Orthogonal Range Searching

نجمه نوری

۹۰۱۱۱۴۴

تاریخ ارائه :۹۱/۰۱/۱۵

Motivation

- Many types of questions - **queries** - about data in a database can be interpreted geometrically
- To this end we transform records in a database into points in a multi-dimensional space, and we transform the queries about the records into queries on this set of points

Example

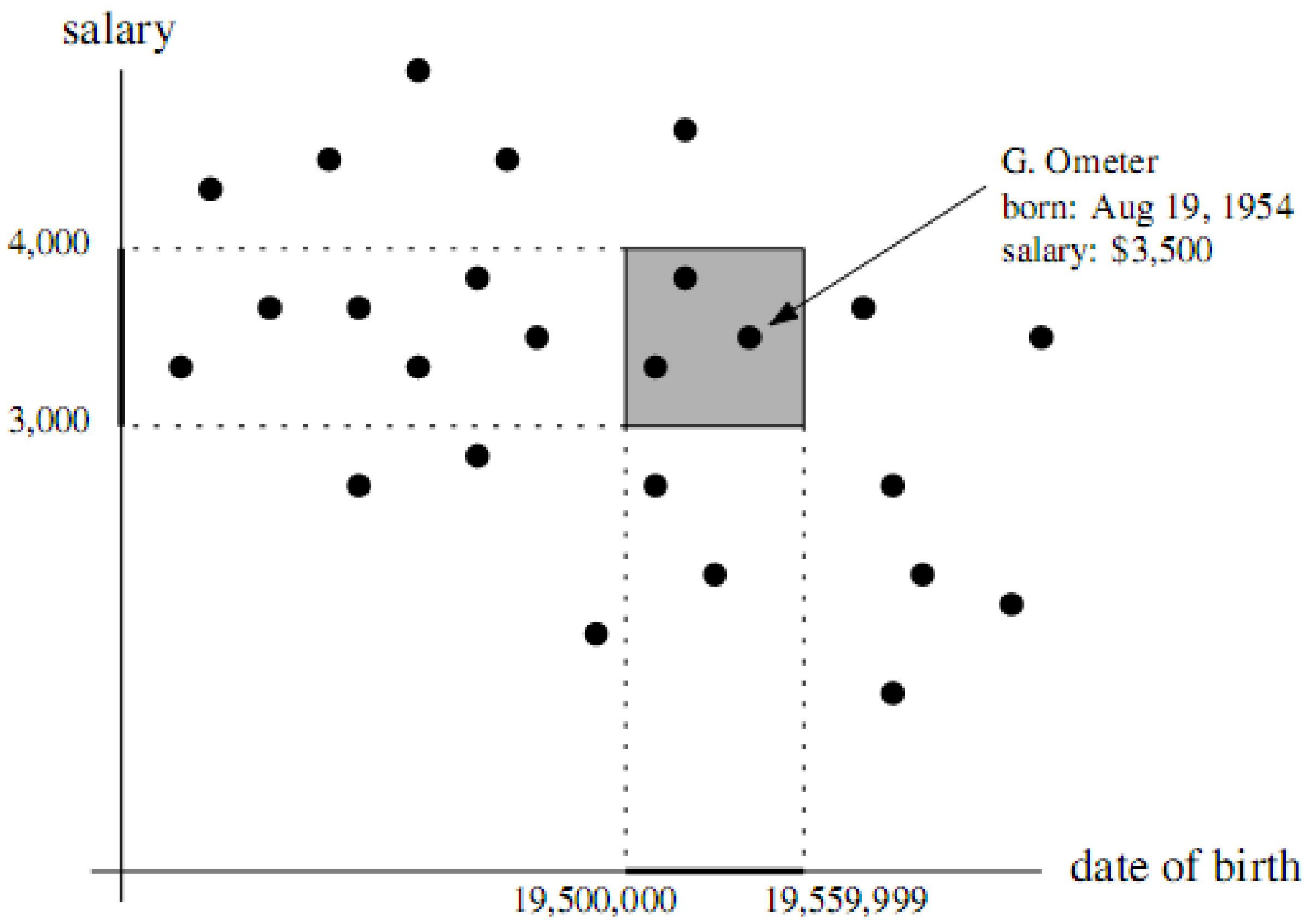
- Database for personnel administration
- In such database the name, address, date of birth, salary, and so on, of each employee are stored
- A typical query one may want to performe is to report all employees born between 1950 and 1955 who earn between \$3,000 and \$4,000 a month
- To formulate this as a geometric problem we represent each employee by a point in the plane
- The first coordinate of the point is the date of birth, represented by the integer

$$10000 \times year + 1000 \times month + day$$

- the second coordinate is the monthly salary
- With the point we store the other information

