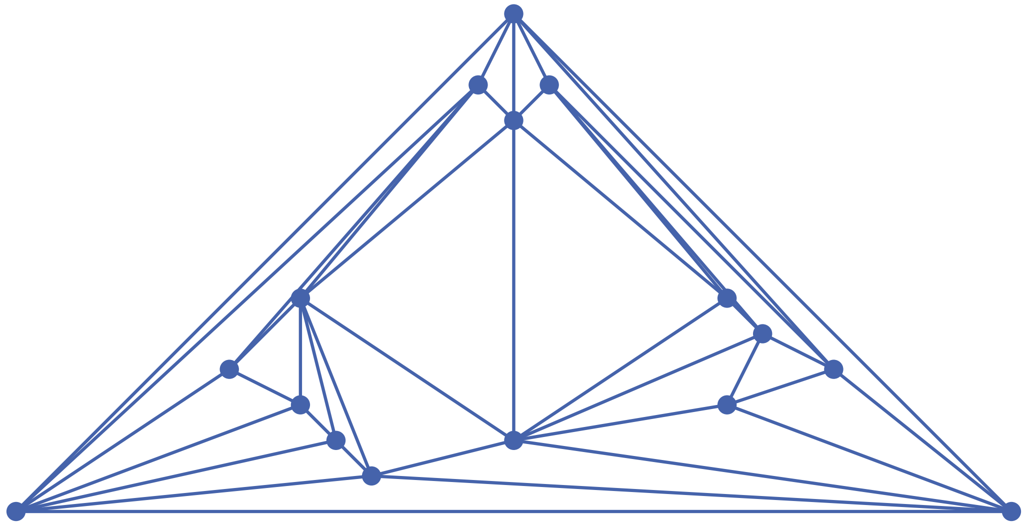
Algorithms for graph visualization

Layouts for planar graphs. Shift method.

Winter semester 2018/2019

Tamara Mchedlidze



KIT – Universität des Landes Baden-Württemberg und
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

Motivation

- Till now we look at planar and straight-line drawings of trees and SP-graphs

Motivation

- Till now we look at planar and straight-line drawings of trees and SP-graphs
- Why straight-line, and Why planar?

Motivation

- Till now we look at planar and straight-line drawings of trees and SP-graphs
- Why straight-line, and Why planar?
- Bennett, Ryall, Spalteholz and Gooch, 2007 "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

- Till now we look at planar and straight-line drawings of trees and SP-graphs
- Why straight-line, and Why planar?
- Bennett, Ryall, Spalteholz and Gooch, 2007 “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

Karlsruhe Institute of Technology

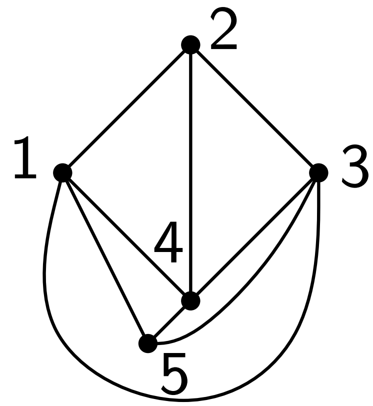
- Till now we look at planar and straight-line drawings of trees and SP-graphs
- Why straight-line, and Why planar?
- Bennett, Ryall, Spalteholz and Gooch, 2007 “The Aesthetics of Graph Visualization”

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.

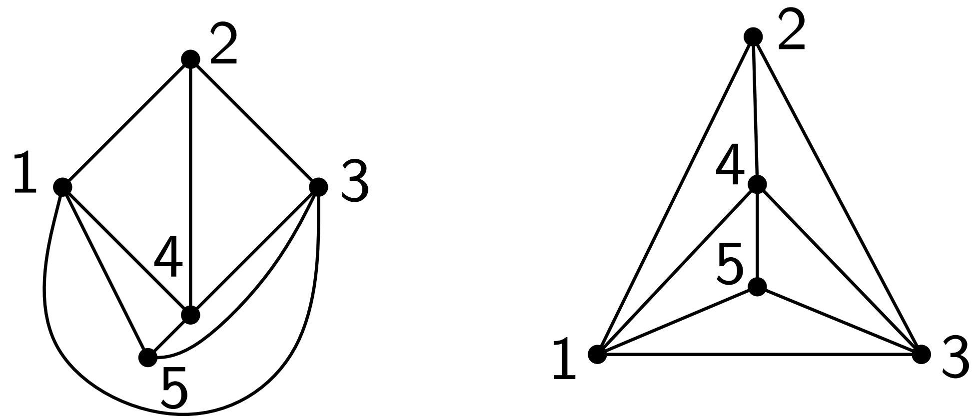
History

- Does every planar graph have a planar straight-line drawing?



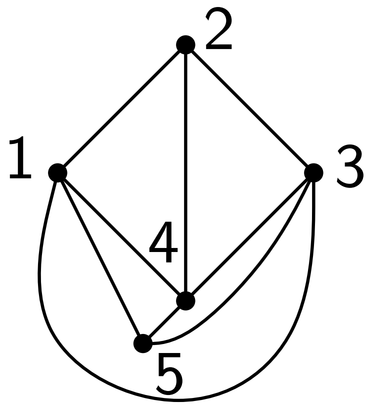
History

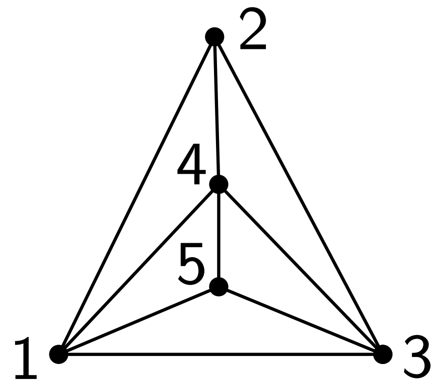
- Does every planar graph have a planar straight-line drawing?



History

- Does every planar graph have a planar straight-line drawing?



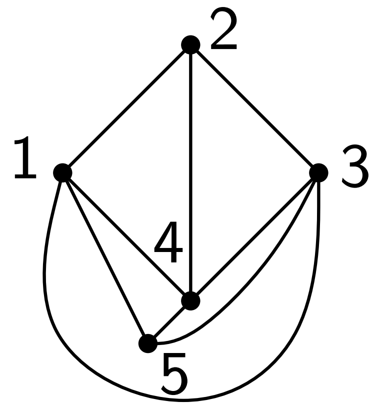


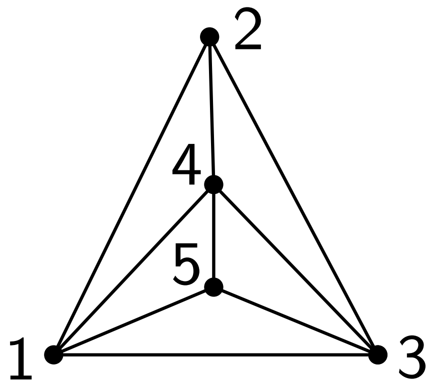
Theorem [Wagner ’36, Fary ’48, Stein ’51]

Every planar graph has a planar straight-line drawing.

History

- Does every planar graph have a planar straight-line drawing?





Theorem [Wagner '36, Fary '48, Stein '51]

Every planar graph has a planar straight-line drawing.

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

3 - 4

Next

Karlsruhe Institute of Technology

This lecture:

Theorem [De Fraysseix, Pach, Pollack '90]

Every n-vertex planar graph has a planar straight-line drawing of a size $(2n-4) \times (n-2)$.

Next lecture:

Theorem [Schnyder '90]

Every n-vertex planar graph has a planar straight-line drawing of a size $(n-2) \times (n-2)$.

Outline

- Canonical ordering. Existence.
- Canonical ordering. Computation.
- Shift algorithm.
- Proof of planarity.
- Implentational details.

Outline

- Canonical ordering. Existence.
- Canonical ordering. Computation.
- Shift algorithm.
- Proof of planarity.
- Implentational details.

Canonical Ordering

Definition: Canonical Ordering

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

- (C1) Vertices $\{v_1, \ldots v_k\}$ induce a 2-connected internally triangulated graph, call it G_k

Institut für Theoretische Informatik
Lehrstuhl Algorithmik I

Canonical Ordering

KIT
Karlsruhe Institute of Technology

Definition: Canonical Ordering

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

- (C1) Vertices $\{v_1, \ldots v_k\}$ induce a 2-connected internally triangulated graph, call it G_k
- (C2) Edge (v_1, v_2) belongs to the outer face of G_k

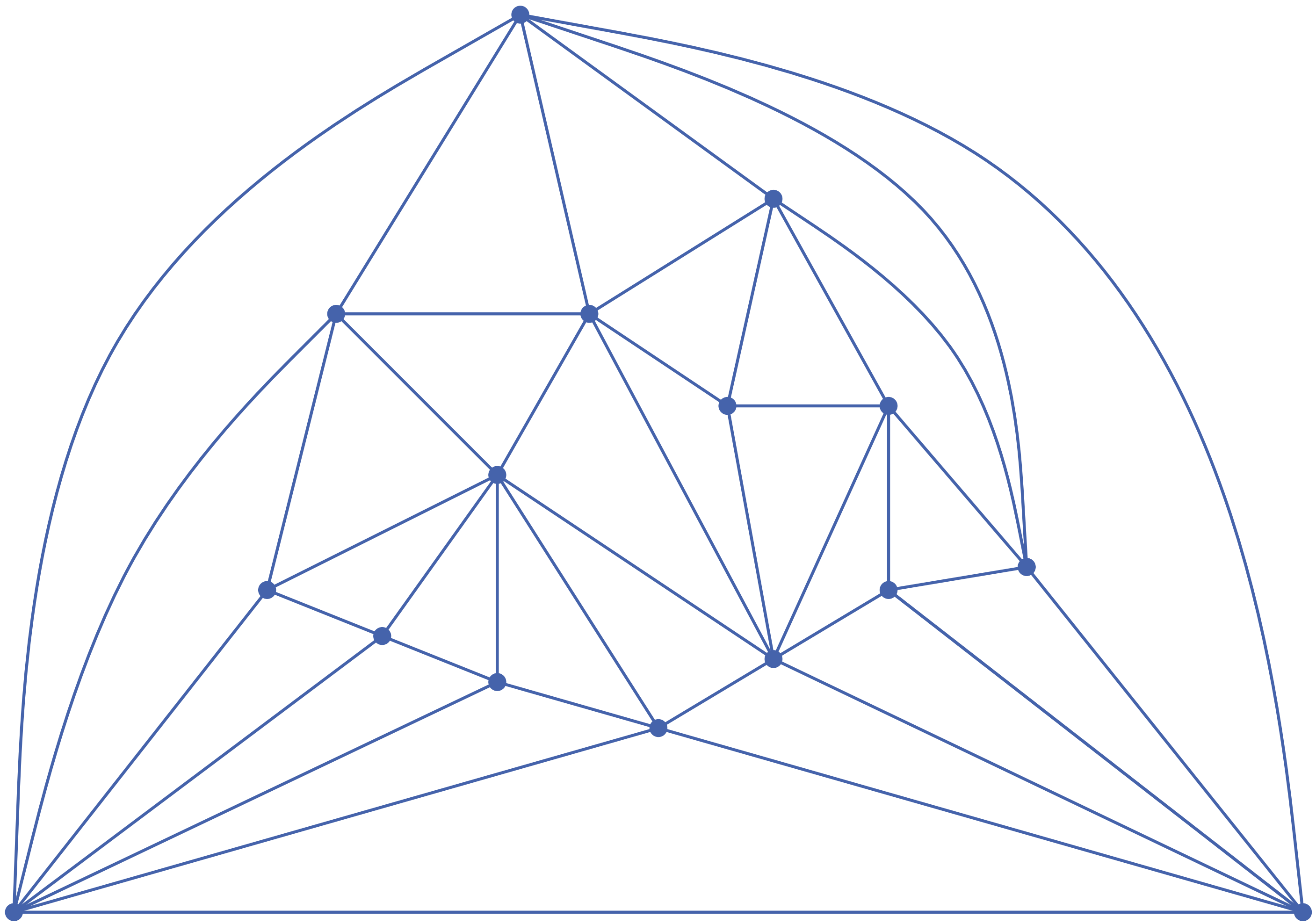
Canonical Ordering

Definition: Canonical Ordering

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

- (C1) Vertices $\{v_1, \ldots v_k\}$ induce a 2-connected 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.

Example of Canonical Ordering

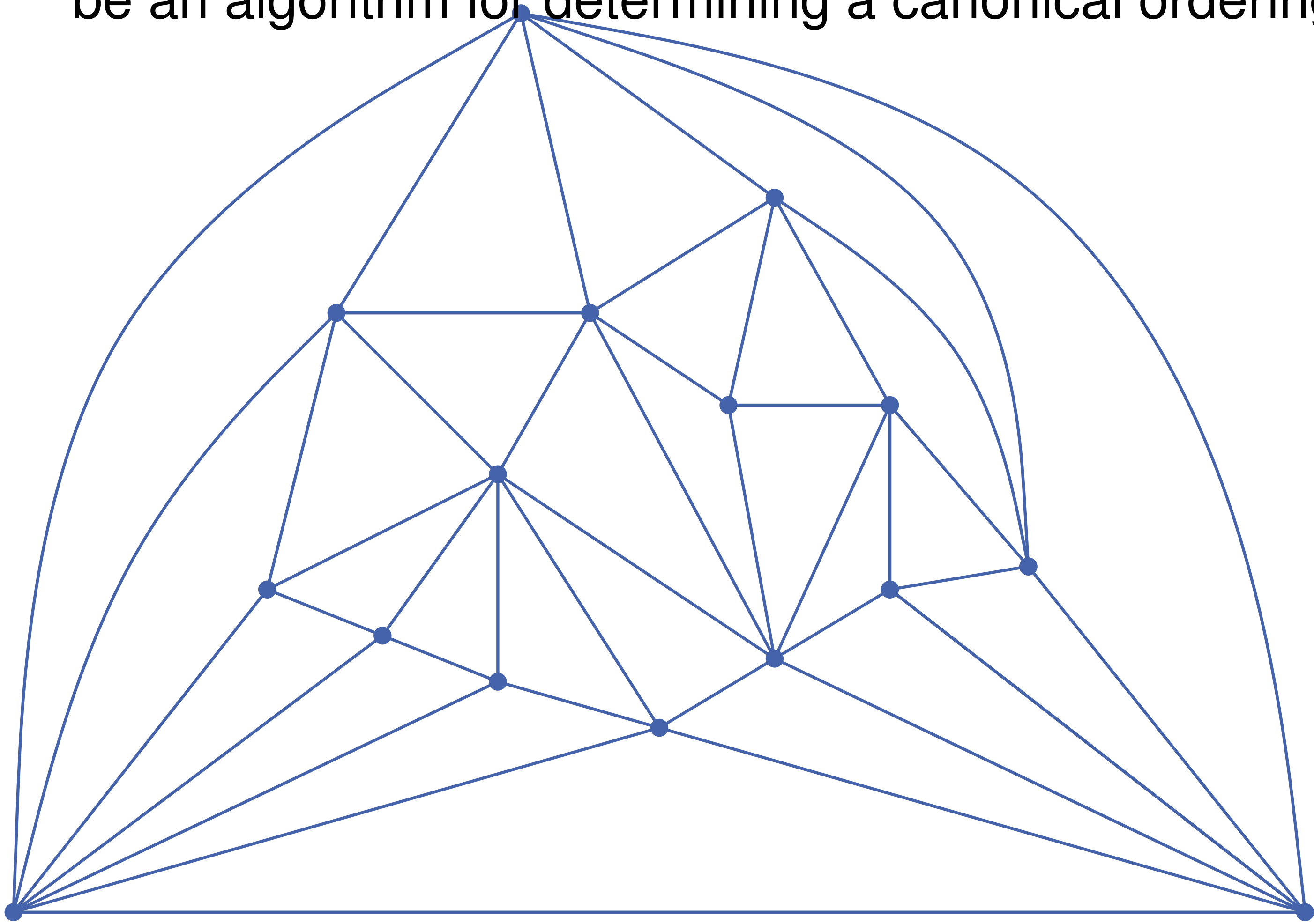


8 - 1

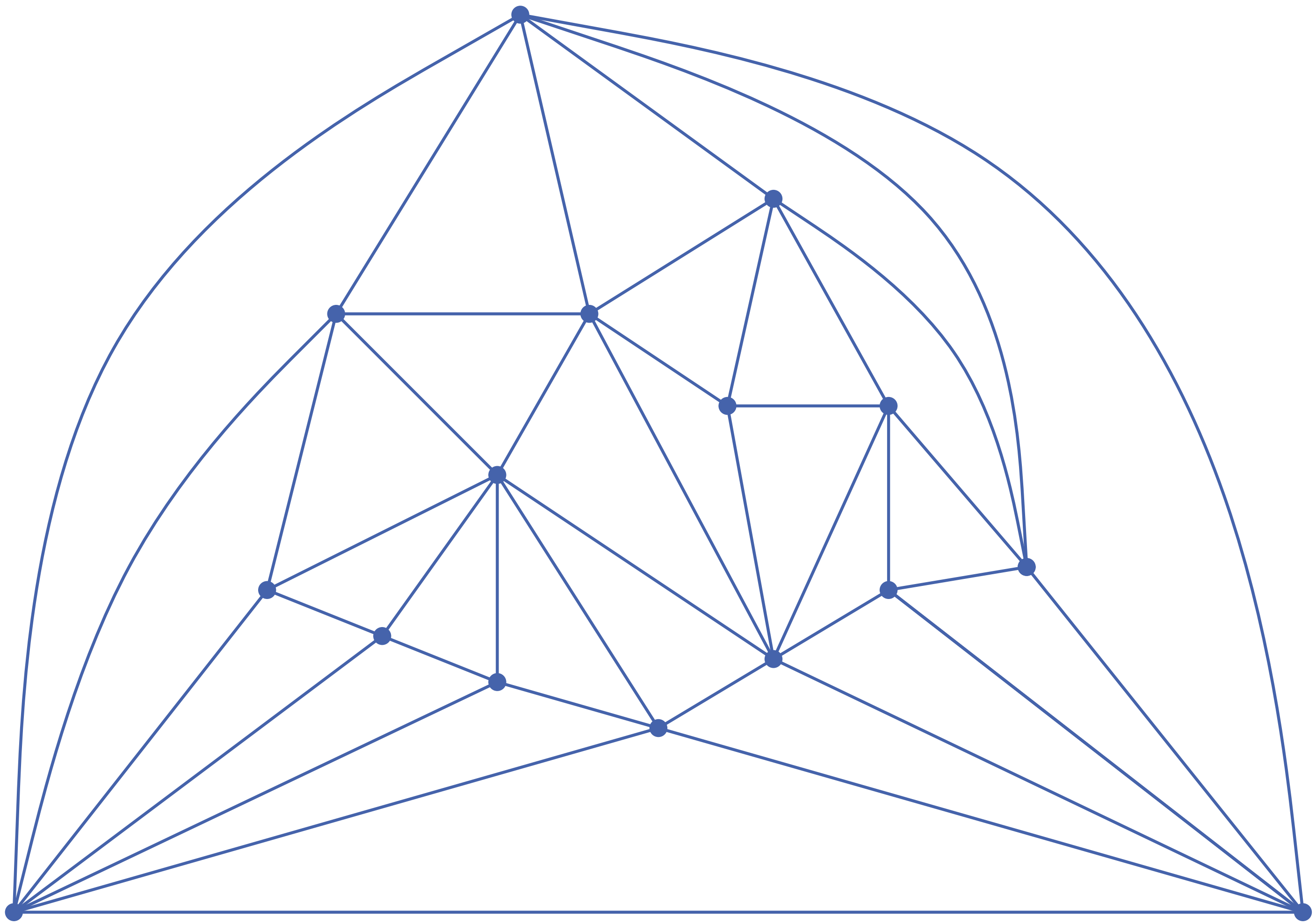
Example of Canonical Ordering

Work with your neighbour(s) and then share **5 min**

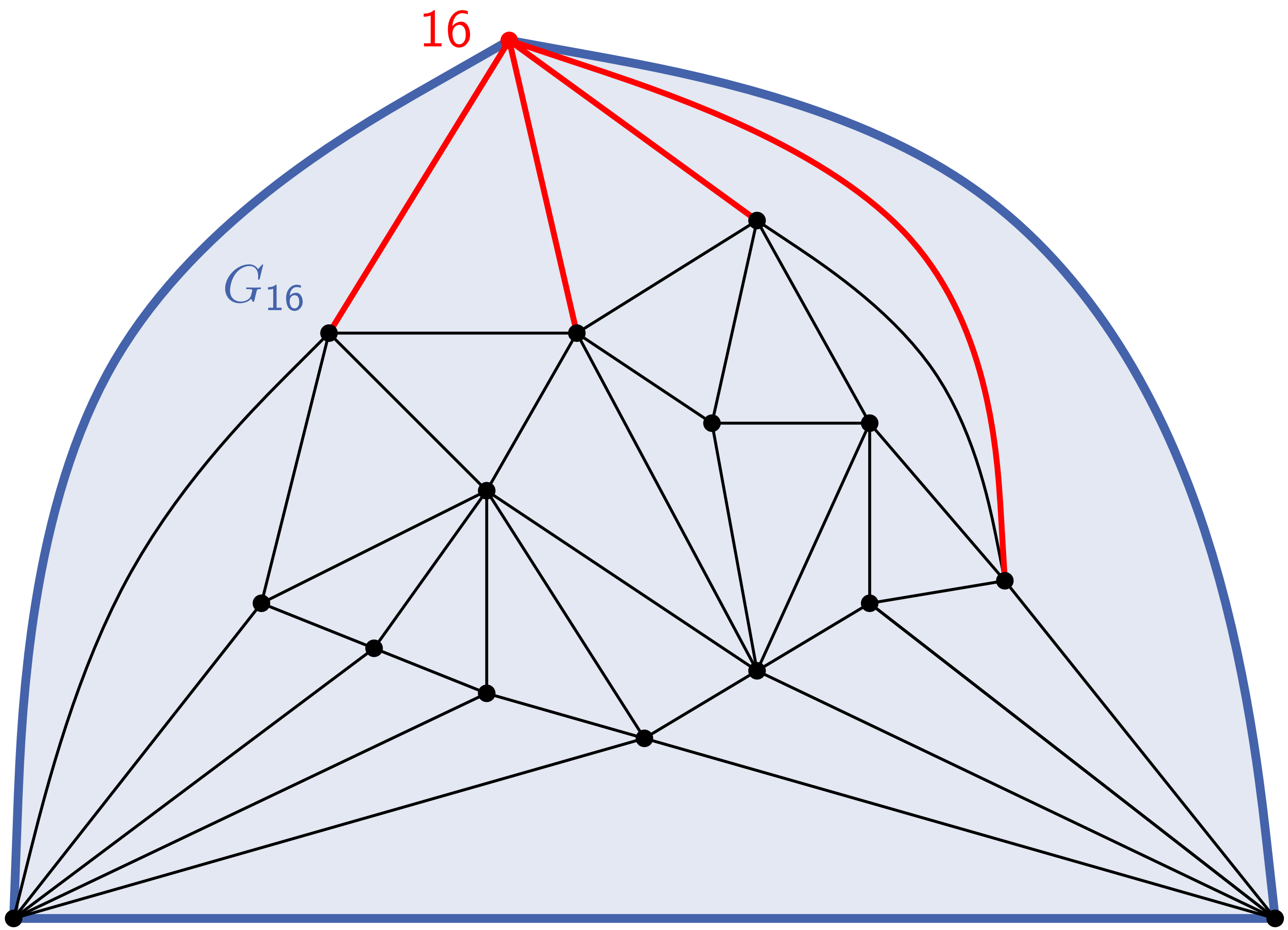
- Compute a canonical ordering of this graph. What could be an algorithm for determining a canonical ordering?



