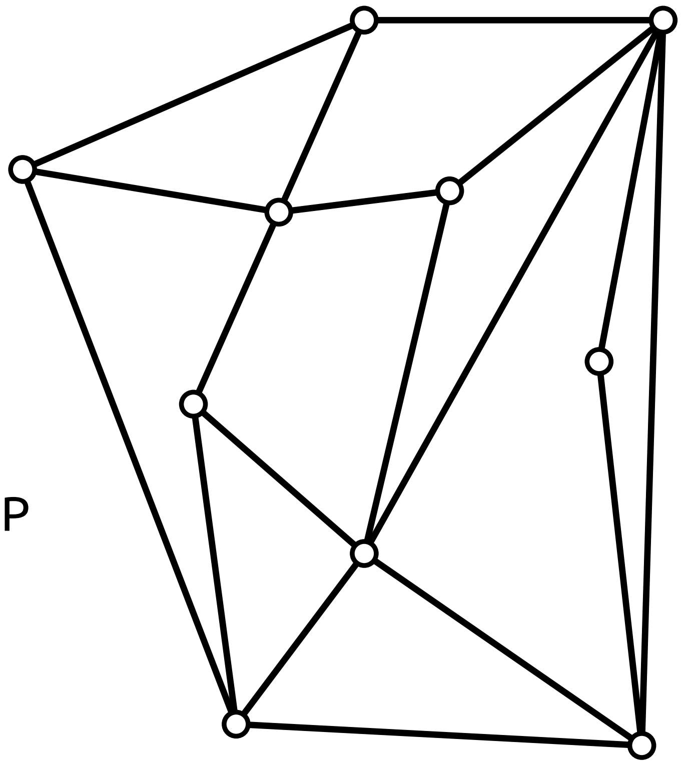
Straight-Line Planar Graph Drawing – Part 1

Lecture Graph Drawing Algorithms · 192.053

Martin Nöllenburg

17.04.2018



ALGORITHMS AND COMPLEXITY GROUP

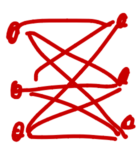
Planar Graphs

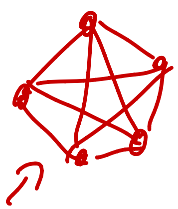
Planar graphs are an important graph class in graph drawing and graph theory.

What do you know about planar graphs?

- average degree ≤ 6
- Euler characteristic: $|V| - |E| + |F| = 2$

$|E| \leq 3|V| - 6$



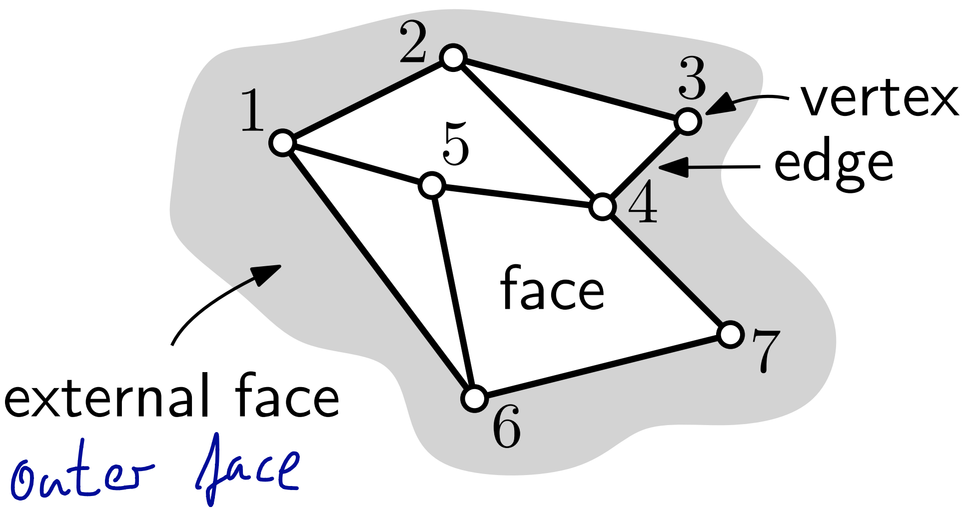


- no $K_{3,3}$ or K_5 as a minor
- 4 colorable

Planar Graphs

Planar graphs are an important graph class in graph drawing and graph theory.

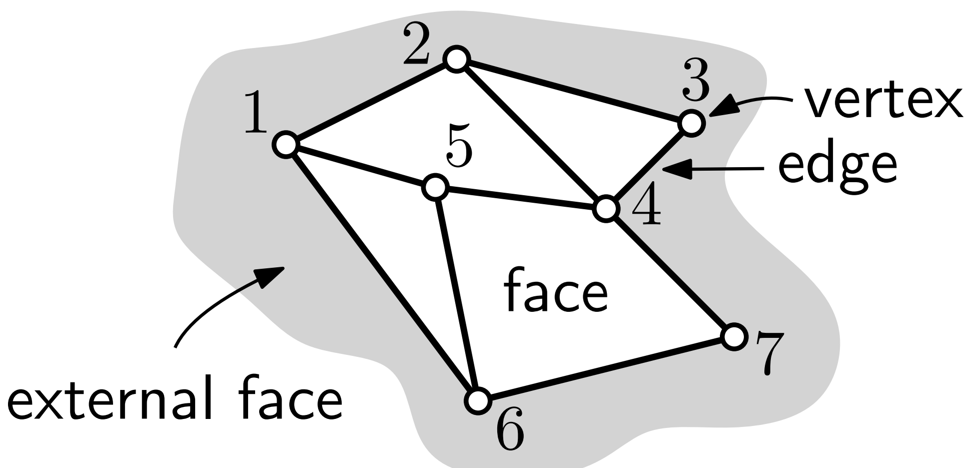
Def: A **planar graph** G is a simple graph that can be drawn/embedded in the plane $\mathbb{R}^2$ without edge crossings.



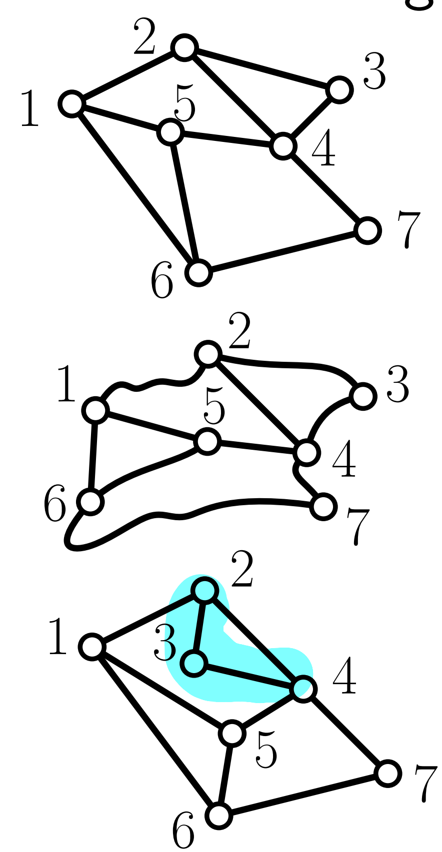
Planar Graphs

Planar graphs are an important graph class in graph drawing and graph theory.

Def: A **planar graph** G is a simple graph that can be drawn/ embedded in the plane $\mathbb{R}^2$ without edge crossings.



different drawings   same embedding?

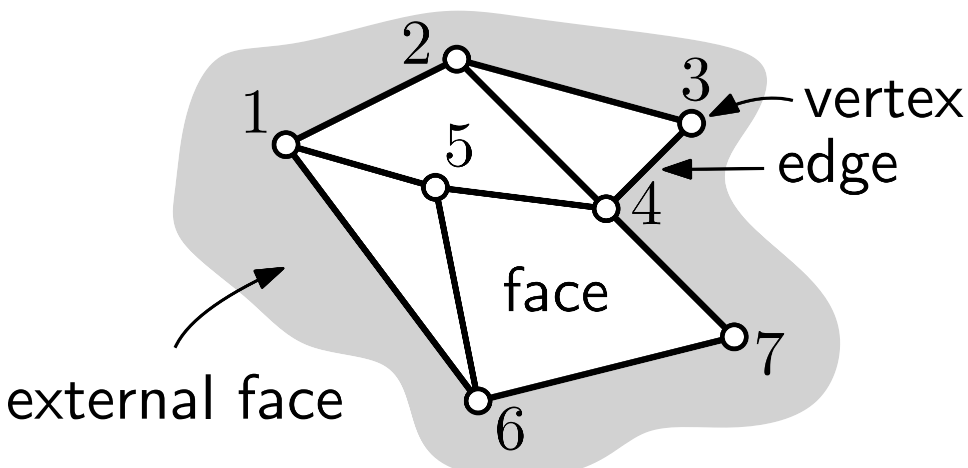


Planar Graphs

Planar graphs are an important graph class in graph drawing and graph theory.

plane graph = planar graph + embedding

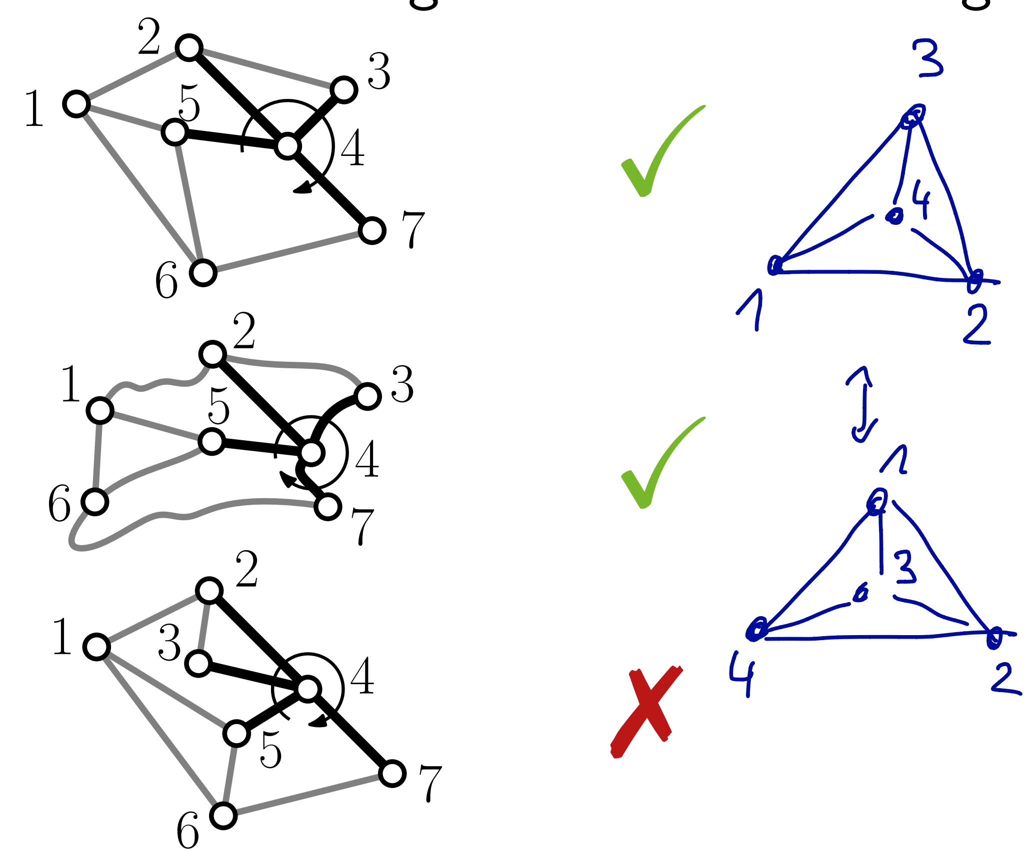
Def: A **planar graph** G is a simple graph that can be drawn/ embedded in the plane $\mathbb{R}^2$ without edge crossings.



Def: The **rotation scheme** of a planar drawing is the circular ordering of the edges incident to each vertex.

Two planar drawings have the same embedding if they have the same rotation scheme and the same external face.

different drawings same embedding?



Planar Graphs

Planar graphs are an important graph class in graph drawing and graph theory.

Def: A **planar graph** G is a simple graph that can be drawn/ embedded in the plane $\mathbb{R}^2$ without edge crossings.

Theorem: G is planar $\Leftrightarrow$ G has no K_5 or $K_{3,3}$ minor.

[Kuratowski 1930]

Planar Graphs

Planar graphs are an important graph class in graph drawing and graph theory.

Def: A **planar graph** G is a simple graph that can be drawn/ embedded in the plane $\mathbb{R}^2$ without edge crossings.

Theorem: G is planar $\Leftrightarrow$ G has no K_5 or $K_{3,3}$ minor.
[Kuratowski 1930]

Theorem: Testing whether a simple graph with n vertices is planar can be done in $O(n)$ time. [Hopcroft, Tarjan 1974]

Planar Graphs

Planar graphs are an important graph class in graph drawing and graph theory.

Def: A **planar graph** G is a simple graph that can be drawn/ embedded in the plane $\mathbb{R}^2$ without edge crossings.

Theorem: G is planar $\Leftrightarrow$ G has no K_5 or $K_{3,3}$ minor.
[Kuratowski 1930]

Theorem: Testing whether a simple graph with n vertices is planar can be done in $O(n)$ time. [Hopcroft, Tarjan 1974]

Theorem: Every planar graph has a planar straight-line drawing.
[Wagner 1936, Fáry 1948, Stein 1951]

Planar Graphs

Planar graphs are an important graph class in graph drawing and graph theory.

Def: A **planar graph** G is a simple graph that can be drawn/ embedded in the plane $\mathbb{R}^2$ without edge crossings.

Theorem: G is planar $\Leftrightarrow$ G has no K_5 or $K_{3,3}$ minor.
[Kuratowski 1930]

Theorem: Testing whether a simple graph with n vertices is planar can be done in $O(n)$ time. [Hopcroft, Tarjan 1974]

Theorem: Every planar graph has a planar straight-line drawing.
[Wagner 1936, Fáry 1948, Stein 1951]

Theorem: G is planar $\Leftrightarrow$ G has a disk contact representation.
[Koebe 1936]

Planar Graphs

Planar graphs are an important graph class in graph drawing and graph theory.

Def: A **planar graph** G is a simple graph that can be drawn/ embedded in the plane $\mathbb{R}^2$ without edge crossings.

Theorem: G is planar $\Leftrightarrow$ G has no K_5 or $K_{3,3}$ minor.
[Kuratowski 1930]

Theorem: Testing whether a simple graph with n vertices is planar can be done in $O(n)$ time. [Hopcroft, Tarjan 1974]

Theorem: Every planar graph has a planar straight-line drawing.
[Wagner 1936, Fáry 1948, Stein 1951]

Theorem: G is planar $\Leftrightarrow$ G has a disk contact representation.
[Koebe 1936]

Theorem: $G = (V, E)$ is planar $\Rightarrow |V| - |E| + |F| = 2$
[Euler 1707-1783]

Drawing Planar Graphs

How do we find good straight-line planar drawings?

Drawing Planar Graphs

How do we find good straight-line planar drawings?

Theorem: A straight-line convex drawing of a 3-connected planar graph can be computed in linear time. [Tutte 1963]

But: no polynomial bound on the required area

Drawing Planar Graphs

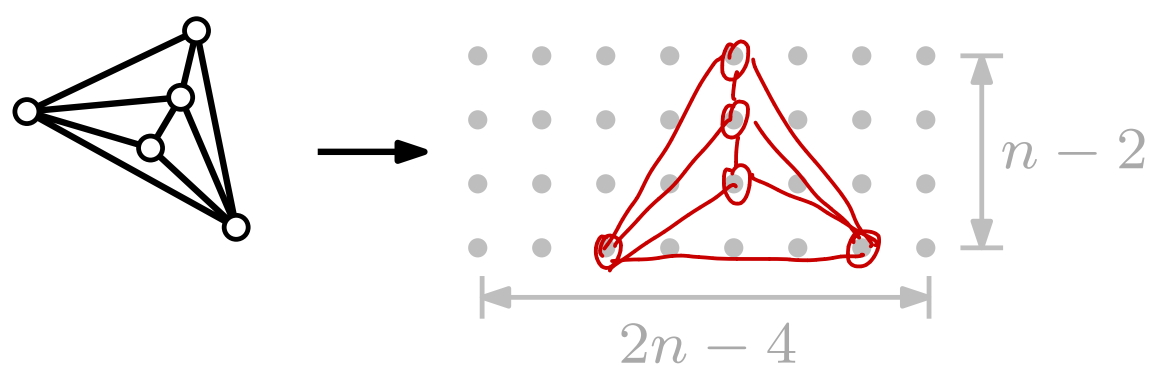
How do we find good straight-line planar drawings?

Theorem: A straight-line convex drawing of a 3-connected planar graph can be computed in linear time. [Tutte 1963]

But: no polynomial bound on the required area

Theorem: Every n-vertex embedded planar graph has a planar straight-line drawing on an $(2n-4) \times (n-2)$ grid.

[de Fraysseix, Pach, Pollack 1988]

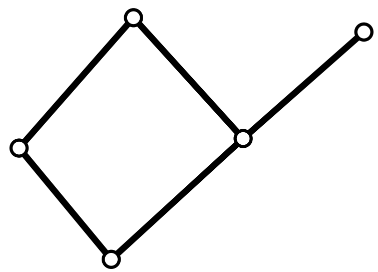


General Algorithm Idea

- sufficient to focus on drawing maximally planar (triangulated) graphs

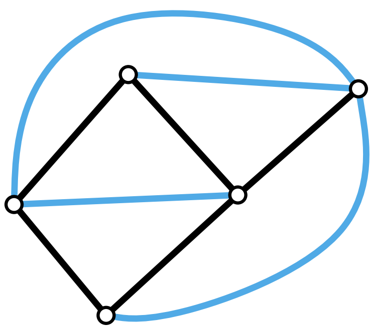
General Algorithm Idea

- sufficient to focus on drawing maximally planar (triangulated) graphs



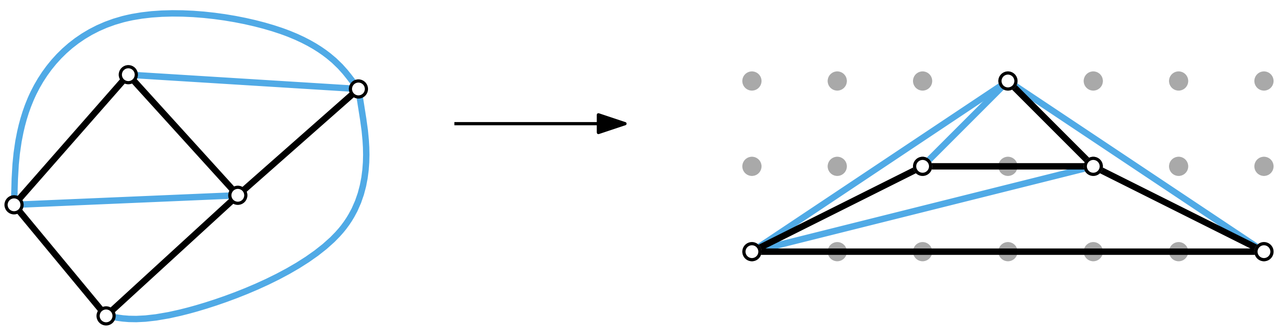
General Algorithm Idea

- sufficient to focus on drawing maximally planar (triangulated) graphs



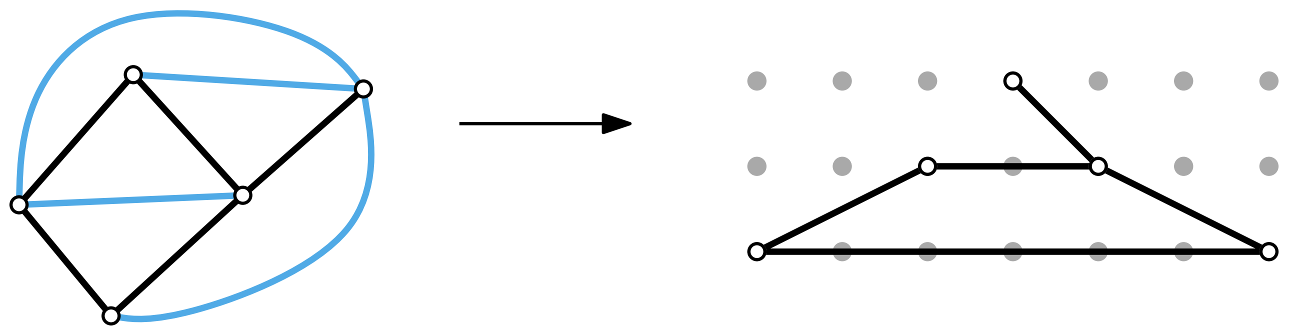
General Algorithm Idea

- sufficient to focus on drawing maximally planar (triangulated) graphs



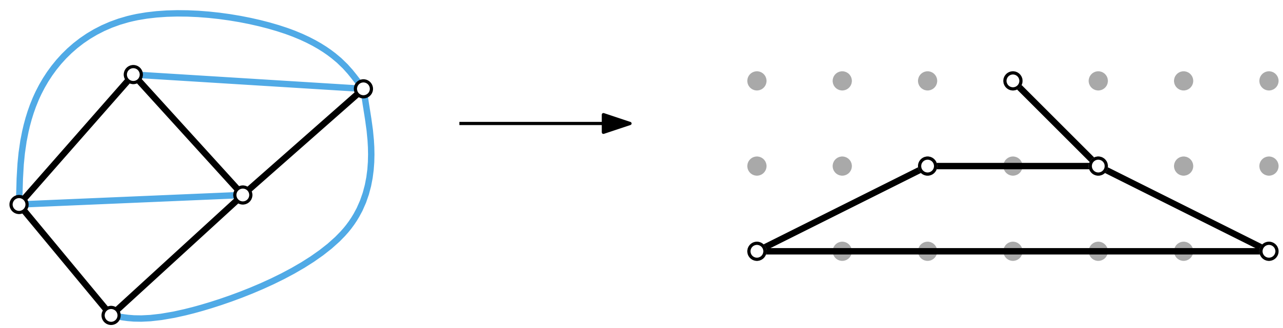
General Algorithm Idea

- sufficient to focus on drawing maximally planar (triangulated) graphs



General Algorithm Idea

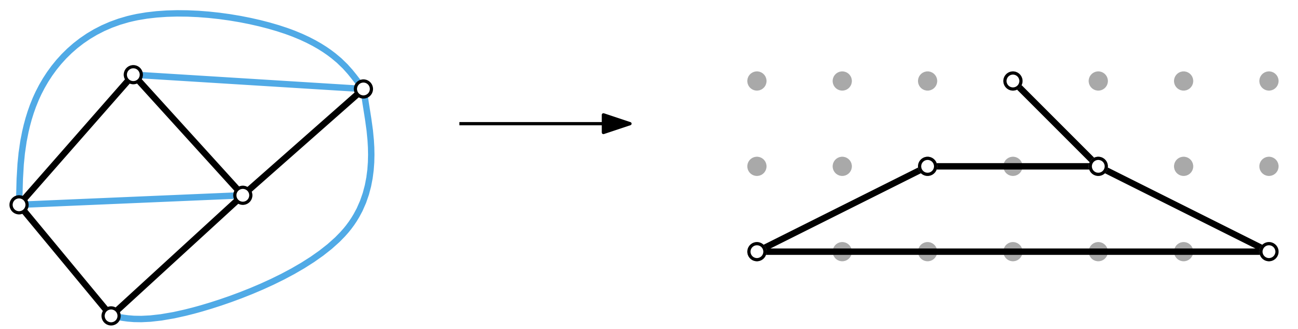
- sufficient to focus on drawing maximally planar (triangulated) graphs



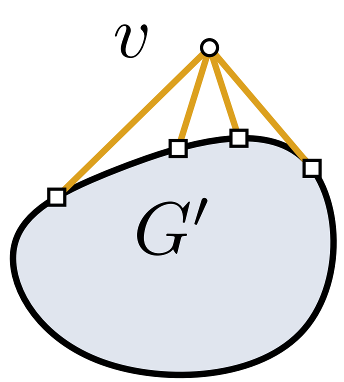
- create drawing incrementally

General Algorithm Idea

- sufficient to focus on drawing maximally planar (triangulated) graphs

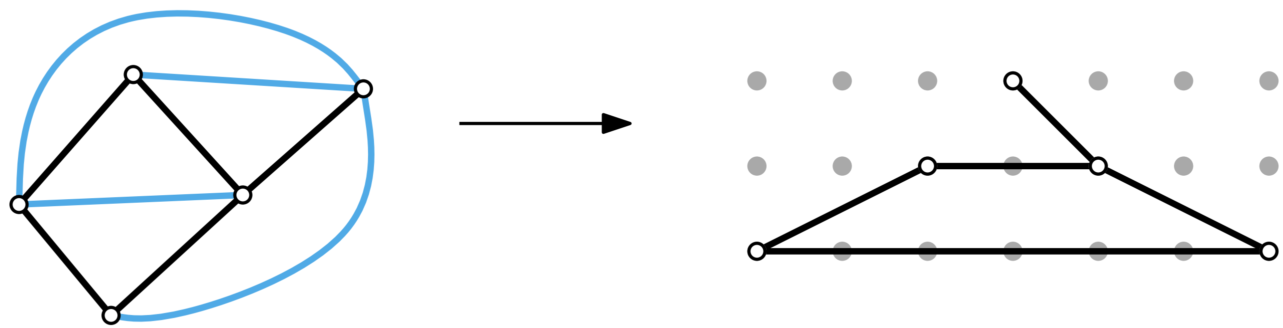


- create drawing incrementally

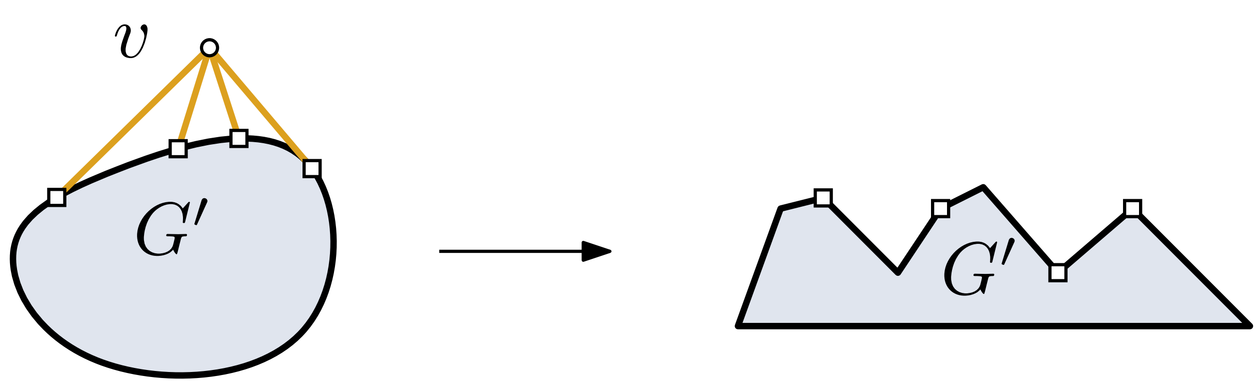


General Algorithm Idea

- sufficient to focus on drawing maximally planar (triangulated) graphs

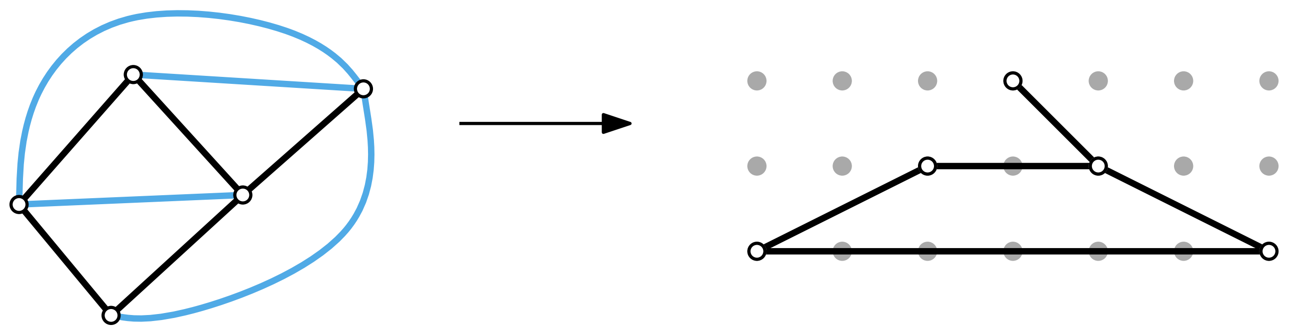


- create drawing incrementally

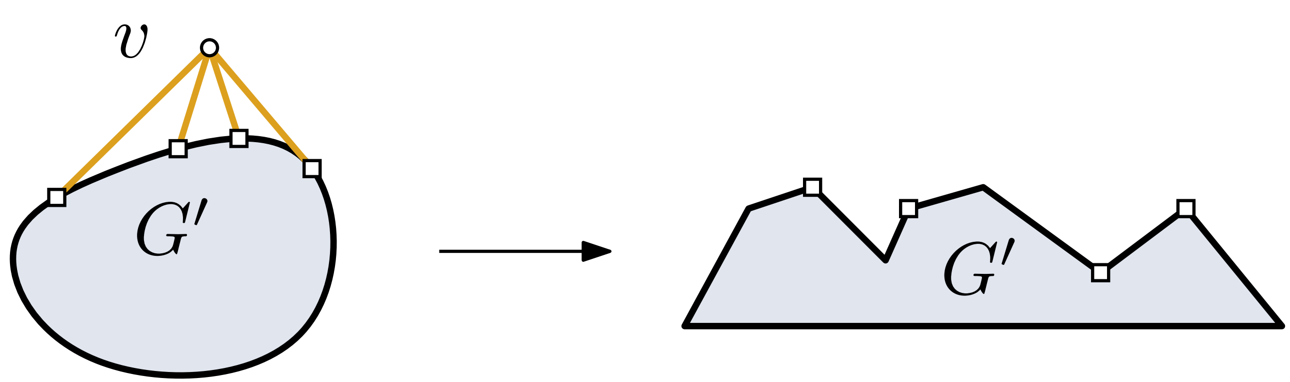


General Algorithm Idea

- sufficient to focus on drawing maximally planar (triangulated) graphs

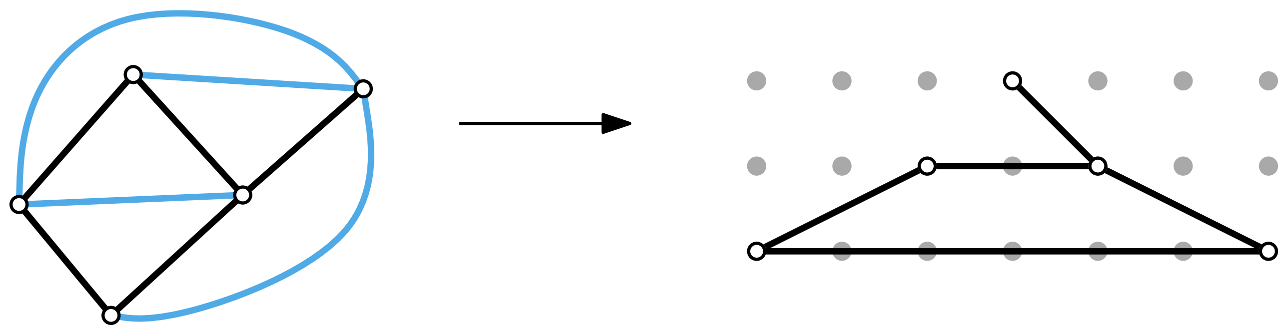


- create drawing incrementally

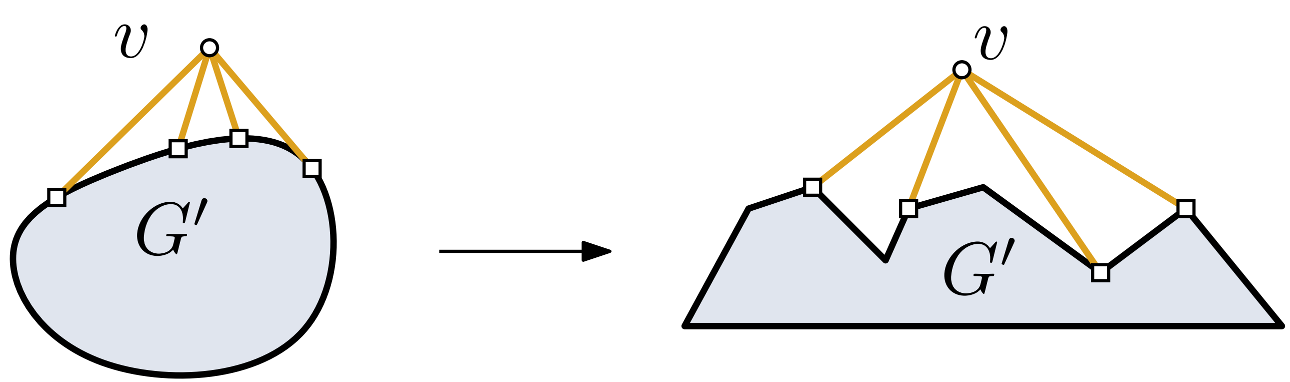


General Algorithm Idea

- sufficient to focus on drawing maximally planar (triangulated) graphs



- create drawing incrementally



Overview

Canonical ordering

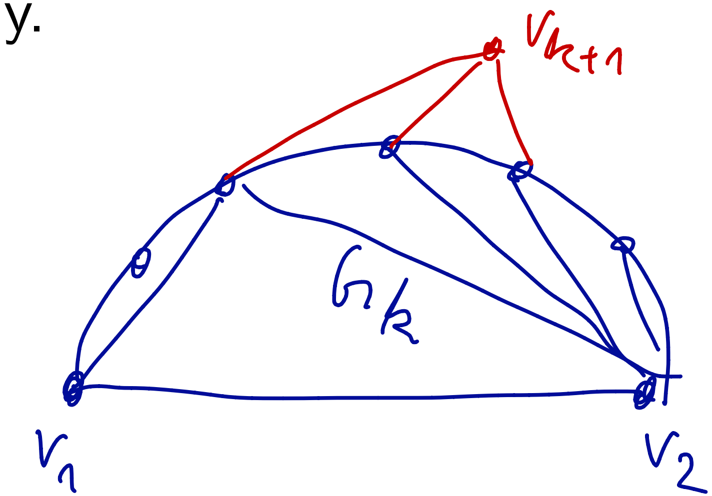
Shift algorithm

Implementation

Canonical Ordering

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

- $\{v_1, \ldots v_k\}$ induce 2-connected internally triangulated graph G_k,
- edge (v_1, v_2) belongs to the outer face of each G_k,
- for all $k < n$ vertex v_{k+1} lies in the external face of G_k, and all neighbors of v_{k+1} in G_k appear on the boundary C_k of G_k consecutively.

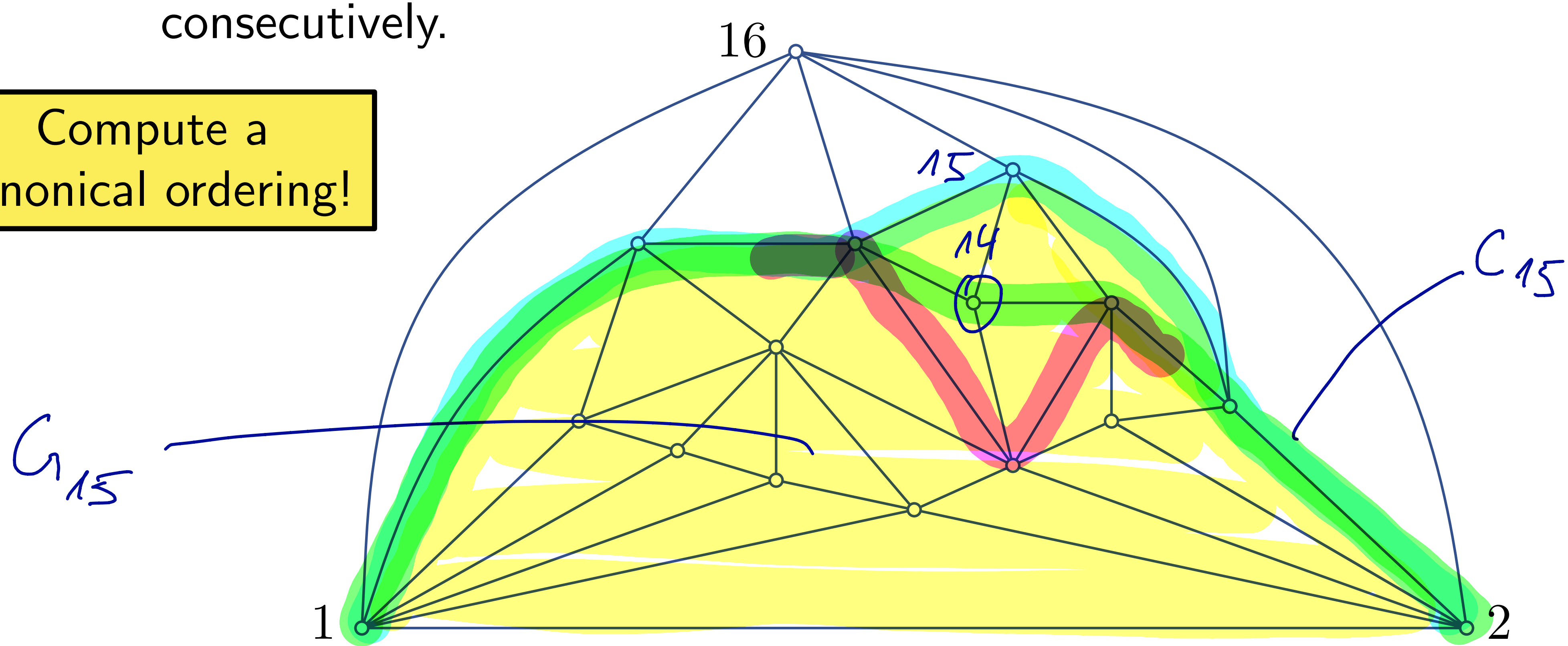


Canonical Ordering

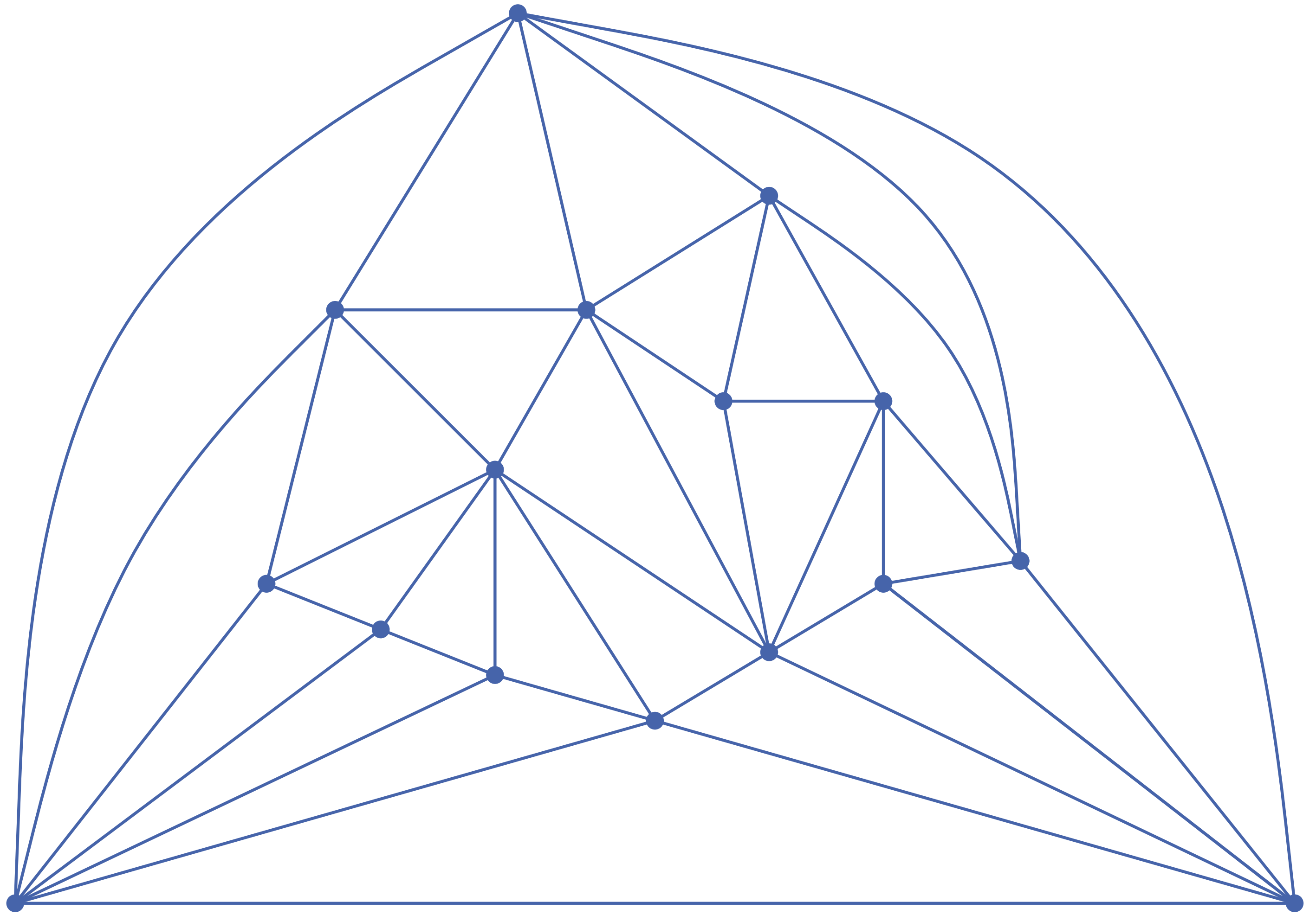
Def: Let $G = (V, E)$ be a **triangulated planar embedded graph** with $n \geq 3$ vertices. An ordering $\pi = (v_1, v_2, \ldots, v_n)$ is called a **canonical ordering**, if the following conditions hold for each k, $3 \leq k \leq n$

- $\{v_1, \ldots v_k\}$ induce 2-connected internally triangulated graph G_k,
- edge (v_1, v_2) belongs to the outer face of each G_k,
- for all $k < n$ vertex v_{k+1} lies in the external face of G_k, and all neighbors of v_{k+1} in G_k appear on the boundary C_k of G_k consecutively.

