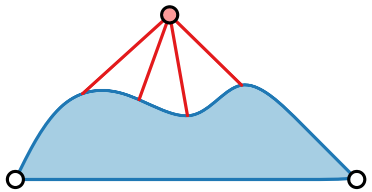
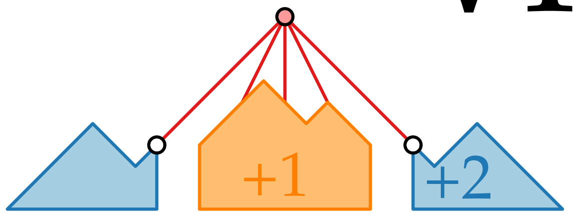
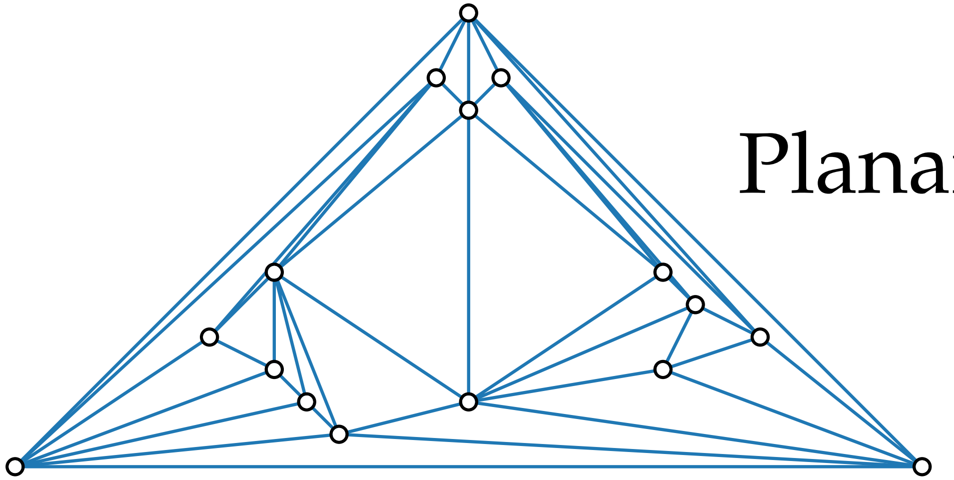


Visualization of Graphs



Lecture 4:
Straight-Line Drawings of Planar Graphs I: Canonical Ordering and Shift Method



Part I:
Planar Straight-Line Drawings

Philipp Kindermann

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 ⇒ planar
- No bends ⇒ straight-line

Drawing aestethics

- Area

Planar Graphs

Planar Graphs

Characterization

Planar Graphs

Characterization

Recognition

Planar Graphs

Characterization

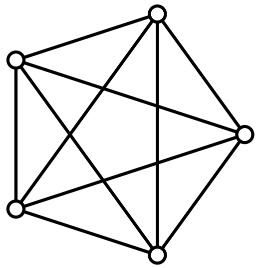
Recognition

Drawing

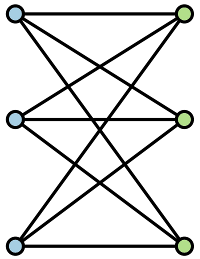
Planar Graphs

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

K_5



$K_{3,3}$



Characterization

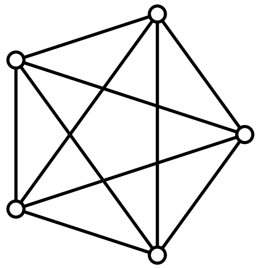
Recognition

Drawing

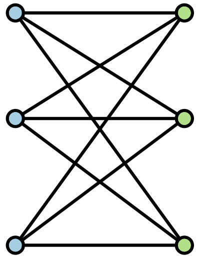
Planar Graphs

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

K_5



$K_{3,3}$



Characterization

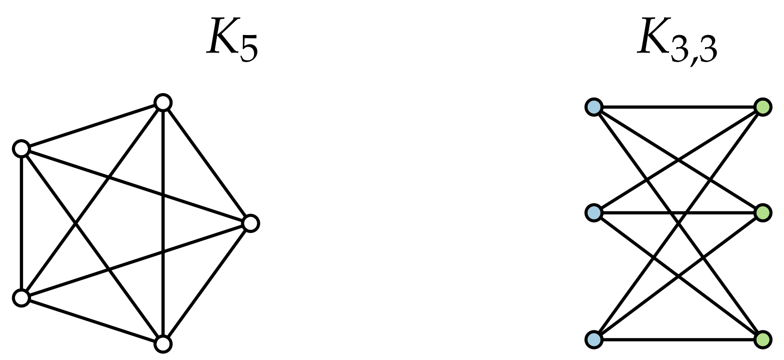
Theorem. [Hopcroft & Tarjan 1974]
For a graph G with n vertices, there is an $\mathcal{O}(n)$ time algorithm to test whether G is planar.

Recognition

Drawing

Planar Graphs

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



Characterization

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

Recognition

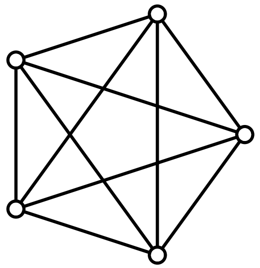
Also computes an embedding in $\mathcal{O}(n)$.

Drawing

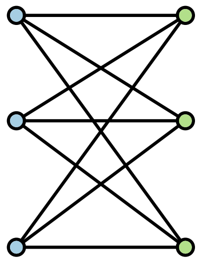
Planar Graphs

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

K_5



$K_{3,3}$



Characterization

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

Recognition

Also computes an embedding in $\mathcal{O}(n)$.

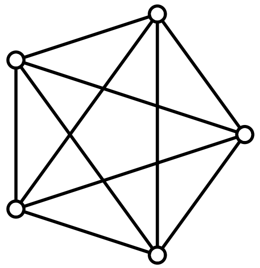
Theorem. [Wagner 1936, Fáry 1948, Stein 1951]
Every planar graph has a planar drawing where the edges are straight-line segments.

Drawing

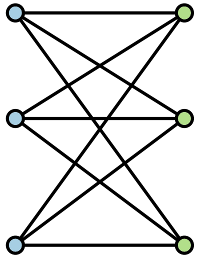
Planar Graphs

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

K_5



$K_{3,3}$



Characterization

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

Recognition

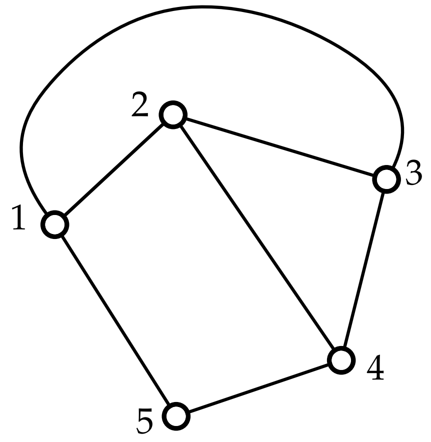
Also computes an embedding in $\mathcal{O}(n)$.

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

Drawing

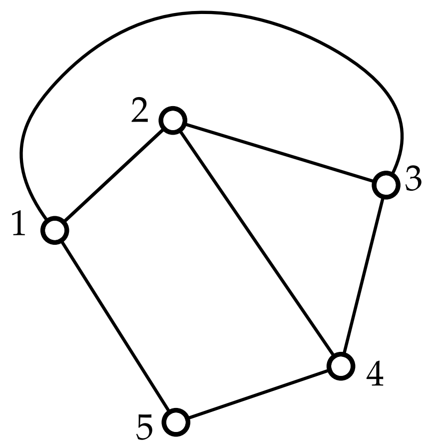
The algorithms implied by this theory produce drawings with area **not** bounded by any polynomial on n.

Triangulations



Triangulations

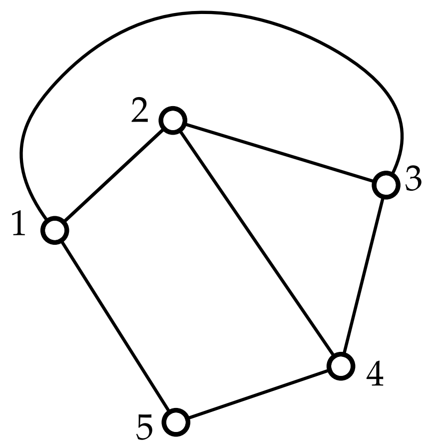
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

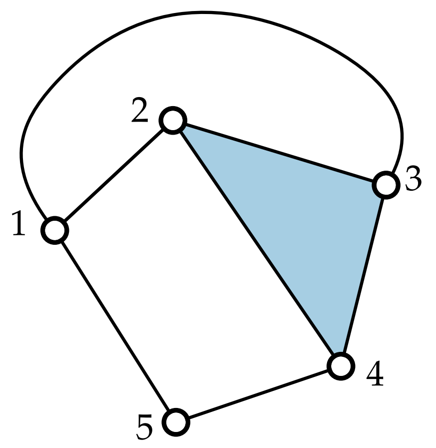
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

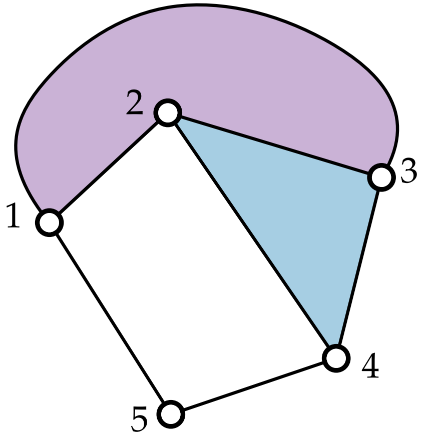
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

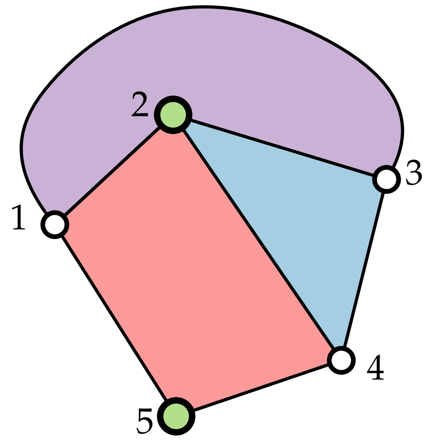
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

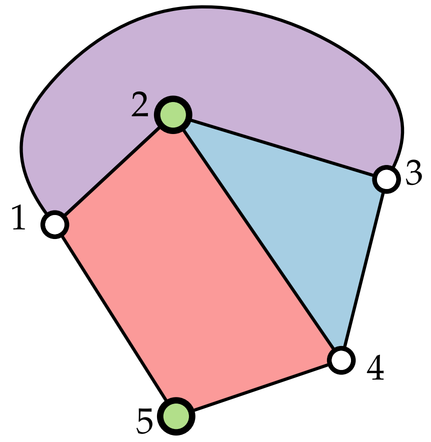
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

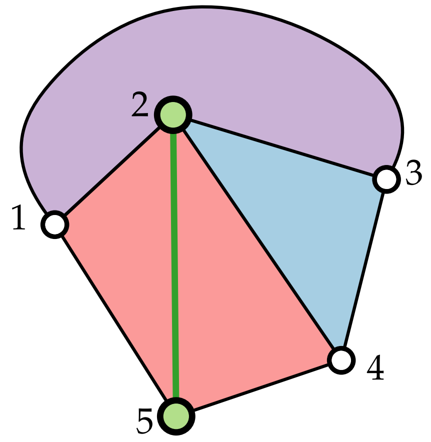
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

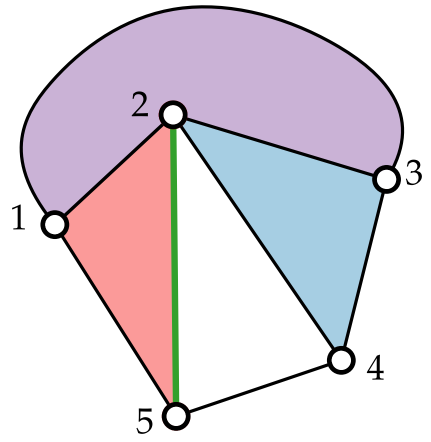
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

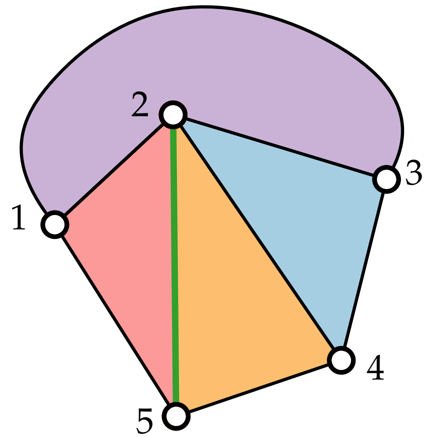
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

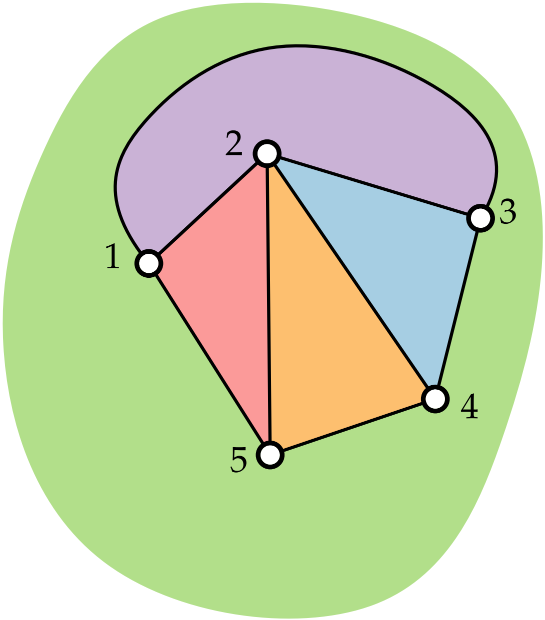
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

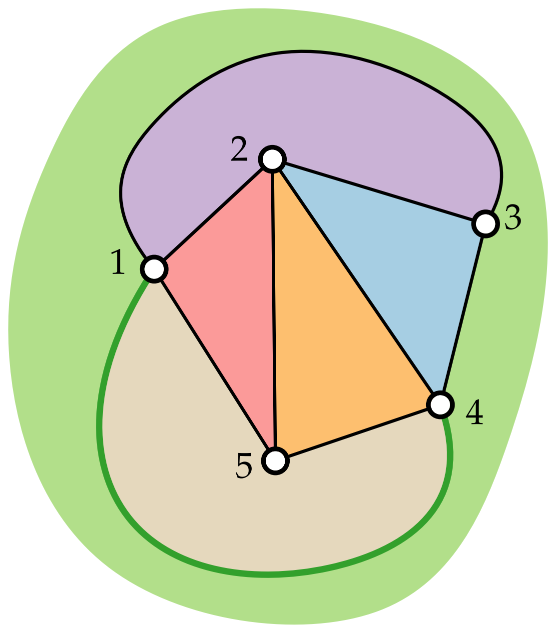
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

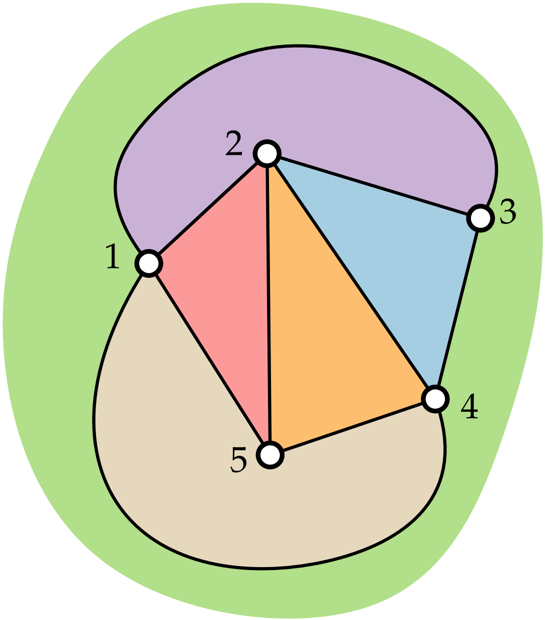
A **plane triangulation** is a plane graph where every face is a triangle.



Triangulations

with planar embedding

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

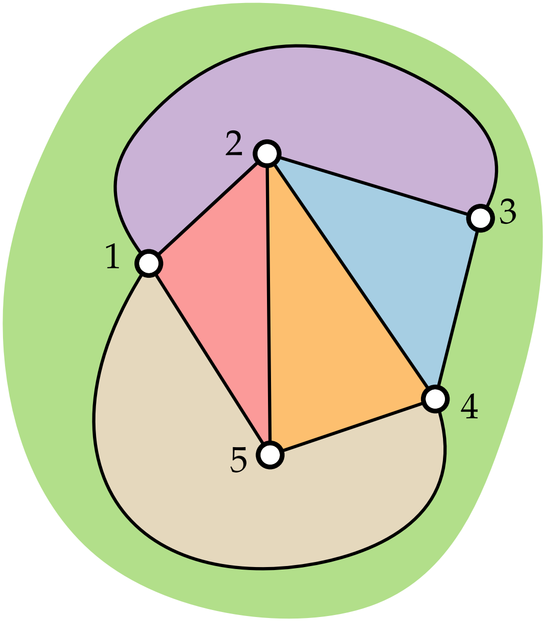


Triangulations

with 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 destroy planarity.

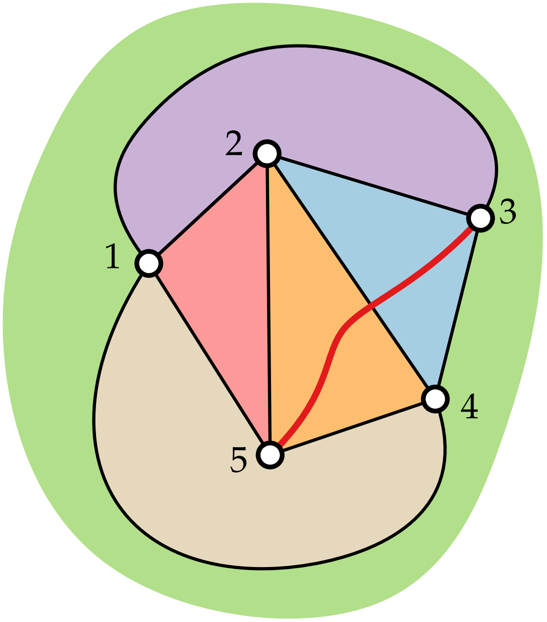


Triangulations

with 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 destroy planarity.

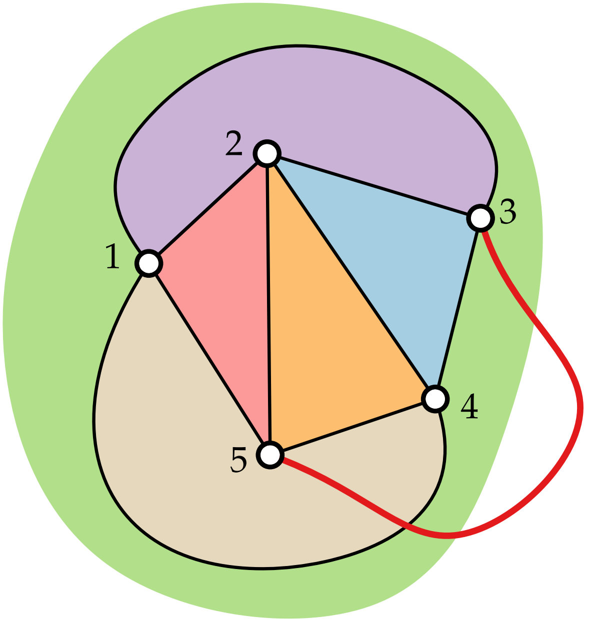


Triangulations

with 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 destroy planarity.

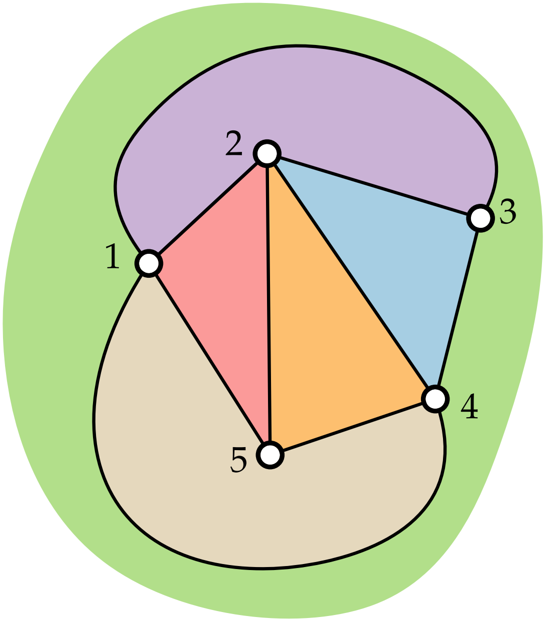


Triangulations

with 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 destroy planarity.



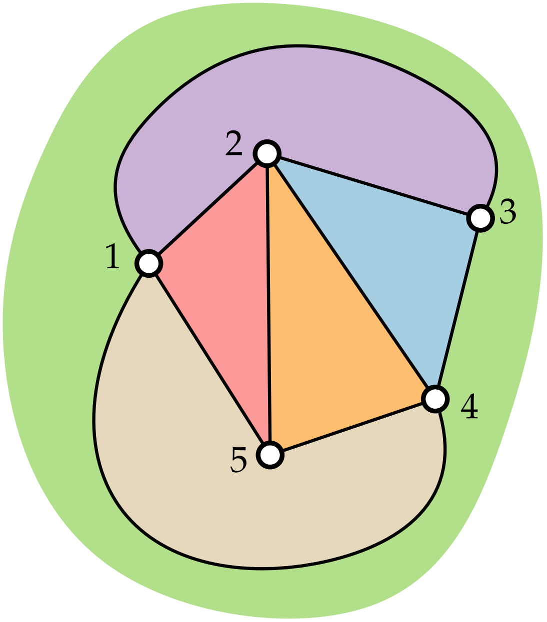
Triangulations

with 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 destroy planarity.

Observation.
A maximal plane graph is a plane triangulation.

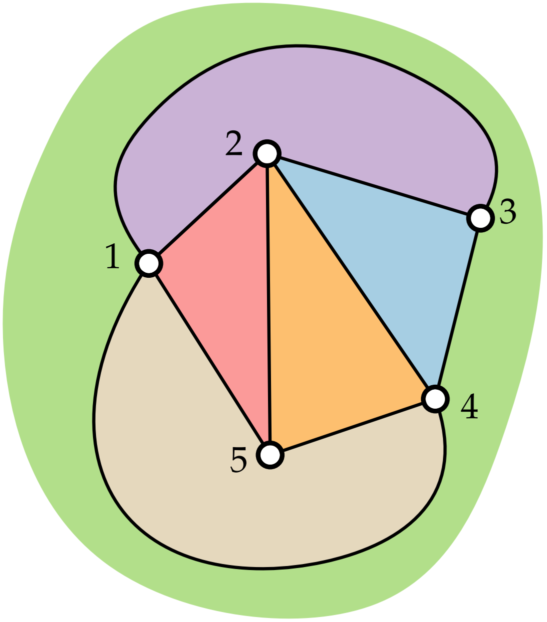


Triangulations

with 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 destroy planarity.



Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.

Triangulations

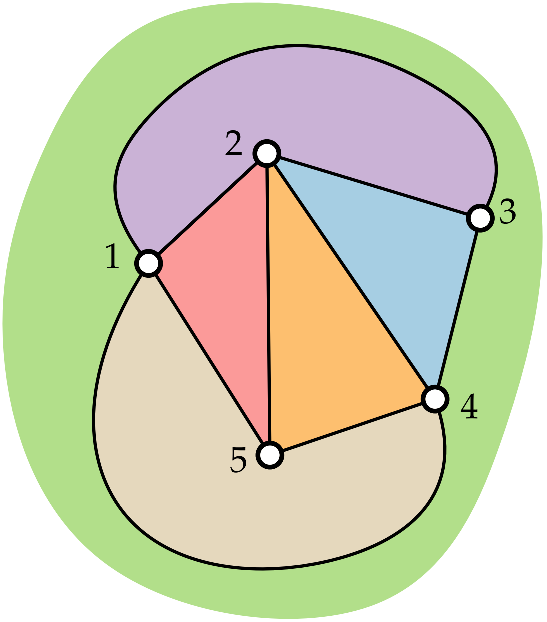
with planar embedding

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 destroy planarity.

Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.



We focus on plane triangulations:

Triangulations

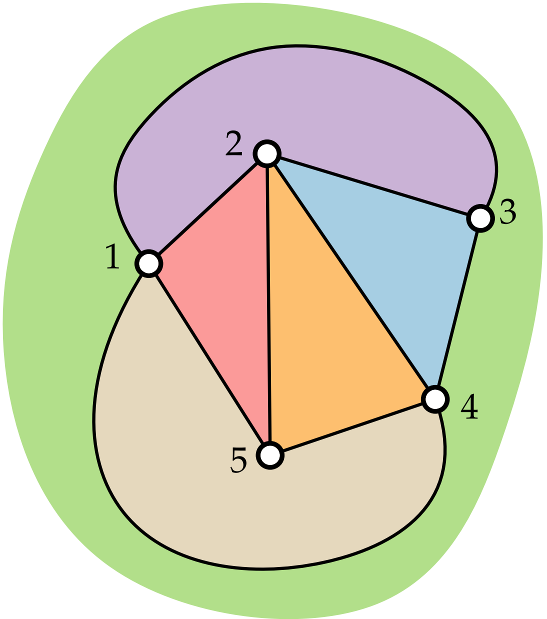
with planar embedding

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 destroy planarity.

Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.



We focus on plane triangulations:

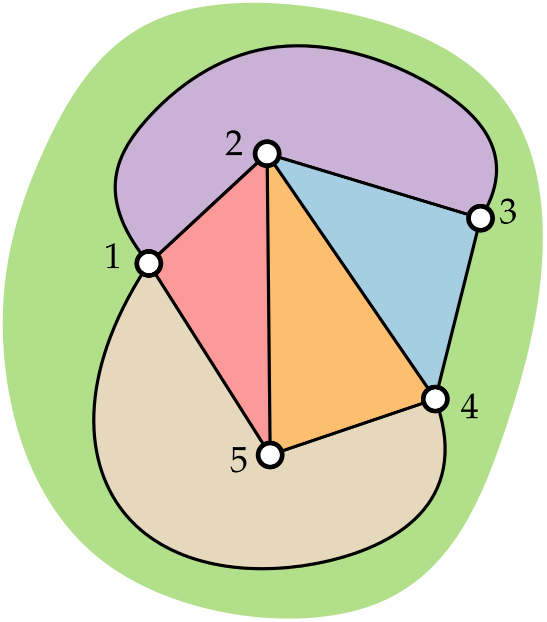
Lemma.
Every plane graph is subgraph of a plane triangulation.

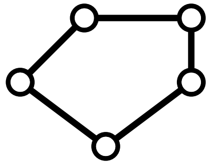
Triangulations

with 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 destroy planarity.





Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.

We focus on plane triangulations:

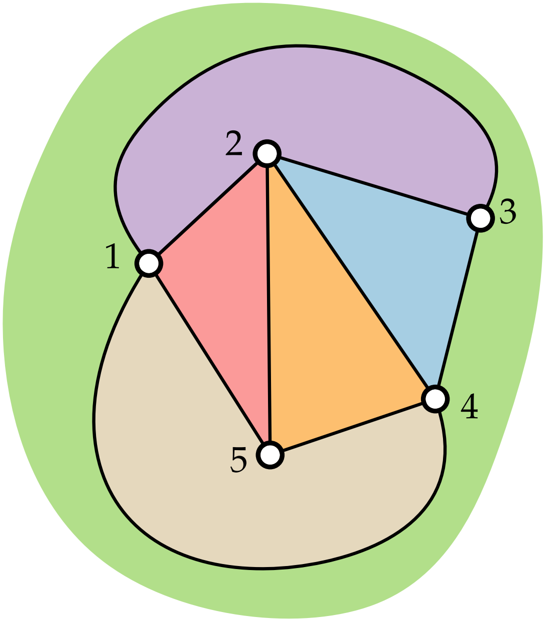
Lemma.
Every plane graph is subgraph of a plane triangulation.

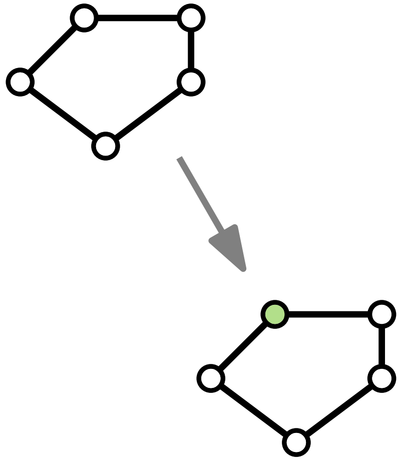
Triangulations

with 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 destroy planarity.





Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.

We focus on plane triangulations:

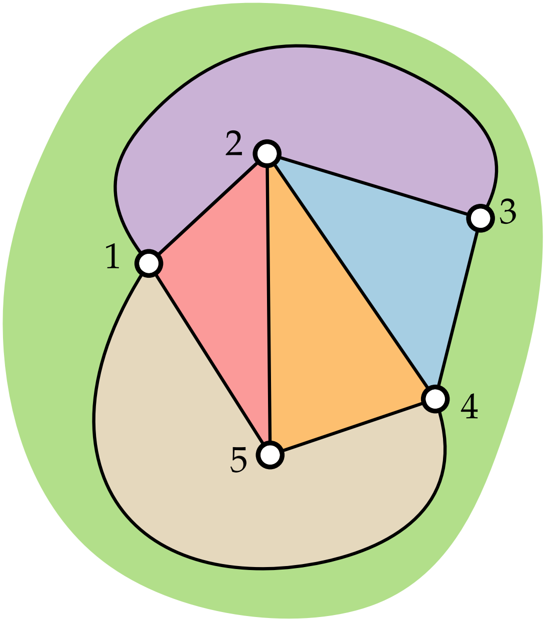
Lemma.
Every plane graph is subgraph of a plane triangulation.

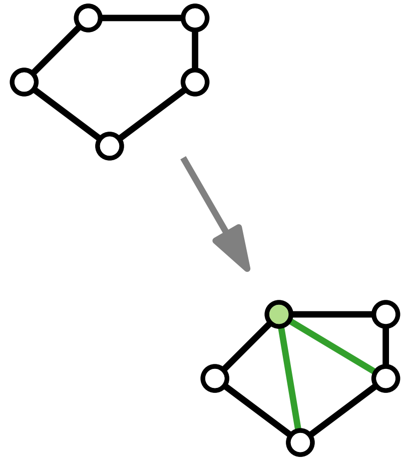
Triangulations

with 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 destroy planarity.





Observation.
A maximal plane graph is a plane triangulation.

We focus on plane triangulations:

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

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.

Triangulations

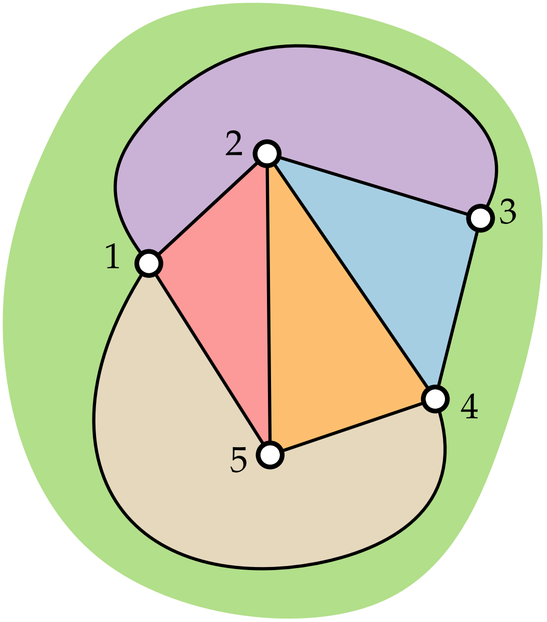
with planar embedding

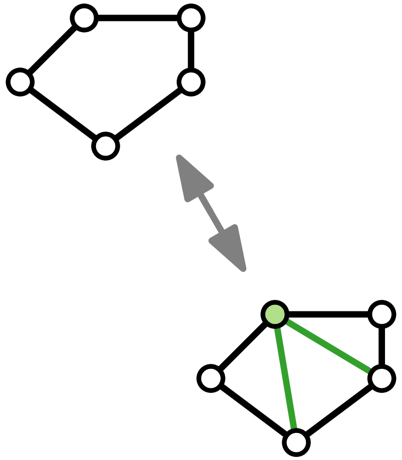
A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

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

Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.





We focus on plane triangulations:

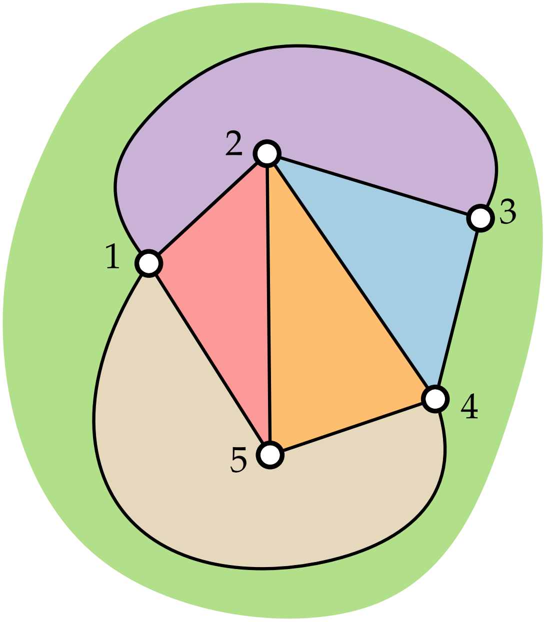
Lemma.
Every plane graph is subgraph of a plane triangulation.

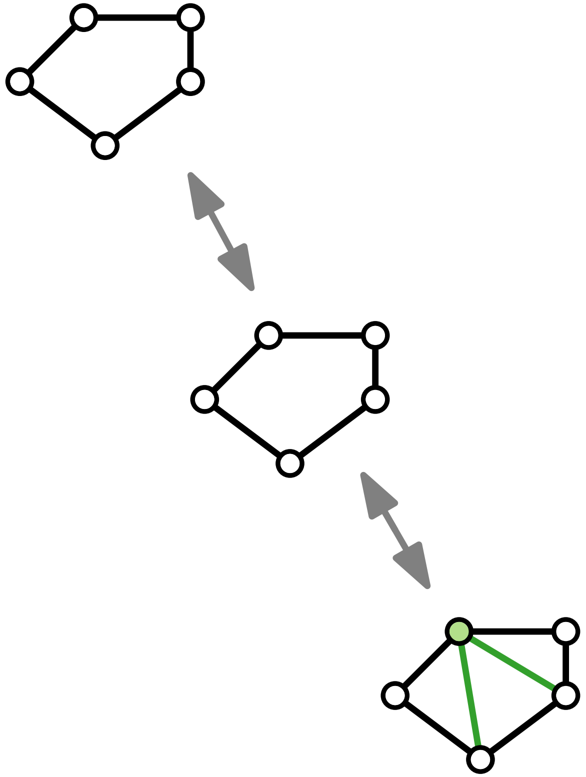
Triangulations

with 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 destroy planarity.





Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.

We focus on plane triangulations:

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

Triangulations

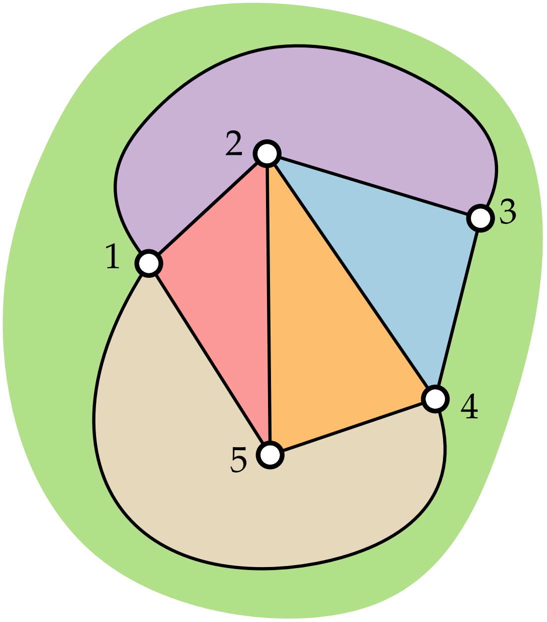
with planar embedding

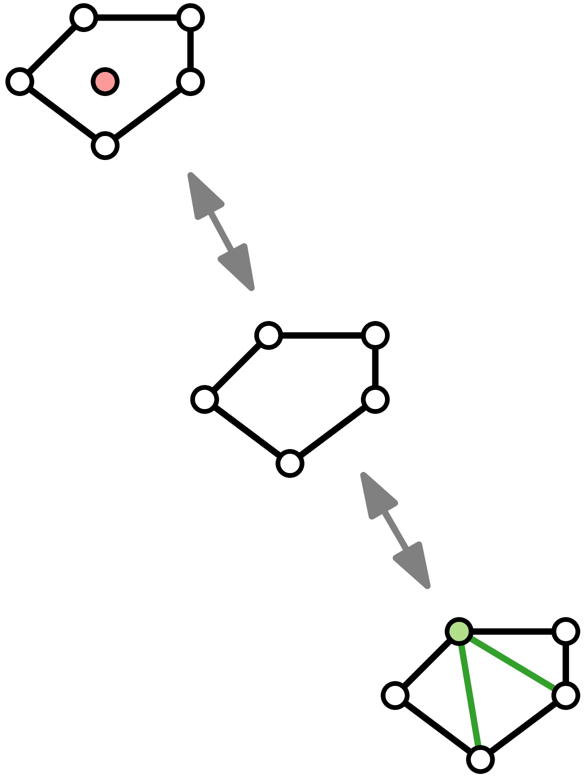
A **plane (inner) triangulation** is a plane graph where every (inner) face is a triangle.

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

Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.





We focus on plane triangulations:

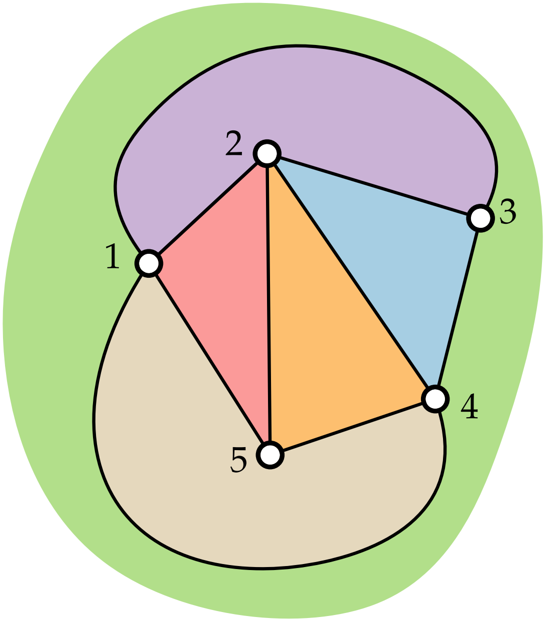
Lemma.
Every plane graph is subgraph of a plane triangulation.

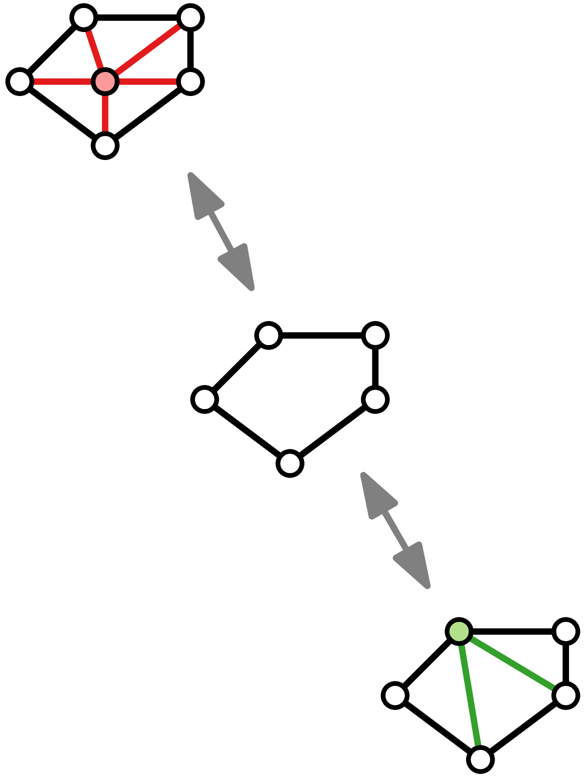
Triangulations

with 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 destroy planarity.





Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.

We focus on plane triangulations:

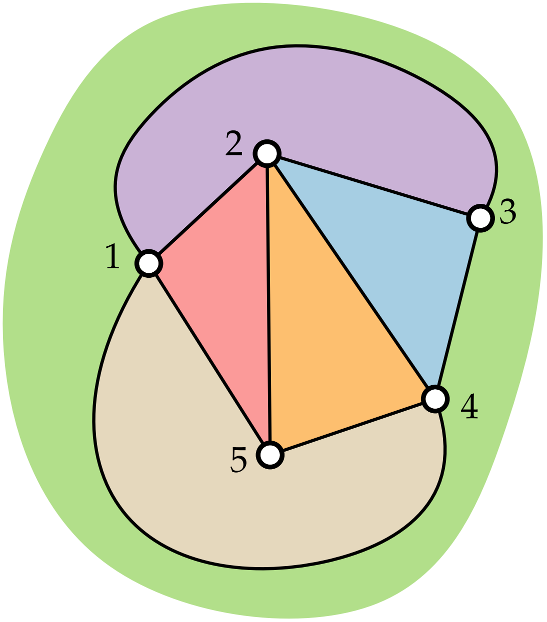
Lemma.
Every plane graph is subgraph of a plane triangulation.

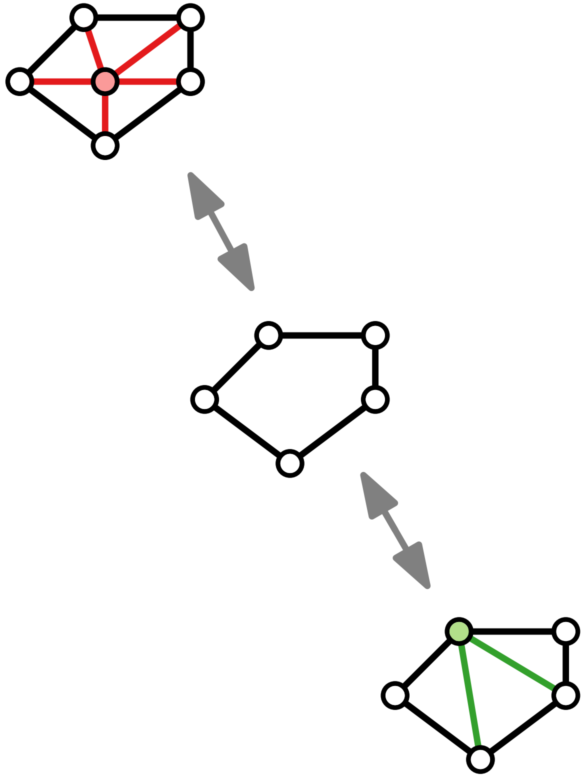
Triangulations

with 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 destroy planarity.





Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.

We focus on plane triangulations:

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

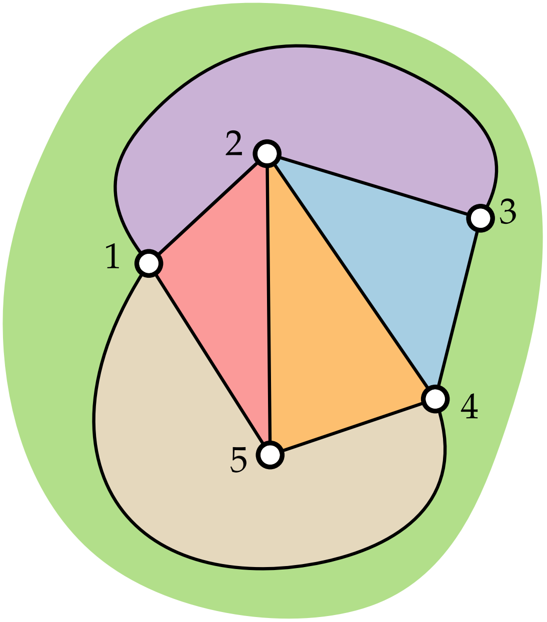
Corollary.
Tutte's algorithm creates a planar straight-line drawing for every planar graph.

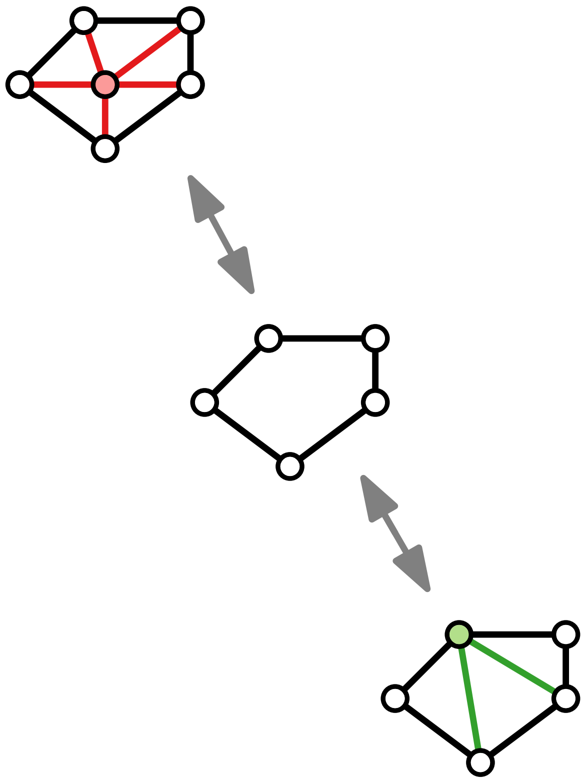
Triangulations

with 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 destroy planarity.





Observation.
A maximal plane graph is a plane triangulation.

Lemma.
A plane triangulation is at least 3-connected and thus has a unique planar embedding.

We focus on plane triangulations:

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

Corollary.
Tutte's algorithm creates a planar straight-line drawing for every planar graph. (but with exponential area)

Planar Straight-Line Drawings

Theorem. [De Fraysseix, Pach, Pollack '90]
Every n-vertex planar graph has a planar straight-line drawing of size

Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size

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

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 $(n-2) \times (n-2)$.

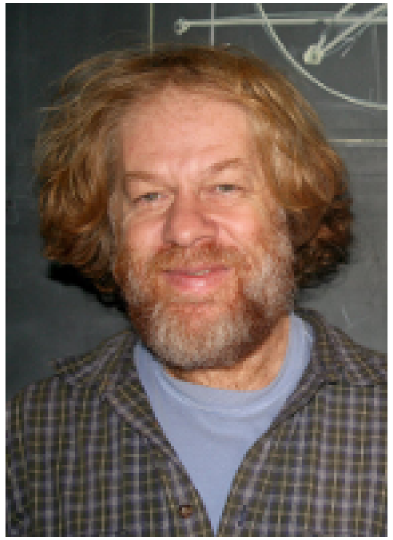
Planar Straight-Line Drawings

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 $(n-2) \times (n-2)$.

Hubert de Fraysseix
*Paris, France

János Pach
*1954, Budapest, Hungary

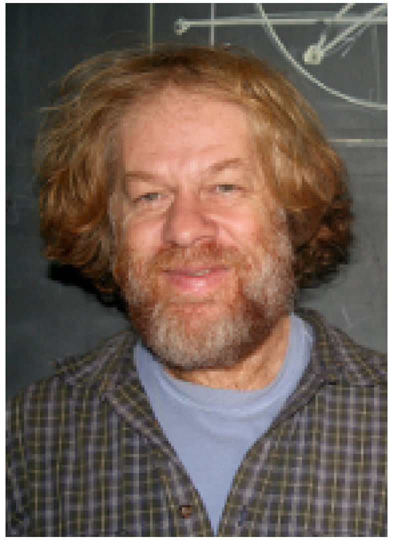


Richard Pollack
*1935, New York, USA
†2018, Montclair, USA

Planar Straight-Line Drawings

Hubert de Fraysseix
*Paris, France

János Pach
*1954, Budapest, Hungary



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.

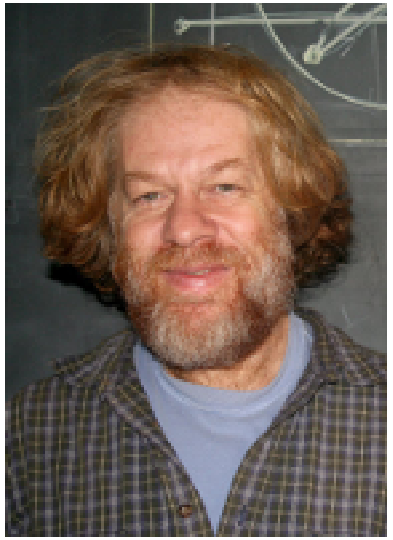
Richard Pollack
*1935, New York, USA
†2018, Montclair, USA

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

Hubert de Fraysseix
*Paris, France

János Pach
*1954, Budapest, Hungary



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.

- Start with single edge (v_1, v_2). Let this be G_2.

Richard Pollack
*1935, New York, USA
†2018, Montclair, USA

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 $(2n-4) \times (n-2)$.

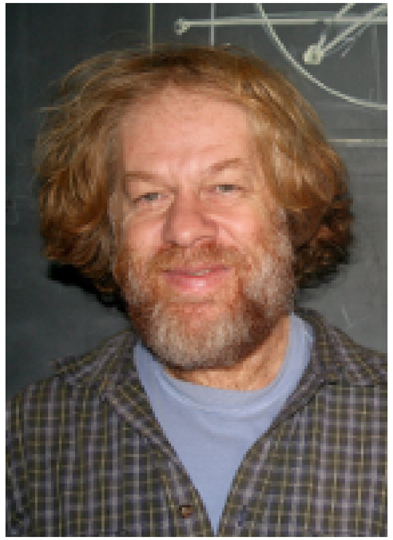
Idea.

- Start with single edge (v_1, v_2). Let this be G_2.
- To obtain G_{i+1}, add v_{i+1} to G_i so that neighbours of v_{i+1} are on the outer face of G_i.

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

Hubert de Fraysseix
*Paris, France

János Pach
*1954, Budapest, Hungary



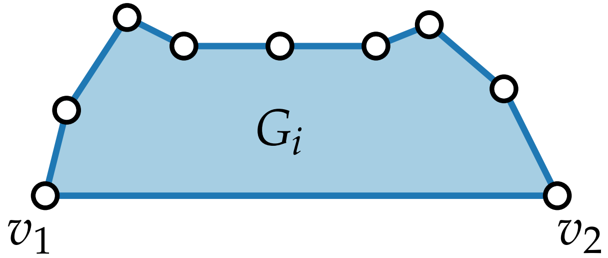
Richard Pollack
*1935, New York, USA
†2018, Montclair, USA

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.

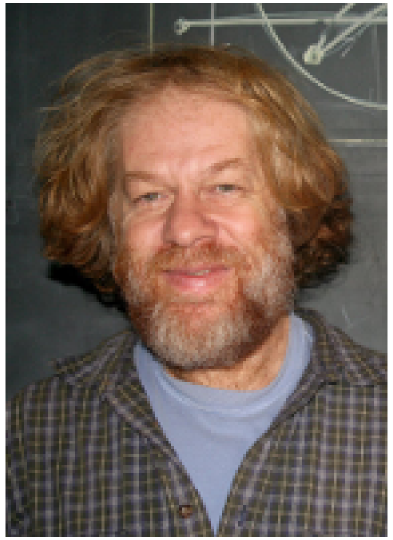
- Start with single edge (v_1, v_2). Let this be G_2.
- To obtain G_{i+1}, add v_{i+1} to G_i so that neighbours of v_{i+1} are on the outer face of G_i.



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

Hubert de Fraysseix
*Paris, France

János Pach
*1954, Budapest, Hungary

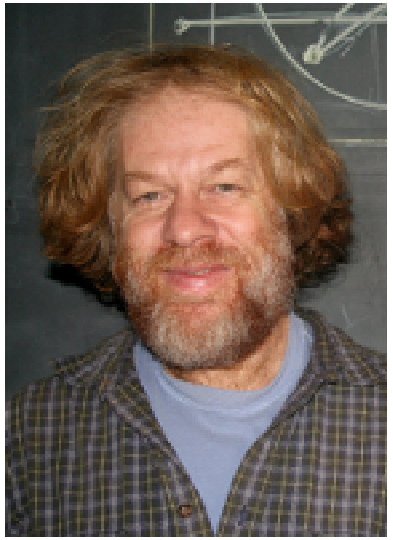


Richard Pollack
*1935, New York, USA
†2018, Montclair, USA

Planar Straight-Line Drawings

Hubert de Fraysseix
*Paris, France

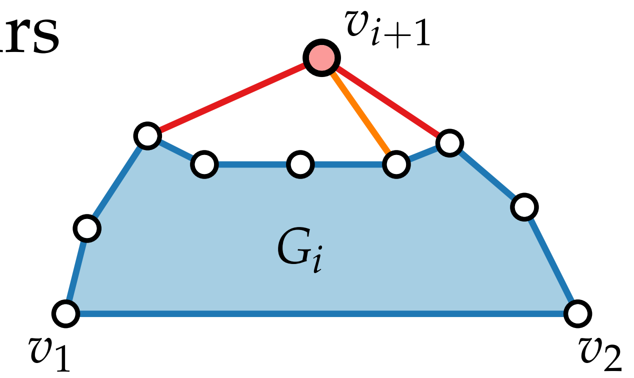
János Pach
*1954, Budapest, Hungary



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.

- Start with single edge (v_1, v_2). Let this be G_2.
- To obtain G_{i+1}, add v_{i+1} to G_i so that neighbours of v_{i+1} are on the outer face of G_i.



Richard Pollack
*1935, New York, USA
†2018, Montclair, USA

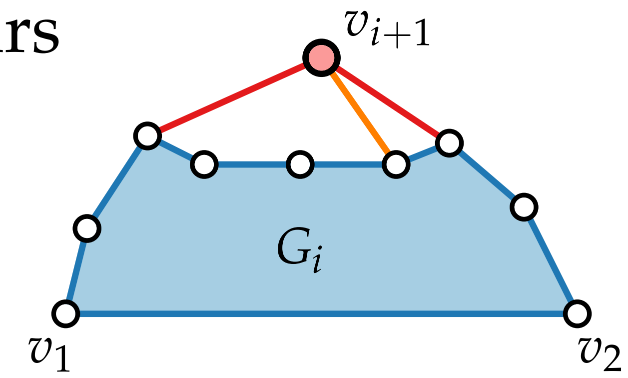
Theorem. [Schnyder '90]
Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

Planar Straight-Line Drawings

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

Idea.

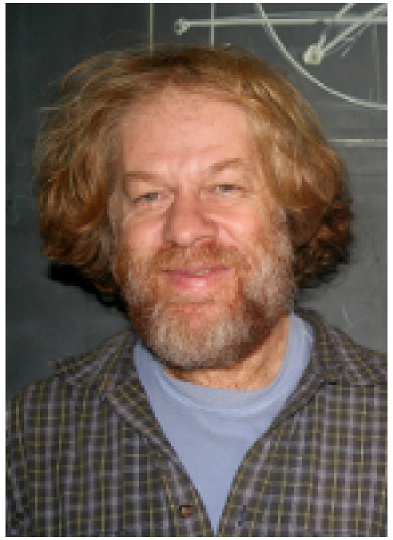
- Start with single edge (v_1, v_2). Let this be G_2.
- To obtain G_{i+1}, add v_{i+1} to G_i so that neighbours of v_{i+1} are on the outer face of G_i.
- Neighbours of v_{i+1} in G_i have to form path of length at least two.



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

Hubert de Fraysseix
*Paris, France

János Pach
*1954, Budapest, Hungary



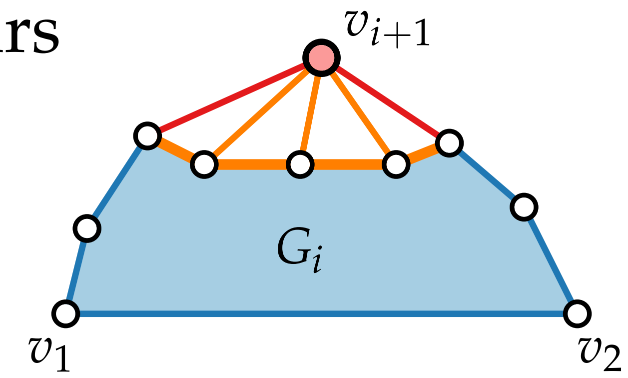
Richard Pollack
*1935, New York, USA
†2018, Montclair, USA

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.

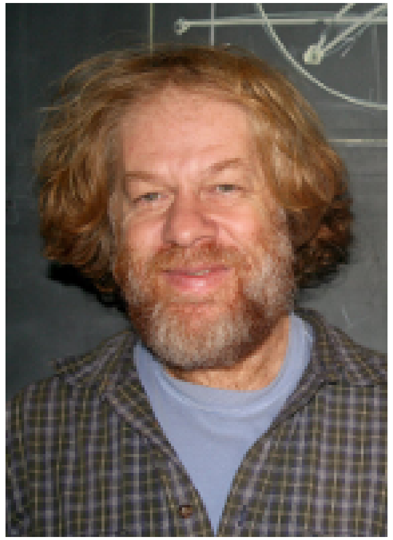
- Start with single edge (v_1, v_2). Let this be G_2.
- To obtain G_{i+1}, add v_{i+1} to G_i so that neighbours of v_{i+1} are on the outer face of G_i.
- Neighbours of v_{i+1} in G_i have to form path of length at least two.



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

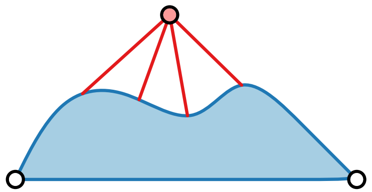
Hubert de Fraysseix
*Paris, France

János Pach
*1954, Budapest, Hungary

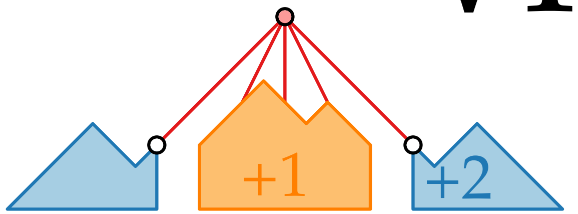


Richard Pollack
*1935, New York, USA
†2018, Montclair, USA

Universität Trier

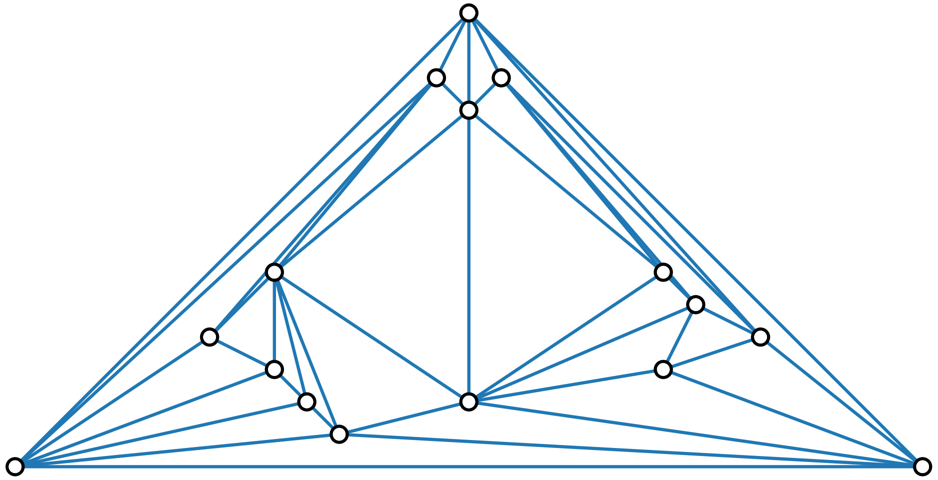


Visualization of Graphs



Lecture 4:
Straight-Line Drawings of Planar Graphs I: Canonical Ordering and Shift Method

Part II:
Canonical Order



Philipp Kindermann

Canonical Order – Definition

Definition.
Let $G = (V, E)$ be a triangulated plane graph on $n \geq 3$ vertices.

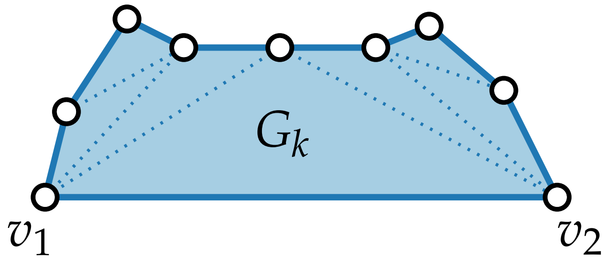
Canonical Order – Definition

Definition.
Let $G = (V, E)$ be a triangulated plane graph on $n \geq 3$ vertices.
An order $\pi = (v_1, v_2, \ldots, v_n)$ is called a **canonical order**, if the following conditions hold for each k, $3 \leq k \leq n$:

Canonical Order – Definition

Definition.
Let $G = (V, E)$ be a triangulated plane graph on $n \geq 3$ vertices.
An order $\pi = (v_1, v_2, \ldots, v_n)$ is called a **canonical order**, if the following conditions hold for each k, $3 \leq k \leq n$:

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

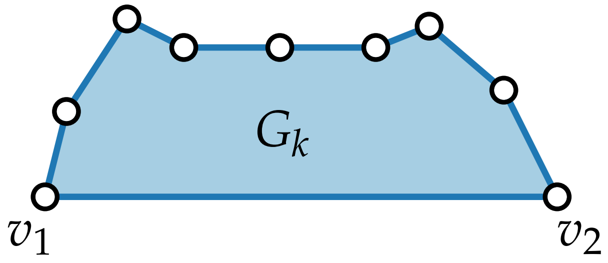


Canonical Order – Definition

Definition.
Let $G = (V, E)$ be a triangulated plane graph on $n \geq 3$ vertices.
An order $\pi = (v_1, v_2, \ldots, v_n)$ is called a **canonical order**, if the following conditions hold for each k, $3 \leq k \leq n$:

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

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



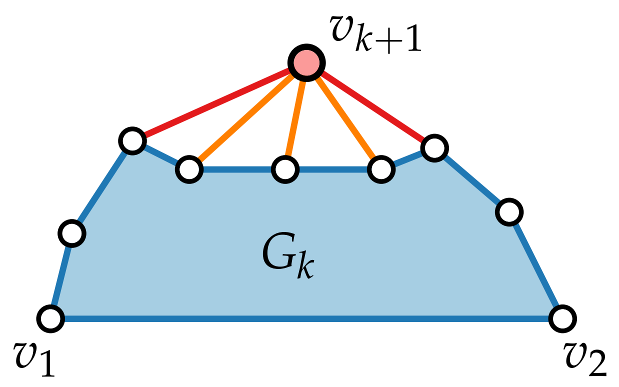
Canonical Order – Definition

Definition.
Let $G = (V, E)$ be a triangulated plane graph on $n \geq 3$ vertices.
An order $\pi = (v_1, v_2, \ldots, v_n)$ is called a **canonical order**, if the following conditions hold for each k, $3 \leq k \leq n$:

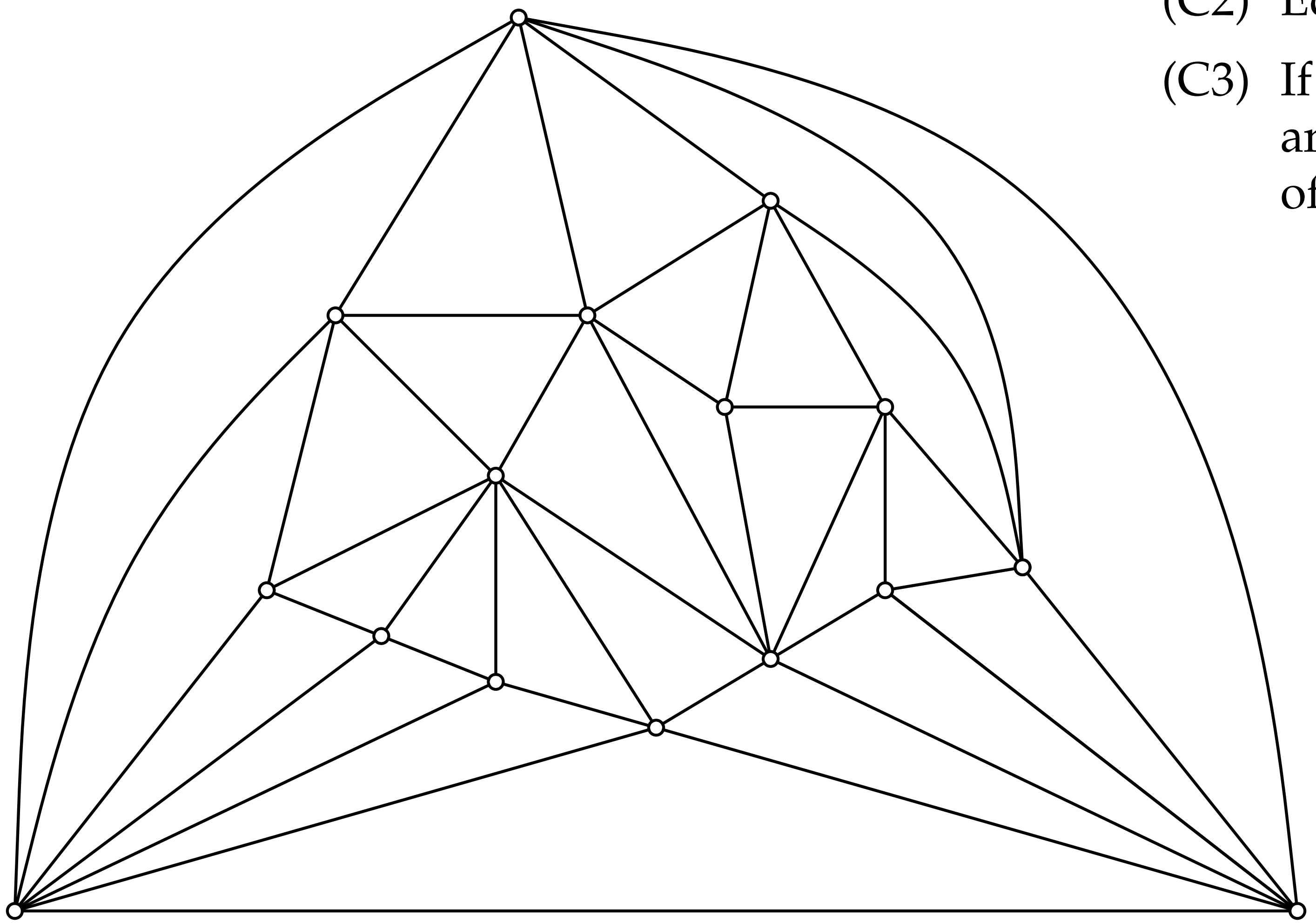
(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.



