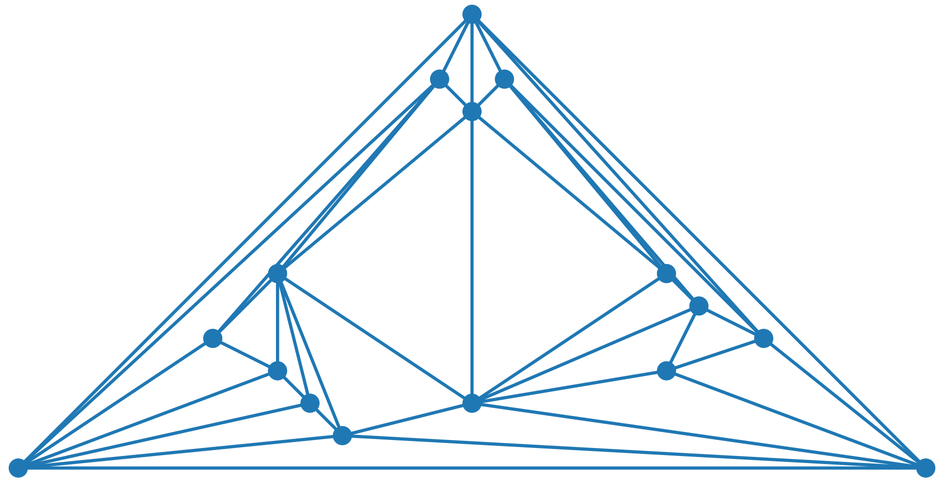
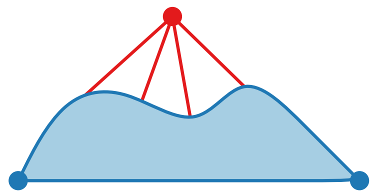
Visualization of Graphs

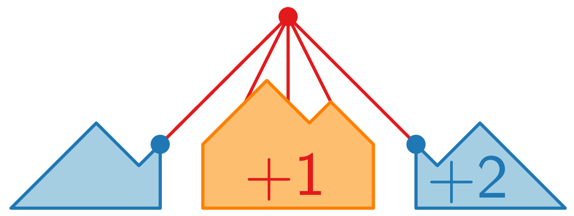
Lecture 3:
Straight-Line Drawings of Planar Graphs I: Canonical Orderings and the Shift Method



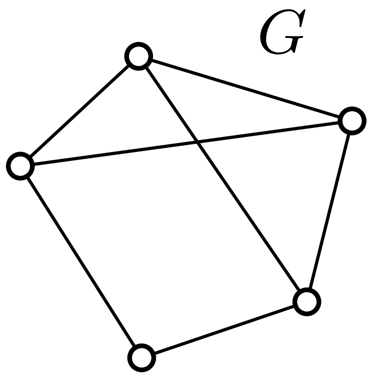
Samuel Wolf

Summer term 2025

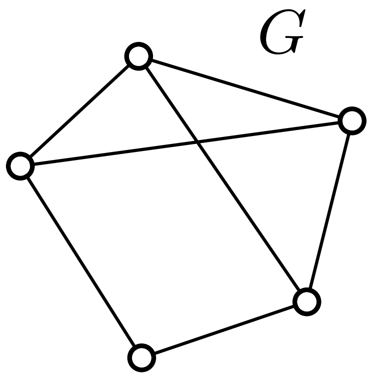


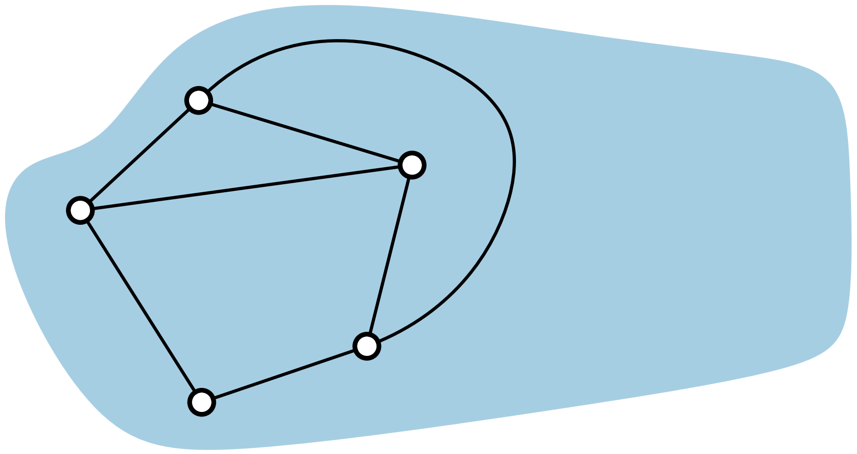


Planar Graphs

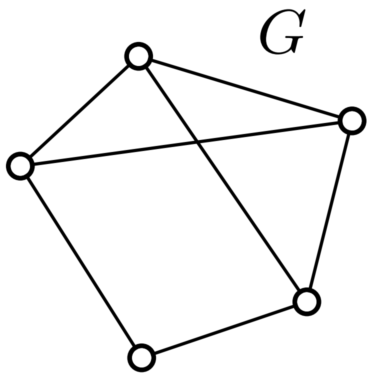


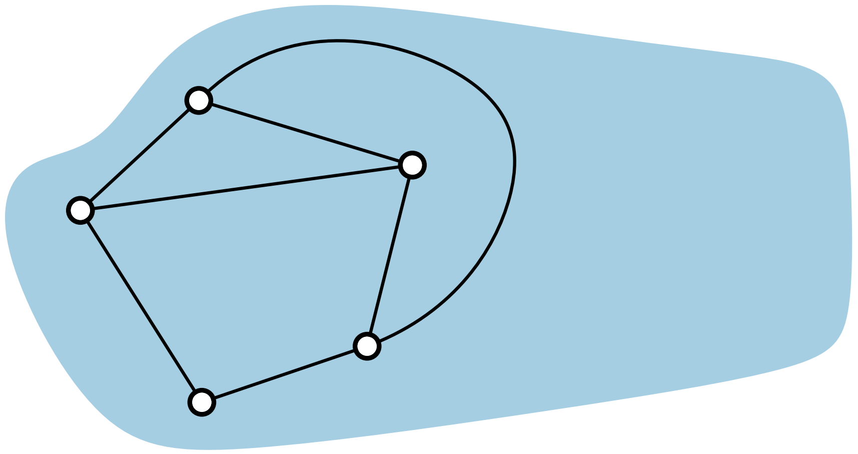
Planar Graphs





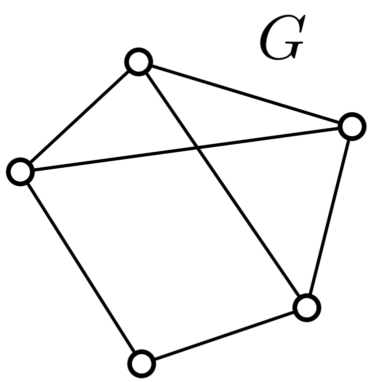
Planar Graphs

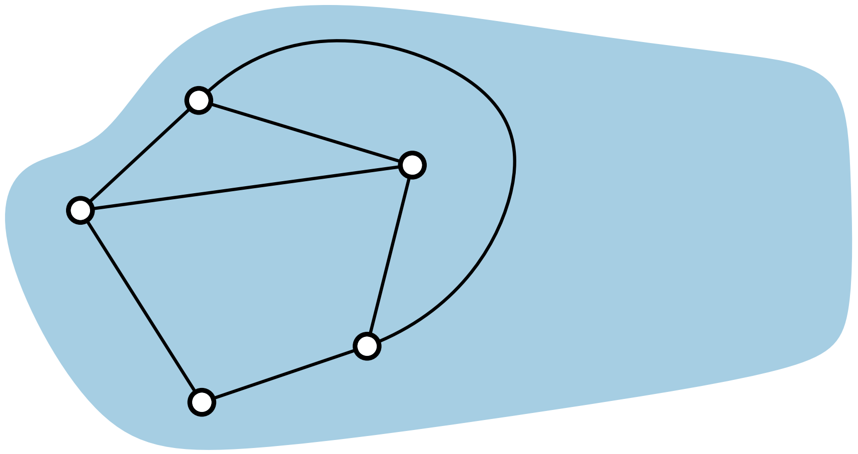




G is **planar**:
it can be drawn in such a way that
no two edges intersect each other.

Planar Graphs

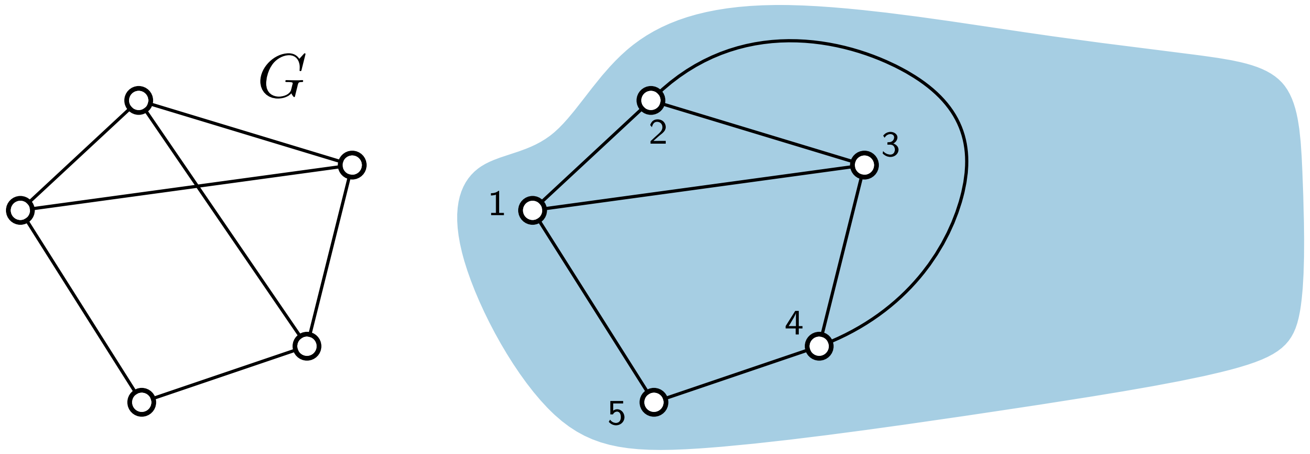




G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

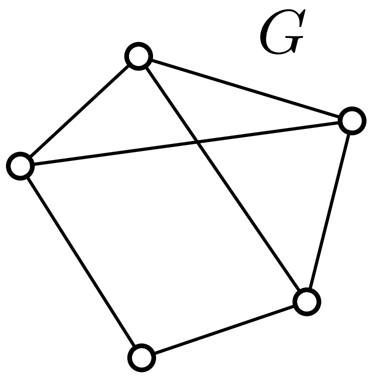
Planar Graphs

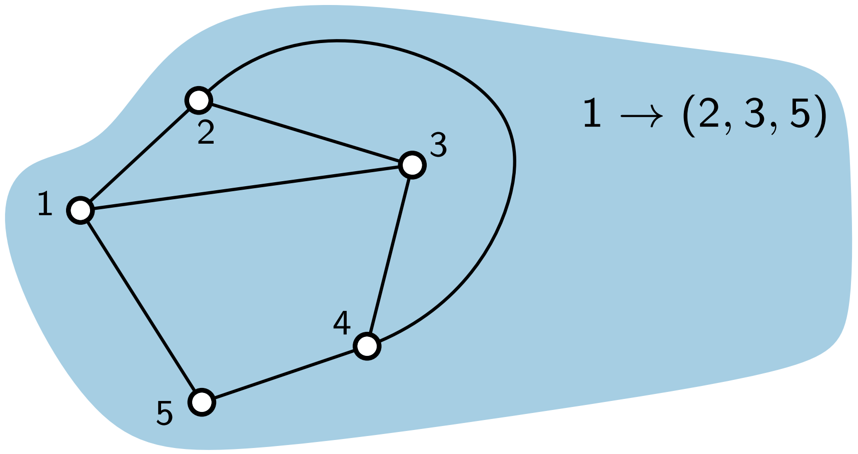


G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

Planar Graphs

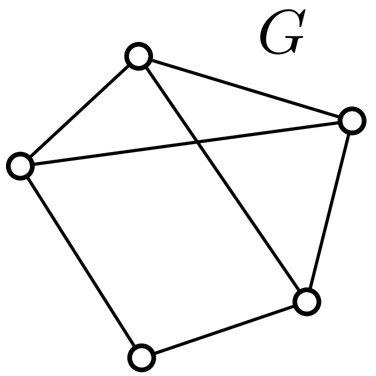


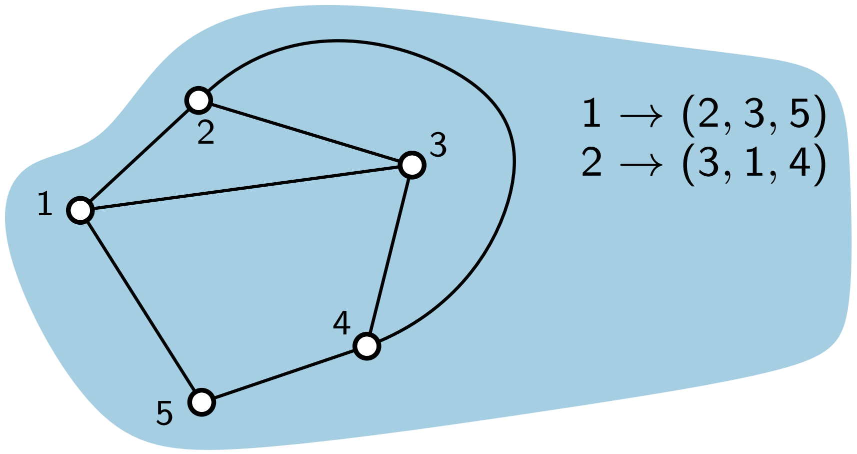


G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

Planar Graphs

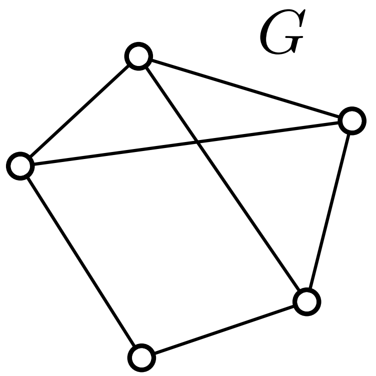


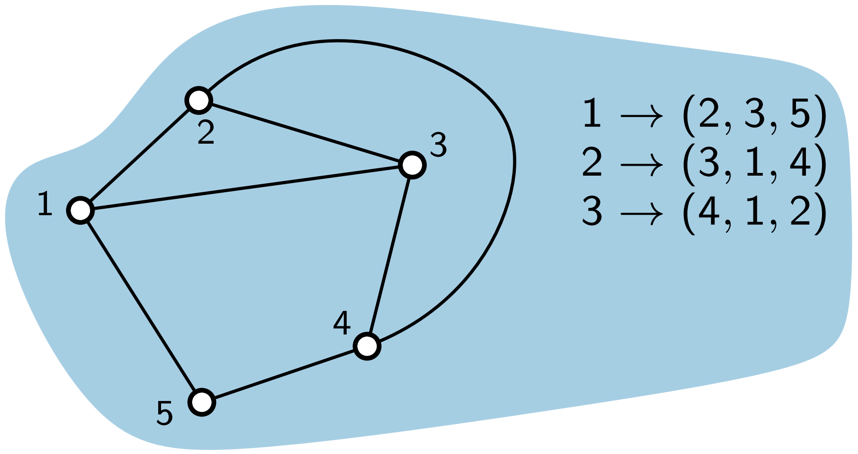


G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

Planar Graphs

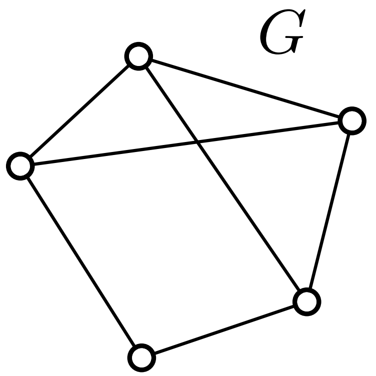


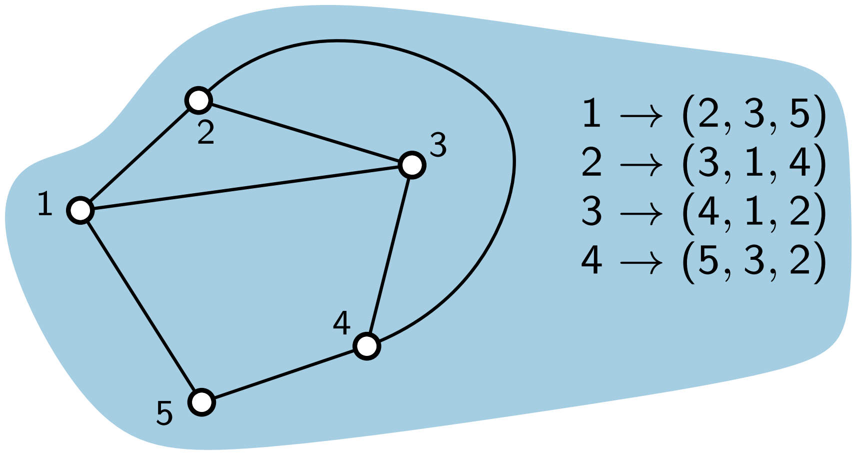


G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

Planar Graphs

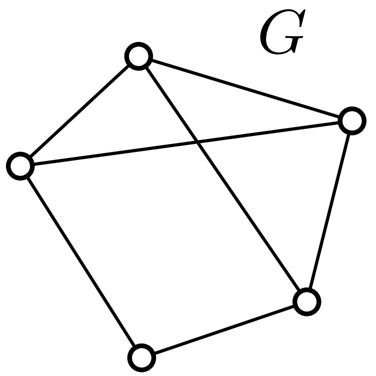


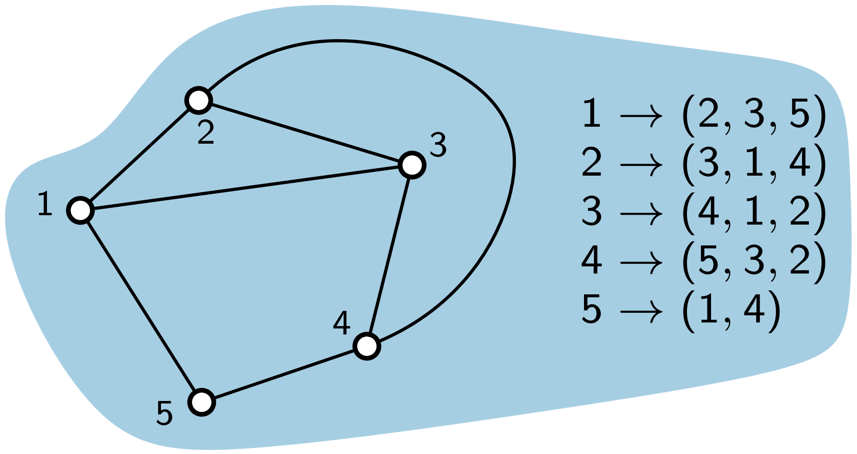


G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

Planar Graphs

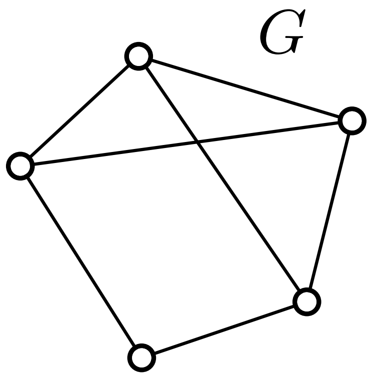


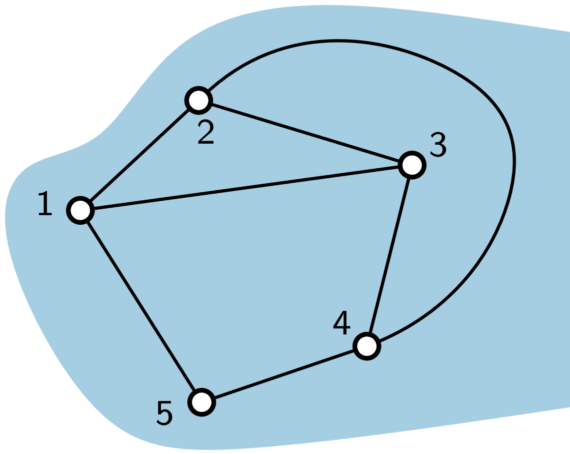


G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

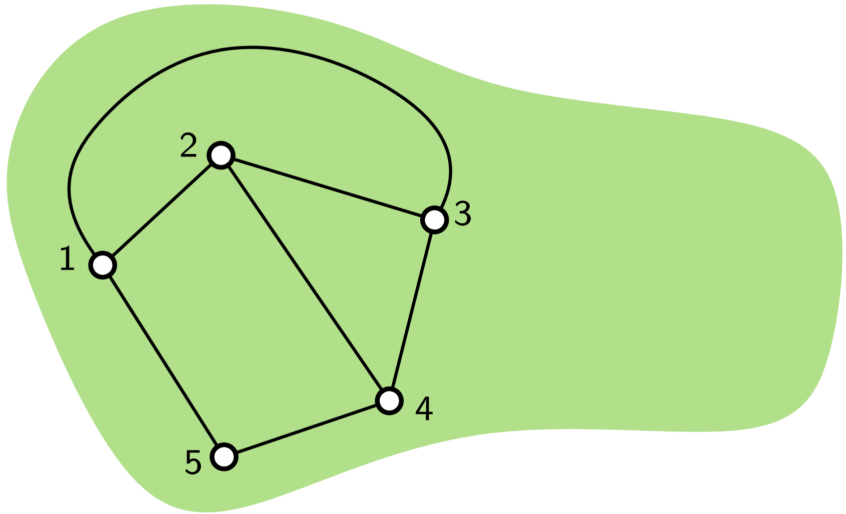
planar embedding:
clockwise orientation of adjacent vertices around each vertex

Planar Graphs





$1 \to (2, 3, 5)$
$2 \to (3, 1, 4)$
$3 \to (4, 1, 2)$
$4 \to (5, 3, 2)$
$5 \to (1, 4)$

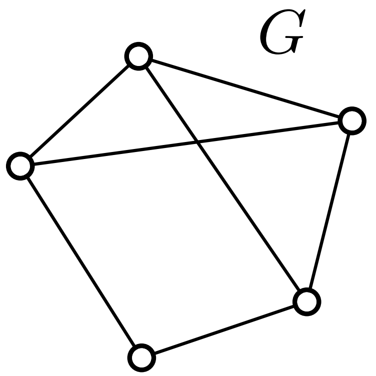


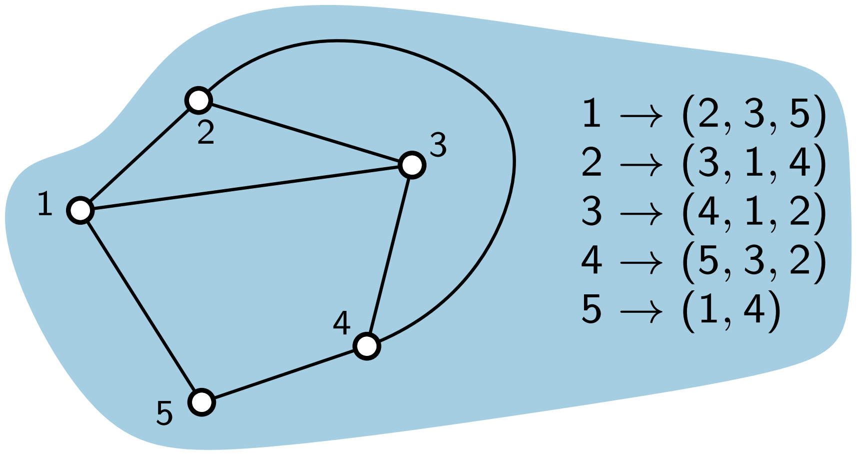
G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

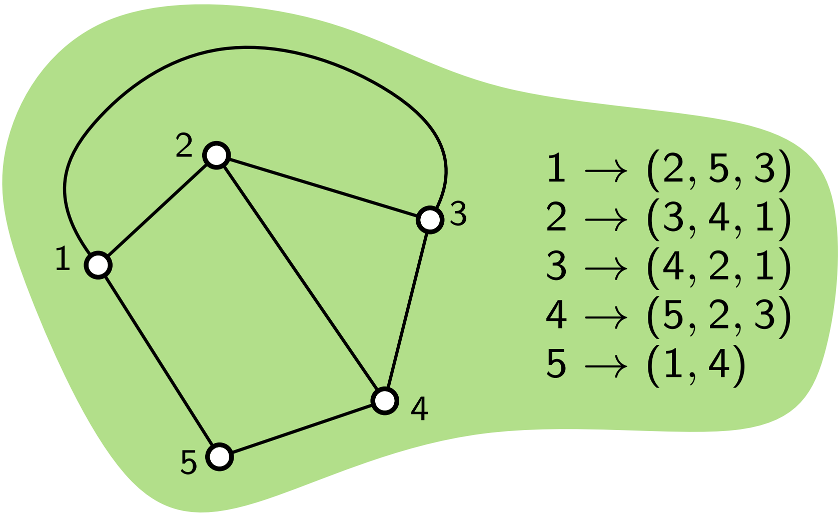
planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

Planar Graphs





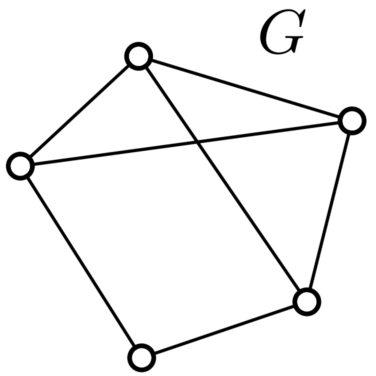


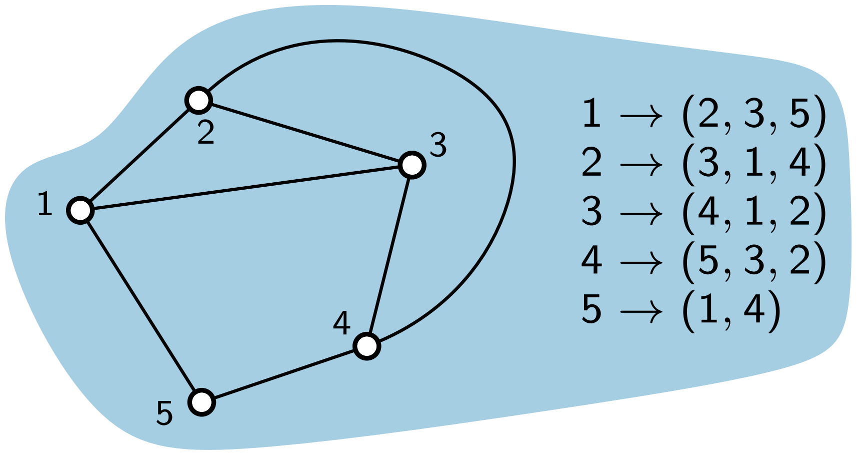
G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

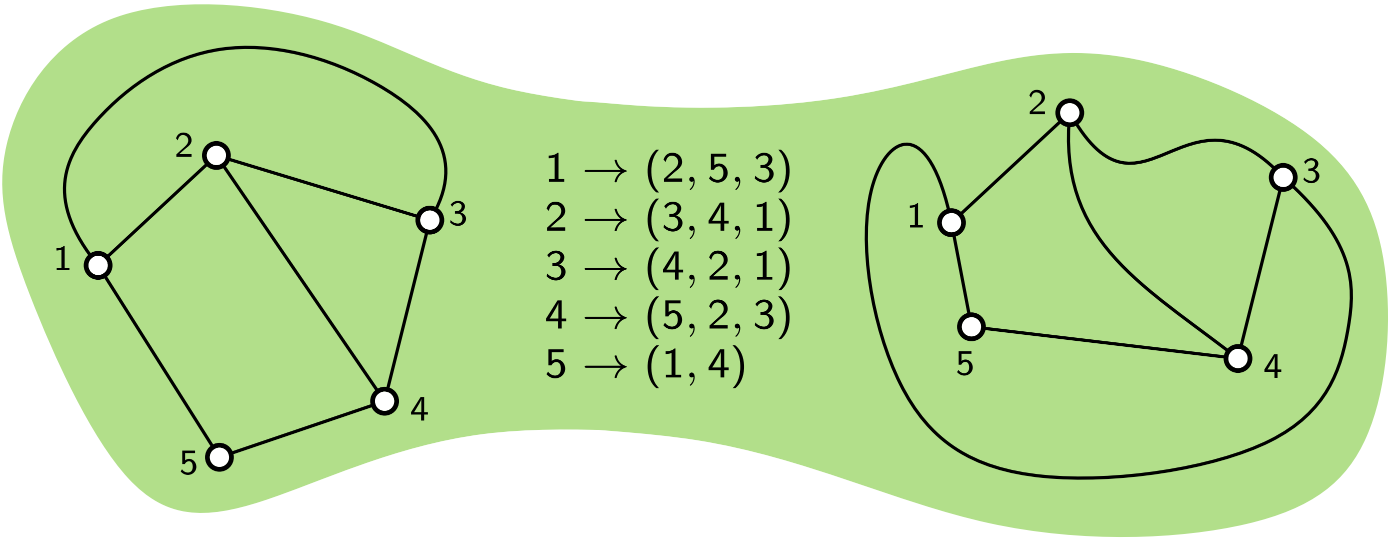
planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

Planar Graphs







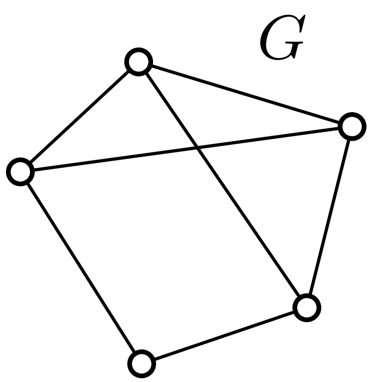
G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

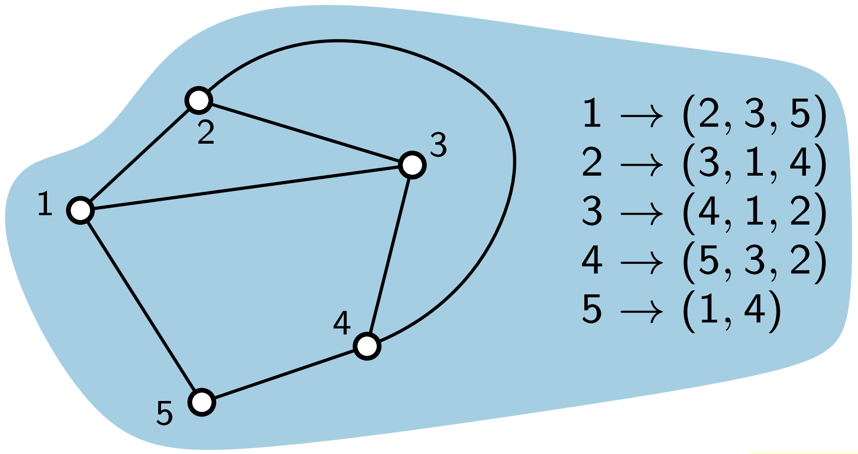
planar embedding:
clockwise orientation of adjacent vertices around each vertex

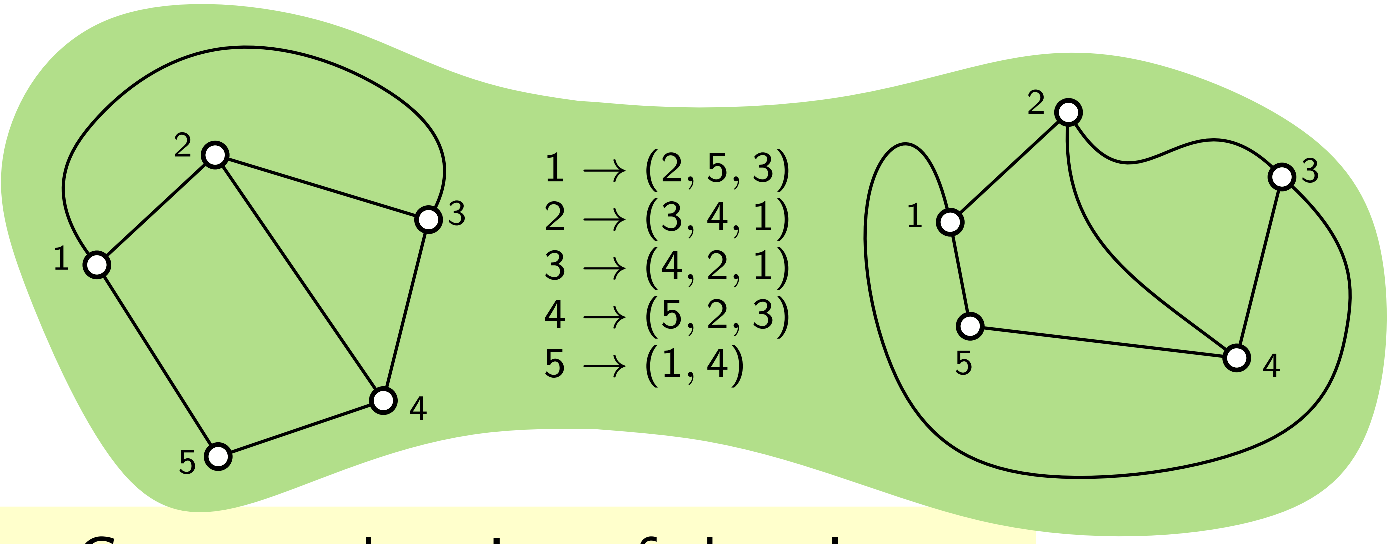
A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

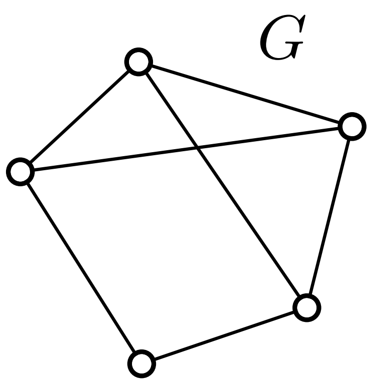
planar embedding:
clockwise orientation of adjacent vertices around each vertex

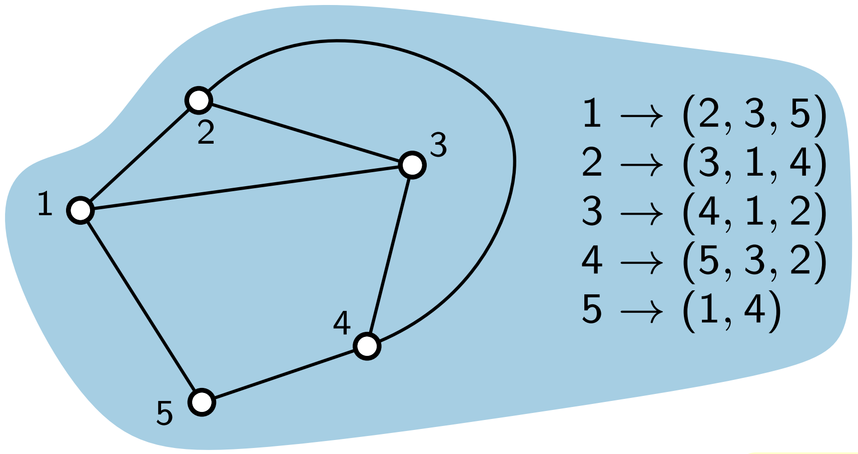
faces: Connected region of the plane bounded by edges

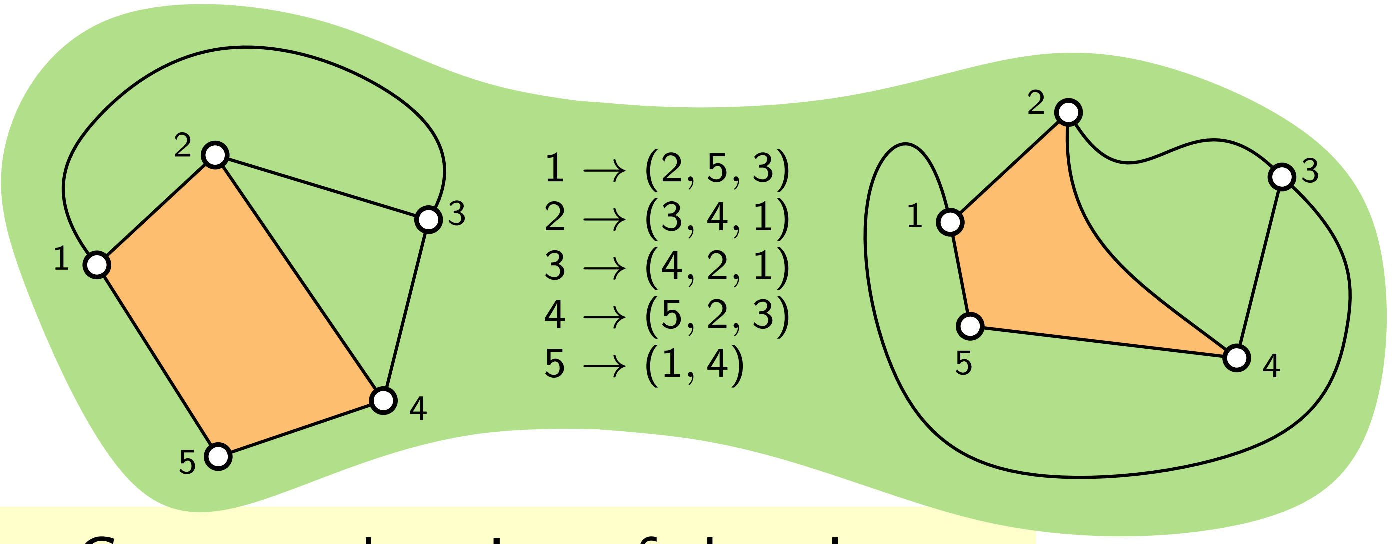
A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

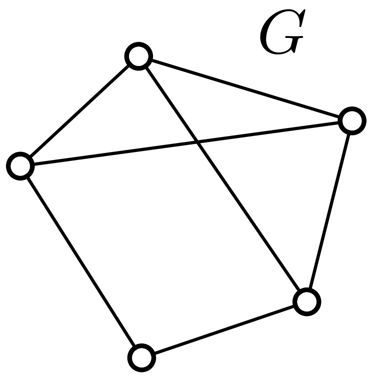
planar embedding:
clockwise orientation of adjacent vertices around each vertex

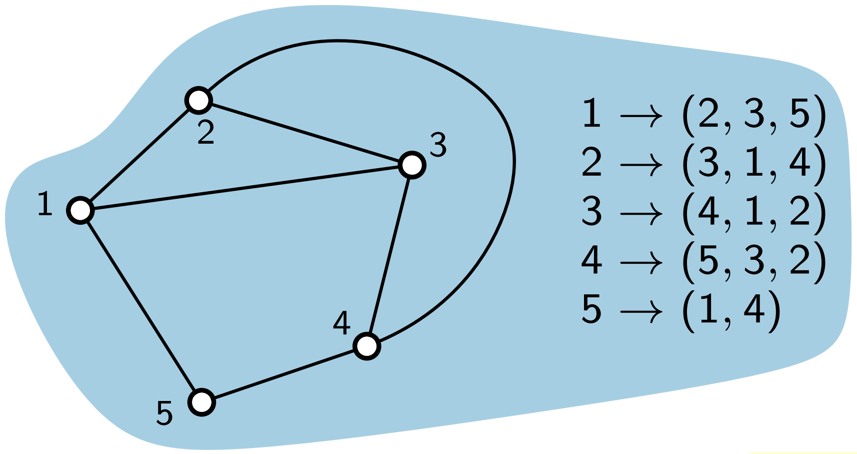
faces: Connected region of the plane bounded by edges

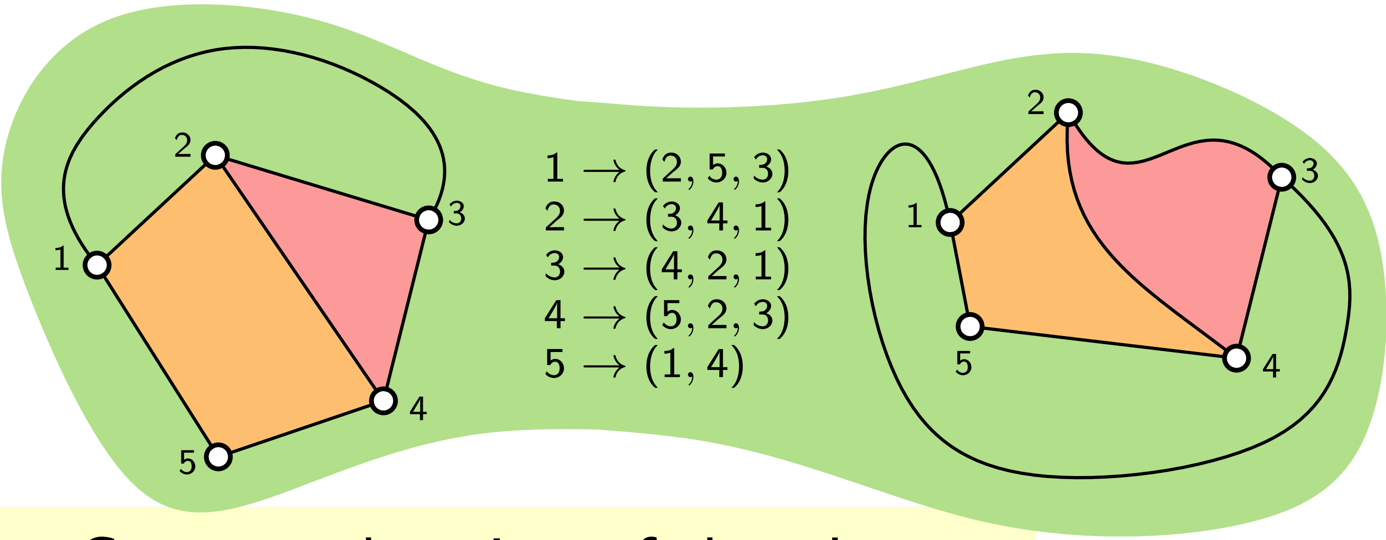
A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

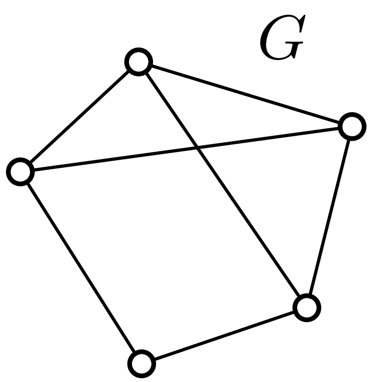
planar embedding:
clockwise orientation of adjacent vertices around each vertex

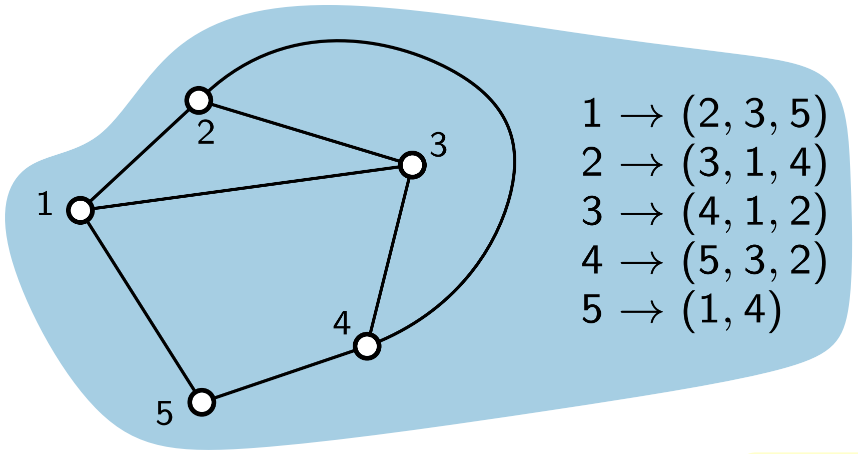
faces: Connected region of the plane bounded by edges

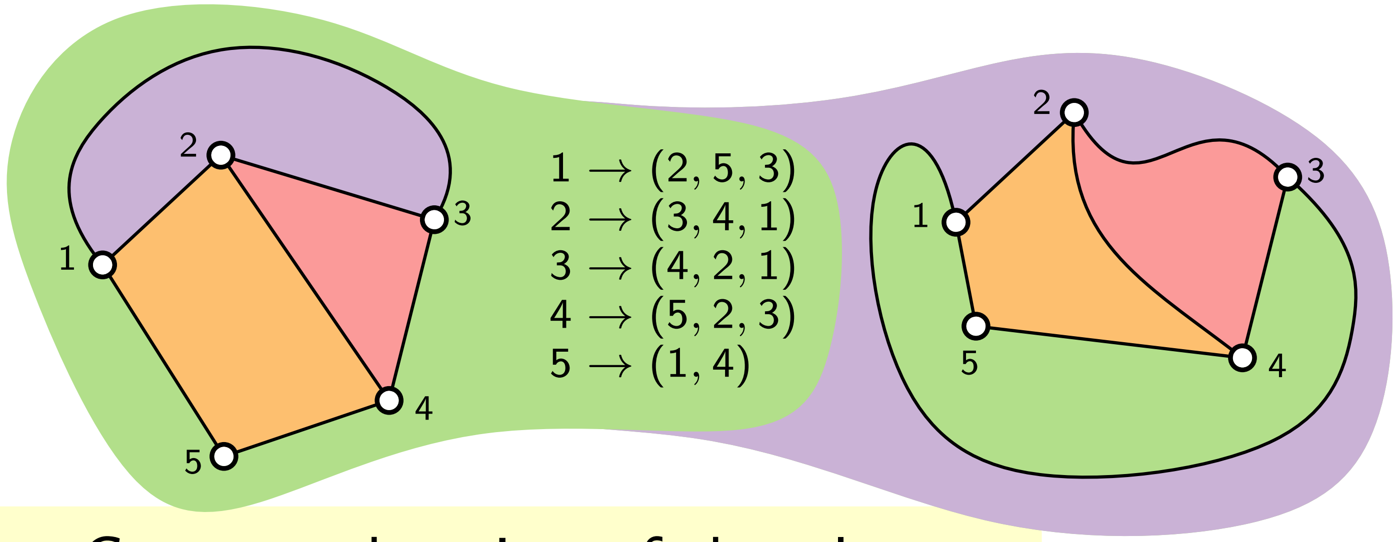
A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

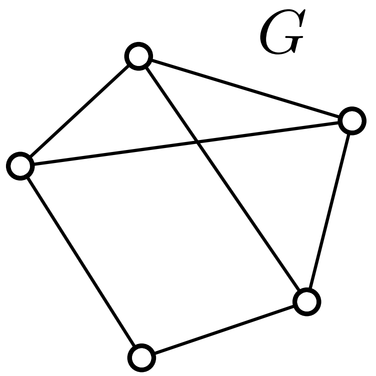
planar embedding:
clockwise orientation of adjacent vertices around each vertex

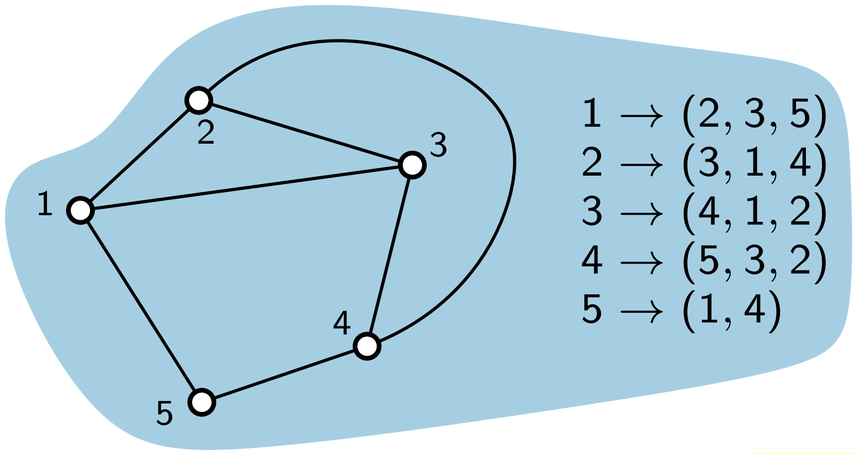
faces: Connected region of the plane bounded by edges

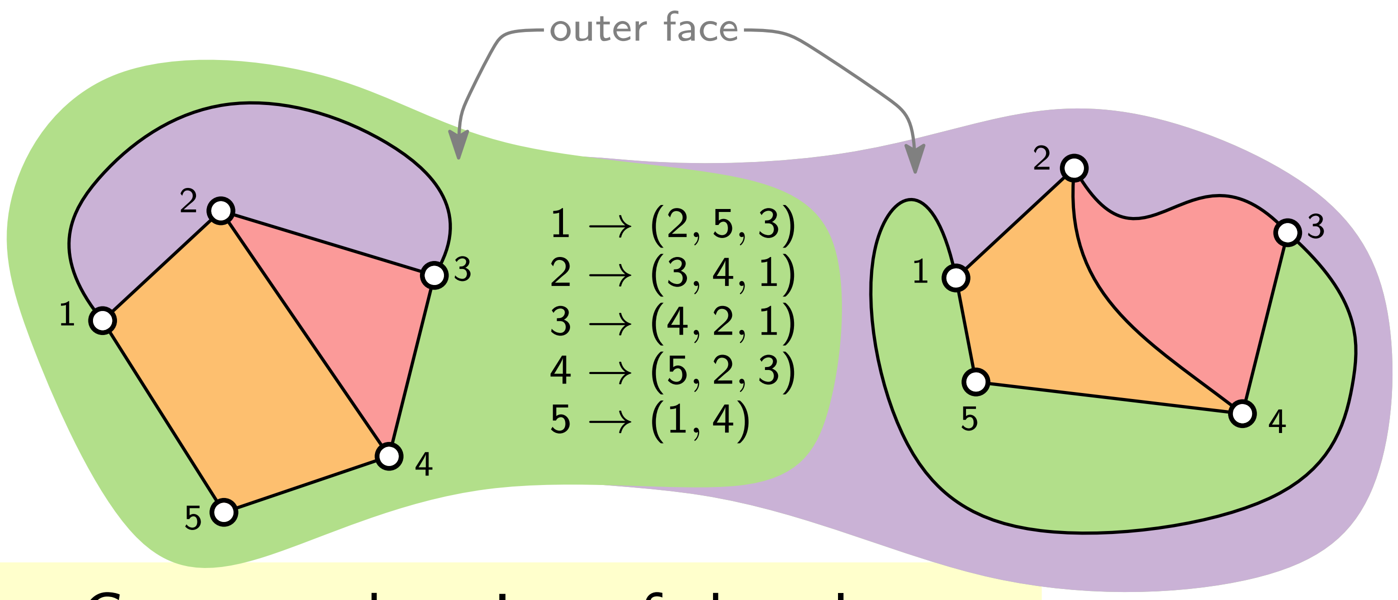
A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

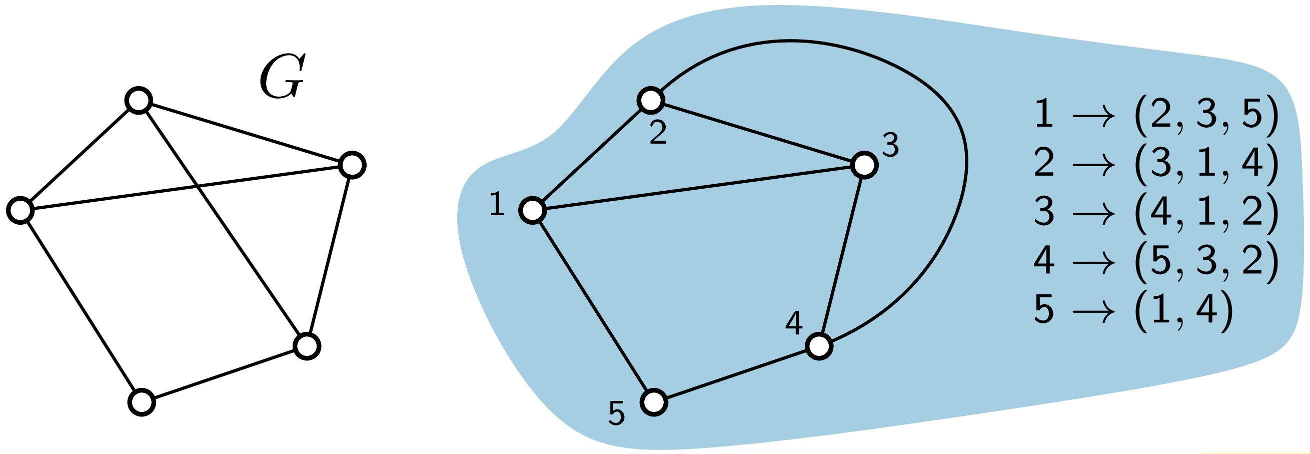
planar embedding:
clockwise orientation of adjacent vertices around each vertex

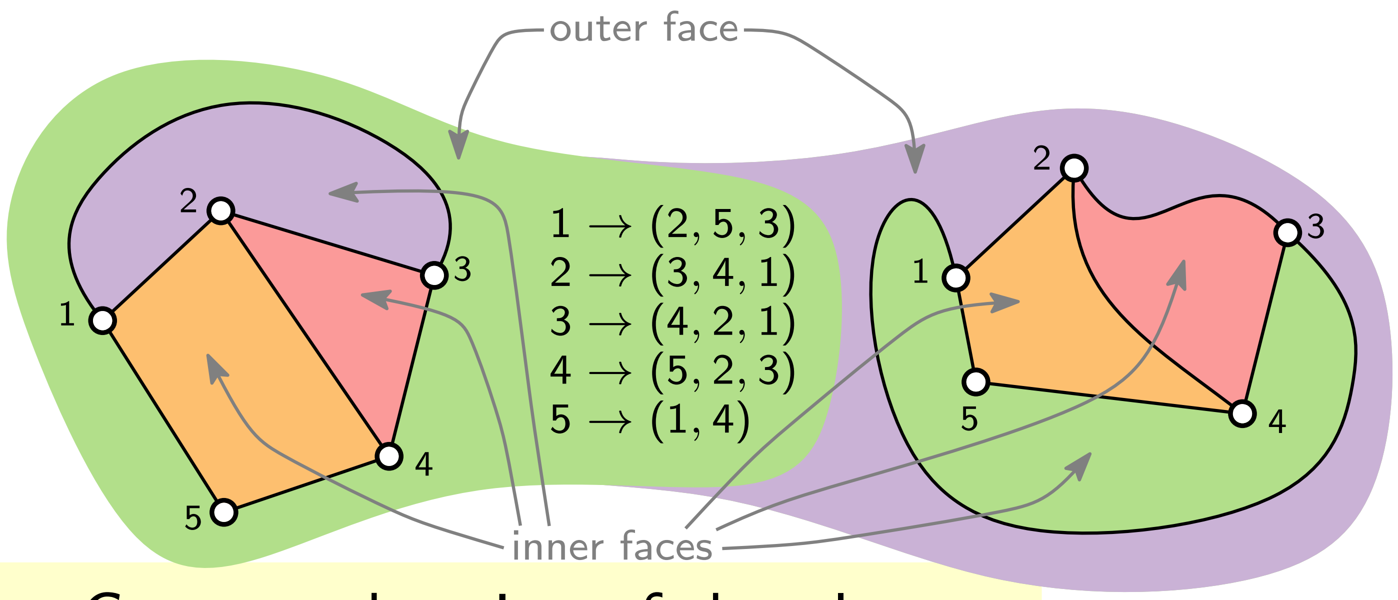
A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Planar Graphs





G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

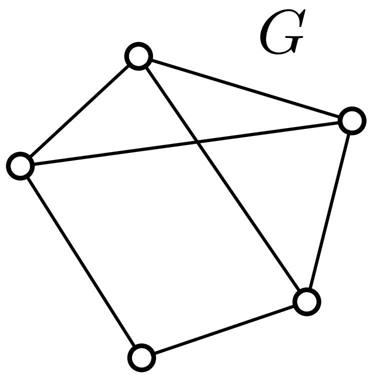
planar embedding:
clockwise orientation of adjacent vertices around each vertex

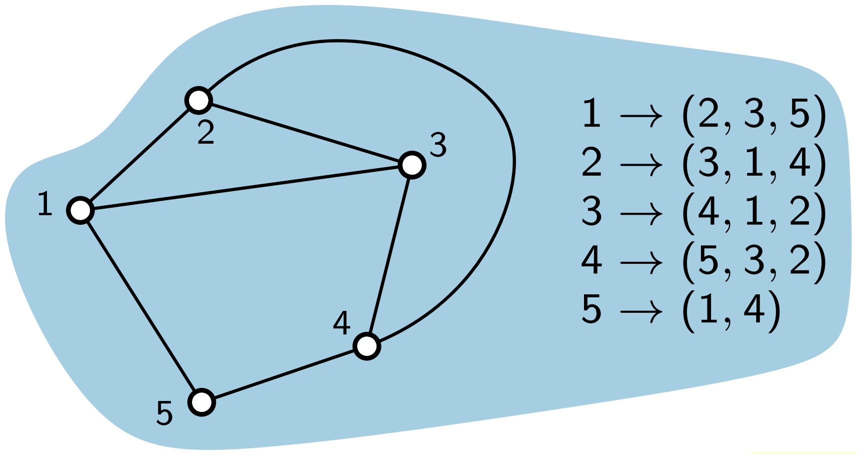
A planar graph can have many planar embeddings.

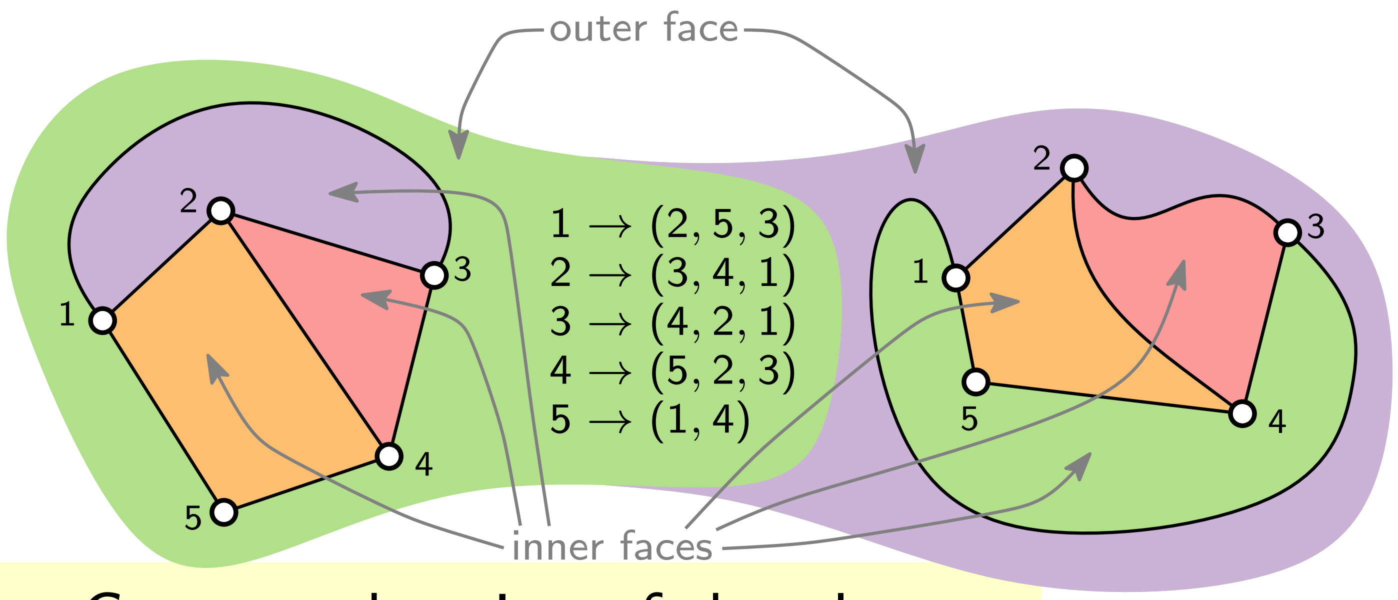
A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

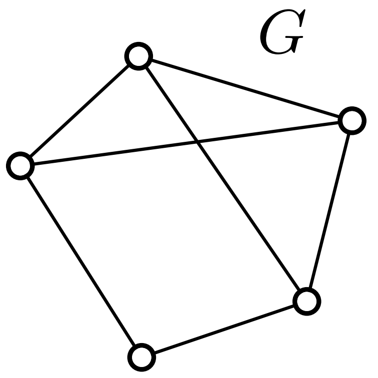
A planar embedding can have many planar drawings!

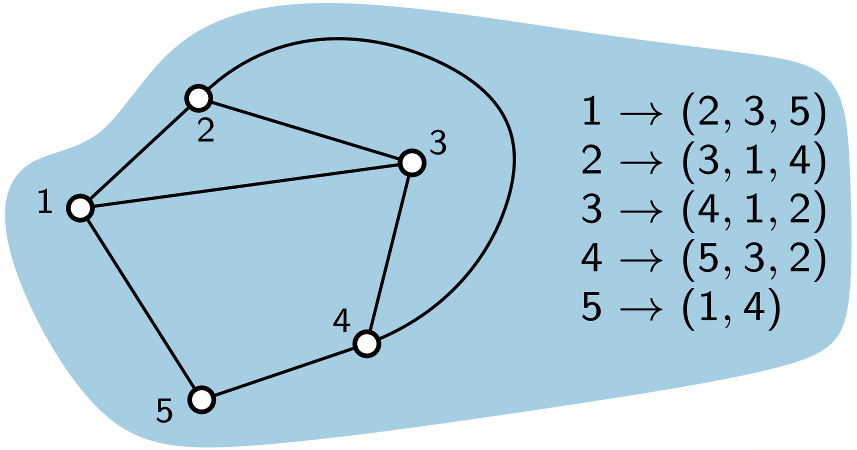
faces: Connected region of the plane bounded by edges

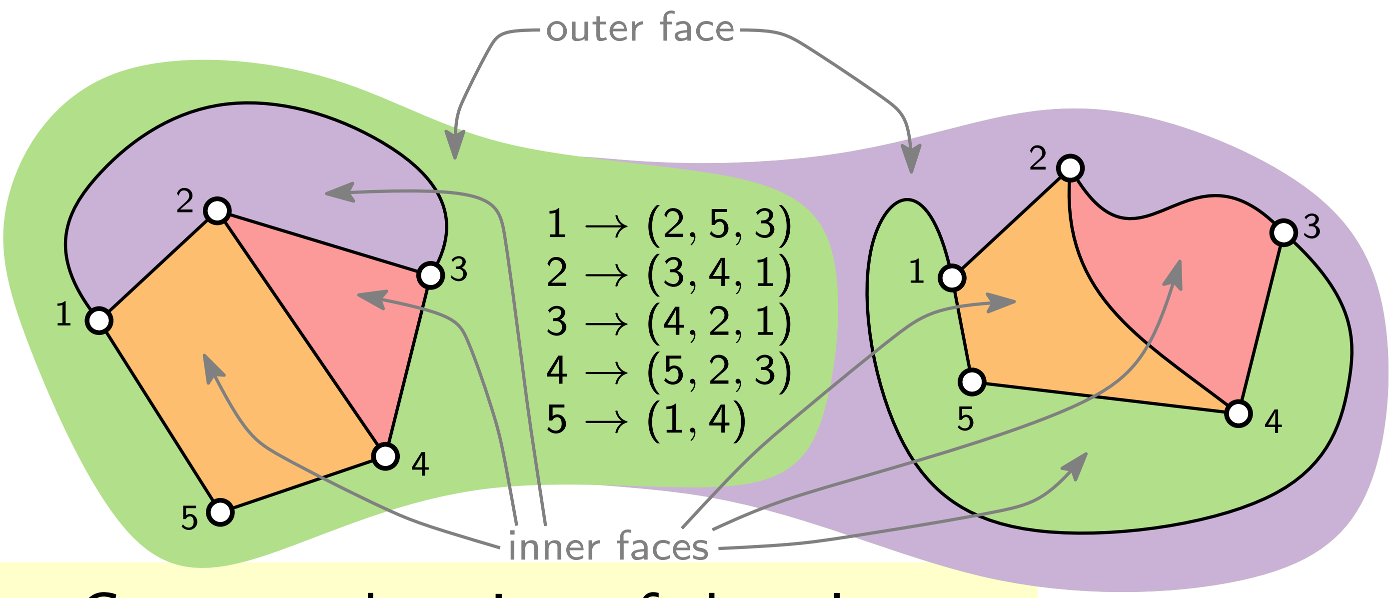
Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

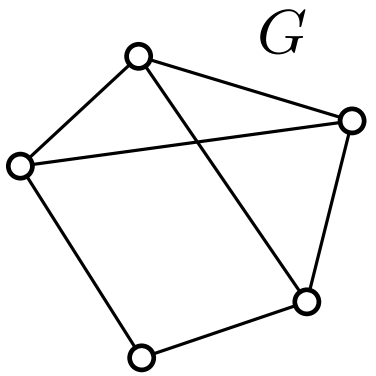
faces: Connected region of the plane bounded by edges

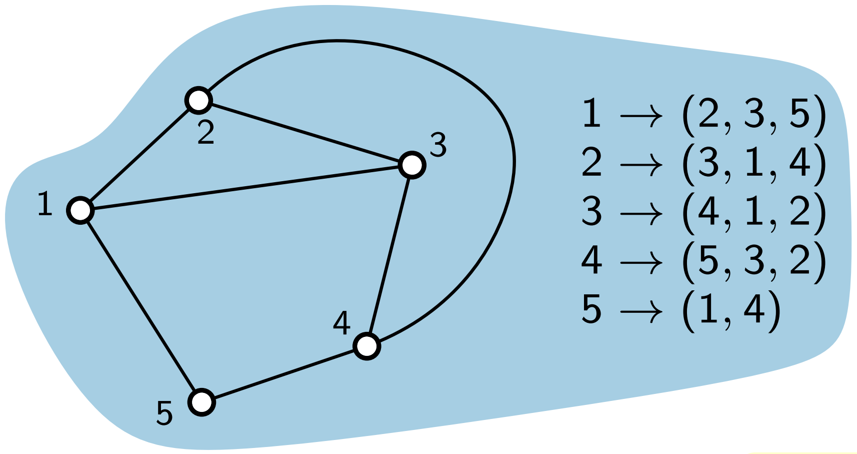
Euler's polyhedra formula.

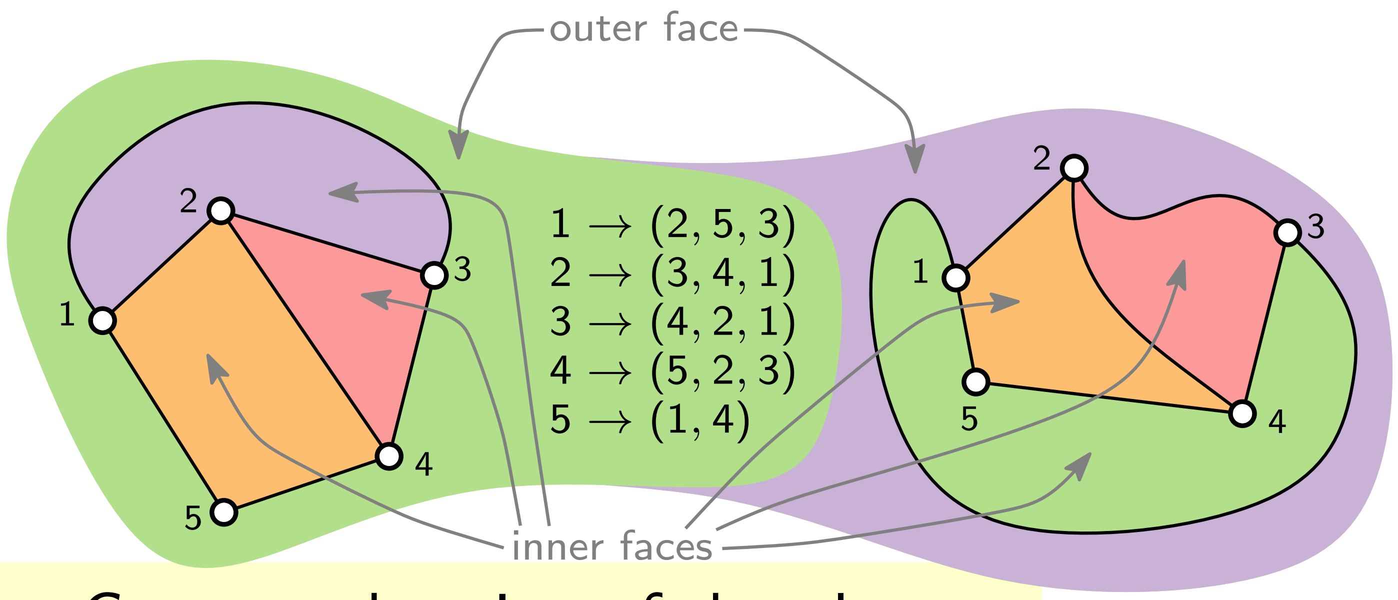
$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Proof.

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

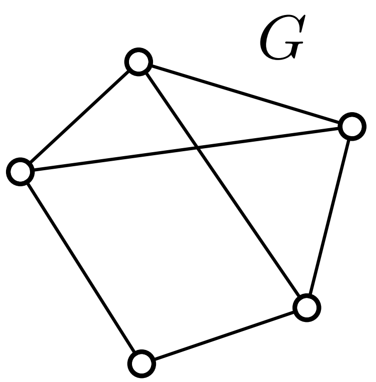
faces: Connected region of the plane bounded by edges

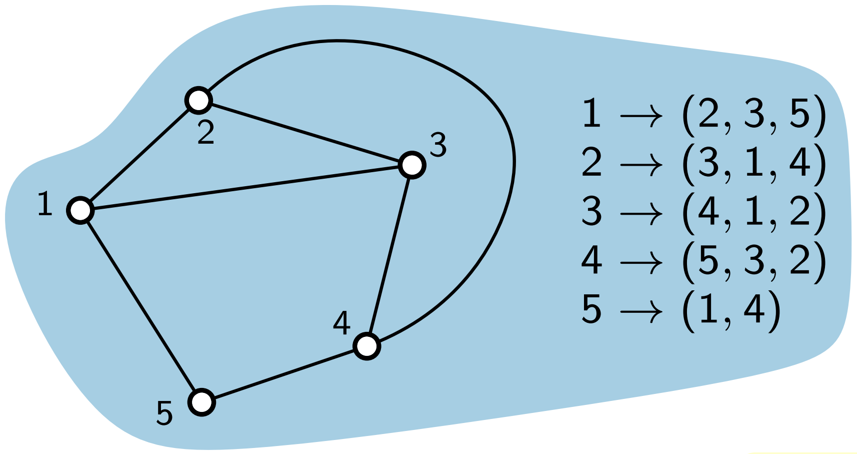
Euler's polyhedra formula.

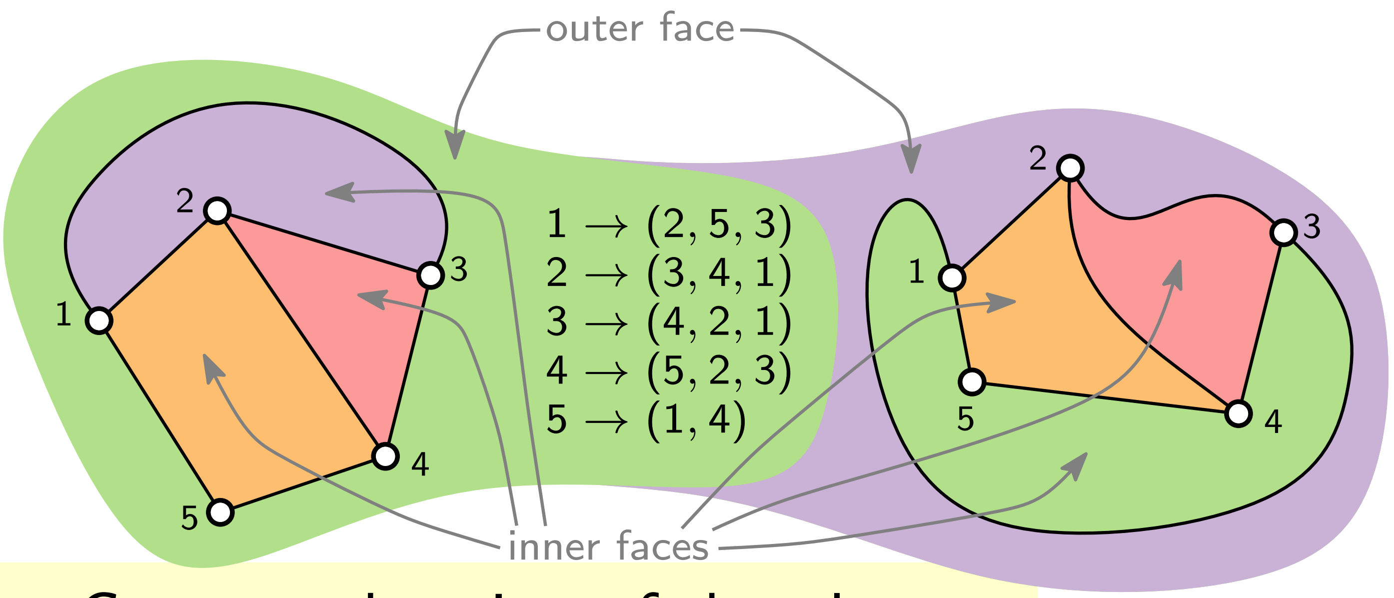
$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Proof. By induction on m:

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.

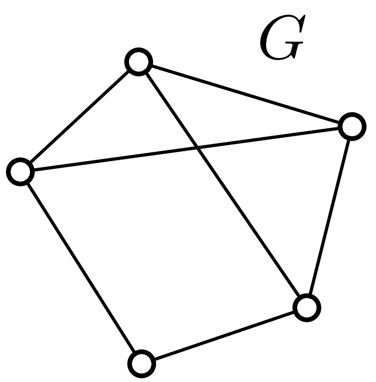
#faces − #edges + #vertices = #conn.comp. + 1

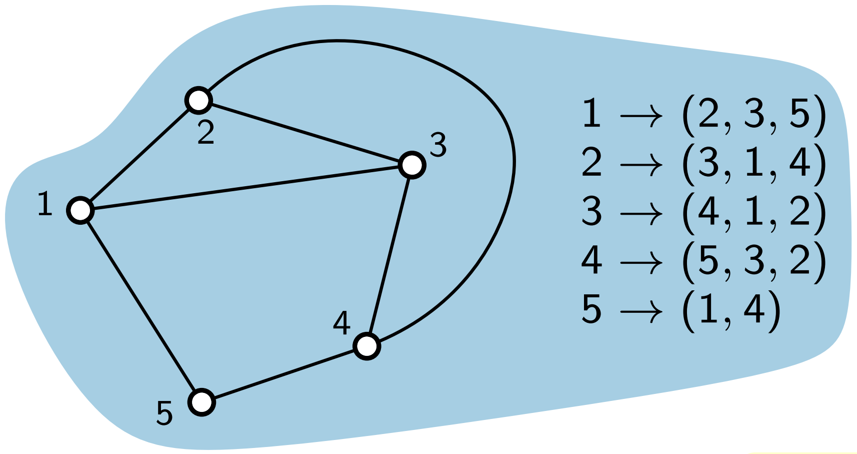
$$f \quad - \quad m \quad + \quad n \quad = \quad c \quad + 1$$

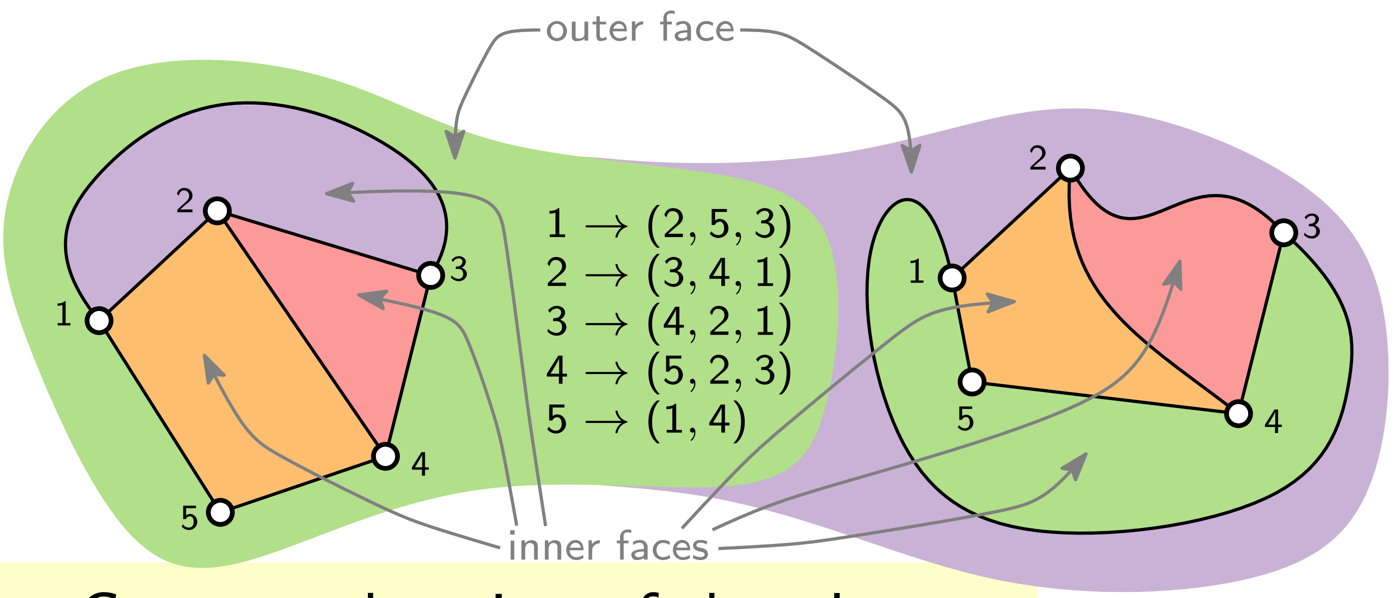
Proof. By induction on m:

$m = 0 \Rightarrow$

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.

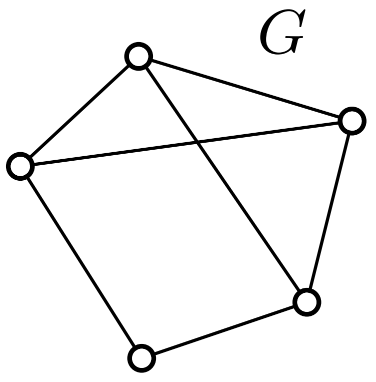
$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$

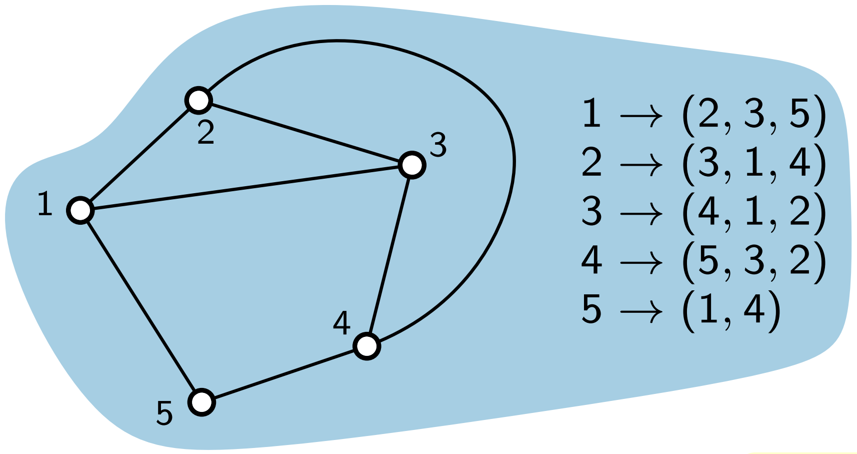
$$f - m + n = c + 1$$

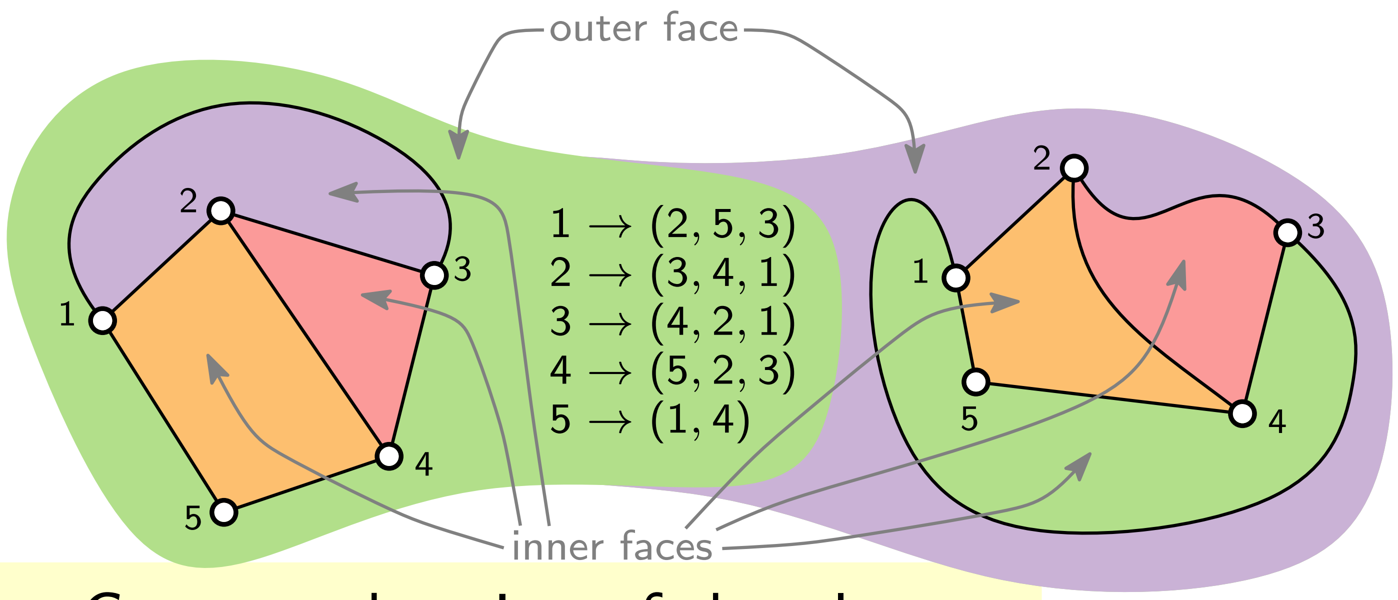
Proof. By induction on m:

$m = 0 \Rightarrow f =$? and $c =$?

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

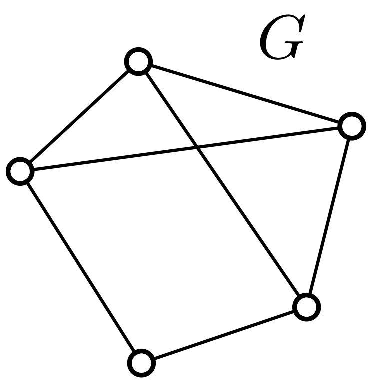
faces: Connected region of the plane bounded by edges

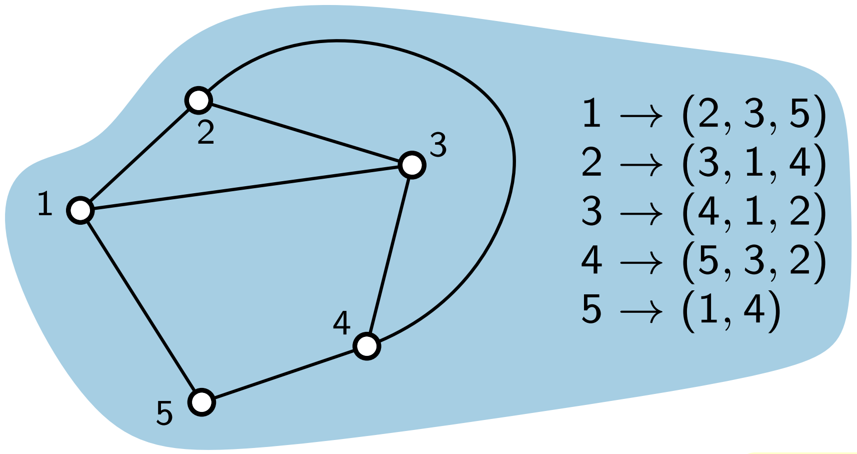
Euler's polyhedra formula.

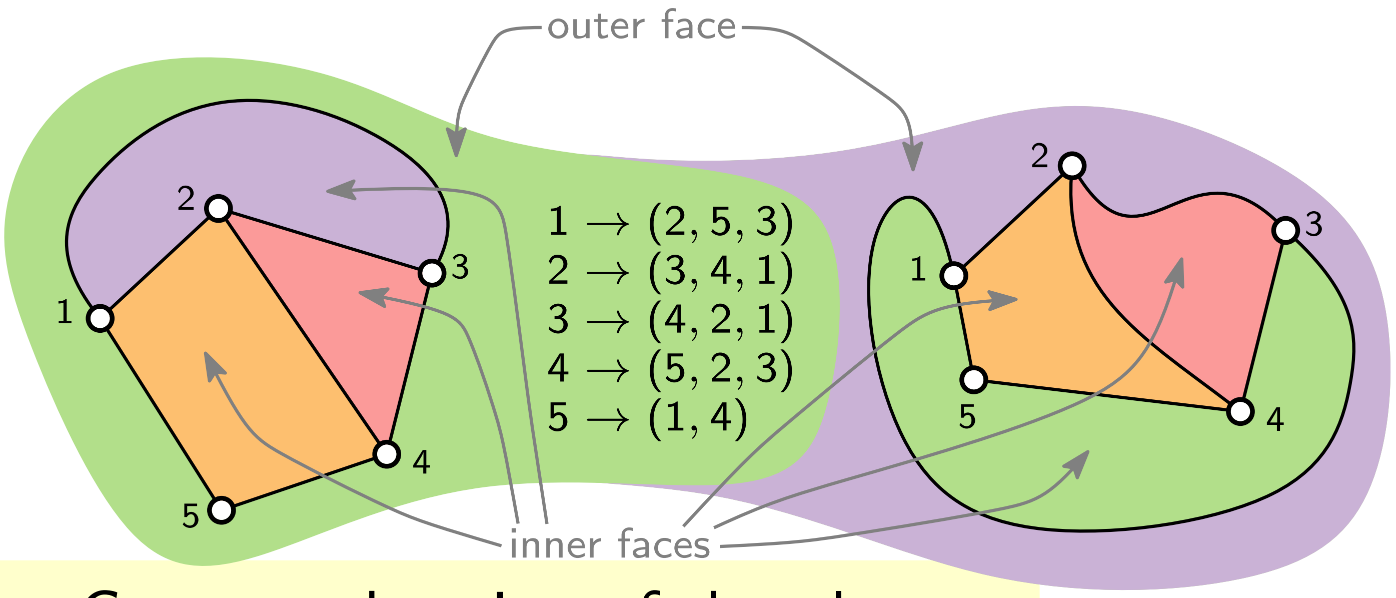
$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Proof. By induction on m:
$m = 0 \Rightarrow f = 1$ and $c = n$

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

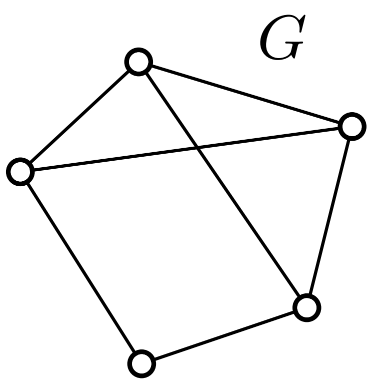
Euler's polyhedra formula.

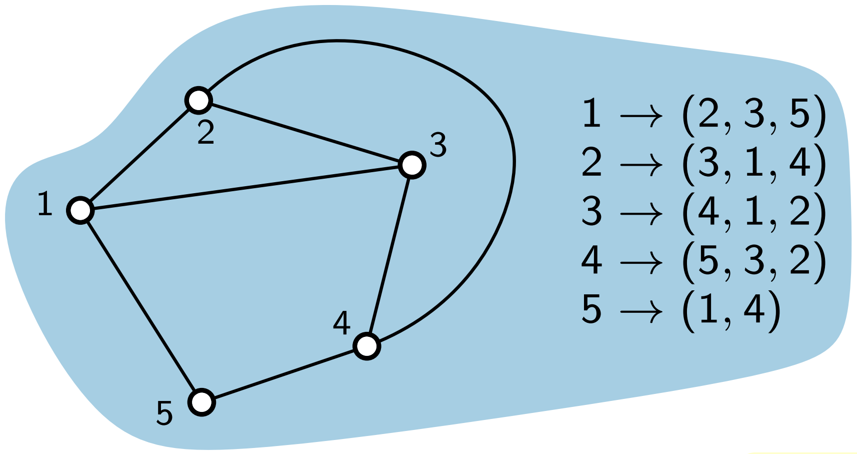
$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

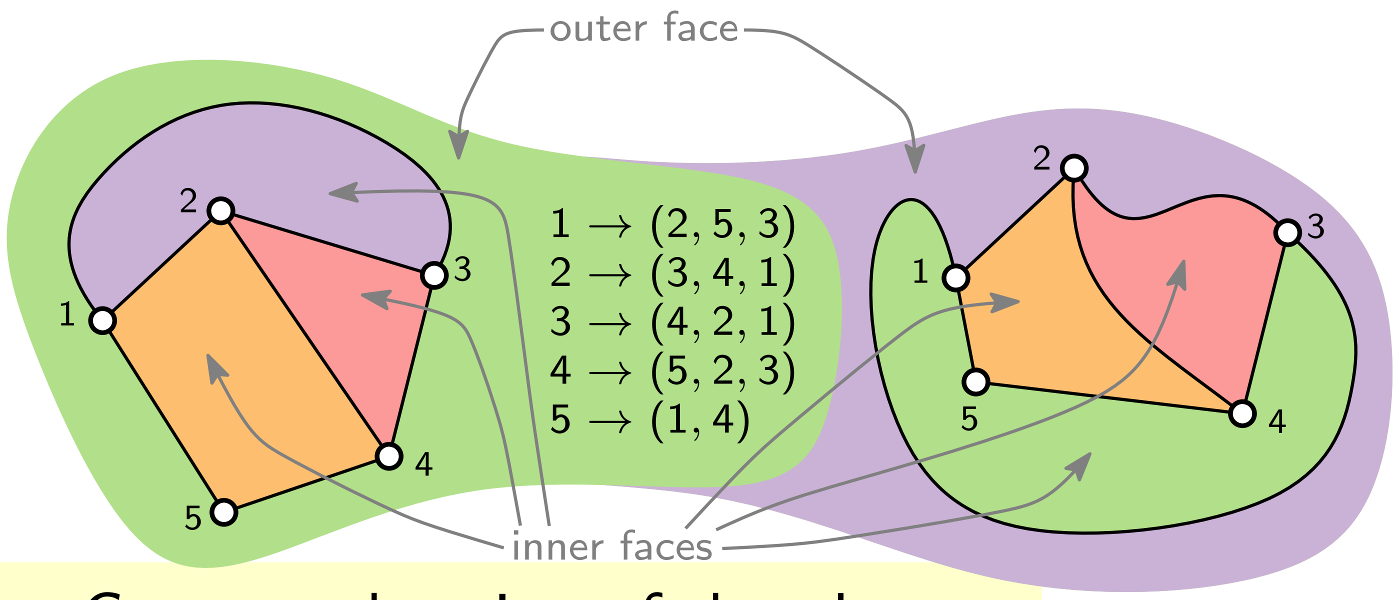
Proof. By induction on m:

$m = 0 \Rightarrow f = 1$ and $c = n$ ✓

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.

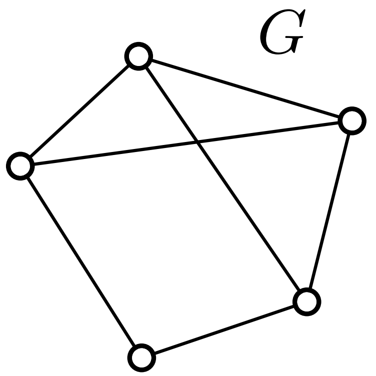
$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

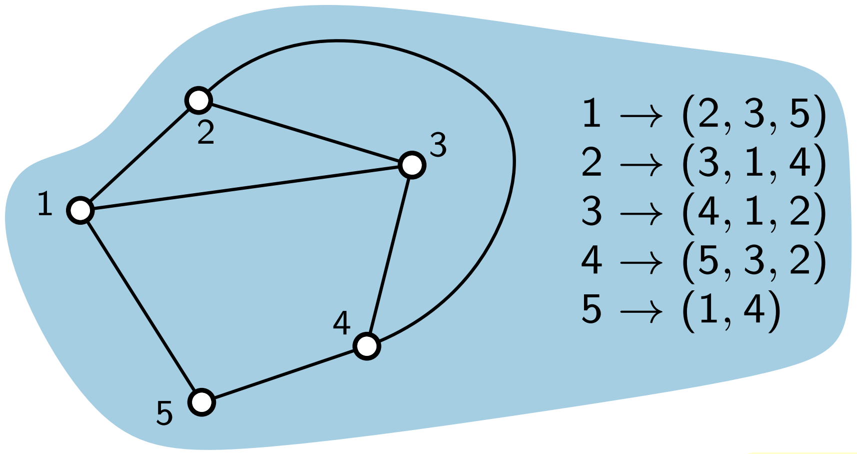
Proof. By induction on m:

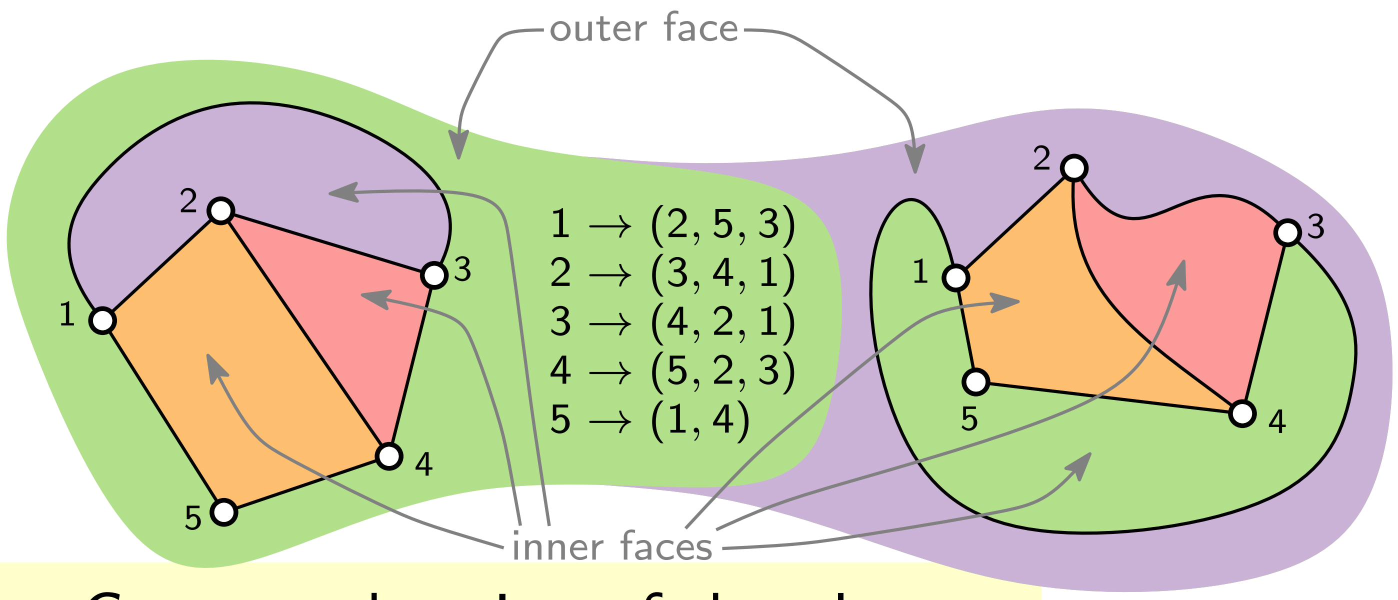
$m = 0 \Rightarrow f = 1$ and $c = n$ ✓

$m \geq 1 \Rightarrow$

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

A planar graph can have many planar embeddings.

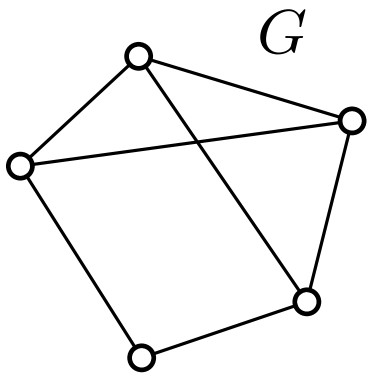
A planar embedding can have many planar drawings!

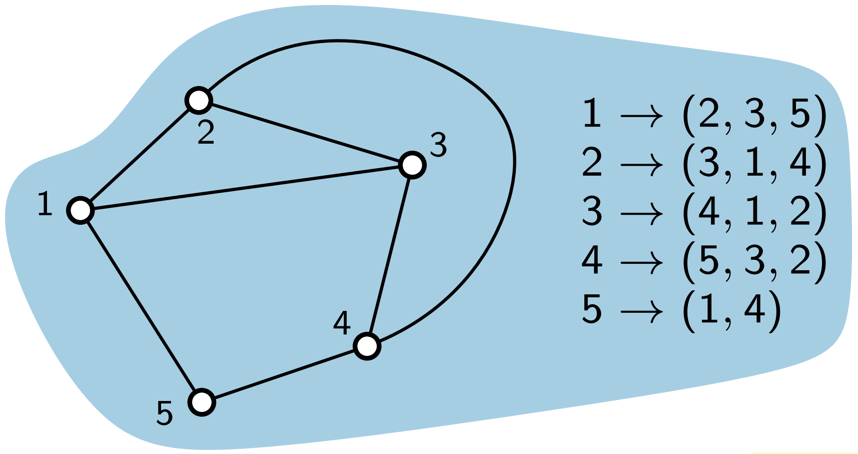
Proof. By induction on m:

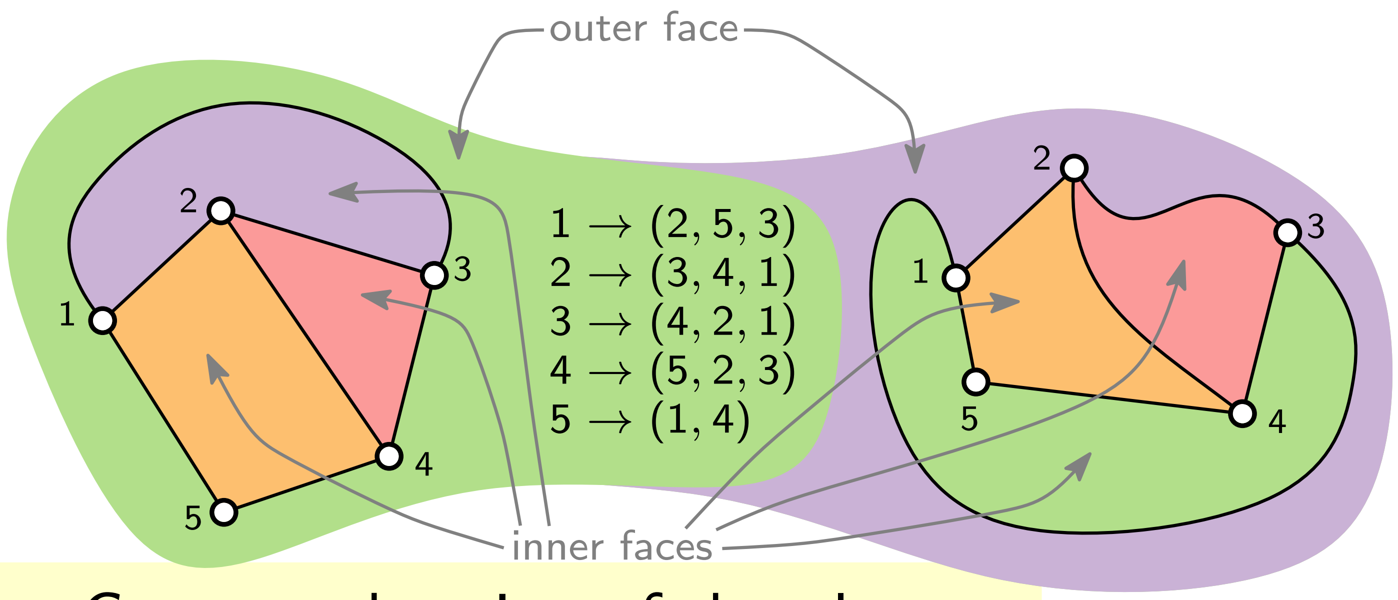
$m = 0 \Rightarrow f = 1$ and $c = n$ ✓

$m \geq 1 \Rightarrow$ delete some edge e

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.

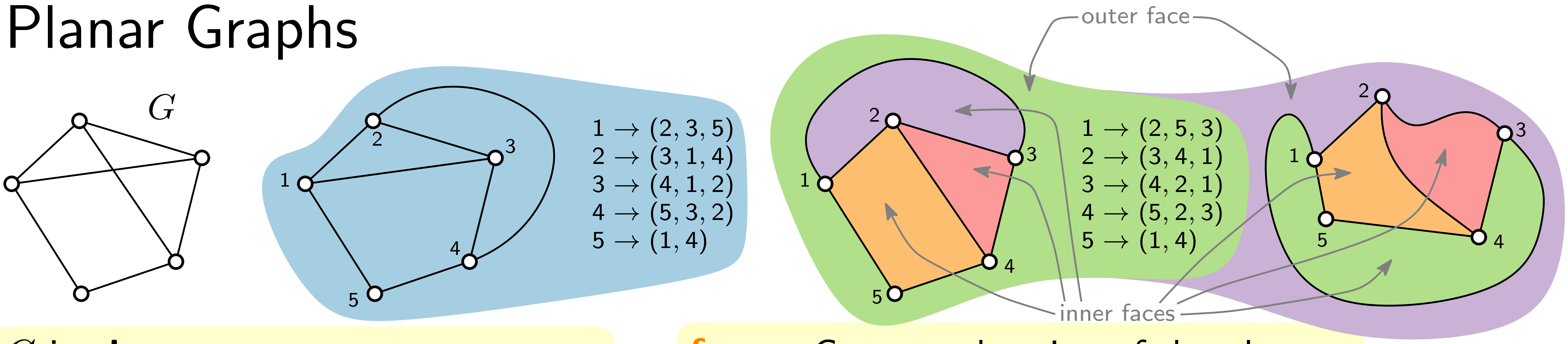
$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Proof. By induction on m:

$m = 0 \Rightarrow f = 1$ and $c = n$ ✓

$m \geq 1 \Rightarrow$ delete some edge e $\Rightarrow m' = m - 1$

Planar Graphs



G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.

