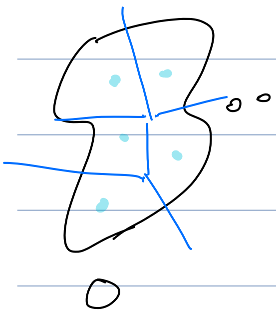
TCS Presentation : Voronoi Diagrams – Jisun Baek

<1. Introduction>



Given n points (sites) that provide certain goods or services.

want For each site where the people live who obtain their goods from that site.

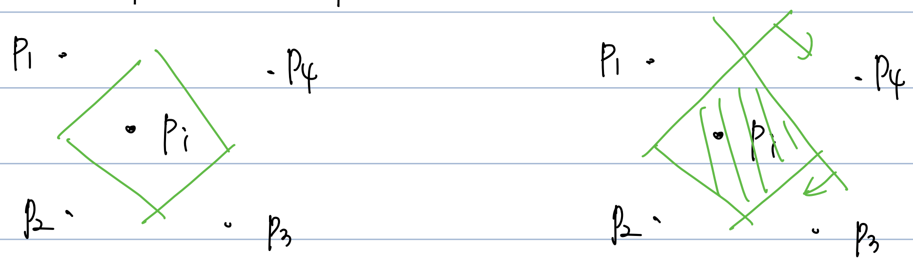
assumption
- Every site : same price of goods
- cost = price + cost of transportation. ∝ Euclidean distance
- People want to minimize the cost.

Divide the plane into n cells!

<2. Basic Properties of Voronoi Diagrams>

– Def
- Euclidean distance : $d(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$
- The set of sites : $P := \{p_1, \cdots, p_n\}$
- $Vor(P)$: Voronoi diagram of P. (subdivision of the plane into n cells)
  or just simply denote its edges/vertices.
- $V(p_i) :=$ Voronoi cell of p_i.

#1. Structure of $\mathcal{V}(p_i)$ for each p_i.



<u>Def</u> $q \in \mathcal{V}(p_i)$ iff $\forall j \neq i \, (d(q, p_i) < d(q, p_j))$.

i.e. $\mathcal{V}(p_i)$ is an intersection of half plane

which divides p_i and p_j $(j \neq i)$ equally and contains p_i.

<u>Def</u> $h(p,q) = \{ r \mid d(r,p) < d(r,q) \}$. Then $\mathcal{V}(p_i) = \bigcap_{\substack{1 \le j \le n \\ j \neq n}} h(p_i, p_j)$. (obs 7.1)

<u>Coro</u> $\mathcal{V}(p_i)$ is an open convex polygonal region bounded by at most $n-1$ vertices/edges.

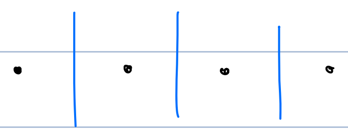
(possibly unbounded − 개방형)

Each edge is a bisector of p_i and p_j.

#2. How does each edge look like among full-line, half-line and line segment?

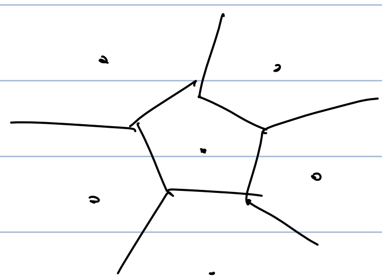
(Thm 7.2)

case 1) $\exists$ a full line iff all sites are collinear (한 직선 위에)



4 cells

case 2) o.w., Vor(P) is <u>connected</u> & its edges are either <u>segments</u> or <u>half-lines.</u>



3. Complexity? (i.e. the total number of vertices and edges)

n-cells & each cell has at most $n-1$ edges $\rightarrow$ complexity $= O(n^2)$.

But it is not the case. (Thm 7.3) Always linear!

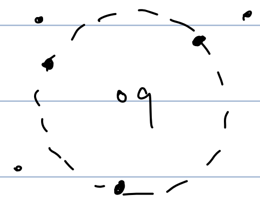
"For $n \geq 3$, # vertices $\leq 2n-5$ and # edges $\leq 3n-6$."

pf) Make the diagram bounded by adding V_∞ and connecting it to every half-lines.

