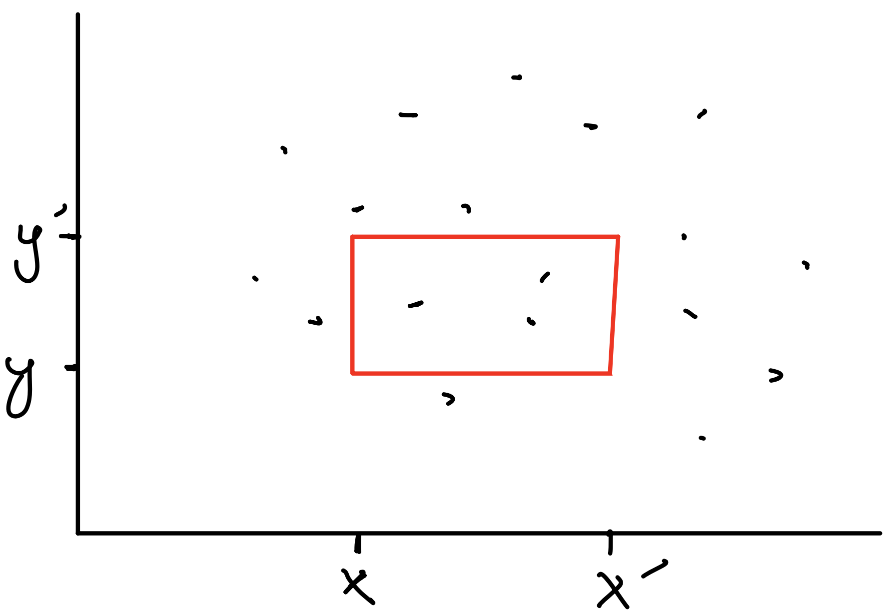
L8 - Orthogonal range searching

- Consider set $P \subseteq \mathbb{R}^d$ and a range $[x_1, x_1'] \times \dots \times [x_d, x_d'] \subseteq \mathbb{R}^d$.

L8 - Orthogonal range searching

- Consider set $P \subseteq \mathbb{R}^d$ and a range $[x_1, x_1'] \times \dots \times [x_d, x_d'] \subseteq \mathbb{R}^d$.

- Find points of P belonging to the range.



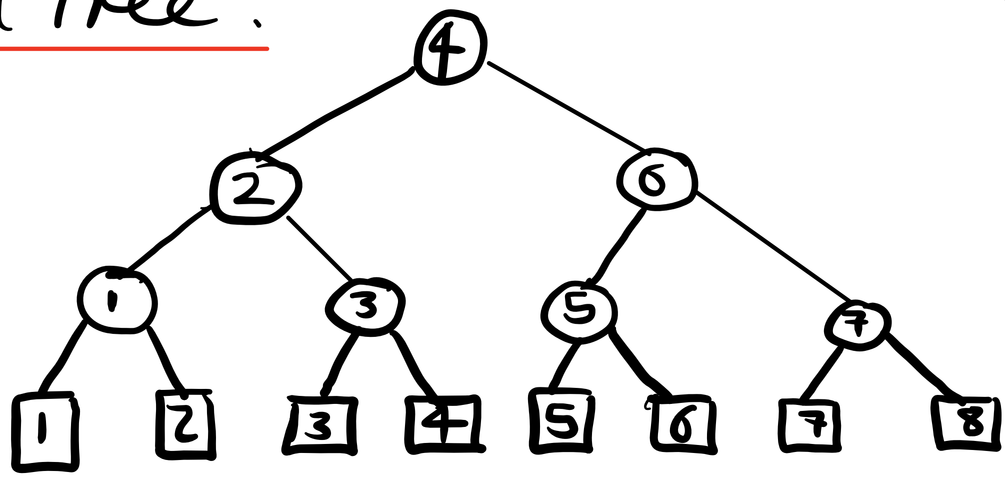
- Relevant to querying databases.

1-d range searching

$P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}$ & $x \leq x'$

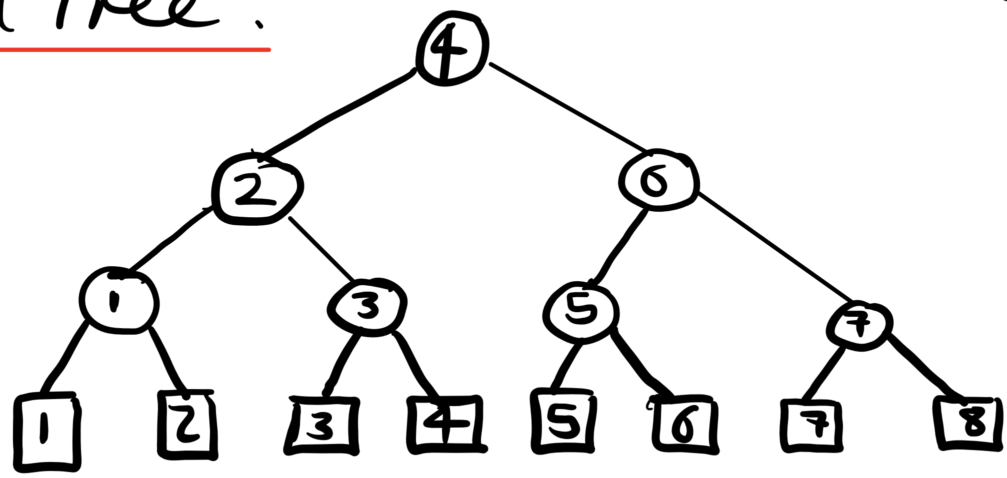
1-d range searching

- $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}$ & $x \leq x'$
- Points of P stored as leaves of binary balanced tree.



1-d range searching

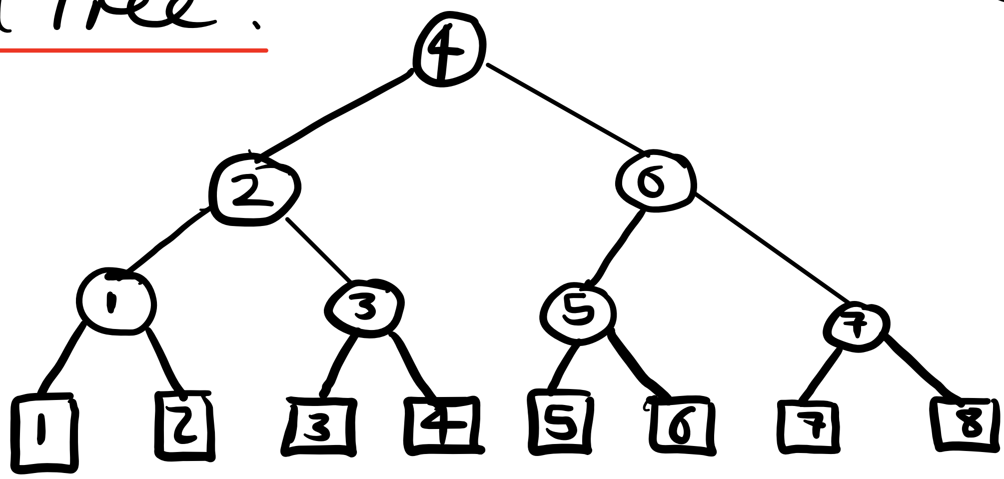
- $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}$ & $x \leq x'$
- Points of P stored as leaves of binary balanced tree.



- Node v: stores max value x_v in left subtree.

1-d range searching

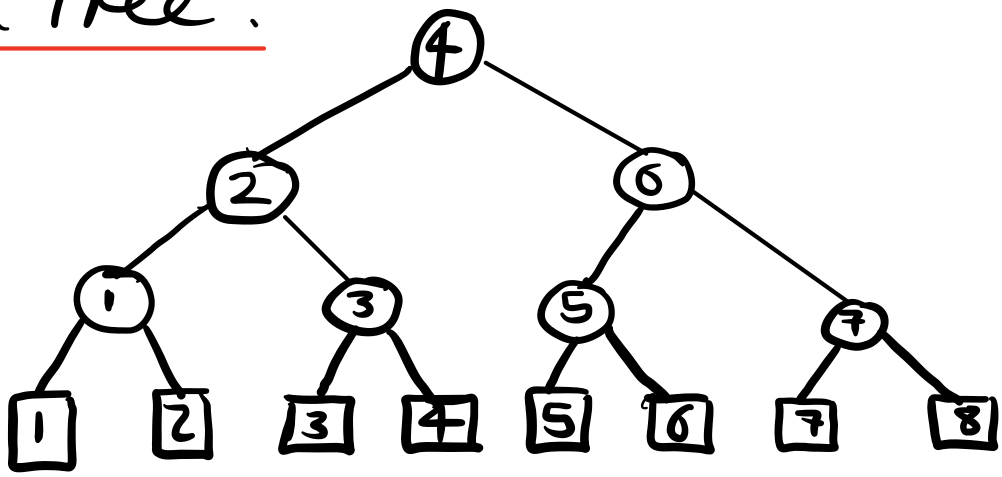
- $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}$ & $x \le x'$
- Points of P stored as leaves of binary balanced tree.



- Node v : stores max value x_v in left subtree.
- Left subtree of v: elements $\le x_v$
- Right subtree of v: elements $> x_v$

1-d range searching

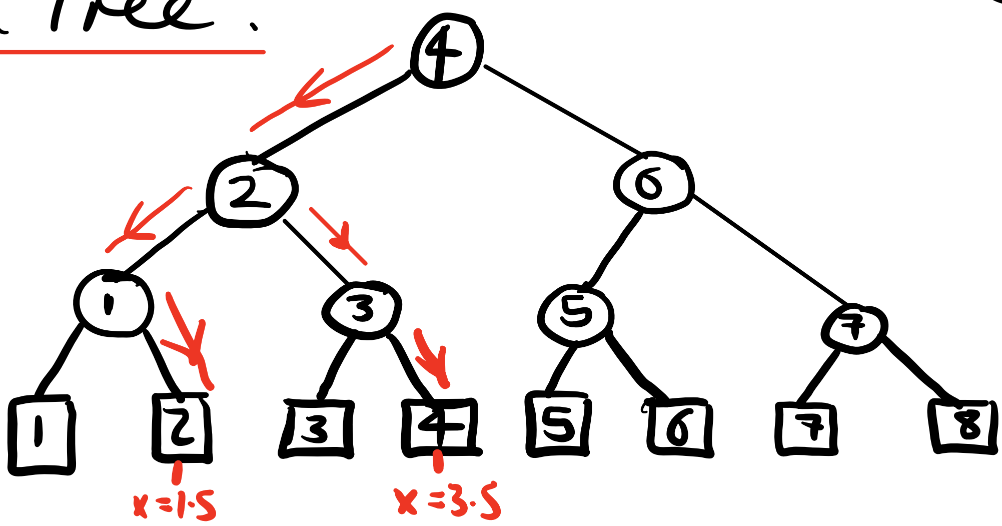
- $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}$ & $x \leq x'$
- Points of P stored as leaves of binary balanced tree.



- Node v: stores max value x_v in left subtree.
- Left subtree of v: elements $\leq x_v$
- Right subtree of v: elements $> x_v$
- Point $x \in \mathbb{R}$ determines path from root to leaf: at node v, go left if $x \leq x_v$, else go right.

1-d range searching

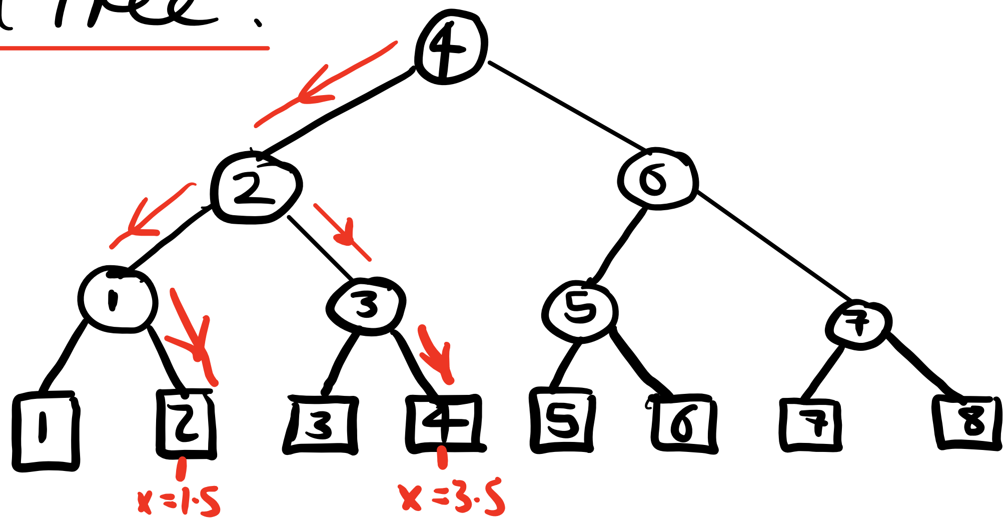
- $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}$ & $x \leq x'$
- Points of P stored as leaves of binary balanced tree.



- Node v: stores max value x_v in left subtree.
- Left subtree of v: elements $\leq x_v$
- Right subtree of v: elements $> x_v$
- Point $x \in \mathbb{R}$ determines path from root to leaf: at node v, go left if $x \leq x_v$, else go right.
- Eg. $x = 1.5$, $x = 3.5$.

1-d range searching

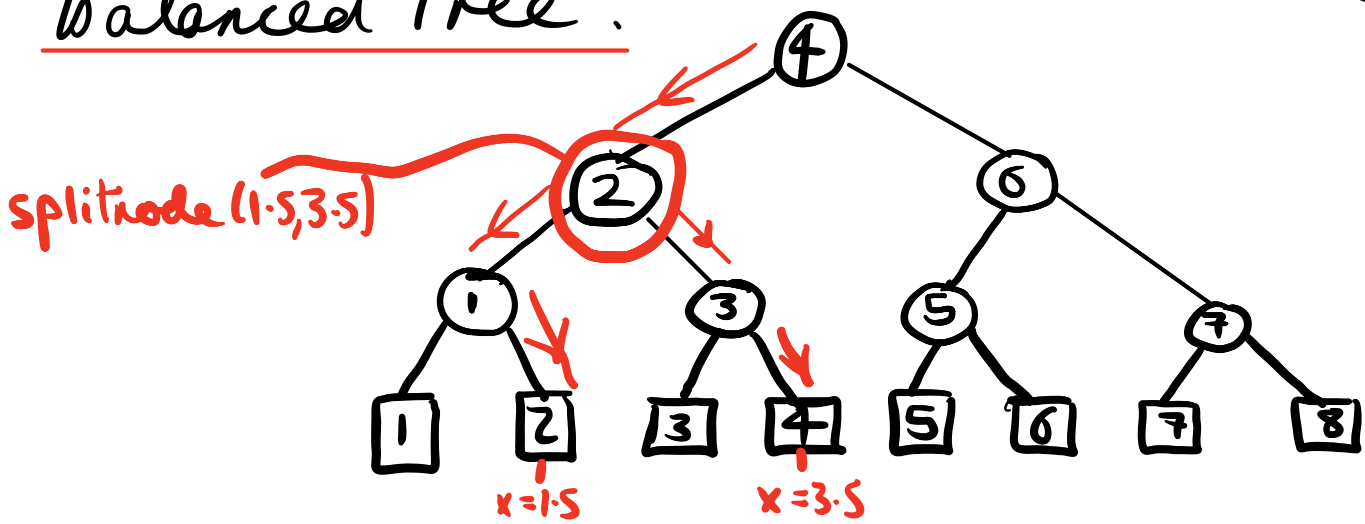
- $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}$ & $x \leq x'$
- Points of P stored as leaves of binary balanced tree.



