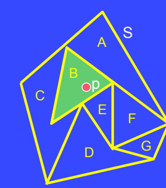
Computational Geometry

Chapter 6

Point Location

Problem Definition

- Preprocess a planar map *S*. Given a query point *p*, report the face of *S* containing *p*.

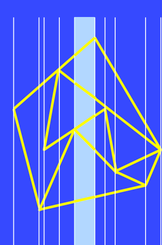


- **Goal:** O(n)-size data structure that enables O(log n) query time.
- **Application:** Which state is Boston located in?
- **Trivial Solution:** O(n) query time, where n is the complexity of the map. Why?

Naïve Solution

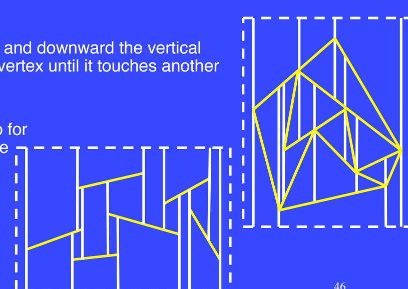
- Draw vertical lines through all the vertices of the subdivision.
- Store the x-coordinates of the vertices in an ordered binary tree.
- Within each slab, sort the segments separately along y.
- Query time: O(log n).
- **Problem**: Too delicate subdivision, of size $\Theta(n^2)$ in the worst case.

(Give such an example!)

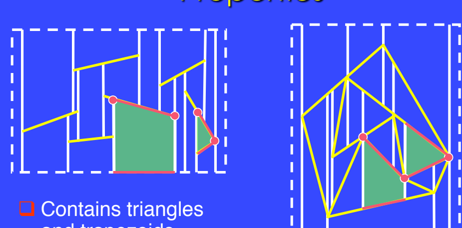


The Trapezoidal Map

- Construct a bounding box.
- Assume general position: unique x coordinates.
- Extend upward and downward the vertical line from each vertex until it touches another segment.
- This works also for noncrossing line segments.



Properties



- Contains triangles and trapezoids.
- Each trapezoid or triangle is determined:
  - By two vertices that define vertical sides; and
  - By two segments that define nonvertical sides.
- A *refinement* of the original map.

Notation

Every trapezoid (triangle) Δ is defined by

- Left(Δ): The left segment of Δ
  - (actually it is enough (will see later why) only an endpoint intersecting by the segment.
  - It is either right endpoint or a left endpoint;
- Right(Δ): a segment endpoint (right or left);
- Top(Δ): a segment;
- Bottom(Δ): a segment.

Complexity

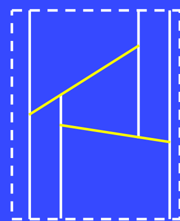
- **Theorem** (linear complexity): A trapezoidal map of n segments contains at most $6n$+4 vertices and at most $3n$+1 faces.
- **Proof:**

1. Vertices:

$$\underbrace{2n}_{\text{original}} + \underbrace{4n}_{\text{extensions}} + \underbrace{4}_{\text{box}} = 6n + 4$$

2. Faces: Count Left (Δ).

$$\underbrace{2n}_{\text{left e.p.}} + \underbrace{n}_{\text{right e.p.}} + \underbrace{1}_{\text{box}} = 3n + 1$$

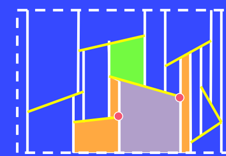


$n = 2$
$V = 16$
$F = 7$

Map Data Structure

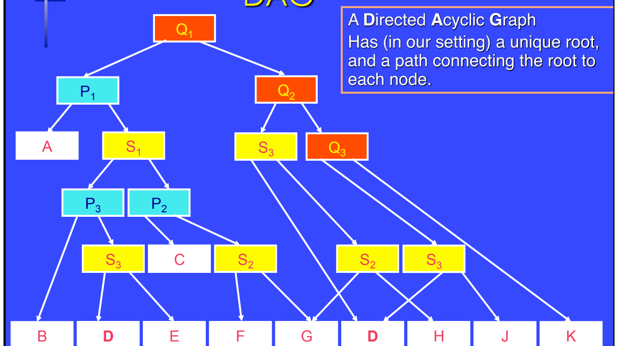
- Possibly by DCEL.

An alternative:
For each trapezoid store:
- The vertices that define its right and left sides;
- The top and bottom segments;
- The (up to *two*) neighboring trapezoids on right and left;
- (Optional) The neighboring trapezoids from above and below. This number might be linear in n, so only the leftmost of these trapezoids is stored.

