程序设计实习（实验班-2024春）距离度量及其计算：欧氏距离

授课教师：姜少峰

助教：冯施源 吴天意

Email: shaofeng.jiang@pku.edu.cn

欧氏距离

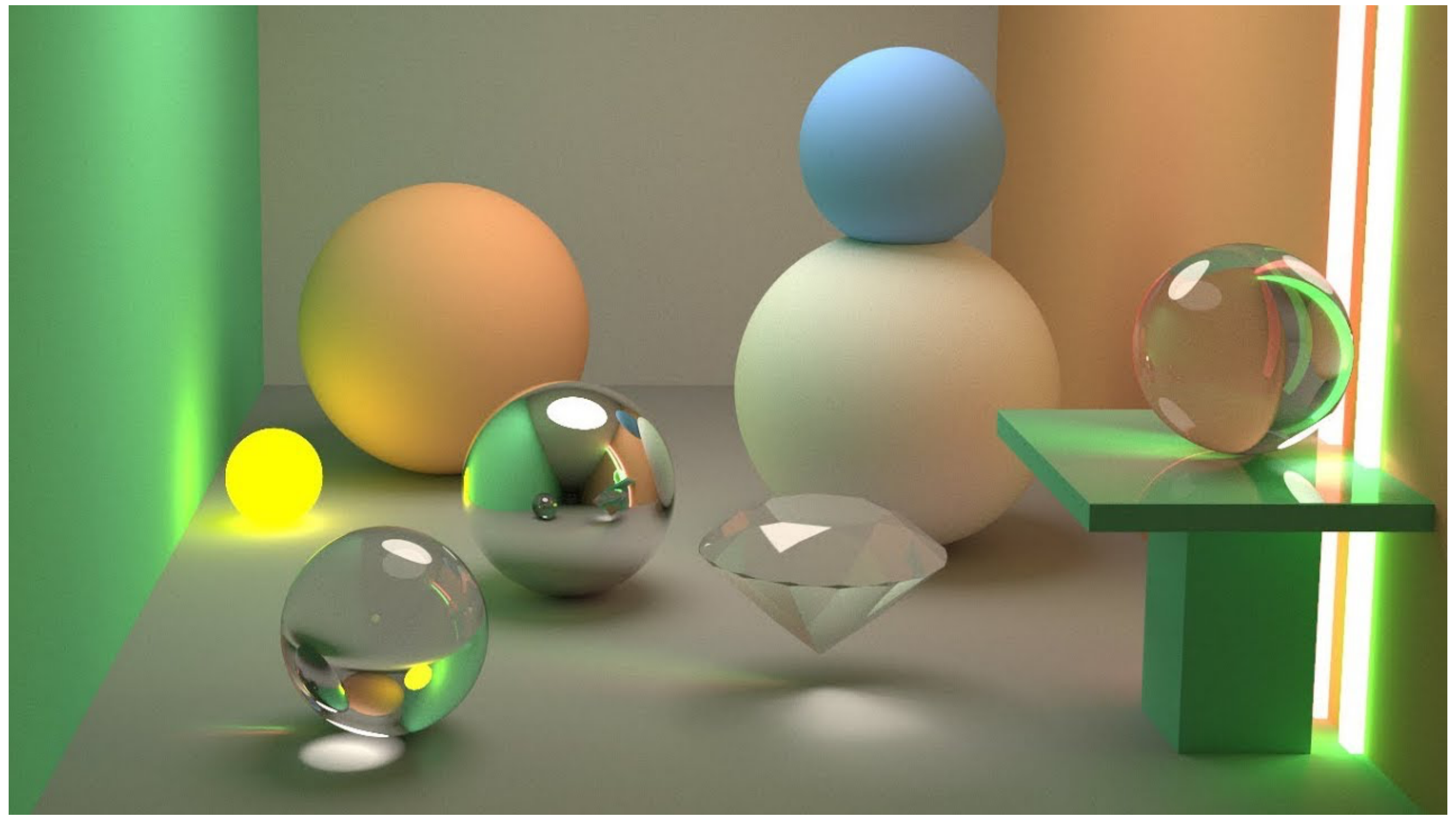
- 是最常见的向量的距离度量
- $$\forall x, y \in \mathbb{R}^d, \ \ \|x - y\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$
- $\|\cdot\|_2$是ℓ_2-norm（2-范数）
- 欧氏空间有三角形不等式：$\forall x, y, z \in \mathbb{R}^d, \ \ \|x - y\| + \|y - z\| \geq \|x - z\|$

为何重要?

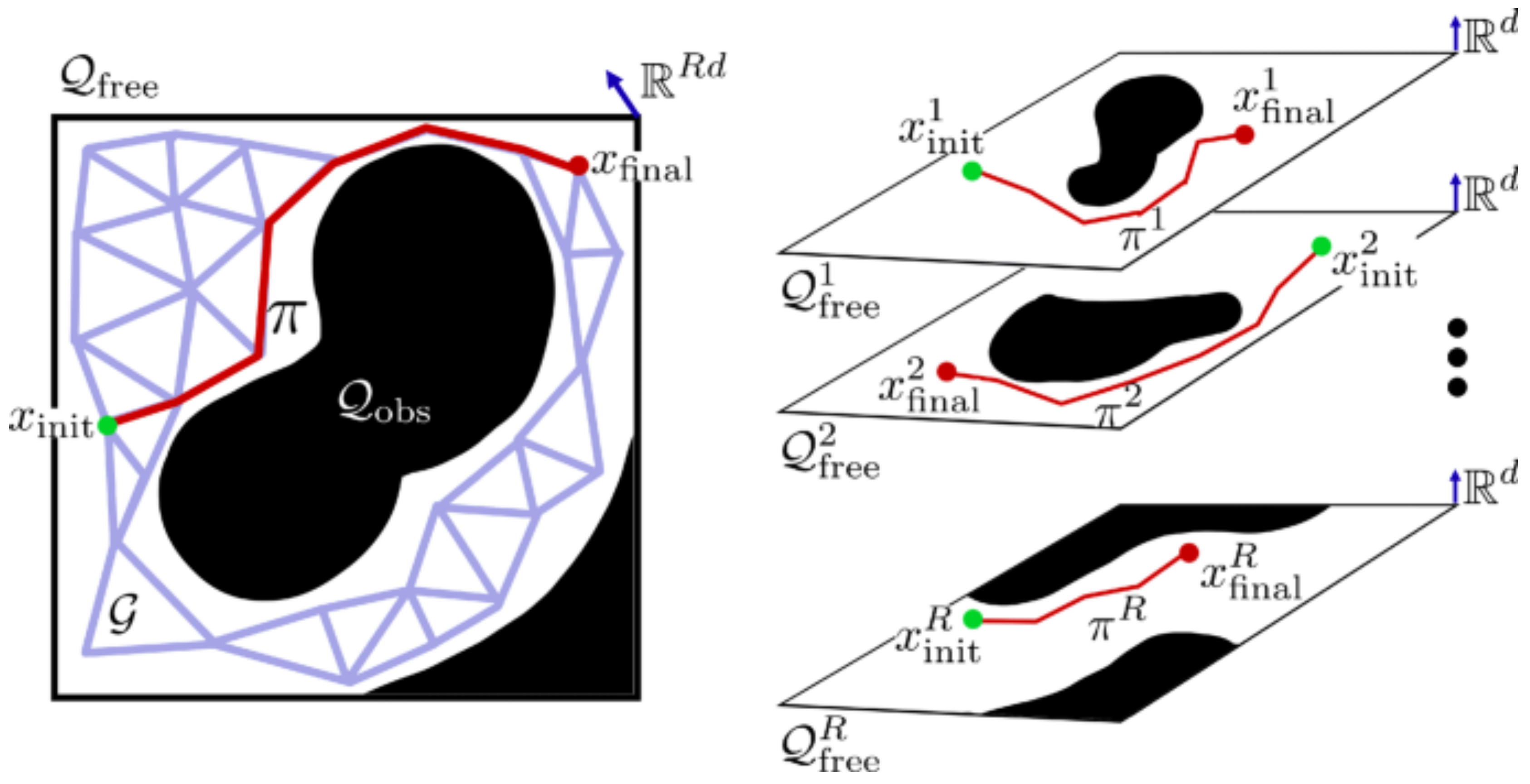
- 我们生存在3维欧氏空间里，距离自然定义为欧氏距离
- computer graphics, computer vision等也主要在欧氏空间内研究

VR

渲染

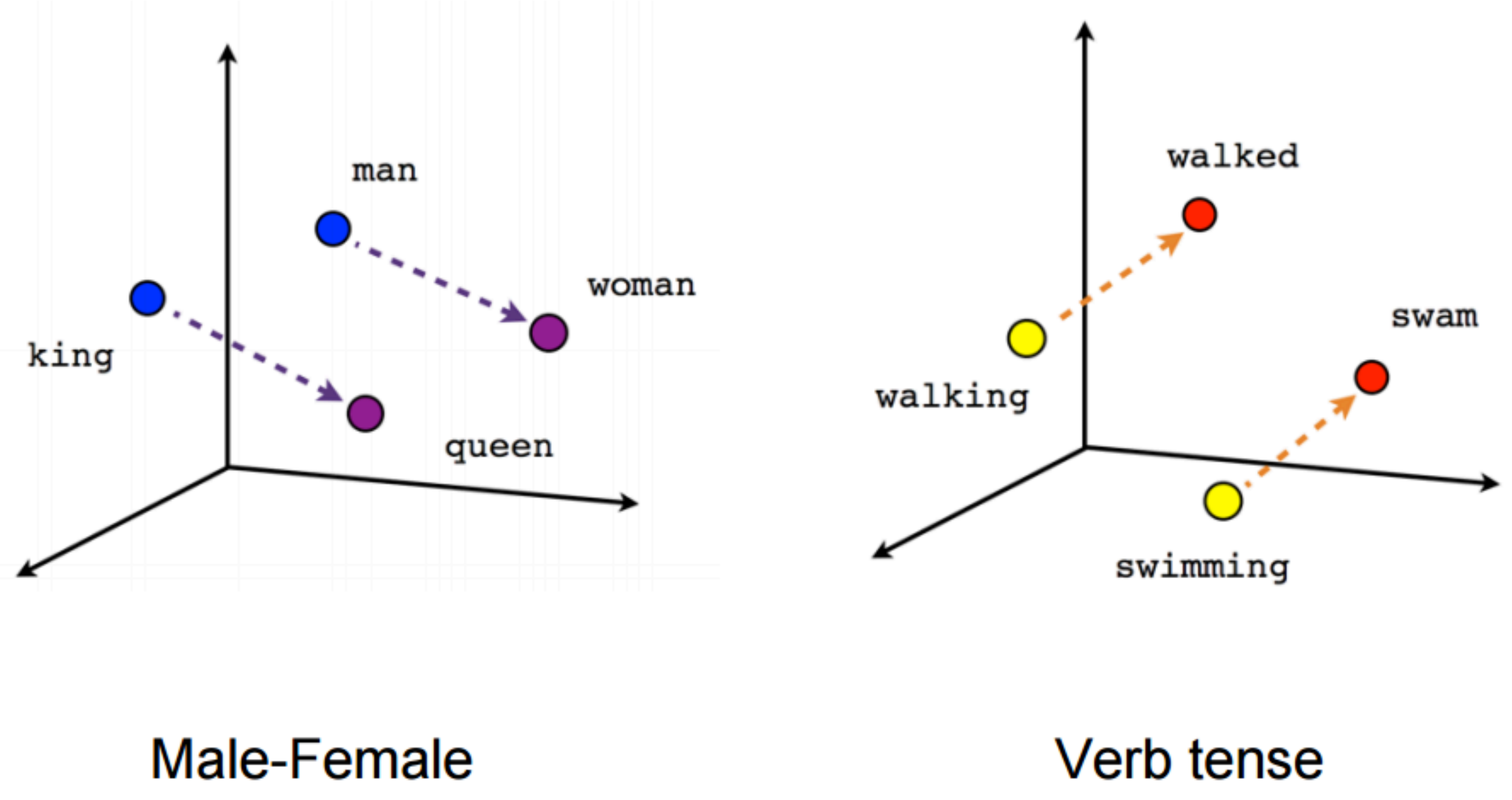


机器人运动规划

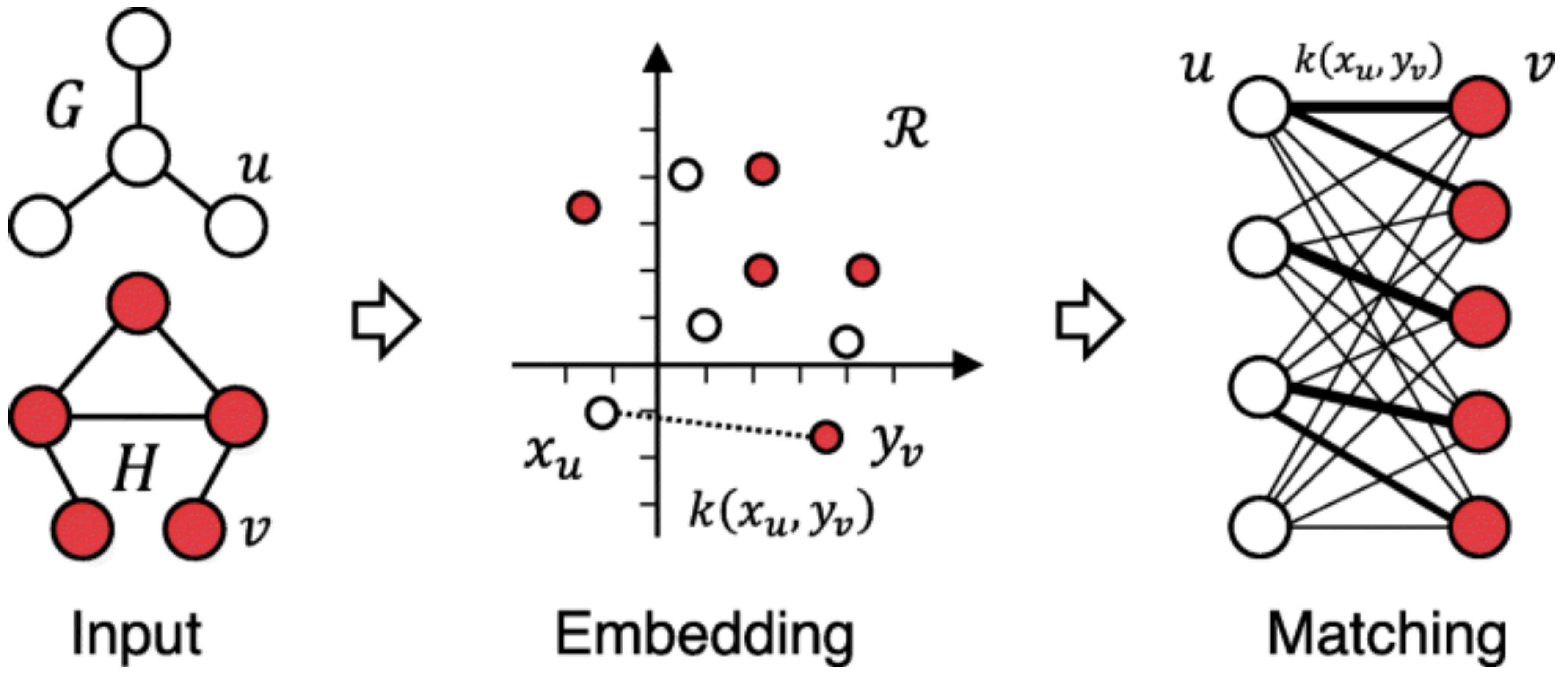


其他数据类型也经常映射到欧氏空间来处理

Word Embedding



Graph Embedding



Kernel与欧氏空间

- Kernel是一种一般的将数据点映射到高维欧氏空间的方法
- 设数据集是U，那么kernel function的形式是$K: U \times U \to \mathbb{R}_+$
- 一个函数K是“合法”kernel，如果存在$\varphi: U \to \mathbb{R}^t$ 使得$\forall x, y \in U$,

$$K(x, y) = \langle \varphi(x), \varphi(y) \rangle$$

φ将数据点映射成（高维）向量
$K(x, y)$就是对应的内积，是一种相似度度量
可以类比cosine similarity

- 例如Gaussian kernel：$K(x, y) = \exp(-\|x - y\|_2^2)$
- 一般只给出K，φ并不显式给出，在不直接知道φ的情况下进行欧氏空间上的计算

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$[\Delta] = \{1,\ldots,\Delta\}$范围的数据点，即数据都在$[\Delta]^d$内

例如考虑unsigned int，$\Delta = 2^{32}$

- 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
- 针对一般高维空间：Tree embedding

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：$(x, x+\ell] \times (y, y+\ell]$

