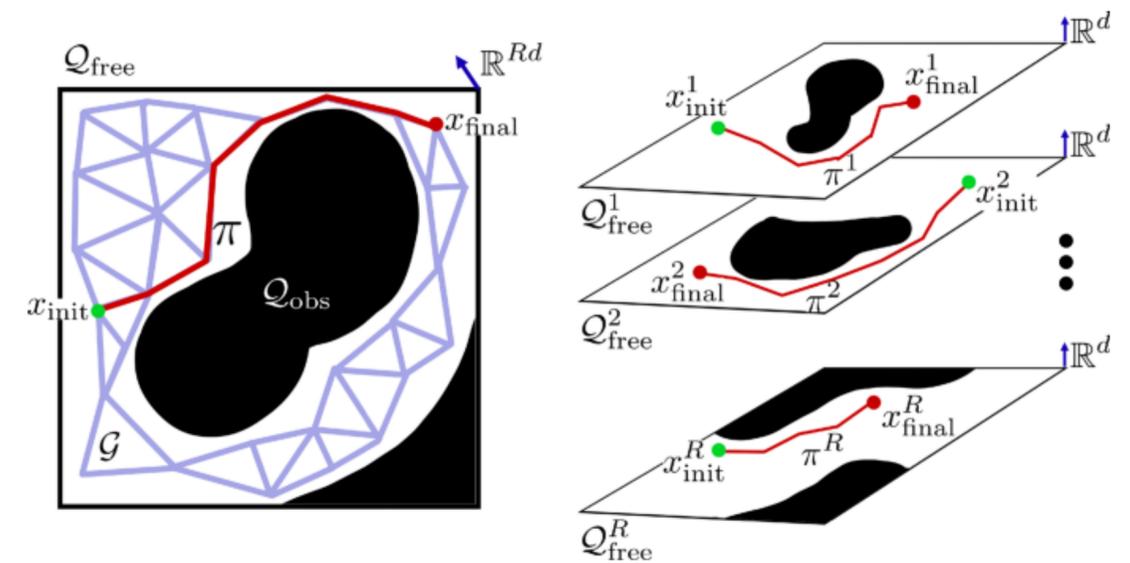
VR



渲染

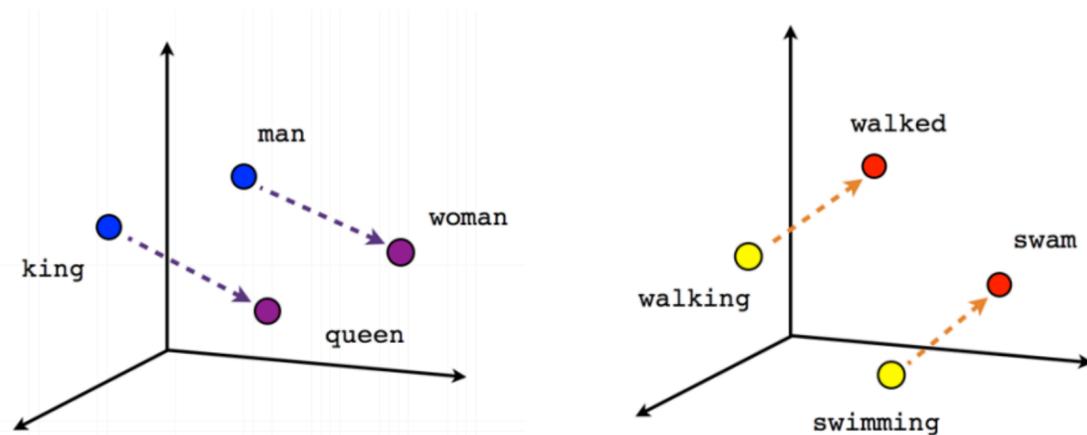


机器人运动规划



其他数据类型也经常映射到欧氏空间来处理

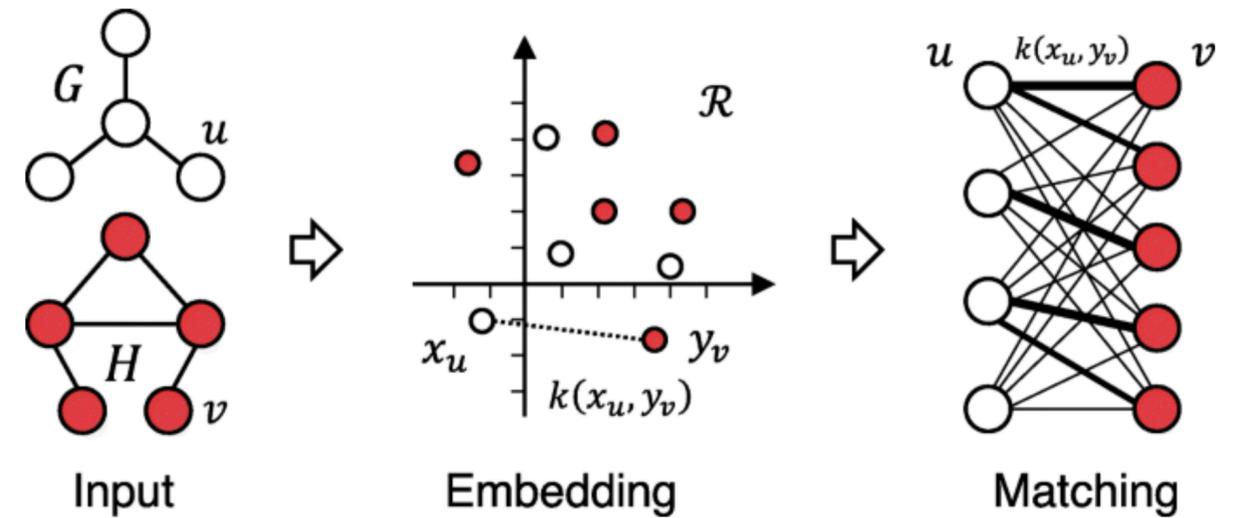
Word Embedding



Male-Female

Verb tense

Graph Embedding



Input

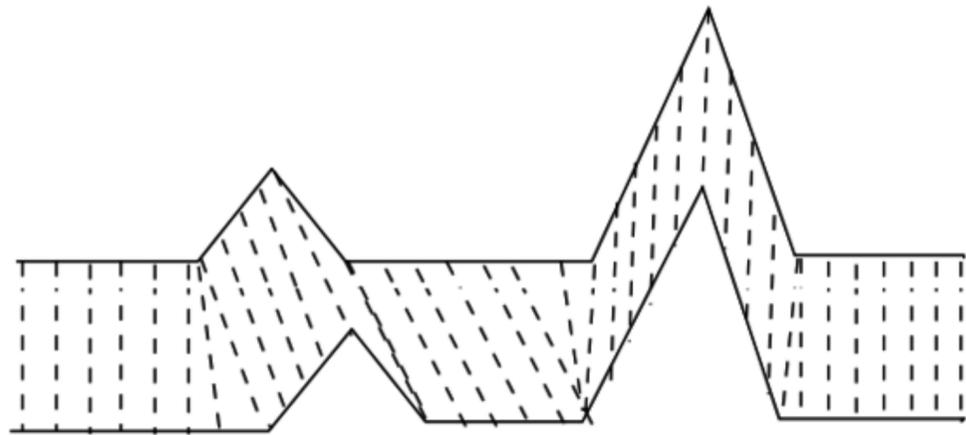
Embedding

Matching

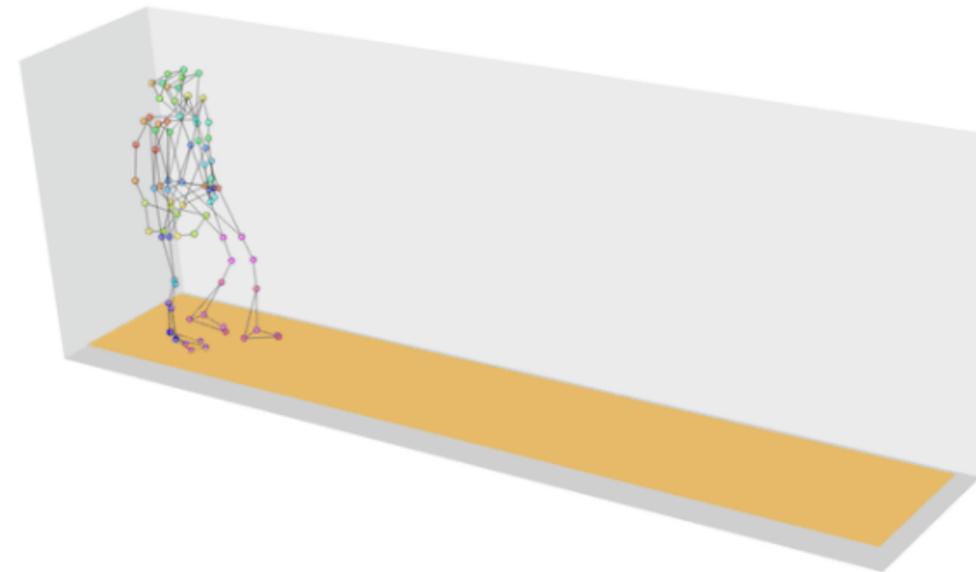
定义在欧氏空间上的其他距离度量

Dynamic Time Warping (DTW) Distance

- 衡量两个时间序列的距离



典型场景：速度不同的两个相似轨迹要匹配



- DTW：序列点集A和B，要求找一个**单调配对**，最小化所有连边的总欧氏长度和

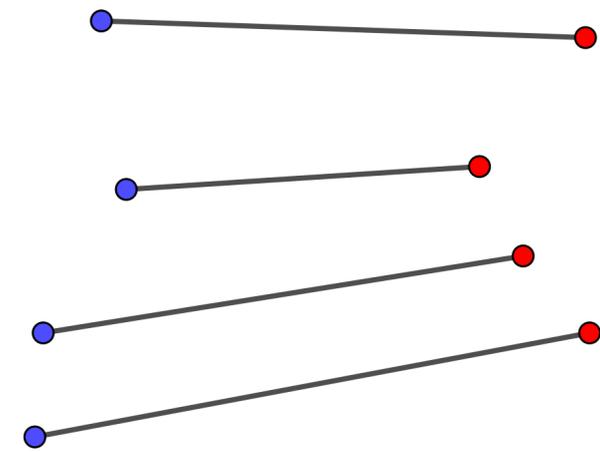
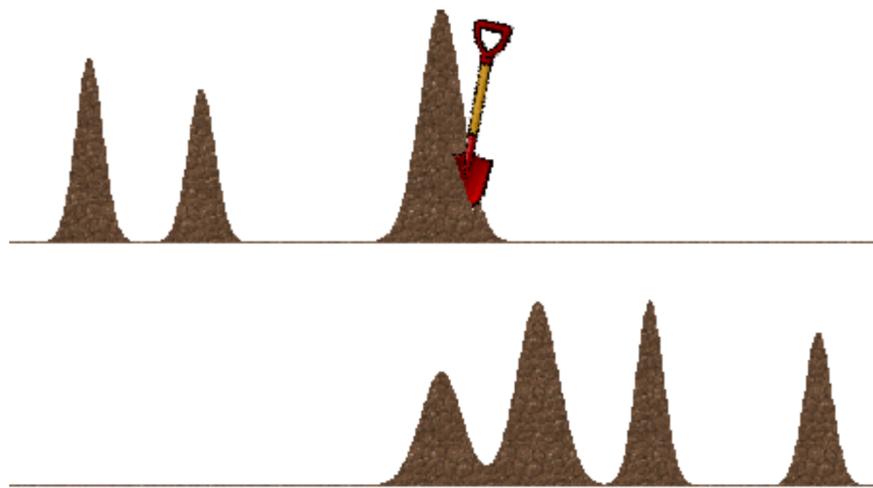
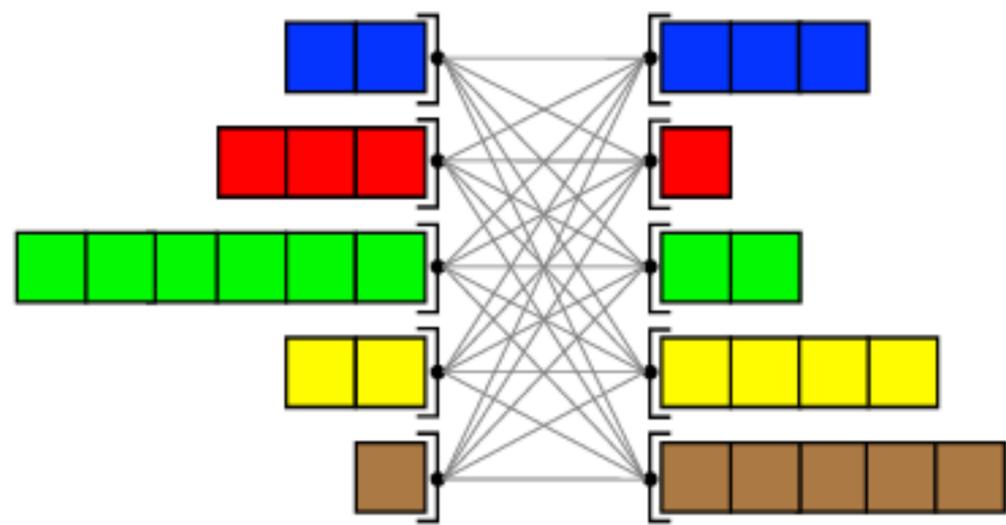
$A_1 \leftrightarrow B_1, A_n \leftrightarrow B_m$ ，但 A_i, B_j 可以随意连边，只是要求必须单调（如 B_{i+1} 连到的点全都必须在 B_i 连的之后）

可用动态规划求解

定义在欧氏空间上的其他距离度量

Earth Mover's Distance (EMD)

- 一般来说是两个（离散）分布的距离度量
 - A、B分布分别画概率分布的直方图，以最小总概率移动将A分布转化成B分布
- 典型场景是欧氏情况：给A和B两个n点的二维点集，求min-cost matching



