Doubly Connect Edge List (DCEL)

Notes from the book by de Berg, Van Krevald, Overmars, and Schwarzkpf.

pp. 29-39

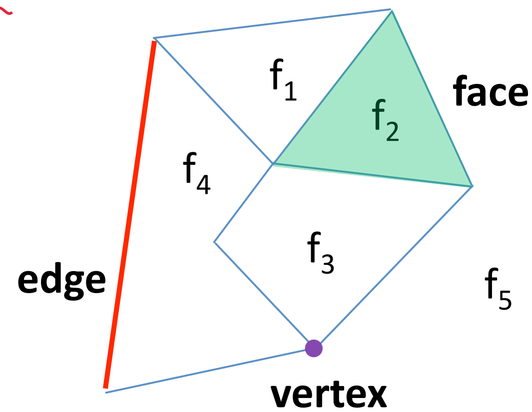
Doubly Connected Edge List (DCEL)

- DCEL is one of the most commonly used representations for planar subdivisions such as Voronoi diagrams.
- It is an edge-based structure which links together the three sets of records:
  - **Vertex**
  - **Edge**
  - **Face**

  Objects
- It facilitates traversing the faces of planar subdivision, visiting all the edges around a given vertex

Doubly Connected Edge List (DCEL)

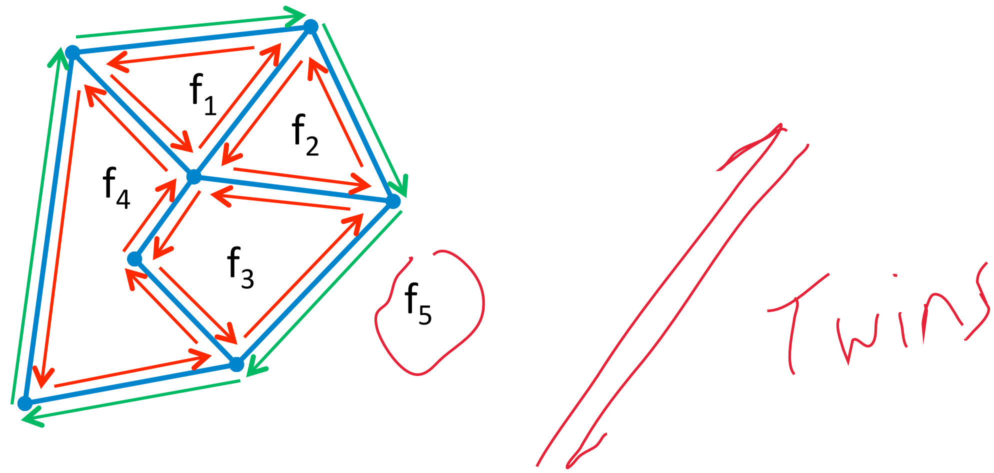
planar subdivision



~~wing-edge~~
~~quadedge~~

- Record for each face, edge, and vertex
  - Geometric information
  - Topological information
  - Attribute information
- Half-edge structure

Doubly Connected Edge List (DCEL)



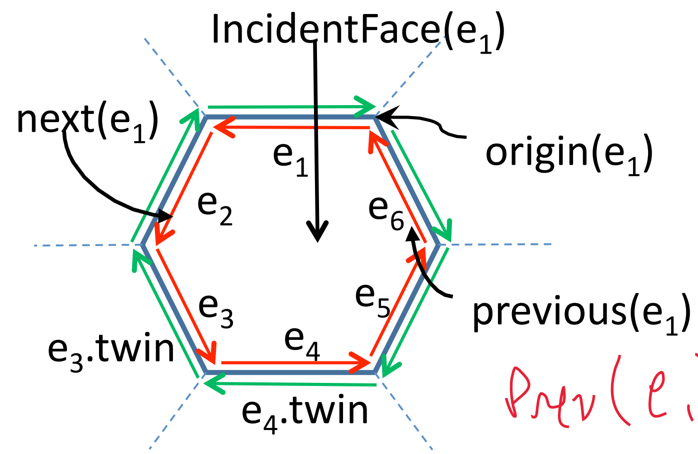
- **Main ideas:**
  - Edges are oriented counterclockwise inside each face
  - Since an edge borders two faces, each edge is replaced by two half-edges, one for each face

Doubly Connected Edge List (DCEL)

(x, y)