#faces − #edges + #vertices = #conn.comp. + 1

$$f - m + n = c + 1$$

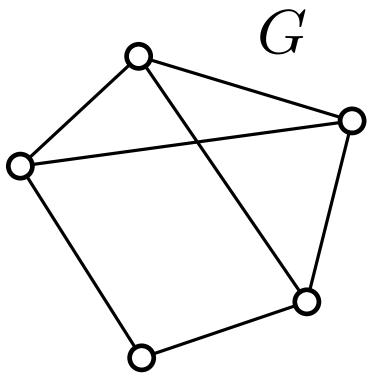
Proof. By induction on m:

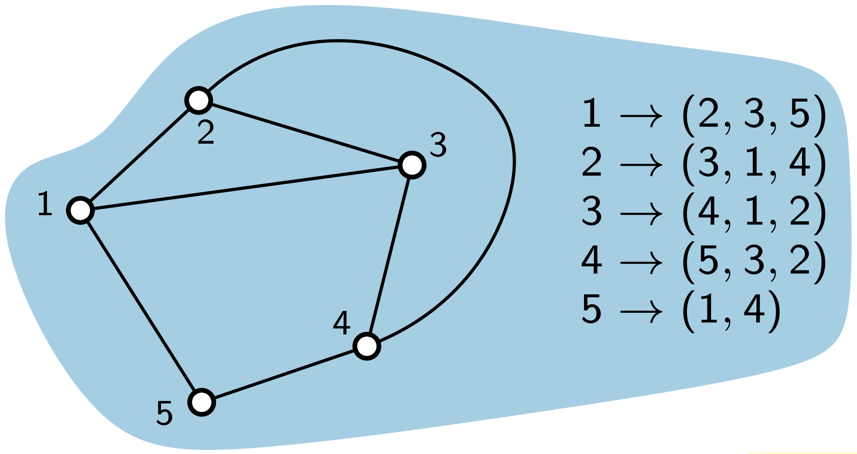
Induction hypothesis in G': $f' - m' + n' = c' + 1$

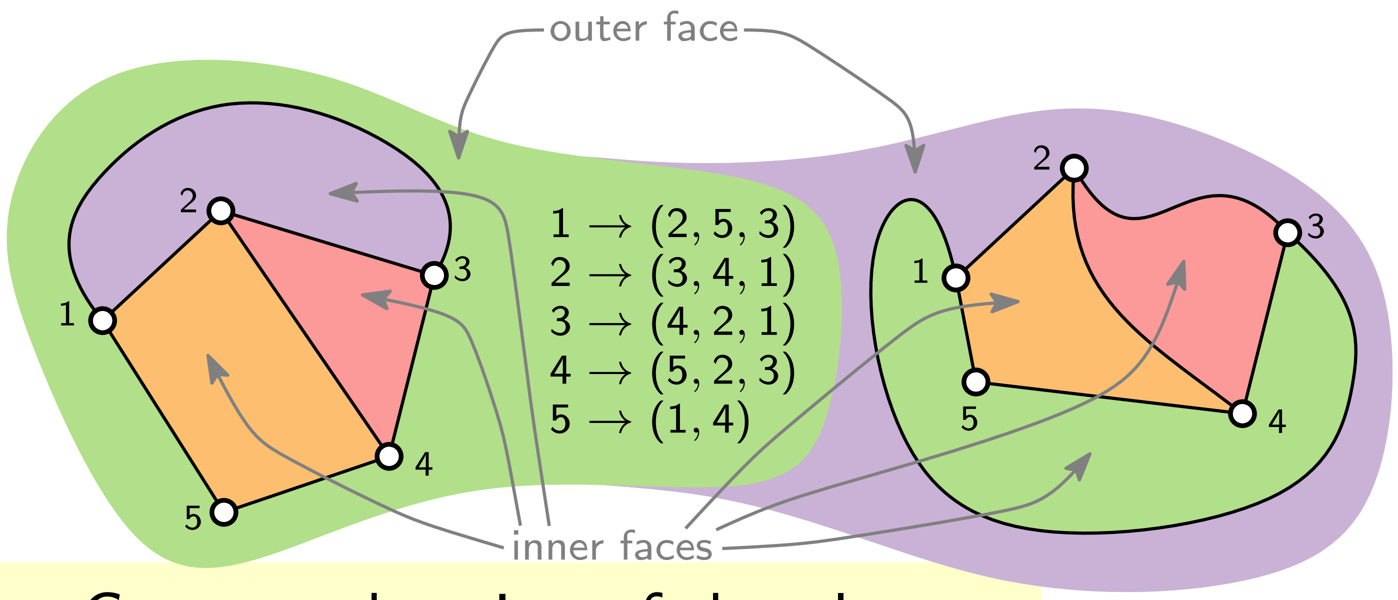
$m = 0 \Rightarrow f = 1$ and $c = n$ ✓

$m \geq 1 \Rightarrow$ delete some edge $e \Rightarrow m' = m - 1$

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.

#faces − #edges + #vertices = #conn.comp. + 1

$$f - m + n = c + 1$$

Proof. By induction on m:

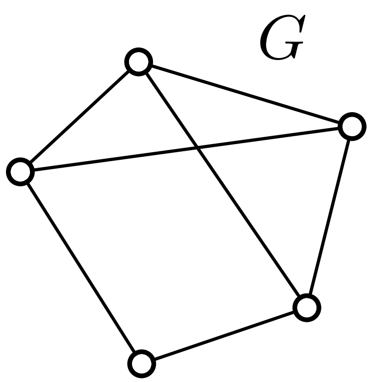
Induction hypothesis in G':
$f' - m' + n' = c' + 1$

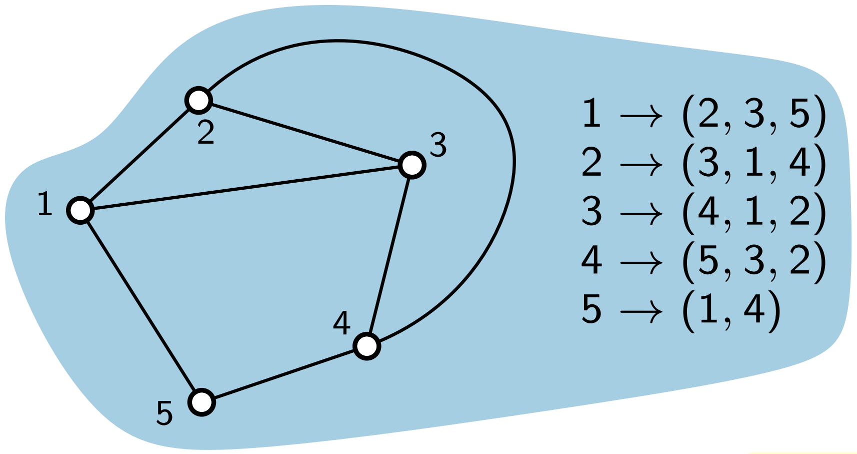
$m = 0 \Rightarrow f = 1$ and $c = n$ ✓

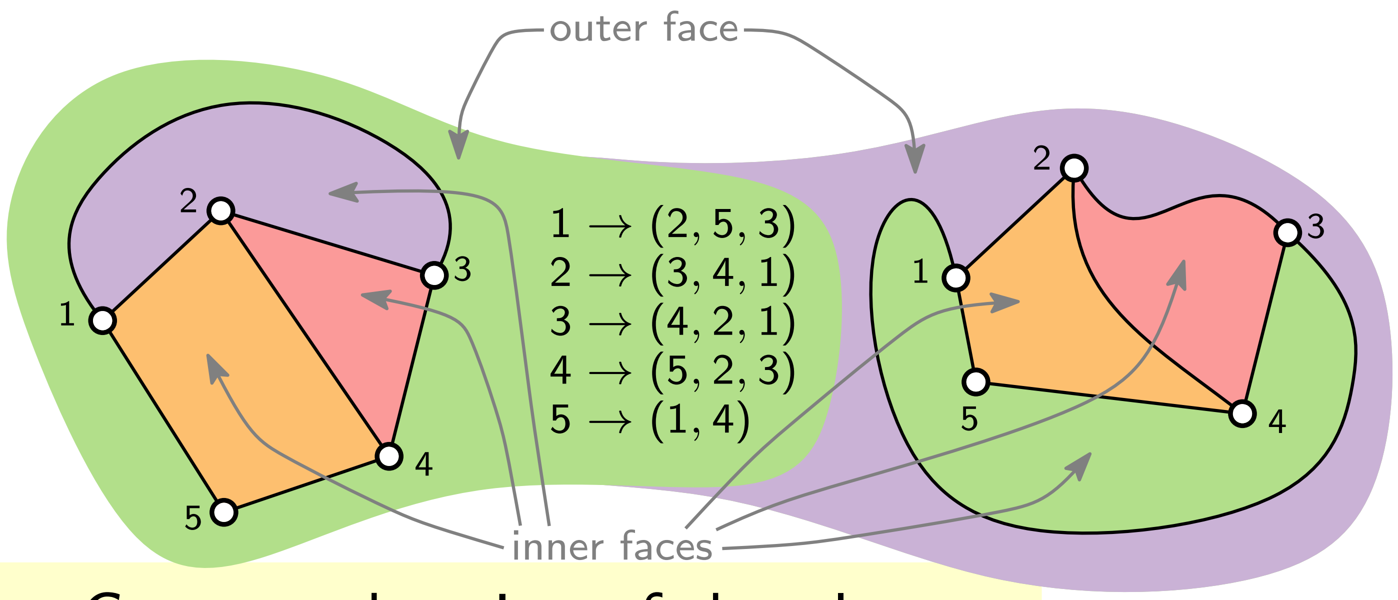
$m \geq 1 \Rightarrow$ delete some edge e $\Rightarrow m' = m - 1$

$\Rightarrow$

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Proof. By induction on m:

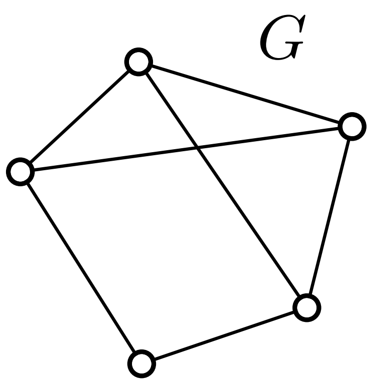
$m = 0 \Rightarrow f = 1$ and $c = n$ ✓

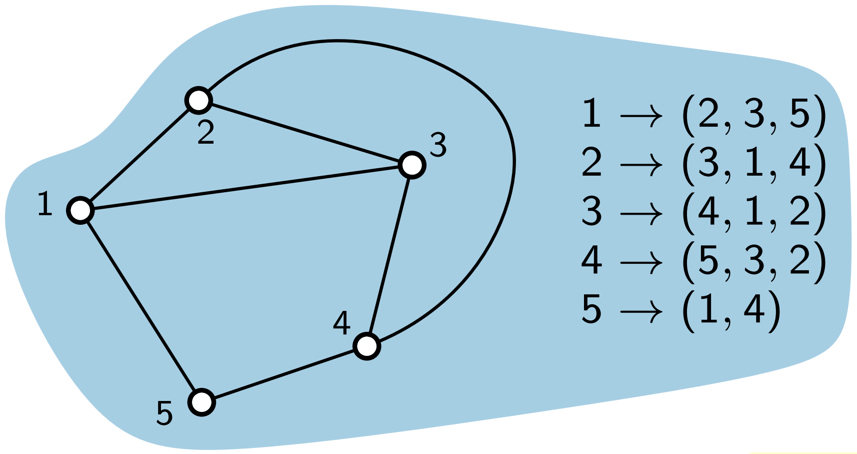
Induction hypothesis in G': $f' - m' + n' = c' + 1$

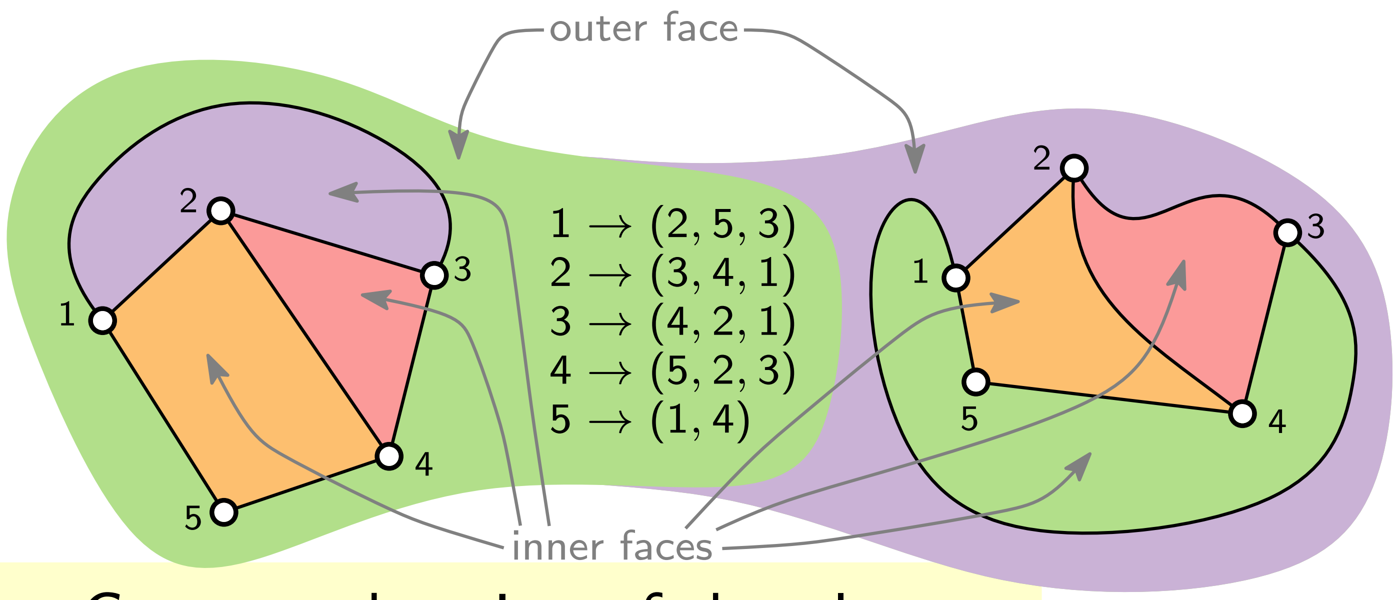
$m \geq 1 \Rightarrow$ delete some edge $e \Rightarrow m' = m - 1$

$\Rightarrow c' = c + 1$

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Proof. By induction on m:

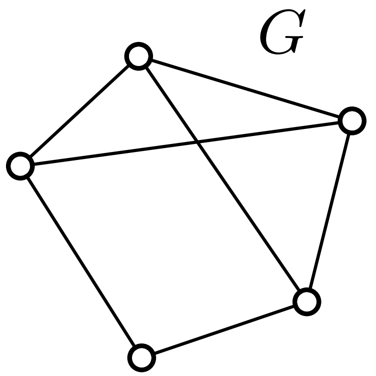
Induction hypothesis in G':
$f' - m' + n' = c' + 1$

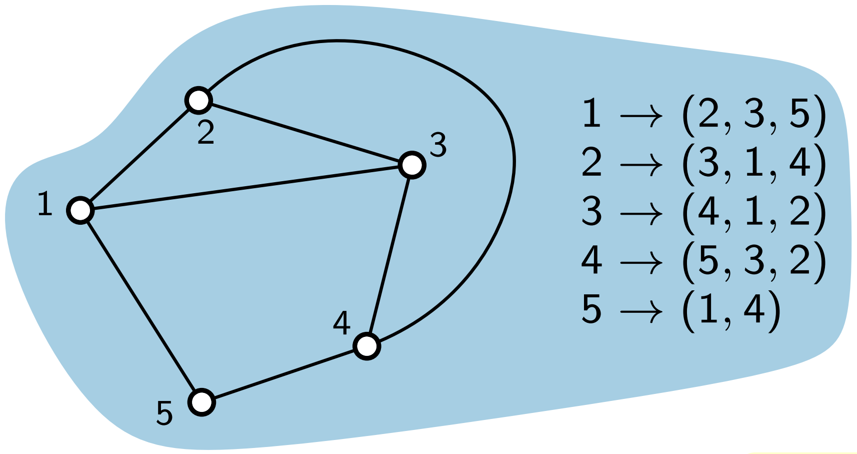
$m = 0 \Rightarrow f = 1$ and $c = n$ ✓

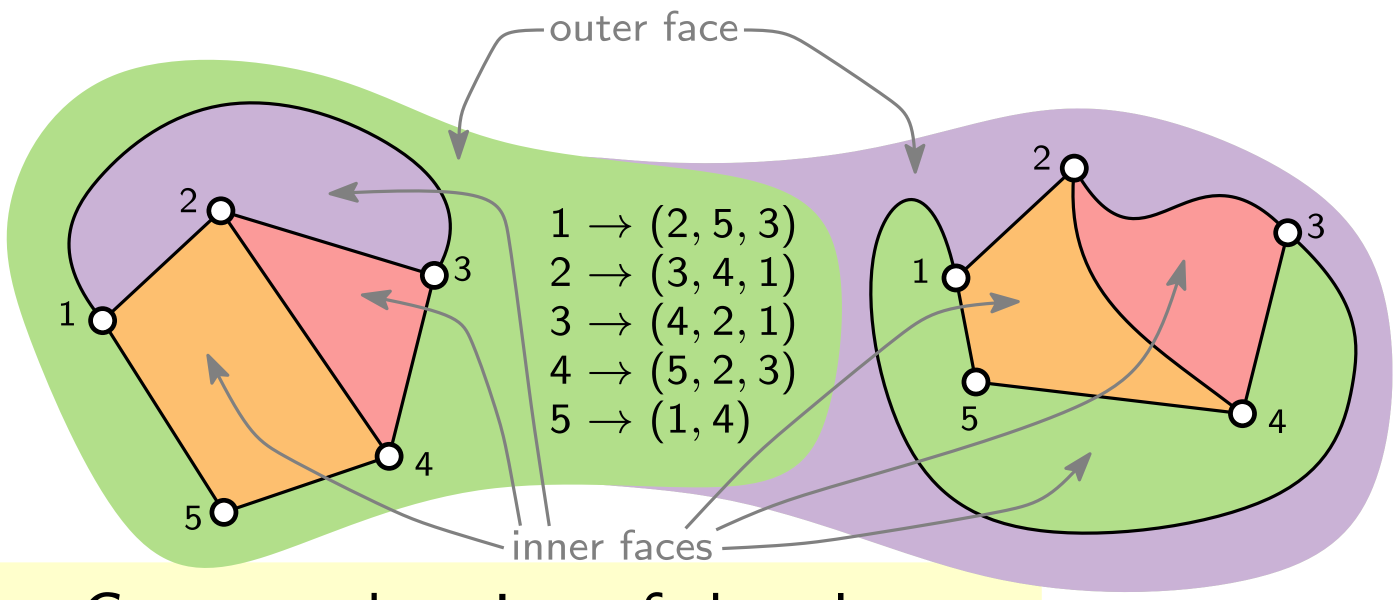
$m \geq 1 \Rightarrow$ delete some edge $e \Rightarrow m' = m - 1$

$\Rightarrow c' = c + 1$

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.
#faces − #edges + #vertices = #conn.comp. + 1

$$f - m + n = c + 1$$

Proof. By induction on m:

$m = 0 \Rightarrow f = 1$ and $c = n$ ✓

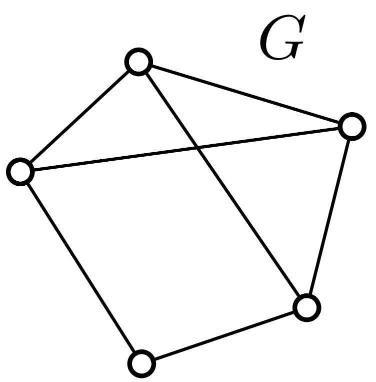
Induction hypothesis in G':
$f' - m' + n' = c' + 1$

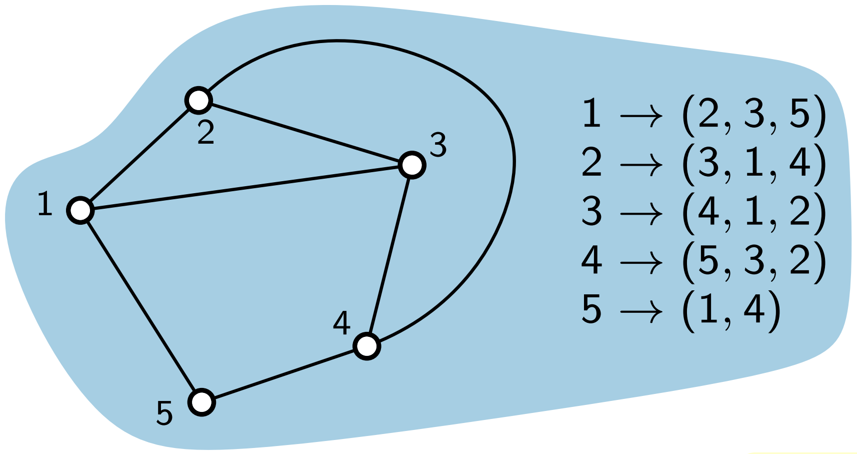
$m \geq 1 \Rightarrow$ delete some edge e $\Rightarrow m' = m - 1$

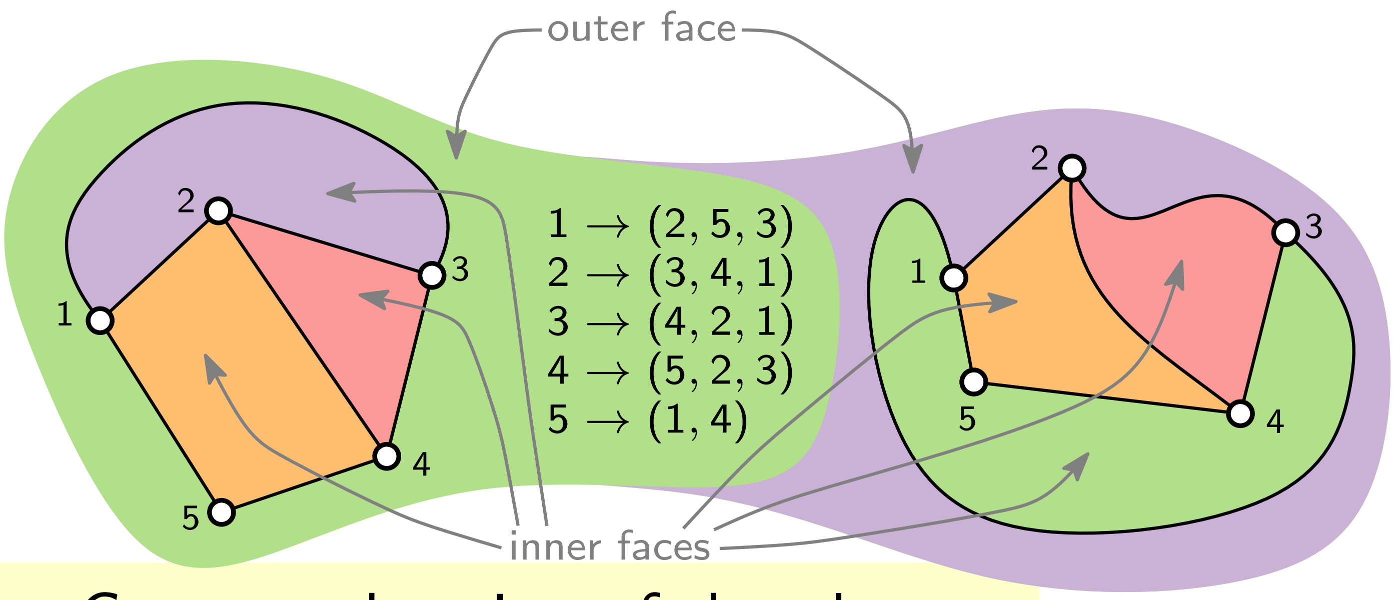
$\Rightarrow c' = c + 1$

$\Rightarrow f' = f - 1$

Planar Graphs







G is **planar**:
it can be drawn in such a way that no two edges intersect each other.

planar embedding:
clockwise orientation of adjacent vertices around each vertex

A planar graph can have many planar embeddings.

A planar embedding can have many planar drawings!

faces: Connected region of the plane bounded by edges

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Proof. By induction on m:

Induction hypothesis in G': $f' - m' + n' = c' + 1$

$m = 0 \Rightarrow f = 1$ and $c = n$ ✓

$m \geq 1 \Rightarrow$ delete some edge $e \Rightarrow m' = m - 1$

$\Rightarrow c' = c + 1$

$\Rightarrow f' = f - 1$ ✓

Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Properties of Planar Graphs

Euler's polyhedra formula.

$$\begin{array}{ccccccccc} \#\text{faces} & - & \#\text{edges} & + & \#\text{vertices} & = & \#\text{conn.comp.} & + & 1 \\ f & - & m & + & n & = & c & + & 1 \end{array}$$

Theorem. G simple planar graph with $n \geq 3$ vtc.

Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.

1. $m \leq 3n - 6$

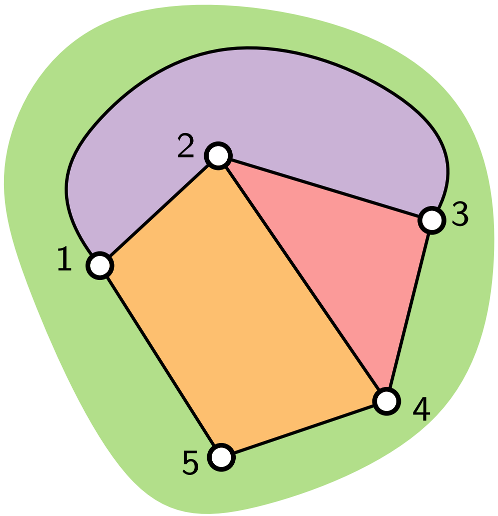
Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.

1. $m \leq 3n - 6$



Proof. 1.

Properties of Planar Graphs

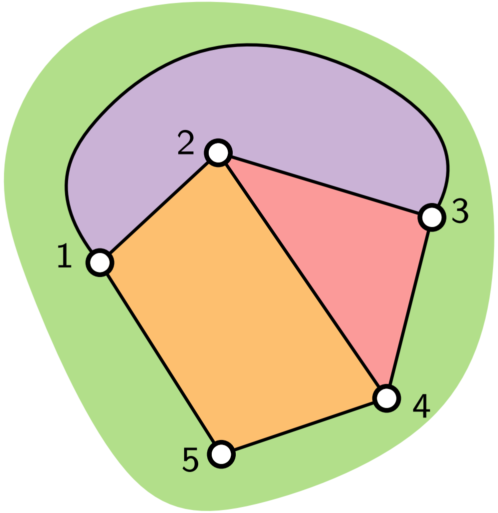
Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$

Proof. **1.**

idea: count edge–face incidences

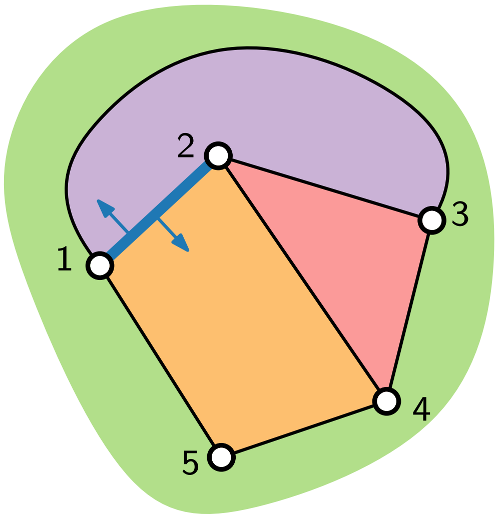


Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$



Proof. **1.** Every edge incident to ≤ 2 faces

idea: count edge–face incidences

Properties of Planar Graphs

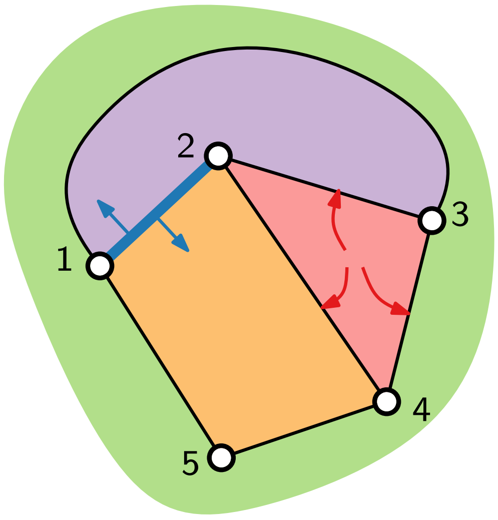
Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$

Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges

idea: count edge–face incidences

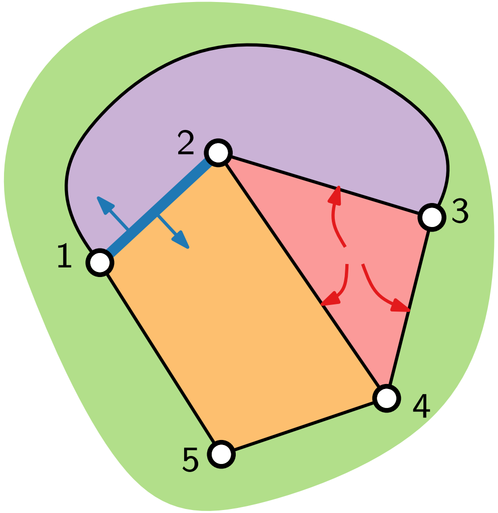


Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f$? $\#$ incidences ? $2m$

idea: count edge–face incidences

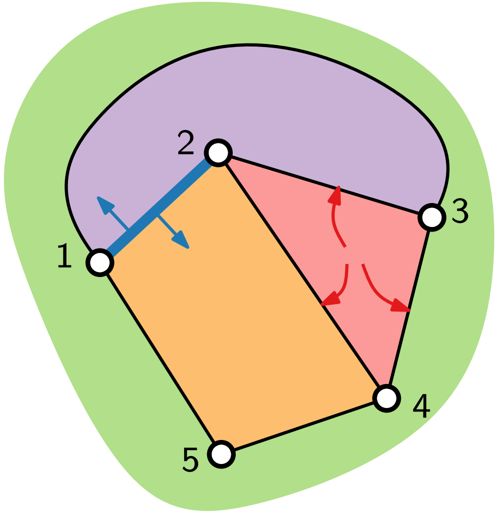
Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.

1. $m \leq 3n - 6$



Proof. **1.** Every edge incident to ≤ 2 faces

Every face incident to ≥ 3 edges

$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$

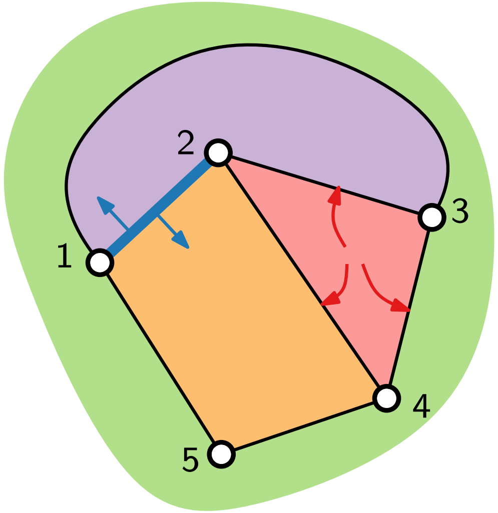
idea: count edge–face incidences

Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$
$\Rightarrow c + 1 = f - m + n$

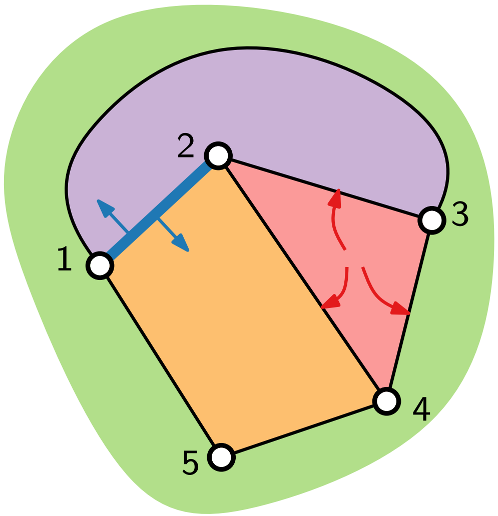
idea: count edge–face incidences

Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$
$\Rightarrow \quad 3c + 3 = 3f - 3m + 3n$

idea: count edge–face incidences

Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.

1. $m \leq 3n - 6$

Proof. 1. Every edge incident to ≤ 2 faces

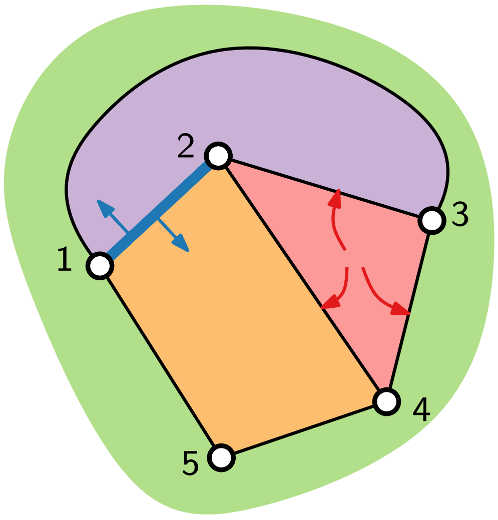
Every face incident to ≥ 3 edges

$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$

$\Rightarrow \quad 3c + 3 = 3f - 3m + 3n$

$c \geq 1$

idea: count edge–face incidences



Properties of Planar Graphs

Euler's polyhedra formula.

#faces − #edges + #vertices = #conn.comp. + 1

$f - m + n = c + 1$

Theorem. G simple planar graph with $n \geq 3$ vtc.

1. $m \leq 3n - 6$

Proof. **1.** Every edge incident to ≤ 2 faces

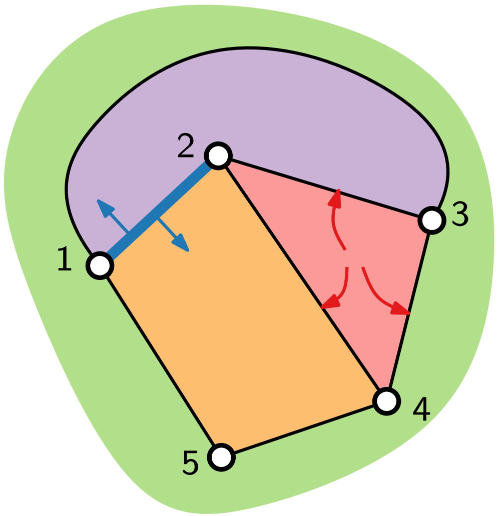
Every face incident to ≥ 3 edges

$\Rightarrow 3f \leq$ # incidences $\leq 2m$

$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n$

$c \geq 1$

idea: count edge–face incidences



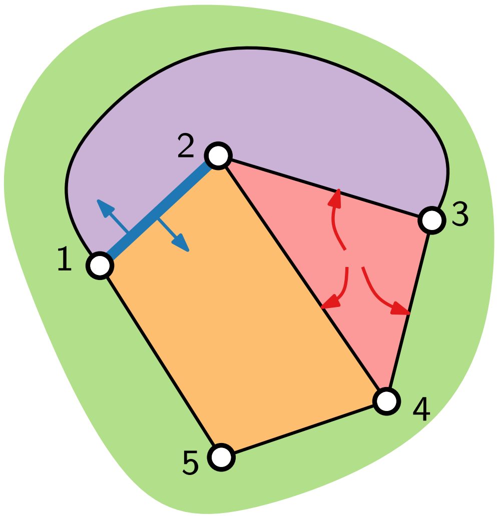
Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.

1. $m \leq 3n - 6$



Proof. **1.** Every edge incident to ≤ 2 faces

Every face incident to ≥ 3 edges

$\Rightarrow 3f \leq \#\text{incidences} \leq 2m$

$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n$

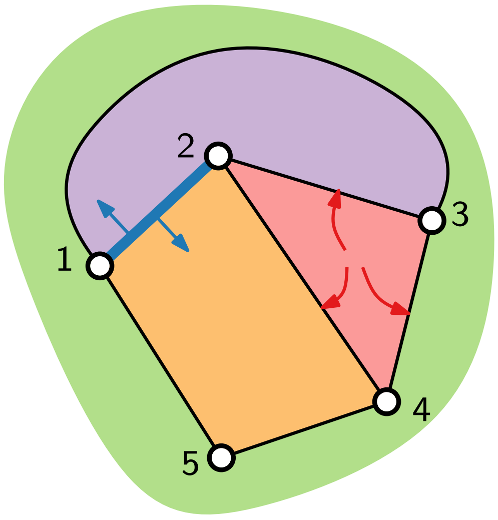
idea: count edge–face incidences

Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n$

idea: count edge–face incidences

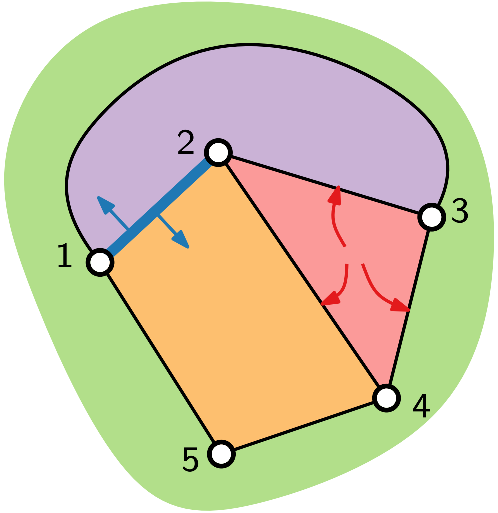
Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.

1. $m \leq 3n - 6$



Proof. **1.** Every edge incident to ≤ 2 faces

Every face incident to ≥ 3 edges

$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$

$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$

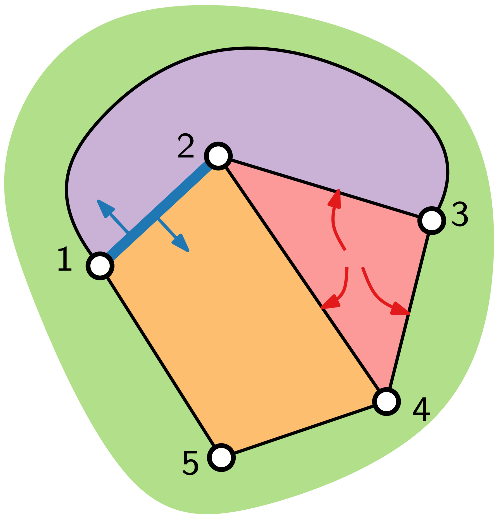
idea: count edge–face incidences

Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\text{incidences} \leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$

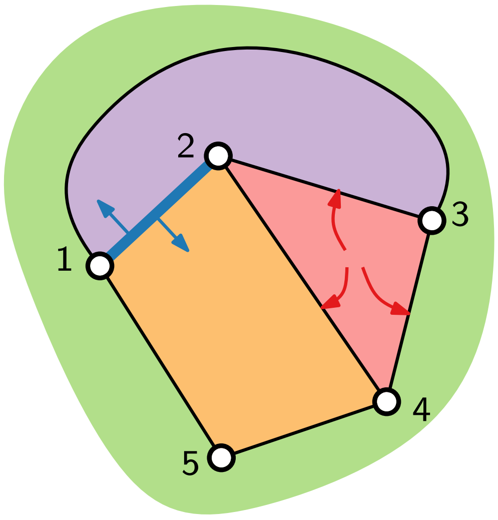
idea: count edge–face incidences

Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$

idea: count edge–face incidences

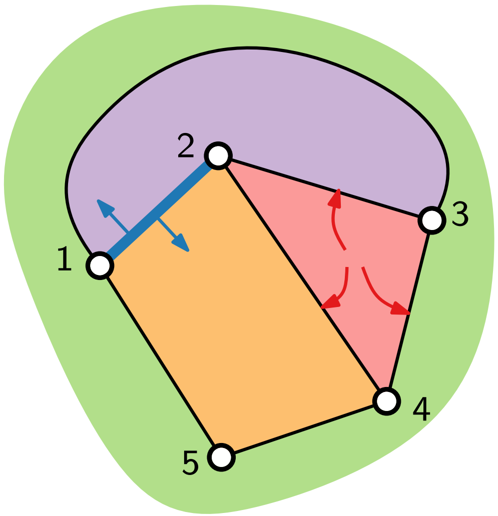
Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.

1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$



Proof. **1.** Every edge incident to ≤ 2 faces

Every face incident to ≥ 3 edges

$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$

$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$

$\Rightarrow m \leq 3n - 6$

2. $3f \leq 2m$

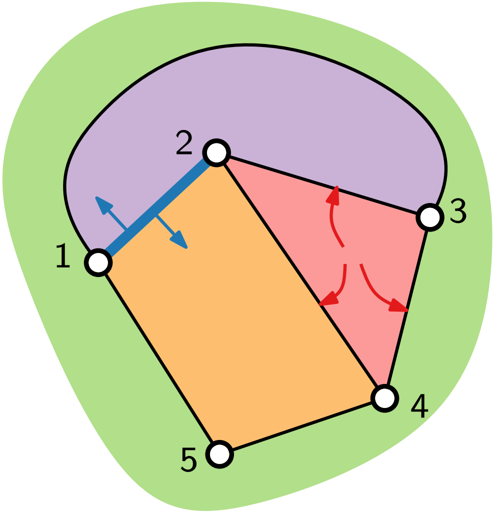
idea: count edge–face incidences

Properties of Planar Graphs

Euler's polyhedra formula.
#faces − #edges + #vertices = #conn.comp. + 1

$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$
2. $3f \leq 2m \leq 6n - 12$

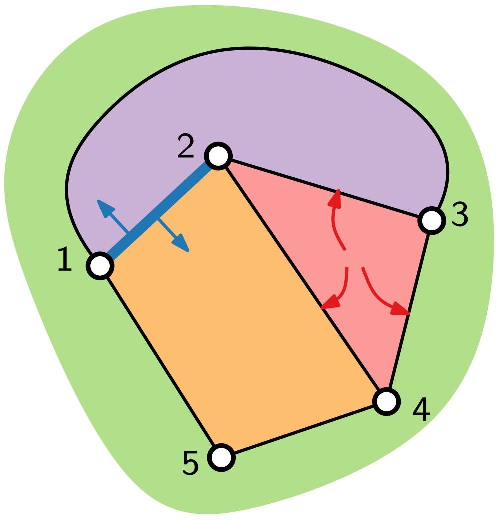
idea: count edge–face incidences

Properties of Planar Graphs

Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$
2. $3f \leq 2m \leq 6n - 12 \Rightarrow f \leq 2n - 4$

idea: count edge–face incidences

Properties of Planar Graphs

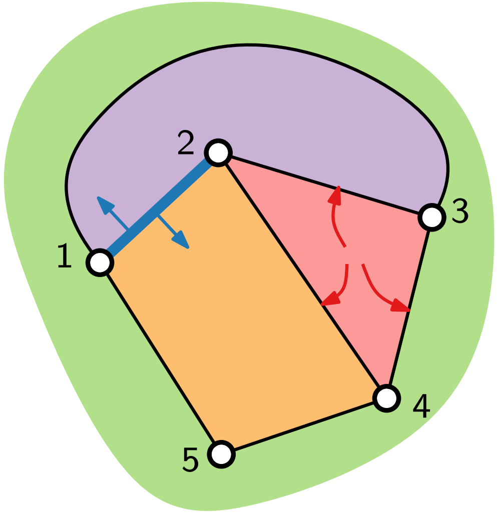
Euler's polyhedra formula.

$$\#\text{faces} - \#\text{edges} + \#\text{vertices} = \#\text{conn.comp.} + 1$$
$$f - m + n = c + 1$$

Theorem. G simple planar graph with $n \geq 3$ vtc.

1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$

3. There is a vertex of degree at most 5.



Proof. **1.** Every edge incident to ≤ 2 faces

Every face incident to ≥ 3 edges

$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$

$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$

$\Rightarrow m \leq 3n - 6$

2. $3f \leq 2m \leq 6n - 12 \Rightarrow f \leq 2n - 4$

idea: count edge–face incidences

Properties of Planar Graphs

Euler's polyhedra formula.

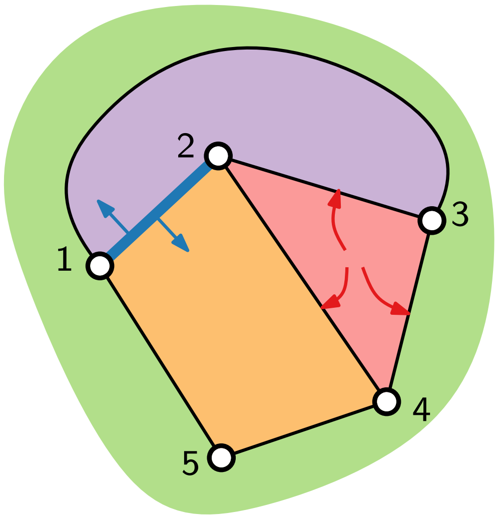
#faces − #edges + #vertices = #conn.comp. + 1

$f - m + n = c + 1$

Theorem. G simple planar graph with $n \geq 3$ vtc.

1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$

3. There is a vertex of degree at most 5.



Proof. **1.** Every edge incident to ≤ 2 faces

Every face incident to ≥ 3 edges

$\Rightarrow 3f \leq$ # incidences $\leq 2m$

$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$

$\Rightarrow m \leq 3n - 6$

idea: count edge–face incidences

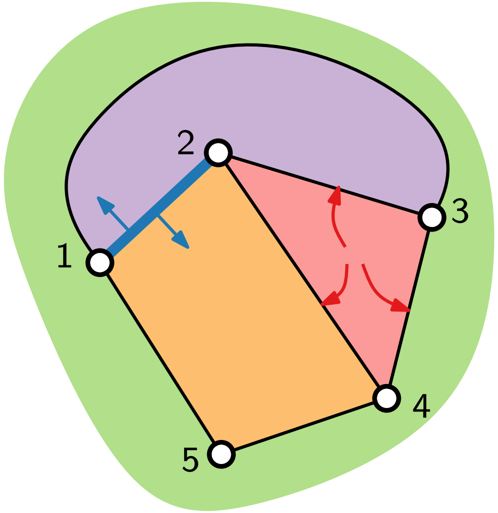
2. $3f \leq 2m \leq 6n - 12 \Rightarrow f \leq 2n - 4$

3. $\sum_{v \in V(G)} \deg(v)$

Properties of Planar Graphs

Euler's polyhedra formula.
#faces − #edges + #vertices = #conn.comp. + 1
$f - m + n = c + 1$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$
3. There is a vertex of degree at most 5.



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\text{incidences} \leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$

idea: count edge–face incidences

2. $3f \leq 2m \leq 6n - 12 \Rightarrow f \leq 2n - 4$

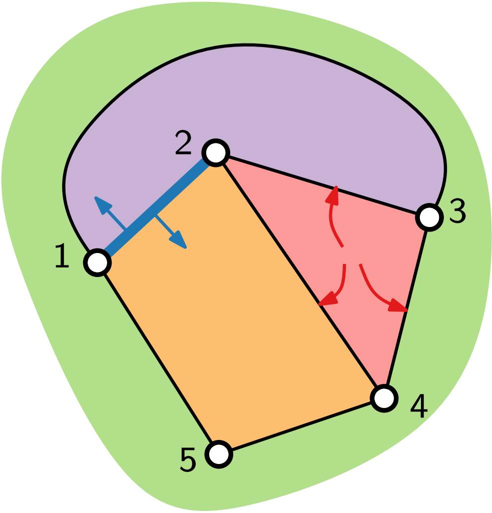
3. $\sum_{v \in V(G)} \deg(v)$

Handshaking lemma.
$\sum_{v \in V(G)} \deg(v) = 2|E(G)|$.

Properties of Planar Graphs

Euler's polyhedra formula.
#faces − #edges + #vertices = #conn.comp. + 1
$f - m + n = c + 1$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$
3. There is a vertex of degree at most 5.



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\,\text{incidences} \leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$

idea: count edge–face incidences

2. $3f \leq 2m \leq 6n - 12 \Rightarrow f \leq 2n - 4$

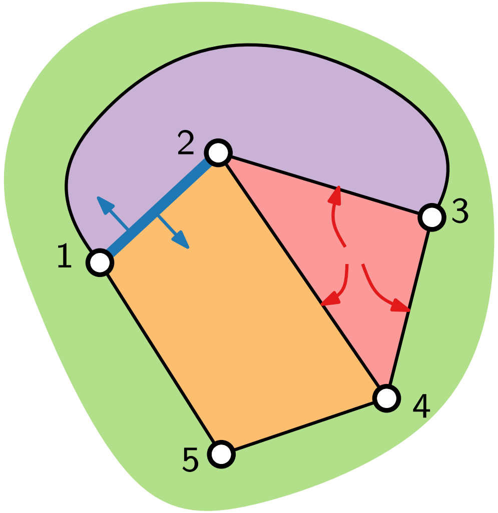
3. $\sum_{v \in V(G)} \deg(v) = 2m$

Handshaking lemma.
$\sum_{v \in V(G)} \deg(v) = 2|E(G)|$.

Properties of Planar Graphs

Euler's polyhedra formula.
#faces − #edges + #vertices = #conn.comp. + 1
$f - m + n = c + 1$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$
3. There is a vertex of degree at most 5.



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq$ # incidences $\leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$
2. $3f \leq 2m \leq 6n - 12 \Rightarrow f \leq 2n - 4$
3. $\sum_{v \in V(G)} \deg(v) = 2m \leq 6n - 12$

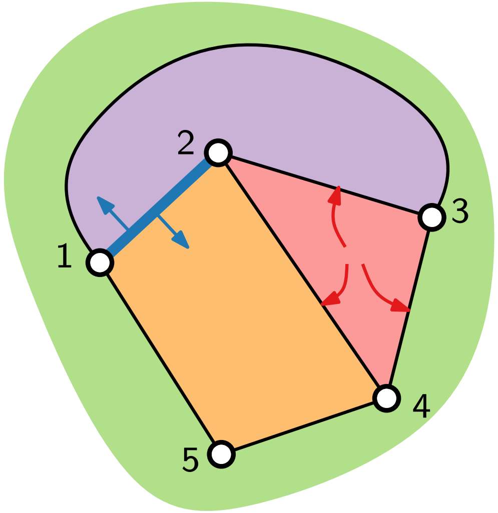
idea: count edge–face incidences

Handshaking lemma.
$\sum_{v \in V(G)} \deg(v) = 2|E(G)|$.

Properties of Planar Graphs

Euler's polyhedra formula.
#faces − #edges + #vertices = #conn.comp. + 1
$f - m + n = c + 1$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$
3. There is a vertex of degree at most 5.



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\text{incidences} \leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$

idea: count edge–face incidences

2. $3f \leq 2m \leq 6n - 12 \Rightarrow f \leq 2n - 4$

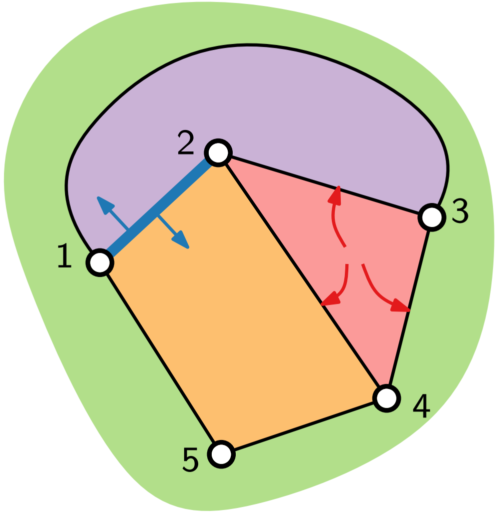
Handshaking lemma.
$\sum_{v \in V(G)} \deg(v) = 2|E(G)|$.

3. $\sum_{v \in V(G)} \deg(v) = 2m \leq 6n - 12$
$\Rightarrow \min_{v \in V(G)} \deg(v)$

Properties of Planar Graphs

Euler's polyhedra formula.
#faces − #edges + #vertices = #conn.comp. + 1
$f - m + n = c + 1$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$
3. There is a vertex of degree at most 5.



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\text{incidences} \leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$

idea: count edge–face incidences

2. $3f \leq 2m \leq 6n - 12 \Rightarrow f \leq 2n - 4$

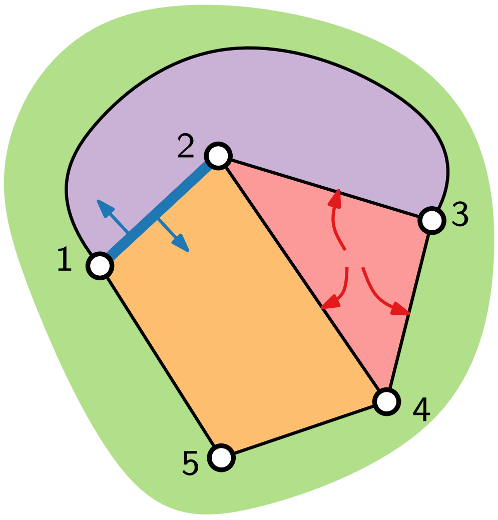
3. $\sum_{v \in V(G)} \deg(v) = 2m \leq 6n - 12$
$\Rightarrow \min_{v \in V(G)} \deg(v) \leq \text{average degree}(G)$

Handshaking lemma.
$\sum_{v \in V(G)} \deg(v) = 2|E(G)|$.

Properties of Planar Graphs

Euler's polyhedra formula.
#faces − #edges + #vertices = #conn.comp. + 1
$f - m + n = c + 1$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$
3. There is a vertex of degree at most 5.



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq \#\text{incidences} \leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$

idea: count edge–face incidences

2. $3f \leq 2m \leq 6n - 12 \Rightarrow f \leq 2n - 4$

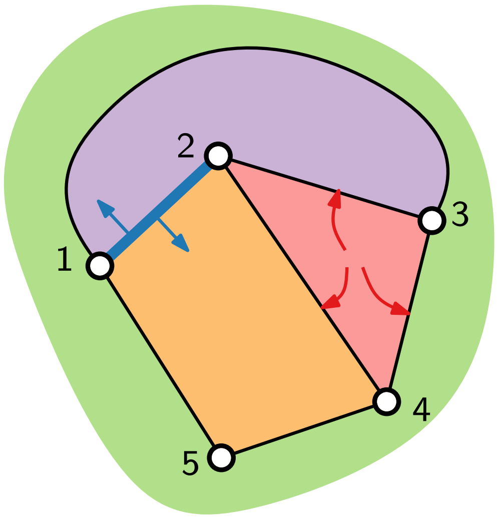
Handshaking lemma.
$\sum_{v \in V(G)} \deg(v) = 2|E(G)|$.

3. $\sum_{v \in V(G)} \deg(v) = 2m \leq 6n - 12$
$\Rightarrow \min_{v \in V(G)} \deg(v) \leq \text{average degree}(G) = \frac{1}{n} \sum_{v \in V(G)} \deg(v)$

Properties of Planar Graphs

Euler's polyhedra formula.
#faces − #edges + #vertices = #conn.comp. + 1
$f - m + n = c + 1$

Theorem. G simple planar graph with $n \geq 3$ vtc.
1. $m \leq 3n - 6$ **2.** $f \leq 2n - 4$
3. There is a vertex of degree at most 5.



Proof. **1.** Every edge incident to ≤ 2 faces
Every face incident to ≥ 3 edges
$\Rightarrow 3f \leq$ # incidences $\leq 2m$
$\Rightarrow 6 \leq 3c + 3 = 3f - 3m + 3n \leq 2m - 3m + 3n = 3n - m$
$\Rightarrow m \leq 3n - 6$

idea: count edge–face incidences

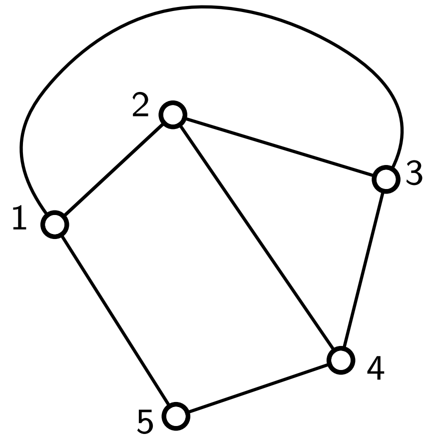
2. $3f \leq 2m \leq 6n - 12 \Rightarrow f \leq 2n - 4$

Handshaking lemma.
$\sum_{v \in V(G)} \deg(v) = 2|E(G)|$.

3. $\sum_{v \in V(G)} \deg(v) = 2m \leq 6n - 12$
$\Rightarrow \min_{v \in V(G)} \deg(v) \leq$ average degree$(G) = \frac{1}{n} \sum_{v \in V(G)} \deg(v) \leq \frac{6n-12}{n} < 6$

Triangulations

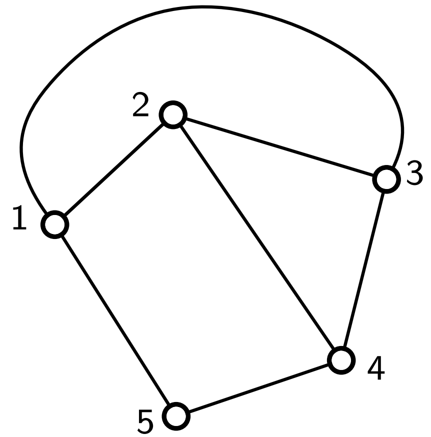
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

planar graph given with a planar embedding

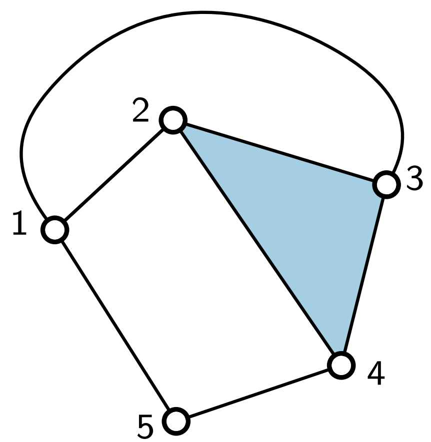
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

planar graph given with a planar embedding

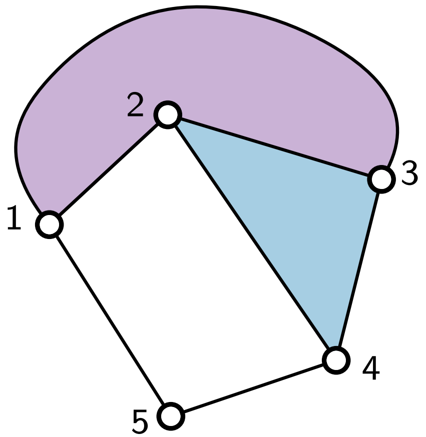
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

planar graph given with a planar embedding

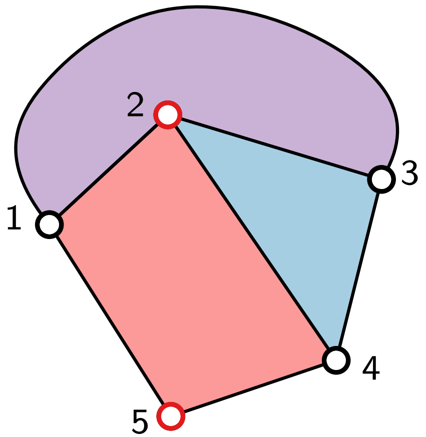
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

planar graph given with a planar embedding

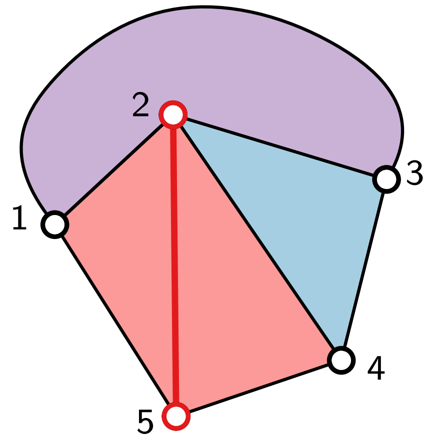
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

planar graph given with a planar embedding

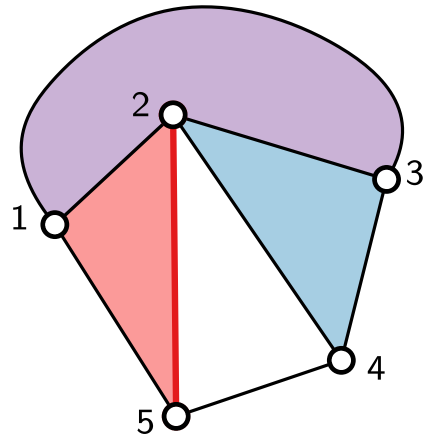
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

planar graph given with a planar embedding

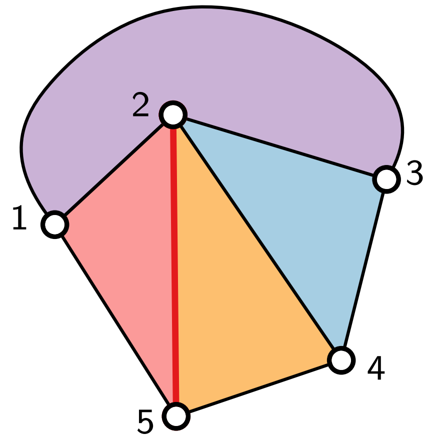
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

planar graph given with a planar embedding

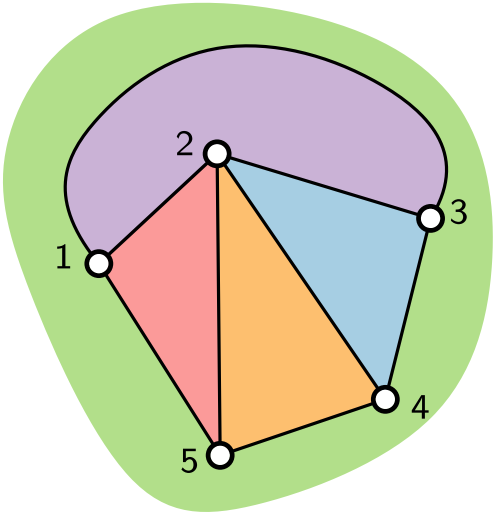
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

planar graph given with a planar embedding

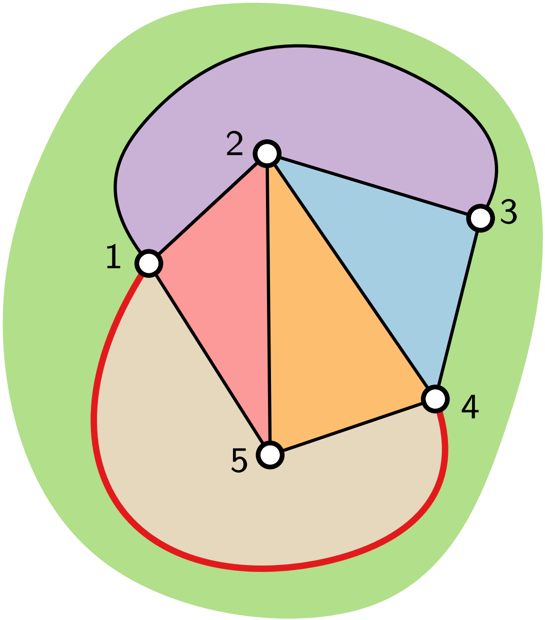
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

planar graph given with a planar embedding

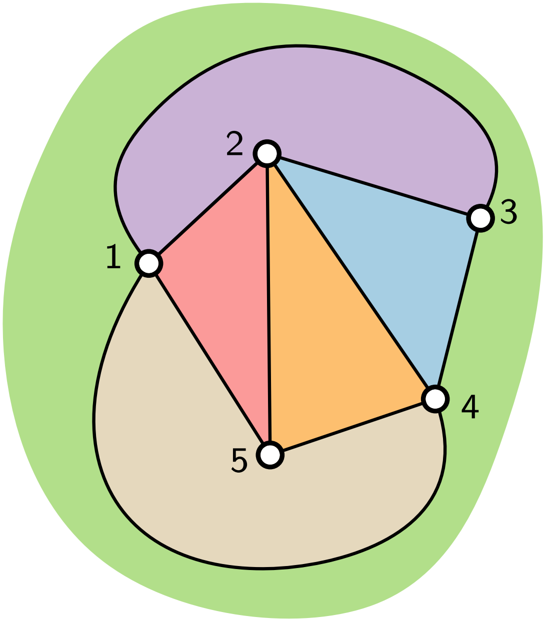
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

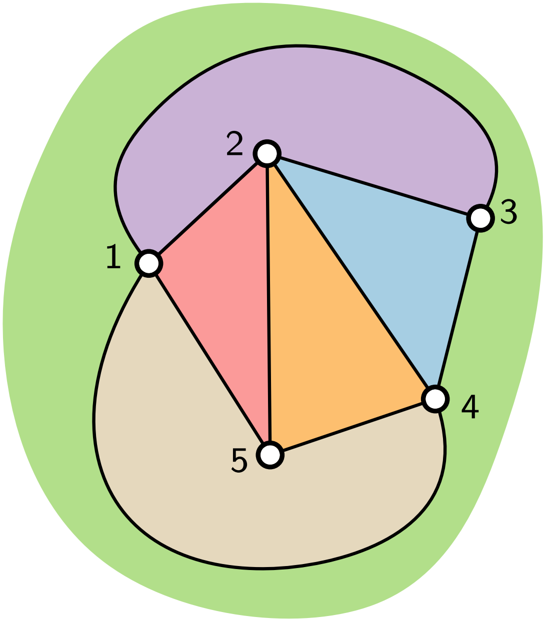


Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

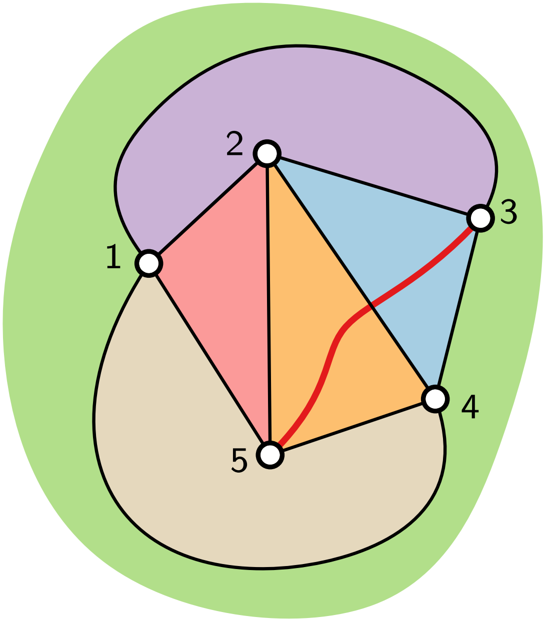


Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

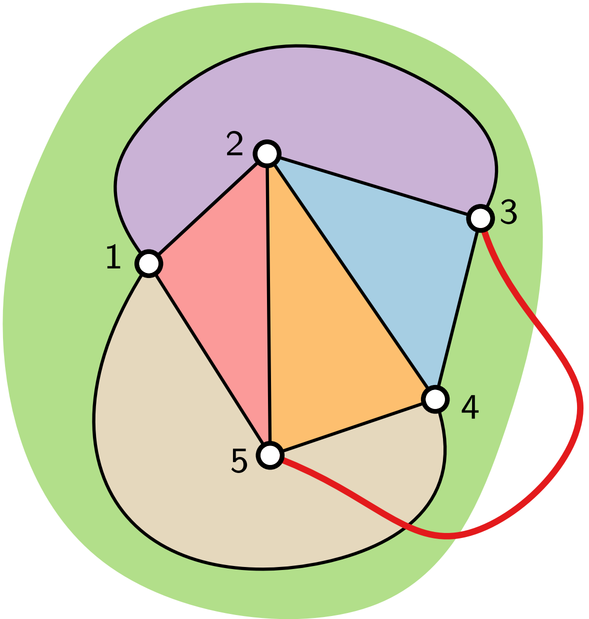


Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

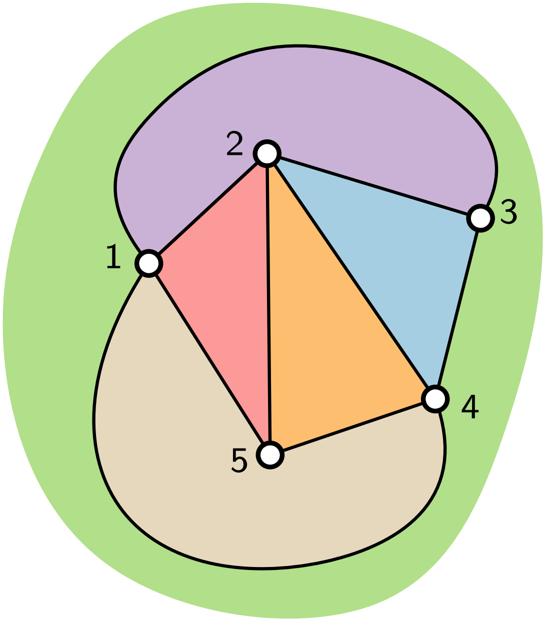


Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

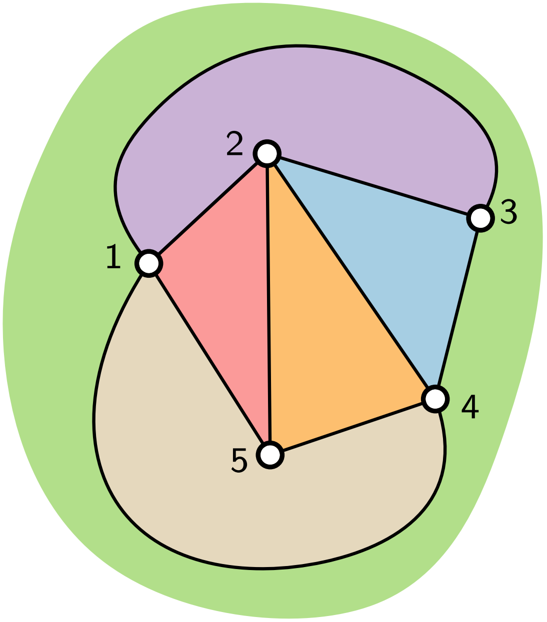


Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.



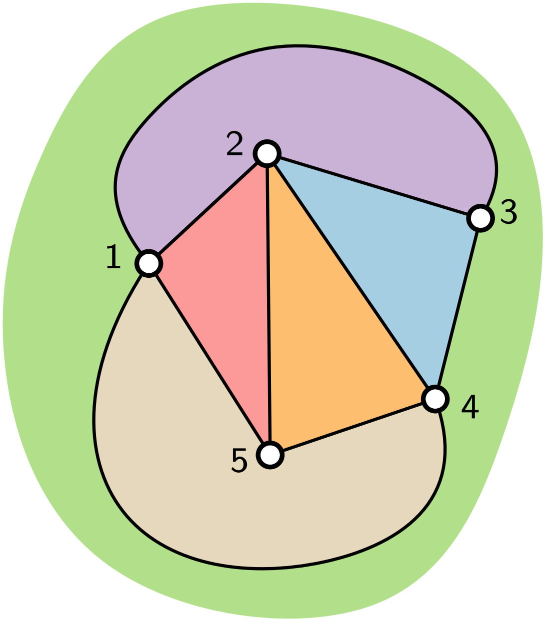
Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.



Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

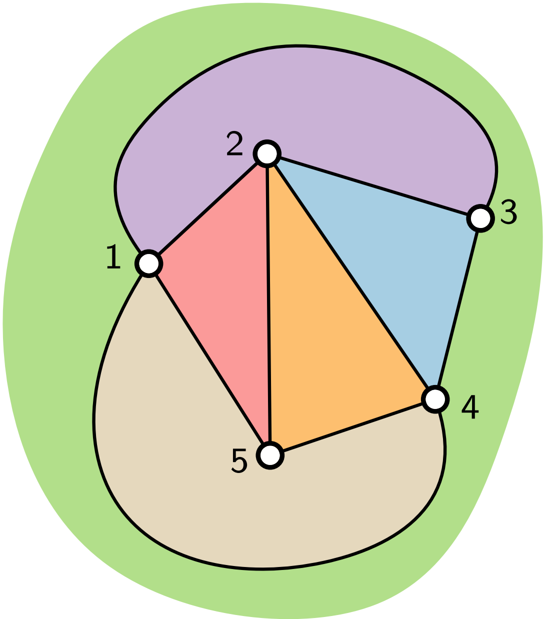
Lemma.
Any plane triangulation is 3-connected and thus has a unique planar embedding (up to mirroring).

Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.



Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

We focus on plane triangulations:

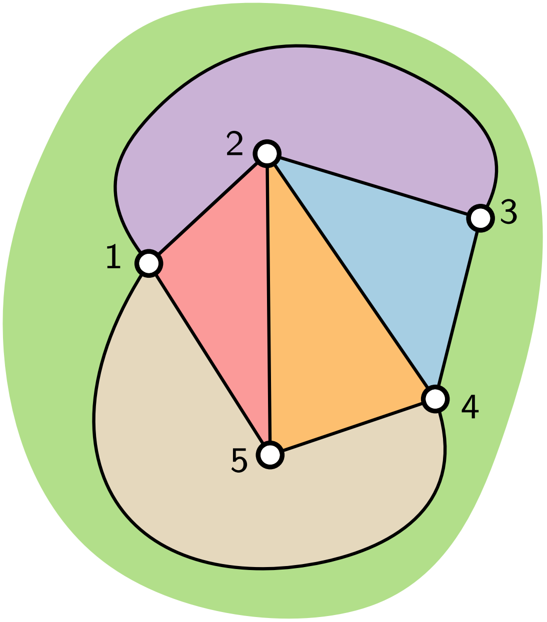
Lemma.
Any plane triangulation is 3-connected and thus has a unique planar embedding (up to mirroring).

Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.



Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

Lemma.
Any plane triangulation is 3-connected and thus has a unique planar embedding (up to mirroring).

We focus on plane triangulations:

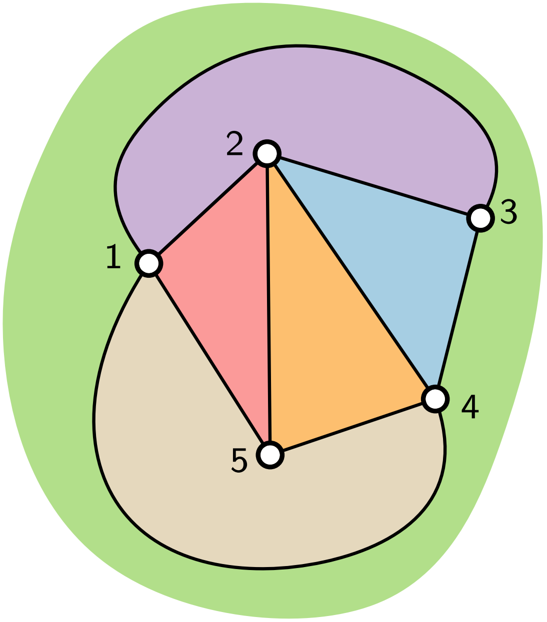
Lemma.
Every plane graph is a subgraph of a plane triangulation.

Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

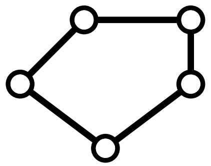


Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

Lemma.
Any plane triangulation is 3-connected and thus has a unique planar embedding (up to mirroring).

We focus on plane triangulations:

Lemma.
Every plane graph is a subgraph of a plane triangulation.

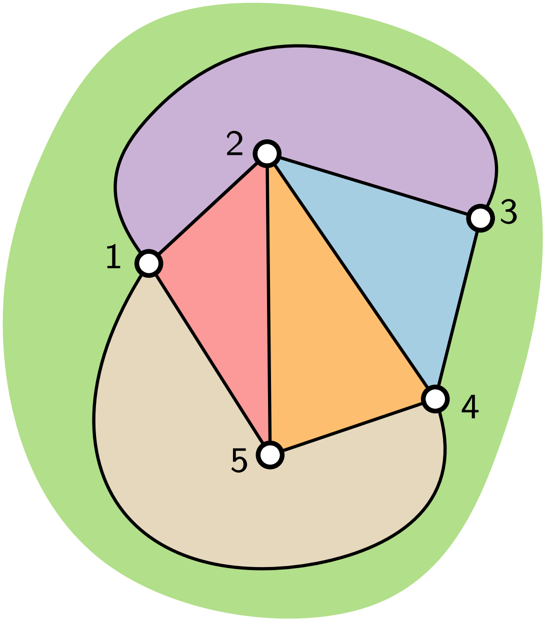


Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

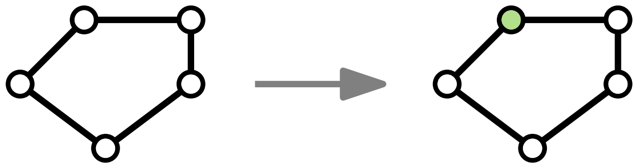


Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

Lemma.
Any plane triangulation is 3-connected and thus has a unique planar embedding (up to mirroring).

We focus on plane triangulations:

Lemma.
Every plane graph is a subgraph of a plane triangulation.

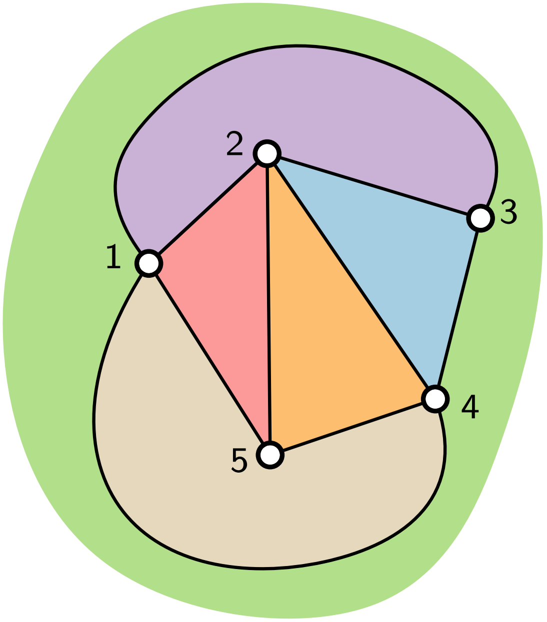


Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

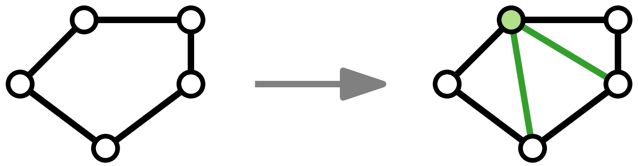


Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

Lemma.
Any plane triangulation is 3-connected and thus has a unique planar embedding (up to mirroring).

We focus on plane triangulations:

Lemma.
Every plane graph is a subgraph of a plane triangulation.



Triangulations

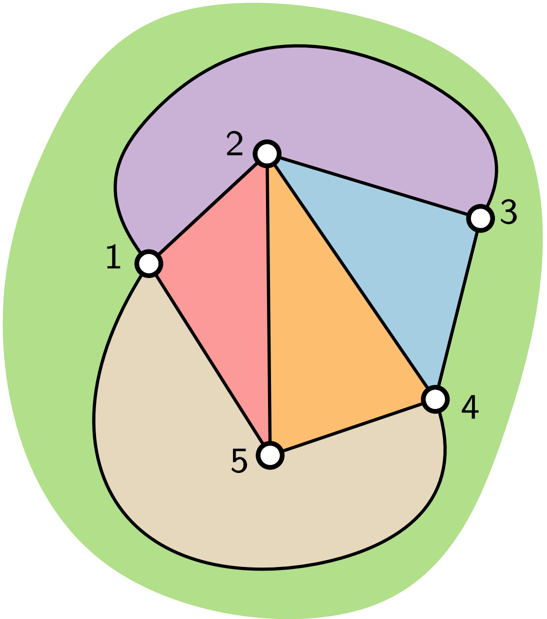
planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

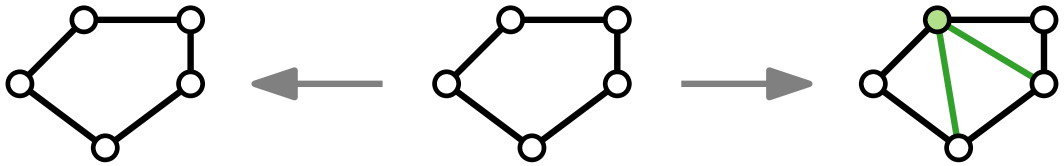
Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

Lemma.
Any plane triangulation is 3-connected and thus has a unique planar embedding (up to mirroring).



We focus on plane triangulations:

Lemma.
Every plane graph is a subgraph of a plane triangulation.

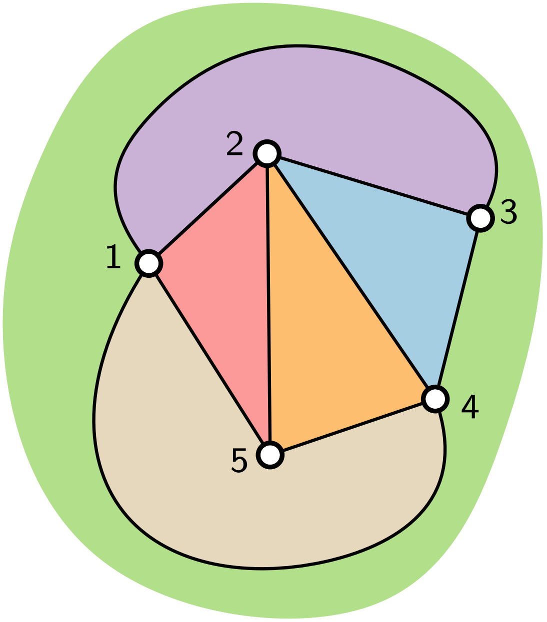


Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

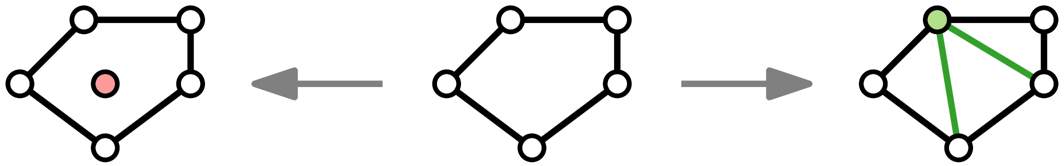


Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

Lemma.
Any plane triangulation is 3-connected and thus has a unique planar embedding (up to mirroring).

We focus on plane triangulations:

Lemma.
Every plane graph is a subgraph of a plane triangulation.



Triangulations

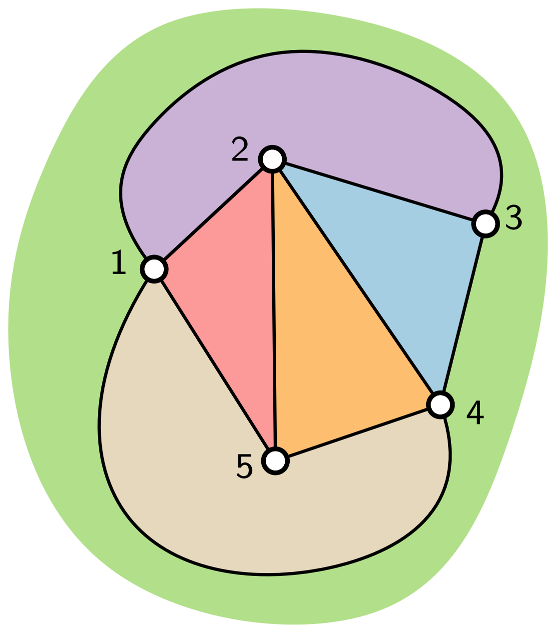
planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

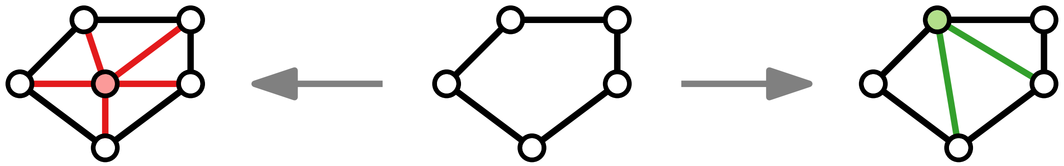
Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

Lemma.
Any plane triangulation is 3-connected and thus has a unique planar embedding (up to mirroring).



We focus on plane triangulations:

Lemma.
Every plane graph is a subgraph of a plane triangulation.

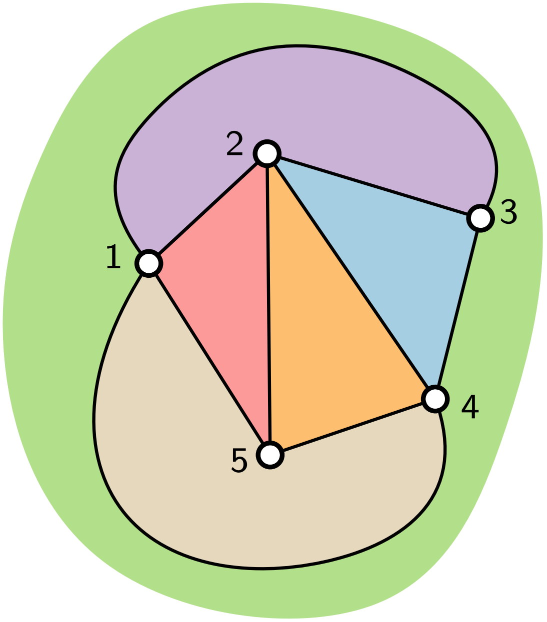


Triangulations

planar graph given with a planar embedding

A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

A **maximal planar graph** is a planar graph where adding any edge would violate planarity.

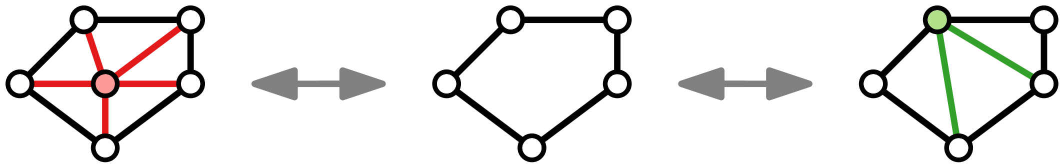


Observation.
Any maximal plane graph is a plane triangulation (and vice versa).

Lemma.
Any plane triangulation is 3-connected and thus has a unique planar embedding (up to mirroring).

We focus on plane triangulations:

Lemma.
Every plane graph is a subgraph of a plane triangulation.



Motivation

- Why planar and straight-line?

Motivation

- Why planar and straight-line?

[Bennett, Ryall, Spaltzeholz and Gooch '07]

The Aesthetics of Graph Visualization

3.2. Edge Placement Heuristics

By far the most agreed-upon edge placement heuristic is to *minimize the number of edge crossings* in a graph [BMRW98, Har98, DH96, Pur02, TR05, TBB88]. The importance of avoiding edge crossings has also been extensively validated in terms of user preference and performance (see Section 4). Similarly, based on perceptual principles, it is beneficial to *minimize the number of edge bends* within a graph [Pur02, TR05, TBB88]. Edge bends make edges more difficult to follow because an edge with a sharp bend is more likely to be perceived as two separate objects. This leads to the heuristic of *keeping edge bends uniform* with respect to the bend's position on the edge and its angle [TR05]. If an edge must be bent to satisfy other aesthetic criteria, the angle of the bend should be as little as possible, and the bend placement should evenly divide the edge.

Motivation

- Why planar and straight-line?

[Bennett, Ryall, Spaltzeholz and Gooch '07]
The Aesthetics of Graph Visualization

3.2. Edge Placement Heuristics

By far the most agreed-upon edge placement heuristic is to *minimize the number of edge crossings* in a graph [BMRW98, Har98, DH96, Pur02, TR05, TBB88]. The importance of avoiding edge crossings has also been extensively validated in terms of user preference and performance (see Section 4). Similarly, based on perceptual principles, it is beneficial to *minimize the number of edge bends* within a graph [Pur02, TR05, TBB88]. Edge bends make edges more difficult to follow because an edge with a sharp bend is more likely to be perceived as two separate objects. This leads to the heuristic of *keeping edge bends uniform* with respect to the bend's position on the edge and its angle [TR05]. If an edge must be bent to satisfy other aesthetic criteria, the angle of the bend should be as little as possible, and the bend placement should evenly divide the edge.

Motivation

- Why planar and straight-line?

[Bennett, Ryall, Spaltzeholz and Gooch '07]

The Aesthetics of Graph Visualization

3.2. Edge Placement Heuristics

By far the most agreed-upon edge placement heuristic is to *minimize the number of edge crossings* in a graph [BMRW98, Har98, DH96, Pur02, TR05, TBB88]. The importance of avoiding edge crossings has also been extensively validated in terms of user preference and performance (see Section 4). Similarly, based on perceptual principles, it is beneficial to *minimize the number of edge bends* within a graph [Pur02, TR05, TBB88]. Edge bends make edges more difficult to follow because an edge with a sharp bend is more likely to be perceived as two separate objects. This leads to the heuristic of *keeping edge bends uniform* with respect to the bend's position on the edge and its angle [TR05]. If an edge must be bent to satisfy other aesthetic criteria, the angle of the bend should be as little as possible, and the bend placement should evenly divide the edge.

Motivation

- Why planar and straight-line?

[Bennett, Ryall, Spaltzeholz and Gooch '07]

The Aesthetics of Graph Visualization

3.2. Edge Placement Heuristics

By far the most agreed-upon edge placement heuristic is to *minimize the number of edge crossings* in a graph [BMRW98, Har98, DH96, Pur02, TR05, TBB88]. The importance of avoiding edge crossings has also been extensively validated in terms of user preference and performance (see Section 4). Similarly, based on perceptual principles, it is beneficial to *minimize the number of edge bends* within a graph [Pur02, TR05, TBB88]. Edge bends make edges more difficult to follow because an edge with a sharp bend is more likely to be perceived as two separate objects. This leads to the heuristic of *keeping edge bends uniform* with respect to the bend's position on the edge and its angle [TR05]. If an edge must be bent to satisfy other aesthetic criteria, the angle of the bend should be as little as possible, and the bend placement should evenly divide the edge.

Drawing conventions

- No crossings $\Rightarrow$ planar
- No bends $\Rightarrow$ straight-line

Motivation

- Why planar and straight-line?

[Bennett, Ryall, Spaltzeholz and Gooch '07]

The Aesthetics of Graph Visualization

3.2. Edge Placement Heuristics

By far the most agreed-upon edge placement heuristic is to *minimize the number of edge crossings* in a graph [BMRW98, Har98, DH96, Pur02, TR05, TBB88]. The importance of avoiding edge crossings has also been extensively validated in terms of user preference and performance (see Section 4). Similarly, based on perceptual principles, it is beneficial to *minimize the number of edge bends* within a graph [Pur02, TR05, TBB88]. Edge bends make edges more difficult to follow because an edge with a sharp bend is more likely to be perceived as two separate objects. This leads to the heuristic of *keeping edge bends uniform* with respect to the bend's position on the edge and its angle [TR05]. If an edge must be bent to satisfy other aesthetic criteria, the angle of the bend should be as little as possible, and the bend placement should evenly divide the edge.

Drawing conventions

- No crossings $\Rightarrow$ planar
- No bends $\Rightarrow$ straight-line

Drawing aesthetics to optimize

- Area

Towards Straight-Line Drawings

Towards Straight-Line Drawings

Characterization

Towards Straight-Line Drawings

Characterization

Recognition

Towards Straight-Line Drawings

Characterization

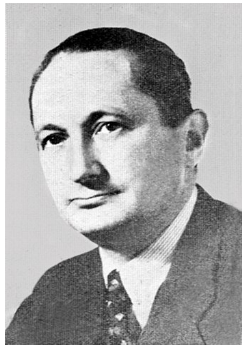
Recognition

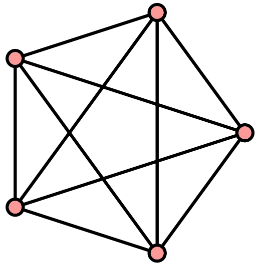
Drawing

Towards Straight-Line Drawings

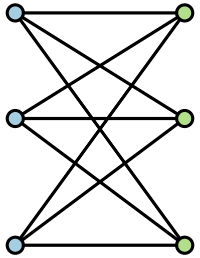
Theorem. [Kuratowski 1930]
G planar $\Leftrightarrow$
neither K_5 nor $K_{3,3}$ minor of G

Kazimierz Kuratowski (1896–1980)





K_5



$K_{3,3}$

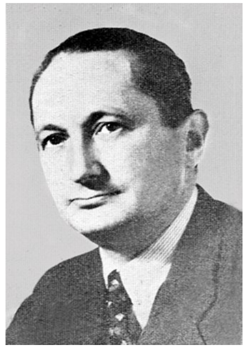
Characterization

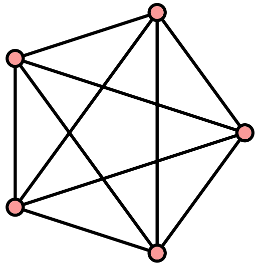
Recognition

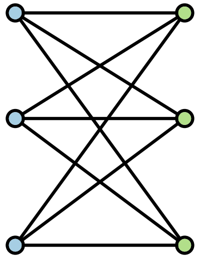
Drawing

Towards Straight-Line Drawings

Theorem. [Kuratowski 1930]
G planar $\Leftrightarrow$
neither K_5 nor $K_{3,3}$ minor of G

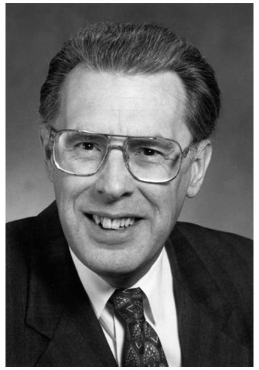

Kazimierz Kuratowski (1896–1980)


K_5


$K_{3,3}$

Characterization

Theorem. [Hopcroft & Tarjan 1974]
Let G be a graph with n vertices. There is an $\mathcal{O}(n)$-time algorithm to test whether G is planar.


John Edward Hopcroft (1939–)
en.wikipedia.org/wiki/User:Shakespeare

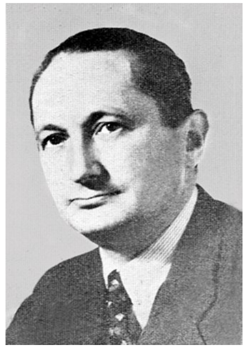

Robert Endre Tarjan (1948–)
Renatokeshet, GFDL via Wikimedia

Recognition

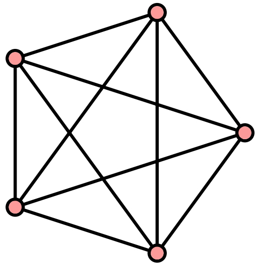
Drawing

Towards Straight-Line Drawings

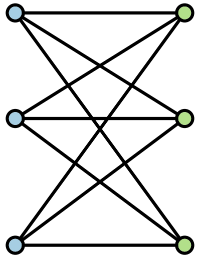
Theorem. [Kuratowski 1930]
G planar $\Leftrightarrow$
neither K_5 nor $K_{3,3}$ minor of G



Kazimierz Kuratowski (1896–1980)



K_5

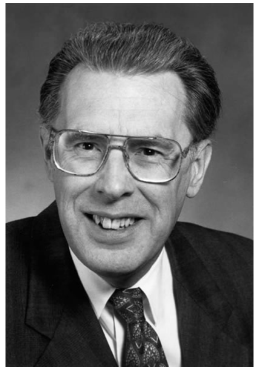


$K_{3,3}$

Characterization

Theorem. [Hopcroft & Tarjan 1974]
Let G be a graph with n vertices. There is an $\mathcal{O}(n)$-time algorithm to test whether G is planar.

Also computes a planar embedding in $\mathcal{O}(n)$ time.



John Edward Hopcroft (1939–)
en.wikipedia.org/wiki/User:Shakespeare

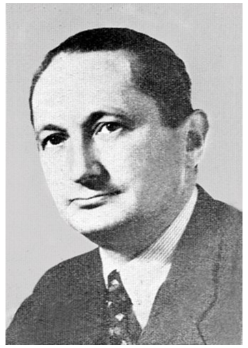
Robert Endre Tarjan (1948–)
Renatokeshet, GFDL via Wikimedia

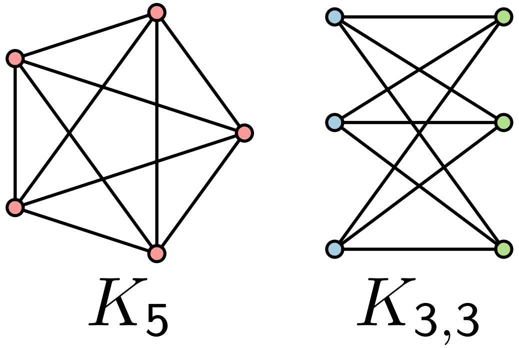
Recognition

Drawing

Towards Straight-Line Drawings

Theorem. [Kuratowski 1930]
G planar $\Leftrightarrow$
neither K_5 nor $K_{3,3}$ minor of G

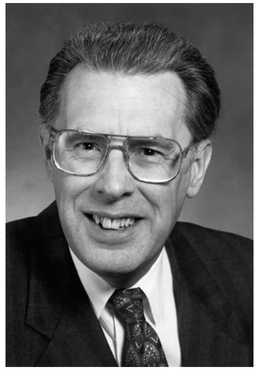

Kazimierz Kuratowski (1896–1980)



Characterization

Theorem. [Hopcroft & Tarjan 1974]
Let G be a graph with n vertices. There is an $\mathcal{O}(n)$-time algorithm to test whether G is planar.

Also computes a planar embedding in $\mathcal{O}(n)$ time.


John Edward Hopcroft (1939–)
en.wikipedia.org/wiki/User:Shakespeare


Robert Endre Tarjan (1948–)
Renatokeshet, GFDL via Wikimedia

Recognition

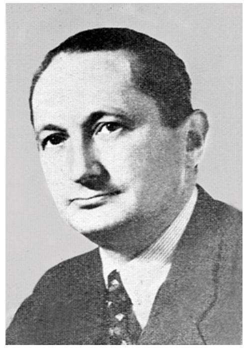
Theorem. [Wagner 1936, Fáry 1948, Stein 1951]
Every planar graph has a planar drawing where the edges are straight-line segments.


Klaus Wagner (1910–2000)
Autor: Konrad Jacobs, wikipedia

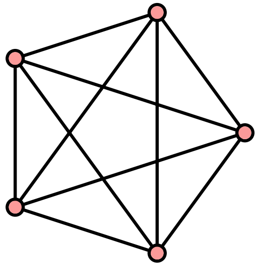
Drawing

Towards Straight-Line Drawings

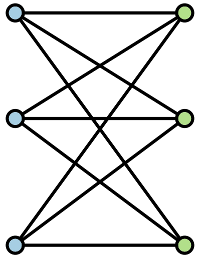
Theorem. [Kuratowski 1930]
G planar $\Leftrightarrow$
neither K_5 nor $K_{3,3}$ minor of G



Kazimierz Kuratowski (1896–1980)



K_5

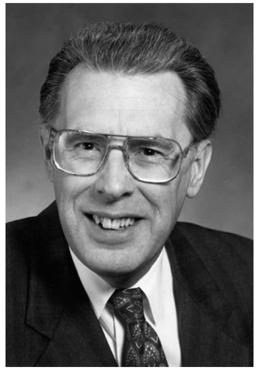


$K_{3,3}$

Characterization

Theorem. [Hopcroft & Tarjan 1974]
Let G be a graph with n vertices. There is an $\mathcal{O}(n)$-time algorithm to test whether G is planar.

Also computes a planar embedding in $\mathcal{O}(n)$ time.



John Edward Hopcroft (1939–)
en.wikipedia.org/wiki/User:Shakespeare

Robert Endre Tarjan (1948–)
Renatokeshet, GFDL via Wikimedia

Recognition

Theorem. [Wagner 1936, Fáry 1948, Stein 1951]
Every planar graph has a planar drawing where the edges are straight-line segments.

The algorithms implied by these theorems produce drawings whose area is **not** bounded by any polynomial in n.

Klaus Wagner (1910–2000)
Autor: Konrad Jacobs, wikipedia

Drawing

Planar Straight-Line Drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$.

Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

Planar Straight-Line Drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$.

Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

(next lecture)

Planar Straight-Line Drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n - 4) \times (n - 2)$.

Idea.

- Find a *canonical order* $(v_1, \ldots, v_n)$ of the vertices of a triangulation.

Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(n - 2) \times (n - 2)$.

(next lecture)

Planar Straight-Line Drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$.

Idea.

- Find a *canonical order* $(v_1, \ldots, v_n)$ of the vertices of a triangulation.
- Start with the single edge (v_1, v_2). Let this be the graph G_2.

Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

(next lecture)

Planar Straight-Line Drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$.

Idea.

- Find a *canonical order* $(v_1, \ldots, v_n)$ of the vertices of a triangulation.
- Start with the single edge (v_1, v_2). Let this be the graph G_2.
- Let $k \in \{3, \ldots, n\}$. To obtain G_{k+1}, add v_{k+1} to G_k so that the neighbors of v_{k+1} are on the outer face of G_k.

Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

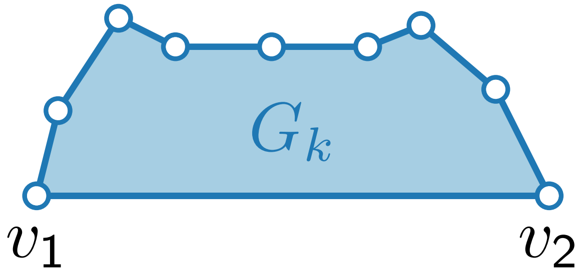
(next lecture)

Planar Straight-Line Drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$.

Idea.

- Find a *canonical order* $(v_1, \ldots, v_n)$ of the vertices of a triangulation.
- Start with the single edge (v_1, v_2). Let this be the graph G_2.
- Let $k \in \{3, \ldots, n\}$. To obtain G_{k+1}, add v_{k+1} to G_k so that the neighbors of v_{k+1} are on the outer face of G_k.



Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

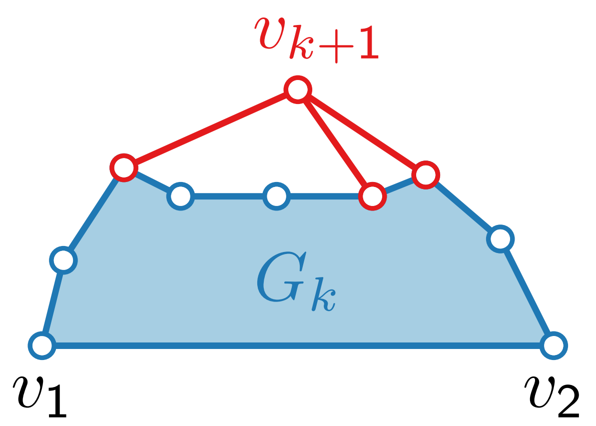
(next lecture)

Planar Straight-Line Drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$.

