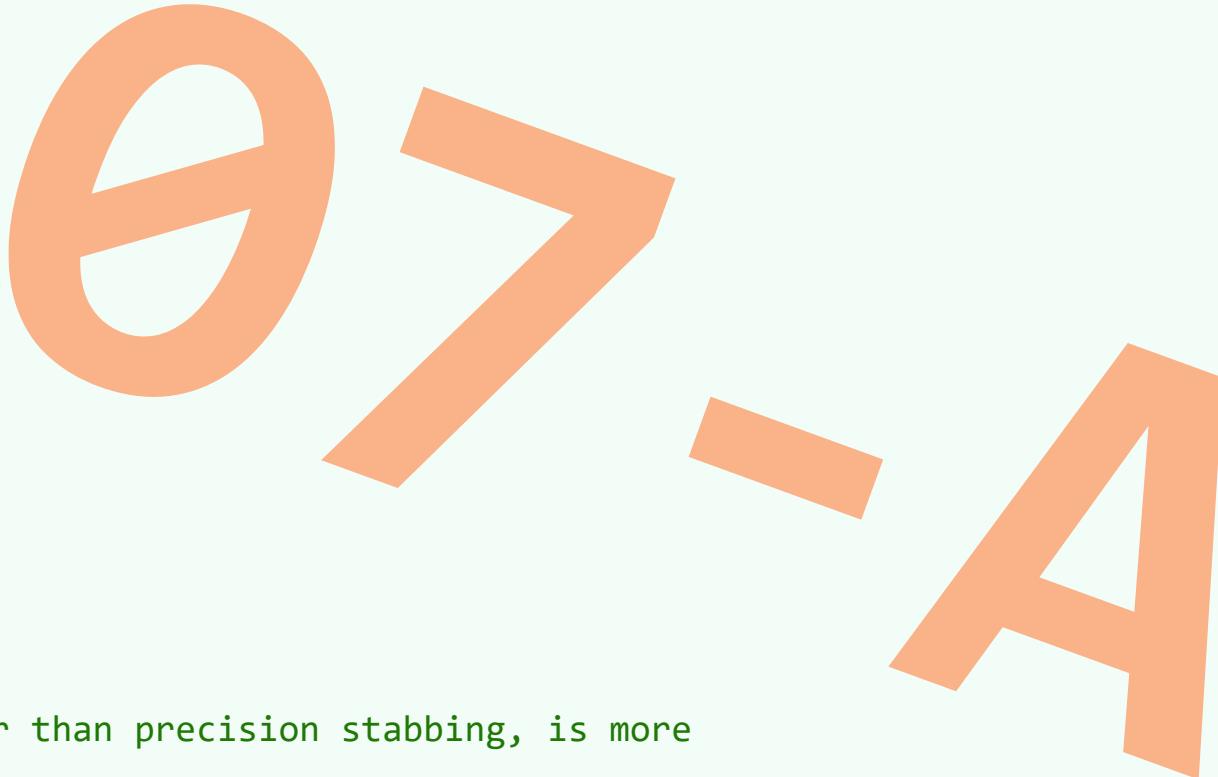


搜索树应用

区间树



Your instinct, rather than precision stabbing, is more
about just random bludgeoning.

邓俊辉

deng@tsinghua.edu.cn

穿刺查询/Stabbing Query

❖ 沿x轴给定一组区间：

$$S = \{s_i = [x_i, x'_i] \mid 1 \leq i \leq n\}$$

对于任何点 q_x ，找出所有包含它的区间：

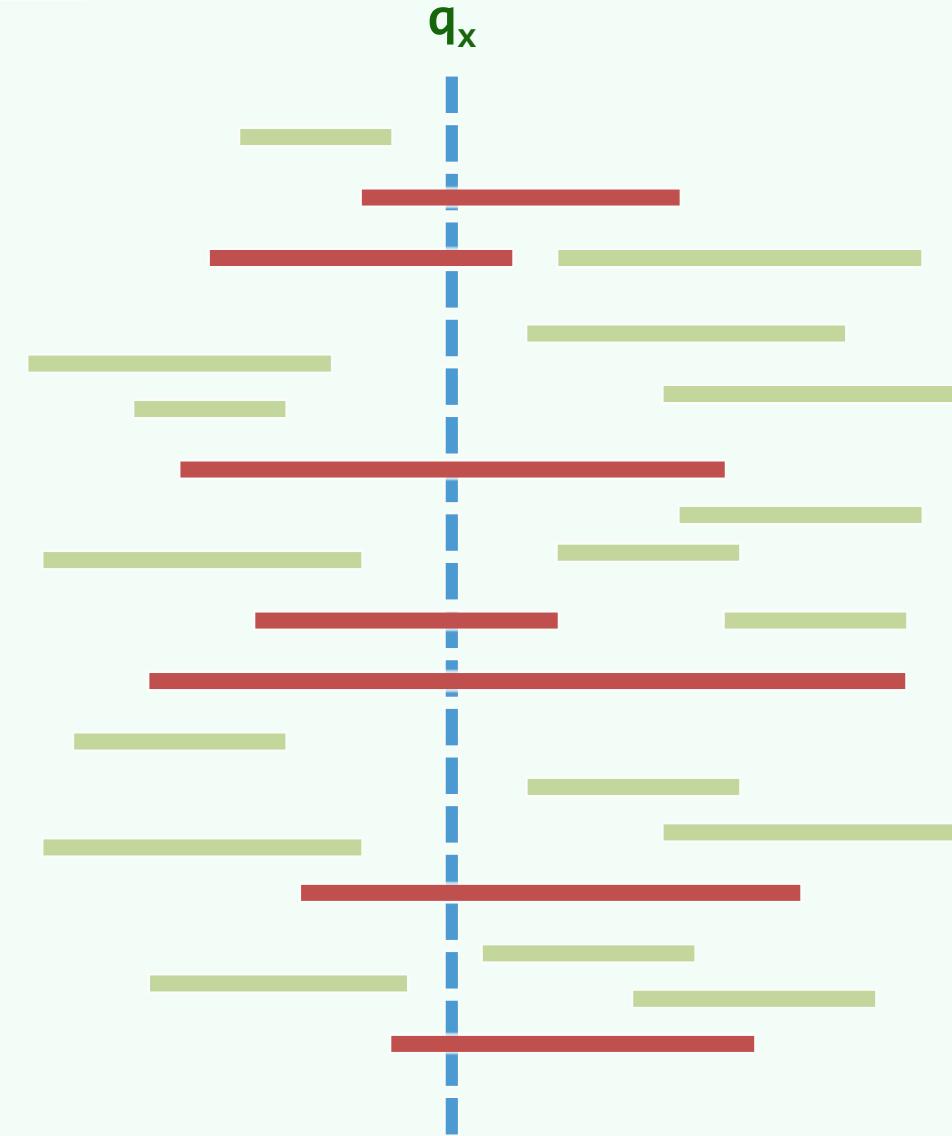
$$S(q_x) = \{s_i = [x_i, x'_i] \mid x_i \leq q_x \leq x'_i\}$$

❖ 蛮力解法不值一提

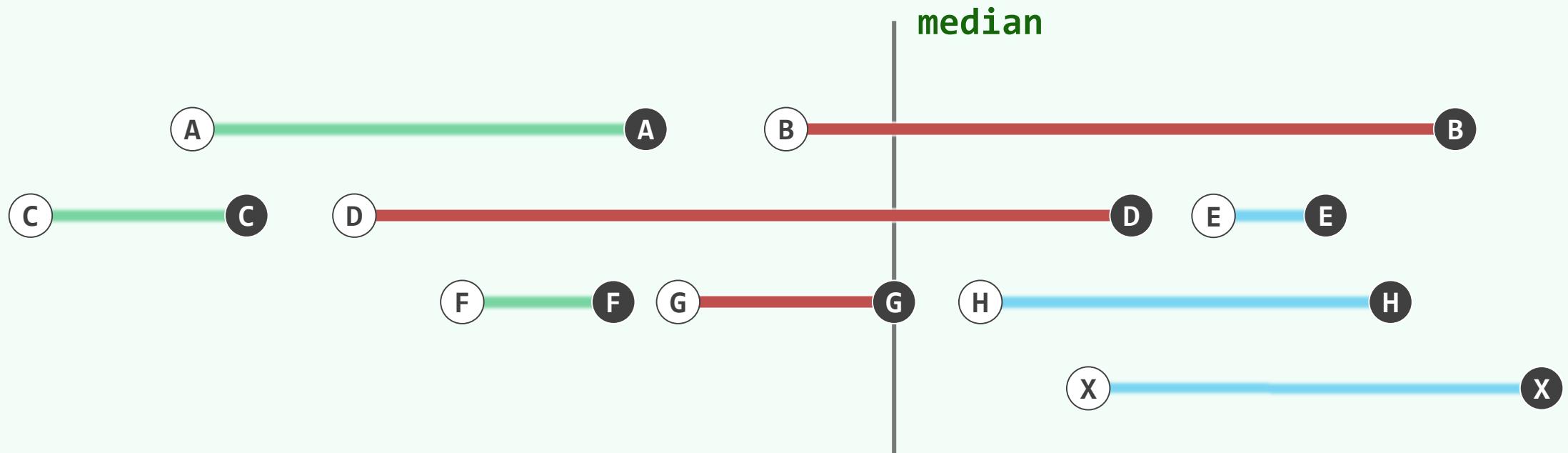
没有利用区间集相对固定的条件

❖ 将区间集预处理成一棵区间树 (Interval Tree)

可以得到一个高效的在线查询算法...



中位数

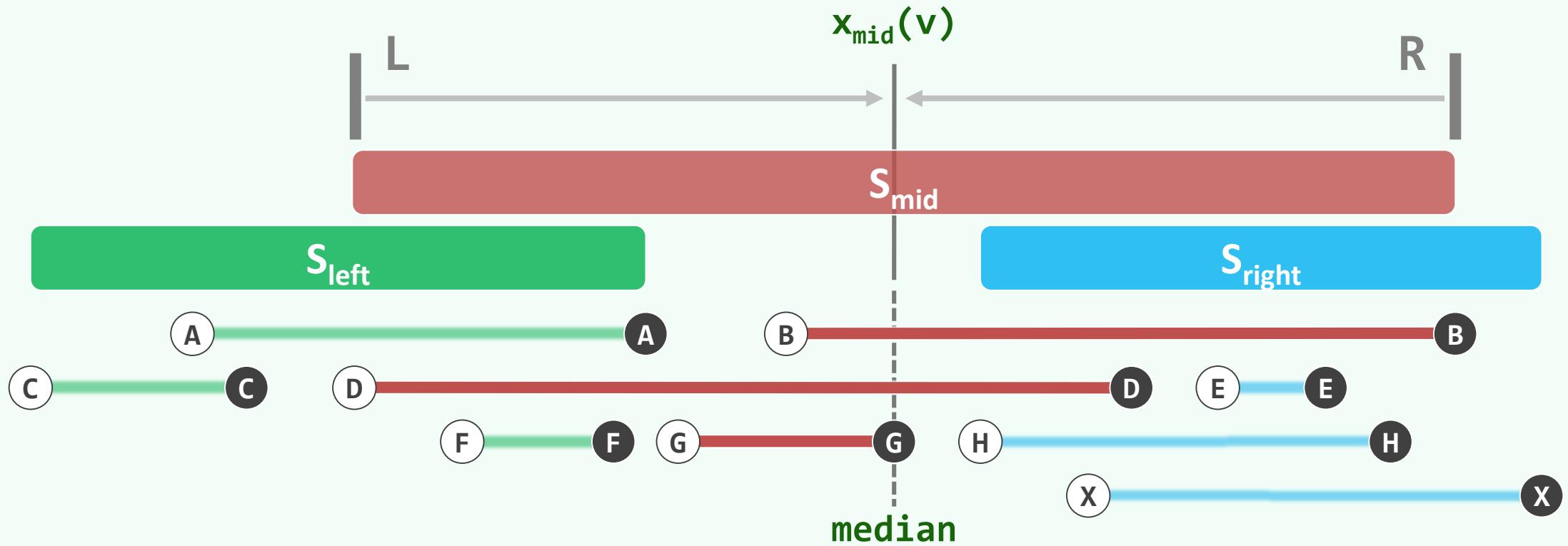


❖ 令所有区间的端点构成集合 $P = \partial S$

不失一般性，假定 $|P| = 2n$

❖ 取其中的中位数 $x_{mid} = median(P)$

分而治之



❖ 所有区间于是分为三类：

$$S_{mid} = \{ S_i \mid x_i \leq x_{mid} \leq x'_i \}$$

$$S_{left} = \{ S_i \mid x'_i < x_{mid} \}$$

$$S_{right} = \{ S_i \mid x_{mid} < x_i \}$$

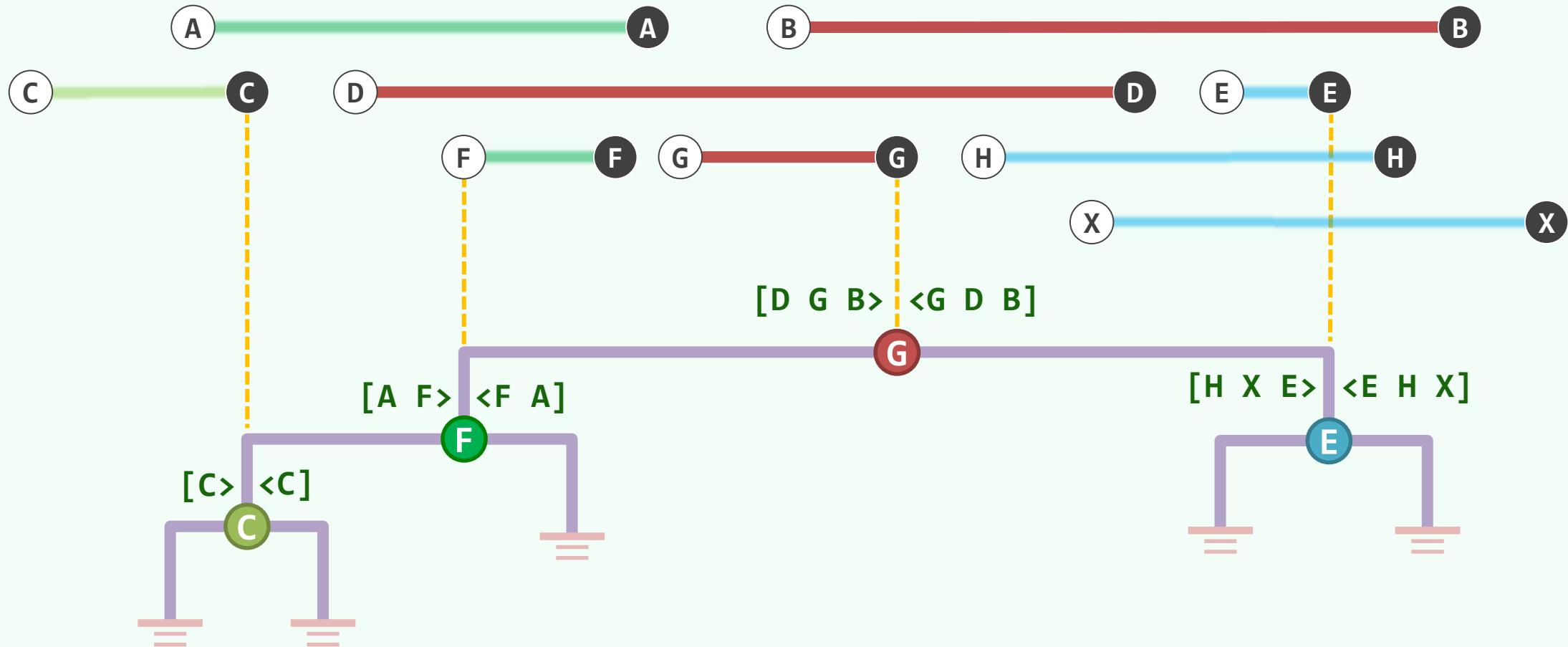
❖ S_{left} 和 S_{right} 可递归地同样划分，直至无需划分

渐近平衡: $\theta(\log n)$ 深度

$\max\{ |S_{left}|, |S_{right}| \} \leq n/2$

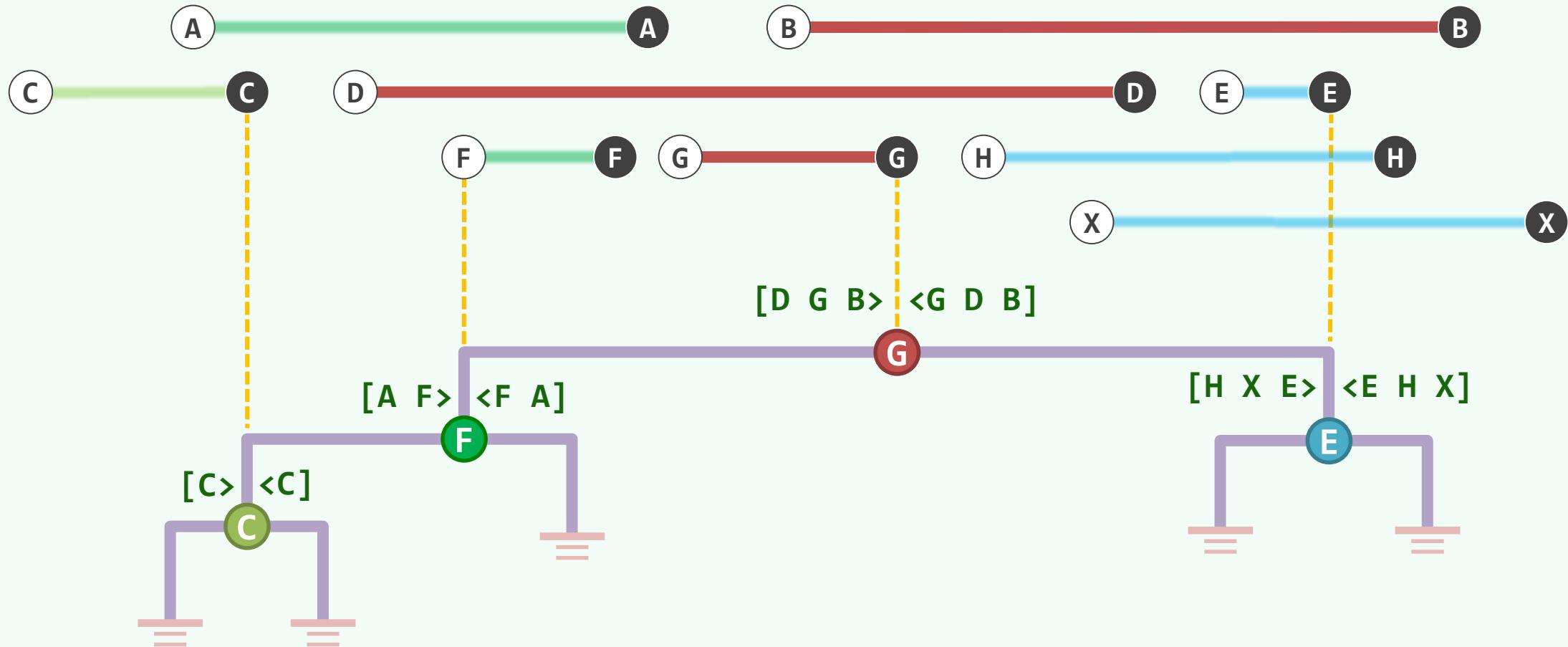
Best case: $|S_{mid}| = n$

Worst case: $|S_{mid}| = 1$



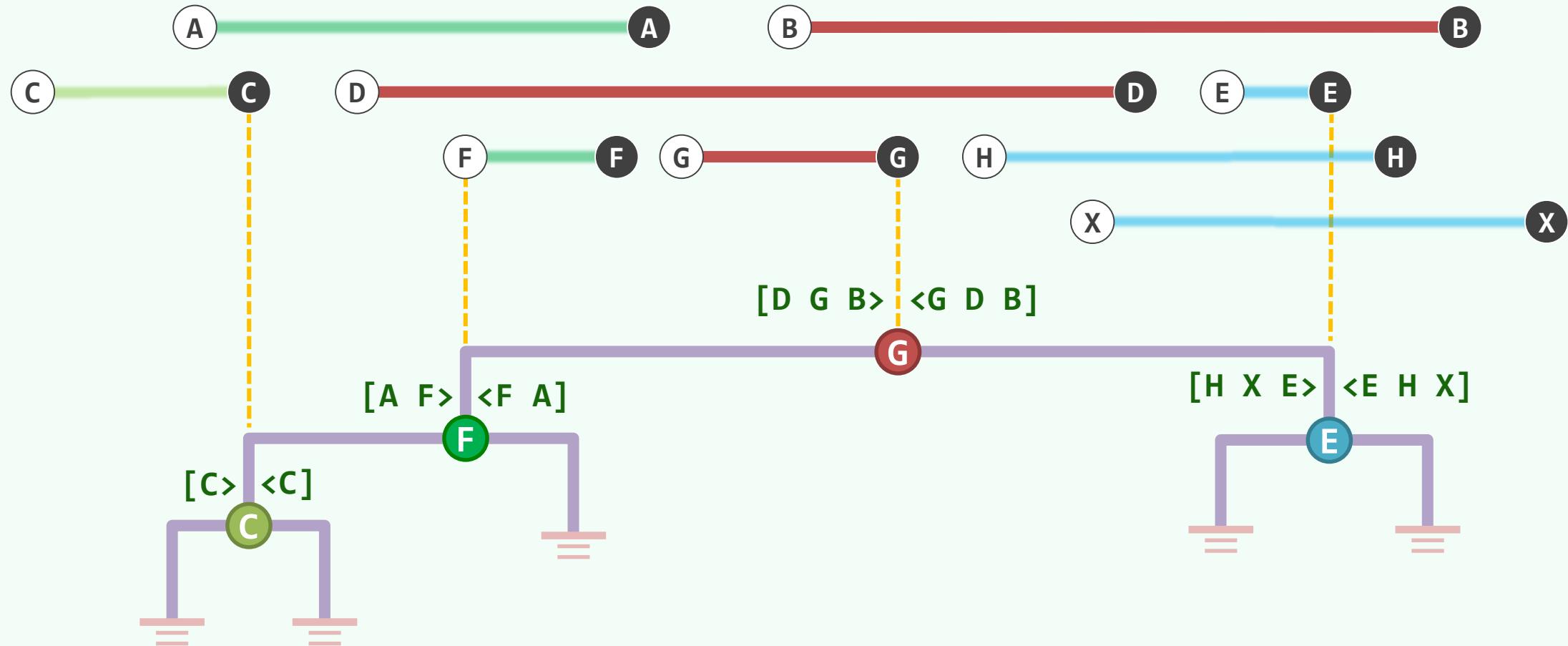
关联列表/Associative Lists

❖ S_{mid} 中的区间，分别按左、右端点的次序，由外向内地排成一对列表：L-list和R-list



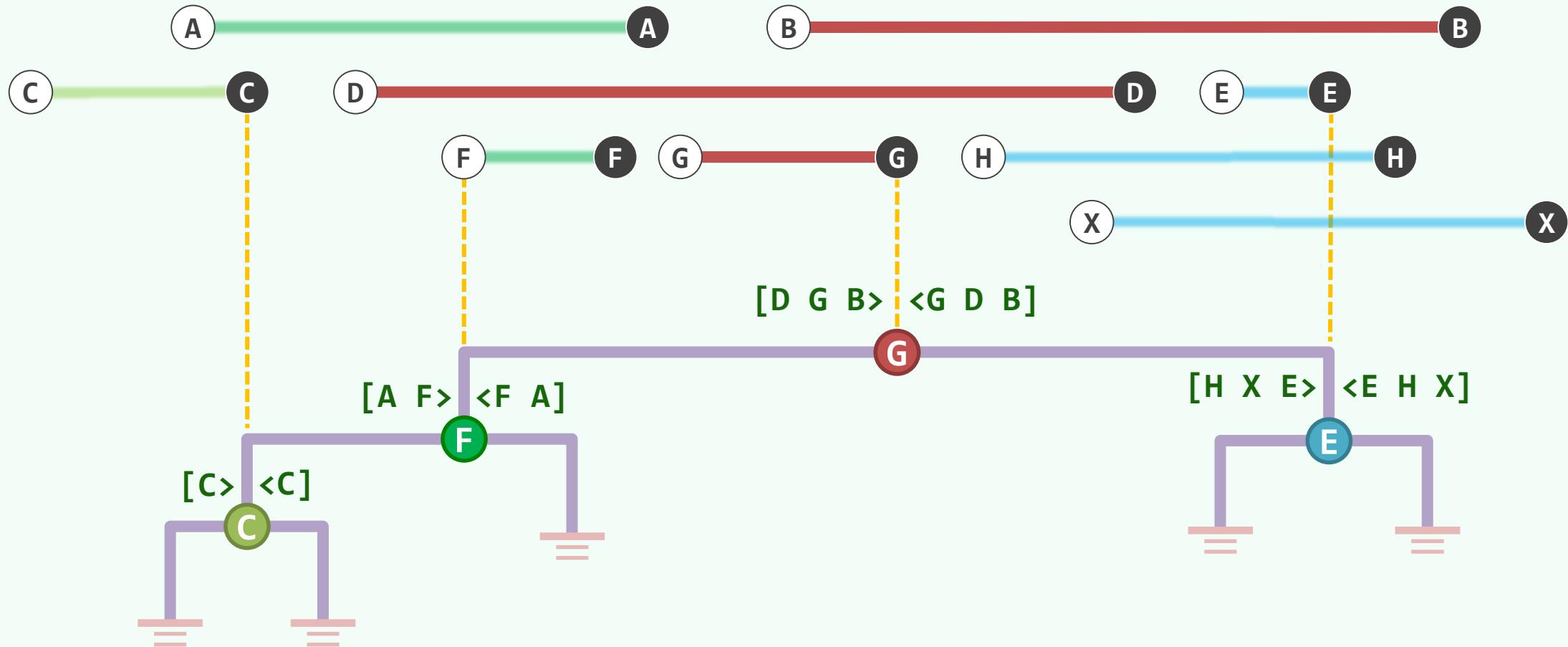
总共占用 $O(n)$ 空间

❖ 该树自身占用 $O(n)$ 空间； 输入的每个区间各存放了两份，总共也是 $O(n)$



构造只需 $O(n \log n)$ 时间

❖ 为此，需要避免反复地对端点排序...



