Orthogonal Range Searching

slides by Andy Mirzaian
(a subset of the original slides are used here)
References:

• [M. de Berge et al] chapter 5

Applications:

• Spatial Databases
• GIS, Graphics: crop-&-zoom, windowing
Orthogonal Range Search: Database Query

2D Query Rectangle \([1980,00,00 : 1989,99,99] \times [13,000 : 14,000]\)

3D Query Orthogonal Range \([1980,00,00 : 1989,99,99] \times [13,000 : 14,000] \times [2 : 4]\)

Mr. G. O. Meter
born: Nov. 6, 1988
Salary: $13,600.
1D-Tree: 1-Dimensional Range Searching

Static:  Binary Search in a sorted array.
Dynamic: Store data points in some balanced Binary Search Tree $T$.

Let the data points be $P = \{ p_1, p_2 , \ldots , p_n \} \subseteq \mathcal{R}$. $T$ is a balanced BST where the data appear at its leaves sorted left to right. The internal nodes are used to split left & right subtrees. Assume $x(v) = \max x(L)$, where $L$ is any leaf in the left subtree of internal node $v$.

Query Range: $[7 : 49]$
**Algorithm 1DRangeQuery**

```
if v is a leaf
   then if x \leq x(v) \leq x' then report data stored at v
else do
   if x \leq x(v) then 1DRangeQuery (leftchild(v), x, x')
   if x(v) < x' then 1DRangeQuery (rightchild(v), x, x')
end
```

**Complexities:**

- **Query Time**: $O(K + \log n)\quad \mathcal{T},[x,x'] \rightarrow \text{output}$
- **Construction Time**: $O(n \log n)\quad P \rightarrow \mathcal{T}$
- **Space**: $O(n)\quad \text{store } \mathcal{T}$

[These are optimal]
Consider dimension $d=2$:

point $p = (x(p), y(p))$, range $R = [x_1 : x_2] \times [y_1 : y_2]$

$p \in R \iff x(p) \in [x_1 : x_2]$ and $y(p) \in [y_1 : y_2]$.

$L = \text{vertical/horizontal median split.}$
Alternate between vertical & horizontal splitting at even and odd depths.
(Assume: no 2 points have equal x or y coordinates.)
Constructing 2D-Tree

Input: \( P = \{ p_1, p_2, \ldots, p_n \} \subseteq \mathbb{R}^2 \) off-line.
Output: 2D-tree storing \( P \).

Step 1: Pre-sort \( P \) on \( x \) & on \( y \), i.e., 2 sorted lists \( \hat{U} = (X\text{sorted}(P), Y\text{sorted}(P)) \).
Step 2: \( \text{root}[\mathcal{T}] \leftarrow \text{Build2DTree} ( \hat{U}, 0 ) \)

Procedure \text{Build2DTree} ( \hat{U}, \text{depth} )

if \( \hat{U} \) contains one point then return a leaf storing this point
else do
  if depth is even then x-median split \( \hat{U} \), i.e., split data points in half by a \text{vertical} line \( L \)
    through x-median of \( \hat{U} \) and reconfigure \( \hat{U}_{\text{left}} \) and \( \hat{U}_{\text{right}} \).
  else y-median split \( \hat{U} \), … by a \text{horizontal} line \( L \),
    and reconfigure \( \hat{U}_{\text{left}} \) and \( \hat{U}_{\text{right}} \).
  \( v \leftarrow \) a newly created node storing line \( L \)
  \( \text{leftchild}(v) \leftarrow \text{Build2DTree} ( \hat{U}_{\text{left}}, 1+\text{depth} ) \)
  \( \text{rightchild}(v) \leftarrow \text{Build2DTree} ( \hat{U}_{\text{right}}, 1+\text{depth} ) \)
  return \( v \)
end

\( T(n) = 2T(n/2) + O(n) = O(n \log n) \) time.
2D-Tree Example
Query Point Search in 2D-Tree
region(v) = rectangular region (possibly unbounded) covered by the subtree rooted at v.

region (root[\mathcal{T}]) = (-\infty : + \infty) \times (-\infty : + \infty)

Suppose region(v) = \langle x_1 : x_2 \rangle \times \langle y_1 : y_2 \rangle
what are region(leftchild(v)) and region(rightchild(v))?

