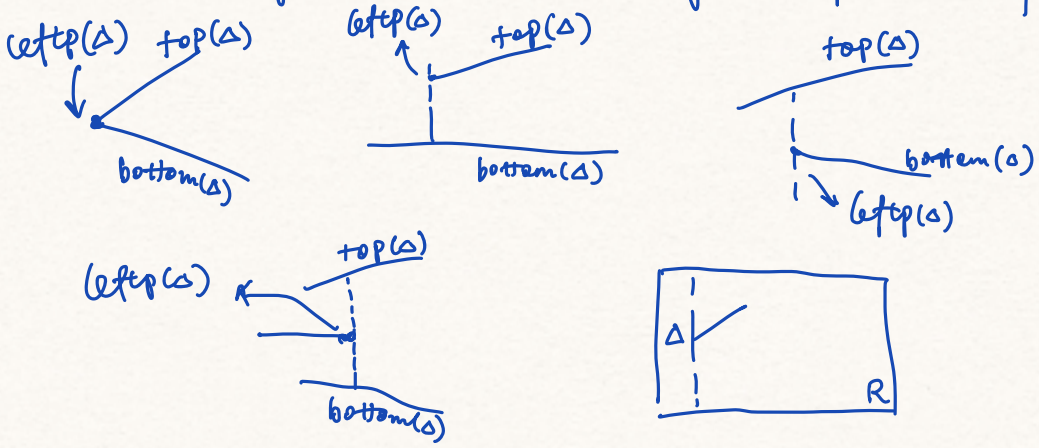
# Randomized Incremental Algorithm for trapezoidal decomposition

## - Some preliminaries:

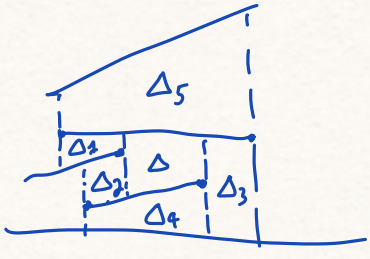
- Trapezoidal Maps is a data structure that support point location query. ( planar point location problem)
  Using this structure, we can do point location query in polylog

- Five cases for the left side of a trapezoidal face $\Delta$




- The trapezoidal map $T(S)$ of a segment set $S$ contains at most $6n+4$ vertices, and at most $3n+1$ trapezoids

- We call two trapezoids are adjacent if they meet along a vertical edge

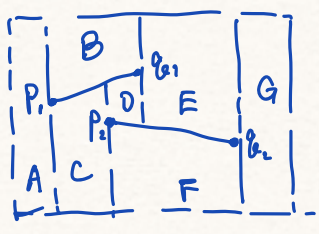


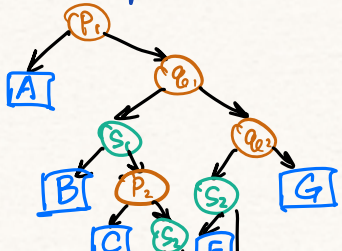

$\Delta_1$ is the upper left neighbor of $\Delta$ (share the upper segment)

$\Delta_2$ is the lower left neighbor of $\Delta$ (share the bottom segment)

## - The randomized Incremental Algorithm

- Foundation: from a trapezoidal map to its corresponding search structure




- Then the problem is: how can we build (update) the search structure when we incrementally add new segments to the existing trapezoidal map?

- The pseudocode of the algorithm:

Algorithm TrapezoidalMaps(S)

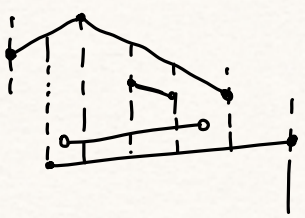
Input: A set $S$ of $n$ non-crossing line segments
Output: Trapezoidal map $\tau(S)$ and a search structure $D$ (DAG)

1. Determine a bounding box $R$ that contains all the segments of $S$. Initialize $\tau$ and $D$
2. Compute a random permutation of $S \rightarrow \langle S_1, S_2, \ldots, S_n \rangle$
3. for $i \leftarrow 1$ to $n$
4. do Find set $\Delta_0, \Delta_1, \ldots, \Delta_k$ of trapezoids in $\tau$ properly intersected by $S_i$
5. Remove $\Delta_1, \Delta_2, \ldots, \Delta_k$ from $D$, and creat leaves for the new trapezoids. Link the leaves to existing inner nodes by adding new inner nodes.

