Orthogonal range searching

Querying a Database

S.Fahimeh Moosavi

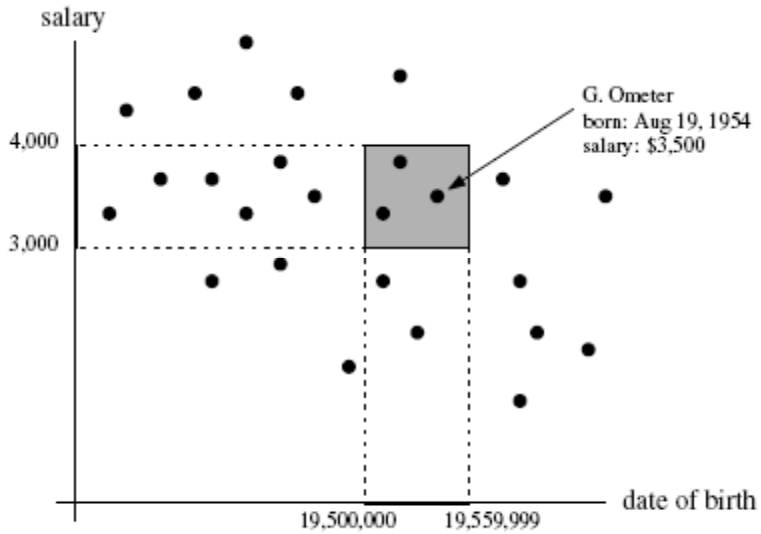
1388-1389

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

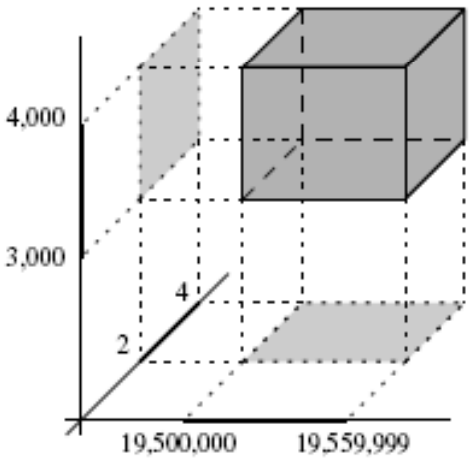
Example:

Database for personnel administration

Query: report all employees born between 1950 and 1955 who earn between $3000 and $4000 a month.



Query: report all employees born between 1950 and 1955 who earn between $3000 and $4000 a month and have between two and four children.



1-dimensional range searching

- The data we are given is a set of points in 1-dimensional space (a set of real number)
- A query asks for the points inside a 1-dimensional query rectangle (an interval [x : x′])

  P := {p_1 , p_2 , … , p_n}: the given set of points on the real line.

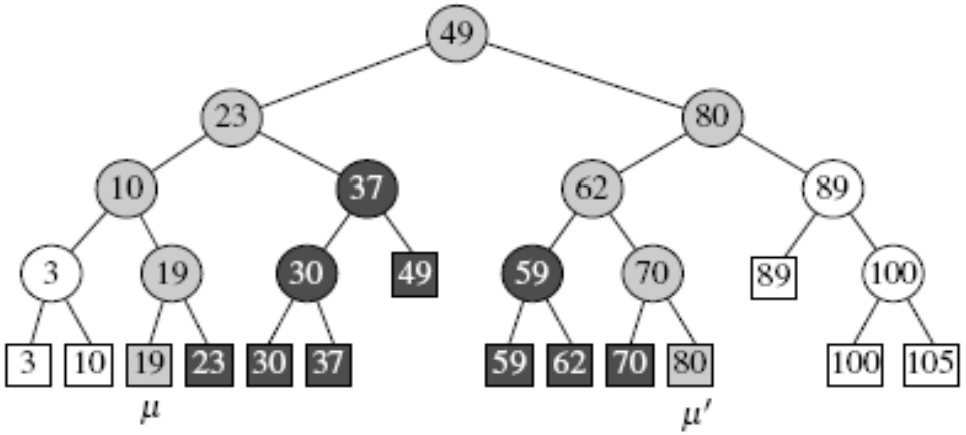
  We can solve 1-dimensional range searching problem efficiently using a balanced binary search tree *T*. the leaves of *T* store the points of P and the internal nodes of *T* store splitting value to guide the search.