Idea.

- Find a *canonical order* $(v_1, \ldots, v_n)$ of the vertices of a triangulation.
- Start with the single edge (v_1, v_2). Let this be the graph G_2.
- Let $k \in \{3, \ldots, n\}$. To obtain G_{k+1}, add v_{k+1} to G_k so that the neighbors of v_{k+1} are on the outer face of G_k.



Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

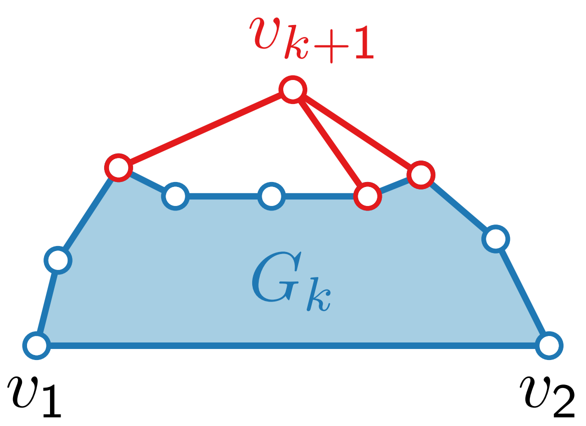
(next lecture)

Planar Straight-Line Drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$.

Idea.

- Find a *canonical order* $(v_1, \ldots, v_n)$ of the vertices of a triangulation.
- Start with the single edge (v_1, v_2). Let this be the graph G_2.
- Let $k \in \{3, \ldots, n\}$. To obtain G_{k+1}, add v_{k+1} to G_k so that the neighbors of v_{k+1} are on the outer face of G_k.
- The neighbors of v_{k+1} in G_k form a path of length at least two.



Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

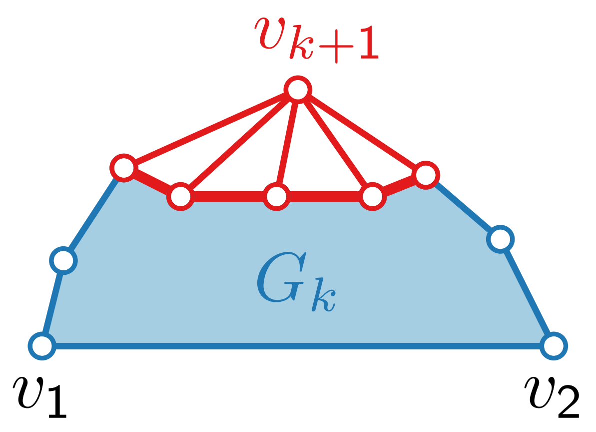
(next lecture)

Planar Straight-Line Drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$.

Idea.

- Find a *canonical order* $(v_1, \dots, v_n)$ of the vertices of a triangulation.
- Start with the single edge (v_1, v_2). Let this be the graph G_2.
- Let $k \in \{3, \dots, n\}$. To obtain G_{k+1}, add v_{k+1} to G_k so that the neighbors of v_{k+1} are on the outer face of G_k.
- The neighbors of v_{k+1} in G_k form a path of length at least two.



Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

(next lecture)

Canonical Order – Definition

Definition.
Let G be a plane triangulation on $n \geq 3$ vertices.

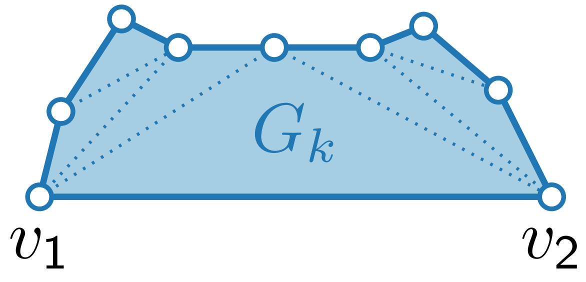
Canonical Order – Definition

Definition.
Let G be a plane triangulation on $n \geq 3$ vertices.
An ordering $\pi = (v_1, v_2, \ldots, v_n)$ of $V(G)$ is a **canonical order**
if the following conditions hold for each $k \in \{3, 4, \ldots, n\}$:

Canonical Order – Definition

Definition.
Let G be a plane triangulation on $n \geq 3$ vertices.
An ordering $\pi = (v_1, v_2, \ldots, v_n)$ of $V(G)$ is a **canonical order** if the following conditions hold for each $k \in \{3, 4, \ldots, n\}$:

(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

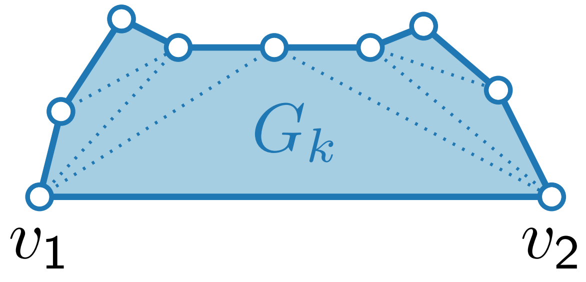


Canonical Order – Definition

Definition.
Let G be a plane triangulation on $n \geq 3$ vertices.
An ordering $\pi = (v_1, v_2, \ldots, v_n)$ of $V(G)$ is a **canonical order** if the following conditions hold for each $k \in \{3, 4, \ldots, n\}$:

(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.



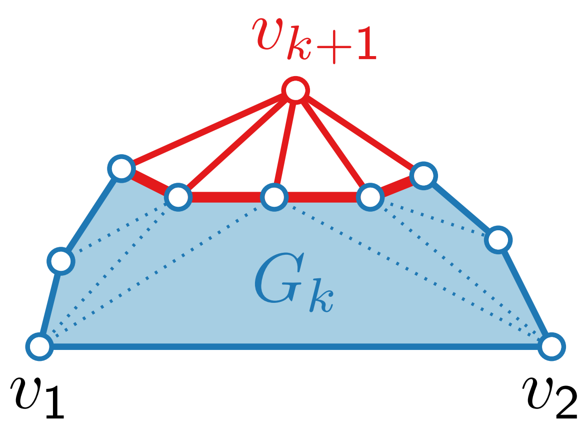
Canonical Order – Definition

Definition.
Let G be a plane triangulation on $n \geq 3$ vertices.
An ordering $\pi = (v_1, v_2, \ldots, v_n)$ of $V(G)$ is a **canonical order** if the following conditions hold for each $k \in \{3, 4, \ldots, n\}$:

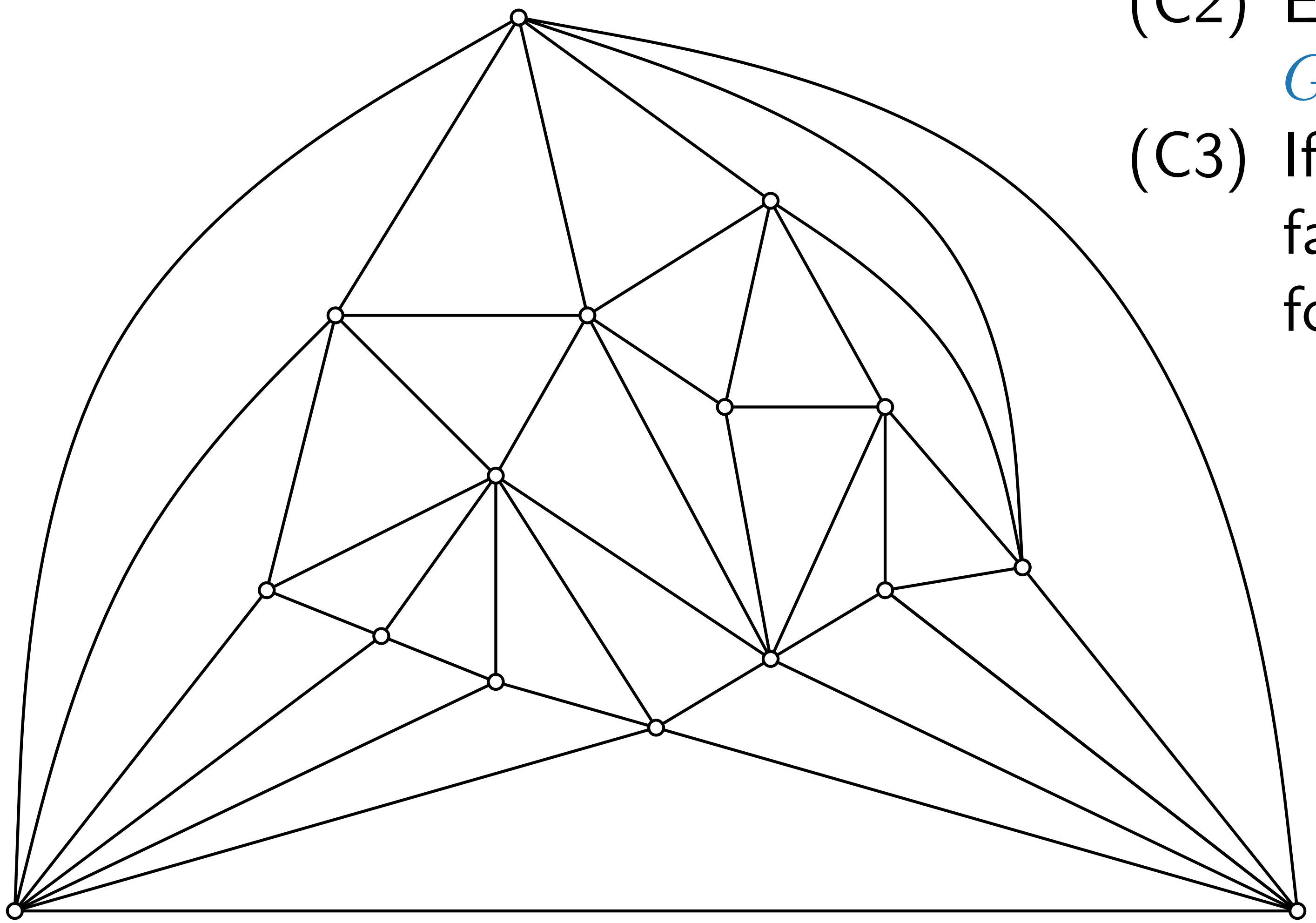
(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.



Canonical Order – Example

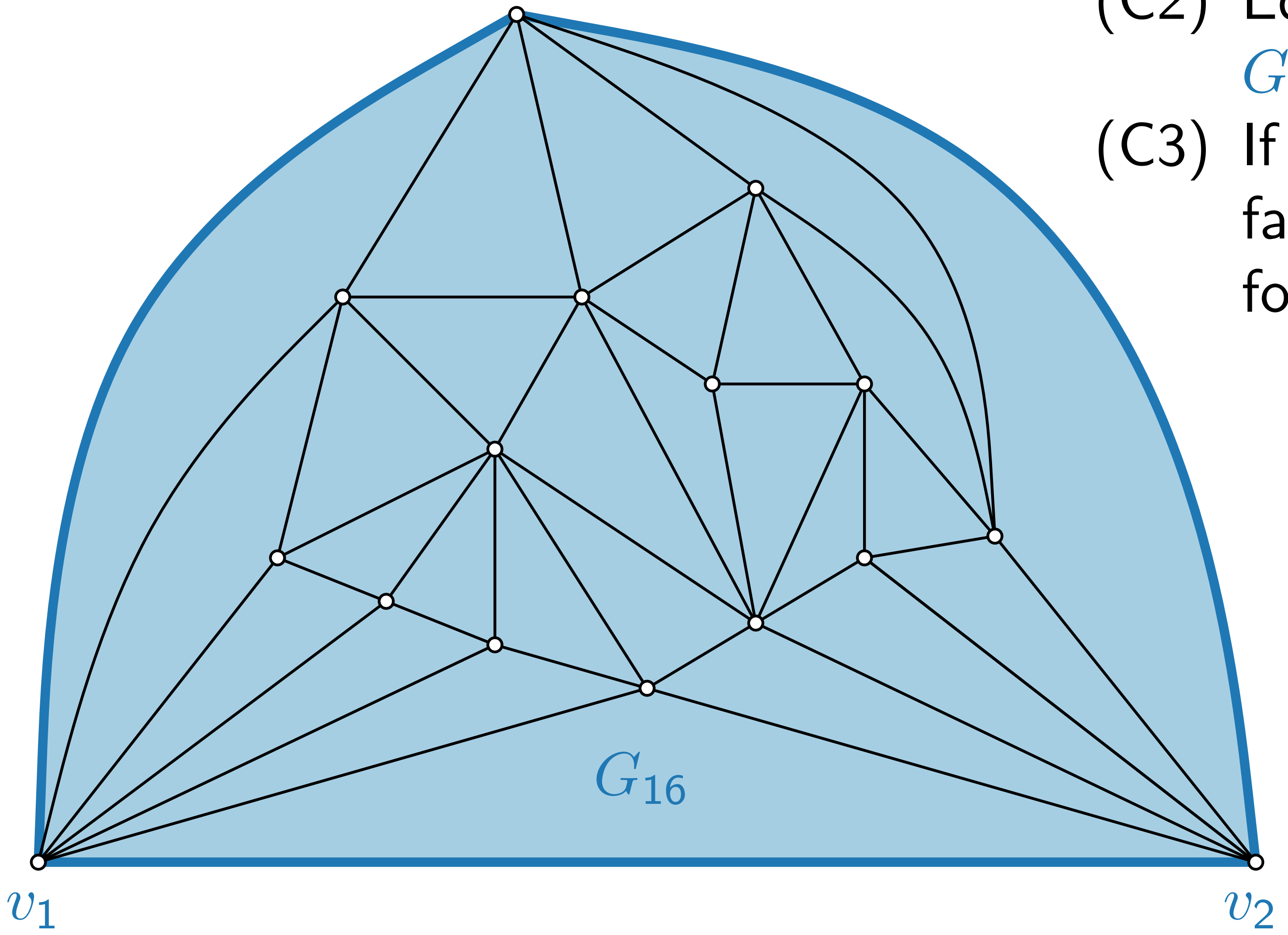


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

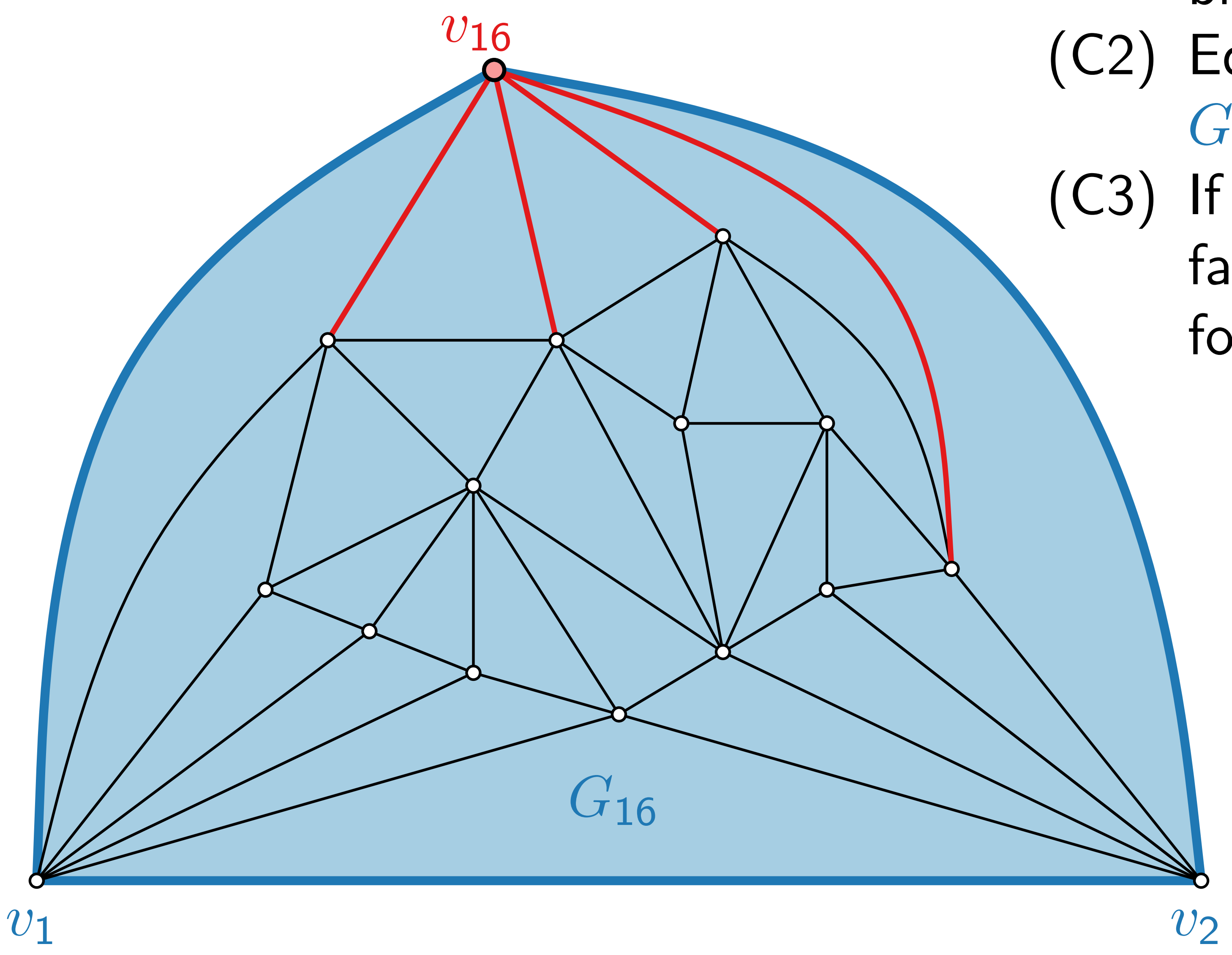
(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example



(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.
(C2) Edge (v_1, v_2) belongs to the outer face of G_k.
(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

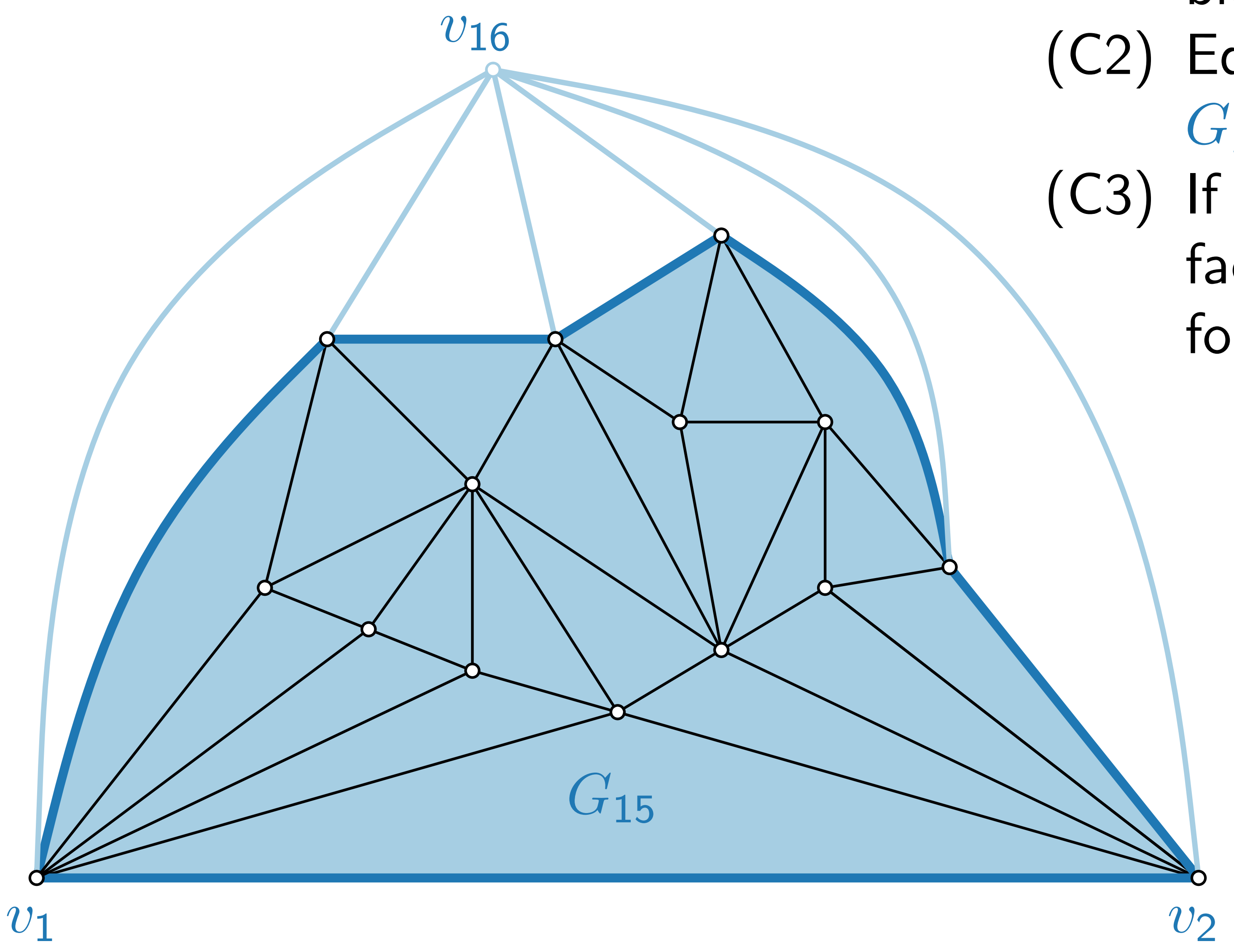


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

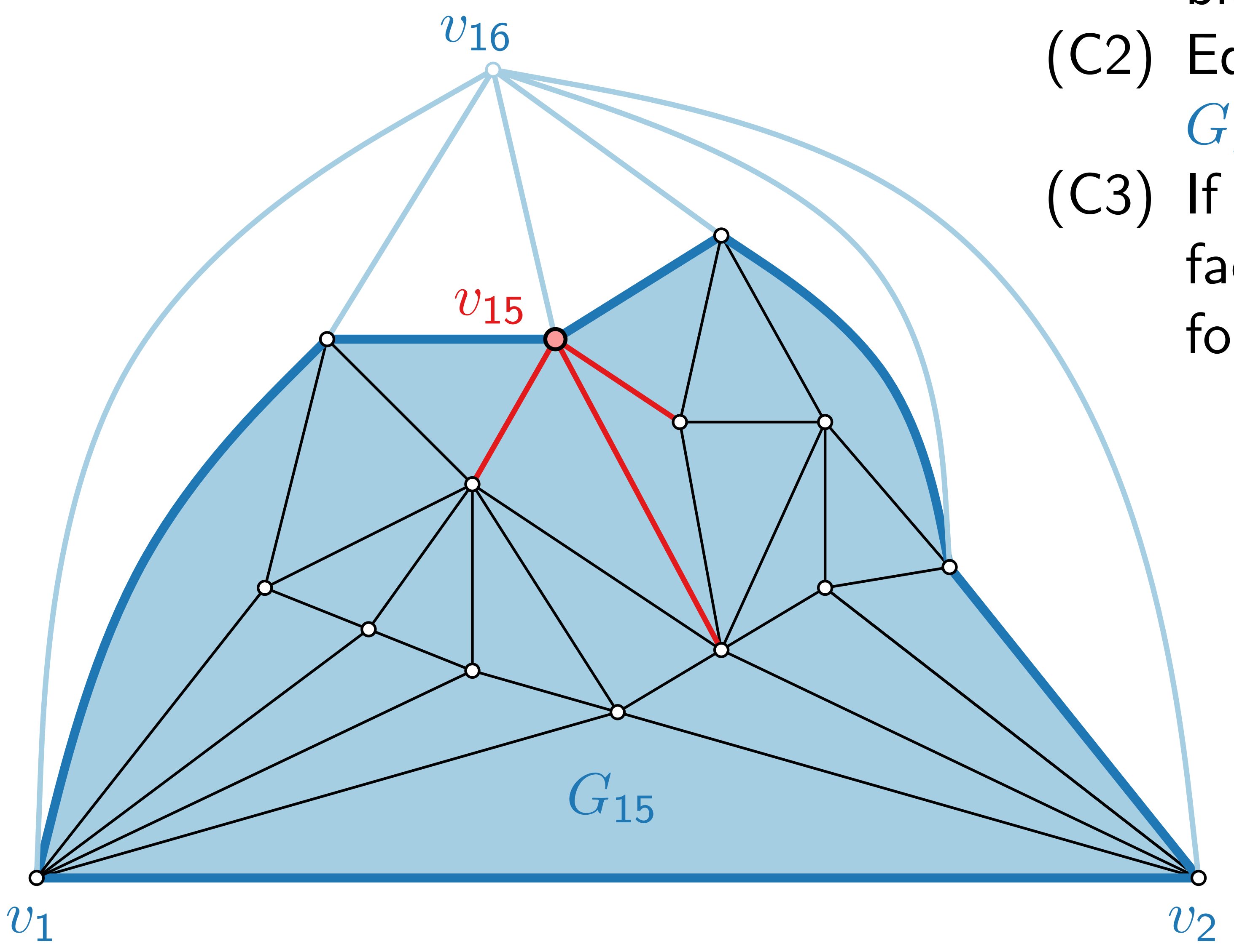


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

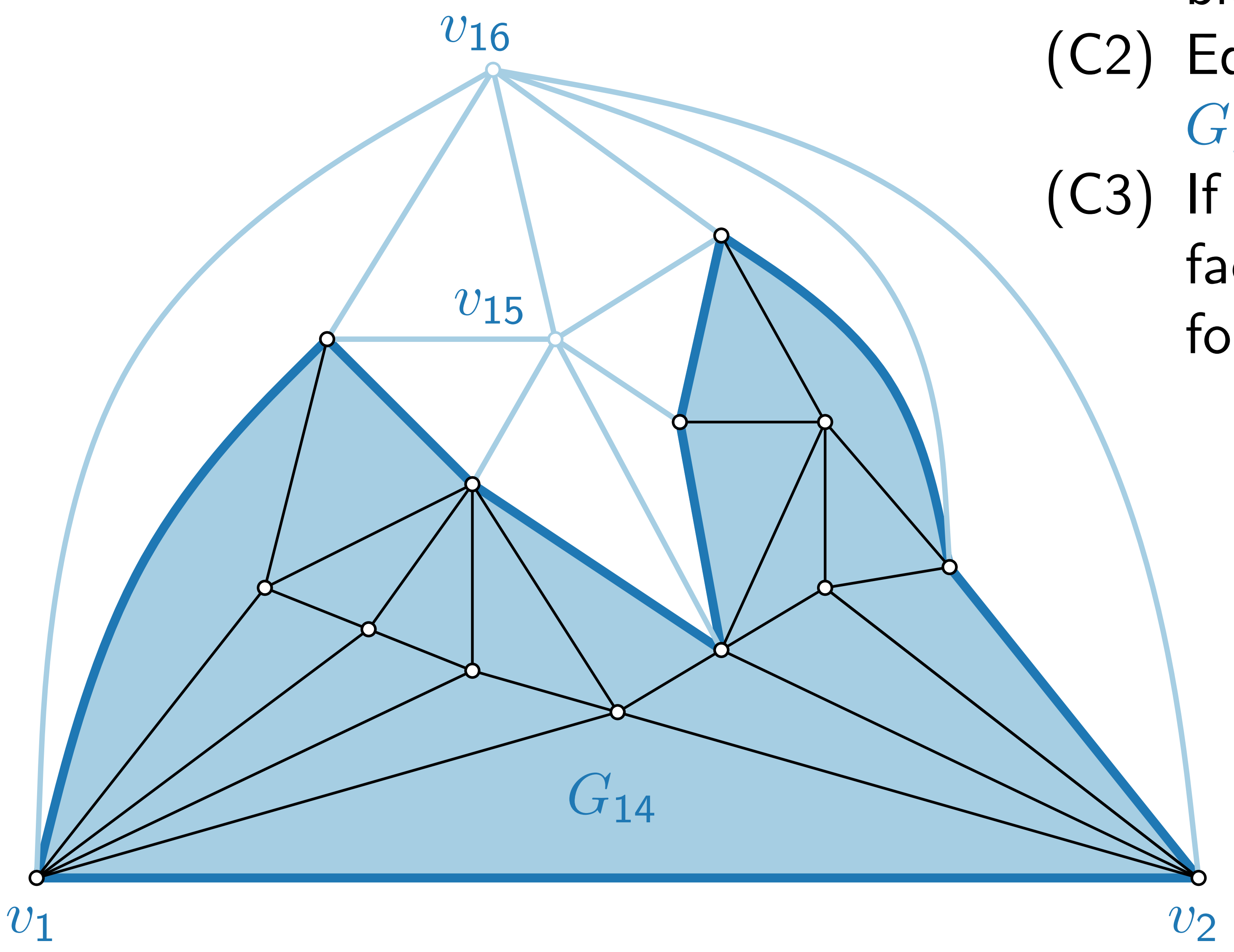


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

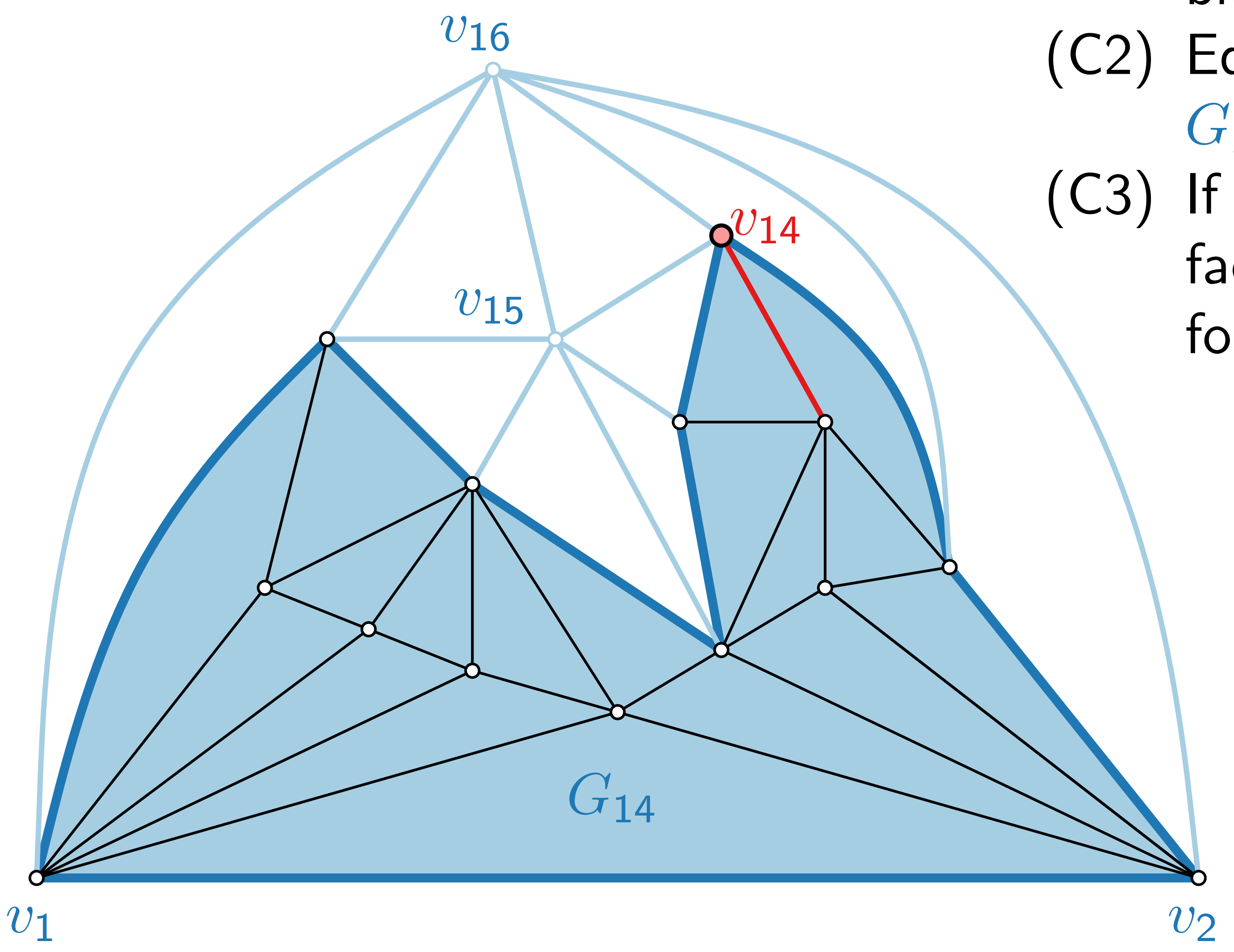


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

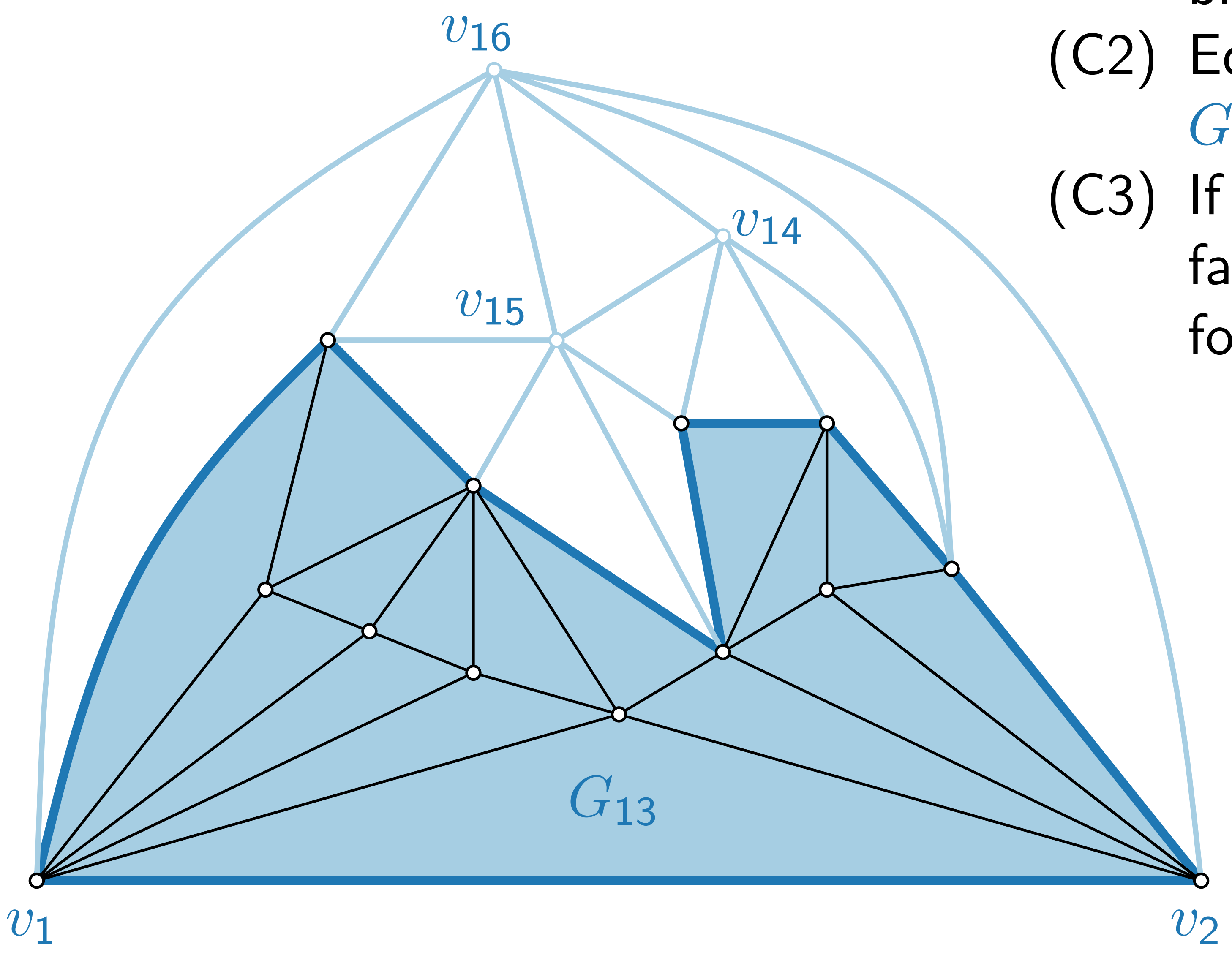


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

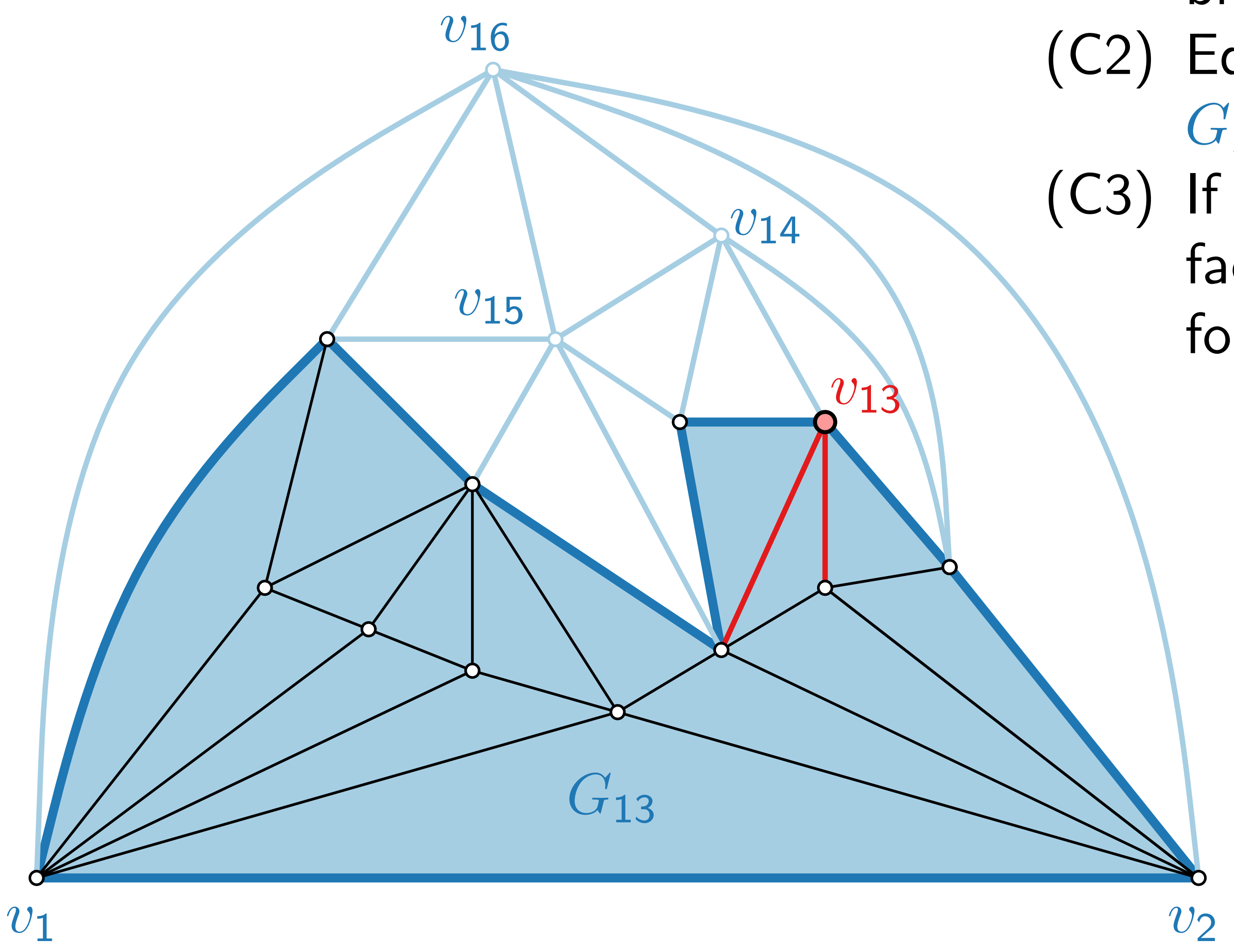


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

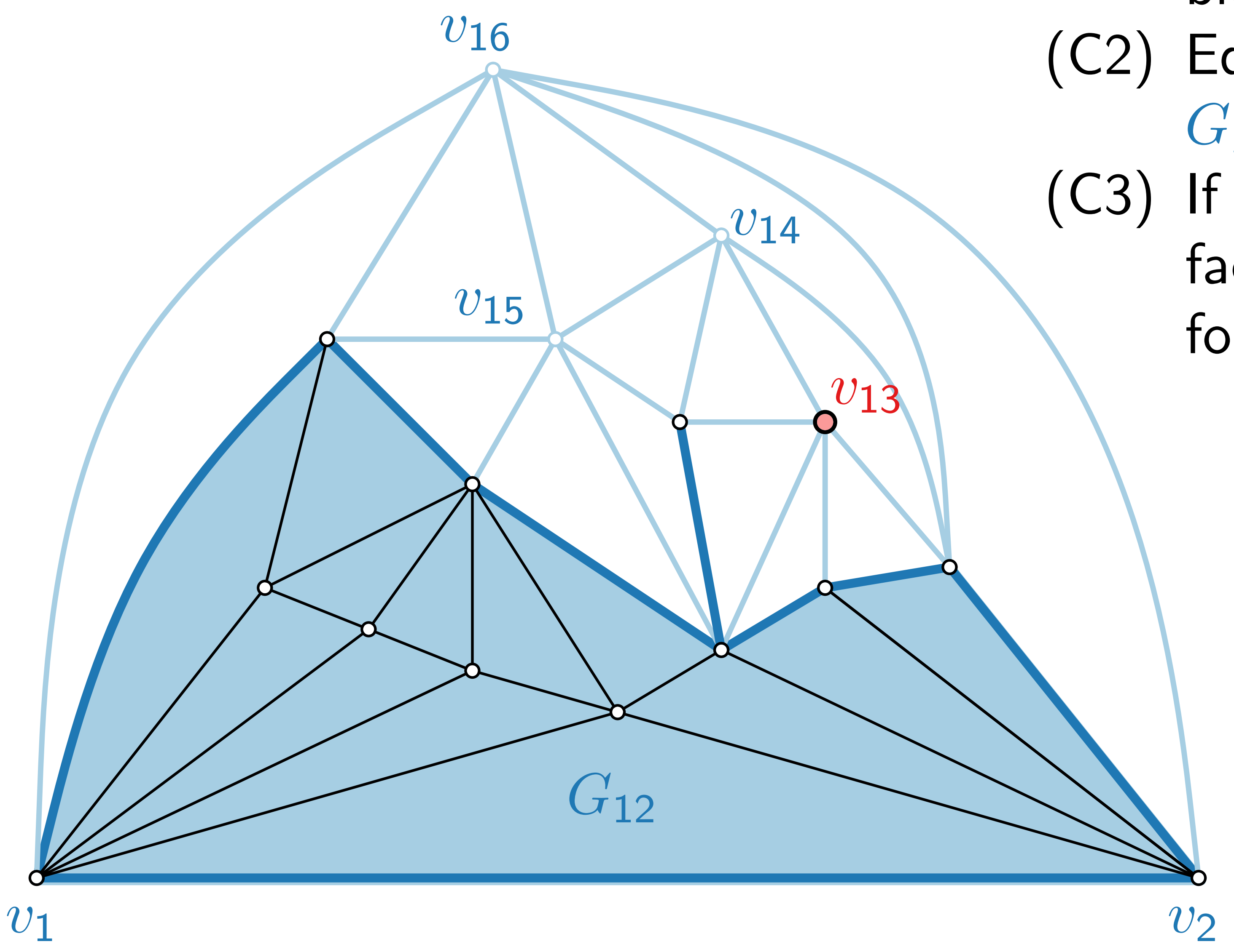


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

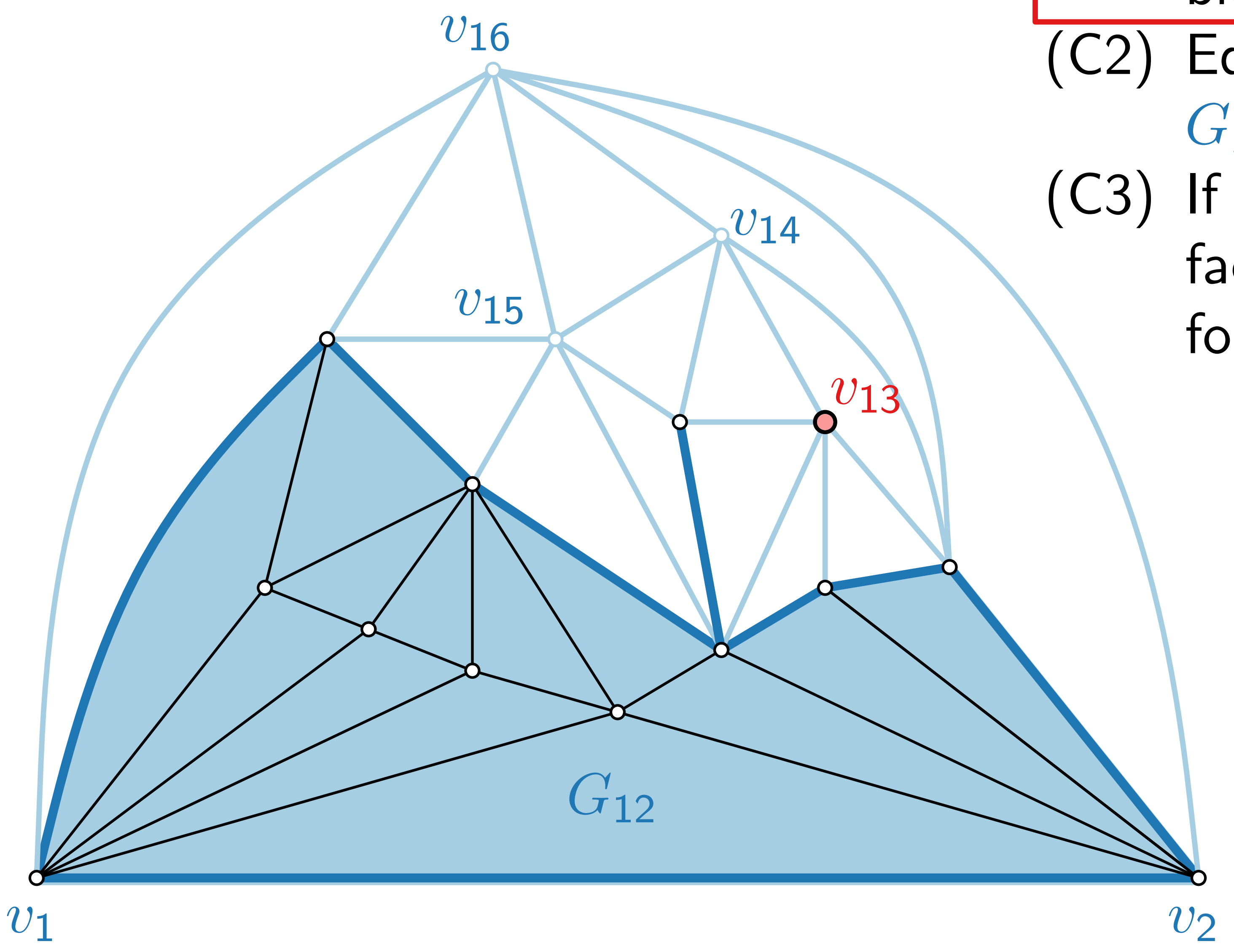


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

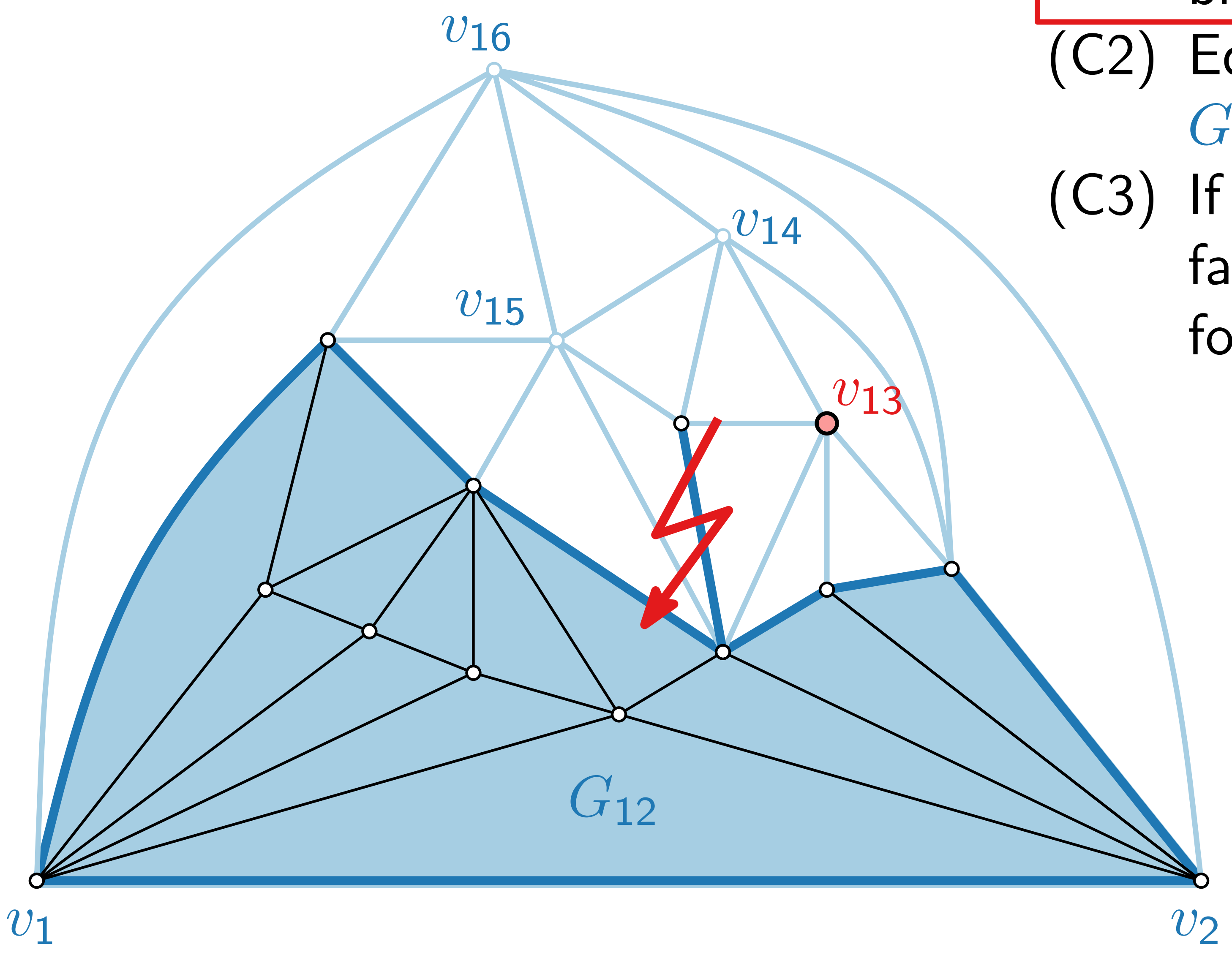


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

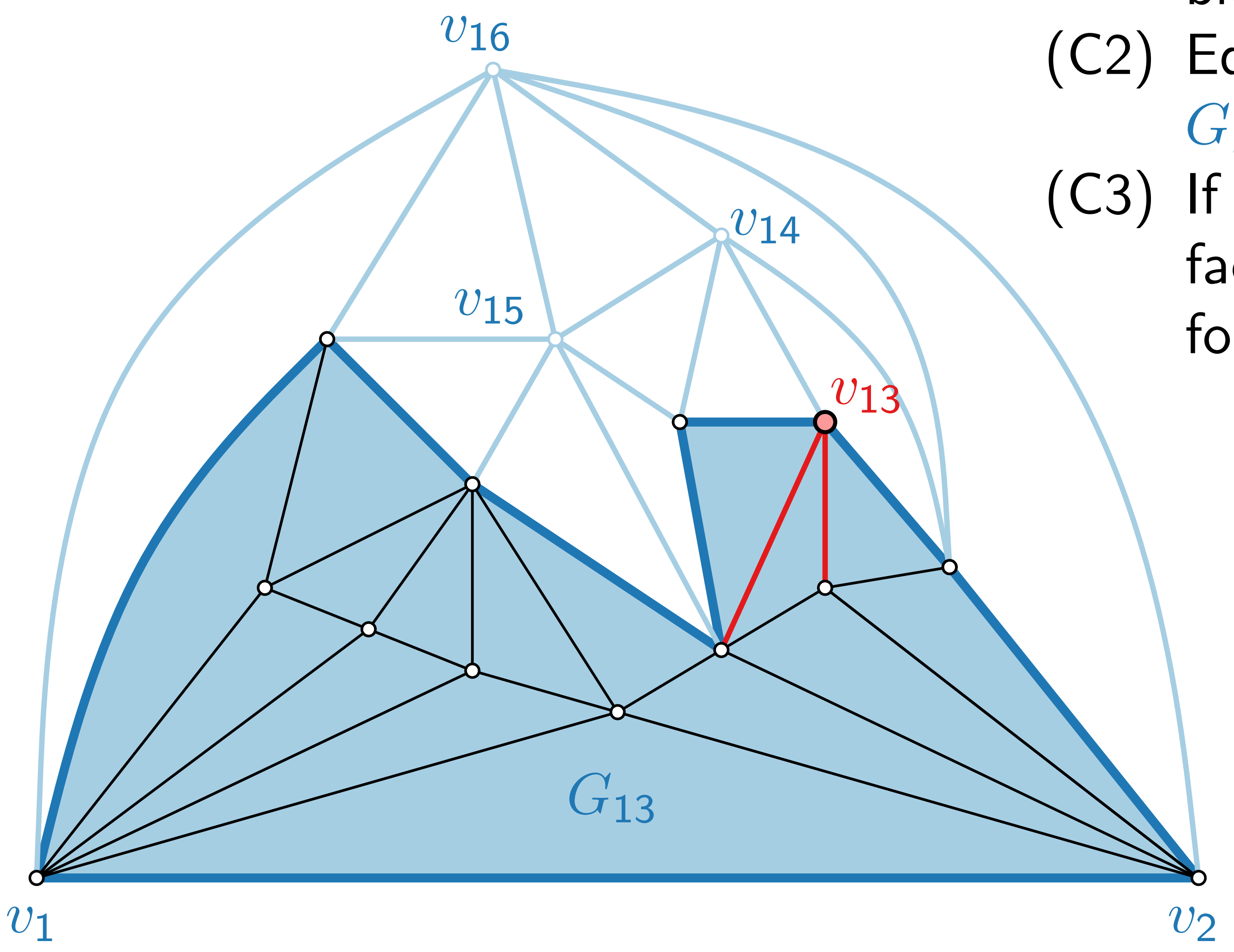


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

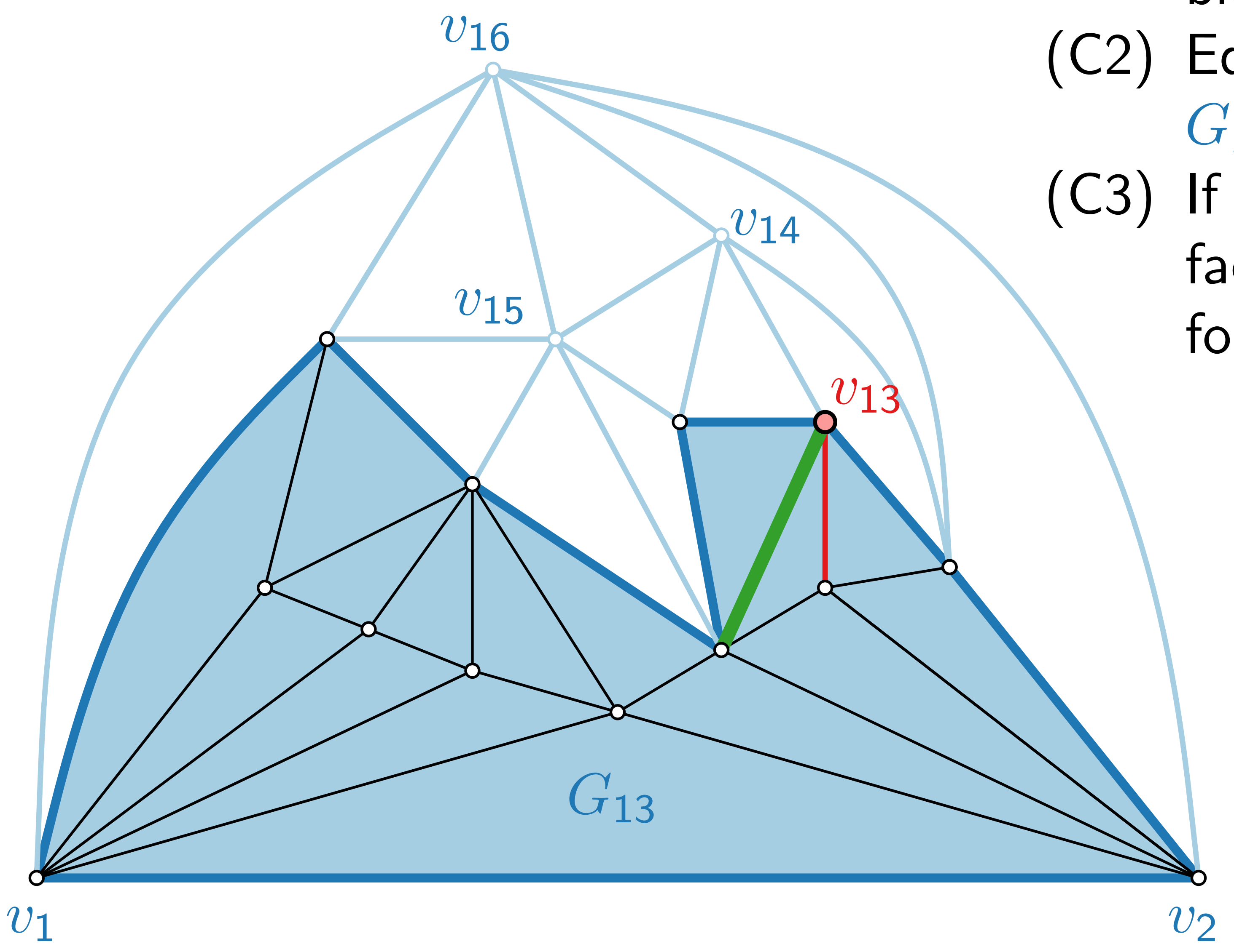


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

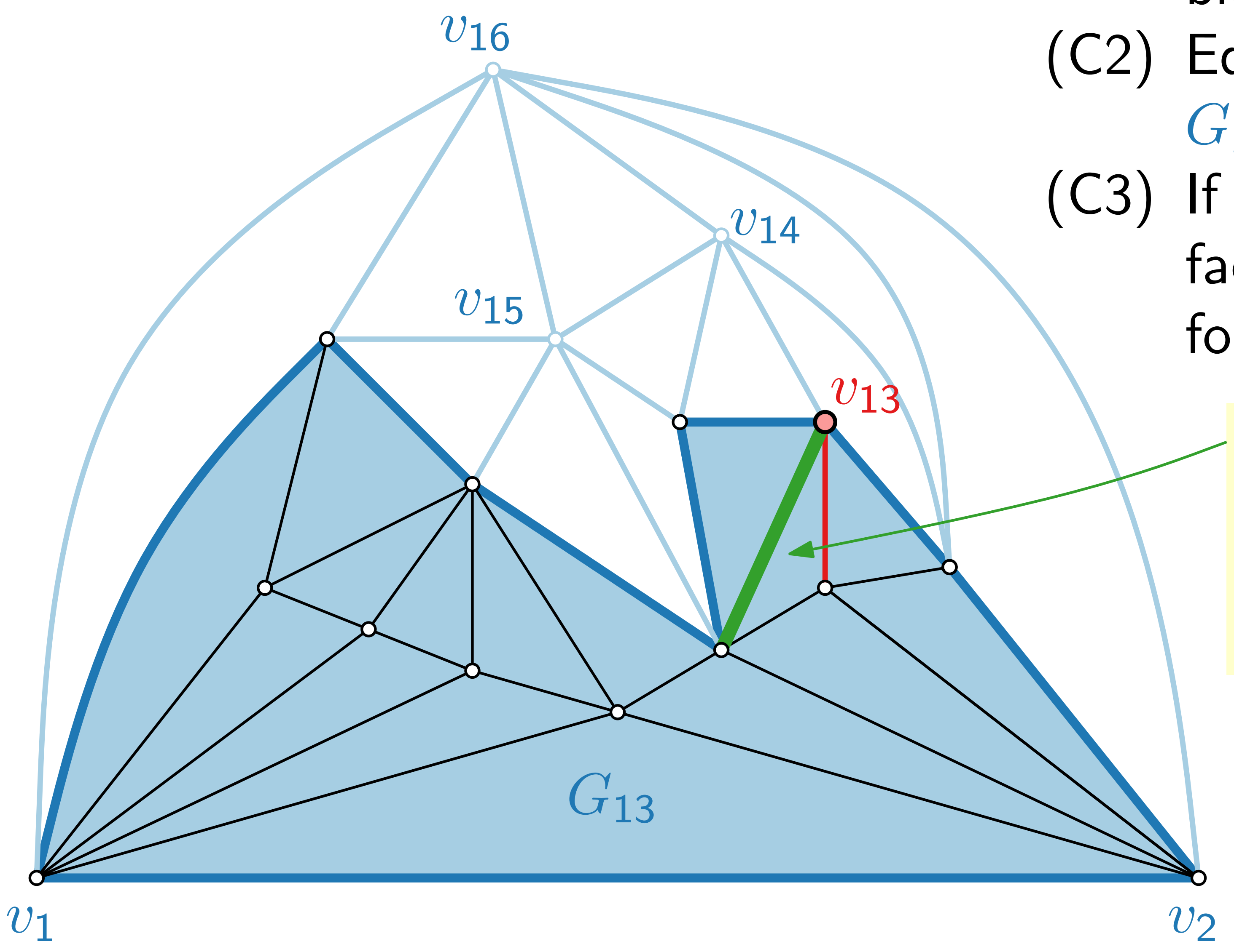


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example



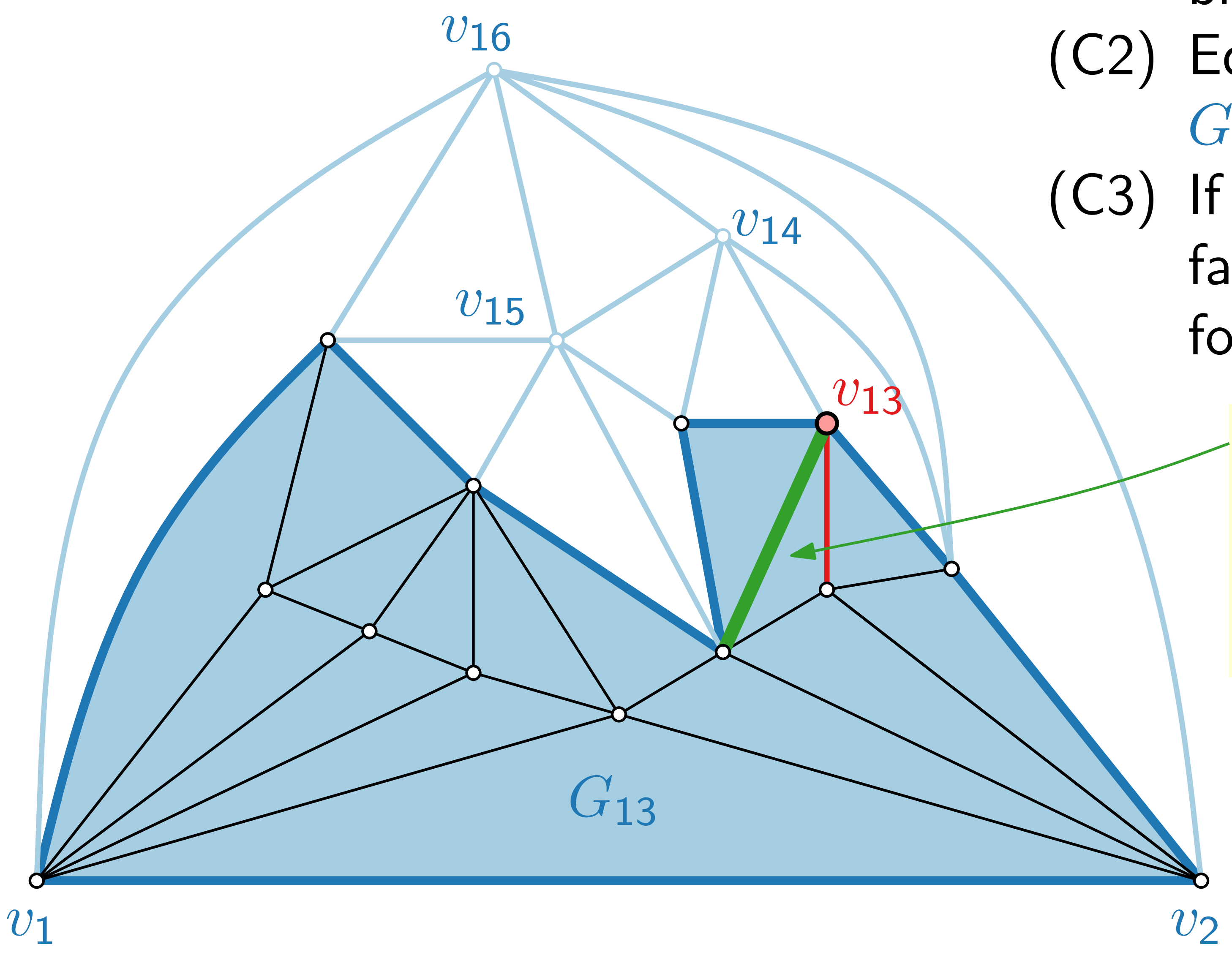
(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

chord:

Canonical Order – Example



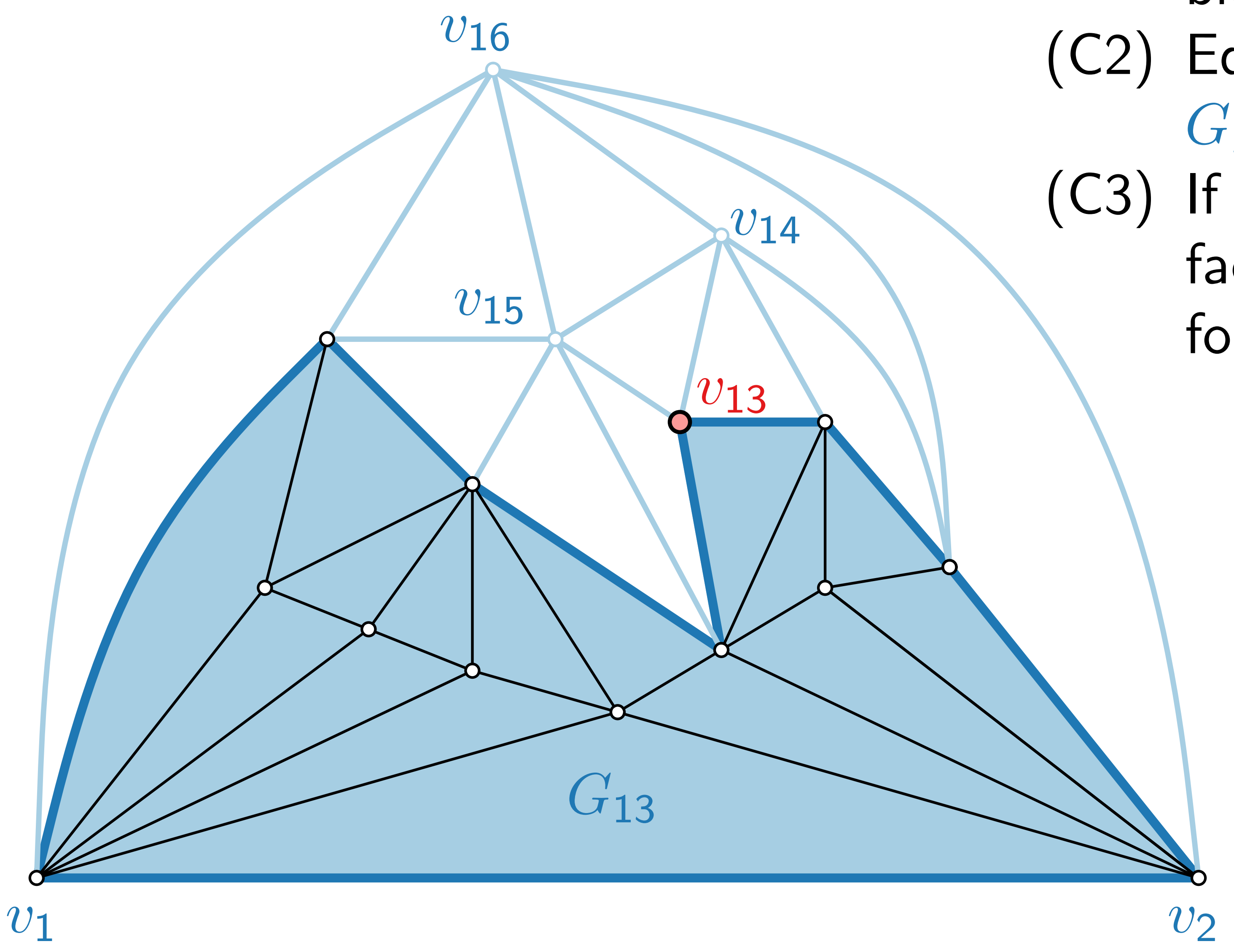
(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