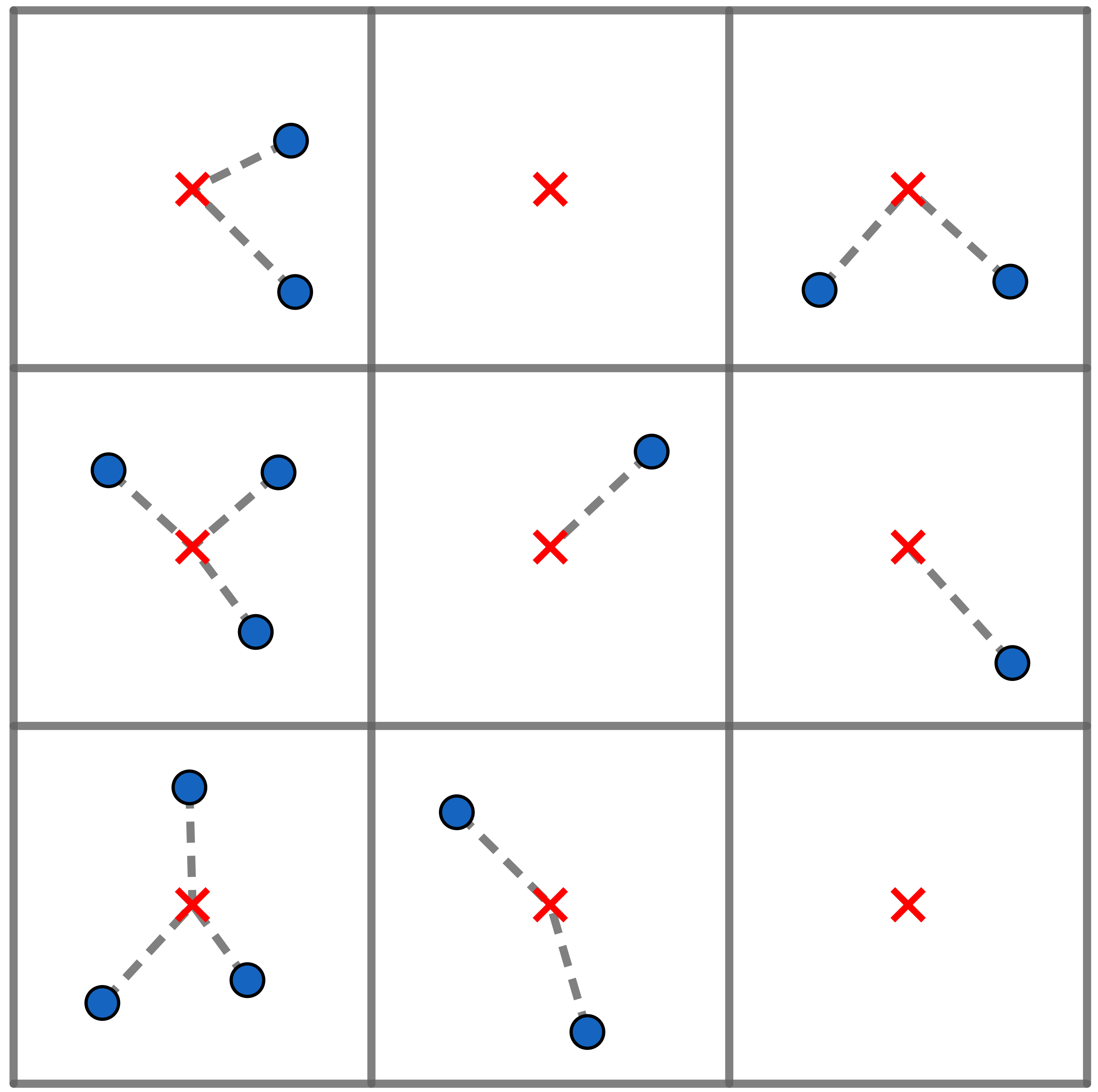
- 将整个空间划分成$\ell \times \ell$的小正方形
- 将数据点“round”到所属格子的格点中心
  - 由于正方形互不相交，这个rounding是唯一的

d维是$\sqrt{d}\ell$

- 性质：每个点移动距离 $\leq \sqrt{2}\ell$

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

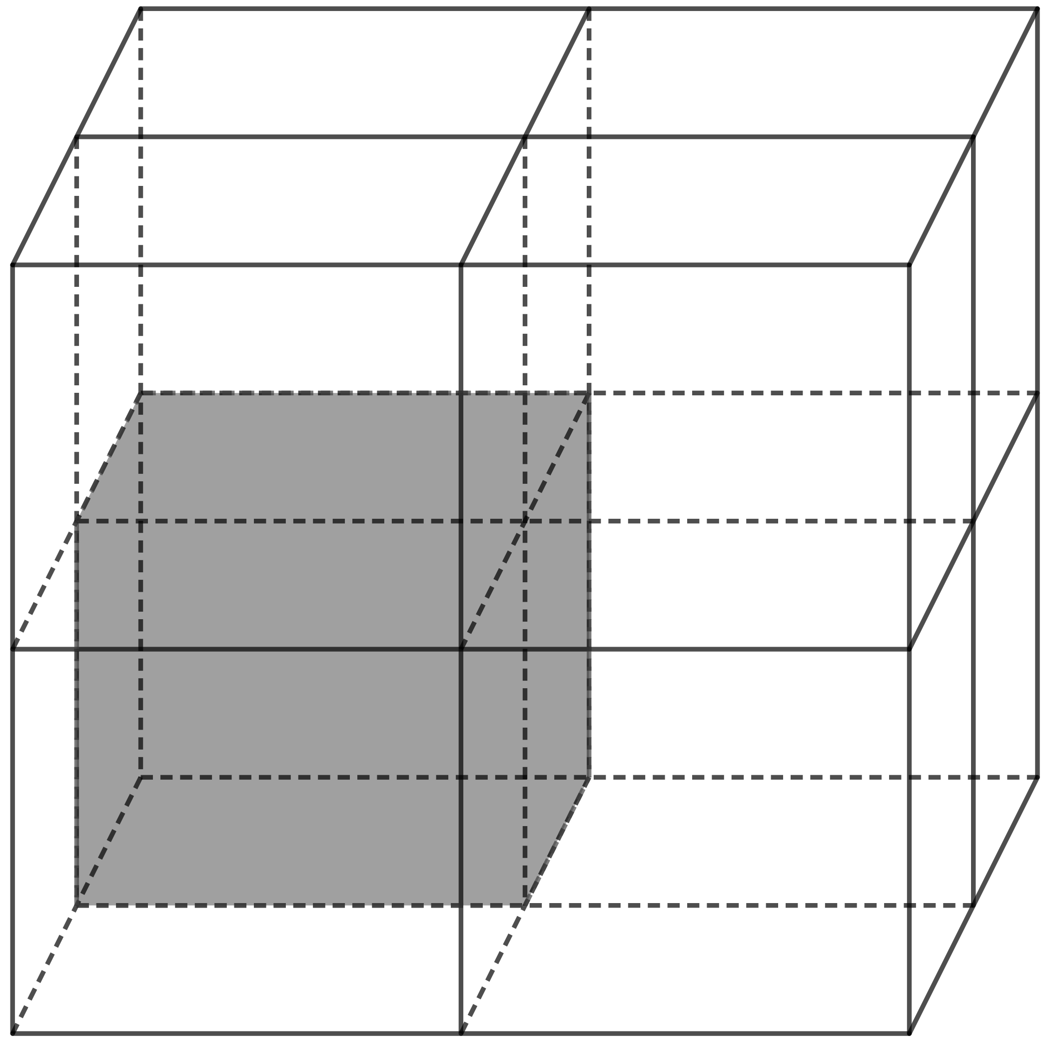
- 性质：在一个$R \times R$的区域内，有至多$\left(\dfrac{R}{\ell}\right)^2$个$\ell \times \ell$正方形

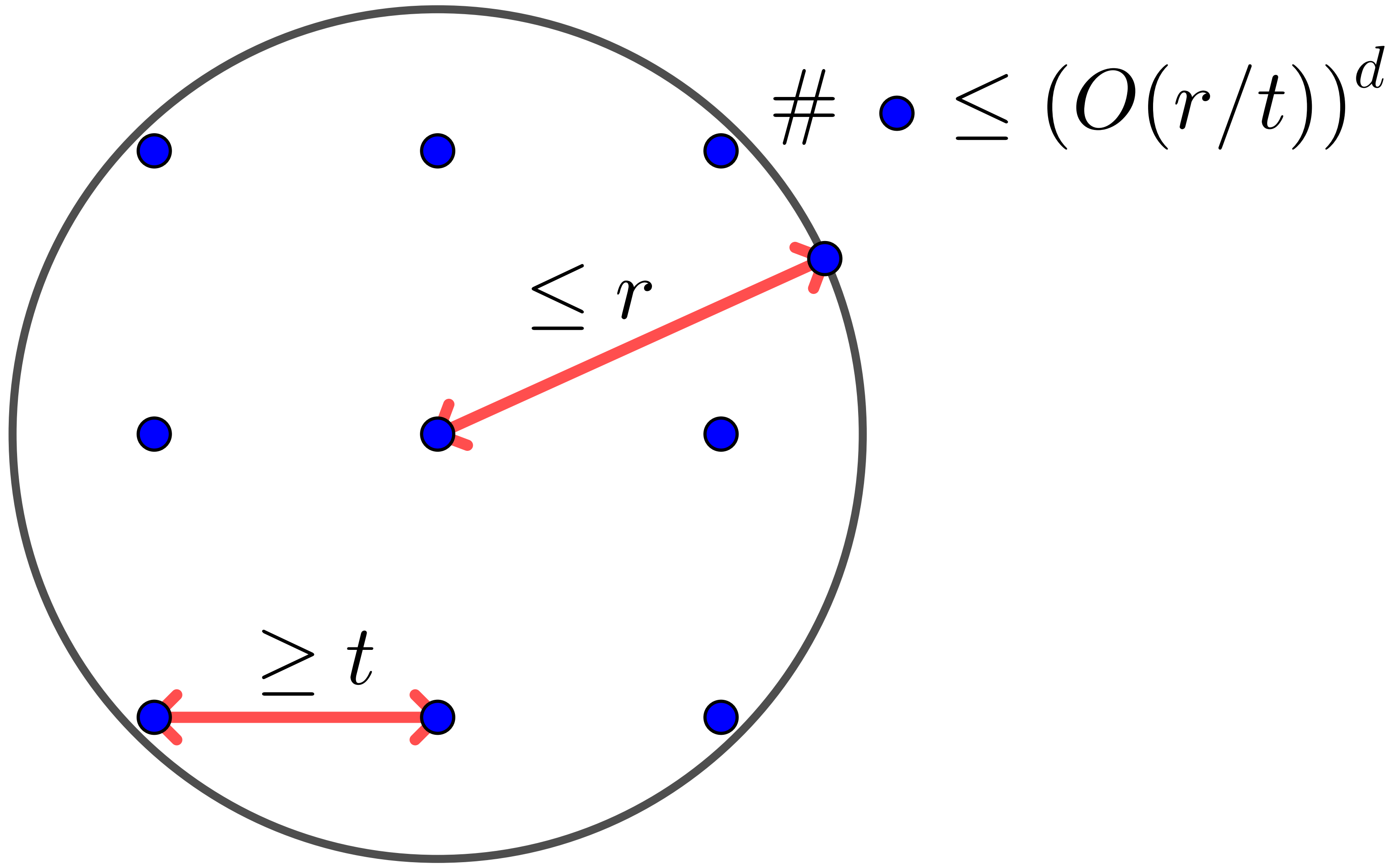


欧氏空间基于体积的根本性质

性质：在一个$\overbrace{R\times\ldots\times R}^{d}$的区域内，有至多$\left(\frac{R}{l}\right)^d$个$\overbrace{\ell\times\ldots\times\ell}^{d}$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$\mathrm{diam}(P) := \max_{x,y \in P} \|x - y\|_2$
- 求$\mathrm{diam}(P)$

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$1/2 \cdot \mathrm{diam}(P) \le T \le \mathrm{diam}(P)$

- 先O(n)时间找一个直径的2-近似值T

事实上可以是方格内任意确定点

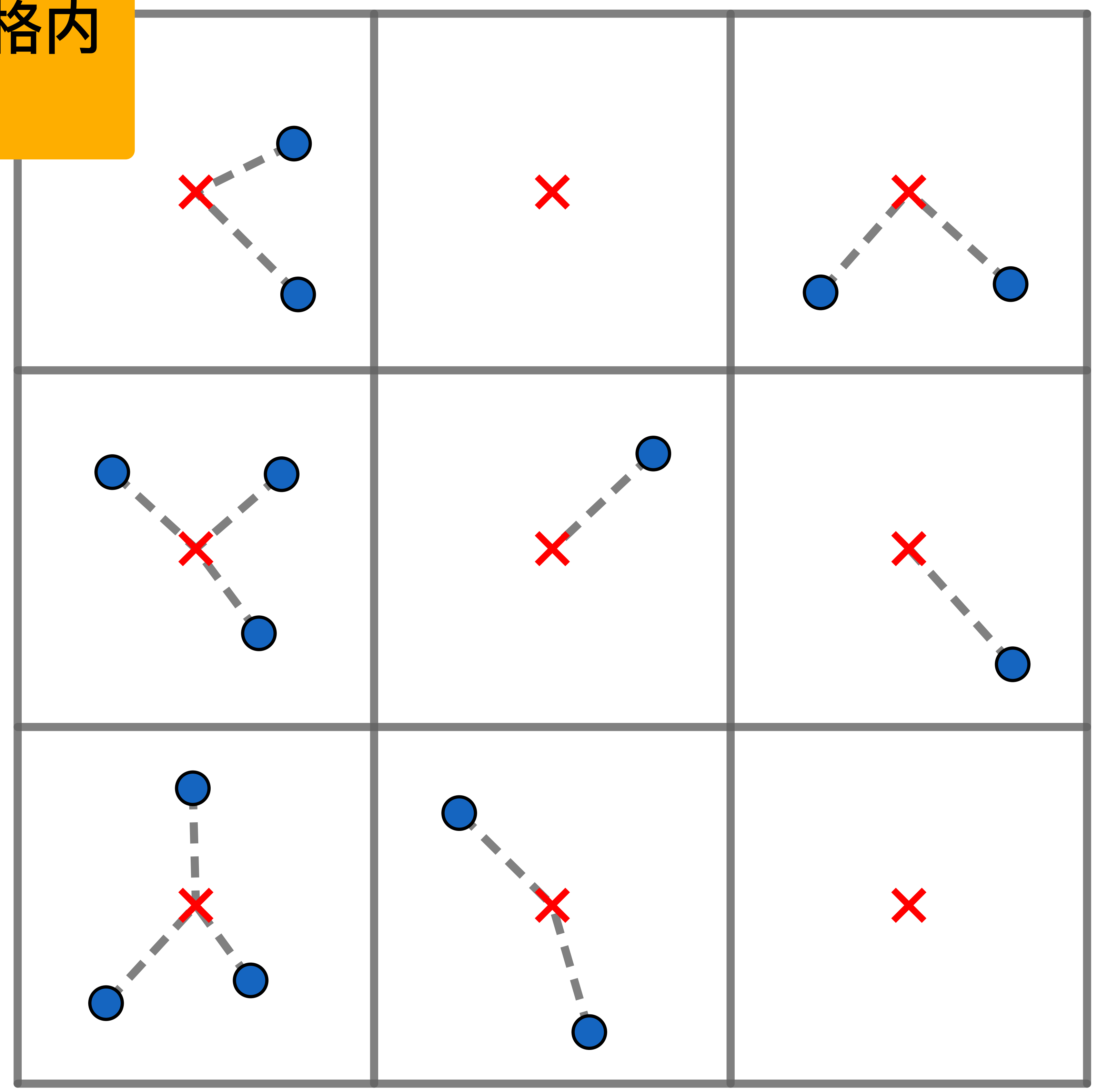
- 作$\ell := \epsilon \cdot T/\sqrt{2}$的网格，并round到中心
  - 因此每个点移动了$\le \sqrt{2}\ell \le \epsilon \cdot \mathrm{diam}(P)$
- 因此新点集P'满足

可在$O(nd\log n)$时间构造P'

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

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

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



总复杂度：$O(nd\log n) + \epsilon^{-O(d)}$

一些实现细节

- 设把点都round为方格坐标较小的一角，且方格都是$\ell \times \ell$的
  - 那么给定$x = (x_1, \ldots, x_d)$，左上角的坐标是$(\lceil x_1/\ell \rceil, \ldots, \lceil x_d/\ell \rceil)$

$O(d)$时间

- 直接round完后的到的P'会有很多的重复点，需要去重
  - 例如：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/24hw10/
- 分值：1分
- Deadline: 4月10日

* 类似算法：最小包围球的coreset

最小包围圆的圆心

- 输入$P = \{x_1, \ldots, x_n\} \subset \mathbb{R}^2$，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 $f(S, c) \geq (1 - \epsilon) \cdot f(P, c)$

$f(S, c) \leq f(P, c)$是根据定义总是成立的

- 算法：

OPT代表最优解的值

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

  - $O(n)$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$|P'| \leq \mathrm{poly}(1/\epsilon)$

  - 得到的点集P'即为ϵ-coreset

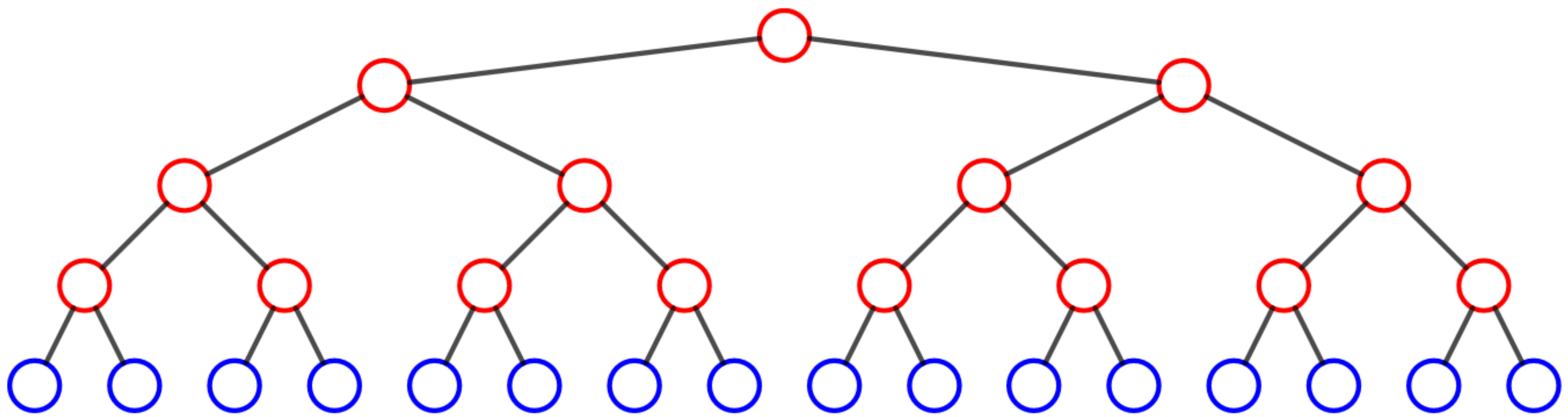
每个点移动距离 $\leq \epsilon \cdot \mathrm{OPT}$，因此对任意$c \in \mathbb{R}^2$，$\|x - c\|_2$也仅会改变$\epsilon \cdot \mathrm{OPT}$

Merge-and-reduce框架

- Coreset重要性质：可合并，即coreset(A) ∪ coreset(B)是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

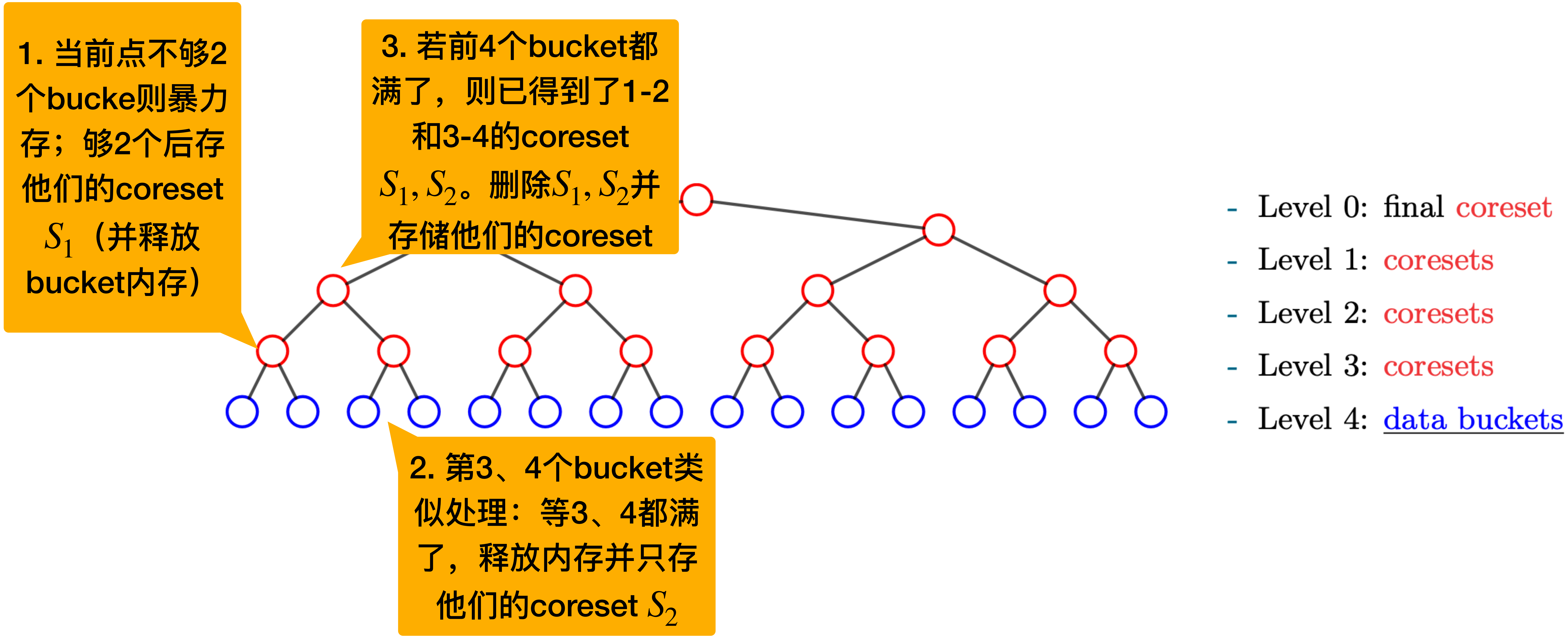
可以很容易的支持数据流算法、并行算法

- $O(\log n)$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



- 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

如果不追求数据流?