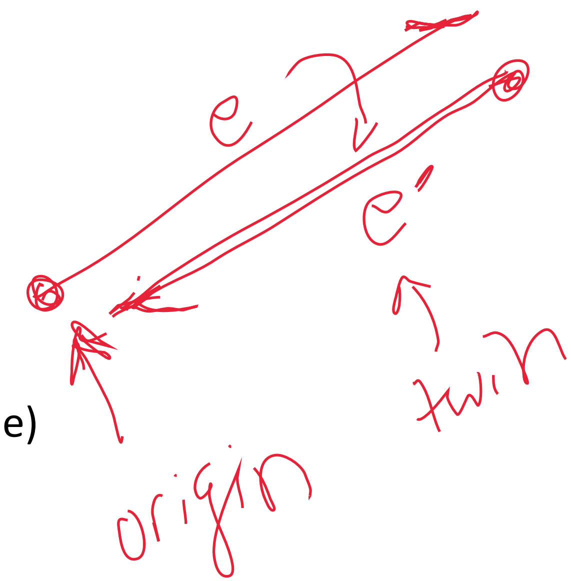
- The vertex record of a vertex v stores the coordinates of v. It also stores a pointer IncidentEdge(v) to an arbitrary half-edge that has v as its origin

- The face record of a face f stores a pointer to some half-edge on its boundary which can be used as a starting point to traverse f in counterclockwise order



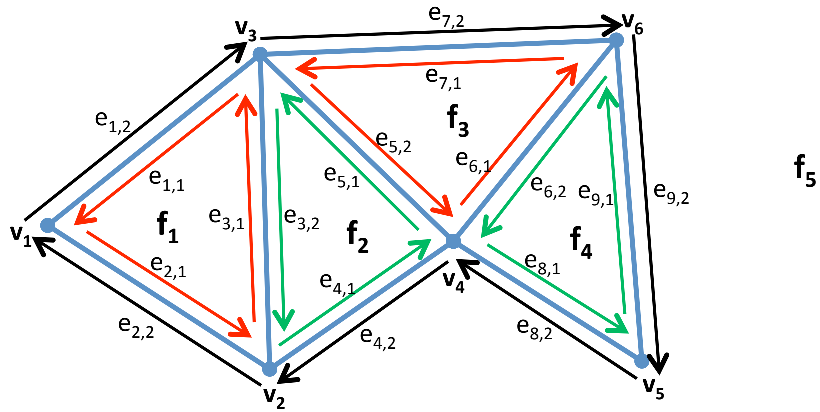
Prev(e_1)

- The half-edge record of a half-edge e stores pointer to:
  - Origin (e) ~ vertex
  - Twin of e, e.twin or twin(e)
  - The face to its left (IncidentFace(e))
  - Next(e) : next half-edge on the boundary of IncidentFace(e)
  - Previous(e) : previous half-edge



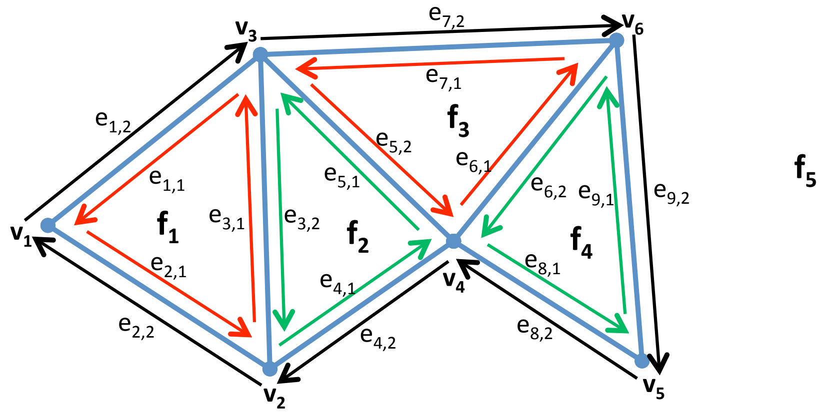
Prev(e)

Doubly Connected Edge List (DCEL)