- Node v: stores max value x_v in left subtree.
- Left subtree of v: elements $\leq x_v$
- Right subtree of v: elements $> x_v$
- Point $x \in \mathbb{R}$ determines path from root to leaf: at node v, go left if $x \leq x_v$, else go right.
- Eg. $x = 1.5$, $x = 3.5$.
- At $x \leq x'$, splitnode(x, x') is last node at which paths for x & x' agree.

1-d range searching

- $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}$ & $x \leq x'$
- Points of P stored as leaves of binary balanced tree.



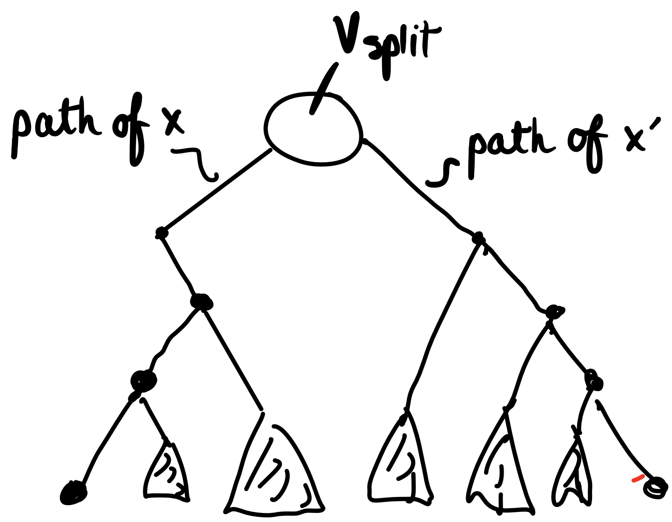
- Node v: stores max value x_v in left subtree.
- Left subtree of v: elements $\leq x_v$
- Right subtree of v: elements $> x_v$
- Point $x \in \mathbb{R}$ determines path from root to leaf: at node v, go left if $x \leq x_v$, else go right.
- Eg. $x = 1.5$, $x = 3.5$.
- At $x \leq x'$, splitnode(x, x') is last node at which paths for x & x' agree.

Algorithm

- Input: $P = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}$ stored in a balanced binary tree & $x \leq x'$.
- Output: points of P in range $x \leq x'$.

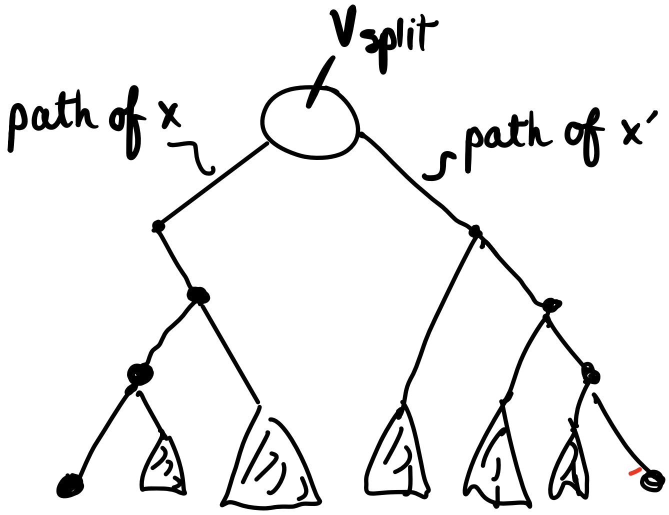
Algorithm

- Input: $P = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}$ stored in a balanced binary tree & $x \leq x'$.
- Output: points of P in range $x \leq x'$.
- Find splitnode v_{split} of x & x':



Algorithm

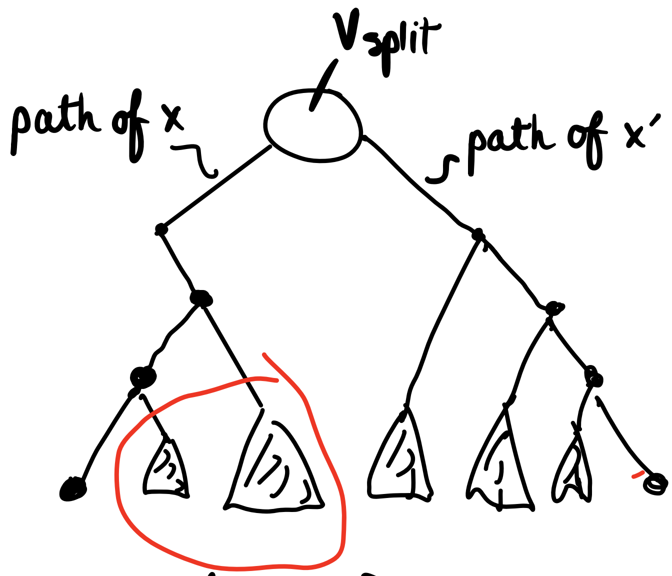
- Input : $P = \{x_1, \dots, x_n\} \subseteq \mathbb{R}$ stored in a balanced binary tree & $x \leq x'$.
- Output : points of P in range $x \leq x'$.
- Find splitnode v_{split} of x & x' :



- Follow path of x from v_{split} to a leaf

Algorithm

- Input: $P = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}$ stored in a balanced binary tree & $x \leq x'$.
- Output: points of P in range $x \leq x'$.
- Find splitnode v_{split} of x & x':

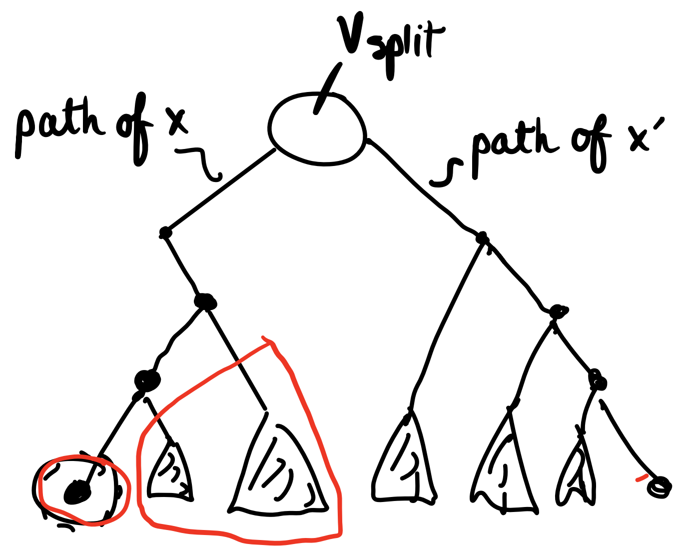


- Follow path of x from v_{split} to a leaf:

a) If at node v, x moves **Left**, **report** all leaves in right subtree of v.

Algorithm

- Input : $P = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}$ stored in a balanced binary tree & $x \leq x'$.
- Output : points of P in range $x \leq x'$.
- Find splitnode v_{split} of x & x' :



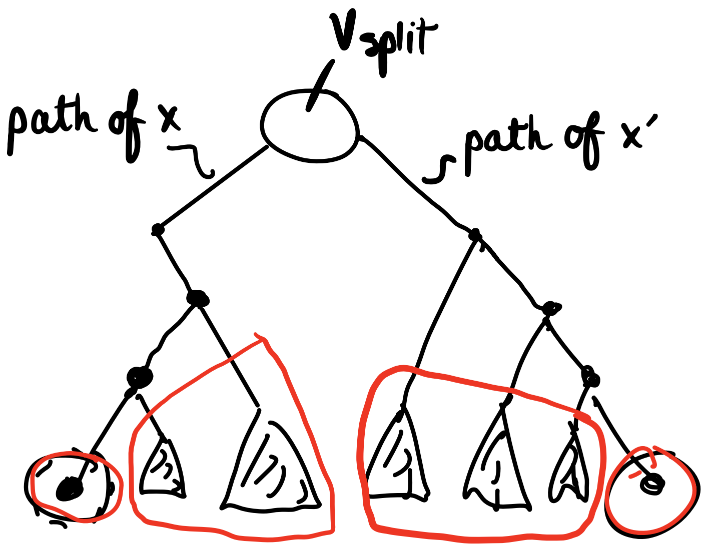
- Follow path of x from v_{split} to a leaf :

a) If at node v, x moves Left, report all leaves in right subtree of v.

b) At leaf, check if it belongs to $[x, x']$.

Algorithm

- Input: $P = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}$ stored in a balanced binary tree & $x \leq x'$.
- Output: points of P in range $x \leq x'$.
- Find splitnode v_{split} of x & x':



- Follow path of x from v_{split} to a leaf:

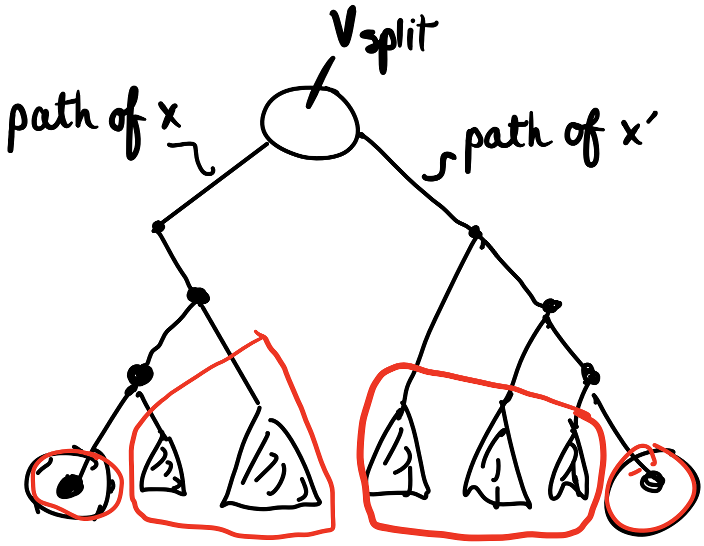
a) If at node v, x moves left, report all leaves in right subtree of v.

b) At leaf, check if it belongs to $[x, x']$.

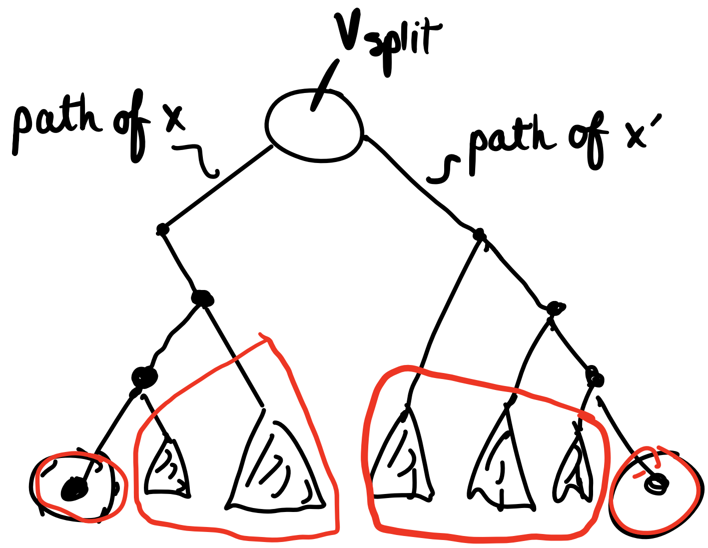
- Follow path of x' from v_{split} to a leaf:

a) If at node v, x' moves right, report all leaves in left subtree of v.

b) At leaf, check if it belongs to $[x, x']$.

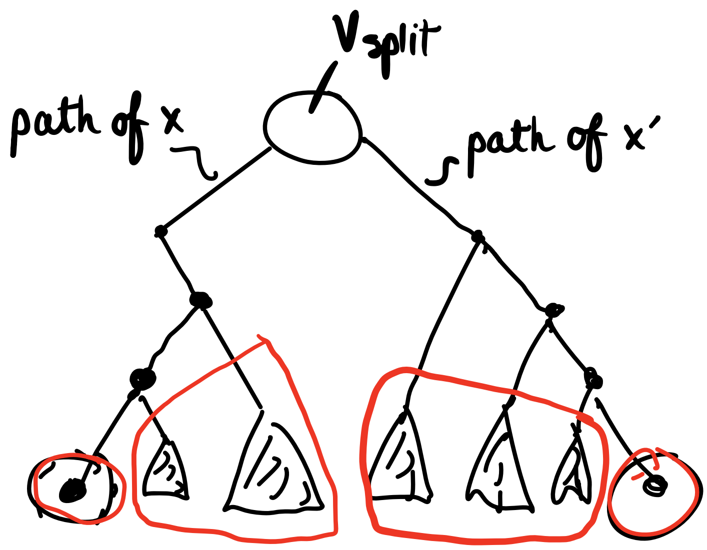


Why does this find all solutions?



Why does this find all solutions?

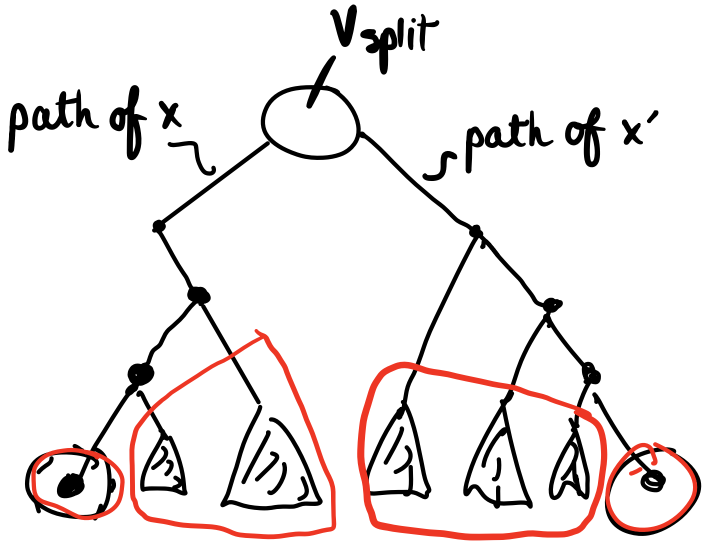
- Suppose $x \leq p \leq v_{spl}$,



Why does this find all solutions?