Canonical Order – Example

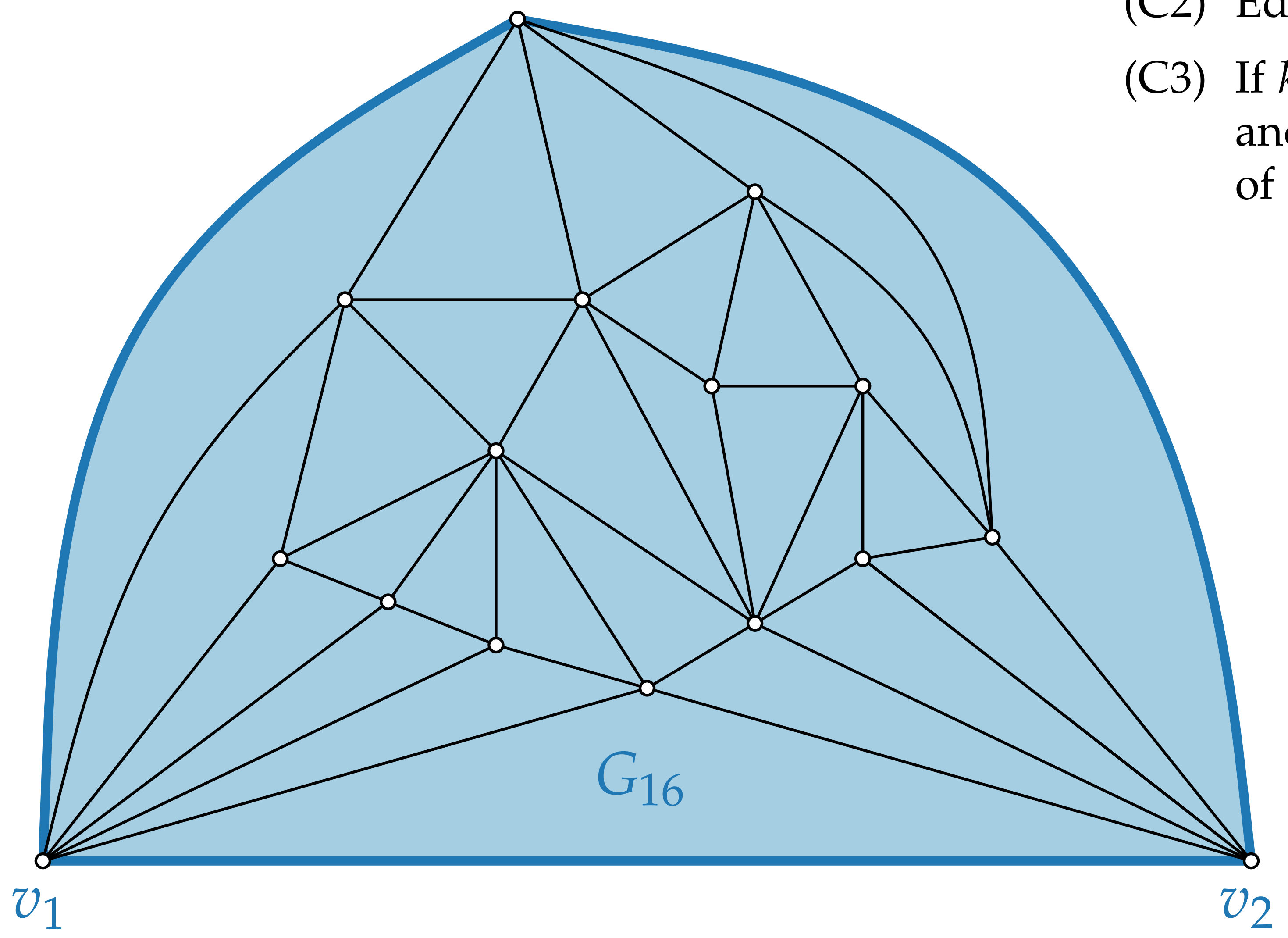


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

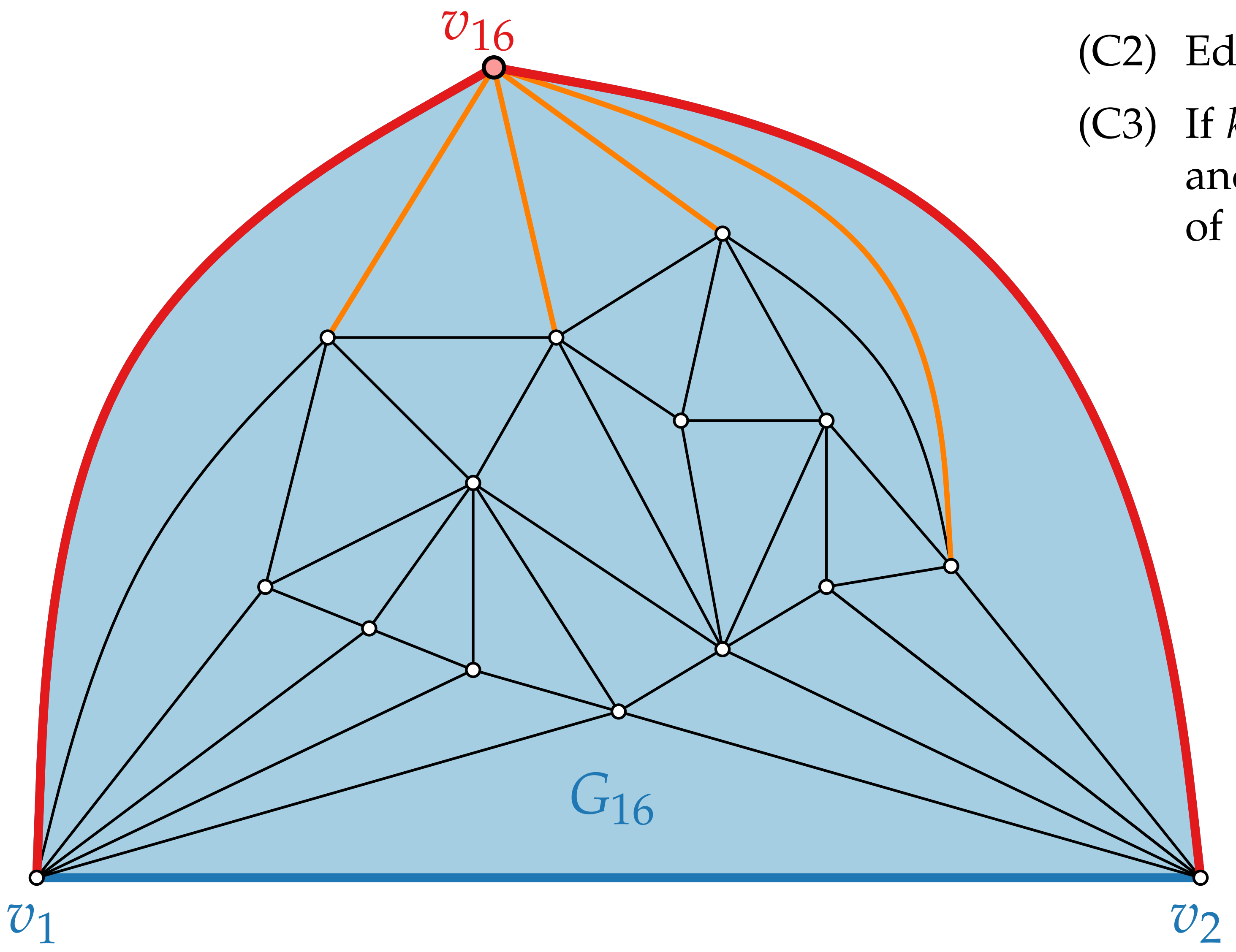


(C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

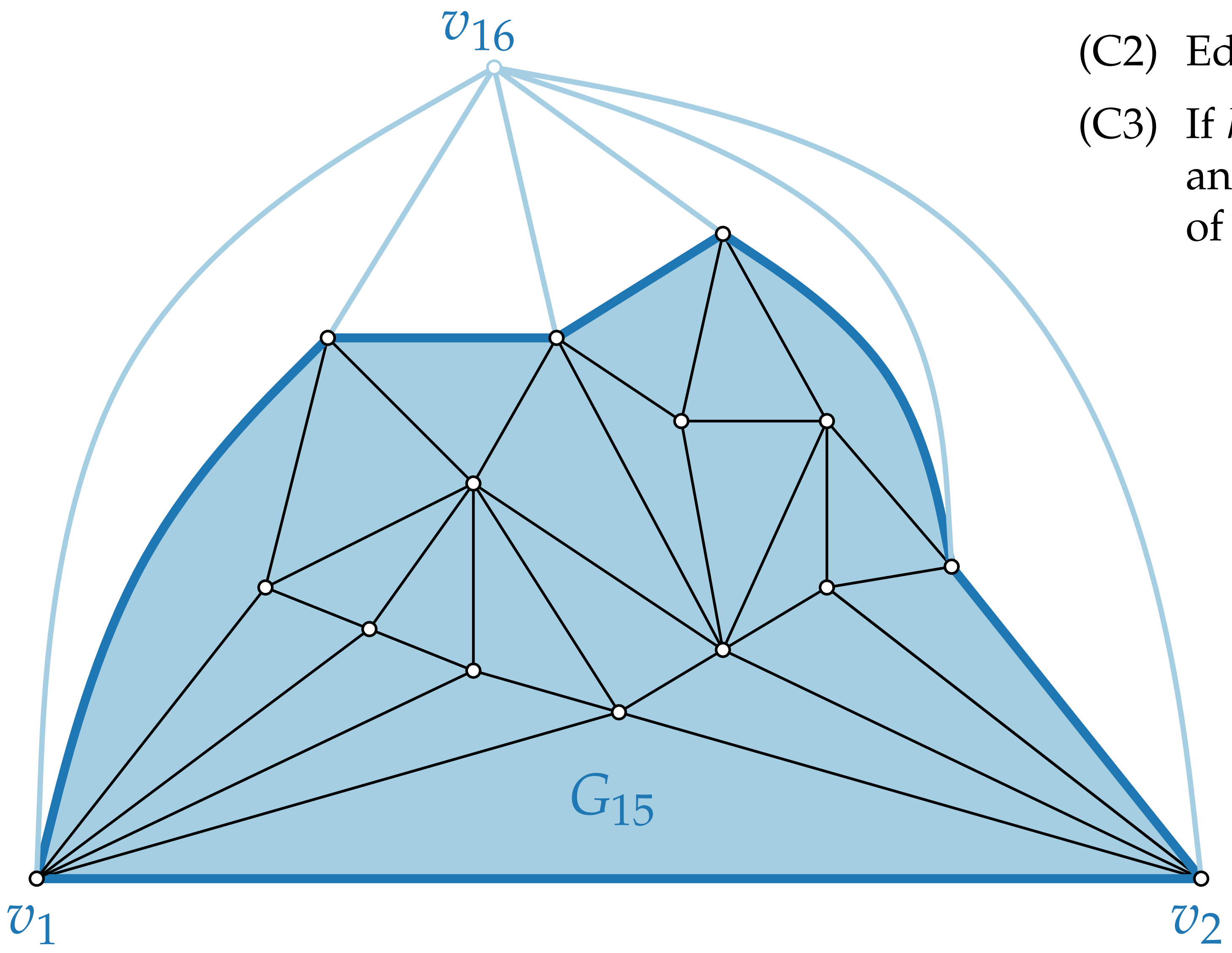


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

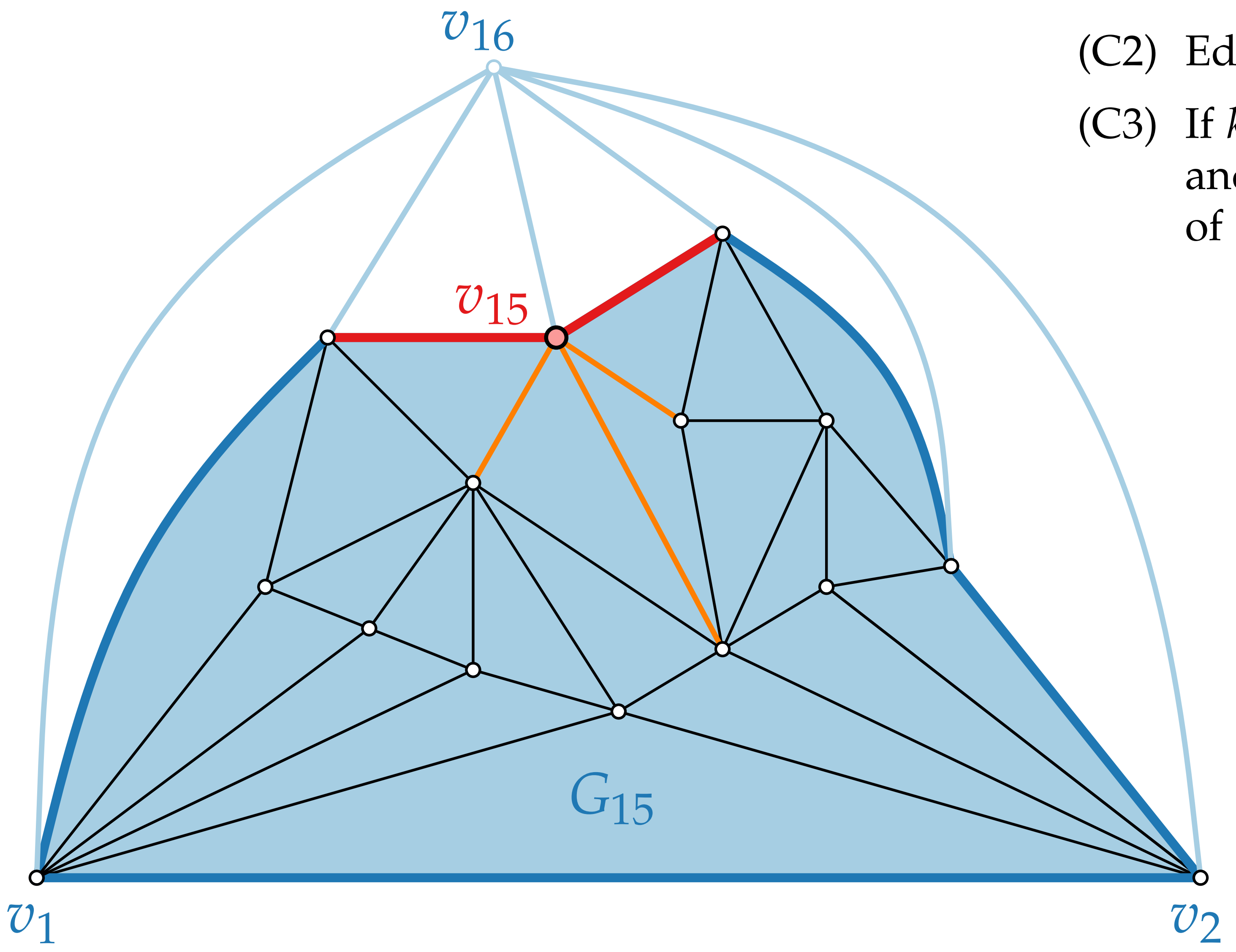


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

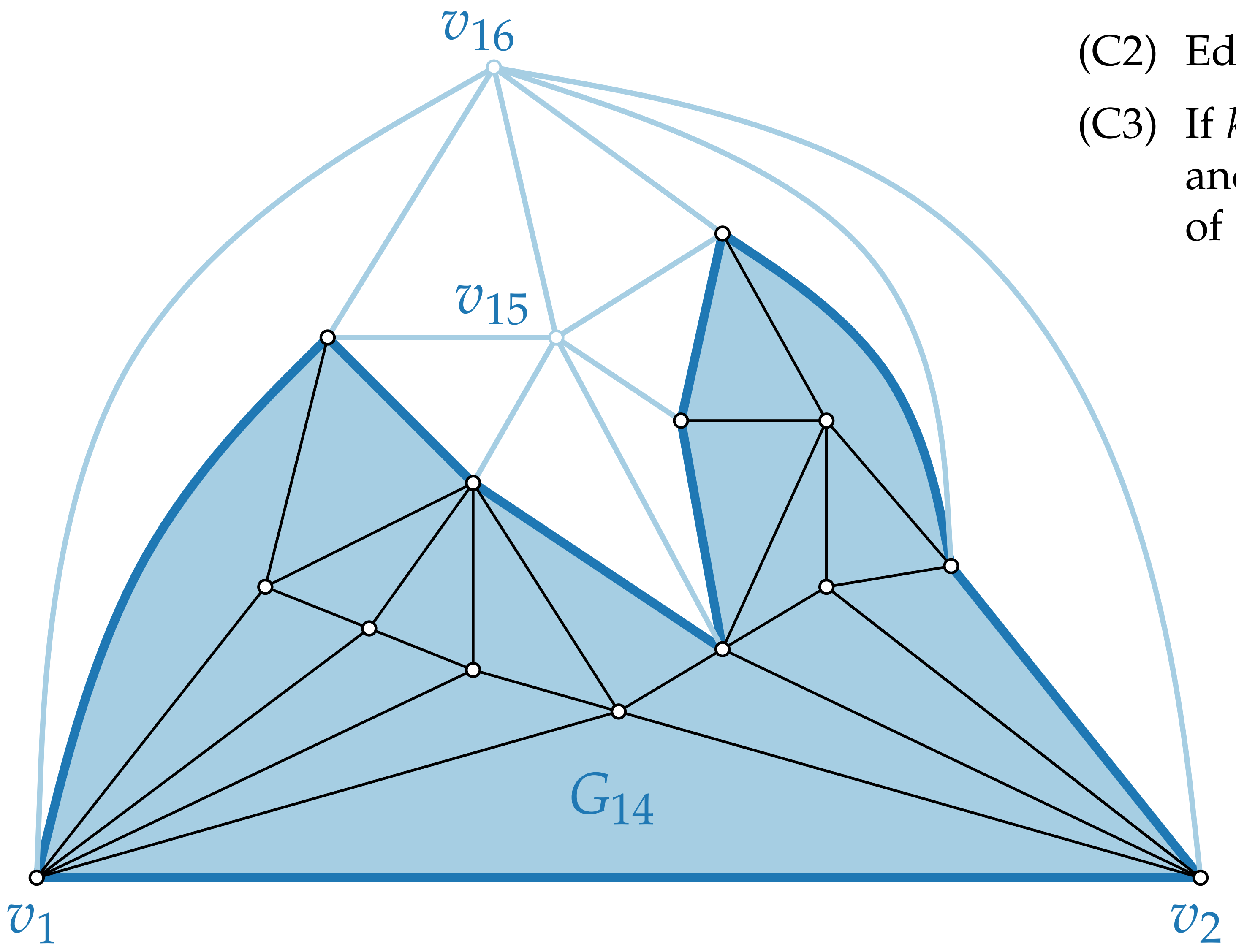


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

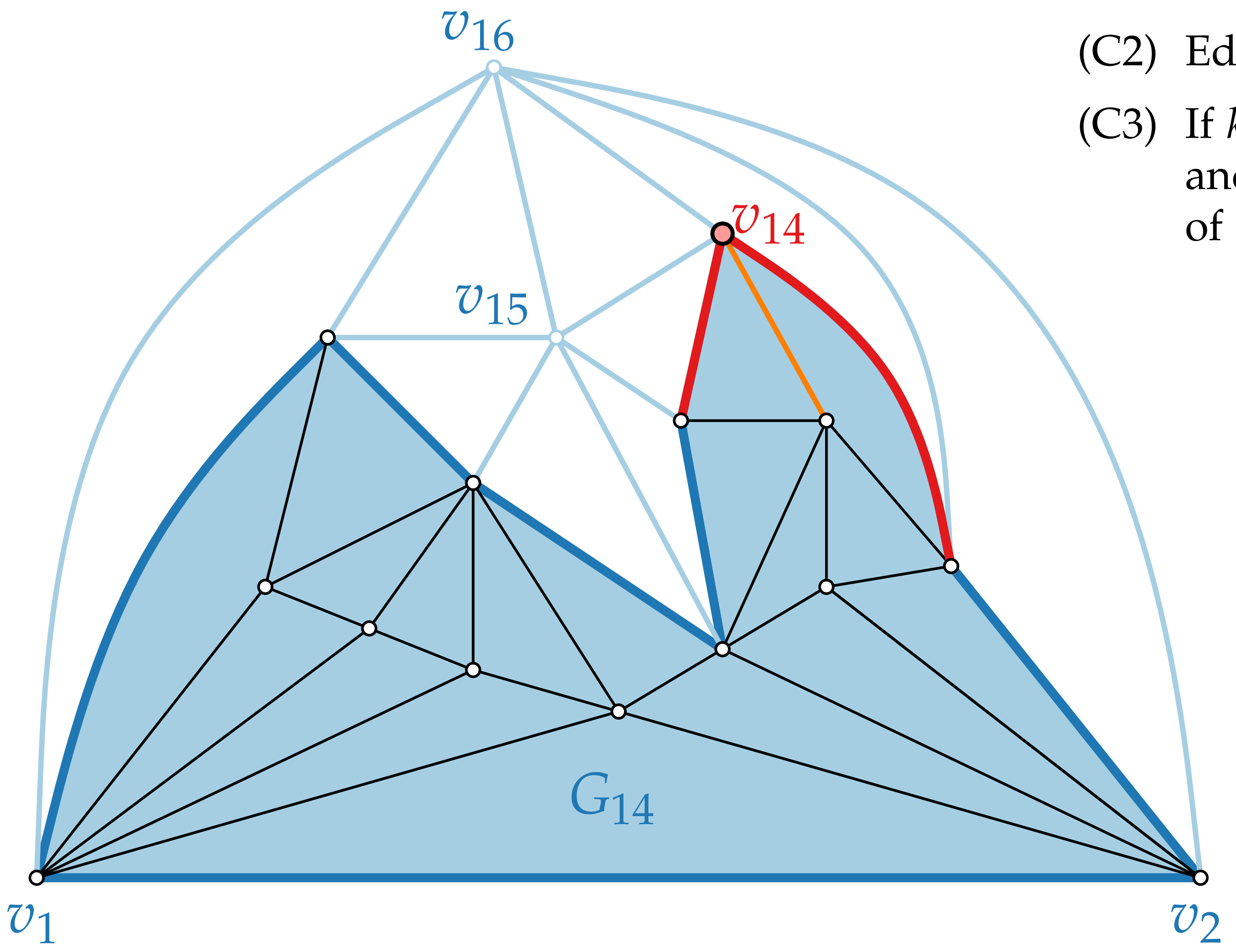


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

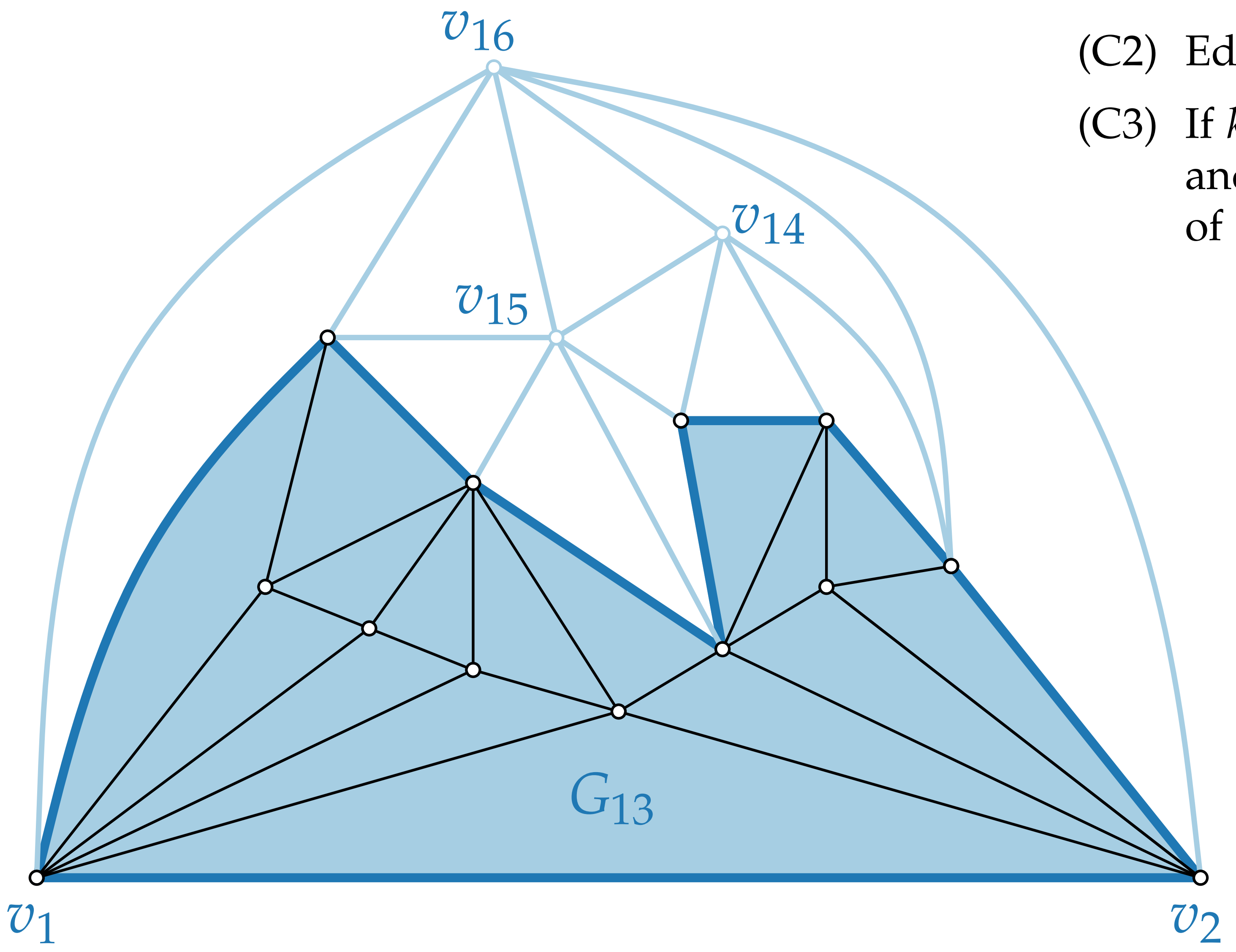


(C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

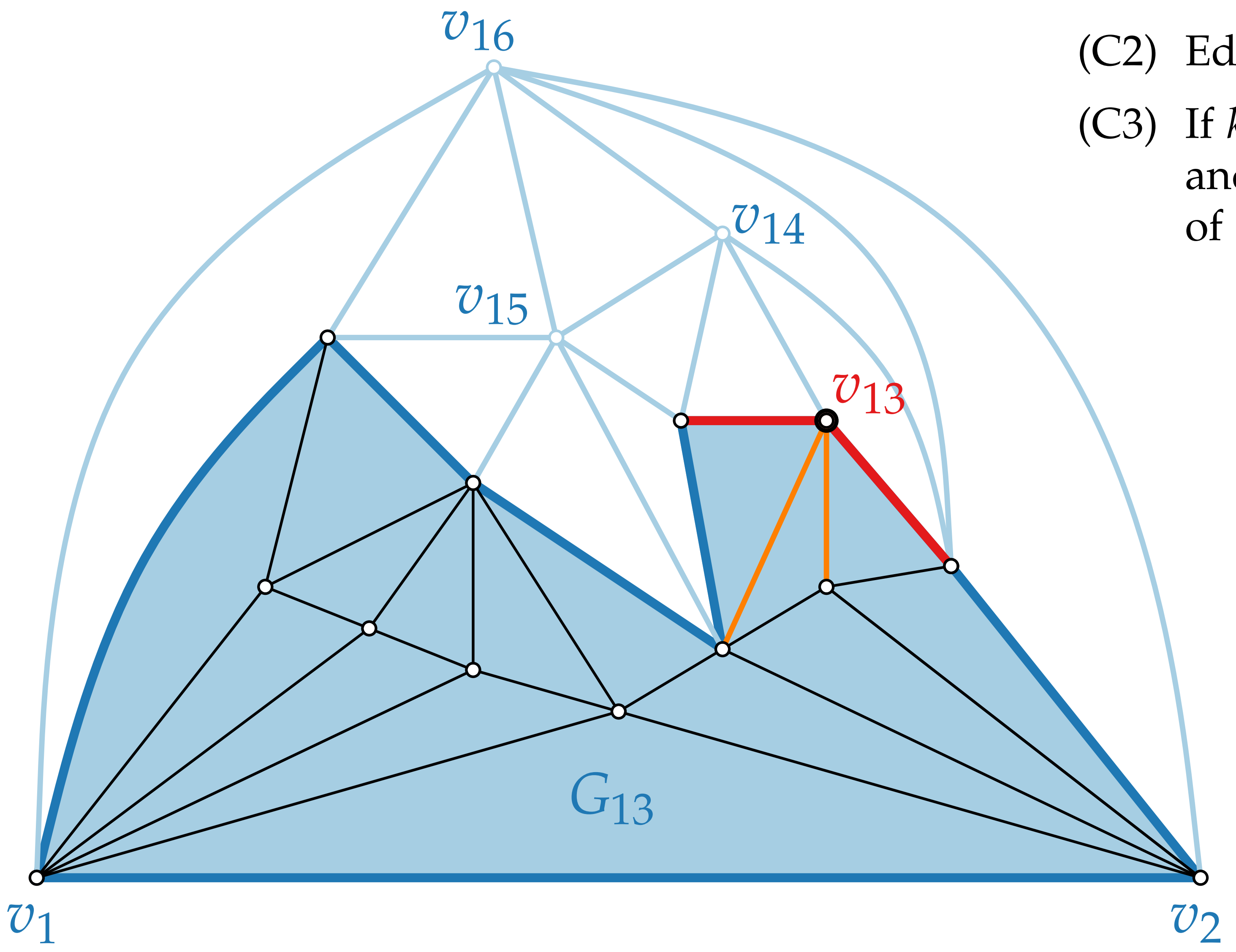


(C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

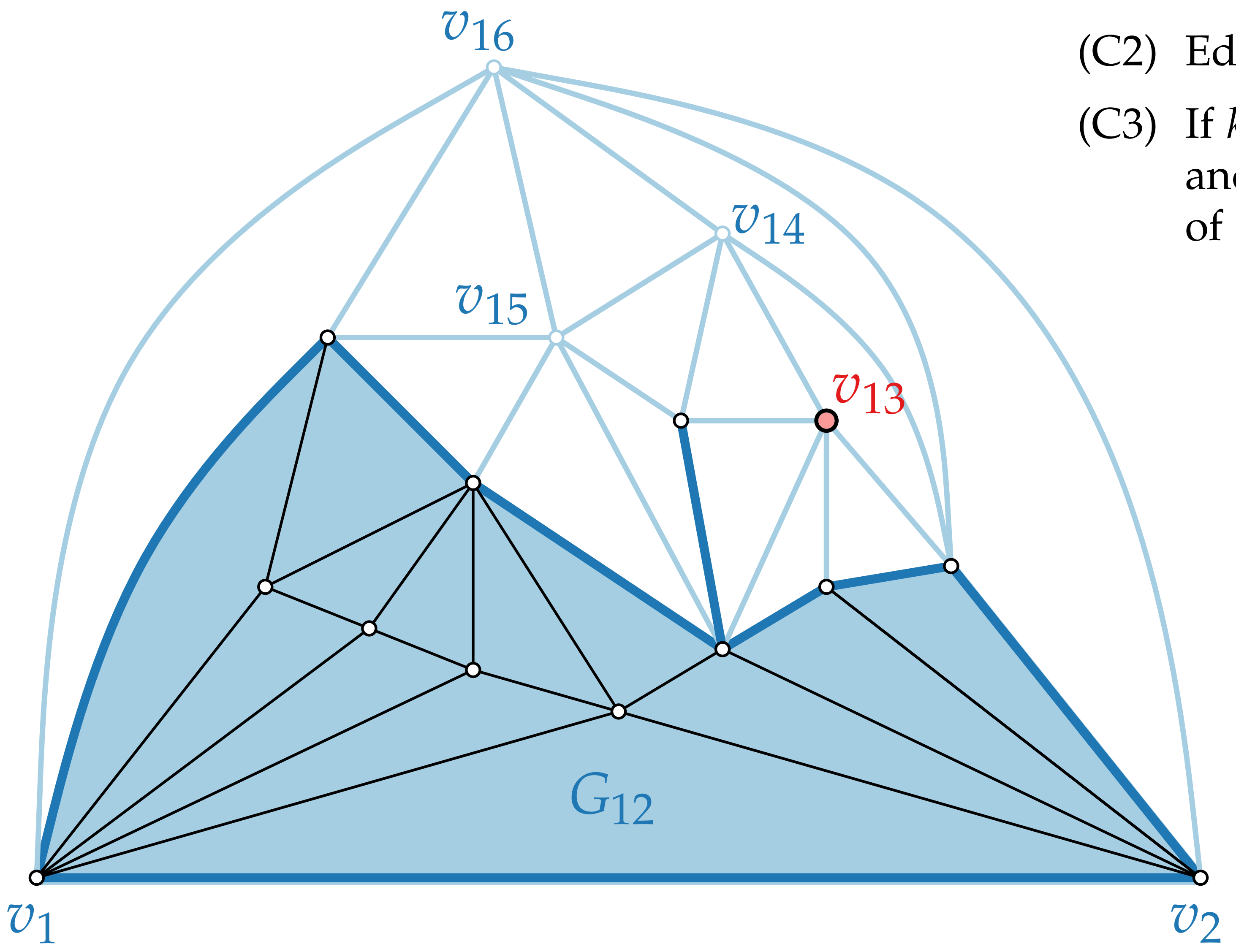


(C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

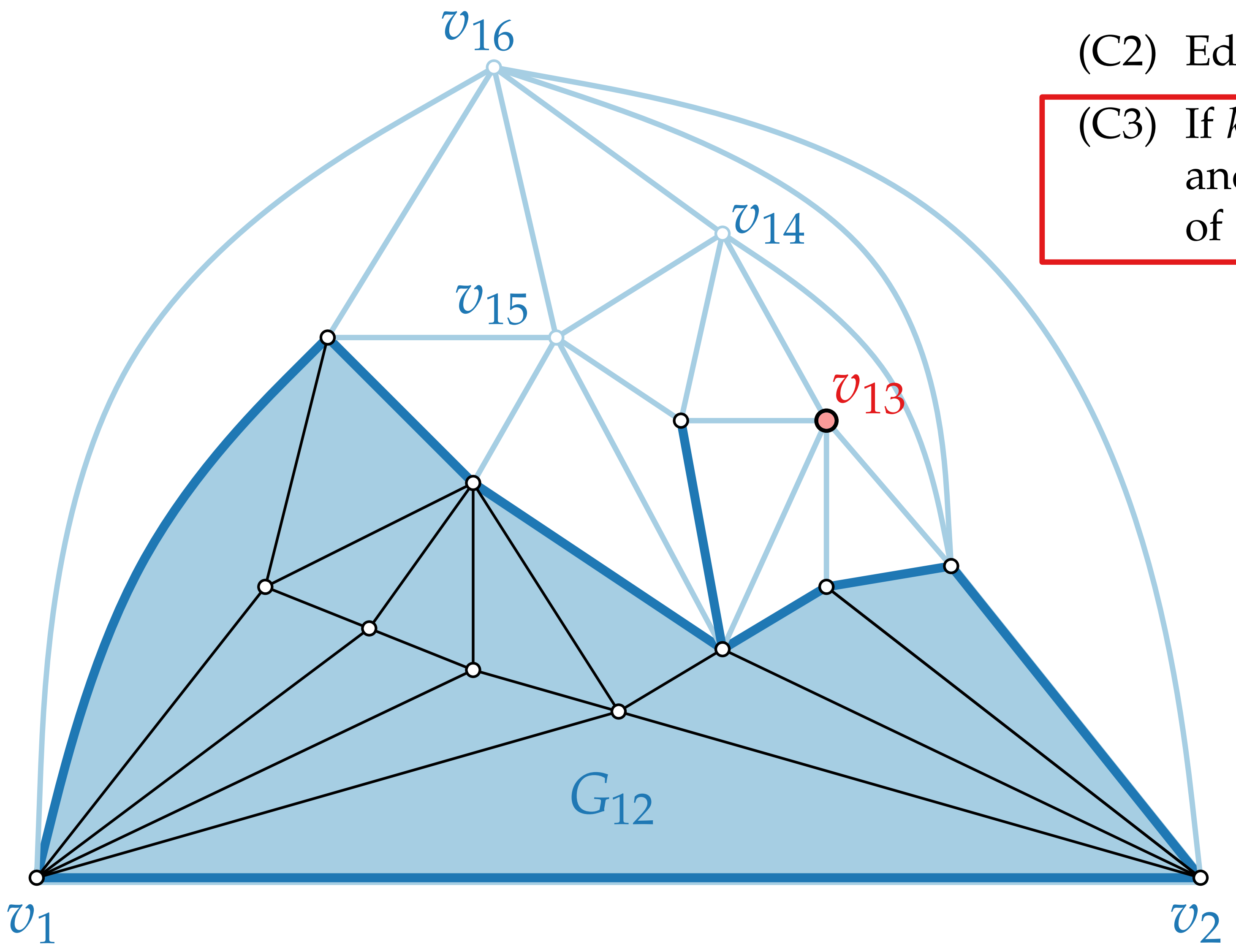


(C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

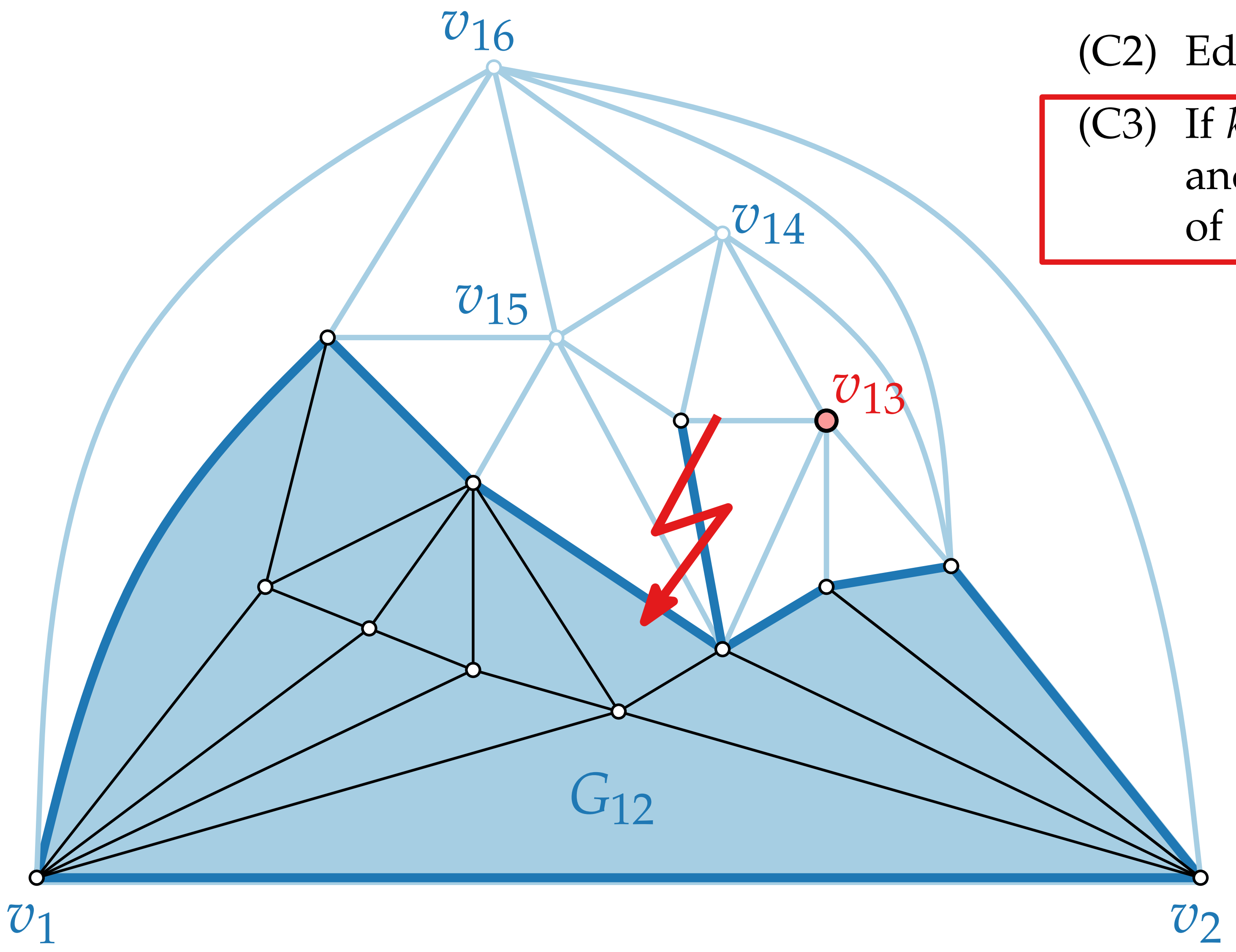


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

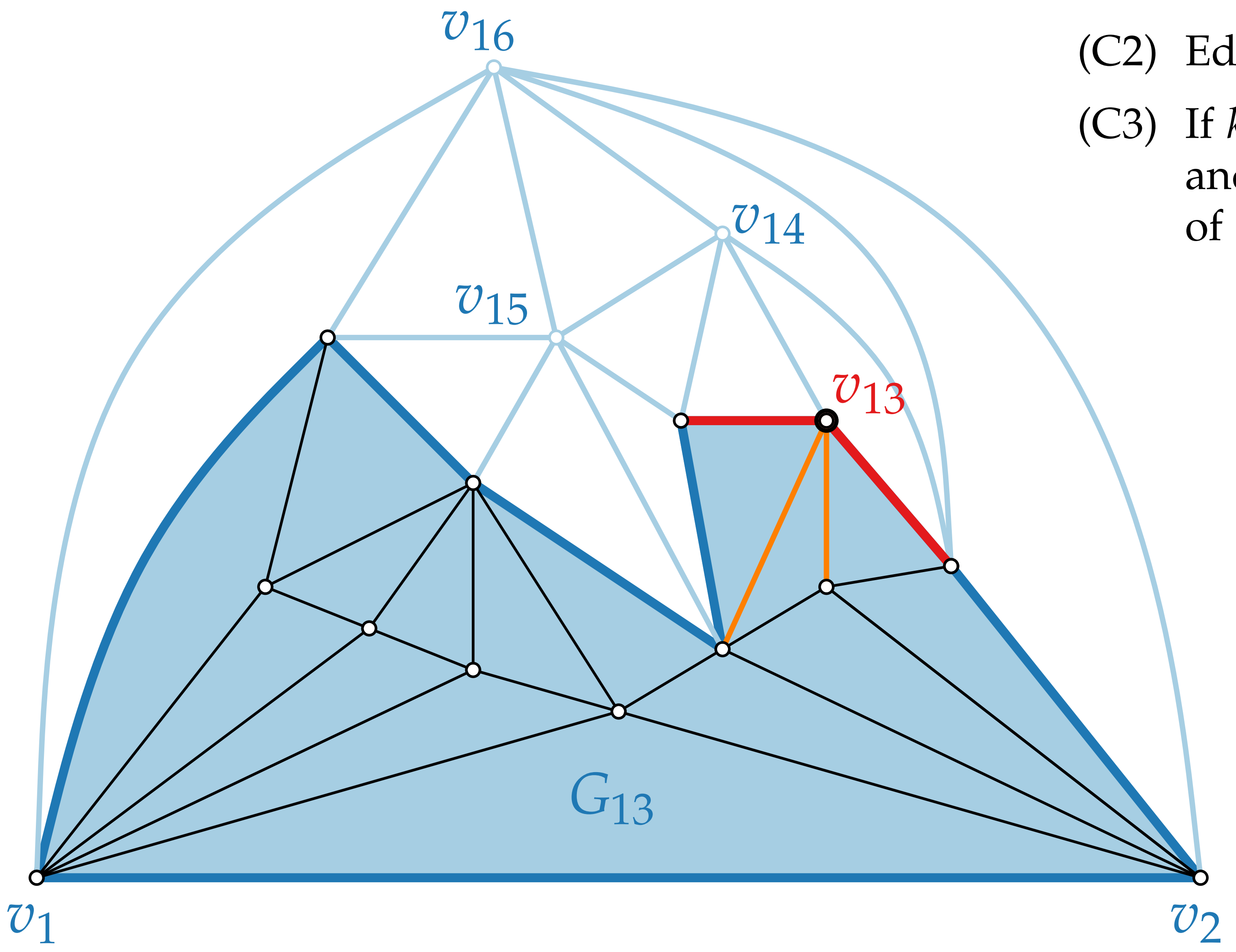


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

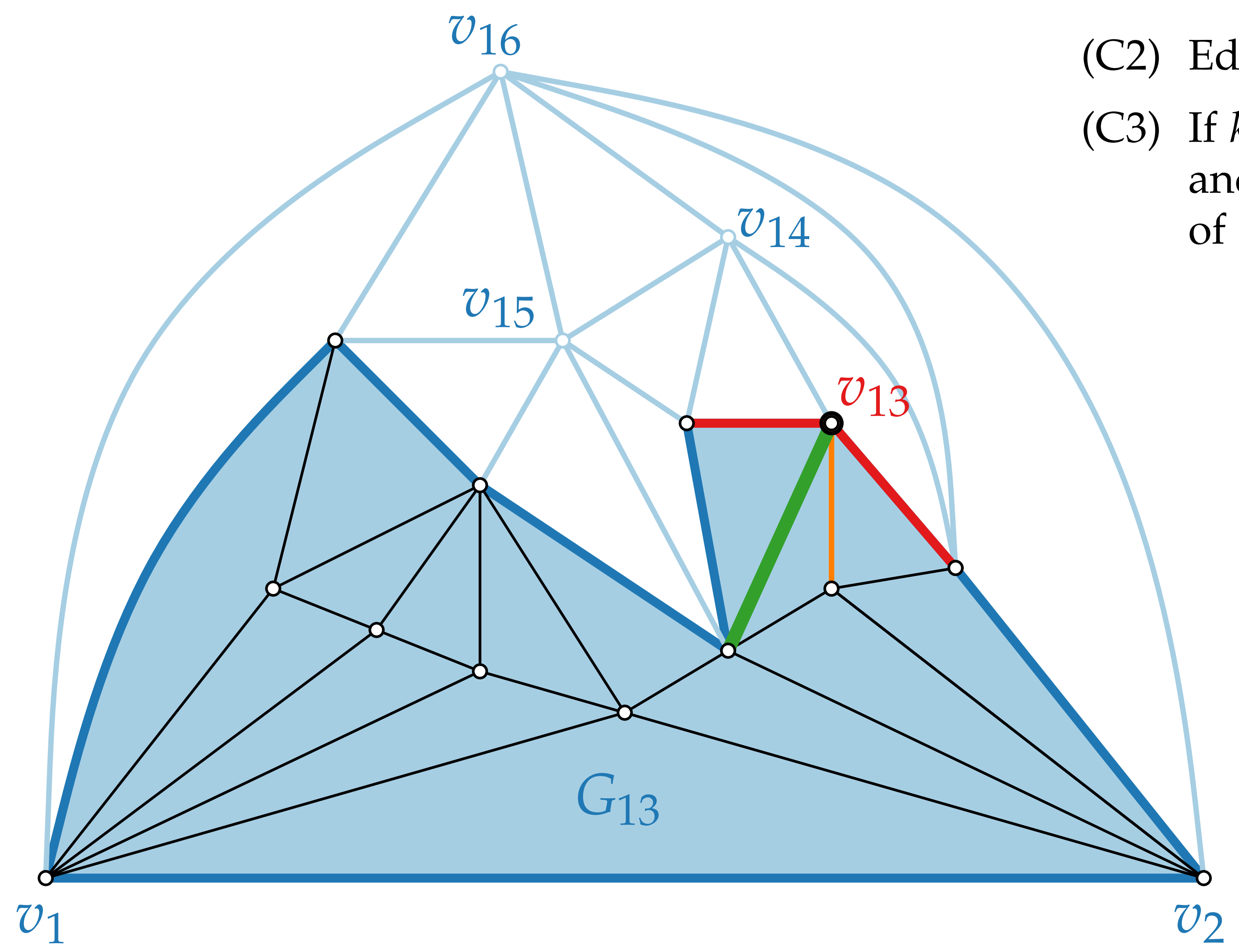


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

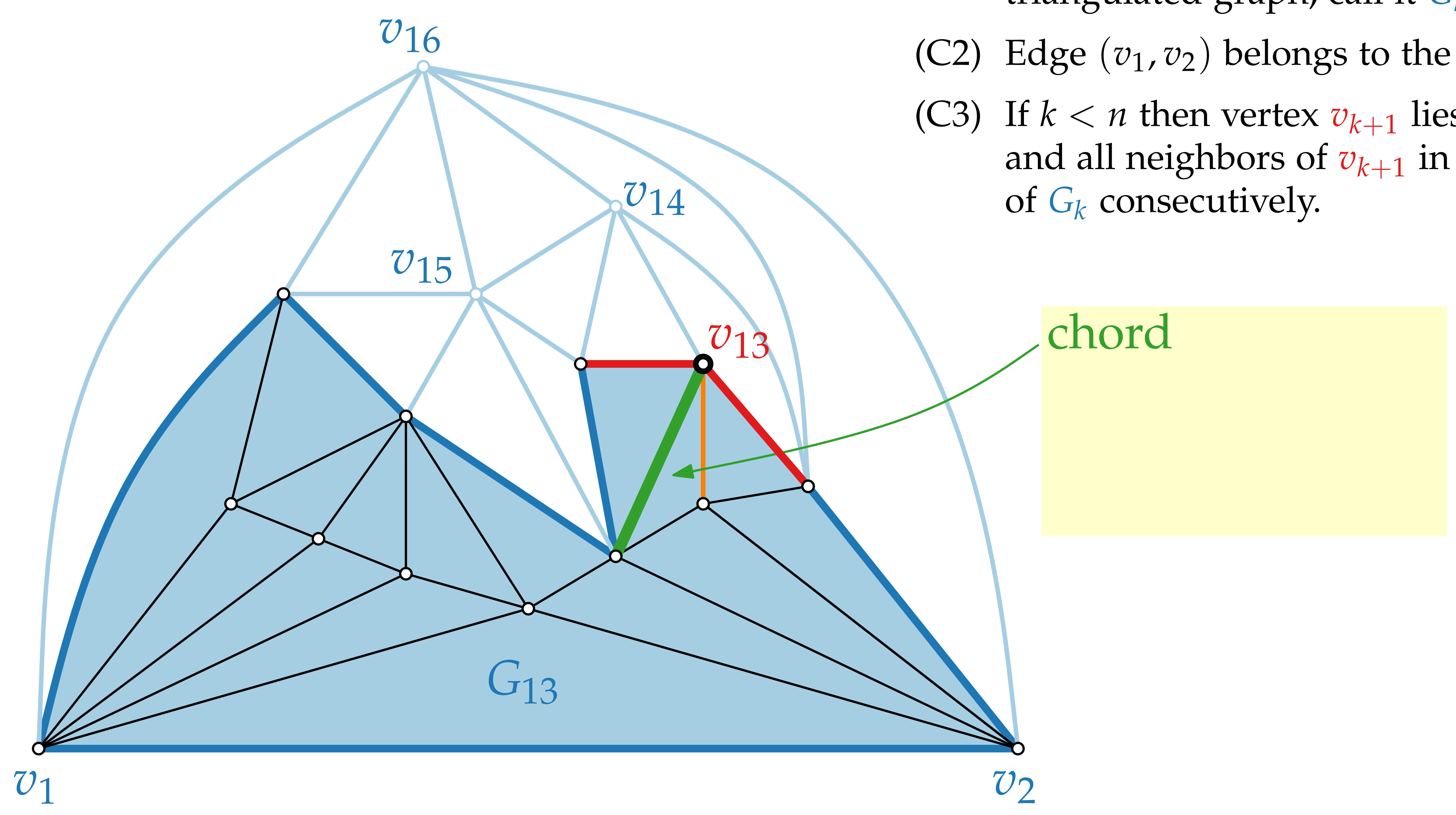


(C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

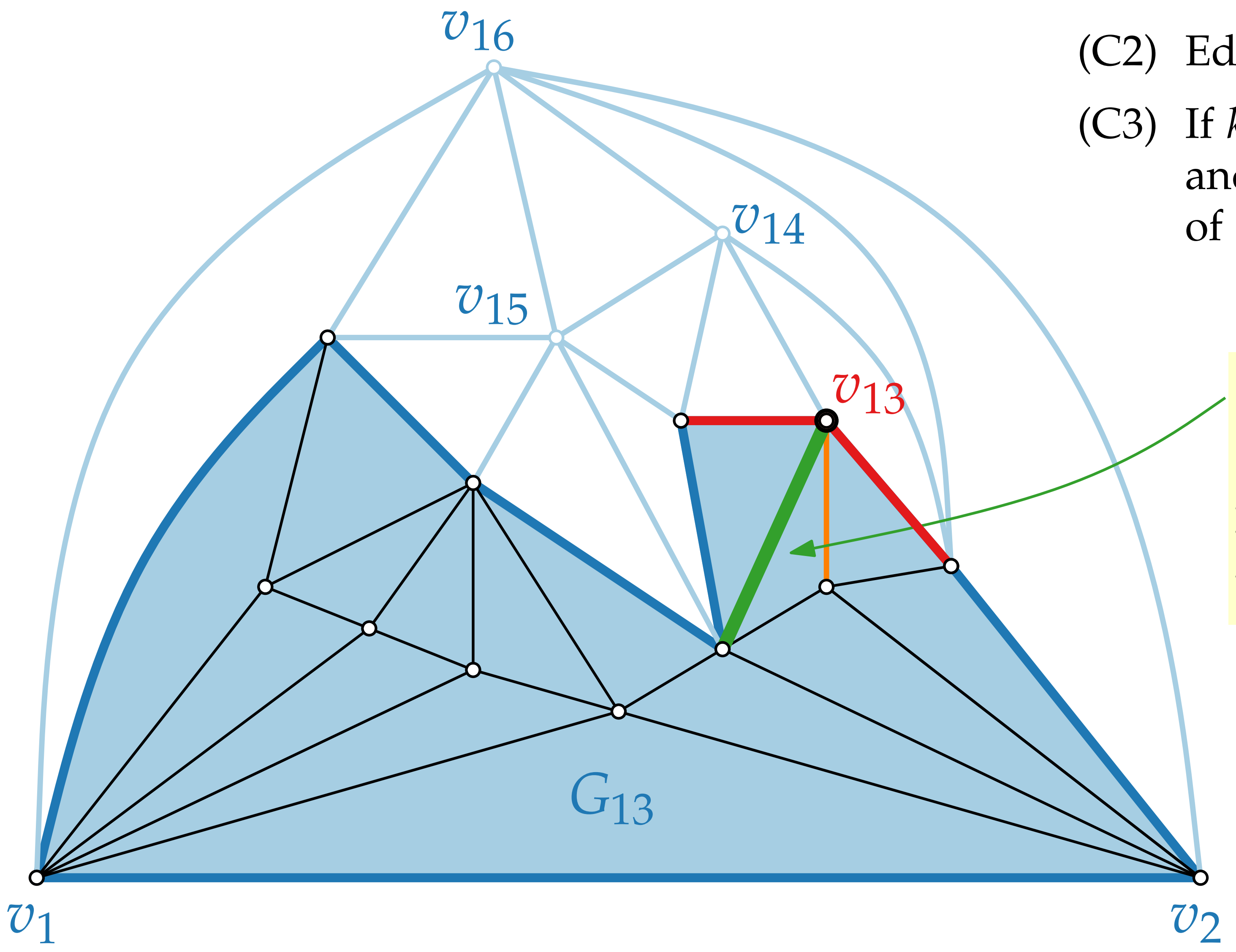


(C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example



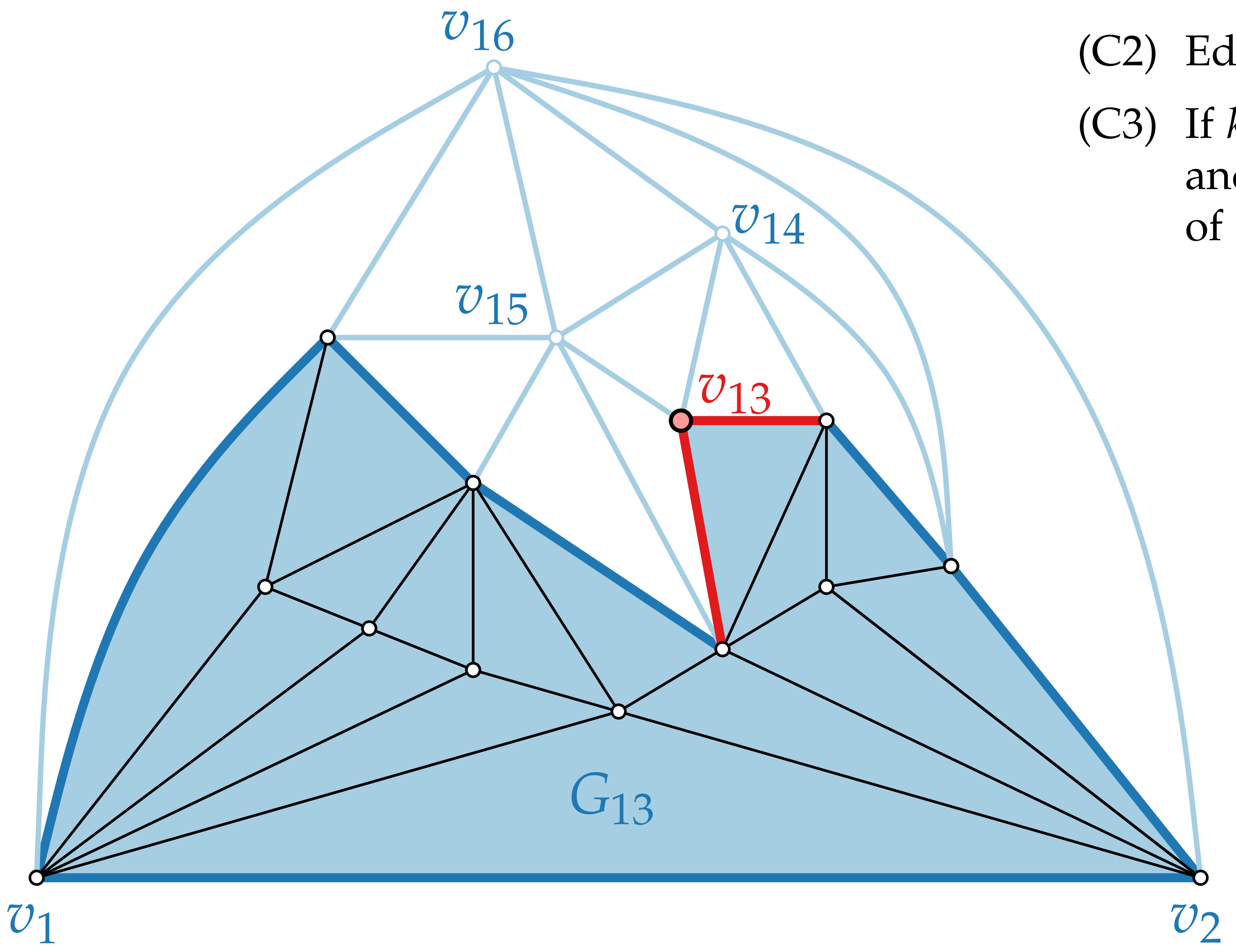
(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

