# kd-tree

# 8.高级搜索树

## （xb1）kd-树：一维

邓俊辉

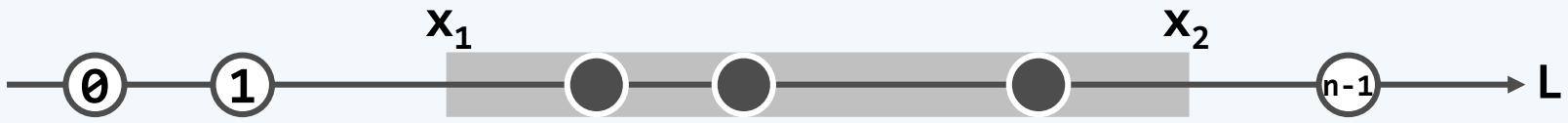
deng@tsinghua.edu.cn

# 范围查找

- 在直线L上，给定点集P = { $p_0$, ..., $p_{n-1}$ }
- 在任意区间R = [$x_1$, $x_2$]内： 有多少点counting？ 有哪些点reporting？
- 限制：点集规模n非常大，以致于需要借助外存

  因此：通过遍历整个点集，逐一测试各点将非常耗时，应尽量避免
- 问题特点： Offline P相对固定，可以离线方式预处理

  Online R数量巨大，以非确定方式在线逐个给出，须在线处理



# 蛮力法

- 依次检查P中的每个点：若位于指定区间之内，则计数（或将其加至查找结果中）
- Θ(n)时间——能否更快？就渐进意义而言似乎不能，因为...
- 最坏情况下，命中的点数 r = Ω(n) ——即便直接输出它们，也要花费Ω(n)时间
- 实际上，相对于查找过程本身，输出过程的效率不甚重要

  对于简单的Counting版，甚至不必输出命中子集

  蛮力查找过程需反复I/O，Θ(n)的常系数很大

因此，理应首先优化查找过程...

# 计数法

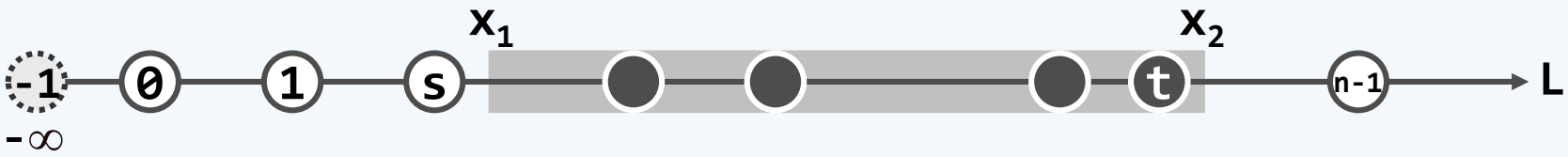
❖ 预处理：点集排序后转换为**有序向量**（哨兵$p_{-1} = -\infty$） //$O(n\log n)$

❖ 查找：针对任意区间 $R = [x_1, x_2]$

1. $t = search(x_2) = max\{ i \mid x_2 \geq p_i \}$ //$O(\log n)$

2. 从$p_t$出发，自右向左检查各点，直至首次离开查询区间的$p_s$

   只要当前点在范围之内，则报告之 //$O(r + 1) = O(t - s + 1)$



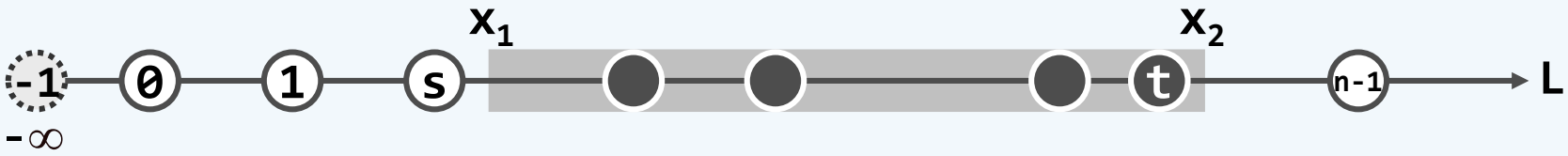
# 计数法

❖ 优势：$O(r + \log n)$，输出敏感

仅涉及$r + 1$个点，且它们依次毗邻，I/O大大节省

对于Counting版，甚至还可进一步优化至$O(\log n)$

⌛ 这个方法似乎不错，但是...



# 平面版本

- 给定平面点集

  $P = \{ p_0, \ldots, p_{n-1} \}$

- 矩形范围查找 Rectangular Range Search

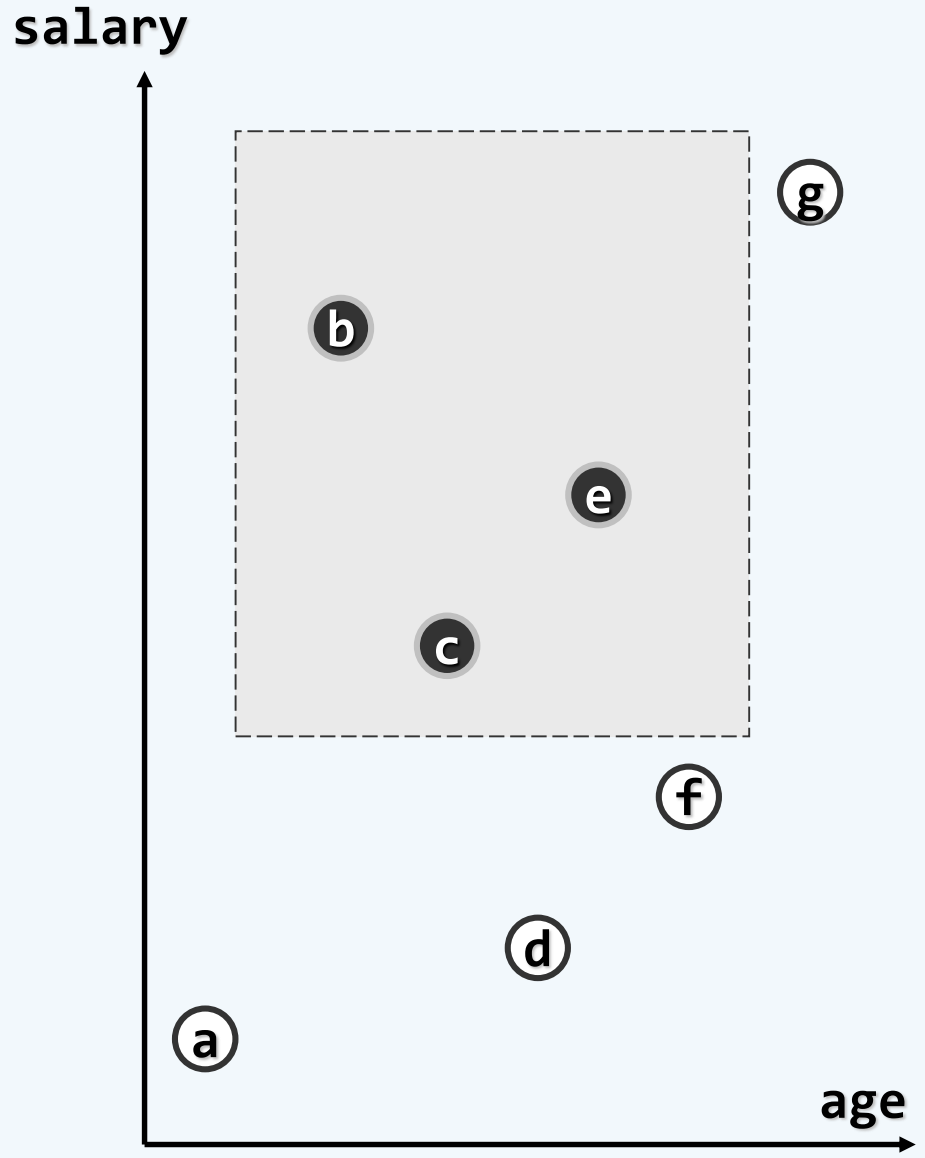
  任意矩形 $R = [x_1, x_2] \times [y_1, y_2]$ 内

  Counting　有多少个点

  Reporting　有哪些点

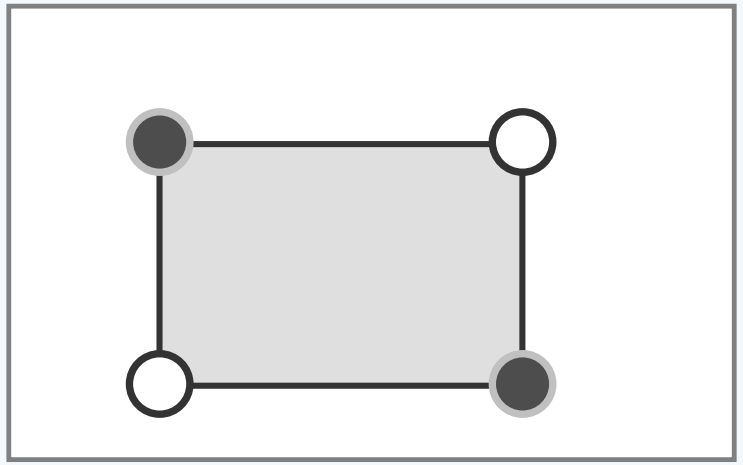
- 此时，计数法还能行得通吗?

