K-D Trees and Quad Trees

CSE 326
Data Structures
Lecture 9

Reading
- Chapter 12.6

Geometric Data Structures
- Organization of points, lines, planes, … to support faster processing
- Applications
  - Astrophysical simulation – evolution of galaxies
  - Graphics – computing object intersections
  - Data compression
    - Points are representatives of 2x2 blocks in an image
    - Nearest neighbor search

k-d Trees
- Jon Bentley, 1975, while an undergraduate
- Tree used to store spatial data.
  - Nearest neighbor search.
  - Range queries.
  - Fast look-up
- k-d tree are guaranteed log₂ n depth where n is the number of points in the set.
  - Traditionally, k-d trees store points in d-dimensional space which are equivalent to vectors in d-dimensional space.

Range Queries
- Rectangular query
- Circular query

Nearest Neighbor Search
- Nearest neighbor is e.
k-d Tree Construction

- If there is just one point, form a leaf with that point.
- Otherwise, divide the points in half by a line perpendicular to one of the axes.
- Recursively construct k-d trees for the two sets of points.
- Division strategies
  - divide points perpendicular to the axis with widest spread.
  - divide in a round-robin fashion (book does it this way)

k-d Tree Construction (1)

divide perpendicular to the widest spread.

k-d Tree Construction (2)

k-d Tree Construction (3)

k-d Tree Construction (4)

k-d Tree Construction (5)
k-d Tree Construction (18)

2-d Tree Decomposition

k-d Tree Splitting

k-d Tree Construction Complexity

Node Structure for k-d Trees

- First sort the points in each dimension.
  - $O(dn \log n)$ time and $dn$ storage.
  - These are stored in $A[1..d,1..n]$
- Finding the widest spread and equally divide into two subsets can be done in $O(dn)$ time.
- We have the recurrence
  - $T(n,d) \leq 2T(n/2,d) + O(dn)$
- Constructing the k-d tree can be done in $O(dn \log n)$ and $dn$ storage.

- A node has 5 fields
  - axis (splitting axis)
  - value (splitting value)
  - left (left subtree)
  - right (right subtree)
  - point (holds a point if left and right children are null)
Rectangular Range Query

- Recursively search every cell that intersects the rectangle.
Rectangular Range Query

```
print_range(xlow, xhigh, ylow, yhigh :integer, root: node pointer) {
    Case {
        root = null: return;
        root.left = null:
        if xlow < root.point.x and root.point.x < xhigh
            and ylow < root.point.y and root.point.y < yhigh
            then print(root);
        else
            if(root.axis = "x" and xlow < root.value ) or 
            (root.axis = "y" and ylow < root.value )
            then print_range(xlow, xhigh, ylow, yhigh, root.left);
            if (root.axis = "x" and xlow > root.value ) or 
            (root.axis = "y" and ylow > root.value )
            then print_range(xlow, xhigh, ylow, yhigh, root.right);
    }
}
```

Analysis of Rectangular Range Query
- Worst case time is $O(n)$ as seen by the pathological example.

k-d Tree Nearest Neighbor Search
- Search recursively to find the point in the same cell as the query.
- On the return search each subtree where a closer point than the one you already know about might be found.
**Main is NNS(q,root,null,infinity)**

**Nearest Neighbor Search**

\[
\text{NNS}(q: \text{point}, n: \text{node}, p: \text{point}, w: \text{distance}) : \text{point} = \begin{cases} 
\text{if } n.\text{left} = \text{null} \text{ then } \text{(leaf case)} \\
\text{if distance}(q,n.\text{point}) < w \text{ then return } n.\text{point} \text{ else return } p; \\
\text{else} \\
\text{if } w = \infty \text{ then} \\
\text{if } q(n.\text{axis}) < n.\text{value} \text{ then } p := \text{NNS}(q,n.\text{left},p,w); \\
\text{else } p := \text{NNS}(q,n.\text{right},p,w); \\
\text{if distance}(p,q) < n.\text{value} \text{ then return } p; \\
\text{else} \\
\text{if } q(n.\text{axis}) + w > n.\text{value} \text{ then } p := \text{NNS}(q,n.\text{right},p,w); \\
\text{else } p := \text{NNS}(q,n.\text{left},p,w); \\
\text{if distance}(p,q) < n.\text{value} \text{ then return } p; \end{cases}
\]

**The Conditional**

\[q(n.\text{axis}) + w > n.\text{value}\]

**Worst-Case for Nearest Neighbor Search**

- Half of the points visited for a query
- Worst case \(O(n)\)
- But, on average (and in practice) nearest neighbor queries are \(O(\log N)\)

**Notes on \(k\)-d NNS**

- Has been shown to run in \(O(\log n)\) average time per search in a reasonable model. (Assume \(d\) a constant)
- Storage for the \(k\)-d tree is \(O(n)\).
- Preprocessing time is \(O(n \log n)\) assuming \(d\) is a constant.

**Quad Trees**

- Space Partitioning
Quad Trees

- Space Partitioning

A Bad Case

Notes on Quad Trees

- Number of nodes is $O(n(1 + \log(\Delta/n)))$ where $n$ is the number of points and $\Delta$ is the ratio of the width (or height) of the key space and the smallest distance between two points
- Height of the tree is $O(\log n + \log \Delta)$

K-D vs Quad

- k-D Trees
  - Density balanced trees
  - Height of the tree is $O(\log n)$ with batch insertion
  - Good choice for high dimension
  - Supports insert, find, nearest neighbor, range queries
- Quad Trees
  - Space partitioning tree
  - May not be balanced
  - Not a good choice for high dimension
  - Supports insert, delete, find, nearest neighbor, range queries

Geometric Data Structures

- Geometric data structures are common.
- The k-d tree is one of the simplest.
  - Nearest neighbor search
  - Range queries
- Other data structures used for
  - 3-d graphics models
  - Physical simulations