欧氏情况：相当于两组均匀在n点集合上的分布，求EMD

欧氏距离上的高效近似技术

问题设定

我们主要以 $d = 2$ 进行讨论，但所有算法很容易推广到一般 d 维

- 输入 n 个 d 维、坐标在 $[\Delta] = \{1, \dots, \Delta\}$ 范围的数据点，即数据都在 $[\Delta]^d$ 内

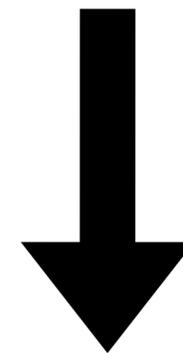
例如考虑unsigned int, $\Delta = 2^{32}$

- 介绍几个技术及如何用于高效（近似）求解若干基本问题

- 格点离散化技术：直径、* 最小包围球

- 四分树技术：最近邻查询

- WSPD = Well Separated Pair Decomposition：全局最近点对，* 最小生成树



层层递进，后者的构造基于前者

核心思想：离散化/找代表点

1. 格点离散化

格点离散化

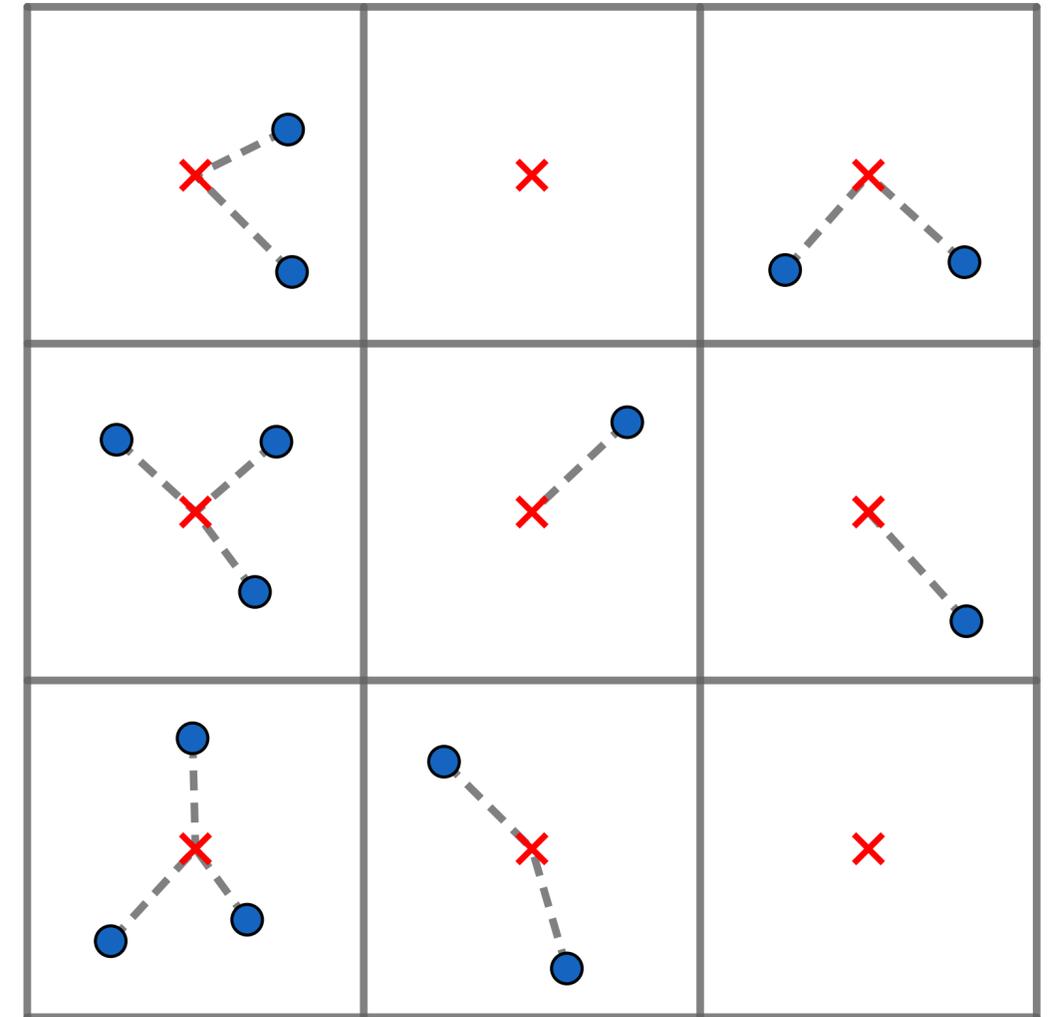
每个正方形对应x、y坐标在长度为 ℓ 区间的点： $(x, x + \ell] \times (y, y + \ell]$

- 将整个空间划分成 $\ell \times \ell$ 的小正方形
- 将数据点“round”到所属格子的格点中心
 - 由于正方形互不相交，这个rounding是唯一的

d维是 $\sqrt{d}\ell$

- 性质：每个点移动距离 $\leq \sqrt{2}\ell$

- 性质：在一个 $R \times R$ 的区域内，有至多 $\left(\frac{R}{\ell}\right)^2$ 个 $\ell \times \ell$ 正方形

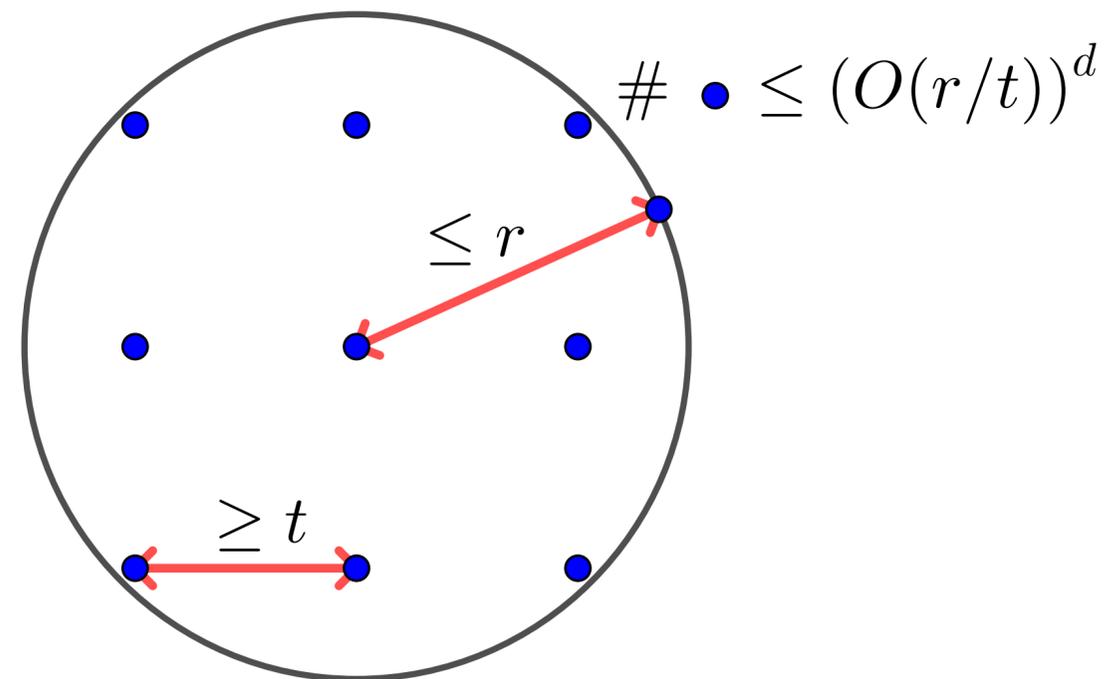
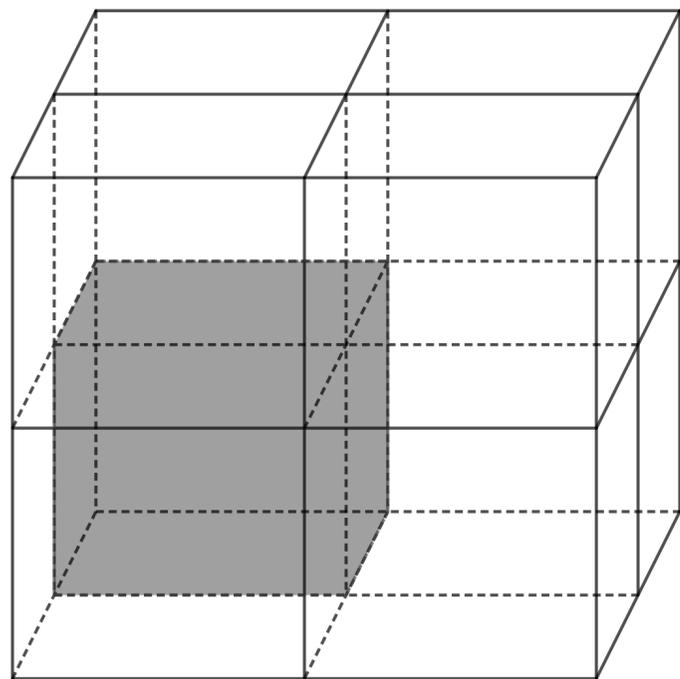


针对d维hypercube是 $(\lceil R/\ell \rceil)^d$

欧氏空间基于体积的根本性质

性质：在一个 $\overbrace{R \times \dots \times R}^d$ 的区域内，有至多 $\left(\frac{R}{l}\right)^d$ 个 $\overbrace{\ell \times \dots \times \ell}^d$ 正方形

性质：在一个半径是 r 的 d 维球里，最多可以存在 $(O(r/t))^d$ 个两两距离 $\geq t$ 的点



基本问题：直径

- 给定一个点集 $P \subseteq [\Delta]^2$
- 定义直径 $\text{diam}(P) := \max_{x,y \in P} \|x - y\|_2$
- 求 $\text{diam}(P)$ 英文diameter

线性时间 $(1 + \epsilon)$ -近似直径

T可通过选取任意点u, 求u到最远点的距离得到 (见第一讲)

即满足 $1/2 \cdot \text{diam}(P) \leq T \leq \text{diam}(P)$

- 先 $O(n)$ 时间找一个直径的2-近似值T

可以是任意确定点

- 作 $\ell := \epsilon \cdot T / \sqrt{2}$ 的网格, 并round到中心

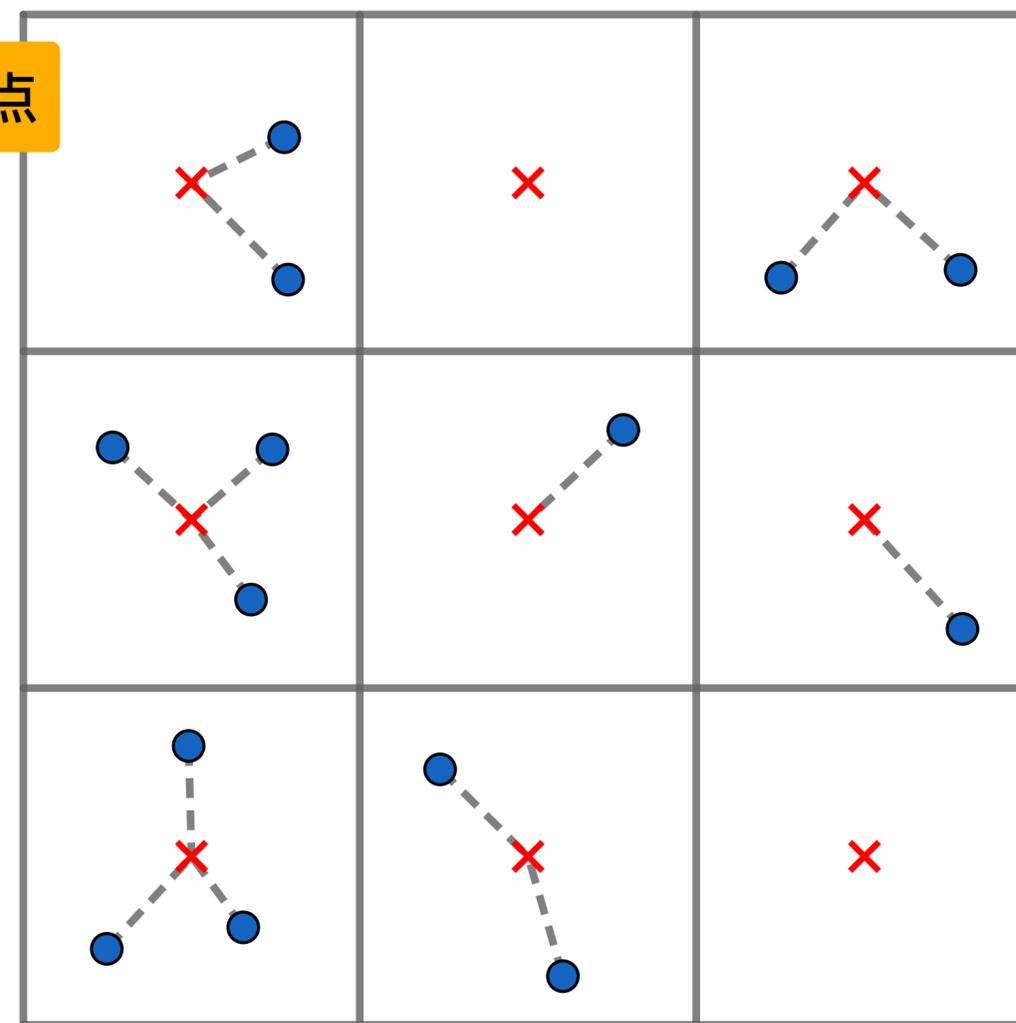
- 因此每个点移动了 $\leq \sqrt{2}\ell \leq \epsilon \cdot \text{diam}(P)$

- 因此新点集 P' 满足 可在 $O(nd \log n)$ 时间构造 P'

$$\text{diam}(P') \in (1 \pm \epsilon) \cdot \text{diam}(P)$$

- 算法: 在 $|P'|$ 上暴力求直径, 复杂度 $|P'|^2 \cdot d$

P' 所有点都在 $\text{diam}(P') \times \text{diam}(P')$ 大方格内, 小方格 $\ell \geq \Omega(\epsilon \cdot \text{diam}(P))$, 故 $|P'| \leq (O(1/\epsilon))^2$



总复杂度: $O(nd \log n) + \epsilon^{-O(d)}$

一些实现细节

- 设把点都round为方格**左下角**，且方格都是 $\ell \times \ell$ 的 $O(d)$ 时间
 - 那么给定 $x = (x_1, \dots, x_d)$ ，左上角的坐标是 $(\lceil x_1/\ell \rceil, \dots, \lceil x_d/\ell \rceil)$
- 直接round完后的到的 P' 会有很多的重复点，需要**去重**
 - 一种通用的不依赖于其他数据结构的去重：**排序** — $O(nd \log n)$ 复杂度
 - 排序后，值相等的元素形成连续的一段，可以 $O(n)$ 时间扫描找到所有段

作业：近似求欧氏点集直径

- <http://cssyb.openjudge.cn/23hw6/>
- Deadline: 4月12日

* 类似算法：最小包围球的coreset

最小包围圆的圆心

• 输入 $P = \{x_1, \dots, x_n\} \subset \mathbb{R}^2$, 求 $c \in \mathbb{R}^2$ 使 $f(P, c) := \max_{x_i \in P} \|x_i - c\|_2$ 最小化

• ϵ -Coreset: $S \subseteq P$ 使得对于任何 $c \in \mathbb{R}^2$ 有 $f(S, c) \geq (1 - \epsilon) \cdot f(P, c)$

• 算法:

OPT代表最优解的值

要求 $\text{OPT} \leq T \leq 2\text{OPT}$

$f(S, c) \leq f(P, c)$ 是根据定义总是成立的

• $O(n)$ 时间找一个OPT的 $O(1)$ -近似T (可选任意点u, 令T为u到最远点距离)