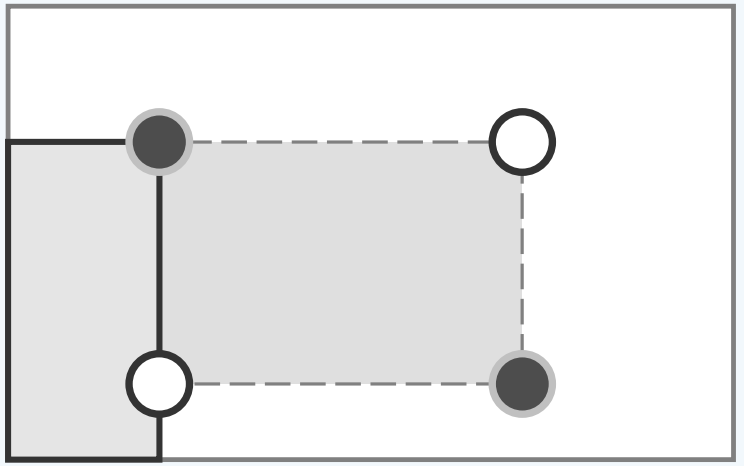
- 否则，有无其它方法?

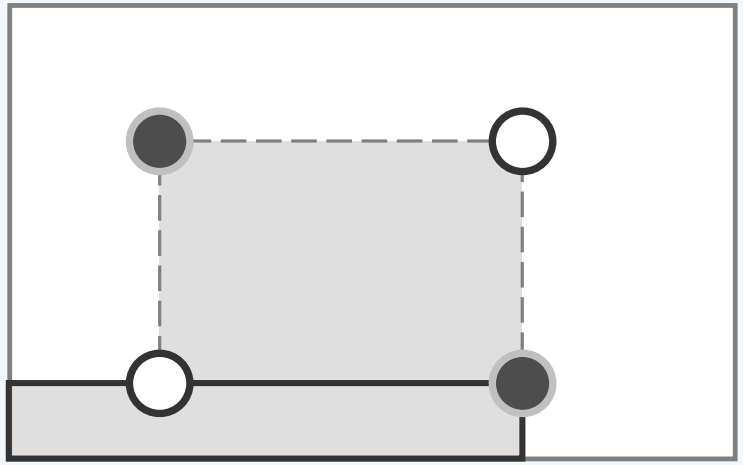


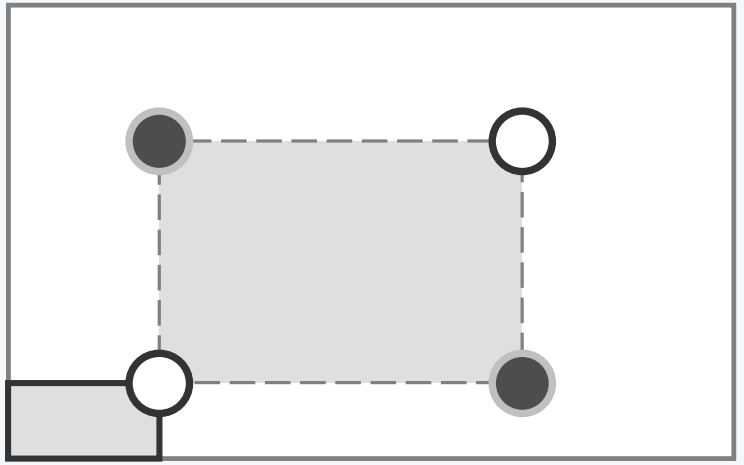
# 容斥原理

❖ 就原理而言，计数法不难推广至二维甚至更高维



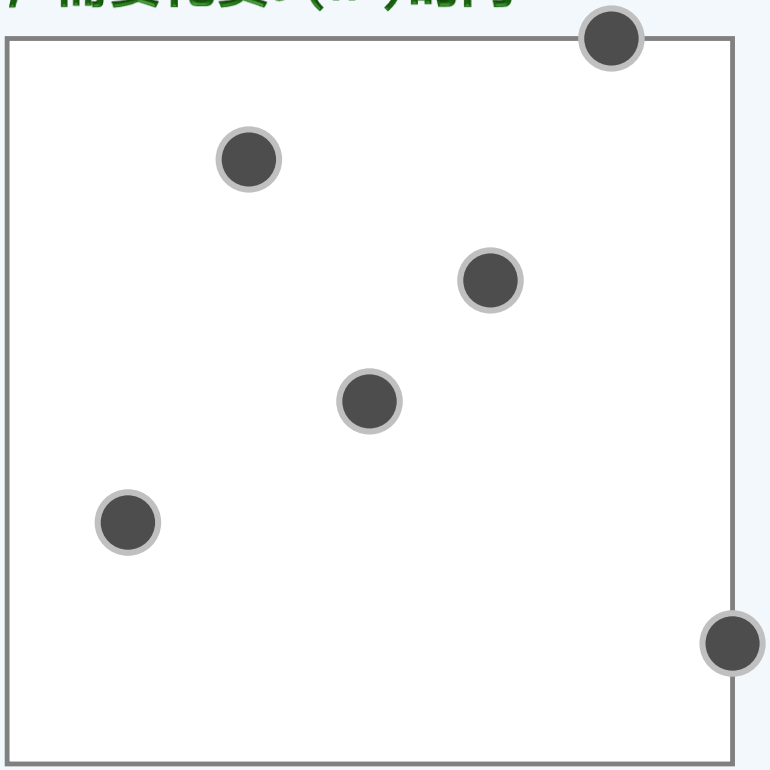


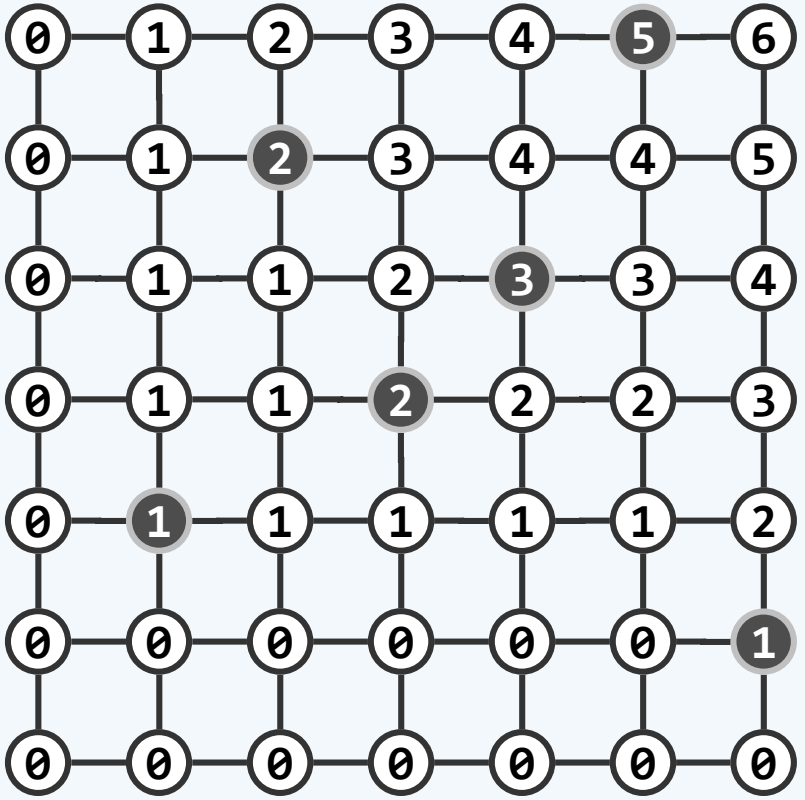




# 覆盖

- 若$u \le x$且$v \le y$，则称点$(u, v)$被点$(x, y)$覆盖dominated
- 预处理：对每个点$(x, y)$，记下$n(x, y) = |(0, x] \times (0, y] \cap P|$
- 为此，需要花费$O(n^2)$时间