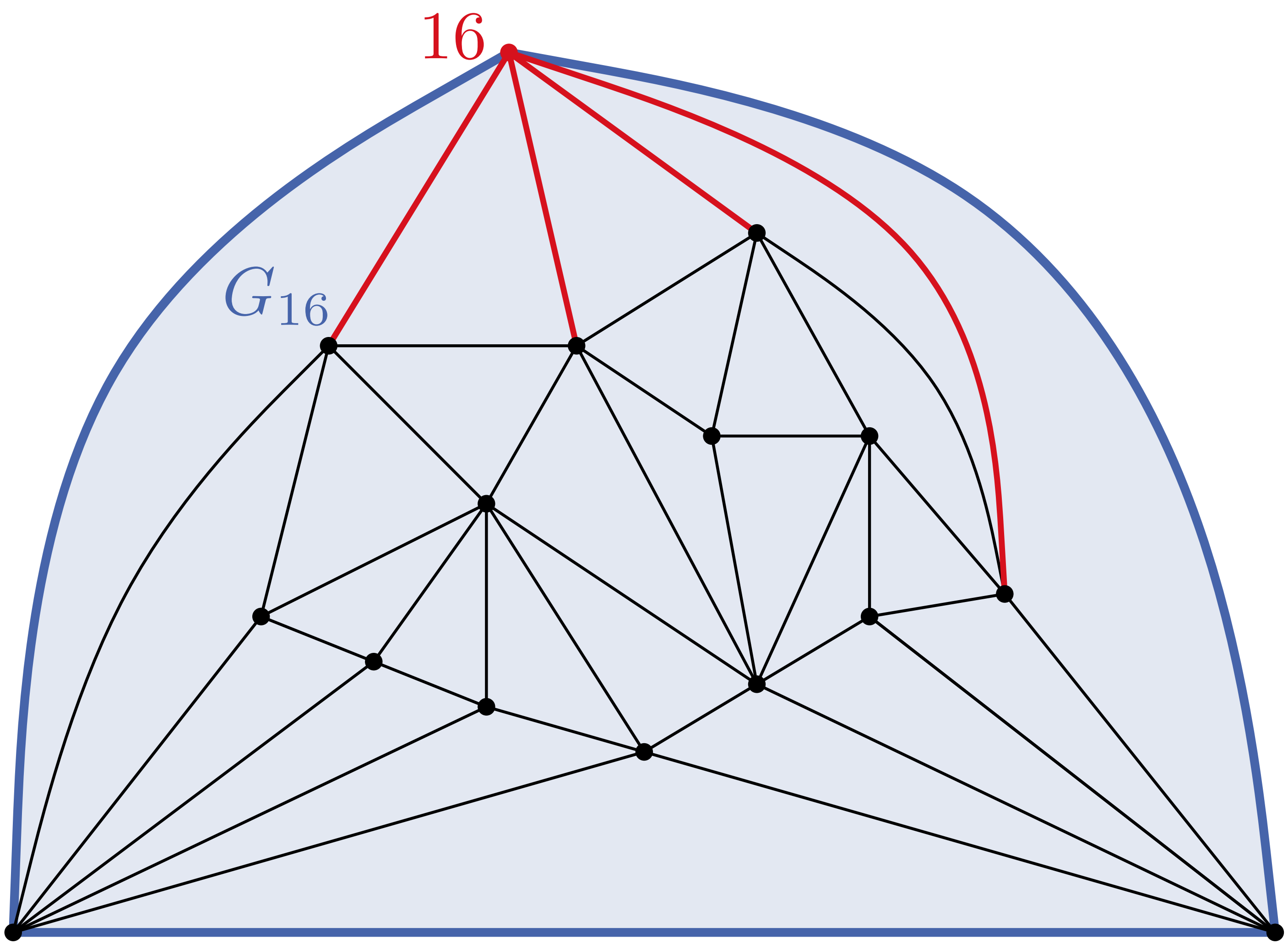
Compute a canonical ordering!



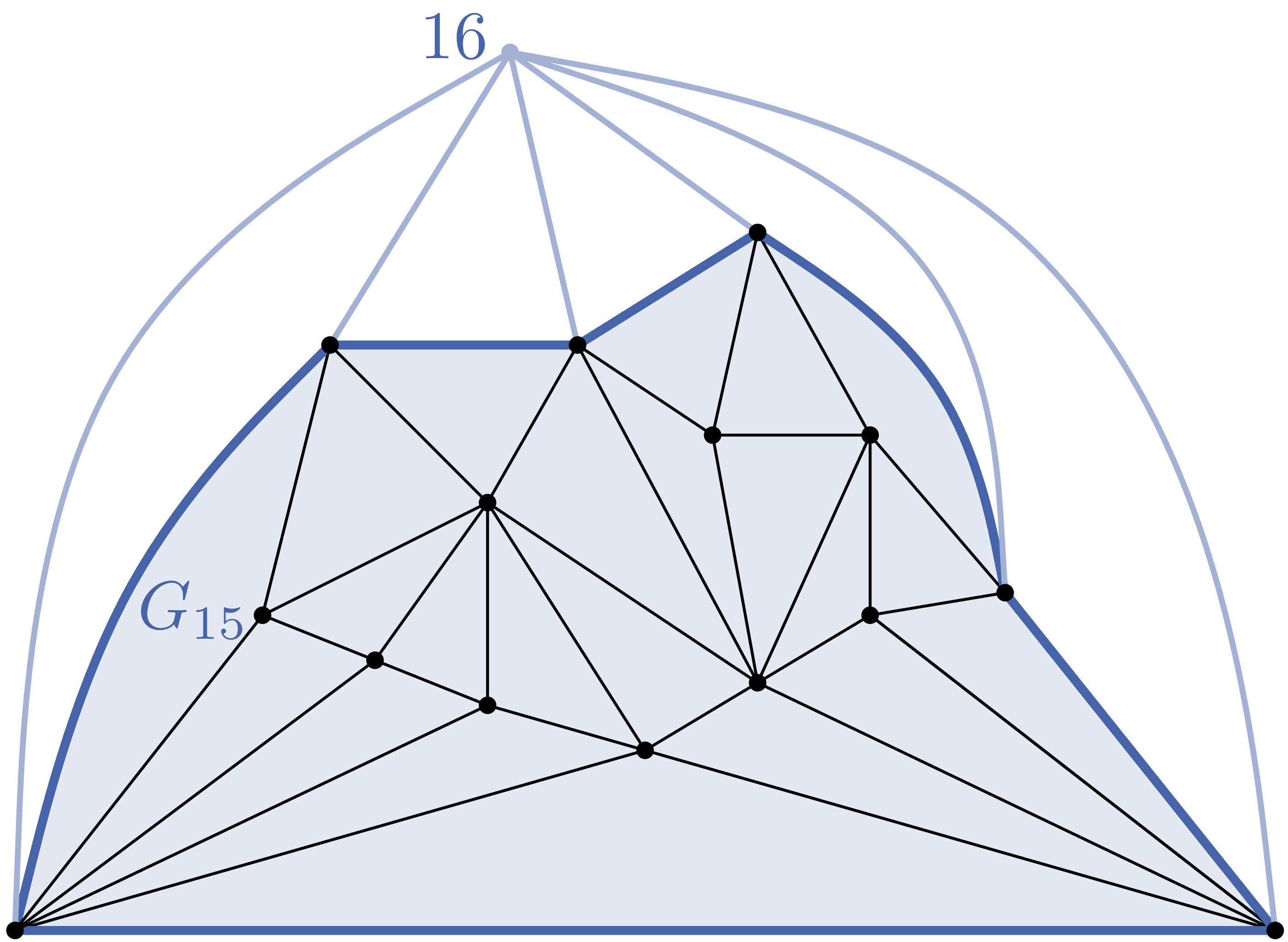
Example: Canonical Ordering



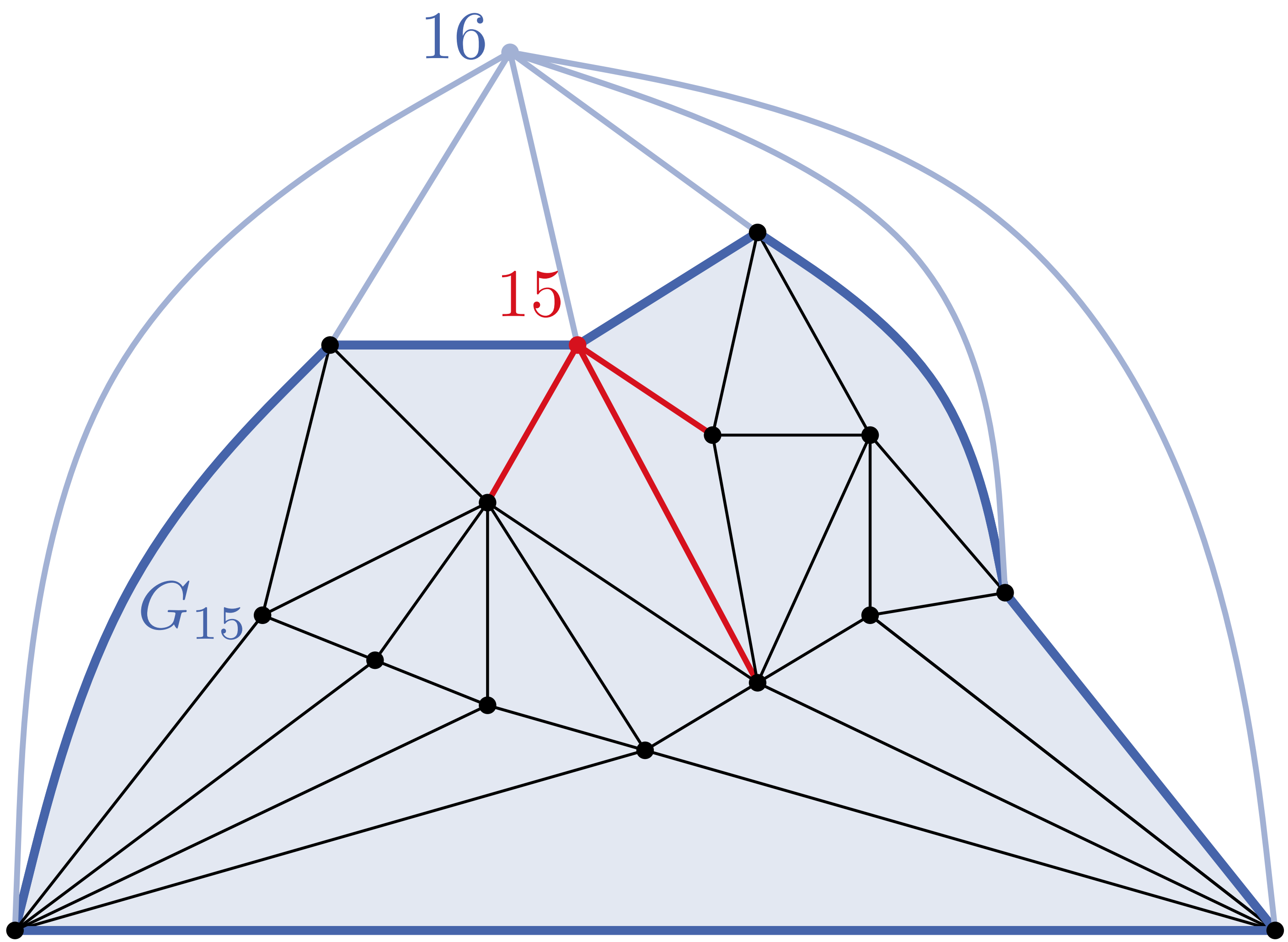
Example: Canonical Ordering



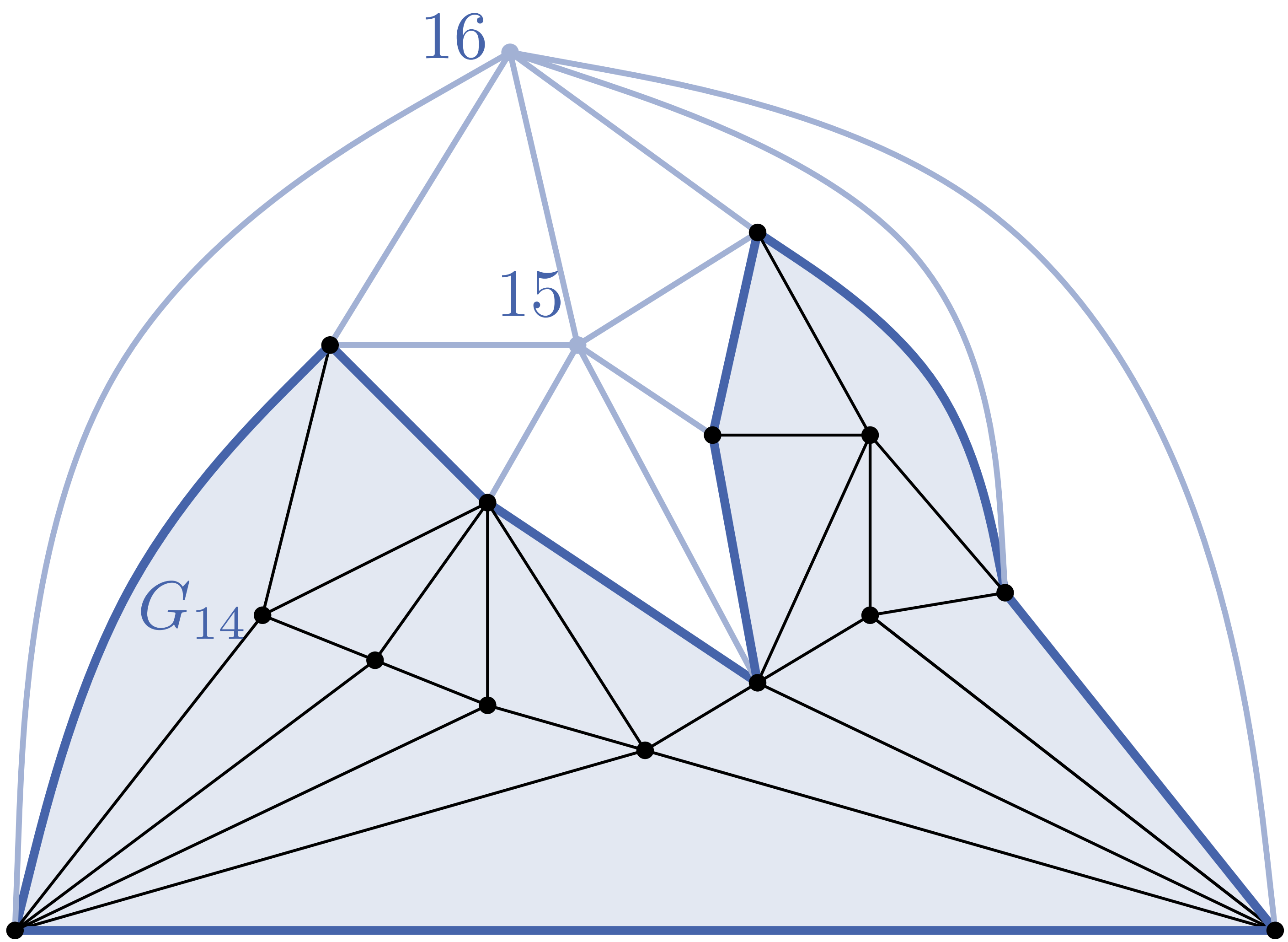
Example: Canonical Ordering



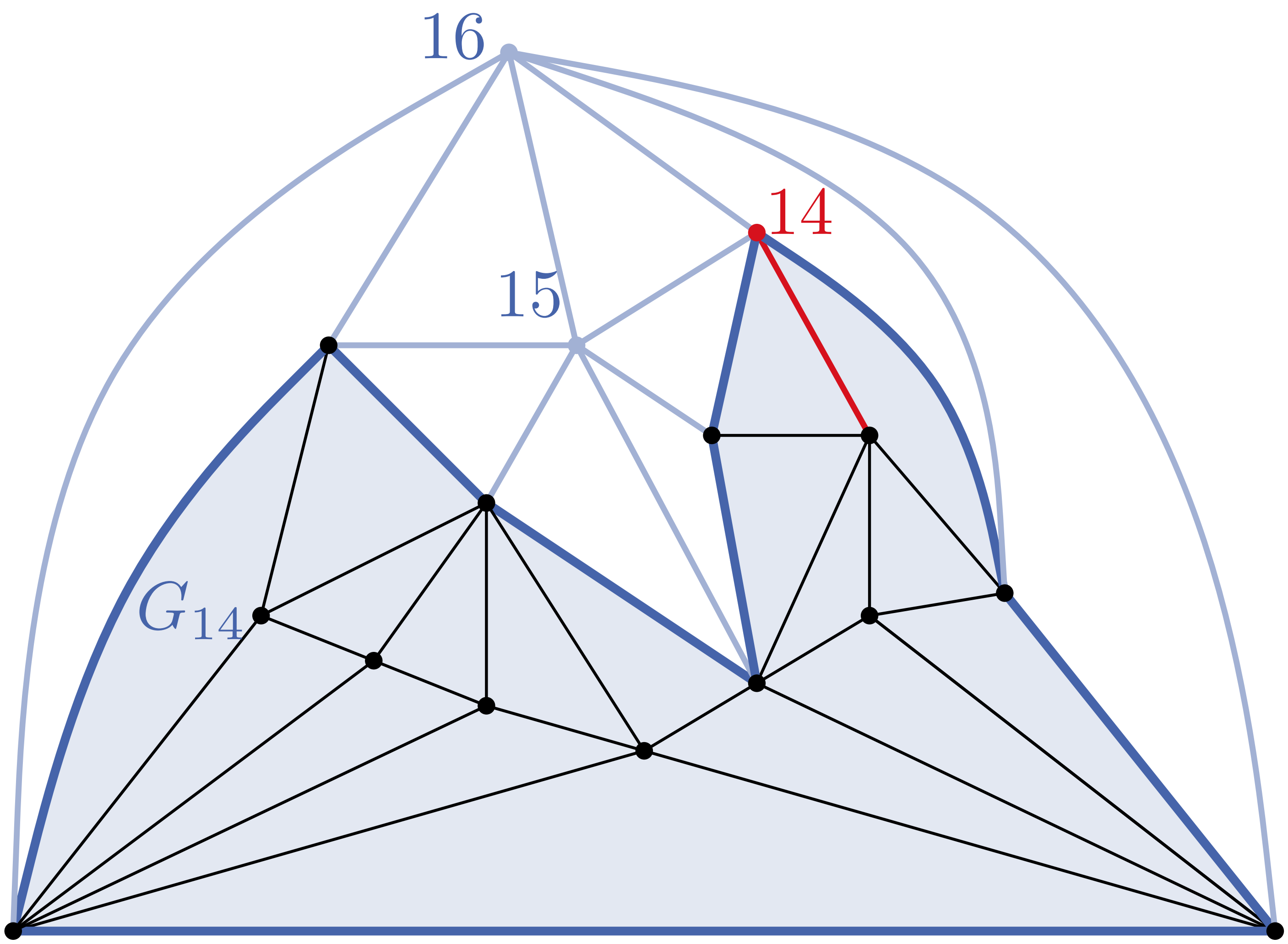
Example: Canonical Ordering



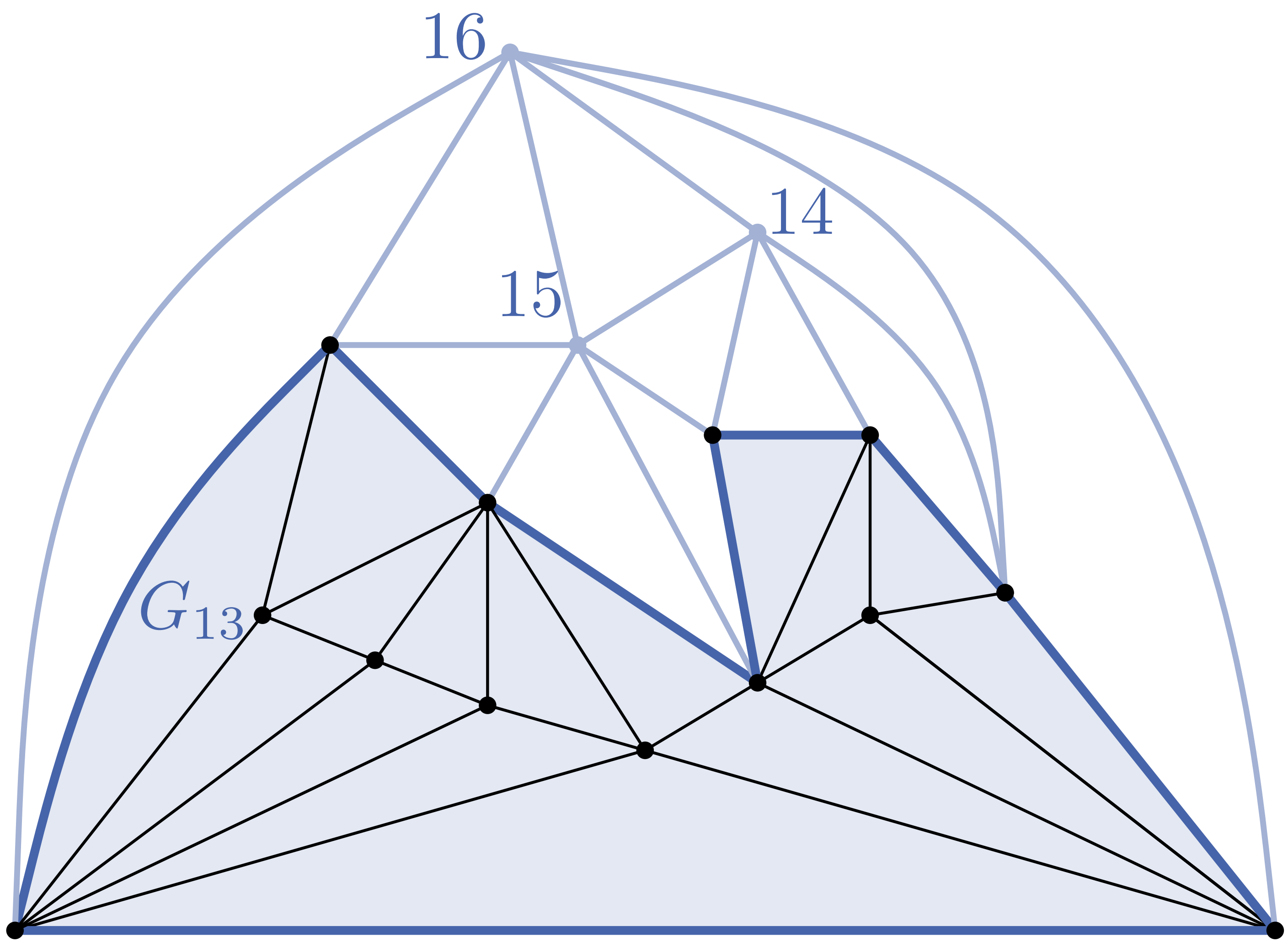
Example: Canonical Ordering



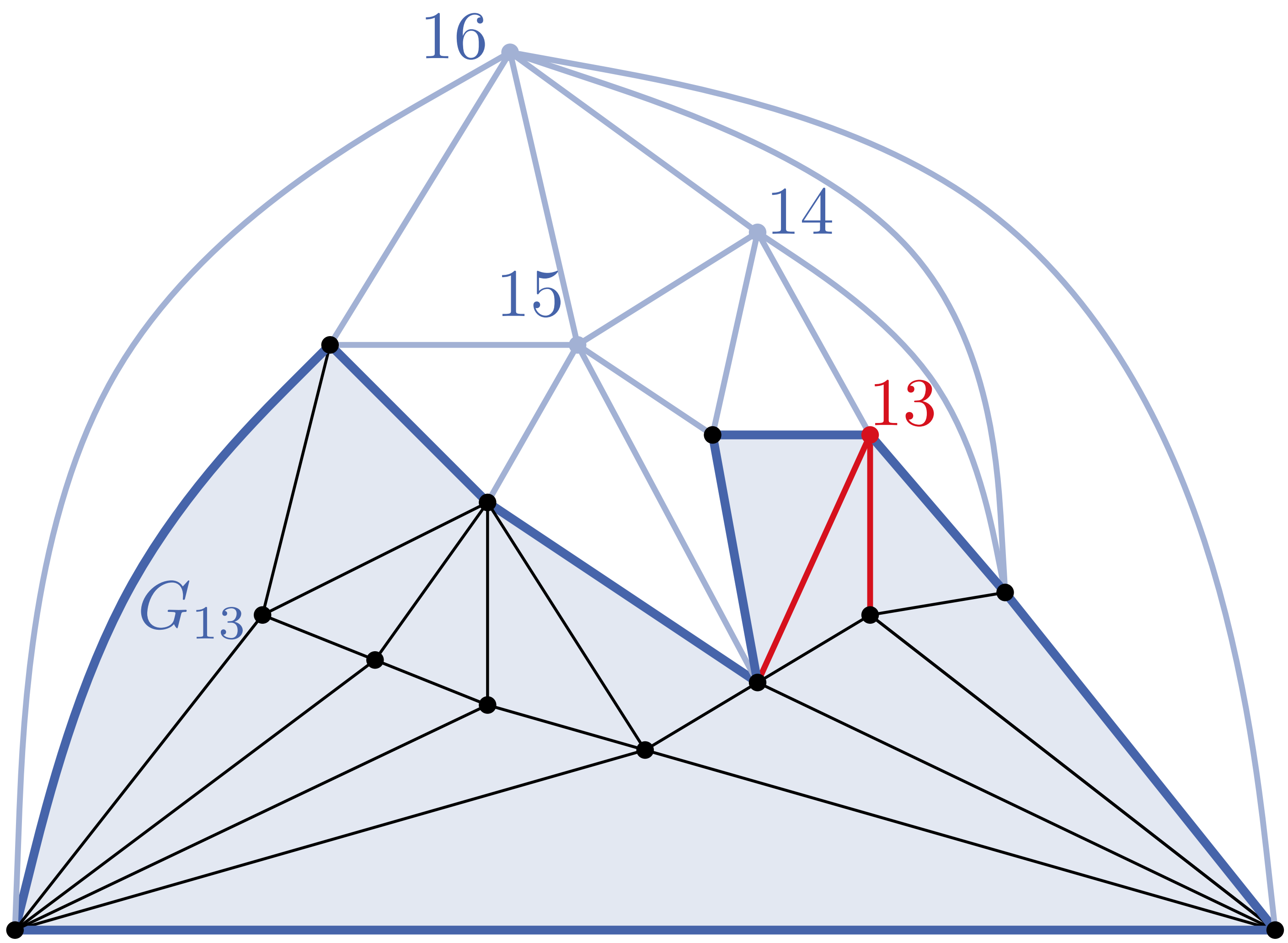
Example: Canonical Ordering



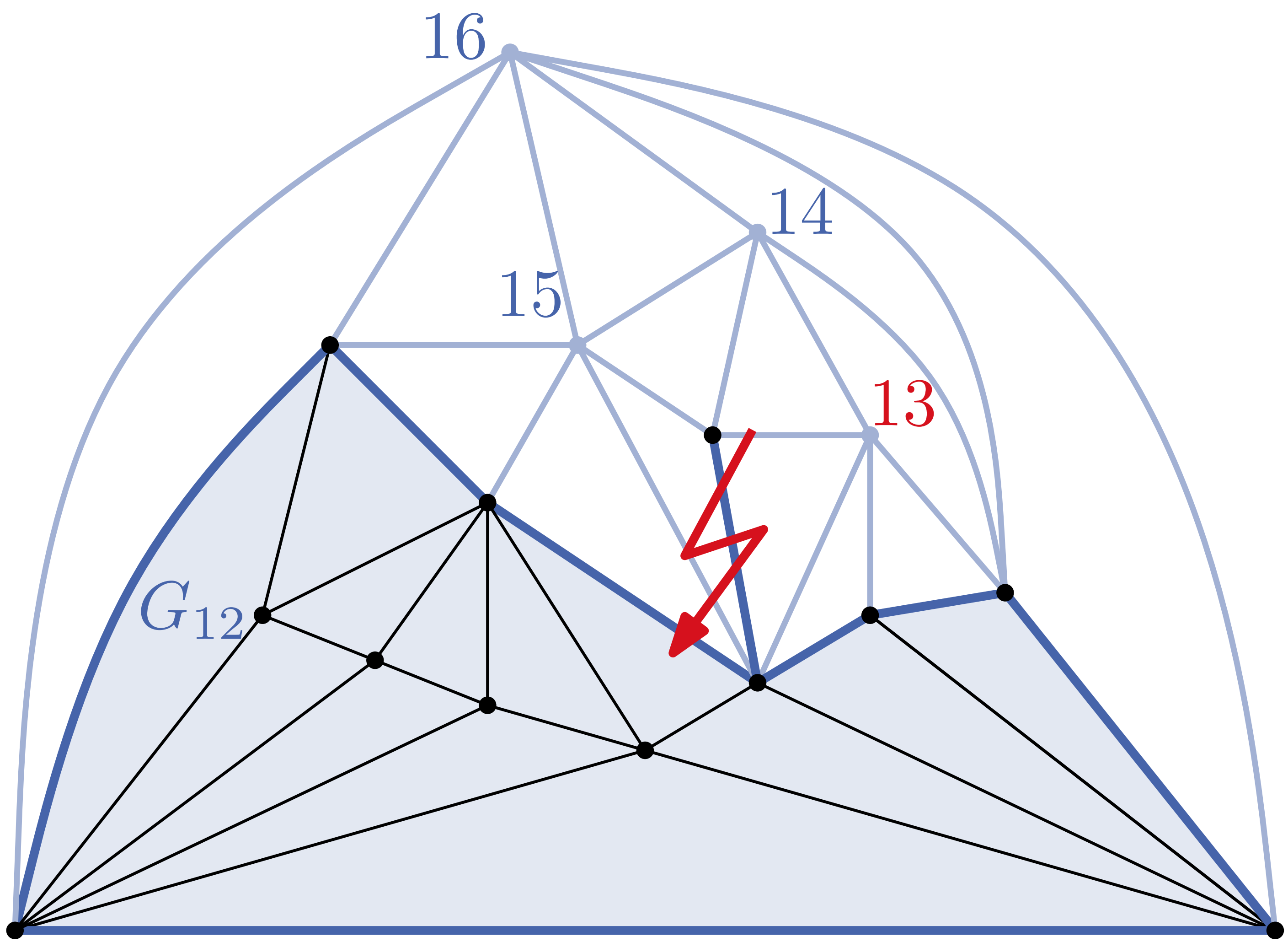
Example: Canonical Ordering



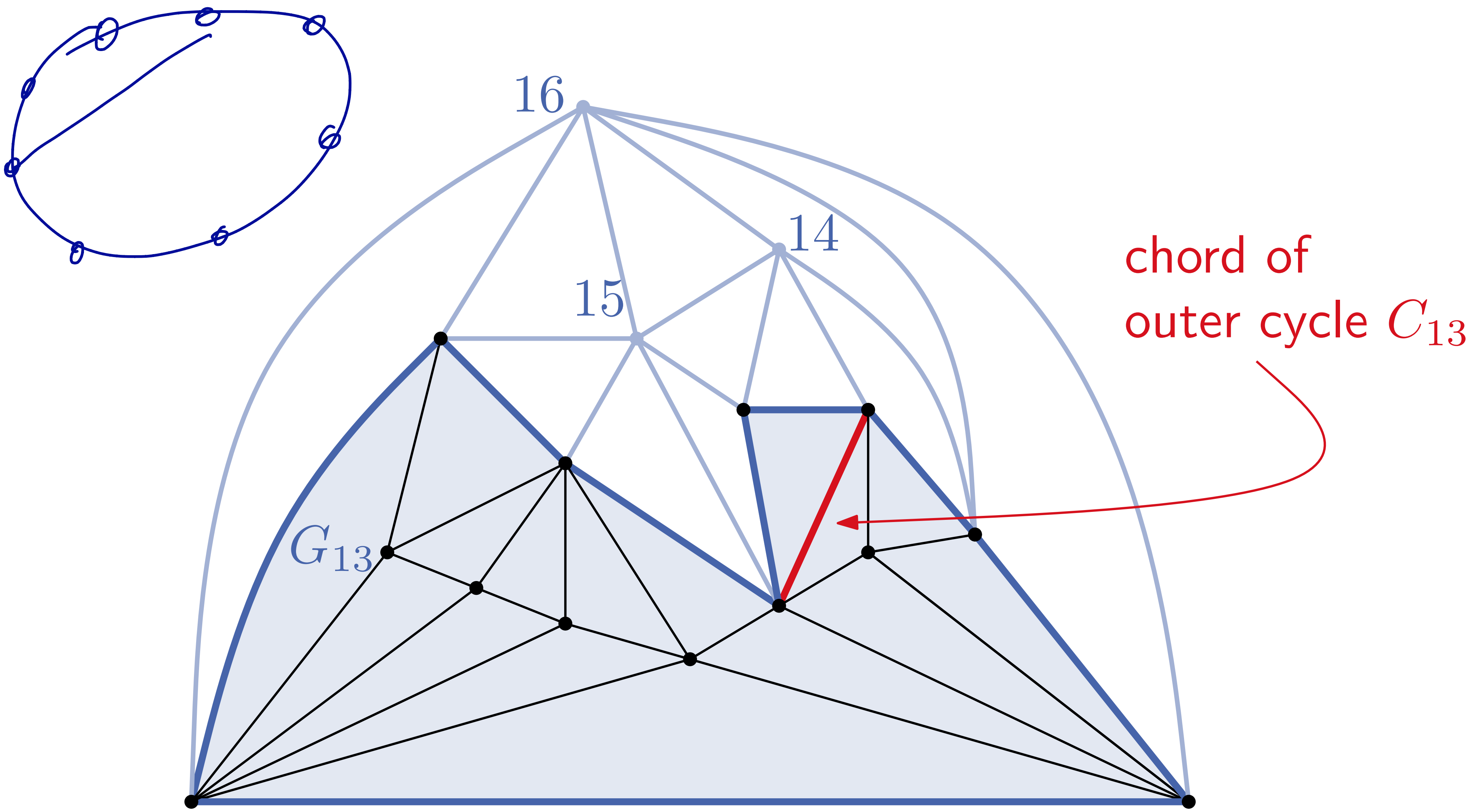
Example: Canonical Ordering



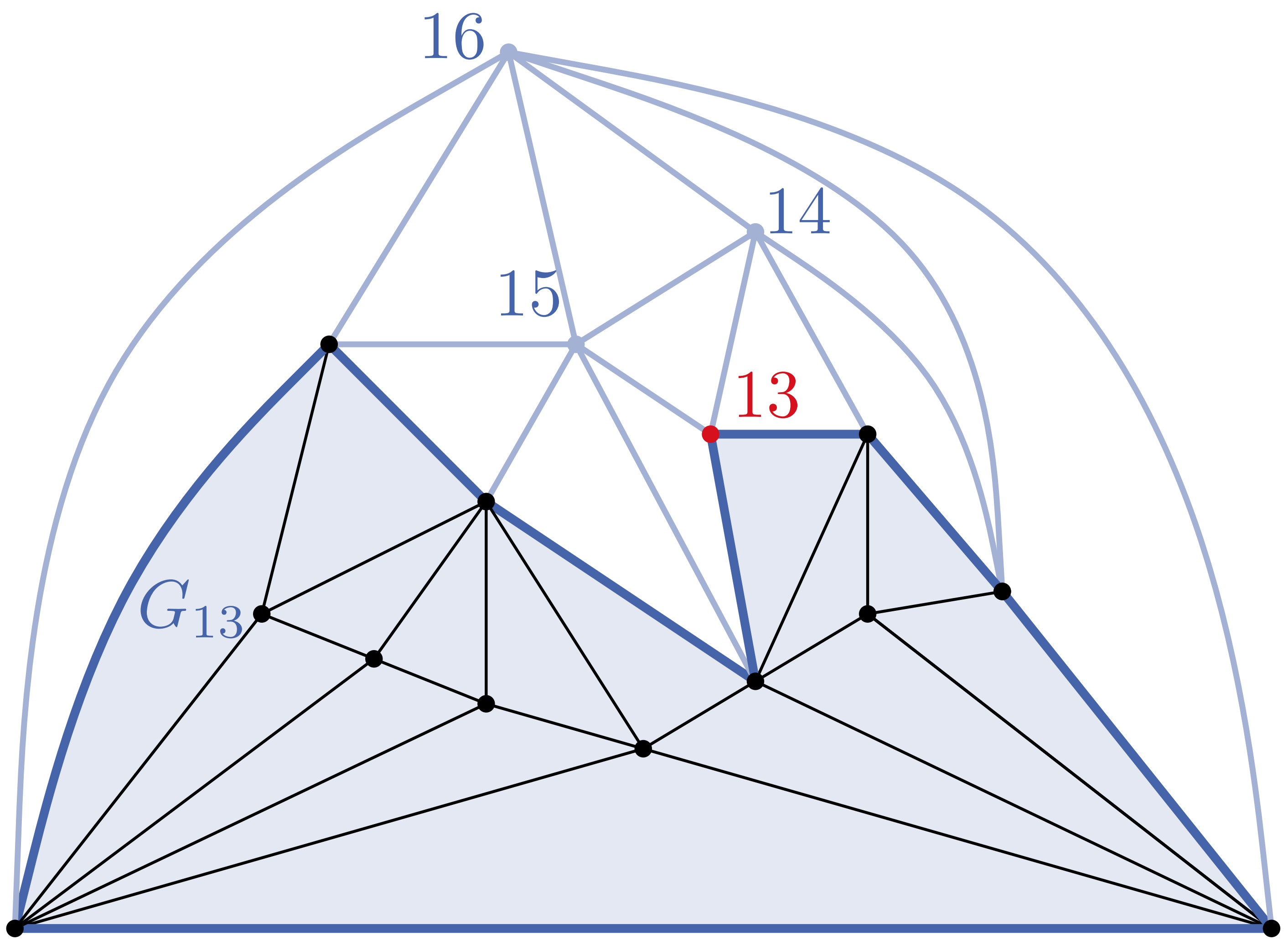
Example: Canonical Ordering



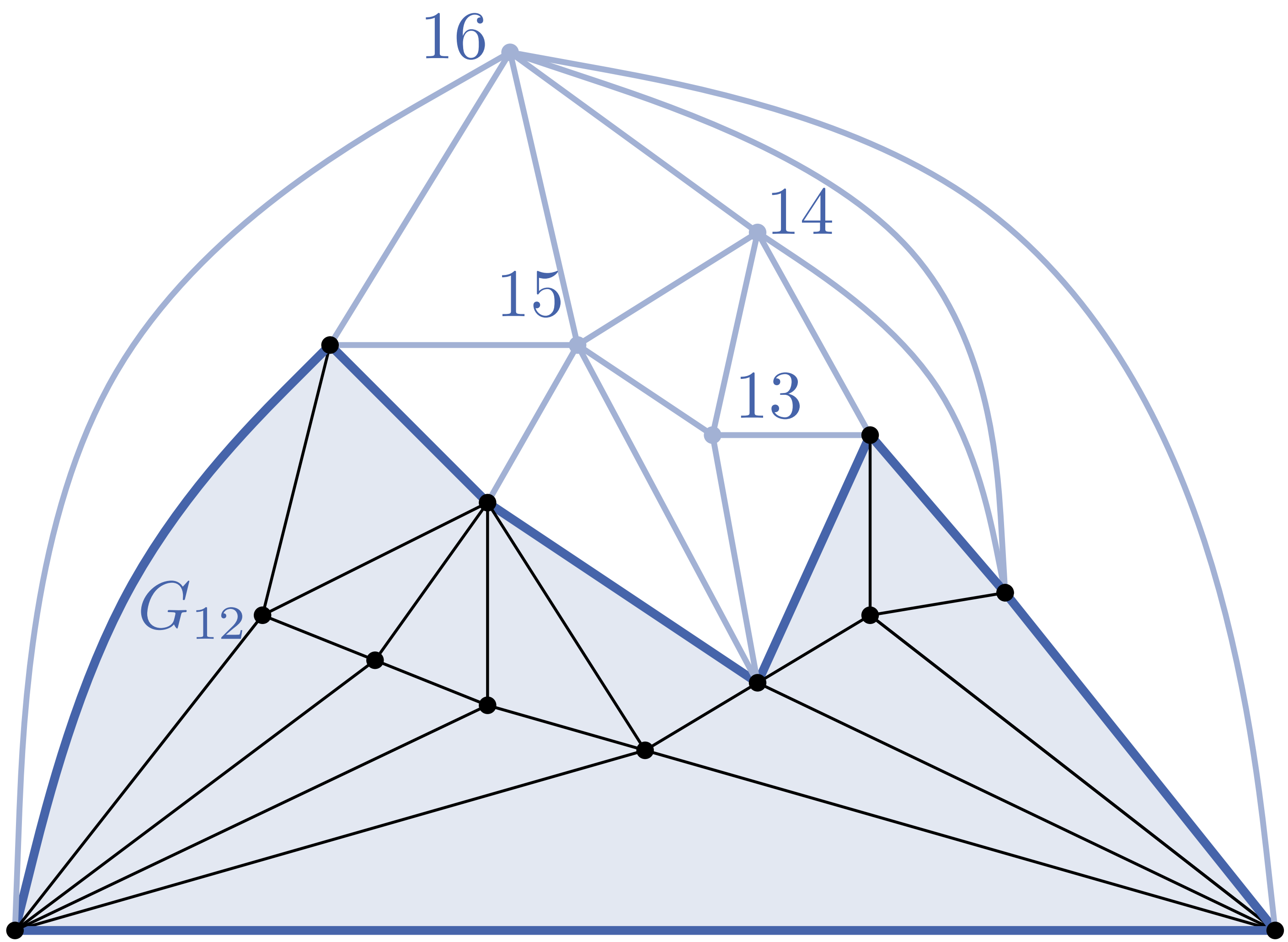
Example: Canonical Ordering



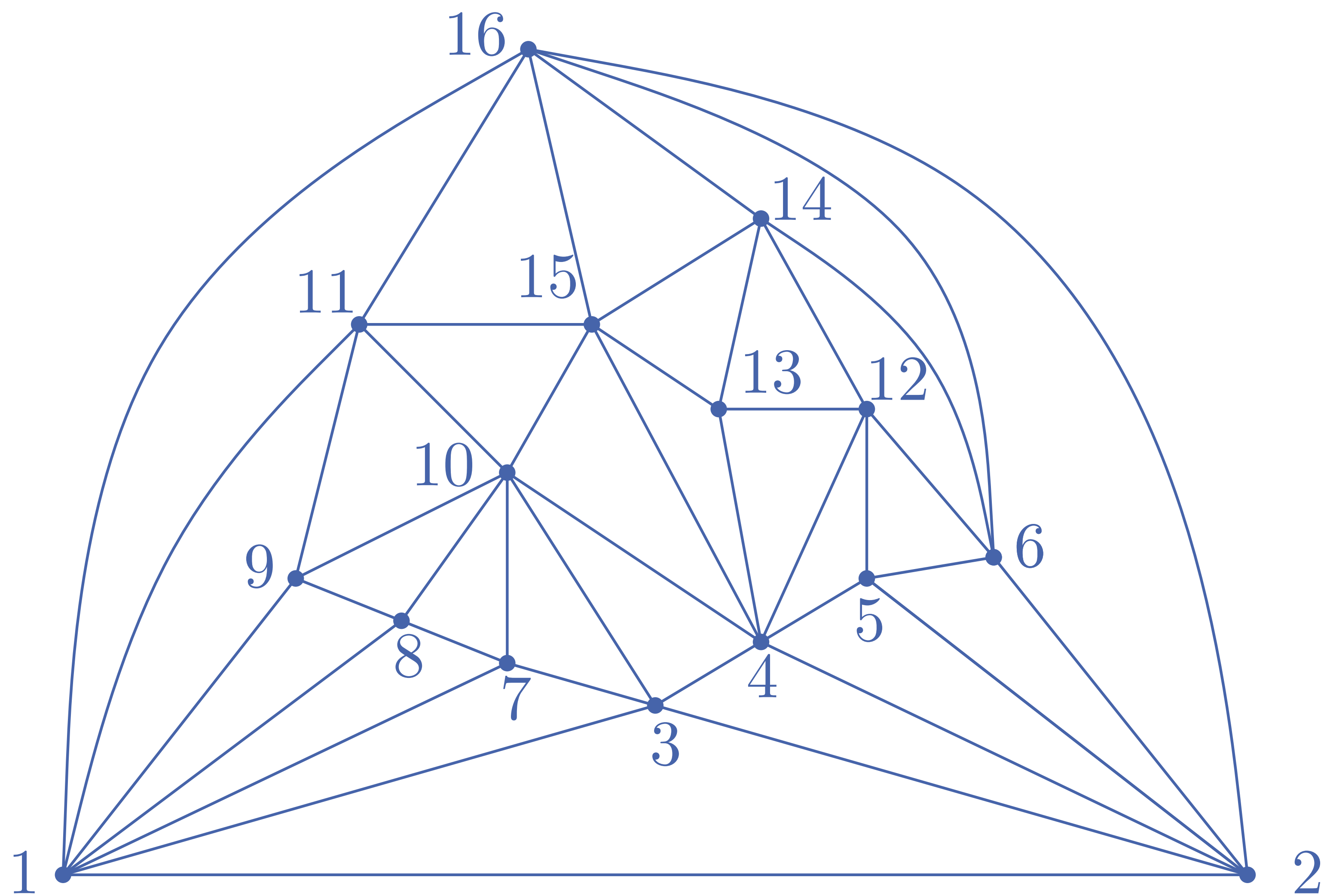
Example: Canonical Ordering



Example: Canonical Ordering



Example: Canonical Ordering



Existence of Canonical Orderings

Lemma. Every triangulated planar graph $G = (V, E)$ has a canonical ordering.

