Computational Geometry

Orthogonal Range Queries or Fast Access to Data Bases

Lecture #5

Dr. Philipp Kindermann

Winter Semester 2018/19
Orthogonal Range Queries

**Example:** Personnel management in a company

Typical queries for data bases!
1D Range Searching

Task: Preprocess a finite set \( P \subset \mathbb{R} \) such that for any interval \([x, x']\) the set \( P \cap [x, x']\) can be reported quickly.

Solution: balanced binary search trees…

1. Search \( x = 6 \).
2. Search \( x' = 21 \).
3. Return all leaves ‘inbetween’.

Small changes: – keys only in leaves
 – inner nodes store max. of their left subtrees
1D Range Searching

Observe: The result of a query is the disjoint union of at most $2h$ canonical subsets, where
- $h \in O(\log n)$ is the tree height,
- a canonical subset is an interval that contains all points stored in a subtree.

Theorem. A set of $n$ real numbers can be preprocessed in $O(n \log n)$ time and $O(n)$ space such that 1D range queries take $O(k + \log n)$ time, where $k = |\text{output}|$. 

output sensitive!
Extensions to 2D

Think... [3 min]

Task: Preprocess a finite set $P \subset \mathbb{R}^2$ such that for any range query $R = [x, x'] \times [y, y']$ the set $P \cap R$ can be reported quickly.

Solutions:

- one tree; query path alternates between $x$- and $y$-coord. \{ $kd$-tree \}
- first-level tree for $x$-coordinates; many second-level trees for $y$-coord. \{ range tree \}

Assume: General position!

Here: no two points have the same $x$- or $y$-coordinate.
• Split any region that contains more than one point.
• Horizontal split lines/segm. belong to the region below. Vertical left.
Kd-Trees: Construction

Pseudo-code:

```
BuildKdTree(points P, int depth)
  if |P| = 1 then
    return (leaf storing the pt in P)
  else
    if depth is even then
      split P with the vertical line
      ℓ: x = x_{median(P)} into
      P_1 (pts left of or on ℓ) and
      P_2 = P \ P_1
    else
      split P horizontally...
      v_{left} ← BuildKdTree(P_1, depth + 1)
      v_{right} ← BuildKdTree(P_2, depth + 1)
    create a node v storing ℓ
    make v_{left} and v_{right} the children of v
  return (v)
```
Kd-Trees: Analysis

Construction time?

\[ T(n) = \begin{cases} 
O(1) & \text{if } n = 1 \\
O(n) + 2T(\lceil n/2 \rceil) & \text{else.} 
\end{cases} \]

\[ = O(n \log n) \]

Lemma: A kd-tree for a set of \( n \) pts in the plane takes \( O(n \log n) \) time to construct and uses \( O(n) \) storage.

See Mergesort!
Kd-Trees: Querying

Lemma. Querying a kd-tree for $n$ pts in the plane with an axis-parallel rectangle $R$ takes $O(k + \sqrt{n})$ time, where $k = |output|$.

Idea: $O(\sqrt{n})$ regions of the kd-tree intersect a vertical/horizontal line.
Extensions to 2D

Task: Preprocess a finite set $P \subset \mathbb{R}^2$ such that for any range query $R = [x, x'] \times [y, y']$ the set $P \cap R$ can be reported quickly.

Solutions:

- **one tree;** query path alternates between $x$- and $y$-coord. \{kd-tree\}

- **first-level tree for $x$-coordinates;** many second-level trees for $y$-coord. \{range tree\}

Assume: *General position!* Here: no two points have the same $x$- or $y$-coordinate.
Range Trees: Query Algorithm

1. Search in main tree for $x$-coordinate
2. For each node $u$ on the path from $v_{\text{split}}$ to $\mu$:
   - For the right child $v$ of $u$:
     - Search in auxiliary tree $\mathcal{T}_v$ for points with $y$-coordinate $\in [y, y']$
3. Symmetrically for the path from $v_{\text{split}}$ to $\mu'$.

$P(v) = \text{canonical subset of } \mathcal{T}_v$
Range Trees: Construction

Build2DRangeTree(point[] P)

construct 2nd-level tree $T_P$ on $P$ (y-order)

if $P = \{p\}$ then
  create leaf $v$:
else
  $x_{\text{mid}} = \text{median x-coordinate of } P$
  $P_{\text{left}} = \text{pts in } P \text{ with x-coordinate } \leq x_{\text{mid}}$
  $P_{\text{right}} = \text{pts in } P \text{ with x-coordinate } > x_{\text{mid}}$
  $v_{\text{left}} = \text{Build2DRangeTree}(P_{\text{left}})$
  $v_{\text{right}} = \text{Build2DRangeTree}(P_{\text{right}})$

create node $v$:

return $v$

Running time?

$O(n \log n)$ :-(

Better:
Pre-sort once, then build tree bottom-up in linear time.

Total construction time $O(n \log n)$
Range Trees: Space Consumption

Each node $v$ of the 1st-level tree has a pointer to a 2nd-level tree $T_v$ with $|T_v| = \Theta(|P(v)|)$.

Q: What’s the total space consumption of all 2nd-level trees?

What’s your guess:
- $\Theta(n^2)$,
- $\Theta(n \log n)$,
- $\Theta(n \log^2 n)$, or
- $\Theta(n)$?

A: Each $p \in P$ is stored in $h = \Theta(\log n)$ 2nd-level trees.

$\Rightarrow$ $\Theta(n \log n)$ space
Range Trees: Query time

\[ T(n,k) = \sum_{u \in \text{paths to } \mu \text{ and } \mu'} O(k_u + \log n) \]

\[ = O(\sum u k_u) + O(\sum u \log n) \]

\[ = O(k) + 2h \cdot O(\log n) \]

\[ = O(k + \log^2 n) \]

\[ \mathbb{R}^d? \]

\[ O(n \log^{d-1} n) \] storage and construction time

\[ O(k + \log^d n) \] query time

See Chapter 5.4 in Comp. Geom A&A
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>kd-tree</th>
<th>range tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>construction time</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>storage</td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>query time</td>
<td>$O(k + \sqrt{n})$</td>
<td>$O(k + \log^2 n)$</td>
</tr>
</tbody>
</table>

Note: *trade-off* between space and query time
General Sets of Points

Idea: use composite numbers $(a|b)$ with lex order

$$p = (x, y) \rightarrow \hat{p} = ((x|y), (y|x)) \rightarrow \text{unique coordin.}$$

range $R = [x, x'] \times [y, y']$

$$\hat{R} = [(x| - \infty), (x'| + \infty)] \times [(y| - \infty), (y'| + \infty)]$$

Show: $p \in R \iff \hat{p} \in \hat{R}$

This removes our assumption about the input points being in general position.

We can use kd-trees and range trees for any set of points; no matter how many points have the same $x$- or $y$-coord.
Fractional Cascading

**Task 1:** Given sets $B \subset A \subset \mathbb{N}$ stored in sorted order in arrays $A[1..n]$ and $B[1..m]$, support 1d range queries in the multiset $A \cup B$ in $k + 1 \cdot \log n$ time! We allow $n \log m$ bits extra space.

**Task 2:** Assuming that task 1 can be solved, speed up 2d range queries: $O(k + \log^2 n) \rightarrow O(k + \log n)$ time!
Layered Range Trees

Theorem: Let \( d \geq 2 \) and let \( P \) be a set of \( n \) pts in \( \mathbb{R}^d \). Given \( O(n \log^{d-1} n) \) preprocessing time & storage, \( d \)-dim range queries on \( P \) can be answered in \( O(k + \log^{d-1} n) \) time.