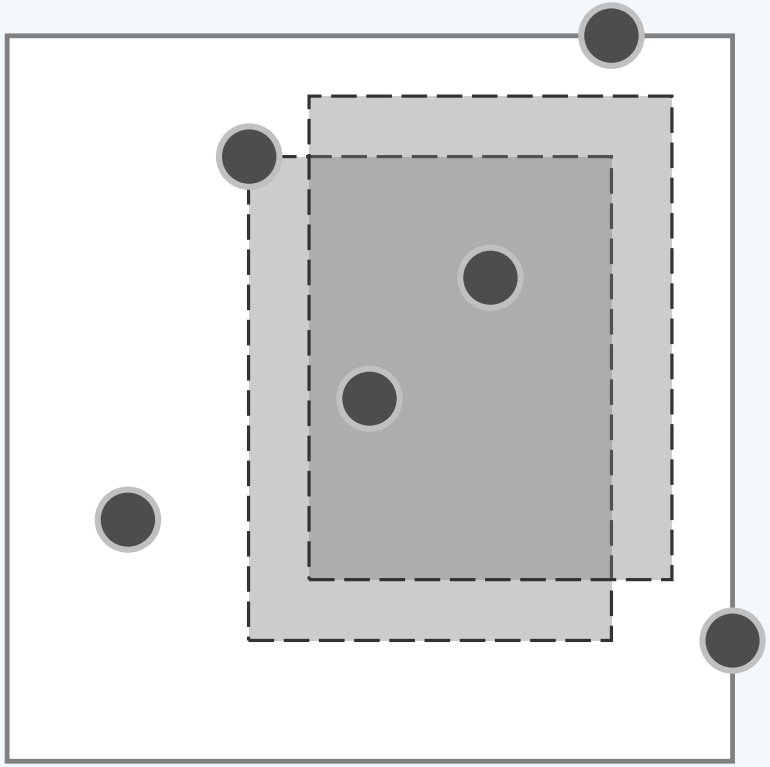


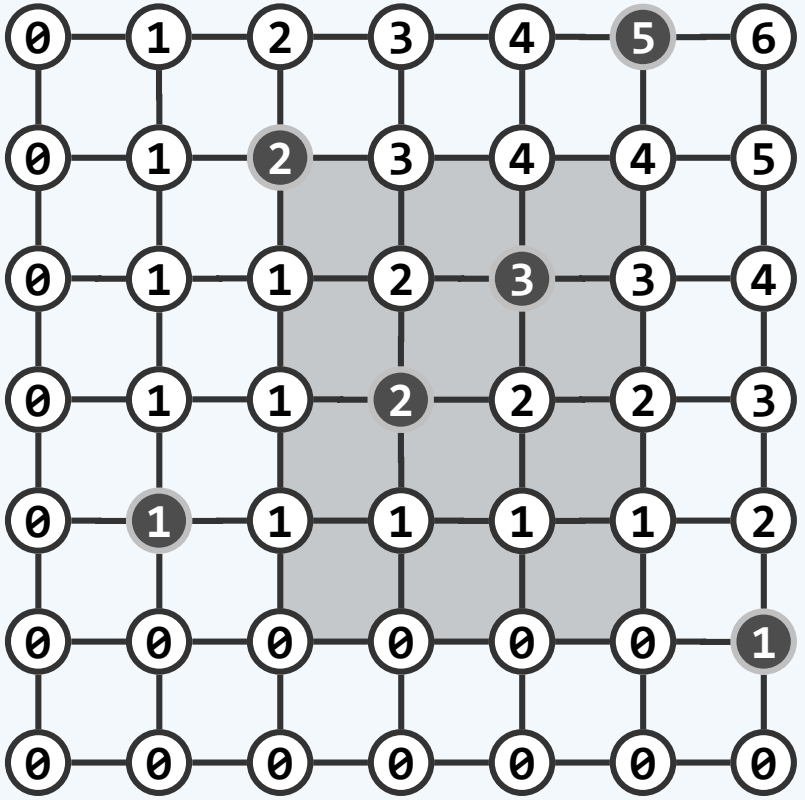
# 容斥原理

❖ 于是对任意的$R = (x_1, x_2] \times (y_1, y_2]$，有

$$|R \cap P| = \boxed{n(X_1, Y_1) + n(X_2, Y_2) - n(X_1, Y_2) - n(X_2, Y_1)}$$

❖ 如何推广至全闭的矩形区域？若允许垂直或水平共线的点呢？

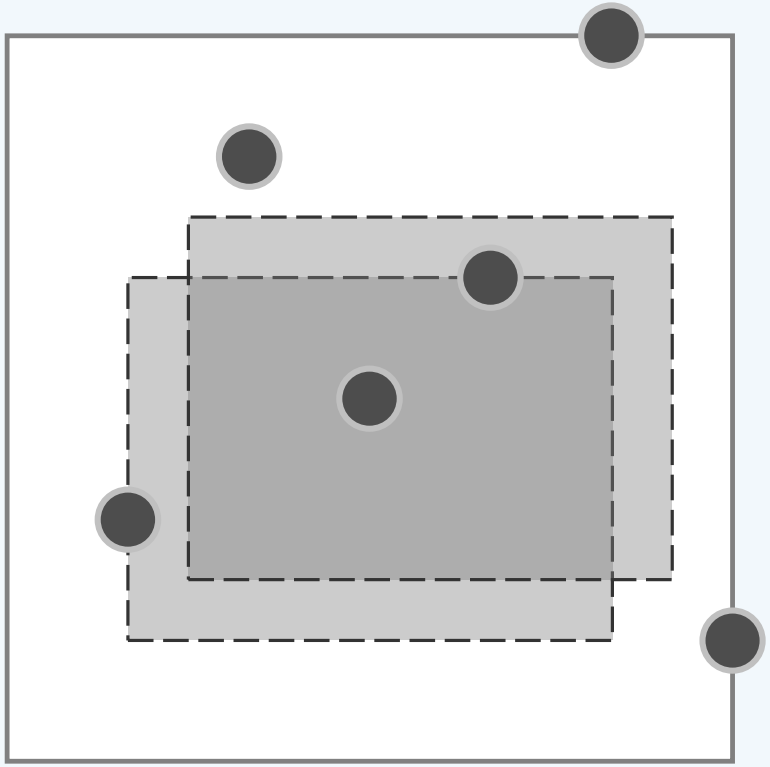


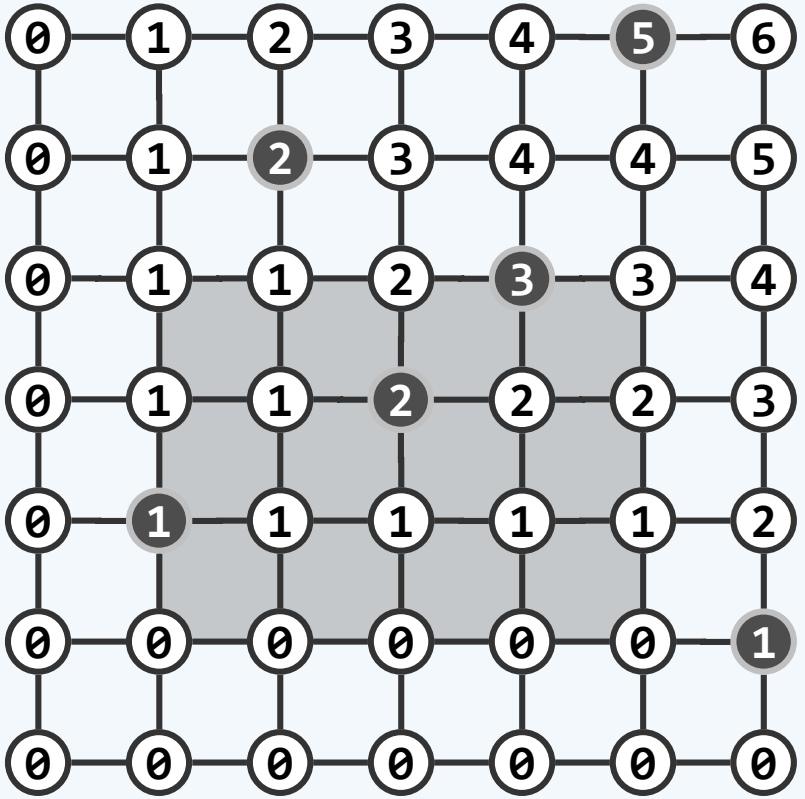


# 容斥原理

❖ 查询非常快，只需$O(\log n)$时间；但共需$\Theta(n^2)$空间——倘若n较大，或者空间的维度更高...

