Geometric Search:
- Nearest neighbors
- Range searching
- Point Location
- Intersection Search

Sofar: 1-dimensional keys
- Multi-dimensional data
- Applications:
  - Spatial databases + maps
  - Robotics + Auton. Systems
  - Vision/Graphics/Games
  - Machine Learning

Partition Trees:
- Tree structure based on hierarchical space partition
  - Each node is associated with a region - cell
  - Each internal node stores a splitter - subdivides the cell

Quadtrees & kd-Trees I
- External nodes store pts.

Point: A d-vector in \( \mathbb{R}^d \)
\( p = (p_0, \ldots, p_d) \) \( p \in \mathbb{R}^d \)

Representations:
- Scalars: Real numbers for coordinates, etc.
- Points: \( p = (p_0, \ldots, p_d) \) in real d-dim space \( \mathbb{R}^d \)
- Other geom objects

Multi-Dim vs. 1-dim Search?

Similarities:
- Tree structure
- Balance \( O(\log n) \)
- Internal nodes - split
- External nodes - data

Differences:
- No(natural) total order
- Need other ways to discriminate + separate
- Tree rotation may not be meaningful

Built from these

Intersection search
- Machine Learning
- Spatial databases + maps
- Maps
- Each node is associated with a region - cell

Class Point

float[] coord // coords
Point(int d)

int getDim() \rightarrow \text{coord.length}
float get(int i) \rightarrow \text{coord[i]}

... others: equality, distance

toString...
**Quadtree**

- Each internal node stores a point
- Cell is split by horiz. + vert. lines through point

- **Quadtree (abstractly)**
  - **Partition trees**
  - **Cell:** Axis-parallel rectangle
  - **Splitter:** Subdivides cell into four subcells

- **Quadtree & kd-Trees**

**Find/Pt Location:**

Given a query point $q$, is it in tree, and if not which leaf cell contains it?

→ Follow path from root down (generalizing BST find)

**History:** Bentley 1975

- Called it 2-d tree ($\mathbb{R}^2$)
- 3-d tree ($\mathbb{R}^3$)
- In short $kd$-tree (any dim)

- Where/which direction to split? → next

**kd-Tree:** Binary variant of quadtree

- **Splitter:** Horiz. or vert. line in 2-d (orthogonal plane $\mathbb{R}^2$)
- **Cell:** Still AABB

Left: left/below right: right/above

- **Numerous variants!**
  - $PR, PMR, QR, QX$ ...
  - Popular in 2-d apps (in 3-d, octtrees)

- Don't scale to high dim

- What to do for higher dims?
Kd-Tree Node:

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

Example:

```
find(q) calls find(q, root)
```

```
Example: find(g) → find(g, root)
```

Analysis:
Find runs in time $O(h)$, where $h$ is height of tree.

Theorem: If pts are inserted in random order, expected height is $O(\log n)$.

Value:

```
find(Point q, KDNode p) {
    if (p == null) return null;
    if (q == p.point) return p.val;
    if (q[cutDim] < p.point[cutDim]) return find(q, p.left);
    if (q[cutDim] > p.point[cutDim]) return find(q, p.right);
    return find(q, p.left);
}
```

Quad trees & kd-trees

How do we choose cutting dim?
- Standard kd-tree: cycle through them (e.g. $d = 3$: x, y, z, x, y, z, ...)
- Optimized kd-tree (Bentley): based on widest dimension of pts in cell.
- Based on tree depth

Helper:
```
class KDNode {
    boolean onLeft(Point q) {
        return q[cutDim] < pt[cutDim];
    }
}
```
KD-Tree Insertion:

(Similar to std. BSTs)

- Descend tree until cutting.
  - Find pt → Error: duplicate
  - Falling out → create new node
  - Set cutting dim

Deletion:

- Descend path to leaf
- If found:
  - Leaf node → just remove internal node
  - Copy replacement
  - Recur. delete replacement

Quadtrees & kd-Trees

Example:

```
insert(3,4)
```

Analysis:

Runtime: $O(h)$

- Rebalance by rebuilding subtrees as they become unbalanced

Can we balance the tree?

Rotation does not make sense!!
**Kd-Trees:**
- **Partition trees**
- **Orthogonal split**
- **Alternate cutting**

**Rectangle methods for kd-cells:**
- Split a cell \( r \) by a split \( pt \ s \in r \), along cut \( \dim cd \)
  \[ r_{\text{low}} \]
  \[ r_{\text{high}} \]

**Queries?**
- **Orthogonal range queries**
  - Given query rect \( (\text{AABB}) \)
  - Count/report pts in this rect.
- **Other range queries?**
  - Circular disks
  - Half-plane

**Nearest neighbor queries**
- Given query pt, return closest pt in the set
- Find \( k \)th closest point
- Find farthest point in \( q \)

**Kd-Tree Queries**

**Axis-Aligned Rect 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 \( r, c \) - Rectangle
  \( q \) - Point
  - \( r \).contains \( (q) \)
  - \( r \).contains \( (c) \)
  - \( r \) is disjoint from \( (c) \)
Public class XkdTree {

abstract class Node {
    // generic funcs for both int & ext
}

class InternalNode extends Node {
    int cutDim
    double cutVal
    // insert helper
}

class ExternalNode extends Node {
    ArrayList<Point> pts
    // insert helper
}

private Node root

public void insert(Point pt) {
    root. insert(pt)
}