Report the points in a query range [x : x′]

- Find the split node
- Continue searching for *x, report all right-subtrees*
- Continue searching for *x′, report all left-subtrees*
- When leaves μ and μ′ are reached, check if they belong to the range.

Example

Interval [18 : 77]

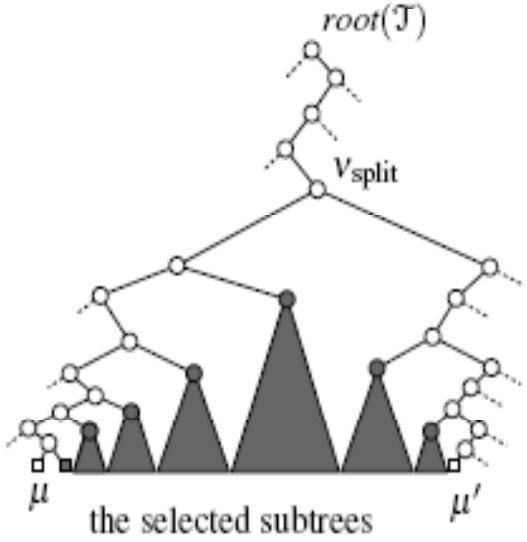


Report all points stored in the dark grey leaves plus the point stored in the leaf μ .

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 ← root(T)
2.  while v is not a leaf and (x' ⩽ x_v or x > x_v)
3.      do if x' ⩽ x_v
4.             then v ← lc(v)
5.             else  v ← rc(v)
6.  return v
```

V_{split} : where the paths to x and x′ split.



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.
```

Lemma 5.1 Algorithm 1DRANGEQUERY reports exactly those points that lie in the query range.

Theorem 5.2 let P be a set of n points in 1-dimensional space. The set P can be stored in a balanced binary search tree, which uses O(n) storage and has O(n log n) construction time, Such that the points in a query range can be reported in time O(k + log n). Where k is the number of reported points.

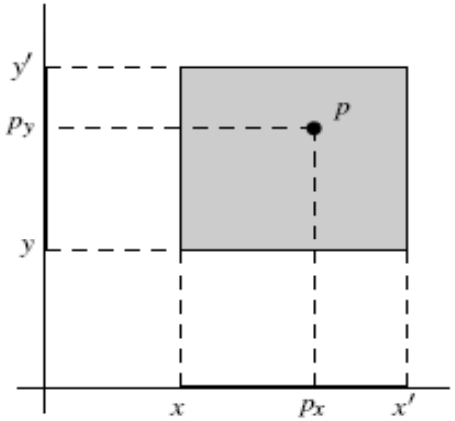
Proof: two traversals down the tree (because *T* is balanced, these path have length O(logn)) plus the *O(k).*

Kd - trees

P: A set of n points in the plane.

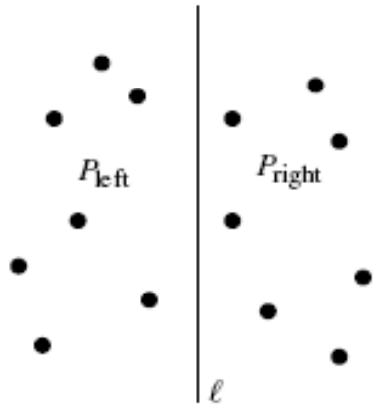
Assumption: no two points have the same *x-coordinate* (the same is true for *y-coordinate)*

- A 2-dimensional rectangular range query on P asks for the points from P lying inside a query rectangle $[x : x'] \times [y : y']$.
- A point $p:=(p_x , p_y)$ lies inside this rectangle if and only if $p_x \in [x : x']$ and $p_y \in [y : y']$.