chord:
edge joining two non-adjacent vertices in a cycle

Canonical Order – Example

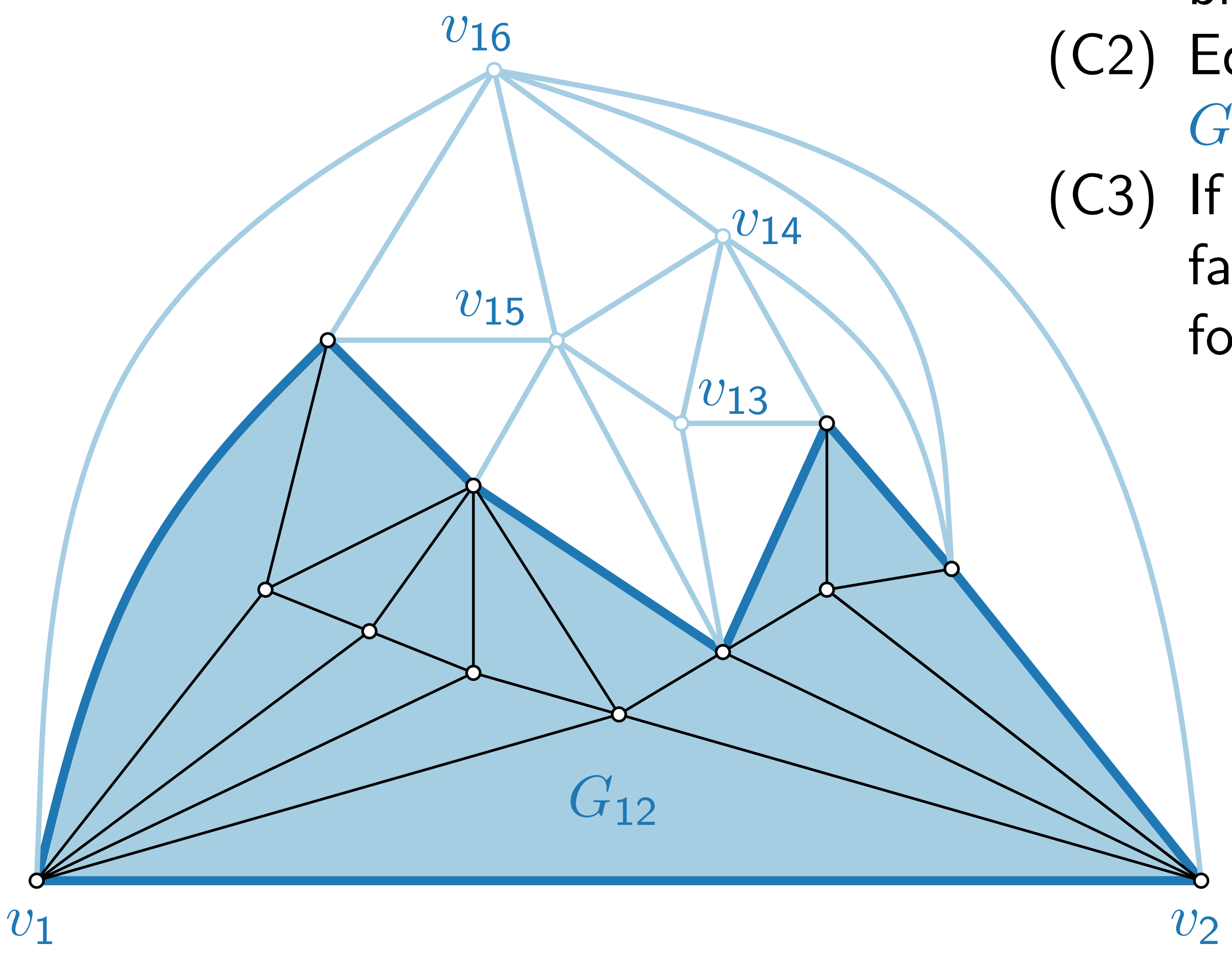


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

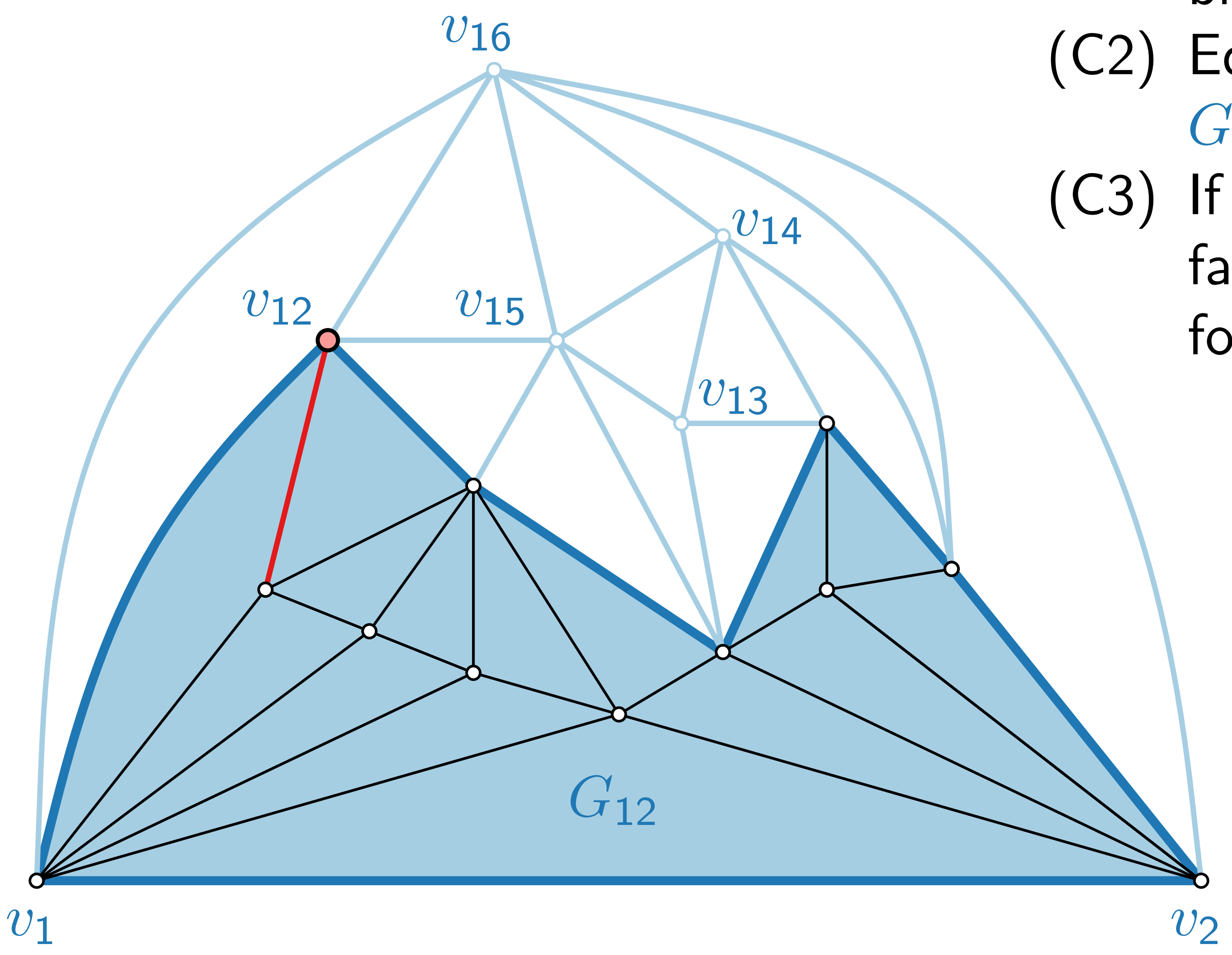


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

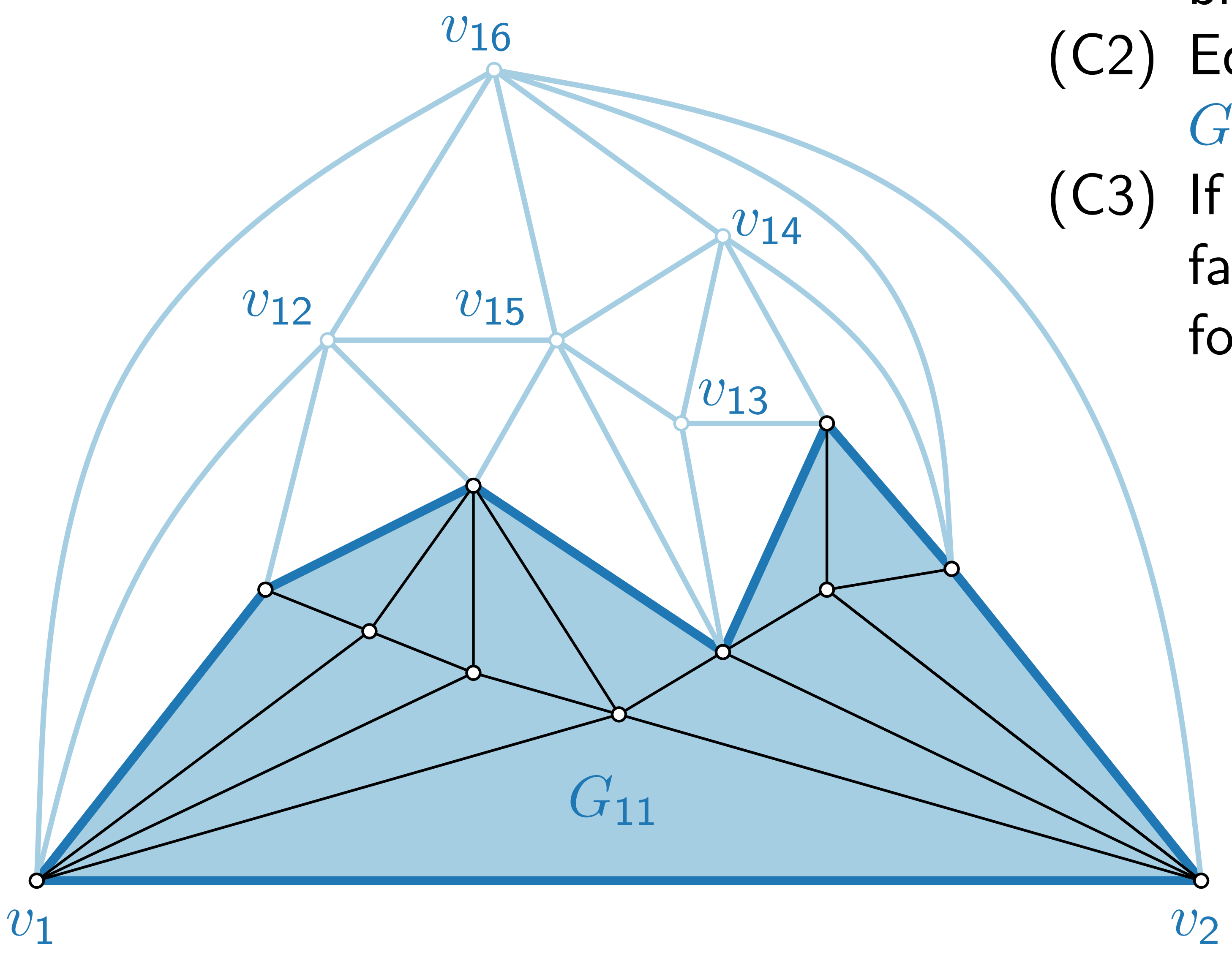


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

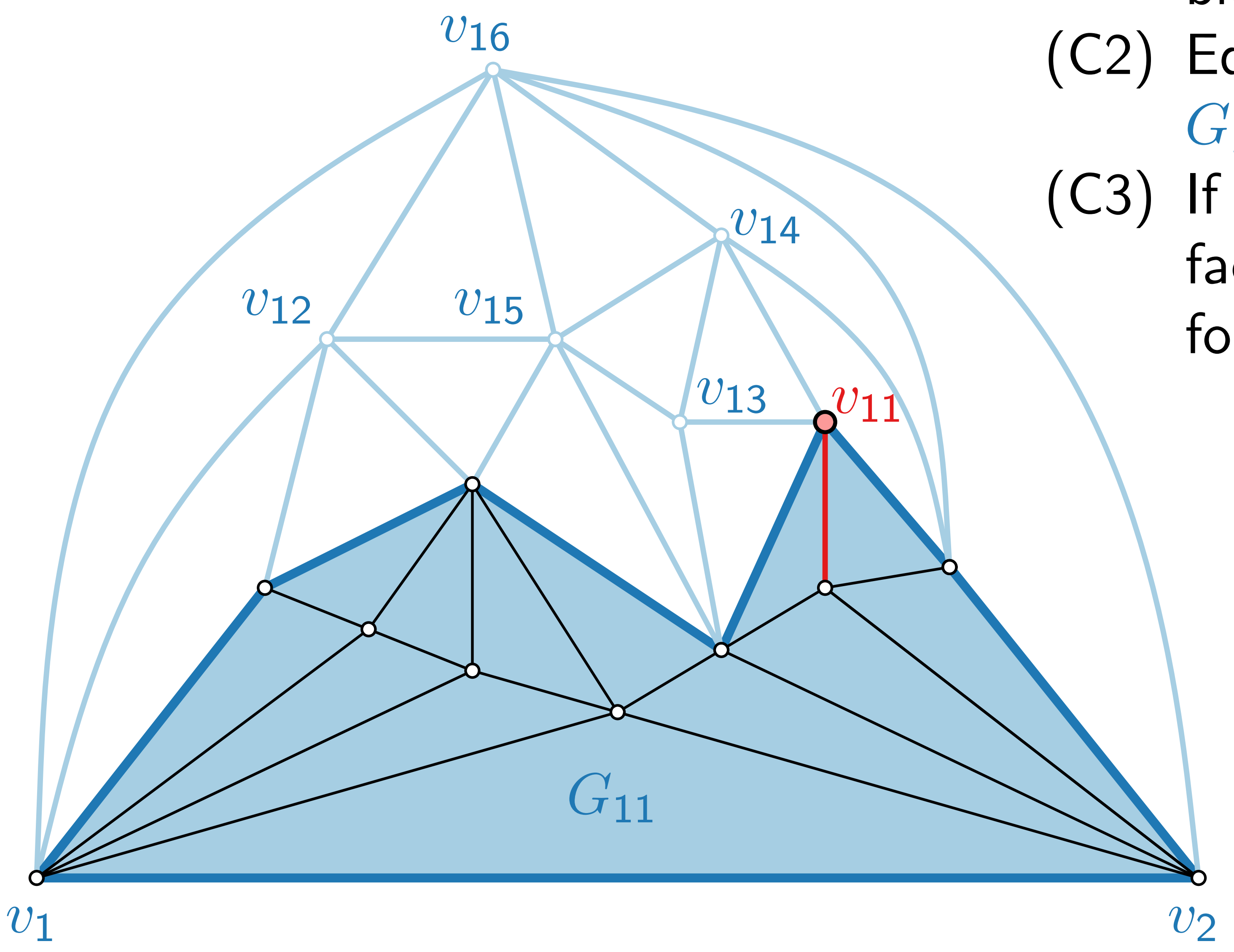


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

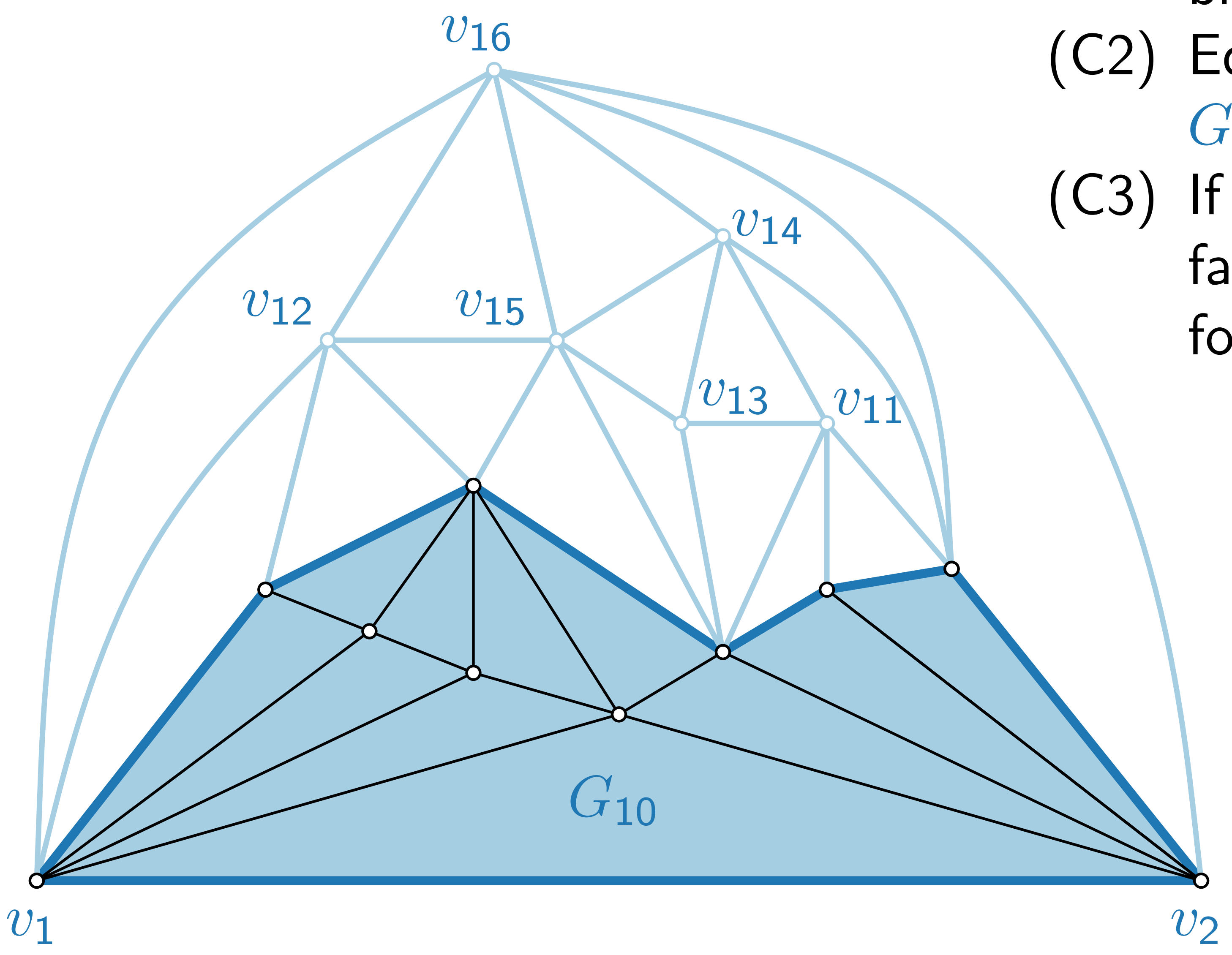


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

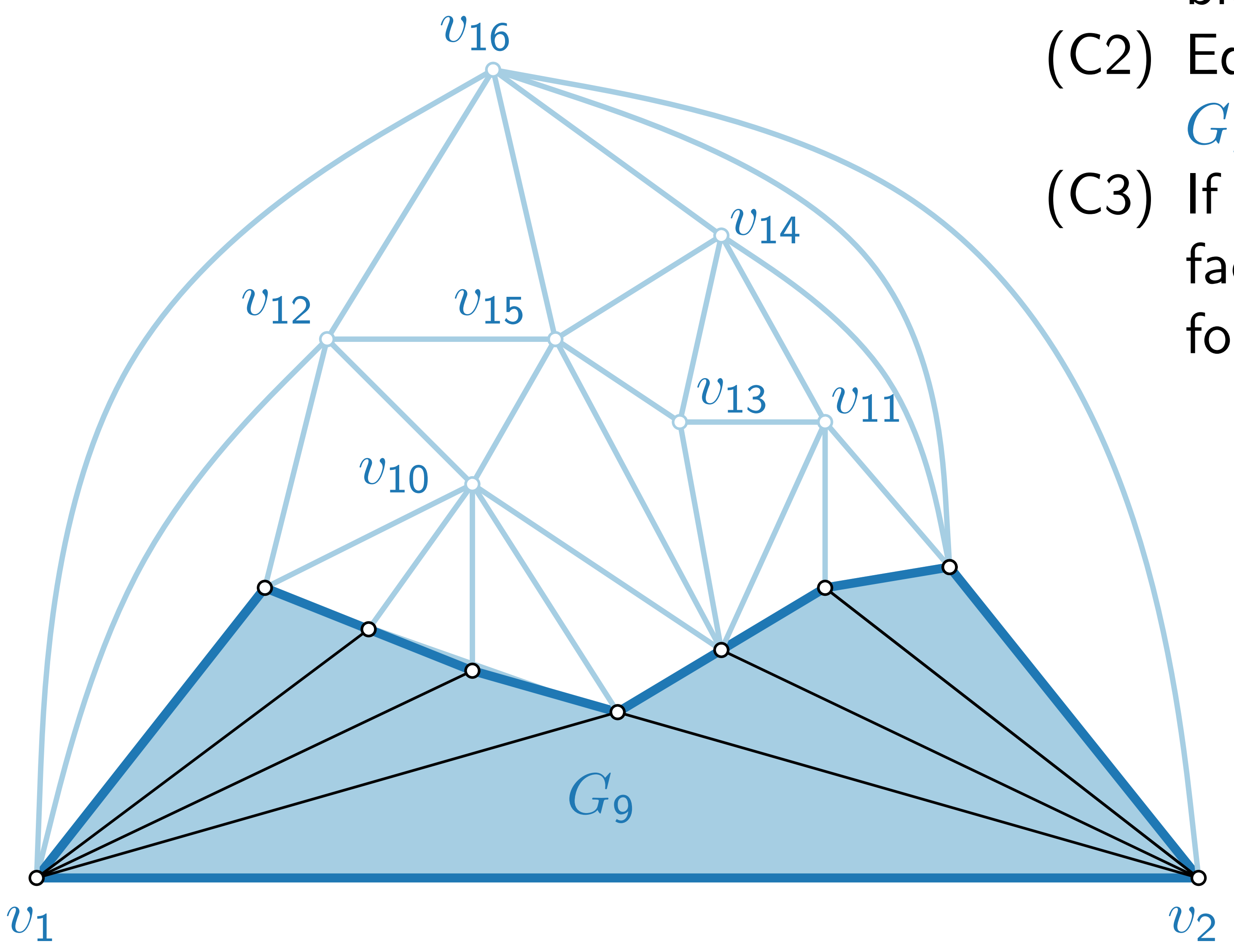


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

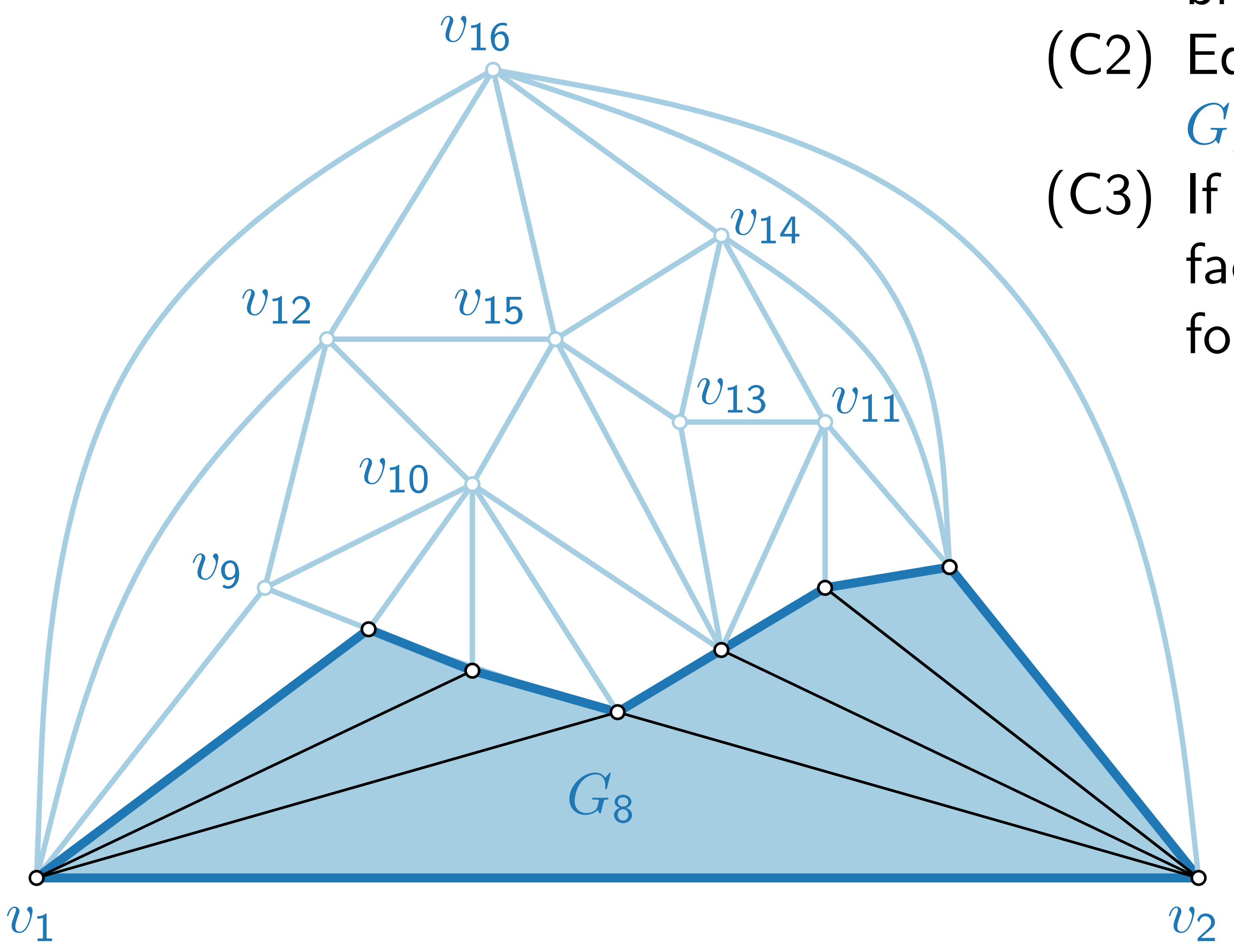


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

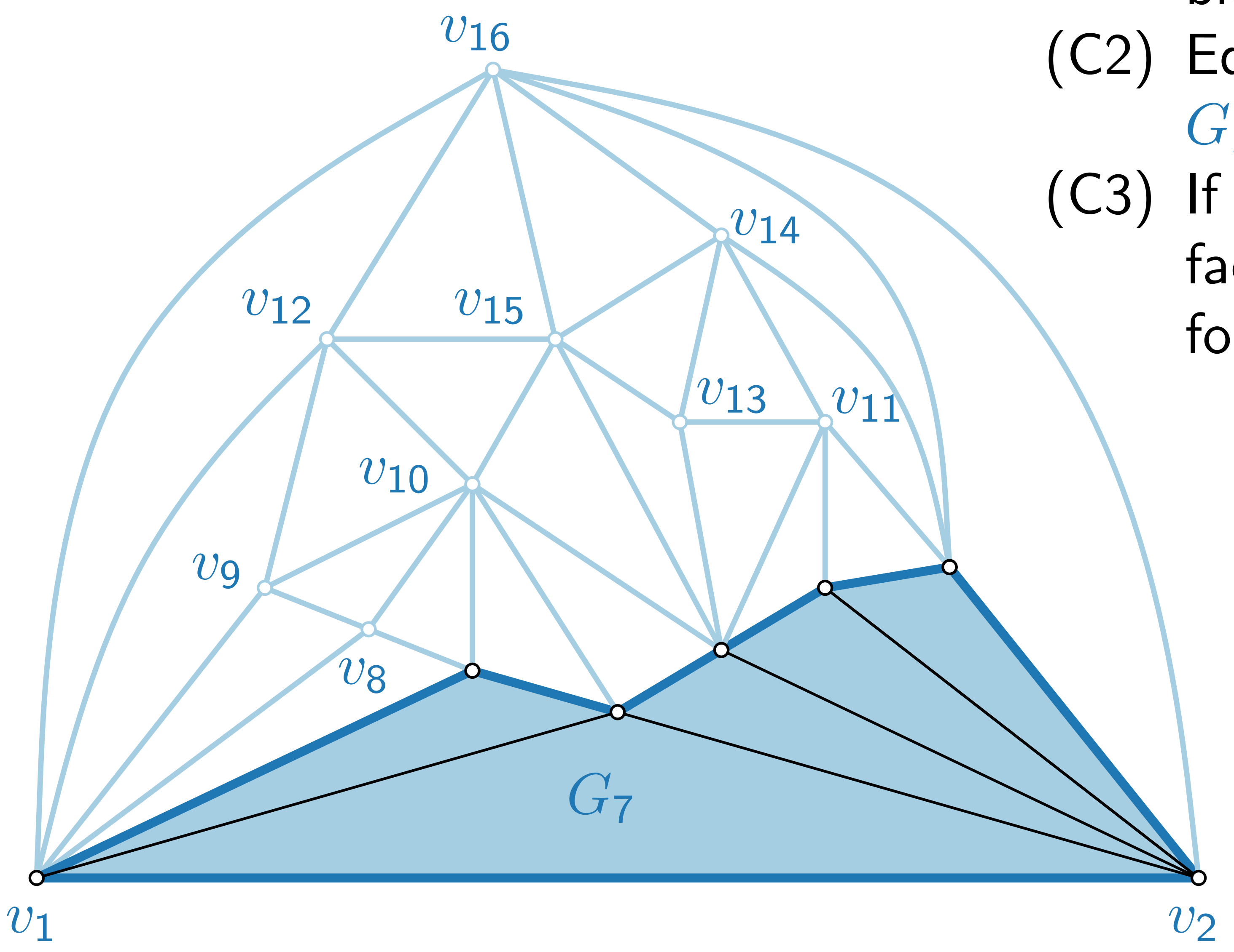


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

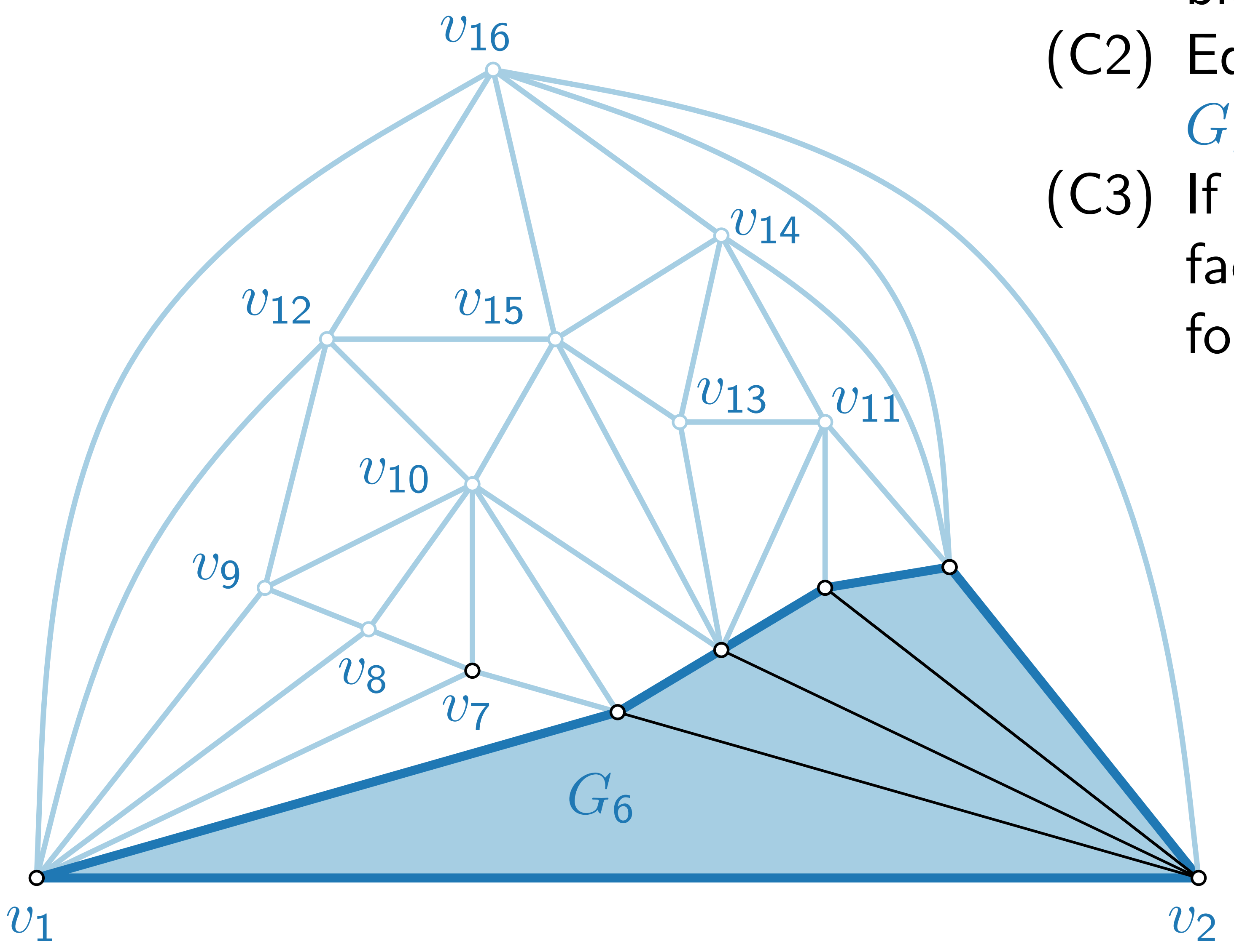


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

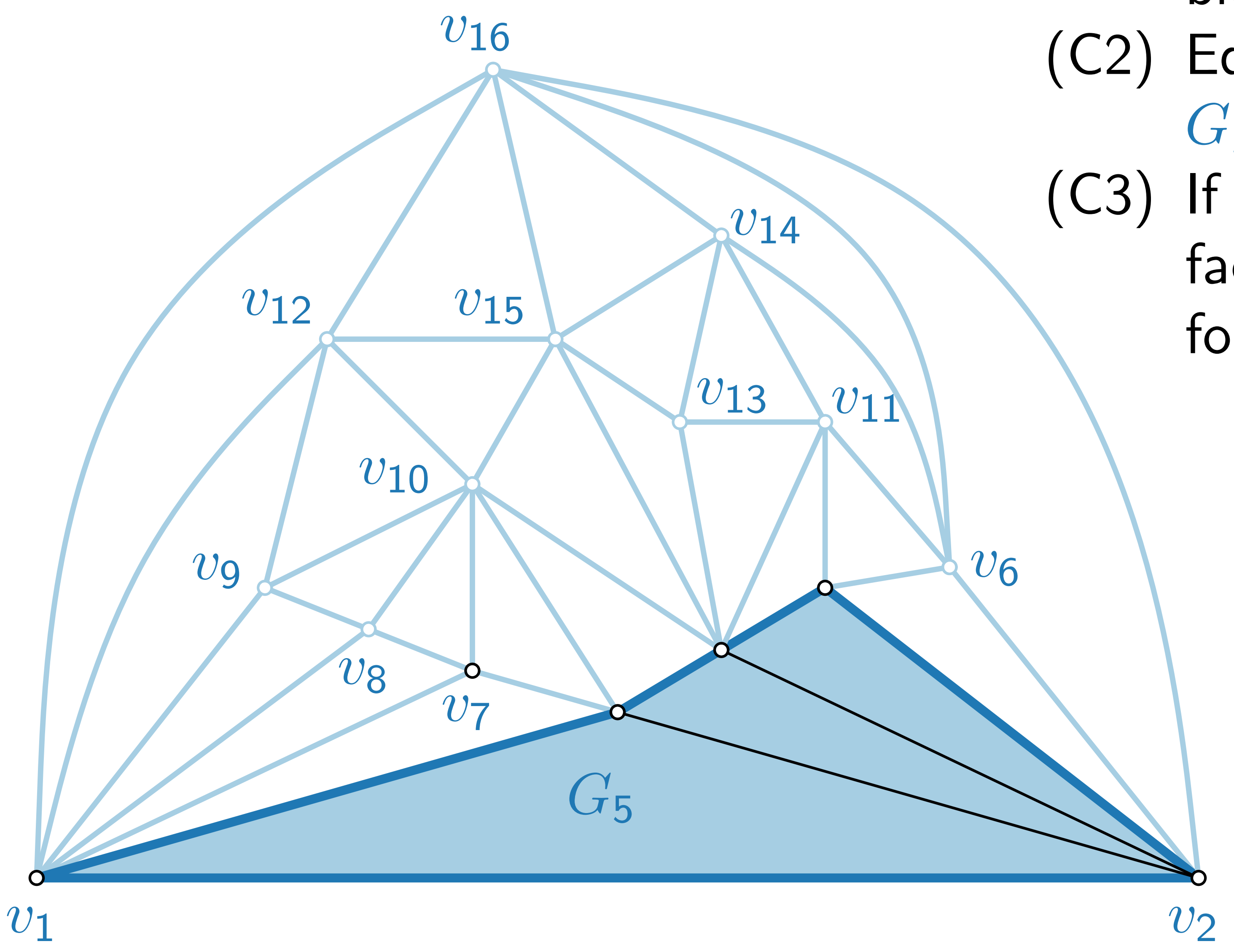


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

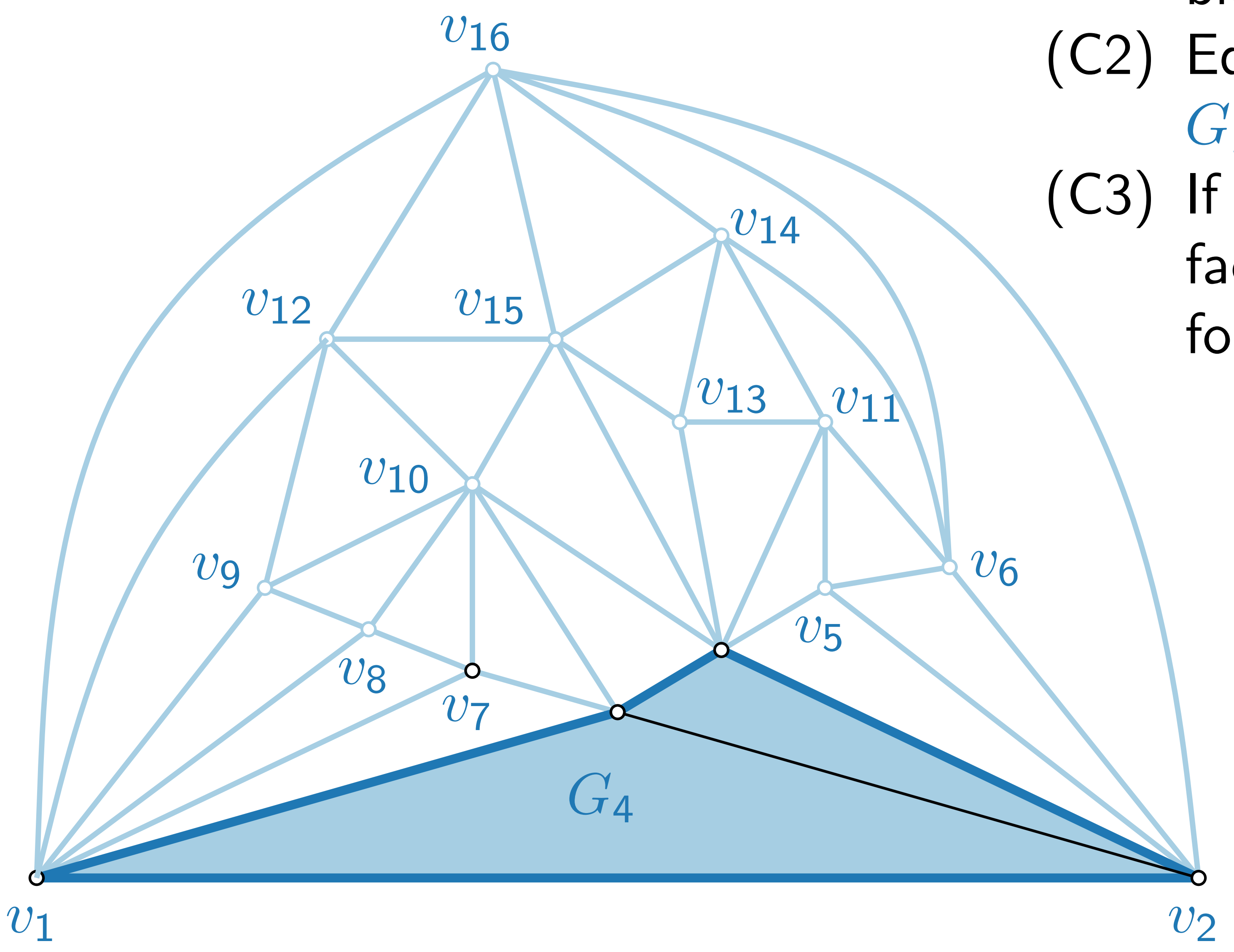


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

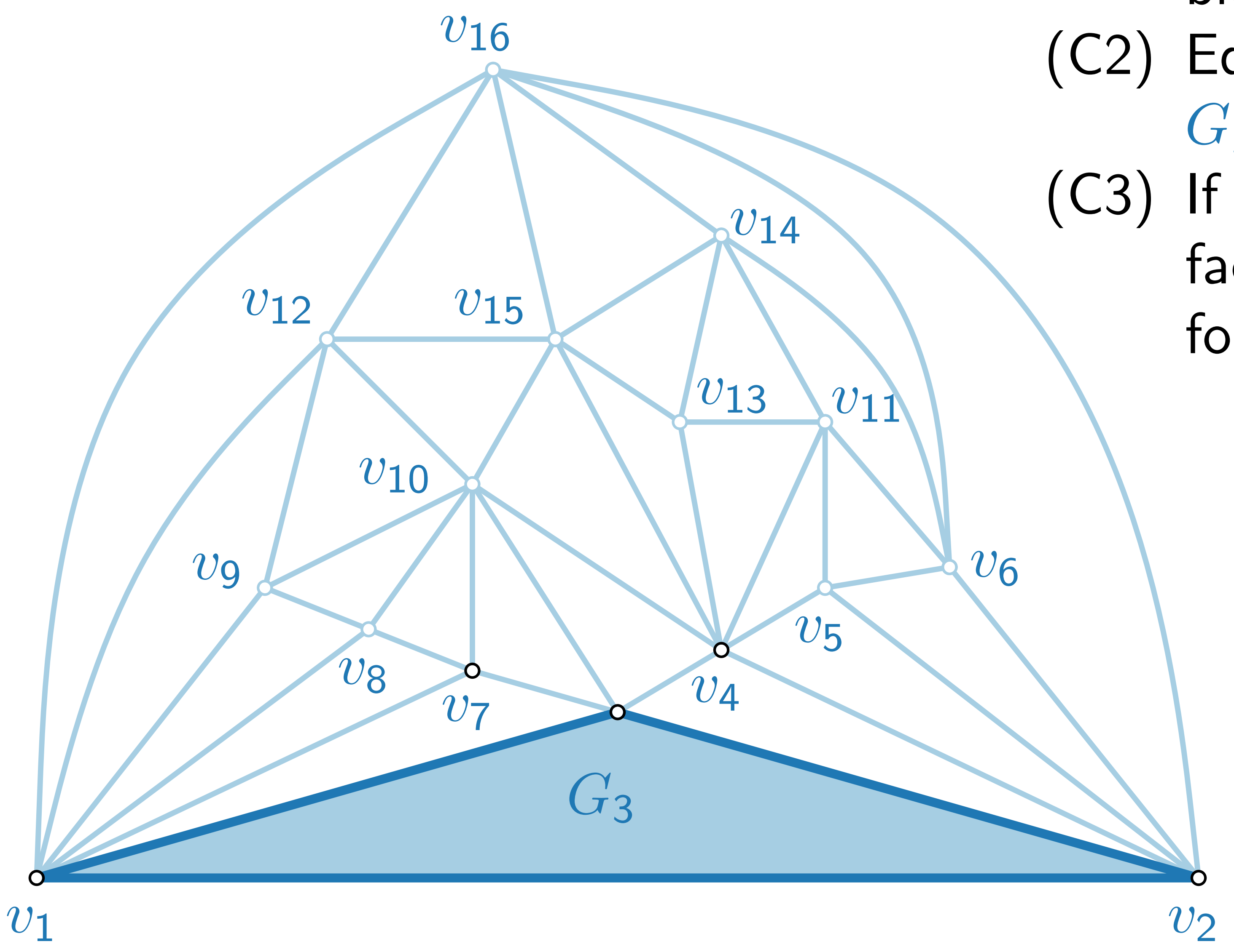


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

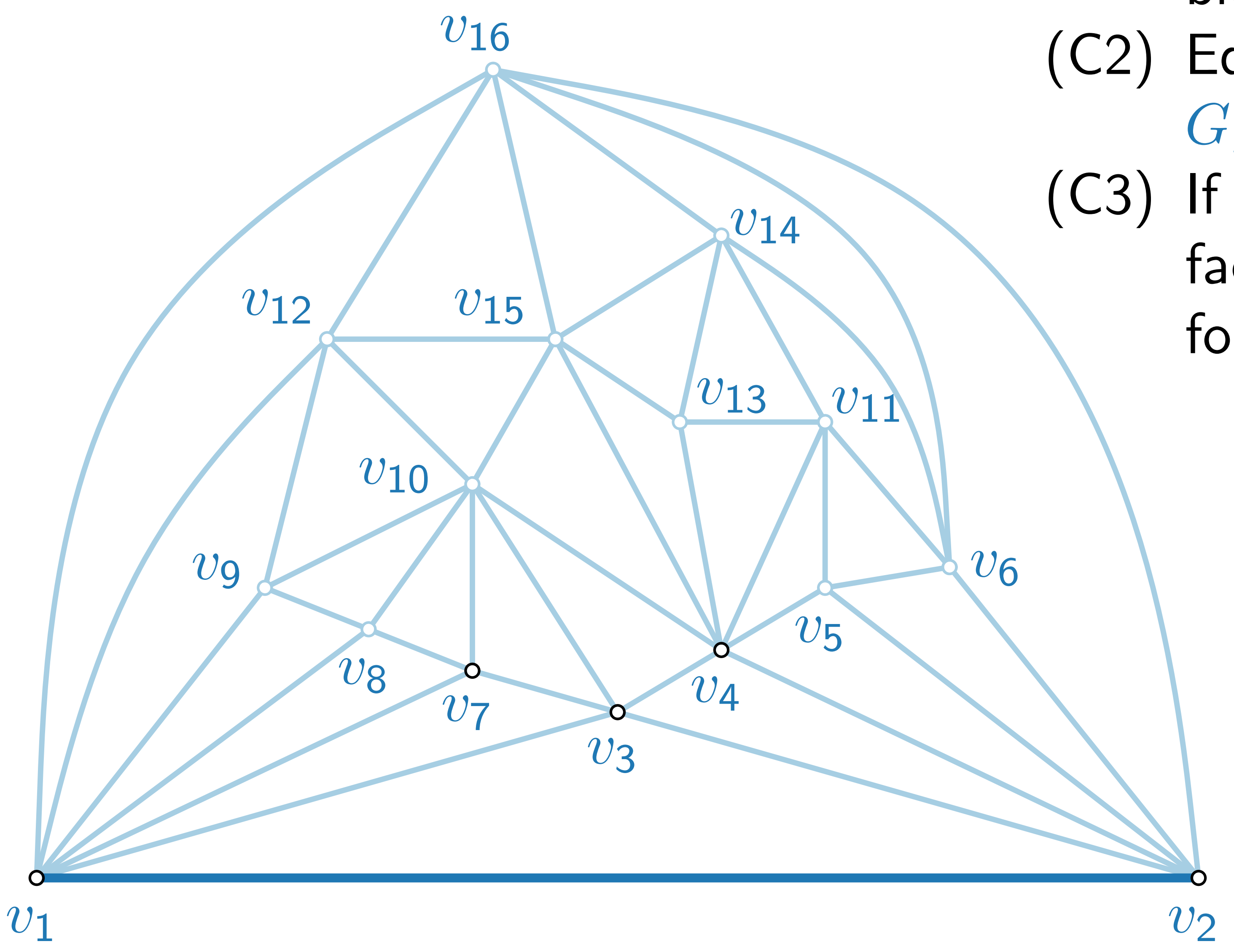


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example

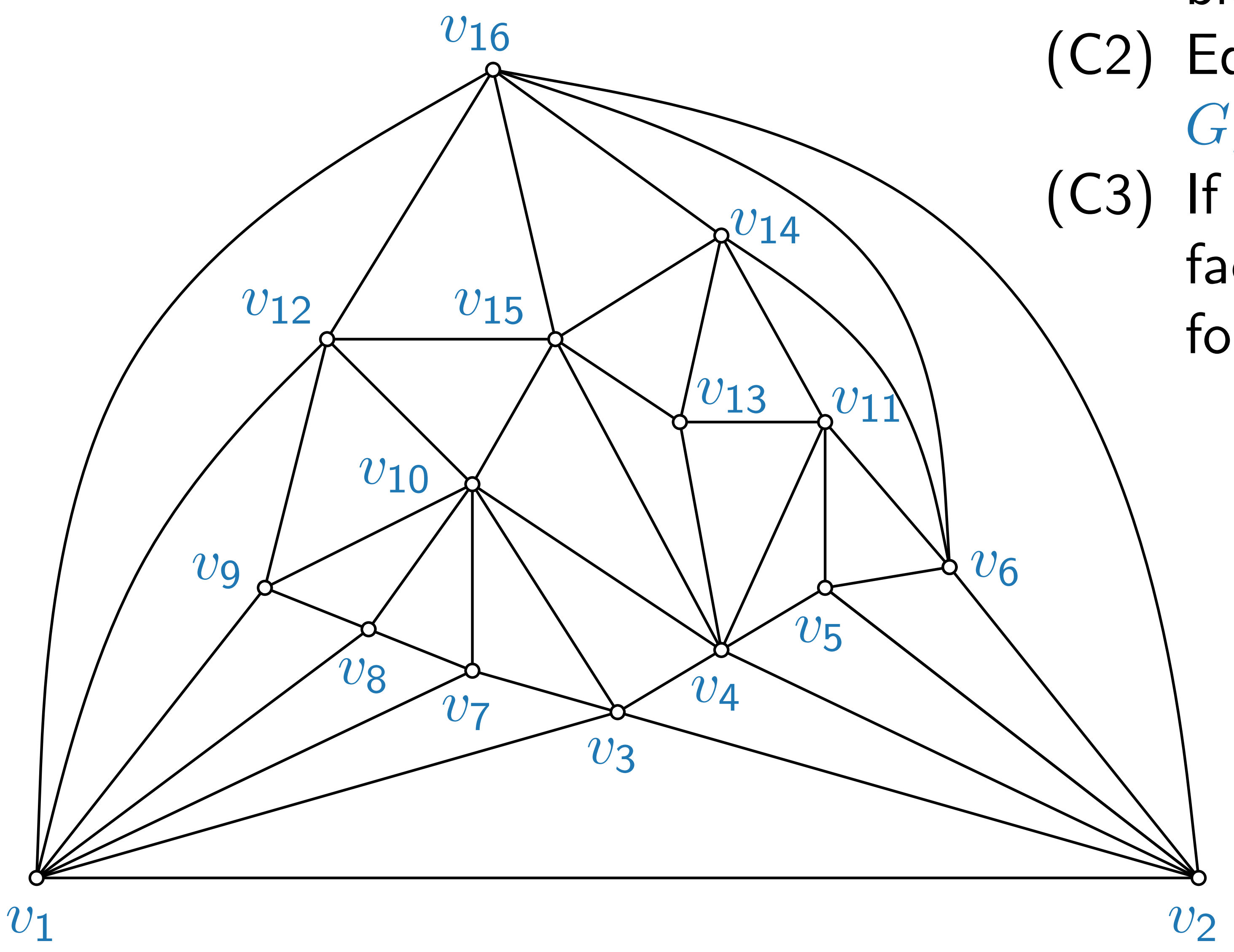


(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Example



(C1) Vertices $\{v_1, \ldots, v_k\}$ induce a biconnected inner triangulation; call it G_k.

(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

(C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and the neighbors of v_{k+1} form a path on the boundary of G_k.

Canonical Order – Existence

Lemma.
Every plane triangulation has a canonical order.

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

Canonical Order – Existence

Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

Canonical Order – Existence

Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n.

Canonical Order – Existence

(C1) G_k biconnected inner triangulation ✓

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n.

Canonical Order – Existence

(C1) G_k biconnected inner triangulation ✓

(C2) (v_1, v_2) on outer face of G_k ✓

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n.

Canonical Order – Existence

(C1) G_k biconnected inner triangulation ✓

(C2) (v_1, v_2) on outer face of G_k ✓

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k ✓

Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n.

Canonical Order – Existence

(C1) G_k biconnected inner triangulation ✓

(C2) (v_1, v_2) on outer face of G_k ✓

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k ✓

Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \ldots, n\}$.

Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

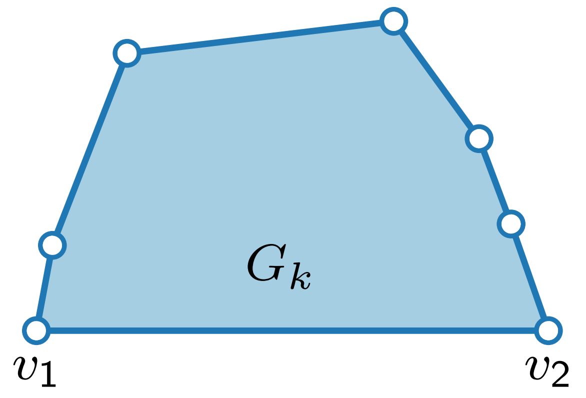
Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \ldots, n\}$.

Induction step: Consider G_k.



Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

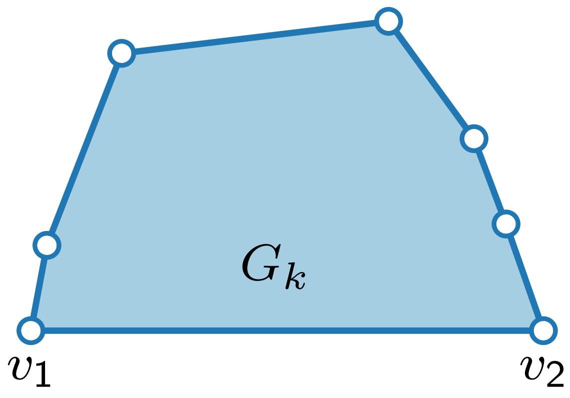
Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \ldots, n\}$.

Induction step: Consider G_k. We search for v_k.



Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

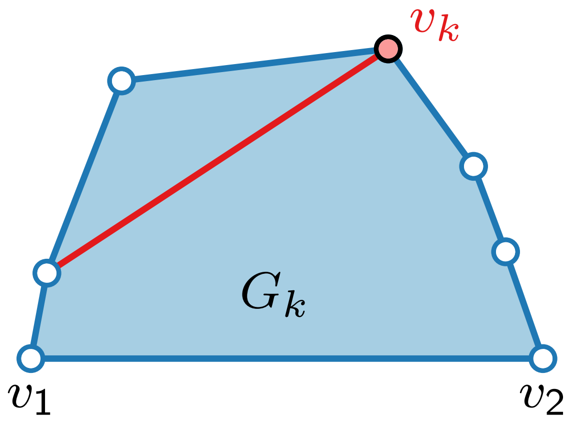
Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \ldots, n\}$.

Induction step: Consider G_k. We search for v_k.



Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

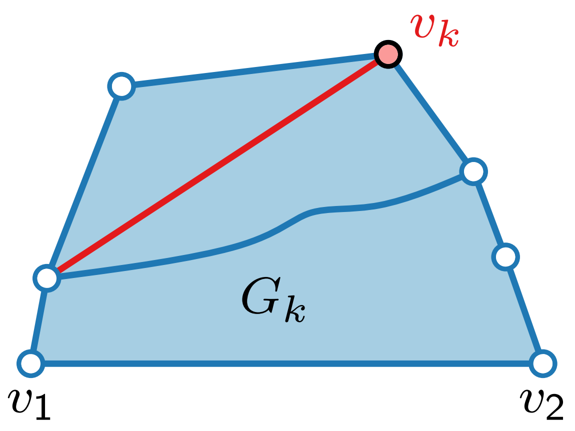
Lemma.
Every plane triangulation has a canonical order.

Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \dots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \dots, n\}$.

Induction step: Consider G_k. We search for v_k.



Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

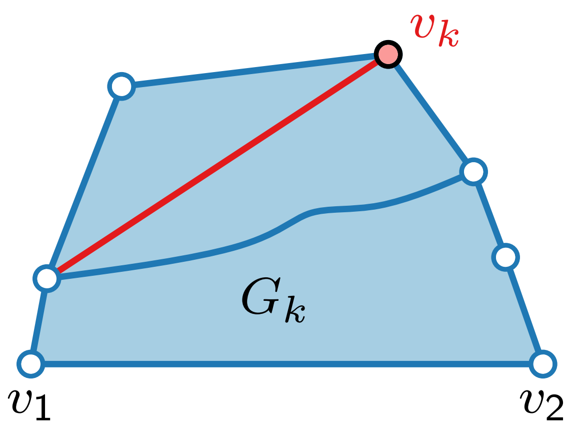
Lemma.
Every plane triangulation has a canonical order.

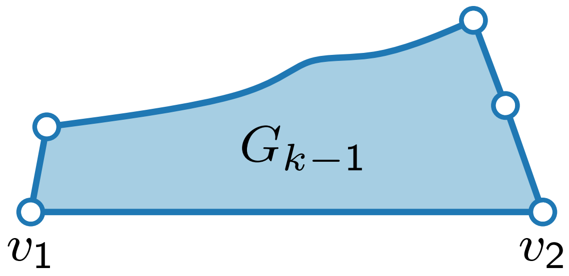
Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \ldots, n\}$.

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

Lemma.
Every plane triangulation has a canonical order.

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

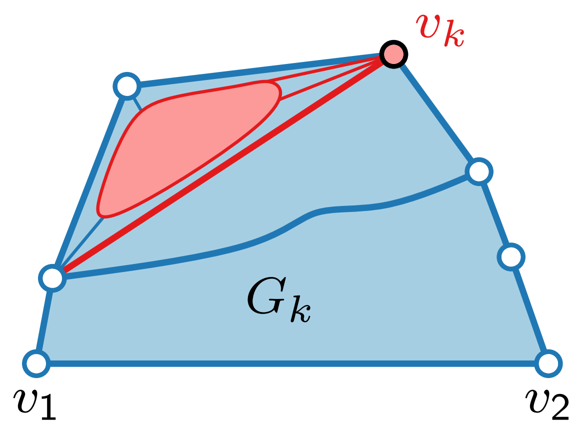
(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

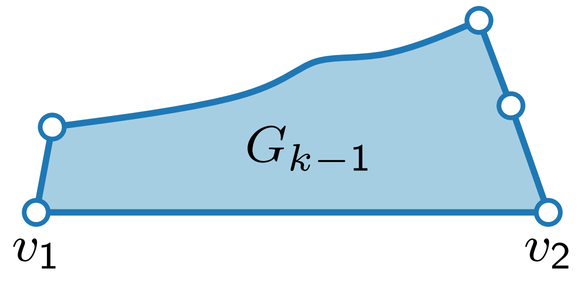
Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \ldots, n\}$.

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

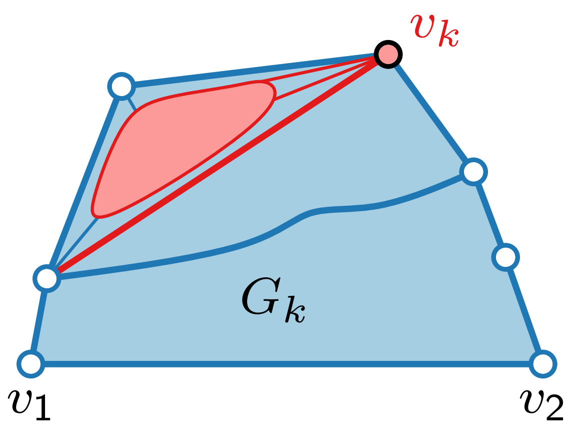
Lemma.
Every plane triangulation has a canonical order.

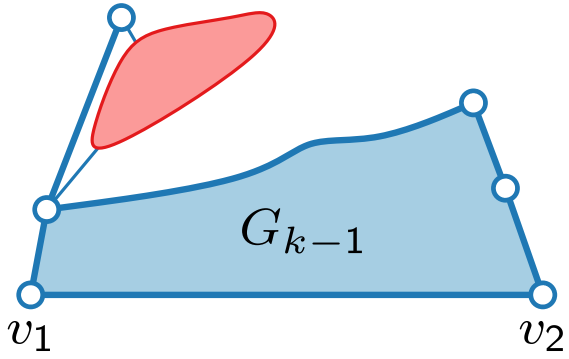
Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \dots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \dots, n\}$.

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

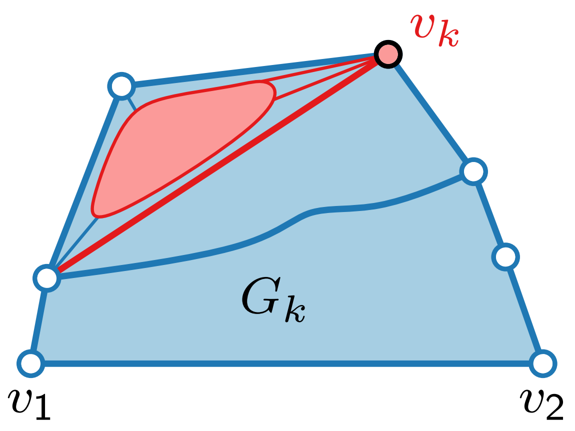
Lemma.
Every plane triangulation has a canonical order.

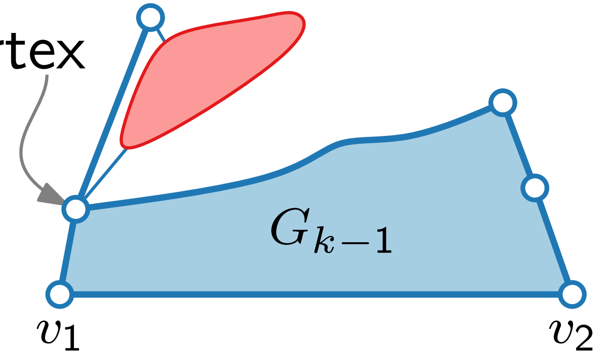
Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \dots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \dots, n\}$.

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

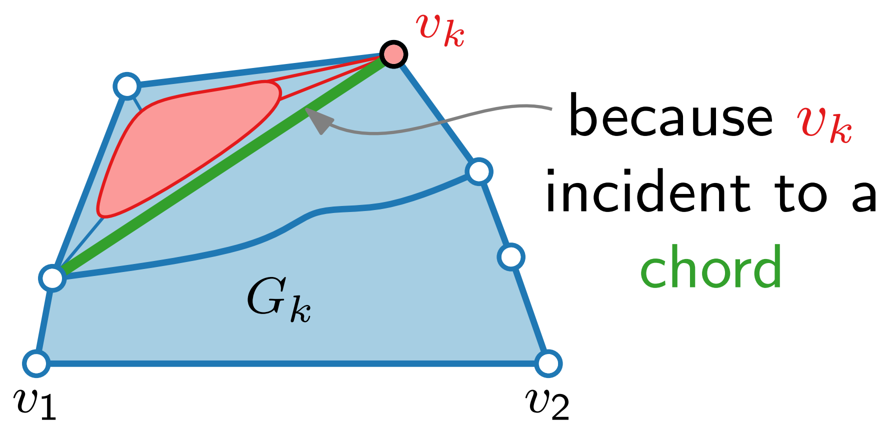
Lemma.
Every plane triangulation has a canonical order.

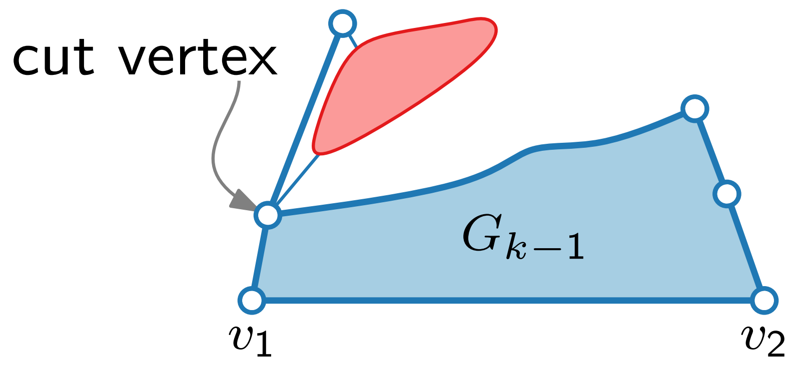
Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \dots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \dots, n\}$.

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

Lemma.
Every plane triangulation has a canonical order.

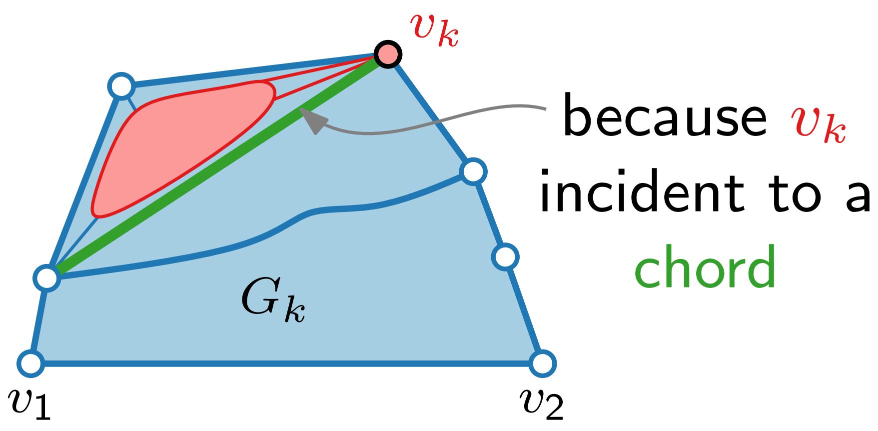
Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

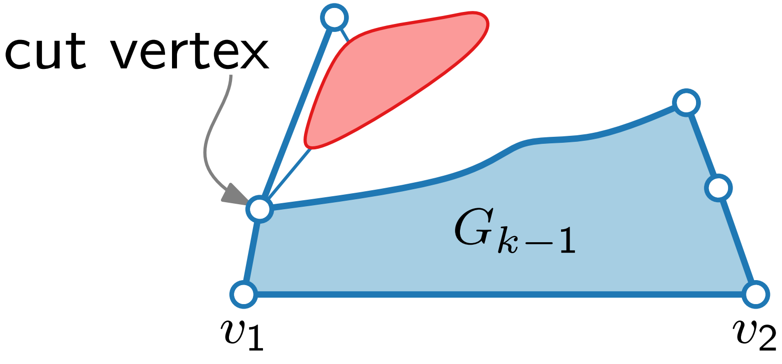
Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \dots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \dots, n\}$.

Induction step: Consider G_k. We search for v_k.

We need to show:





Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

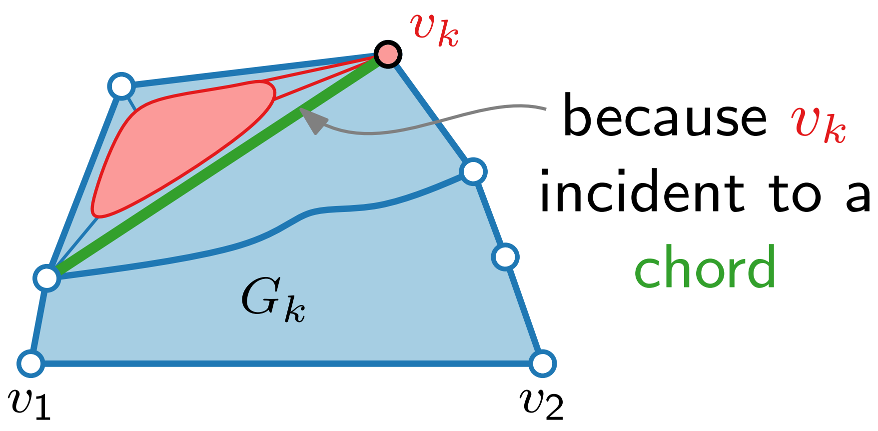
Lemma.
Every plane triangulation has a canonical order.

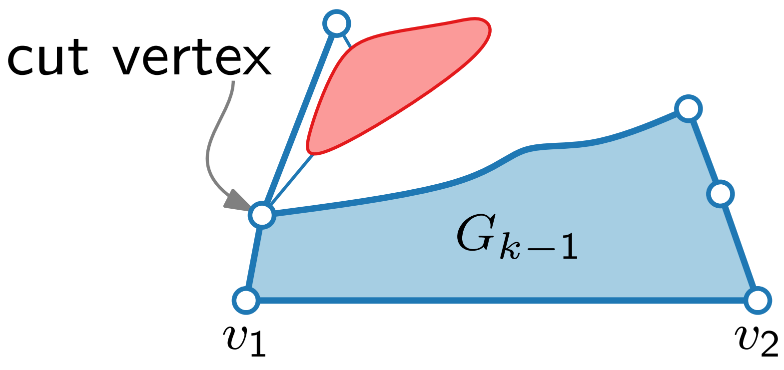
Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \dots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \dots, n\}$.

Induction step: Consider G_k. We search for v_k.





We need to show:

1. v_k not incident to chord is sufficient.

Canonical Order – Existence

(C1) G_k biconnected inner triangulation

(C2) (v_1, v_2) on outer face of G_k

(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} form path on boundary of G_k

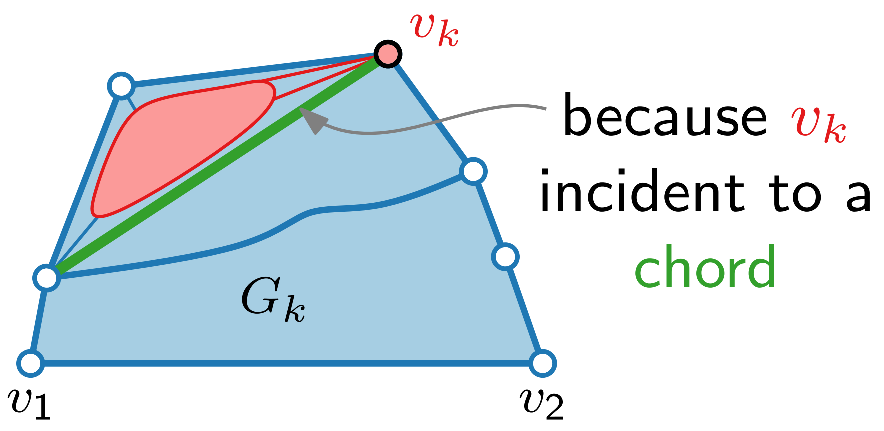
Lemma.
Every plane triangulation has a canonical order.

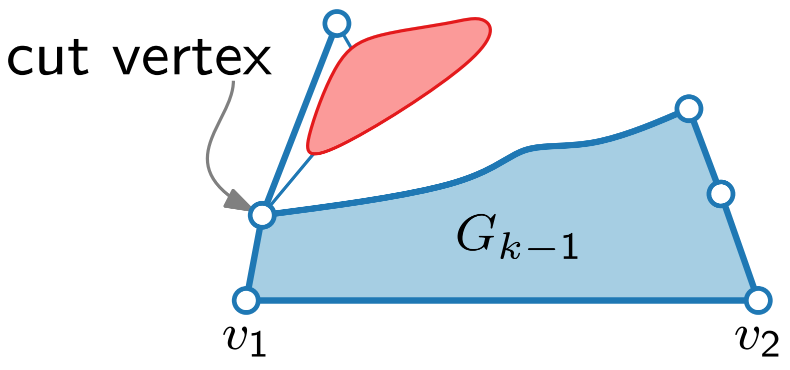
Consider any n-vertex plane triangulation. We show this statement by induction on k from n down to 3.

Induction base $(k = n)$: Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1)–(C3) hold.

Induction hypothesis: Vertices $v_{n-1}, \dots, v_{k+1}$ have been chosen such that conditions (C1)–(C3) hold for every $i \in \{k+1, \dots, n\}$.

Induction step: Consider G_k. We search for v_k.





We need to show:

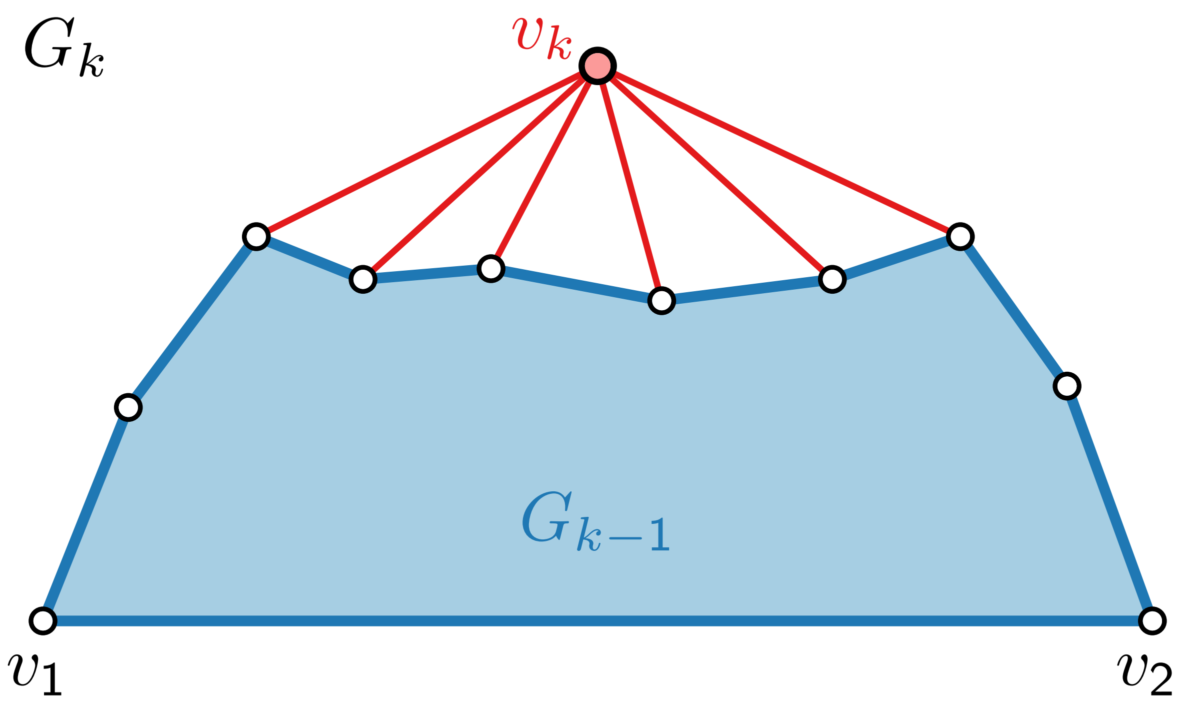
1. v_k not incident to chord is sufficient.
2. Such v_k exists.

Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.

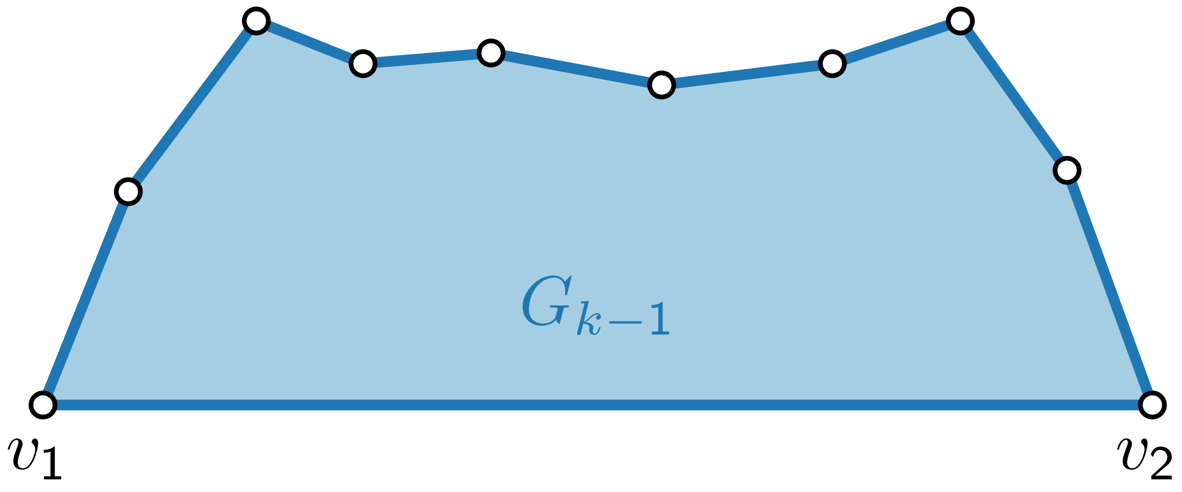
Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.



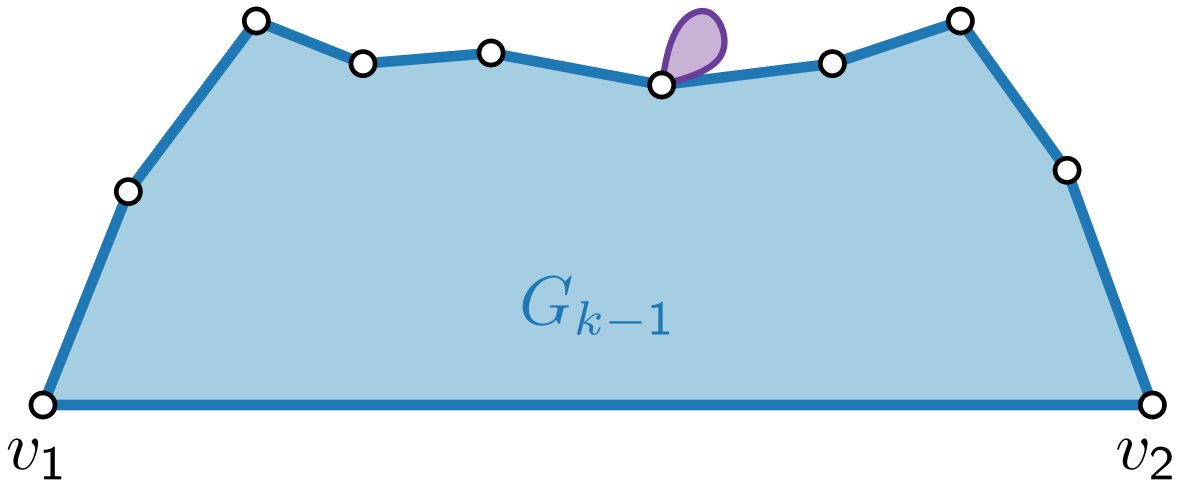
Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.



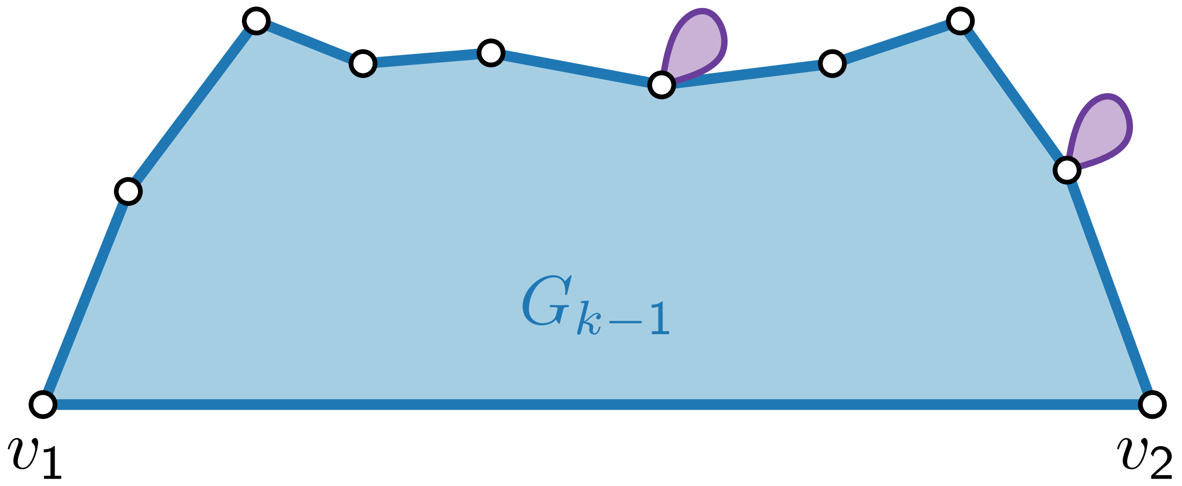
Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.



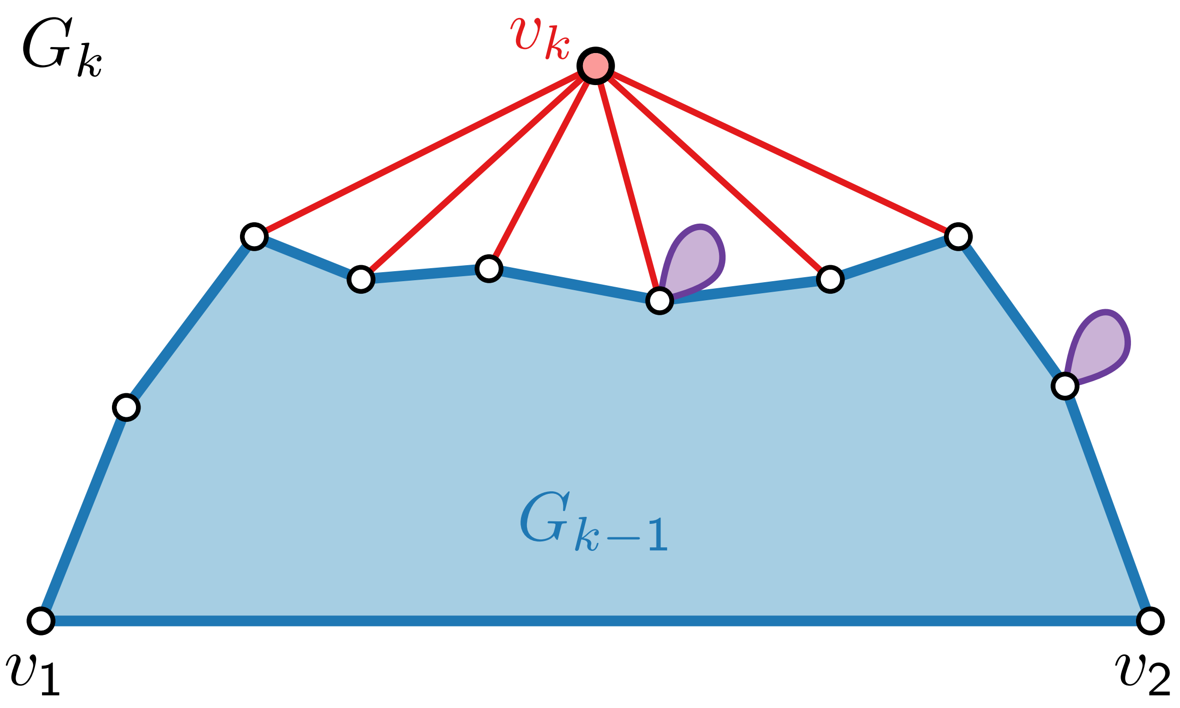
Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.



Canonical Order – Existence

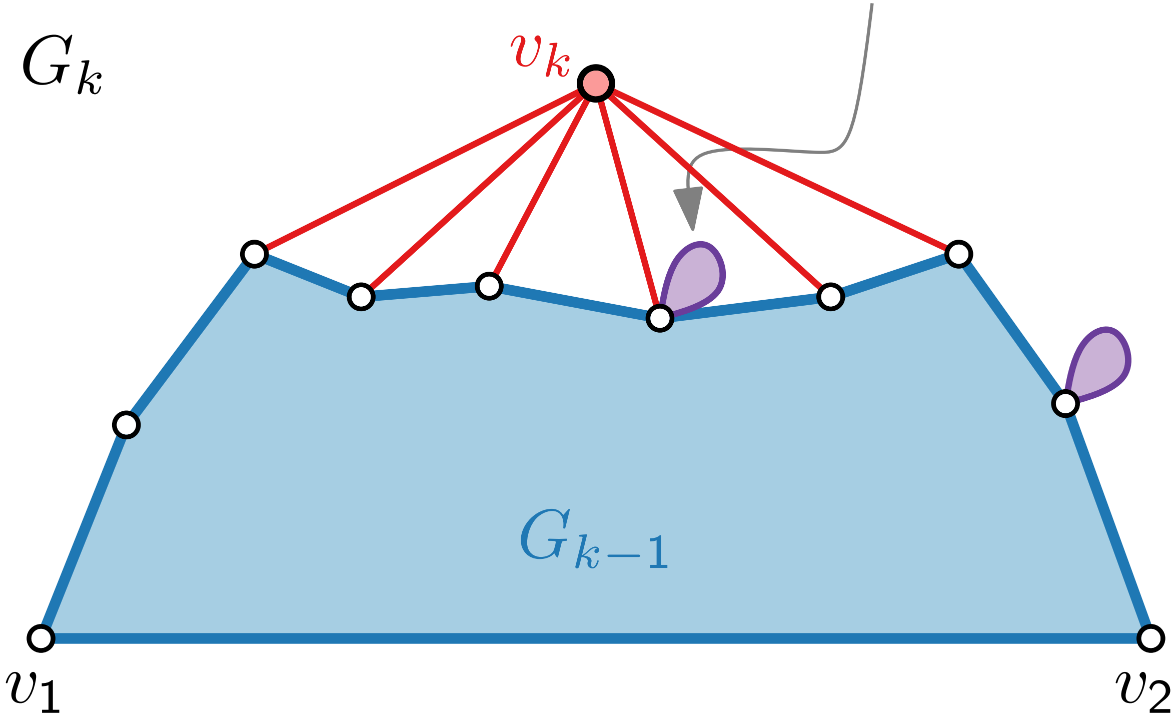
Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.



Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.

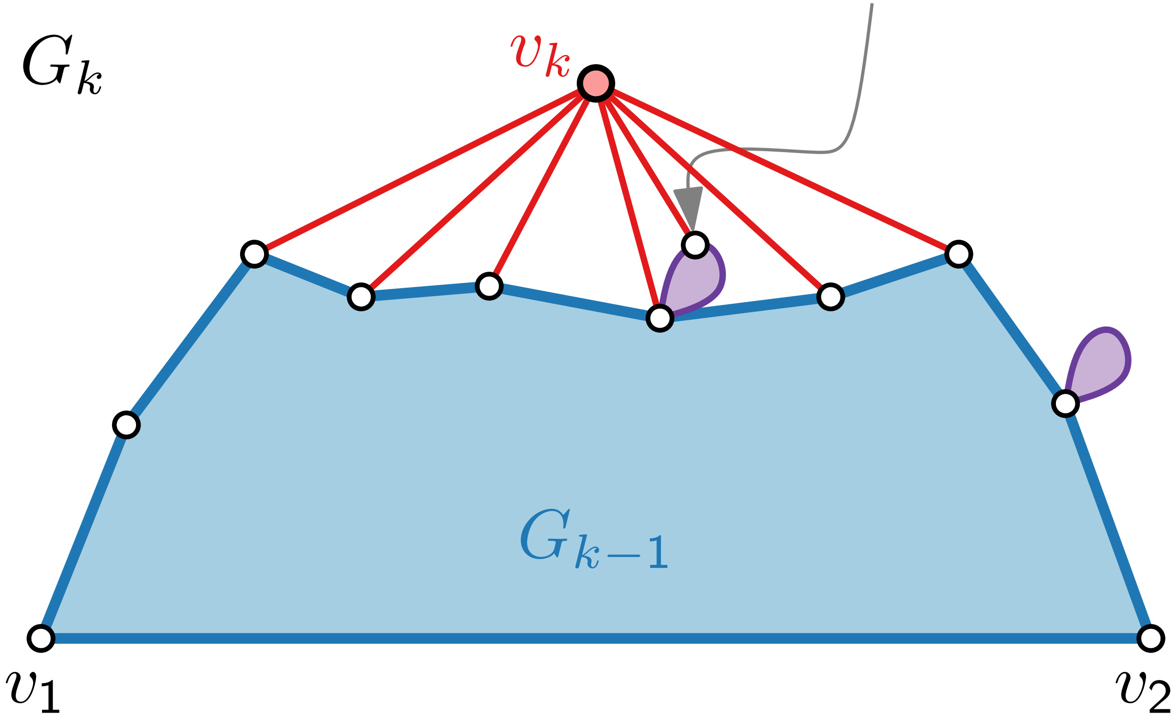
Contradiction to neighbors of v_k forming a path on ∂G_{k-1}!



Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.

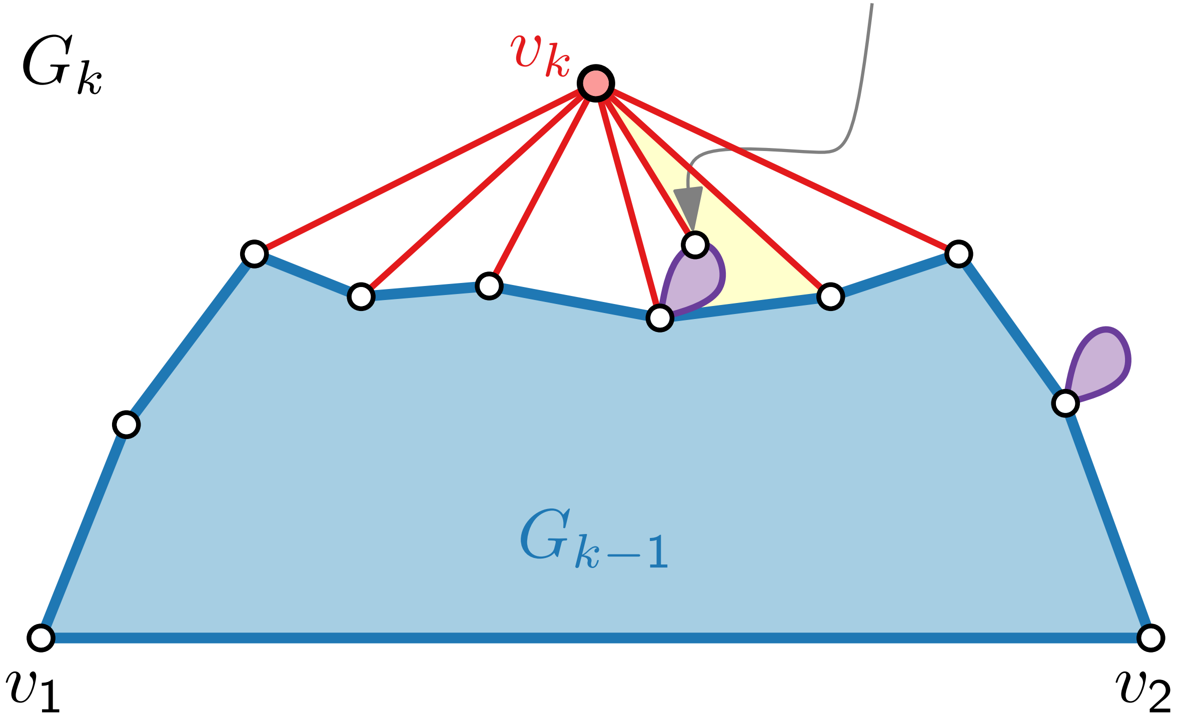
Contradiction to neighbors of v_k forming a path on ∂G_{k-1}!



Canonical Order – Existence

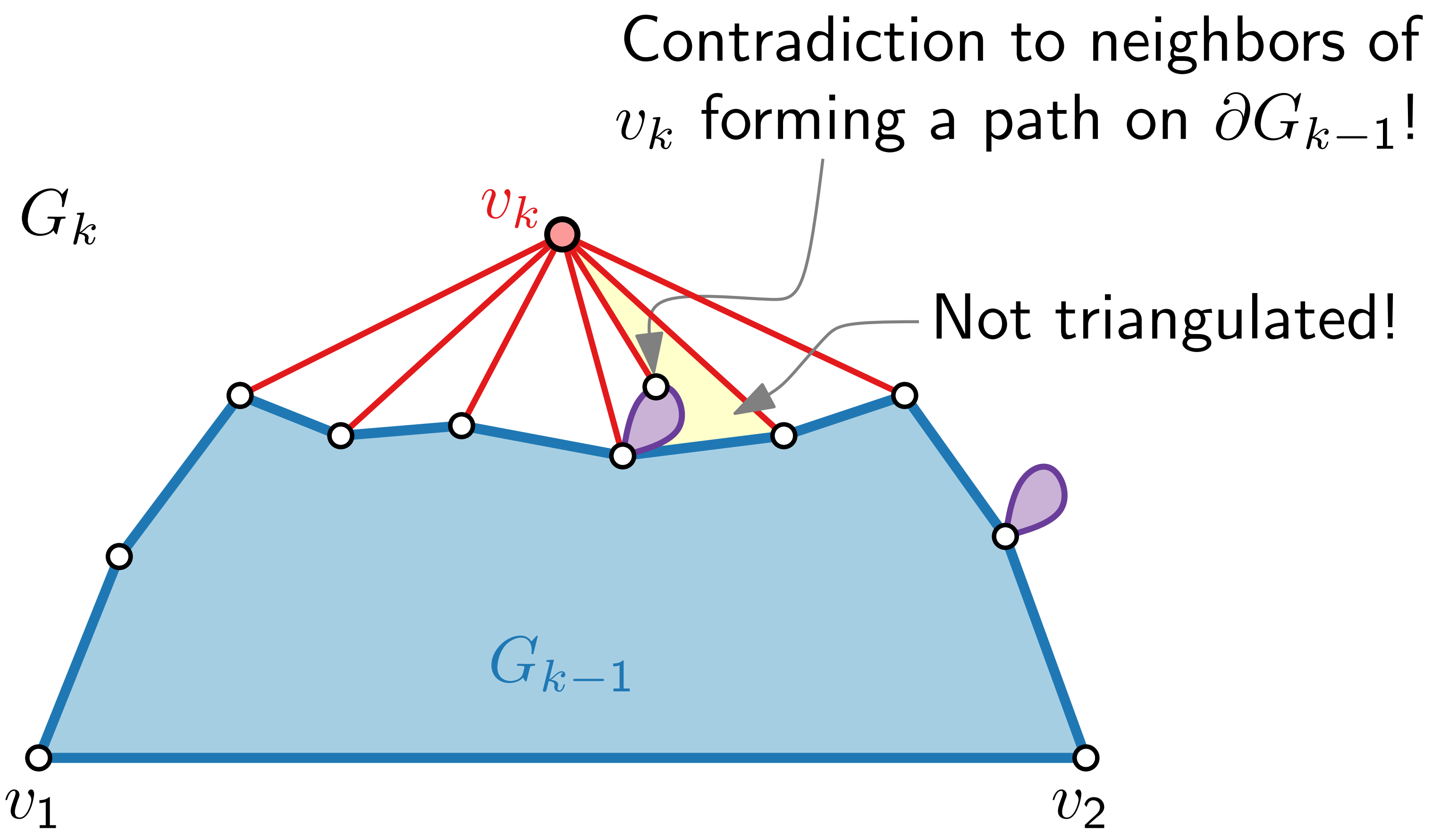
Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.

Contradiction to neighbors of
v_k forming a path on ∂G_{k-1}!



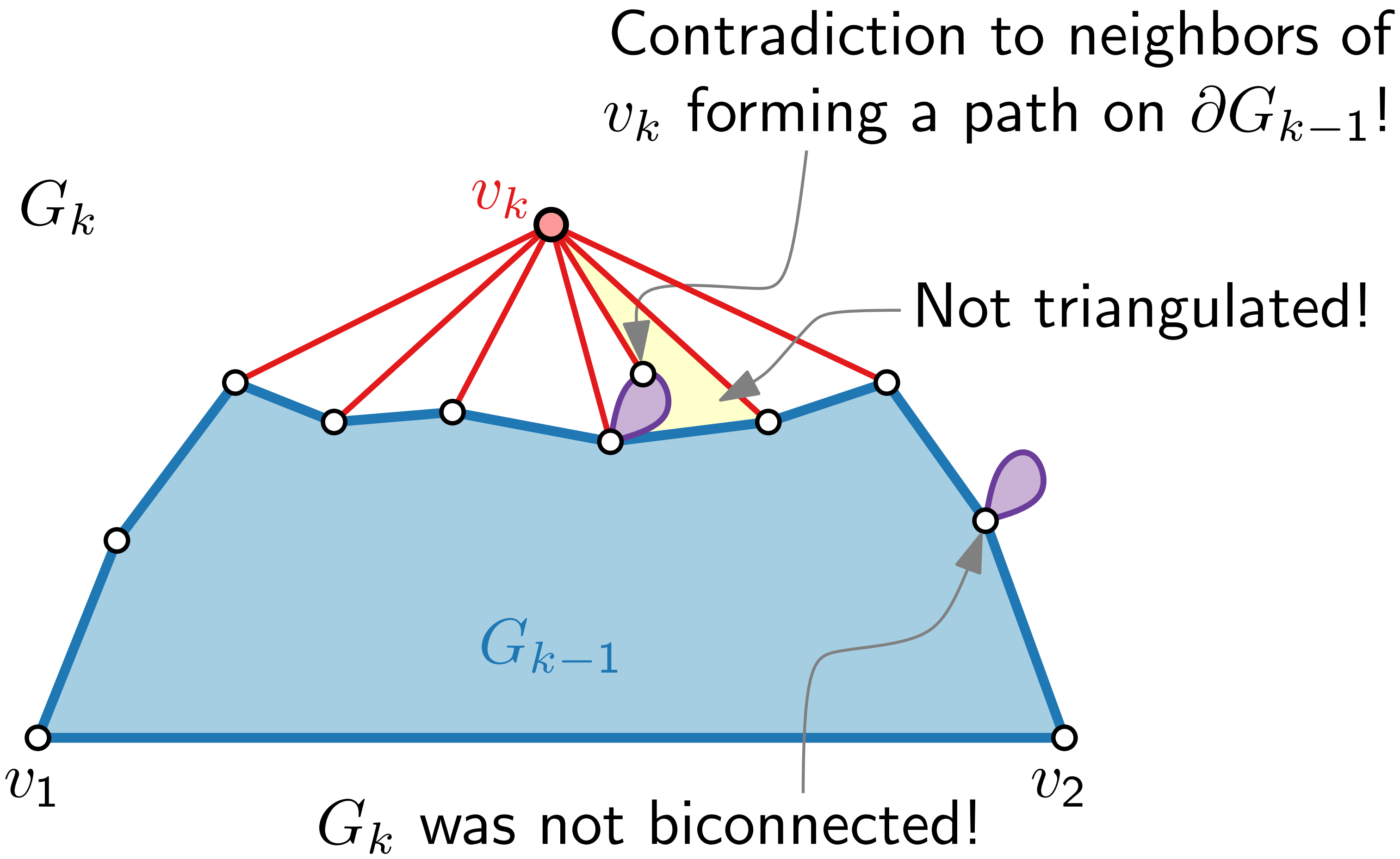
Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.



Canonical Order – Existence

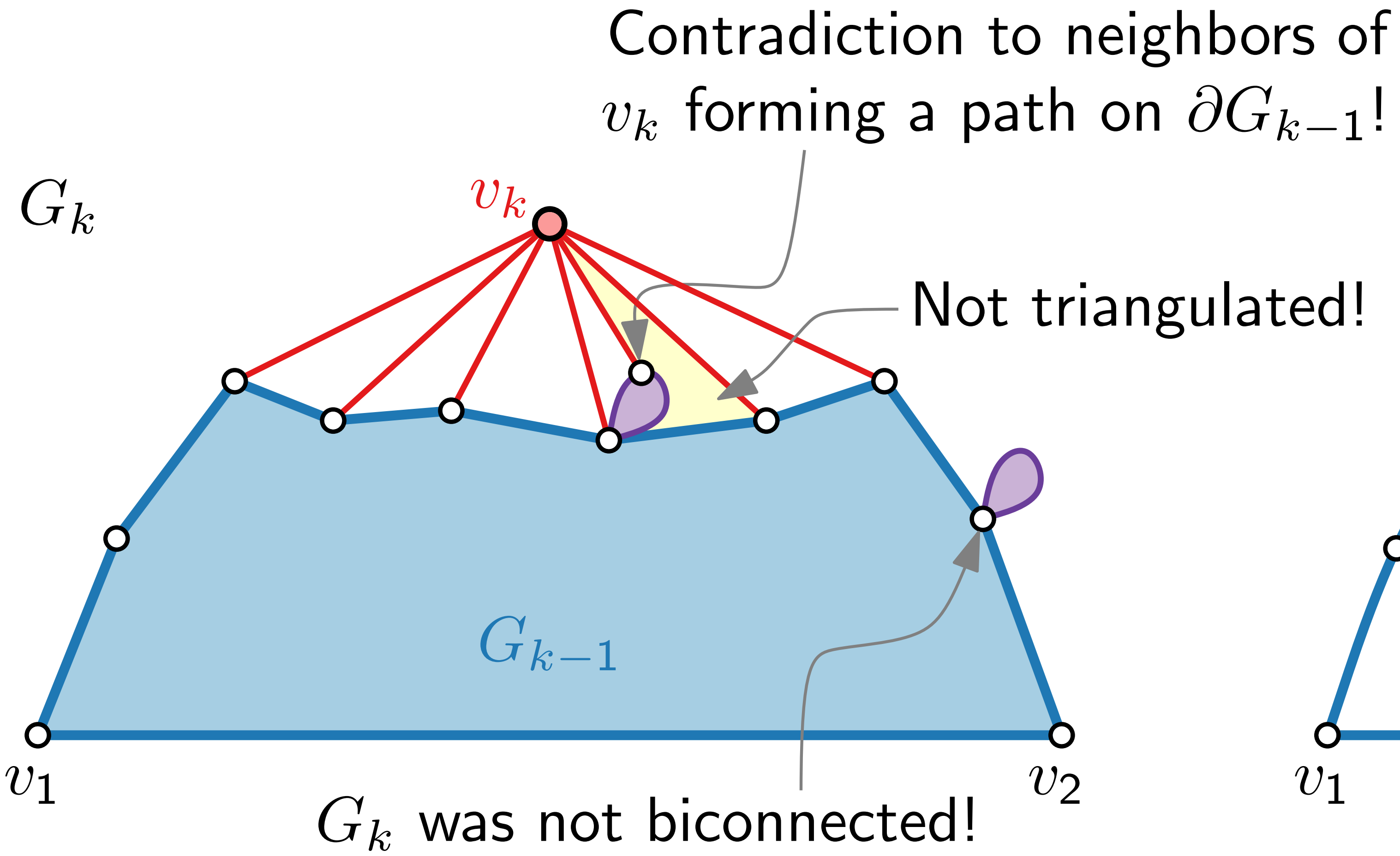
Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.

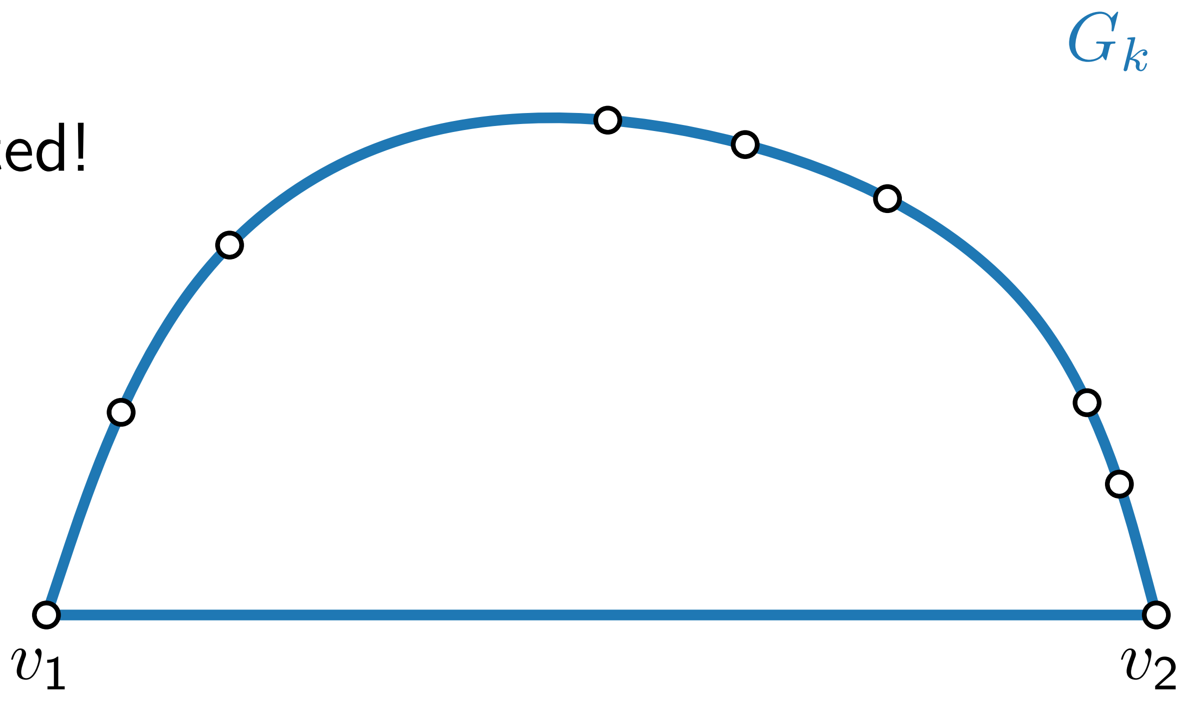


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord, then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

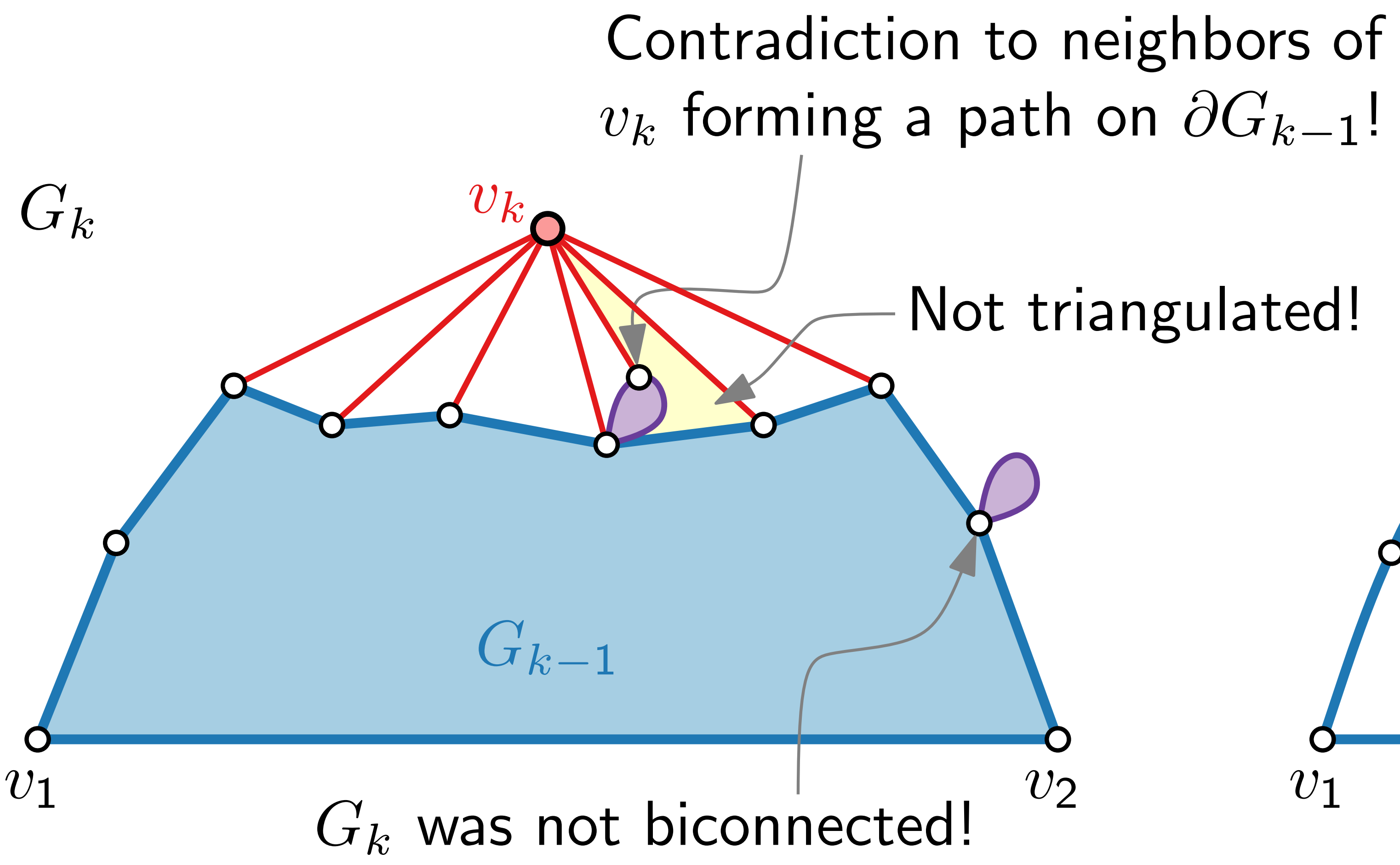


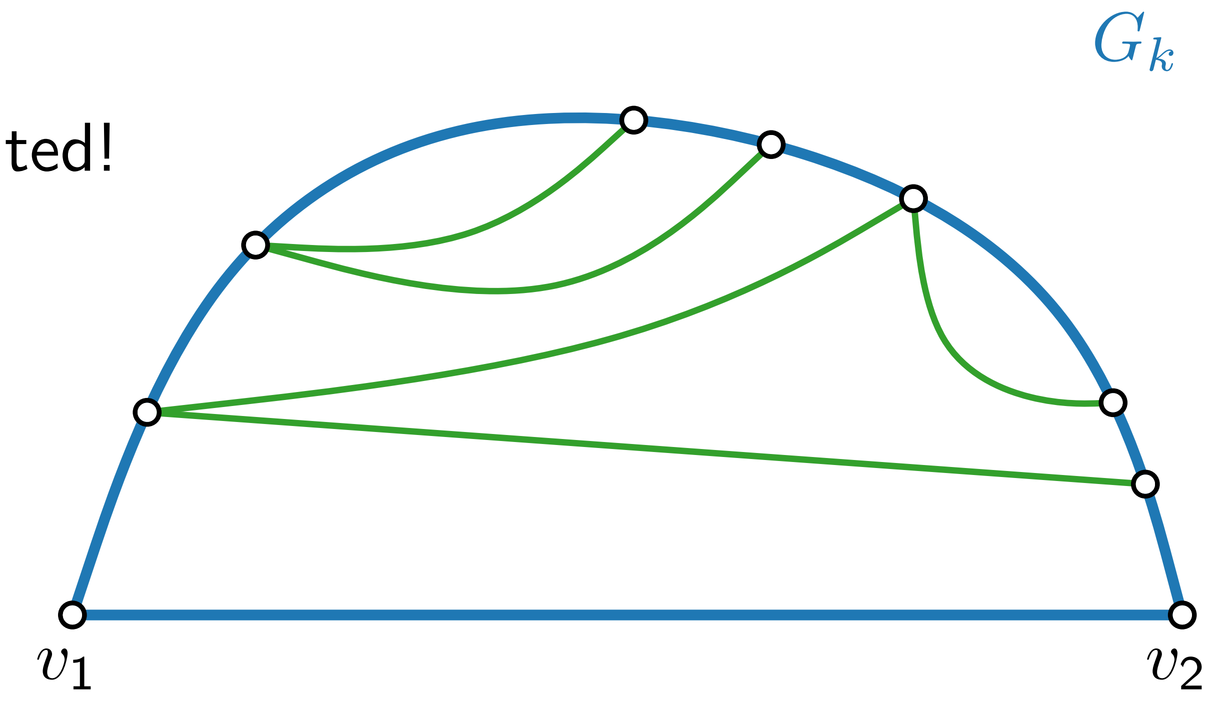


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord, then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

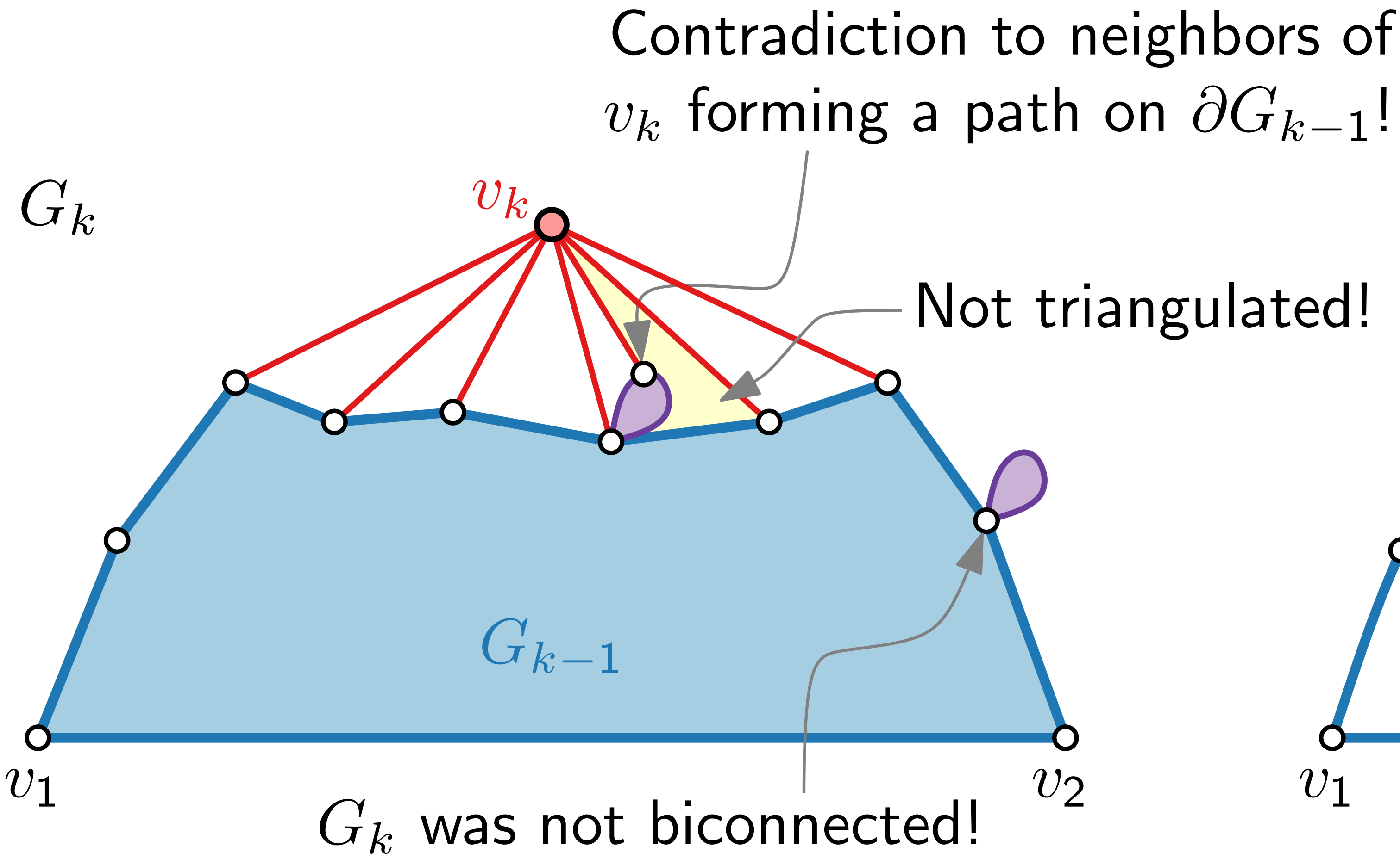


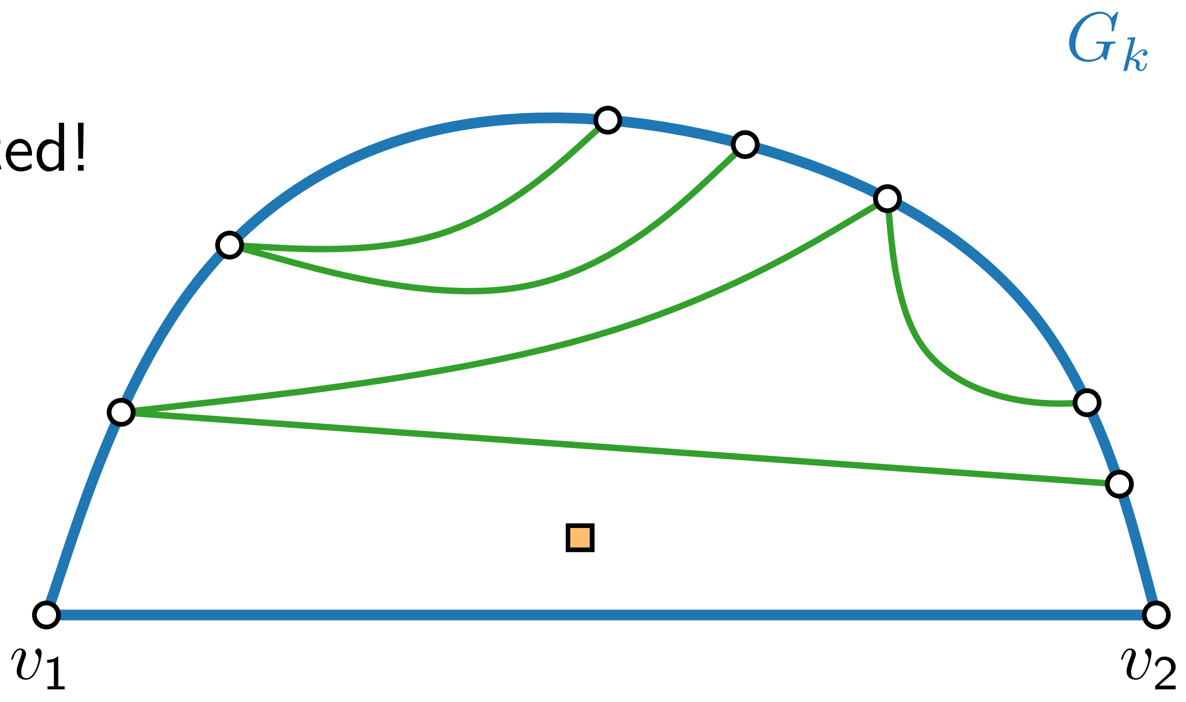


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord, then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

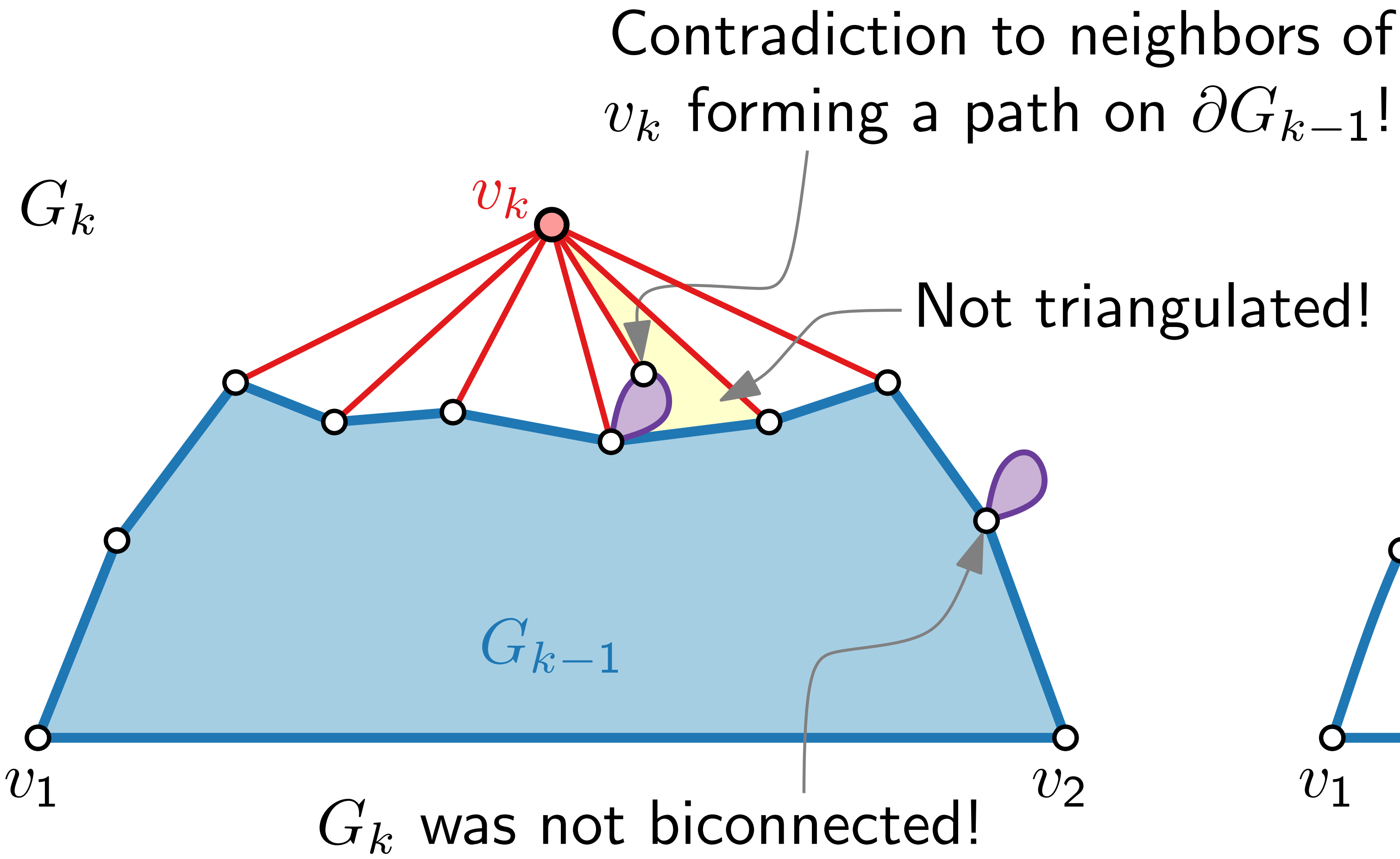


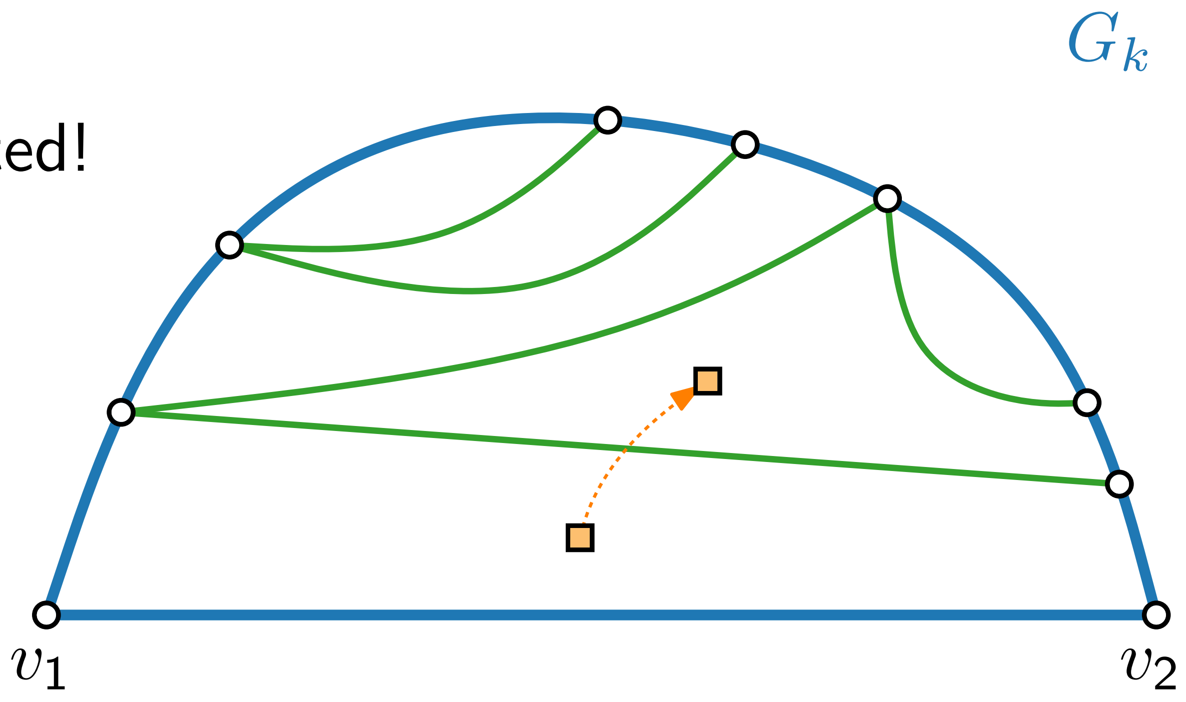


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord, then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

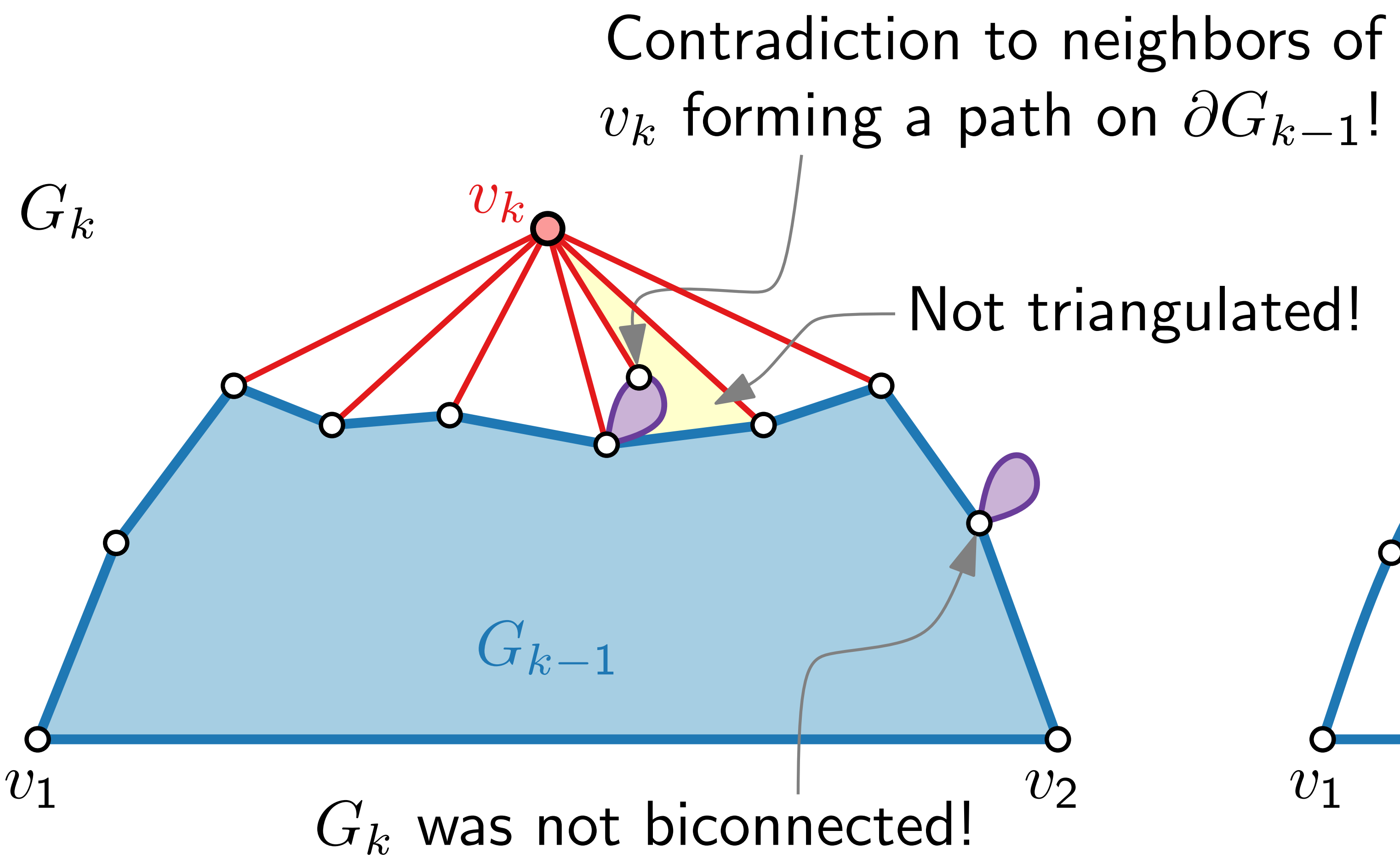


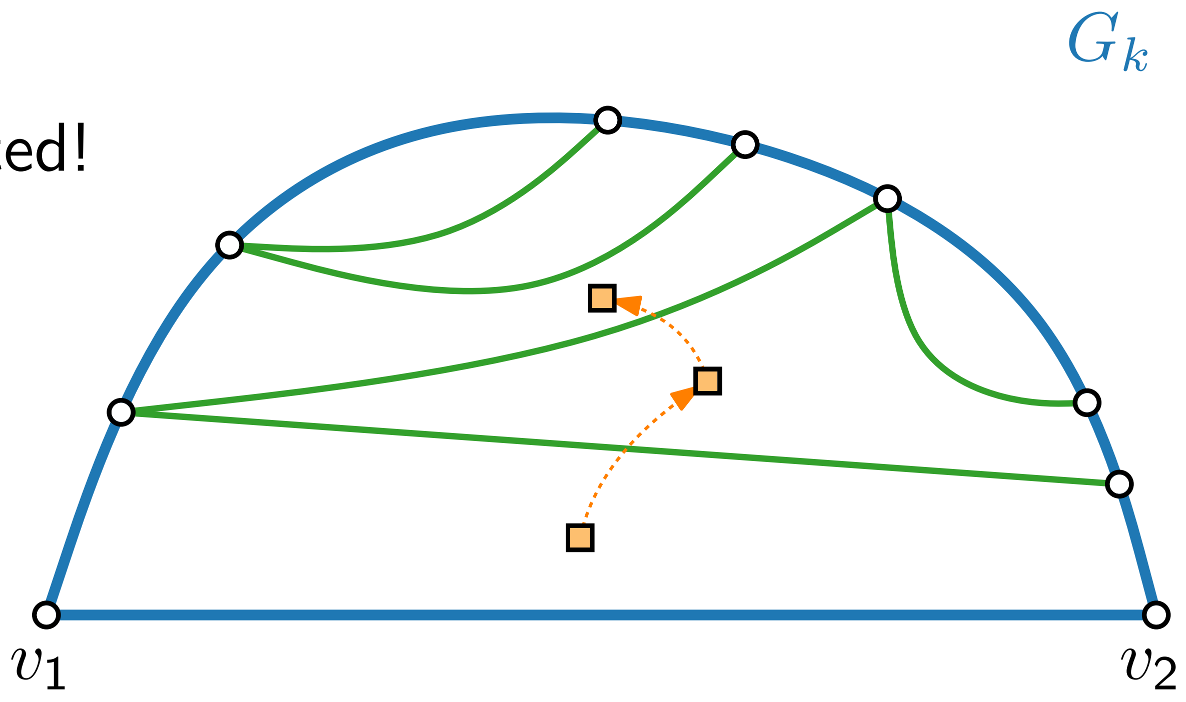


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord, then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

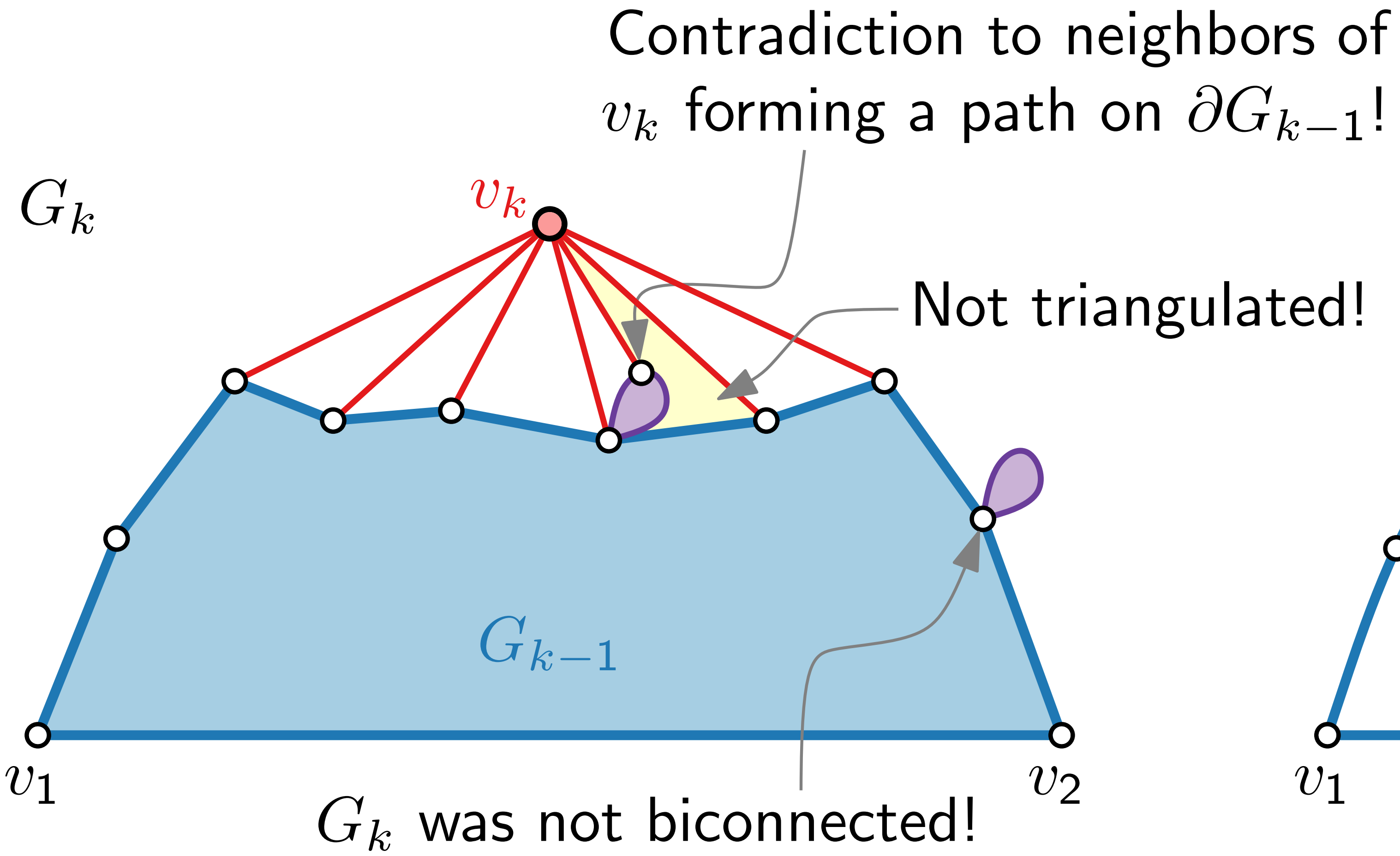


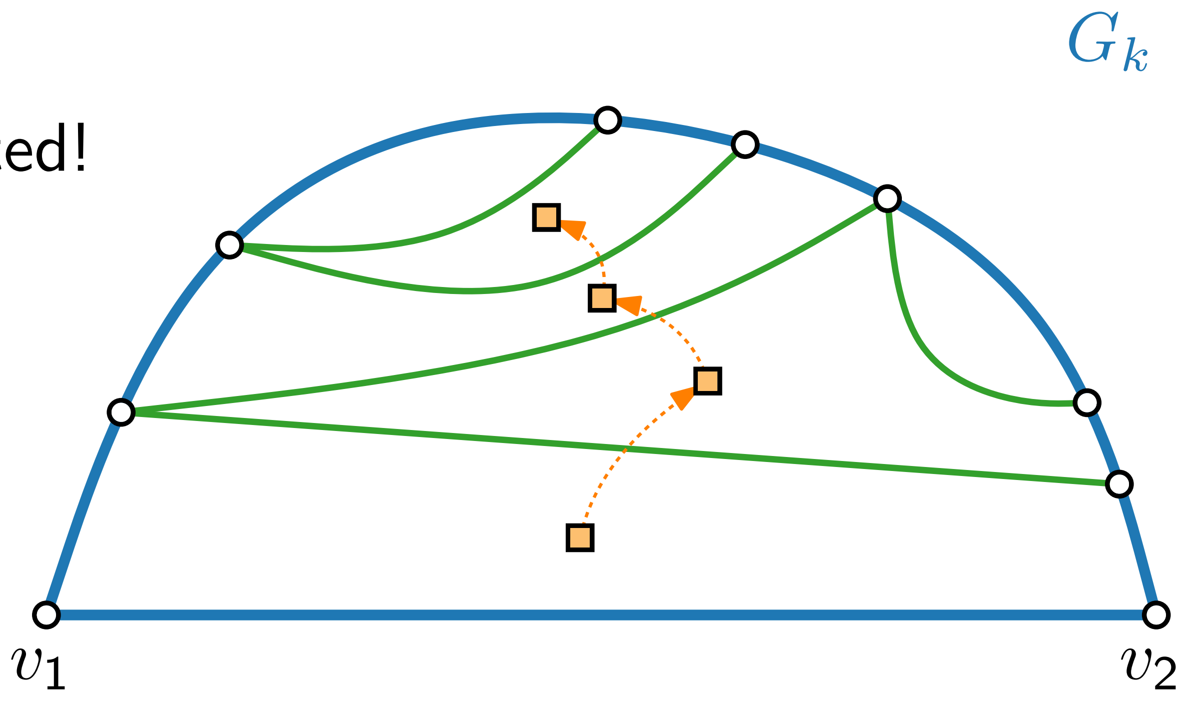


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord, then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

