CS633 Lecture 08
Point Location

Jyh-Ming Lien

Dept of Computer Science

George Mason University

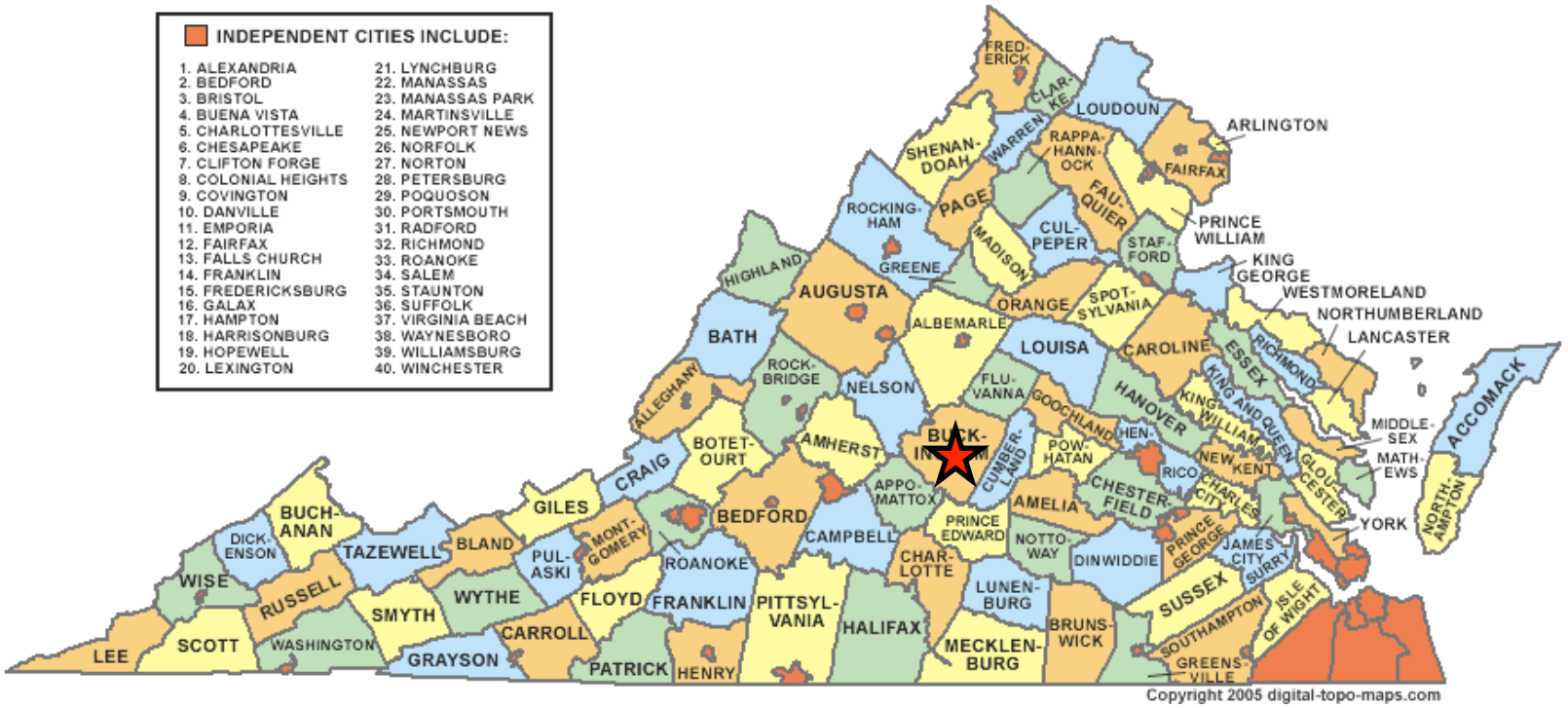
Based on Chapter 6 of the textbook and Ming Lin's lecture note at UNC and Robert Bless 's lecture note at WUSTL

Point Location

- Reading: Chapter 6 of the Textbook
- Driving Applications
  - Knowing Where You Are in GIS
  - Seismic ray tracing
- Related Applications
  - Triangulation using Trapezoidal Maps
  - Motion Planning: Cell decomposition

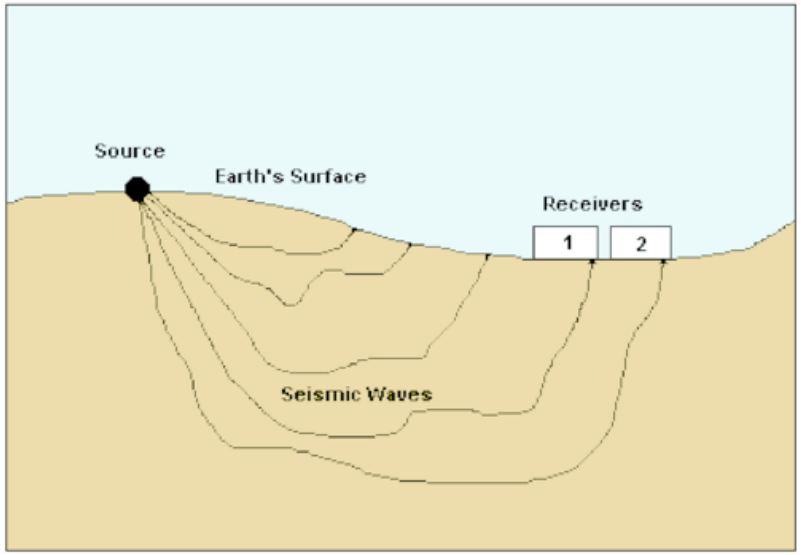
Knowing Where You Are

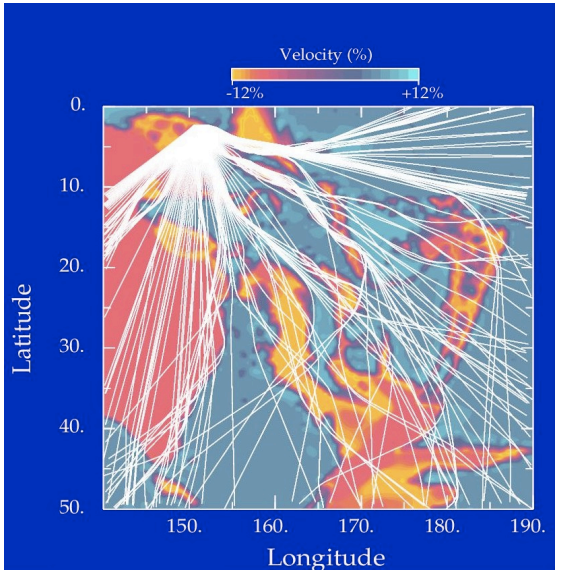
- Which county is the star located in?



Knowing Where You Are

- Simulating seismic ray tracing
  - 3D point location is needed
  - Many rays are propagated in side the earth



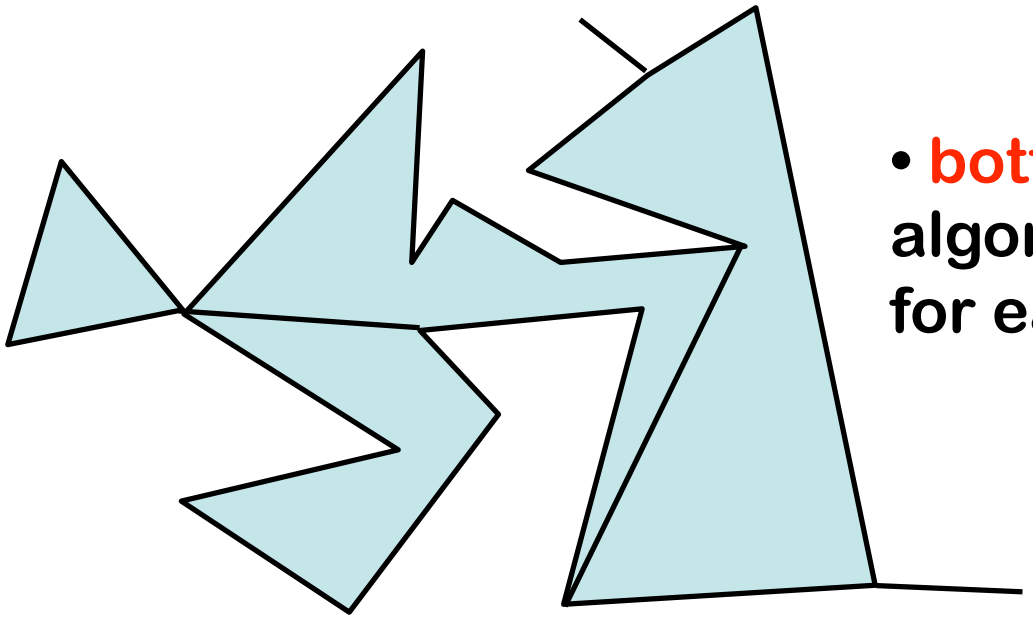


Knowing Where You Are

- Given a map and a query point q specified by its coordinates, find the region of the map containing q
- A map can be treated as a subdivision of the plane into regions, or planar subdivision

Planar Point Location

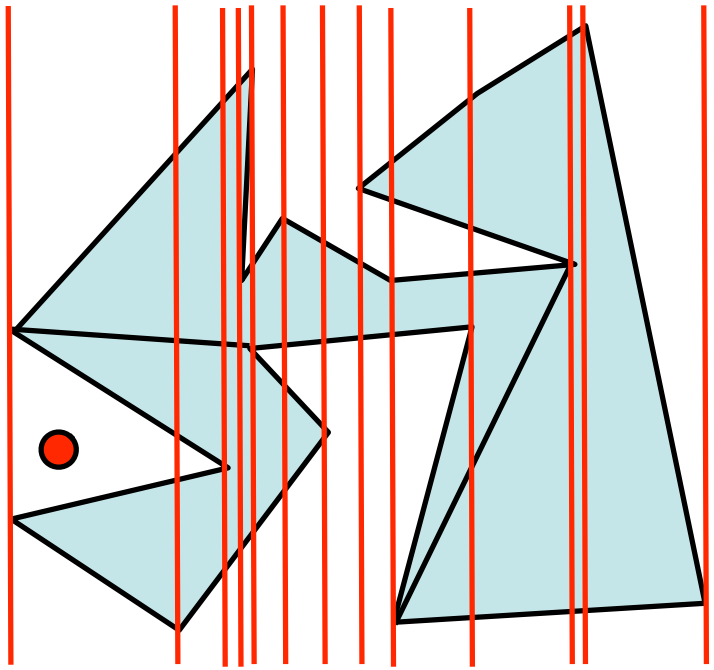
- Let S be a planar subdivision with n edges. The planar point location problem is to store S in such a way that we can answer: given a query point q, report the face f of S that contains q. If q lies on an edge or a vertex, report so



- **bottom line: brute force algorithm takes $O(n)$ time for each query**

Planar Point Location

- Divide the subdivision into a finer subdivision (slabs) using vertical lines passing through each vertex

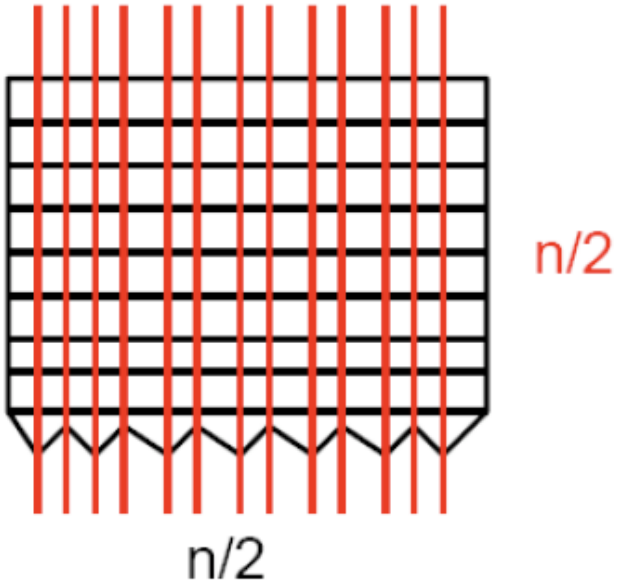


- **Using binary search to find which slab the point is in O(log n)**
- **Point location in a slab in O(log n)**

Total time O(log n)

A Possible Solution

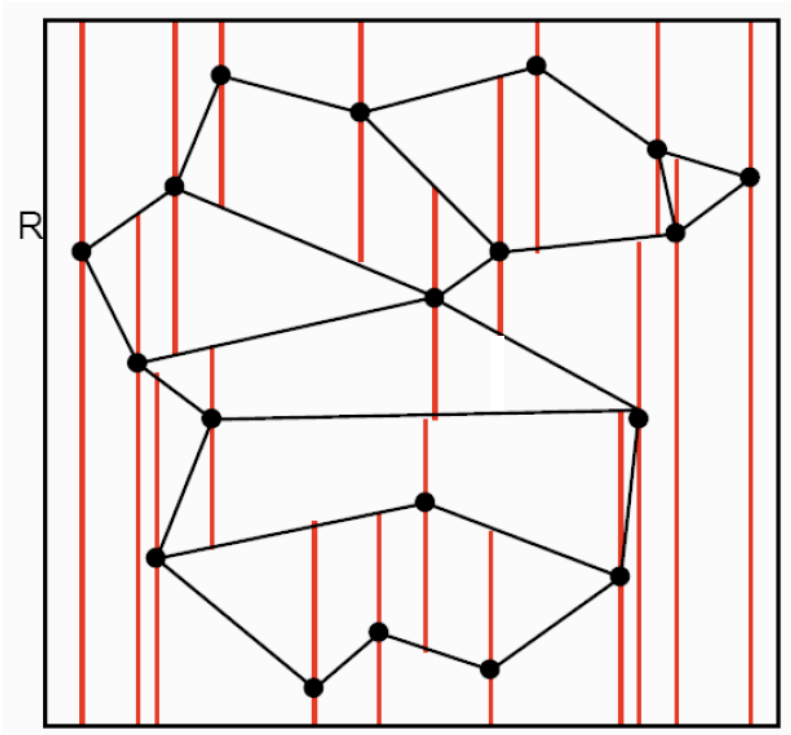
- But the storage requirement is high: $O(n^2)$!!!