chord
edge joining two nonadjacent vertices in a cycle

Canonical Order – Example

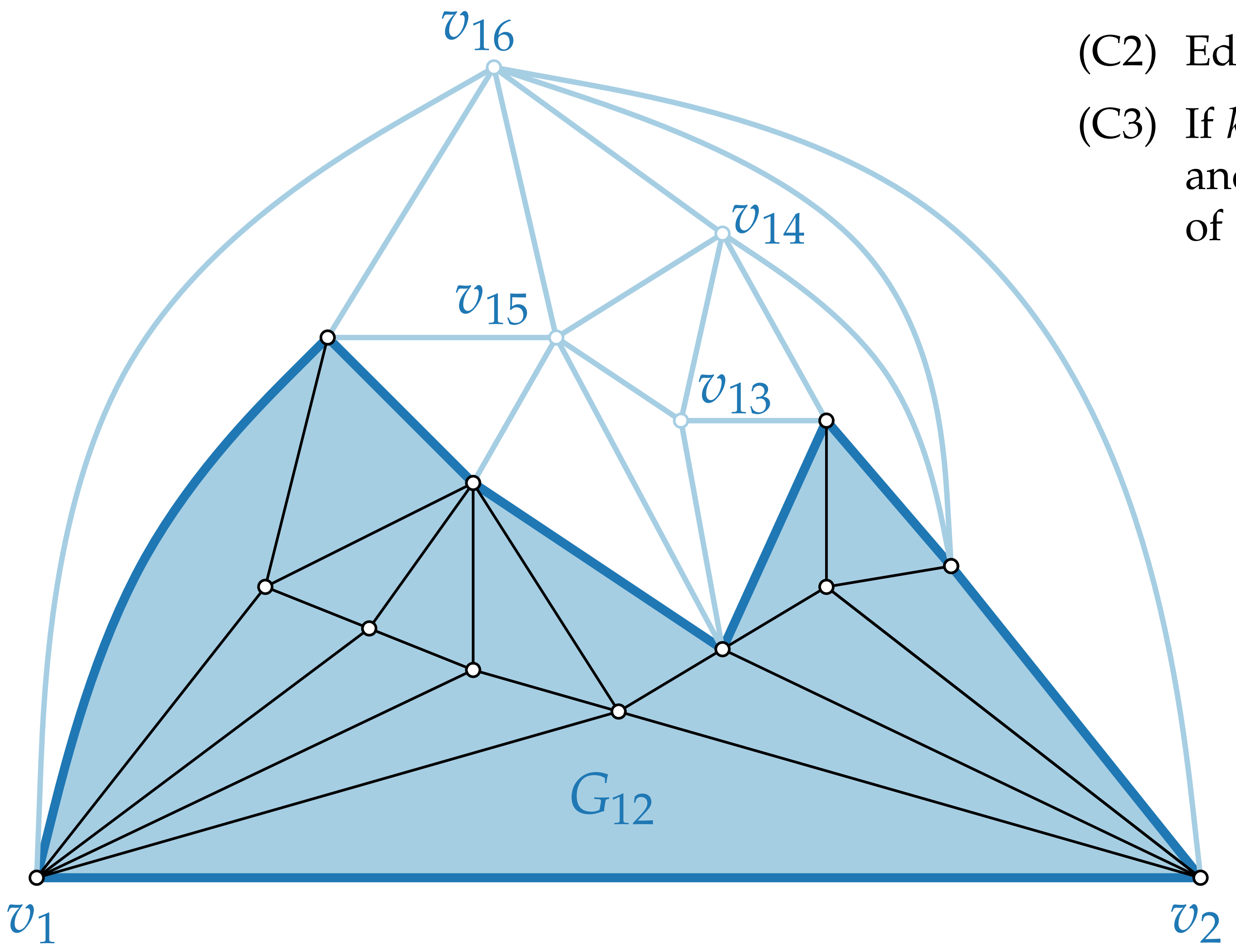


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

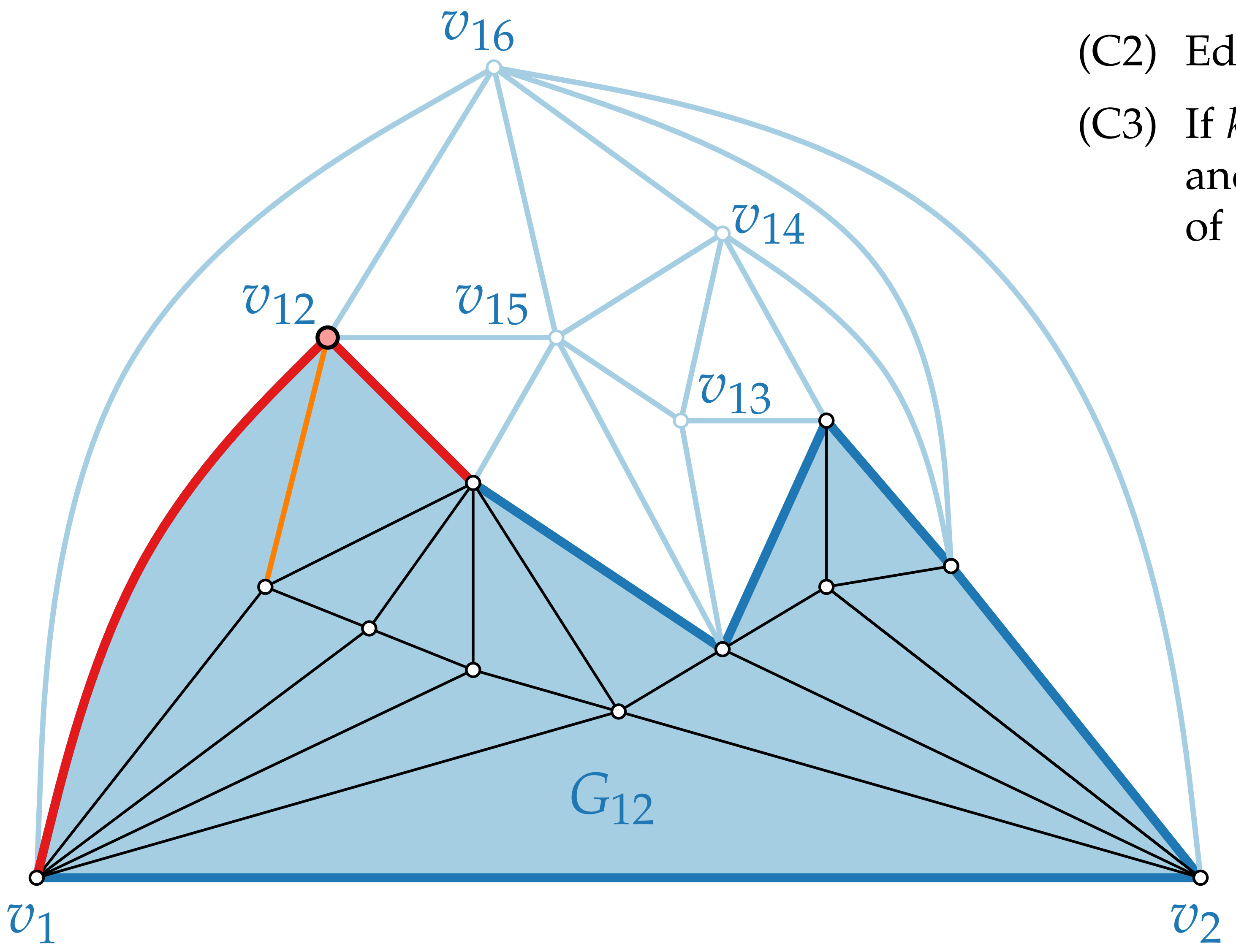


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

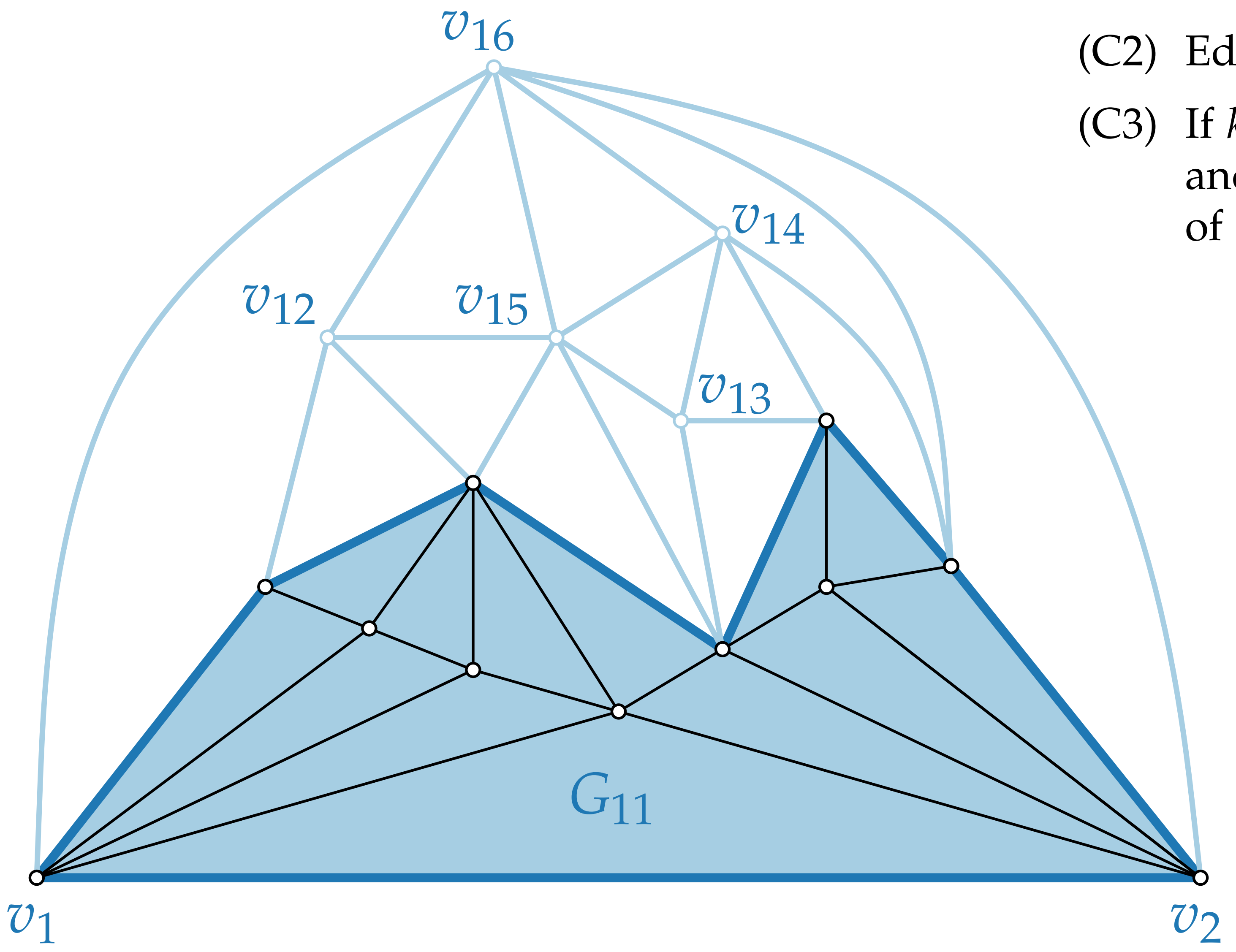


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

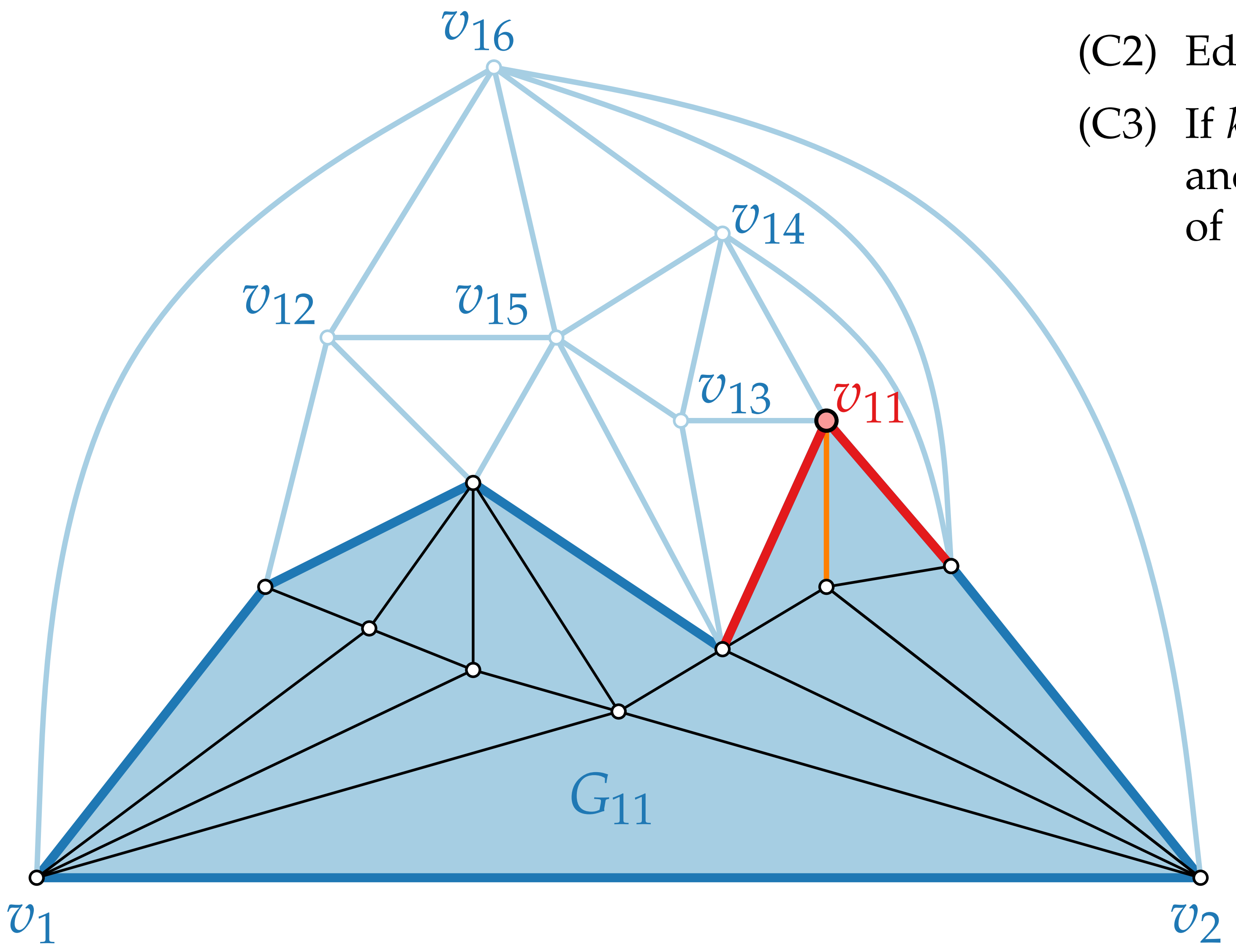


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

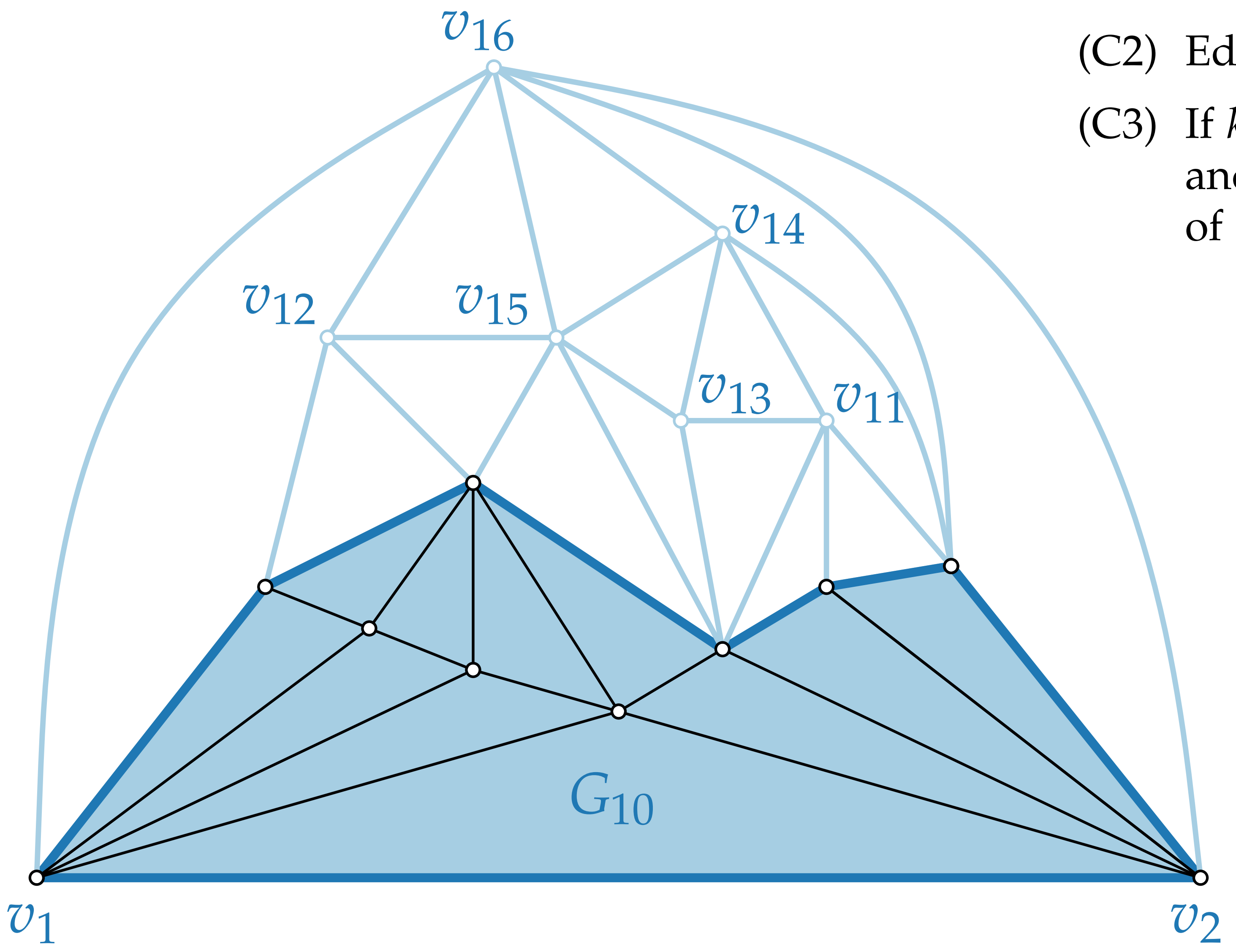


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

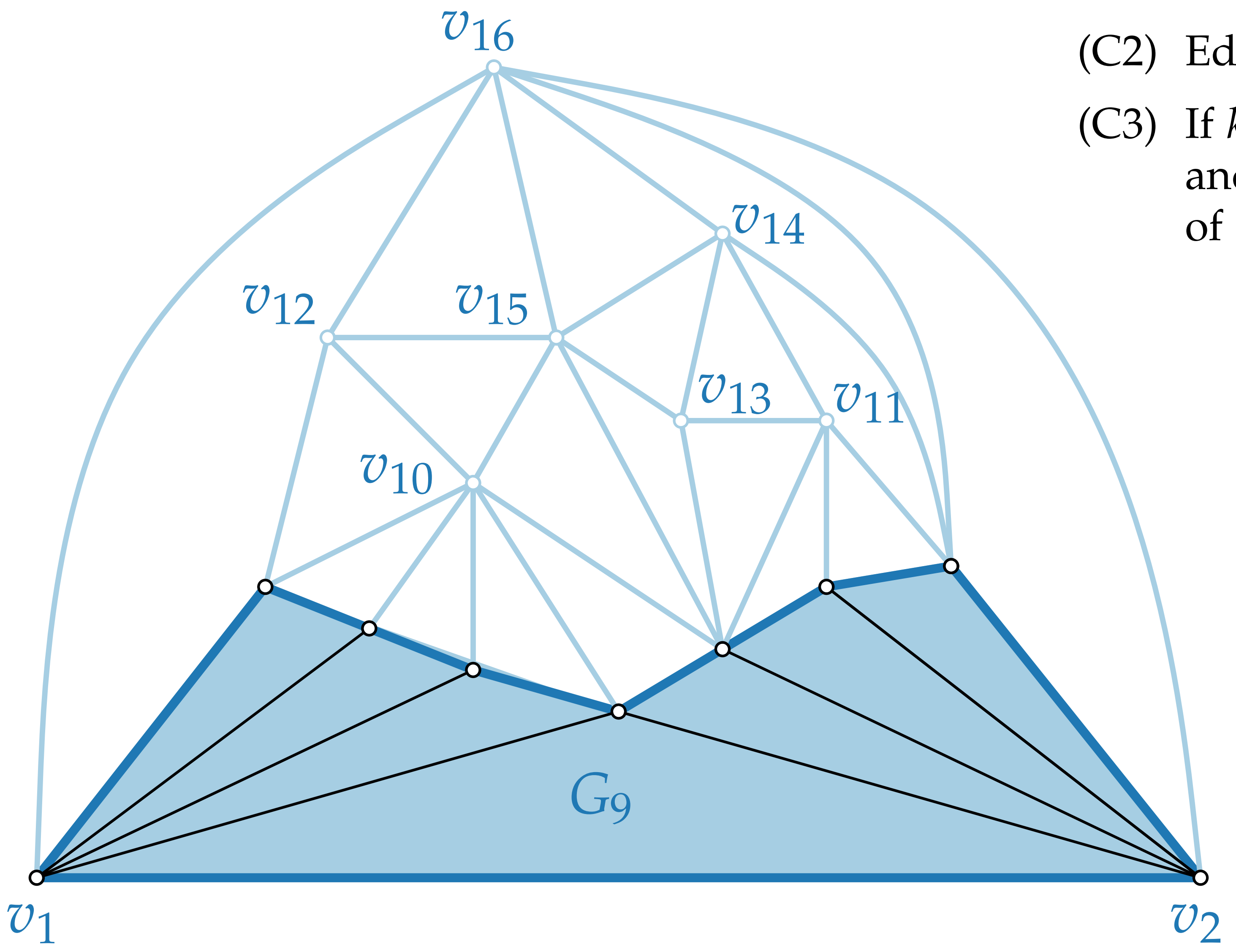


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

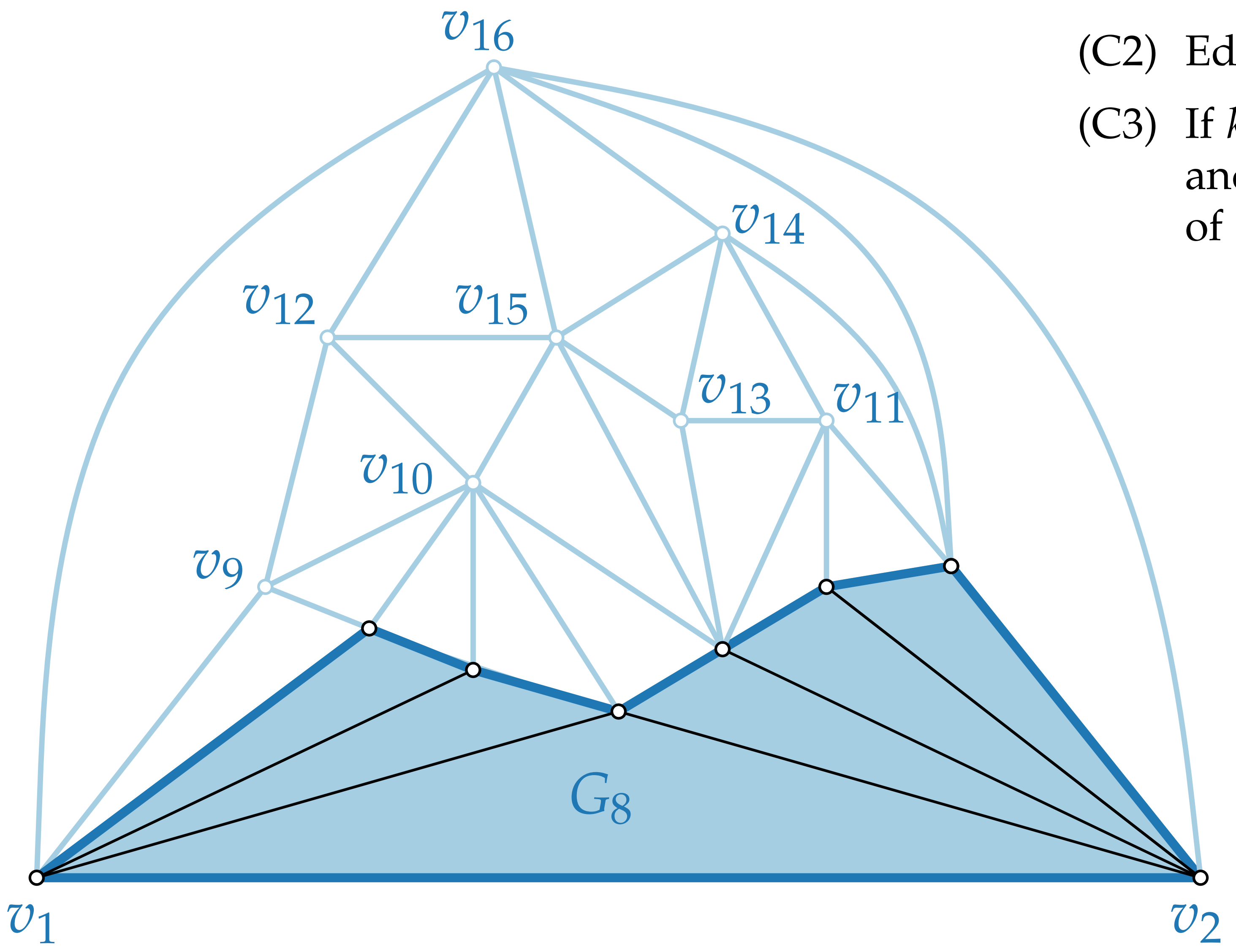


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

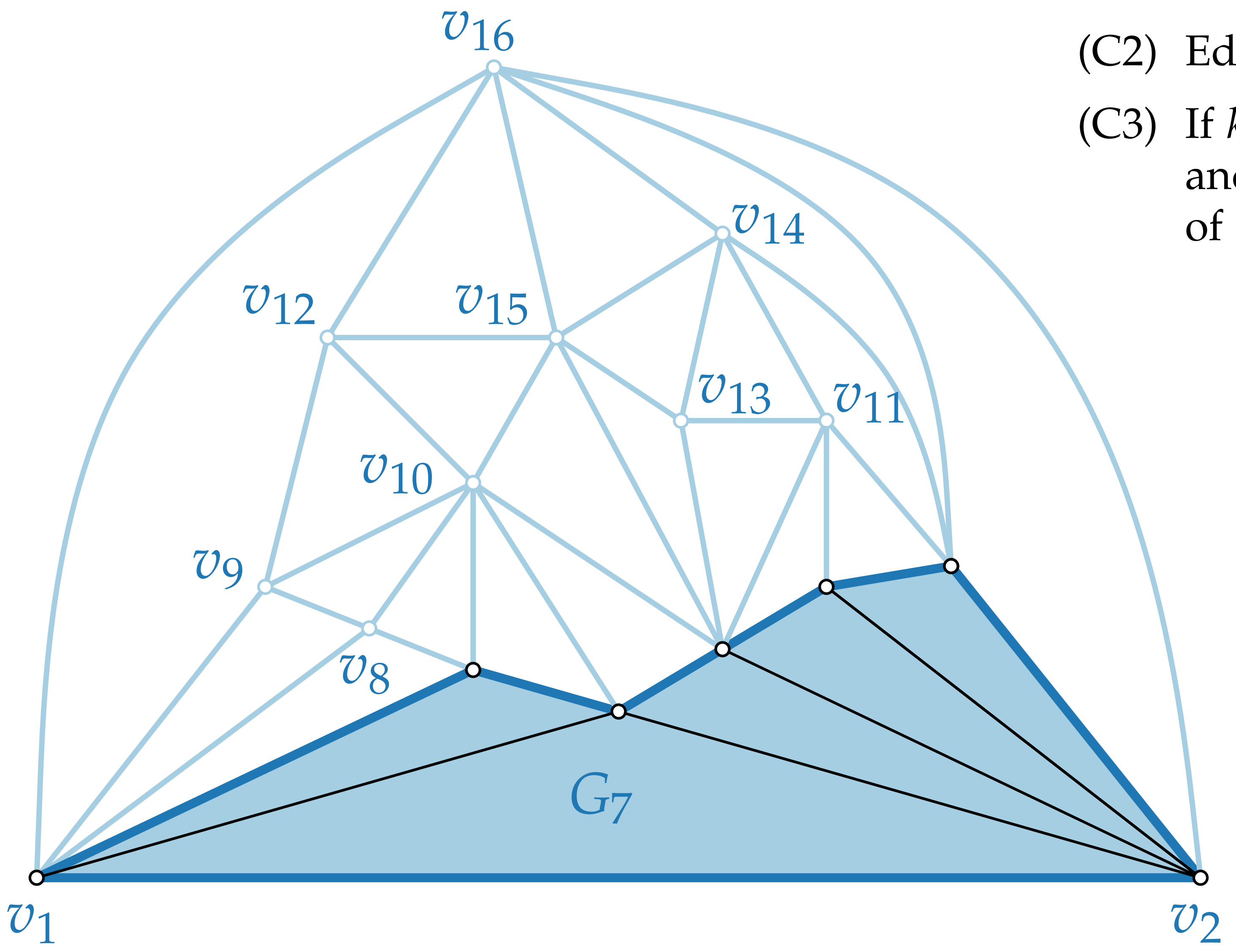


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

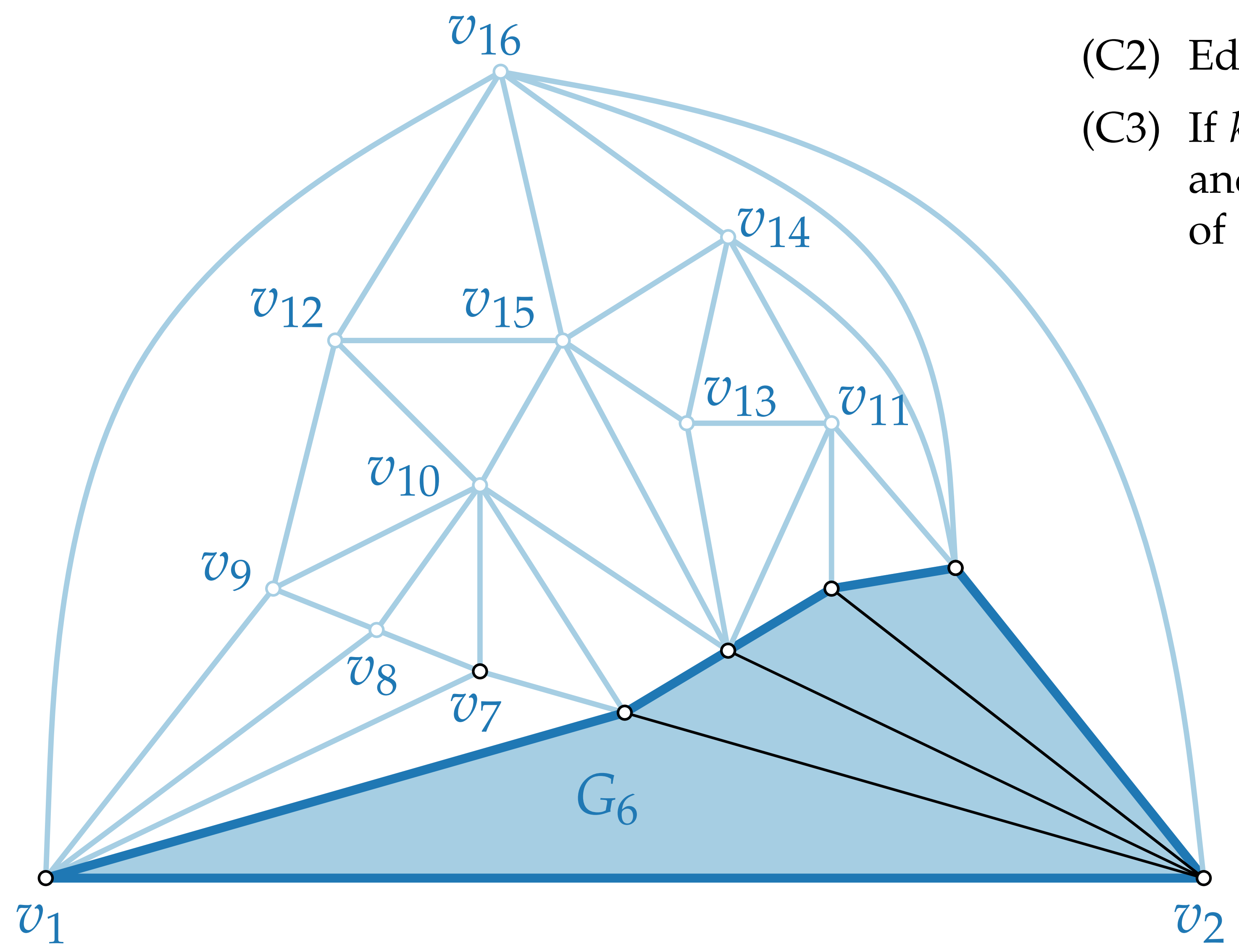


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

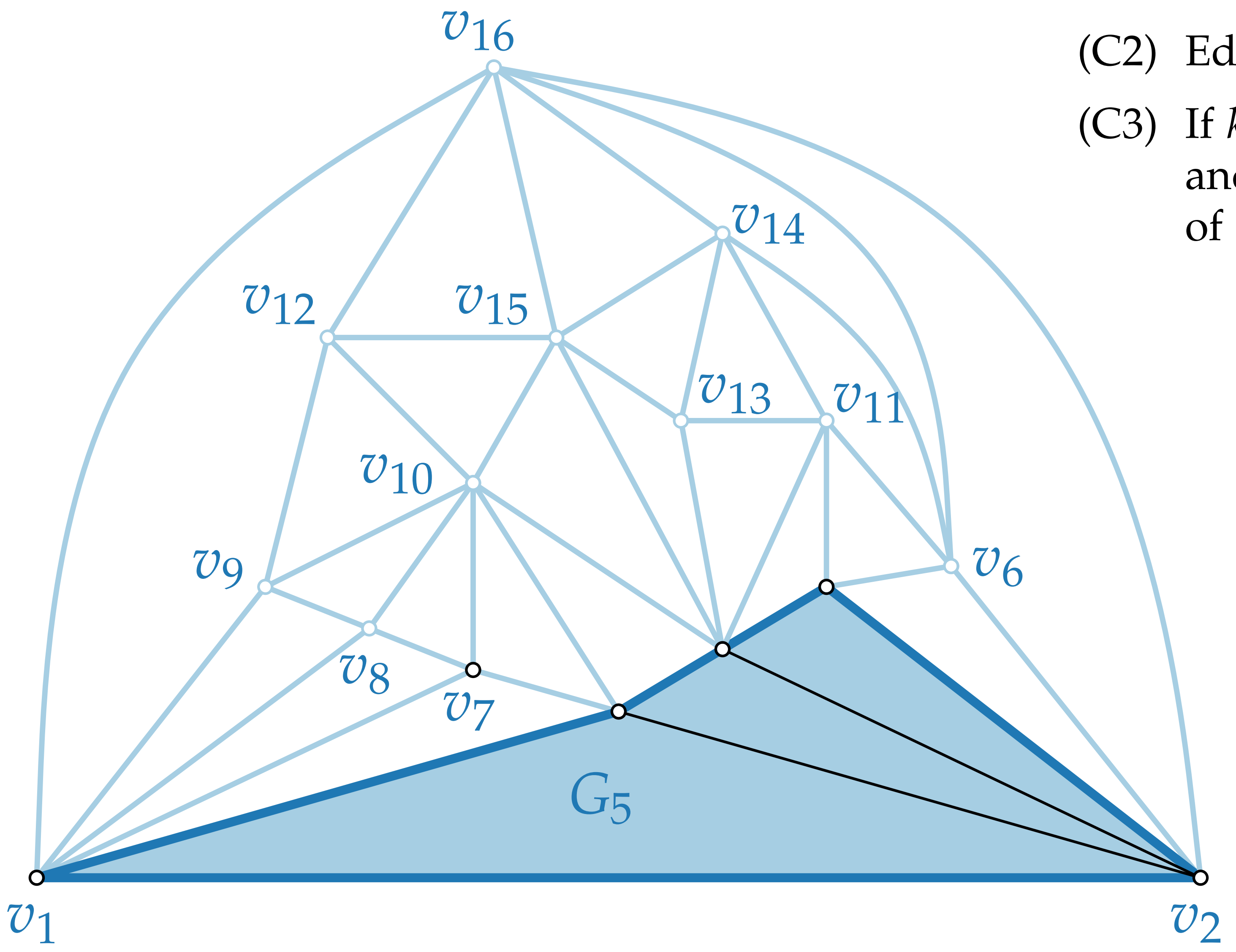


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

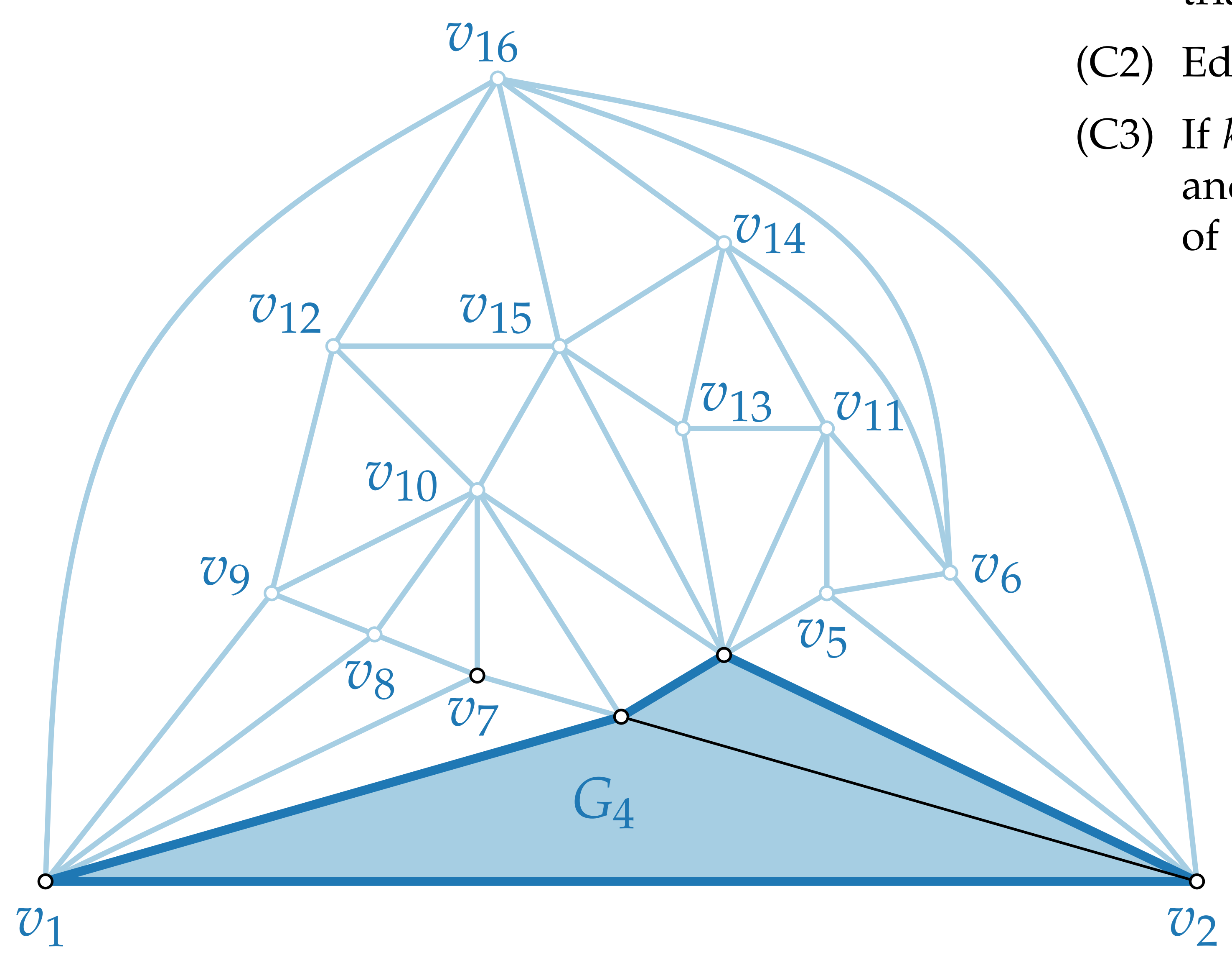


(C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

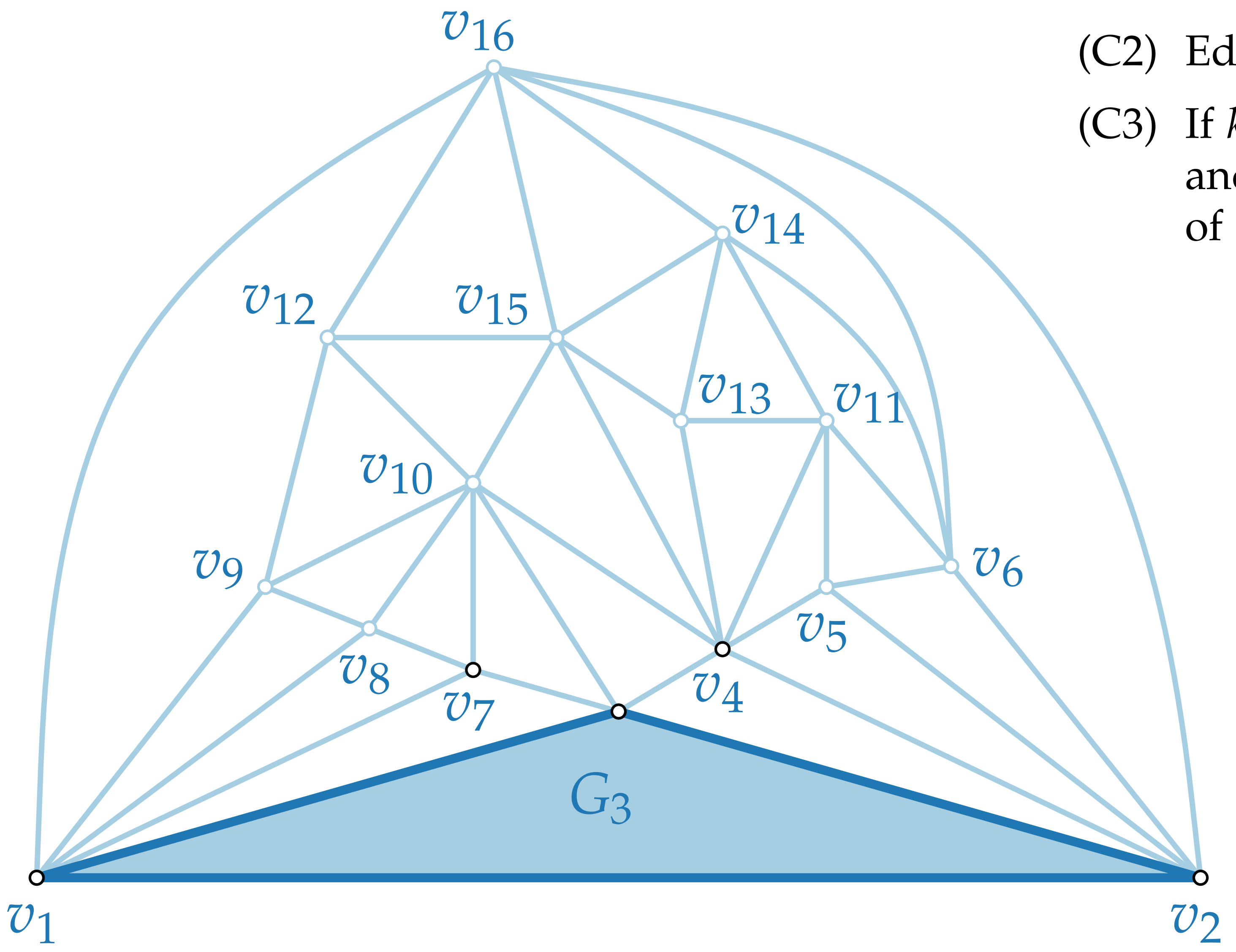


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

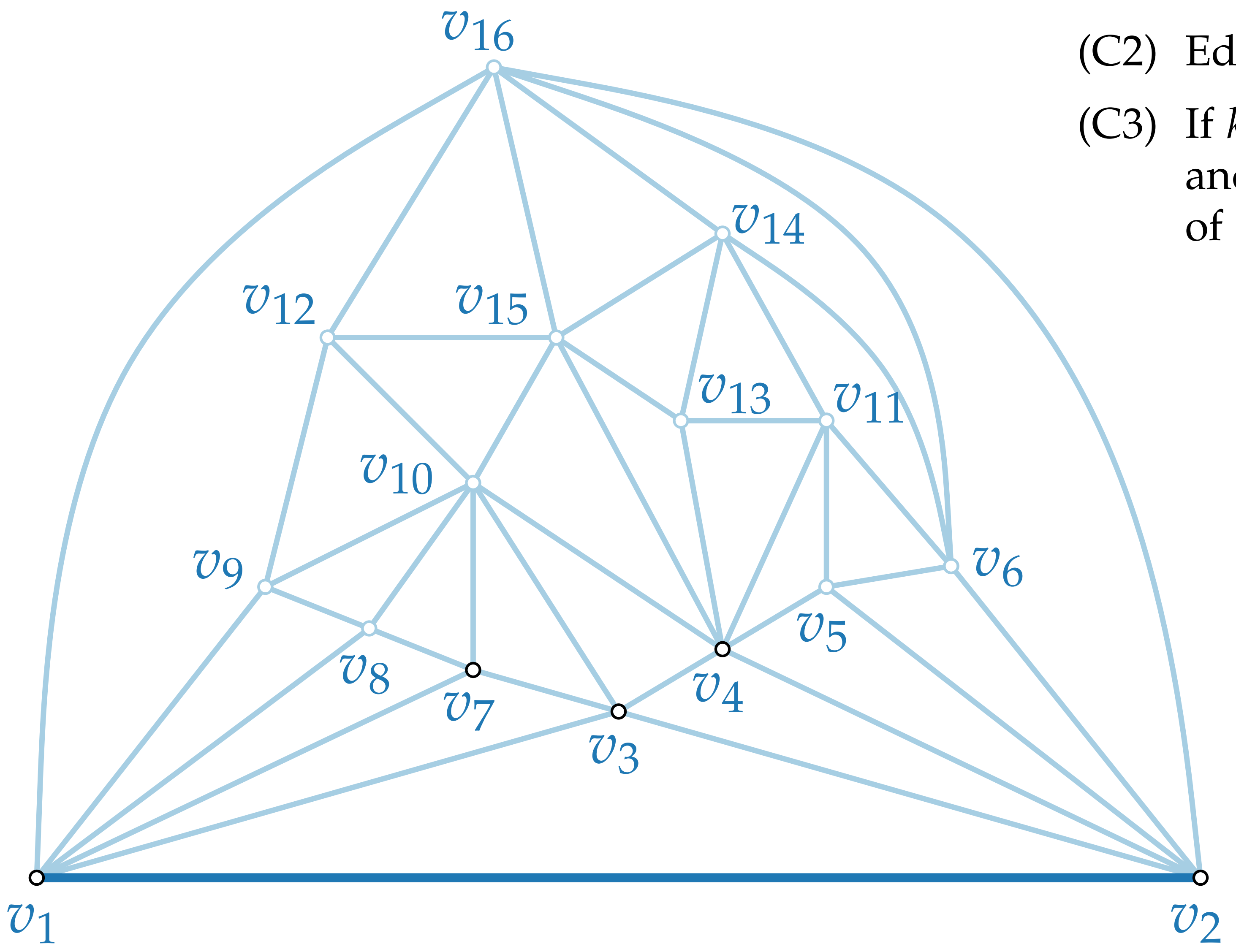


(C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

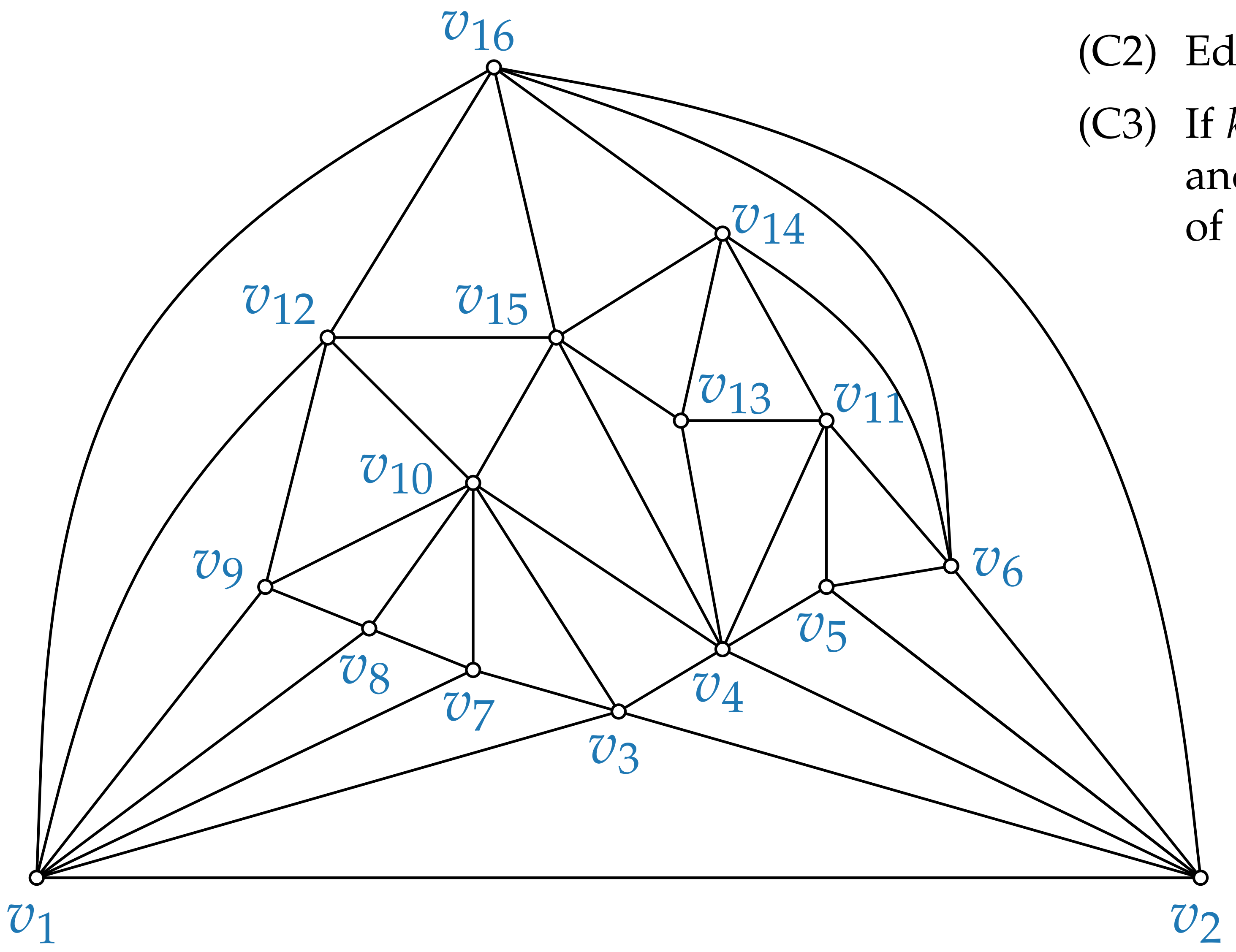


(C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

(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 all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Canonical Order – Example

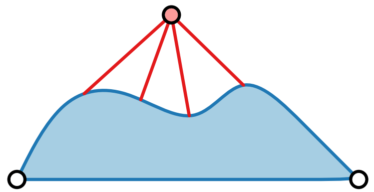


(C1) Vertices $\{v_1, \dots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.

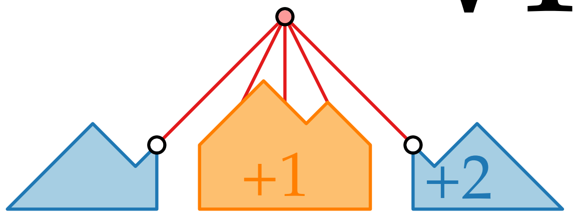
(C2) Edge (v_1, v_2) belongs to the outer face of G_k.

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

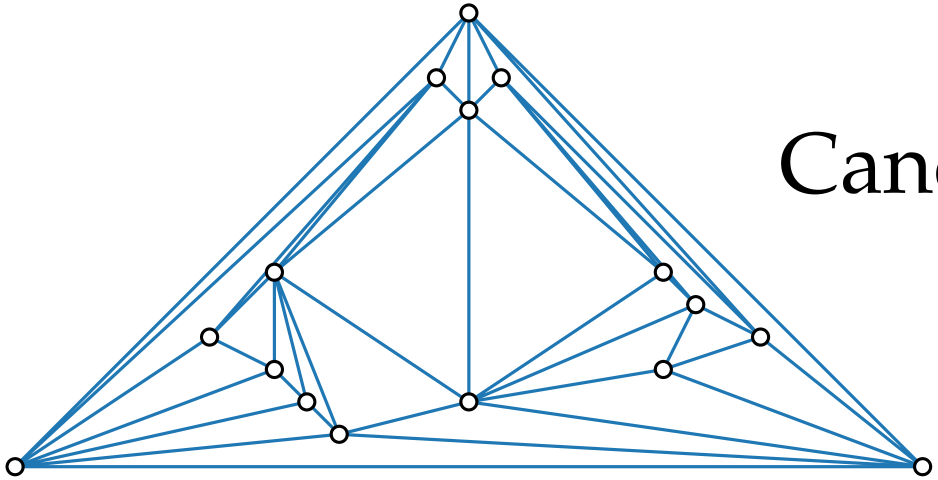
Universität Trier



Visualization of Graphs



Lecture 4:
Straight-Line Drawings of Planar Graphs I: Canonical Ordering and Shift Method



Part III:
Canonical Order – Existence

Philipp Kindermann

Canonical Order – Existence

Lemma.
Every triangulated plane graph has a canonical order.

(C1) G_k biconnected and internally triangulated

(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} in G_k consecutive on boundary

Canonical Order – Existence

Lemma.
Every triangulated plane graph has a canonical order.

(C1) G_k biconnected and internally triangulated

(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} in G_k consecutive on boundary

Base Case:

Induction hypothesis:

Induction step:

Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

(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} in G_k consecutive on boundary

Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n.

Induction hypothesis:

Induction step:

Canonical Order – Existence

Lemma.
Every triangulated plane graph has a canonical order.

(C1) G_k biconnected and internally triangulated ✓

(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} in G_k consecutive on boundary

Base Case:
Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n.

Induction hypothesis:

Induction step:

Canonical Order – Existence

Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n.

Induction hypothesis:

Induction step:

(C1) G_k biconnected and internally triangulated ✓

(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} in G_k consecutive on boundary

Canonical Order – Existence

(C1) G_k biconnected and internally triangulated ✓

(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} in G_k consecutive on boundary ✓

Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n.

Induction hypothesis:

Induction step:

Canonical Order – Existence

(C1) G_k biconnected and internally triangulated ✓

(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} in G_k consecutive on boundary ✓

Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
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:

Induction step:

Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

(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} in G_k consecutive on boundary

Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
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 $k + 1 \leq i \leq n$.

Induction step:

Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

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

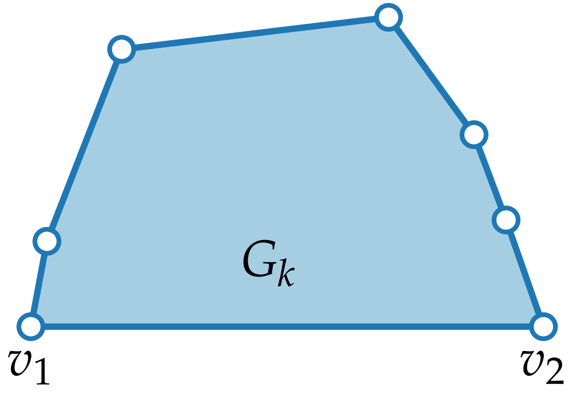
(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} in G_k consecutive on boundary

Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
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 $k + 1 \leq i \leq n$.

Induction step: Consider G_k.



Canonical Order – Existence

Lemma.
Every triangulated plane graph has a canonical order.

(C1) G_k biconnected and internally triangulated

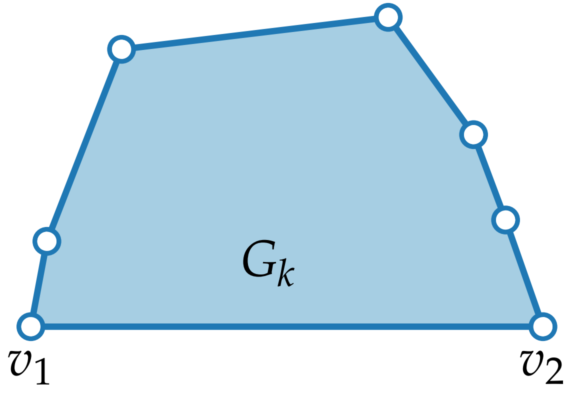
(C2) (v_1, v_2) on outer face of G_k

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

Base Case:
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 $k + 1 \leq i \leq n$.

Induction step: Consider G_k. We search for v_k.



Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

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

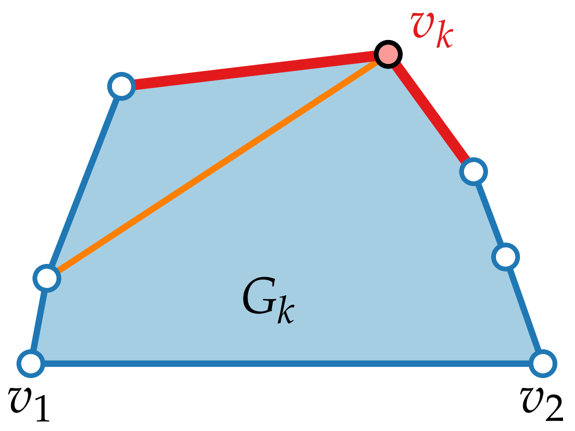
(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} in G_k consecutive on boundary

Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
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 $k + 1 \leq i \leq n$.

Induction step: Consider G_k. We search for v_k.



Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

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

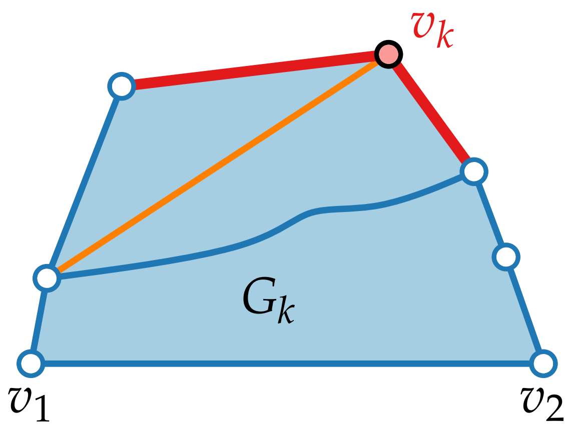
(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} in G_k consecutive on boundary

Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
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 $k + 1 \leq i \leq n$.

Induction step: Consider G_k. We search for v_k.



Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

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

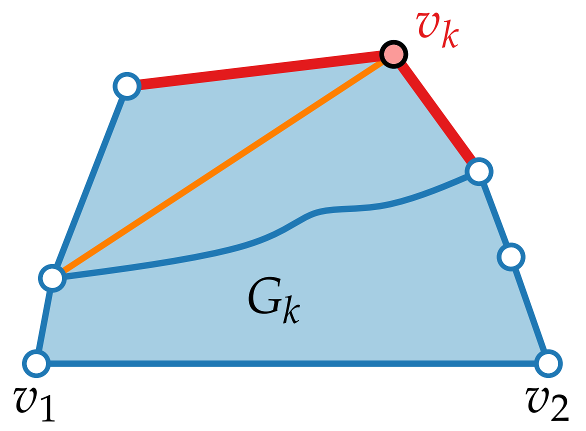
(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} in G_k consecutive on boundary

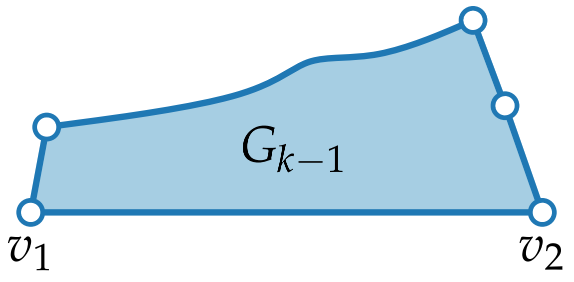
Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
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 $k + 1 \leq i \leq n$.

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

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

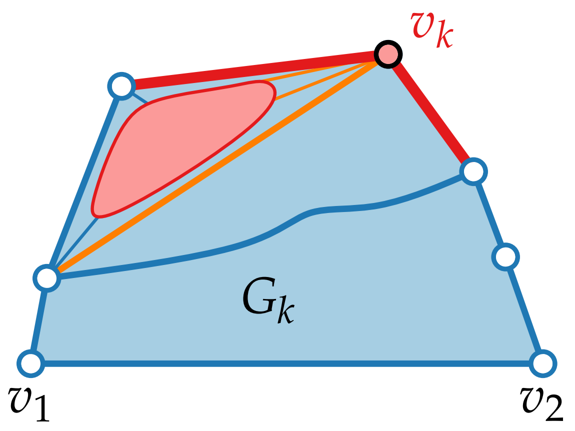
(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} in G_k consecutive on boundary

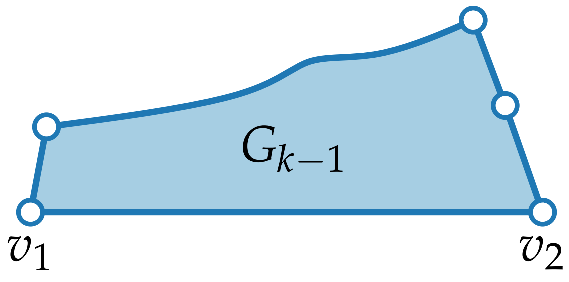
Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
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 $k + 1 \leq i \leq n$.

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

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

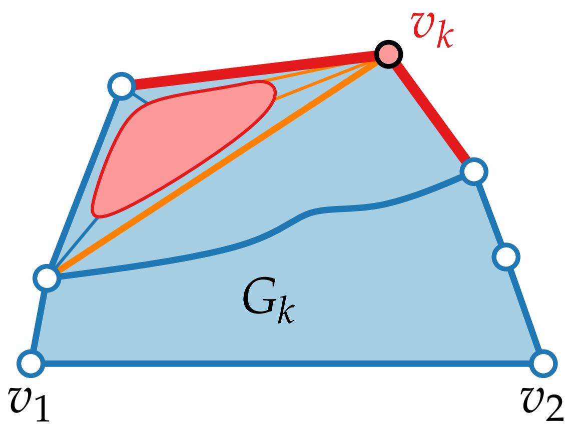
(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} in G_k consecutive on boundary

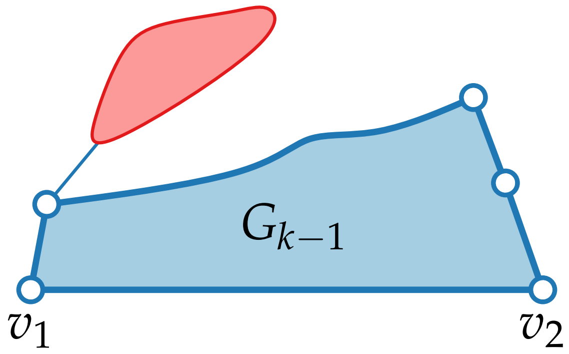
Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
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 $k + 1 \leq i \leq n$.

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

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

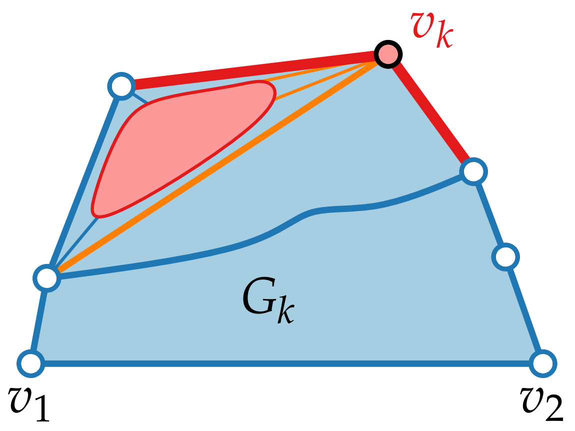
(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} in G_k consecutive on boundary

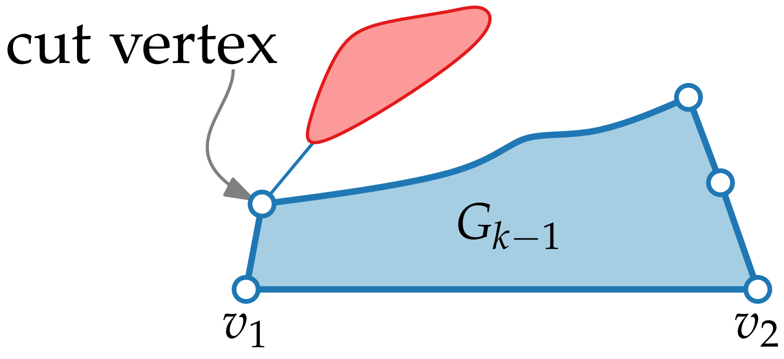
Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
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 $k + 1 \leq i \leq n$.

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

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

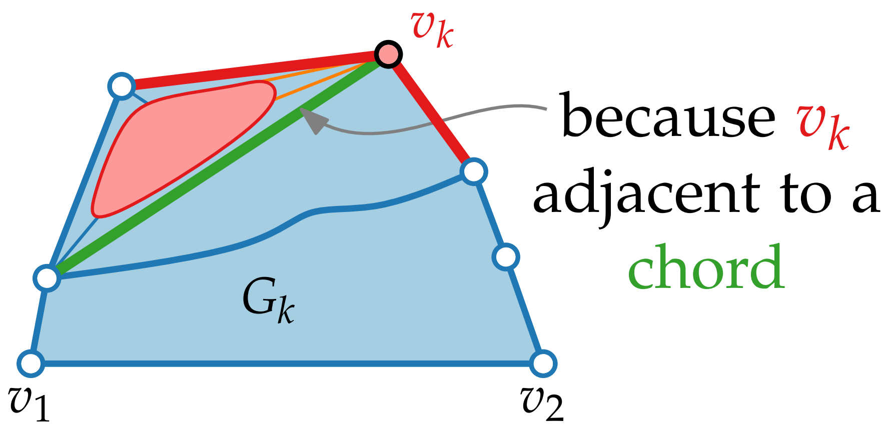
(C3) $k < n \Rightarrow v_{k+1}$ in outer face of G_k, neighbors of v_{k+1} in G_k consecutive on boundary

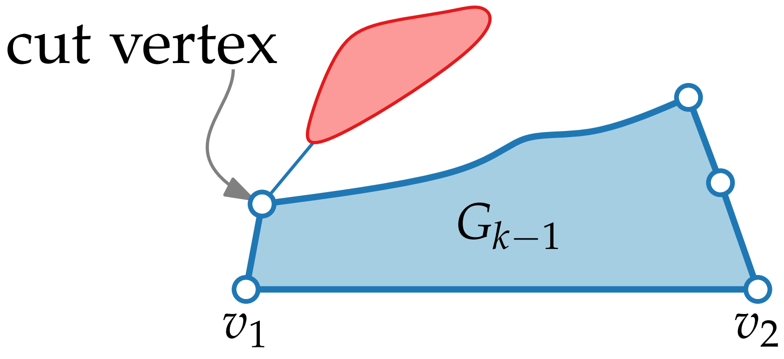
Lemma.
Every triangulated plane graph has a canonical order.

Base Case:
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 $k + 1 \leq i \leq n$.

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

(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} in G_k consecutive on boundary

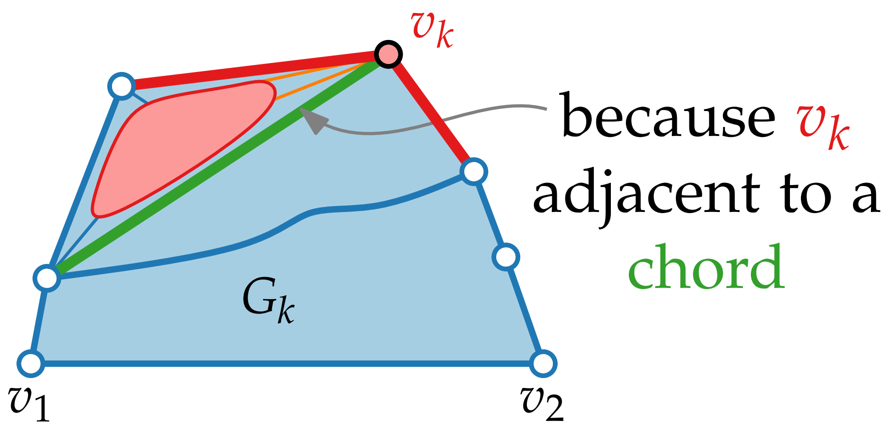
Lemma.
Every triangulated plane graph has a canonical order.

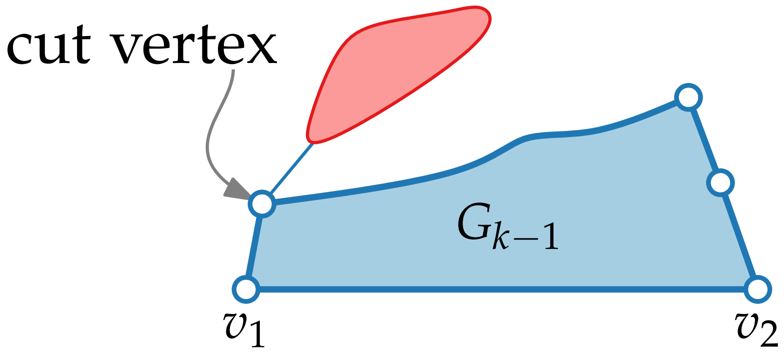
Base Case:
Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1) – (C3) hold.

Have to show:

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

Induction step: Consider G_k. We search for v_k.





Canonical Order – Existence

(C1) G_k biconnected and internally triangulated

(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} in G_k consecutive on boundary

Lemma.
Every triangulated plane graph has a canonical order.

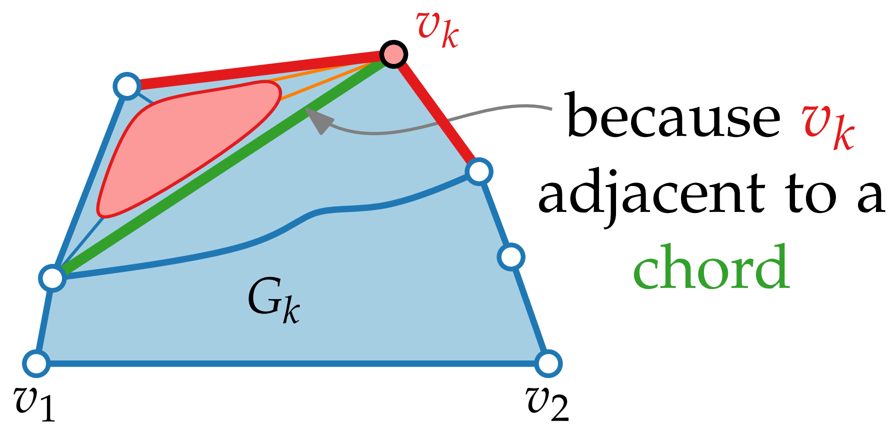
Base Case:
Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1) – (C3) hold.

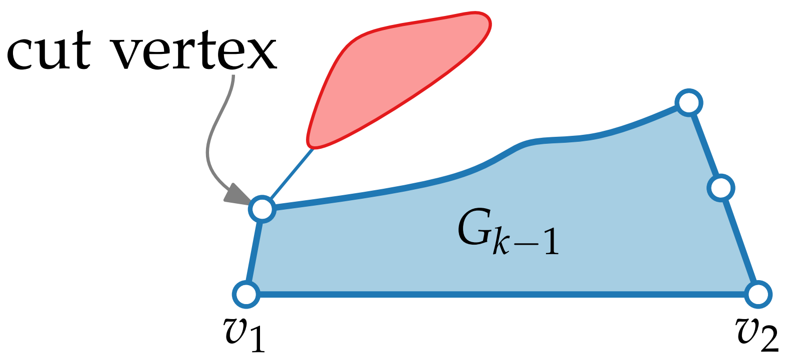
Induction hypothesis:
Vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions (C1) – (C3) hold for $k + 1 \leq i \leq n$.

Induction step: Consider G_k. We search for v_k.

Have to show:

1. v_k not adjacent to chord is sufficient





Canonical Order – Existence

Lemma.
Every triangulated plane graph has a canonical order.

(C1) G_k biconnected and internally triangulated

(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} in G_k consecutive on boundary

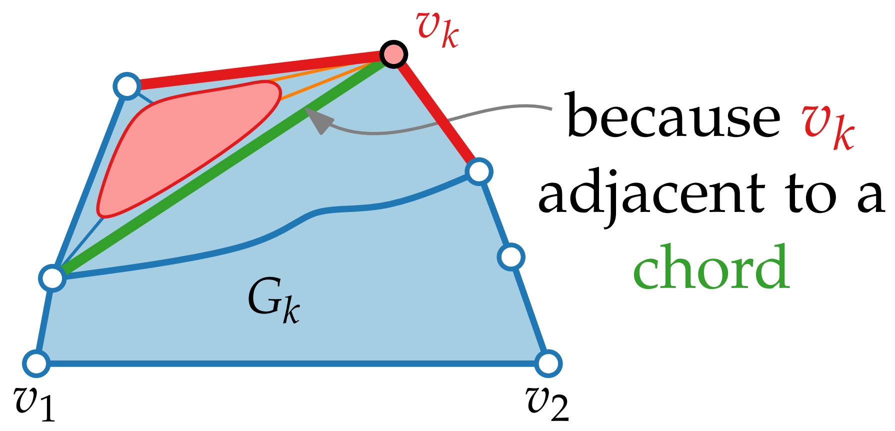
Base Case:
Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions (C1) – (C3) hold.

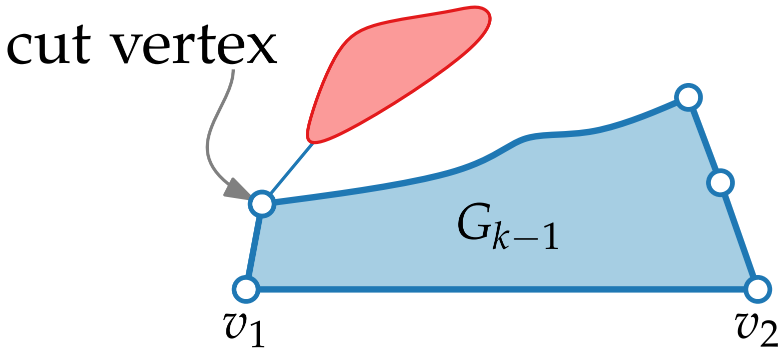
Induction hypothesis:
Vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions (C1) – (C3) hold for $k + 1 \leq i \leq n$.

Induction step: Consider G_k. We search for v_k.

Have to show:

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



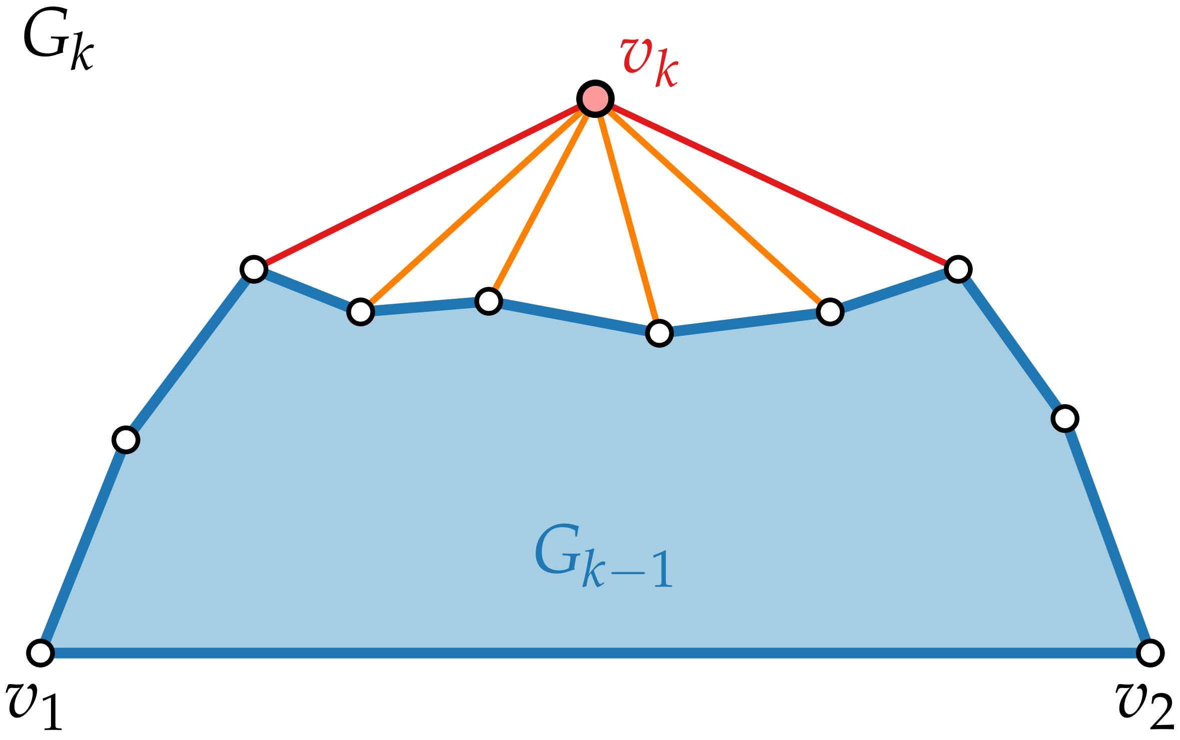


Canonical Order – Existence

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

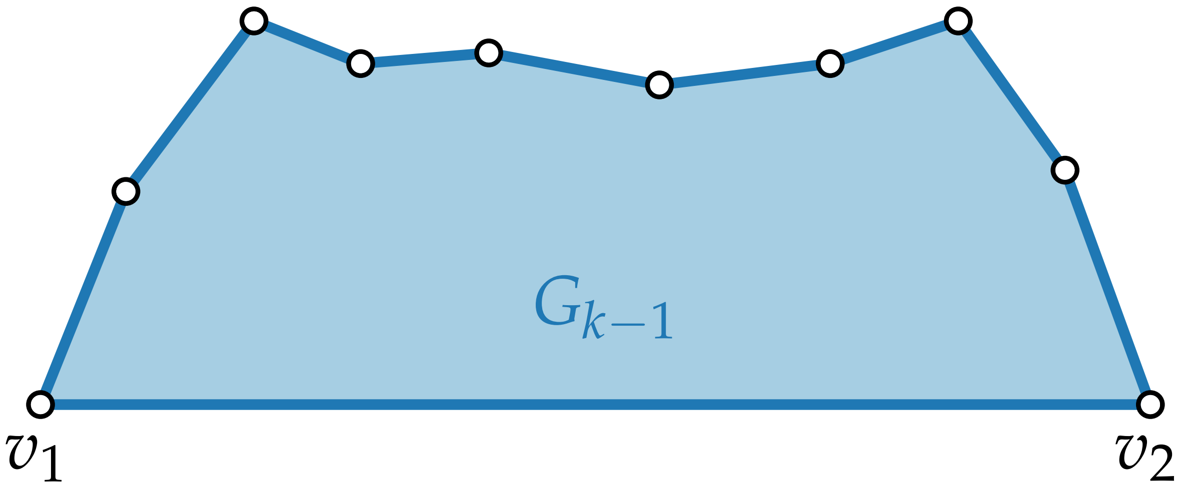
Canonical Order – Existence

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



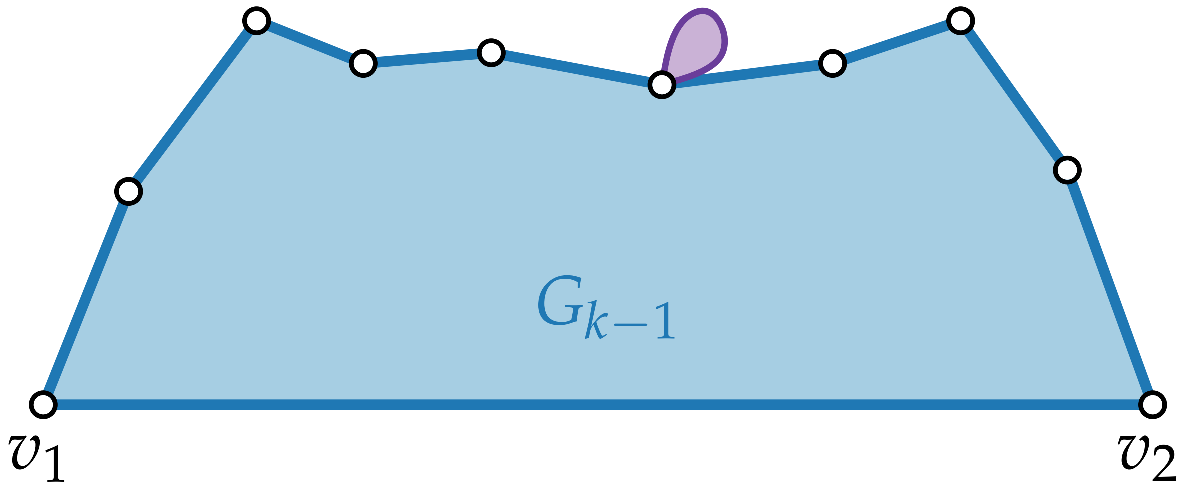
Canonical Order – Existence

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



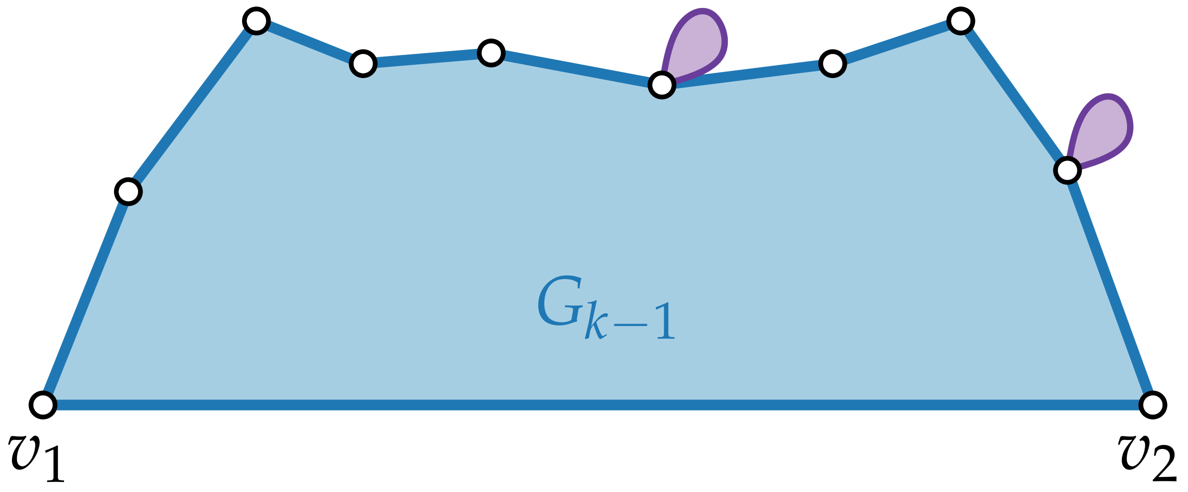
Canonical Order – Existence

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



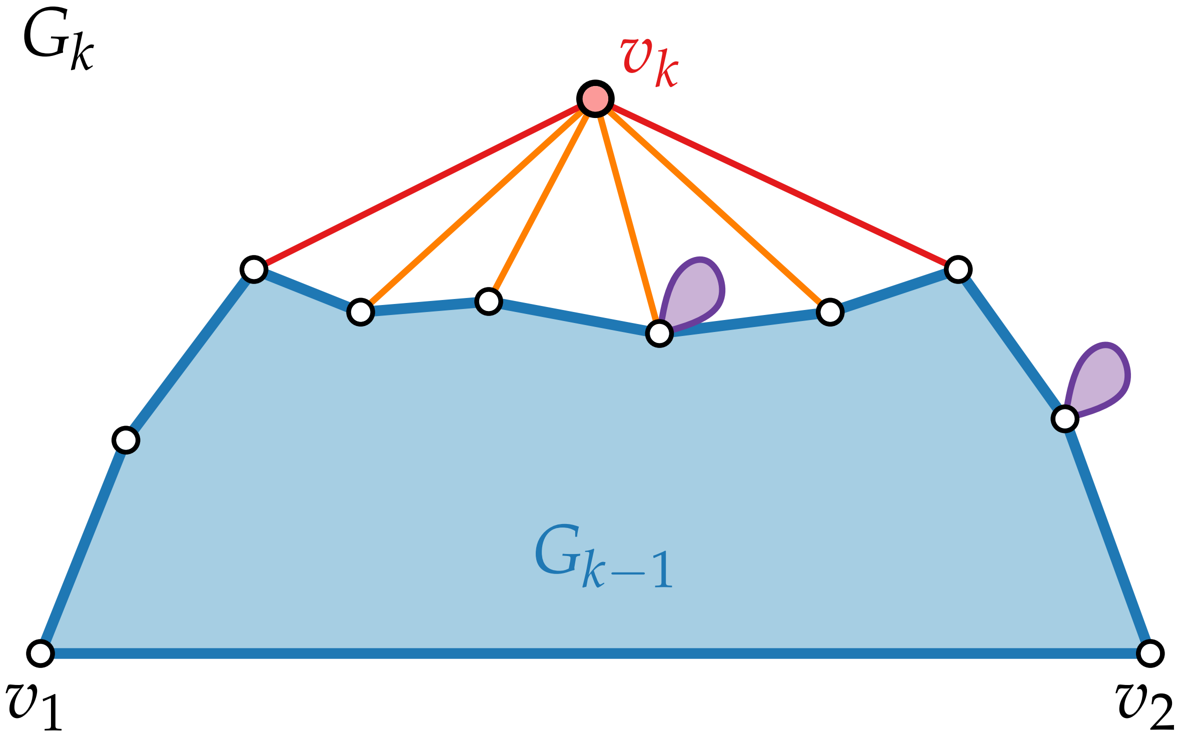
Canonical Order – Existence

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



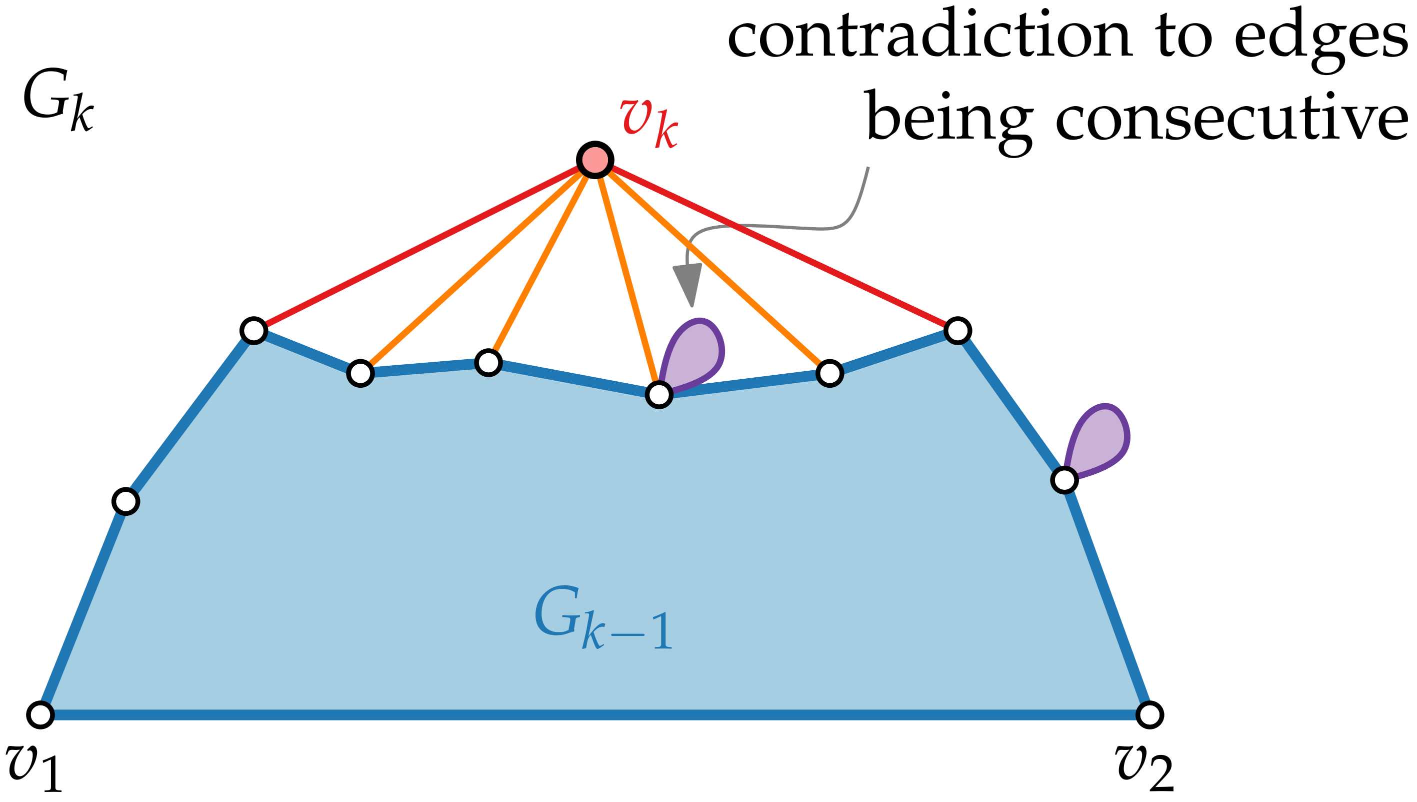
Canonical Order – Existence

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



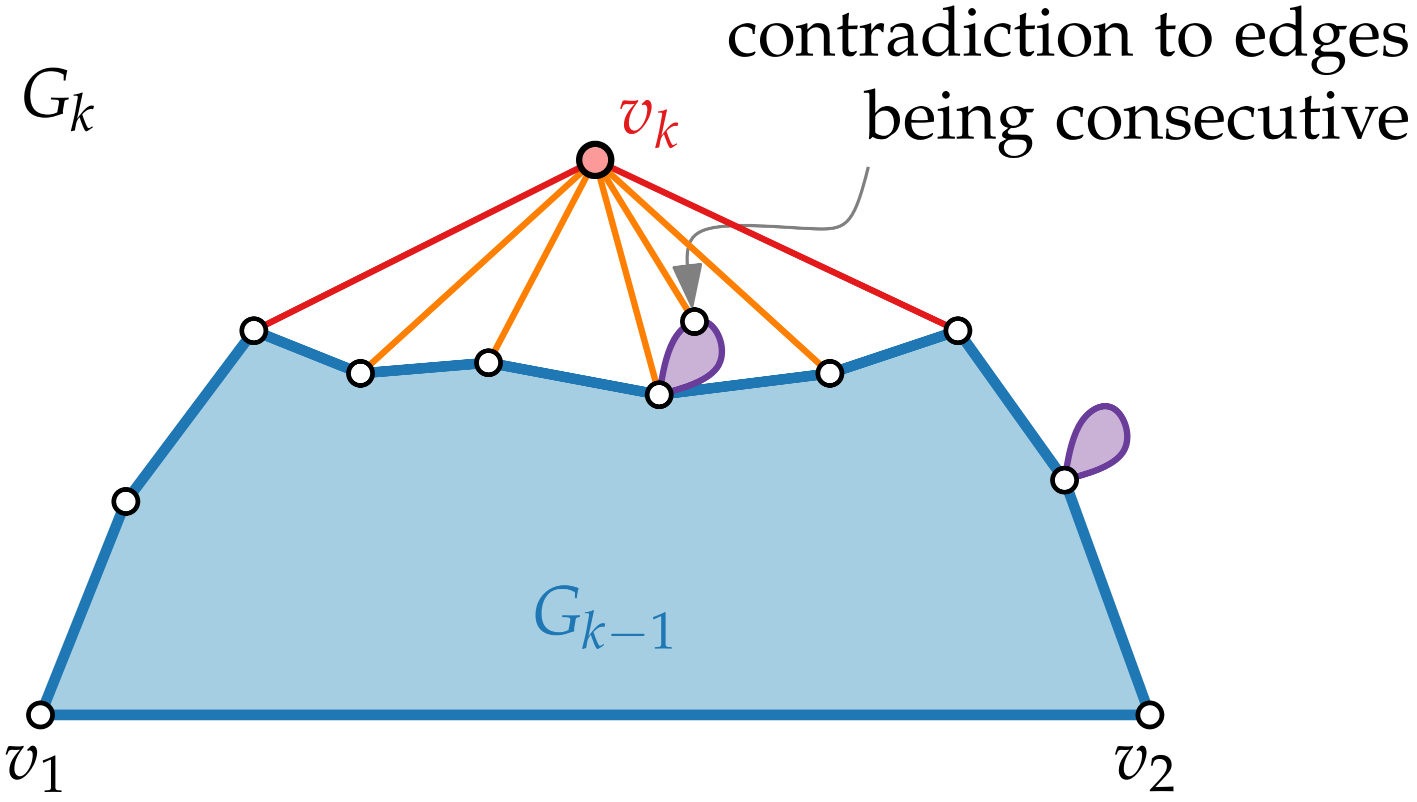
Canonical Order – Existence

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



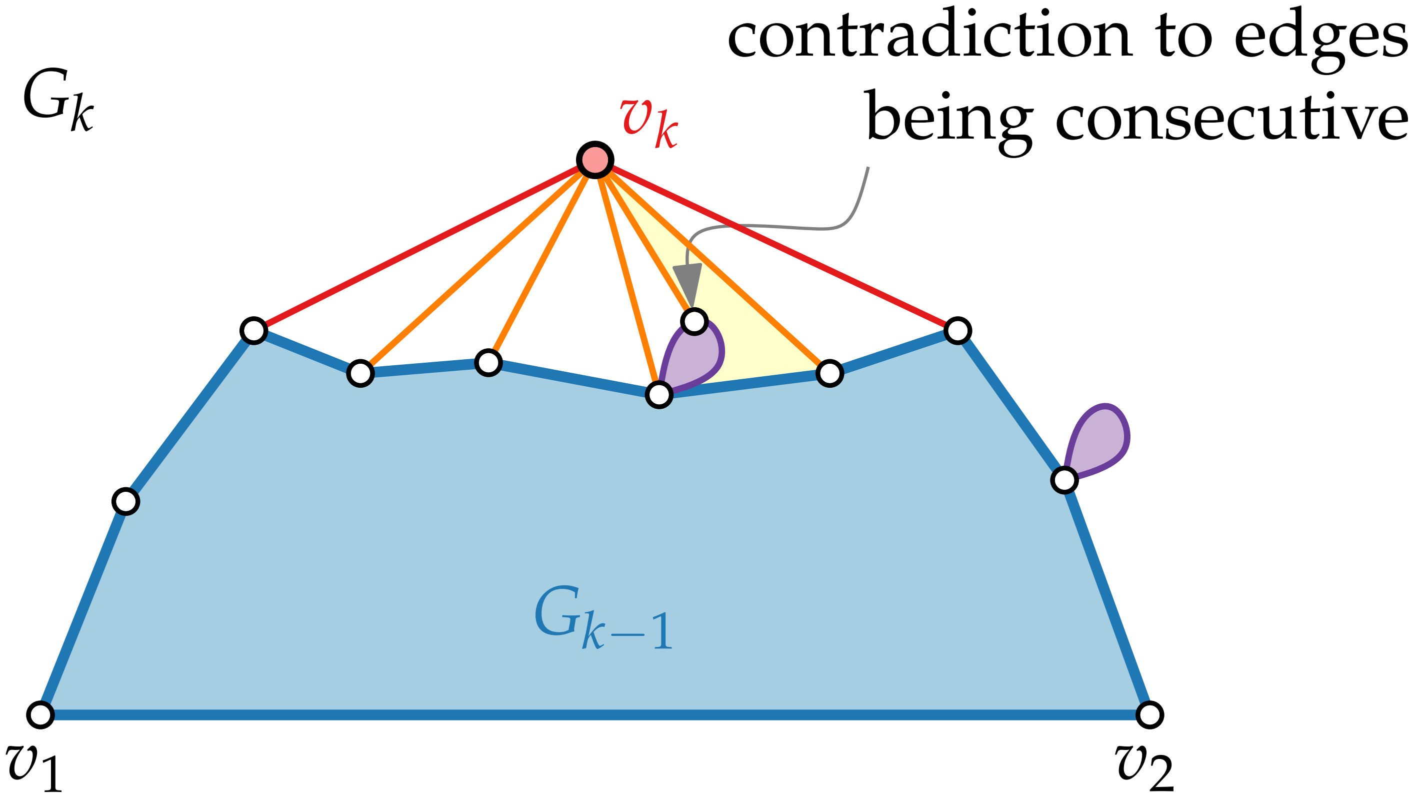
Canonical Order – Existence

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



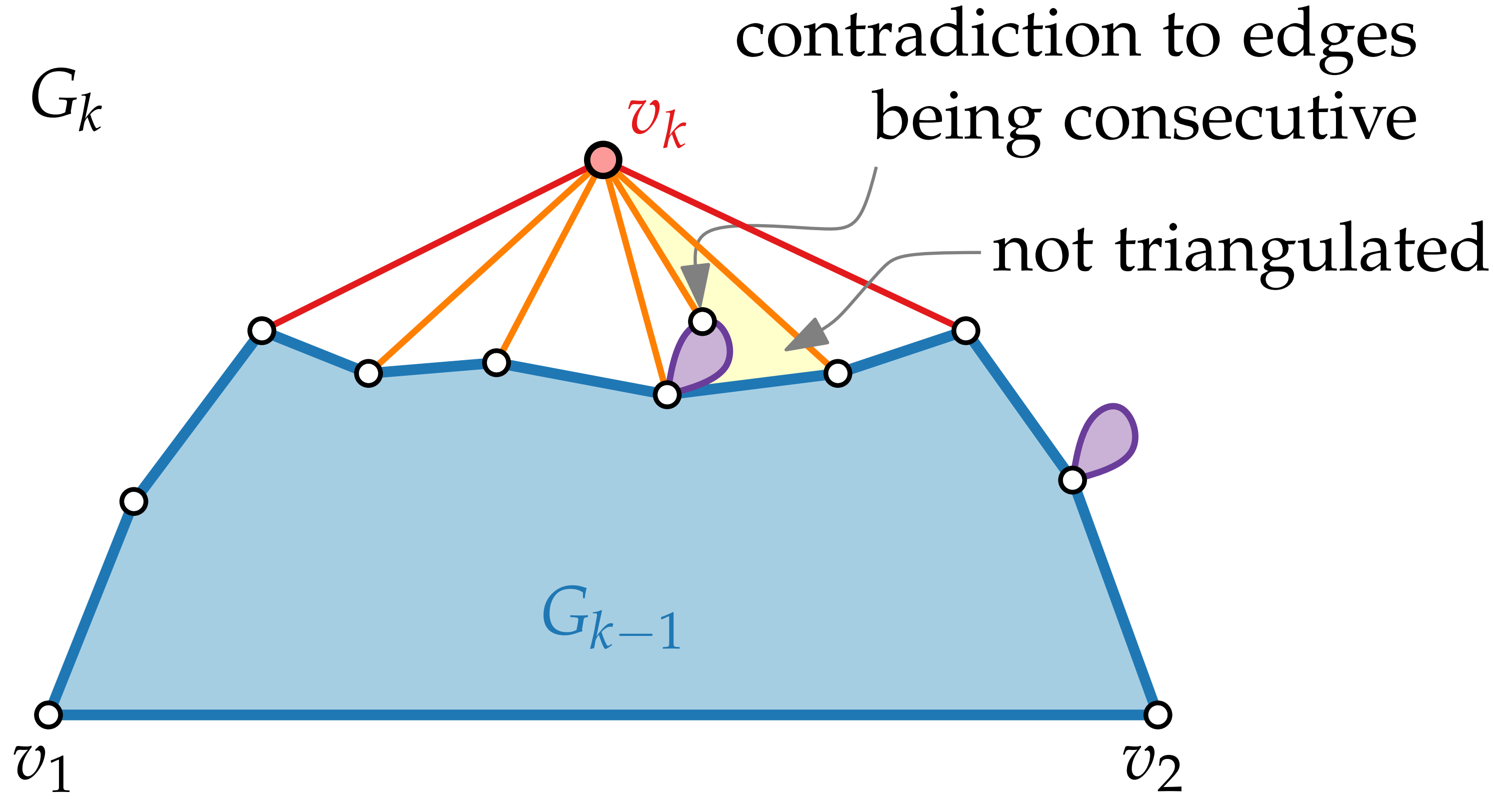
Canonical Order – Existence

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



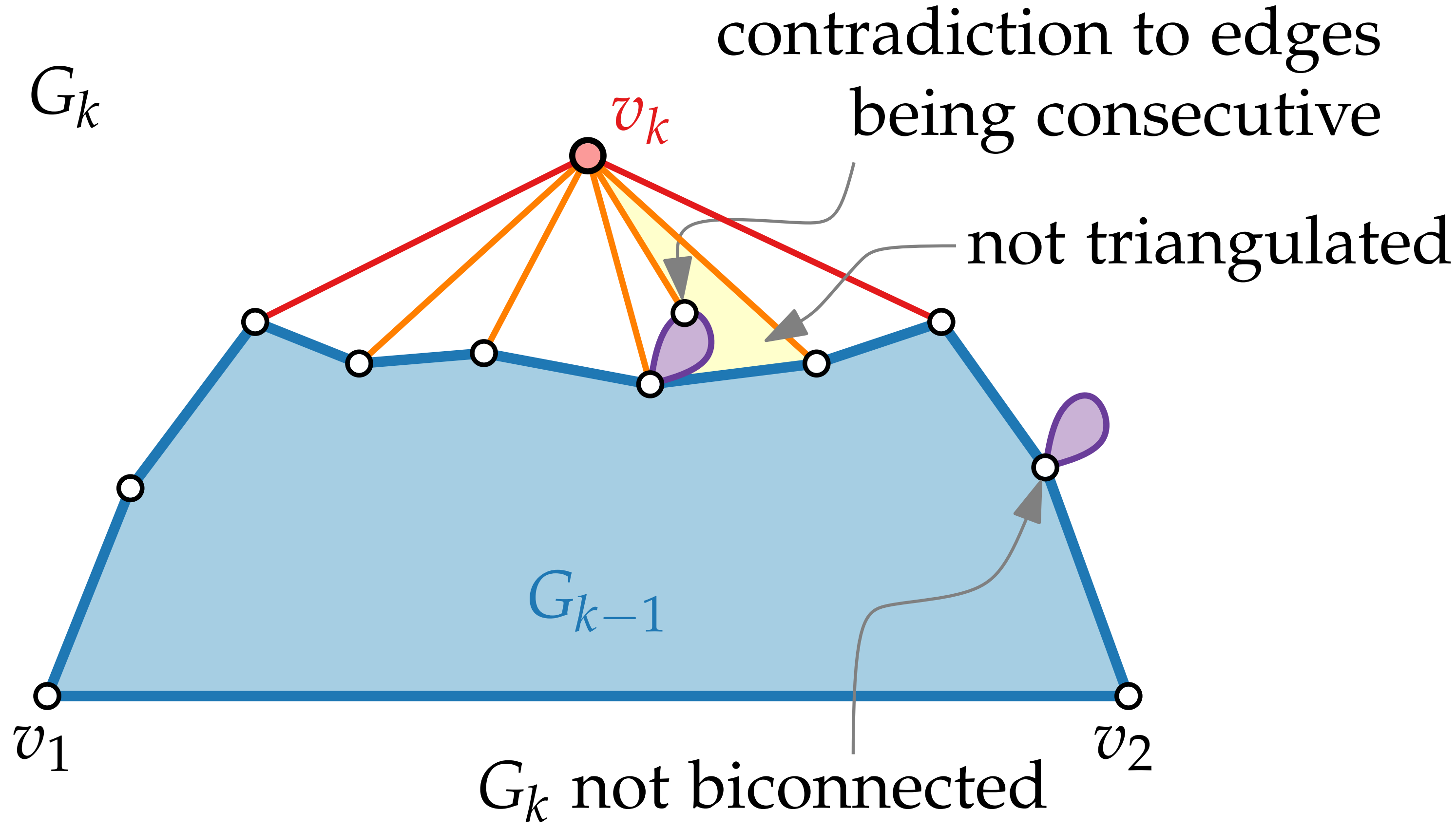
Canonical Order – Existence

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



Canonical Order – Existence

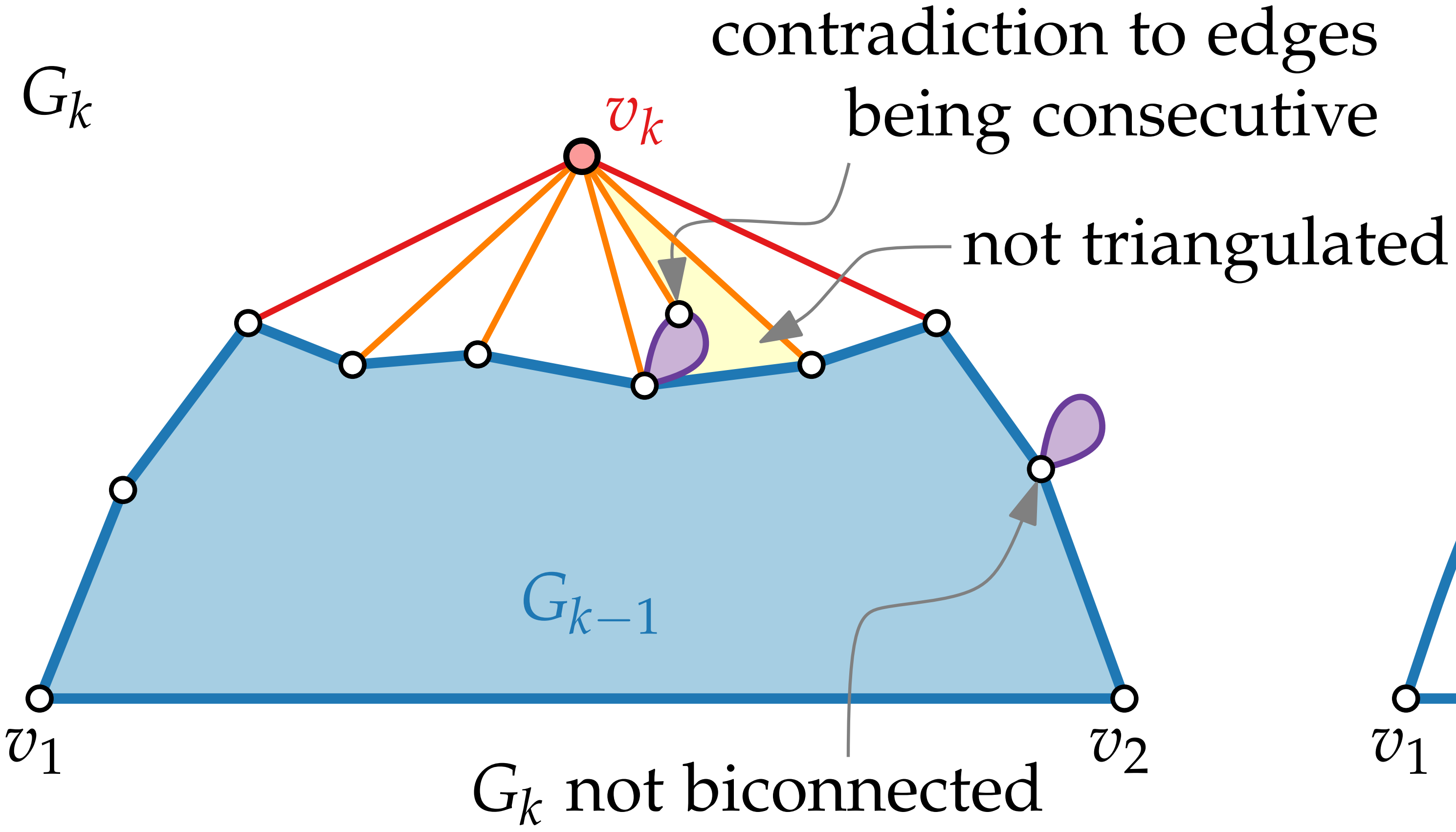
Claim 1.
If v_k is not adjacent to a chord,
then G_{k-1} is biconnected.

