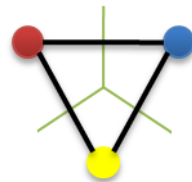
KOREA ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY
KAIST
한국과학기술원
SINCE 1971

Range Searching



Geometric Computing

Range Searching

Given a set P of geometric objects in E^d
and a query range(a connected region) q,
report (or count) all objects in P intersecting the range q.

- Objects : points, line segments, …
- Range : interval, rectangle, simplex,…

Point Enclosure

- Set R of ranges in E^d
- Point enclosure query : given a query point q, report all ranges in R containing q.
- Dual of the range searching problem (also called *inverse orthogonal range searching*).
- For d=1, R : set of intervals.
- For d=2, R : set of rectangles.

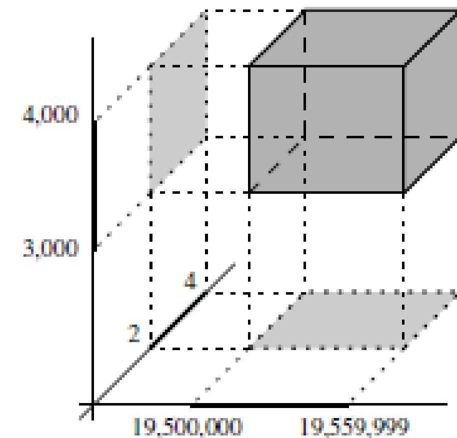
Point Location

- Let S be a planar subdivision with n edges.
- Planar point location problem : given a query point q, report the face f of S that contains q.

Orthogonal range searching

- Set P of points in d-dimensional space E^d.
- Range query : report the points of P contained in a query range r.
- Rectangular or orthogonal range query :

  $r = (a_1, b_1) \times (a_2, b_2) \times \ldots \times (a_d, b_d)$
- Collection of 1-d searches

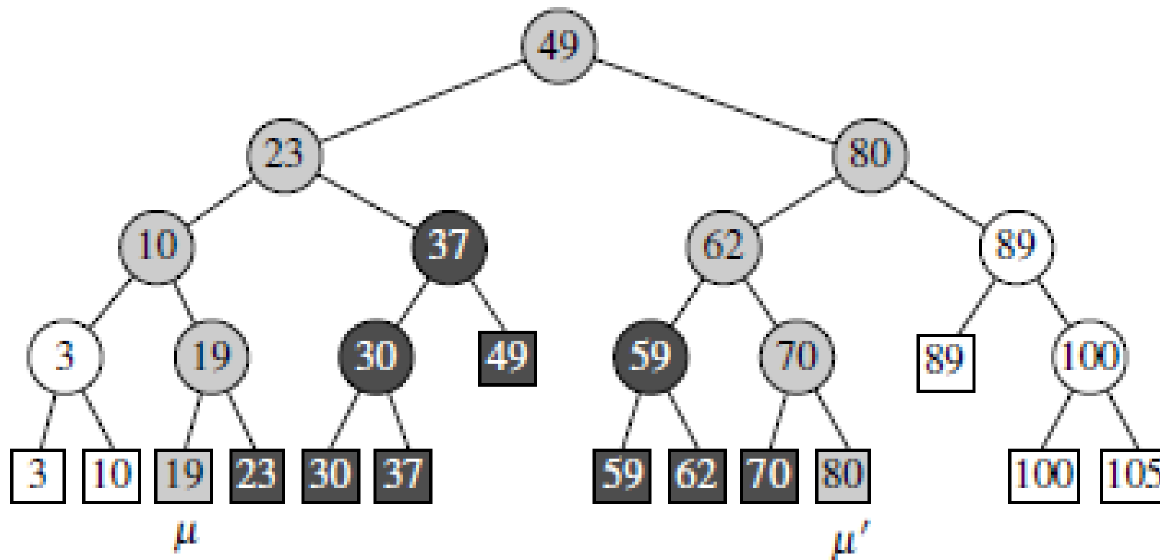


1-dimensional range searching

Given a set of points P = { $p_1, p_2, \ldots, p_n$ } on the real line, and a query interval [x:x'], report all the points in P in [x:x'].