⇒We need a better solution: trapezoidal maps!

Trapezoidal Maps

- A.k.a *vertical decomposition*
- A.k.a *trapezoidal decomposition*



Drawing two vertical extensions from every endpoint

- **one going upwards**
- **one going downwards**

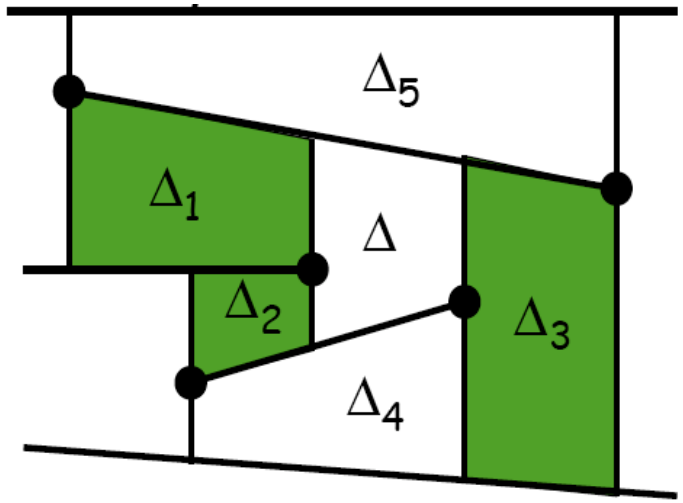
Assumption

- We add an outer boundary R

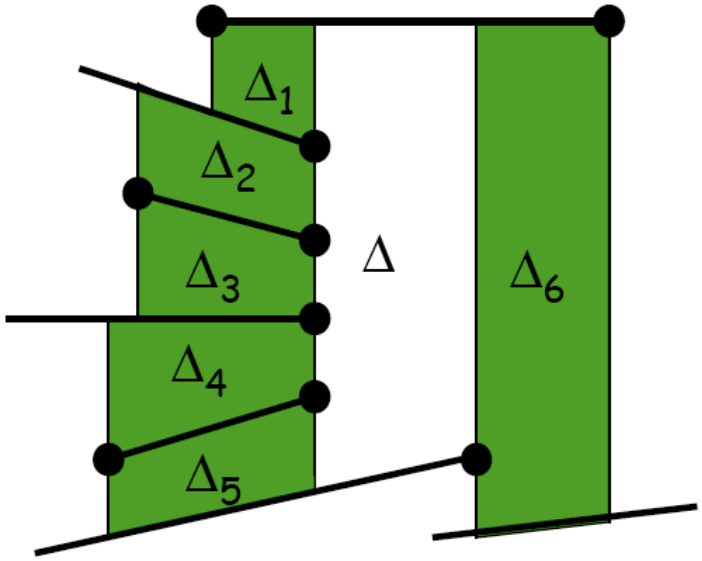
- No points with same x coordinates

Adjacent trapezoids

- Two trapezoids Δ and Δ´ are adjacent if they meet along a vertical edge



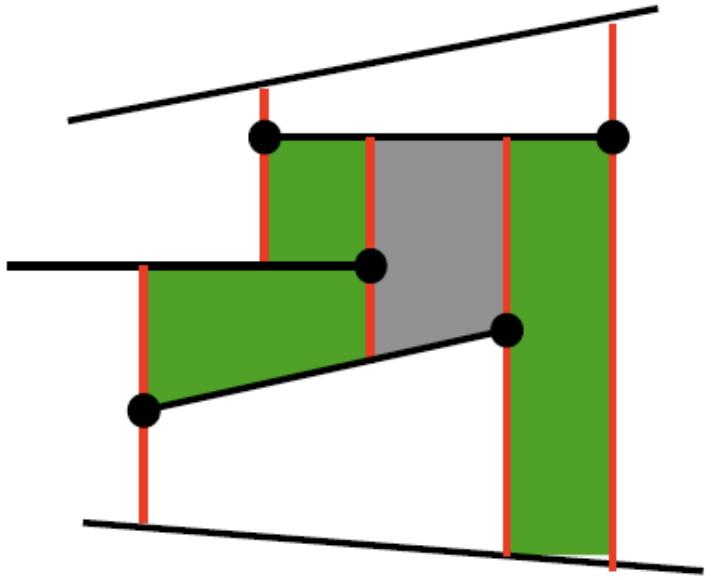
Segments in general position : A trapezoid has almost four adjacent trapezoids



Not true for this case

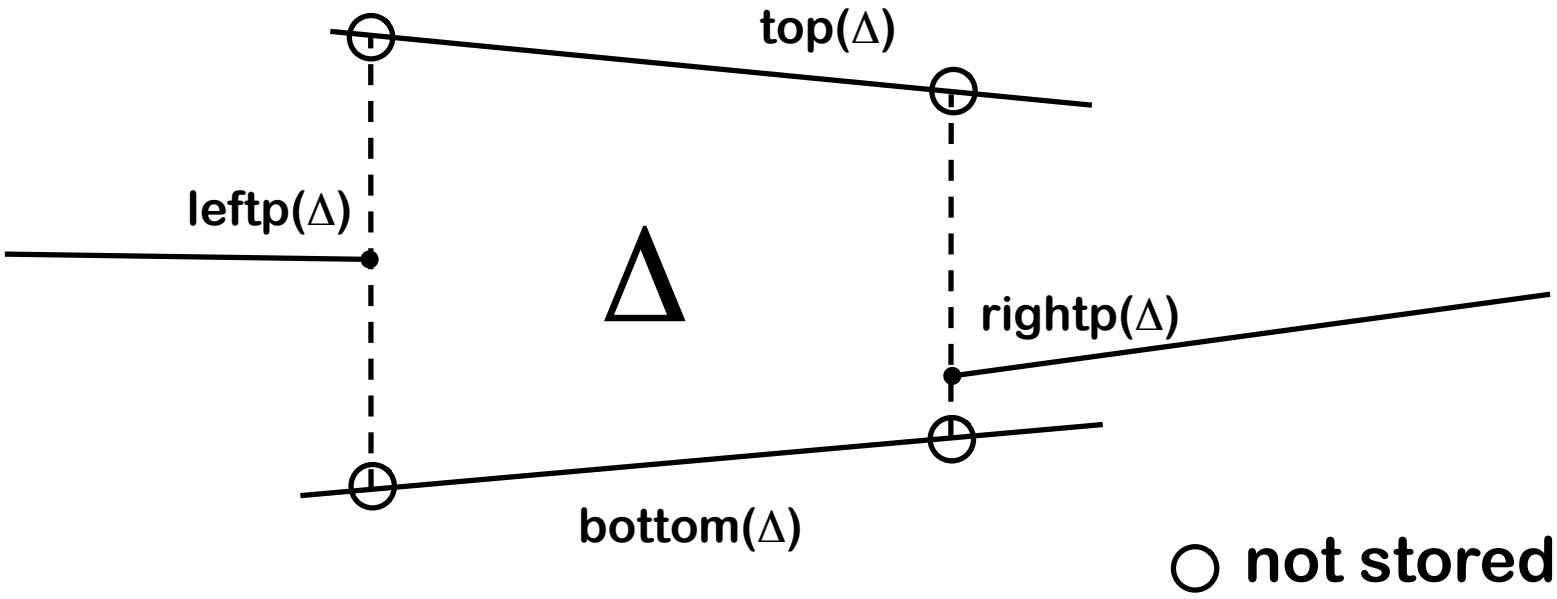
Adjacent trapezoids

- We keep pointers to neighboring trapezoids
  - How do we know which one is which one???
    - Left upper
    - Left lower
    - Right lower
    - Right upper



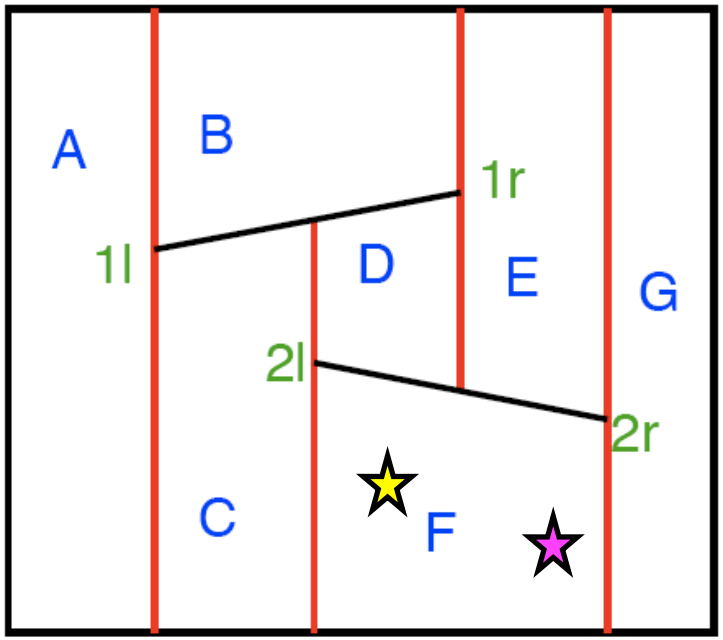
Trapezoidal Maps

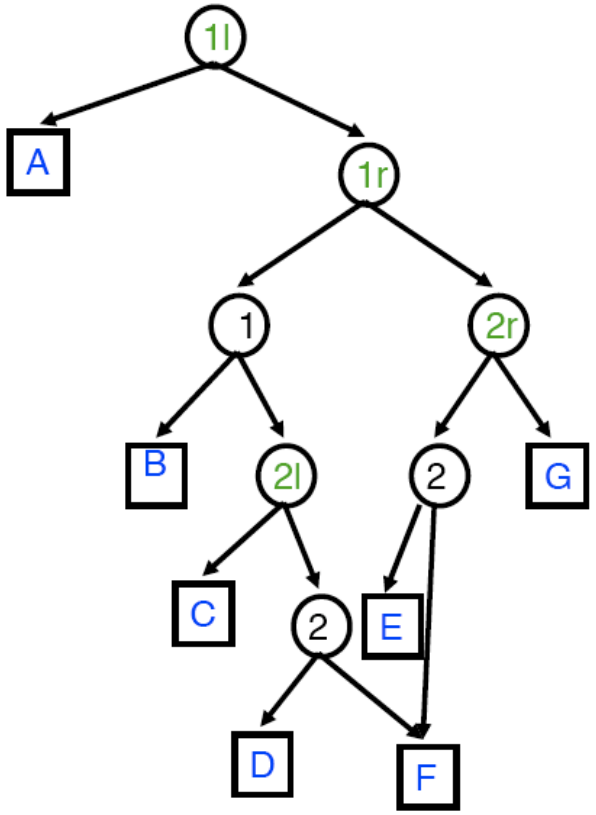
- A face is a trapezoid-like shape (trapezoid, triangle)
- In each trapezoid Δ we store:
  - leftp(Δ) - 4 pointers to its neighbors
  - rightp(Δ)
  - top(Δ)
  - bottom(Δ)



Building Trapezoidal Map

- Use randomized incremental algorithm to construct the trapezoidal map *T(S)*
- Maintain a point location data structure *D*





Building Trapezoidal Map

- The search structure D is a directed acyclic graph (DAG) with
  - a single root
  - exactly one leaf for every trapezoid of $T(S)$.
- There are 2 types of inner nodes (of out-degree 2):
  - x-nodes, labeled with an endpoint of some segment in S
  - y-nodes, labeled with a segment itself
- A query with q starts at the root and proceeds along a directed path to a leaf
  - At an x-node, the test is "Does q lie to the left or right of the vertical line through endpoint stored at this node?"
  - At a y-node, the test is "Does q lie above or below the segment s stored here?"