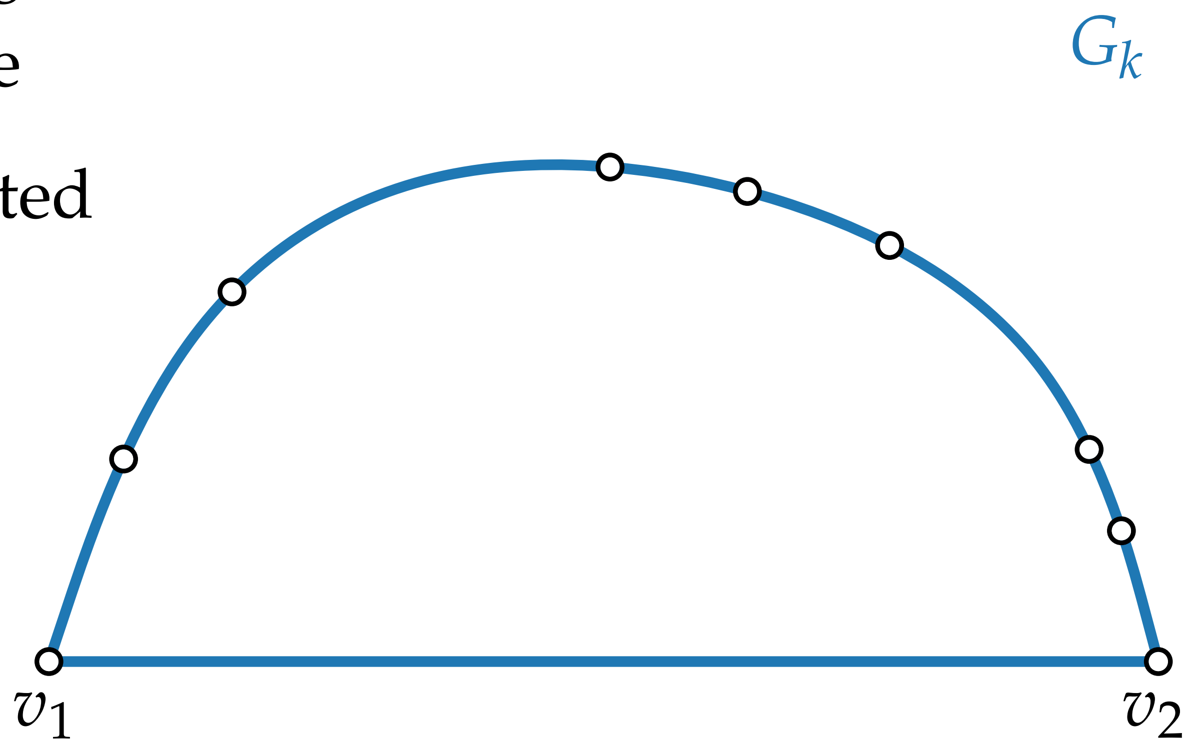


Canonical Order – Existence

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

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

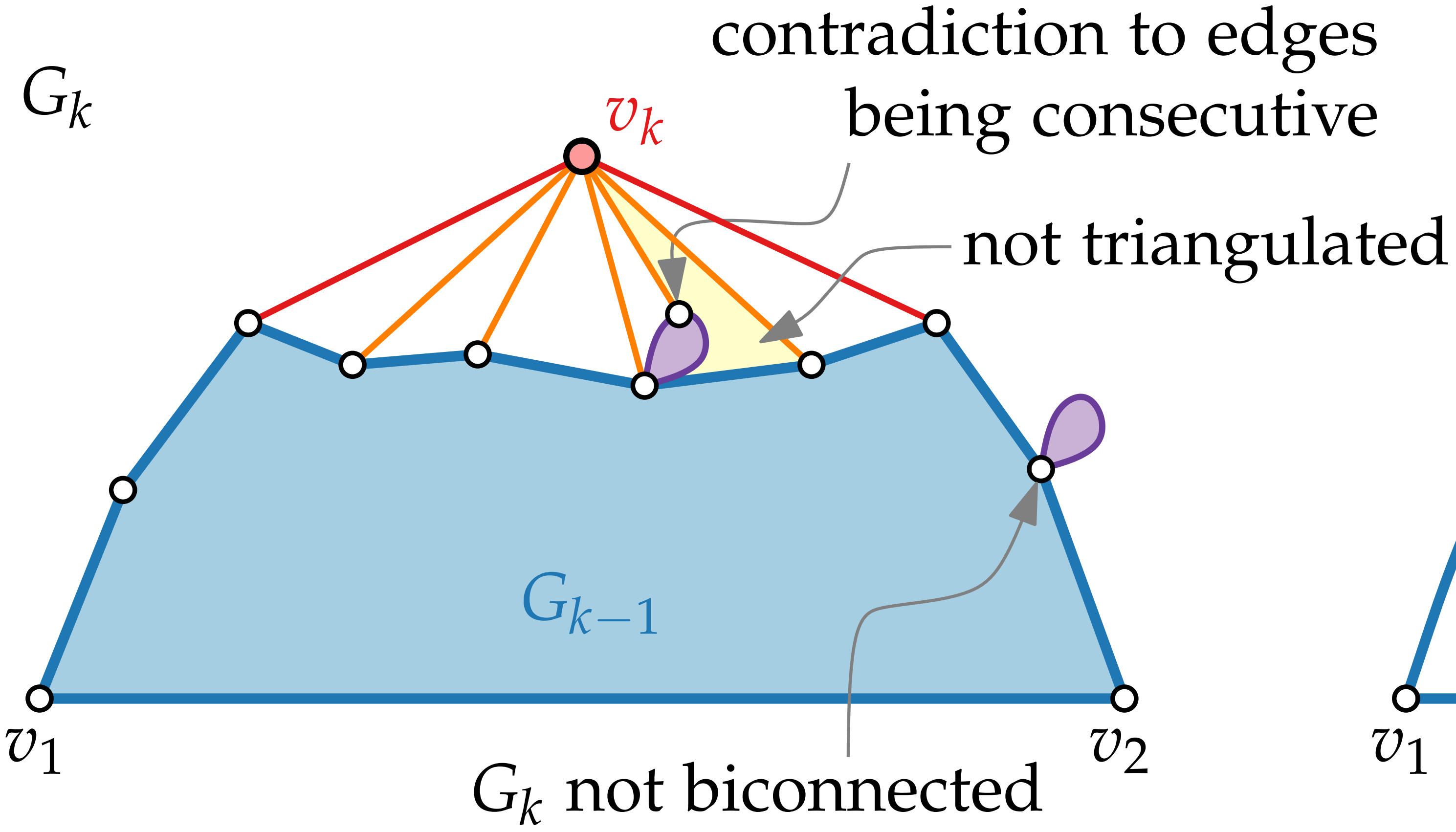


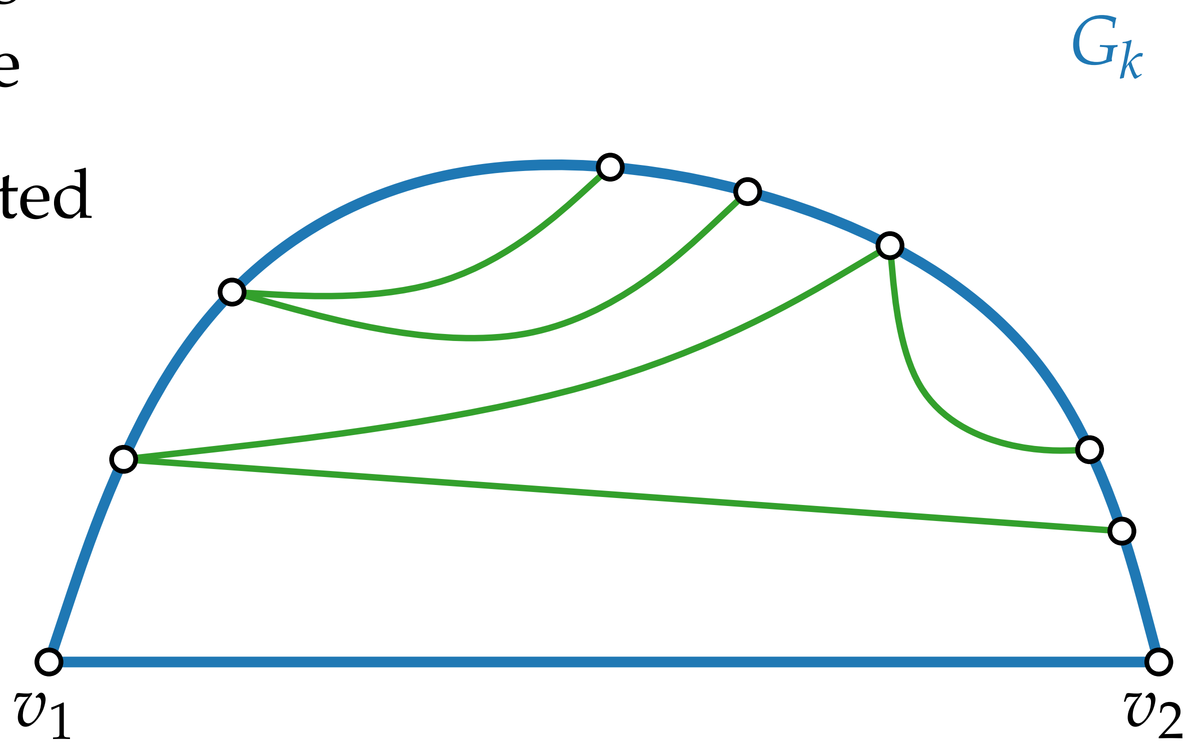


Canonical Order – Existence

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

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

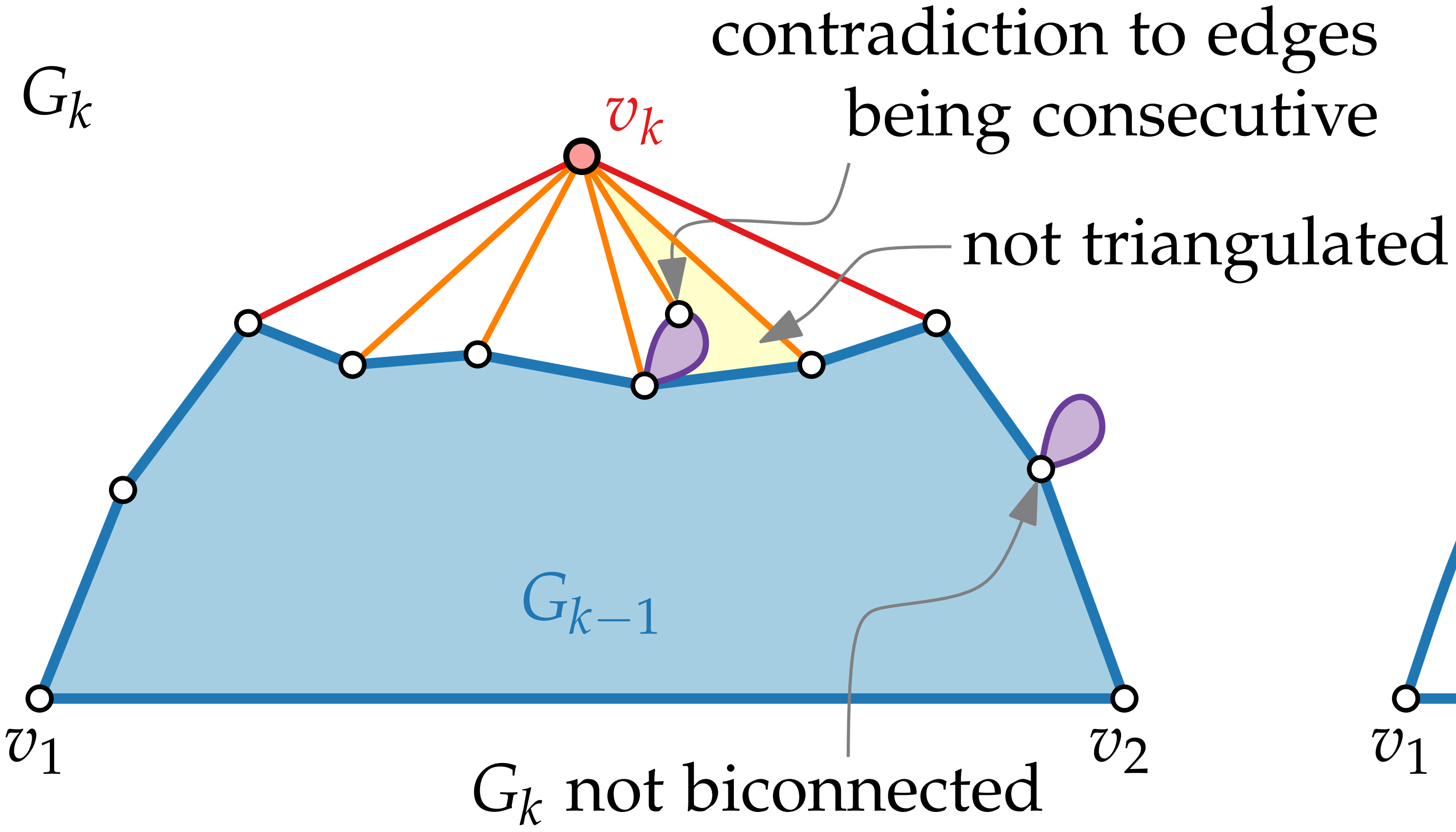


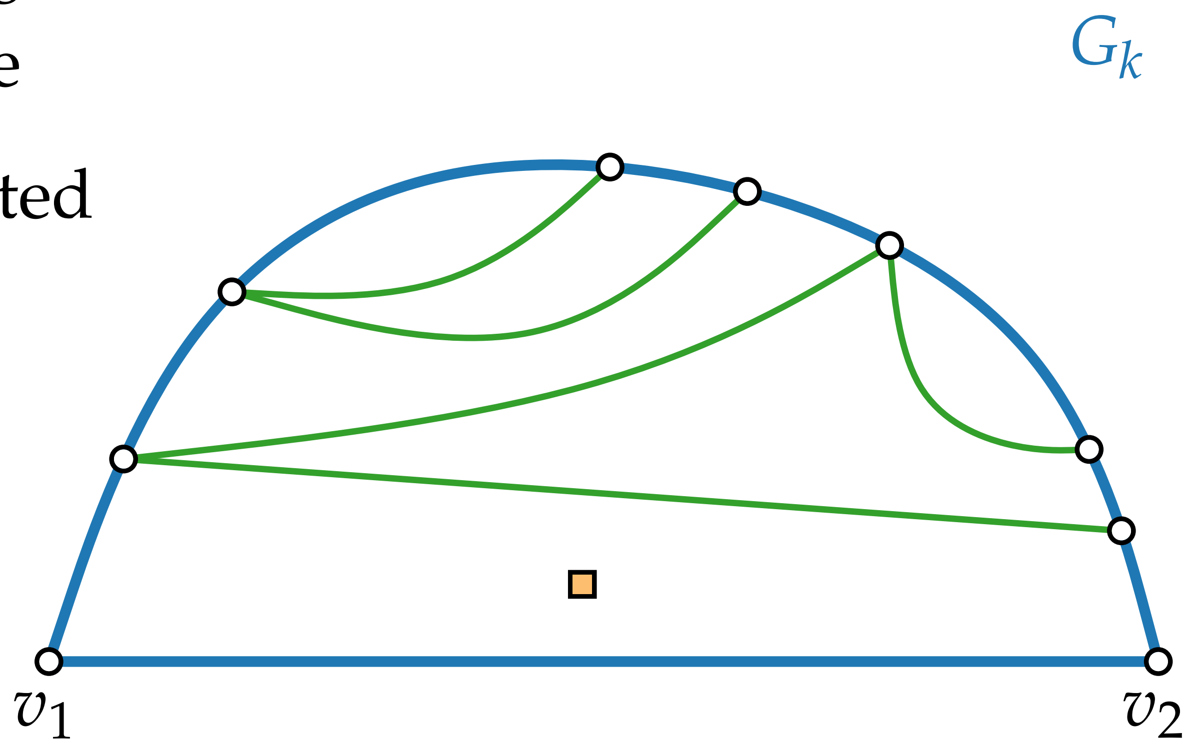


Canonical Order – Existence

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

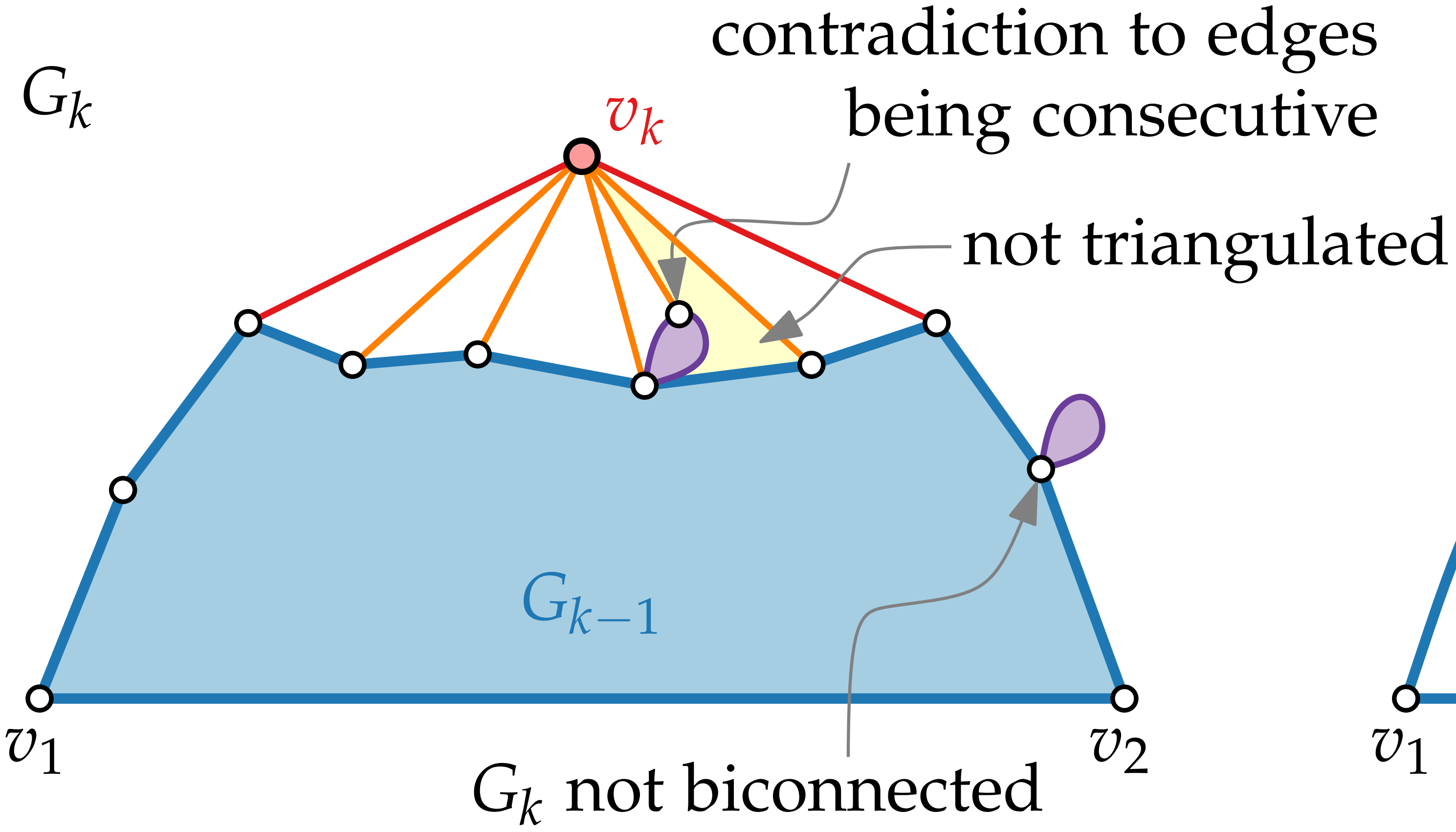
Claim 2.
There exists a vertex in G_k that is not adjacent to a chord as choice for v_k.



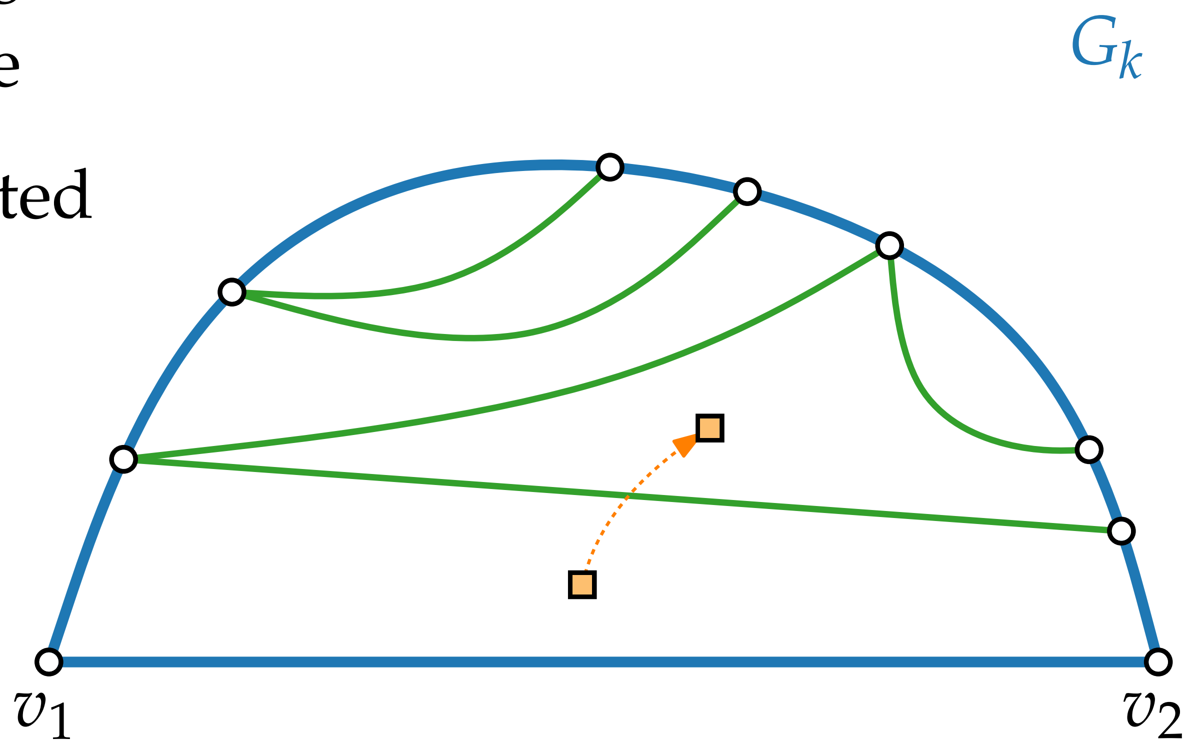


Canonical Order – Existence

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



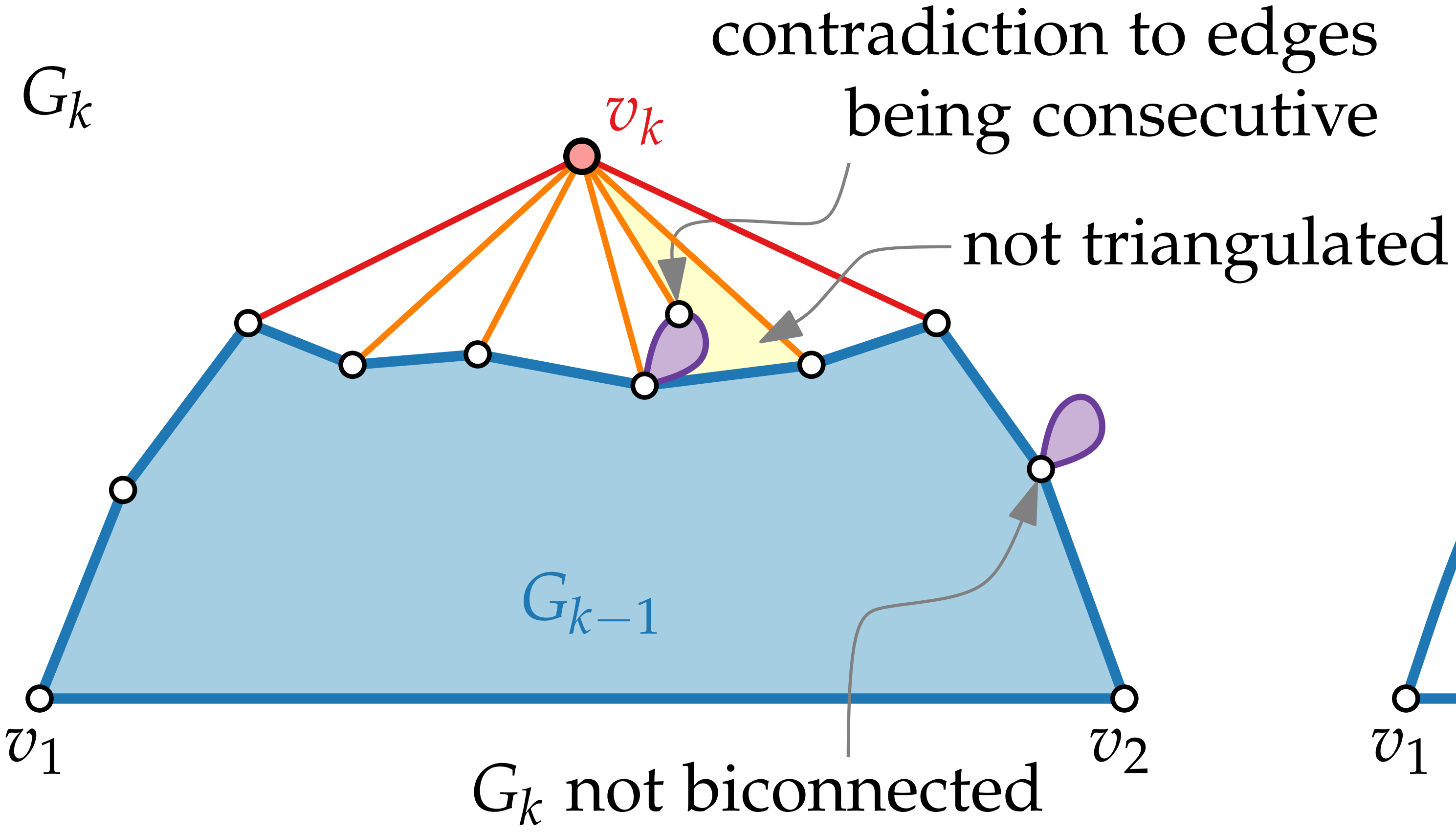
Claim 2.
There exists a vertex in G_k that is not adjacent to a chord as choice for v_k.

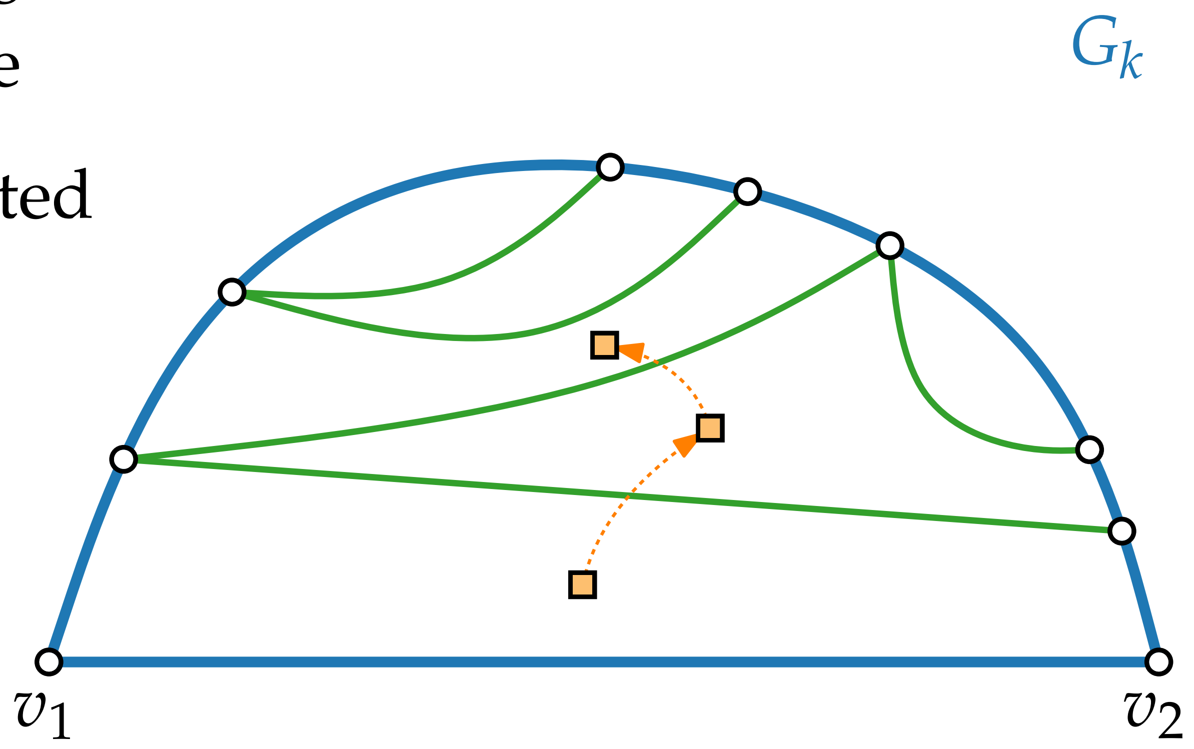


Canonical Order – Existence

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

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

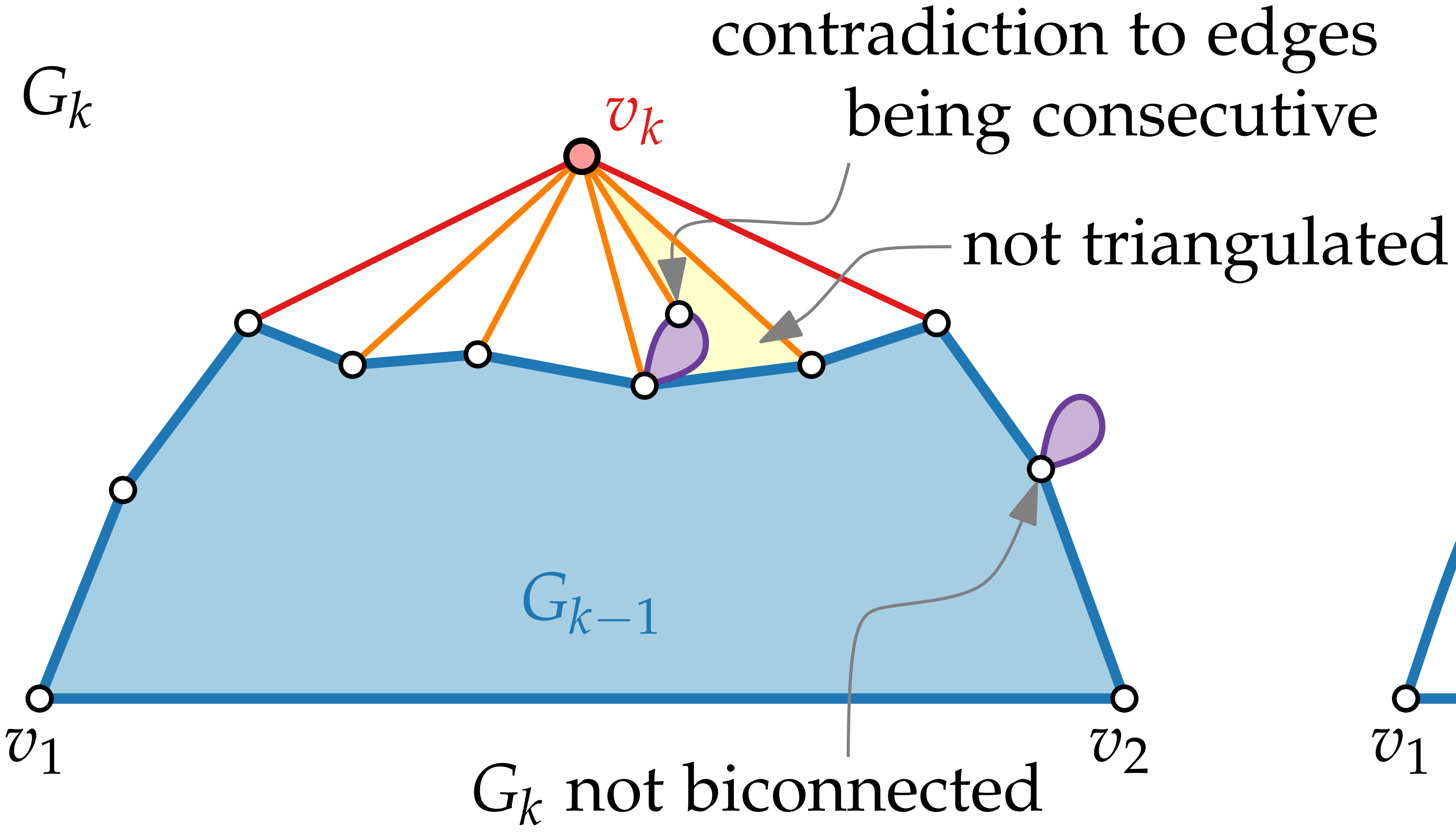


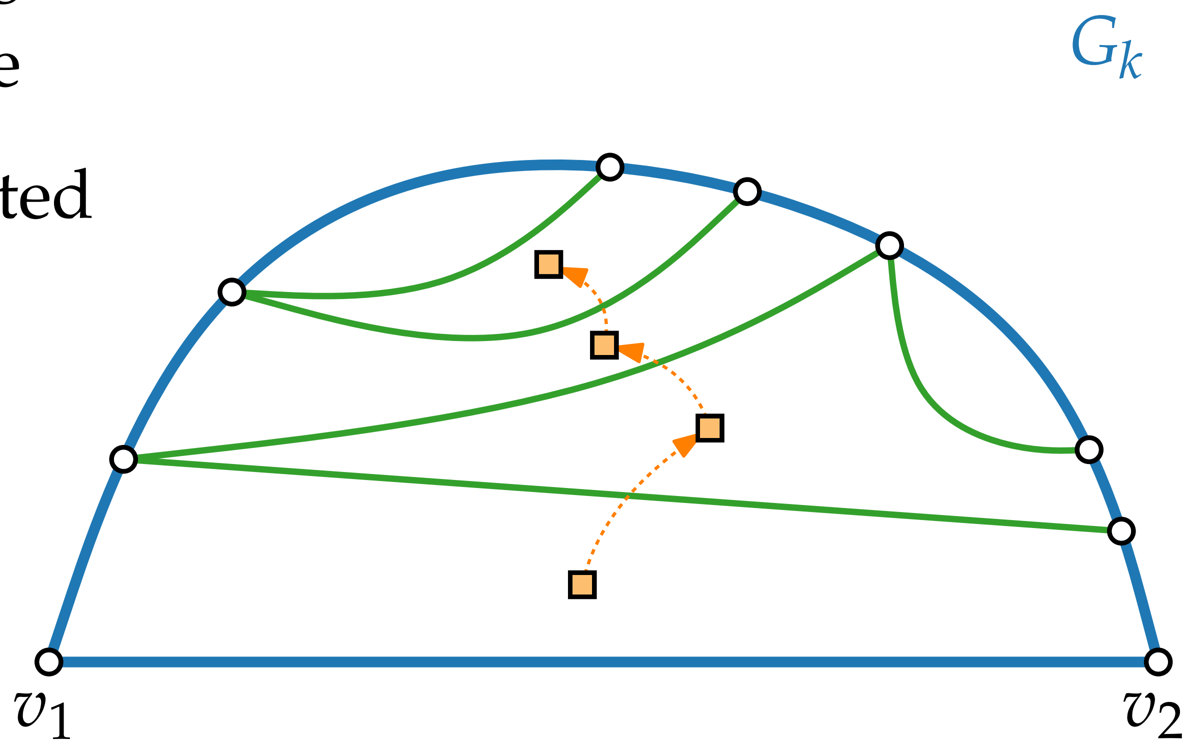


Canonical Order – Existence

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

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

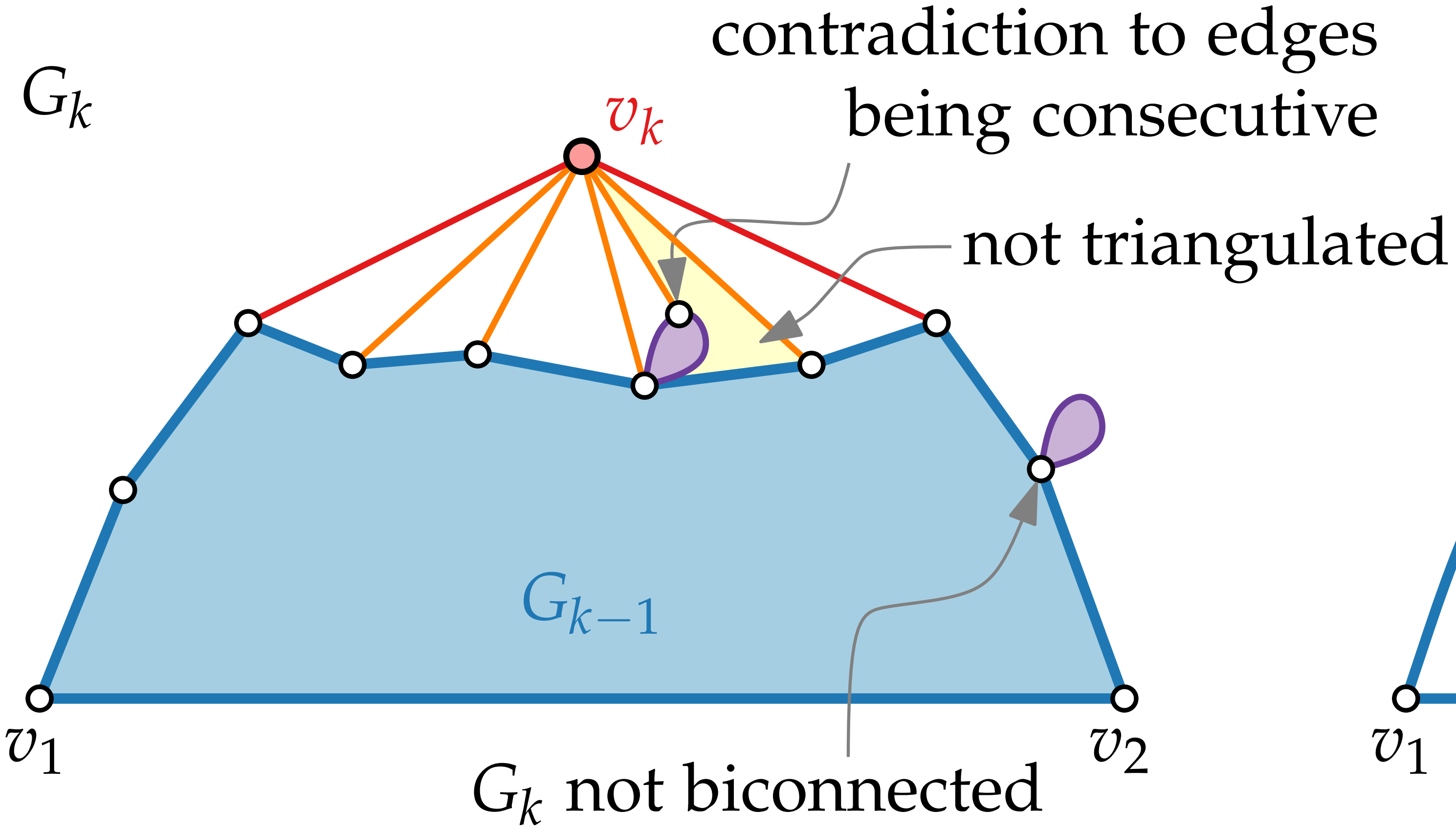


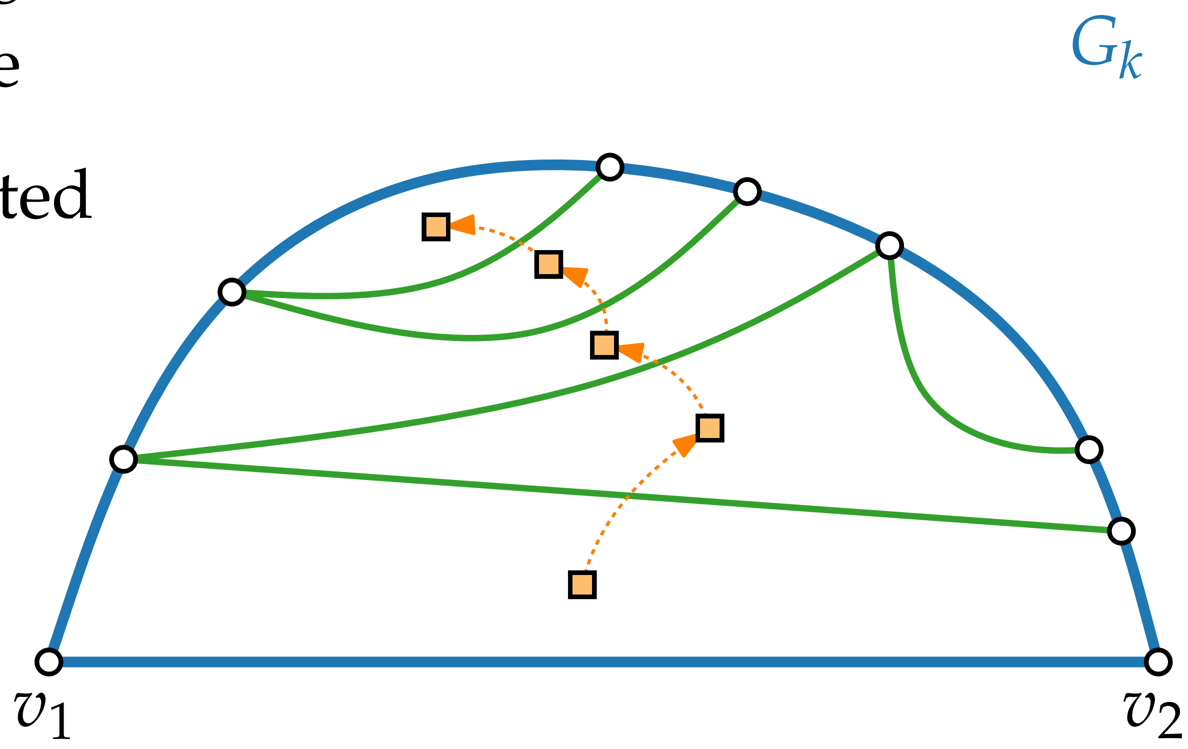


Canonical Order – Existence

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

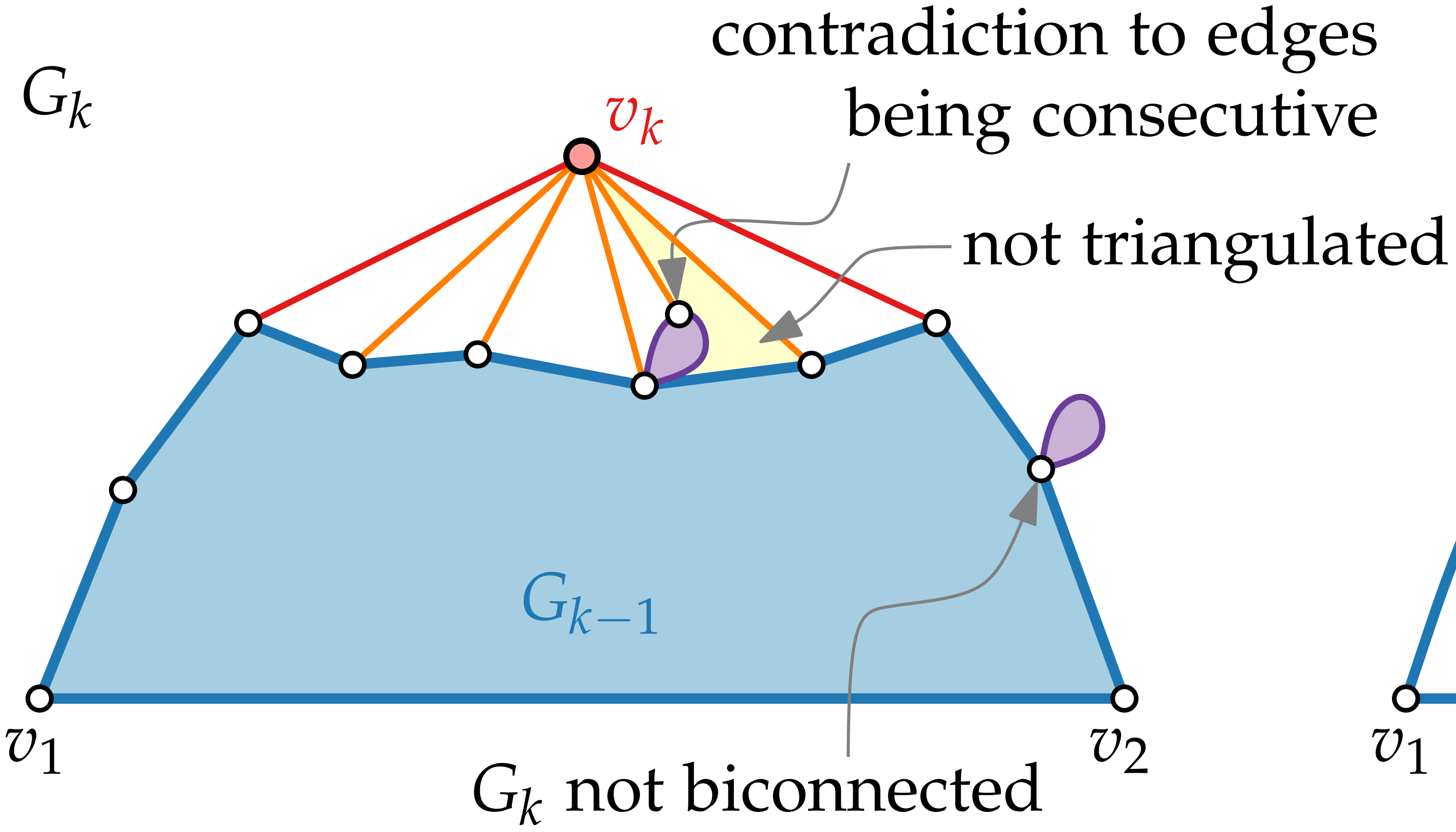
Claim 2.
There exists a vertex in G_k that is not adjacent to a chord as choice for v_k.



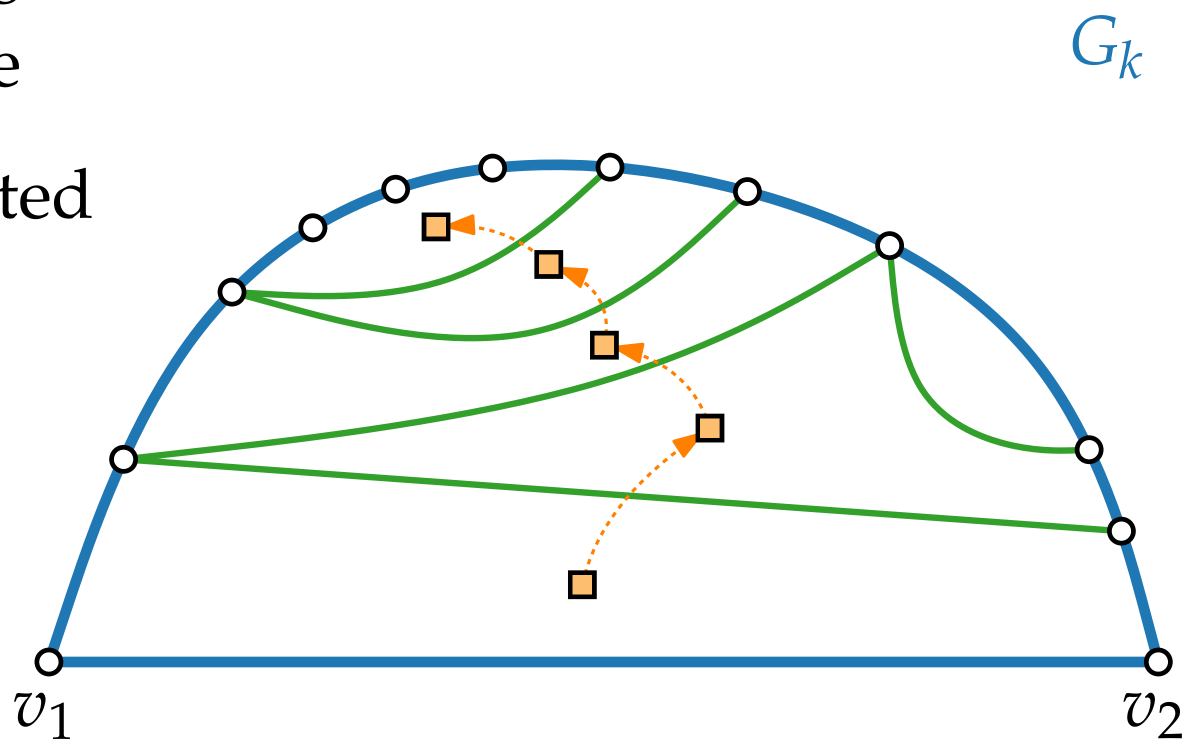


Canonical Order – Existence

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



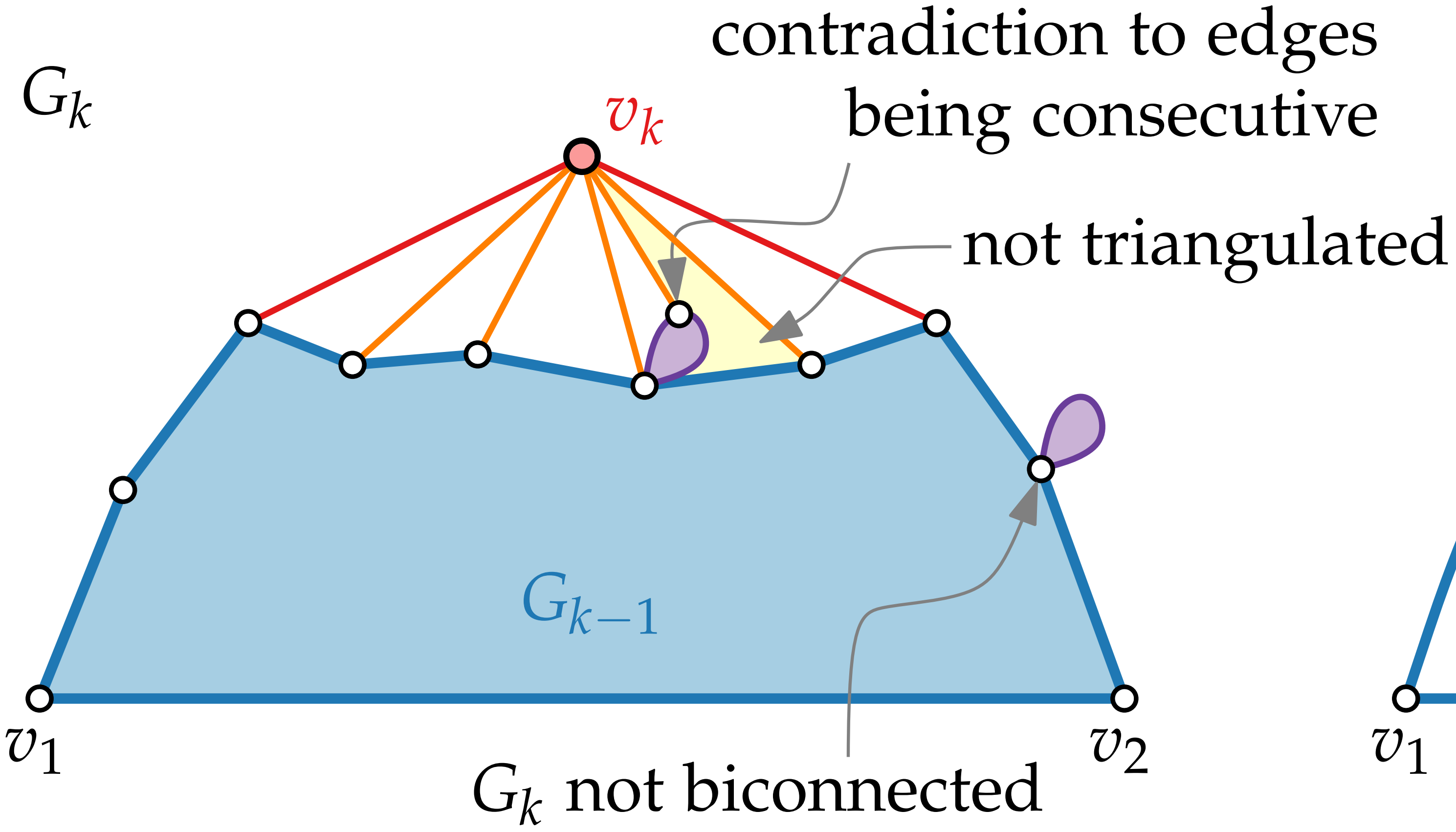
Claim 2.
There exists a vertex in G_k that is not adjacent to a chord as choice for v_k.

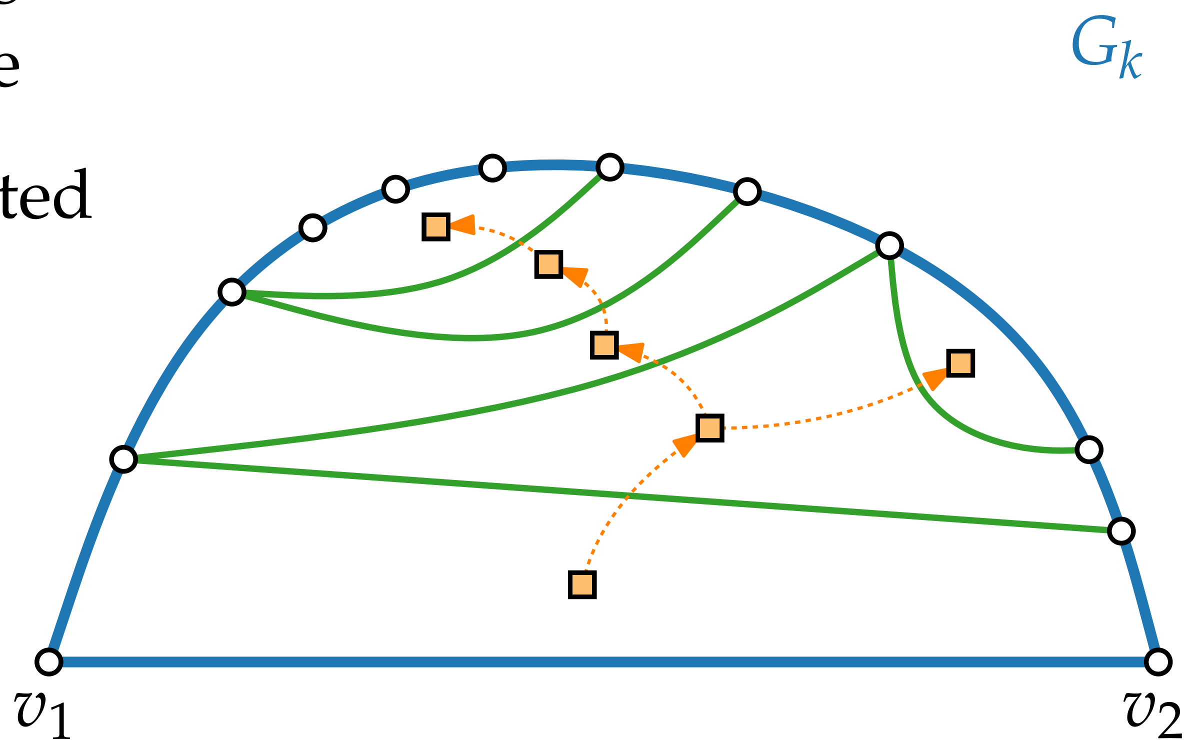


Canonical Order – Existence

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

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

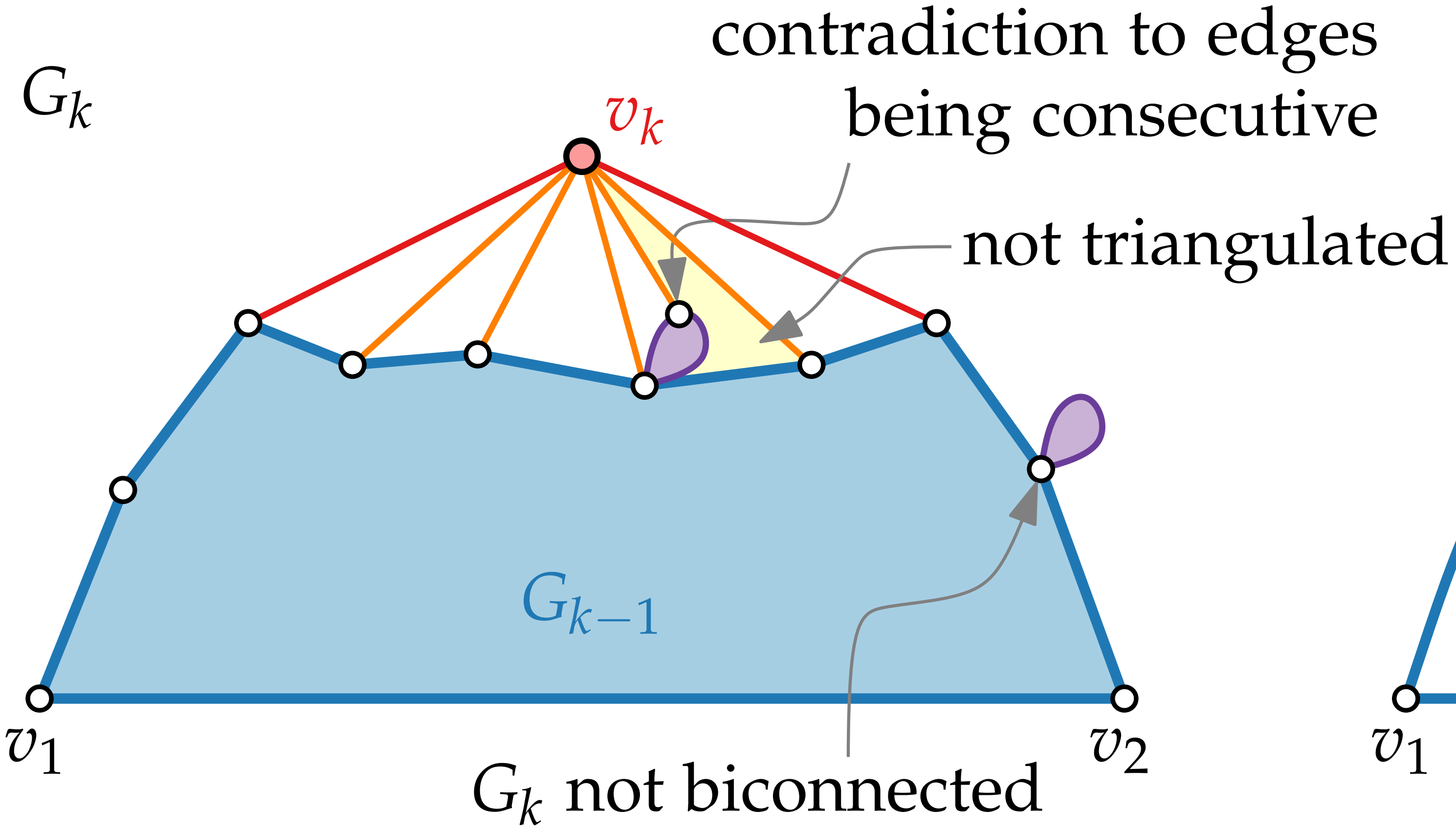


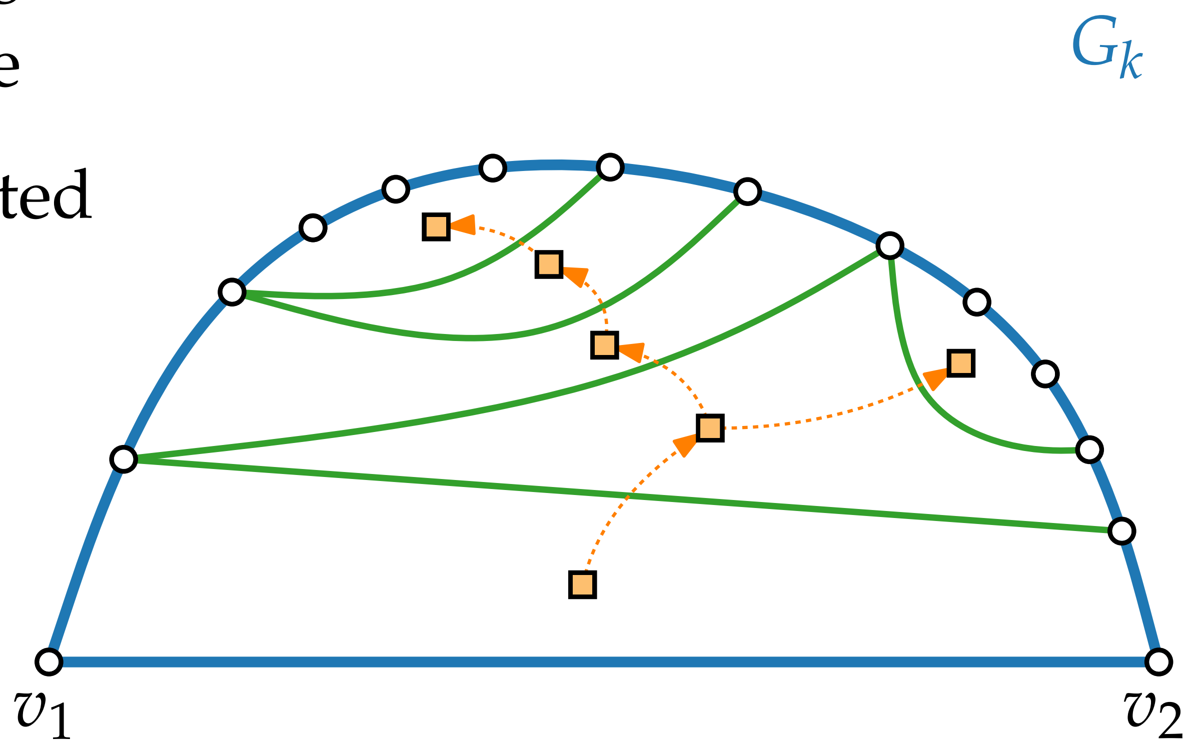


Canonical Order – Existence

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

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

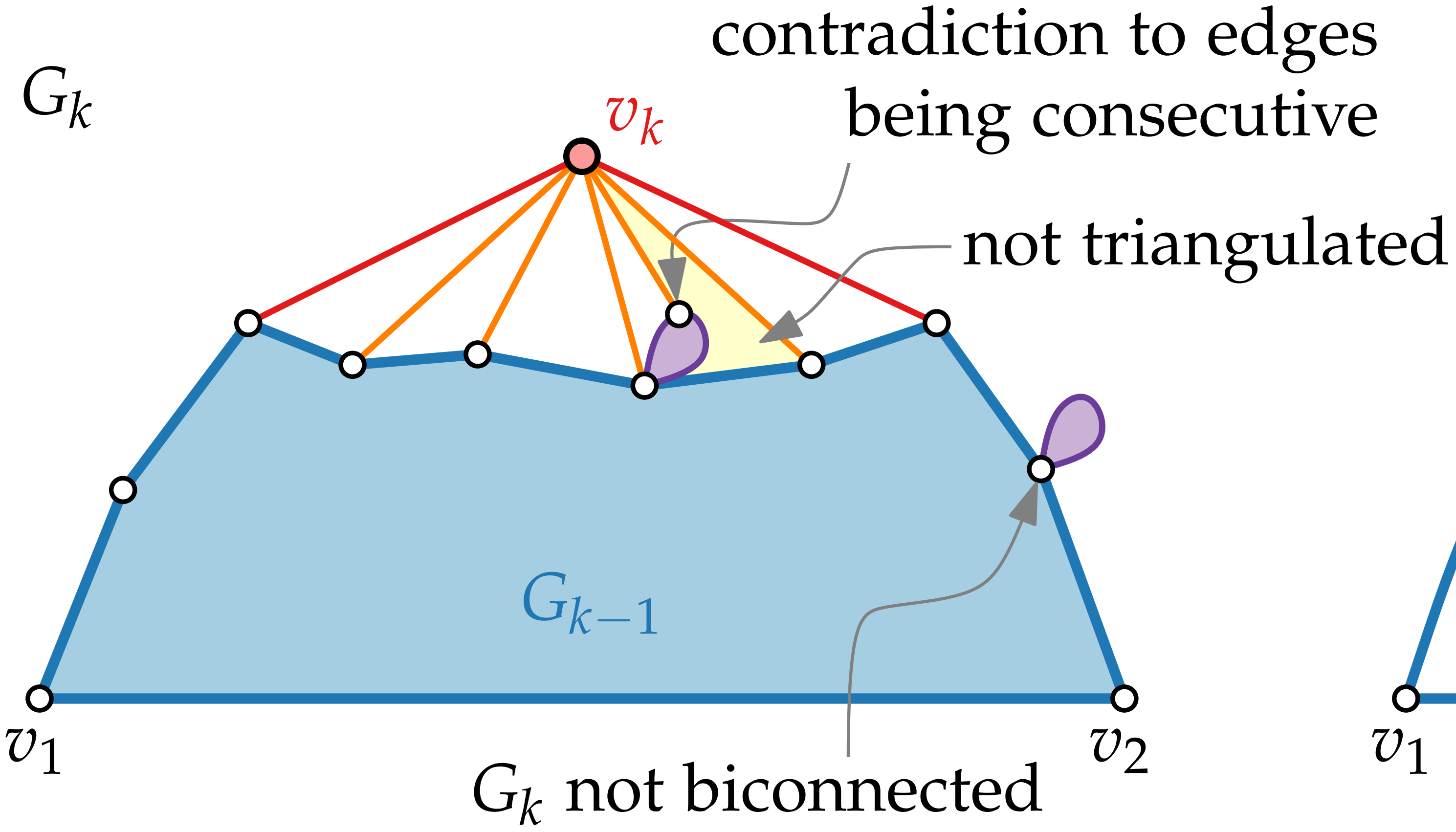


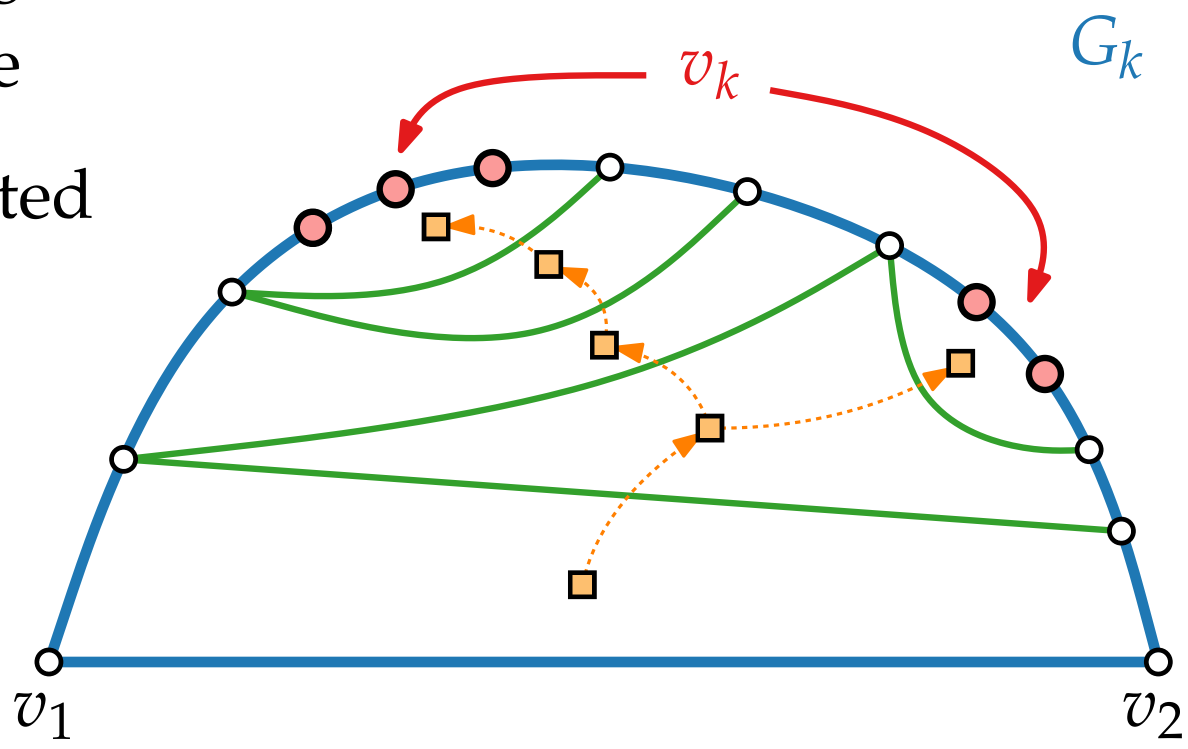


Canonical Order – Existence

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

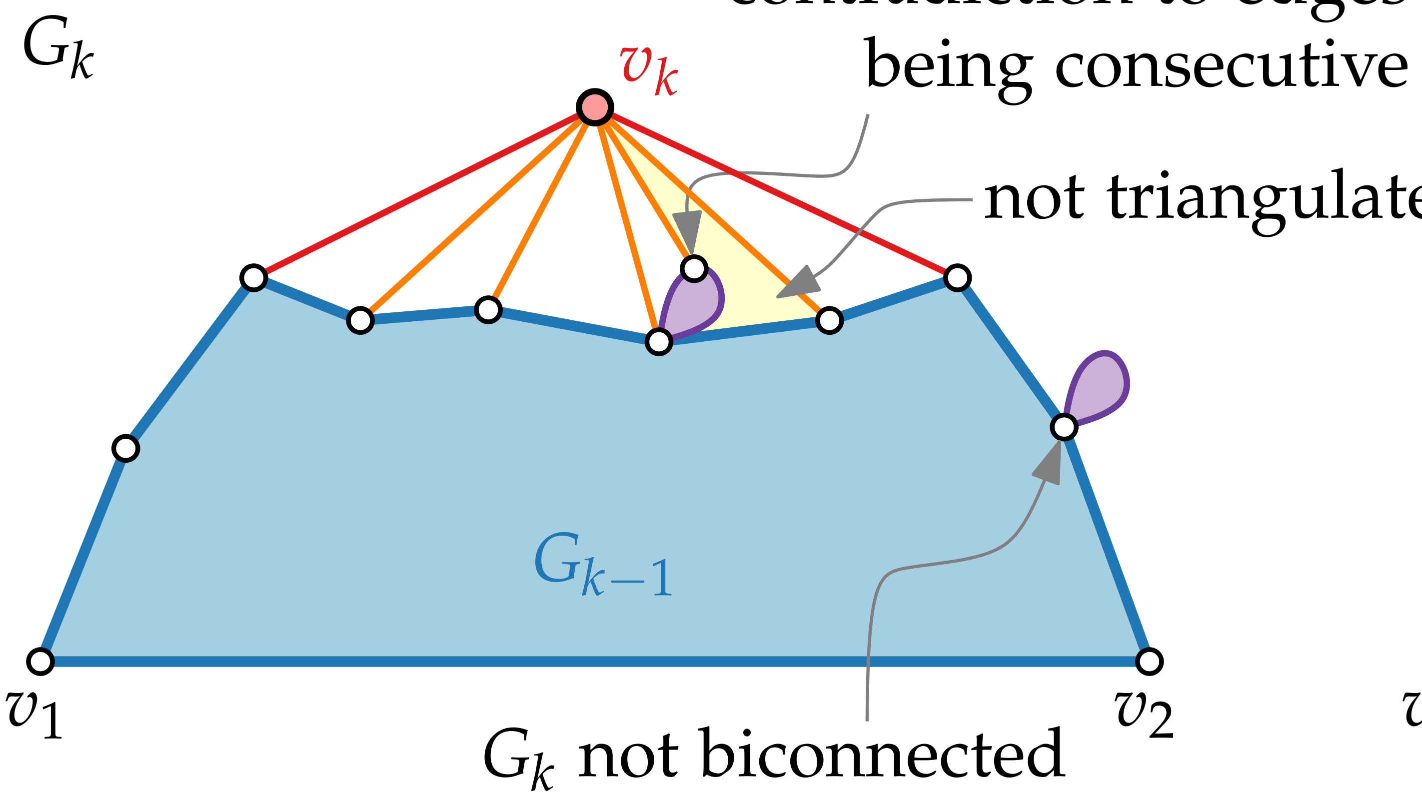
Claim 2.
There exists a vertex in G_k that is not adjacent to a chord as choice for v_k.



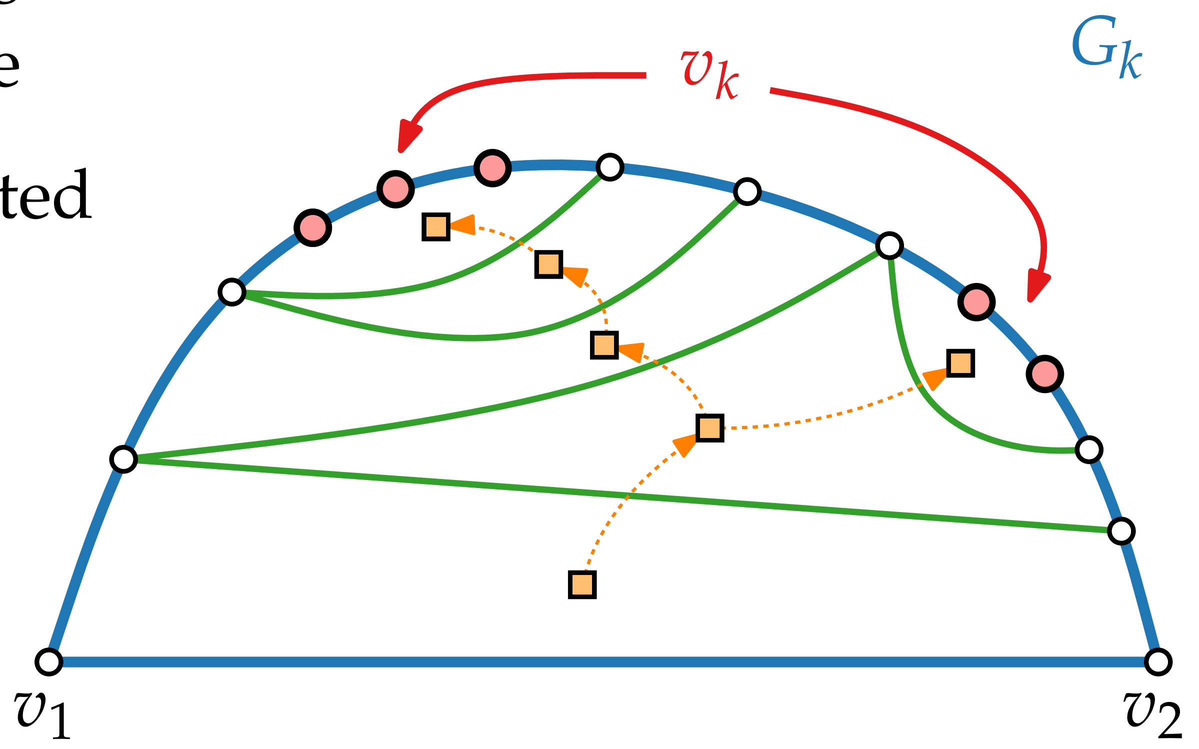


Canonical Order – Existence

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



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



This completes proof of Lemma. □

Canonical Order – Implementation

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

forall $v \in V$ **do**

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

- chord(v): # chords adjacent to v

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

forall $v \in V$ **do**

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

- chord(v): # chords adjacent to v

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

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

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

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

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

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E)$, (v_1, v_2, v_n))

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

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

for $k = n$ **to** 3 **do**
  choose v such that mark(v) = false, out(v) = true,
   and chords(v) = 0

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E)$, (v_1, v_2, v_n))

forall $v \in V$ **do**

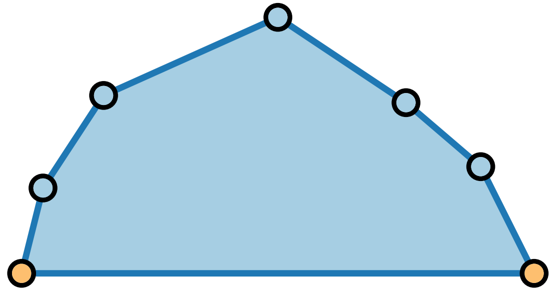
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

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

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

  choose v such that mark(v) = false, out(v) = true,
   and chords(v) = 0

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

forall $v \in V$ **do**

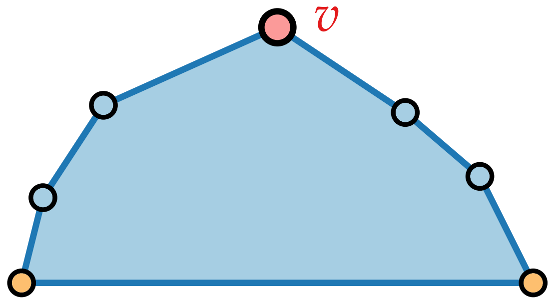
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false

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

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

  choose v such that mark(v) = false, out(v) = true, and chords(v) = 0

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



Canonical Order – Implementation

outer face

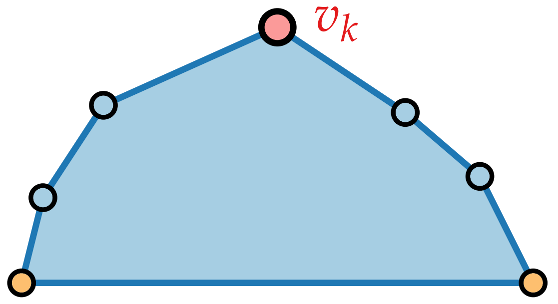
CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

forall $v \in V$ **do**
  chords(v) ← 0; out(v) ← false; mark(v) ← false

mark(v_1), mark(v_2), out(v_1), out(v_2), out(v_n) ← true

for $k = n$ **to** 3 **do**
  choose v such that mark(v) = false, out(v) = true,
   and chords(v) = 0
  $v_k \leftarrow v$; mark(v) ← true

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



Canonical Order – Implementation

outer face

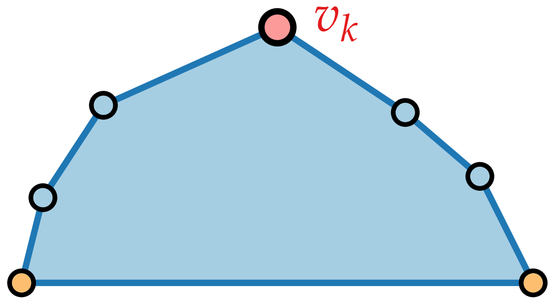
CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

forall $v \in V$ **do**
  chords(v) ← 0; out(v) ← false; mark(v) ← false

mark(v_1), mark(v_2), out(v_1), out(v_2), out(v_n) ← true

for $k = n$ **to** 3 **do**
  choose v such that mark(v) = false, out(v) = true, and chords(v) = 0
  $v_k \leftarrow v$; mark(v) ← true
  // Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1}

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



Canonical Order – Implementation

outer face

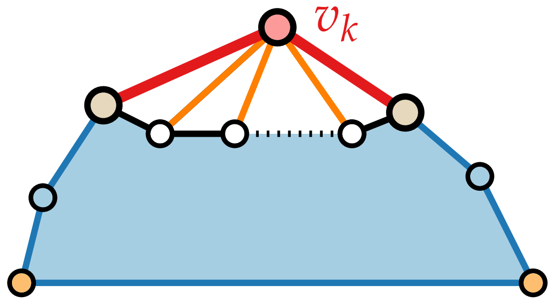
CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

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

for $k = n$ **to** 3 **do**
  choose v such that mark(v) = false, out(v) = true,
   and chords(v) = 0
  $v_k \leftarrow v$; mark(v) $\leftarrow$ true
  // Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1}

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



Canonical Order – Implementation

outer face

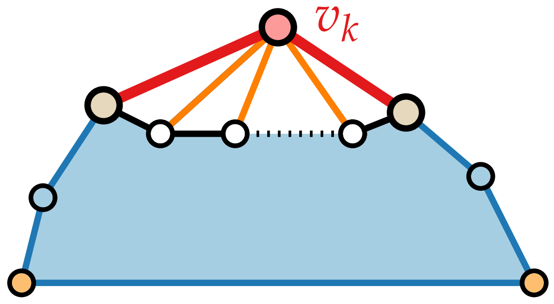
CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

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

for $k = n$ **to** 3 **do**
  choose v such that mark(v) = false, out(v) = true, and chords(v) = 0
  $v_k \leftarrow v$; mark(v) $\leftarrow$ true
  // Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1} in G_{k-1} and let $w_p, \ldots, w_q$ be the neighbors of v_k

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



Canonical Order – Implementation

outer face

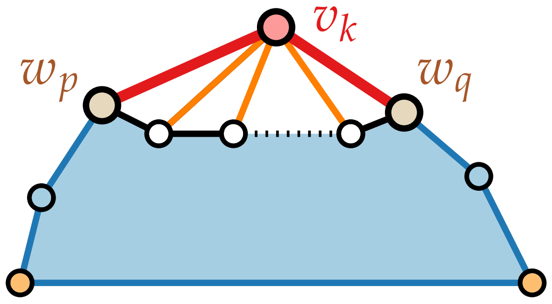
CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

forall $v \in V$ **do**
  chords(v) ← 0; out(v) ← false; mark(v) ← false

mark(v_1), mark(v_2), out(v_1), out(v_2), out(v_n) ← true

for $k = n$ **to** 3 **do**
  choose v such that mark(v) = false, out(v) = true, and chords(v) = 0
  $v_k \leftarrow v$; mark(v) ← true
  // Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1} in G_{k-1} and let $w_p, \ldots, w_q$ be the neighbors of v_k

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



Canonical Order – Implementation

outer face

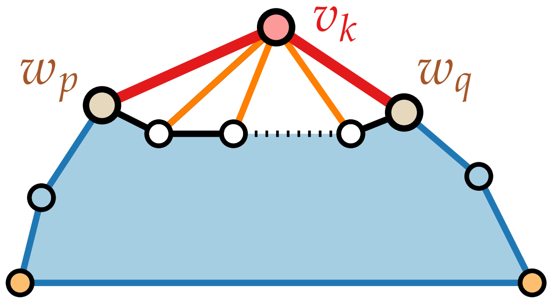
CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

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

for $k = n$ **to** 3 **do**
 choose v such that mark(v) = false, out(v) = true, and chords(v) = 0
 $v_k \leftarrow v$; mark(v) $\leftarrow$ true
 // Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1} in G_{k-1} and let $w_p, \ldots, w_q$ be the neighbors of v_k
 out(w_i) $\leftarrow$ true for all $p < i < q$

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



Canonical Order – Implementation

outer face

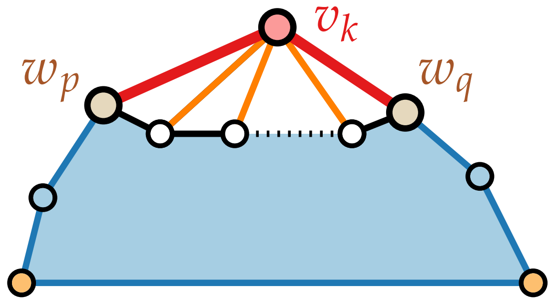
CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

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

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

for $k = n$ **to** 3 **do**
  choose v such that mark(v) = false, out(v) = true, and chords(v) = 0
  $v_k \leftarrow v$; mark(v) $\leftarrow$ true
  // Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1} in G_{k-1} and let $w_p, \ldots, w_q$ be the neighbors of v_k
  out(w_i) $\leftarrow$ true for all $p < i < q$
  update number of chords for w_i and its neighbours

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



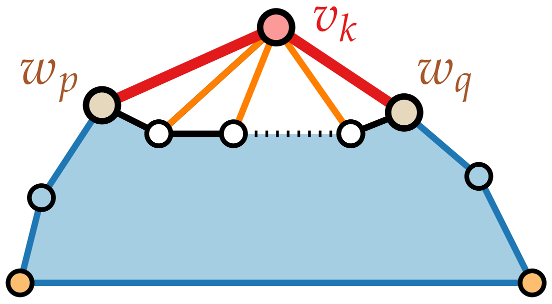
Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

forall $v \in V$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false
mark(v_1), mark(v_2), out(v_1), out(v_2), out(v_n) $\leftarrow$ true
for $k = n$ **to** 3 **do**
  choose v such that mark(v) = false, out(v) = true, and chords(v) = 0
  $v_k \leftarrow v$; mark(v) $\leftarrow$ true
  // Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1} in G_{k-1} and let $w_p, \ldots, w_q$ be the neighbors of v_k
  out(w_i) $\leftarrow$ true for all $p < i < q$
  update number of chords for w_i and its neighbours

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



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

Canonical Order – Implementation

outer face

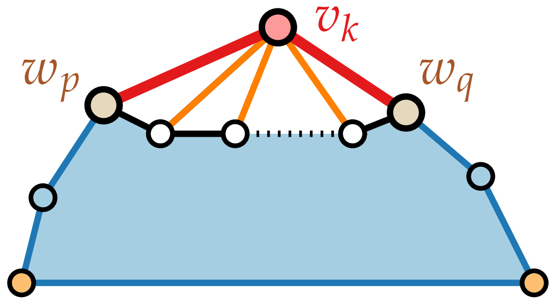
CanonicalOrder$(G = (V, E), (v_1, v_2, v_n))$

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

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

for $k = n$ **to** 3 **do**
  choose v such that mark(v) = false, out(v) = true,
   and chords(v) = 0 *// keep list with candidates*
  $v_k \leftarrow v$; mark$(v) \leftarrow$ true
  // Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1} in G_{k-1} and let $w_p, \ldots, w_q$ be the neighbors of v_k
  out$(w_i) \leftarrow$ true for all $p < i < q$
  update number of chords for w_i
  and its neighbours

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



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

Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

forall $v \in V$ **do**

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

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

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

 choose v such that mark(v) = false, out(v) = true,
 and chords(v) = 0 *// keep list with candidates*

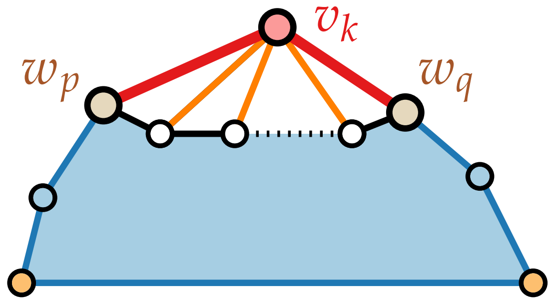
 $v_k \leftarrow v$; mark(v) $\leftarrow$ true

 // Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1} in G_{k-1} and let $w_p, \ldots, w_q$ be the neighbors of v_k

 out(w_i) $\leftarrow$ true for all $p < i < q$ *// $O(n)$ in total*

 update number of chords for w_i
 and its neighbours

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number



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

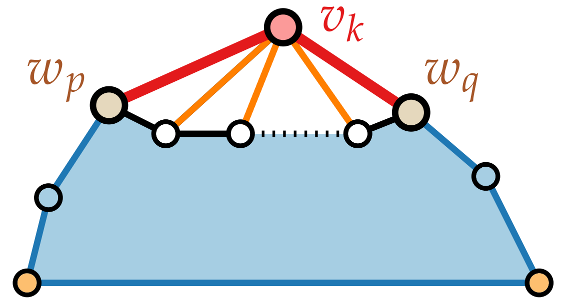
Canonical Order – Implementation

outer face

CanonicalOrder($G = (V, E), (v_1, v_2, v_n)$)

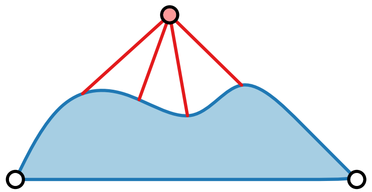
forall $v \in V$ **do**
  chords(v) $\leftarrow$ 0; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false
mark(v_1), mark(v_2), out(v_1), out(v_2), out(v_n) $\leftarrow$ true
for $k = n$ **to** 3 **do**
  choose v such that mark(v) = false, out(v) = true,
   and chords(v) = 0   *// keep list with candidates*
  $v_k \leftarrow v$; mark(v) $\leftarrow$ true
  // Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1} in G_{k-1} and let $w_p, \ldots, w_q$ be the neighbors of v_k
  out(w_i) $\leftarrow$ true for all $p < i < q$ *// $O(n)$ in total*
  update number of chords for w_i
  and its neighbours   *// $O(m) = O(n)$ in total*

- chord(v): # chords adjacent to v
- out(v) = true iff v is currently outer vertex
- mark(v) = true iff v has received its number

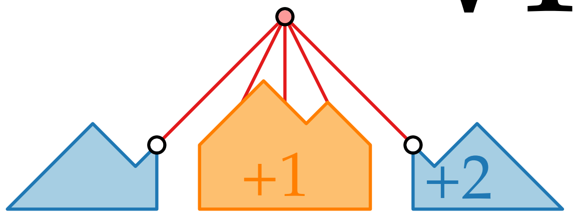


Lemma.
Algorithm CanonicalOrder computes a canonical order of a plane graph in $\mathcal{O}(n)$ time.