• 作 $\ell := \epsilon \cdot T / (2\sqrt{2})$ 的网格, 并round

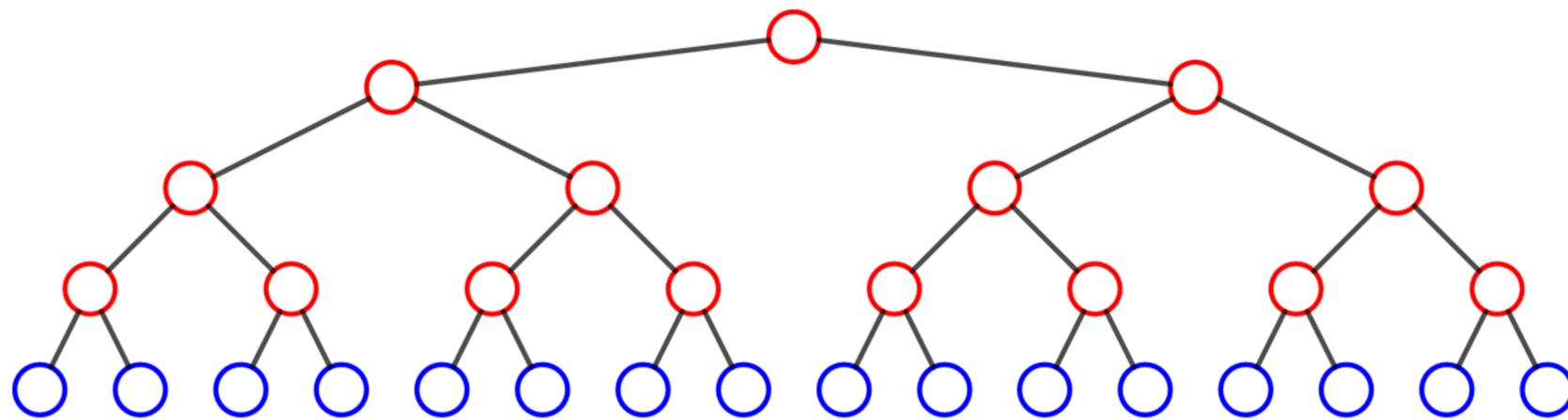
类似直径, 可证 $|P'| \leq \text{poly}(1/\epsilon)$

• 得到的点集 P' 即为 ϵ -coreset

每个点移动距离 $\leq \epsilon \cdot \text{OPT}$, 因此对任意 $c \in \mathbb{R}^2$, $\|x - c\|_2$ 也仅会改变 $\epsilon \cdot \text{OPT}$

* Merge-and-reduce框架

- Coreset重要性质：可合并，即 $\text{coreset}(A) \cup \text{coreset}(B)$ 是 $\text{coreset}(A \cup B)$
- 每次合并两个coreset S_1, S_2 后再做 $S_1 \cup S_2$ 的coreset
- $O(\log n)$ 层后变成单个coreset



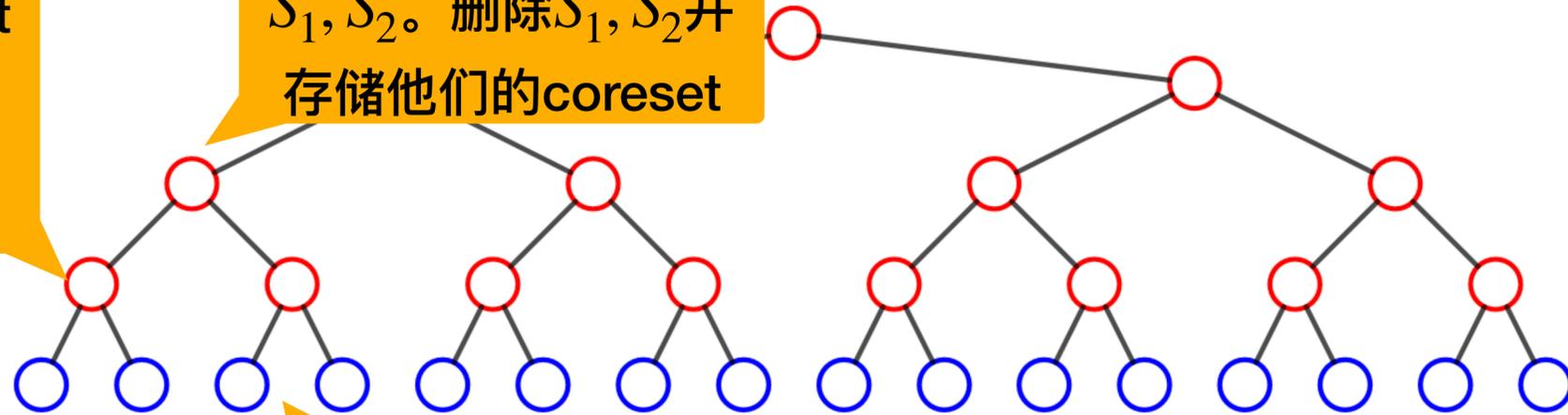
- Level 0: final **coreset**
- Level 1: **coresets**
- Level 2: **coresets**
- Level 3: **coresets**
- Level 4: data buckets

* 数据流算法

1. 当前点不够2个bucket则暴力存；够2个后存他们的coreset S_1 (并释放bucket内存)

3. 若前4个bucket都满了，则已得到了1-2和3-4的coreset S_1, S_2 。删除 S_1, S_2 并存储他们的coreset

2. 第3、4个bucket类似处理：等3、4都满了，释放内存并只存他们的coreset S_2



- Level 0: final **coreset**
- Level 1: **coresets**
- Level 2: **coresets**
- Level 3: **coresets**
- Level 4: data buckets

- 随着数据的到来尽可能进行merge-reduce；任何时候每层至多存 $O(1)$ 个coreset

2. 递归格点离散化：四分树

递归划分：四分树

以二维为例

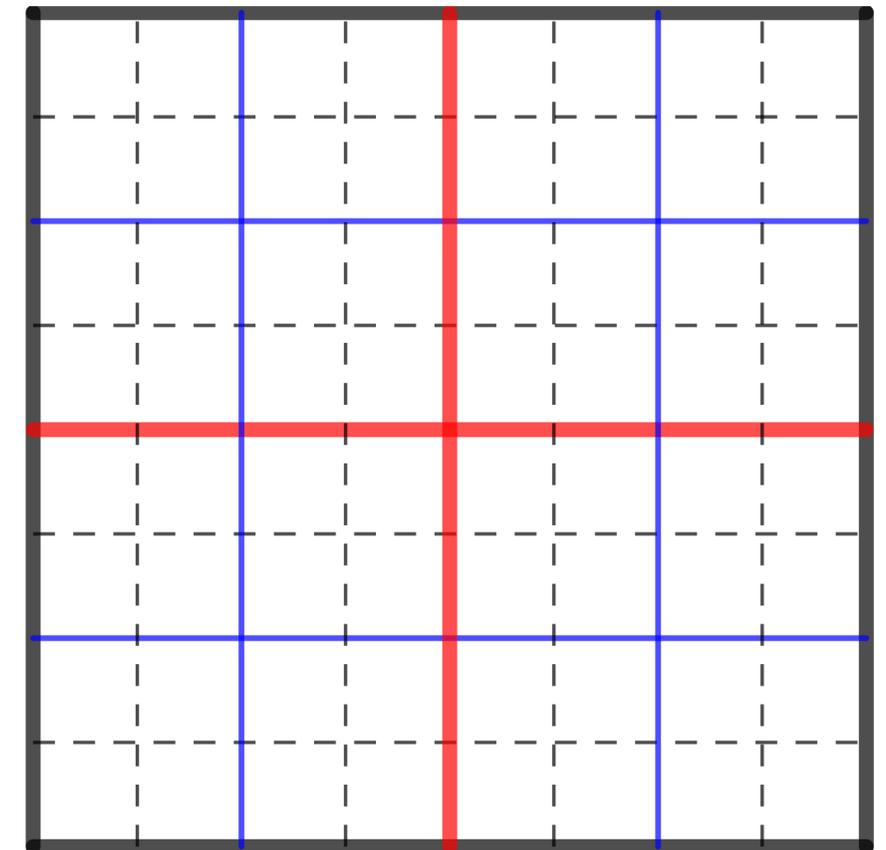
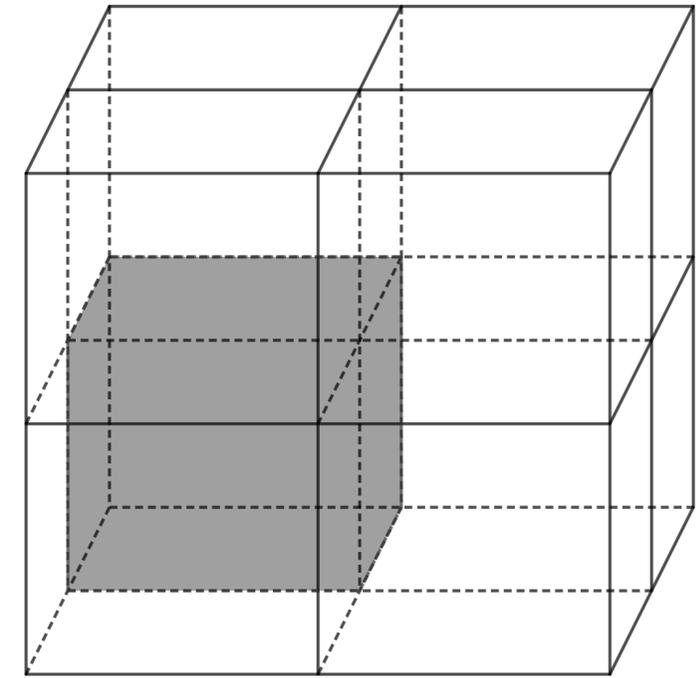
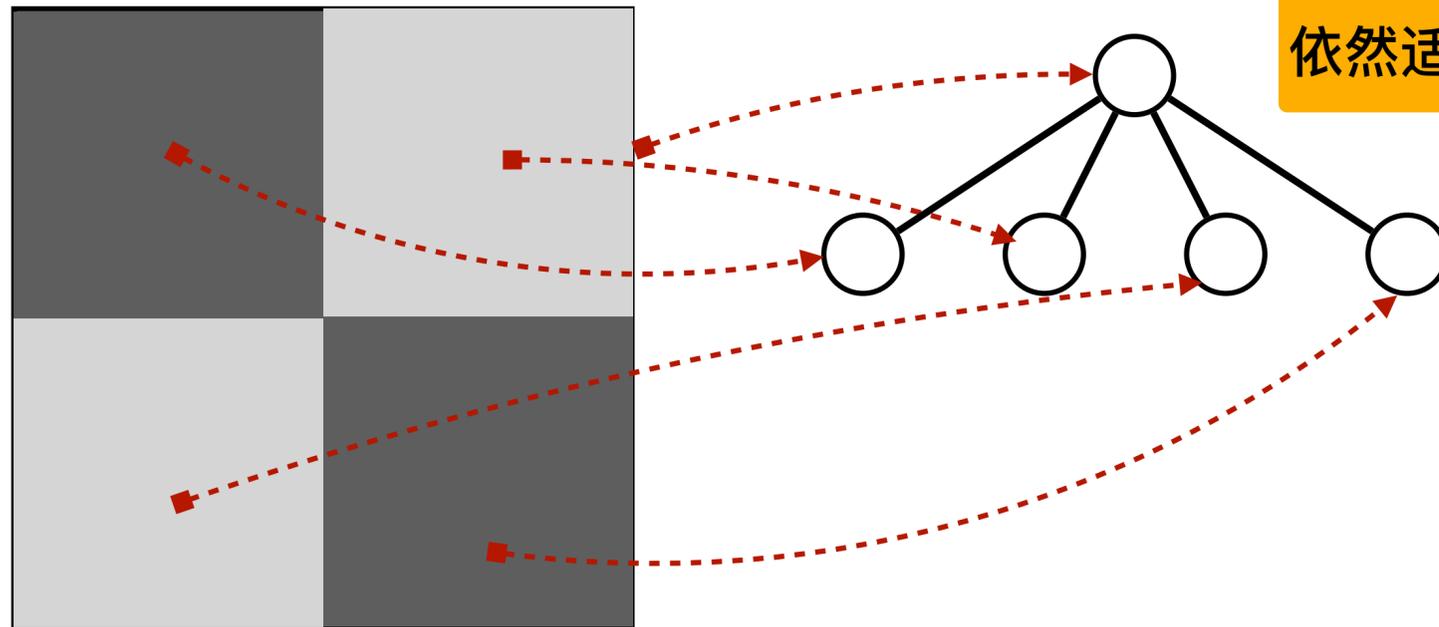
d维叫“d维四分树”

d维是从 $[\Delta]^d$ 划分

- 从最大的 $\Delta \times \Delta$ 的正方形开始
- 递归将当前 \square 等分成4个边长为一半的 \square
- 组织成树形结构，直到只含单点停止，至多 $\log_2 \Delta$ 层

d维分成 2^d 个边长为一半的 \square

依然适用于d维



伪代码

d维

全局调用 $\square = [\Delta]^d$

BuildTree(\square, P)

两个停止情况：不含数据点直接返回空；只含1个点返回当前 \square

if ($P \cap \square = \emptyset$) return NULL

let T.root = ($\square, \square \cap P$)