3

Interpreting a database query gmetrically



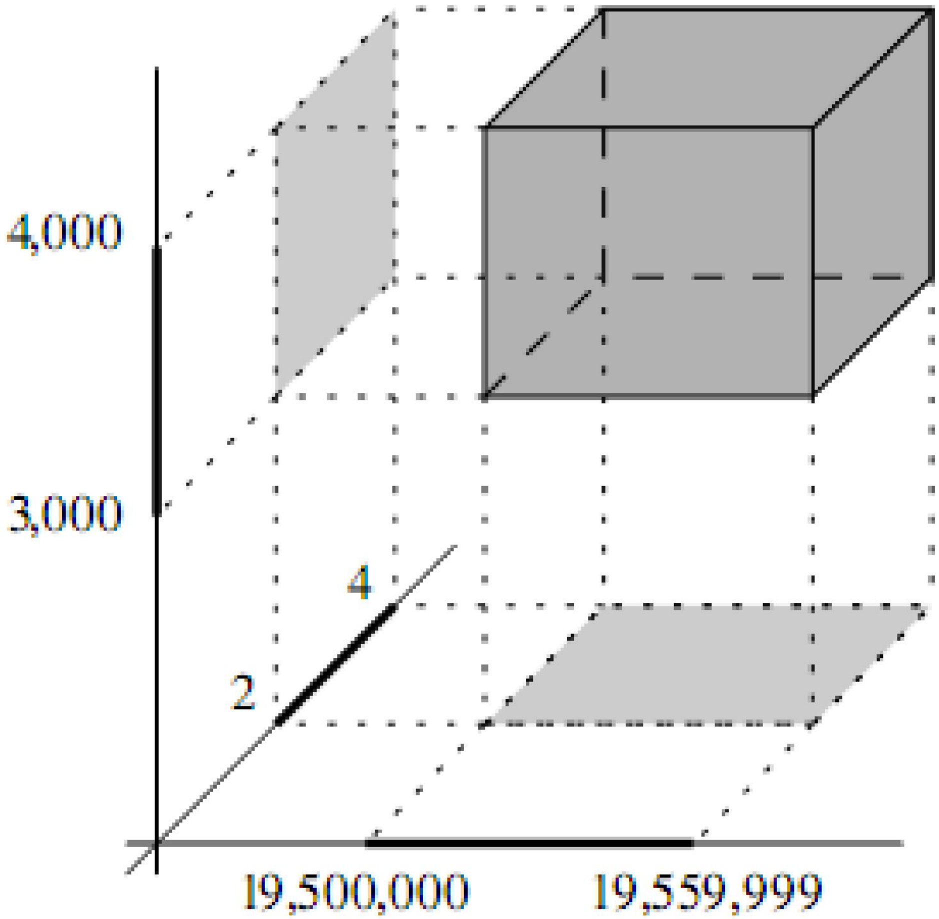
4

Other Query

- If we have information about the number of children, to ask queries like“ report all employees born between 1950 and 1955 who earn between $3,000 and $4,000 a month and have between two and four children “
- Represent each employee by a point in 3-dimensional space. The first coordinate represents the date of birth, the second coordinate the salary, the third coordinate the number of children

5

Interpreting of query



Such a query is called a rectangular range query or an orthogonal range query

6

1-Dimensional Range Searching

- $P := \{p_1, p_2, \ldots, p_n\}$ is set of point on the rial line
- Data structure : a balanced binary search tree T
- Leaves store a point
- internal nodes store splitting values to guide the search
- Denote the splitting value stored at a node v by