- Suppose $x \leq p \leq V_{spl}$.

Then p is reported when paths for x and p diverge, or at leaf p itself.

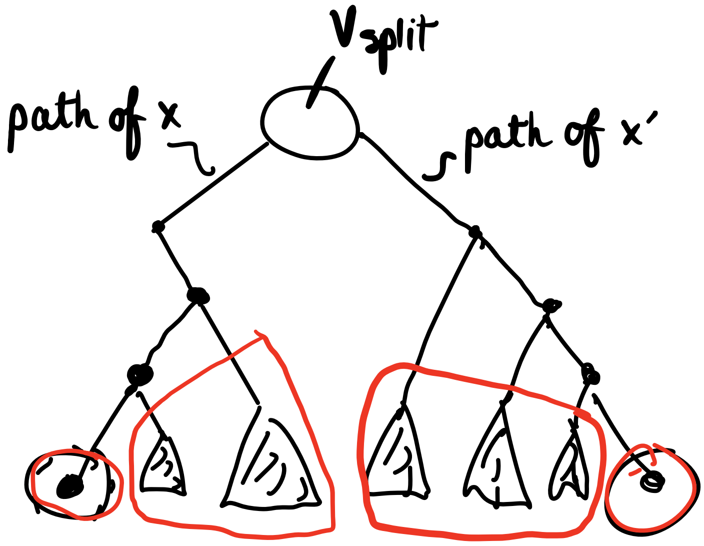


Why does this find all solutions?

- Suppose $x \leq p \leq V_{spl}$.

Then p is reported when paths for x and p diverge, or at leaf p itself.

- Similarly if $V_{spl} \leq p \leq x'$.



Why does this find all solutions?

- Suppose $x \leq p \leq V_{spl}$.

Then p is reported when paths for x and p diverge, or at leaf p itself.

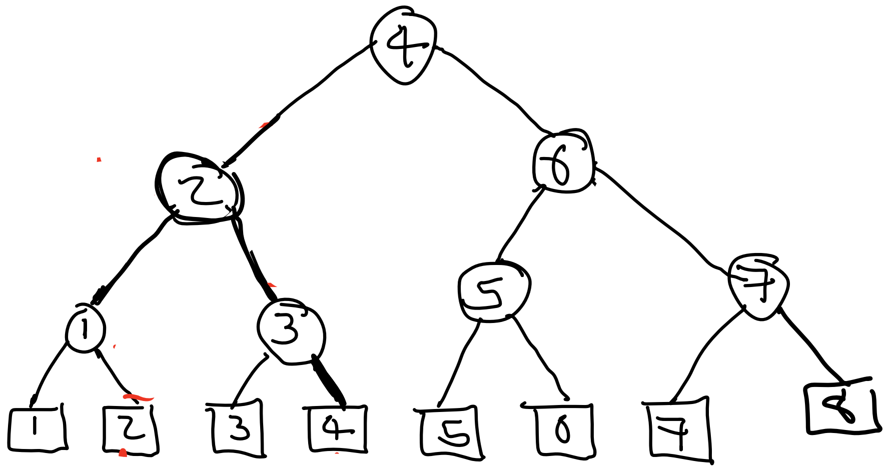
- Similarly if $V_{spl} \leq p \leq x'$.

Complexity

- Time $O(\log n)$ to follow path of x.
- Likewise for x'.
- Time to report k solutions is $O(k)$.

Total complexity $O(\underbrace{\log n}_{\text{no of leaves}} + \underbrace{k}_{\text{no of solutions}})$

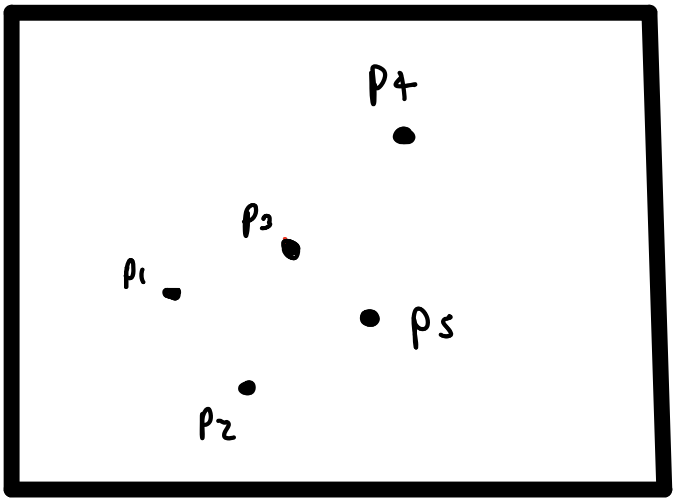
[5.5, 9]



Do in class.

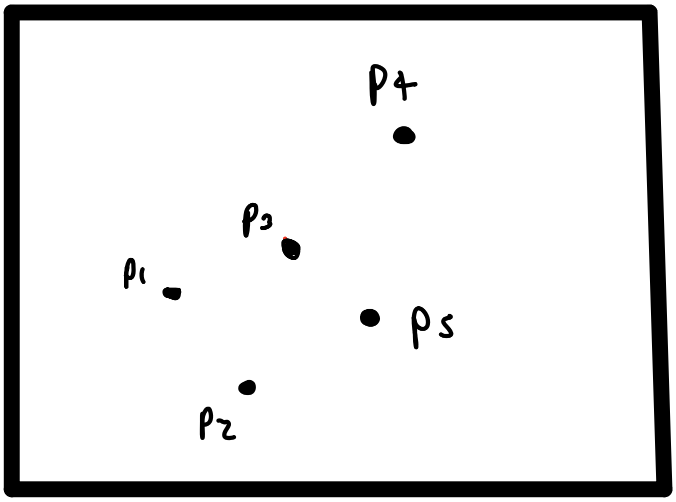
2-d range searching

- $P \subseteq \mathbb{R}^2$ (assume no pts have same x or y coord)



2-d range searching

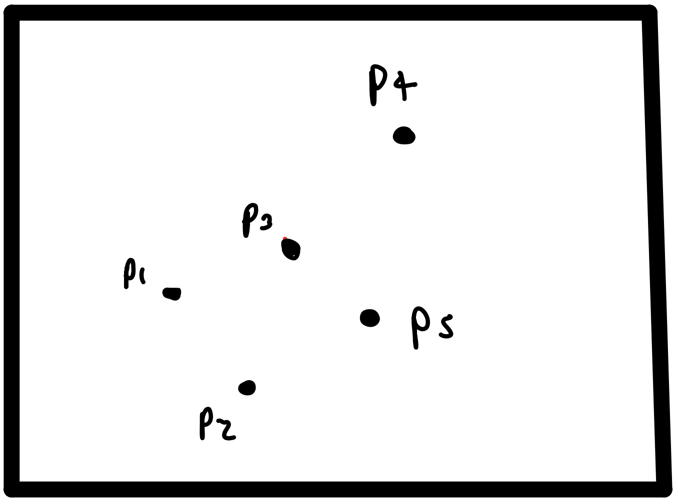
- $P \subseteq \mathbb{R}^2$ (assume no pts have same x or y coord)



- Using vertical line, split through median pt ordered by x-coord.

2-d range searching

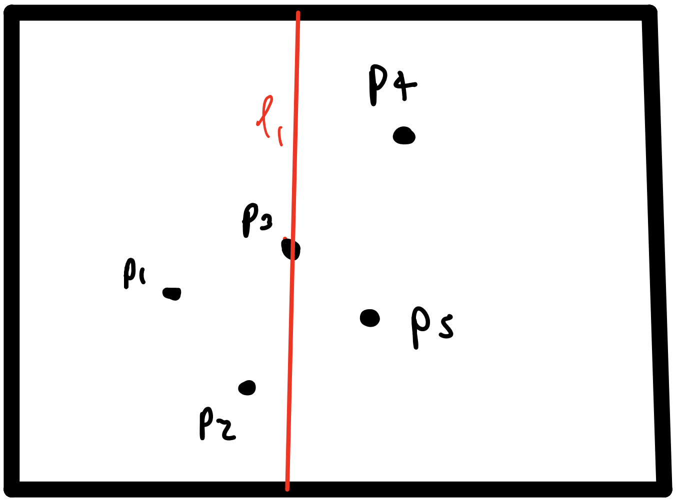
- $P \subseteq \mathbb{R}^2$ (assume no pts have same x or y coord)



- Using vertical line, split through median pt ordered by x-coord. Count point on line in left region (so should be same no or one more on left)

2-d range searching

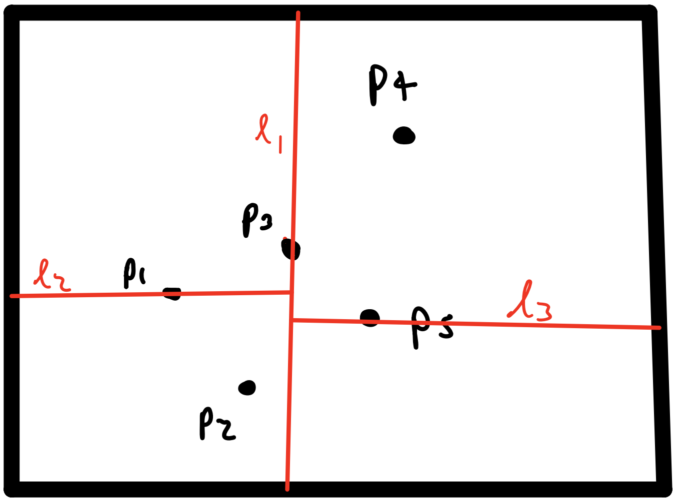
- $P \subseteq \mathbb{R}^2$ (assume no pts have same x or y coord)



- Using vertical line, split through median pt ordered by x-coord. Count point on line in left region (so should be same no or one more on left)

2-d range searching

- $P \subseteq \mathbb{R}^2$ (assume no pts have same x or y coord)

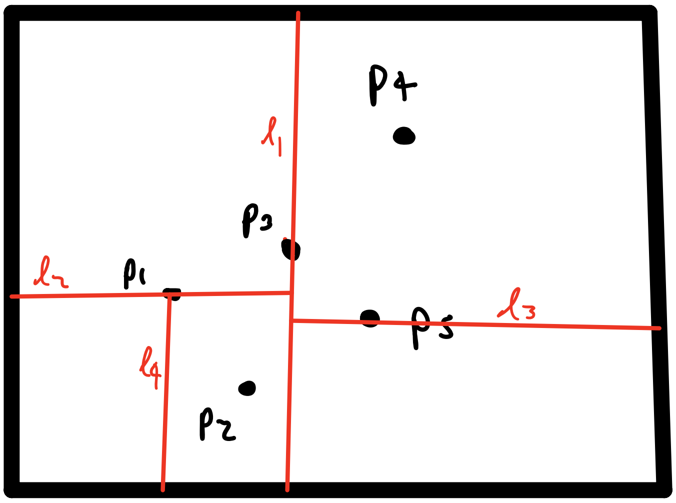


- Using vertical line, split through median pt ordered by x-coord. Count point on line in left region (so should be same no or one more on left)

- Split left and right regions using horizontal lines through point with median y-coord (should be same no or 1 more in lower (left) region compared to upper (right) region.)

2-d range searching

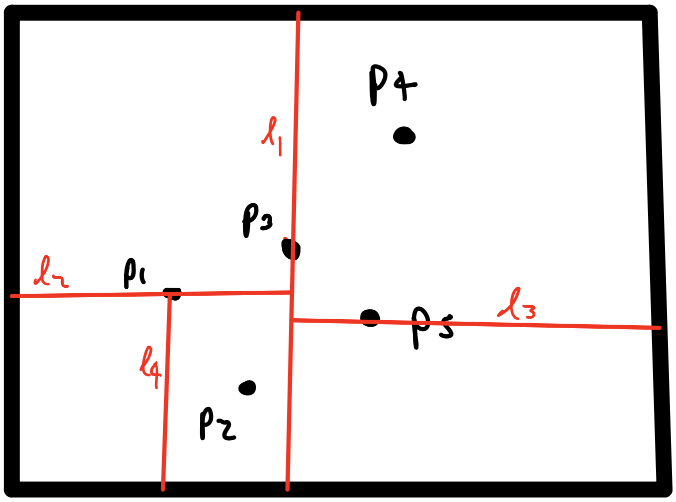
- $P \subseteq \mathbb{R}^2$ (assume no pts have same x or y coord)



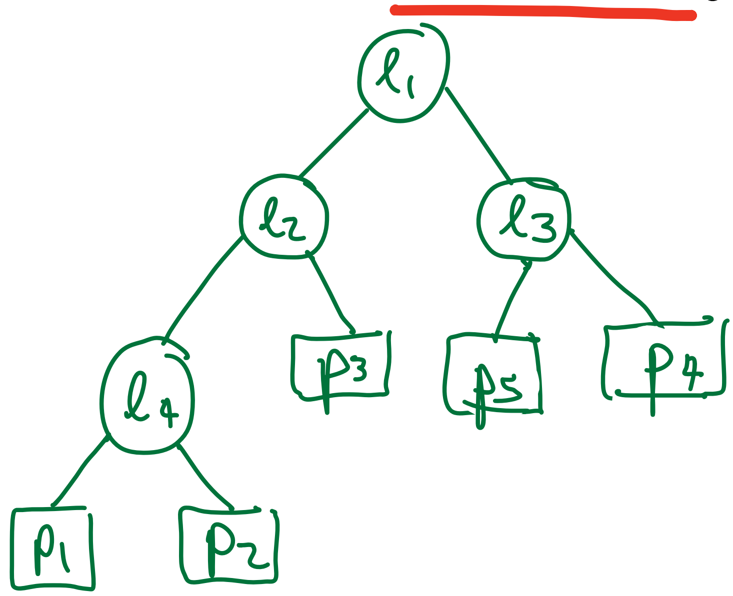
- Using vertical line, split through median pt ordered by x-coord. Count point on line in left region (so should be same no or one more on left)
- Split left and right regions using horizontal lines through point with median y-coord (should be same no or 1 more in lower (left) region compared to upper (right) region.)
- Repeat until each region has 1 pt.

