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
- External nodes store pts.

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

Point: A d-vector in $\mathbb{R}^d$

Class Point

```
float[] coord // coords
Point(int d)
...
coord = new float[d]
```

int getDim() -> coord.length
float get(int i) -> coord[i]

... others: equality, distance toString...
Point Quadtree:
- Each internal node stores a point
- Cell is split by horiz. + vertic. lines through point

(5,4) (2,2) (7,3) (4,1)

Quadtree: (abstractly)
- Partition trees
- Cell: Axis-parallel rectangle
  
  - Splitter: Subdivides cell into four (generally 2^d) subcells

Quadtree & kd-Trees II

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:
- Splitter: Horiz. or vertic. line in 2-d (orthogonal plane ow.)
- Cell: Still AABB
  left: left/below right: right/above

kd-Tree: Binary variant of quadtree

Quadtree Analysis:
- Numerous variants!
  PR, PMR, QR, QX, ... see Samet's book
- Popular in 2-d apps
  (in 3-d, octtrees)
- Don't scale to high dim
  out degree = 2^d
- What to do for higher dims?
**Kd-Tree Node:**

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

**Example:**

Find \( q \) in subtree rooted at \( p \) with \( \text{cutDim} = d \):
- if \( q \) matches \( p \):
  - return \( p \).
- if \( q[\text{cutDim}] < p\text{.point}[\text{cutDim}] \): return \( p \text{.left} \)
- else return \( p \text{.right} \)

**Example:**

Find \( q \) calls \( \text{find}(q, \text{root}) \)

**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**

\[ \text{find}(\text{Point} q, \text{KDNode} p) \{
    \text{if (} p == \text{null} \text{) return null;}
    \text{else if (} q == p\text{.pt} \text{)} \text{... all coords match?}
    \text{return } p\text{.value;}
    \text{else if (} p\text{.onLeft}(q) \text{)} \text{...}
    \text{return } \text{find}(q, p\text{.left});
    \text{else return } \text{find}(q, p\text{.right});
\}

**Quad trees & kd-Trees III**

- Standard kd-tree: cycle through them (e.g. \( d = 3: 1,2,3,2,3,.. \)) based on tree depth.
- Optimized kd-tree (Bentley): based on widest dimension of pts in cell.

**How do we choose cutting dim?**

- Standard kd-tree: cycle through them (e.g. \( d = 3: 1,2,3,2,3,.. \)) based on tree depth.
- Optimized kd-tree (Bentley): based on widest dimension of pts in cell.
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)
    return p
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. → find pt → Error: duplicate
cutting. → falling out
-Create new node
-Set cutting dim

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

Quadtrees & KD-Trees IV

This is the hardest part. See Latex notes.

Rebalance by Rebuilding:
- Rebuild subtrees as they become unbalanced
- $O(h)$ amortized
- Find: $O(h)$ guaranteed

Example:
insert(3,4)

Run time: $O(h)$

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

Rotation does not make sense!!
Announcements: 10/25

- HWZ + Exam grading - End of this week
- Next midterm, Nov 17
- Proj. Assign 2 - Soon
  \[ \rightarrow \text{kd-Tree} \]
- Final exam - Thu, Dec 15 8-10am
  \[ \rightarrow \text{As in syllabus} \]
  \[ \rightarrow \text{Final exam schedule} \]
  \[ \text{Not} \quad \text{common exam for 420} \]