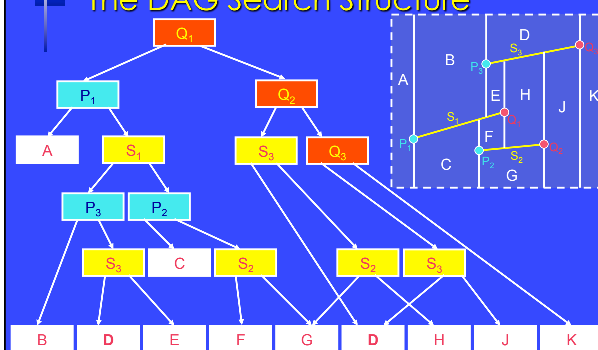


DAG

A **D**irected **A**cyclic **G**raph
Has (in our setting) a unique root, and a path connecting the root to each node.



The DAG Search Structure



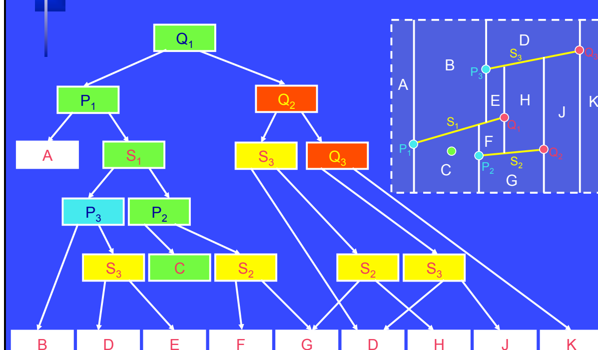
White – trapezoids. Red/Blue – vertical walls. Yellow – original segments

DAG Branching Rules

Given a query point q how can we find the trapezoid containing q ?

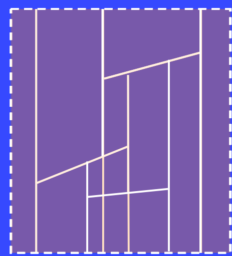
Assume a search-structure node s is given
(initially the root of the DAG)

Search(q, s):
- /* Query point q, search-structure node s. */
- If s is a segment-**endpoint** then
  - q is to the **right** of s: go **right**;
  - q is to the **left** of s: go **left**;
  - /*No use of the y-coordinates of s */
  - Else:
- If s is a segment:
  - q is **below** s: go **right**;
  - q is **above** s: go **left**;



Construction Algorithm (high level description)

Input: A set of segment
Output- the DAG

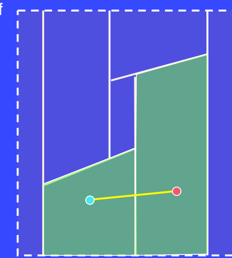


- Find a Bounding Box.
- Randomly permute the segments.
- Insert the segments one by one into the map.
- Update the map and search structure in each insertion.
- The map is independent of the order of insertion and its size is $\Theta(n)$.
- The DAG and its size depends on the order of insertion.

136.

Updating the Structures

- Find **in the existing structure** the face that contains the left endpoint of the new segment.
- Find all other trapezoids intersected by this segment by moving to the right.
- Update the map M_i and the DAG D_i.

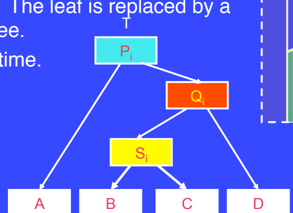


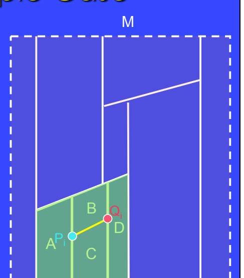
146.

Update *D*: Simple Case

The segment is contained entirely in one trapezoid.

- In M_i: Split the trapezoid into four trapezoids.
- In D_i: The leaf is replaced by a subtree.
- O(1) time.



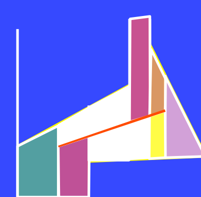


156.

Update *M*: General Case

The i^{th} segment intersects k_i>1 trapezoids.

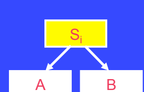
- Split trapezoids.
- Merge trapezoids that can be united.
- O(k_i) time.

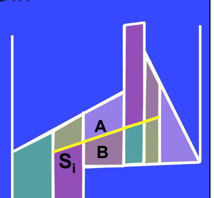


166.

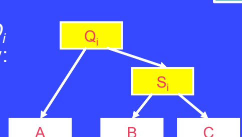
Updating *D*: Split

- Each *inner* trapezoid in D_i is replaced by:





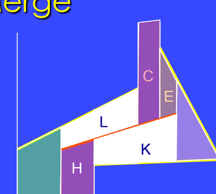
- Each *outer* trapezoid in D_i is replaced by:

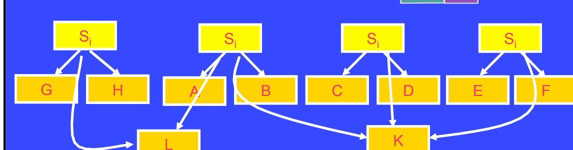


176.