2-d range searching

- $P \subseteq \mathbb{R}^2$ (assume no pts have same x or y coord)

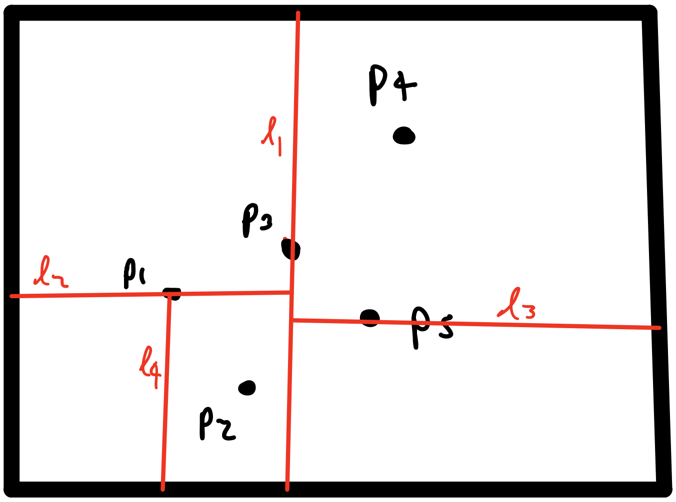


- Using vertical line, split through median pt ordered by x-coord. Count point on line in left region (so should be same no or one more on left)
- Split left and right regions using horizontal lines through point with median y-coord (should be same no or 1 more in lower (left) region compared to upper (right) region.)
- Repeat until each region has 1 pt.
- Creates a structure called a KD-tree:

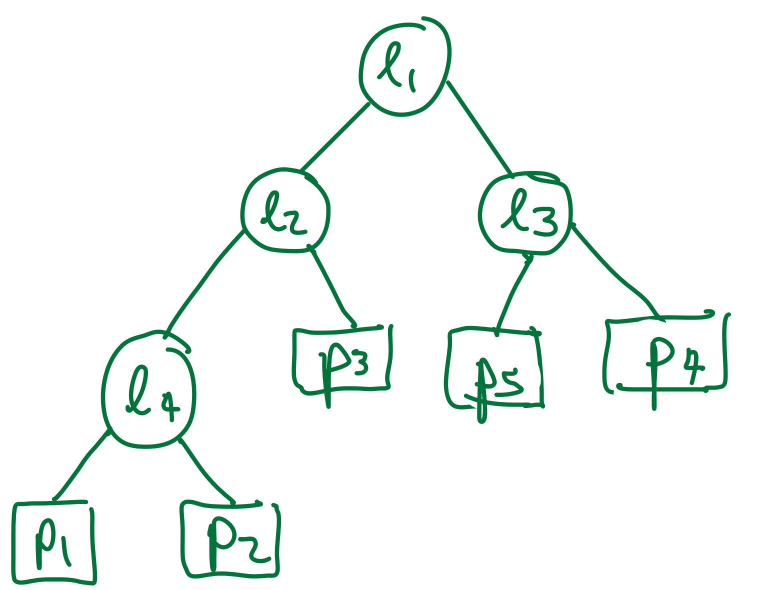


2-d range searching

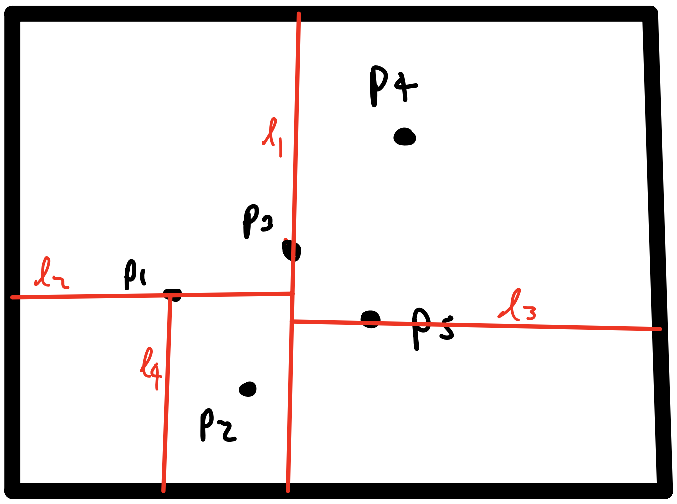
- $P \subseteq \mathbb{R}^2$ (assume no pts have same x or y coord)

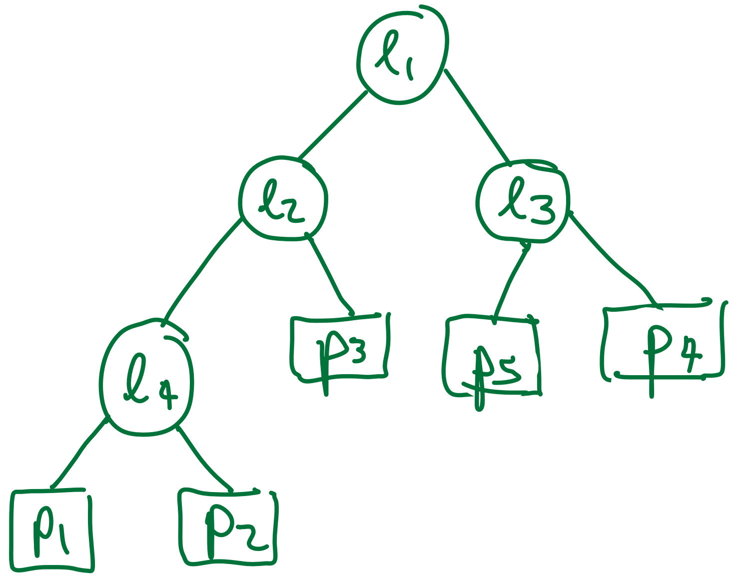


- Using vertical line, split through median pt ordered by x-coord. Count point on line in left region (so should be same no or one more on left)
- Split left and right regions using horizontal lines through point with median y-coord (should be same no or 1 more in lower (left) region compared to upper (right) region.)
- Repeat until each region has 1 pt.
- Creates a structure called a KD-tree:



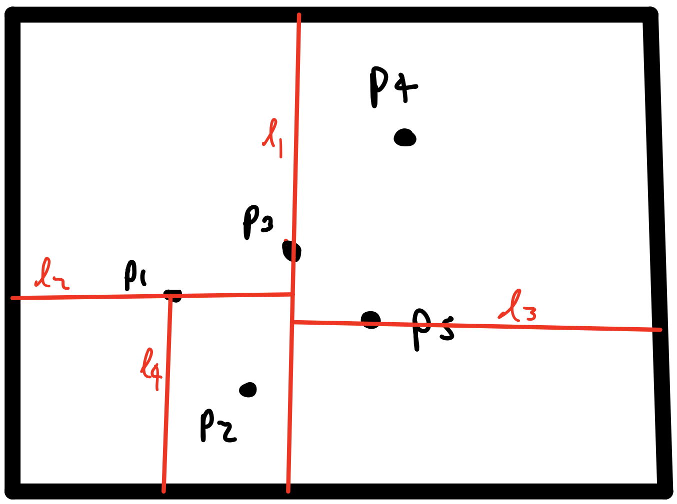
KD Trees

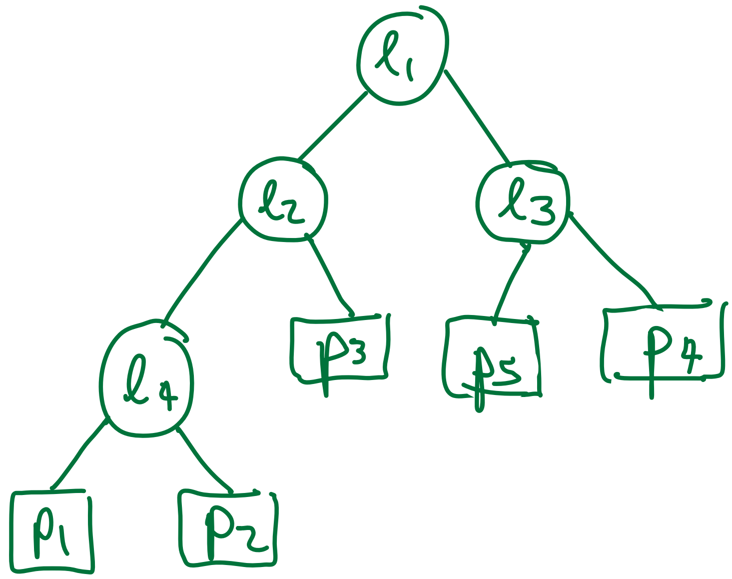




- balanced binary tree

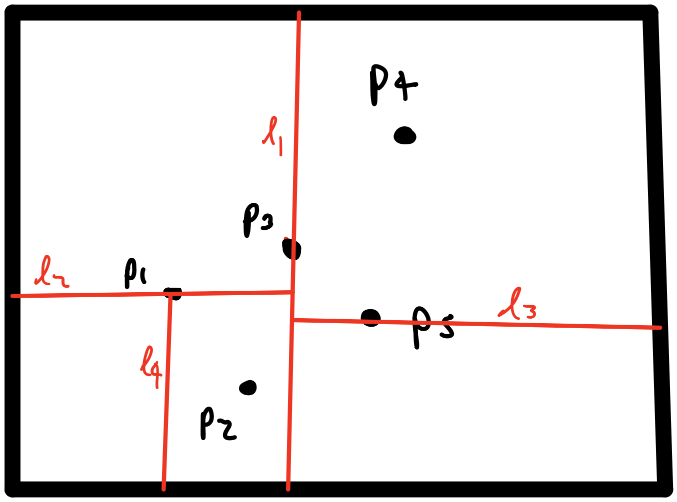
KD Trees

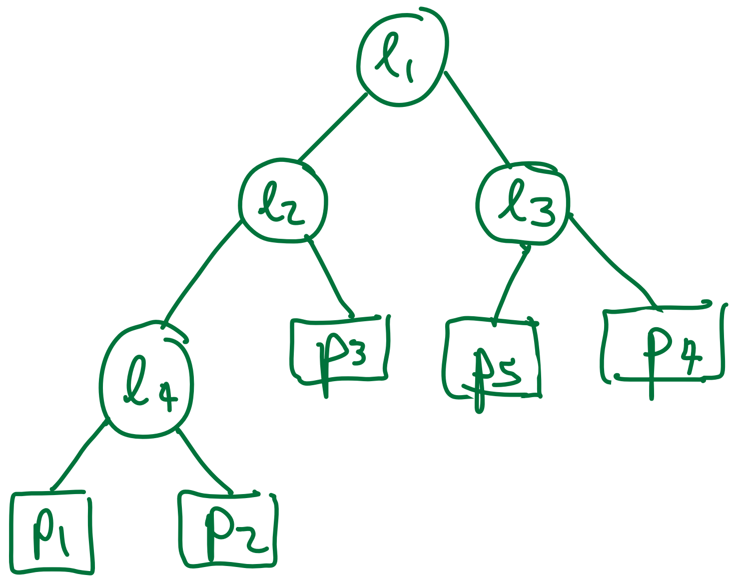




- balanced binary tree
- nodes of even depth store vert lines by x-coord.

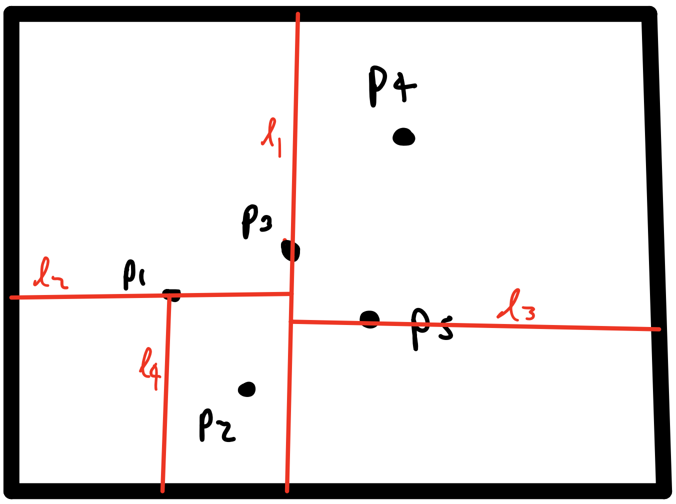
KD Trees

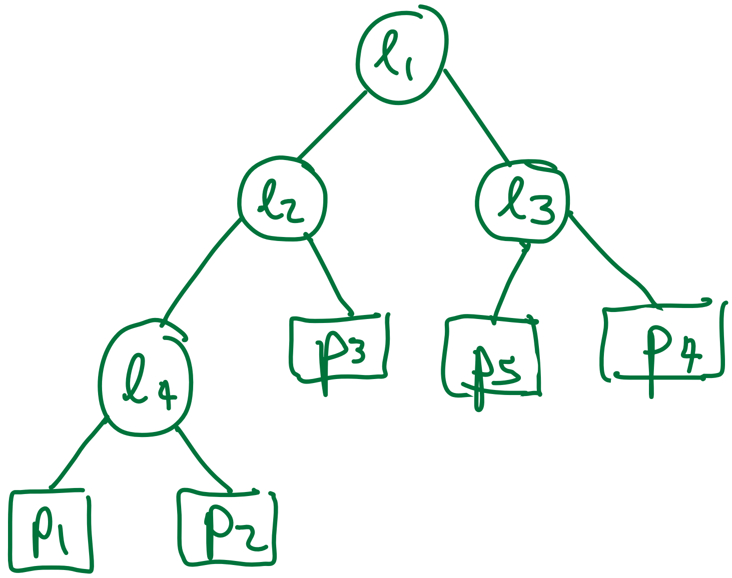




- balanced binary tree
- nodes of even depth store vert lines by x-coord.
- .... odd depth store hor. lines by y-coord.