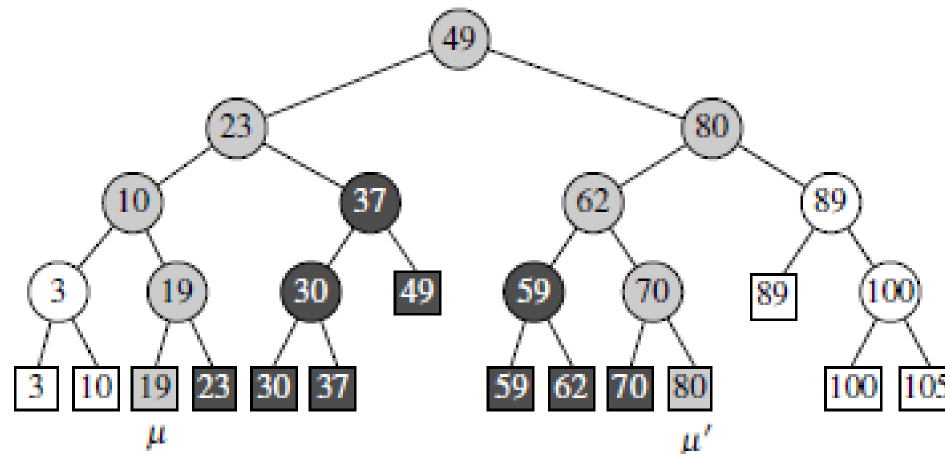
1-dimensional range searching

- Use a balanced binary search tree (internal node labeled with the largest key in its left subtree)
- Search for x and x'. (e.g. [18:77])
- How to report points in between?



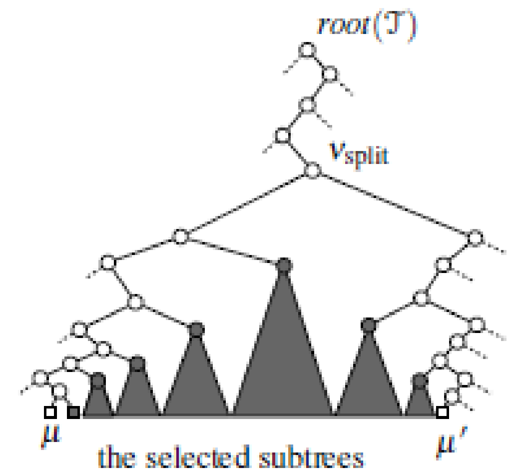
1-dimensional range searching

- Associate each node with the subset of points in its subtree → O(n) canonical subsets.
- The canonical subset for any range can be identified in O(log n) time.



1-dimensional range searching

- Search the tree to find the leftmost leaf μ and rightmost leaf μ'.
- The search paths to μ and μ' may share some common subpath.
- Once the paths diverge, as we follow the left path to μ, whenever the path goes to the left child, add the right child.
- Similarly, as we follow the right path to μ', whenever the path goes to the right child, add the left child.



1-dimensional range searching

- Space for n points : O(n)
- Preprocessing : O(n log n) time
- Reporting query : O(log n + k) where k is the number of points reported.
- Counting query : store total number of points in each subtree (preprocessing) and sum all these over canonical subsets ⇒ O(log n) time.
- Time for insertion/deletion of a point : O(log n)

1-dimensional range searching

- Space for n points : O(n)
- Preprocessing : O(n log n) time
- Reporting query : O(log n + k) where k is the number of points reported.
- Counting query : store total number of points in each subtree (preprocessing) and sum all these over canonical subsets ⇒ O(log n) time.
- Time for insertion/deletion of a point : O(log n)

