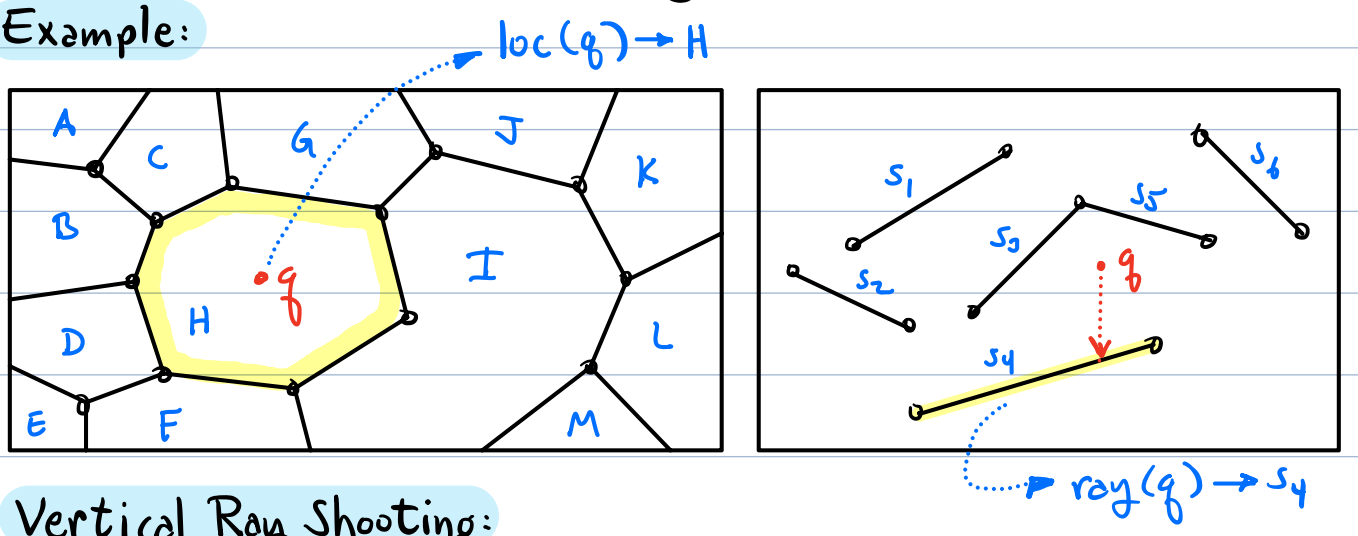
CMSC 754 - Computational Geometry
Lecture 9: Planar Point Location (+ Trapezoidal Maps)

Planar Point Location -

Given a planar subdivision (e.g., a cell complex), build a data structure, s.t. for any query point q, can find the cell containing q.

Example:



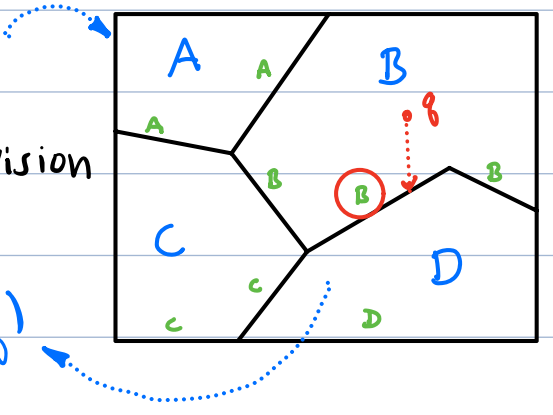
Vertical Ray Shooting:

Given a set of disjoint line segments in $\mathbb{R}^2$, build a data structure s.t. for any query point q, can report the segment directly below q (or bound. box)

Point location reducible to vert. ray shooting

- Label each segment of subdivision with region directly above

label(ray(q)) = B = loc(q)



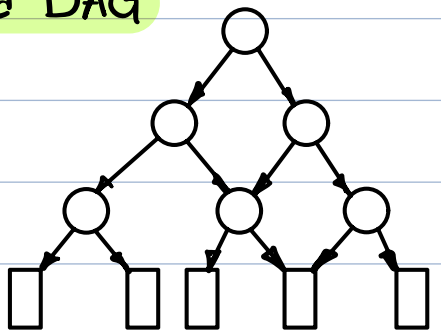
History DAG - Data structure for **vert. ray shooting**

Approach: Build a **trapezoidal map** using the **randomized inc. construction** (RIC) + record the **history** of the construction.