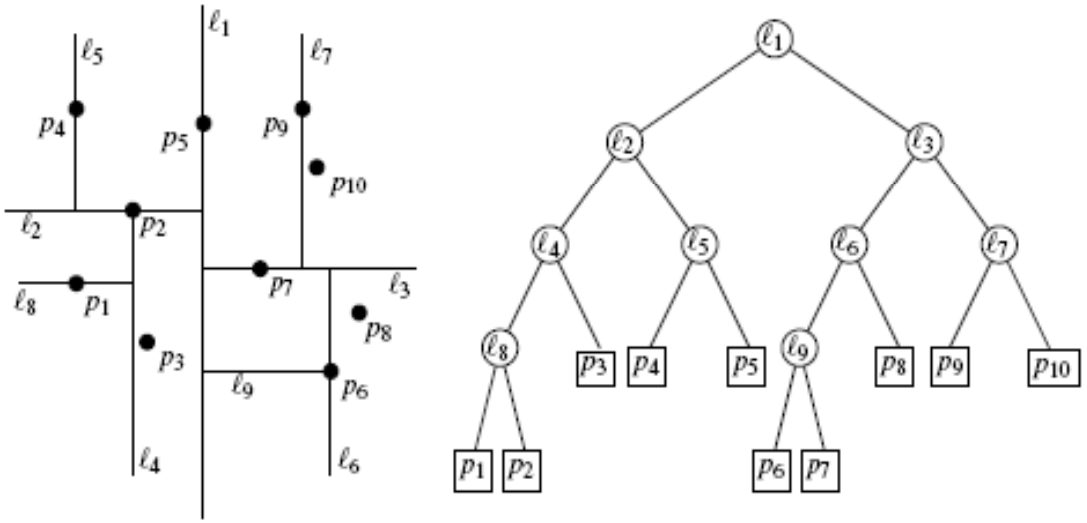
Recursive definition of the binary search tree:

The set of (1-dimensional) points is split into two subsets of roughly equal size; one subset contains the points smaller than or equal to the splitting value, the other subset contains the points larger than the splitting value.



Kd-tree: a binary tree

- Data points stored at leaves
- For each internal node ν *:*
  - ✓ *x-coordinates of left subtree* $\leq \nu <$ *x-coordinates of right* subtree, if depth of ν *is even (split with vertical line)*
  - ✓ *y-coordinates of left subtree* $\leq \nu <$ *y-coordinates of right* subtree, if depth of ν *is odd (split with horizontal line)*



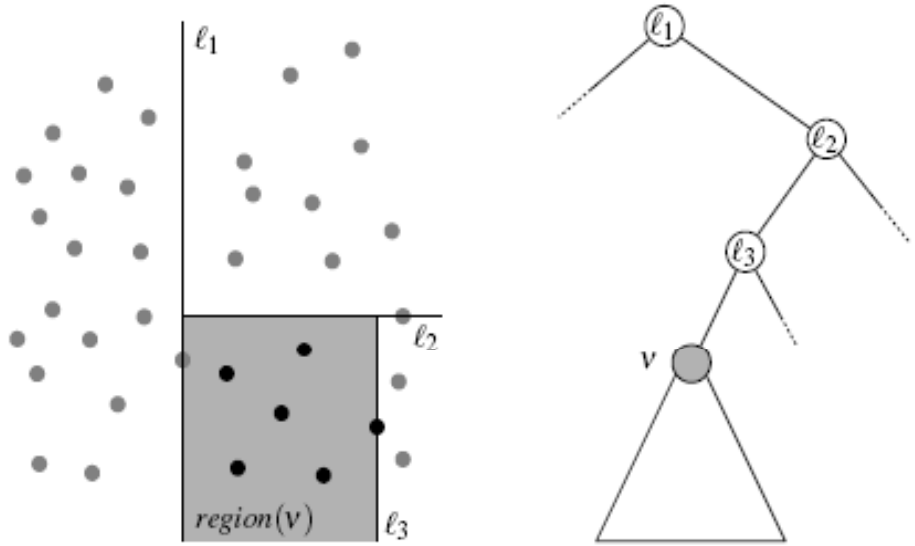
Algorithm BUILDKDTREE($P, depth$)
Input. A set of points P and the current depth *depth*.
Output. The root of a kd-tree storing P.

```
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 P1 be the set of points to the left of ℓ or on ℓ, and let
                    P2 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 P1 be the set of points below ℓ or on ℓ,
                    and let P2 be the set of points above ℓ.
6.          v_left ← BUILDKDTREE(P1, depth + 1)
7.          v_right ← BUILDKDTREE(P2, 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
```

Lemma 5.3 A Kd-tree for a set of n points uses O(n) storage and can be constructed in O(n log n) time.