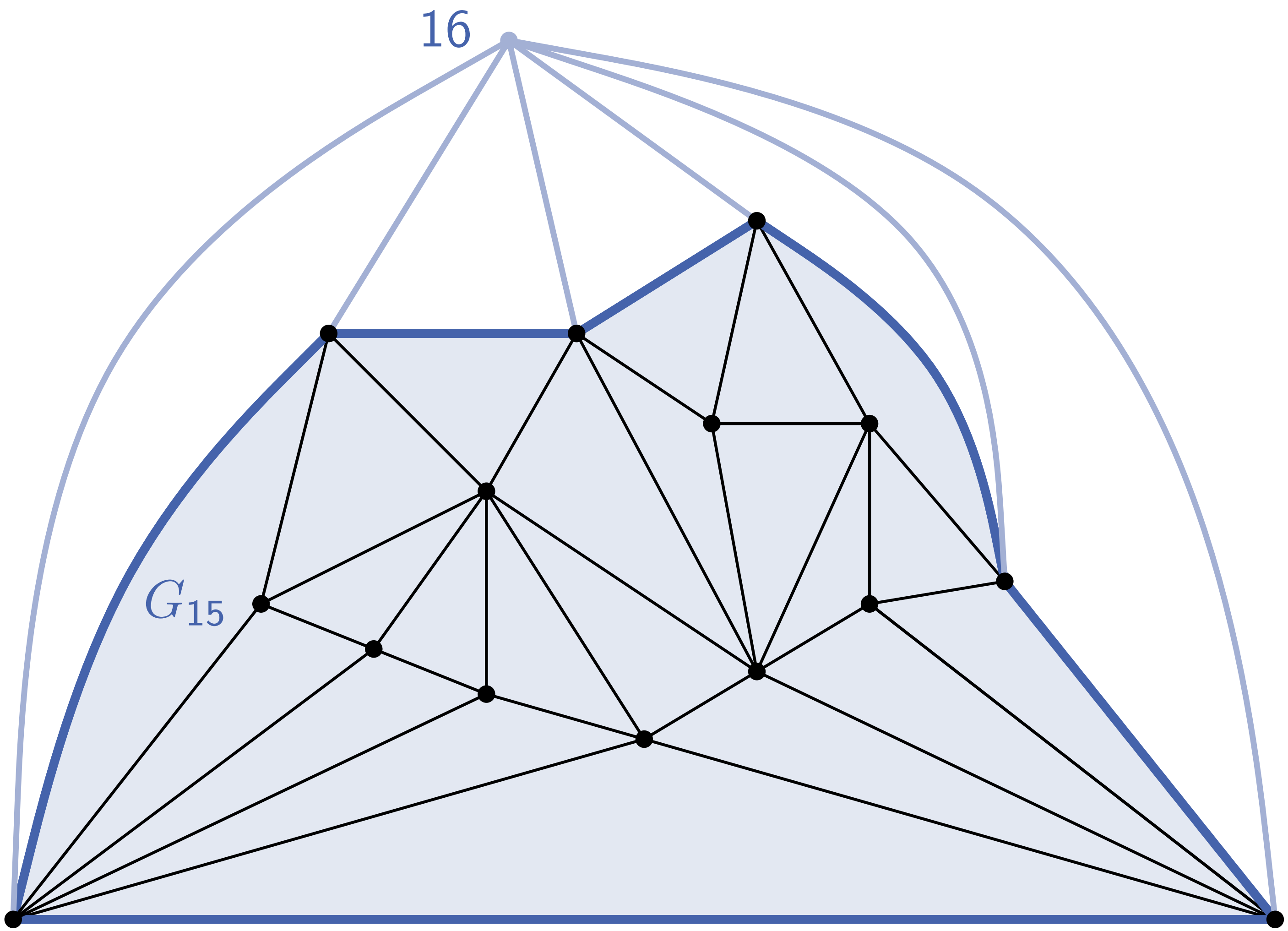
Example of Canonical Ordering



Example of Canonical Ordering

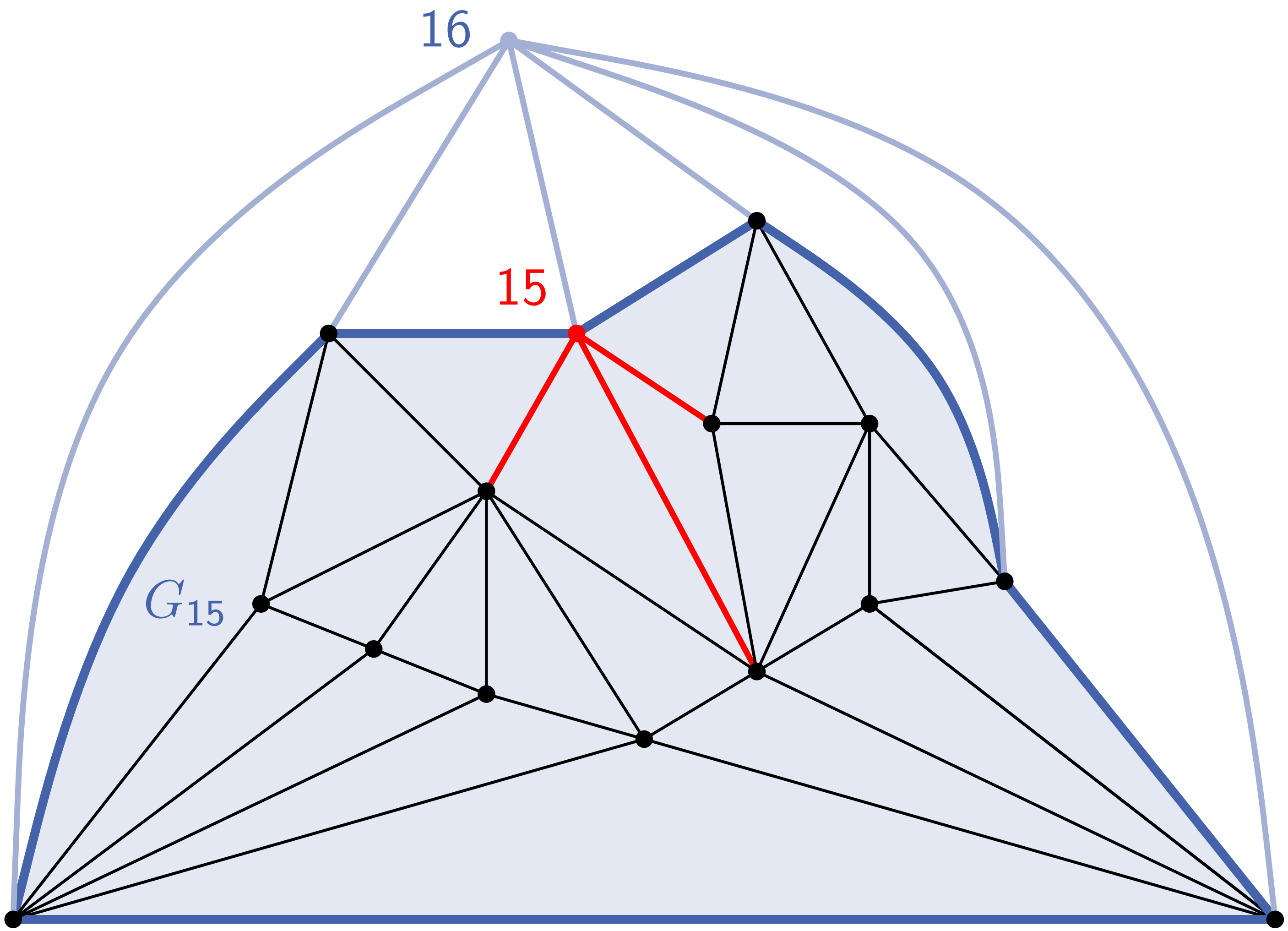


Example of Canonical Ordering

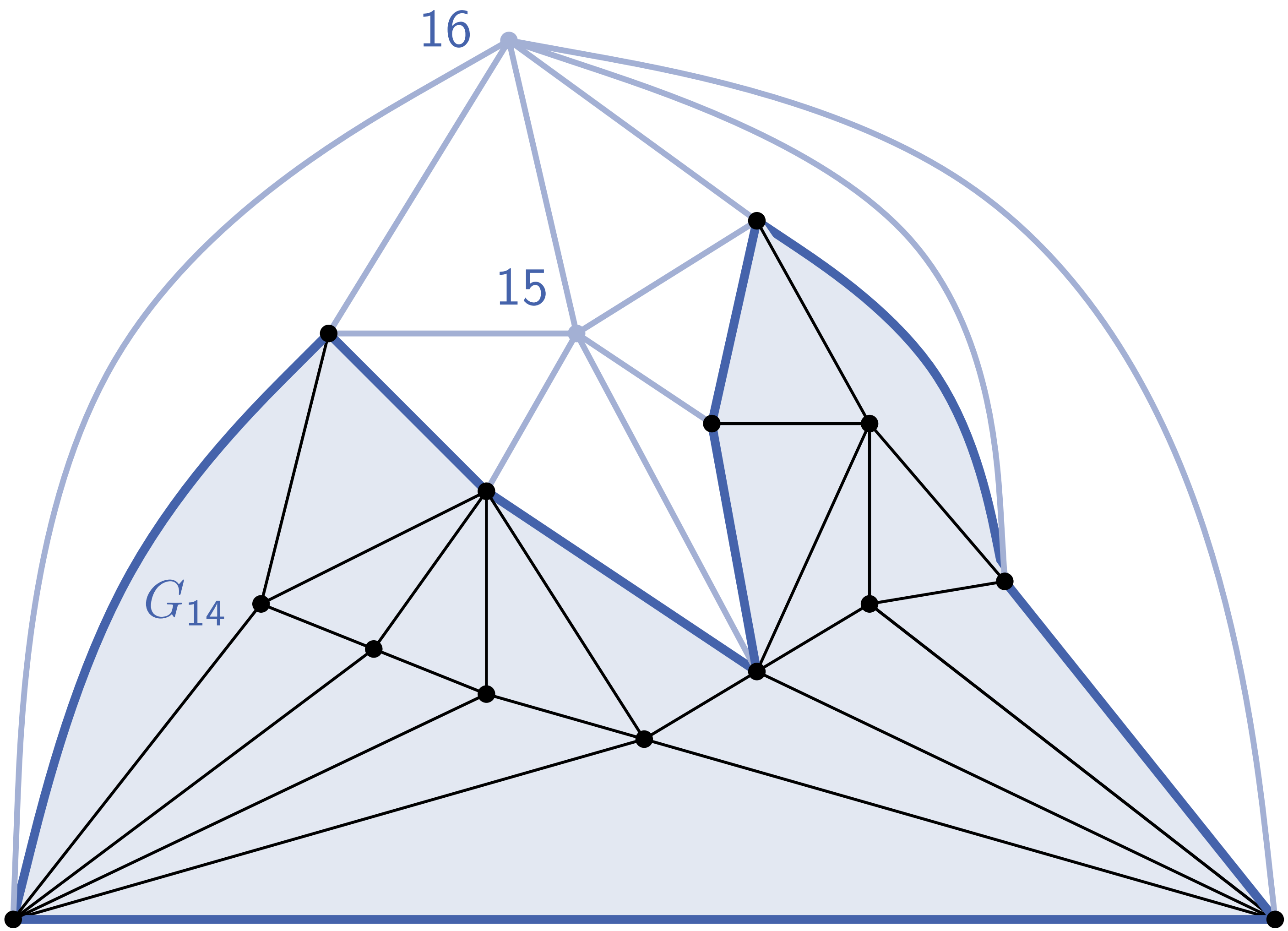


8 - 5

Example of Canonical Ordering

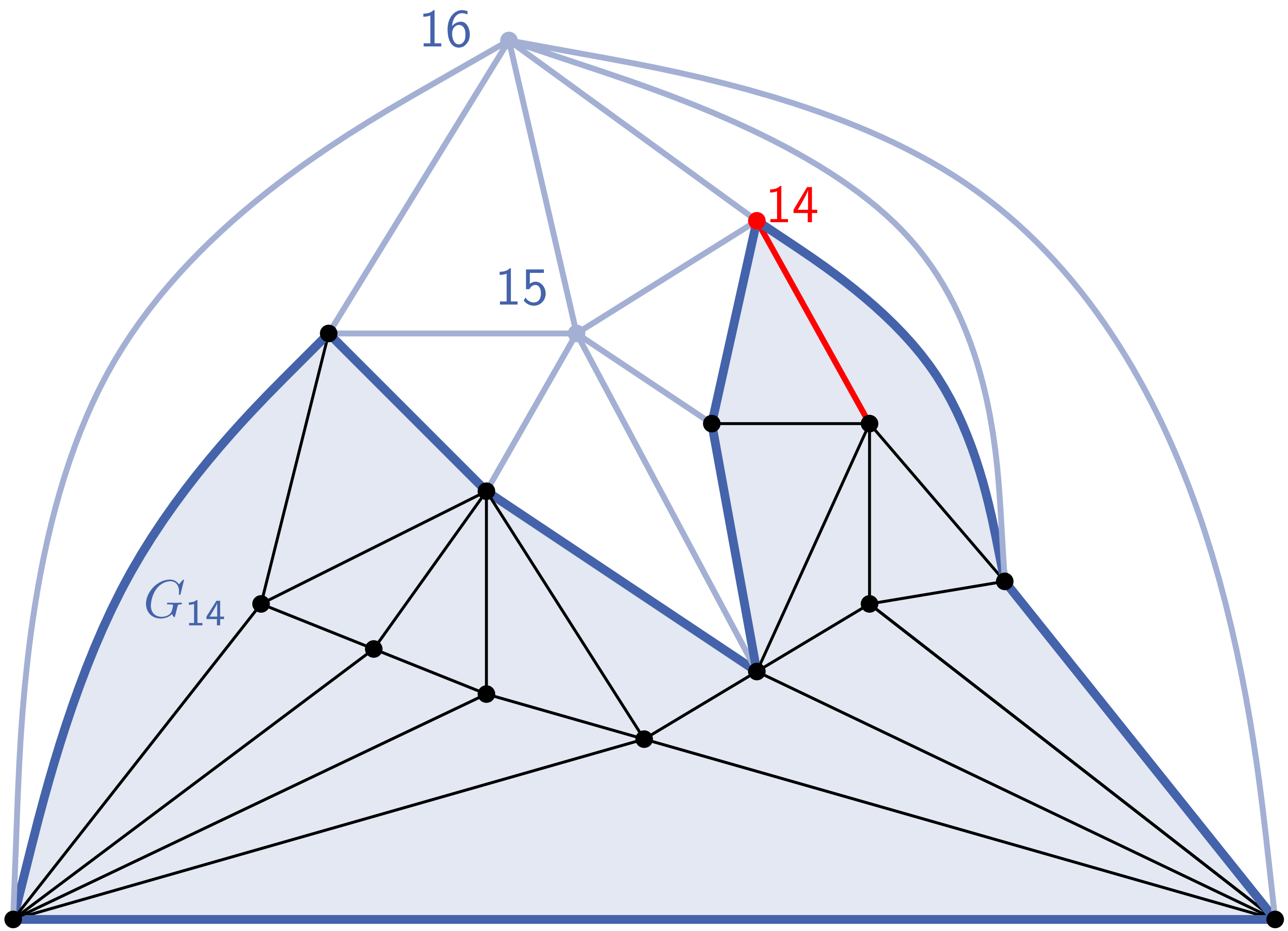


Example of Canonical Ordering



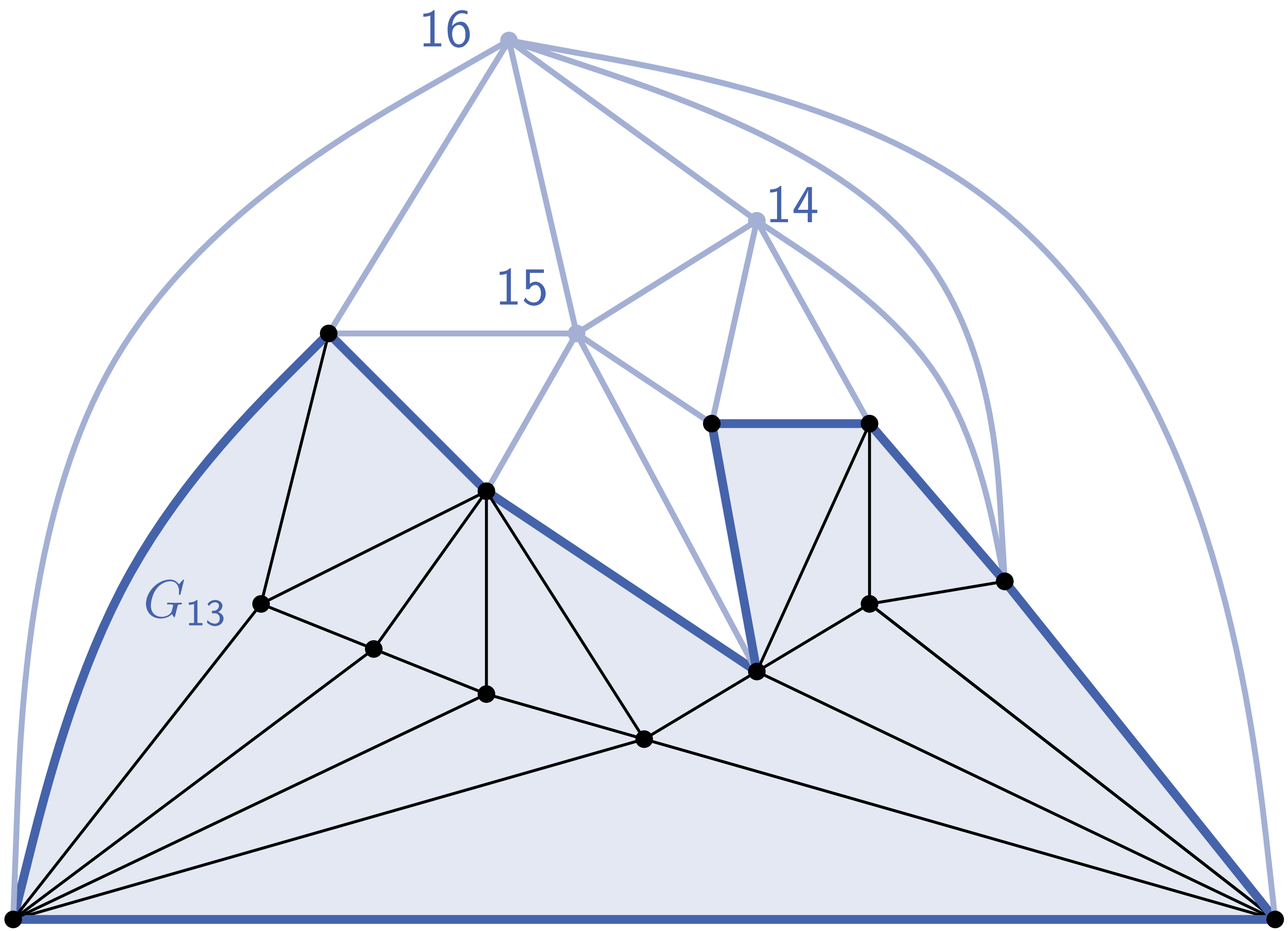
8 - 7

Example of Canonical Ordering

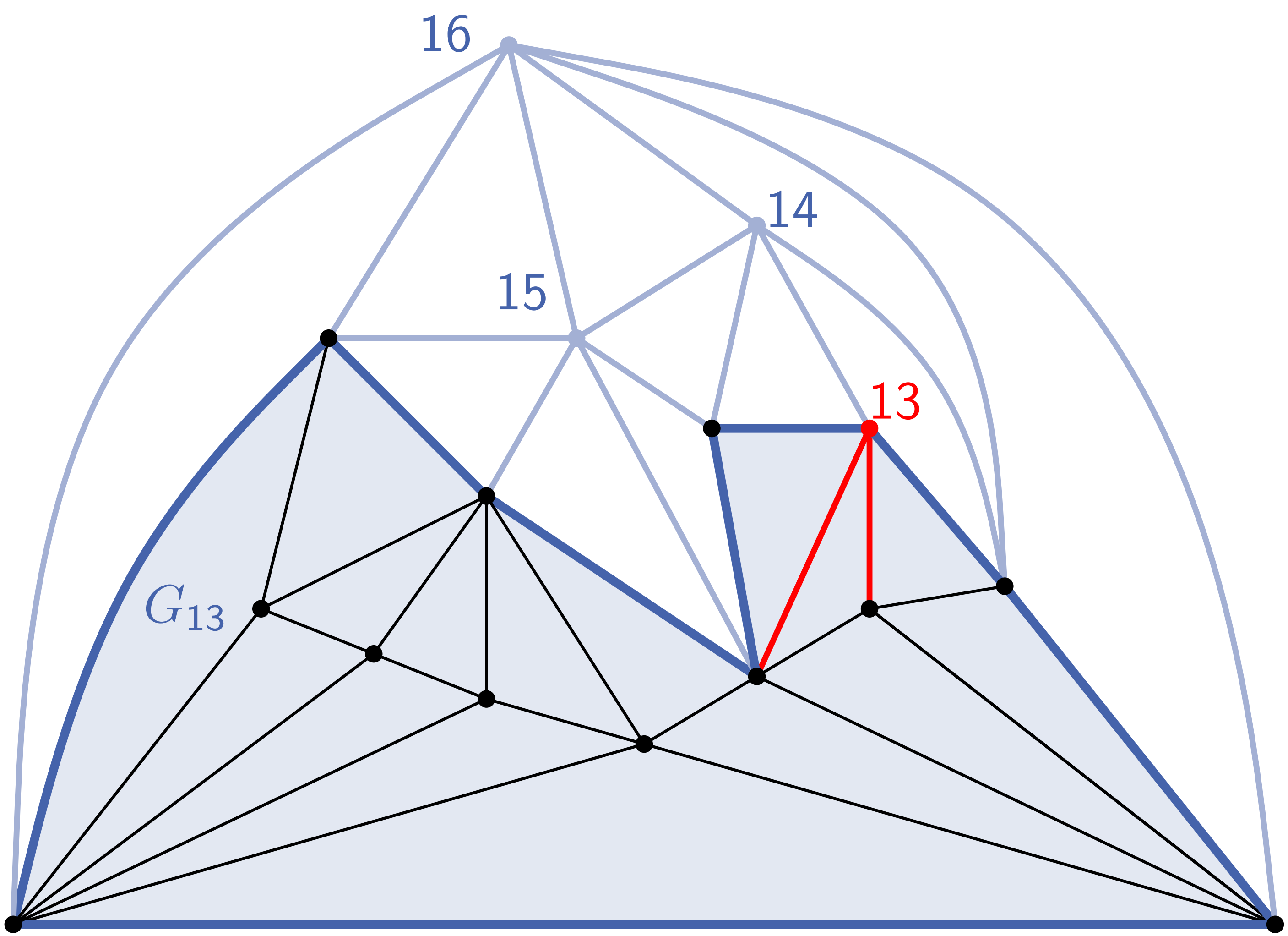


8 - 8

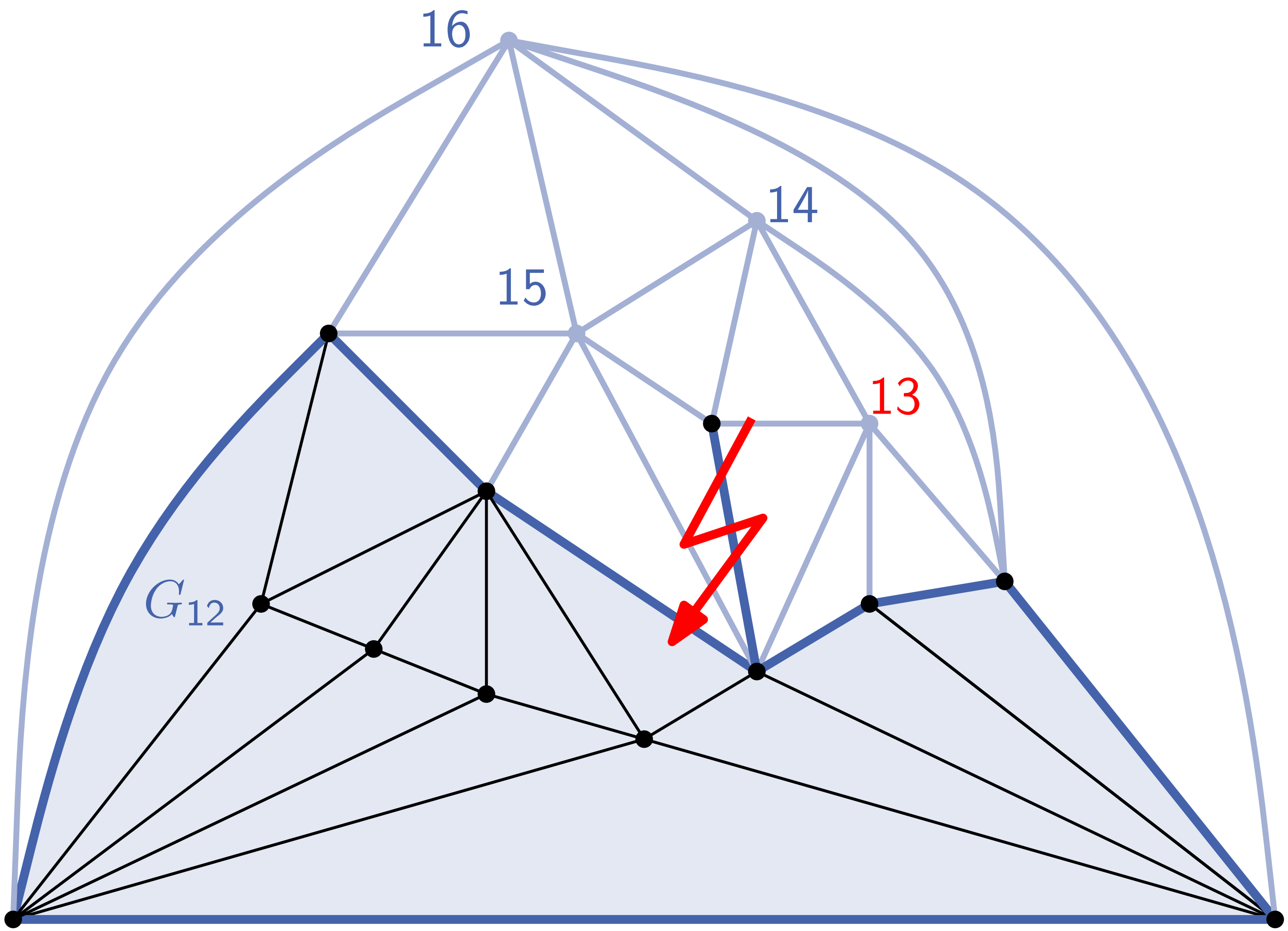
Example of Canonical Ordering



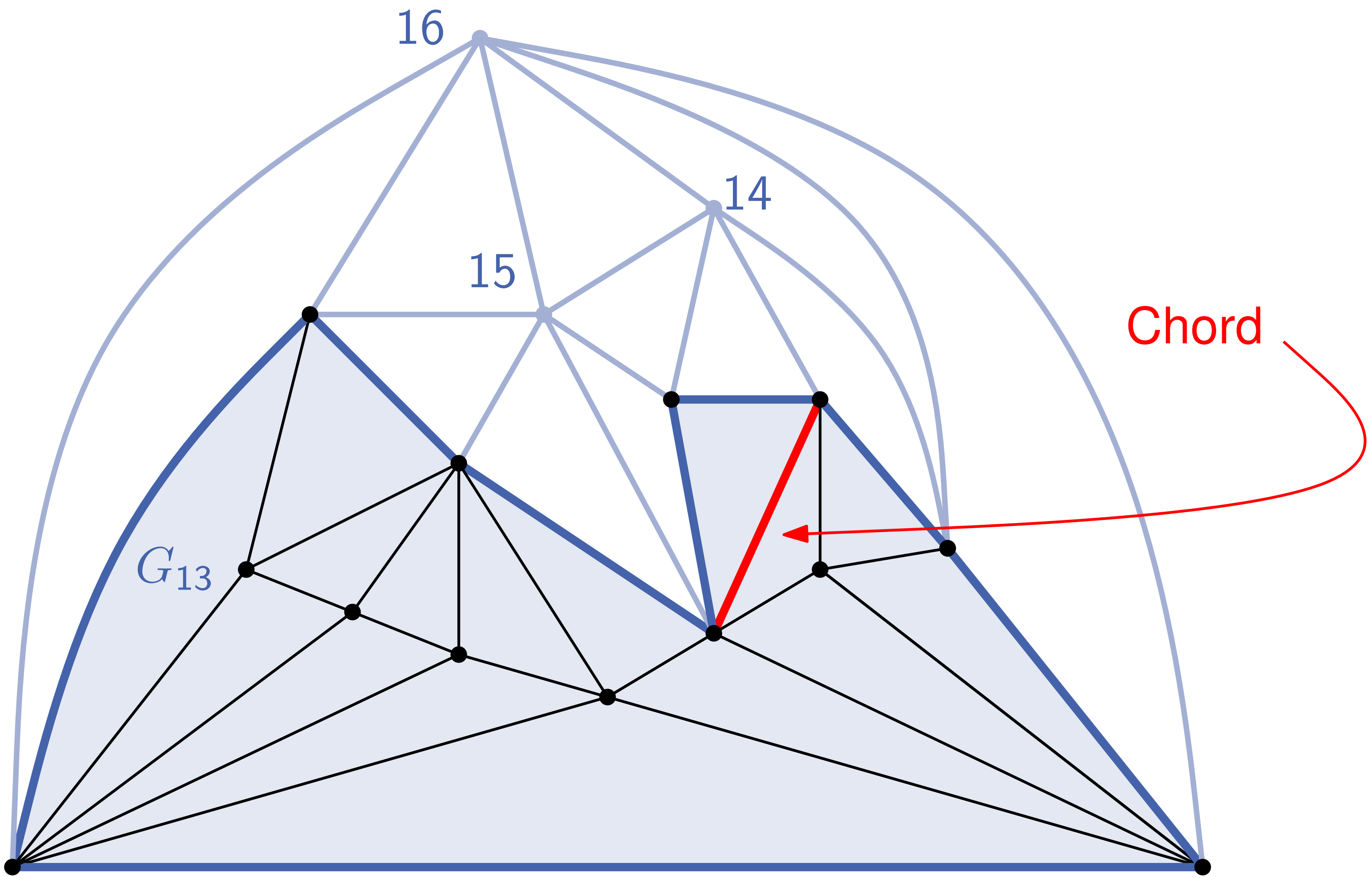
Example of Canonical Ordering



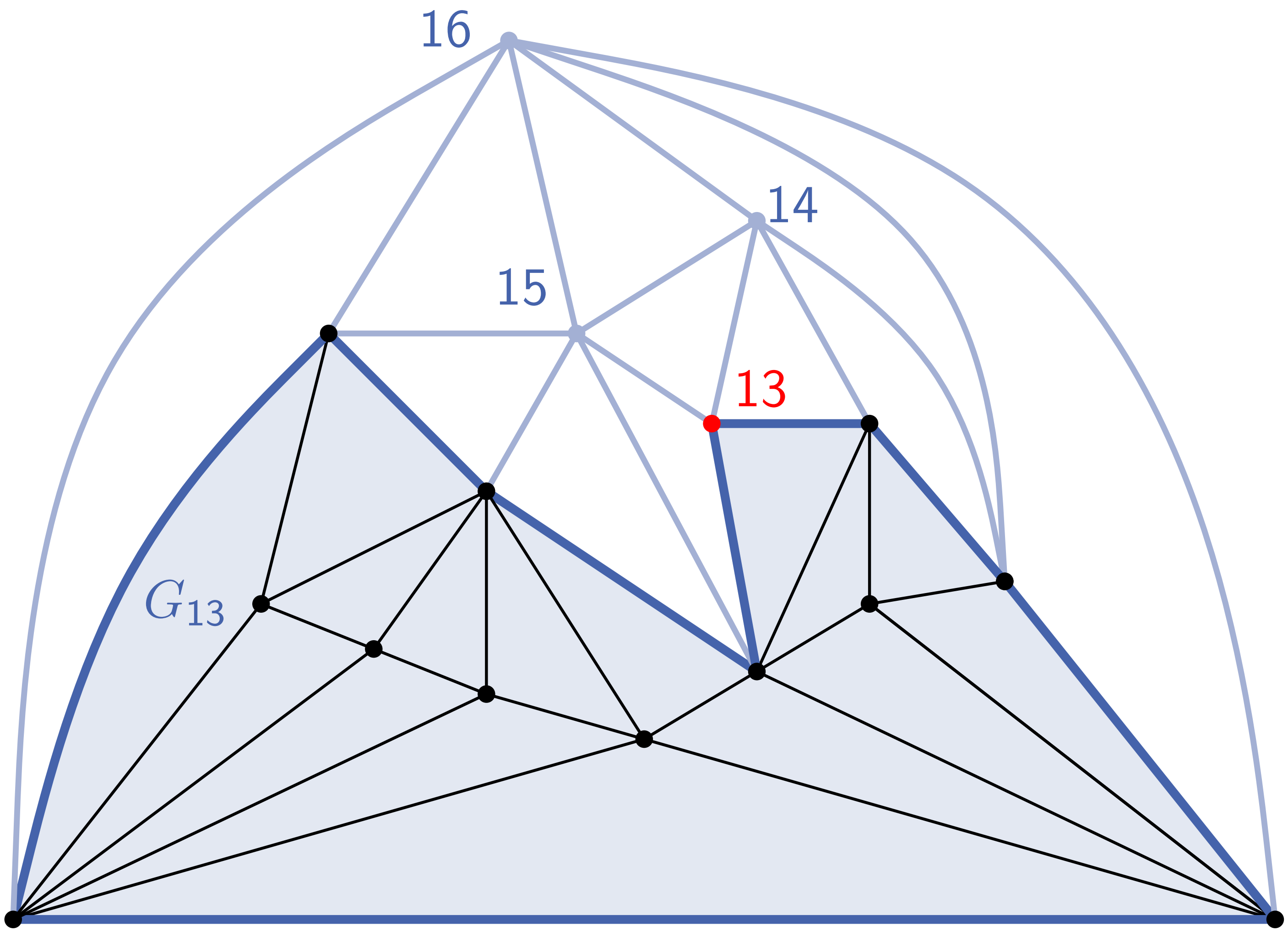
Example of Canonical Ordering



Example of Canonical Ordering

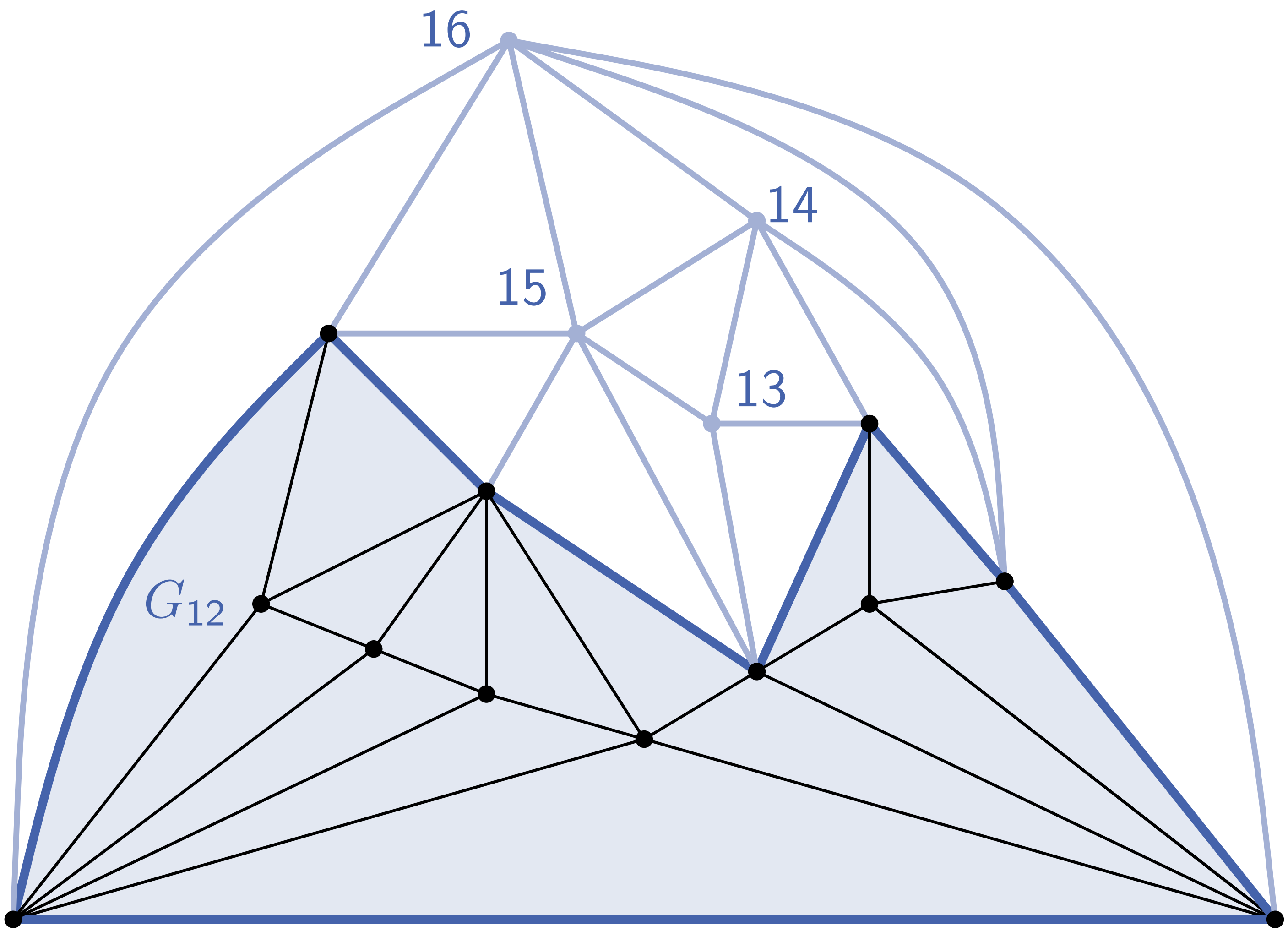


Example of Canonical Ordering



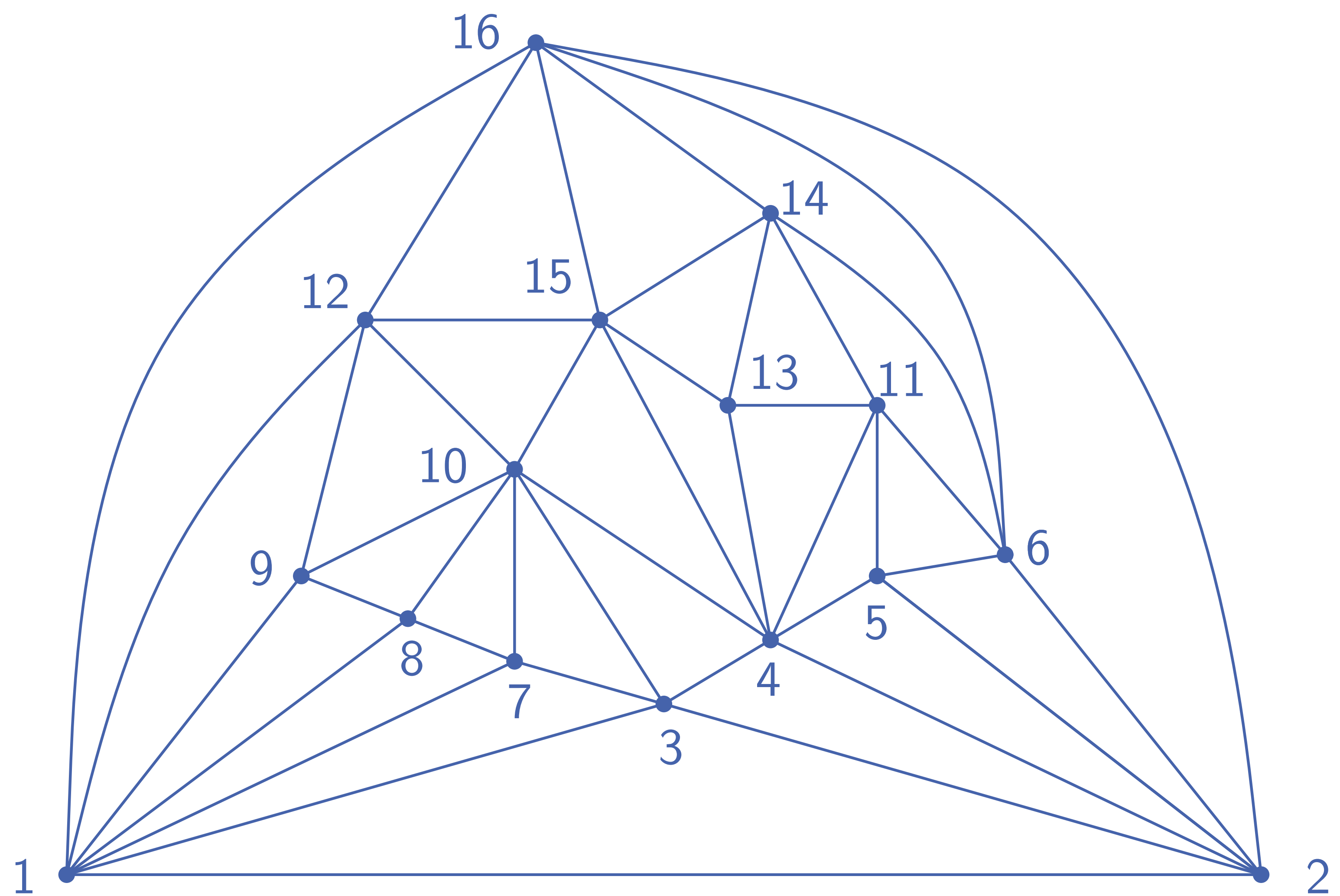
8 - 13

Example of Canonical Ordering



8 - 14

Example of Canonical Ordering



8 - 15

Canonical Ordering. Existence.

Lemma

Every triangulated plane graph has a canonical ordering.

- Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions C1-C3 hold.

Canonical Ordering. Existence.

Lemma

Every triangulated plane graph has a canonical ordering.

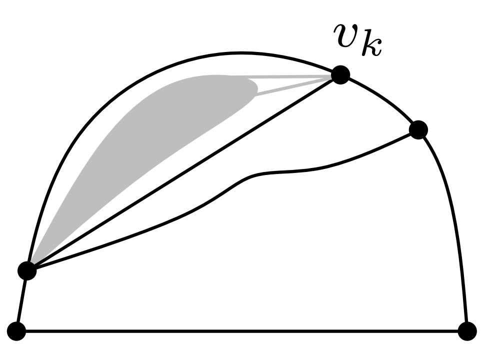
- Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n. Conditions C1-C3 hold.
- Induction hypothesis: vertices $v_{n-1}, \ldots, v_{k+1}$ have been chosen such that conditions C1-C3 hold for $k+1 \leq i \leq n$.

Canonical Ordering. Existence.

Lemma

Every triangulated plane graph has a canonical ordering.

- 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 $k+1 \leq i \leq n$.
- Consider G_k. We search for v_k.

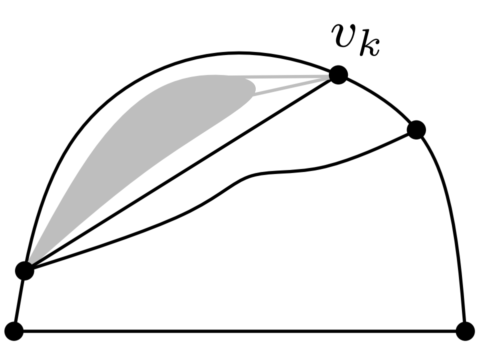


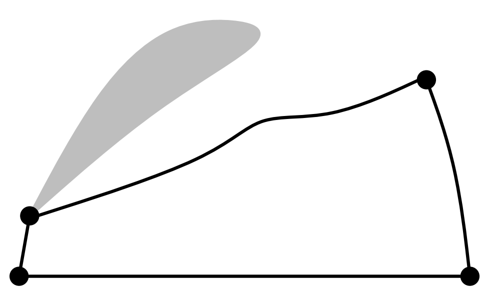
Canonical Ordering. Existence.

Lemma

Every triangulated plane graph has a canonical ordering.

- 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 $k+1 \leq i \leq n$.
- Consider G_k. We search for v_k.



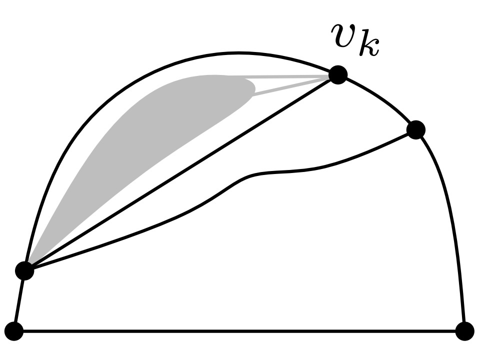


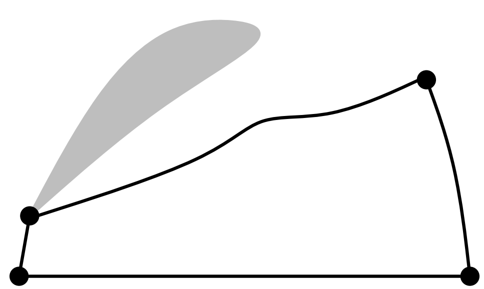
Canonical Ordering. Existence.

Lemma

Every triangulated plane graph has a canonical ordering.

- 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 $k + 1 \leq i \leq n$.
- Consider G_k. We search for v_k.





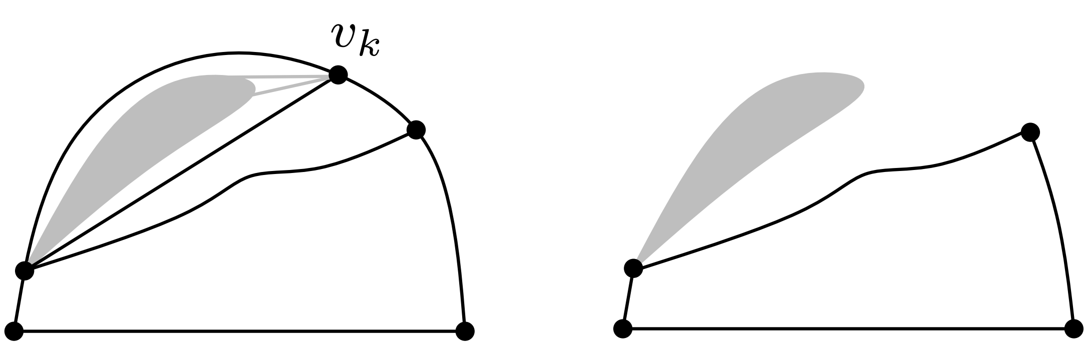
v_k should not be adjacent to a chord

Canonical Ordering. Existence.

Lemma

Every triangulated plane graph has a canonical ordering.

- 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 $k + 1 \leq i \leq n$.
- Consider G_k. We search for v_k.

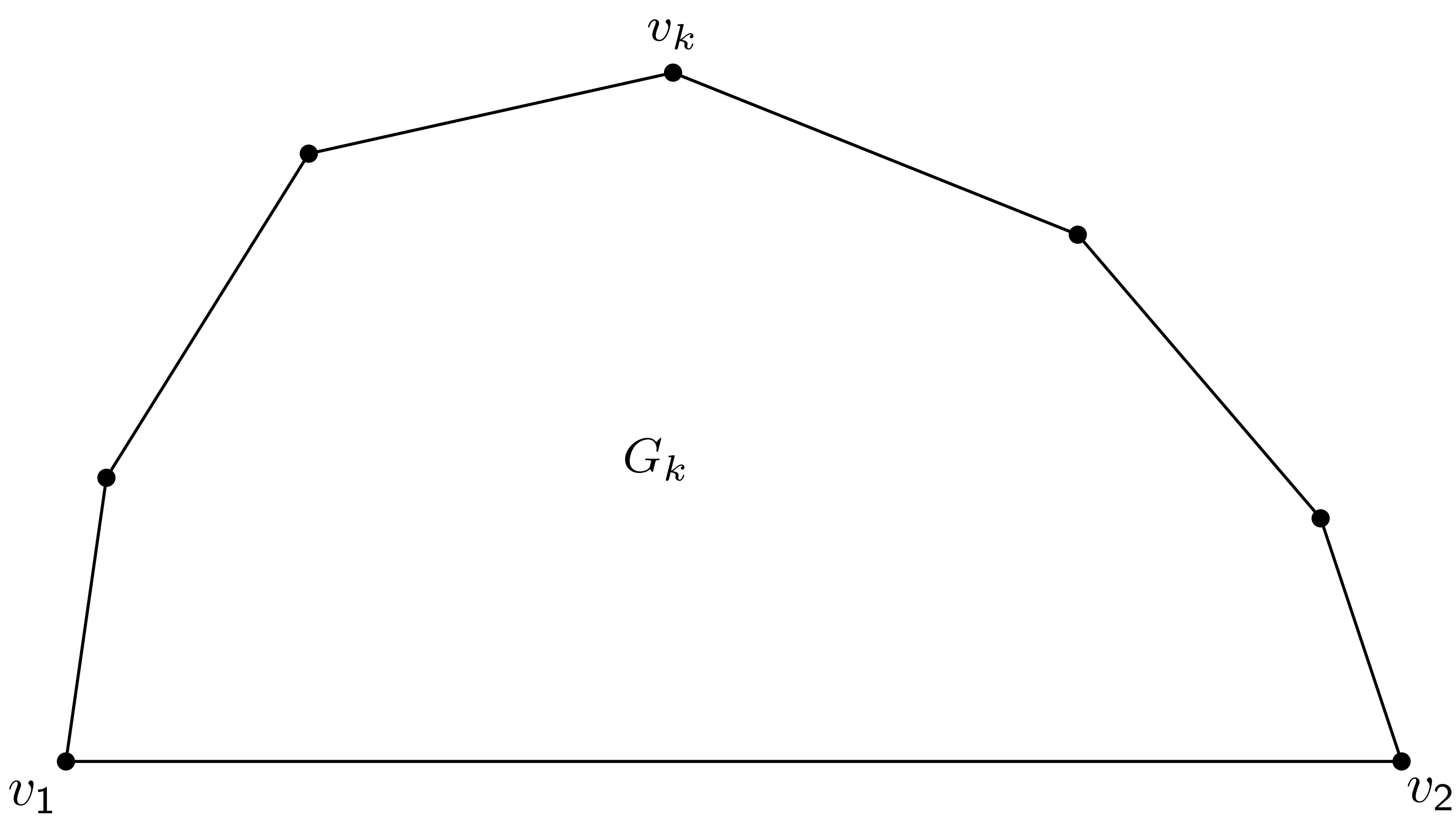


v_k should not be adjacent to a chord

Is it sufficient?

Canonical Ordering. Existence.

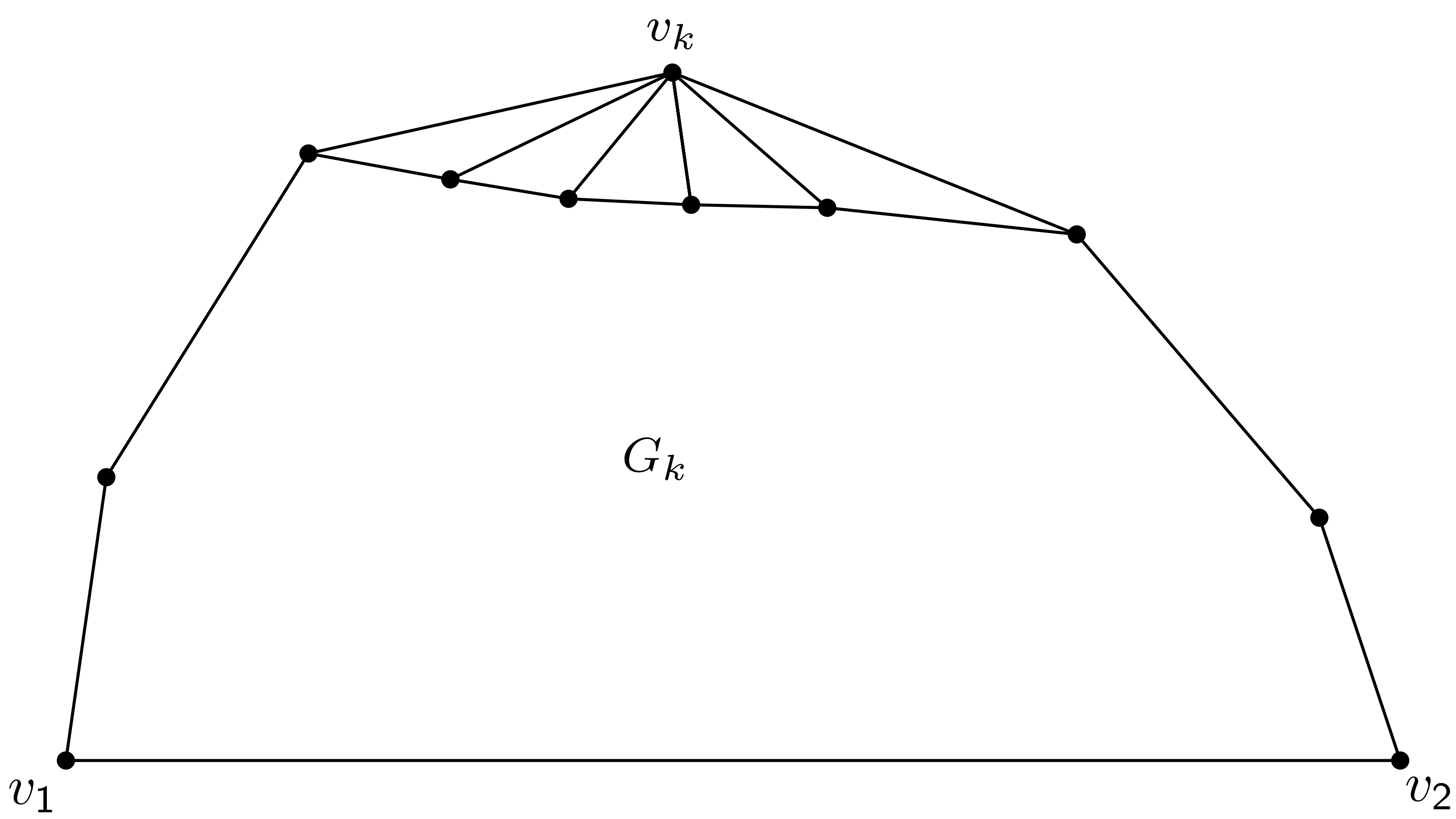
Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



10 - 1

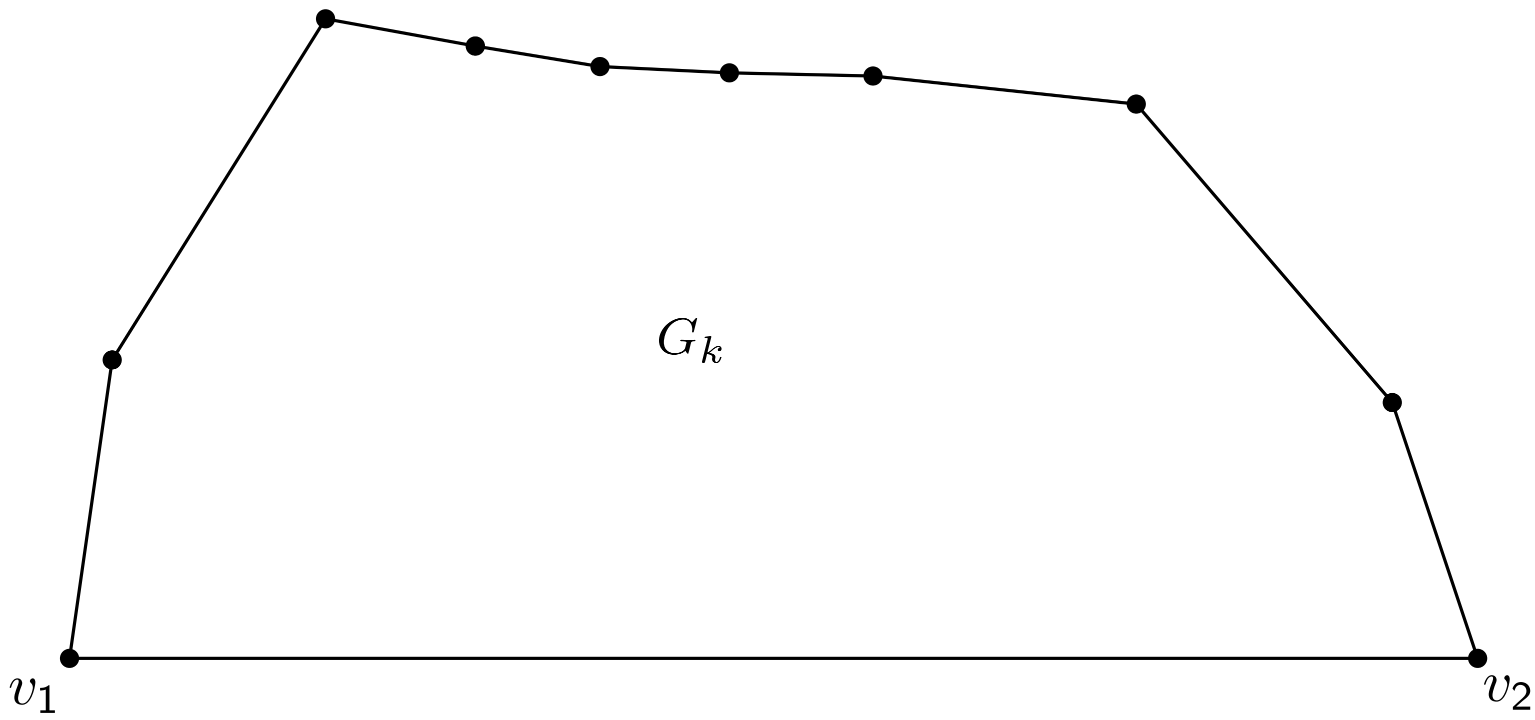
Canonical Ordering. Existence.

Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



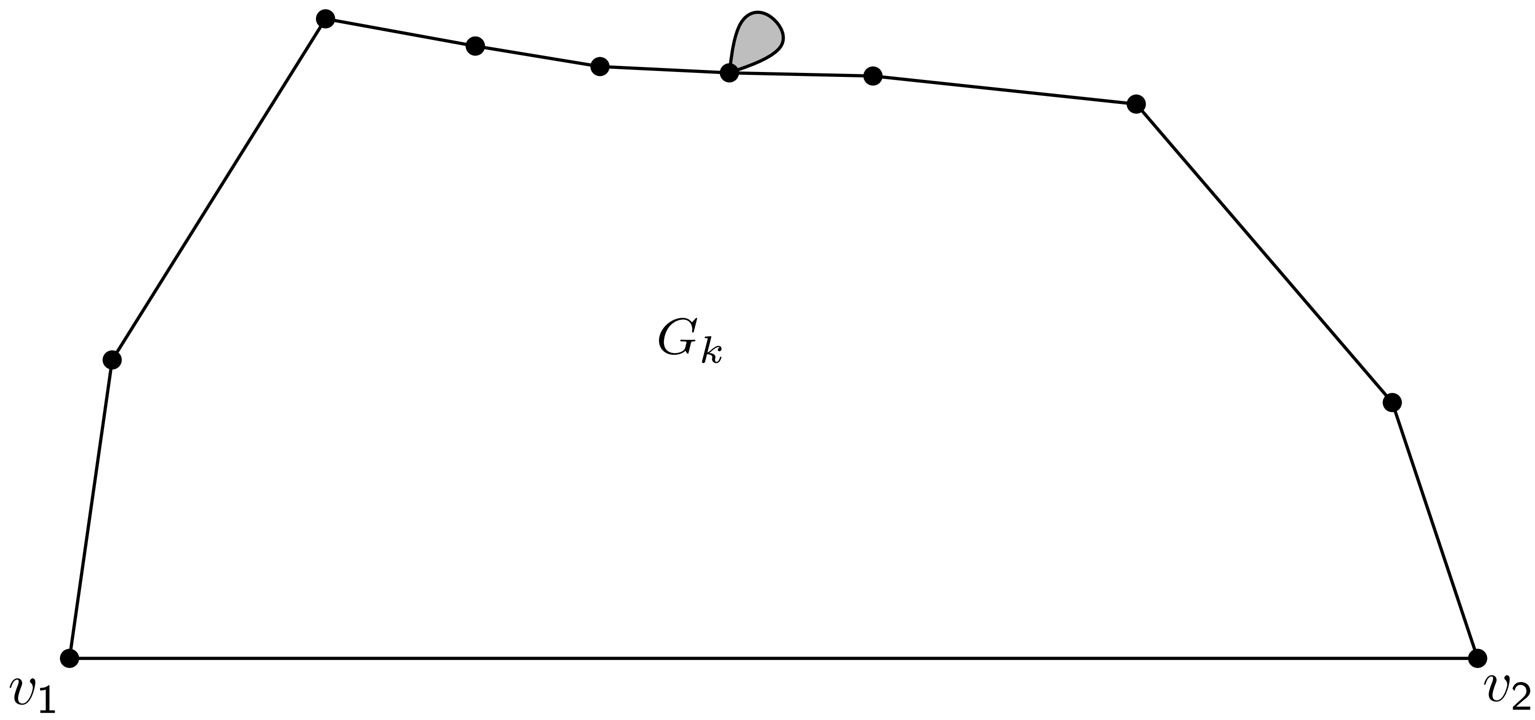
Canonical Ordering. Existence.

Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



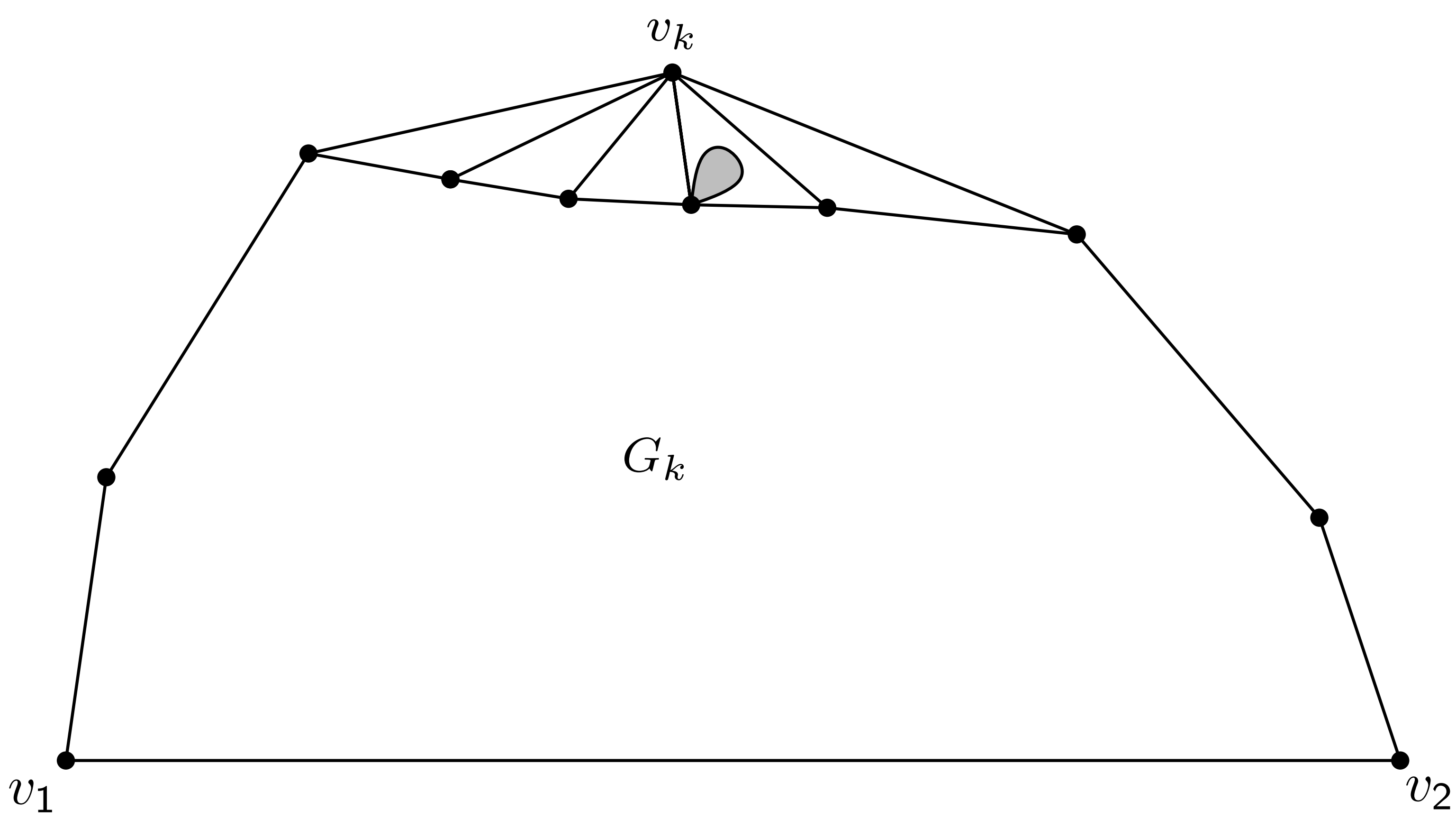
Canonical Ordering. Existence.

Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



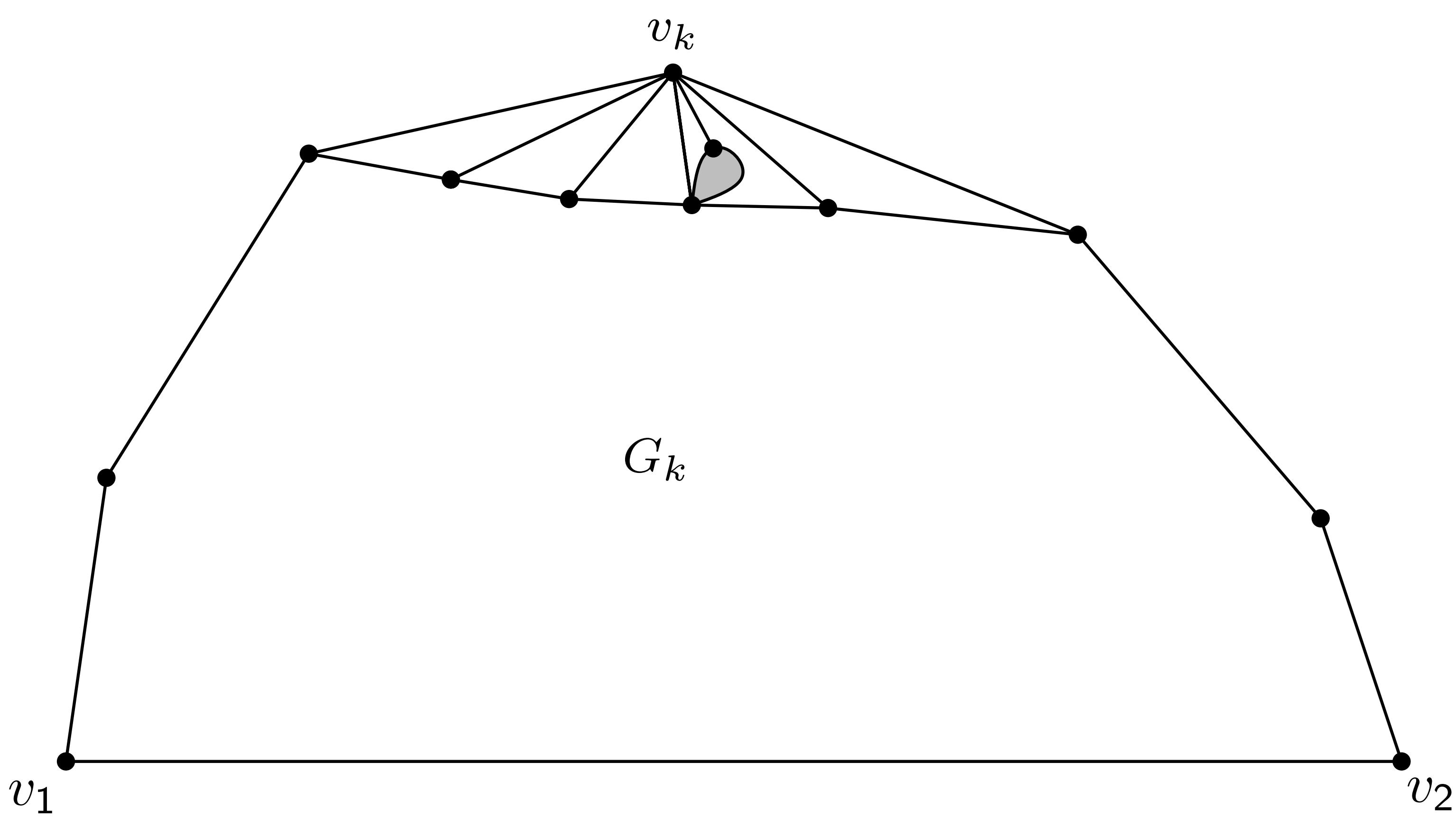
Canonical Ordering. Existence.

Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



Canonical Ordering. Existence.

Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.

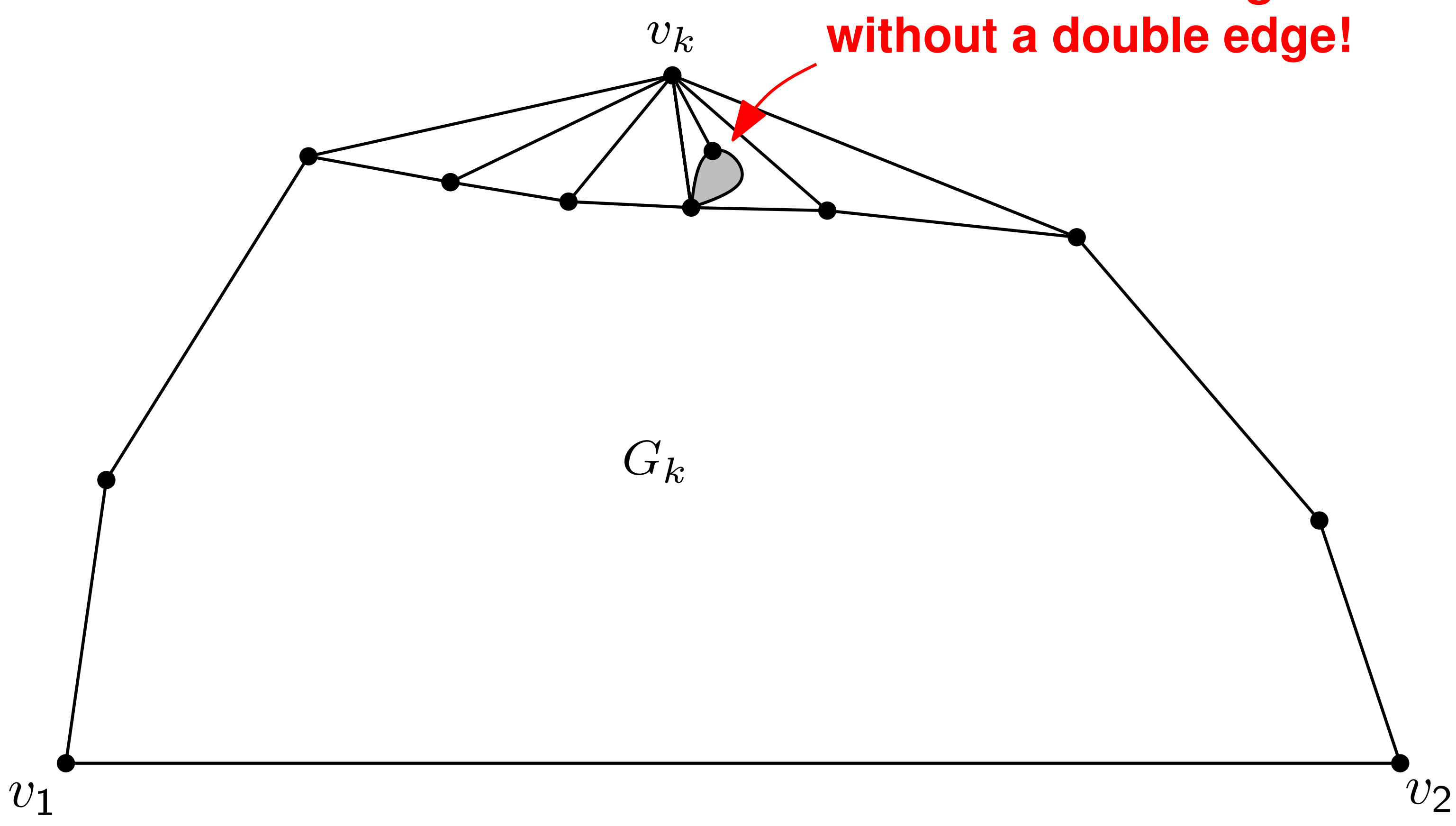


10 - 6

Canonical Ordering. Existence.

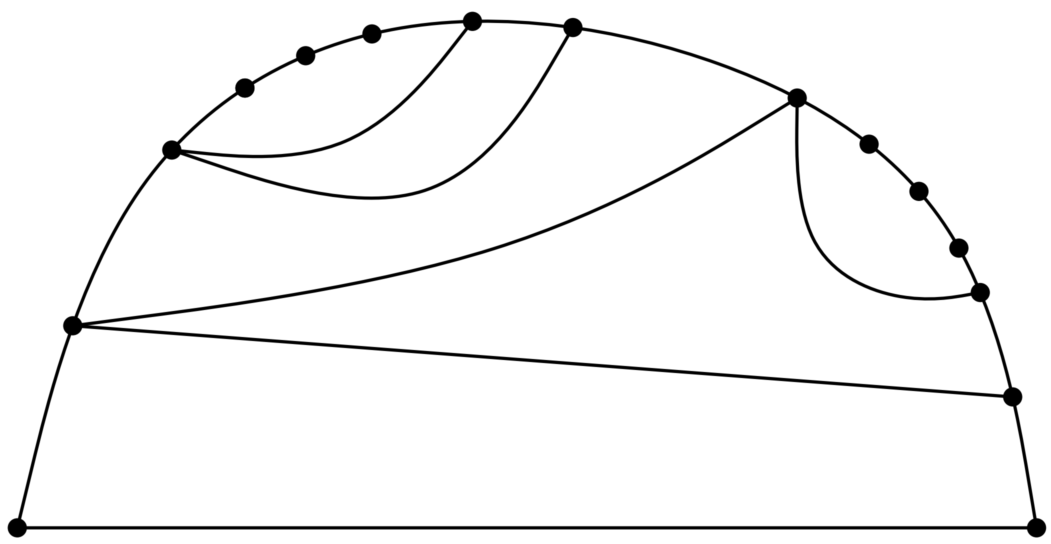
Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.

We can not triangulate this face without a double edge!



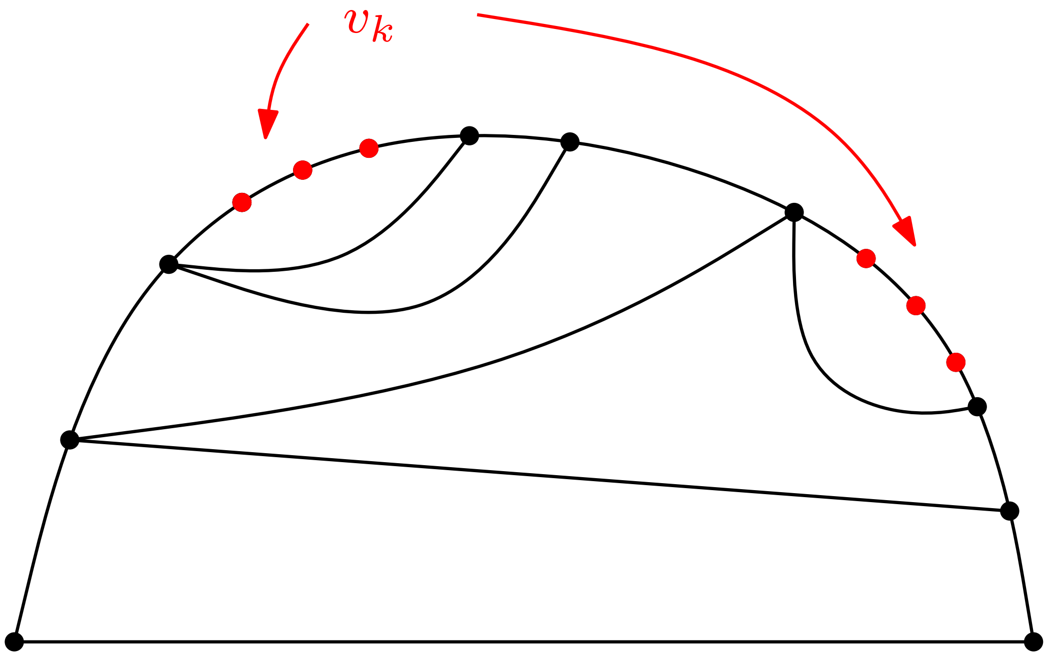
Canonical Ordering. Existence.

- Why a vertex not adjacent to a chord exists?



Canonical Ordering. Existence.

- Why a vertex not adjacent to a chord exists?



Outline

- Canonical ordering. Existence.
- Canonical ordering. Computation.
- Shift algorithm.
- Proof of planarity.
- Implentational details.

Computing Canonical Ordering

Algorithm CO

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 $v \neq v_1, v_2$ such that mark(v) = false, out(v) = true,
    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};
  and let $w_p, \ldots, w_q$ be the unmarked neighbors v_k;
  out(w_i) $\leftarrow$ true for all $p < i < q$;
  update number of chords for w_i and its neighbors;

- chord(v) - number of chords adjacent to v
- mark(v) = true iff vertex v was numbered
- out(v)=true iff v is the outer vertex of current plane graph

Computing Canonical Ordering

Algorithm CO

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 $v \neq v_1, v_2$ such that mark(v) = false, out(v) = true,
    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};
  and let $w_p, \ldots, w_q$ be the unmarked neighbors v_k;
  out(w_i) $\leftarrow$ true for all $p < i < q$;
  update number of chords for w_i and its neighbors;

- chord(v) - number of chords adjacent to v
- mark(v) = true iff vertex v was numbered
- out(v)=true iff v is the outer vertex of current plane graph

Computing Canonical Ordering

Algorithm CO

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 $v \neq v_1, v_2$ such that mark(v) = false, out(v) = true, 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}; and let $w_p, \ldots, w_q$ be the unmarked neighbors v_k;*
- out(w_i) $\leftarrow$ true for all $p < i < q$;
- update number of chords for w_i and its neighbors;

Lemma

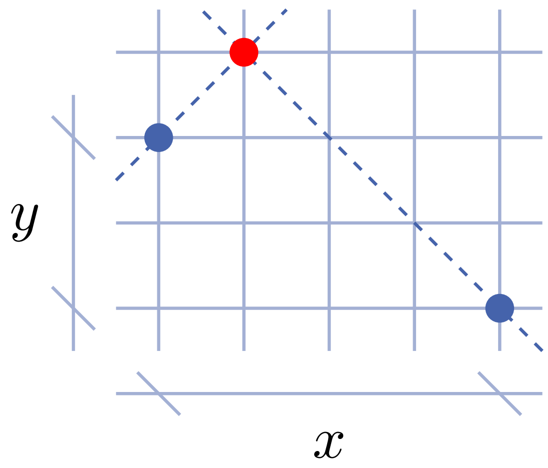
Algorithm CO computes a canonical ordering of a graph in $O(n)$ time.

Outline

- Canonical ordering. Existence.
- Canonical ordering. Computation.
- Shift algorithm.
- Proof of planarity.
- Implentational details.

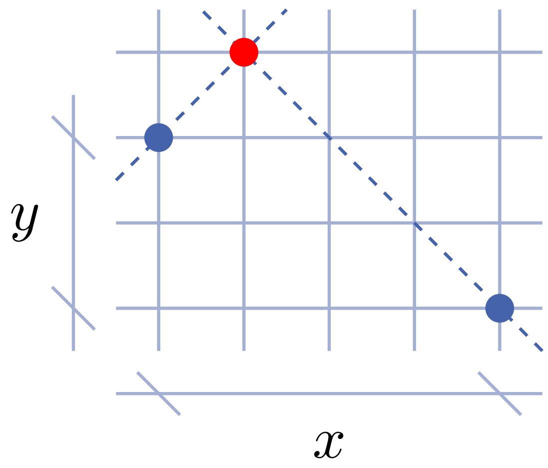
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



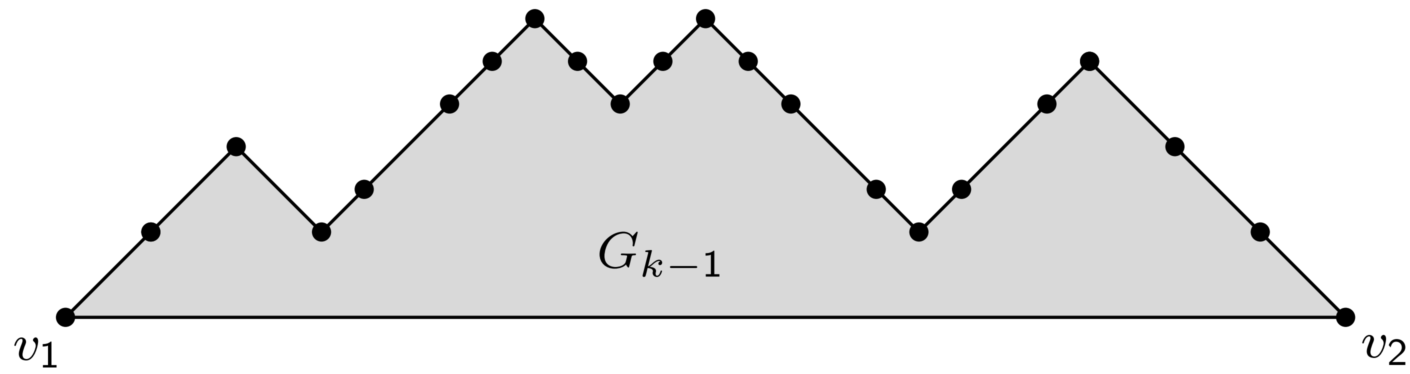
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

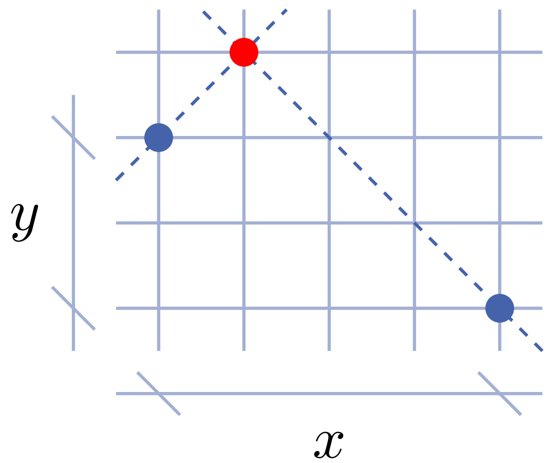
- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$
- Boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1



15 - 2

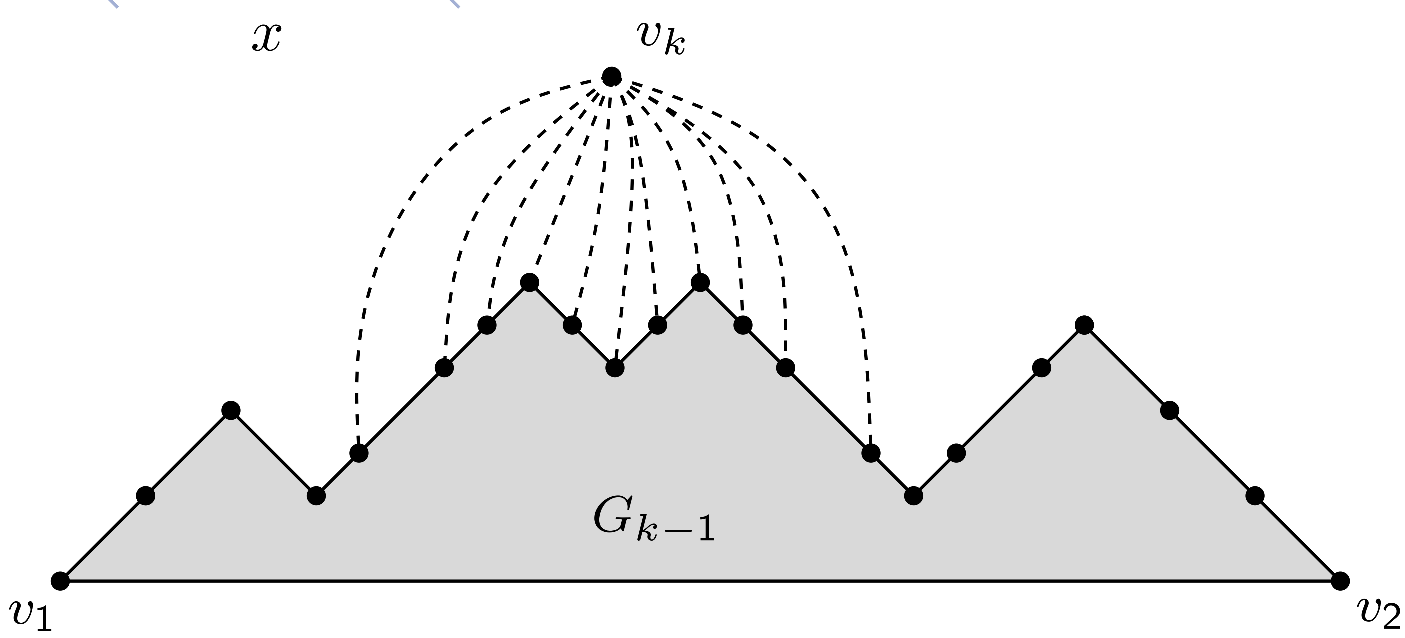
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



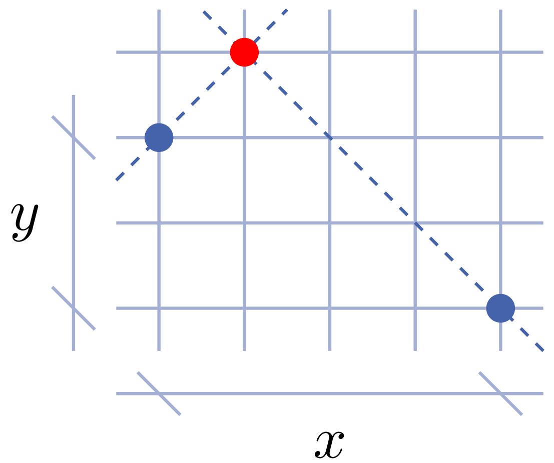
Algorithm invariants: G_{k-1} is drawn such that

- 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



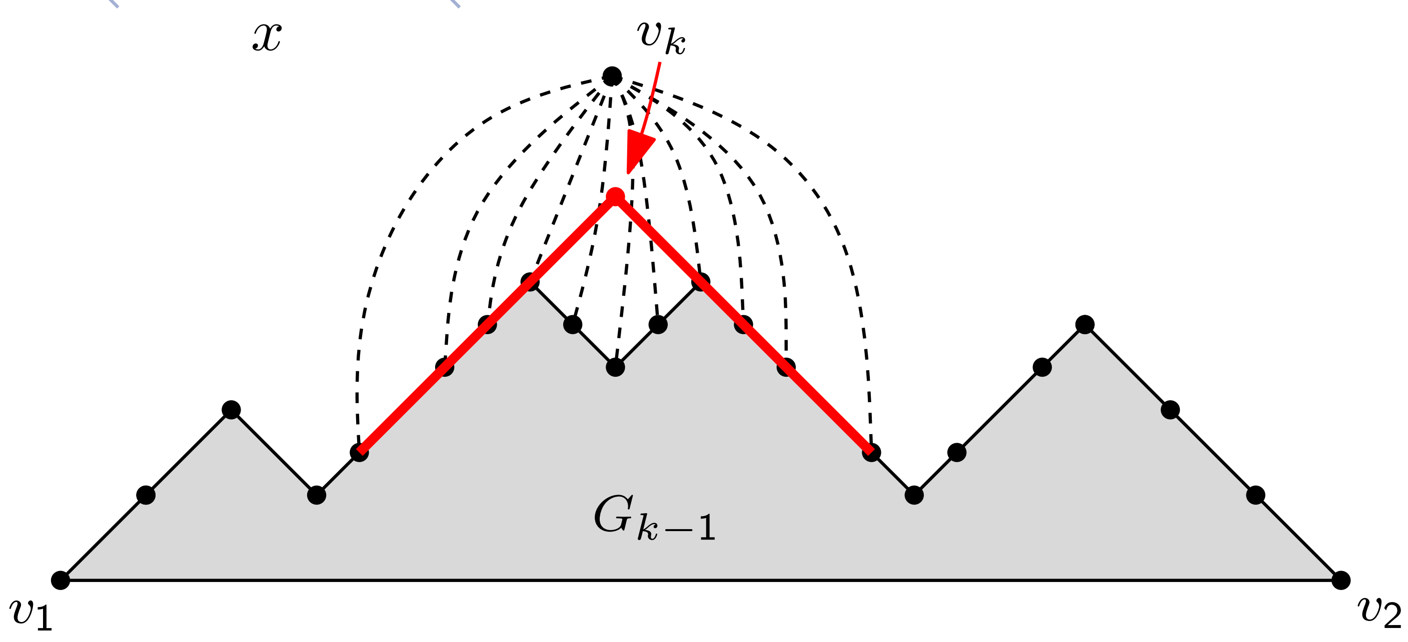
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance

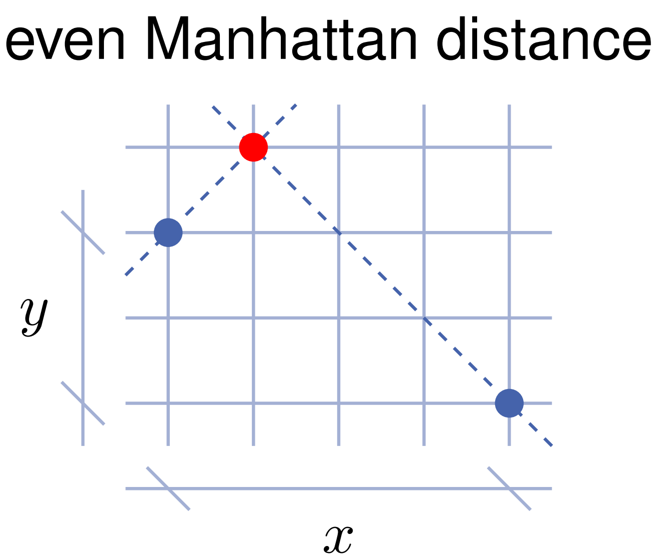


Algorithm invariants: G_{k-1} is drawn such that

- 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

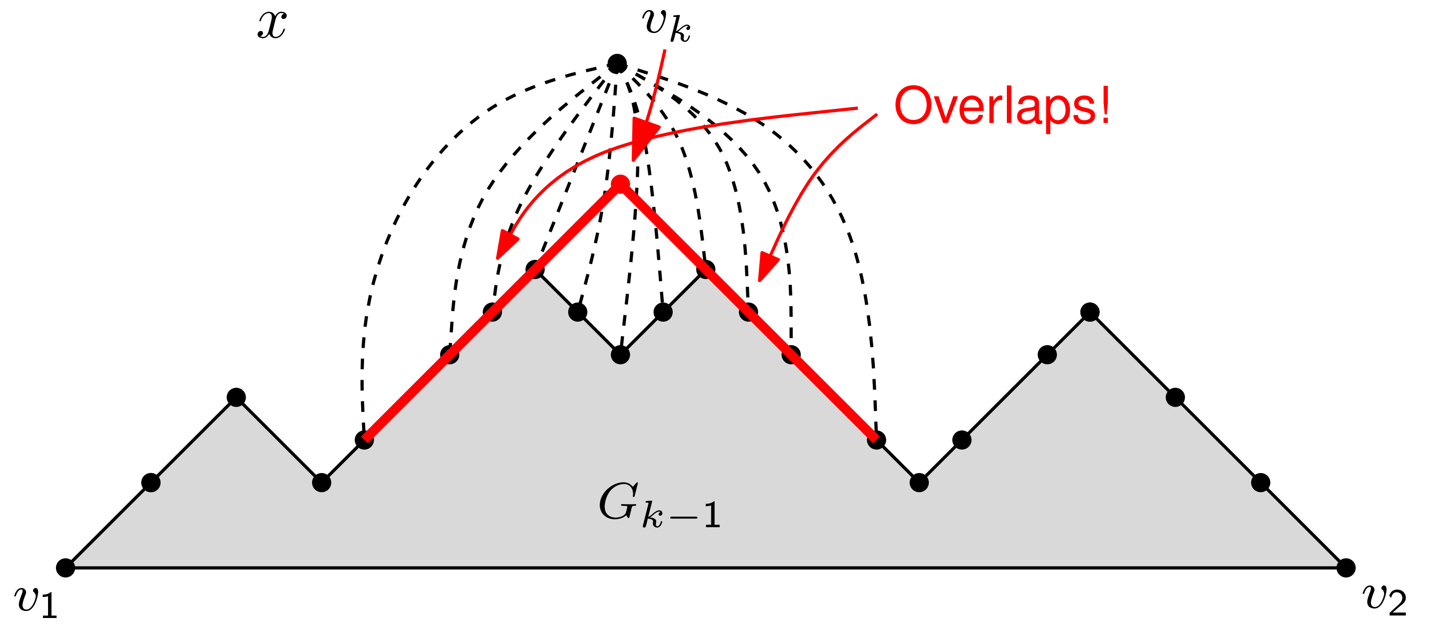


De Fraysseix Pach Pollack (Shift) Algorithm



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

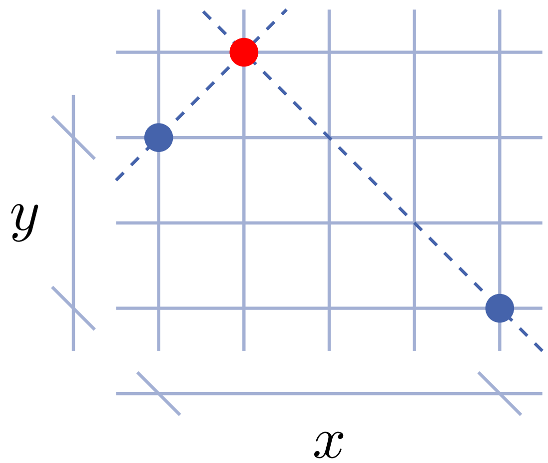
- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$
- Boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1