- Not trivial for how to link/add the nodes. The pseudocode:

Algorithm FollowSegment($\tau$, $D$, $S_i$)

Input: $\tau$, $D$, newly added segment $S_i$
Output: $\Delta_0, \Delta_1, \ldots, \Delta_k$ intersected by $S_i$



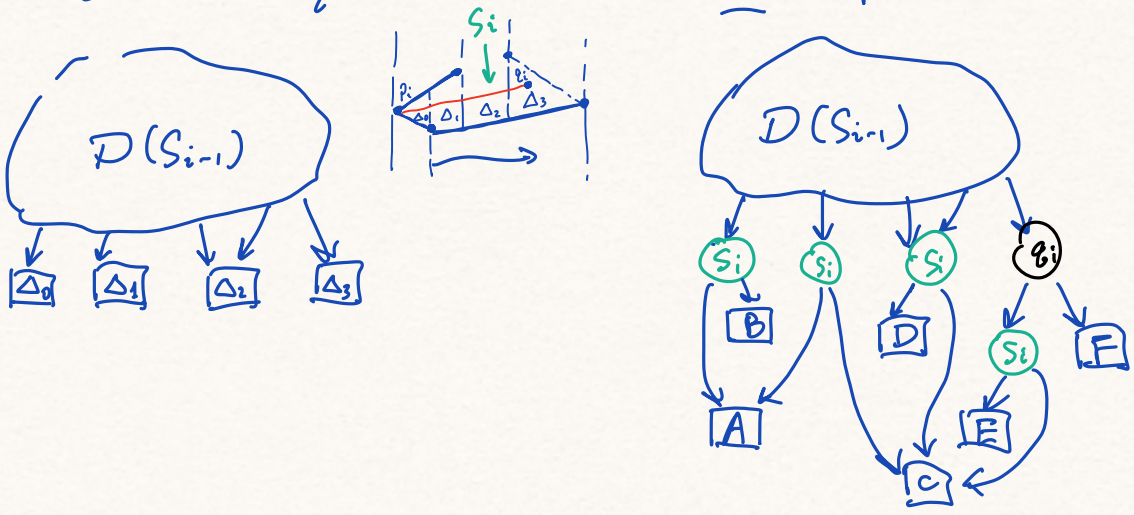
1. $p, q \leftarrow$ left & right end point of $S_i$
2. Search in the $D$ to locate the $\Delta_0$ for point $p$
3. $j \leftarrow 0$
4. while $q$ lies to the right of rightp($\Delta_j$)
5. do if rightp($\Delta_j$) lies above $S_i$
6. then Let $\Delta_{j+1}$ be the lower right neighbor of $\Delta_j$
7. else Let $\Delta_{j+1}$ be the upper right neighbor of $\Delta_j$
8. $j \leftarrow j+1$
9. Return $\Delta_0, \ldots, \Delta_k$

So we can see clearly from the pseudocode. firstly we can find the location of the left end point $p_i$ by searching from the search structure. Then we got $\Delta_0$. From $\Delta_0$, always find its right neighbor until $q_i$ is not on the right of rightp($\Delta_j$)

- The next step should be how to edit & update the new one

Notice: The efficiency of the algorithm (building part) comes from the step, which costs $O(1)$

Let's consider $s_i$ intersects with two / more vertical extensions which means it destroys $\Delta_0, \Delta_1, \cdots, \Delta_k$, $k$ trapezoids in total:




**Wrap-up**

The steps for updating the search structure $D$:

- Remove the leaves for $\Delta_0, \Delta_1, \cdots, \Delta_k$
- Creat leaves for the new trapezoids
- Introduce extra inner nodes

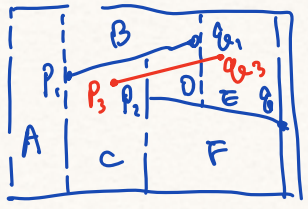
More precisely:

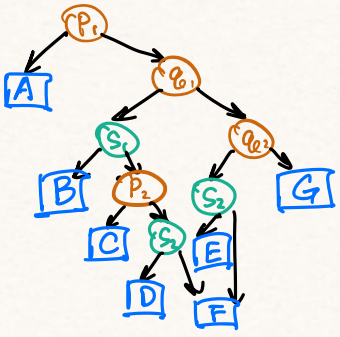
- For the left end point $p_i$, if it locates inside of the trapezoid $\Delta_0$, we can create an x-node denoting the partitions for left-side and right-side, followed with another y-node on its right-child leaf position, denoting the upper and lower partitions for the two trapezoids divided by the segment $s_i$
- The same for the right-end point $q_i$
- For the rest $\Delta_1, \Delta_2 \cdots \Delta_k$, we can simply insert y-nodes for them
- Having the inner nodes described above, the remaining job

is to point the created inner nodes to the newly created leaves. Note that some of the new trapezoids will be merged (Those are intersected by the $S_i$ throughly)

[ Moreee precisely, we need to keep track of those "inner" trapezoids that are removed, they'll be merged ]

- A detail for finding the neighbors: When we need to figure out whether the neighbor should be the upper-right or lower-right one, we can actually get the information from the search structure itself.