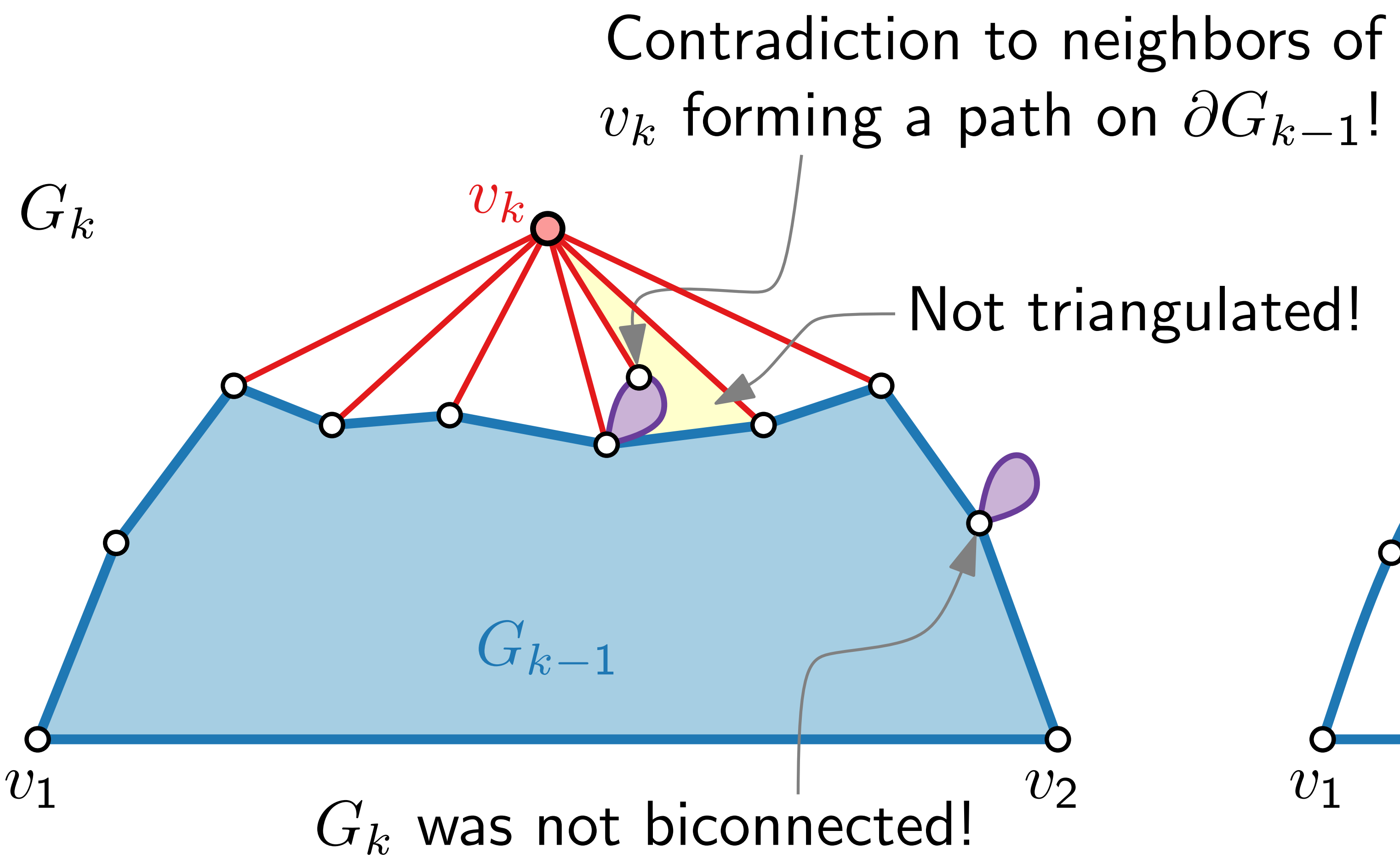


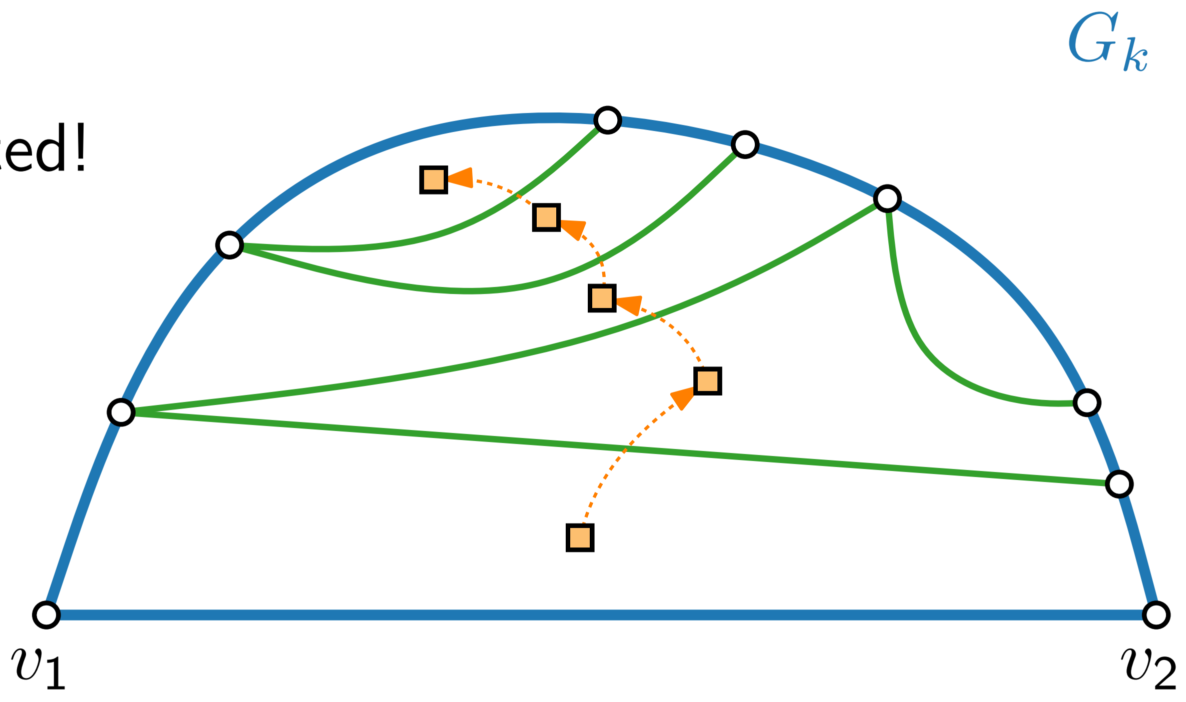


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord, then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

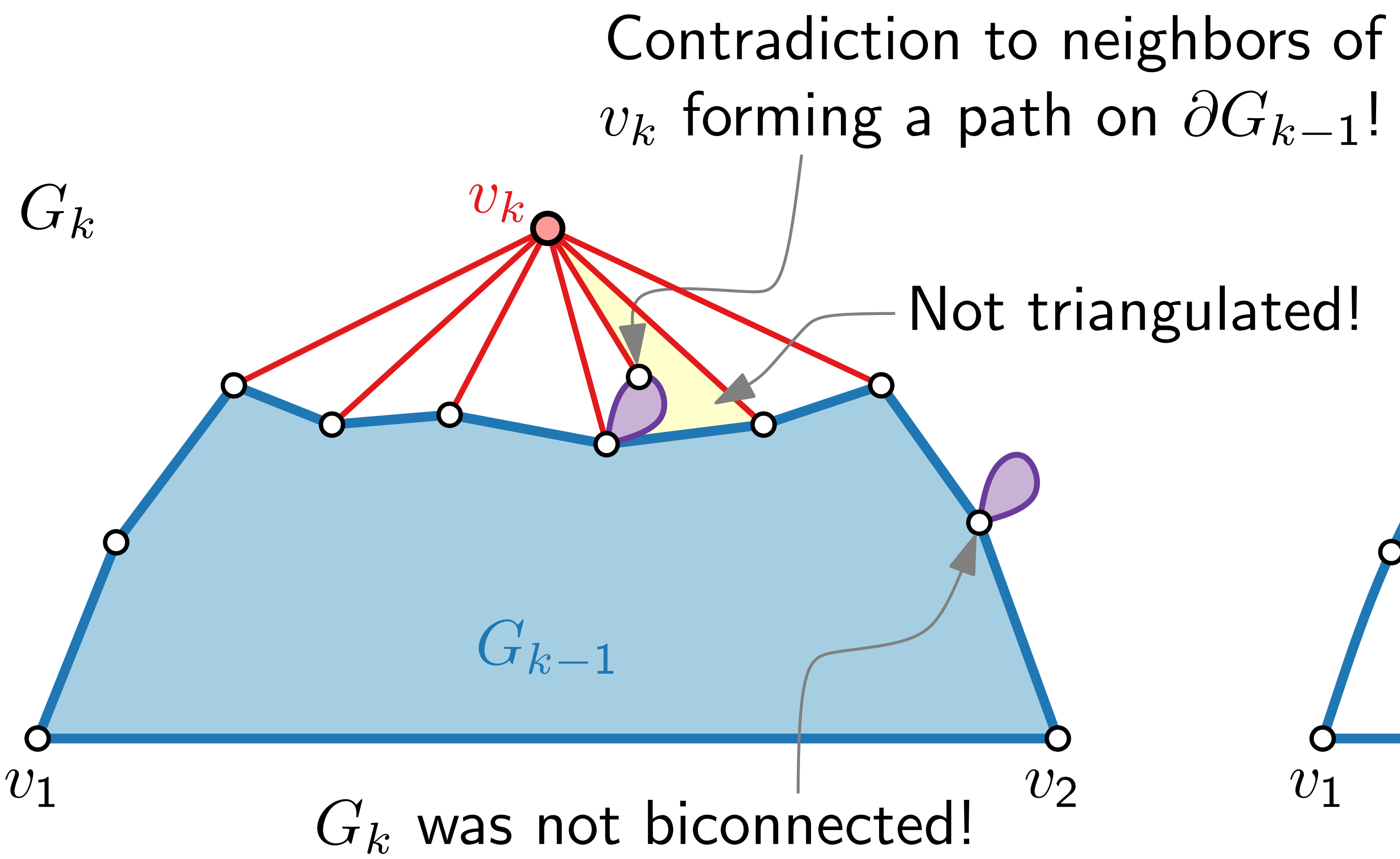


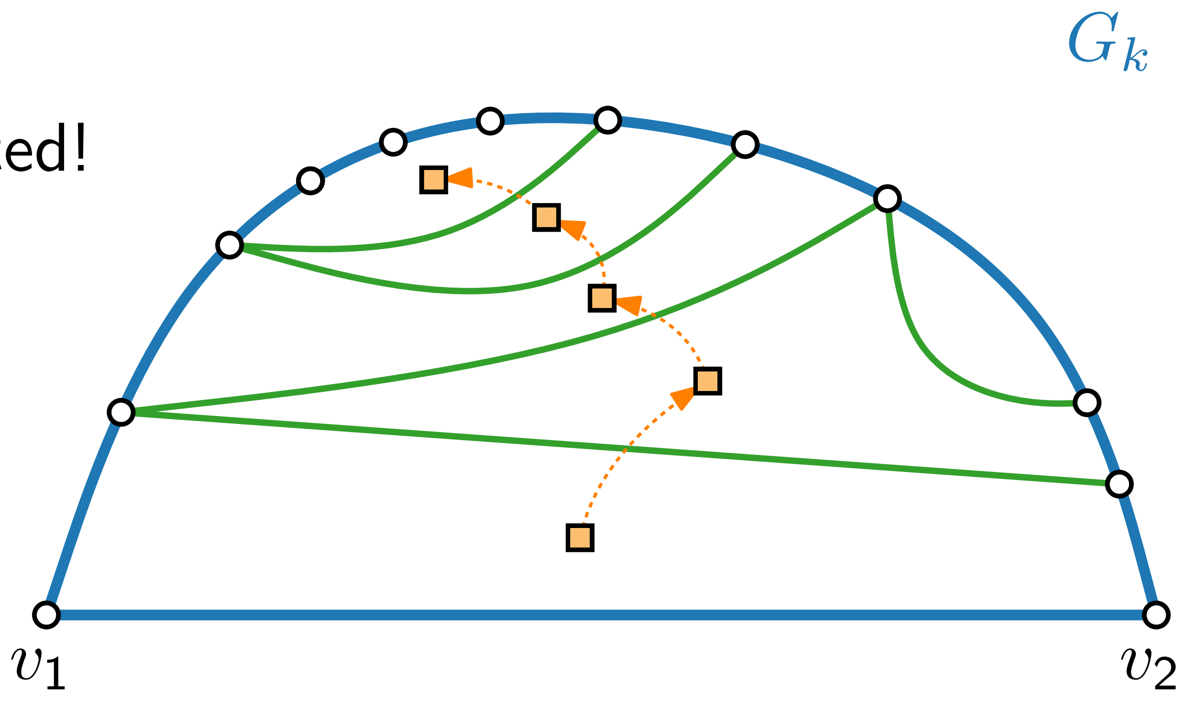


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord, then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

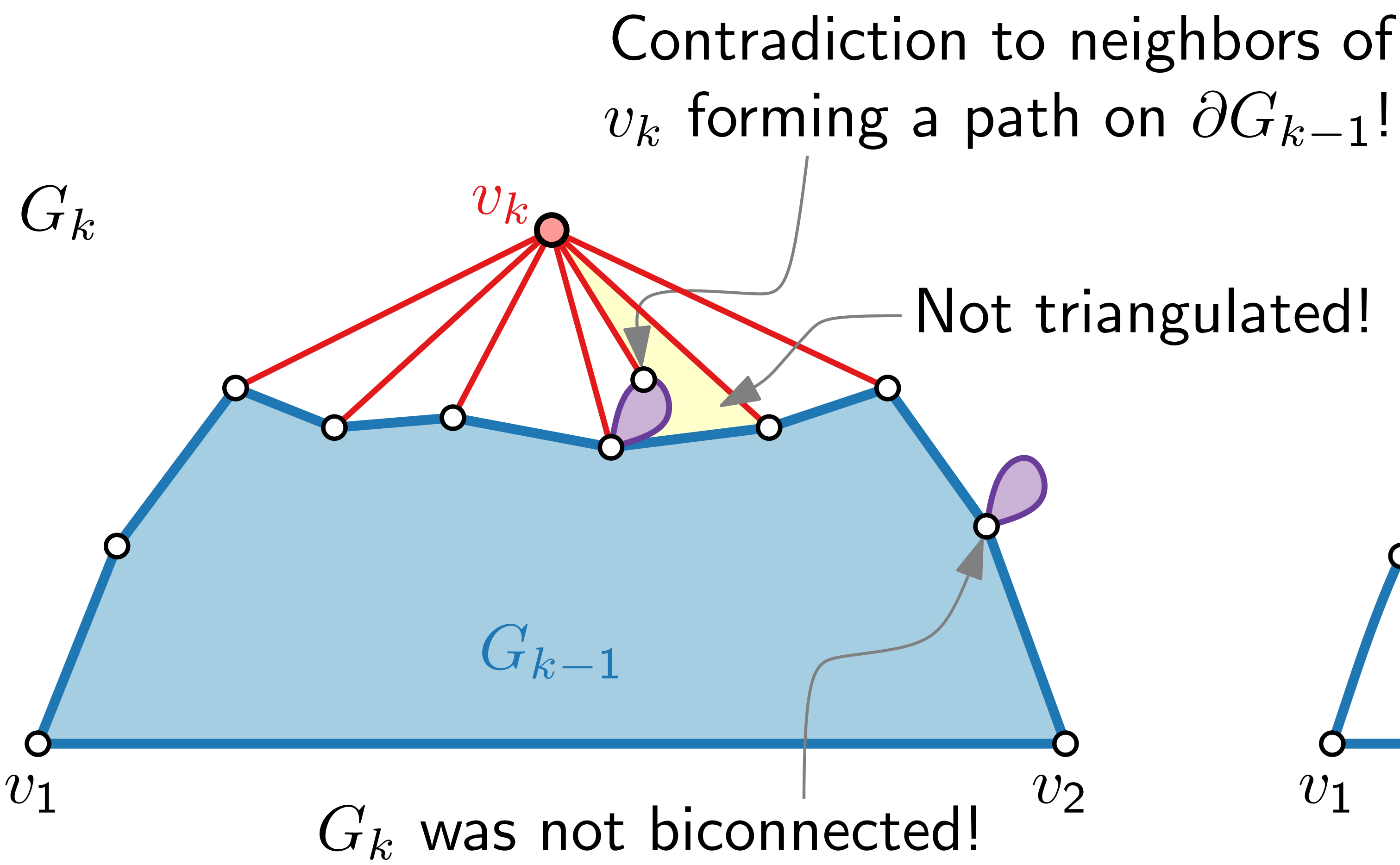


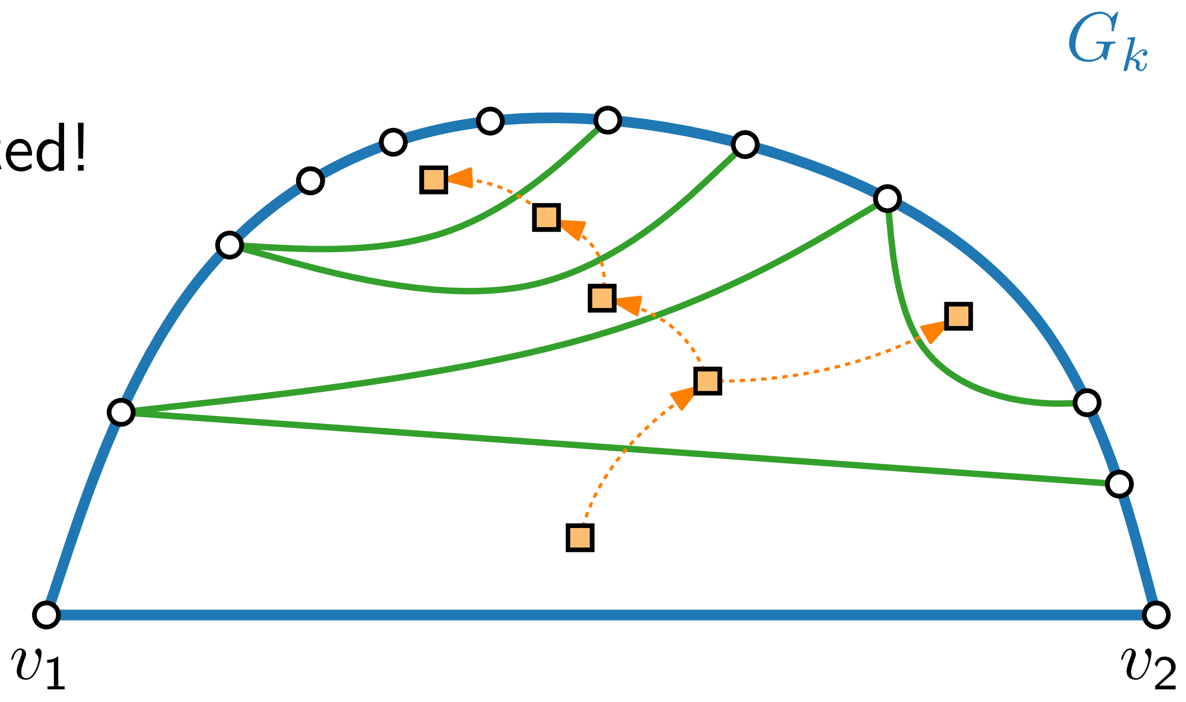


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord, then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

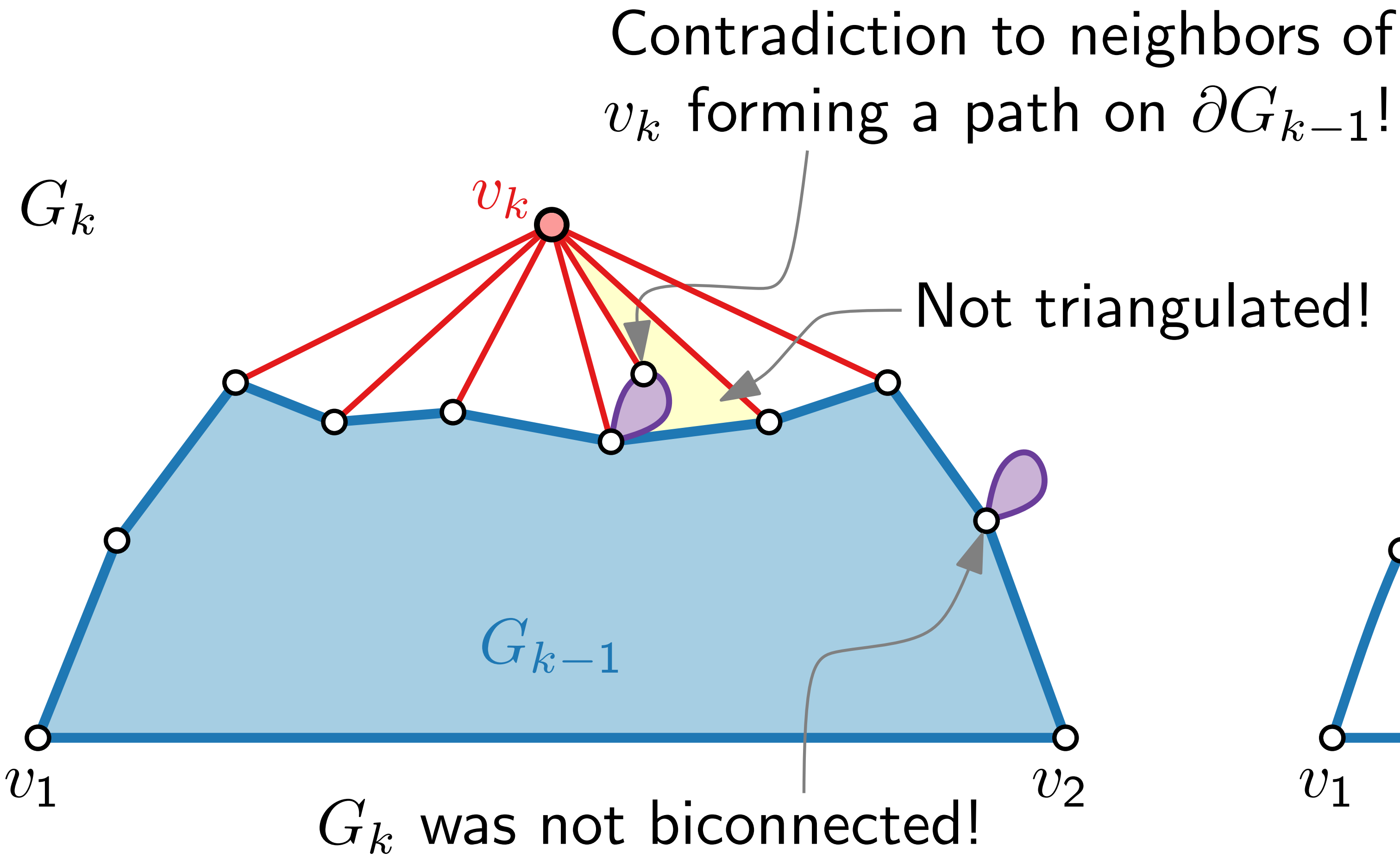


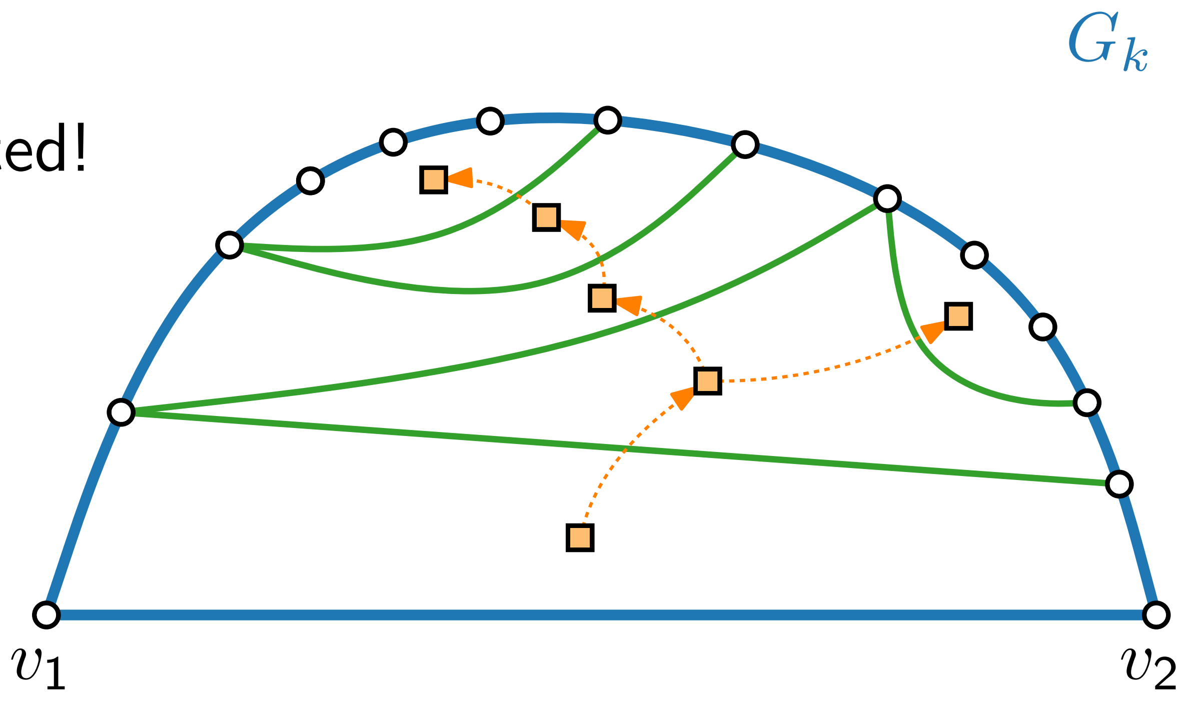


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

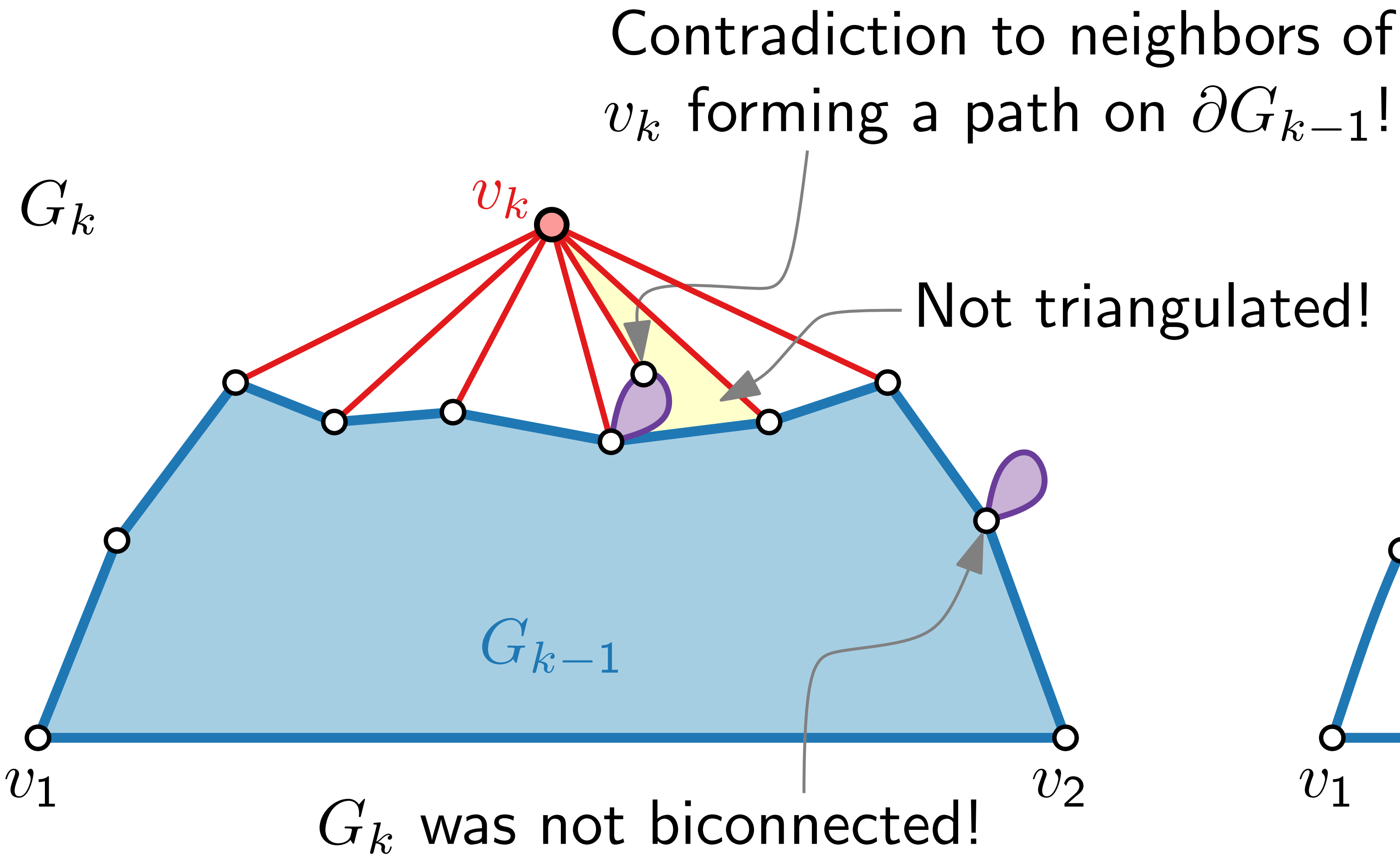


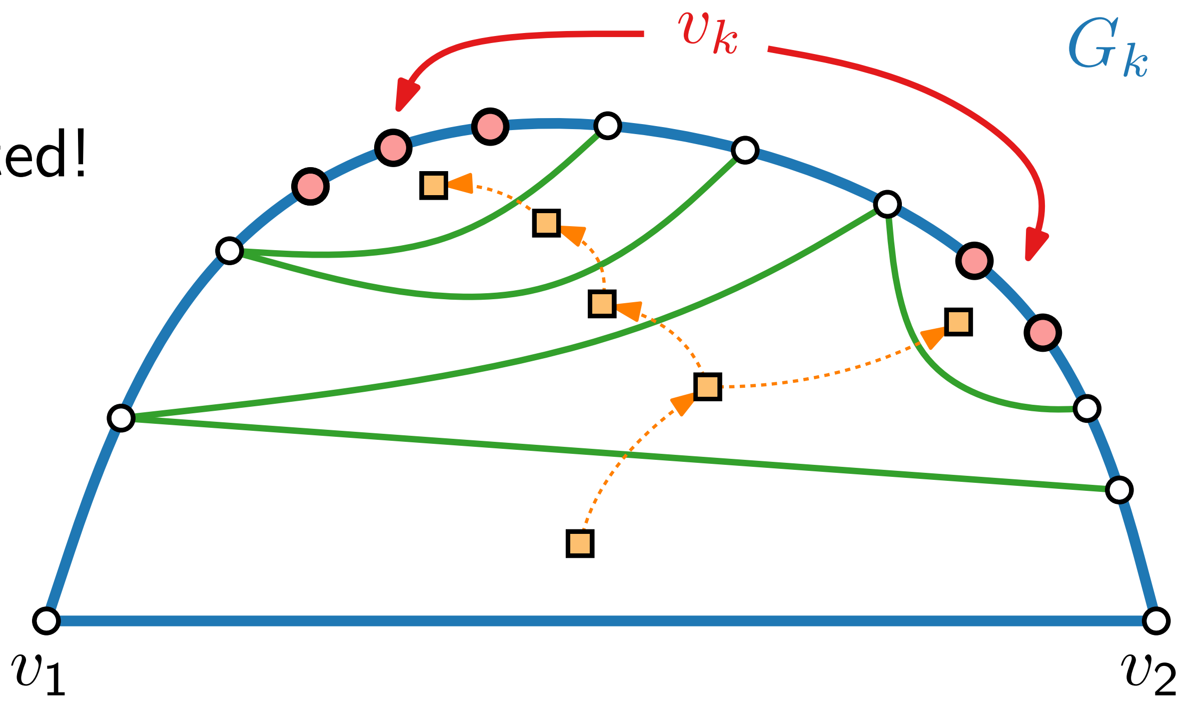


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord,
then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.

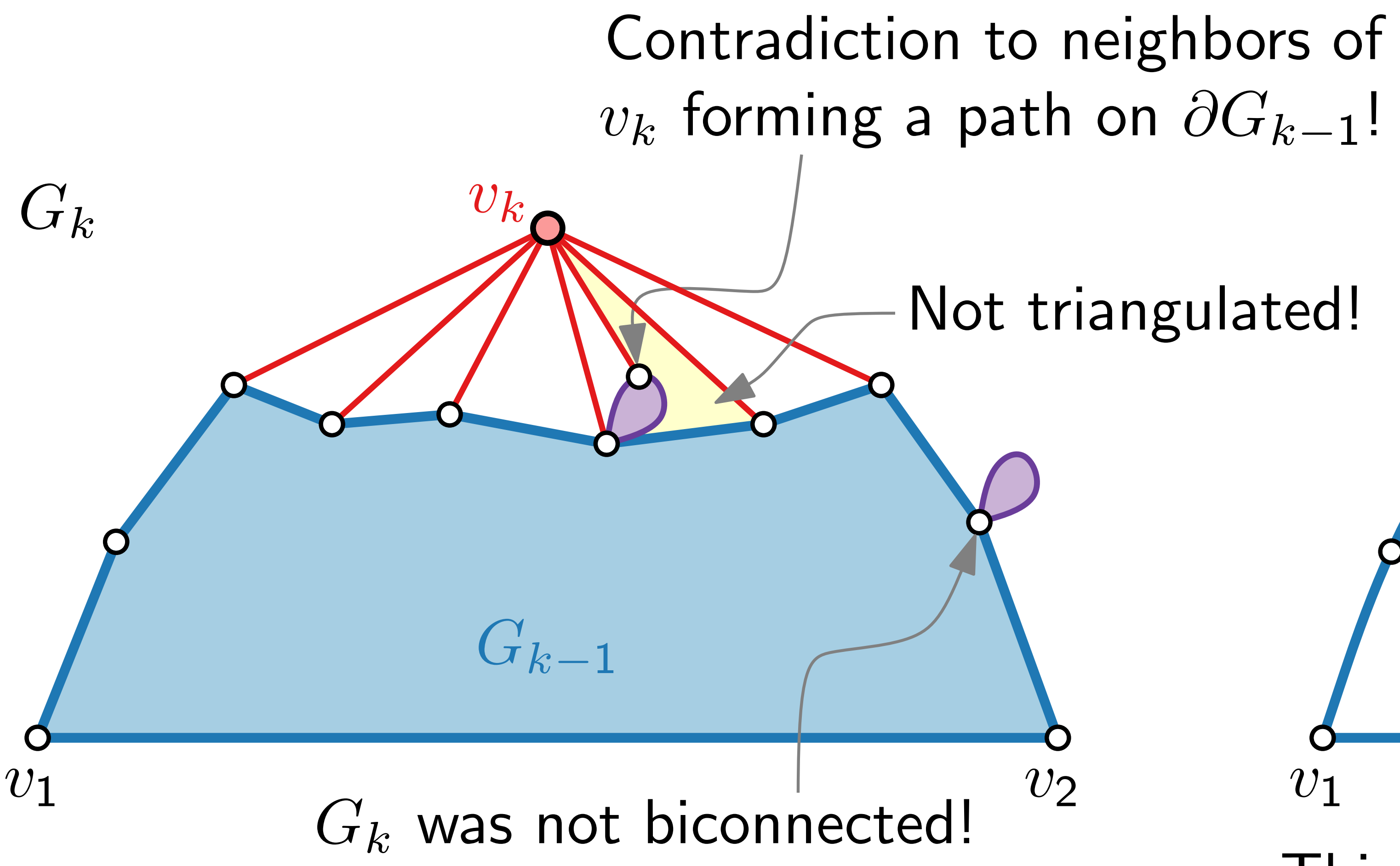


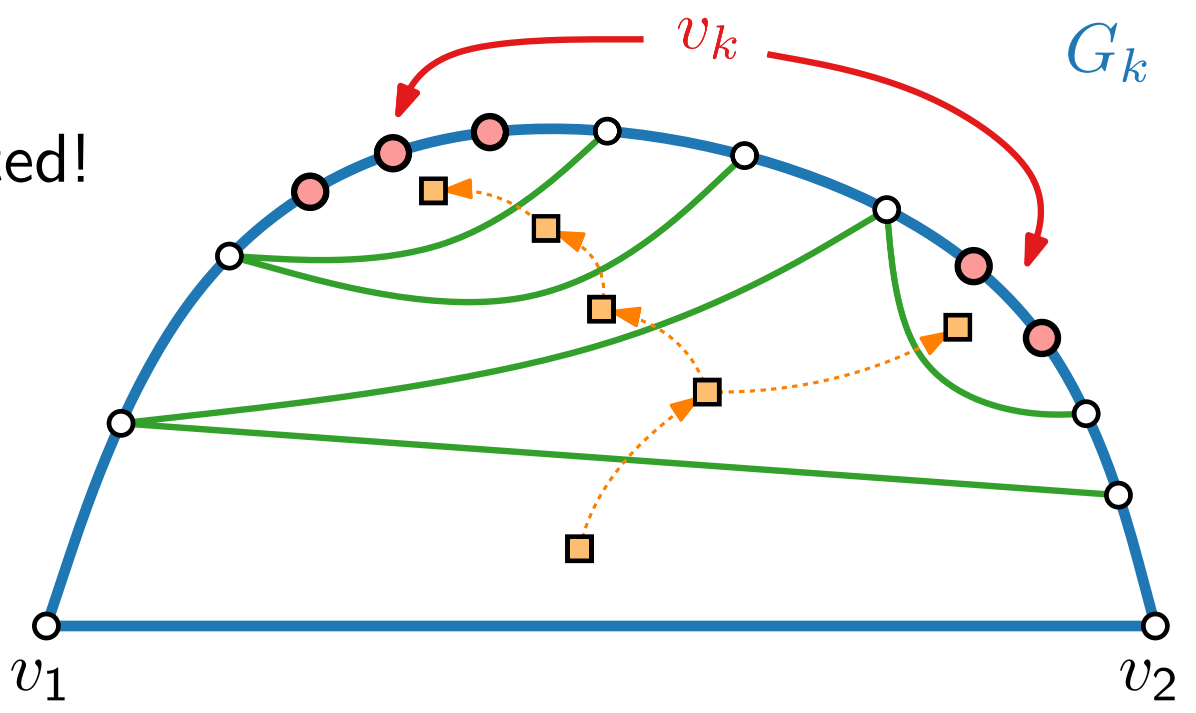


Canonical Order – Existence

Claim 1.
If v_k is not incident to a chord, then G_{k-1} is biconnected.

Claim 2.
There exists a vertex in G_k that is not incident to a chord as choice for v_k.





This completes the proof of the lemma. □

Canonical Order – Implementation

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow$ 0;

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow 0$;

- chord(v)= # chords incident to v

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false;

- chord(v)= # chords incident to v

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**

  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false;

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**

 chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
 chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

- chord(v)= # chords incident to v
- out(v) = true iff v on boundary of current outer face
- mark(v) = true iff v has received a number $\geq k$

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **downto** 3 **do**

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$

Canonical Order – Implementation

outer face

CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow 0$; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **downto** 3 **do**
  choose $v \in V(G) \setminus \{v_1, v_2\}$ such that mark(v) $=$ false, out(v) $=$ true, chords(v) $= 0$

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$

Canonical Order – Implementation

outer face

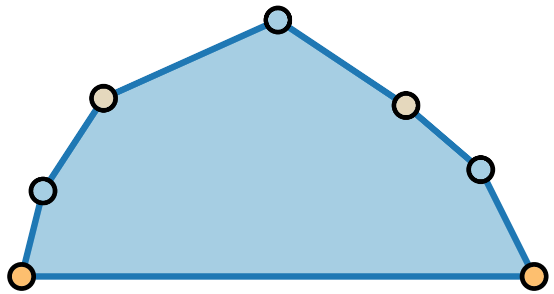
CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **downto** 3 **do**
choose $v \in V(G) \setminus \{v_1, v_2\}$ such that mark(v) $=$ false, out(v) $=$ true, chords(v) $=$ 0

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$



Canonical Order – Implementation

outer face

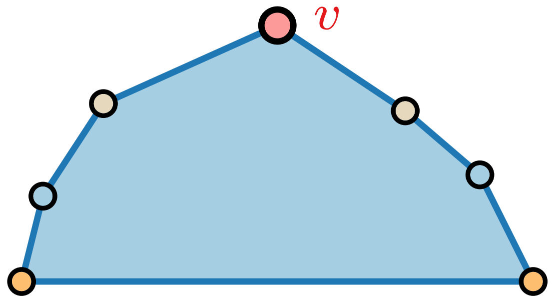
CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **downto** 3 **do**
  choose $v \in V(G) \setminus \{v_1, v_2\}$ such that mark(v) $=$ false, out(v) $=$ true, chords(v) $=$ 0

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$



Canonical Order – Implementation

outer face

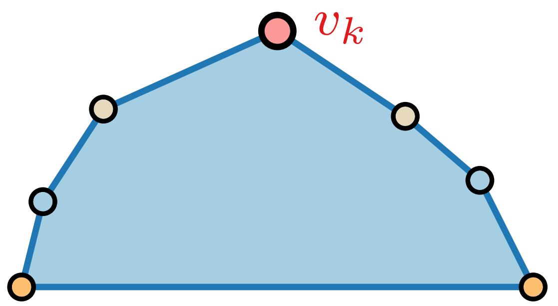
CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
- chords(v) $\leftarrow 0$; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **downto** 3 **do**
- choose $v \in V(G) \setminus \{v_1, v_2\}$ such that mark(v) $=$ false, out(v) $=$ true, chords(v) $= 0$
- $v_k \leftarrow v$; mark(v_k) $\leftarrow$ true; out(v_k) $\leftarrow$ false

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$



Canonical Order – Implementation

outer face

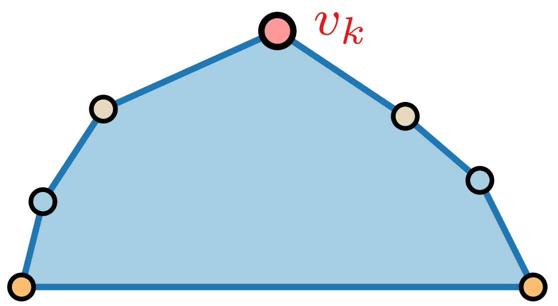
CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
- chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **downto** 3 **do**
- choose $v \in V(G) \setminus \{v_1, v_2\}$ such that mark(v) $=$ false, out(v) $=$ true, chords(v) $=$ 0
- $v_k \leftarrow v$; mark(v_k) $\leftarrow$ true; out(v_k) $\leftarrow$ false
- let $w_p, \ldots, w_q$ be the ordered unmarked neighbors of v_k

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$



Canonical Order – Implementation

outer face

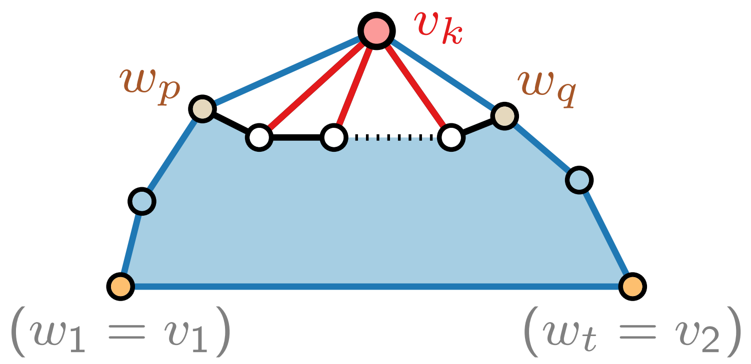
CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **downto** 3 **do**
  choose $v \in V(G) \setminus \{v_1, v_2\}$ such that mark(v) $=$ false, out(v) $=$ true, chords(v) $=$ 0
  $v_k \leftarrow v$; mark(v_k) $\leftarrow$ true; out(v_k) $\leftarrow$ false
  let $w_p, \ldots, w_q$ be the ordered unmarked neighbors of v_k

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$



Canonical Order – Implementation

outer face

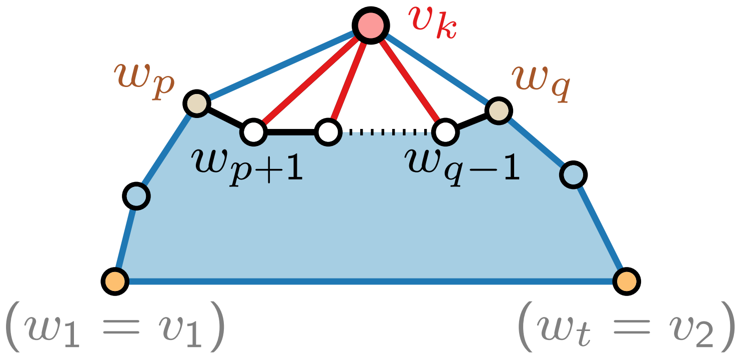
CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **downto** 3 **do**
  choose $v \in V(G) \setminus \{v_1, v_2\}$ such that mark(v) $=$ false, out(v) $=$ true, chords(v) $= 0$
  $v_k \leftarrow v$; mark(v_k) $\leftarrow$ true; out(v_k) $\leftarrow$ false
  let $w_p, \ldots, w_q$ be the ordered unmarked neighbors of v_k
  for $i = p + 1$ **to** $q - 1$ **do**

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$



Canonical Order – Implementation

outer face

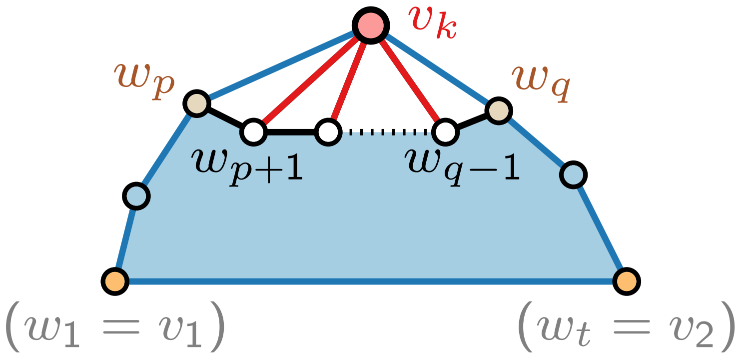
CanonicalOrder($G, \langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

out(v_1), out(v_2), out(v_n) $\leftarrow$ true

for $k = n$ **downto** 3 **do**
  choose $v \in V(G) \setminus \{v_1, v_2\}$ such that mark(v) $=$ false, out(v) $=$ true, chords(v) $=$ 0
  $v_k \leftarrow v$; mark(v_k) $\leftarrow$ true; out(v_k) $\leftarrow$ false
  let $w_p, \dots, w_q$ be the ordered unmarked neighbors of v_k
  for $i = p + 1$ **to** $q - 1$ **do**
    out(w_i) $\leftarrow$ true

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$

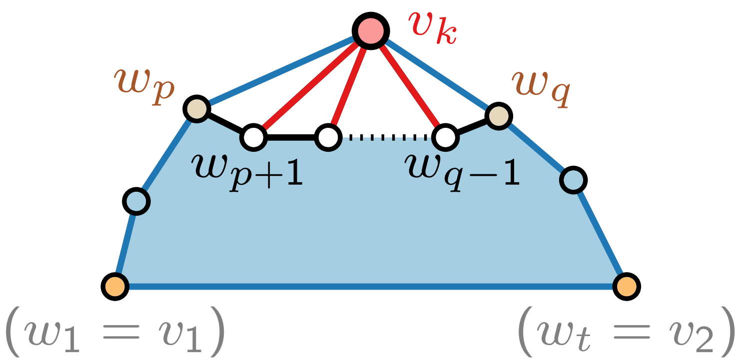


Canonical Order – Implementation

outer face

```
CanonicalOrder(G, ⟨v_1, v_2, v_n⟩)
foreach v ∈ V(G) do
  └ chords(v) ← 0; out(v) ← false; mark(v) ← false
out(v_1), out(v_2), out(v_n) ← true
for k = n downto 3 do
  | choose v ∈ V(G) \ {v_1, v_2} such that mark(v) = false,
  |   out(v) = true, chords(v) = 0
  | v_k ← v; mark(v_k) ← true; out(v_k) ← false
  | let w_p, ..., w_q be the ordered unmarked neighbors of v_k
  | for i = p + 1 to q − 1 do
  |   | out(w_i) ← true
  |   | foreach u ∈ Adj[w_i] \ {w_{i−1}, w_{i+1}} do
  |   |   └ if out(u) then chords(w_i)++, chords(u)++
  |   └
  └ if p + 1 = q then chords(w_p)--, chords(w_q)--
```

- chord(v)= # chords incident to v
- out(v) = true iff v on boundary of current outer face
- mark(v) = true iff v has received a number $\geq k$

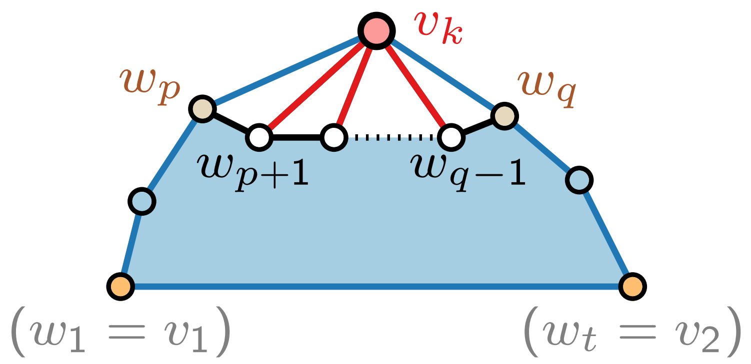


Canonical Order – Implementation

outer face

```
CanonicalOrder(G, ⟨v_1, v_2, v_n⟩)
foreach v ∈ V(G) do
  └ chords(v) ← 0; out(v) ← false; mark(v) ← false
out(v_1), out(v_2), out(v_n) ← true
for k = n downto 3 do
  | choose v ∈ V(G) \ {v_1, v_2} such that mark(v) = false,
  |   out(v) = true, chords(v) = 0
  | v_k ← v; mark(v_k) ← true; out(v_k) ← false
  | let w_p, ..., w_q be the ordered unmarked neighbors of v_k
  | for i = p + 1 to q − 1 do
  |   | out(w_i) ← true
  |   | foreach u ∈ Adj[w_i] \ {w_{i−1}, w_{i+1}} do
  |   |   └ if out(u) then chords(w_i)++, chords(u)++
  |   └
  └ if p + 1 = q then chords(w_p)−−, chords(w_q)−−
```

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$



Lemma.
Algorithm CanonicalOrder computes a canonical order of a plane graph in $\mathcal{O}(n)$ time.

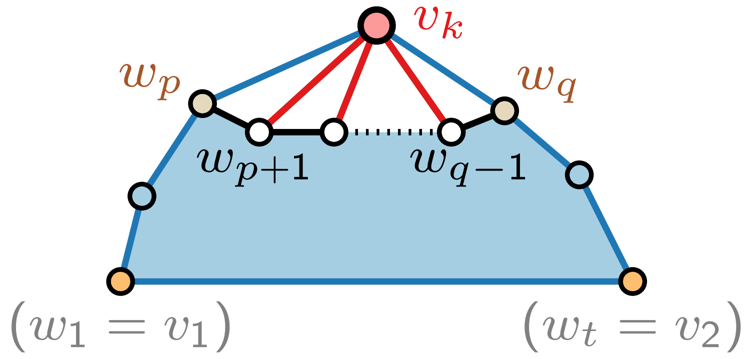
Canonical Order – Implementation

outer face

```
CanonicalOrder(G, ⟨v_1, v_2, v_n⟩)

foreach v ∈ V(G) do
  chords(v) ← 0; out(v) ← false; mark(v) ← false
out(v_1), out(v_2), out(v_n) ← true
for k = n downto 3 do
    choose v ∈ V(G) \ {v_1, v_2} such that mark(v) = false,
     out(v) = true, chords(v) = 0  // use list of candidates
    v_k ← v; mark(v_k) ← true; out(v_k) ← false
    let w_p, ..., w_q be the ordered unmarked neighbors of v_k
    for i = p + 1 to q − 1 do
        out(w_i) ← true
        foreach u ∈ Adj[w_i] \ {w_{i−1}, w_{i+1}} do
            if out(u) then chords(w_i)++, chords(u)++
    if p + 1 = q then chords(w_p)−−, chords(w_q)−−
```

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$



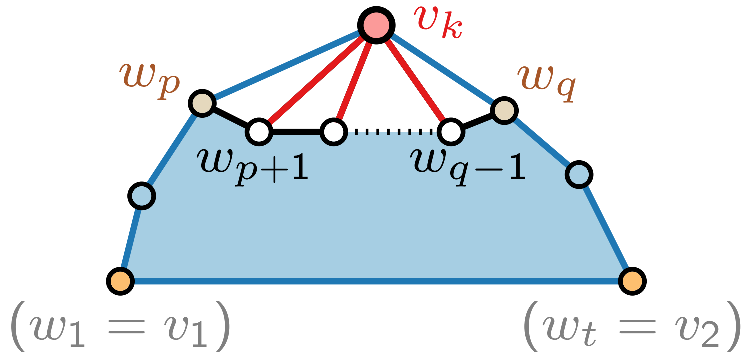
Lemma.
Algorithm CanonicalOrder computes a canonical order of a plane graph in $\mathcal{O}(n)$ time.

Canonical Order – Implementation

outer face

```
CanonicalOrder(G, ⟨v1, v2, vn⟩)
foreach v ∈ V(G) do
  chords(v) ← 0; out(v) ← false; mark(v) ← false
out(v1), out(v2), out(vn) ← true
for k = n downto 3 do
    choose v ∈ V(G) \ {v1, v2} such that mark(v) = false,
     out(v) = true, chords(v) = 0 // use list of candidates
    vk ← v; mark(vk) ← true; out(vk) ← false
    let wp, ..., wq be the ordered unmarked neighbors of vk
    for i = p + 1 to q − 1 do   // O(n) time in total
        out(wi) ← true
        foreach u ∈ Adj[wi] \ {wi−1, wi+1} do
            if out(u) then chords(wi)++, chords(u)++
    if p + 1 = q then chords(wp)--, chords(wq)--
```

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$



Lemma.
Algorithm CanonicalOrder computes a canonical order of a plane graph in $\mathcal{O}(n)$ time.

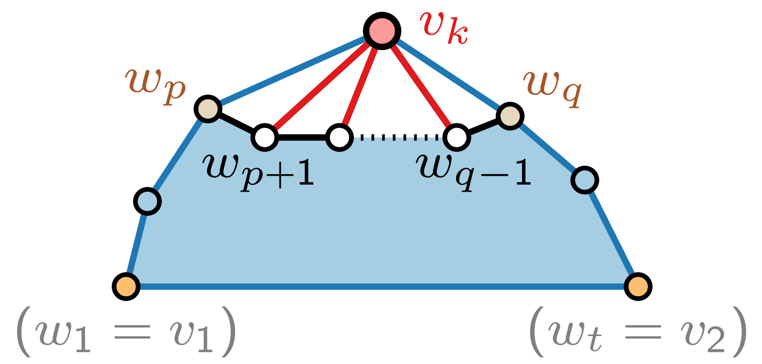
Canonical Order – Implementation

outer face

CanonicalOrder(G, $\langle v_1, v_2, v_n \rangle$)

foreach $v \in V(G)$ **do**
  chords(v) $\leftarrow 0$; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false
out(v_1), out(v_2), out(v_n) $\leftarrow$ true
for $k = n$ **downto** 3 **do**
  choose $v \in V(G) \setminus \{v_1, v_2\}$ such that mark(v) $=$ false,
   out(v) $=$ true, chords(v) $= 0$ // *use list of candidates*
  $v_k \leftarrow v$; mark(v_k) $\leftarrow$ true; out(v_k) $\leftarrow$ false
  let $w_p, \dots, w_q$ be the ordered unmarked neighbors of v_k
  for $i = p + 1$ **to** $q - 1$ **do** // *$O(n)$ time in total*
    out(w_i) $\leftarrow$ true // *$O(m) = O(n)$ in total*
    foreach $u \in$ Adj$[w_i] \setminus \{w_{i-1}, w_{i+1}\}$ **do**
      if out(u) **then** chords(w_i)++, chords(u)++
  if $p + 1 = q$ **then** chords(w_p)−−, chords(w_q)−−

- chord(v)= # chords incident to v
- out(v) $=$ true iff v on boundary of current outer face
- mark(v) $=$ true iff v has received a number $\geq k$



Lemma.
Algorithm CanonicalOrder computes a canonical order of a plane graph in $\mathcal{O}(n)$ time.

Shift Method – Idea

Drawing invariants:

G_k is drawn such that

G_k

Shift Method – Idea

Drawing invariants:

G_k is drawn such that

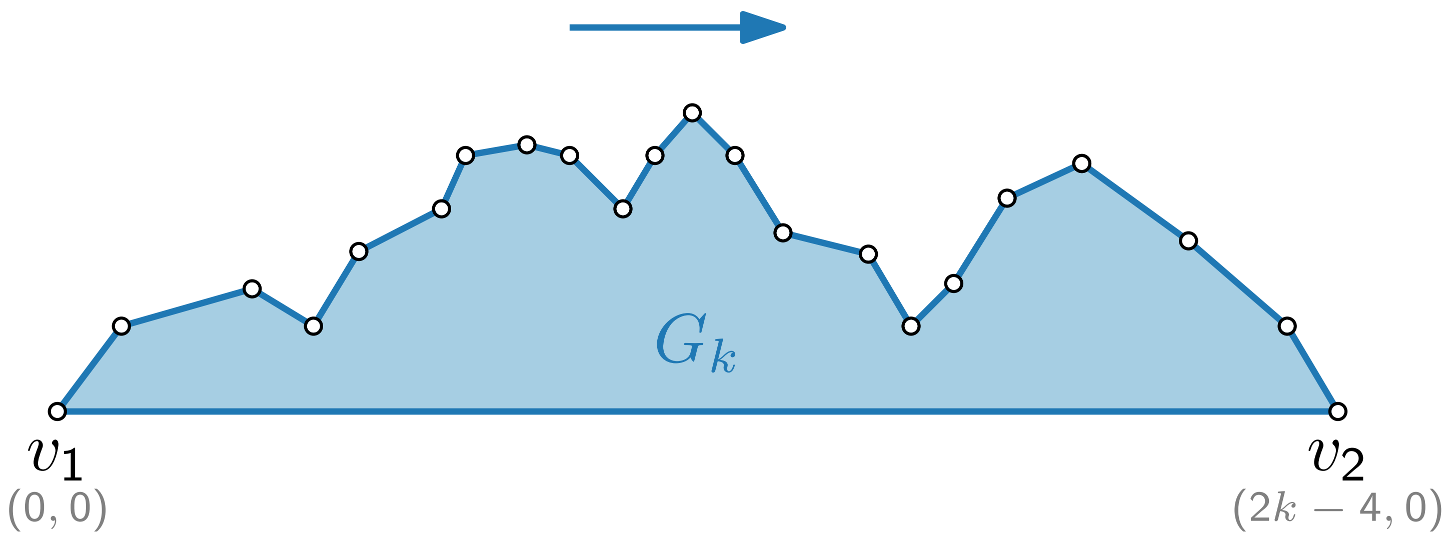
- v_1 is at $(0, 0)$, v_2 is at $(2k-4, 0)$,

Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0, 0)$, v_2 is at $(2k - 4, 0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,

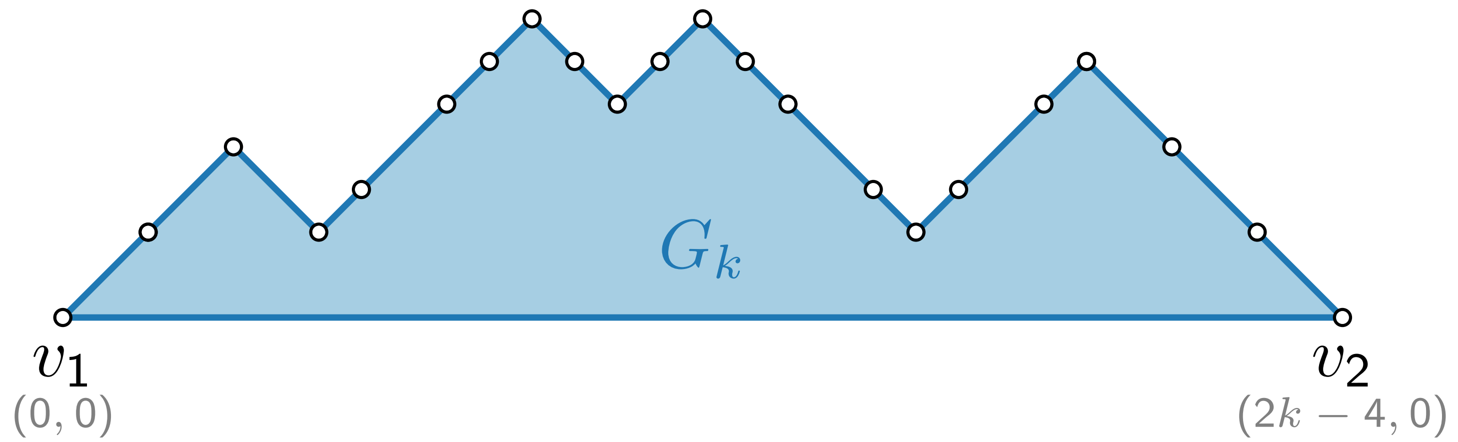


Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0,0)$, v_2 is at $(2k-4,0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

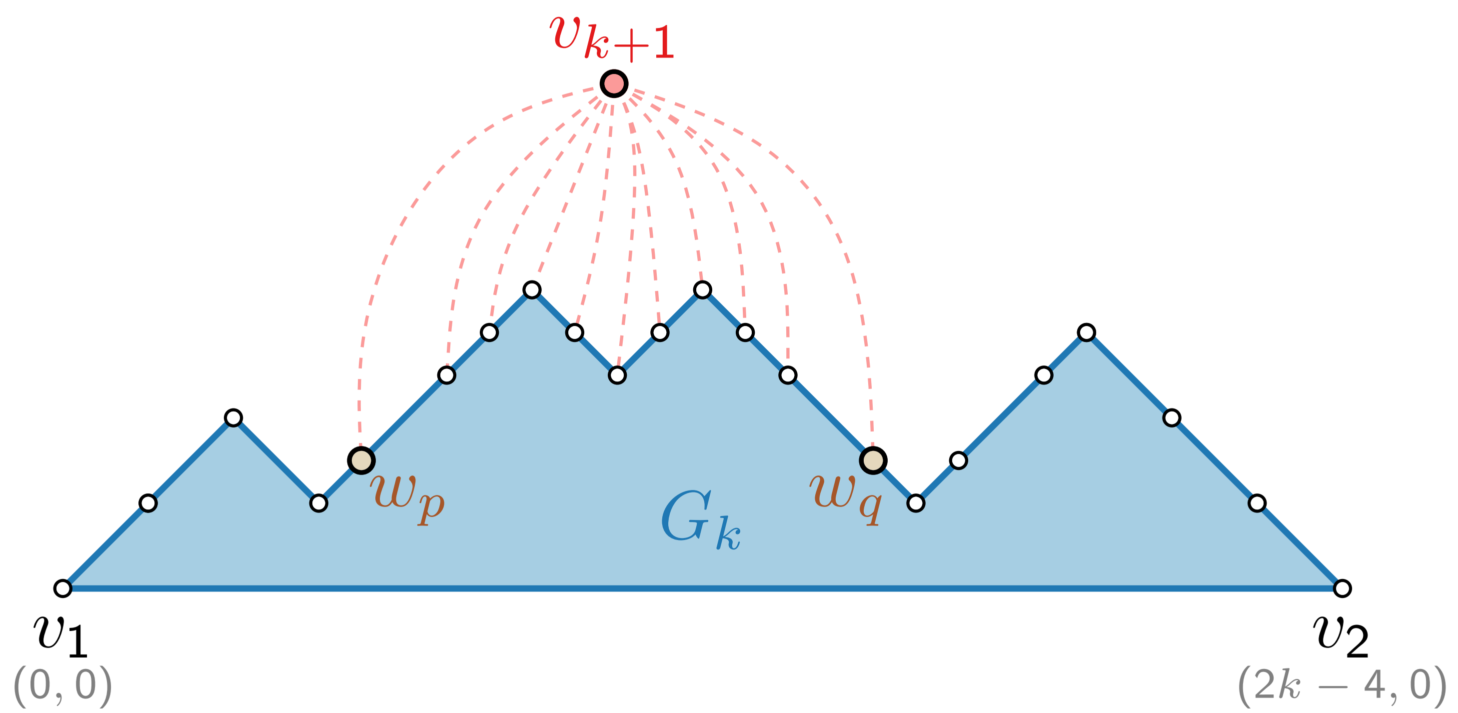


Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0, 0)$, v_2 is at $(2k - 4, 0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

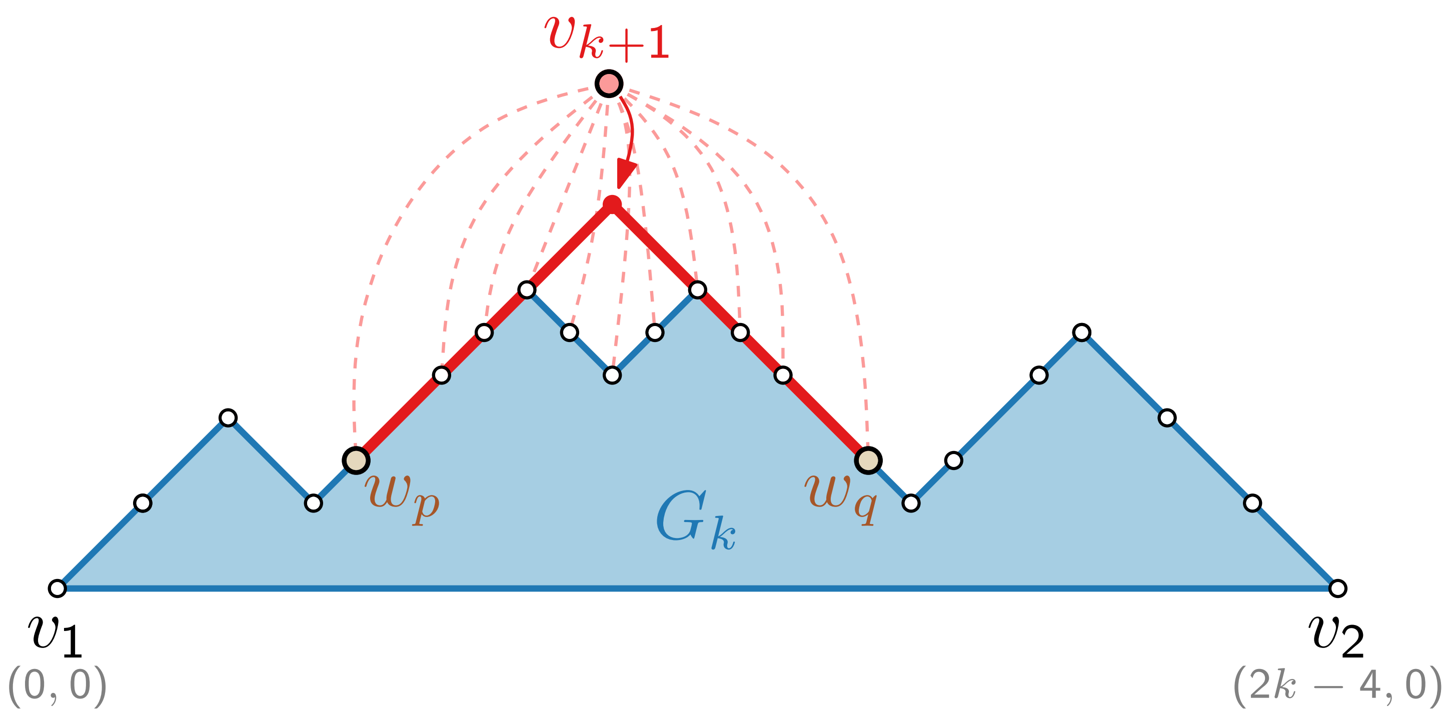


Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0, 0)$, v_2 is at $(2k - 4, 0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

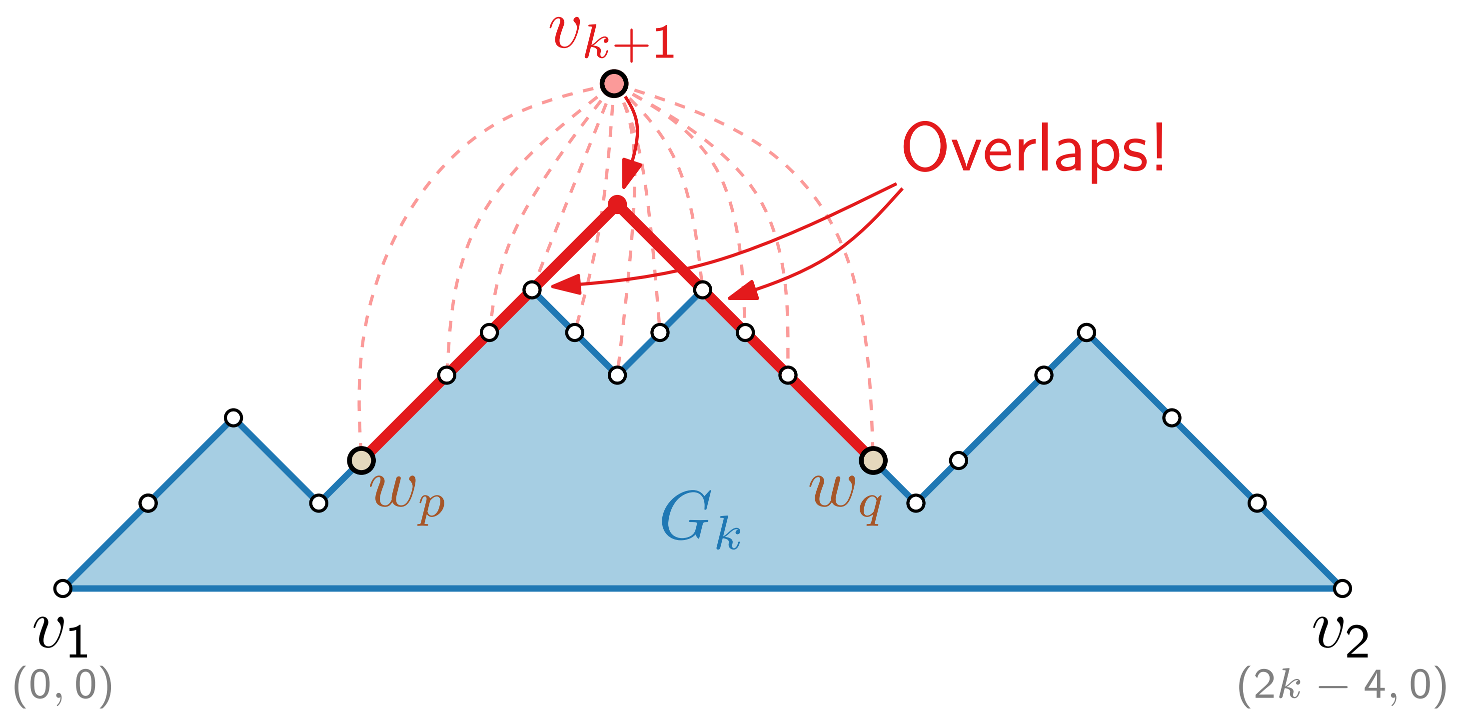


Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0,0)$, v_2 is at $(2k-4,0)$,
- boundary of G_k (minus edge $\{v_1,v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1,v_2\}$) is drawn with slopes ± 1.

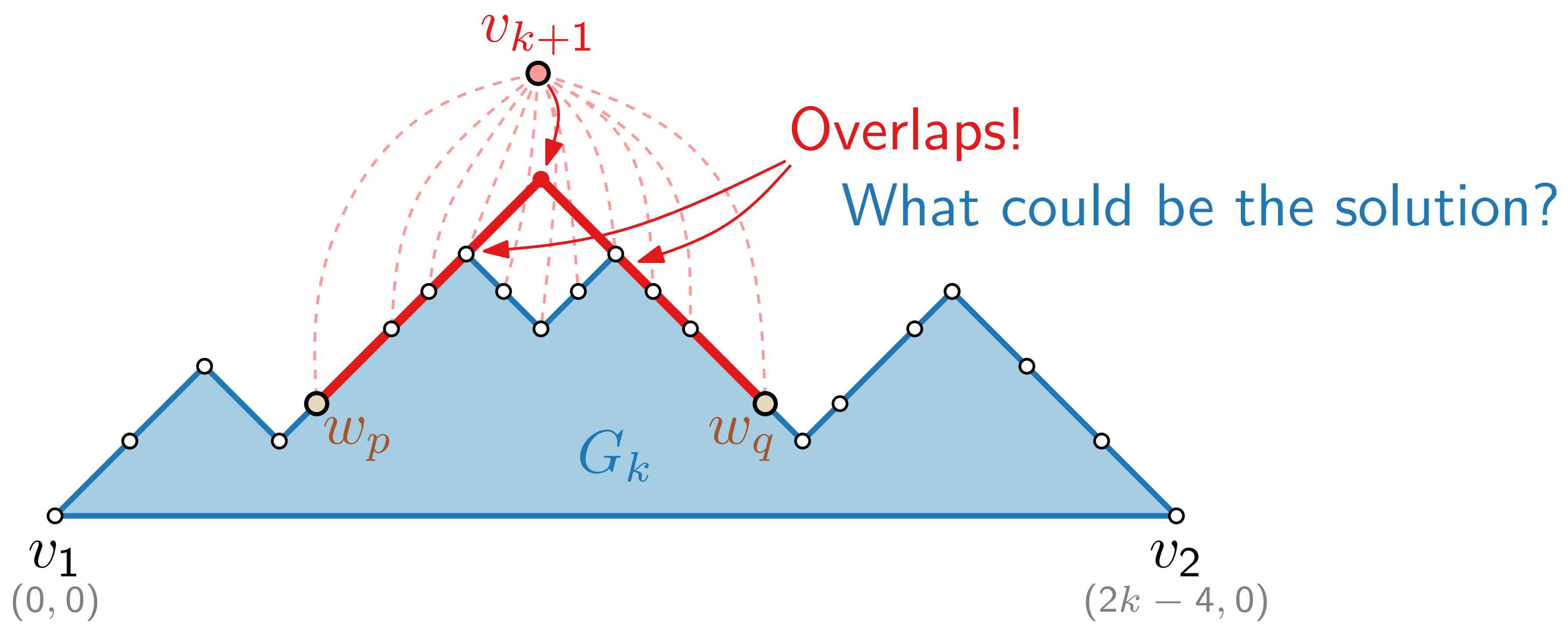


Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0,0)$, v_2 is at $(2k-4,0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

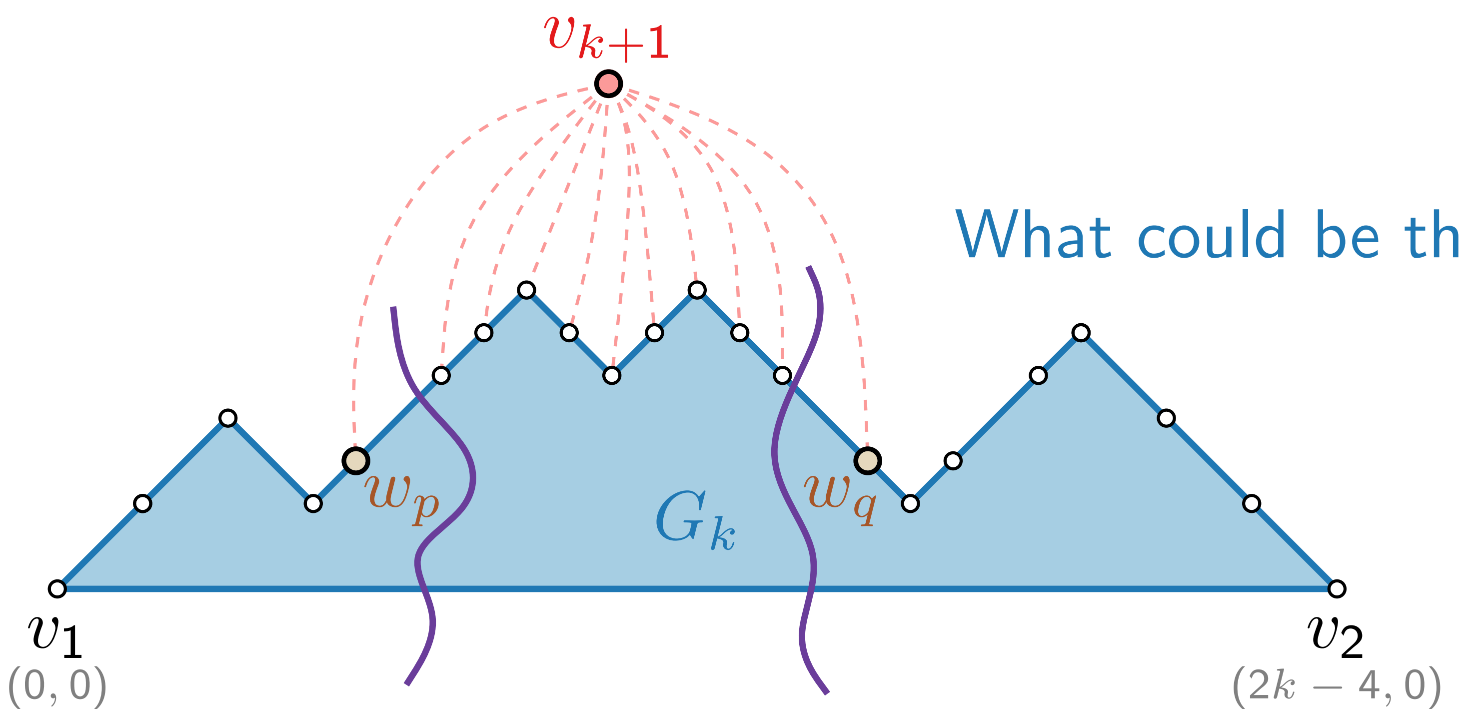


Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0,0)$, v_2 is at $(2k-4,0)$,
- boundary of G_k (minus edge $\{v_1,v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1,v_2\}$) is drawn with slopes ± 1.



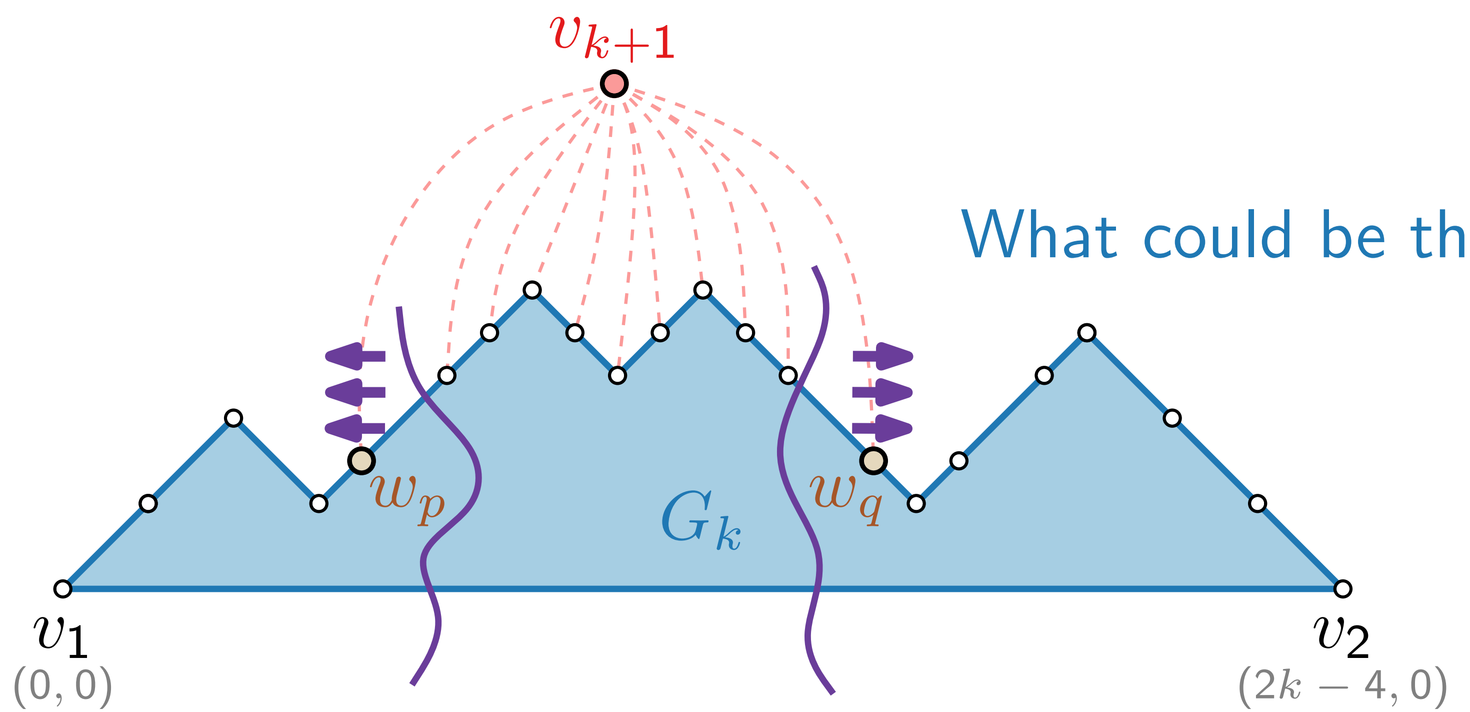
What could be the solution?

Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0,0)$, v_2 is at $(2k-4,0)$,
- boundary of G_k (minus edge $\{v_1,v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1,v_2\}$) is drawn with slopes ± 1.



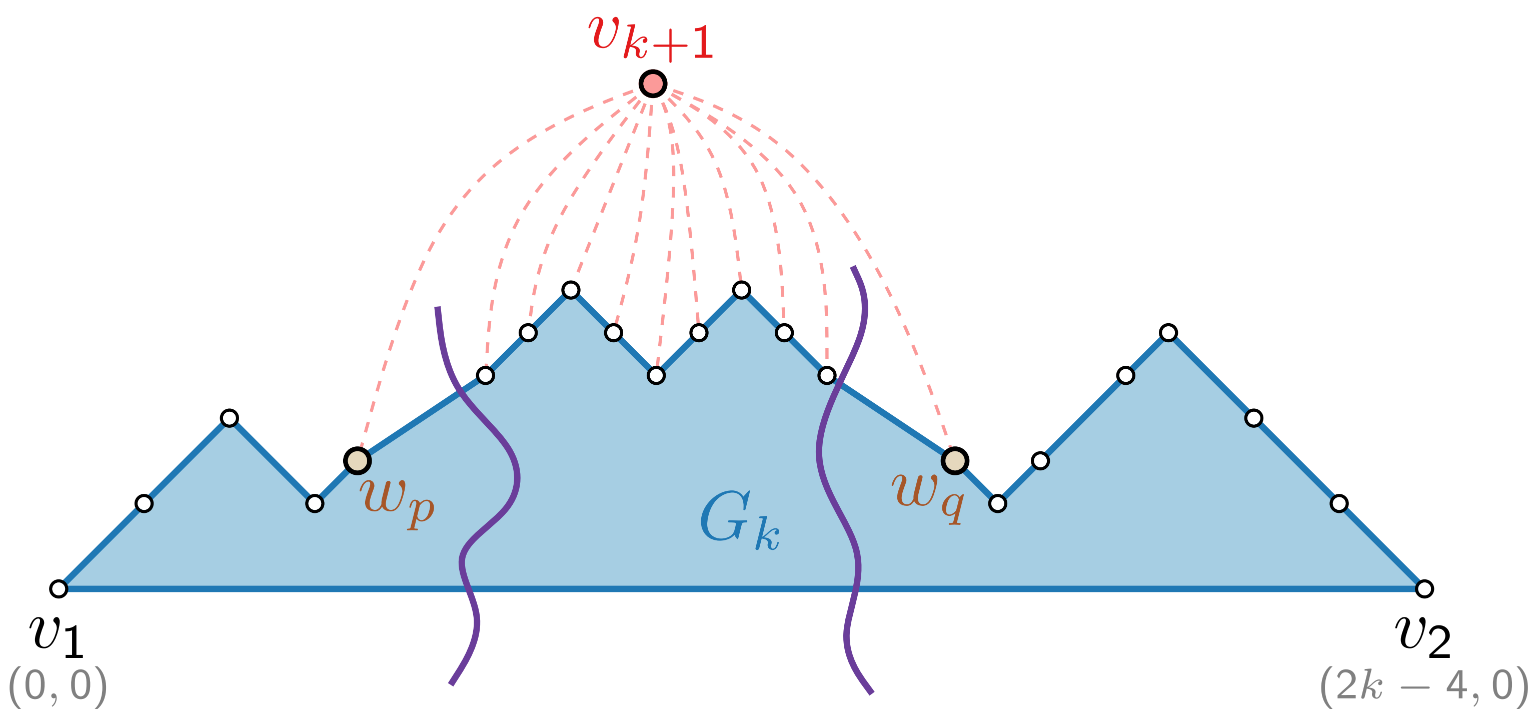
What could be the solution?

Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0,0)$, v_2 is at $(2k-4,0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

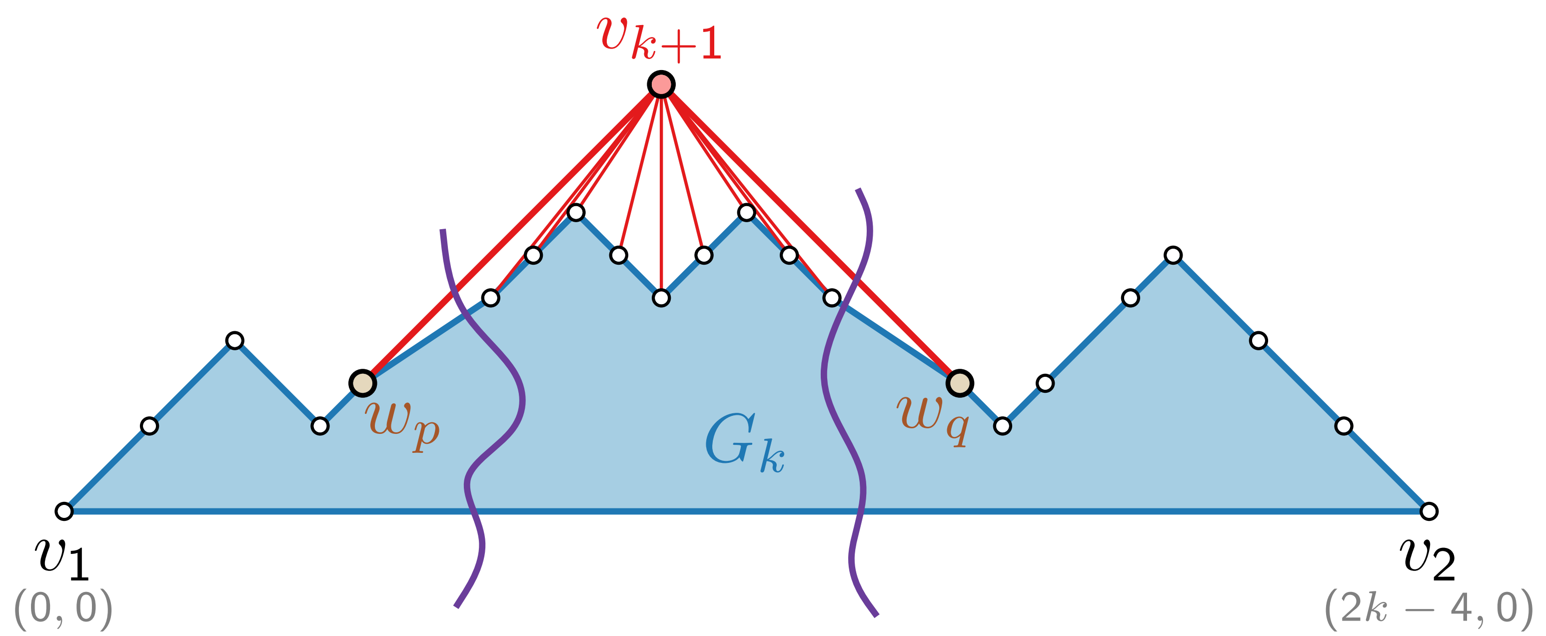


Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0,0)$, v_2 is at $(2k-4,0)$,
- boundary of G_k (minus edge $\{v_1,v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1,v_2\}$) is drawn with slopes ± 1.



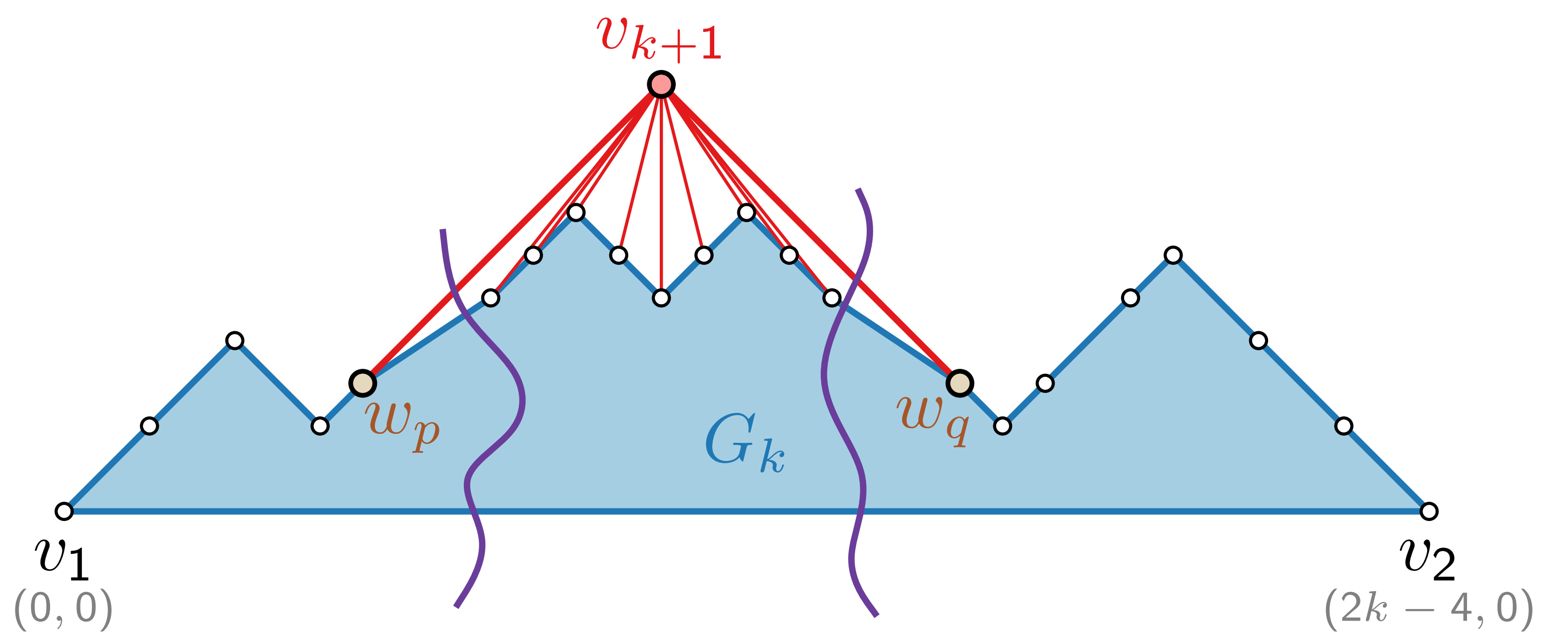
Shift Method – Idea

Drawing invariants:

G_k is drawn such that

- v_1 is at $(0, 0)$, v_2 is at $(2k - 4, 0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

Will v_{k+1} lie on the grid?



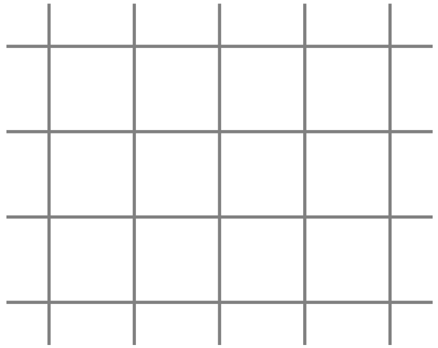
Shift Method – Idea

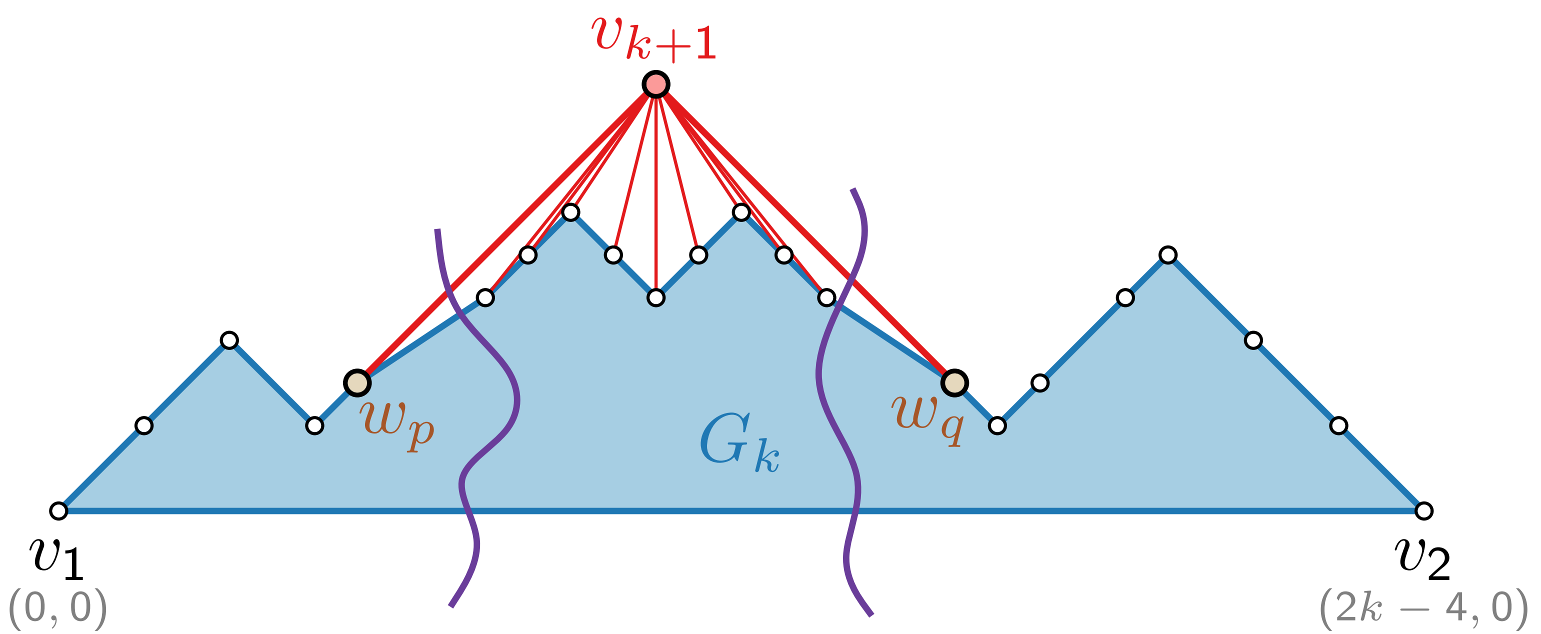
Drawing invariants:

G_k is drawn such that

- v_1 is at $(0, 0)$, v_2 is at $(2k - 4, 0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

Will v_{k+1} lie on the grid?





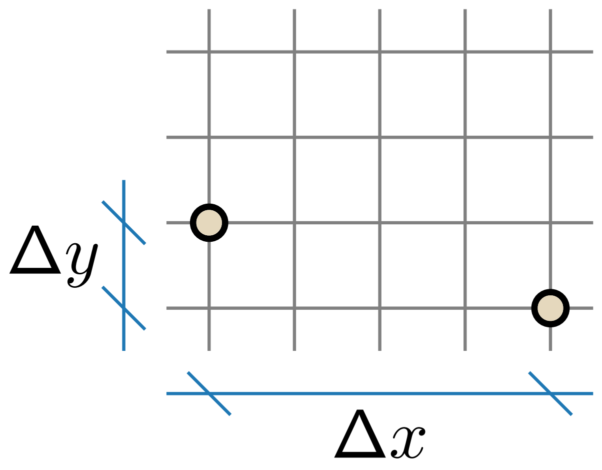
Shift Method – Idea

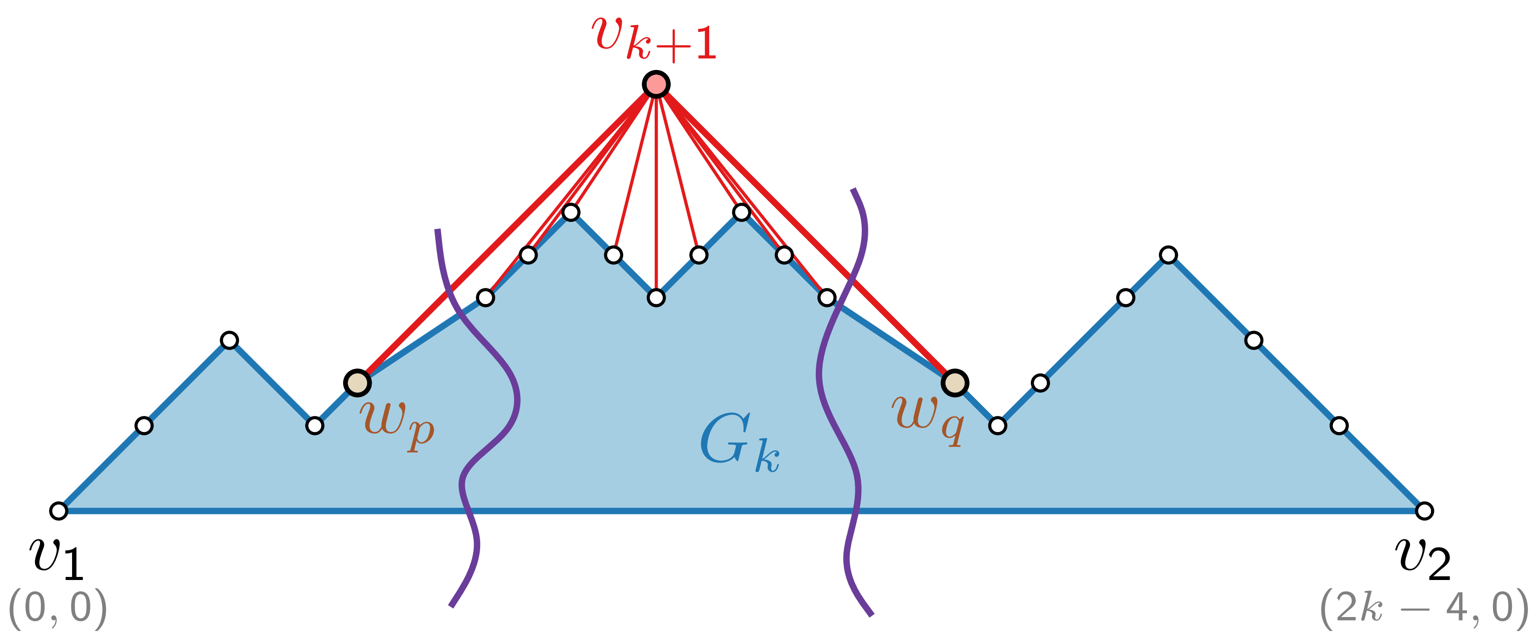
Drawing invariants:

G_k is drawn such that

- v_1 is at $(0, 0)$, v_2 is at $(2k - 4, 0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

Will v_{k+1} lie on the grid?