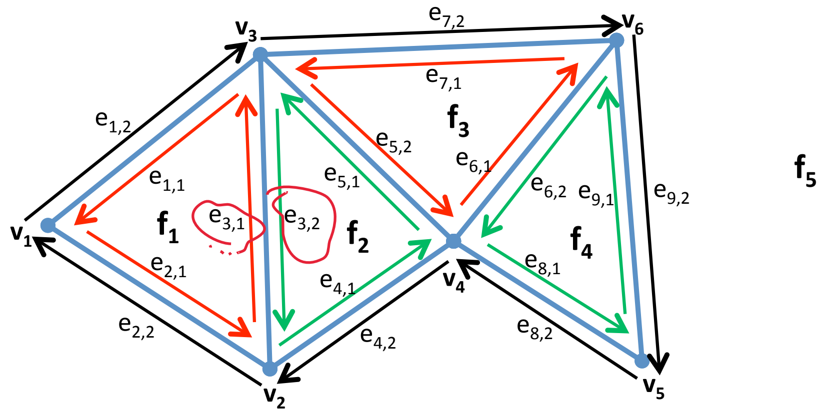
Vertex	Coordinates	IncidentEdge
v_1	(x_1, y_1)	$e_{2,1}$
v_2	(x_2, y_2)	$e_{4,1}$
v_3	(x_3, y_3)	$e_{3,2}$
v_4	(x_4, y_4)	$e_{6,1}$
v_5	(x_5, y_5)	$e_{9,1}$
v_6	(x_6, y_6)	$e_{7,1}$

Doubly Connected Edge List (DCEL)



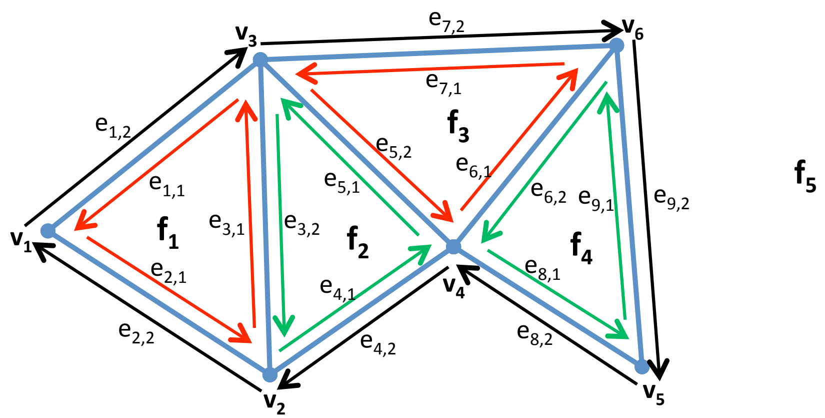
Face	Edge
f_1	$e_{1,1}$
f_2	$e_{5,1}$
f_3	$e_{5,2}$
f_4	$e_{8,1}$
f_5	$e_{9,2}$

Doubly Connected Edge List (DCEL)



Half-edge	Origin	Twin	IncidentFace	Next	Previous
$e_{3,1}$	v_2	$e_{3,2}$	f_1	$e_{1,1}$	$e_{2,1}$
$e_{3,2}$	v_3	$e_{3,1}$	f_2	$e_{4,1}$	$e_{5,1}$
$e_{4,1}$	v_2	$e_{4,2}$	f_2	$e_{5,1}$	$e_{3,2}$
$e_{4,2}$	v_4	$e_{4,1}$	f_5	$e_{2,2}$	$e_{8,2}$
…	…	…	…	…	…

Doubly Connected Edge List (DCEL)



planar graph

- **<u>Storage space requirement:</u>**
  - Linear in the number of vertices, edges, and faces

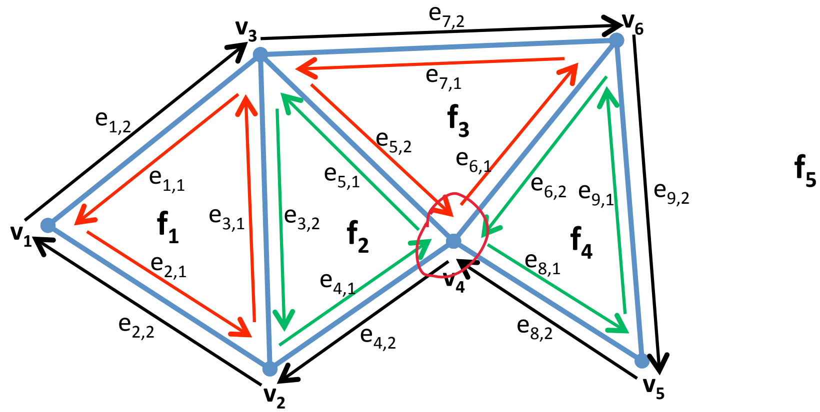
n = # vertices — no edges cross

~~e~~ m = # edges $\le 3n-6$

faces $\le 2n$

$O(n)$ space

Doubly Connected Edge List (DCEL)