查询算法: queryIntTree(v , q_x)

```
if ( ! v ) return; //递归基
```

```
if (  $q_x < x_{mid}(v)$  )
```

借助L-list, 从 $s_{mid}(v)$ 中分拣出包含 q_x 的区间

```
queryIntTree( lc(v), q_x )
```

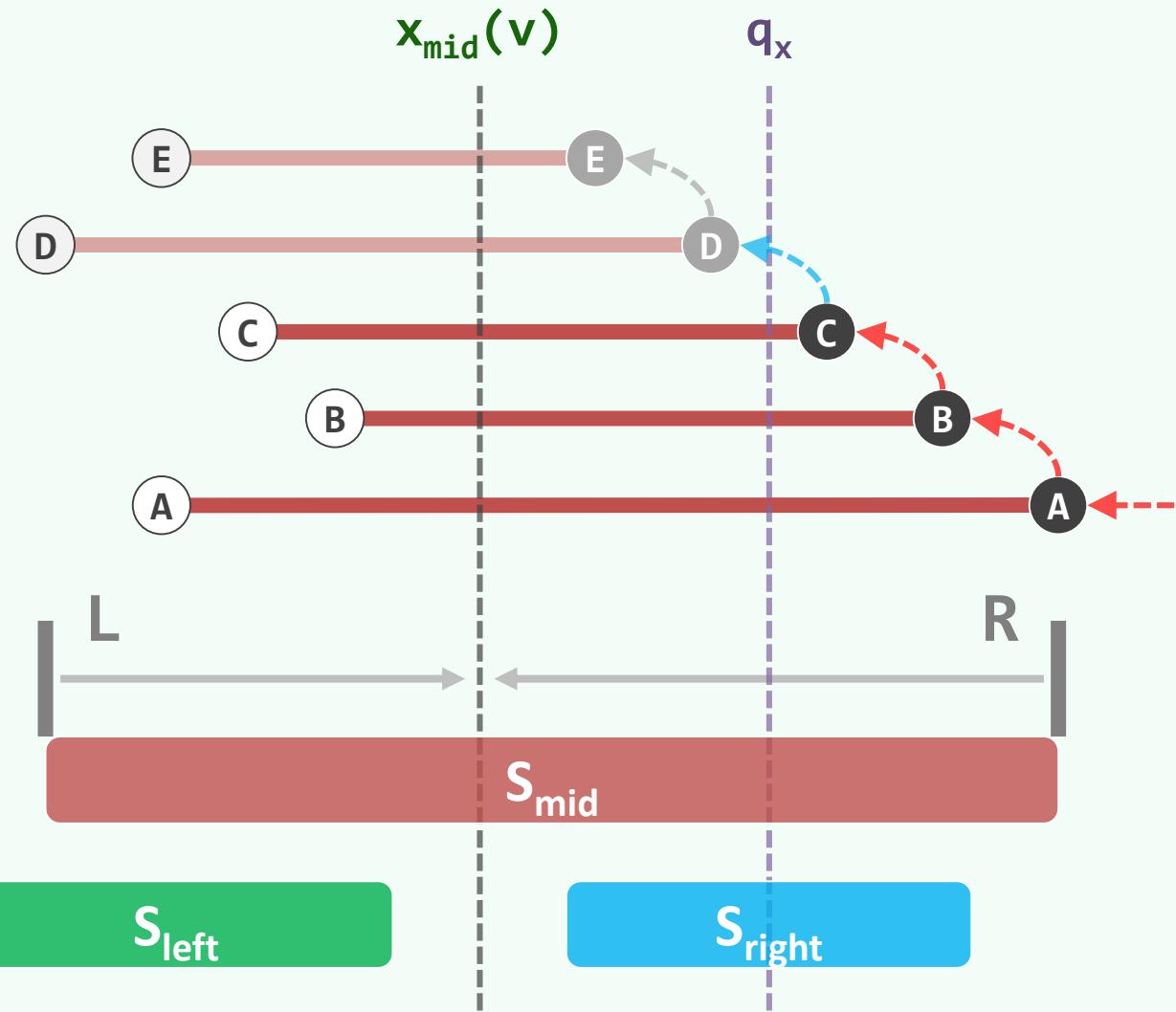
```
else if (  $x_{mid}(v) < q_x$  )
```

借助R-list, 从 $s_{mid}(v)$ 中分拣出包含 q_x 的区间

```
queryIntTree( rc(v), q_x )
```

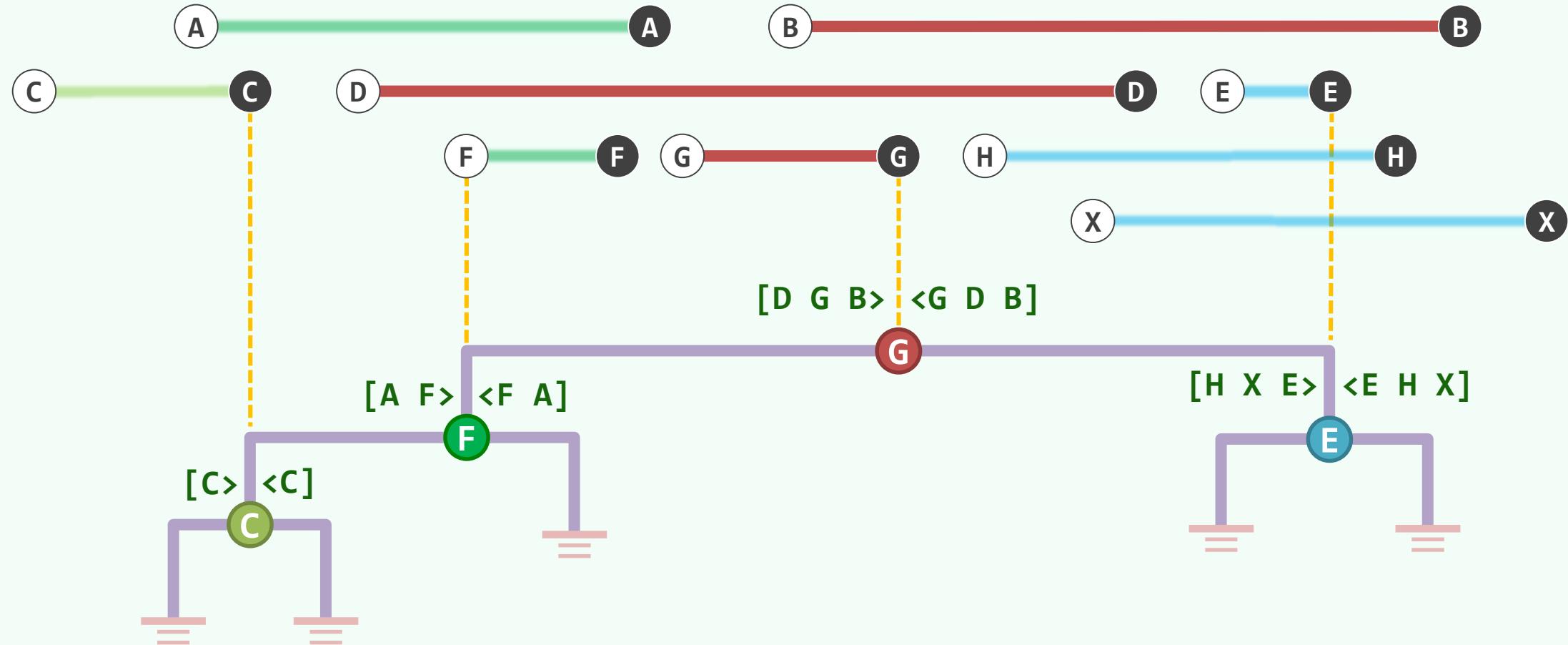
```
else //概率无限接近于0
```

直接报告 $s_{mid}(v)$



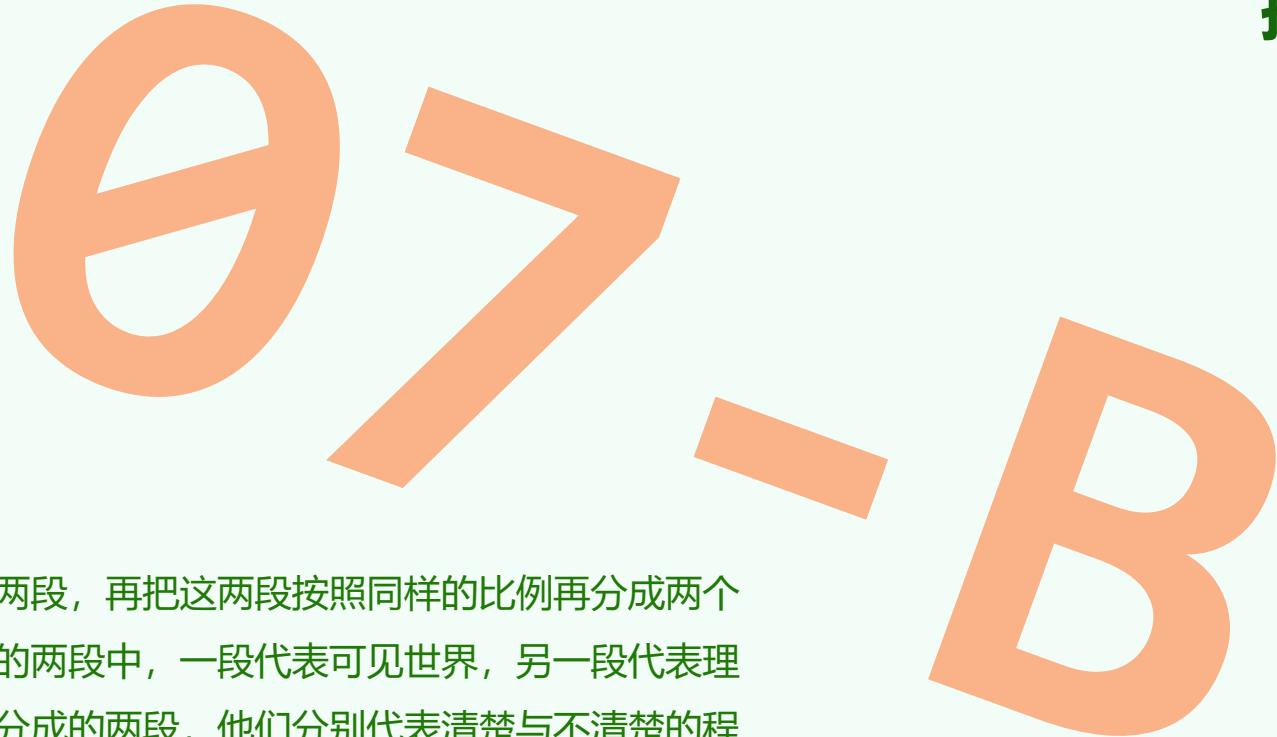
$\Theta(r + \log n)$ 查询时间

❖ 线性递归，只会访问 $\Theta(\log n)$ 个节点；对L/R-List的访问：累计 $\Theta(r)$ 个元素成功， $\Theta(\log n)$ 个失败



搜索树应用

线段树



把一条线分割成不相等的两段，再把这两段按照同样的比例再分成两个部分。假设第一次分出来的两段中，一段代表可见世界，另一段代表理智世界。然后再看第二次分成的两段，他们分别代表清楚与不清楚的程度，你便会发现，可见世界那一段的第一部分是它的影像

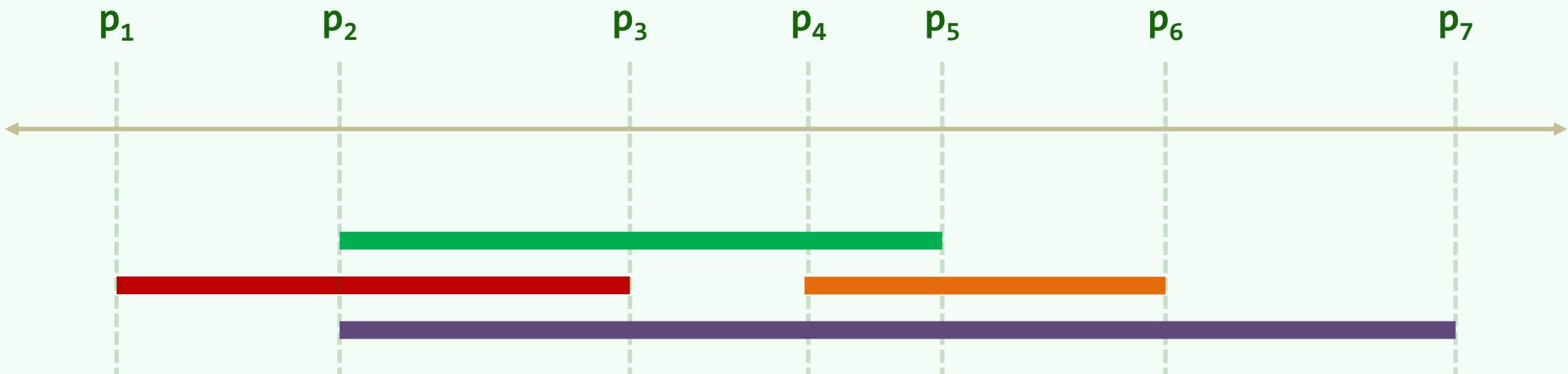
邓俊辉

deng@tsinghua.edu.cn

基本区间/Elementary Interval

❖ 任给x-轴上的n段区间: $I = \{ [x_i, x'_i] \mid i = 1, 2, 3, \dots, n \}$

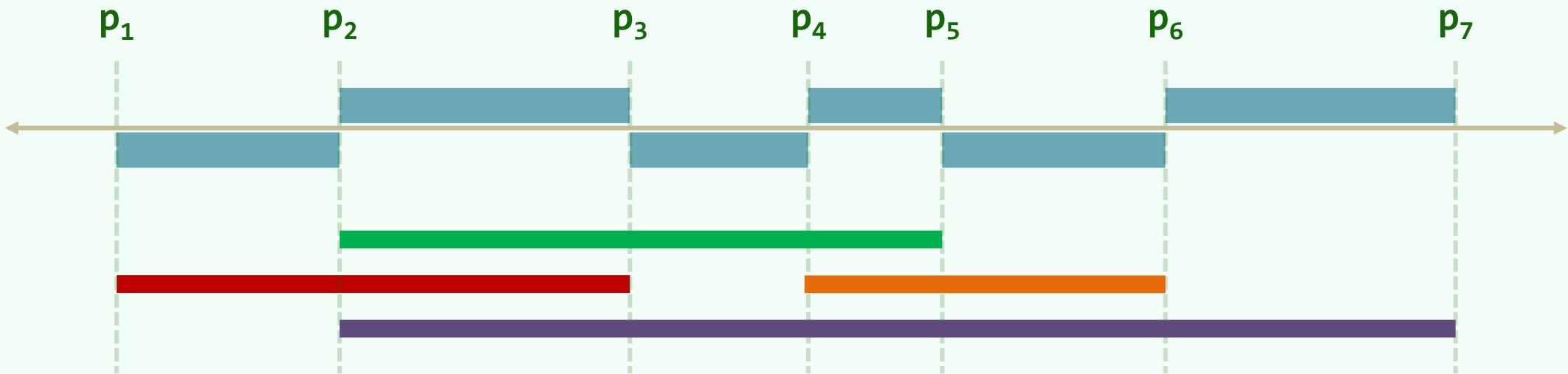
❖ 将它们的端点排序为: $\{ p_1, p_2, p_3, \dots, p_m \}, m \leq 2n$



❖ 于是, 便得到m+1段基本区间/EI: $(-\infty, p_1], (p_1, p_2], (p_2, p_3], \dots, (p_{m-1}, p_m], (p_m, +\infty]$

离散化/Discretization

- ❖ 观察：同一段EI内的任何位置，穿刺查询的输出必然完全一样
- ❖ 于是，只要将所有所有EI预处理为有序向量，并使每段EI记录其对应的查询输出，那么...



- ❖ ...一旦确定穿刺位置，便可快速地完成查询：二分查找 + 输出结果 $\text{/\!/} \mathcal{O}(\log n + r)$

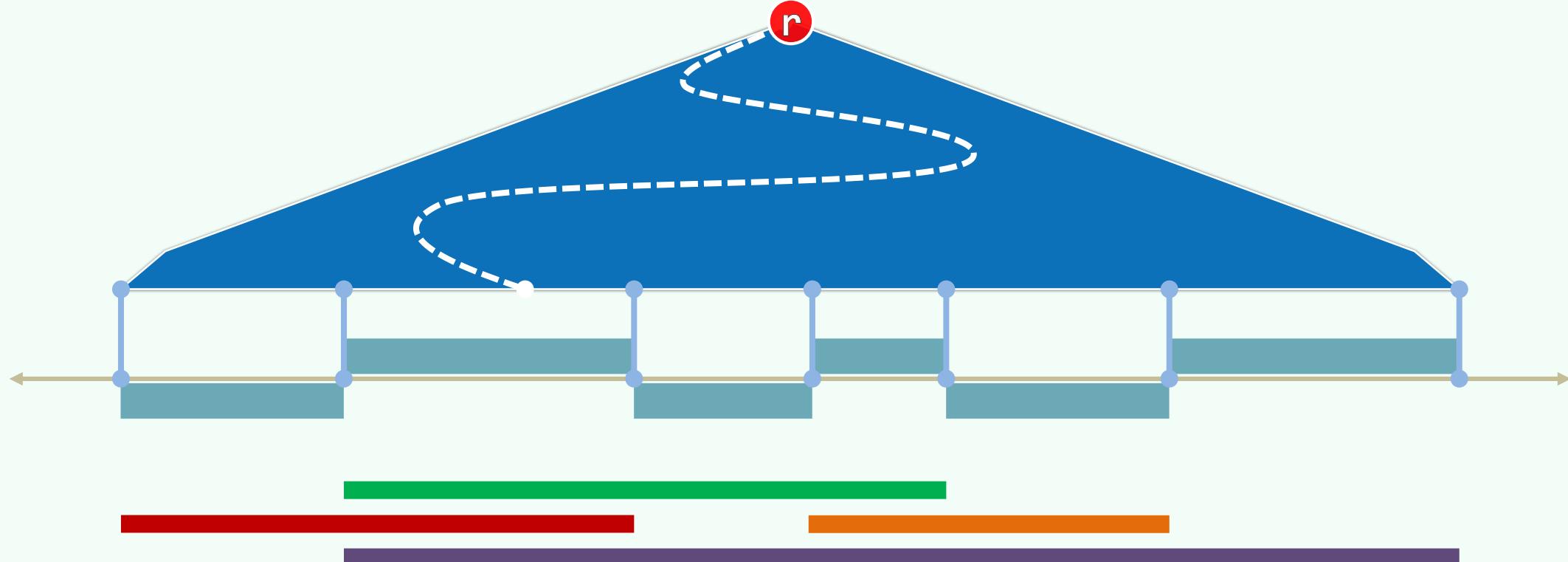
空间爆炸

- ❖ 在最坏情况下，输入的每段区间都会横跨 $\Omega(n)$ 段EI，总共需要 $\Omega(n^2)$ 附加空间



将有序向量升级为BBST

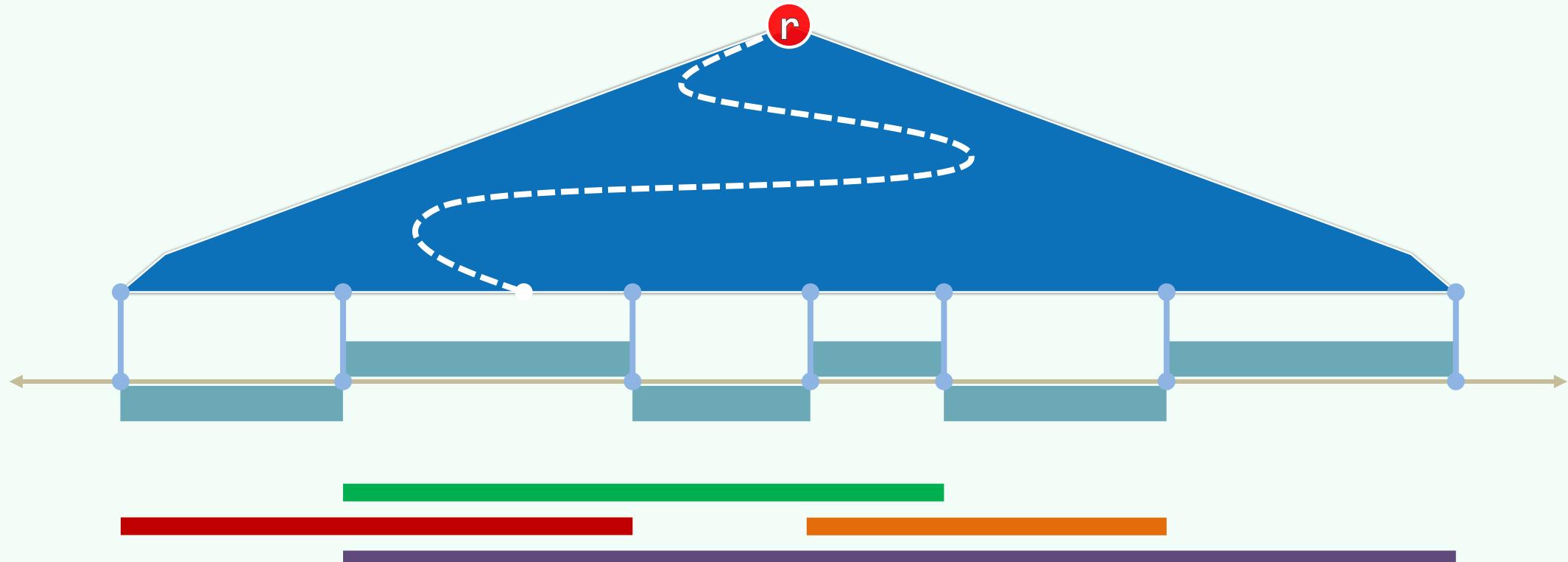
- ❖ 节点 v 的作用域 $R(v)$: 叶节点 v 的 $R(v)$, 就是其对应的EI; 内部节点 v 的 $R(v)$, 为其孩子作用域之并
- ❖ 每个叶节点 v 各自预先记录 $\text{Int}(v)$: 即横跨 $R(v)$ 的所有输入区间, 作为该EI对应的查询输出



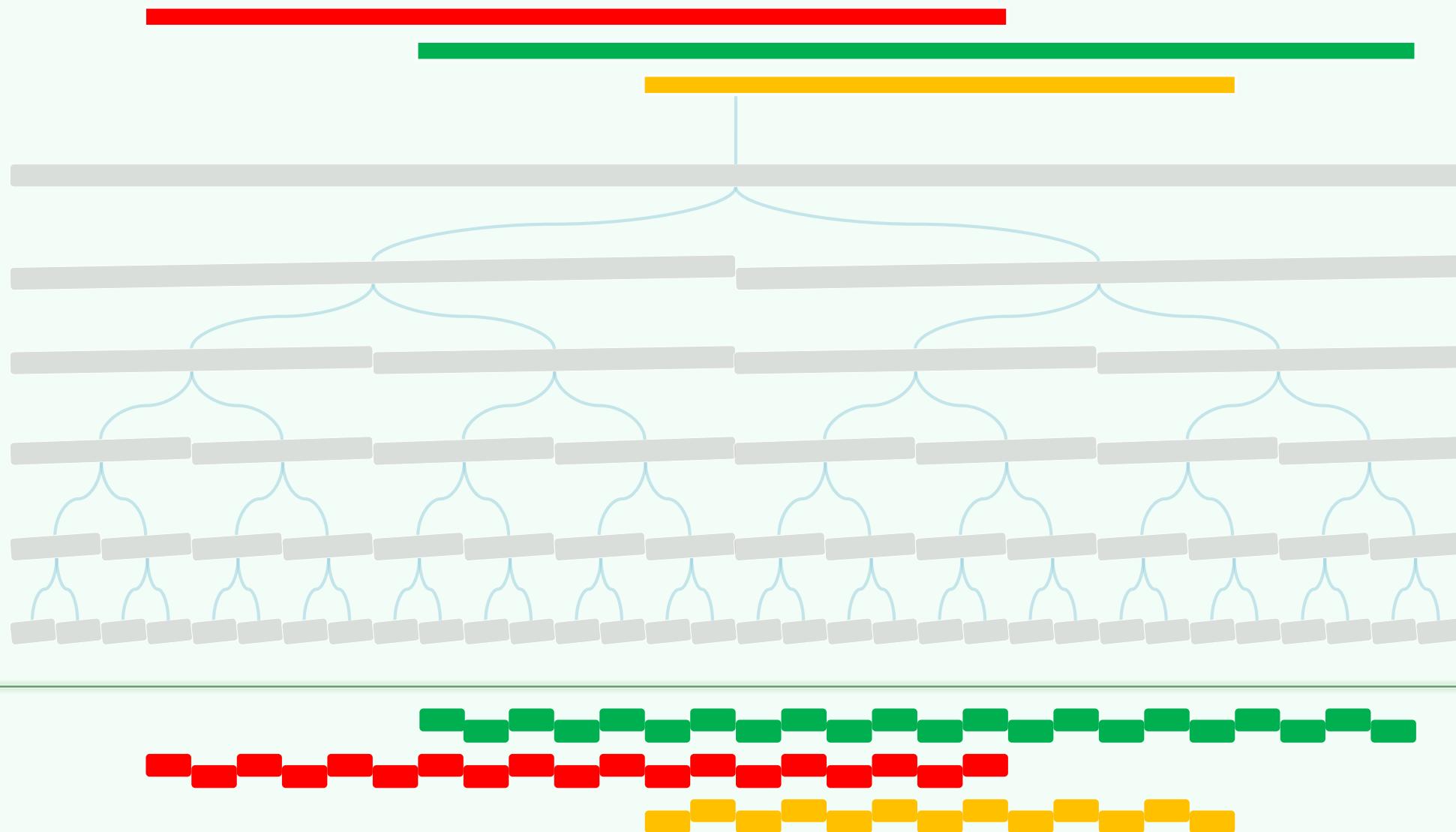
支持穿刺查询

❖ 对任何穿刺位置 q_x , 只需

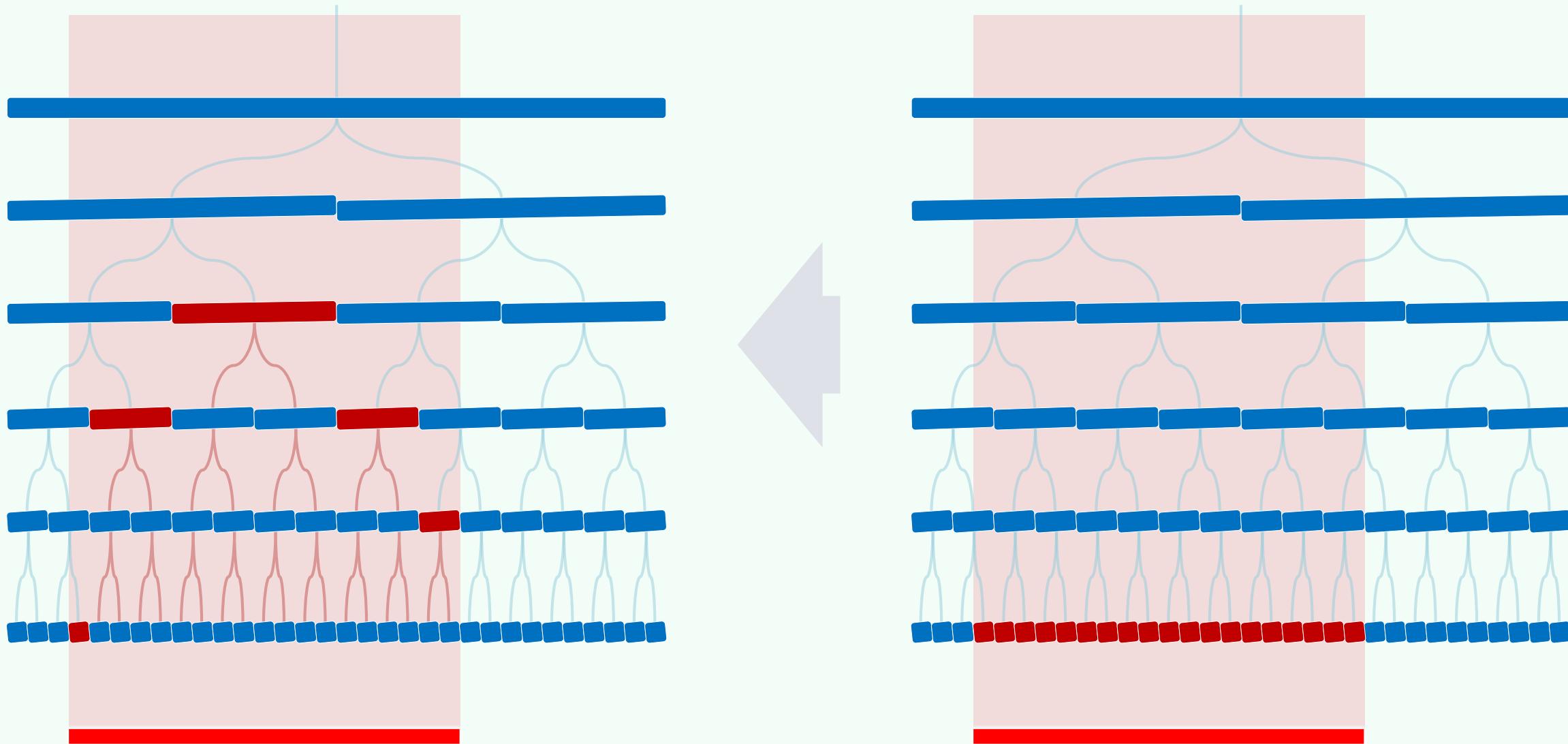
通过BBST::search(q_x)找到对应的叶节点 v , 并输出预先记录的集合Int(v) // $\mathcal{O}(\log n + r)$



