Introduction to Algorithms
6.046J/18.401J/SMA5503

Lecture 12
Prof. Erik Demaine
Computational geometry

Algorithms for solving “geometric problems” in 2D and higher.

Fundamental objects:
- point
- line segment
- line

Basic structures:
- point set
- polygon
Computational geometry

Algorithms for solving “geometric problems” in 2D and higher.

Fundamental objects:
- point
- line segment
- line

Basic structures:
- triangulation
- convex hull
Orthogonal range searching

**Input:** \( n \) points in \( d \) dimensions

- E.g., representing a database of \( n \) records each with \( d \) numeric fields

**Query:** Axis-aligned *box* (in 2D, a rectangle)

- Report on the points inside the box:
  - Are there any points?
  - How many are there?
  - List the points.
Orthogonal range searching

**Input:** $n$ points in $d$ dimensions

**Query:** Axis-aligned box (in 2D, a rectangle)
- Report on the points inside the box

**Goal:** Preprocess points into a data structure to support fast queries
- Primary goal: *Static data structure*
- In 1D, we will also obtain a dynamic data structure supporting insert and delete
1D range searching

In 1D, the query is an interval:

First solution using ideas we know:
- Interval trees
  - Represent each point \( x \) by the interval \([x, x]\).
  - Obtain a dynamic structure that can list \( k \) answers in a query in \( O(k \log n) \) time.
1D range searching

In 1D, the query is an interval:

Second solution using ideas we know:
- Sort the points and store them in an array
- Solve query by binary search on endpoints.
- Obtain a static structure that can list $k$ answers in a query in $O(k + \lg n)$ time.

Goal: Obtain a dynamic structure that can list $k$ answers in a query in $O(k + \lg n)$ time.
1D range searching

In 1D, the query is an interval:

New solution that extends to higher dimensions:

- Balanced binary search tree
- New organization principle:
  Store points in the *leaves* of the tree.
- Internal nodes store copies of the leaves
to satisfy binary search property:
  - Node $x$ stores in $key[x]$ the maximum
    key of any leaf in the left subtree of $x$.  

© 2001 by Erik D. Demaine
Example of a 1D range tree

© 2001 by Erik D. Demaine

Introduction to Algorithms
Example of a 1D range tree
Example of a 1D range query

\[\text{RANGE-QUERY}([7, 41])\]
General 1D range query

root

split node
Pseudocode, part 1: Find the split node

1D-RANGE-QUERY\((T, [x_1, x_2])\)

\[ w \leftarrow \text{root}[T] \]

\textbf{while} \( w \) is not a leaf \textbf{and} \((x_2 \leq \text{key}[w] \text{ or } \text{key}[w] < x_1)\)

\textbf{do if} \( x_2 \leq \text{key}[w] \)

\hspace{1em} \textbf{then} \( w \leftarrow \text{left}[w] \)

\hspace{1em} \textbf{else} \( w \leftarrow \text{right}[w] \)

\hspace{1em} \triangleright w \text{ is now the split node}

\[ \text{[traverse left and right from } w \text{ and report relevant subtrees]} \]
Pseudocode, part 2: Traverse left and right from split node

1D-RANGE-QUERY($T, [x_1, x_2]$)

[find the split node]
▷ $w$ is now the split node
if $w$ is a leaf
    then output the leaf $w$ if $x_1 \leq \text{key}[w] \leq x_2$
else $v \leftarrow \text{left}[w]$
    ▷ Left traversal
    while $v$ is not a leaf
        do if $x_1 \leq \text{key}[v]$
            then output the subtree rooted at $\text{right}[v]$
                $v \leftarrow \text{left}[v]$
            else $v \leftarrow \text{right}[v]$
        output the leaf $v$ if $x_1 \leq \text{key}[v] \leq x_2$
[ symmetrically for right traversal ]
Analysis of 1D-RANGE-QUERY

**Query time:** Answer to range query represented by $O(lg n)$ subtrees found in $O(lg n)$ time. Thus:

- Can test for points in interval in $O(lg n)$ time.
- Can count points in interval in $O(lg n)$ time if we augment the tree with subtree sizes.
- Can report the first $k$ points in interval in $O(k + lg n)$ time.