除了记录 \square ，也要记录 \square 里含有的数据点集

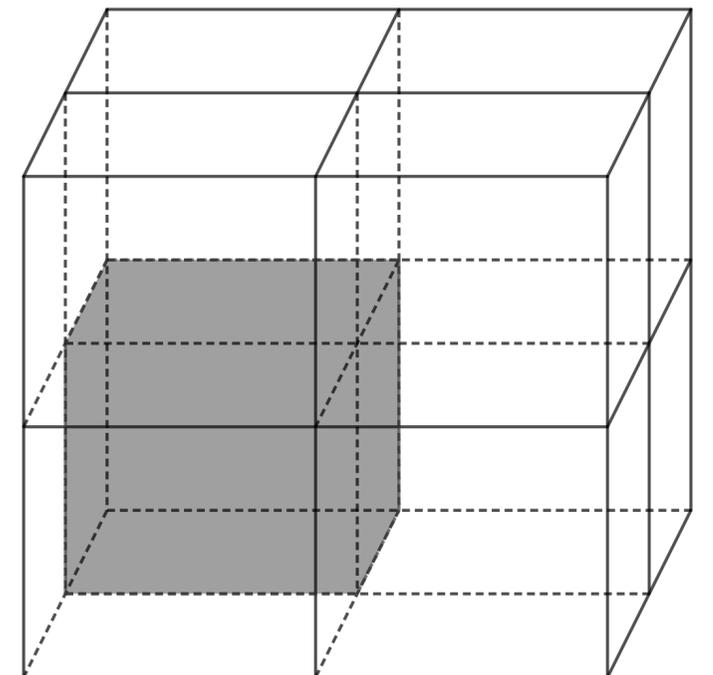
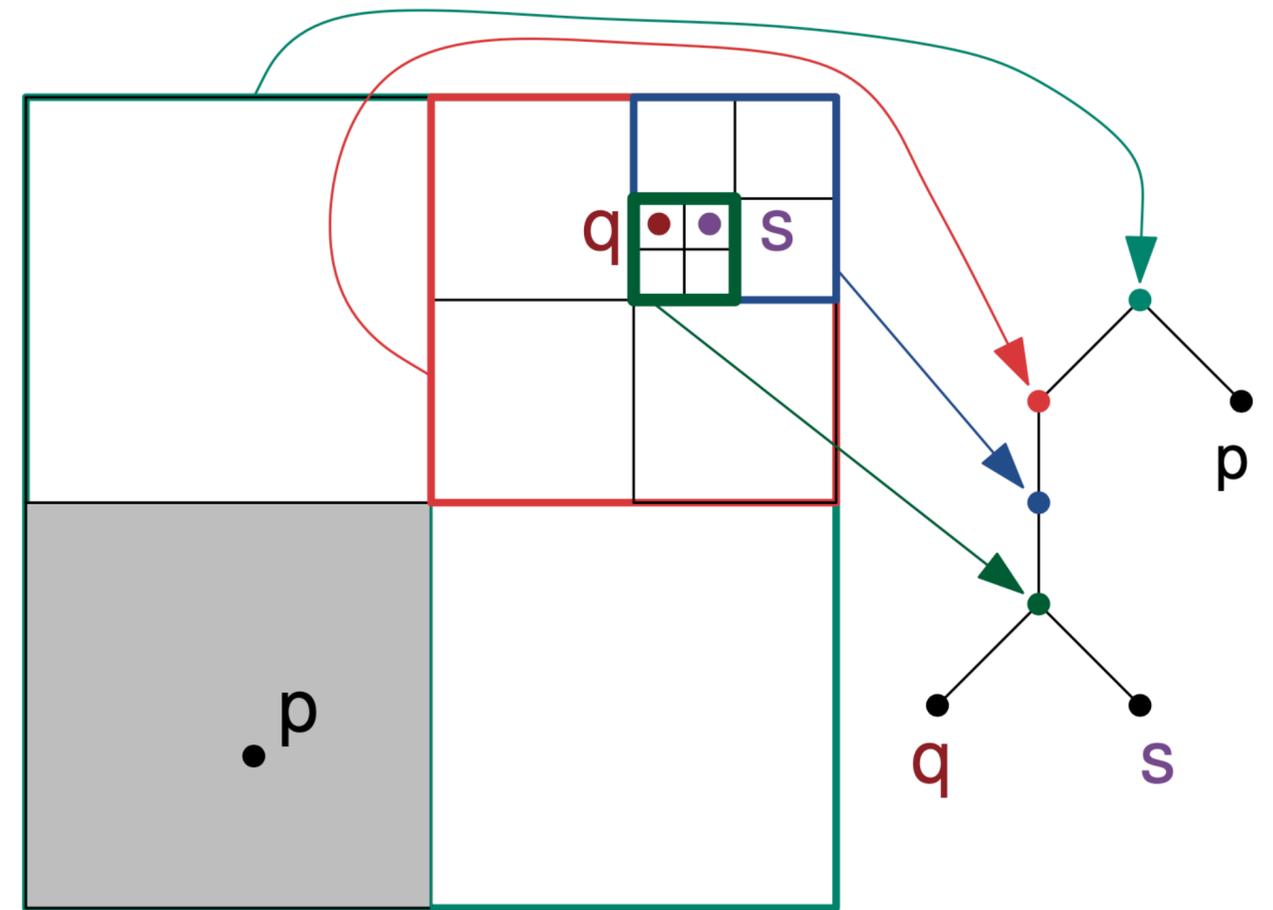
if ($|P \cap \square| = 1$) return T

evenly divide \square into 2^d sub-squares $\square_1, \dots, \square_{2^d}$

for $i = 1, \dots, 2^d$, let T.child $_i$ = BuildTree($\square_i, P \cap \square_i$)

return T

整个算法只会产生 $O(n \log \Delta)$ 个非空 \square (非空即 $\square \cap P \neq \emptyset$)



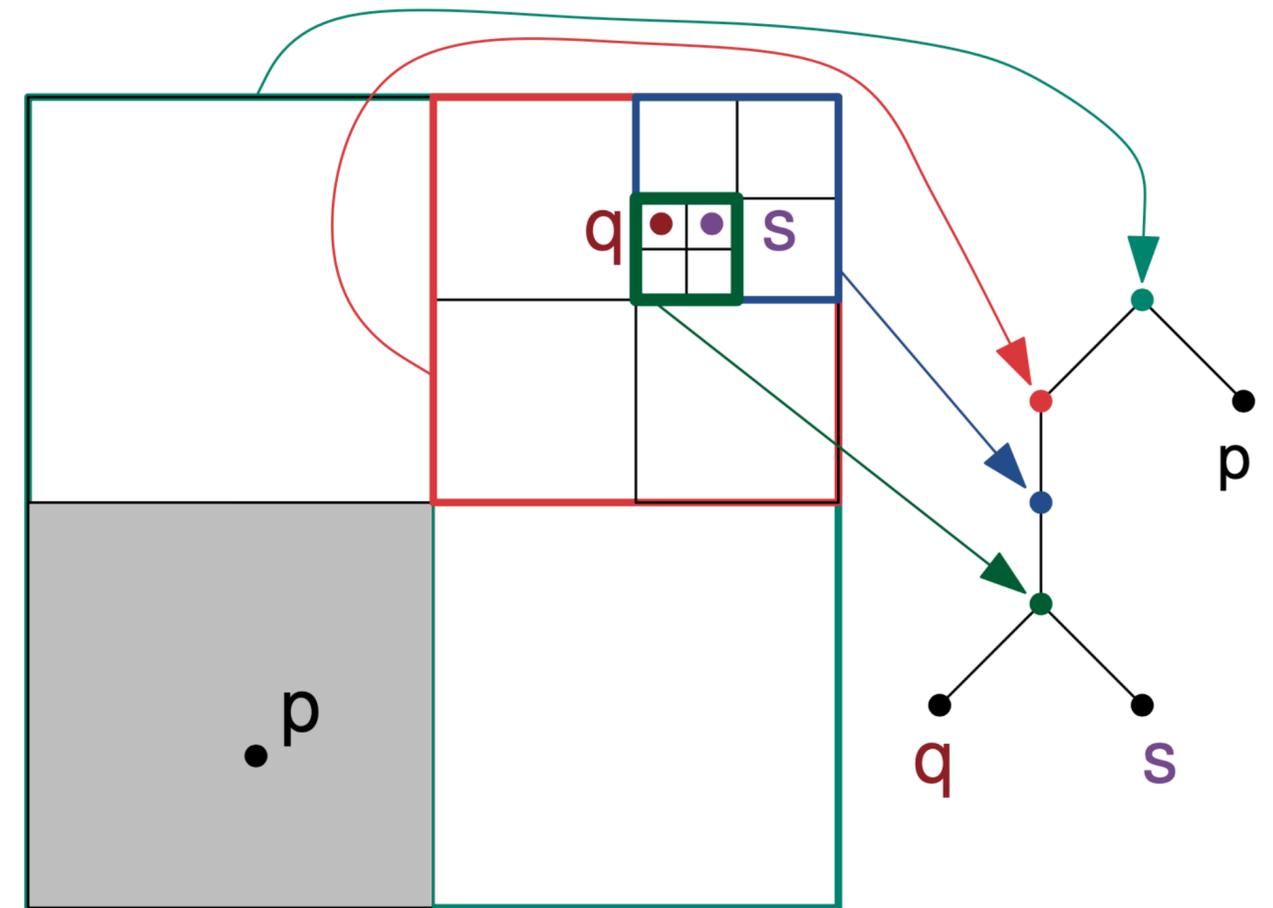
实现细节

存储

- 每个 \square 对应一个树上的节点
- 另需开一个数组/vector<int>来存 $P \cap \square$
- 还需要存储 2^d 个 \square_i 对应的指针
- 总空间复杂度: $O(n \cdot 2^d \log \Delta)$
- 树的每层 \square 的并集存储整个点集 P
- 共 (至多) $\log \Delta$ 层

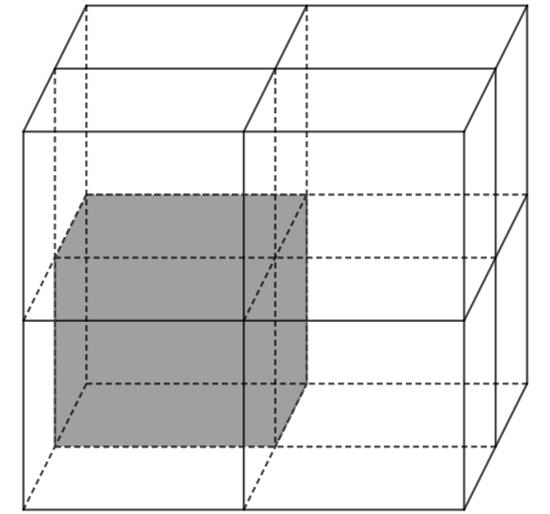
2^d 来自于存 2^d 个 \square_i 对应指针所需空间

注意: 该树中没有空 \square



实现细节

对 $i = 1, \dots, 2^d$ 求 $P \cap \square_i$ 总共需要多少时间?



- 暴力: $n \cdot 2^d \cdot d$, 即所有 $x \in P$ 与 \square_i 比较一次 (每次比较代价是 d)
- 可以做到 $n \cdot d$:
 - 设当前 $\square = [\ell] \times [\ell]$, 则 2^d 个 \square_i 是每维从 $(0, \ell/2]$ 和 $(\ell/2, \ell]$ 2选1进行组合
 - \square_i 每维选哪个区间可用 i 的二进制表示来确定
- 算法:
 - 对 $x \in P$, 判断 x 每一维属于 $(0, \ell/2]$ 还是 $(\ell/2, \ell]$, $O(d)$ 时间确定所属 \square_i

实现细节

可选优化：省去 2^d

evenly divide \square into 2^d sub-squares $\square_1, \dots, \square_{2^d}$

for $i = 1, \dots, 2^d$, let $T.\text{child}_i = \text{BuildTree}(\square_i, P \cap \square_i)$

- 这两行直接实现需要对每个树中节点穷举 2^d 个 \square_i ，共需 $2^d \cdot n \log \Delta$ 时间和空间
- 接着上页，把主循环检查点集 P ，只保留非空的 \square_i ，这样就是 $O(|P|d)$ 时空
 - 所有层总共就是 $nd \log \Delta$
- 总时空复杂度： $O(nd \log \Delta)$ 若使用该优化， 否则是 $O(n2^d \log \Delta)$

不采用该优化会使算法简单一些，若 d 没有很大，建议不采用

利用四分树进行 ϵ -近似最近邻查询

- 问题:

- 对输入点集 $P \subseteq [\Delta]^d$ 进行预处理

即 \hat{q} 到 q 的距离是 q 到 q 在 P 最近邻距离的 $(1 + \epsilon)$ 倍

- 对任意 $q \in [\Delta]^d$ 回答 $\hat{q} \in P$, 使得 $\|\hat{q} - q\| \leq (1 + \epsilon) \cdot \min_{q' \in P} \|q' - q\|$

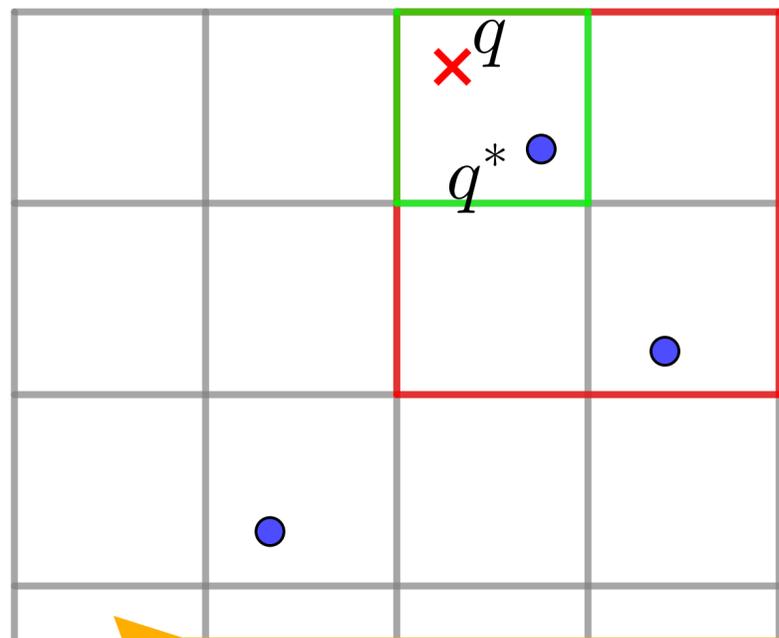
- 预处理: BuildTree(\square , P), 这里调用 $\square = [\Delta]^d$

- 我们将给出一个 $O(\epsilon^{-d} \log \Delta)$ 时间的查询算法返回满足要求的 \hat{q}

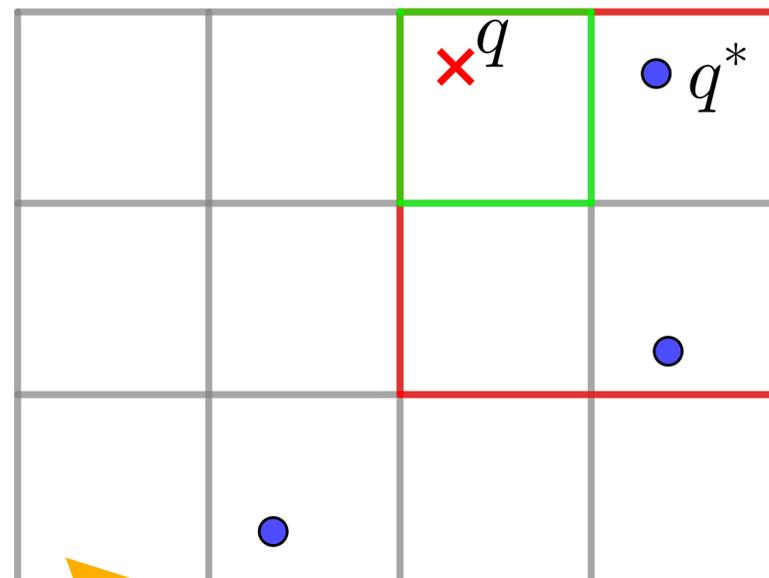
查询：一个（错误的）理想化算法

- 设查询点是 q ，最近邻是 q^* ，距离是 $r^* = \|q - q^*\|$
- 从根 \square 开始，每次递归进入 q 所在的某个 \square_i ，直到当前 \square 只有一个点 q' ，返回 q'

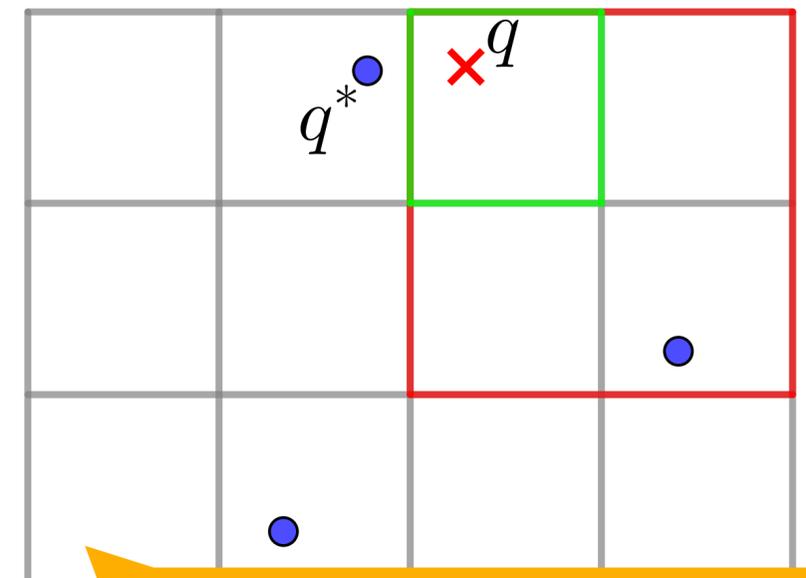
这对应于沿着 $\log \Delta$ 层的路径走到底



成功的情况：经历红方格-绿方格，最后找到了 q^*



问题：绿方格可能是空的，只能从红方格找。但这样有很多其他点，而不是唯一点



问题： q^* 甚至可能不在红格子里，因此不能只递归查找 q 所在格子，周围格子也要考虑

正确的算法?