- Let $S = \{s_1, \ldots, s_n\}$ be segments (**randomly permuted**)
- For $0 \le i \le n$, let $S_i = \{s_1, \ldots, s_i\}$ - prefix of segments
  let $\tau_i = \tau(S_i)$ - partial trap map
- Recall (from prior lecture) the **insertion** of each segment results in **$O(1)$ new cells** in expectation

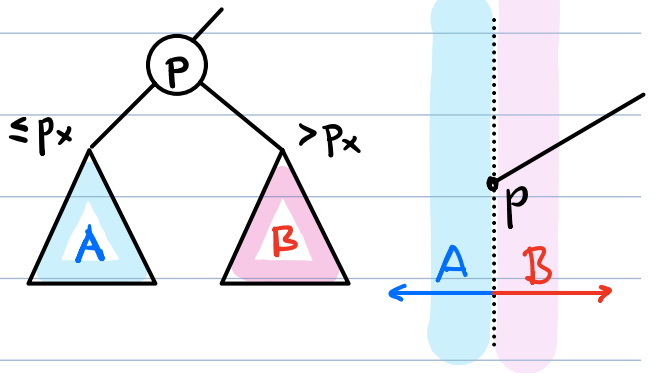
Overview:

- Final data structure is a **rooted binary tree** with **shared subtrees** → **rooted DAG**
- **Leaves** are in **1-1 correspondence** with **trapezoids** of $\tau(S)$



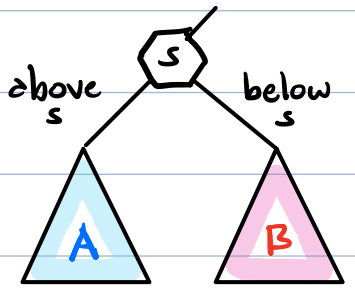
Internal Nodes - Two types

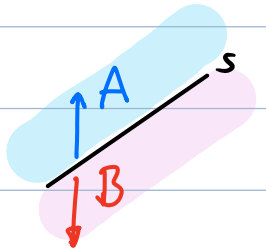
x-node: Labeled with **segment endpoint p.** **Left/Right** subtrees based on whether $q_x \le p_x$ or $q_x > p_x$



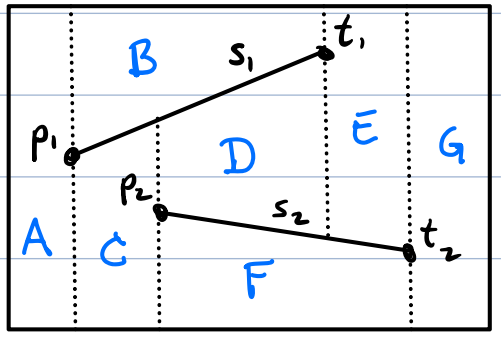
y-node: Labeled with segment s. Left/Right subtrees based on whether q is above/below s

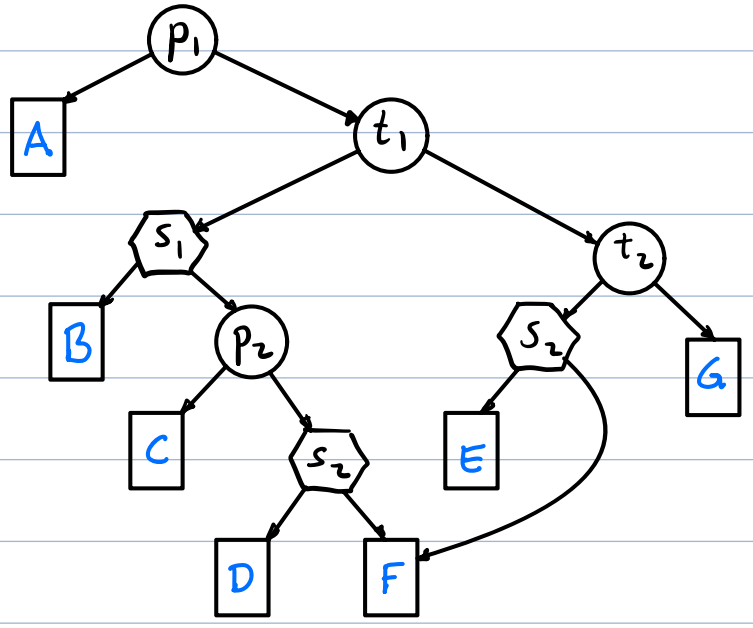
Note: q's x-extents must lie within s's x-extents.