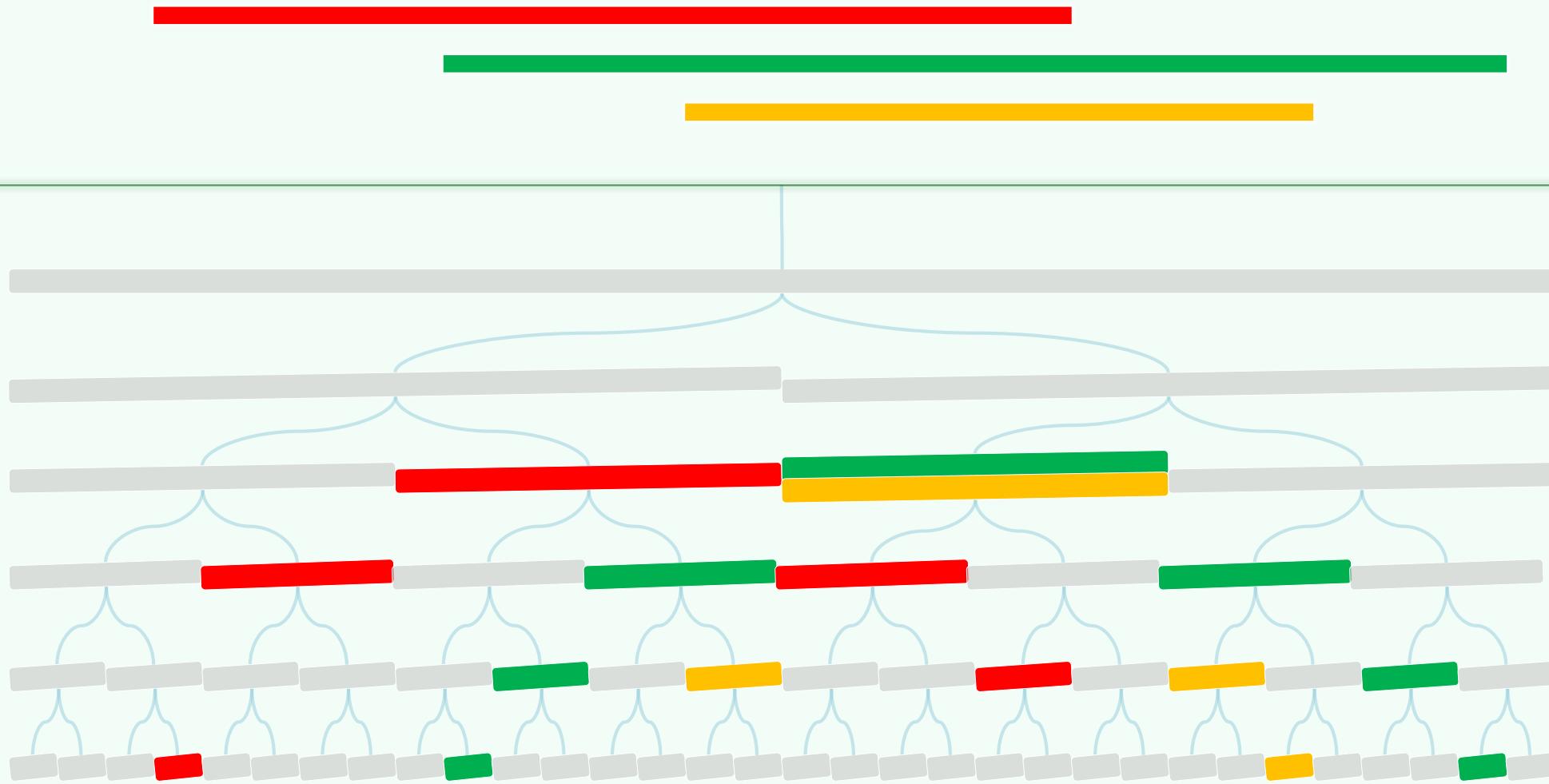
然而，最坏情况下仍需 $\Omega(n^2)$



通过贪心地合并，使每段输入区间至多被 $O(\log n)$ 个祖先记录



从而，空间复杂度降至 $\Theta(n \log n)$



构造算法: BuildSegTree(I)

对I中所有区间的端点排序 $\text{ // } \mathcal{O}(n \log n)$

确定所有的EI

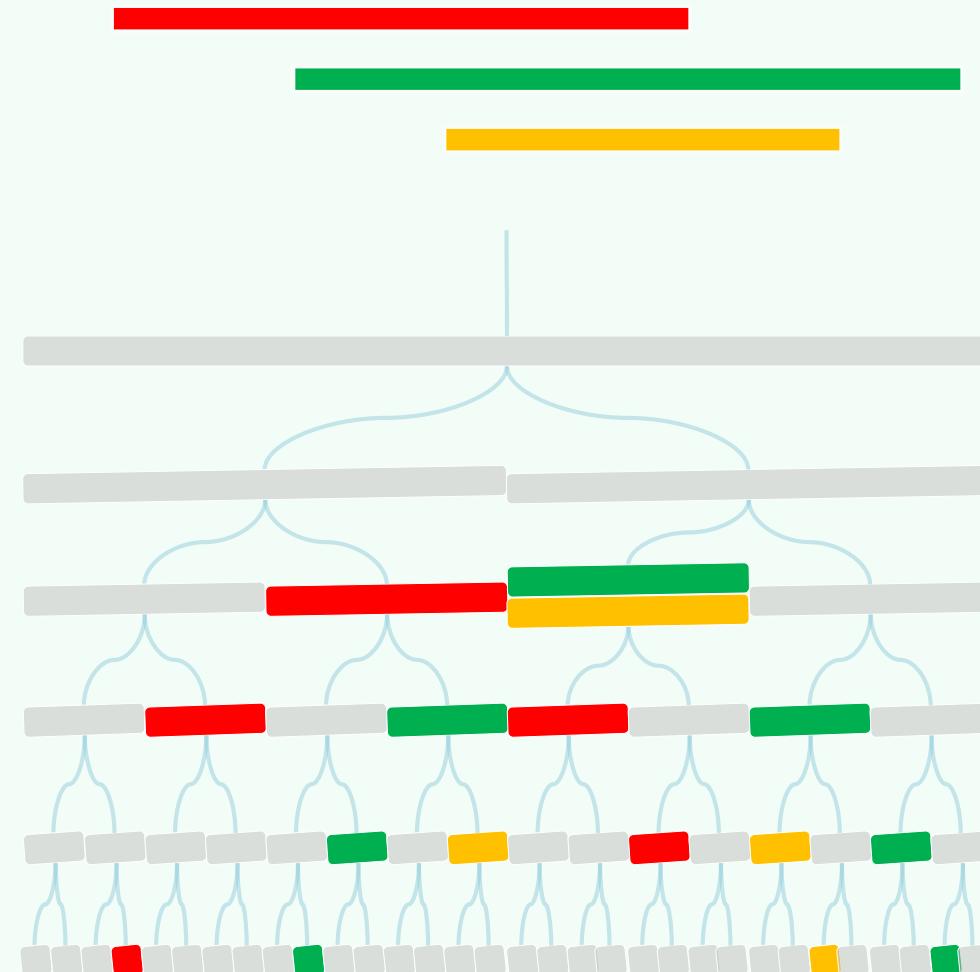
并将它们组织为一棵BBST $\text{ // } \mathcal{O}(n)$

自底而上, 确定

每个节点v的作用域R(v) $\text{ // } \mathcal{O}(n)$

for (I中的每一段区间s)

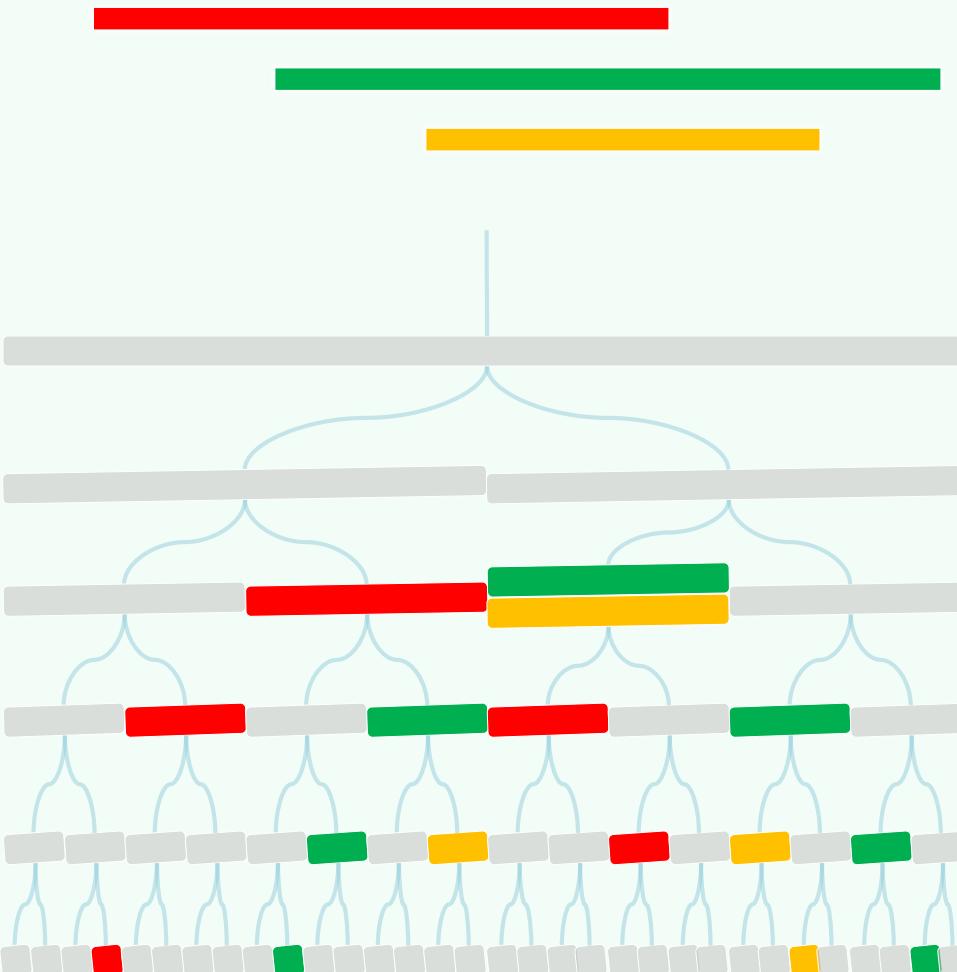
InsertSeg(T.root, s)



构造算法: InsertSeg(v, s)

```
if ( R(v) ⊆ s ) //greedy by top-down  
    store s at v and return;  
  
if ( R( lc(v) ) ∩ s ≠ ∅ ) //recurse  
    InsertSeg( lc(v), s );  
  
if ( R( rc(v) ) ∩ s ≠ ∅ ) //recurse  
    InsertSeg( rc(v), s );
```

- ❖ 在树中的每一层，至多访问4个节点
(2个存放s的副本，另2个递归)
- 故每段区间s的插入，只需 $\Theta(\log n)$ 时间



查询算法: $\text{Query}(v, q_x)$

report all the intervals in $\text{Int}(v)$

if (v is a leaf)

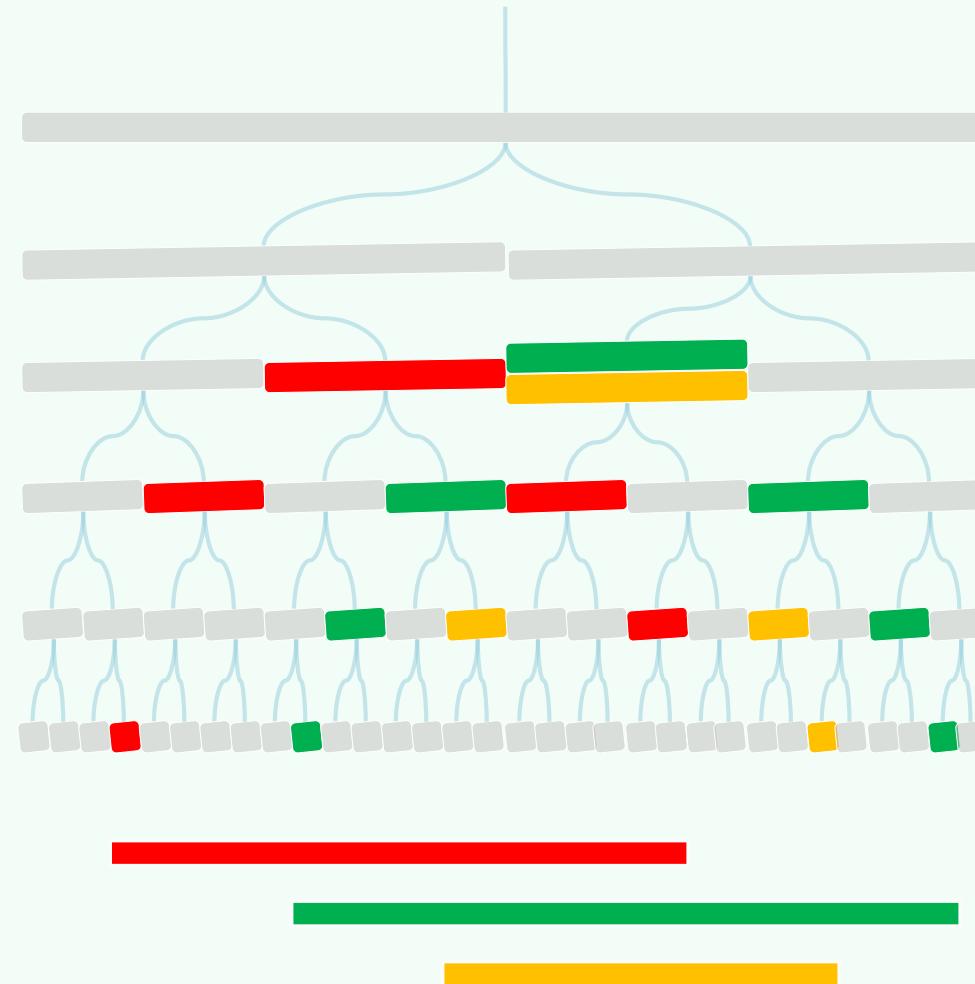
return

if ($q_x \in R(\text{lc}(v))$)

$\text{Query}(\text{lc}(v), q_x)$

else // $q_x \in R(\text{rc}(v))$

$\text{Query}(\text{rc}(v), q_x)$



查询时间: $\Theta(r + \log n)$

❖ 树中的每层仅需访问一个节点

累计 $\Theta(\log n)$ 个

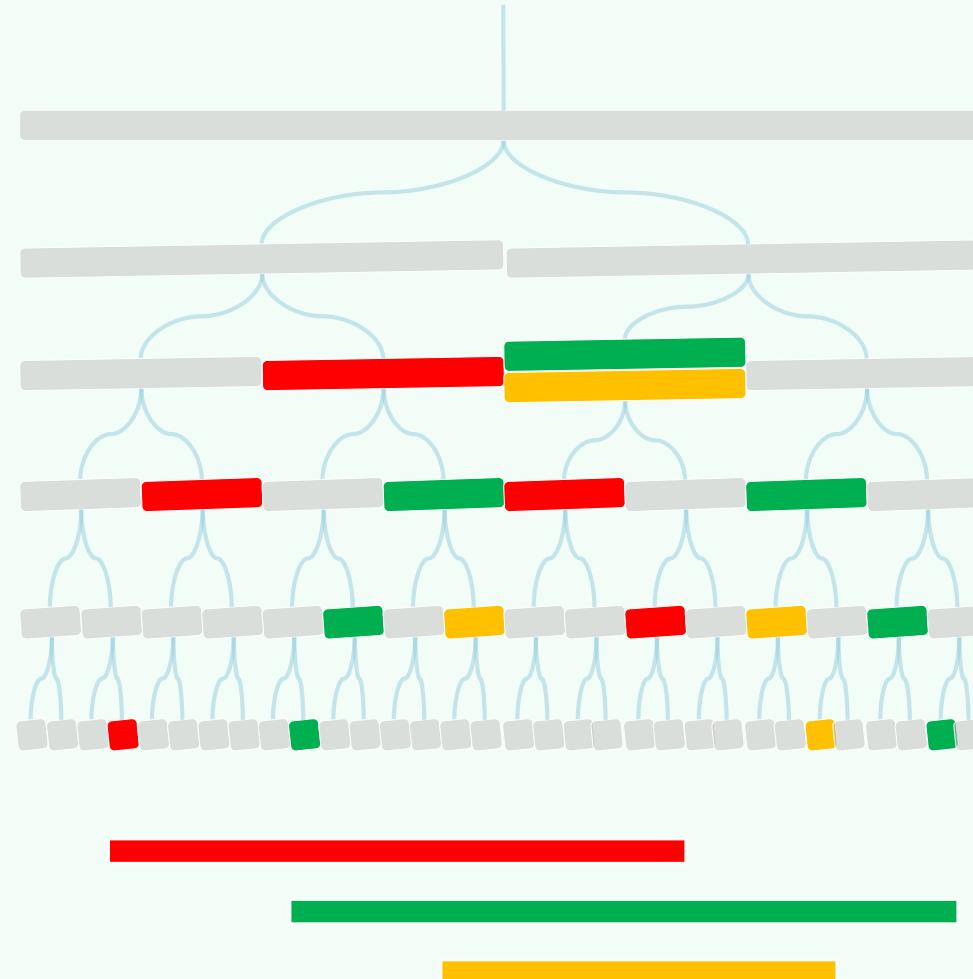
❖ 在每个节点 v 处

为输出 $\text{Int}(v)$ 所需的时间仅为

$$1 + |\text{Int}(v)| = \Theta(1 + r_v)$$

❖ 因此, 为输出所有命中区间

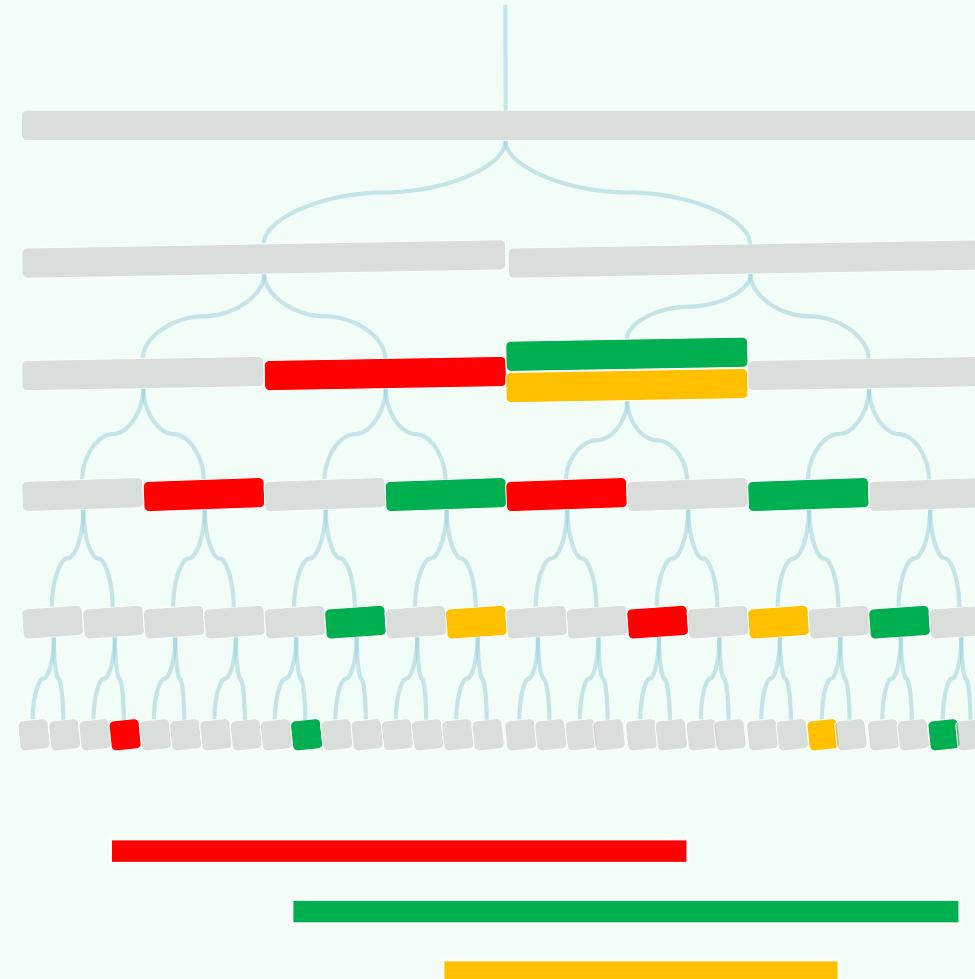
仅需 $\Theta(r)$ 时间



结论

❖ 任给x轴上的n段区间

- 可以在 $\Theta(n \log n)$ 时间内构造出
- 一棵占用 $\Theta(n \log n)$ 空间的线段树
- 借助它，每次穿刺查询都能
在 $\Theta(r + \log n)$ 时间内完成



搜索树应用

范围查询：一维

e> - c₁

邓俊辉

deng@tsinghua.edu.cn

你这个人太敏感了。这个社会什么都需要，唯独不需要敏感

1D Range Query

❖ 考查x轴上的n个点: $P = \{ p_1, p_2, p_3, \dots, p_n \}$



❖ 任给区间 $I = (x_1, x_2]$

- 计数/COUNTING: P 中有多少点落在其中?
- 报告/REPORTING: 列出 $I \cap P$ 中的所有点

❖ [Online] 通常 P 相对固定, 而 I 是不断随机更新的

❖ 如何将 P 预处理为适当的数据结构, 使得每次查询都能高效地完成?

Brute-Force

- ❖ 逐个地取出 P 中的点 p ，在 $\mathcal{O}(1)$ 时间内判断是否 $p \in (x_1, x_2]$

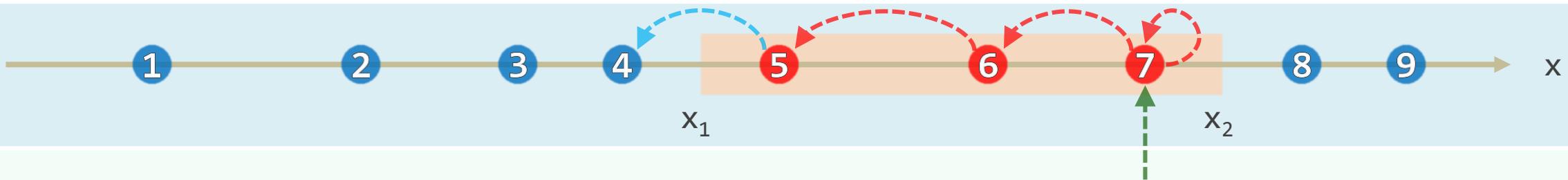


如此，可以在线性时间内完成每次查询

- ❖ 可以做得更快吗？貌似不可能，毕竟...
- ❖ 在最坏情况下会有 $\Omega(n)$ 个点命中
即便是列举出它们，也需要 $\Omega(n)$ 时间
- ❖ 然而，要是暂且不计列举输出的环节，仅考查搜索的环节呢？

Binary Search

❖ 将所有点预处理为一个有序向量 (并添加哨兵 $p_0 = -\infty$)



❖ 任给区间 $I = (x_1, x_2]$

- 通过二分查找确定 $t = \text{search}(x_2) = \max\{ i \mid p_i \leq x_2 \} // O(\log n)$

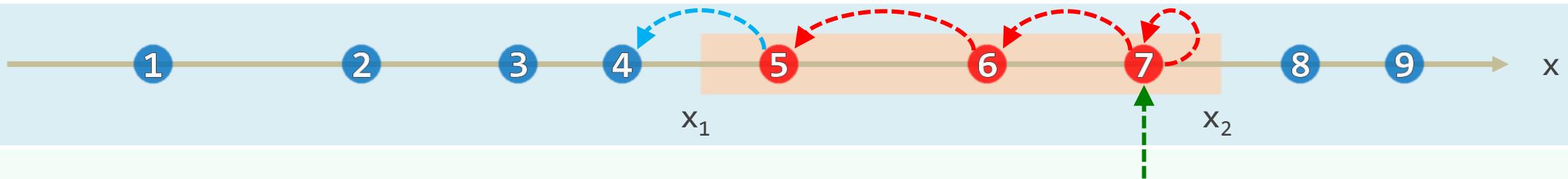
- 从 p_t 出发, 自后向前遍历向量, 直到在 p_s 处越出查询区间 I // $O(r)$

在 $(s, t]$ 内的每一个点, 都予输出 // 它们恰好就是所有的输出

- 返回命中计数 $r = t - s // \text{output size}$

Output-Sensitivity

❖ 如此，每次查询的结果，都可在 $\mathcal{O}(1 + r + \log n)$ 时间内报告出来



❖ 自然，借助二分查找， p_s 也可 $\mathcal{O}(\log n)$ 时间内确定

于是，如果仅仅着眼于计数查询，每次只需 $\mathcal{O}(\log n)$ 时间 //与 r 无关

❖ 关键是，上述策略如何推广到二维的范围查询？

甚至于，可否推广？

很遗憾，据我所知，还不能！

搜索树应用

范围查询：二维

θ> - C₂

昔者明王必尽知天下良士之名；既知其名，又知其數；既知其數，又知其所生。

邓俊辉

deng@tsinghua.edu.cn

Planar Range Query

❖ 给定平面点集 $P = \{ p_1, p_2, p_3, \dots, p_n \}$

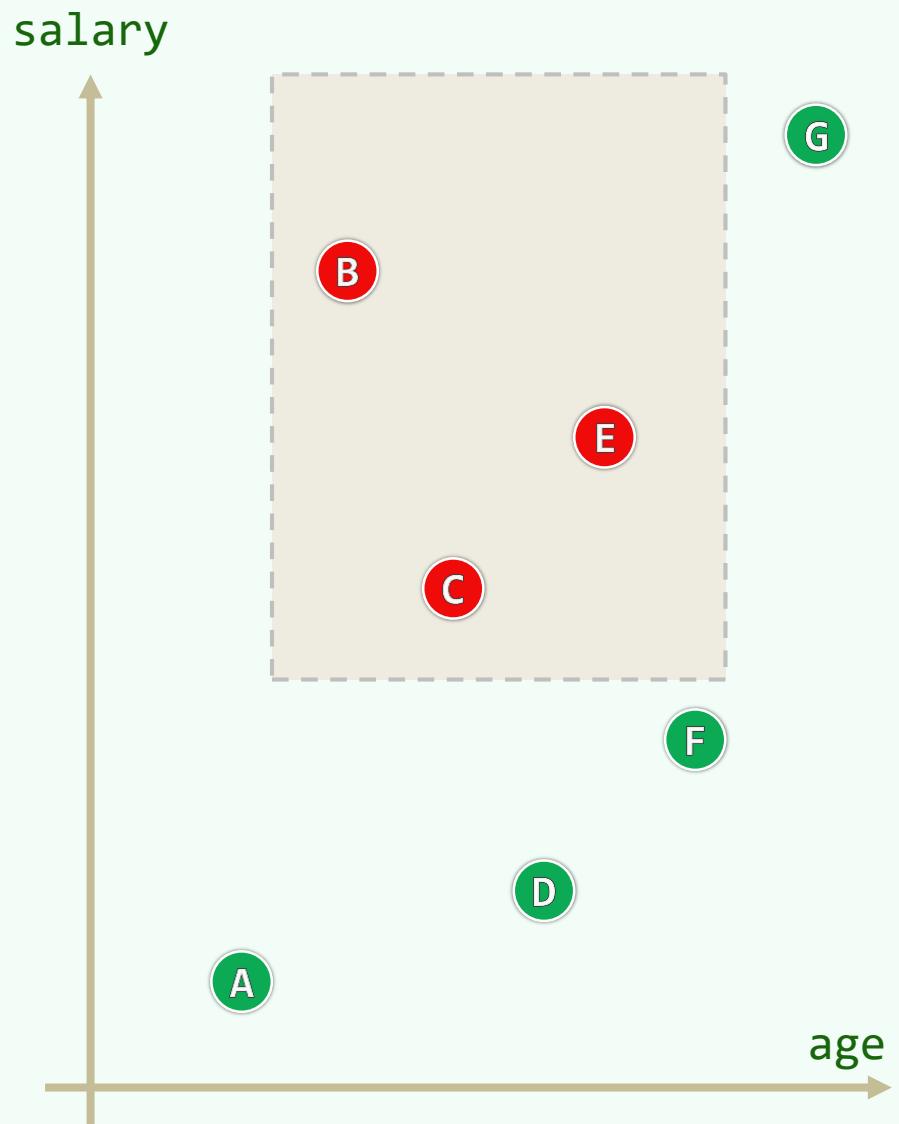
❖ 对任何矩形区域 $R = (x_1, x_2] \times (y_1, y_2]$

- 计数/COUNTING: $|R \cap P| = ?$
- 报告/REPORTING: $R \cap P = ?$

❖ 二分查找, 对此问题无能为力

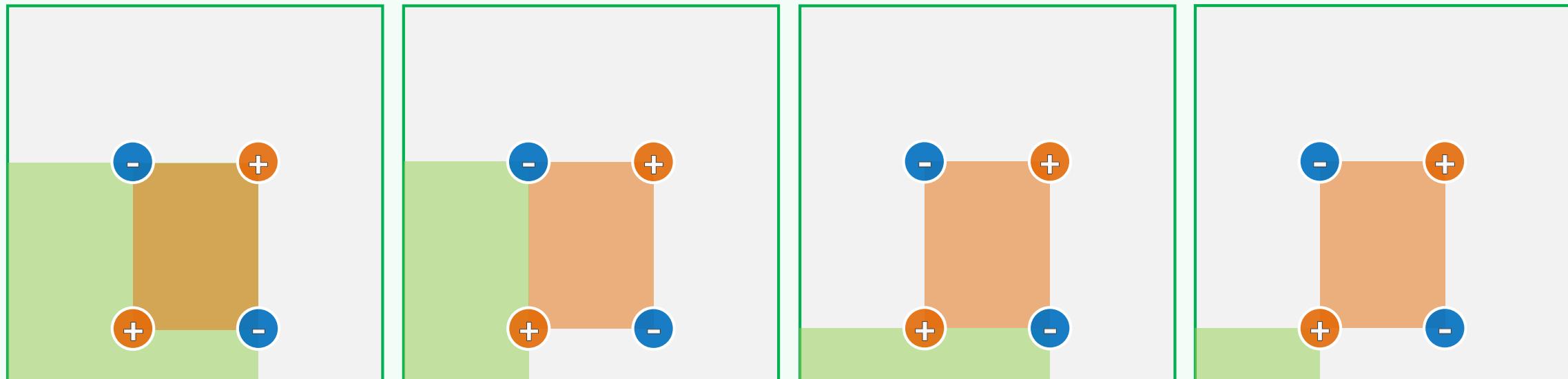
❖ 如果只考虑计数查询

或许可以借助**容斥原理**...



