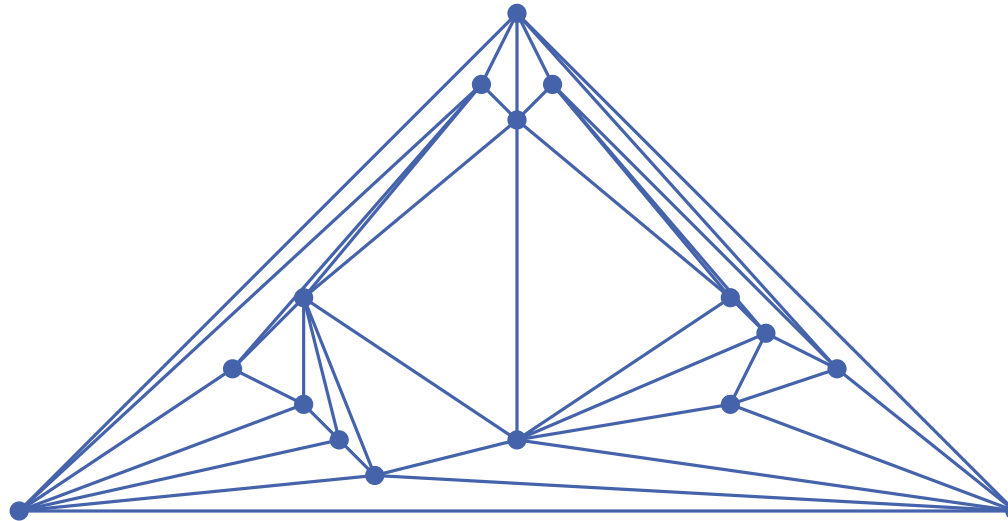


Algorithms for graph visualization

Layouts for planar graphs. Shift method.

WINTER SEMESTER 2018/2019

Tamara Mchedlidze



1

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

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

- 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.

- 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

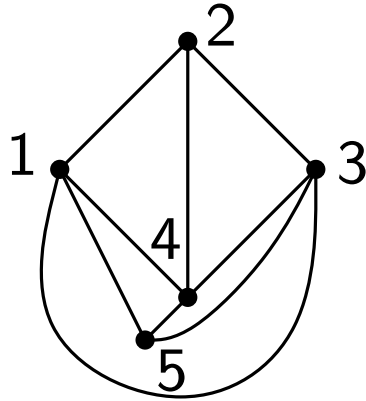
By far the most agreed-upon edge placement heuristic is to minimize the number of edge crossings in a graph [BMRW98, Har98, DH96, Pur02, TR05, TBB88]. The importance of avoiding edge crossings has also been extensively validated in terms of user preference and performance (see Section 4). Similarly, 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.

- 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

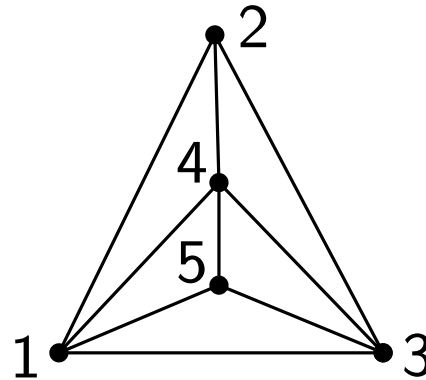
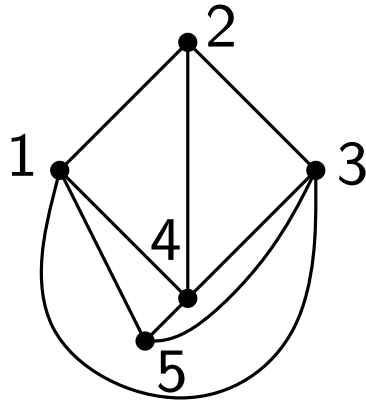
By far the most agreed-upon edge placement heuristic is to minimize the number of edge crossings in a graph [BMRW98, Har98, DH96, Pur02, TR05, TBB88]. The importance of avoiding edge crossings has also been extensively validated in terms of user preference and performance (see Section 4). Similarly, 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.

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



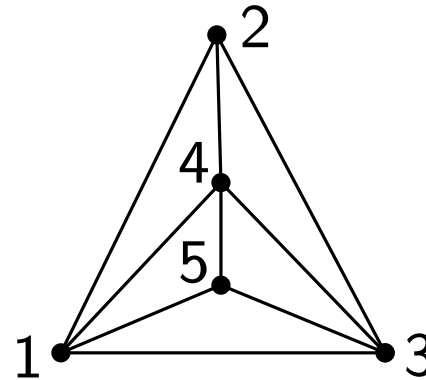
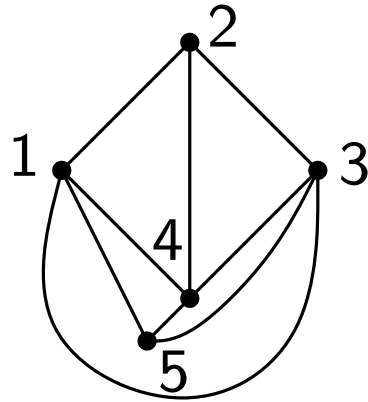
3 - 1

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



3 - 2

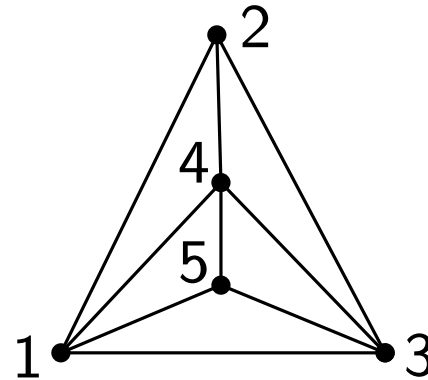
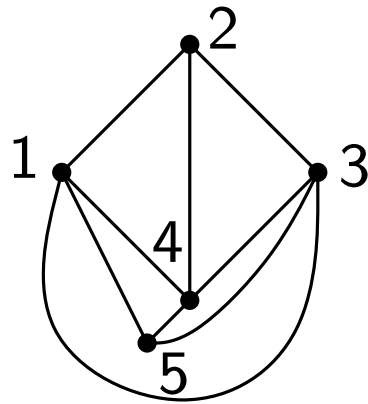
- Does every planar graph have a planar straight-line drawing?



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

Every planar graph has a planar straight-line drawing.

- 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

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.
- Implementational details.

Outline

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

Definition: Canonical Ordering

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

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

Definition: Canonical Ordering

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

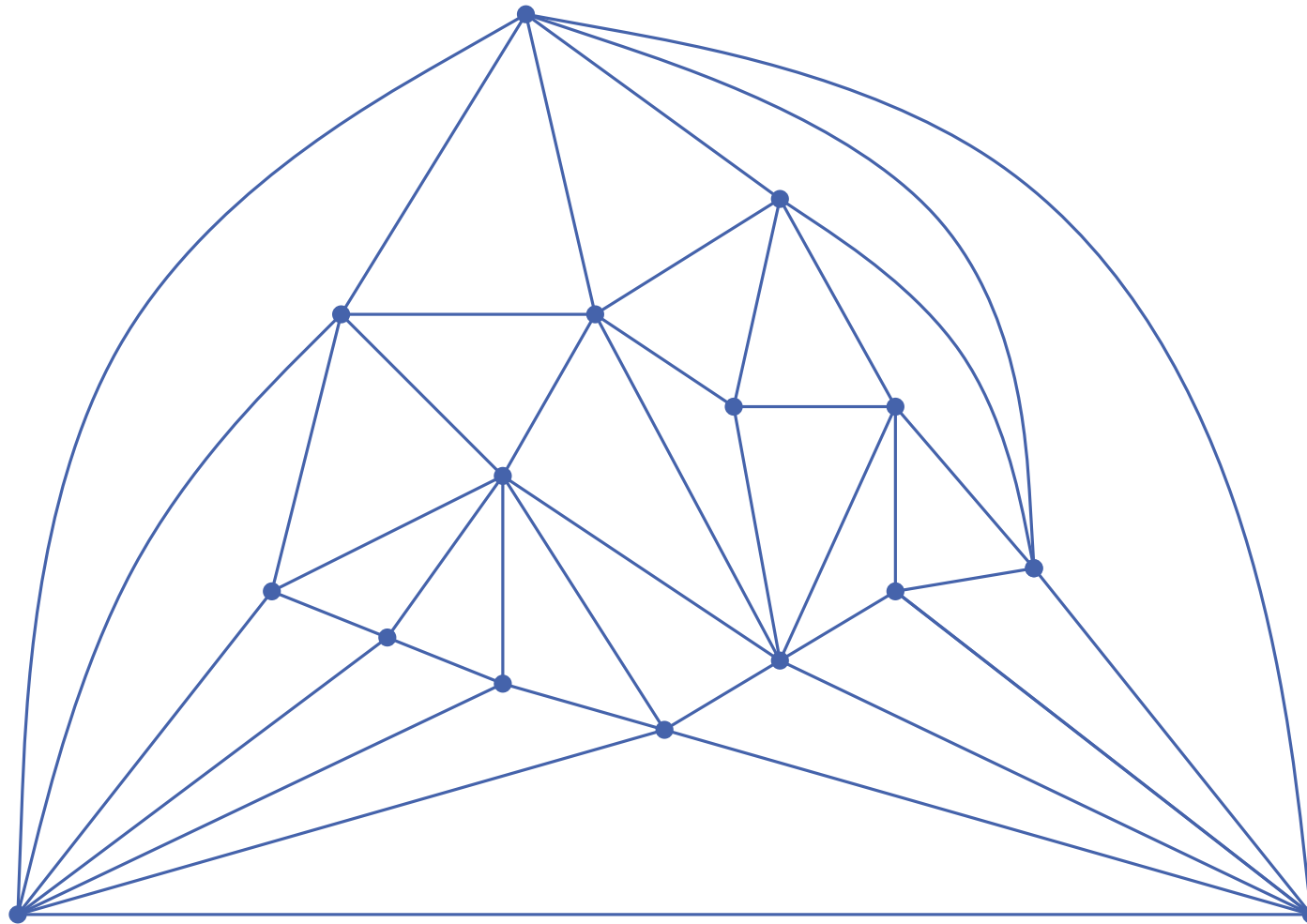
- (C1) Vertices $\{v_1, \dots, v_k\}$ induce a 2-connected internally triangulated graph, call it G_k
- (C2) Edge (v_1, v_2) belongs to the outer face of G_k

Definition: Canonical Ordering

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

- (C1) Vertices $\{v_1, \dots, v_k\}$ 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