  x_v

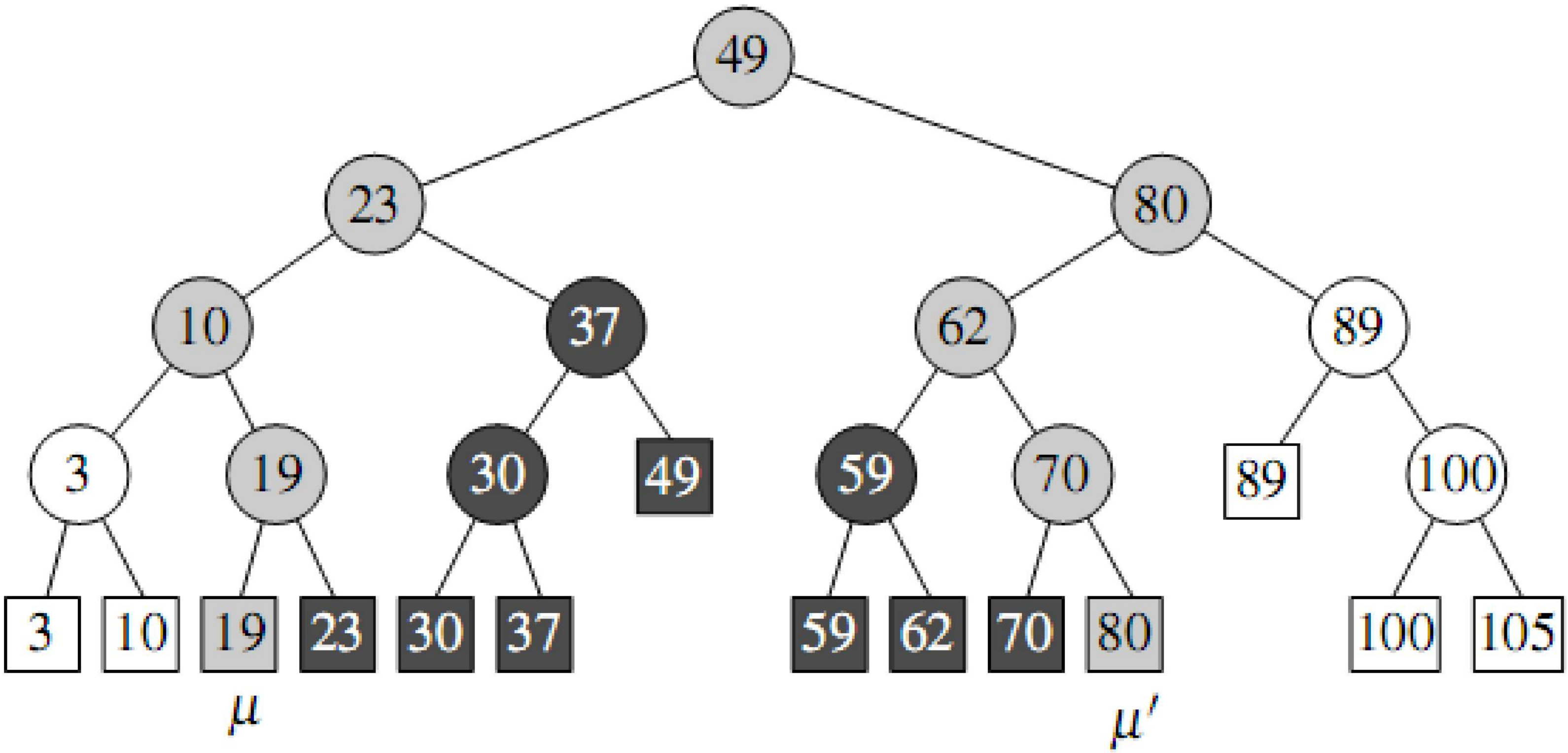
1-Dimensional Range Searching

To report the points in a query range $[x, x']$:

- Search with x and x' in T
- μ and μ' be the two leaves where the search end , respectively
- point in the interval $[x, x']$ are the onse stored in the leaves in between μ and μ' plus, possibly the point store at μ and μ'

A 1-dimensional range query in a binary search tree

search with the interval $[18,77]$



9

1-Dimensional Range Searching

How can we find the leaves in between μ and μ'?

- They are the leaves of certain subtrees in between the search paths to μ and μ'
- The subtrees that we select are rooted at nodes v in between the two search paths whose parents are on the search path

To find these nodes :