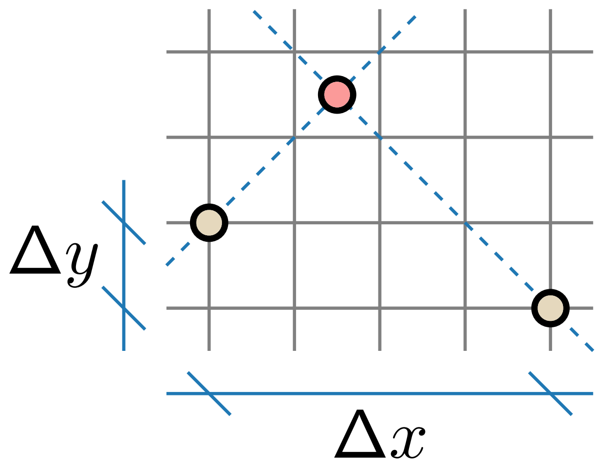
Shift Method – Idea

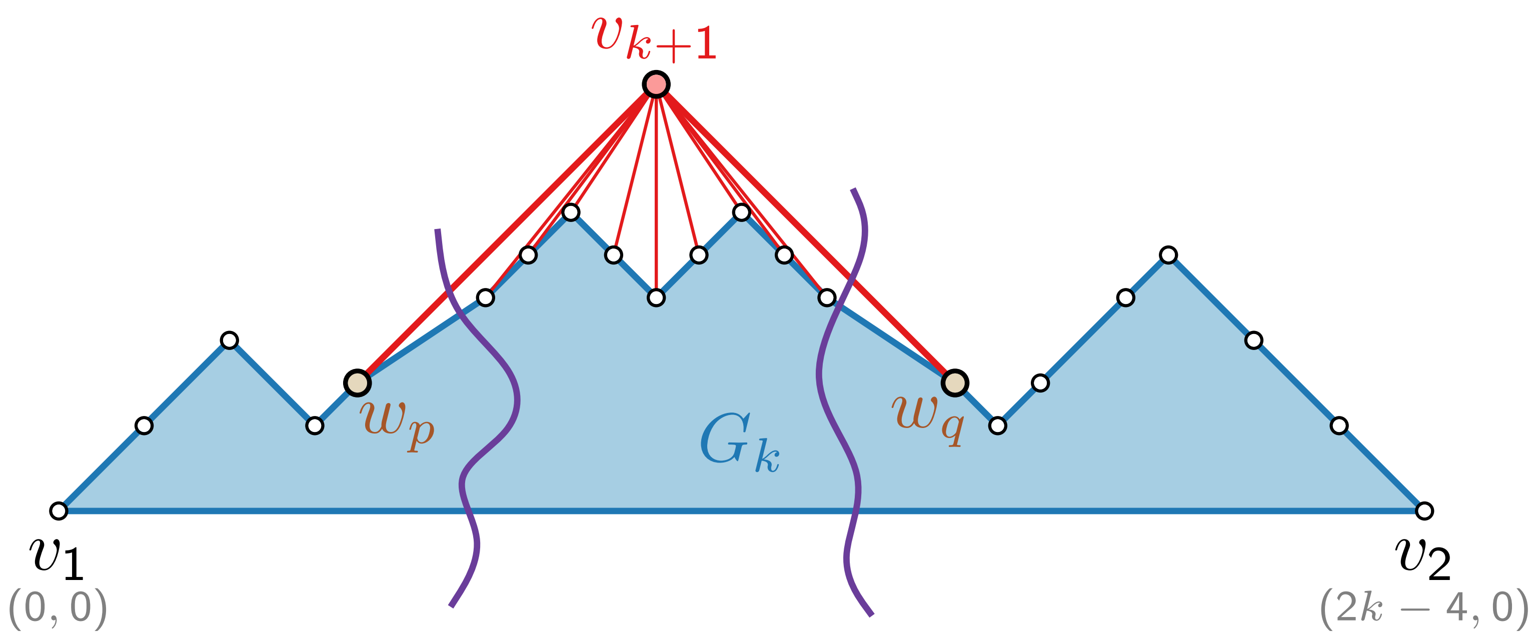
Drawing invariants:

G_k is drawn such that

- v_1 is at $(0, 0)$, v_2 is at $(2k - 4, 0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

Will v_{k+1} lie on the grid?





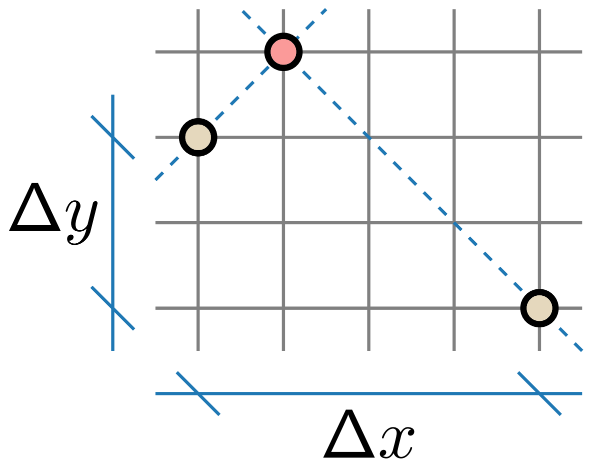
Shift Method – Idea

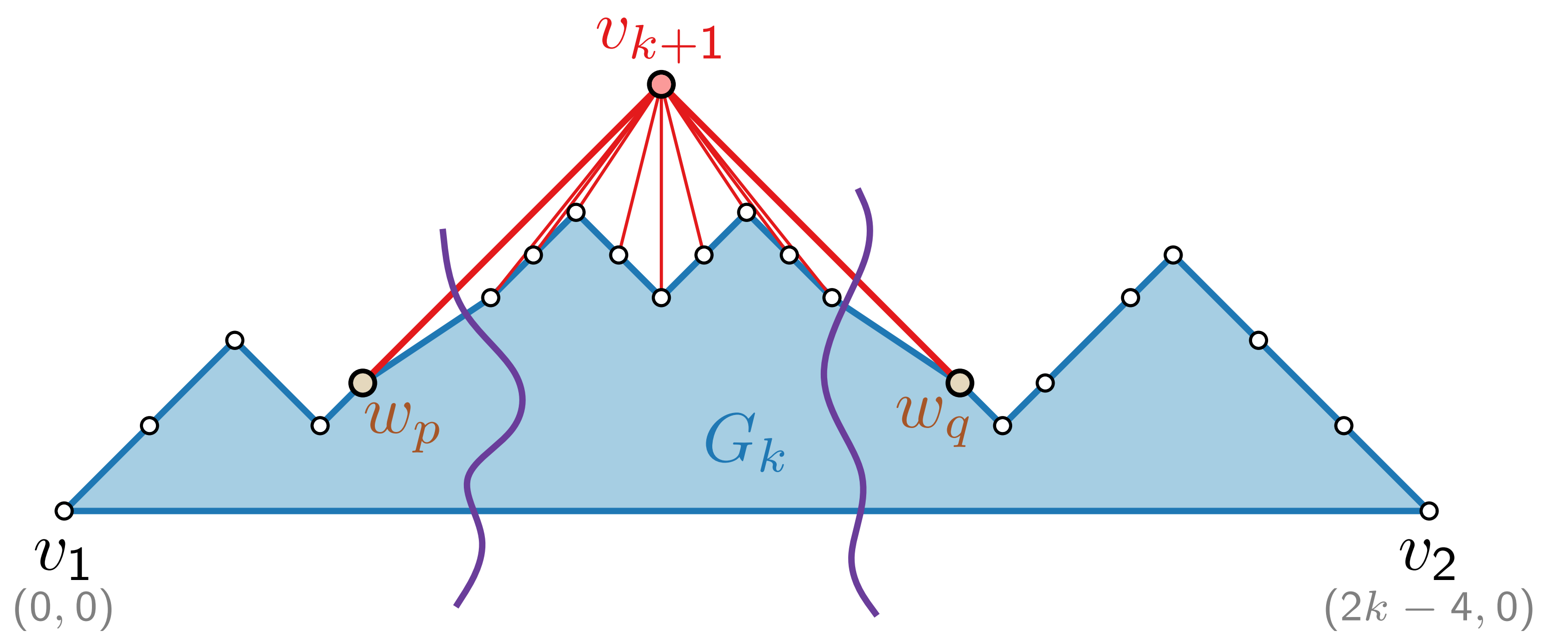
Drawing invariants:

G_k is drawn such that

- v_1 is at $(0, 0)$, v_2 is at $(2k - 4, 0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

Will v_{k+1} lie on the grid?





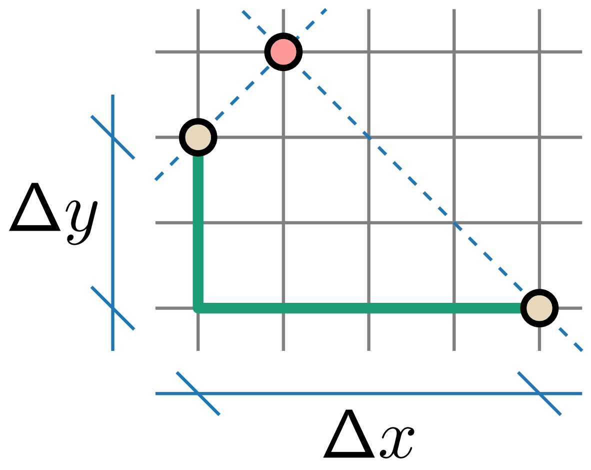
Shift Method – Idea

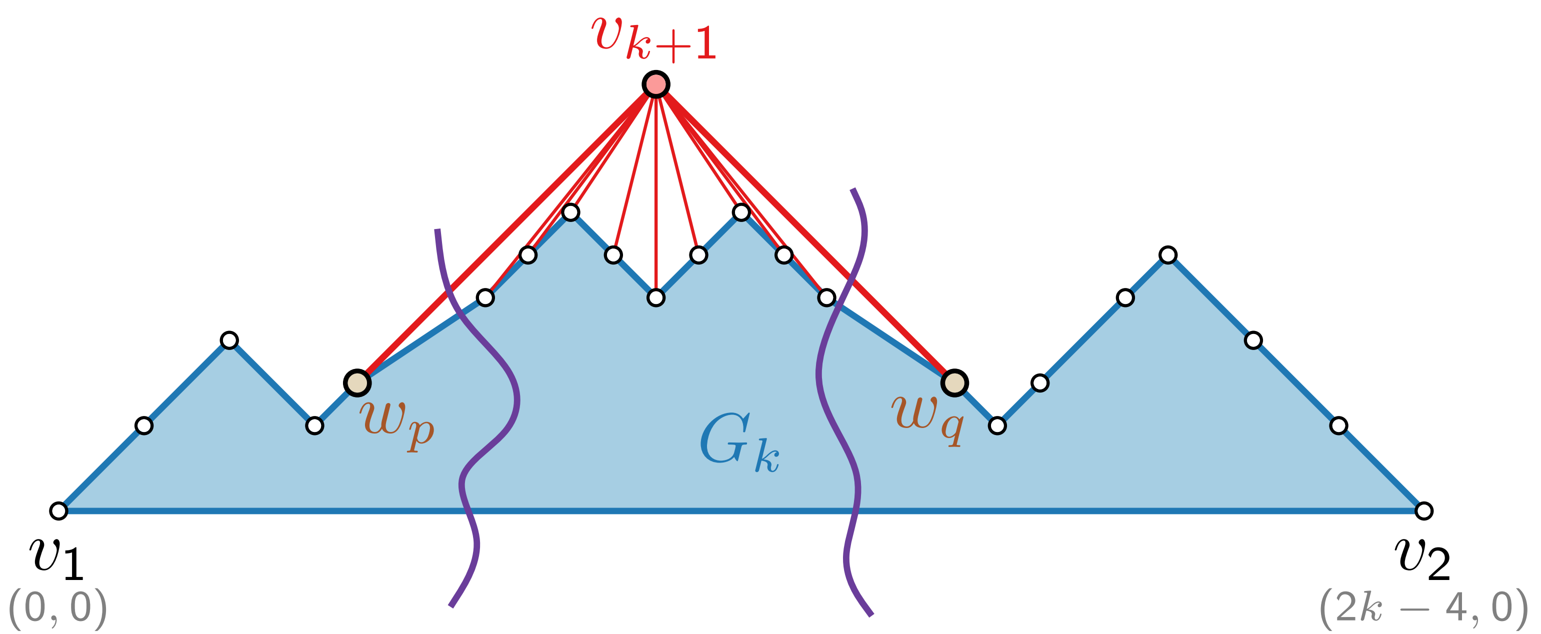
Drawing invariants:

G_k is drawn such that

- v_1 is at $(0, 0)$, v_2 is at $(2k-4, 0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.

Will v_{k+1} lie on the grid?



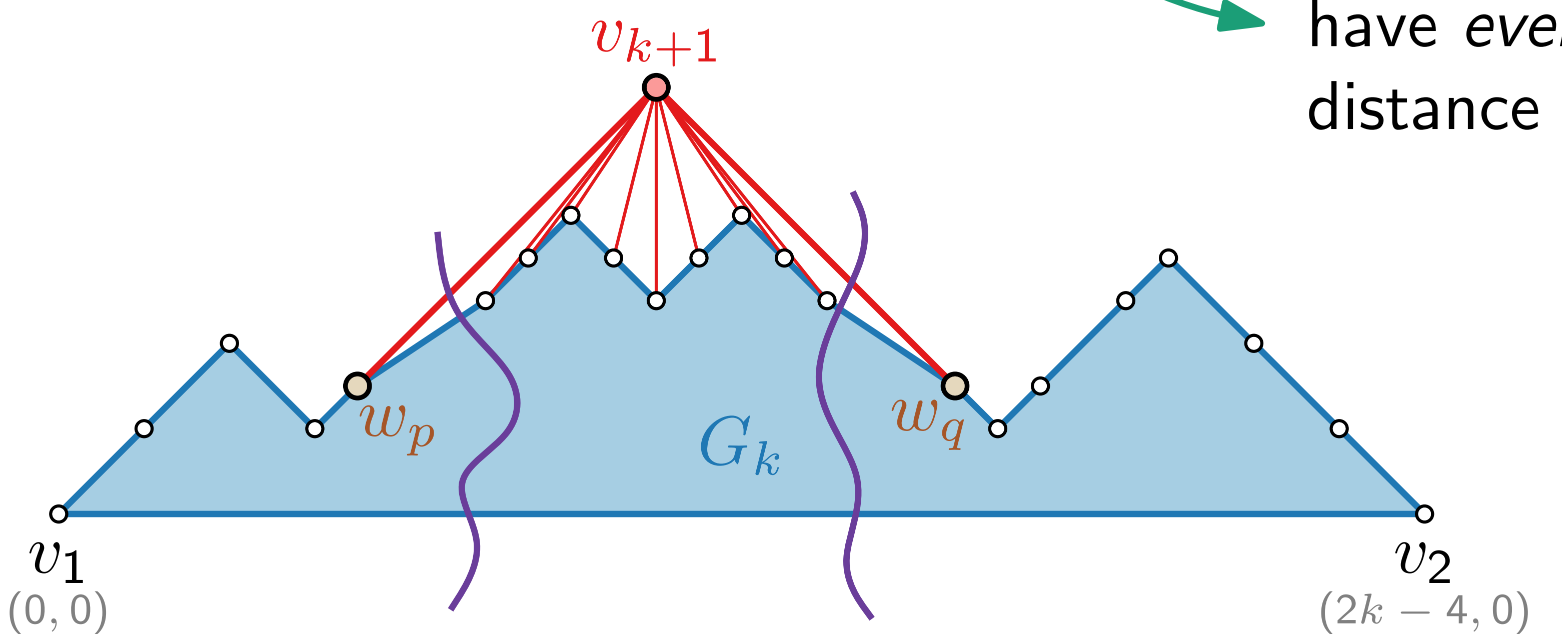


Shift Method – Idea

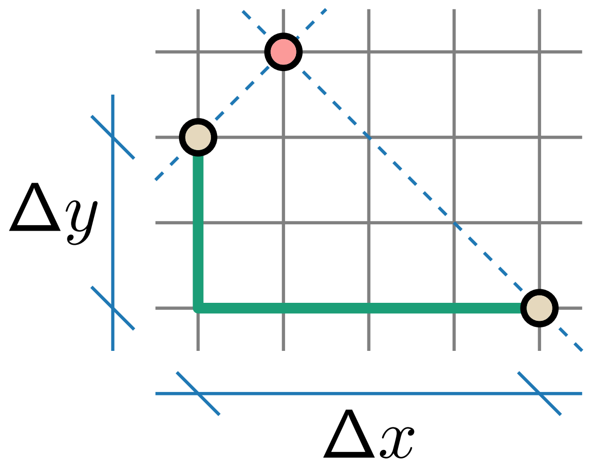
Drawing invariants:

G_k is drawn such that

- v_1 is at $(0, 0)$, v_2 is at $(2k - 4, 0)$,
- boundary of G_k (minus edge $\{v_1, v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1, v_2\}$) is drawn with slopes ± 1.



Will v_{k+1} lie on the grid?



Yes, because w_p and w_q have *even* Manhattan distance $\Delta x + \Delta y$.

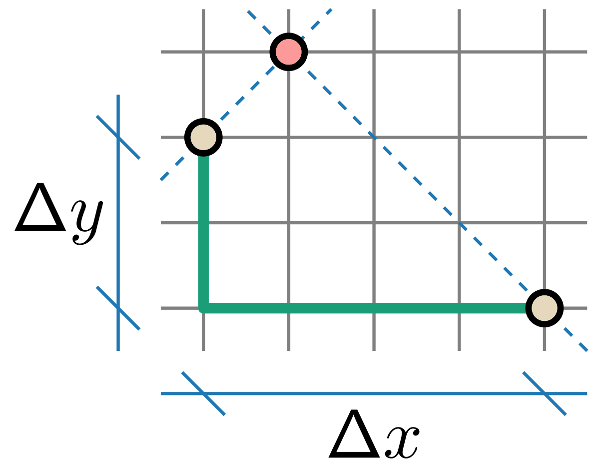
Shift Method – Idea

Drawing invariants:

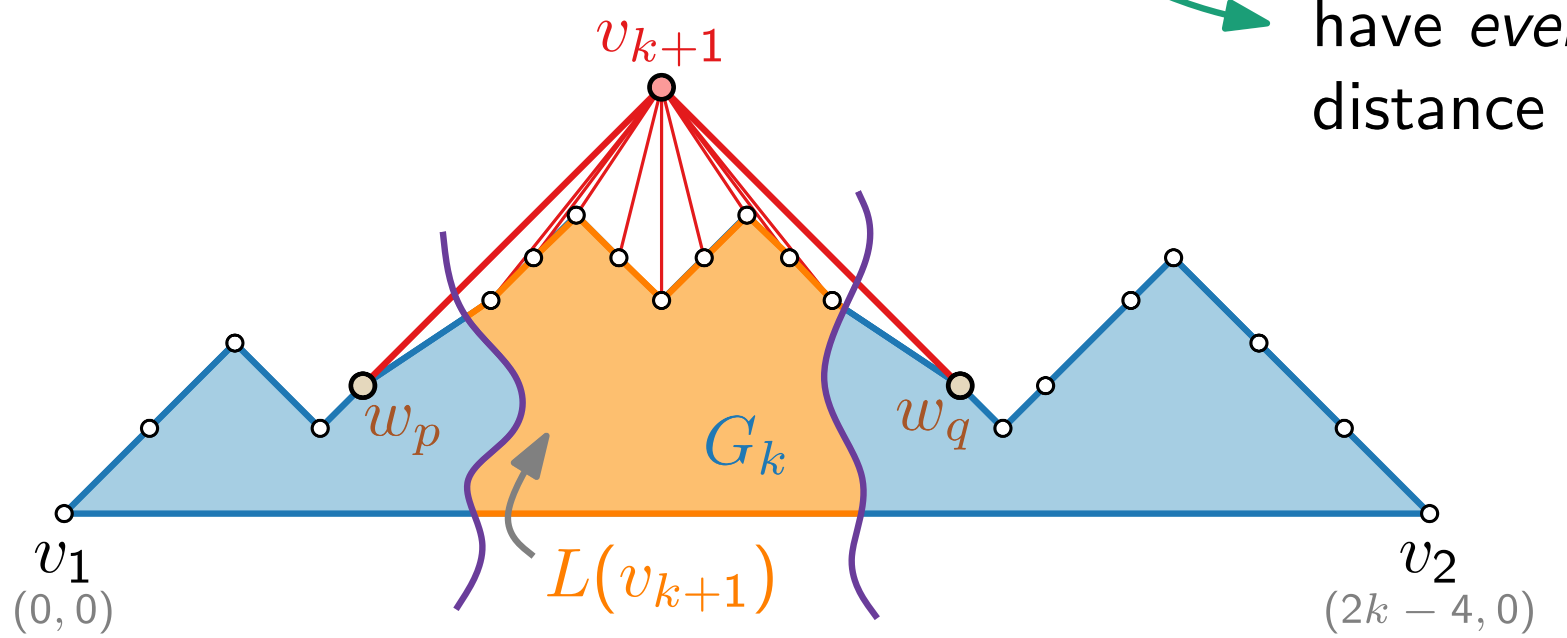
G_k is drawn such that

- v_1 is at $(0,0)$, v_2 is at $(2k-4,0)$,
- boundary of G_k (minus edge $\{v_1,v_2\}$) is drawn x-monotone,
- each edge on the boundary of G_k (except $\{v_1,v_2\}$) is drawn with slopes ± 1.

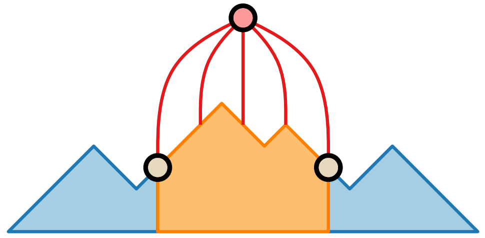
Will v_{k+1} lie on the grid?



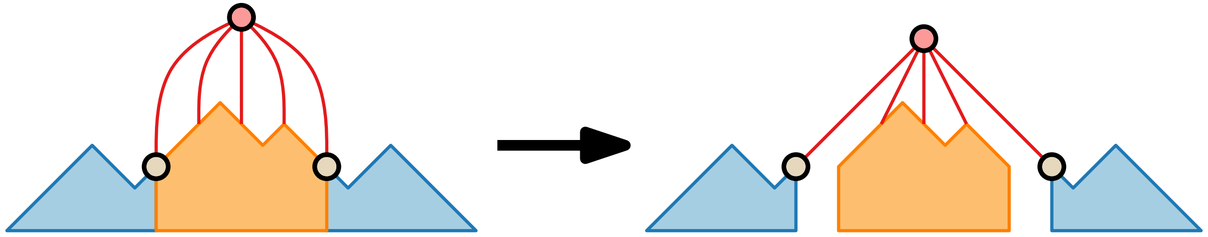
Yes, because w_p and w_q have *even* Manhattan distance $\Delta x + \Delta y$.