TrapezoidalMap(S)

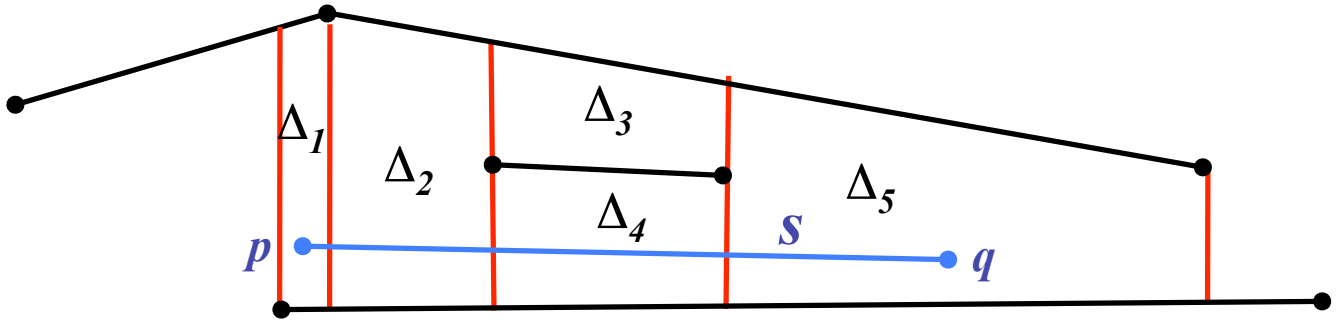
Input: A set S of n non-crossing line segments
Output: The trapezoidal map $T(S)$ and a search data structure D for $T(S)$ in a bounding box

1. Determine a bounding box R that contains all segments of S, and initialize trapezoidal map structure T & search structure D for it
2. Compute a random permutation $s_1, s_2, \ldots, s_n$ of the elements of S.
3. for $i \leftarrow 1$ *to* n do

   - Find set $\Delta_0, \Delta_1, \ldots, \Delta_k$ of trapezoids in T properly intersected by s_i

   - Remove $\Delta_0, \Delta_1, \ldots, \Delta_k$ from T and replace them by new trapezoids that appear because of the insertion of s_i

Find Intersecting Δs



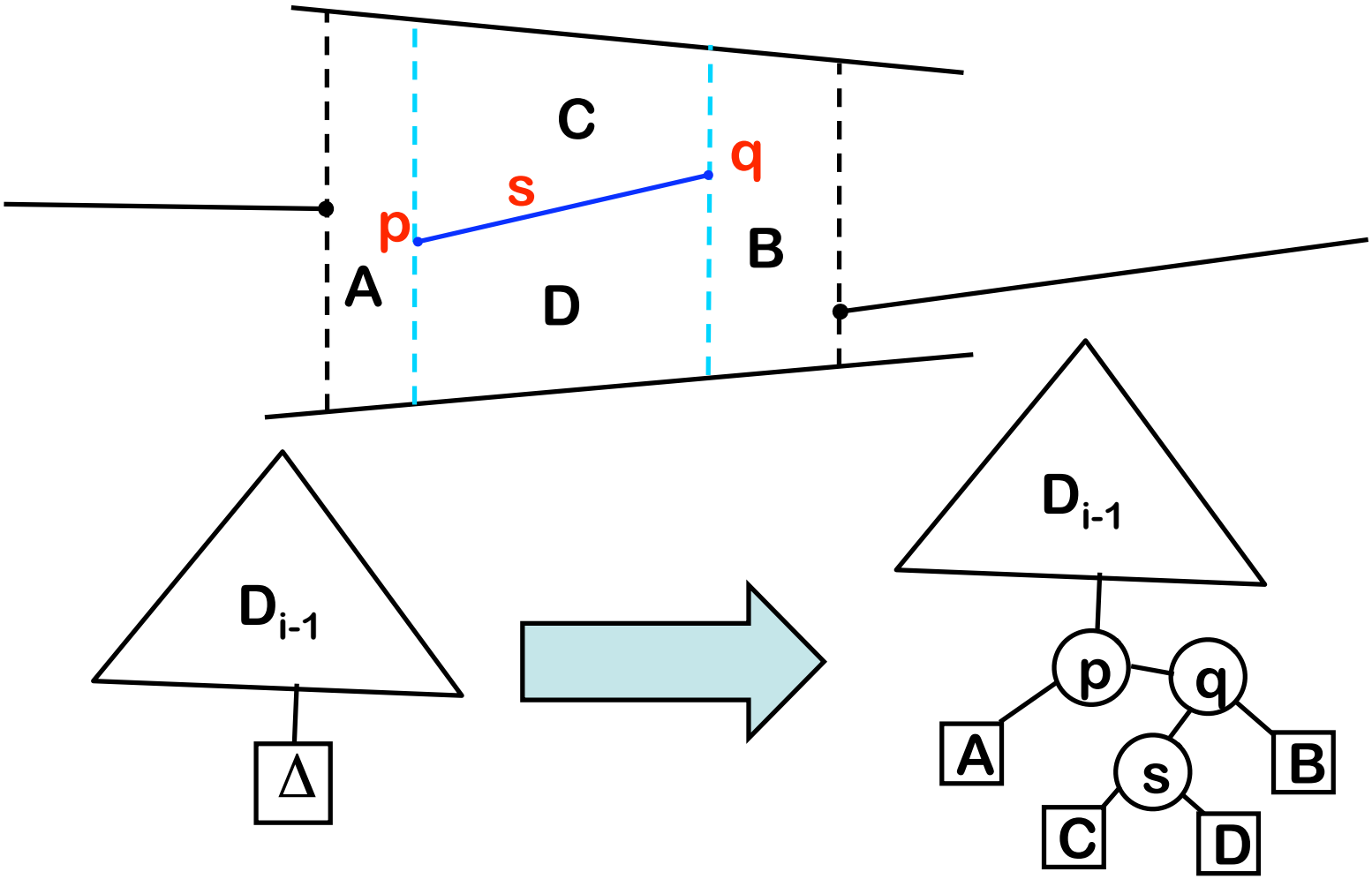
- Δ_1 can be found using the data structure D

FollowSegment(T, s_i)

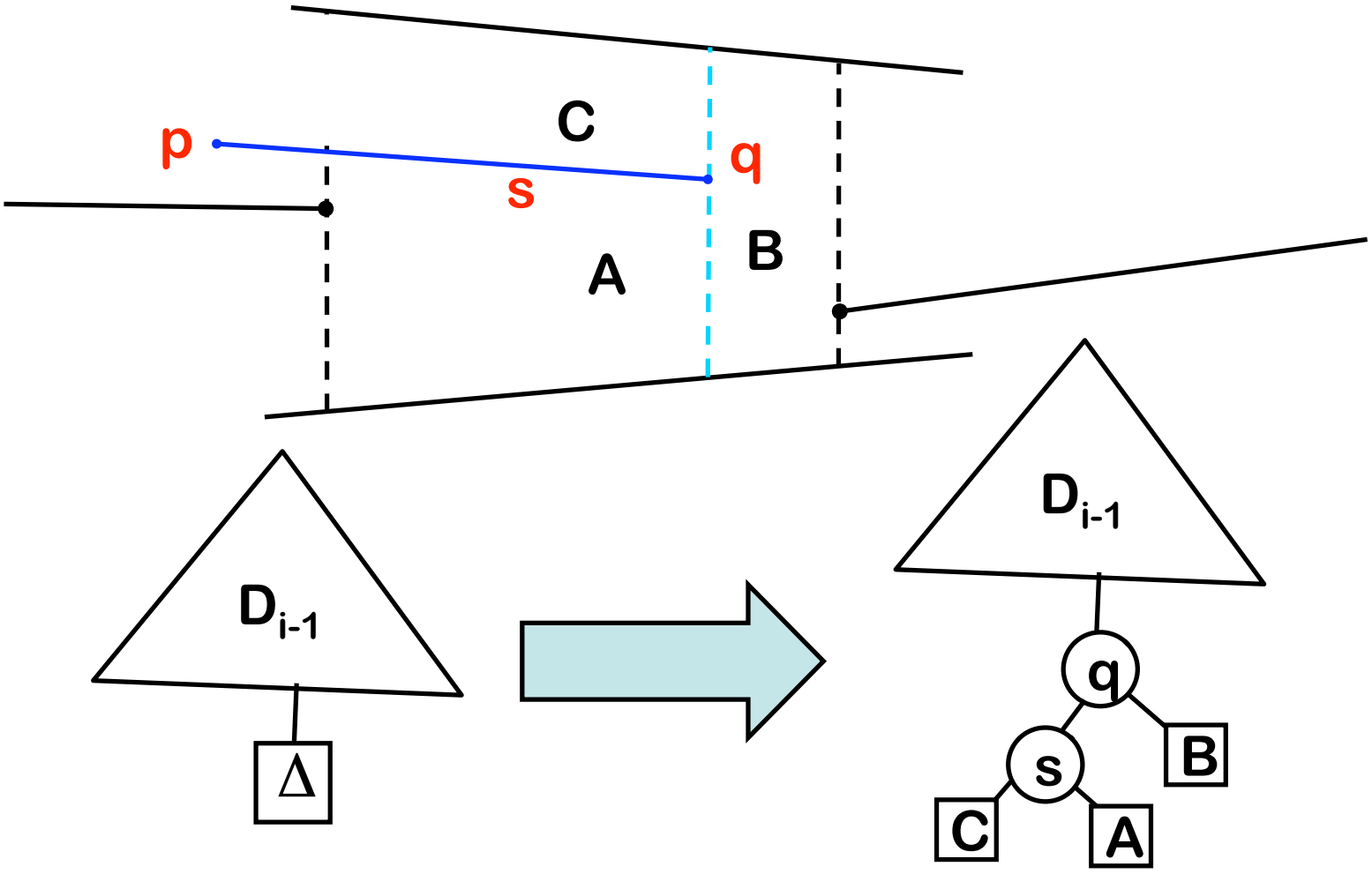
Input: A trapezoidal map T and a new segment s_i
Output: The sequence $\Delta_0, \Delta_1, \ldots, \Delta_k$ of trapezoids intersected by s_i

1. Let p and q be the left and right endpoint of s_i
2. Search with p in the search structure to find Δ_0
3. $j \leftarrow 0$
4. while q lies to the right of $rightp(\Delta_i)$
5. do if $rightp(\Delta_i)$ lies above s_i
6. then Let Δ_{j+1} be the lower right neighbor of Δ_j
7. else Let Δ_{j+1} be the upper right neighbor of Δ_j
8. $j \leftarrow j + 1$
9. return $\Delta_0, \Delta_1, \ldots, \Delta_j$