Correspondence between nodes in a kd-tree and regions in the plane

region(ν): the region corresponding to a node ν.

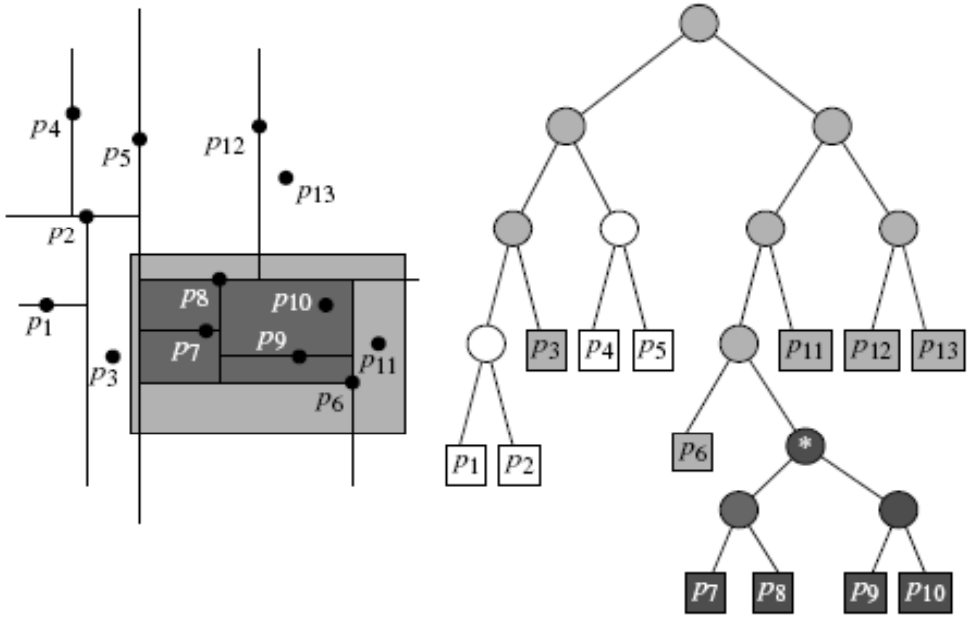


a point is stored in the subtree rooted at a node ν if and only if it lies in region(ν)

query the kd-tree

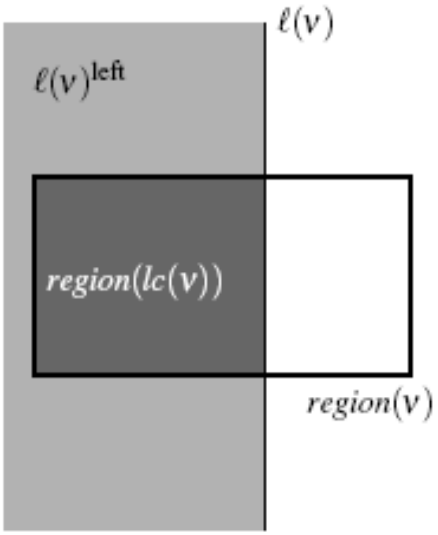
The range query algorithm (range *R):*

- If *region(v) does not intersect R, do not go* deeper into the subtree rooted at v
- If *region(v) is fully contained in R, report all* points in the subtree rooted at v
- If *region(v) only intersects with R, go* recursively into v's *children*