KD Trees

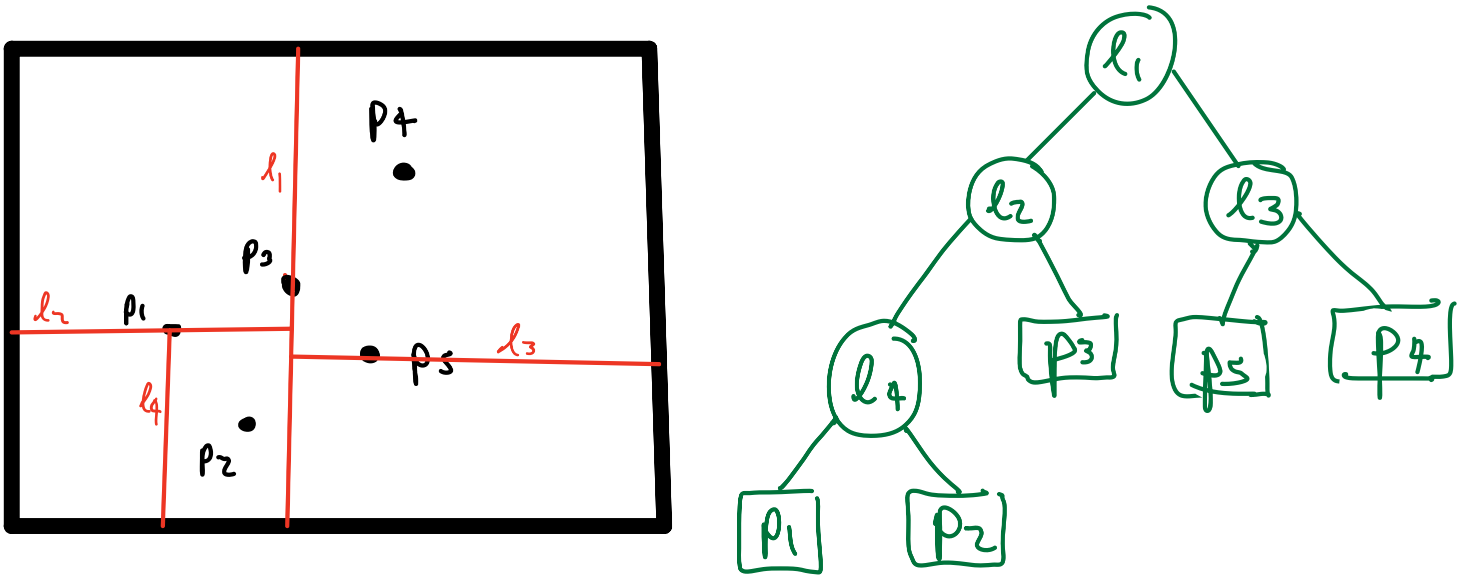




- balanced binary tree
- nodes of even depth store vert lines by x-coord.
- . . . . odd depth store hor. lines by y-coord.

- Takes O(n) storage where n is no of leaves.
- O(nlogn) to create kd tree.

KD Trees



- balanced binary tree
- nodes of even depth store vert lines by x-coord.
- .... odd depth store hor. lines by y-coord.

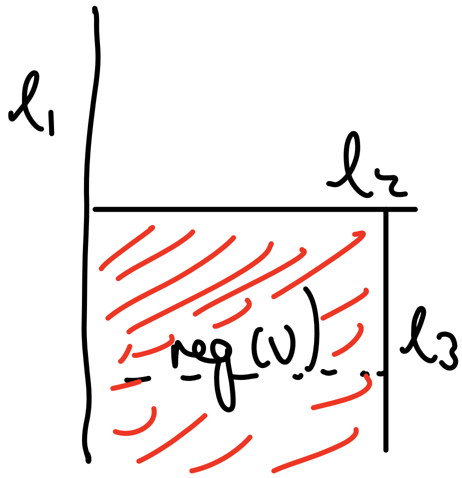
- Takes $O(n)$ storage where n is no of leaves.
- $O(n \log n)$ to create kd tree.
- To search kd-tree, need concept of region of a node.

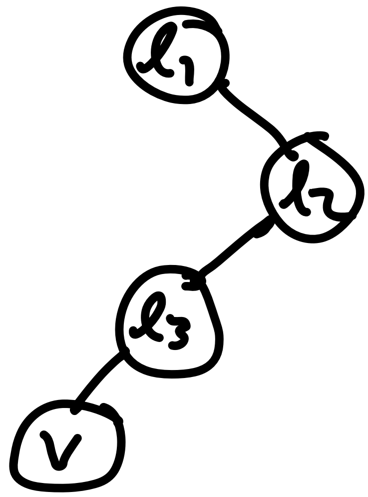
Region of a node

- Region of node v :
rectangular region bounded by ancestors of v
(i.e. if v represents a line, reg(v) is the area which the line divides in two)

Region of a node

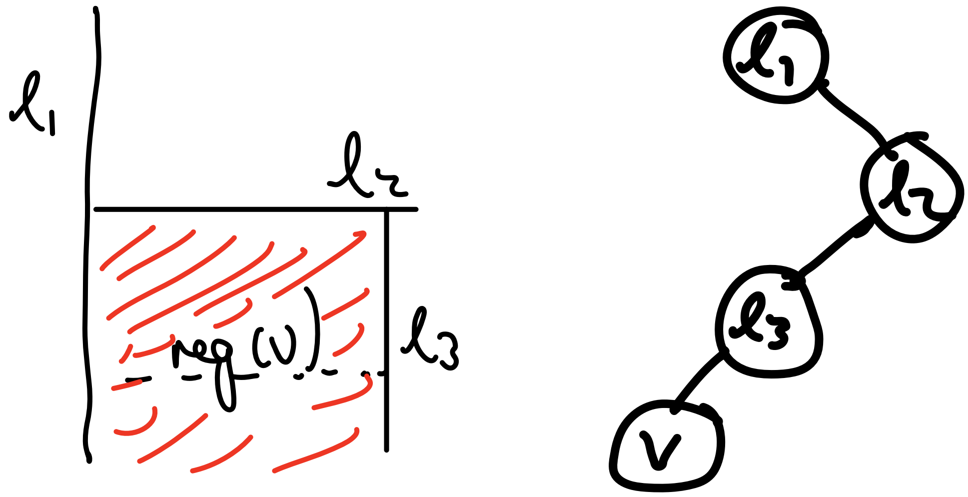
- Region of node v :
rectangular region bounded by ancestors of v
(ie. if v represents a line, reg(v) is the area which the line divides in two)





Region of a node

- Region of node v :
  rectangular region bounded by ancestors of v
  (ie. if v represents a line, reg(v) is the area which the line divides in two)



Recursively,

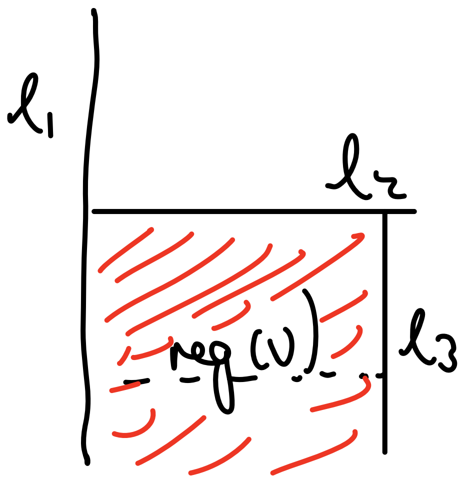
- Region (root) $= \mathbb{R}^2$
- Region $(lc(v)) =$ Region$(v) \cap$ left(v)

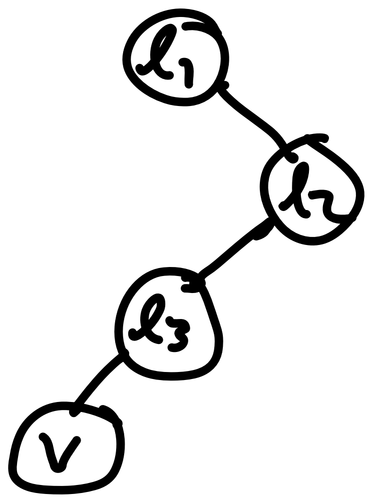
  left child
- Region $(rc(v)) =$ Region$(v) \cap$ right(v)

  right child

Region of a node

- **Region of node v :**
  rectangular region bounded by ancestors of v
  (ie. if v represents a line, reg(v) is the area which the line divides in two)





Recursively,

- Region (root) = $\mathbb{R}^2$
- Region ($lc(v)$) = Region(v) $\cap$ left(v)

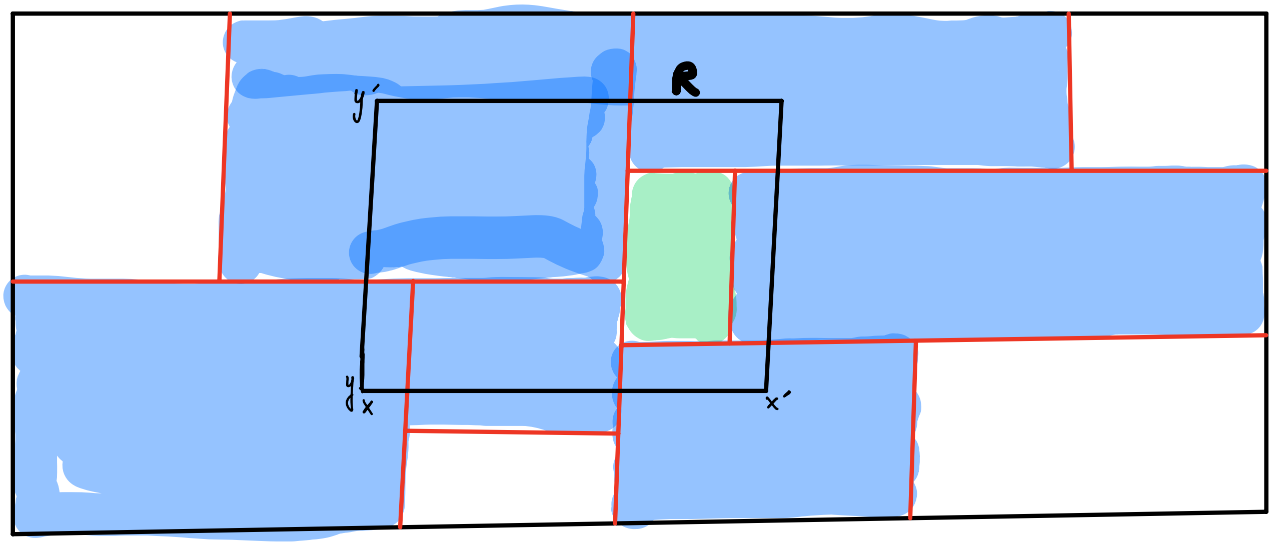
  left child

- Region ($rc(v)$) = Region(v) $\cap$ right(v)

  right child

Fact: a point p of P belongs to region(v) $\Longleftrightarrow$ p belongs to subtree under v.

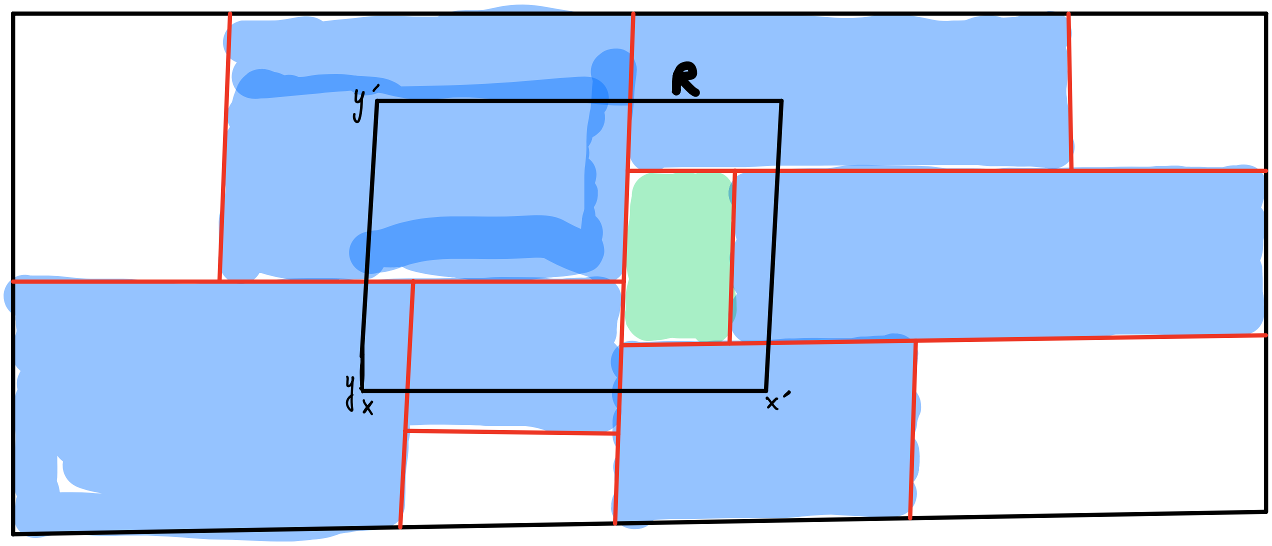
Search algorithm - idea



Idea of algorithm

- Want to find all points from P in R.
- At node v, can have

Search algorithm - idea

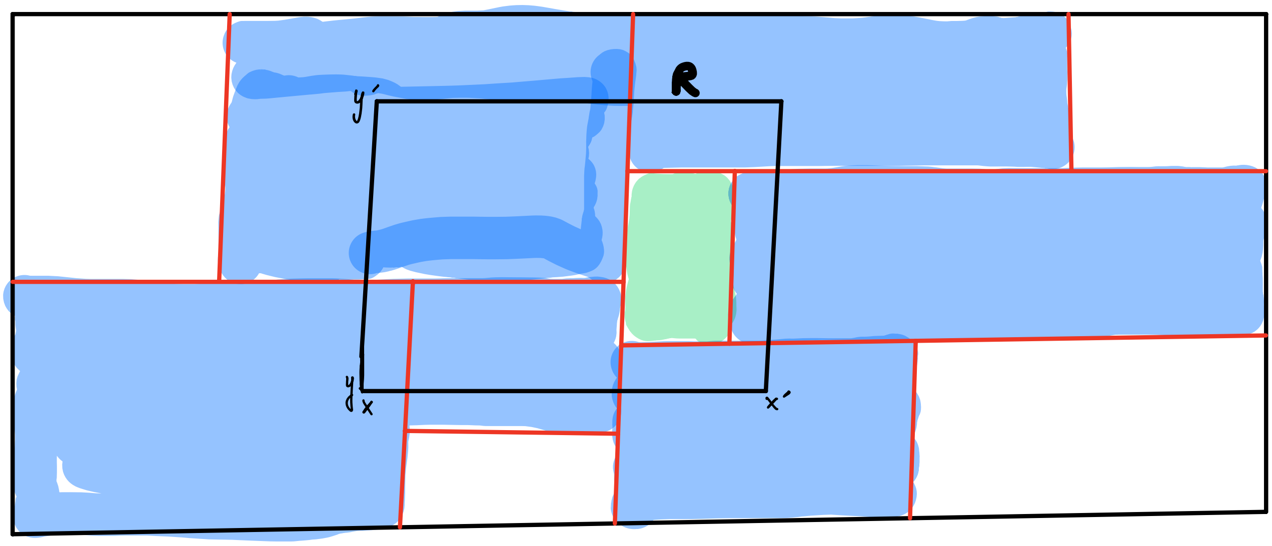


Idea of algorithm

- Want to find all points from P in R.
- At node v, can have

1) region(v) $\subseteq$ R (green above)