Universität Trier

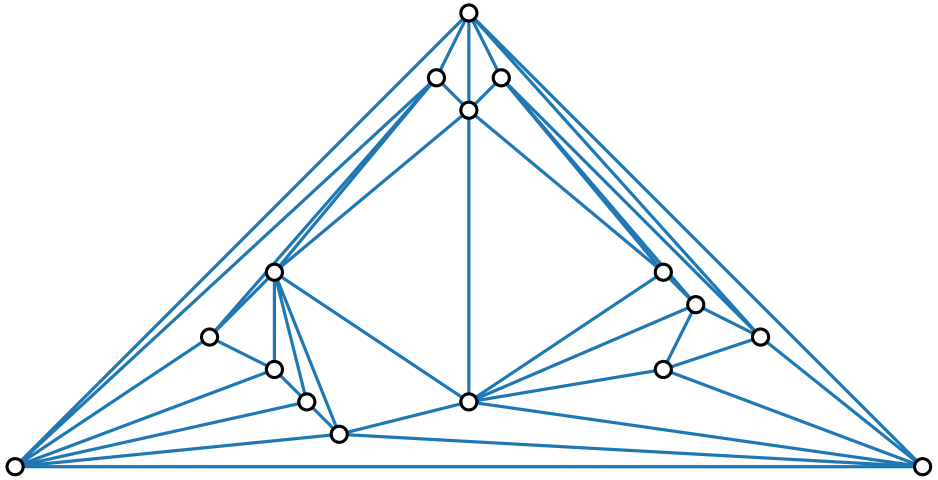


Visualization of Graphs



Lecture 4:
Straight-Line Drawings of Planar Graphs I: Canonical Ordering and Shift Method

Part IV:
Shift Method



Philipp Kindermann

Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

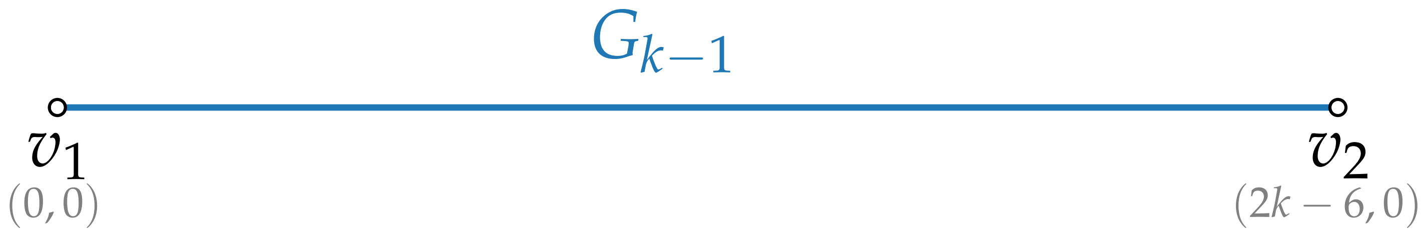
G_{k-1}

Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

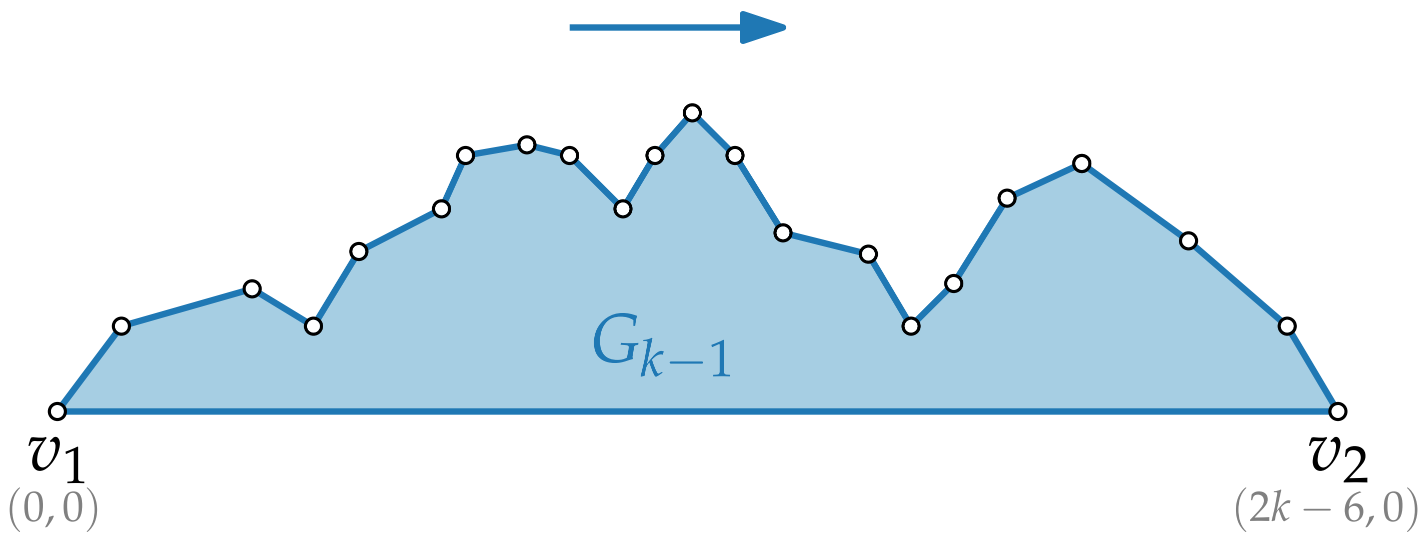
- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,



Shift Method – Idea

Drawing invariants:
G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone,

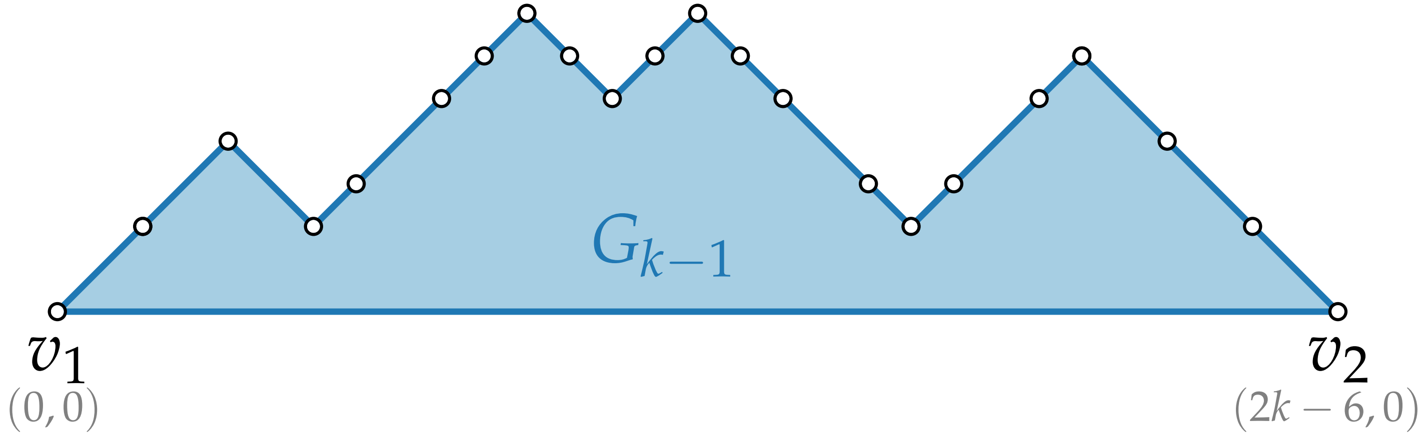


Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1.

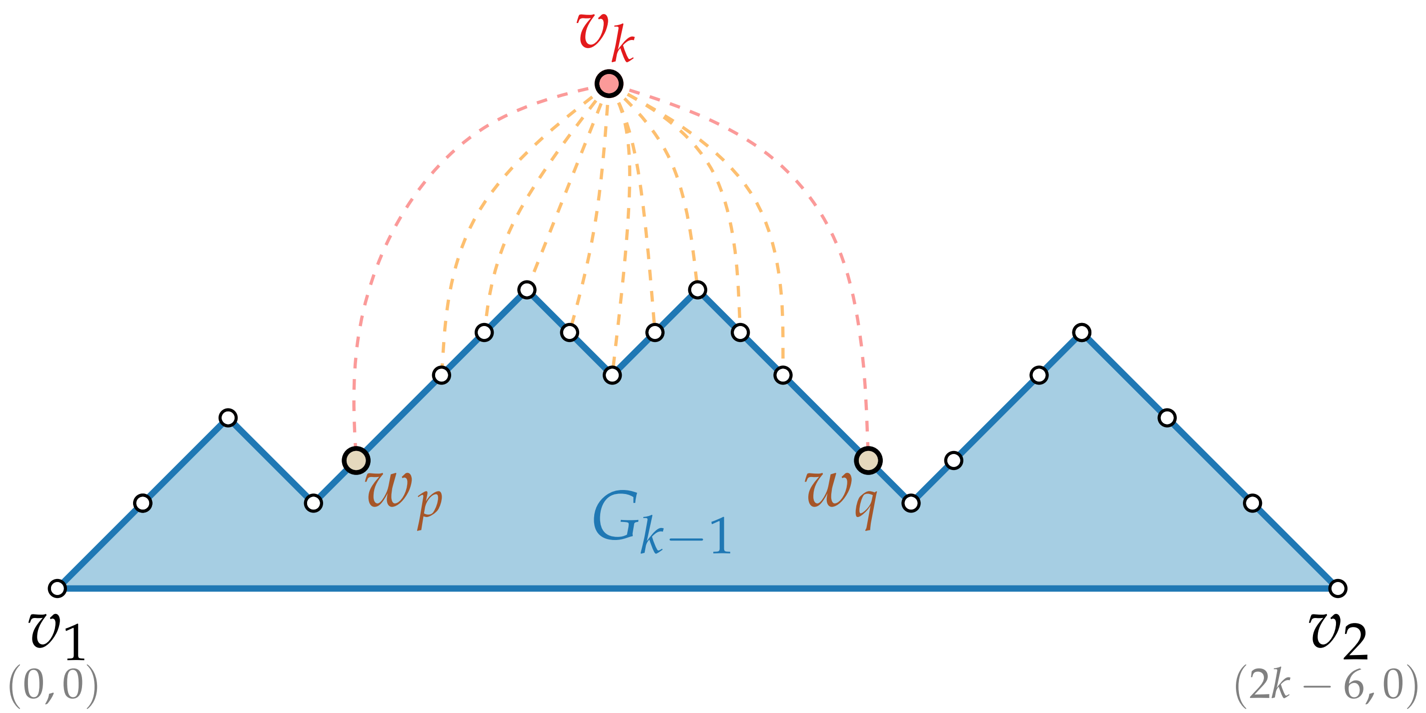


Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1.

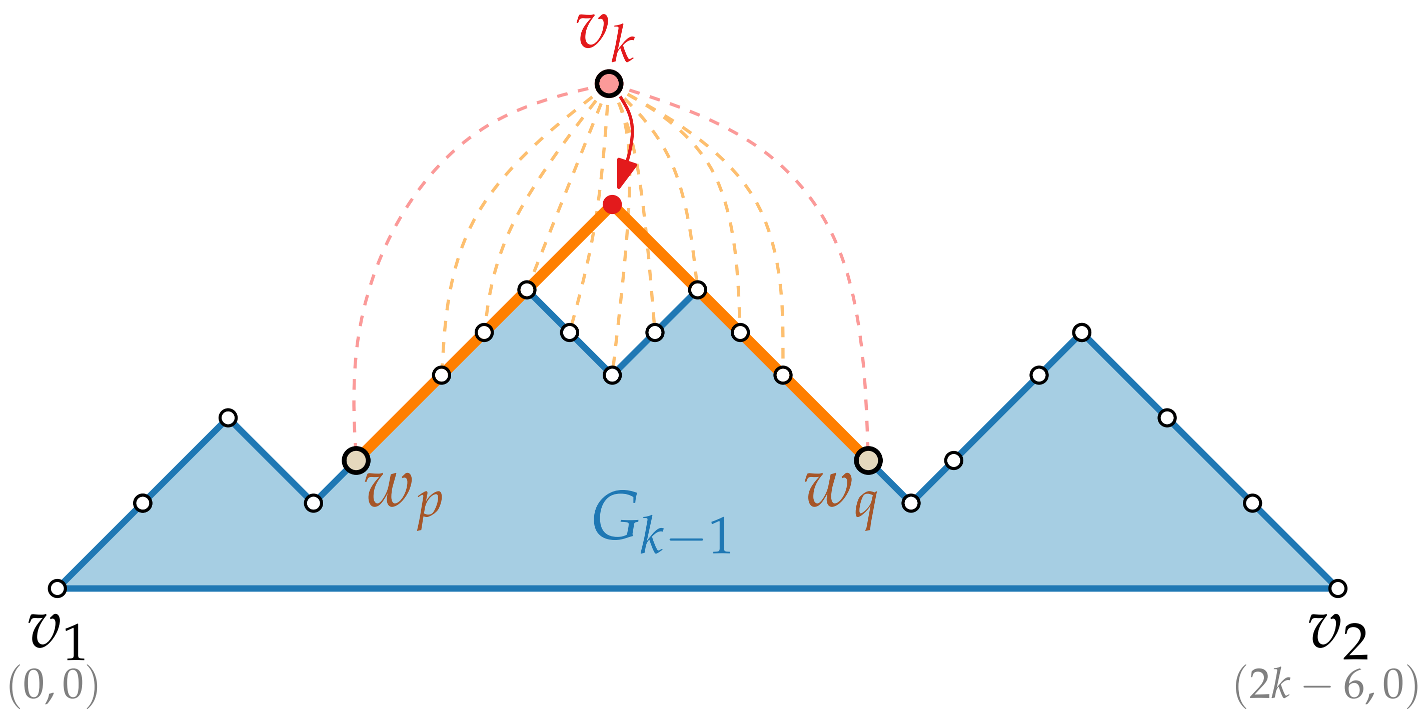


Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1.

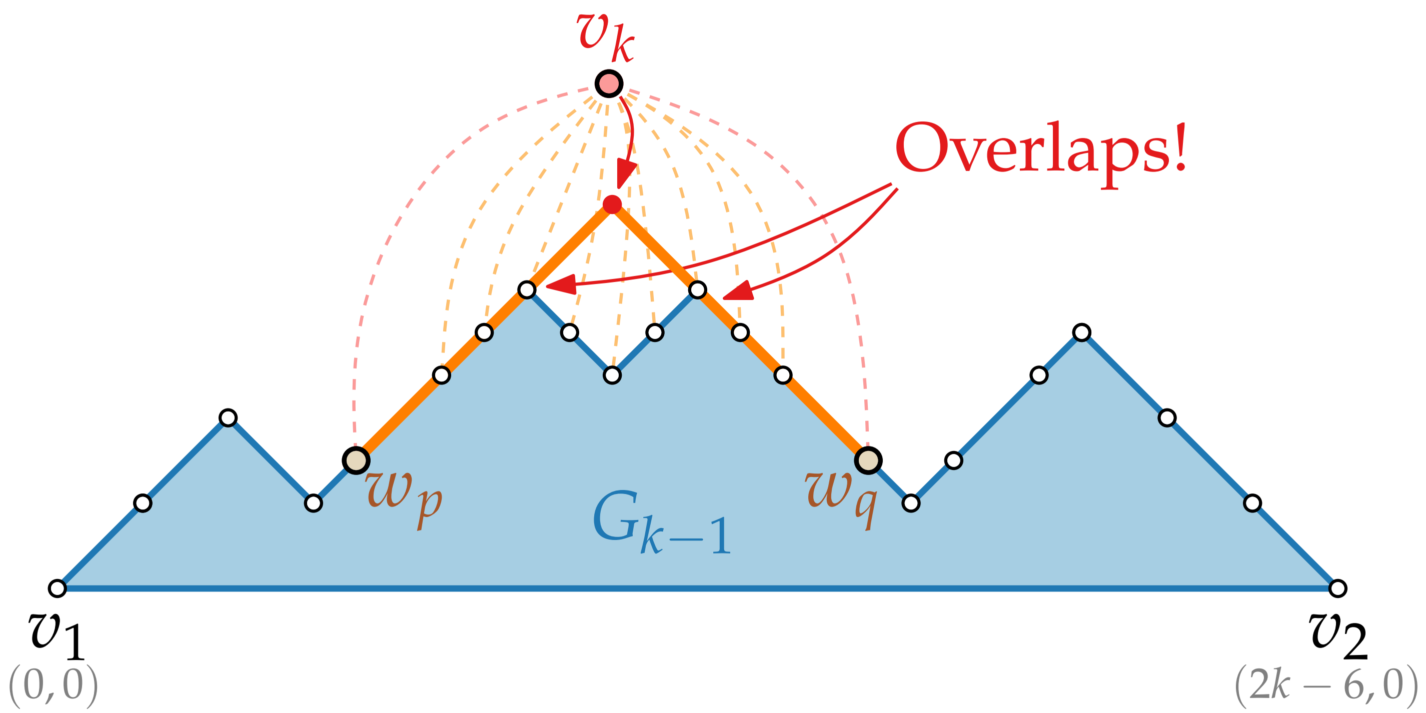


Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1.

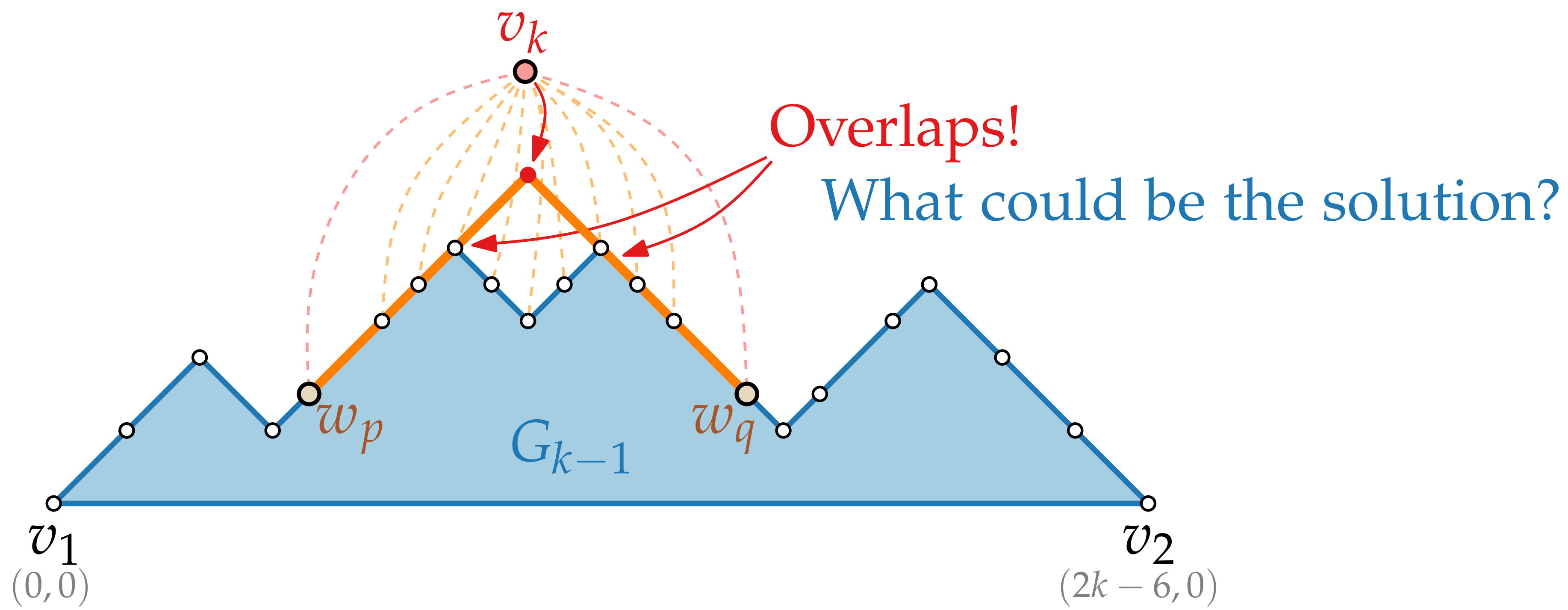


Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1.

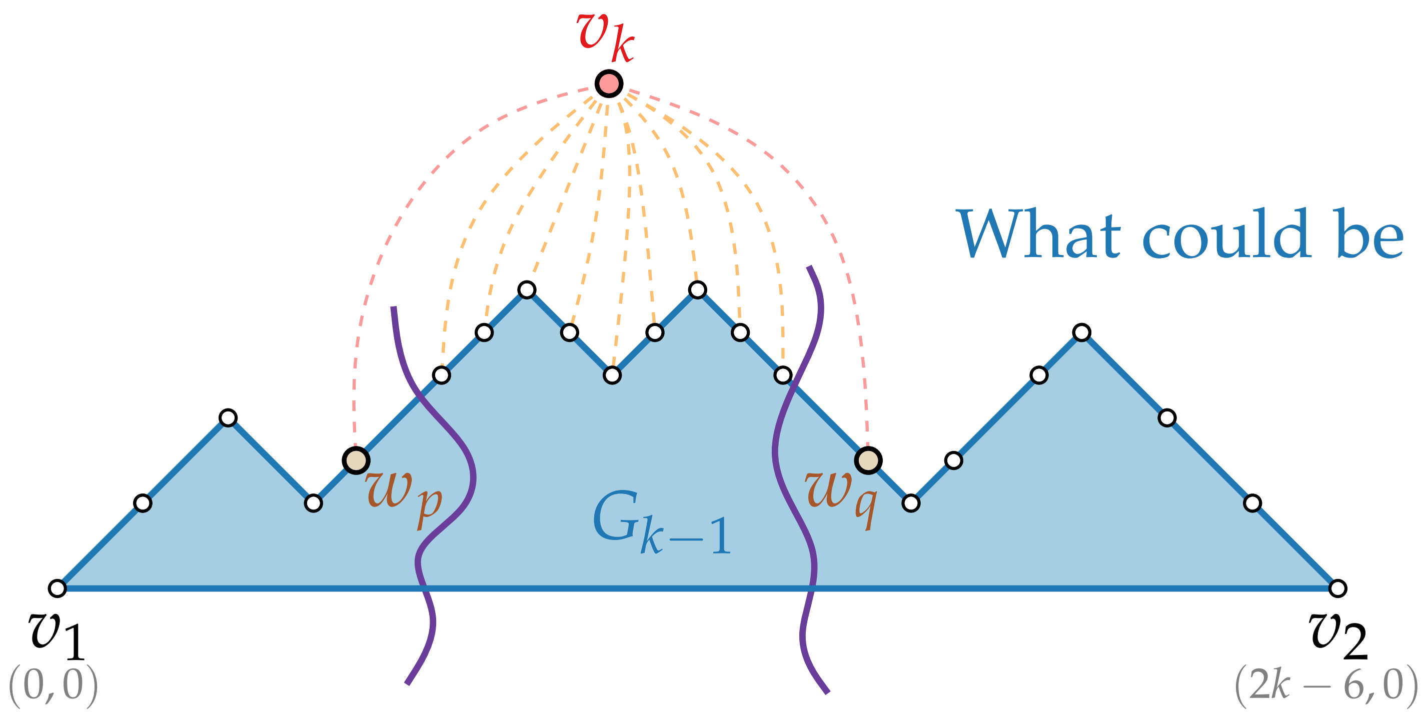


Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1.



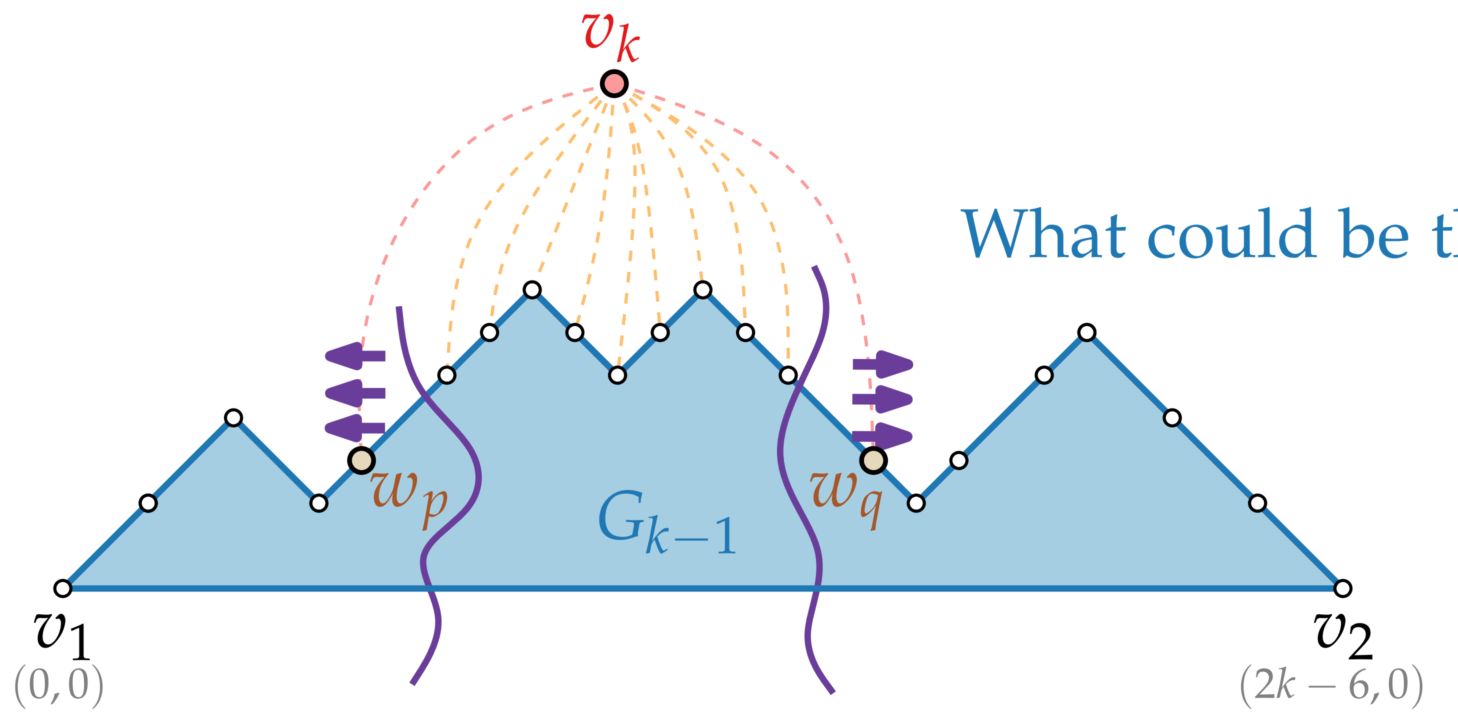
What could be the solution?

Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1.



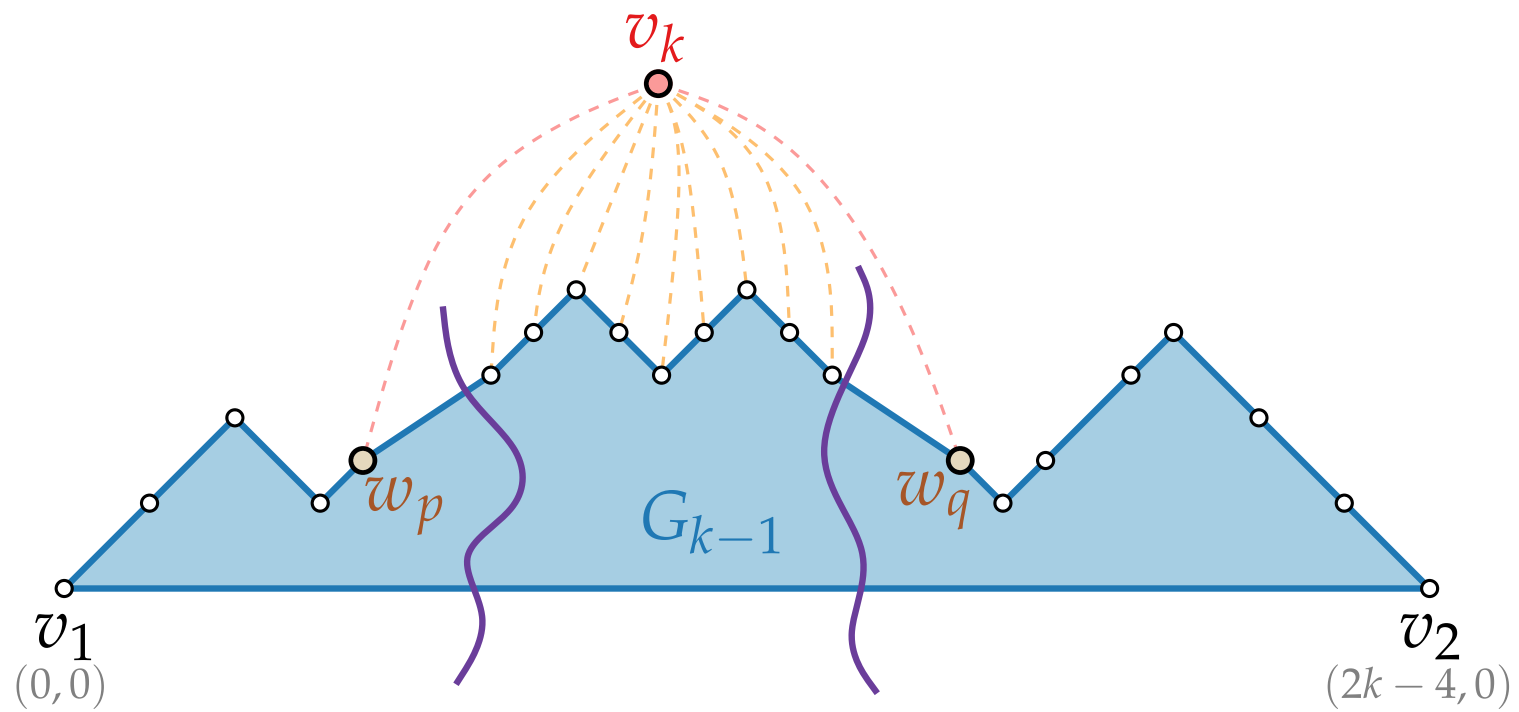
What could be the solution?

Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1.

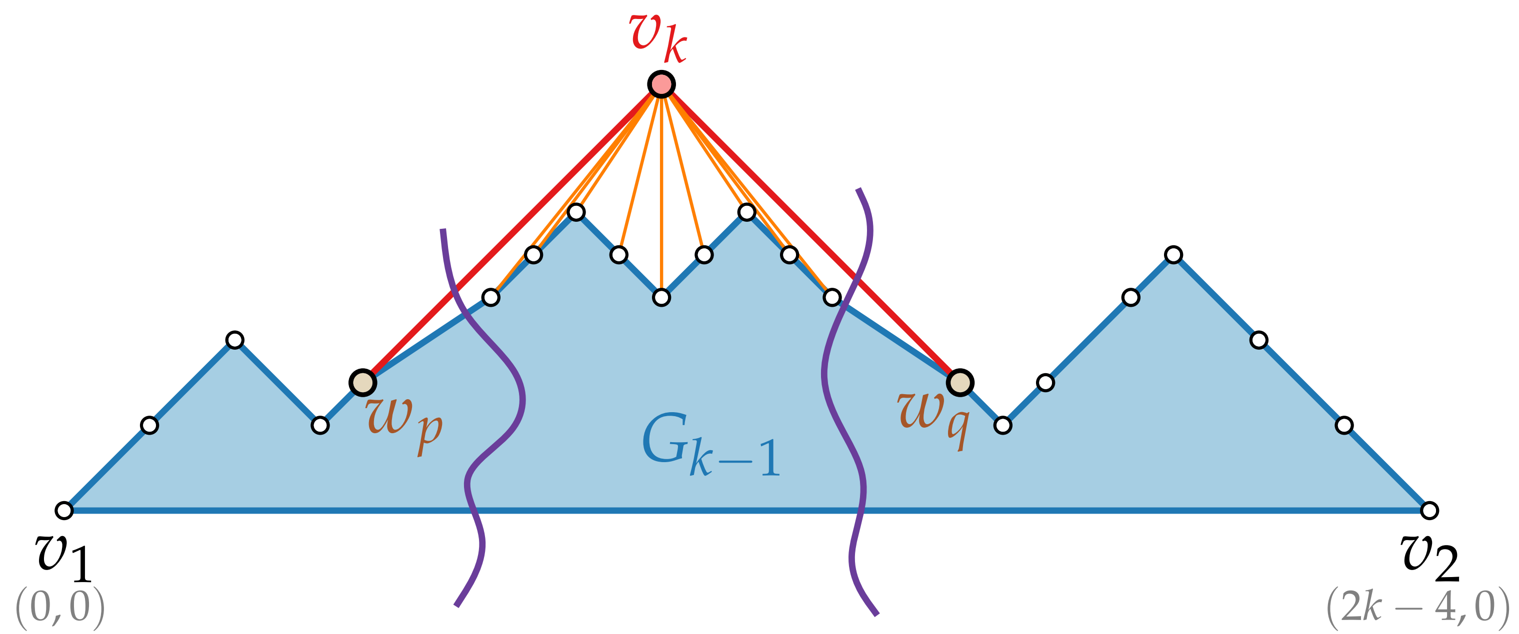


Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1.



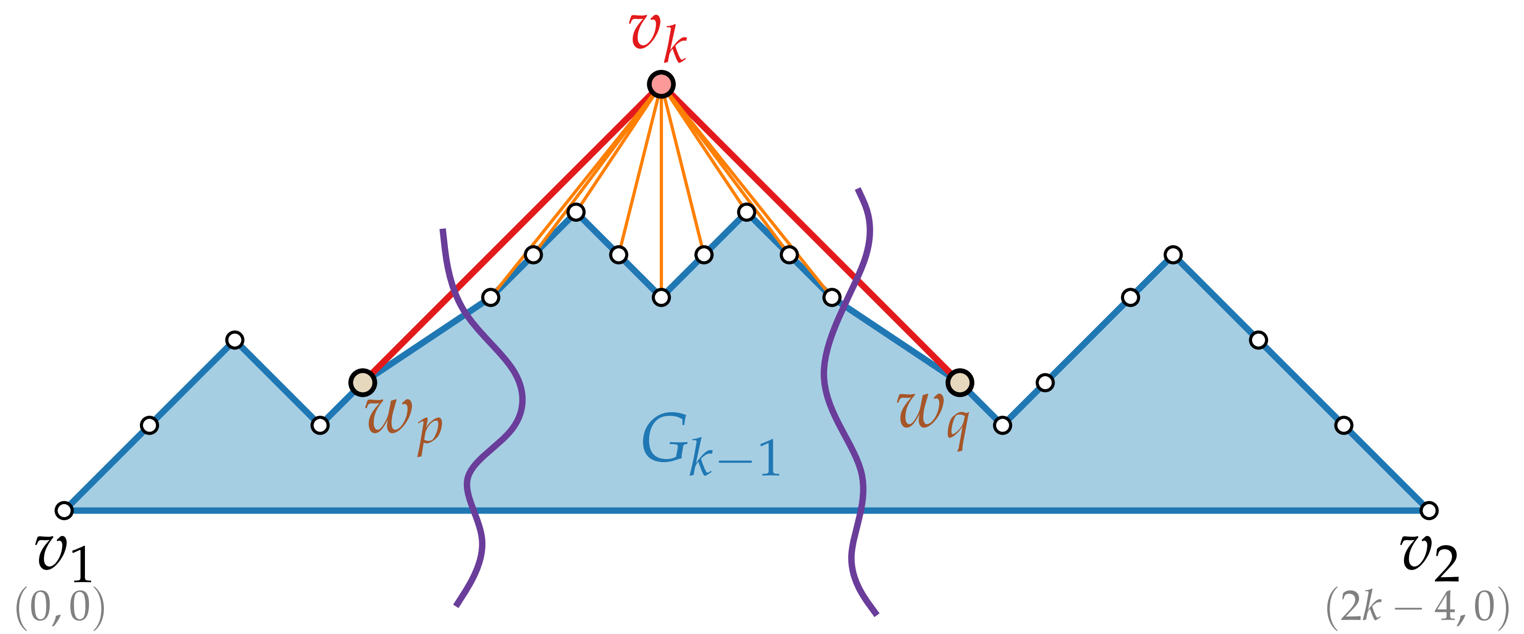
Shift Method – Idea

Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6, 0)$,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1.

Does v_k land on grid?



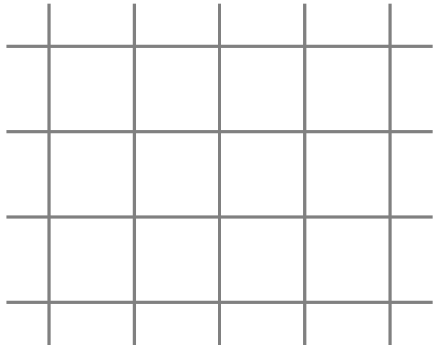
Shift Method – Idea

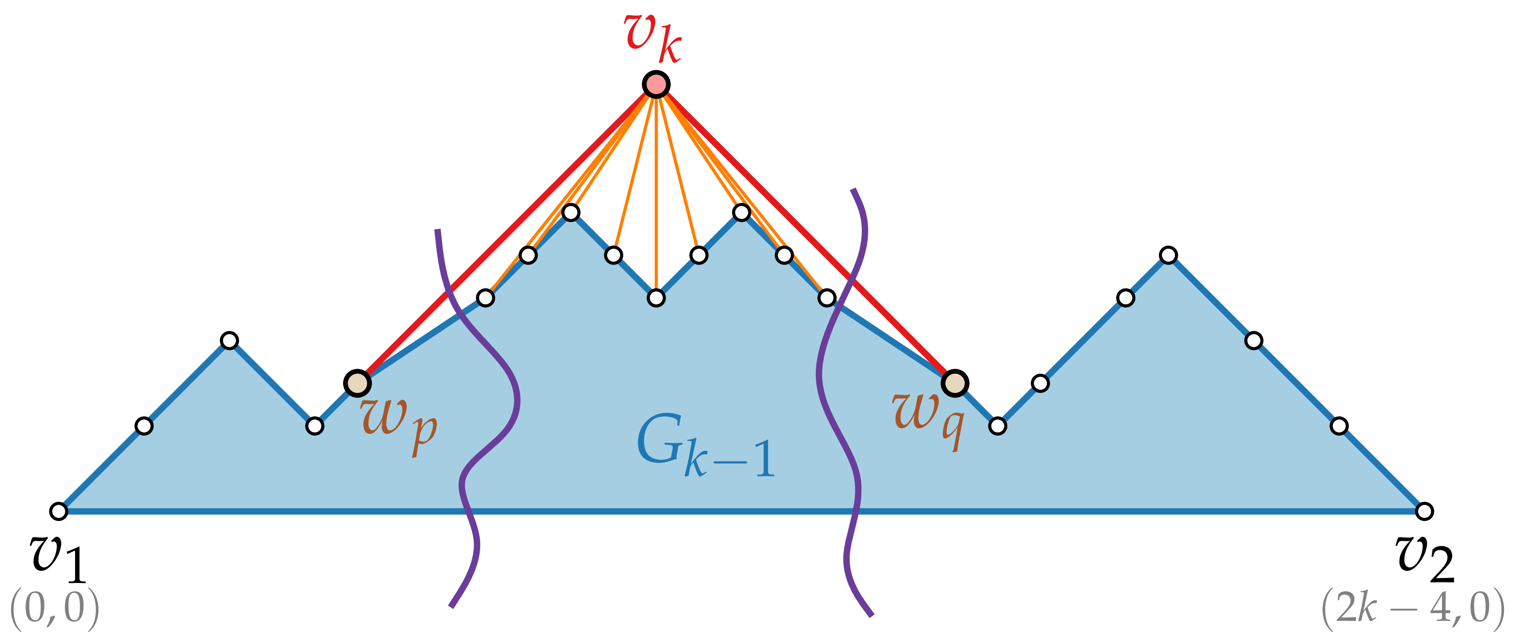
Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1.

Does v_k land on grid?





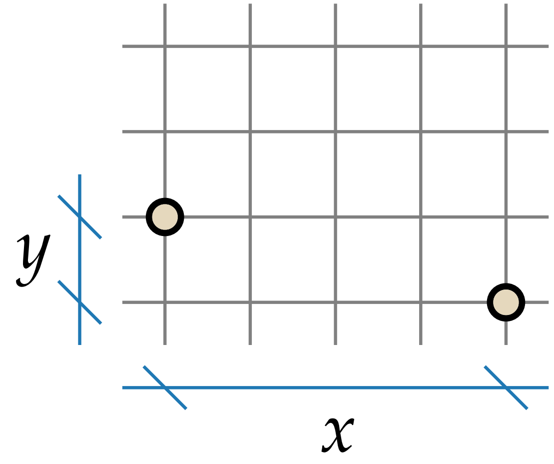
Shift Method – Idea

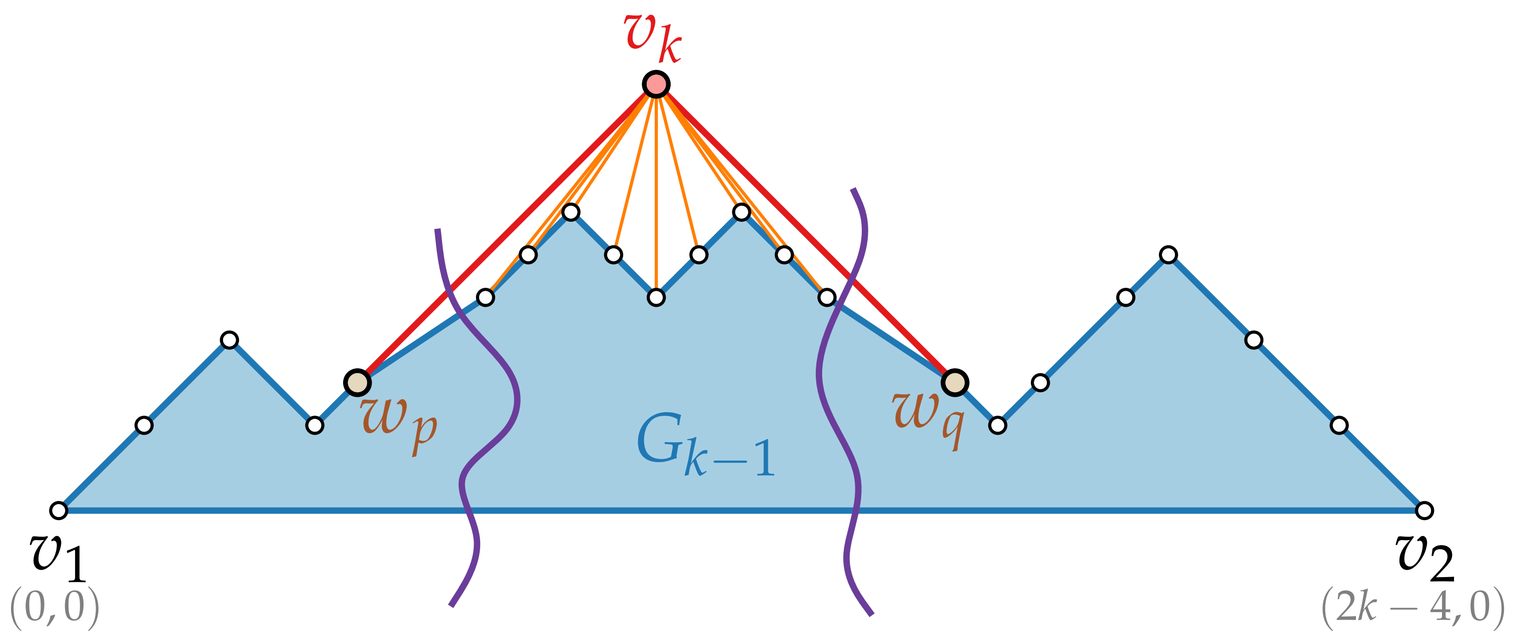
Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1.

Does v_k land on grid?





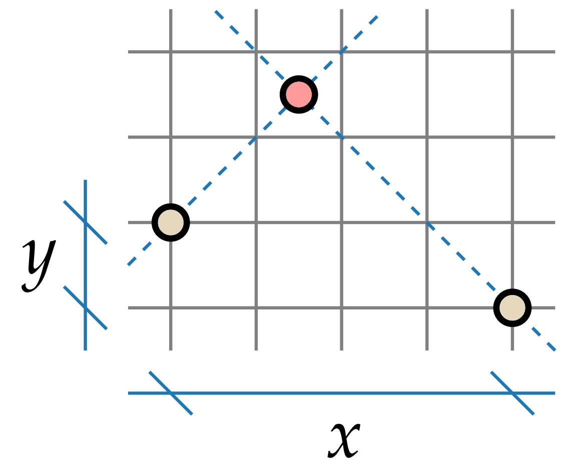
Shift Method – Idea

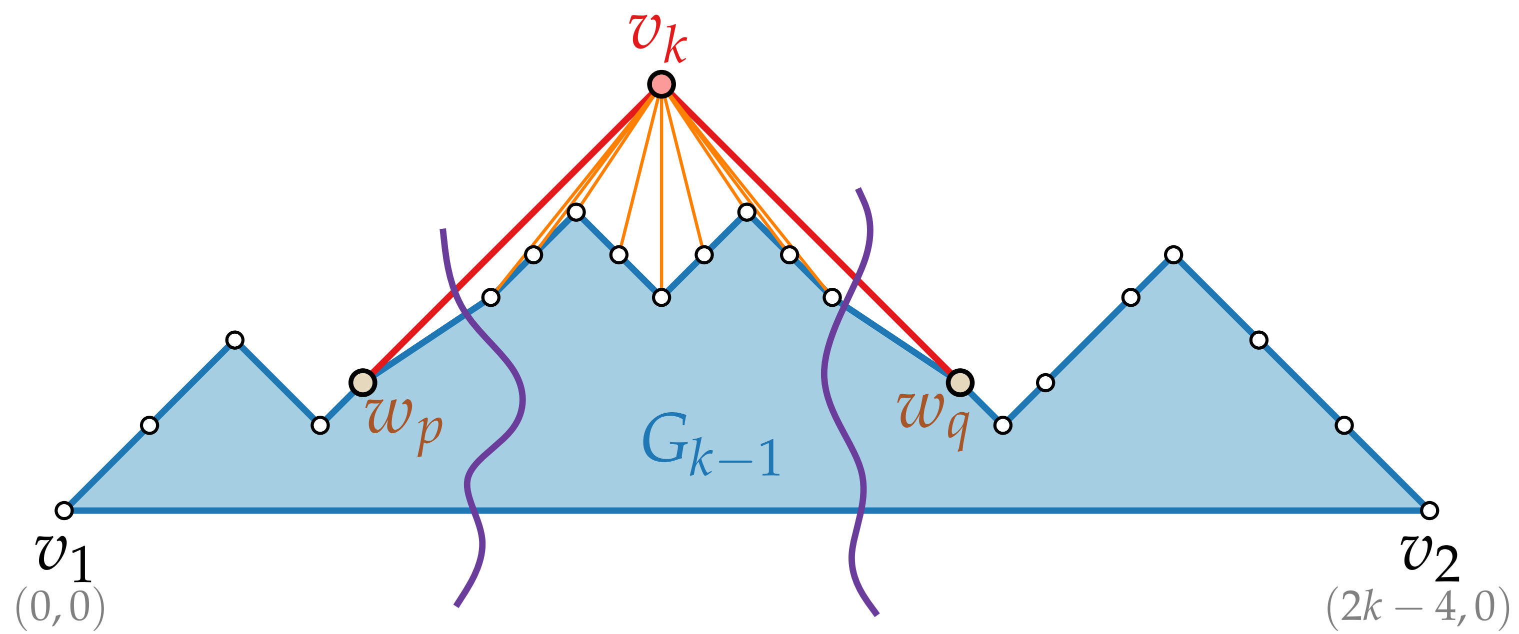
Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1.

Does v_k land on grid?





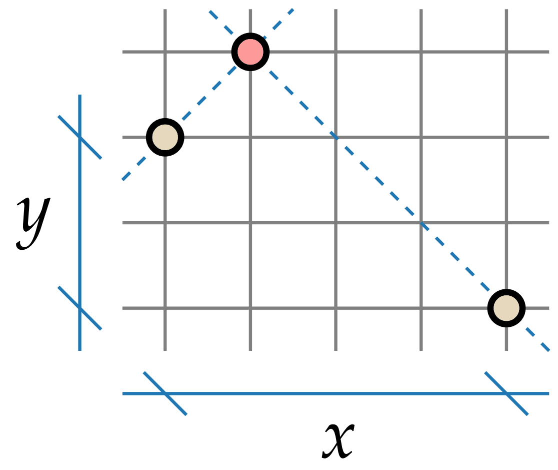
Shift Method – Idea

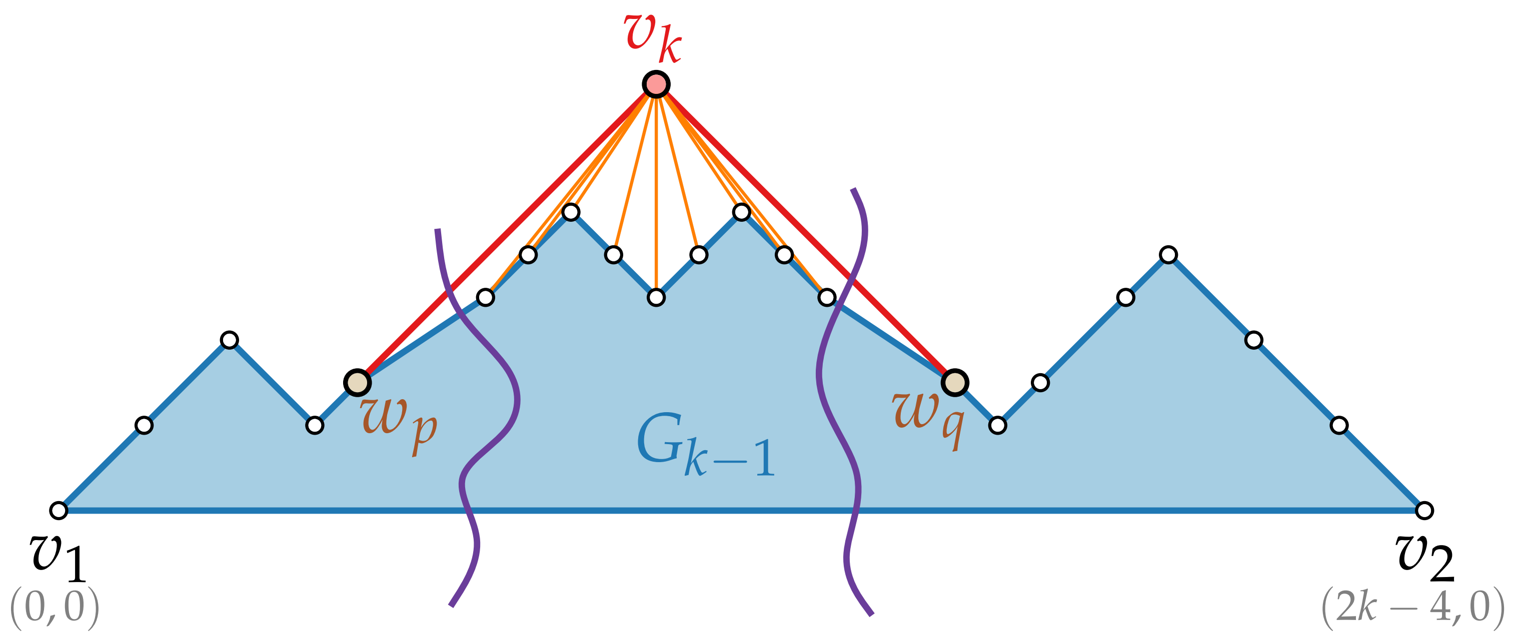
Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1.

Does v_k land on grid?





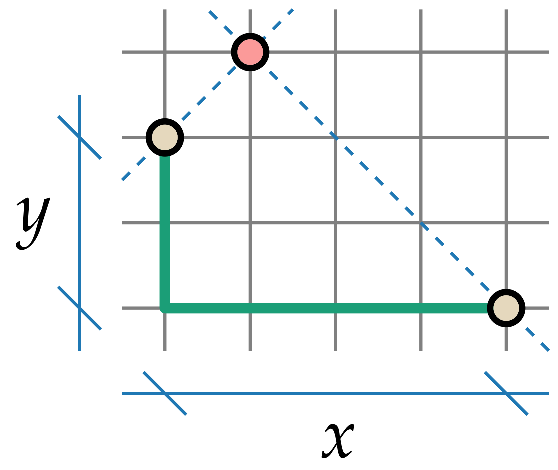
Shift Method – Idea

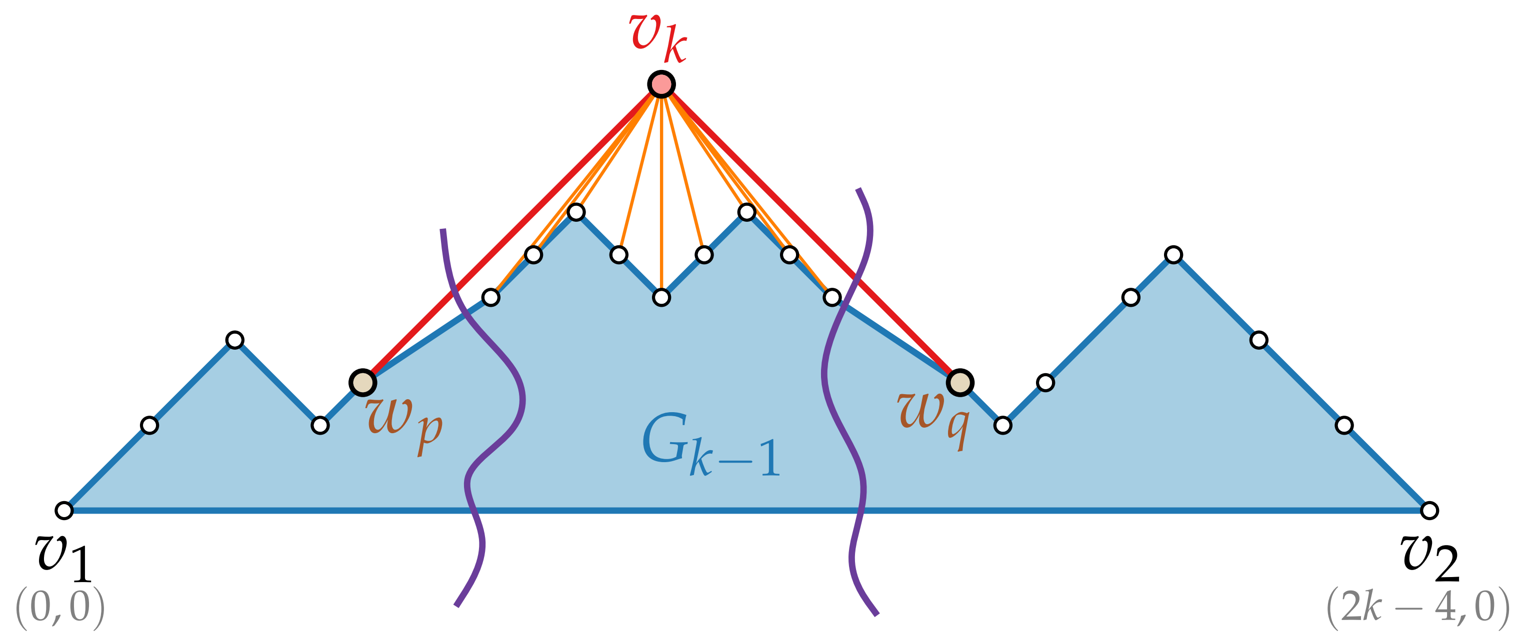
Drawing invariants:

G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1.

Does v_k land on grid?





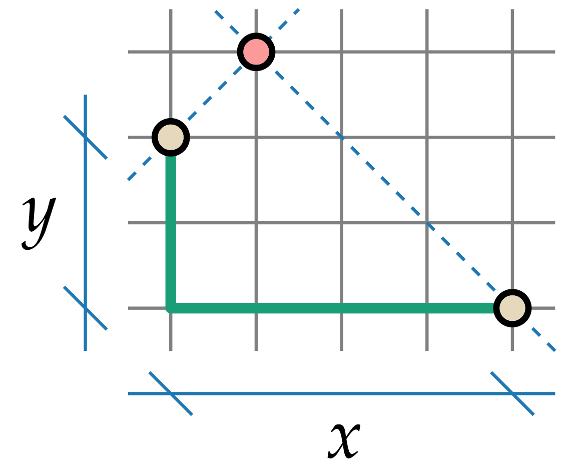
Shift Method – Idea

Drawing invariants:

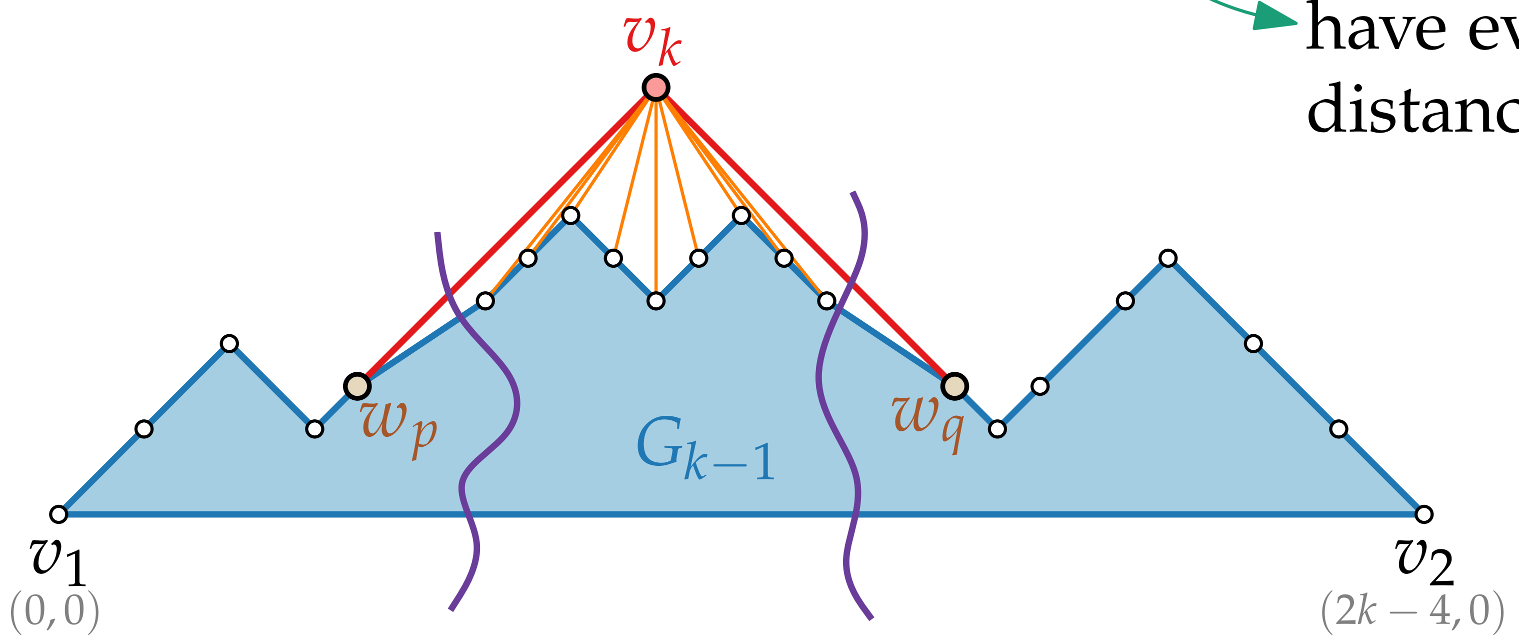
G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1.

Does v_k land on grid?



yes, beause w_p and w_q have even Manhattan distance



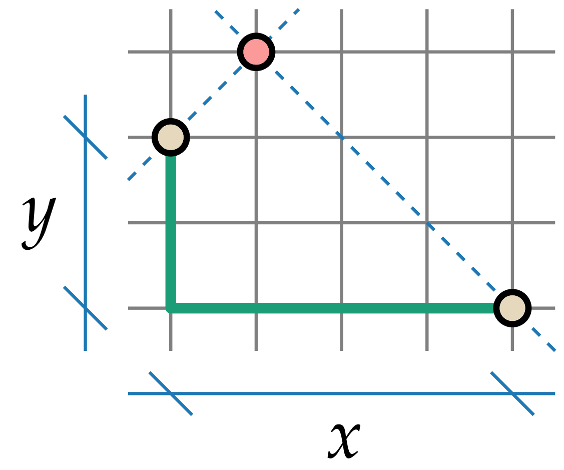
Shift Method – Idea

Drawing invariants:

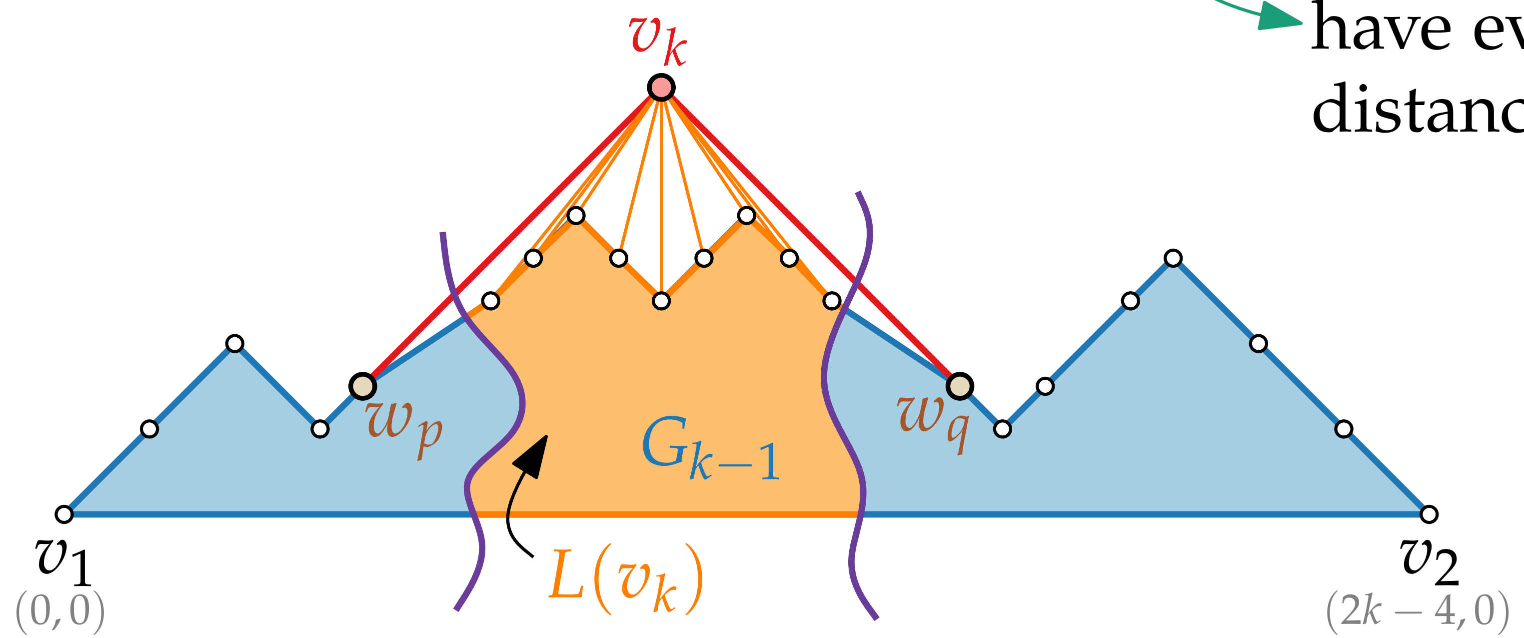
G_{k-1} is drawn such that

- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$,
- boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone,
- each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1.

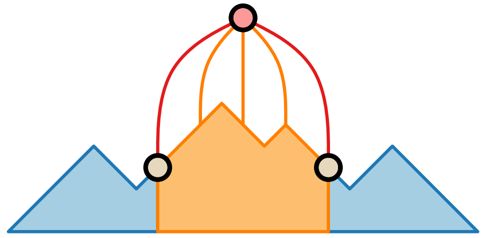
Does v_k land on grid?



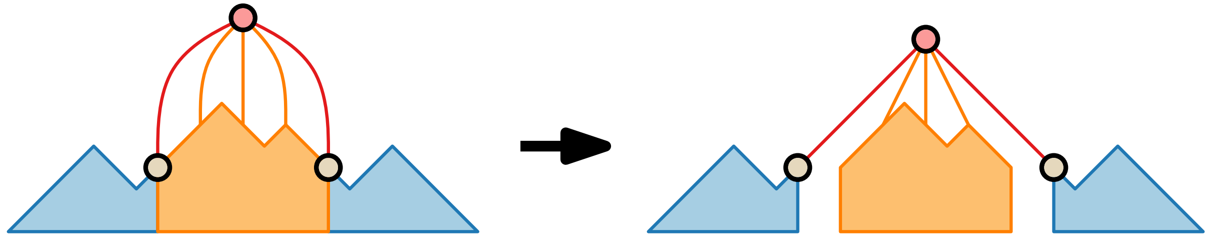
yes, beause w_p and w_q have even Manhattan distance



