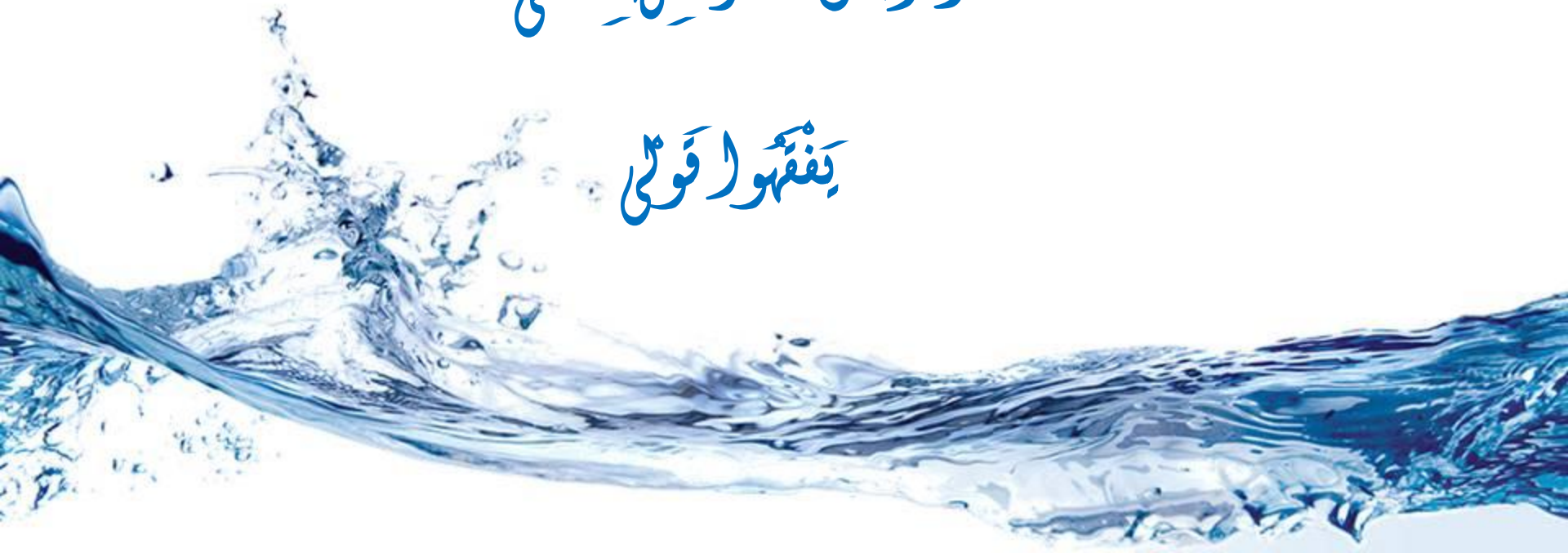
بسم الله الرحمن الرحيم

رَبِّ اشْرَحْ لِي صَدْرِي

وَيَسِّرْ لِي أَمْرِي

وَاحْلُلْ عُقْدَةً مِنْ لِسَانِي

يَفْقَهُوا قَوْلِي



Orthogonal range searching

سیده مرضیه عبدالهی

پاییز ۹۳

ارتباط پایگاه‌داده و هندسه

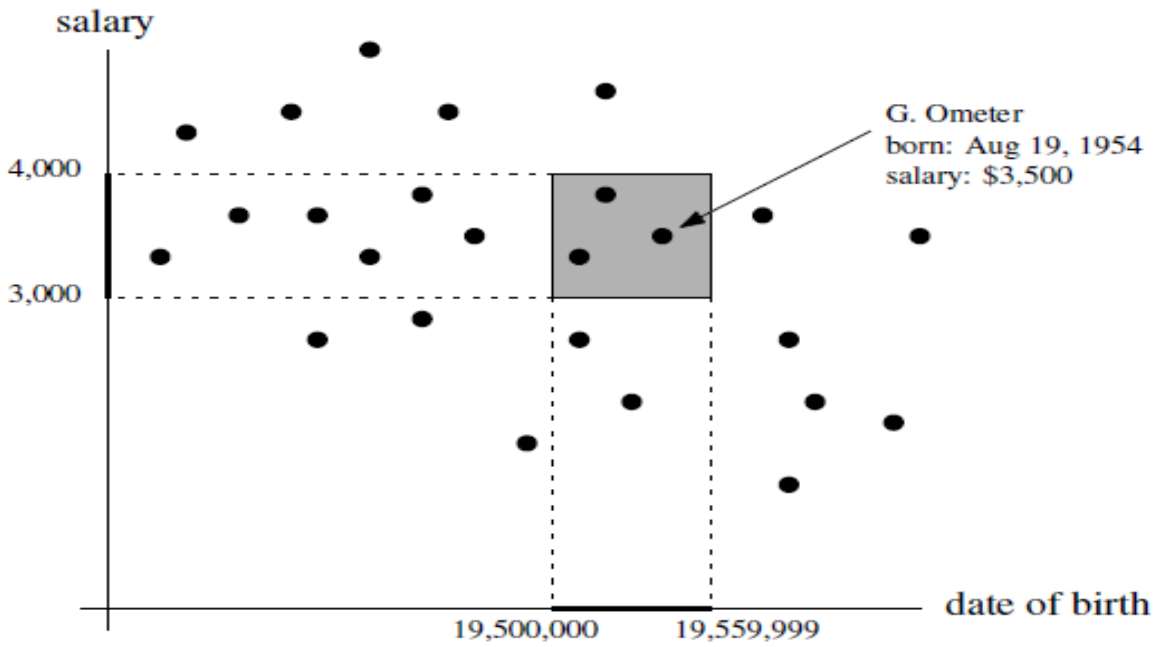
- تبدیل رکوردهای پایگاه‌داده به نقاط درفضای چند بعدی
- تبدیل کوئری روی رکوردها به کوئری روی مجموعه نقاط

پایگاه‌داده‌ی اداره‌ای را در نظر بگیرید.

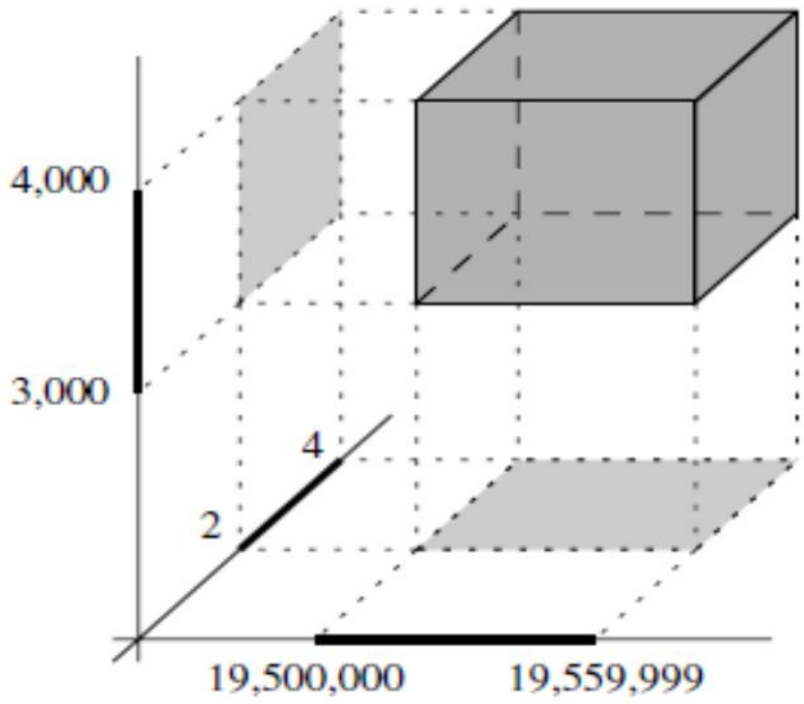
در این پایگاه‌داده اطلاعاتی نظیر نام، آدرس، تاریخ تولد، حقوق و ... کارمندان ذخیره می‌شود.

ما می‌خواهیم همه‌ی کارمندانی را که تاریخ تولد آن‌ها بین سال ۱۹۵۰ تا ۱۹۵۵است و حقوق ماهیانه‌ی آن‌ها بین ۳۰۰۰$ تا ۴۰۰۰$ است را گزارش کنیم.

نمایش هر کارمند به صورت یک نقطه در فضا



نمایش در فضای سه‌بعدی



دامنه‌ی جستجوی یک‌بعدی:

یک مجموعه از نقاط در فضای یک‌بعدی داده شده‌است.

می‌خواهیم نقاط موجود در بازه‌ی [x:x’] را گزارش کنیم.