Welzl's Algorithm (1991)

- 存在一个期望$O((d+1)(d+1)!n)$时间的精确解算法
- 核心观察：设C_i为前i个点的MEB，若$p_i \notin C_{i-1}$则必有$p_i \in C_i$
- 如何利用观察：尝试求C_{i-1}，然后若$p_i \notin C_{i-1}$，递归求包含p_i的前i - 1点MEB
  - “包含p_i”的MEB可以递归求解
  - 最后如果边界上要求必须包括的点到达$d+1$则可以停止递归

Welzl's Algorithm

meb(P, R)

if $P = \emptyset$ or $|R| = d + 1$ **return** the ball with R on the boundary

else

choose $p \in P$ u.a.r., let $D :=$ meb($P \backslash \{p\}, R$)

if $p \notin D$ **return** meb($P \backslash \{p\}, R \cup \{p\}$)

return D

高维?

- 存在一个$O(nd/\epsilon)$时间的$1 + \epsilon$近似算法
- Smaller core-sets for balls, Bâdoiu and Clarkson, SODA 03

2. 递归格点离散化：四分树

递归划分：四分树

以二维为例

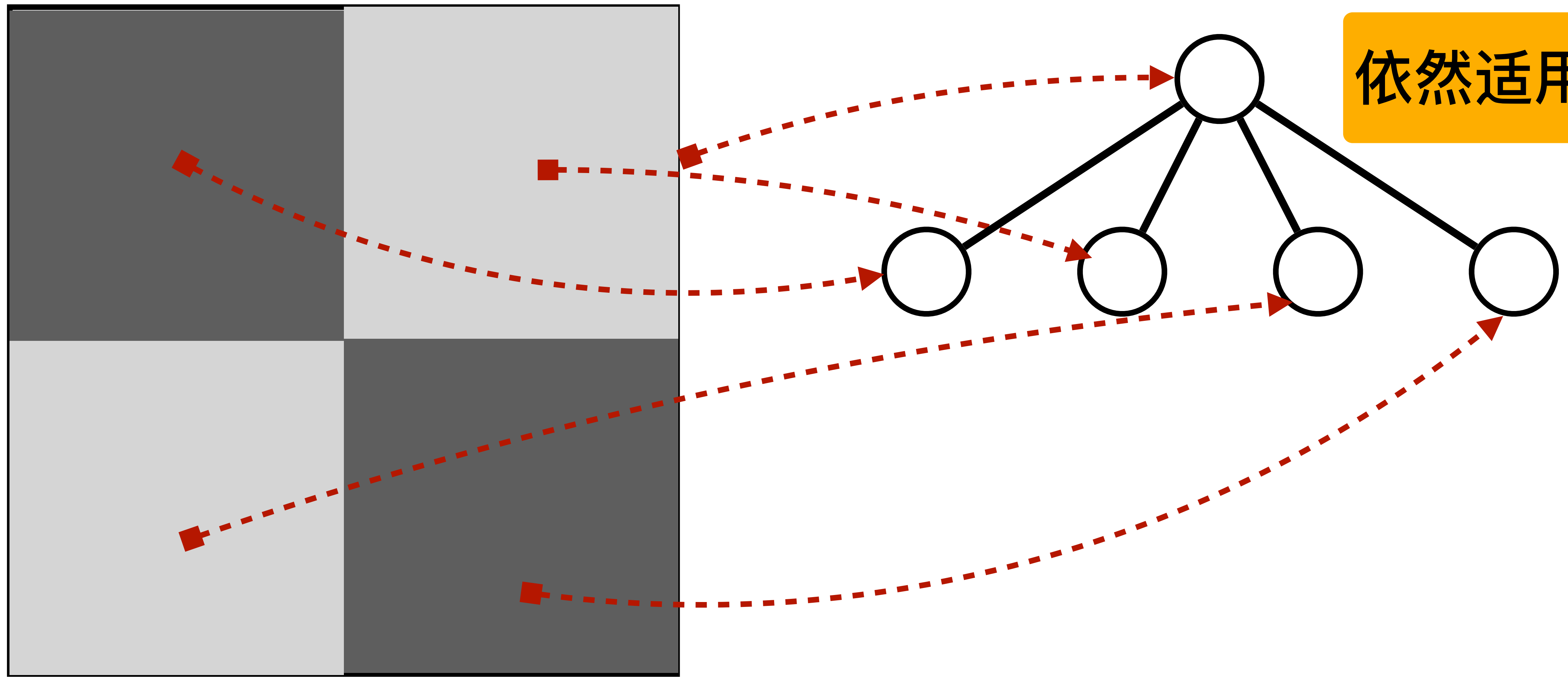
d维叫“d维四分树”