15 - 5

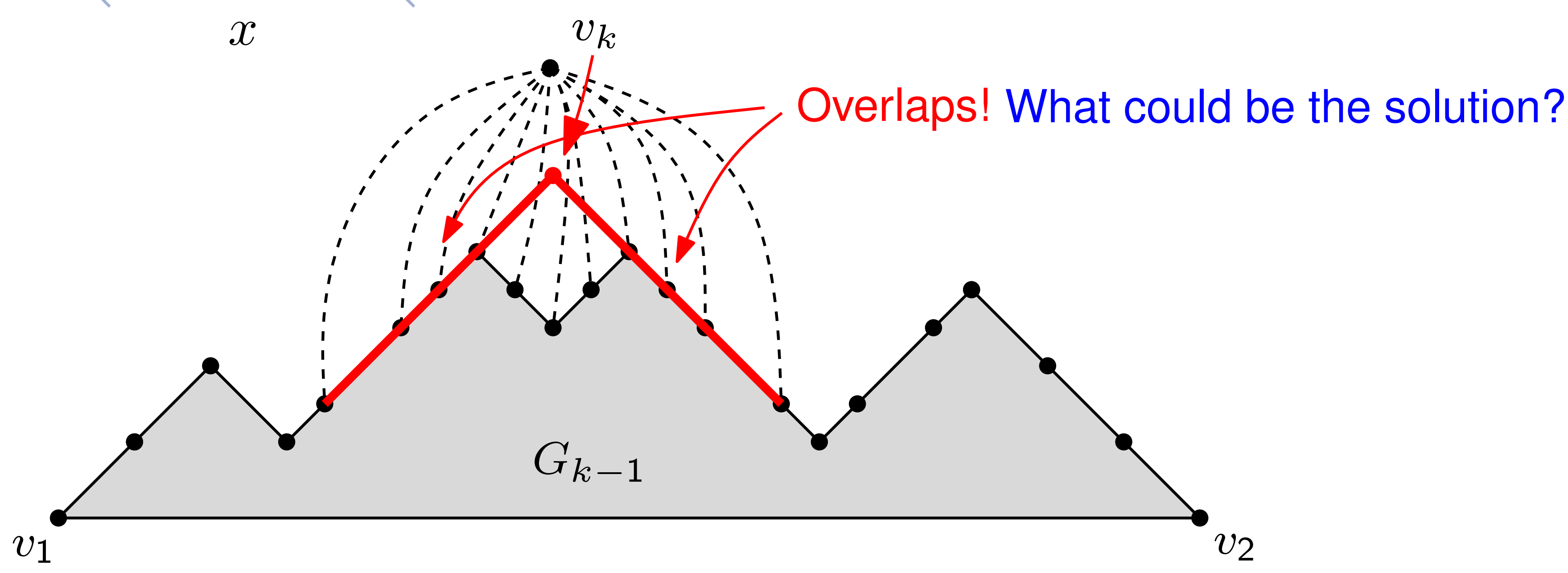
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

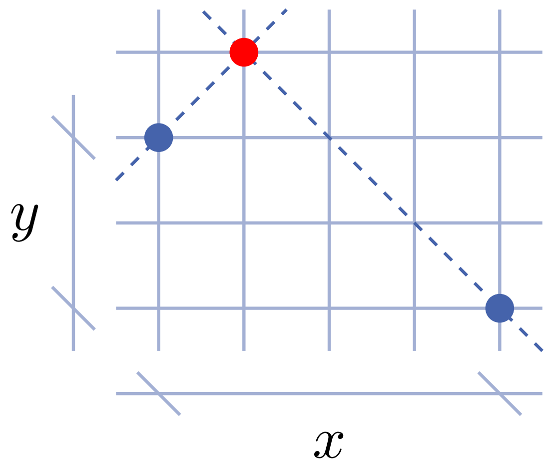
- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$
- Boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn x-monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1,v_2)) is drawn with slopes ± 1



15 - 6

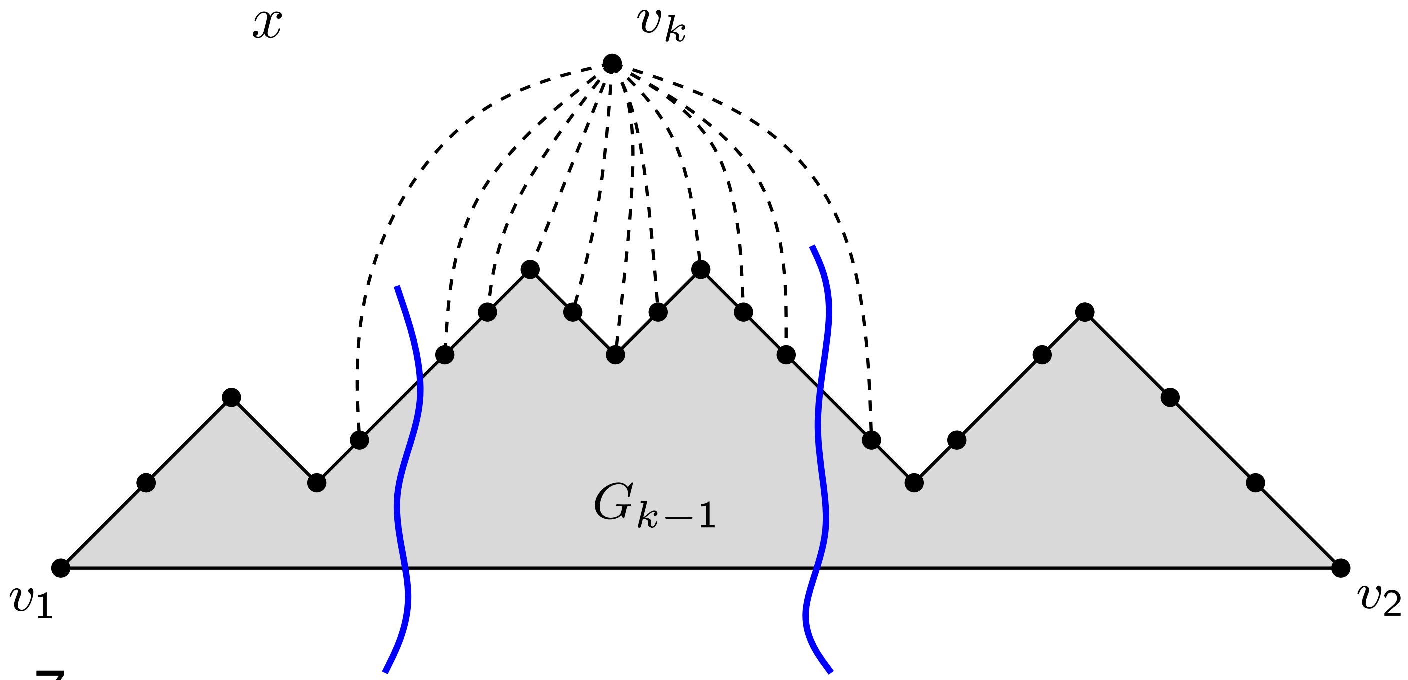
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



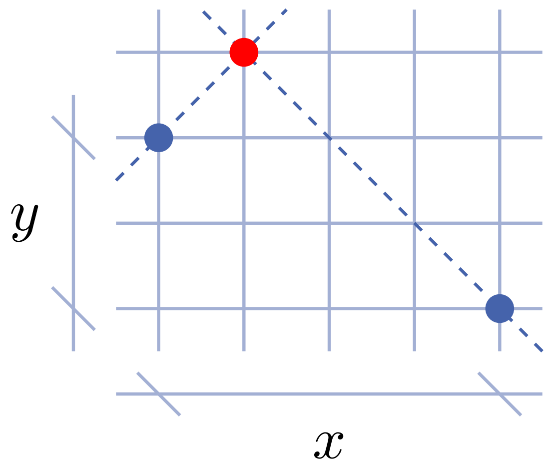
Algorithm invariants: G_{k-1} is drawn such that

- 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



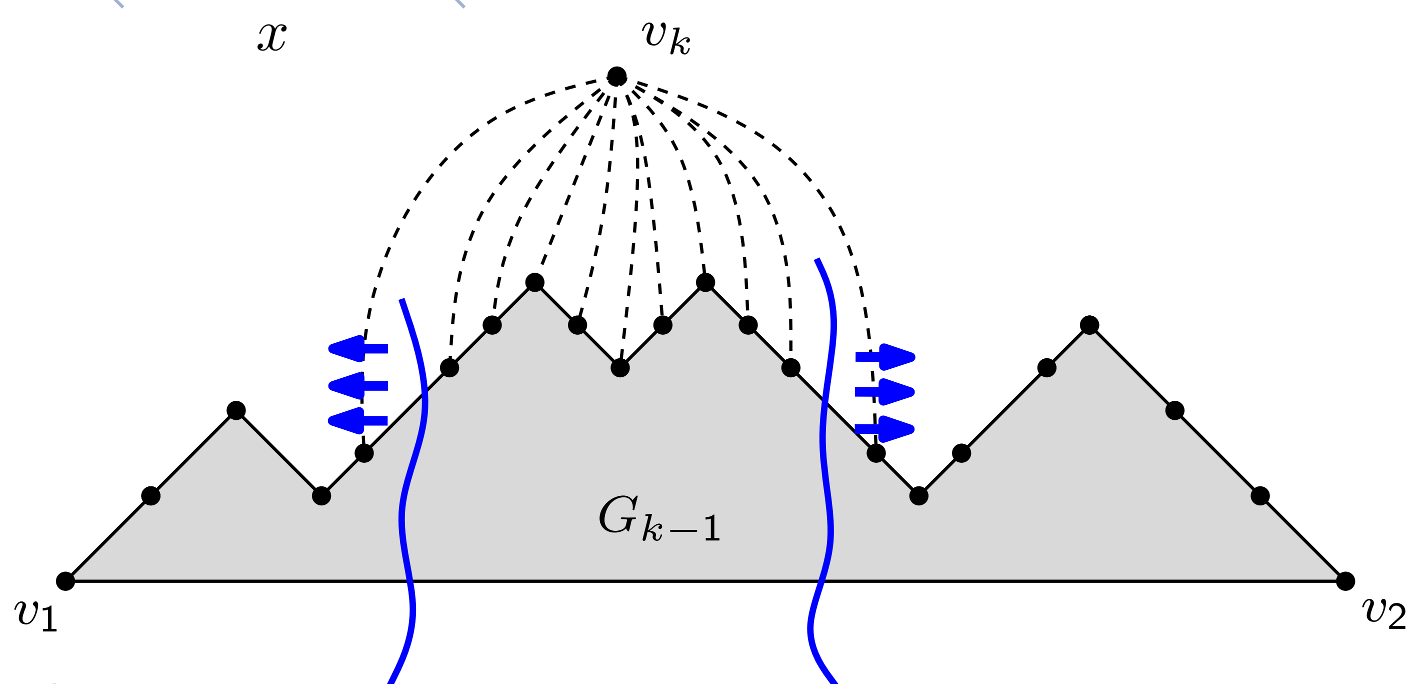
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



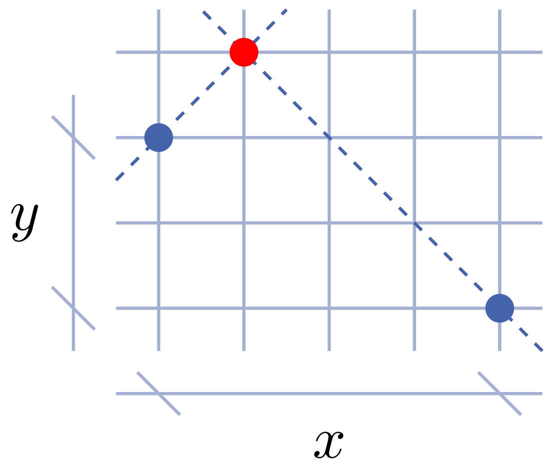
Algorithm invariants: G_{k-1} is drawn such that

- 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



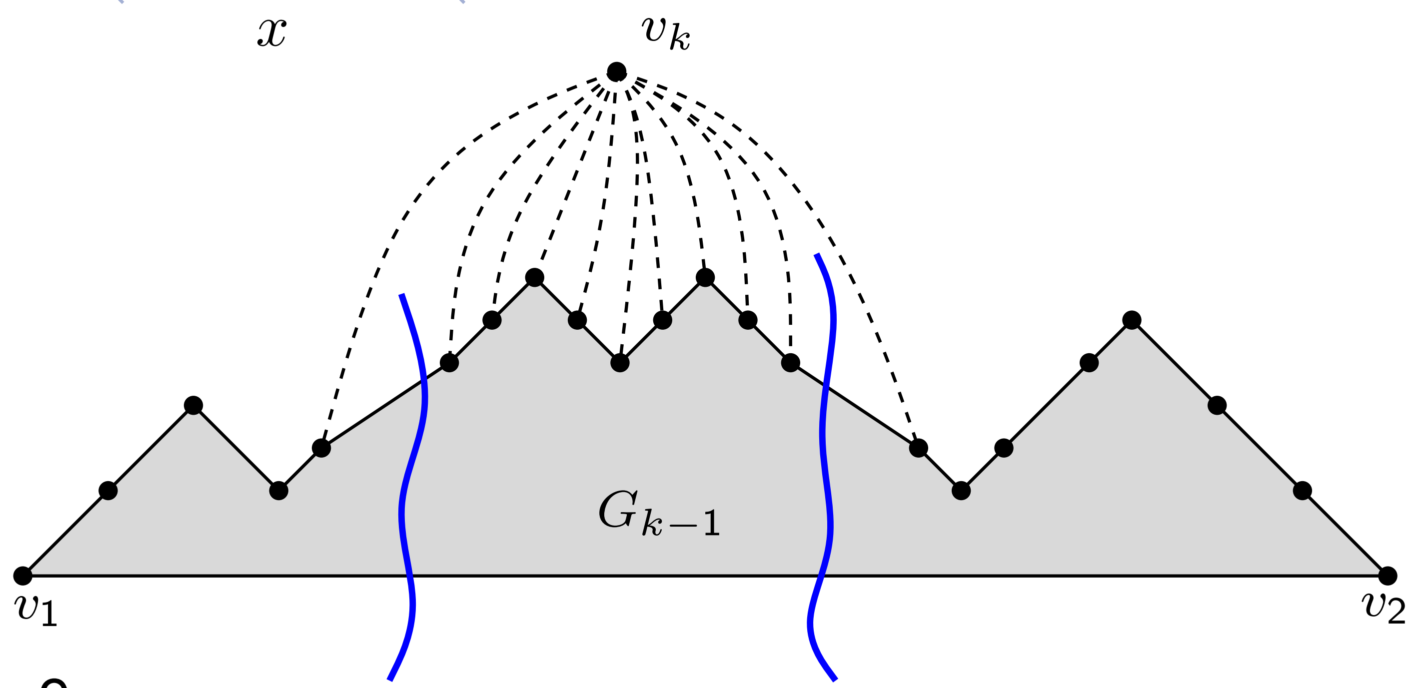
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



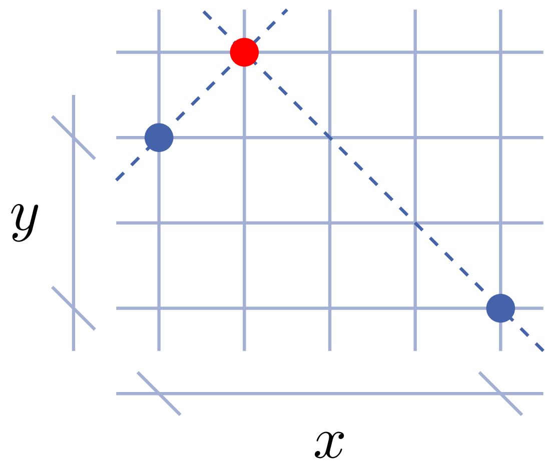
Algorithm invariants: G_{k-1} is drawn such that

- 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



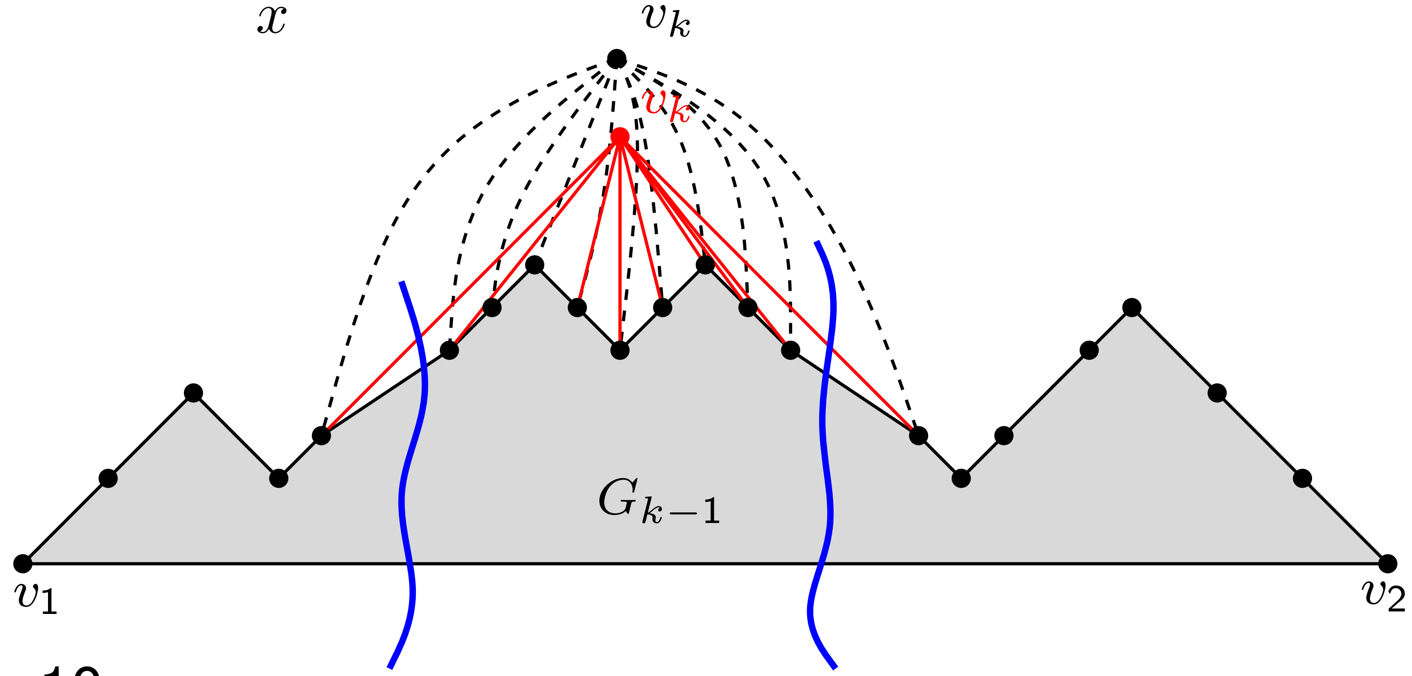
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

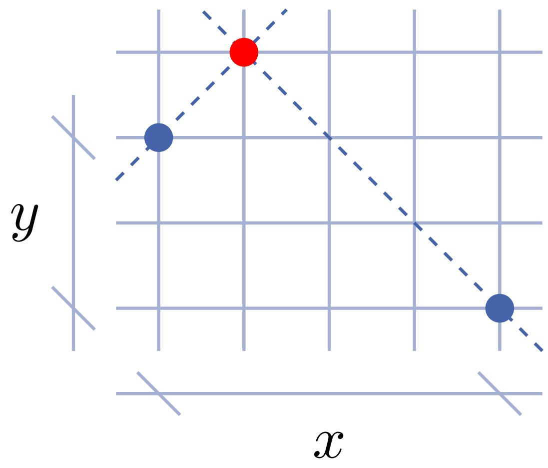
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 10

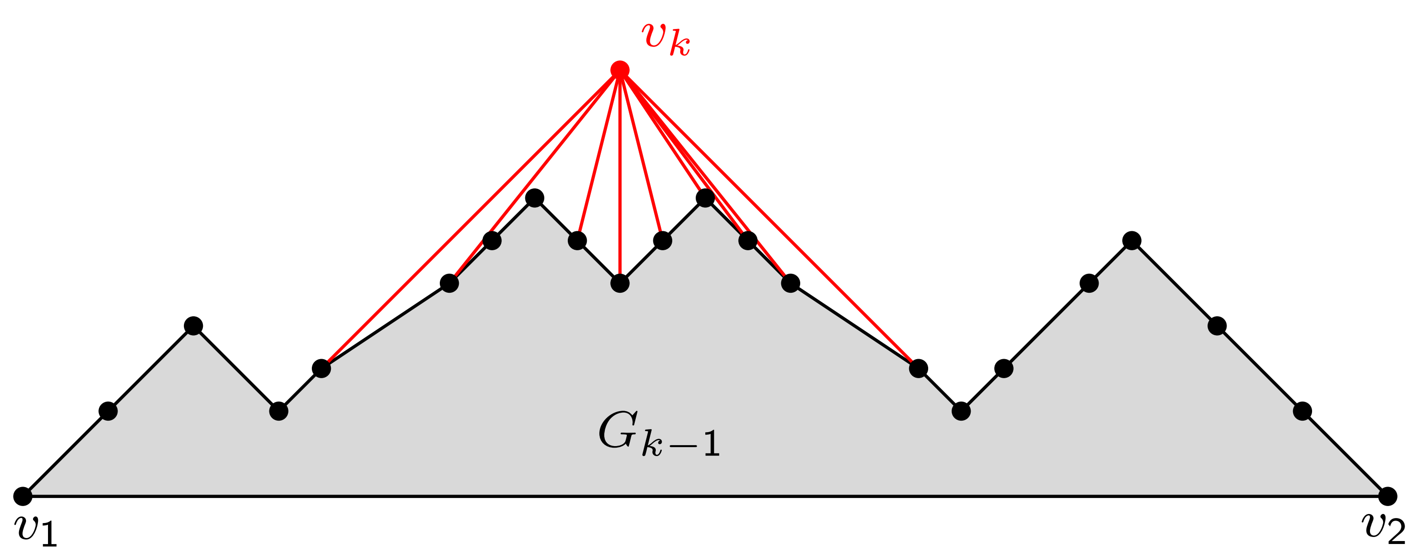
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

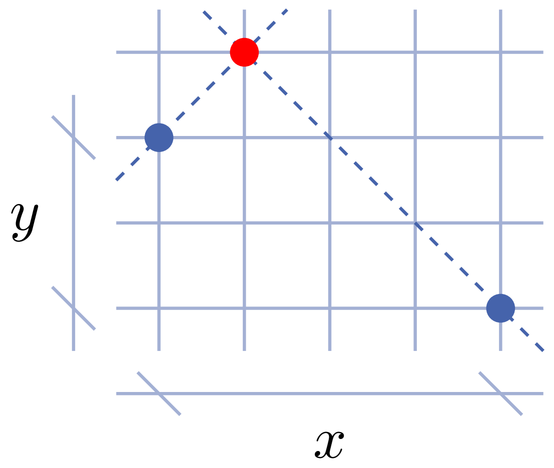
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 11

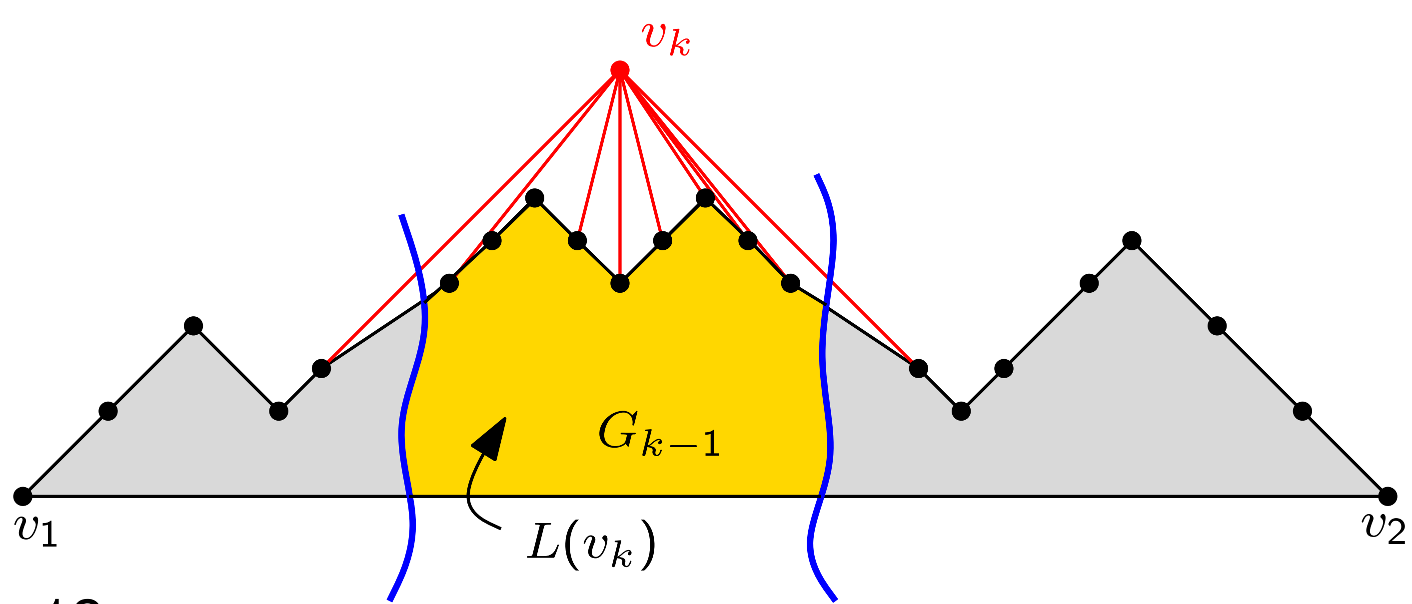
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance

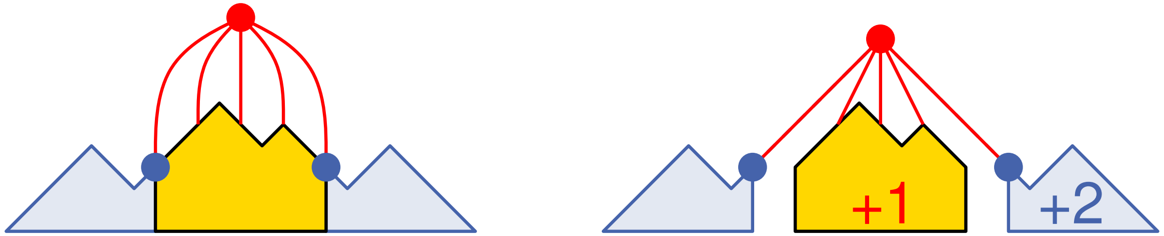


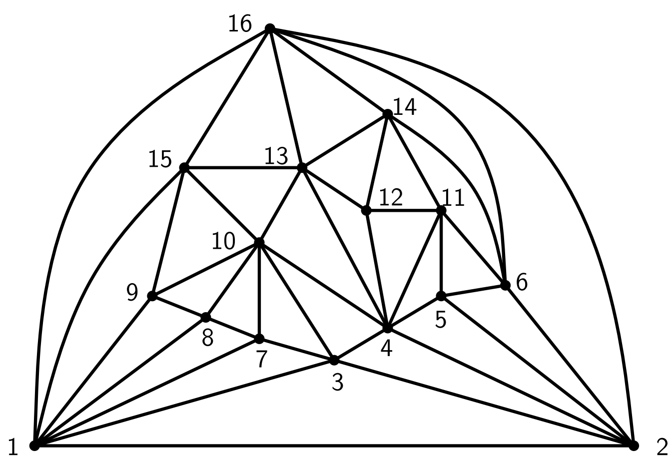
Algorithm invariants: G_{k-1} is drawn such that

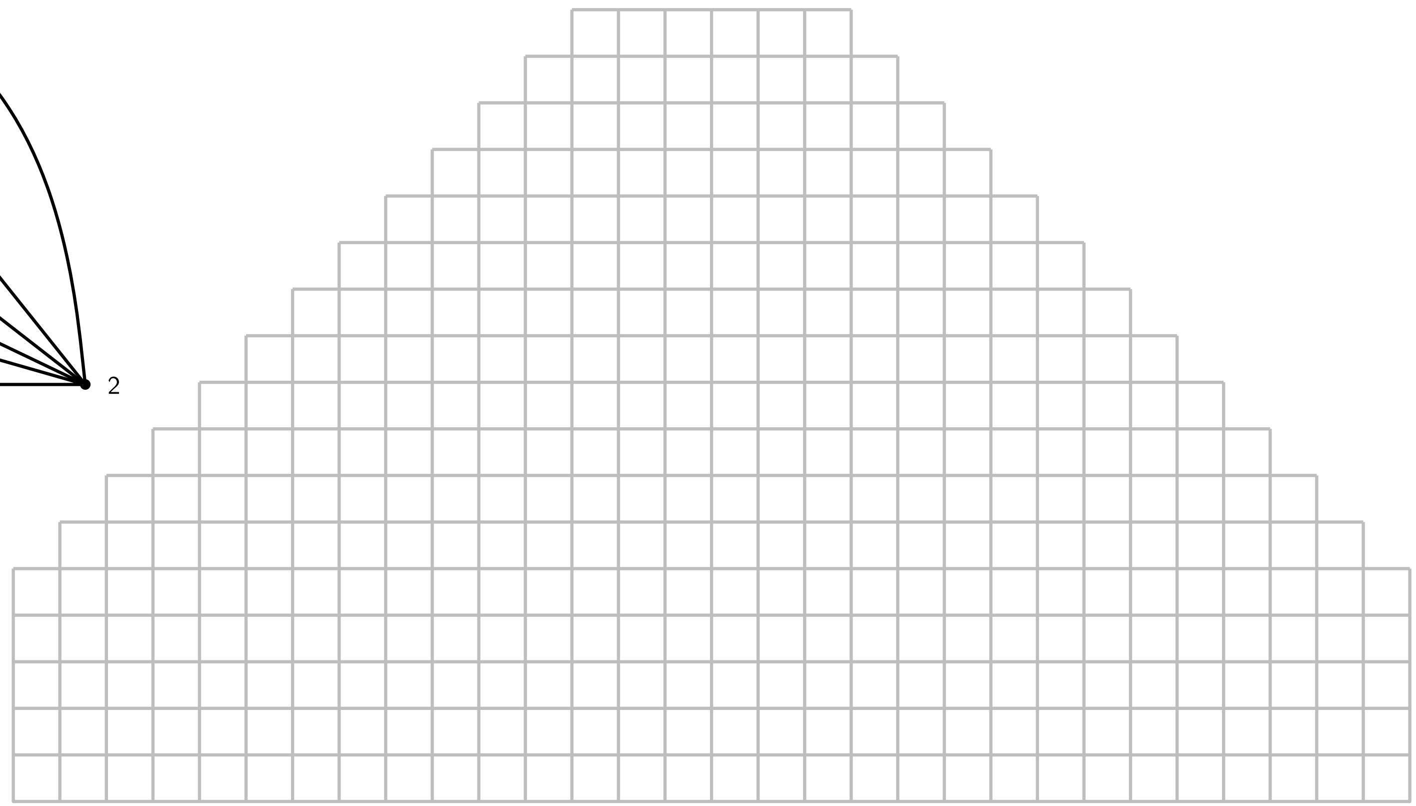
- v_1 is on $(0,0)$, v_2 is on $(2k-6,0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