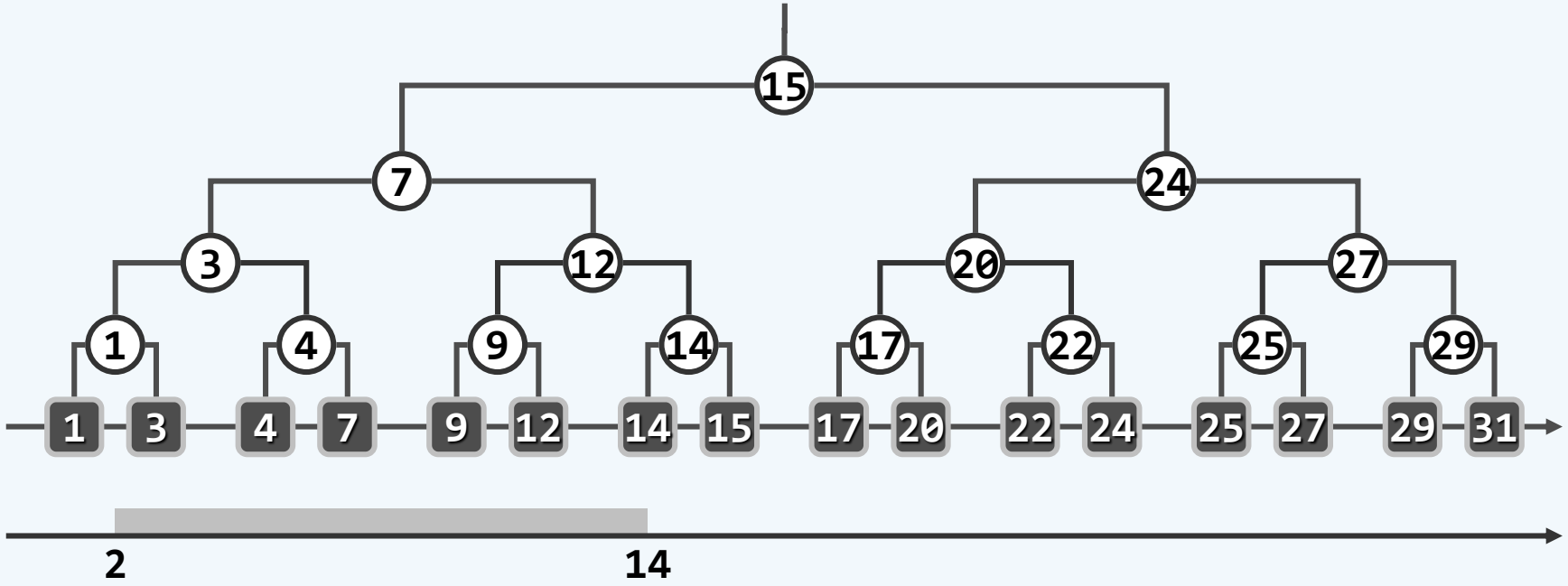
❖ 还是再次回到一维情况，找出更为一般性的方法，以便推广...





# BBST：结构

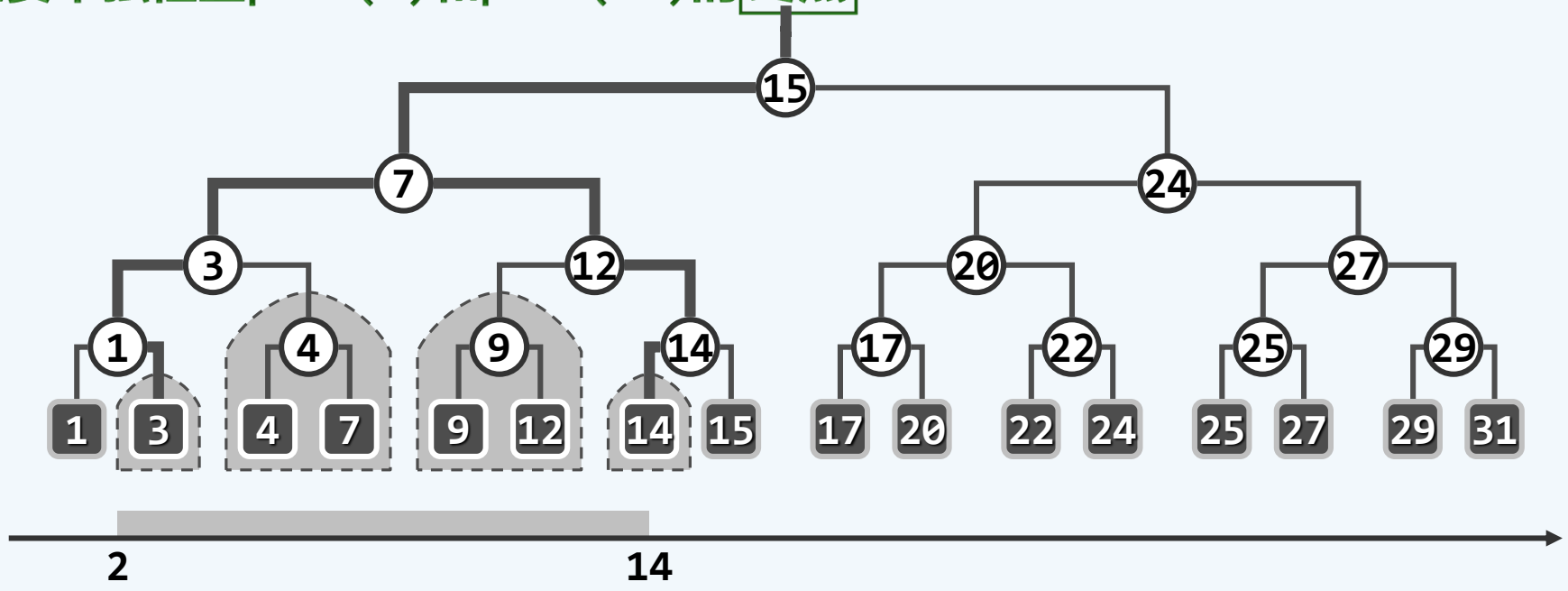
- 每个内部节点v，记录相应的划分位置x(v)
- 有序性：$\boxed{\text{LTree}(v) \le x(v) < \text{RTree}(v)}$
- 比如x(v) = 12，有max{9, 12} ≤ 12 < min{14, 15}
- 以R = [2, 14]为例，范围查找如何实现?



# BBST：查找

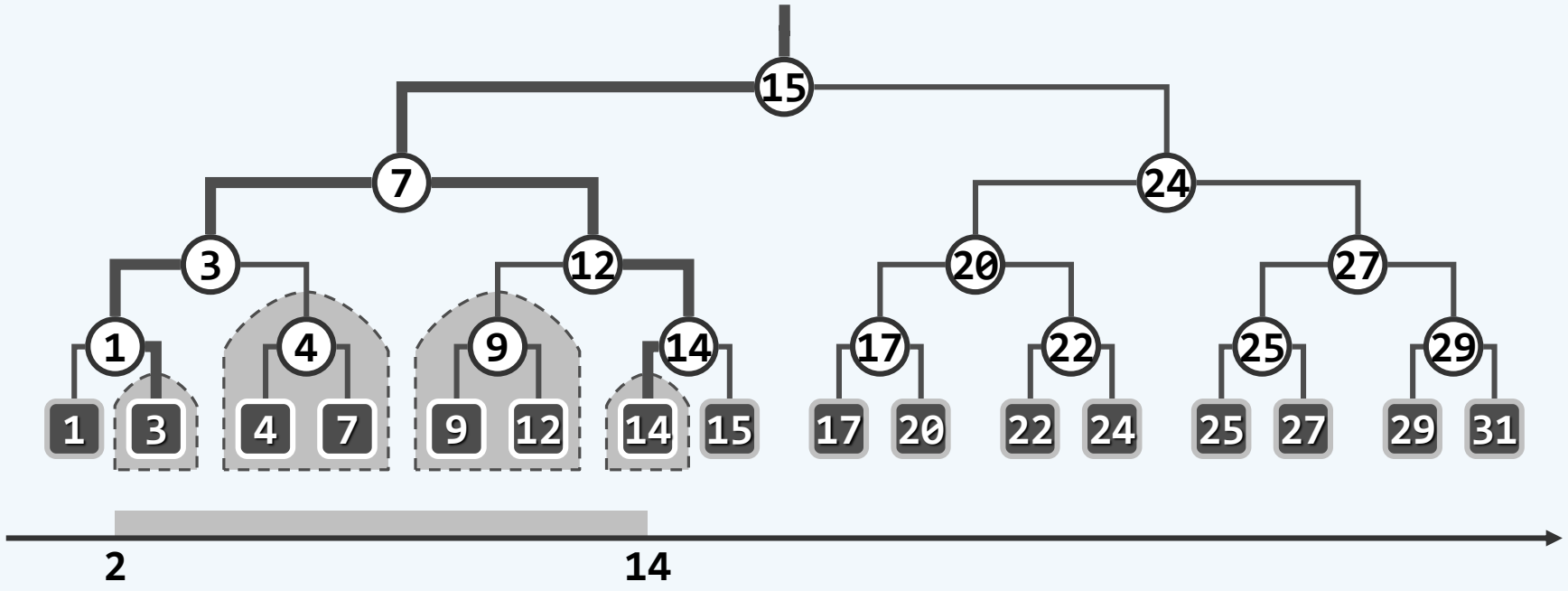
- 分别查找2和14，终止于3和14　　//其最低共同祖先LCA(3，14) = 7
- 从LCA起向下，重走一遍path(3)和path(14)