- 正确的算法不能只从q所在格子递归
 - 递归搜索的时候，需要把全部 2^d 个 \square_i 都检查一下
- 但这样复杂度无保证.....
- 为优化复杂度，需要一些“剪枝”，即排除掉一些“过远”的 \square_i

剪枝策略

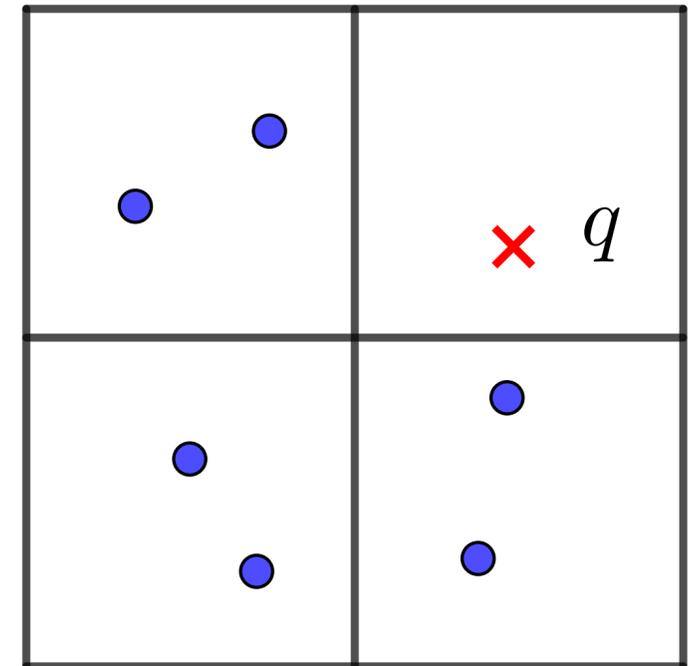
- 以第一层的4个 \square_i 为例：什么情况下才不用在 \square_i 继续递归？
- 根本要求：不能miss掉最近邻 q^* 的 $(1 + \epsilon)$ -近似
- 设搜索过程中维护一个当前最优解 \hat{q} ，设 $r := \|q - \hat{q}\|$
- 剪枝策略：若对任何 $q' \in \square_i$ 都有

$$r \leq (1 + \epsilon) \cdot \|q - q'\|$$

则即使 $q' = q^*$ 也不再需要考虑 \square_i 了

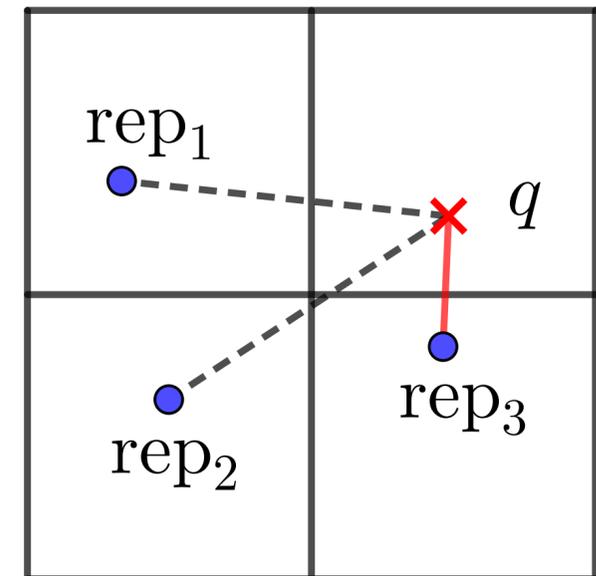
但这个判据很难高效实现

说明当前解是 \square_i 中任何点的 $(1 + \epsilon)$ -近似



进一步离散化：选取代表点

- 优化策略：每个 \square 任选一个数据点作为代表点，记为 rep_{\square}
- 为优化复杂度，算法只会从 rep 中搜索（近似）最近点



新的剪枝规则

- 因为现在只考虑代表点 rep ，剪枝规则也需要跟着改
- 依然设 r 为“当前最优解” \hat{q} 到查询点 q 的距离
- 新剪枝： \square_i 不需要继续考虑，若

\square_i 的直径； $\ell \times \ell$ 的是 $\sqrt{2}\ell$

$$r \leq (1 + \epsilon) \cdot (\|q - \text{rep}_i\| - \text{diam}(\square_i))$$

将 \leq 替换成 $>$ 则可得到“需要继续考虑 \square_i ”的判据

新旧剪枝条件的对比

- **旧剪枝**: \square_i 不需被考虑, 若 $\forall p \in \square_i$, 有 $r \leq (1 + \epsilon) \cdot \|q - p\|$
- **新剪枝**: \square_i 不需被考虑, 若 $r \leq (1 + \epsilon)(\|q - \text{rep}_i\| - \text{diam}(\square_i))$
- 新剪枝条件更“苛刻”: 被新规则剪掉的是旧规则的子集
 - 即某个 \square_i 在新条件不考虑 \Rightarrow \square_i 在旧条件不考虑:
 - 需要验证: 若 $r \leq (1 + \epsilon)(\|q - \text{rep}_i\| - \text{diam}(\square_i))$, 则
$$\forall p \in \square_i, \text{ 有 } r \leq (1 + \epsilon) \cdot \|q - p\|$$

进一步解释

- 现验证：若 $r \leq (1 + \epsilon)(\|q - \text{rep}_i\| - \text{diam}(\square_i))$ ，则

$$\forall p \in \square_i, \text{ 有 } r \leq (1 + \epsilon) \cdot \|q - p\|$$

- 性质： $\forall p \in \square_i, \text{ 有 } \|q - \text{rep}_i\| - \text{diam}(\square_i) \leq \|q - p\|$

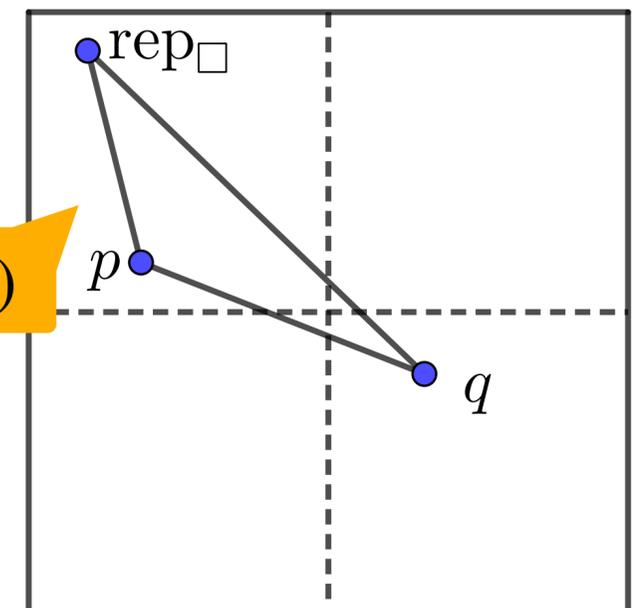
- 观察：任何 p 可由 rep_i 移动至多 $\text{diam}(\square_i)$ 距离到达

- 数学语言：

$$\|q - p\| \geq \|q - \text{rep}_i\| - \|\text{rep}_i - p\| \geq \|q - \text{rep}_i\| - \text{diam}(\square_i)$$

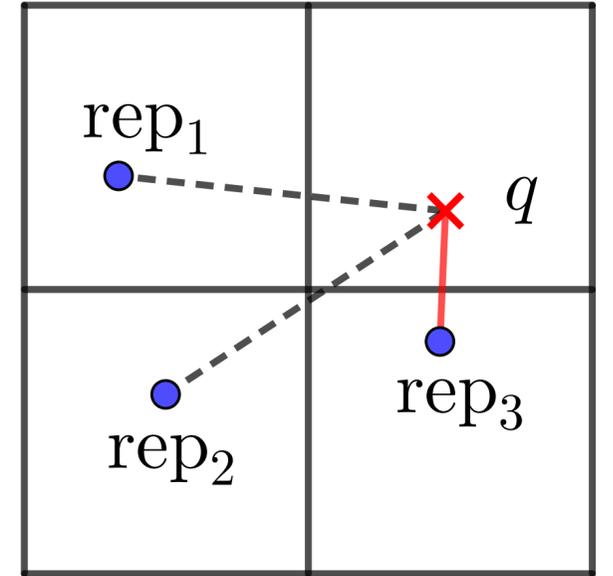
三角形不等式

利用 $\|\text{rep}_i - p\| \leq \text{diam}(\square_i)$



完整算法

输入：查询点 q ，误差参数 $0 < \epsilon < 1$ ；查询所需时间 $O(\epsilon^{-d} \log \Delta)$



设集合 A_i 为需要检查的四分树第 i 层的格子的集合 ($0 \leq i \leq \log \Delta$)

初始 A_0 放入最大格子 $\square = [\Delta]^d$, $A_i = \emptyset$ ($1 \leq i \leq \log \Delta$), $r = \infty$, $\hat{q} = \text{NULL}$

for $i = 1, \dots, \log \Delta$

i 是层数，按层进行迭代

\hat{q} 是当前解， r 是 $\|q - \hat{q}\|$

设 S 为所有 A_{i-1} 的格子的 2^d 个子 \square 的集合

更新当前解为本层最近的rep

对每个 $\square_w \in S$ ，若 $\|q - \text{rep}_w\| < r$ ，更新 $r = \|q - \text{rep}_w\|$ 及 $\hat{q} = \text{rep}_w$

对每个 $\square_w \in S$ ，若 $r > (1 + \epsilon) \cdot (\|q - \text{rep}_w\| - \text{diam}(\square_w))$ ，将 \square_w 放入 A_i

此处是“需要放入 \square_w ”的判据，因此是 $>$

返回 \hat{q}

作业：欧氏空间近似最近邻查询

- <http://cssyb.openjudge.cn/23hw7/>
- Deadline: 4月19日

* 算法分析

- 我们要考虑的：
 - 正确性：最后是否有 $r \leq (1 + \epsilon)r^*$
 - 效率：总运行时间是否是 $\epsilon^{-d} \log \Delta$

* 性质1: 若 $r > (1 + \epsilon)r^*$ 则 q^* 所在 \square 不会被剪枝

- 回忆算法步骤:

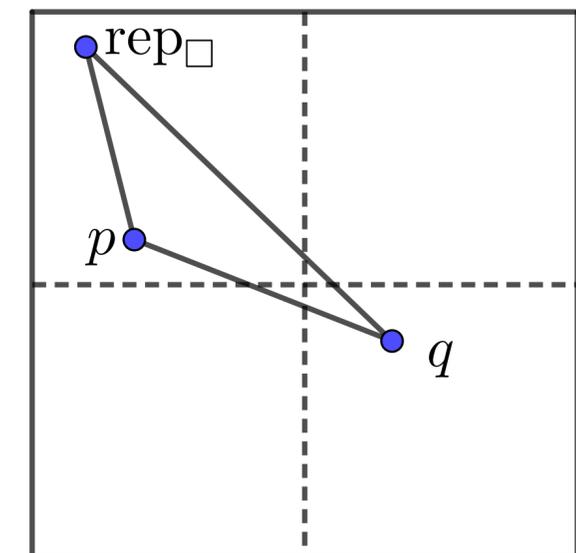
对每个 $\square_w \in S$, 若 $r > (1 + \epsilon) \cdot (\|q - \text{rep}_w\| - \text{diam}(\square_w))$, 将 \square_w 放入 A_i

- 设 \square_w 为 q^* 所在格子, 下面验证 \square_w 满足放入 A_i 的判定条件

Claim: 设当前在第 i 层且 $r > (1 + \epsilon)r^*$, 则 q^* 所在格子一定可以放入 A_i

$$\begin{aligned} & (1 + \epsilon) \cdot (\|q - \text{rep}_w\| - \text{diam}(\square_w)) \\ & \leq (1 + \epsilon) \|q - q^*\| \\ & = (1 + \epsilon) \cdot r^* \\ & < r \end{aligned}$$

利用性质: $\forall p \in \square_i$, 有
 $\|q - \text{rep}_i\| - \text{diam}(\square_i) \geq \|q - p\|$



* 性质2: 算法运行到 $\text{diam}(\square) \leq \epsilon/2 \cdot r^*$ 就会停止

- 回忆算法步骤:

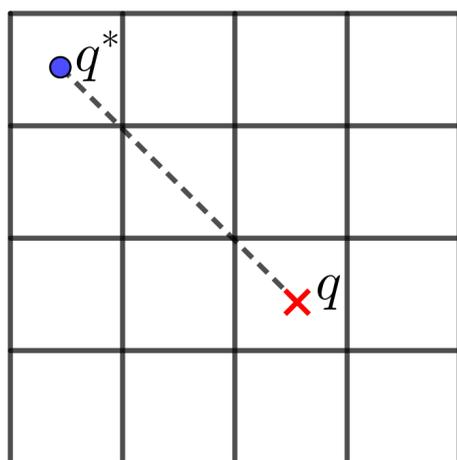
对每个 $\square_w \in S$, 若 $\|q - \text{rep}_w\| < r$, 更新 $r = \|q - \text{rep}_w\|$ 及 $\hat{q} = \text{rep}_w$

对每个 $\square_w \in S$, 若 $r > (1 + \epsilon) \cdot (\|q - \text{rep}_w\| - \text{diam}(\square_w))$, 将 \square_w 放入 A_i

- Claim:** 若 $\text{diam}(\square_w) \leq \epsilon/2 \cdot r^*$, 则 $|A_i| = \emptyset$

即验证: 任何 $\square_w \in S$ 都被剪枝, 无法放入 A_i

可推出 $\leq \epsilon/2 \cdot r$, 因为根据定义 $r \geq r^*$



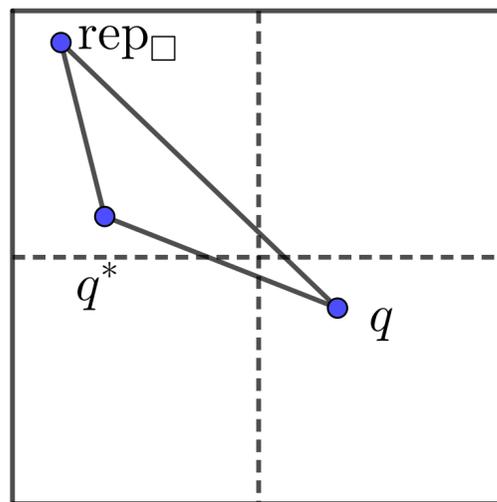
直觉: $\text{diam}(\square)$ 很小 ($\leq O(\epsilon) \cdot r^*$) 说明近似的足够精确了, 没有任何点加进去可以显著改进精度了

$$\begin{aligned} & (1 + \epsilon) \cdot (\|q - \text{rep}_w\| - \text{diam}(\square_w)) \\ & \geq (1 + \epsilon) \cdot (r - \text{diam}(\square_w)) \\ & \geq (1 + \epsilon) \cdot (1 - \epsilon/2) \cdot r \\ & \geq r \end{aligned}$$