d维是从$[\Delta]^d$划分

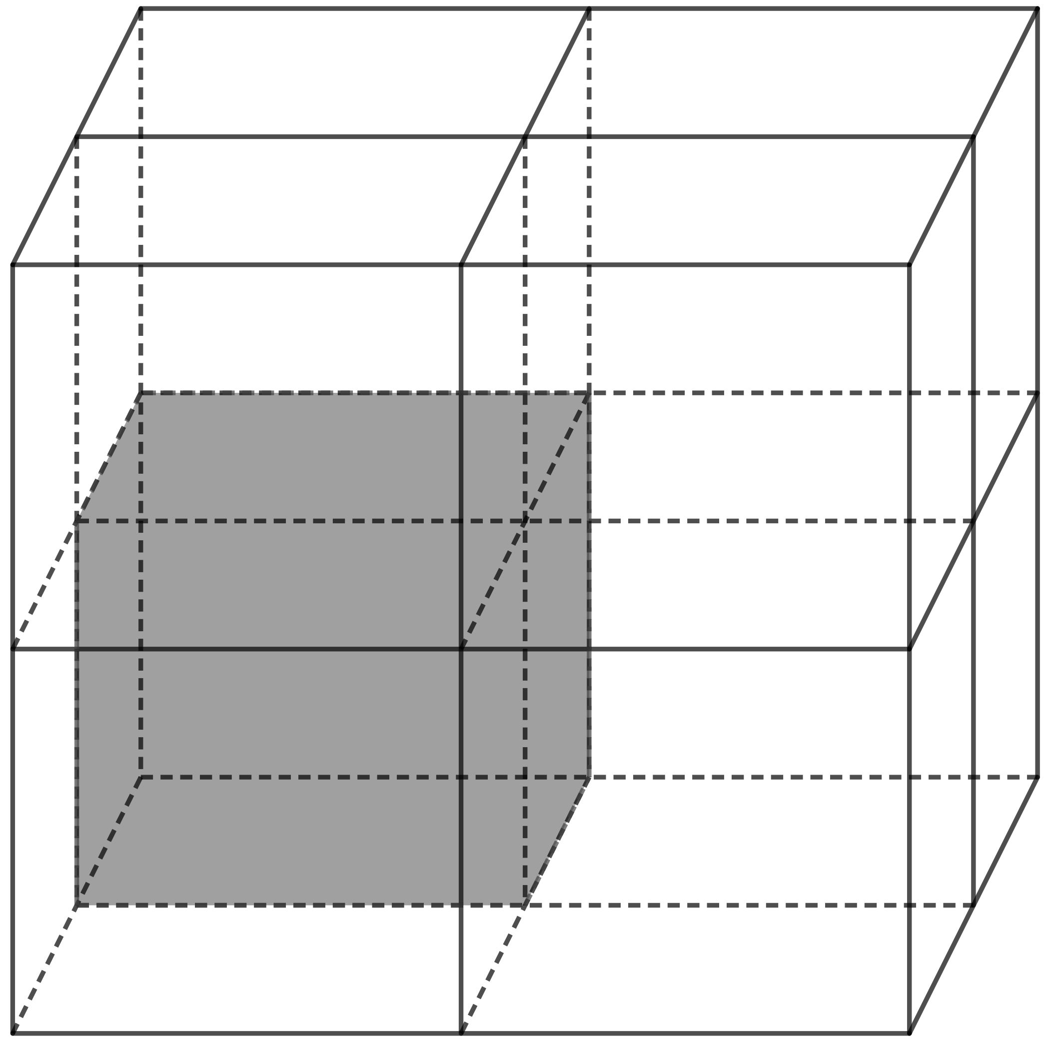
- 从最大的$\Delta\times\Delta$的正方形开始

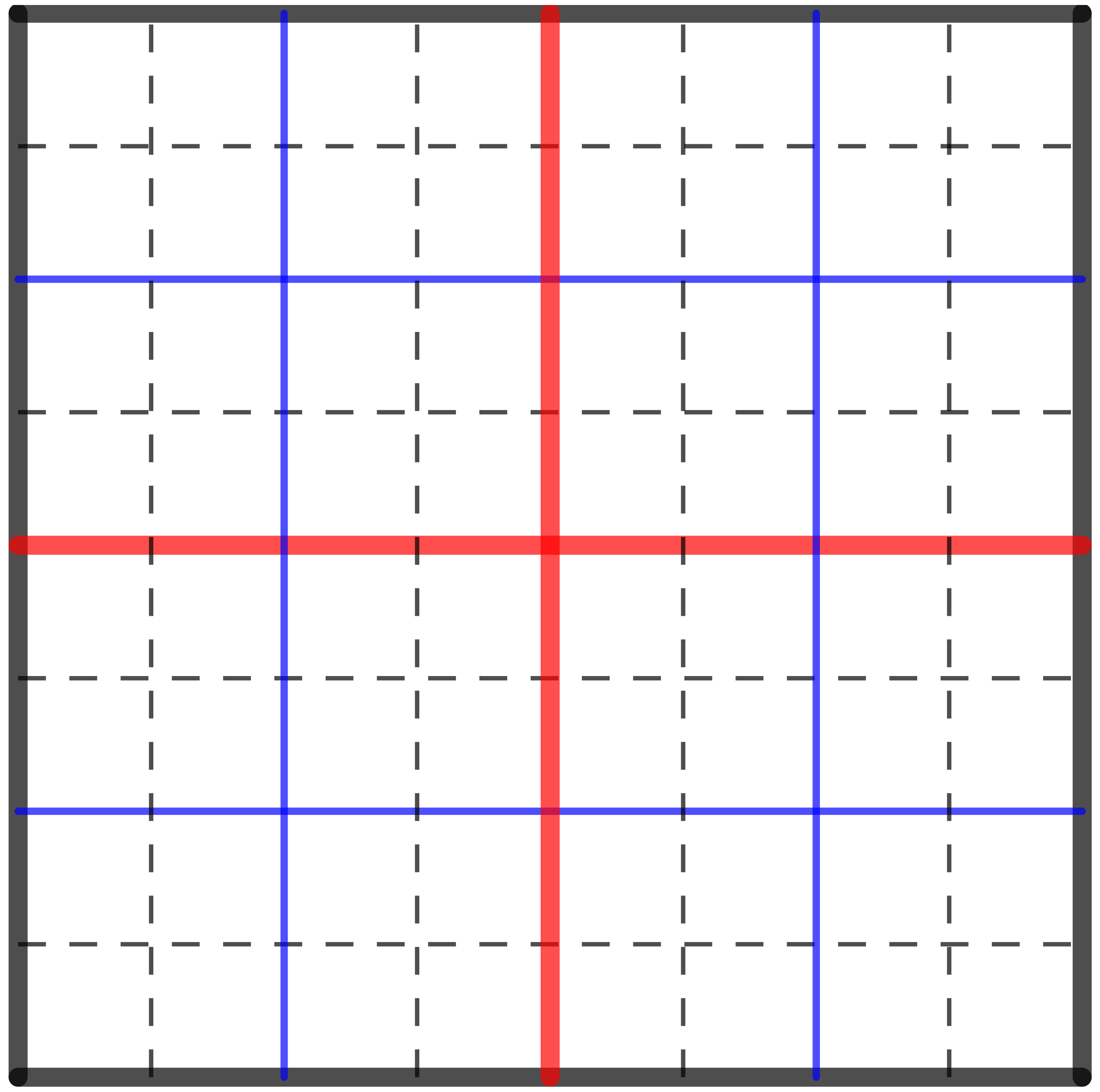
d维分成2^d个边长为一半的□

- 递归将当前□等分成4个边长为一半的□
- 组织成树形结构，直到只含单点停止，至多$\log_2\Delta$层



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

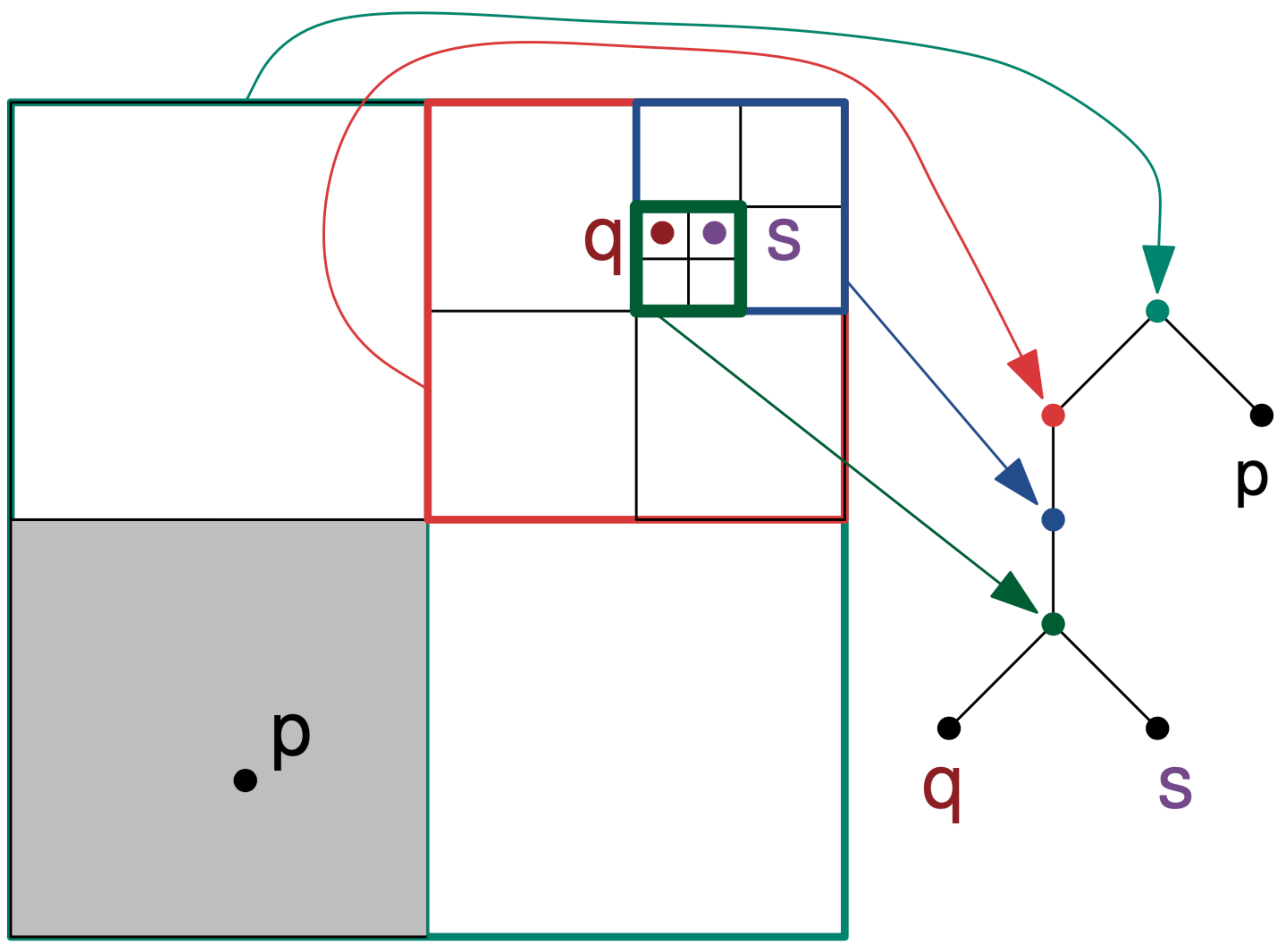
if ($|P \cap \square| = 1$) return T

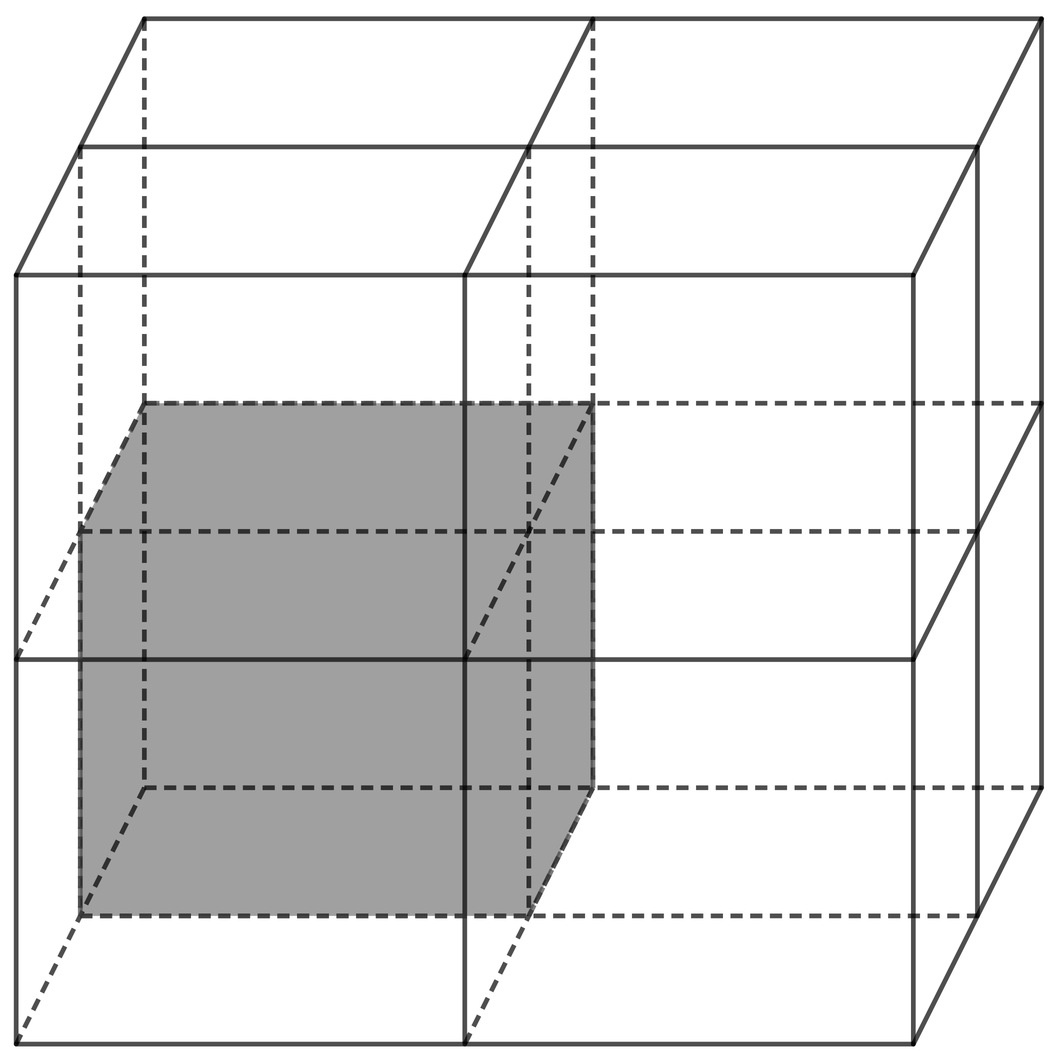
evenly divide $\square$ into 2^d sub-squares $\square_1, \ldots, \square_{2^d}$

for i = 1, …, 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

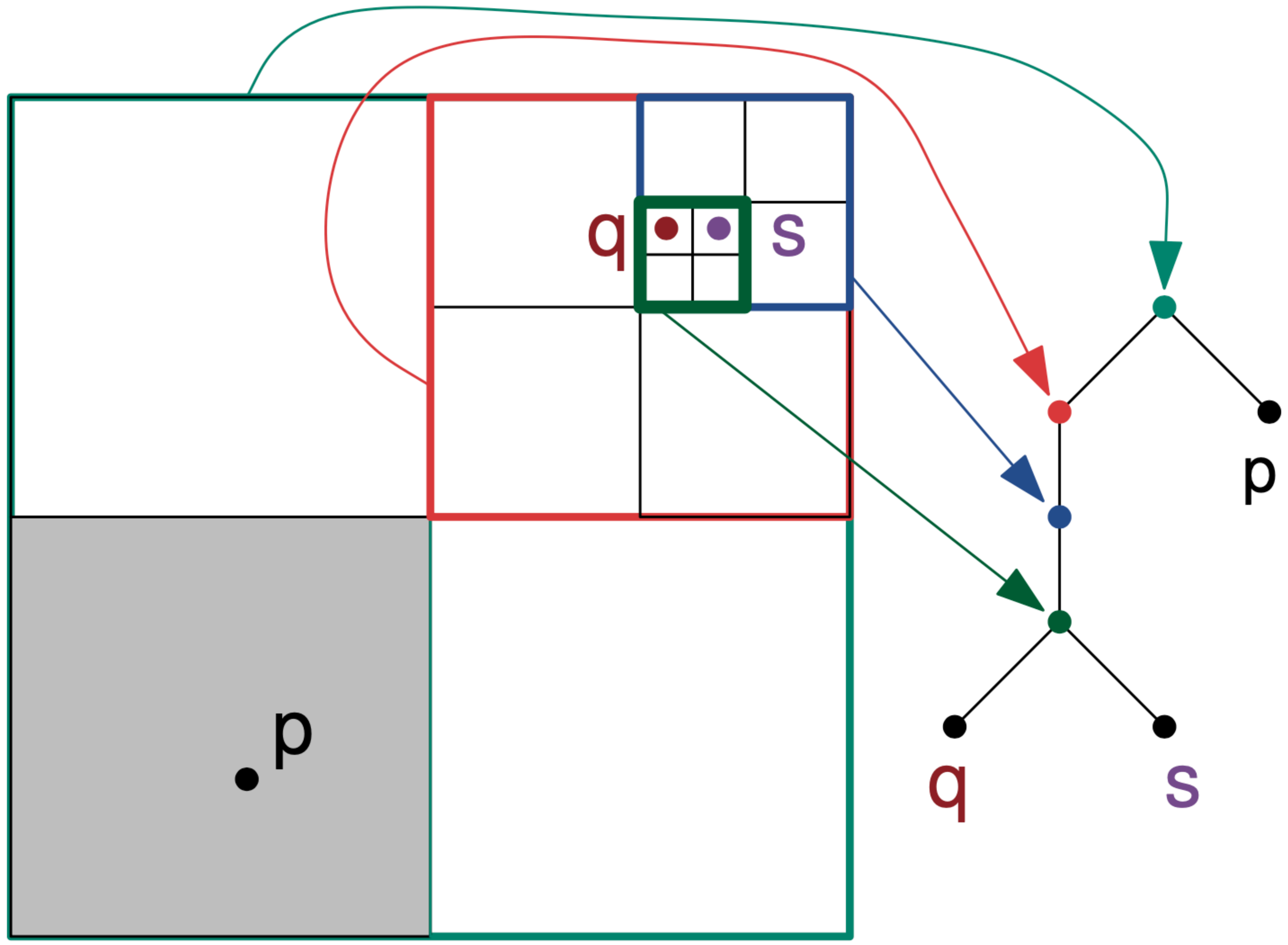
存储

- 每个□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：$O(n \cdot 2^d \log \Delta)$

2^d来自于存2^d个$\square_i$对应指针所需空间

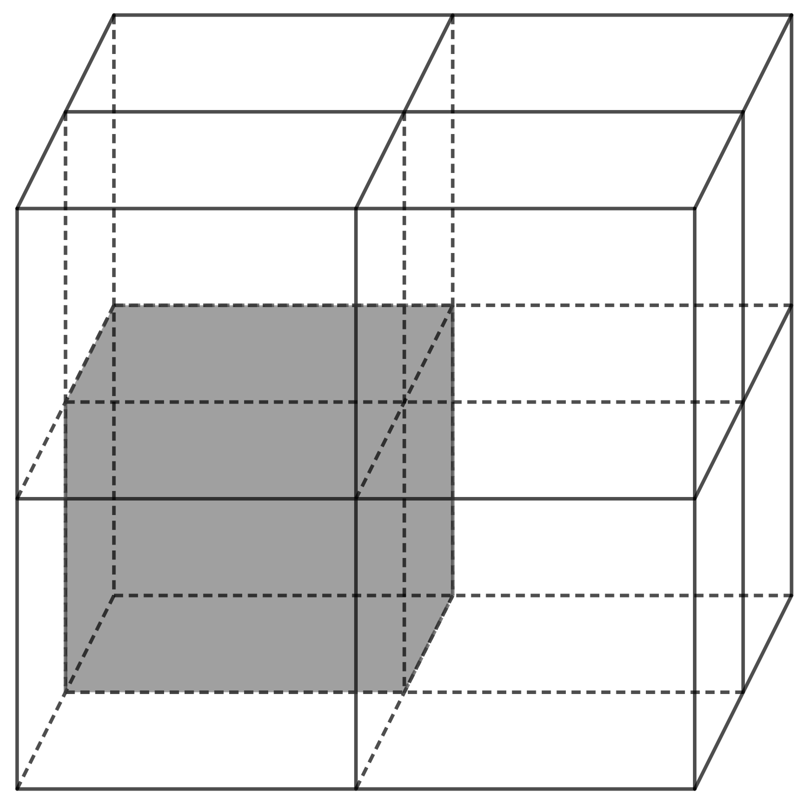
  - 树的每层□的并集存储整个点集P
  - 共（至多）$\log \Delta$层

注意：该树中没有空□



实现细节

对$i = 1,\ldots,2^d$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 $\square_1, \ldots, \square_{2^d}$

for i = 1, …, 2^d, let T.child$_i$ = 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$nd \log \Delta$

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

- 总时空复杂度：$O(nd \log \Delta)$若使用该优化，否则是$O(n2^d \log \Delta)$

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

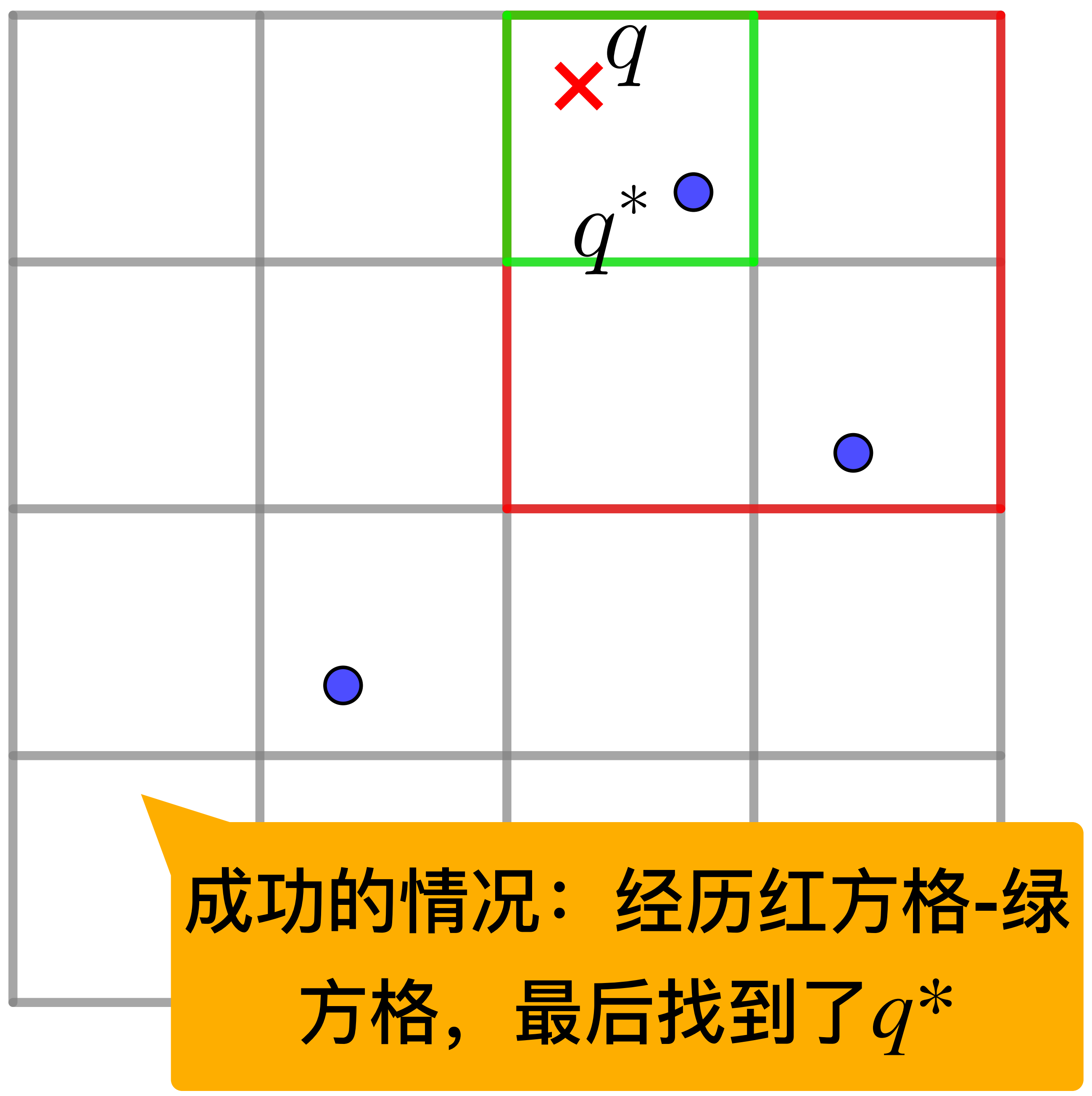
即$\hat{q}$到q的距离是q到q在P最近邻距离的$(1+\epsilon)$倍

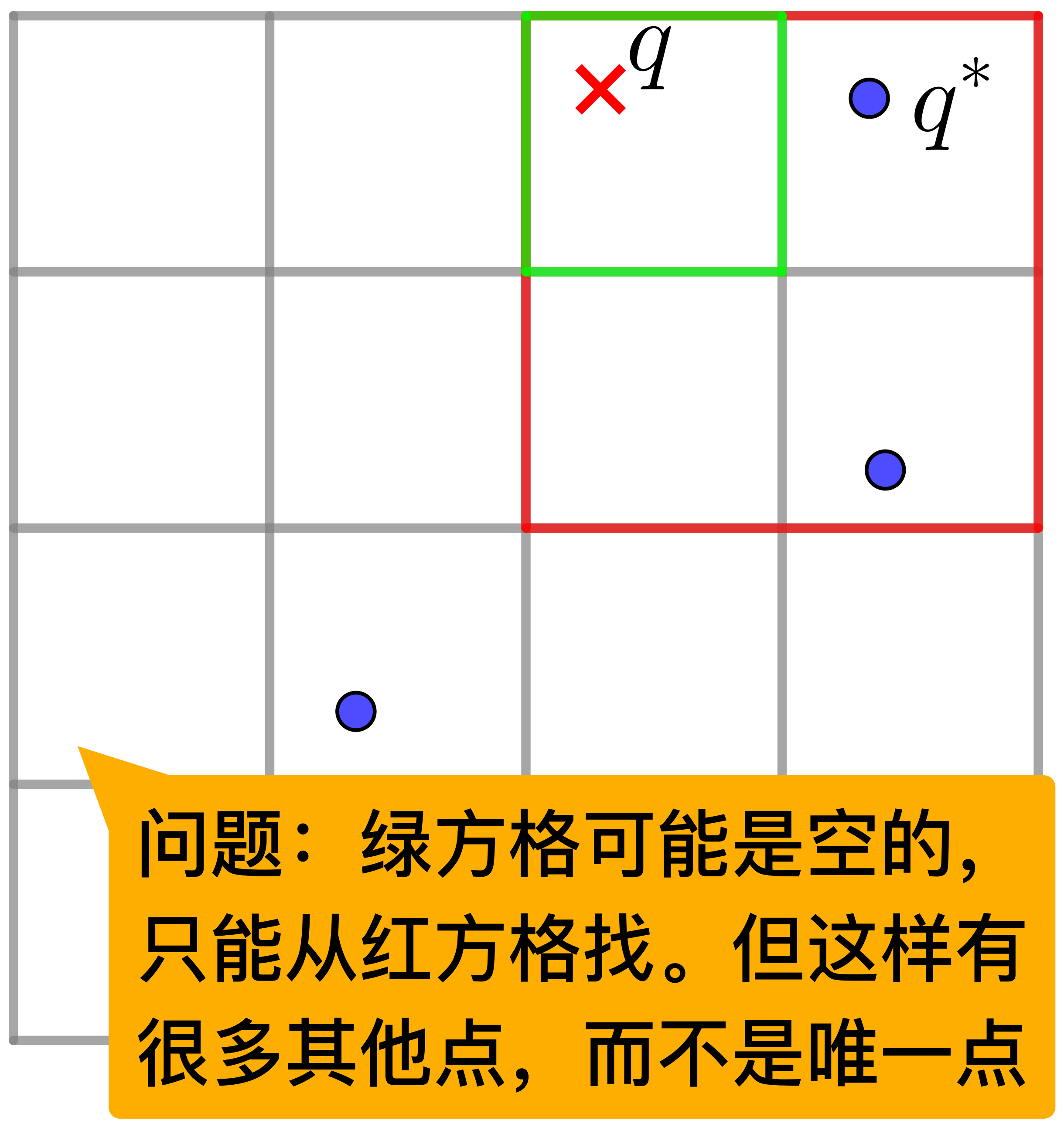
- 预处理：BuildTree($\square$, P)，这里调用 $\square = [\Delta]^d$
- 我们将给出一个$O(\epsilon^{-d} \log \Delta)$时间的查询算法返回满足要求的$\hat{q}$