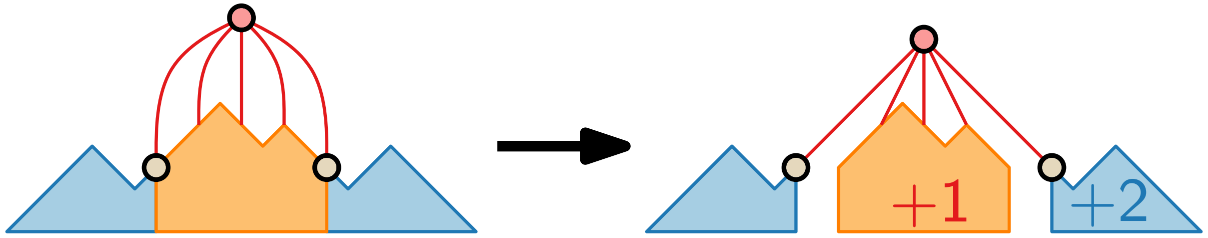
Shift Method – Example



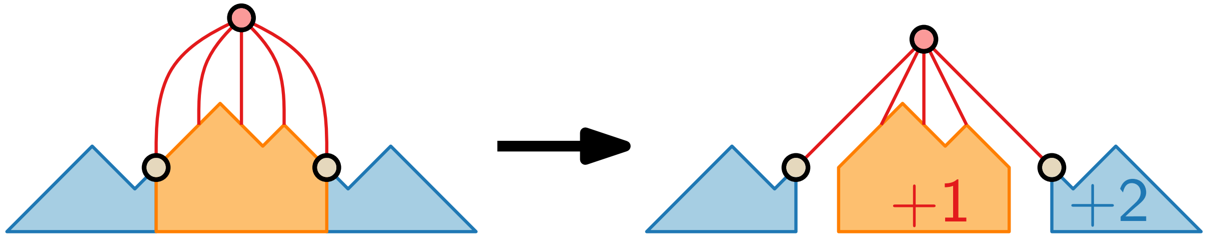
Shift Method – Example

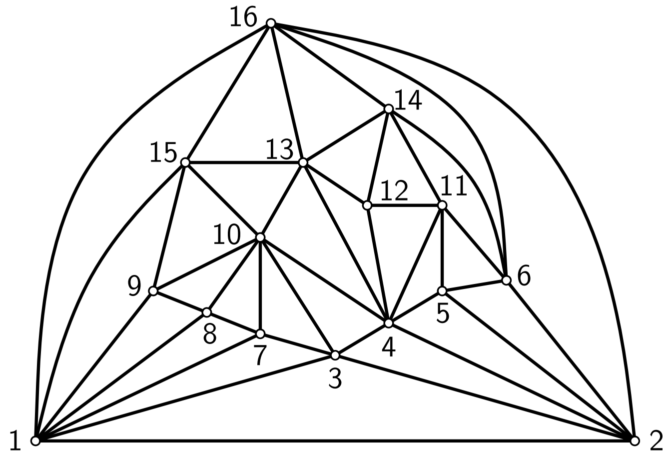


Shift Method – Example

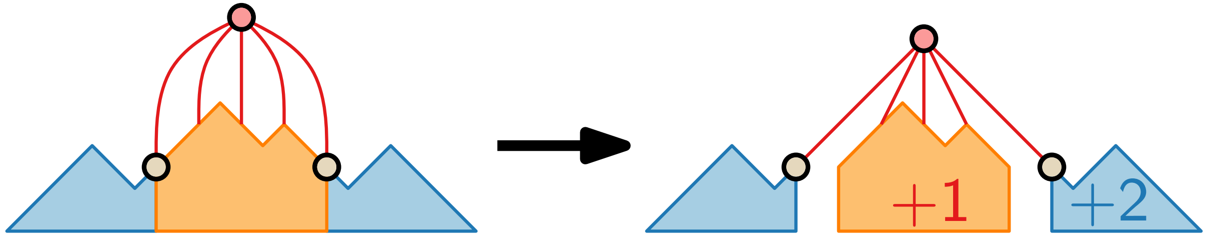


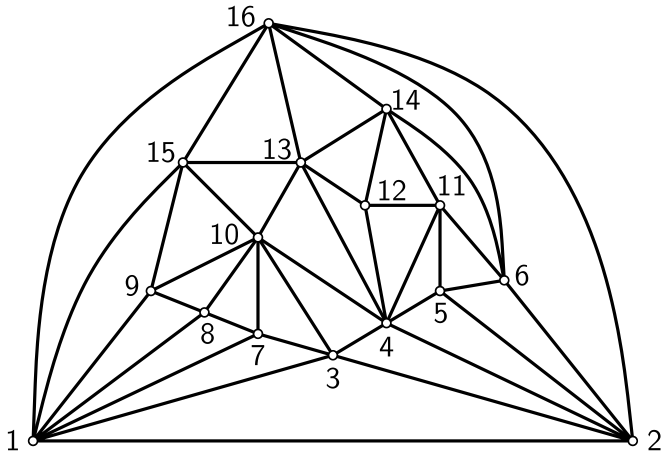
Shift Method – Example

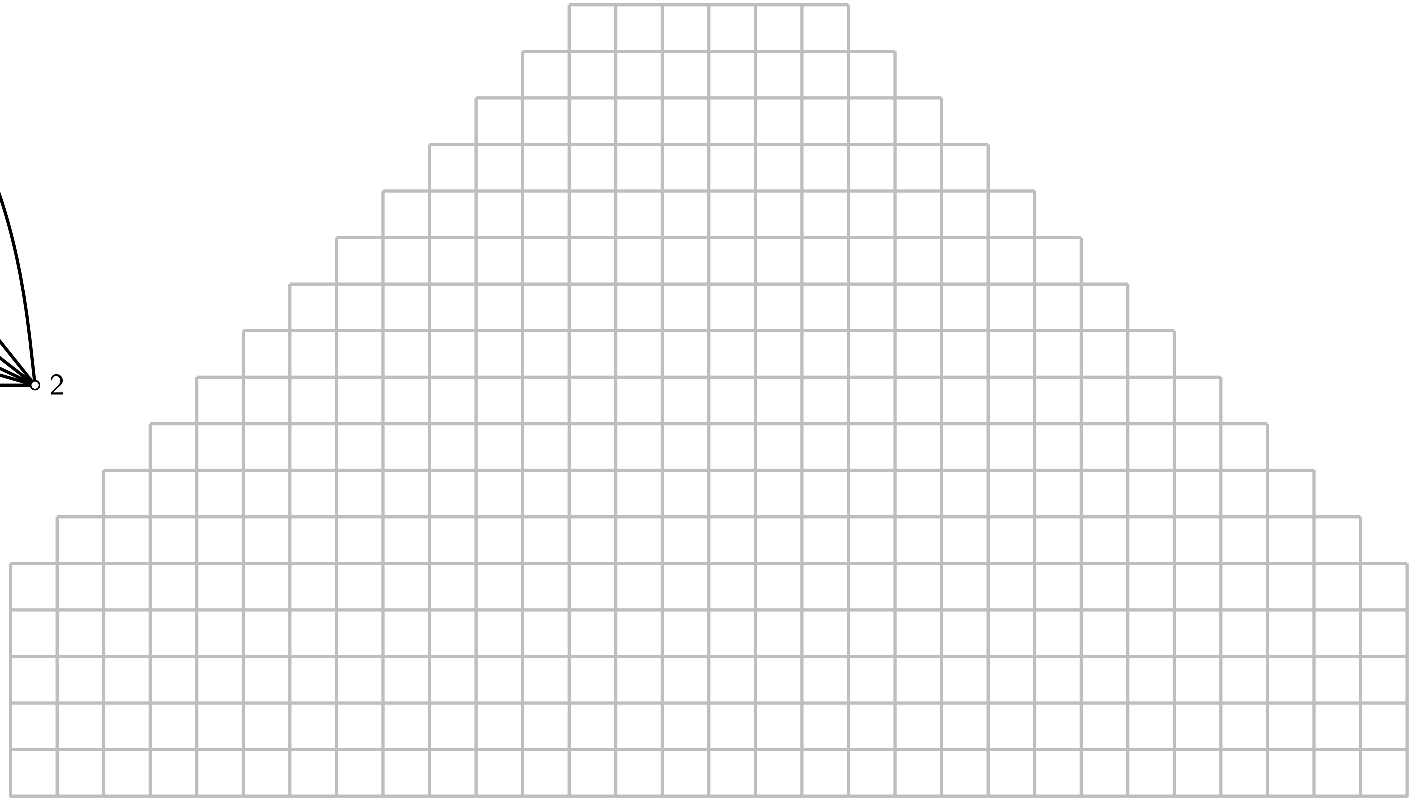




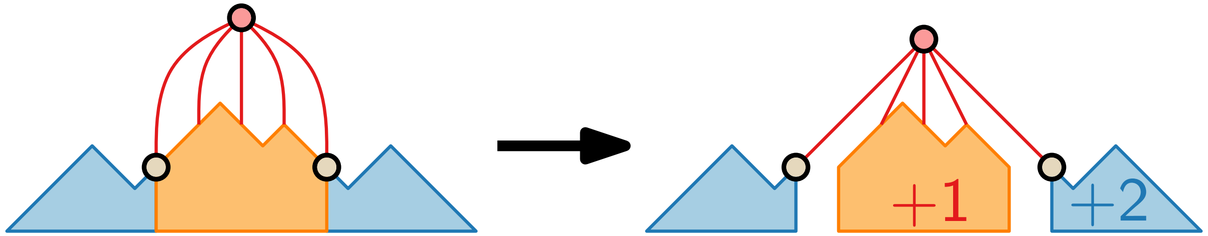
Shift Method – Example

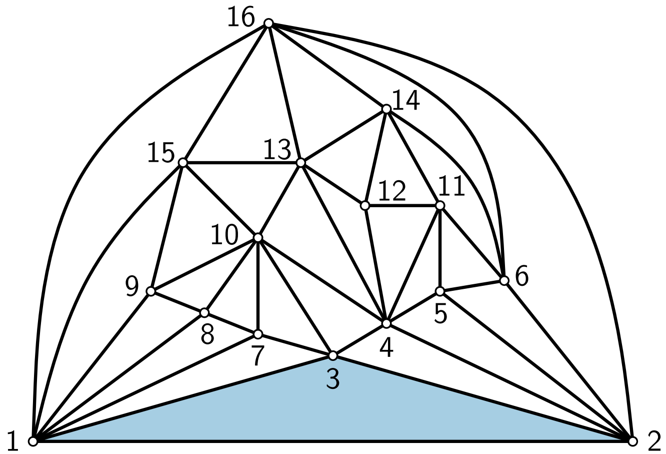


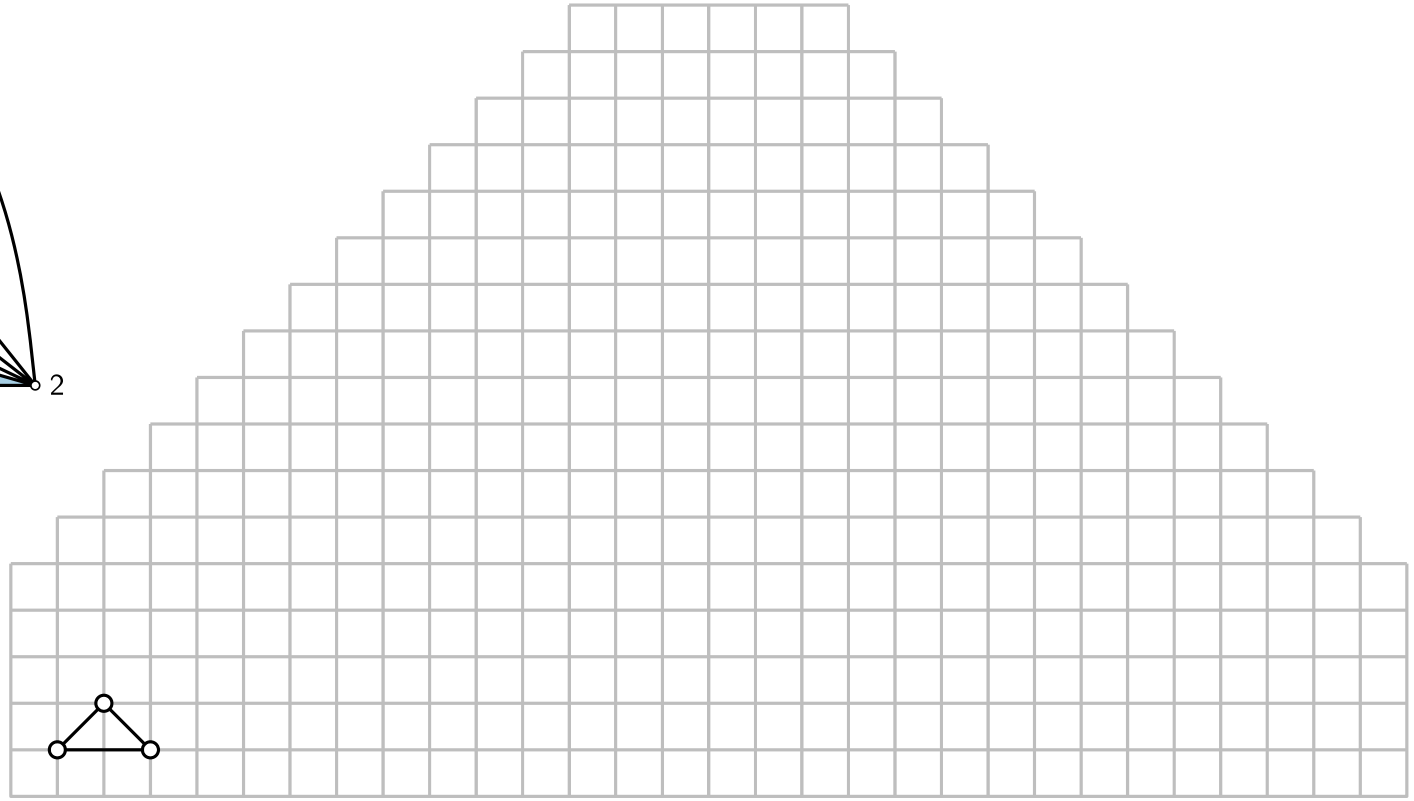




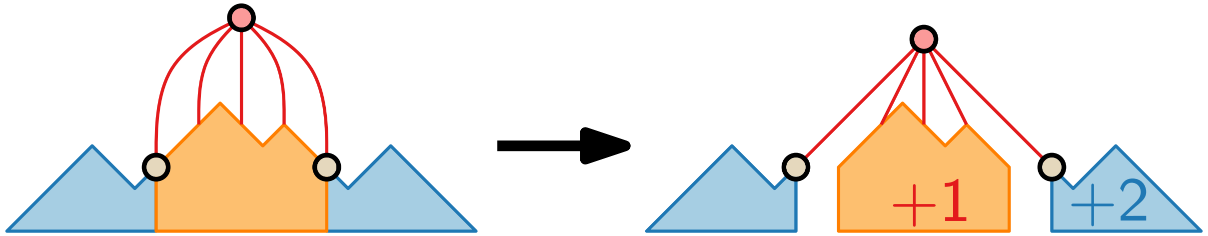
Shift Method – Example

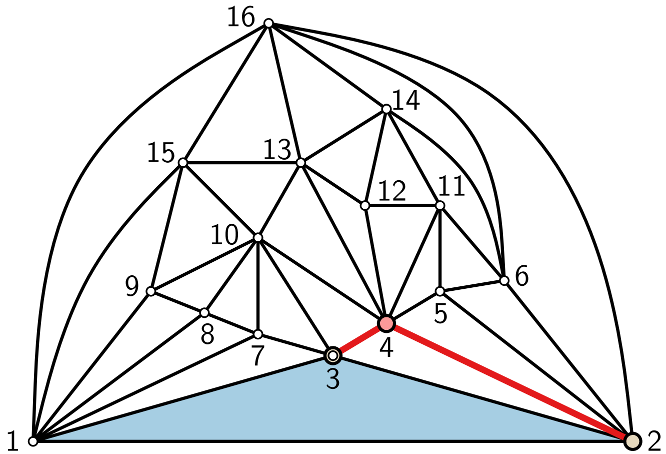


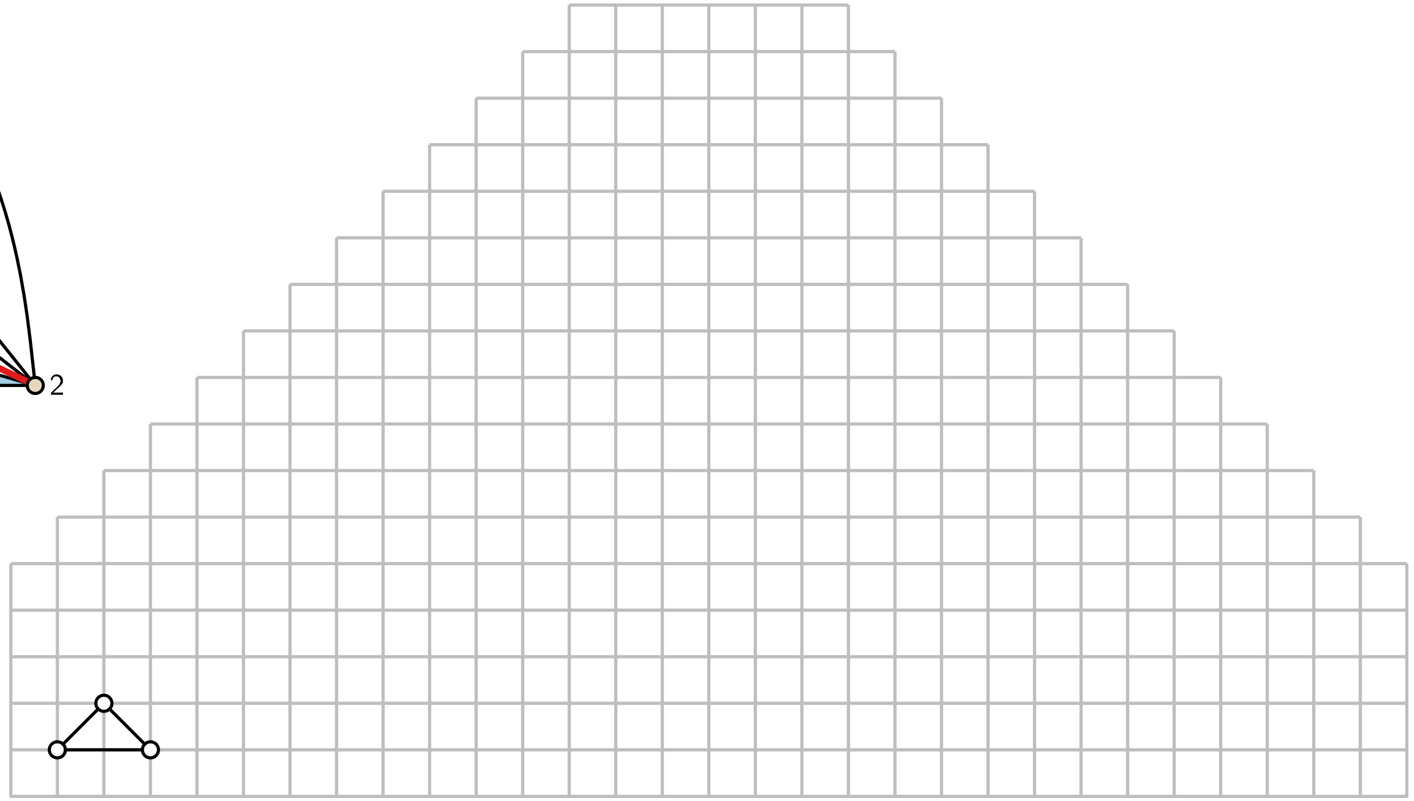




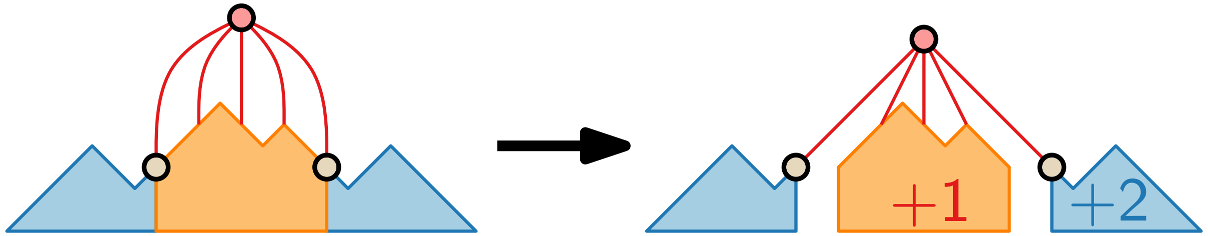
Shift Method – Example

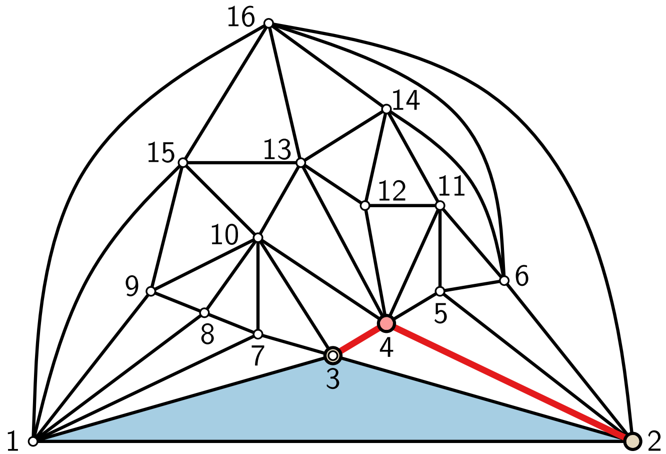


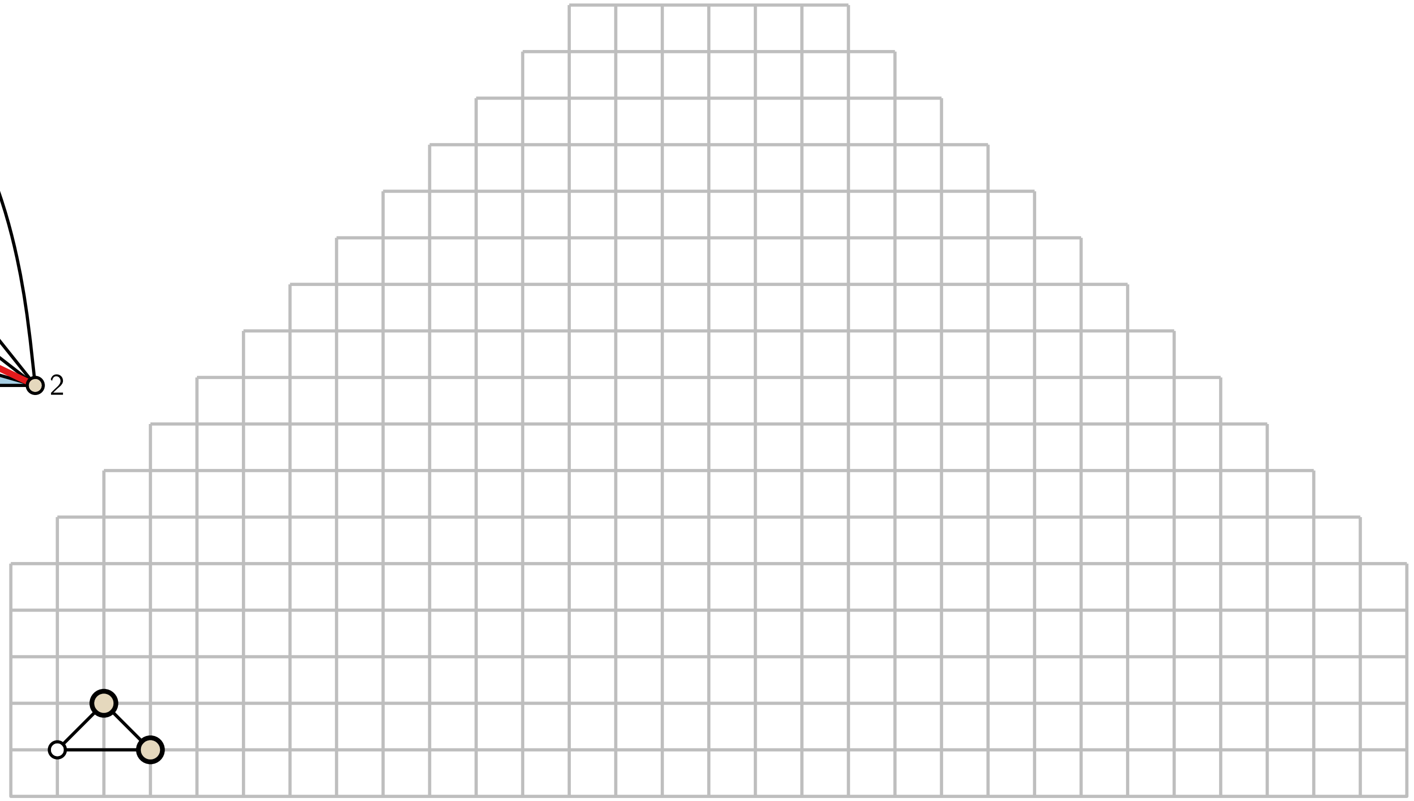




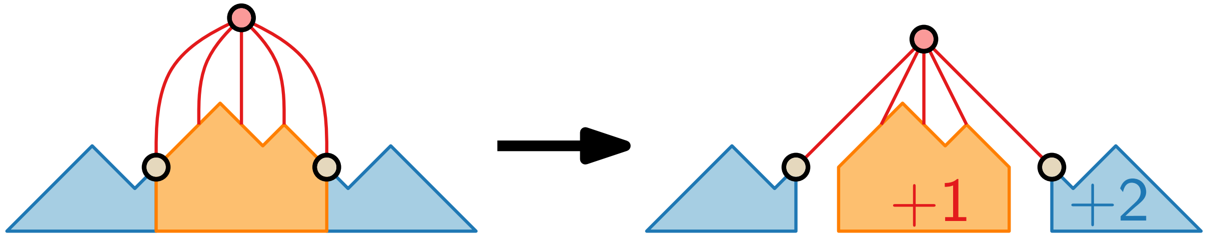
Shift Method – Example

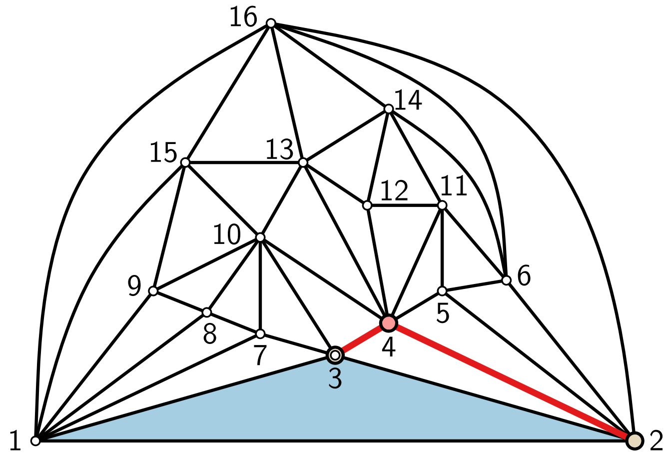


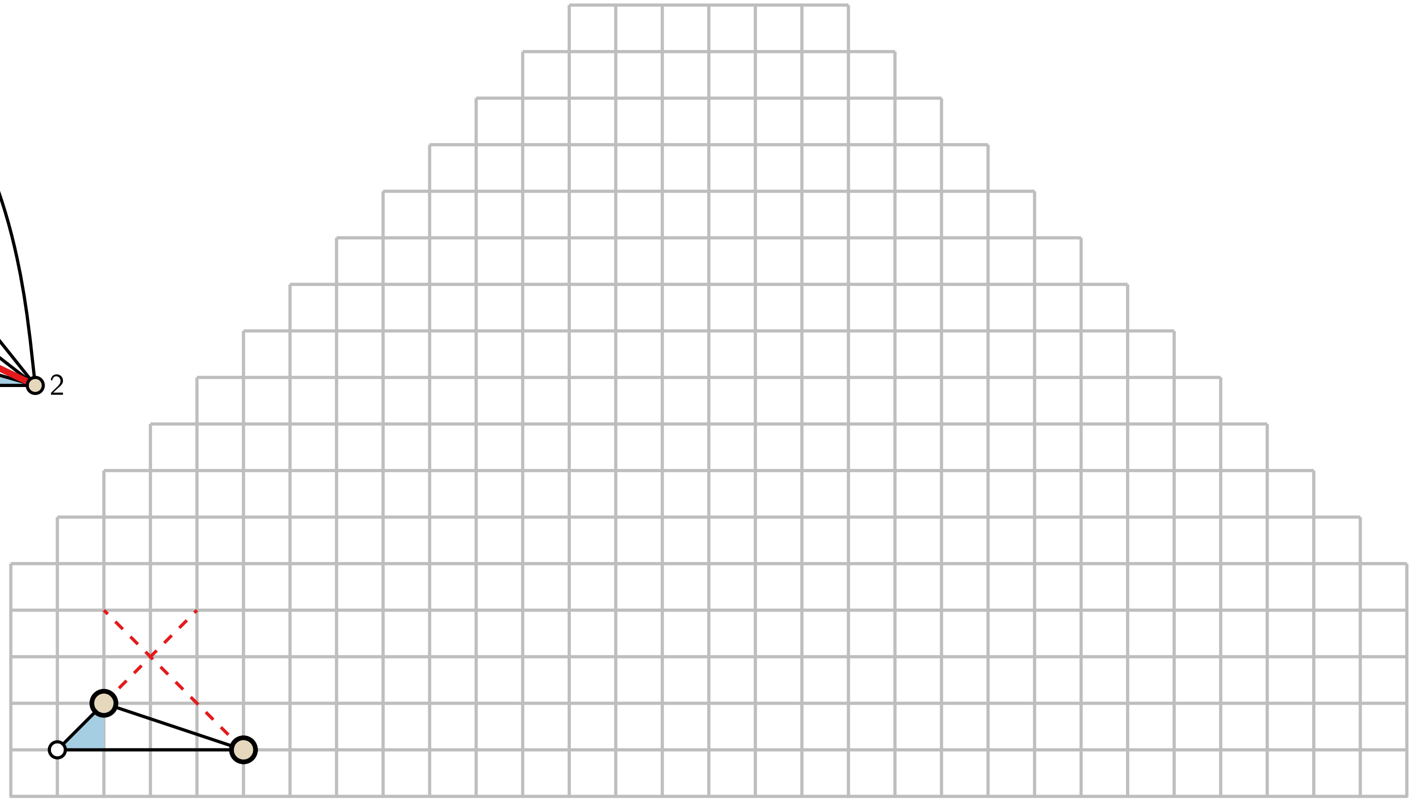




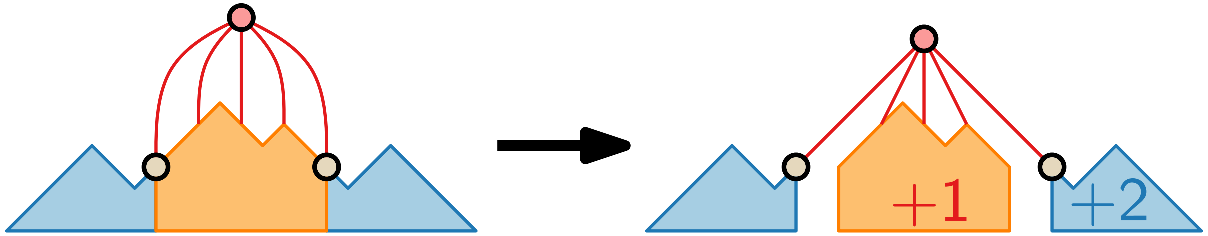
Shift Method – Example

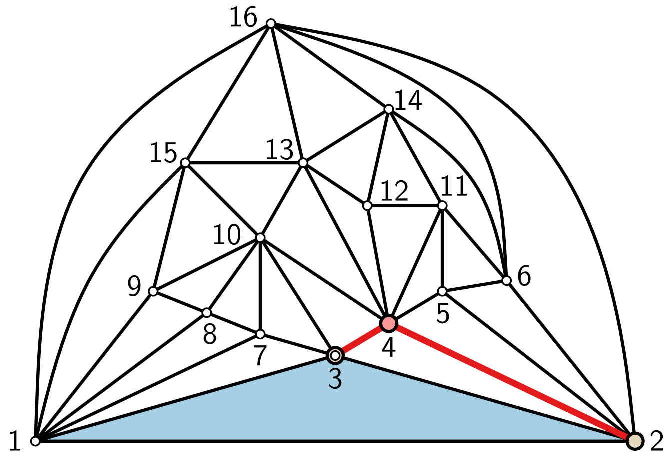


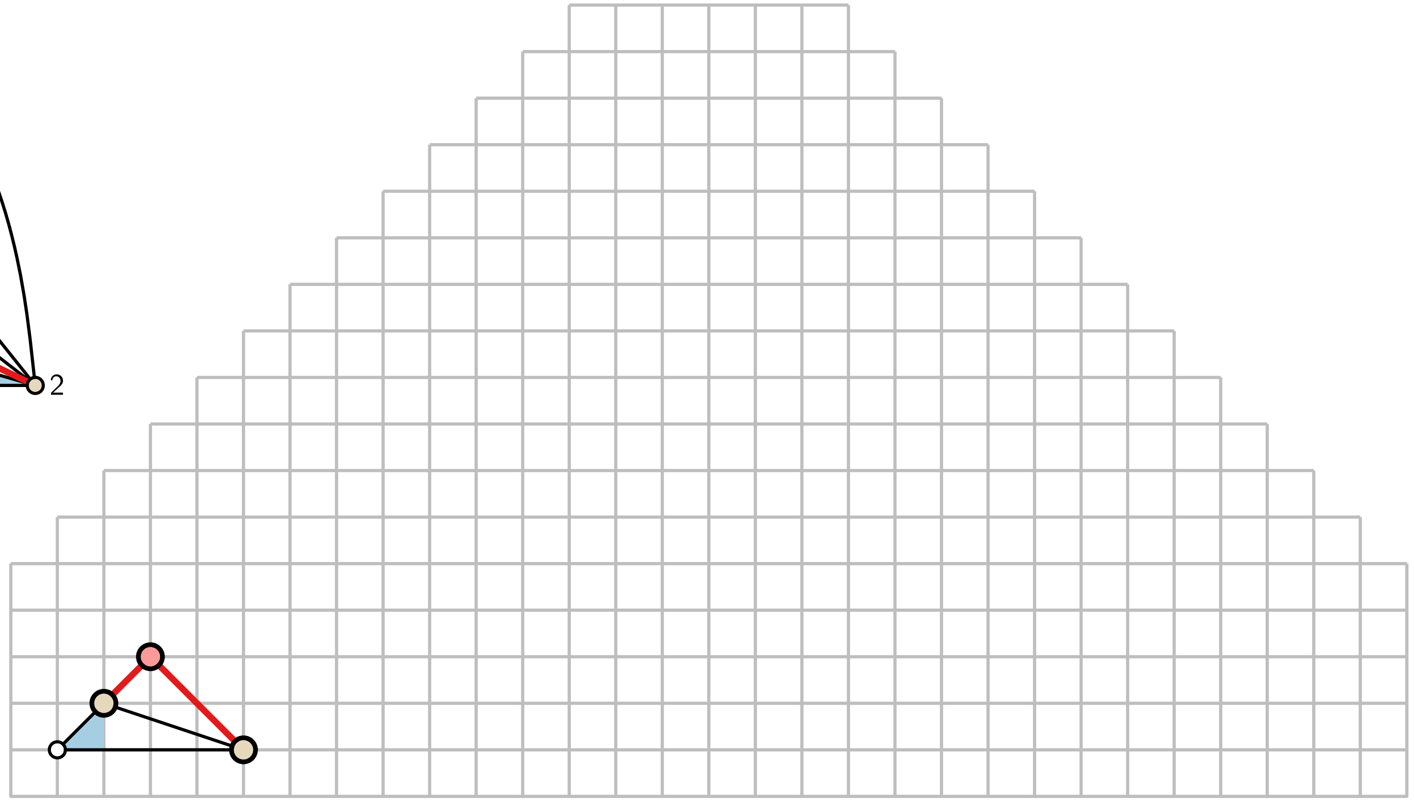




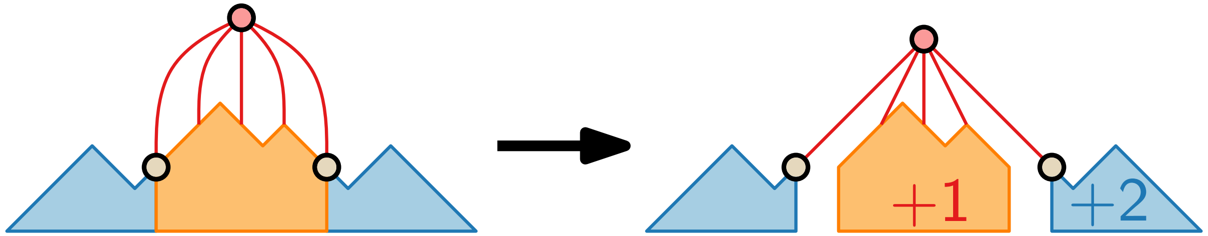
Shift Method – Example

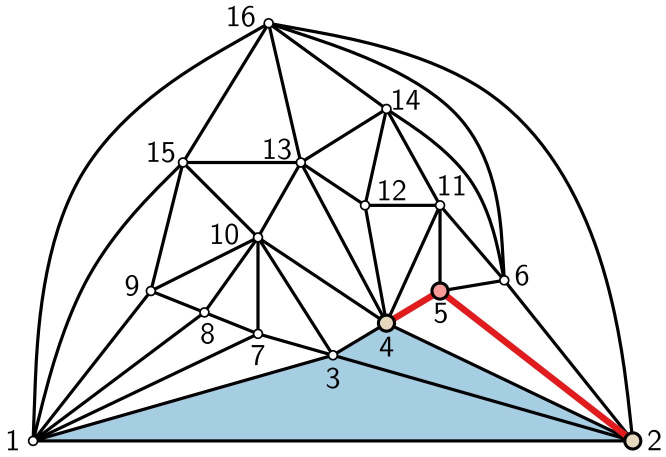


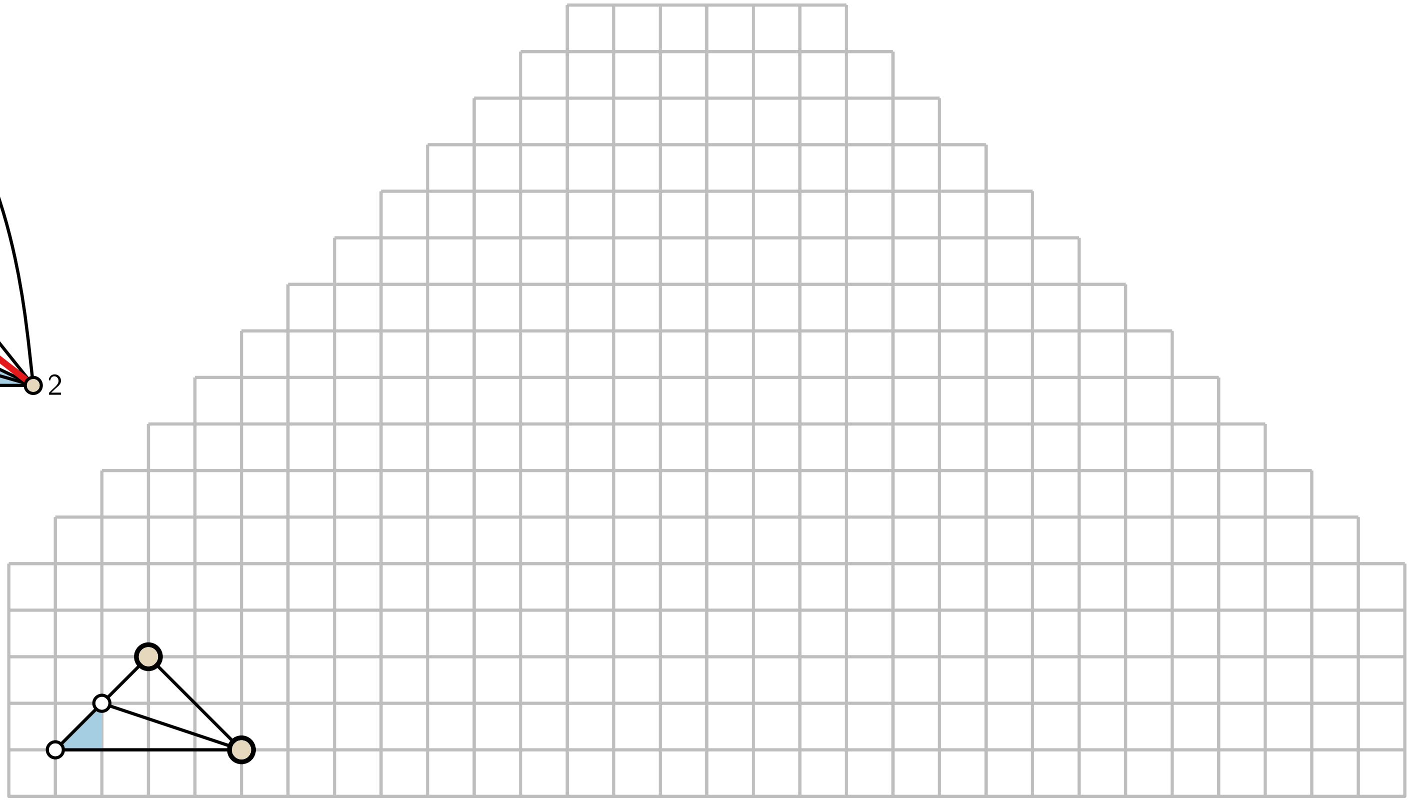




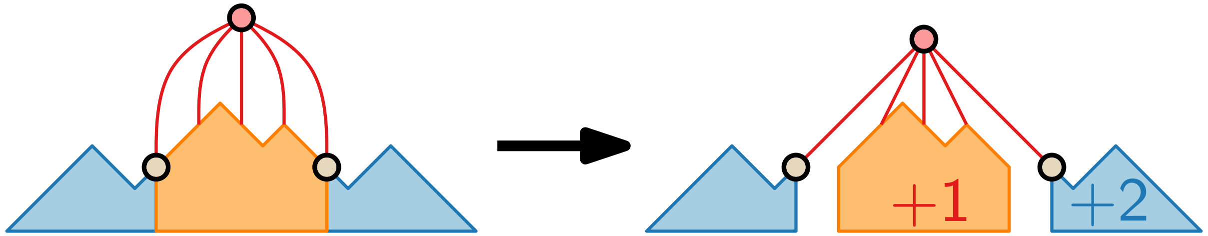
Shift Method – Example

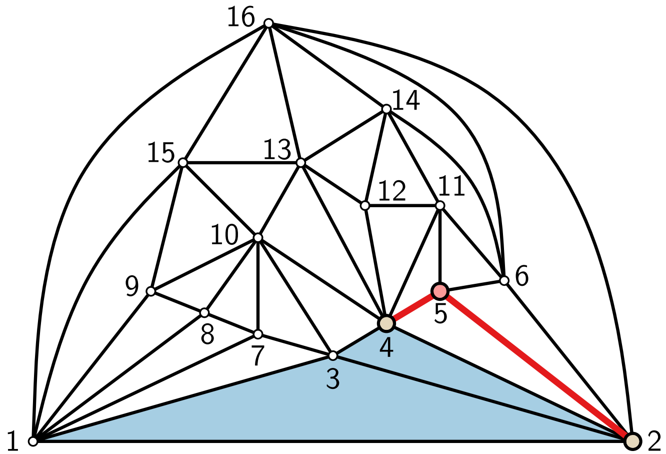


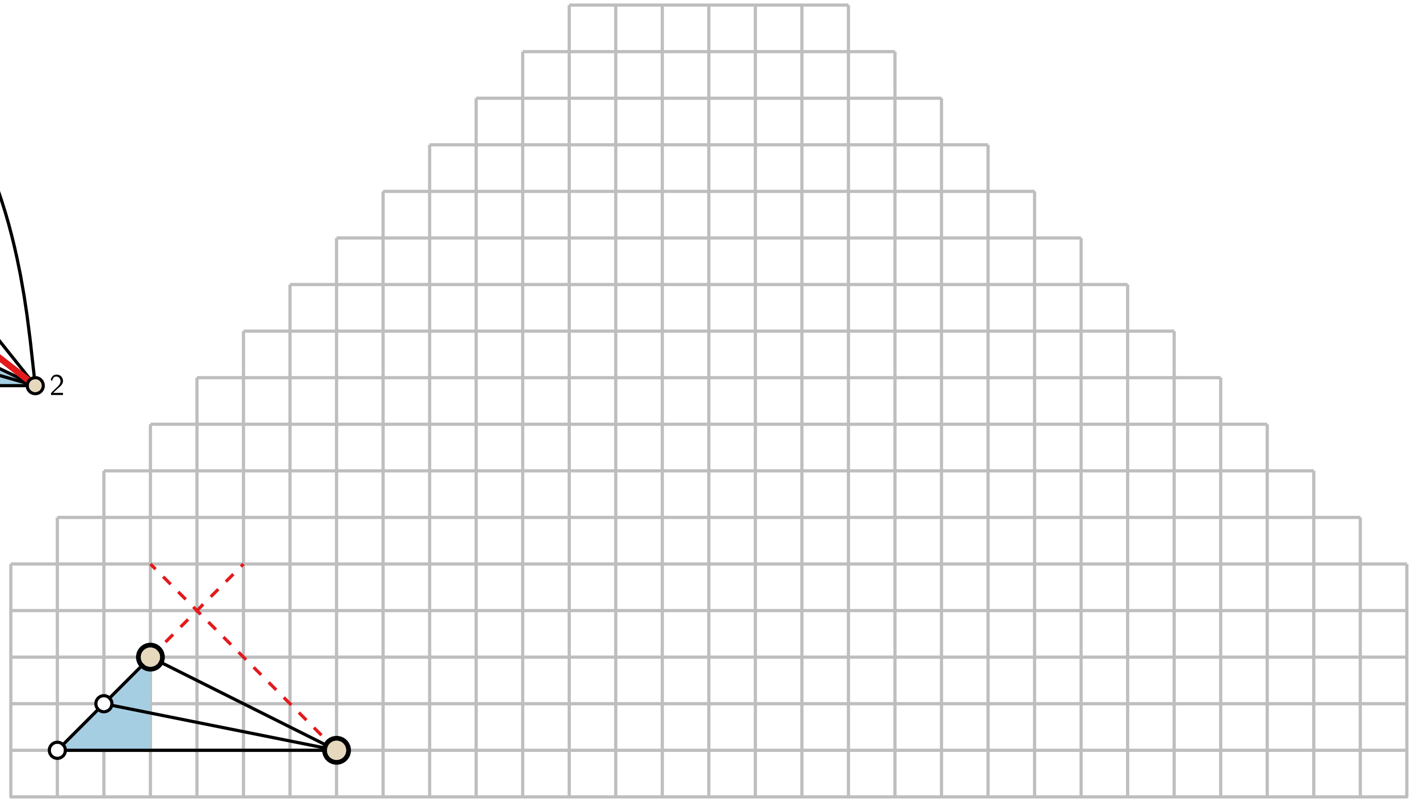




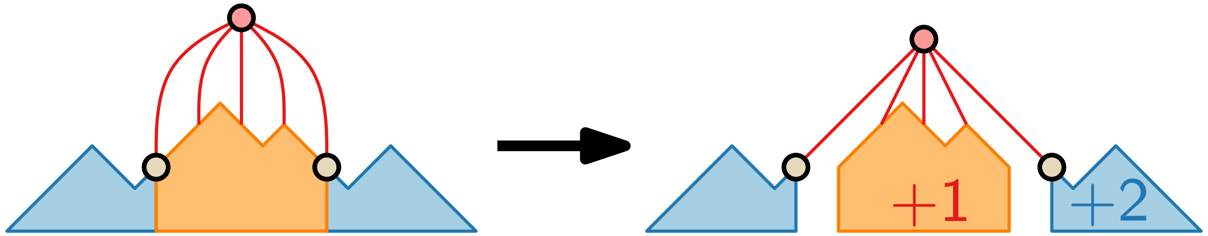
Shift Method – Example

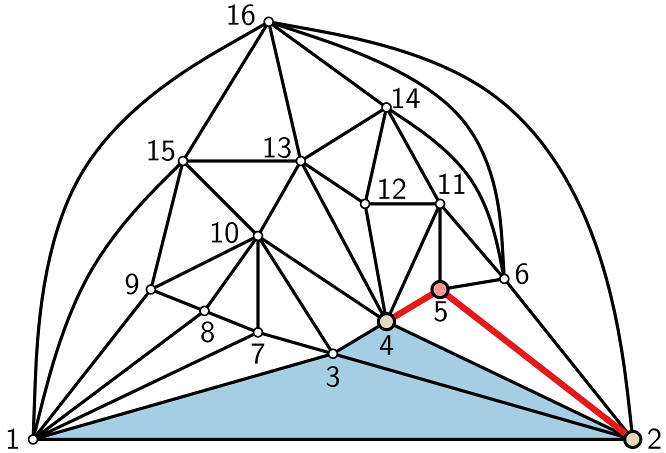


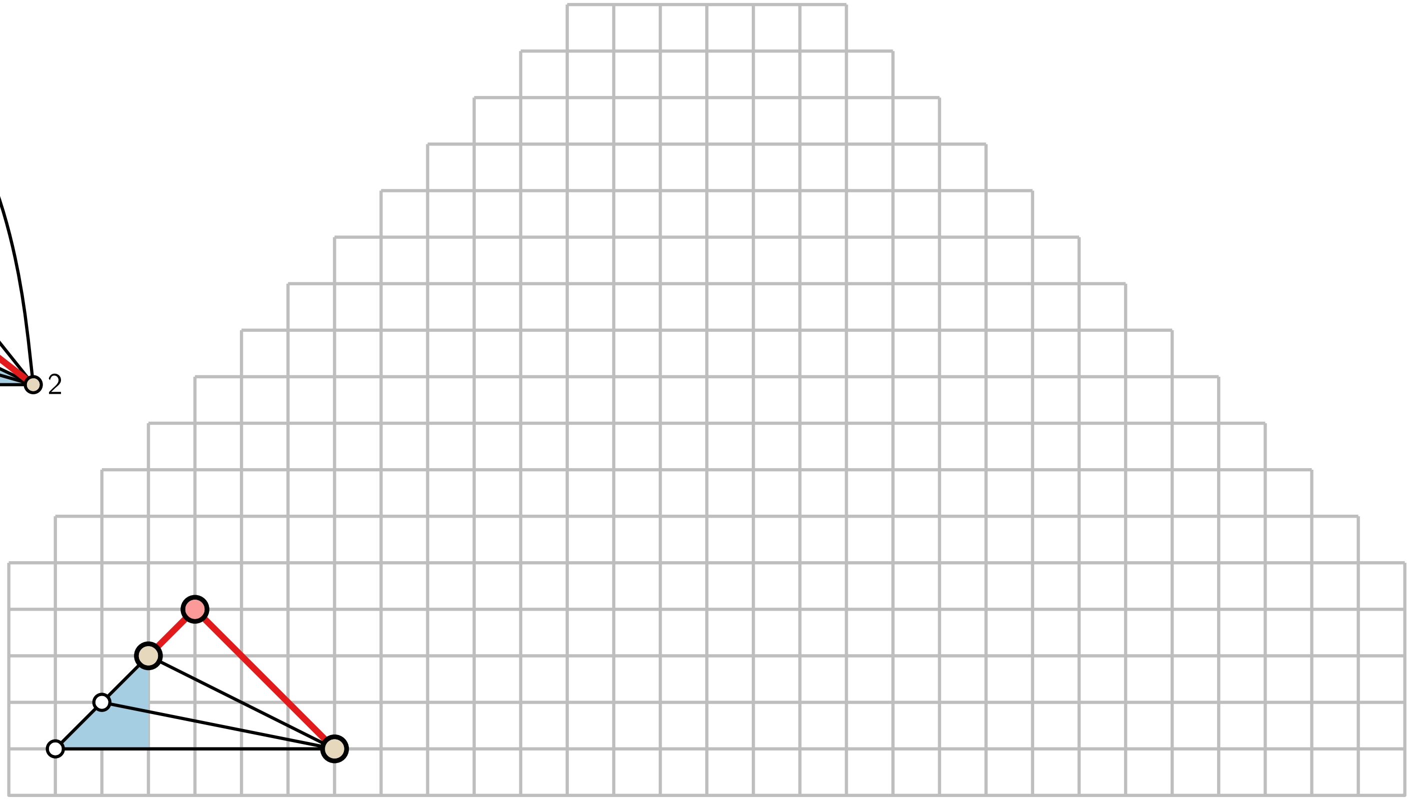




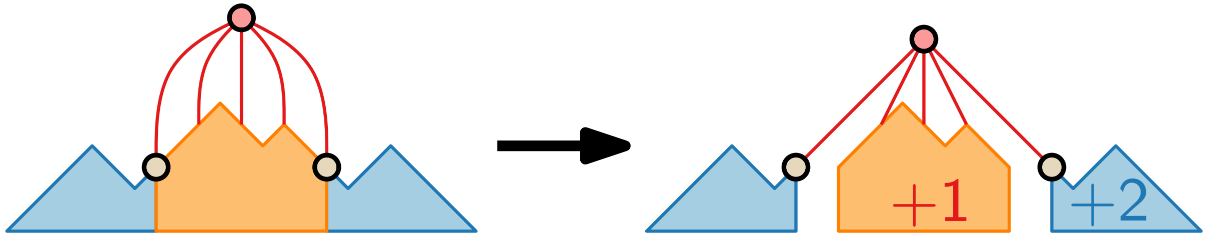
Shift Method – Example

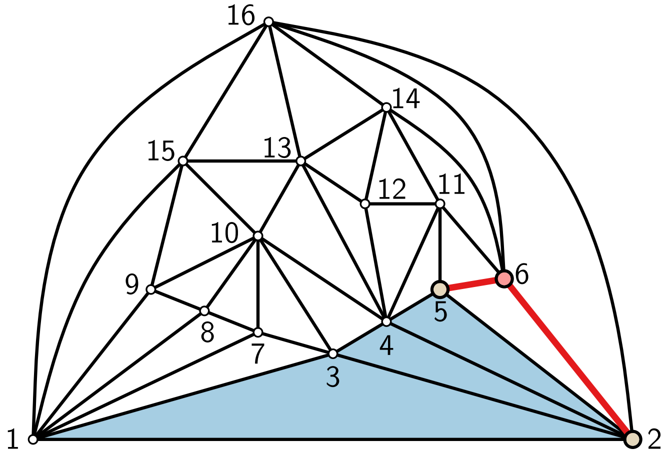


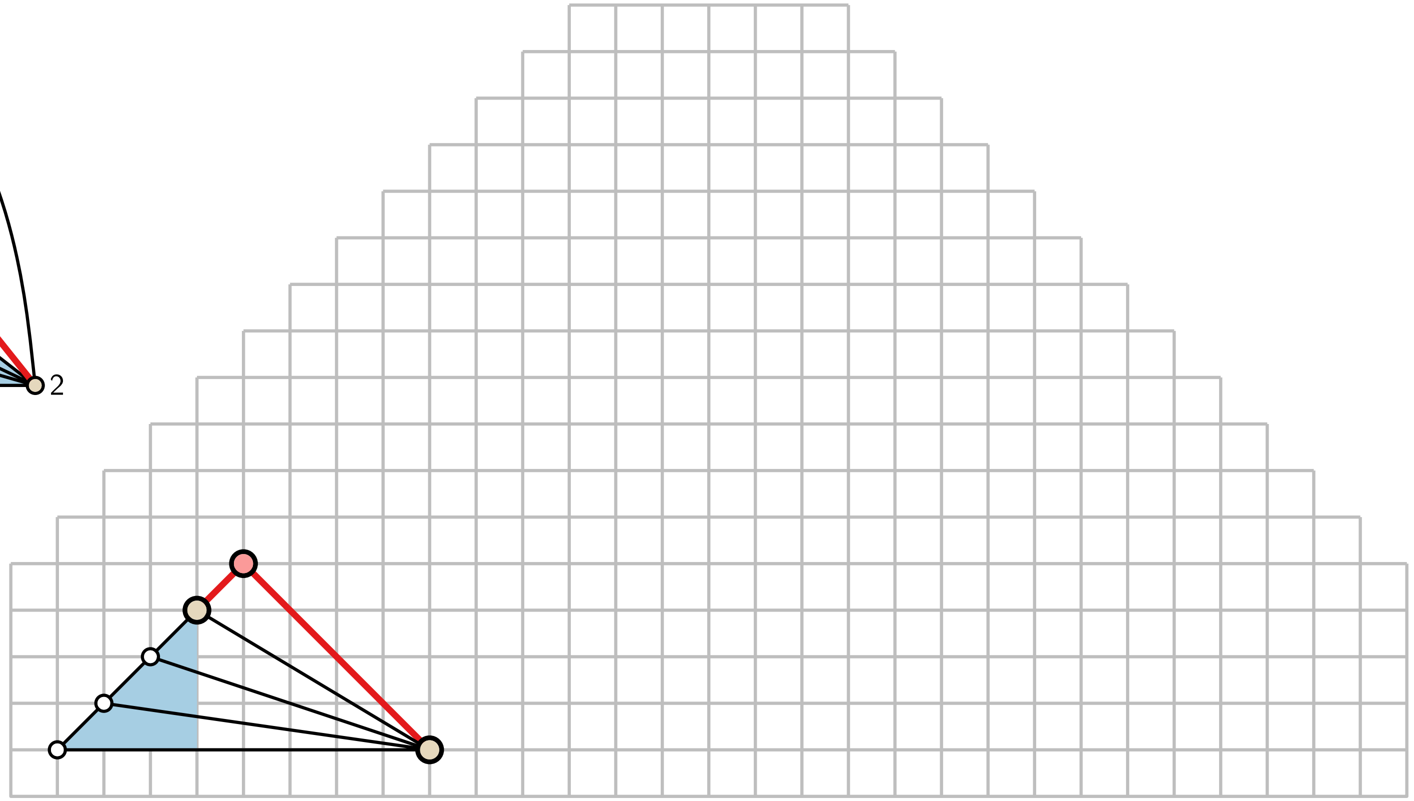




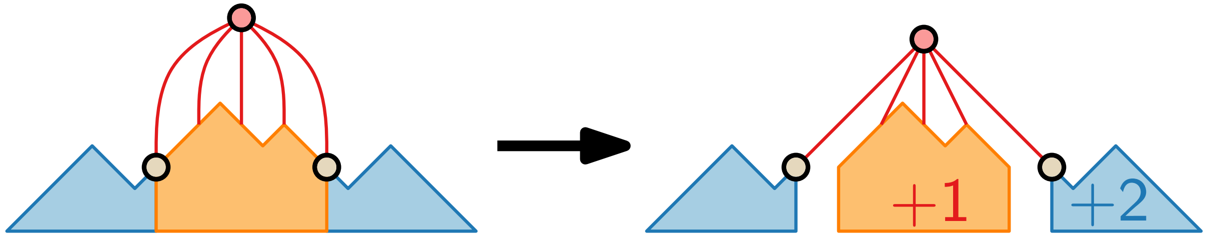
Shift Method – Example

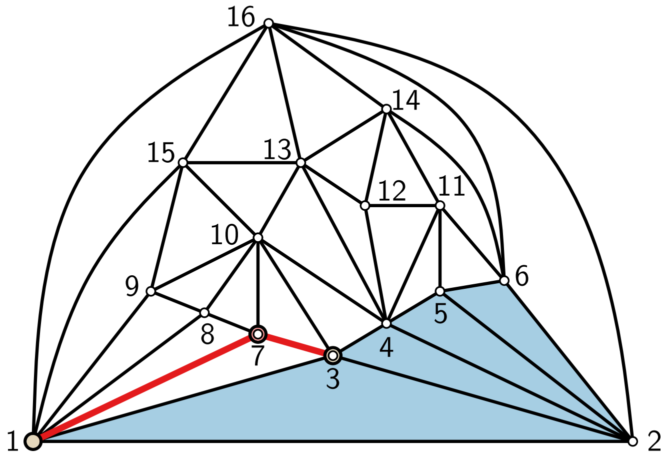


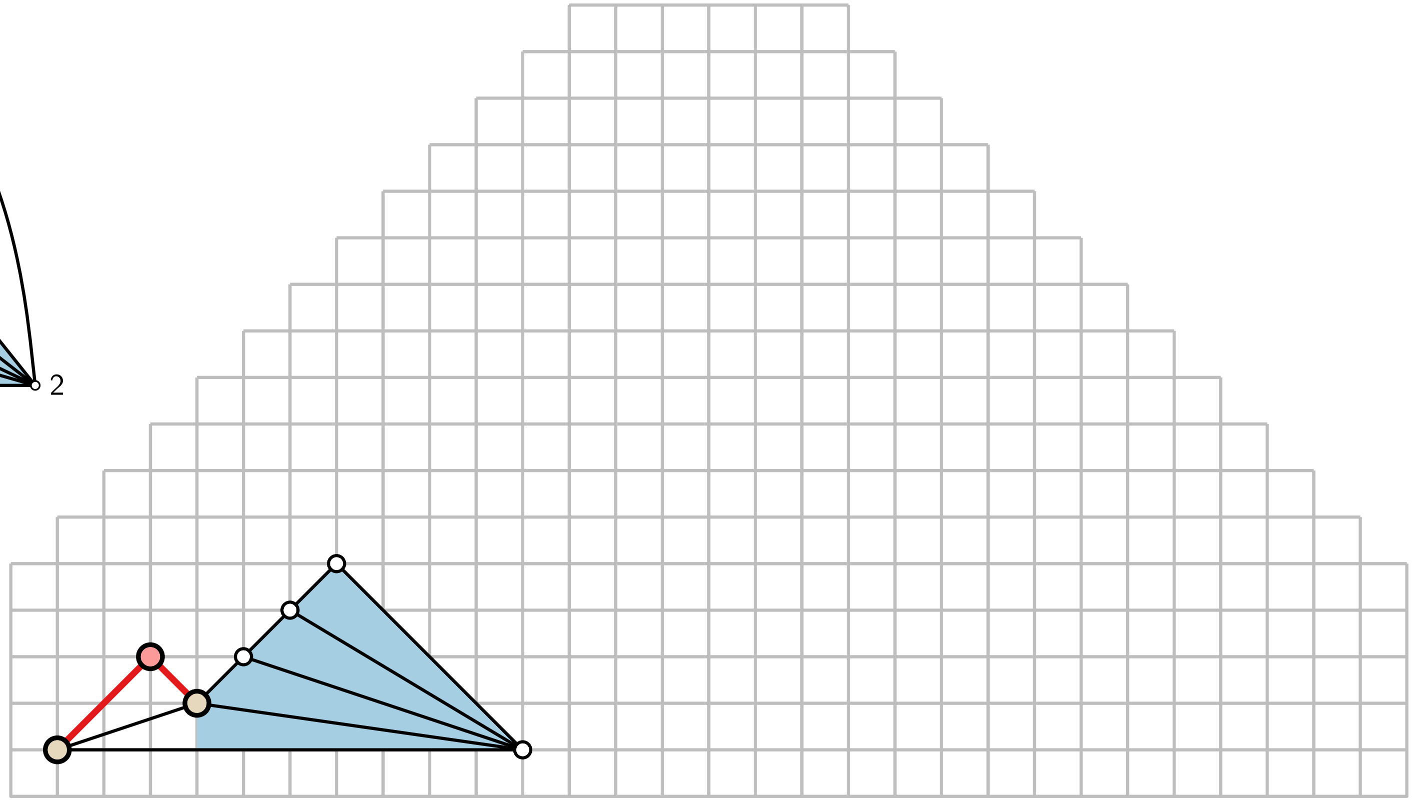




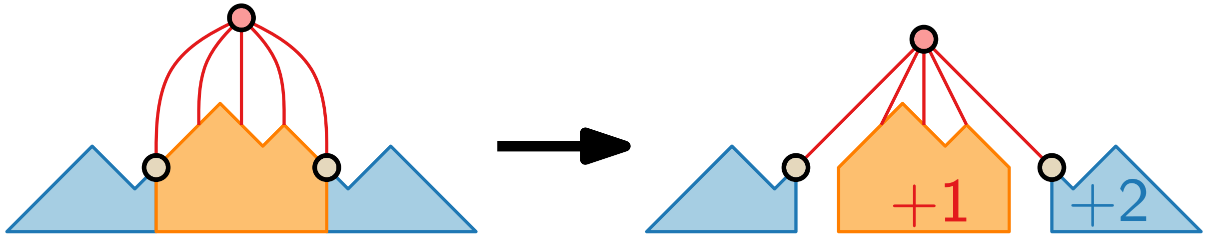
Shift Method – Example

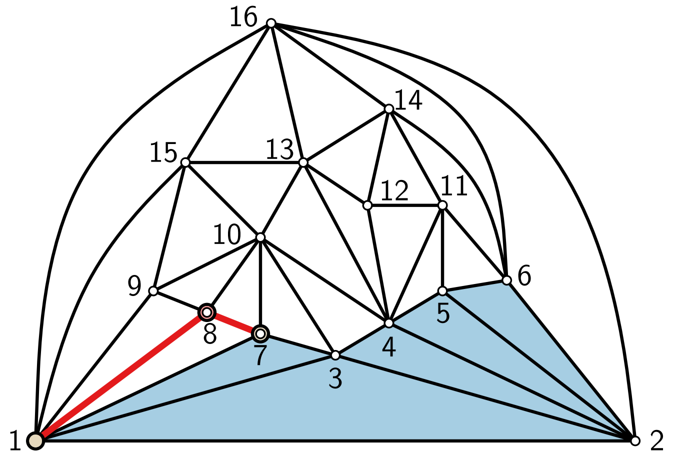


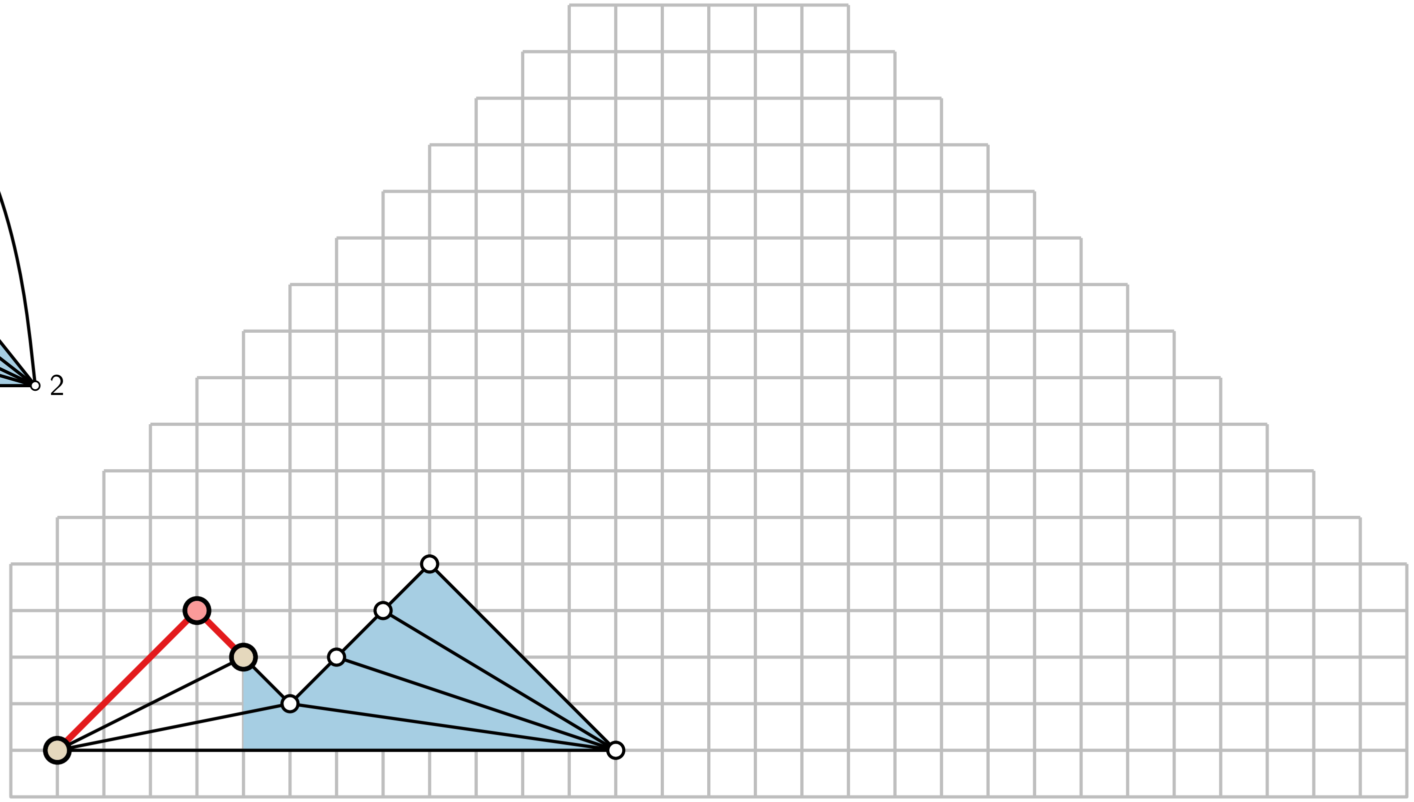




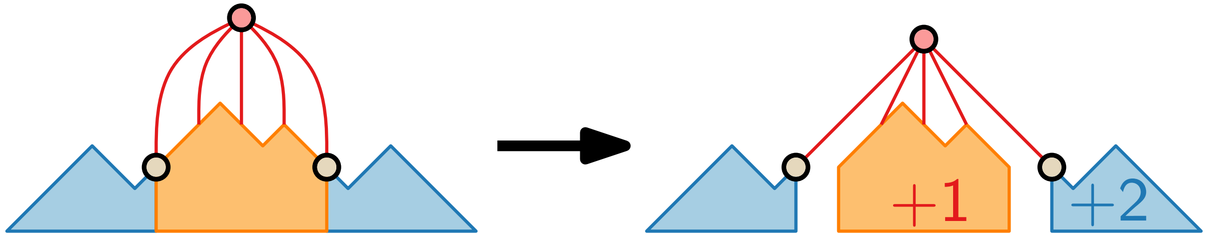
Shift Method – Example

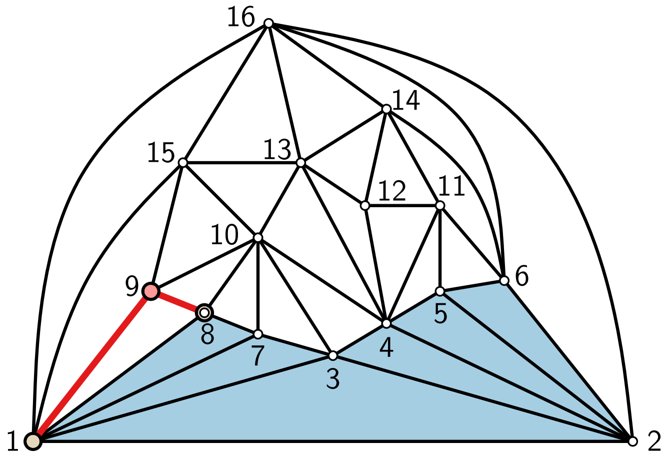


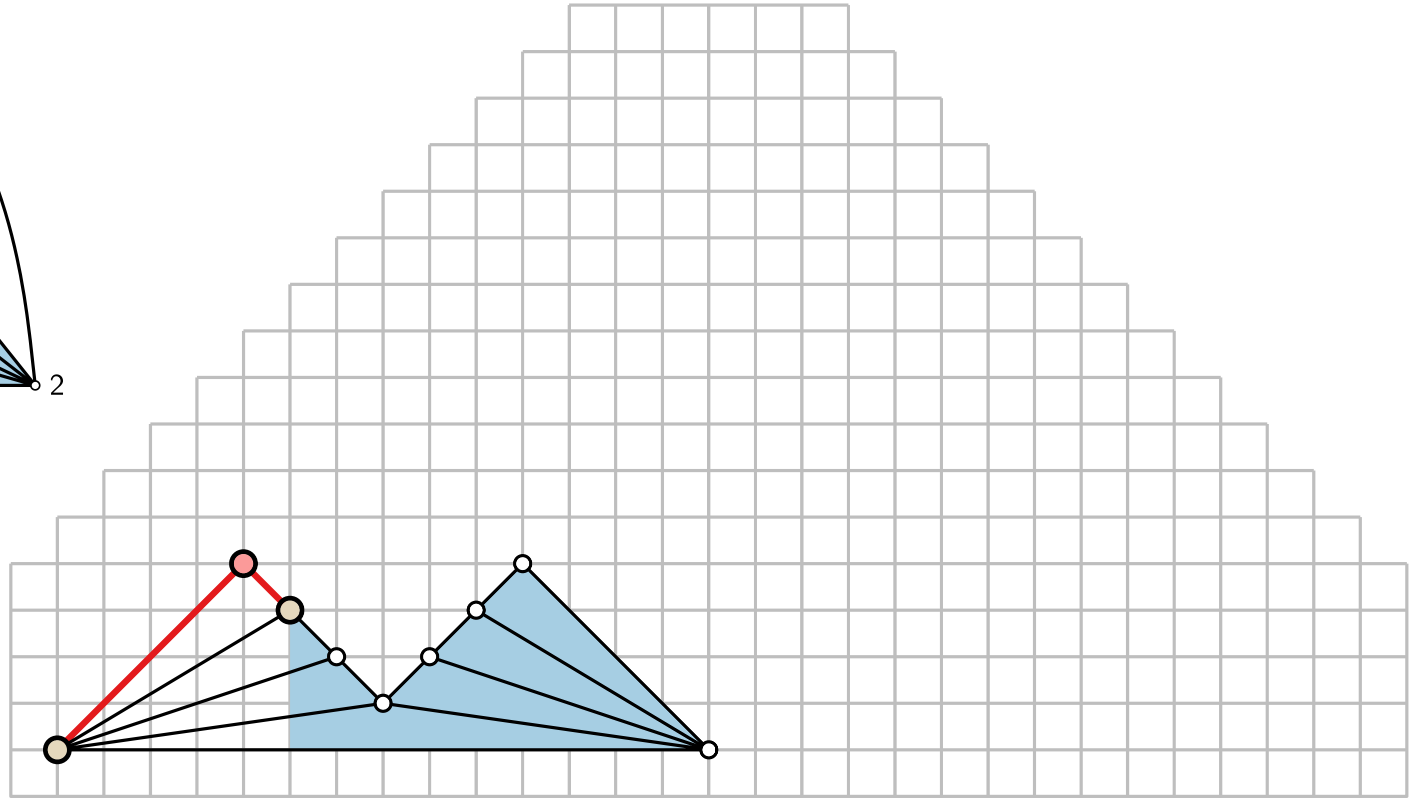




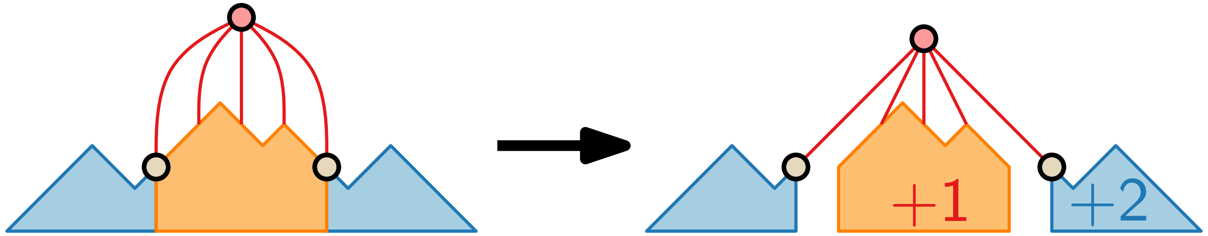
Shift Method – Example

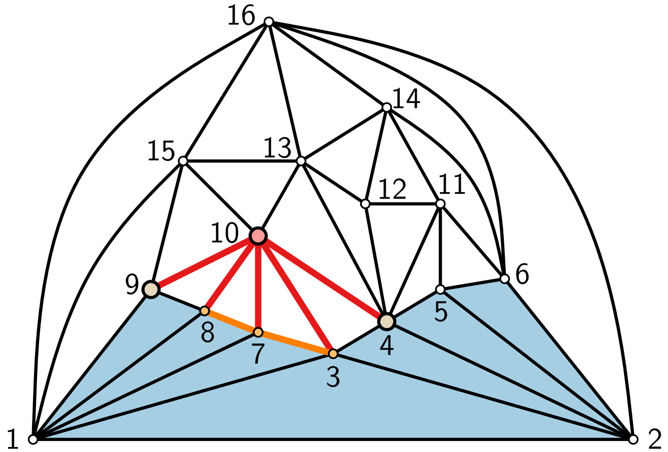


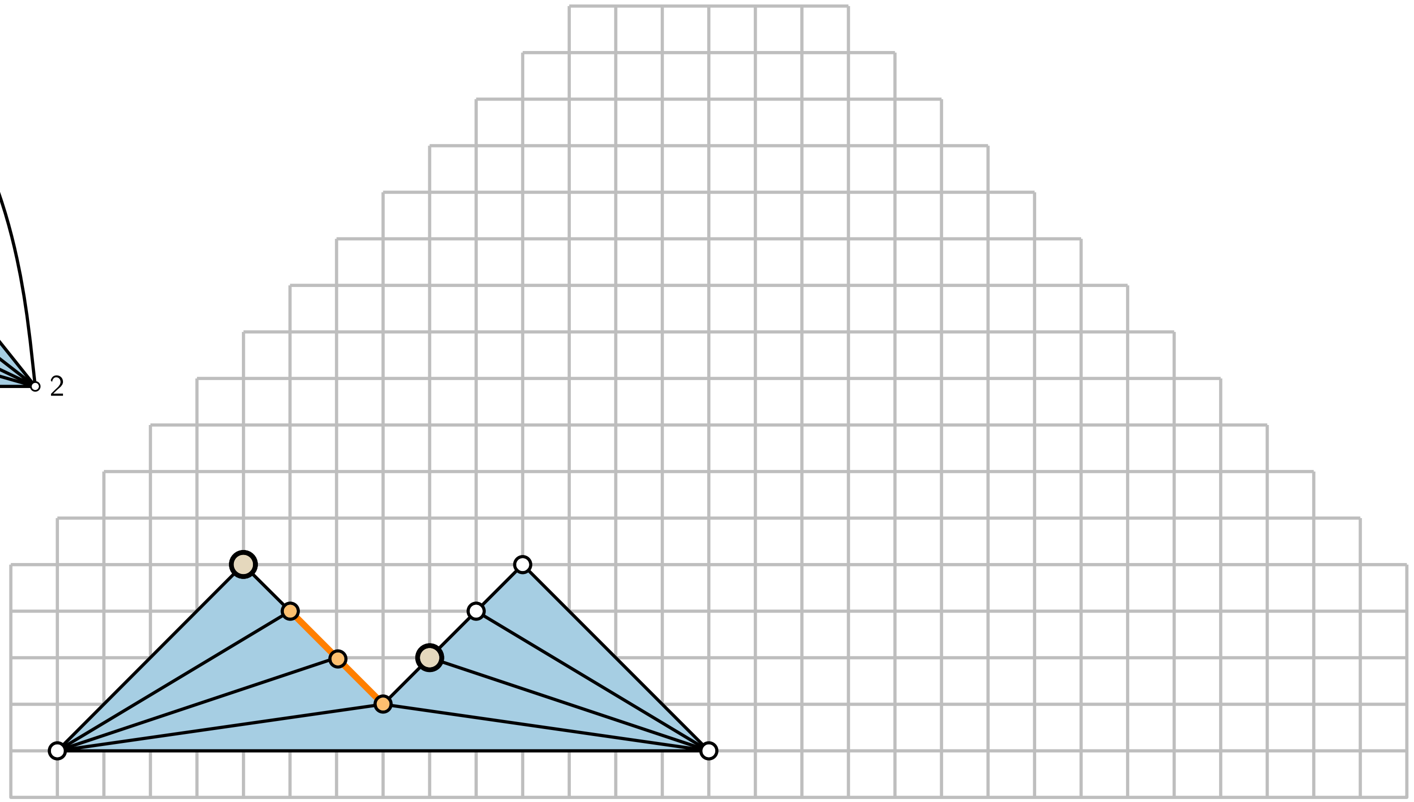




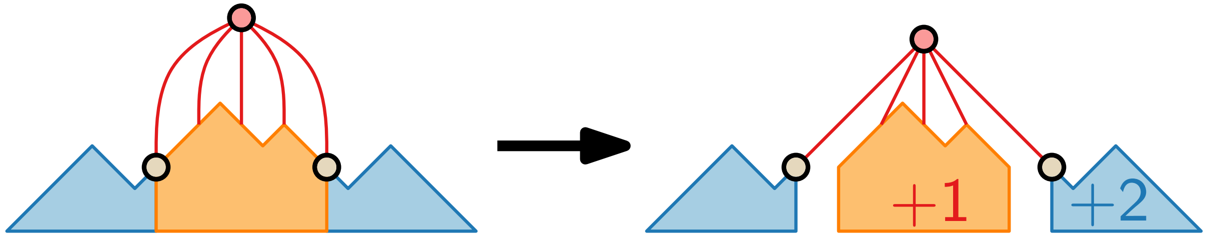
Shift Method – Example

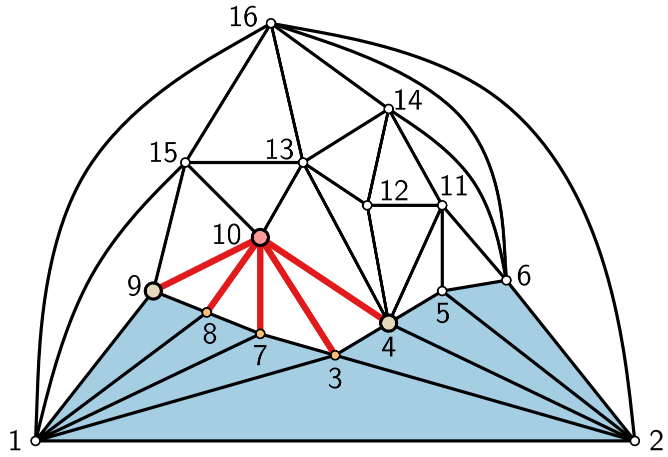


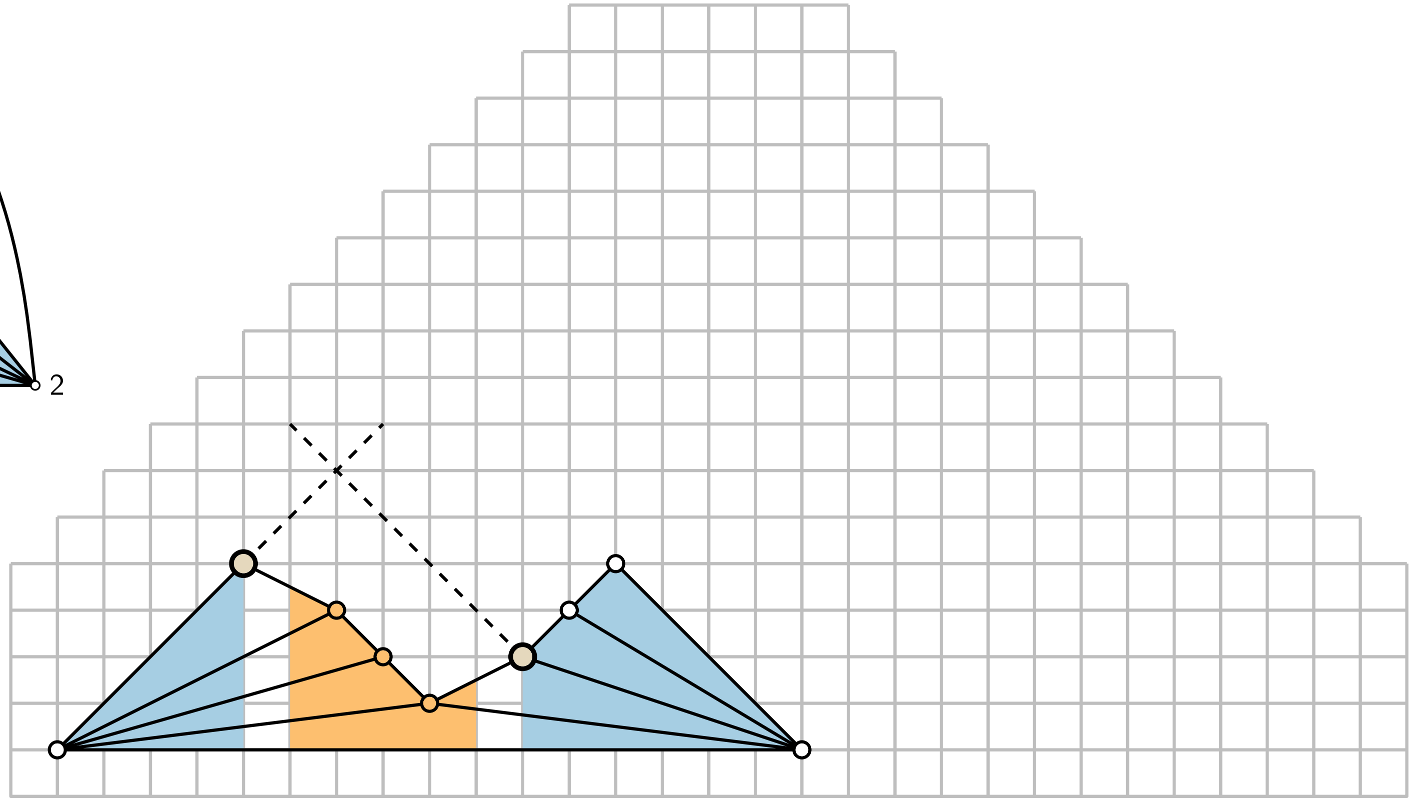




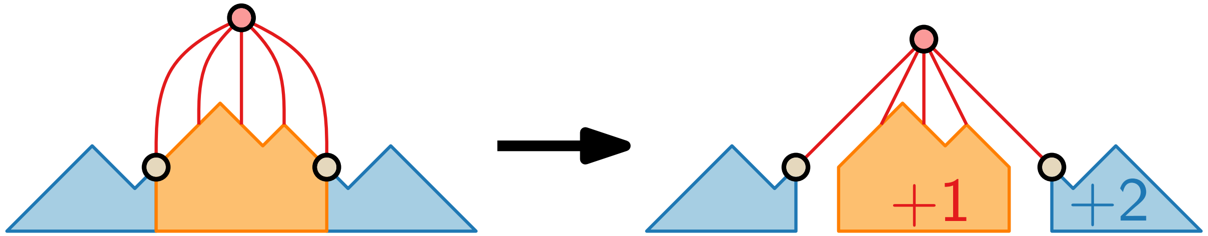
Shift Method – Example

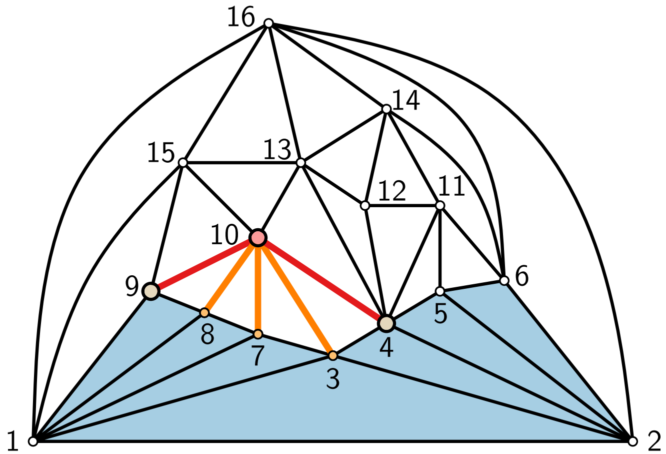


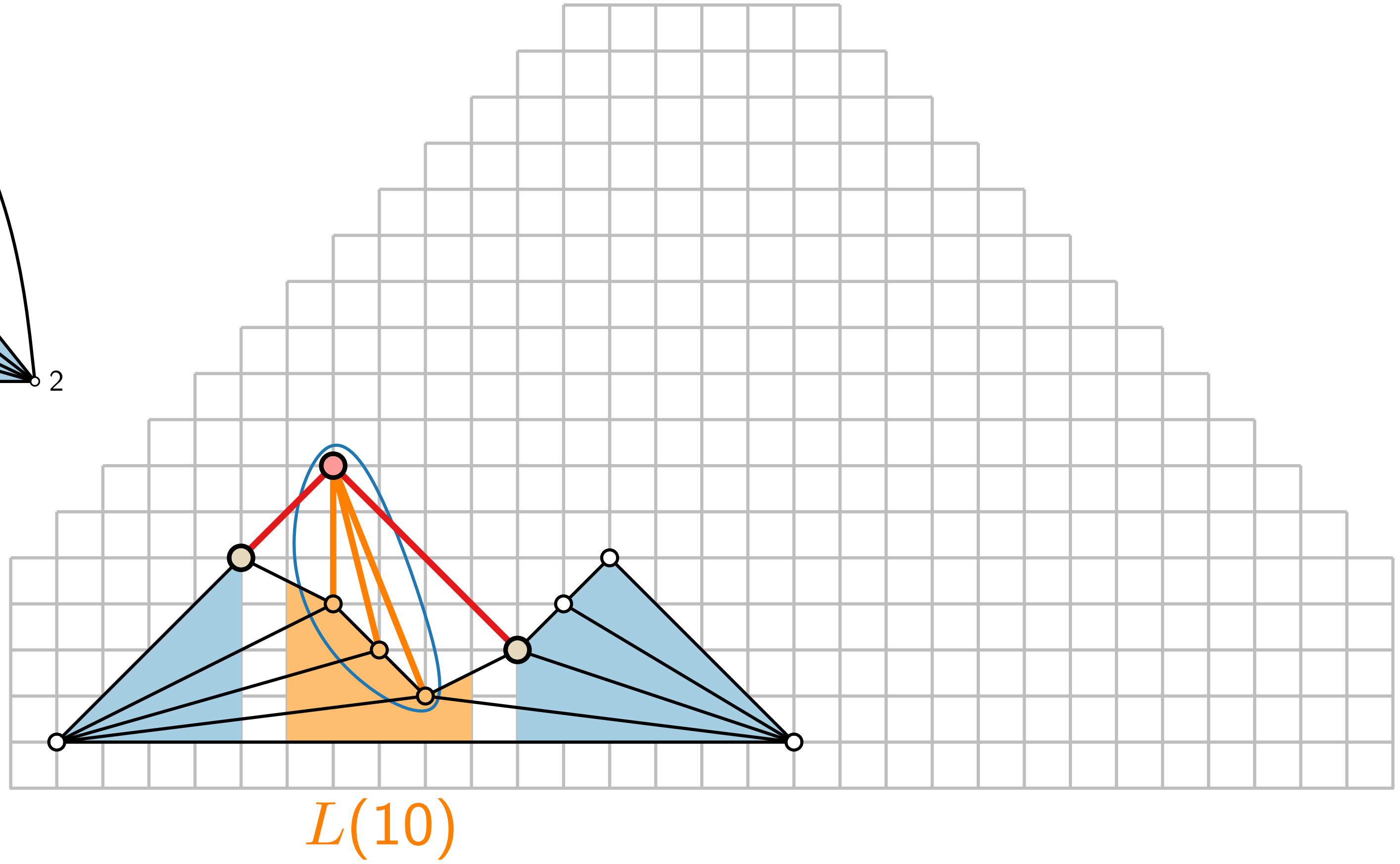




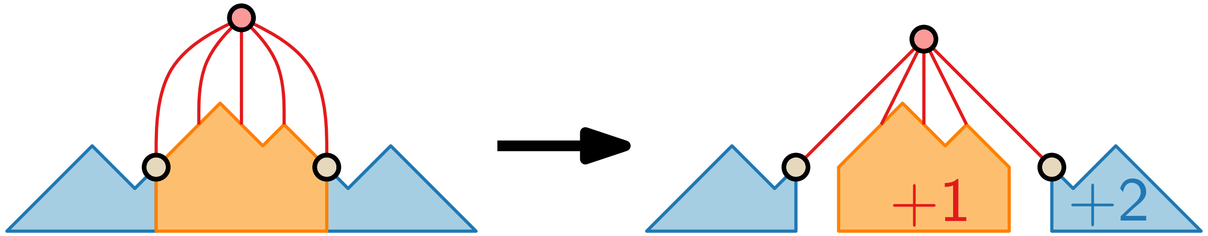
Shift Method – Example

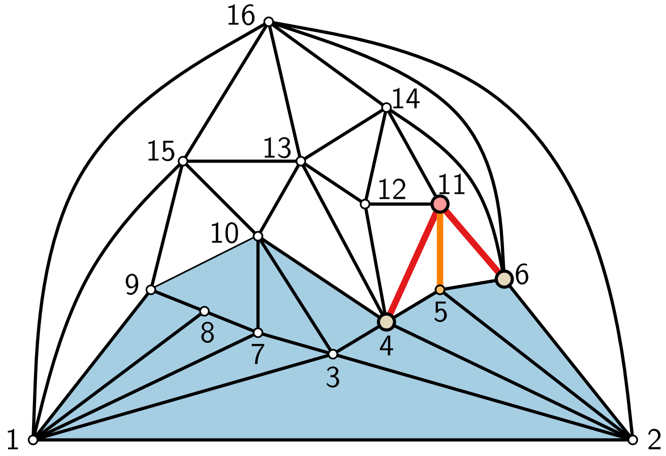


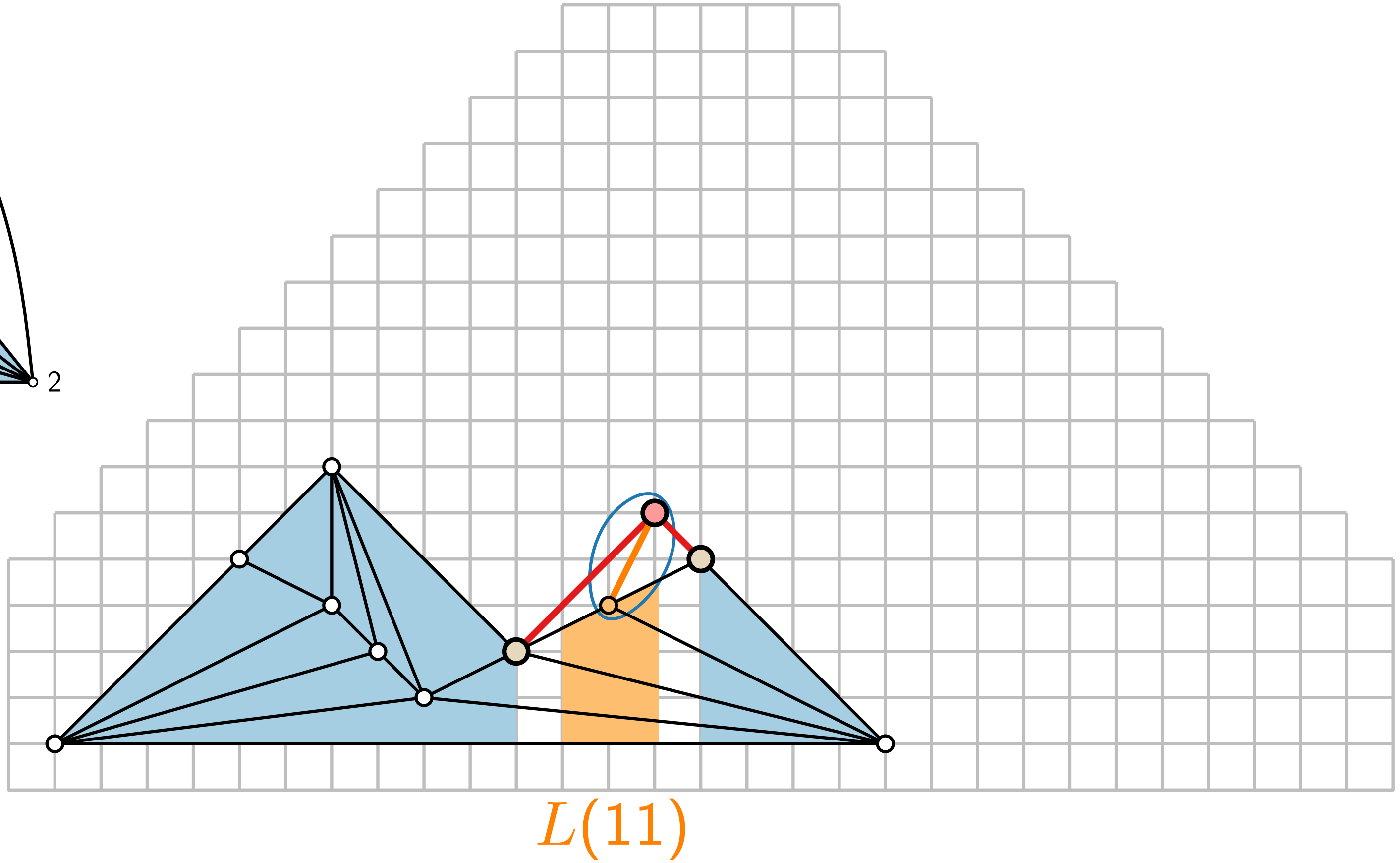




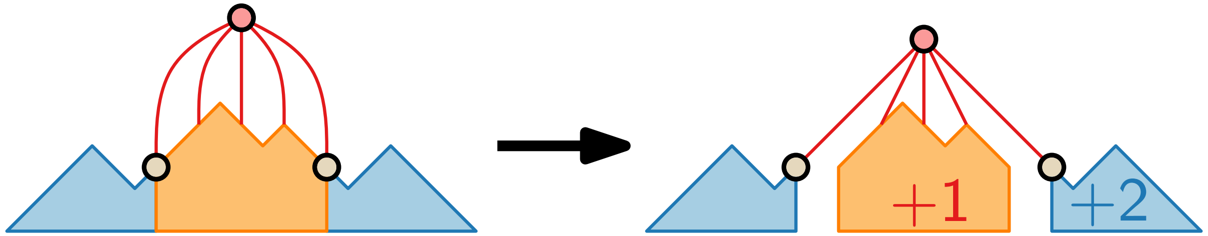
Shift Method – Example

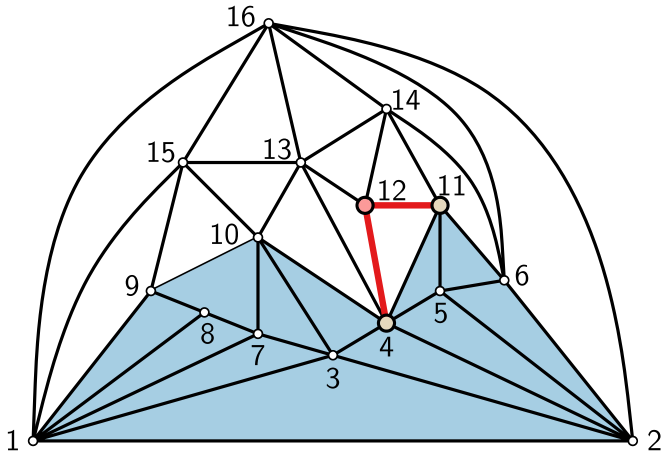


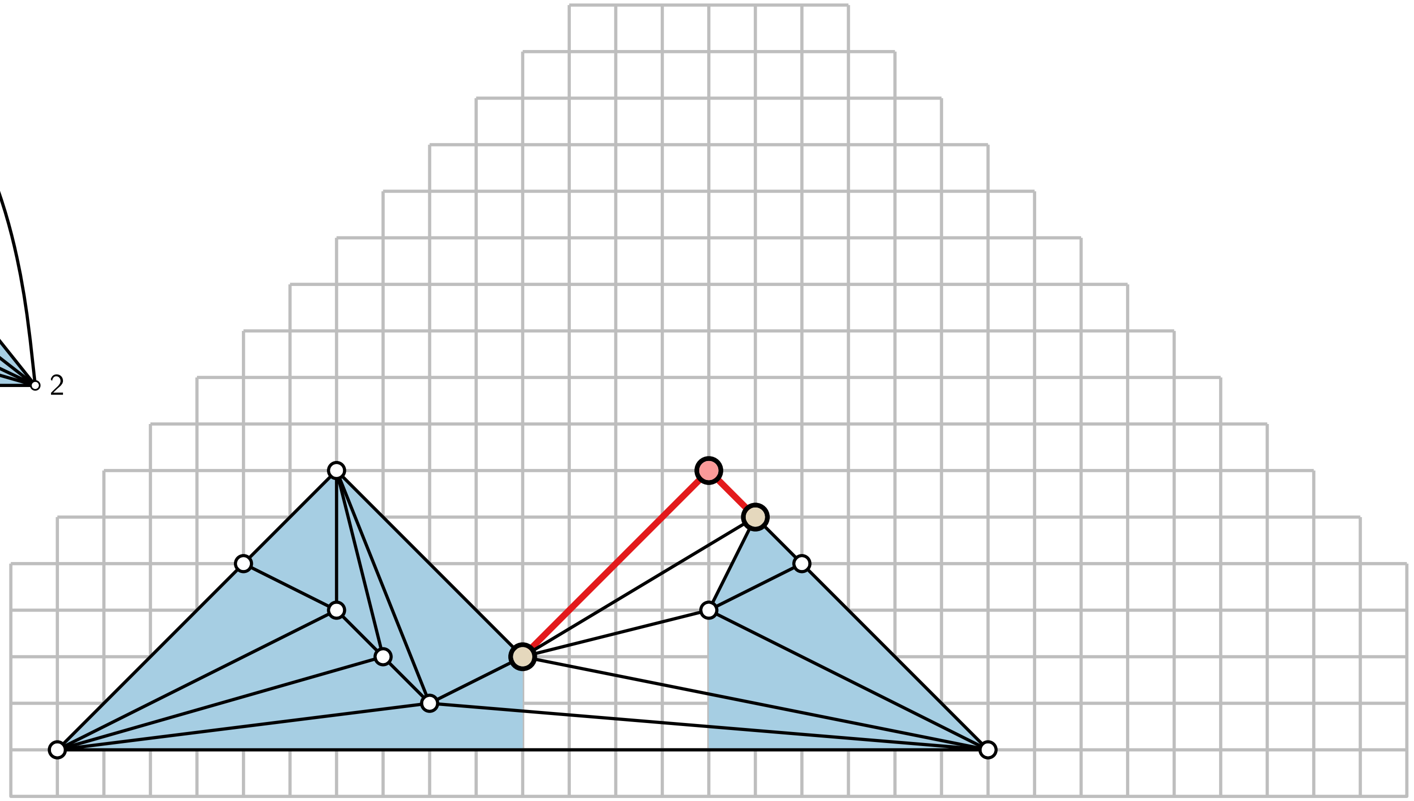




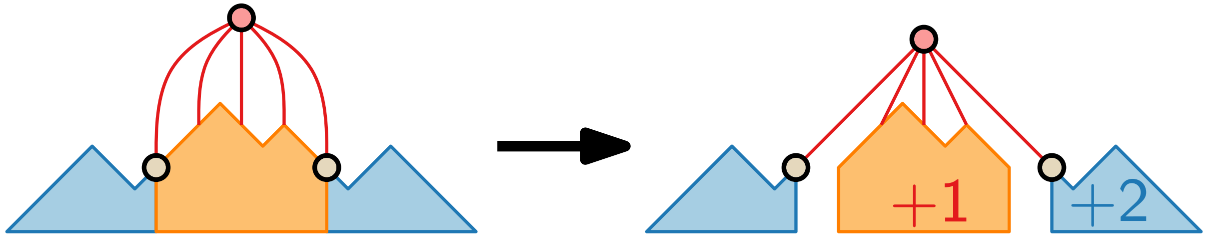
Shift Method – Example

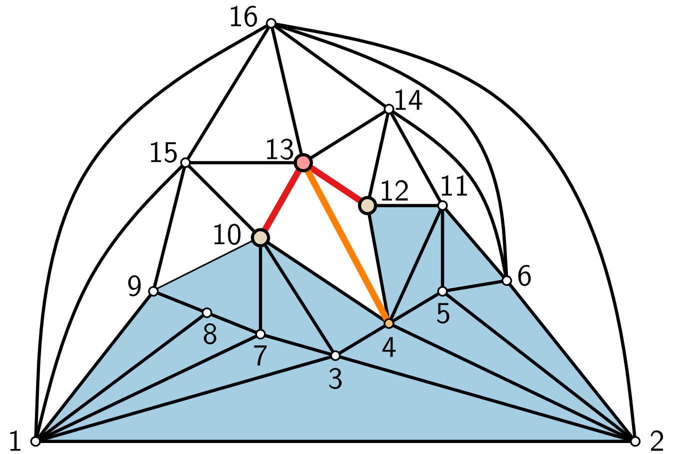


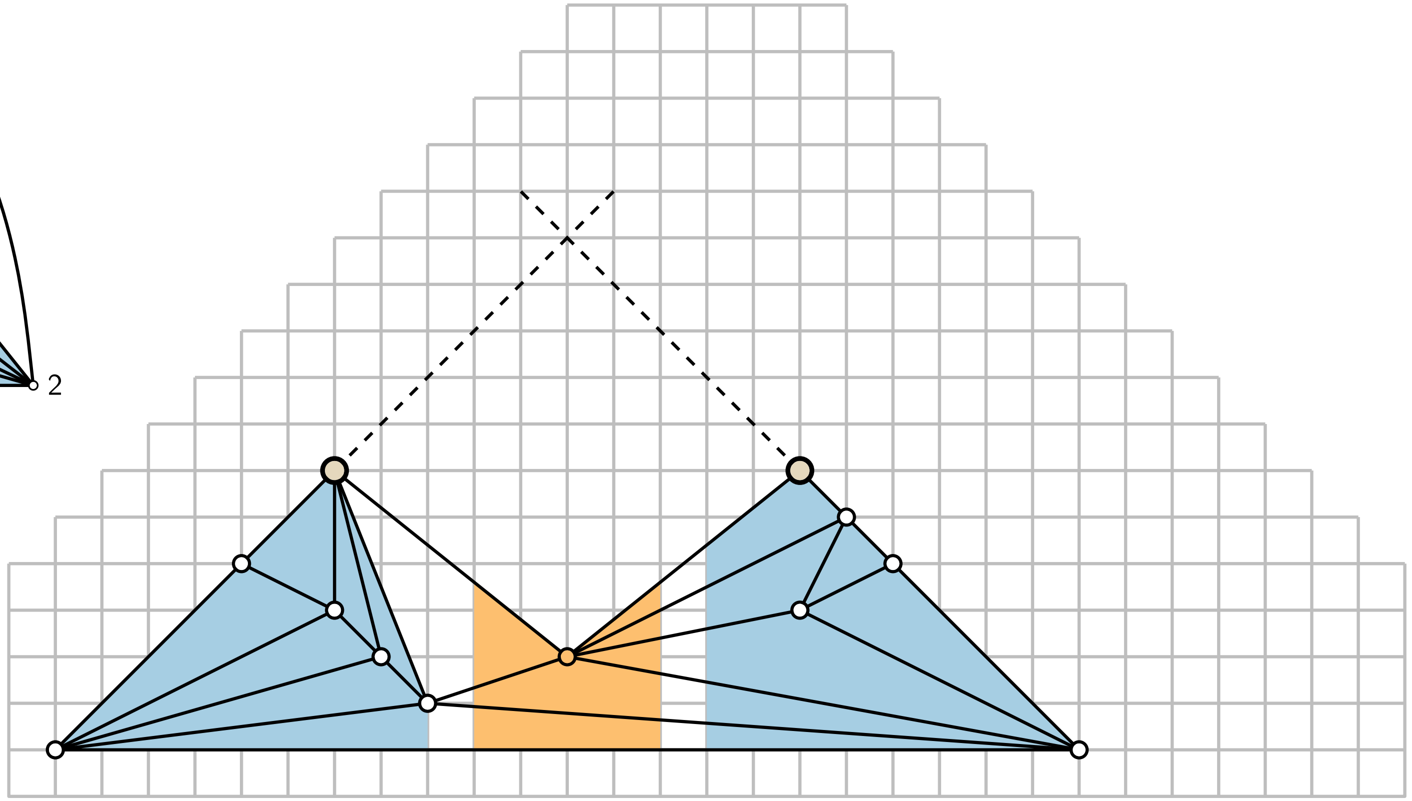




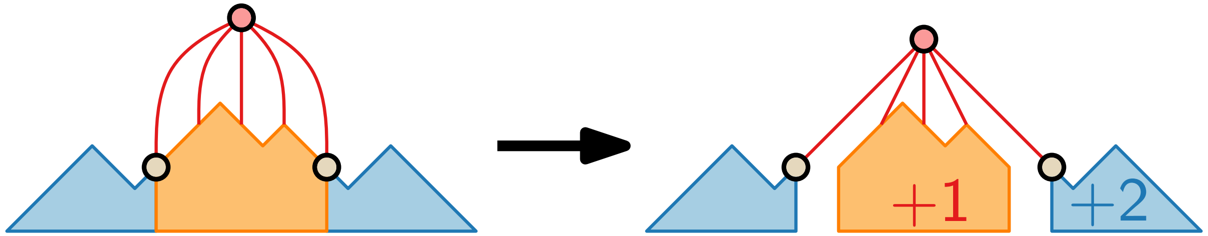
Shift Method – Example

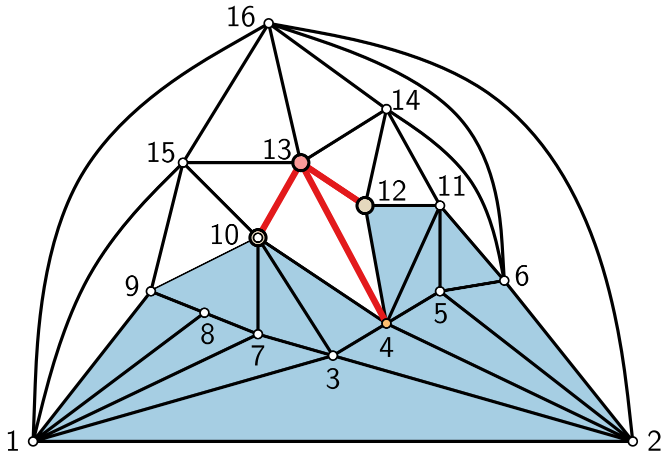


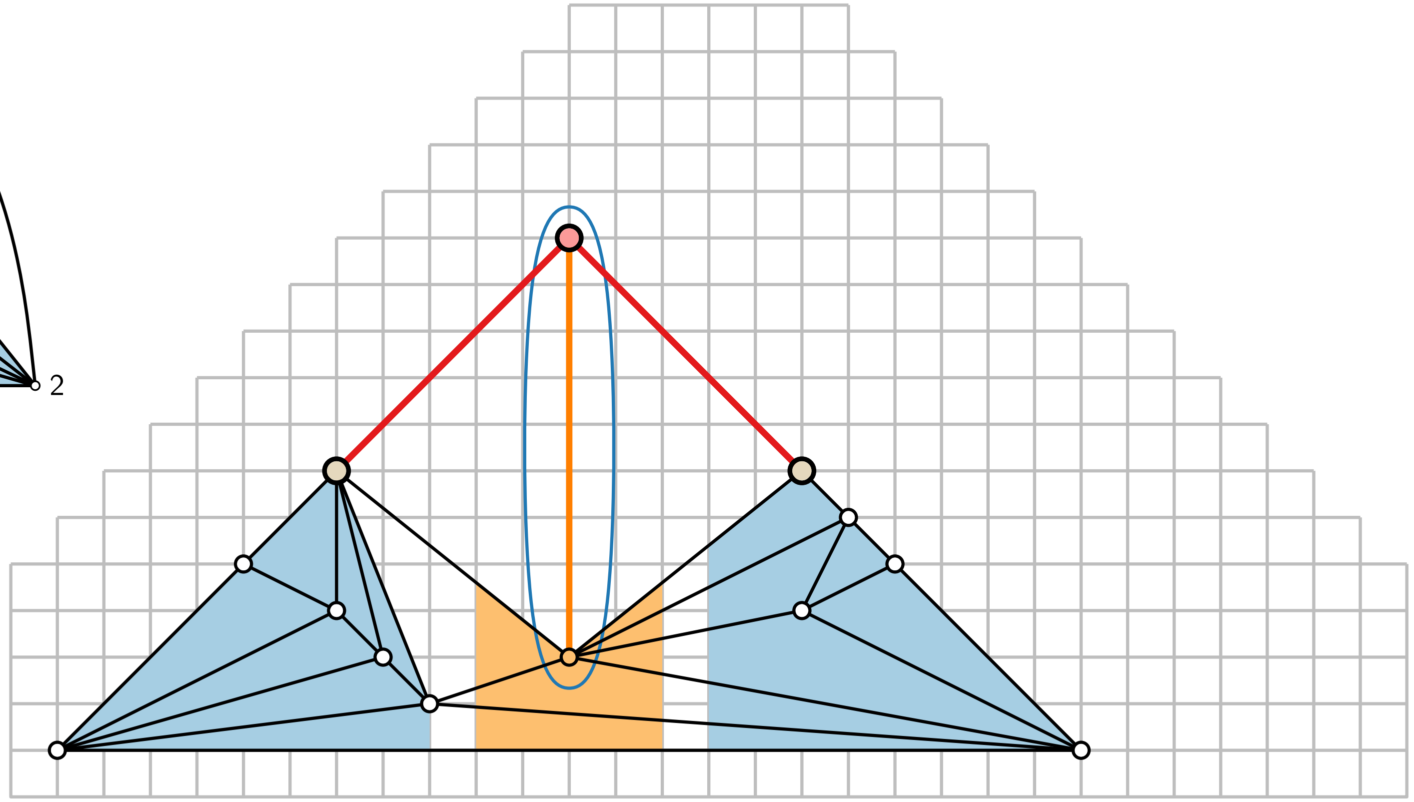




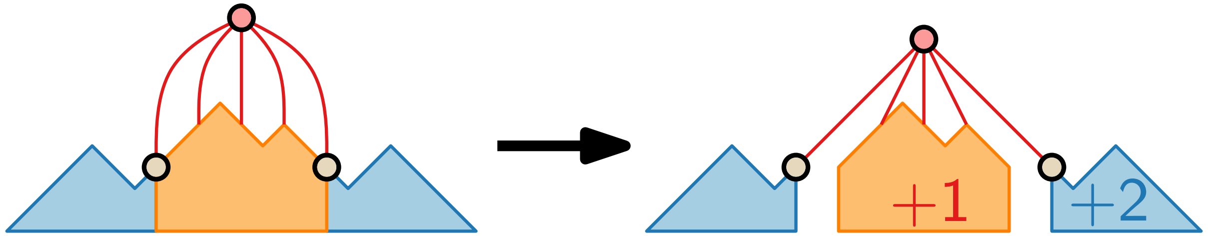
Shift Method – Example

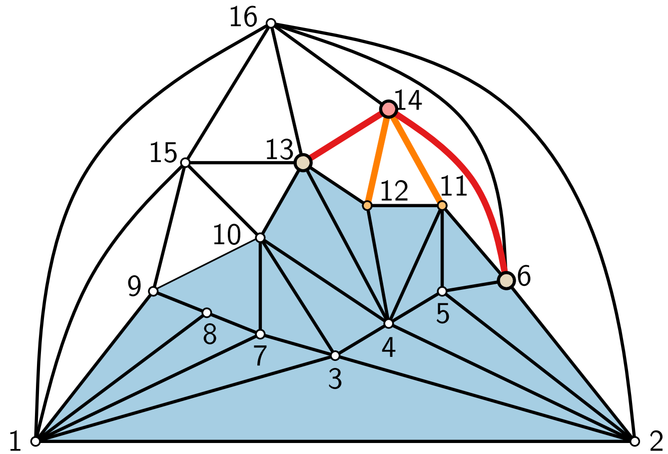


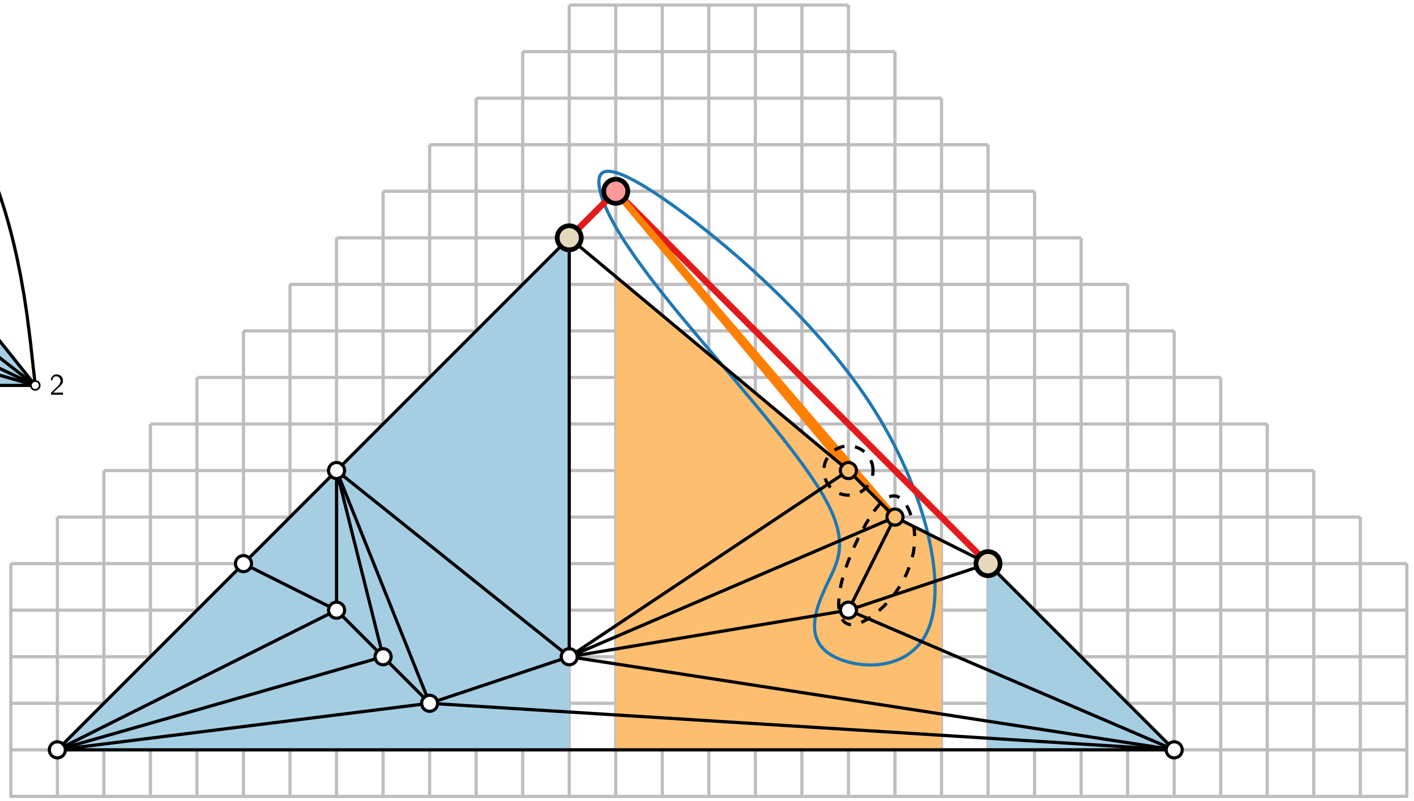




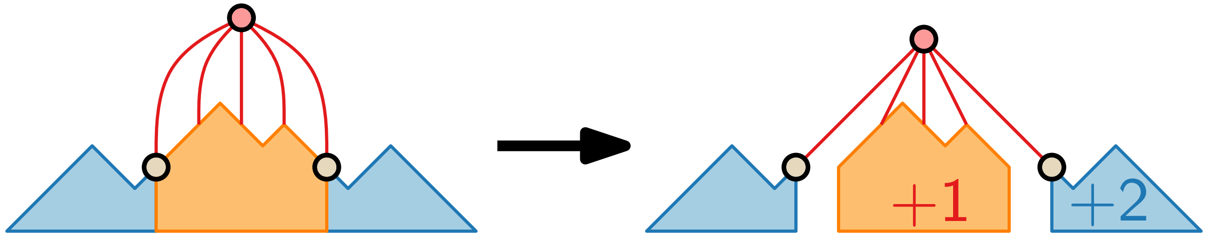
Shift Method – Example

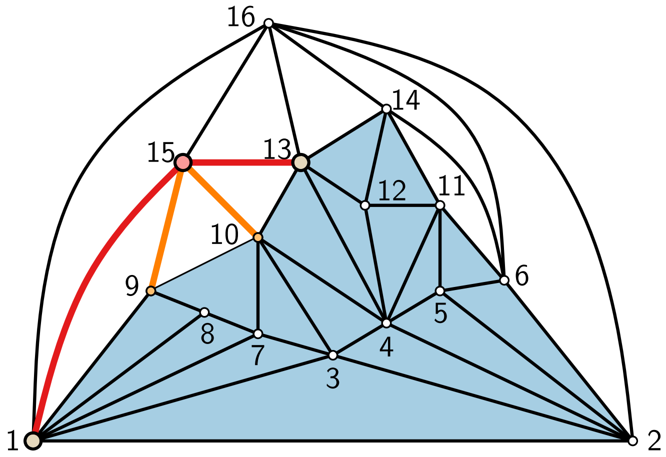


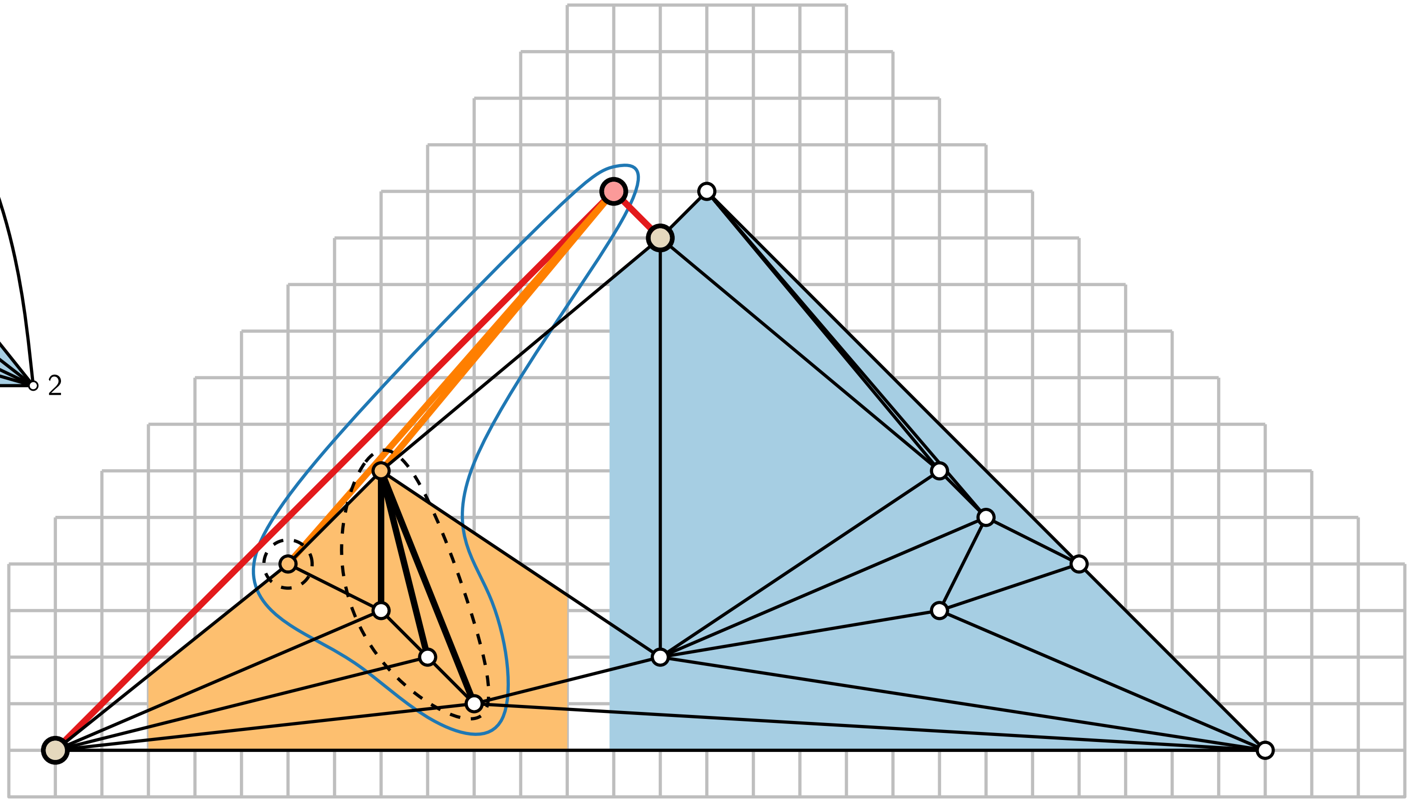




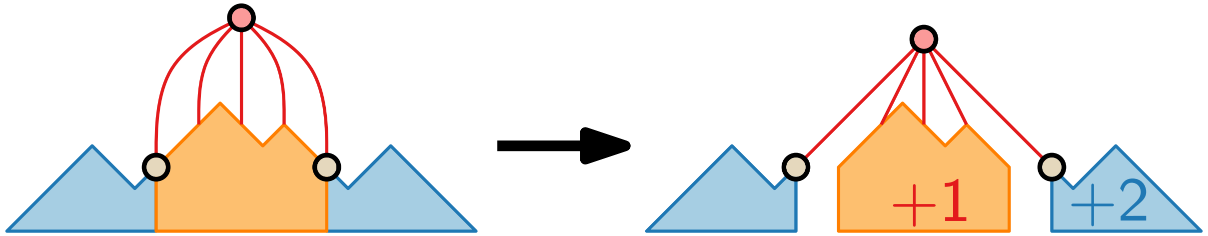
Shift Method – Example

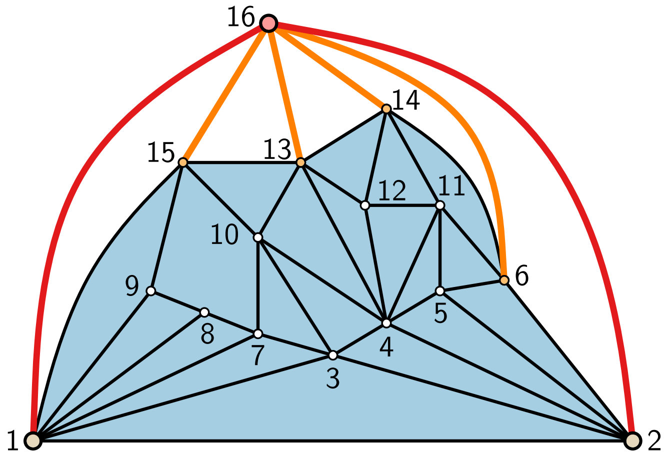


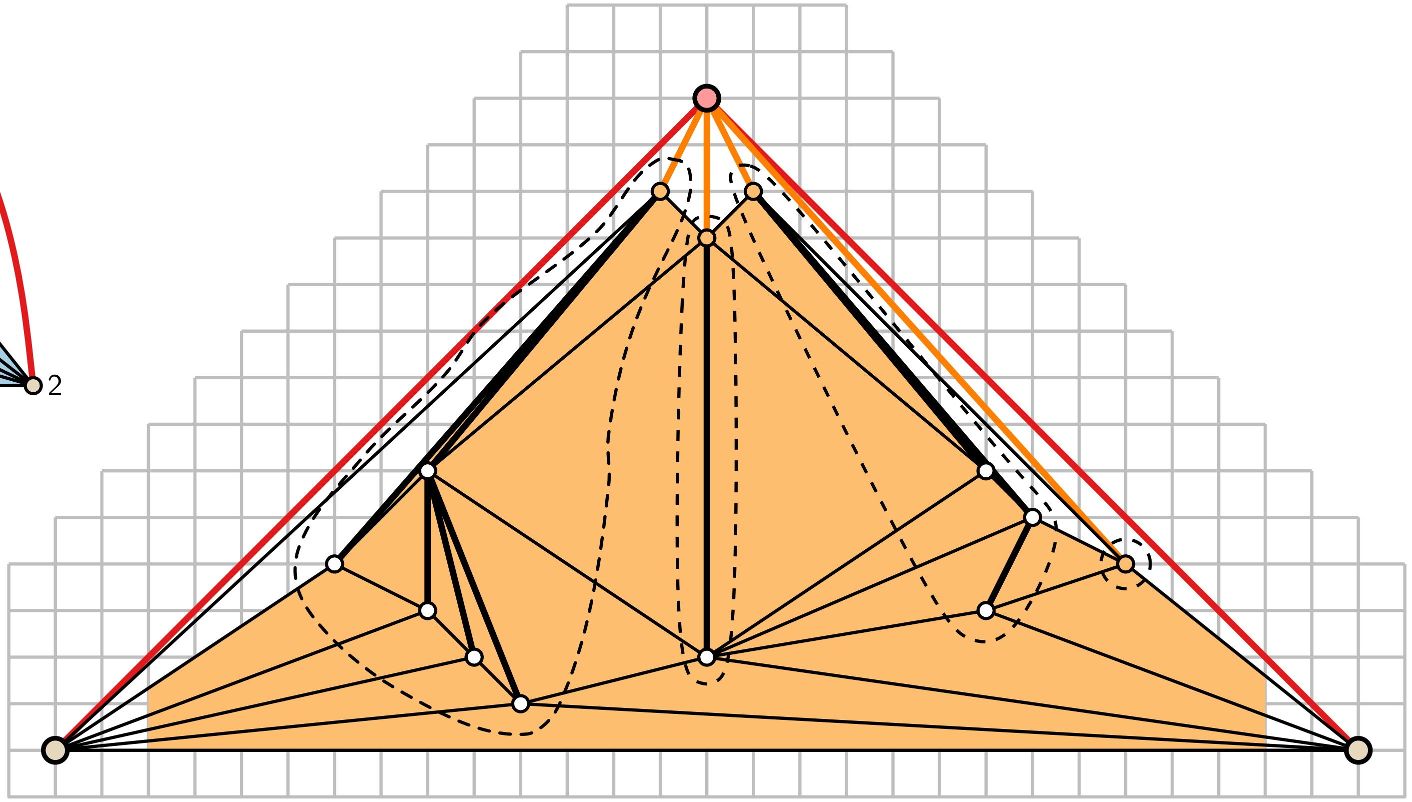




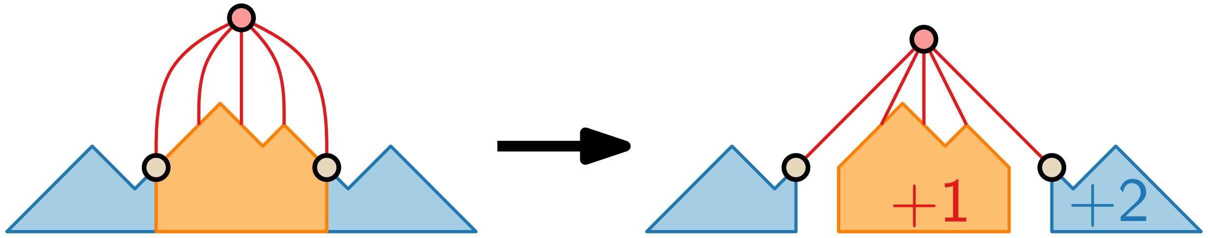
Shift Method – Example

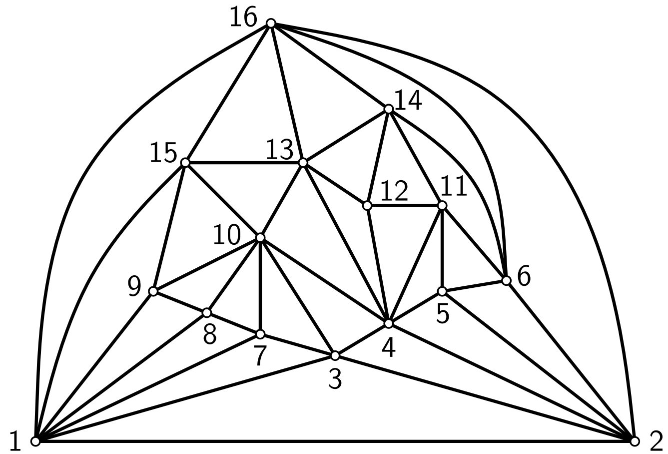


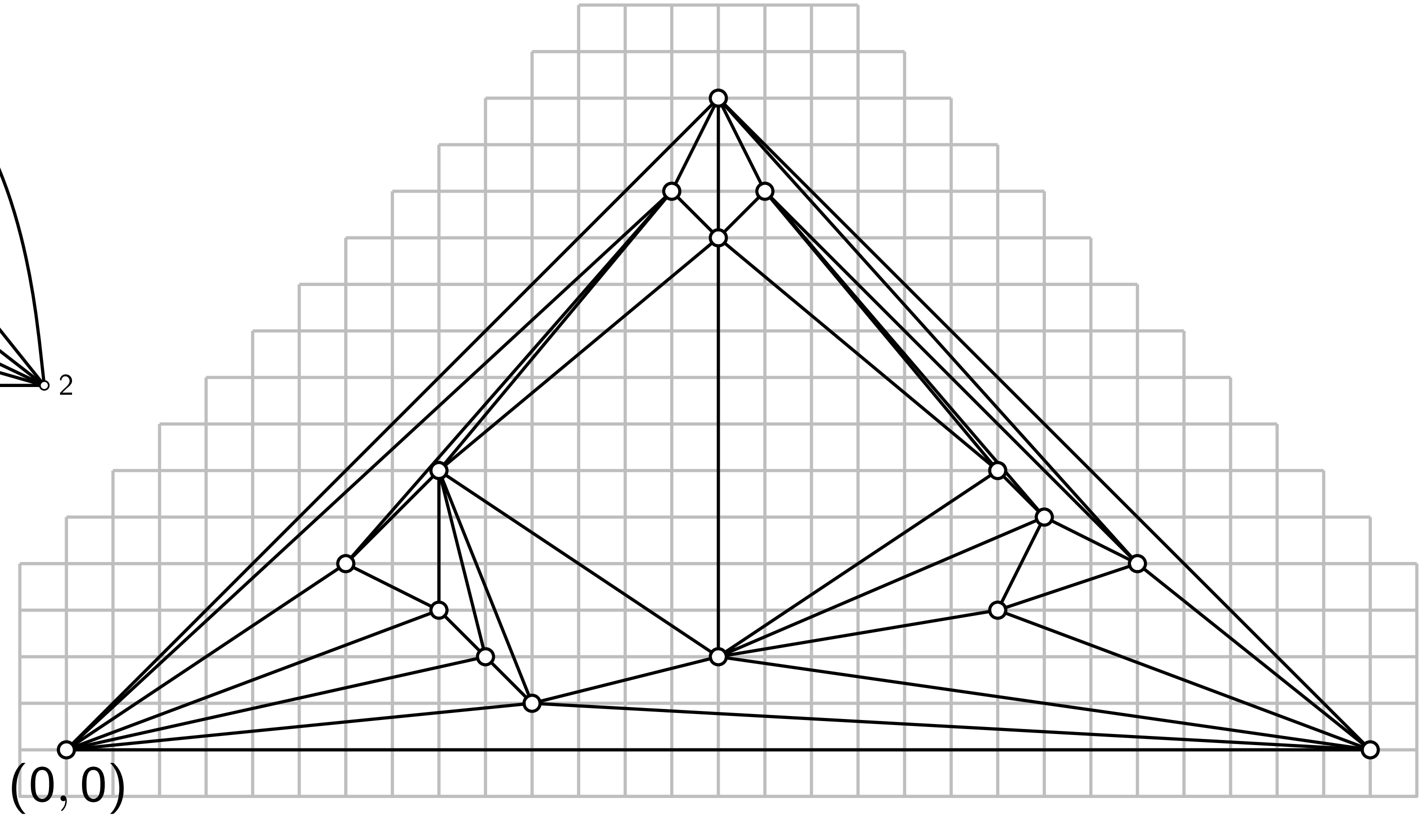




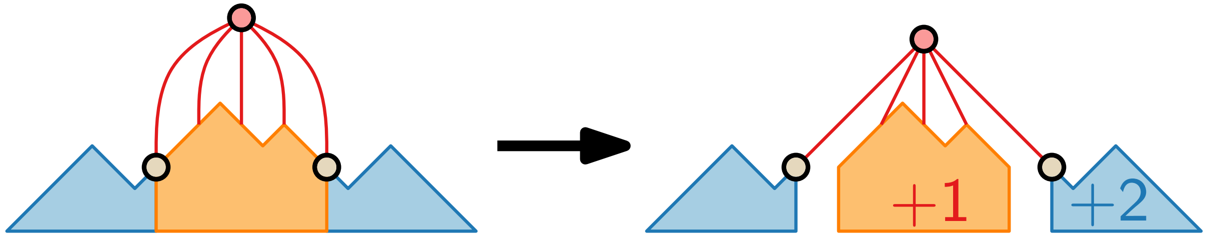
Shift Method – Example

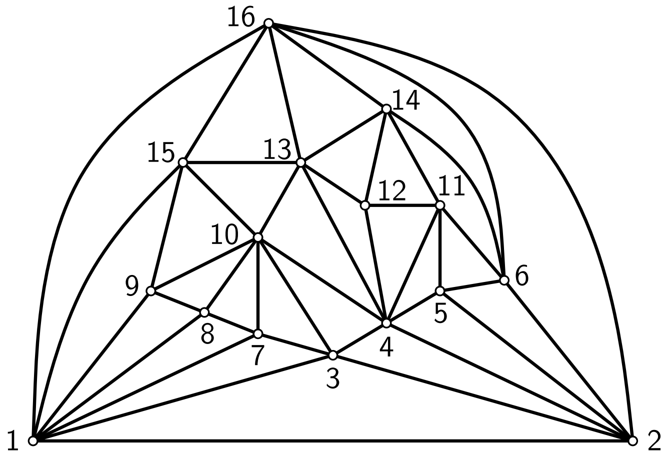


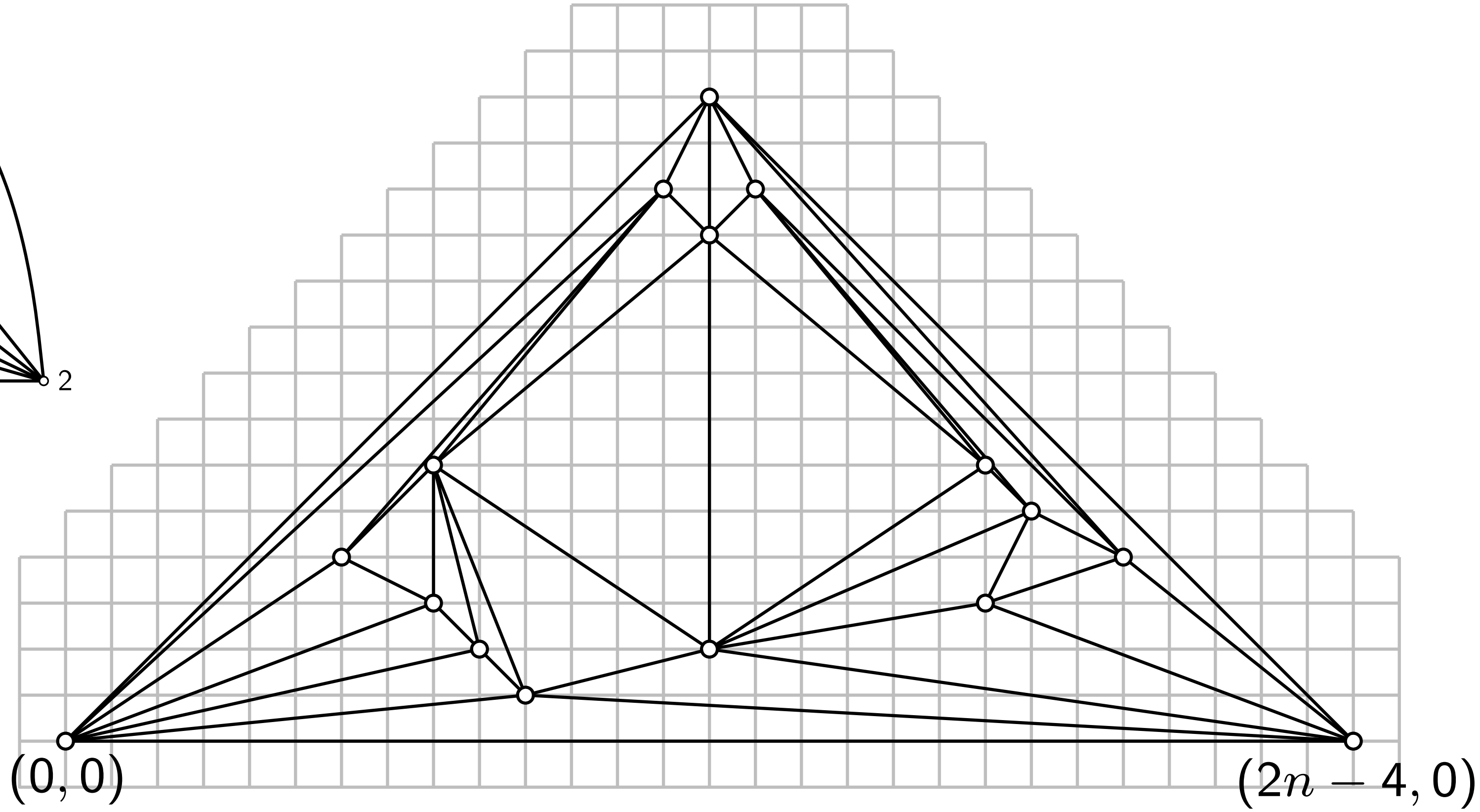




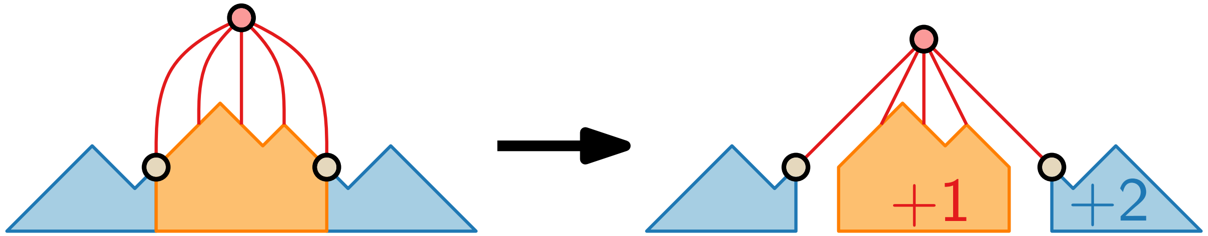
Shift Method – Example

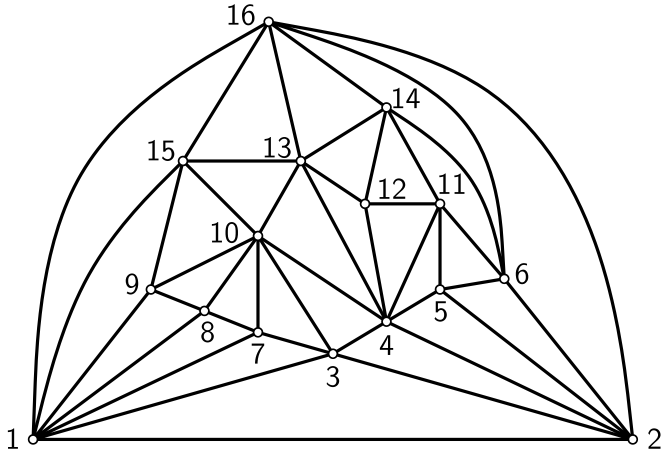


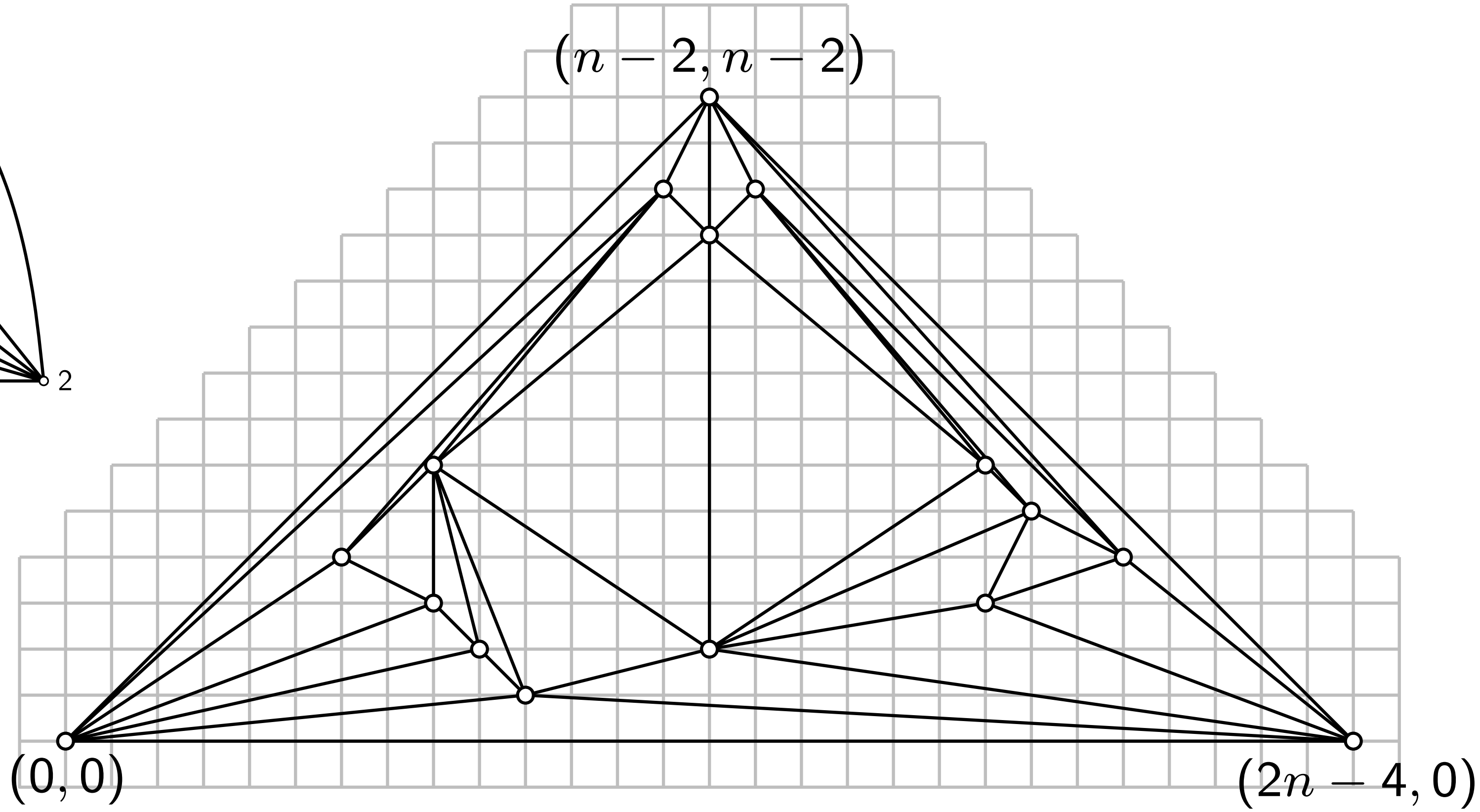




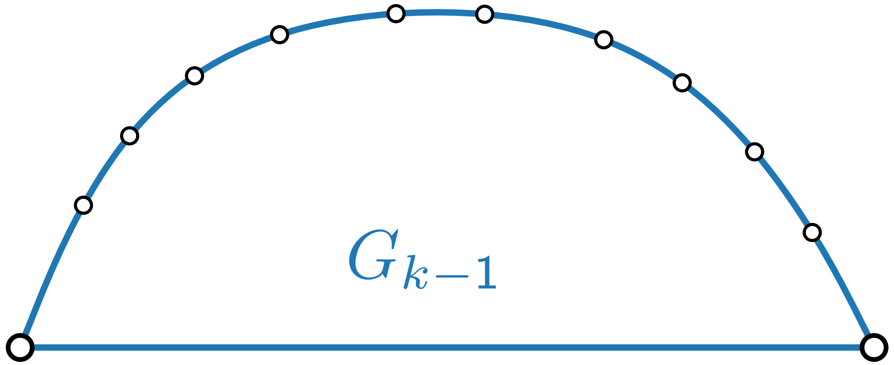
Shift Method – Example



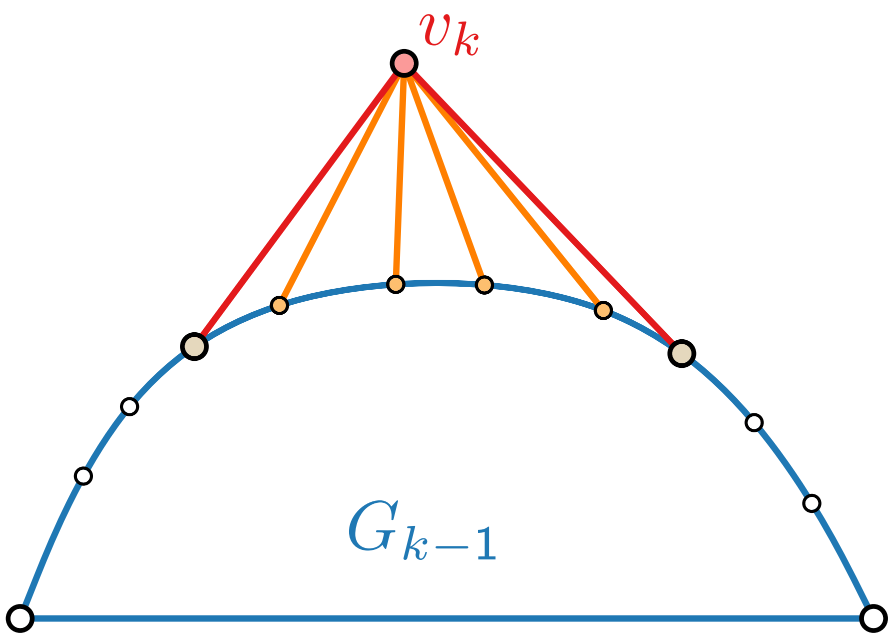




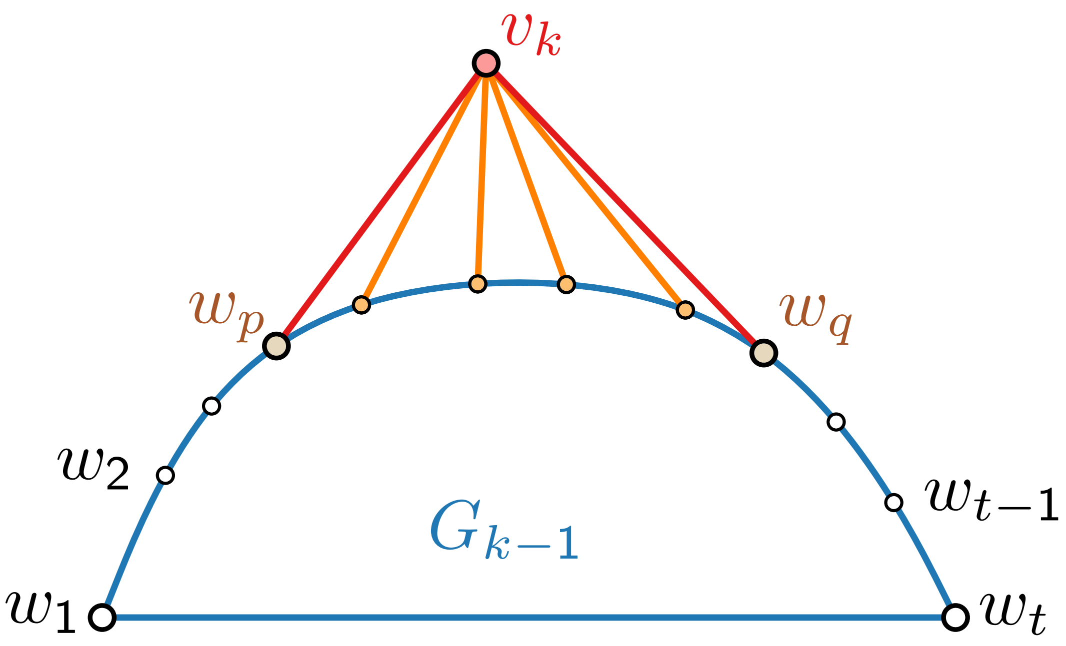
Shift Method – Planarity



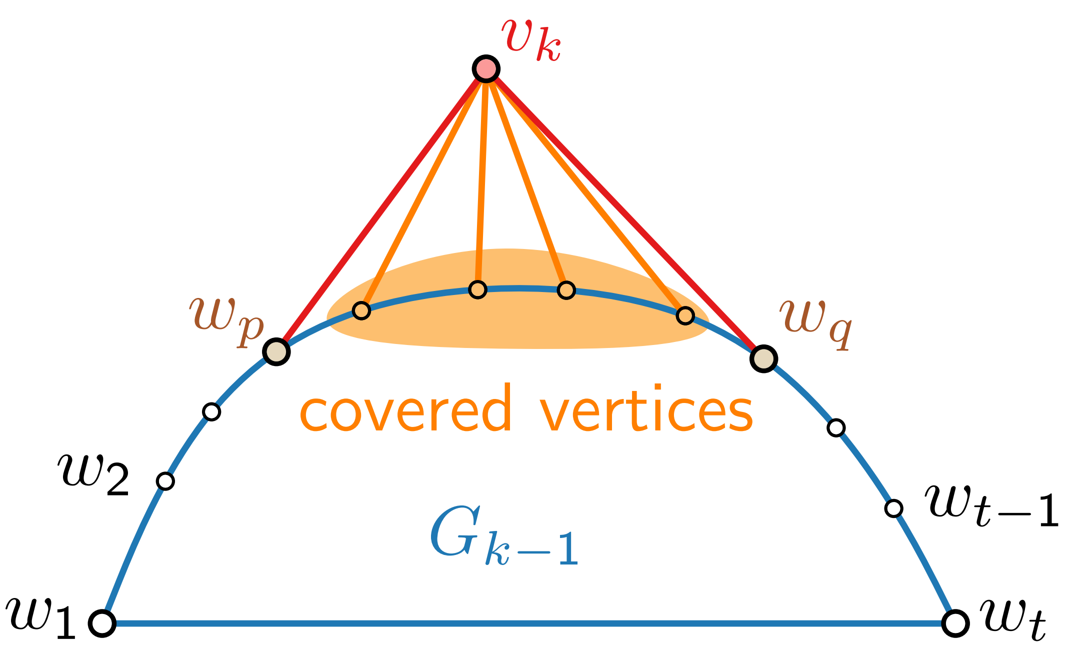
Shift Method – Planarity



Shift Method – Planarity



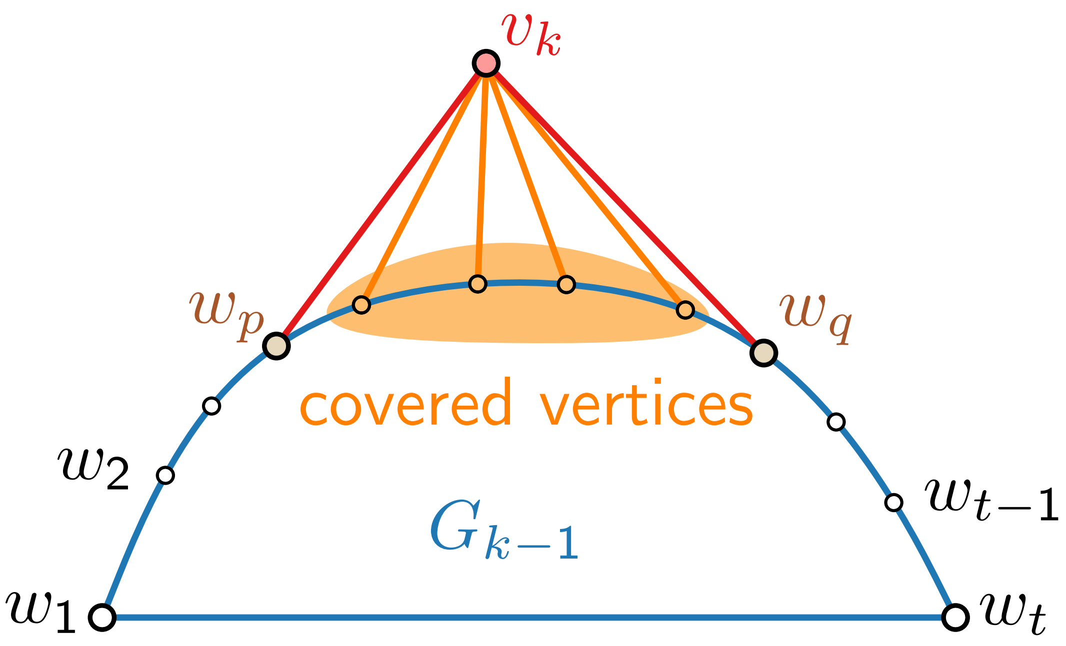
Shift Method – Planarity



Shift Method – Planarity

Observations.

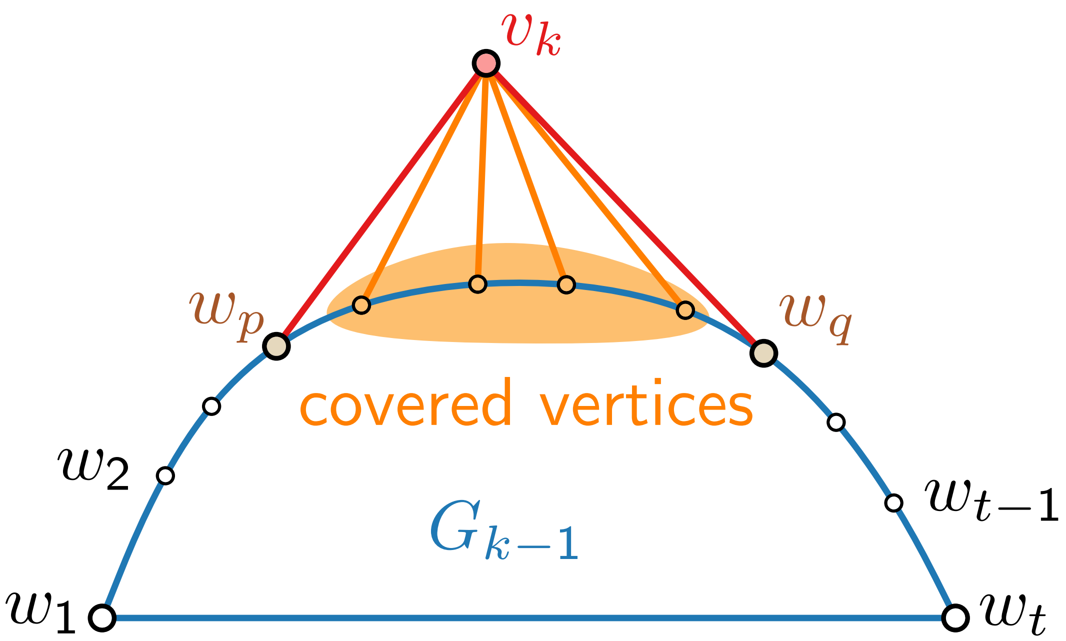
- Each internal vertex is covered exactly once.



Shift Method – Planarity

Observations.

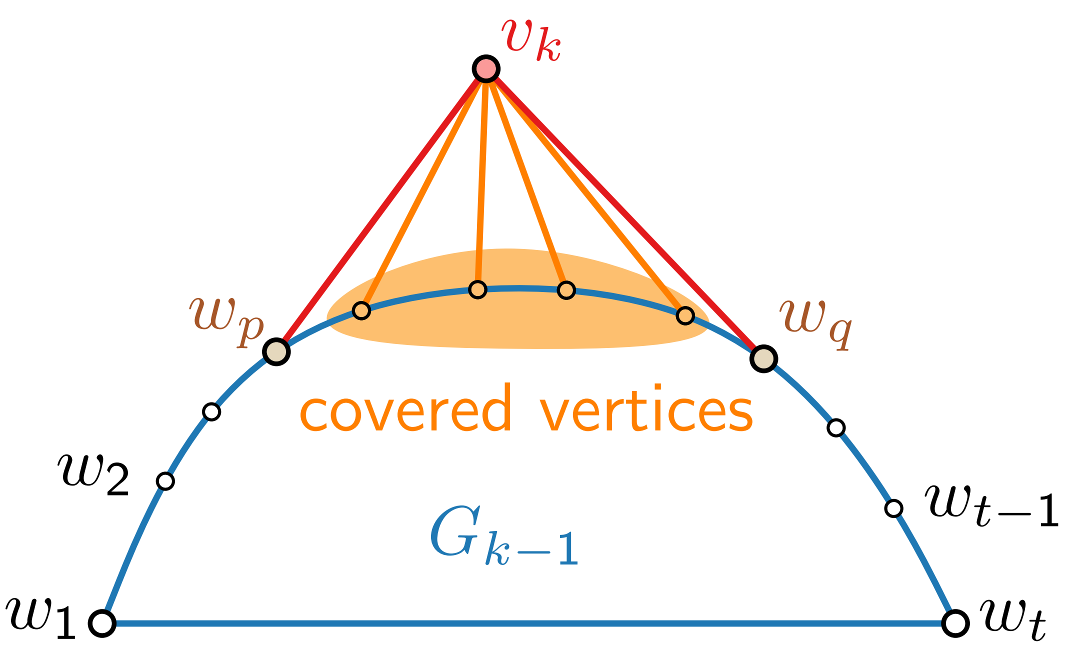
- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G



Shift Method – Planarity

Observations.

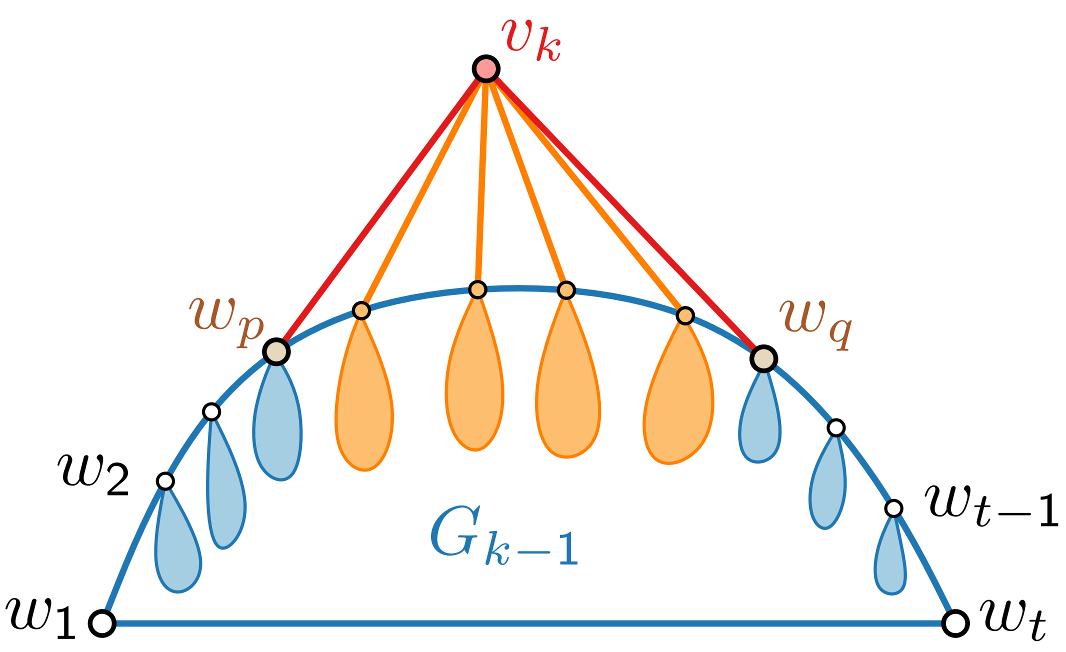
- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.



Shift Method – Planarity

Observations.

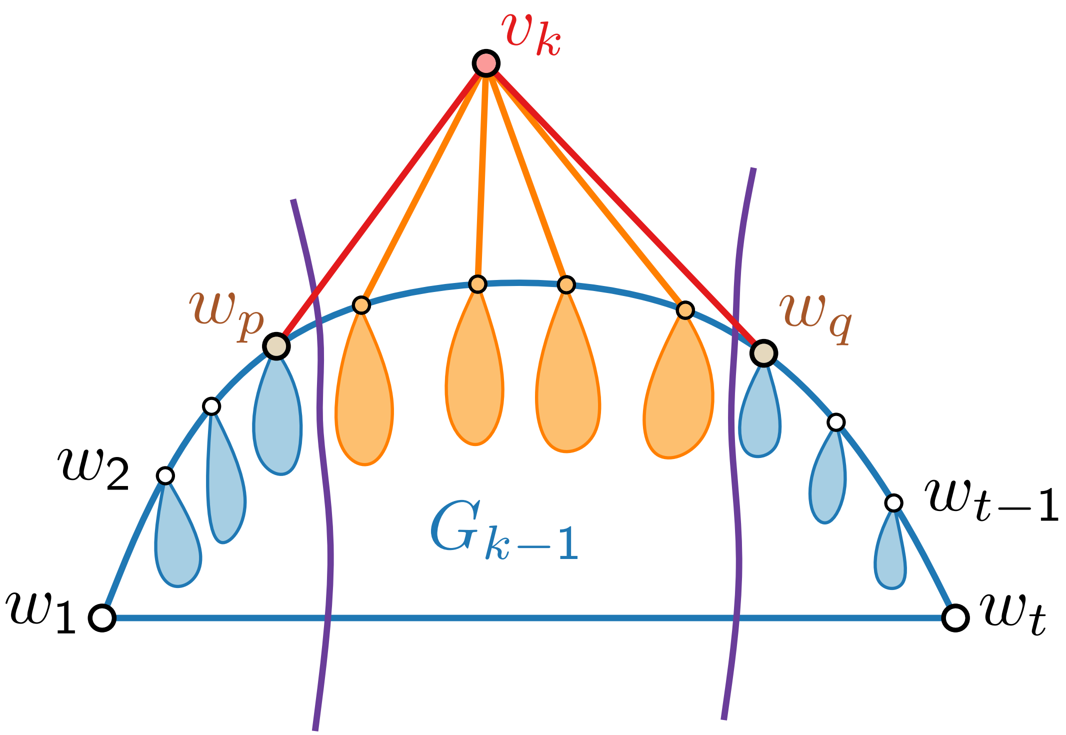
- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.



Shift Method – Planarity

Observations.

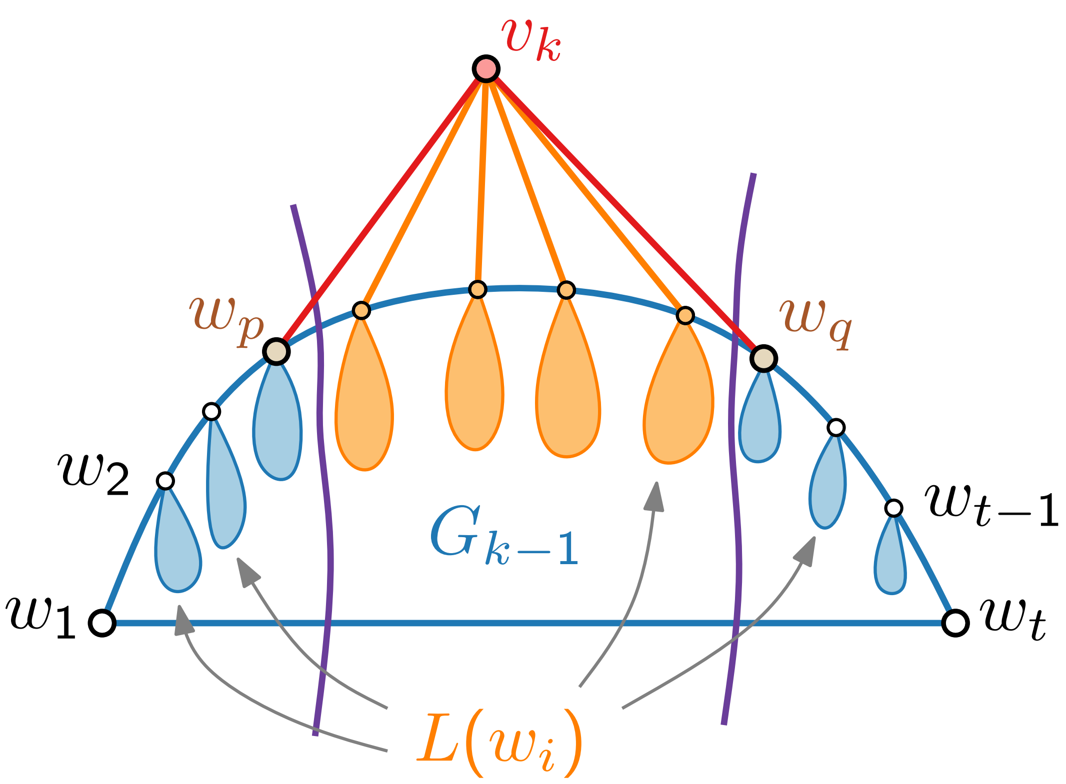
- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.



Shift Method – Planarity

Observations.

- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.

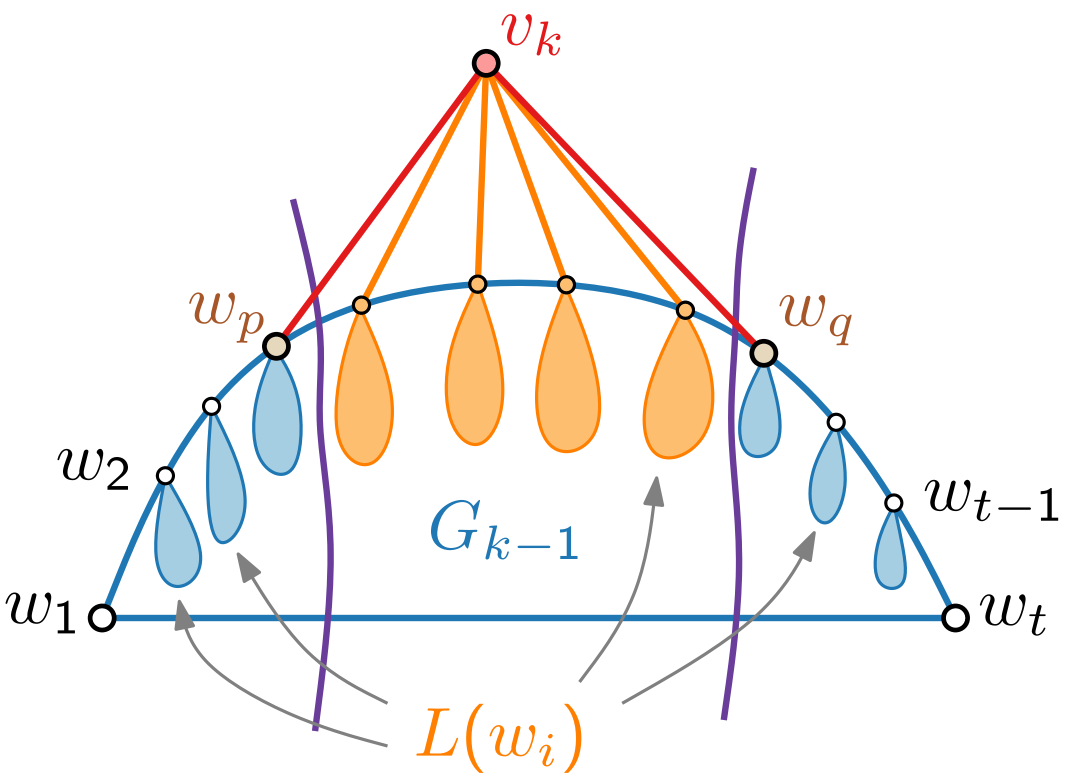


Shift Method – Planarity

Observations.

- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.

Lemma.
Let $0 \leq \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, s.t. $\delta_{p+1} - \delta_p \geq 1$, $\delta_q - \delta_{q-1} \geq 1$, $\delta_q - \delta_p \geq 2$ and even.

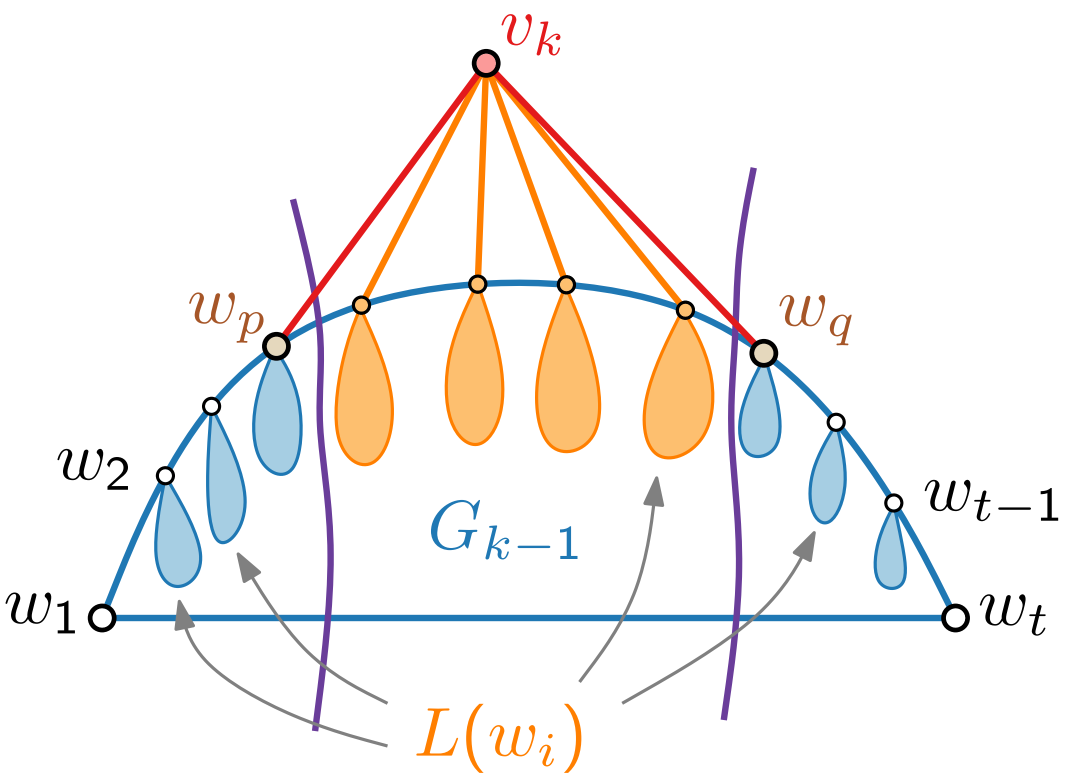


Shift Method – Planarity

Observations.

- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.

Lemma.
Let $0 \leq \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, s.t. $\delta_{p+1} - \delta_p \geq 1$, $\delta_q - \delta_{q-1} \geq 1$, $\delta_q - \delta_p \geq 2$ and even. If we shift $L(w_i)$ by δ_i to the right, then we get a planar straight-line drawing.

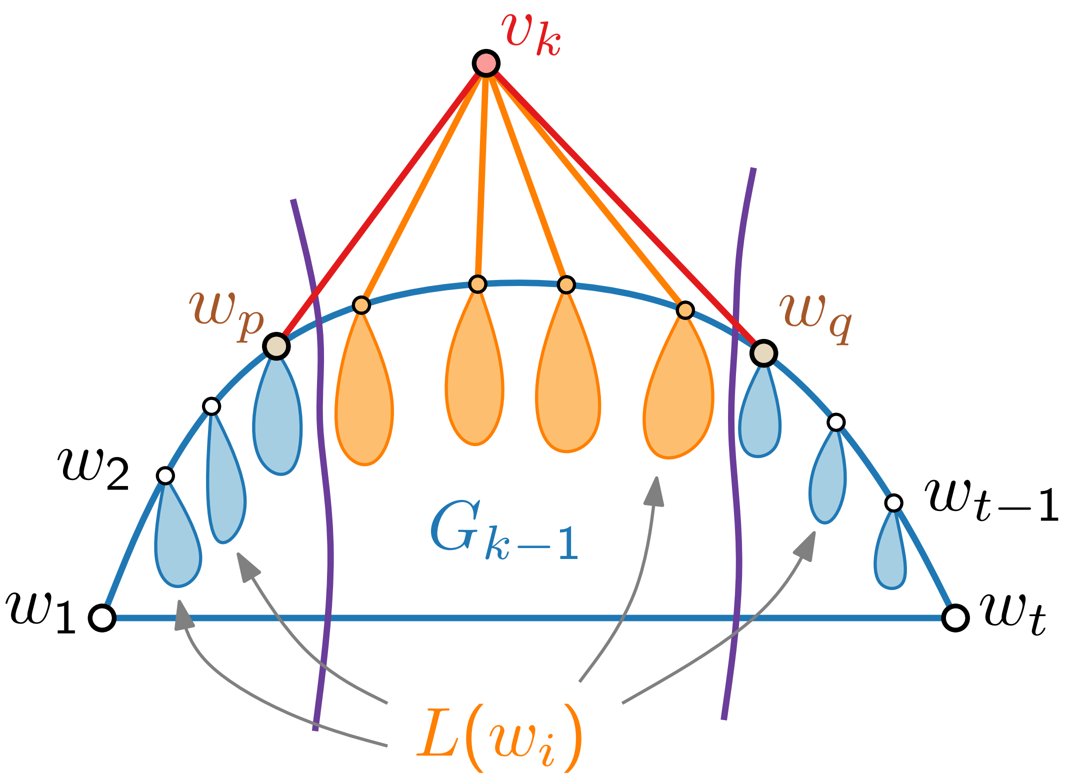


Shift Method – Planarity

Observations.

- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.

Lemma.
Let $0 \leq \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, s.t. $\delta_{p+1} - \delta_p \geq 1$, $\delta_q - \delta_{q-1} \geq 1$, $\delta_q - \delta_p \geq 2$ and even. If we shift $L(w_i)$ by δ_i to the right, then we get a planar straight-line drawing.



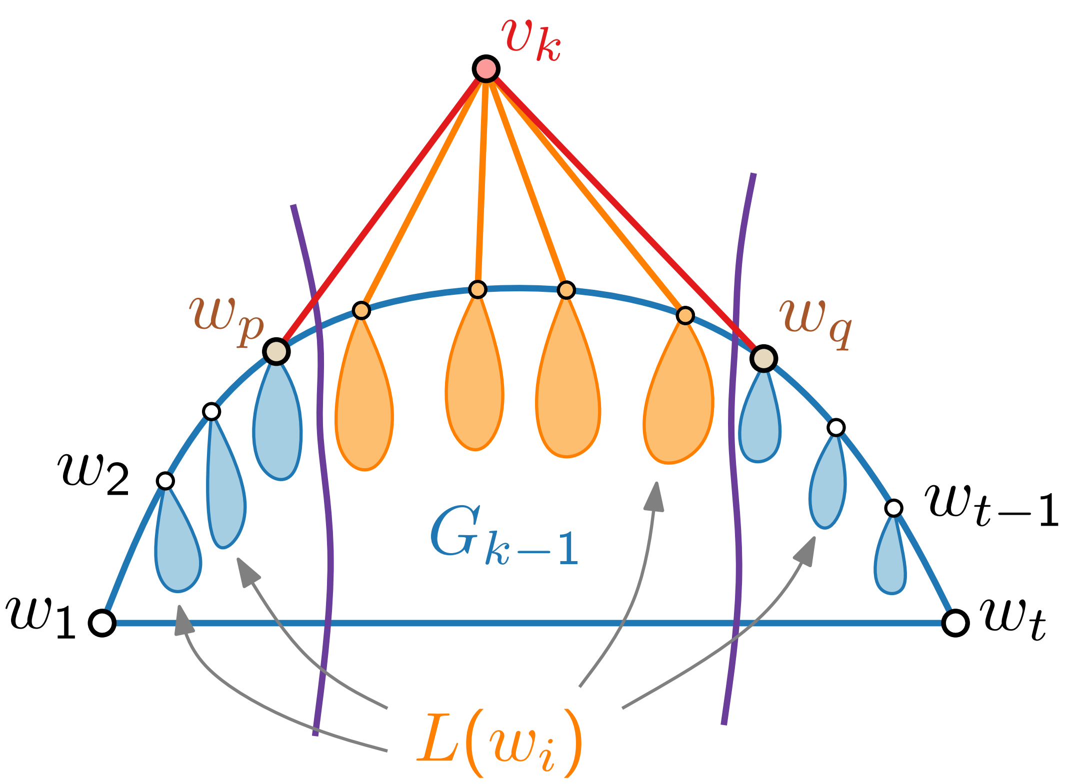
Proof by induction:
If G_{k-1} is drawn planar and straight-line, then so is G_k.

Shift Method – Planarity

Observations.

- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.

Lemma.
Let $0 \leq \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, s.t. $\delta_{p+1} - \delta_p \geq 1$, $\delta_q - \delta_{q-1} \geq 1$, $\delta_q - \delta_p \geq 2$ and even. If we shift $L(w_i)$ by δ_i to the right, then we get a planar straight-line drawing.



Proof by induction:

If G_{k-1} is drawn planar and straight-line, then so is G_k.

Ideas:

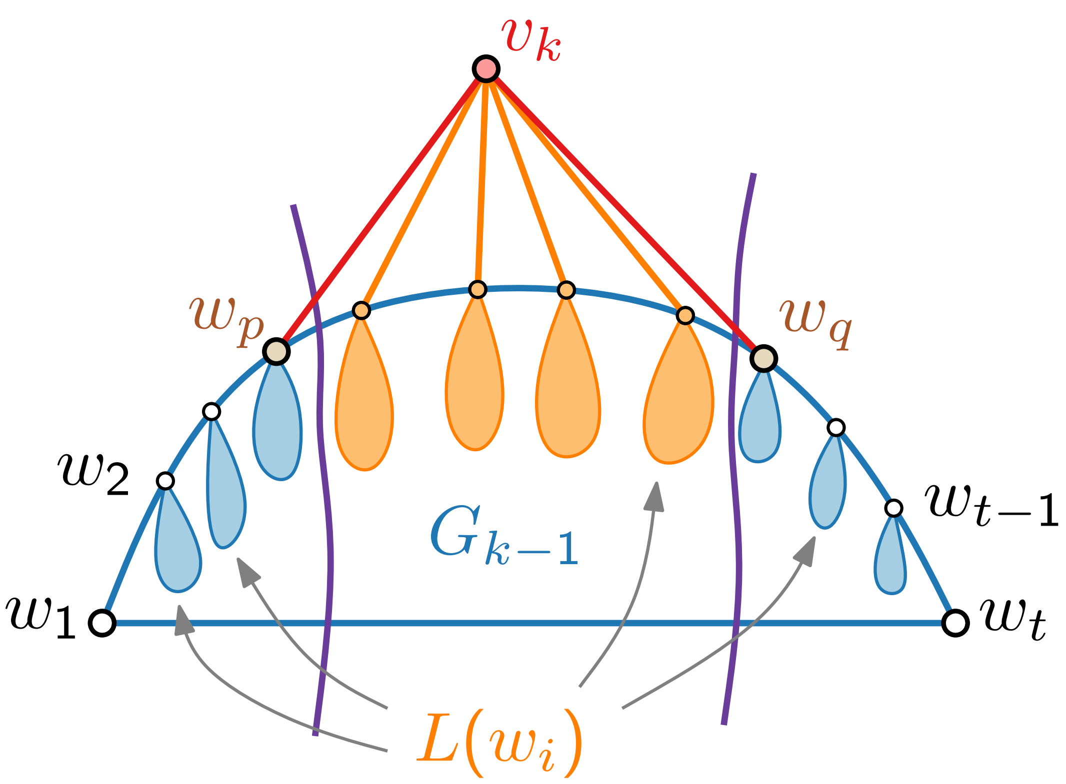
- New edges don't intersect other edges ($\rightarrow$ invariants).
- Edges within each $L(w_i)$ do not change.
- Other edges lie within triangles that only become flatter without causing new intersections.