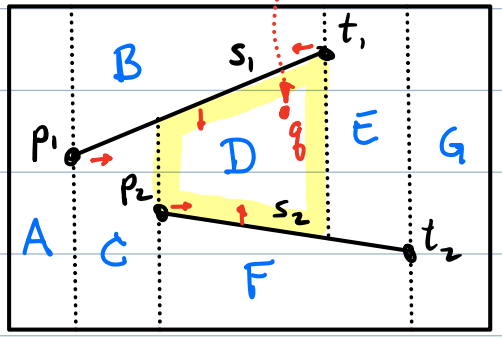


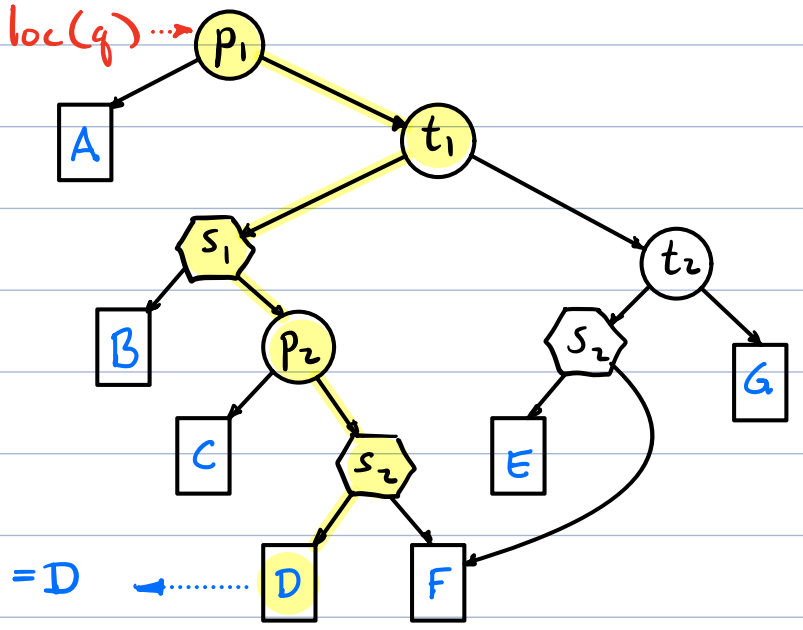
Example:





Query Processing:





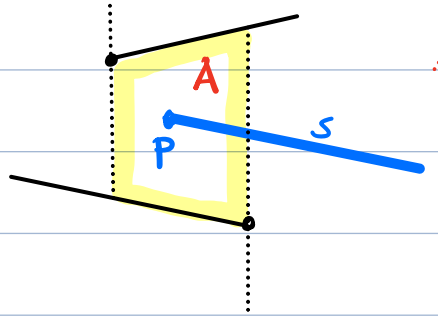
answer = D

Incremental Construction:

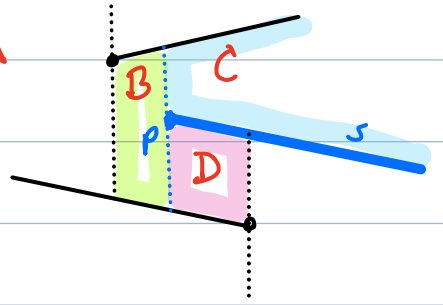
- As segments are added to the map
  - some cells are deleted
    → we delete the associated leaves
  - some cells are created
    → we create the associated leaves
  - we add internal nodes as needed to connect old with new
  - multiple paths may lead to same cell
    → share subtrees (leaves $\overset{1\text{-}1}{\longleftrightarrow}$ cells)
- Updates follow the construction process:

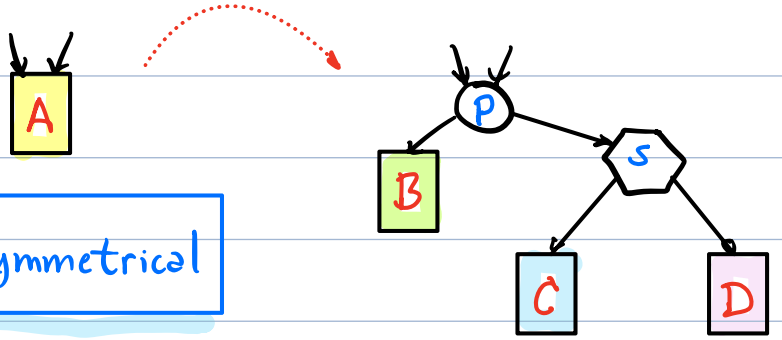
① Insert a single segment endpt. (left or right) in a cell

Before:



After:

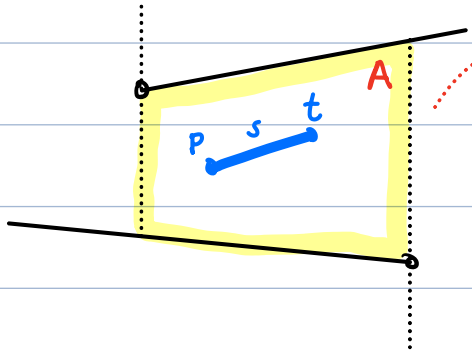




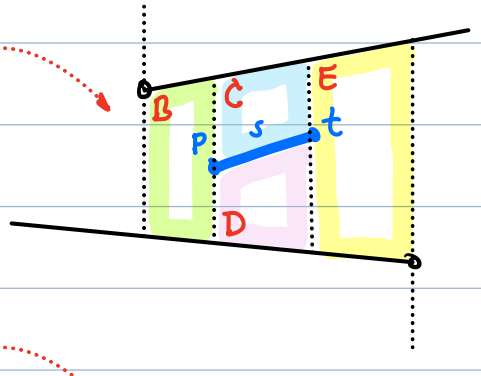
Right endpt is symmetrical

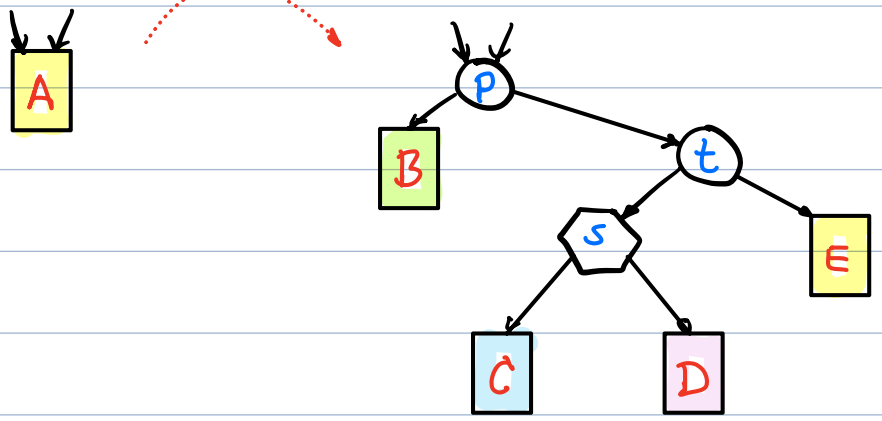
② Insert an entire segment into a single cell

Before:



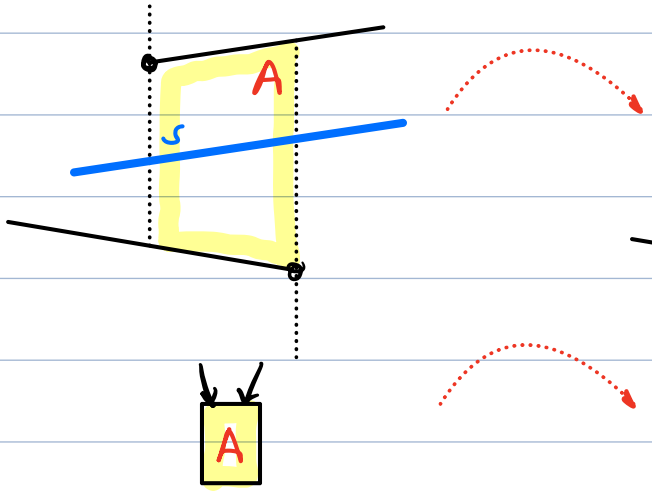
After:



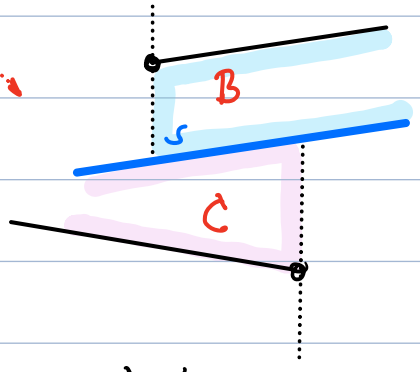


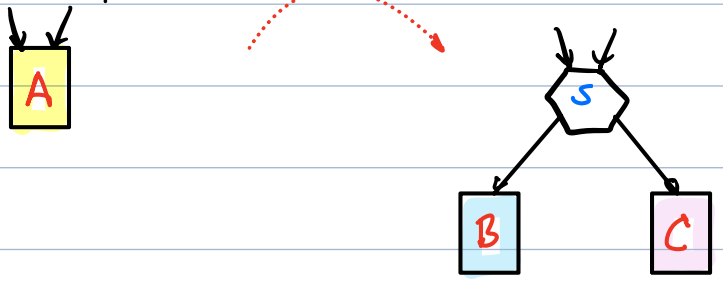
③ Segment cuts completely through a cell

Before:

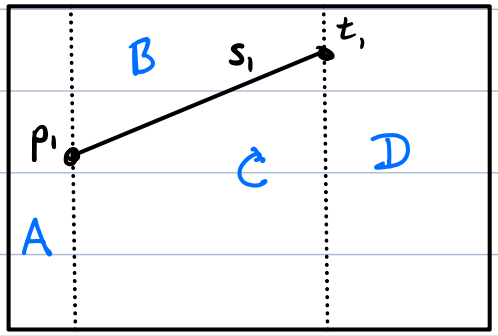


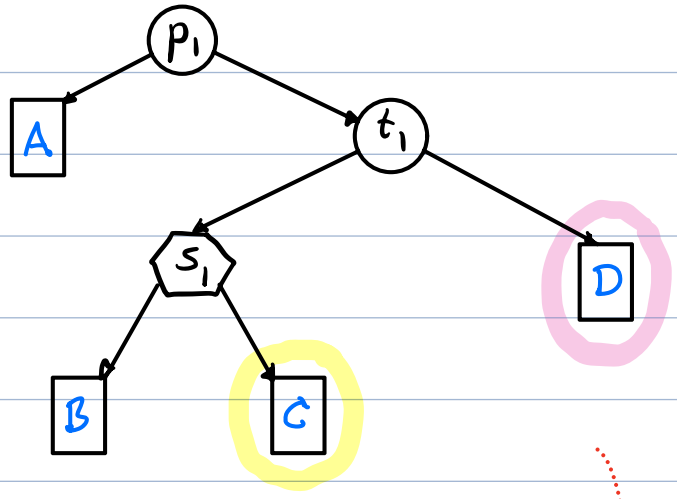
After:



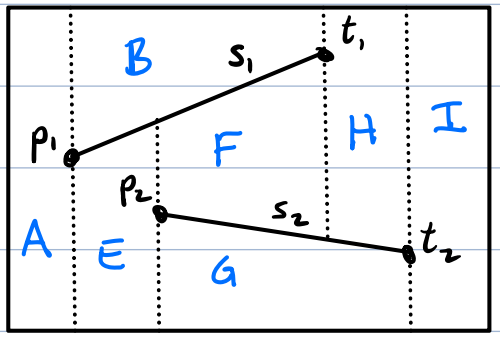


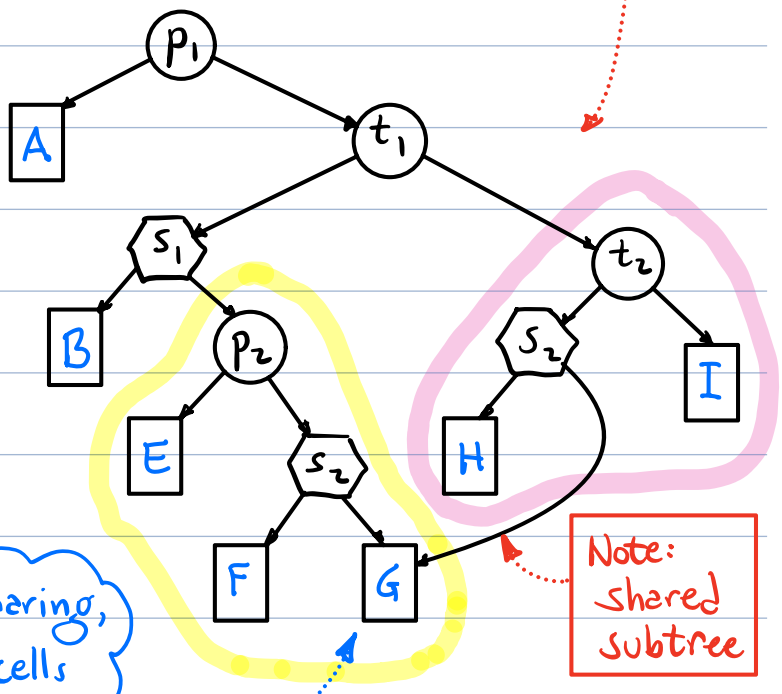
Example:





insert $s_2 = \overline{p_2 t_2}$





Due to sharing, leaves $\overset{1\text{-}1}{\longleftrightarrow}$ cells

Note: shared subtree

Analysis:

We'll show that if segments are inserted in random order:

expected storage = $O(n)$

expected query time = $O(\log n)$

Theorem: Expected number of nodes is $O(n)$

Proof: For each insertion

- number of new nodes ~ num. of new leaves
- number of new leaves = num. of new cells
- (Previous lecture) Expected num. new cells = $O(1)$

□

Theorem: The search depth for any fixed point is $O(\log n)$ in expectation over all possible insertion orders.

Note: This is not quite ideal. Better would be to replace "any fixed point" with "every point". (See latex notes for more info.)

Proof: Let q be any fixed query point.

For $0 \le i \le n$, let $\Delta_i(q)$ be the cell containing q after inserting $s_1, \ldots, s_i$.

Cases:

① $\Delta_i(q) = \Delta_{i-1}(q)$ 

$\Rightarrow$ s_i has no impact on q's search depth

② $\Delta_i(q) \neq \Delta_{i-1}(q)$ 

$\Rightarrow$ After inserting s_i, q's depth increases

Intuition: Over entire construction, ① happens most times + ② only happens $O(\log n)$ times.

For $1 \le i \le n$, let

$$X_i(q) = \begin{cases} 1 & \text{if } \Delta_i(q) \neq \Delta_{i-1}(q) \\ 0 & \text{o.w.} \end{cases}$$

[bad case]

[good case]

Obs: In the good case, depth doesn't change

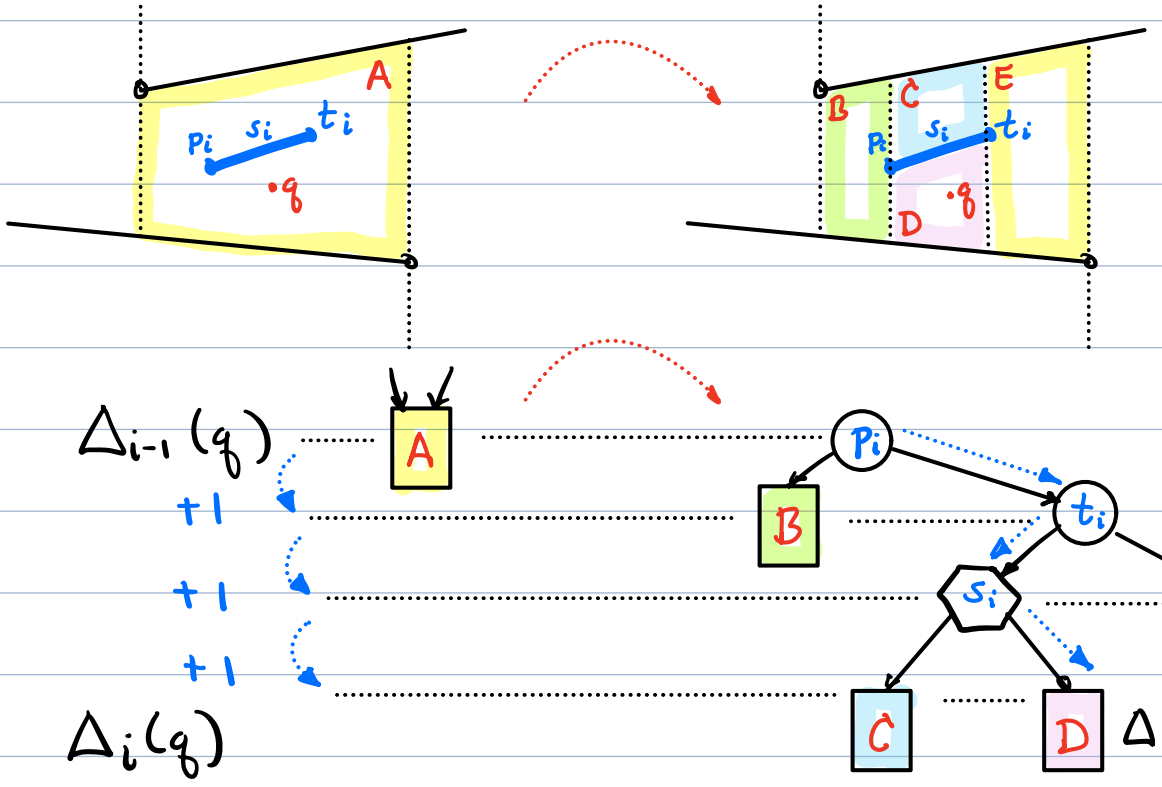
In the bad case, depth increases by constant