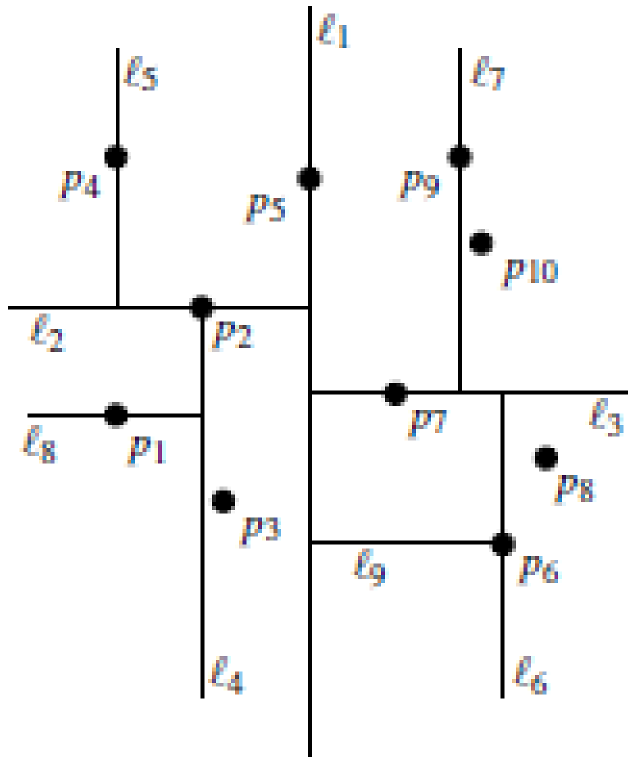
2-dimensional range searching

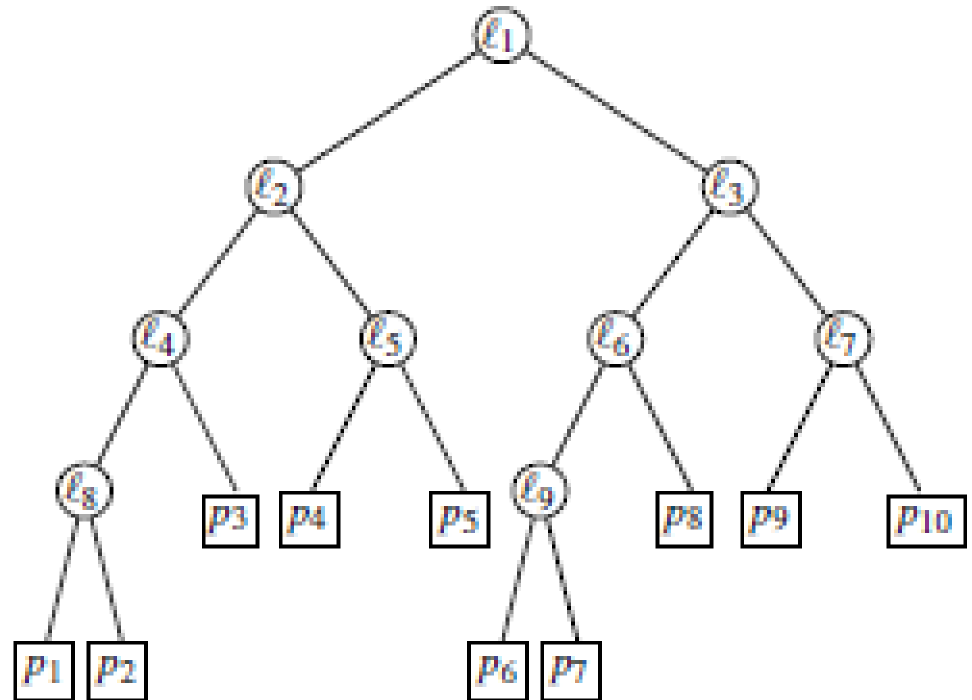
- Query range : rectangles
- How to extend binary search tree?

Kd-trees

- Split on x-coordinate, next on y-coordinate, then again on x-coordinate, and so on.
- Easy to implement and practical (but not asymptotically optimal).

Kd-tree





Kd-trees

- How to build?
- How to query?