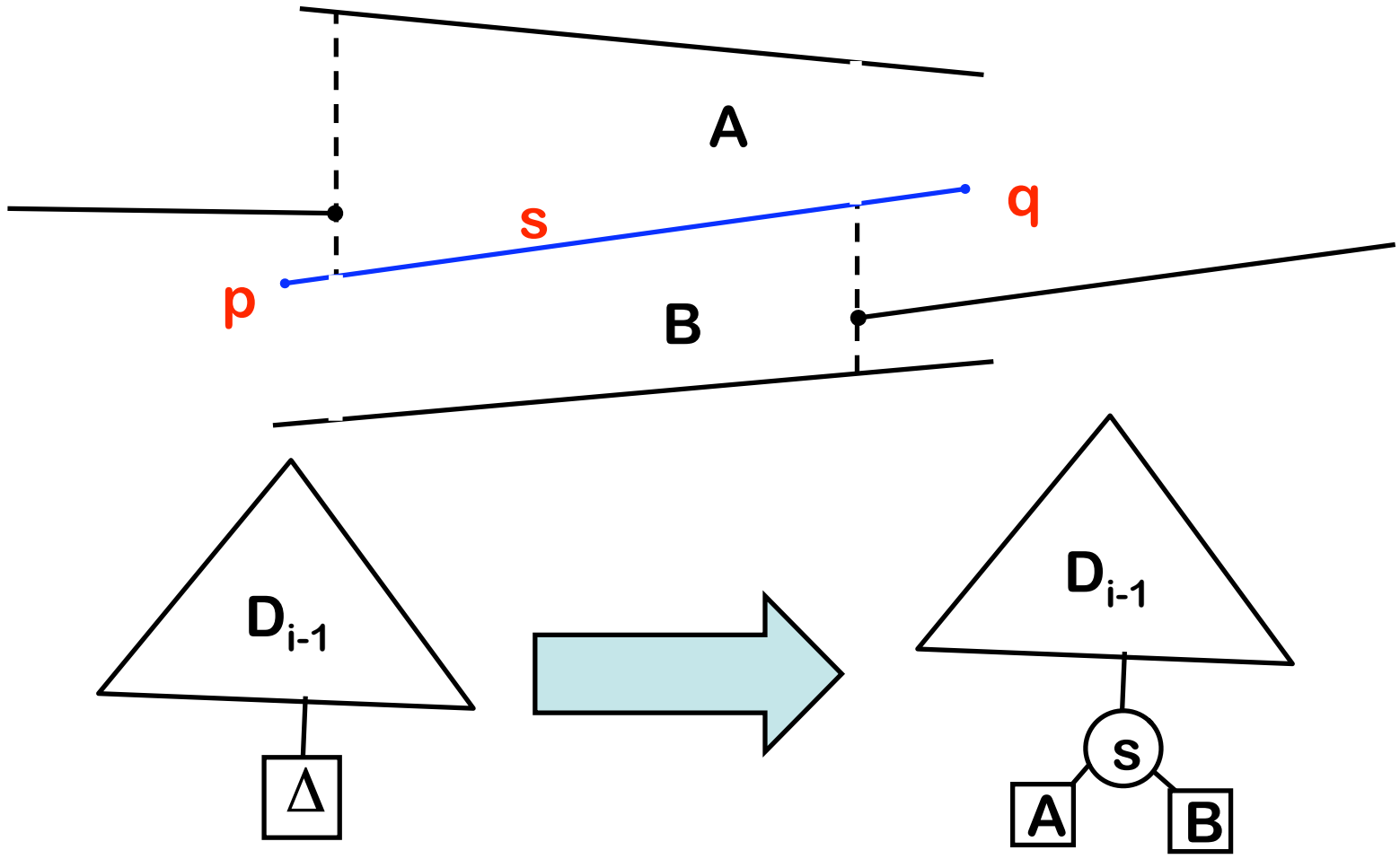
Case 1: New segment completely contained in trapezoid



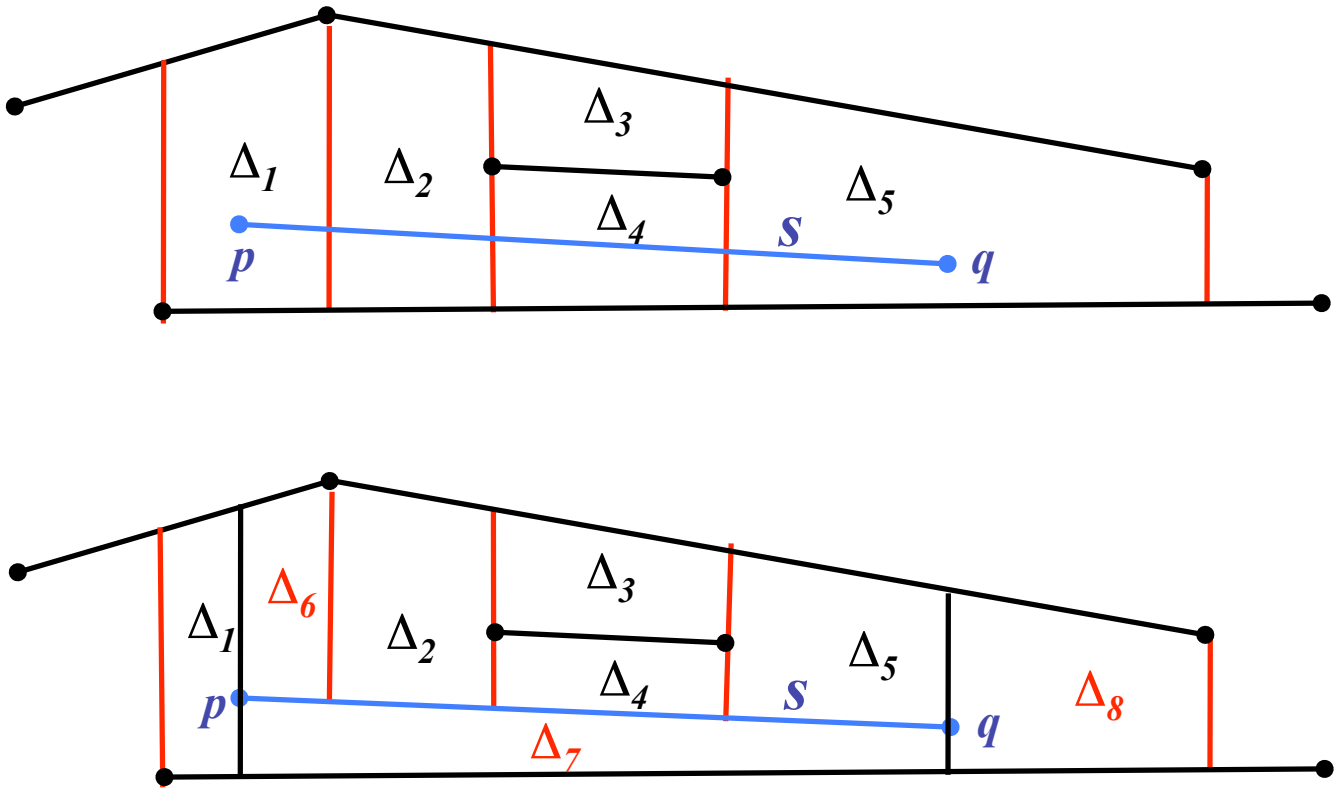
Case 2: New segment partially contained in trapezoid



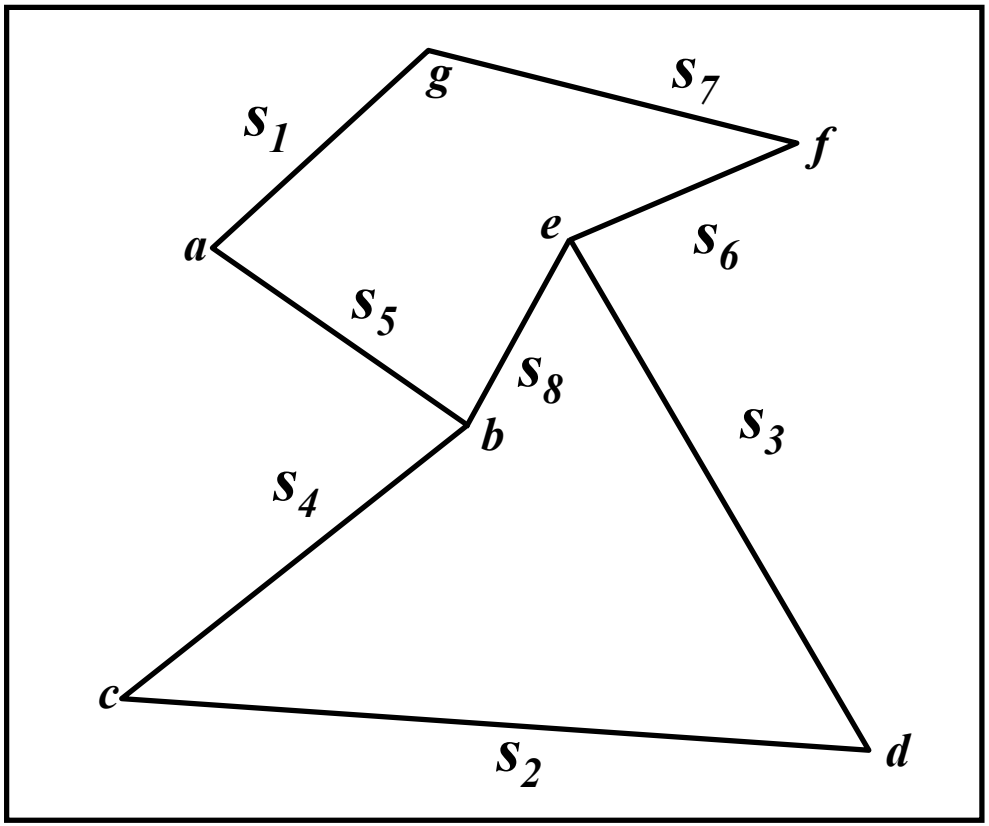
Case 3: New segment completely outside trapezoid



