Range Trees and Priority Search Trees
Range Trees

• Originally invented by Bentley and Maurer
• **Task**: find all points in a given range
  
  + asymptotically faster search structures than point quadtrees and k-d trees
  - Significant higher storage requirements
1-d range searching

• Recall: in 1-d the problem is easily solved using classical search data structures
1-d range search

• balanced search tree constructions

Leaves stores the data points they are linked in a doubly-linked list
1-d range search

- balanced search tree constructions

Search for Left endpoint of interval
1-d range search

• balanced search tree constructions

Search for Right endpoint of interval
1-d range search

- balanced search tree constructions

all elements between the left and the right search are output
1-d range search

• **Theorem**: 1-d range search takes $O(\log n + k)$, where $k$ is the output size. The data structure uses linear space.

• **Note**: One only has to do one binary search. We can search for the Left and then list the elements in order until the element to be printed next exceeds the Right.
2-d range search

• **Operation:** RangeSearch([Lx:Rx], [Ly:Ry])

• **Task:** report all points p(x,y) who’s coordinates satisfy the following conditions:
  – Lx ≤ x ≤ Rx and Ly ≤ y ≤ Ry

• **Idea:** generalize the 1-d concept
2-d range search

• 1. Build a balanced binary search tree with leaves sorted by x-coordinates

```
   /
  /  \
 /    \
/      \
x1 ....          ....    xn
```

2-d range search

• 2. In each internal node, build a range tree on all elements in that subtree sorted by y-coordinates
the search

- The search \([\text{Lx:Rx}, \text{Ly:Ry}]\) has two phases:
  - I. we search for \text{Lx and Rx} in the x-sorted tree
  - then we identify \(Q := \text{LCA}(\text{Lx,Rx})\)
the search

• The search ([Lx:Rx], [Ly:Ry]) has two phases:
  • II. for all nodes Li on the paths from Q to Lx do
    Search 1-d RangeTree(RightChild(Li)) range [Ly:Ry]
the search

• II. cont’d
• for all nodes Ri on the paths from Q to Rx do
  Search 1-d RangeTree(LeftChild(Ri)) range [Ly:Ry]
Analysis

• **Theorem**: A 2-d range search can be carried out in $O(\log^2 n + \text{output})$ using $O(n \log n)$ storage.

Notes: for each of the $\log n$ elements on the search from the root to $Lx$ ($Rx$) we perform a search in a 1-d range tree on the $y$-coordinates again costing $O(\log n)$ time.

Each element is stored at most $O(\log n)$ times.
k-d range trees

• **Generalization:** The concept of generalizes easily to k-d by repeating the idea.

• Build a balanced binary search tree on the 1\textsuperscript{st} coordinate. Then, in each internal node, store a (k-1)-d range tree.

• **Theorem:** A k-d range search can be carried out in $O(\log^k n + \text{output})$ time using $O(n \log^{k-1} n)$ storage.
other structures

• Edelsbrunner improved on the 2-d range query to get $O(\log n + \text{output})$ still using only $O(n \log n)$ storage.
McCreight`s priority search tree

- Data structure for queries involving semi-infinite ranges in 2-d.
- \textbf{Search}([Lx:Rx], [Ly:∞])
Priority Search Tree

• Building a priority search tree
• 1. store values in leaf-positions of a balanced binary search tree sorted by x-coordinates.
• 2. heap-order
   starting at the root, store with each internal node I, a point (y-coordinate) in subtree(I) with max-y coordinate (among all nodes in subtree) that has not been stored at a node with shallower depth.
Priority Search Tree
Note

• The information in a max-heap is not strong enough to find elements in the range.
Note cont’d

• Report all keys between 5 and 45

45

All elements

65

may be some, all or no elements at all
Rangequery([Lx:Rx], [Ly:Ry])

- This is not easily answered!
- The x-search is easy because the elements are sorted by x-coordinate.
cont’d

• \([\text{Ly:Ry}]\) is the difficult part!
• we know that the y-coordinates of all points stored in a subtree under a node labelled \(c\) have y-coordinates at most \(c\).
• Thus \([\text{Ly:}\infty]\) is easy
• Analogously, we can answer \([-\infty, \text{Ry}]\).
• The final step is then to combine the answer obtained for the two queries.
• This is an intersection of two sets.
cont’d

the half-open query
[Ly:∞]
cont’d

the half-open query
\([-\infty, \text{Ry}]\)
The common intersection of the two semi-open queries may be quite small (or even nil). Thus the intersection may produce little (or no) output violating the desired $O(\log n + \text{output})$ time.