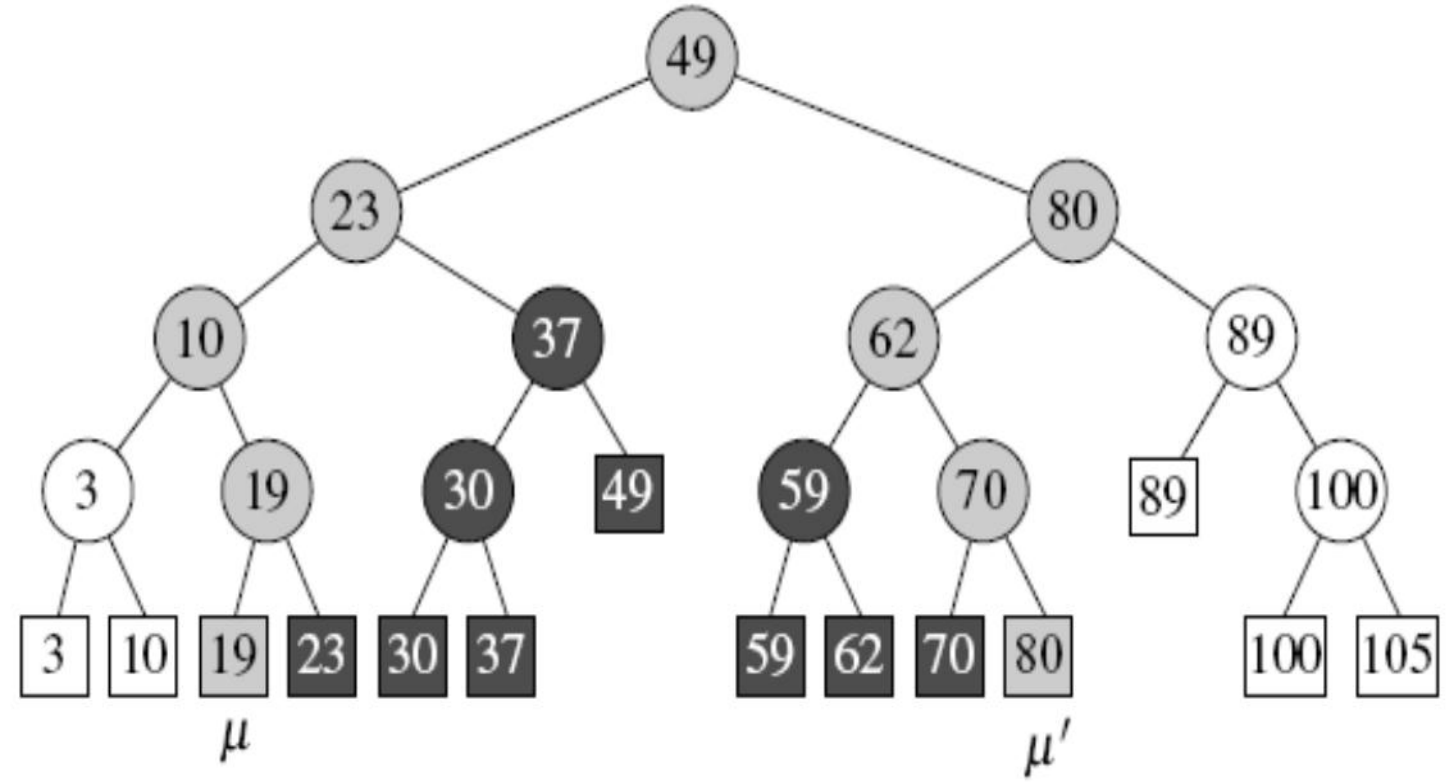
فرض کنید $P:=\{p_1,p_2,\ldots,p_n\}$ یک مجموعه از نقاط روی خط حقیقی داده شده‌است. ما می‌توانیم مسئله‌ی جستجو در دامنه‌ی یک‌بعدی را با استفاده از درخت جستجوی دودویی متوازن T حل کنیم.

- برگ‌های درخت T نقاط را ذخیره می کنند.
- گره‌های داخلی مقادیر جداکننده را برای هدایت جستجو ذخیره می‌کنند.
- مقدار جداکننده‌ی ذخیره‌شده در گره v را با x_v نشان می‌دهیم.
- زیردرخت چپ گره v شامل همه‌ی نقاط کوچکتر یا مساوی x_v است.
- زیردرخت راست شامل همه‌ی گره‌های اکیدا بزرگتر از x_v است.

گزارش نقاط در بازه [x:x’]

- جستجوی x و x’ در درخت T
- فرض کنید μ و μ' دو برگی باشند که جستجو در آنجا خاتمه یافت.
- *نقاط در بازه‌ی* [x:x’] در برگ‌های بین μ و μ' و شاید در خود *این* دو گره ذخیره شده‌اند.

جستجوی بازه [18:77]



چگونه برگ‌های بین μ و μ' را پیدا کنیم؟

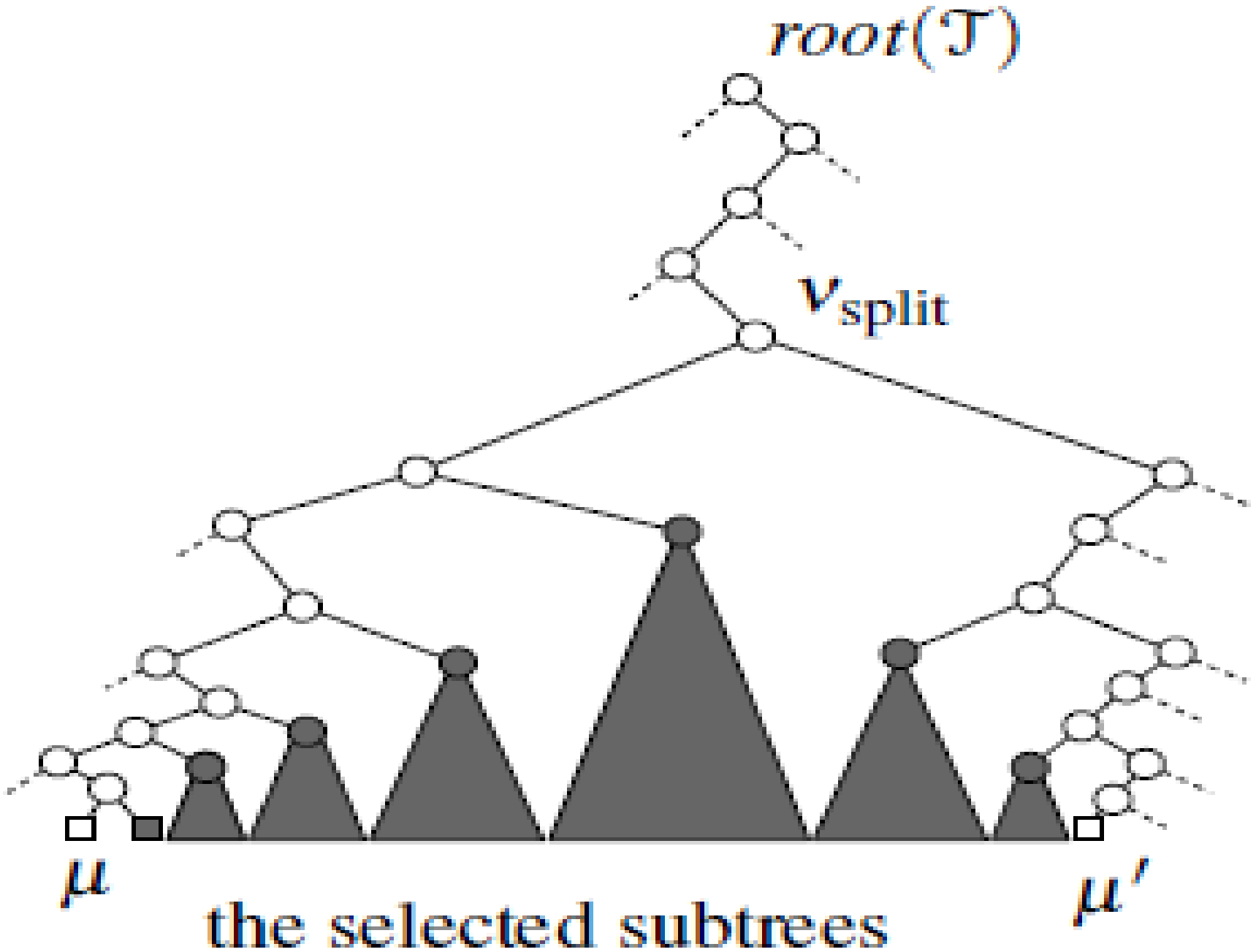
- V_{split} : گره ای که مسیر جستجوی x و x’ از هم جدا می شود.
- lc(v) : فرزند چپ v
- rc(v) : فرزند راست v

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


$root(\mathcal{T})$
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 1DRANGE QUERY دقیقا نقاطی که در بازه قرار دارد را گزارش می‌کند.