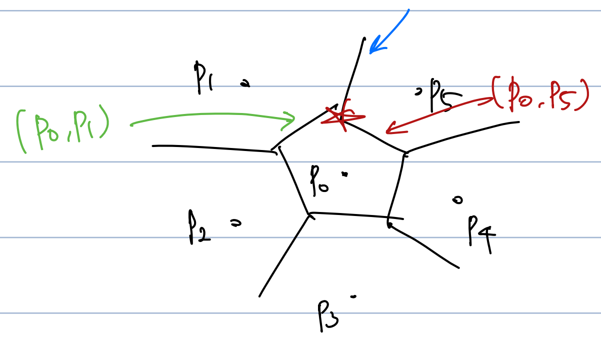
And use Euler's formula. $(v-e+f=2)$

4. Characterization of edges / vertices. (Thm 7.4)

<u>Def</u> $C_P(q)$:= the largest circle with q as a center that does not contain any site of P in its interior.



p_1, p_5로부터 거리 같은 점 모임 (bisector)



$*$: p_0, p_1, p_5로부터 거리 같은 점 : Vertex

Q. 왜 p_1과 p_4의 bisector는 edge 만들지 않는가?

A. 그 bisector는 p_0랑 가까워서.

(p_0가 원 안에 들어와서)

(1) q is a vertex of Vor(P) iff $C_P(q)$ contains three or more sites on its boundary.

q는 세 개 이상의 점으로부터 거리가 같은만큼 떨어져 있다.

(2) The bisector between sites p_i and p_j defines an edge of Vor(P)

iff $\exists q \in$ bisector s.t. $C_P(q)$ contains both p_i & p_j on its boundary but no other side.

<2. Compute the Voronoi Diagram>

Obvious bound, $O(n \cdot n\log n)$: 각 cell 마다, half plane 계산 $(n\log n)$

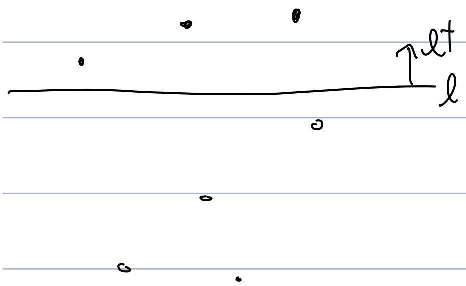
Goal : $O(n\log n)$: Fortune's Algorithm.

사실 이게 최선. ∵ the problem of sorting n real numbers.

#1. Overall Strategy.

① Sweep a horizontal line from top to bottom over the plane.

② Which informations do we have to collect while sweeping?



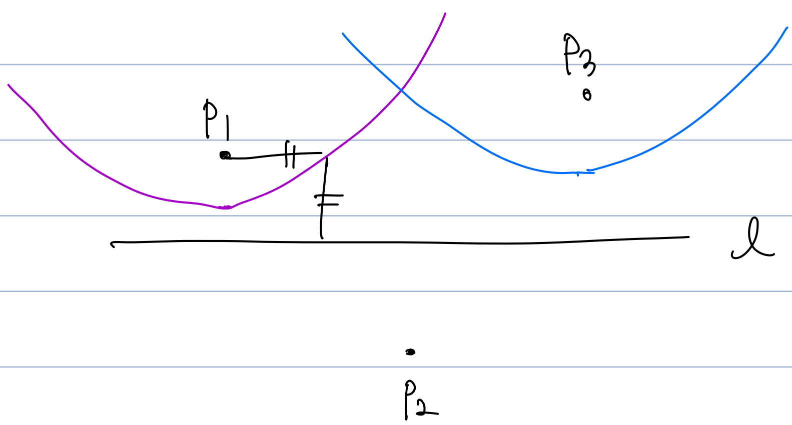
The structure of $V(p_i)$ depends on every point (even below ℓ).
So it's important to collect informations which are independent from sites below ℓ.

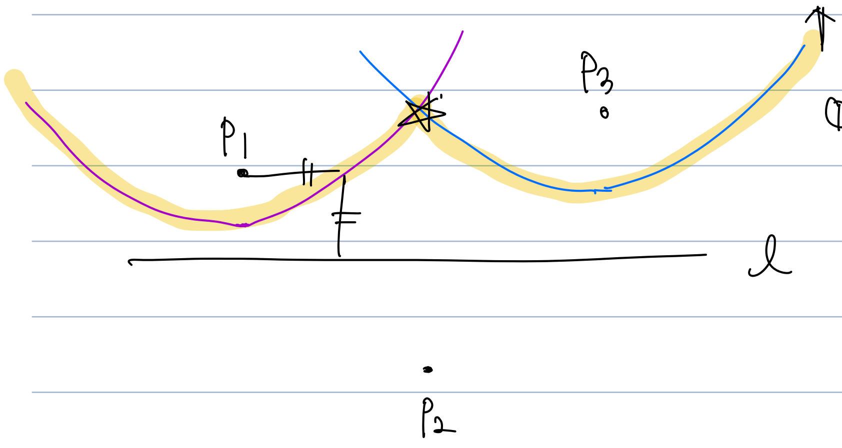
$\Leftrightarrow$ Which $q \in \ell^+$ have its nearest site above ℓ for sure?

