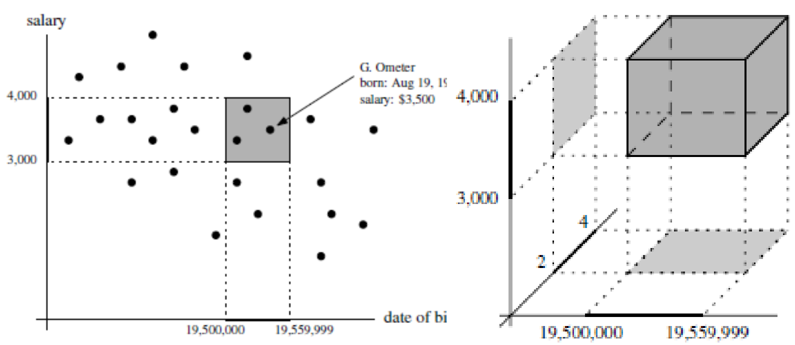
Orthogonal Range Searching
Querying a Database

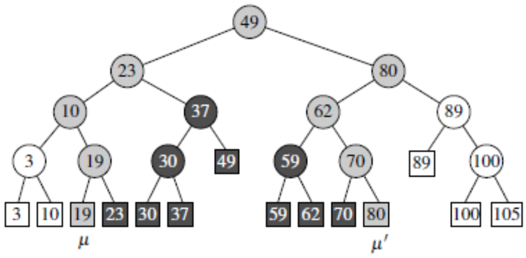
Min-Te Sun, Ph.D.

Interpreting Query Geometrically



1-D Range Searching

- Let $P := \{p_1, p_2, \ldots, p_n\}$ be the given set of points on the real line, a query asks for the points inside [x:x’]
  – At the moment each point is assumed to have distinct coordinate value
- Balanced binary search tree
  – Internal node stores the largest value in left subtree
  – Point value is stored at leaf node

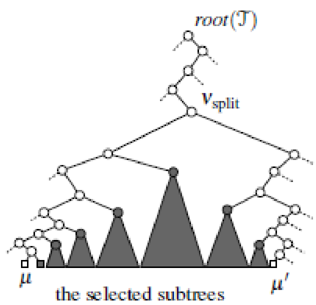


Basic Idea

- Find the split node!

FINDSPLITNODE($\mathcal{T}, x, x'$)
Input. A tree $\mathcal{T}$ and two values x and x' with $x \leqslant x'$.
Output. The node ν where the paths to x and x' split, or the leaf where both paths end.

$\nu \leftarrow root(\mathcal{T})$
while ν is not a leaf **and** $(x' \leqslant x_\nu$ **or** $x > x_\nu)$
    do if $x' \leqslant x_\nu$
        then $\nu \leftarrow lc(\nu)$
        else $\nu \leftarrow rc(\nu)$
return ν



1-D Range Searching Algorithm

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.

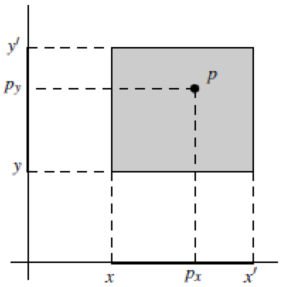
```
v_split ← FINDSPLITNODE(T, x, x')
if v_split is a leaf
  then Check if the point stored at v_split must be reported.
  else (* Follow the path to x and report the points in subtrees right of the
          path. *)
       v ← lc(v_split)
       while v is not a leaf
         do if x ≤ x_v
              then REPORTSUBTREE(rc(v))
                   v ← lc(v)
              else v ← rc(v)
       Check if the point stored at the leaf v must be reported.
       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.
```

Analysis of Algorithm

- *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.*

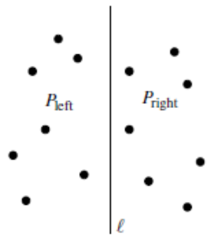
2-D Range Searching

- 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 $p_y \in [y : y']$.