De Fraysseix Pach Pollack (Shift) Algorithm

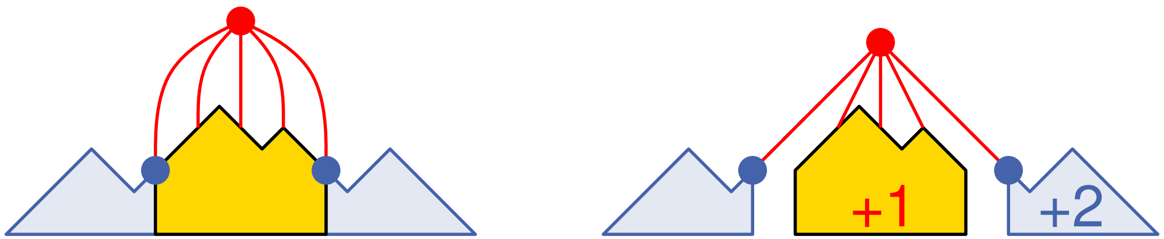


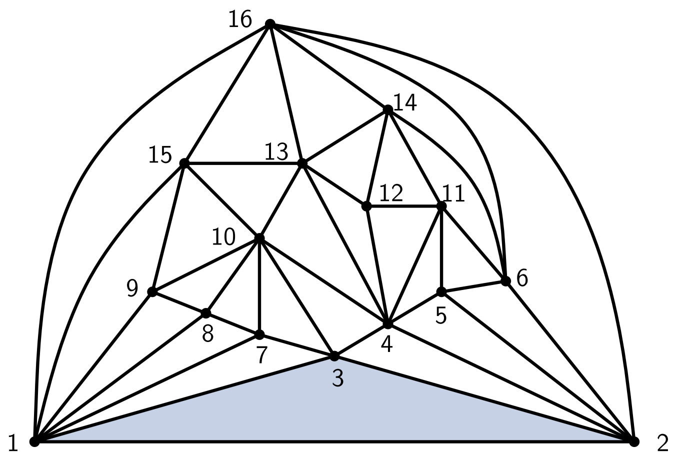


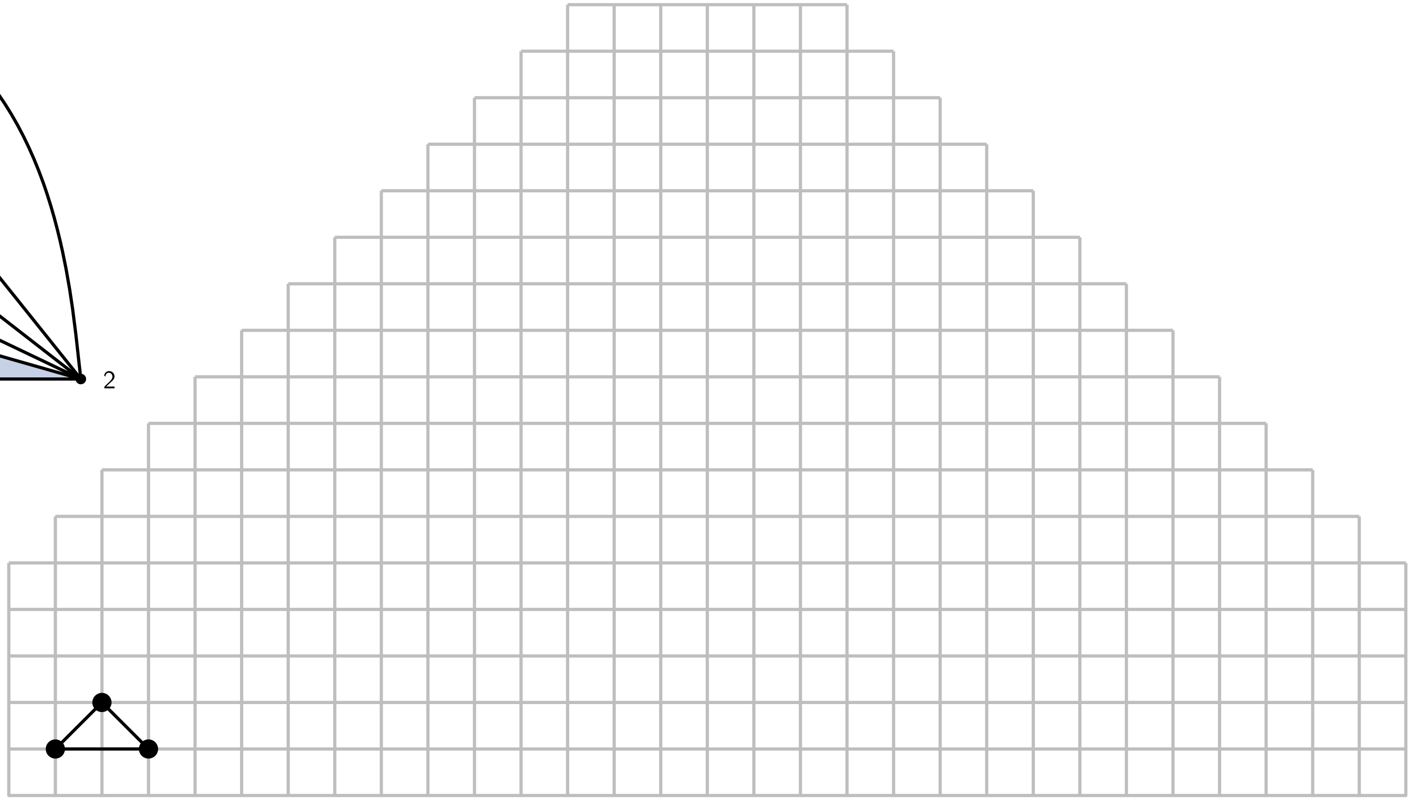


16 - 1

De Fraysseix Pach Pollack (Shift) Algorithm

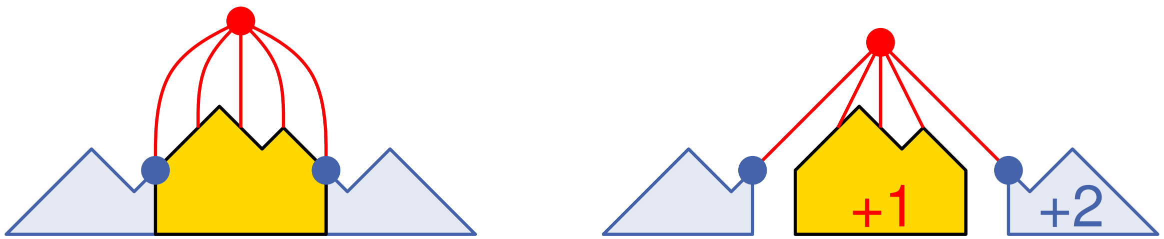


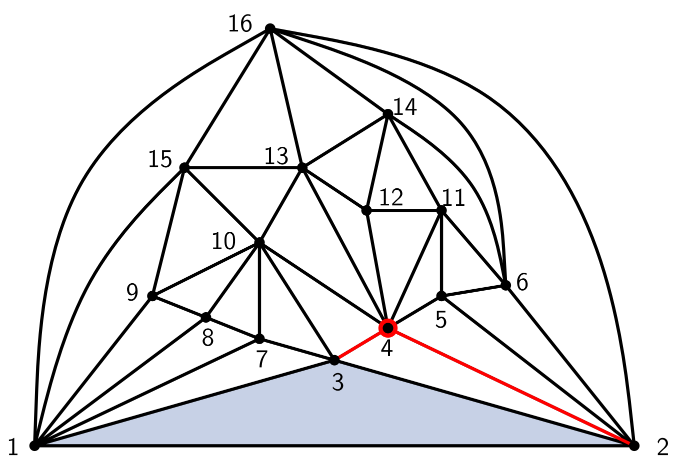


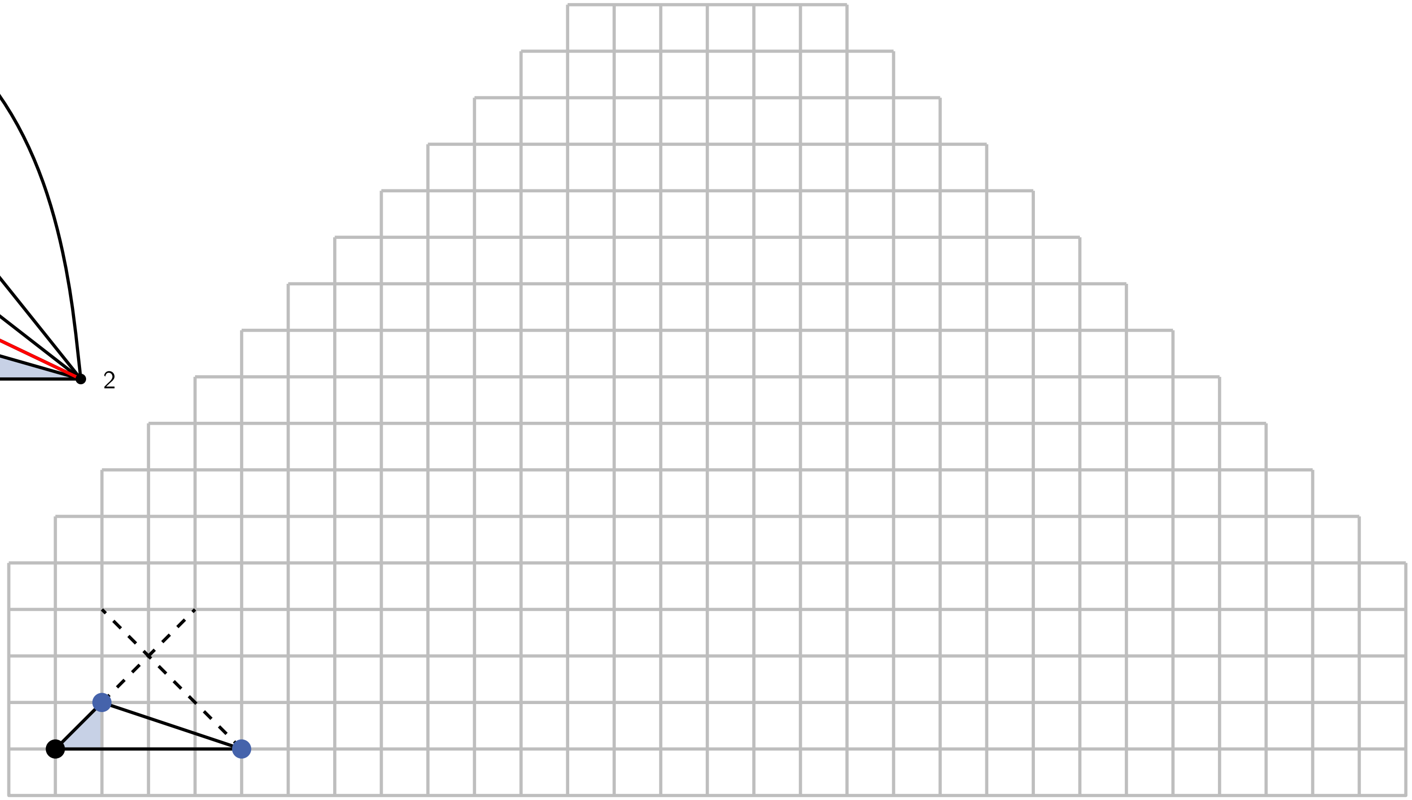


16 - 2

De Fraysseix Pach Pollack (Shift) Algorithm

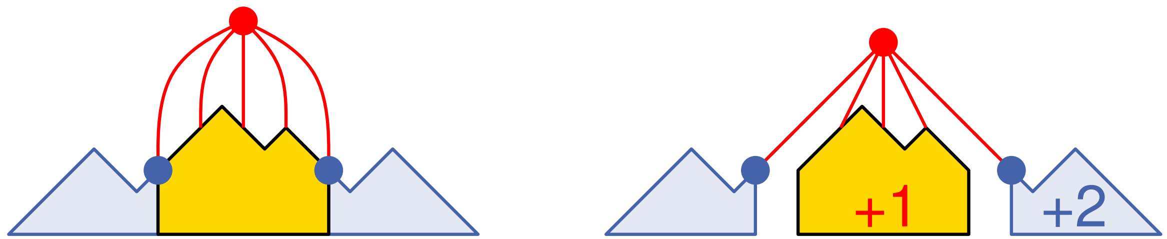


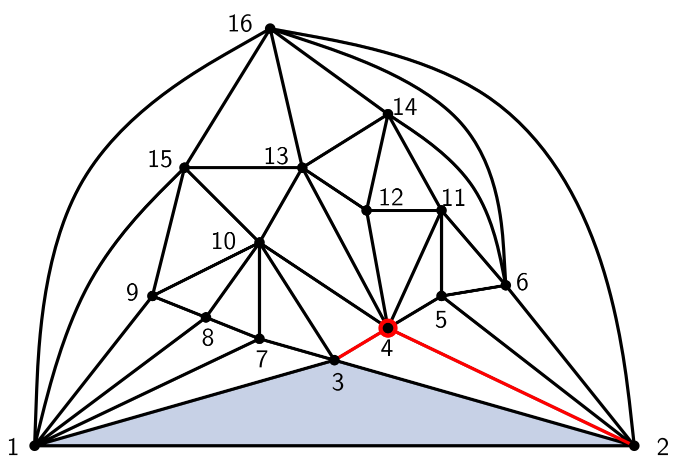


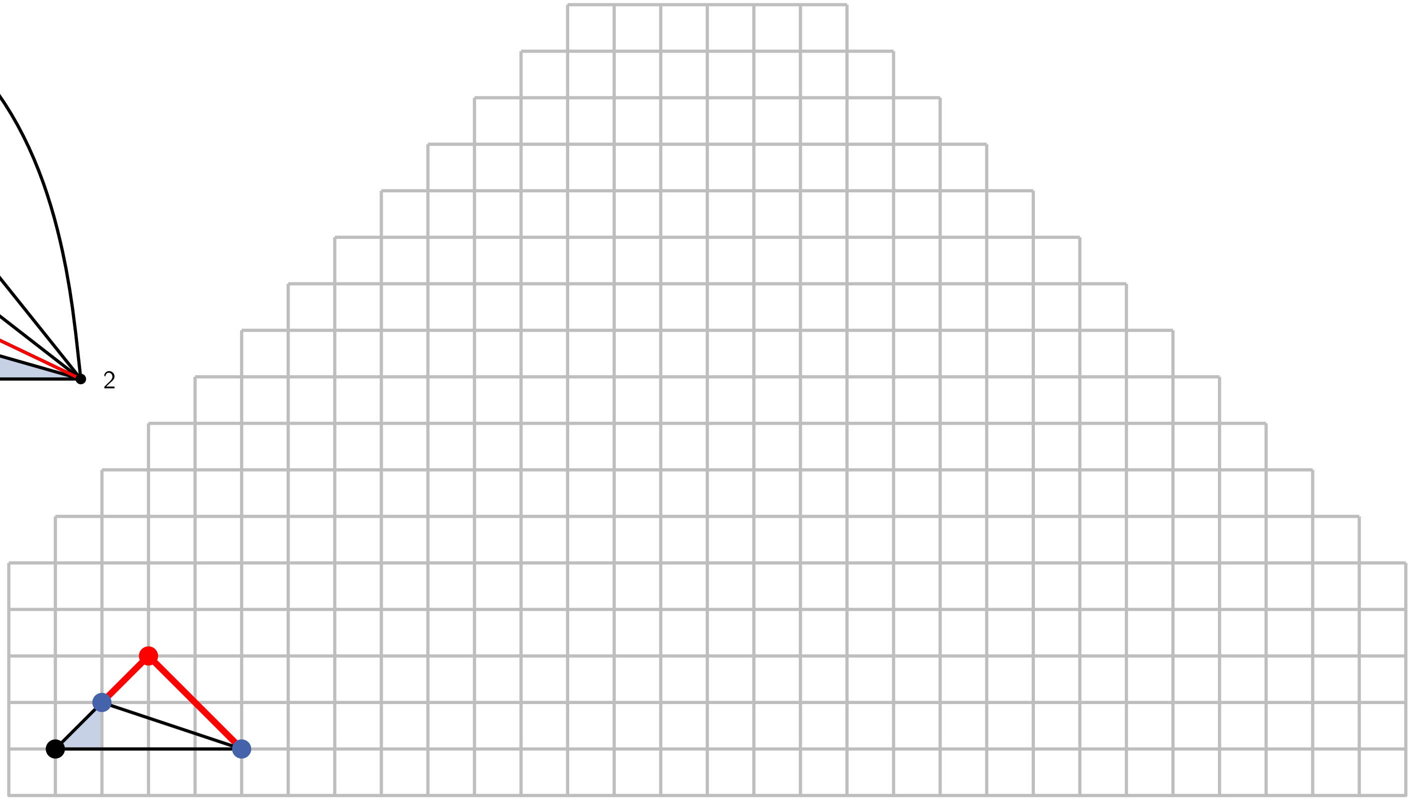


16 - 3

De Fraysseix Pach Pollack (Shift) Algorithm

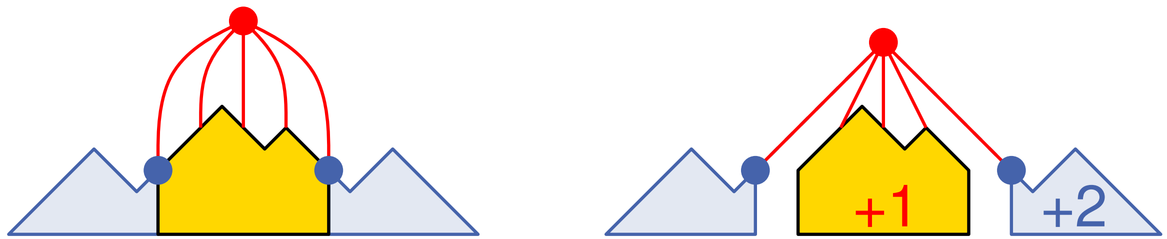


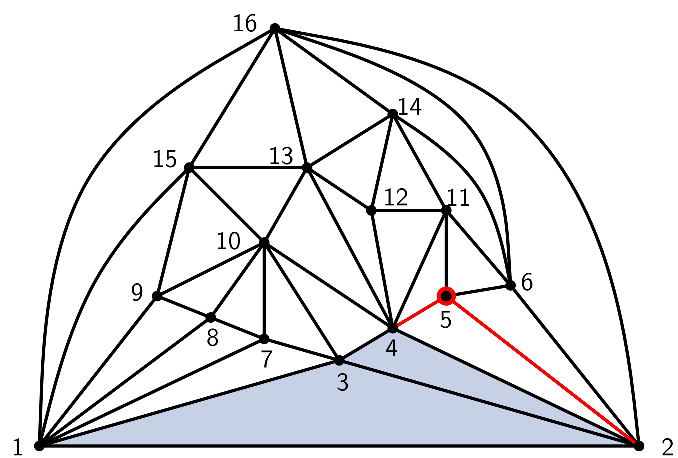


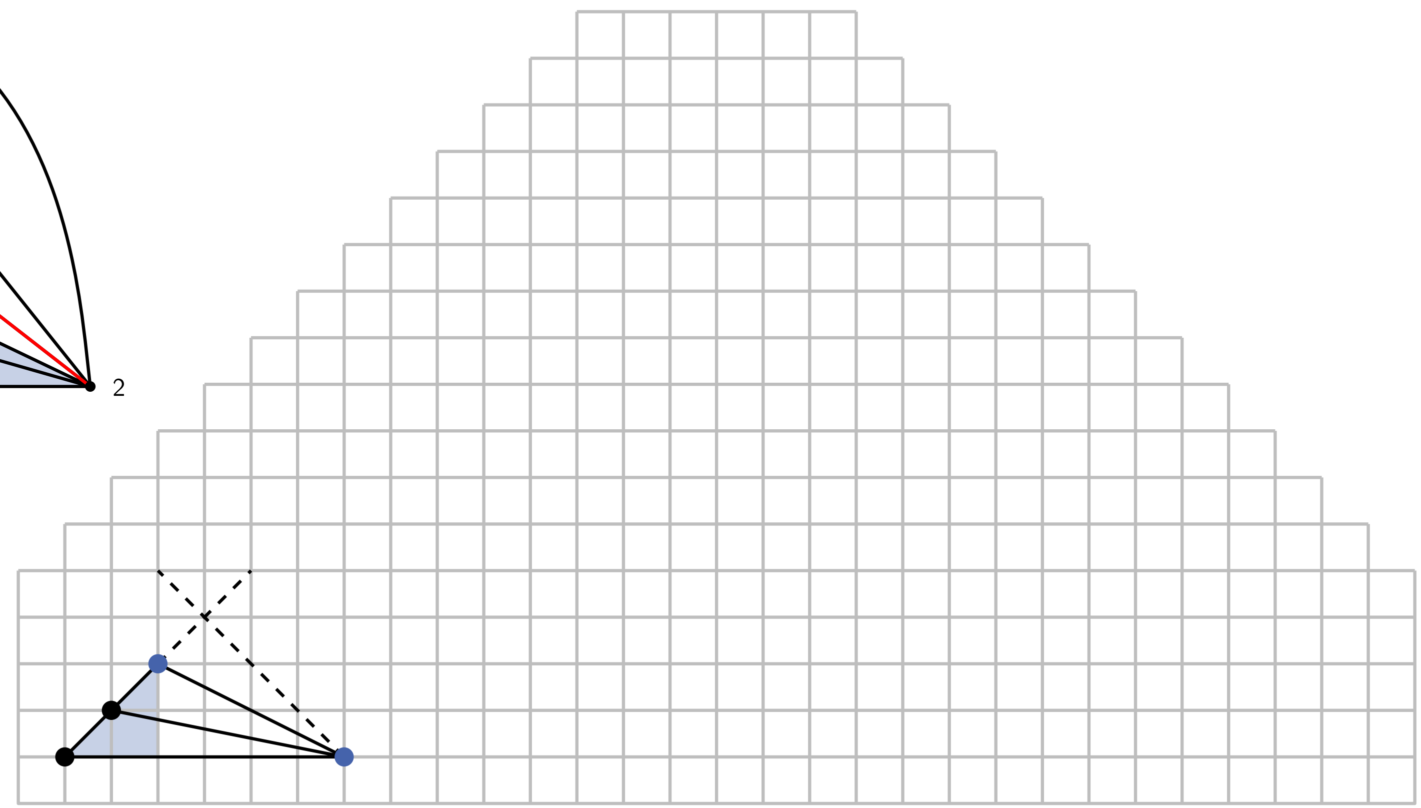


16 - 4

De Fraysseix Pach Pollack (Shift) Algorithm

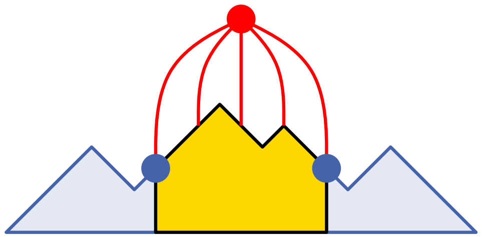


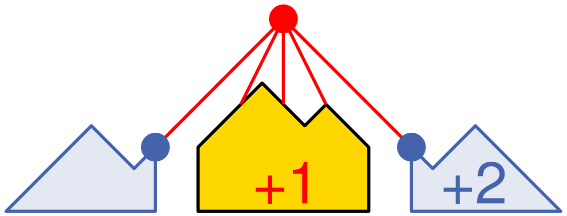


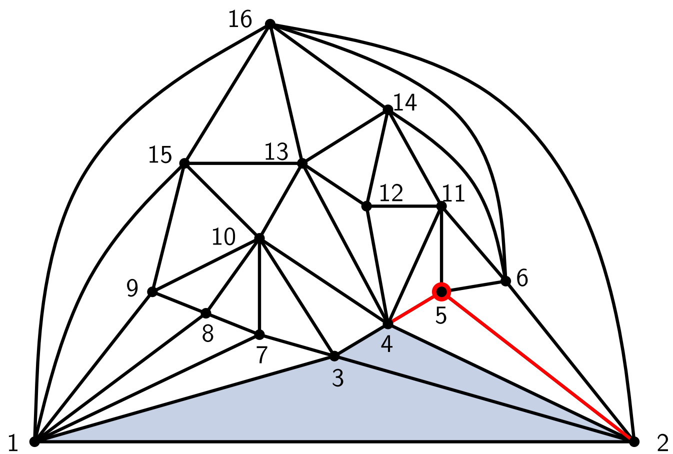


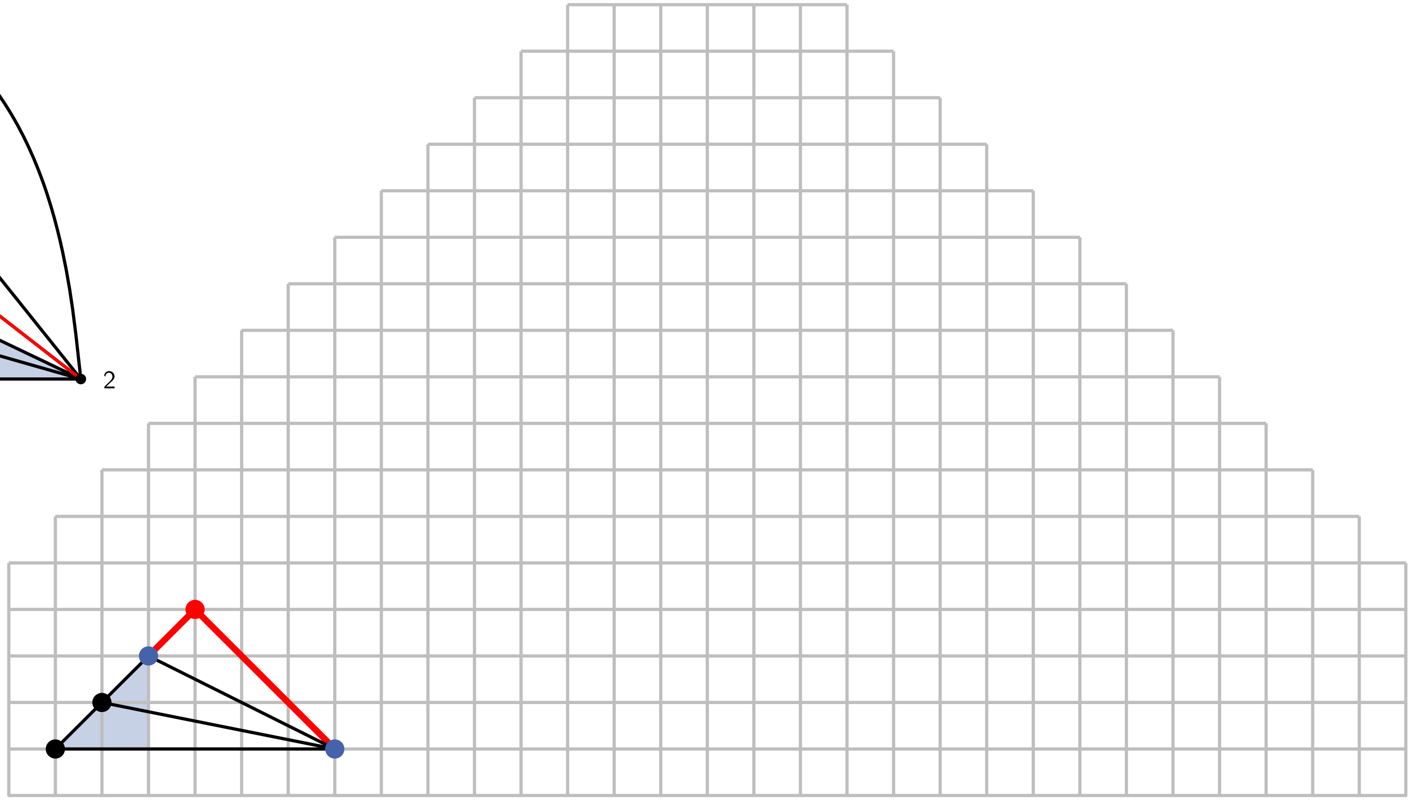
16 - 5

De Fraysseix Pach Pollack (Shift) Algorithm





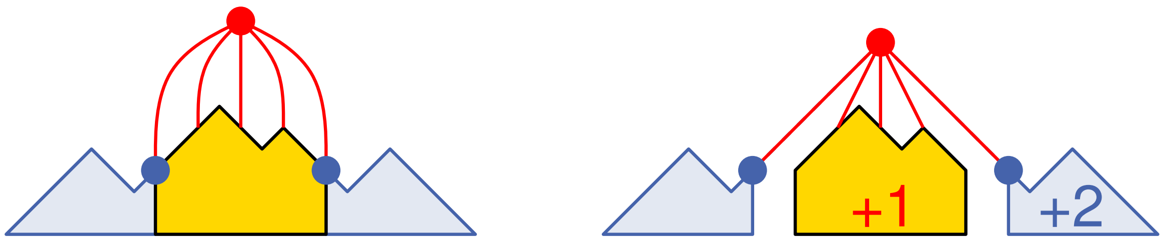


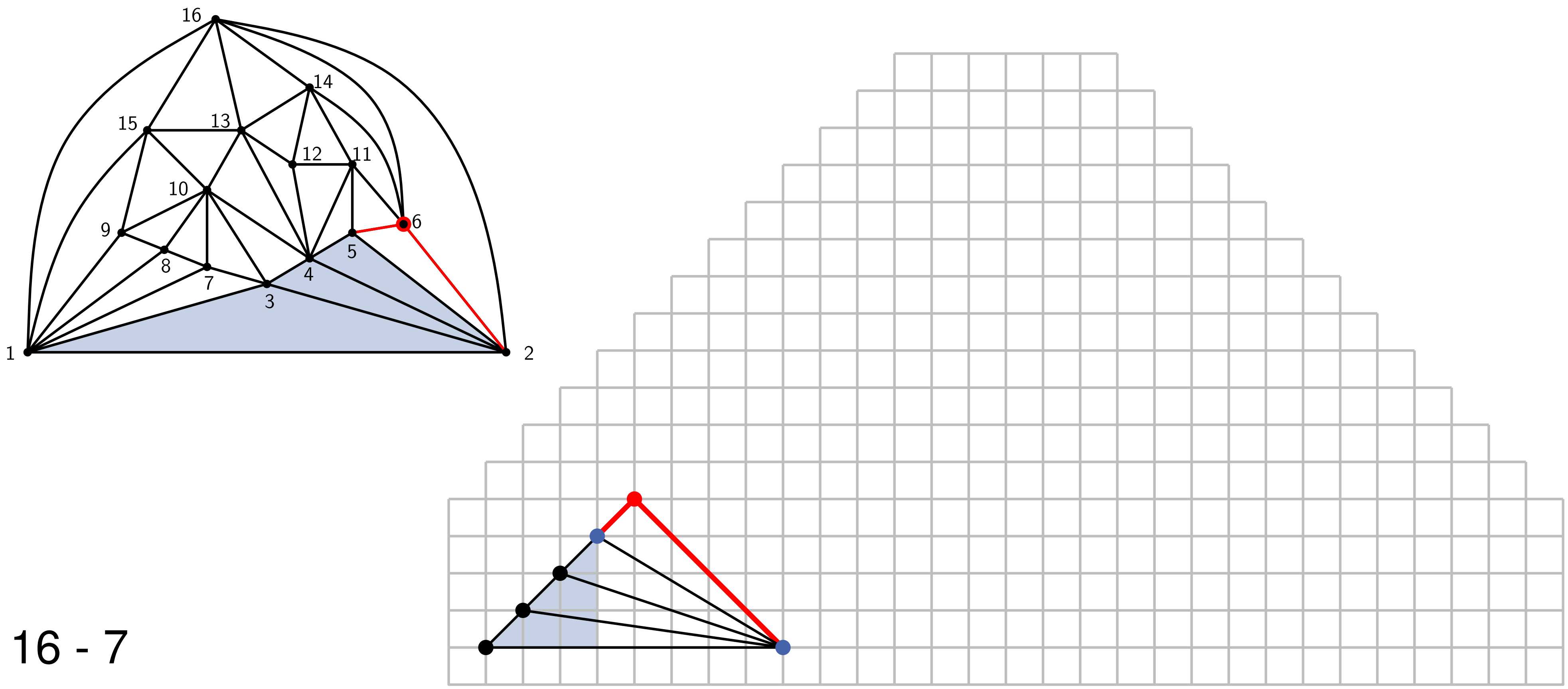


16 - 6

Institut für Theoretische Informatik
Lehrstuhl Algorithmik I

De Fraysseix Pach Pollack (Shift) Algorithm

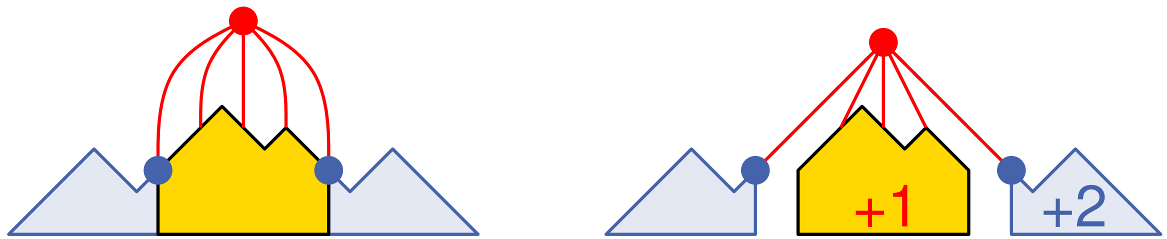


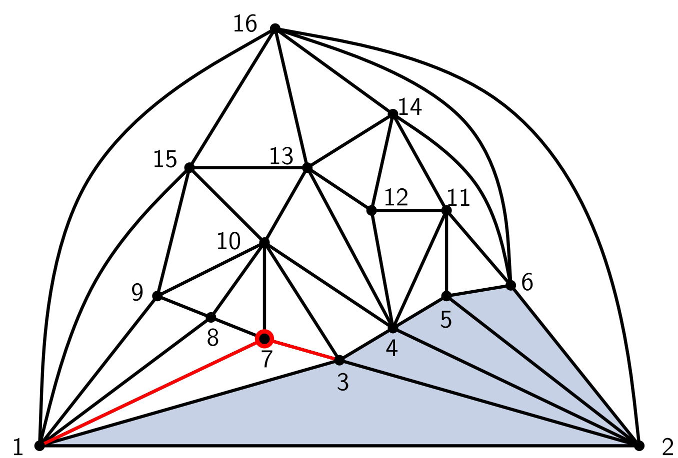


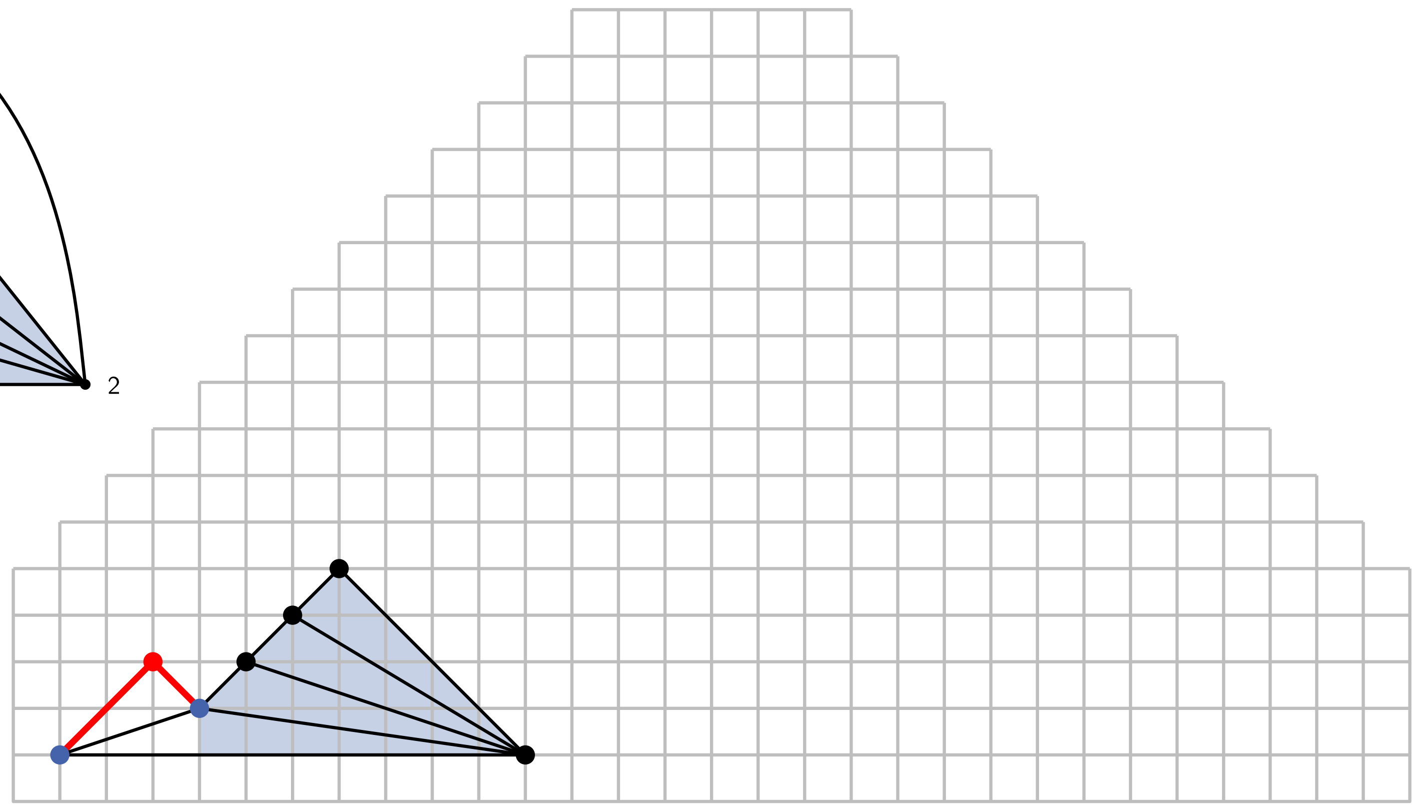
16 - 7