* 正确性分析

总是能找到 $r \leq (1 + \epsilon)r^*$

- 因为算法在 $\text{diam}(\square) < \epsilon/2 \cdot r^*$ 后就停止了，只需考虑 $\text{diam}(\square) \geq \epsilon/2 \cdot r^*$
- 若停止前的某个时刻 $r \leq (1 + \epsilon) \cdot r^*$ ，则已经达到了要求
- 否则， $r > (1 + \epsilon) \cdot r^*$ ，那么性质1说明 q^* 所在格子依然存活
- 当 $\text{diam}(\square) \approx \epsilon/2 \cdot r^*$ ，我们有 $r \leq r^* + \text{diam}(\square) \leq (1 + \epsilon)r^*$



根据三角形不等式：

$$r \leq \|q - \text{rep}\| \leq \|q - q^*\| + \|\text{rep} - q^*\| \leq r^* + \text{diam}(\square)$$

* 运行时间分析

当 $\text{diam}(\square_w) > \epsilon/2 \cdot r^*$ 时 $|A_i|$ 的上界

- 回忆算法步骤:

对每个 $\square_w \in S$, 若 $\|q - \text{rep}_w\| < r$, 更新 $r = \|q - \text{rep}_w\|$ 及 $\hat{q} = \text{rep}_w$

对每个 $\square_w \in S$, 若 $r > (1 + \epsilon) \cdot (\|q - \text{rep}_w\| - \text{diam}(\square_w))$, 将 \square_w 放入 A_i

- 若 $\square_w \in S$ 放入 A_i , 则 $\|q - \text{rep}_w\| < r/(1 + \epsilon) + \text{diam}(\square_w)$

- Claim:** $r/(1 + \epsilon) + \text{diam}(\square_w) \leq O(1/\epsilon) \cdot \text{diam}(\square_w)$

- 先假定Claim正确: 则若 \square_w 放入了 A_i , $\|q - \text{rep}_w\| \leq O(1/\epsilon) \cdot \text{diam}(\square_w)$

* 推出 $|A_i|$ 上界

- 因为 $\|q - \text{rep}_w\| \leq O(1/\epsilon) \cdot \text{diam}(\square_w)$
- 所以所有放入 A_i 的 rep_w 都在以 q 为球心 $O(1/\epsilon) \cdot \text{diam}(\square_w)$ 为半径的 d 维球里

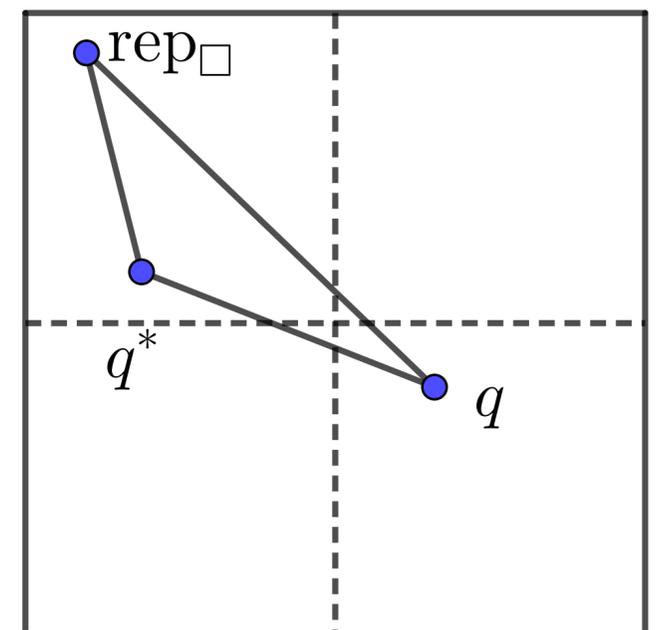
性质：在一个 $\overbrace{R \times \dots \times R}^d$ 的区域内，有至多 $\left(\frac{R}{l}\right)^d$ 个 $\overbrace{\ell \times \dots \times \ell}^d$ 正方形

- 因此有 $|A_i| \leq (O(1/\epsilon))^{-d}$ 至多有 $\log \Delta$ 个 $|A_i|$ 非零，因此总复杂度 $(O(1/\epsilon))^d \cdot \log \Delta$

* 验证Claim

Claim: $r/(1 + \epsilon) + \text{diam}(\square_w) \leq O(1/\epsilon) \cdot \text{diam}(\square_w)$

- 假定算法执行中总有 $\text{diam}(\square_w) > \epsilon/2 \cdot r^*$ 根据性质2: 否则会导致 $|A_i| = 0$, 不需要考虑
- 若 $r \leq (1 + \epsilon)r^*$, 则 $r/(1 + \epsilon) + \text{diam}(\square_w) \leq O(1/\epsilon) \cdot \text{diam}(\square_w)$
- 否则 q^* 所在方格未被剪枝, 有 $r \leq r^* + \text{diam}(\square_w)$, 推出 $\|q - \text{rep}_w\| \leq O(1/\epsilon) \cdot \text{diam}(\square_w)$
- 综上 $\|q - \text{rep}_w\| \leq O(1/\epsilon) \cdot \text{diam}(\square_w)$



* 其他可能的剪枝策略?

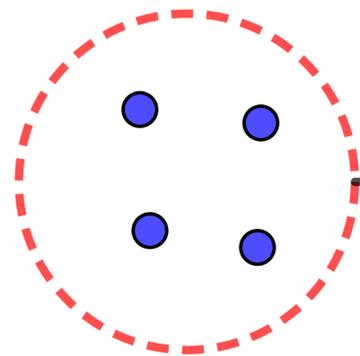
- 有同学提出: 改成用 $r \leq (1 + \epsilon) \cdot \text{dist}(q, \square_i)$ 来断定 \square_i 应该剪掉
- 与我们的剪枝条件 $r \leq (1 + \epsilon)(\|q - \text{rep}_i\| - \text{diam}(\square_i))$ 如何比较?
 - 是否有一者推出另一者?
 - 思考: 求 $\text{dist}(q, \square_i)$ 需要多少时间? 正确性? 总运行时间?

3. 考虑点对的离散化: WSPD

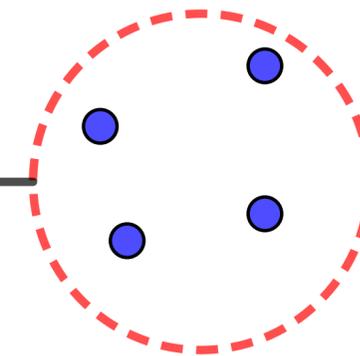
WSPD

直观讨论

- 输入点集 $P \subseteq [\Delta]^d$, 设 $|P| = n$
- 考虑所有点对的距离 (共有 $O(n^2)$ 对), 试图将点对距离离散化
- 直觉: 设有 $S, T \subseteq P$ 两组点, 且他们距离很远
 - 那么可以把 S 和 T 分别“缩点”, 从而将 $S \times T$ 上的距离等于同一个值



即使不显式缩点, 左右两边点的距离相对来说都差不多; 例如左右两圆直径=1, 但相距100



WSPD

定义

- 一个 ϵ -WSPD是一系列 $\{\{A_i, B_i\}\}_i$, 使得

点集之间的距离等于最近点的距离, 即

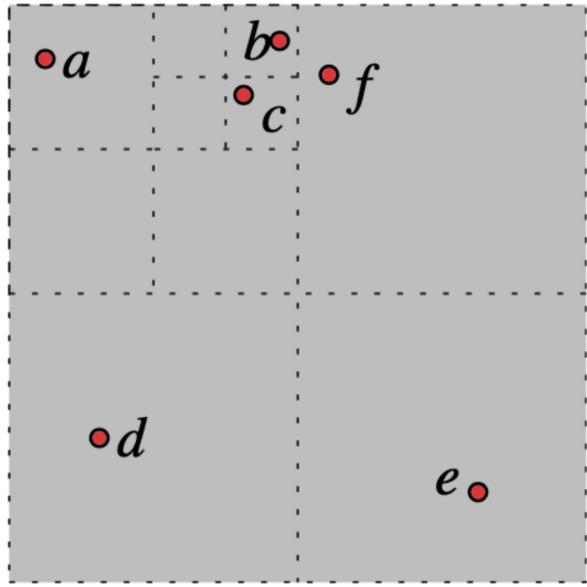
$$\text{dist}(S, T) := \min_{x \in S, y \in T} \|x - y\|$$

- $\forall i, A_i, B_i \subseteq P$ 且 $\max(\text{diam}(A_i), \text{diam}(B_i)) \leq \epsilon \cdot \text{dist}(A_i, B_i)$ 即两组点 A_i, B_i 距离很远

- $\bigcup_i A_i \times B_i = P \times P$, 即任何 $p \neq q \in P$ 可以找到某个 i 使得 $p \in A_i, q \in B_i$

可以理解为对 $O(n^2)$ 个距离对的划分





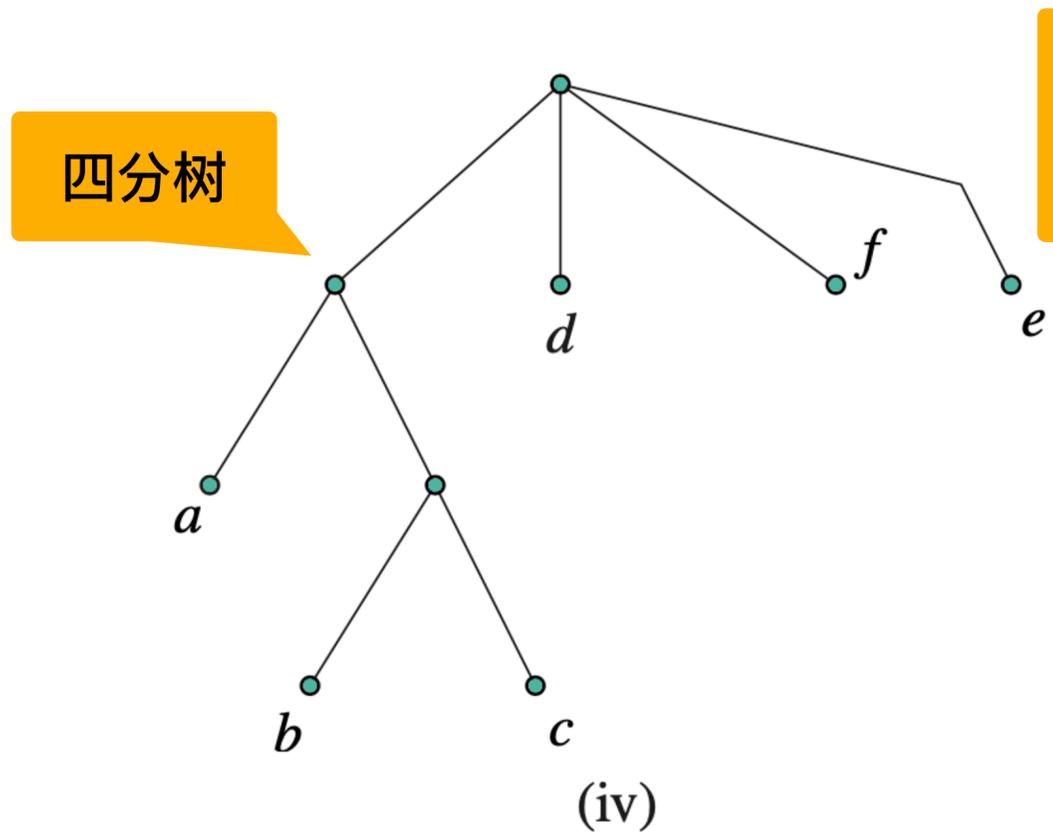
(i)

- $A_1 = \{d\}, B_1 = \{e\}$
- $A_2 = \{a, b, c\}, B_2 = \{e\}$
- $A_3 = \{a, b, c\}, B_3 = \{d\}$
- $A_4 = \{a\}, B_4 = \{b, c\}$
- $A_5 = \{b\}, B_5 = \{c\}$
- $A_6 = \{a\}, B_6 = \{f\}$
- $A_7 = \{b\}, B_7 = \{f\}$
- $A_8 = \{c\}, B_8 = \{f\}$
- $A_9 = \{d\}, B_9 = \{f\}$
- $A_{10} = \{e\}, B_{10} = \{f\}$

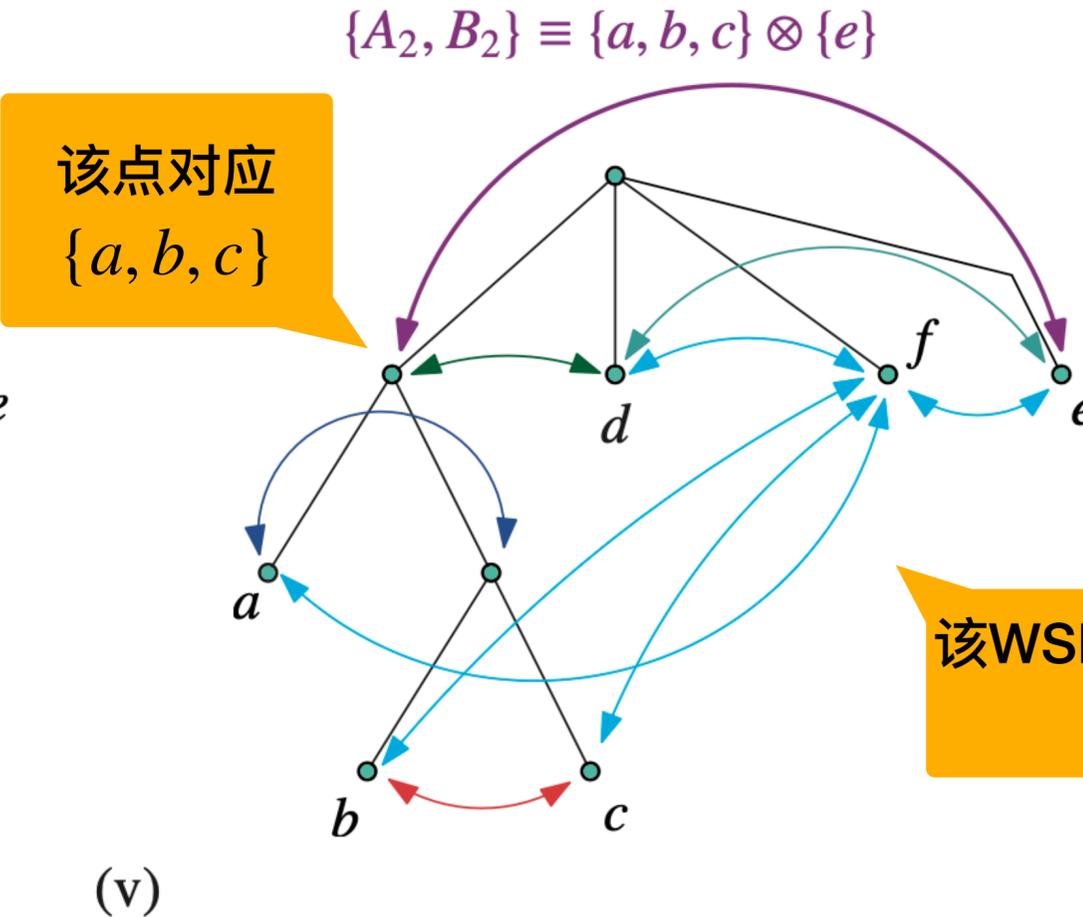
(ii)

$$\mathcal{W} = \left\{ \begin{array}{l} \{A_1, B_1\}, \\ \{A_2, B_2\}, \\ \{A_3, B_3\}, \\ \{A_4, B_4\}, \\ \{A_5, B_5\}, \\ \{A_6, B_6\}, \\ \{A_7, B_7\}, \\ \{A_8, B_8\}, \\ \{A_9, B_9\}, \\ \{A_{10}, B_{10}\} \end{array} \right\}$$