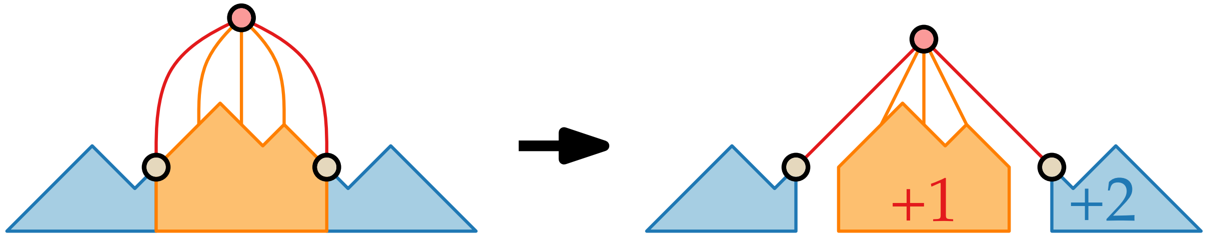
Shift Method – Example



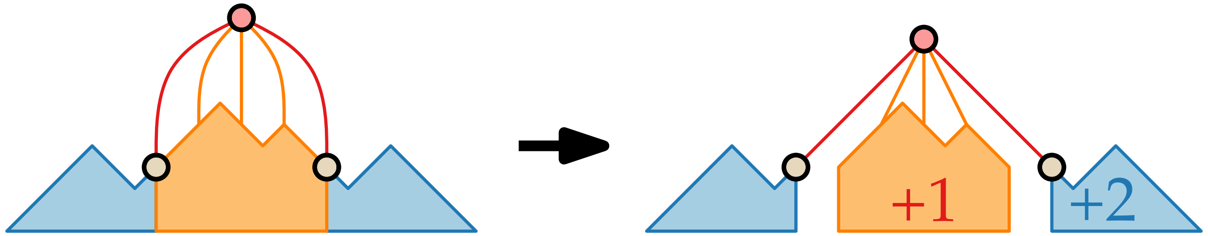
Shift Method – Example

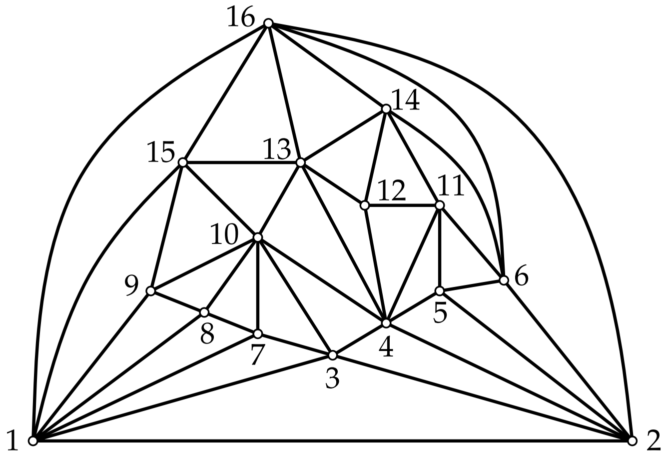


Shift Method – Example

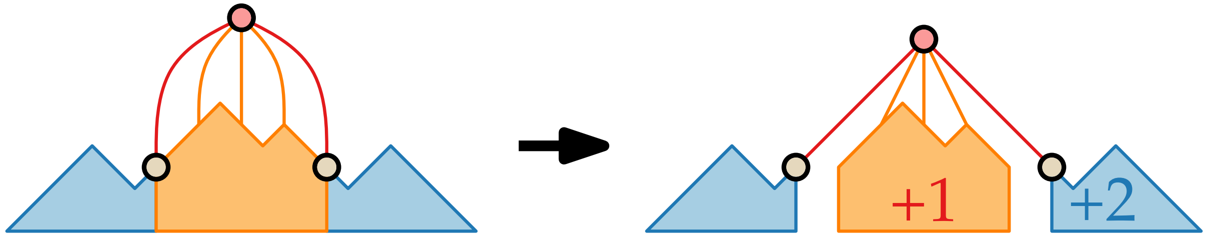


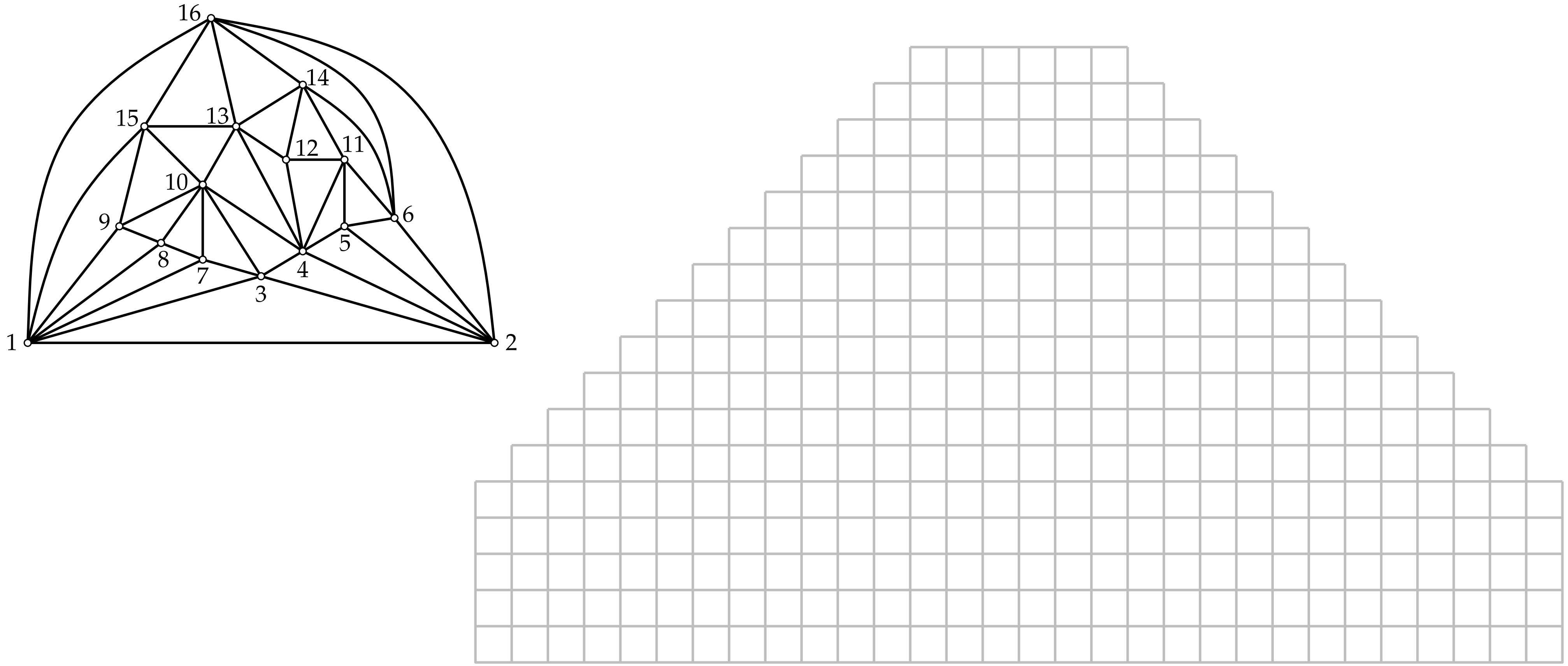
Shift Method – Example



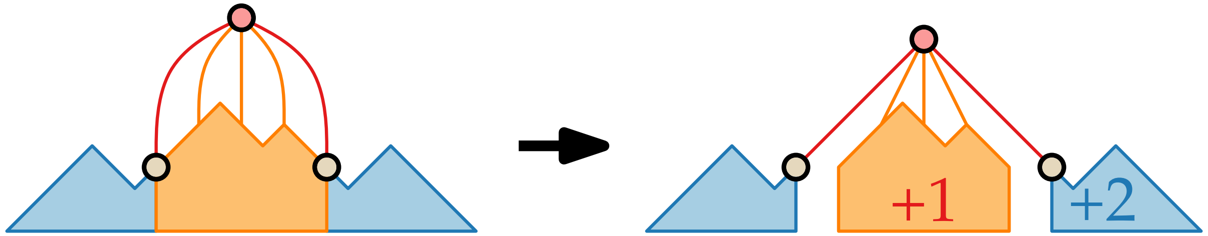


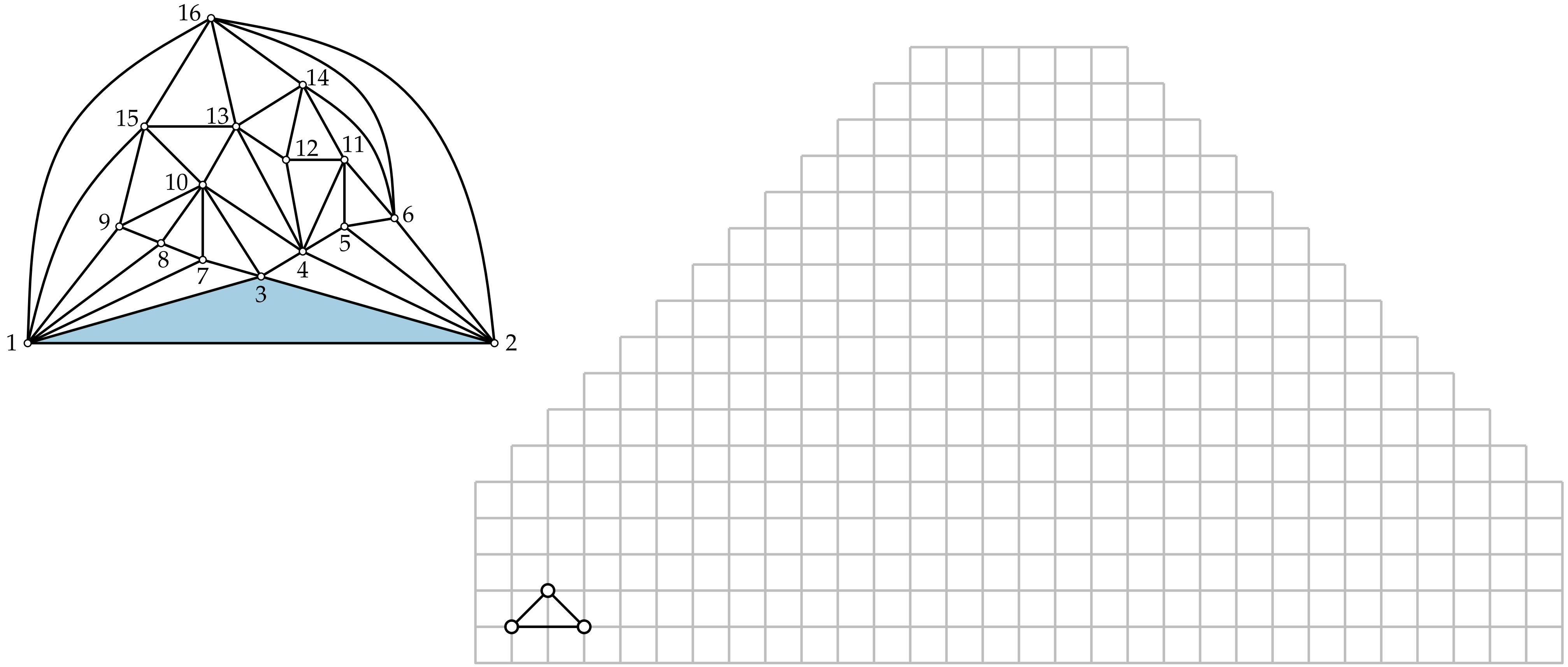
Shift Method – Example



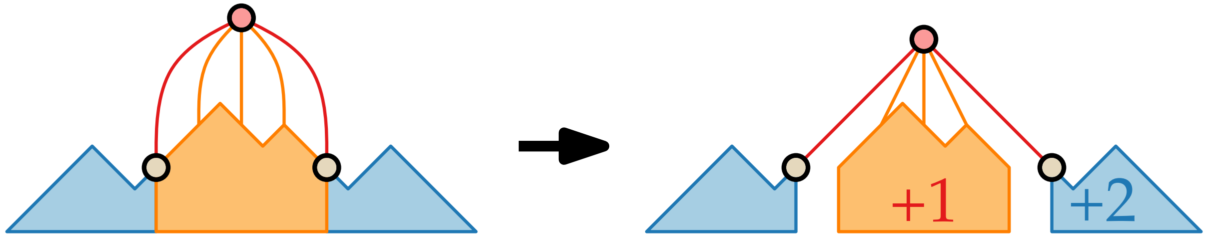


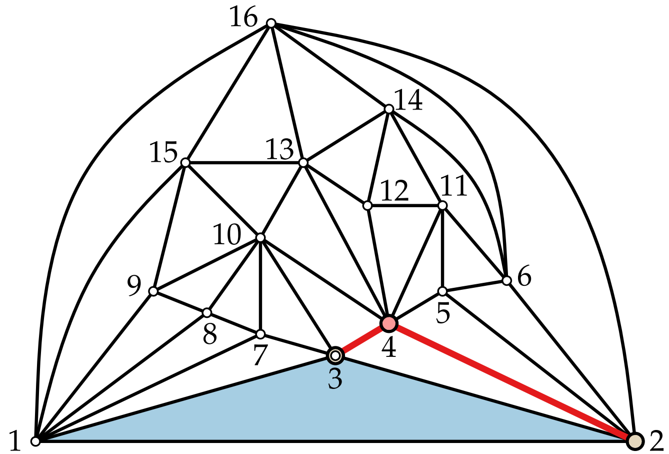
Shift Method – Example

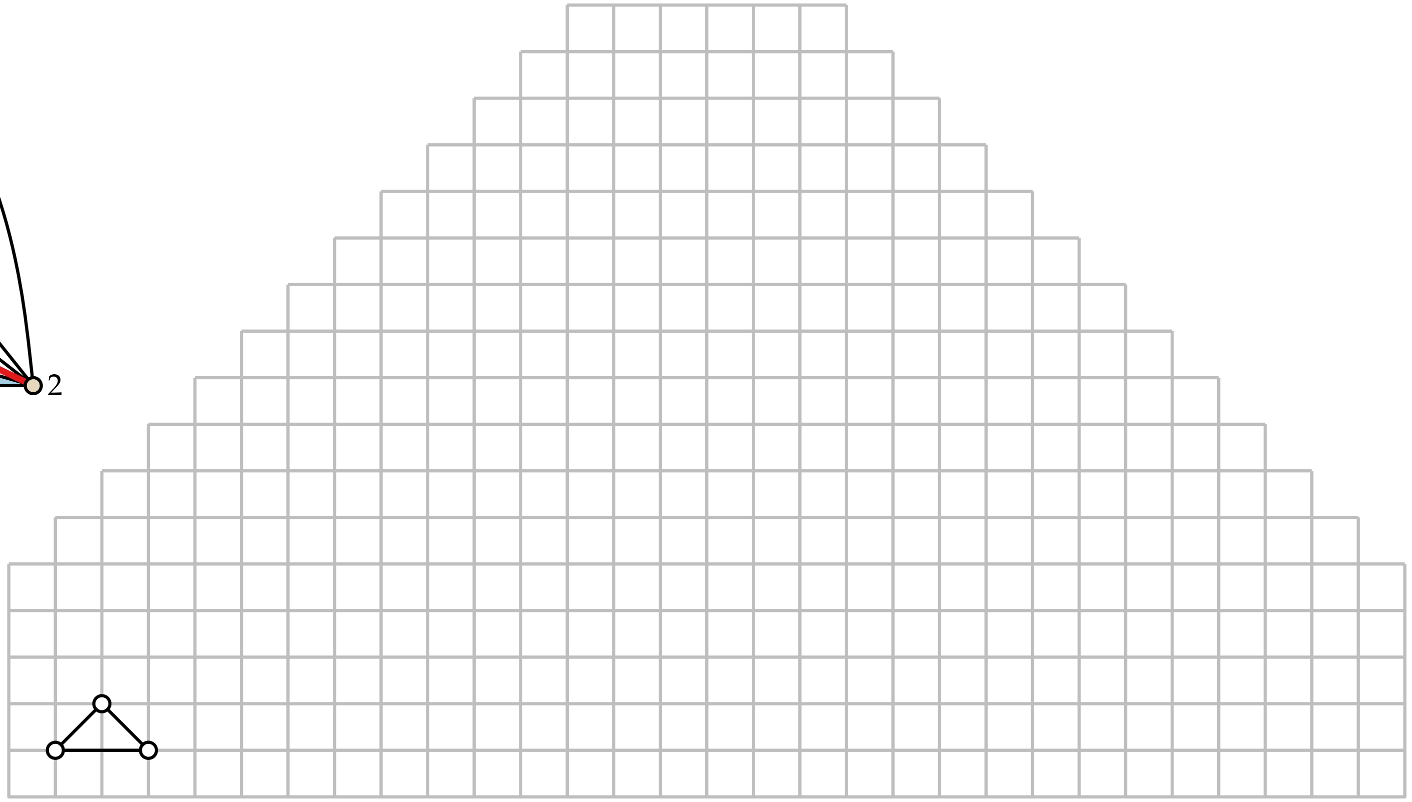




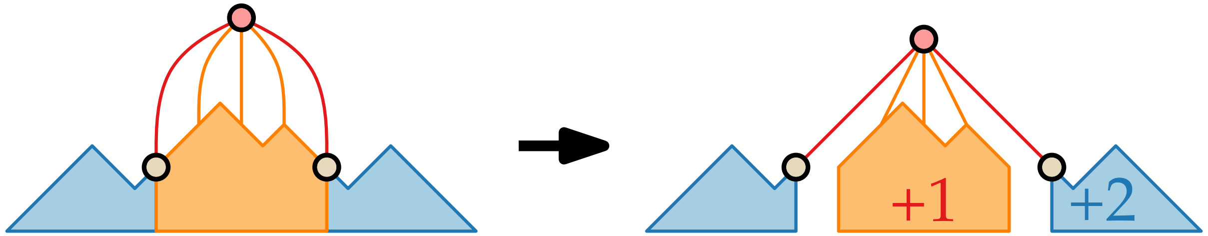
Shift Method – Example

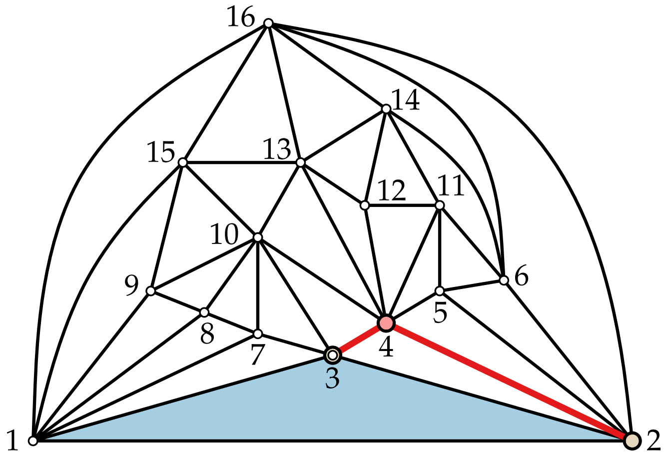


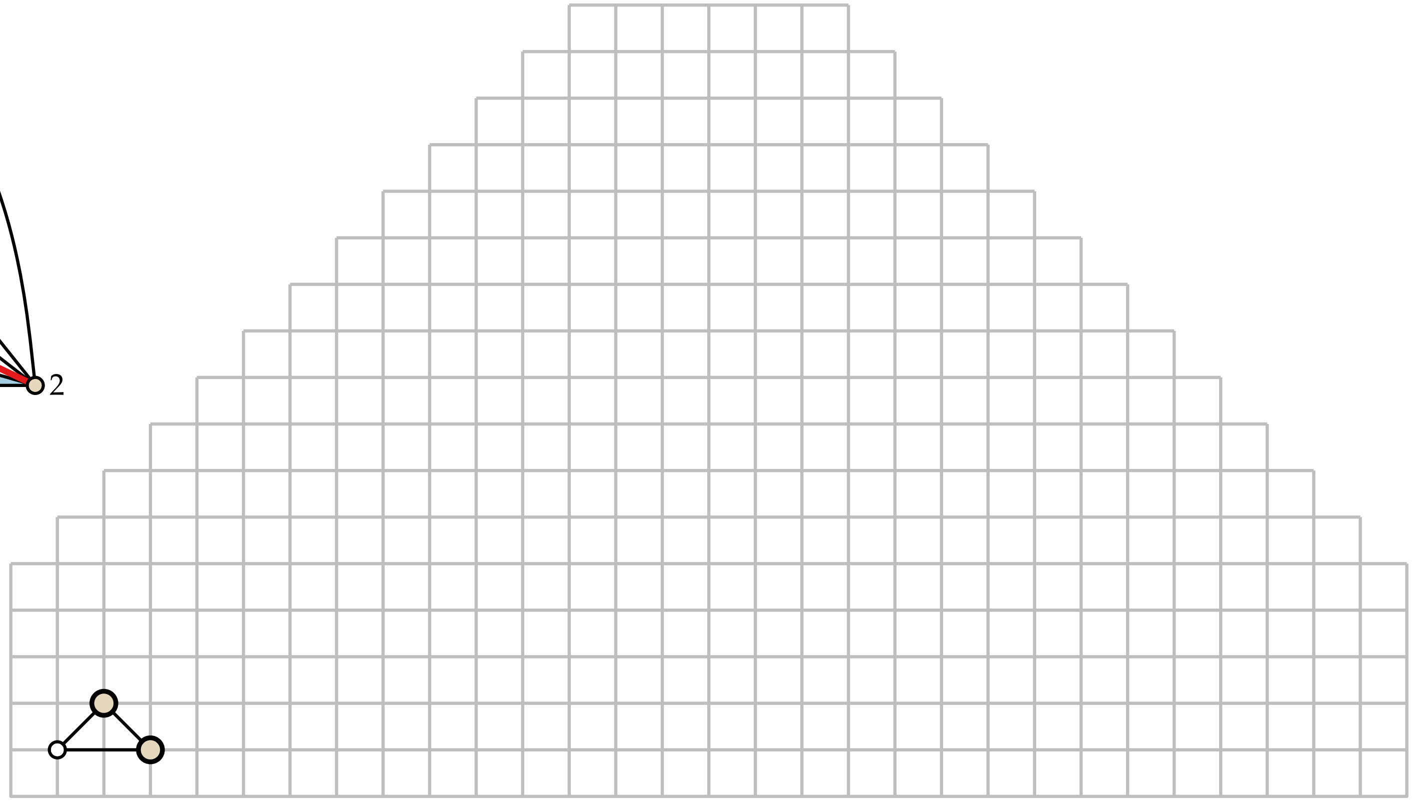




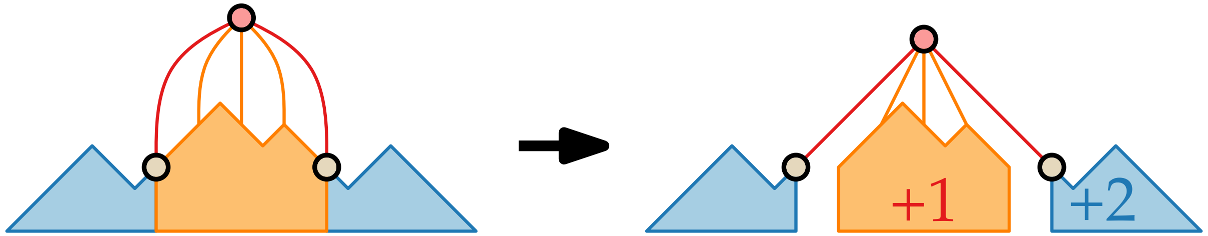
Shift Method – Example

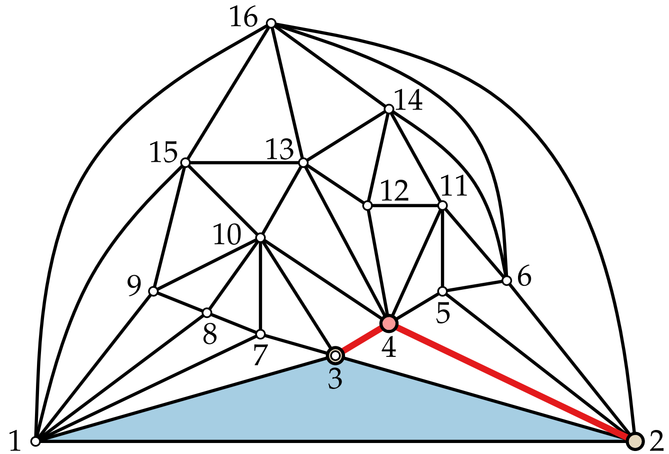


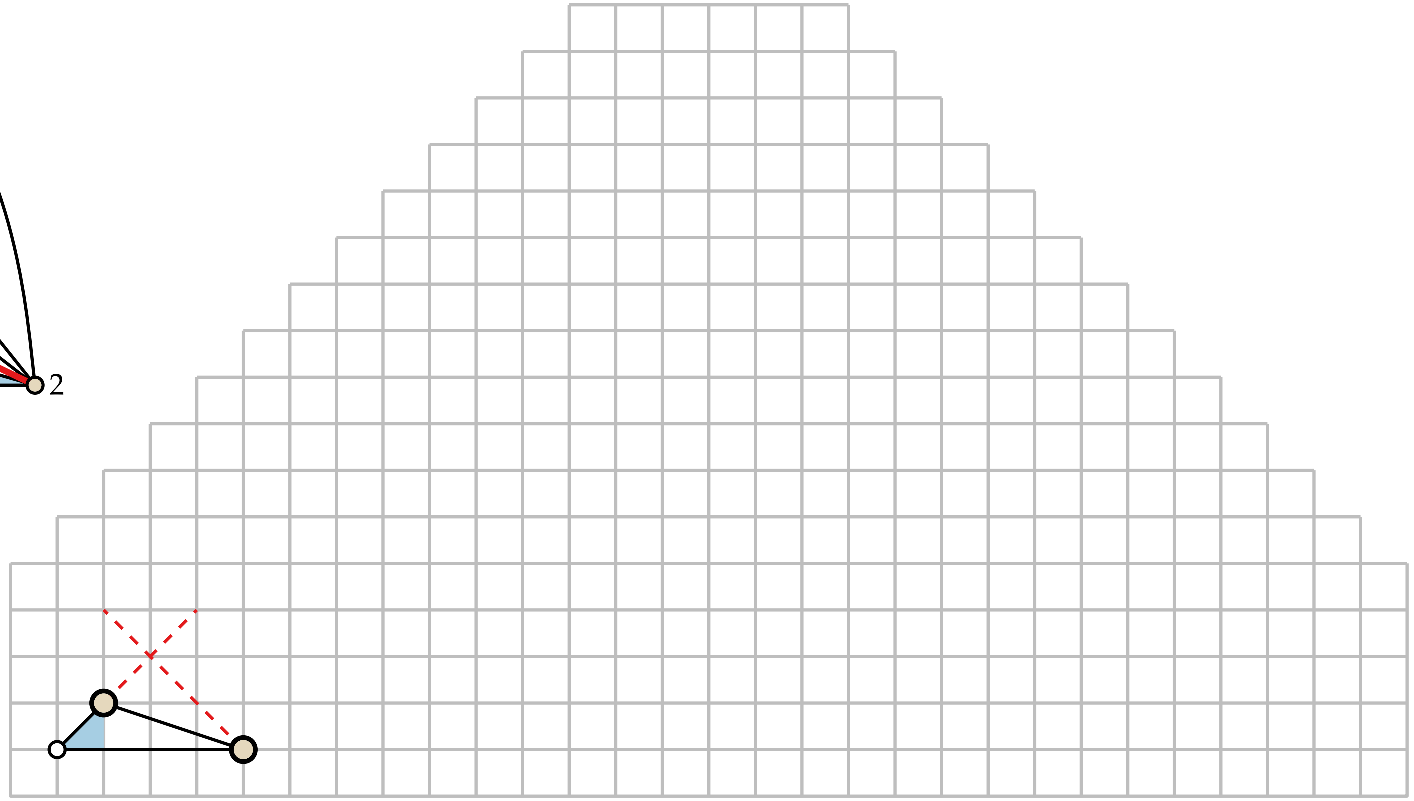




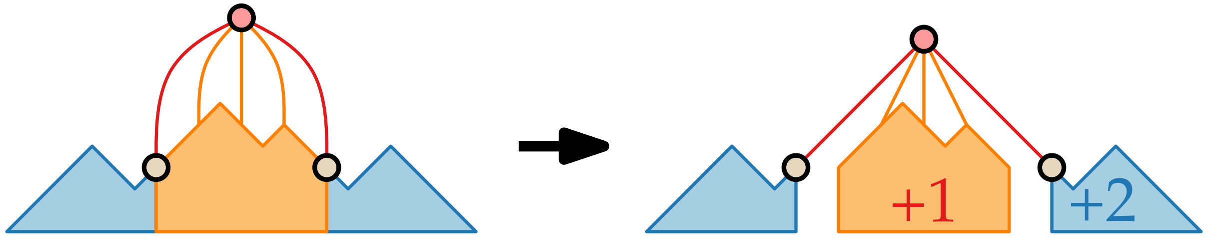
Shift Method – Example

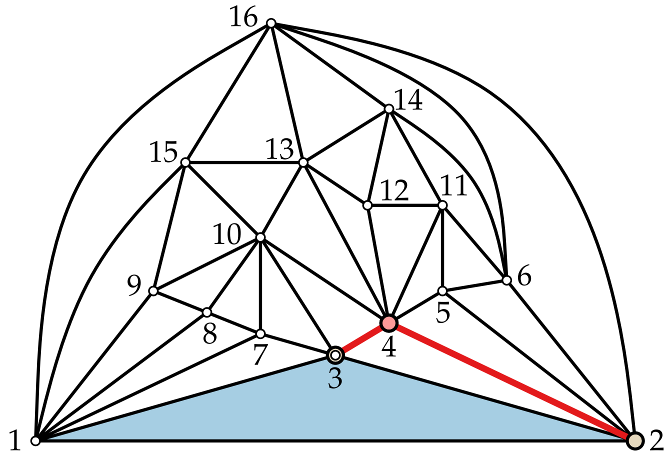


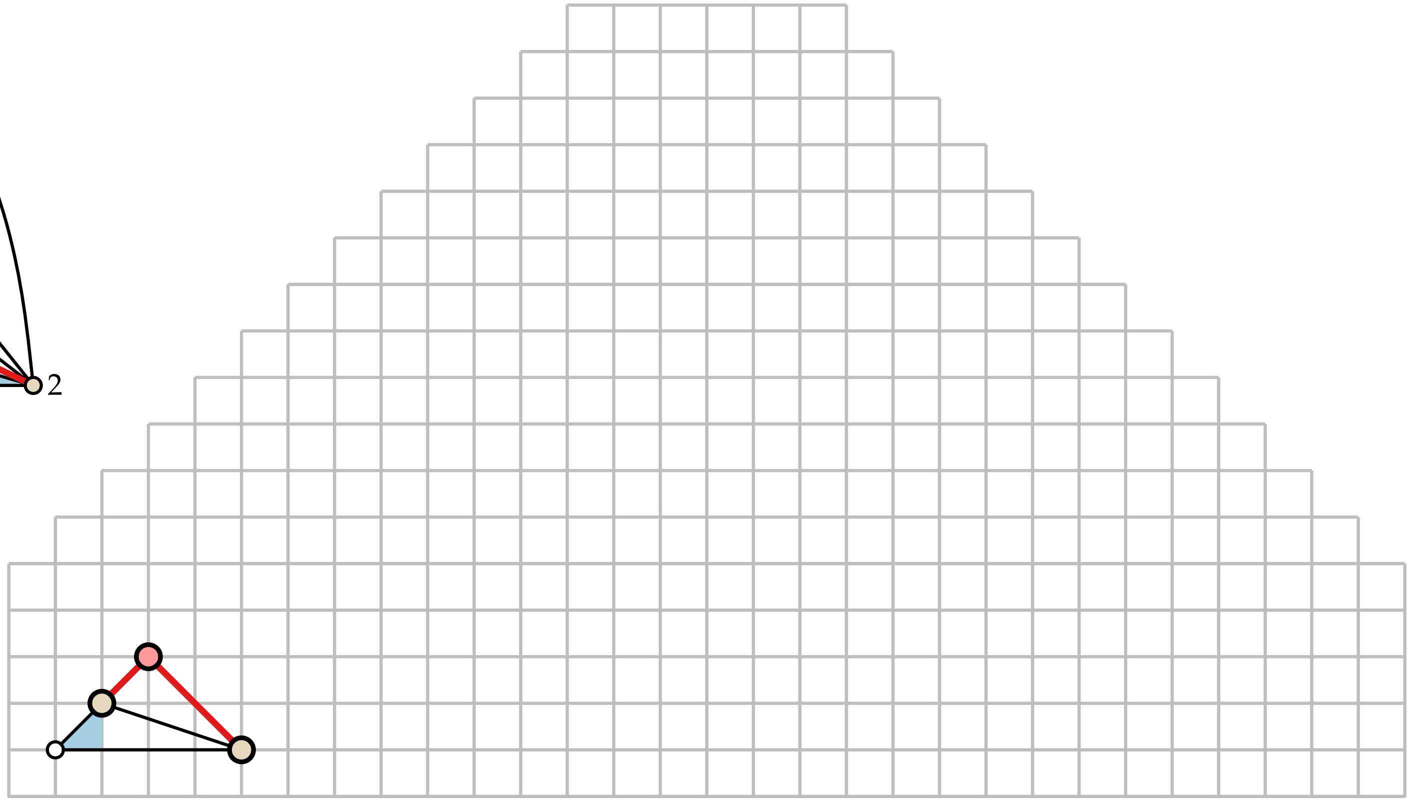




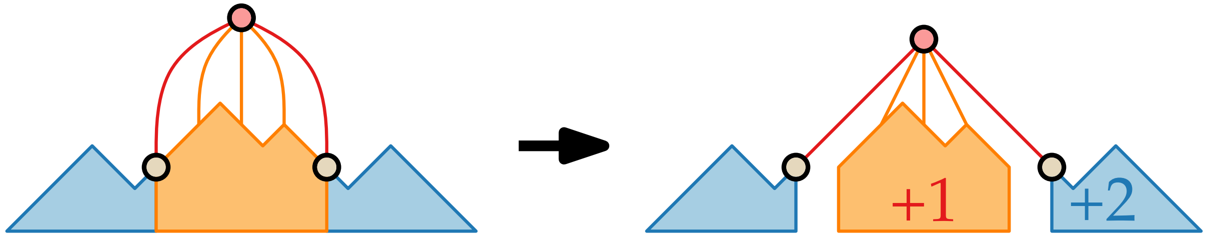
Shift Method – Example

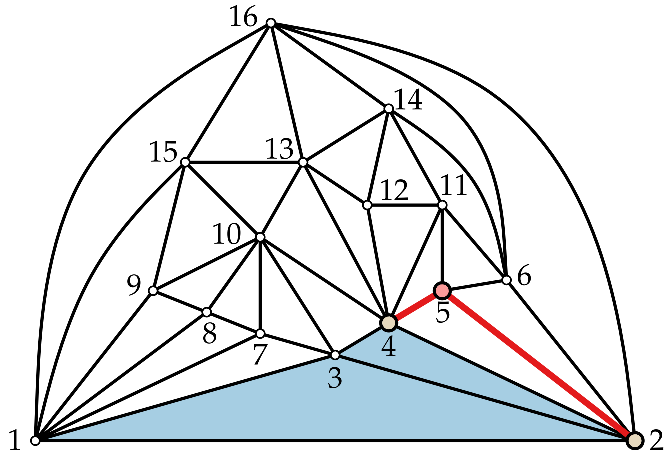


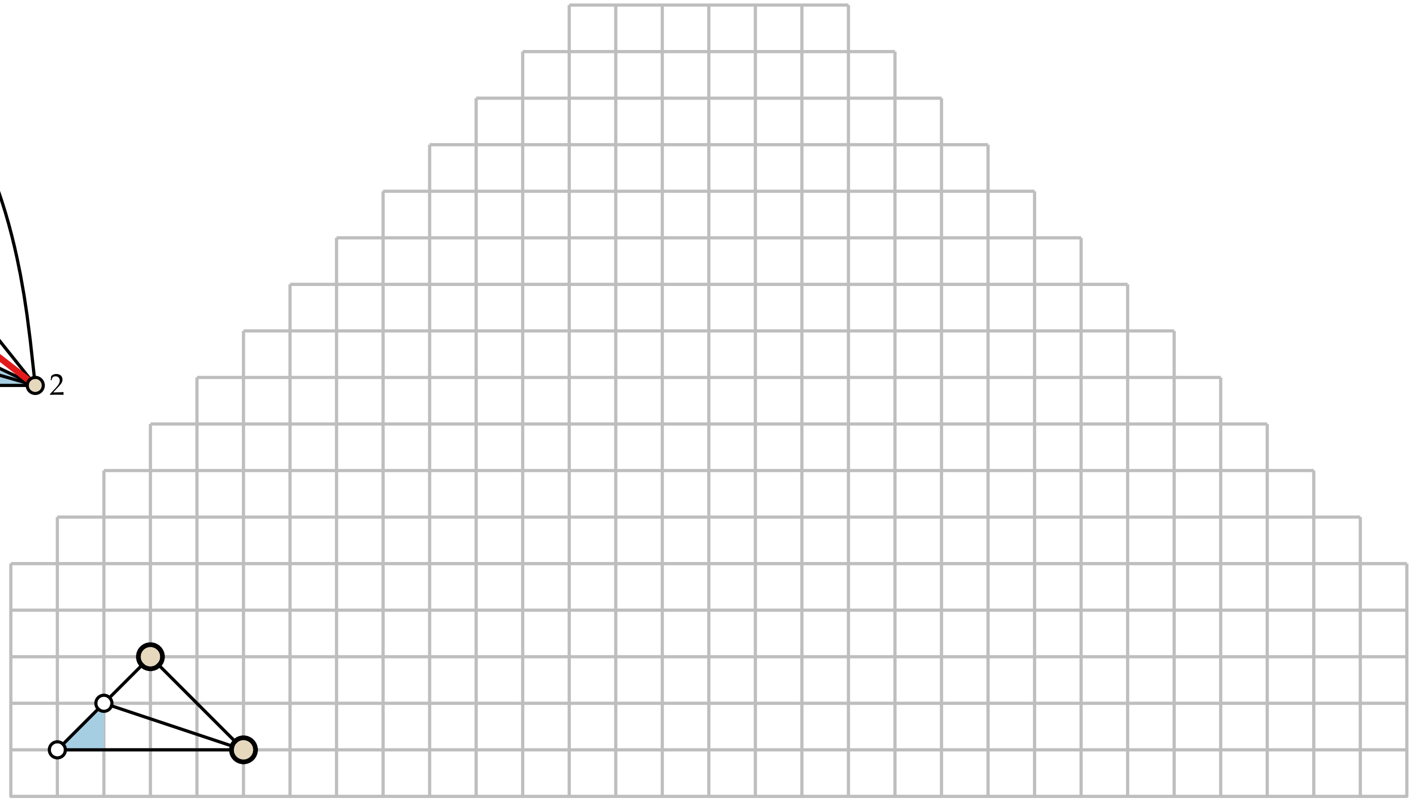




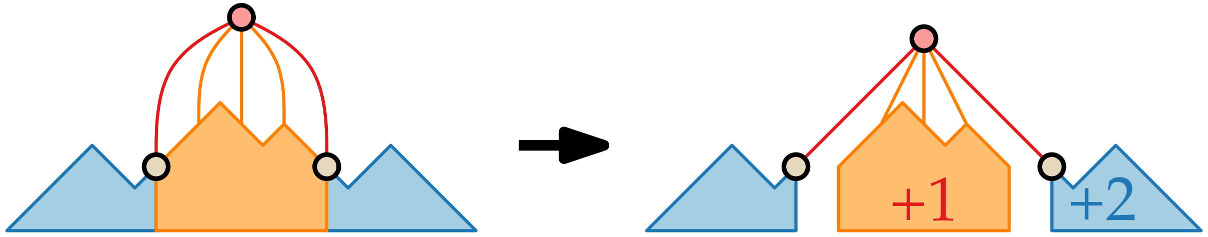
Shift Method – Example

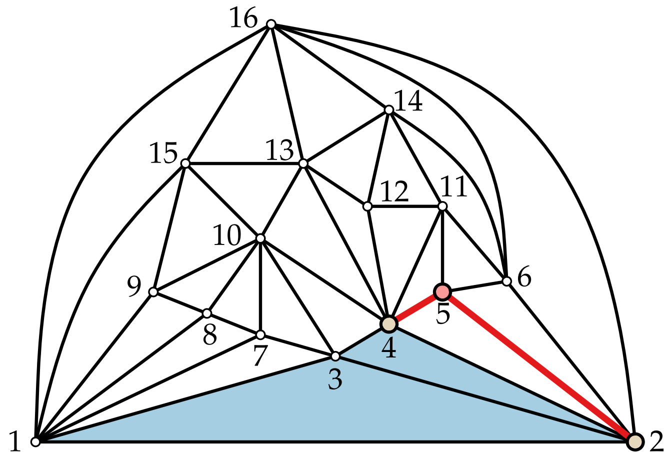


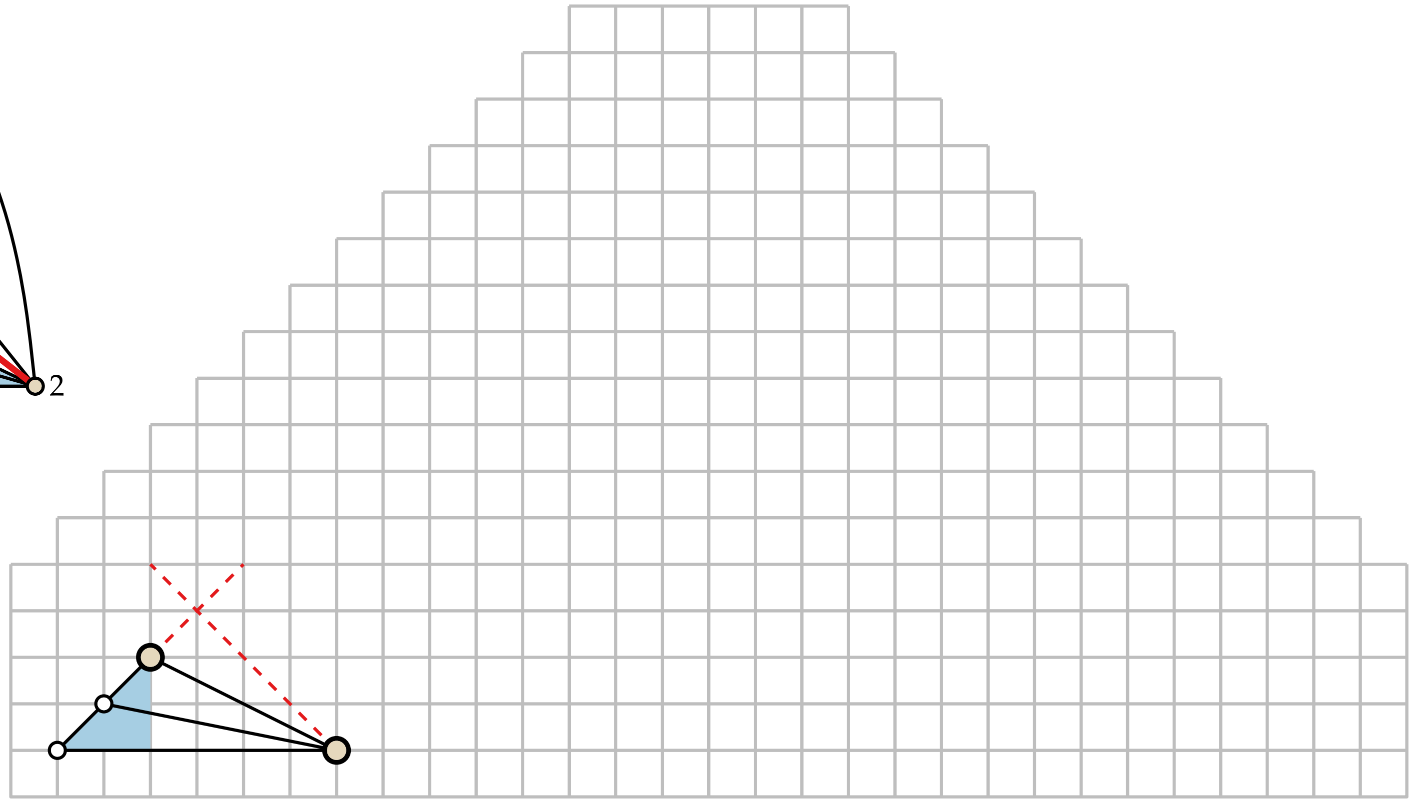




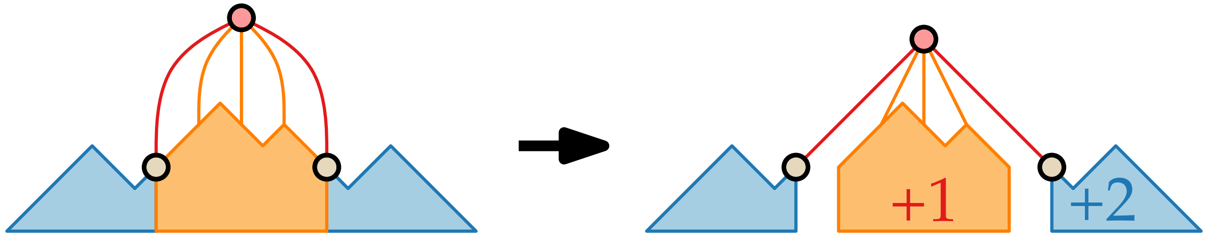
Shift Method – Example

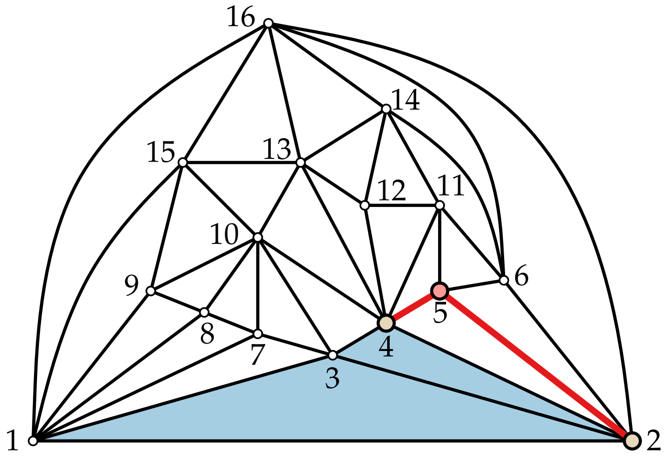


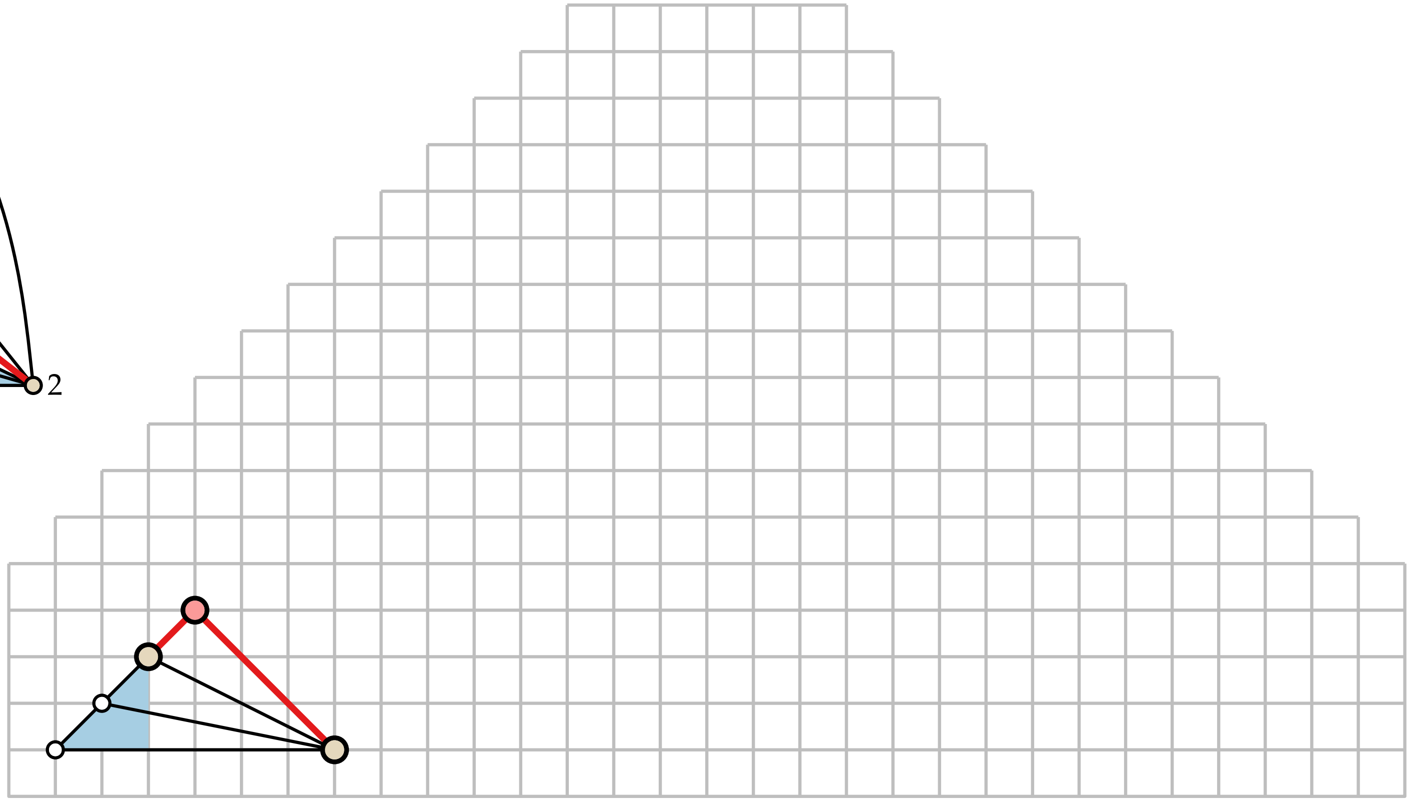




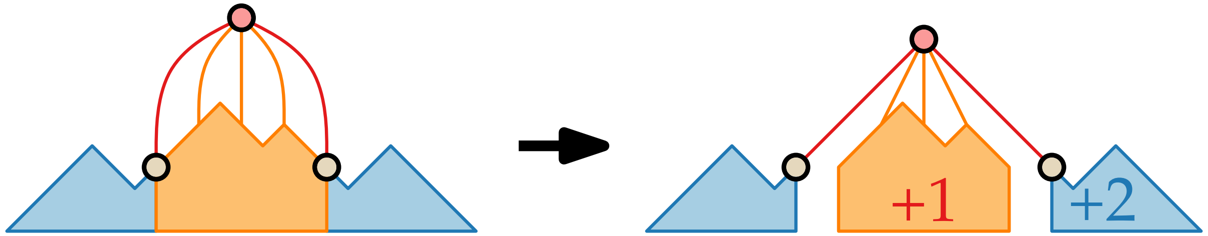
Shift Method – Example

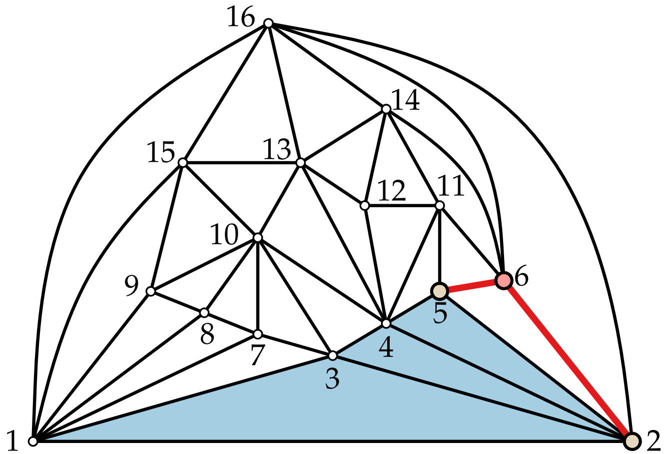


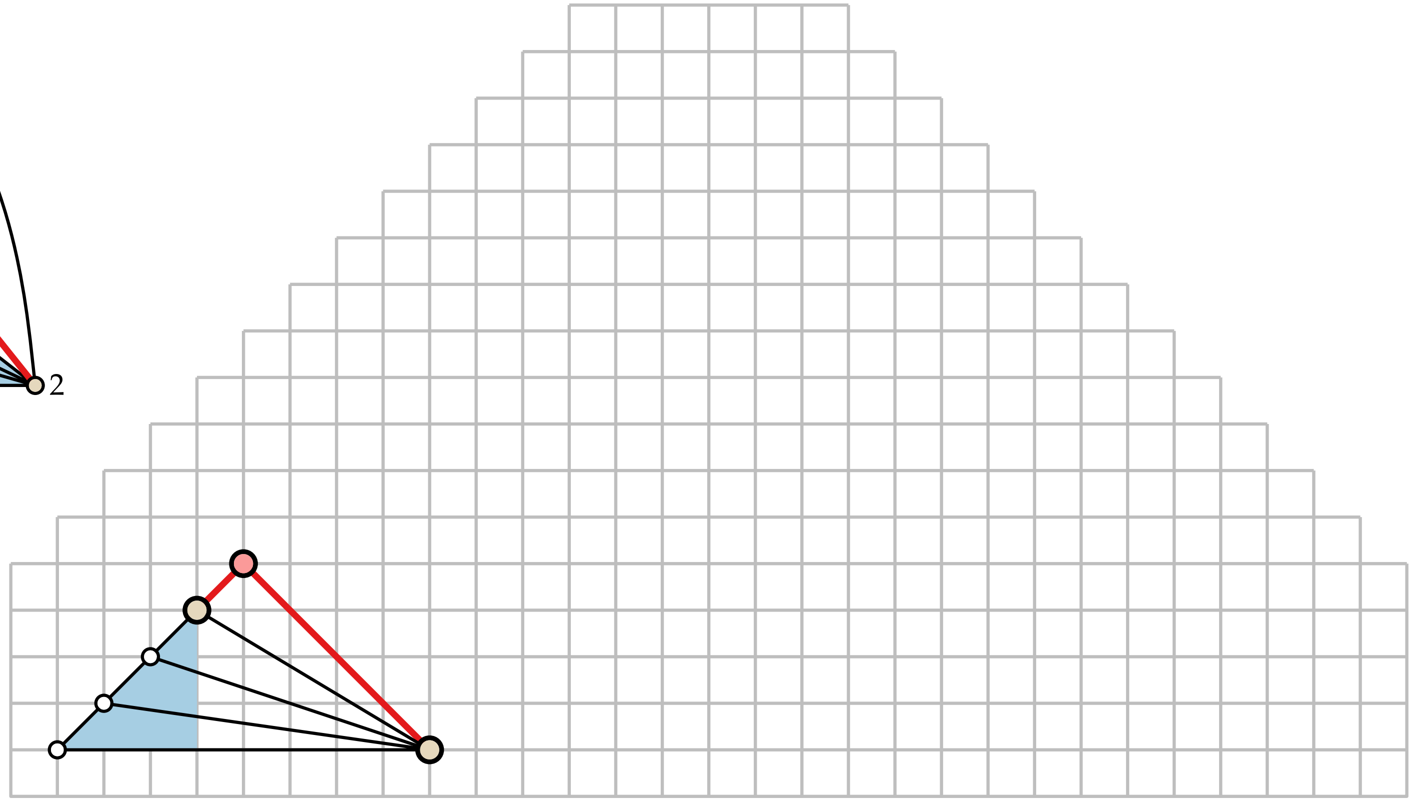




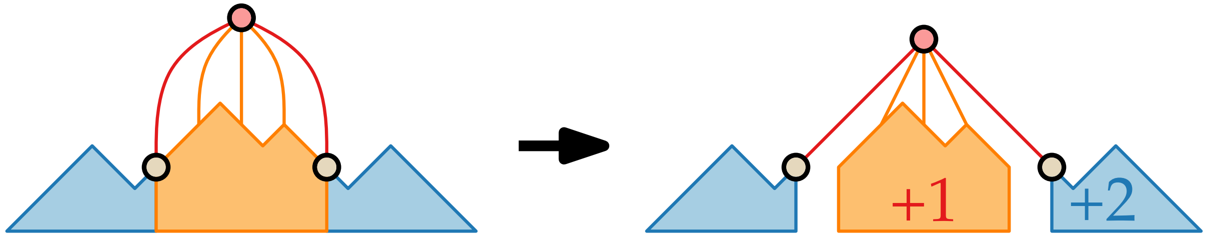
Shift Method – Example

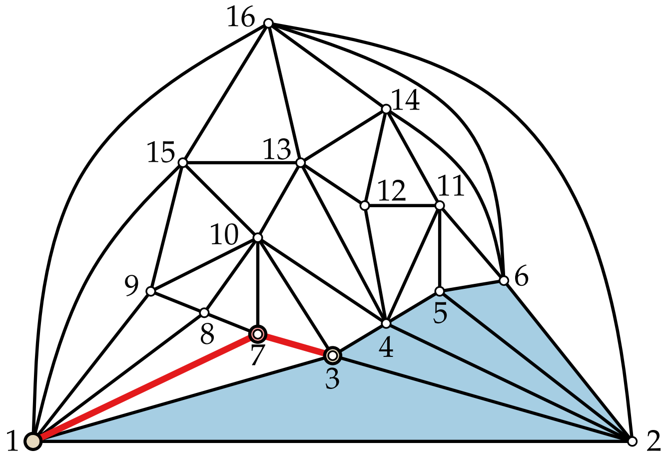


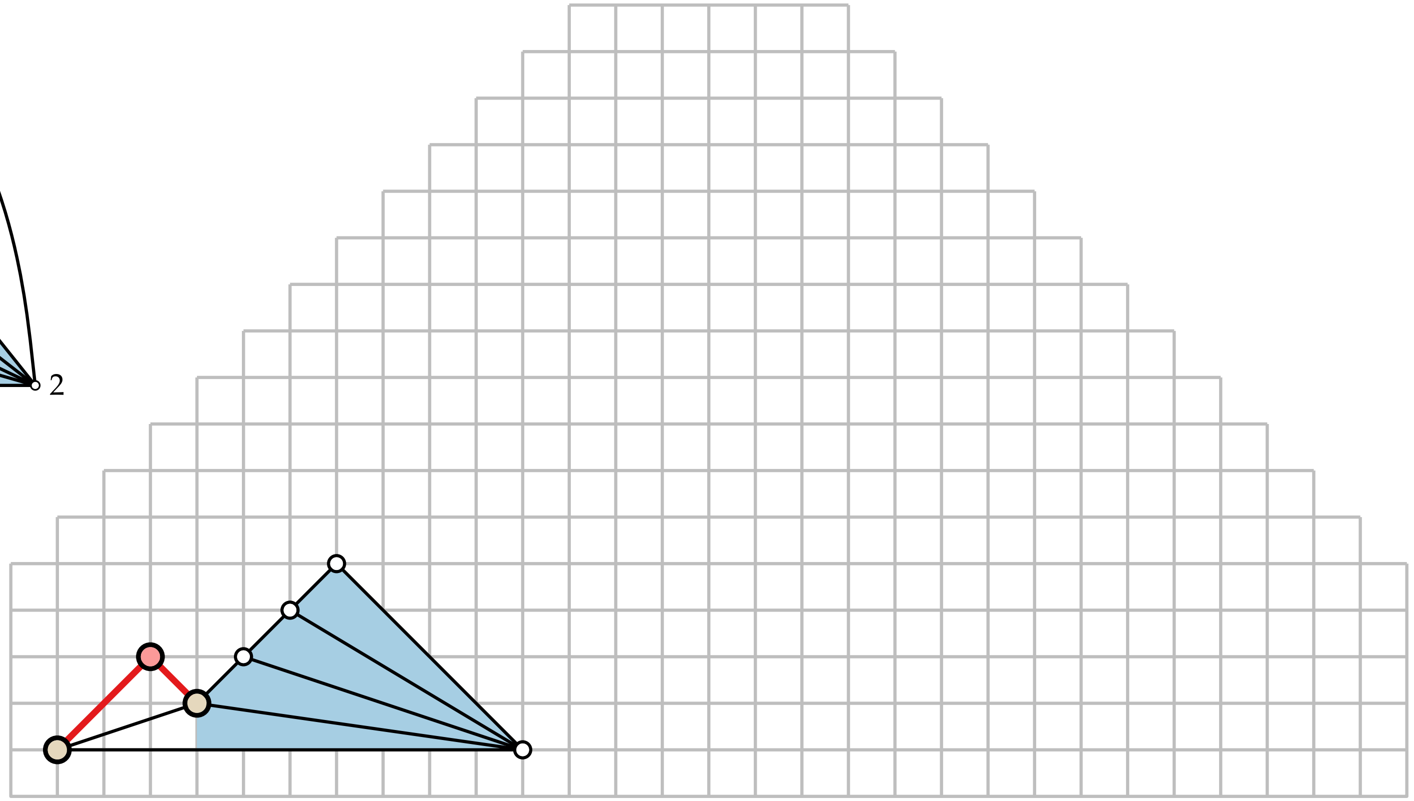




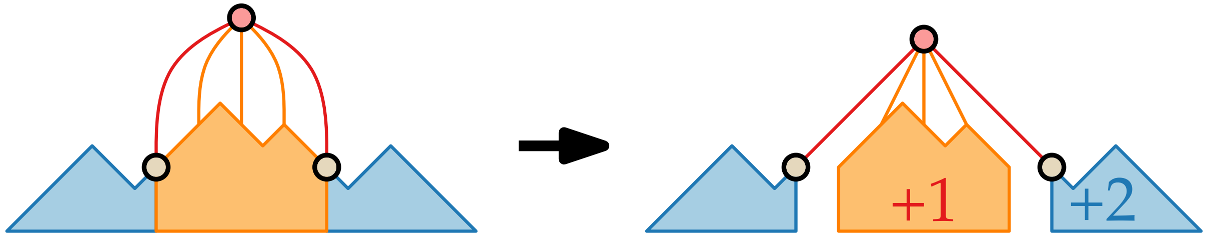
Shift Method – Example

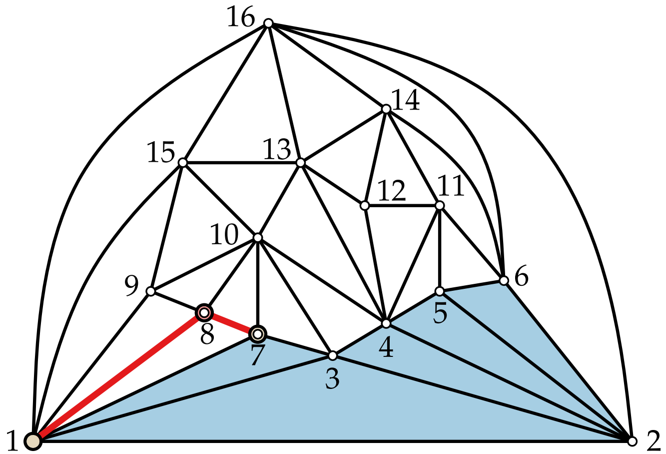


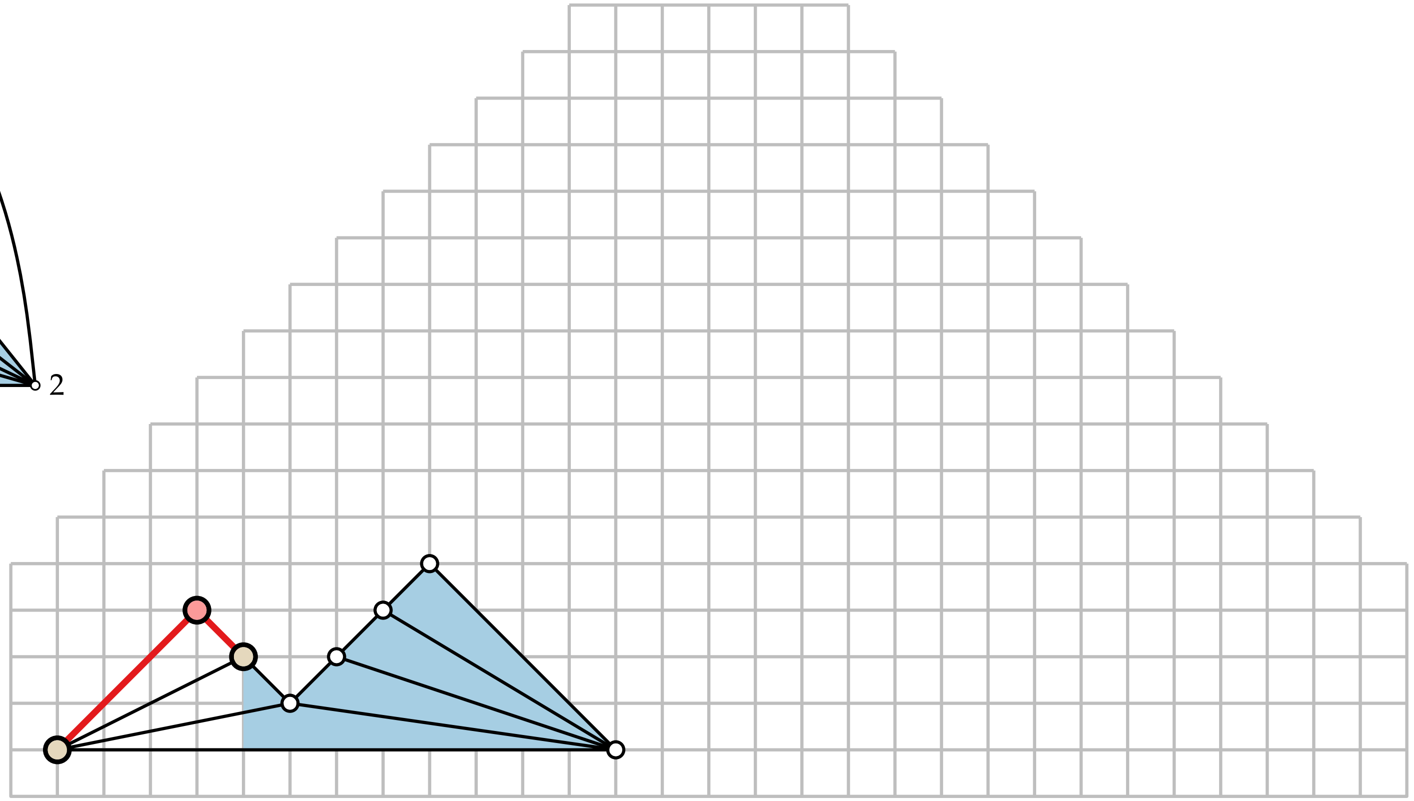




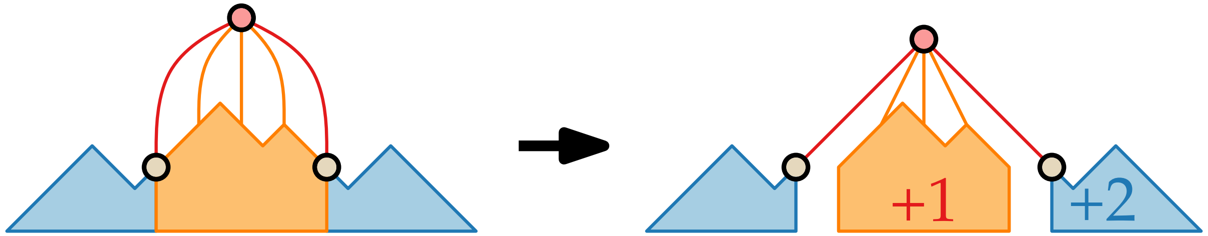
Shift Method – Example

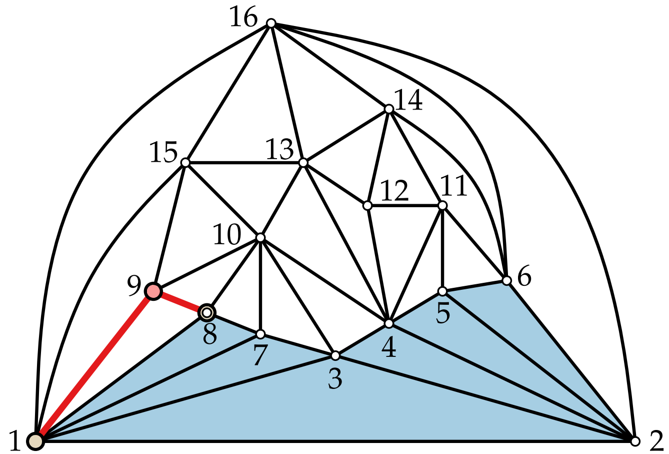


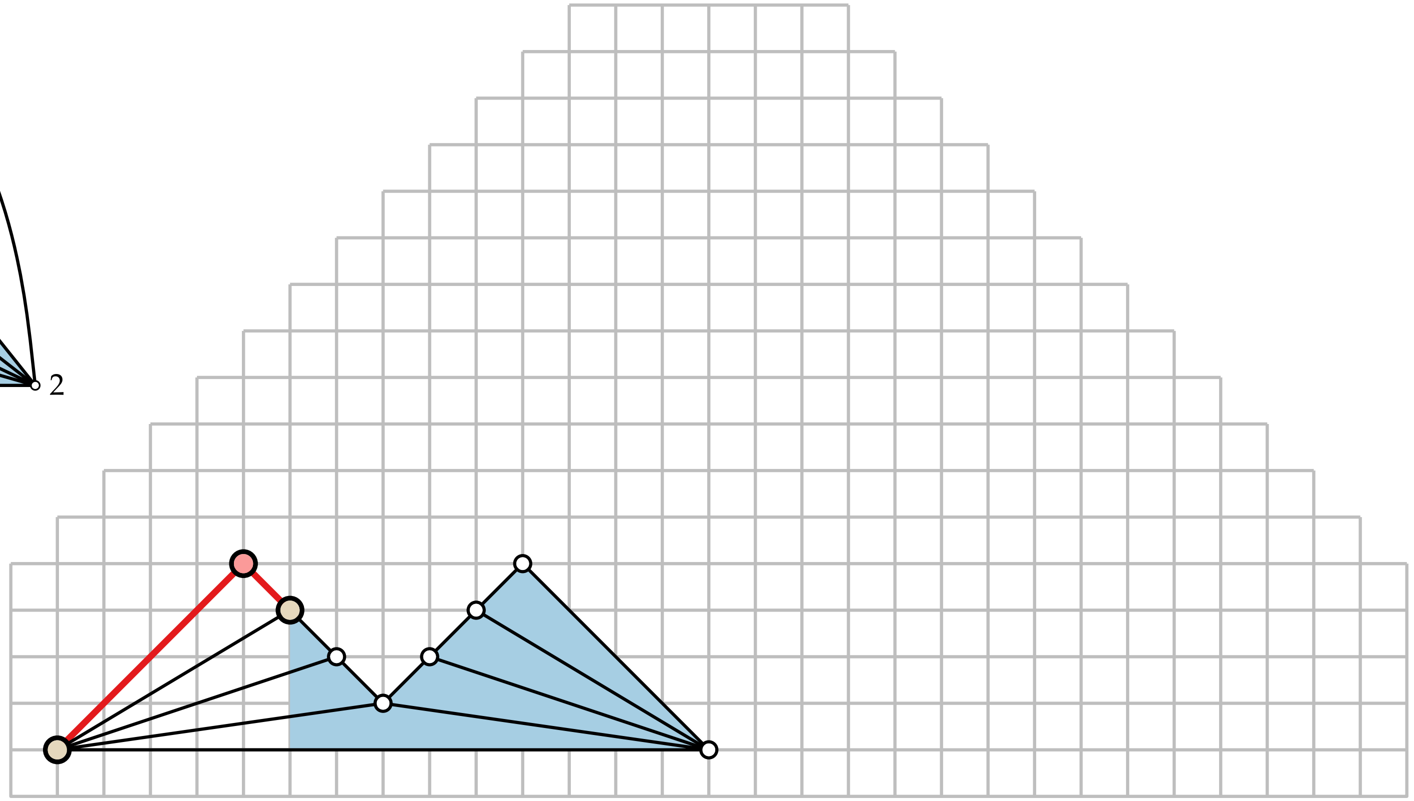




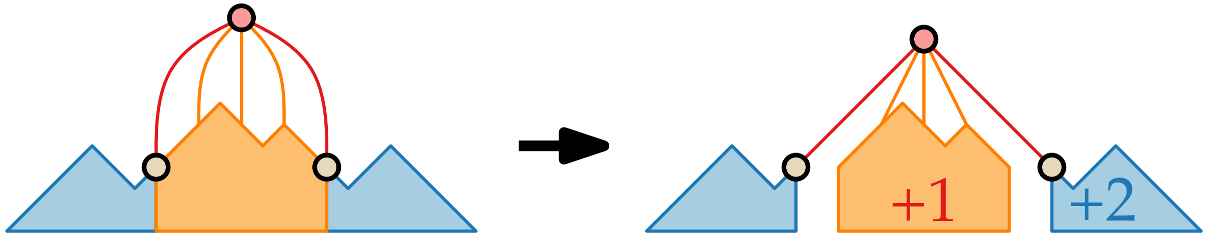
Shift Method – Example

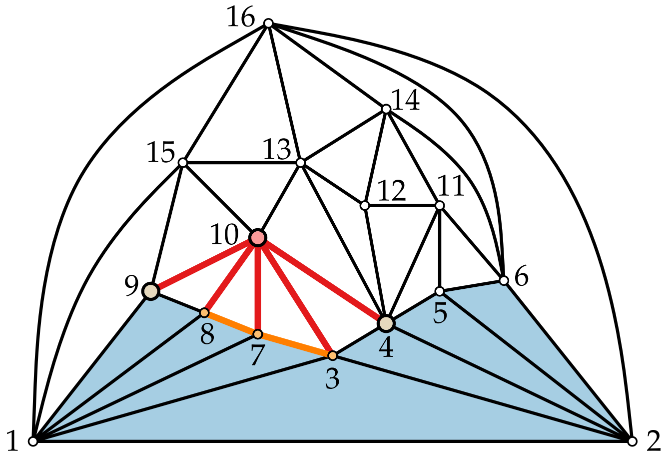


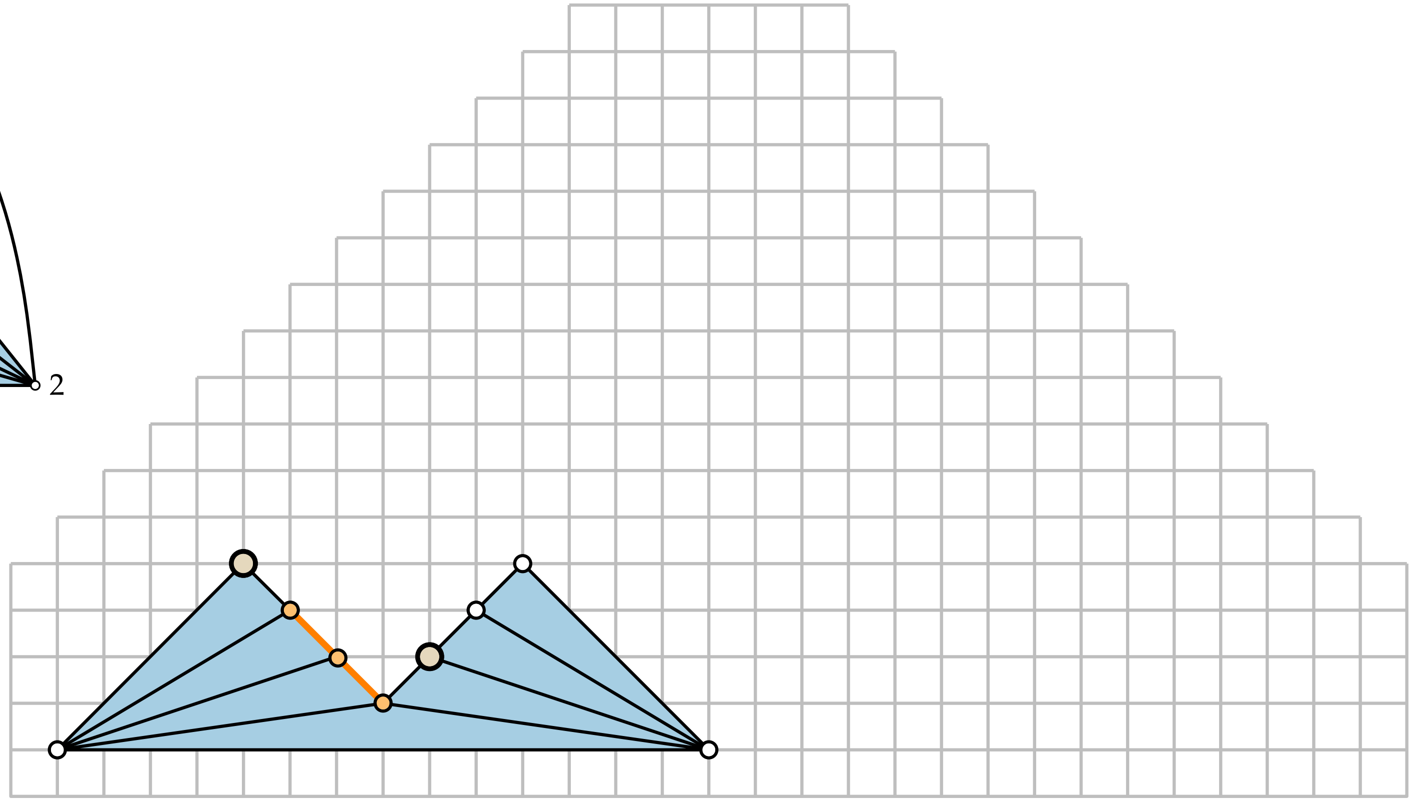




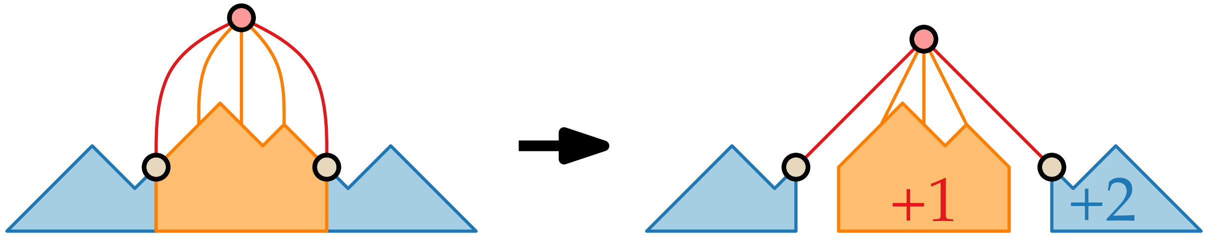
Shift Method – Example

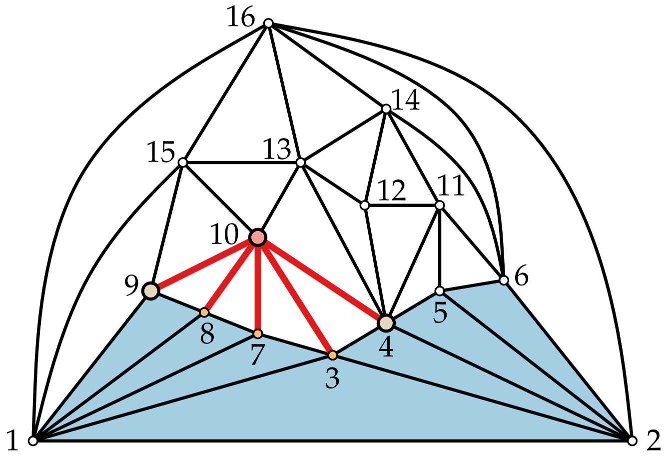


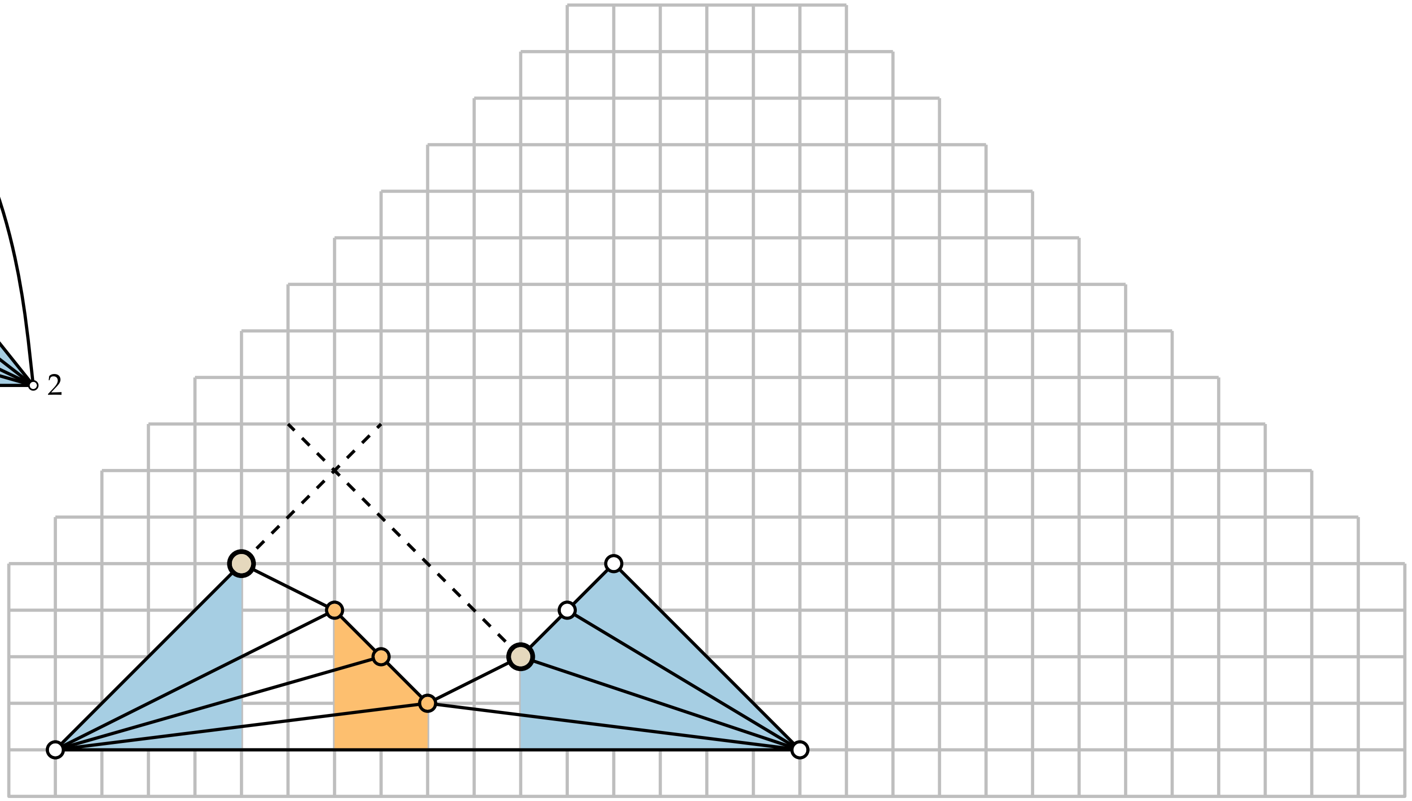




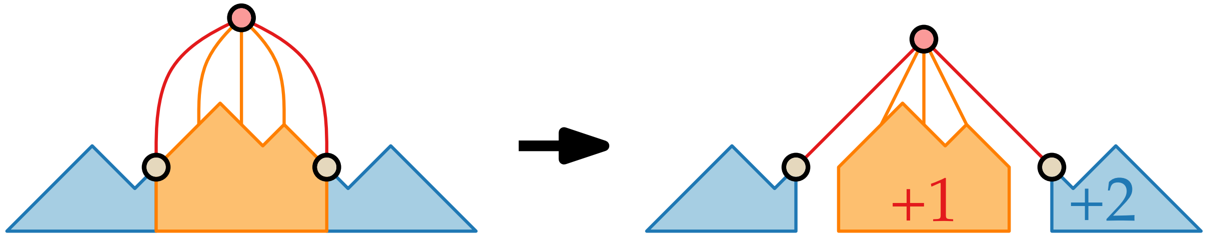
Shift Method – Example

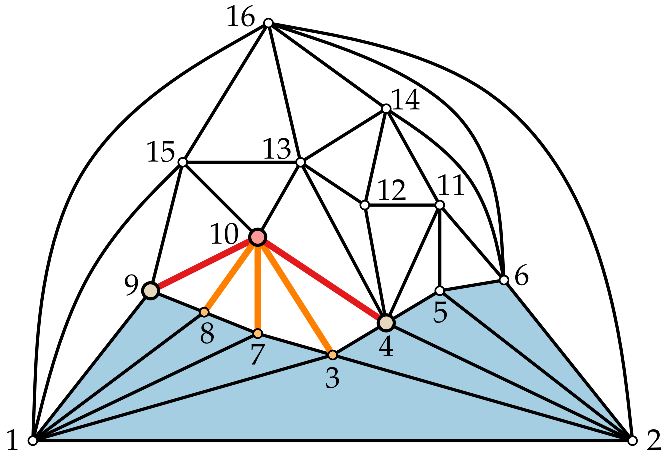


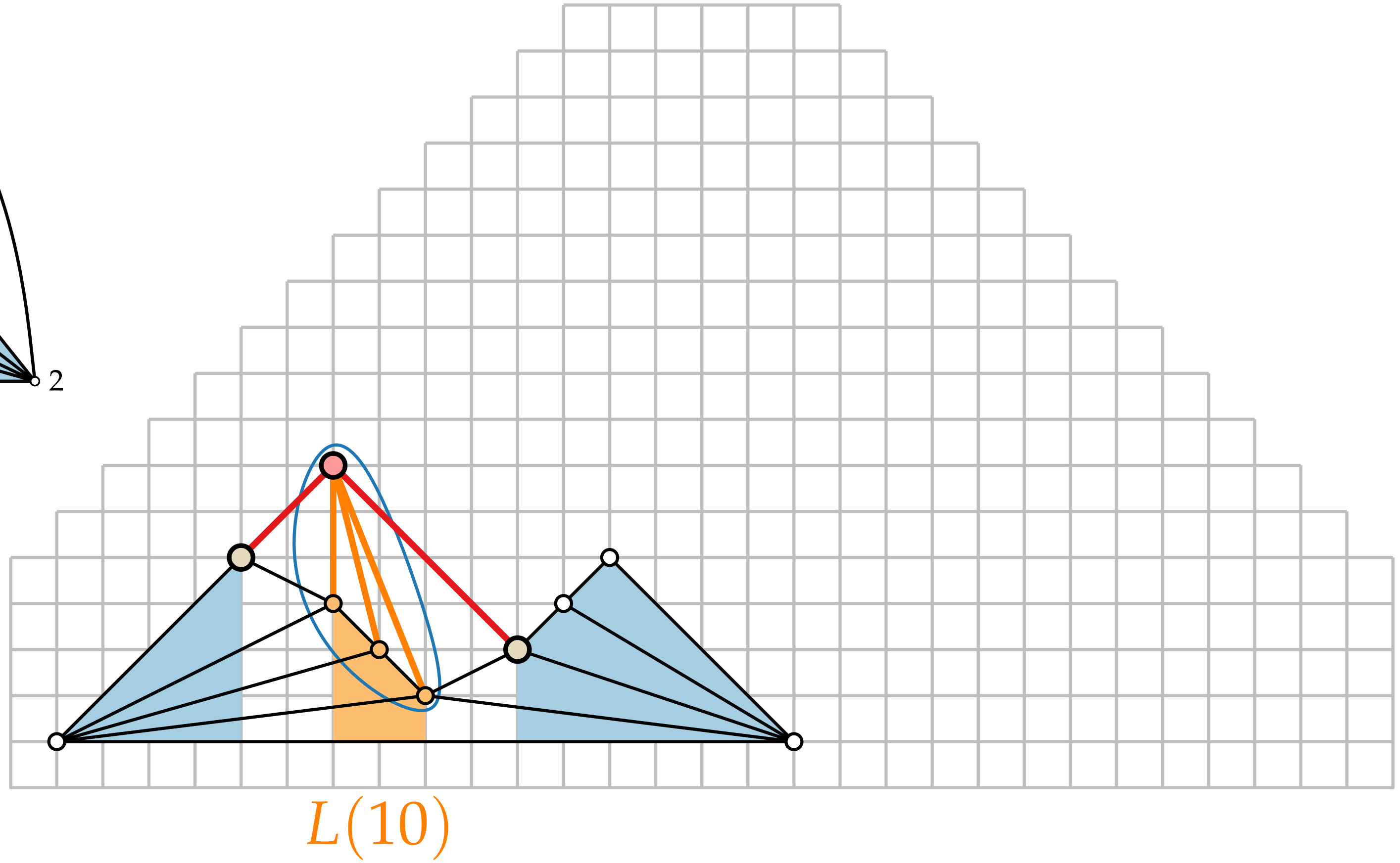




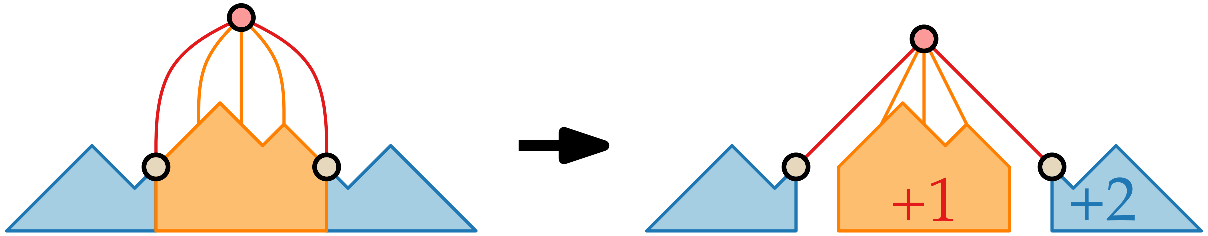
Shift Method – Example

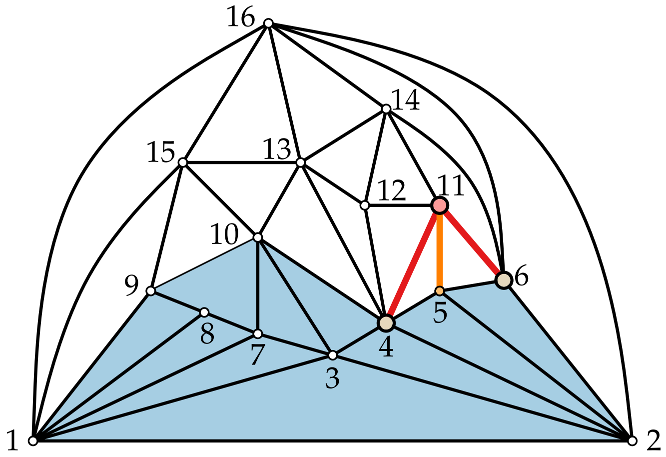


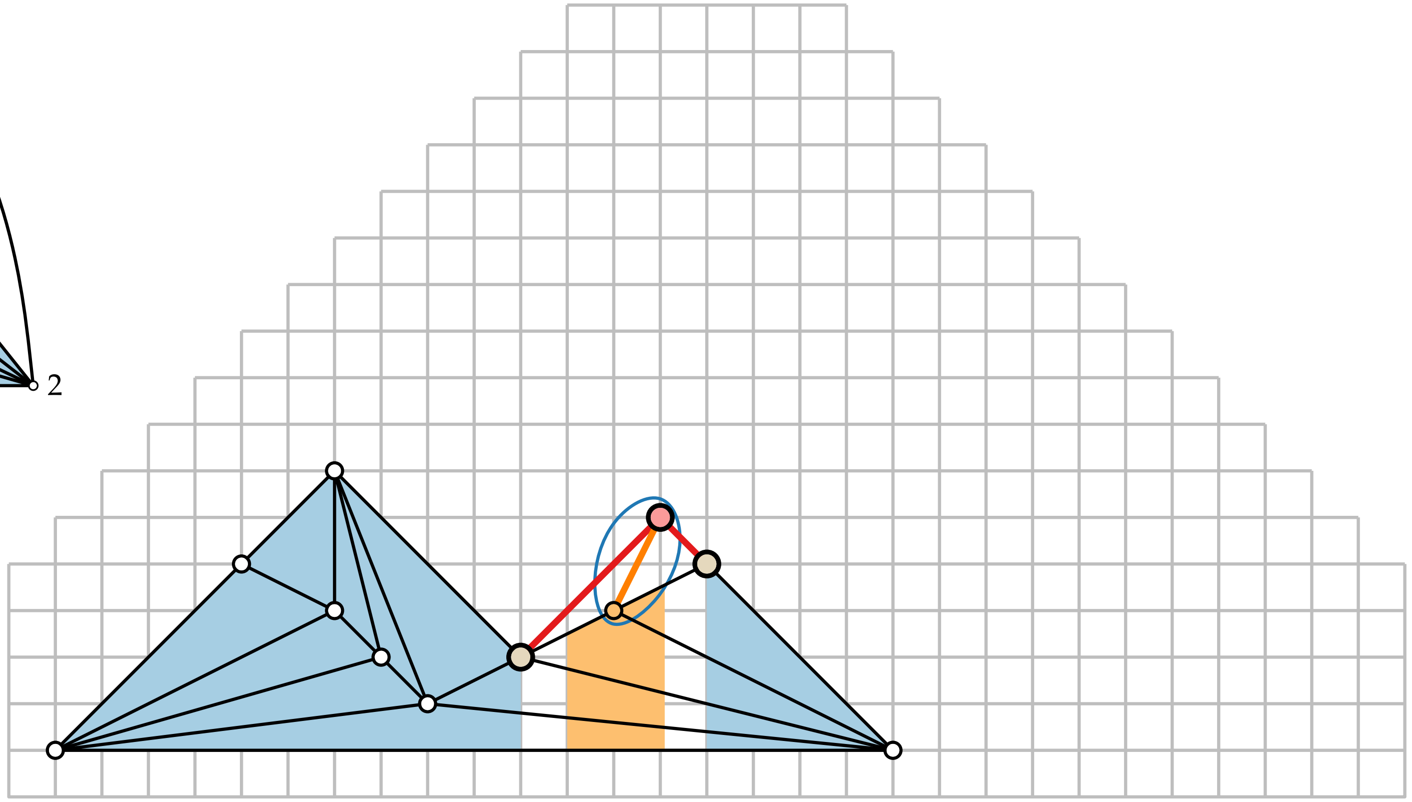




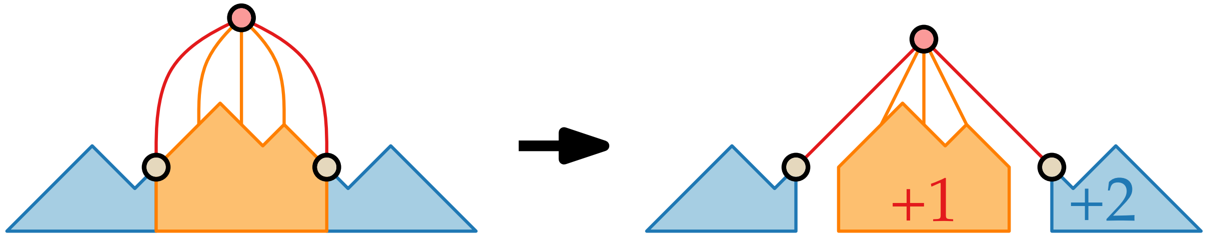
Shift Method – Example

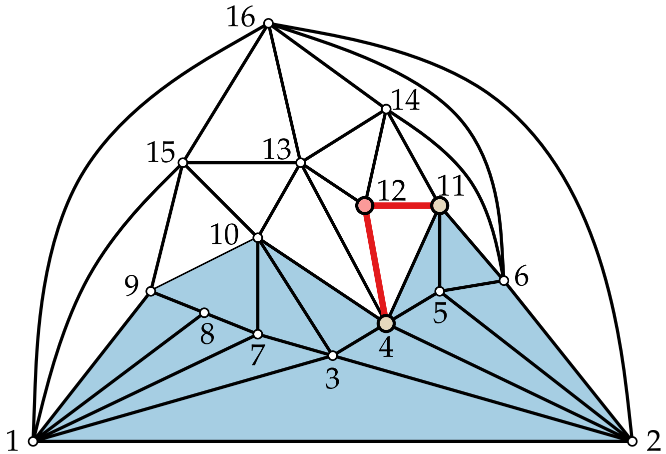


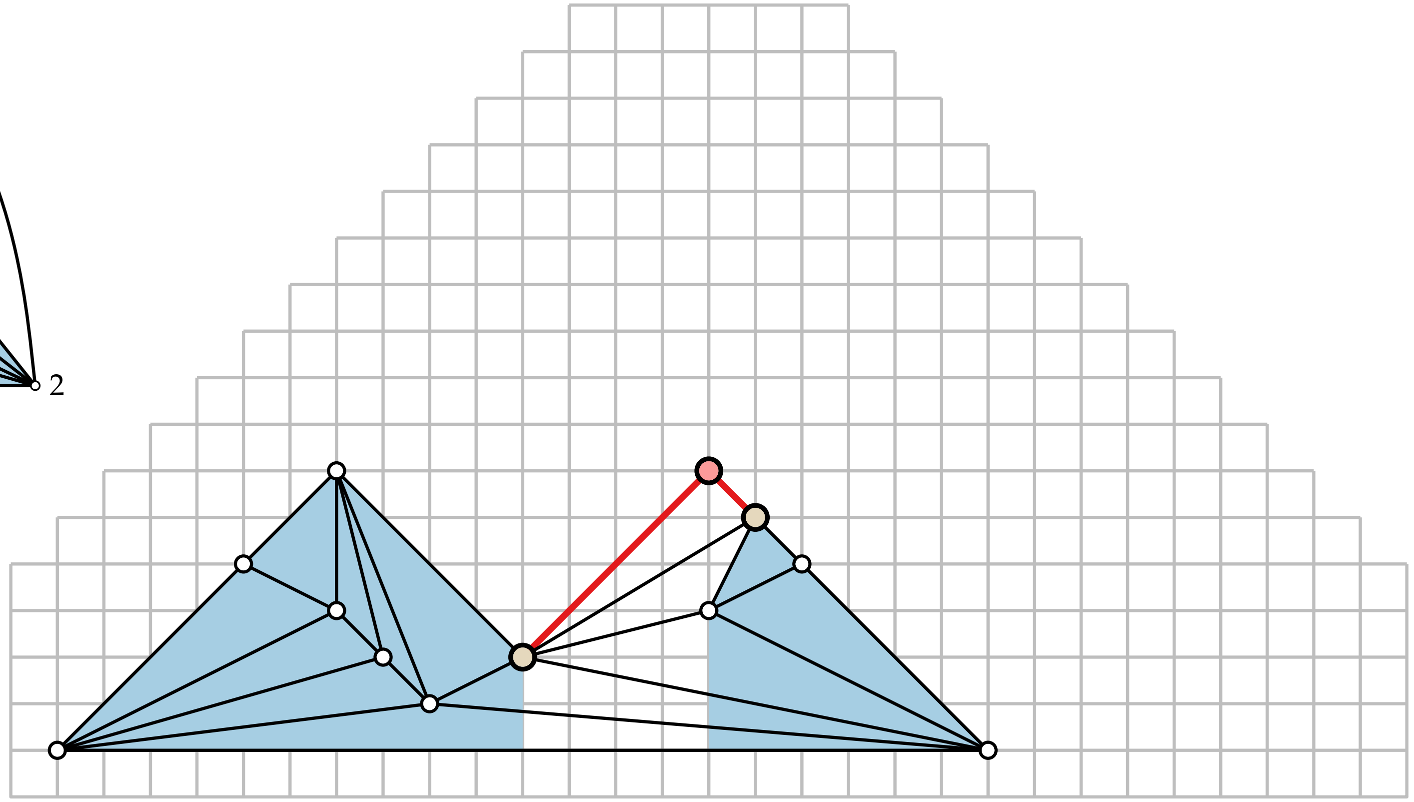




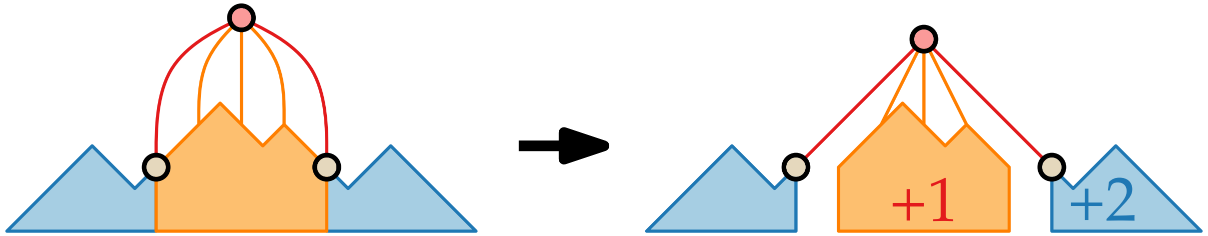
Shift Method – Example

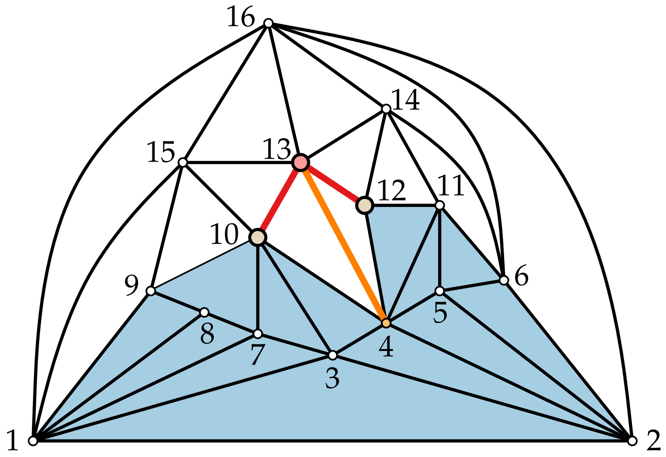


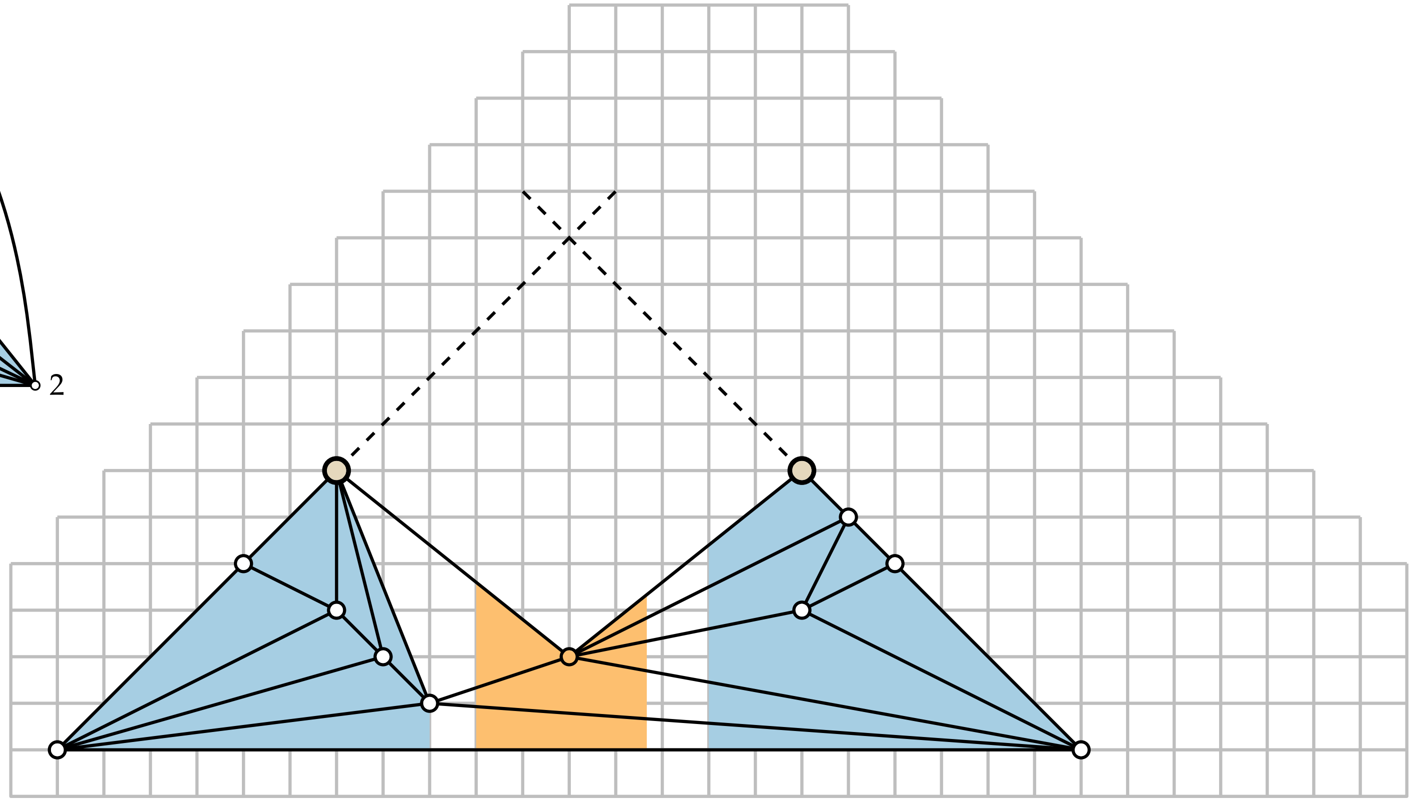




