Geometric Algorithms

Range searching
Quadtrees, 2D trees, kD trees
Intersections of geometric objects

Geometric search: overview

Types of data: points, lines, planes, polygons, circles, ...
This lecture: sets of N objects.

Geometric problems extend to higher dimensions.
  - Good algorithms also extend to higher dimensions.

Basic problems.
  - Range searching.
  - Nearest neighbor.
  - Finding intersections of geometric objects.

1D Range Search

Extension to symbol-table ADT with comparable keys.
  - Insert key-value pair.
  - Search for key k.
  - How many records have keys between k₁ and k₂?
  - Iterate over all records with keys between k₁ and k₂.

Application: database queries.

Geometric intuition.
  - Keys are point on the line.
  - How many points in a given interval?

1D Range Search Implementations

Range search: how many records have keys between k₁ and k₂?

Ordered array. Slow insert, binary search for k₁ and k₂ to find range.
Hash table. No reasonable algorithm (key order lost in hash).
BST. In each node x, maintain number of nodes in tree rooted at x.
Search for smallest element ≥ k₁ and largest element ≤ k₂.

<table>
<thead>
<tr>
<th></th>
<th>insert</th>
<th>count</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>ordered array</td>
<td>N</td>
<td>log N</td>
<td>R + log N</td>
</tr>
<tr>
<td>hash table</td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>BST</td>
<td>log N</td>
<td>log N</td>
<td>R + log N</td>
</tr>
</tbody>
</table>

N = # records
R = # records that match

* nodes examined
* within interval
* not touched
2D Range Search

Extension to symbol-table ADT with 2D keys.
- Insert a 2D key.
- Search for a 2D key.
- Range search: find all keys that lie in a 2D range?

Applications: networking, circuit design, databases.

Geometric interpretation.
- Keys are point in the plane.
- Find all points in a given h-v rectangle?

Grid implementation. (Sedgewick 3.18)
- Divide space into M-by-M grid of squares.
- Create linked list for each square.
- Use 2D array to directly access relevant square.
- Insert: insert (x, y) into corresponding grid square.
- Range search: examine only those grid squares that could have points in the rectangle.

Choose grid square size to tune performance.
- Too small: wastes space.
- Too large: too many points per grid square.
- rule of thumb: \( \sqrt{N} \) by \( \sqrt{N} \) grid.

Time costs.
- Initialize: \( O(N) \) to initialize 2D array of lists.
- Insert: \( O(1) \).
- Range: \( O(1) \) per point in range.

Clustering

Grid implementation. Fast, simple solution for well-distributed points.

Problem. Clustering is a well-known phenomenon in geometric data.

Ex: USA map data.
- 80,000 points, 20,000 grid squares.
- Half the grid squares are empty.
- Half the points have \( \geq 10 \) others in same grid square.
- Ten percent have \( \geq 100 \) others in same grid square.

Need data structure that gracefully adapts to data.
Space Partitioning Trees

Space partitioning tree. Use a tree to represent the recursive hierarchical subdivision of d-dimensional space.

BSP tree. Recursively divide space into two regions.
Quadtree. Recursively divide plane into four quadrants.
Octree. Recursively divide 3D space into eight octants.
kD tree. Recursively divide k-dimensional space into two half-spaces.

Applications.
- Ray tracing.
- Flight simulators.
- N-body simulation.
- Collision detection.
- Astronomical databases.
- Adaptive mesh generation.
- Accelerate rendering in Doom.
- Hidden surface removal and shadow casting.

Quad Trees

Quad tree. Recursively partition plane into 4 quadrants.

Implementation: 4-way tree.

Good clustering performance is a primary reason to choose quad trees over grid methods.

Curse of Dimensionality

Range search / nearest neighbor in k dimensions?
Main application. Multi-dimensional databases.

3D space. Octrees: recursively divide 3D space into 8 octants.
100D space. Centrees: recursively divide into 100 centrants???

2D Trees

2D tree. Recursively partition plane into 2 halfplanes.

Implementation: BST, but alternate using x and y coordinates as key.
- Search gives rectangle containing point.
- Insert further subdivides the plane.
**kD Trees**

- **kD tree.** Recursively partition k-dimensional space into 2 halfspaces.
- **Implementation:** BST, but cycle through dimensions ala 2D trees.

Efficient, simple data structure for processing k-dimensional data.
- Adapts well to high dimensional data.
- Adapts well to clustered data.
- Discovered by an undergrad in an algorithms class!

---

**Geometric Intersection**

- **Problem:** find all intersecting pairs among set of N geometric objects.
- **Applications:** CAD, games, movies, virtual reality.

**Simple version:** 2D, all objects are horizontal or vertical line segments.

**Brute force:** test all O(N^2) pairs of line segments for intersection.
**Sweep line:** efficient solution extends to 3D and general objects.

---

**Orthogonal Segment Intersection: Sweep Line Algorithm**

- Use horizontal sweep line moving from left to right.
  - Sweep line: sort segments by x-coordinate and process in this order.
  - Left endpoint of h-segment: insert y coordinate into ST.
  - Right endpoint of h-segment: remove y coordinate from ST.
  - v-segment: range search for interval of y endpoints.