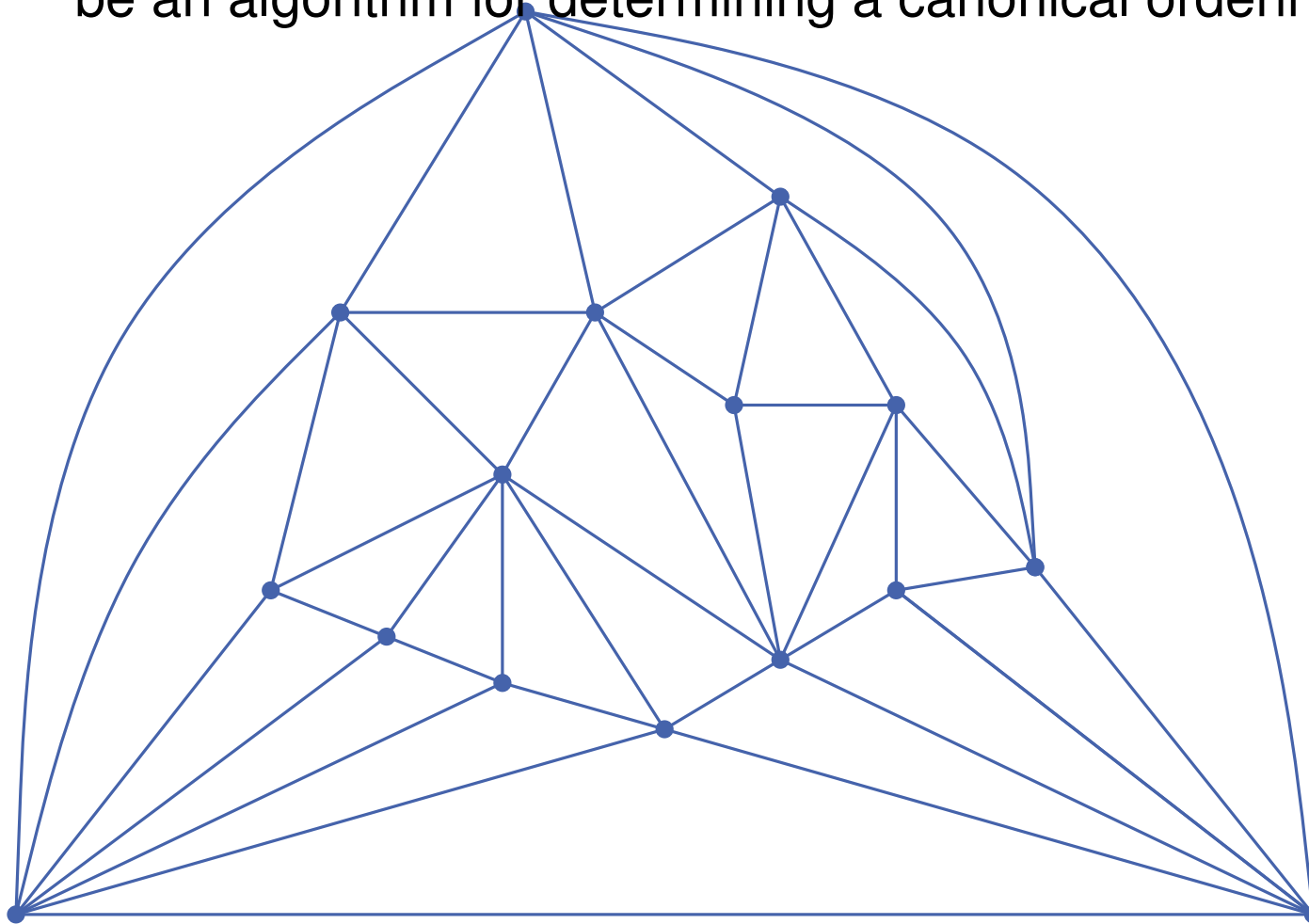
Example of Canonical Ordering



Work with your neighbour(s) and then share

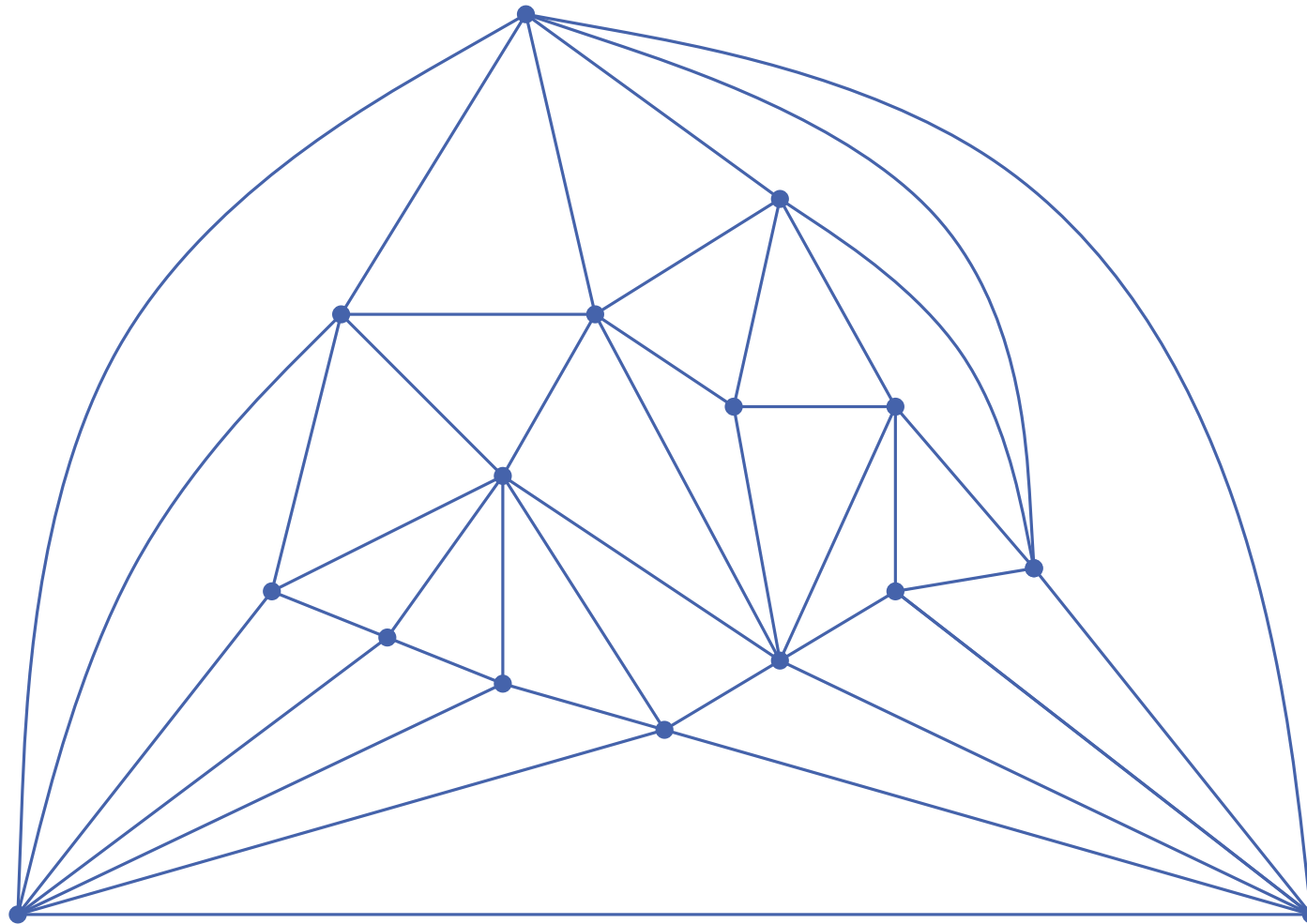
5 min

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



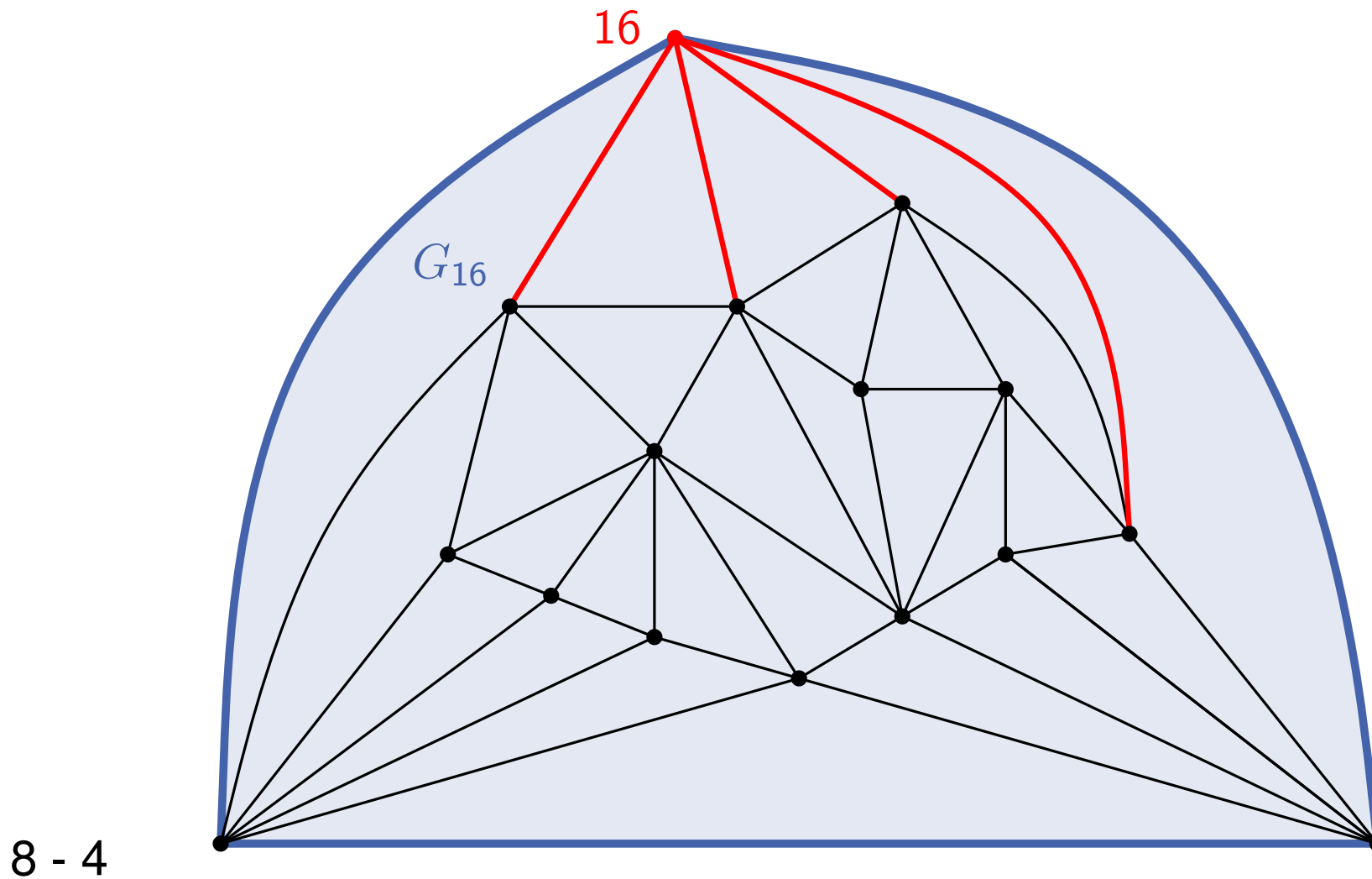
8 - 2

Example of Canonical Ordering

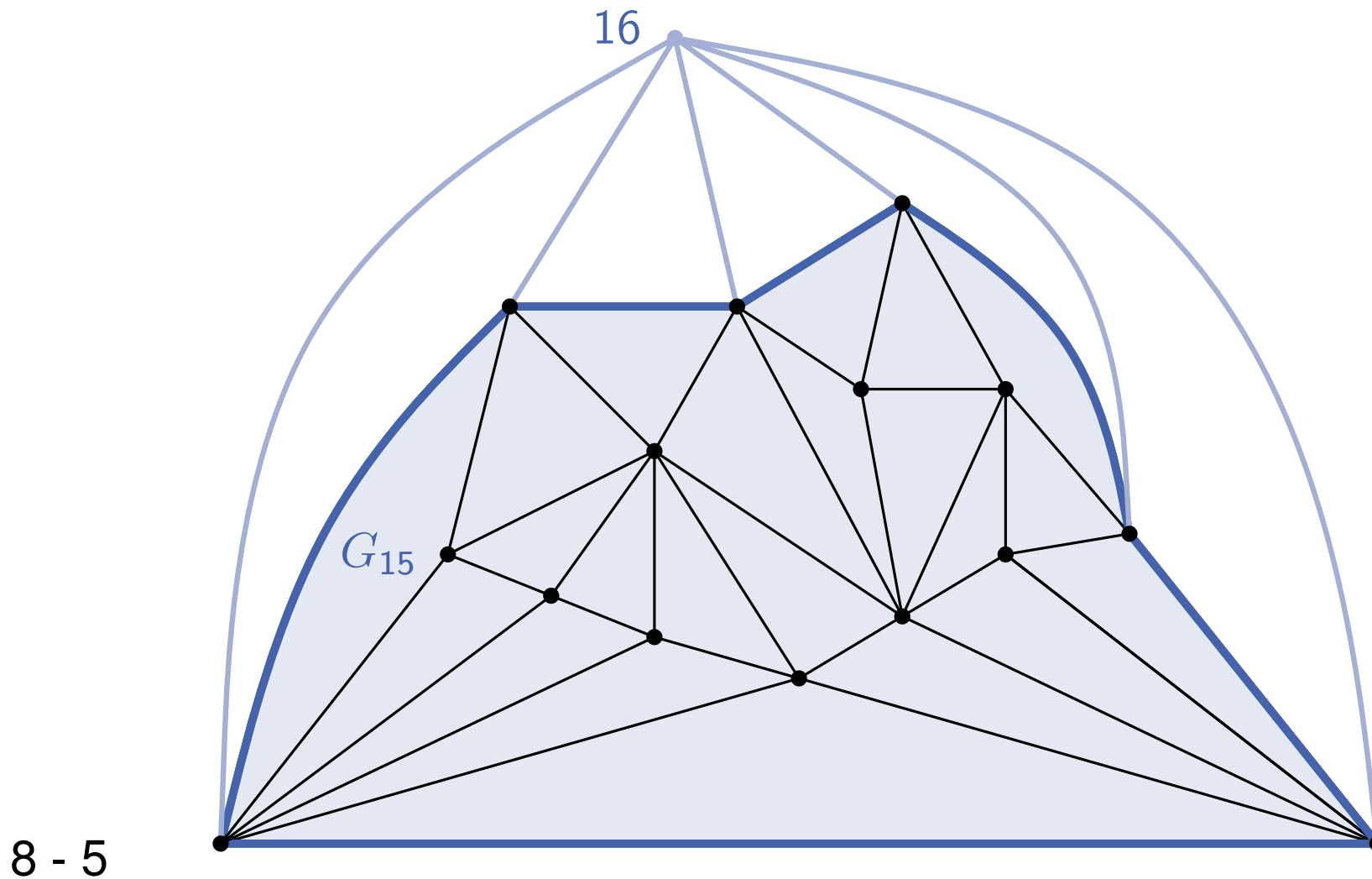


8 - 3

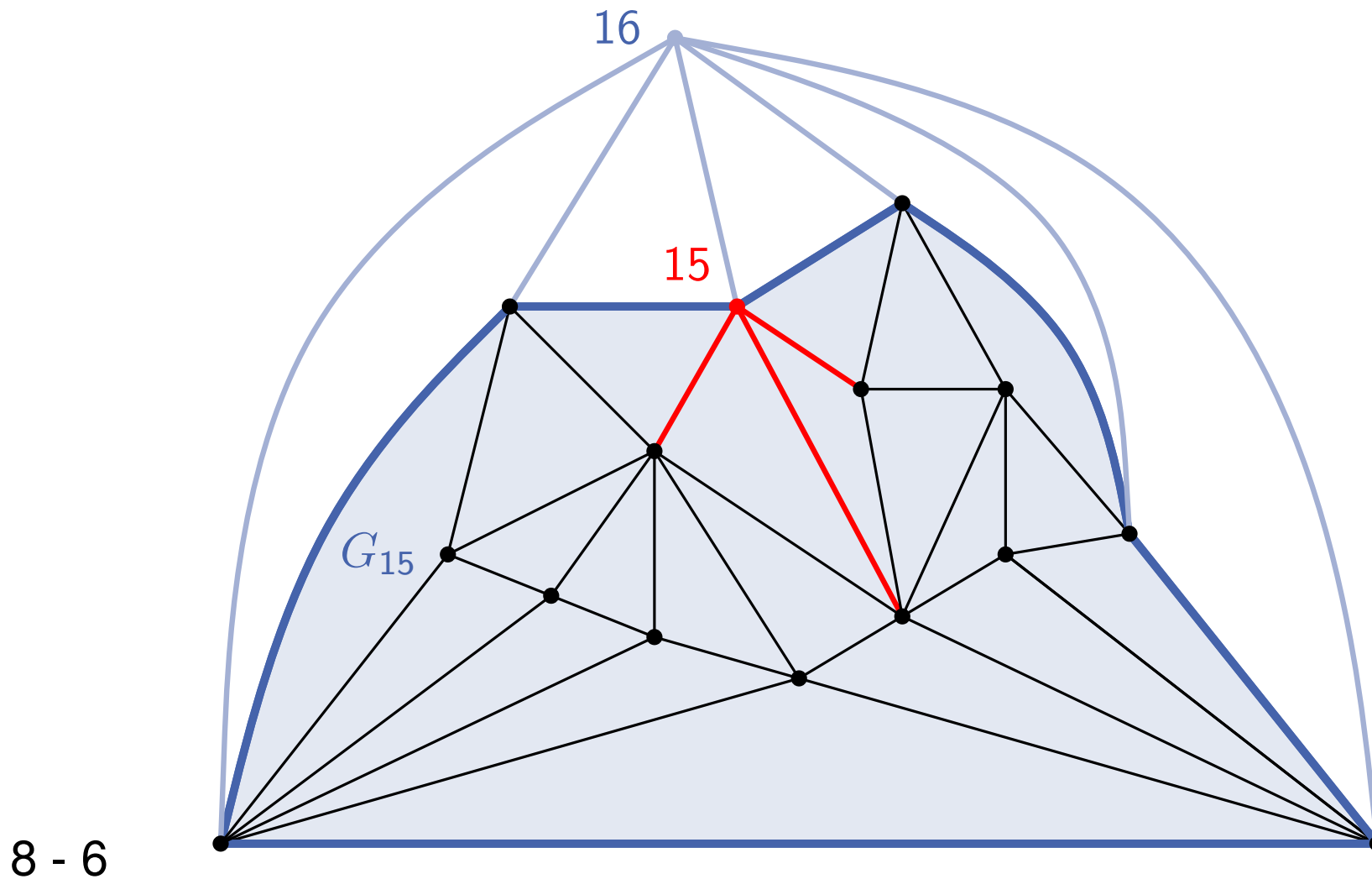
Example of Canonical Ordering



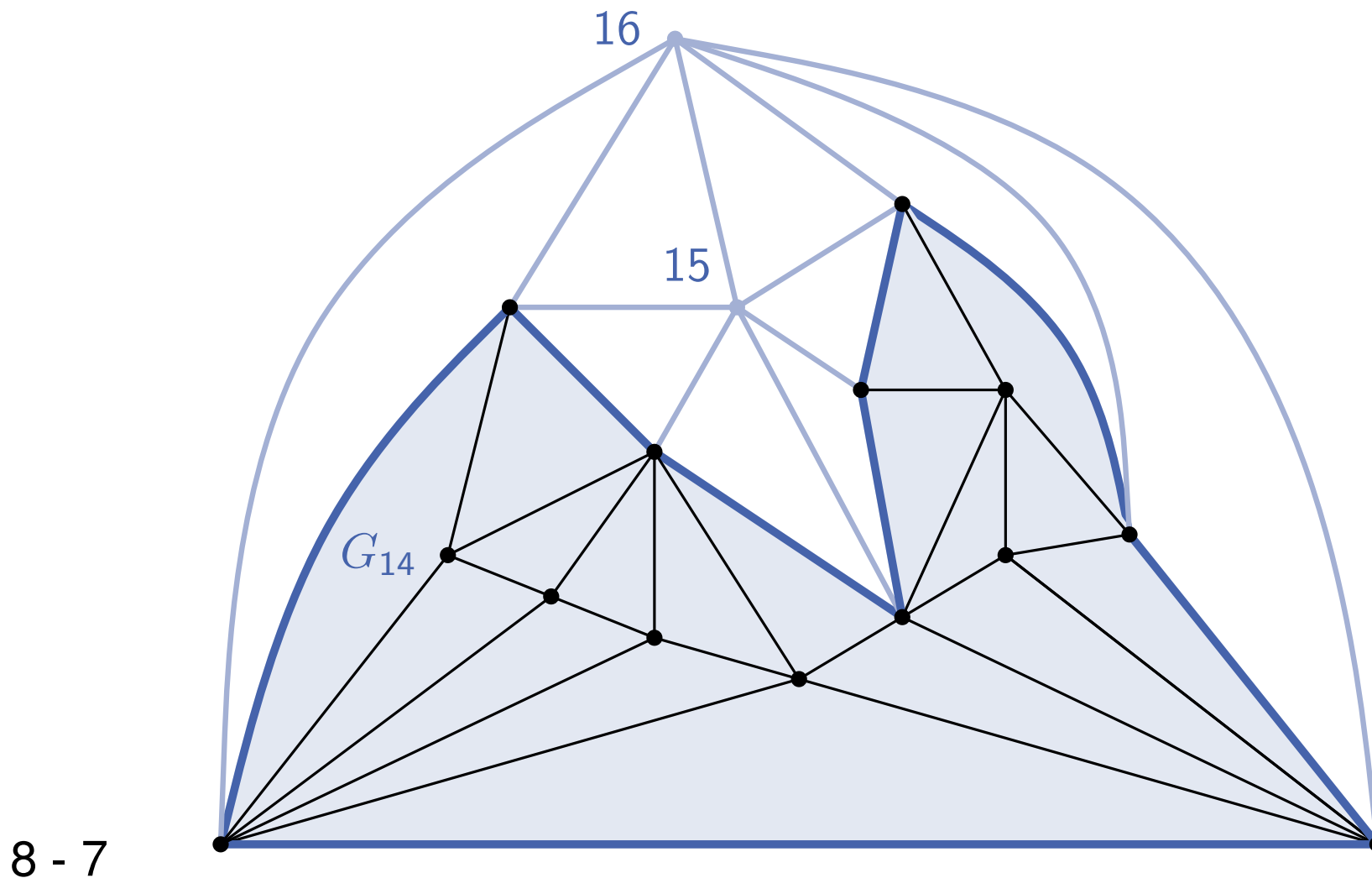
Example of Canonical Ordering



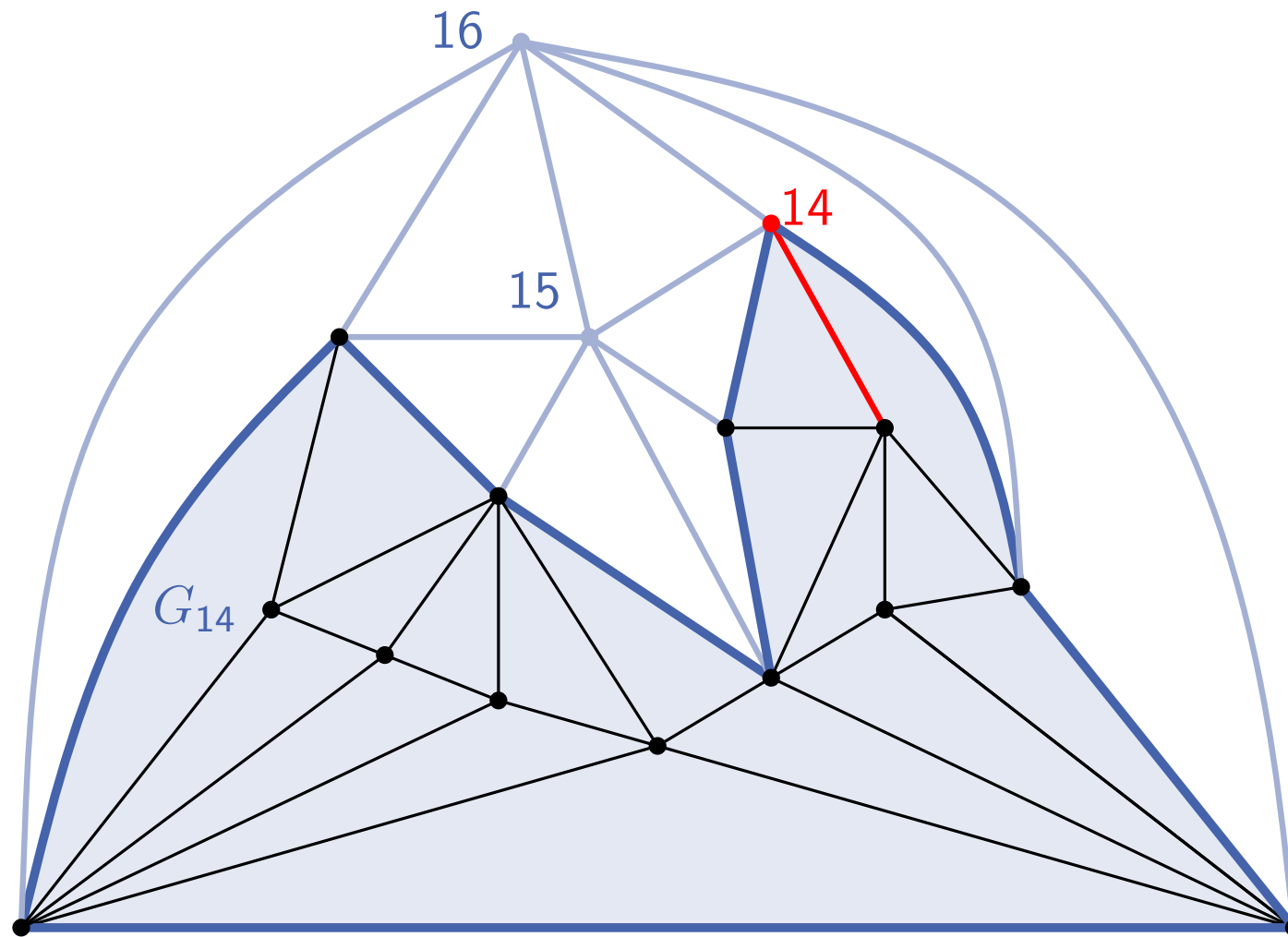
Example of Canonical Ordering



Example of Canonical Ordering

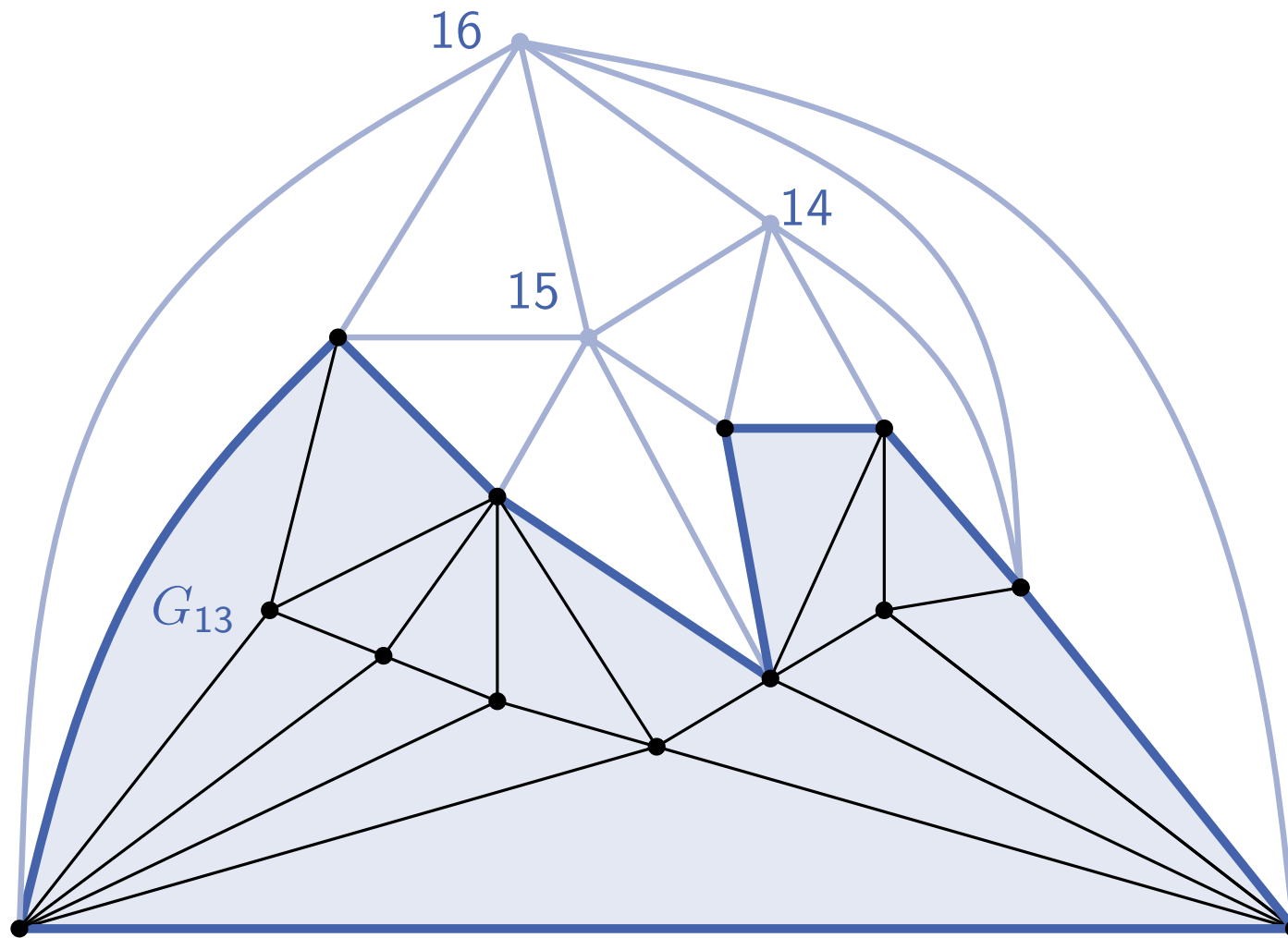


Example of Canonical Ordering



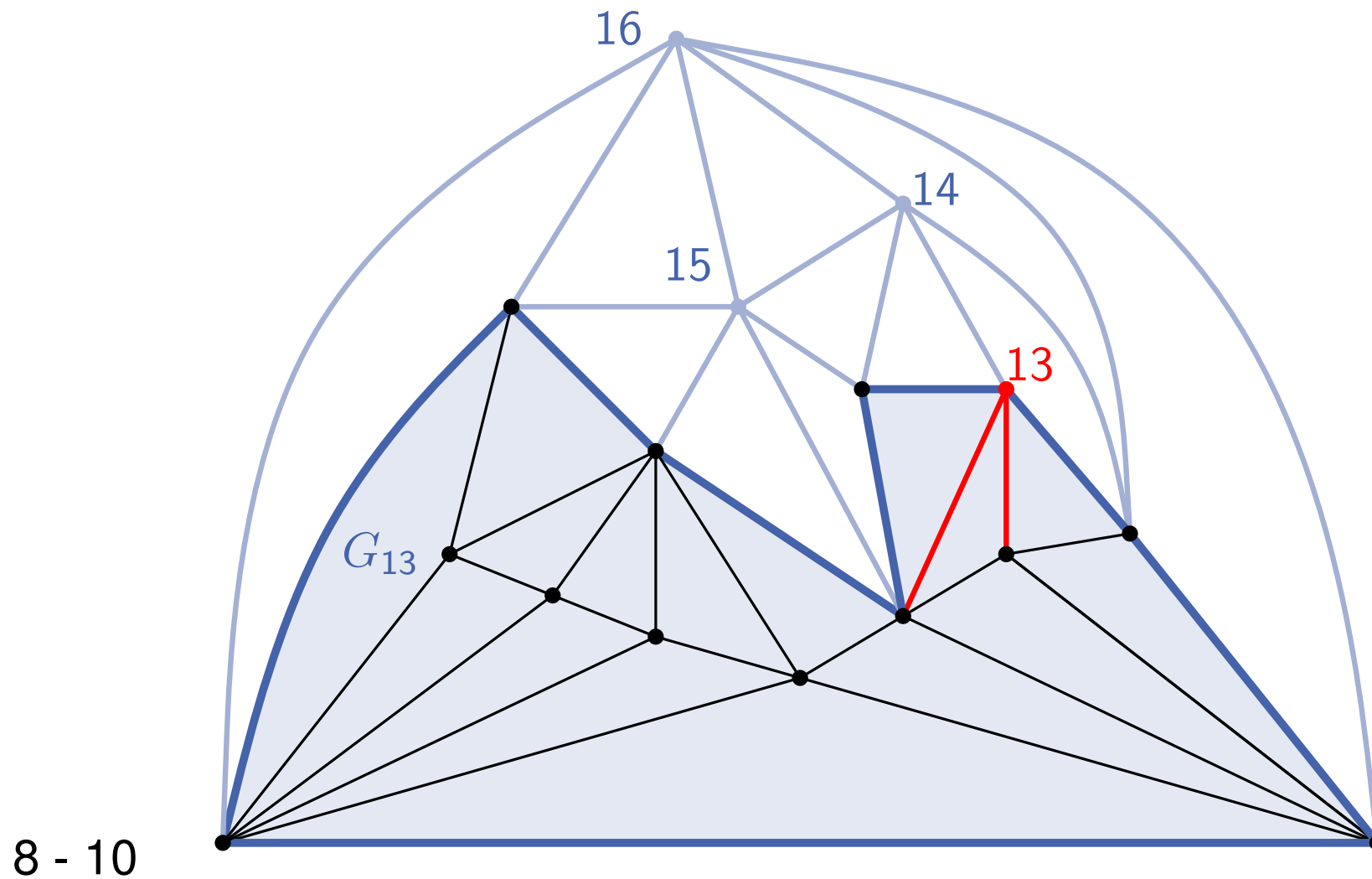
8 - 8

Example of Canonical Ordering

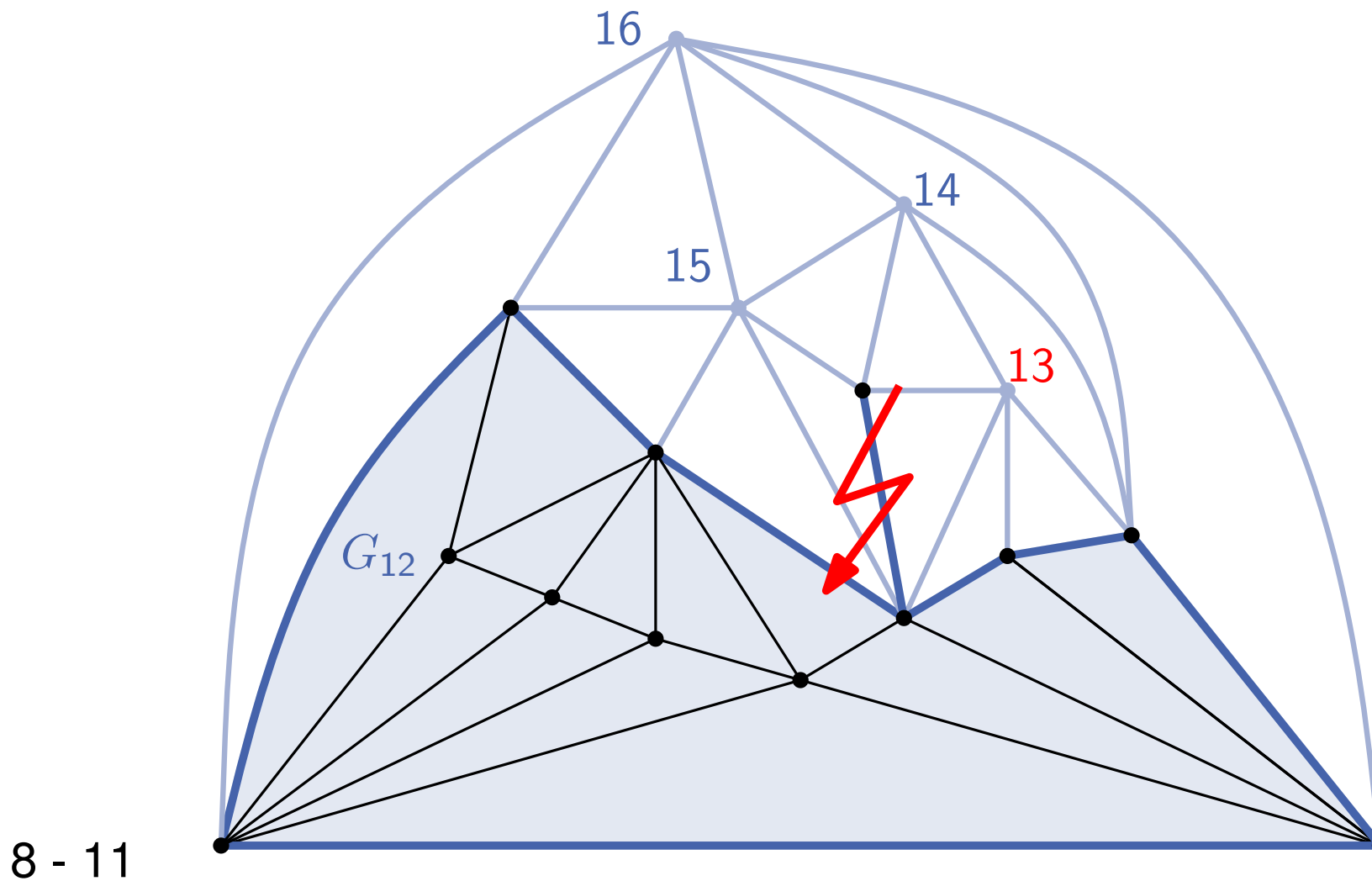


8 - 9

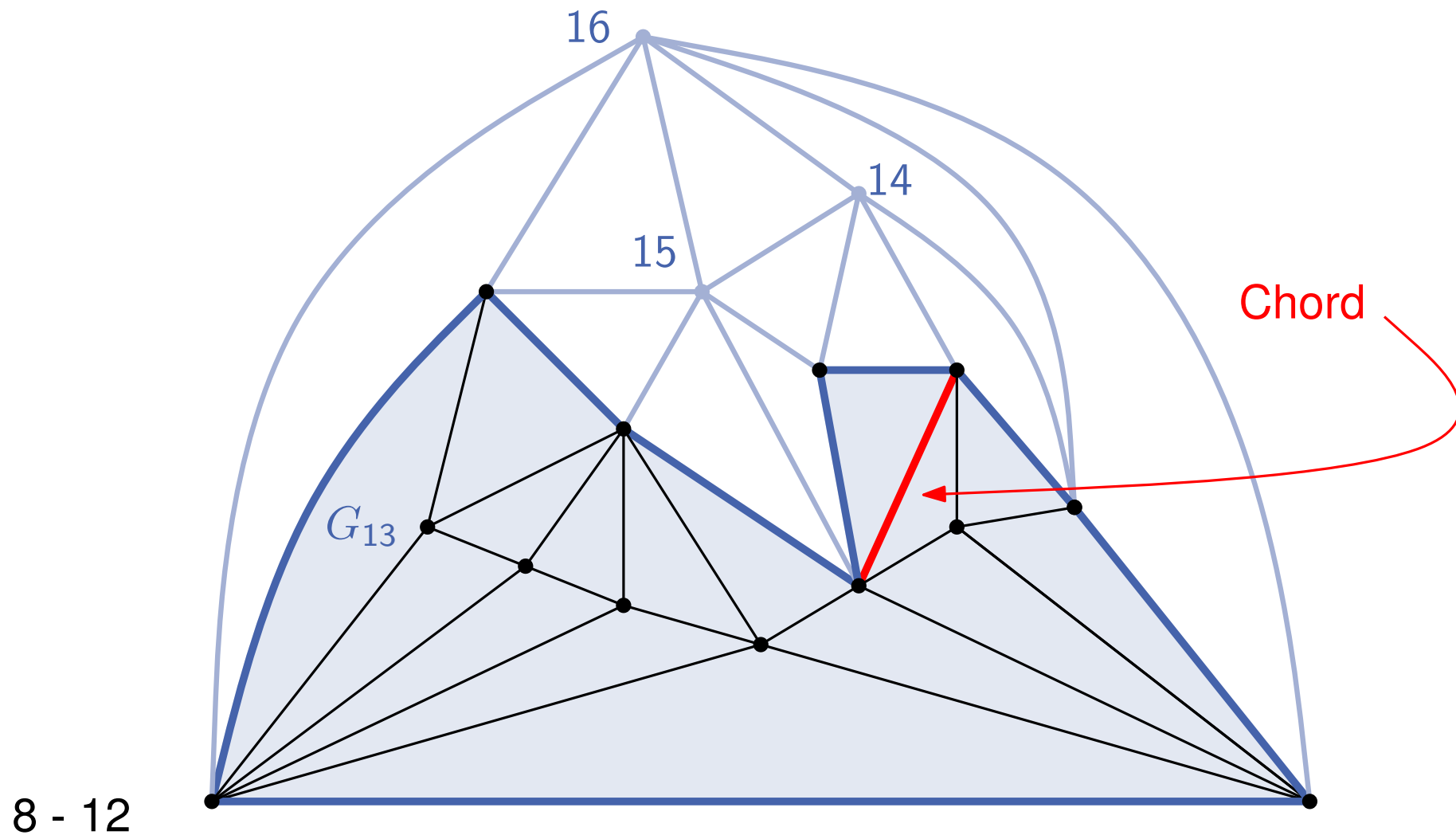
Example of Canonical Ordering



Example of Canonical Ordering

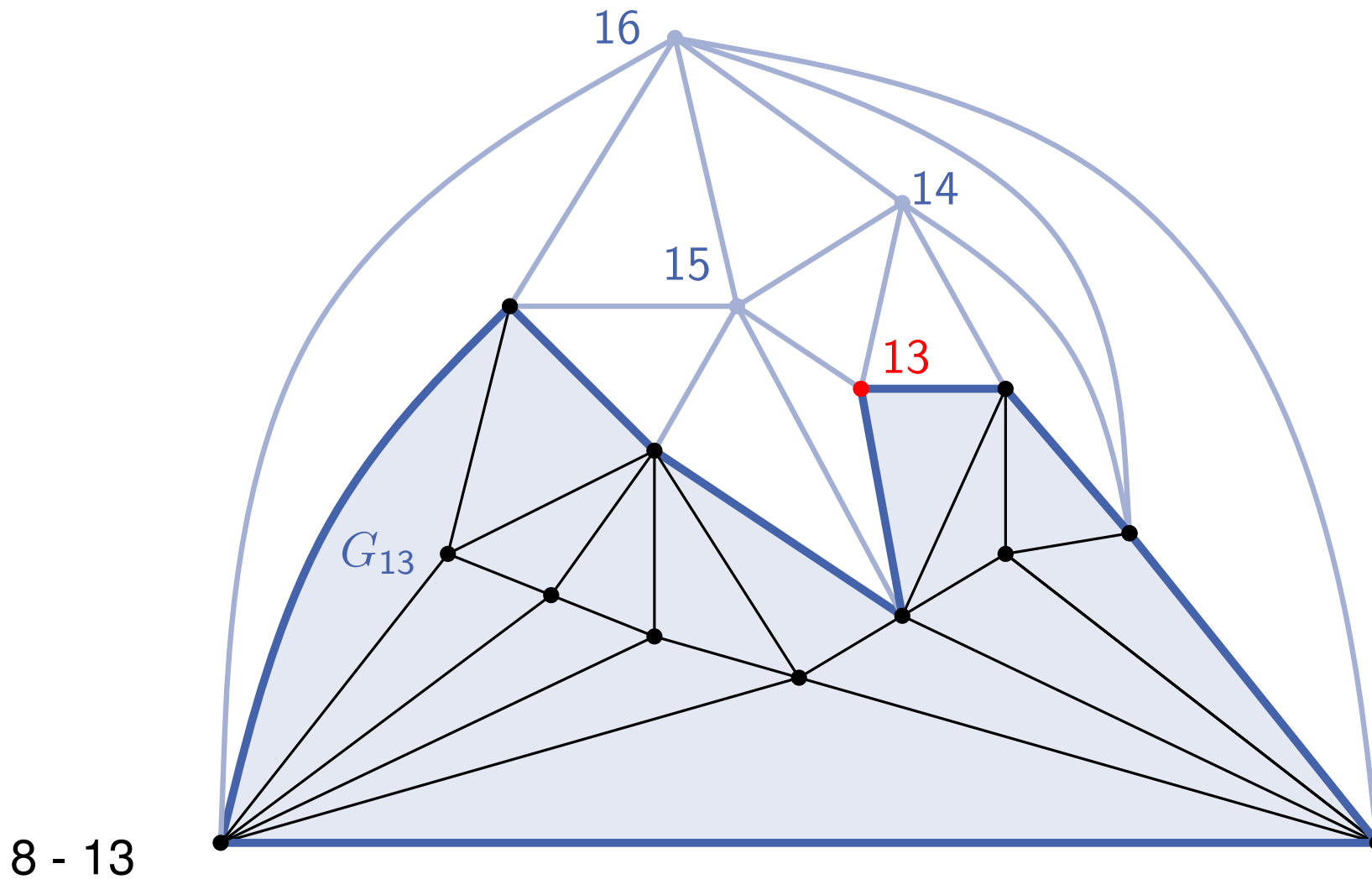


Example of Canonical Ordering

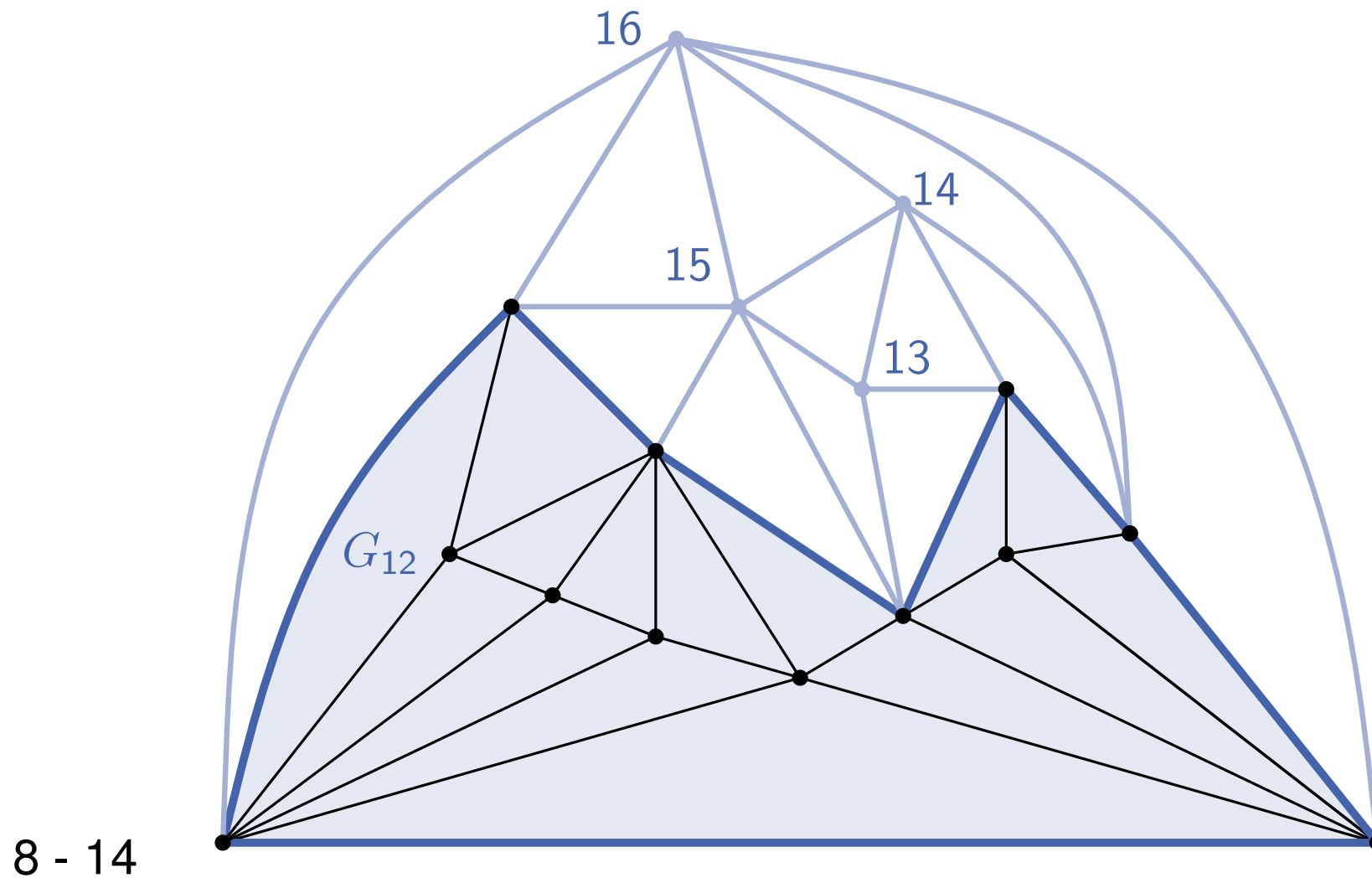


8 - 12

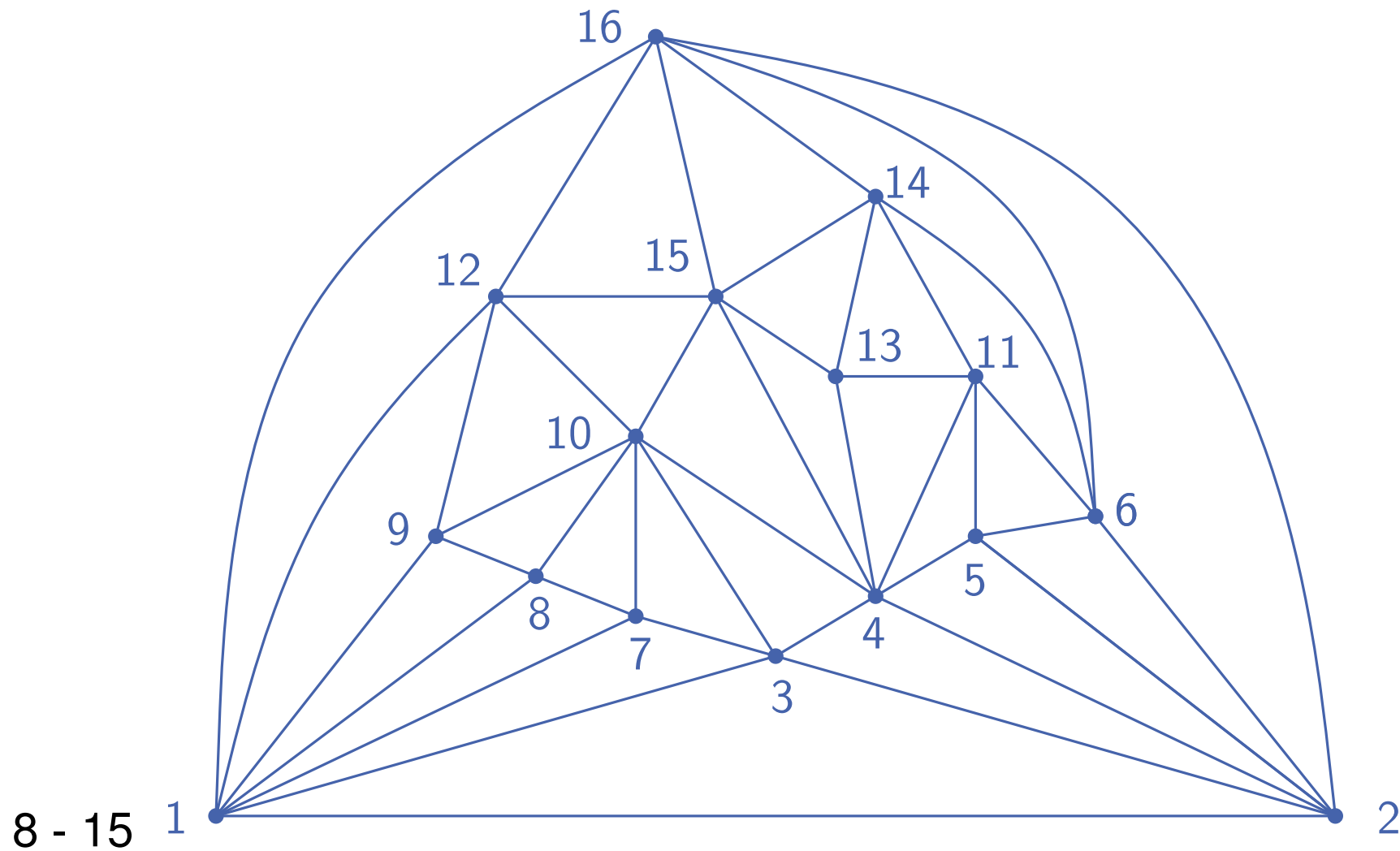
Example of Canonical Ordering



Example of Canonical Ordering



Example of Canonical Ordering



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.

Lemma

Every triangulated plane graph has a canonical ordering.

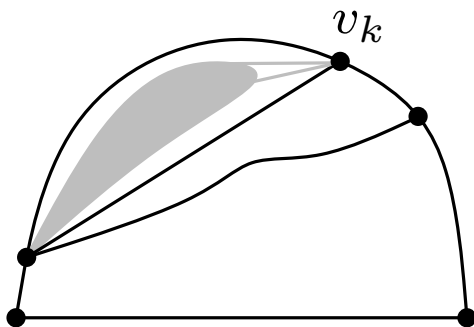