(iii)



(iv)



(v)

该WSPD可用四分树高效表示

基于四分树的WSPD：完整算法

WSPD用四分树的 \square 表示，总复杂度 $O(n\epsilon^{-d} \log \Delta)$ ，所得WSPD有 $O(n\epsilon^{-d} \log \Delta)$ 项

全局对 $\Delta = [\Delta]^d$ 调用WSPD(\square, \square)

WSPD(\square_u, \square_v)

定义： $\delta(\square_u) = \text{diam}(\square_u)$ 若 $|\square_u \cap P| \geq 2$ ，否则 $\delta(\square_u) = 0$

if ($u = v$ and $\delta(\square_u) = 0$) return // 单点 \square 不要配对到自己身上

if ($\delta(\square_u) < \delta(\square_v)$) swap u, v

if ($\delta(\square_u) \leq \epsilon \cdot \text{dist}(\square_u, \square_v)$)

直接看 \square 而不是 $P \cap \square$

return $\{ \{ \square_u, \square_v \} \}$

运行至此， $\delta(\square_u) \geq \delta(\square_v)$ ，因此 $\{ \square_u, \square_v \}$ 确实满足WSPD条件，又因为算法会尝试一直递归到底，因此**确保最后输出的是合法WSPD**

return $\bigcup_{i=1}^{2^d} \text{WSPD}(\square_{u_i}, v)$

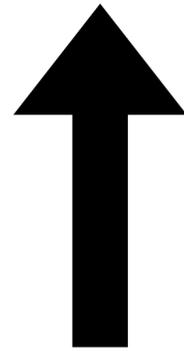
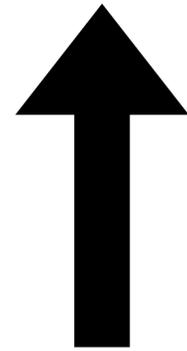
\square_{u_i} 是 \square_u 的子 \square

* 算法性能分析

基于压缩四分树

运行时间也是这个量级

- 总结论：算法输出的WSPD有 $O(n\epsilon^{-d} \log \Delta)$ 个 $\{A_i, B_i\}$



- 对每个四分树方格 \square_v ，考虑所有加入WSPD的 $\{\square_u, \square_v\}$ 的集合 S_v
- **Claim:** $|S_v| \leq O(\epsilon^{-O(d)})$ 精确写法应该是 $O(\epsilon^{-1})^{O(d)}$ ，我们之后都采用这种简略的不严格写法
- 利用Claim: 该四分树共 $O(n \log \Delta)$ 个（非空） \square ，WSPD共有 $O(n\epsilon^{-d} \log \Delta)$ 项

* 算法性质

初始 $\square = [\Delta]^d$

- **性质:** 除初始调用 (\square, \square) 外, 任何 (\square_u, \square_v) 递归调用必是从 $(\bar{p}(\square_u), \square_v)$ 和 $(\square_u, \bar{p}(\square_v))$ 之一产生的

$\hat{p}(\square)$ 表示 \square 的父格子

- 可以推出: 任何递归调用 (\square_u, \square_v) 满足 \square_u 和 \square_v 差至多一层
- 原因: 算法可以产生新递归调用的地方仅有两处

• if $(\delta(\square_u) < \delta(\square_v))$ swap u, v

不改变 \square_u, \square_v , 只是交换

• return $\bigcup_{i=1}^{2^d} \text{WSPD}(\square_{u_i}, v)$

下次调用中仅 \square_u 被 \square_u 的某个子 \square 替代, 但 \square_v 不改变

* 验证Claim

Claim: 对任何方格 \square_v , $|S_v| \leq O(\epsilon^{-O(d)})$

对四分树方格 \square_v , S_v 为算法中所有加入WSPD的 \square_u 的集合

• 辅助结论: $\forall \square_u \in S_v, \text{dist}(\square_u, \square_v) \leq O(1/\epsilon) \cdot \delta(\square_u)$

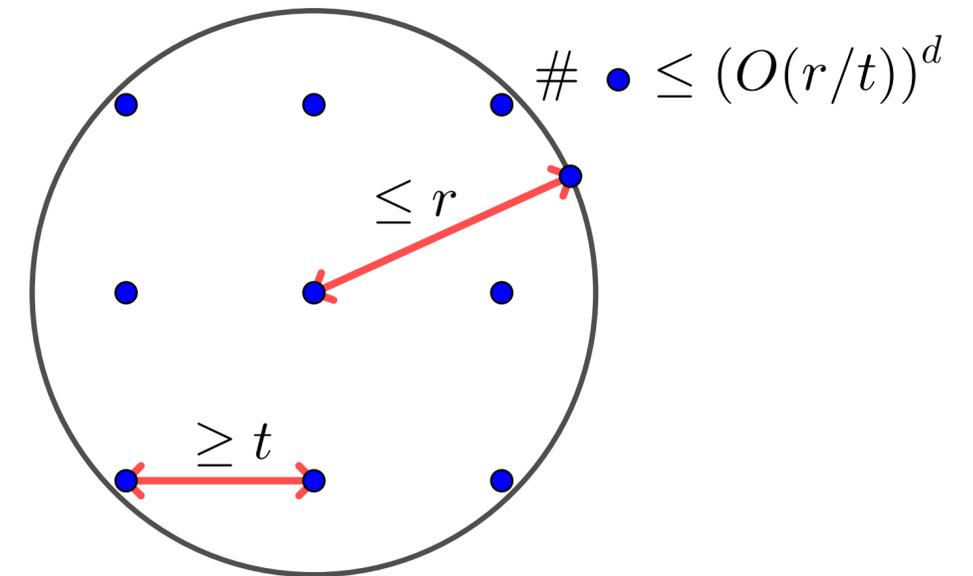
下页验证, 现在假定成立

• 若 $\delta(\square_u)$ 固定 = t , 则满足 $\text{dist}(\square_u, \square_v) \leq O(1/\epsilon) \cdot \delta(\square_u)$ 的 \square_u 有几个?

• 任何两个这样 \square_u 的格点中心距离至少 $\Omega(t)$

• 所有这些格点中心都在距离 \square_v 的 $O(1/\epsilon) \cdot t$ 范围内

• 因此至多有 $\epsilon^{-O(d)}$ 个! 利用欧氏空间基本性质



• $\delta(\square_u) = t$ 的值有至多2种可能性, 因此 $|S_v| \leq O(\epsilon^{-O(d)})$

算法性质: \square_u 和 \square_v 至多差一层

* 验证辅助结论

$$\forall \square_u \in S_v, \text{dist}(\square_u, \square_v) \leq O(1/\epsilon) \cdot \delta(\square_u)$$

- 回忆算法步骤: $\text{if } (\delta(\square_u) \leq \epsilon \cdot \text{dist}(\square_u, \square_v)) \text{ return } \{ \{ \square_u, \square_v \} \}$
- 考虑一个递归调用 (\square_u, \square_v) 令上述if成立 (即此 $\square_u \in S_v$)
- 此次 (\square_u, \square_v) 递归调用一定是从 $(\bar{p}(\square_u), \square_v)$ 和 $(\square_u, \bar{p}(\square_v))$ 之一产生的
- 且当次调用上述if判定为false

利用算法性质

否则就会return且不可能产生 (\square_u, \square_v) 调用

* 验证辅助结论 cont.

$$\forall \square_u \in S_v, \text{dist}(\square_u, \square_v) \leq O(1/\epsilon) \cdot \delta(\square_u)$$

另一种情况($\bar{p}(\square_v), \square_u$)类似

if ($\delta(\square_u) \leq \epsilon \cdot \text{dist}(\square_u, \square_v)$)

- 设(\square_u, \square_v)是从($\bar{p}(\square_u), \square_v$)调用产生的, 且“if”判断为false, 也就是

$$\delta(\bar{p}(\square_u)) > \epsilon \cdot \text{dist}(\bar{p}(\square_u), \square_v)$$

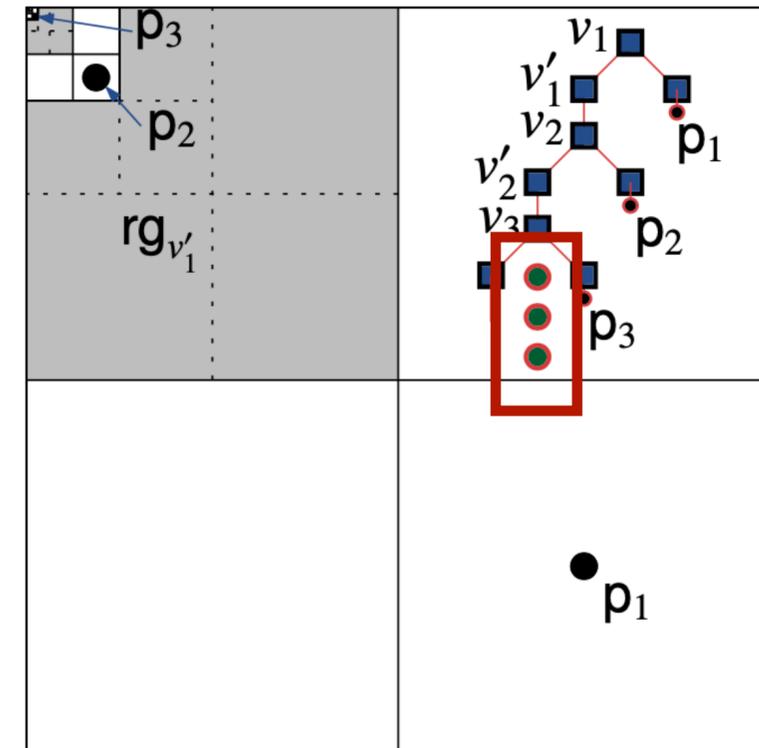
- 这可以推出: $\text{dist}(\square_u, \square_v) \leq O(1/\epsilon) \cdot \delta(\bar{p}(\square_u))$ 进一步 $\leq O(1/\epsilon) \cdot \delta(\square_u)$

- 三角形不等式推出 $\text{dist}(\square_u, \square_v) \leq \text{dist}(\bar{p}(\square_u), \square_v) + \delta(\bar{p}(\square_u))$

- 代入 $\text{dist}(\bar{p}(\square_u), \square_v) < O(1/\epsilon) \cdot \delta(\bar{p}(\square_u))$ 后得出结论

* 其他改进：压缩四分树

- 之前的四分树有 $O(n \log \Delta)$ 个 \square ：仅能得到 $n \cdot O(\epsilon)^{-d} \log \Delta$ 大小的 WSPD
- 我们可以将四分树“压缩”，使得只有 $O(n)$ 个 \square



- $\log \Delta$ 哪来的？如何压缩？
 - 树中度 = 2 的点构成一个长 $O(\log \Delta)$ 路径
 - 具体：设路径起点 \square_u 终点 \square_v ，则设置 \square_u 的唯一孩子为 \square_v ，中间全都删除

同样的WSPD算法依然适用，且可以达到 $O(n \cdot \epsilon^{-d})$ 项，但分析更加复杂

WSPD应用：精确求最近点对

- 这是一个 $O(n \cdot 2^d \log \Delta)$ 时间的精确算法：

- 构造 $\epsilon = 0.5$ 的WSPD $\{\{A_i, B_i\}\}_i$ $O(n \cdot 2^d \log \Delta)$ 时间

- 对所有的 $\{A_i, B_i\}$ ，若 $|A_i| = |B_i| = 1$ ，记录 A_i 和 B_i 中唯一点的距离

- 在所有距离中的最近点对就是答案

$O(n \cdot 2^d \log \Delta)$ 时间

因此算法不会
遗漏最近点对

- 原因：设最近点对是 (p, q) 且 $p \in A_i$ ， $q \in B_i$ ，则 $|A_i| = |B_i| = 1$

若有多余点 x 在 A_i ，则容易看出 $\|p - x\| < \|p - q\|$ ，矛盾



作业：最近点对精确求解

- <http://cssyb.openjudge.cn/23hw8/>
- Deadline: 4月19日

* WSPD也可以用来近似直径

- 算法:

- 构造 $\epsilon/2$ -WSPD

- 对每个 $\{A_i, B_i\}$ 对, 分别找 rep_i^A , rep_i^B , 然后求 $\|\text{rep}_i^A - \text{rep}_i^B\|$

- 求所有 $\|\text{rep}_i^A - \text{rep}_i^B\|$ 里面最大的就是直径的 $(1 - \epsilon)$ -近似

- 设 q, s 两点是直径的两个端点 ($\|q - s\| = \text{diam}$), 且设 $q \in A_i, s \in B_i$

WSPD性质

$$\|\text{rep}_i^A - \text{rep}_i^B\| \geq \text{dist}(A_i, B_i) \geq \|q - s\| - \text{diam}(A_i) - \text{diam}(B_i) \geq (1 - \epsilon) \cdot \|q - s\|$$

根据定义

三角形不等式; 类似最近邻剪枝条件的场景

* WSPD应用：求 $(1 + \epsilon)$ -Spanner

$$t \geq 1$$

- t -Spanner是数据集 $P \subseteq [\Delta]^d$ 的一个欧氏子图 H
 - 欧氏子图：点集还是 P ，边集是 $P \times P$ 的子集，边权重是端点间的欧氏距离
 - **t-Stretch**：要求对任何 $x, y \in P$ 有 $\|x - y\| \leq \text{dist}_H(x, y) \leq t \cdot \|x - y\|$
 - 追求： H 满足t-Stretch条件，且边数尽量少
- 可以作是欧氏完全图的稀疏“压缩”，后续可使用稀疏图上的算法来解决欧氏问题

