Example:

Kd-Tree Node:

```java
class KDNode {
    Point pt // splitting point
    int cutDim // cutting coordinate
    KDNode left // low side
    KDNode right // high side
}
```

Balanced kd-tree:
- Cutting dimension alternates (x, y, x, y, ...)
- Balanced
  - \( \text{size}(p.\text{child}) \leq \frac{1}{2} \cdot \text{size}(p) \)

How do we choose cutting dim?:
- Standard Kd-tree: cycle through them (e.g. \( d = 3: 1, 2, 3, 1, 2, 3, ... \)) based on tree depth
- Optimized Kd-tree (Bentley)
  - Based on widest dimension of pts in cell.

Quad trees & kd-Trees III

Helper:
```java
class KDNode {
    boolean onLeft(Point q) {
        \( \text{return } q[\text{cutDim}] < pt[\text{cutDim}] \)}
    }
```
KDNode insert(Point pt, KDNode p, int cd){
    if (p == null)  // fell out?
        p = new KDNode(pt, cd)
        // new leaf node
    else if (p.point == pt)
        Error! Duplicate key
    else if (p.onLeft(pt))
        p.left = insert(pt, p.left, (cd+1)%dim)
    else
        p.right = insert(pt, p.right, (cd+1)%dim)
    return p
}

Kd-Tree Insertion:
(Similar to std. BSTs)
- Descend tree until cutting dimension to use
  → find pt → Error - duplicate
  → falling out
  (Although we draw extended trees, lets assume standard trees)
  → create new node
  → set cutting dim
  → find replacement
  → recur. delete replacement

Example:

Quadtrees & Kd-Trees IV

Analysis:
Runtime: $O(h)$

Can we balance the tree?
-Rotation does not make sense!!

Deletion:
- Descend path to leaf
- If found:
  - leaf node → just remove
  - internal node
    → find replacement
    → copy here
    → recur. delete replacement

Rebalance by Rebuilding:
- Rebuild subtrees as with scapegoat trees
- $O(\log n)$ amortized
- Find: $O(\log n)$ guaranteed.

Although we draw extended trees, lets assume standard trees.

This is the hardest part. See Latex notes.

Rotation does not make sense!!
Replacement Node for Deletion
- Assume deleted node is vertical (x) splitter
- Find smallest x-coord in right subtree

- Utility: \( \text{findMin}(p, q) \)
  - cut dim \( 0 \rightarrow x \)
  - \( 1 \rightarrow y \)
  - node \( p \rightarrow q \)
  - \( p \rightarrow \text{right} \)

\( \text{delete}(q) \)

\( \Delta \)

\( o \)
Kd-Trees:
- Partition trees
- Orthogonal split
- Alternate cutting
dimension $x,y,z,...$

Cells are axis-aligned rectangles (AABB)

Queries?
- 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 $q$

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

General $\mathbb{R}^d$: $O(n^{1-1/d})$

Rectangle methods for kd-cells:
- Split a cell $r$ by a split pt $s \in r$, along cut dim $d$
- Left part
- Right part

Kd-Tree Queries

Axis-Aligned Rect $m \mathbb{R}^d$
- 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 $r,c$ - Rectangle $q$ - Point
- $r$ contains $q$
- $r$ contains $c$
- $r$ is disjoint from $c$

General $\mathbb{R}^d$: $O(n^{1-1/d})$
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 → 0
- Query rect is disjoint from p's cell → return 0
  → no point of p contributes to answer
- Query rect contains p's cell → return p.size
  → every point of p's subtree contributes to answer
- Otherwise: Rect cell overlap - Recurse on both children

class Rectangle {
    private Point low, high
    public Rect (Point l, Point h)
    " boolean contains(Point p)
    " boolean contains(Rect c)
    " Rect leftPart(int cd, Points)
    " Rect rightPart()
}

Kd-Tree Queries

int rangeCount(Rect R, KDNode p, Rect cell)
if (p == null) return 0 // fell out of tree
else if (R is Disjoint From (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)
    return ct
}
Theorem: Given a balanced kd-tree storing n pts in \( \mathbb{R}^2 \) (using alternating cut dim), orthog. range queries can be answered in \( O(n) \) time.

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

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

Master Thm:
\[
T(n) = aT\left(\frac{n}{b}\right) + \Theta(d) \log_b a
\]

For us: \( a = 2 \)
\[
T(n) = n \log_2 n
\]

Simpler: Extend R's sides to 4 lines+analyze each one.

Lemma: Given a kd-tree (as in Thm above) and horiz. or vert. line \( l \), at most \( O\left(\sqrt{n}\right) \) cells can be stabbed by \( l \).

Proof: w.l.o.g. \( l \) horiz.
- Cases: \( p \) splits vertically
- \( \Rightarrow \) Stab both

Stabbing: 3 cases
- cell is disjoint (easy)
- cell is contained (easy)
- cell partially overlaps or is stabbed by the query range (hard!)

Kd-Tree Queries III

Since tree is balanced a child has half the pts + grandchild has quarter.
Recurrence: \( T(n) = 2 + 2T\left(\frac{n}{4}\right) \)

If we consider 2 consecutive levels of kd-tree, \( l \) stabs at most 2 of 4 cells:
- p splits horizontally
- \( l \) stabs only one...