Computational Geometry

[CLRS Chapter 33, except not § 33.3]

Collection of problems dealing with relationships between objects.

Important for

- Computer-Aided Design
- Computer Vision
- Computer Animation
- Molecular Modeling
- Geographic Information Systems

Segment Intersection Problem

Given: A set of \( n \) distinct line segments \( S_1, \ldots, S_n \) in the plane, represented by the coordinates of their end points.

Detecting \( \rightarrow \) Goal I: Determine whether there is an intersection

Reporting \( \rightarrow \) Goal II: Report all pairs of intersecting segments

Can this be solved in polynomial time?

In \( \Theta(n^2) \) time can examine each pair of line segments in turn and check for intersection.
Can we do better than $O(n^2)$?

For the general worst case, Goal II can not be improved below $O(n^2)$, because just reporting the output would be $O(n^2)$:

\[
\left(\frac{n}{2}\right)\left(\frac{n}{2}\right) = \frac{n^2}{4} \rightarrow O(n^2) \text{ intersections}
\]

We will focus on Goal I and show an $O(n \log n)$ algorithm

Two assumptions - not necessary
1. No perfectly vertical line segments
2. There are no instances of three line segments intersecting at a single point

Concepts: Sweepline and Preorder

Notice that there is a region where the sweepline places intersecting segments adjacent to one another in the preorder just before and just after an intersection.
The algorithm works by ordering the segment endpoints by x-coordinate and sweeping left-to-right. It maintains a preorder data structure by inserting a segment into the data structure (by its y-coordinate) at its left endpoint and deleting it from the data structure when the sweepline reaches its right endpoint. Every time a pair of segments become adjacent in the preorder, they are checked for intersection.

[Upon insertion, newly inserted segment check with segment just above and just below; upon deletion, two segments surrounding deleted segment are checked.]

If intersection found → Return TRUE

Else → Return FALSE

Demands on Data Structure

\[ \text{data structure } \mathcal{T} \]

- Insert \((T, s)\)
- Delete \((T, s)\)
- Above \((T, s)\) - report segment just above \(s\) in preorder
- Below \((T, s)\) - "" "" below \(s\) ""

Book uses Red-Black trees for data structure with \(O(\log n)\) time per operation.
Running Time:

Sorting endpoints by x-coordinate: $O(n \log n)$

Loop over segment endpoints executes $O(2n)$ times: Each iteration $O(\log n)$ for data structure operations and $O(1)$ for intersection check

$O(n \log n)$

Correctness

Claim: Algorithm returns true if and only if an intersection exists.

Proof:

1. Algorithm returns true $\Rightarrow$ an intersection exists
   - Because the algorithm only returns true immediately after finding an intersection, it is clear that if it returns true an intersection exists.

2. An intersection exists $\Rightarrow$ Algorithm returns true
   - This is more complex, but the argument is essentially that as the Sweepline moves left-to-right approaching an intersection, the preorder will be correct and the segments about to intersect will become adjacent in the preorder in time to be checked for intersection before the actual intersection occurs.
Finding Closest Pair of Points

Given: Set of $Q$ points in the plane
Find: The two points from the set whose Euclidean
distance is shortest

To check all pairs is $\frac{N(N-1)}{2} \rightarrow O(N^2)$

We seek $O(n \log n)$ algorithm (divide-and-conquer)

Recursive calls use $P \subseteq Q$ set of points and arrays $X$ and $Y$
containing points in $P$ sorted by $x$- or $y$-coordinate, respectively.

If $|P| \leq 3$, use brute force to find shortest distance.

```
Divide:
```

```
form $X_L$ & $Y_L$
```

```
divide set $P$
into 2 sets by
x-coordinate w/
vertical line
```

```
form $X_R$ & $Y_R$
```

Recurse

Recurse
Combine:

Let $\delta_L$ and $\delta_R$ be shortest distance in $P_L$ and $P_R$, respectively.

Let $\delta = \text{min}(\delta_L, \delta_R)$

Then, shortest distance in $P$ is either $\delta$ or is a distance that spans $P_L - P_R$ (one point in $P_L$ and one point in $P_R$).

Only need to check near vertical line dividing $P$ into $P_L$ and $P_R$.

1. Construct $Y'$ as $Y$ with only points in $2\delta$-strip included.

2. Iterate through $Y'$, checking the following 7 points for being with distance $\delta$ of each point, keeping track of shortest.
Correctness: Straightforward, because all cases correctly handled.

Running Time:
A number of implementation details must be properly handled.
To achieve $O(n \log n)$ need recurrence $T(n) = 2T(n/2) + O(n)$

1. Resort points by x- and y-coordinates into X and Y at very start to pay $O(n \log n)$ up front and avoid further sorting.

2. At every divide, "parse" X into $X_L$ and $X_R$ and also "parse" Y into $Y_L$ and $Y_R$ in linear time to avoid $O(n \log n)$ sorting cost.

3. Combine entails $Y \rightarrow Y'$ and checking each point in $Y'$ against $F$ following points. Each can be done in linear time as well.