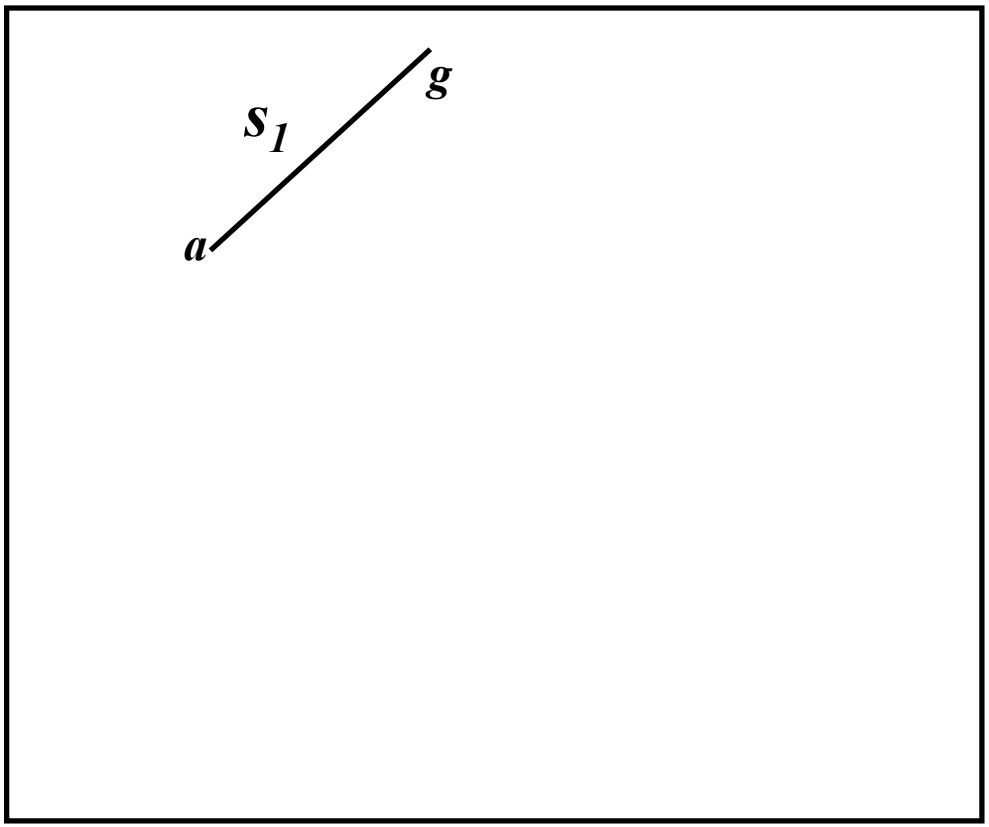
Short Example



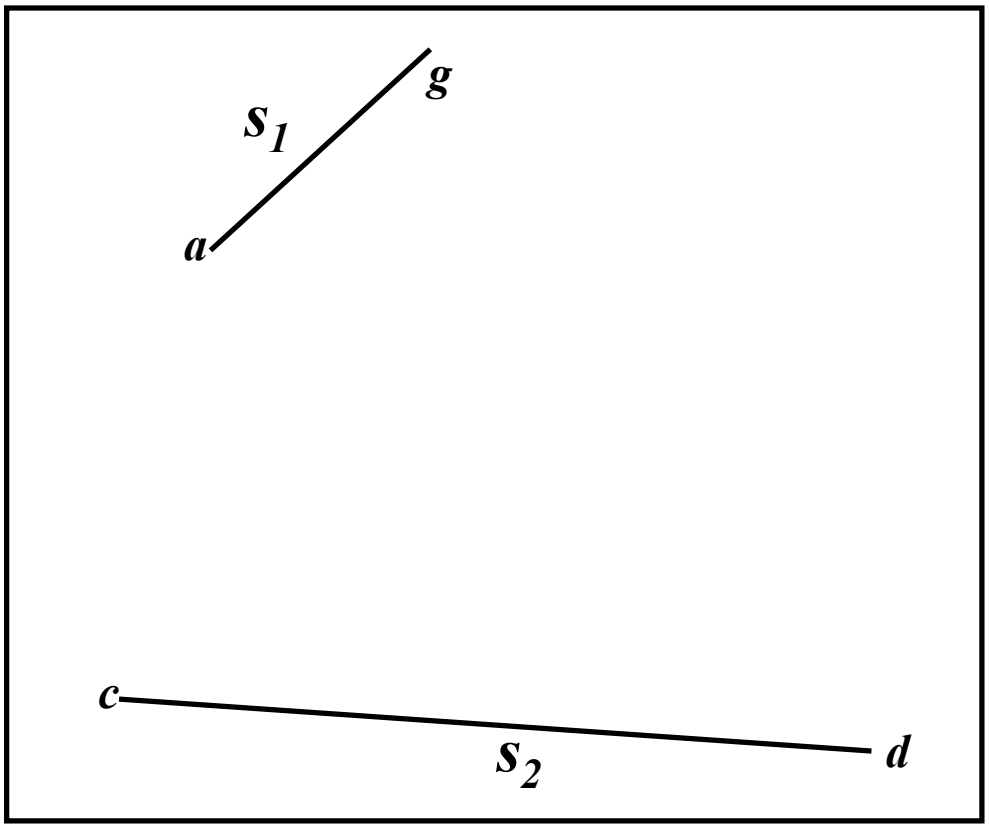
Another Example



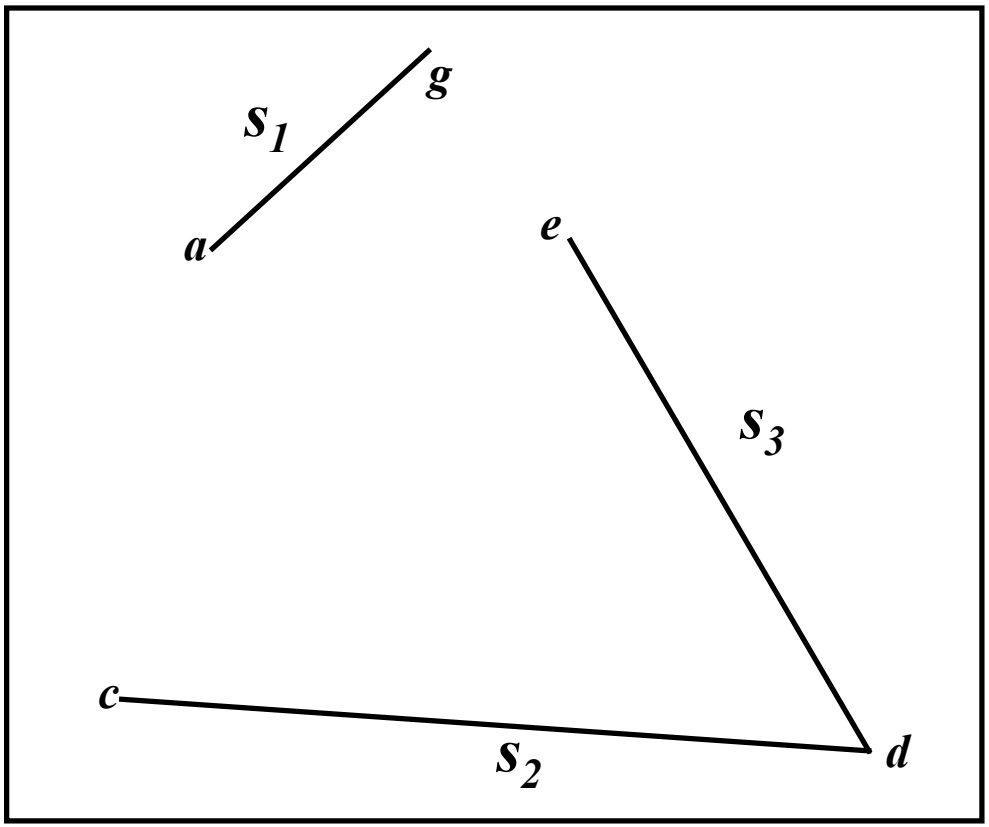
Another Example



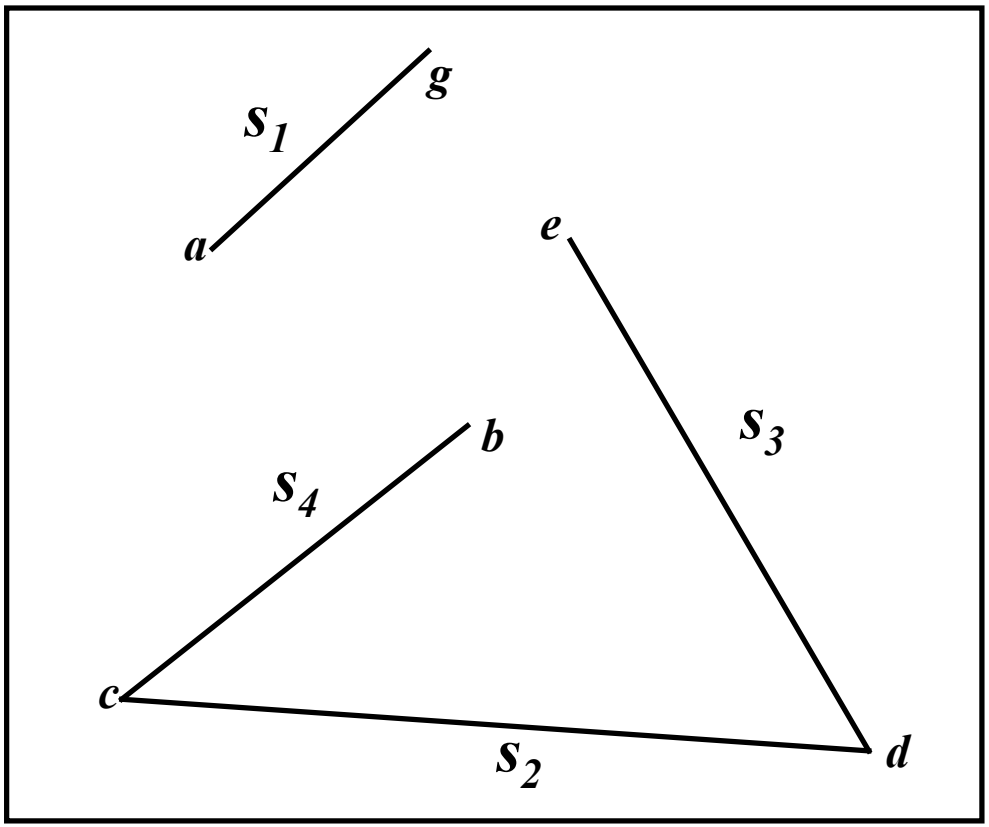
Another Example



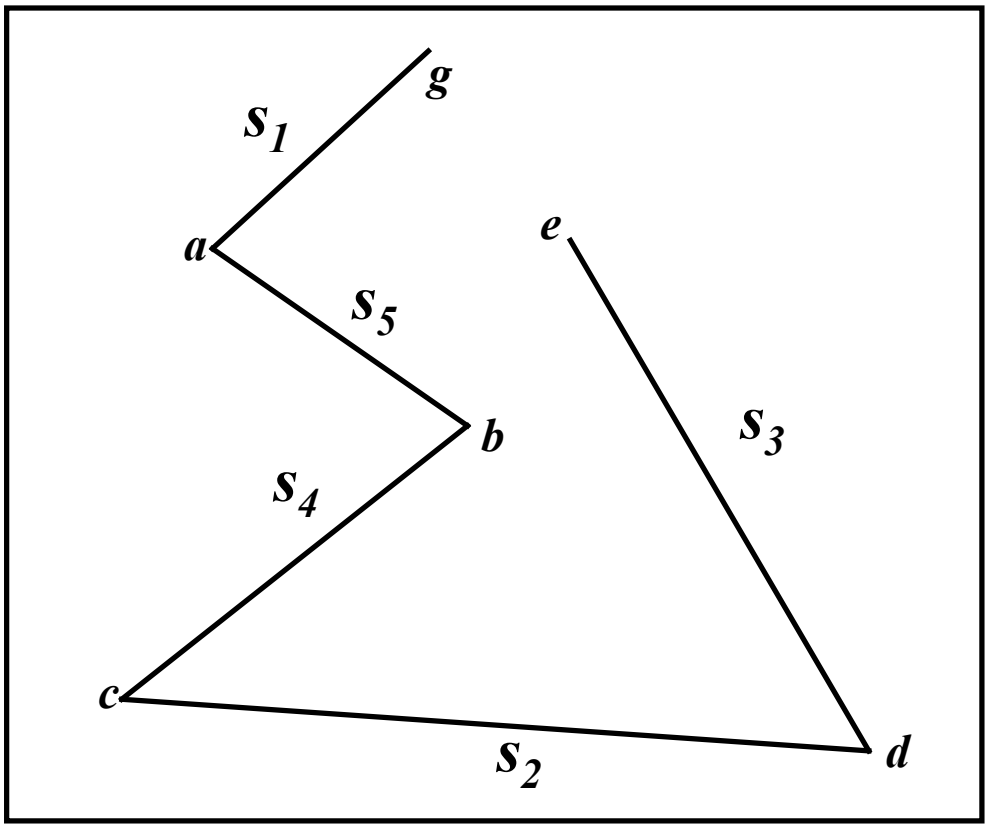
Another Example



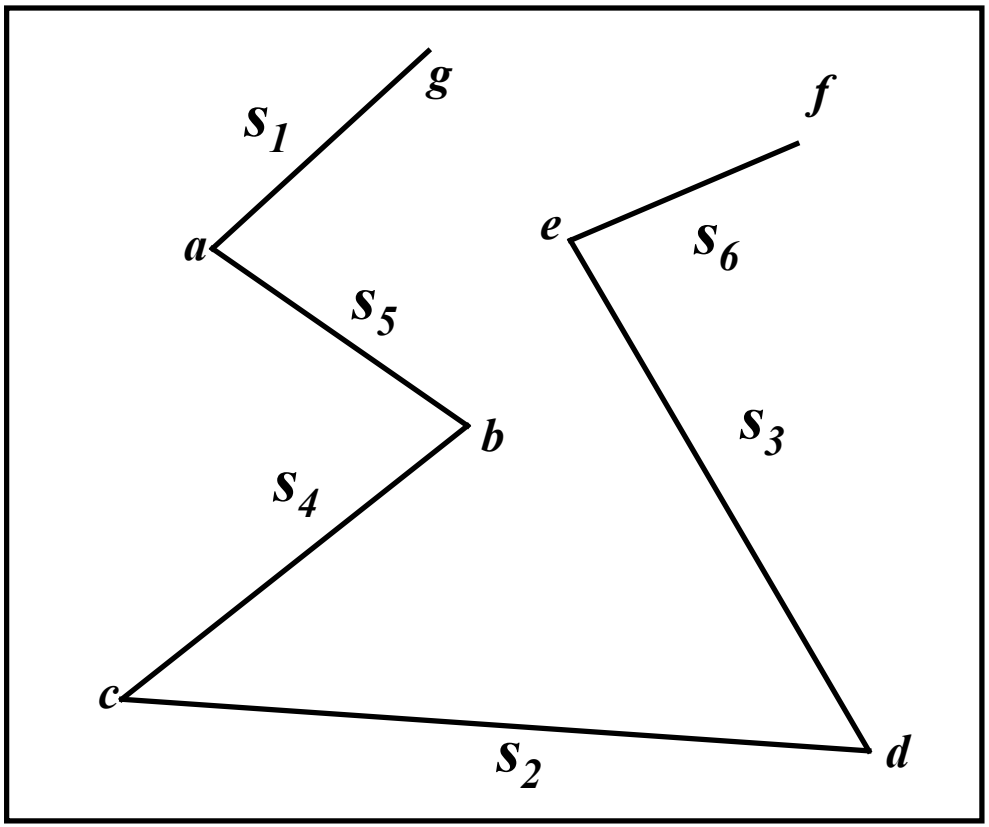
Another Example



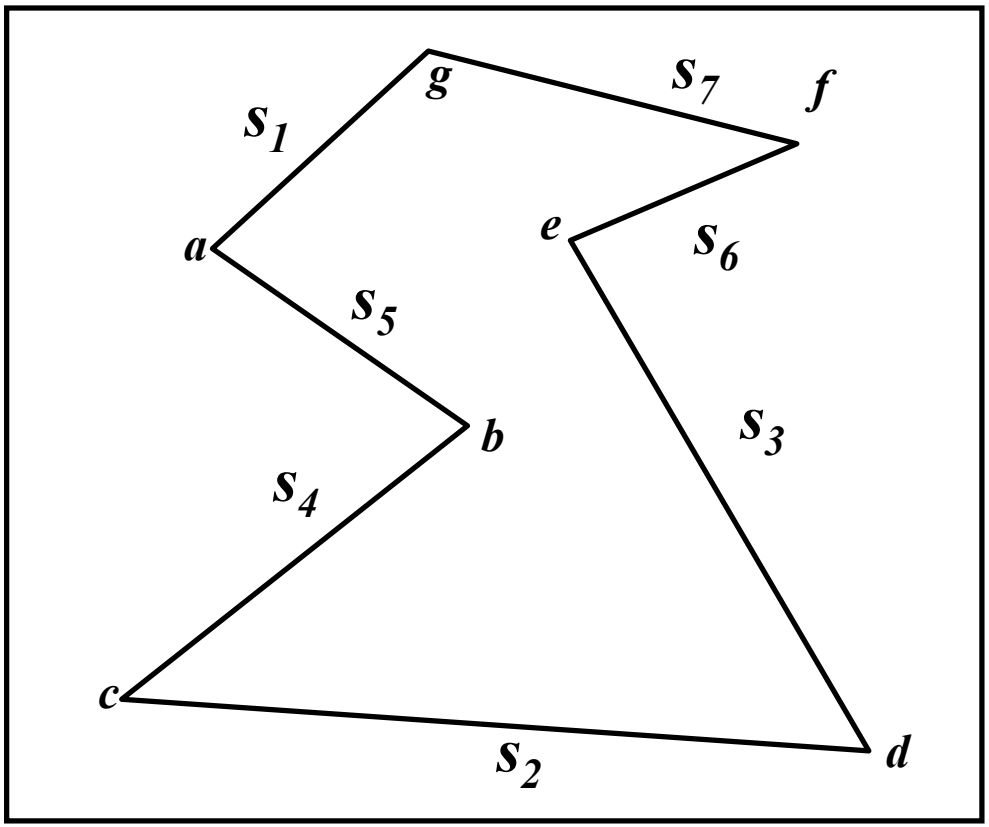
Another Example



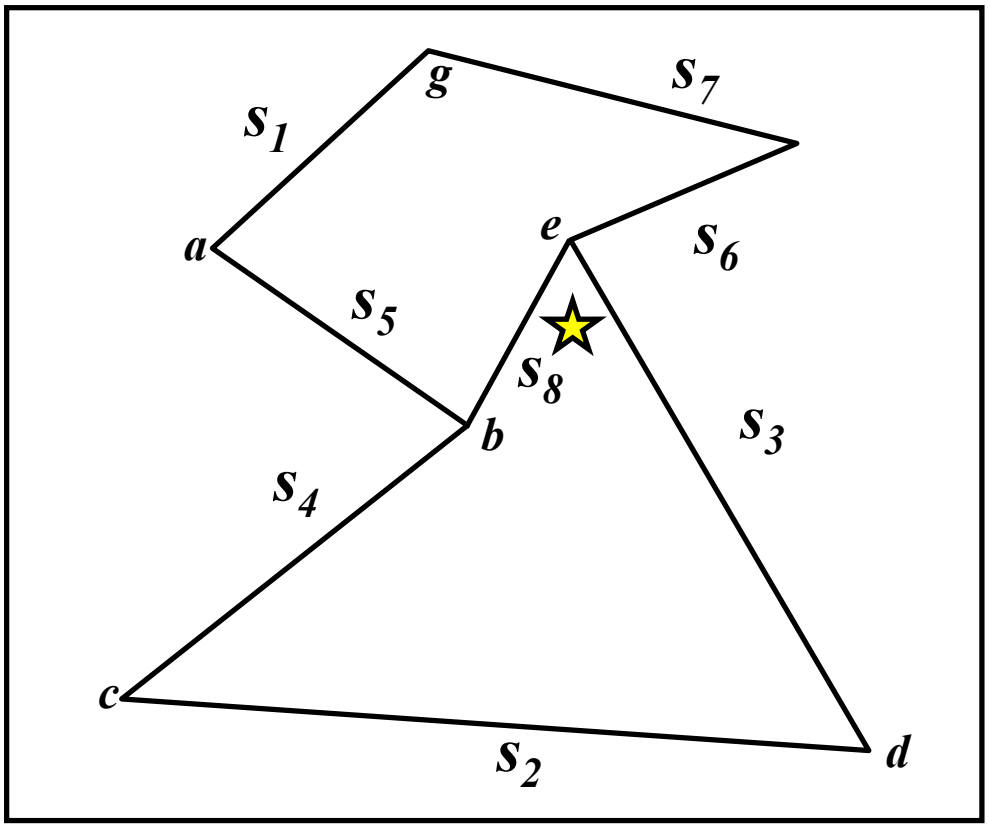
Another Example



Another Example

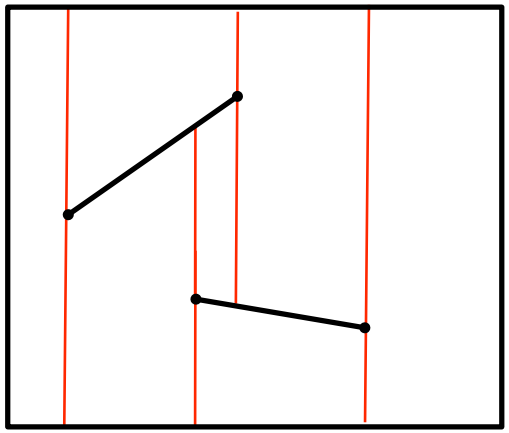


Another Example



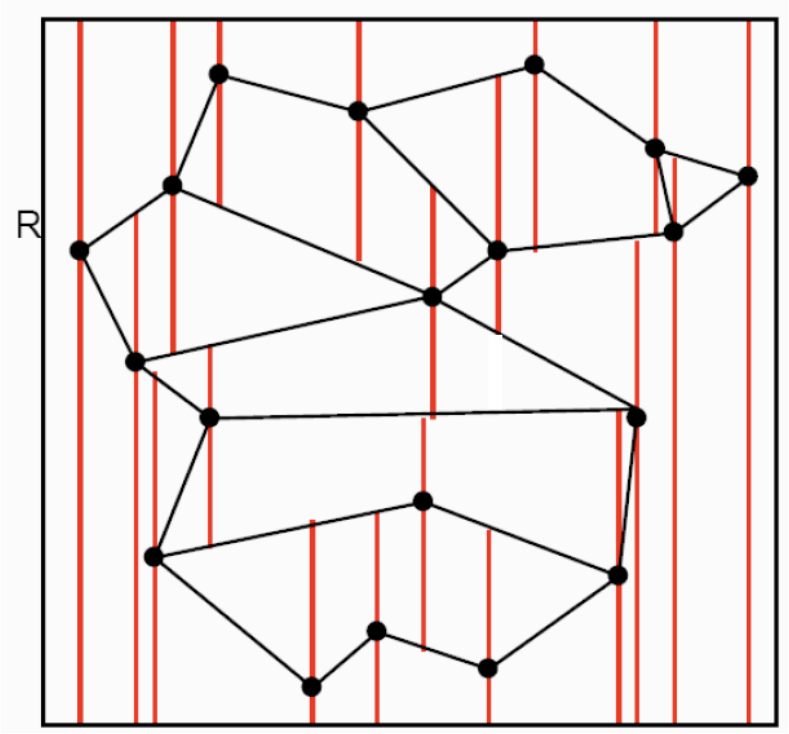
Size of Trapezoidal Map

- What's the size of such a decomposition?
