Orthogonal Range Searching

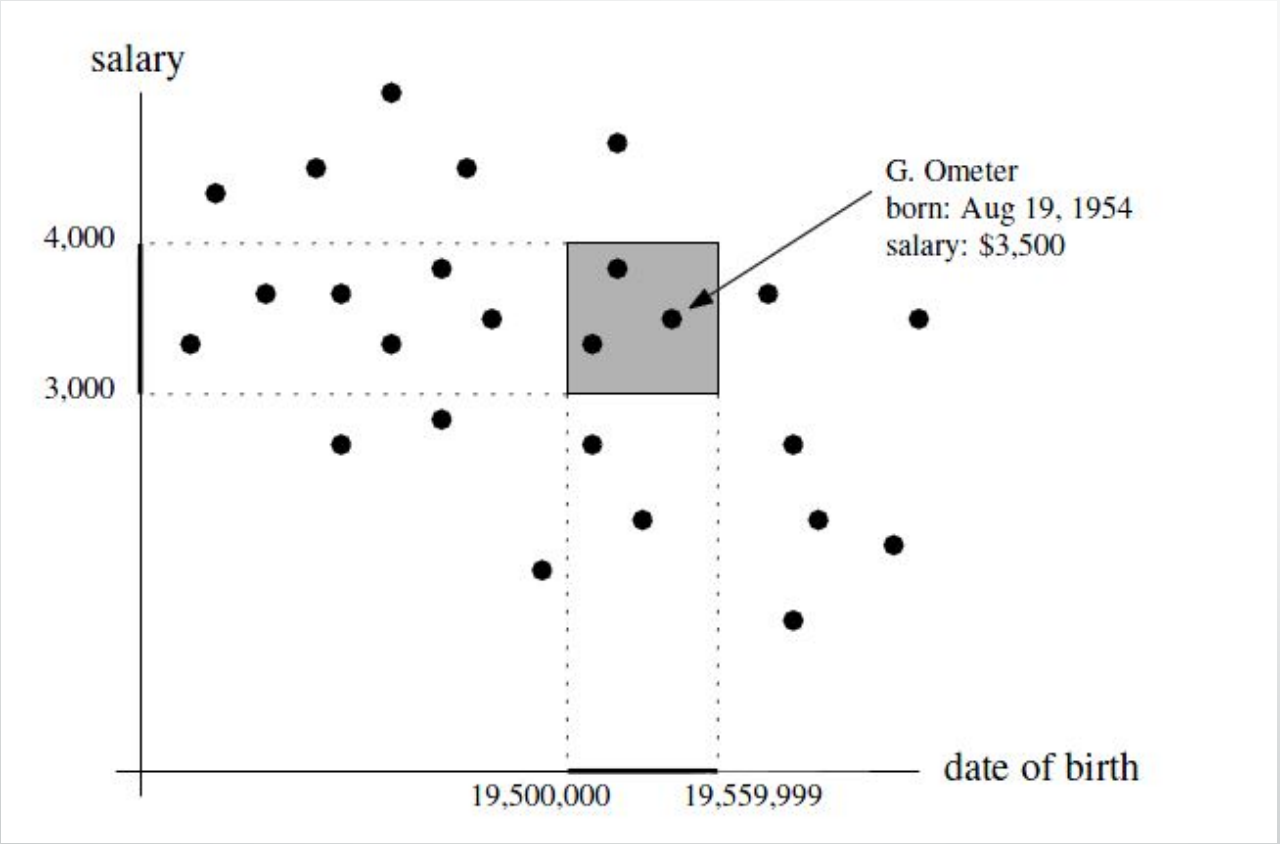
Querying a Database

جستجو محدوده متعامد

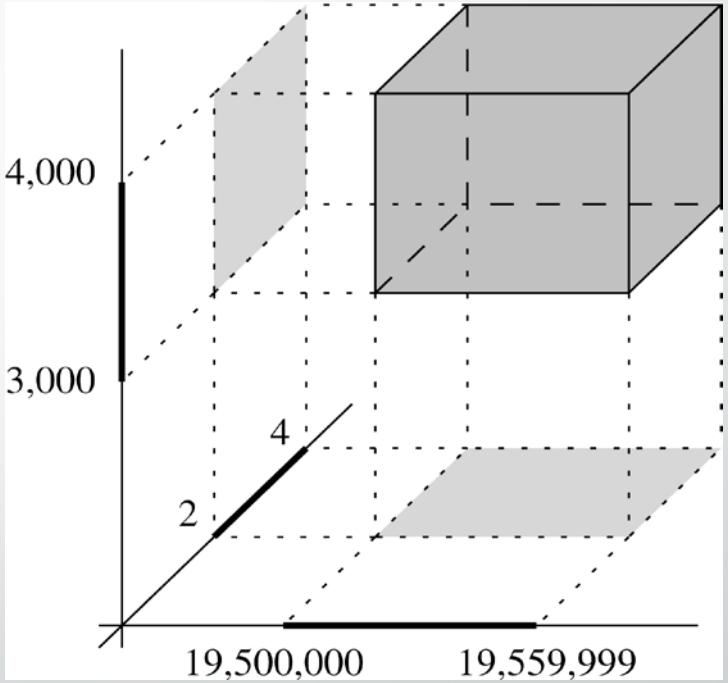
- Many types of questions - queries - about data in a database can be interpreted geometrically
- To this end we transform records in a database into points in a multi-dimensional space, and we transform the queries about the records into queries on this set of points