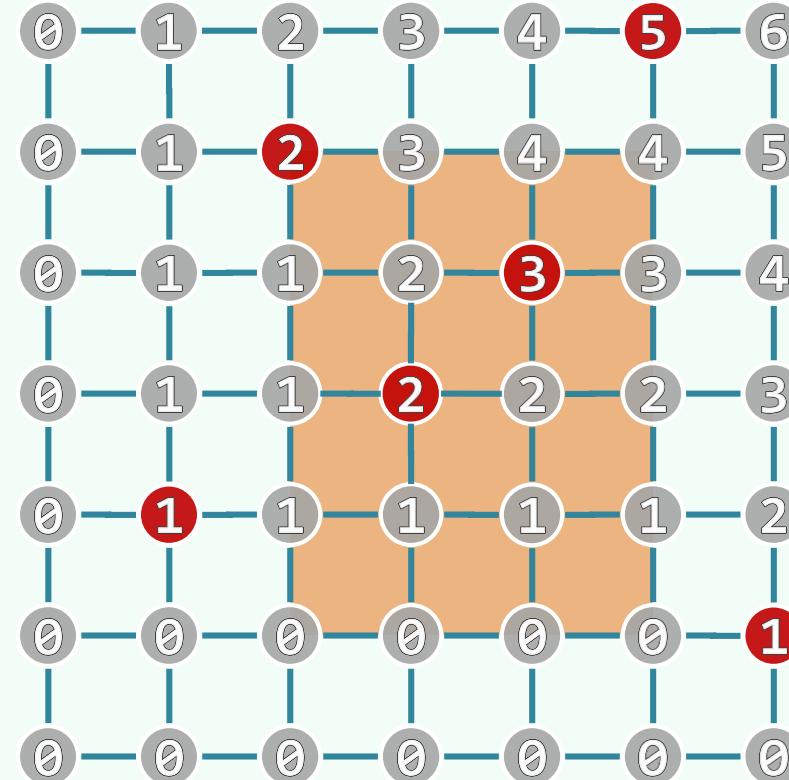
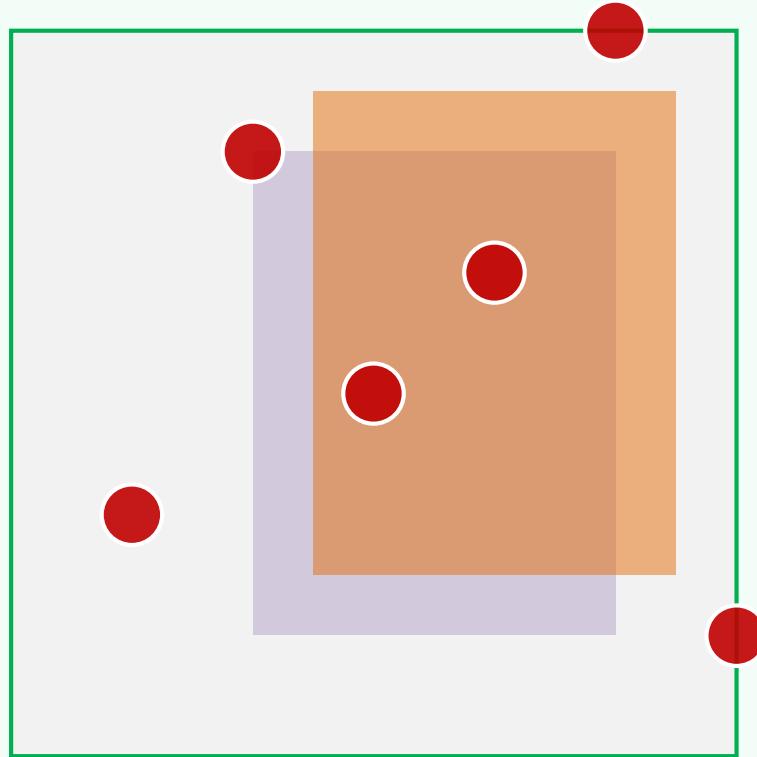
Preprocessing

- ❖ 对任何一点 (x, y) , 令 $n(x, y) = |((0, x] \times (0, y]) \cap P|$
- ❖ 若共有 n 个输入点, 则需预先计算并记录 $\Omega(n^2)$ 项, 至少耗费 $\Omega(n^2)$ 时间和空间



Domination

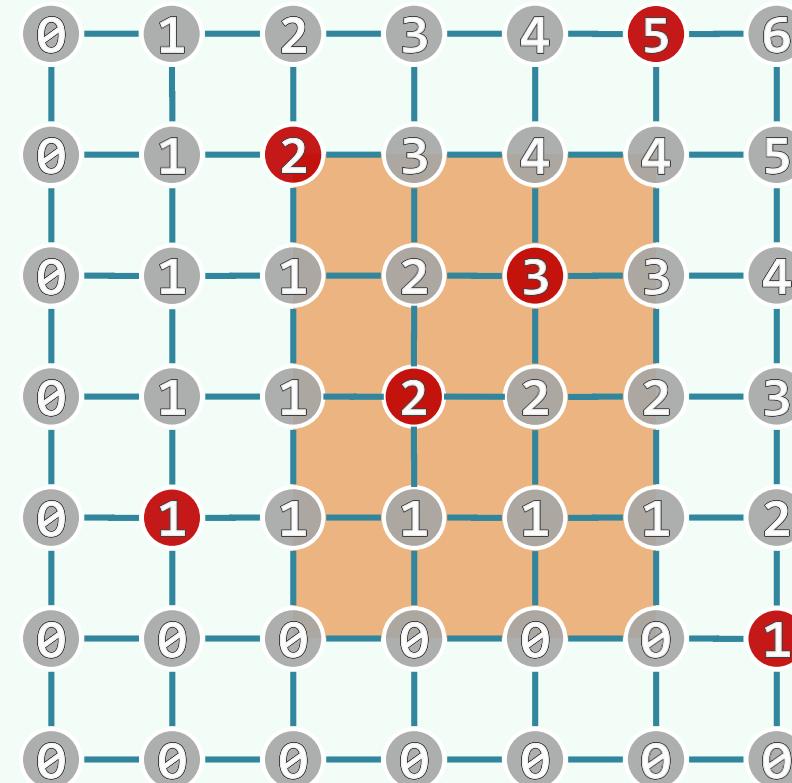
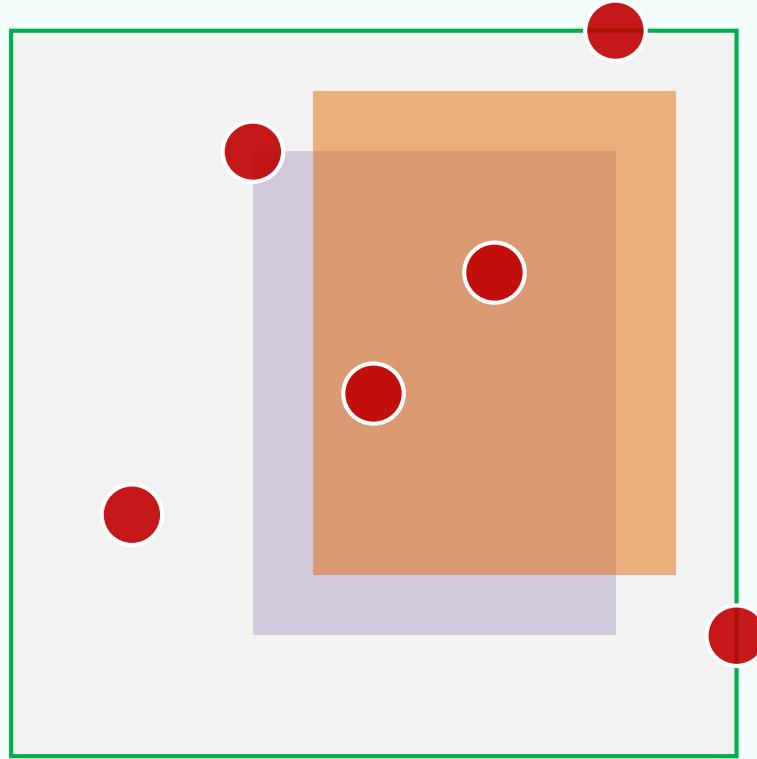
- ❖ 如果 $u \leq x$ 且 $v \leq y$, 则称点 (u, v) 被点 (x, y) 覆盖
- ❖ 通过二分查找, 查询矩形的每个角点均可在 $\mathcal{O}(\log n)$ 时间内, 吸附 (Snap) 到预记录过的某个点



Inclusion-Exclusion Principle

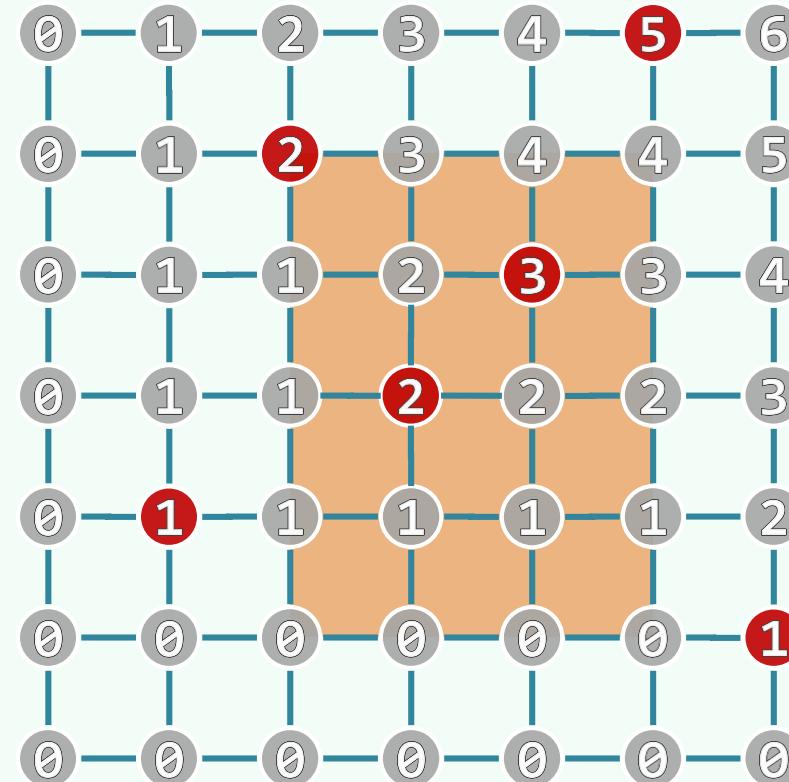
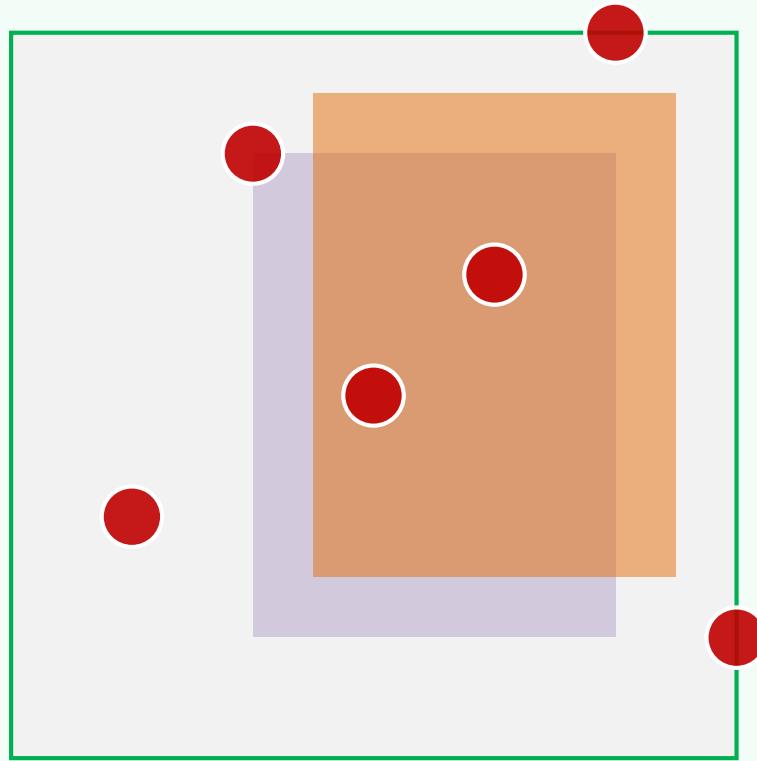
❖ 于是，对任何的矩形查询范围 $\mathcal{R} = (x_1, x_2] \times (y_1, y_2]$ ，都有

$$|\mathcal{R} \cap \mathcal{P}| = n(x_1, y_1) + n(x_2, y_2) - n(x_1, y_2) - n(x_2, y_1)$$



Performance

- ❖ 如此，每次查询只需 $\mathcal{O}(\log n)$ 时间；为此，需要耗费 $\Omega(n^2)$ 空间（并随维度呈指数增长）
- ❖ 为找到可行的方法，我们还是需要重新审视一维的情况...



搜索树应用

多层搜索树：一维

e> - D1

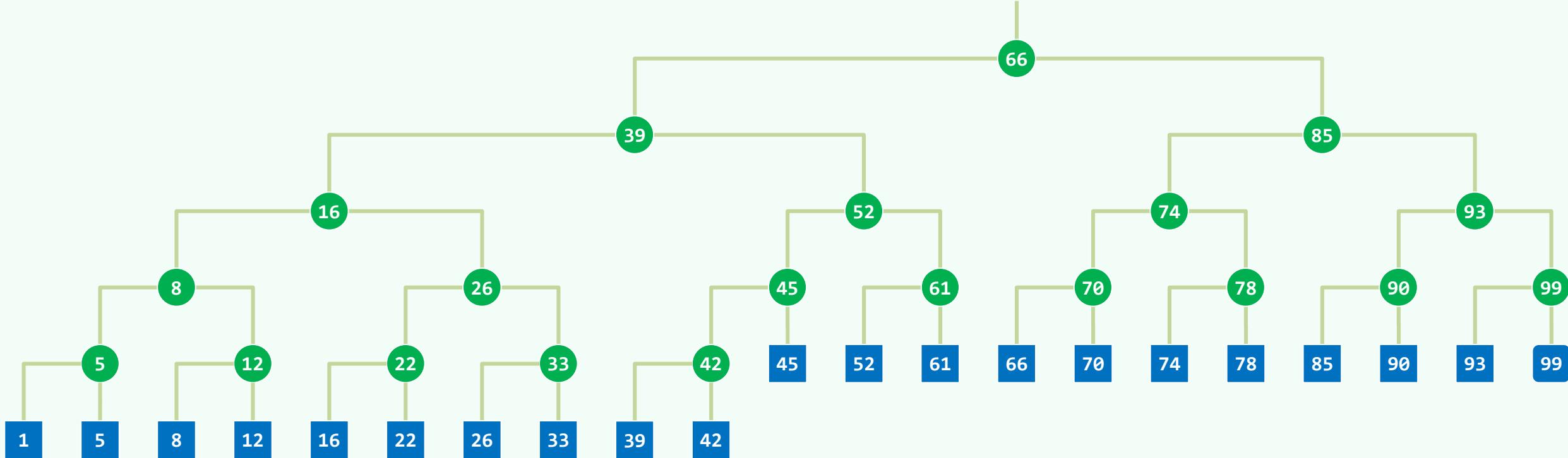
只在此山中，云深不知处

邓俊辉

deng@tsinghua.edu.cn

完全的二叉搜索树

- ❖ $\forall v, v.key = \min\{ u.key \mid u \in v.rTree \} = v.succ.key$



$\forall u \in v.lTree, u.key < v.key$

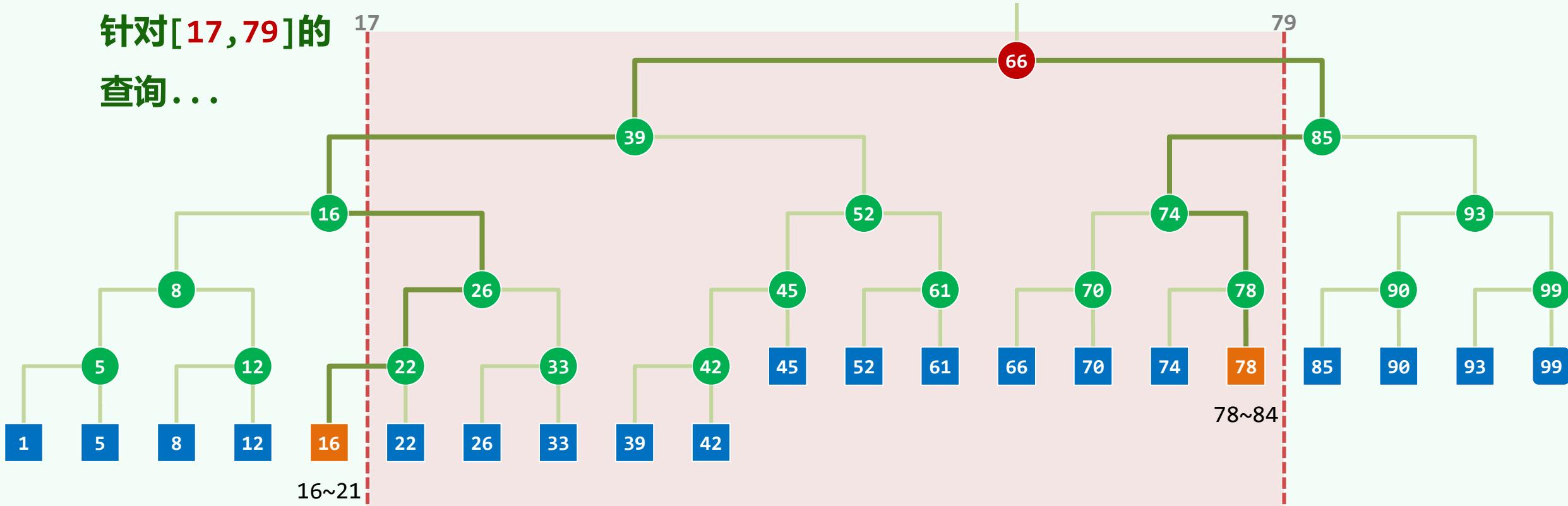
$\forall u \in v.rTree, u.key \geq v.key$

- ❖ `search(x)` : 返回不超过x的最大者

最低公共祖先

- ❖ 作为实例，考查

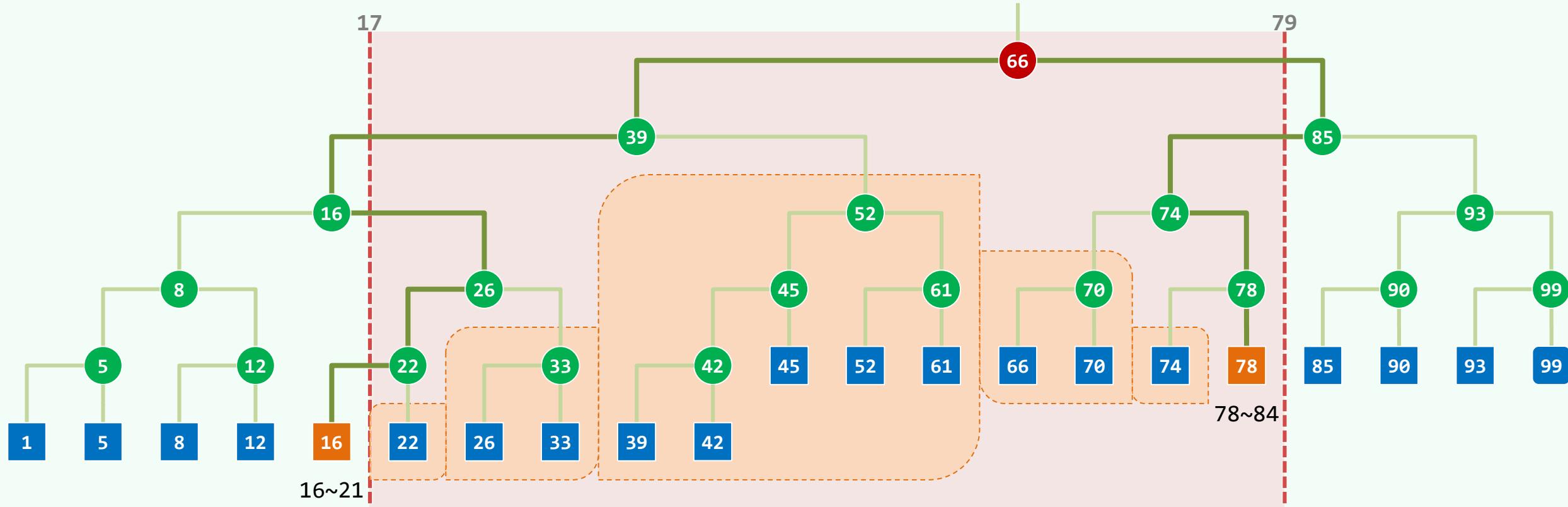
针对[17, 79]的
查询...



- ❖ 两次搜索: $\text{search}(17) = 16$ (可能排除) + $\text{search}(79) = 78$ (必然接受)
- ❖ 考查 $\text{LCA}(16, 78) = 66$ // Lowest Common Ancestor

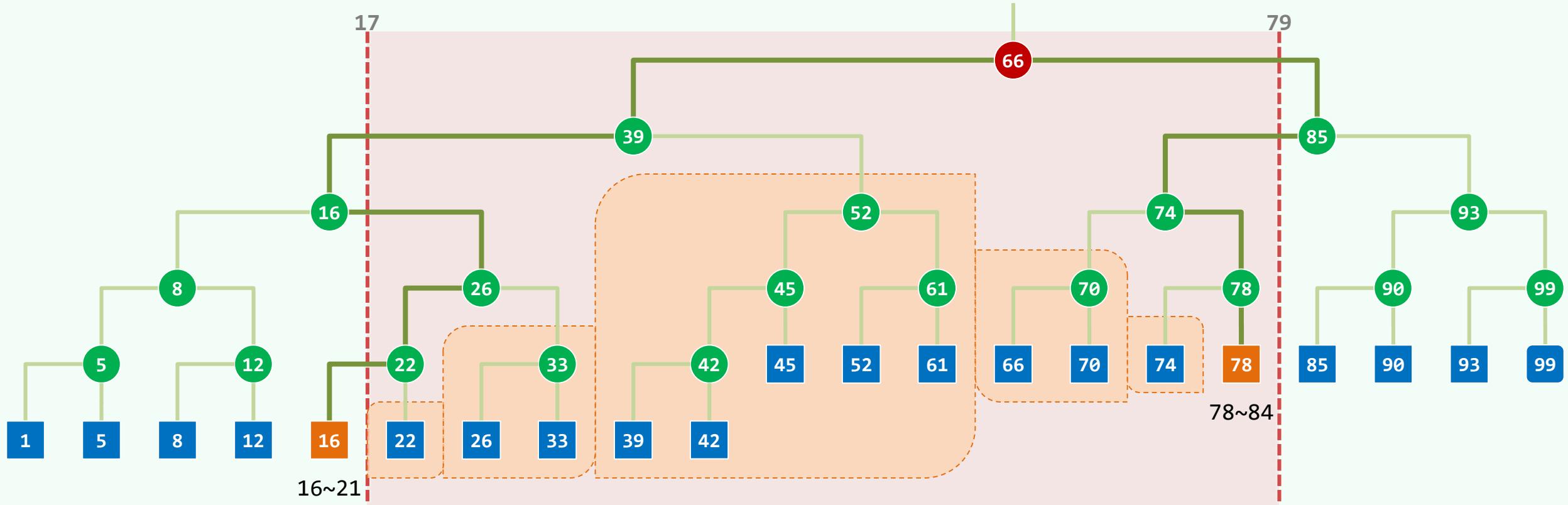
查询结果 = $\theta(\log n)$ 棵彼此无交的子树

- ❖ 从LCA出发，分别检视path(16)和path(78)



