Kd-Trees:

- Partition trees
- Orthogonal split
- Alternate cutting
dimension \( x, y, x, y \)
- Cells are axis-aligned rectangles (AABB)

Orthogonal range queries

- Given query rect. (AABB) count/report pts in this rect.
- Other range queries?
  - Circular disks
  - Halfplane

Nearest neighbor queries

- Given query pt, return closest pt in the set
- Find \( k \)th closest point
- Find farthest point from \( q \)

Queries?

Kd-Tree Queries

Rectangle methods for kd-cells:

- Split a cell \( r \) by a split pt \( s \in r \), along cut dim \( cd \)
  \[ \text{r.leftPart}(cd, s) \rightarrow \text{return rect with } \text{low} = r.\text{low} + \text{high} = r.\text{high} \]
  \[ \text{r.rightPart}(cd, s) \rightarrow \text{high} = r.\text{high} + \text{low} = r.\text{low} \]
  \[ \text{low}[cd] \leftarrow s[cd] \]

Axis-Aligned Rect \( m \in \mathbb{R}^d \)

- Defined by two pts:
  \[ \text{low}, \text{high} \]
- Contains pt \( q \in \mathbb{R}^d \) iff
  \[ \text{low}_i \leq q_i \leq \text{high}_i \]

Useful methods:

- Let \( r, c - \text{Rectangle2D} \)
- \( q - \text{Point} \)
- \( r.\text{contains}(q) \)  
- \( r.\text{contains}(c) \)  
- \( r.\text{isDisjointFrom}(c) \)

This Lecture: \( O(\sqrt{n}) \) time algo. for orthogonal range counting queries in \( \mathbb{R}^2 \)

General \( \mathbb{R}^d \): \( O(n^{1-\frac{1}{d}}) \)
Ortho. Range Query
- Assume: Each node p stores:
  - p.pt : splitting point
  - p.cutDim : cutting dim
  - p.size : no. of pts in p's subtree
- Tree stores ptr. to root and bounding box for all pts.
- Recursive helper stores current node p + p's cell.

Cases:
- p == null → fell out of tree: return 0
  → no point of p contributes to answer.
- Query rect is disjoint from p's cell
  → return p.size
  → every point of p's subtree contributes to answer.
- Otherwise: Rect + cell overlap: Recurse on both children.

Kd-Tree Queries

```java
int rangeCount(Rect R, KDNode p, Rect cell)
{
    if (p == null) return 0; // fell out of tree
    else if (!R.isDisjointFrom(cell)) return 0; // overlap
    else if (R.contains(cell)) return p.size; // take all
    else { int ct = 0;
            if (R.contains(p.pt)) ct++; // p's pt in range
            ct += rangeCount(R, p.left, cell.leftPart(p.cutDim, p.pt));
            ct += rangeCount(R, p.right, cell.rightPart(p.cutDim, p.pt));
        return ct;
    }
}
```
Theorem: Given a balanced kd-tree storing n pts in $\mathbb{R}^d$ (using alternating cut dim), orthog. range queries can be answered in $O(n \log n)$ time.

Analysis: How efficient is our algorithm?
- Tricky to analyze
- At some nodes we recurse on both children
  $\Rightarrow O(n)$ time?
- At some we don’t recurse at all!

Solving the Recurrence:
- Macho: Expand it
- Wimpy: Master Thm (CLRS)

Master Thm:

For us: $a = 2$

Since tree is balanced a child has half the pts + grandchild has quarter.

Recurrence: $T(n) = 2 + 2T(n/4)$

If we consider 2 consecutive levels of kd-tree, $l$ stabs at most 2 of 4 cells:

Lemma: Given a kd-tree (as in Thm above) and horiz. or vert. line $l$, at most $O(n \log n)$ cells can be stabbed by $l$.
Nearest Neighbor Query

- Maintain closest point seen so far - best
- On visiting any node p
  - if p point is closer
    best ← p point
  - else if worthwhile, recurse on other child

Point near Neighbor (Point q, KDNode p, Rect cell, Point best)

```plaintext
if (p ≠ null) {
  if (dist(q, p point) < dist(q, best))
    best ← p
  leftCell ← cell of p's left child
  rightCell ← cell of p's right child
  if (q is left of splitting line) {
    best = nearNeighbor(q, p.left, leftCell, best)
  } else {
    best = nearNeighbor(q, p.right, rightCell, best)
  }
}
```

return best
Announcements: 11/03

- Updated Prog 2 Skeleton/Handout
- Homework 3 - Soon