K-D Trees

James Fogarty

Spring 2009
Range Queries

• Think of a range query.
  – “Give me all customers aged 45-55.”
  – “Give me all accounts worth $5m to $15m”

• Assume you have stored your information in a tree according to the desired key

• How do you answer this query?
Range Query
Range Queries

• How expensive was that?

• Binary search trees are not really designed for range queries, and it gets worse

• What if we want both:
  – “Give me all customers aged 45-55 with accounts worth between $5m and $15m.”
2D Range Query
Geometric Data Structures

• Organization of points, lines, planes, etc. in support of faster processing

• Applications
  – Map information
  – Graphics - computing object intersections
  – Data compression - nearest neighbor search
  – Decision Trees - machine learning
Quad Trees

• Space Partitioning
Quad Trees

• Space Partitioning
Quad Trees

- Space Partitioning

What’s different about this tree structure versus the others we have seen?
A Bad Case

\[\text{Diagram of a bad case with points labeled}\]
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 \( O(\log n + \log \Delta) \)

• Let's consider something based on density
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 $\log_2 n$ 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 (6)
k-d Tree Construction (7)
k-d Tree Construction (8)
k-d Tree Construction (9)
k-d Tree Construction (10)
k-d Tree Construction (11)
k-d Tree Construction (12)
k-d Tree Construction (13)
k-d Tree Construction (14)
k-d Tree Construction (15)
k-d Tree Construction (16)
k-d Tree Construction (17)
k-d Tree Construction (18)
2-d Tree Decomposition
k-d Tree Splitting

sorted points in each dimension

1  2  3  4  5  6  7  8  9

x  a  d  g  b  e  i  c  h  f
y  a  c  b  d  f  e  h  g  i

• max spread is the max of \( f_x - a_x \) and \( i_y - a_y \).

• In the selected dimension the middle point in the list splits the data.

• To build the sorted lists for the other dimensions scan the sorted list adding each point to one of two sorted lists.
k-d Tree Splitting

sorted points in each dimension

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>a</td>
<td>d</td>
<td>g</td>
<td>b</td>
<td>e</td>
<td>i</td>
<td>c</td>
<td>h</td>
<td>f</td>
</tr>
<tr>
<td>y</td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>d</td>
<td>f</td>
<td>e</td>
<td>h</td>
<td>g</td>
<td>i</td>
</tr>
</tbody>
</table>

indicator for each set

```
[0 0 1 0 0 1 0 1 1]
```

scan sorted points in y dimension and add to correct set

```
y [a b d e g c f h i]
```
k-d Tree Construction Complexity

• 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) ≤ 2T(n/2,d) + O(dn)

• Constructing the k-d tree can be done in O(dn log n) and dn storage
Node Structure for k-d Trees

- 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 (1)
Rectangular Range Query (2)
Rectangular Range Query (3)
Rectangular Range Query (4)
Rectangular Range Query (5)
Rectangular Range Query (6)
Rectangular Range Query (7)
Rectangular Range Query (8)
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);
    }
}
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.
k-d Tree NNS (1)

query point
k-d Tree NNS (2)

query point

---

The diagram illustrates a k-d tree with nodes labeled from s1 to s8. The tree structure is depicted with nodes partitioned by x and y coordinates. The left side of the diagram shows a 2D space divided into sections s1, s2, s3, s4, s5, s6, s7, and s8, each containing points labeled a, b, d, e, g, h, i, and c. The right side of the diagram illustrates the hierarchical organization of the tree, with nodes s1, s2, s3, s4, s5, s6, s7, and s8, and leaves representing the points a, b, c, d, e, g, c, f, h, and i.
k-d Tree NNS (3)

query point
k-d Tree NNS (4)
k-d Tree NNS (5)

query point
k-d Tree NNS (6)

query point

---

A diagram showing a k-d tree with nodes and points, illustrating the process of finding the nearest neighbors. The tree splits the space into regions along the x and y axes, with points positioned accordingly. The query point is marked, and the nearest neighbors are highlighted.
k-d Tree NNS (7)

query point
k-d Tree NNS (10)
k-d Tree NNS (11)
k-d Tree NNS (12)
k-d Tree NNS (13)
k-d Tree NNS (14)
k-d Tree NNS (15)

query point
Notes on k-d NNS

• Has been shown to run in $O(\log n)$ average time per search in a reasonable model.

• Storage for the k-d tree is $O(n)$.

• Preprocessing time is $O(n \log n)$ assuming $d$ is a constant.
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 $O(\log N)$
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