- Let $G_n = G$, and let v_1, v_2, v_n be the vertices of the outer face of G_n . Conditions C1-C3 hold.
- Induction hypothesis: vertices v_{n-1}, \dots, v_{k+1} have been chosen such that conditions C1-C3 hold for $k + 1 \leq i \leq n$.

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}, \dots, 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 .

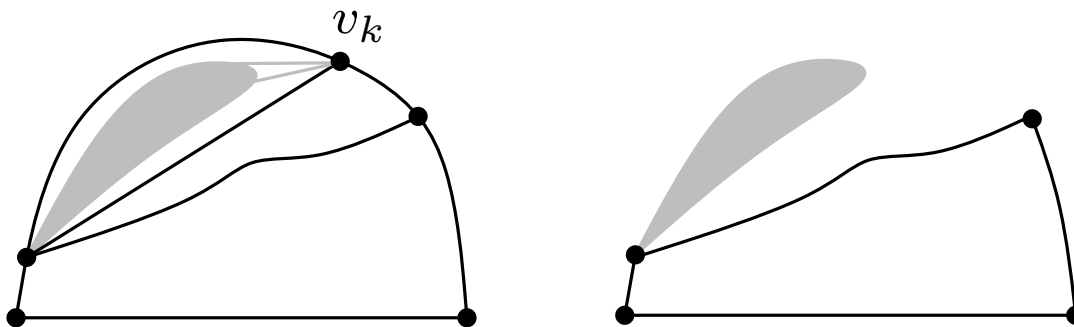


9 - 3

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}, \dots, 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 .

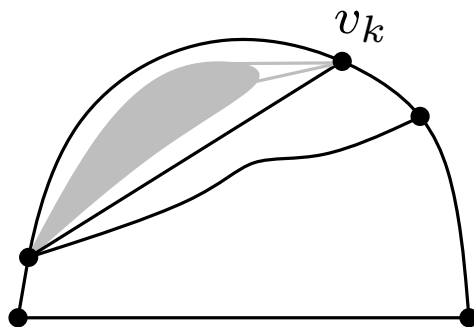


9 - 4

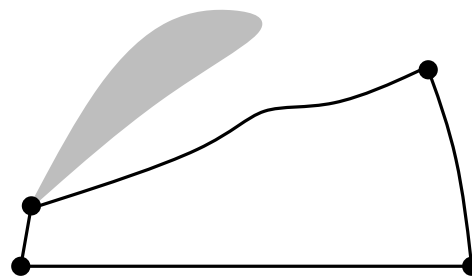
Lemma

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}, \dots, 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 .



9 - 5



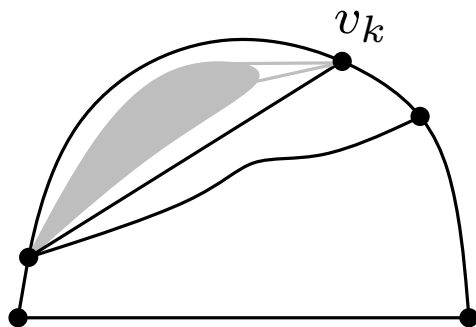
v_k should not be adjacent to a chord

Canonical Ordering. Existence.

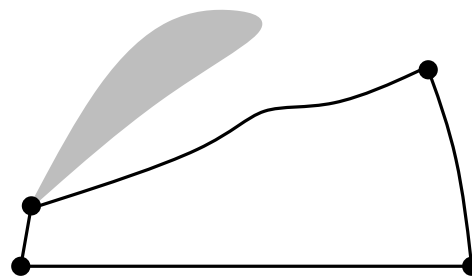
Lemma

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}, \dots, 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 .



9 - 6

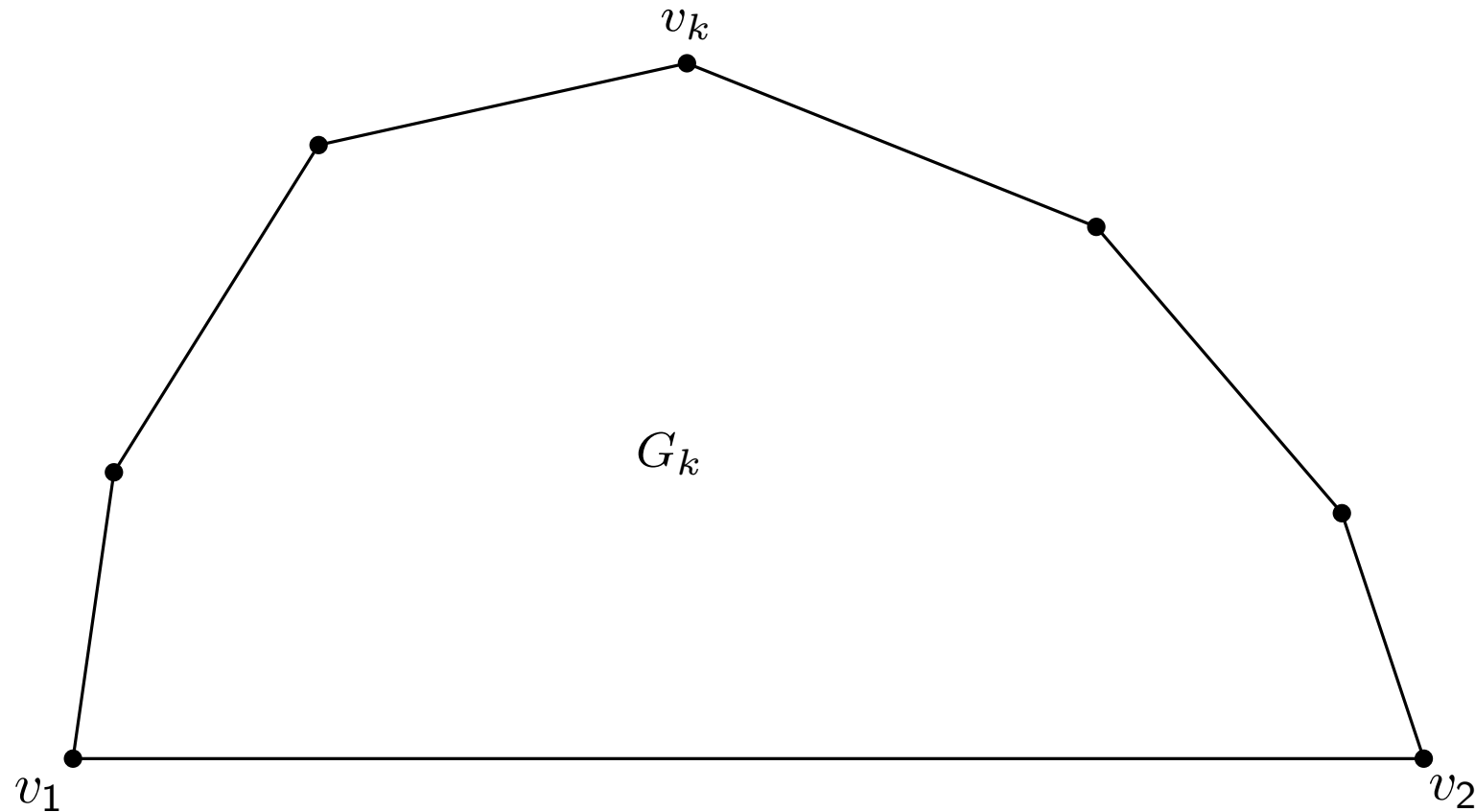


v_k should not be adjacent to a chord

Is it sufficient?

Canonical Ordering. Existence.

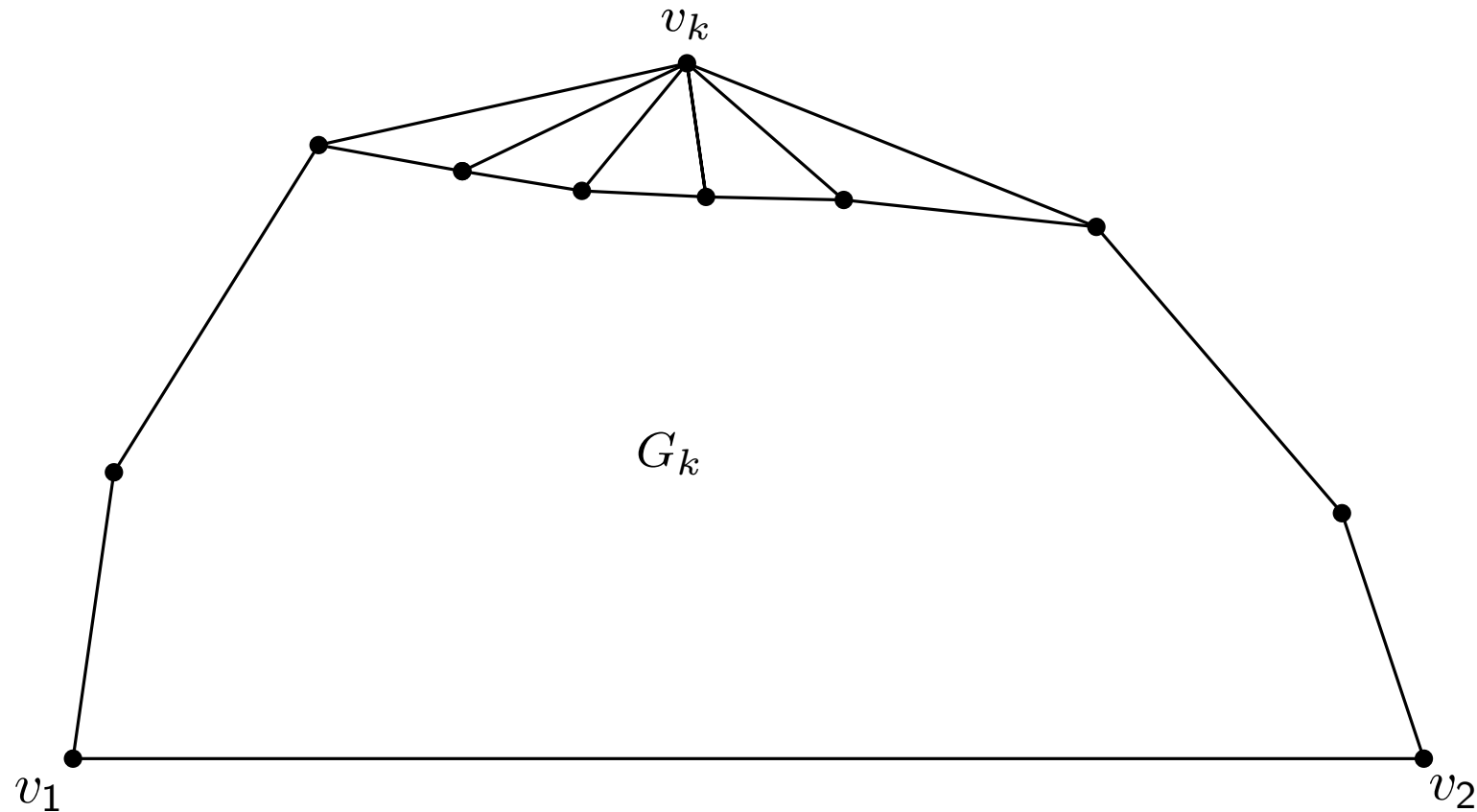
Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



10 - 1

Canonical Ordering. Existence.