Existence of Canonical Orderings

Lemma. Every triangulated planar graph $G = (V, E)$ has a canonical ordering.

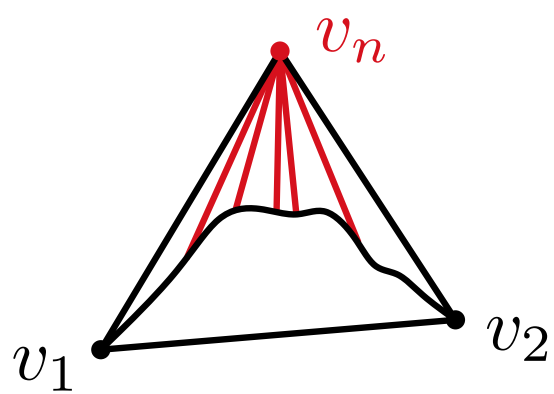
Proof: inverse induction on k

Existence of Canonical Orderings

Lemma. Every triangulated planar graph $G = (V, E)$ has a canonical ordering.

Proof: inverse induction on k

- $k = n$:



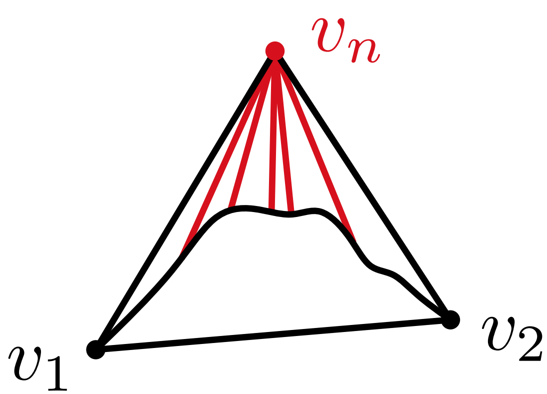
- induction hypothesis: vertices $v_n, v_{n-1}, \ldots, v_{k+1}$ were chosen such that they satisfy the three conditions.

Existence of Canonical Orderings

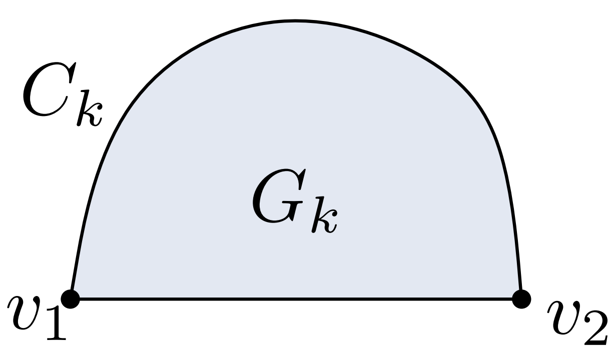
Lemma. Every triangulated planar graph $G = (V, E)$ has a canonical ordering.

Proof: inverse induction on k

- $k = n$:



- induction hypothesis: vertices $v_n, v_{n-1}, \ldots, v_{k+1}$ were chosen such that they satisfy the three conditions.
- consider G_k and find suitable vertex v_k on C_k



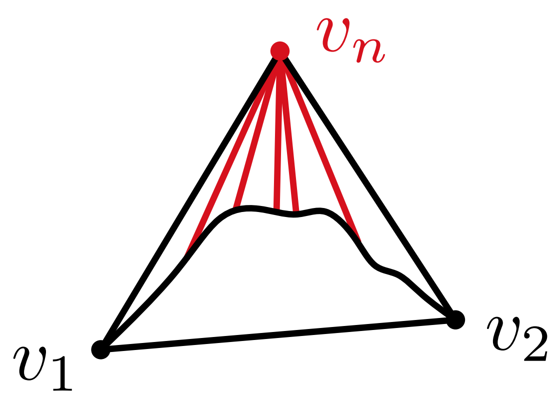
- G_k is 2-connected and internally triangulated

Existence of Canonical Orderings

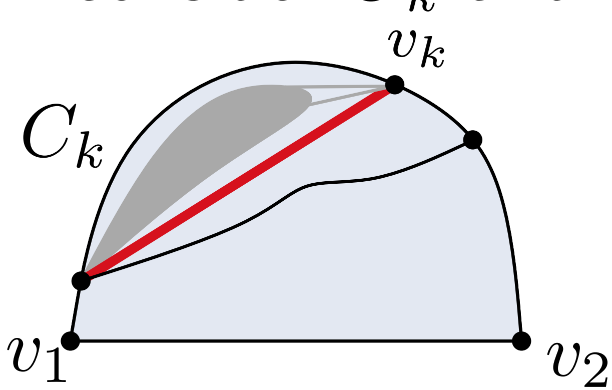
Lemma. Every triangulated planar graph $G = (V, E)$ has a canonical ordering.

Proof: inverse induction on k

- $k = n$:



- induction hypothesis: vertices $v_n, v_{n-1}, \ldots, v_{k+1}$ were chosen such that they satisfy the three conditions.
- consider G_k and find suitable vertex v_k on C_k



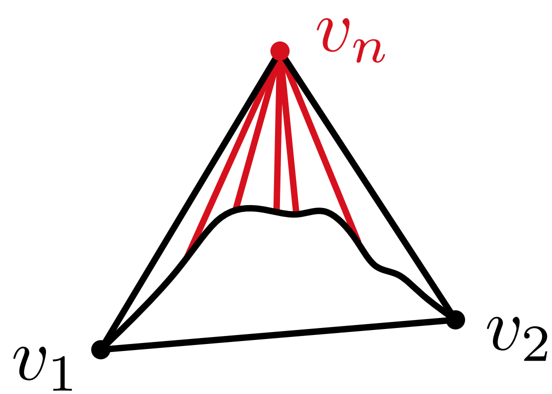
- G_k is 2-connected and internally triangulated
- v_k should not be incident to chord of C_k

Existence of Canonical Orderings

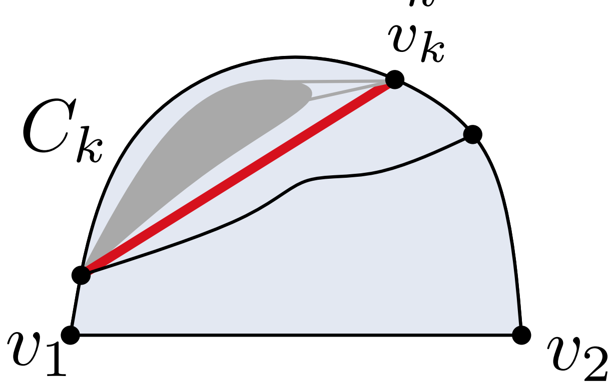
Lemma. Every triangulated planar graph $G = (V, E)$ has a canonical ordering.

Proof: inverse induction on k

- $k = n$:



- induction hypothesis: vertices $v_n, v_{n-1}, \ldots, v_{k+1}$ were chosen such that they satisfy the three conditions.
- consider G_k and find suitable vertex v_k on C_k



- G_k is 2-connected and internally triangulated
- v_k should not be incident to chord of C_k

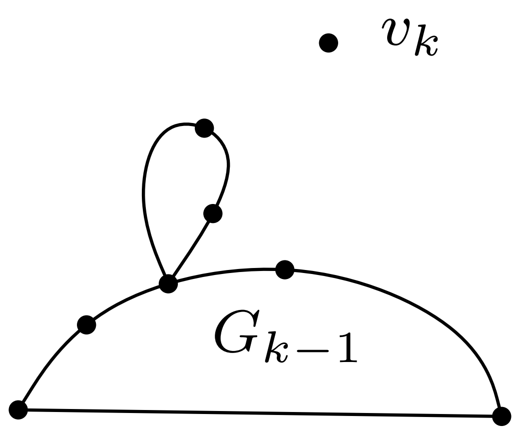
Is this sufficient?

Existence of Canonical Orderings

We show: v_k not incident to a chord $\Rightarrow G_{k-1}$ is 2-connected.

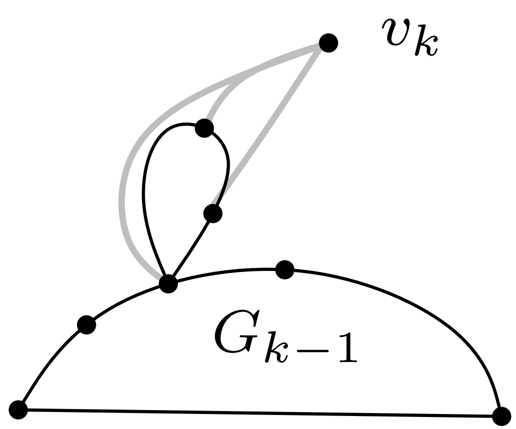
Existence of Canonical Orderings

We show: v_k not incident to a chord $\Rightarrow G_{k-1}$ is 2-connected.



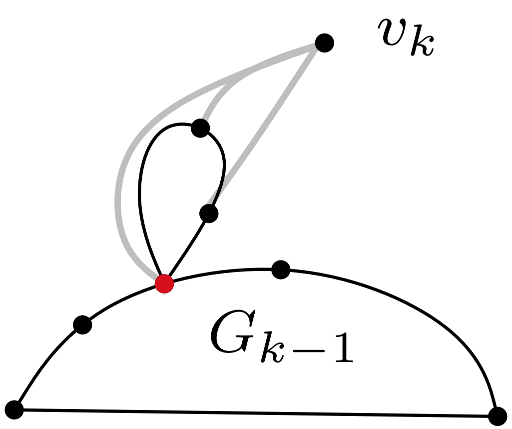
Existence of Canonical Orderings

We show: v_k not incident to a chord $\Rightarrow$ G_{k-1} is 2-connected.



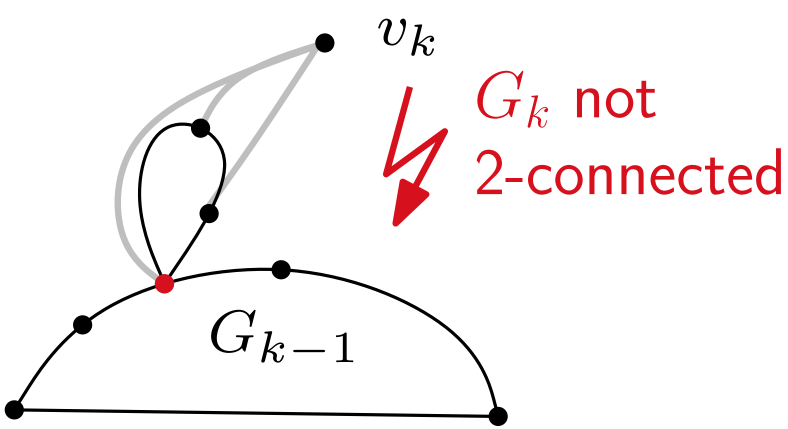
Existence of Canonical Orderings

We show: v_k not incident to a chord $\Rightarrow$ G_{k-1} is 2-connected.



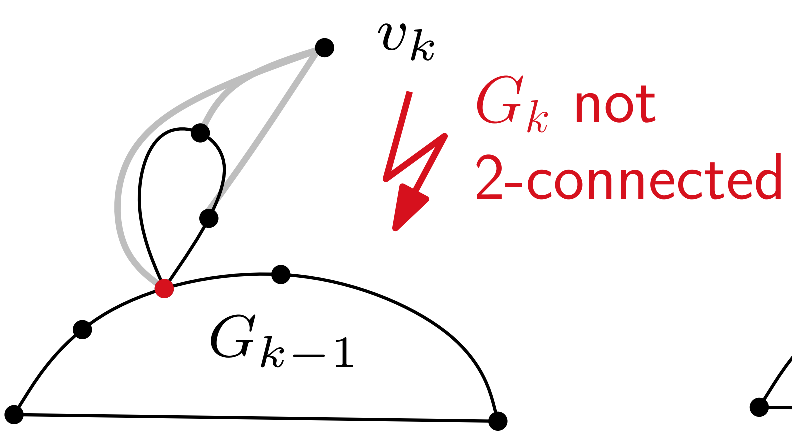
Existence of Canonical Orderings

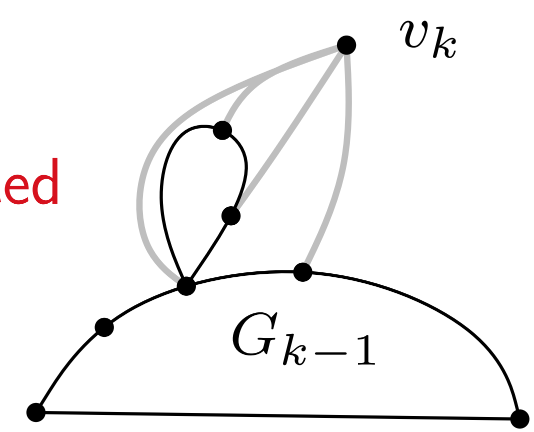
We show: v_k not incident to a chord $\Rightarrow G_{k-1}$ is 2-connected.



Existence of Canonical Orderings

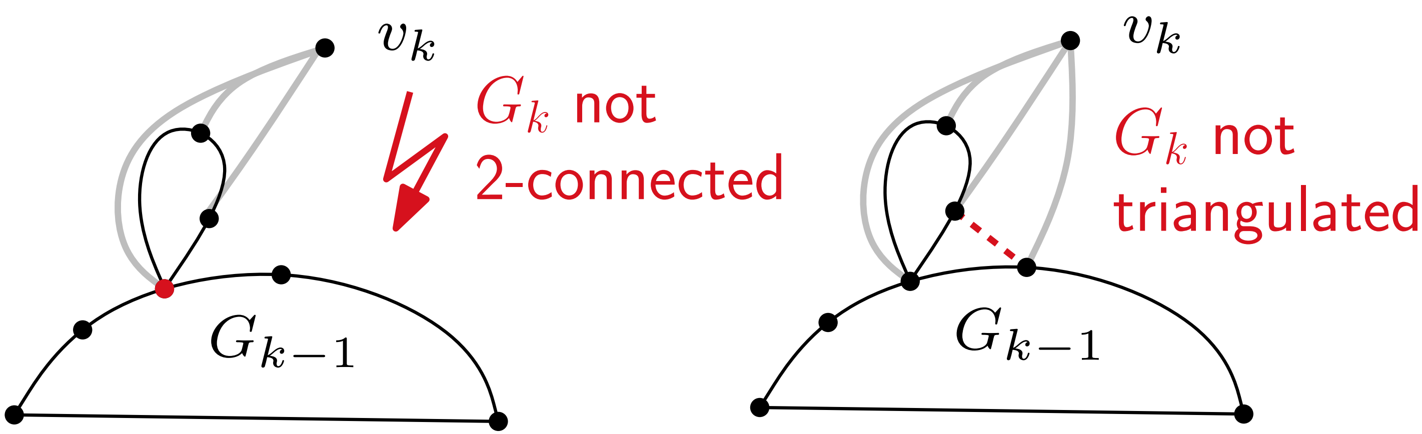
We show: v_k not incident to a chord $\Rightarrow G_{k-1}$ is 2-connected.





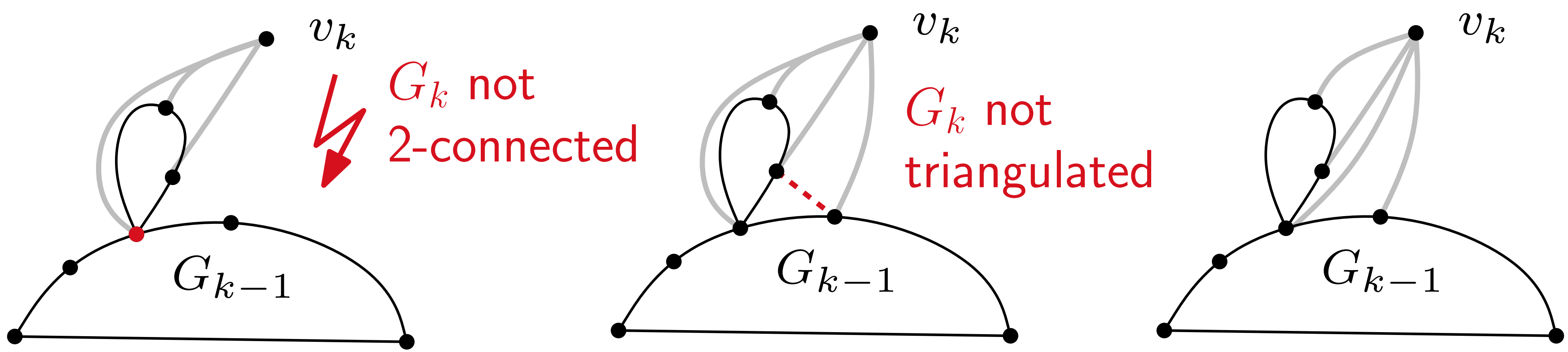
Existence of Canonical Orderings

We show: v_k not incident to a chord $\Rightarrow G_{k-1}$ is 2-connected.



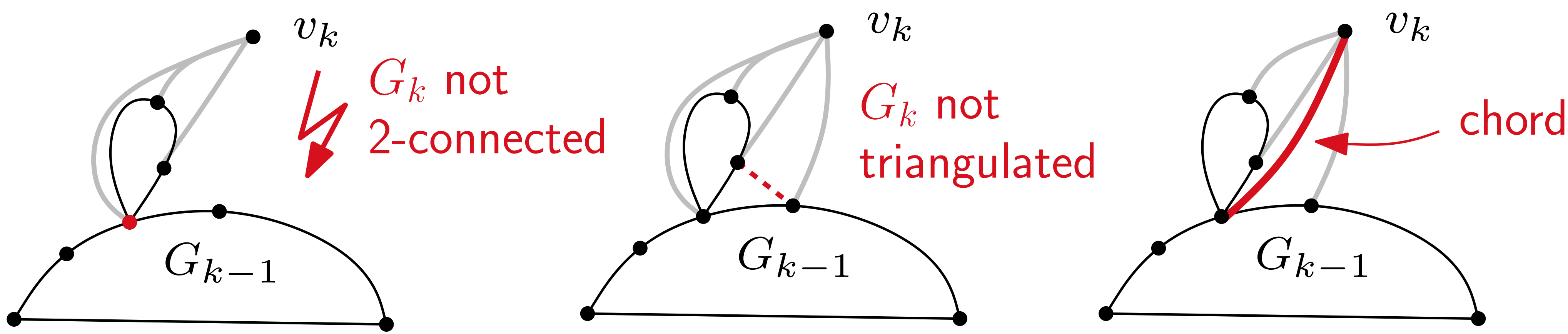
Existence of Canonical Orderings

We show: v_k not incident to a chord $\Rightarrow G_{k-1}$ is 2-connected.



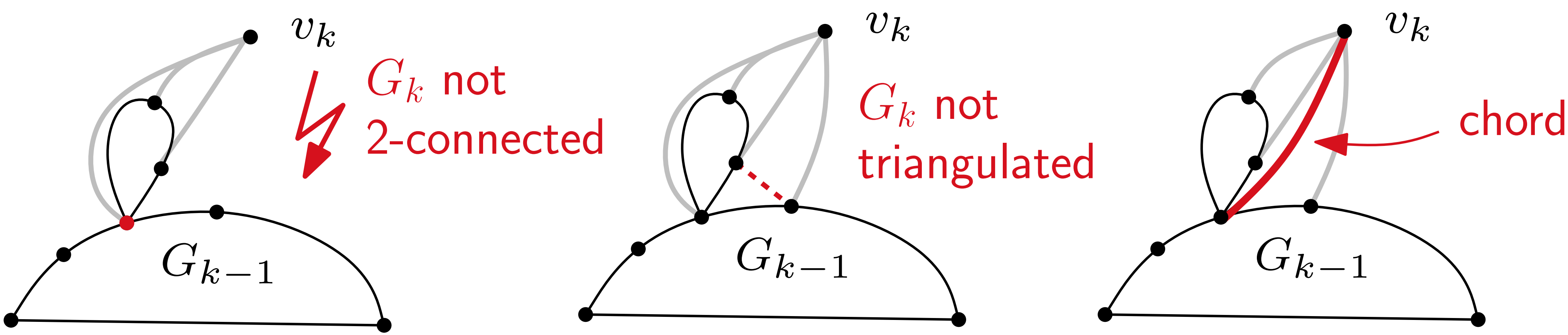
Existence of Canonical Orderings

We show: v_k not incident to a chord $\Rightarrow G_{k-1}$ is 2-connected.

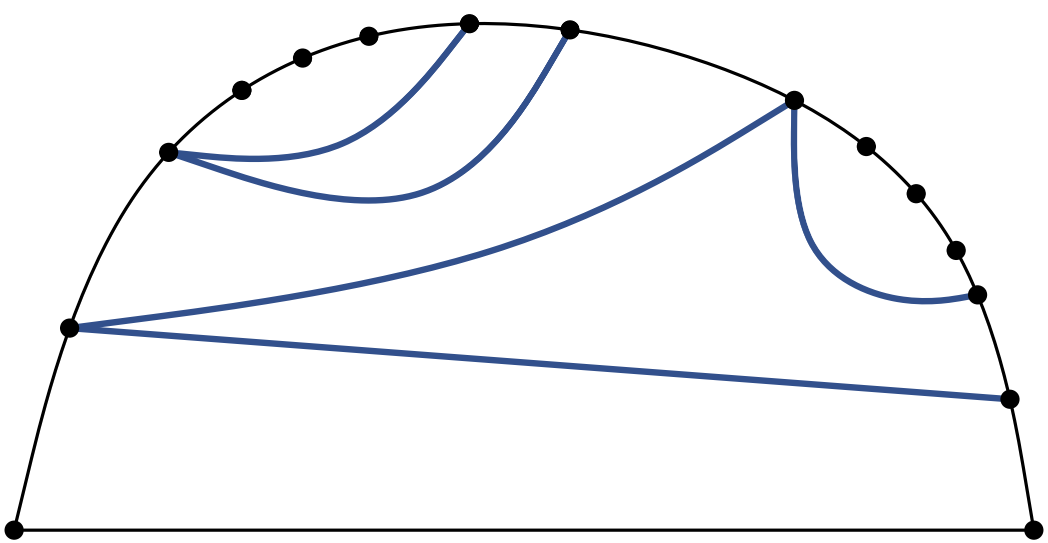


Existence of Canonical Orderings

We show: v_k not incident to a chord $\Rightarrow G_{k-1}$ is 2-connected.



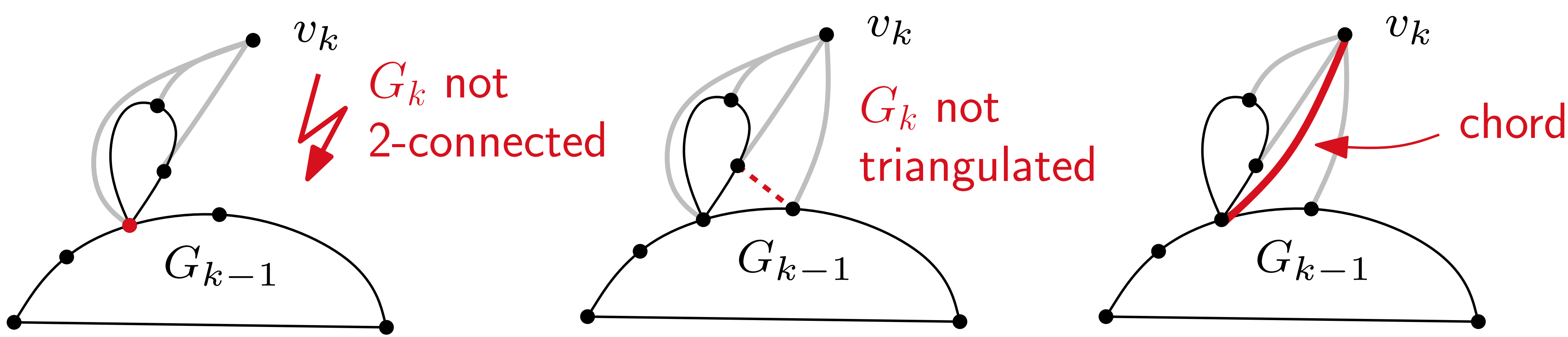
Can we always find a vertex not incident to a chord?



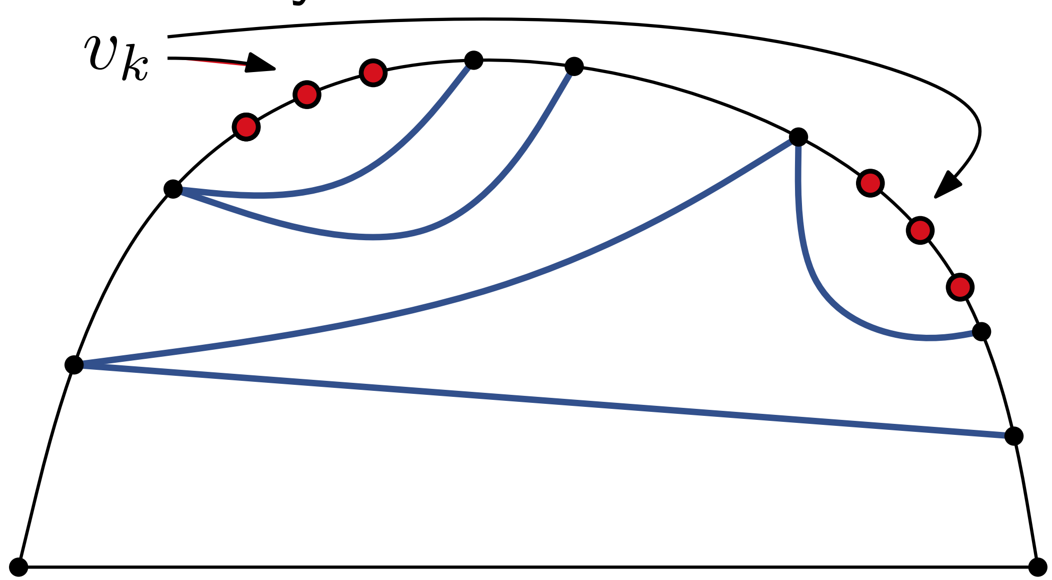
- chords form proper nesting

Existence of Canonical Orderings

We show: v_k not incident to a chord $\Rightarrow G_{k-1}$ is 2-connected.



Can we always find a vertex not incident to a chord?



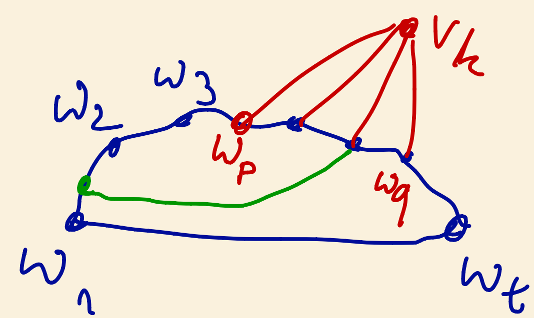
- chords form proper nesting
- innermost chords span at least one such vertex

□

Computing a Canonical Ordering

`CanonicalOrdering`(plane graph $G = (V, E)$)

forall $v \in V$ **do**
  chords(v) $\leftarrow 0$; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false;
out(v_1), out(v_2), out(v_n) $\leftarrow$ true
for $k = n$ **to** 3 **do**
  choose $v \neq v_1, v_2$ with mark(v) = false, out(v) = true, chords(v) = 0
  $v_k \leftarrow v$; mark(v) $\leftarrow$ true
  $(w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2) \leftarrow C_{k-1}$
  $(w_p, \ldots, w_q) \leftarrow$ unmarked neighbors of v_k
  out(w_i) $\leftarrow$ true for all $p < i < q$
  update chords($\cdot$) for these w_i and their neighbors



- chord(v) = number of chords incident to v
- mark(v) = true iff vertex v was numbered
- out(v) = true iff v is currently in the external face

Computing a Canonical Ordering

`CanonicalOrdering`(plane graph $G = (V, E)$)

forall $v \in V$ **do**
  chords(v) $\leftarrow 0$; out(v) $\leftarrow$ false; mark(v) $\leftarrow$ false;
out(v_1), out(v_2), out(v_n) $\leftarrow$ true
for $k = n$ **to** 3 **do**
  choose $v \neq v_1, v_2$ with mark(v) = false, out(v) = true, chords(v) = 0
  $v_k \leftarrow v$; mark(v) $\leftarrow$ true
  $(w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2) \leftarrow C_{k-1}$
  $(w_p, \ldots, w_q) \leftarrow$ unmarked neighbors of v_k
  out(w_i) $\leftarrow$ true for all $p < i < q$
  update chords($\cdot$) for these w_i and their neighbors

- chord(v) = number of chords incident to v
- mark(v) = true iff vertex v was numbered
- out(v) = true iff v is currently in the external face

Lemma: Algorithm `CanonicalOrdering` computes a canonical ordering of G in $O(n)$ time.

Overview

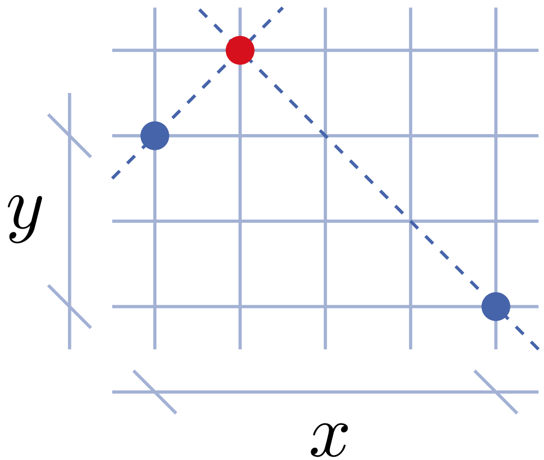
Canonical ordering

Shift algorithm

Implementation

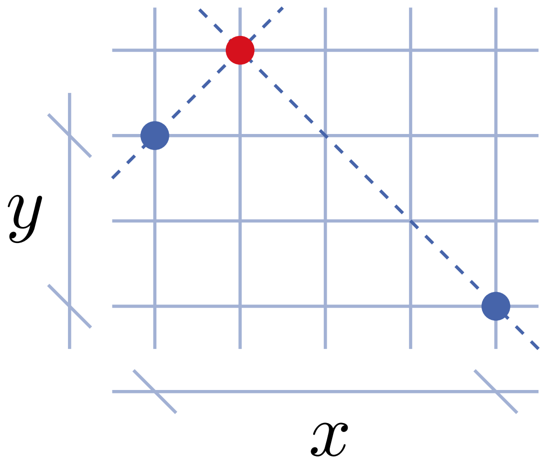
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance



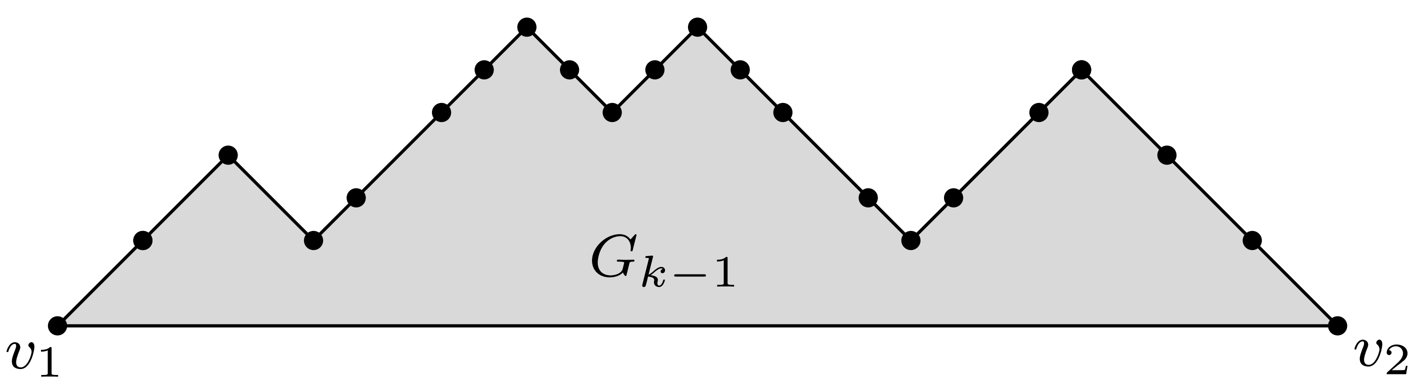
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance



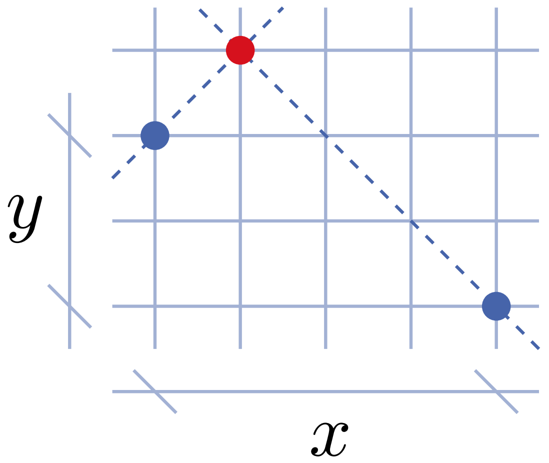
Algorithm invariants: G_{k-1} is drawn such that

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



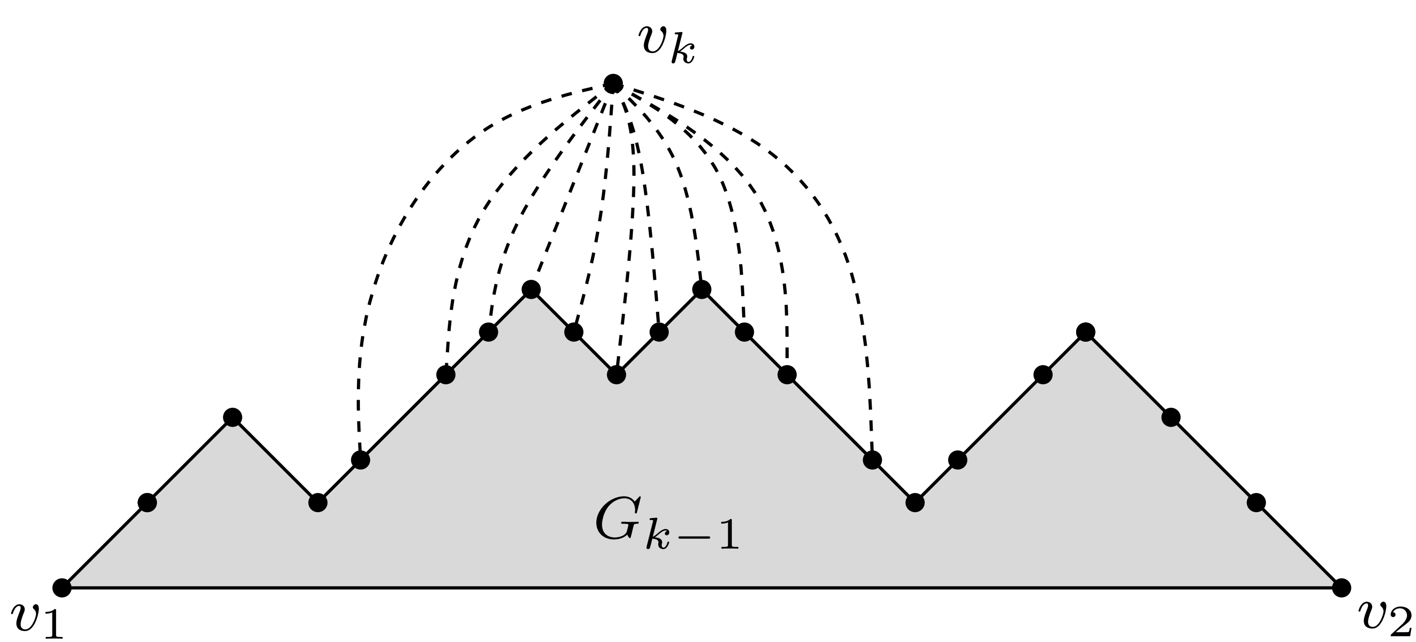
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance



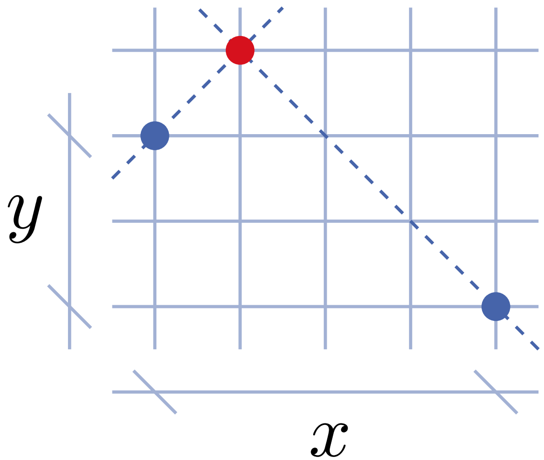
Algorithm invariants: G_{k-1} is drawn such that

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



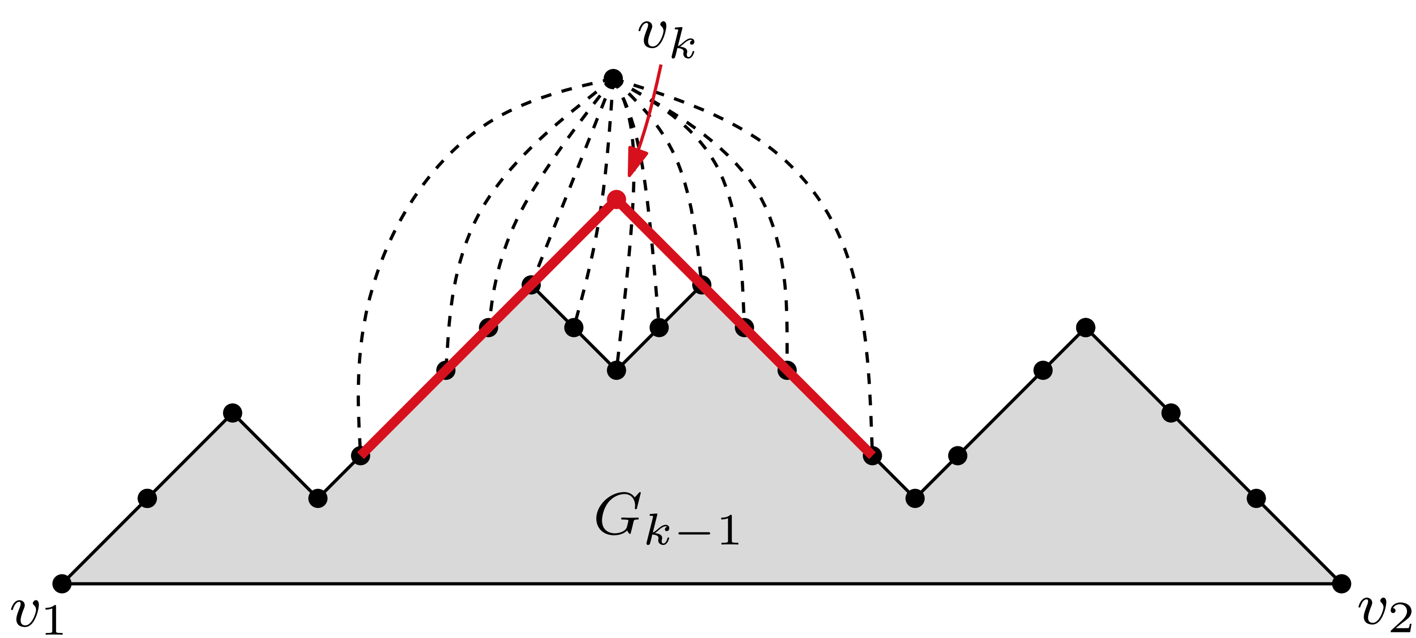
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance

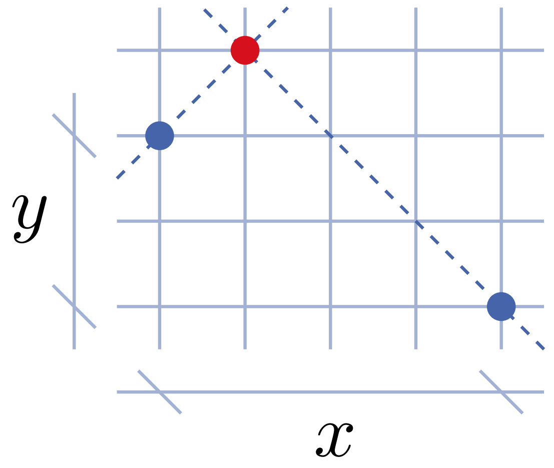


Algorithm invariants: G_{k-1} is drawn such that

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

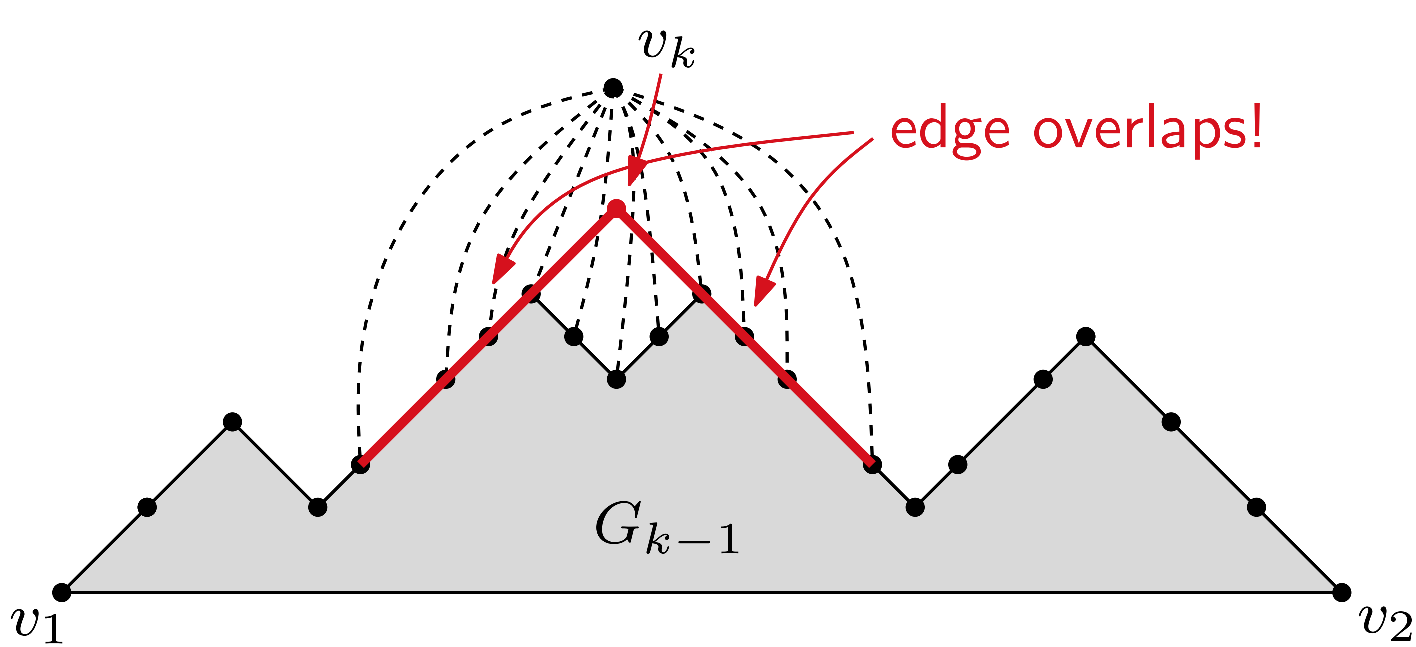


even L_1-distance



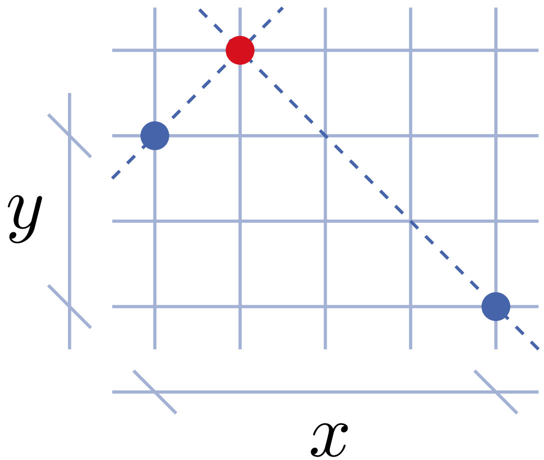
Algorithm invariants: G_{k-1} is drawn such that

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



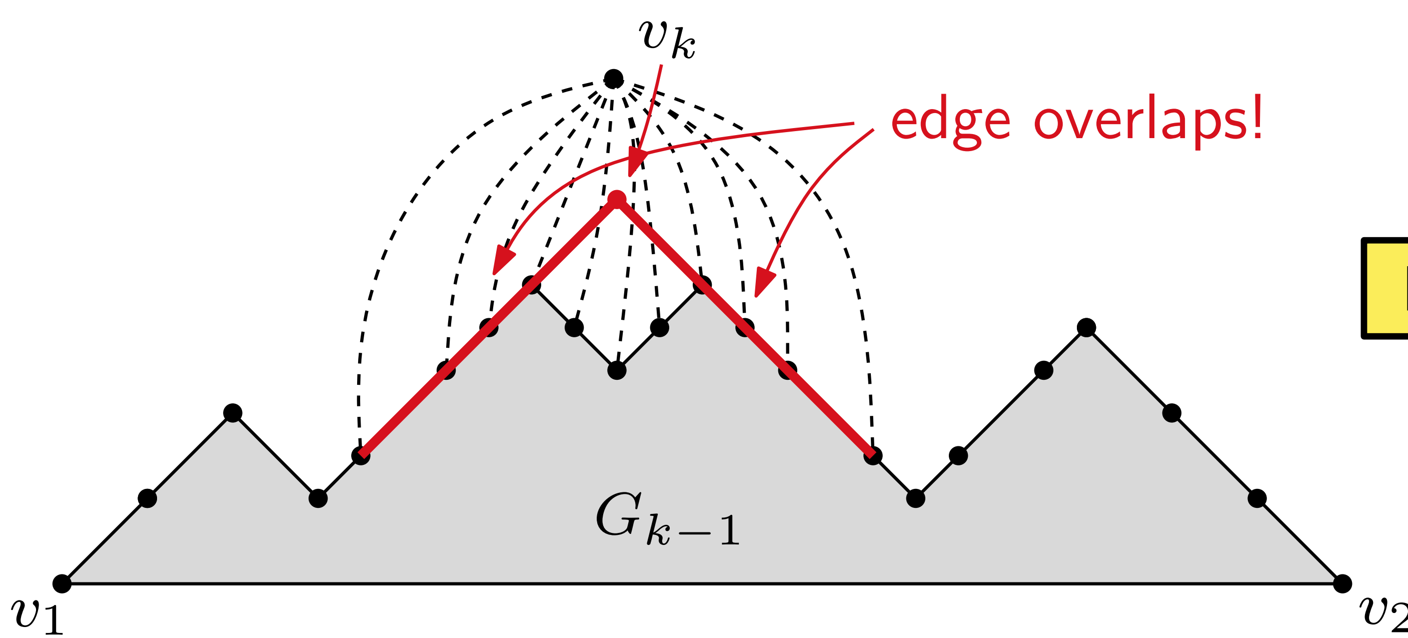
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance



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

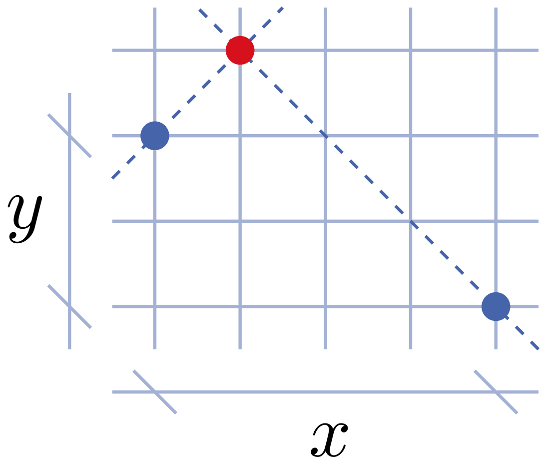
- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$
- boundary C_{k-1} of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone
- each edge of C_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1



How to fix this?

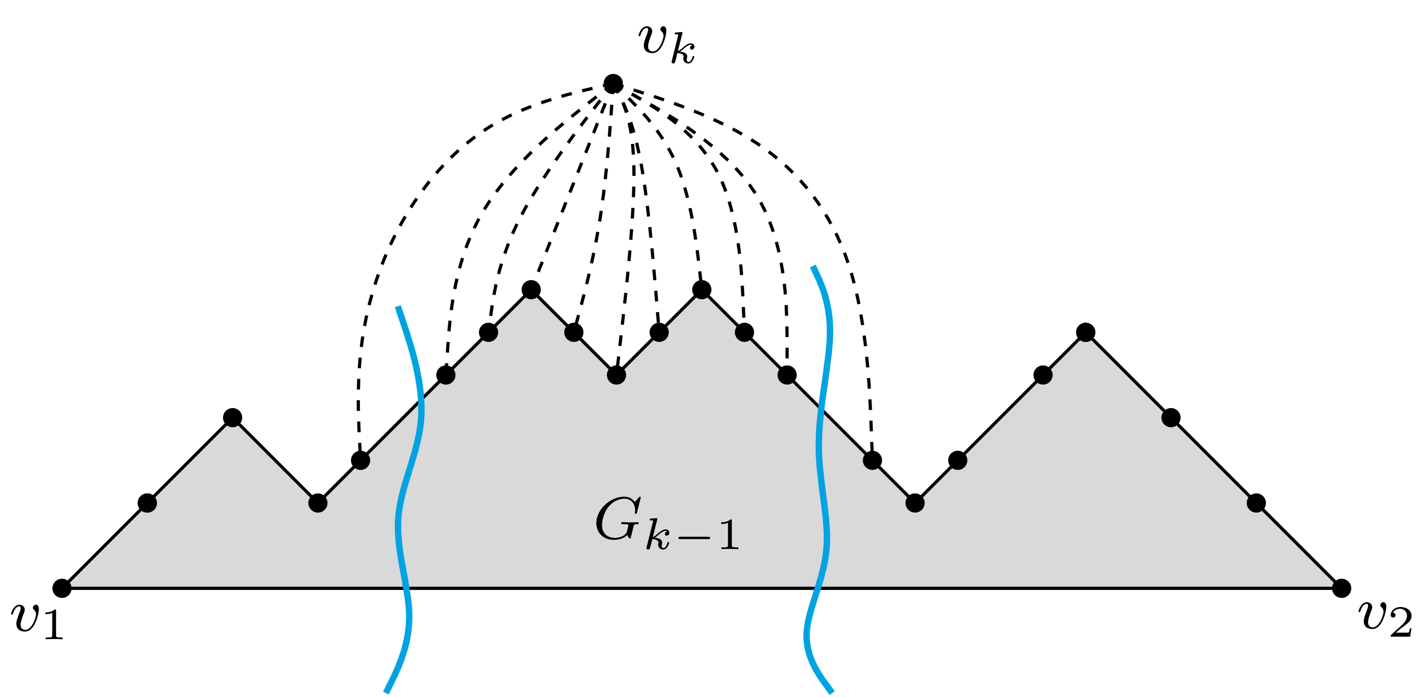
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance



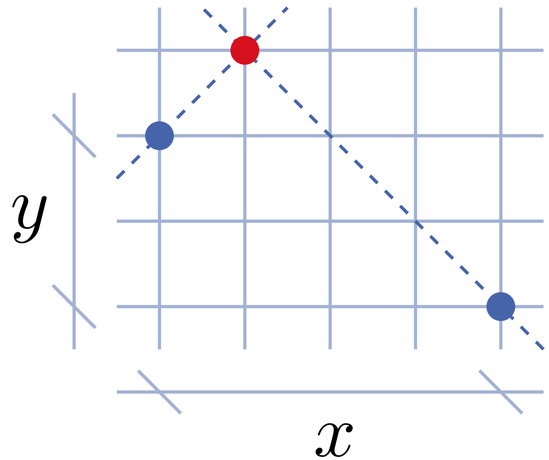
Algorithm invariants: G_{k-1} is drawn such that

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



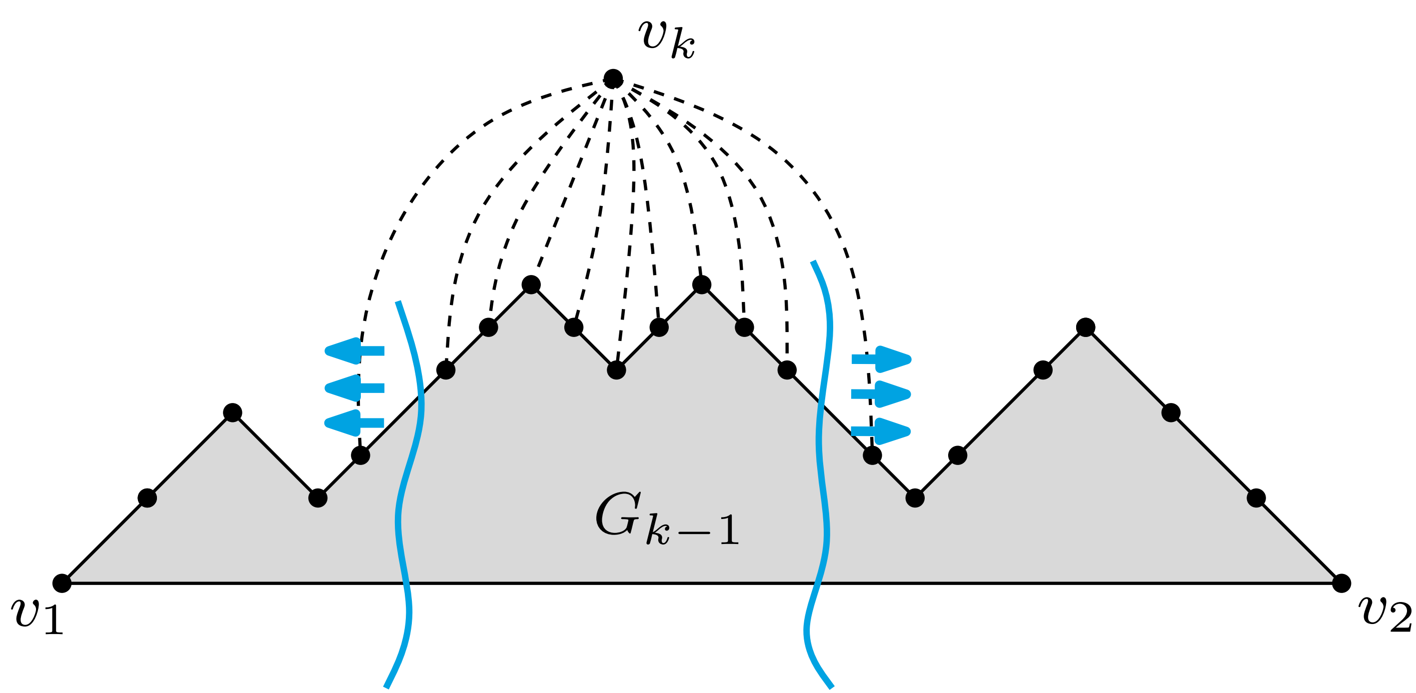
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance



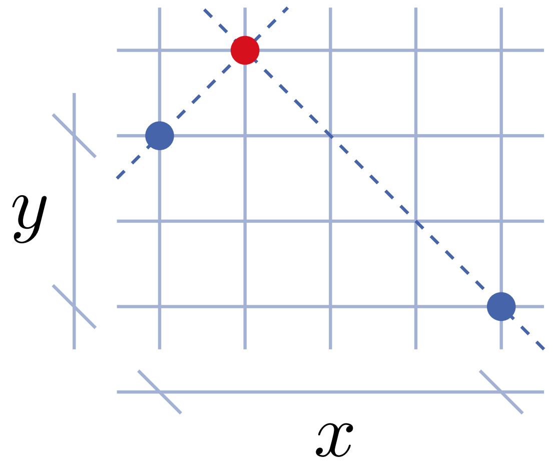
Algorithm invariants: G_{k-1} is drawn such that

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



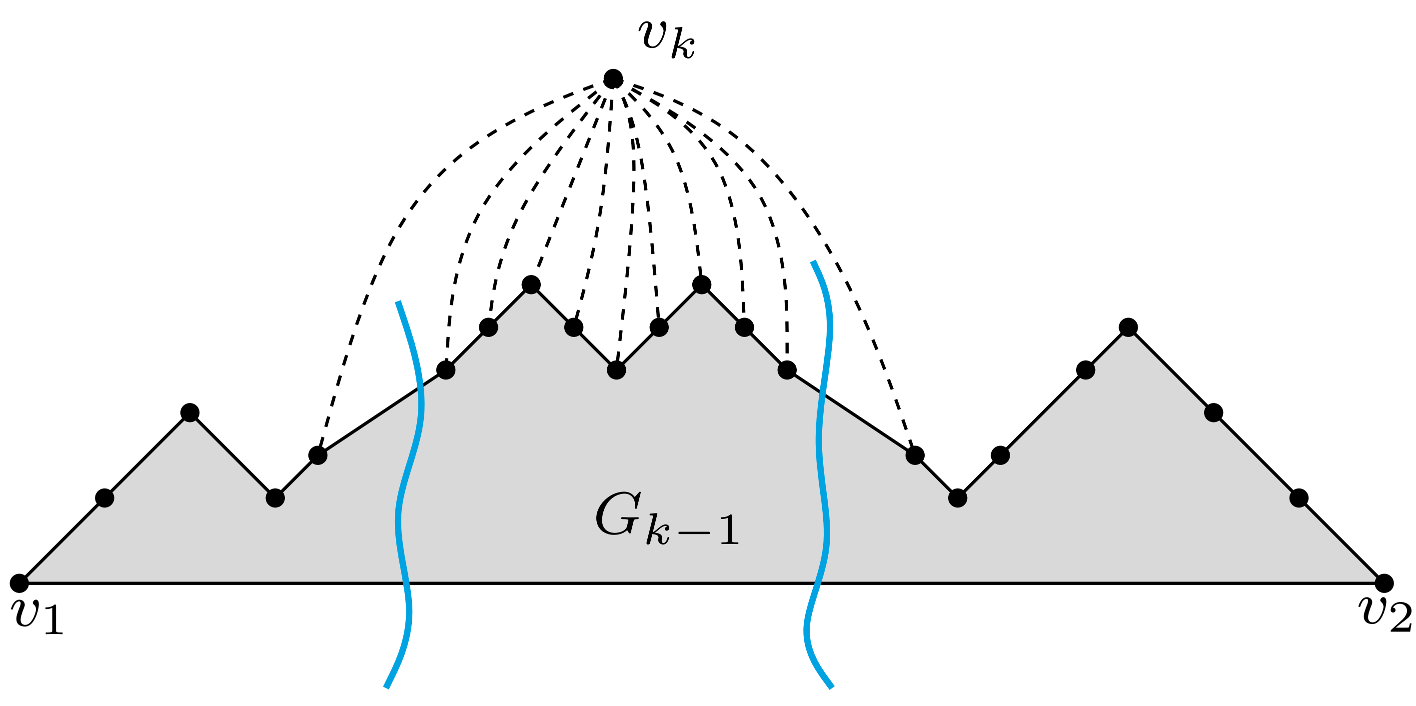
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance



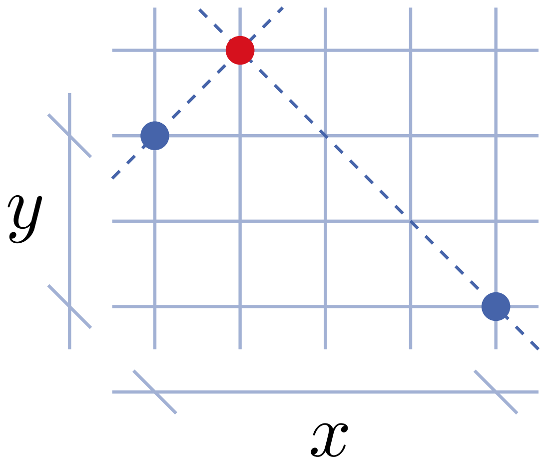
Algorithm invariants: G_{k-1} is drawn such that

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



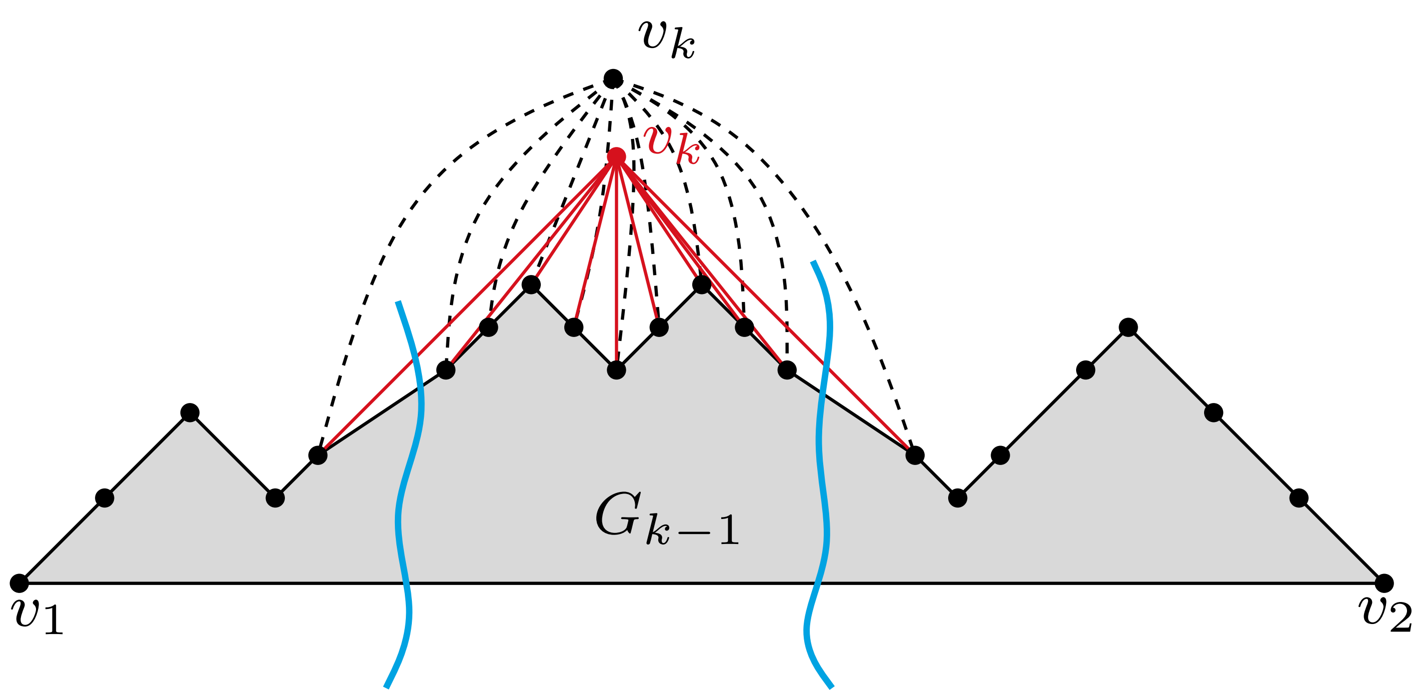
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance



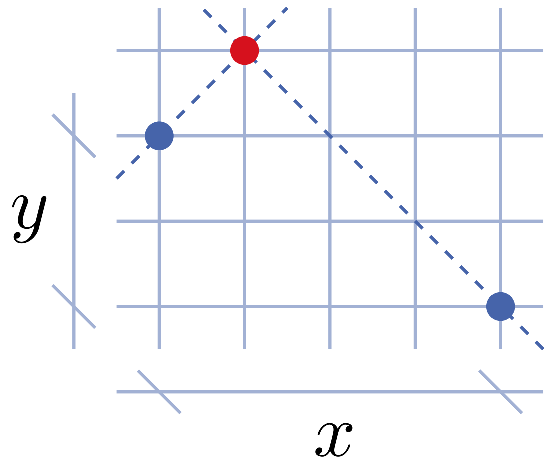
Algorithm invariants: G_{k-1} is drawn such that

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



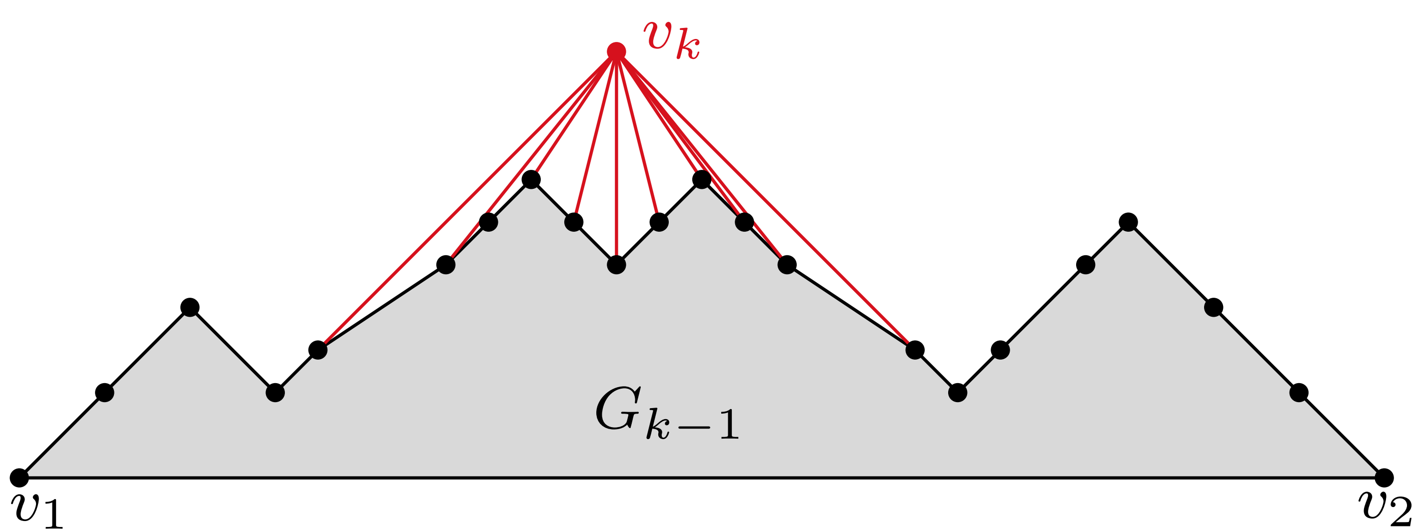
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance



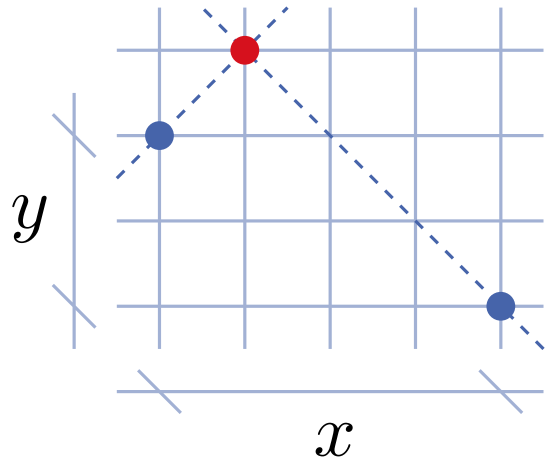
Algorithm invariants: G_{k-1} is drawn such that

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



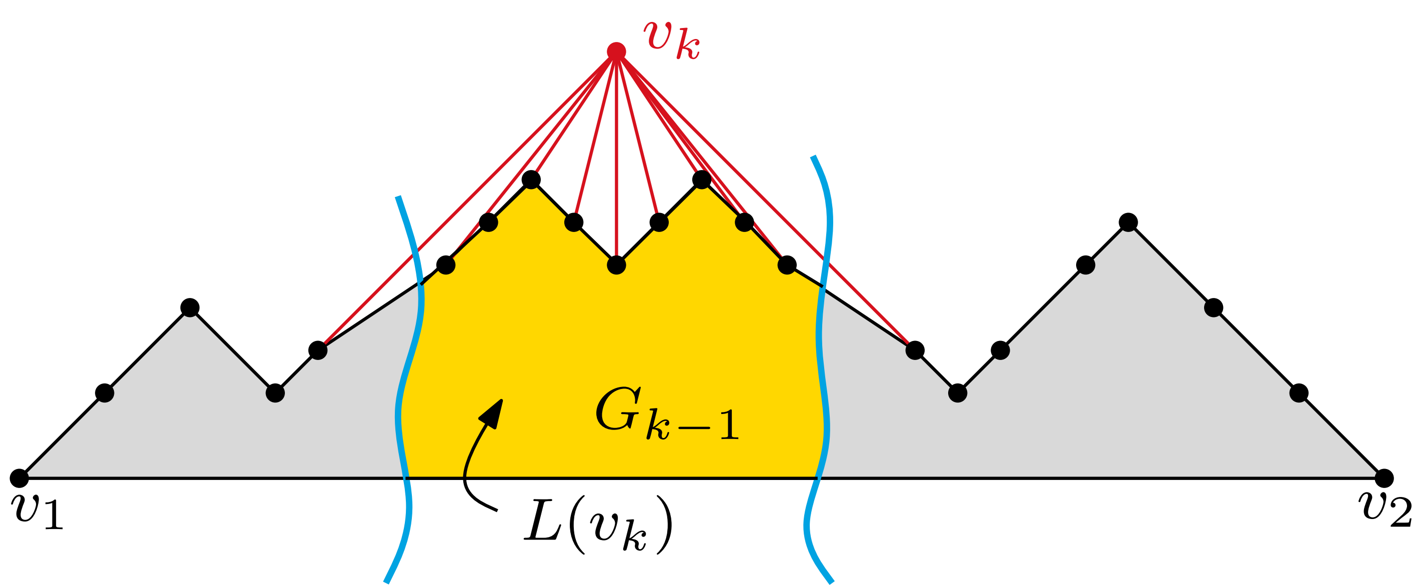
Shift Algorithm [de Fraysseix, Pach, Pollack 1988]

even L_1-distance

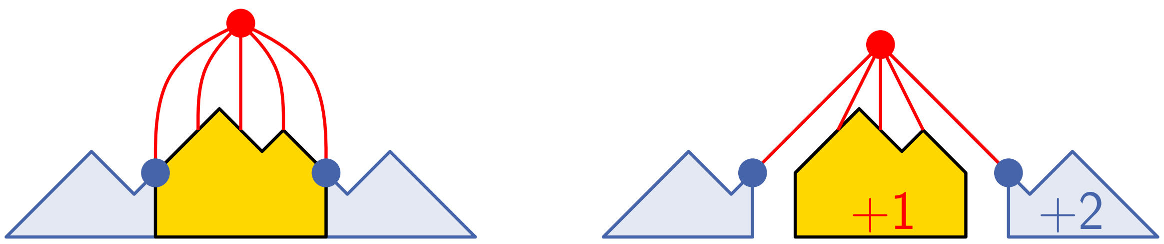


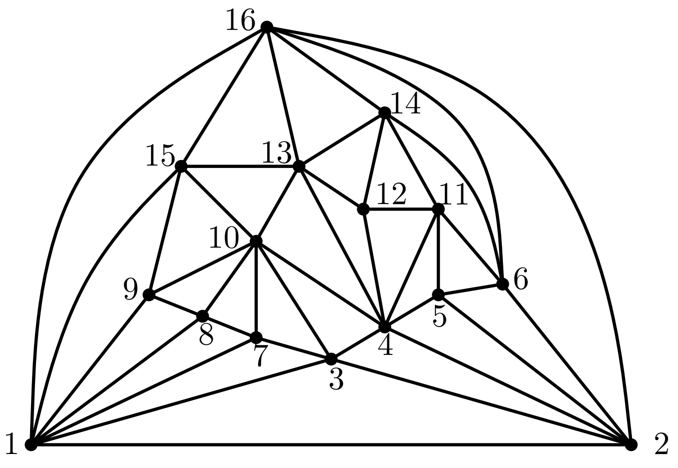
Algorithm invariants: G_{k-1} is drawn such that

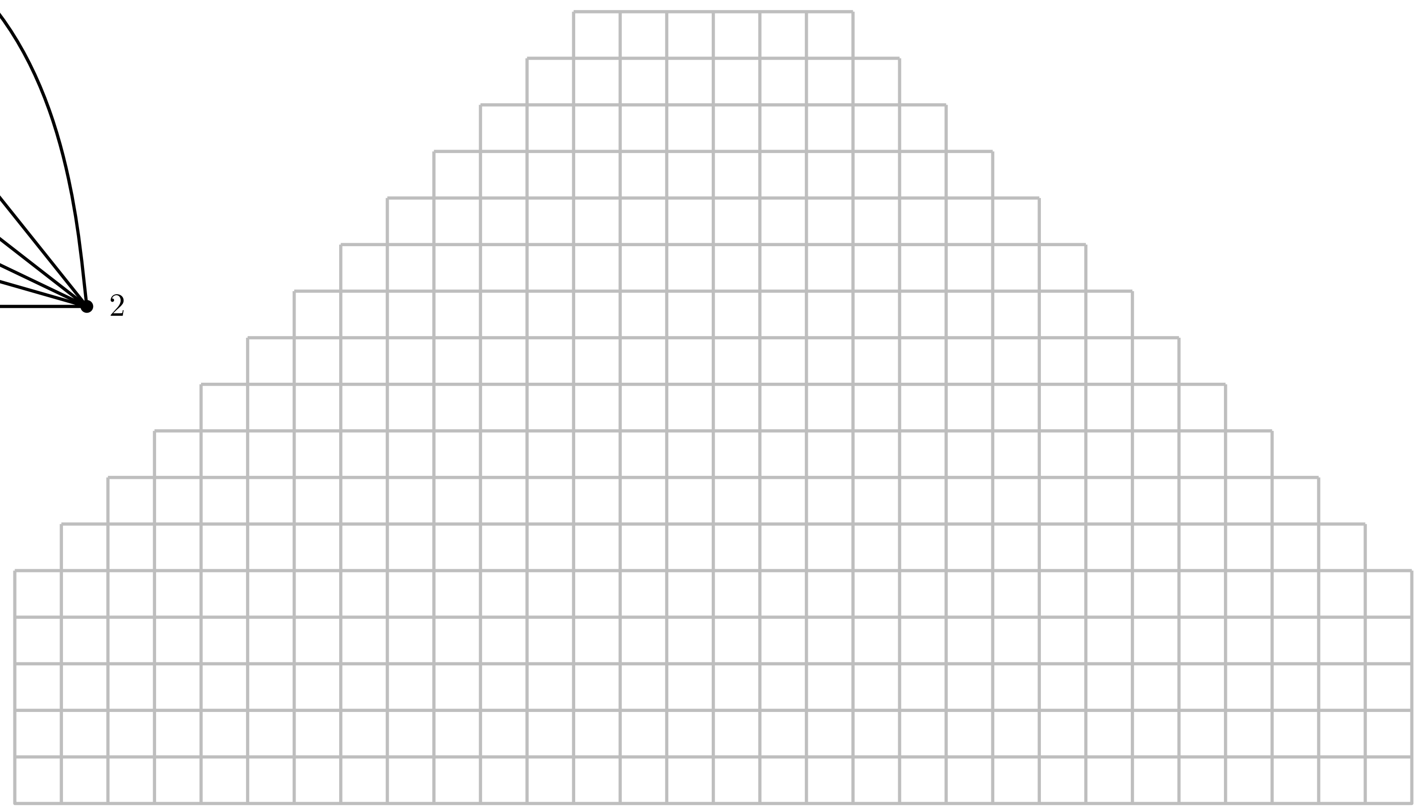
- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$
- boundary C_{k-1} of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone
- each edge of C_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1