For example, for the new $P_3$ $q_{e_3}$, we locates at $\Delta_c$ firstly. then we can easily get the rightp($\Delta_c$) in constant time. Then we need to compare the rightp($\Delta_c$) with $S_3$ to see the location relation.

---

## Analysis (Backwards Analysis)

- So finally we finish the building/updating period, including locating, removing and adding new ones.
- The next problem will be the analysis of this set of algorithm. What about the complexity for worst case, and what about the randomized order?

Theorem:

The construction of $T(S)$ costs $O(n \log n)$ in expectation
The size of $D(S)$ is $O(n)$ in expectation
The query time for single point location costs $O(\log n)$

- Worst case occurs when every insertion causes a 3-depth increase  ← This one will increase the depth by 3

Then we got the upper bound $3n$.

- In expectation, there's $n!$ possible insertion orders in total
  Suppose $X_i$ denotes the number of nodes created in iteration $i$.
  Note that $X_i$ is a random variable, then we have

$$\mathbb{E}\left[\sum_i X_i\right] = \sum_i \mathbb{E}[X_i]$$

and since $\mathbb{E}[X_i] \le 3\mathbb{P}_i$ → $\mathbb{P}_i$ denotes that there exists a node that is created in the iteration round $i$.

$$\mathbb{P}_i = \mathbb{P}[\Delta_q(S_i) \neq \Delta_q(S_{i-1})]$$

- The bound $\mathbb{P}_i$ can be seen as: iteration $i$ contributes a node to the search path of $q$ exactly if $\Delta_q(S_{i-1})$, the trapezoid containing $q$ in $\mathcal{T}(S_{i-1})$, is not the same as $\Delta_q(S_i)$, the trapezoid containing $q$ in $\mathcal{T}(S_i)$

If $\Delta_q(S_i)$ is not the same as $\Delta_q(S_{i-1})$, then $\Delta_q(S_i)$ must be one of trapezoids created in the interation round $i$. Note that all the trapezoids $\Delta$ created in iteration $i$ are adjacent to $S_i$, the segment that is inserted in that iteration step: either $top(\Delta)$, $bottom(\Delta)$ is $S_i$, or $leftp(\Delta)$, $rightp(\Delta)$ is an end point of $S_i$.

Backwards Analysis here. For a constructed $\mathcal{T}(S)$ and $\mathcal{D}(S)$, every segment $S_i$ from the set $S$ has the same probability to be the last one inserted. Then we need to analysis the trapezoids that are created in the last insertion step.

$$\text{let} \quad K_i = \sum_{j=1}^{i} [\#\text{ trapezoids created if } S_j \text{ is the last inserted}]$$

- Then we change a perspective to count the trapezoids:

  For each trapezoid, there're are at most 4 segments' insertion have created it (mentioned above)

$$\text{then} \quad K_i = \sum_{\Delta \in \mathcal{T}_i} [\#\text{ of segment would create } \Delta] \le 4$$

$$\le \sum_{\Delta} 4 = 4 \cdot (\#\text{ trapezoids}) \le 3i+1$$

$$= 12i + 4$$

then $\frac{K_i}{i} = 12 + \frac{4}{i} \leq 13$

then sum up all the iteration steps:

$$\sum_{i=1}^{n} \frac{K_i}{i} = 13n$$

- In this case we know that #trapezoids is linear.

- Then move on to the query time for any point $q_0$

Backwards analysis again:

Take the situation after $S_i$ is inserted, ask the question:

How many of the $i$ line segments made the search path (to $q_0$) longer?

As we described above, the search path to $q_0$ becomes longer if $q_0$ is in a trapezoid that was just created by a insertion. The maximum number of segments that define the trapezoid is 4, so the probability is $4/i$

Target: Analyze the bound ↓

$$\sum_{i=1}^{n} \mathbb{P}[\text{search path became longer due to } i\text{-th insertion}]$$

$$\leq \sum_{i=1}^{n} \frac{4}{i} = 4 \sum \frac{1}{i} = \boxed{O(\log n)}$$

- Having the analysis for query time & #trapezoids in expectation we finally get the construction time in expectation, which is $\boxed{O(n \log n)}$