- ❖ 沿着path(16)/path(78): 所有的右转/左转均可忽略，而每棵右子树/左子树则直接报告

复杂度



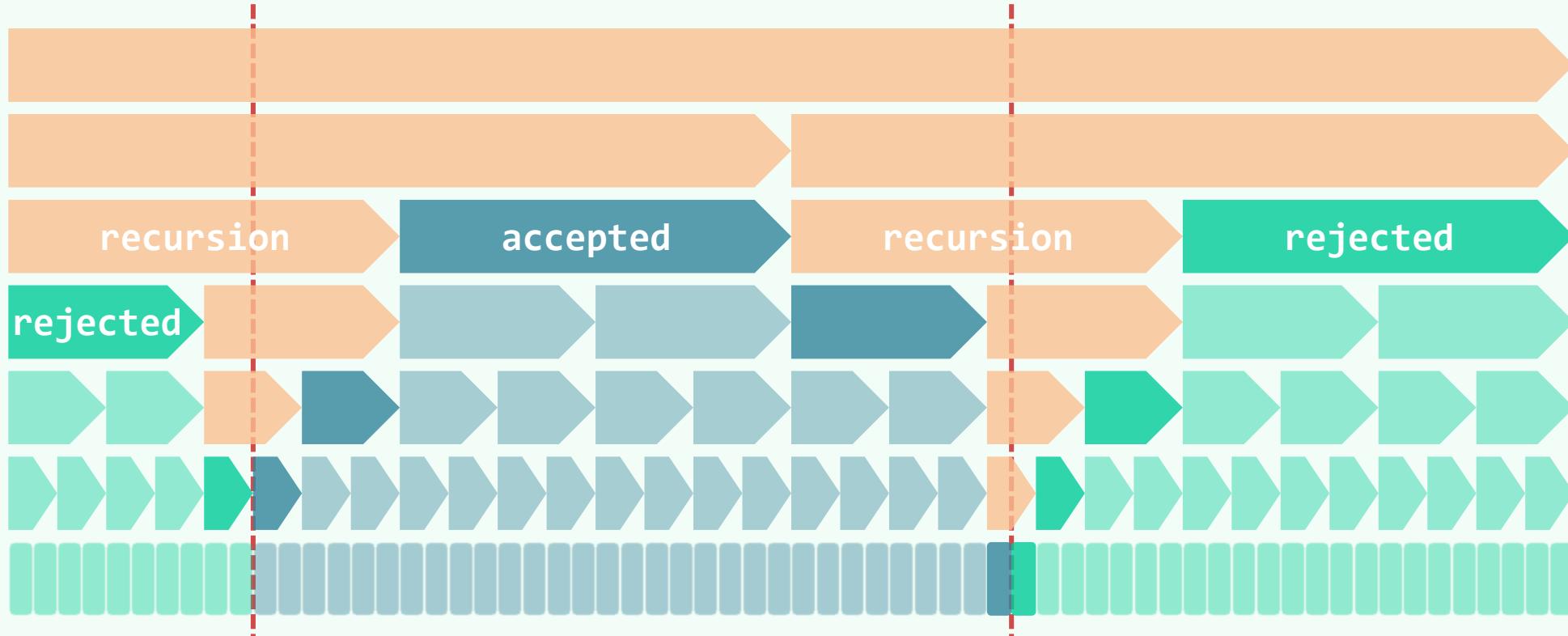
查询: $\mathcal{O}(\log n)$

预处理: $\mathcal{O}(n \log n)$

存储: $\mathcal{O}(n)$

热刀来切 $\Theta(\log n)$ 层的巧克力蛋糕

- ❖ Region(u)被Region(v)包含，当且仅当节点u是v的后代
- ❖ Region(u)与Region(v)无交，当且仅当u和v不是直系血缘关系



- ❖ 所谓的查询，无非是将节点分为三类：accepted + rejected + recursion

搜索树应用

多层搜索树：二维及多维



几株不知名的树，已脱下了黄叶
只有那两三片，多么可怜在枝上抖怯
它们感到秋来到，要与世间离别

邓俊辉

deng@tsinghua.edu.cn

二维范围查询 = x-查询 + y-查询

❖ 推而广之，每一次 m 维的正交范围查询

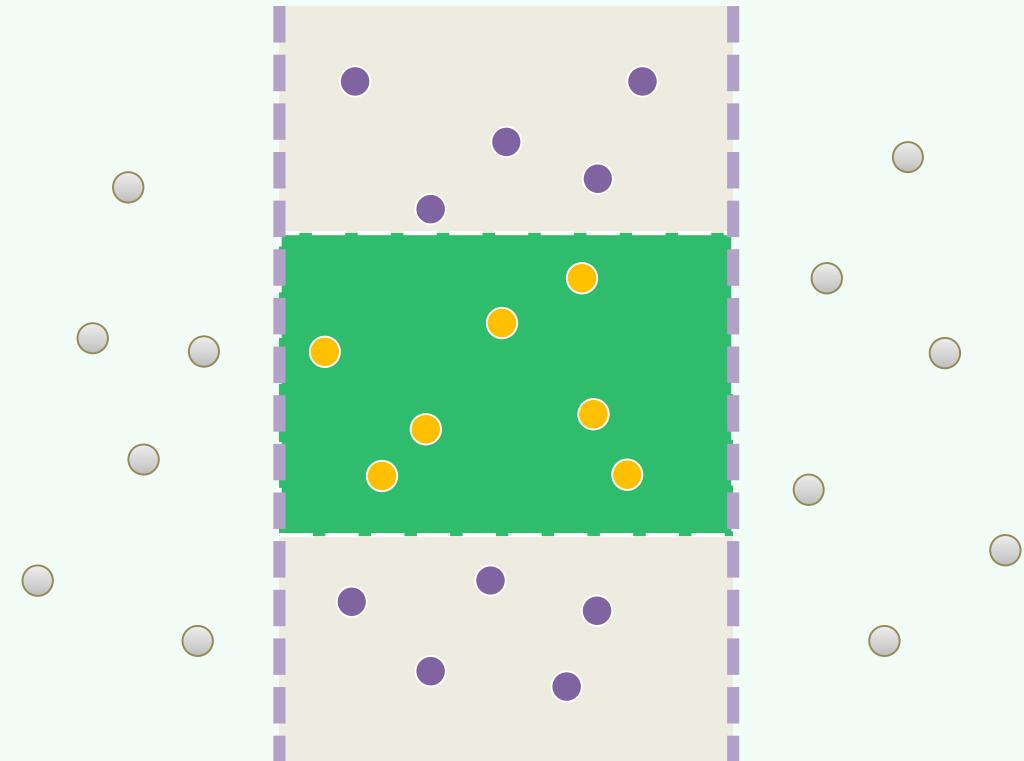
都可通过 m 次的1维正交查询来回答

❖ 比如，每一次2维范围查询

都可由2次的1维正交查询来回答

- 先查找出沿x轴落在 $[x_1, x_2]$ 内的点，进而
- 再筛选出沿y轴落在 $[y_1, y_2]$ 内的点

❖ 事情果然就如此简单？



最坏情况

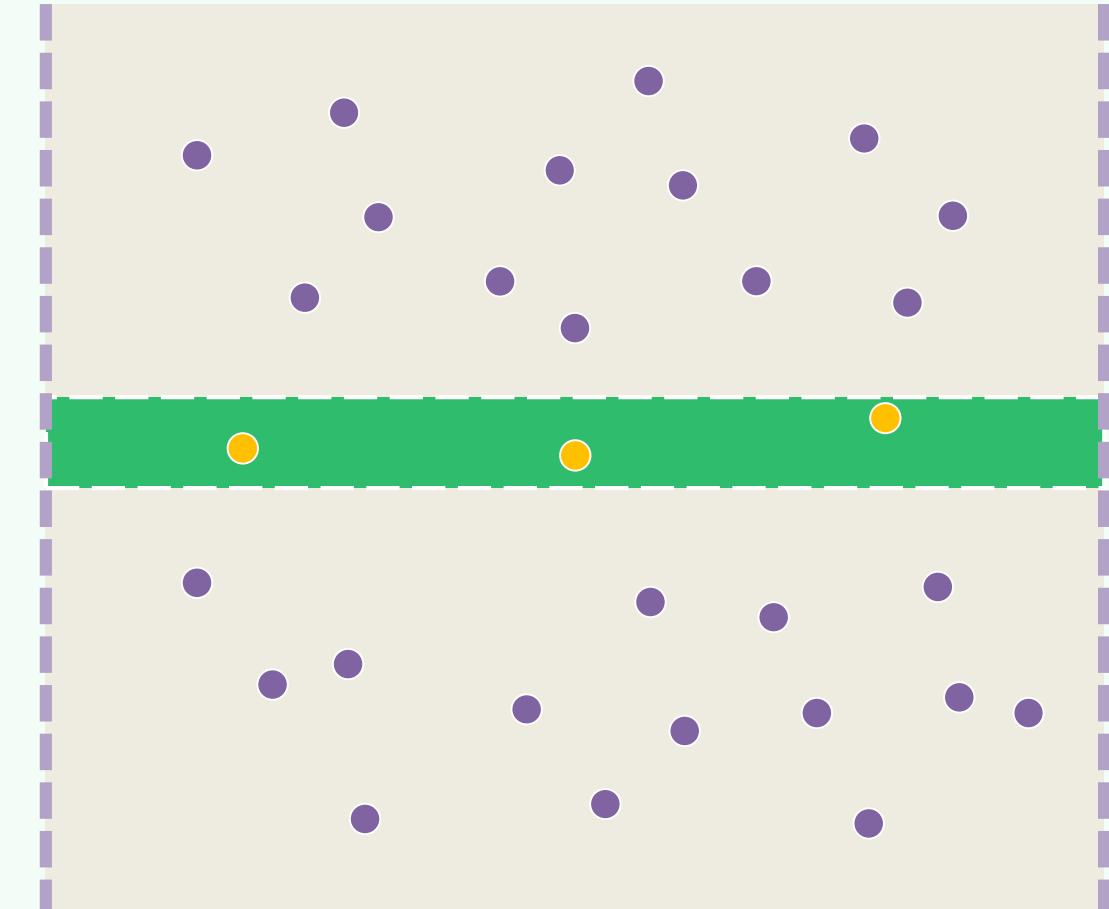
❖ 完全有可能...

- **x**查询虽命中了（几乎）所有点，而接下来
- **y**查询却排除了（几乎）所有点

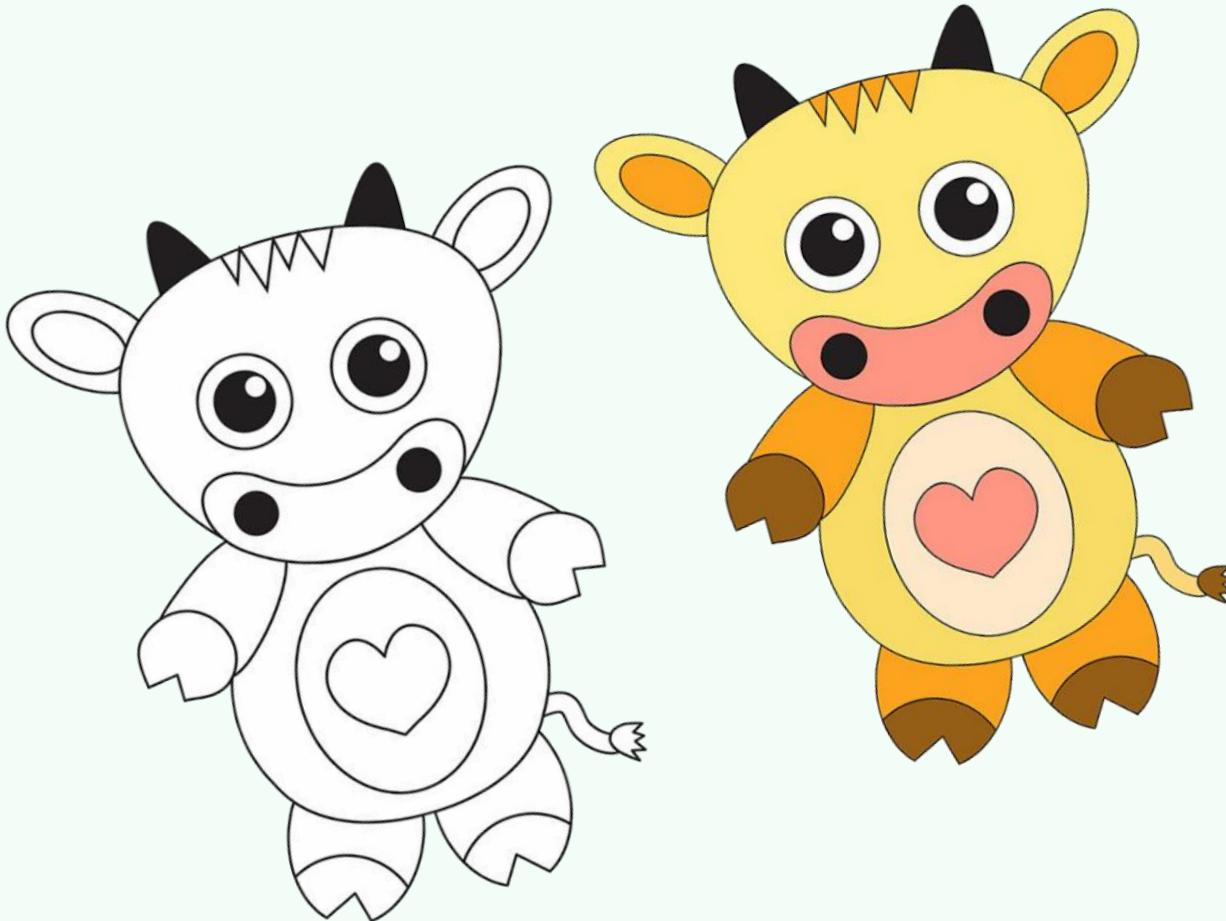
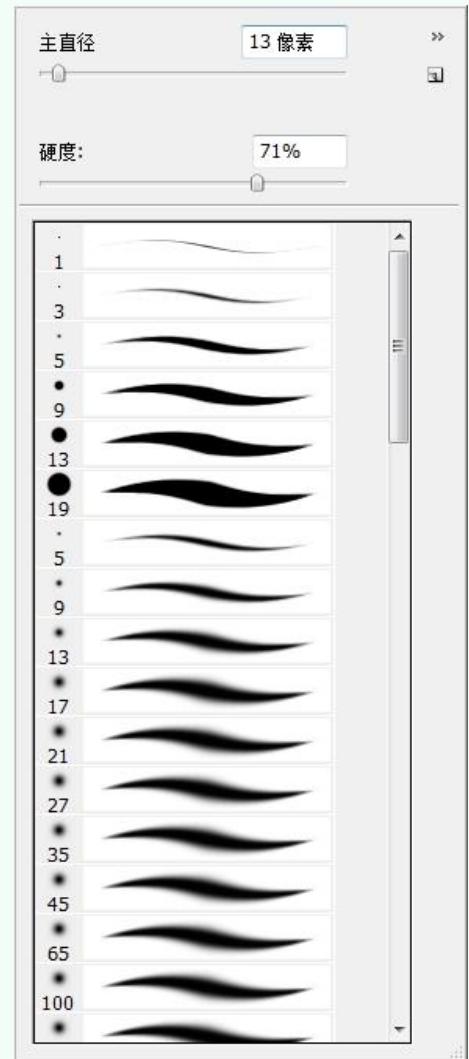
❖ 如此，花费 $\Omega(n)$ 时间却几乎一无所获

输出量（几乎）为0

❖ 如何做到**输出敏感** (output sensitivity) ?

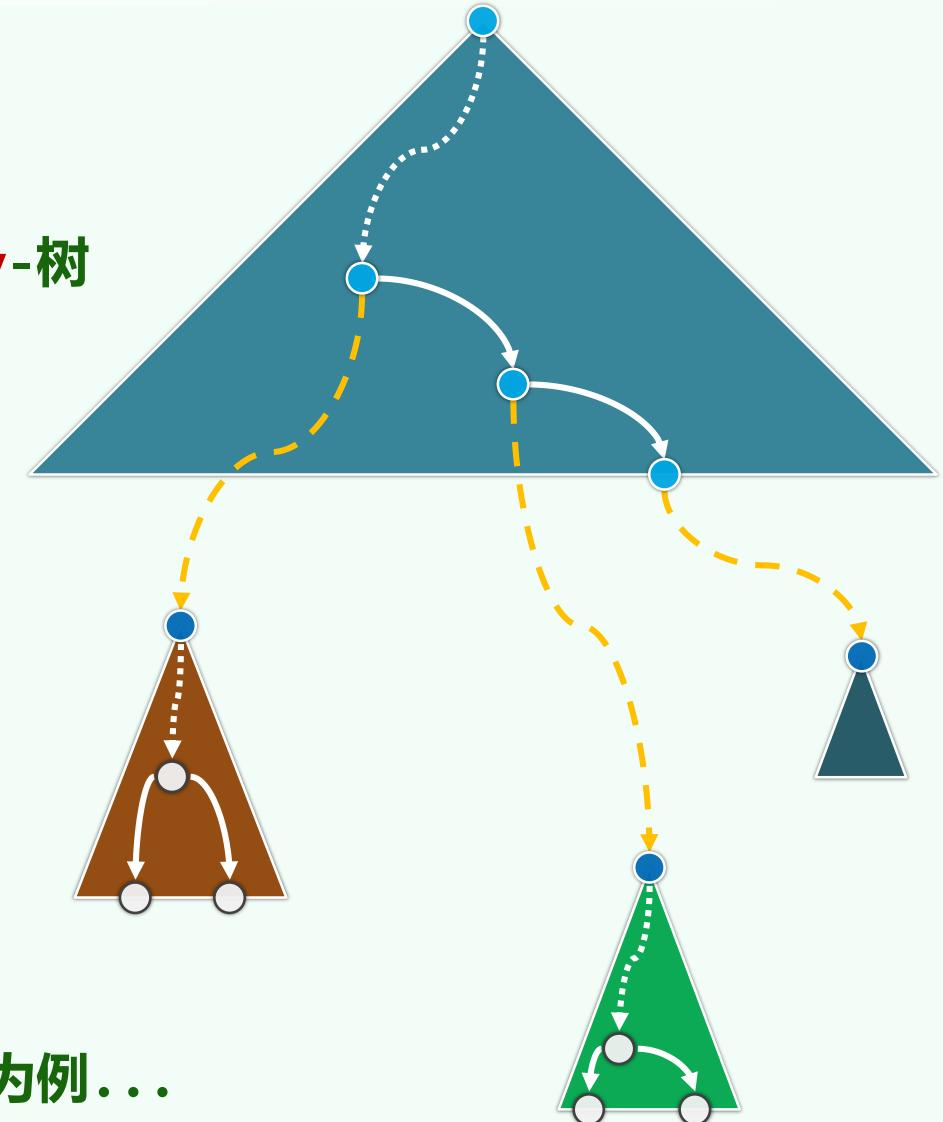


艺术与技术



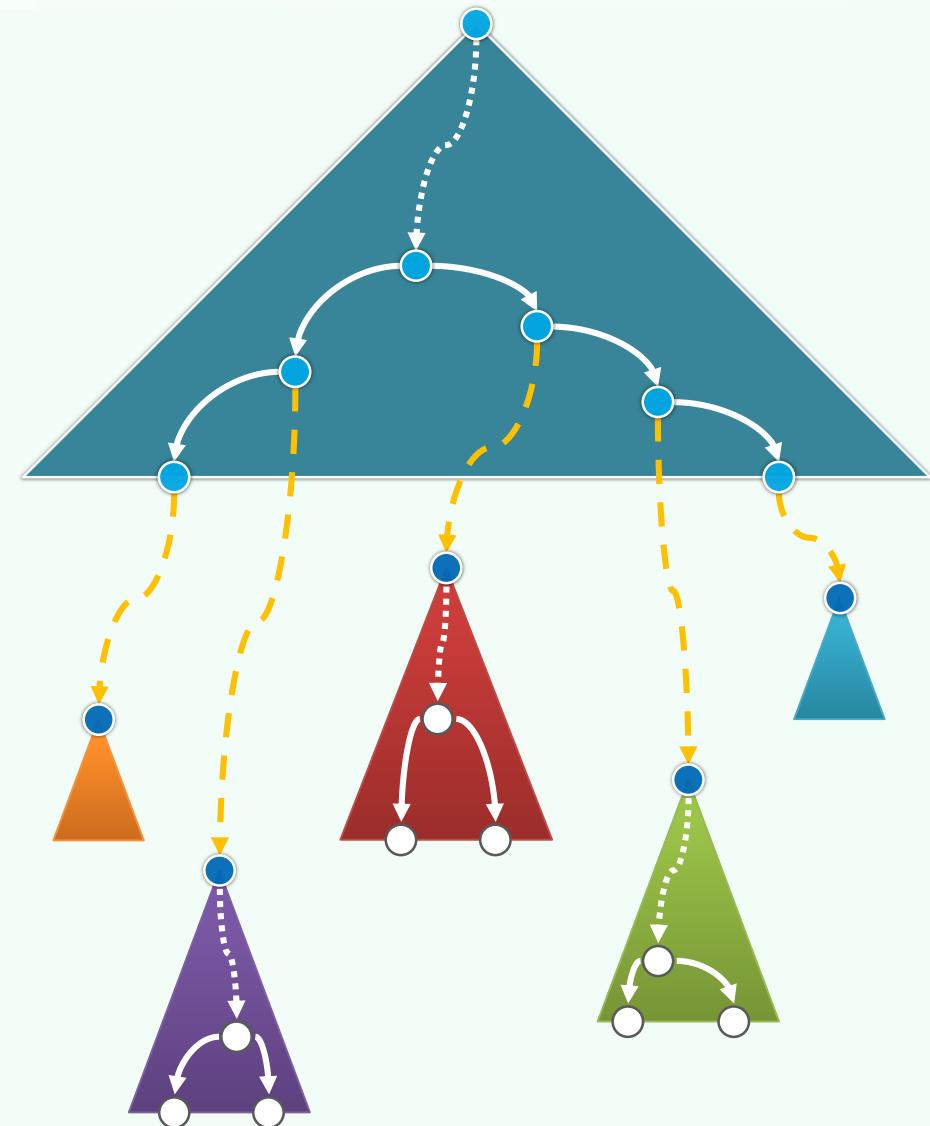
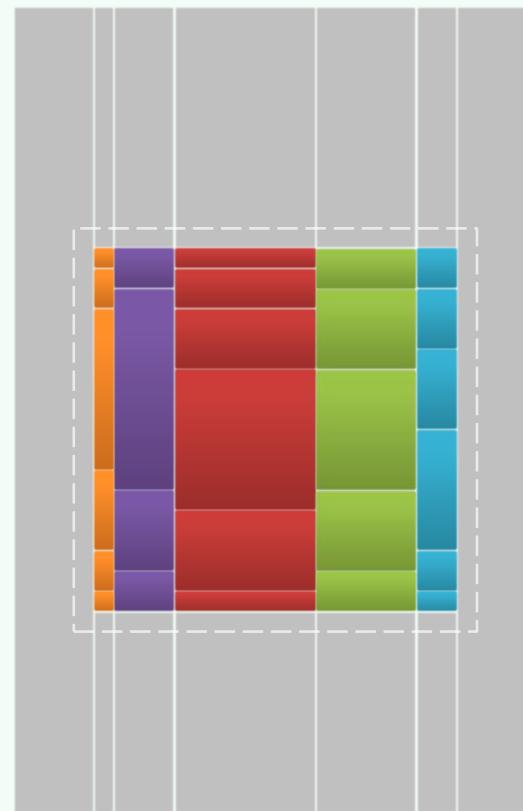
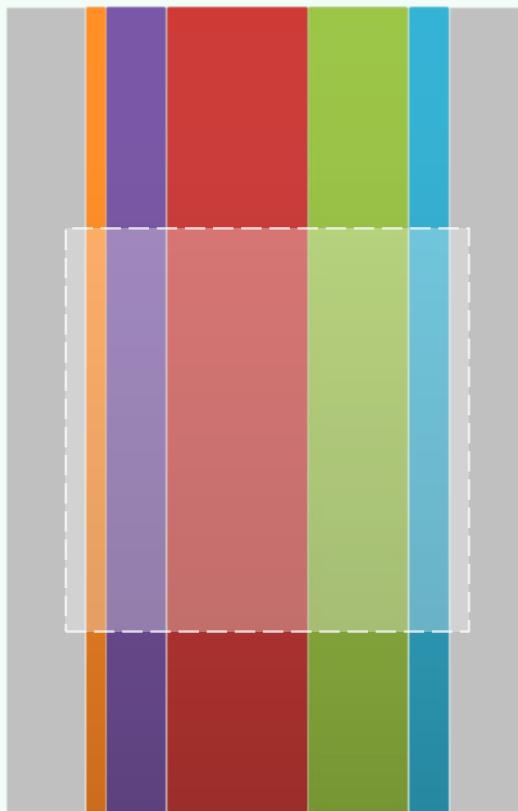
$$\text{MLST} = \text{x-Tree} * \text{y-Trees}$$

- ❖ 首先，针对 x -查询，构建一棵1维BBST（ x -树）
 - ❖ 然后，对于 x -树中的任何一棵子树 v ，另外构造一棵关联的 y -树
 - 二者对应于输入点集的同一子集，只不过
 - 顾名思义， y -树是将这些点按 y -方向排序
 - ❖ 每棵 x -子树，可以通过引用，直接找到与之对应的 y -树
 - ❖ 如果还有更多维度，也可照此逐层推广
- 整个结构也称作多层搜索树（Multi-Level Search Tree）
- ❖ 那么，如何借助MLST来高效地完成范围查询呢？以下以2维为例...



二维范围查询 = x-查询 * y-查询

❖ 查询时间 = $\mathcal{O}(r + \log^2 n)$ ~ $\mathcal{O}(r + \log n)$



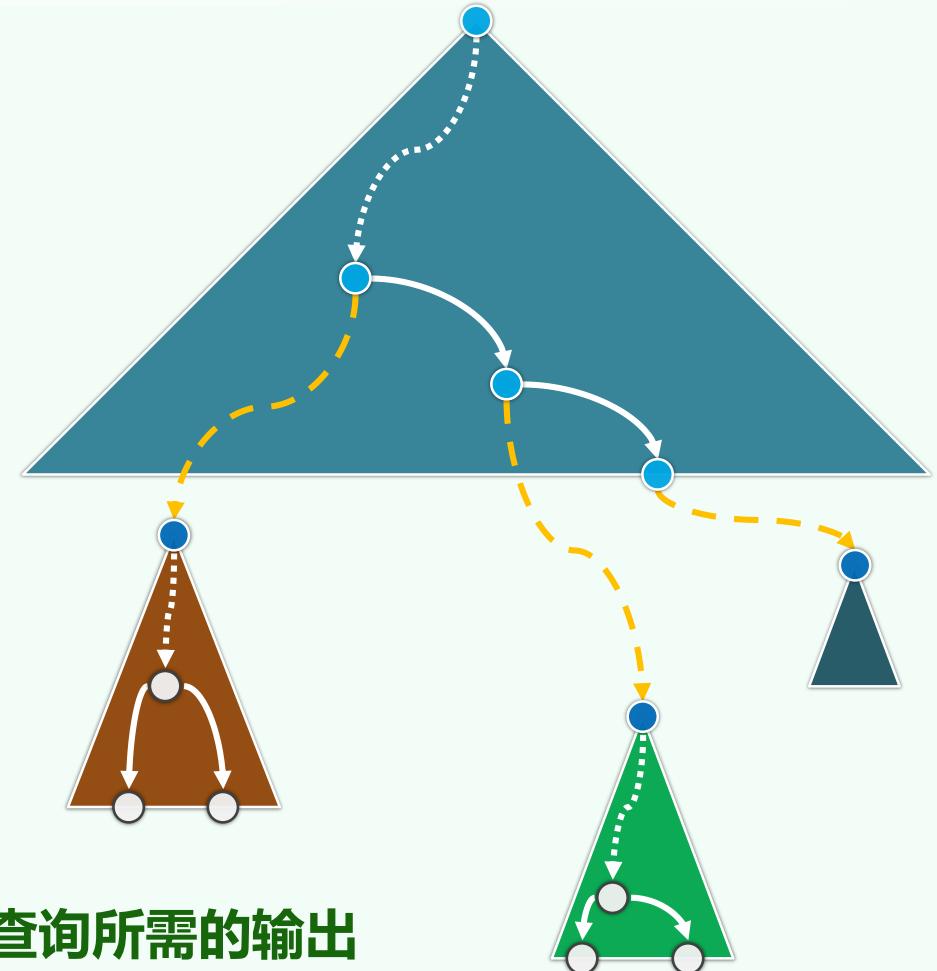
查询算法

❖ 过程

- 首先，通过 $x\text{-query}(x_1, x_2)$ 从 x -树中找出 $\mathcal{O}(\log n)$ 棵 x -子树
- 再分别在与之对应的 $\mathcal{O}(\log n)$ 棵 y -树中通过 $y\text{-query}(y_1, y_2)$ ，分批地筛选出所有命中的点

❖ 性质

- $x\text{-query}$ 和 $y\text{-query}$ 都是基本的一维范围查询
- $y\text{-query}$ 各自输出的子集相互无交，且其并集正是原查询所需的输出
- 除却输出本身所需的 $\mathcal{O}(r)$ 时间， $x\text{-query}$ 及所有的 $y\text{-query}$ 总共只需 $\mathcal{O}(\log n)$ 时间



复杂度：查询时间

❖ 声明：

在将平面上的任何一组点整理为一棵2层的MLST之后

每次2维范围查询都只需要 $\mathcal{O}(r + \log^2 n)$ 时间

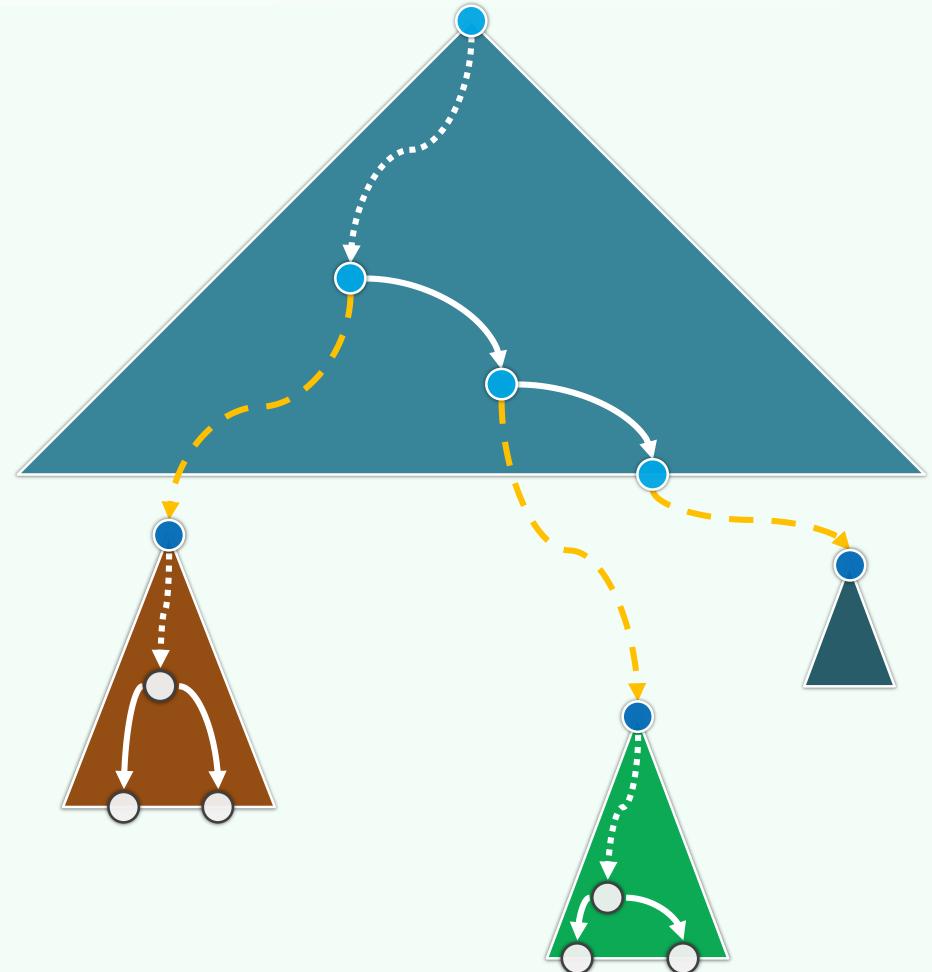
❖ 证明：

- **x-query**可在 $\mathcal{O}(\log n)$ 时间内

锁定 $\mathcal{O}(\log n)$ 棵**x-子树**

- 接下来的 $\mathcal{O}(\log n)$ 次**y-query**

各自也仅需 $\mathcal{O}(\log n)$ 时间



复杂度：预处理 + 存储空间

❖ 对于平面上任意的 n 个点

A> 都可以在 $\mathcal{O}(n \log n)$ 时间内建立一棵2层的MLST

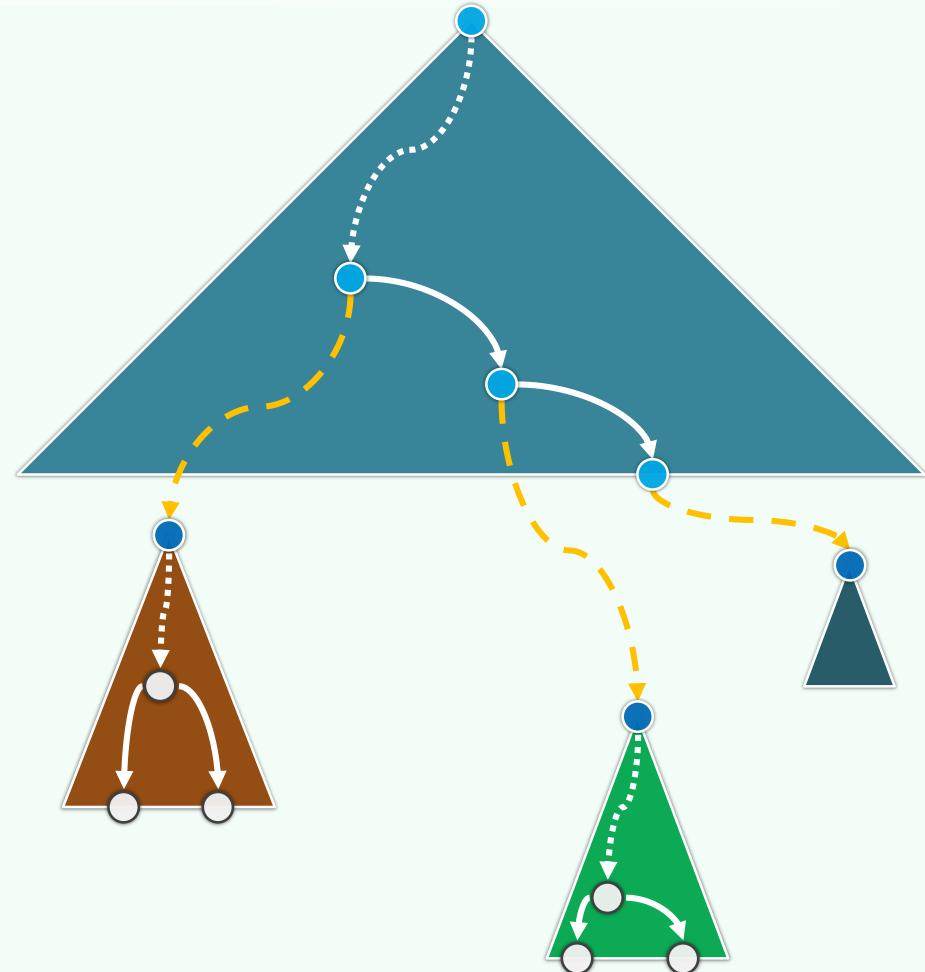
B> 该结构只需 $\mathcal{O}(n \log n)$ 空间

❖ 为做到A，可以“自底而上、逐层合并”的方式来构造

❖ 为理解B，只需注意到

输入的每一个点，都记录在 $\mathcal{O}(\log n)$ 棵y-树中

❖ 也可以按照深度，对y-树分类统计...