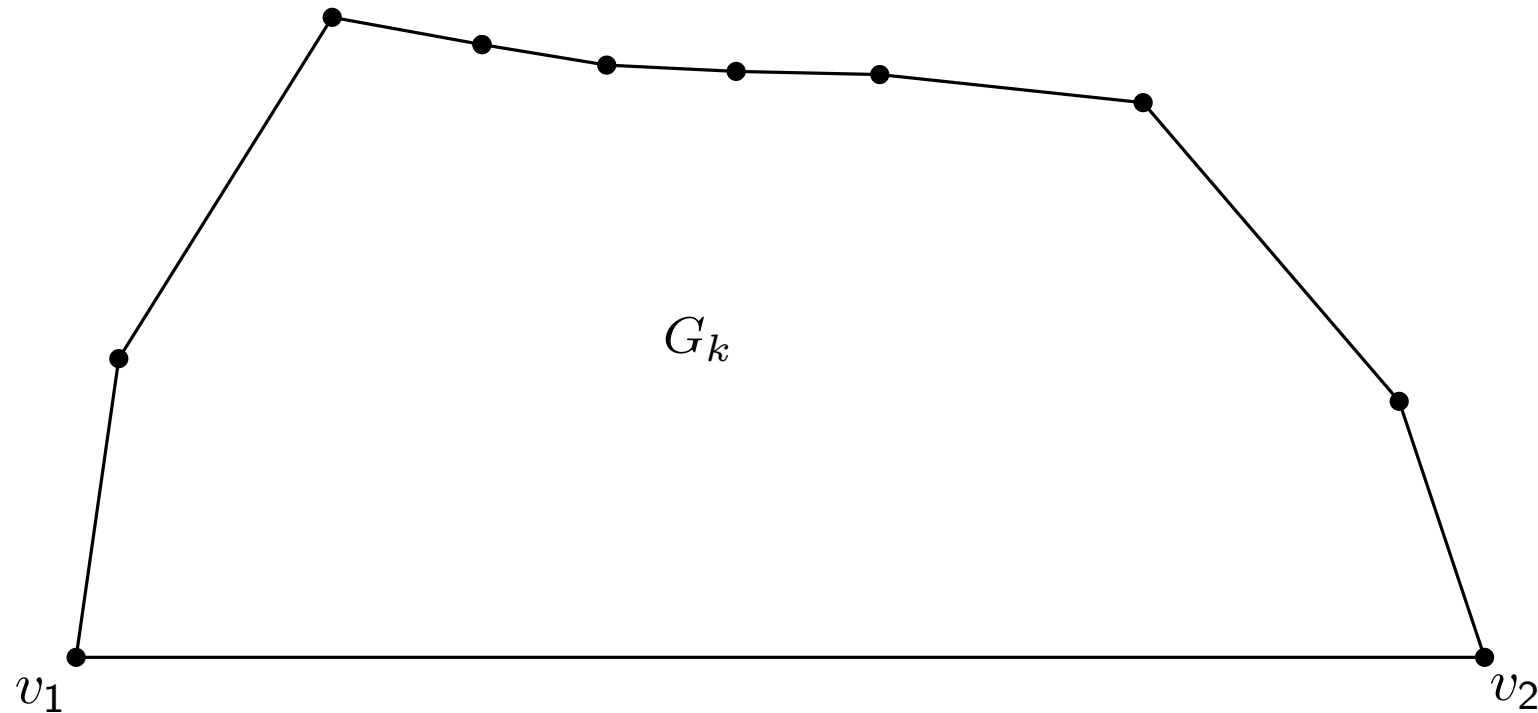
Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



10 - 2

Canonical Ordering. Existence.

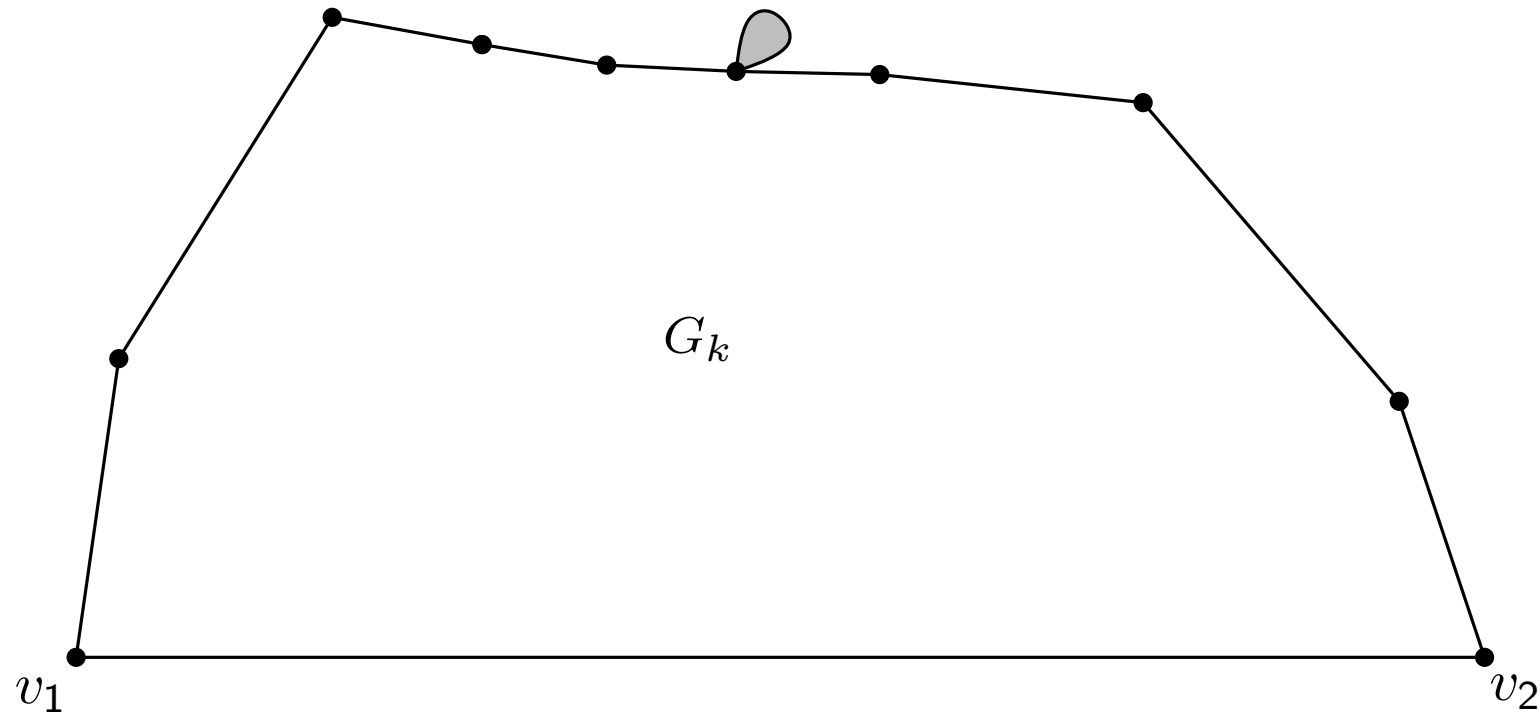
Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



10 - 3

Canonical Ordering. Existence.

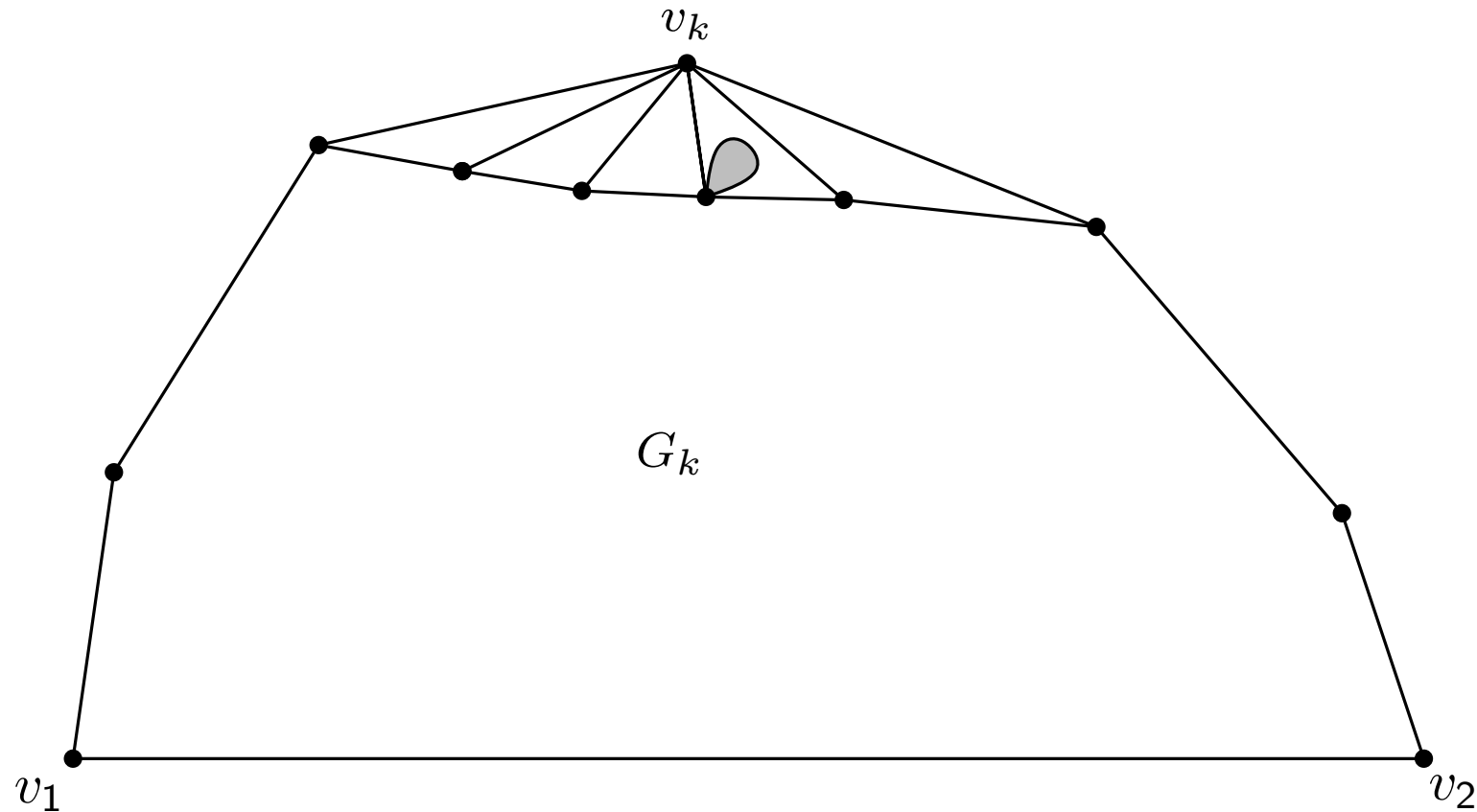
Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



10 - 4

Canonical Ordering. Existence.

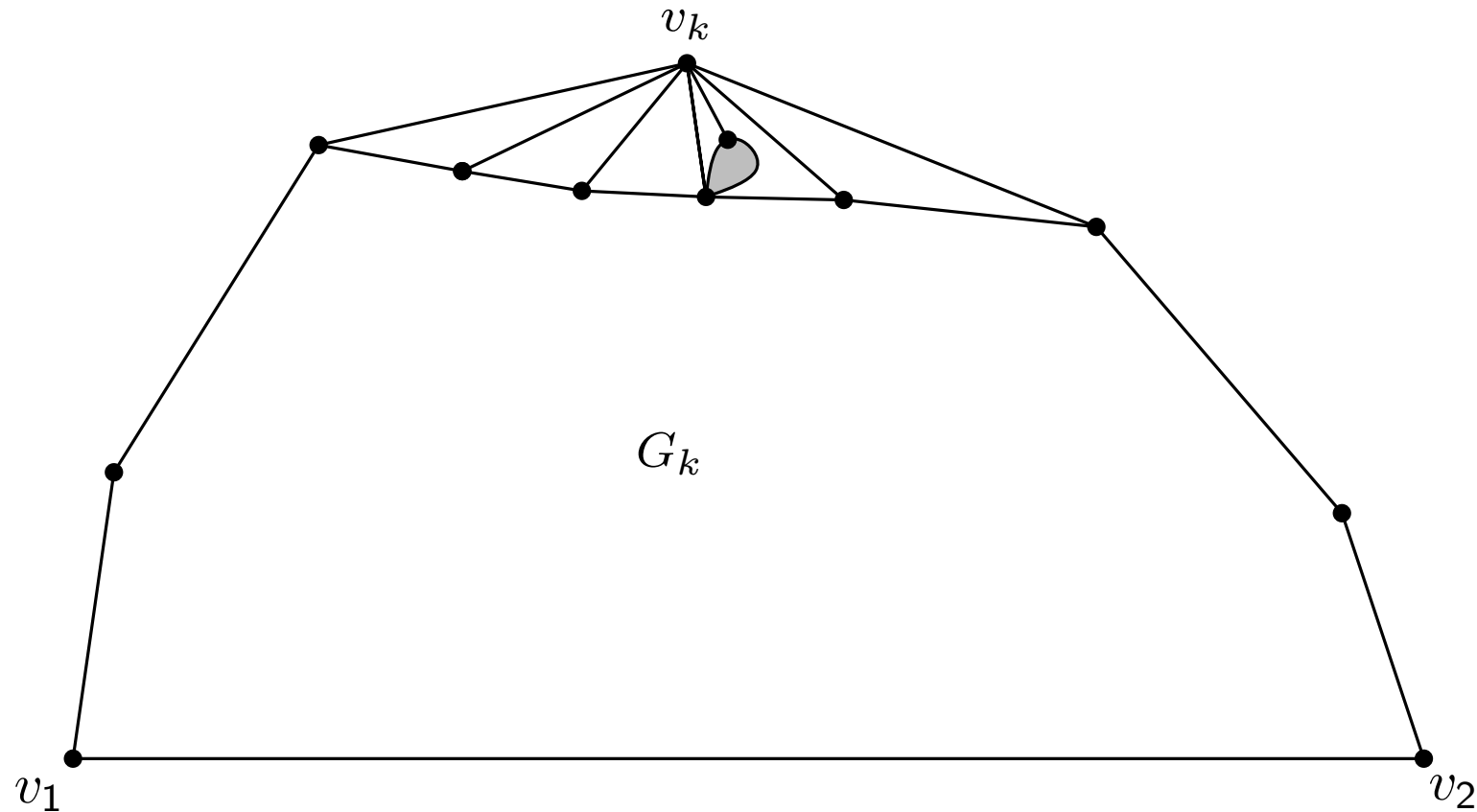
Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



10 - 5

Canonical Ordering. Existence.

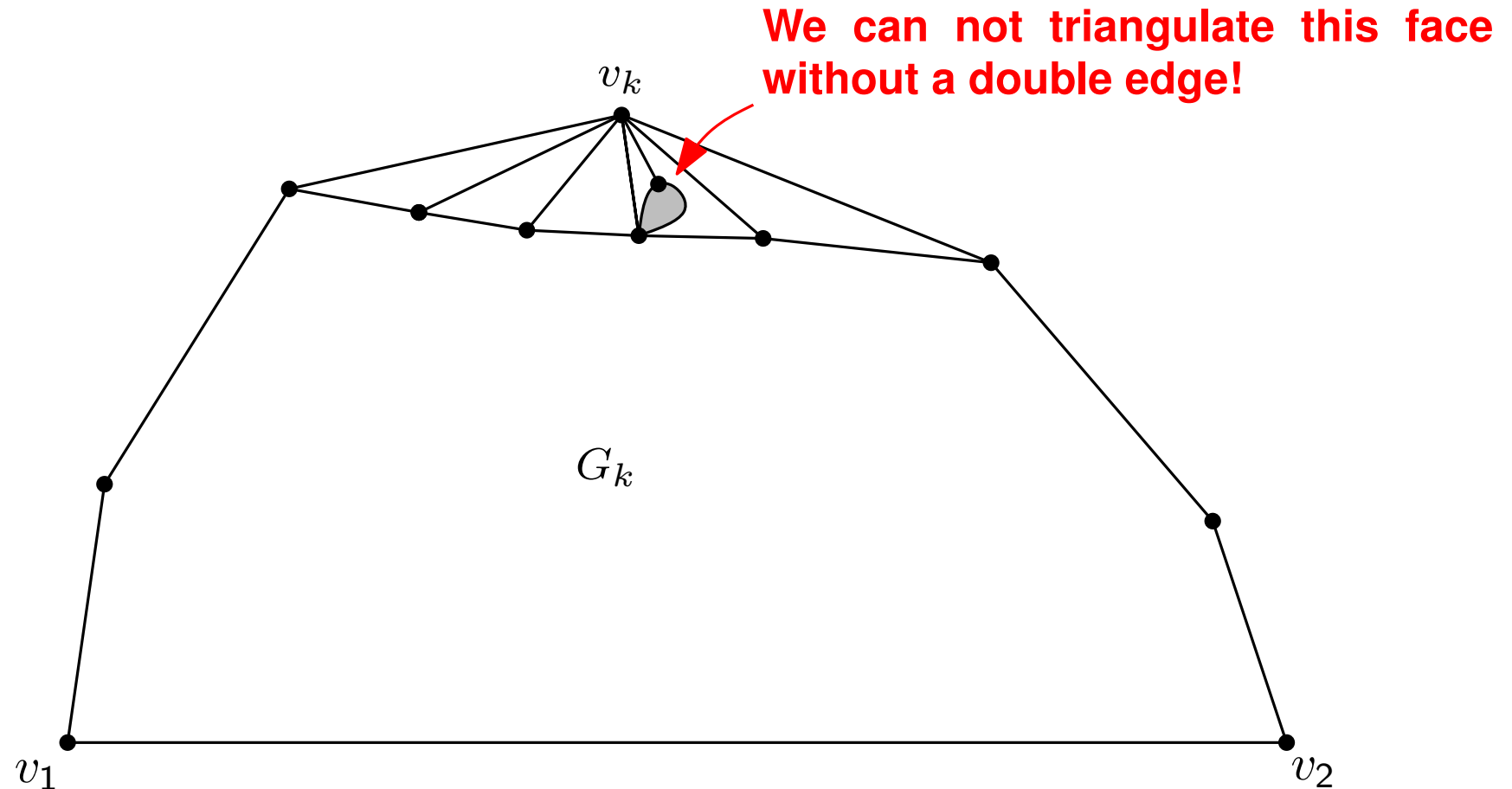
Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



10 - 6

Canonical Ordering. Existence.

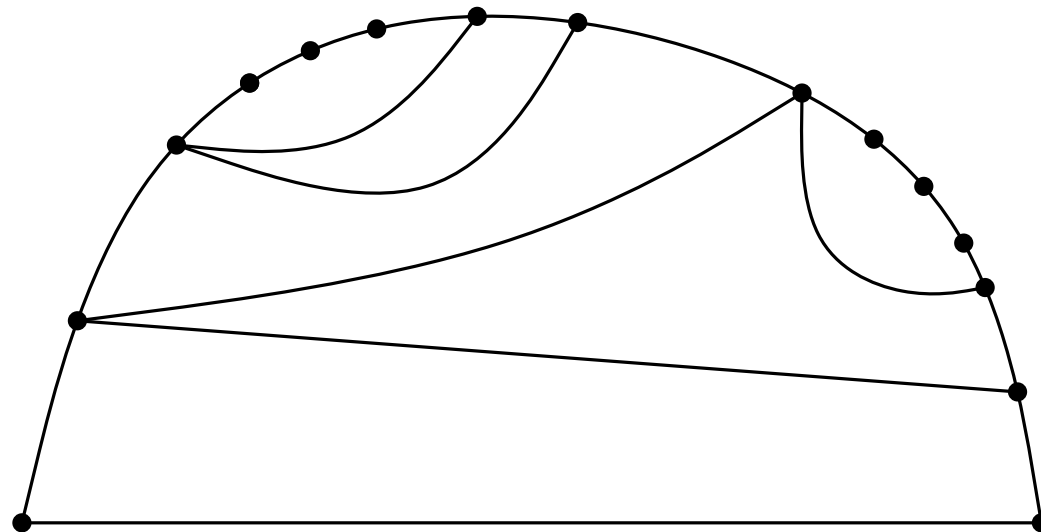
Statement If v_k is not adjacent to a chord then removal of v_k leaves the graph biconnected.



10 - 7

Canonical Ordering. Existence.

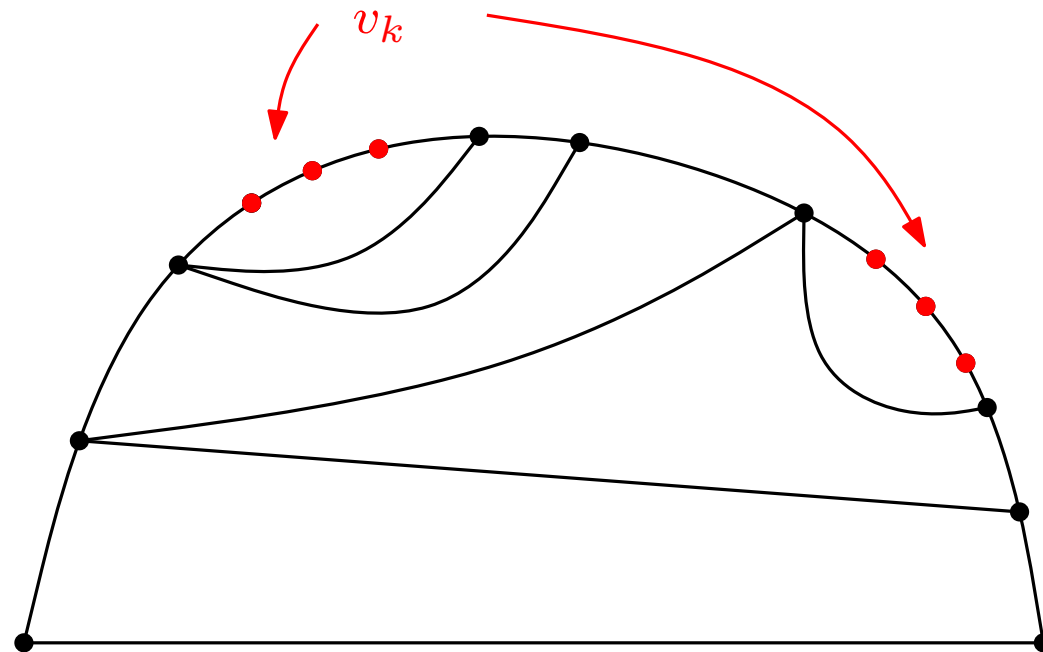
- Why a vertex not adjacent to a chord exists?



11 - 1

Canonical Ordering. Existence.

- Why a vertex not adjacent to a chord exists?