Ans $d(q, p_1) <$ distance from q to ℓ. (∵ $< d(q, p_2)$ below ℓ)

How can we visualize this?

When $d(q, p_1) =$ distance from q to ℓ, the trace of q makes parabola.





① The points above this yellow line are independent from the informations below l.
↳ "<u>the beach line</u>" is the sequence of parabolic arcs.
(Obs 7.5 : The beach line is x-monotone.)

② ✱ is called the "<u>breakpoints</u>".
$d(p_1, ✱) =$ distance from ✱ ~ l $= d(p_3, ✱)$ i.e. ✱ $\in$ bisector between p_1 and p_3.

$\Rightarrow$ the breakpoints exactly trace out the Voronoi diagram while the sweep line moves.

✱를 기준으로 p_1에 가까운 쪽은 $\mathcal{V}(p_1)$에, p_3에 가까운 쪽은 $\mathcal{V}(p_3)$에.

<u>recall</u>

(1) q is a vertex of $Vor(P)$ iff $C_P(q)$ contains three or more sites on its boundary.
q는 세 개 이상의 점으로부터 거리가 같은만큼 떨어져 있다.

(2) The bisector between sites p_i and p_j defines an edge of $Vor(P)$
iff $\exists q \in$ bisector s.t. $C_P(q)$ contains both p_i & p_j on its boundary but no other site.

Youtube : Sweep line algorithm — Voronoi tesselation , Kevin Schaal

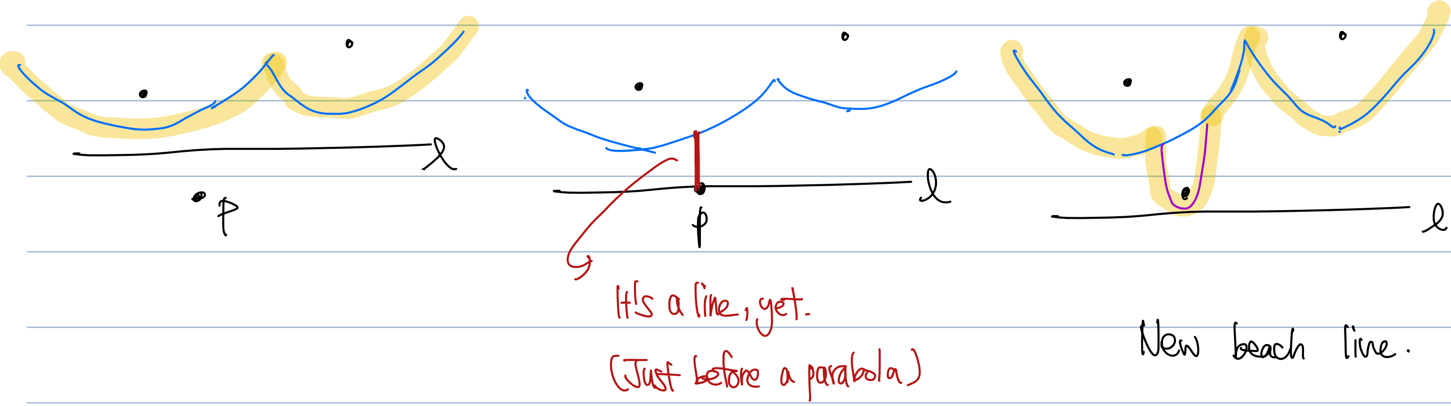
#2. Event points

We cannot simulate the movement of the beach line continuously.
So we have to figure out when "something" happens on the beach line.

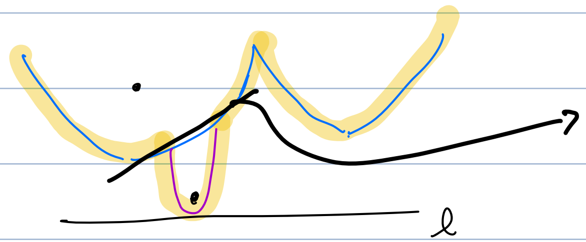
There are two types of "something". [When new arc appears. / When existing arc disappears.

Type1 : When new arc appears. [site event]

(lem 7.6) The only way in which a new arc can appear on the beach line is when a new site is encountered. We call the event "a site event".



result



New edge generates.