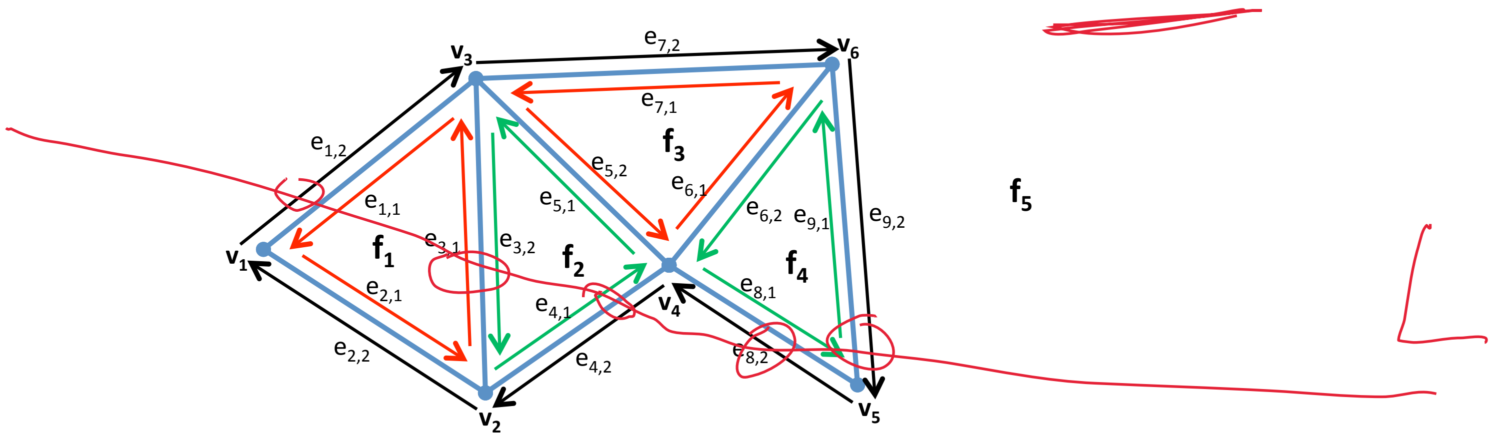


Next(Twin($e_{6,1}$))

$e_{8,1}$

- **Operations:**
  - Walk around the boundary of a given face in CCW order / CW
  - Access a face from an adjacent one edge
  - Visit all the edges around a given vertex

Doubly Connected Edge List (DCEL)



- **Interesting Queries:**
  - Given a DCEL description, a line L and a half-edge that this line cuts, efficiently find all the faces cut by L.

Doubly Connected Edge List (DCEL)

- **Traversing face f:**
  - Given: an edge of f
    1. Determine the half-edge e incident on f
    2. Start_edge ← e
    3. While next(e) ≠ start_edge then

       e ← next (e)

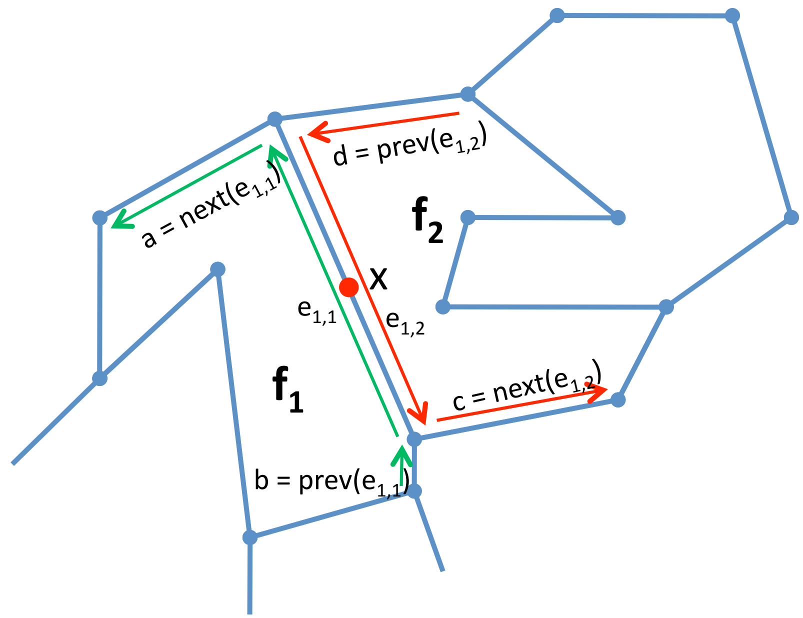
Time: size of f

Doubly Connected Edge List (DCEL)