更高维度

❖ 由欧氏空间 \mathcal{E}^d ($d \geq 2$) 中的任意 n 个点，都可以

A> 在 $\mathcal{O}(n \cdot \log^{d-1} n)$ 时间内

构造出一棵 d 层的 MLST

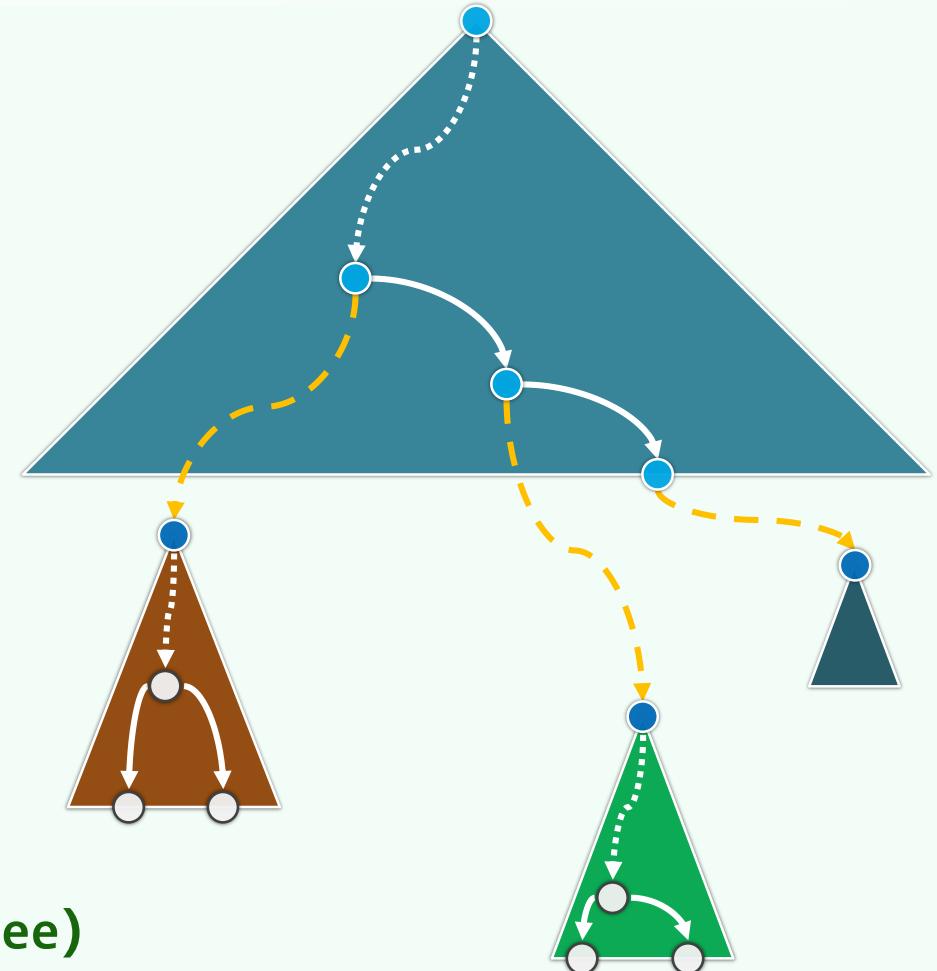
B> 该结构只需 $\mathcal{O}(n \cdot \log^{d-1} n)$ 空间

C> 借助该结构，每次 d 维范围查询

都可以在 $\mathcal{O}(r + \log^d n)$ 时间内完成

❖ 借助分散层叠的技巧，将 MLST 升级为范围树 (Range Tree)

即可将 C> 改进至 $\mathcal{O}(r + \log^{d-1} n) \dots$



搜索树应用

范围树

邓俊辉

deng@tsinghua.edu.cn



顺藤摸瓜

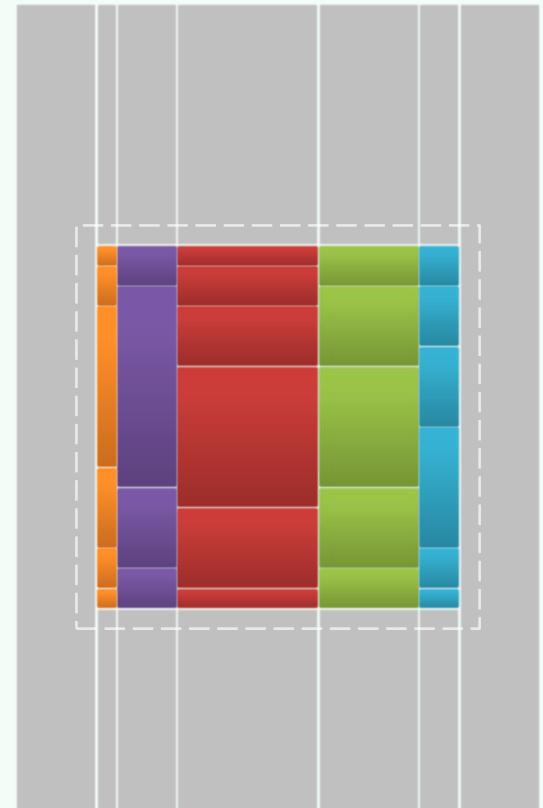
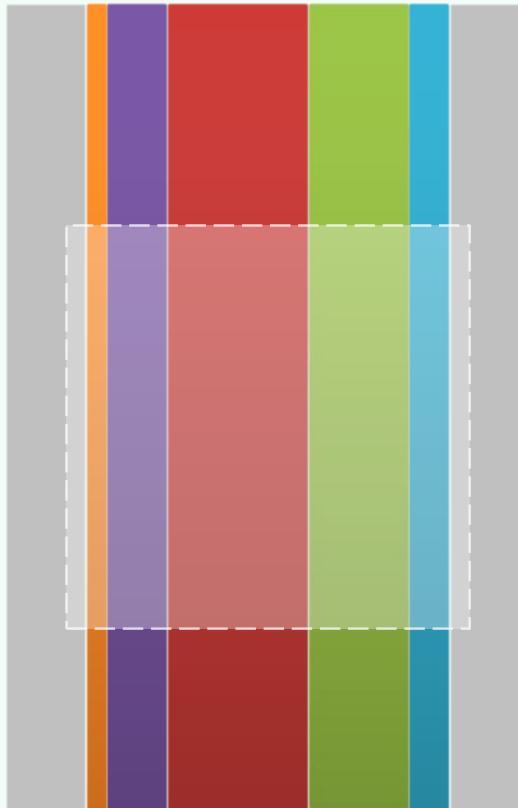
y-query的相关性 (Coherence)

❖ 以2维范围查询为例，不难观察到：

- 虽然会做 $\mathcal{O}(\log n)$ 次的y-query(y_1, y_2)
- 但只是针对的y-树各自不同
- 而对应的范围则完全相同

❖ 既然所有y-query之间有如此紧密的相关性

目前这样令它们各自独立的完成，很不科学...



BBST<BBST<T>> --> BBST<List<T>>

❖ 作为最后一个维度，每次y-query

都是不需递归的1维范围查询

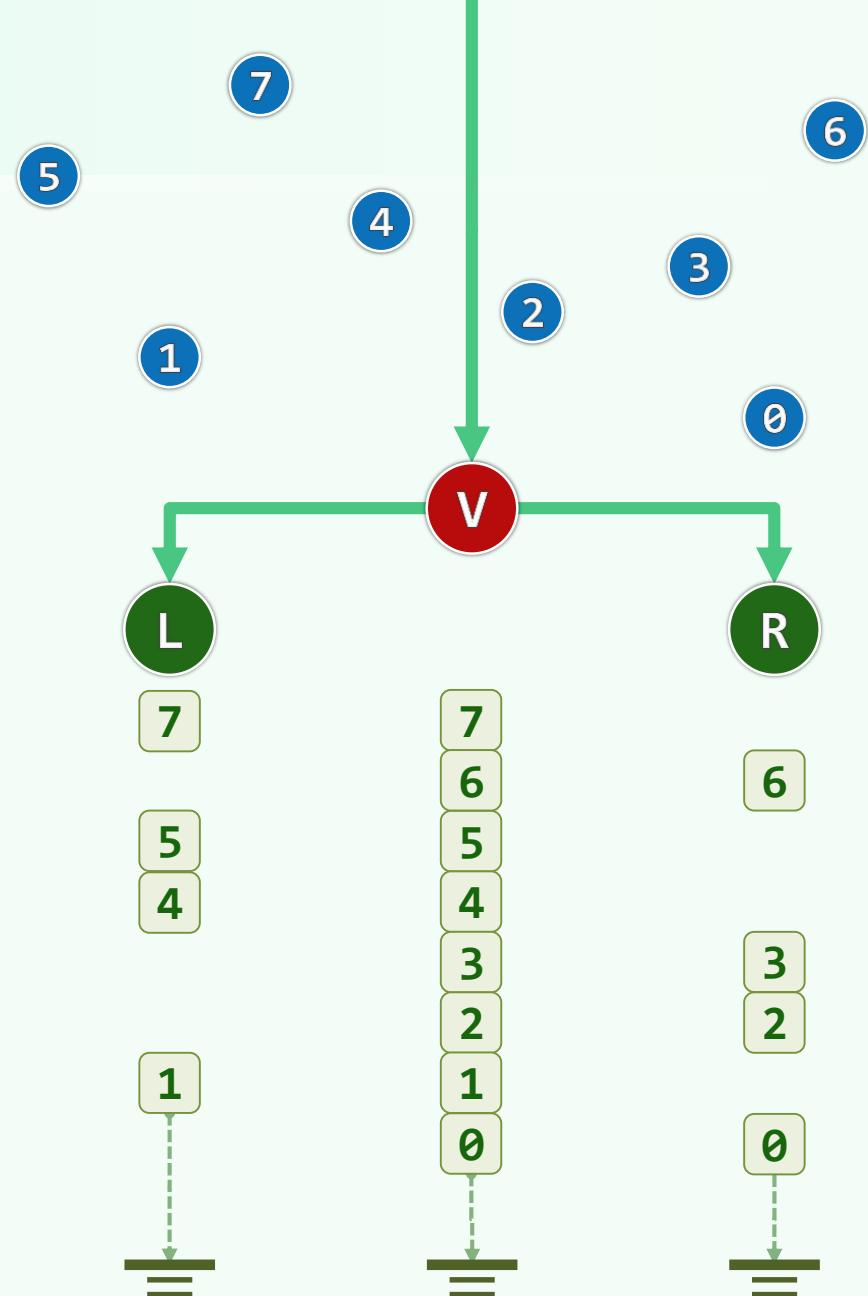
❖ 既如此，完全不必将其组织为一棵BBST

实际上，一个有序序列（列表或向量）即足矣

❖ 于是，可以将整个结构理解并改造为

- 一棵x-树，以及

- 与其中每个节点/子树相关联的一个有序序列



分散层叠: Shortcuts

❖ Fractional Cascading:

进一步地，可在任意节点 v 的关联序列

及其孩子的关联序列之间，建立快捷引用

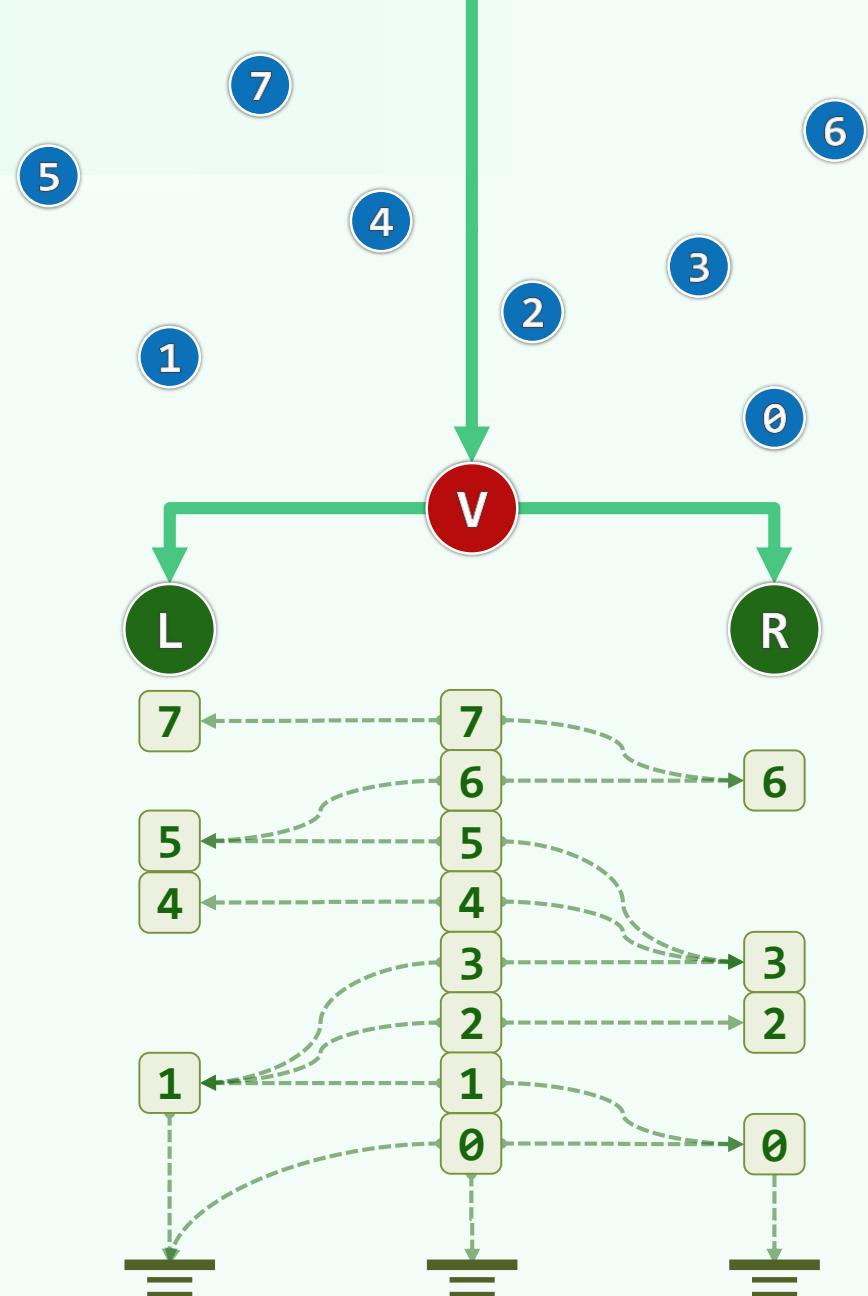
❖ 对于同一 y 坐标值，由 $v.\text{search}(y)$

可以直接得到 $L.\text{search}(y)$ 和 $R.\text{search}(y)$

❖ 也就是说，在执行一批 y -query()的过程中

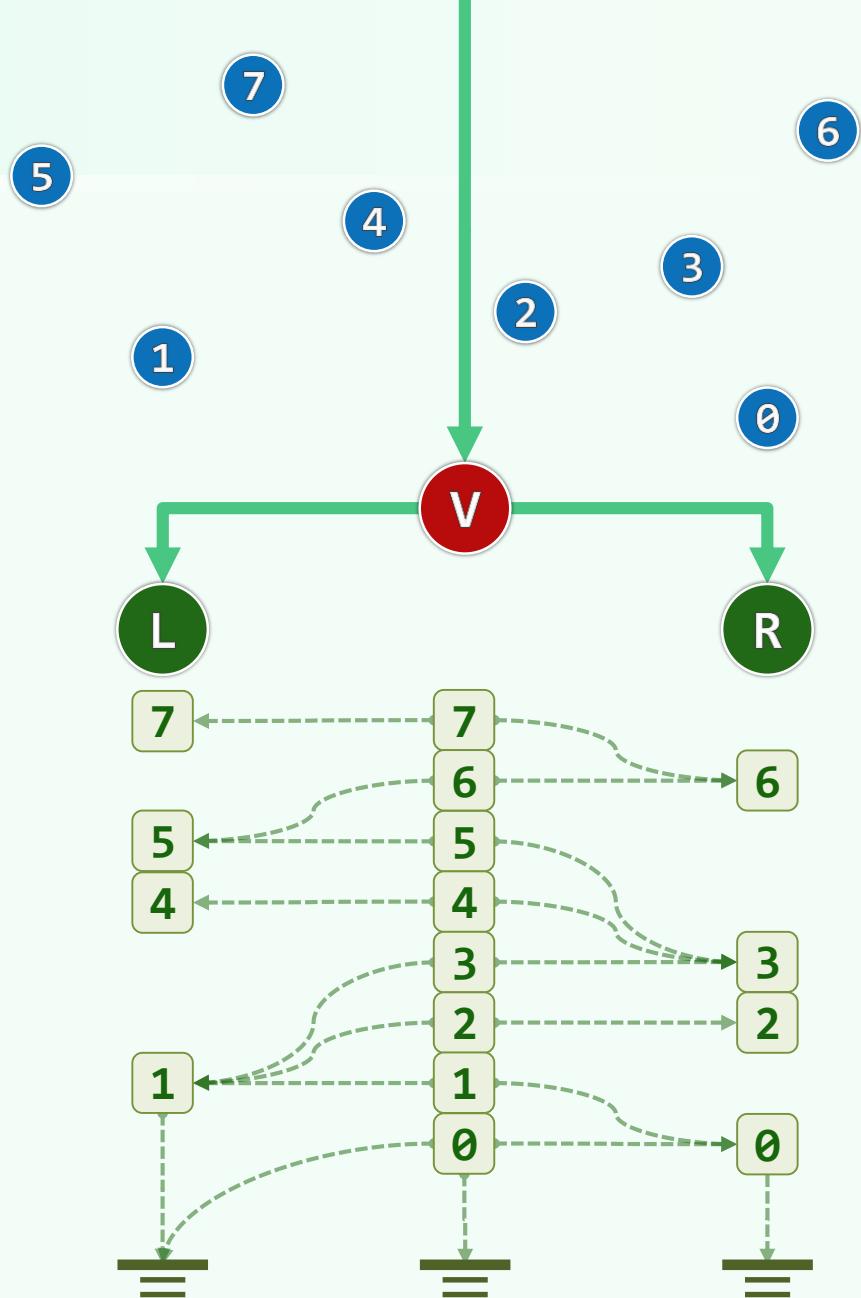
- 一旦耗费 $\mathcal{O}(\log n)$ 时间完成了 $\text{LCA}.\text{query}()$

- 所有后代们的 y -query都各自仅需 $\mathcal{O}(1)$ 时间



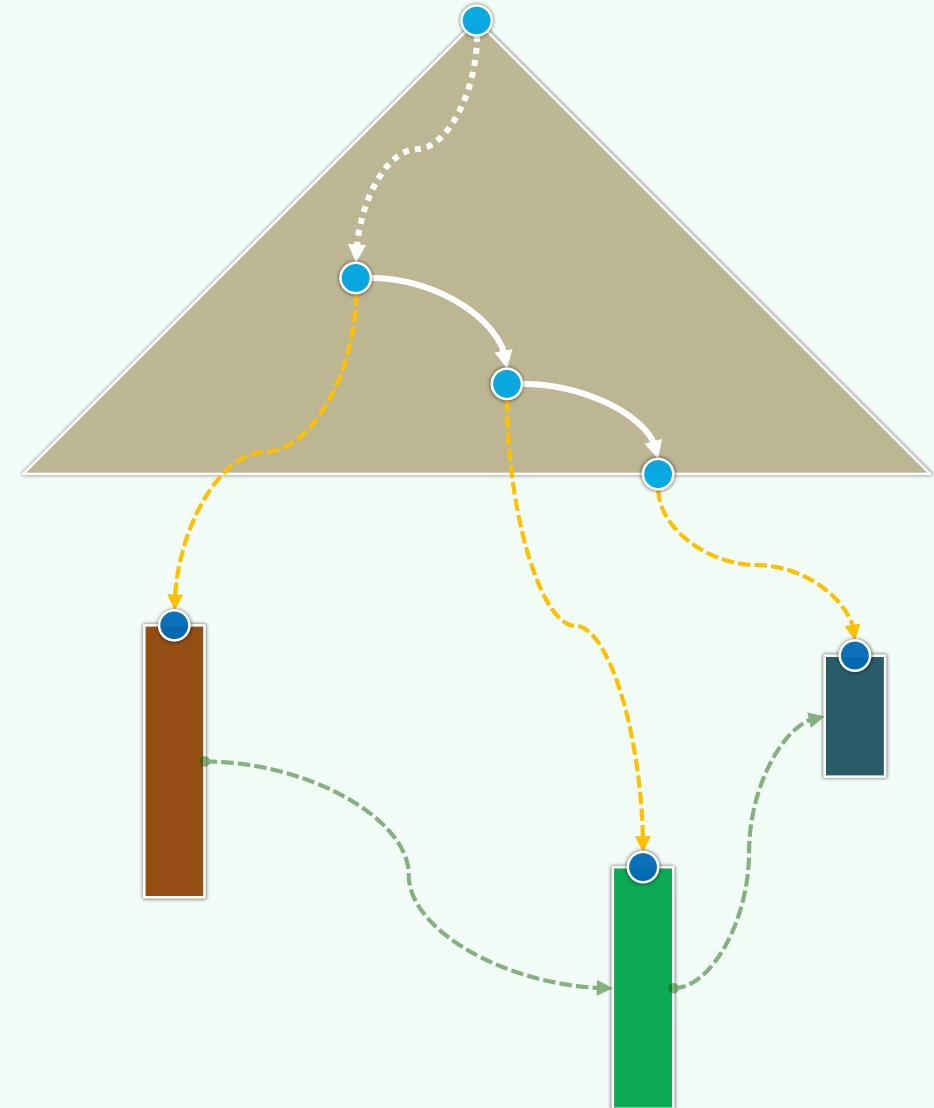
分散层叠：二路归并

- ❖ 一方面，**v的关联列表，总是等于L和R的关联列表之并**
- ❖ 另一方面，正如此前所推荐地 **x-树应该自底而上地逐层合并而成**
- ❖ 既如此，**x-树中各节点的关联列表也可以通过二路归并自底而上地依次生成**
- ❖ 如此，并**不会增加预处理的整体时间复杂度**



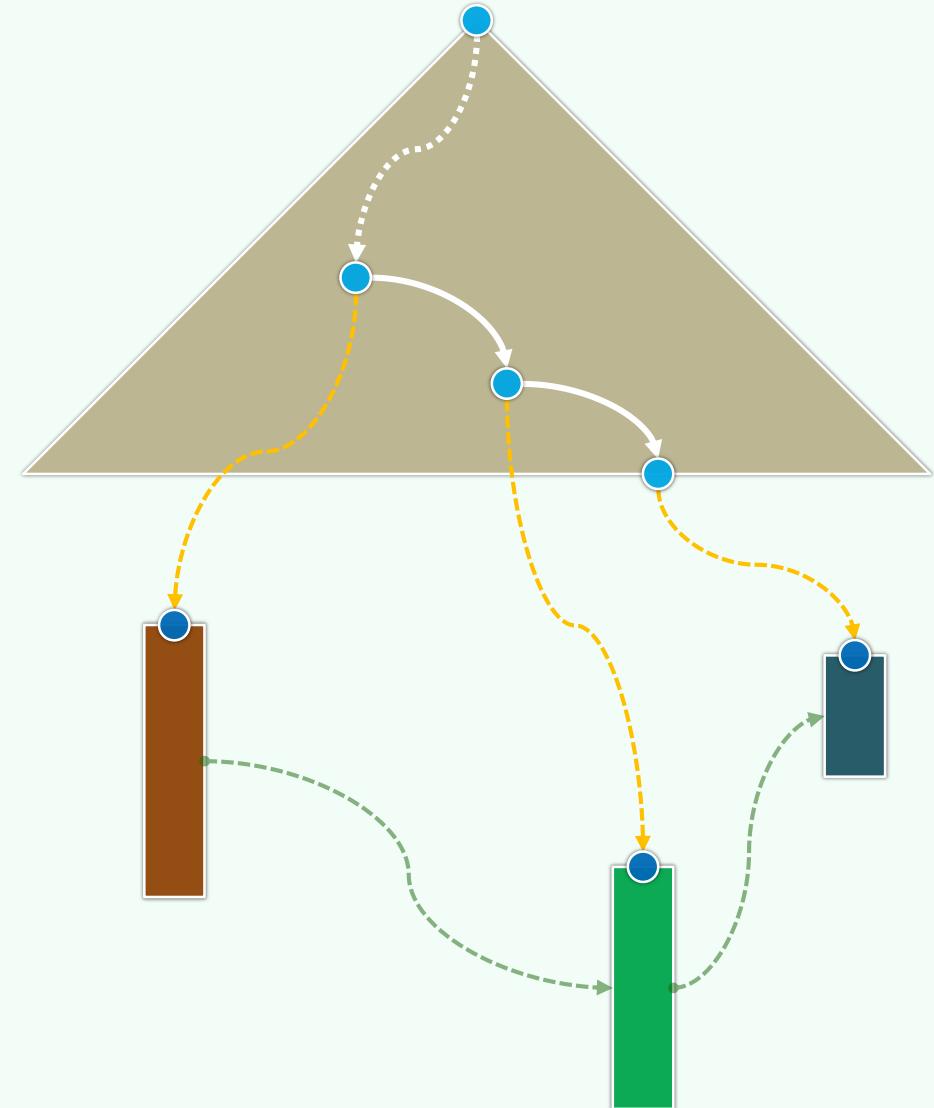
范围树

- ❖ 引入了分散层叠技术的MLST
称作**范围树** (Range Tree)
- ❖ 由平面上的任意n个点，都可以
 - A> 在 $\mathcal{O}(n \cdot \log n)$ 时间内
构造出一棵2层的范围树
 - B> 该结构只需 $\mathcal{O}(n \cdot \log n)$ 空间
 - C> 借助该结构，每次二维范围查询
都可以在 $\mathcal{O}(r + \log n)$ 时间内完成



