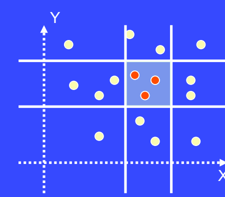
Computational Geometry

Chapter 5

Range Searching

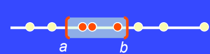
Orthogonal Range Searching

- **Problem:** Given a set of *n* points in R^d, preprocess them such that reporting or counting the *k* points inside a *d*-dimensional axis-parallel box will be most efficient.
- Desired *output-sensitive* query time complexity – O(*k*+f(*n*)) for reporting and O(f(*n*)) for counting, where f(*n*)=o(*n*), e.g. f(*n*)=log*n*.
- **Sample application:** Report all cities within 100 mile radius of Boston.



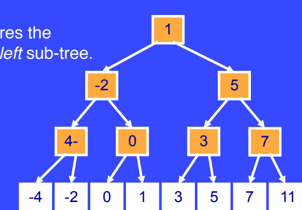
Range Searching – 1D

- In a one-dimensional world, points are real numbers and the query is two numbers (*a*,*b*).
- Simple O(log*n*) algorithm:
  - Preprocessing: Sort points in O(*n*log*n*).
  - Query: (Binary) search for *a* and *b* in list in O(log*n*).
    List all values inbetween.
- Cannot be easily generalized to higher dimensions (why not ?).



Range Searching – 1D Tree

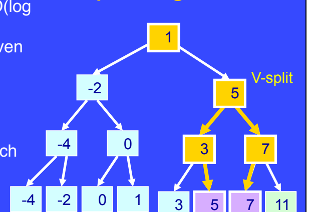
- Range tree solution:
  - Sort points.
  - Construct a binary balanced tree, storing the points in its leaves.
  - Each tree node stores the largest value of its *left* sub-tree.



Range Searching in 1D Tree

Input Range: 3.5-8.2

- Required time for finding a leaf: O(log *n*).
- Find the two boundaries of the given range in the leaves *u* and *v*.
- Report all the leaves in *maximal subtrees* between *u* and *v*.
- Mark the vertex at which the search paths diverge as V-split.

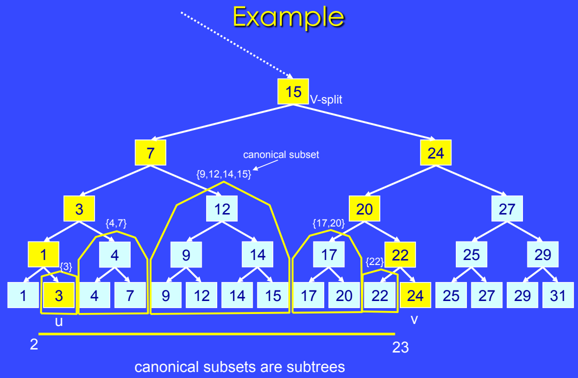


- Continue to find the two boundaries, reporting values in the subtrees:
  When going left (right), report the entire right (left) subtree.
- When reaching a leaf, check it exhaustively.

General Idea

- Build a data structure storing a "small" number of canonical subsets, such that:
  - The c.s. may overlap.
  - Every query may be answered as the union of a "small" number of c.s.
- The geometry of the space enables this.
- Two extremes:
  - Singletons – O(*k*) query time, even for counting.
  - Power set – O(1) query time. O(2^n) storage.

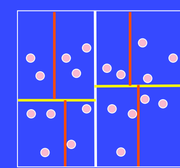
Example



75.

2D-Trees

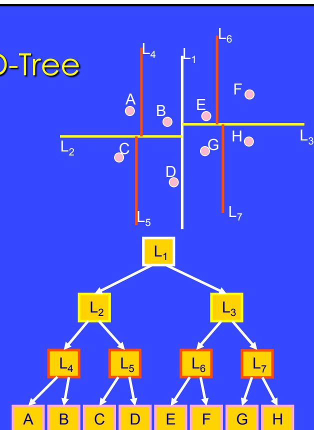
- Given a set of points in 2D.
- Bound the points by a rectangle.
- Split the points into two (almost) equal size groups, using a horizontal or vertical line.
- Continue recursively to partition the subsets, until they are small enough.
- Canonical subsets are subtrees.



85.

2D-Tree

- Partitions 2D space into axis-aligned rectangular regions.
- Nodes represent partition lines, and leaves represent input points.



construction complexity:

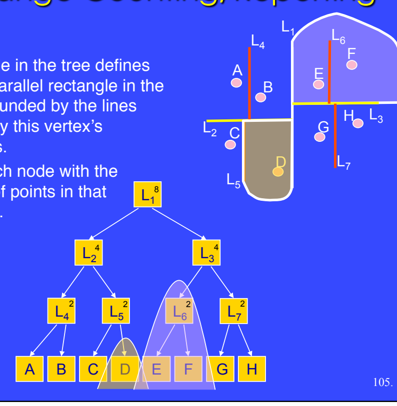
$$T(n) = \begin{cases} O(1) & n = 1 \\ O(n) + 2T\left(\frac{n}{2}\right) & n > 1 \end{cases}$$

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

95.

Range Counting/Reporting

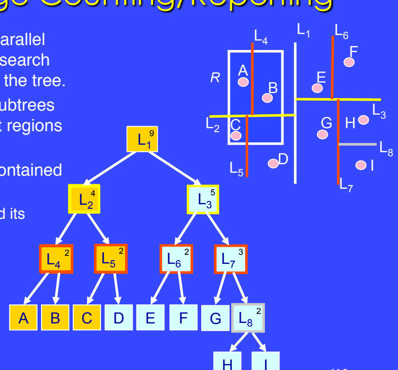
- Each node in the tree defines an axis-parallel rectangle in the plane, bounded by the lines marked by this vertex's ancestors.
- Label each node with the number of points in that rectangle.



105.

Range Counting/Reporting

- Given an axis-parallel range query *R*, search for this range in the tree.
- Traverse only subtrees which represent regions intersecting *R*.
- If a subtree is contained entirely in *R*:
  - Counting: Add its count.
  - Reporting: Report entire subtree.



115.

Runtime Complexity

- *k* nodes are reported. How much time is spent on internal nodes? The nodes visited are those that are **stabbed** by *R* but not contained in *R*. How many such nodes are there ?
- **Theorem**: Every side of *R* stabs O(√*n*) cells of the tree.
- **Proof**: Extend the side to a full line (WLOG - horizontal):
  - In the first level it stabs two children.
  - In the next level it stabs (only) two of the four grandchildren.
  - Thus, the recursive equation is:

$$Q(n) = \begin{cases} 1 & n = 1 \\ 2 + 2Q\left(\frac{n}{4}\right) & otherwise \end{cases}$$

$$= O\left(\sqrt{n}\right)$$

- Total query time: O(√*n* + *k*).

125.

Kd-Trees – Higher Dimensions

- For a d-dimensional space:
  - Construction time: $O(d\, n\log n)$.
  - Space Complexity: $O(dn)$.
  - Query time complexity: $O(dn^{1-1/d}+k)$.

Question: Are kd trees useful for non-orthogonal range queries, e.g. disks, convex polygons ?

Fact: Using *interval trees*, orthogonal range queries may be solved in $O(\log^{d-1}n+k)$ time and $O(n\log^{d-1}n)$ space.

135.