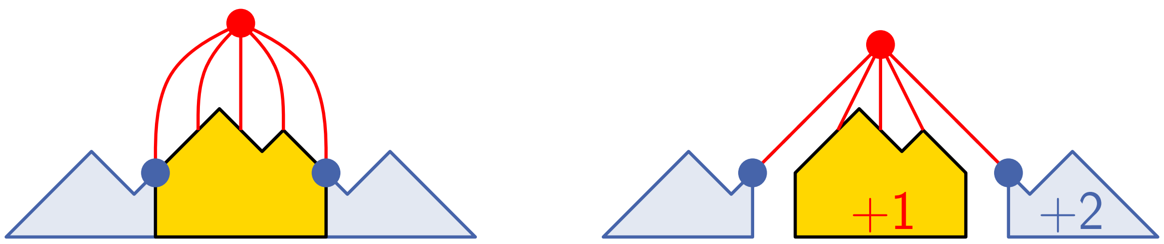
Shift Algorithm: Example

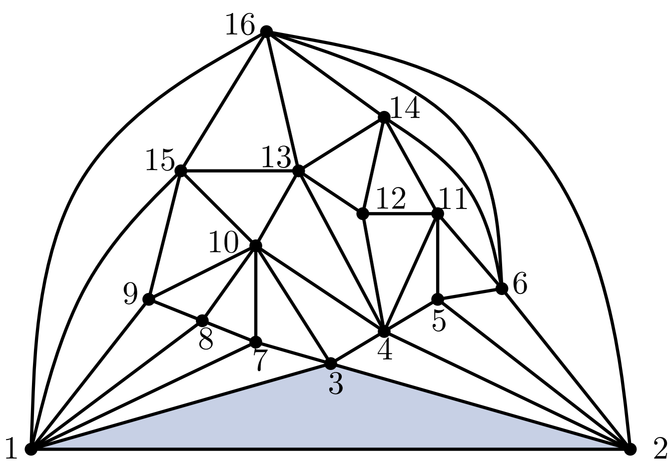


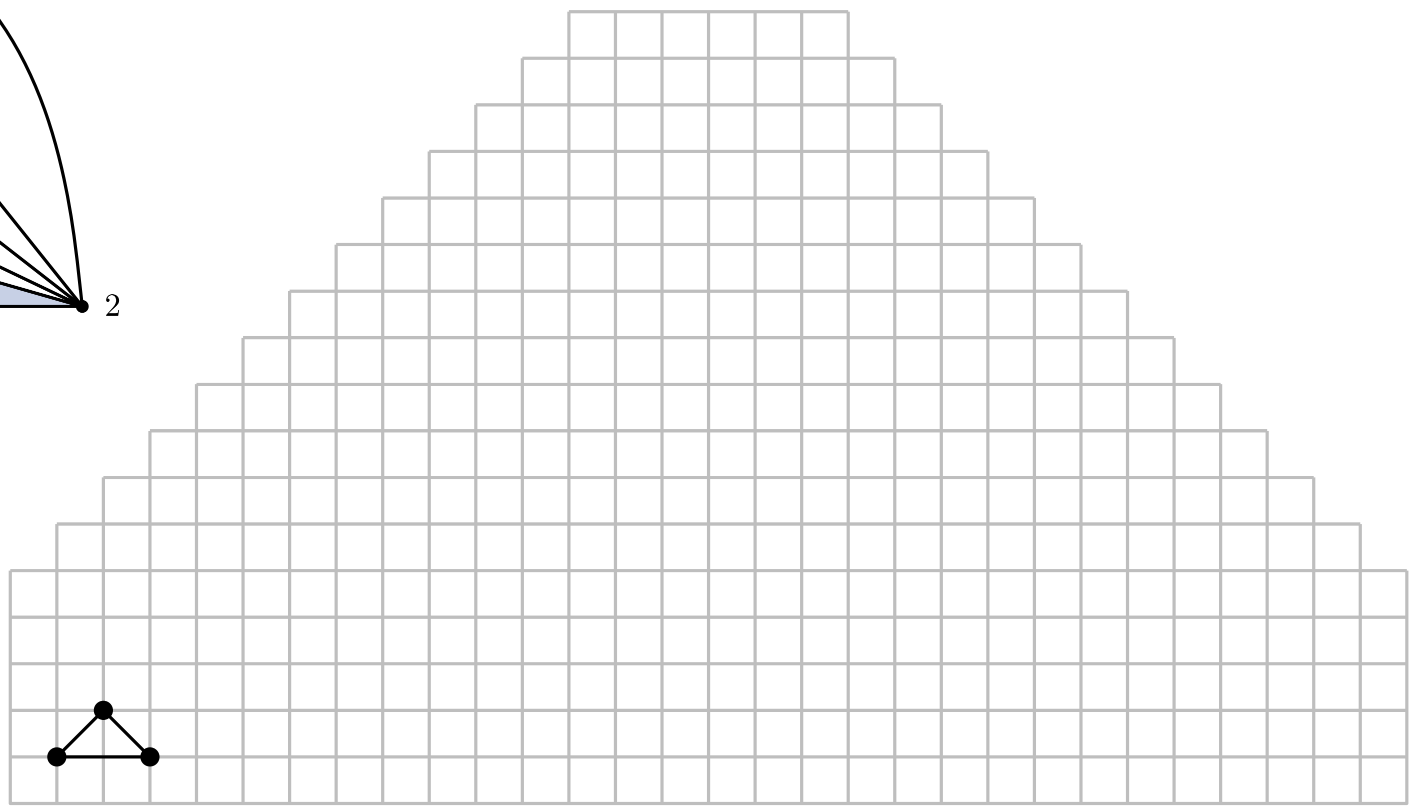




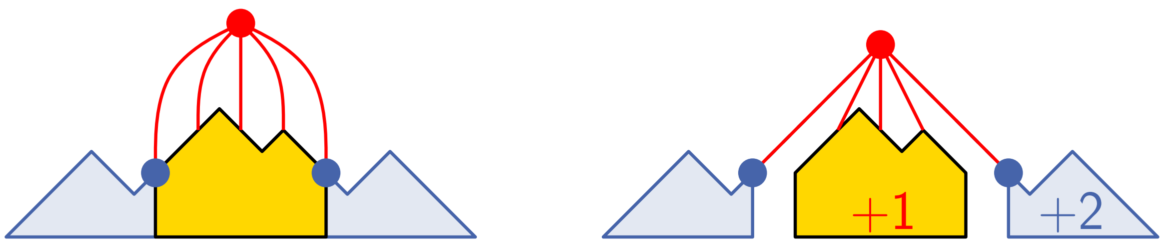
Shift Algorithm: Example

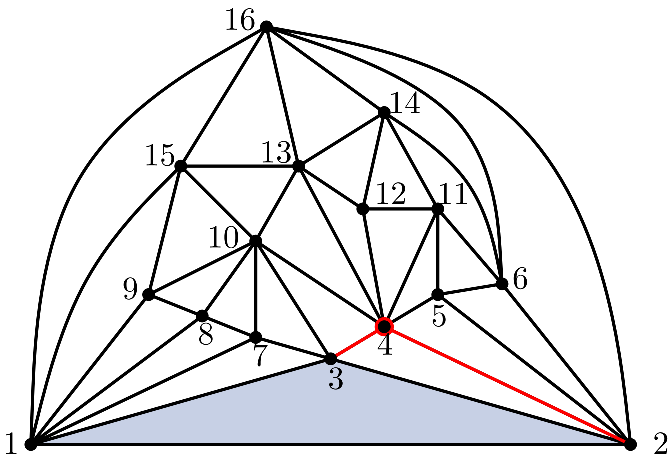


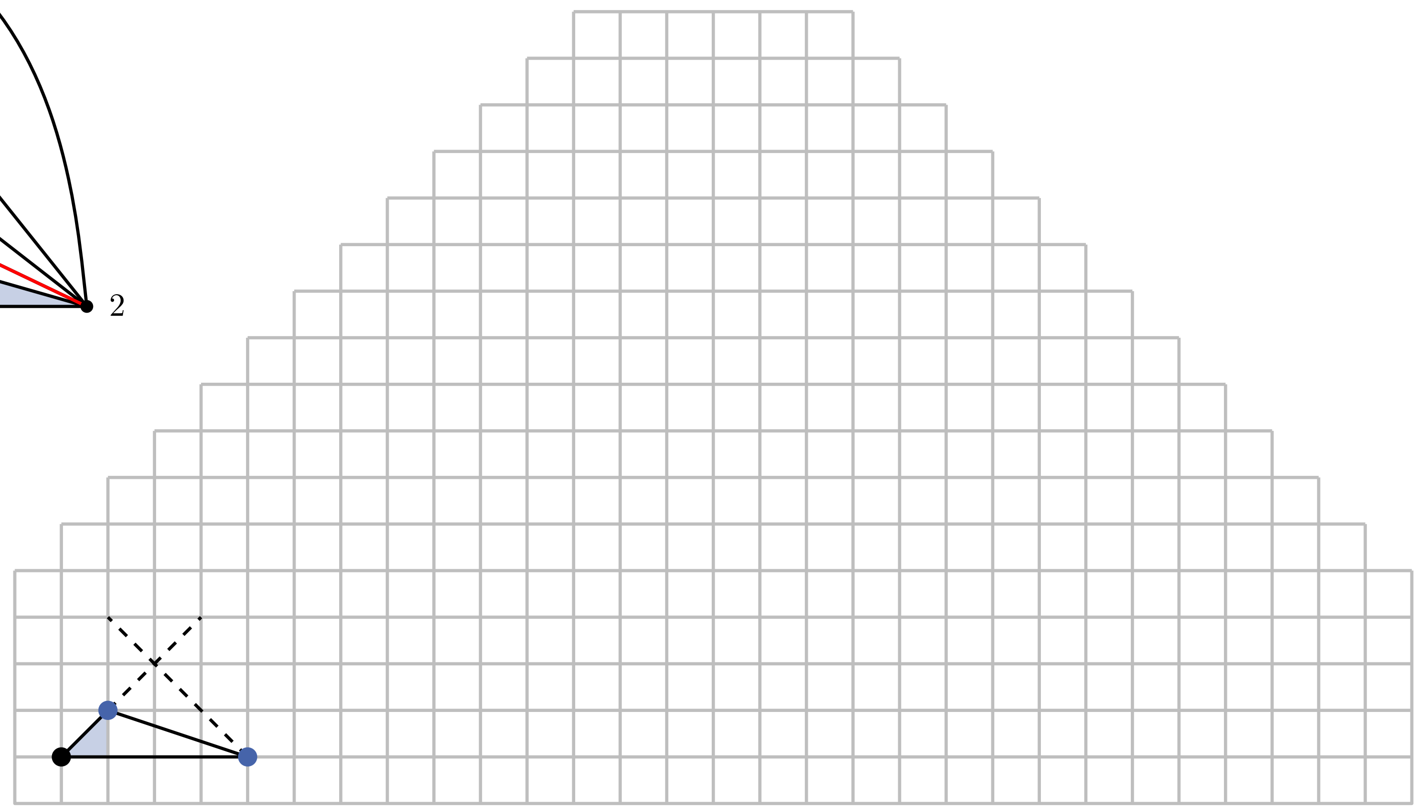




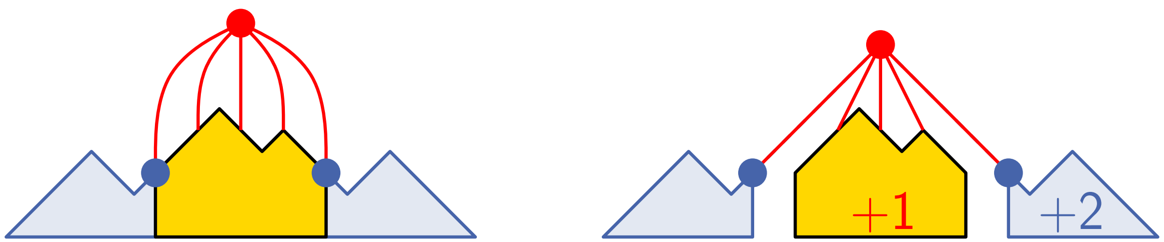
Shift Algorithm: Example

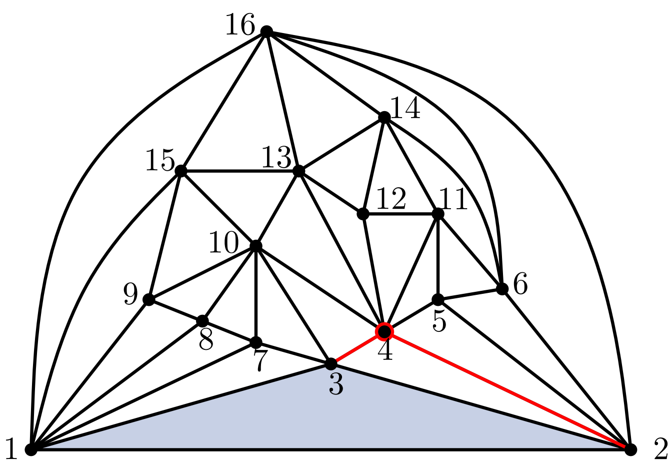


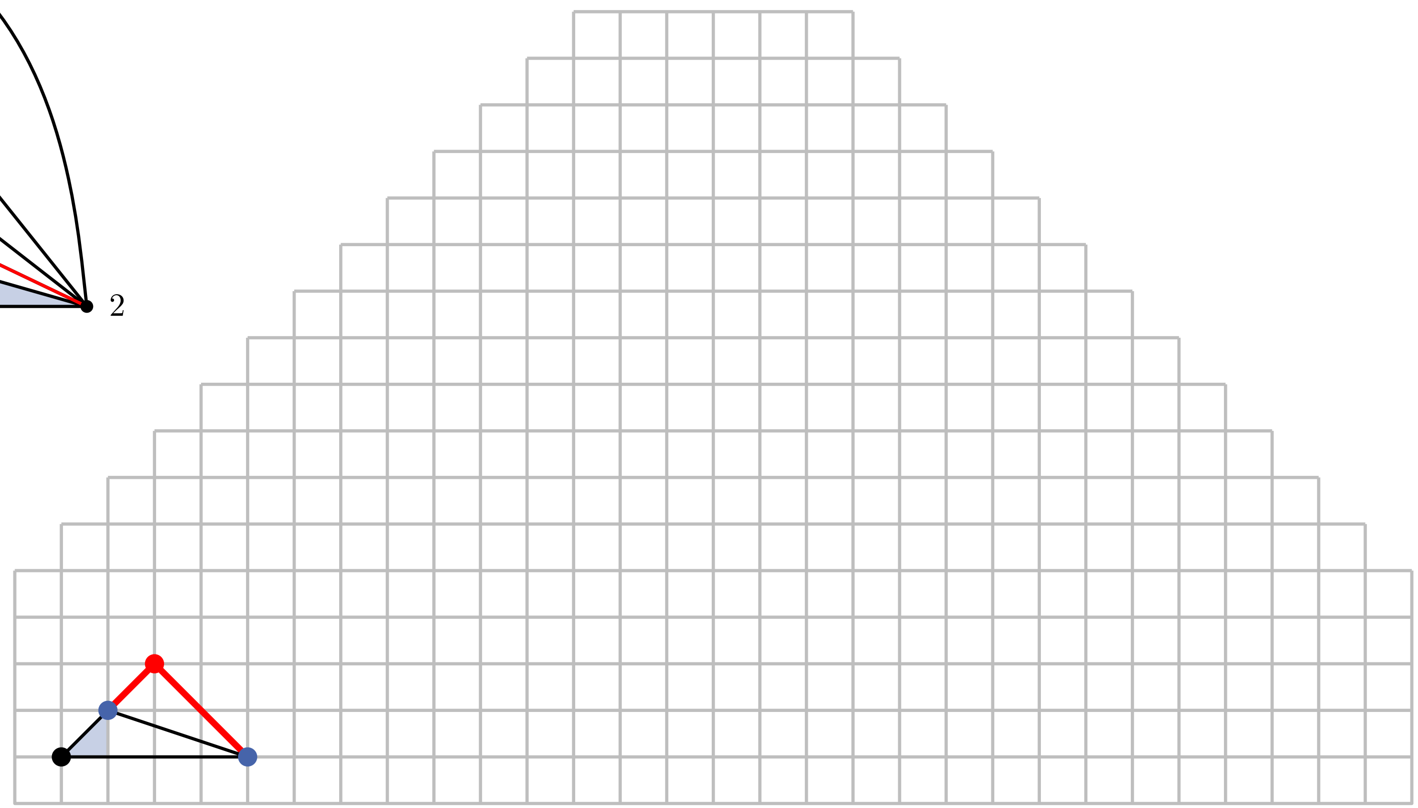




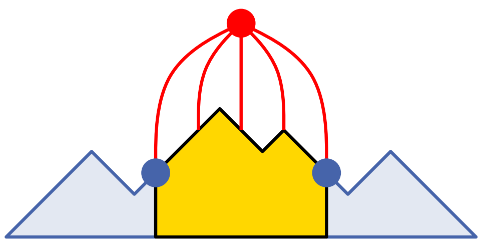
Shift Algorithm: Example

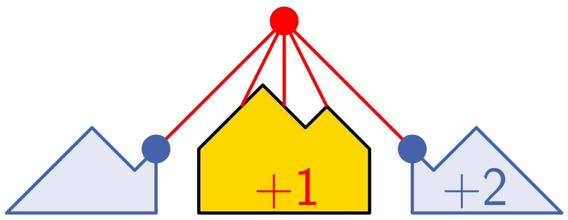


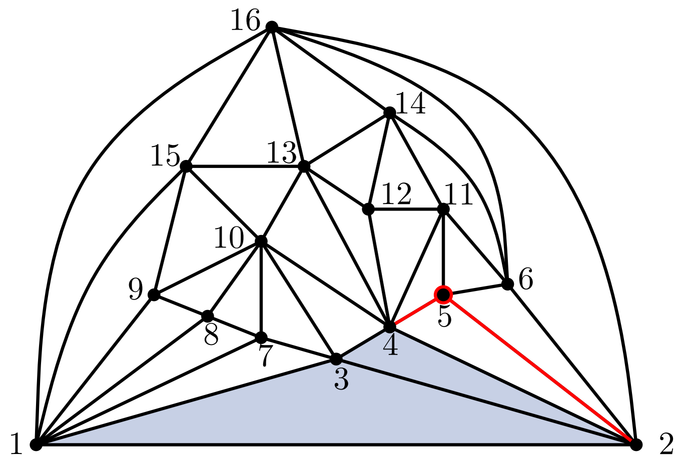


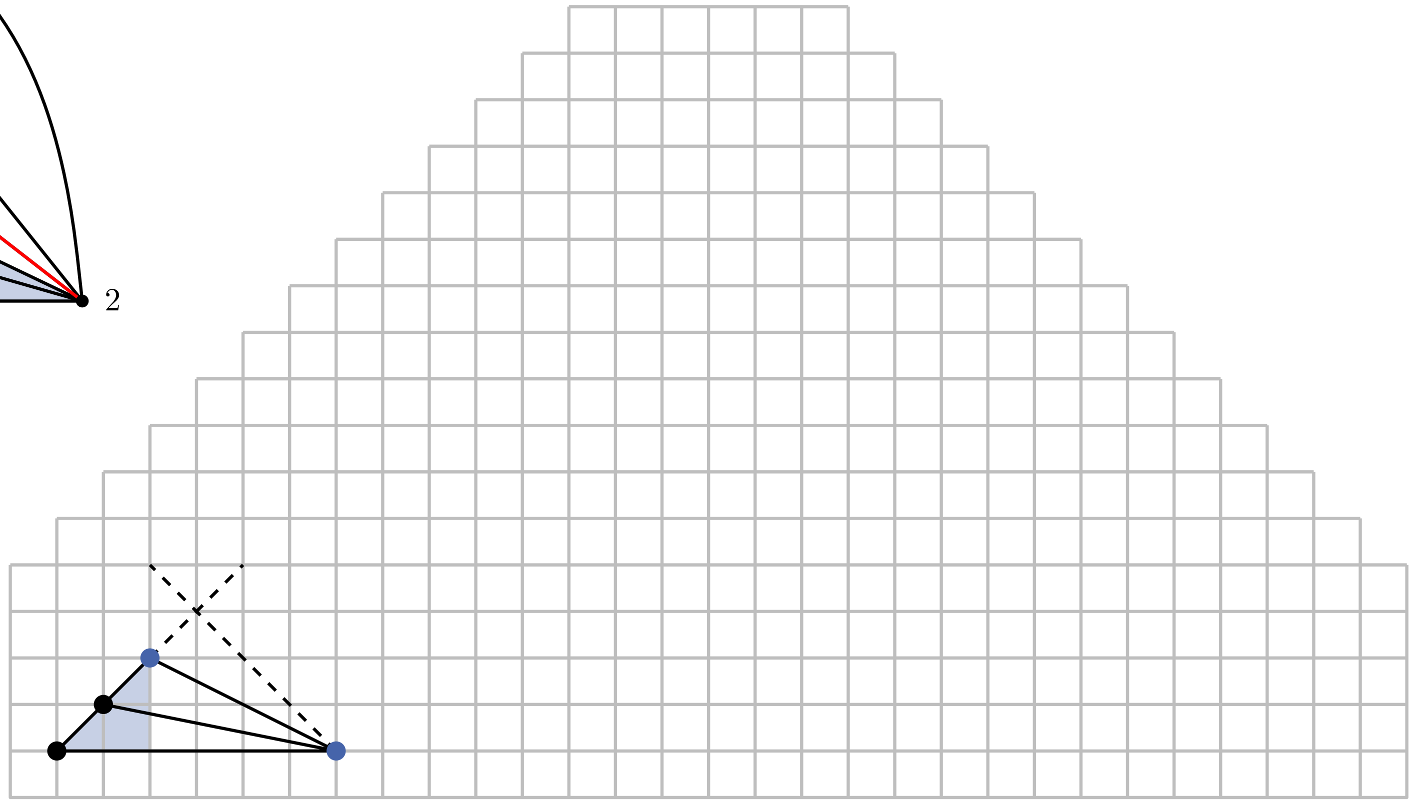


Shift Algorithm: Example

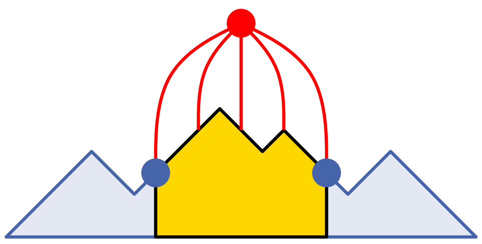


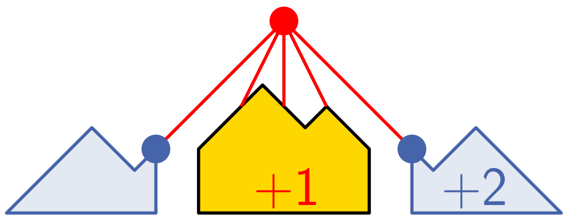


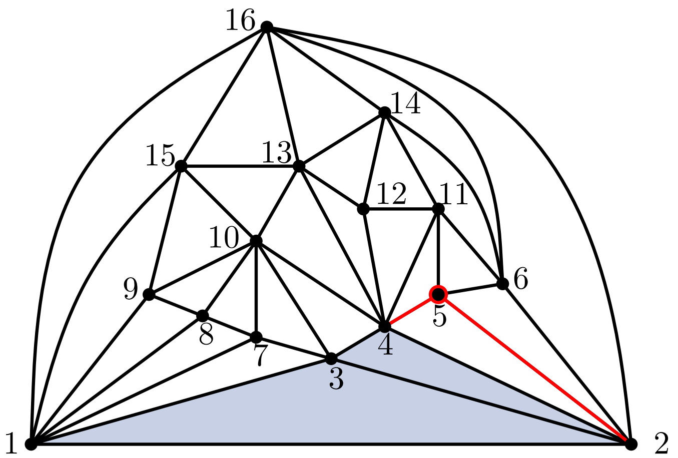


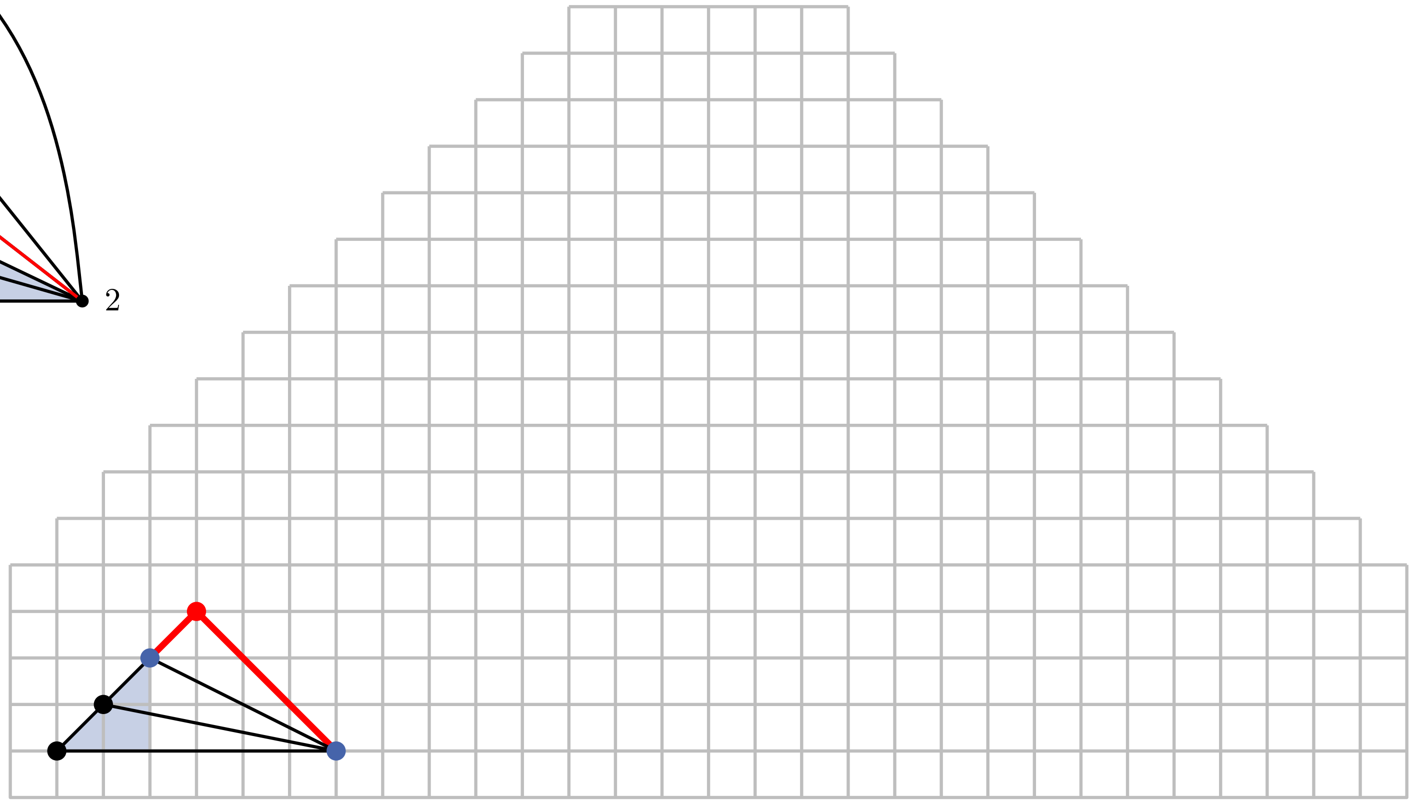


Shift Algorithm: Example

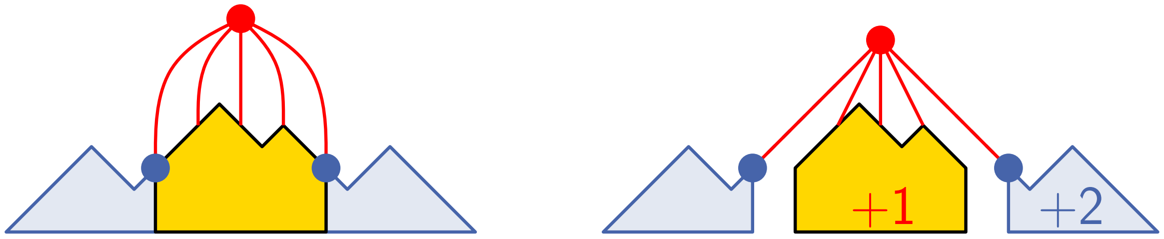


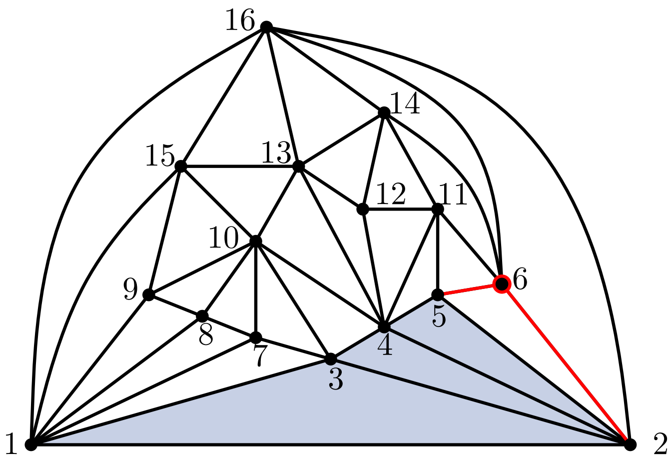


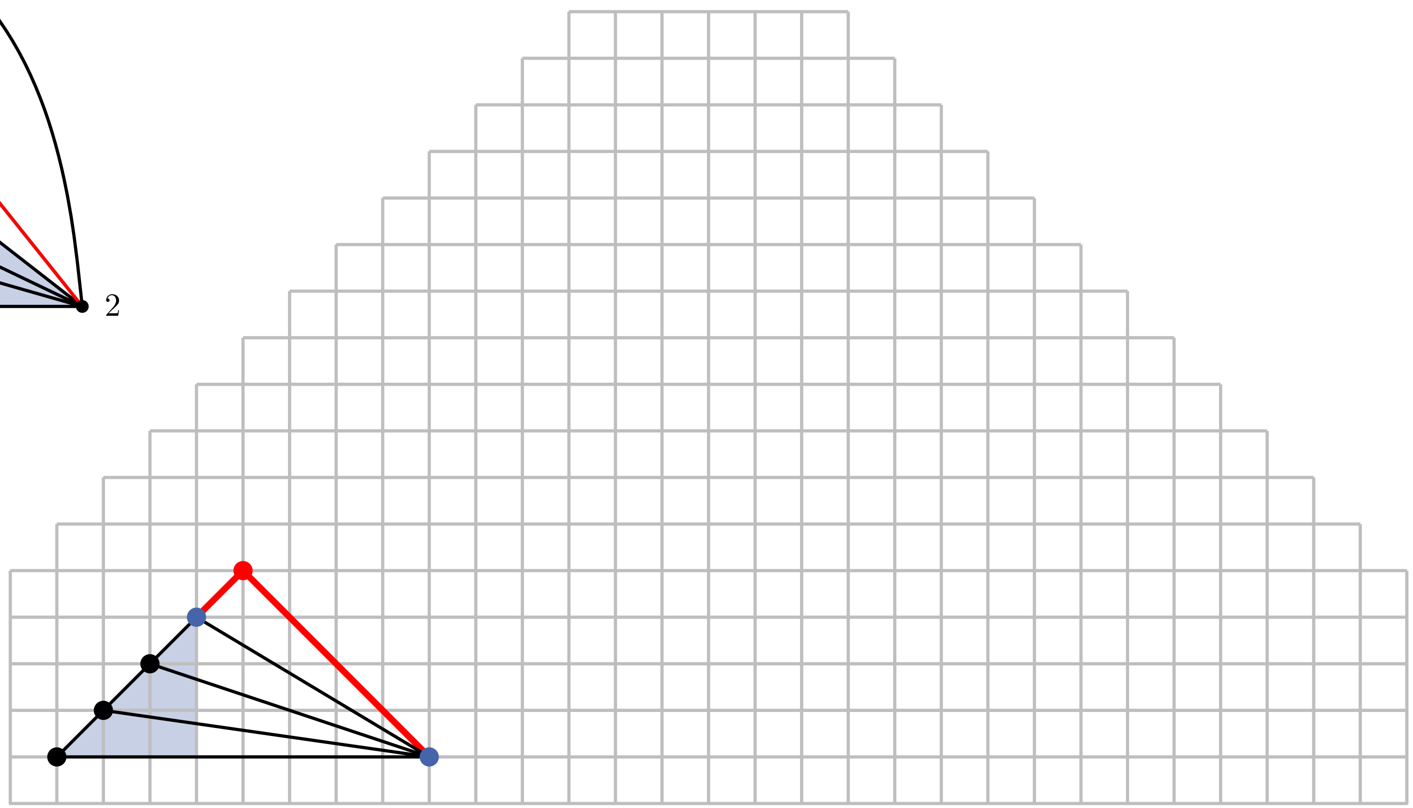




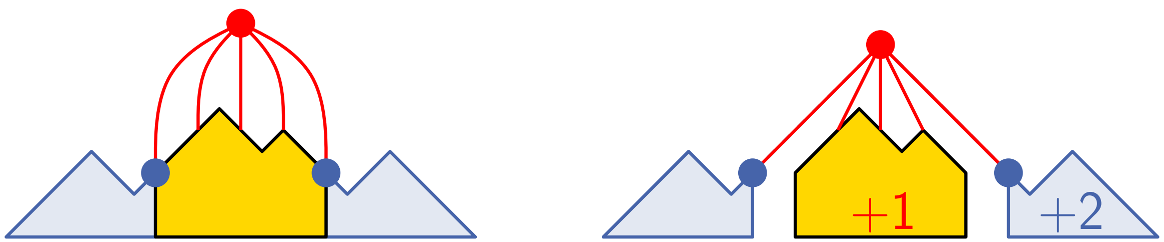
Shift Algorithm: Example

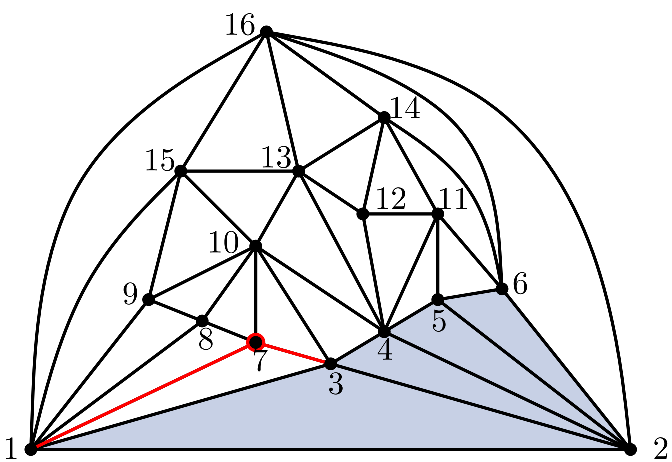


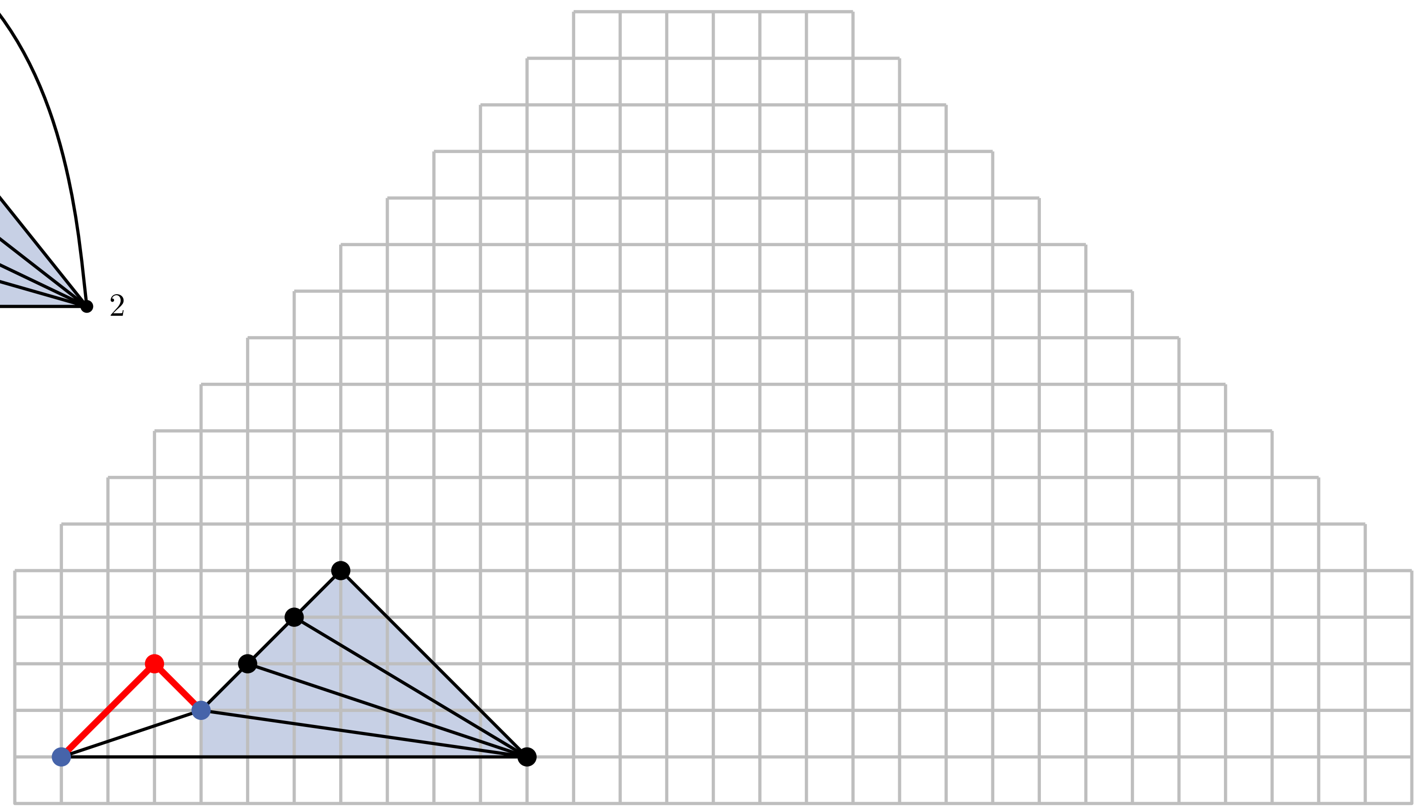




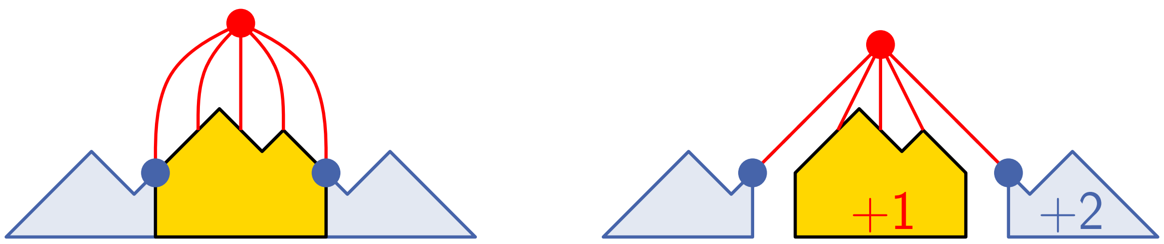
Shift Algorithm: Example

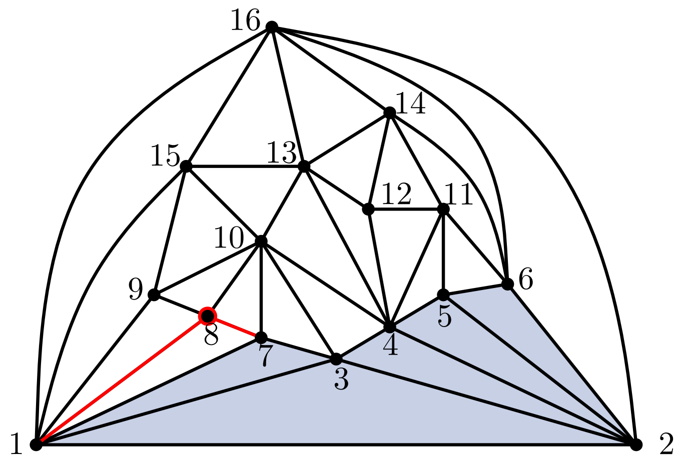


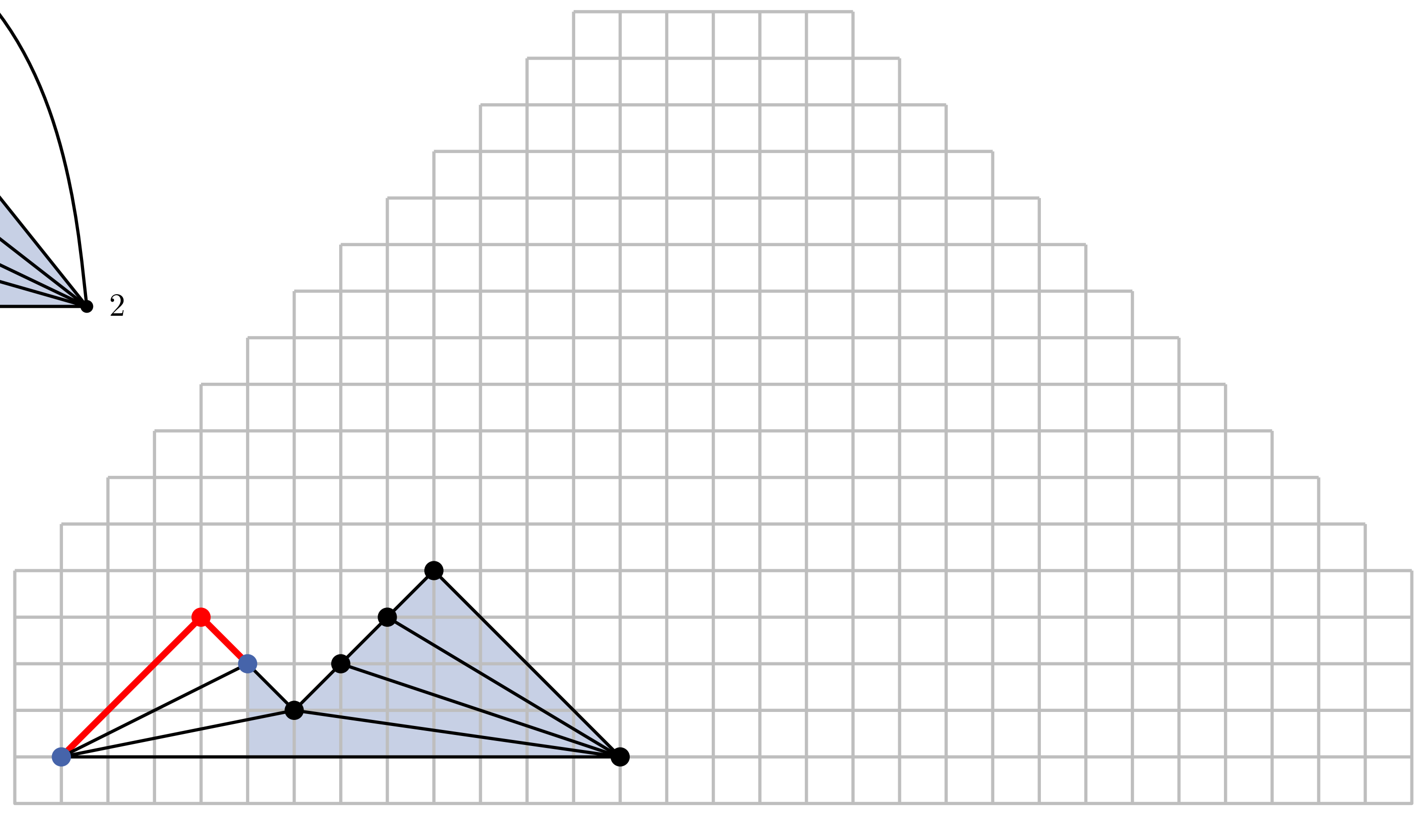




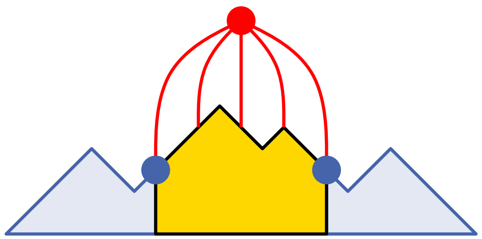
Shift Algorithm: Example

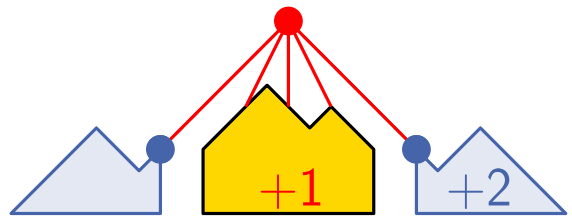


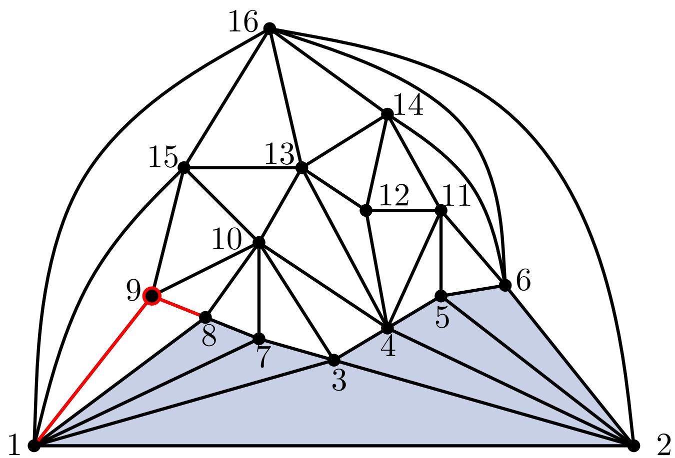


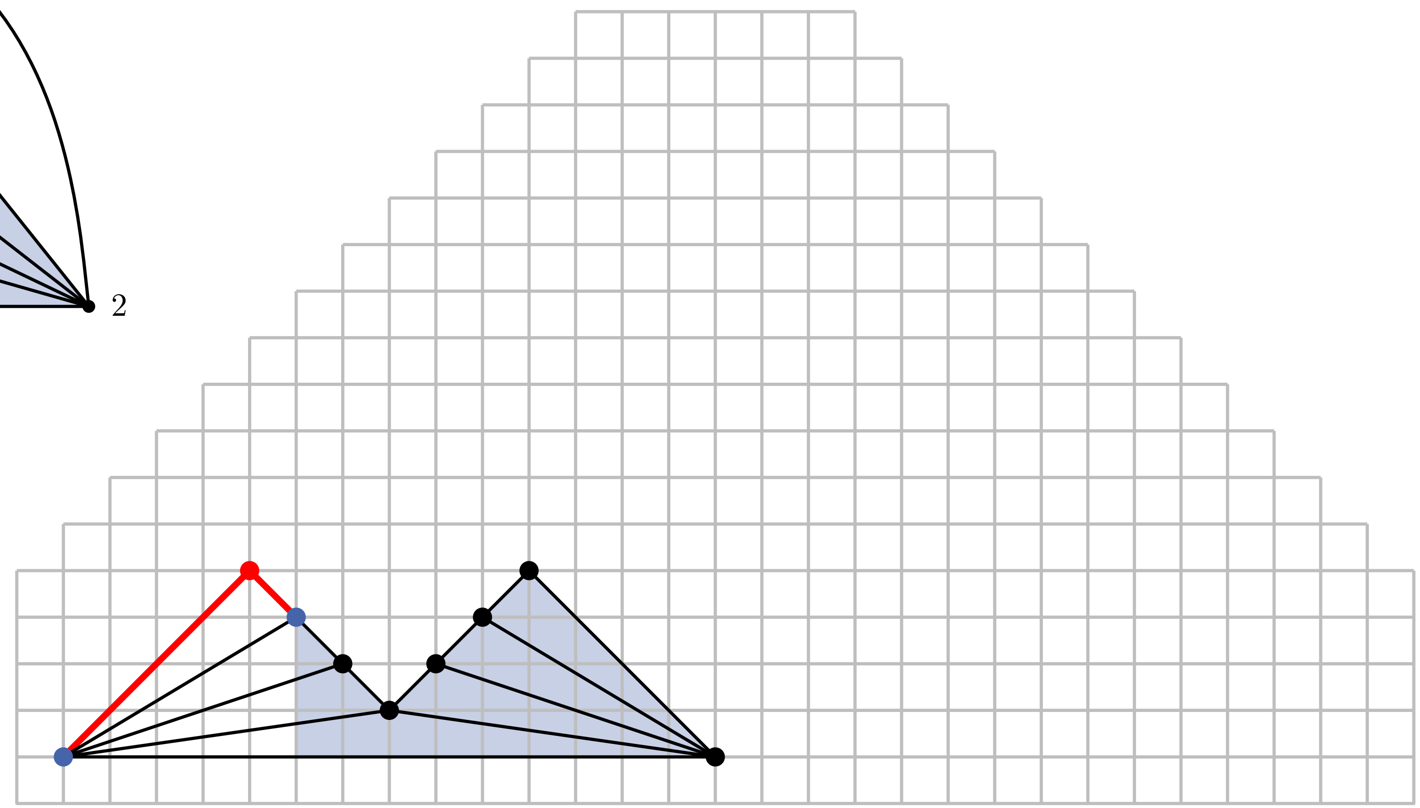


Shift Algorithm: Example

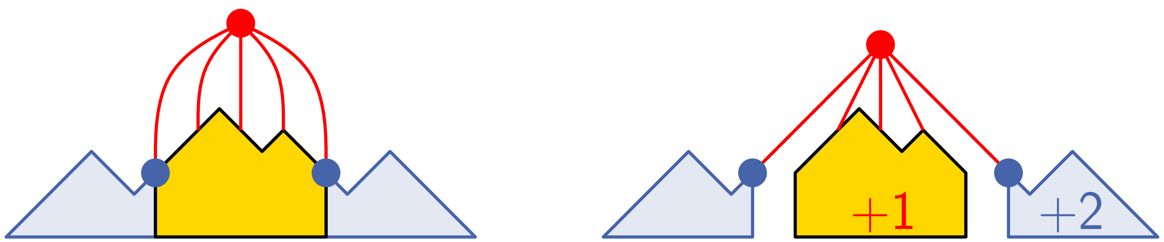


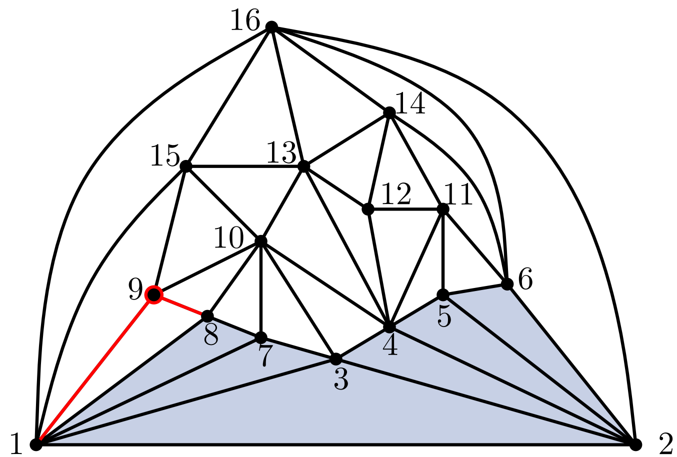


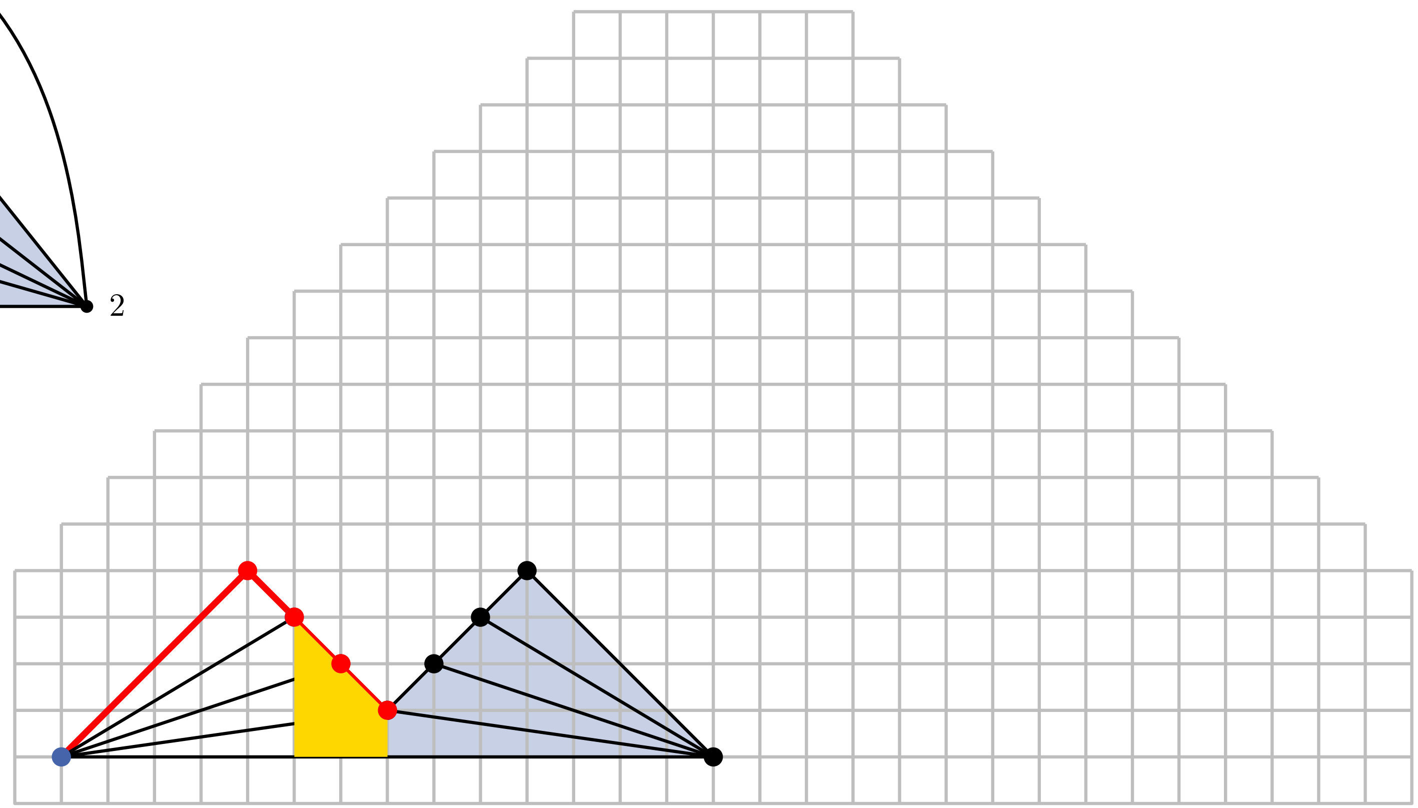




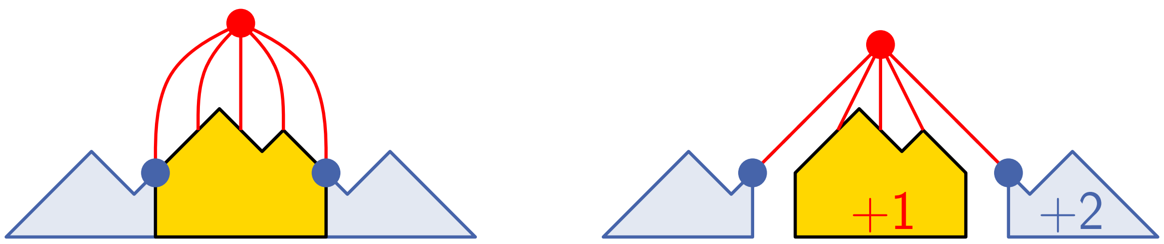
Shift Algorithm: Example

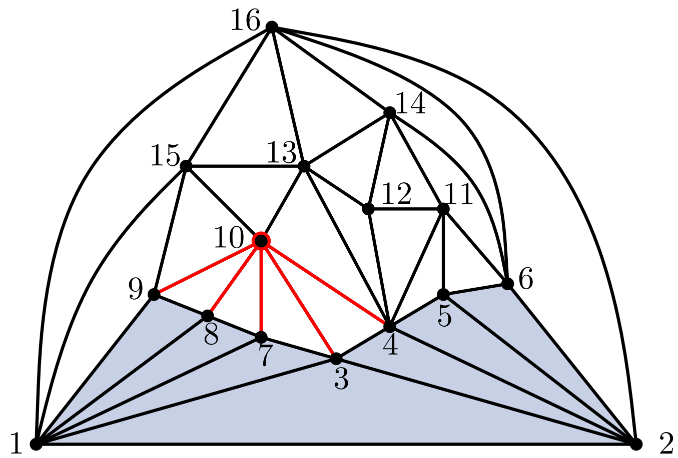


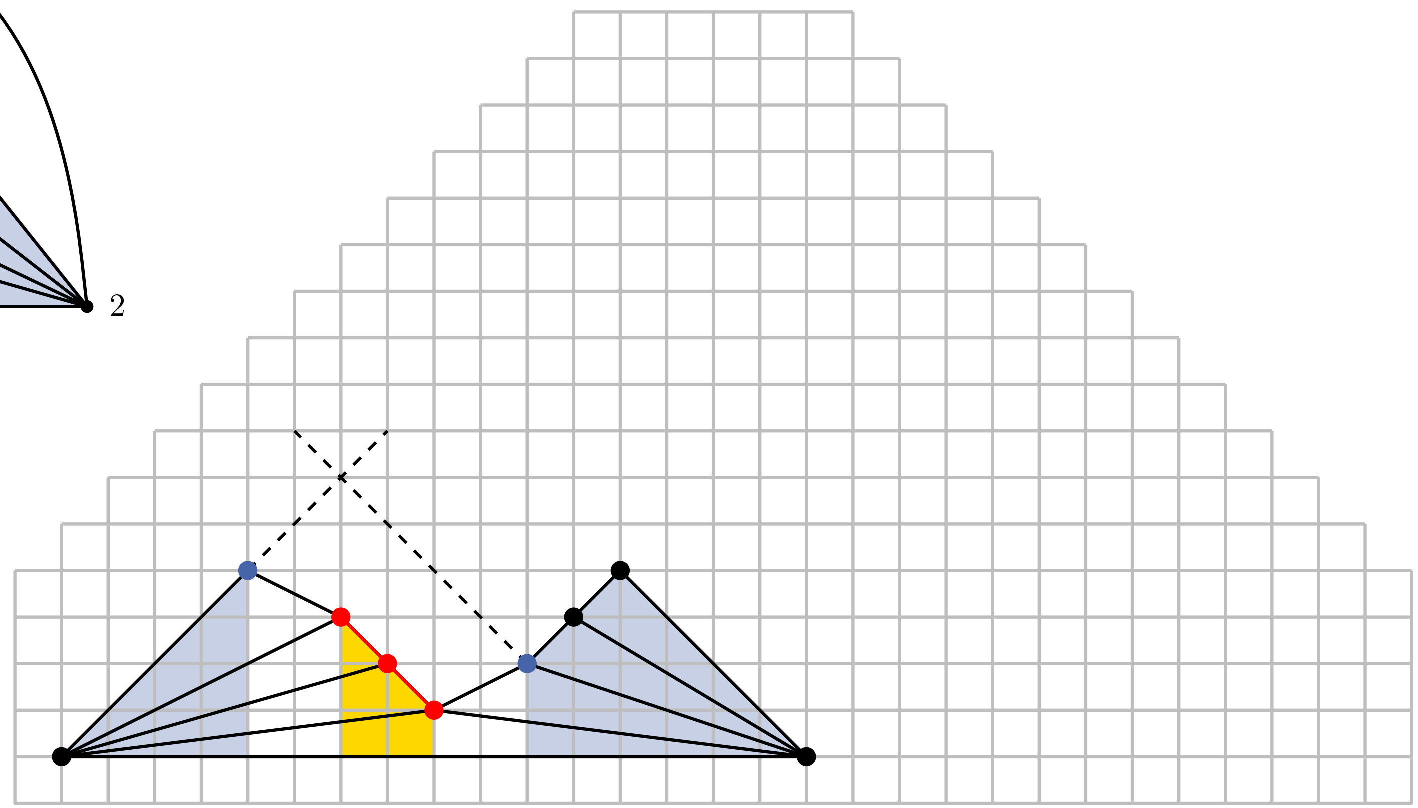




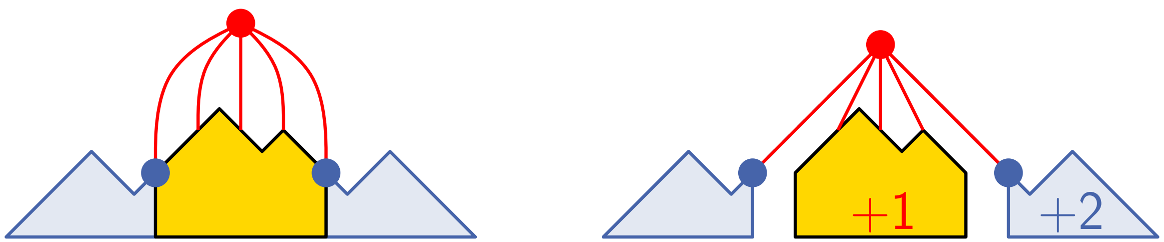
Shift Algorithm: Example

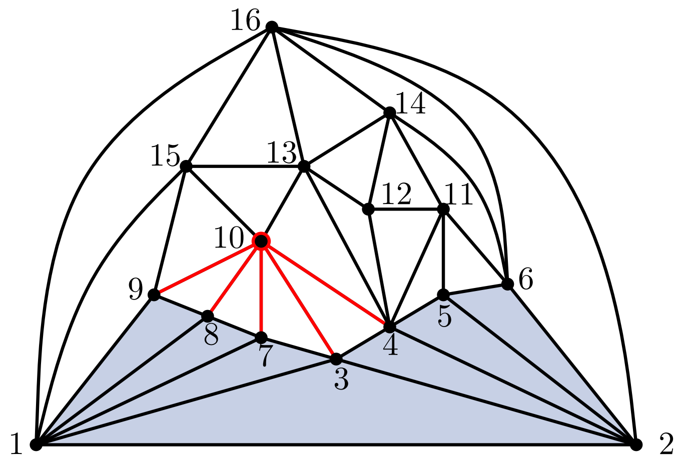


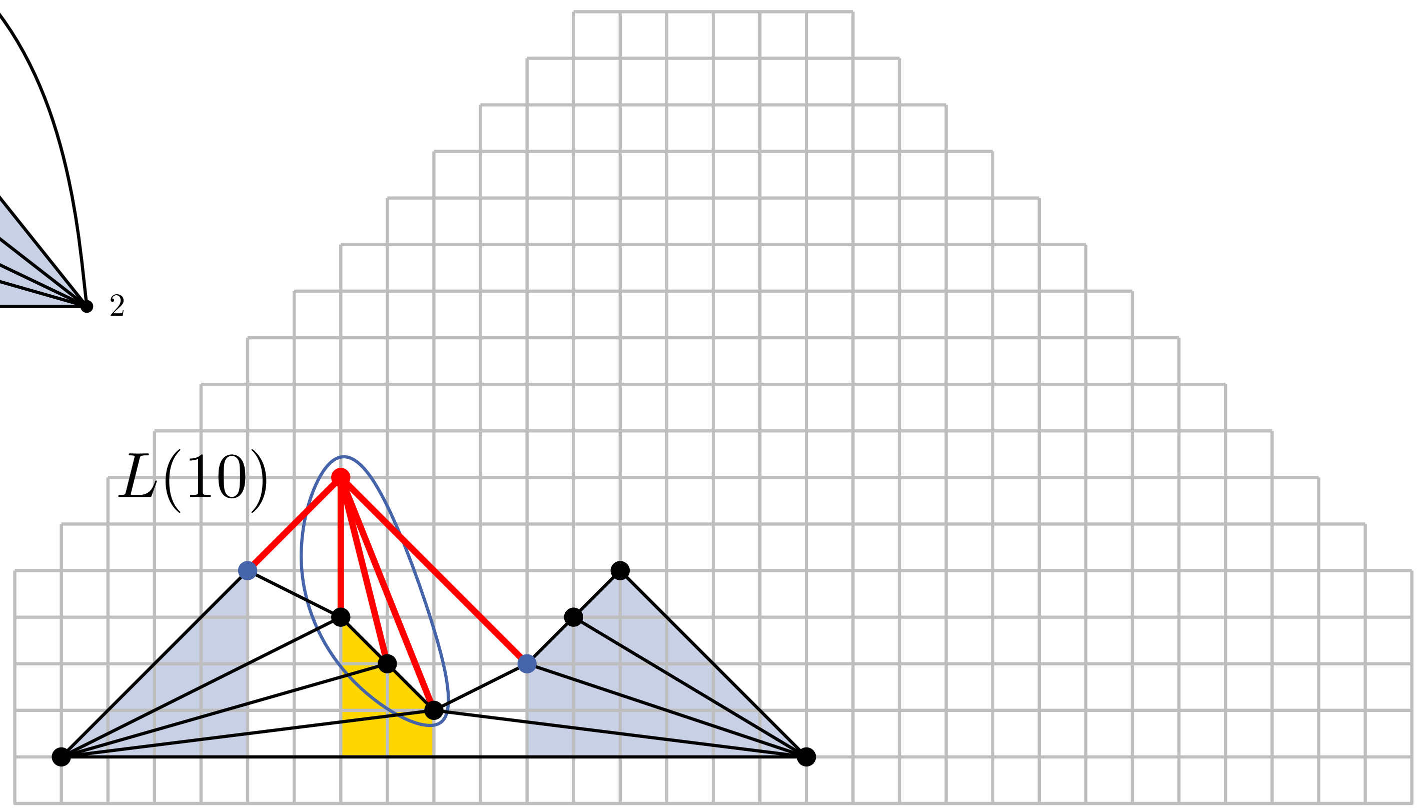




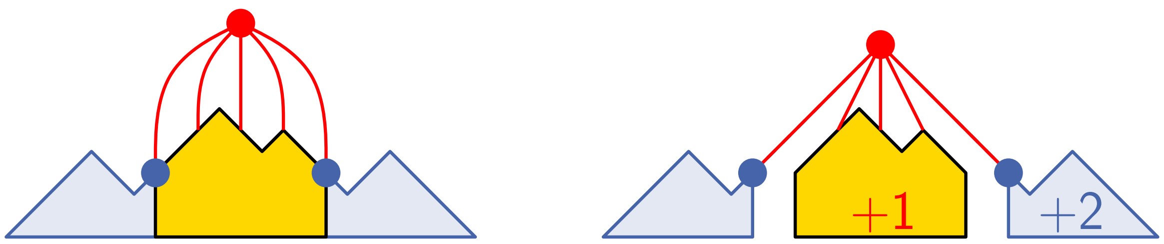
Shift Algorithm: Example

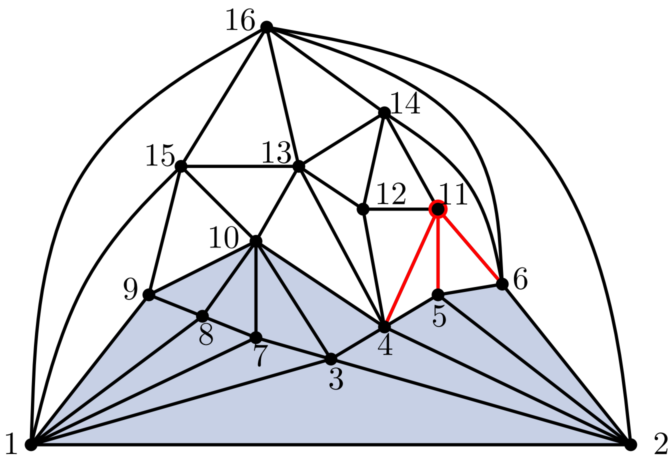


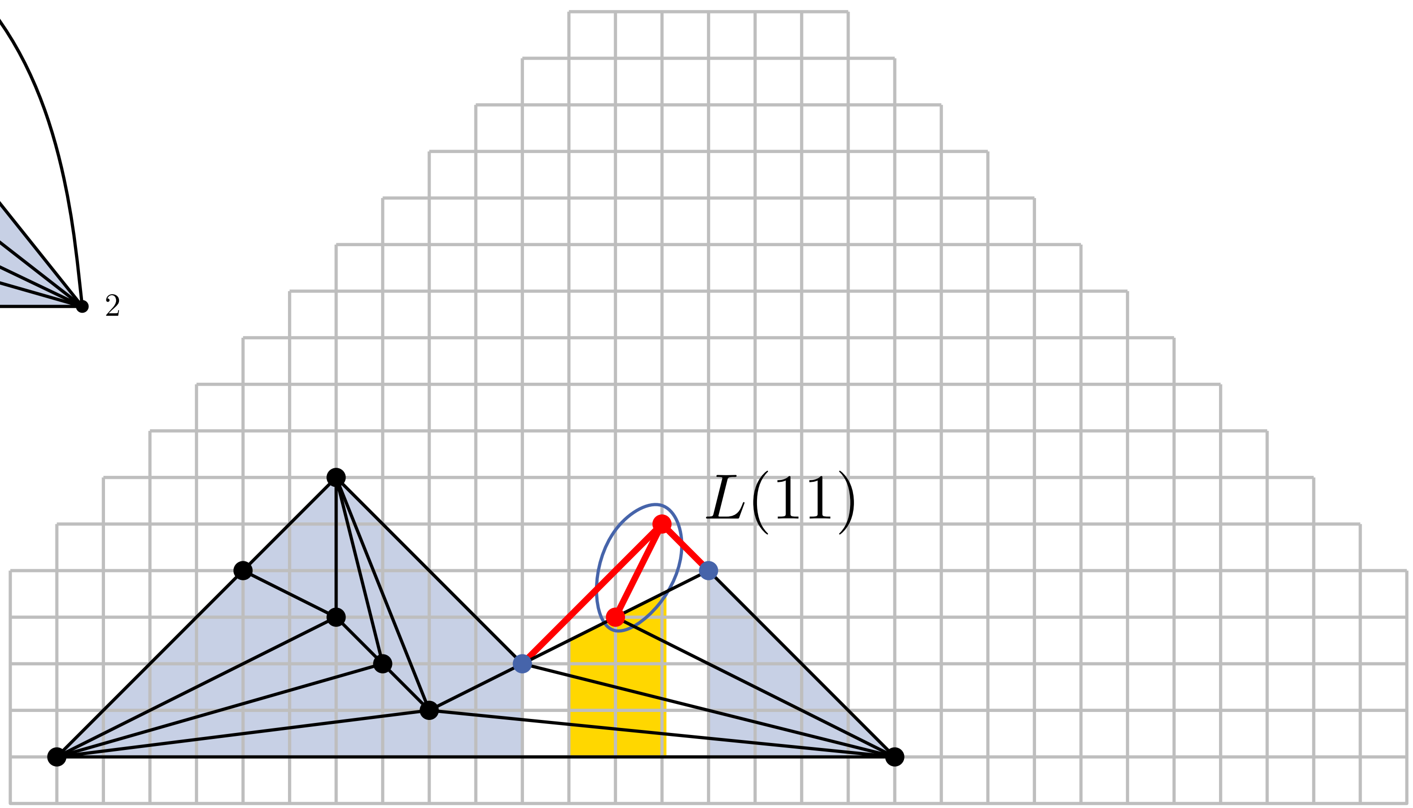




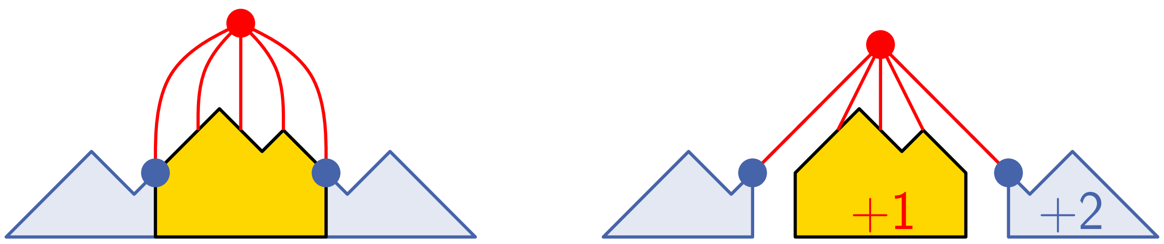
Shift Algorithm: Example

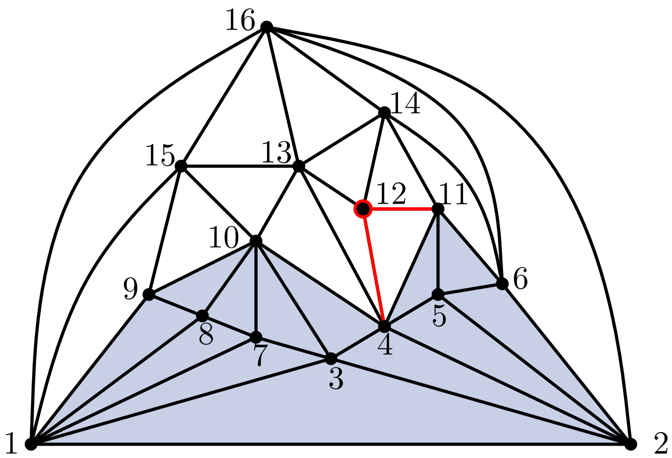


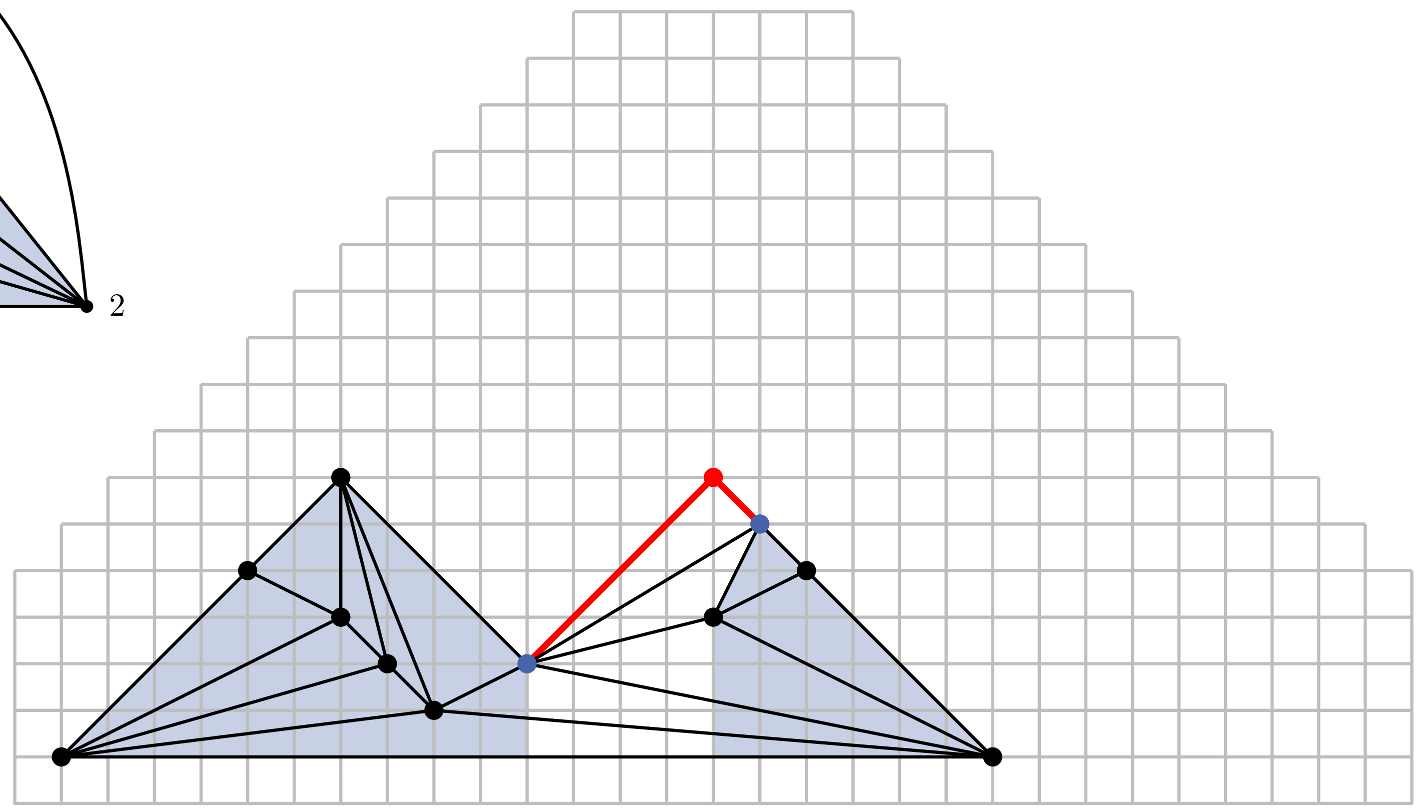




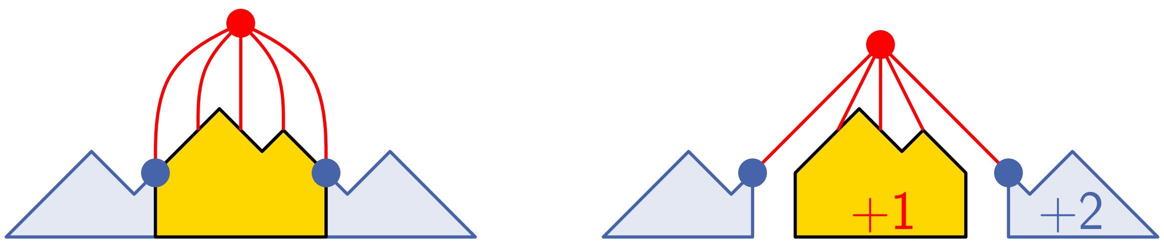
Shift Algorithm: Example

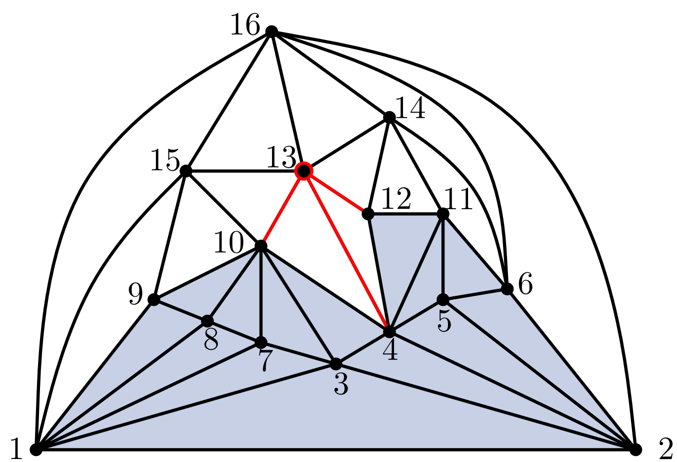


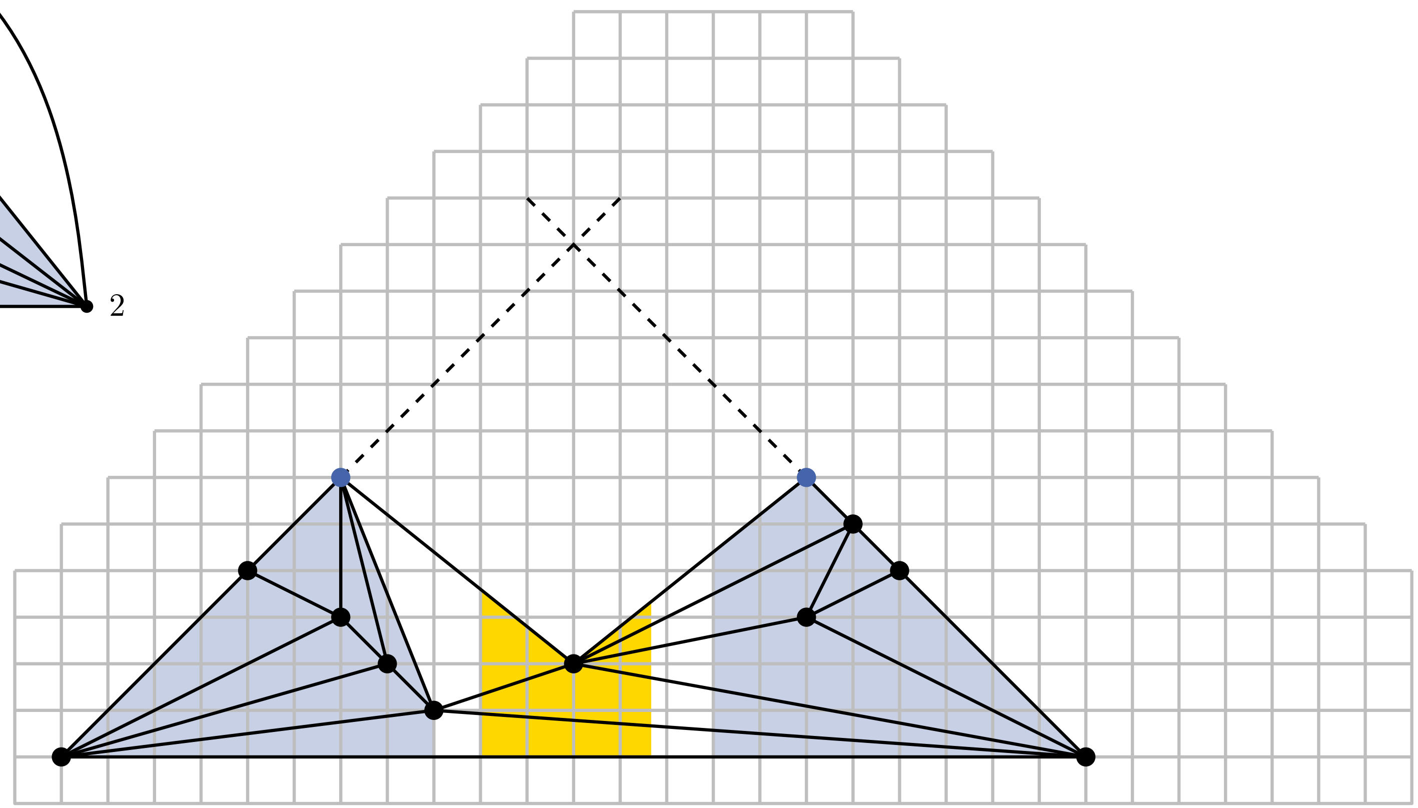




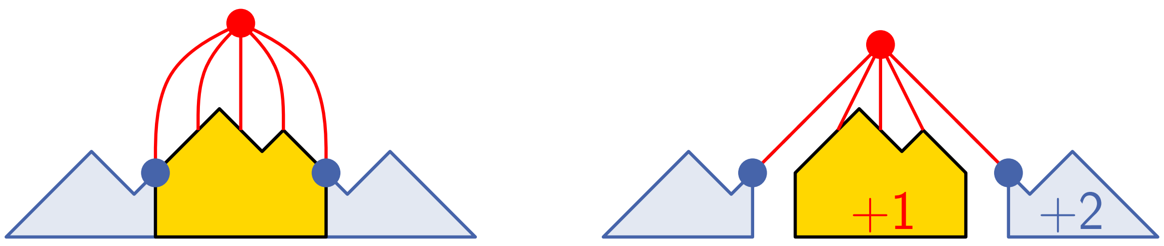
Shift Algorithm: Example