- search for the node v_{split} where the paths to x and x' split

$lc(v)$ and $rc(v)$ denote the left and right child of

10

Find split node

$FINDSPLITNODE(\mathrm{T}, x, x')$

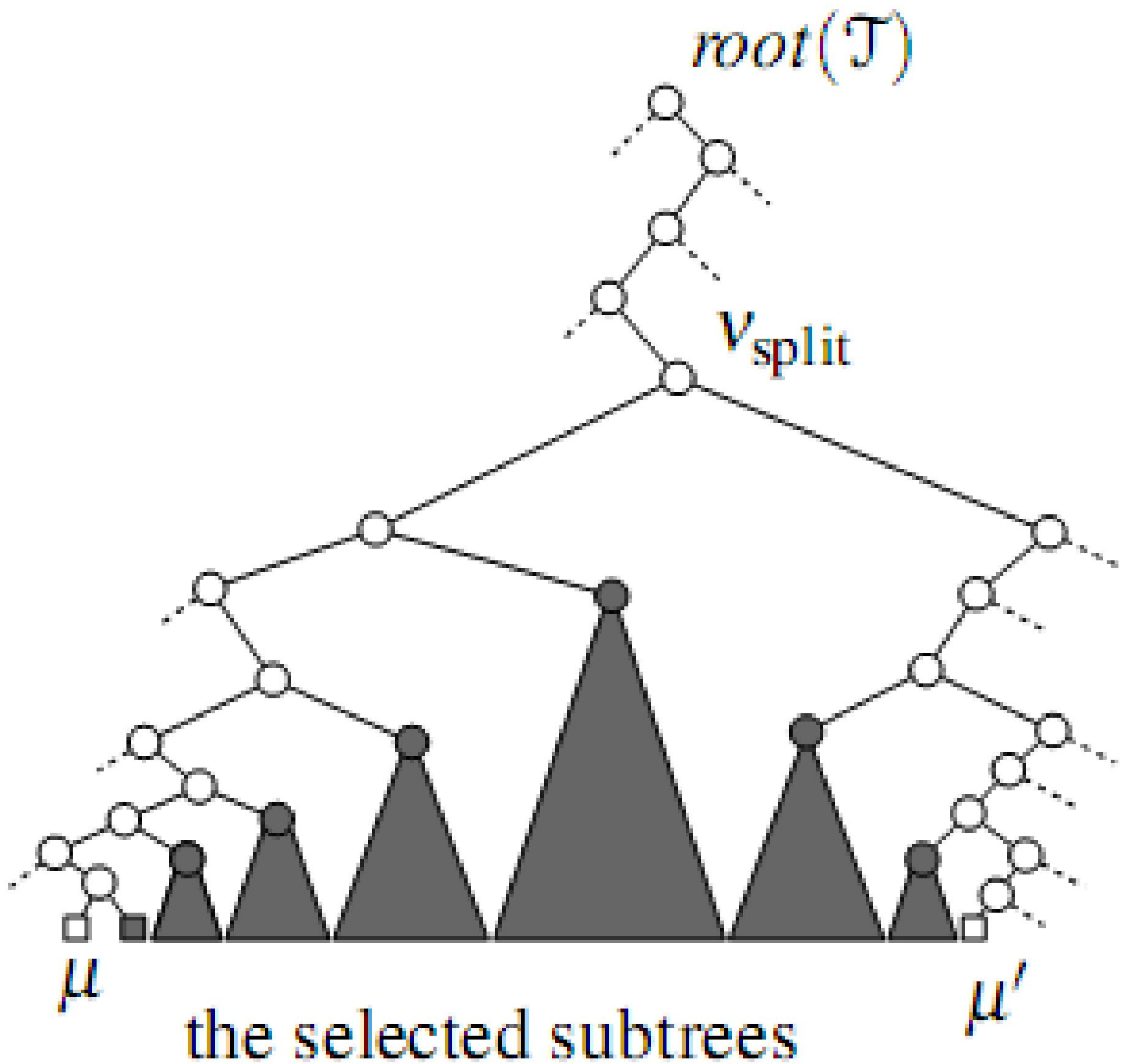
$Input.A$ tree and two values x and x' with $x \le x'$.

Output. The node v where the paths to x and x' split,

or the leaf where both paths end.

1. $v \leftarrow root(T)$
2. While v is not a leaf and $(x' \le x_v \; or \; x \ge x_v)$
3. do if $x' \le x_v$
4. then $v \leftarrow lc(v)$
5. else $v \leftarrow rc(v)$
6. return v

The selected subtree



12

The selected subtree

$Algorithm$ 1DRANGEQUERY(T,[x,x'])

Input. A binary search tree T and a range [x, x'].

Output. All points stored in T that lie in the range.

1. $v_{split} \leftarrow FINDSPLITNODE(T, x, x')$
2. if v_{split} is a leaf
3. then Check if the point stored at v_{split} must be reported.
4. else (*Follow the path to x and report the points in subtrees right of the path.*)
5. $v \leftarrow lc(v_{split})$
6. while v is not a leaf
7. do if $x \leq x_v$
8. then REPORTSUBTREE($rc(v)$)
9. $v \leftarrow lc(v)$
10. else $v \leftarrow rc(v)$
11. *Check* if the point stored at the leaf v must be reported.
12. Similarly, follow the path to x', report the points in subtree left of the path, and check if the poin stored at the leaf where the path ends must be reported.

Prove Correctness Of The Algorithm

lemma 5.1

- Algorithm 1DRANGEQUERY reports exactly those point that lie in the query range

proof

- Any reported point p lies in the query range
- Any point p in the range is reported