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 10,000 year + 100 month + day, the second coordinate is the monthly salary
- With the point we store the other information


salary
4,000
3,000
G. Ometer
born: Aug 19, 1954
salary: $3,500
19,500,000
19,559,999
date of birth

- 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



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

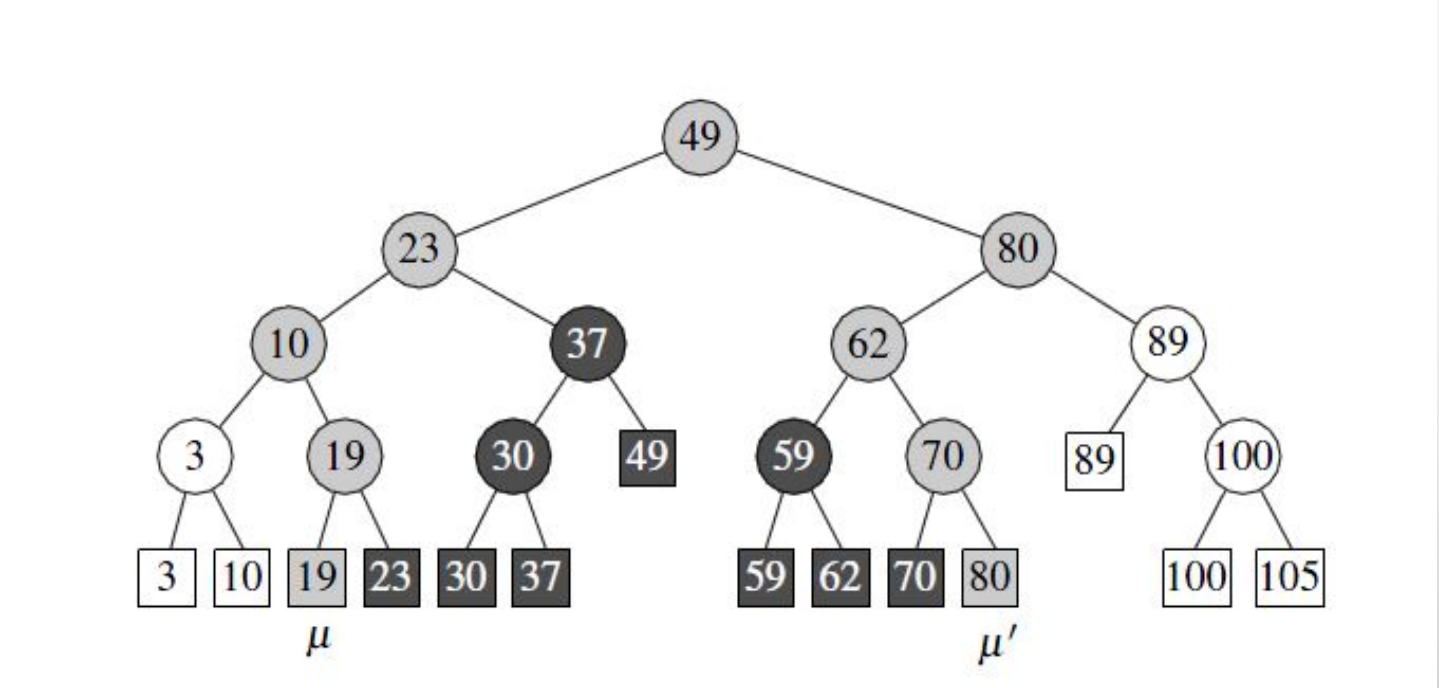
Dimensional Range Searching

- P := { p1, p2 ,..., pn} 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 Xv

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'

- search with the interval [18 : 77]