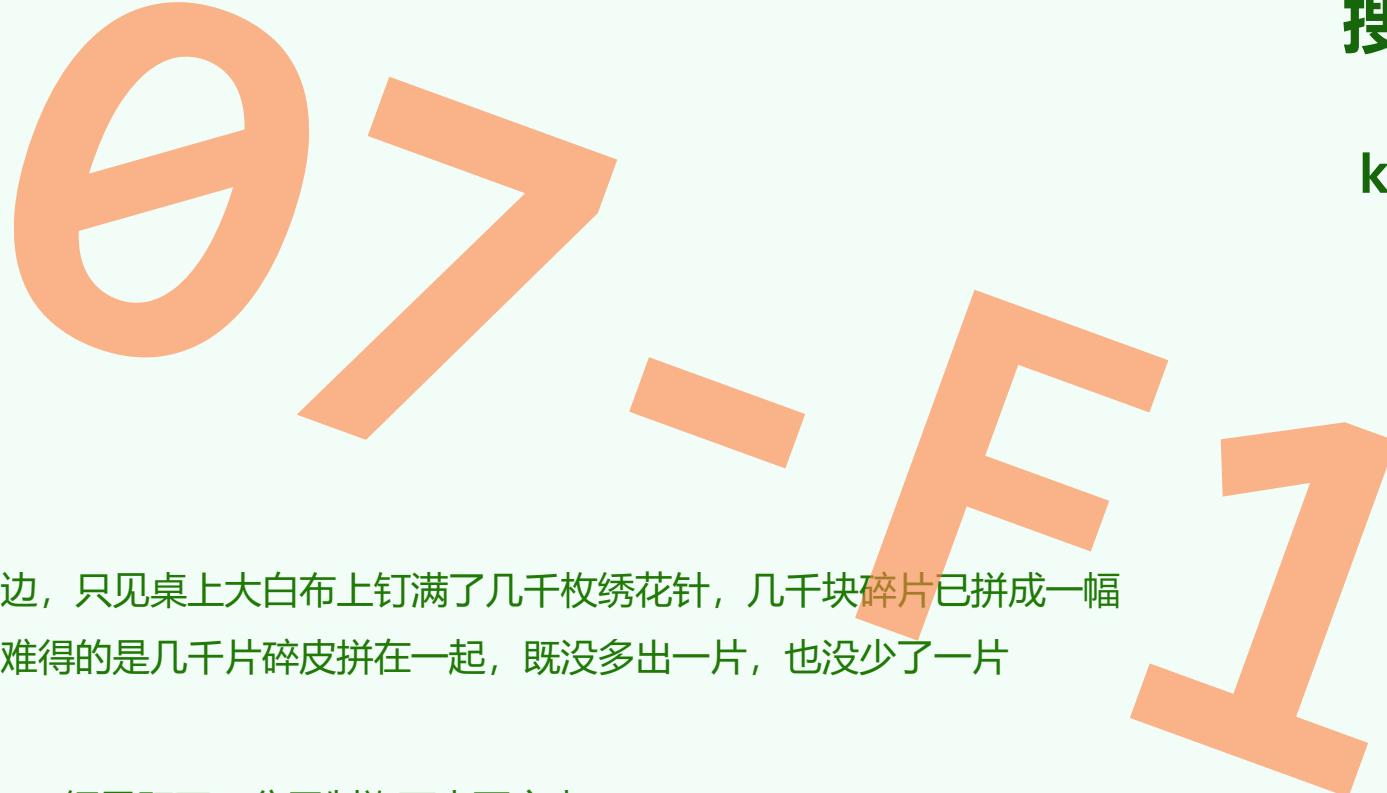
更高维度

- ❖ 遗憾的是，对MLST结构而言
分散层叠的技巧只能运用于最后一个维度
- ❖ 由欧氏空间 \mathcal{E}^d ($d \geq 2$) 中的任意 n 个点，都可以
 - A> 在 $\mathcal{O}(n \cdot \log^{d-1} n)$ 时间内
构造出一棵 d 层的MLST
 - B> 该结构只需 $\mathcal{O}(n \cdot \log^{d-1} n)$ 空间
 - C> 借助该结构，每次 d 维范围查询
都可以在 $\mathcal{O}(r + \log^{d-1} n)$ 时间内完成



搜索树应用

kd-树：二维



韦小宝跟著她走到桌边，只见桌上大白布上钉满了几千枚绣花针，几千块碎片已拼成一幅完整无缺的大地图，难得的是几千片碎皮拼在一起，既没多出一片，也没少了一片

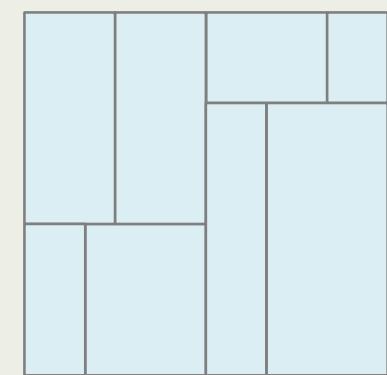
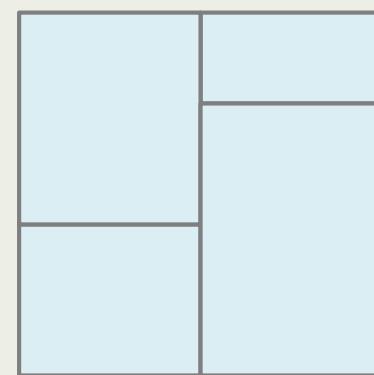
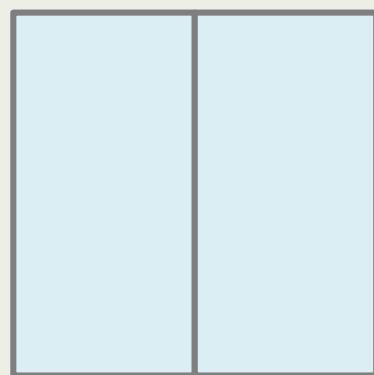
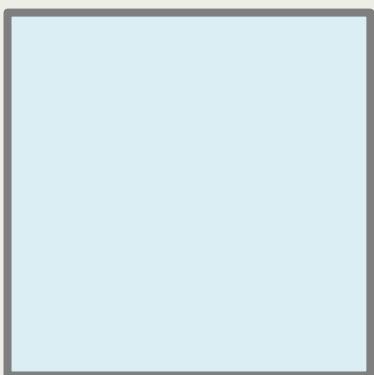
夫仁政，必自經界始...經界既正，分田制祿可坐而定也

邓俊辉

deng@tsinghua.edu.cn

思路

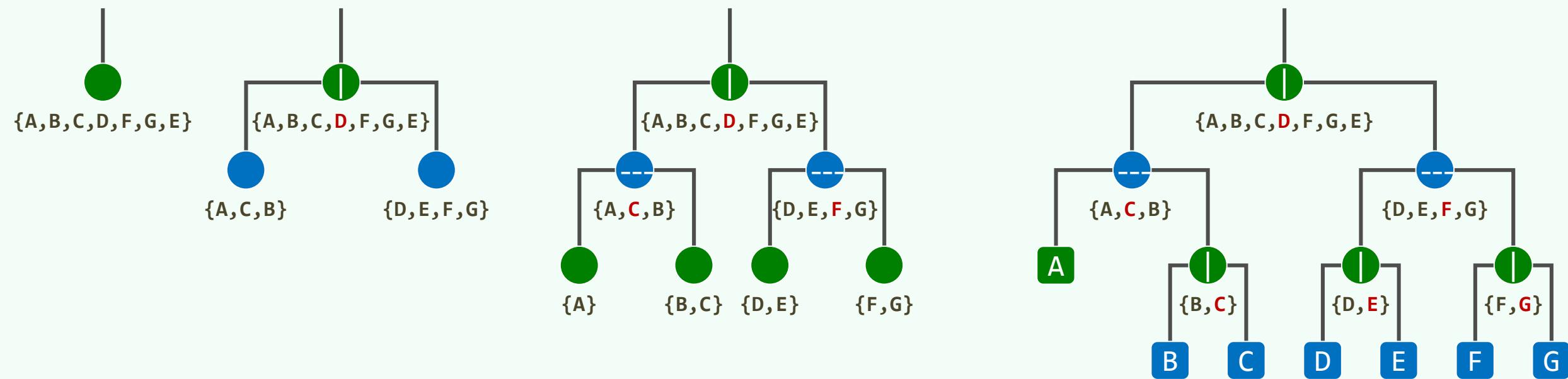
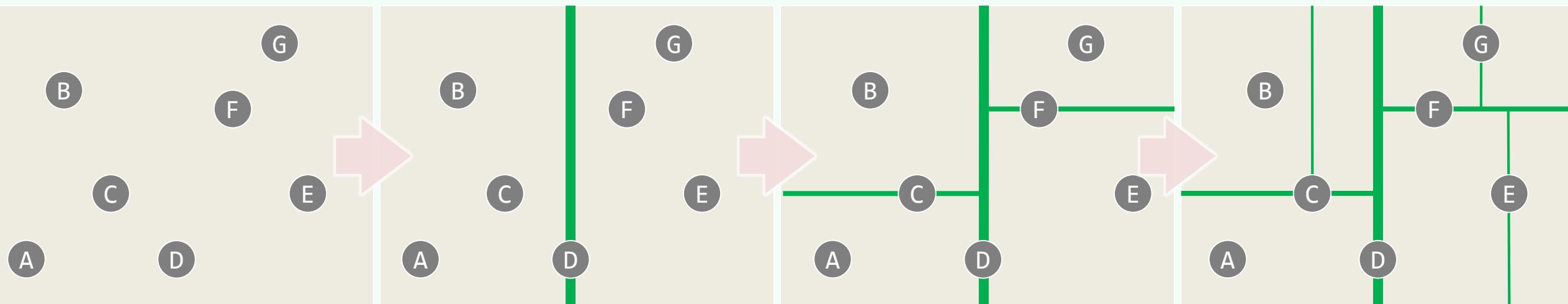
- 为利用BBST来支持二维的区域查询，可以递归地划分平面，并将分出来的矩形区域组织为一棵二叉树
- 首先，根节点对应于整个平面
然后，交替地按x、y坐标划分，直至…
- 为避免歧义，可约定每个矩形区域都是左闭右开、下闭上开



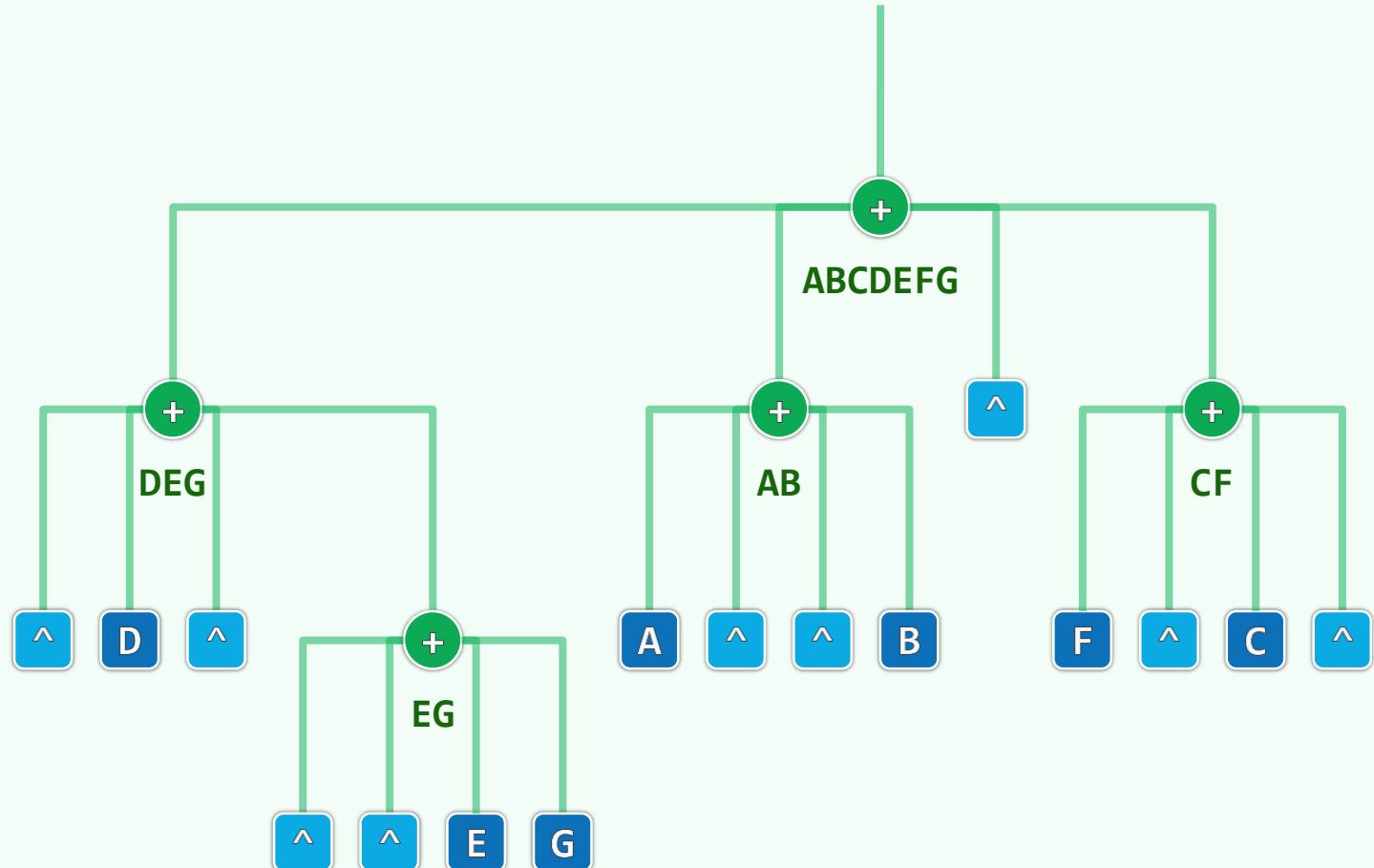
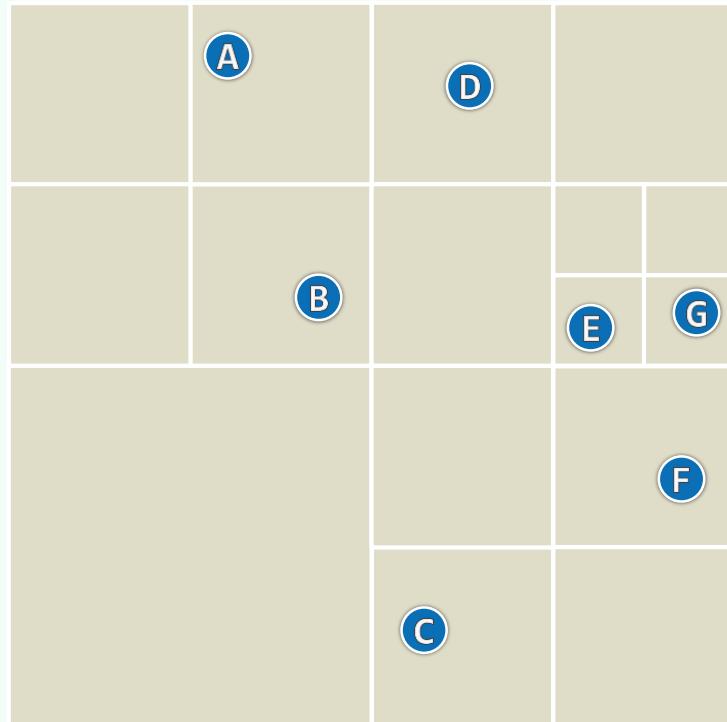
构造：算法buildKdTree(P,d)

```
// Construct a 2d-tree for point set P at depth d  
  
if ( P == {p} ) return createLeaf( p ) //base  
  
Root = createKdNode()  
  
Root->SplitDirection = even(d) ? VERTICAL : HORIZONTAL  
  
Root->SplitLine = findMedian( root->SplitDirection, P ) // $\mathcal{O}(n)$ !  
  
( P1, P2 ) = divide( P, Root->SplitDirection, Root->SplitLine ) //DAC  
  
Root->LC = buildKdTree( P1, d + 1 ) //recurse  
  
Root->RC = buildKdTree( P2, d + 1 ) //recurse  
  
return( Root )
```

构造：实例



特例：Quadtree



性质

❖ 树中的每个节点 v ，都对应于平面上的一个矩形区域 $region(v)$ 以及子集 $P(v) = P \cap region(v)$

❖ 树高 $height(T) = \mathcal{O}(\log n)$

❖ 任何同层的节点 v 和 u ，都有：

$$region(v) \cap region(u) = \emptyset$$

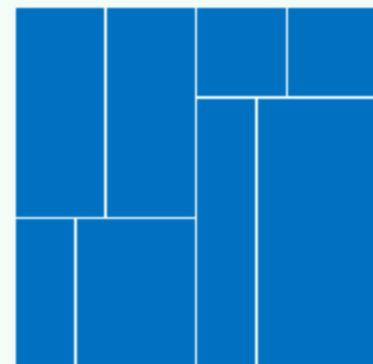
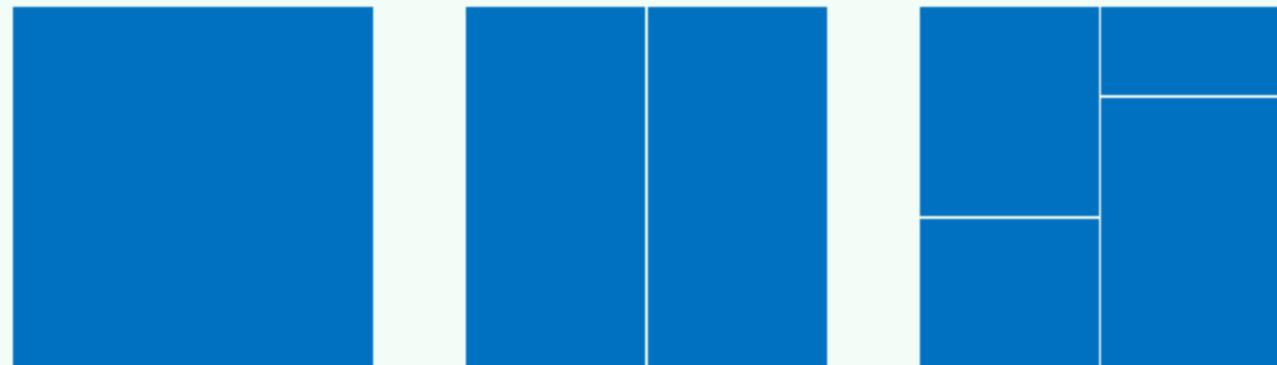
❖ 若兄弟节点 v 和 u 的父亲为 p ，则有：

$$region(p) = region(v) \dot{\cup} region(u)$$

❖ 同层的所有节点所对应的区域，无缝地覆盖了整个平面：

$$\forall 0 \leq d \leq height(T), \quad \bigcup_{depth(v)=d} region(v) = \mathbb{R}^2 \quad \text{且} \quad \bigcup_{depth(v)=d} P(v) = P$$

❖ 任何一次矩形查询的解，都可以表示为若干个节点所对应 $P(v)$ 的并 //多少个？



搜索树应用

kd-树：查询

e> - F2

我们竟为这无用的找寻浪费了这么多天
我想找寻的心上人绝对不会在这里出现

邓俊辉

deng@tsinghua.edu.cn

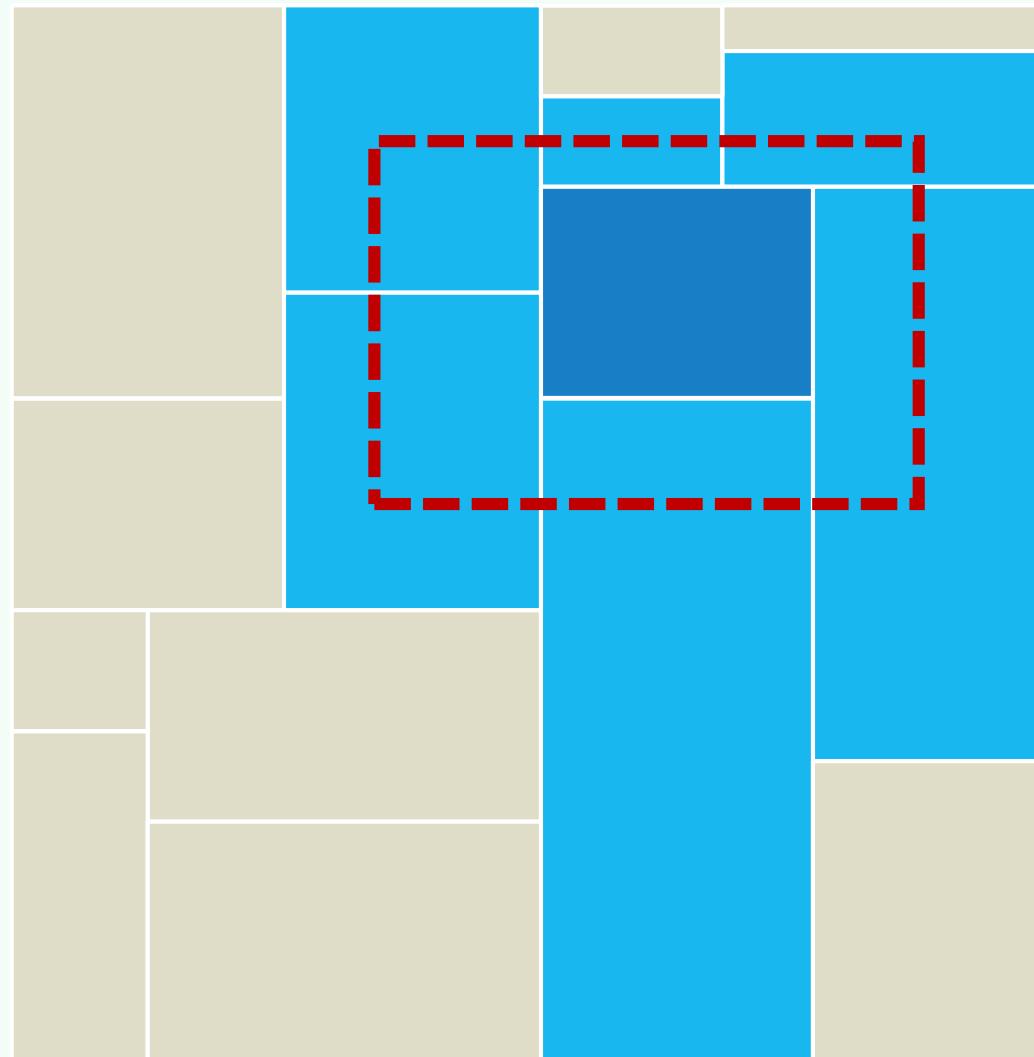
查询：算法kdSearch(v, R): 热刀来切logn层巧克力

```
// Pre-condition: region(v) ∩ ∂R ≠ ∅

- if ( isLeaf( v ) )
    if ( inside( v, R ) ) report(v)
    return

- if ( region( v->lc ) ⊆ R )
    reportSubtree( v->lc )
else if ( region( v->lc ) ∩ R ≠ ∅ )
    kdSearch( v->lc, R ) //recurse

- if ( region( v->rc ) ⊆ R )
    reportSubtree( v->rc )
else if ( region( v->rc ) ∩ R ≠ ∅ )
    kdSearch( v->rc, R ) //recurse
```



实例

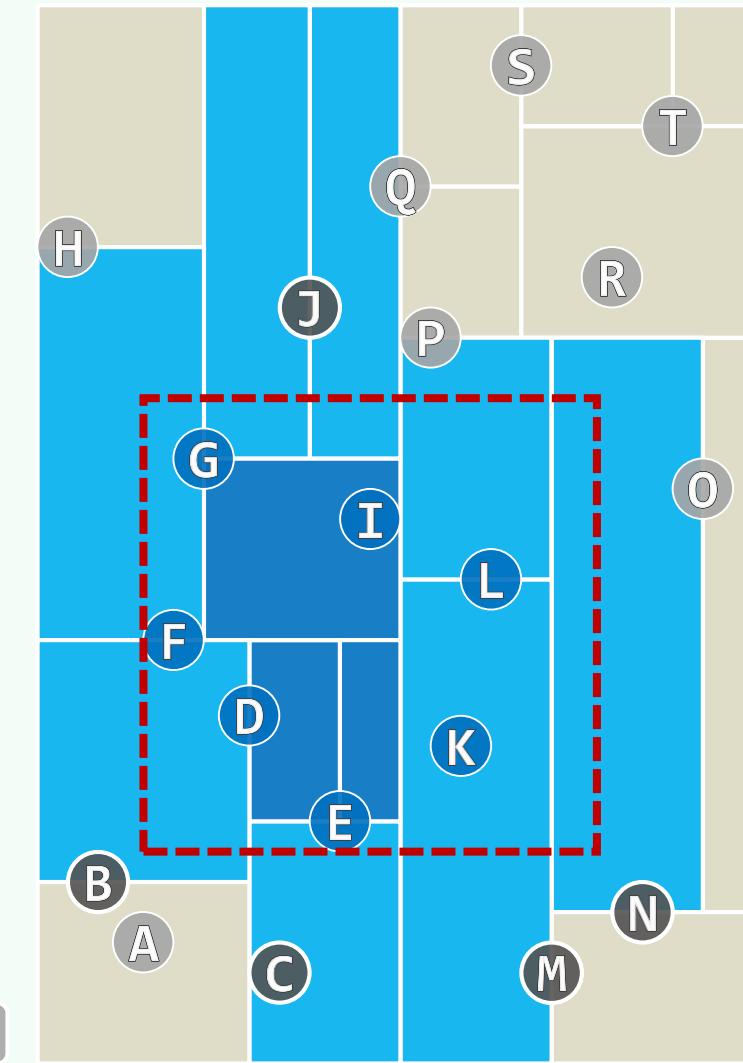
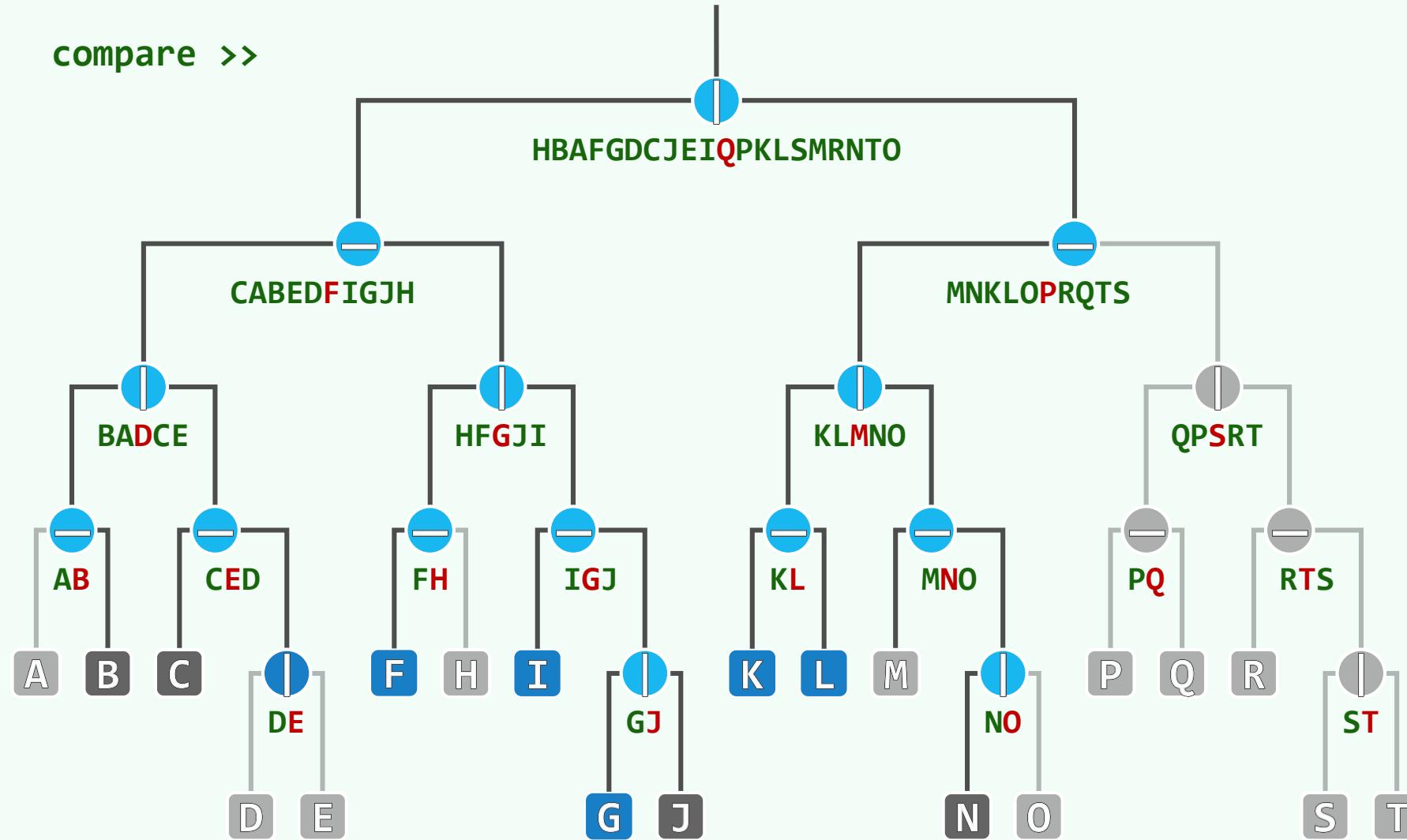
◆ 因待判而递归