Summarizes the results for 1-dimensional range searching

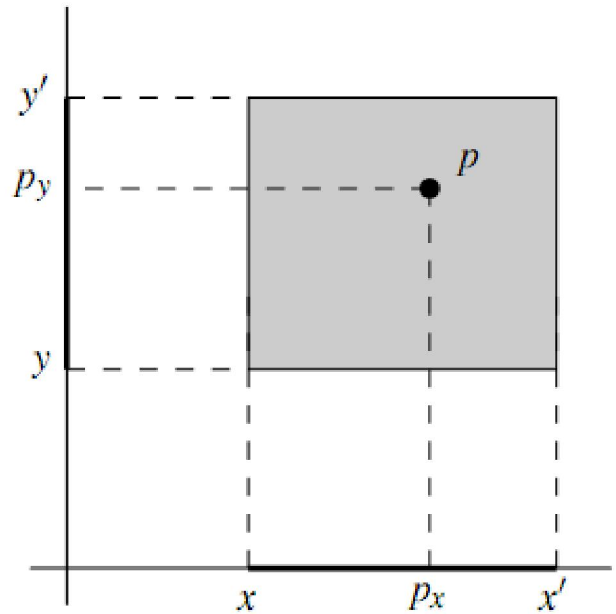
Theorem 5.2

- Let p be a set of n point in 1-dimensional space. The set P can be stored in a balanced binery search tree , which uses $O(n)$ storage and has $O(n \log n)$ construction time, such that the pionts a query range can be reported in time $O(k + \log n)$, where k is the number of reported points.

Kd-Trees

2-dimensional rectangular range searching problem

- In this section we assume that no two points in p have the same x-coordinate, and no two points have the same y-coordinate
- A 2-dimensional rectangular range query on p asks for the point from p lying inside a query rectangle $[x:x']\times[y:y']$



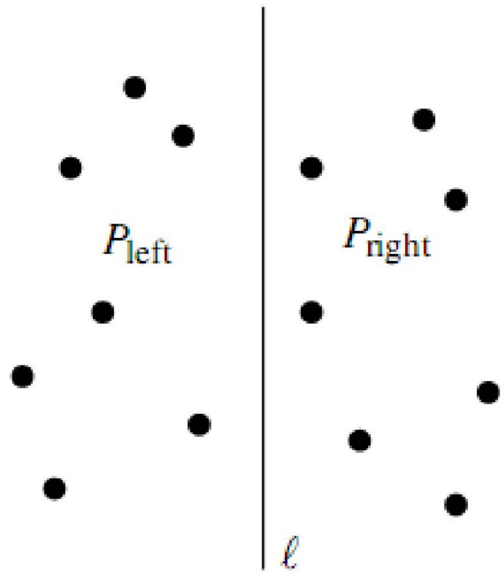
- A point $p:=(p_x, p_y)$ lise inside this rectangle if and only if