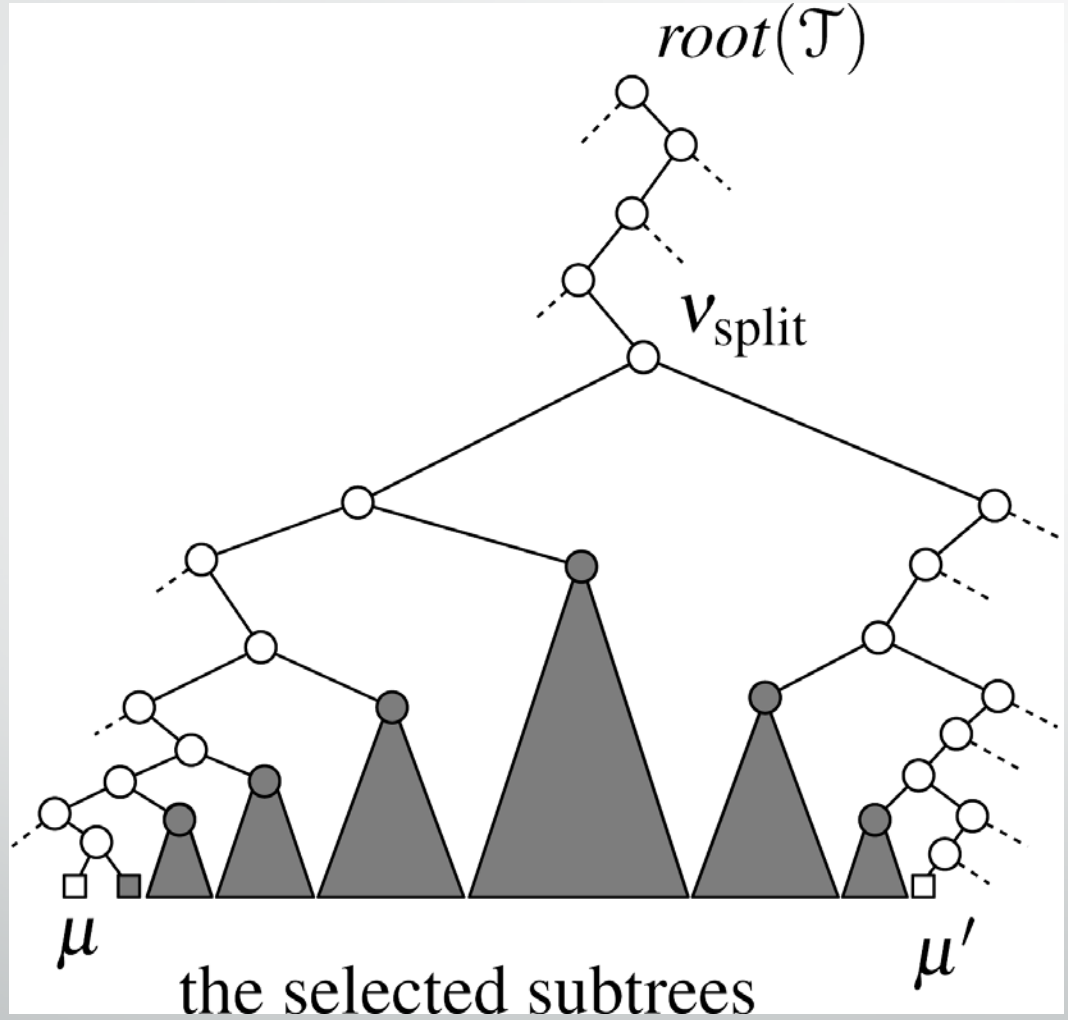
- How can we find the leaves in between μ and μ' ?
- They are the leaves of certain subtrees in between the search paths to μ and μ '
- The subtrees that we select are rooted at nodes v in between the two search paths whose parents are on the search path
- To find these nodes :
- search for the node vsplit where the paths to x and x' split
- lc(v) and rc(v) denote the left and right child of v

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

Input. A tree $\mathcal{T}$ and two values x and x' with $x \leqslant x'$.

Output. The node v where the paths to x and x' split, or the leaf where both paths end.