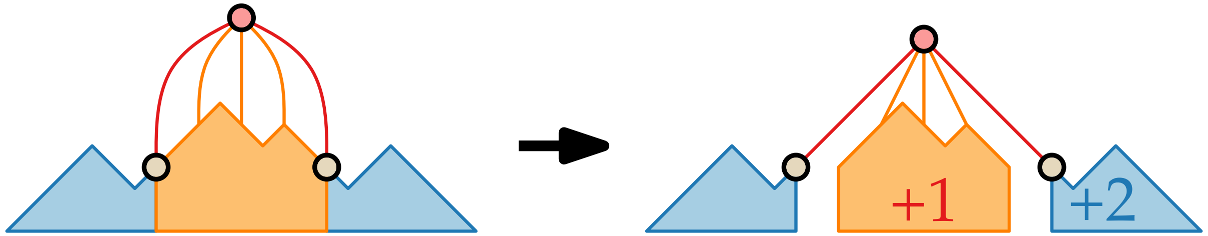
Shift Method – Example

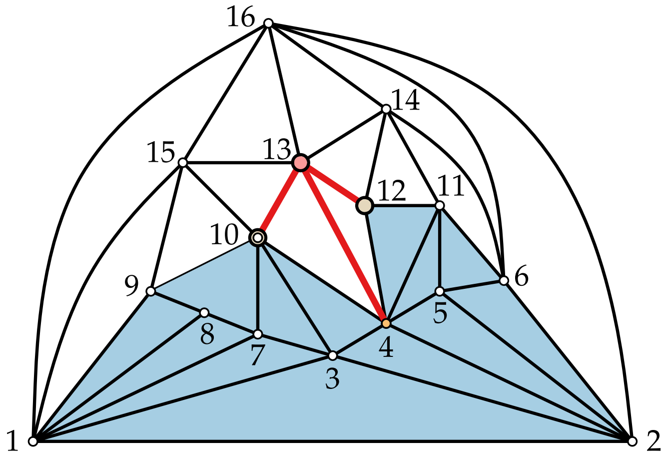


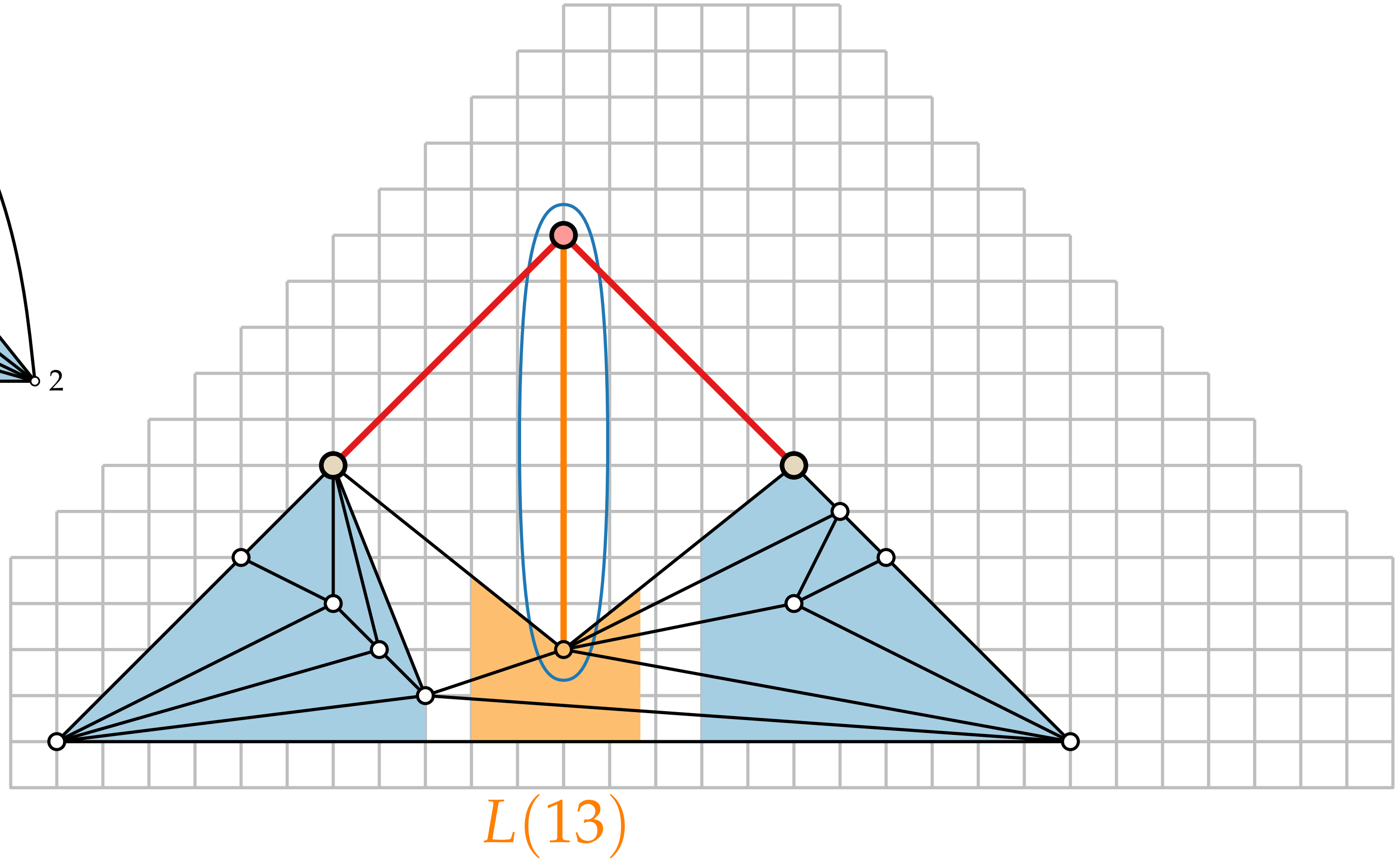




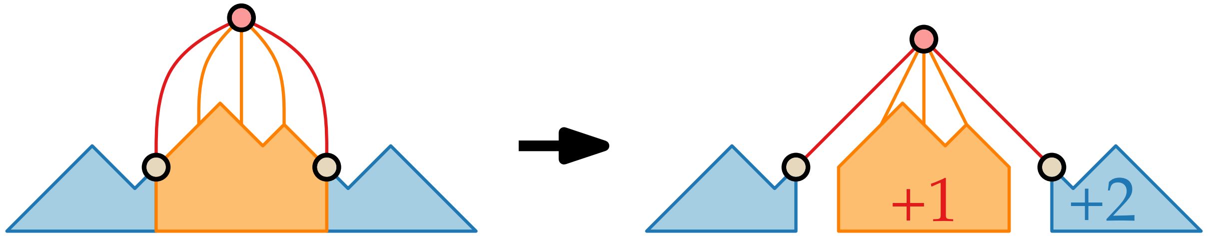
Shift Method – Example

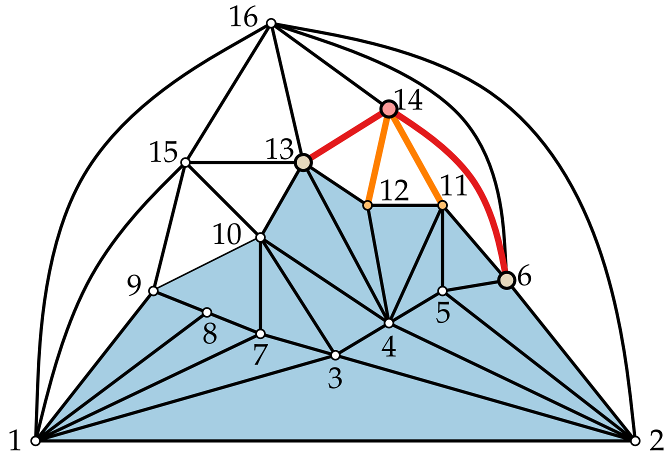


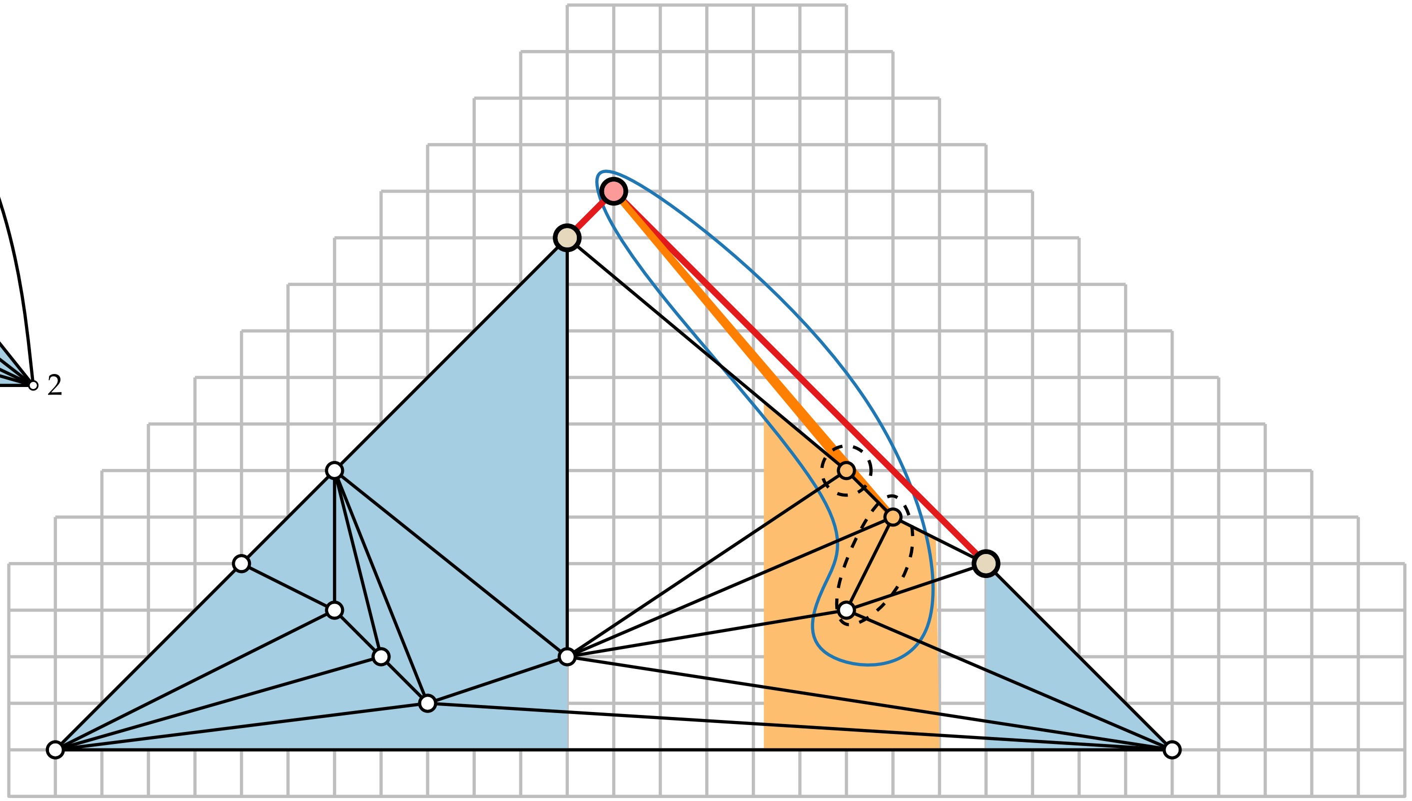




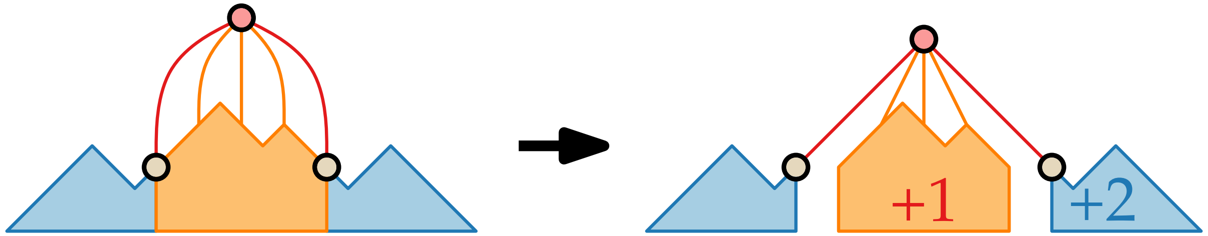
Shift Method – Example

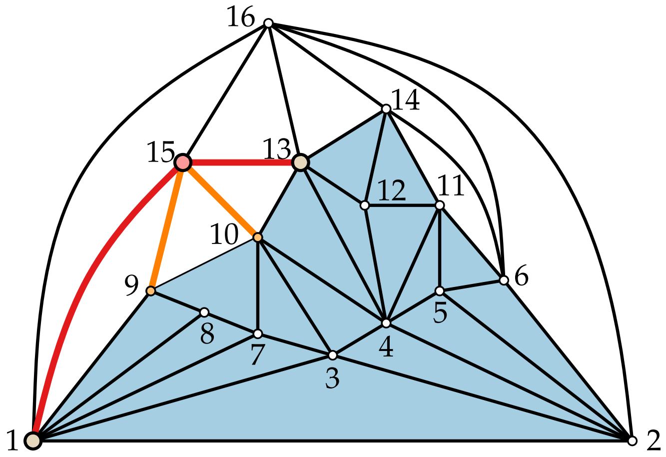


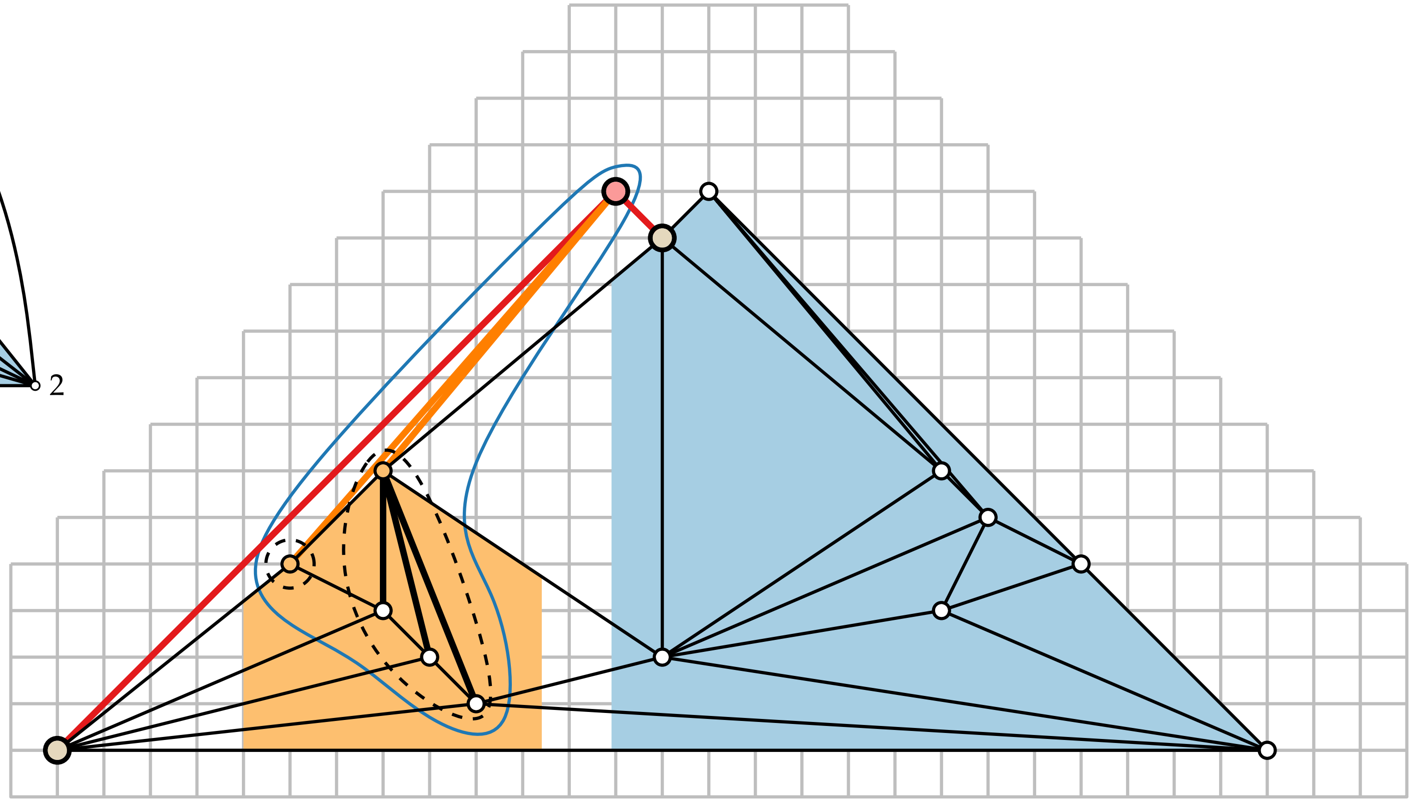




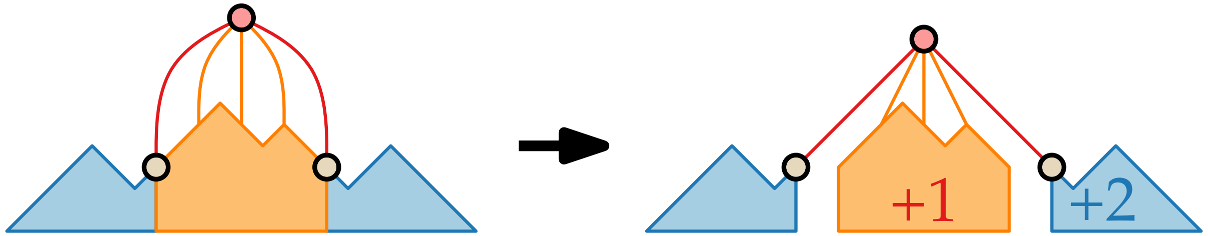
Shift Method – Example

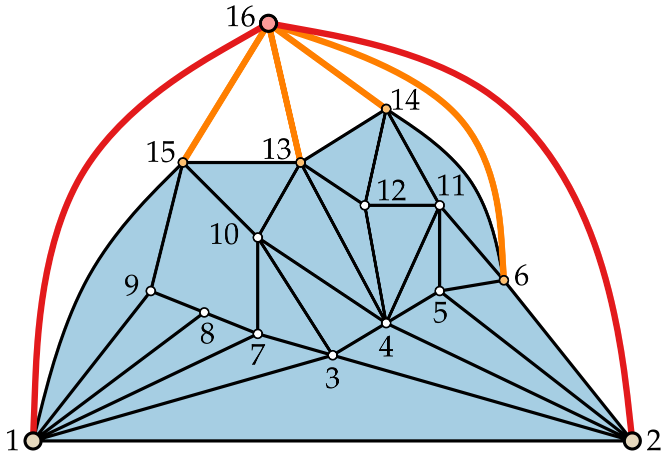


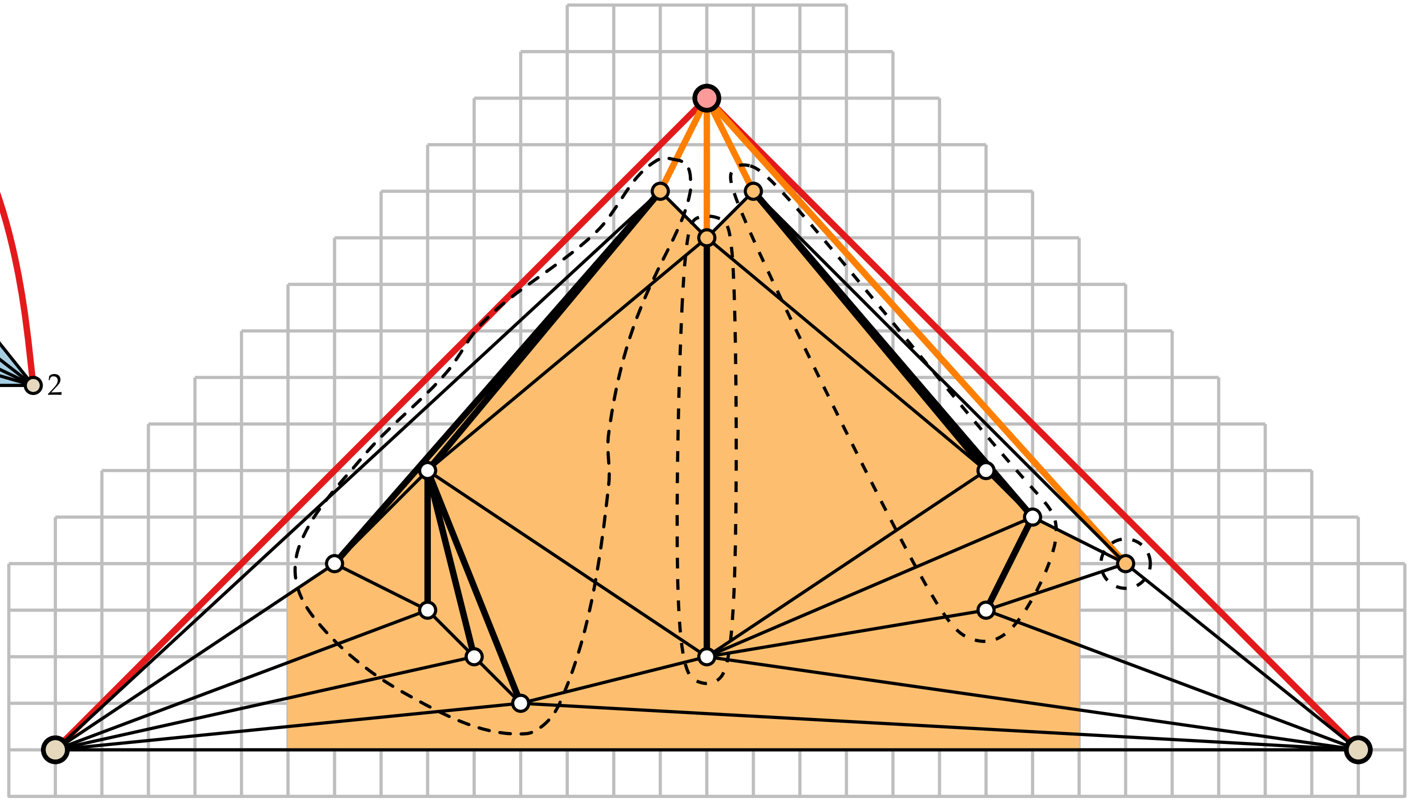




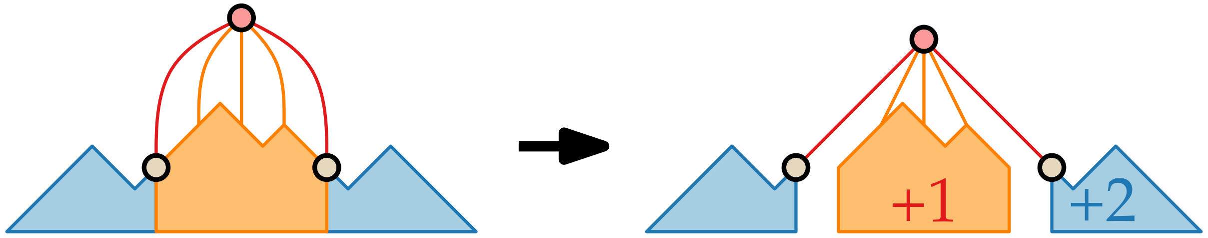
Shift Method – Example

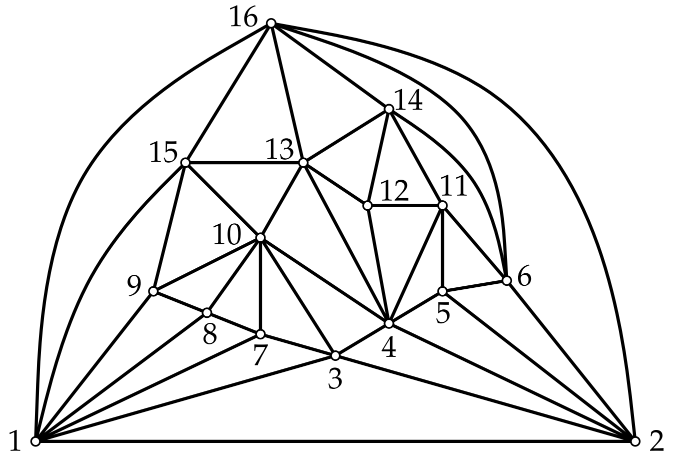


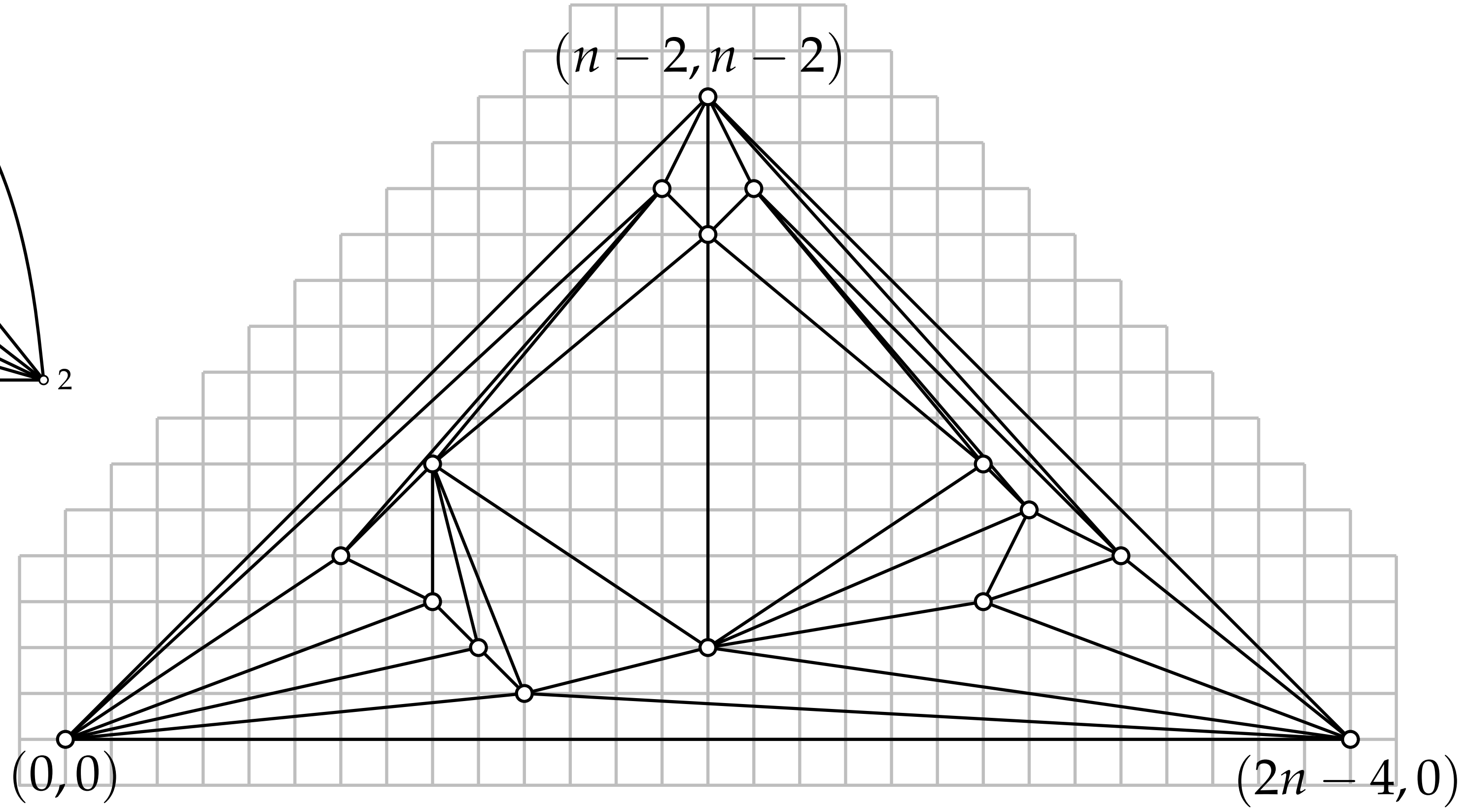




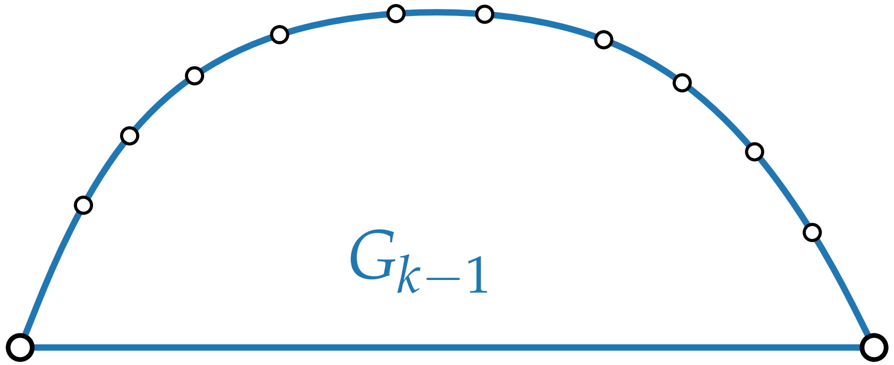
Shift Method – Example



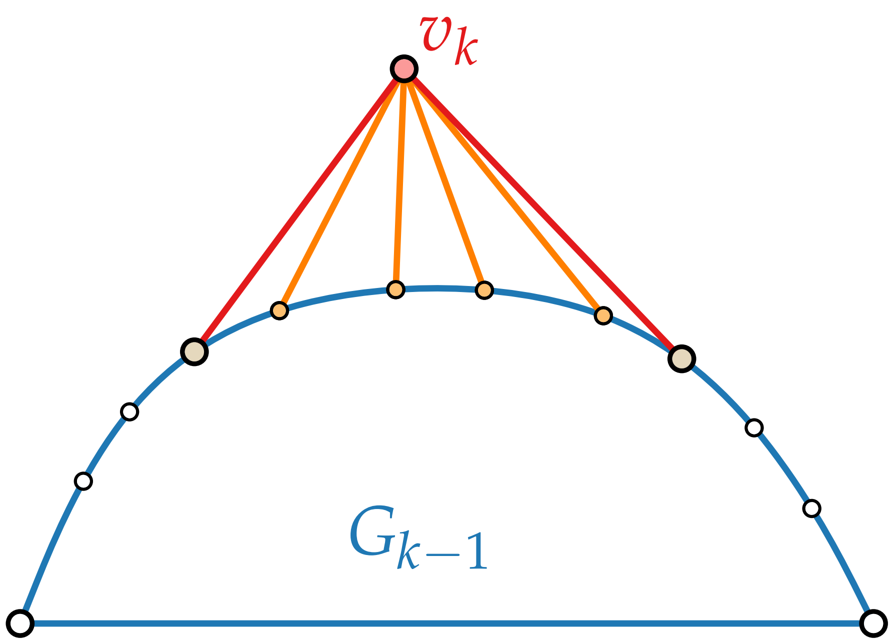




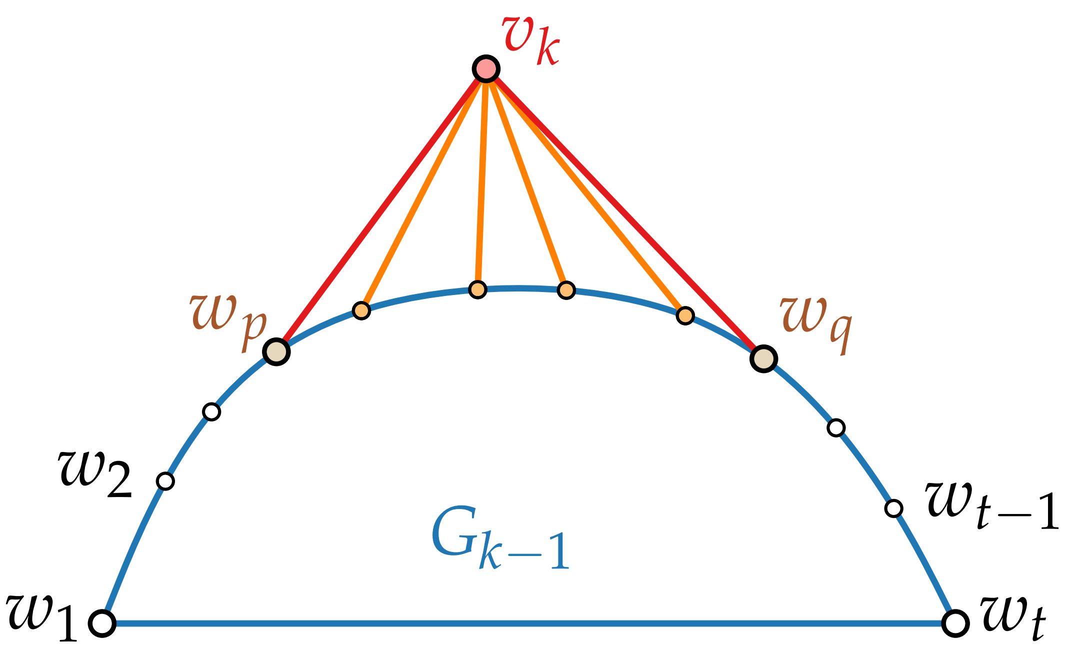
Shift Method – Planarity



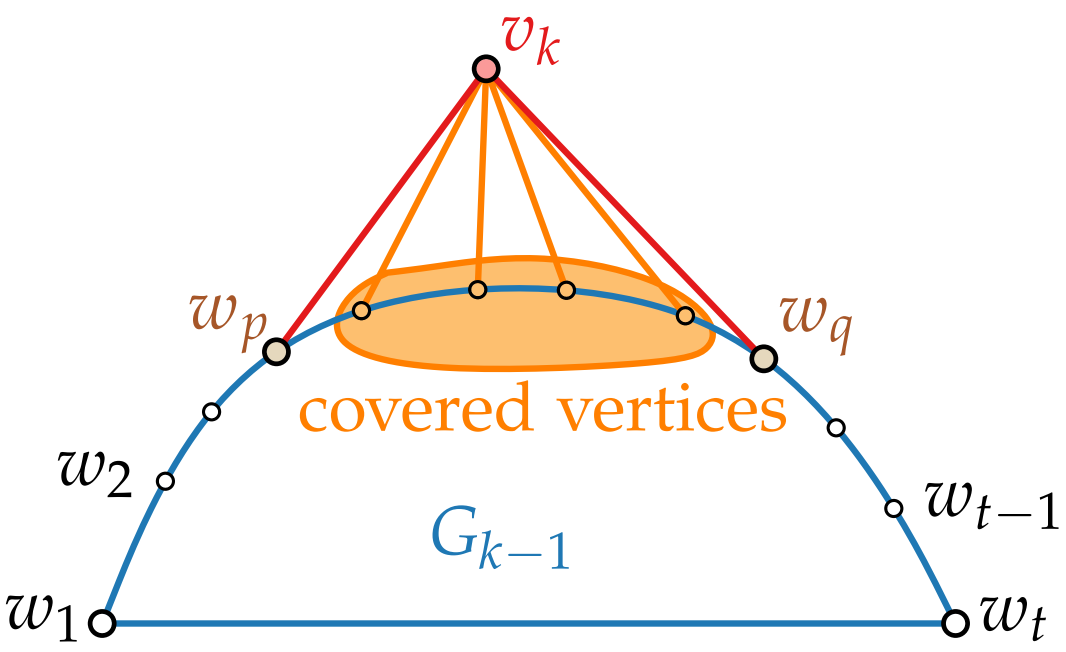
Shift Method – Planarity



Shift Method – Planarity



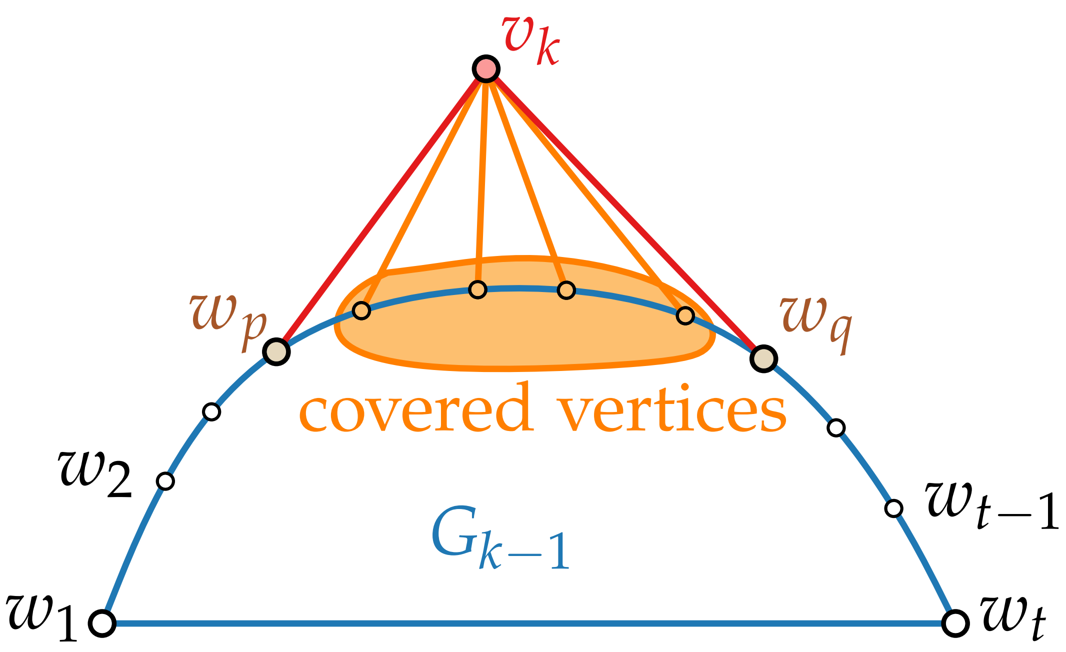
Shift Method – Planarity



Shift Method – Planarity

Observations.

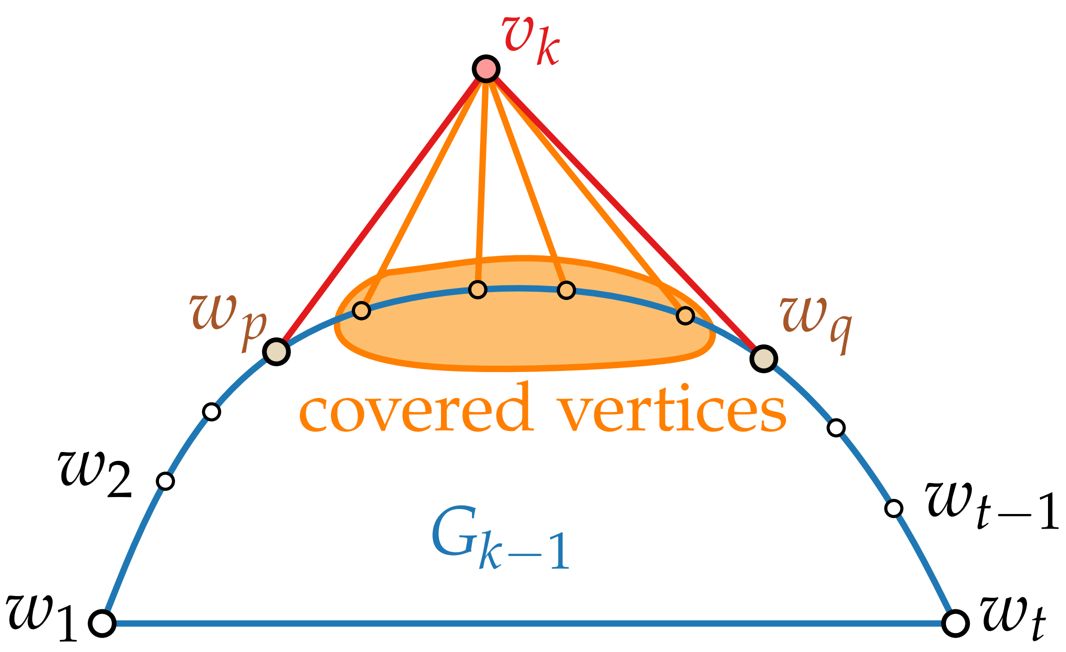
- Each internal vertex is covered exactly once.



Shift Method – Planarity

Observations.

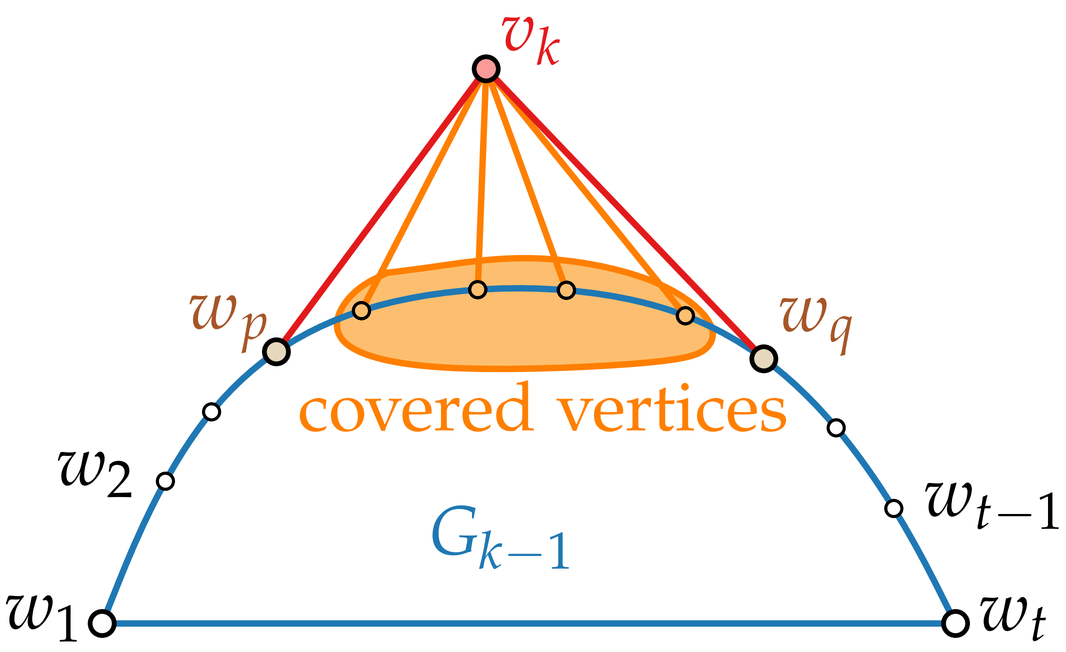
- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G



Shift Method – Planarity

Observations.

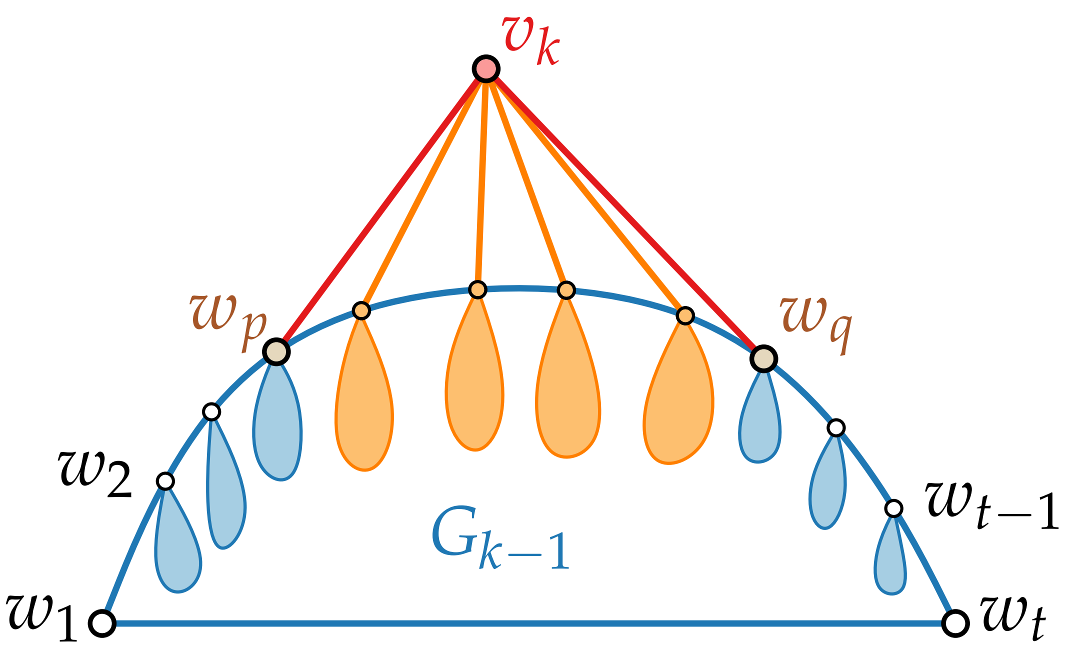
- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.



Shift Method – Planarity

Observations.

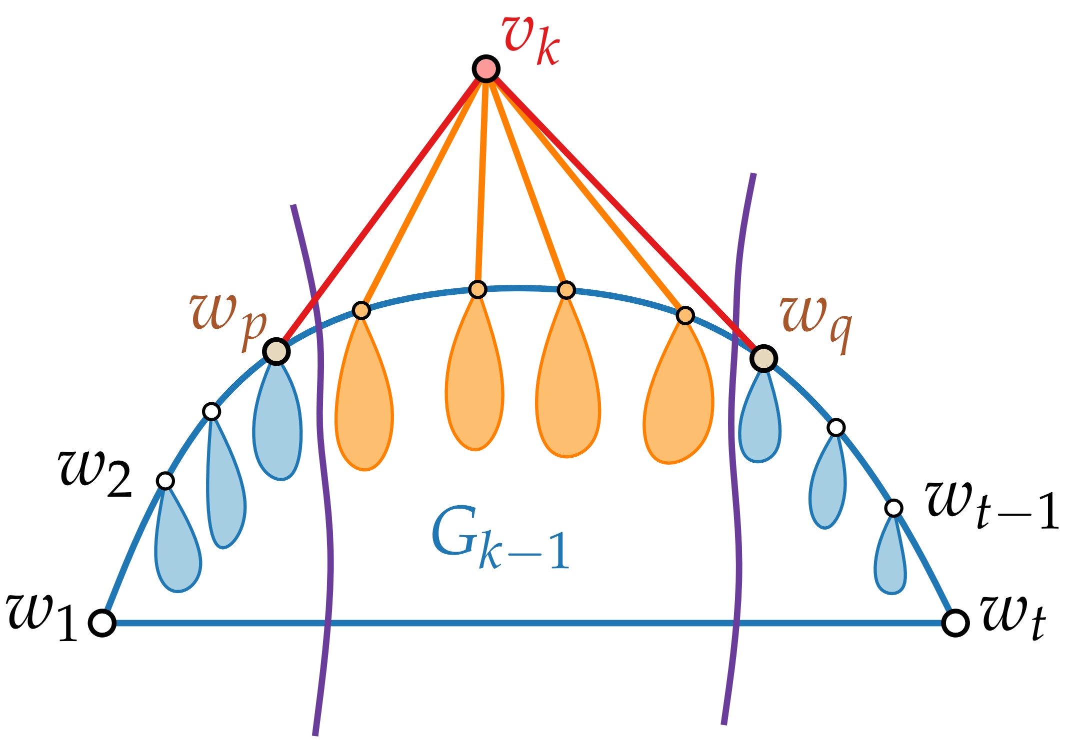
- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.



Shift Method – Planarity

Observations.

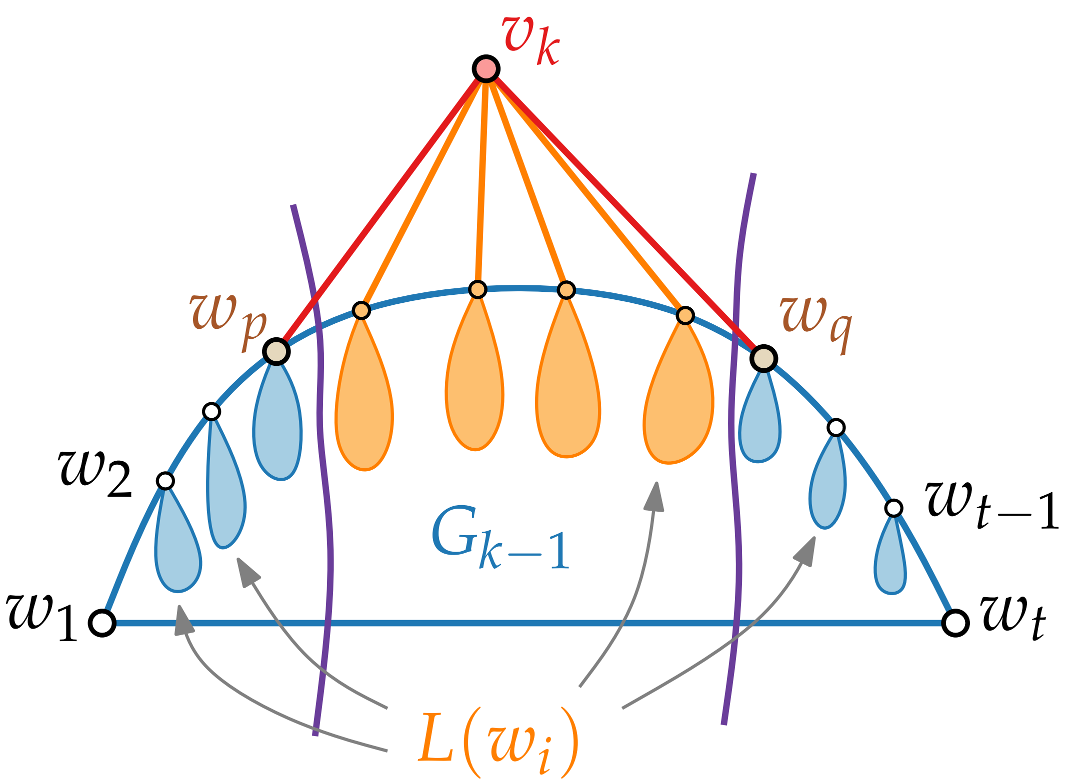
- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.



Shift Method – Planarity

Observations.

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

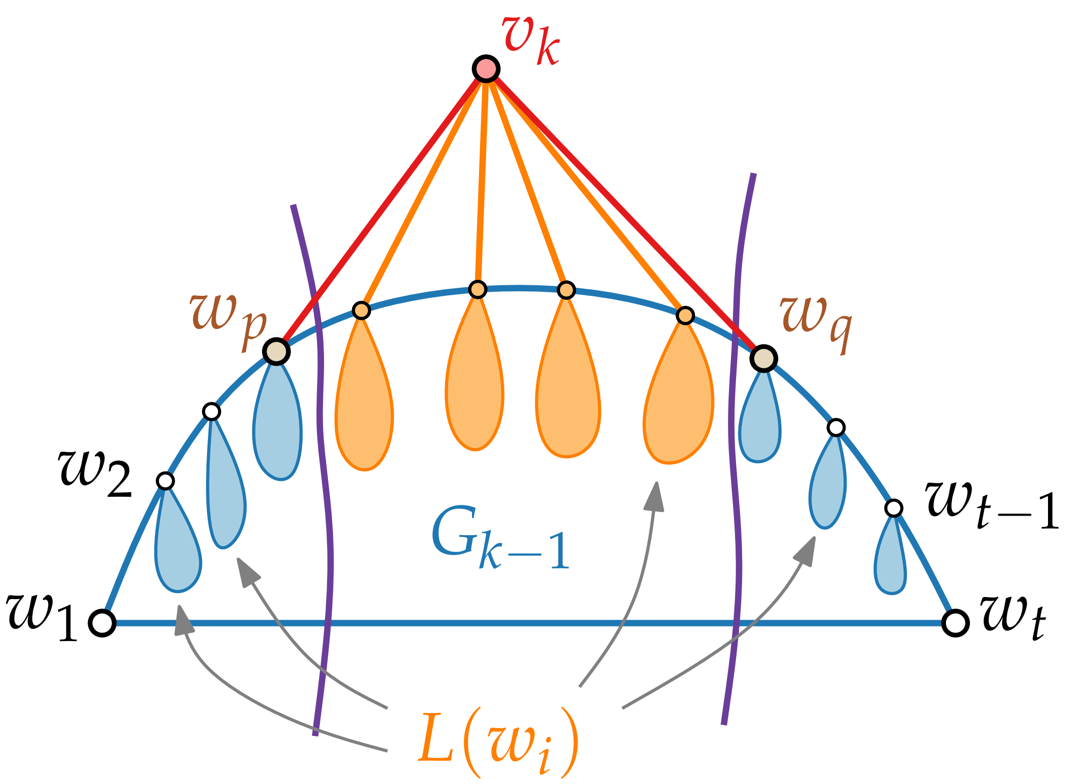


Shift Method – Planarity

Observations.

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

Lemma.
Let $0 < \delta_1 \le \delta_2 \le \cdots \le \delta_t \in \mathbb{N}$, such that $\delta_q - \delta_p \ge 2$ and even.

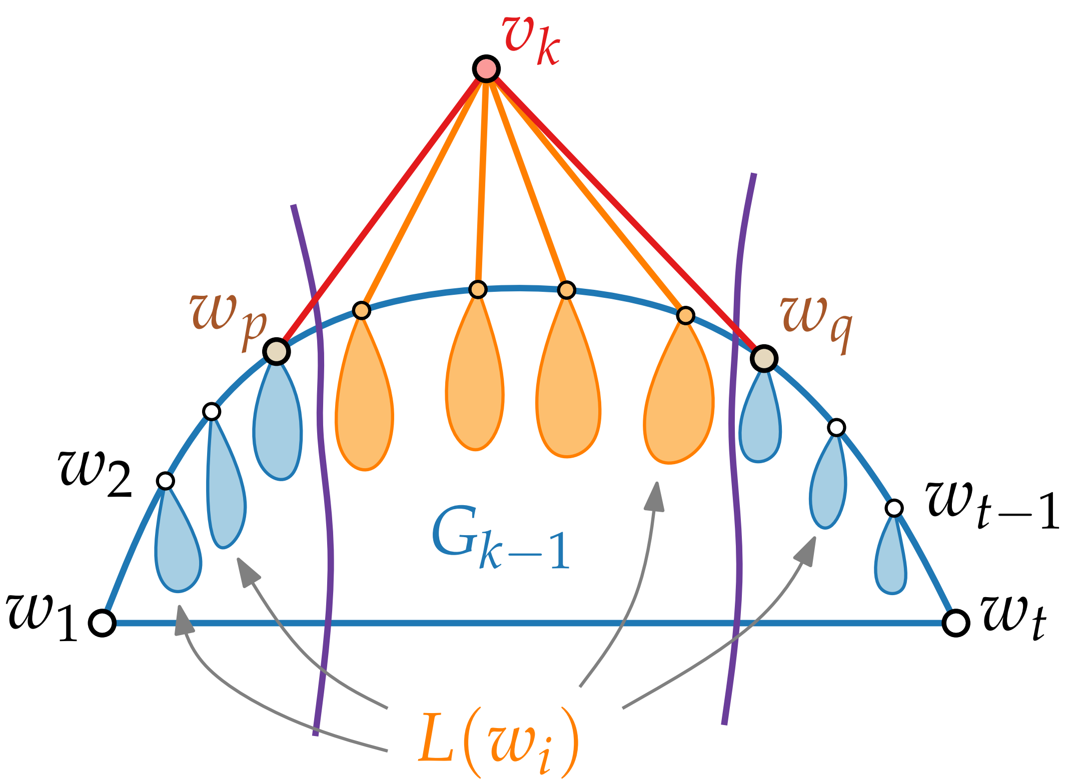


Shift Method – Planarity

Observations.

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

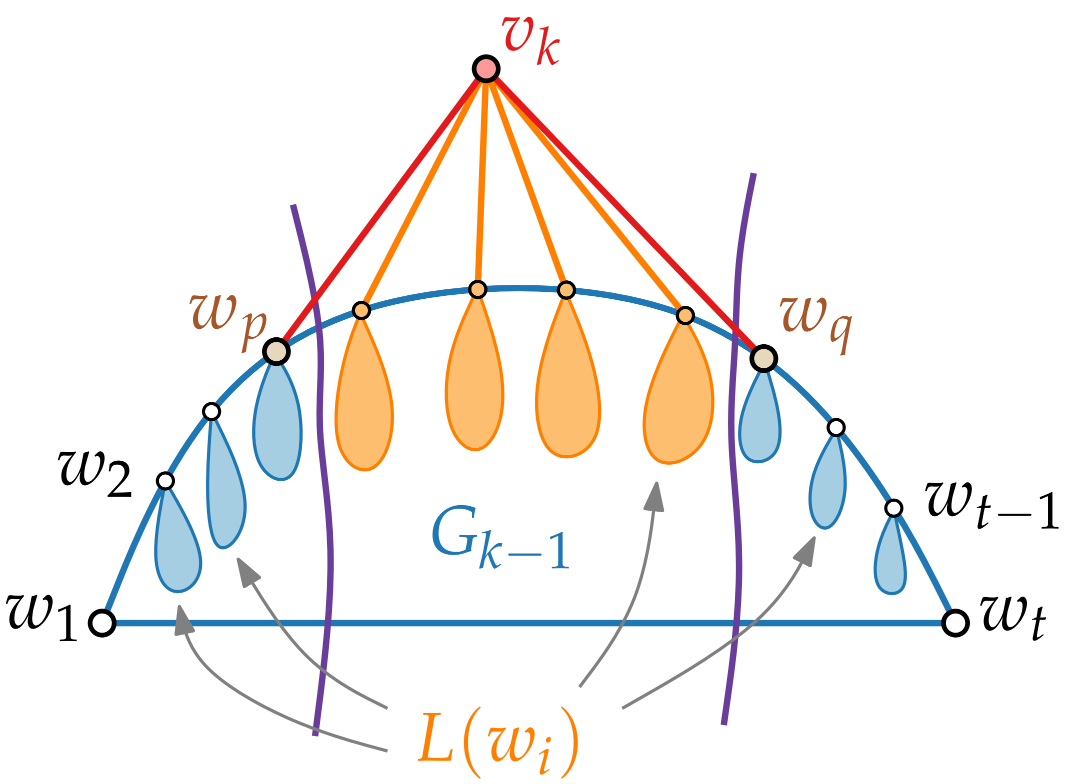
Lemma.
Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, such that $\delta_q - \delta_p \geq 2$ and even. If we shift $L(w_i)$ by δ_i to the right, then we get a planar straight-line drawing.



Shift Method – Planarity

Observations.

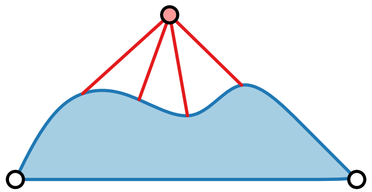
- Each internal vertex is covered exactly once.
- Covering relation defines a tree in G
- and a forest in G_i, $1 \leq i \leq n-1$.



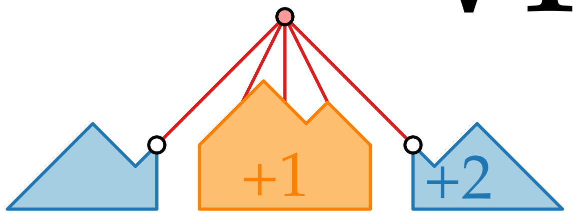
Lemma.
Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, such that $\delta_q - \delta_p \geq 2$ and even. If we shift $L(w_i)$ by δ_i to the right, then we get a planar straight-line drawing.