11 - 2

Outline

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

Algorithm CO

```
forall  $v \in V$  do  
   $\lfloor$  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  $w_1 = v_1, w_2, \dots, w_{t-1}, w_t = v_2$  denote the boundary of  $G_{k-1}$ ;  
  // and let  $w_p, \dots, w_q$  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

13 - 1 ■ out(v)=true iff v is the outer vertex of current plane graph

Algorithm CO

```
forall  $v \in V$  do  
   $\lfloor$  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  $w_1 = v_1, w_2, \dots, w_{t-1}, w_t = v_2$  denote the boundary of  $G_{k-1}$ ;  
  // and let  $w_p, \dots, w_q$  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

13 - ■ out(v)=true iff v is the outer vertex of current plane graph

Algorithm CO

```
forall  $v \in V$  do  
   $\lfloor$  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  $w_1 = v_1, w_2, \dots, w_{t-1}, w_t = v_2$  denote the boundary of  $G_{k-1}$ ;  
    // and let  $w_p, \dots, w_q$  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

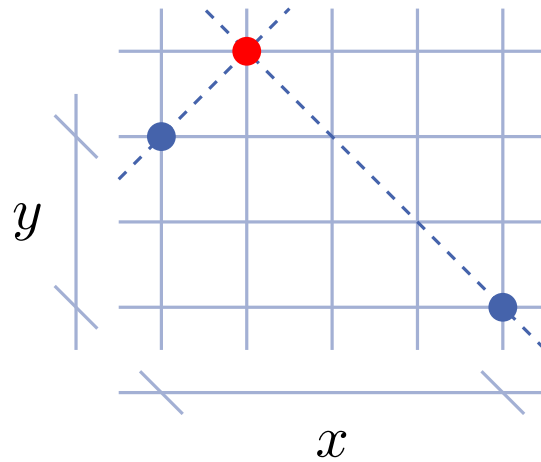
Algorithm CO computes a canonical ordering of a graph in $O(n)$ time.

Outline

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

De Fraysseix Pach Pollack (Shift) Algorithm

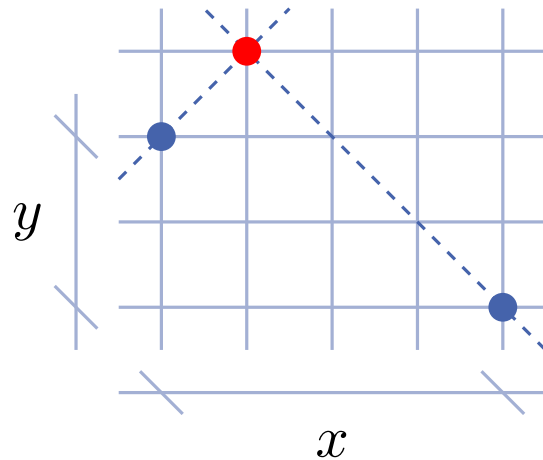
even Manhattan distance



15 - 1

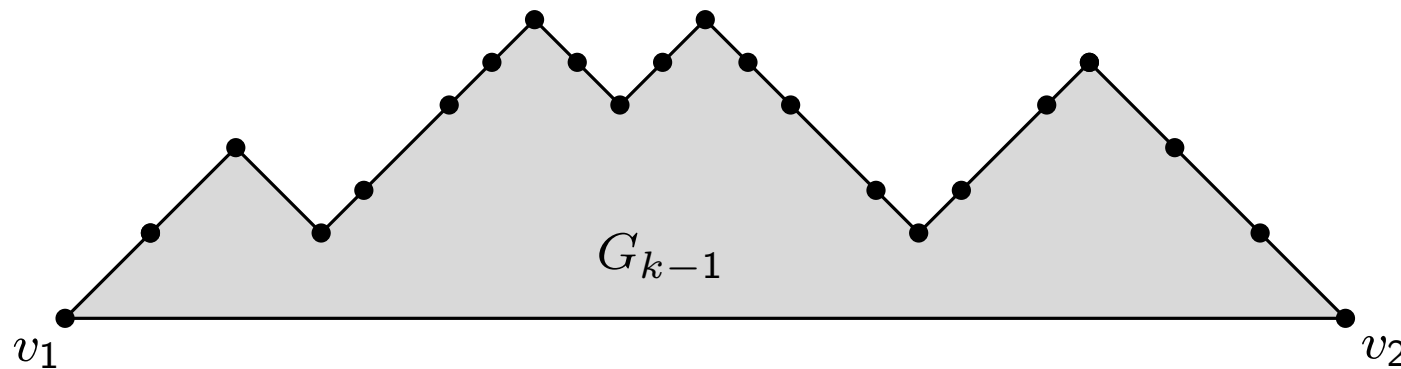
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

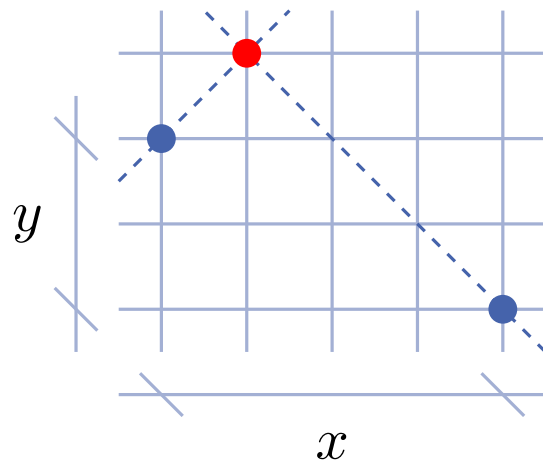
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x -monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 2