1. $v \leftarrow root(\mathcal{T})$
2. **while** v is not a leaf **and** ($x' \leqslant x_v$ **or** $x > x_v$)
3. **do if** $x' \leqslant x_v$
4. **then** $v \leftarrow lc(v)$
5. **else** $v \leftarrow rc(v)$
6. **return** v


root(T)
v_split
μ
μ′
the selected subtrees

Algorithm 1DRANGEQUERY($\mathcal{T}, [x : x']$)
Input. A binary search tree $\mathcal{T}$ and a range $[x : x']$.
Output. All points stored in $\mathcal{T}$ that lie in the range.

```
1.  v_split ← 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 ← lc(v_split)
6.          while v is not a leaf
7.             do if x ≤ x_v
8.                   then REPORTSUBTREE(rc(v))
9.                        v ← lc(v)
10.                  else  v ← 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 subtrees left of the path, and
            check if the point stored at the leaf where the path ends must be reported.
```

لم ۵.۱:

الگوریتم 1DRANGEQUERY به طور دقیق نقاطی را گزارش می کند که صرفاً درون بازه ی query قرار داشته باشد.

اثبات:

هر نقطه ی گزارش شده p ، در بازه query قرار دارد.

هر نقطه ای در بازه ، حتماً گزارش شده است.

قضیه ۵.۲

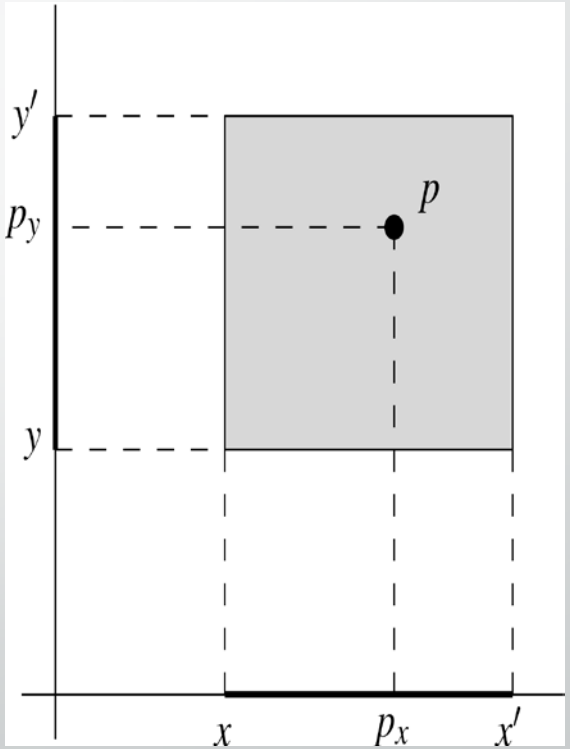
فرض کنید که P مجموعه ای از n نقطه در فضای یک بعدی باشد. مجموعه p می تواند در یک درخت جستجوی دودویی متوازن ذخیره شود که از O(n) حافظه استفاده می کند و زمان ساخت آن O(nlogn) است. بازه query برای چنین نقاطی را می توان در زمان O(k+logn) گزارش داد که k برابر است با تعداد نقاط گزارش شده.

مسئله جستجو در بازه مستطیلی ۲ بعدی:

در این بخش ما فرض می کنیم که هیچ دو نقطه ای در p ، مختصه x یا y برابر با هم نداشته باشند.

یک query با بازه مستطیلی دو بعدی بر روی p، از ما راجع به نقاطی از p می پرسد که درون یک مستطیل query [x;x'][y;y'] قرار داشته باشند.

یک نقطه $p=(p_x , p_y)$ فقط و فقط زمانی درون مستطیل query قرار دارد که : $p_x \in [x : x']$ $p_y \in [y : y']$