اثبات:

۱. هر نقطه‌ای که گزارش می‌شود در بازه است.

۲. هر نقطه‌ای که در بازه قرار دارد حتما گزارش می‌شود.

ابتدا قسمت اول را ثابت می‌کنیم .

فرض می‌کنیم p نقطه‌ای باشد که گزارش شده‌است،

می‌خواهیم بگوییم p حتما در بازه قرار دارد، ۳ حالت رخ می‌دهد:

۱. p در برگی ذخیره شده باشد که در انتهای مسیر جستجوی x یا x’ به آن برخورد کردیم در این صورت به طور صریح بررسی می‌شود که آیا در بازه قرار دارد یا خیر.

در غیر این‌صورت p در یک فراخوانی ReportSubTree گزارش می شود.فرض کنید این فراخوانی زمانی باشد که در جستجوی xهستیم و فرض کنید که v نودی باشد که p در فراخوانی ReportSubTree(rc(v)) اعلام شده باشد.

چون v و rc(v) در زیر درخت چپ v_{split} هستند پس $p<x_{vsplit}$ و چون برای پیدا کردن x’ به زیر درخت راست v_{split} می رویم داریم:

$x_{vsplit}<x'$

$P<x_{vsplit}$ $\xrightarrow{*}$ $P<x'$

برای پیدا کردن x به سمت چپ v رفتیم و p هم در زیر درخت راست v قرار دارد پس $p>x$ **

از * و ** نتیجه می گیریم:

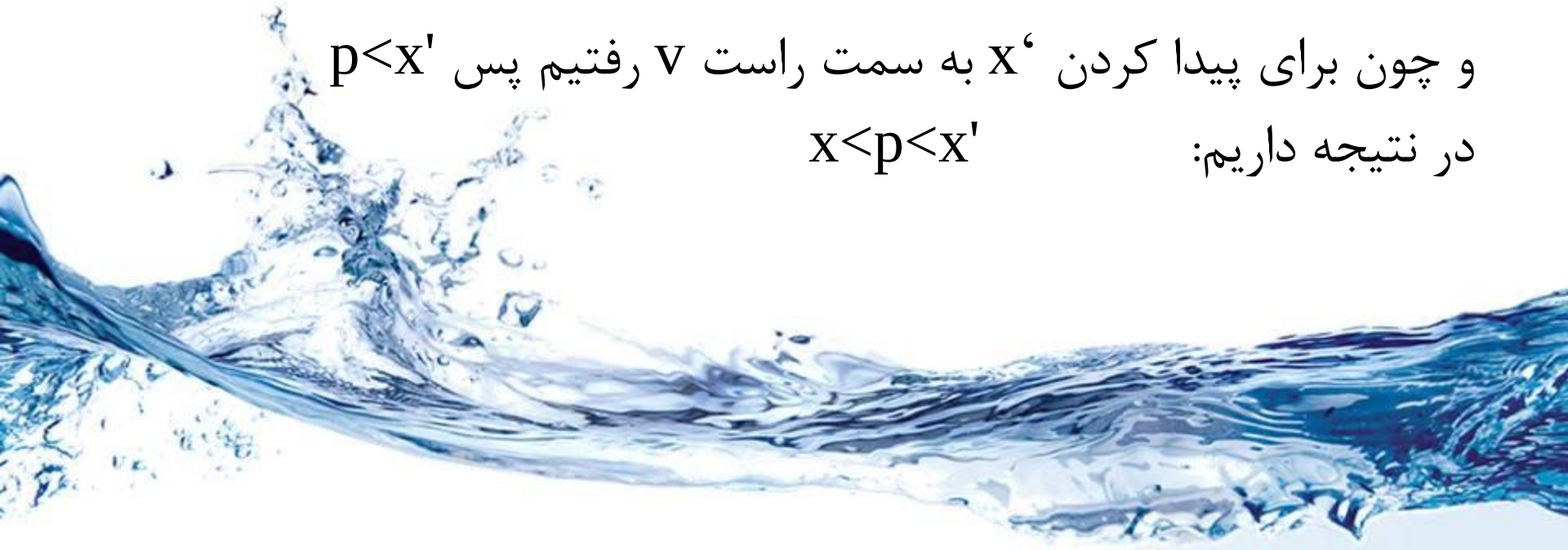
$$x<p<x'$$

۳.اگر در جستجوی ‘x باشیم و فرض کنیم v نودی باشد که p در فراخوانی ReportSubTree(lc(v)) اعلام شده باشد. چون v و lc(v) در زیردرخت راست v_{split} قرار دارند پس $x_{vsplit}<p$ و چون برای پیدا کردن x به سمت چپ v_{split} می‌رویم پس داریم :

$$\left.\begin{array}{l} x<x_{vsplit} \\ x_{vsplit}<p \end{array}\right| \longrightarrow x<p$$

و چون برای پیدا کردن ‘x به سمت راست v رفتیم پس $p<x'$

در نتیجه داریم: $x<p<x'$



حال باید قسمت دوم را اثبات کنیم:

اگر نقطه‌ای در بازه هست حتما گزارش می‌شود.

فرض خلف : p گزارش نشده است. (p را نقطه‌ای درون بازه [x:x’] در نظر می‌گیریم.)

فرض می‌کنیم p در زیردرخت سمت چپ v_{split} قرار دارد. از p به سمت ریشه حرکت می‌کنیم به آخرین راسی که توسط الگوریتم دیده شده‌است می‌رسیم و آن را v می‌نامیم.

۴ حالت رخ می‌دهد:

۱) اگر p در زیردرخت راست v باشد و ما برای پیدا کردن x به سمت چپ حرکت کنیم پس حتما p توسط reportsubtree(rc(v)) گزارش شده-است.(تناقض)

۲) اگر p در زیردرخت چپ v باشد و ما به سمت چپ حرکت کنیم با اینکه فرض کردیم v آخرین راس ملاقات شده است در تناقض است .

۳) اگر p در زیردرخت سمت راست v باشد و ما برای یافتن x به سمت راست حرکت کنیم با اینکه v آخرین راس ملاقات شده است در تناقض است.

۴) اگر p در زیردرخت چپ v باشد و ما برای یافتن x به سمت راست حرکت کنیم در اینصورت p اصلا در بازه نیست .

پس فرض خلف باطل است.

قضیه ۵.۲:

فرض کنید p مجموعه‌ای از n نقطه در فضای یک بعدی باشد.

مجموعه‌ی p می‌تواند در یک درخت جستجوی دودویی متوازن با فضای ذخیره سازی O(n) وزمان ساخت O(n logn) ذخیره شود.به طوریکه نقاط در دامنه‌ی کوئری در زمان O(k + log n) می‌توانند گزارش شوند که K تعداد نقاط گزارش شده است.

❖چون درخت ما درخت جستجوی دودویی متوازن است تعداد نودهای داخلی از تعداد برگ‌ها کمتر است پس حافظه‌ی مصرفی آن از مرتبه O(n) و زمان ساخت آن O(n logn) می‌باشد.

❖ زمان لازم برای الگوریتم کوئری به ۲ عامل بستگی دارد.