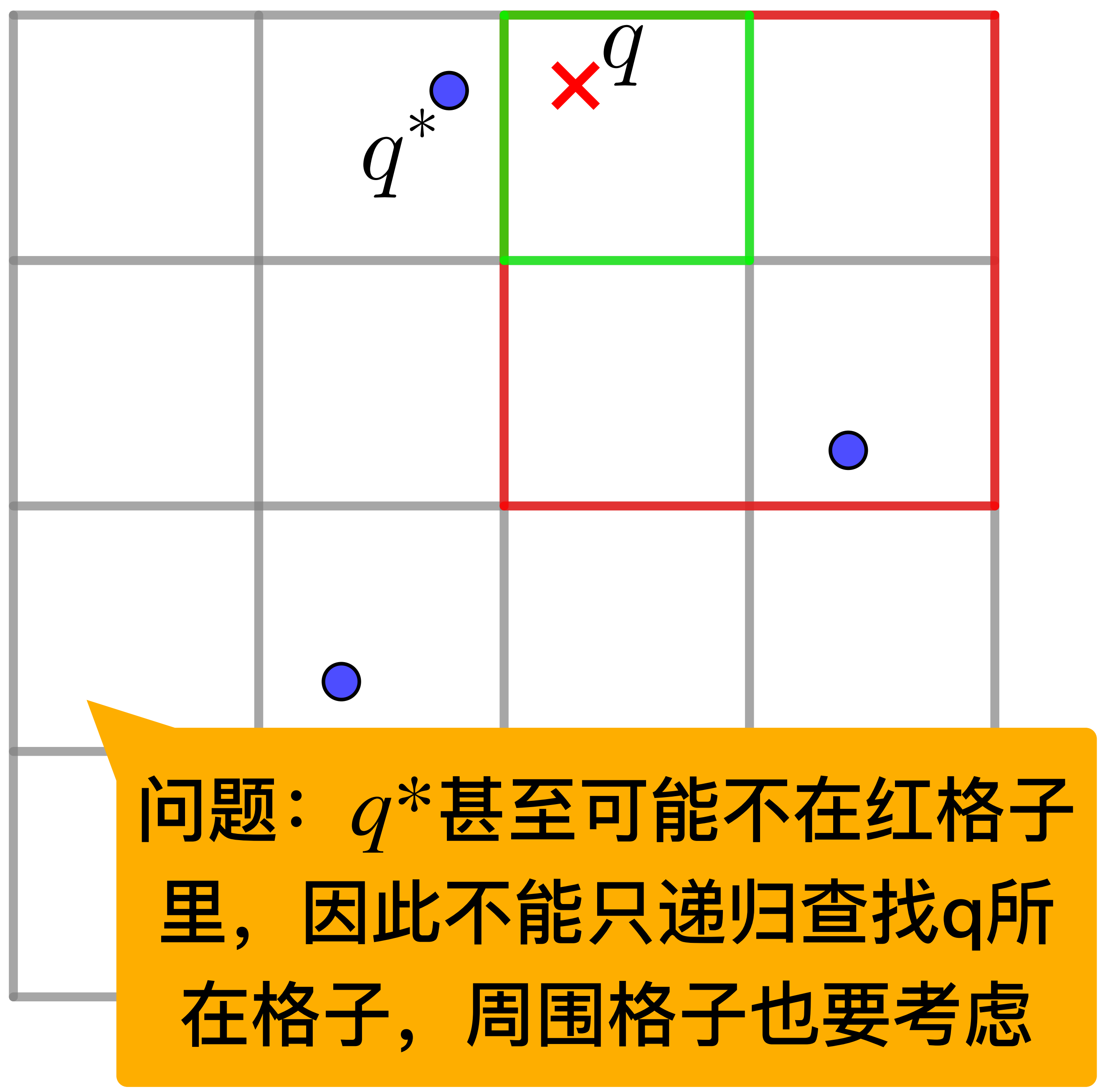
查询：一个（错误的）理想化算法

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

这对应于沿着$\log \Delta$层的路径走到底







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

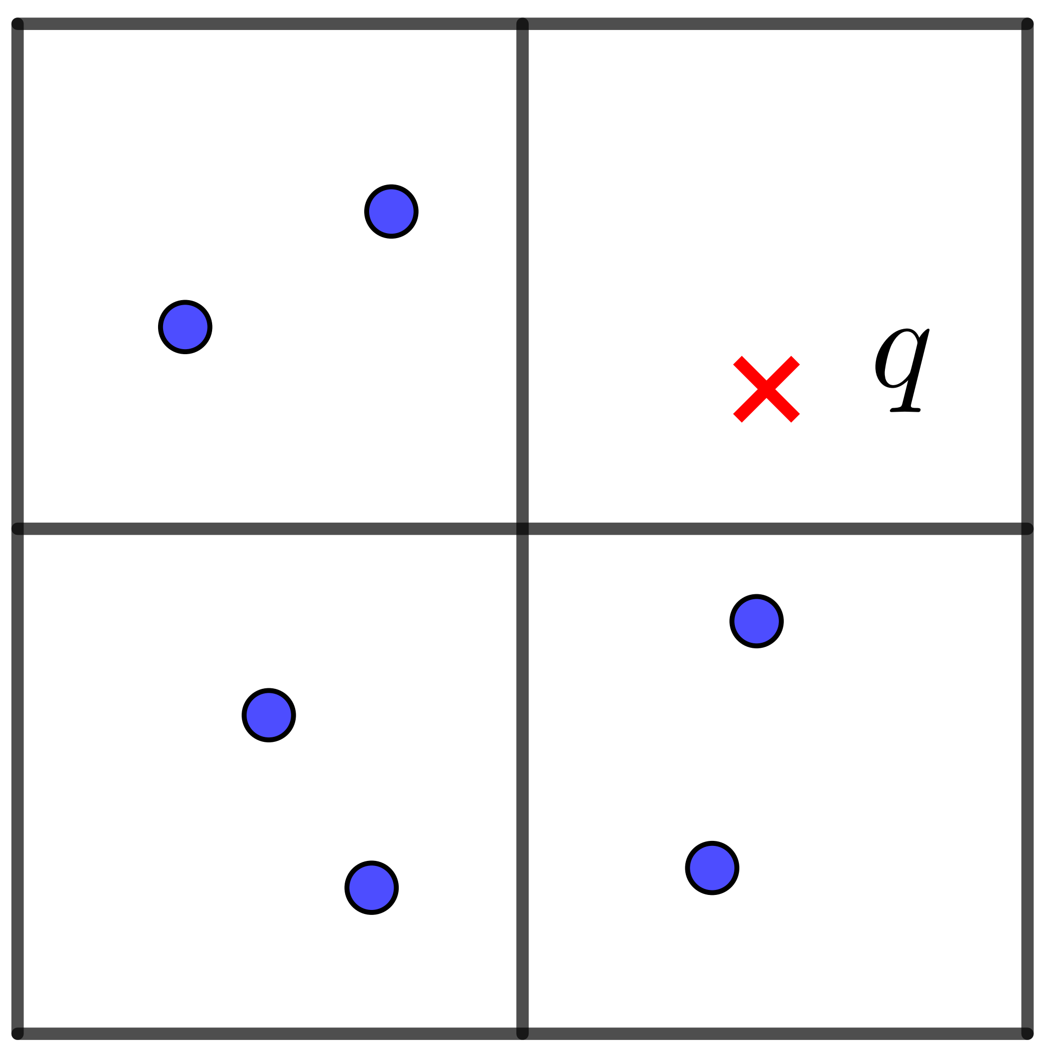
- 以第一层的4个$\square_i$为例：什么情况下才不用在$\square_i$继续递归?
- 根本要求：不能miss掉最近邻q^*的$(1+\epsilon)$-近似
- 设搜索过程中维护一个当前最优解$\hat{q}$，设$r := \|q - \hat{q}\|$
- 剪枝策略：若对任何$q' \in \square_i$都有

但这个判据很难高效实现

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

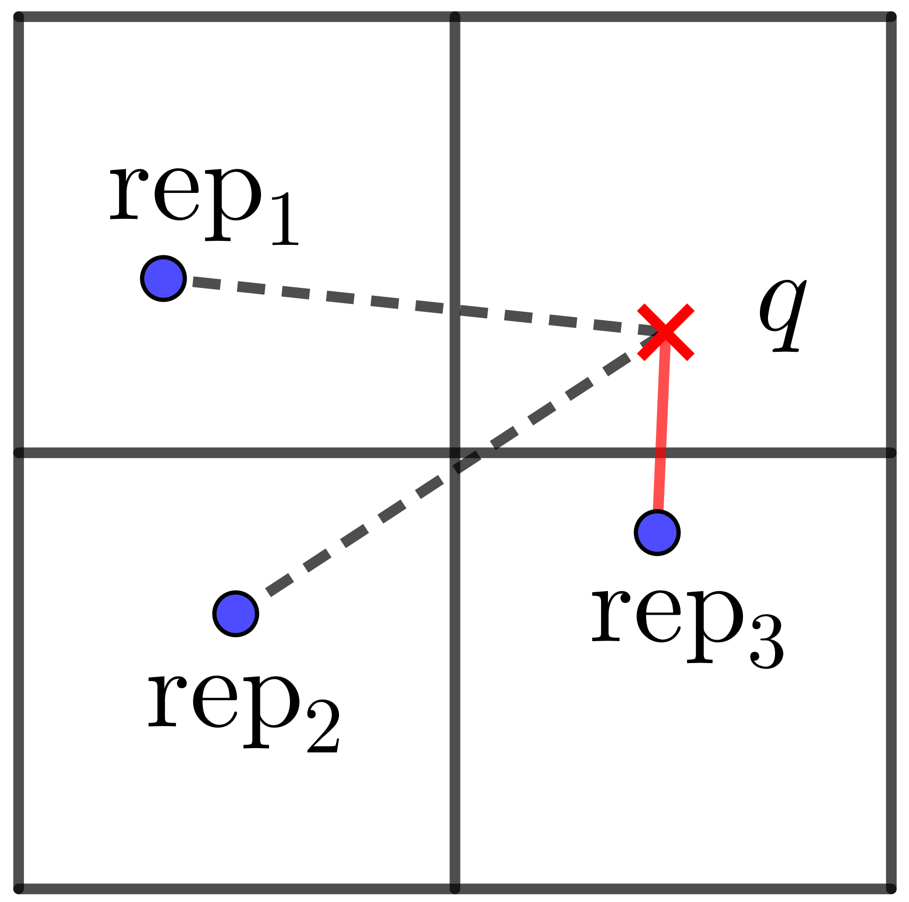
则即使$q' = q^*$也不再需要考虑$\square_i$了

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$\mathrm{rep}_{\square}$
- 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$\sqrt{2}\ell$

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

将≤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：$\Box_i$不需被考虑，若$\forall p \in \Box_i$，有$r \leq (1+\epsilon) \cdot \|q-p\|$
- 新剪枝：$\Box_i$不需被考虑，若$r \leq (1+\epsilon)(\|q-\mathrm{rep}_i\| - \mathrm{diam}(\Box_i))$
- 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$\Box_i$在新剪枝剪掉$\Rightarrow$$\Box_i$在旧剪枝剪掉：
    - 需要验证：若$r \leq (1+\epsilon)(\|q-\mathrm{rep}_i\| - \mathrm{diam}(\Box_i))$，则

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

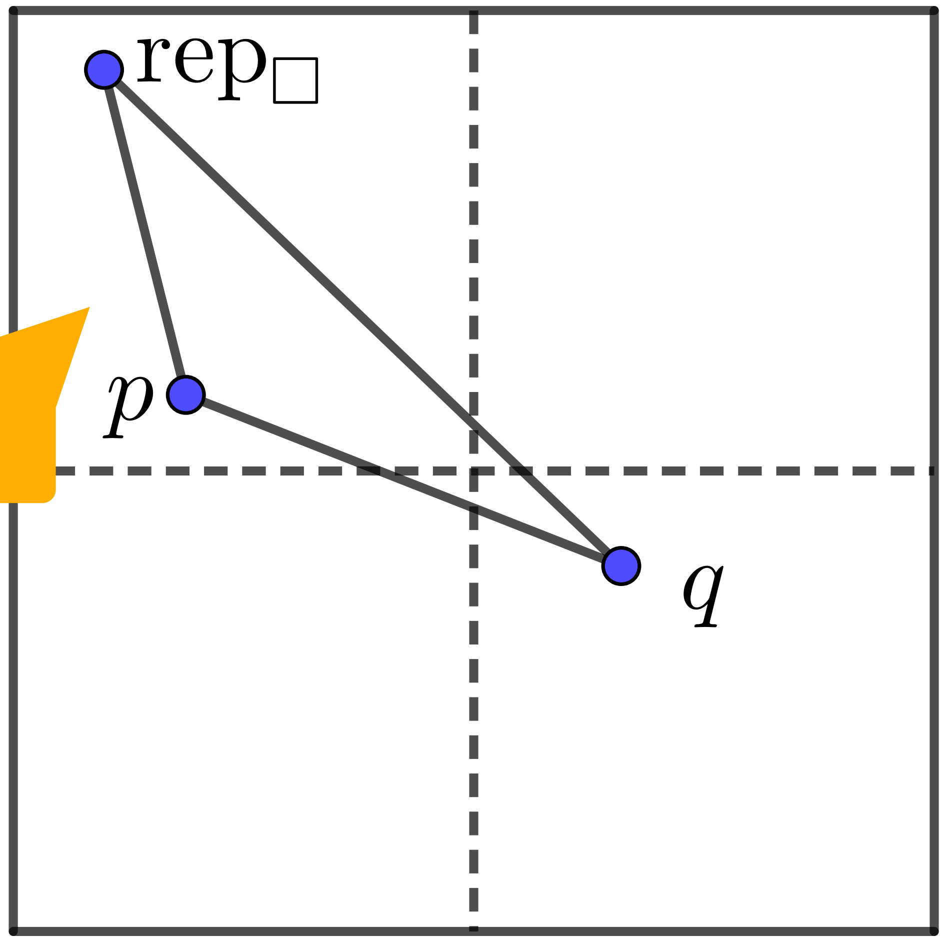
进一步解释

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

$$\forall p \in \square_i，有 r \le (1+\epsilon)\cdot \|q - p\|$$

- 性质：$\forall p \in \square_i$，有$\|q - \mathrm{rep}_i\| - \mathrm{diam}(\square_i) \le \|q - p\|$
  - 观察：任何p可由rep_i移动至多$\mathrm{diam}(\square_i)$距离到达
  - 数学语言：

$\|\mathrm{rep}_i - p\| \le \mathrm{diam}(\square_i)$



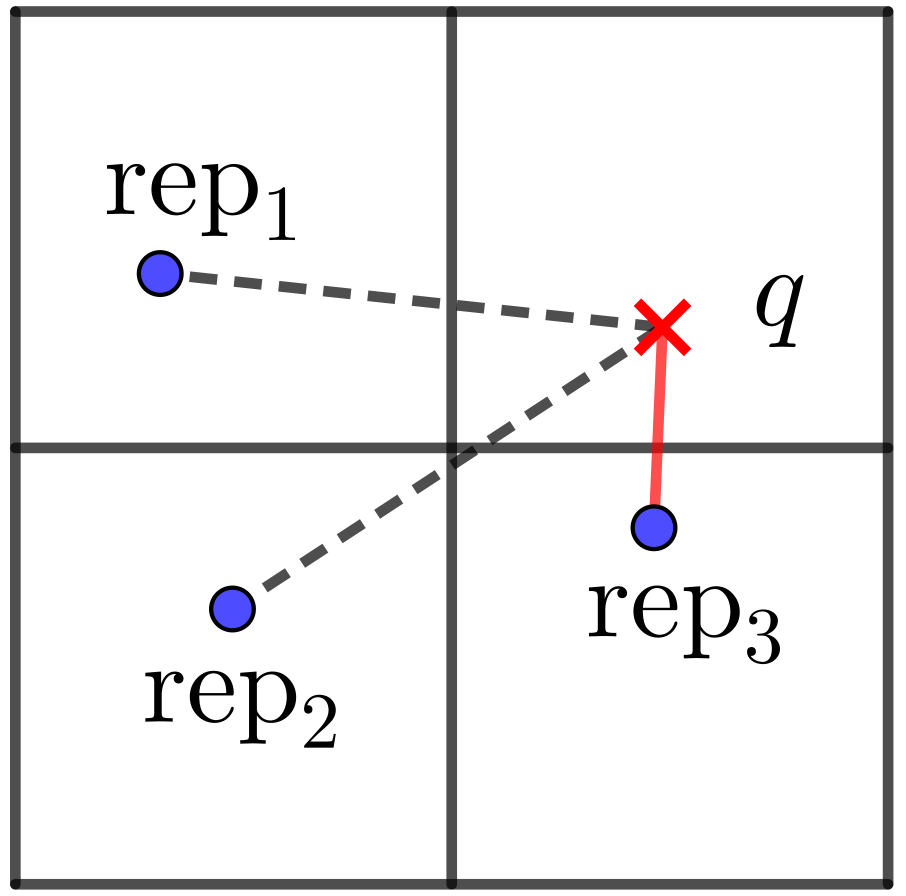
$$\|q - p\| \ge \|q - \mathrm{rep}_i\| - \|\mathrm{rep}_i - p\| \ge \|q - \mathrm{rep}_i\| - \mathrm{diam}(\square_i)$$

三角形不等式

利用$\|\mathrm{rep}_i - p\| \le \mathrm{diam}(\square_i)$

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（$0 \le i \le \log \Delta$）

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

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

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/24hw11/
- 分值：2分
- Deadline: 4月10日

* 算法分析

- 我们要说明的：
  - 正确性：算法结束时找到的点满足$r \leq (1+\epsilon)r*$
  - 效率：总运行时间是$O(\epsilon^{-O(d)}\log\Delta)$
- 先给出两个性质，然后推出 正确性 和 效率 的分析

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

“算法在当前解依然很坏的时候，从来不会miss掉最优解”

- 回忆算法步骤：

即剪枝规则

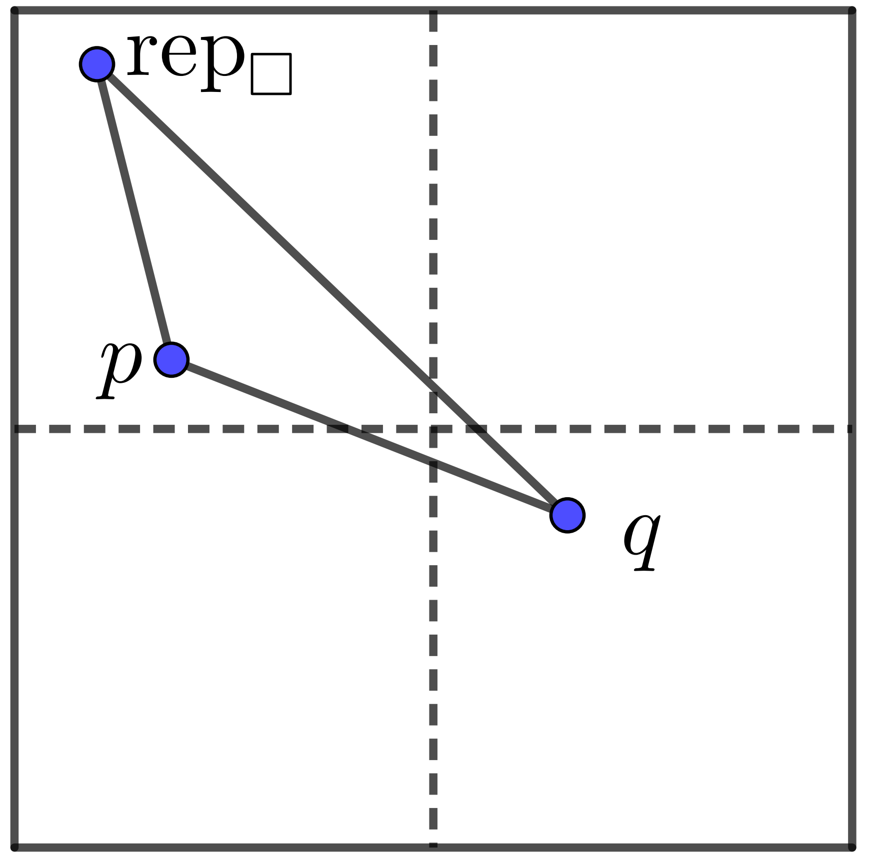
对每个$\square_w \in S$，若$r > (1+\epsilon)\cdot(\|q - \mathrm{rep}_w\| - \mathrm{diam}(\square_w))$，将$\square_w$放入$A_i$