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

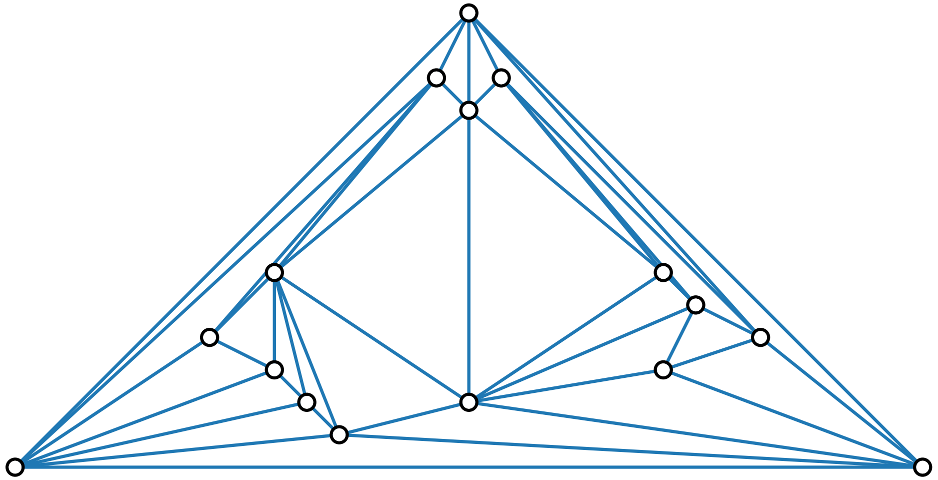
Universität Trier



Visualization of Graphs



Lecture 4:
Straight-Line Drawings of Planar Graphs I: Canonical Ordering and Shift Method



Part V:
Linear Time

Philipp Kindermann

Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G

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

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

Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G

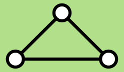
for $i = 1$ to 3 **do**

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

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

Shift Method – Pseudocode

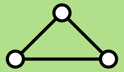
Let $v_1, \ldots, v_n$ be a canonical order of G
for $i = 1$ to 3 **do**
  $L(v_i) \leftarrow \{v_i\}$
$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$



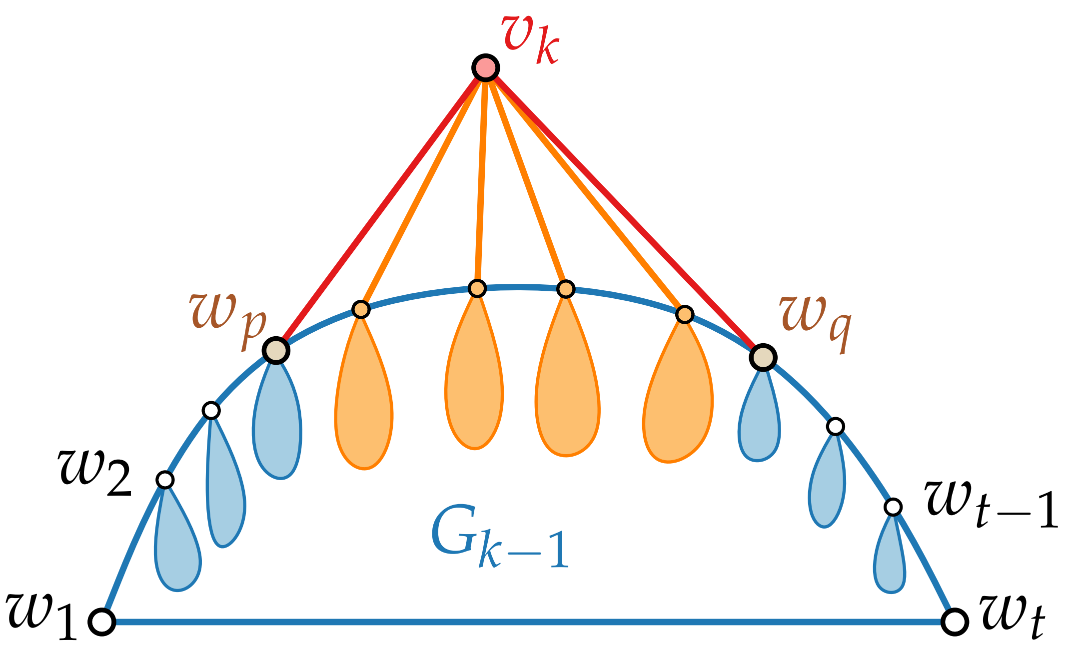
for $i = 4$ to n **do**

Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G
for $i = 1$ to 3 **do**
  $L(v_i) \leftarrow \{v_i\}$
$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$

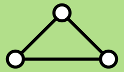


for $i = 4$ to n **do**
  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$
  denote the boundary of G_{i-1}
  and let $w_p, \ldots, w_q$ be the neighbours of v_i

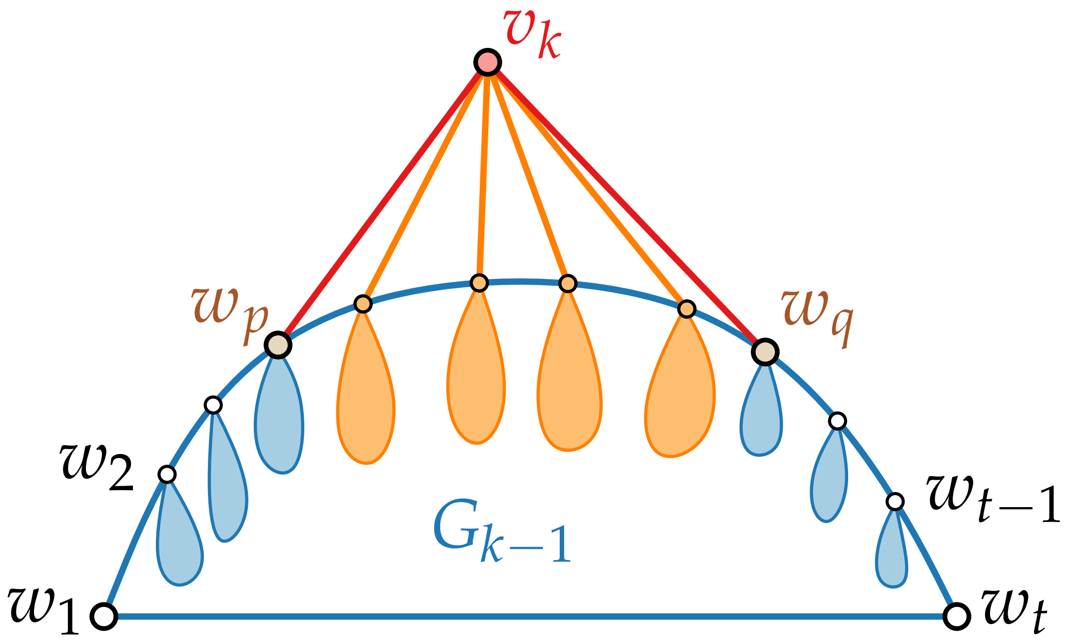


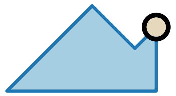
Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G
for $i = 1$ to 3 **do**
  $L(v_i) \leftarrow \{v_i\}$
$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$



for $i = 4$ to n **do**
  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$
  denote the boundary of G_{i-1}
  and let $w_p, \ldots, w_q$ be the neighbours of v_i





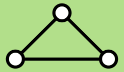
Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G

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

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

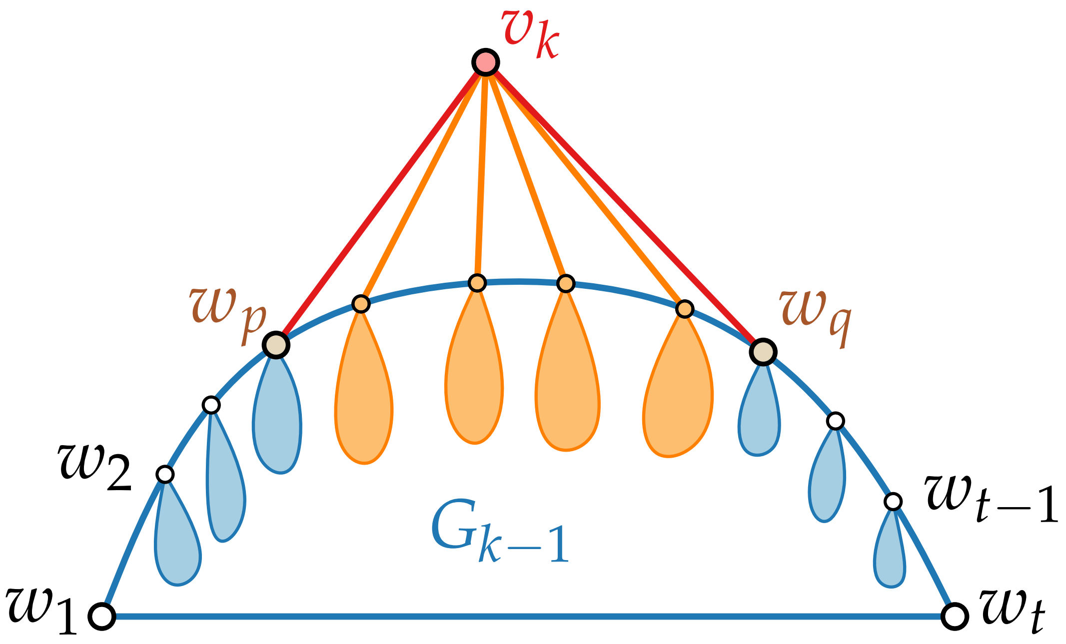
$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$

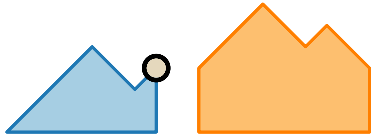


for $i = 4$ to n **do**

  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$
  denote the boundary of G_{i-1}
  and let $w_p, \ldots, w_q$ be the neighbours of v_i

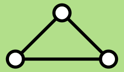
  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do**



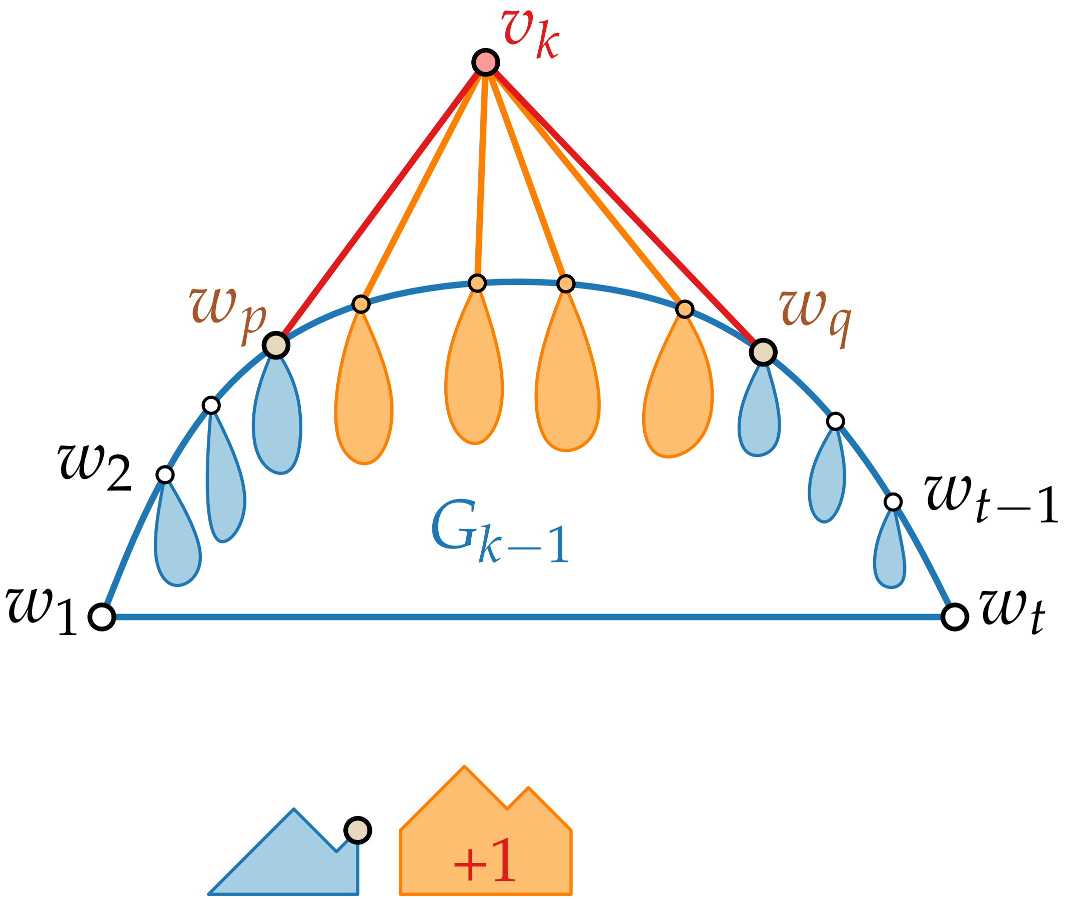


Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G
for $i = 1$ to 3 **do**
  $L(v_i) \leftarrow \{v_i\}$
$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$

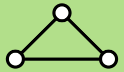


for $i = 4$ to n **do**
  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{i-1} and let $w_p, \ldots, w_q$ be the neighbours of v_i
  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do**
    $x(v) \leftarrow x(v) + 1$

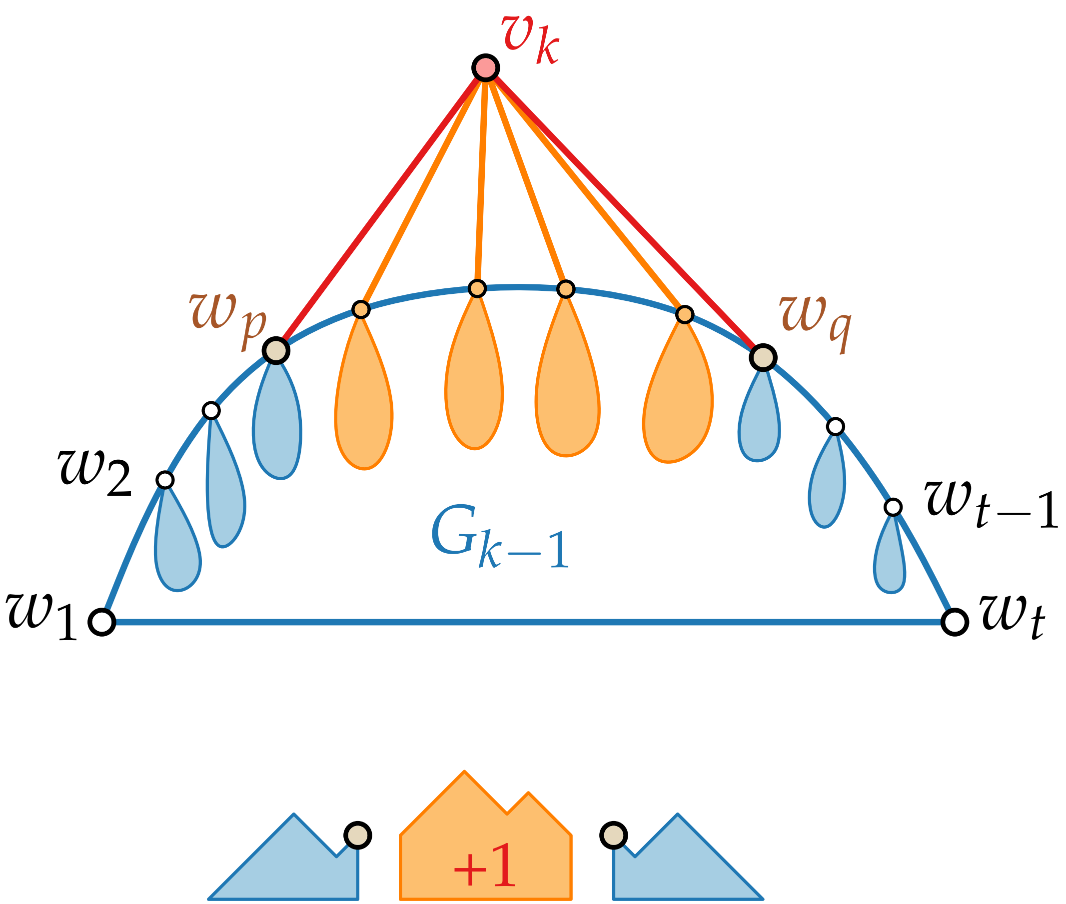


Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G
for $i = 1$ to 3 **do**
  $L(v_i) \leftarrow \{v_i\}$
$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$



for $i = 4$ to n **do**
  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$
  denote the boundary of G_{i-1}
  and let $w_p, \ldots, w_q$ be the neighbours of v_i
  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do**
    $x(v) \leftarrow x(v) + 1$
  for $\forall v \in \cup_{j=q}^{t} L(w_j)$ **do**



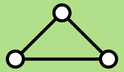
Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G

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

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

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



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

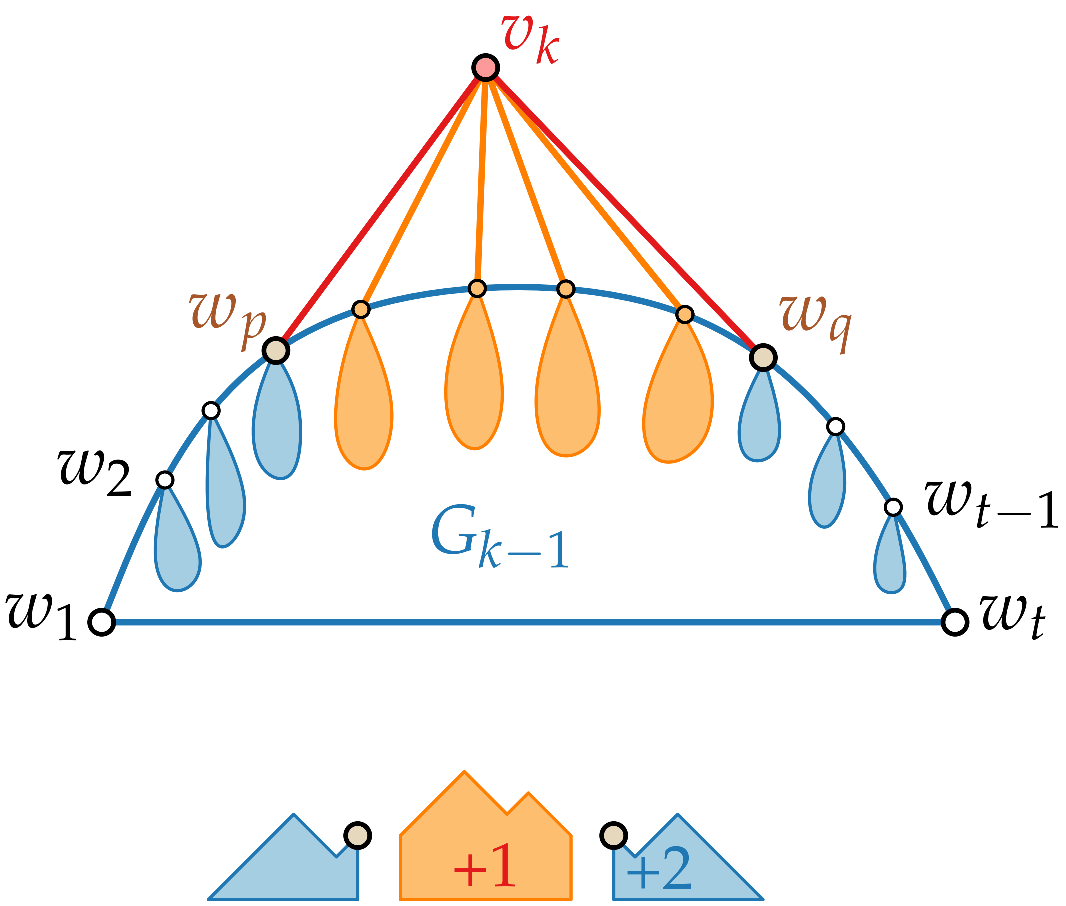
  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{i-1} and let $w_p, \ldots, w_q$ be the neighbours of v_i

  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do**

    $x(v) \leftarrow x(v) + 1$

  for $\forall v \in \cup_{j=q}^{t} L(w_j)$ **do**

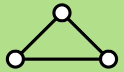
    $x(v) \leftarrow x(v) + 2$



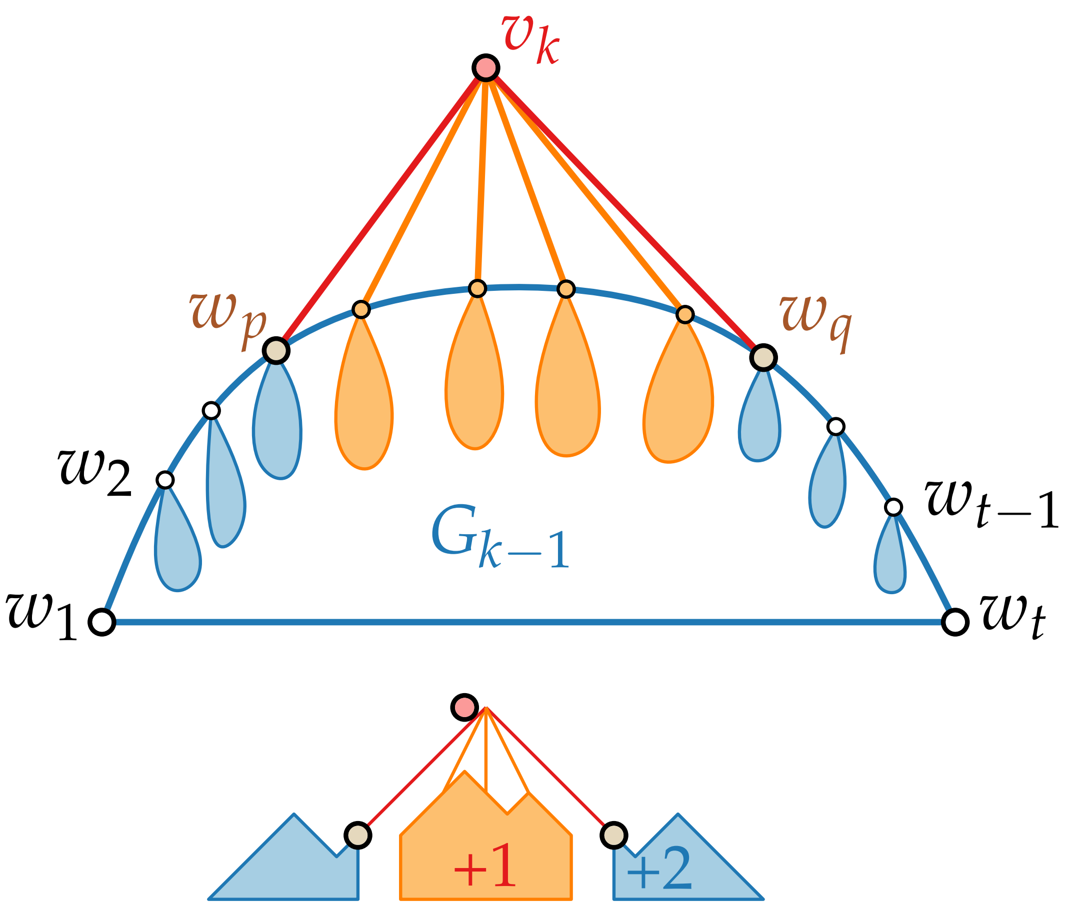
Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G

for $i = 1$ to 3 **do**
  $L(v_i) \leftarrow \{v_i\}$

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

for $i = 4$ to n **do**
  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$
  denote the boundary of G_{i-1}
  and let $w_p, \ldots, w_q$ be the neighbours of v_i
  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do**
    $x(v) \leftarrow x(v) + 1$
  for $\forall v \in \cup_{j=q}^{t} L(w_j)$ **do**
    $x(v) \leftarrow x(v) + 2$
  $P(v_i) \leftarrow$ intersection of $+1/-1$ diagonals
    through $P(w_p)$ and $P(w_q)$



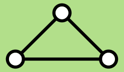
Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G

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

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

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



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

  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$
  denote the boundary of G_{i-1}
  and let $w_p, \ldots, w_q$ be the neighbours of v_i

  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do**

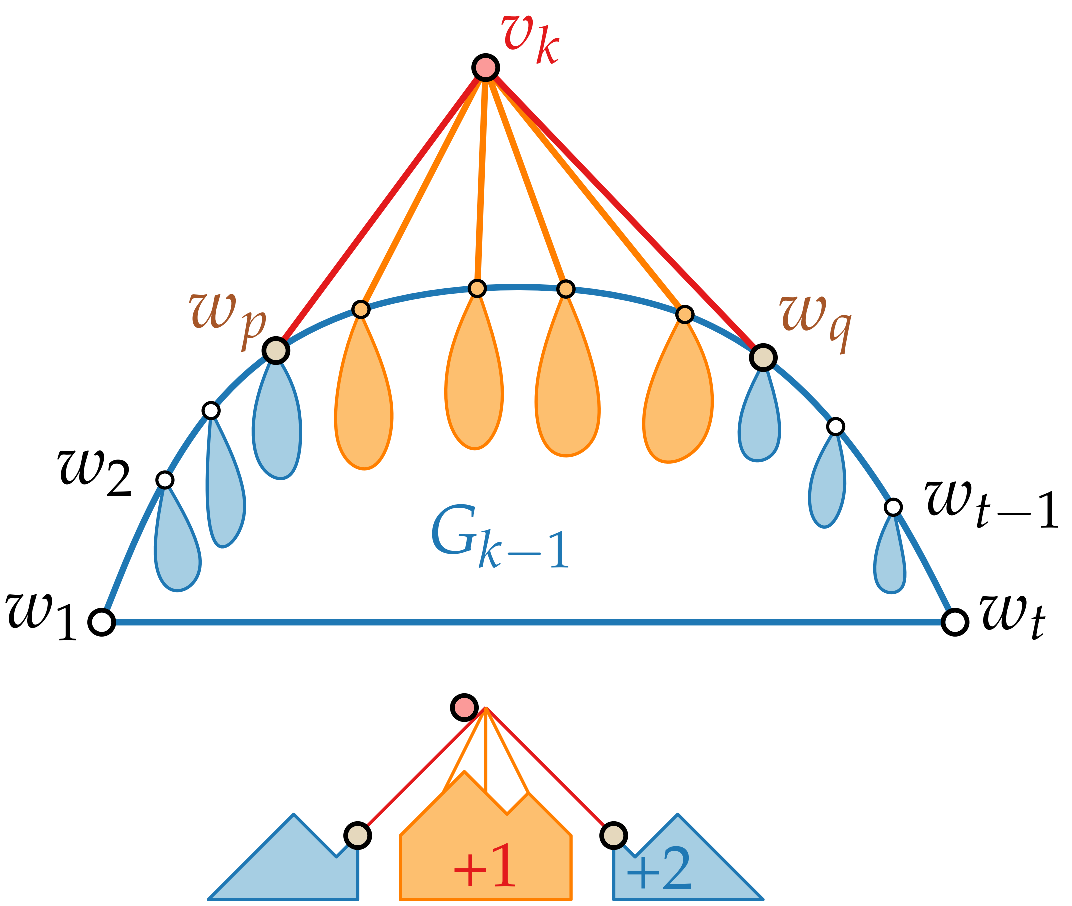
    $x(v) \leftarrow x(v) + 1$

  for $\forall v \in \cup_{j=q}^{t} L(w_j)$ **do**

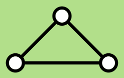
    $x(v) \leftarrow x(v) + 2$

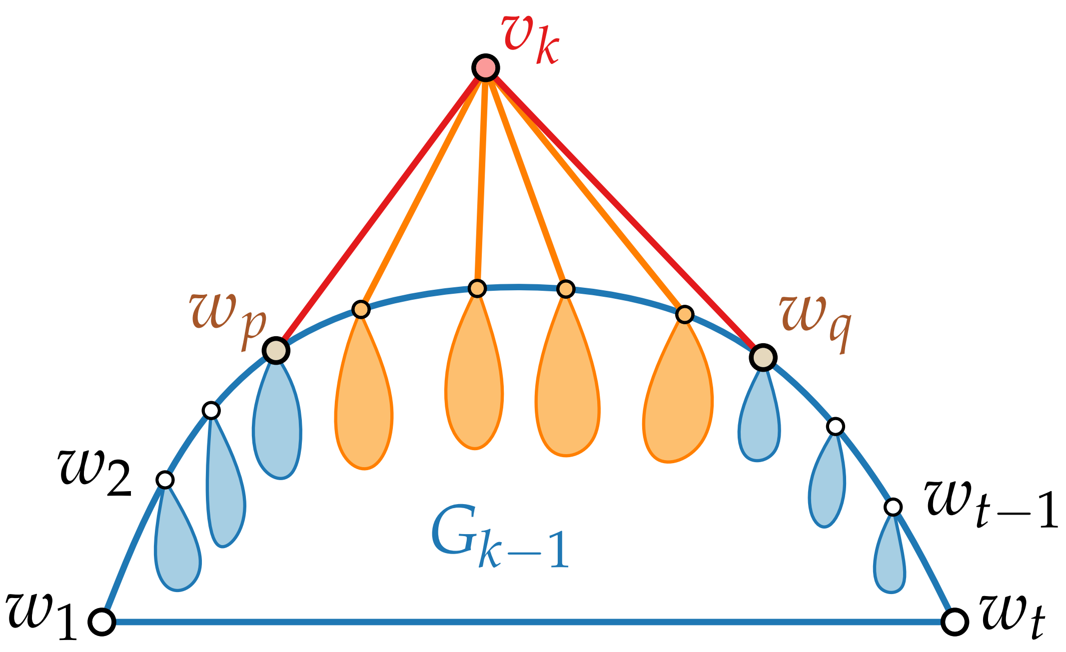
  $P(v_i) \leftarrow$ intersection of $+1/-1$ diagonals
  through $P(w_p)$ and $P(w_q)$

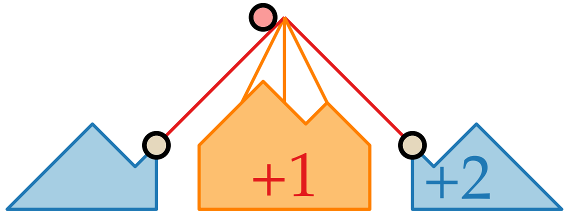
  $L(v_i) \leftarrow \cup_{j=p+1}^{q-1} L(w_j) \cup \{v_i\}$



Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G
for $i = 1$ to 3 **do**
  $L(v_i) \leftarrow \{v_i\}$
$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$ 
for $i = 4$ to n **do**
  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$
  denote the boundary of G_{i-1}
  and let $w_p, \ldots, w_q$ be the neighbours of v_i
  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do**
    $x(v) \leftarrow x(v) + 1$
  for $\forall v \in \cup_{j=q}^{t} L(w_j)$ **do**
    $x(v) \leftarrow x(v) + 2$
  $P(v_i) \leftarrow$ intersection of $+1/-1$ diagonals
    through $P(w_p)$ and $P(w_q)$
  $L(v_i) \leftarrow \cup_{j=p+1}^{q-1} L(w_j) \cup \{v_i\}$





Running Time?

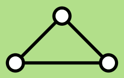
Shift Method – Pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G

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

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

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



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

  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{i-1} and let $w_p, \ldots, w_q$ be the neighbours of v_i

  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do** // $\mathcal{O}(n^2)$ *in total*

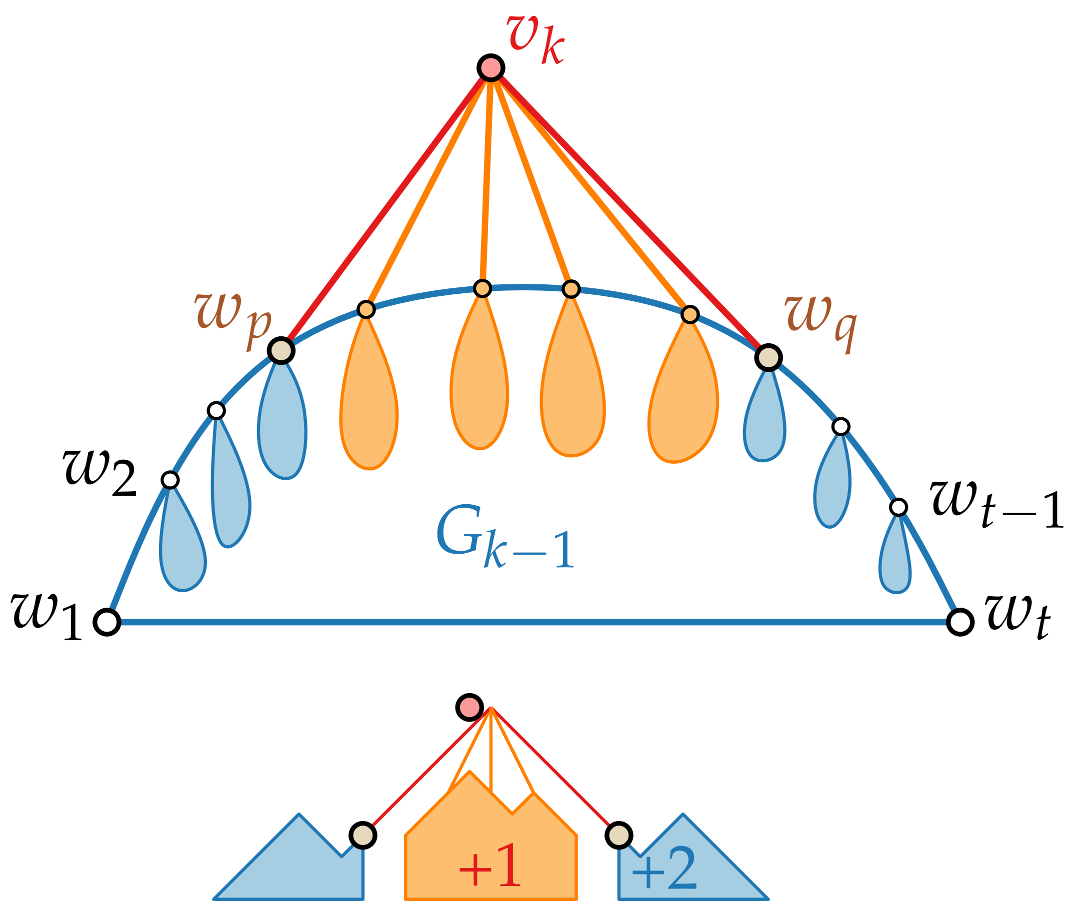
    $x(v) \leftarrow x(v) + 1$

  for $\forall v \in \cup_{j=q}^{t} L(w_j)$ **do** // $\mathcal{O}(n^2)$ *in total*

    $x(v) \leftarrow x(v) + 2$

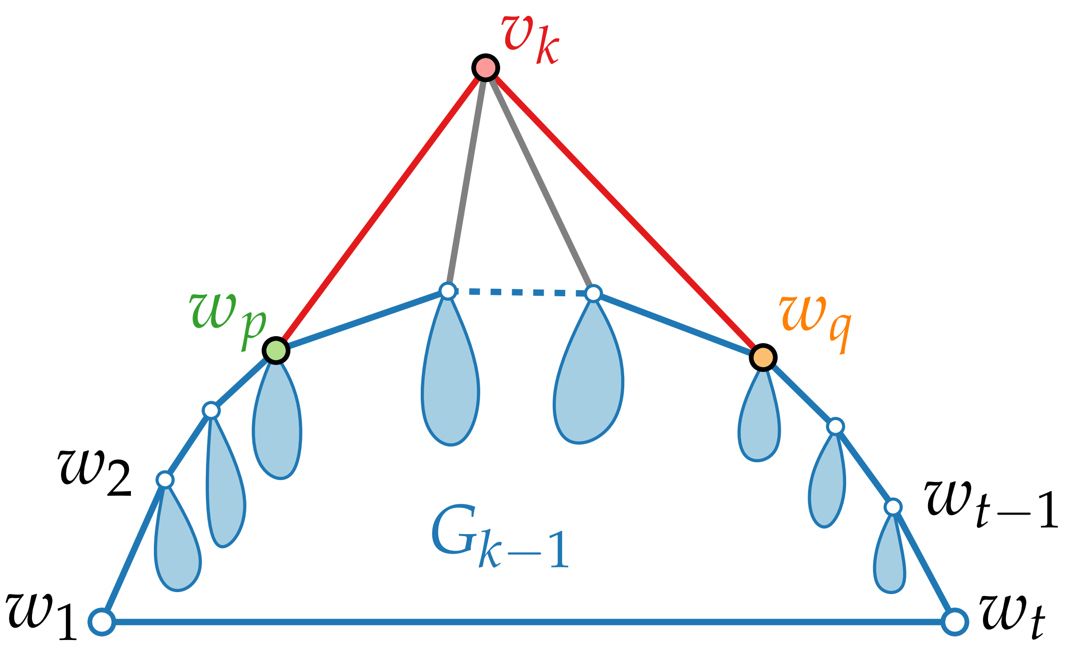
  $P(v_i) \leftarrow$ intersection of $+1/-1$ diagonals through $P(w_p)$ and $P(w_q)$

  $L(v_i) \leftarrow \cup_{j=p+1}^{q-1} L(w_j) \cup \{v_i\}$



Running Time?

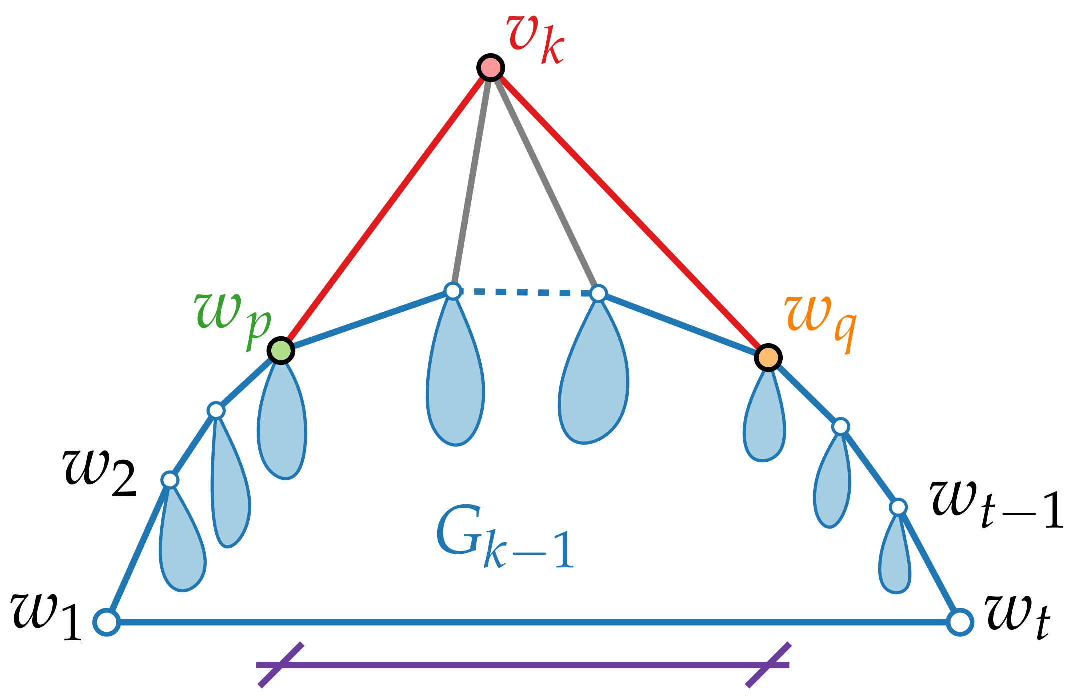
Shift Method – Linear Time Implementation



Shift Method – Linear Time Implementation

Idea 1.

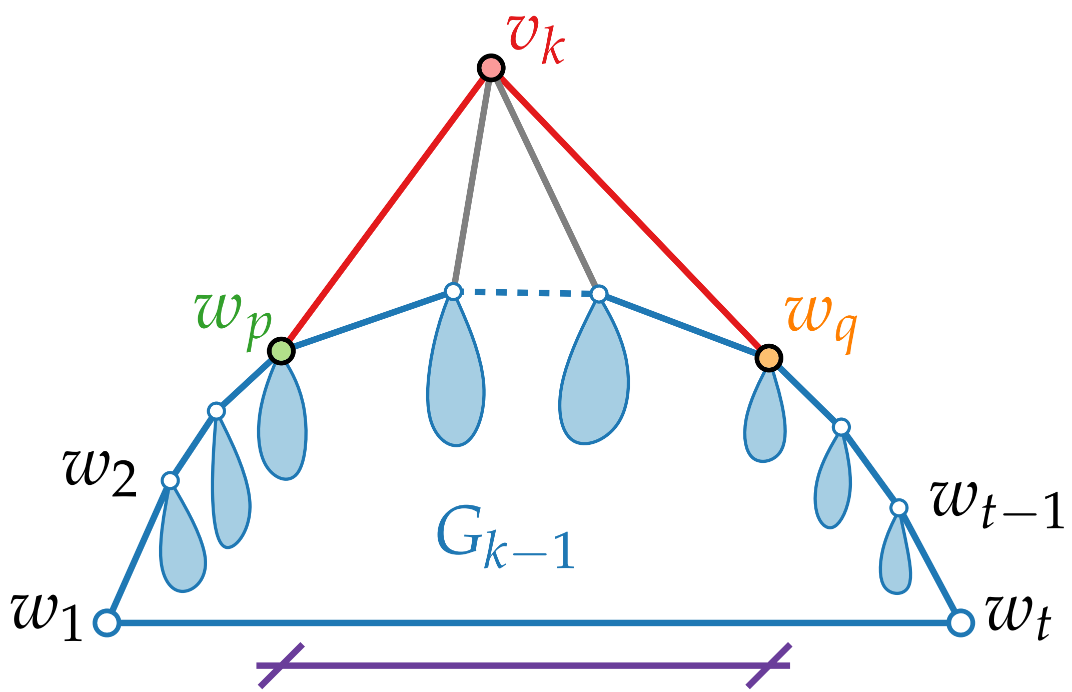
To compute $x(v_k)$ & $y(v_k)$,
we only need $y(w_p)$ and $y(w_q)$ and $x(w_q) - x(w_p)$



Shift Method – Linear Time Implementation

Idea 1.

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



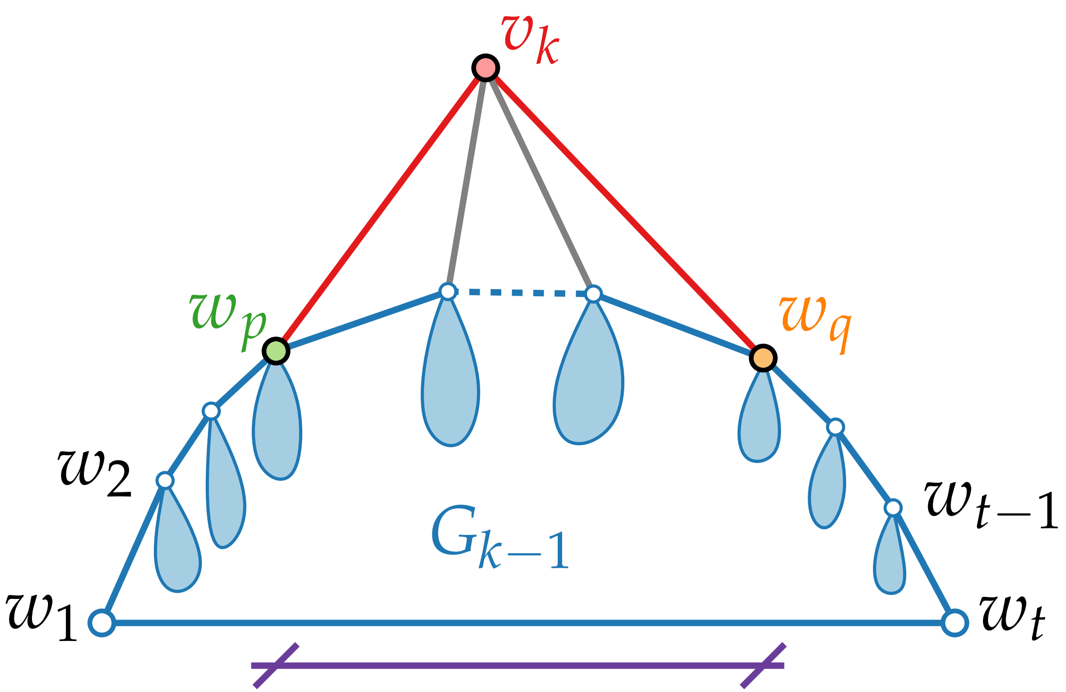
(1) $x(v_k) =$

(2) $y(v_k) =$

Shift Method – Linear Time Implementation

Idea 1.

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

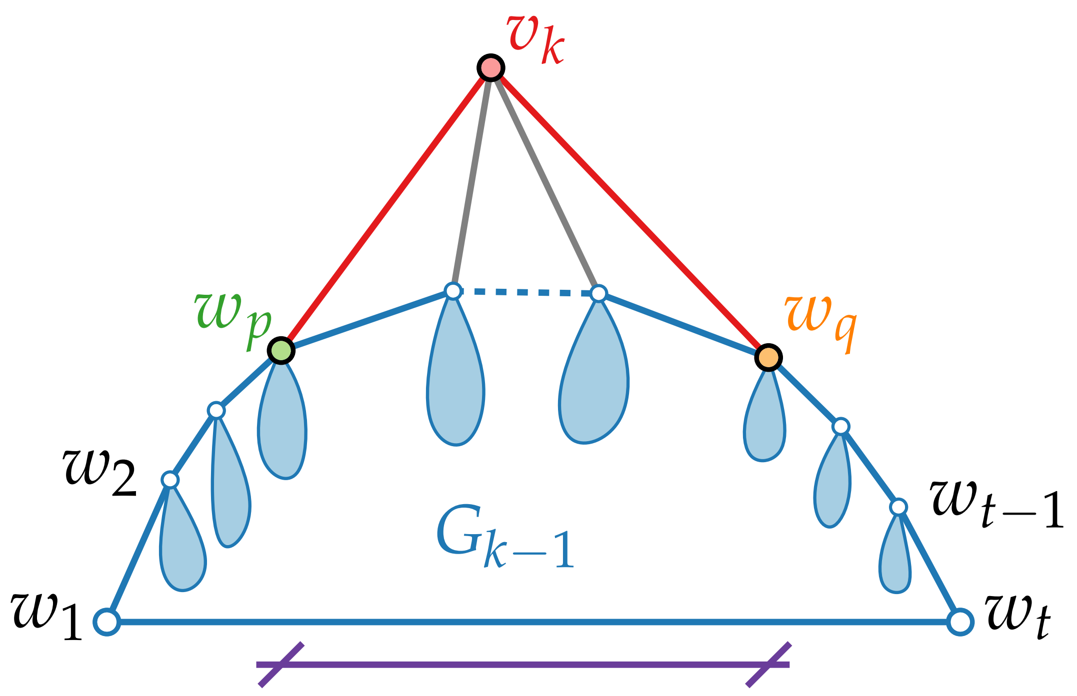


(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

(2) $y(v_k) =$

Shift Method – Linear Time Implementation

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



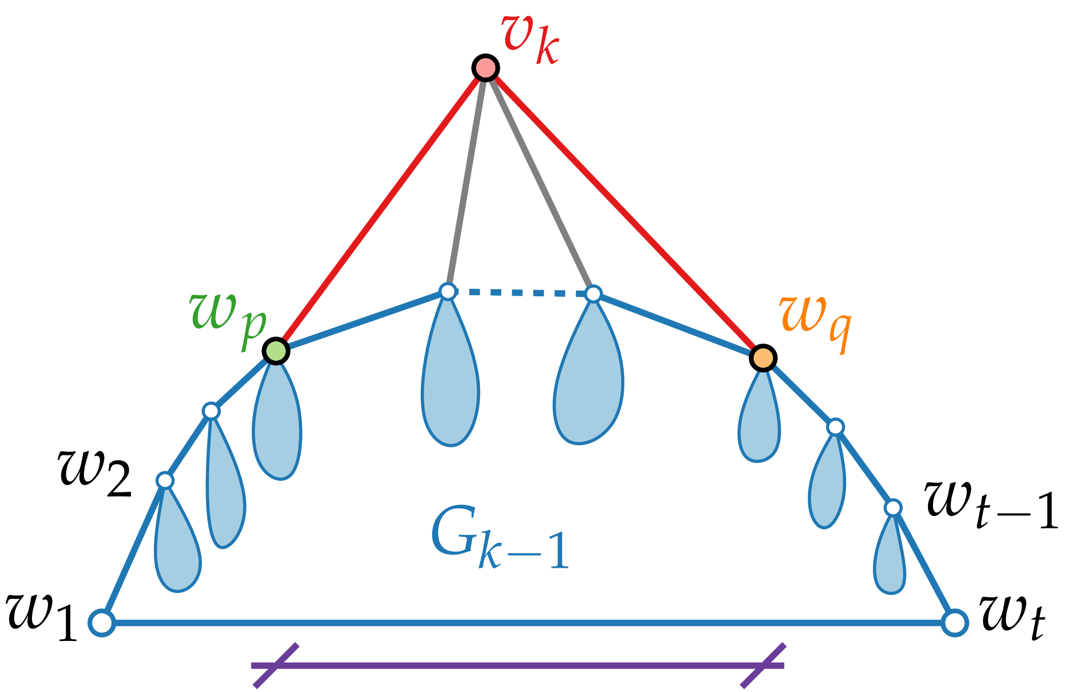
(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

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

Idea 2.
Instead of storing explicit x-coordinates,
we store x-distances.



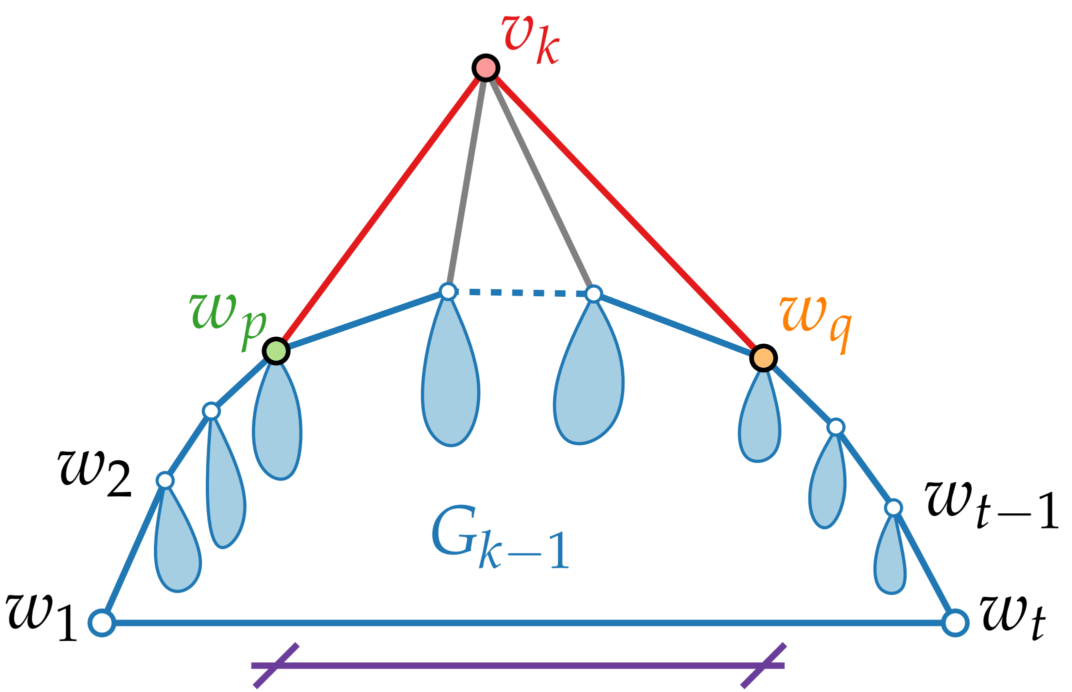
(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

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

Idea 2.
Instead of storing explicit x-coordinates,
we store x-distances.



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

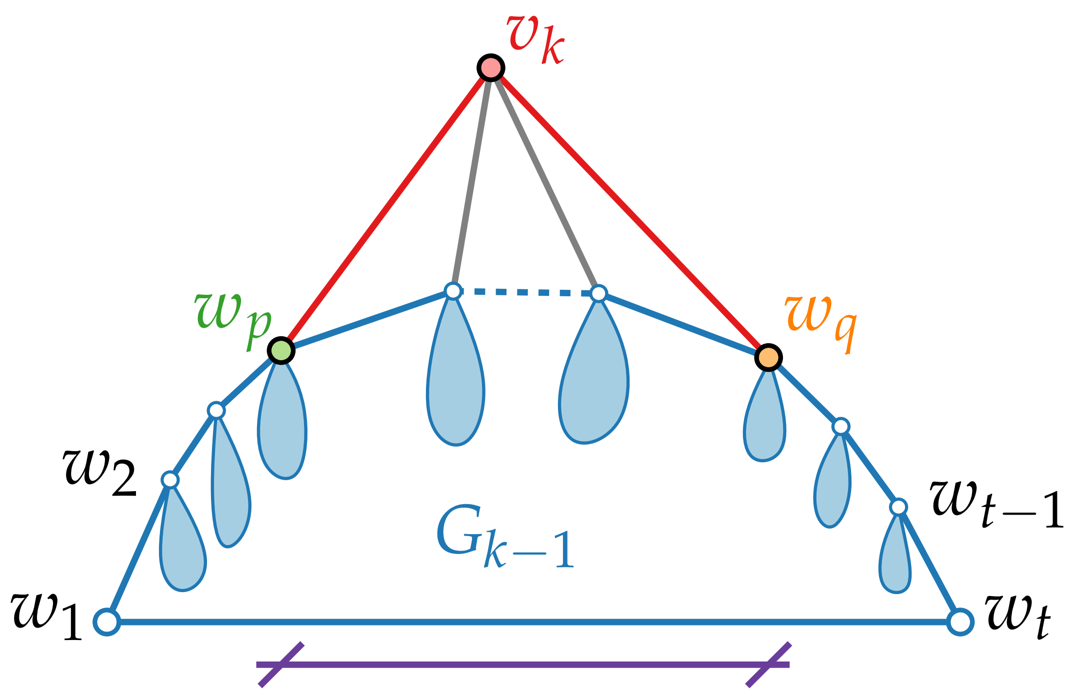
(2) $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$

(3) $x(v_k) - x(w_p) =$

Shift Method – Linear Time Implementation

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

Idea 2.
Instead of storing explicit x-coordinates,
we store x-distances.



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

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

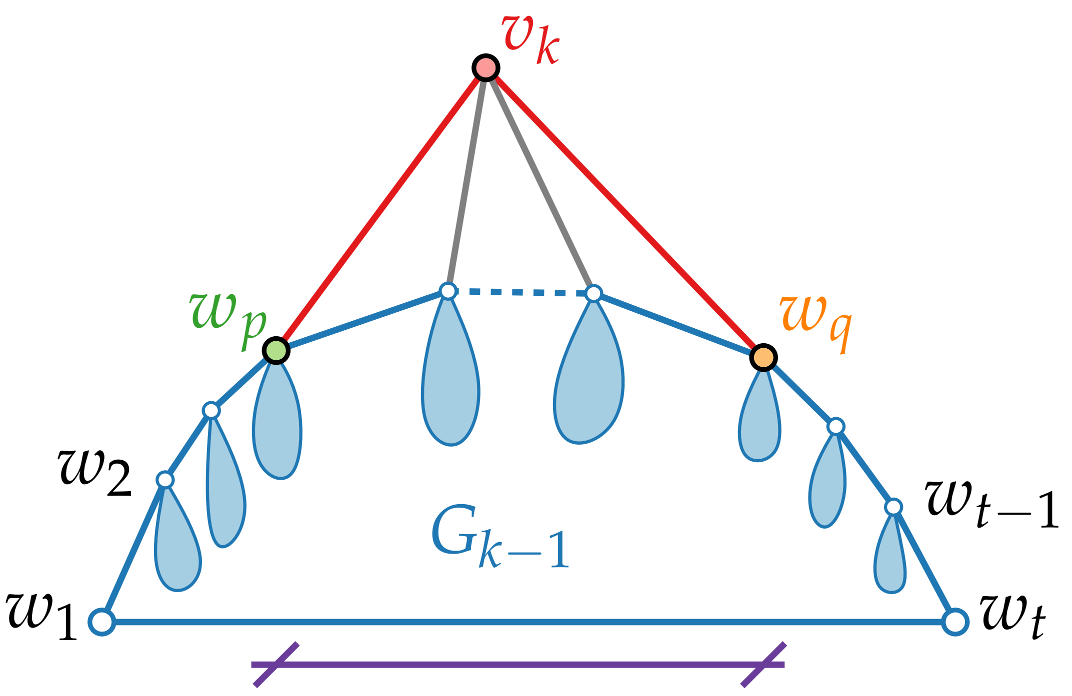
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear Time Implementation

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

Idea 2.
Instead of storing explicit x-coordinates,
we store x-distances.

After x distance for v_n computed, use preorder traversal to compute all x-coordinates.



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

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

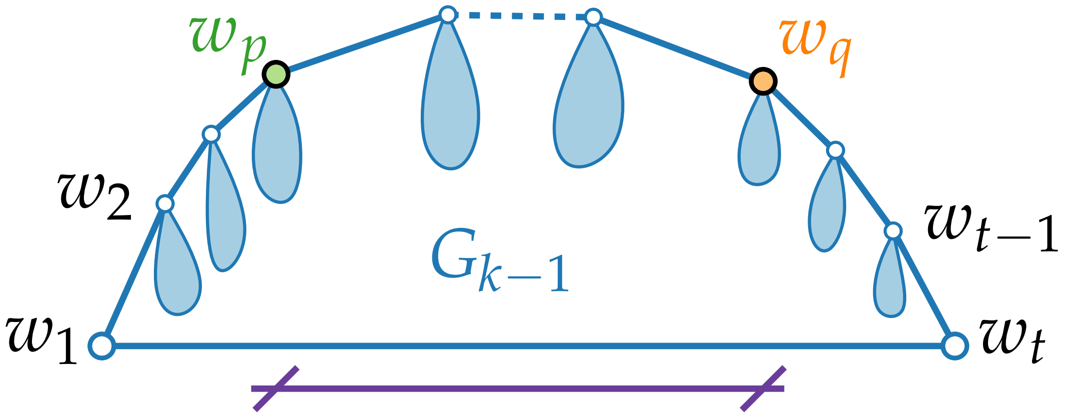
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

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

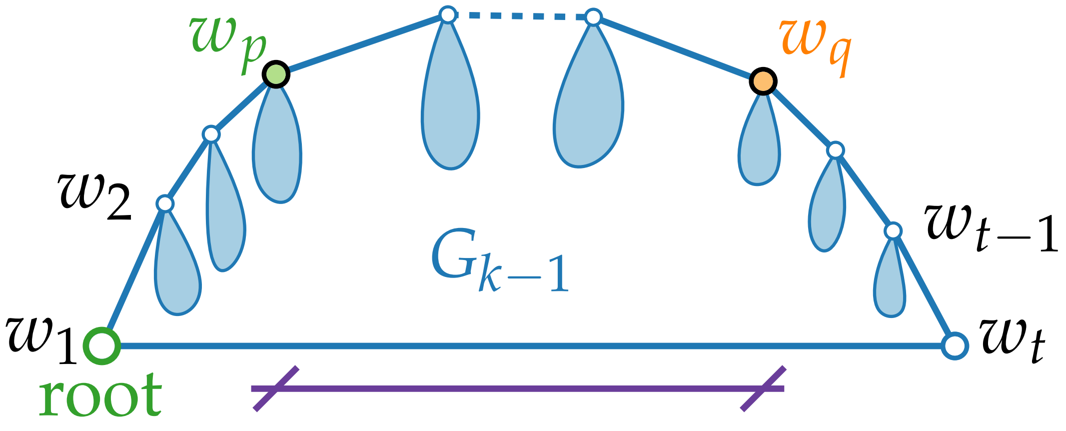
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

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

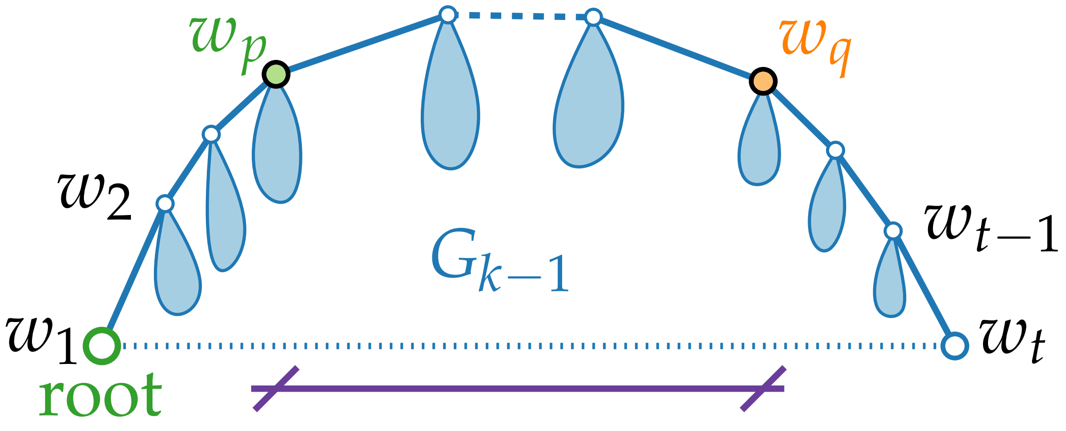
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

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

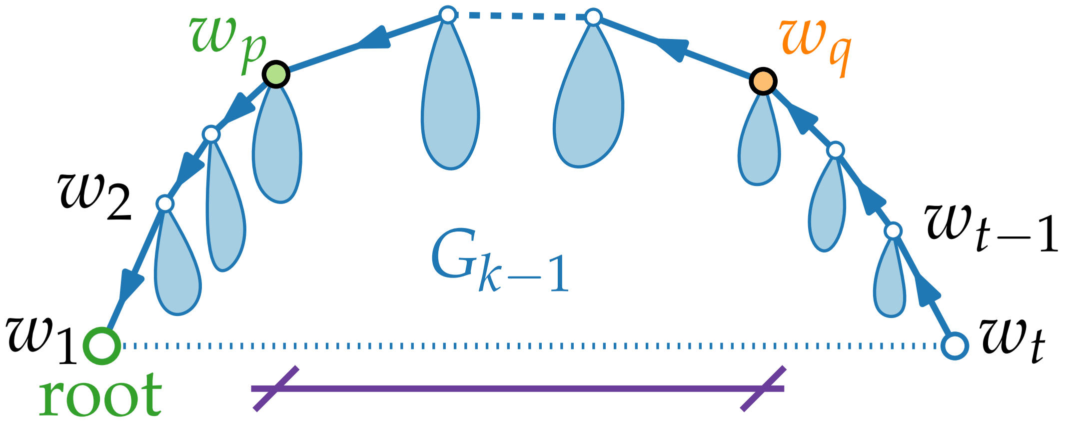
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift Method – Linear Time Implementation

Relative x-distance tree.

For each vertex v store

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



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

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

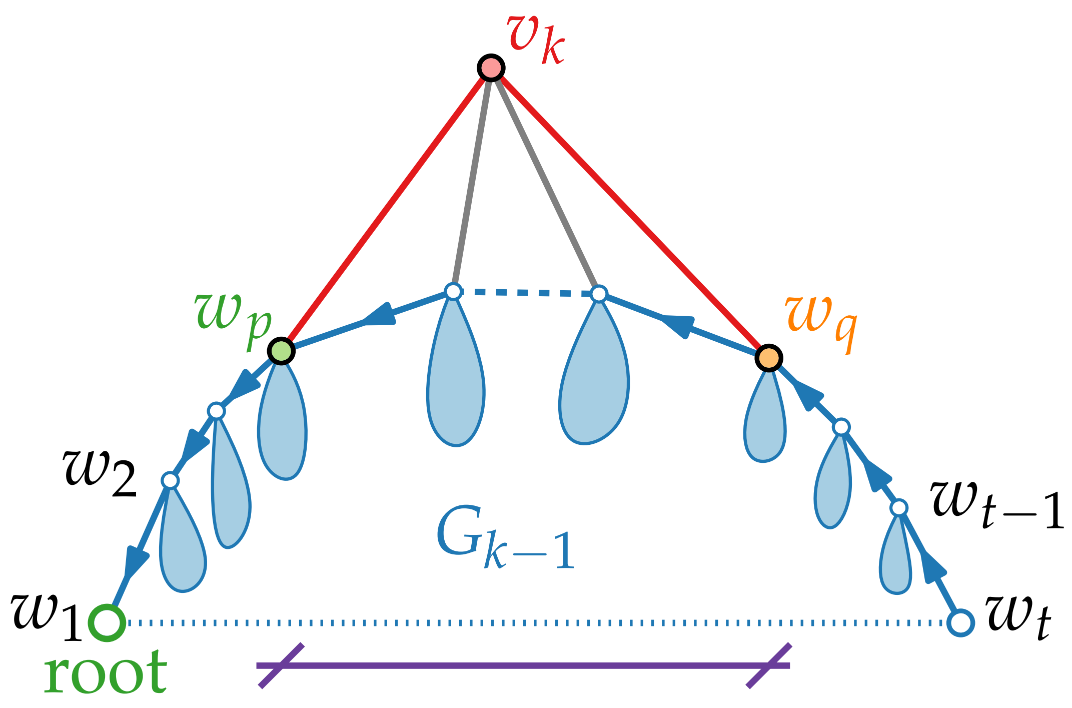
$$(3)\quad x(v_k) - x(w_p) = \tfrac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$$

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

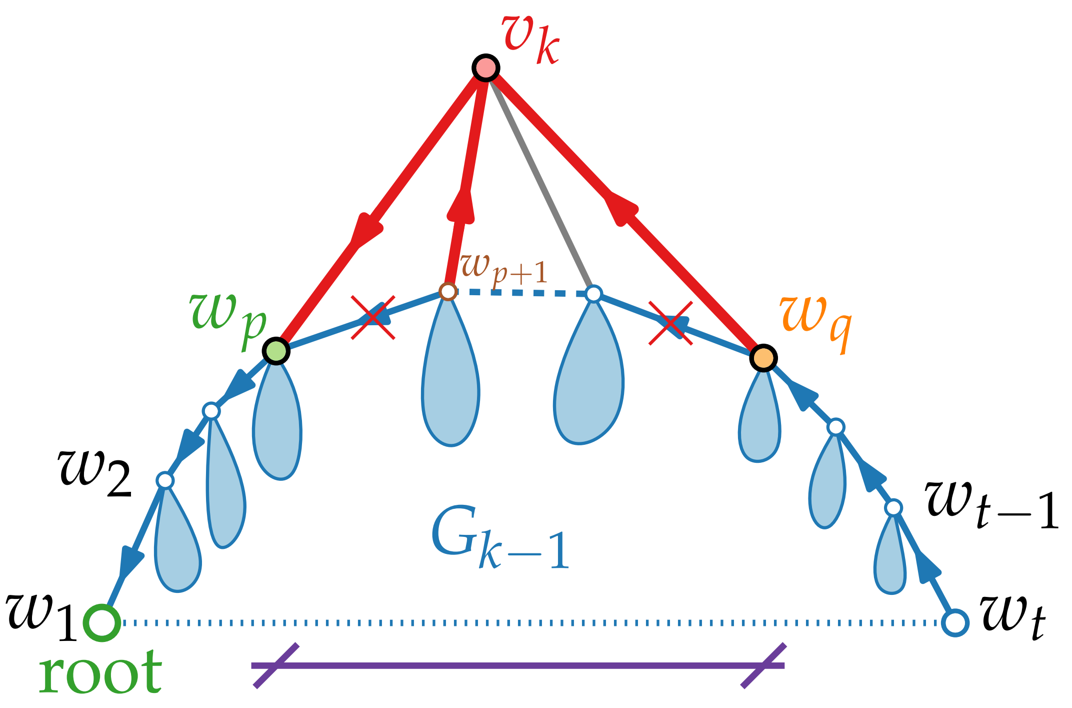
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

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

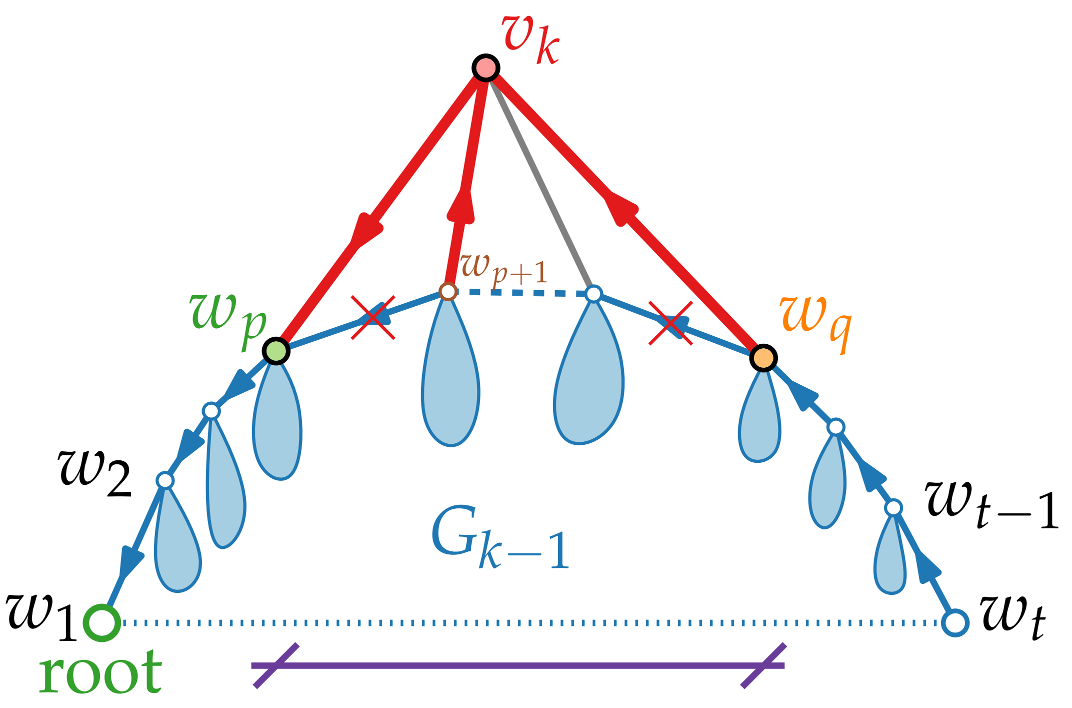
Relative x-distance tree.

For each vertex v store

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

Calculations.

- $\Delta_x(w_{p+1})$++, $\Delta_x(w_q)$++



(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

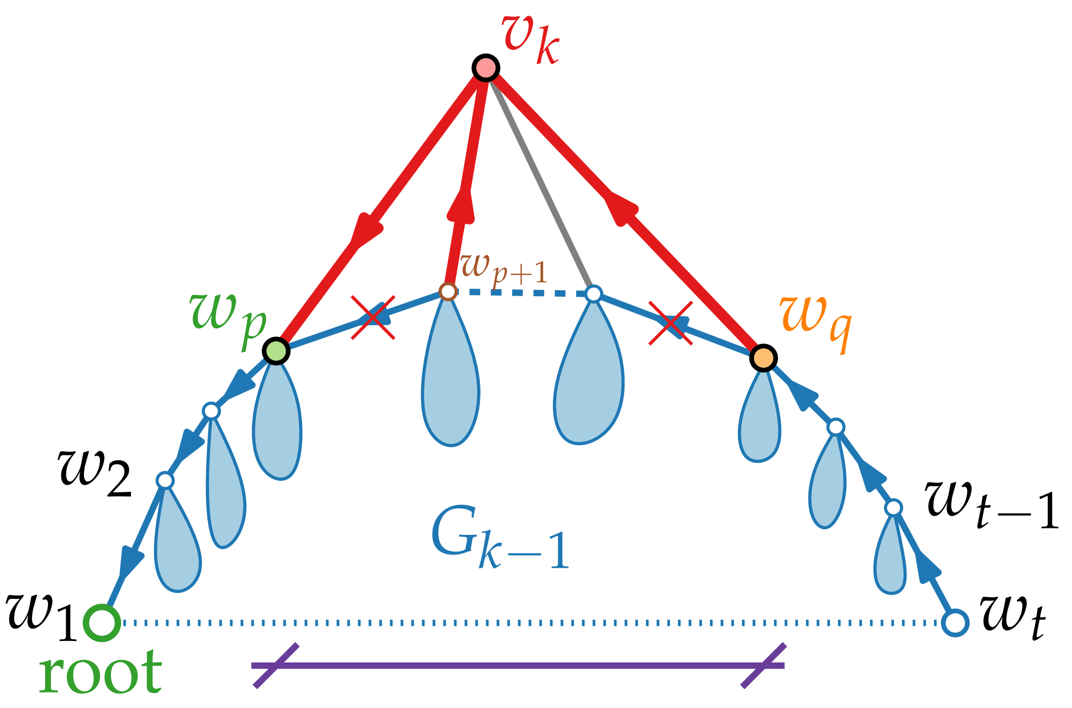
Relative x-distance tree.

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

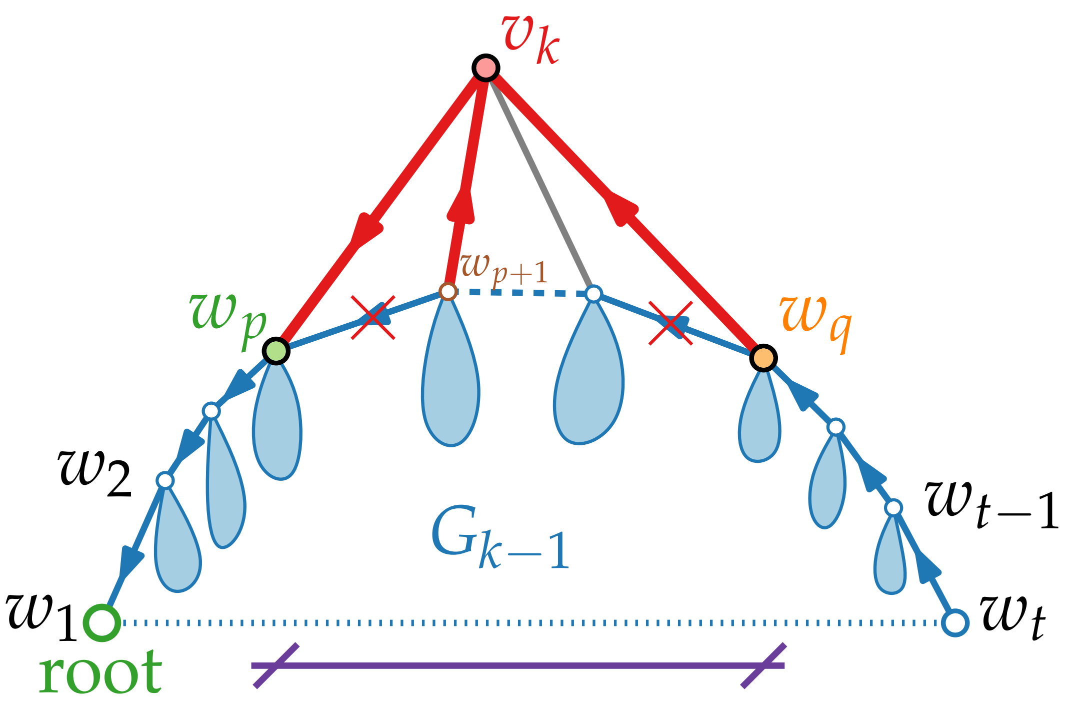
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

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

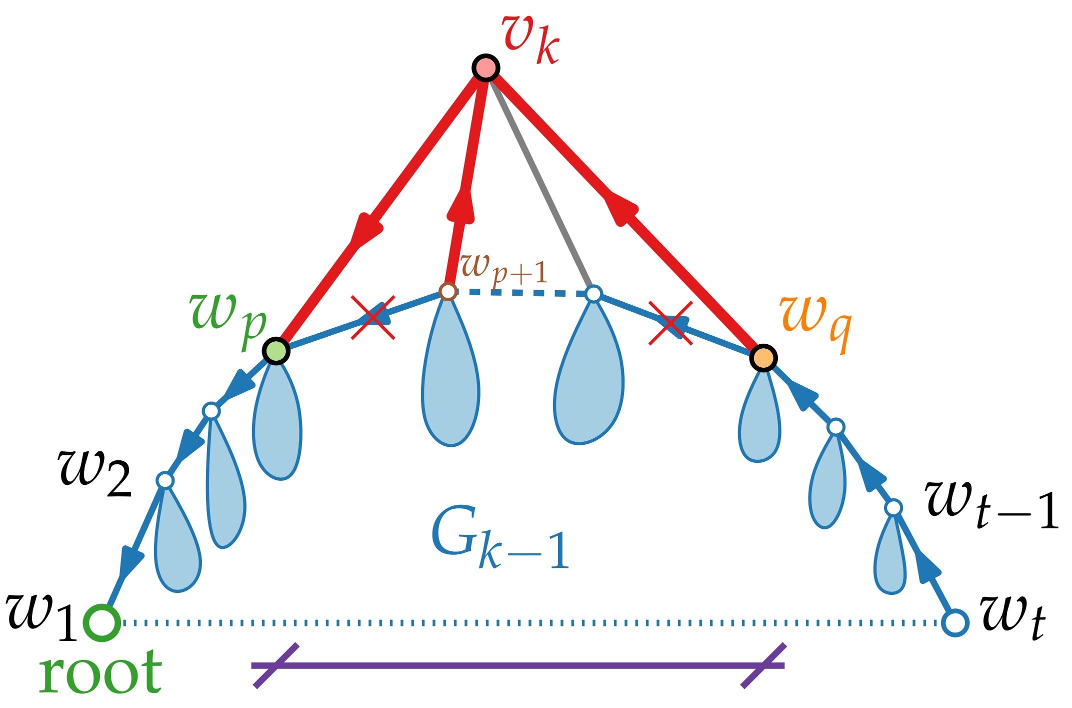
Relative x-distance tree.

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

Shift Method – Linear Time Implementation

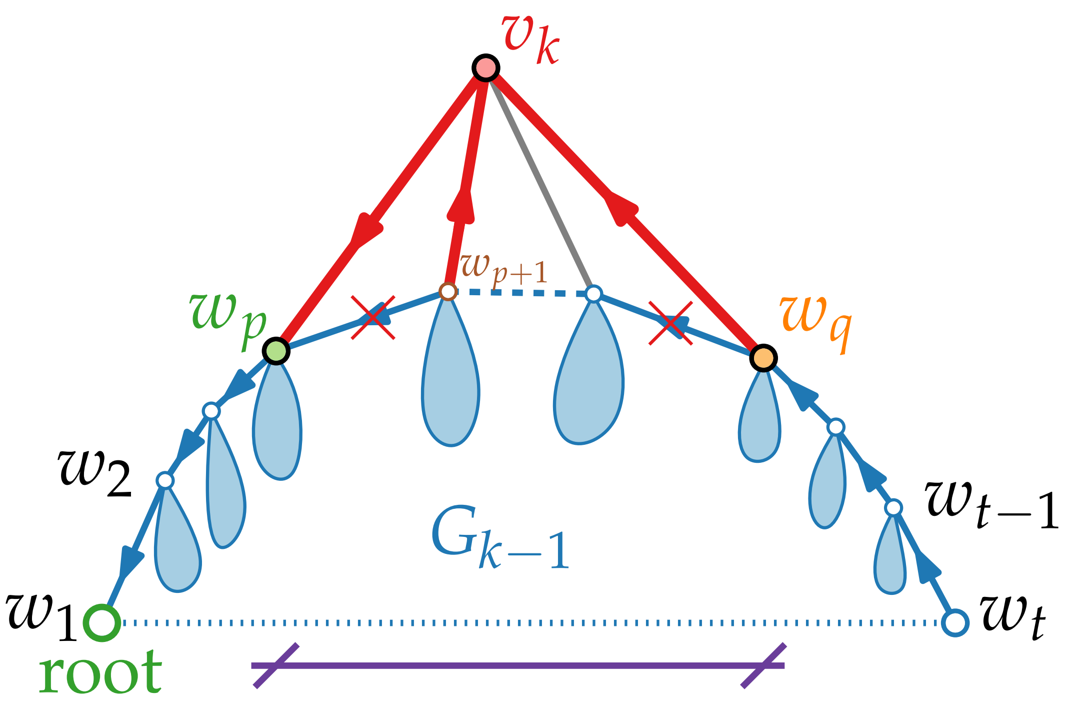
Relative x-distance tree.

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

Shift Method – Linear Time Implementation

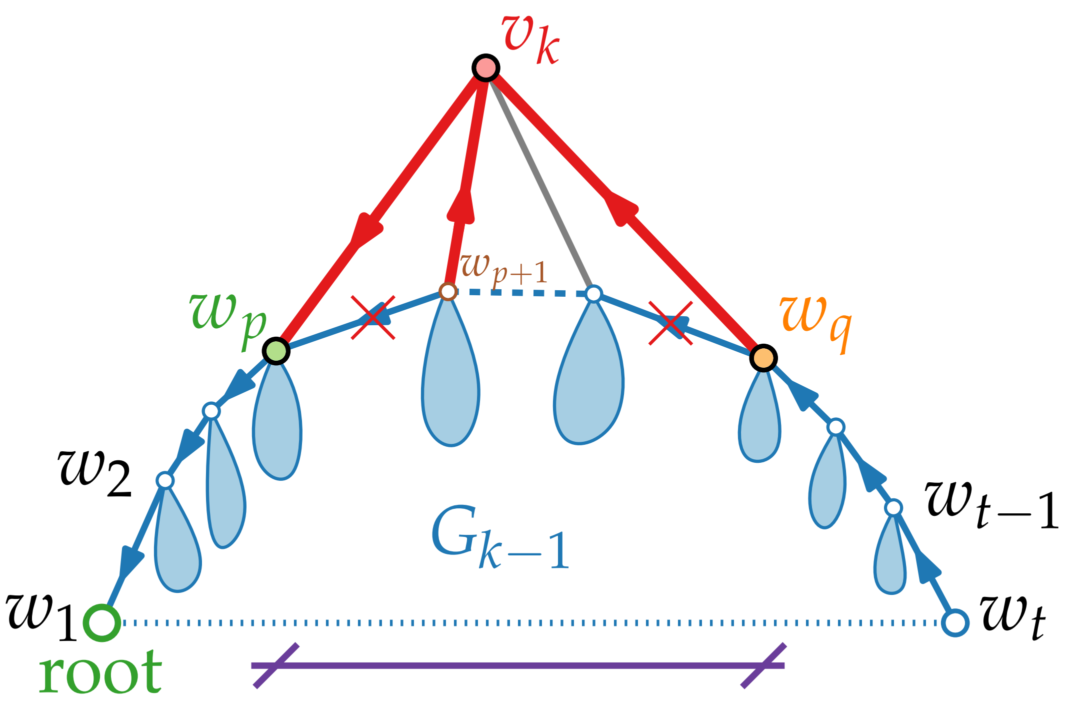
Relative x-distance tree.

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

Shift Method – Linear Time Implementation

Relative x-distance tree.

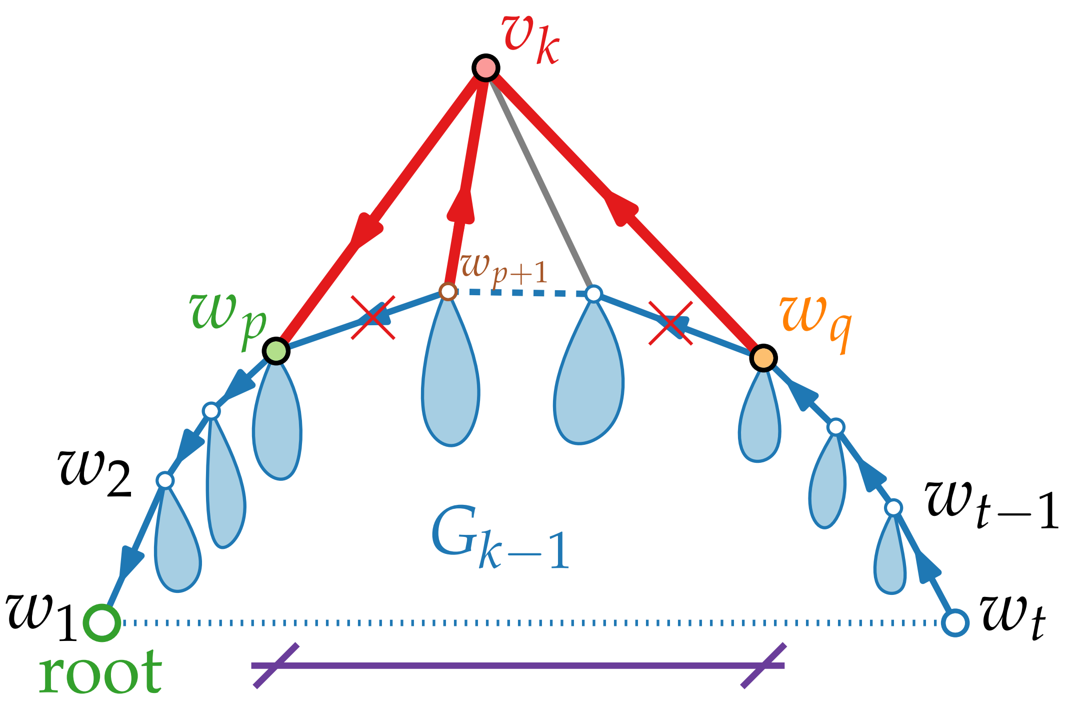
For each vertex v store

- 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)
- $y(v_k)$ by (2)
- $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$
- $\Delta_x(w_{p+1}) = \Delta_x(w_{p+1}) - \Delta_x(v_k)$

$\mathcal{O}(n)$ in total



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

Result & Variations

Theorem. [De Fraysseix, Pach, Pollack ’90]
Every n-vertex planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$. Such a drawing can be computed in $O(n)$ time.

Result & Variations

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

Theorem. [Kant '96]
Every n-vertex 3-connected planar graph has a planar straight-line drawing of size $(2n-4) \times (n-2)$ where all faces are drawn convex. Such a drawing can be computed in $O(n)$ time.

Result & Variations

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

Theorem. [Chrobak & Kant '97]
Every n-vertex 3-connected planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$ where all faces are drawn convex. Such a drawing can be computed in $O(n)$ time.

Result & Variations

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

Theorem. [Chrobak & Kant '97]
Every n-vertex 3-connected planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$ where all faces are drawn convex. Such a drawing can be computed in $O(n)$ time.

Theorem. [Brandenburg '08]
Every n-vertex planar graph has a planar straight-line drawing of size $\frac{4}{3}n \times \frac{2}{3}n$. Such a drawing can be computed in $O(n)$ time.