  沿path(3)/path(14)，忽略右/左拐，一旦左/右拐则输出右/左子树
- 最后，还要单独检查path(3)和path(14)的终点



# BBST：效率

- 预处理　O(nlogn)
- 空间　O(n)
- 查询　O(r + logn), r = 查找命中的点数 输出敏感



# 8.高级搜索树

## （xb2）kd-树：二维

**邓俊辉**

deng@tsinghua.edu.cn

# 构思

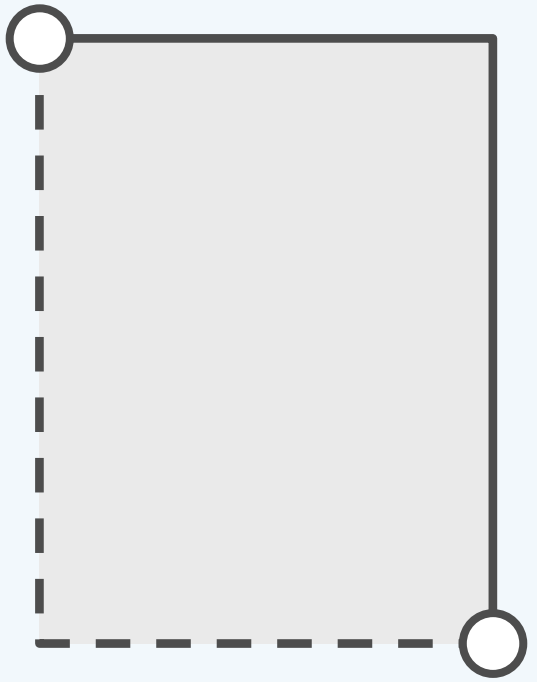
- 上述数据结构，如何扩展到二维?
- 通过递归的平面划分，构造kd-树!

  偶/奇数深度层：做垂直/水平划分

  两个子集规模尽可能接近median

  半平面：左开右闭、下开上闭

  非退化约定：各点x坐标互异、y坐标互异



1975 ACM Student Award
Paper: Second Place

# Multidimensional Binary Search Trees Used for Associative Searching

Jon Louis Bentley
Stanford University

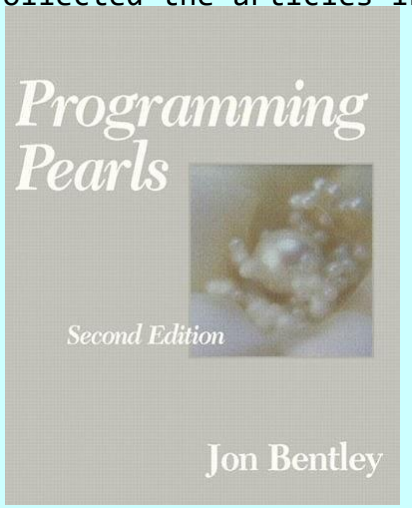
**This paper develops the multidimensional binary search tree (or $k$-d tree, where $k$ is the dimensionality of the search space) as a data structure for storage of information to be retrieved by associative searches. The $k$-d tree is defined and examples are given. It is shown to be quite efficient in its storage requirements. A significant advantage of this structure is that a single data structure can handle many types of queries very efficiently. Various utility algorithms are developed; their proven average running times in an $n$ record file are: insertion, $O(\log n)$; deletion of the root, $O(n^{(k-1)/k})$; deletion of a random node, $O(\log n)$; and optimization (guarantees logarithmic performance of searches), $O(n \log n)$. Search algorithms are given for partial match queries with $t$ keys specified [proven maximum running time of $O(n^{(k-t)/k})$] and for nearest neighbor queries [empirically observed average running time of $O(\log n)$.] These performances far surpass the best currently known algorithms for these tasks. An algorithm is presented to handle any general intersection query. The main focus of this paper is theoretical. It is felt, however, that $k$-d trees could be quite useful in many applications, and examples of potential uses are given.**

**Key Words and Phrases: associative retrieval, binary search trees, key, attribute, information retrieval system, nearest neighbor queries, partial match queries, intersection queries, binary tree insertion**
**CR Categories: 3.63, 3.70, 3.74, 4.49**

Jon Bentley 世界著名计算机科学家，被誉为影响算法发展的十位大师之一。他先后任职于卡内基-梅隆大学（1976~1982）、贝尔实验室（1982~2001）和Avaya实验室（2001年至今）。在卡内基-梅隆大学担任教授期间，他培养了包括Tcl语言设计者John Ousterhout、Java语言设计者James Gosling、《算法导论》作者之一Charles Leiserson在内的许多计算机科学大家。2004年荣获Dr.Dobb's程序设计卓越奖。

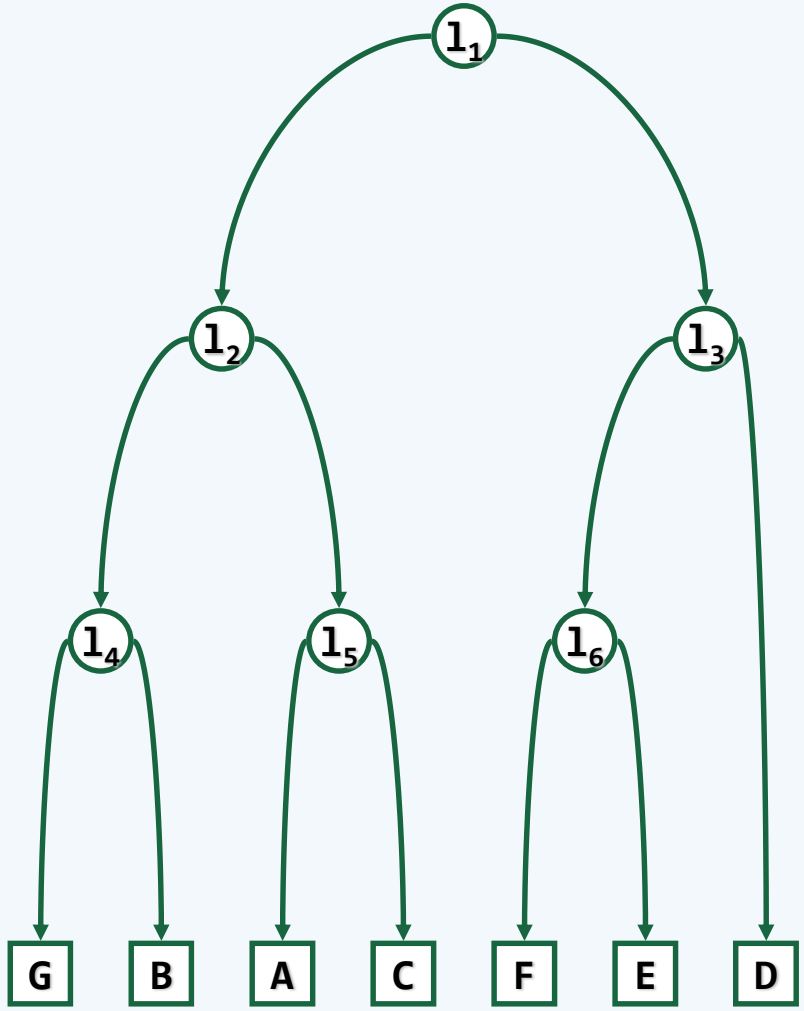
Bentley received a B.S. in mathematical sciences from Stanford University in 1974, and M.S. and Ph.D in 1976 from the University of North Carolina at Chapel Hill; while a student, he also held internships at the Xerox Palo Alto Research Center and Stanford Linear Accelerator Center. Bentley moved to Bell Laboratories, where he co-authored an optimized Quicksort algorithm with Doug McIlroy