- 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

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

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



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

“算法走到四分树足够深的地方就会停止，从而不继续访问不再需要访问的格子”

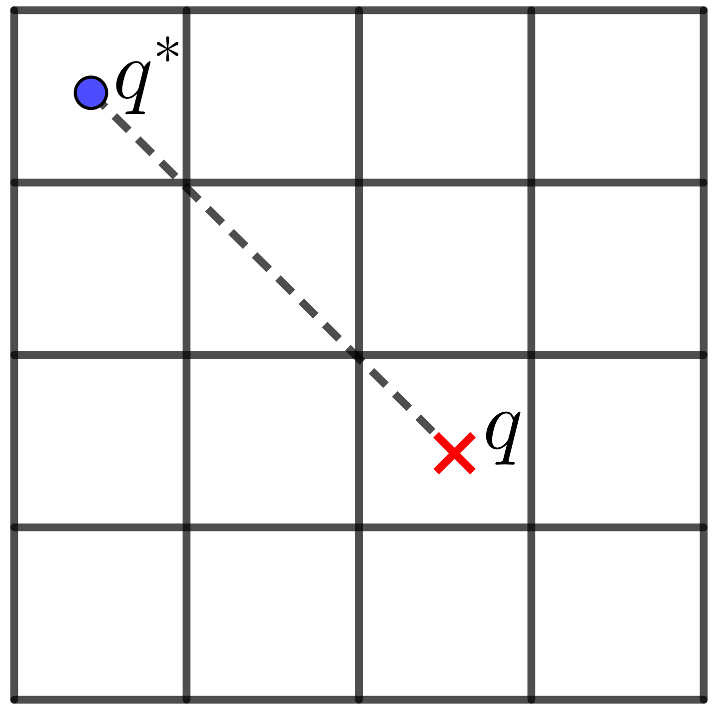
- 回忆算法步骤：

(*) 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diam$(\square_w) \leq \epsilon/2 \cdot r^*$，则$|A_i| = \varnothing$

即验证：任何 $\square_w \in S$都被剪枝，无法放入A_i



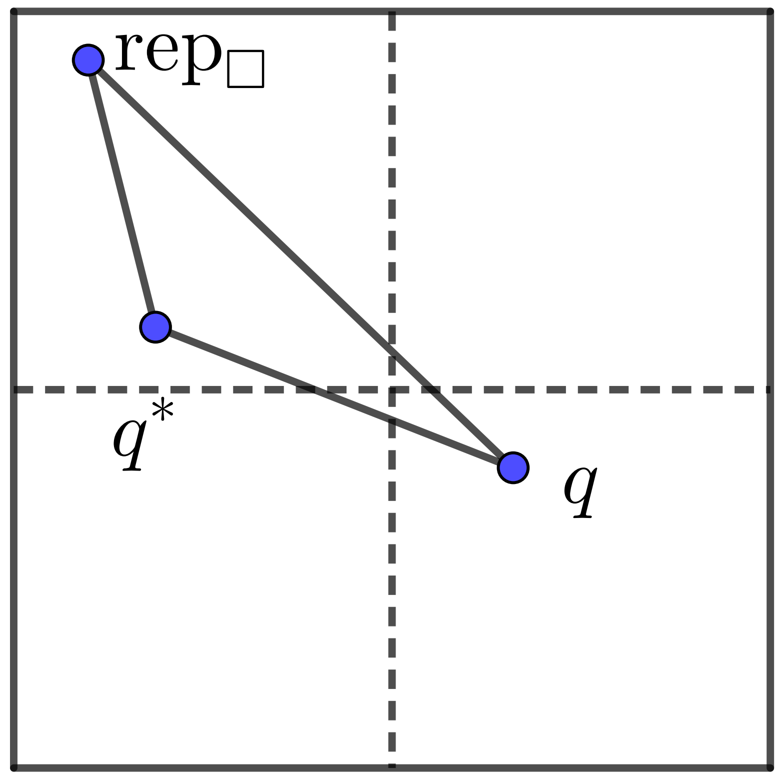
直觉：diam($\square$)很小 ($\leq O(\epsilon)\cdot r^*$)说明近似的足够精确了，没有任何点加进去可以显著改进精度了

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

* 正确性分析

总是能找到$r \le (1+\epsilon)r^*$

- 因为算法在$\mathrm{diam}(\square) < \epsilon/2 \cdot r^*$后就停止了，只需考虑$\mathrm{diam}(\square) \ge \epsilon/2 \cdot r^*$
  - 若停止前的某个时刻$r \le (1+\epsilon) \cdot r^*$，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$\mathrm{diam}(\square) \approx \epsilon/2 \cdot r^*$，我们有$r \le r^* + \mathrm{diam}(\square) \le (1+\epsilon)r^*$



根据三角形不等式：

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

* 运行时间分析

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

- 回忆算法步骤：

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

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

- 若$\square_w \in S$放入A_i，则$\|q - \mathrm{rep}_w\| < r/(1+\epsilon) + \mathrm{diam}(\square_w)$ 根据 (**)
- Claim: $r/(1+\epsilon) + \mathrm{diam}(\square_w) \le O(1/\epsilon) \cdot \mathrm{diam}(\square_w)$
- 先假定Claim正确：则若$\square_w$放入了A_i，$\|q - \mathrm{rep}_w\| \le O(1/\epsilon) \cdot \mathrm{diam}(\square_w)$

* 推出 $|A_i|$ 上界

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

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

每个rep_w对应一个正方形！

- 因此有 $|A_i| \le (O(1/\epsilon))^d$

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

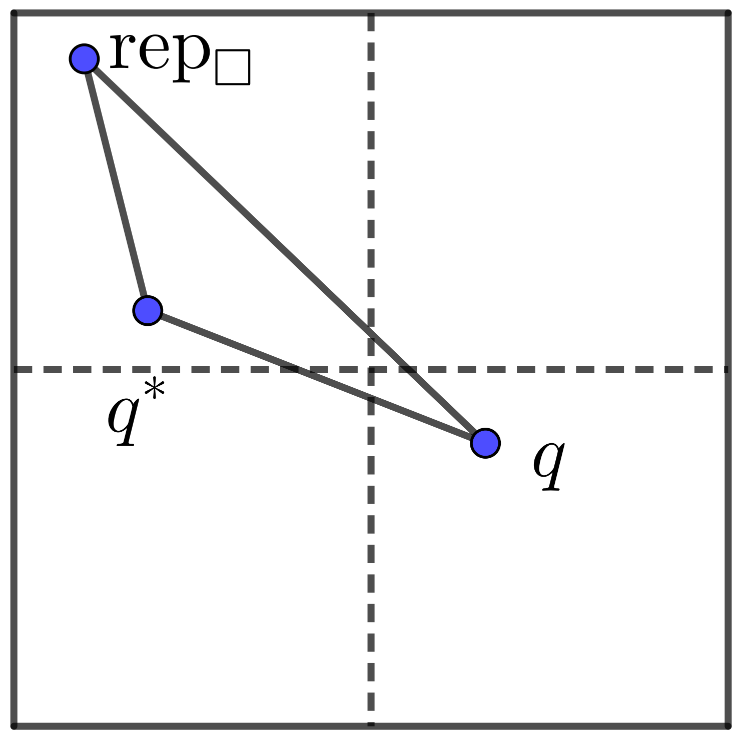
* 验证Claim

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

- 等价于证$\mathrm{diam}(\square_w) \geq O(\epsilon/(1+\epsilon)) \cdot r$

根据性质2；否则会导致$|A_i| = 0$，不需要考虑

- 假定算法执行中总有$\mathrm{diam}(\square_w) > \epsilon/2 \cdot r^*$
- 若$r \leq (1+\epsilon)r^*$，则$\mathrm{diam}(\square_w) > \epsilon/2 \cdot r^* \geq \epsilon/(2(1+\epsilon))r$
- 否则$r > (1+\epsilon)r^*$根据性质1，q^*所在方格未被剪枝



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

有$r \leq r^* + \mathrm{diam}(\square_w)$，推出$\mathrm{diam}(\square_w) \geq r - r^* > r - r/(1+\epsilon) = \epsilon/(1+\epsilon) \cdot r$

* 其他可能的剪枝策略?

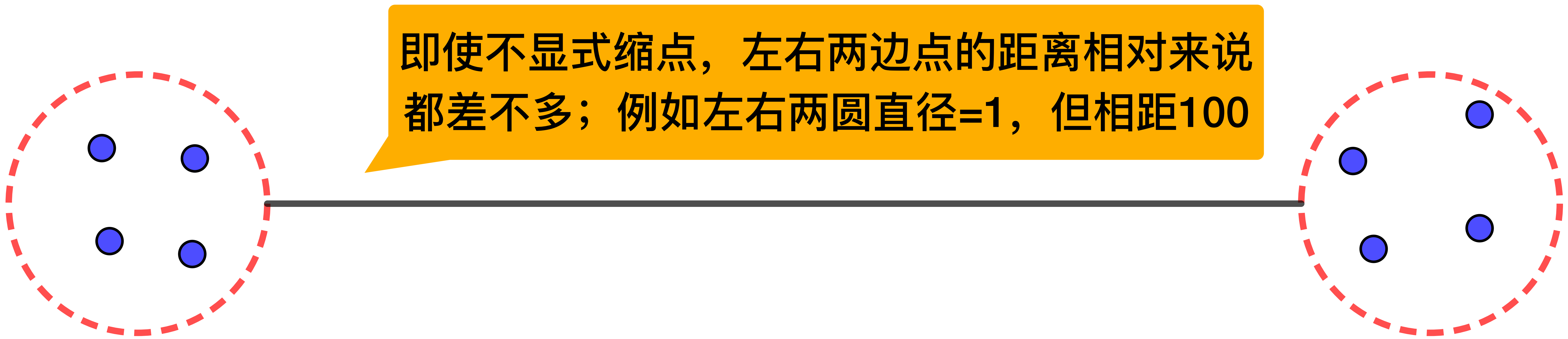
- 有同学提出：改成用$r \leq (1+\epsilon) \cdot \mathrm{dist}(q, \square_i)$ 来断定$\square_i$应该剪掉
- 与我们的剪枝条件$r \leq (1+\epsilon)(\|q - \mathrm{rep}_i\| - \mathrm{diam}(\square_i))$如何比较?
    - 是否有一者推出另一者?
    - 思考：求$\mathrm{dist}(q, \square_i)$需要多少时间？正确性？总运行时间?

3. 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



WSPD

定义

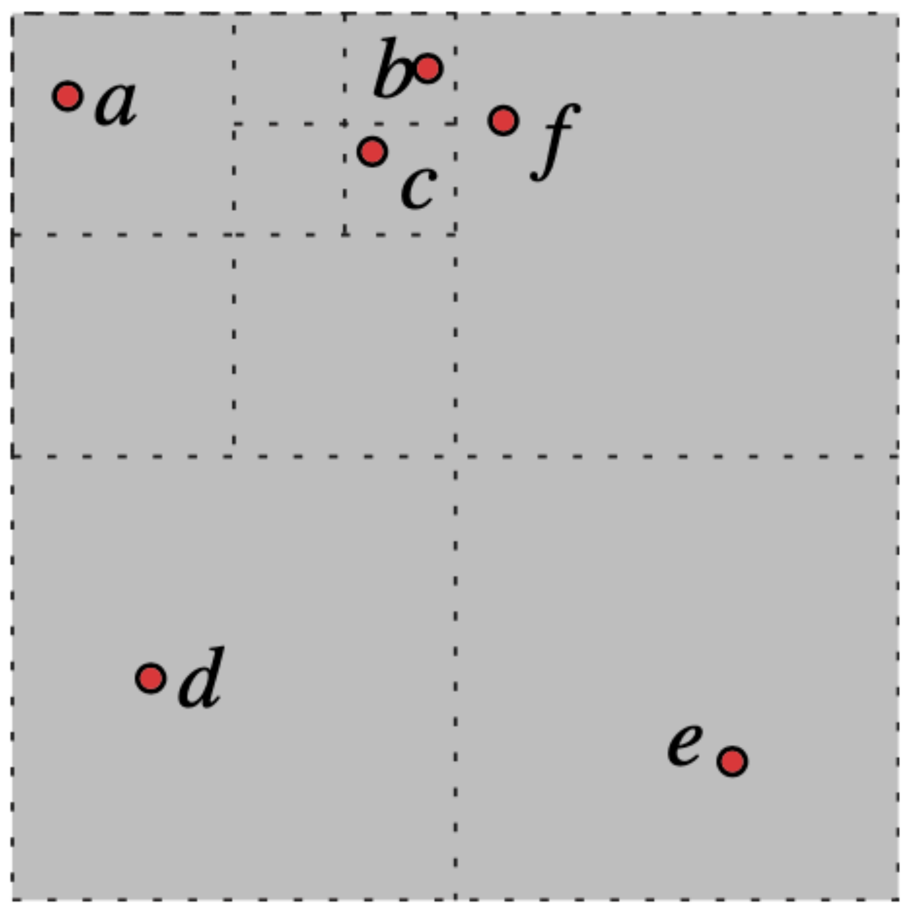
点集之间的距离等于最近点的距离，即 $\mathrm{dist}(S,T) := \min_{x\in S, y\in T} \|x-y\|$

- 一个ϵ-WSPD是一系列$\{\{A_i, B_i\}\}_i$，使得
  - $\forall i$，$A_i, B_i \subseteq P$且$\max(\mathrm{diam}(A_i), \mathrm{diam}(B_i)) \le \epsilon \cdot \mathrm{dist}(A_i, B_i)$

即A_i, B_i相对直径来说距离很远

  - $\bigcup_i A_i \times B_i = P \times P$，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



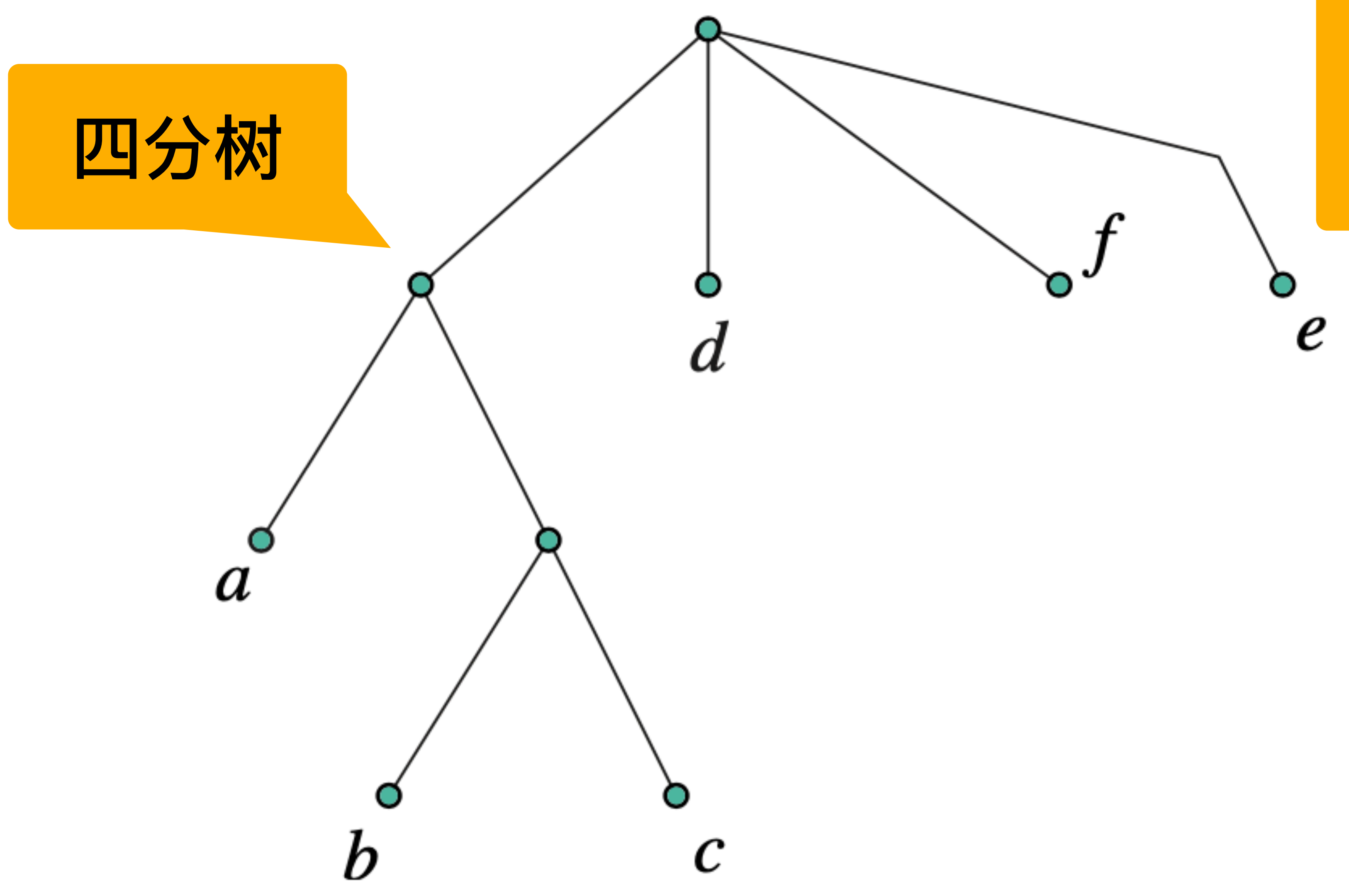
(i)