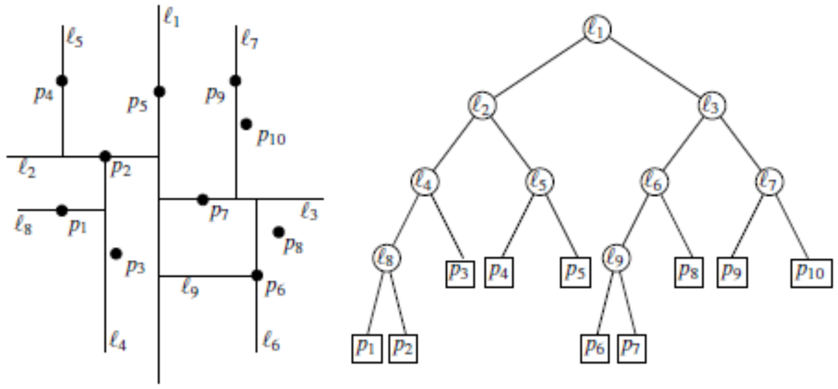


Kd-Trees

- At the root we split the set P with a vertical line into two subsets of roughly equal size. The splitting line l_1 is stored at the root.
  - P_{left}, the subset of points to the left or on the splitting line, is stored in the left subtree, and the rest to P_{right}
    - P_{left} is again split into two subsets with a horizontal line l_2; the points below or on it are stored in the left subtree of the left child, and the rest are stored in the right subtree.
      - …



Kd-Tree Example



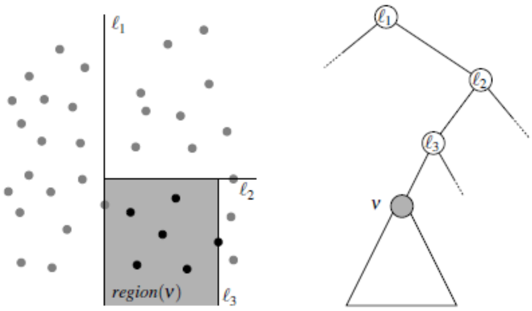
Kd-Tree Construction Algorithm

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

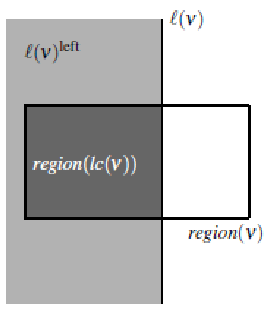
Complexity Analysis and Basic Query Idea

- A kd-tree for a set of n points uses O(n) storage and can be constructed in O(nlogn) time.
- How to do range query over a kd-tree?



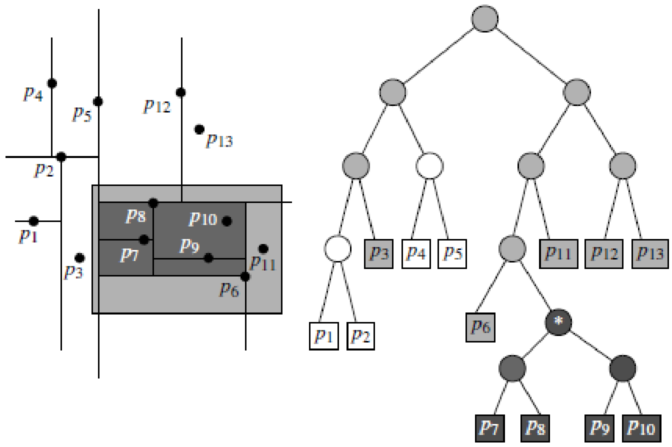
Kd-tree Query Algorithm

```
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.
  if v is a leaf
     then Report the point stored at v if it lies in R.
     else if region(lc(v)) is fully contained in R
             then REPORTSUBTREE(lc(v))
             else if region(lc(v)) intersects R
                     then SEARCHKDTREE(lc(v), R)
          if region(rc(v)) is fully contained in R
             then REPORTSUBTREE(rc(v))
             else if region(rc(v)) intersects R
                     then SEARCHKDTREE(rc(v), R)
```

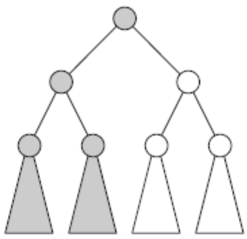


- The main issue is how to determine region(lc(*v*))
  - Ex: The region corresponding to the left child of *v* at even depth can be computed from *region*(*v*) as: *region*(*lc*(*v*)) = *region*(*v*)∩(*v*)$^{\text{left}}$

A Query Example



Complexity Analysis



$$Q(n) = \begin{cases} O(1), & \text{if } n = 1, \\ 2 + 2Q(n/4), & \text{if } n > 1. \end{cases}$$

- A query with an axis-parallel rectangle in a kd-tree storing n points can be performed in *in O(√n+k) time, where k is the number of of reported points*