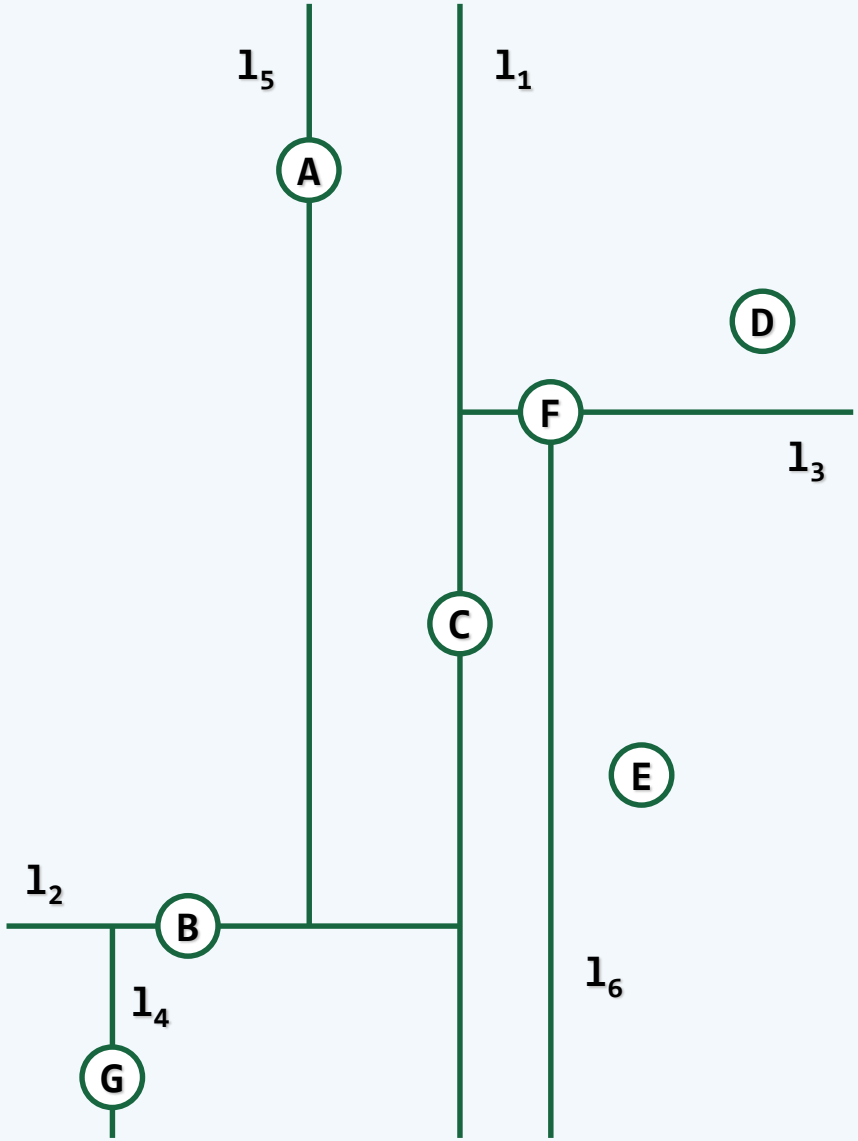
He wrote the Programming Pearls column for the Communications of the ACM magazine, and later collected the articles into two books of the same name.




Bentley, J. L. (1975). "Multidimensional binary search trees used for associative searching". Communications of the ACM. 18 (9): 509–517. doi:10.1145/361002.361007.

# 构思实例

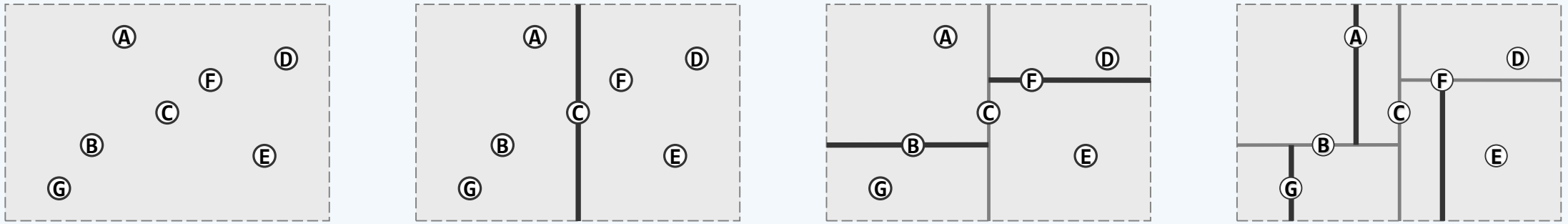


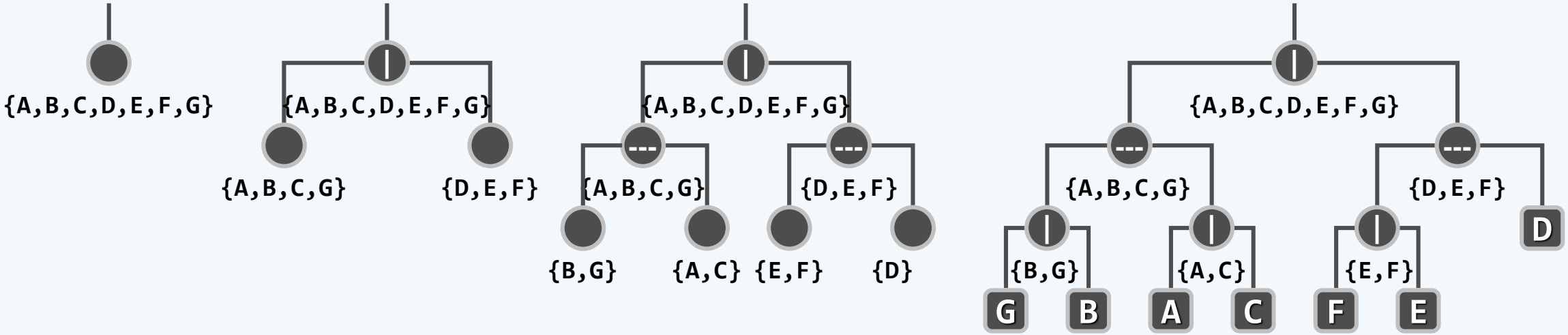



# 构造算法

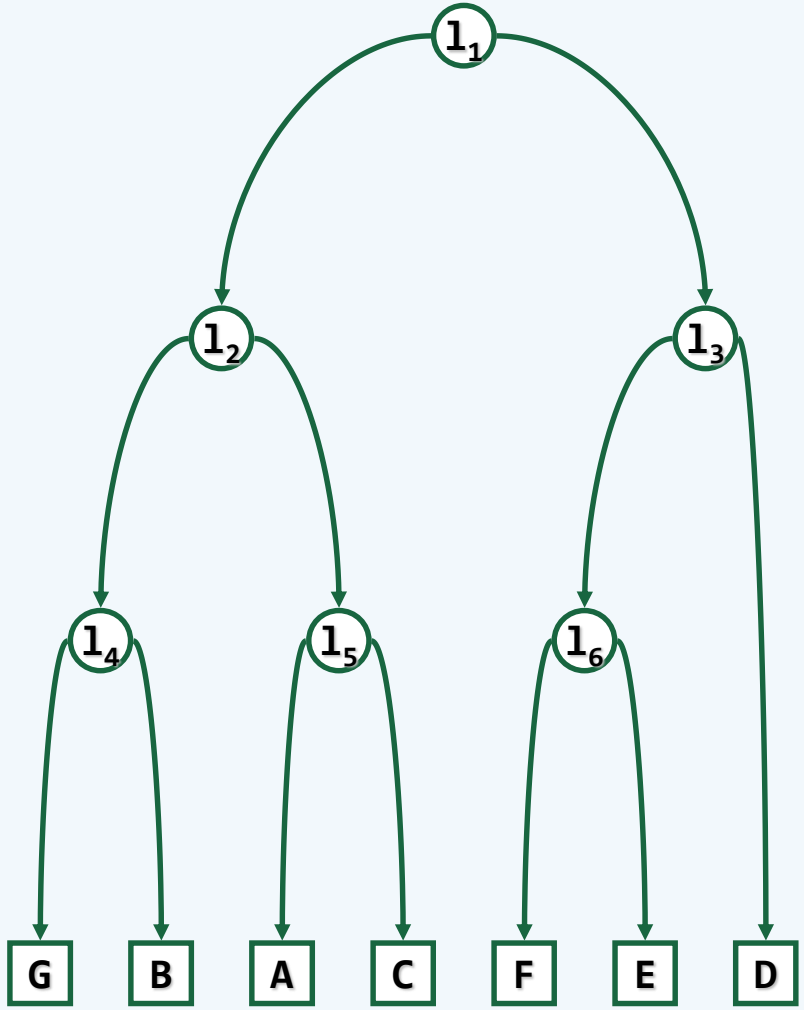
```
KdTree* createKdTee( P, depth ) { //自顶而下，递归地逐层构造
   if ( P == {p} ) return createLeaf( p ); //递归基：区域缩小至仅含一个点
   root = createKdNode(); //创建节点
   root->splitDirection = depth % 2 ? HORIZONTAL : VERTICAL; //切分方向
   root->splitLine = median( root->splitDirection, P ); //切分位置
   ( P1, P2 ) = divide( P, root->splitDirection, root->splitLine ); //切分
   root->lc = createKdTee( P1, depth + 1 ); //递归构造左或上子树
   root->rc = createKdTee( P2, depth + 1 ); //递归构造右或下子树
   return root;
}
```