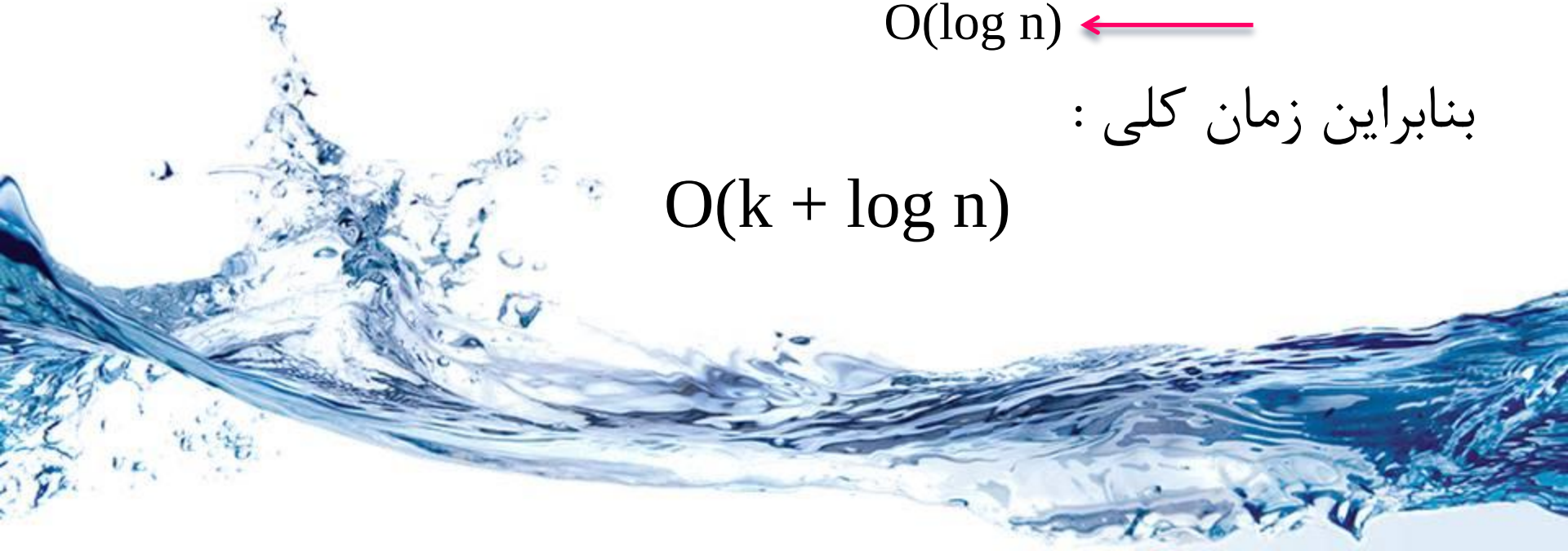
۱.زیر روال reportsubtree ← O(k)

۲. پیمایش گره‌هایی که در مسیر جستجوی x ,x’ به آنها برخورد کردیم.

← O(log n)

بنابراین زمان کلی :

$$O(k + \log n)$$



Kd-tree

- P مجموعه‌ای از n نقطه در صفحه
- هیچ ۲ نقطه‌ای در p مختصات x و y یکسان ندارند.
- یک کوئری با بازه مستطیلی دوبعدی بر روی p از ما راجع به نقاطی می پرسد که درون یک مستطیل کوئری $[x:x'] \times [y:y']$ قرار داشته باشند.
- یک نقطه $p:=(p_x,p_y)$ در مستطیل قرار دارد اگر و فقط اگر

$$P_x \in [x: x']$$

و

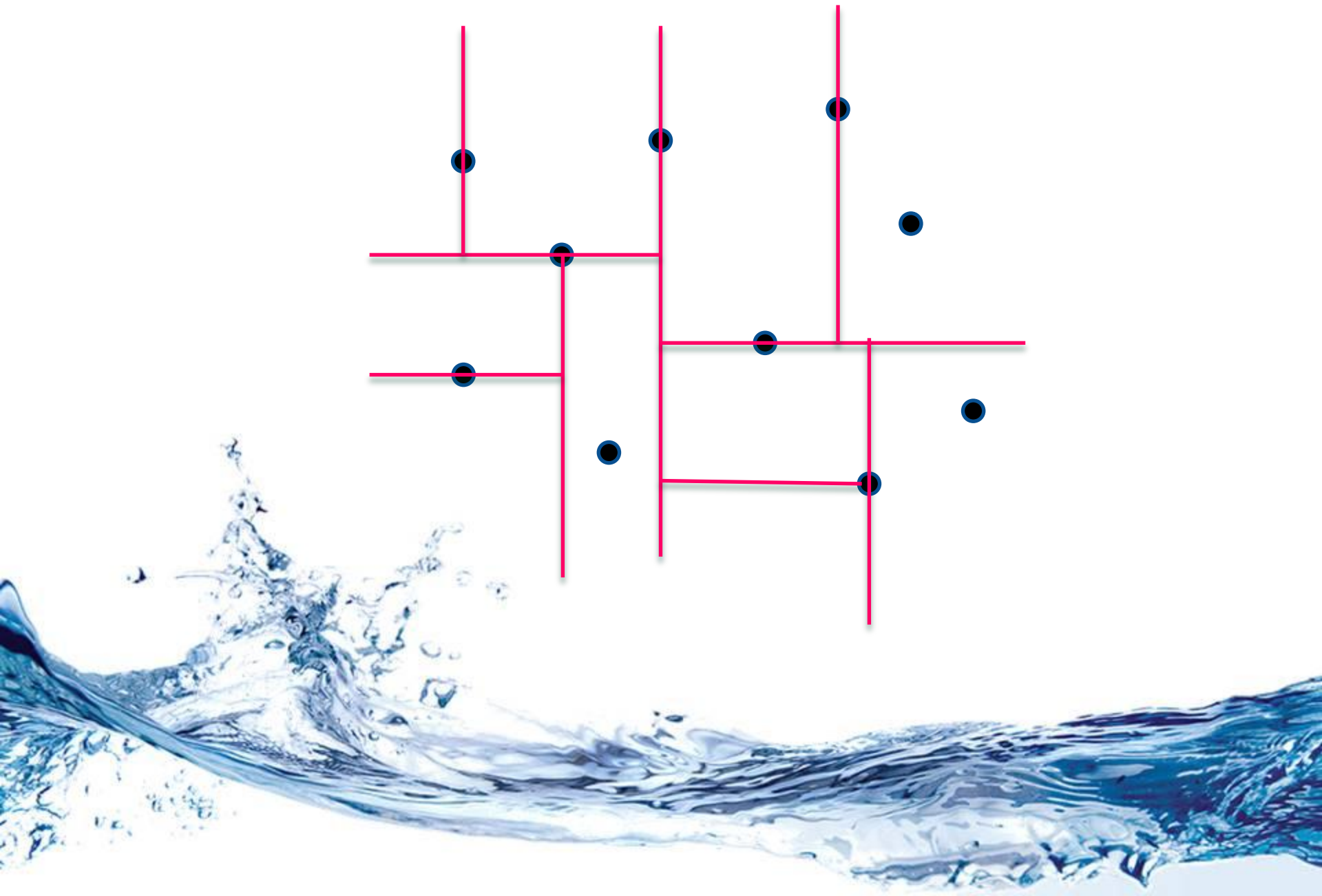
$$P_y \in [y:y']$$

چگونه ساختار حالت یک‌بعدی را به دوبعدی تعمیم دهیم؟

- مجموعه P را با استفاده از خط عمودی به دو زیر مجموعه تقریبا مساوی P_{right}، P_{left} تقسیم می‌کنیم. (خط عمودی به واسطه ی میانه، براساس مختصه x نقاط موجود در P رسم می‌شود.)
- مجموعه P_{left}(به ترتیب P_{right}) را با استفاده از پاره خط افقی به دو قسمت تقسیم می‌کنیم .(خط افقی به وسیله‌ی میانه، براساس مختصه y نقاط مجموعه‌ی P_{left}(بترتیب P_{right})رسم می شود.)

ما این پروسه‌ی بازگشتی را با استفاده از خط عمودی و افقی به طور متناوب تا زمانی ادامه می‌دهیم که هر نود تنها یک نقطه داشته باشد.

در حالت کلی هرگاه عمق زوج باشد با استفاده از خط عمودی و هرگاه عمق فرد باشد با استفاده از خط افقی جداسازی صورت می‌گیرد.

