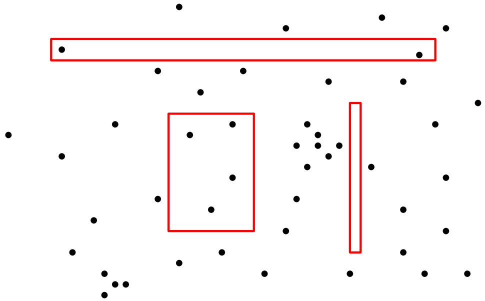
Range queries in 2D



Range queries in 2D

Question: Why can't we simply use a balanced binary tree in x-coordinate?

Or, use one tree on x-coordinate and one on y-coordinate, and query the one where we think querying is more efficient?

Kd-trees

Kd-trees, the idea: Split the point set alternatingly by x-coordinate and by y-coordinate

split by x-coordinate: split by a vertical line that has half the points left and half right

split by y-coordinate: split by a horizontal line that has half the points below and half above

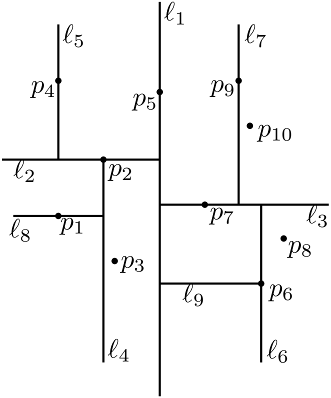
Kd-trees

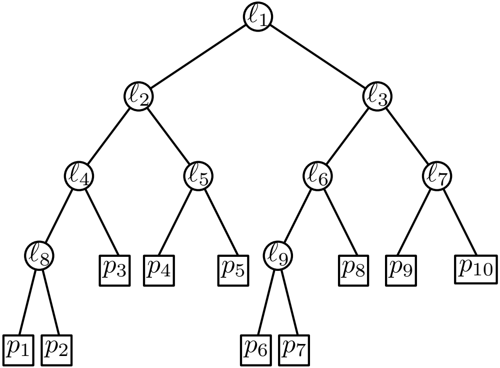
Kd-trees, the idea: Split the point set alternatingly by x-coordinate and by y-coordinate

split by x-coordinate: split by a vertical line that has half the points left or on, and half right

split by y-coordinate: split by a horizontal line that has half the points below or on, and half above

Kd-trees


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


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

Kd-tree construction

Algorithm $\textsc{BuildKdTree}(P, depth)$

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 with a vertical line ℓ through the median x-coordinate into P_1 (left of or on ℓ) and P_2 (right of ℓ)
5. **else** Split P with a horizontal line ℓ through the median y-coordinate into P_1 (below or on ℓ) and P_2 (above ℓ)
6. $v_{\text{left}} \leftarrow \textsc{BuildKdTree}(P_1, depth + 1)$
7. $v_{\text{right}} \leftarrow \textsc{BuildKdTree}(P_2, depth + 1)$
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

Kd-tree construction

The median of a set of n values can be computed in $O(n)$ time (randomized: easy; worst case: much harder)

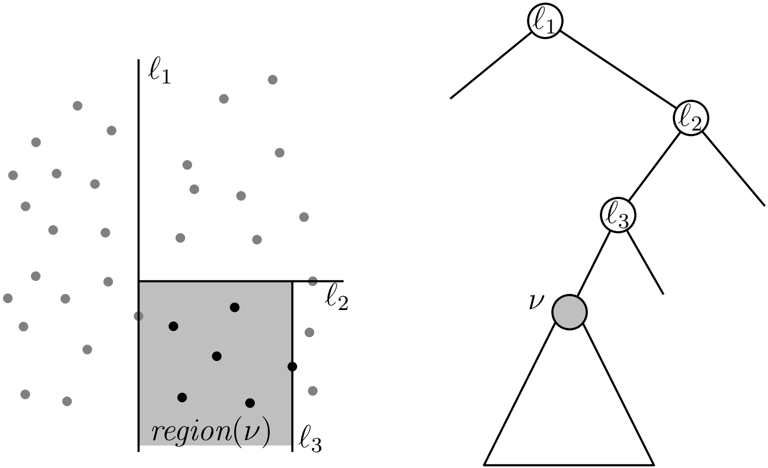
Let $T(n)$ be the time needed to build a kd-tree on n points

$$T(1) = O(1)$$
$$T(n) = 2 \cdot T(n/2) + O(n)$$

A kd-tree can be built in $O(n \log n)$ time

Question: What is the storage requirement?