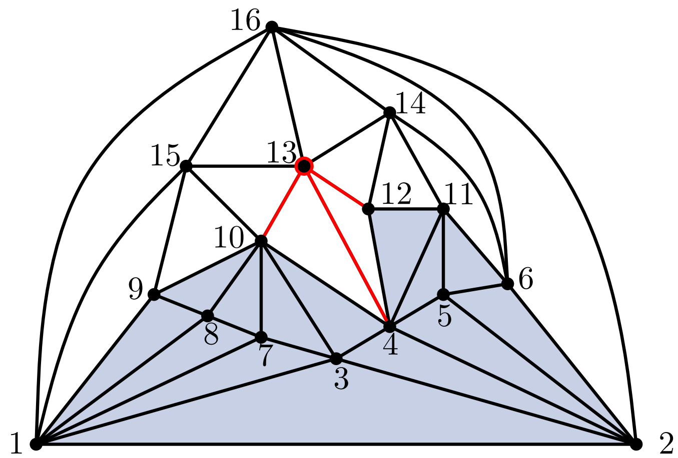


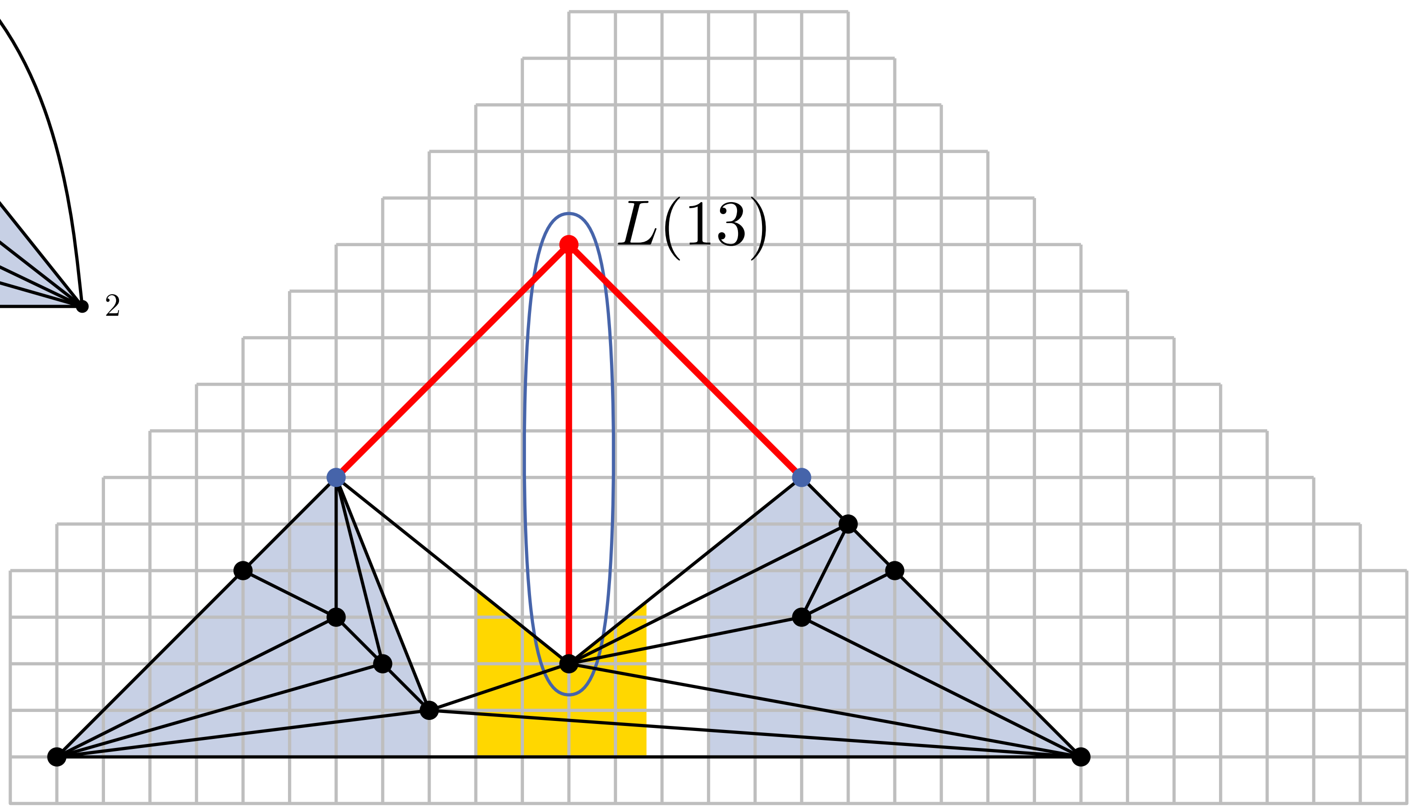




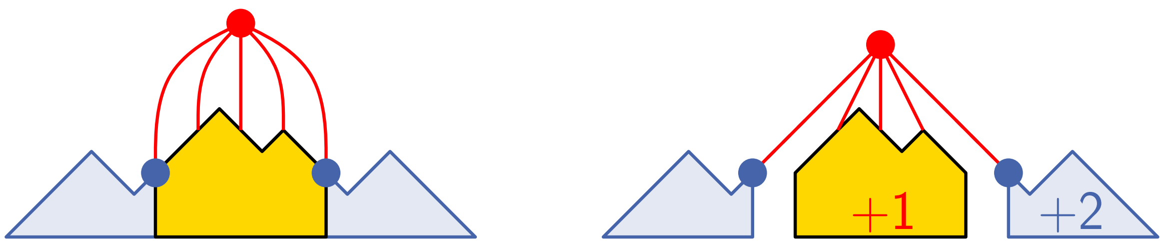
Shift Algorithm: Example

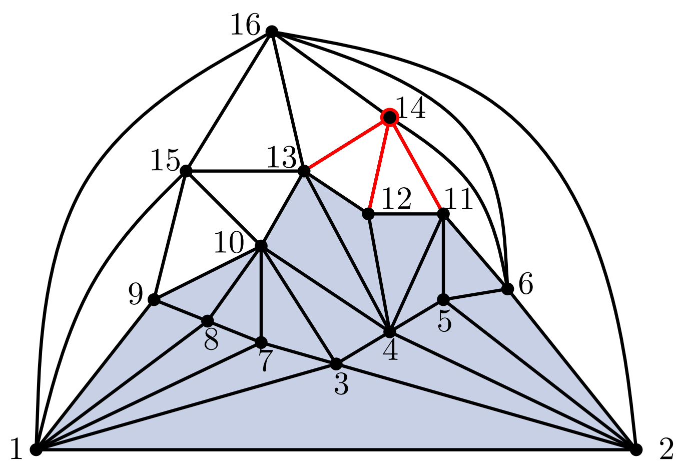


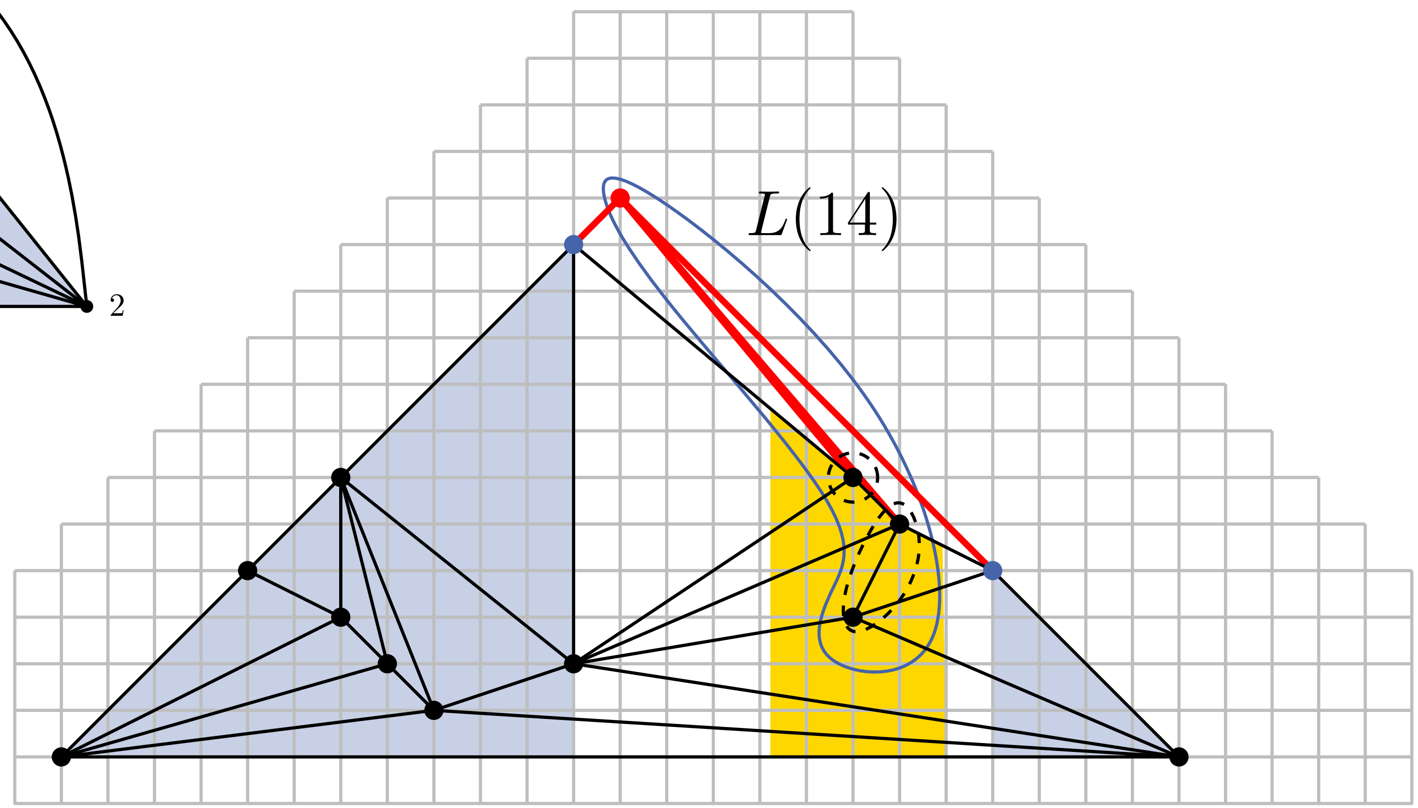




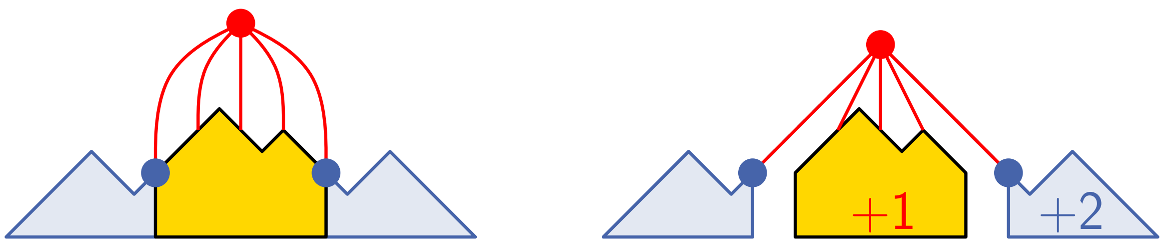
Shift Algorithm: Example

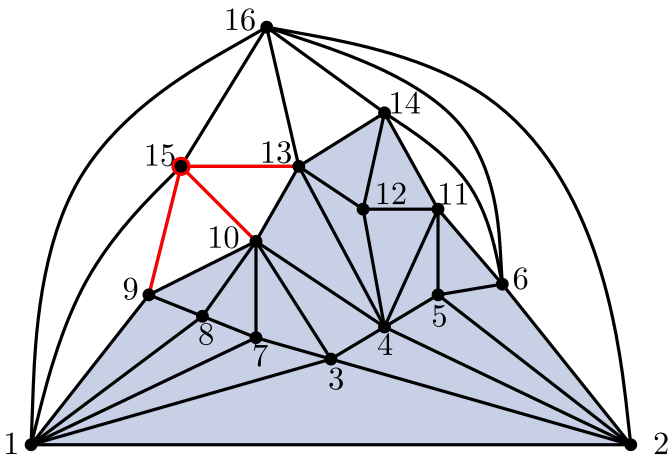


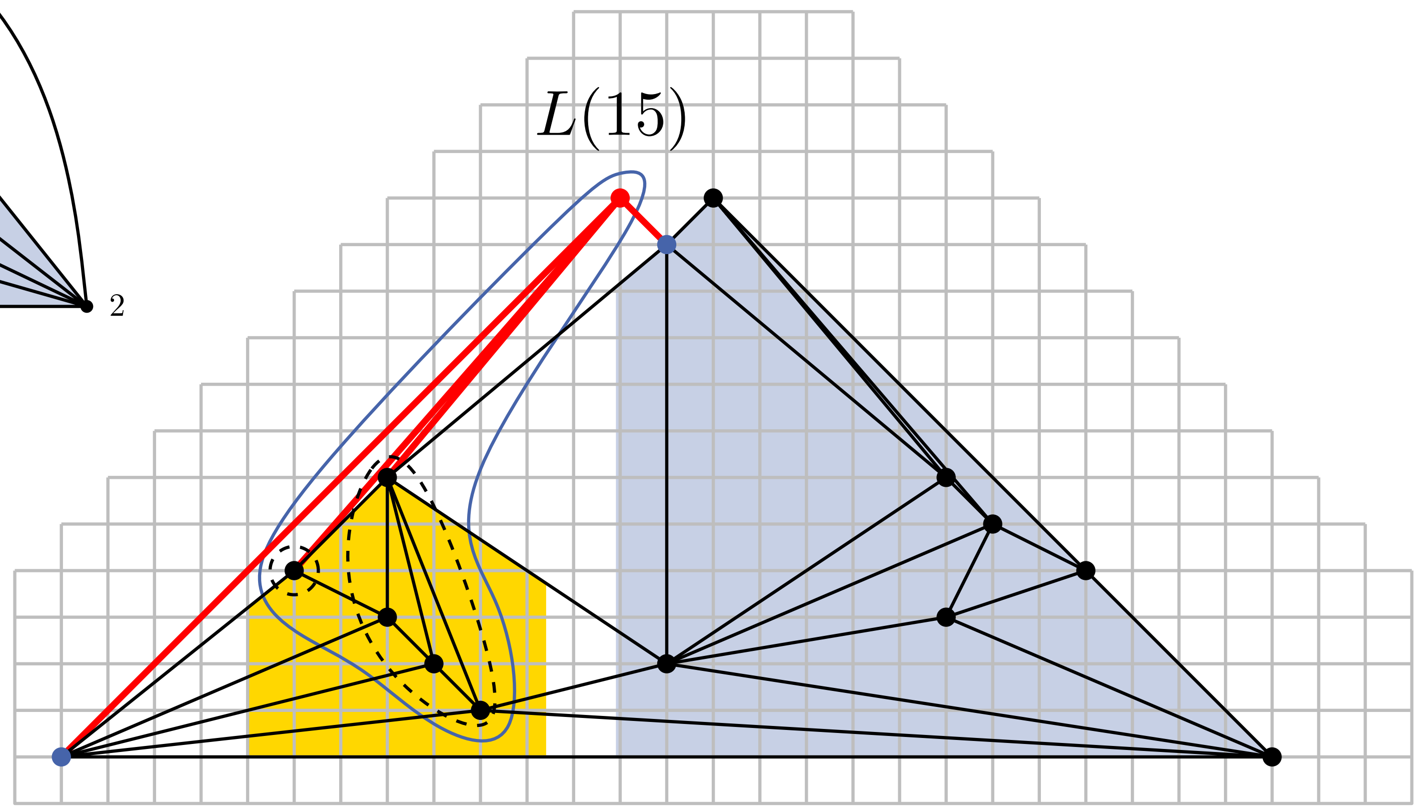




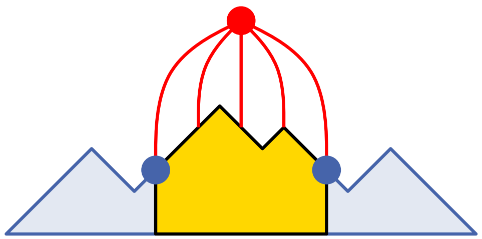
Shift Algorithm: Example

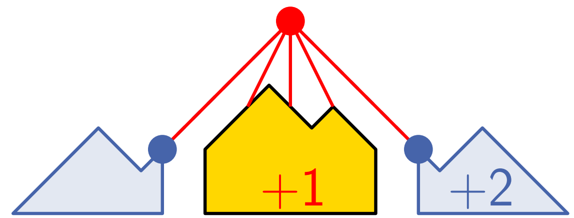


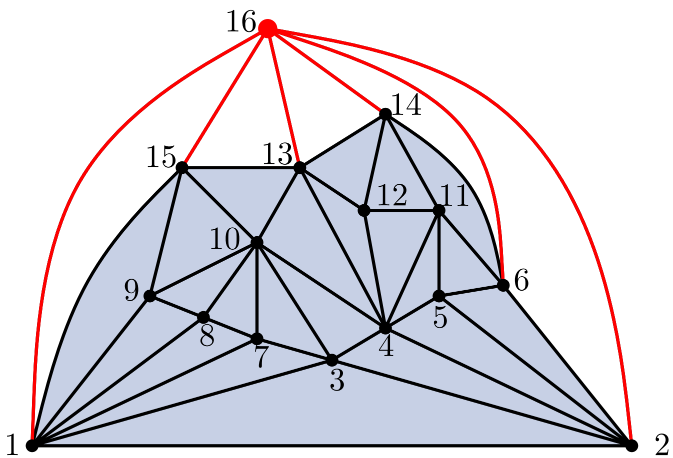


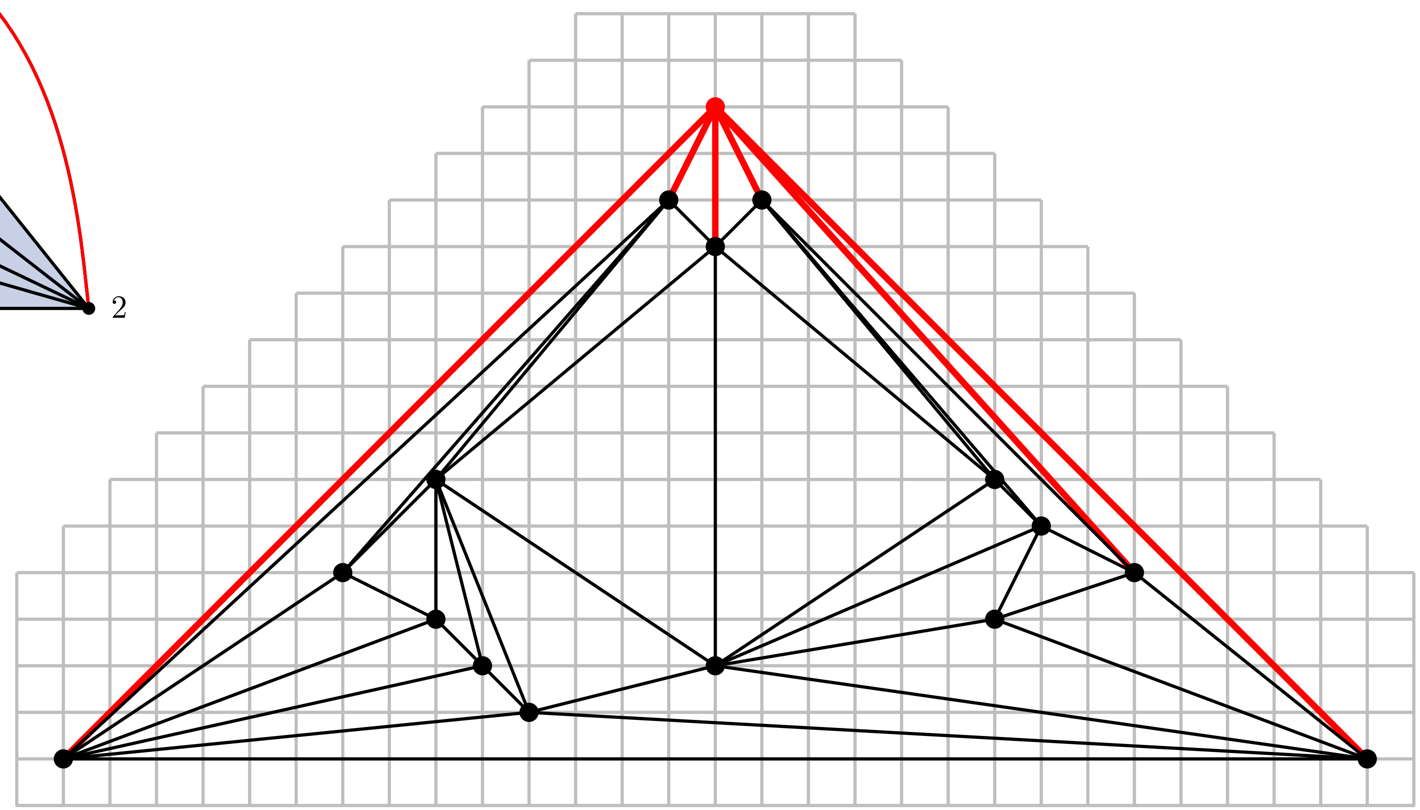


Shift Algorithm: Example

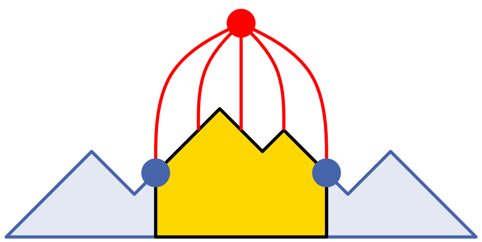


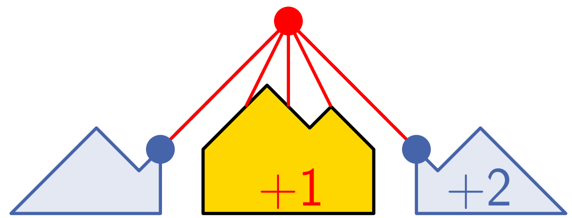


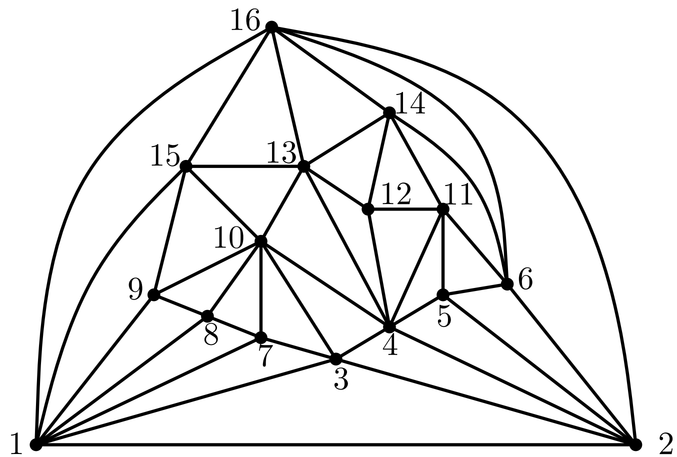


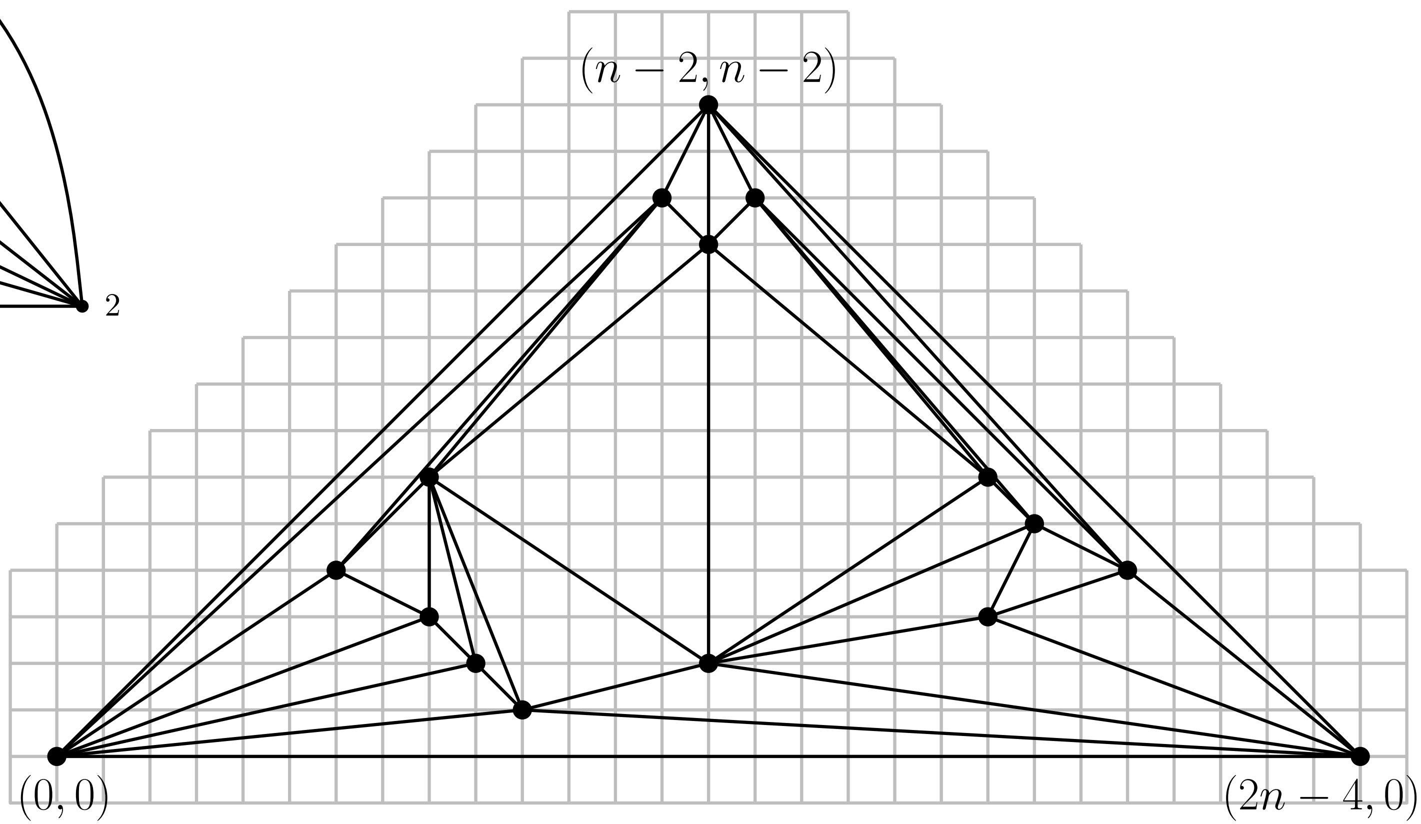


Shift Algorithm: Example

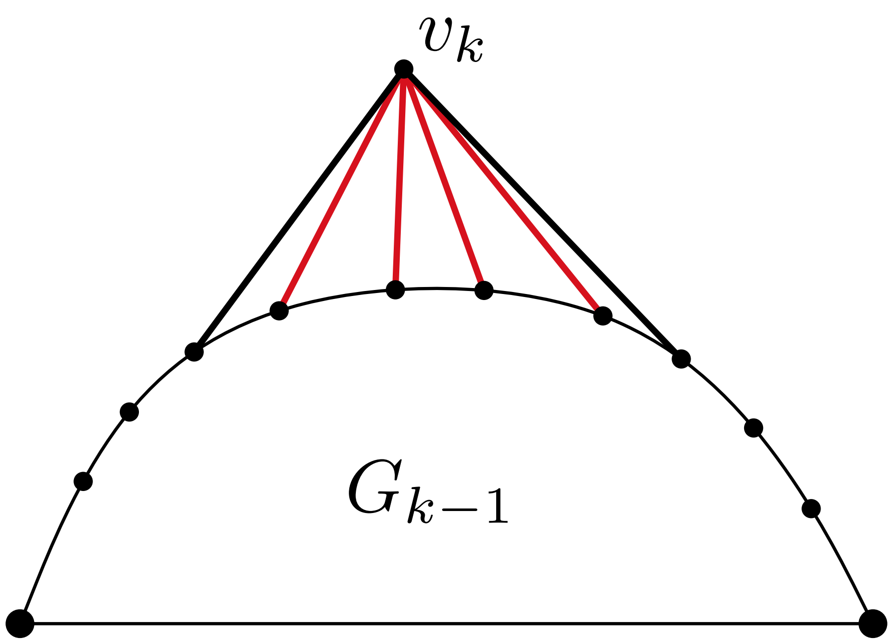




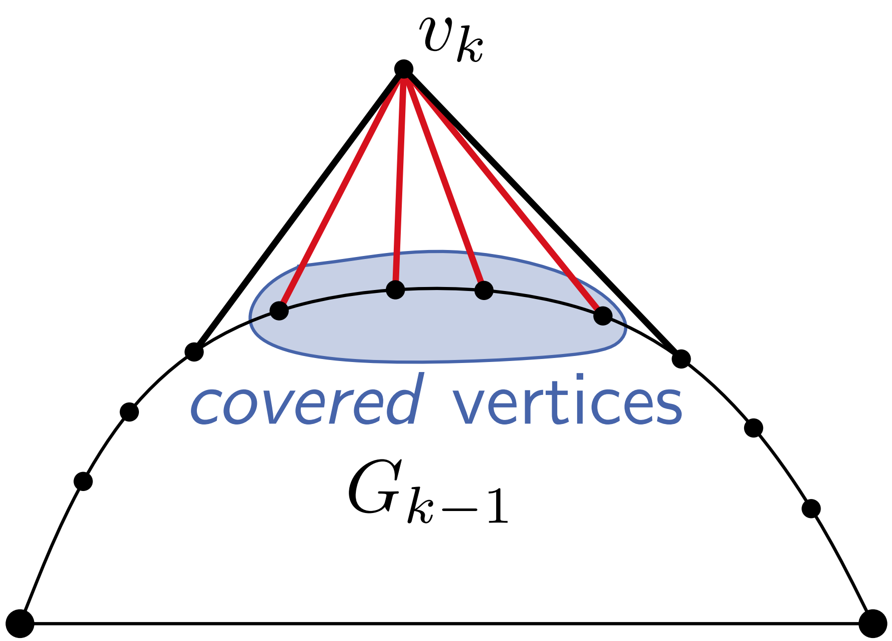




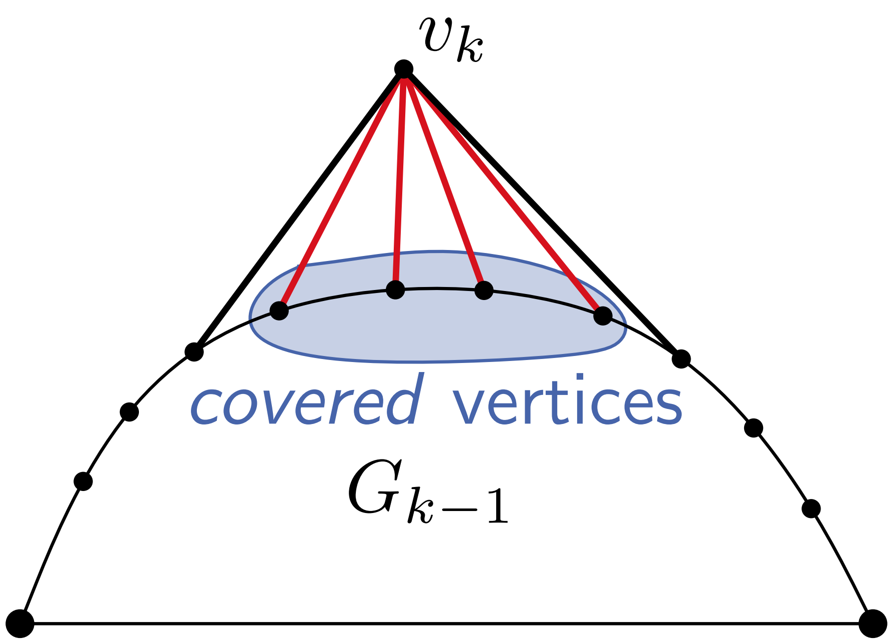
Proof of Planarity



Proof of Planarity

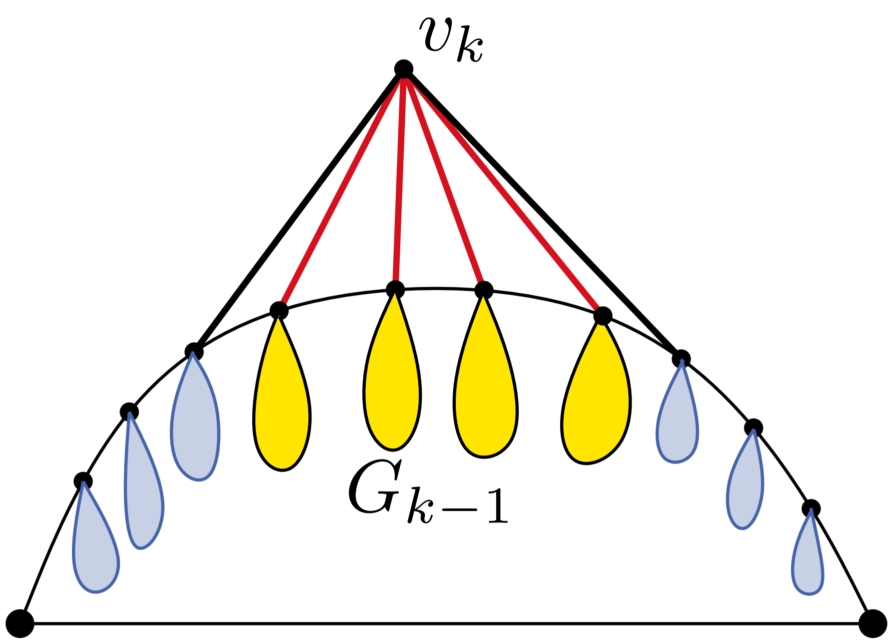


Proof of Planarity



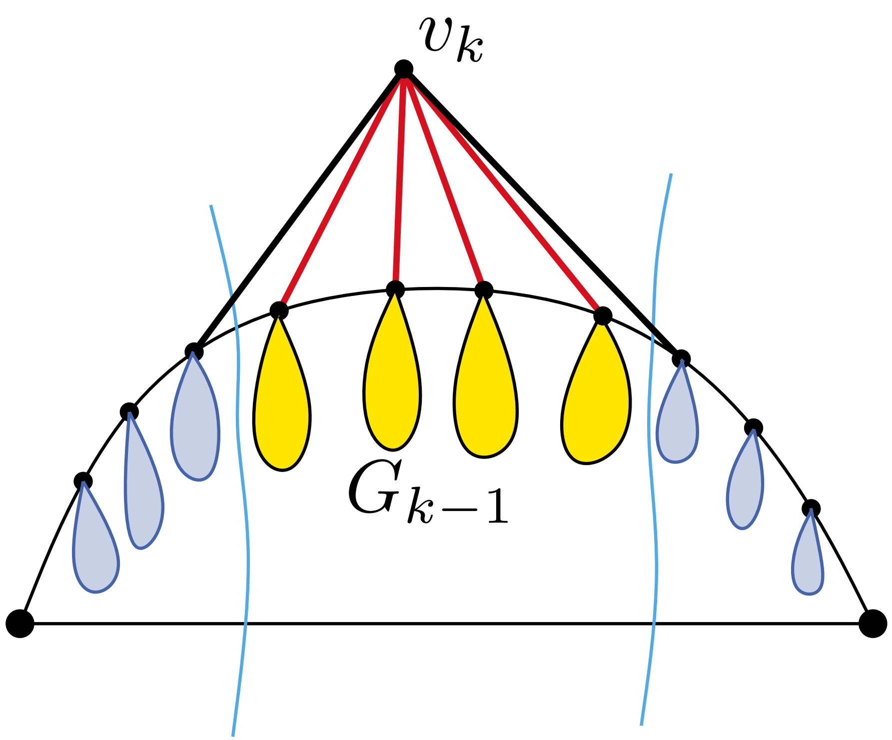
- each internal vertex is covered exactly once
- covering relation defines a tree in G
- forest in G_i, $3 \leq i \leq n-1$

Proof of Planarity



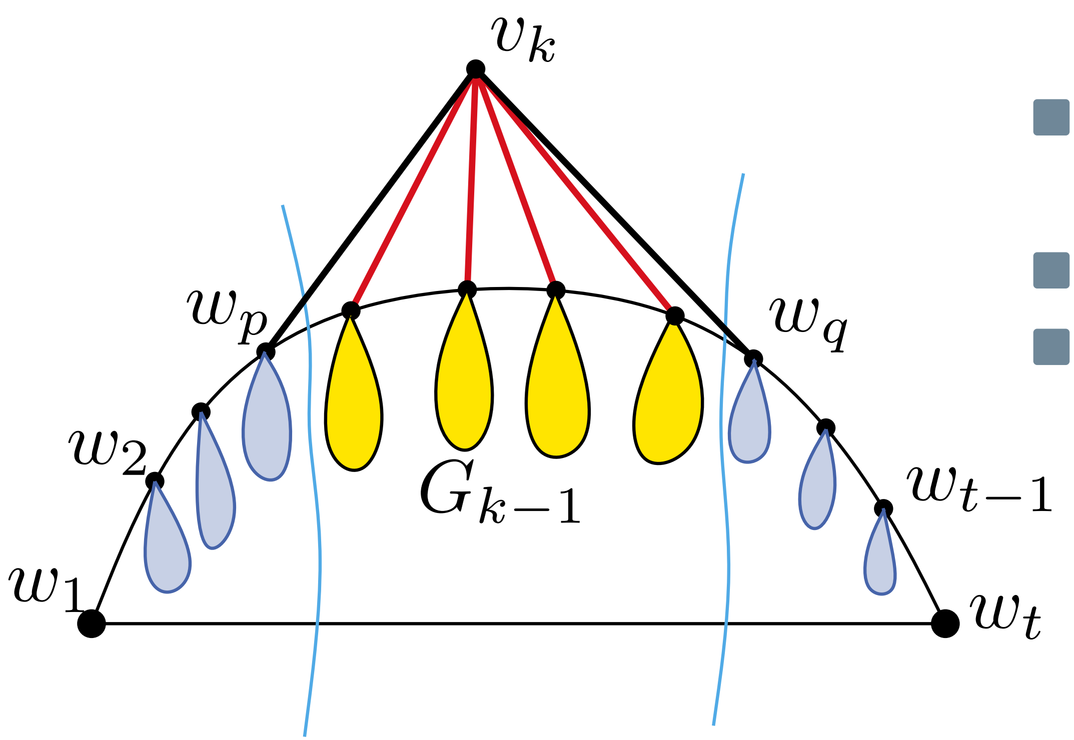
- each internal vertex is covered exactly once
- covering relation defines a tree in G
- forest in G_i, $3 \leq i \leq n-1$

Proof of Planarity



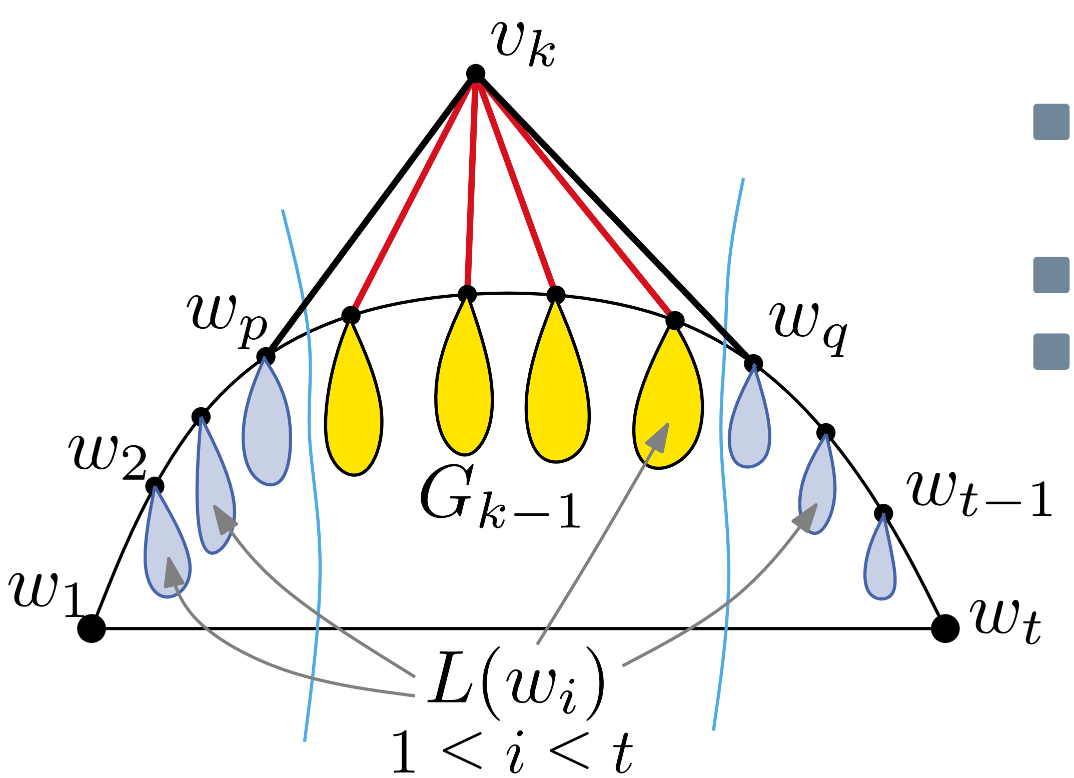
- each internal vertex is covered exactly once
- covering relation defines a tree in G
- forest in G_i, $3 \leq i \leq n-1$

Proof of Planarity



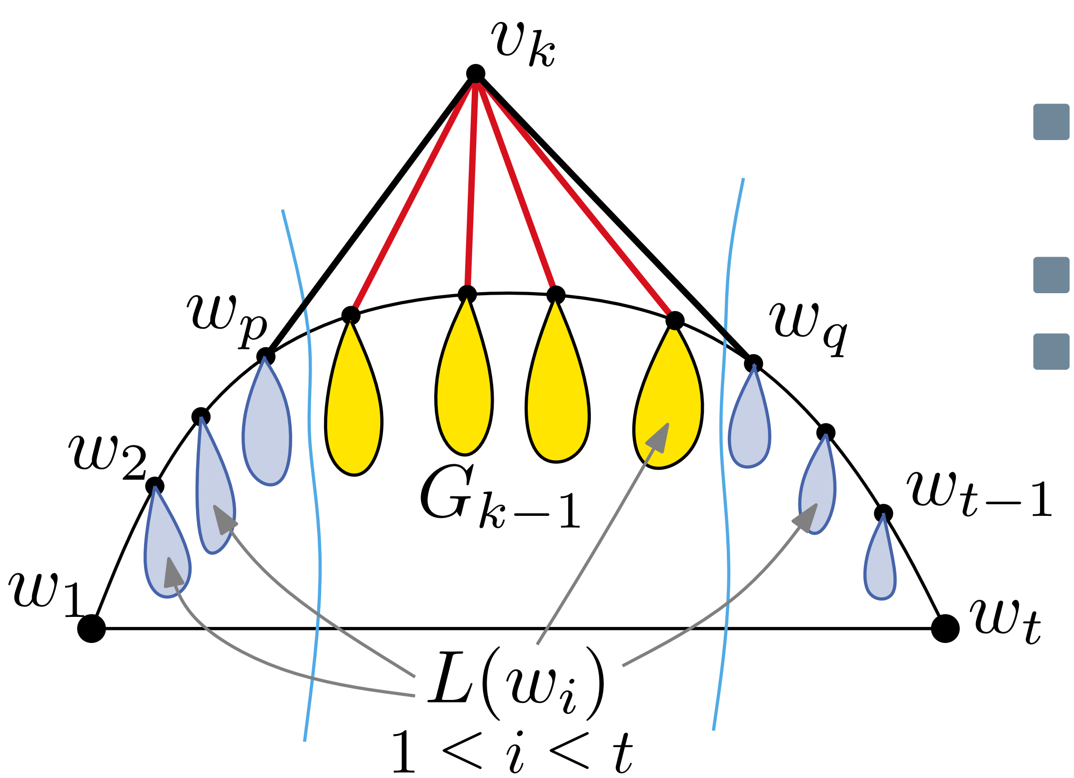
- each internal vertex is covered exactly once
- covering relation defines a tree in G
- forest in G_i, $3 \leq i \leq n-1$

Proof of Planarity



- each internal vertex is covered exactly once
- covering relation defines a tree in G
- forest in G_i, $3 \leq i \leq n-1$

Proof of Planarity



- each internal vertex is covered exactly once
- covering relation defines a tree in G
- forest in G_i, $3 \leq i \leq n-1$

Lemma: Let G_k be a planar straight-line grid drawing. Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift each $L(w_i)$ by δ_i to the right, we get another planar straight-line grid drawing.

Proof of Planarity

Lemma: Let G_k be a planar straight-line grid drawing. Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift each $L(w_i)$ by δ_i to the right, we get another planar straight-line grid drawing.

Proof of Planarity