Claim:

$$X_i(q) = 0 \Rightarrow \text{depth}(\Delta_i(q)) = \text{depth}(\Delta_{i-1}(q))$$

$$X_i(q) = 1 \Rightarrow \text{depth}(\Delta_i(q)) \le 3 + \text{depth}(\Delta_{i-1}(q))$$

Proof: Worst case is ①



□

$\Rightarrow$ Final expected search depth is:

$$\mathbb{E}(\text{depth}(q)) \leq 3 \cdot \mathbb{E}(\text{no. of times bad case})$$
$$= 3 \sum_{i=1}^{n} \text{Prob}(X_i(q) = 1)$$

Suffices to bound $\text{Prob}(X_i(q) = 1)$

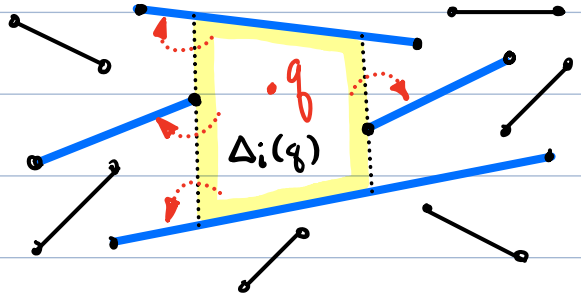
Backwards analysis -

- Since segs are inserted randomly every segment is equally likely to be last

$\Rightarrow X_i(q) = 1$

$\Leftrightarrow \Delta_i(q) \neq \Delta_{i-1}(q)$

$\Leftrightarrow$ Segment s_i is incident to $\Delta_i(q)$



$\Delta_{i-1}(q) \neq \Delta_i(q)$ $\Leftrightarrow$ one of these segs is s_i

$\Rightarrow \text{Prob}(X_i(q) = 1) \leq 4/i$

Harmonic series

$$\mathbb{E}(\text{depth}(q)) \leq 3 \sum_{i=1}^{n} \text{Prob}(X_i(q) = 1)$$
$$\leq 3 \sum_{i=1}^{n} 4/i = 12 \sum_{i=1}^{n} 1/i = 12 \cdot \ln n + O(1)$$

□

Trapezoidal map revisited:

- Last lecture we showed that RIC algorithm for trapezoidal maps runs in time:
  - ① (Expected time to locate left endpoints)
  - ② + (Expected time to update structure)
- Previous lecture — ② = $O(n)$
  This lecture — ① $\sum_{i=1}^{n} \log i = O(n \log n)$

⇒ Total expected time = $O(n \log n)$

Note: Space + query times for point-location data structure hold in expectation, not worst case.

- But we can simply repeat the construction until we obtain the desired space + query time.

⇒ Slower construction but guaranteed performance.

Also:

Works even if segments intersect:

Exp. time: $O(n \log n + m)$ → Beats plane sweep!

Space: $O(n + m)$

Query time: $O(\log n)$

m = no. of intersections