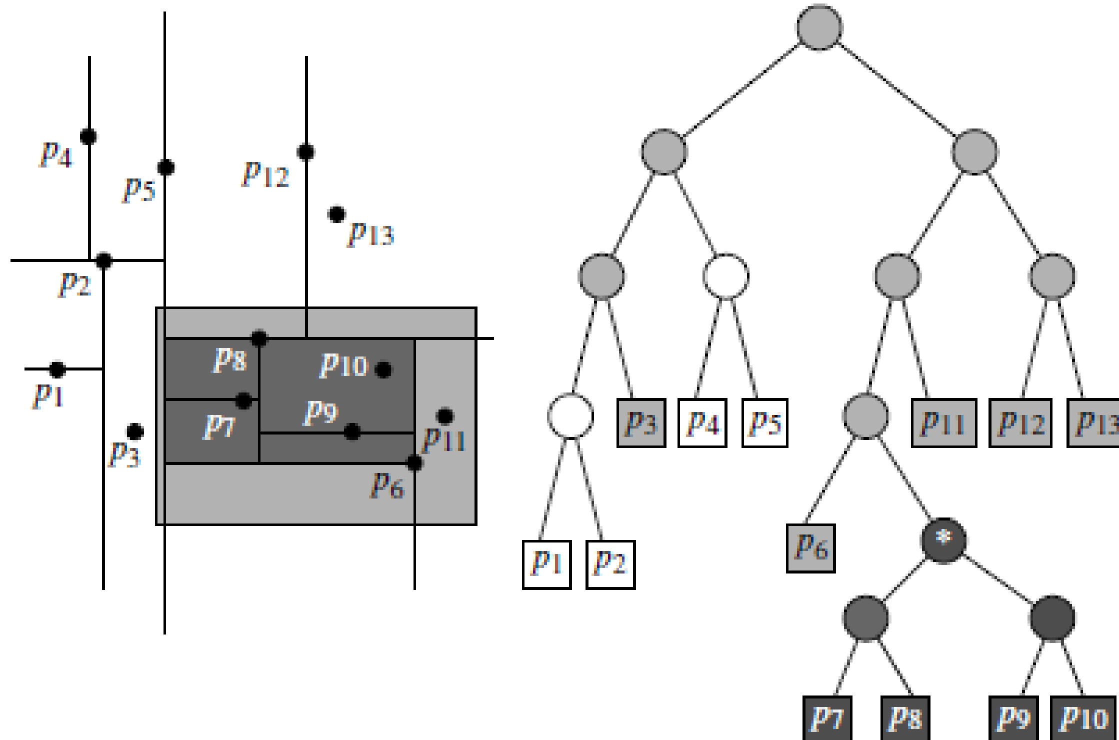
Constructing kd-tree

- Top-down recursive procedure
- Maintain 2 lists (sorted by x and y).
- Find median (O(1)).
- Split list (O(n))
- $T(n) = 2\,T(n/2) + n = O(n \log n)$ time
- O(n) space.

Range searching in kd-trees

- Let Q denote the desired range.
- Let t denote the current node.
- Let C denote the rectangular cell associated with t.
- Traverse the tree recursively.
- At a leaf, check whether its point lies in the range Q in O(1) time.
- At an internal node, if C is disjoint with Q, do nothing. If C is contained within Q, report every point in its subtree. If C partially overlaps Q, recurse on t's 2 children.

Range searching in kd-trees



Analysis of query time

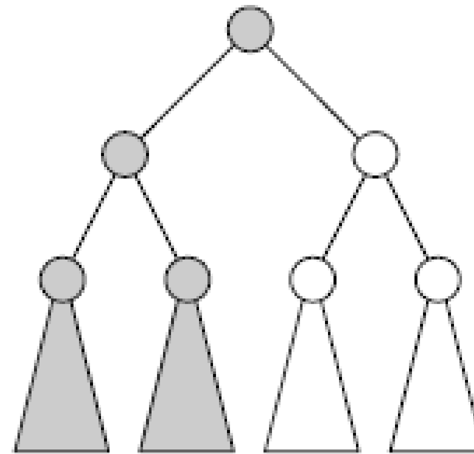
- Time to traverse a subtree and report the points in its leaves is O(k).
- Need to bound the number of nodes visited that are not in one of the traversed subtrees.
- For each such node, its cell overlaps the range but is not contained.
- We say that such a cell is *stabbed* by the query.

Analysis of query time

- Lemma : Given a balanced kd-tree with n points using alternating splitting rule, any vertical or horizontal line stabs $O(\sqrt{n})$ cells of the tree.