Institut für Theoretische Informatik
Lehrstuhl Algorithmik I

De Fraysseix Pach Pollack (Shift) Algorithm

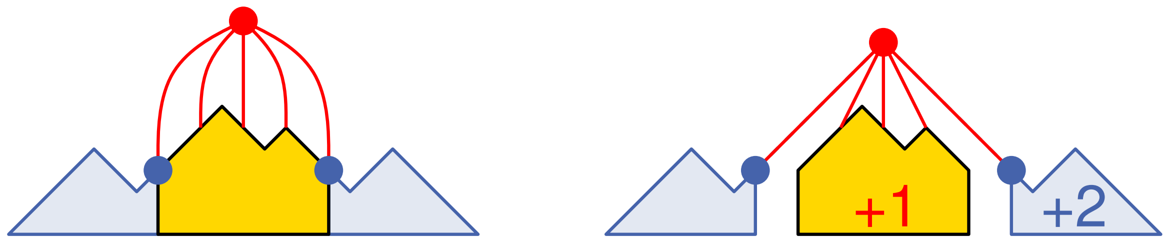


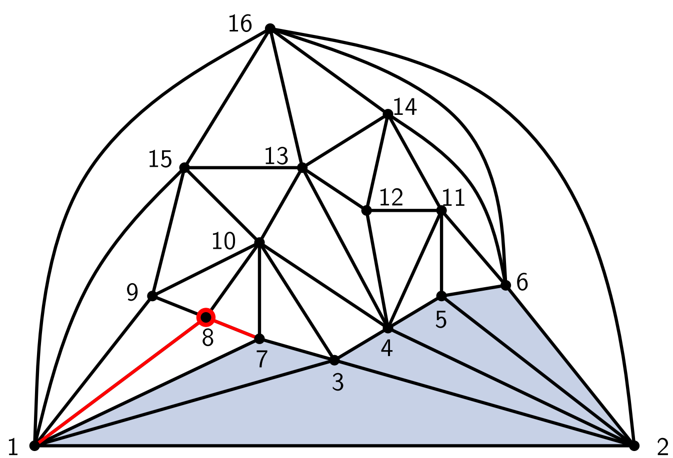


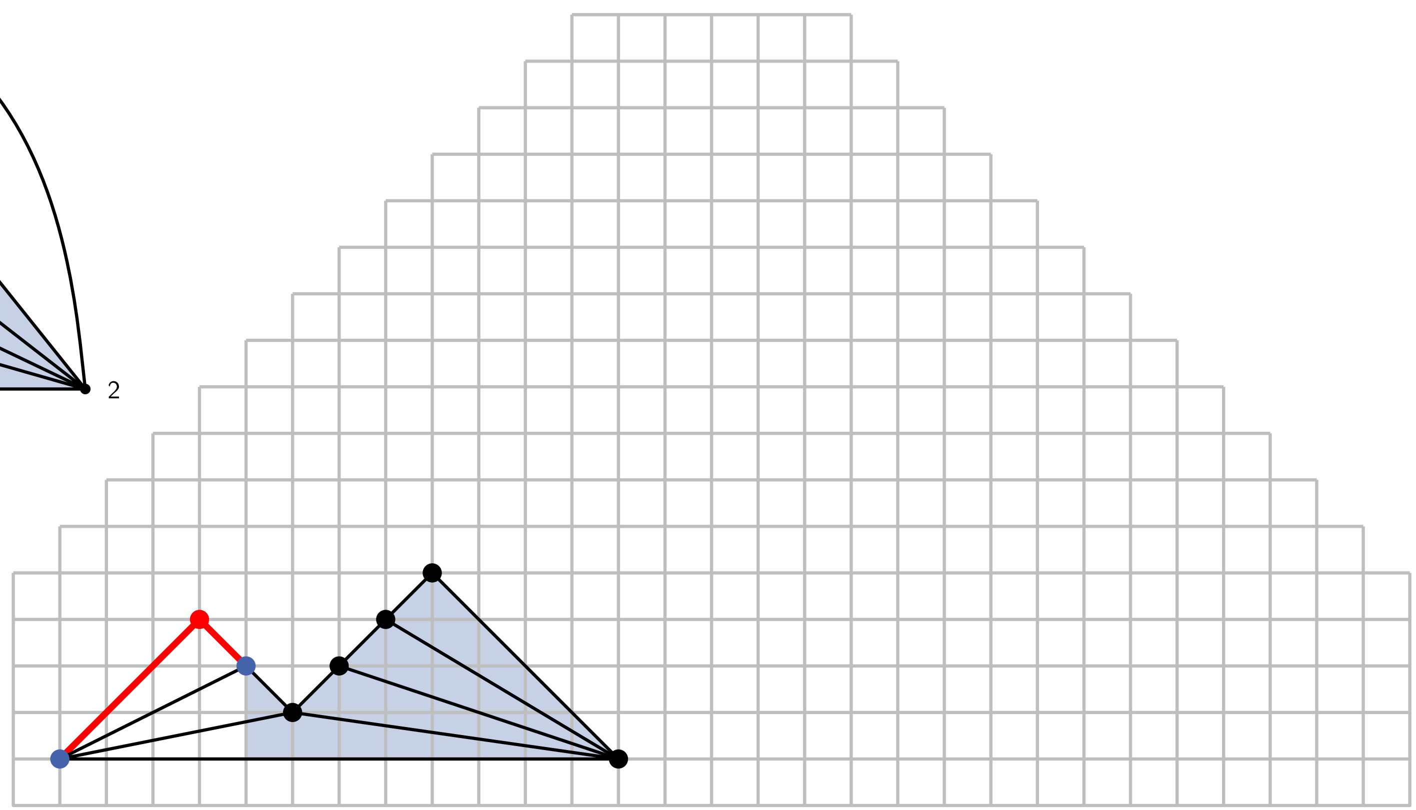


16 - 8

De Fraysseix Pach Pollack (Shift) Algorithm

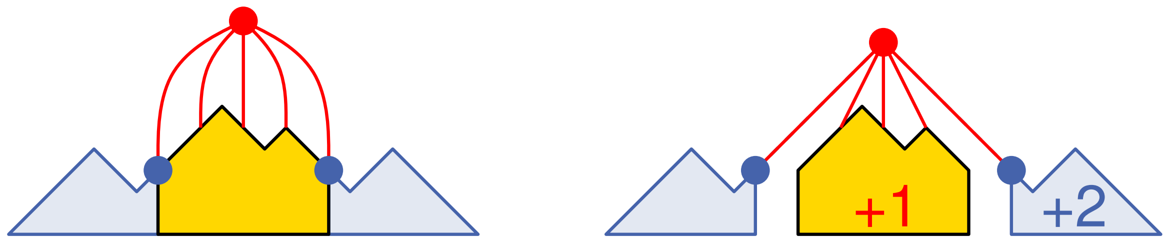


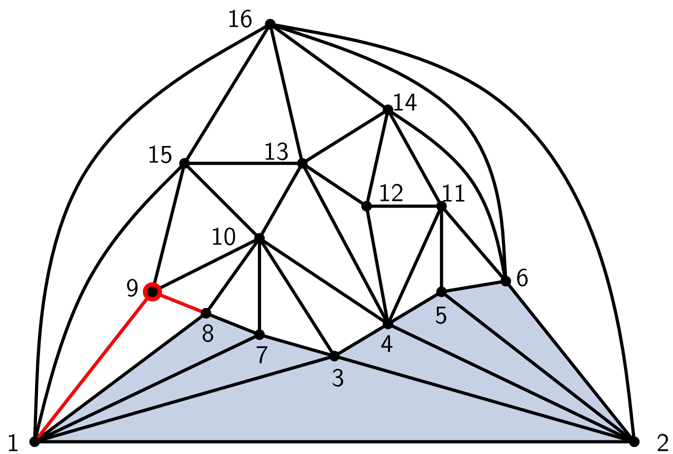


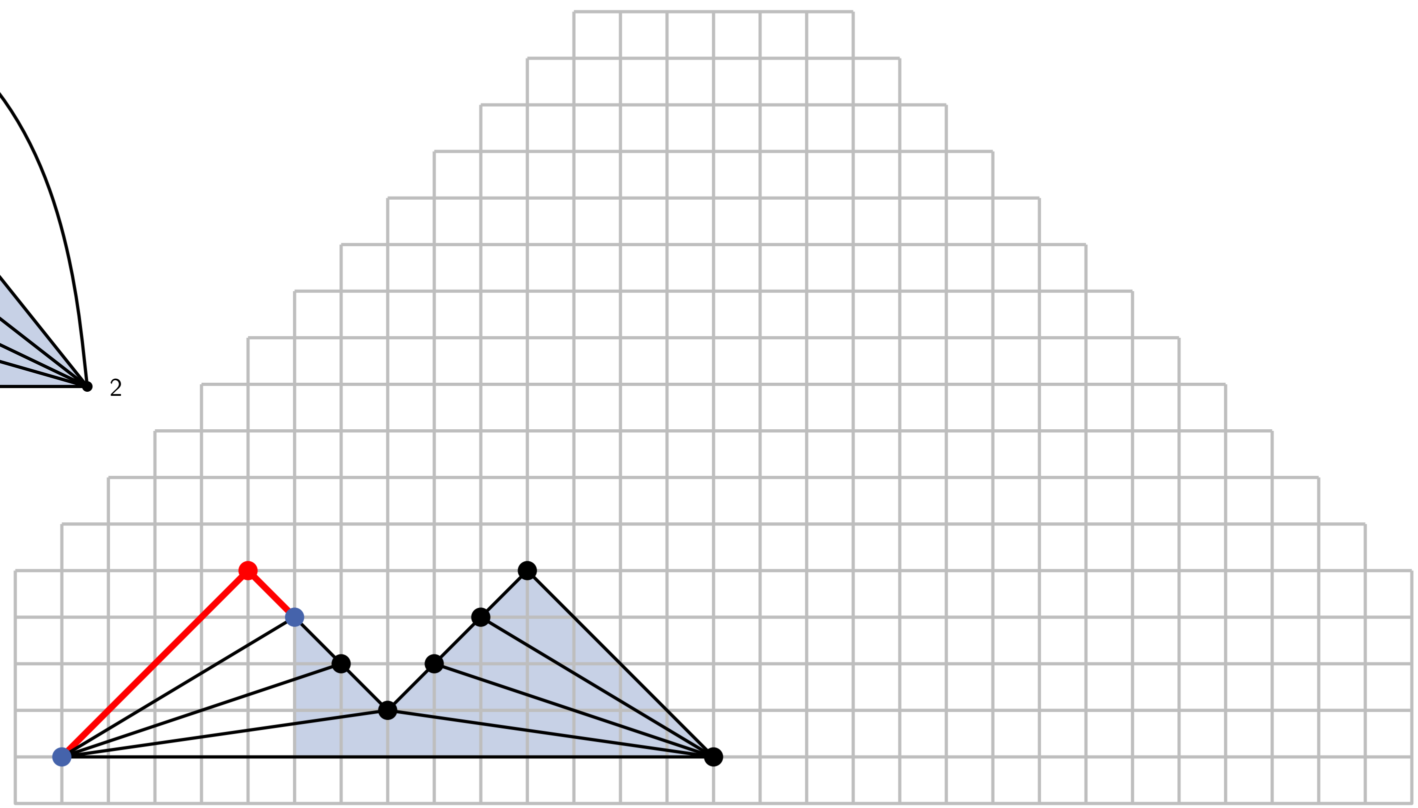


16 - 9

De Fraysseix Pach Pollack (Shift) Algorithm

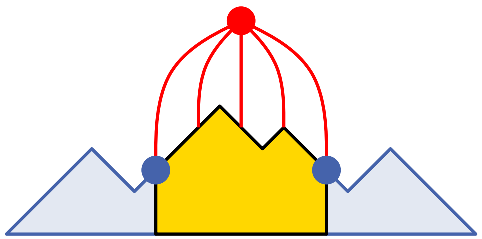


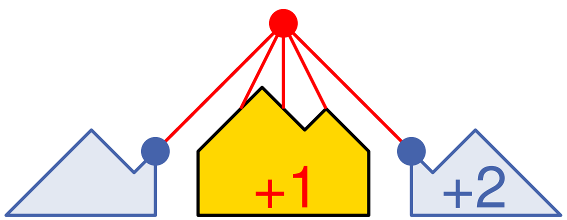


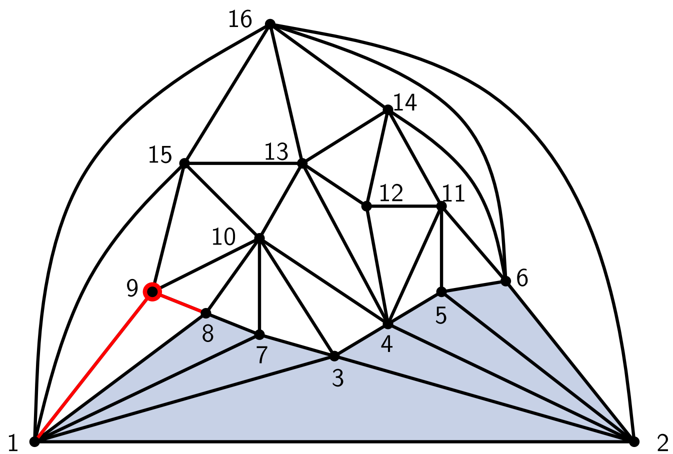


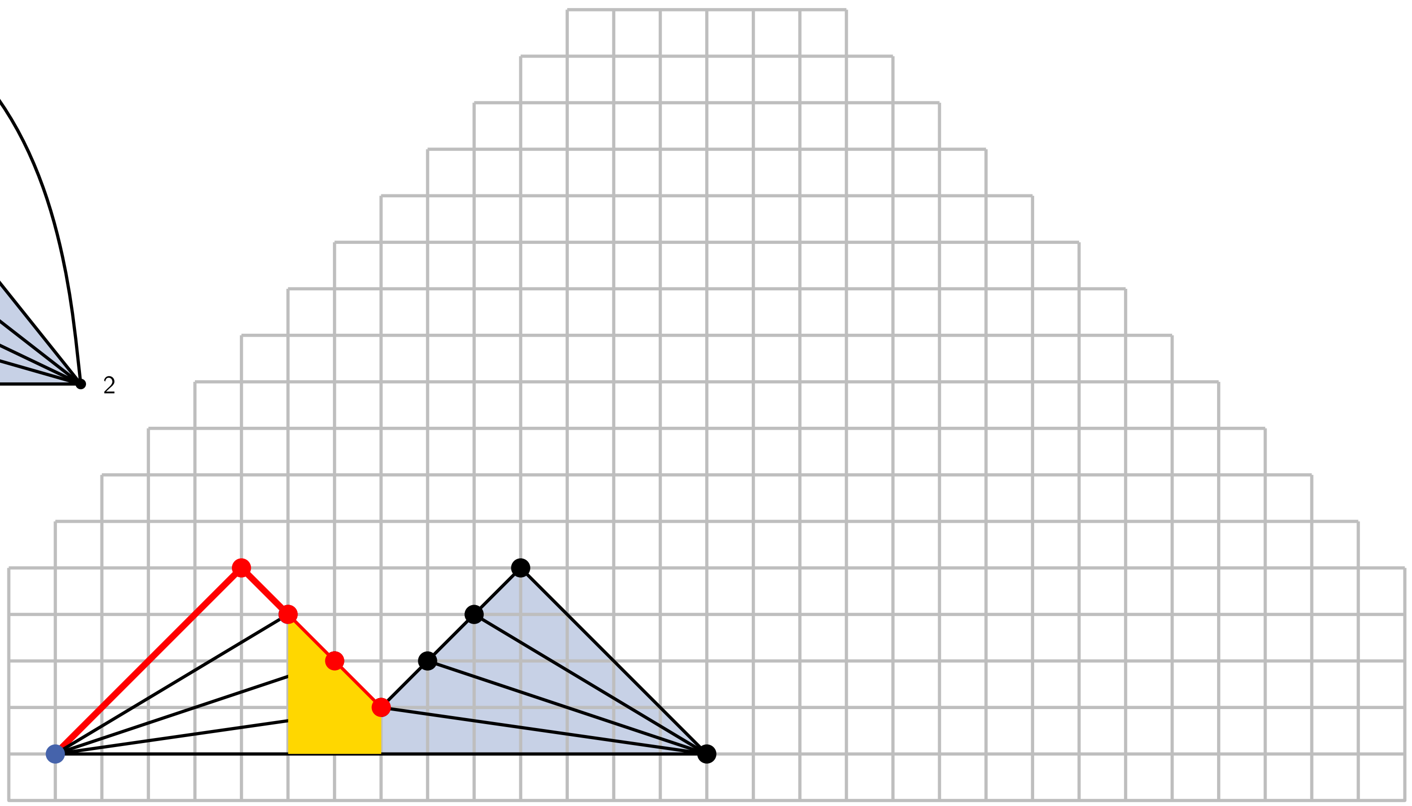
16 - 10

De Fraysseix Pach Pollack (Shift) Algorithm



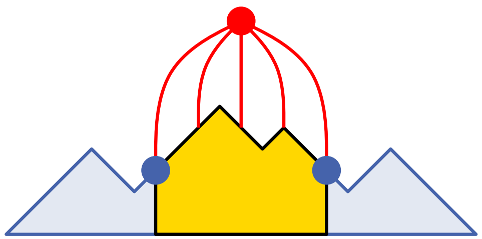


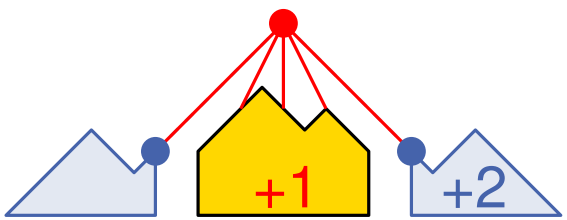


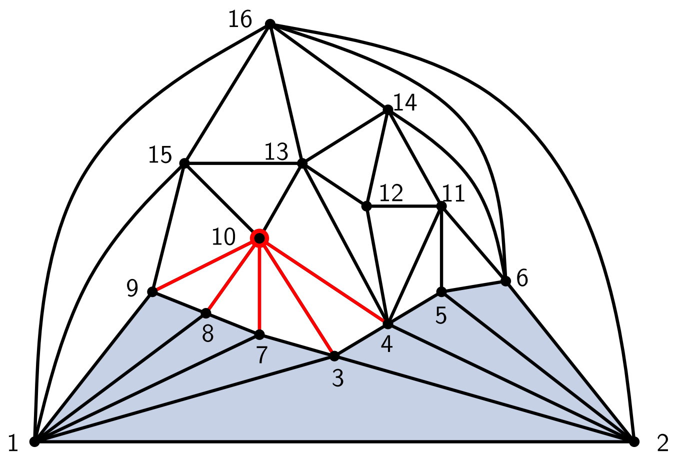


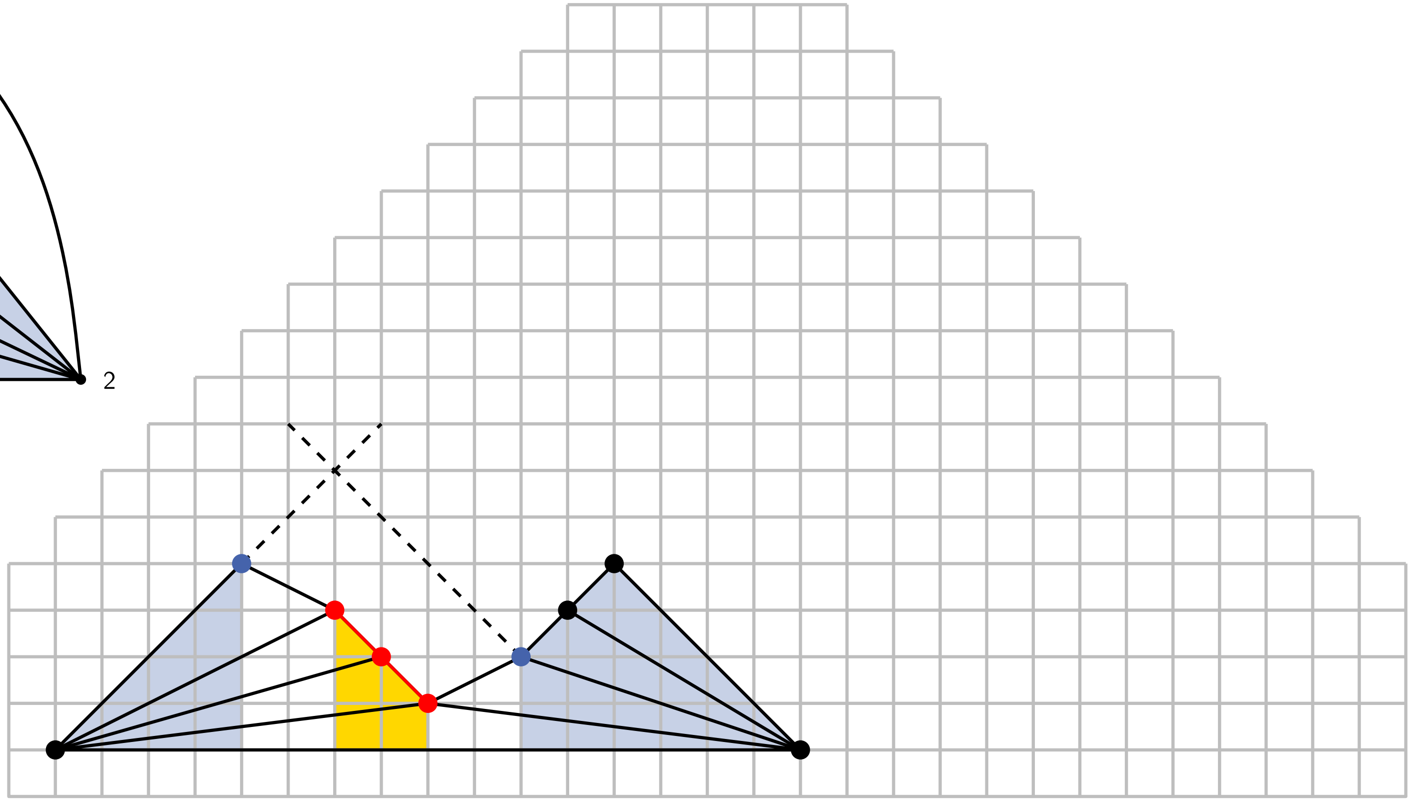
16 - 11

De Fraysseix Pach Pollack (Shift) Algorithm



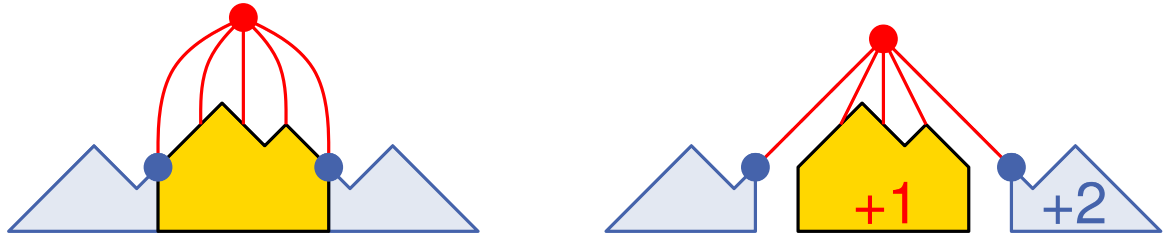


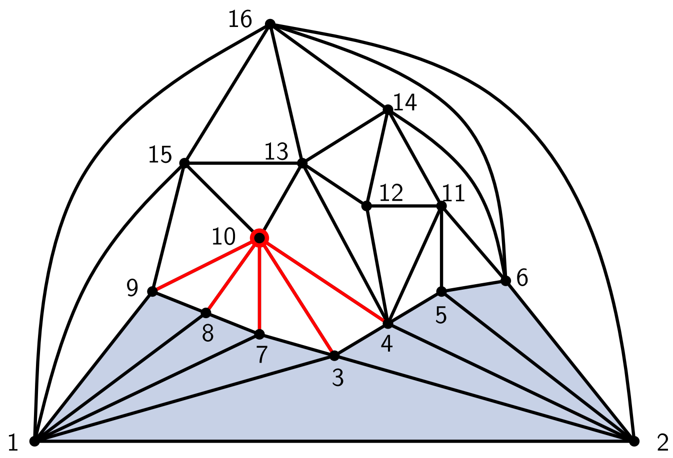


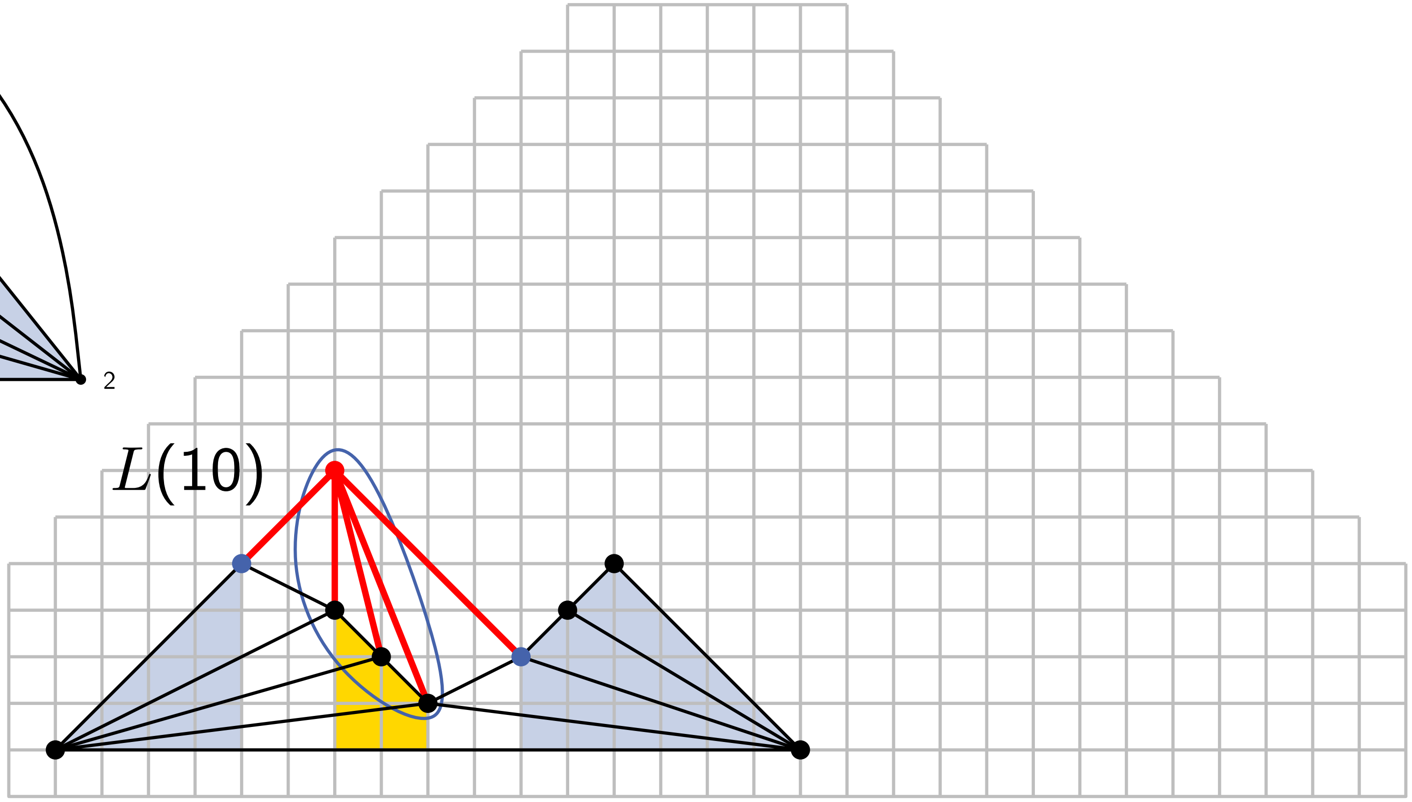


16 - 12

De Fraysseix Pach Pollack (Shift) Algorithm

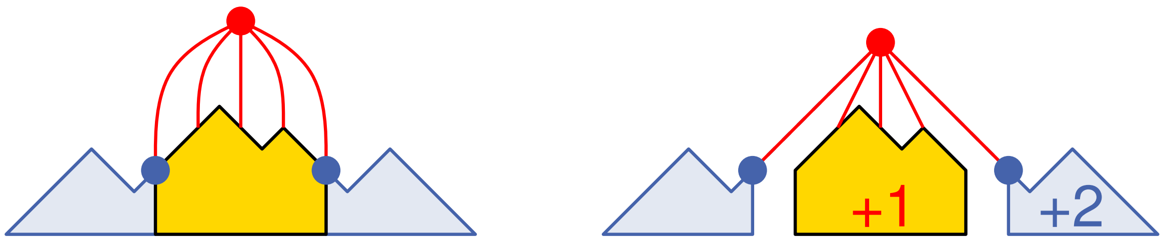


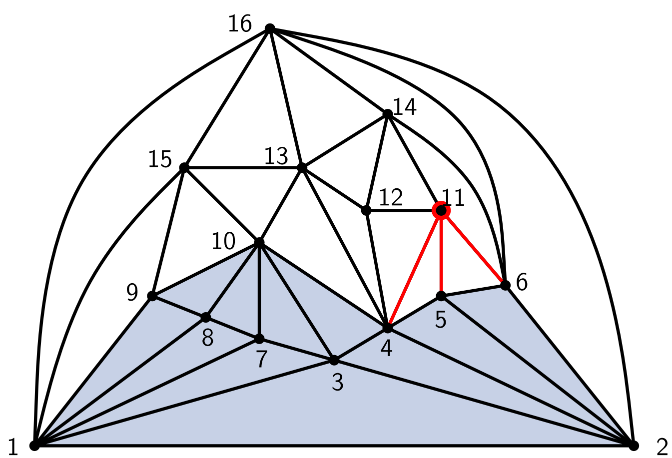


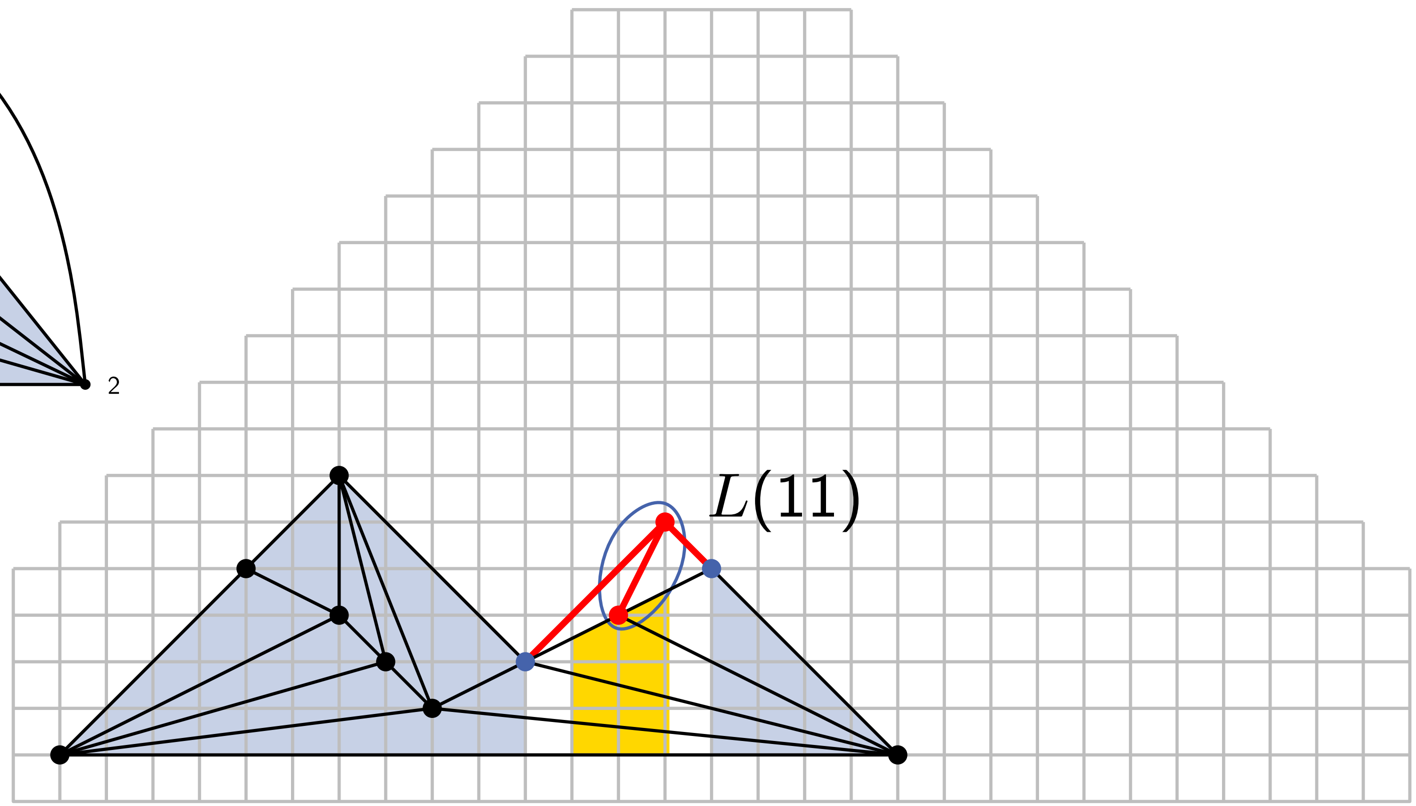


16 - 13

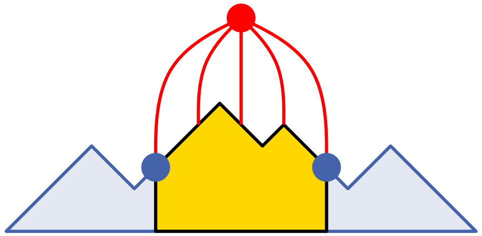
De Fraysseix Pach Pollack (Shift) Algorithm