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)
```



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

Lemma 5.4 A query with an axis-parallel rectangle in a kd-tree storing n points can be performed in $O(\sqrt{n} + k)$ time, where k is the number of reported points.

Theorem 5.5 A kd-tree for a set P of n points in the plane uses O(n) storage and can be built in O(n log n) time. A rectangular range query on the kd-tree takes $O(\sqrt{n} + k)$ time, where k is number of reported points.

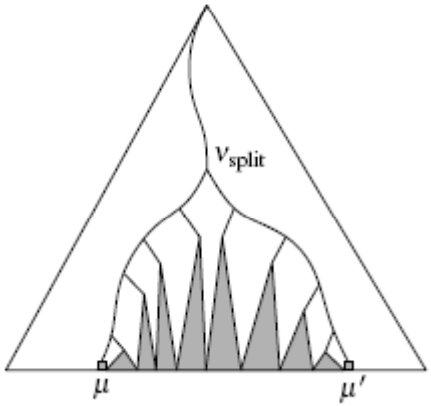
kd-trees can also be used for points sets in 3- or higher-dimensional space, query time is bounded by $O(n^{1-1/d} + k)$.

Range trees

when performing search on x-coordinate, start filtering points on y-coordinate

Canonical subset of ν ($P(\nu)$): the subset of points stored in the leaves of the subtree rooted at node ν.

- select a collection of O(logn) subtrees that together contain exactly the points whose x-coordinate lies in the x-interval of the query rectangle (disjoint union of O(logn) canonical subsets)



- Report the points in such a canonical subset P(ν) whose y-coordinate lies in the interval [y : y′].

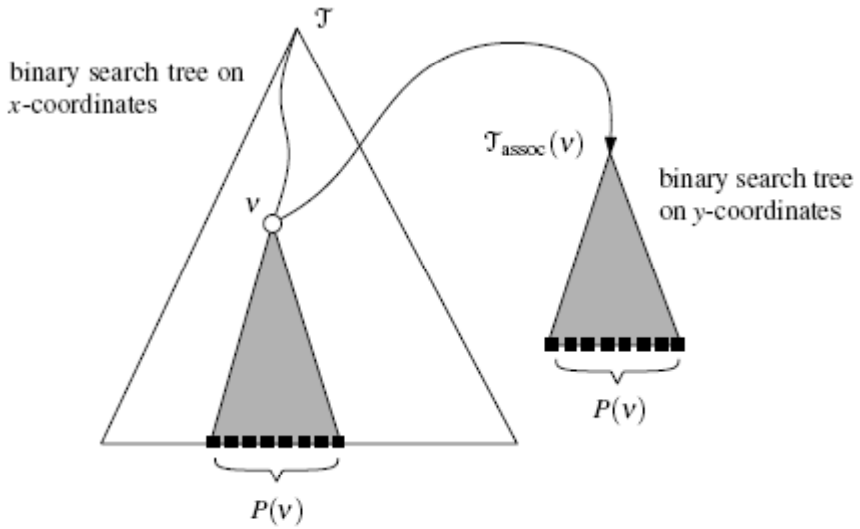
Data structure

Range tree is a multi-level data structure:

- The main tree is a balanced binary search tree *T* built on the x-coordinate of the points in P (first-level tree)

- For any internal or leaf node v in *T*, the canonical subset P(v) is stored in a balanced binary search tree $T_{assoc}(v)$ on the y-coordinate of the points. The node v stores a pointer to the root of $T_{assoc}(v)$, which is called the *associated structure* of v (second-level tree)

A 2-dimensional range tree



Algorithm BUILD2DRANGETREE(P)

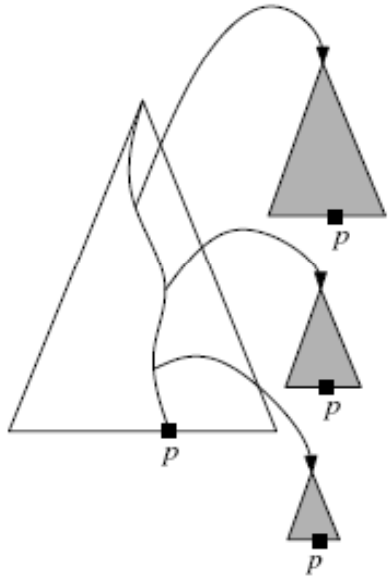
Input. A set P of points in the plane.

Output. The root of a 2-dimensional range tree.

1. Construct the associated structure: Build a binary search tree $\mathcal{T}_{assoc}$ on the set P_y of y-coordinates of the points in P. Store at the leaves of $\mathcal{T}_{assoc}$ not just the y-coordinate of the points in P_y, but the points themselves.
2. **if** P contains only one point
3. **then** Create a leaf v storing this point, and make $\mathcal{T}_{assoc}$ the associated structure of v.
4. **else** Split P into two subsets; one subset P_{left} contains the points with x-coordinate less than or equal to x_{mid}, the median x-coordinate, and the other subset P_{right} contains the points with x-coordinate larger than x_{mid}.
5. $v_{left} \leftarrow$ BUILD2DRANGETREE(P_{left})
6. $v_{right} \leftarrow$ BUILD2DRANGETREE(P_{right})
7. Create a node v storing x_{mid}, make v_{left} the left child of v, make v_{right} the right child of v, and make $\mathcal{T}_{assoc}$ the associated structure of v.
8. **return** v

Lemma 5.6 A range tree on a set of n points in the plane requires O(n logn) storage.