Pf) Q(n) : number of intersected regions

$$Q(n) = 2 + 2\,Q(n/4) = O(\sqrt{n})$$



Analysis of query time

- Lemma : Given a balanced kd-tree with n points using alternating splitting rule, any vertical or horizontal line stabs $O(\sqrt{n})$ cells of the tree.
- Extend 4 sides of Q into lines ⇒ total number of cells stabbed by all 4 lines is at most $O(\sqrt{n})$
- Since we only make recursive calls when a cell is stabbed, total number of nodes visited is $O(\sqrt{n})$
- Given a balanced kd-tree with n points, reporting query takes $O(\sqrt{n} + k)$ time where k is the number of reported points and counting query takes $O(\sqrt{n})$.
- d-dimensional kd-tree ?

Can we make query time faster?

Canonical subset

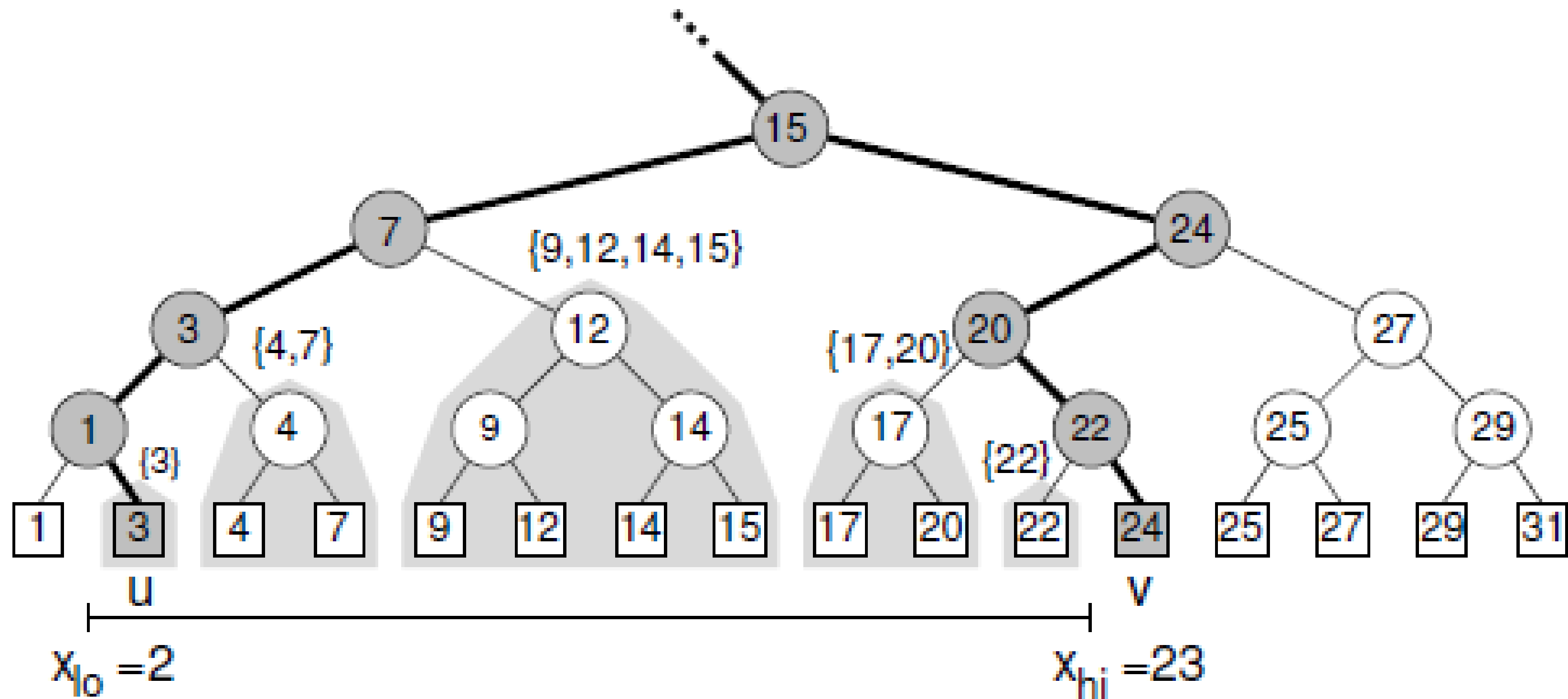
- A common approach to solve range queries is to represent P as a collection of *canonical subsets* $\{P_1, P_2 \ldots, P_k\}$, each $P_i \subseteq P$ s.t. any set can be formed as the disjoint union of canonical subsets.
- n singleton sets : O(n) space needed, k sets needed to answer a query with k objects.
- The power set of P : any query can be answered with a single canonical subset but we need 2^n subsets.
- Balance total number of canonical subsets (space) and number of canonical subsets needed to answer a query (time).