  - Idea: relate size of the trapezoidal map to the size of the segments
  - Each face in a trapezoidal map of a set S of line segments in general position has 1 or 2 vertical sides and exactly two non-vertical sides



Size of Trapezoidal Map

- Left edge of a trapezoid in a trapezoidal map

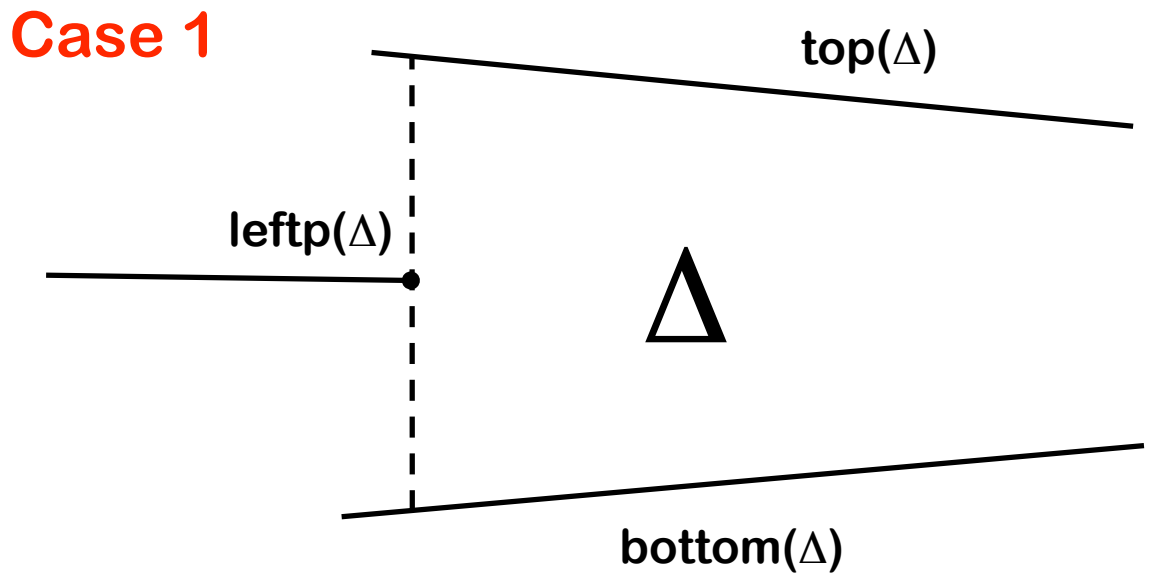


For every trapezoid Δ, except the left most one, the left edge is defined by an end point of a segment leftp(Δ)

Idea: There are as many "left edges of trapezoids" as trapezoids

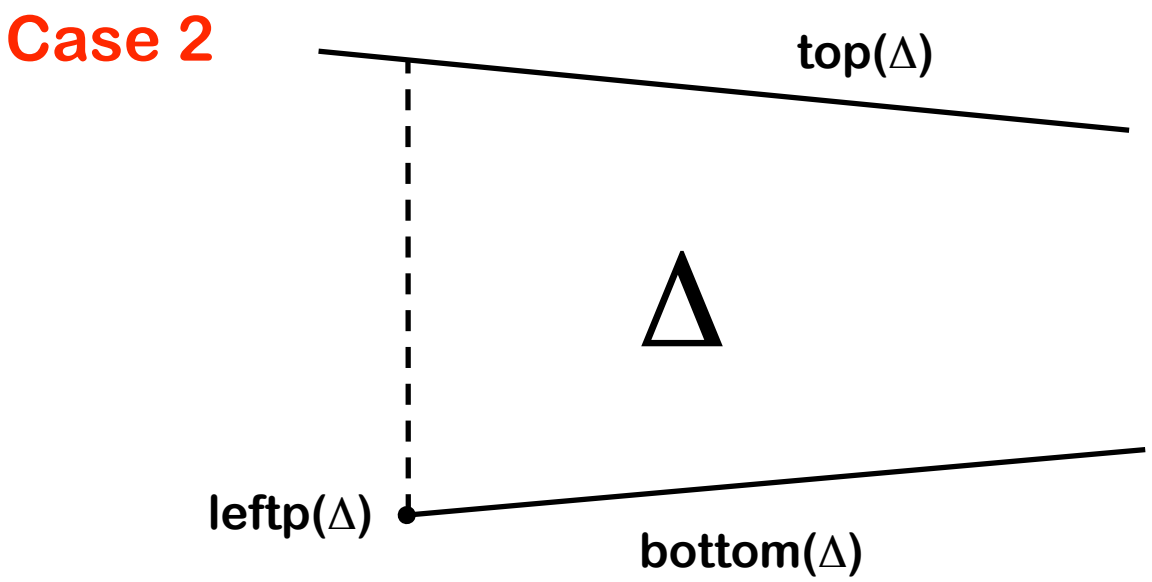
Size of Trapezoidal Map

- There are only five cases (for left edge of a trapezoid)