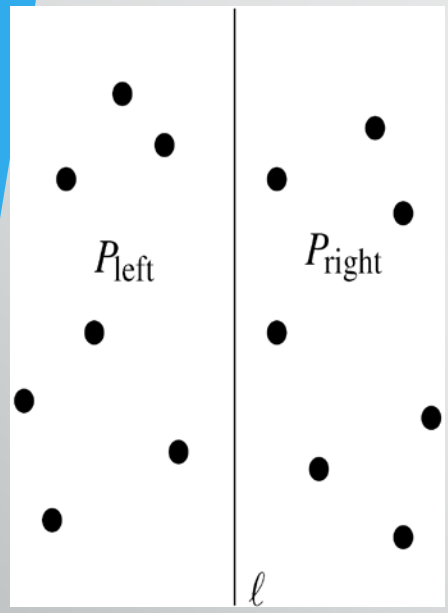


چگونه می توانیم ساختار حالت یک بعدی را تعمیم دهیم؟

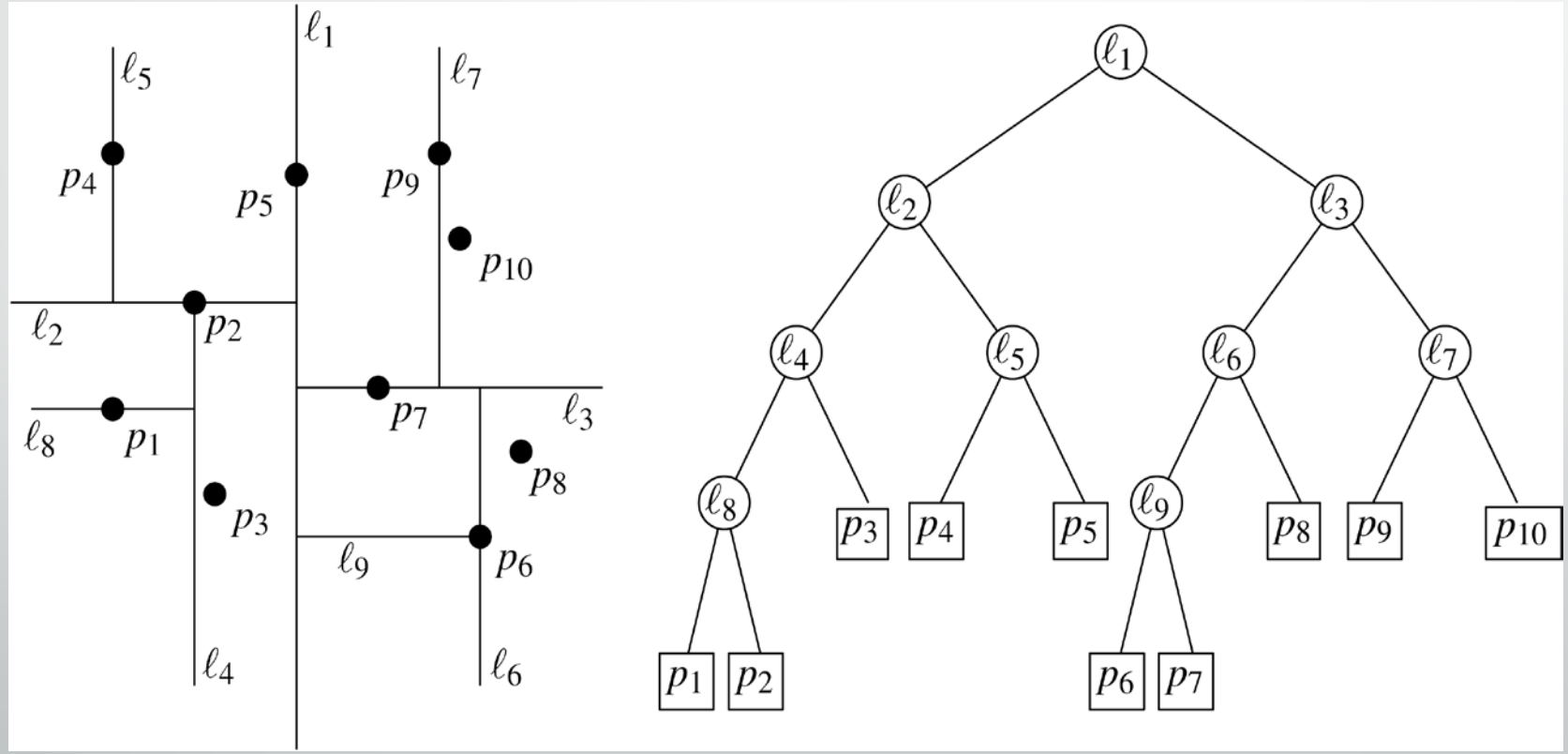
ابتدا روی مختصه x جداسازی انجام می دهیم، سپس بر روی مختصه y و دوباره بر روی x و به همین ترتیب ادامه می دهیم.

خط جدا کننده در ریشه ذخیره می شود.

Pleft زیرمجموعه ای از نقاط است که در سمت چپ یا روی خط جداکننده قرار داشته باشند و در زیردرخت چپ ذخیره می شوند و Pright زیر مجموعه ای از نقاط است که در سمت راست خط جداکننده قرار داشته باشند و در زیر درخت راست ذخیره می شوند.