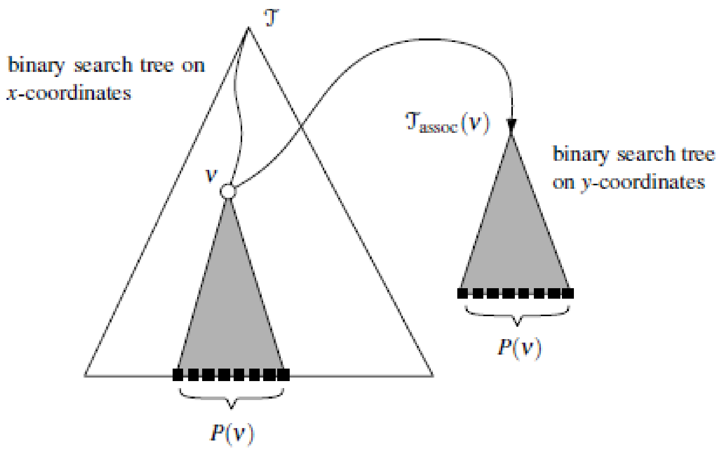
Summary of Kd-tree Results

- *A kd-tree for a set P of n points in the plane uses O(n) storage and can be built in O(nlogn) time. A rectangular range query on the kd-tree takes O(√n+k) time, where k is the number of reported points*
- *The d-dimension kd-tree requires O(dn) storage and O(dnlogn) time for construction and the query is bounded by* $O(n^{1-1/d}+k)$
- *Can we have faster query time complexity by sacrificing more storage space?*
  - *Range trees!*

Basic Idea of Range Trees

- Query is processed one coordinate at a time
- One balanced binary search tree for each coordinate
  - If there are more dimensions, a node is associated with another balanced binary search tree for additional coordinates
  - For the last coordinate, the node stores the point info

An Example of 2-D Range Tree



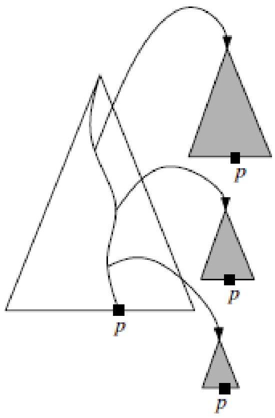
Range Tree Construction Algorithm

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 ν storing this point, and make $\mathcal{T}_{assoc}$ the associated structure of ν.
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. $\nu_{left} \leftarrow$ BUILD2DRANGETREE(P_{left})
6. $\nu_{right} \leftarrow$ BUILD2DRANGETREE(P_{right})
7.   Create a node ν storing x_{mid}, make ν_{left} the left child of ν, make ν_{right} the right child of ν, and make $\mathcal{T}_{assoc}$ the associated structure of ν.
8. **return** ν

Complexity Analysis

- *A range tree on a set of n points in the 2-D plane requires* $O(n\log n)$ *storage and* $O(n\log n)$ time



Range Tree Query Algorithm

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.

$\nu_{\text{split}} \leftarrow$ FINDSPLITNODE($\mathcal{T}, x, x'$)
if ν_{split} is a leaf
  then Check if the point stored at ν_{split} must be reported.
  else (∗ Follow the path to x and call 1DRANGEQUERY on the subtrees right of the path. ∗)
    $\nu \leftarrow lc(\nu_{\text{split}})$
    while ν is not a leaf
      do if $x \leqslant x_\nu$
        then 1DRANGEQUERY($\mathcal{T}_{\text{assoc}}(rc(\nu)), [y:y']$)
          $\nu \leftarrow lc(\nu)$
        else $\nu \leftarrow rc(\nu)$
    Check if the point stored at ν must be reported.
    Similarly, follow the path from $rc(\nu_{\text{split}})$ to x', call 1DRANGE-QUERY 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.

Complexity Analysis

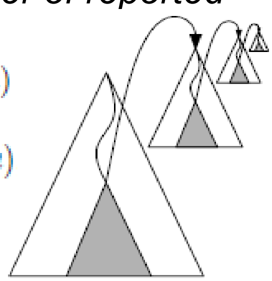
- *A query with an axis-parallel rectangle in a range tree storing n points takes* $O(\log^2 n+k)$ *time, where k is the number of reported points*
- *Let P be a set of n points in the plane. A range tree for P uses* $O(n\log n)$ *storage and can be constructed in* $O(n\log n)$ *time. By querying this 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 n points in d-dimensional space, where d >= 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*

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

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



Deal with Same Coordinate Value

- Treat each coordinate of a point as the "composite number"!
  - Ex: If a point p's coordinate is (p_x, p_y), then its composite coordinate will be (($p_x|p_y$), ($p_y|p_x$)).
  - The sorting of points are done by looking at the values in composite one by one, from left to right.
  - No two points will have the same composite value as long as no two points have exactly the same coordinate
- *Let p be a point and R a rectangular range and p' and R' be the corresponding composites, then* $p \in R \Leftrightarrow p' \in R'$.