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

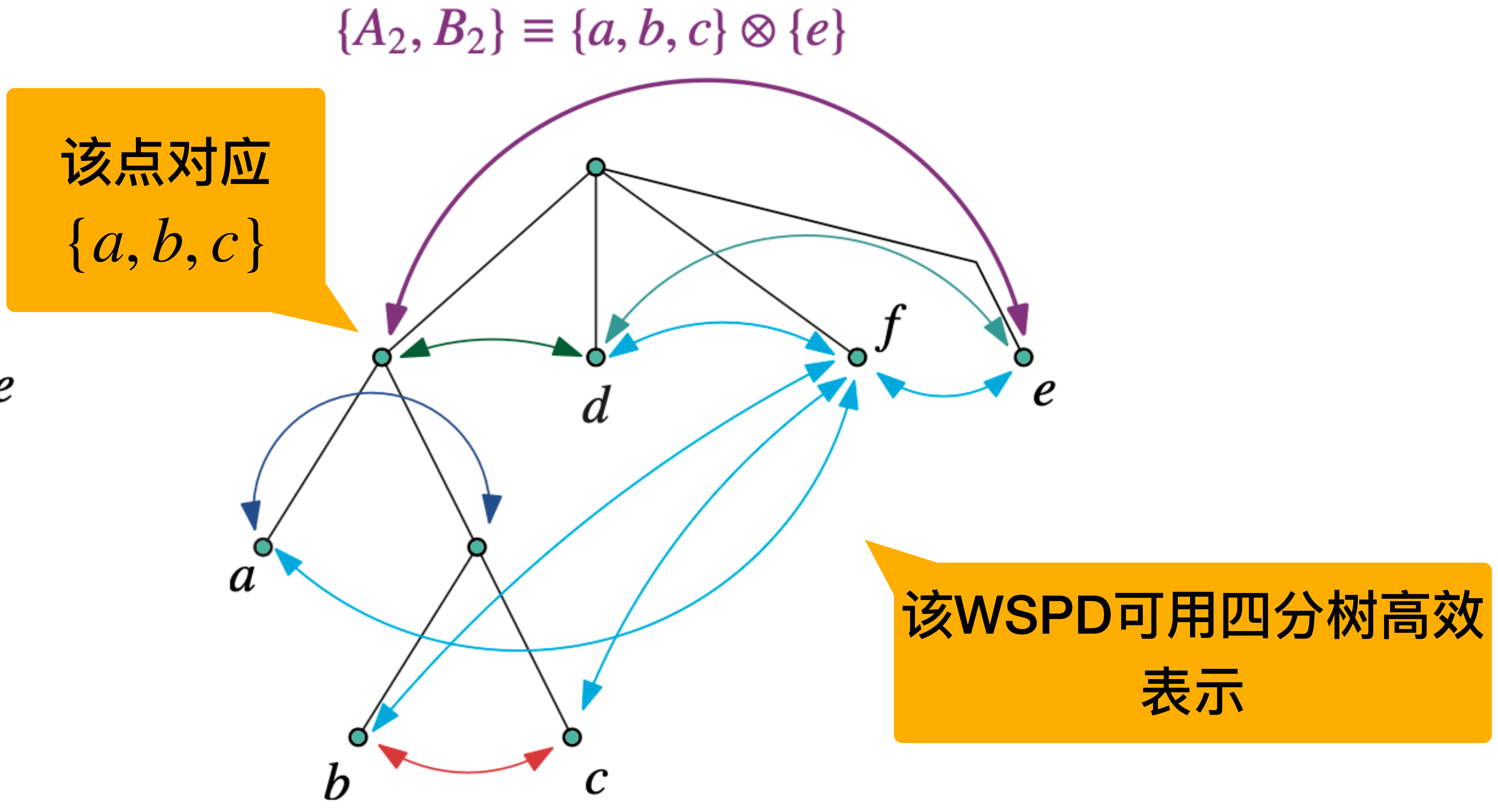
(ii)

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

(iii)



(iv)



(v)

基于四分树的WSPD：完整算法

WSPD用四分树的□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(□, □)

WSPD($\square_u$, $\square_v$)

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

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

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

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

dist直接看□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 $\bigcup_{i=1}^{2^d}\mathrm{WSPD}(\square_{u_i},v)$

$\square_{u_i}$是$\square_u$的子□

* 算法性能分析

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$\{A_i, B_i\}$

$\delta(\square_u) \geq \delta(\square_v)$

- 对每个四分树方格$\square_v$考虑所有加入WSPD的$\{\square_u, \square_v\}$，将对应的$\square_u$加入S_v
- Claim: $|S_v| \leq O(\epsilon^{-O(d)})$
- 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($\hat{p}(\square_u)$, $\square_v$)和($\square_u$, $\hat{p}(\square_v)$)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 $\bigcup_{i=1}^{2^d} \mathrm{WSPD}(\square_{u_i}, v)$

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$\square_v$，$|S_v| \le O(\epsilon^{-O(d)})$

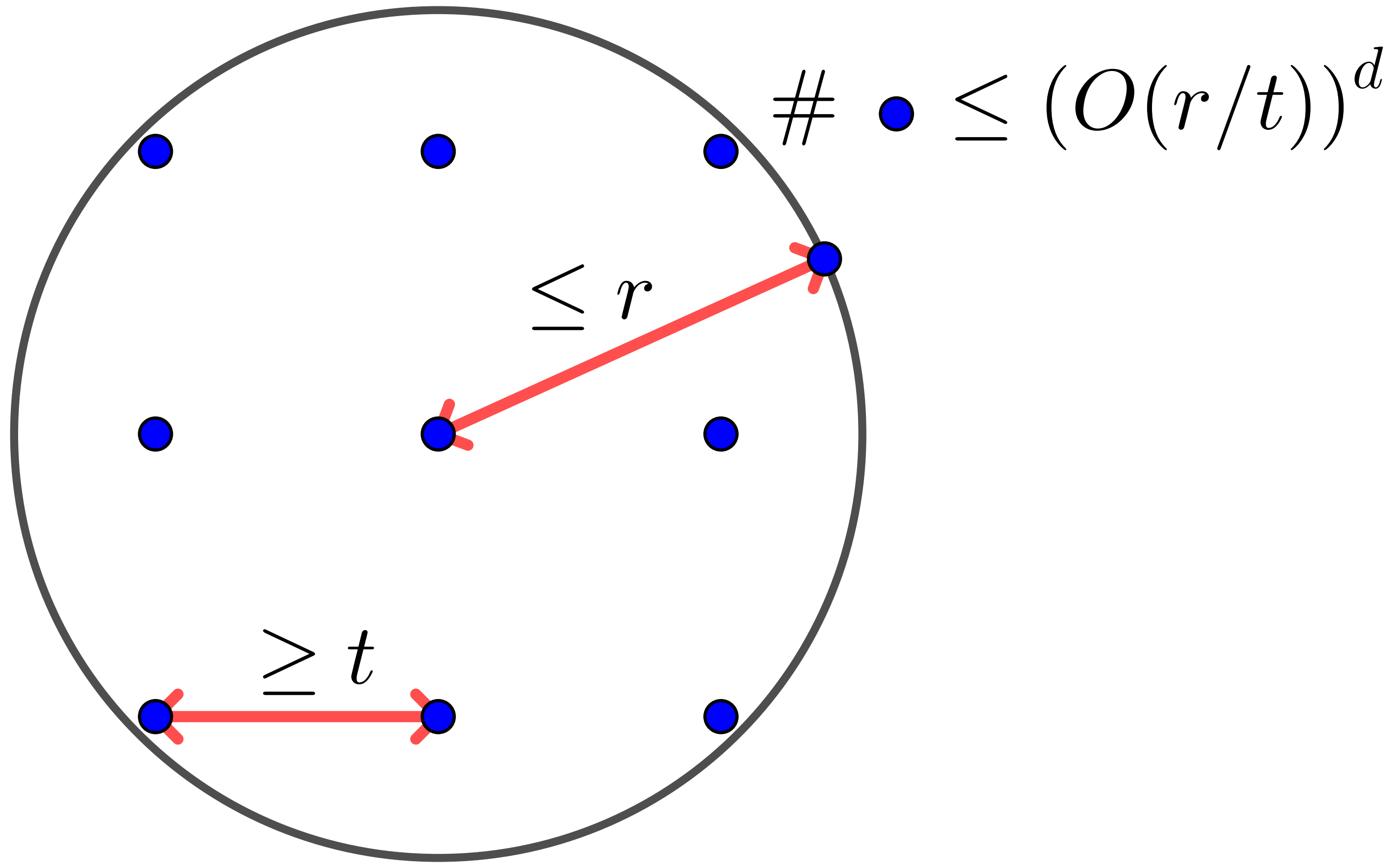
对四分树方格$\square_v$，S_v为算法中所有加入WSPD的$\square_u$的集合

- 辅助结论：$\forall \square_u \in S_v$，$\mathrm{dist}(\square_u, \square_v) \le O(1/\epsilon) \cdot \delta(\square_u)$

下页验证，现在假定成立

- 若$\delta(\square_u)$固定 $= t$，则满足$\mathrm{dist}(\square_u, \square_v) \le O(1/\epsilon) \cdot \delta(\square_u)$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$|S_v| \le O(\epsilon^{-O(d)})$

算法性质：$\square_u$和$\square_v$至多差一层

* 验证辅助结论

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

- 回忆算法步骤： if $(\delta(\square_u) \leq \epsilon \cdot \mathrm{dist}(\square_u, \square_v))$ 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$(\hat{p}(\square_u), \square_v)$和$(\square_u, \hat{p}(\square_v))$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且不可能会产生$(\square_u, \square_v)$调用

* 验证辅助结论 cont.

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

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

if $(\delta(\square_u) \le \epsilon \cdot \mathrm{dist}(\square_u, \square_v))$

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

$$\delta(\hat{p}(\square_u)) > \epsilon \cdot \mathrm{dist}(\hat{p}(\square_u), \square_v)$$

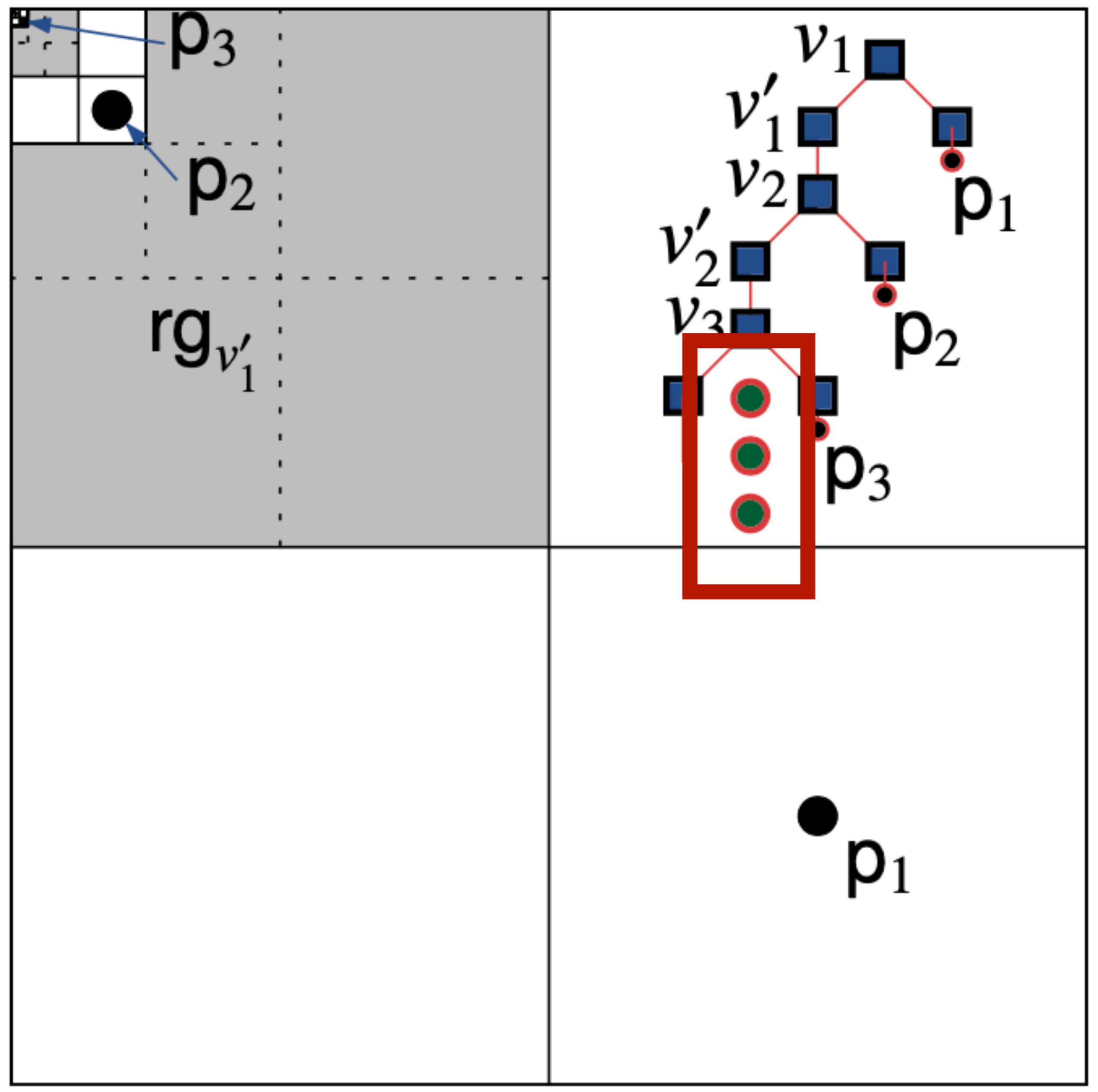
- 可以得出：$\mathrm{dist}(\square_u, \square_v) \le O(1/\epsilon) \cdot \delta(\hat{p}(\square_u))$

进一步 $\le O(1/\epsilon) \cdot \delta(\square_u)$

- 三角形不等式推出$\mathrm{dist}(\square_u, \square_v) \le \mathrm{dist}(\hat{p}(\square_u), \square_v) + \delta(\hat{p}(\square_u))$
- 代入$\mathrm{dist}(\hat{p}(\square_u), \square_v) < O(1/\epsilon) \cdot \delta(\hat{p}(\square_u))$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□：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□



- $\log \Delta$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$\square_v$，中间全部删除

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$\epsilon < 1$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

    $O(n \cdot 2^d \log \Delta)$时间
  - 在所有距离中的最近点对就是答案
- 原因：设最近点对是(p, q)且$p \in A_i$，$q \in B_i$，则$|A_i| = |B_i| = 1$

  因此算法不会遗漏最近点对

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/24hw12/
- 分值：2分
- Deadline: 4月10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$q \in A_i$，$s \in B_i$

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

WSPD性质

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$P \subseteq [\Delta]^d$的一个欧氏图的子图H
  - 欧氏图：点集还是P，边集是$P \times P$，边权重是端点间的欧氏距离
  - 子图：点集相同，边集是子集
  - t-Stretch：要求对任何$x, y \in P$有$\|x - y\| \leq \mathrm{dist}_H(x, y) \leq t \cdot \|x - y\|$
  - 追求：给定t，H满足t-stretch条件，且边数尽量少