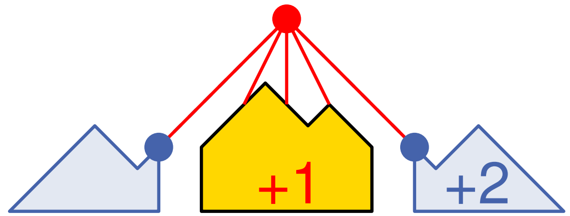


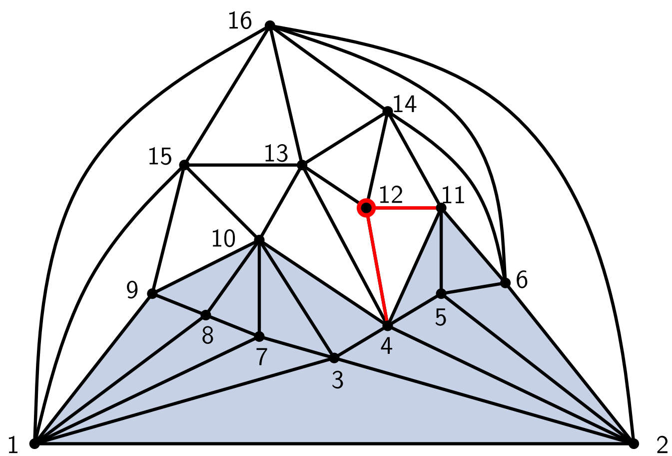


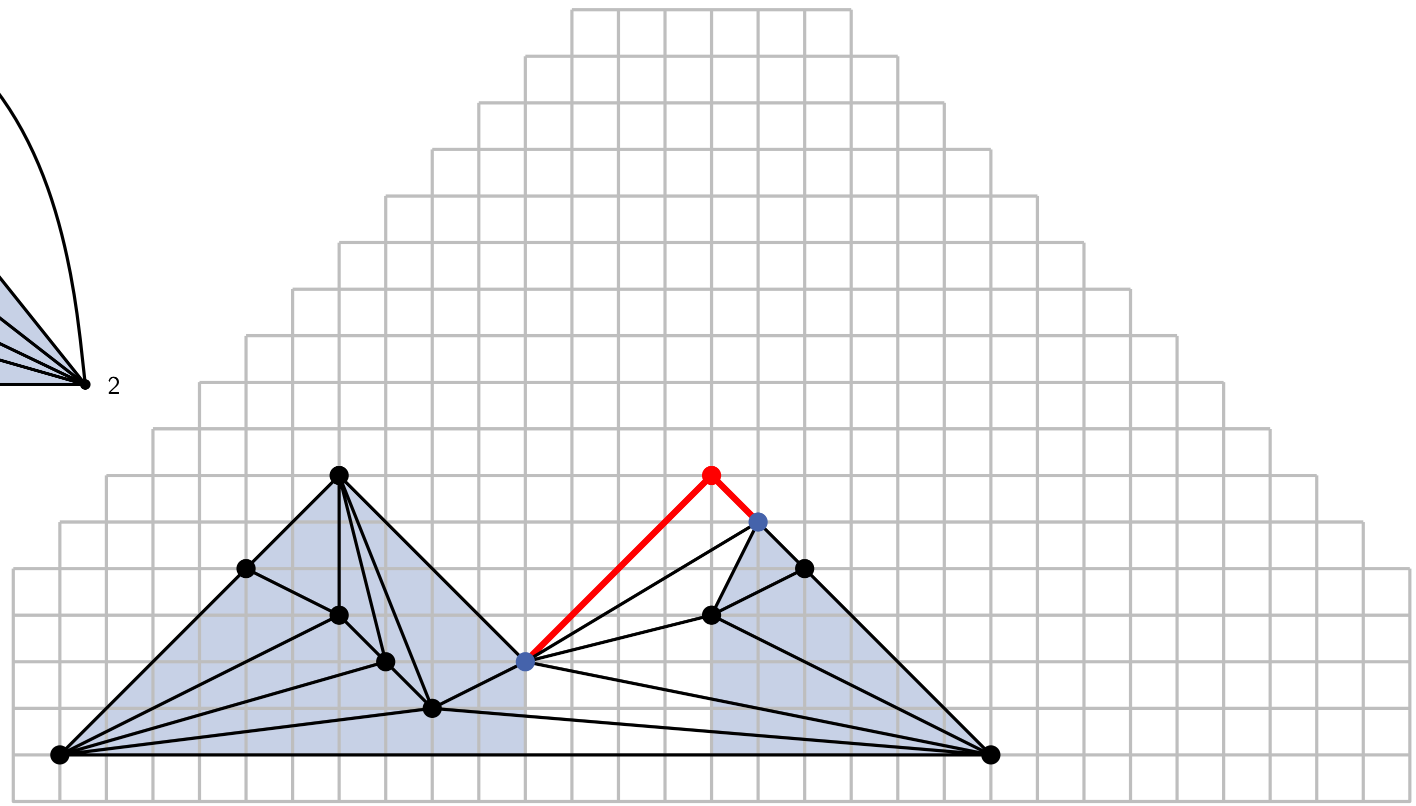


De Fraysseix Pach Pollack (Shift) Algorithm



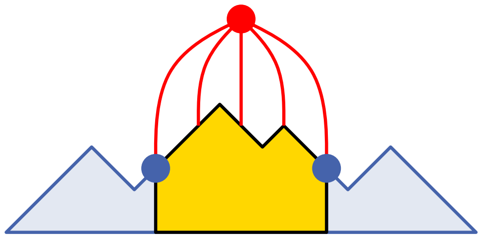


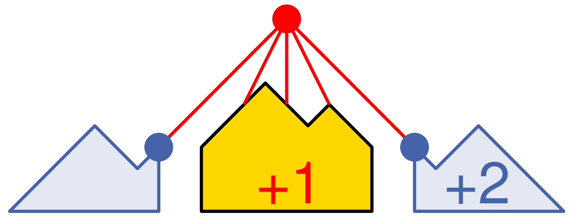


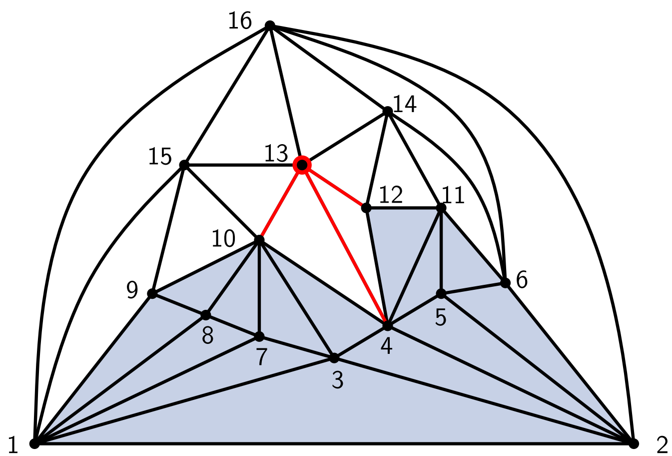


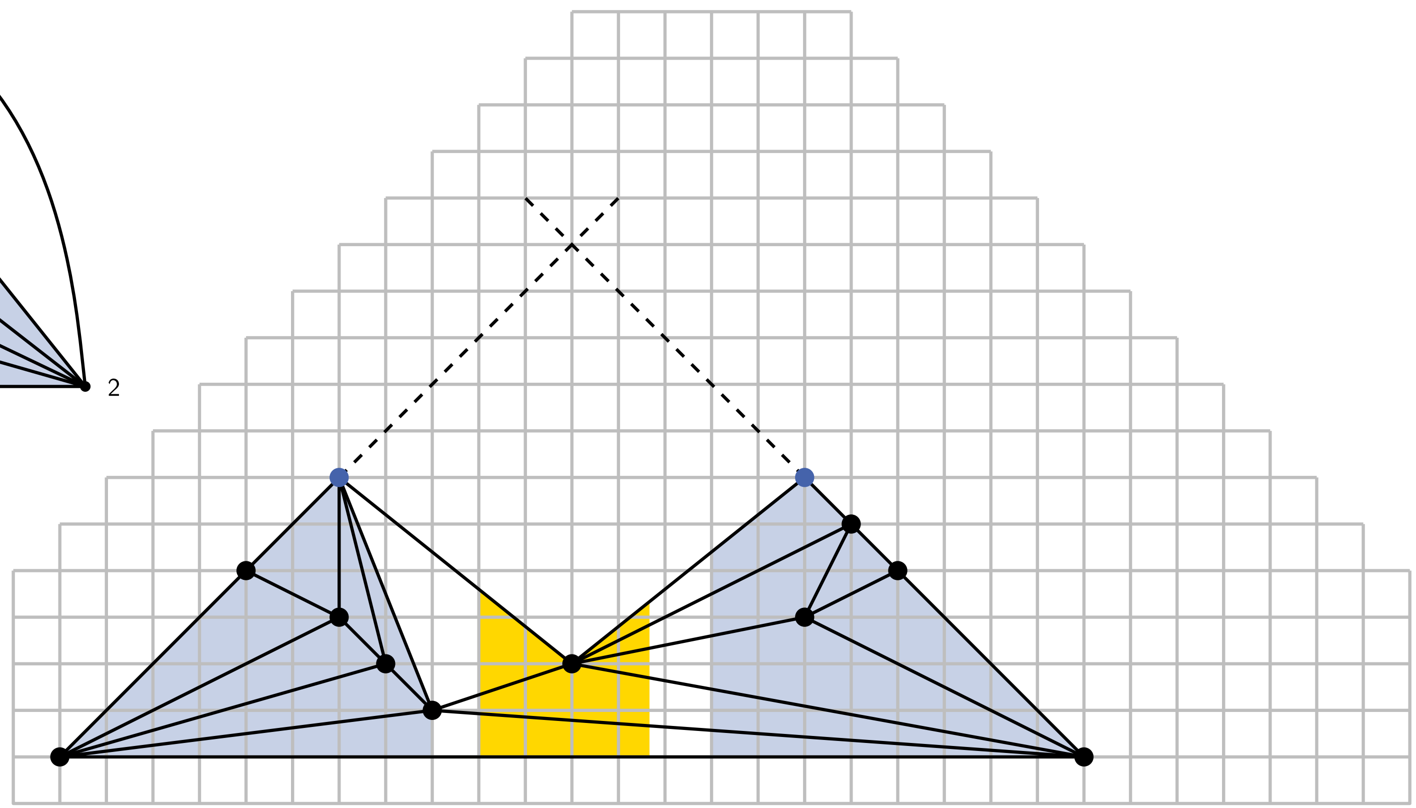
16 - 15

De Fraysseix Pach Pollack (Shift) Algorithm



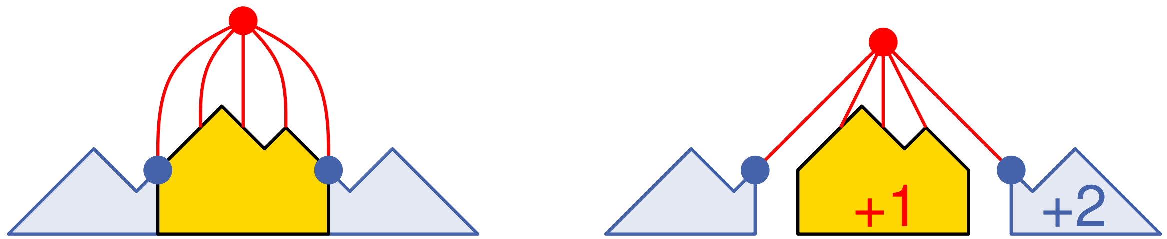


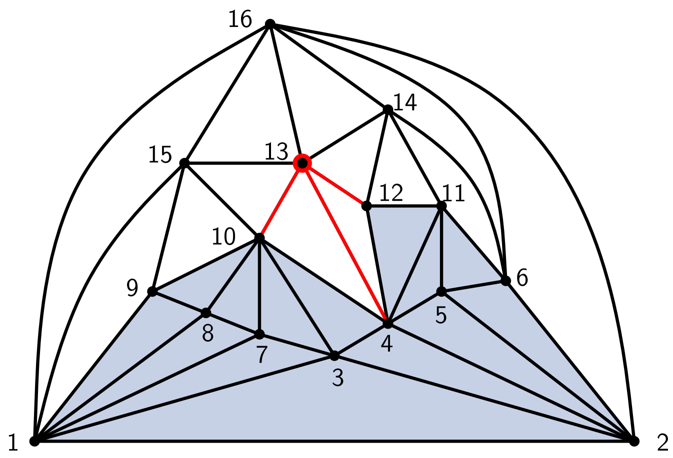


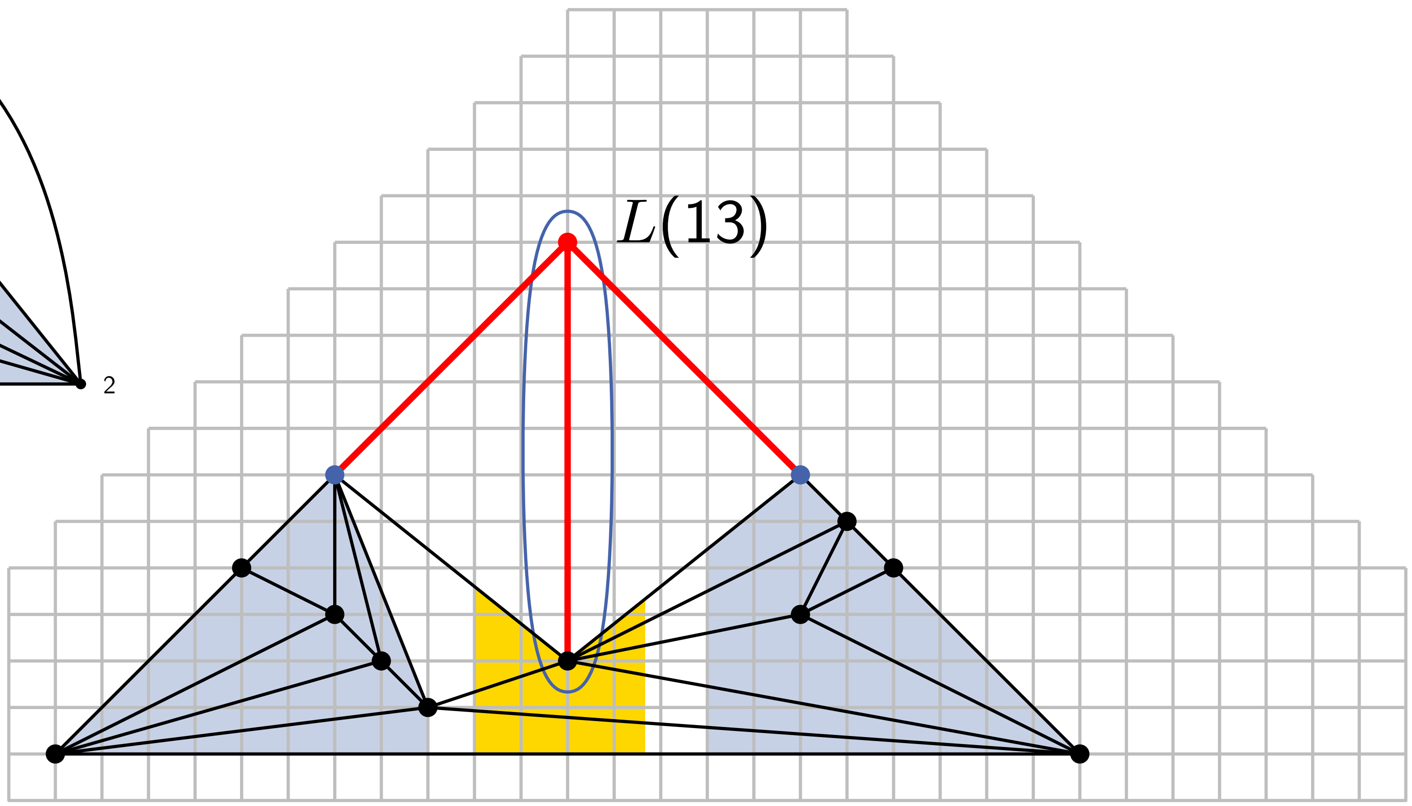


16 - 16

De Fraysseix Pach Pollack (Shift) Algorithm

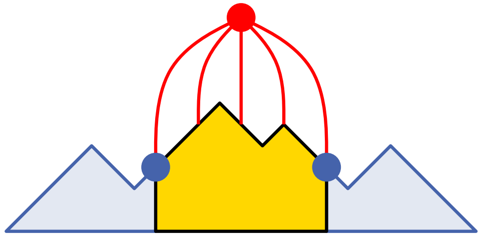


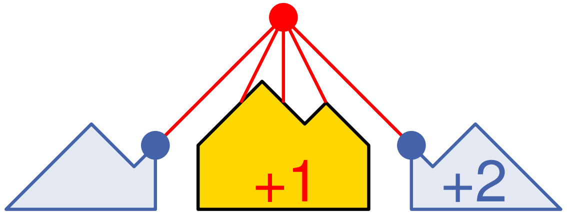


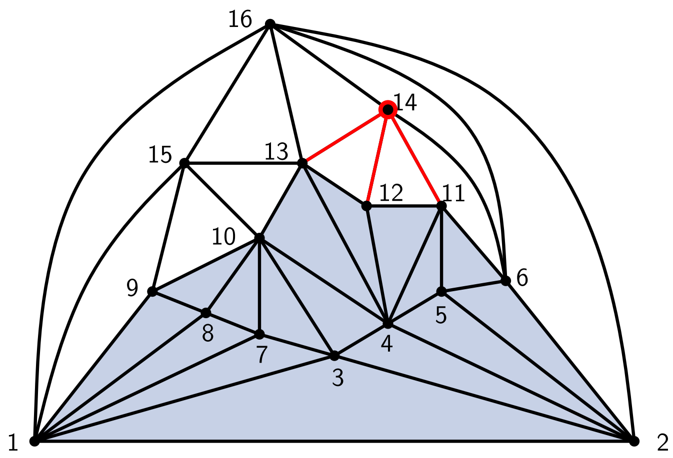


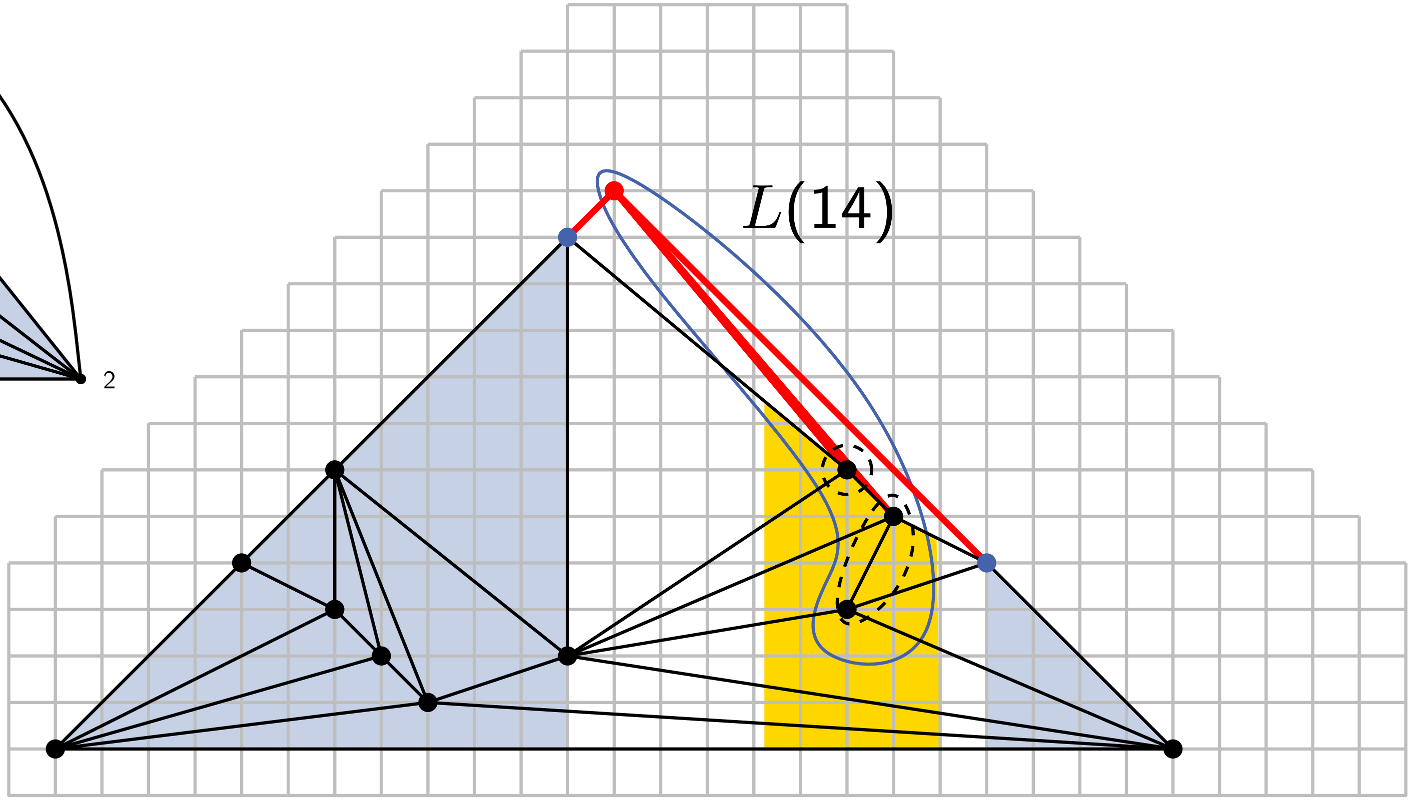
16 - 17

De Fraysseix Pach Pollack (Shift) Algorithm



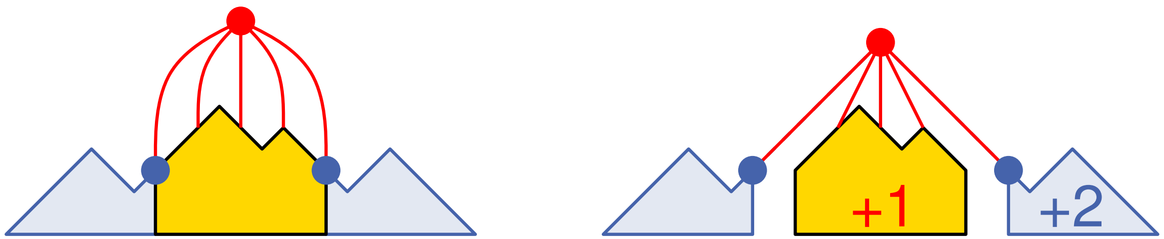


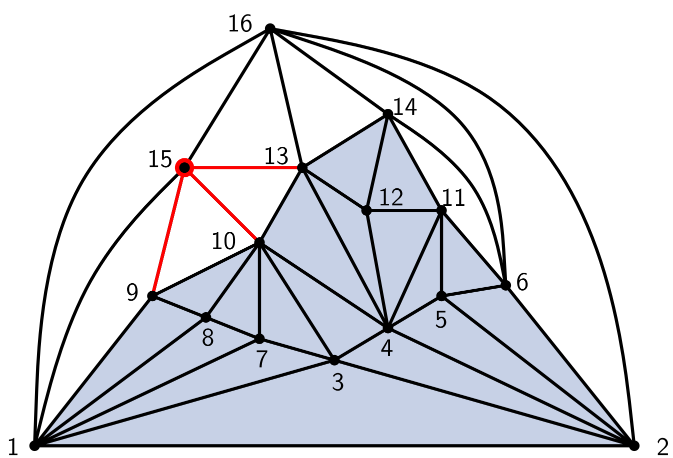


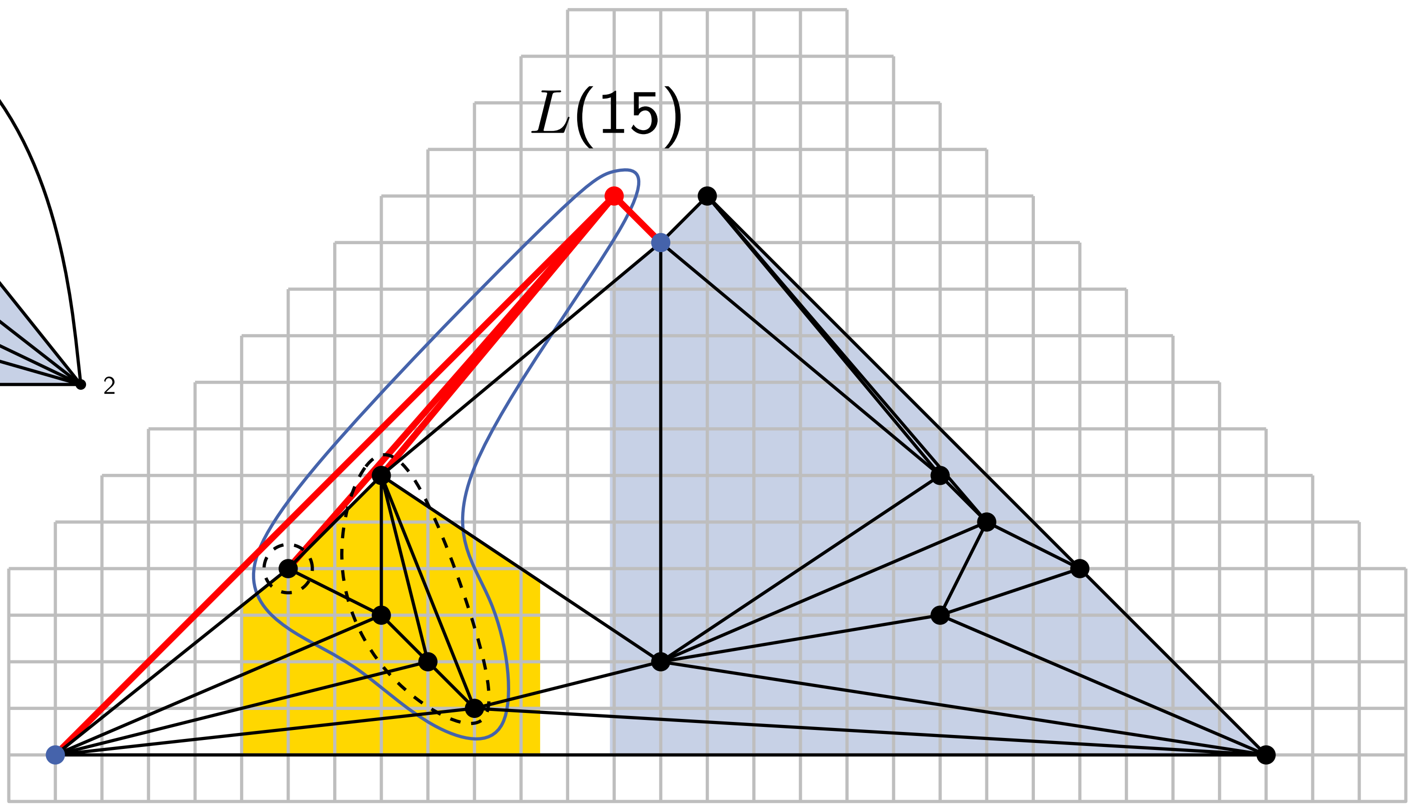


16 - 18

De Fraysseix Pach Pollack (Shift) Algorithm



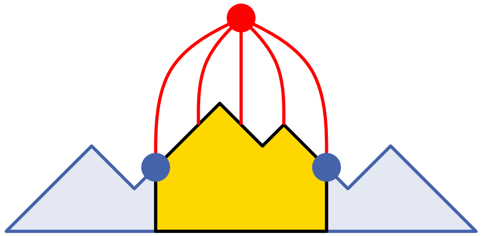


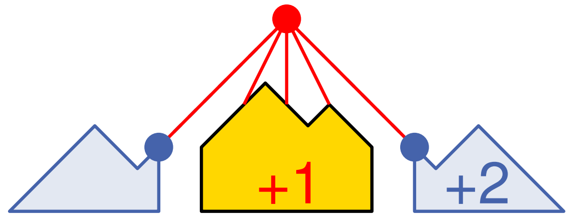


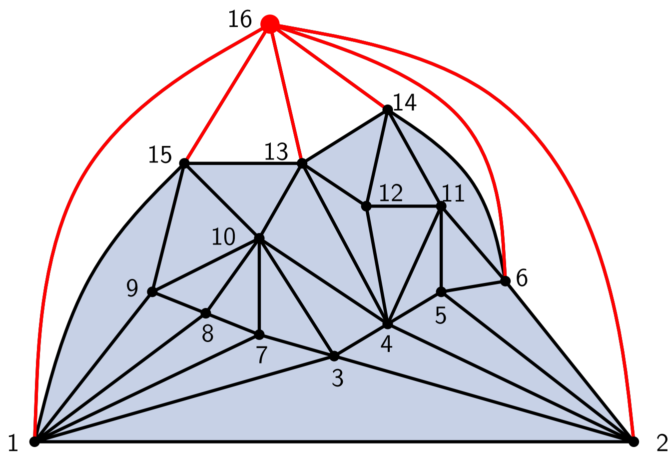
16 - 19

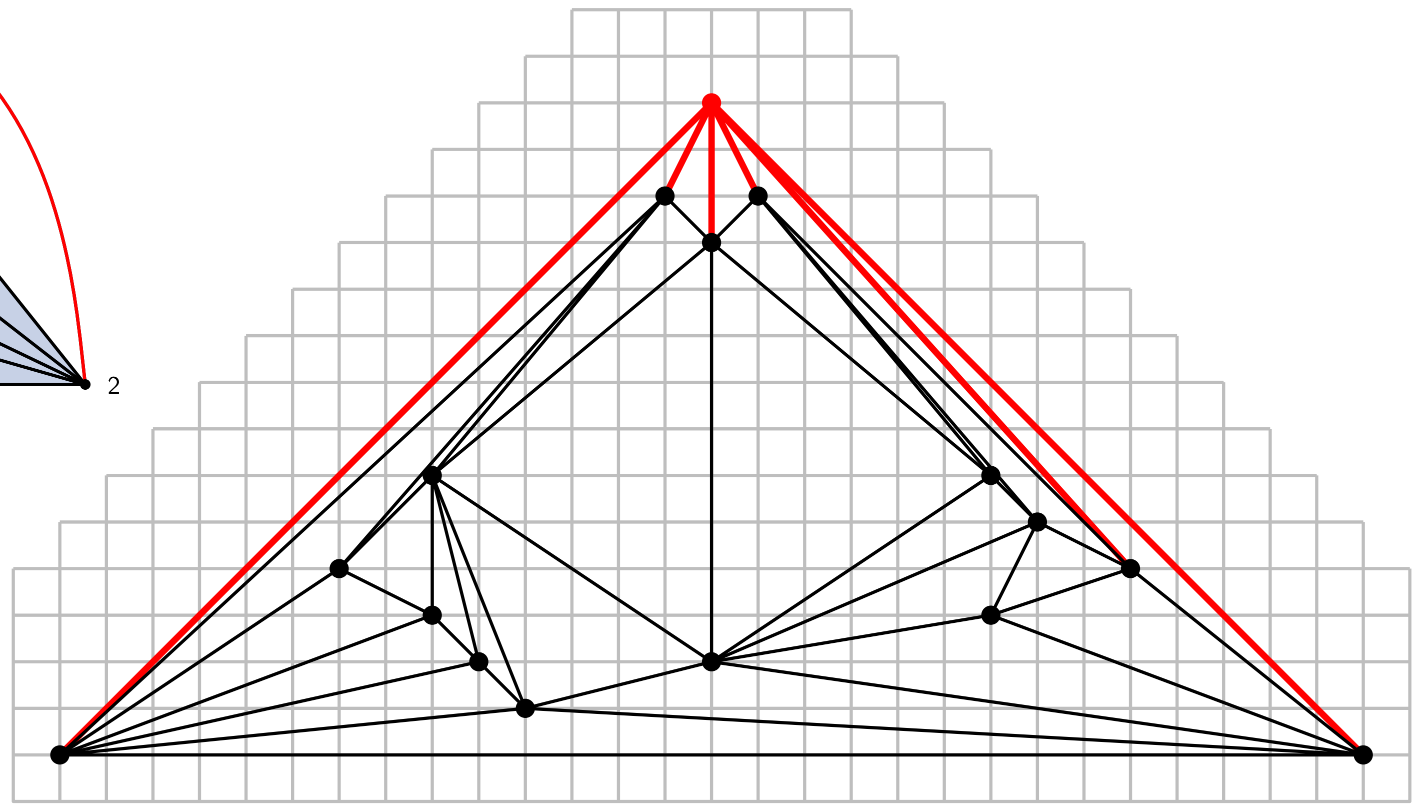
De Fraysseix Pach Pollack (Shift) Algorithm

KIT
Karlsruhe Institute of Technology





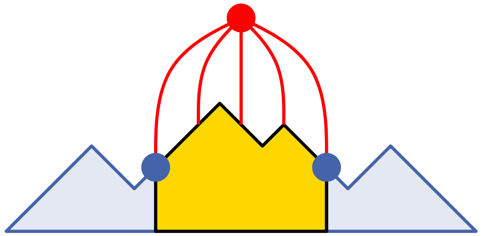


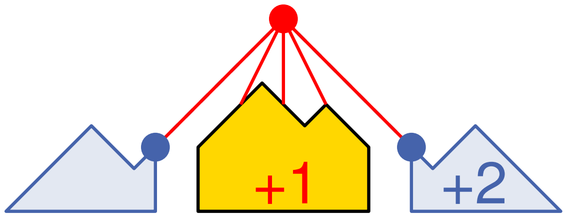


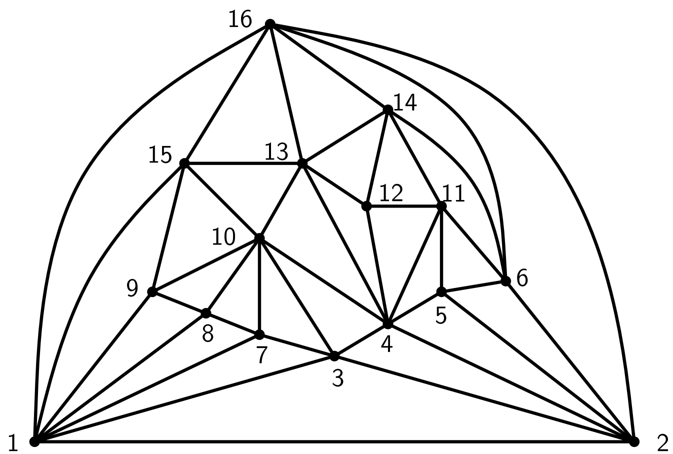
16 - 20

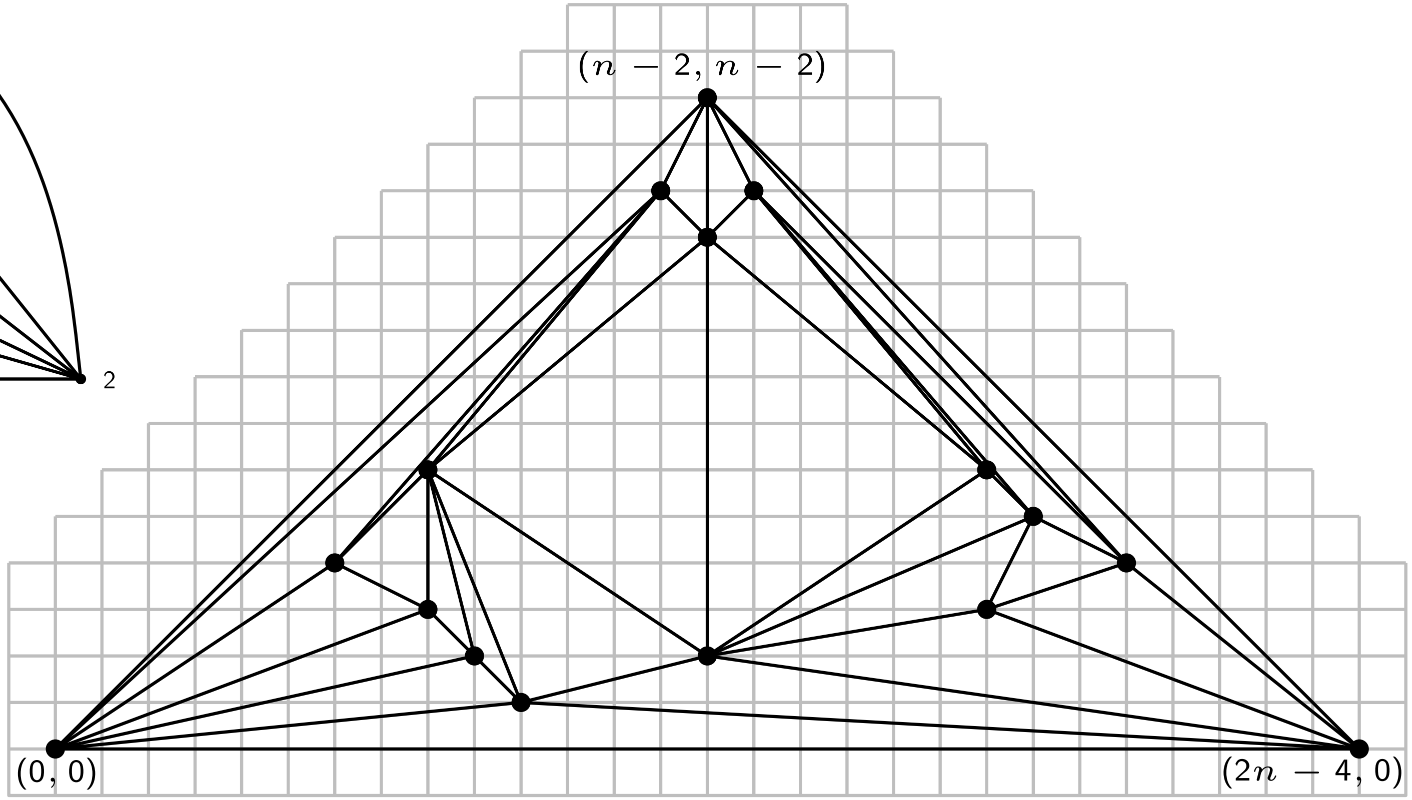
Institut für Theoretische Informatik
Lehrstuhl Algorithmik I

De Fraysseix Pach Pollack (Shift) Algorithm



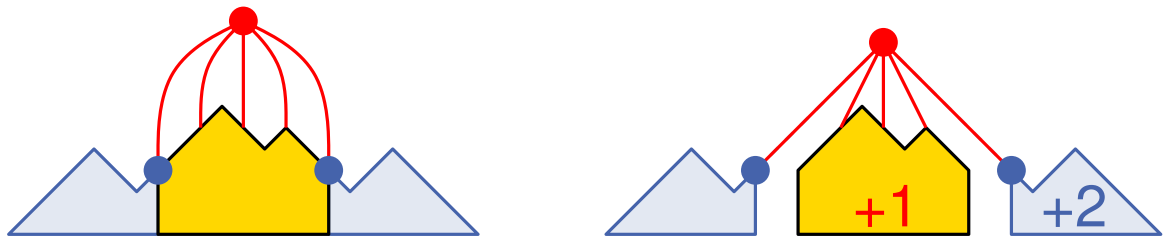






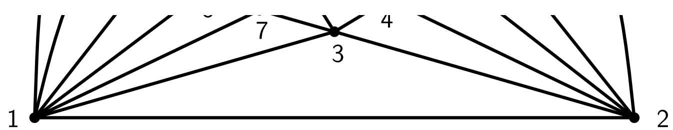
16 - 21

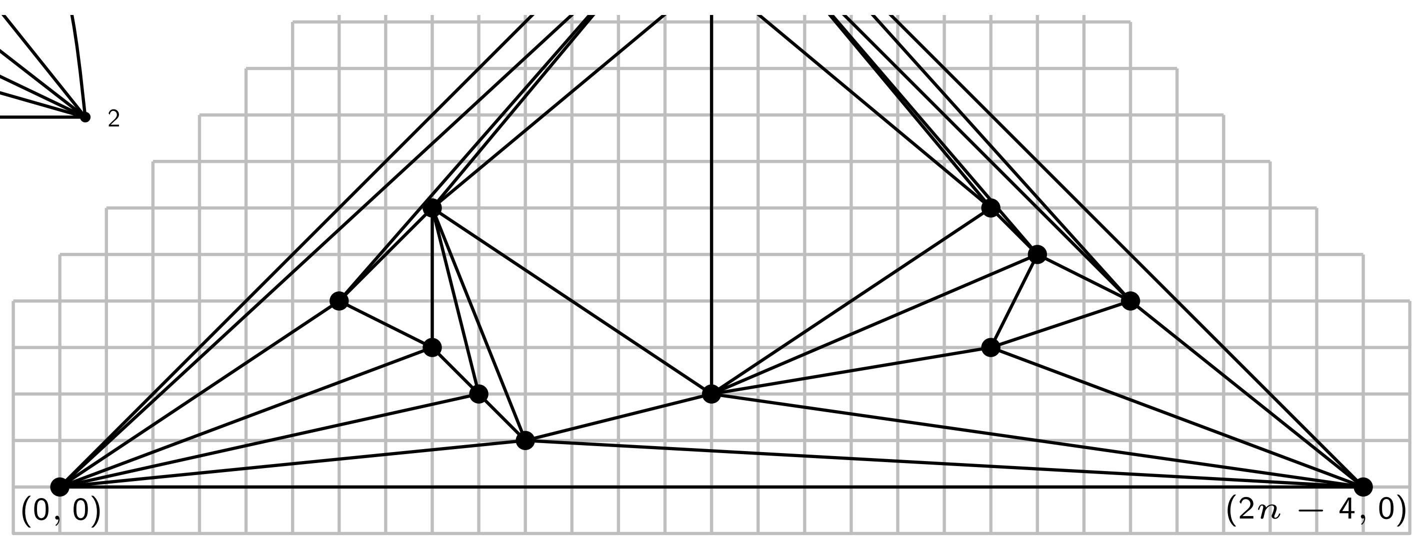
De Fraysseix Pach Pollack (Shift) Algorithm



Take a minute to think about the algorithm. Any questions?

1 min



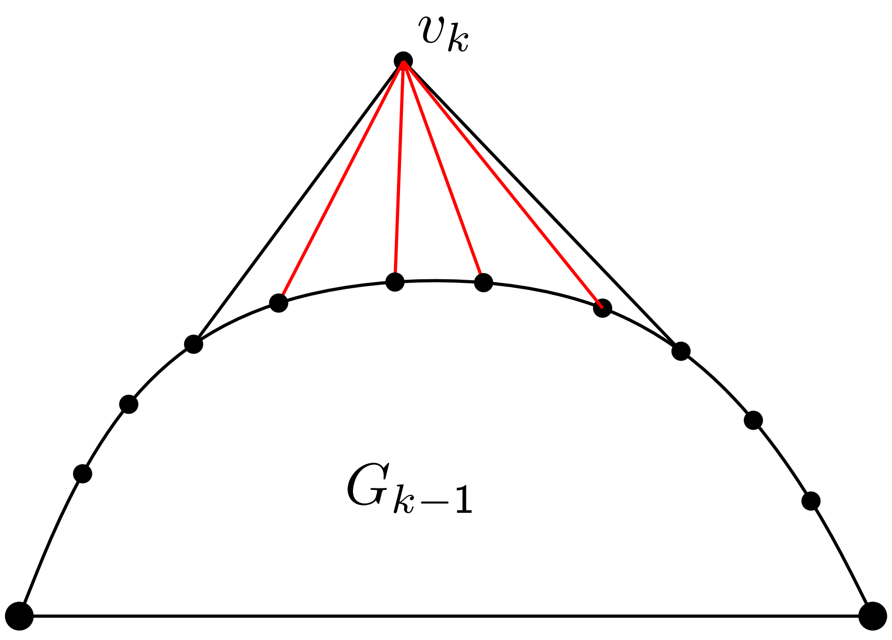


Institut für Theoretische Informatik
Lehrstuhl Algorithmik I

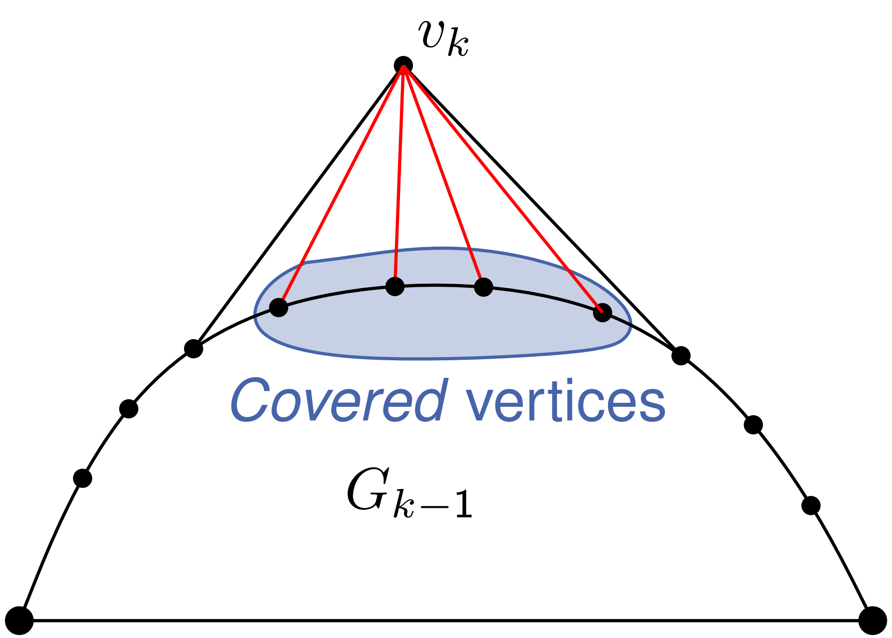
Outline

- Canonical ordering. Existence.
- Canonical ordering. Computation.
- Shift algorithm.
- Proof of planarity.
- Implentational details.

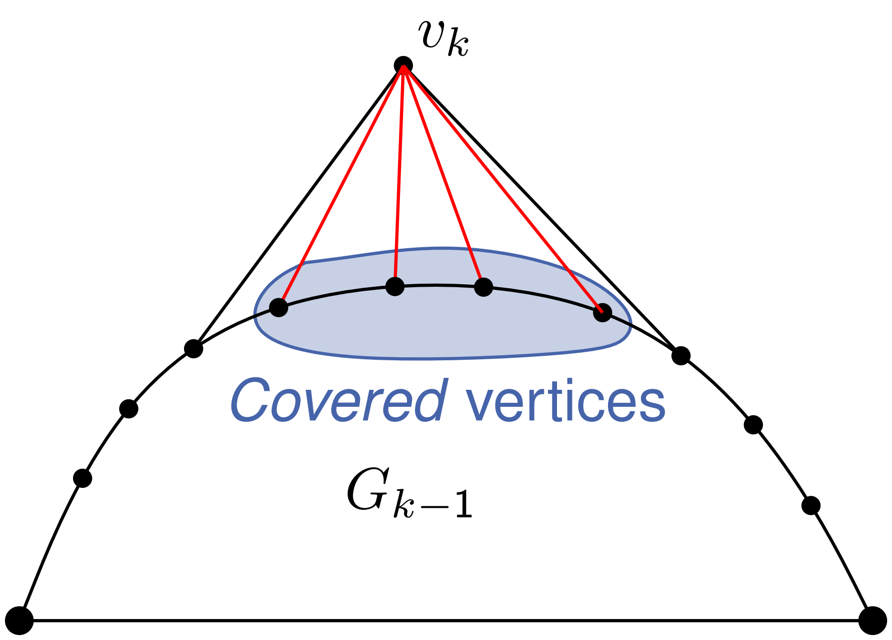
Proof of Planarity



Proof of Planarity

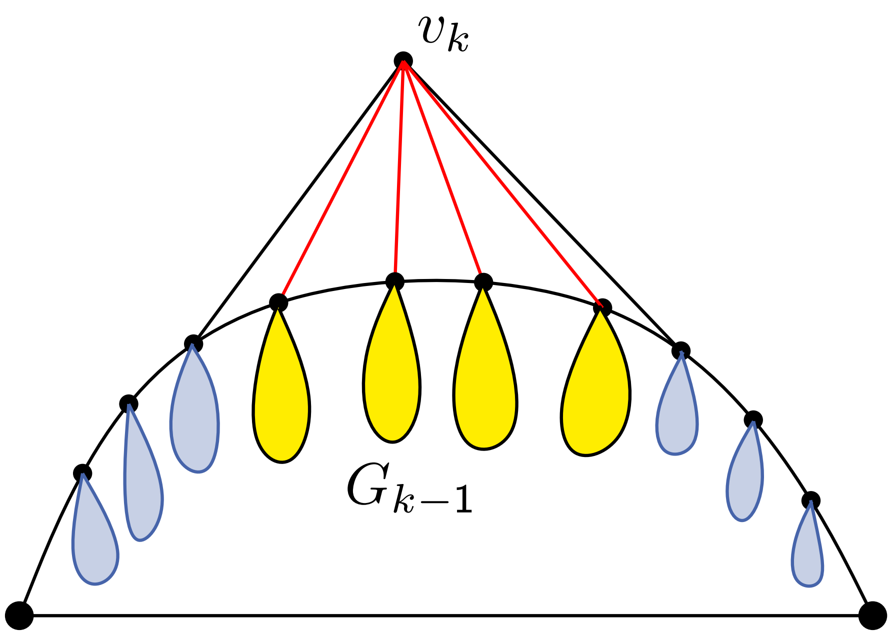


Proof of Planarity



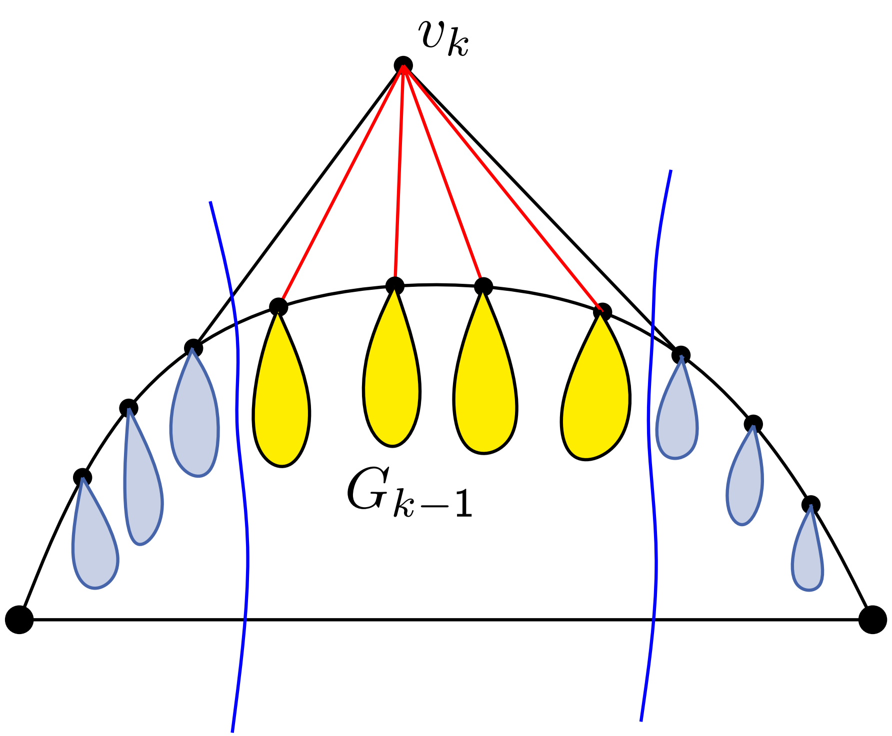
- Each internal vertex is covered exactly once
- Coverence relation defines a tree in G
- But a forest in G_i, $1 \leq i \leq n-1$

Proof of Planarity



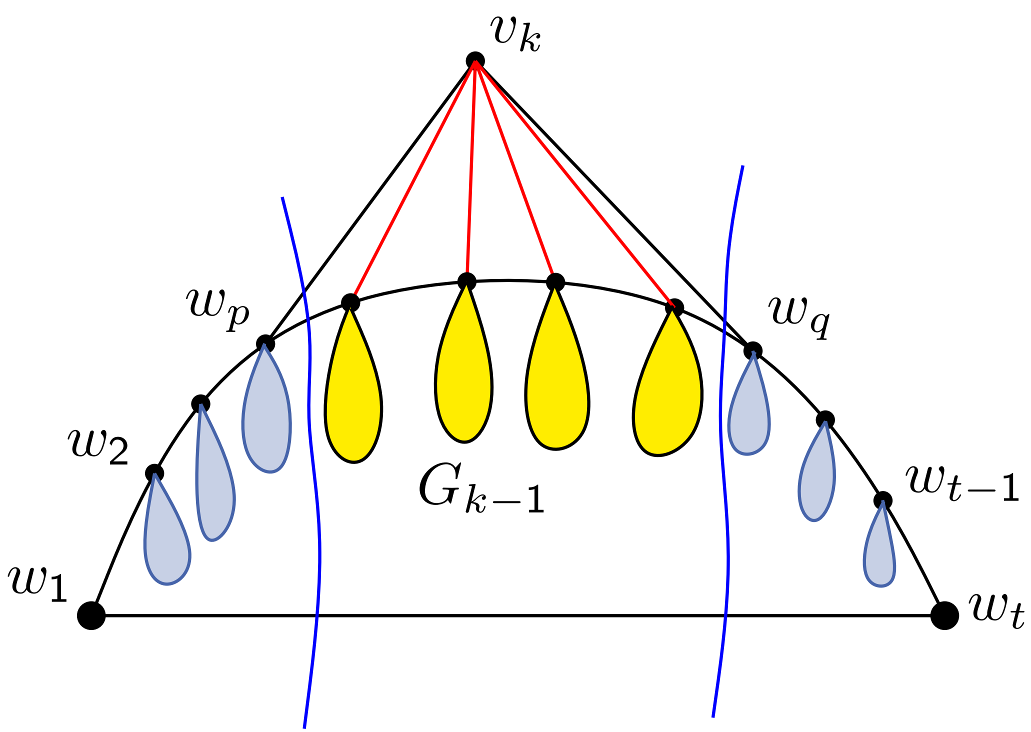
- Each internal vertex is covered exactly once
- Coverence relation defines a tree in G
- But a forest in G_i, $1 \leq i \leq n-1$