Observe that : the left arc and the right arc of the new arc come from the same site.

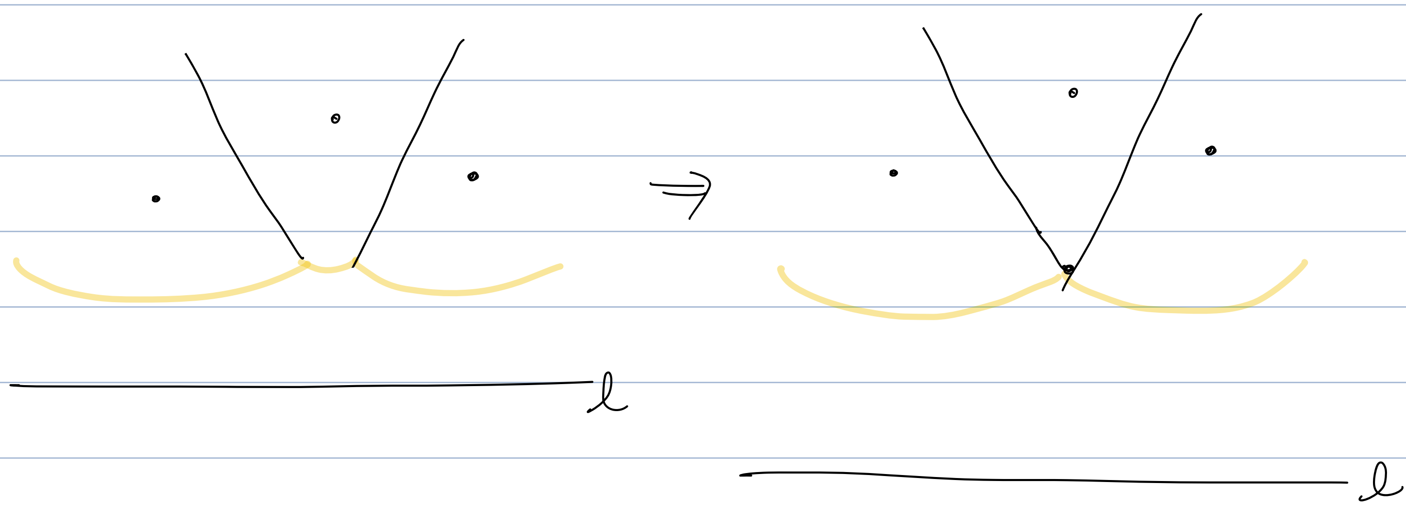
Coro The beach line consists of $\leq 2n-1$ parabolic arcs.

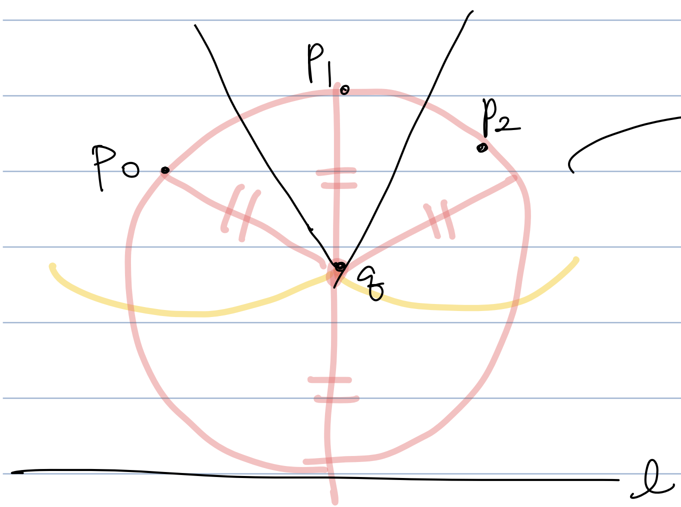
$\because$ Each site encountered give rise to one new arc and the splitting of at most one existing arc into two

Type2 When existing arc disappears. [Circle events]

(lem 7.7) The only way in which an existing arc can disappear from the beach line is through a circle event.

result It represents when two existing edges merge.