**Sweep line reduces** 2D orthogonal segment intersection problem to 1D range searching!

**Running time of sweep line algorithm.**
- Sort by x-coordinate. \(O(N \log N)\)
- Insert y-coordinate into ST. \(O(N \log N)\)
- Delete y-coordinate from ST. \(O(N \log N)\)
- Range search. \(O(R + N \log N)\)

Efficiency relies on judicious use of data structures.
**Line Segment Intersection: General Version**

Use horizontal sweep line moving from left to right.
- Maintain order of segments that intersect sweep line by y-coordinate.
- Intersections can only occur between adjacent segments.
- Add/delete line segment ⇒ one new pair of adjacent segments.
- Intersection ⇒ two new pairs of adjacent segments.

**Efficient implementation of sweep line algorithm.**
- Maintain PQ of important x-coordinates - endpoints and intersections.
- Maintain ST of segments intersecting sweep line, sorted by y.
- \( O(R \log N + N \log N) \).

---

**Algorithms and Moore’s Law**

VLSI database problem: Find all intersections among h-v rectangles.
Application: microprocessor design.

Early 1970s: microprocessor design became a geometric problem
- Very Large Scale Integration (VLSI).
- Computer-Aided Design (CAD).
- Design-rule checking.

Moore’s Law: processing power doubles every 18 months.
- \( 197x \): need to check \( N \) rectangles.
- \( 197(x+1.5) \): need to check \( 2N \) rectangles on a 2x-faster computer.

Quadratic algorithm: compare each rectangle against all others.
- \( 197x \): takes \( M \) days.
- \( 197(x+1.5) \): takes \( (4M)/2 = 2M \) days. (!)

Need \( O(N \log N) \) CAD algorithms to sustain Moore’s Law.
VLSI Database Problem

Move a vertical "sweep line" from left to right.
- Sweep line: sort rectangles by x-coordinate and process in this order.
- Maintain data structure of intervals intersecting sweep line.
- Key operation: given a new interval, does it intersect an existing one?

Interval Search Trees

Support following operations.
- Insert an interval \((lo, hi)\).
- Delete the interval \((lo, hi)\).
- Search for an interval that overlaps \((lo, hi)\).

Non-degeneracy assumption. No rectangles share same x-coordinate.

Interval tree implementation with BST.
- BST nodes contain interval.
- BST sorted on \(lo\) endpoint.
- Additional info: store max endpoint in subtree rooted at node.

Finding an Overlapping Interval

Search for an interval that overlaps \(I = (lo, hi)\).

\[
x = \text{root};
while (x \neq \text{null}) {
  if (x.interval.overlaps(lo, hi))
    return x.interval;
  if (x.left == \text{null})  x = x.right;
  else if (x.left.max < lo) x = x.right;
  else                      x = x.left;
}\]
return null;
Finding an Overlapping Interval

Search for an interval that overlaps \( I = (lo, hi) \).

### Case 1 (right).
If search goes right, then there exists an overlap in right subtree or no overlap in either.

**Proof.** Suppose no overlap in right.
- \((x.left == null) \Rightarrow \) no overlap in left.
- \((x.left.max < lo) \Rightarrow \) no overlap in left.

```java
x = root;
while (x != null) {
    if (x.interval.overlaps(lo, hi)) return x.interval;
    if (x.left == null) x = x.right;
    else if (x.left.max < lo) x = x.right;
    else x = x.left;
}
return null;
```

### Case 2 (left).
If search goes left, then there exists an overlap in left subtree or no overlap in either.

**Proof.** Suppose no overlap in left.
- \((x.left.max >= lo) \Rightarrow \) no interval \((a, b)\) in right subtree overlaps \((lo, hi)\).

VLSI Database Problem

Move a vertical "sweep line" from left to right.

- Sweep line: sort rectangles by \( x \)-coordinates and process in this order, stopping on left and right endpoints.
- Store set of rectangles that intersect the sweep line in an interval search tree (using \( y \)-interval of rectangle).
- Left side: interval search for \( y \)-interval of rectangle, insert \( y \)-interval.
- Right side: delete \( y \)-interval.

VLSI Database Problem: Sweep Line Algorithm

Sweep line reduces 2D orthogonal rectangle intersection problem to 1D interval searching!

Running time of sweep line algorithm.

- Sort by \( x \)-coordinate. \( O(N \log N) \)
- Insert \( y \)-interval into ST. \( O(N \log N) \)
- Delete \( y \)-interval from ST. \( O(N \log N) \)
- Interval search. \( O(R + N \log N) \)

Efficiency relies on judicious extension of BST.
Summary

Basis of many geometric algorithms: search in a planar subdivision.

<table>
<thead>
<tr>
<th>basis</th>
<th>grid</th>
<th>2D tree</th>
<th>Voronoi diagram</th>
<th>intersecting lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>representation</td>
<td>(\sqrt{N \text{ h-v lines}})</td>
<td>(N \text{ points})</td>
<td>(N \text{ points})</td>
<td>(\sqrt{N \text{ lines}})</td>
</tr>
<tr>
<td>cells</td>
<td>(2D) array of (N) lists</td>
<td>(N)-node BST</td>
<td>(N)-node multilist</td>
<td>(\sim N)-node BST</td>
</tr>
<tr>
<td>search cost</td>
<td>(\sim N \text{ squares})</td>
<td>(N \text{ rectangles})</td>
<td>(N \text{ polygons})</td>
<td>(\sim N \text{ triangles})</td>
</tr>
<tr>
<td>extend to (kD)?</td>
<td>too many cells</td>
<td>easy</td>
<td>cells too complicated</td>
<td>use ((k-1)D) hyperplane</td>
</tr>
</tbody>
</table>