$$p_x \in [x, x'] \quad \text{and} \quad p_y \in [y, y'].$$

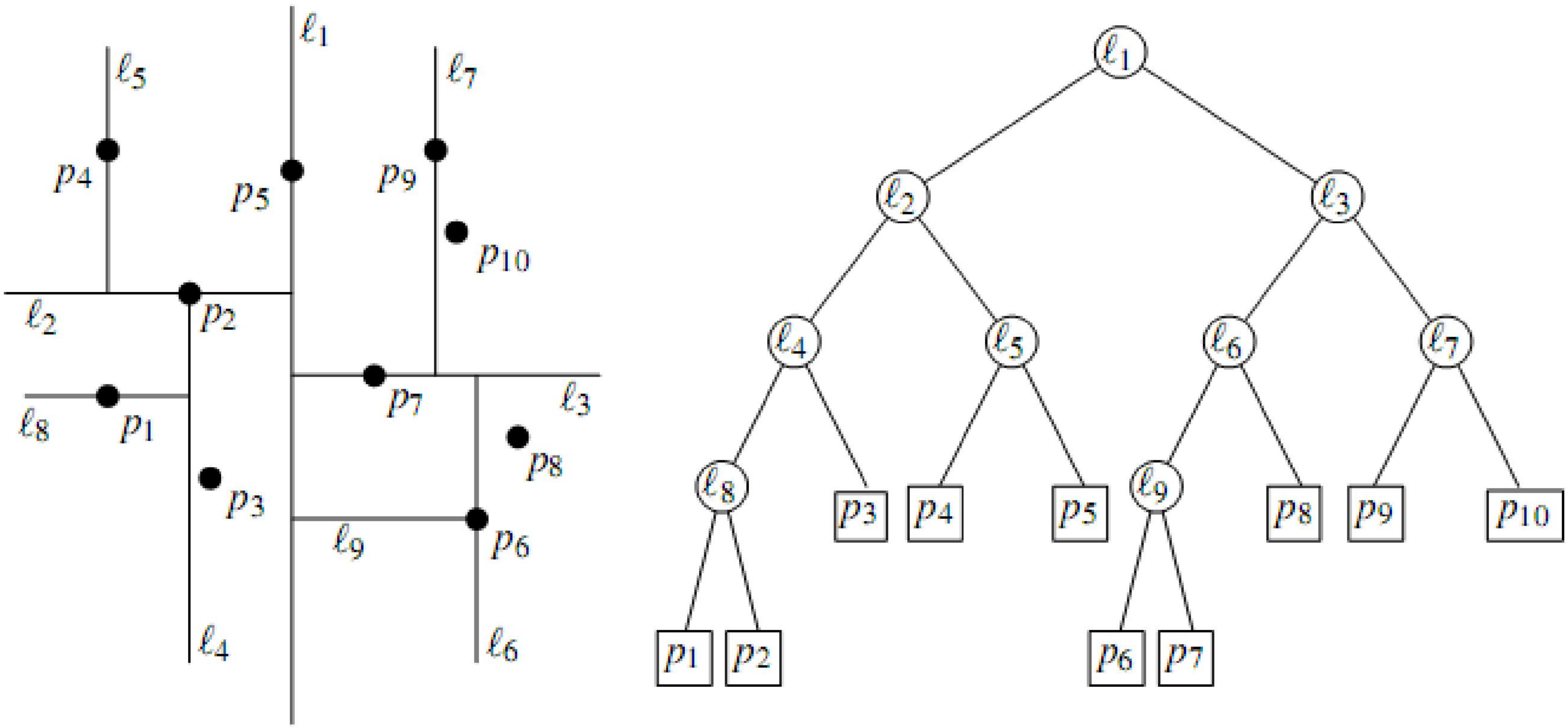
Kd-Trees

How can we generalize structure for 1-dimensional ?

- First split on x-coordinate, next on y-coordinate, then again on x-coordinate, and so on
- The splitting line is stored at the root
- P_{left} the subset of point to the left or on the slitting line, is stored in the left subtree, and P_{right} , the subset to the right of it,is stored in the right subtree



A kd-tree



- A tree like this is called a kd -tree

18

Construct a kd-tree

$A\lg orithmBUIL\ \ DKDTREE\ \ (P, depth\,)$

$Input\,.A$ set of points P and the current depth depth.

Output. The root of a kd - tree storing P.

1. if P contains only one point

2. then return a leaf storing this point

3. else if depth is even

4. the n Split P into two subsets with a vertical line l through t he median x - coordinate of the points in P. Let p_1 be the set of points to th e left of l or on l and p_2 be the set of points to the right of l.

5. else Split P into two subsets with a horizontal line l through t he median y - coordinate of the points in P. Let p_1 be the set of points to th e left of l or on l and p_2 be the set of points to the right of l.

6. $v_{left} \leftarrow BUILDKDTRE\ \ E(p_1, depth+1)$

7. $v_{right} \leftarrow BUILDKDTRE\ \ E(p_2, depth+1)$

8. Create a node v storing l, make v_{left} the left child of v, nd make v_{right} the right child of v

9. return v

Sorted list x-coordinate

1	4	2	3	5	7	9	10	6	8

Sorted list y-coordinate

6	3	8	1	7	2	10	5	9	4

1	4	2	3	5	7	9	10	6	8
3	1	2	5	4	6	8	7	10	9

1 2 | 4 3 5 | 7 6 | 9 10 8

3 1 2 | 5 4 | 6 8 7 | 10 9

Storage And Building Time A kd-tree

The buiding time T(n) :

$$T(n) = \begin{cases} O(1), & \text{if} \quad n = 1 \\ O(n) + 2T(\left\lceil \frac{n}{2} \right\rceil), & \text{if} \quad n > 1 \end{cases}$$

which solves to O (nlogn)