Range trees

- 1-dimensional range tree is just a balanced binary search tree.
- Each node of the tree is associated with its *canonical subset* of points. (points in the subtree rooted at the node.)
- The answer to any 1-dimensional range query can be represented as the disjoint union of a small collection of $m = O(\log n)$ canonical subsets, $\{S_1, \ldots, S_m\}$ where each subset corresponds to a node in the search.

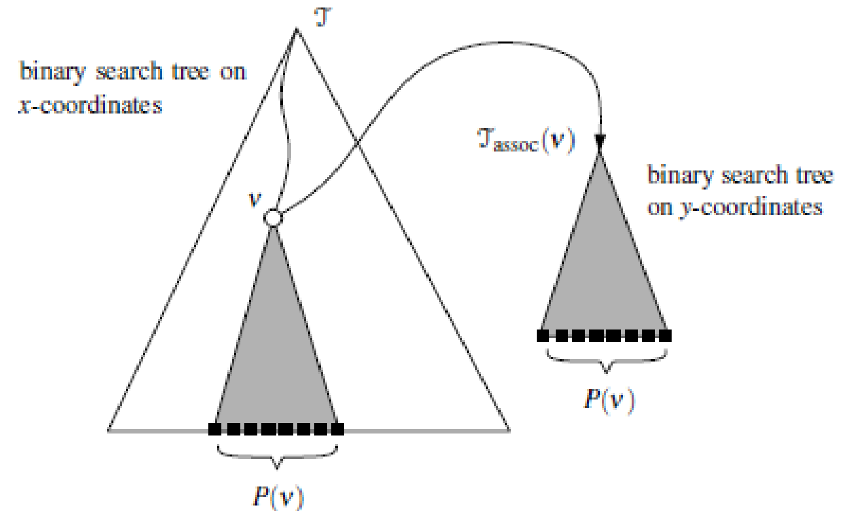
Range trees

Canonical sets for interval query



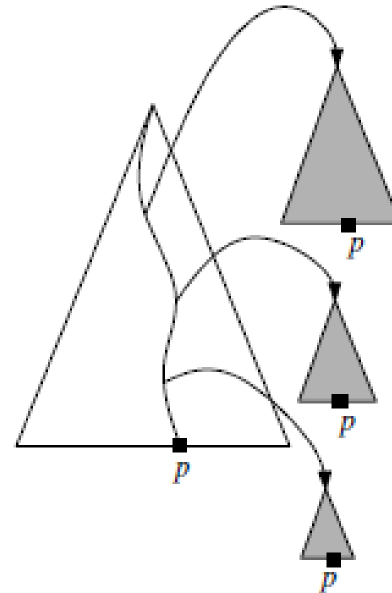
Range trees

- *Primary (first-level)* tree is a balanced binary search tree in x-coordinate.
- *Associated structure (second-level tree)* of v : For any internal or leaf node v of primary tree, its canonical subset is stored in a balanced binary search tree on the y-coordinate.



Space of range tree

- Main tree : O(n)
- Depth of the main tree is O(log n)
- Each point appears in O(log n) secondary trees.
- Total space : O(n log n)



Constructing range trees

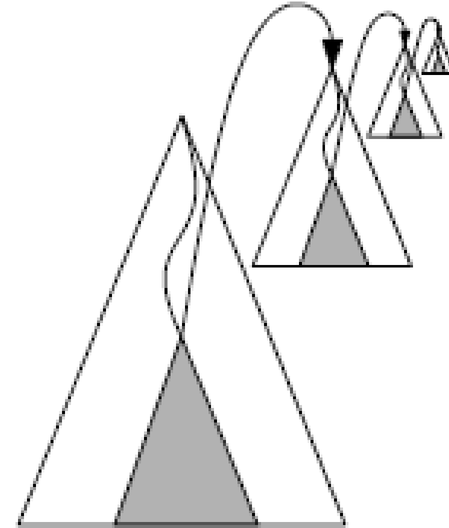
- Maintain 2 sorted lists : 1 sorted on x, 1 sorted on y.
- Then, for a node in the main tree, it takes linear in the size of its canonical subset to construct secondary tree.
- Total construction time is the same as the space : $O(n \log n)$