- **<u>Traversing all edges incident on a vertex v</u>**
  - Note: we only output the half-edges whose origin is v
  - Given: a half-edge e with the origin at v
    1. Start_edge ← e
    2. While next(twin(e)) ≠ start_edge then

       e ← next(twin(e))

Adding a Vertex / edge O(1) time



Adding a Vertex

- New vertex x
- New edges: $e_{1,2}'$ and $e_{1,2}''$

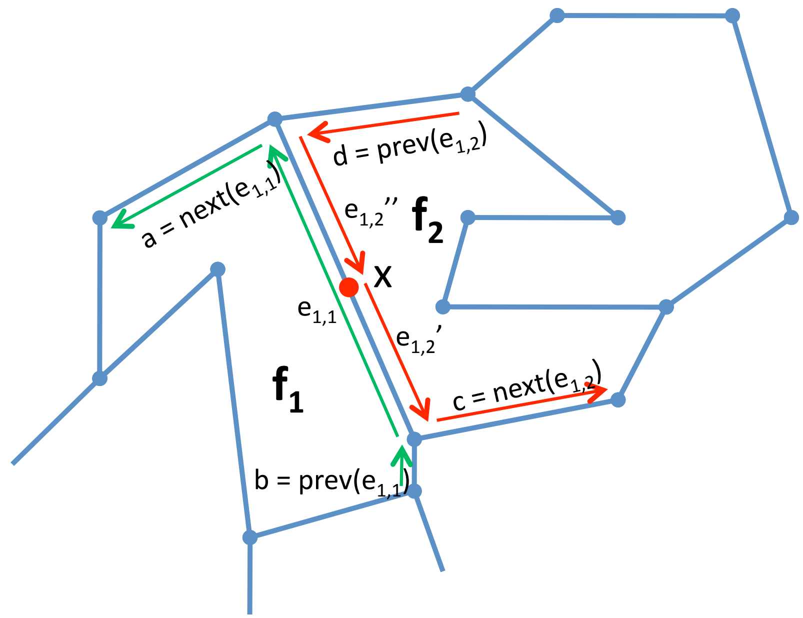
- IncidentEdge(x) = $e_{1,2}'$

- Origin($e_{1,2}'$) = x
- Next($e_{1,2}'$) = next ($e_{1,2}$)
- Prev($e_{1,2}'$) = $e_{1,2}''$
- IncidentFace($e_{1,2}'$) = f_2

- Origin($e_{1,2}''$) = origin($e_{1,2}$)
- Next($e_{1,2}''$) = $e_{1,2}'$
- Prev($e_{1,2}''$) = prev($e_{1,2}$)
- IncidentFace($e_{1,2}''$) = f_2

- Next(Prev($e_{1,2}$)) = $e_{1,2}''$
- Prev(Next($e_{1,2}$)) = $e_{1,2}'$

- Delete edge $e_{1,2}$



updating
the records

Adding a Vertex

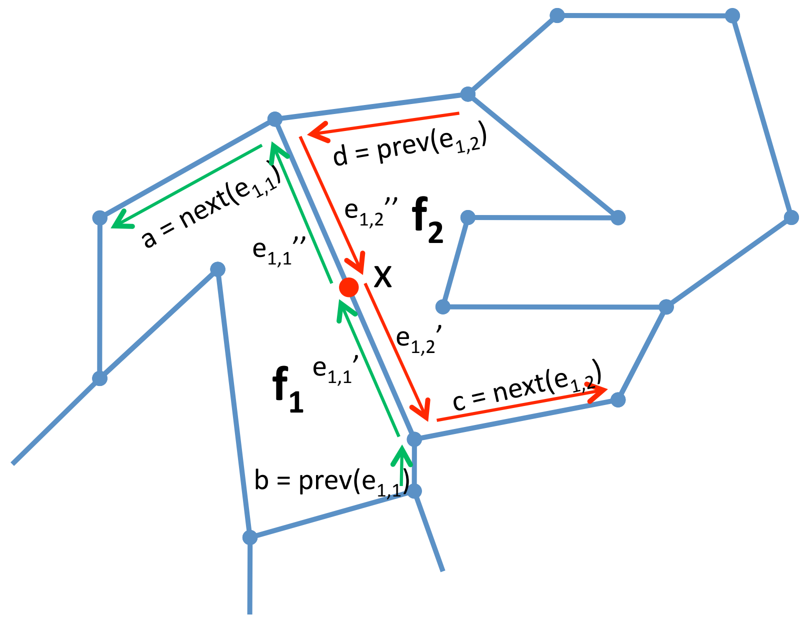
- New edges: $e_{1,1}'$ and $e_{1,1}''$

- Origin($e_{1,1}'$) = origin($e_{1,1}$)
- Next($e_{1,1}'$) = $e_{1,1}''$
- Prev($e_{1,1}'$) = prev($e_{1,1}$)
- IncidentFace($e_{1,1}'$) = f_1