**Space:** $O(n)$

**Preprocessing time:** $O(n lg n)$
2D range trees

Store a primary 1D range tree for all the points based on $x$-coordinate.
Thus in $O(lg n)$ time we can find $O(lg n)$ subtrees representing the points with proper $x$-coordinate. How to restrict to points with proper $y$-coordinate?
2D range trees

Idea: In primary 1D range tree of $x$-coordinate, every node stores a secondary 1D range tree based on $y$-coordinate for all points in the subtree of the node. Recursively search within each.
Analysis of 2D range trees

Query time: In $O(lg^2 n) = O((lg n)^2)$ time, we can represent answer to range query by $O(lg^2 n)$ subtrees. Total cost for reporting $k$ points: $O(k + (lg n)^2)$.

Space: The secondary trees at each level of the primary tree together store a copy of the points. Also, each point is present in each secondary tree along the path from the leaf to the root. Either way, we obtain that the space is $O(n lg n)$.

Preprocessing time: $O(n lg n)$
d-dimensional range trees

Each node of the secondary $y$-structure stores a tertiary $z$-structure representing the points in the subtree rooted at the node, etc.

**Query time:** $O(k + \lg^d n)$ to report $k$ points.
**Space:** $O(n \lg^{d-1} n)$
**Preprocessing time:** $O(n \lg^{d-1} n)$

---

**Best data structure to date:**
**Query time:** $O(k + \lg^{d-1} n)$ to report $k$ points.
**Space:** $O(n (\lg n / \lg \lg n)^{d-1})$
**Preprocessing time:** $O(n \lg^{d-1} n)$
Primitive operations: Crossproduct

Given two vectors $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$, is their counterclockwise angle $\theta$

- **convex** ($< 180^\circ$),
- **reflex** ($> 180^\circ$), or
- **borderline** ($0$ or $180^\circ$)?

**Crossproduct** $v_1 \times v_2 = x_1 y_2 - y_1 x_2$

$$= |v_1| |v_2| \sin \theta.$$

Thus, $\text{sign}(v_1 \times v_2) = \text{sign}(\sin \theta)$

- $> 0$ if $\theta$ convex,
- $< 0$ if $\theta$ reflex,
- $= 0$ if $\theta$ borderline.
Primitive operations: Orientation test

Given three points $p_1, p_2, p_3$ are they
- in \textit{clockwise (cw) order},
- in \textit{counterclockwise (ccw) order}, or
- \textit{collinear}?

\[(p_2 - p_1) \times (p_3 - p_1)\]
\[
> 0 \text{ if ccw} \\
< 0 \text{ if cw} \\
= 0 \text{ if collinear}
\]
Primitive operations: Sidedness test

Given three points $p_1, p_2, p_3$ are they
- in *clockwise (cw) order*,
- in *counterclockwise (ccw) order*, or
- *collinear*?

Let $L$ be the oriented line from $p_1$ to $p_2$. Equivalently, is the point $p_3$
- *right* of $L$,
- *left* of $L$, or
- *on* $L$?
Line-segment intersection

Given \( n \) line segments, does any pair intersect?

Obvious algorithm: \( O(n^2) \).

![Diagram of line segments](image)
Sweep-line algorithm

• Sweep a vertical line from left to right (conceptually replacing $x$-coordinate with time).
• Maintain dynamic set $S$ of segments that intersect the sweep line, ordered (tentatively) by $y$-coordinate of intersection.
• Order changes when
  • new segment is encountered,  
  • existing segment finishes, or  
  • two segments cross
• Key *event points* are therefore segment endpoints.
Sweep-line algorithm

Process event points in order by sorting segment endpoints by $x$-coordinate and looping through:

- For a left endpoint of segment $s$:
  - Add segment $s$ to dynamic set $S$.
  - Check for intersection between $s$ and its neighbors in $S$.
- For a right endpoint of segment $s$:
  - Remove segment $s$ from dynamic set $S$.
  - Check for intersection between the neighbors of $s$ in $S$. 
Analysis

Use red-black tree to store dynamic set $S$.
Total running time: $O(n \lg n)$.
Correctness

**Theorem:** If there is an intersection, the algorithm finds it.

**Proof:** Let $X$ be the leftmost intersection point. Assume for simplicity that
- only two segments $s_1, s_2$ pass through $X$, and
- no two points have the same $x$-coordinate.

At some point before we reach $X$, $s_1$ and $s_2$ become consecutive in the order of $S$. Either initially consecutive when $s_1$ or $s_2$ inserted, or became consecutive when another deleted.