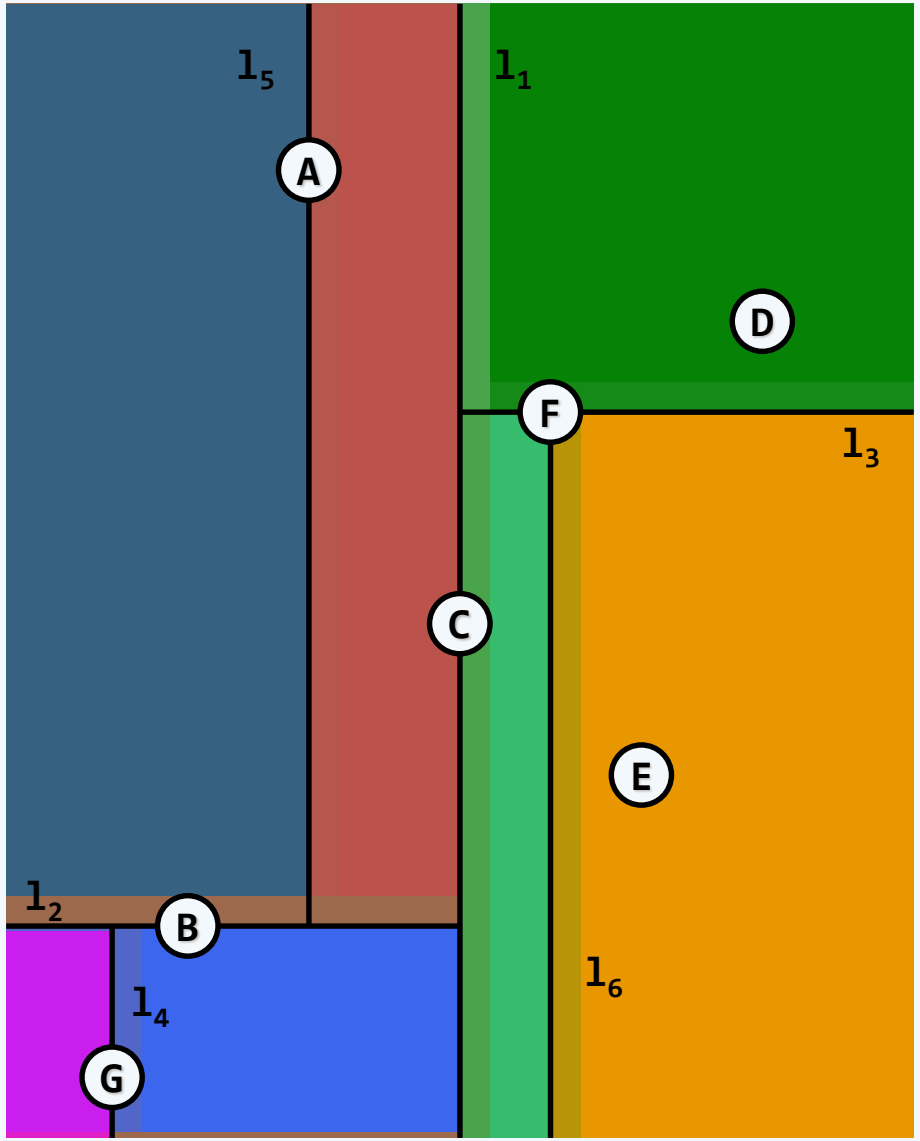
# 构造实例



# 构造实例



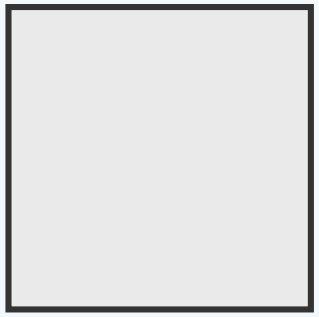
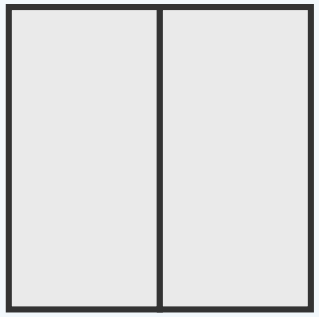
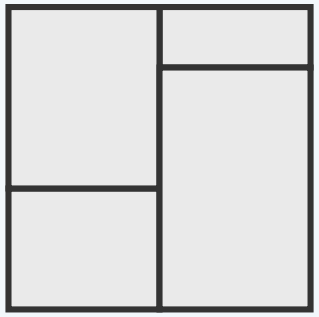
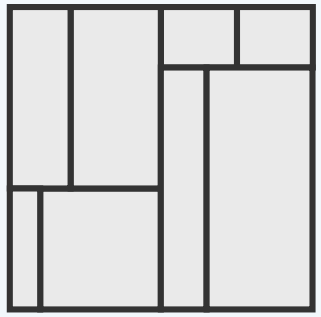