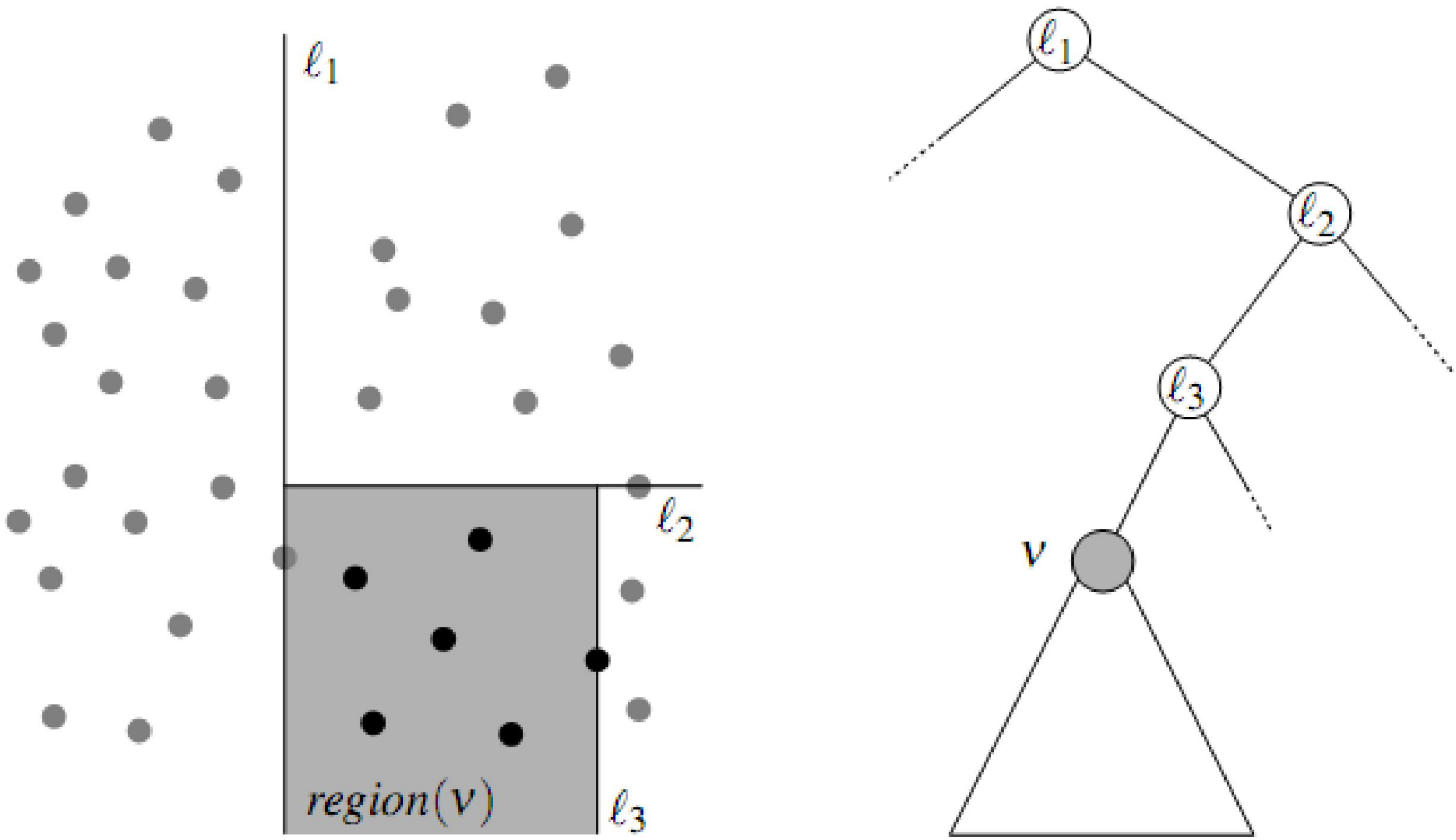
lemma 5.3

- A kd-tree for a set of points uses O (n) storage and can be constructed in O (nlogn) time

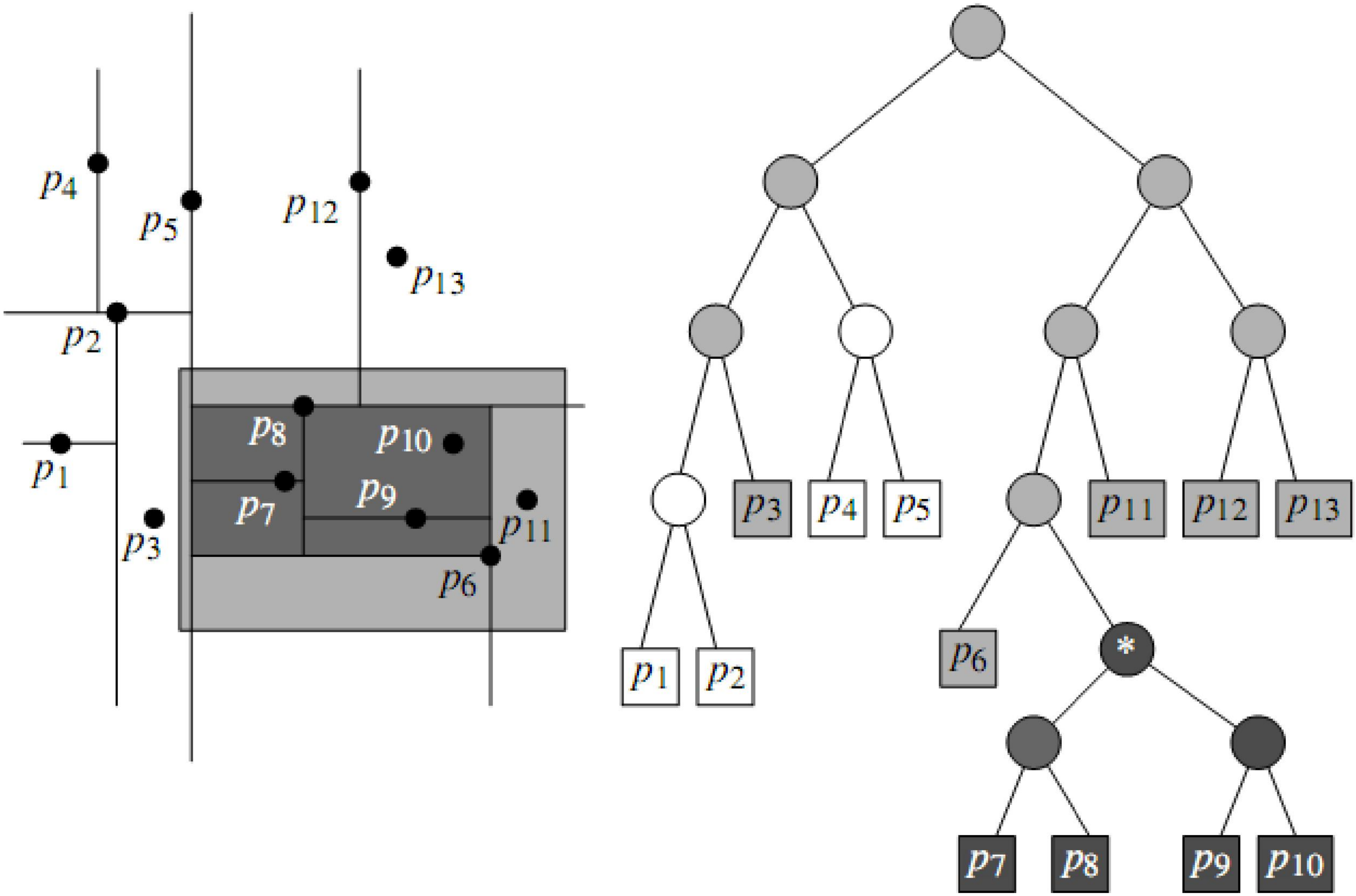
Corresponding Between Nodes In A KD-tree And Regions In The Plane