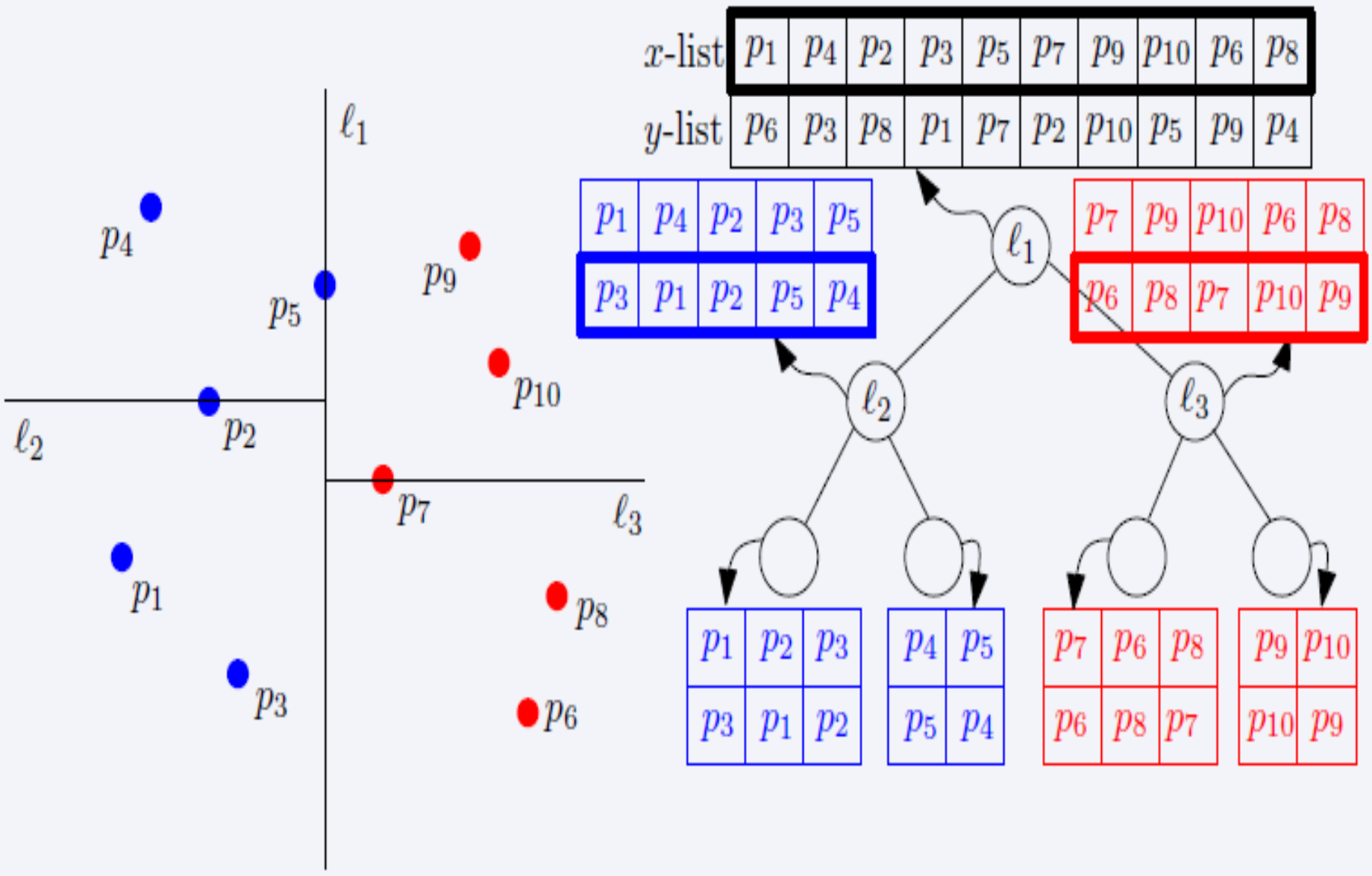


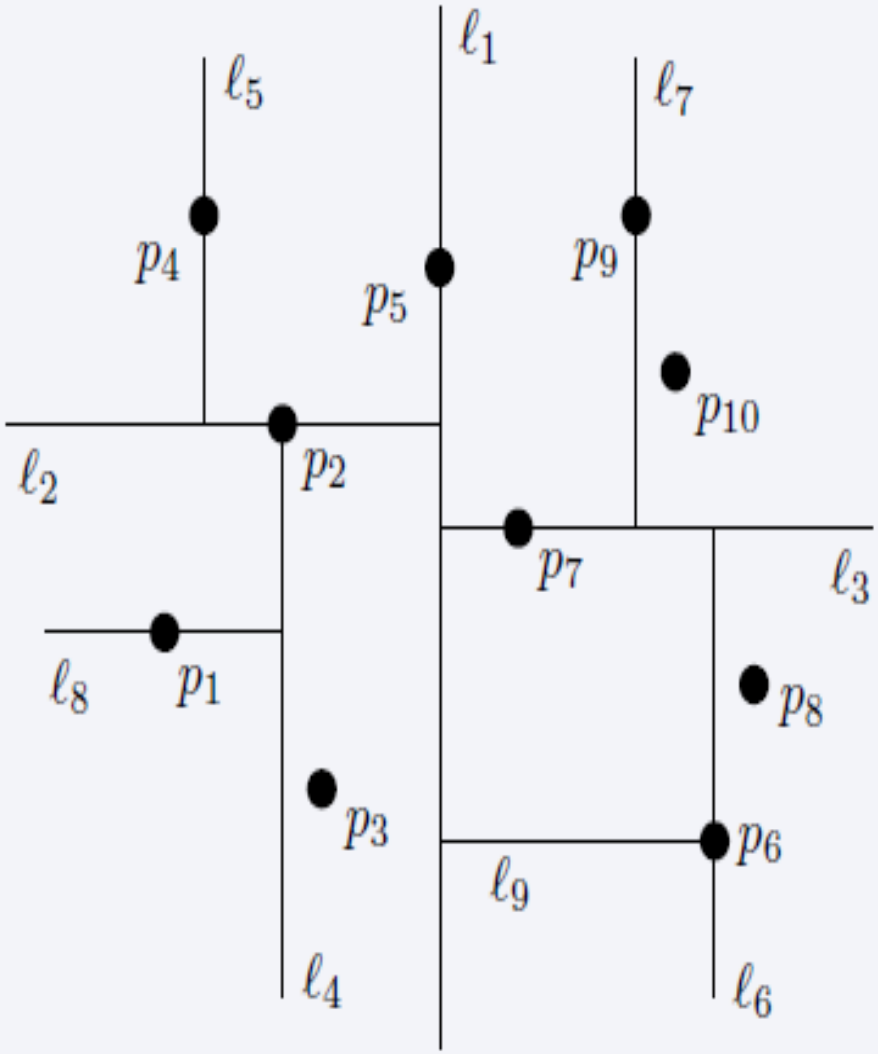
Algorithm BUILDKDTREE(P, *depth*)

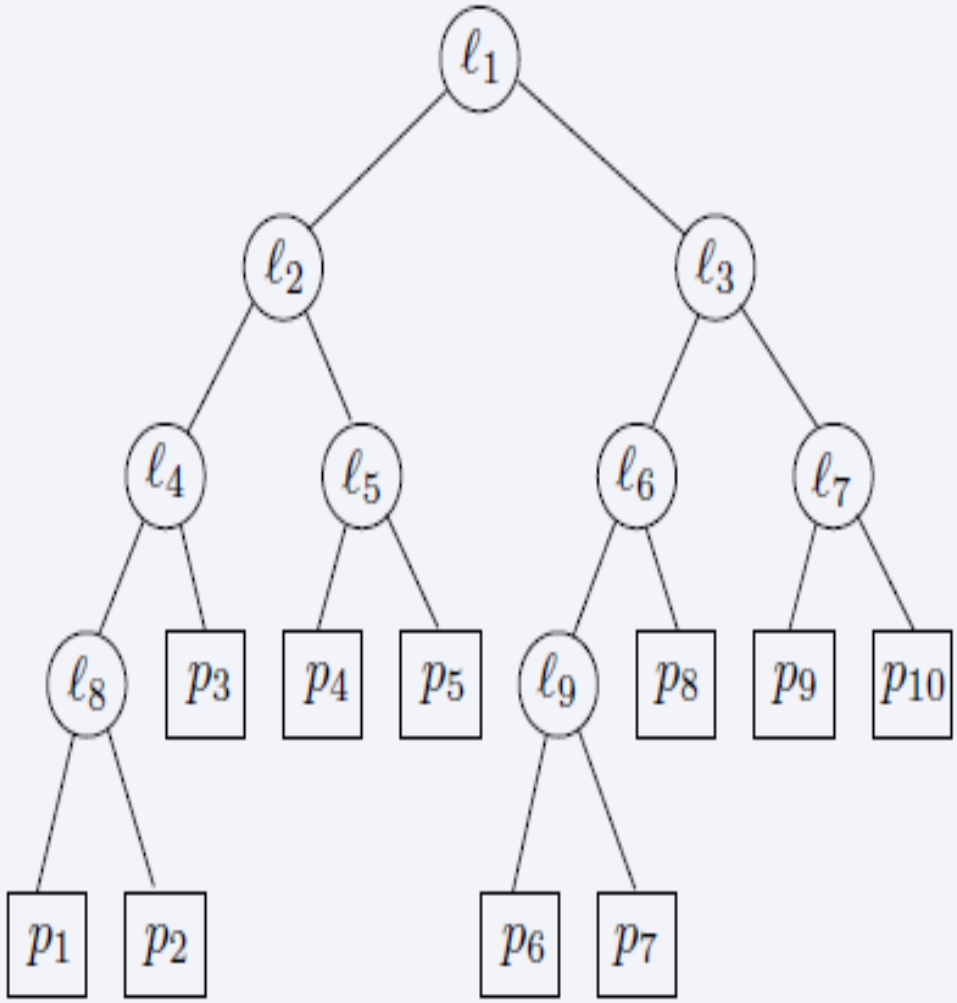
Input. A set of points P and the current depth *depth*.