Search algorithm - idea



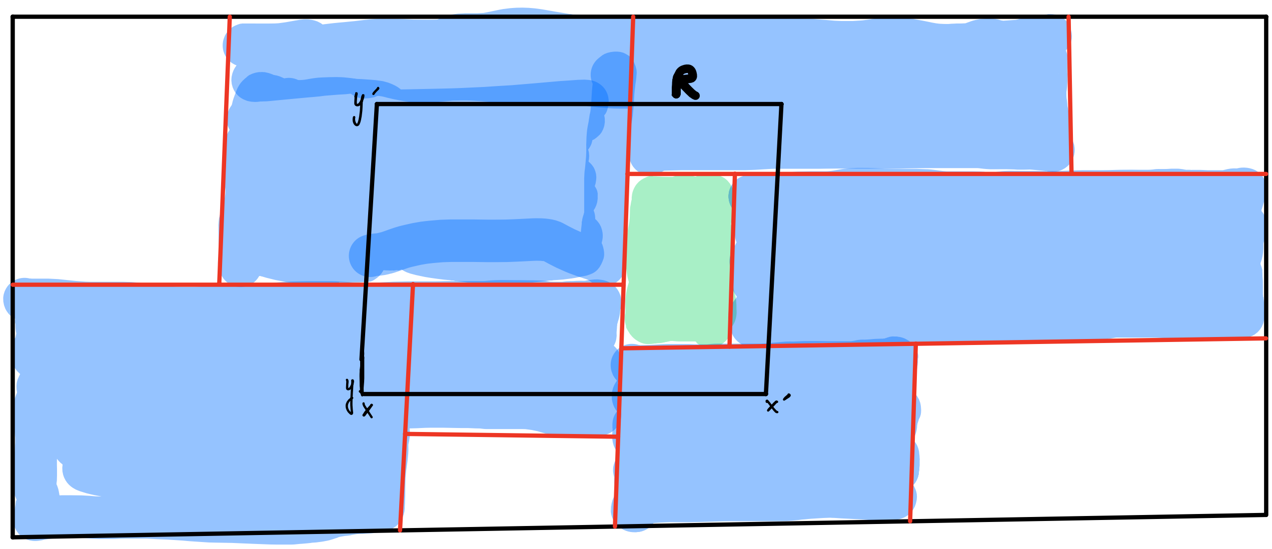
Idea of algorithm

- Want to find all points from P in R.
- At node v, can have

1) region(v) $\subseteq$ R (green above)

2) region(v) $\cap$ R is empty (white above)

Search algorithm - idea

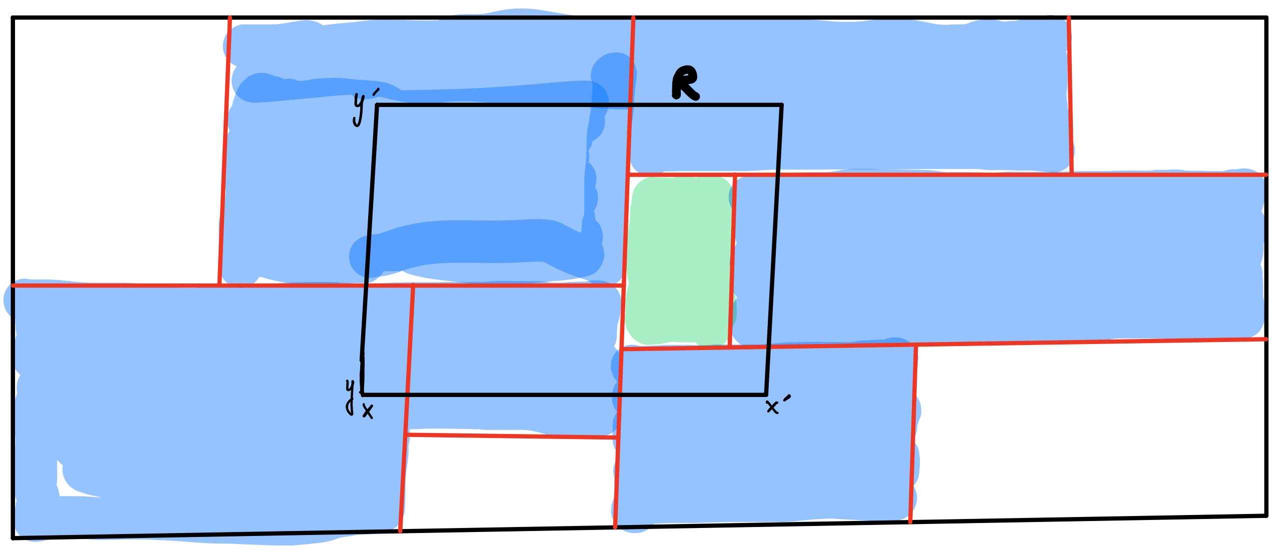


Idea of algorithm

- Want to find all points from P in R.
- At node v, can have

1) region(v) $\subseteq$ R (green above)
2) region(v) $\cap$ R is empty (white above)
3) region(v) $\cap$ R is nonempty but reg(v) $\not\subseteq$ R (blue above)

Search algorithm - idea



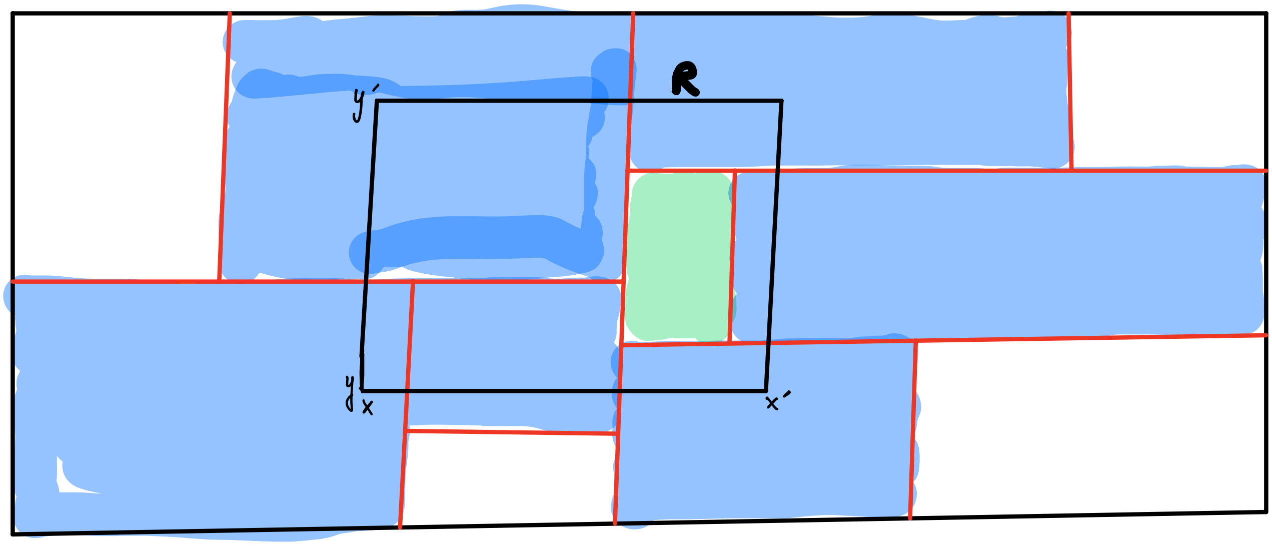
Idea of algorithm

- Want to find all points from P in R.
- At node v, can have

1) region(v) $\subseteq$ R (green above)
2) region(v) $\cap$ R is empty (white above)
3) region(v) $\cap$ R is nonempty but reg(v) $\nsubseteq$ R
   (blue above)

- In case 1, report all points in region.

Search algorithm - idea

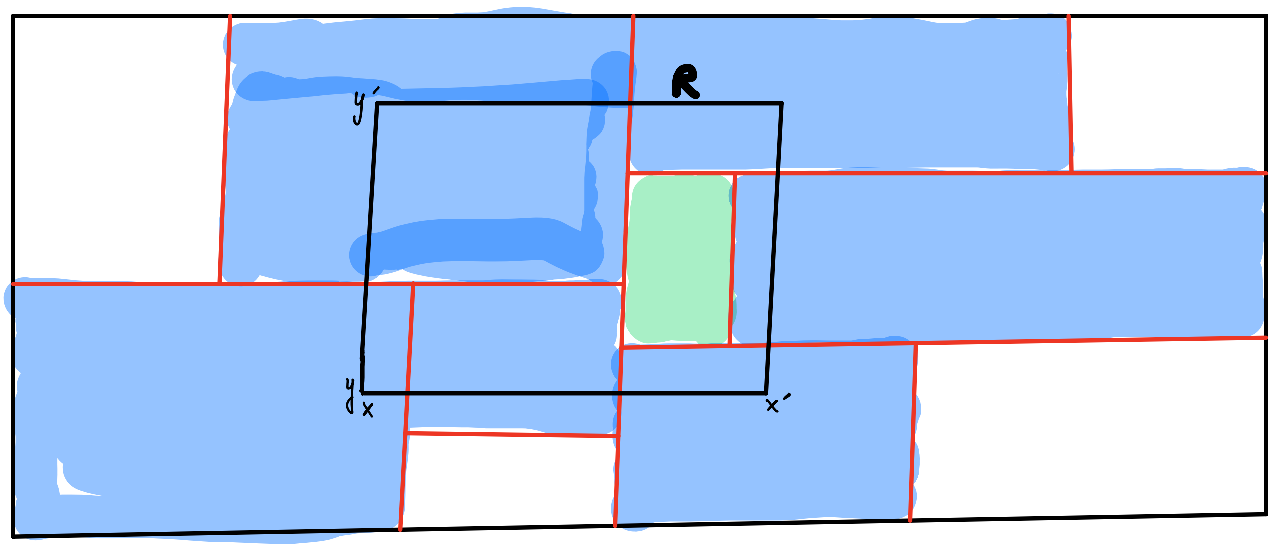


Idea of algorithm

- Want to find all points from P in R.
- At node v, can have
  1) region(v) $\subseteq$ R (green above)
  2) region(v) $\cap$ R is empty (white above)
  3) region(v) $\cap$ R is nonempty but reg(v) $\not\subseteq$ R (blue above)

- In case 1, report all points in region.
- In case 2, don't search below v.

Search algorithm - idea



Idea of algorithm

- Want to find all points from P in R.
- At node v, can have

1) $region(v) \subseteq R$ (green above)
2) $region(v) \cap R$ is empty (white above)
3) $region(v) \cap R$ is nonempty but $reg(v) \not\subseteq R$ (blue above)

- In case ①, report all points in region.
- In case ②, don't search below v.
- In case ③, search regions for left & right child of v.

Algorithm SearchKDtree(v, R)

Input: range R + KD-tree of points P, node v in KD-tree.

Output: leaves under v in range R

Algorithm SearchKDtree(v, R)

Input: range R + KD-tree of points P, node v in KD-tree.

Output: leaves under v in range R

- If v a leaf
  then report v if belongs to R

Algorithm SearchKDtree(v, R)

Input: range R + KD-tree of points P, node v in KD-tree.

Output: leaves under v in range R

- If v a leaf
  then report v if belongs to R

  else if region(lc(v)) ⊆ R,
  then report subtree of lc(v)

Algorithm SearchKDtree(v, R)

Input: range R + KD-tree of points P,
node v in KD-tree.

Output: leaves under v in range R

- If v a leaf
  then report v if belongs to R

  else if region(lc(v)) $\subseteq$ R,
  then report subtree of lc(v)

  else if region(lc(v)) intersects R
  then SearchKDtree(lc(v), R)

Algorithm SearchKDtree(v, R)

Input: range R + KD-tree of points P, node v in KD-tree.

Output: leaves under v in range R

- If v a leaf
    then report v if belongs to R

    else if region(lc(v)) ⊆ R,
        then report subtree of lc(v)

        else if region(lc(v)) intersects R
            then SearchKDtree(lc(v), R)

    if region(rc(v)) ⊆ R,
        then report subtree of rc(v)

Algorithm SearchKDtree(v, R)

Input: range R + KD-tree of points P, node v in KD-tree.

Output: leaves under v in range R

- If v a leaf
  then report v if belongs to R

  else if region(lc(v)) $\subseteq$ R,
  then report subtree of lc(v)

  else if region(lc(v)) intersects R
  then SearchKDtree(lc(v), R)

  if region(rc(v)) $\subseteq$ R,
  then report subtree of rc(v)

  else if region(rc(v)) intersects R
  then SearchKDtree(rc(v), R)

Algorithm SearchKDtree(v, R)

Input: range R + KD-tree of points P, node v in KD-tree.

Output: leaves under v in range R

- If v a leaf
  then report v if belongs to R

  else if $region(lc(v)) \subseteq R$,
  then report subtree of $lc(v)$

  else if $region(lc(v))$ intersects R
  then SearchKDtree($lc(v)$, R)

  if $region(rc(v)) \subseteq R$,
  then report subtree of $rc(v)$

  else if $region(rc(v))$ intersects R
  then SearchKDtree($rc(v)$, R)

Complexity

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

no of pts in P

no of solutions

Removing assumption that no points in P have same x or y-coordinate

Observation: did not need points to be real numbers - only needed them to be elements of a totally ordered set:

Removing assumption that no points in P have same x or y-coordinate

Observation: did not need points to be real numbers – only needed them to be elements of a totally ordered set:
to compare points & find medians

Removing assumption that no points in P have same x or y-coordinate

Observation : did not need points to be real numbers - only needed them to be elements of a totally ordered set : to compare points & find medians

- Pass from $\mathbb{R}$ to $\widetilde{C} = (\mathbb{R} \cup \{-\infty, \infty\})^2$

elements of form $(a \mid b)$

Removing assumption that no points in P have same x or y-coordinate

Observation: did not need points to be real numbers - only needed them to be elements of a totally ordered set: to compare points & find medians

- Pass from $\mathbb{R}$ to $C = (\mathbb{R} \cup \{-\infty, \infty\})^2$
  elements of form $(a|b)$
- C has lexicographic order:
  $(a|b) < (c|d) \iff a < c$ or ($a = c$ & $b < d$)

Removing assumption that no points in P have same x or y-coordinate

Observation: did not need points to be real numbers - only needed them to be elements of a totally ordered set:
to compare points & find medians

- Pass from $\mathbb{R}$ to $C = (\mathbb{R} \cup \{-\infty, \infty\})^2$
  elements of form $(a|b)$
- C has lexicographic order:
  $(a|b) < (c|d) \iff a < c$ or ($a = c$ & $b < d$)

$$\mathbb{R}^2 \longrightarrow C^2$$

$$(p_x, p_y) = p \longmapsto \hat{p} = ((p_x|p_y), (p_y|p_x))$$

Removing assumption that no points in P have same x or y-coordinate

Observation: did not need points to be real numbers - only needed them to be elements of a totally ordered set: to compare points & find medians

- Pass from $\mathbb{R}$ to $C = (\mathbb{R} \cup \{-\infty, \infty\})^2$ (elements of form $(a|b)$)
- C has lexicographic order:

$$(a|b) < (c|d) \iff a < c \text{ or } (a = c \ \& \ b < d)$$

$$\mathbb{R}^2 \longrightarrow C^2$$

$$(p_x, p_y) = p \longmapsto \hat{p} = ((p_x|p_y), (p_y|p_x))$$

Set $\hat{P} = \{\hat{p} : p \in P\}$.