dist_H代表H上的最短路距离

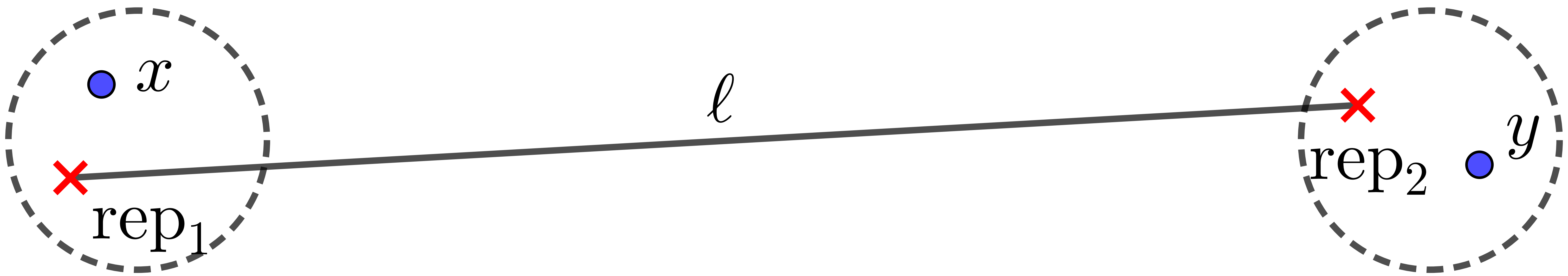
- 可看作是欧氏图的稀疏“压缩”，后续可使用稀疏图上的算法来解决欧氏问题

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



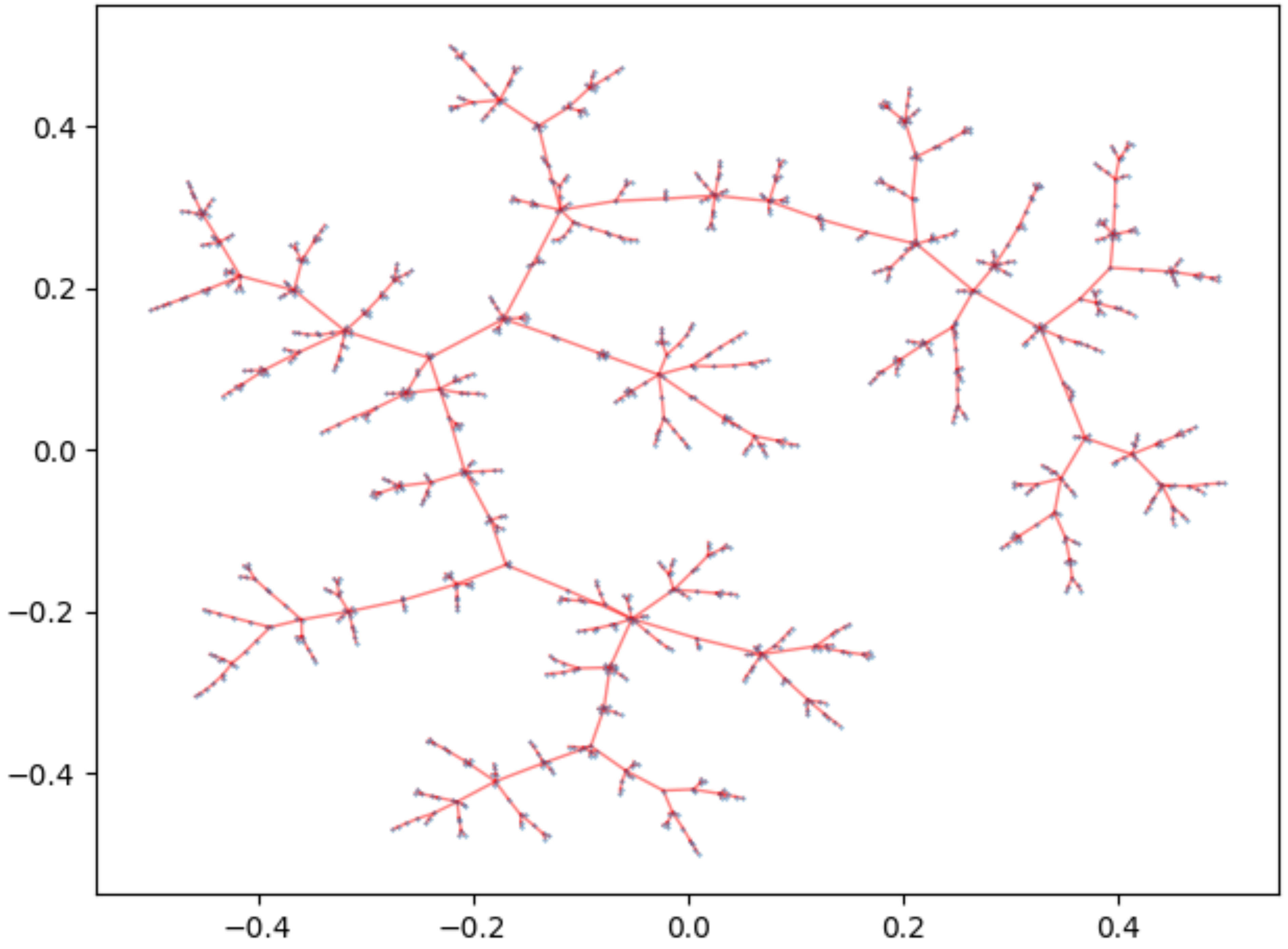
- 考虑一对点x和y，设$x \in A_i$，$y \in B_i$，且rep_1和rep_2是代表点、其连边被加入H
  - 不失一般性
  - 观察：$\mathrm{dist}(x, y) \geq \mathrm{dist}(A_i, B_i) \geq 1/\epsilon \cdot \mathrm{diam}(A_i) \geq 1/\epsilon \cdot \mathrm{diam}(B_i)$
  - x, y足够远
  - $\ell = \mathrm{dist}(\mathrm{rep}_1, \mathrm{rep}_2)$
  - 推出：$\ell \leq \mathrm{dist}(A_i, B_i) + \mathrm{diam}(A_i) + \mathrm{diam}(B_i) \leq (1+\epsilon/2) \cdot \mathrm{dist}(x, y)$
  - rep距离足够近似x, y距离
- 再利用归纳假设，

$$\begin{aligned} \mathrm{dist}_H(x, y) &\leq \mathrm{dist}_H(x, \mathrm{rep}_1) + \mathrm{dist}_H(y, \mathrm{rep}_2) + \ell \\ &\leq 2(1+\epsilon) \cdot \epsilon/4 \cdot \ell + \ell \leq (1+\epsilon) \cdot \mathrm{dist}(x, y) \end{aligned}$$

用H上的rep距离近似x, y距离

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的欧氏图上的MST



利用$(1+\epsilon)$-Spanner求$(1+\epsilon)$-近似最小生成树

- 算法：
  - 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

$\mathrm{MST}(H) \le (1+\epsilon)\cdot \mathrm{MST}(P)$

H是$P \times P$的子图，因此H中的边都在P的欧氏图中

作业：$(1+\epsilon)$-近似欧氏最小生成树

- http://cssyb.openjudge.cn/24hw13/
- Deadline: 4月17日

4. 随机平移四分树与 Tree Embedding

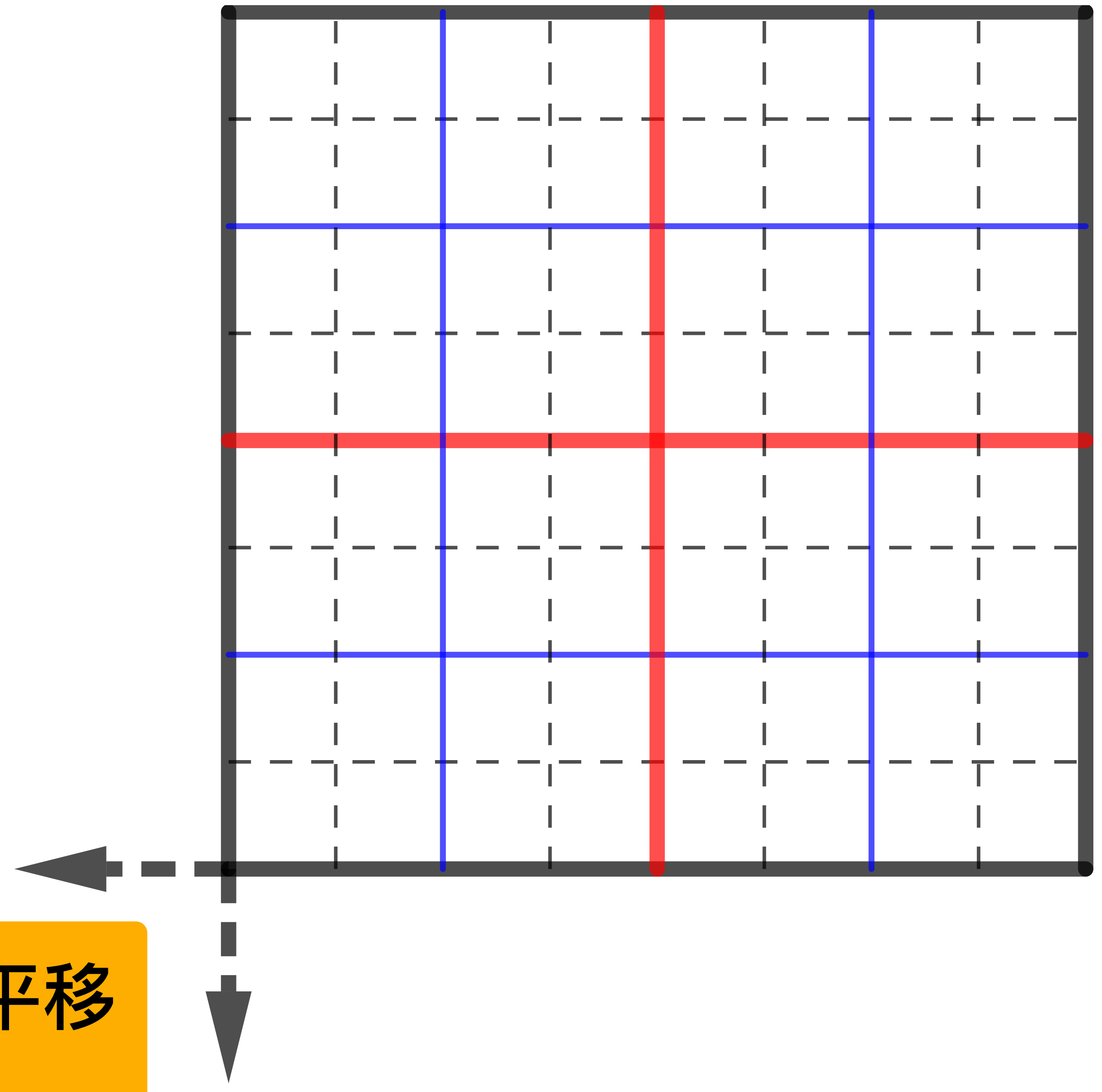
Tree Embedding

- 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$\forall x, y \in P$，$\mathrm{dist}_T(x, y) \geq \|x - y\|$
- 主要性能衡量叫distortion:

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

是 ≥ 1的量，越小越好

构造算法和结论



- 选取$[0,\Delta]^d$上的均匀随机向量v，将P所有点平移v
- 构造四分树BuildTree($\square = [2\Delta]^d, P$)

$[2\Delta]$是为了依然包含随机平移后的P

- 定义树T：每个□定义一个树节点，数据点都放在树的叶子上
  - 定义边：每个□连接到所有的直接孩子，赋予$\sqrt{d} \cdot 2^i$的权值，设□边长2^i
- 对于$x, y \in P$（都是树的叶子），距离$\mathrm{dist}_T(x, y)$就是树上的距离
- 结论：$\forall x, y \in P$，$\mathrm{dist}_T(x, y) \geq \|x - y\|$，$\mathbb{E}[\mathrm{dist}_T(x, y)] \leq O(dh) \cdot \|x - y\|$

distortion是随机的

h是四分树的高度

应用

$O(dh)$近似MST

- 结论：$\forall x, y \in P$，$\mathrm{dist}_T(x, y) \geq \|x - y\|$，$\mathbb{E}[\mathrm{dist}_T(x, y)] \leq O(dh) \cdot \|x - y\|$
- 考虑高维MST
  - 先建立T，然后在T上找（所有叶子的）MST，记为F_T，并设精确MST解为F^*

观察到$w(F_T) \leq \sum_{x,y \in F^*} \mathrm{dist}_T(x, y)$，所以

$$\mathbb{E}_T[w(F_T)] \leq \mathbb{E}[\sum_{x,y \in F^*} \mathrm{dist}_T(x, y)] \leq O(dh) \sum_{x,y} \|x - y\| = O(dh) \cdot w(F^*)$$

作业：$(1+\epsilon)$-近似欧氏最小生成树

- http://cssyb.openjudge.cn/24hw14/
- Deadline: 4月17日

对比

- 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，对于高维也适用

* 分析

考虑最小的同时包含x, y的□，即，也直接对应树上距离计算

对一路上的边权求和

Fact: 若$\mathrm{LCA}(x, y)$的□边长2^i，那么树上距离是$2 \cdot \sqrt{d} \sum_{j=0}^{i} 2^j = \Theta(\sqrt{d} \cdot 2^i)$

Claim： $\mathrm{dist}_T(x, y) \geq \|x - y\|$

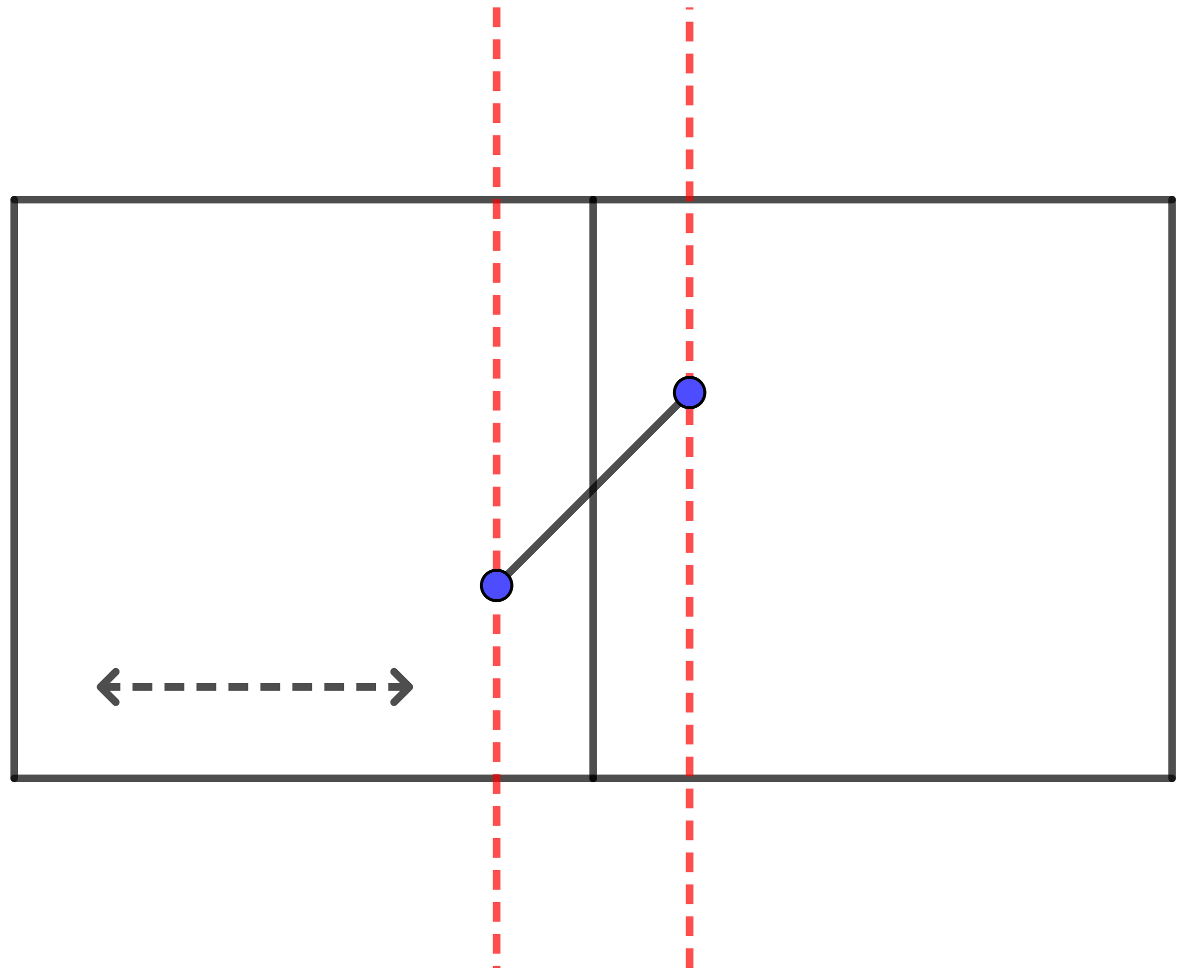
验证：设$\mathrm{LCA}(x, y)$的□边长2^i，那么$\|x - y\| \leq \sqrt{d}2^i \leq \sum_{j=0}^{i} \sqrt{d}2^j$

* 分析 (cont.)

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

- 回忆：考虑x, y，设LCA(x,y)的□边长2^i，则

$$\mathrm{dist}_T(x,y) = 2\cdot\sqrt{d}\sum_{j=0}^{i} 2^j = \Theta(\sqrt{d}\cdot 2^i)$$



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

也就是x, y形成的线段跨越了level-$(i+1)$的□

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

"线段跨hypercube"仅当存在某一维跨越hypercube

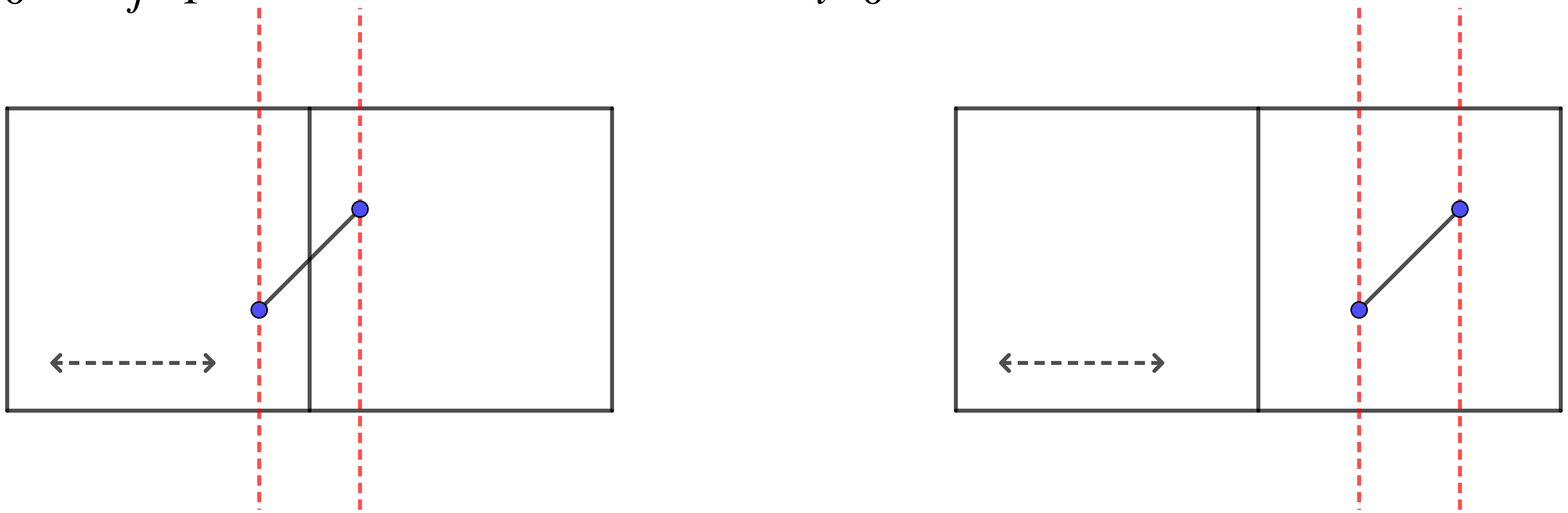
$$\le \sum_{i=0}^{h} \Pr[\exists \text{coordinate } j, \text{such that } x,y \text{ are separated}]\cdot O(\sqrt{d}2^i)$$

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

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

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

$$\leq \sum_{i=0}^{h} O(\sum_{j=1}^{d} |x_j - y_j| / 2^i \cdot \sqrt{d}2^i) \leq \sum_{i=0}^{h} O(d)\|x - y\| \leq O(dh) \cdot \|x - y\|$$



其他应用：数据流算法

- 该tree embedding可以实现成数据流算法
  - 在数据流开始前采样平移量v
  - 之后对每个数据点x计算$x+v$
  - 可用$O(d)$空间时间计算所处的格子及其所有祖先
- 思考：如何实现数据流估计MST的值算法?

数据流下无法计算解，因为存不下

其他应用：近似最小权匹配

- 最小权匹配问题：2n个$\mathbb{R}^d$上的点的最小欧氏权值的匹配
- 性质：在tree embedding上，子树内的点应该优先与子树内的匹配
- 因此：仅当一个子树有奇数个后继时，需要将其跨子树匹配
  - 注意到：所有数据点都在叶子上，所以具体用哪个点匹配并不重要
- 思考：算法是什么?

其他讨论

- 可以改进到$O(\log n)$的distortion，但构造复杂，且无法在数据流实现
  - 例如： https://www.cs.jhu.edu/~mdinitz/classes/ApproxAlgorithms/Spring2019/Lectures/lecture15.pdf
  - 我们介绍的算法的本质优势：data oblivious
- $\log n$的distortion也是紧的，不可改进的
- $\mathbb{E}[\mathrm{dist}_T(x, y)]$的保证，即无法保证每对点同时有distortion界，也是必须的