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


x-list p_1 p_4 p_2 p_3 p_5 p_7 p_9 p_{10} p_6 p_8
y-list p_6 p_3 p_8 p_1 p_7 p_2 p_{10} p_5 p_9 p_4
ℓ_1
ℓ_2
ℓ_3
p_1 p_4 p_2 p_3 p_5
p_3 p_1 p_2 p_5 p_4
p_7 p_9 p_{10} p_6 p_8
p_6 p_8 p_7 p_{10} p_9
p_1 p_2 p_3
p_3 p_1 p_2
p_4 p_5
p_5 p_4
p_7 p_6 p_8
p_6 p_8 p_7
p_9 p_{10}
p_{10} p_9


ℓ_1
ℓ_5
ℓ_7
p_4
p_9
p_5
p_{10}
ℓ_2
p_2
p_7
ℓ_3
ℓ_8
p_1
p_8
p_3
p_6
ℓ_9
ℓ_4
ℓ_6


ℓ_1
ℓ_2
ℓ_3
ℓ_4
ℓ_5
ℓ_6
ℓ_7
ℓ_8
p_3
p_4
p_5
ℓ_9
p_8
p_9
p_{10}
p_1
p_2
p_6
p_7

لم ۵.۳:

یک kd-tree برای مجموعه‌ای از n نقطه، از فضای ذخیره سازی $O(n)$ استفاده می‌کند و در زمان $O(n \log n)$ ساخته می‌شود.

تحلیل زمان ساخت:

۱. هر گره یک x-list (مرتب شده بر اساس x) و یک y-list (مرتب شده بر اساس y) دارد.

۲. هم x-list و هم y-list می‌توانند در زمان خطی با استفاده از دو لیست والد خود ساخته شوند.(چون تنها نیاز به جابجایی دارند)

۳. زمان کلی برای ساختن همه‌ی x-list ها و y-list ها برابر است با:

$$T_1(n) = O(n) + 2T_1(n/2) = O(n \log n)$$

۴.از آنجاییکه هر خط جداکننده در زمان $O(1)$ پیدا می‌شود و $n-1$ خط جداکننده در یک kd-tree داریم :

$$T_2(n) = (n-1) * O(1) = O(n)$$

بنابراین زمان کلی: $T_1(n) + T_2(n) = O(n \log n)$

تحلیل فضای ذخیره سازی:

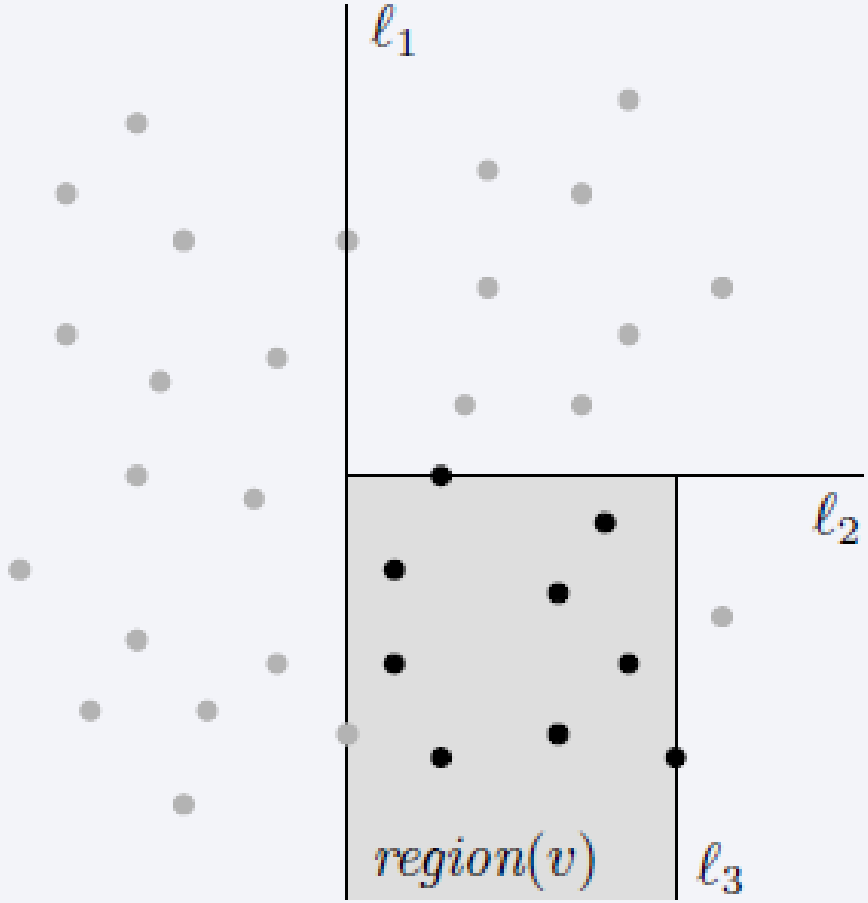
- هر برگ در kd-tree یک نقطه‌ی مجزا از P را ذخیره می‌کند .بنابراین n برگ داریم از آنجاییکه kd-tree یک درخت جستجوی دودویی است هر برگ و هر گره داخلی از فضای O(1) استفاده می‌کنند پس داریم:

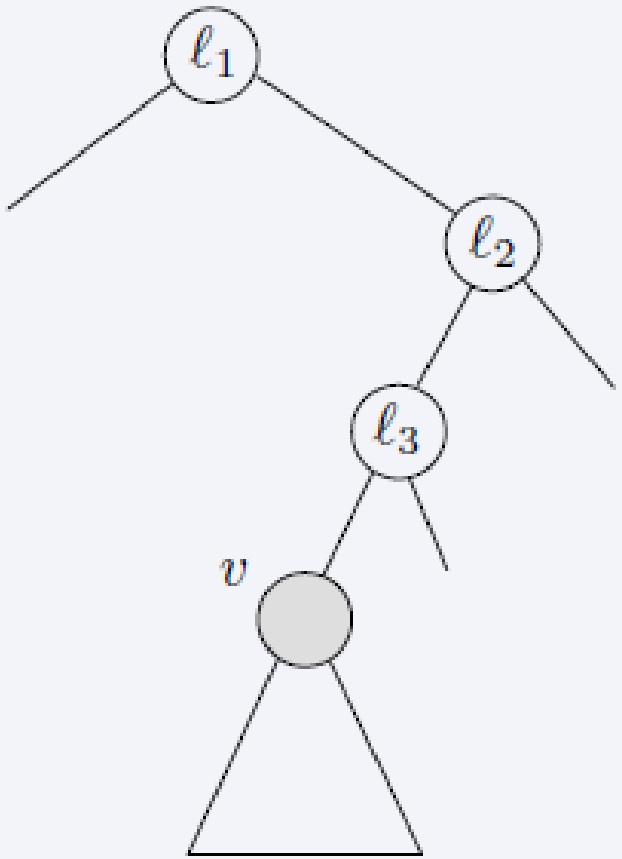
$$n^* O(1) = O(n)$$

- علاوه بر این زمانی که یک x-list یا y-list به دو زیر لیست تبدیل شود لیست اولیه می‌تواند حذف شود بنابراین در هر لحظه از زمان O(n) فضا برای ذخیره سازی همه‌ی x-list ها و y-list ها نیاز است.

در نتیجه فضای ذخیره سازی مورد نیاز:O(n)

Region(v): ناحیه‌ی مطابق با گره v که مرزهای آن خطوط جداکننده-ی ذخیره شده در اجداد v هستند.