◆ 命中 (个别或批量报告)

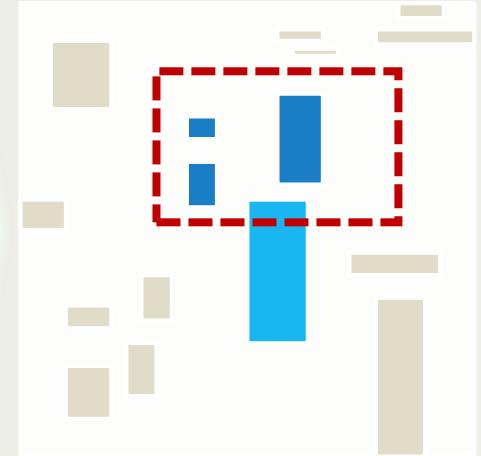
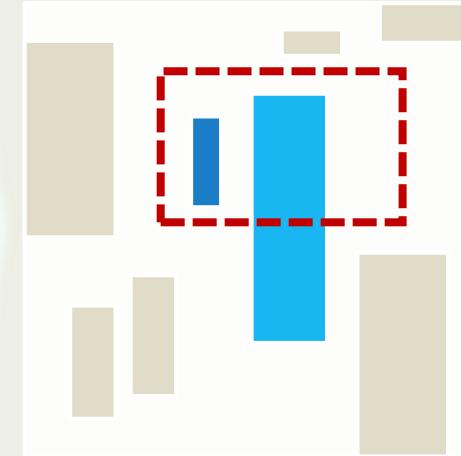
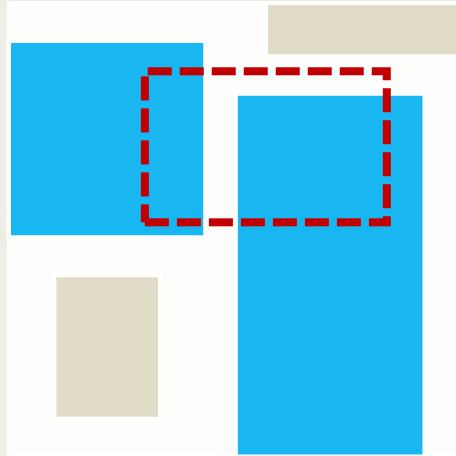
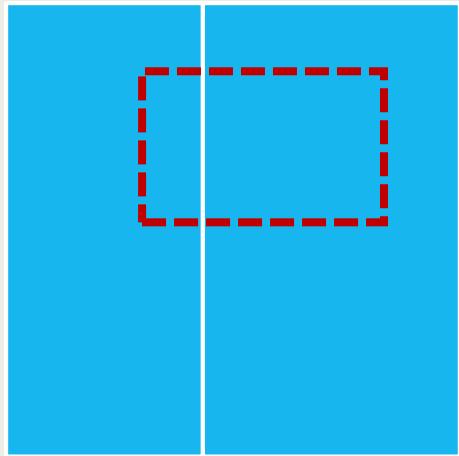
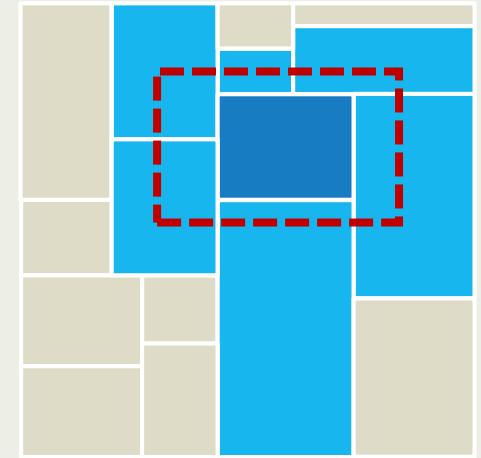
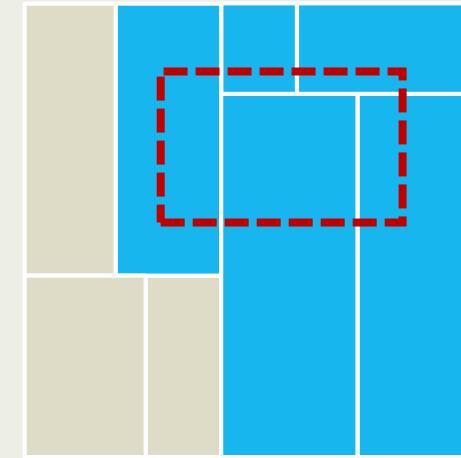
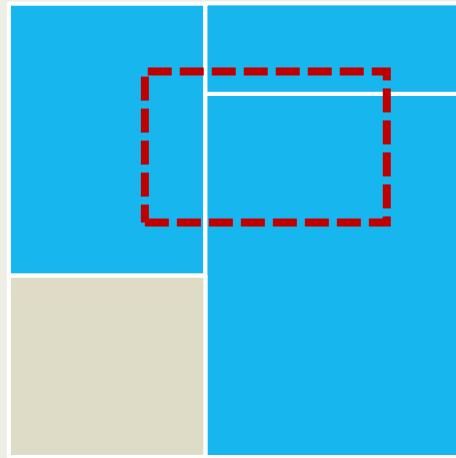
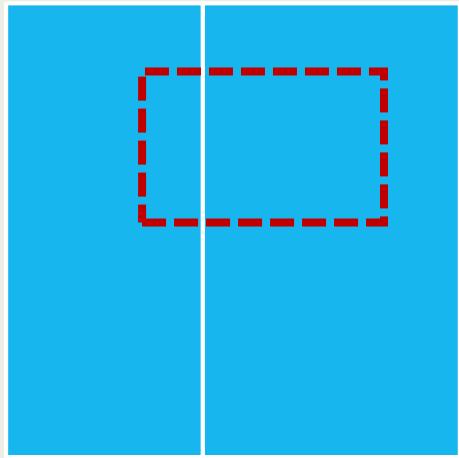
◆ 经核对后排除

◆ 因剪枝未访问

compare >>



Bounding Box



实例：Bounding Box

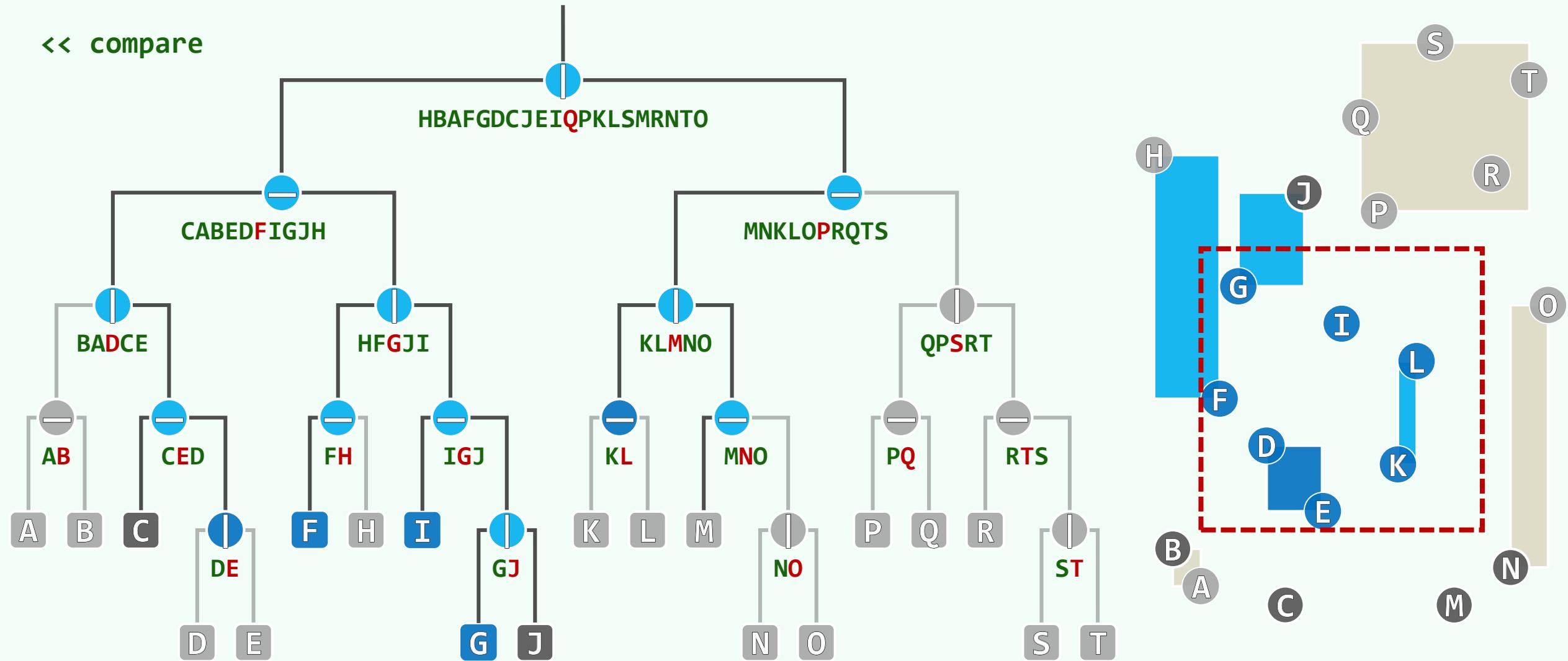
◆ 因待判而递归

◆ 命中（个别或批量报告）

◆ 经核对后排除

◆ 因剪枝未访问

<< compare



搜索树应用

kd-树：复杂度

θ> - F3

肉眼看不清细节，但他们都知道那是木星所在的位置，
这颗太阳系最大的行星已经坠落到二维平面上了

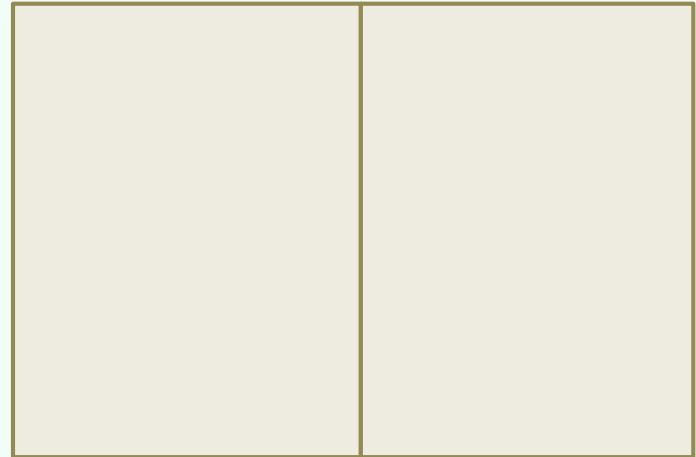
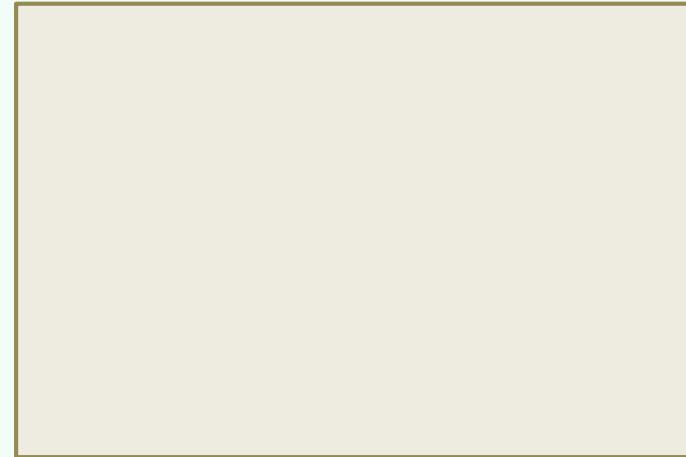
有人嘲笑这种体系说：为了能发现这个比例中项并组成政府
共同体，按照我的办法，只消求出人口数字的平方根就行了

邓俊辉

deng@tsinghua.edu.cn

预处理 + 空间消耗

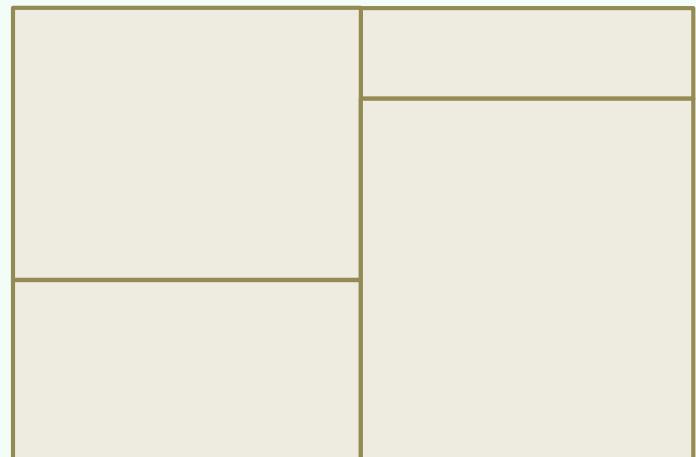
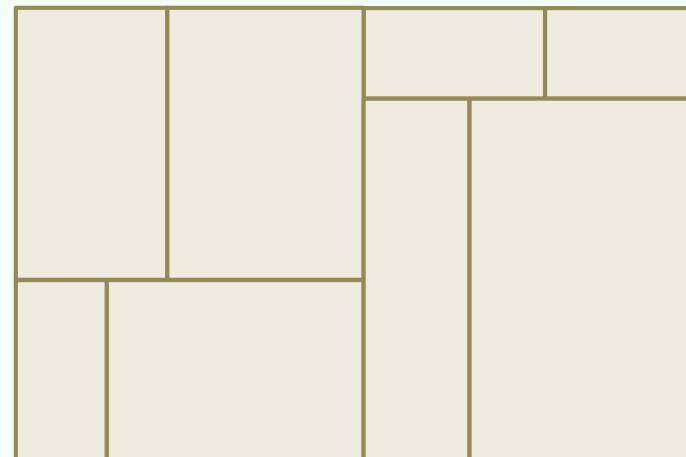
❖ 时间 $T(n) = 2*T(n/2) + \Theta(n)$
 $= \Theta(n \log n)$



❖ 树高 $= \Theta(\log n)$

❖ 空间 $= 1$

$$\begin{aligned} &+ 2 \\ &+ 4 \\ &+ \dots \\ &+ \Theta(2^{\log n}) \\ &= \Theta(n) \end{aligned}$$



查询时间 (1/2)

❖ 声明：查询 = 查找+报告 = $\mathcal{O}(\sqrt{n} + r)$

❖ 以下均就**最坏情况**而言...

❖ 查找的时间成本，决定于

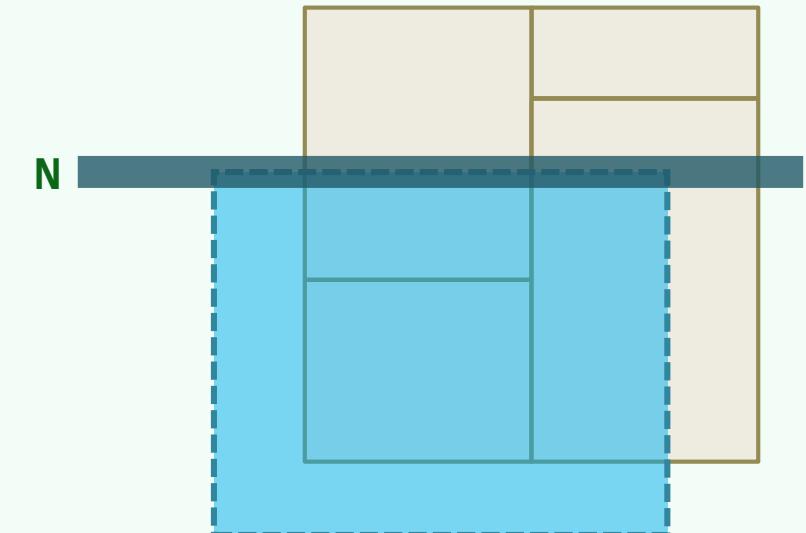
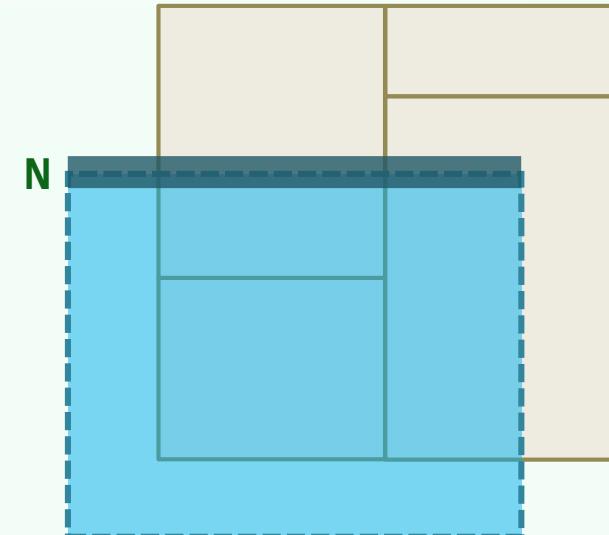
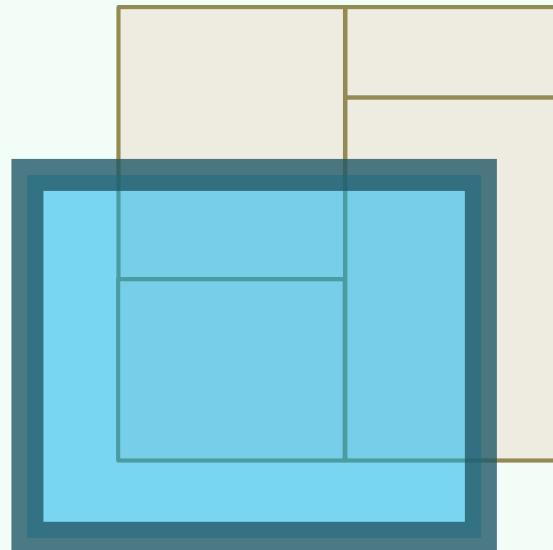
$Q(n)$ = 递归调用的总次数，亦即

与**R**边界相交的子区域（节点）总数

❖ 观察：任何一段**边界**所引发的递归总数，与 $Q(n)$ 演近相等

因此，只需考查**其中一段**，**WOLG**，**北方的那段N**

❖ 进一步地，可将**N**放大为一条直线



查询时间 (2/2)

❖ 以下，我们来导出 $Q(n)$ 的递推式

❖ 很遗憾，通常 “由父及子” 的思路

在这里行不通，需改成 “由祖及子” ...

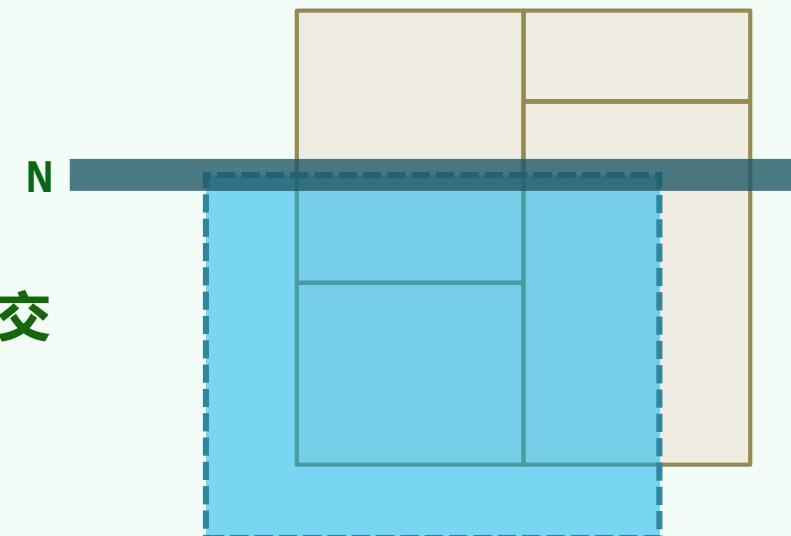
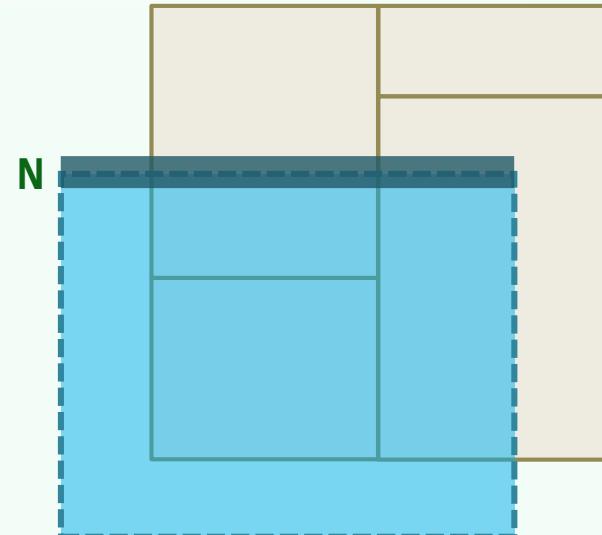
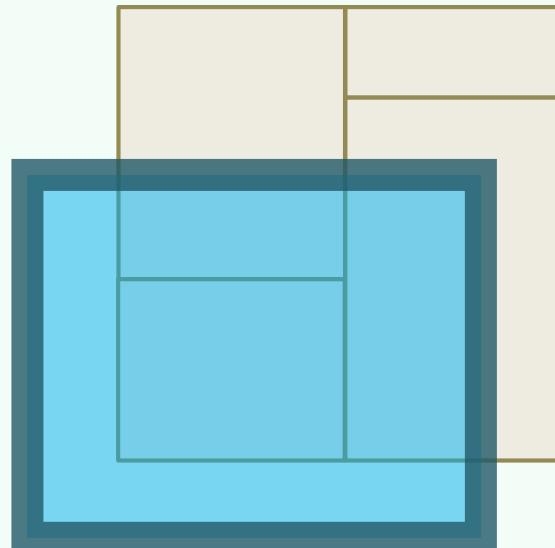
❖ 幸运的是

奇数层上发生递归的总次数，也与 $Q(n)$ 渐近相等

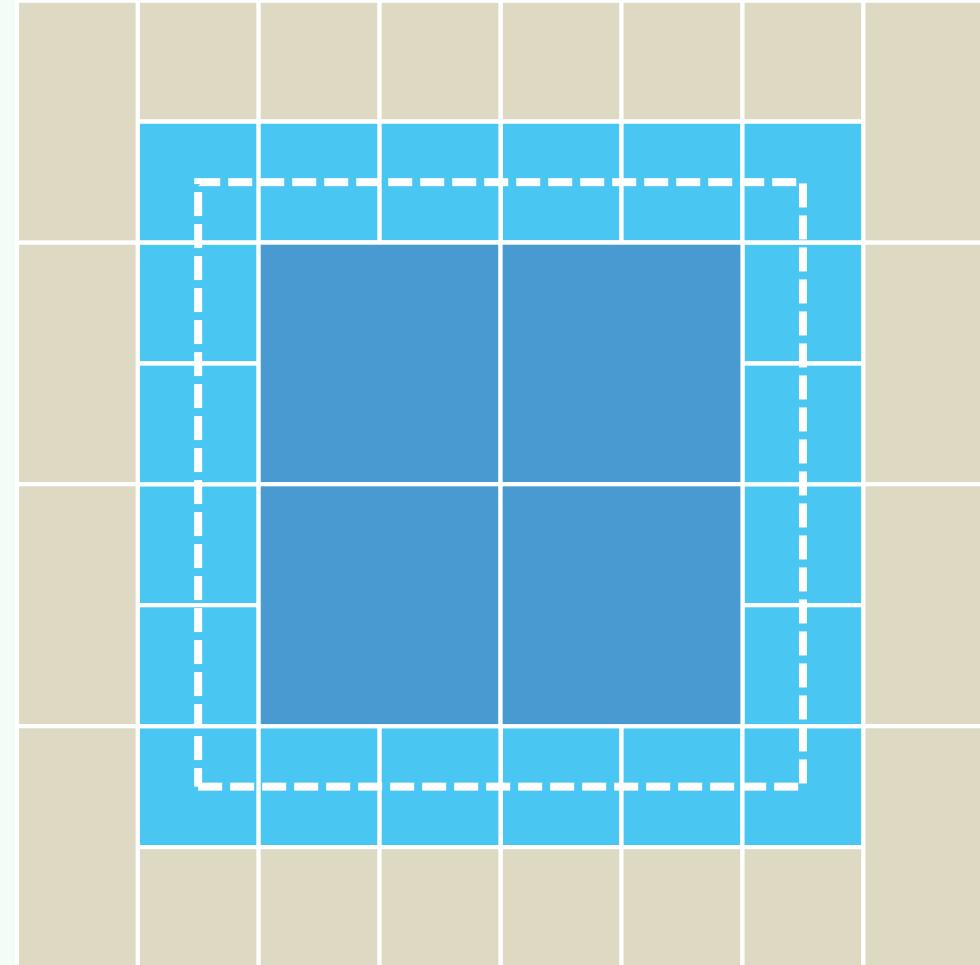
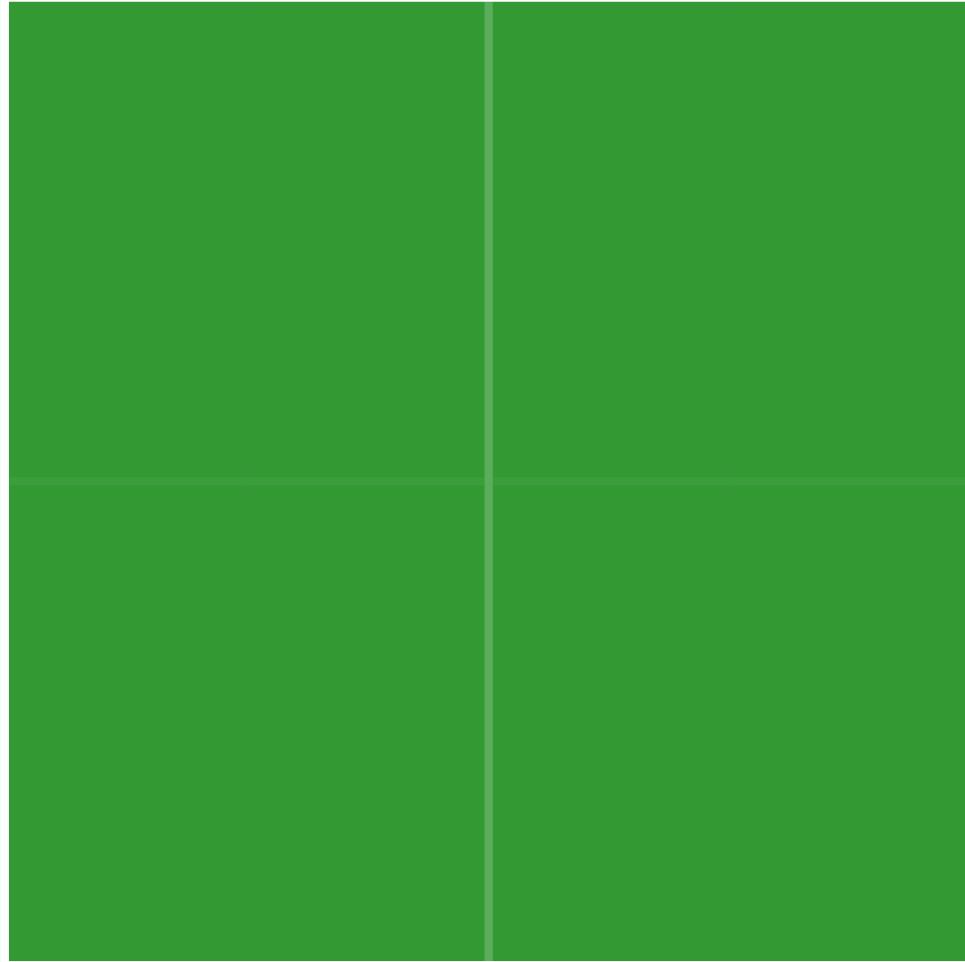
❖ 观察：

任一区域若与 N 有交，则在其四个孙子中，至多会有两个也与 N 有交

❖ 亦即： $Q(n) = 2 \cdot Q(n/4) + \mathcal{O}(1) = \mathcal{O}(\sqrt{n})$



最坏情况



更高维度

❖ 2d-树可否推广，以支持高维空间的范围查询？

在这类场合，其时间、空间效率如何？

❖ 实际上，推广并不难：

在k维空间中，只需循环地依次沿第1、第2、第3、...、第k维切分

❖ 在d维欧氏空间 \mathcal{E}^d 中，对于任意给定的 n 个点，我们都可以

- 在 $\mathcal{O}(n \log n)$ 时间内构造出
- 一棵占用 $\mathcal{O}(n)$ 空间的kd-树，并借助它
- 在 $\mathcal{O}(r + n^{1-1/d})$ 时间内完成每次正交范围查询