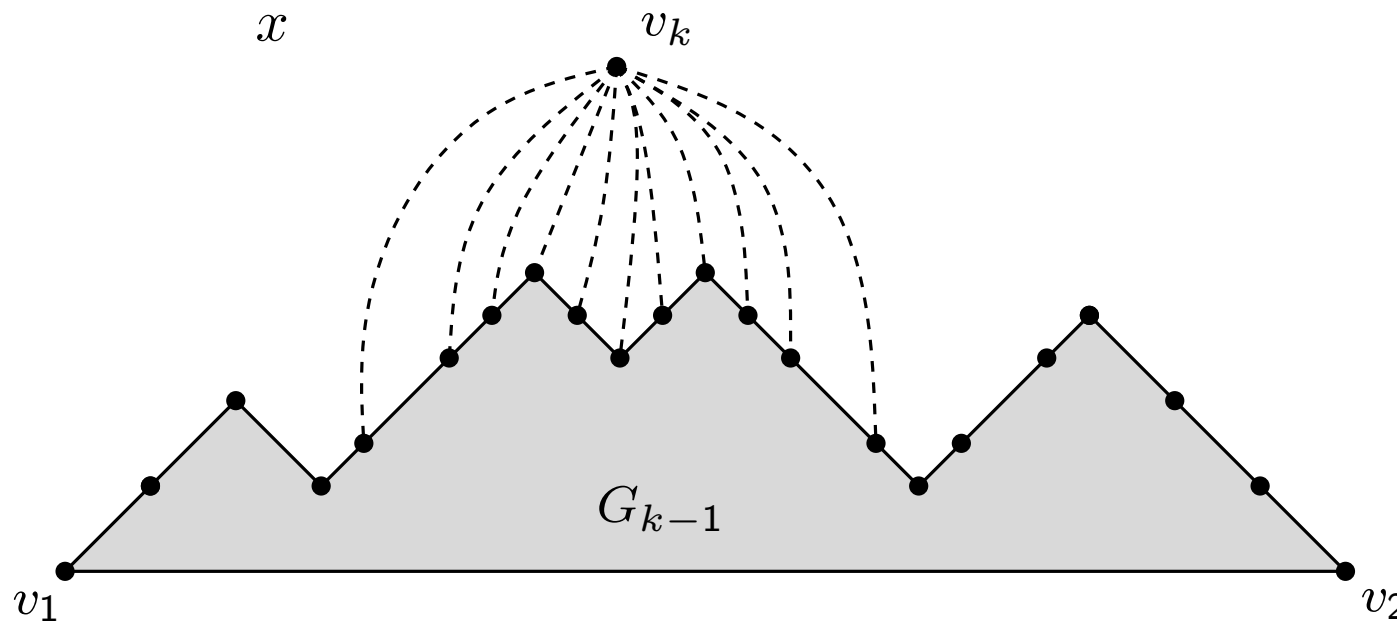
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

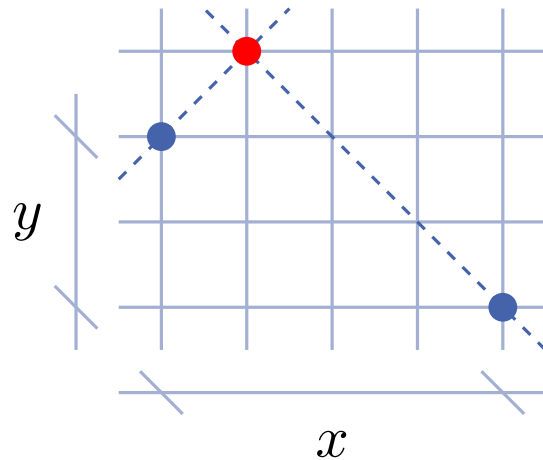
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x -monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 3

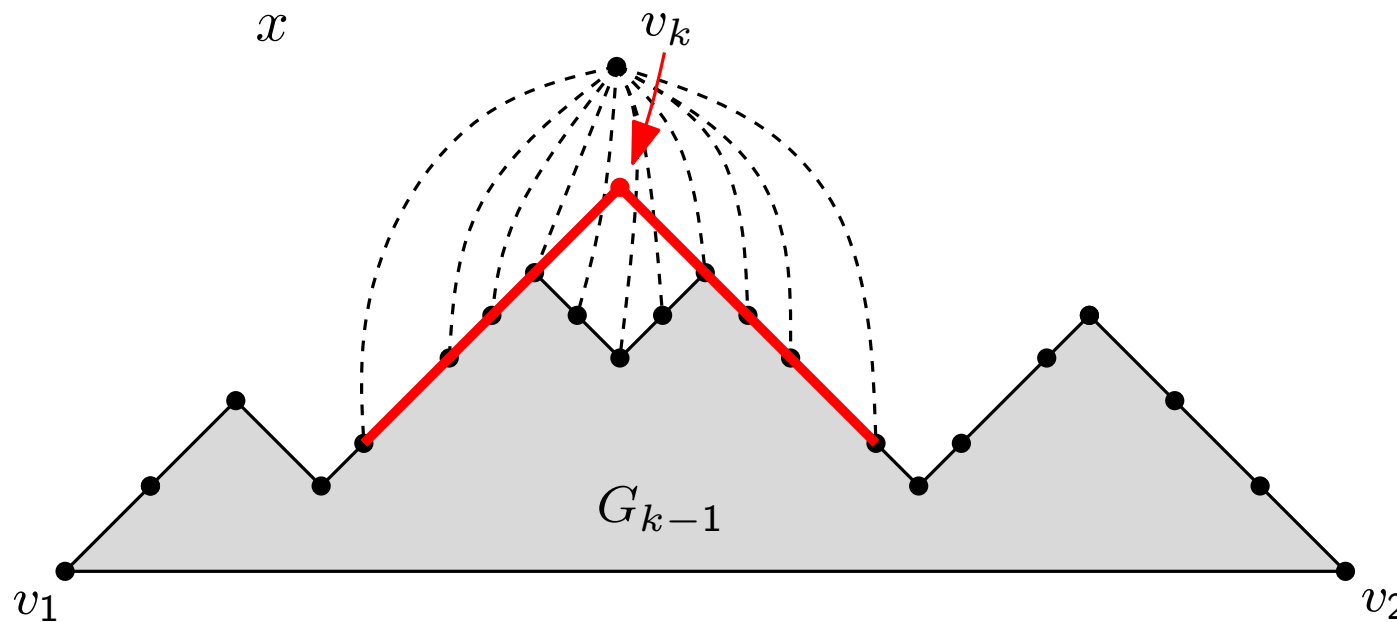
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

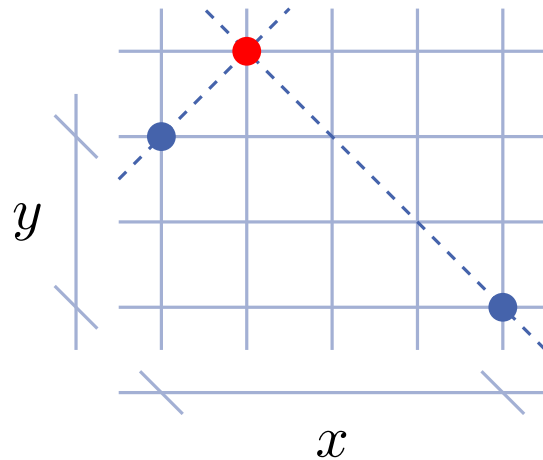
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x -monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 4

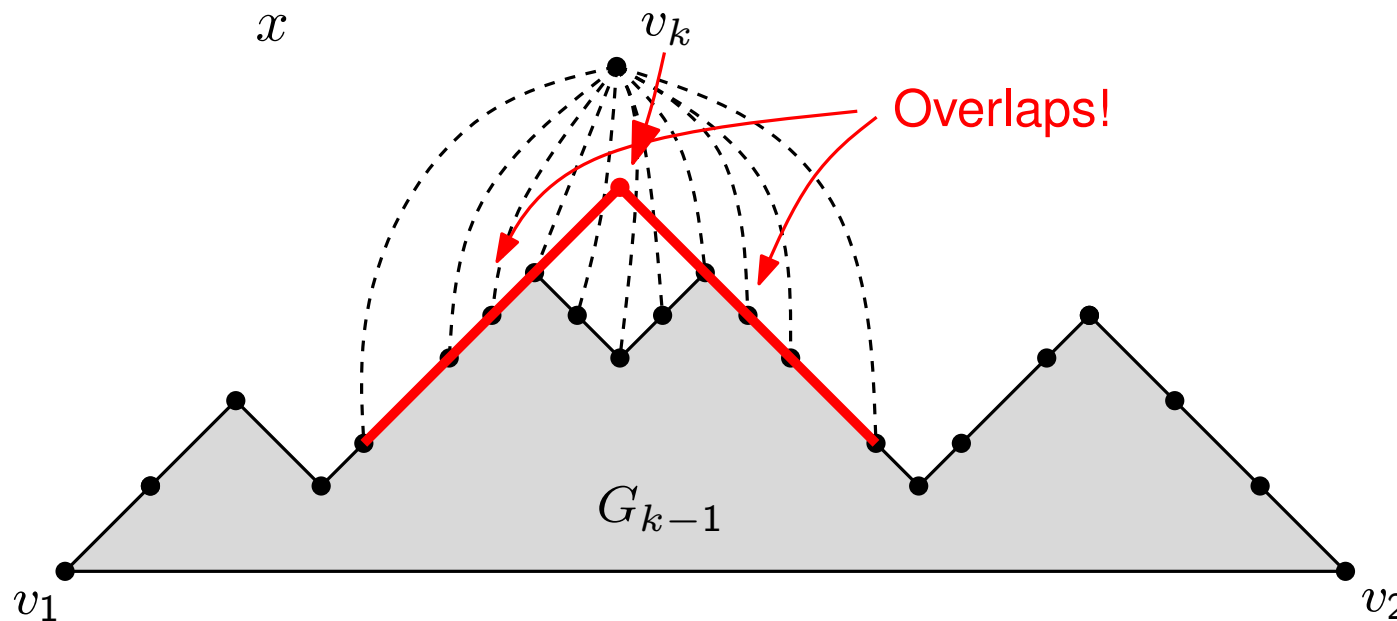
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

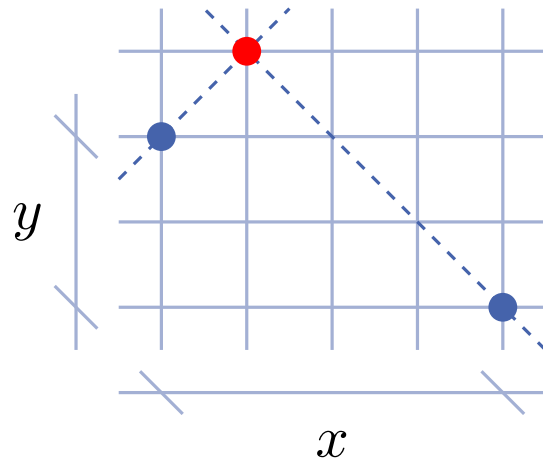
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x -monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 5

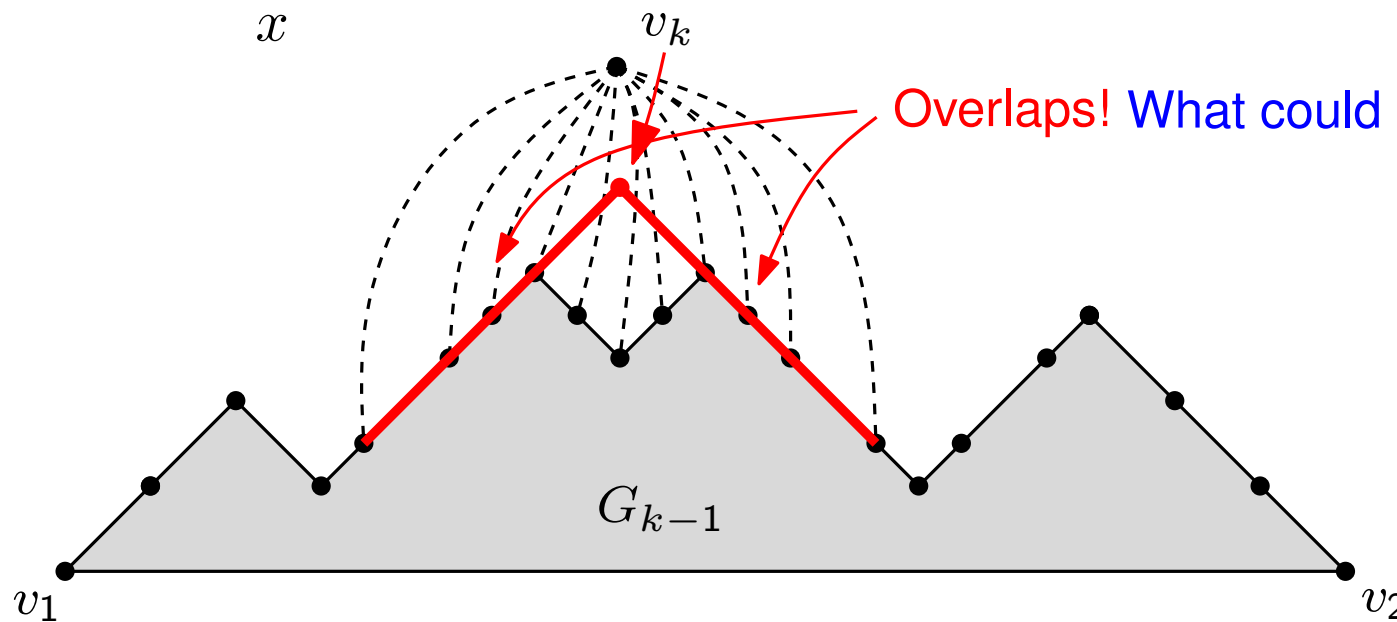
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

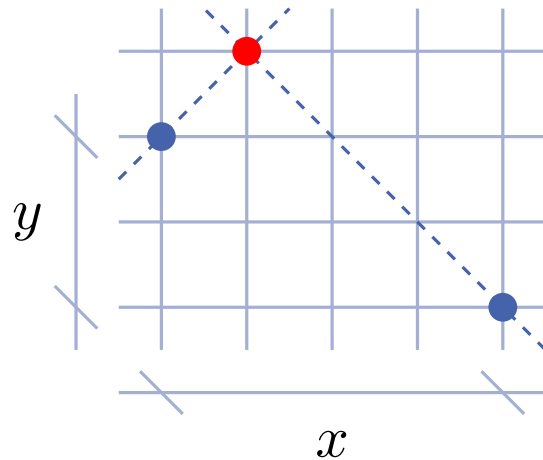
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x -monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 6

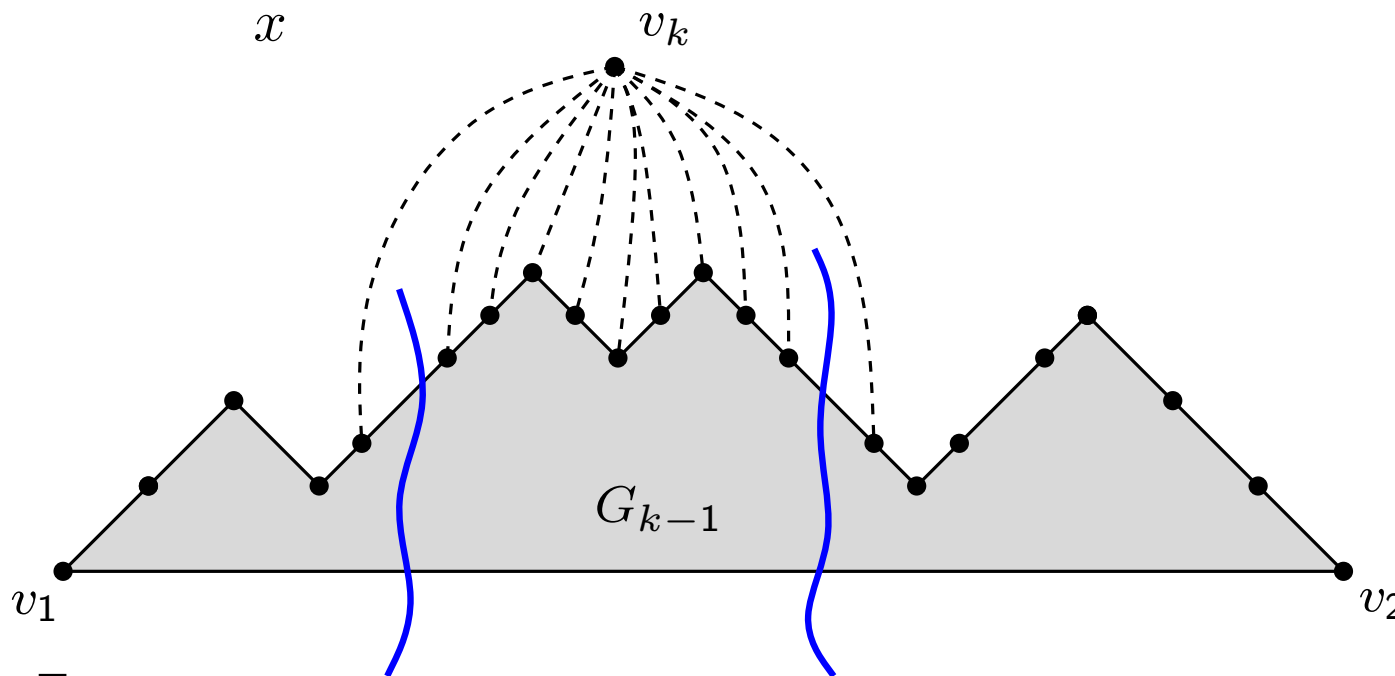
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

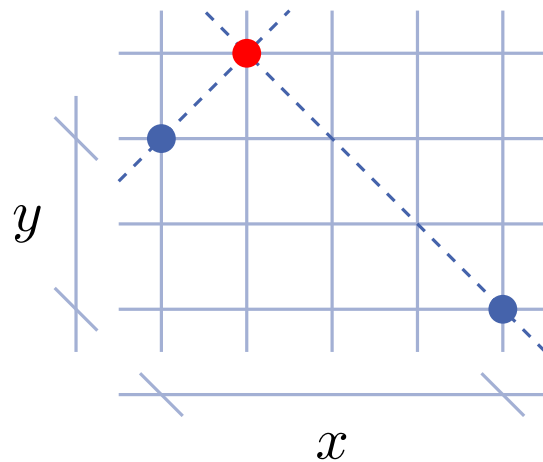
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x -monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 7

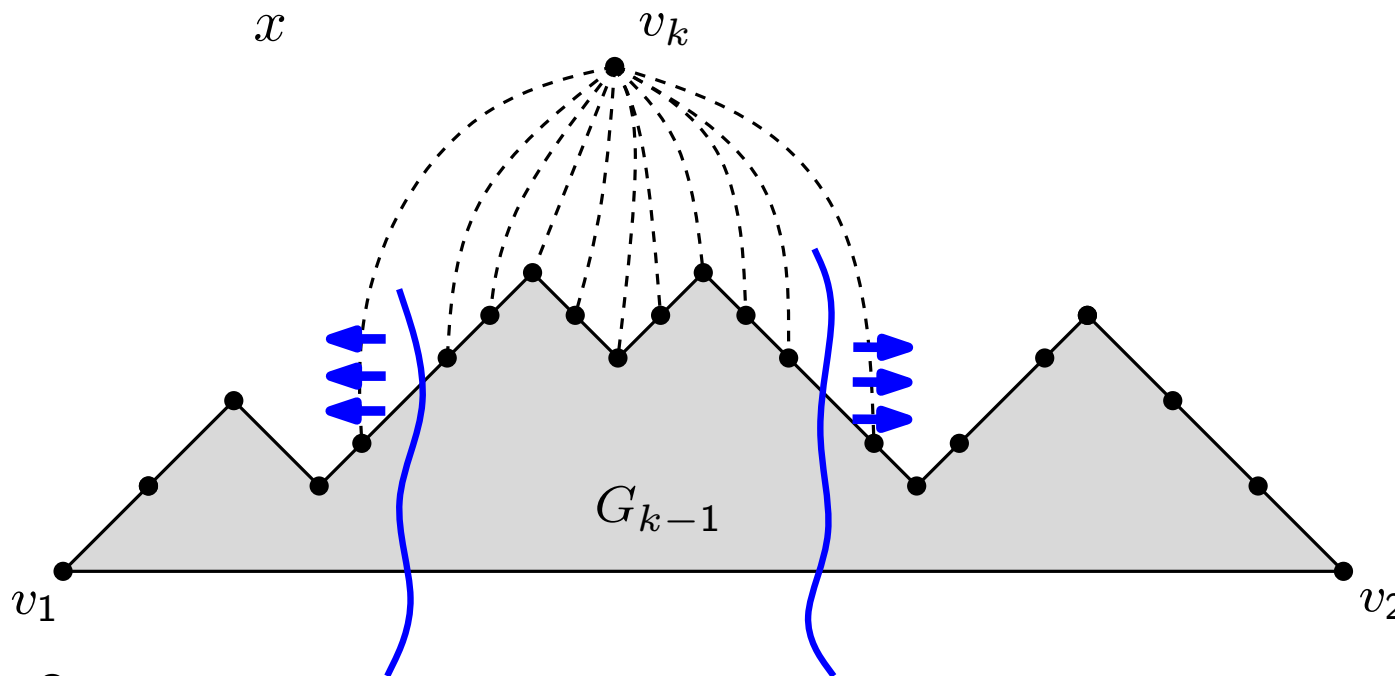
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

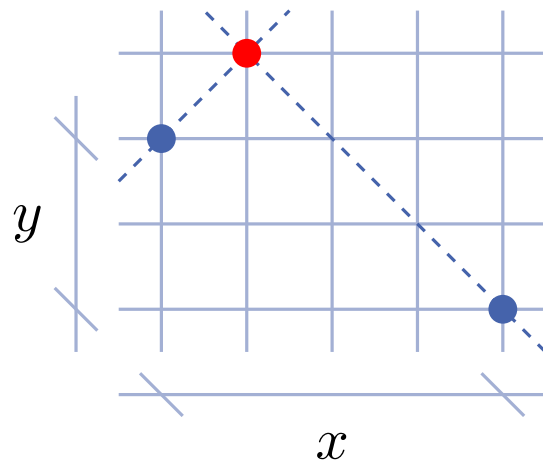
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x -monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 8

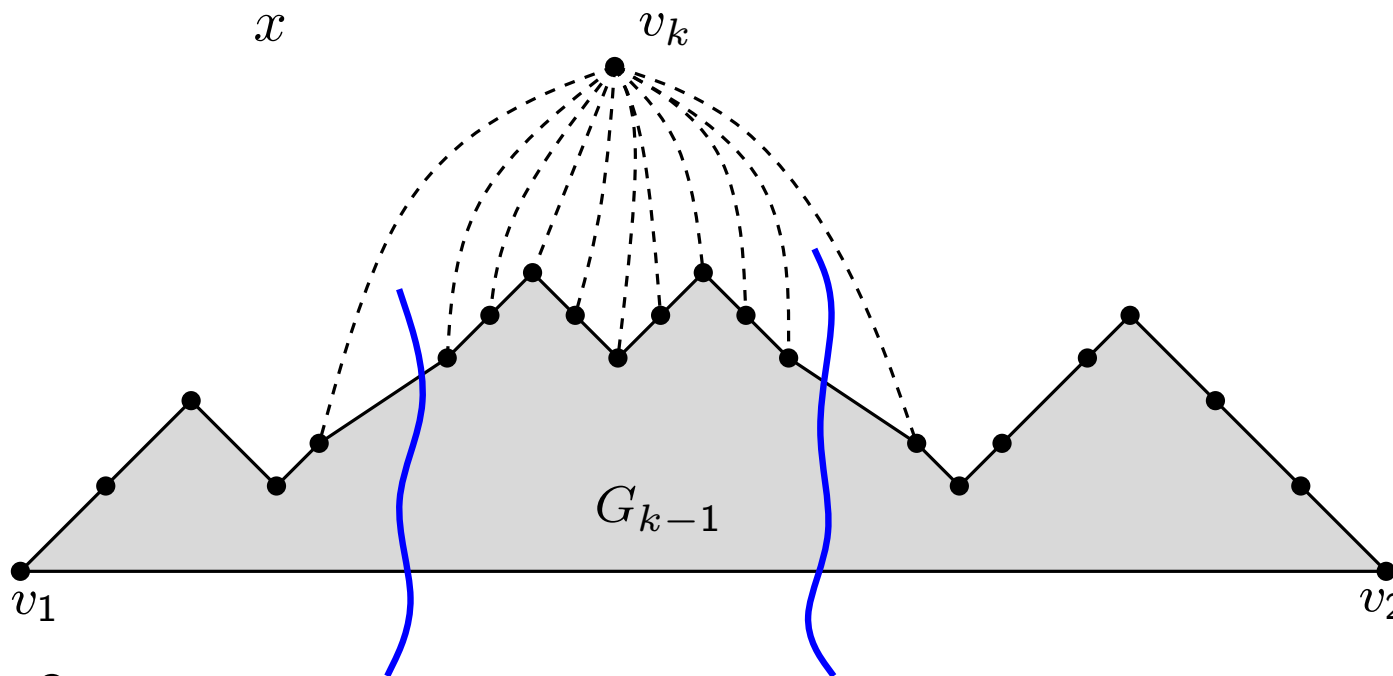