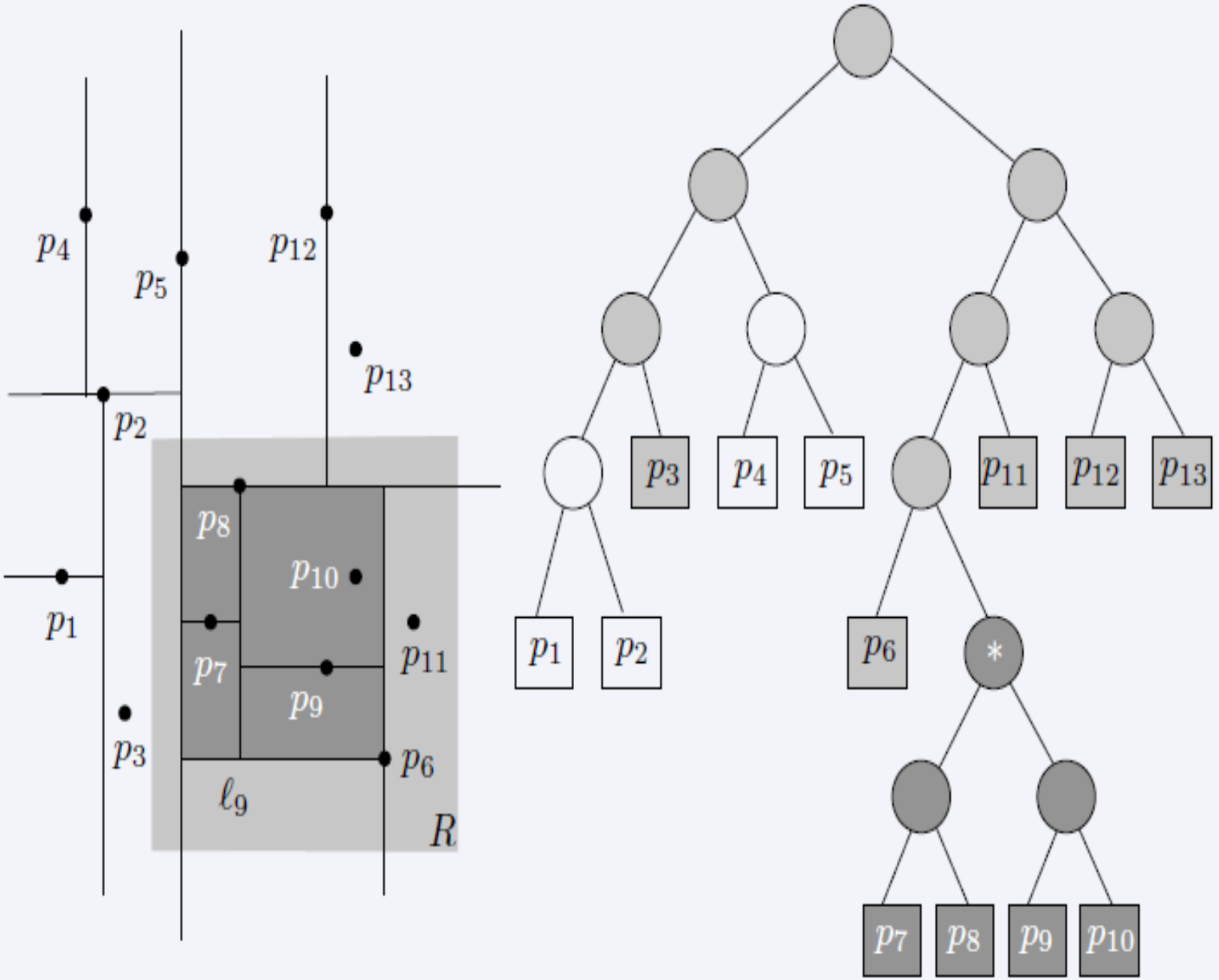


ساختمان یک کوئری:

یک مستطیل کوئری داده می‌شود.

ما فقط زمانی زیردرخت v را جستجو می‌کنیم که مستطیل کوئری با ناحیه‌ی region(v) اشتراک داشته باشد.

زمانی که region(v) به طور کامل در مستطیل کوئری قرار گیرد تمام نقاط ذخیره شده دراین زیر درخت گزارش می‌شود.


p_4
p_5
p_{12}
p_{13}
p_2
p_8
p_{10}
p_1
p_7
p_{11}
p_9
p_3
p_6
ℓ_9
R
p_3
p_4
p_5
p_{11}
p_{12}
p_{13}
p_1
p_2
p_6
*
p_7
p_8
p_9
p_{10}

Algorithm SEARCHKDTREE(v, R)

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

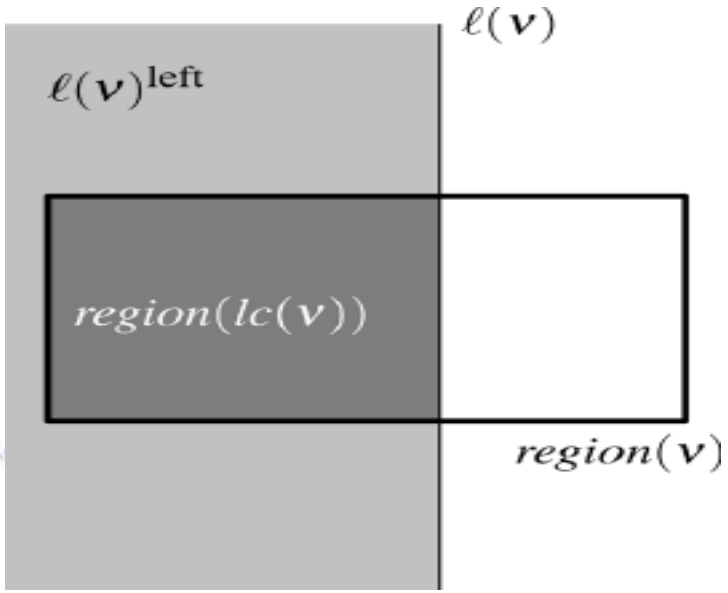
Output. All points at leaves below v that lie in the range.

```
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)
```

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

l(v) : خط جداکننده‌ی ذخیره شده در گره

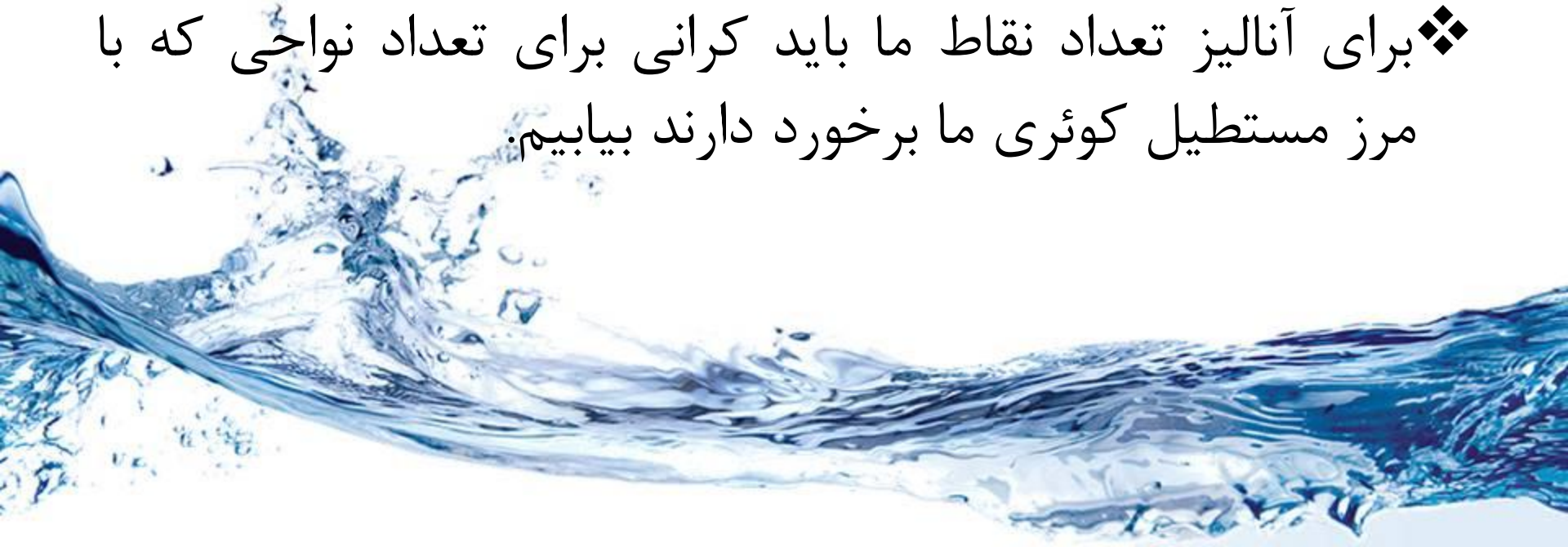
$l(v)^{left}$: نیم صفحه‌ای که سمت چپ l(v) قرار دارد.