Removing assumption that no points in P have same x or y-coordinate

Observation: did not need points to be real numbers - only needed them to be elements of a totally ordered set: to compare points & find medians

- Pass from $\mathbb{R}$ to $C = (\mathbb{R} \cup \{-\infty, \infty\})^2$
  elements of form $(a|b)$
- C has lexicographic order:
  $(a|b) < (c|d) \iff a < c$ or ($a = c$ & $b < d$)

$$\mathbb{R}^2 \longrightarrow C^2$$

$$(p_x, p_y) = p \longmapsto \hat{p} = ((p_x|p_y), (p_y|p_x))$$

Set $\hat{P} = \{\hat{p} : p \in P\}$.

No two points in $\hat{P}$ have same first or second coord.

Removing assumption that no points in P have same x or y-coordinate

Observation: did not need points to be real numbers - only needed them to be elements of a totally ordered set: to compare points & find medians

- Pass from $\mathbb{R}$ to $C = (\mathbb{R} \cup \{-\infty, \infty\})^2$

  elements of form $(a \mid b)$

- C has lexicographic order:

  $(a \mid b) < (c \mid d) \iff a < c$ or $(a = c$ & $b < d)$

$$\mathbb{R}^2 \longrightarrow C^2$$

$$(p_x, p_y) = p \longmapsto \hat{p} = ((p_x \mid p_y), (p_y \mid p_x))$$

- Set $\hat{P} = \{\hat{p} : p \in P\}$.
- No two points in $\hat{P}$ have same first or second coord.

- At $R = [x, x'] \times [y, y']$ let

$$\hat{R} = [(x \mid -\infty), (x' \mid \infty)] \times [(y \mid -\infty), (y' \mid \infty)]$$

Removing assumption that no points in P have same x or y-coordinate

Observation: did not need points to be real numbers - only needed them to be elements of a totally ordered set: to compare points & find medians

- Pass from $\mathbb{R}$ to $C = (\mathbb{R} \cup \{-\infty, \infty\})^2$

elements of form $(a|b)$

- C has lexicographic order:

$$(a|b) < (c|d) \iff a<c \text{ or } (a=c \ \& \ b<d)$$

$$\mathbb{R}^2 \longrightarrow C^2$$

$$(p_x, p_y) = p \longmapsto \hat{p} = ((p_x|p_y), (p_y|p_x))$$

- Set $\hat{P} = \{\hat{p} : p \in P\}$.
- No two points in $\hat{P}$ have same first or second coord

- At $R = [x, x'] \times [y, y']$ let

$$\hat{R} = [(x|-\infty), (x'|\infty)] \times [(y|-\infty), (y'|\infty)]$$

- Then $p \in R \iff \hat{p} \in \hat{R}$

i.e. $(p_x|p_y) \in [(x|-\infty), (x'|\infty)]$

$\iff (x|-\infty) < (p_x|p_y) < (x'|\infty)$.

First ineq. $x < p_x$ or $x = p_x$ ~ $x \leq p_x$

Second ineq. $p_x \leq x'$.

Removing assumption that no points in P have same x or y-coordinate

Observation: did not need points to be real numbers - only needed them to be elements of a totally ordered set: to compare points & find medians

- Pass from $\mathbb{R}$ to $C = (\mathbb{R} \cup \{-\infty, \infty\})^2$
  elements of form $(a|b)$
- C has lexicographic order:
  $(a|b) < (c|d) \iff a<c$ or ($a=c$ & $b<d$)

$$\mathbb{R}^2 \longrightarrow C^2$$

$$(p_x, p_y) = p \longmapsto \hat{p} = ((p_x|p_y), (p_y|p_x))$$

- Set $\hat{P} = \{\hat{p} : p \in P\}$.
- No two points in $\hat{P}$ have same first or second coord.

- At $R = [x,x'] \times [y,y']$ let

$$\hat{R} = [(x|-\infty), (x'|\infty)] \times [(y|-\infty), (y'|\infty)]$$

- Then $p \in R \iff \hat{p} \in \hat{R}$ so only need to run our original alg (gen. to a totally ord. set) on $(\hat{P}, \hat{R})$ instead

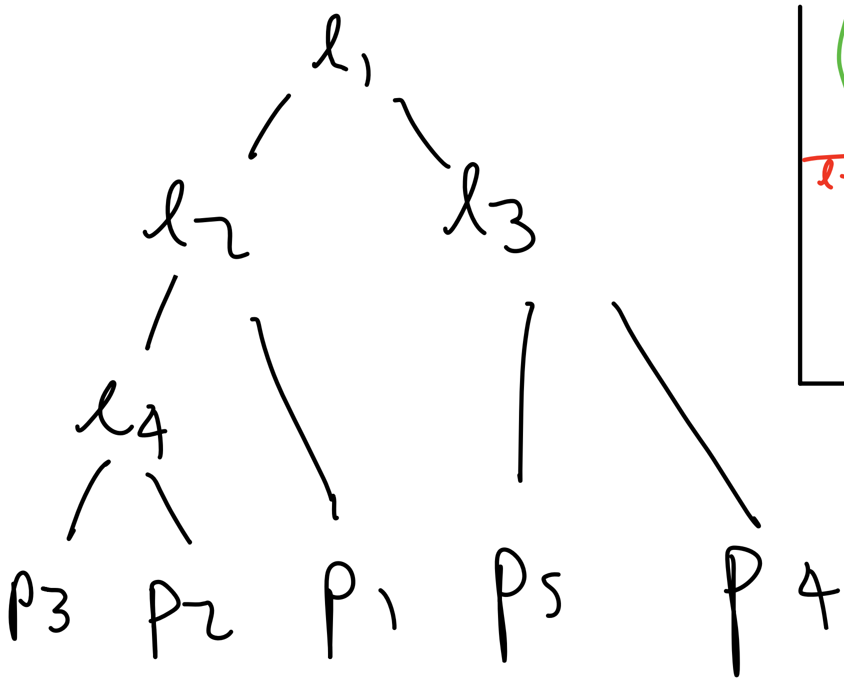
i.e. $(p_x|p_y) \in [(x|-\infty), (x'|\infty)]$

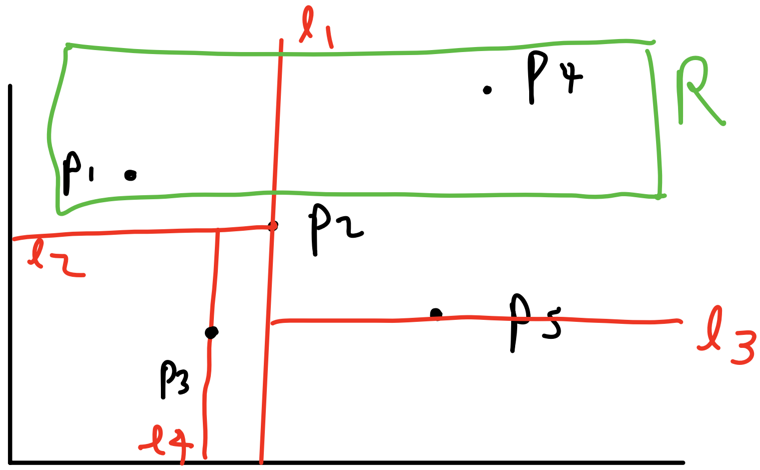
$\iff (x|-\infty) < (p_x|p_y) < (x'|\infty)$.

First ineq. $x < p_x$ or $x = p_x$ ~ $x \le p_x$

Second ineq. $p_x \le x'$.

Example





- At l_1, look at l_2, l_3

→ Both $reg(l_2), reg(l_3) \cap R$, but not cont. in

At l_2, look at l_4, p_1.

- $reg(l_4) \cap R = \emptyset$.
- $p_1 \in R \Rightarrow$ report p_1.

At l_3, look at p_5, p_4.

$p_4 \in R$.

Second approach - range trees

Idea: Given $P \subseteq \mathbb{R}^2$ & $R = [x, x'] \times [y, y']$

Second approach - range trees

Idea: Given $P \subseteq \mathbb{R}^2$ & $R = [x, x'] \times [y, y']$

① Use a 1-d search to find points of P whose x-coord belongs to $[x, x']$.

Second approach - range trees

Idea: Given $P \subseteq \mathbb{R}^2$ & $R = [x, x'] \times [y, y']$

① Use a 1-d search to find points of P whose x-coord belongs to $[x, x']$.

② Search amongst those points to find those whose y-coord belongs to $[y, y']$.

Second approach - range trees

Idea: Given $P \subseteq \mathbb{R}^2$ & $R = [x, x'] \times [y, y']$

① Use a 1-d search to find points of P whose x-coord belongs to $[x, x']$.

② Search amongst these points to find those whose y-coord belongs to $[y, y']$.

Data structure : range tree.

- A binary tree whose leaves are elements of P, ordered by x-coord (assume no 2 pts have same x or y coord.)

Second approach - range trees

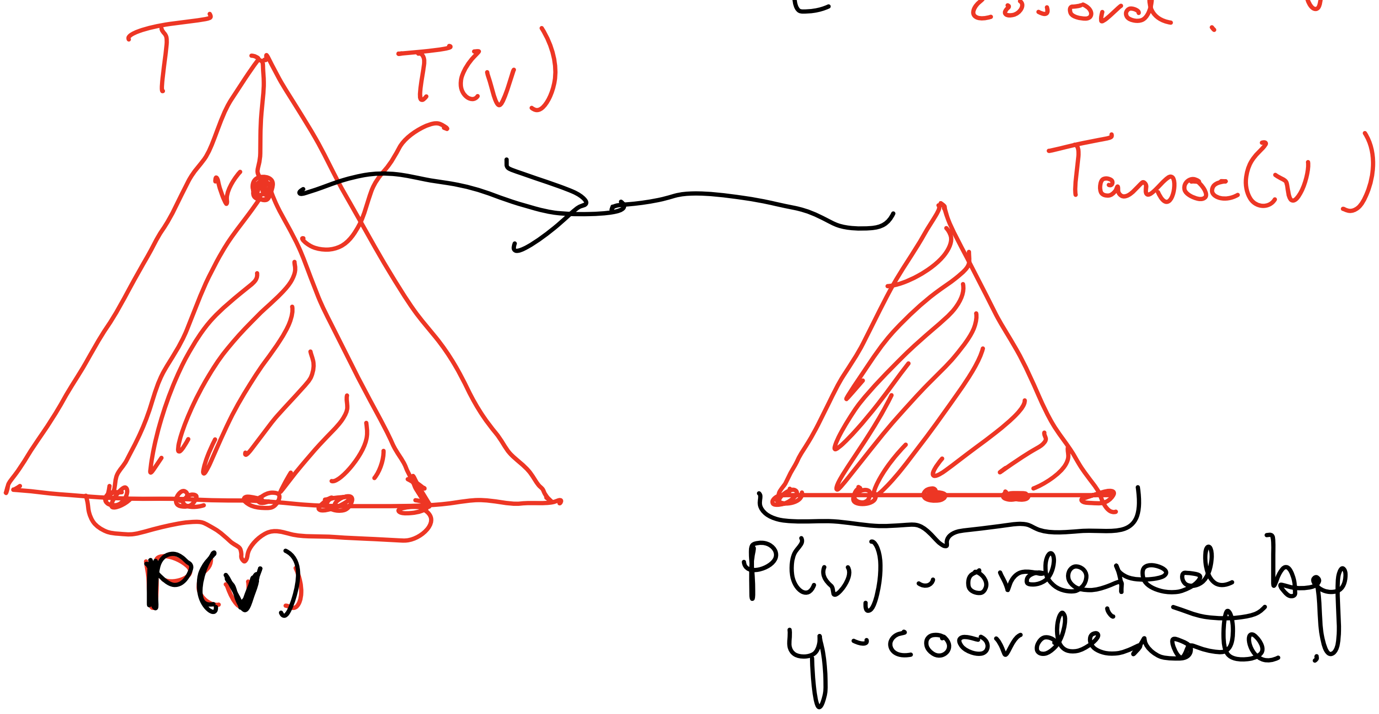
Idea: Given $P \subseteq \mathbb{R}^2$ & $R = [x, x'] \times [y, y']$

① Use a 1-d search to find points of P whose x-coord belongs to $[x, x']$.

② Search amongst those points to find those whose y-coord belongs to $[y, y']$.

Data structure : range tree.

- A binary tree whose leaves are elements of P, ordered by x-coord (assume no 2 pts have same x or y coord.)