Querying range trees

- First, determine the canonical sets that satisfy the first query.
- For each canonical subset, 1-dimensional range search on y using auxiliary tree.
- Query time:
  - $O(\log n + k(v))$ for each node v visited in primary tree where $k(v)$ is output for v.
  - $O(\log^2 n + k)$

Higher-dimensional range trees

- Build primary tree on x-coordinate,
  - For each node, build structure for (d-1)-dimensional case.
- Query time: $O(\log^d n + k)$
- Space : $O(n \log^{d-1} n)$
- Construction time : $O(n \log^{d-1} n)$



Can we make query time even faster?

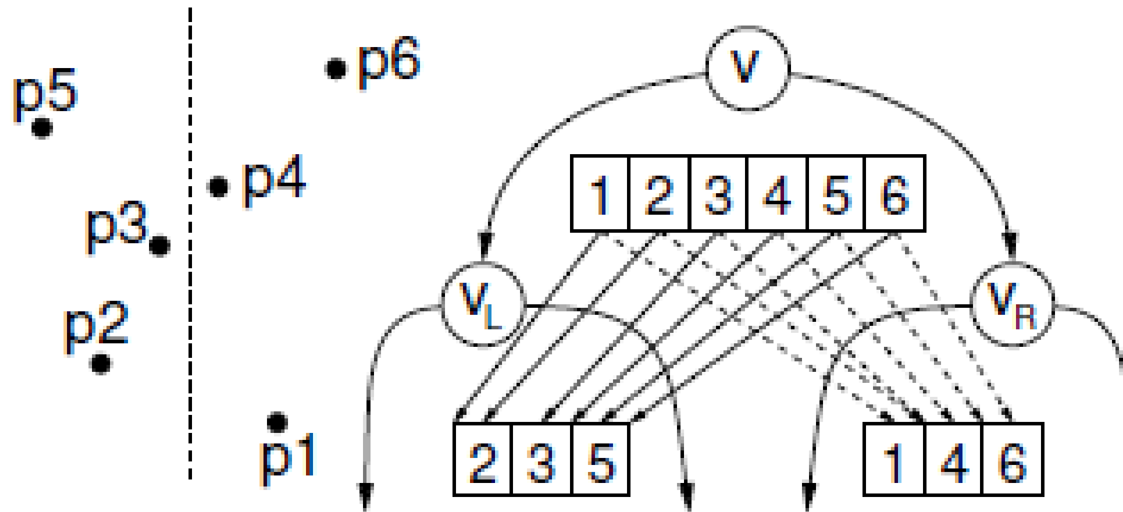
Fractional Cascading

- Search y interval in O(1) time?
- We are repeatedly searching different lists, but always with the same key.
- Merge all the different lists into a single massive list, do one search in this list in O(log n) time, and then use the information about the location of the key to answer all the remaining queries in O(1) time each.

Fractional Cascading

- Let v be an arbitrary internal node in the range tree of x coordinates and v(L) and v(R) be its left and right children.
- Let A(v) be the sorted auxiliary list for v and let A(L) and A(R) be the sorted auxiliary list for its respective children.
- A(v) is a disjoint union of A(L) and A(R). For each element in A(v), store two pointers, one to the smallest y-coord of equal or larger value in A(L) and the other to the smallest y-coord of equal or larger value in A(R).
- Once we know a position of an item in A(v), we can determine its position either in A(L) or A(R) in O(1) additional time.

Fractional Cascading



Fractional Cascading

- In 2-D,
  - $O(\log n + k)$ query time
- In d-D
  - $O(\log^{d-1} n + k)$ query time

Homework

- Due 3/12 10 AM
- Introduce yourself. (name, advisor, research interests, why you take CS504)
- Exercises 5.1, 5.3, 5.10