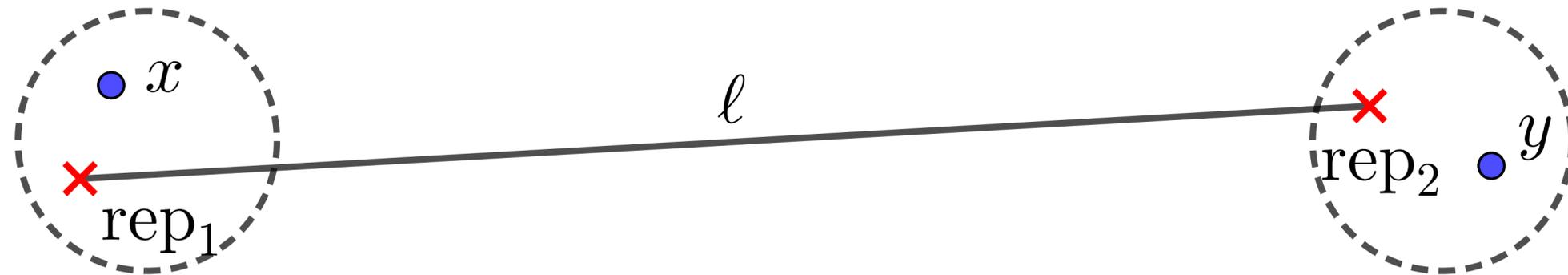
dist_H 代表 H 上的最短路距离

* Spanner算法

- 算法：先求 $\epsilon/4$ -WSPD，再对每个 A_i, B_i 都找代表点 rep_1 和 rep_2 连边
- 因此总共是 $O(n\epsilon^{-d} \log \Delta)$ 条边

* Spanner算法的分析

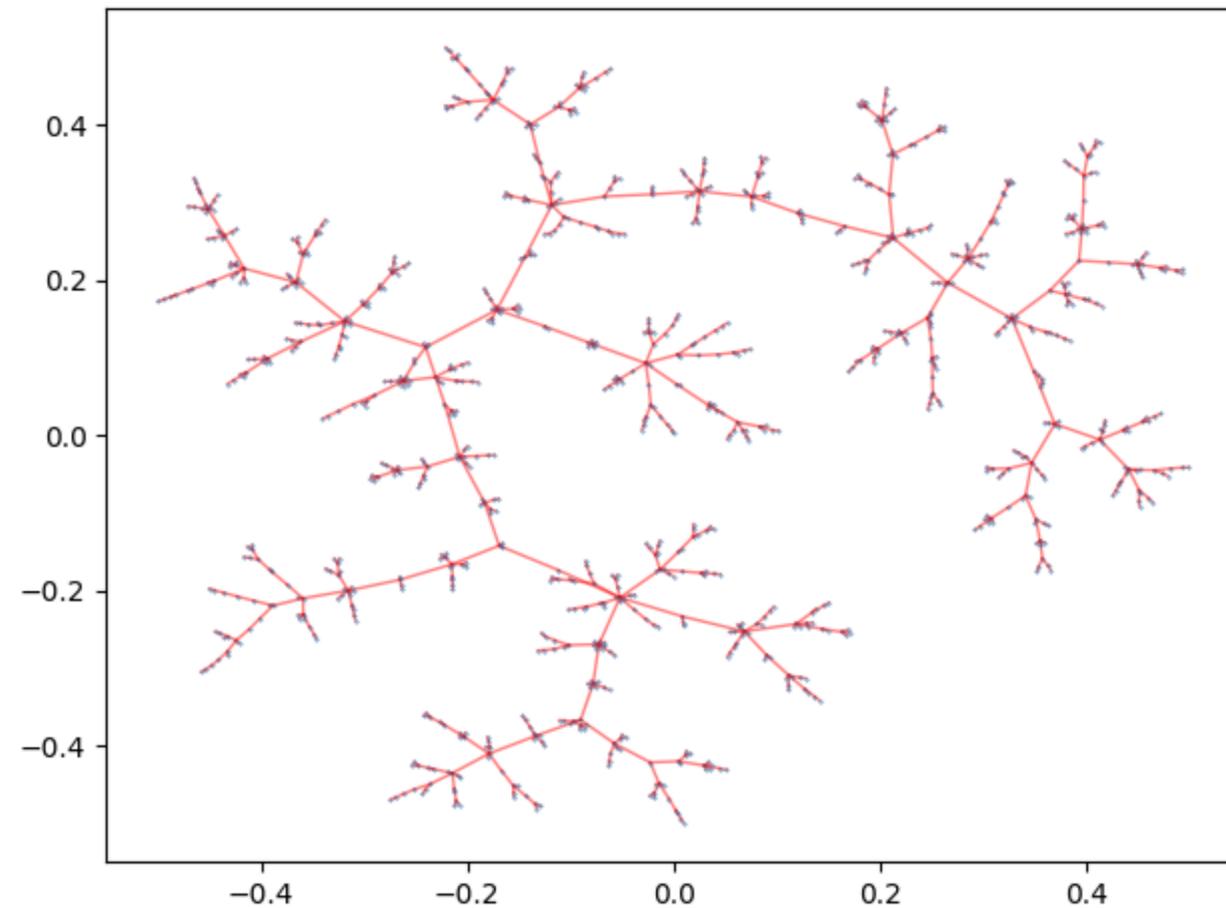
归纳法：按照 x, y 距离 D 递增进行归纳，假设是若 $\|x - y\| \leq D$ 则 x, y 满足 $(1 + \epsilon)$ -Stretch



- 考虑一对点 x 和 y ，设 $x \in A_i$ ， $y \in B_i$ ，且 rep_1 和 rep_2 是代表点、其连边被加入H
- 观察： $\text{dist}(x, y) \geq \text{dist}(A_i, B_i) \geq 1/\epsilon \cdot \text{diam}(A_i) \geq 1/\epsilon \cdot \text{diam}(B_i)$
- 推出： $\ell \leq \text{dist}(A_i, B_i) + \text{diam}(A_i) + \text{diam}(B_i) \leq (1 + \epsilon/2) \cdot \text{dist}(x, y)$
- 再利用归纳假设，
$$\begin{aligned} \text{dist}_H(x, y) &\leq \text{dist}_H(x, \text{rep}_1) + \text{dist}_H(y, \text{rep}_2) + \ell \\ &\leq 2(1 + \epsilon) \cdot \epsilon/4 \cdot \ell + \ell \leq (1 + \epsilon) \cdot \text{dist}(x, y) \end{aligned}$$

* 欧氏最小生成树

- 图的最小生成树 (MST) : 最小权的使所有点连通的子图
- 欧氏数据点集 P 的最小生成树: 定义在 $P \times P$ 这个欧氏完全图上的MST



* $(1 + \epsilon)$ -Spanner \rightarrow $(1 + \epsilon)$ -近似最小生成树

- 算法:

- 利用刚刚提到的算法基于WSPD构造 P 的 $(1 + \epsilon)$ -spanner H

- 在 H 上求MST, 则该MST是一个 P 上的 $(1 + \epsilon)$ -近似的MST

- 解释:

- 任何 P 上的MST可以对应于 H 上一个 $(1 + \epsilon)$ 倍权重的子连通图 H'

- 因此 H 上的MST比 H' 要小, 而 H 上的MST也是 P 上的生成树合法解

$$\text{MST}(H) \leq (1 + \epsilon) \cdot \text{MST}(P)$$

H 是 $P \times P$ 的子图, 因此 H 中的边都在 P 的欧氏图中

* 利用随机平移的四分树构造Tree Embedding

- Tree embedding: 将数据集 P 从 \mathbb{R}^d 映射到一棵树 $T(V, E)$ 上, 其中 $P \subseteq V$
 - 旨在用树上的距离来“近似”/“代替”欧氏距离
 - 首先要求 $\forall x, y \in P, \text{dist}_T(x, y) \geq \|x - y\|$
- 主要性能衡量叫Distortion:

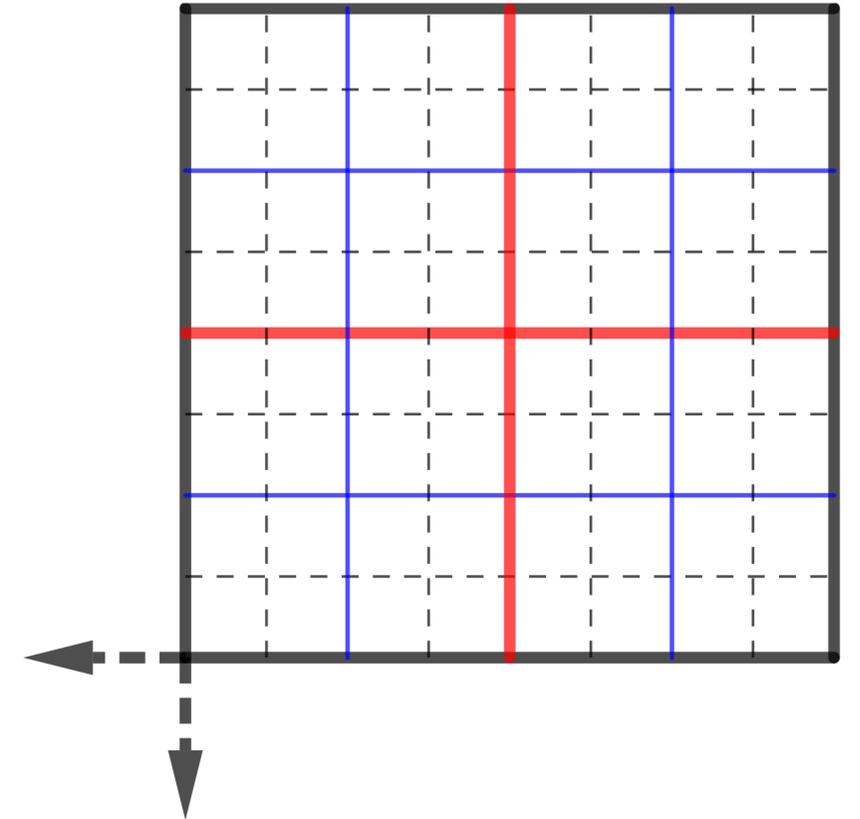
是 ≥ 1 的量, 越小越好

$$\max_{x \neq y \in P} \frac{\text{dist}_T(x, y)}{\|x - y\|}$$

* 构造算法和结论

$[2\Delta]$ 主要为了随机平移后依然包含 $[\Delta]$

- 构造四分树 $\text{BuildTree}(\square = [2\Delta]^d, P)$
- 选取 $[-\Delta, 0]^d$ 上的均匀随机向量 v ，将整个四分树平移 v
- 构造：每个 \square 是一个树节点，数据点都在树的叶子上，从深度 i 连接到深度 $i + 1$ 的边赋予 $\sqrt{d} \cdot 2^i$ 的权值
- 对于 $x, y \in P$ ，对应的就是树的叶子，距离 $\text{dist}_T(x, y)$ 就是树上的距离
- 结论： $\forall x, y \in P, \text{dist}_T(x, y) \geq \|x - y\|, \mathbb{E}[\text{dist}_T(x, y)] \leq O(dh) \cdot \|x - y\|$



Distortion 是随机的

h 是四分树的高度

* 对比/应用

- 劣势：近似比大 $O(dh)$

- 优势：

最小生成树、聚类、匹配等各类问题；
很多NP-hard问题在树上仍有有效算法

- 因为每对点期望距离都能保持，因此广泛适用于目标函数是点距离和的问题
- 树上算法性能较好，可解决问题多，有树算法就有欧氏空间近似算法
- 没有 2^d ，对于高维也相对适用

* 分析

- 考虑最小的同时包含 x, y 的 \square ，这对应树的LCA，也直接对应树上距离计算

- 若该 \square 边长 2^i ，那么树上距离是 $2 \cdot \sqrt{d} \sum_{j=0}^i 2^j = \Theta(\sqrt{d} \cdot 2^i)$
对一路上的边权求和

- 验证： $\text{dist}_T(x, y) \geq \|x - y\|$

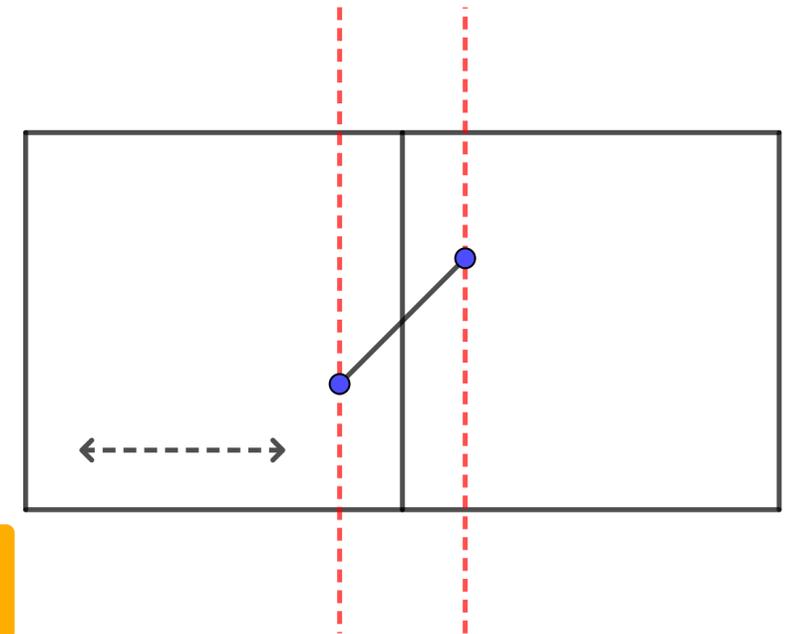
- 设该方格边长 2^i ，那么 $\|x - y\| \leq \sqrt{d} 2^i \leq \sum_{j=0}^i \sqrt{d} 2^j$

* 分析 (cont.)

验证: $\mathbb{E}[\text{dist}_T(x, y)] \leq dh \cdot \|x - y\|$

- 考虑最小的同时包含 x, y 的 \square , 这对应树的 LCA, 也直接对应树上距离计算

若该 \square 边长 2^i , 那么树上距离是 $2 \cdot \sqrt{d} \sum_{j=0}^i 2^j = \Theta(\sqrt{d} \cdot 2^i)$



$$\mathbb{E}[\text{dist}_T(x, y)] = \sum_{i=0}^h \Pr[\text{LCA is at level } i] \cdot \Theta(\sqrt{d} 2^i)$$

也就是 x, y 形成的线段跨越了 level- $(i + 1)$ 的 \square

$$\leq \sum_{i=0}^h \Pr[x, y \text{ separated at level-}(i + 1)] \cdot O(\sqrt{d} 2^i)$$

“线段跨 hypercube”仅当存在某一维跨越 hypercube

$$\leq \sum_{i=0}^h \Pr[\exists \text{ coordinate } j, \text{ such that } x, y \text{ are separated}] \cdot O(\sqrt{d} 2^i)$$

$$\begin{aligned}
& \sum_{i=0}^h \Pr[\exists \text{ coordinate } j, \text{ such that } x, y \text{ are separated}] \cdot O(\sqrt{d}2^i) \\
& \leq \sum_{i=0}^h \sum_{j=1}^d \Pr[x, y \text{ are separated at coordinate } j] \cdot O(\sqrt{d}2^i) \\
& \leq \sum_{i=0}^h O\left(\sum_{j=1}^d |x_j - y_j| / 2^i \cdot \sqrt{d}2^i\right) \leq \sum_{i=0}^h O(d)\|x - y\| \leq O(dh) \cdot \|x - y\|
\end{aligned}$$

$\forall a \in \mathbb{R}^d, \|a\|_1 \leq \sqrt{d} \cdot \|a\|_2$

