10. Computational Geometry and Line Segment Intersection

CPSC 535 ~ Spring 2019

Kevin A. Wortman

CALIFORNIA STATE UNIVERSITY FULLERTON

April 22, 2019

Big Idea: Generality versus Specialization

Common sense: very general problems are harder to solve than specific problems.

Mathematics:

- Theorems are stated "If A then B."
- A the *antecedent*, B the *consequent*.
- More constraints in A means either B is easier to prove; or we can prove a stronger version of B.

Example: Shortest Paths

single source shortest paths problem
input: weighted graph $G = (V, E)$, start vertex $s \in V$
output: for each $v \in V$, a shortest path from s to v

As stated: Bellman-Ford algorithm takes $\Theta(mn)$ time.

Constrain all edge weights to be non-negative
$\implies$ Dijkstra's algorithm takes only $\Theta(m + n \log n)$ time.

The constraint that weights are nonnegative makes the problem easier to solve (negative cycles d.n.e.) so it admits a faster algorithm.

Big Idea: Output Sensitive Algorithm

- **input sensitive**: (time) efficiency is a function of the input
  e.g. size n, $\#$ edges m, maximum word W
- **output sensitive**: efficiency is also a function of the *output*
  e.g. $\#$ items returned
- most relevant when the size of the output may or may not be a bottleneck

Example: Matching Index Pairs

matching index pairs problem
input: sets $L[0..\ell], R[0..r]$
output: each pair (i, j) where $L[i] = R[j]$

Let $k \equiv$ number of pairs in output

Nested for loops: $\Theta(\ell r)$, regardless of k

Using a hash table:

- $\Theta(\ell + r + k)$
- $k \leq \ell r$; hash alg. is same speed for large k
- but is much faster for small k
- improvement when small k is likely or guaranteed

Computational Geometry

computational X: interdisciplinary study of computer science with X
(computational sociology/epidemiology/physics/finance/etc.)

computational geometry (CG): algorithms, data structures, asymptotic analysis, of geometric objects: points, lines, circles, triangle meshes, etc.

Applications

- computer graphics, user interfaces
- GIS, geographic databases
- scene reconstruction (e.g. LIDAR)
- business operations research (e.g. linear programming, aircraft control)
- manufacturing (e.g. feasibility of assembly, castings)

Putting the Geo in CG

Some general algorithms can actually solve geometric problems efficiently, without any awareness of the geometry.

bounding box problem
input: set of 2D points $P = \{p_1, p_2, \ldots, p_n\}$
output: points $tl = (x_l, y_t)$ and $rb = (x_r, y_b)$ such that the rectangle with top-left corner tl and bottom-right corner rb contains P

Naïve, optimal algorithm: x_l, y_t, x_r, y_b = min x, min y, max x, max y respectively; $\Theta(n)$

Computational geometers are most interested when geometric properties matter.

Line Segment Predicates

We can use arithmetic to answer any of the following predicates (questions) about points p_0, p_1, p_2, p_3 in $\Theta(1)$ time:

1. Is line segment $\overline{p_0 p_1}$ clockwise from $\overline{p_0 p_2}$ around the common endpoint p_0?
2. If we follow $\overline{p_0 p_1}$ and then $\overline{p_1 p_2}$, do we turn right or left?
3. Do line segments $\overline{p_0 p_1}$ and $\overline{p_2 p_3}$ intersect?

$\Longrightarrow$ We may use any of these in pseudocode.

Degeneracy and Non-Degeneracy Assumptions

degenerate object: has the proper shape/type, but the values are a special case that betrays the spirit of the definition

Example: triangle ≡ three points (p_1, p_2, p_3)
degenerate triangle: $p_1 = p_2 = p_3$; or p_1, p_2, p_3 colinear; etc.

Non-degeneracy assumption:

- constraint that input to a CG algorithm is not degenerate in specific ways
- simplifies algorithm design
- assume that in practice, some combination of
  - degeneracies do not occur
  - input can be preprocessed to remove degeneracies
  - implementer can modify algorithm to handle degeneracies

Line Segment Intersection

line segment intersection problem
input: set of n line segments
$L = \{((x_1, y_1), (x_2, y_2)) : x_1, y_1, x_2, y_2 \in \mathbb{R}\}$
output: some pair $\ell, \ell' \in L$ that intersect, or NIL if no segments in L intersect

Non-degeneracy assumptions:

- no segments are vertical
- no three segments intersect in a common point

Thought exercise: How realistic is this? How hard would it be to sanitize input without affecting the output?

Baseline algorithm: nested for loops, $\Theta(n^2)$ time.

Sweep Algorithms

A pattern in CG algorithms:

- *line sweep:* envision a line "sweeping" through the input
- e.g. a vertical line sweeping left-to-right
- helps us visualize a 2D situation as a 1D situation that changes over time
- like duality, doesn't actually change the problem, but might help us problem-solve
- generalizes to higher dimensions e.g. plane sweep in 3D, hyperplane sweep in any dimension

Geometric Insight

- Visualize vertical line sweeping left-to-right.
- Consider some segment ℓ; at some point ℓ's left endpoint will strike the sweep line; then the common point will slide a bit as the sweep continues; then the sweep will move past the ℓ's right endpoint.
- These time steps are discrete **events** that matter; fast-forward past in-between moments.
- Consider the ordering of active line segments along the sweep line in top-to-bottom order.
- *If two segments swap order between time events, then they must intersect.*

Line Segment Intersection Pseudocode

```
function LINE-SEGMENT-INTERSECTION(L)
    T = new BST of points ordered by y-coordinate
    S = sort all endpoints in L by x-coordinate
    for endpoint p in S do
        if p is a left endpoint then
            T.insert(p)
            if p intersects with p's predecessor or successor then
                return p and that intersecting neighbor
            end if
        else                                        ▷ p must be a right endpoint
            if p's pred. and succ. both exist and intersect then
                return the predecessor and successor
            end if
            T.delete(p)
        end if
    end for
    return NIL
end function
```

Non-Degeneracy Assumptions Revisited

Algorithm assumes that

$$\text{sweep line} \cap \text{each line segment}$$

is only one point
$\implies$ require that no segment is vertical.

Algorithm assumes that intersecting segments only move *one* step in top-to-bottom order
$\implies$ require that 3+ segments may never intersect at the same point.

BST Operations Review

create empty: $\Theta(1)$
search, insert, delete: $\Theta(\log n)$

Predecessor/successor query:

- esoteric BST operation, yet still available in e.g. C++ STL
- given pointer to a node, find its inorder predecessor/successor
- can visualize as moving to the previous/next step in an Euler tour (sketch)
- $\Theta(\log n)$

Line Intersection Analysis

sort points: $\Theta(n \log n)$

for loop: $2n \in \Theta(n)$ events
body of loop involves $\Theta(1)$ BST operations
$\implies \Theta(\log n)$ time per iteration

$\Theta(n \log n + n \log n) = \Theta(n \log n)$ total time

Example of reduction to both sorting and BST operations.