Size of Trapezoidal Map

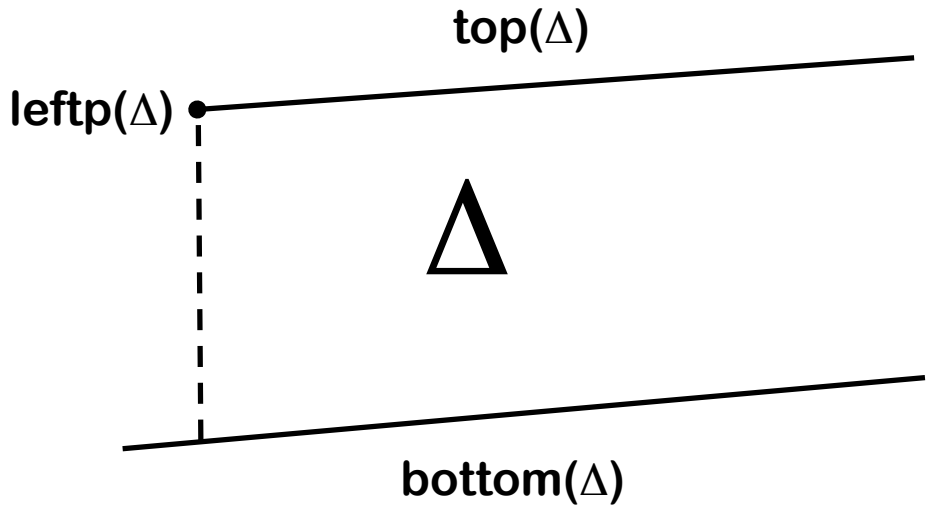
- There are only five cases (for left edge of a trapezoid)



Size of Trapezoidal Map

- There are only five cases (for left edge of a trapezoid)

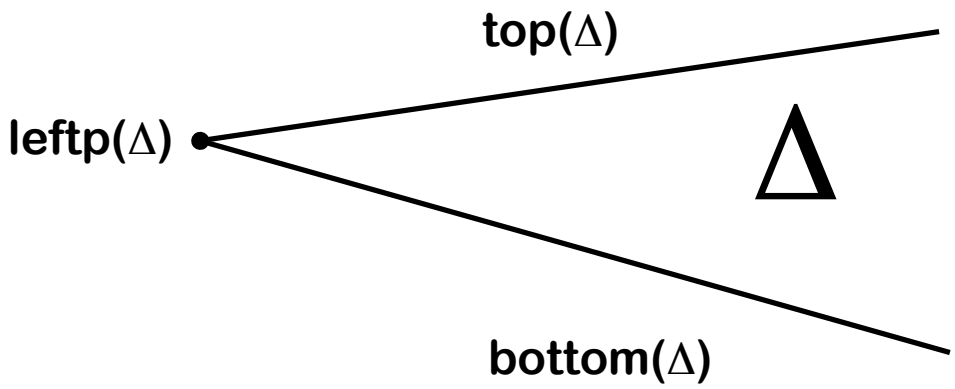
Case 3



Size of Trapezoidal Map

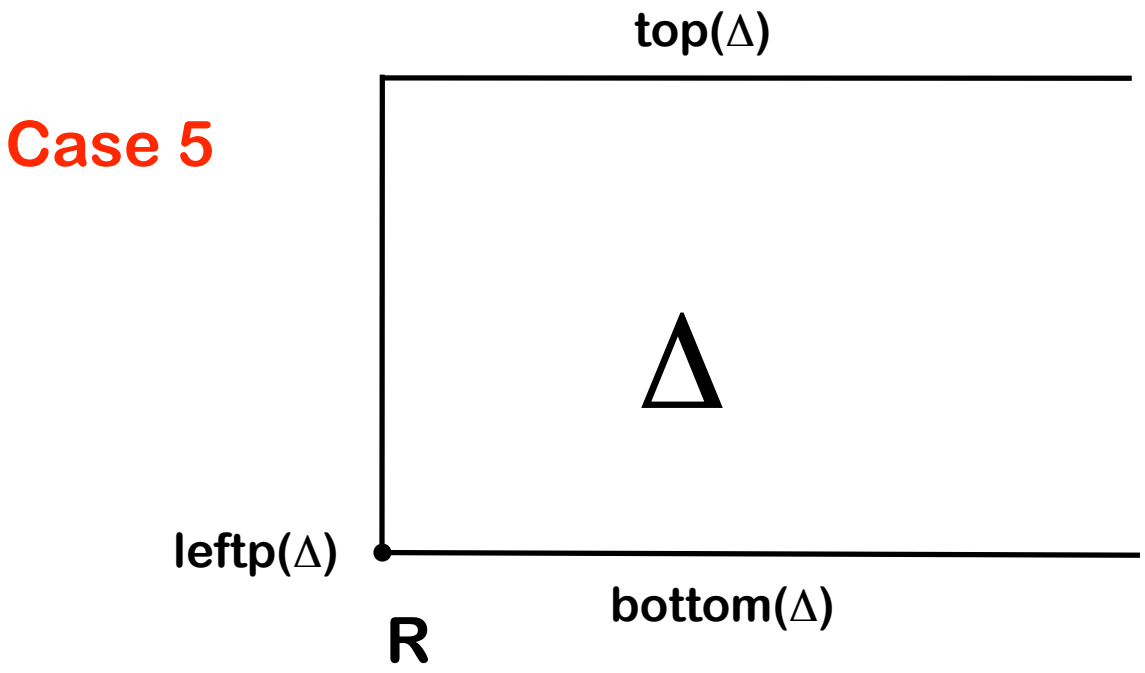
- There are only five cases (for left edge of a trapezoid)

Case 4



Size of Trapezoidal Map

- There are only five cases (for left edge of a trapezoid)



Size of Trapezoidal Map

- Theorem: The trapezoidal map *T(S)* of a set of n line segments *S* in general position contains at most 6n + 4 vertices

- Proof:
  - Bounding box R has *4* vertices
  - End points *2n* vertices
  - Vertical line end points *4n*
  - Total *6n+4*

Size of Trapezoidal Map

- Theorem: The trapezoidal map *T(S)* of a set of n line segments *S* in general position contains at most 3n + 1 trapezoids (i.e., left points of trapezoids)

- Proof:
  - R has *1* left point
  - Left point of a segment *2n*
    - Each such a point can be leftp(Δ) of 2 trapezoids
  - Right point of a segment *1n*
    - Each such a point can be leftp(Δ) of 1 trapezoids
  - Total *3n+1*

Randomized Incremental Algorithm

- Construction of the search structure is incremental:
  - It adds one segment at a time
  - After each addition, it updates *D* and *T(S)*
- The order in which the segments are added affects the construction of search structure *D*
  - In worst case, construction time is $O(n^2)$
    - Each new segment intersects all existing trapezoids
  - In worst case, query time is O(n)
  - Therefore we randomly order of the input and hope for good "expected" time complexity

Algorithm Analysis

- The expected running time is the average running time taken over all $n!$ permutations
  - The expected size of D is the average sizes of all $n!$ resulting search structures
  - The expected query time for q is the average query time for point q over all runs

Theorem:
Algorithm TrapezoidalMap computes the trapezoidal map $T(S)$ of a set S of n line segments in general position and a search structure D for $T(S)$ in $O(n \log n)$ expected time. The expected size of the search structure D is $O(n)$ and for any query point q the expected query time is $O(\log n)$.

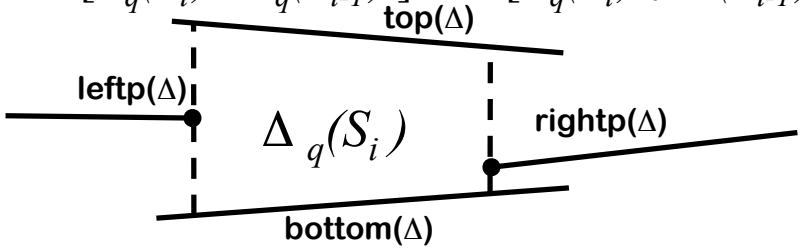
Average Query Time Analysis

- The query time for q is linear in the length of the path in D that is traversed when querying with q
- It is increased by at most 3 in every iteration based on case analysis
- So, $3n$ is the best possible worst-case bound over all possible insertion order for S
- But, we're interested in *average* query time w.r.t. all $n!$ possible insertion orders
- Let X_i, *for* $1 \leq i \leq n$, for denote the number of nodes on the path created in i*'th* iteration. We can express the expected path length as:

Average Query Time Analysis

$$X_i \le 3 \Rightarrow \mathrm{E}[X_i] \le 0*(1-P_i) + 3*P_i \Rightarrow \mathrm{E}[X_i] \le 3P_i$$

- What is P_i?
  - The probability that $\Delta_q(S_i)$ is created (i.e, $\Delta_q(S_i) \neq \Delta_q(S_{i-1})$) because of the segment s_i
  - Since s_i is selected at random, it has the same probability that $\Delta_q(S_i)$ is destroyed by randomly pick a segment from S_i (backward analysis!!)
  - $P_i = \Pr[\Delta_q(S_i) \neq \Delta_q(S_{i-1})] = \Pr[\Delta_q(S_i) \notin T(S_{i-1})] \le 4/i$



To destroy $\Delta_q(S_i)$ we can remove any of the four bounding edges

Average Query Time Analysis

$$E[\sum_{1 \le i \le n} X_i] \le \sum_{1 \le i \le n} 3P_i \le \sum_{1 \le i \le n} 12/i = 12 \sum_{1 \le i \le n} 1/i = 12H_n$$

$$H_n = 1/1 + 1/2 + 1/3 + \ldots + 1/n$$

$$\ln n < H_n < (\ln n + 1)$$

$\Rightarrow$ Therefore, the query time takes $O(\log n)$

Expected Size of the Structure

- To bound the size, it suffices to bound the number of nodes in D.
- The leaves in D are in one-to-one correspondence with the trapezoids in $T(S)$, of which there are $O(n)$.
- Let k_i be no. of new trapezoids created in iteration i, due to insertion of segment s_i :

$$O(n) + \mathrm{E}[\sum_{1 \le i \le n} (\textit{number of inner nodes created in iteration } i)]$$

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

Expected Size of the Structure

- Backward analysis
  - $\sum_{1 \le i \le n} E[k_i]$ is the # of the "segment" nodes
  - $E[k_i]$ is the expected number of new trapezoids created by the *i-th* segment
  - Each s_i has the same probability of being the *i-th* segment
    - $E[k_i]$=(# of trapezoids destroyed by removing s_0+
      # of trapezoids destroyed by removing s_1+
      # of trapezoids destroyed by removing s_2+
      .....
      # of trapezoids destroyed by removing s_i) / i

    - $E[k_i] = (1/i) \sum_{s \in Si} \sum_{\Delta \in T(Si)} \delta(\Delta, s_i)$
      - $\delta(\Delta, s_i) = 1$ if Δ is bounded by S_i
      - $\delta(\Delta, s_i) = 0$ otherwise
      - $\sum_{s \in Si} \sum_{\Delta \in T(Si)} \delta(\Delta, s) \le 4 |T(S_i)| = O(i)$

    - $E[k_i] = (1/i) \sum_{s \in Si} \sum_{\Delta \in T(Si)} \delta(\Delta, s) \le O(i)/i = O(1)$

Expected Construction Time

- The time to insert segment s_i is $O(k_i)$ plus the time needed to locate the left endpoint of s_i in $T(S_{i-1})$. Use the earlier bound on k_i, we get the expected running time for the construction:

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

Naïve Assumptions

- General position statement assumes no two distinct points have the same x-coordinate.

- Assume that a query point never lies on the vertical line of an x-node on its search path, nor on the segment of a y-node.

Dealing with Degeneracies

- Use symbolic perturbation. In effects, use an affine mapping called "shear transform" along the x-axis.
  - (x,y) becomes $(x+\varepsilon y, y)$

- The algorithm does not compute any geometric objects; it never actually computes coordinates of the endpoints.

- All it does is to apply 2 elementary operations to the input points:
  - Take 2 distinct points p & q and decides whether q lies to the left, right or on the vertical line through p.

  - Take 1 of the input segments, specified by p_1 & p_2, and tests whether a third point q lies above, below, or on this segment. It is only applied when a vertical line through q intersects with this segment.