به این چنین درختی یک درخت kd گفته می شود

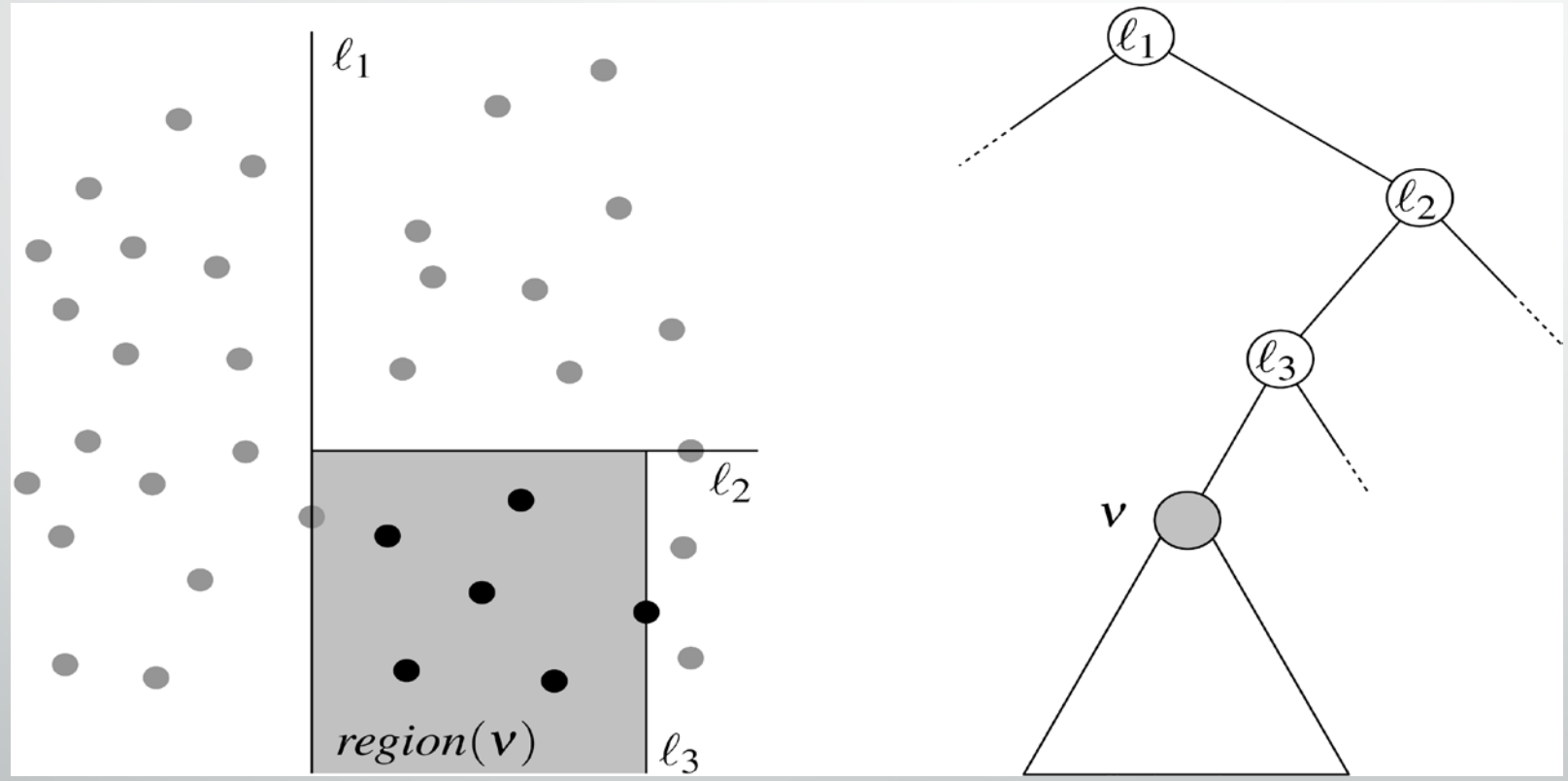


Algorithm BUILDKDTREE($P, depth$)
Input. A set of points P and the current depth *depth*.
Output. The root of a kd-tree storing P.

1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** *depth* is even
4. **then** Split P into two subsets with a vertical line ℓ through the median x-coordinate of the points in P. Let P_1 be the set of points to the left of ℓ or on ℓ, and let 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 the median y-coordinate of the points in P. Let P_1 be the set of points below ℓ or on ℓ, and let P_2 be the set of points above ℓ.
6. $v_{\text{left}} \leftarrow$ BUILDKDTREE($P_1, depth + 1$)
7. $v_{\text{right}} \leftarrow$ BUILDKDTREE($P_2, depth + 1$)
8. Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
9. **return** v

- زمان ساخت T(n) :
- $$T(n) = \begin{cases} O(1), & \text{if } n = 1 \\ O(n)+2T(n/2), & \text{if } n > 1 \end{cases}$$
- مرتبه زمانی رابطه O(nlogn) است
- lemma 5.3
- یک درخت kd برای مجموعه ای از n نقطه، از O(n) حافظه استفاده می کند و در زمان O(nlogn) ساخته می شود.