A point p in P is stored only in the associated structure of nodes on the path in *T* towards the leaf containing p.



Algorithm 2DRANGEQUERY($\mathcal{T}, [x : x'] \times [y : y']$)
Input. A 2-dimensional range tree $\mathcal{T}$ and a range $[x : x'] \times [y : y']$.
Output. All points in $\mathcal{T}$ that lie in the range.

1. $v_{\text{split}} \leftarrow$FINDSPLITNODE($\mathcal{T}, x, x'$)
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 call 1DRANGEQUERY on the subtrees right of the path. $*$)
5. $v \leftarrow lc(v_{\text{split}})$
6. **while** v is not a leaf
7. **do if** $x \leqslant x_v$
8. **then** 1DRANGEQUERY($\mathcal{T}_{\text{assoc}}(rc(v)), [y : y']$)
9. $v \leftarrow lc(v)$
10. **else** $v \leftarrow rc(v)$
11. Check if the point stored at v must be reported.
12. Similarly, follow the path from $rc(v_{\text{split}})$ to x', call 1DRANGEQUERY with the range $[y : y']$ on the associated structures of subtrees left of the path, and check if the point stored at the leaf where the path ends must be reported.

Lemma 5.7 A query with an axis-parallel rectangle in a range tree storing n points takes $O(\log^2 n + k)$ time, where k is the number of reported points.

Theorem 5.8 Let P be a set of n points in the plane. A range tree for P uses O(n logn) storage and can be constructed in O(n logn) time. By querying this range tree one can report the points in P that lie in a rectangular query range in $O(\log^2 n + k)$ time, where k is the number of reported points.

Higher-dimensional range trees

Let P be a set of points in d-dimensional space.

- Construct a balanced binary search tree on the first coordinate of the points (first-level tree),
- For each node ν construct an associated structure $T_{assoc}(\nu)$; the second-level tree $T_{assoc}(\nu)$ is a (d-1)-dimensional range tree for the points in $P(\nu)$, restricted to their last d-1 coordinates.
- ...
- The recursion stops when points restricted to their last coordinate; these are stored in a 1-dimentional range tree.

Theorem 5.9 let P be a set of n points in d-dimensional space, where $d \geq 2$. a range tree for P uses $O(n \log^{d-1} n)$ storage and it can be constructed in $O(n \log^{d-1} n)$ time. One can report the points in P that lie in a rectangular query range in $O(\log^{d} n + k)$ time,where k is the number of reported points.

Proof:

$T_d(n)=O(n\log n)+O(\log n).T_{d-1}(n)$

$T_2(n)=O(n\log n)$

$Q_d(n)=O(\log n)+O(\log n).Q_{d-1}(n)$

$Q_2(n)=O(\log 2n)$

General Sets of points

- Composite number of two reals a and b: $(a \mid b)$
- Order on the composite-number space:

$(a \mid b) < (a' \mid b') \Leftrightarrow a<a'$ or $(a=a'$ and $b<b')$

- $p:=(p_x , p_y) \rightarrow \tilde{p} := ((p_x \mid p_y) , (p_y \mid p_x))$
- $R := [x : x'] \times [y : y'] \rightarrow \tilde{R} := [(x \mid -\infty) : (x' \mid +\infty)] \times [(y \mid -\infty) : (y' \mid +\infty)]$

Lemma 5.10 let p be a point and R a rectangular range. then

$$p \in R \Leftrightarrow \tilde{p} \in \tilde{R}$$