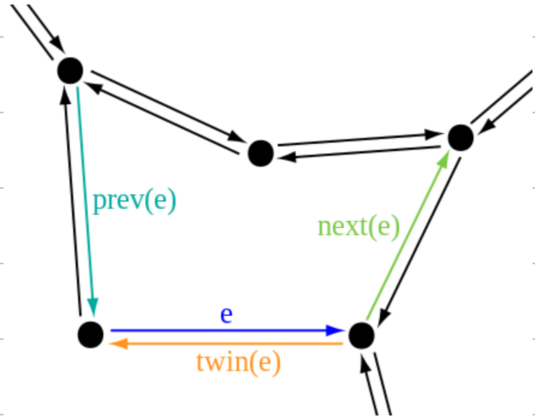


"Circle event" happens exactly when q is the center of a circle which passes through p_0. p_1. p_2. and whose lowest point lies on ℓ.

#3. Store the information

1) Voronoi Diagram : Doubly-connected edge list.

Our usual data structure for subdivisions. Each edge is divided into two directional edges.



2) Beach Line — balanced binary search tree T.

3) Event Queue — Priority queue.

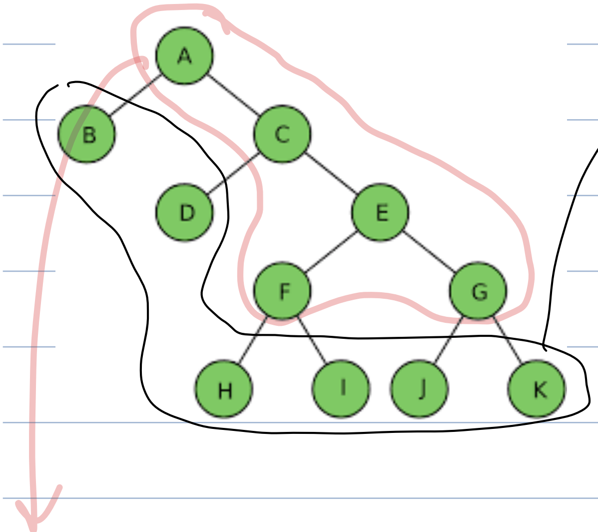
1. Each element has an associated priority.
2. Elements with high probability are served before elements with low priority
3. If two elements have the same priority, they are served in the same order in which they were enqueued.

Goal

Event Queue에 정확히 site event와 circle event를 잘 넣을 수 있겠는가?
(Beach line을 나타내는 tree T 의 도움을 받아서.)

In order to manage the information of the beach line, we cannot store the whole equation of parabolic arcs, but the "sites". More specifically, we store the information as follows.

Recall : Our beach line is X-monotone.



Each leaf

- represents each arc (left to right)
- stores each corresponded site
- points the circle event (in event queue) *
  where the arc disappears.

If [it will not disappear / corresponded circle event hasn't been detected yet] → "nil"

Each internal node.

- represents : the breakpoint between its left subtree's rightmost site and right subtree's leftmost site
- stores : tuple of sites. $\langle p_i, p_j \rangle$
- ~~points : one of the half-edge of the edge being traced out by the breakpoint.~~

In the event queue Q,

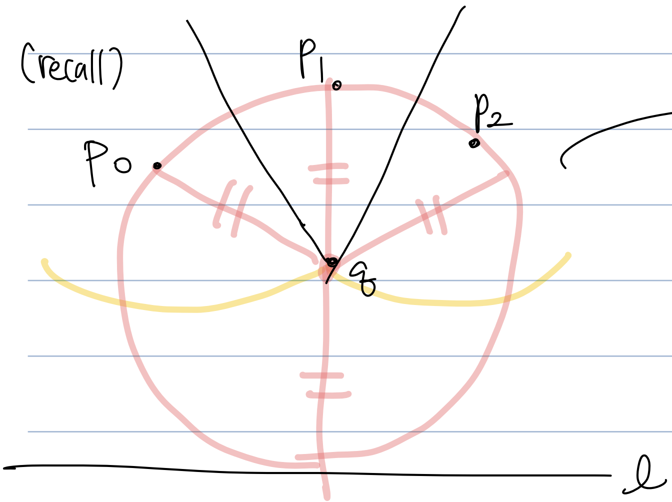
- priority : y-coordinate (위에서부터 아래로 훑는것)
- stores : the upcoming events that are already known.
  - site event : its coordinate
  - circle event : lowest point of the circle (tangent to the sweep line)
    w/ pointer : the arc which will disappear. *

All the site events are known in advance, but circle events are not.

One final issue : detection of circle events.

#4. Detection of circle Events.

(recall)

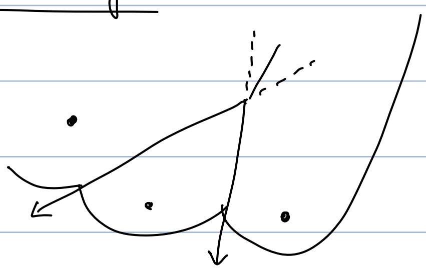


"Circle event" happens exactly when
q is the center of a circle
which passes through p_0, p_1, p_2. ⇒ Three consequtive arcs.
and whose lowest point lies on ℓ.