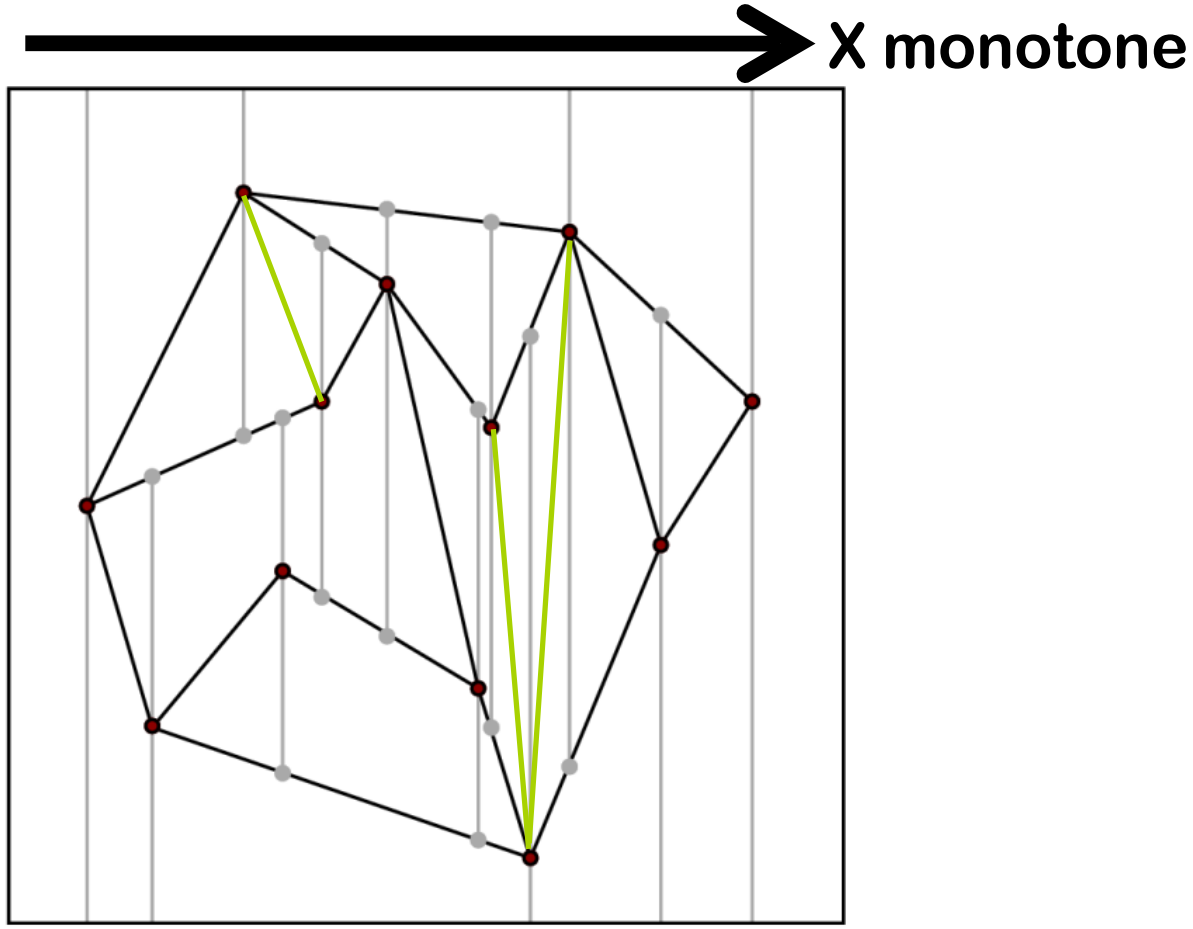
Application: Triangulation

- **1. Decompose the Polygon into Trapezoids.**
  - Seidel proves that if each permutation of **s1 .. sN** is equally likely then trapezoid formation takes $O(n \log^* n)$ expected time

- **2. Decompose the Trapezoids into Monotone Polygons.**
  - A monotone polygon is a polygon whose boundary consists of two y-monotone chains. These polygons are computed from the trapezoidal decomposition by checking whether the two vertices of the original polygon lie on the same side. This is a linear $O(n)$ time operation

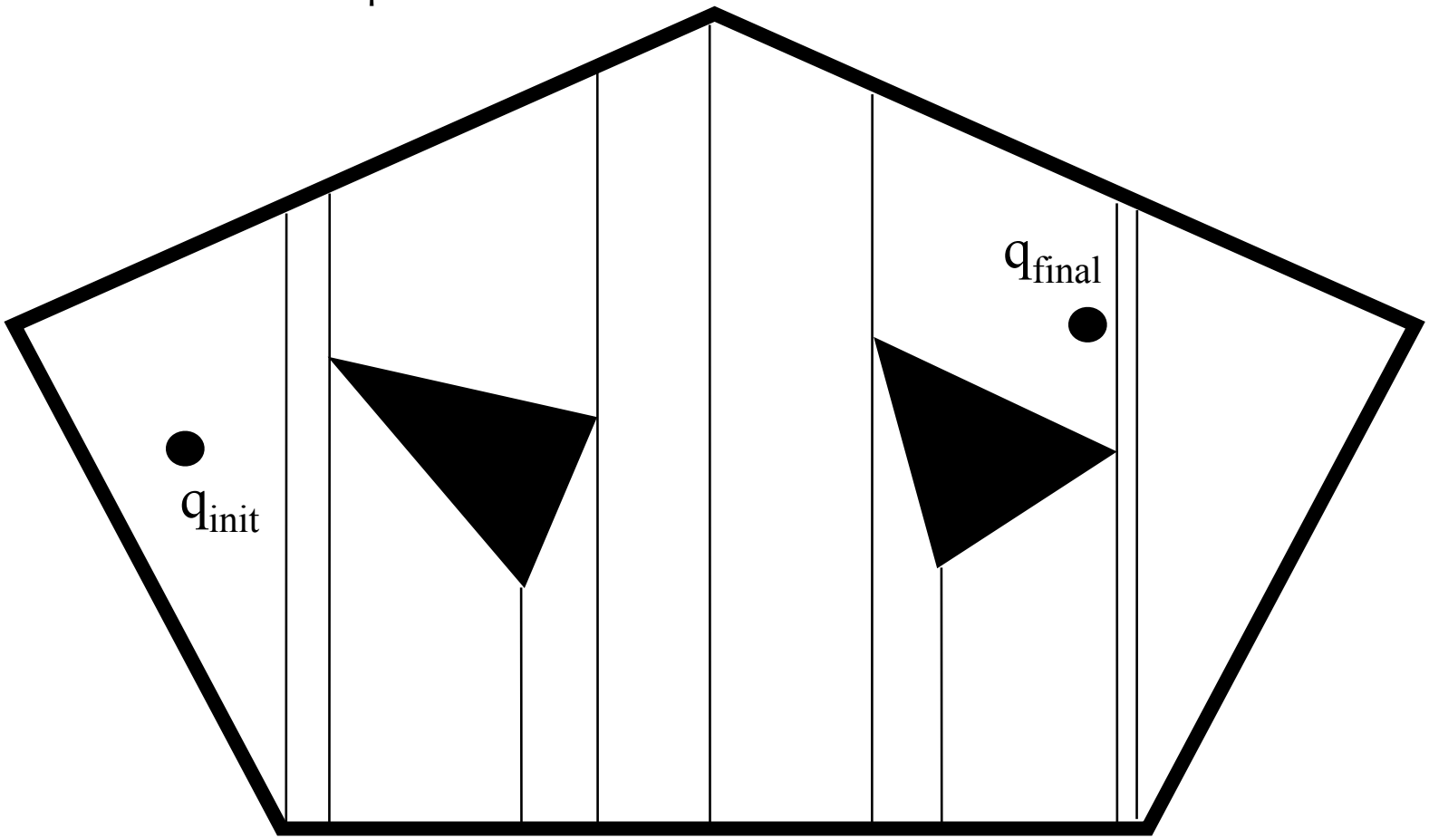
- **3. Triangulate the Monotone Polygons.**
  - all the monotone polygons can be triangulated in $O(n)$ time

Application: Triangulation



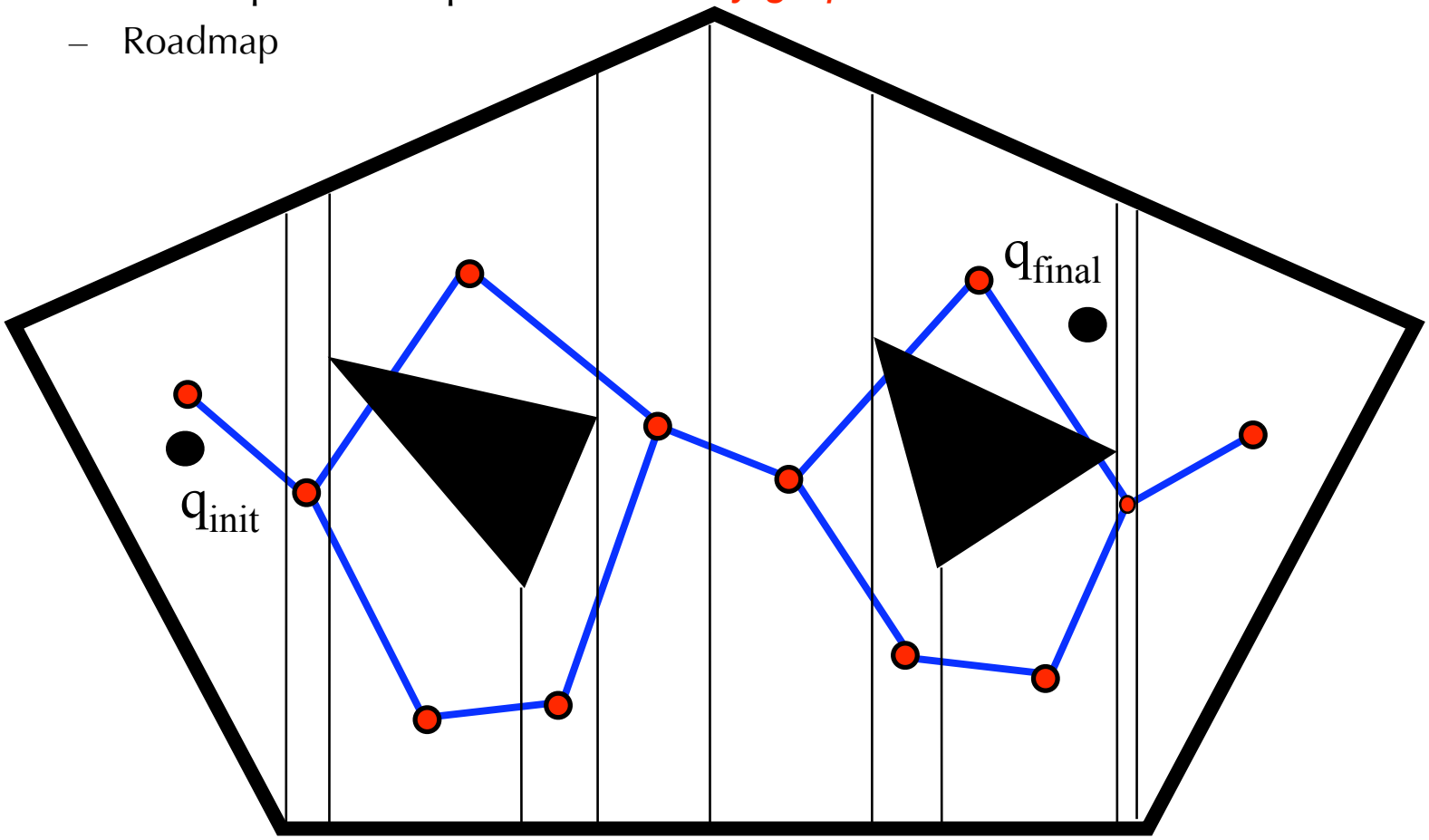
Application: Motion Planning

- Exact cell decomposition – from Latombe 2000



Application: Motion Planning

- Convert trapezoid map to *connectivity graph*
  - Roadmap



Conclusion

- Point location
  - Brute force has O(n) time complexity
  - Slab approach has O(log n) time but O(n^2) space complexity
  - Trapezoid decomposition allows O(log n) expected time but O(n) space complexity
- Trapezoid decomposition
  - randomized incremental algorithm
  - Sweep line algorithm (does not provide the point location data structure D)

Homework Assignment

- 6.5, 6.6, 6.7