Shift Method – Planarity

Observations.

- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.

Lemma.
Let $0 \leq \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, s.t. $\delta_{p+1} - \delta_p \geq 1$, $\delta_q - \delta_{q-1} \geq 1$, $\delta_q - \delta_p \geq 2$ and even. If we shift $L(w_i)$ by δ_i to the right, then we get a planar straight-line drawing.

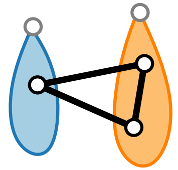


Proof by induction:

If G_{k-1} is drawn planar and straight-line, then so is G_k.

Ideas:

- New edges don't intersect other edges ($\rightarrow$ invariants).
- Edges within each $L(w_i)$ do not change.
- Other edges lie within triangles that only become flatter without causing new intersections.

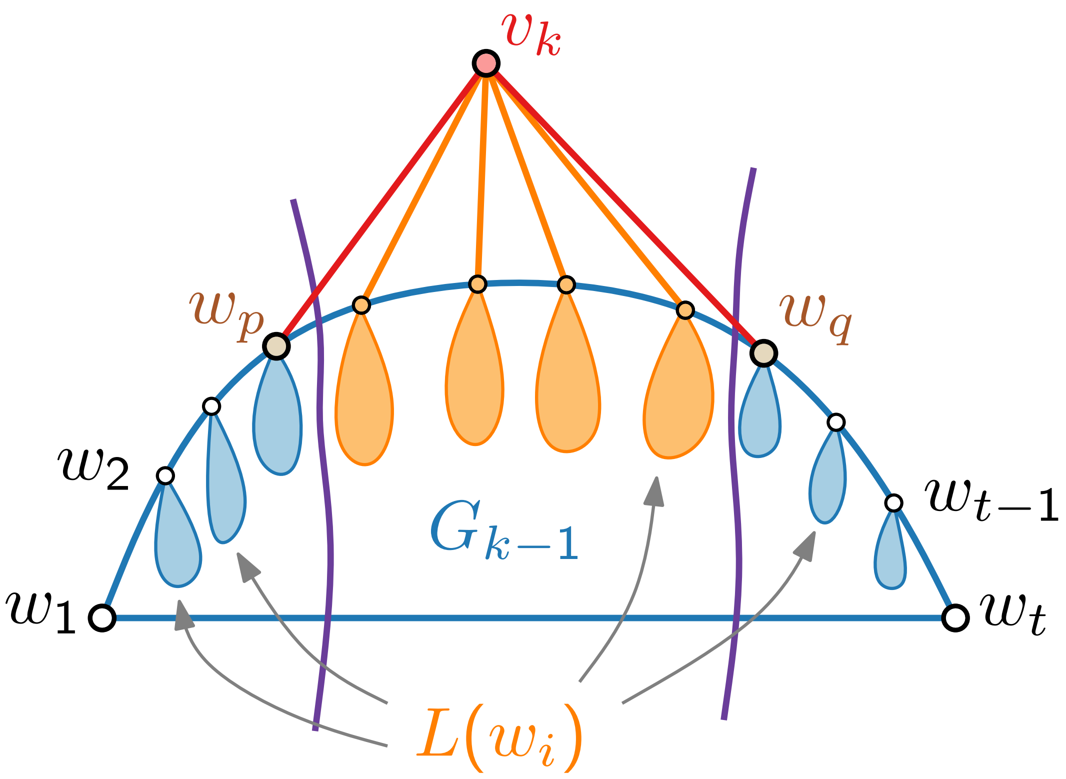


Shift Method – Planarity

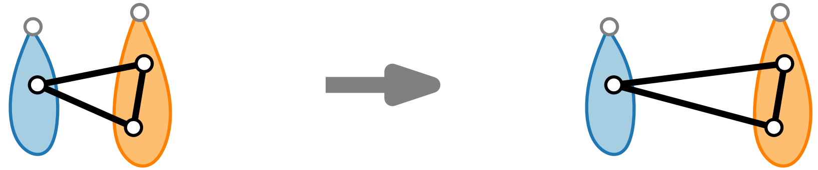
Observations.

- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.

Lemma.
Let $0 \leq \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, s.t. $\delta_{p+1} - \delta_p \geq 1$, $\delta_q - \delta_{q-1} \geq 1$, $\delta_q - \delta_p \geq 2$ and even. If we shift $L(w_i)$ by δ_i to the right, then we get a planar straight-line drawing.



Proof by induction:
If G_{k-1} is drawn planar and straight-line, then so is G_k.
Ideas:
- New edges don't intersect other edges ($\rightarrow$ invariants).
- Edges within each $L(w_i)$ do not change.
- Other edges lie within triangles that only become flatter without causing new intersections.



Shift Method – Pseudocode

canonical order of $V(G)$

ShiftMethod($G, (v_1, v_2, \ldots, v_n)$)

for $k = 1$ to 3 **do**

for $k = 4$ to n **do**

Shift Method – Pseudocode

canonical order of $V(G)$

ShiftMethod$(G, (v_1, v_2, \ldots, v_n))$

for $k = 1$ to 3 **do**

  $L(v_k) \leftarrow \{v_k\}$

for $k = 4$ to n **do**

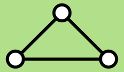
Shift Method – Pseudocode

canonical order of $V(G)$

ShiftMethod(G, $(v_1, v_2, \ldots, v_n)$)

for $k = 1$ to 3 **do**
  $L(v_k) \leftarrow \{v_k\}$

$P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$



for $k = 4$ to n **do**

return $P(v_1), \ldots, P(v_n)$

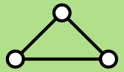
Shift Method – Pseudocode

canonical order of $V(G)$

ShiftMethod(G, $(v_1, v_2, \dots, v_n)$)

for $k = 1$ to 3 **do**

  $L(v_k) \leftarrow \{v_k\}$

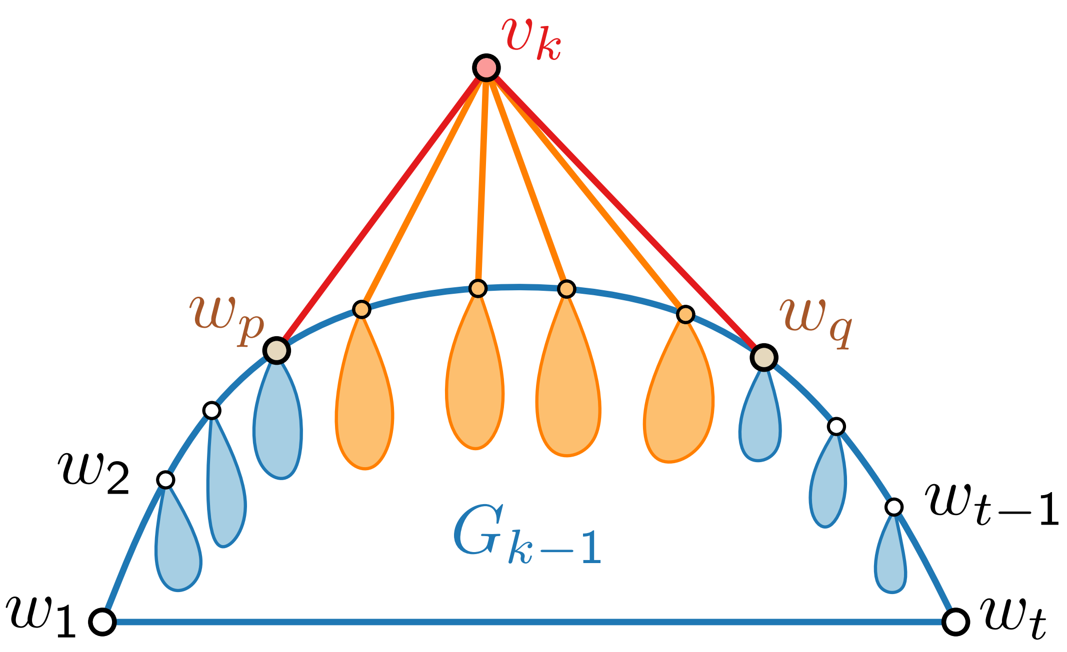
$P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$ 

for $k = 4$ to n **do**

  Let ∂G_{k-1} be $v_1 = w_1,\ w_2,\ \dots,\ w_{t-1},\ w_t = v_2$.

  Let $w_p, \dots, w_q$ be the neighbors of v_k.

return $P(v_1), \dots, P(v_n)$



Shift Method – Pseudocode

canonical order of $V(G)$

ShiftMethod(G, $(v_1, v_2, \ldots, v_n)$)

for $k = 1$ to 3 **do**

  $L(v_k) \leftarrow \{v_k\}$

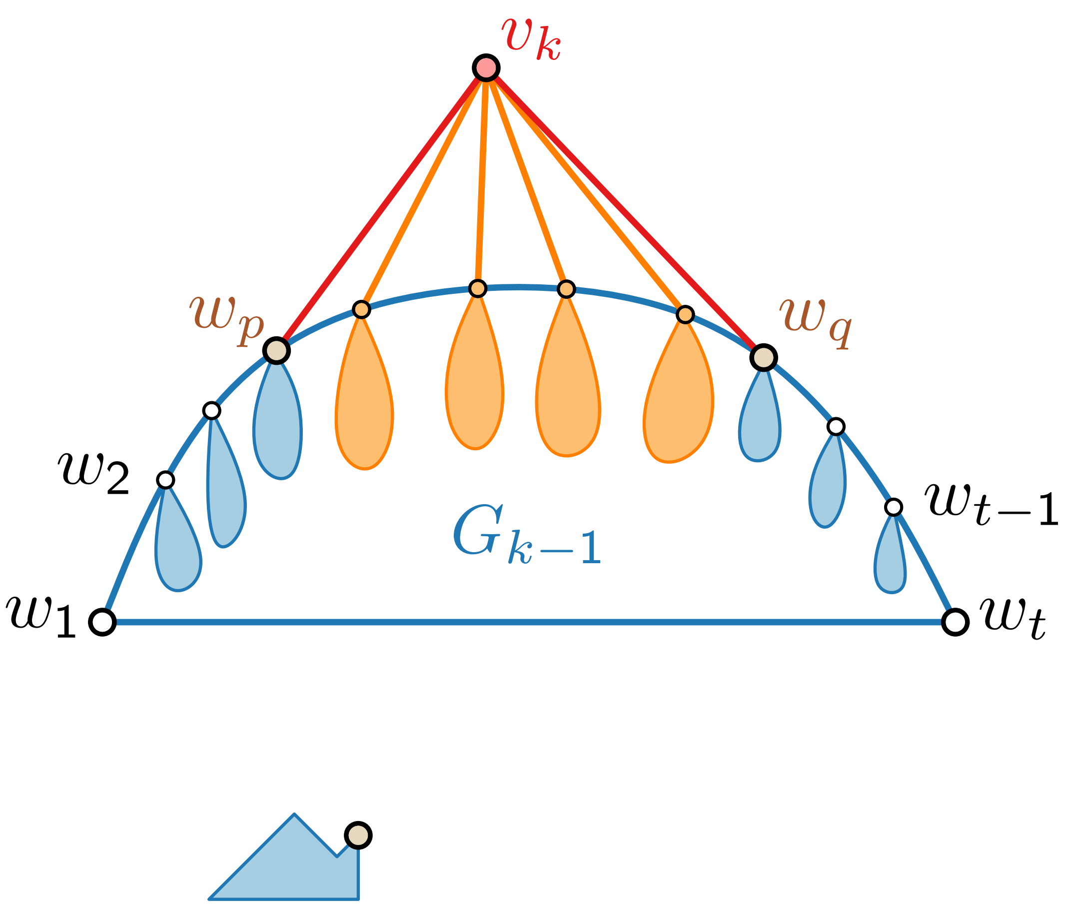
$P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$

for $k = 4$ to n **do**

  Let ∂G_{k-1} be $v_1 = w_1, w_2, \ldots, w_{t-1}, w_t = v_2$.

  Let $w_p, \ldots, w_q$ be the neighbors of v_k.

return $P(v_1), \ldots, P(v_n)$



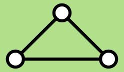
Shift Method – Pseudocode

canonical order of $V(G)$

ShiftMethod(G, $(v_1, v_2, \ldots, v_n)$)

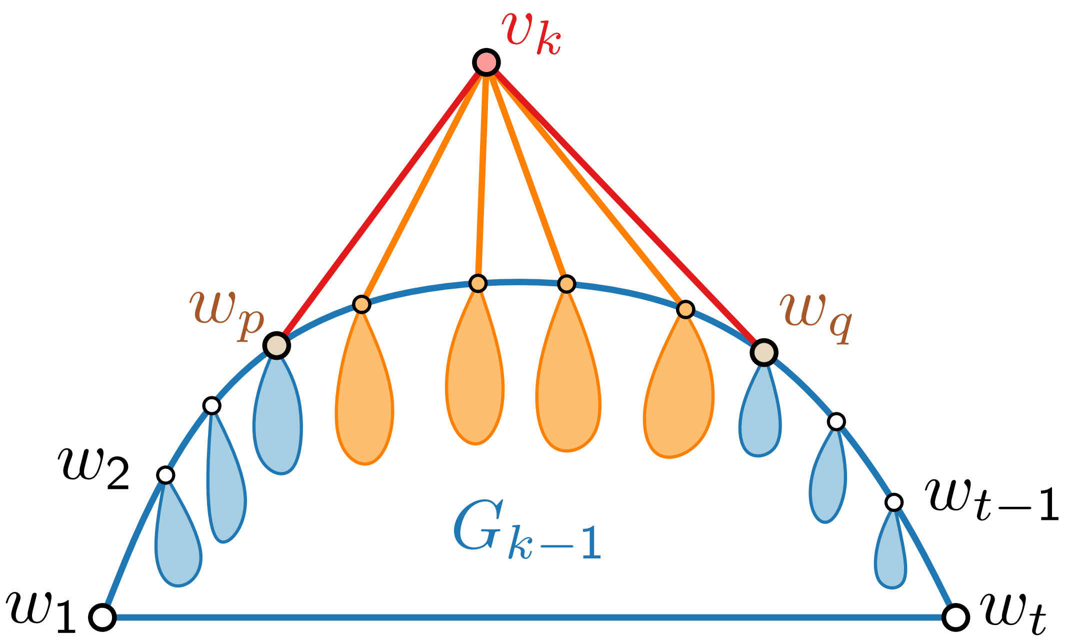
for $k = 1$ to 3 **do**
  $L(v_k) \leftarrow \{v_k\}$

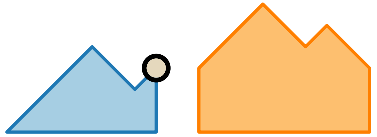
$P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$



for $k = 4$ to n **do**
  Let ∂G_{k-1} be $v_1 = w_1, w_2, \ldots, w_{t-1}, w_t = v_2$.
  Let $w_p, \ldots, w_q$ be the neighbors of v_k.
  foreach $v \in \bigcup_{i=p+1}^{q-1} L(w_i)$ **do**

return $P(v_1), \ldots, P(v_n)$



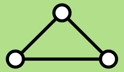


Shift Method – Pseudocode

canonical order of $V(G)$

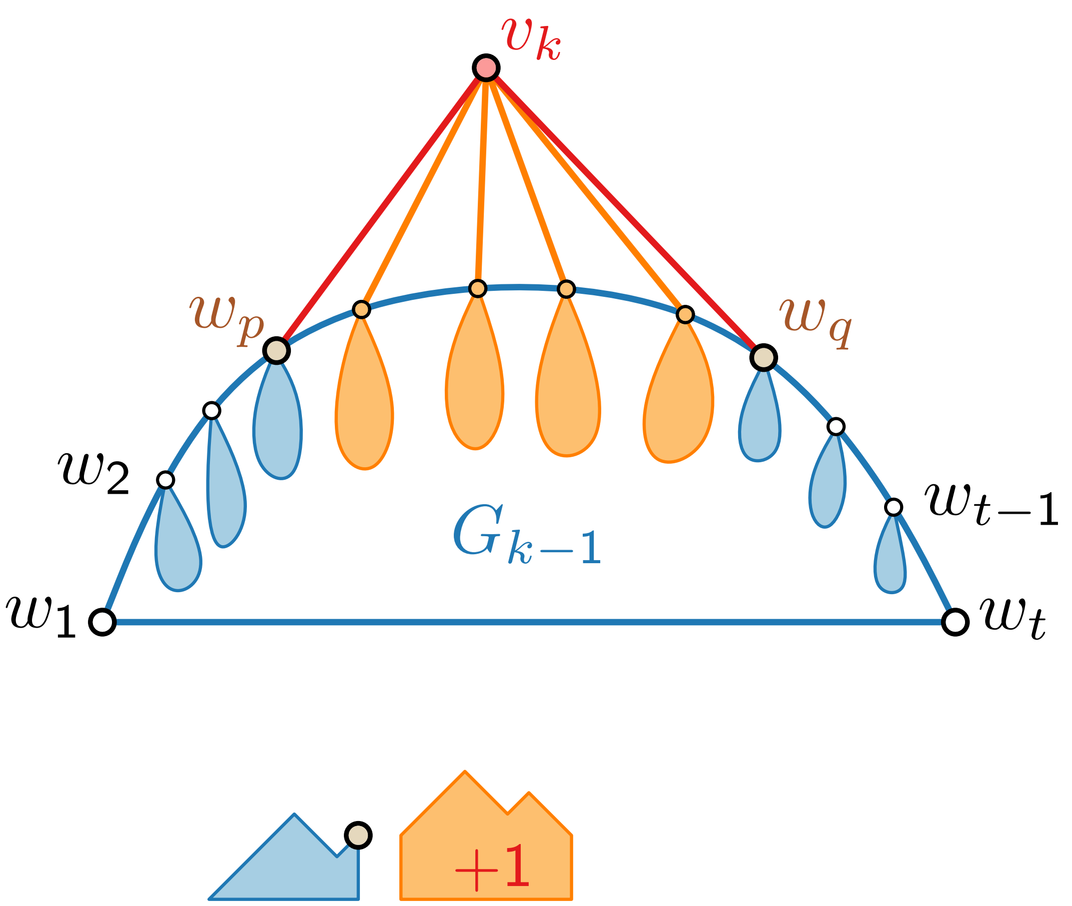
ShiftMethod(G, $(v_1, v_2, \ldots, v_n)$)

for $k = 1$ to 3 **do**
  $L(v_k) \leftarrow \{v_k\}$

$P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$ 

for $k = 4$ to n **do**
  Let ∂G_{k-1} be $v_1 = w_1, w_2, \ldots, w_{t-1}, w_t = v_2$.
  Let $w_p, \ldots, w_q$ be the neighbors of v_k.
  foreach $v \in \bigcup_{i=p+1}^{q-1} L(w_i)$ **do**
    $x(v) \leftarrow x(v) + 1$

return $P(v_1), \ldots, P(v_n)$



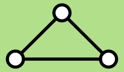
Shift Method – Pseudocode

canonical order of $V(G)$

ShiftMethod(G, $(v_1, v_2, \ldots, v_n)$)

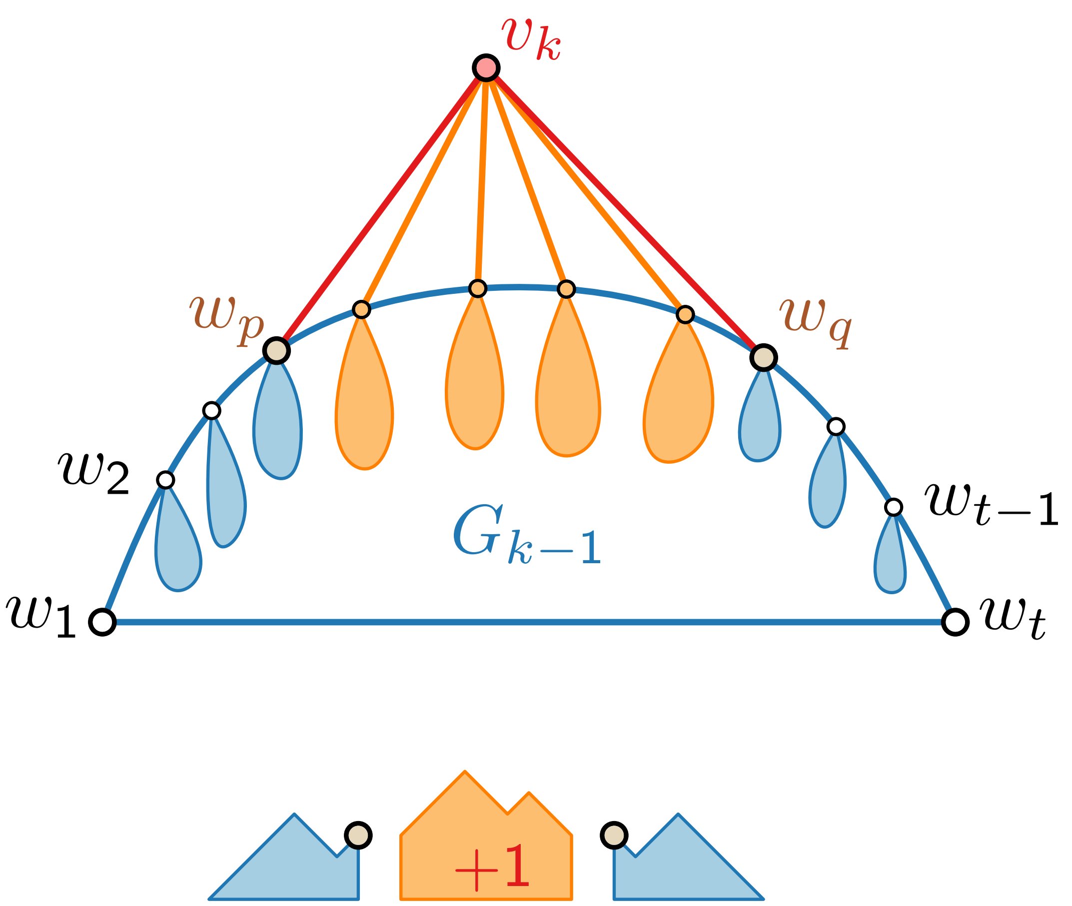
for $k = 1$ to 3 **do**
- $L(v_k) \leftarrow \{v_k\}$

$P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$



for $k = 4$ to n **do**
- Let ∂G_{k-1} be $v_1 = w_1, w_2, \ldots, w_{t-1}, w_t = v_2$.
- Let $w_p, \ldots, w_q$ be the neighbors of v_k.
- **foreach** $v \in \bigcup_{i=p+1}^{q-1} L(w_i)$ **do**
  - $x(v) \leftarrow x(v) + 1$
- **foreach** $v \in \bigcup_{i=q}^{t} L(w_i)$ **do**

return $P(v_1), \ldots, P(v_n)$

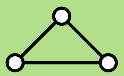


Shift Method – Pseudocode

canonical order of $V(G)$

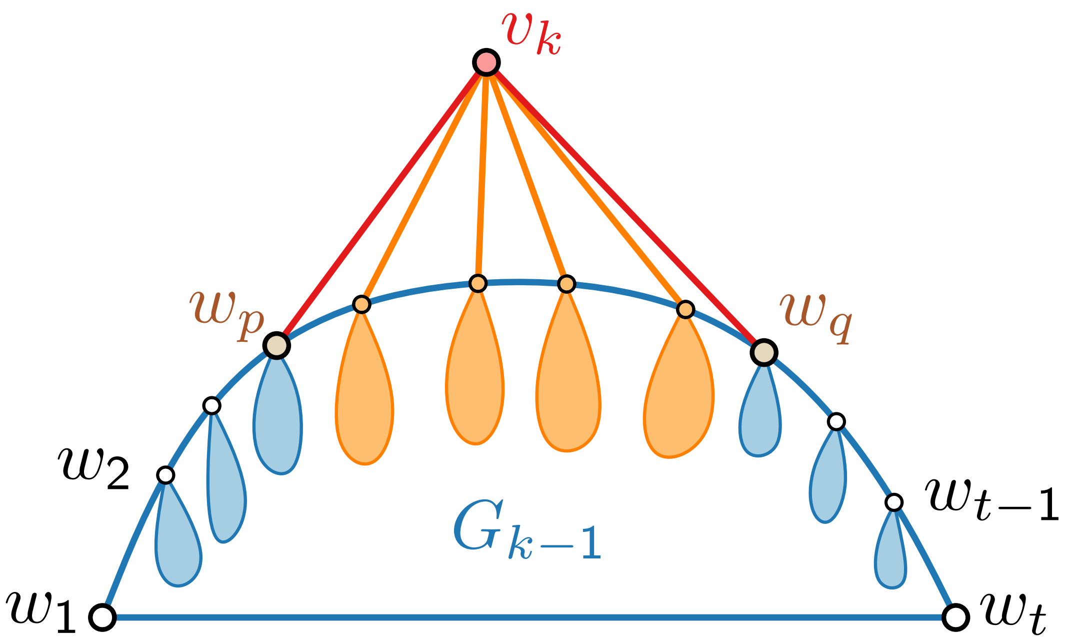
ShiftMethod(G, $(v_1, v_2, \ldots, v_n)$)

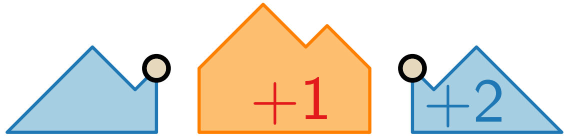
for $k = 1$ to 3 **do**
  $L(v_k) \leftarrow \{v_k\}$

$P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$ 

for $k = 4$ to n **do**
  Let ∂G_{k-1} be $v_1 = w_1, w_2, \ldots, w_{t-1}, w_t = v_2$.
  Let $w_p, \ldots, w_q$ be the neighbors of v_k.
  foreach $v \in \bigcup_{i=p+1}^{q-1} L(w_i)$ **do**
    $x(v) \leftarrow x(v) + 1$
  foreach $v \in \bigcup_{i=q}^{t} L(w_i)$ **do**
    $x(v) \leftarrow x(v) + 2$

return $P(v_1), \ldots, P(v_n)$





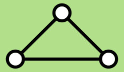
Shift Method – Pseudocode

canonical order of $V(G)$

ShiftMethod(G, $(v_1, v_2, \ldots, v_n)$)

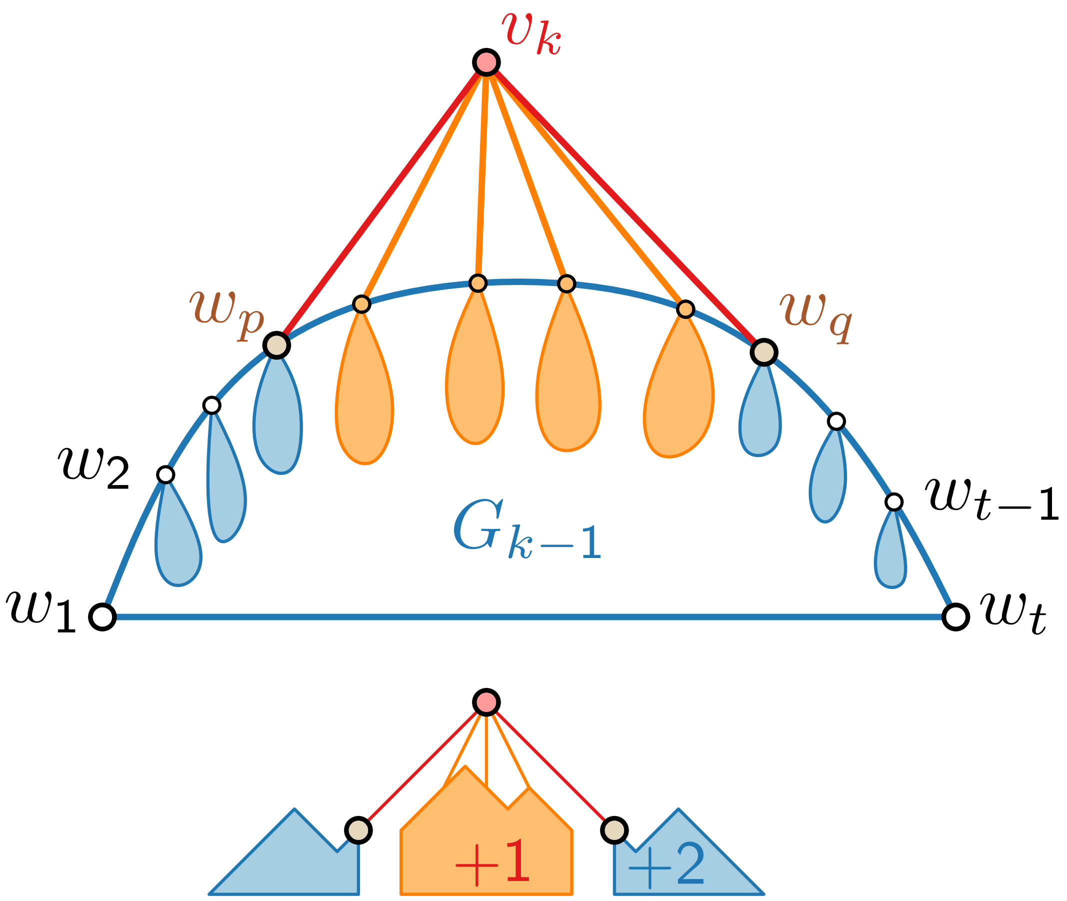
for $k = 1$ to 3 **do**
  $L(v_k) \leftarrow \{v_k\}$

$P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$



for $k = 4$ to n **do**
  Let ∂G_{k-1} be $v_1 = w_1, w_2, \ldots, w_{t-1}, w_t = v_2$.
  Let $w_p, \ldots, w_q$ be the neighbors of v_k.
  foreach $v \in \bigcup_{i=p+1}^{q-1} L(w_i)$ **do**
    $x(v) \leftarrow x(v) + 1$
  foreach $v \in \bigcup_{i=q}^{t} L(w_i)$ **do**
    $x(v) \leftarrow x(v) + 2$
  $P(v_k) \leftarrow$ intersection of slope-± 1 diagonals through $P(w_p)$ and $P(w_q)$

return $P(v_1), \ldots, P(v_n)$

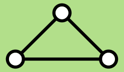


Shift Method – Pseudocode

canonical order of $V(G)$

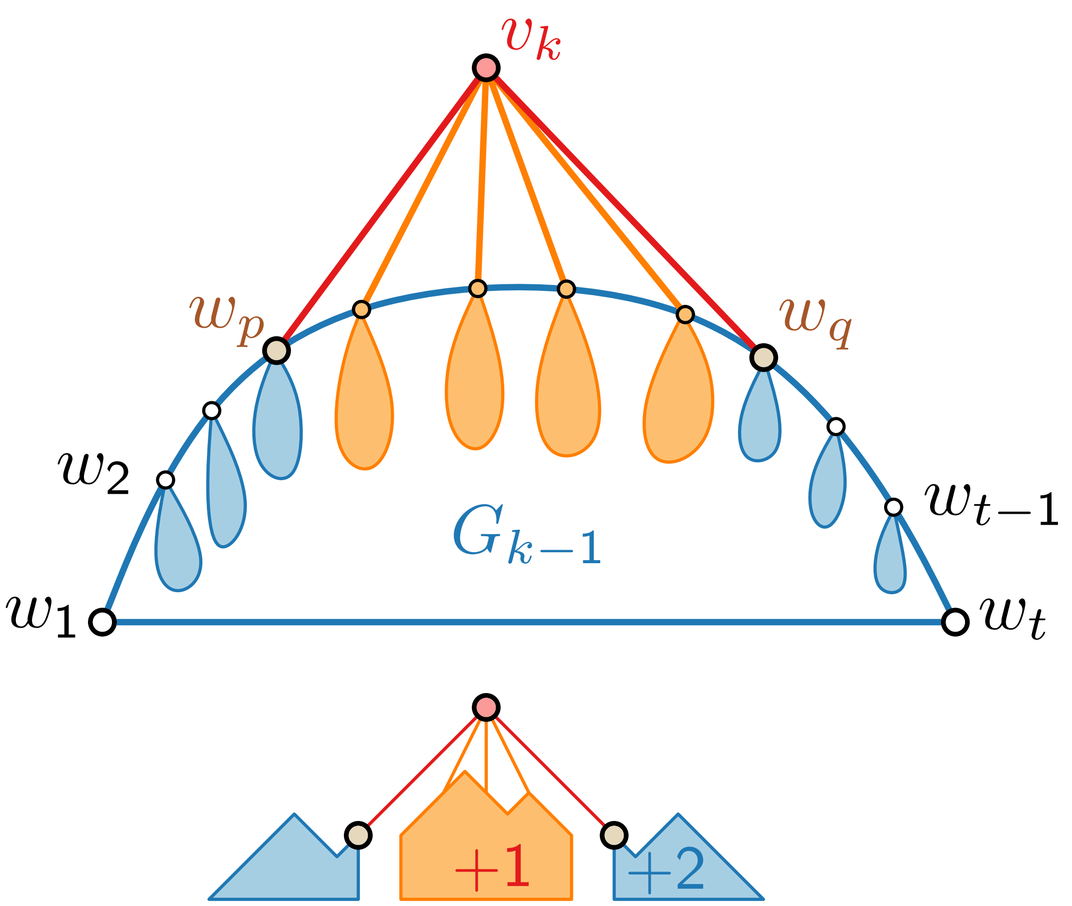
ShiftMethod$(G, (v_1, v_2, \ldots, v_n))$

for $k = 1$ to 3 **do**
- $L(v_k) \leftarrow \{v_k\}$

$P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$ 

for $k = 4$ to n **do**
- Let ∂G_{k-1} be $v_1 = w_1, w_2, \ldots, w_{t-1}, w_t = v_2$.
- Let $w_p, \ldots, w_q$ be the neighbors of v_k.
- **foreach** $v \in \bigcup_{i=p+1}^{q-1} L(w_i)$ **do**
  - $x(v) \leftarrow x(v) + 1$
- **foreach** $v \in \bigcup_{i=q}^{t} L(w_i)$ **do**
  - $x(v) \leftarrow x(v) + 2$
- $P(v_k) \leftarrow$ intersection of slope-± 1 diagonals through $P(w_p)$ and $P(w_q)$
- $L(v_k) \leftarrow \bigcup_{i=p+1}^{q-1} L(w_i) \cup \{v_k\}$

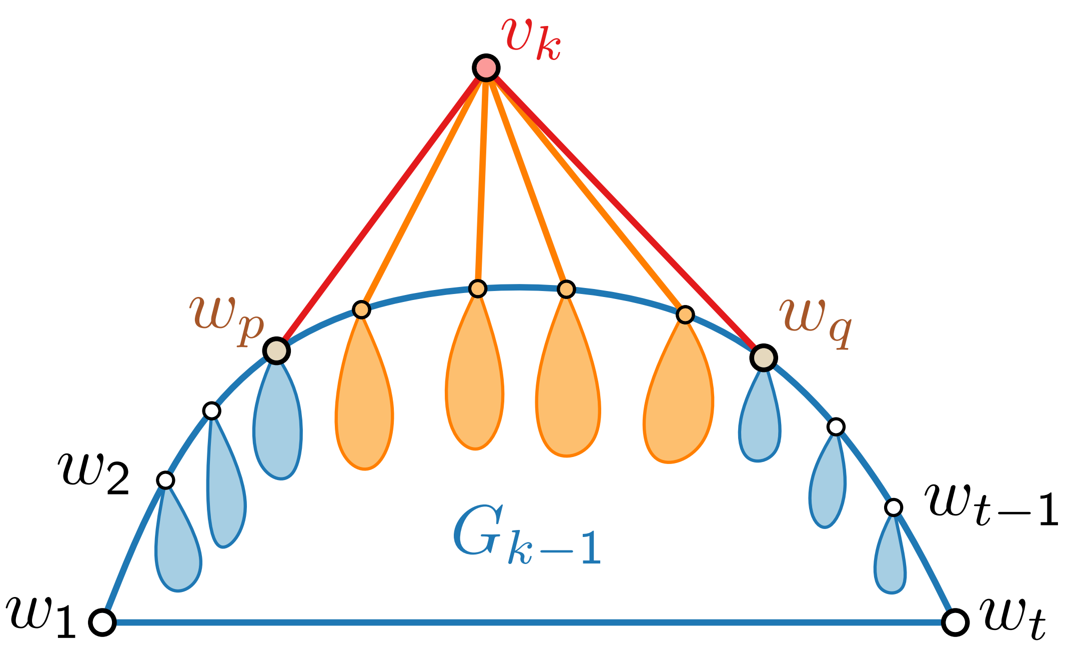
return $P(v_1), \ldots, P(v_n)$

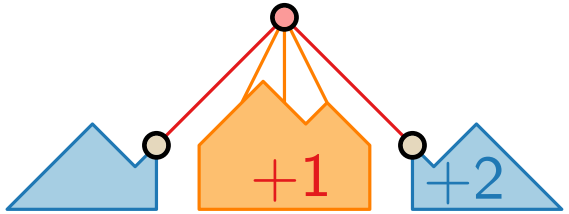


Shift Method – Pseudocode

canonical order of $V(G)$

ShiftMethod(G, $(v_1, v_2, \dots, v_n)$)
- **for** $k = 1$ to 3 **do**
  - $L(v_k) \leftarrow \{v_k\}$
- $P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$
- **for** $k = 4$ to n **do**
  - Let ∂G_{k-1} be $v_1 = w_1, w_2, \dots, w_{t-1}, w_t = v_2$.
  - Let $w_p, \dots, w_q$ be the neighbors of v_k.
  - **foreach** $v \in \bigcup_{i=p+1}^{q-1} L(w_i)$ **do**
    - $x(v) \leftarrow x(v) + 1$
  - **foreach** $v \in \bigcup_{i=q}^{t} L(w_i)$ **do**
    - $x(v) \leftarrow x(v) + 2$
  - $P(v_k) \leftarrow$ intersection of slope-± 1 diagonals through $P(w_p)$ and $P(w_q)$
  - $L(v_k) \leftarrow \bigcup_{i=p+1}^{q-1} L(w_i) \cup \{v_k\}$
- **return** $P(v_1), \dots, P(v_n)$





Running Time?

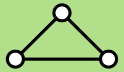
Shift Method – Pseudocode

canonical order of $V(G)$

ShiftMethod(G, $(v_1, v_2, \dots, v_n)$)

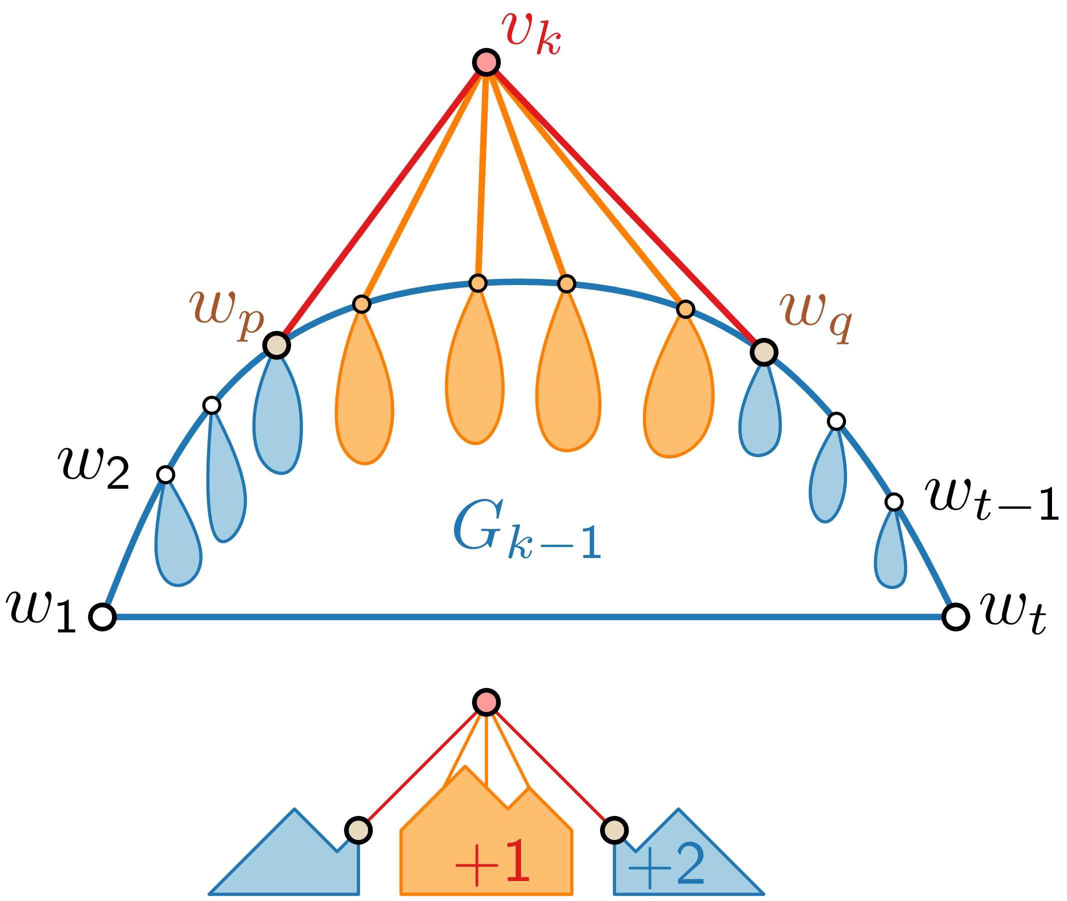
for $k = 1$ to 3 **do**
- $L(v_k) \leftarrow \{v_k\}$

$P(v_1) \leftarrow (0,0)$; $P(v_2) \leftarrow (2,0)$, $P(v_3) \leftarrow (1,1)$



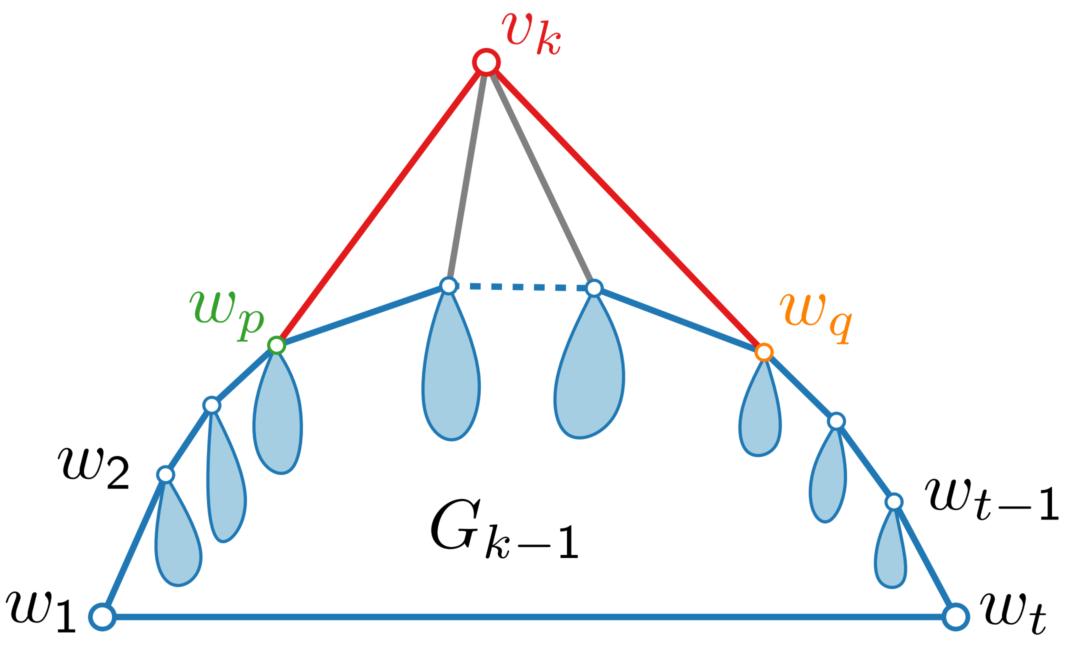
for $k = 4$ to n **do**
- Let ∂G_{k-1} be $v_1 = w_1, w_2, \dots, w_{t-1}, w_t = v_2$.
- Let $w_p, \dots, w_q$ be the neighbors of v_k.
- **foreach** $v \in \bigcup_{i=p+1}^{q-1} L(w_i)$ **do** // *$\mathcal{O}(n^2)$ in total*
  - $x(v) \leftarrow x(v) + 1$
- **foreach** $v \in \bigcup_{i=q}^{t} L(w_i)$ **do** // *$\mathcal{O}(n^2)$ in total*
  - $x(v) \leftarrow x(v) + 2$
- $P(v_k) \leftarrow$ intersection of slope-± 1 diagonals through $P(w_p)$ and $P(w_q)$
- $L(v_k) \leftarrow \bigcup_{i=p+1}^{q-1} L(w_i) \cup \{v_k\}$

return $P(v_1), \dots, P(v_n)$



Running Time?

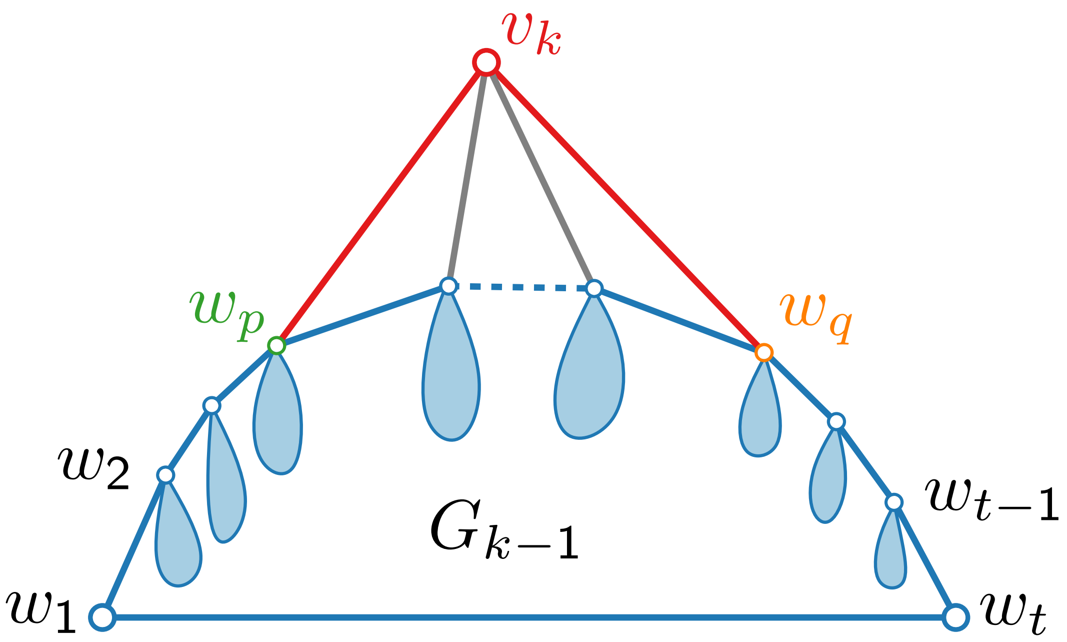
Shift Method – Linear-Time Implementation



Shift Method – Linear-Time Implementation

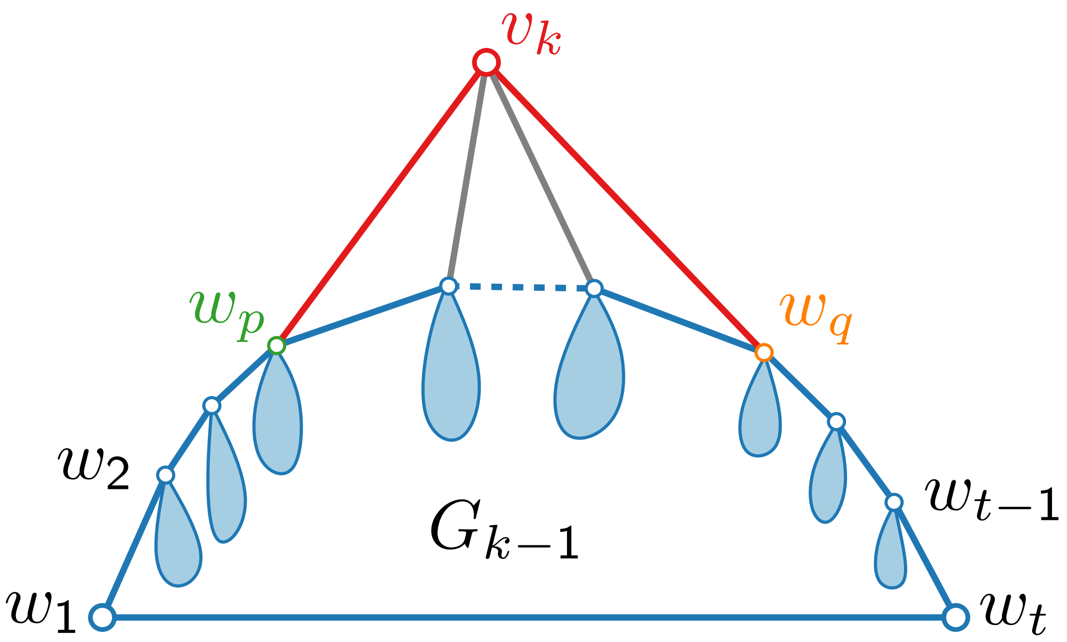
Idea 1.

To compute $x(v_k)$ and $y(v_k)$,
we need only $y(w_p)$, $y(w_q)$, and $x(w_q) - x(w_p)$



Shift Method – Linear-Time Implementation

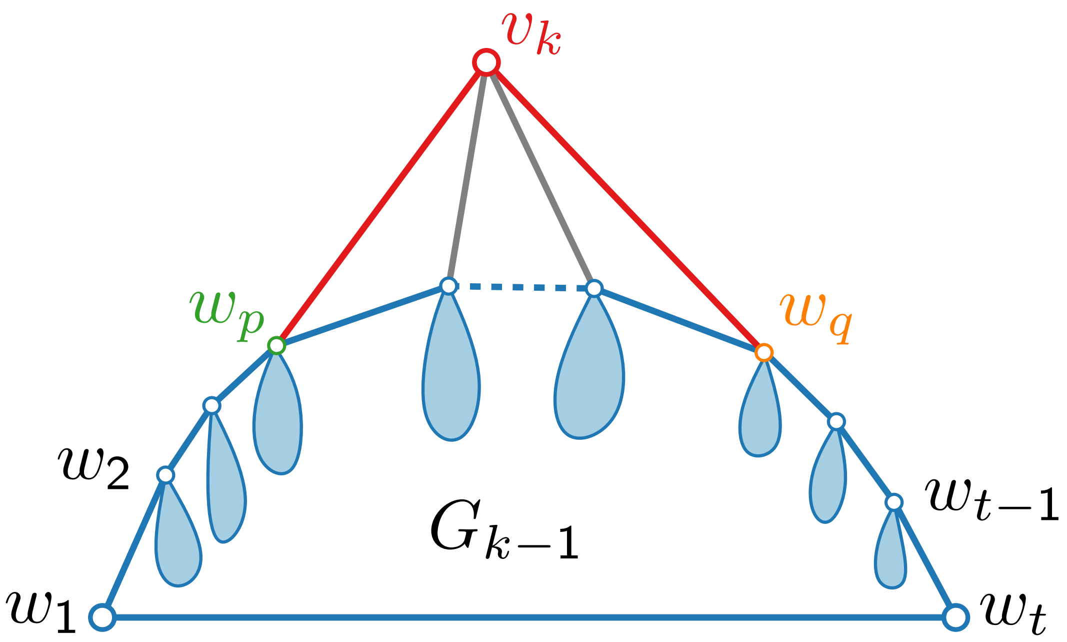
Idea 1.
To compute $x(v_k)$ and $y(v_k)$,
we need only $y(w_p)$, $y(w_q)$, and $x(w_q) - x(w_p)$



(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

Idea 1.
To compute $x(v_k)$ and $y(v_k)$,
we need only $y(w_p)$, $y(w_q)$, and $x(w_q) - x(w_p)$



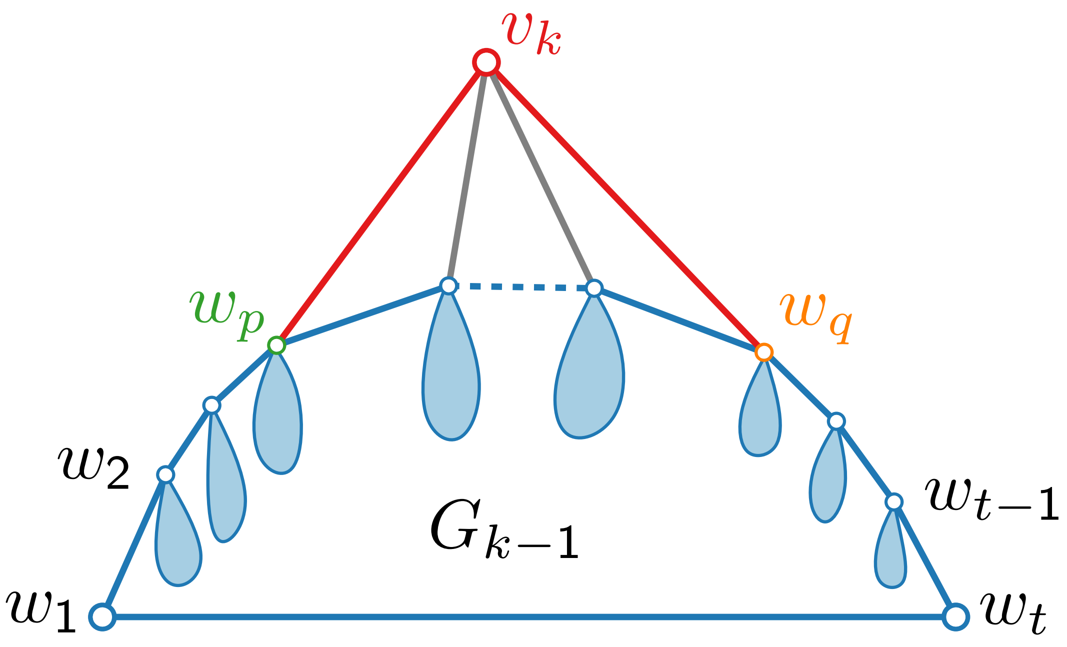
$$(1)\quad x(v_k) = \tfrac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$$

$$(2)\quad y(v_k) = \tfrac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$$

Shift Method – Linear-Time Implementation

Idea 1.
To compute $x(v_k)$ and $y(v_k)$,
we need only $y(w_p)$, $y(w_q)$, and $x(w_q) - x(w_p)$

Idea 2.
Instead of storing explicit x-coordinates,
we store, for each vertex within a specific spanning tree,
the x-distance to its parent (v_1 is the root).



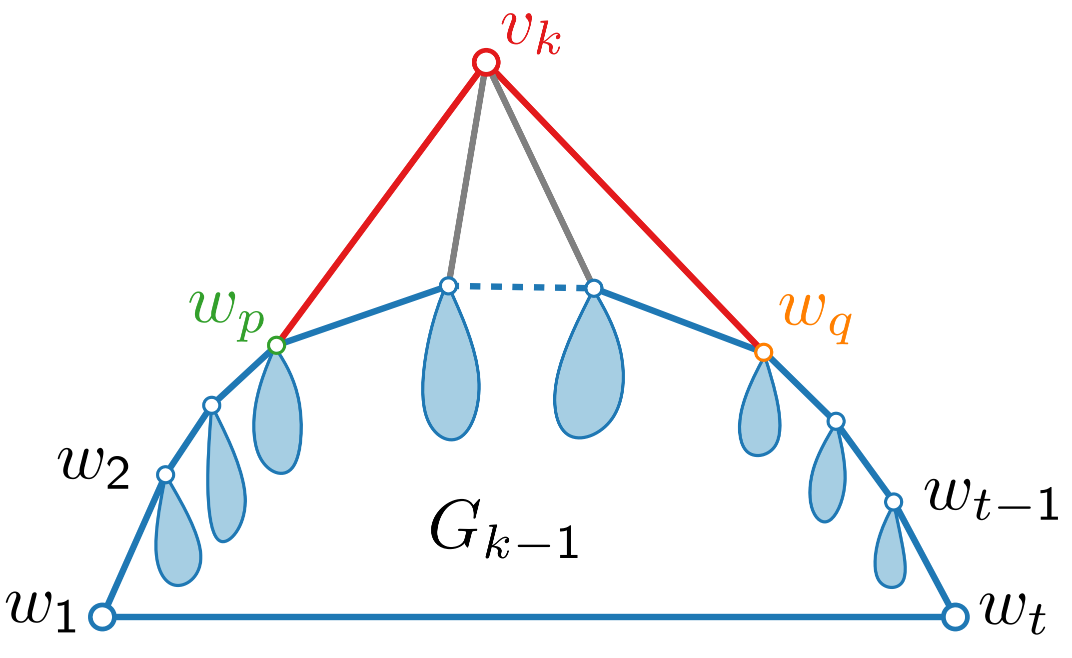
(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

Shift Method – Linear-Time Implementation

Idea 1.
To compute $x(v_k)$ and $y(v_k)$,
we need only $y(w_p)$, $y(w_q)$, and $x(w_q) - x(w_p)$

Idea 2.
Instead of storing explicit x-coordinates,
we store, for each vertex within a specific spanning tree,
the x-distance to its parent (v_1 is the root).



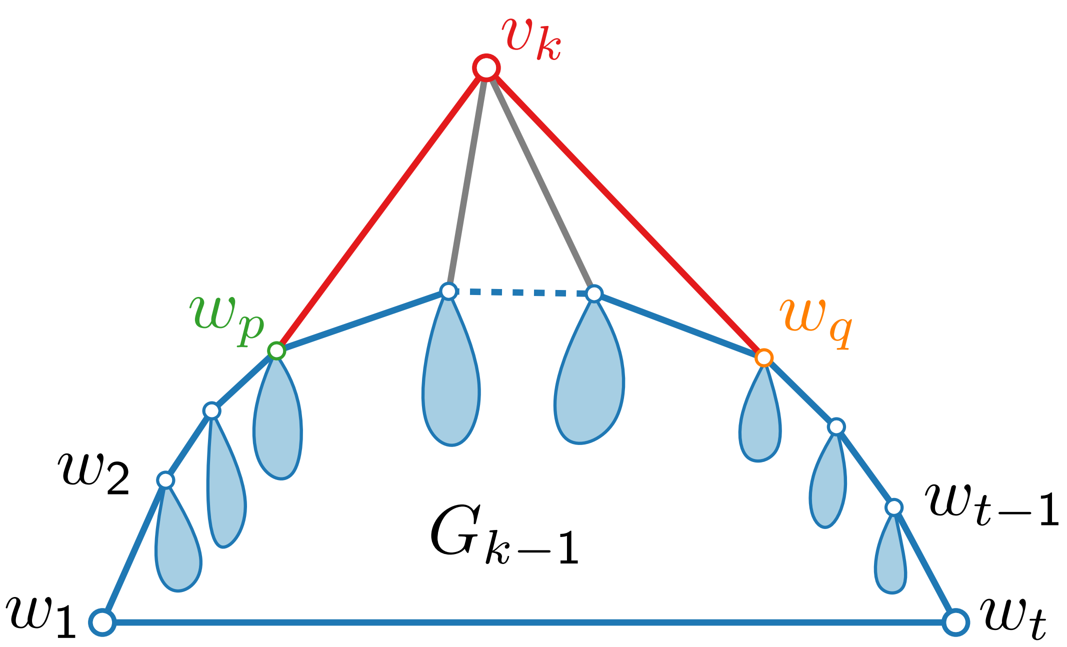
(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

Idea 1.
To compute $x(v_k)$ and $y(v_k)$,
we need only $y(w_p)$, $y(w_q)$, and $x(w_q) - x(w_p)$



Idea 2.
Instead of storing explicit x-coordinates,
we store, for each vertex within a specific spanning tree,
the x-distance to its parent (v_1 is the root).

After an x-distance is computed for each v_k,
use preorder traversal to compute all x-coordinates.

$$(1)\quad x(v_k) = \tfrac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$$

$$(2)\quad y(v_k) = \tfrac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$$

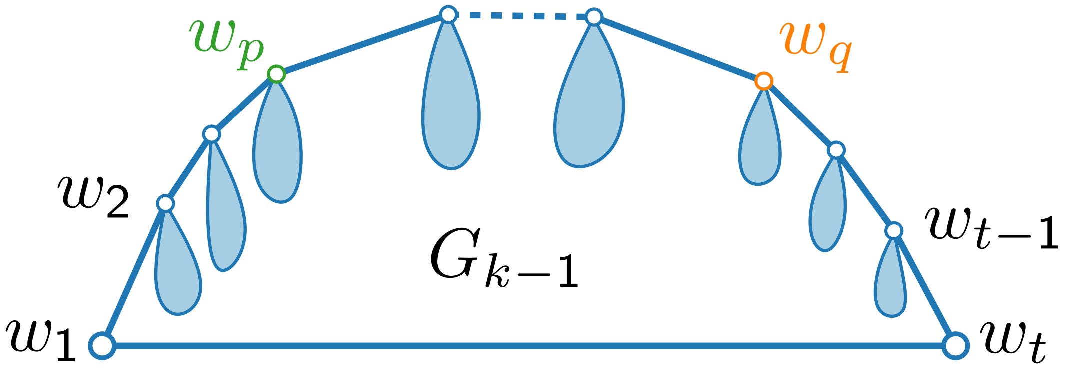
$$(3)\quad x(v_k) - x(w_p) = \tfrac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$$

Shift Method – Linear-Time Implementation

Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$



(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

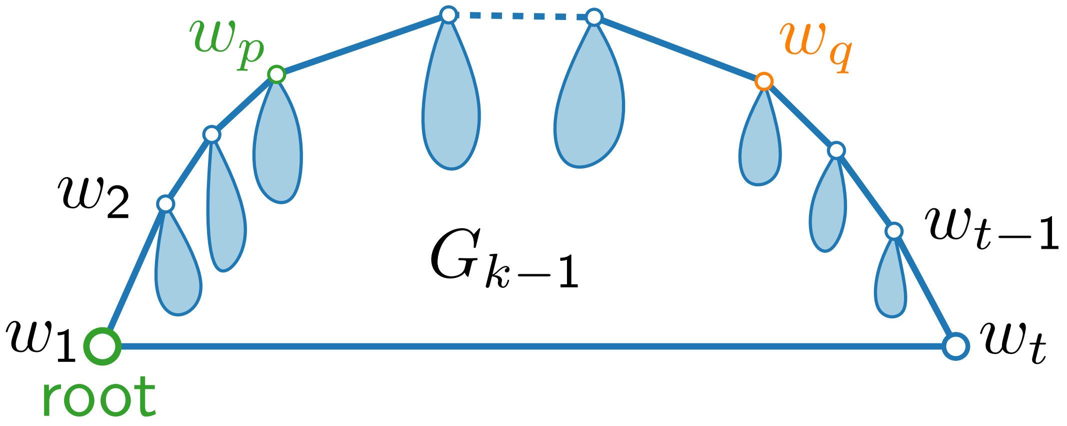
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$



(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

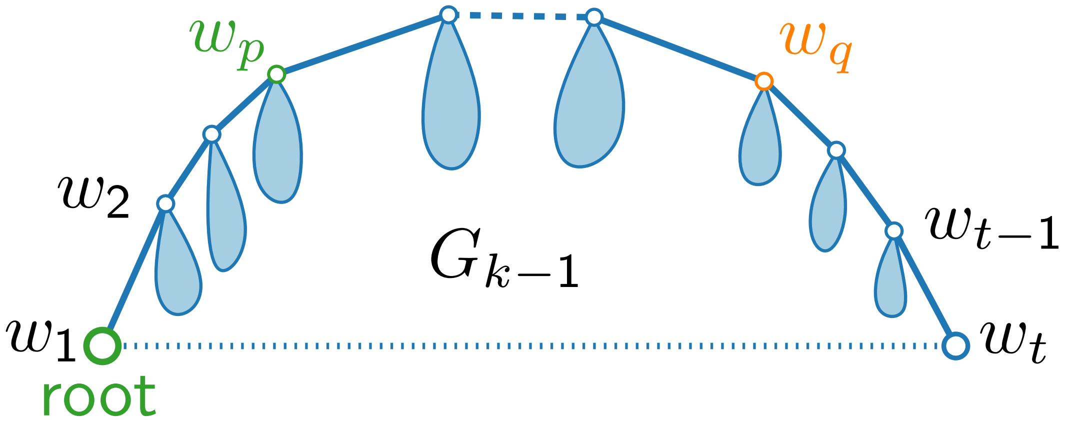
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$



(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

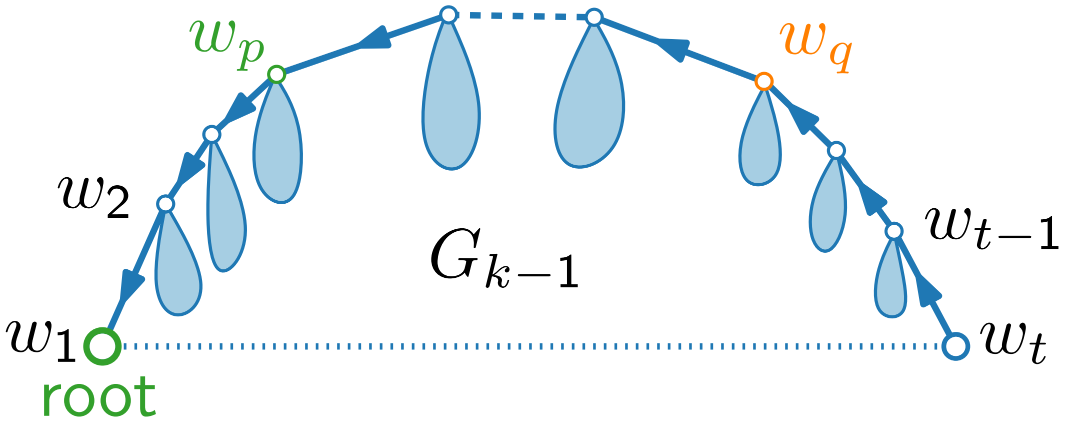
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$



(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

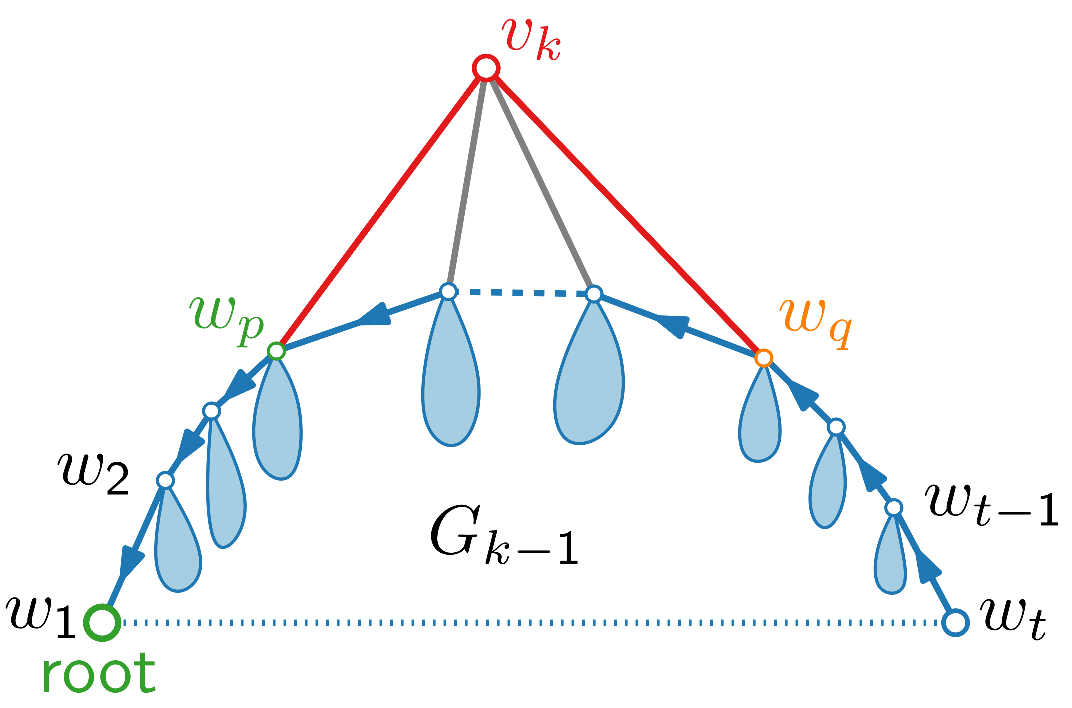
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$



(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

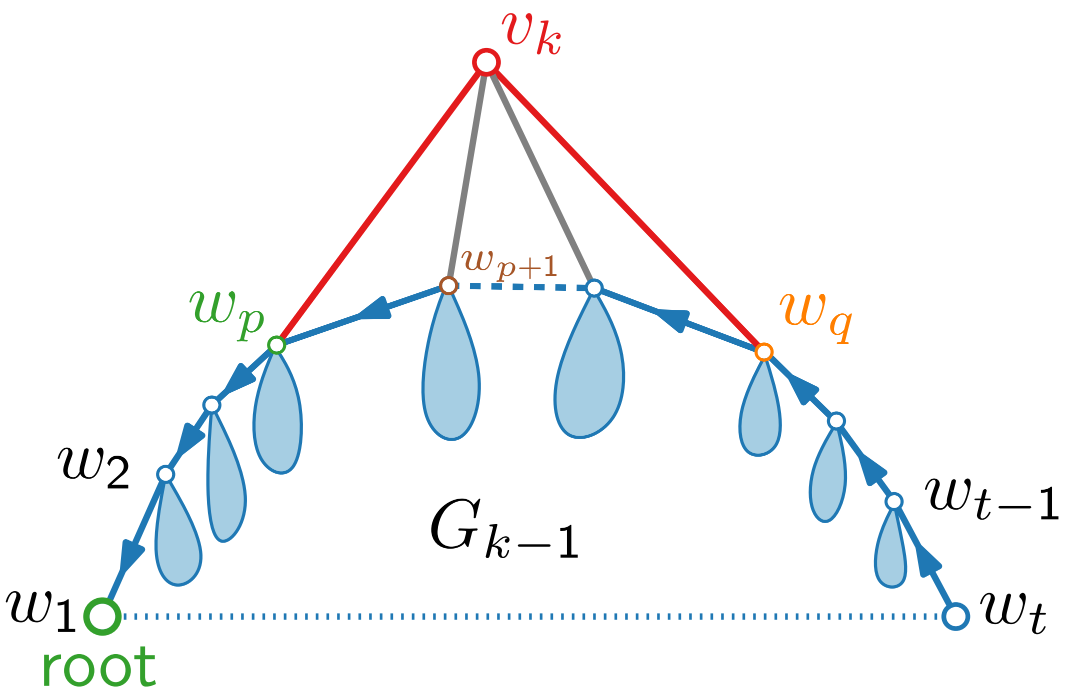
Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.

- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++



$$(1)\quad x(v_k) = \tfrac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$$

$$(2)\quad y(v_k) = \tfrac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$$

$$(3)\quad x(v_k) - x(w_p) = \tfrac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$$

Shift Method – Linear-Time Implementation

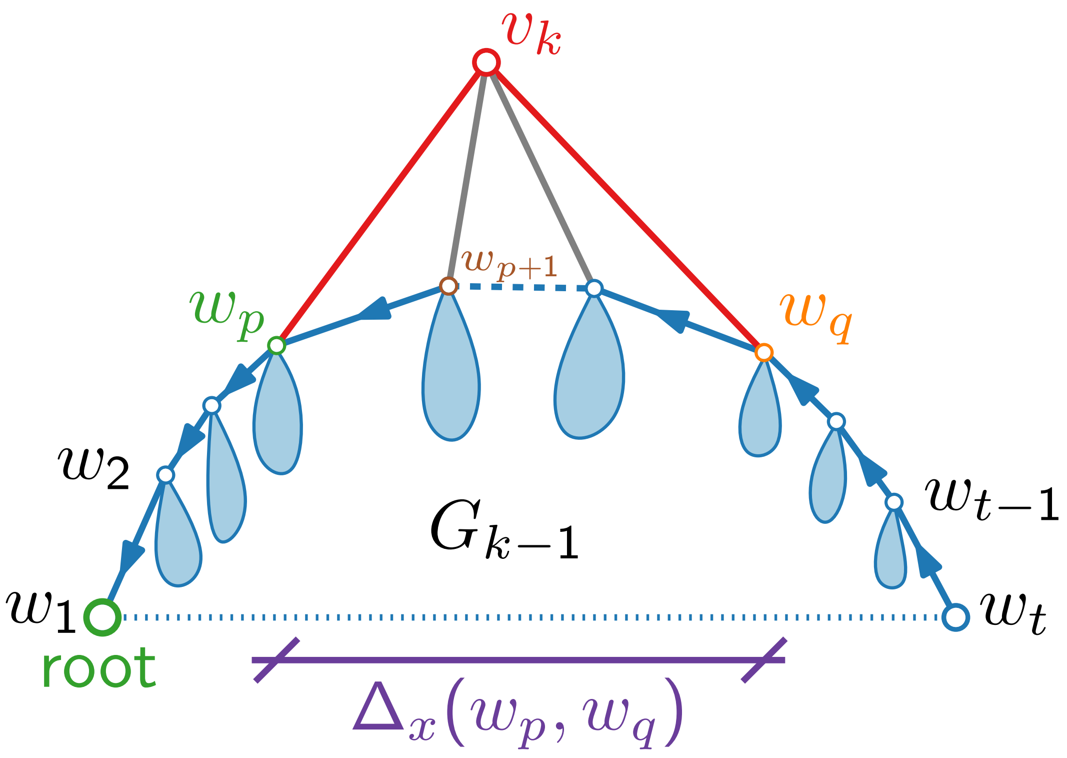
Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.

- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$



(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

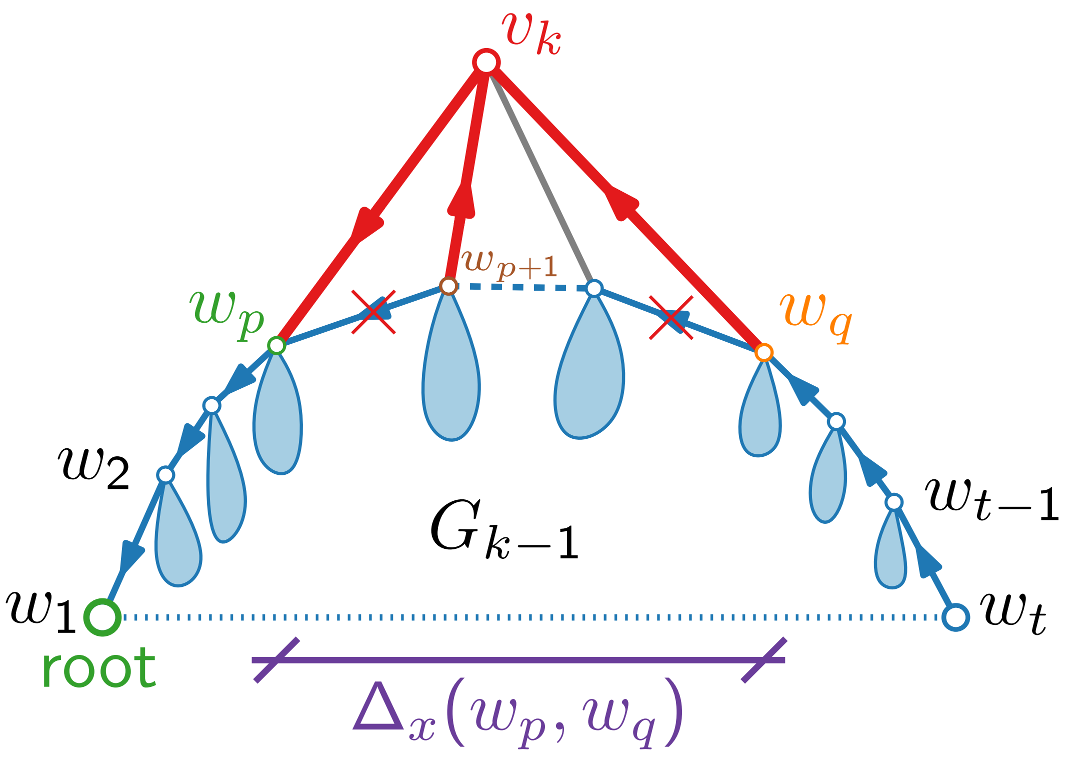
Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.

- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$



(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

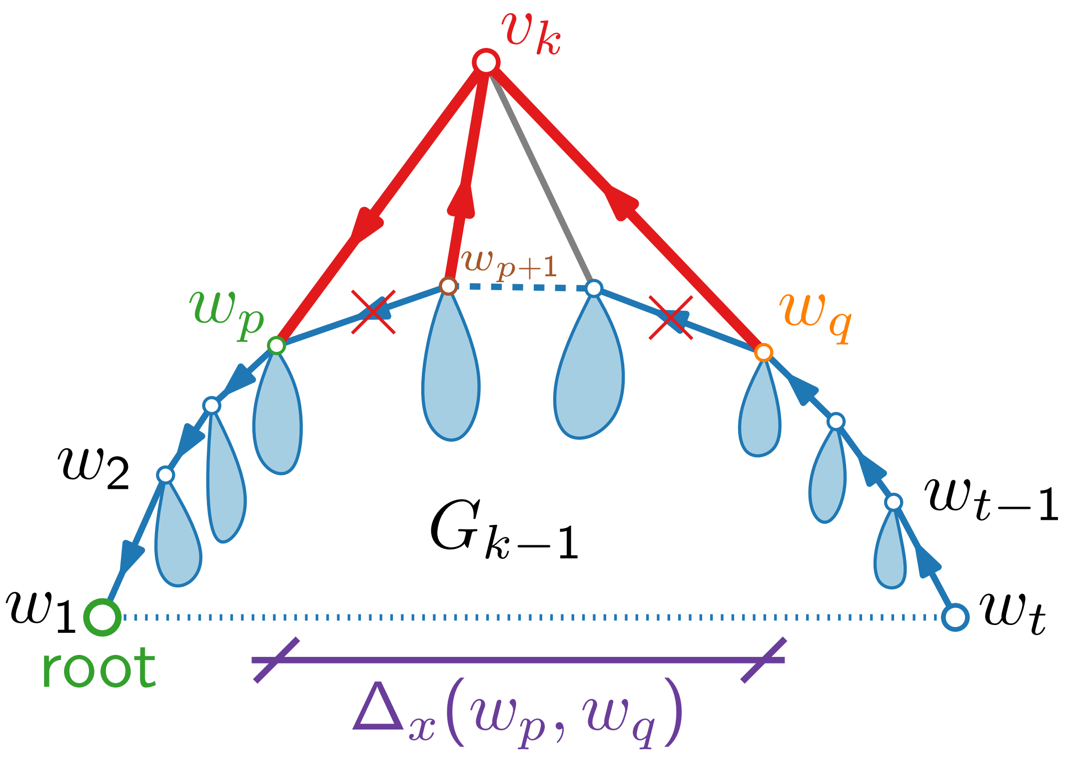
Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.

- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$
- $\Delta_x(v_k)$ by (3)

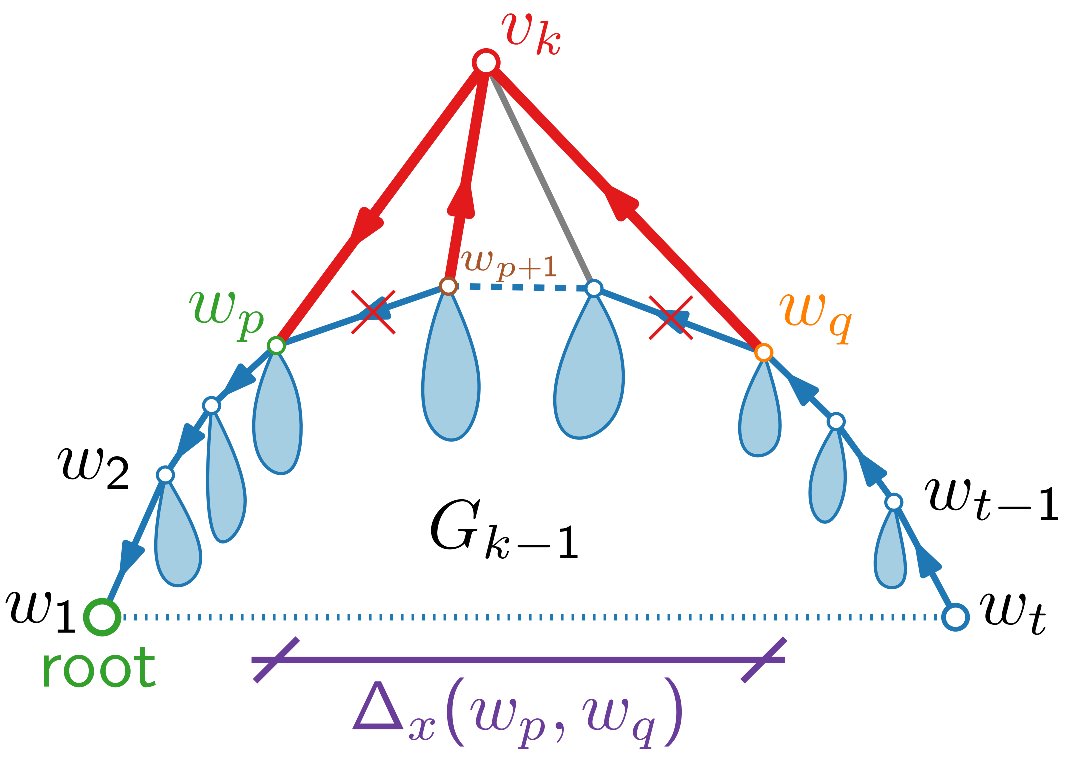


(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation



Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.

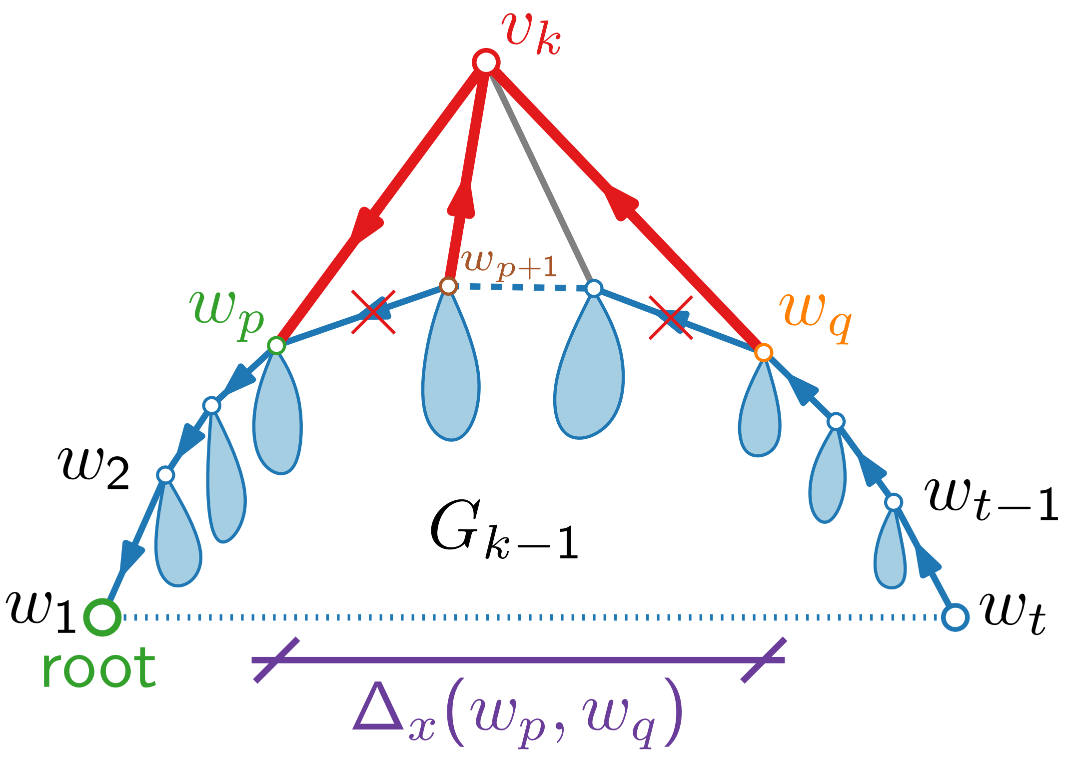
- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$
- $\Delta_x(v_k)$ by (3)

(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

(3) $\underbrace{x(v_k) - x(w_p)}_{\Delta_x(v_k)} = \frac{1}{2}(\underbrace{x(w_q) - x(w_p)}_{\Delta_x(w_p, w_q)} + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation



Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.

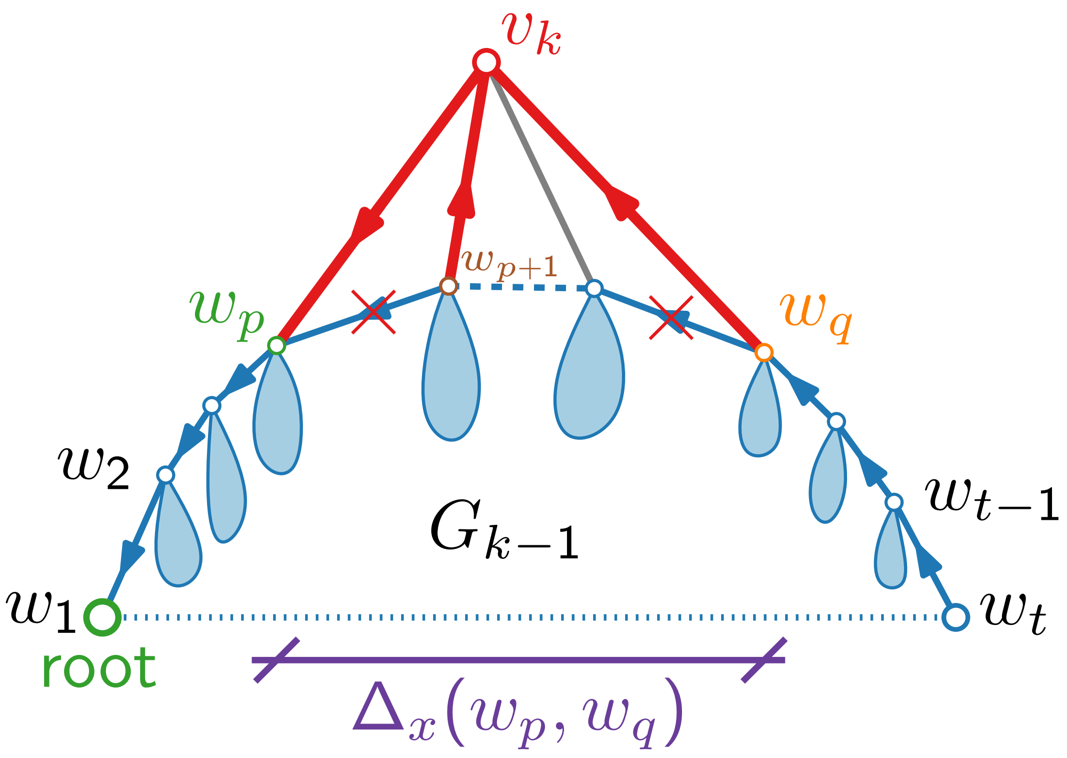
- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$
- $\Delta_x(v_k)$ by (3)
- $y(v_k)$ by (2)

(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

(3) $\underbrace{x(v_k) - x(w_p)}_{\Delta_x(v_k)} = \frac{1}{2}(\underbrace{x(w_q) - x(w_p)}_{\Delta_x(w_p, w_q)} + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation



Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.

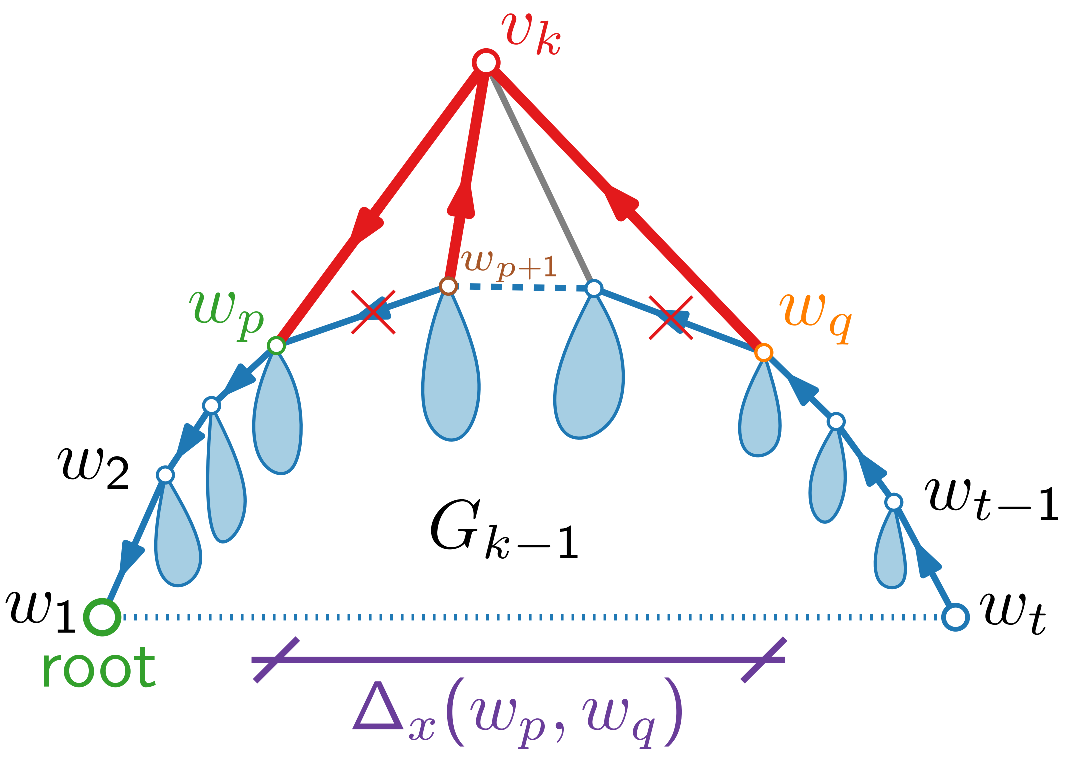
- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$
- $\Delta_x(v_k)$ by (3)
- $y(v_k)$ by (2)
- $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$

(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

(3) $\underbrace{x(v_k) - x(w_p)}_{\Delta_x(v_k)} = \frac{1}{2}(\underbrace{x(w_q) - x(w_p)}_{\Delta_x(w_p, w_q)} + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation



Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.

- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$
- $\Delta_x(v_k)$ by (3)
- $y(v_k)$ by (2)
- $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$
- $\Delta_x(w_{p+1}) = \Delta_x(w_{p+1}) - \Delta_x(v_k)$

(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

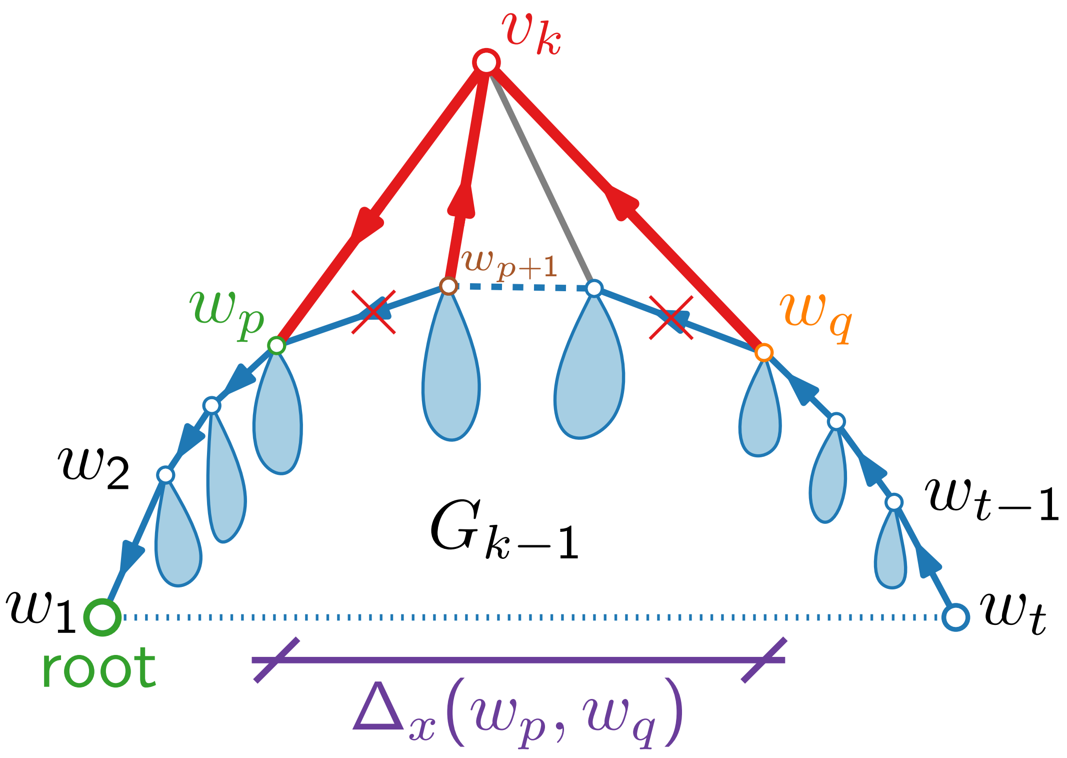
(3) $\underbrace{x(v_k) - x(w_p)}_{\Delta_x(v_k)} = \frac{1}{2}(\underbrace{x(w_q) - x(w_p)}_{\Delta_x(w_p, w_q)} + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$



Calculations.

- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$
- $\Delta_x(v_k)$ by (3)
- $y(v_k)$ by (2)
- $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$
- $\Delta_x(w_{p+1}) = \Delta_x(w_{p+1}) - \Delta_x(v_k)$

takes ? time

$$(1)\quad x(v_k) = \tfrac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$$

$$(2)\quad y(v_k) = \tfrac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$$

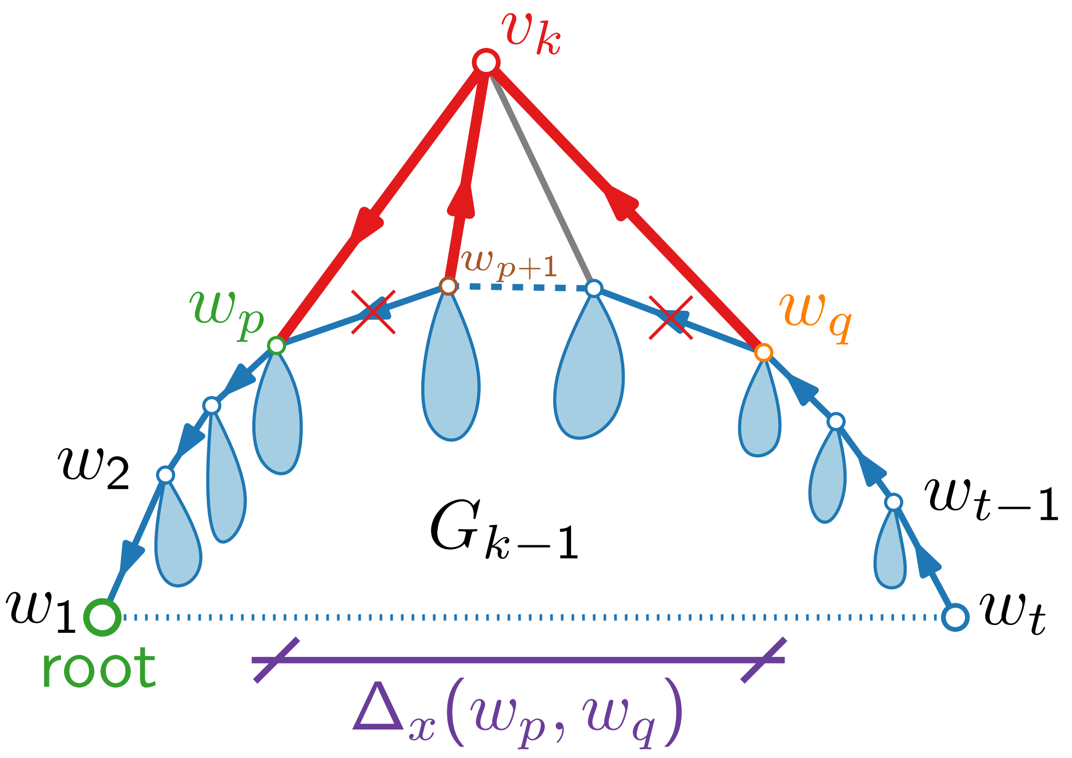
$$(3)\quad \underbrace{x(v_k) - x(w_p)}_{\Delta_x(v_k)} = \tfrac{1}{2}(\underbrace{x(w_q) - x(w_p)}_{\Delta_x(w_p, w_q)} + y(w_q) - y(w_p))$$

Shift Method – Linear-Time Implementation

Relative x-distance tree.

For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$



Calculations.

- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$ ← takes $\mathcal{O}(n)$ time
- $\Delta_x(v_k)$ by (3)
- $y(v_k)$ by (2)
- $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$
- $\Delta_x(w_{p+1}) = \Delta_x(w_{p+1}) - \Delta_x(v_k)$

(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

(3) $\underbrace{x(v_k) - x(w_p)}_{\Delta_x(v_k)} = \frac{1}{2}(\underbrace{x(w_q) - x(w_p)}_{\Delta_x(w_p, w_q)} + y(w_q) - y(w_p))$

Shift Method – Linear-Time Implementation

Relative x-distance tree.

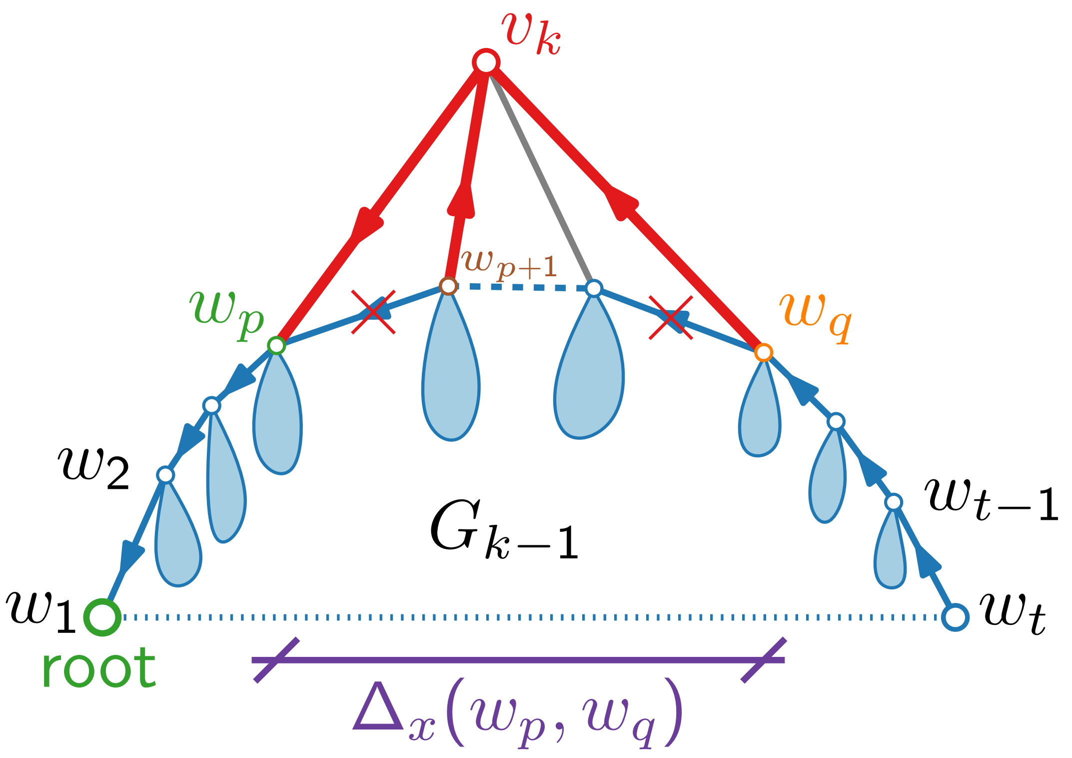
For each vertex v store

- x-offset $\Delta_x(v)$ from parent
- y-coordinate $y(v)$

Calculations.

- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \ldots + \Delta_x(w_q)$
- $\Delta_x(v_k)$ by (3)
- $y(v_k)$ by (2)
- $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$
- $\Delta_x(w_{p+1}) = \Delta_x(w_{p+1}) - \Delta_x(v_k)$

takes $\mathcal{O}(n)$ time in total ☺



$$(1)\quad x(v_k) = \tfrac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$$

$$(2)\quad y(v_k) = \tfrac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$$

$$(3)\quad \underbrace{x(v_k) - x(w_p)}_{\Delta_x(v_k)} = \tfrac{1}{2}(\underbrace{x(w_q) - x(w_p)}_{\Delta_x(w_p, w_q)} + y(w_q) - y(w_p))$$

Discussion

- The shift method by de Fraysseix, Pach, and Pollack provides an algorithmic tool to efficiently draw a plane graph onto a polynomial-size grid using only straight-line edges.

Discussion

- The shift method by de Fraysseix, Pach, and Pollack provides an algorithmic tool to efficiently draw a plane graph onto a polynomial-size grid using only straight-line edges.

- The linear-time implementation was later proposed by Chrobak and Payne.

Discussion

- The shift method by de Fraysseix, Pach, and Pollack provides an algorithmic tool to efficiently draw a plane graph onto a polynomial-size grid using only straight-line edges.
- The linear-time implementation was later proposed by Chrobak and Payne.
- Although we are guaranteed to get a very small grid, only straight-line edges, and no edge crossings, the resulting drawings are not always visually pleasing: the drawings tend to have very small angles and a big variance in the size of the triangular faces.

Discussion

- The shift method by de Fraysseix, Pach, and Pollack provides an algorithmic tool to efficiently draw a plane graph onto a polynomial-size grid using only straight-line edges.
- The linear-time implementation was later proposed by Chrobak and Payne.
- Although we are guaranteed to get a very small grid, only straight-line edges, and no edge crossings, the resulting drawings are not always visually pleasing: the drawings tend to have very small angles and a big variance in the size of the triangular faces.
- A quite different approach yielding similar results is by Schnyder ($\rightarrow$ next lecture).

Literature

- [PGD Ch. 4.2] for detailed explanation of the shift method
- [de Fraysseix, Pach, Pollack 1990] "How to draw a planar graph on a grid"
  – original paper introducing the shift method
- [Chrobak, Payne 1995] "A linear-time algorithm for drawing a planar graph on a grid"
  – original paper on how to implement the shift method in linear time