# 正则子集

- 树节点 ~ 矩形子区域
- 同一节点左、右孩子所对应子区域之并，即该节点对应的子区域
- 同一层的所有节点对应的子区域

  互不相交，而且

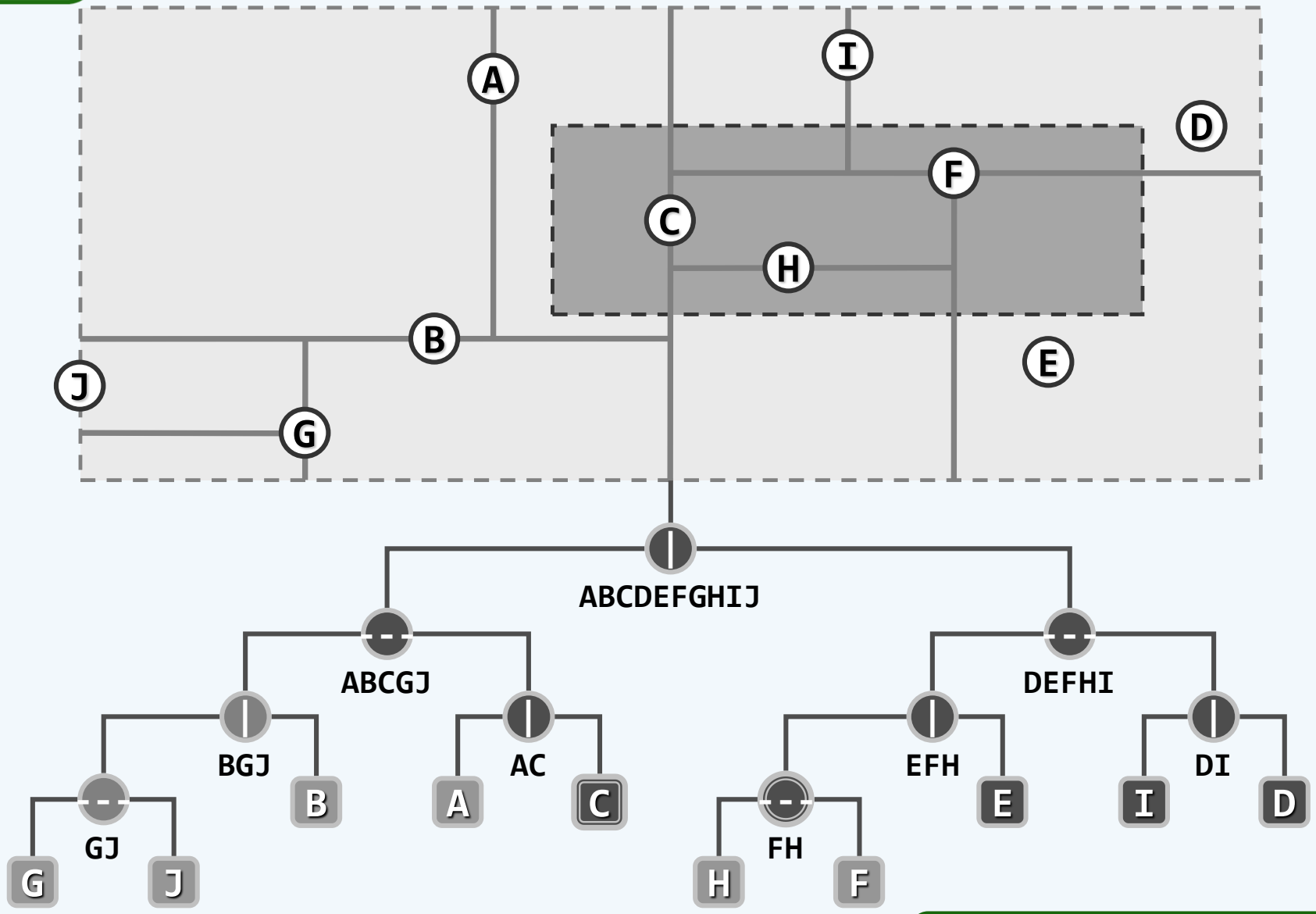
  其并恰好覆盖整个平面

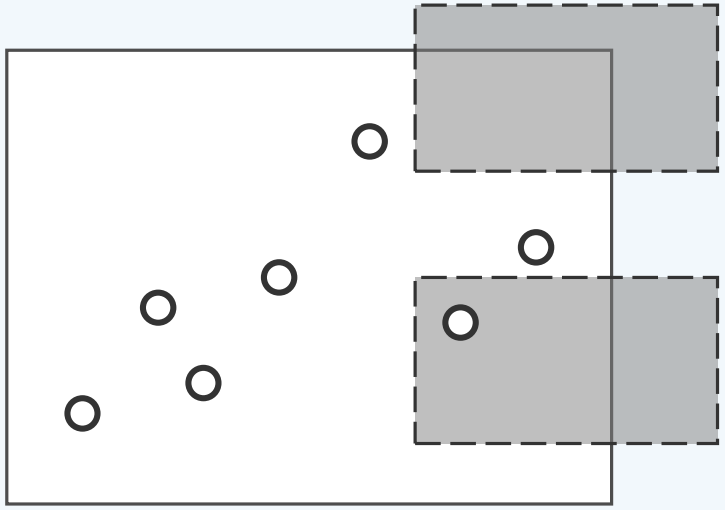
# 查找算法

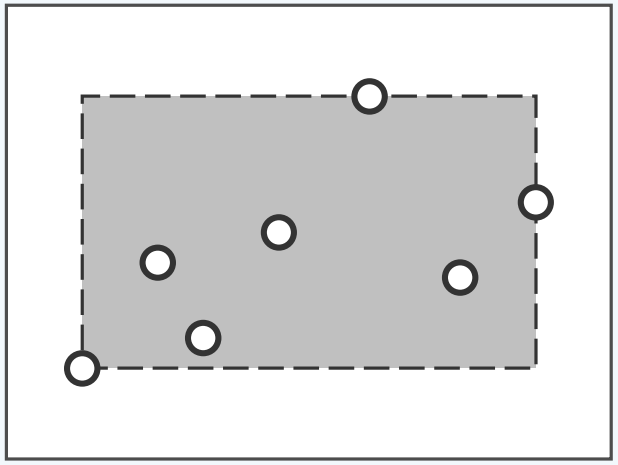
```
kdSearch( v, R ) {
   if ( isLeaf( v ) ) { if ( inside( v, R ) ) report( v ); return; }
   if ( region( v->lc ) ⊆ R ) reportSubtree( v->lc );
   else if ( intersect( region( v->lc ), R ) ) kdSearch( v->lc, R );
   if ( region( v->rc ) ⊆ R ) reportSubtree(v->rc);
   else if ( intersect( region( v->rc ), R ) ) kdSearch( v->rc, R );
}
```

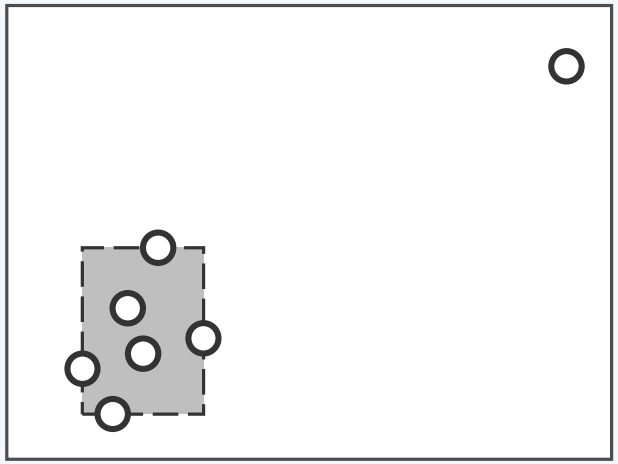
# 查找实例

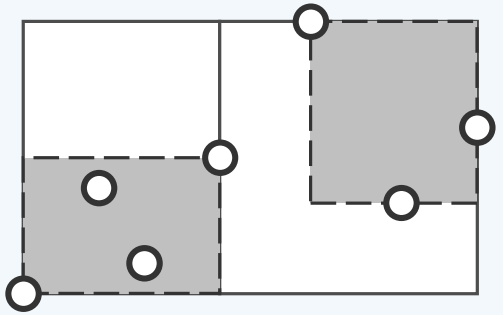

A
I
D
F
C
H
B
E
J
G
ABCDEFGHIJ
ABCGJ
DEFHI
BGJ
AC
EFH
DI
GJ
FH
G
J
B
A
C
H
F
E
I
D

# 包围盒







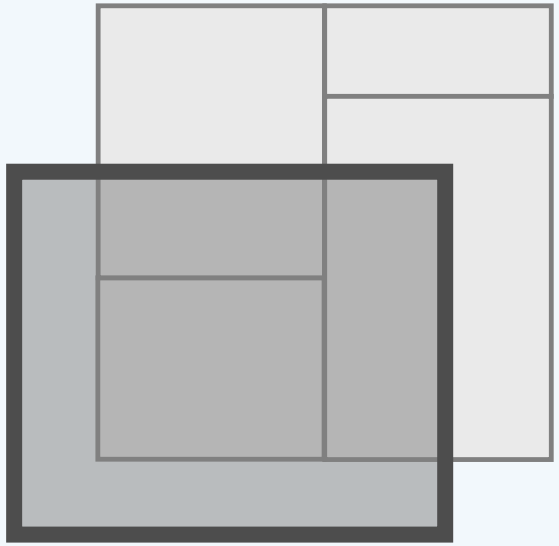


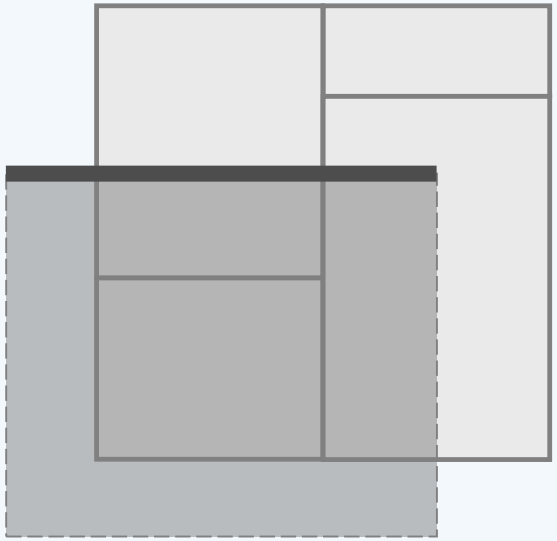
# 效率

- 空间　　O(n)

  预处理　　O(nlogn)

  查询　　O($\sqrt{n}$ + r)

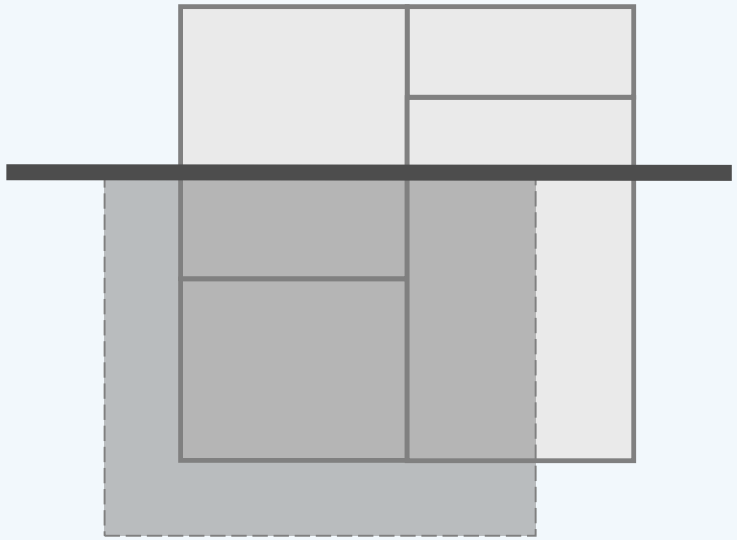




- 除去O(r)，其余查找时间决定于

  递归发生的次数，或

  待查找区域边界与子区域相交的数目Q(n)



- 递推关系：Q(1) = O(1)，Q(n) = 2 + 2*Q(n/4)

- 解得：Q(n) = O( $\sqrt{n}$ )

# 高维推广

- kd = k-dimensional
- 构造　深度 = 0 / 1 / ... / d-1时，沿第0 / 1 / ... / d-1维划分

  深度 = d / d+1 / ... / 2d-1时，沿第0 / 1 / ... / d-1维划分

  ...

  深度 = k时，沿第k % d维划分
- 空间:　O(n)
- 预处理:　O(nlogn)
- 查询:　O( $n^{1-1/d}$ + r )

## 课后

- 在高维情况下，kd-树的效率如何评价？
- 如何消除（多点共垂直、水平线等）退化情况？
- 不借助median算法，如何构造2d-树？你所给算法的效率如何？
- 若只需计数，kd-树可在多少时间内给出解答？为此需对树做何调整？
- 适用于低（2~3）维数据的简化变种：quadtree，octree
- 针对三维以上的情况，更好的结构：multi-level search tree
  range tree
  interval tree
  priority search tree
  segment tree

# Reference


- *Bentley, J. L. (1975). "Multidimensional binary search trees used for associative searching". Communications of the ACM. 18 (9): 509–517. doi:10.1145/361002.361007.*

# Next

- 拓扑排序
- 数据结构（C++语言版）第三版 Chapter 6.8