- Origin($e_{1,1}''$) = $e_{1,1}'$
- Next($e_{1,1}''$) = next($e_{1,1}$)
- Prev($e_{1,1}''$) = $e_{1,1}'$
- IncidentFace($e_{1,1}''$) = f_1

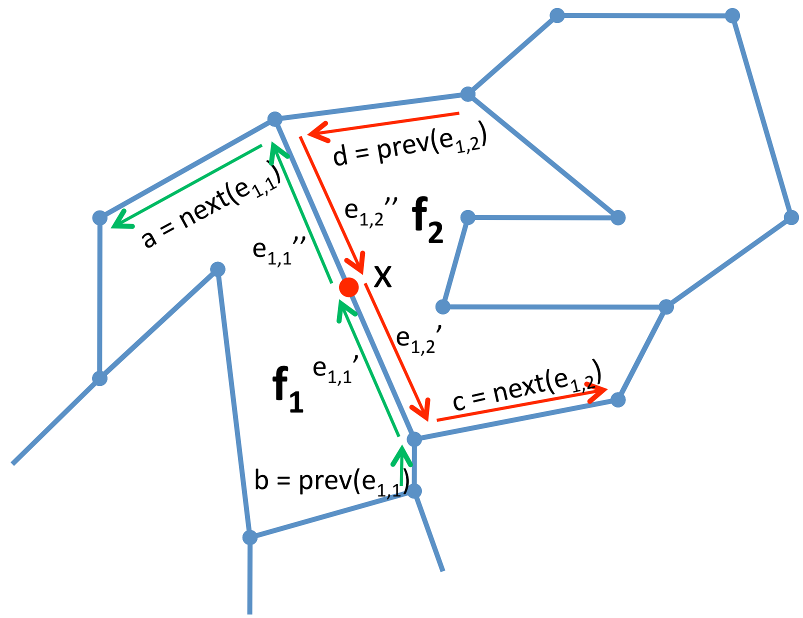
- Next(prev($e_{1,1}$)) = $e_{1,1}'$
- Prev(next($e_{1,1}$)) = $e_{1,1}''$

- Twin($e_{1,2}'$) = $e_{1,1}'$
- Twin($e_{1,1}'$) = $e_{1,2}'$
- Twin($e_{1,2}''$) = $e_{1,1}''$
- Twin($e_{1,1}''$) = $e_{1,2}''$
- Delete edge $e_{1,1}$

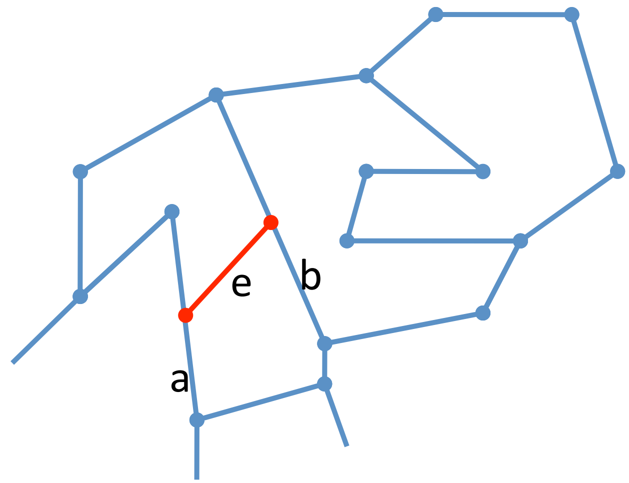


Adding a Vertex

- If $e_{1,1}$ was starting edge of f_1, need to change it to either one of the new edges
- If $e_{1,2}$ was starting edge of f_2, need to change it to either one of the new edges



Other Operations on DCEL



- **<u>Add an Edge</u>**
  - Planar subdivision
  - e is added
  - DCEL can be updated in constant time once the edges a and b are known