- The region corresponding to a node v is a rectangle
- It is bounded by spliting lines stored at ancestors of v which denote by $region(v)$
- We have to search the subtree rooted at v only if the query rectangle intersects $region(v)$

Corresponding Between Nodes In A KD-tree And Regions In The Plane



A query on a kd-tree



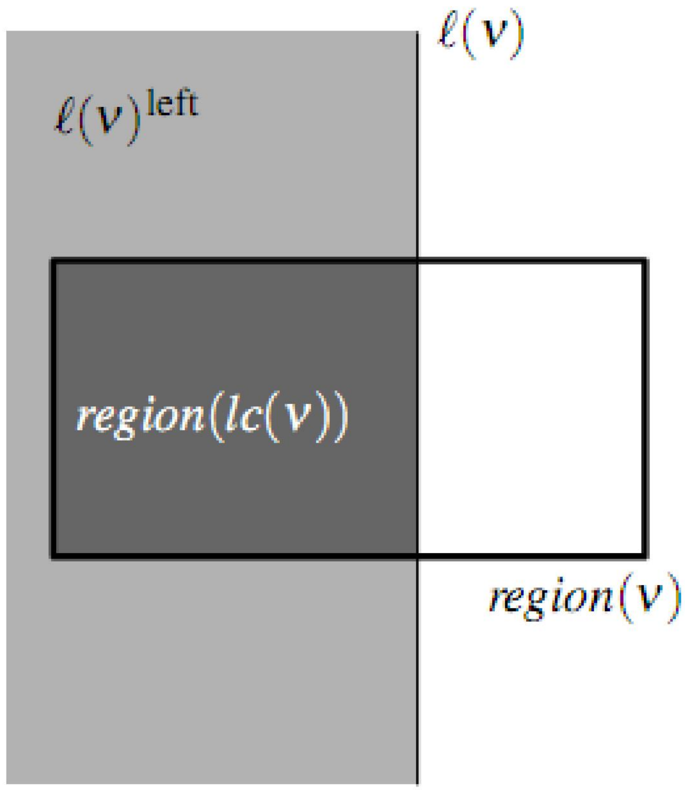
Algorithm SEARCHKDTREE(v, R)

Input.The root of (a subtree of) a kd-tree, and a range R.

Output. All point at leaves below v that lie in the range.

if v is a leaf
    then Report the point stored at v if it lies in R.
    else if $region(lc(v))$ is fully contained in R
        then REPORTSUBTREE($lc(v)$)
        else if $region(lc(v))$ intersects R
            then SEARCHKDTREE($lc(v)$, R)
        if $region(rc(v))$ is fully contained in R
        then REPORTSUBTREE($rc(v)$)
        else if $region(rc(v))$ intersects R
            then SEARCHKDTREE($rc(v)$, R)

$$region\, lc(v) = region\,(v) \cap l(v)^{left}$$



26