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

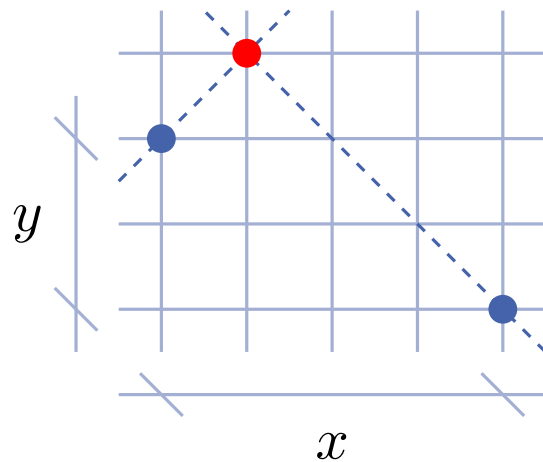
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x -monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 9

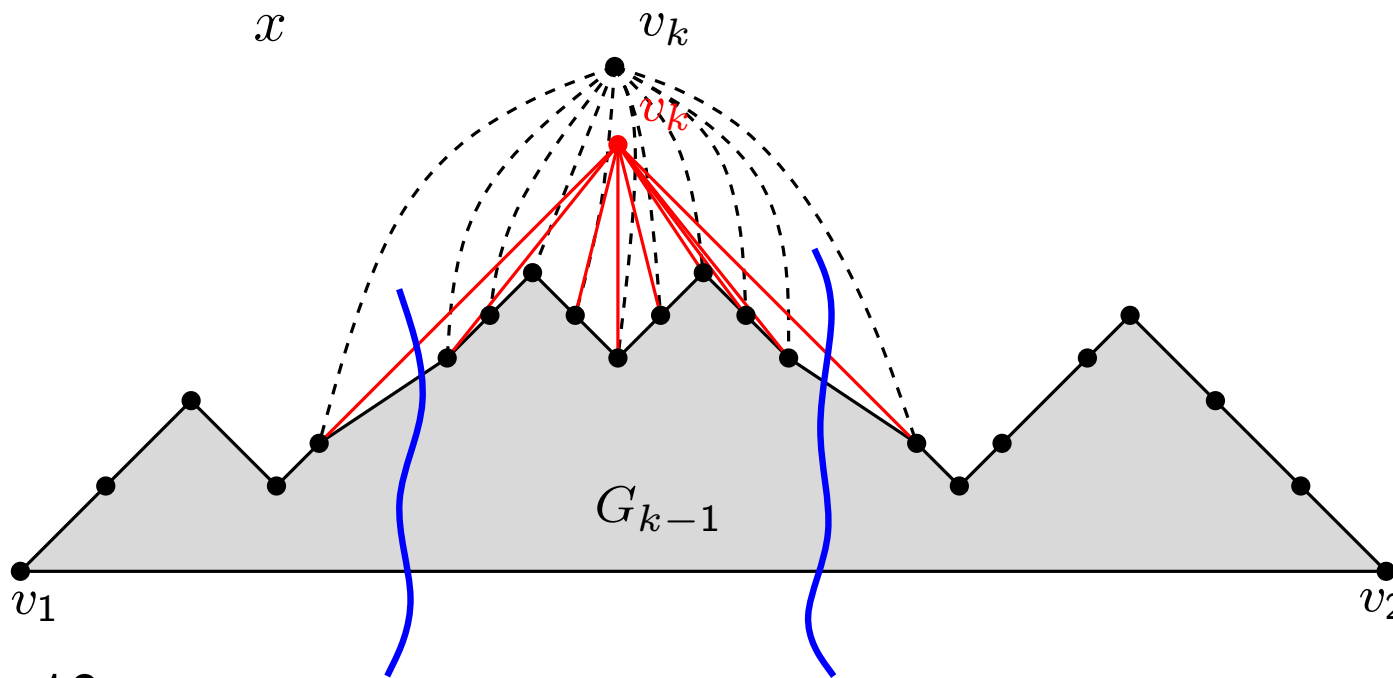
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

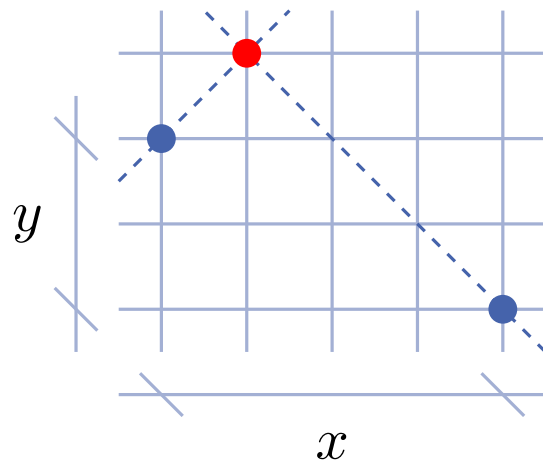
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x -monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 10

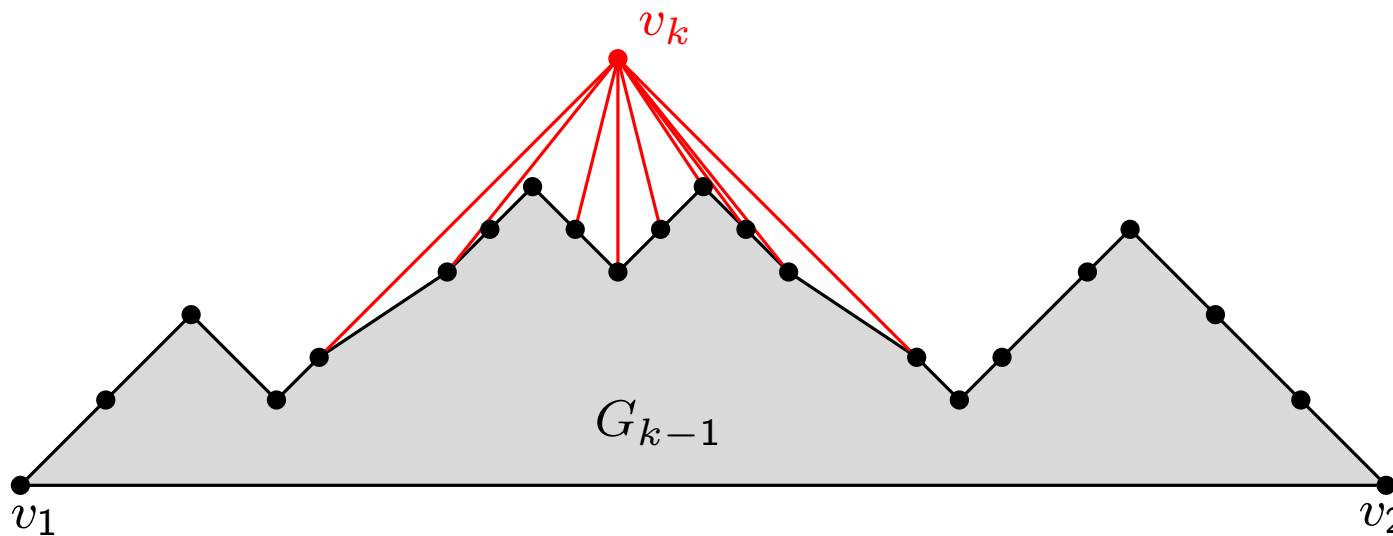
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



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

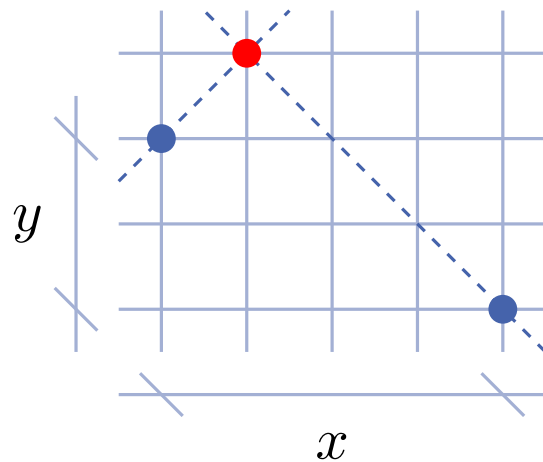
- v_1 is on $(0, 0)$, v_2 is on $(2k - 6, 0)$
- Boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x -monotone
- Each edge of the boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slopes ± 1



15 - 11

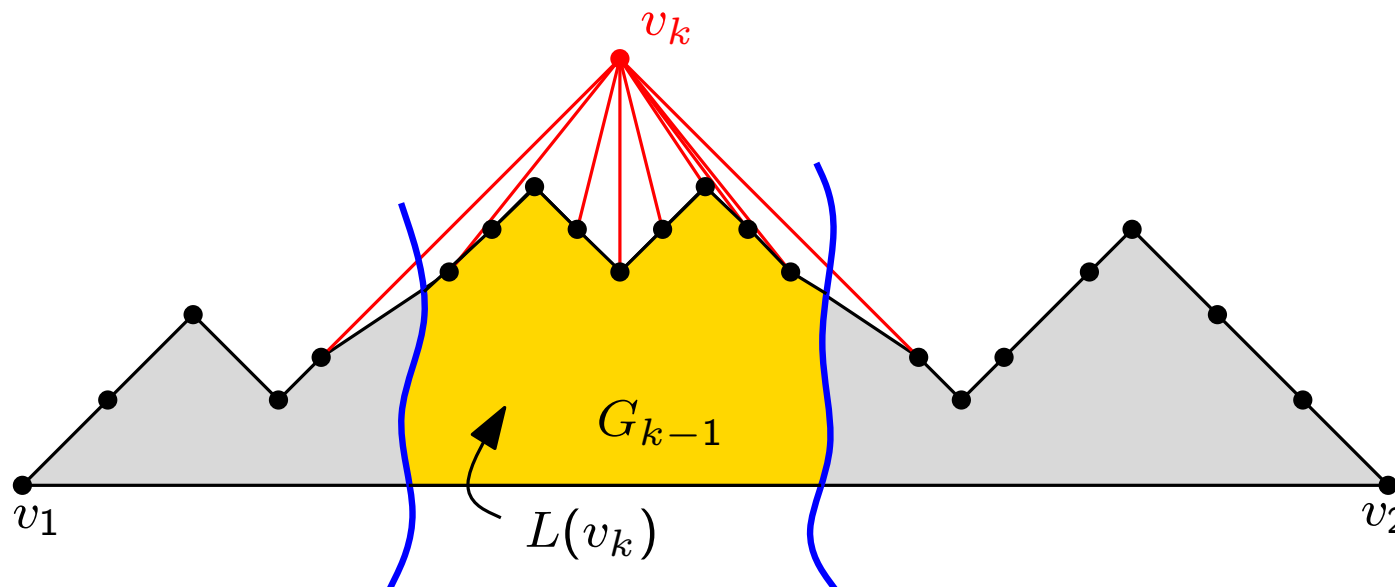
De Fraysseix Pach Pollack (Shift) Algorithm

even Manhattan distance



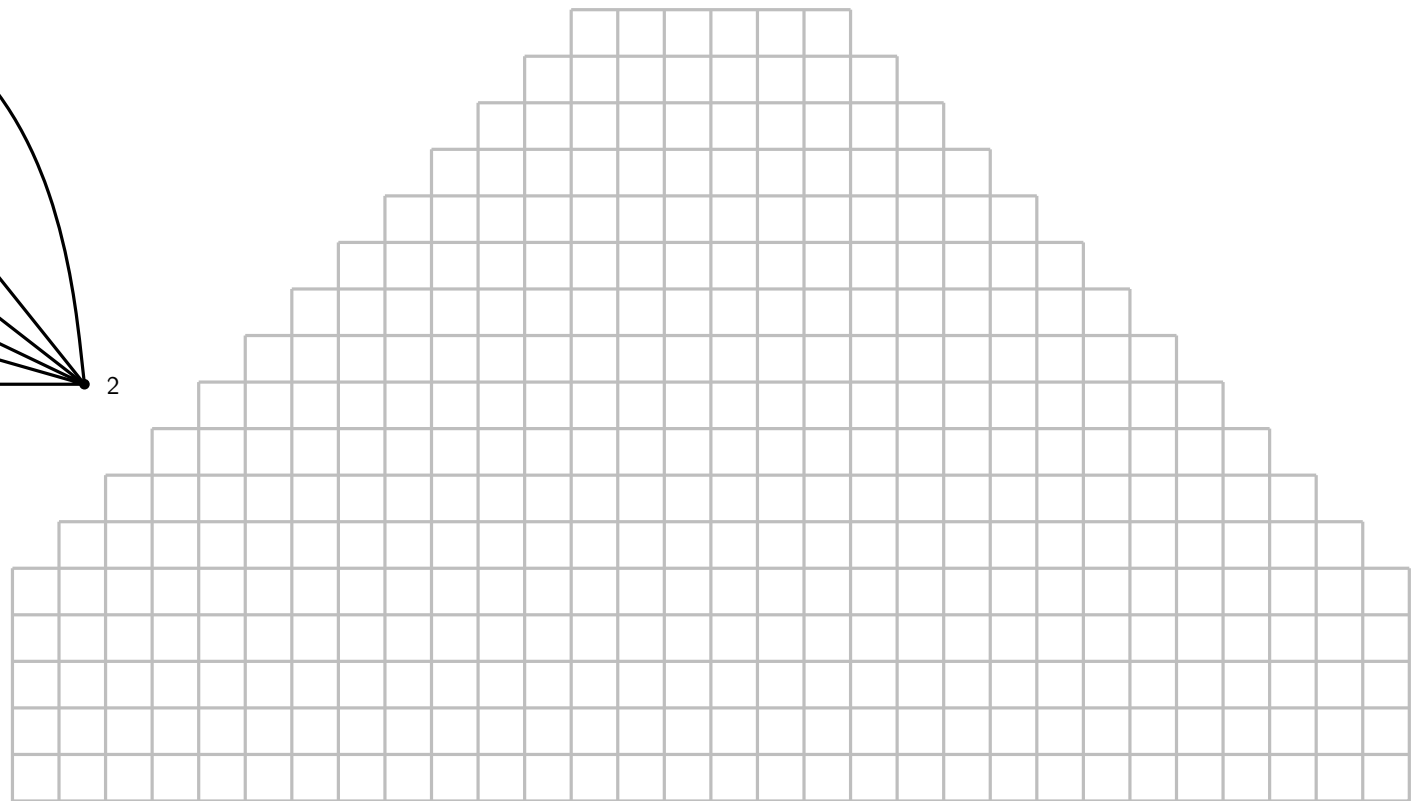
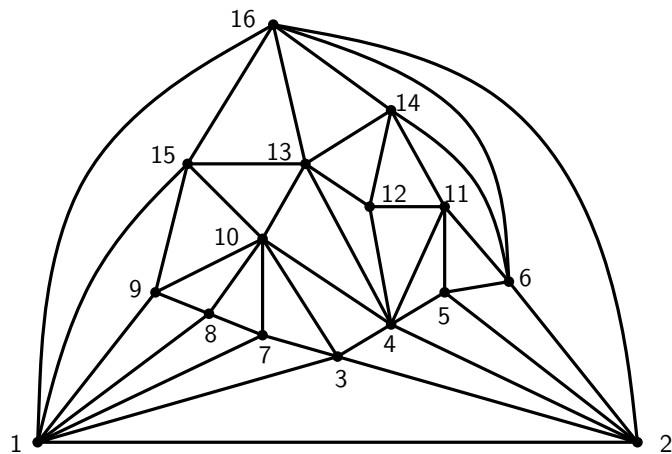
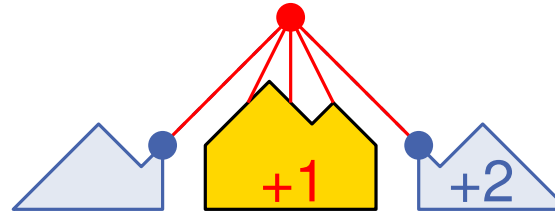
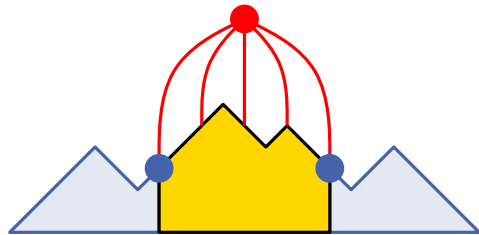
Algorithm invariants: G_{k-1} is drawn such that

- 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



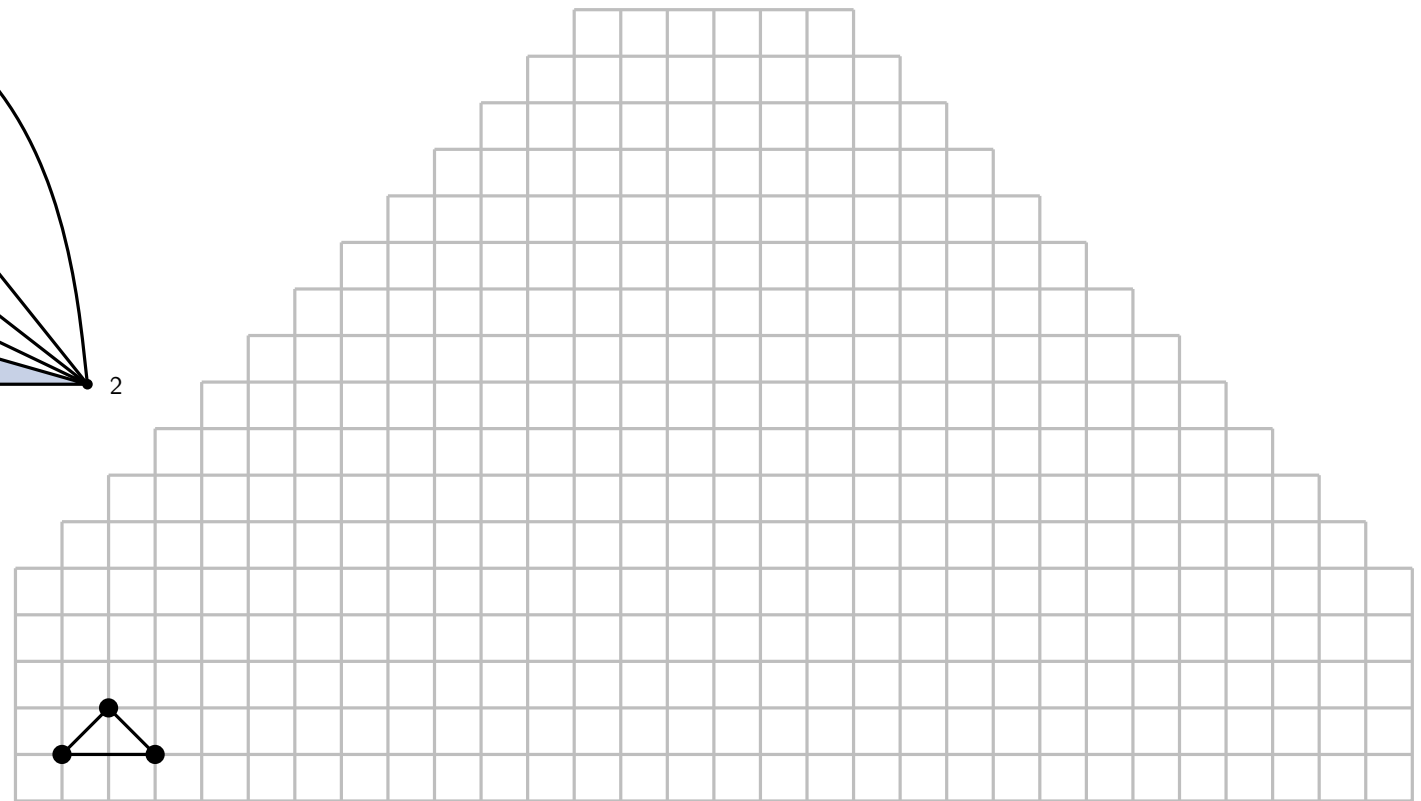
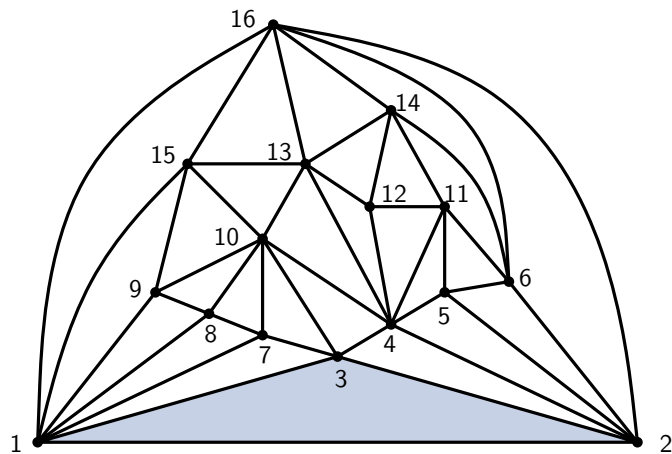
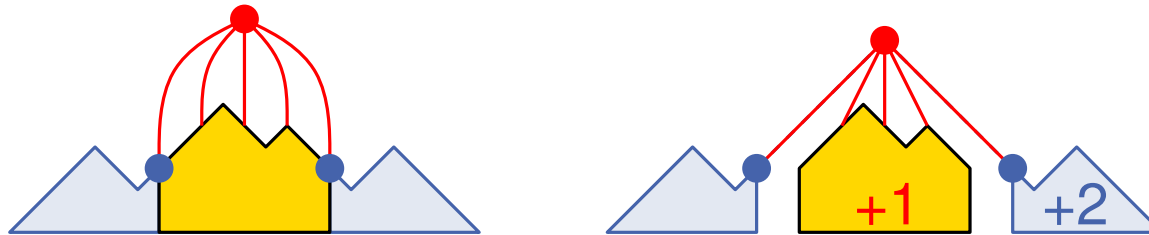
15 - 12

De Fraysseix Pach Pollack (Shift) Algorithm



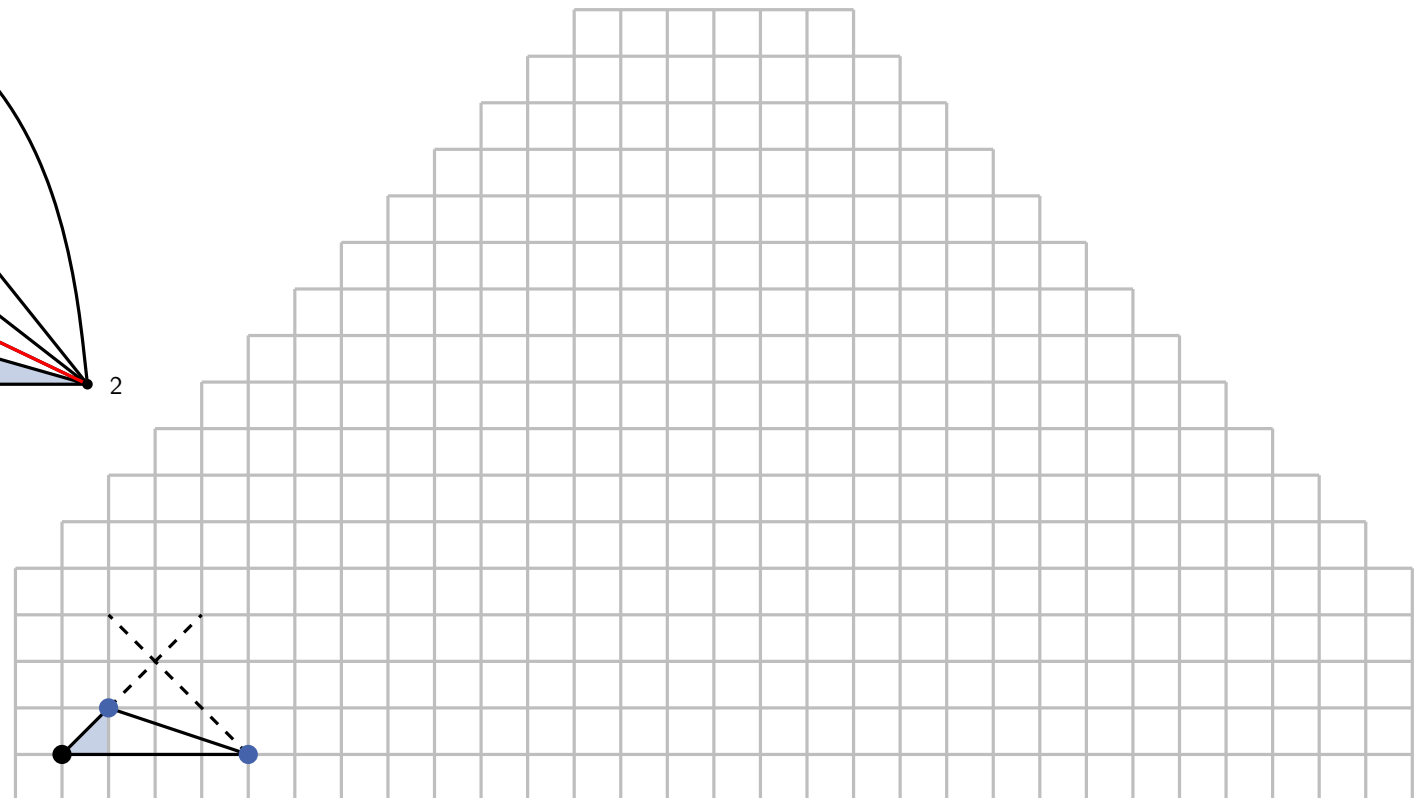
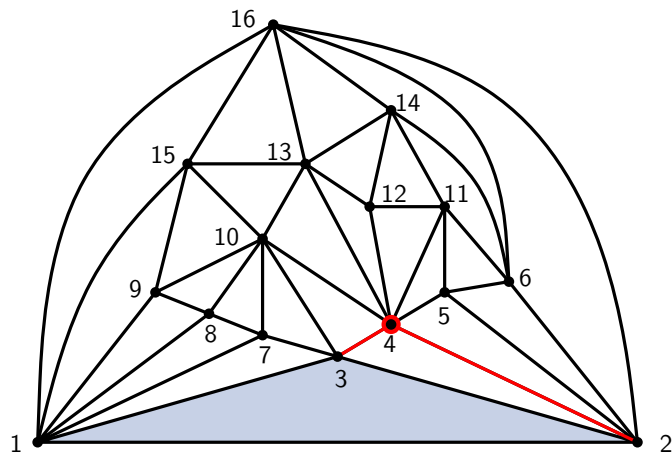
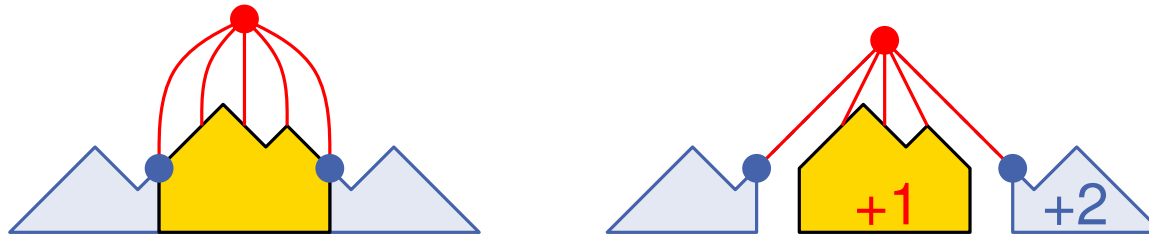
16 - 1

De Fraysseix Pach Pollack (Shift) Algorithm