With x-split:
region(lc(v)) = \langle x_1 : x(L) \rangle \times \langle y_1 : y_2 \rangle
region(rc(v)) = \langle x(L) : x_2 \rangle \times \langle y_1 : y_2 \rangle

With y-split:
region(lc(v)) = \langle x_1 : x_2 \rangle \times \langle y_1 : y(L) \rangle
region(rc(v)) = \langle x_1 : x_2 \rangle \times \langle y(L) : y_2 \rangle
2D-Tree Range Search

For range $R = [x_1 : x_2] \times [y_1 : y_2]$ call $\text{Search2DTree} \left( \text{root}[\mathcal{T}] , R \right)$

**Algorithm** $\text{Search2DTree} \left( v , R \right)$

1. if $v$ is a leaf then if $p(v) \in R$ then report $p(v)$
2. else if $\text{region}(\text{l}(v)) \subseteq R$
3. then $\text{ReportSubtree}\left(\text{l}(v)\right)$
4. else if $\text{region}(\text{l}(v)) \cap R \neq \emptyset$
5. then $\text{Search2DTree}\left(\text{l}(v) , R \right)$
6. if $\text{region}(\text{r}(v)) \subseteq R$
7. then $\text{ReportSubtree}\left(\text{r}(v)\right)$
8. else if $\text{region}(\text{r}(v)) \cap R \neq \emptyset$
9. then $\text{Search2DTree}\left(\text{r}(v) , R \right)$
end

- $\text{region}(v)$ can either be passed as input parameter, or explicitly stored at node $v$, $\forall v \in \mathcal{T}$.

- $\text{ReportSubtree}(v)$ is a simple linear-time in-order traversal that reports every leaf descendent of node $v$. 
Running Time of Search2DTree

- K = # of points reported.
- Lines 3 & 7 take O(K) time over all recursive calls.
- Total # nodes visited (reported or not) is proportional to # times conditions of lines 4 & 8 are true.
- region(v) ∩ R ≠ ∅ & region(v) ∉ R ⇔ a bounding edge e of R intersects region(v).
- R has ≤ 4 bounding edges. Let e (assume vertical) be one of them.
- Define H(n) (resp. V(n)) = worst-case number of nodes v that intersect e for a 2D-tree of n leaves, assuming root corresponds to an x-split (resp. y-split).

$$\begin{align*}
\begin{cases}
H(n) = V(n/2) + 1 \\
V(n) = 2H(n/2) + 1
\end{cases}
\Rightarrow
\begin{cases}
H(n) = 2H(n/4) + 2 \\
V(n) = 2V(n/4) + 3
\end{cases}
\Rightarrow
\begin{cases}
H(n) = 3\sqrt{n} - 2 \\
V(n) = 4\sqrt{n} - 3
\end{cases}
\Rightarrow \text{ Running Time} = O(K + \sqrt{n}).
\end{align*}$$
**dD-Tree Complexities**

### 2D-Tree
- **Query Time:** $O(K + \sqrt{n})$ worst-case, $O(K + \log n)$ average
- **Construction Time:** $O(n \log n)$
- **Storage Space:** $O(n)$

### dD-Tree d-dimensions
Use round-robin splitting at successive levels on the $d$ dimensions $x_1, x_2, \ldots, x_d$.

- **Query Time:** $O(dK + d n^{1-1/d})$
- **Construction Time:** $O(d n \log n)$
- **Space:** $O(dn)$

**How can we improve the query time?**
2D-Tree

- Query Time: $O(K + \sqrt{n})$
- Construction Time: $O(n \log n)$
- Space: $O(n)$

2D Range Tree

- Query Time: $O(K + \log^2 n)$
  - $O(K + \log n)$ by Fractional Cascading
- Construction Time: $O(n \log n)$
- Space: $O(n \log n)$

Range $R = [x : x'] \times [y : y']$

1D Range Tree on $x$-coordinates:

- $O(\log n)$
- Each $x$-range $[x : x']$ can be expressed as the disjoint union of $O(\log n)$ canonical $x$-ranges.
Range Trees

2-level data structure:

Primary Level:
BST on x-coordinates

Secondary level:
BST on y-coordinates

\( \text{min}(v) \)
\( \text{max}(v) \)

Primary Level: \( \text{root}[\mathcal{T}] \)

Secondary Level: \( \mathcal{T}_{\text{assoc}}(v) \)

\( P(v) \)
ALGORITHM Build 2D Range Tree (P)

Input: \( P = \{ p_1, p_2, \ldots, p_n \} \subseteq \mathbb{R}^2 \), \( P = (P_x, P_y) \)
represented by pre-sorted list on x (named \( P_x \)) and on y (named \( P_y \)).

Output: pointer to the root of 2D range tree for P.

Construct \( T_{assoc} \), bottom up, based on \( P_y \),
but store in each leaf the points, not just their y-coordinates.

if \( |P| > 1 \) then do
\( P_{left} \leftarrow \{ p \in P | p_x \leq x_{med} \text{ of } P \} \) (* both lists \( P_x \) and \( P_y \) should split *)
\( P_{right} \leftarrow \{ p \in P | p_x > x_{med} \text{ of } P \} \)
\( \text{lc}(v) \leftarrow \text{Build 2D Range Tree (} P_{left} \text{)} \)
\( \text{rc}(v) \leftarrow \text{Build 2D Range Tree (} P_{right} \text{)} \)

od
\( \text{min}(v) \leftarrow \text{min}(P_x) \); \( \text{max}(v) \leftarrow \text{max}(P_x) \)
\( T_{assoc}(v) \leftarrow T_{assoc} \)

return \( v \)
end

\( T(n) = 2T(n/2) + O(n) = O(n \log n) \) time.
This includes time for pre-sorting.
2D Range Query