[Strategy] Check EVERY new triples of consequtive arcs whenever an event occurs.
We can easily eliminate some obvious not-circle-events.

There are two types of three consequtive arcs. (Assume the noncollinear case.)

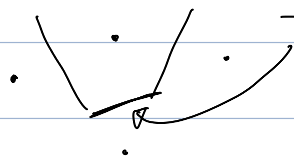
Type. Diverge

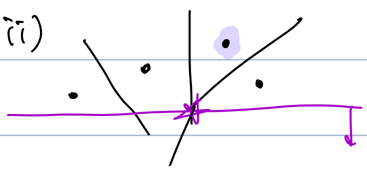


Two edges made by these three consequtive arcs diverge.
That is, they will not meet in the future.
⇒ eliminate the divergence case.

Type. Converge

Put every diverge triples of consequtive arcs in Q.
However, not every convergence case is a circle event, too.
There is a "FALSE ALARM".

i)  Two converge edges are blocked by a new edge made by site events.

ii)  (made by circle events).

*가 닫혀버린 이후에는, ●가 포함되었던 triple은 볼 필요가 없어짐

Lem 7.8 Every Voronoi vertex is detected by means of a circle event.

#5. Algorithm except the Voronoi diagram.

① Algorithm VORONOIDIAGRAM(P)

Input: $P = \{p_1, \dots, p_n\}$. a set of sites in the plane.

Initialize: the event queue Q with all site events.

the balanced search tree $\tau = \emptyset$.

While Q not empty

Do Remove the event with largest y-coordinate from Q.

If the event is a site event at p_i,

Then HANDLESITEEVENT(p_i)

Else (i.e. The event is a circle event.

It points the arc γ by site p_i that will disappear.)

HANDLESITEEVENT(γ)

② HANDLESITEEVENT(p_i)

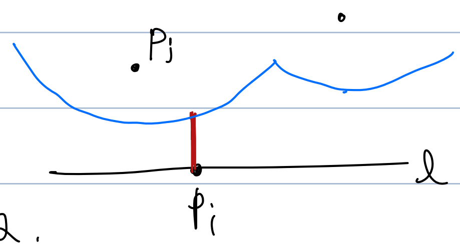
1. If $\tau = \emptyset$ → insert p_i. Otherwise, compute step 2-4.

2. Search in T for the arc by p_j vertically above p_i.

Delete a false alarm in Q.

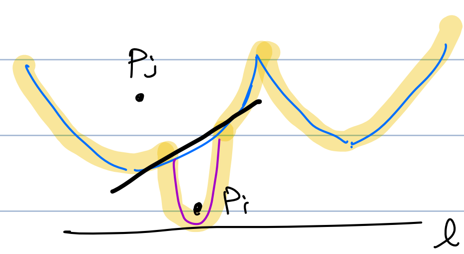
If the arc by p_j has a pointer to a circle event in Q,

then it is a false alarm-type converge-(i). ⇒ Delete from Q.



3. Replace the leaf p_j in τ with a subtree having three leaves.

$p_j \Rightarrow$ p_j p_i p_j



Modify the tuples stored in the internal nodes, as well.

Perform rebalancing operations on τ if necessary

4. Add new converge cases in Q.

New triples : (?, p_j, p_i) , (p_j, p_i, p_j), (p_i, p_j, ?)

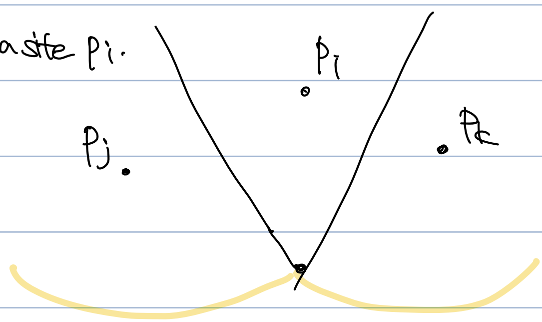
⇓

~~It's not a circle event.~~

Because the left site and the right site is the same.

If they converge, then insert this triple into Q and add the pointers

② HANDLECIRCLEEVENT(γ) Say that the arc γ is made by a site p_i.



1. Delete the leaf p_i in $\mathcal{T}$.

Update the internal nodes.