لم ۵.۴:

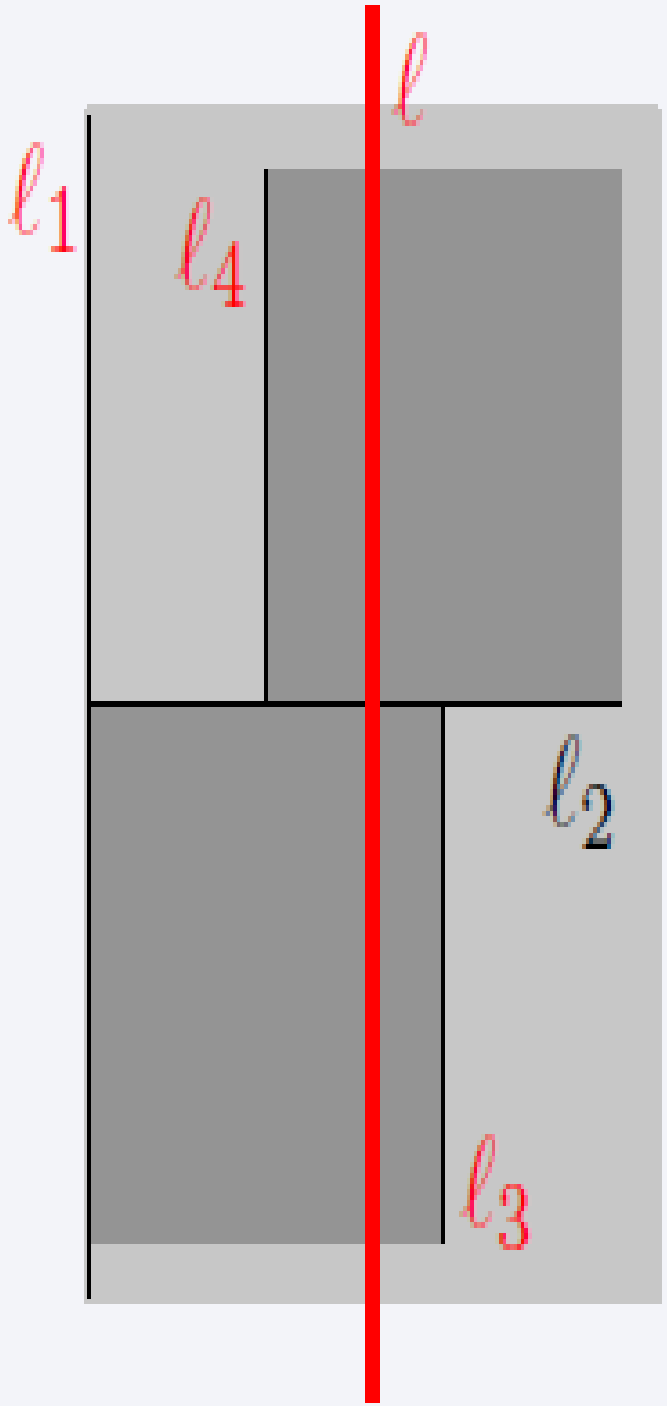
یک کوئری با یک مستطیل موازی المحور در یک kd-tree که n نقطه را ذخیره می‌کند را میتوان در زمان $O(\sqrt{n} + k)$ اجرا کرد، که k تعداد نقاط گزارش شده است.

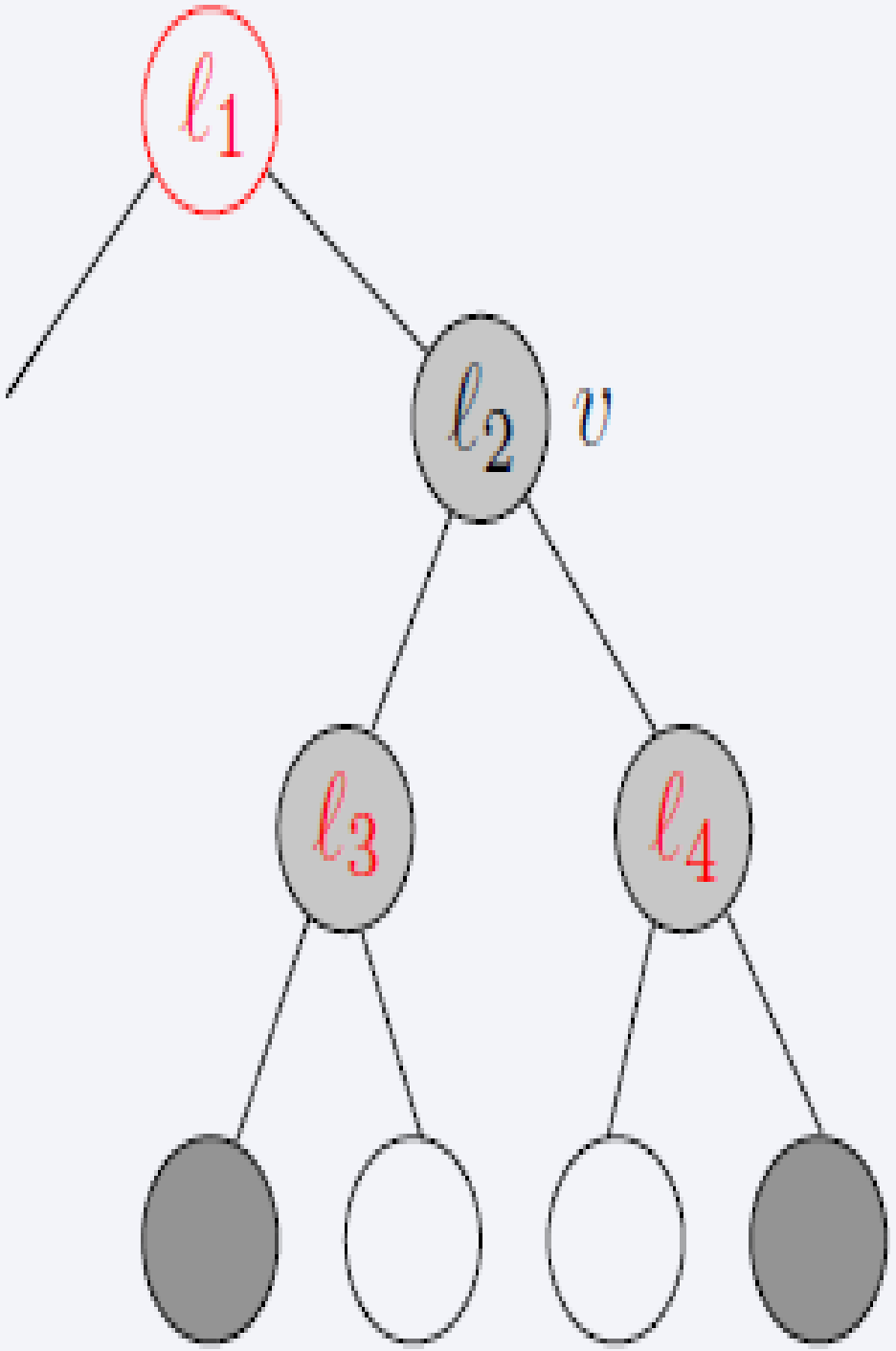
- برای نواحی که به طور کامل درون دامنه‌ی کوئری قرار می‌گیرند گزارش نقاط به زمان خطی نیاز دارد که متناسب با تعداد نقاطی است که گزارش می‌شود.$O(k)$
- آنچه باقی می‌ماند این است که یک کران برای تعداد گره‌هایی که در الگوریتم کوئری دیده می‌شوند ولی در هیچکدام از زیردرخت‌های پیمایش شده نیستند بیابیم.
- برای آنالیز تعداد نقاط ما باید کرانی برای تعداد نواحی که با مرز مستطیل کوئری ما برخورد دارند بیابیم.



چون مستطیل کوئری ما از ۴ یال تشکیل شده است ما کران را برای یک یال بدست آورده و در ۴ ضرب می‌کنیم.

خط l را به صورت عمودی در نظر بگیرید این خط یا فرزند چپ ریشه را قطع می‌کند یا فرزند راست. برای اینکه بتوانیم یک رابطه‌ی بازگشتی درست بنویسیم باید دو سطح پایین برویم.در سطح دوم خط عمودی حداکثر با ۲ ناحیه برخورد می‌کند.


ℓ
ℓ_1
ℓ_4
ℓ_2
ℓ_3


ℓ_1
ℓ_2
v
ℓ_3
ℓ_4

پس داریم:

$$Q(n) = 2 + 2Q\left(\frac{n}{4}\right) \longrightarrow O(\sqrt{n})$$

$$\Rightarrow O(\sqrt{n} + k)$$

قضیه ۵.۵:

یک kd-tree برای مجموعه p شامل n نقطه در صفحه از $O(n)$ حافظه استفاده می‌کند و در زمان $O(n \log n)$ ساخته می‌شود. یک کوئری با بازه مستطیلی روی kd-tree، $O(\sqrt{n} + k)$ زمان می‌گیرد که k تعداد نقاط گزارش شده است.