Computational Geometry
CS 531

POINT QUERIES
• Range Search Trees

Prof. Christoph Hoffmann
Acknowledgements to MIT OCW

Range Search
Orthogonal Range Query
Report all points in query rectangle

1D Solutions

• For 1D range searches, could use sorted arrays and binary search, but we want a data structure that is dynamic
• Data structure should allow going to higher-dimensional problems as well
• Balanced search trees a candidate, such as AVL trees, 2-3 trees, red-black trees, and so on.
1D Range Search

- Build a balanced search tree
- Find keys closest to upper and lower bound
- Report all keys in-between

- Advantageous to put indices at nodes
  - E.g., maximum key in left subtree
Query Method

- Root to split node:
  - If T.key < lower bound, go to the right
  - If T.key ≥ upper bound, go to the left
  - If T.key ∈ range, then split the path

- Split path left:
  - If T is a leaf, report it
  - If T.key < lower bound, go to the right
  - If T.key ≥ lower bound, then
    - Report right subtree of T
    - Go to the left
Example: Query Range [7, 41]

Query range: [40, 59]
Analysis

• Building the tree is $O(n \cdot \log(n))$
• Finding the split node and bounding paths is $O(\log(n))$
• Reporting a subtree $T$ is $O(|T|)$

• So, entire algorithm is
  • $O(n \cdot \log(n))$ preprocessing, and
  • $O(\log(n) + k)$ reporting $k$ points in the 1D range

2D Range Search

• Build a search tree for the $x$-coordinates
• For each tree node $T$ build a secondary search tree for the $y$-coordinates of the points in the subtree rooted in $T$

• Search for the split node and forked paths
• For each subtree so identified, search the secondary tree for the points
Searching Time

- We search the $y$-search trees for every node on the two split paths, i.e., $O(\log(n))$ searches.
- Each $y$-search takes $O(\log(n))$ time
- As result of the $y$-search we report only hits
- Total search time for $k$ hits:
  
  $O(\log^2(n) + k)$
Tree Construction, 2D

- Any point appears in the x-tree on one path from leaf to root, i.e., $\log(n)$ times, hence in $\log(n)$ y-trees.
- For $n = 2^k$ we have 1 tree size $2^k$, 2 trees size $2^{k-1}$, 4 trees size $2^{k-2}$, etc. The range tree thus requires $O(n \cdot \log(n))$ space therefore.
- Preprocessing time also $O(n \cdot \log(n))$ time, using pre-sorted x- and y-lists.

Range Tree in Higher Dimensions

Generalizes to $d$-dimensions with trees of size $O(n \cdot \log^{d-1}(n))$ space and time, and search time of $O(\log^d(n) + k)$