Perform rebalancing.

2. Delete a false alarm in Q.

: false alarm. type converge. (ii). delete all circle events involving p_i from Q.

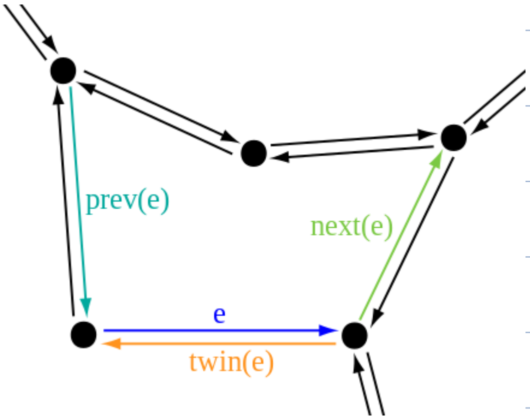
3. Add new converge cases in Q.

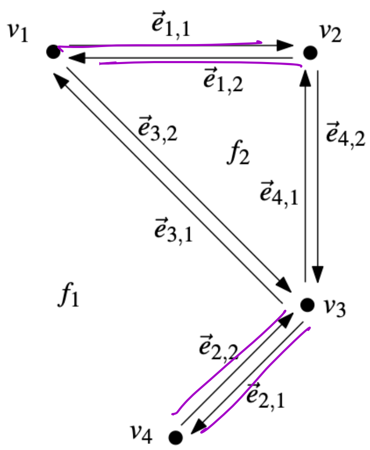
Check (?, p_j, p_k) and (p_j, p_k, ?).

If it is a converge case, insert into Q. Set pointers between Q and $\mathcal{T}$.

#6. Algorithm with the Voronoi diagram.

* Voronoi Diagram : Doubly-connected edge list.





data　　　　pointer

Vertex	Coordinates	IncidentEdge
v_1	$(0,4)$	$\vec{e}_{1,1}$
v_2	$(2,4)$	$\vec{e}_{4,2}$
v_3	$(2,2)$	$\vec{e}_{2,1}$
v_4	$(1,1)$	$\vec{e}_{2,2}$

; vertex에서 출발하는 edge 아무거나 저장

outer boundary의 some half edge.

Face	OuterComponent	InnerComponents
f_1	nil unbdd face	$\vec{e}_{1,1}$
f_2	$\vec{e}_{4,1}$	nil no inner hole.

출발점　반대　edge의 왼쪽 face　→ 같은 face에서 전/이후.

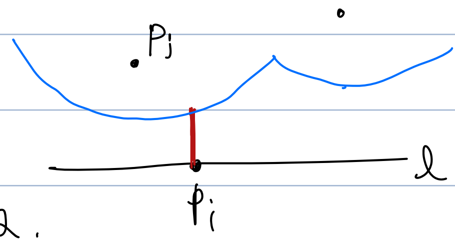
Half-edge	Origin	Twin	IncidentFace	Next	Prev
$\vec{e}_{1,1}$	v_1	$\vec{e}_{1,2}$	f_1	$\vec{e}_{4,2}$	$\vec{e}_{3,1}$
$\vec{e}_{1,2}$	v_2	$\vec{e}_{1,1}$	f_2	$\vec{e}_{3,2}$	$\vec{e}_{4,1}$
$\vec{e}_{2,1}$	v_3	$\vec{e}_{2,2}$	f_1	$\vec{e}_{2,2}$	$\vec{e}_{4,2}$
$\vec{e}_{2,2}$	v_4	$\vec{e}_{2,1}$	f_1	$\vec{e}_{3,1}$	$\vec{e}_{2,1}$
$\vec{e}_{3,1}$	v_3	$\vec{e}_{3,2}$	f_1	$\vec{e}_{1,1}$	$\vec{e}_{2,2}$
$\vec{e}_{3,2}$	v_1	$\vec{e}_{3,1}$	f_2	$\vec{e}_{4,1}$	$\vec{e}_{1,2}$
$\vec{e}_{4,1}$	v_3	$\vec{e}_{4,2}$	f_2	$\vec{e}_{1,2}$	$\vec{e}_{3,2}$
$\vec{e}_{4,2}$	v_2	$\vec{e}_{4,1}$	f_1	$\vec{e}_{2,1}$	$\vec{e}_{1,1}$

//

② HANDLESITEEVENT(p_i)

1. If $\mathcal{T} = \emptyset$ → insert p_i. Otherwise, compute step 2–4.

2. Search in $\mathcal{T}$ for the arc by p_j vertically above p_i.

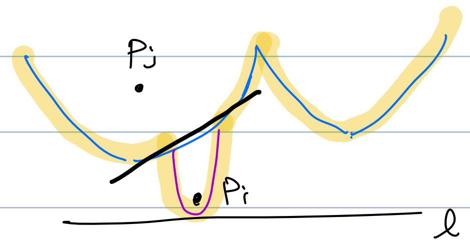


Delete a false alarm in Q.