Lemma: Let G_k be a planar straight-line grid drawing. Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift each $L(w_i)$ by δ_i to the right, we get another planar straight-line grid drawing.

Proof

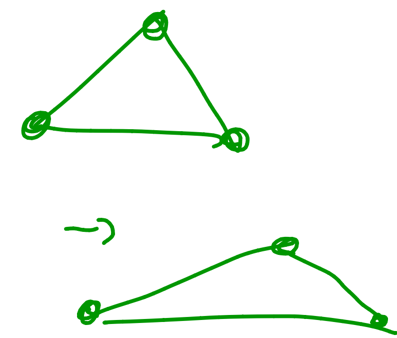
- induction on i, i.e. we consider $G_3, \ldots, G_n$

Proof of Planarity

Lemma: Let G_k be a planar straight-line grid drawing. Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$. If we shift each $L(w_i)$ by δ_i to the right, we get another planar straight-line grid drawing.

Proof

- induction on i, i.e. we consider $G_3, \dots, G_n$
- clearly true for G_3; assume the lemma holds for G_{k-1}

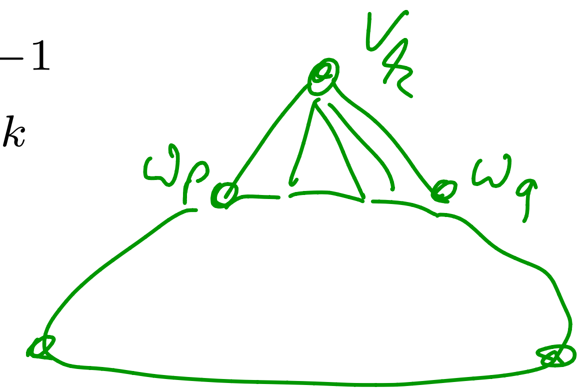


Proof of Planarity

Lemma: Let G_k be a planar straight-line grid drawing. Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift each $L(w_i)$ by δ_i to the right, we get another planar straight-line grid drawing.

Proof

- induction on i, i.e. we consider $G_3, \ldots, G_n$
- clearly true for G_3; assume the lemma holds for G_{k-1}
- let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of G_k



Proof of Planarity

Lemma: Let G_k be a planar straight-line grid drawing. Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift each $L(w_i)$ by δ_i to the right, we get another planar straight-line grid drawing.

Proof

- induction on i, i.e. we consider $G_3, \ldots, G_n$
- clearly true for G_3; assume the lemma holds for G_{k-1}
- let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of G_k
- let $\delta_1 \leq \cdots \leq \delta_p \leq \delta \leq \delta_q \leq \cdots \leq \delta_t$

Proof of Planarity

Lemma: Let G_k be a planar straight-line grid drawing. Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift each $L(w_i)$ by δ_i to the right, we get another planar straight-line grid drawing.

Proof

- induction on i, i.e. we consider $G_3, \ldots, G_n$
- clearly true for G_3; assume the lemma holds for G_{k-1}
- let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of G_k
- let $\delta_1 \leq \cdots \leq \delta_p \leq \delta \leq \delta_q \leq \cdots \leq \delta_t$
- define $\delta'_i = \begin{cases} \delta_i & \text{for } 1 \leq i \leq p \\ \delta + 1 & \text{for } p+1 \leq i \leq q-1 \text{ (covered vertices of } v_k\text{)} \\ \delta_i + 2 & \text{for } q \leq i \leq t \end{cases}$

Proof of Planarity

Lemma: Let G_k be a planar straight-line grid drawing. Let $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$. If we shift each $L(w_i)$ by δ_i to the right, we get another planar straight-line grid drawing.