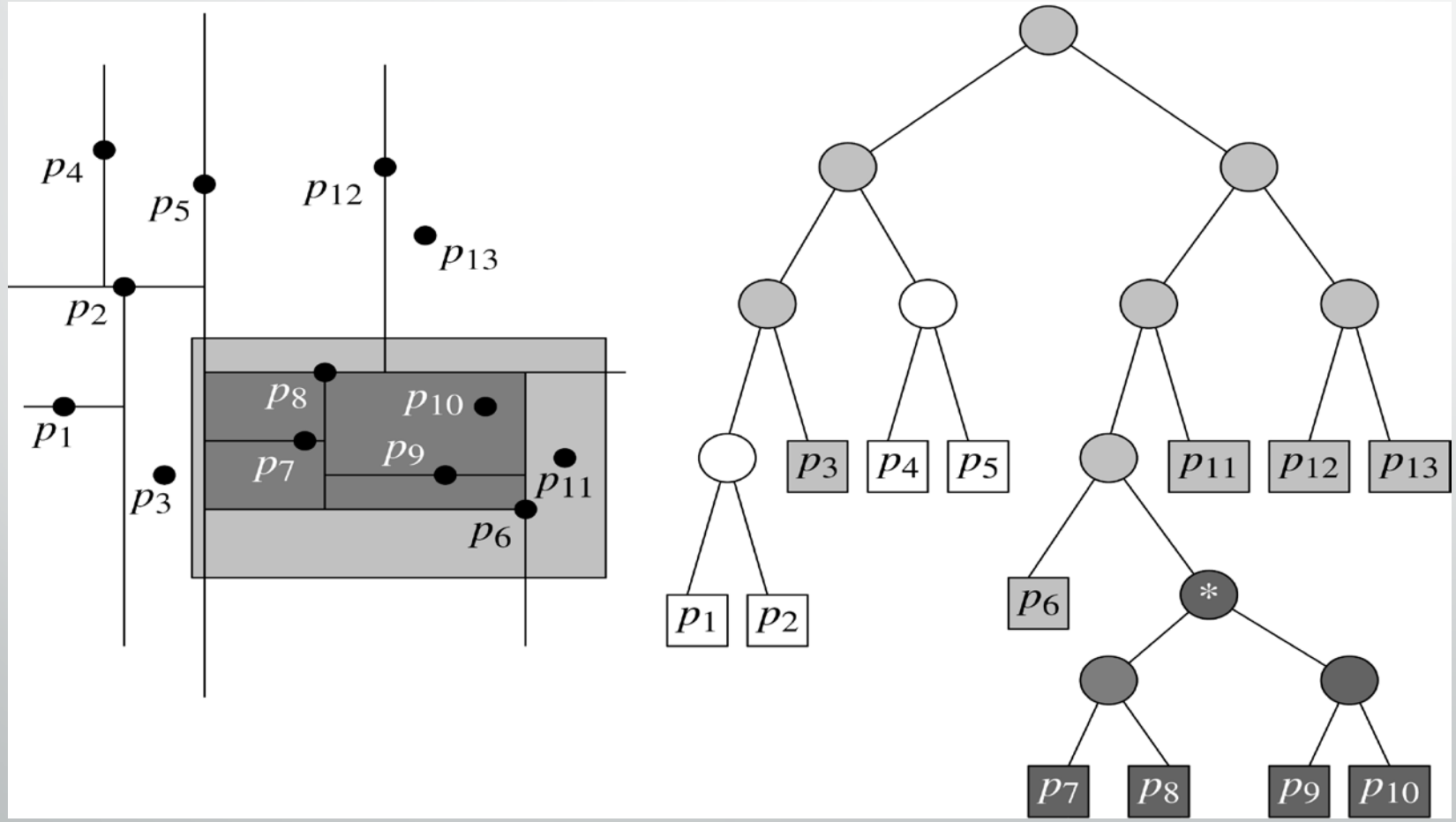
تناظر بین گره ها در درخت kdو مناطق درون صفحه



منطقه متناظر با یک گره v ، یک مستطیل می باشد که مرزهای آن خطوط جداکننده ی ذخیره شده در نیاکان v هستند و این منطقه با region (v) نشان داده می شود.

- ما باید فقط زمانی زیردرخت v را جستجو کنیم که مستطیل query با region (v) تلاقی کند.