If the arc by p_j has a pointer to a circle event in Q, then it is a false alarm-type converge-(i). ⇒ Delete from Q.

3. Replace the leaf p_j in $\mathcal{T}$ with a subtree having three leaves.

$p_j \Rightarrow p_j \quad p_i \quad p_j$

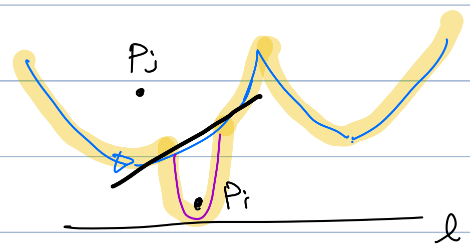


Modify the tuples stored in the internal nodes, as well.

Perform rebalancing operations on $\mathcal{T}$ if necessary

4. (In Voronoi diagram)

- Create new half-edge records on DCEL for the edge separating $V(p_i)$ & $V(p_j)$ and pointer $\langle p_i, p_j \rangle \in \mathcal{T}$.



5. Add new converge cases in Q.

New triples : $(?, p_j, p_i)$, (p_j, p_i, p_j), $(p_i, p_j, ?)$

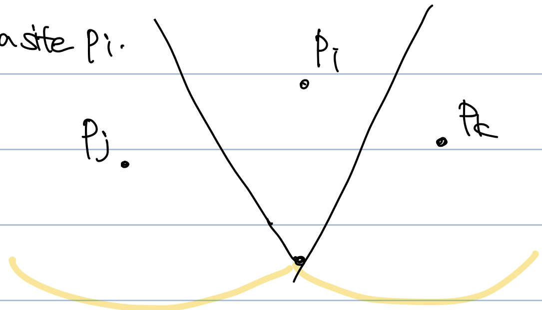
⇓

It's not a circle event.

Because the left site and the right site is the same.

If they converge, then insert this triple into Q and add the pointers

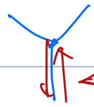
⑦ HANDLECIRCLEEVENT(γ)   Say that the arc γ is made by a site p_i.



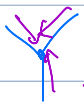
1. Delete the leaf p_i in $\mathcal{T}$.
   Update the internal nodes.
   Perform rebalancing.

2. Delete a false alarm in Q.
   : false alarm, type converge. (ii). delete all circle events involving p_i from Q.

3. (In Voronoi Diagram)
   - Add the center of the circle causing the event as a vertex record to DCEL.



   - Create two half-edge records corr. to the new break point of the beach line.
   - Set the pointers btwn them properly.



   - Attach the three new records to the half-edge records that end at the vertex.

4. Add new converge cases in Q.
   Check (?, p_j, p_k) and (p_j, p_k, ?).
   If it is a converge case, insert into Q. Set pointers between Q and $\mathcal{T}$.

① Algorithm VORONOIDIAGRAM(P)

Input: $P = \{p_1, \dots, p_n\}$. a set of sites in the plane.

Output: Vor(P). given inside a bounding box in a DCEL.

Initialize: the event queue Q with all site events.

the balanced search tree $\mathcal{T} = \emptyset$.

While Q not empty

Do Remove the event with largest y-coordinate from Q.

If the event is a site event at p_i,

Then HANDLESITEEVENT(p_i)

Else (i.e. The event is a circle event.

It points the arc γ by site p_i that will disappear.)

HANDLESITEEVENT(γ)

- Left internal nodes in $\mathcal{T}$: half-infinite edges.

  So, compute a bounding box that contains all vertices of the Voronoi diagram in its interior.

- Attach the half-infinite edges to the bounding box

  by updating the DCEL properly.

- (Output) Traverse the half-edges of the DCEL to add the cell records

  and the pointers to and from them.

  (face 개념 붙이기).

– Lemma 7.9

The algorithm runs in $O(n \log n)$ time. and it uses $O(n)$ storage.

pf) operations on $\mathcal{T}$ and Q : $O(\log n)$

" DCEL : $O(1)$

site event : n개, circle event = # of Vor(P). $\leq 2n-5$

(false alarm : deleted.) $\therefore O(n \log n)$ //