Second approach - range trees

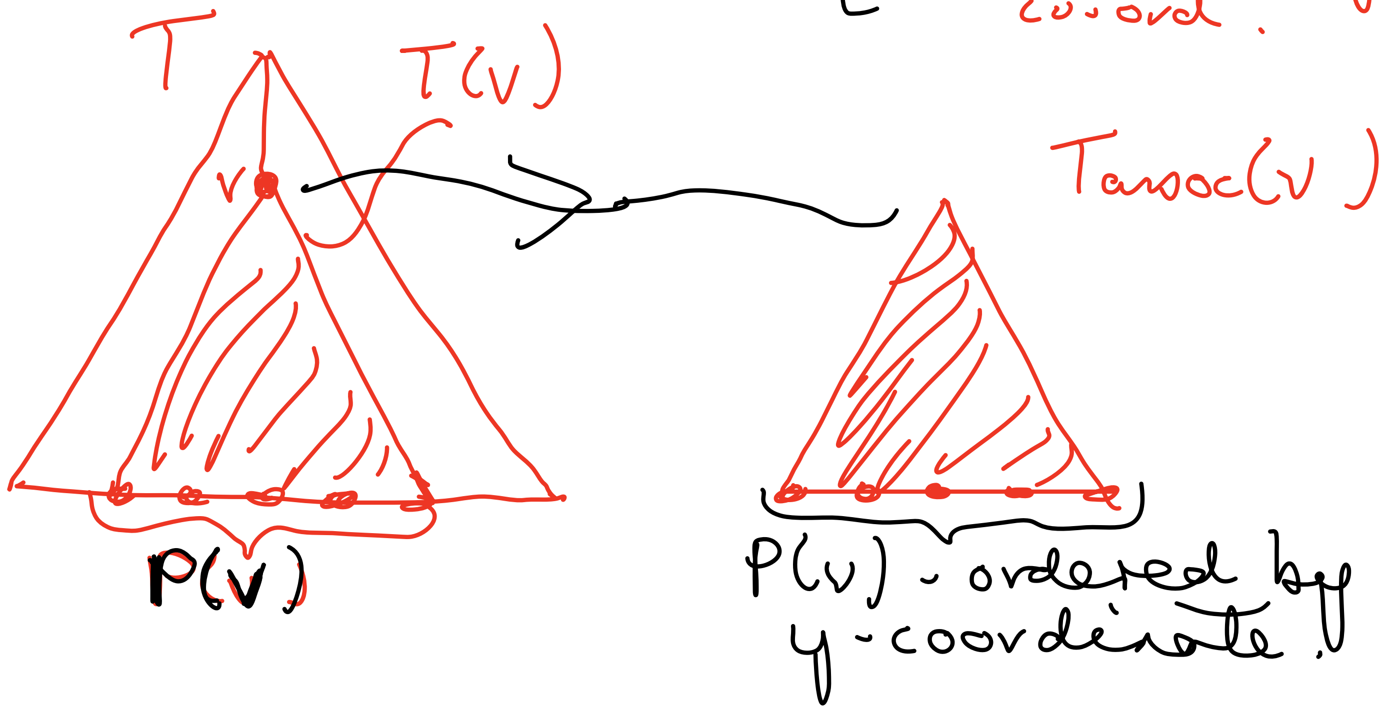
Idea: Given $P \subseteq \mathbb{R}^2$ & $R = [x, x'] \times [y, y']$

① Use a 1-d search to find points of P whose x-coord belongs to $[x, x']$.

② Search amongst these points to find those whose y-coord belongs to $[y, y']$.

Data structure: range tree.

- A binary tree whose leaves are elements of P, ordered by x-coord (assume no 2 pts have same x or y coord.)



- Each node v determines subtree $T(v)$ with set of leaves $P(v)$; For each such node we have another bin. tree $T_{assoc}(v)$ with leaves $P(v)$ ordered **by** y-coordinate

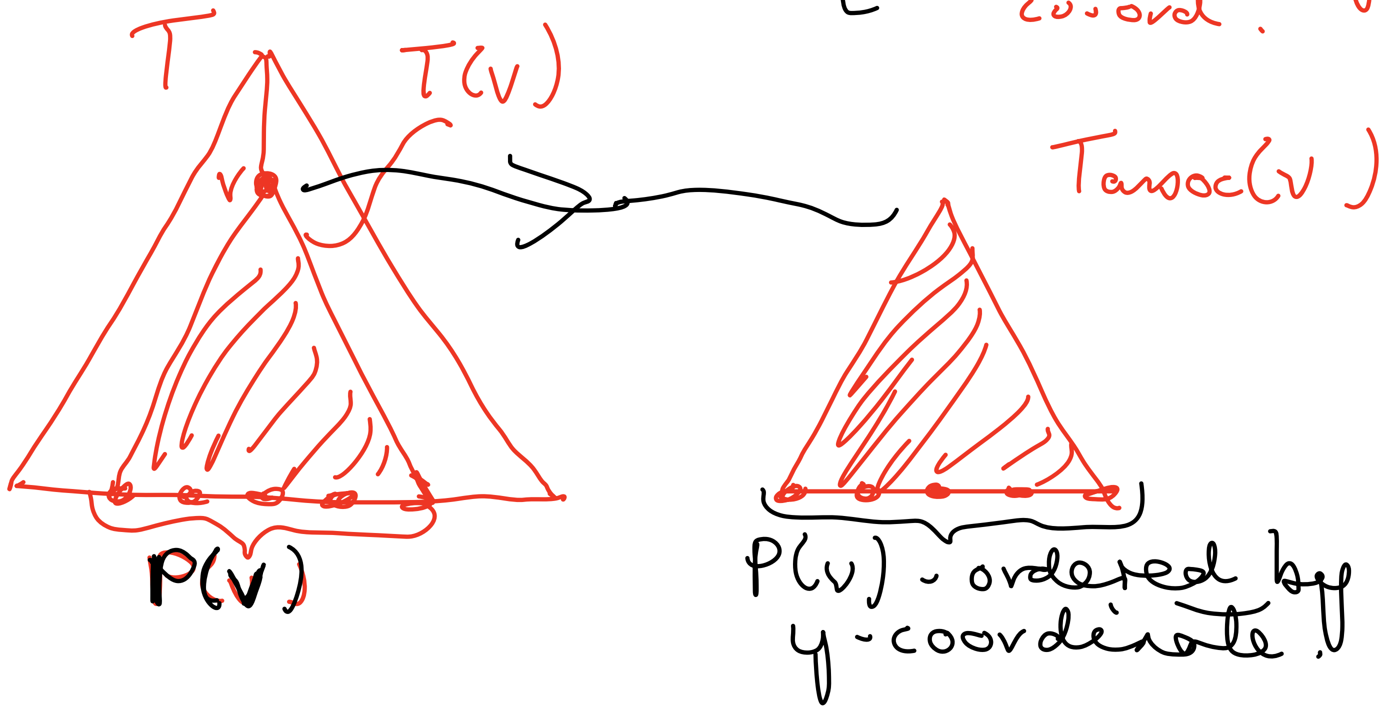
Second approach - range trees

Idea: Given $P \subseteq \mathbb{R}^2$ & $R = [x,x'] \times [y,y']$

1. Use a 1-d search to find points of P whose x-coord belongs to $[x,x']$.
2. Search amongst these points to find those whose y-coord belongs to $[y,y']$.

Data structure: range tree.

- A binary tree whose leaves are elements of P, ordered by x-coord (assume no 2 pts have same x or y coord.)



- Each node v determines subtree $T(v)$ with set of leaves $P(v)$; For each such node we have another bin. tree $T_{assoc}(v)$ with leaves $P(v)$ ordered by y-coordinate
- Storage $O(n \log n)$

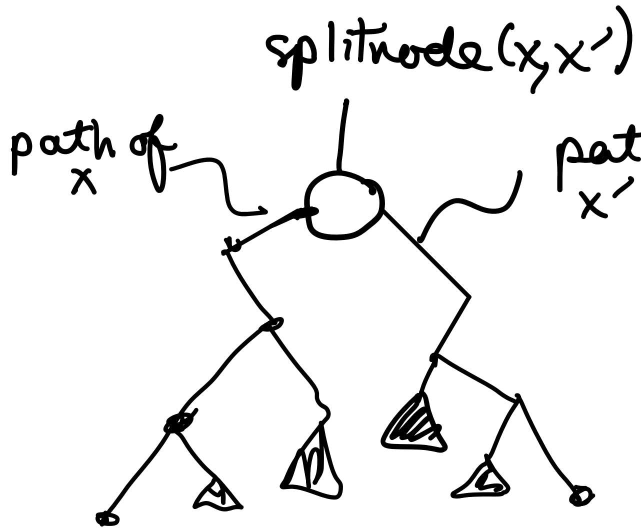
Searching a range tree

$R = [x, x'] \times [y, y']$

Searching a range tree

$R = [x, x'] \times [y, y']$

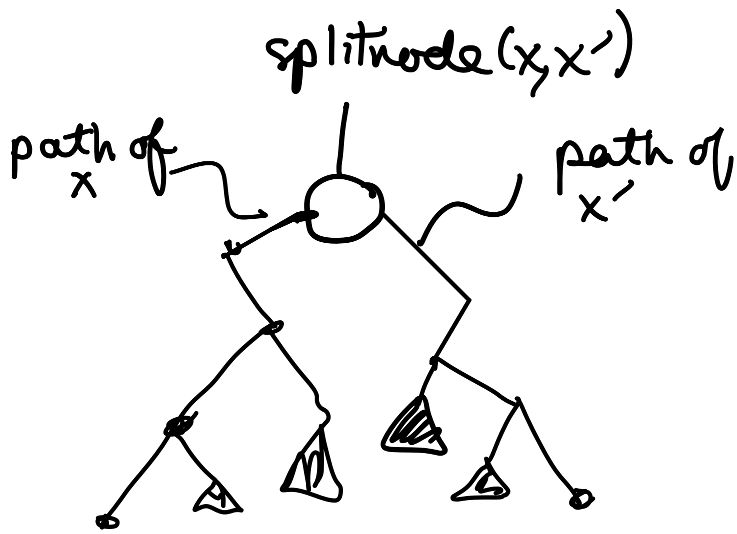
- Look at tree ordered by x-coord, find split node of x & x'



Searching a range tree

$R = [x, x'] \times [y, y']$

- Look at tree ordered by x-coord, find split node of x & x'

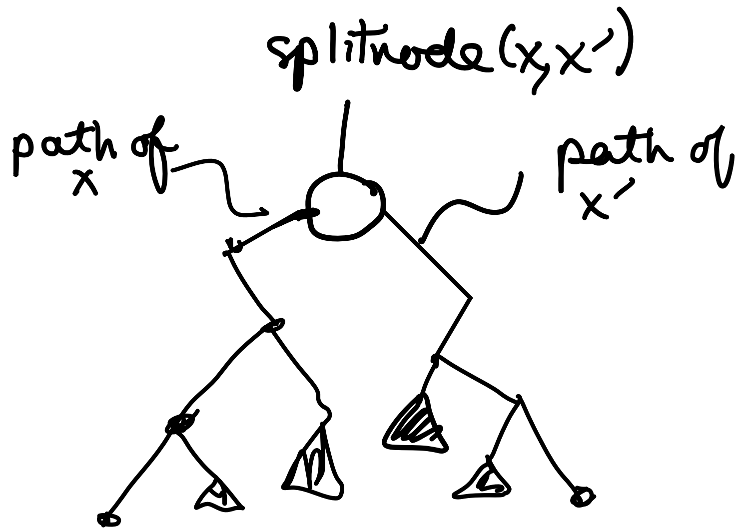


- If path for x moves left at v, each leaf in right subtree belongs to $[x, x']$

Searching a range tree

$R = [x, x'] \times [y, y']$

- Look at tree ordered by x-coord, find split node of x & x'



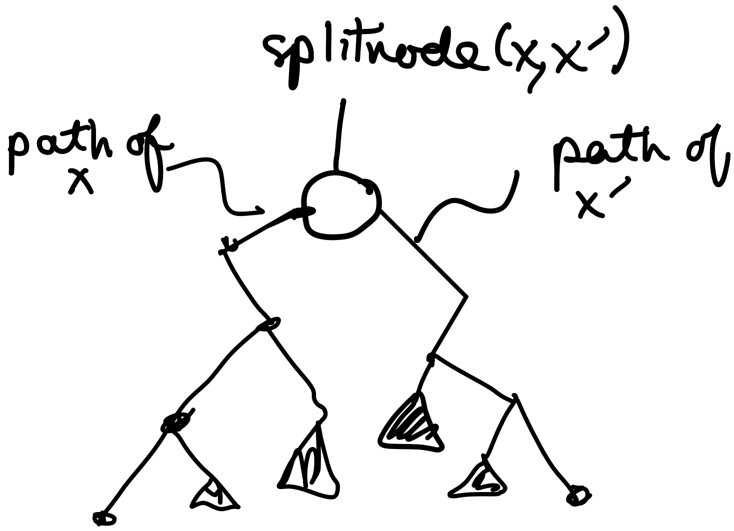
- If path for x moves left at v, each leaf in right subtree belongs to $[x, x']$

Then we use a 1-d range search on $T_{assoc}(rc(v))$ to find those whose y-coord belongs to $[y, y']$.

Searching a range tree

$R = [x, x'] \times [y, y']$

- Look at tree ordered by x-coord, find split node of x & x'



- If path for x moves left at v, each leaf in right subtree belongs to [x,x']

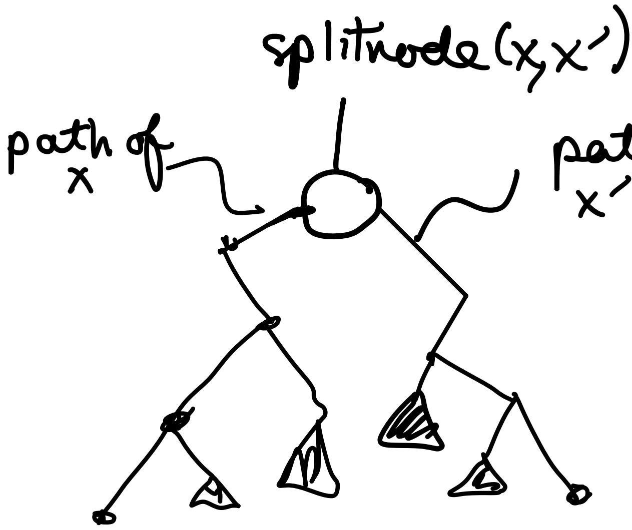
  Then we use a 1-d range search on Tassoc(rc(v)) to find those whose y-coord belongs to [y,y'].

- If v is a leaf, test whether it belongs to R.

Searching a range tree

$R = [x, x'] \times [y, y']$

- Look at tree ordered by x-coord, find split node of x & x'



- If path for x moves left at v, each leaf in right subtree belongs to $[x, x']$

  Then we use a 1-d range search on $T_{assoc}(rc(v))$ to find those whose y-coord belongs to $[y, y']$.

- If v is a leaf, test whether it belongs to R.

- Similarly search path of x' below split node.

Complexity

$O(\log n^2 + k)$

no. of points

k no. of solutions

Complexity

$O(\underbrace{\log n}_{\text{no. of points}}{}^2 + \underbrace{k}_{k \text{ no. of solutions}})$

- Both kd-trees & range trees can be gen. to higher dimensions. See E-Learning for this.

Complexity

$O(\log n^2 + K)$

n: no. of points; K: no. of solutions

- Both kd-trees & range trees can be gen. to higher dimensions. See E-Learning for this.
- **kd-trees require less storage space but are slower to search; range trees require more storage but are faster to search.**

Complexity

$O(\log n^2 + k)$

no. of points — k no. of solutions

- Both kd-trees & range trees can be gen. to higher dimensions. See E-Learning for this.
- kd-trees require less storage space but are slower to search; range trees require more storage but are faster to search:

Comparison in d dimensions

	kd-trees	range trees
Storage	$O(n)$	$O(n \log^{d-1} n)$
Construction	$O(n \log n)$	$O(n \log^{d-1} n)$
Search	$O(n^{1-1/d} + k)$	$O(n \log^{d} n + k)$