ALGORITHM 2DRangeQuery ( v, [x : x’] × [y : y’] )
1. if x ≤ min(v) & max(v) ≤ x’
2. then 1DRangeQuery ( T\text{assoc}(v) , [y : y’] )
3. else if v is not a leaf do
4. if x ≤ max(lc(v))
5. then 2DRangeQuery ( lc(v), [x : x’] × [y : y’] )
6. if min(rc(v)) ≤ x’
7. then 2DRangeQuery ( rc(v), [x : x’] × [y : y’] )
8. od
end

• Line 2 called at roots of red canonical sub-trees, a total of O(log n) times. Each call takes O(K_v + log | T\text{assoc}(v) | ) = O(K_v + log n) time.
• Lines 5 & 7 called at blue shoulder paths. Total cost O(log n).
• Total Query Time = O(log n + \sum_v(K_v + log n)) = O(\sum_v K_v + \log^2 n) = O(K + \log^2 n).

Query Time: O( K + \log^2 n ) will be improved to O(K + log n) by Fractional Cascading
Construction Time: O(n log n)
Space: O(n log n)
Higher Dimensional Range Trees

\[ P = \{ p_1, p_2, \ldots, p_n \} \subseteq \mathbb{R}^d, \quad p_i = (x_{i1}, x_{i2}, \ldots, x_{id}), \quad i=1..n. \]
Higher Dimensional Range Trees

d-level data structure
Higher Dimensional Range Trees

Query Time: \( Q_d(n) = O(K + \log^d n) \) improved to \( O(K + \log^{d-1} n) \) by Frac. Casc.

Construction Time: \( T_d(n) = O(n \log^{d-1} n) \)

Space: \( S_d(n) = O(n \log^{d-1} n) \)

\[
\begin{align*}
T_d(n) &= 2T_d\left(\frac{n}{2}\right) + T_{d-1}(n) + O(n) \\
T_2(n) &= O(n \log n)
\end{align*}
\]

\( \Rightarrow T_d(n) = O(n \log^{d-1} n) \)

\[
\begin{align*}
S_d(n) &= 2S_d\left(\frac{n}{2}\right) + S_{d-1}(n) + O(1) \\
S_2(n) &= O(n \log n)
\end{align*}
\]

\( \Rightarrow S_d(n) = O(n \log^{d-1} n) \)

\[
\begin{align*}
Q_d(n) &= O(K) + \hat{Q}_d(n) \\
\hat{Q}_d(n) &= O(\log n) + O(\log n) \cdot \hat{Q}_{d-1}(n) \\
\hat{Q}_2(n) &= O(\log^2 n)
\end{align*}
\]

\( \Rightarrow \)

\[
\begin{align*}
\hat{Q}_d(n) &= O(\log^d n) \\
Q_d(n) &= O(K + \log^d n)
\end{align*}
\]
IDEA: Save repeated cost of binary search in many sorted lists for the same range \([y : y']\) if the list contents for one are a subset of the other.

- \(A_2 \subseteq A_1\)
- Binary search for \(y\) in \(A_1\) to get to \(A_1[i]\).
- Follow pointer to \(A_2\) to get to \(A_2[j]\).
- Now walk to the right in each list.

```
A_1:  1  3  5  7  9  13  15  19  23  26  31  36  45  63  92
A_2:  5  13  26  36  45  nil  nil
```
Fractional Cascading

- $A_2 \subseteq A_1$, $A_3 \subseteq A_1$.
- No binary search in $A_2$ and $A_3$ is needed.
- Do binary search in $A_1$.
- Follow blue and red pointers from there to $A_2$ and $A_3$.
- Now we have the starting point in each sorted list. Walk to the right & report.
Layered 2D Range Tree

\[ T \]

\[ T_{assoc}(v) \]

\[ T_{assoc}(lc(v)) \]

\[ T_{assoc}(rc(v)) \]

\[ P(lc(v)) \subseteq P(v) \]

\[ P(rc(v)) \subseteq P(v) \]
Layered 2D Range Tree

Associated Structures at the secondary level by Fractional Cascading
Layered 2D Range Tree (by Fractional Cascading)

Query Time:
\[
Q_2(n) = O(\log n + \sum_v (K_v + \log n)) = O(\sum_v K_v + \log^2 n) = O(K + \log^2 n)
\]

improves to:
\[
Q_2(n) = O(\log n + \sum_v (K_v + 1)) = O(\sum_v K_v + \log n) = O(K + \log n).
\]

For d-dimensional range tree query time improves to:

\[
\begin{align*}
Q_d(n) &= O(K) + \hat{Q}_d(n) \\
\hat{Q}_d(n) &= O(\log n) + O(\log n) \cdot \hat{Q}_{d-1}(n) \\
\hat{Q}_2(n) &= O(\log n)
\end{align*}
\]

\[
\Rightarrow Q_d(n) = O(K + \log^{d-1} n)
\]