k-d trees

Store a set of points in k-dimensional space, supporting various search queries.

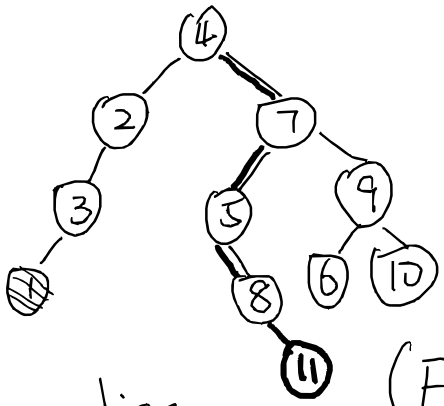
- Range searching
- Nearest neighbor searching

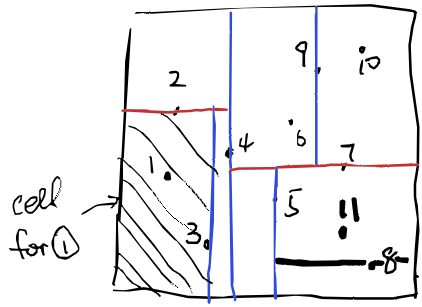
This lecture: focus on 2d trees

Representation: Let $P = (p_1, \ldots, p_n)$ be a set of points in $\mathbb{R}^2$. 2d tree is a tree data structure:

- root is a point p_i
- left subtree contains all points p_j with x-coordinate smaller than p_i
- right - - - - - - - - - - - larger - - - p_i.
- each subtree is constructed recursively using y-coordinates, then recursively in x-coord, y....

(generalizing BSTs)





(insert 11)

Insertion

(Each node also corresponds to a rectangle, a "cell")

Similar to BST

traverse down the tree and create a new leaf

$O(\text{depth})$ time

- may be large without rebalancing

Rebalancing the tree

If any subtree, which has size s, has less than $s/4$ nodes in its either left or right subtree, we rebuild the whole subtree.

- suppose we start with x-coordinates, pick the median by x-coord as the new root, and recurse
- height of the new subtree is $\log s$.

Rebuilding can be done in $O(s \log s)$ time.

Amortized analysis

Thm: n insertions take at most $O(n \log^2 n)$ time.

Proof. The height of the tree is at most $O(\log n)$ due to the rebuild. Thus, each insertion without the rebuild takes $O(\log n)$ time. Next, we analyze rebuilds.

Consider a rebuild of a subtree rooted at u, of size s. Consider the previous time this subtree was rebuilt, possibly as part of rebuilding a larger tree, let s' be the size at that time.

Both subtrees had $\approx \frac{s'}{2}$ size right after the previous rebuild, and one must insert at least $\Omega(s')$ keys to make it unbalanced. Exactly $s - s'$ keys are inserted in-between. Suppose $s - s' = c \cdot s'$, then

$$s - s' = \frac{c}{1+c} s = \Omega(s).$$

We "charge" the cost of this rebuild to each of these insertions: each is charged a cost of $O(\log s) \leq O(\log n)$.

Now, consider an insertion, the new node has $O(\log n)$ ancestors. Each ancestor may be rebuilt in the future, which would charge this insertion a cost of $O(\log n)$. This insertion is charged a total cost of $O(\log^2 n)$. The total time is $O(n \log^2 n)$. □

Deletion omitted.

Othorgonal range searching

- Count # points in a rectangle $R = [x_1, x_2] \times [y_1, y_2]$.

Query algorithm(u) u is a node into tree starting with the root

- Let C be the cell corresponding to u.
- If C is entirely contained in R
  - return size of the subtree
- If C is disjoint from R
  - return 0

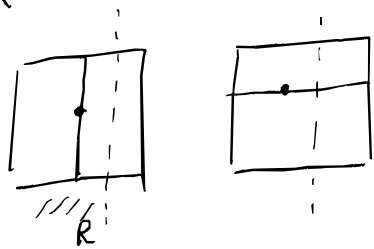
- return Query(u.left) + Query(u.right)
  + [point at $u \in R$]. (C intersects R)

Thm: For a perfectly balanced 2d tree, a range searching query takes $O(\sqrt{n})$ time.

Proof. The searching time is bounded by
cells that R intersects with

First, assume for simplicity, $R = (-\infty, x] \times (-\infty, +\infty)$

If a node u splits x-coordinate,
only one child may intersect R.



If u splits y-coordinate,
both children may intersect R.

The depth is $\log n$, we visit $2^{\frac{1}{2}\log n} = \sqrt{n}$ nodes.

For general R, each cell that intersects R must intersect one side of R.

There are at most $O(\sqrt{n})$ such cells. □

Nearest neighbor search

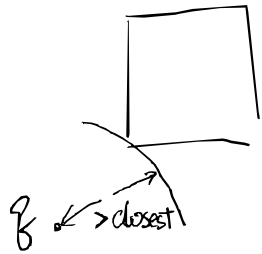
find the closest point to a given query point q.

NNS(q):

best = p_1

search(root, q)

return best



search(u, q)

If dist(q, cell corresponding to u) > ||best − q||

return.

If $||p_u - q|| < ||best - q||$

best = p_u

let v be the child of u that is closer to q

v' be the other child

search(v, q)

search(v', q)

Thm: For a random set of points, NNS(q) runs in $O(\log n)$ time.