Kd-tree regions of nodes



Kd-tree regions of nodes

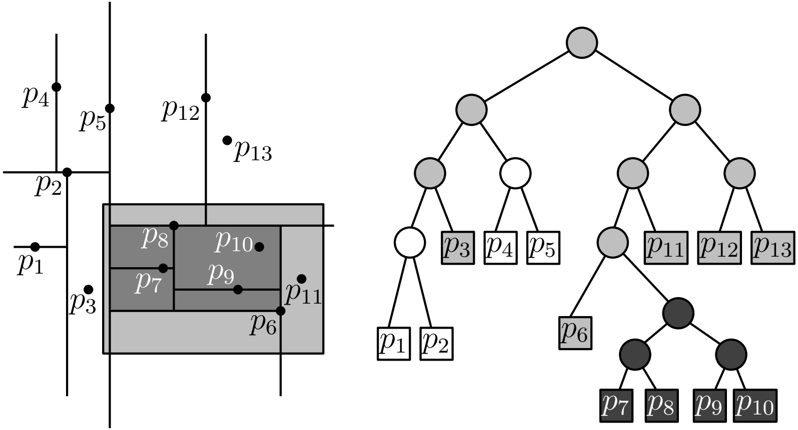
How do we know $region(v)$ when we are at a node v?

Option 1: store it explicitly with every node

Option 2: compute it on-the-fly, when going from the root to v

Question: What are reasons to choose one or the other option?

Kd-tree querying



Kd-tree querying

Algorithm $\textsc{SearchKdTree}(\nu, R)$
Input. The root of (a subtree of) a kd-tree, and a range R
Output. All points at leaves below ν that lie in the range.

if ν is a leaf
    then Report the point stored at ν if it lies in R
    else **if** $region(lc(\nu))$ is fully contained in R
        then $\textsc{ReportSubtree}(lc(\nu))$
        else **if** $region(lc(\nu))$ intersects R
            then $\textsc{SearchKdTree}(lc(\nu), R)$
        if $region(rc(\nu))$ is fully contained in R
        then $\textsc{ReportSubtree}(rc(\nu))$
        else **if** $region(rc(\nu))$ intersects R
            then $\textsc{SearchKdTree}(rc(\nu), R)$

Kd-tree querying

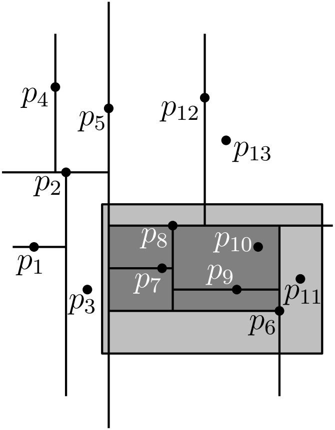
Question: How about a range *counting* query?
How should the code be adapted?

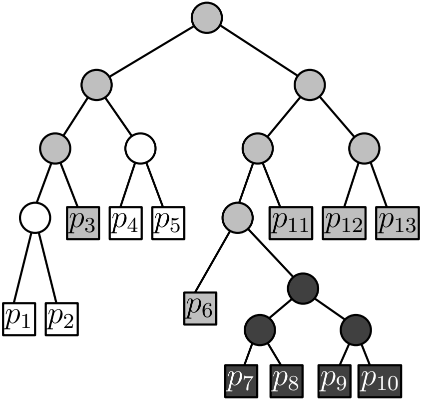
Kd-tree query time analysis

To analyze the query time of kd-trees, we use the concept of white, grey, and black nodes

- **White nodes:** never visited by the query; no time spent
- **Grey nodes:** visited by the query, unclear if they lead to output; time determines dependency on n
- **Black nodes:** visited by the query, whole subtree is output; time determines dependency on k, the output size

Kd-tree query time analysis





Kd-tree query time analysis

White, grey, and black nodes with respect to $region(v)$:

- **White node v:** R does not intersect $region(v)$
- **Grey node v:** R intersects $region(v)$, but $region(v) \not\subseteq R$
- **Black node v:** $region(v) \subseteq R$