Updating *D*: Merge

- Leaves are eliminated and replaced by one common leaf.
- O(k_i) time.



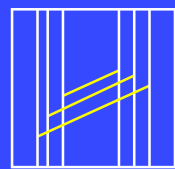


186.

Size of D: Worst-Case Analysis

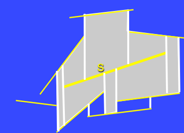
- Each segment adds trees of depth at most 3, so the depth of D_i is $\leq 3i$.
- Query time (depth of D_i): O(i), $\Theta(i)$ in the worst case.
- The i^{th} segment – s_i - may intersect with k_i = **O(i)** trapezoids !
- The size of D and its construction time is in the worst case:

$$\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$$

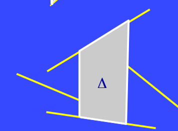


Segment/Trapezoid Interaction

- One segment may affect many trapezoids



- One trapezoid may affect at most **four** segments



Average-Case Analysis

Compute the expected depth of D:
- q: A point, to be searched for in D.
- p_i: The probability that a new node was created in the path leading to q in the i^{th} iteration.

Compute p_i by backward analysis:
- $\Delta_q(M_{i-1})$: The trapezoid containing q in M_{i-1}.
- Since a new node was created, $\Delta_q(M_i) \neq \Delta_q(M_{i-1})$.
- Delete s_i from M_i.

  Prob$[\Delta_q(M_i) \neq \Delta_q(M_{i-1})] \leq 4/i$.

Average-Case Analysis

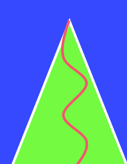
Compute the expected depth of D:
- q: A point, to be searched for in D.
- p_i: The probability that a new node was created in the path leading to q in the i^{th} iteration.

Compute p_i by backward analysis:
- $\Delta_q(M_{i-1})$: The trapezoid containing q in M_{i-1}.
- Since a new node was created, $\Delta_q(M_i) \neq \Delta_q(M_{i-1})$.
- Delete s_i from M_i.

  Prob$[\Delta_q(M_i) \neq \Delta_q(M_{i-1})] \leq 4/i$.

Expected Depth of D

- x_i: The number of nodes **created** in the i^{th} iteration in the path leading to the leaf q.



- The expected length of the path leading to q:

$$E\left[\sum_{i=1}^{n} x_i\right] = \sum_{i=1}^{n} E[x_i] \leq \sum_{i=1}^{n} (3p_i) \leq \sum_{i=1}^{n} \tfrac{12}{i} = O(\log n).$$

Theorems

- The expected size is $O(n)$
- The expected query time is $O(\log n)$
- (The proof from this point are not required, but we will handwave a bit)

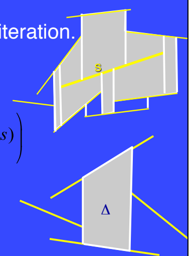
Expected Size of *D*

- Define an indicator

$$\delta(\Delta, s) = \begin{cases} 1 & \Delta \text{ disappears from } M_i \text{ if } s \text{ is removed} \\ 0 & \text{otherwise} \end{cases}$$

- k_i: Number of leaves created in the i^{th} iteration.
- S_i: The set of the first i segments.
- Average on s:

$$\begin{aligned} \mathrm{E}[k_i] &= \tfrac{1}{i} \sum_{s \in S_i} \left(\sum_{\Delta \in M_i} \delta(\Delta, s) \right) = \tfrac{1}{i} \sum_{\Delta \in M_i} \left(\sum_{s \in S_i} \delta(\Delta, s) \right) \\ &\le \tfrac{1}{i} \sum_{\Delta \in M_i} 4 \\ &= \tfrac{1}{i} (4 \, | M_i |) \\ &= \tfrac{\mathrm{O}(i)}{i} = \mathrm{O}(1). \end{aligned}$$



256.

Expected Size of *D* (cont.)

- k_i-1: Number of internal nodes created in the i^{th} step.
- Total size:

$$\mathrm{O}(n) + \mathrm{E}\left(\sum_{i=1}^{n} (k_i - 1) \right) = \mathrm{O}(n) + \mathrm{E}\left(\sum_{i=1}^{n} k_i \right) = \mathrm{O}(n).$$

↑ leaves ↑ internal

266.

Expected Construction Time of *D*

$$\sum_{i=1}^{n} \big(\mathrm{O}(\log i) + \mathrm{O}(\mathrm{E}[k_i]) \big) = O(n \log n)$$

Finding the first trapezoid

The rest of the work in the i^{th} step

276.