Proof of Planarity



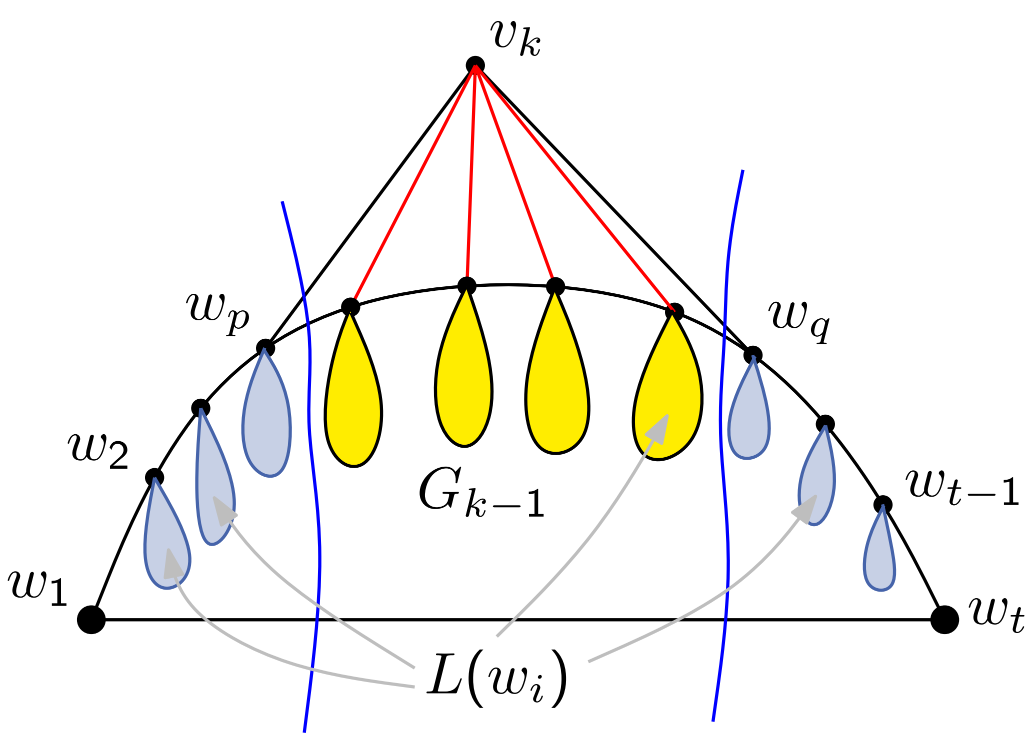
- Each internal vertex is covered exactly once
- Coverence relation defines a tree in G
- But a forest in G_i, $1 \leq i \leq n-1$

Proof of Planarity



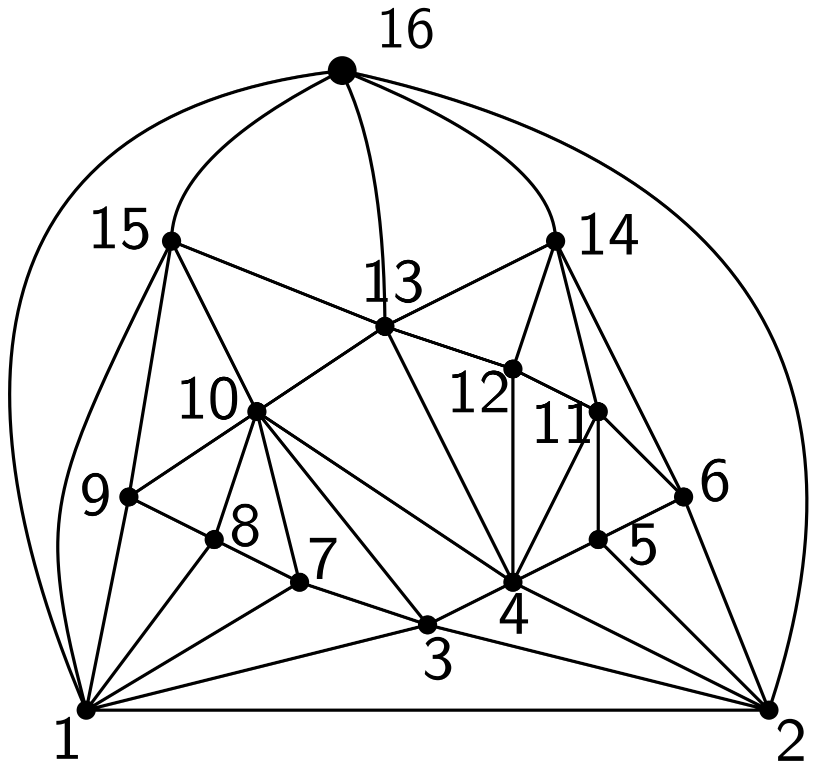
- Each internal vertex is covered exactly once
- Coverence relation defines a tree in G
- But a forest in G_i, $1 \leq i \leq n-1$

Proof of Planarity



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

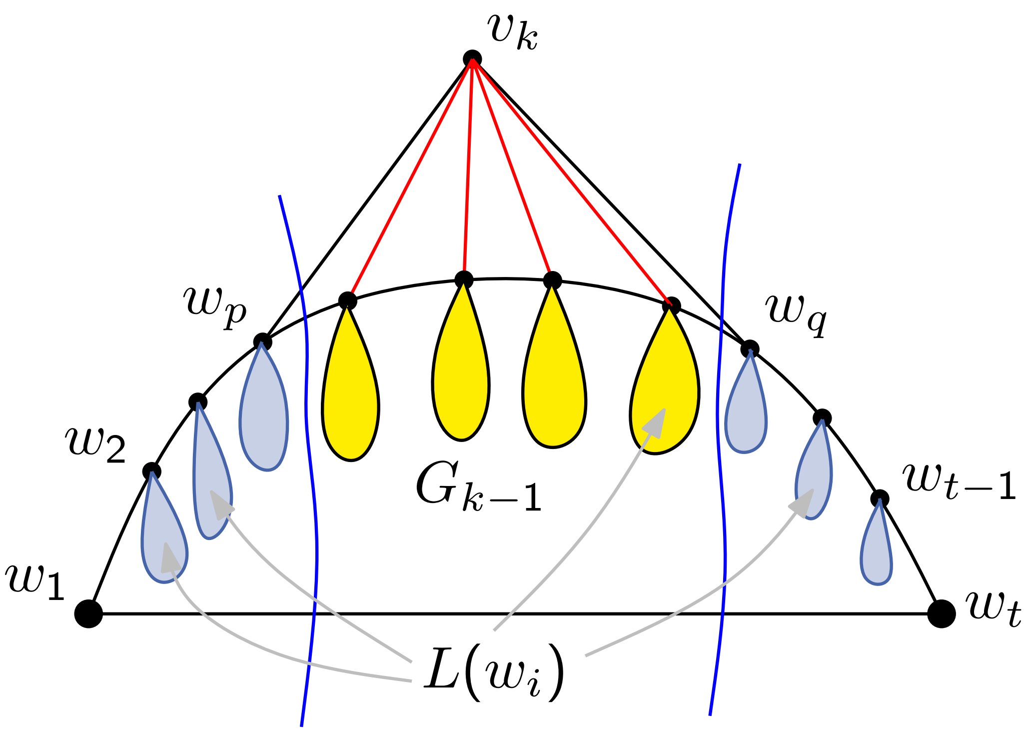
Proof of Planarity



Work with your neighbour(s) and then share **5 min**

- Compute the cover trees of vertices 15, 13, and 14.

Proof of Planarity



- Each internal vertex is covered exactly once
- Coverence relation defines a tree in G
- But 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 is even. If we shift $L(w_i)$ by δ_i to the right, we get a planar straight line grid drawing.

Proof of Planarity

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 is even. If we shift $L(w_i)$ by δ_i to the right, we get a planar straight line grid drawing.

Proof of Planarity

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 is even. If we shift $L(w_i)$ by δ_i to the right, we get a planar straight line grid drawing.

Proof

- The proof is by induction on i, i.e. we consider $G_3, \ldots, G_n$.

Proof of Planarity

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 is even. If we shift $L(w_i)$ by δ_i to the right, we get a planar straight line grid drawing.

Proof

- The proof is by induction on i, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true for G_{k-1}.

Proof of Planarity

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 is even. If we shift $L(w_i)$ by δ_i to the right, we get a planar straight line grid drawing.

Proof

- The proof is by induction on i, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true 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 $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, such that $\delta_q - \delta_p \geq 2$ and is even. If we shift $L(w_i)$ by δ_i to the right, we get a planar straight line grid drawing.

Proof

- The proof is by induction on i, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true 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 $0 < \delta_1 \leq \delta_2 \leq \cdots \leq \delta_t \in \mathbb{N}$, such that $\delta_q - \delta_p \geq 2$ and is even. If we shift $L(w_i)$ by δ_i to the right, we get a planar straight line grid drawing.

Proof

- The proof is by induction on i, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true for G_{k-1}.
- Let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of G_k.
- Let $\delta_1 \leq \cdots \leq \delta_p \leq \delta \leq \delta_q \leq \cdots \leq \delta_t$.
- We set $\delta'_i = \delta_i$ for $1 \leq i \leq p$,
- $\delta'_i = \delta$ for $p+1 \leq i \leq q-1$ (for the neighbors of v_k)
- $\delta'_i = \delta_i$ for $q \leq i \leq t$.

Proof of Planarity

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 is even. If we shift $L(w_i)$ by δ_i to the right, we get a planar straight line grid drawing.

Proof

- The proof is by induction on i, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true for G_{k-1}.
- Let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of G_k.
- Let $\delta_1 \leq \cdots \leq \delta_p \leq \delta \leq \delta_q \leq \cdots \leq \delta_t$.
- We set $\delta'_i = \delta_i$ for $1 \leq i \leq p$,
- $\delta'_i = \delta$ for $p + 1 \leq i \leq q - 1$ (for the neighbors of v_k)
- $\delta'_i = \delta_i$ for $q \leq i \leq t$.
- By induction hypothesis we can move $w_1 \ldots, w_t$ by $\delta'_1 \ldots \delta'_t$, respectively.

Proof of Planarity

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 is even. If we shift $L(w_i)$ by δ_i to the right, we get a planar straight line grid drawing.

Proof

- The proof is by induction on i, i.e. we consider $G_3, \ldots, G_n$.
- Assume that this is true for G_{k-1}.
- Let $w_1, \ldots, w_p, v_k, w_q, \ldots, w_t$ be the boundary of G_k.
- Let $\delta_1 \leq \cdots \leq \delta_p \leq \delta \leq \delta_q \leq \cdots \leq \delta_t$.
- We set $\delta'_i = \delta_i$ for $1 \leq i \leq p$,
- $\delta'_i = \delta$ for $p+1 \leq i \leq q-1$ (for the neighbors of v_k)
- $\delta'_i = \delta_i$ for $q \leq i \leq t$.
- By induction hypothesis we can move $w_1 \ldots, w_t$ by $\delta'_1 \ldots \delta'_t$, respectively.
- We can complete the drawing by placing v_k, v_k is moved with $L(w_{p+1}), \ldots, L(w_{q-1})$ by δ.

Outline

- Canonical ordering. Existence.
- Canonical ordering. Computation.
- Shift algorithm.
- Proof of planarity.
- Implentational details.

Implementation Details

Algorithm Shift

Let $v_1, \ldots, v_n$ be a 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 $i = 4$ **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 neighbors 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 $+1$ and -1 edges from $P(w_p)$ and $P(w_q)$;
  $L(v_i) = \cup_{j=p+1}^{q-1} L(w_j) \cup \{v_i\}$;

Implementation Details

Karlsruhe Institute of Technology

Algorithm Shift

Let $v_1, \ldots, v_n$ be a 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 $i = 4$ **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 neighbors 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 $+1$ and -1 edges from $P(w_p)$ and $P(w_q)$;

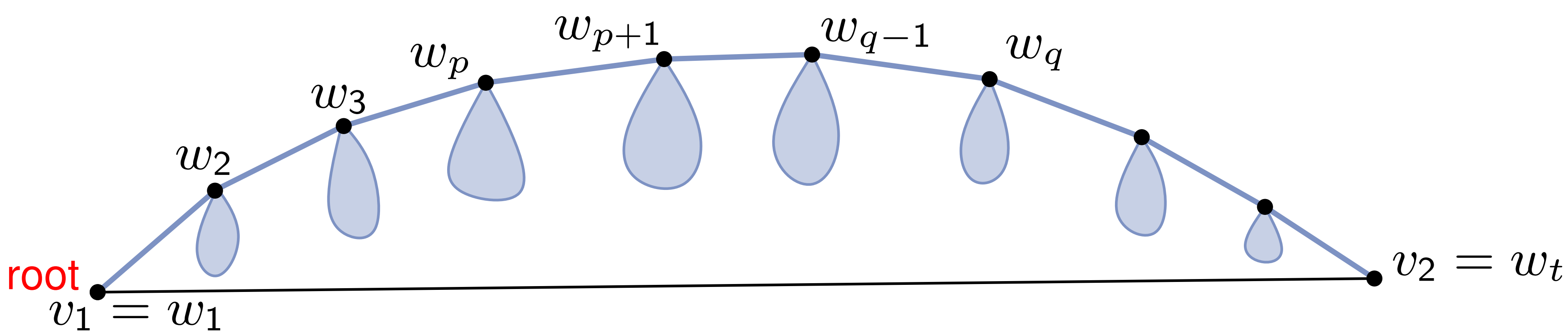
  $L(v_i) = \cup_{j=p+1}^{q-1} L(w_j) \cup \{v_i\}$;

Take a minute to think about the time complexity of the algorithm.
Can we do better? **2 min**

Implementation Details

relative x-distance tree

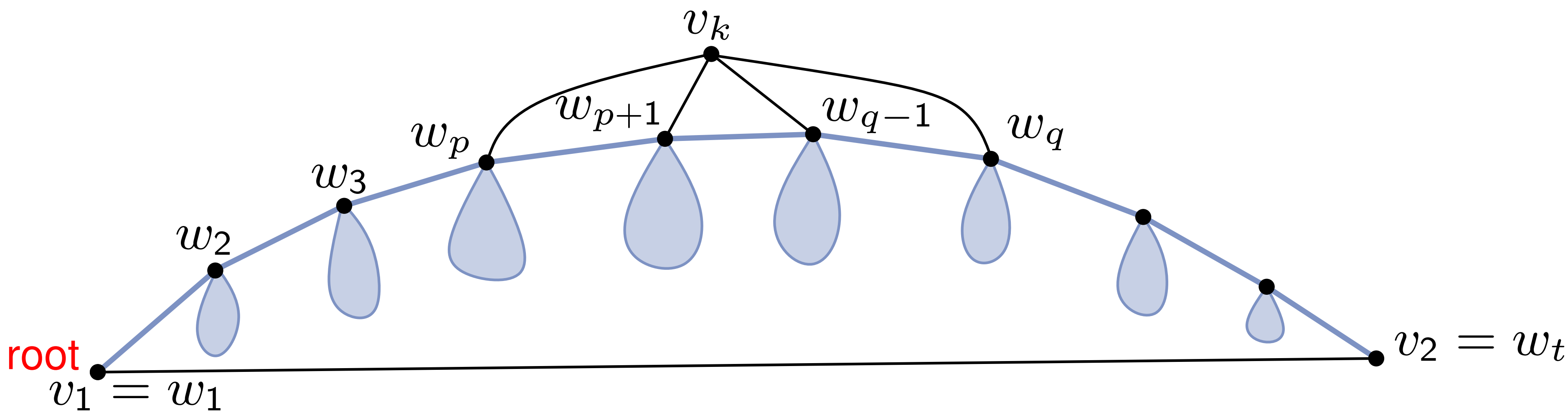
For each vertex store the x-offset from its parent and the y-coordinate



Implementation Details

relative x-distance tree

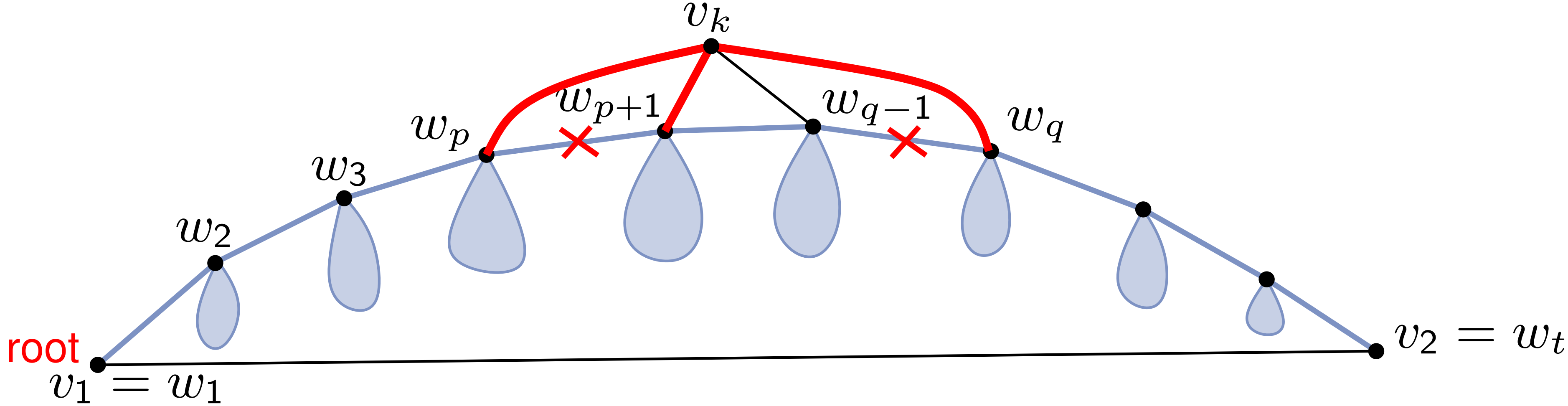
For each vertex store the x-offset from its parent and the y-coordinate



Implementation Details

relative x-distance tree

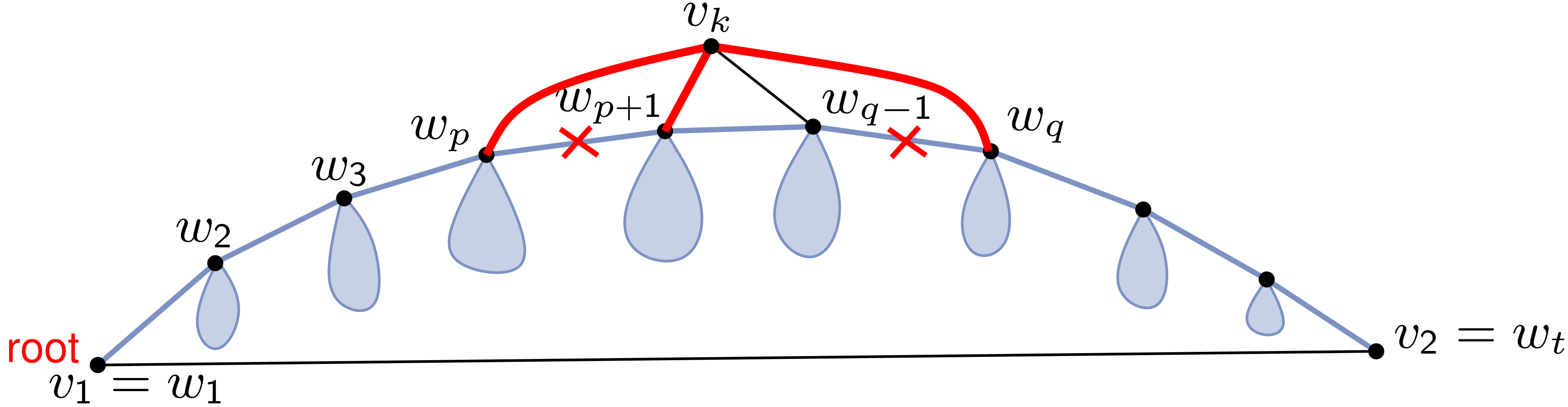
For each vertex store the x-offset from its parent and the y-coordinate



Implementation Details

relative x-distance tree

For each vertex store the x-offset from its parent and the y-coordinate



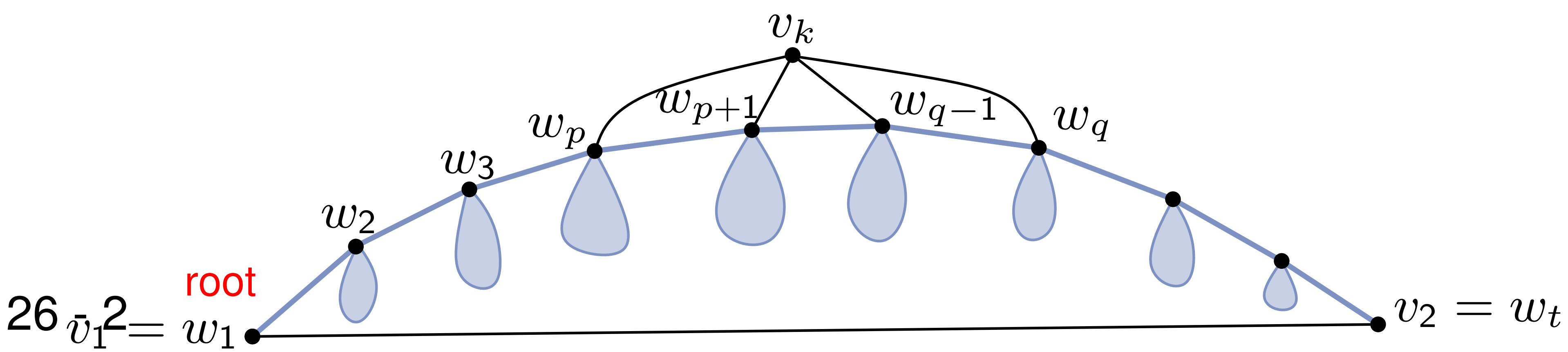
- $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$ (1)
- $y(v_k) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) + y(w_p))$ (2)
- $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$ (3)

Implementation Details

- In the binary tree 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 $x(v_k) - x(w_p)$

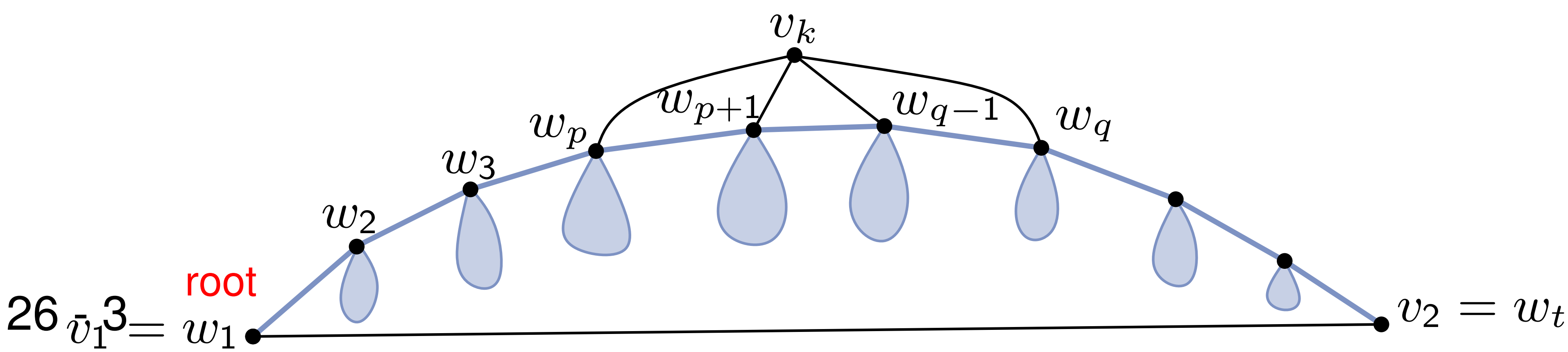
Implementation Details

- In the binary tree 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 $x(v_k) - x(w_p)$



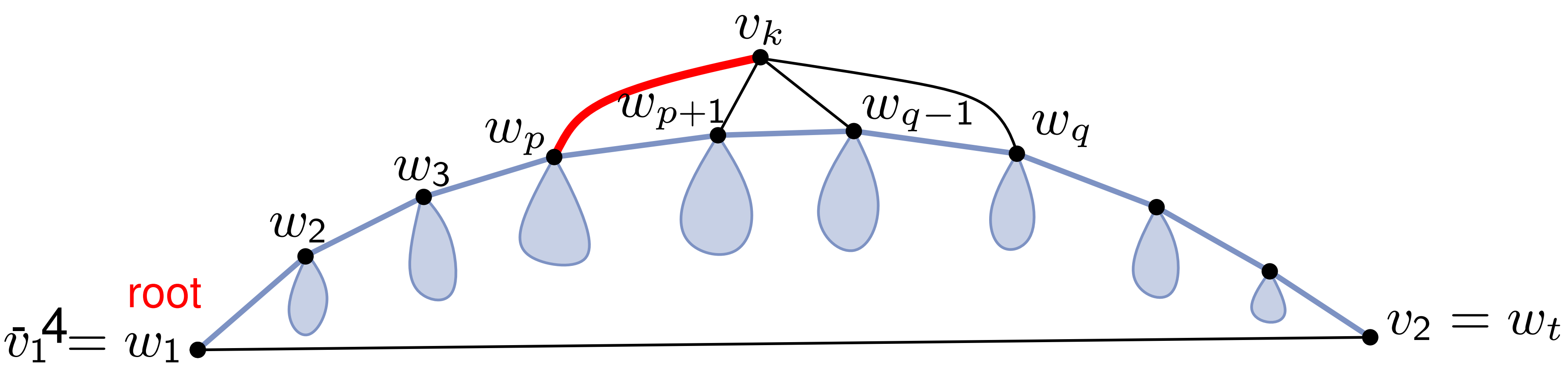
Implementation Details

- In the binary tree 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 $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, here $\Delta_x(w_q)$ is x-distance from the parent, $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



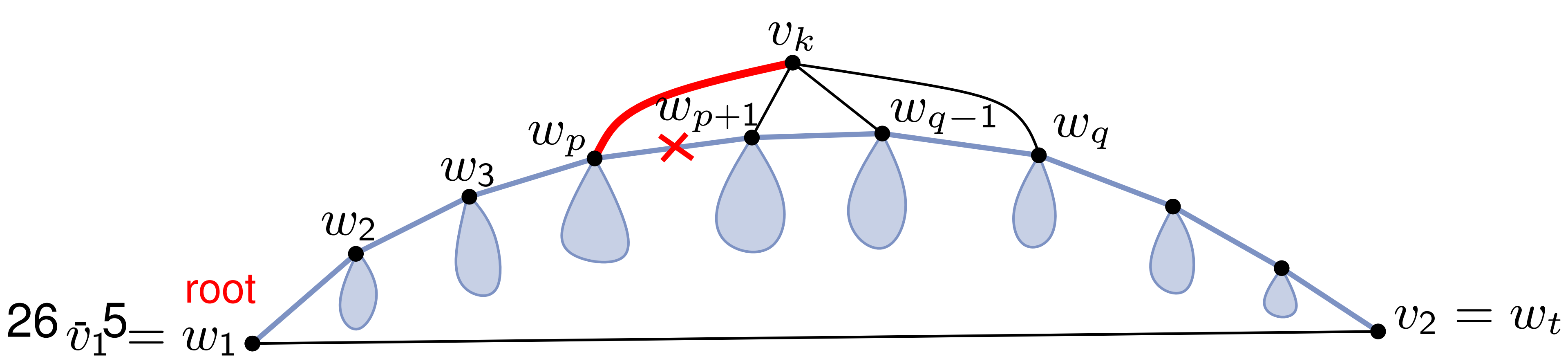
Implementation Details

- In the binary tree 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 $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, here $\Delta_x(w_q)$ is x-distance from the parent, $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



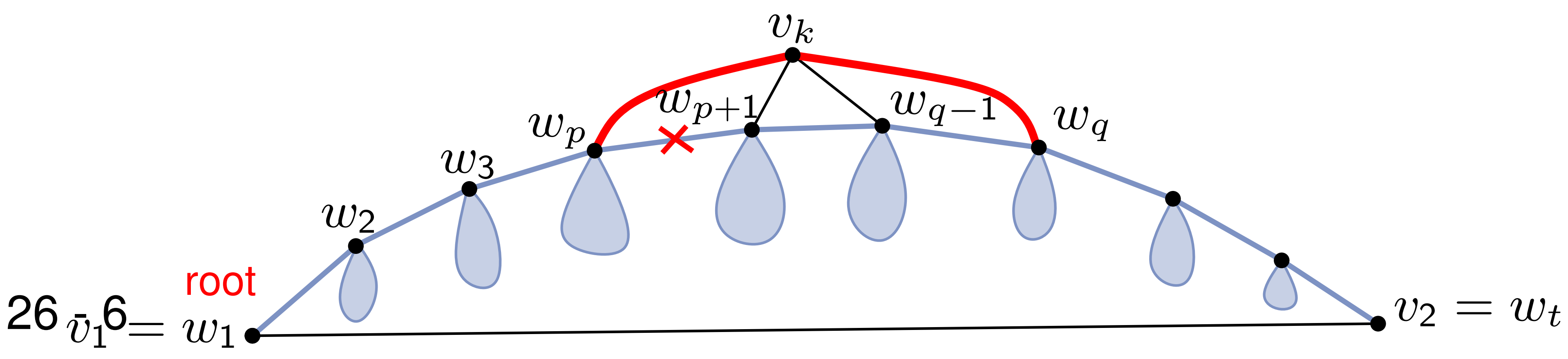
Implementation Details

- In the binary tree 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 $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, here $\Delta_x(w_q)$ is x-distance from the parent, $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



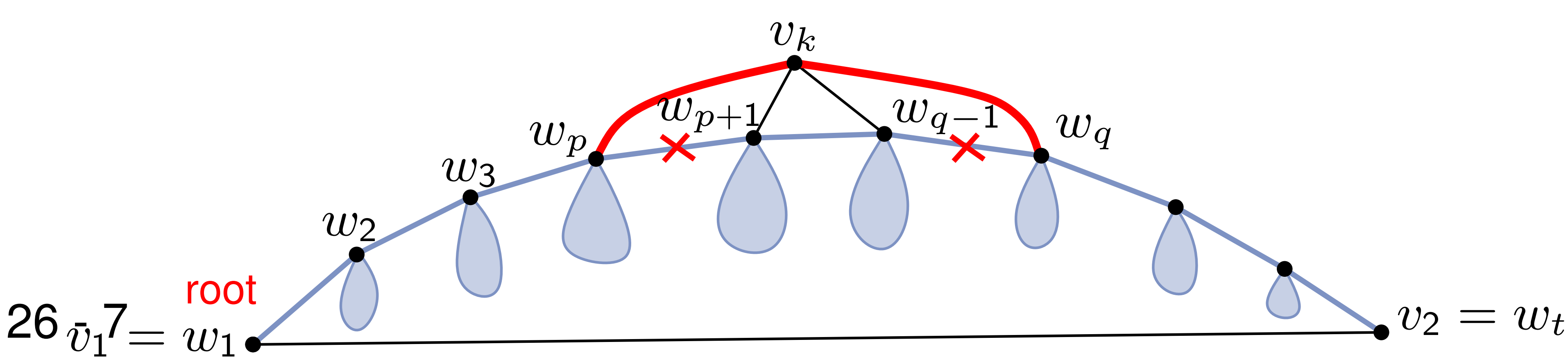
Implementation Details

- In the binary tree 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 $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, here $\Delta_x(w_q)$ is x-distance from the parent, $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



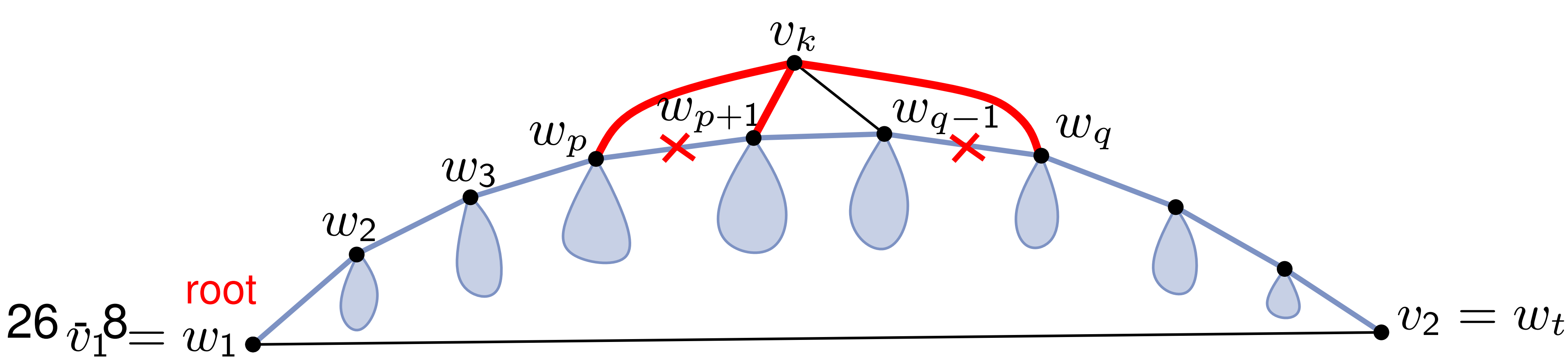
Implementation Details

- In the binary tree 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 $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, here $\Delta_x(w_q)$ is x-distance from the parent, $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)



Implementation Details

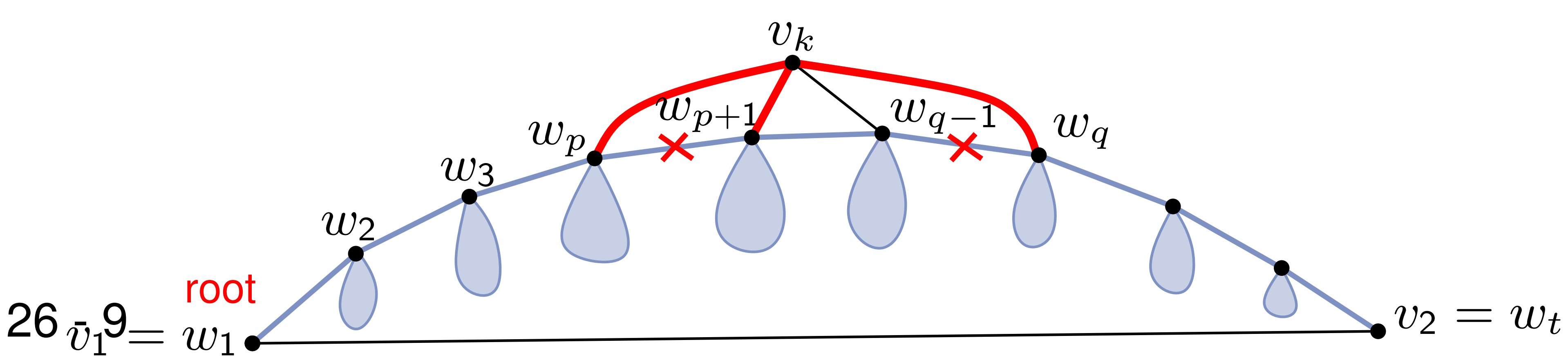
- In the binary tree at each vertex we keep its relative x-distance from its parent and its y-coordinate
- If we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$,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



Implementation Details

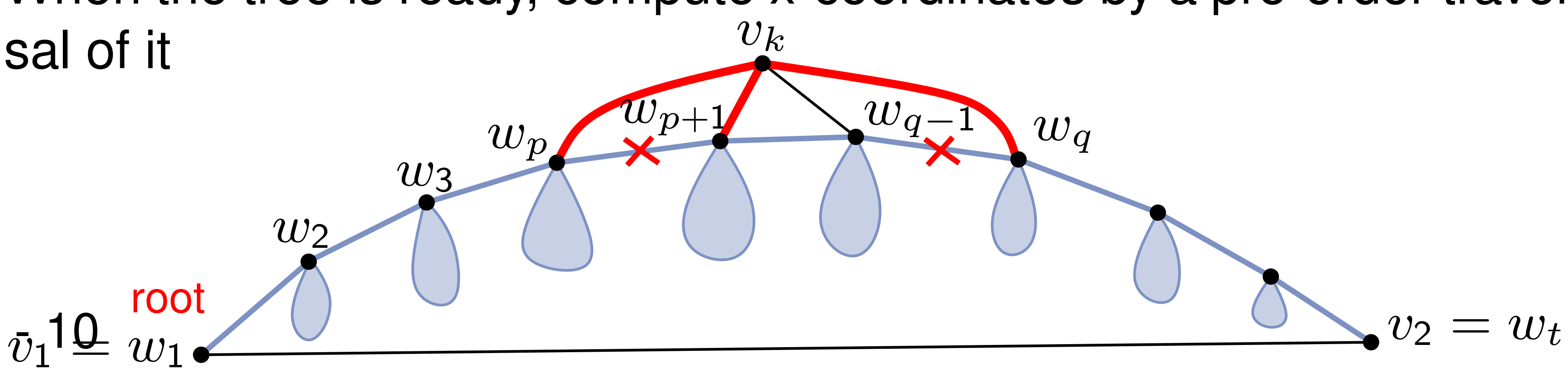
- In the binary tree at each vertex we keep its relative x-distance from its parent and its y-coordinate
- If we know the y-coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$,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

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



Implementation Details

- In the binary tree 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 $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$, here $\Delta_x(w_q)$ is x-distance from the parent, $\Delta_x(w_p, w_q)$ is x-distance of w_p and w_q
- Calculate $\Delta_x(v_k)$ by eq. (3)
- Calculate $y(v_k)$ by eq. (2)
  - $\Delta_x(w_q) = \Delta_x(w_p, w_q) - \Delta_x(v_k)$
  - $\Delta_x(w_{p+1}) = \Delta_x(w_{p+1}) - \Delta_x(v_k)$
- When the tree is ready, compute x-coordinates by a pre-order traversal of it



Summary

This lecture:

Theorem [De Fraysseix, Pach, Pollack '90]

Every n-vertex planar graph has a planar straight-line drawing of a size $(2n-4) \times (n-2)$.

Shift Algorithm

- [NR04] Book T. Nishizeki, Md. S. Rahman "'Planar Graph Drawing. Chapter 4.2.

Summary

This lecture:

Theorem [De Fraysseix, Pach, Pollack '90]

Every n-vertex planar graph has a planar straight-line drawing of a size $(2n-4) \times (n-2)$.

Next lecture:

Theorem [Schnyder '90]

Every n-vertex planar graph has a planar straight-line drawing of a size $(n-2) \times (n-2)$.