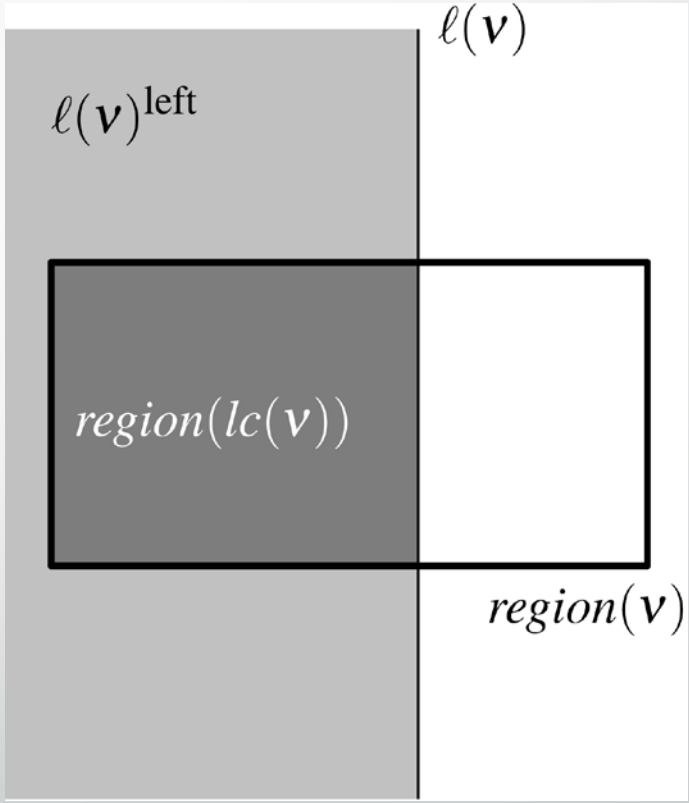
A query on a kd-tree



Algorithm SEARCHKDTREE(v, R)
Input. The root of (a subtree of) a kd-tree, and a range R.
Output. All points at leaves below v that lie in the range.

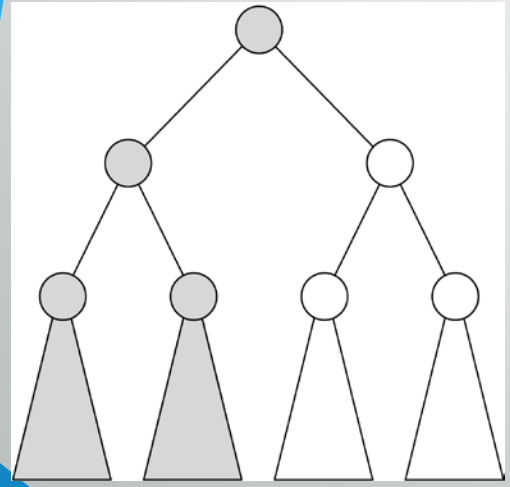
```
1.  if v is a leaf
2.     then Report the point stored at v if it lies in R.
3.     else  if region(lc(v)) is fully contained in R
4.              then REPORTSUBTREE(lc(v))
5.              else  if region(lc(v)) intersects R
6.                       then SEARCHKDTREE(lc(v), R)
7.           if region(rc(v)) is fully contained in R
8.              then REPORTSUBTREE(rc(v))
9.              else  if region(rc(v)) intersects R
10.                      then SEARCHKDTREE(rc(v), R)
```

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



لم ۵.۴ – یک query را می توان با یک مستطیل موازی المحور در یک درخت kd که n نقطه را ذخیره می کند در زمان $O(\sqrt{n}+k)$ انجام داد که k برابر است با تعداد نقاط گزارش شده.

- کل زمان مورد نیاز برای عبور از زیردرخت در مرحله ۴ و ۸ برابر با $O(k)$ است.
- $Q(n)$ برابر است با تعداد مناطق متقاطع در یک درخت kd که n نقطه را ذخیره می کند و ریشه اش نیز یک خط جداکننده عمودی را شامل می شود.



- $$Q(n) = \begin{cases} O(1) \text{ if } n = 1 \\ 2 + 2Q(n/4) \text{ if } n > 1 \end{cases}$$
- $$Q(n) = O(\sqrt{n})$$
- هر خط عمودی در یک درخت kd با $O(\sqrt{n})$ منطقه تلاقی می کند.

خلاصه سازی عملکرد درخت kd:

- قضیه ۵.۵
- یک درخت kd برای مجموعه p شامل n نقطه در صفحه از O(n) حافظه استفاده می کند و در زمان O(nlogn) ساخته می شود. یک query با بازه مستطیلی روی درخت kd ، $O(\sqrt{n} + k)$ زمان می گیرد که k تعداد نقاط گزارش شده است.

درخت های kd برای فضای ۳ بعدی یا بالاتر:

- حافظه : O(n)
- زمان ساخت: O(nlogn)
- زمان query : $O(n^{1-\frac{1}{d}} + k)$