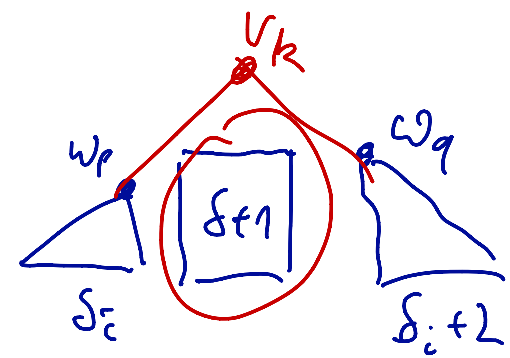
Proof

- induction on i, i.e. we consider $G_3, \ldots, G_n$
- clearly true for G_3; assume the lemma holds for G_{k-1}
- let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of G_k
- let $\delta_1 \leq \cdots \leq \delta_p \leq \delta \leq \delta_q \leq \cdots \leq \delta_t$
- define $\delta'_i = \begin{cases} \delta_i & \text{for } 1 \leq i \leq p \\ \delta + 1 & \text{for } p+1 \leq i \leq q-1 \text{ (covered vertices of } v_k\text{)} \\ \delta_i + 2 & \text{for } q \leq i \leq t \end{cases}$
- by induction hypothesis we can move each $L(w_1), \ldots, L(w_t)$ by $\delta'_1, \ldots, \delta'_t$, respectively

Proof of Planarity

Lemma: Let G_k be a planar straight-line grid drawing. Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$. If we shift each $L(w_i)$ by δ_i to the right, we get another planar straight-line grid drawing.

Proof



- induction on i, i.e. we consider $G_3, \dots, G_n$
- clearly true for G_3; assume the lemma holds for G_{k-1}
- let $w_1, \dots, w_p, v_k, w_q, \dots, w_t$ be the boundary of G_k
- let $\delta_1 \leq \dots \leq \delta_p \leq \delta \leq \delta_q \leq \dots \leq \delta_t$
- define $\delta'_i = \begin{cases} \delta_i & \text{for } 1 \leq i \leq p \\ \delta + 1 & \text{for } p+1 \leq i \leq q-1 \text{ (covered vertices of } v_k\text{)} \\ \delta_i + 2 & \text{for } q \leq i \leq t \end{cases}$
- by induction hypothesis we can move each $L(w_1), \dots, L(w_t)$ by $\delta'_1, \dots, \delta'_t$, respectively
- complete the drawing of G_k by placing v_k; now v_k is moved rigidly with $L(w_{p+1}), \dots, L(w_{q-1})$ by δ

Overview

Canonical ordering

Shift algorithm

Implementation

Naïve Implementation

$\texttt{Shift}$(plane graph $G = (V, E)$)

let $v_1, \ldots, v_n$ canonical ordering of G
for $i = 1$ **to** n **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 $k = 4$ **to** n **do**
- let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ be vertices of C_{k-1}
- let $w_p, \ldots, w_q$ be neighbors of v_k
- **for** $v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do** $x(v) \leftarrow x(v) + 1$
- **for** $v \in \cup_{j=q}^{t} L(w_j)$ **do** $x(v) \leftarrow x(v) + 2$
- $P(v_k) \leftarrow$ intersection point of lines with slope ± 1 from $P(w_p)$ and $P(w_q)$
- $L(v_k) \leftarrow \cup_{j=p+1}^{q-1} L(w_j) \cup \{v_{\cancel{i}\,k}\}$

Naïve Implementation

$\texttt{Shift}$(plane graph $G = (V, E)$)

let $v_1, \ldots, v_n$ canonical ordering of G
for $i = 1$ **to** n **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 $k = 4$ **to** n **do**
  let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ be vertices of C_{k-1}
  let $w_p, \ldots, w_q$ be neighbors of v_k
  for $v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do** $x(v) \leftarrow x(v) + 1$
  for $v \in \cup_{j=q}^{t} L(w_j)$ **do** $x(v) \leftarrow x(v) + 2$
  $P(v_k) \leftarrow$ intersection point of lines with slope ± 1 from $P(w_p)$ and $P(w_q)$
  $L(v_k) \leftarrow \cup_{j=p+1}^{q-1} L(w_j) \cup \{v_{\cancel{i}\,k}\}$

$O(n)$ time

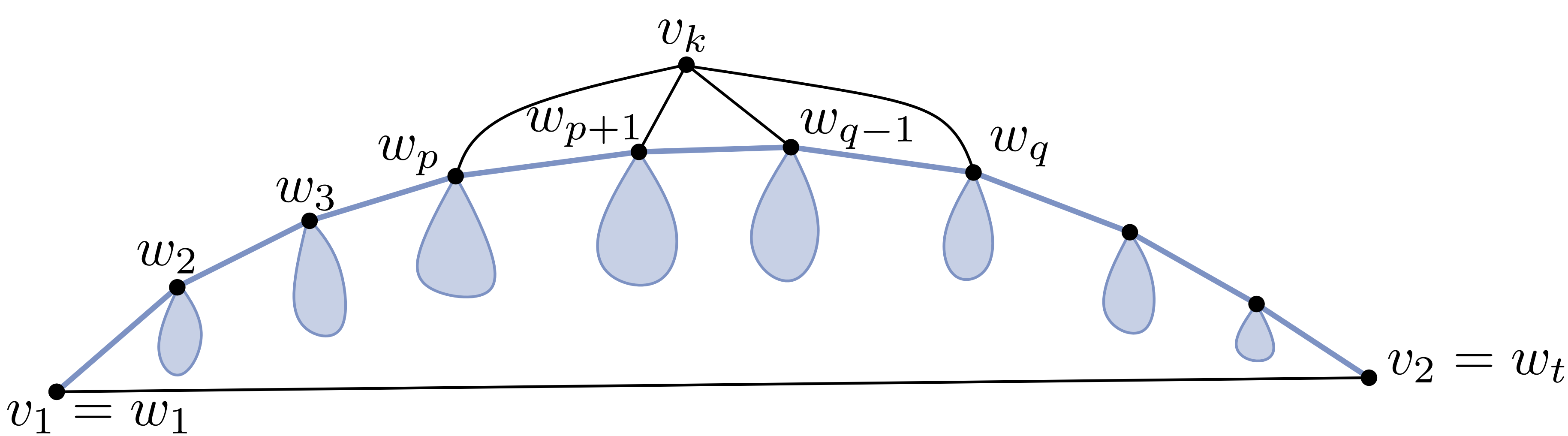
$O(n)$ times

Running time?

$\rightarrow O(n^2)$

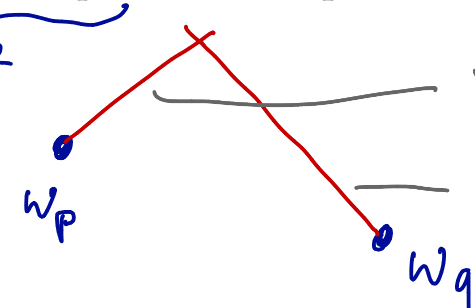
Towards a Faster Implementation

Vertex coordinates



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

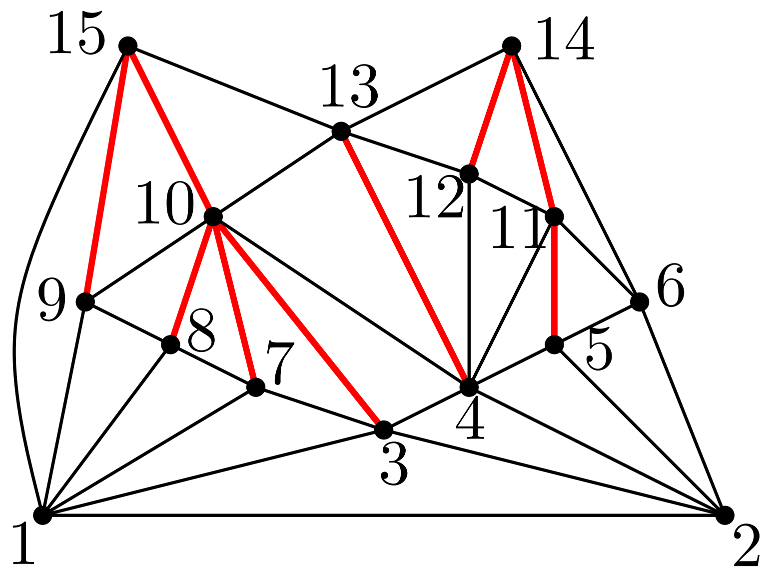
compute intersection point



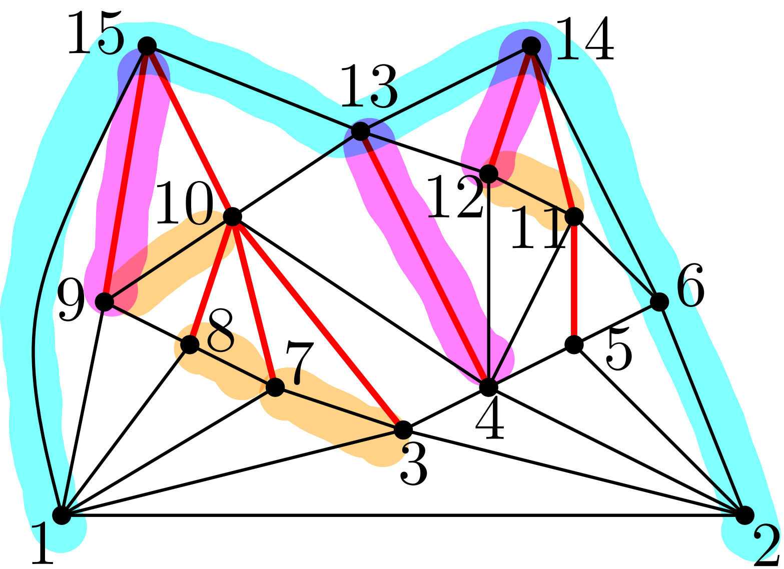
$y = y(w_p) + 1 \cdot (x - x(w_p))$

$y = y(w_q) - 1 \cdot (x - x(w_q))$

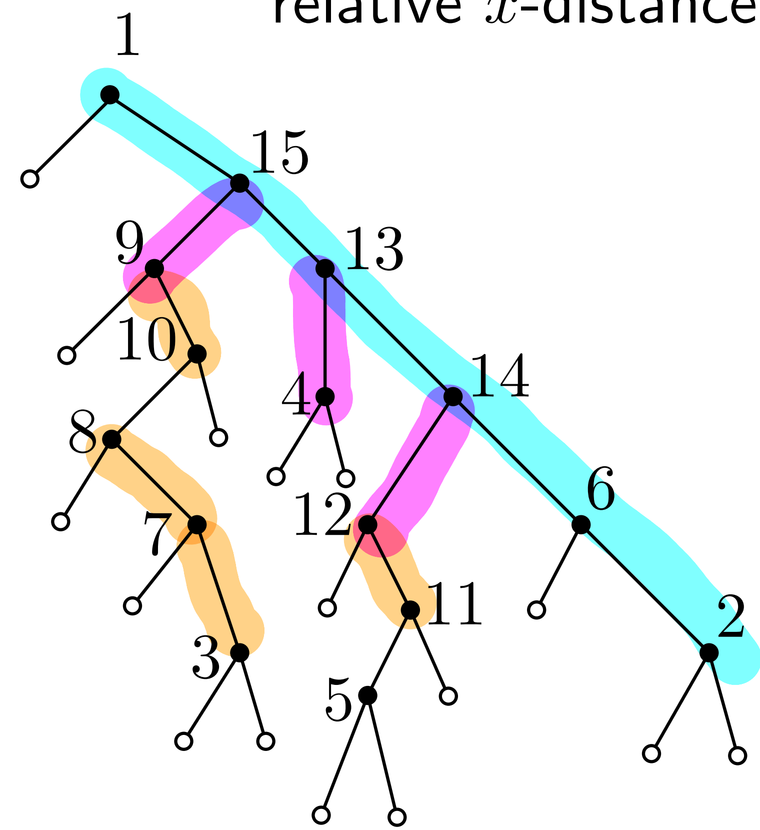
Relative x-Distances



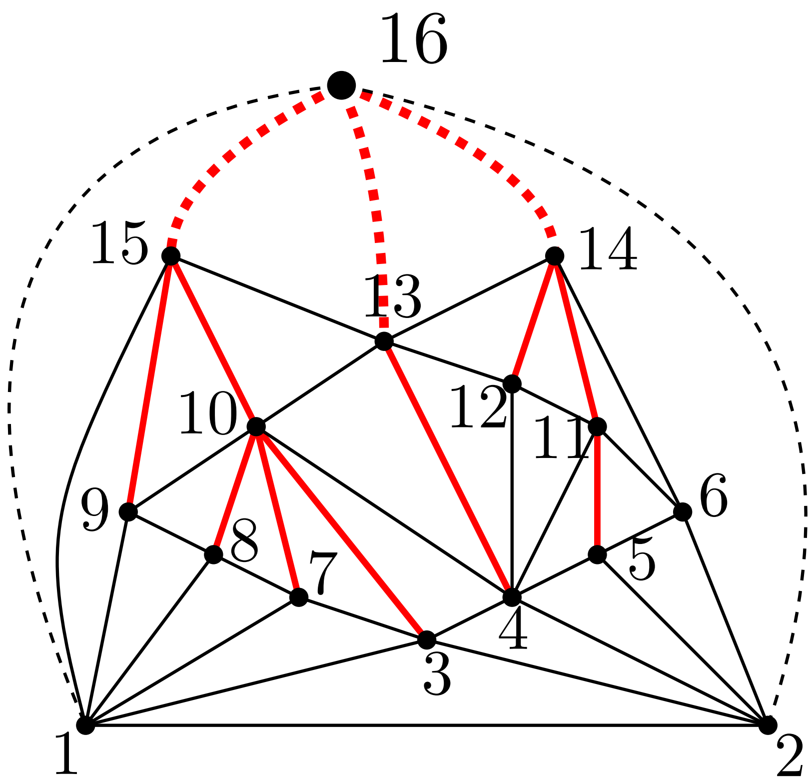
Relative x-Distances

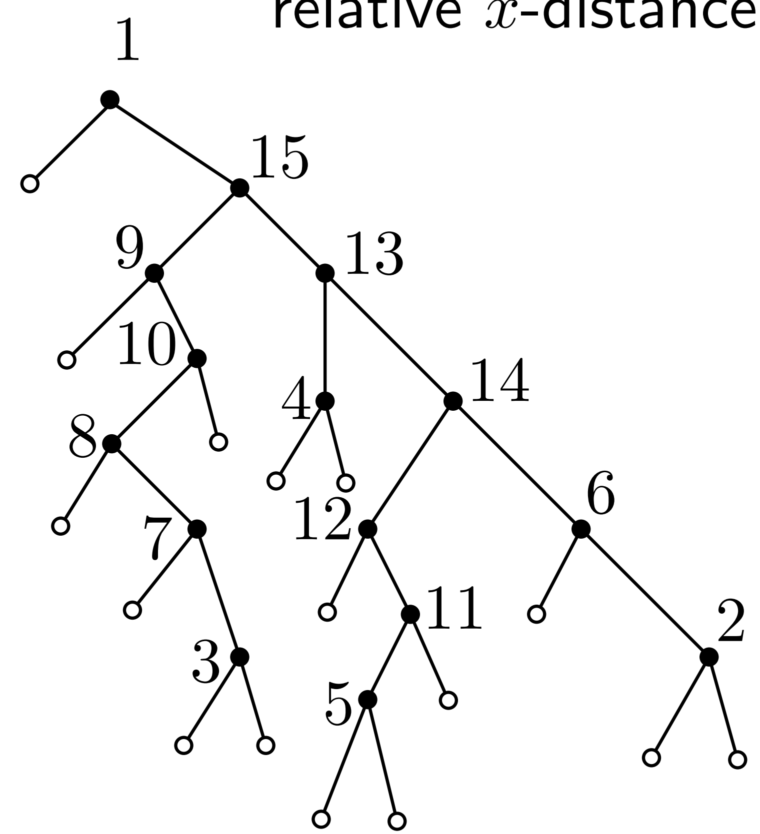


relative x-distance tree T

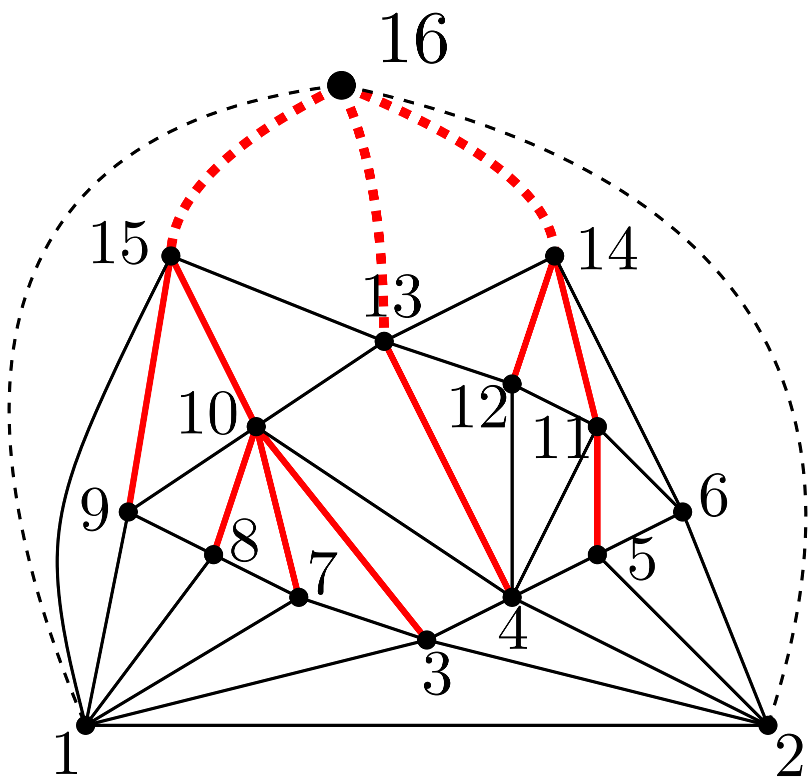


Relative x-Distances

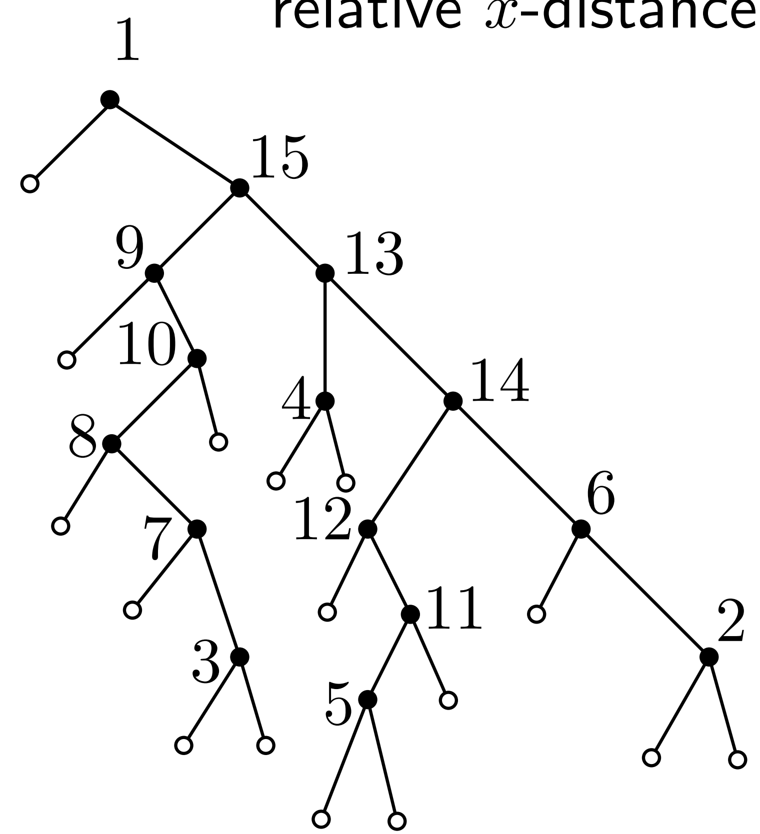




Relative x-Distances

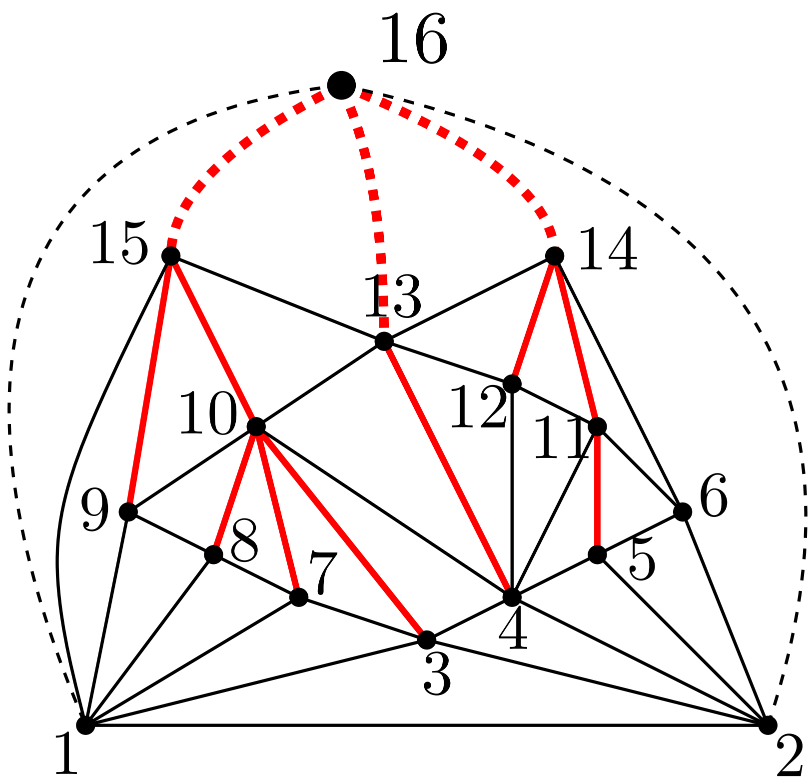


relative x-distance tree T

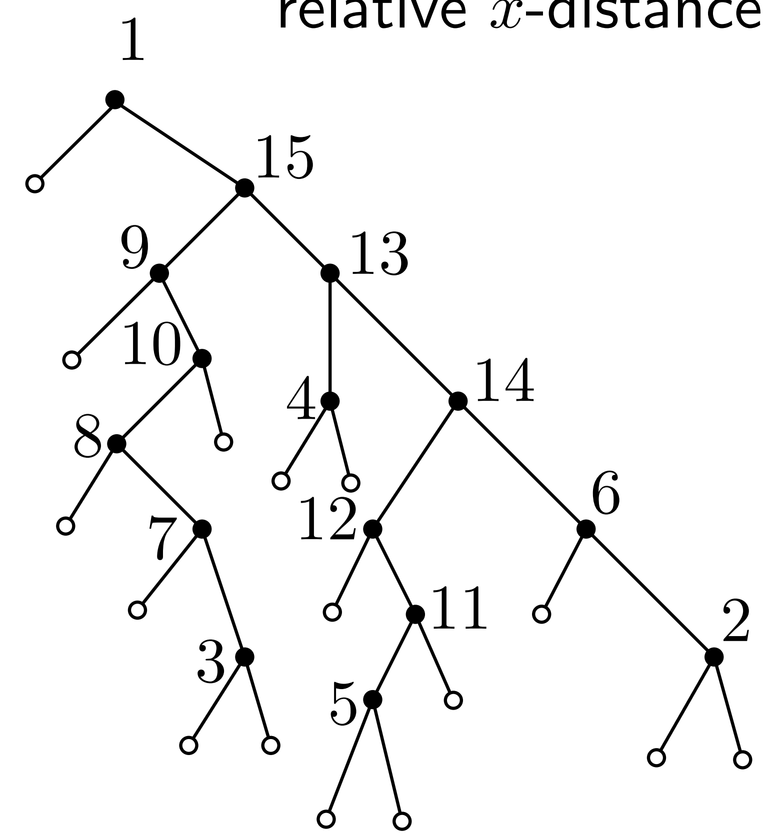


- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate

Relative x-Distances



relative x-distance tree T



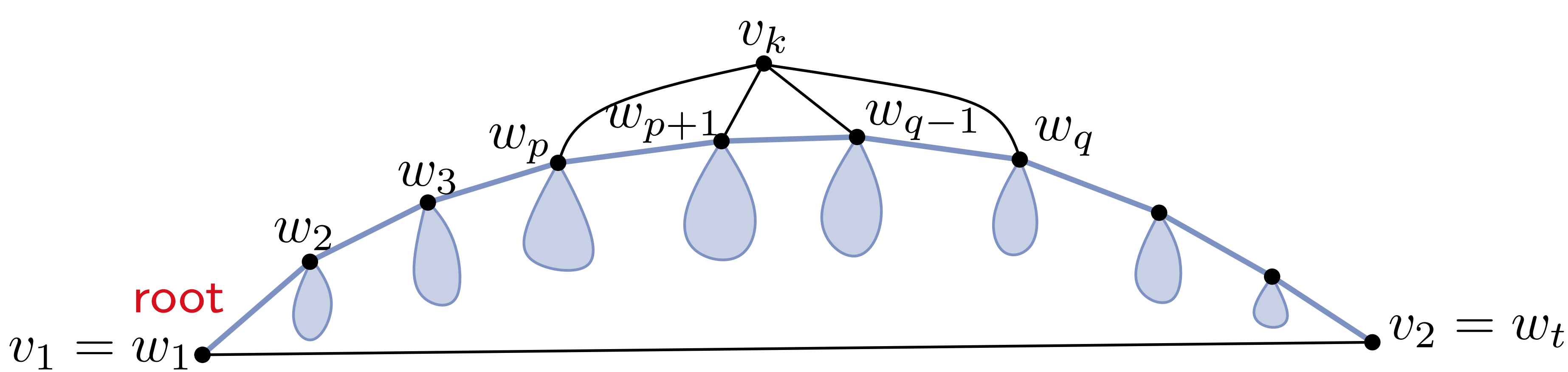
- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]

Linear-Time Implementation

- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]

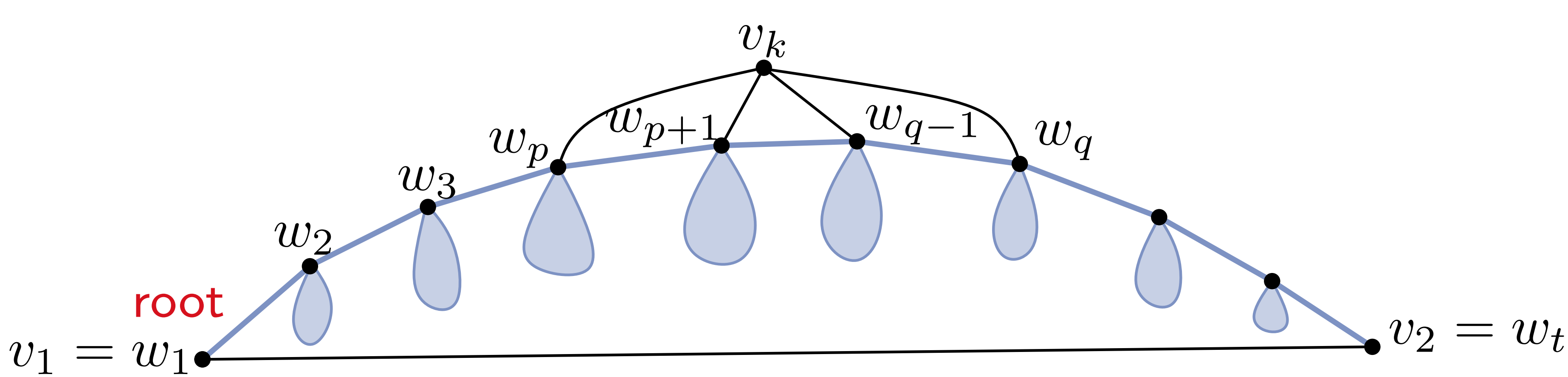
Linear-Time Implementation

- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]



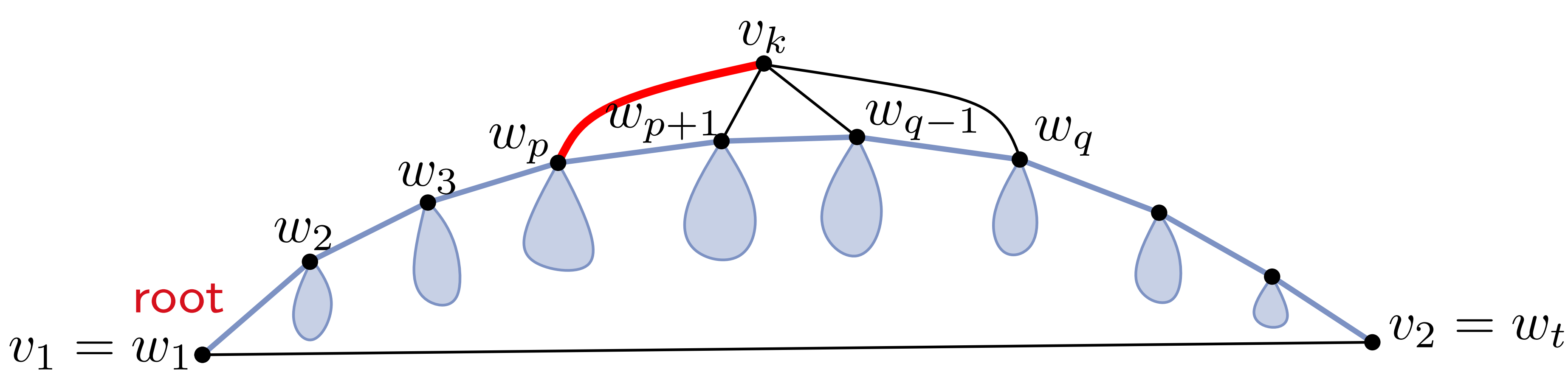
Linear-Time Implementation

- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, where $\Delta_x(w_q)$ is x-distance from the parent and $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



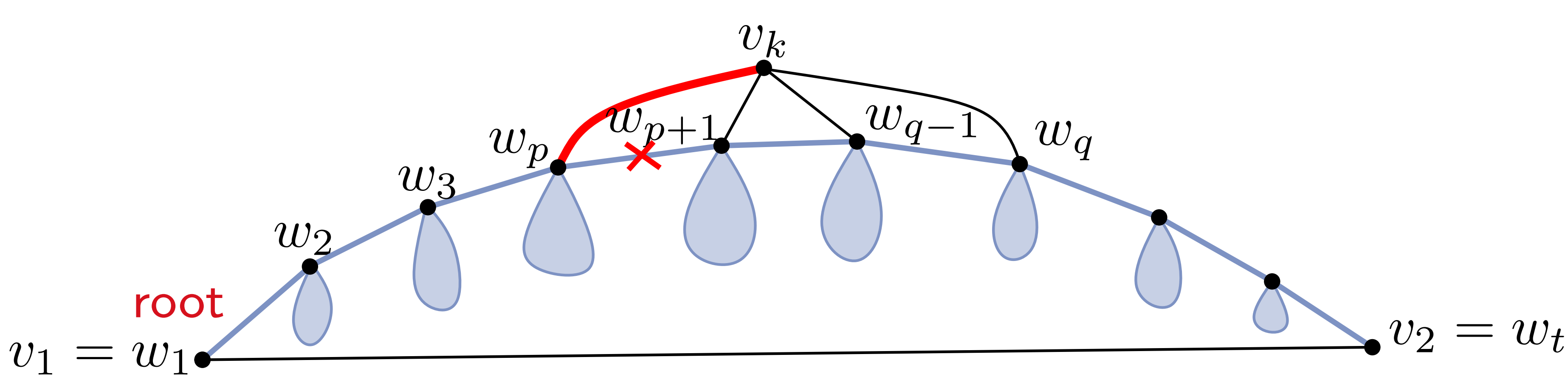
Linear-Time Implementation

- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, where $\Delta_x(w_q)$ is x-distance from the parent and $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



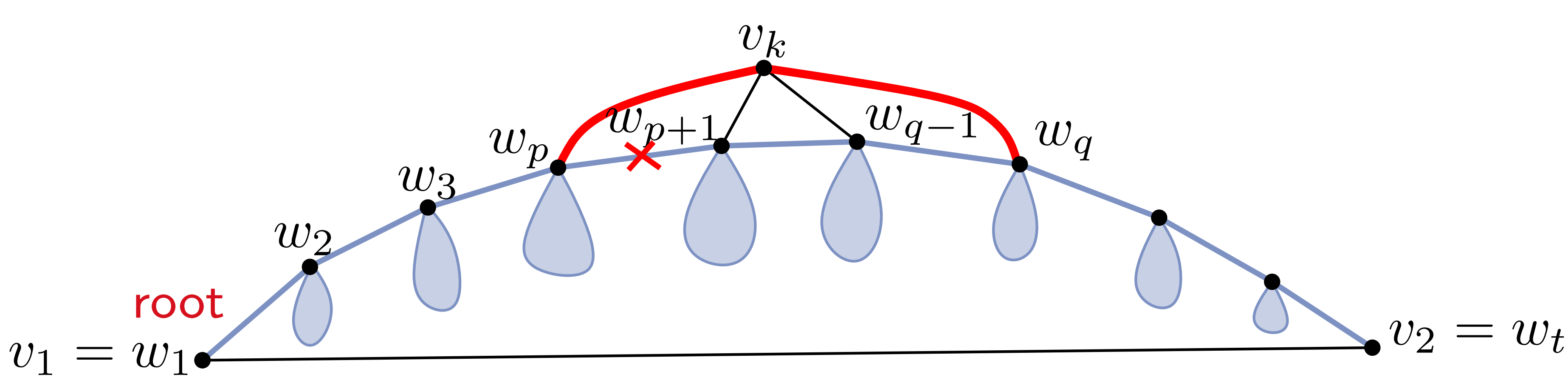
Linear-Time Implementation

- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, where $\Delta_x(w_q)$ is x-distance from the parent and $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



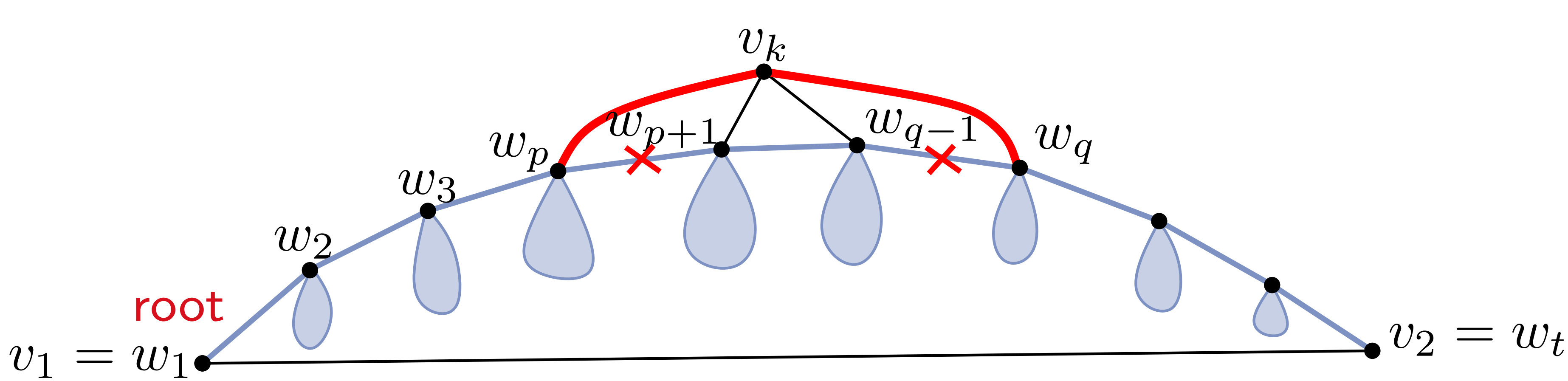
Linear-Time Implementation

- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, where $\Delta_x(w_q)$ is x-distance from the parent and $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



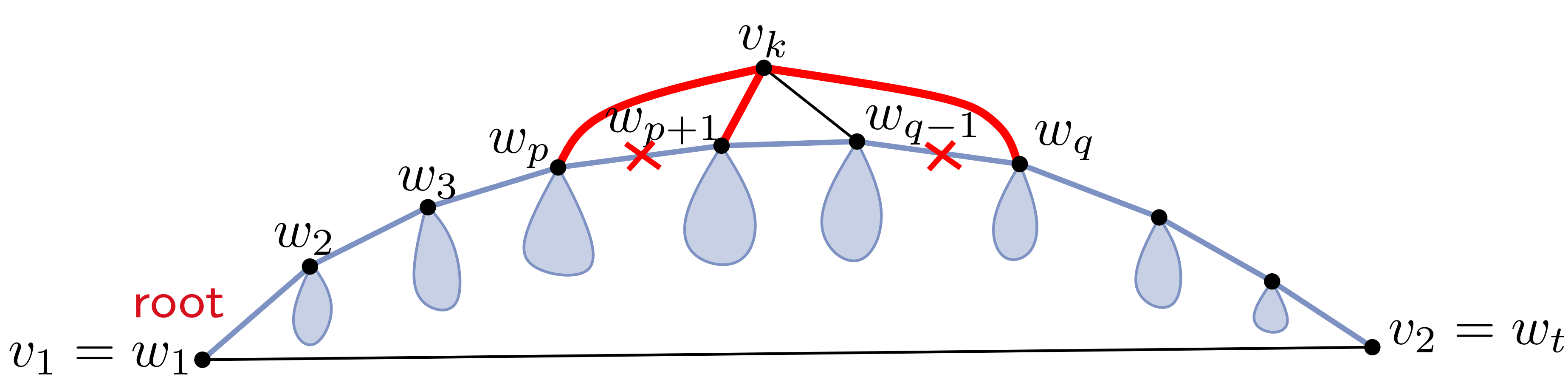
Linear-Time Implementation

- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, where $\Delta_x(w_q)$ is x-distance from the parent and $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



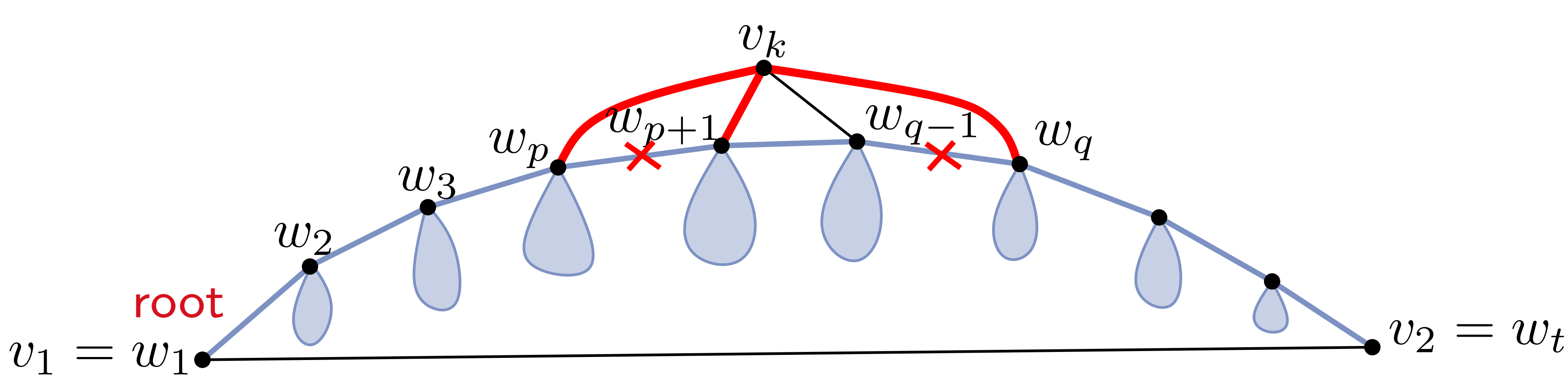
Linear-Time Implementation

- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, where $\Delta_x(w_q)$ is x-distance from the parent and $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



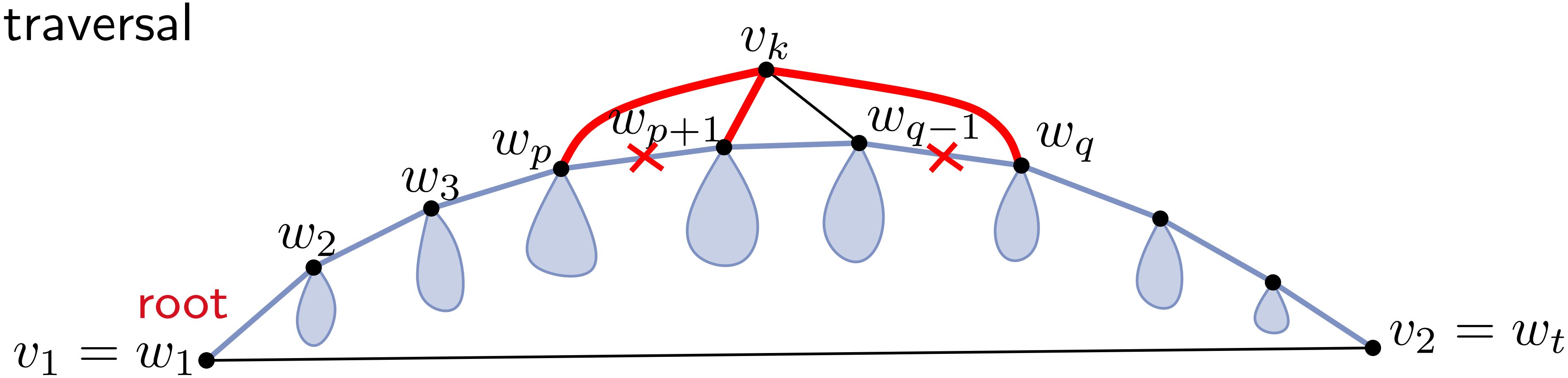
Linear-Time Implementation

- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, where $\Delta_x(w_q)$ is x-distance from the parent and $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)
- $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$
- $\Delta_x(w_{p+1}) = \Delta_x(w_{p+1}) - \Delta_x(v_k)$



Linear-Time Implementation

- in x-distance tree T at each vertex we keep its relative x-distance from its parent and its y-coordinate
- if we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$ [eq. (3)]
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, where $\Delta_x(w_q)$ is x-distance from the parent and $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)
  - $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$
  - $\Delta_x(w_{p+1}) = \Delta_x(w_{p+1}) - \Delta_x(v_k)$
- Once T is completed, compute x-coordinates by a linear-time pre-order traversal



Summary

Theorem: Every n-vertex embedded planar graph $G = (V, E)$ has a straight-line planar drawing on a grid of size $(2n - 4) \times (n - 2)$. [de Fraysseix, Pach, Pollack 1988]

Theorem: The corresponding shift algorithm can be implemented to run in $O(n)$ time. [Chrobak, Payne 1995]