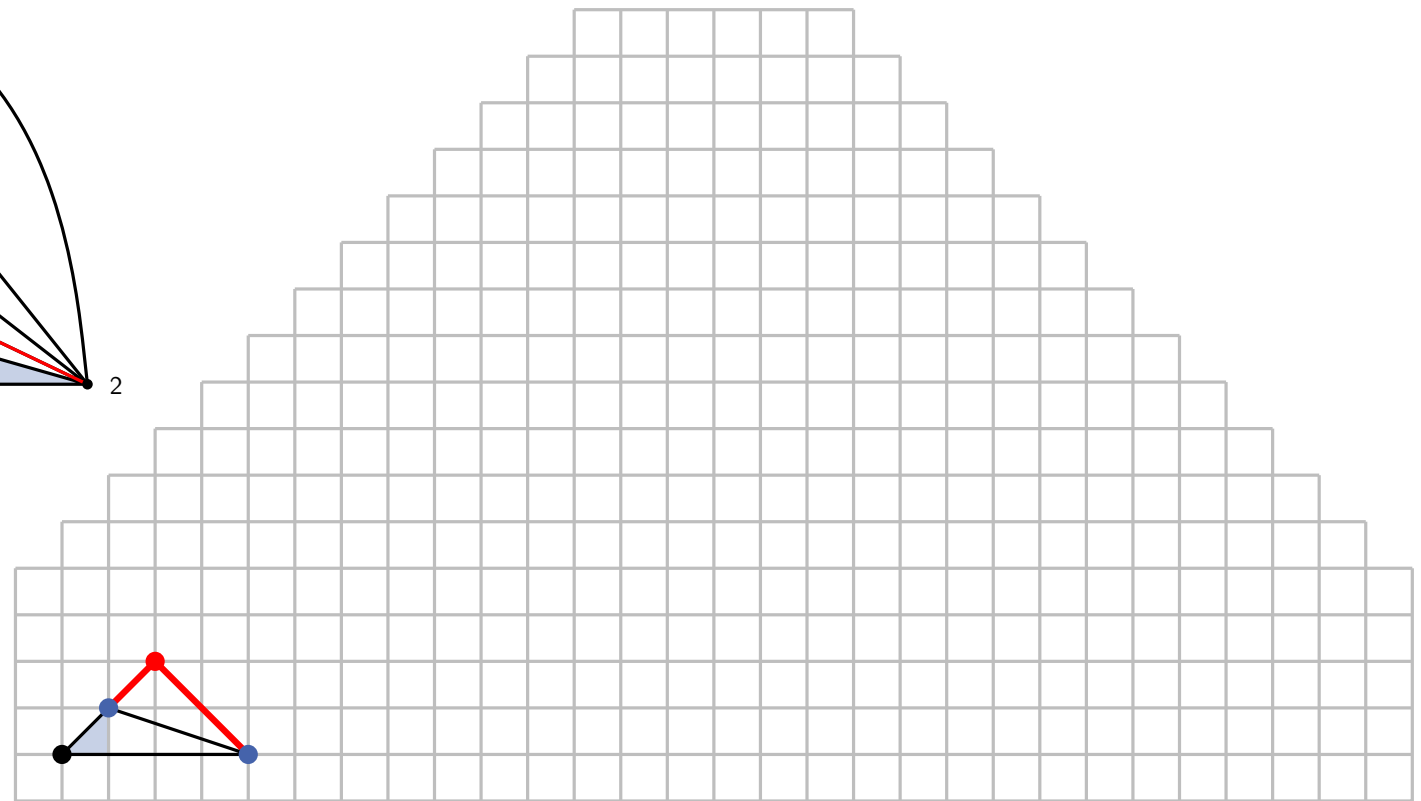
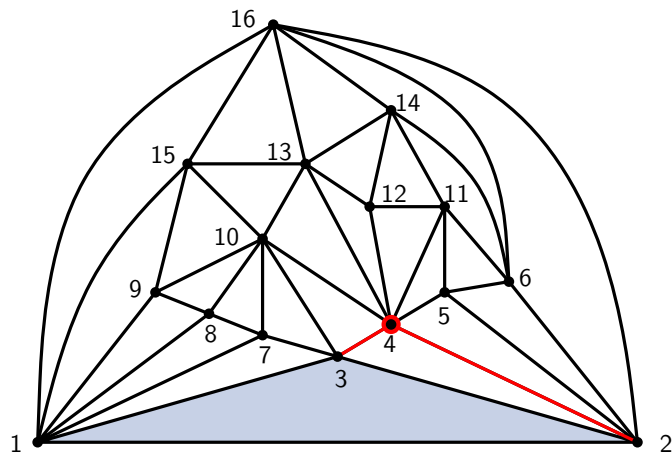
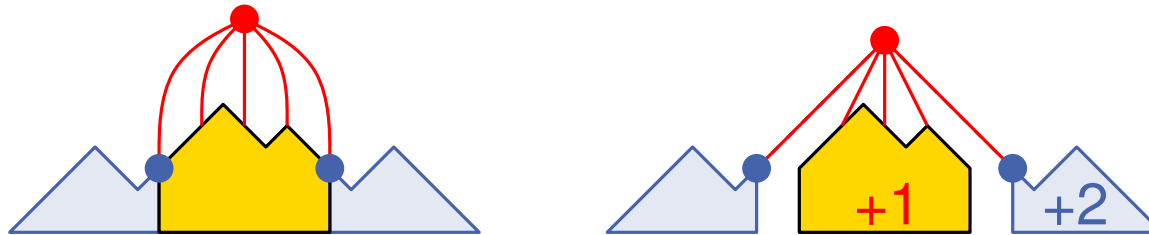
16 - 2

De Fraysseix Pach Pollack (Shift) Algorithm



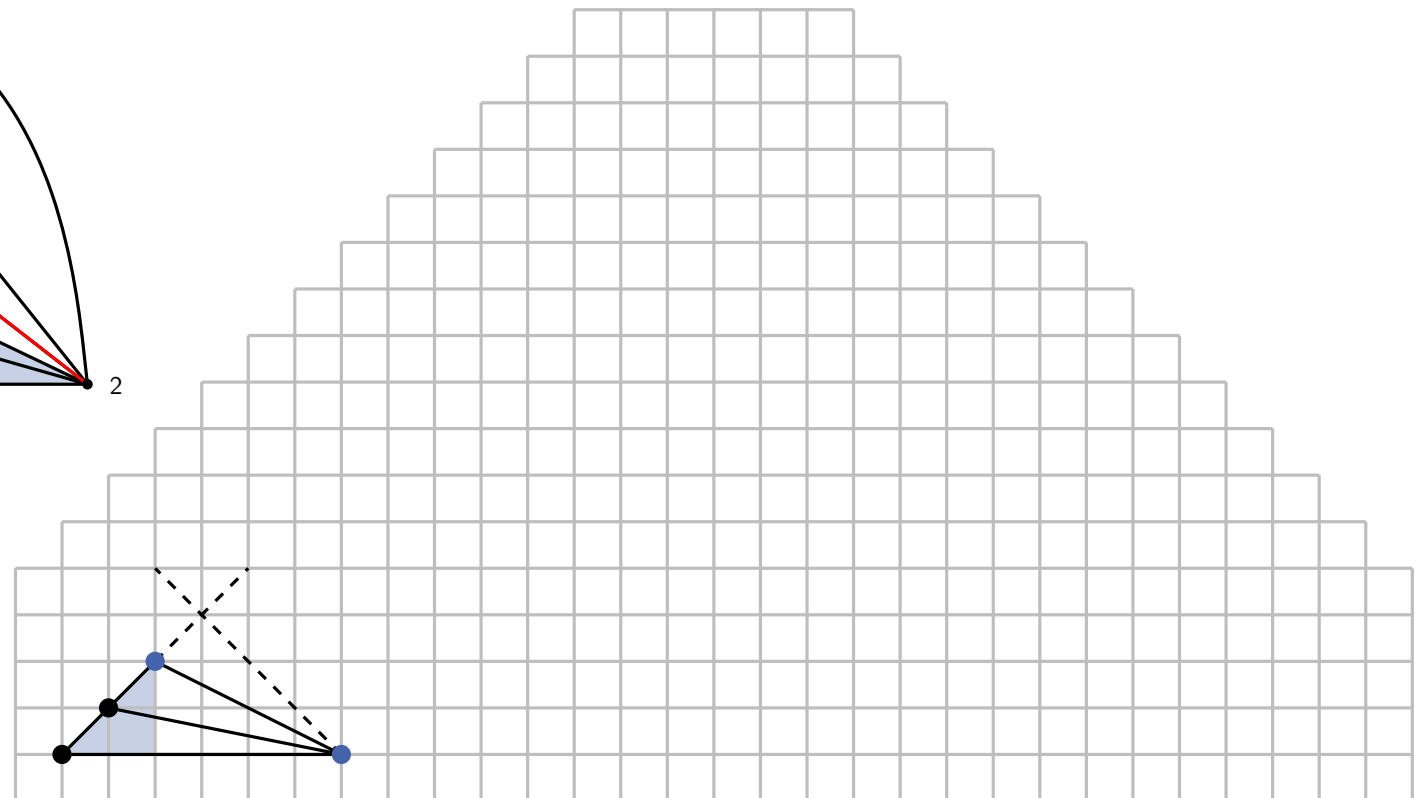
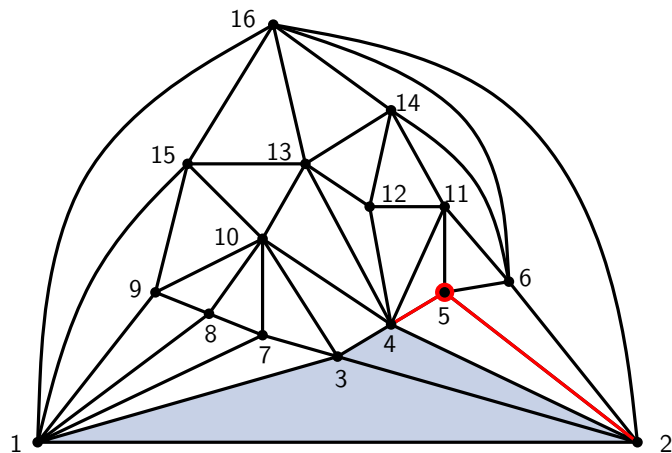
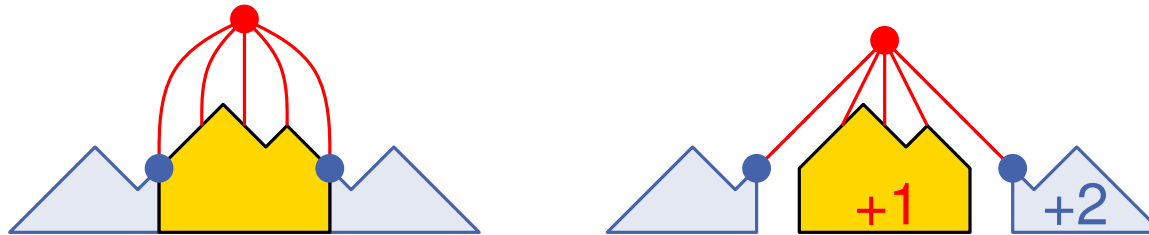
16 - 3

De Fraysseix Pach Pollack (Shift) Algorithm



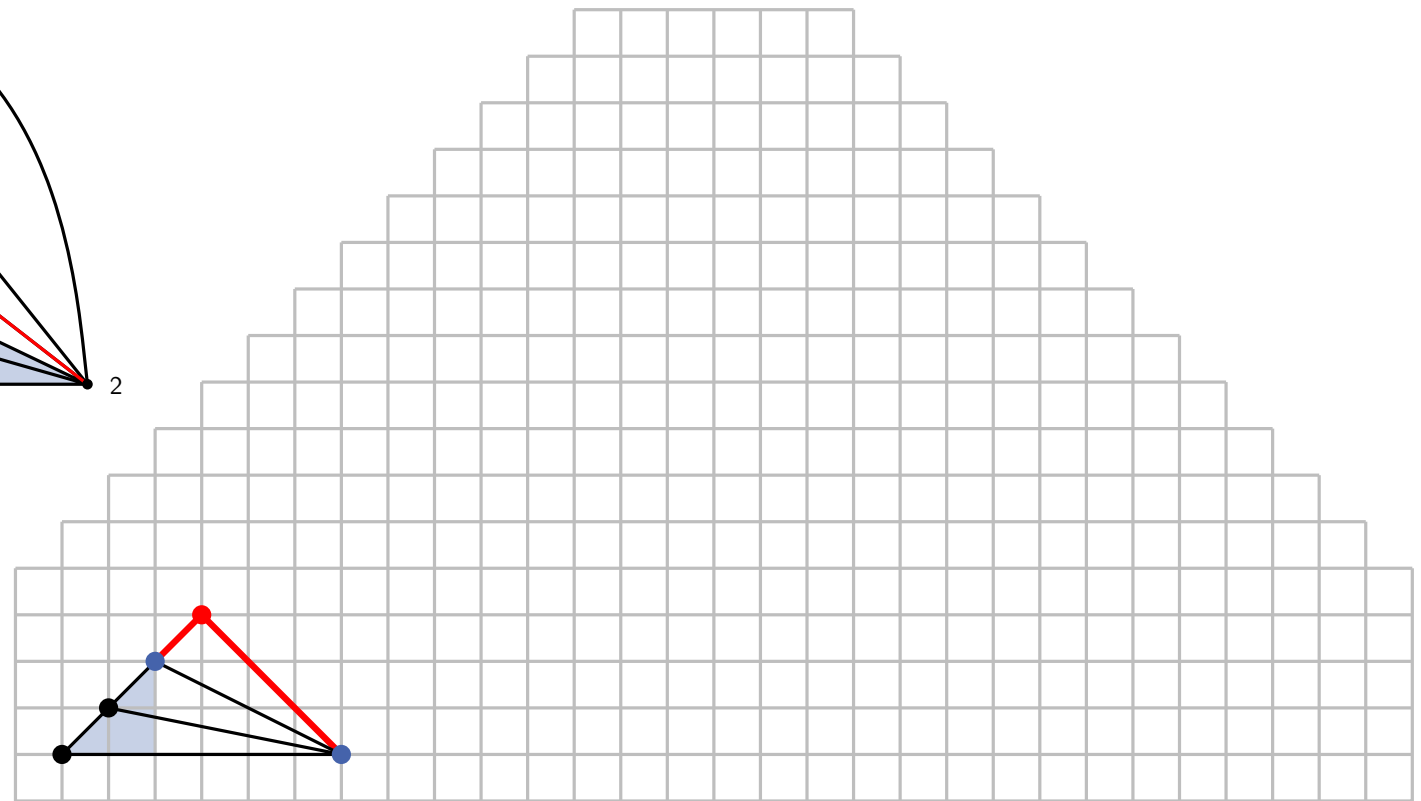
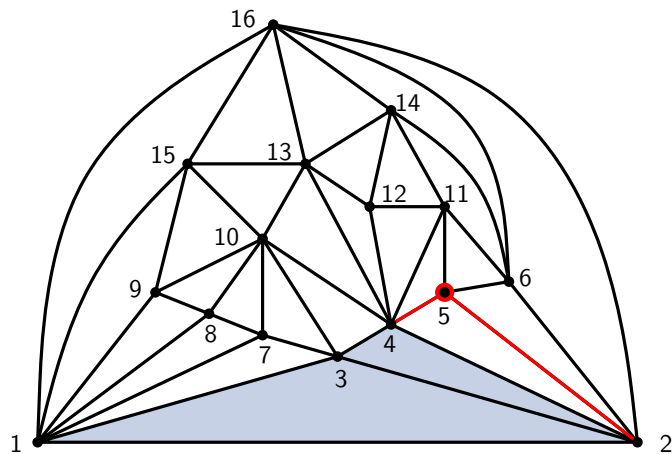
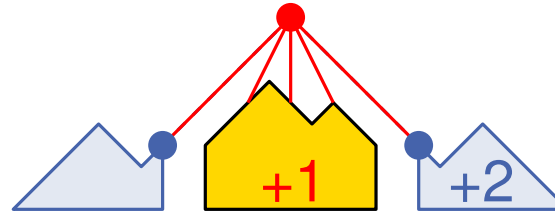
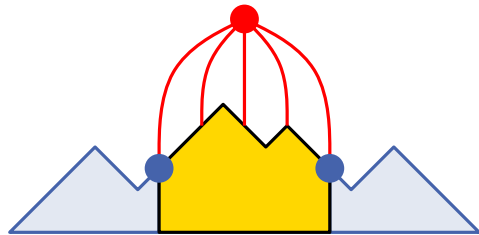
16 - 4

De Fraysseix Pach Pollack (Shift) Algorithm



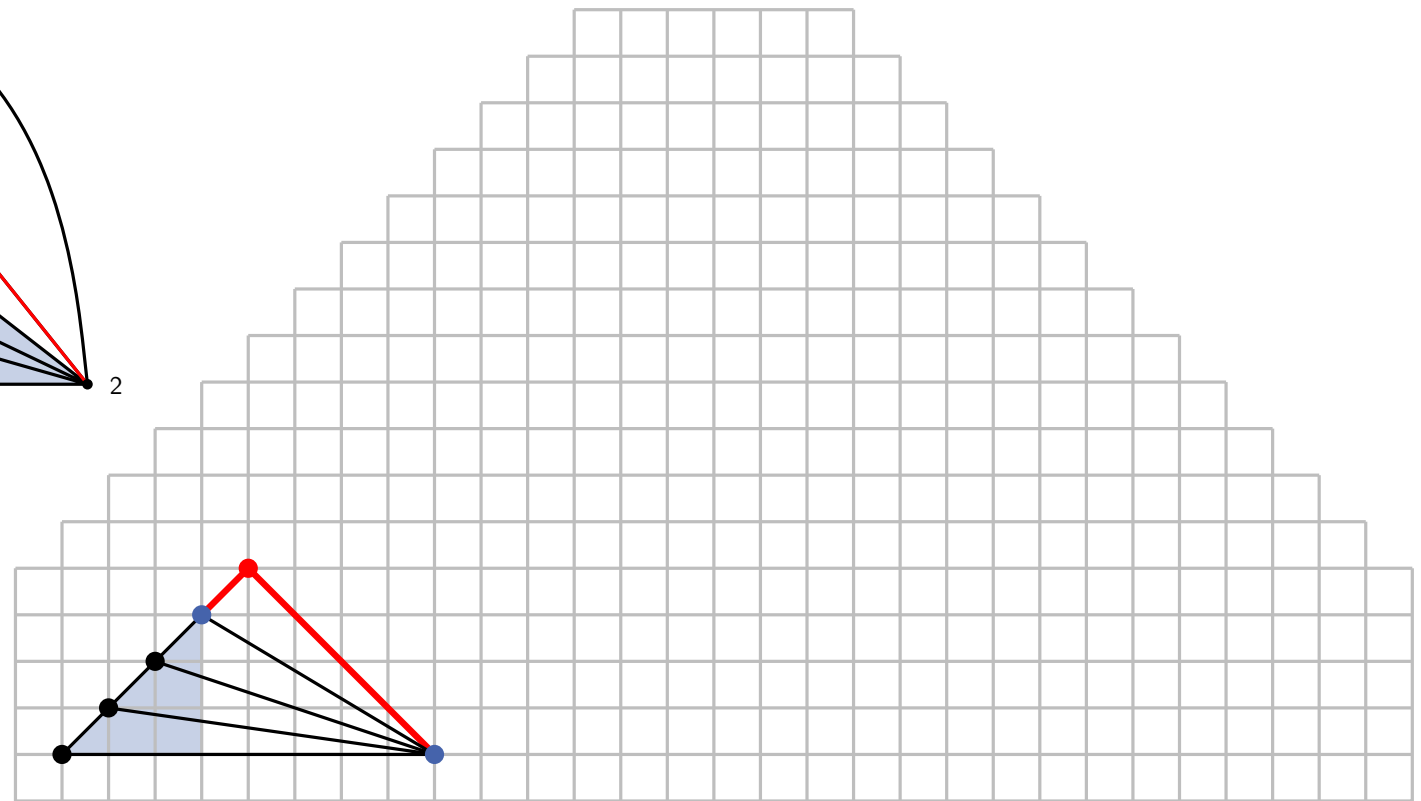
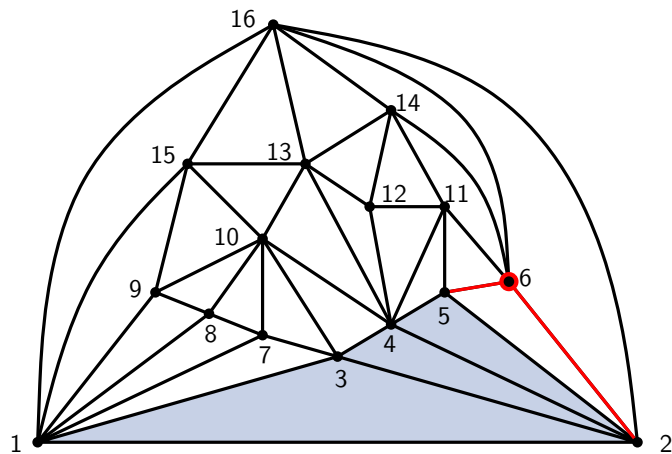
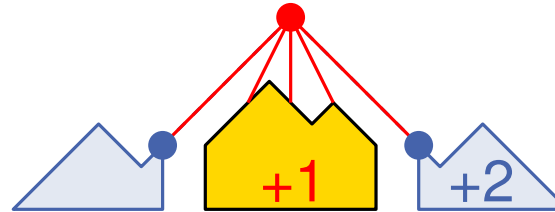
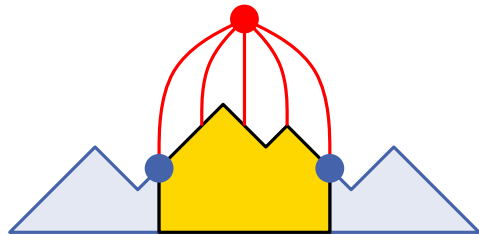
16 - 5

De Fraysseix Pach Pollack (Shift) Algorithm



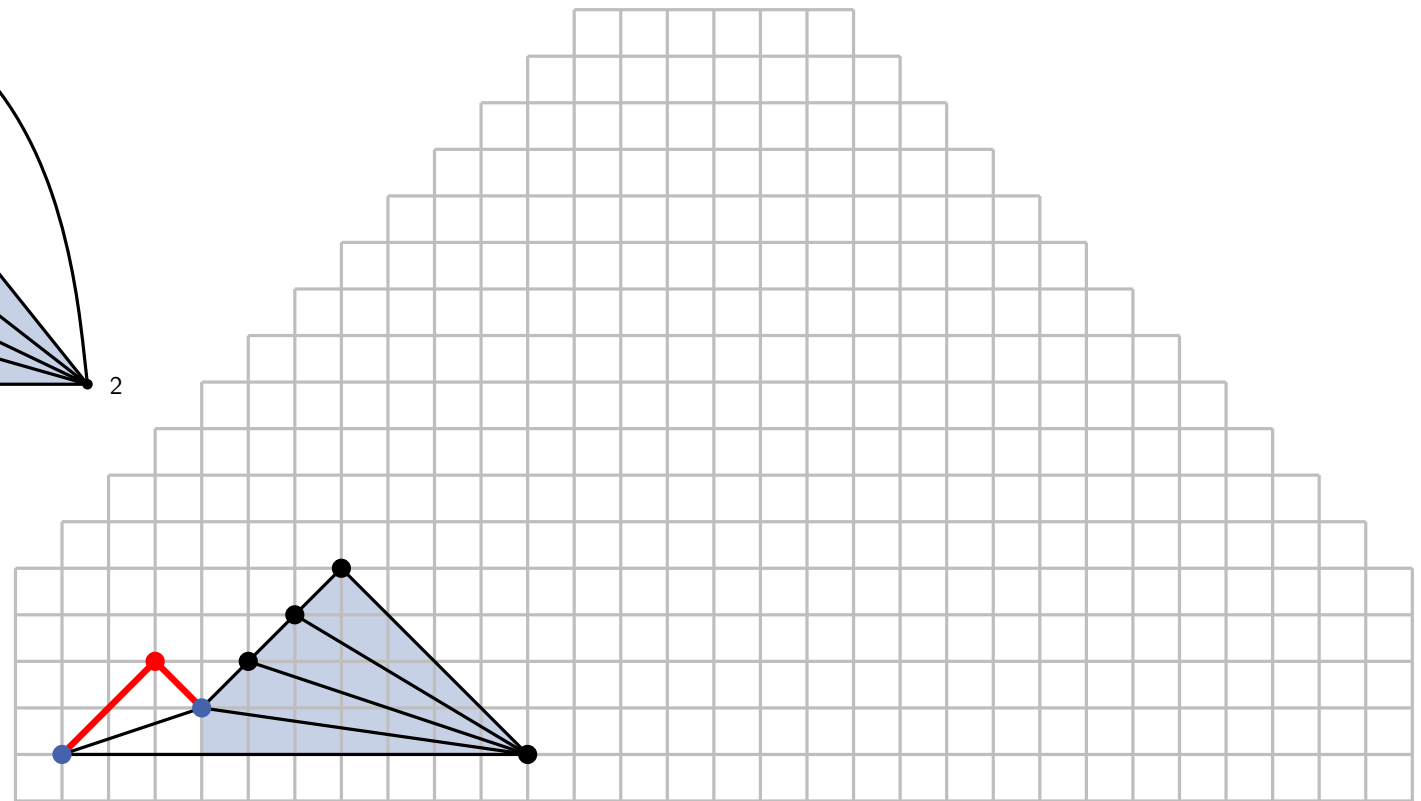
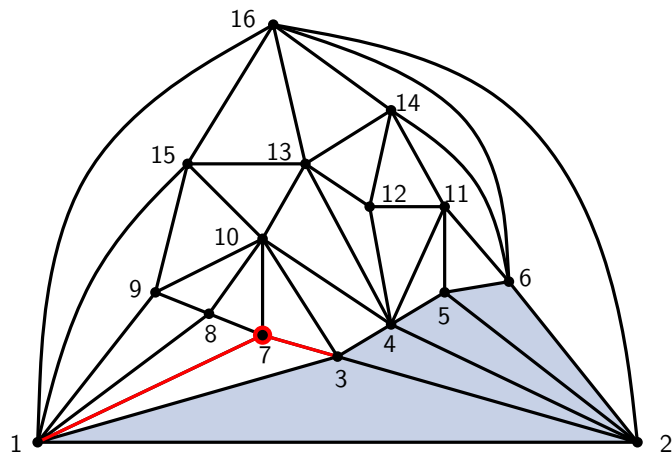
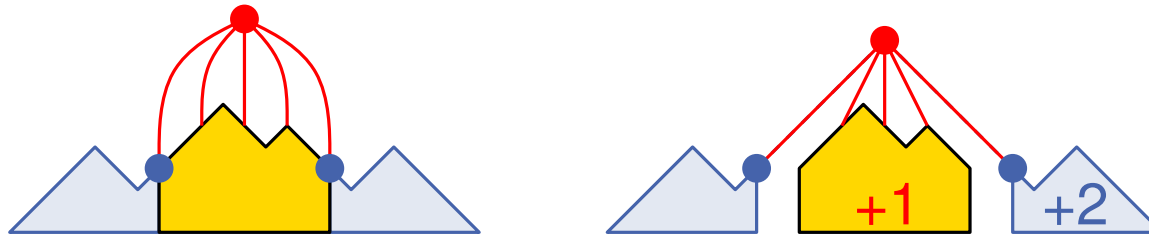
16 - 6

De Fraysseix Pach Pollack (Shift) Algorithm



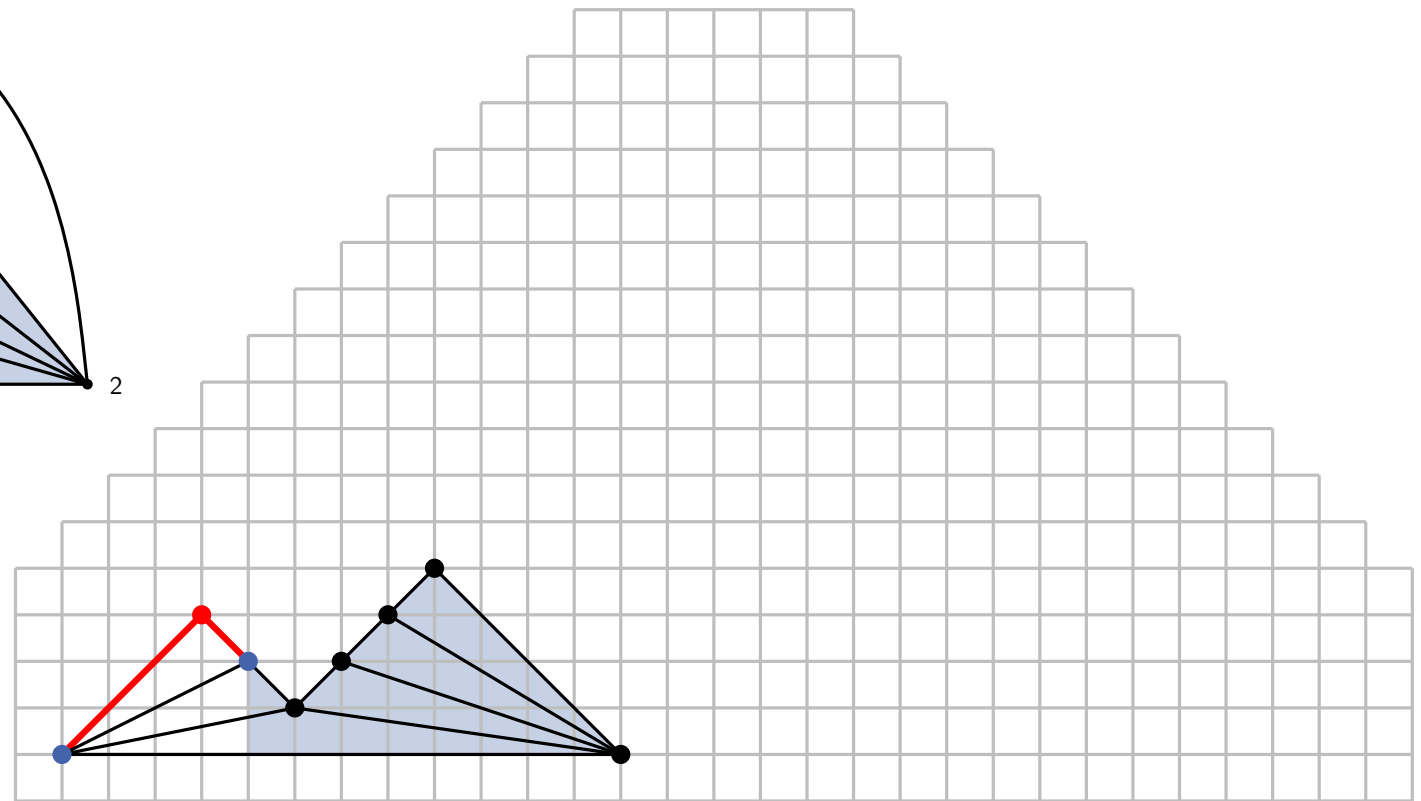
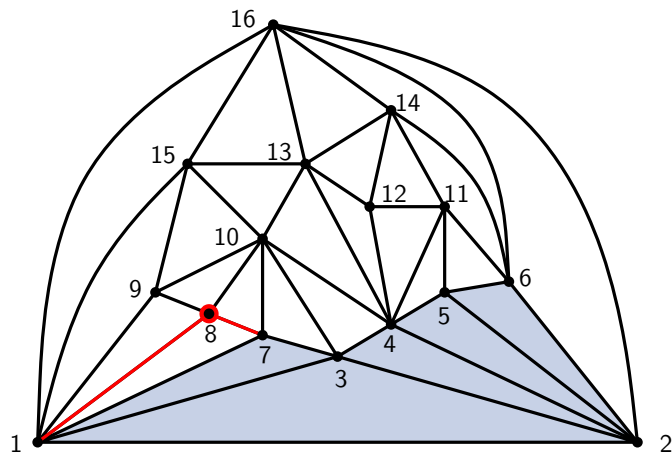
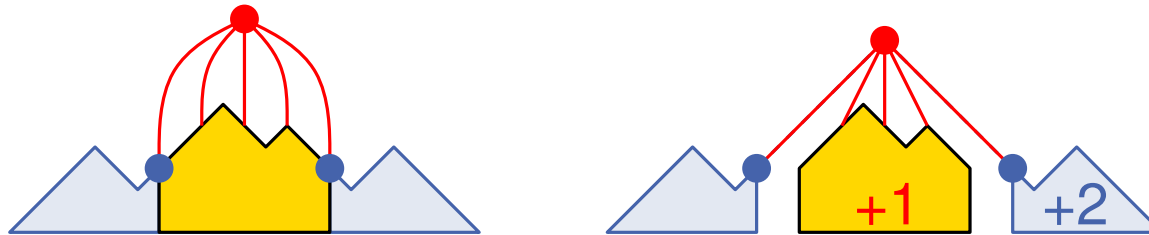
16 - 7

De Fraysseix Pach Pollack (Shift) Algorithm



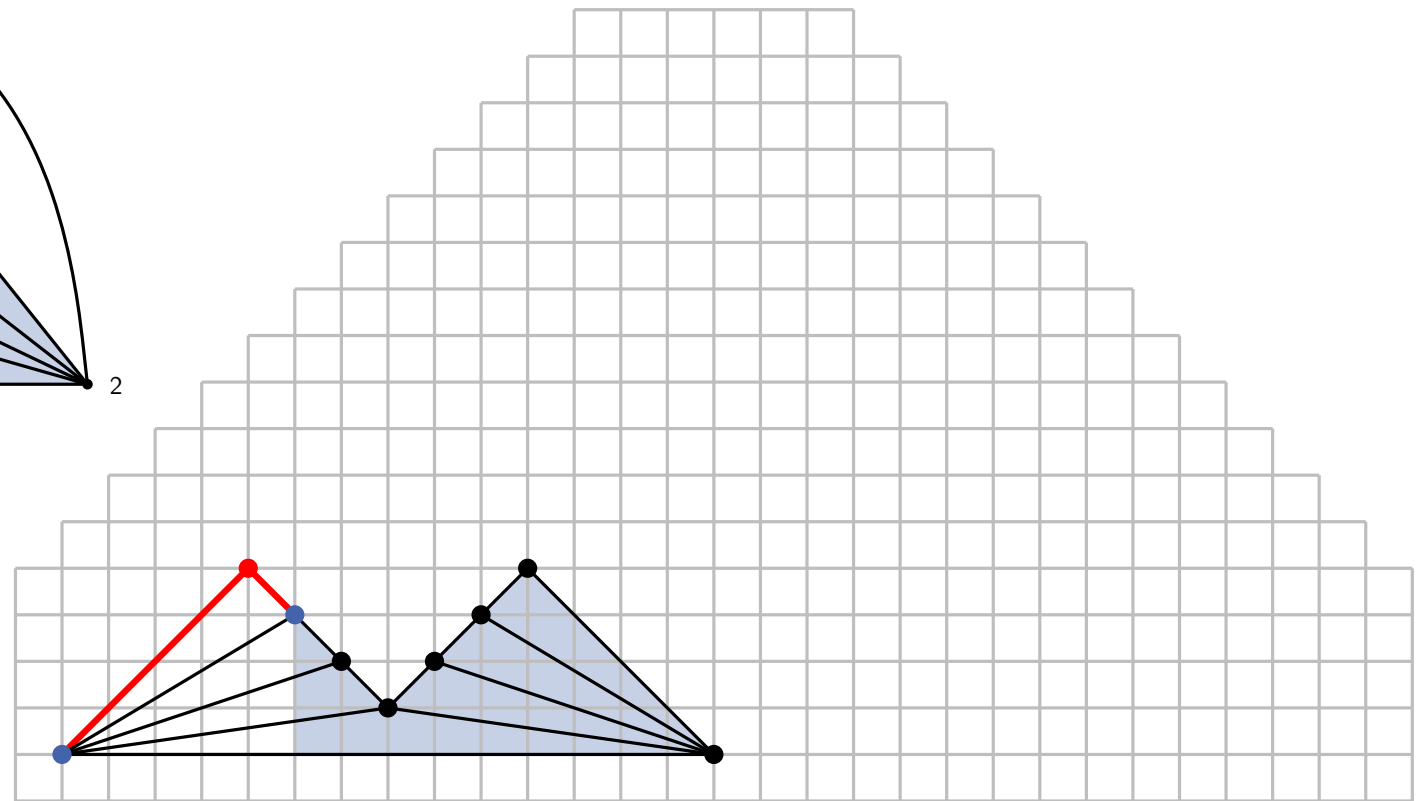
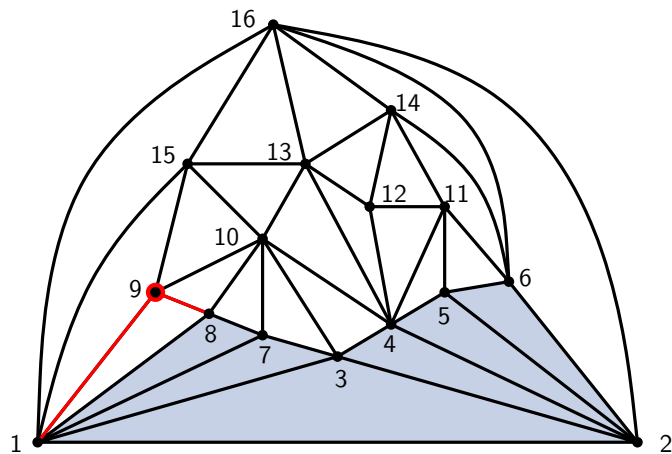
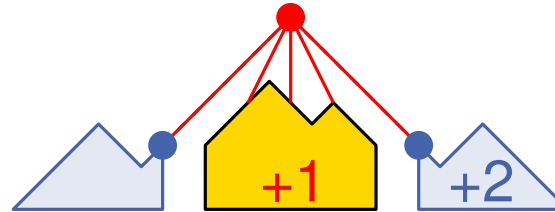
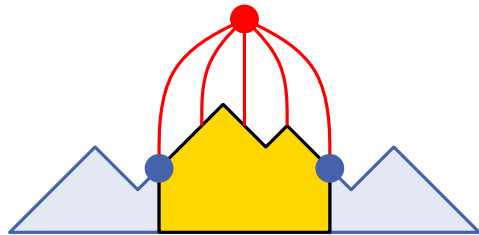
16 - 8

De Fraysseix Pach Pollack (Shift) Algorithm



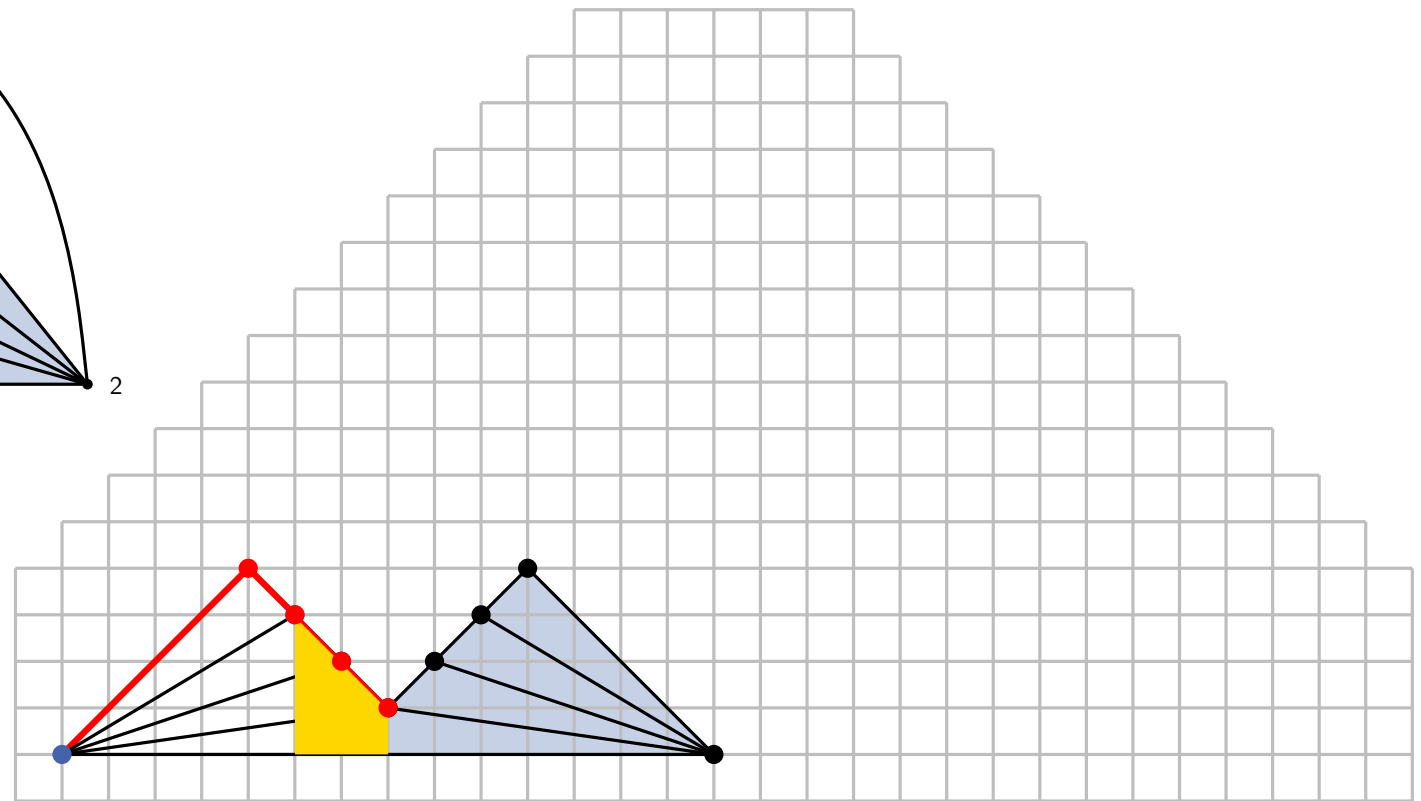
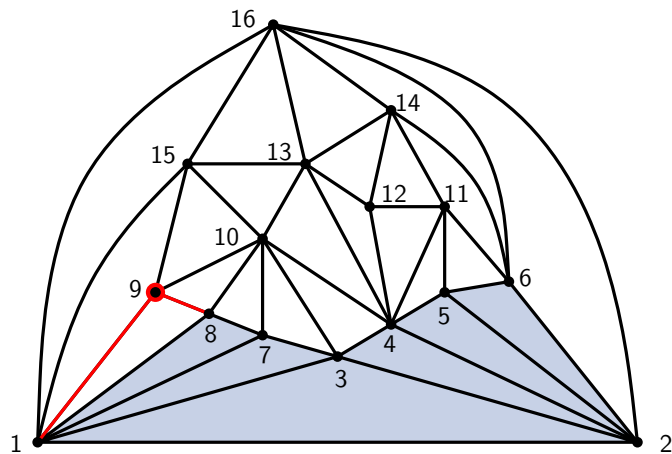
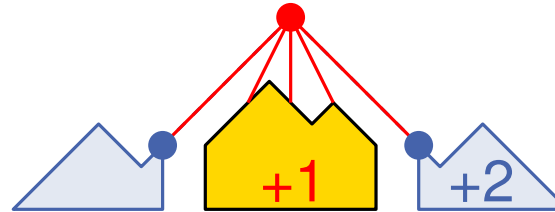
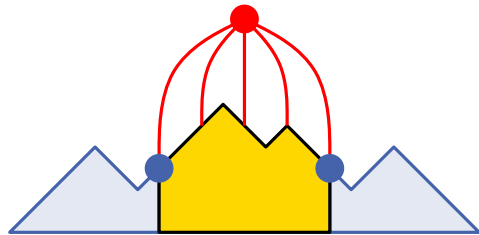
16 - 9

De Fraysseix Pach Pollack (Shift) Algorithm



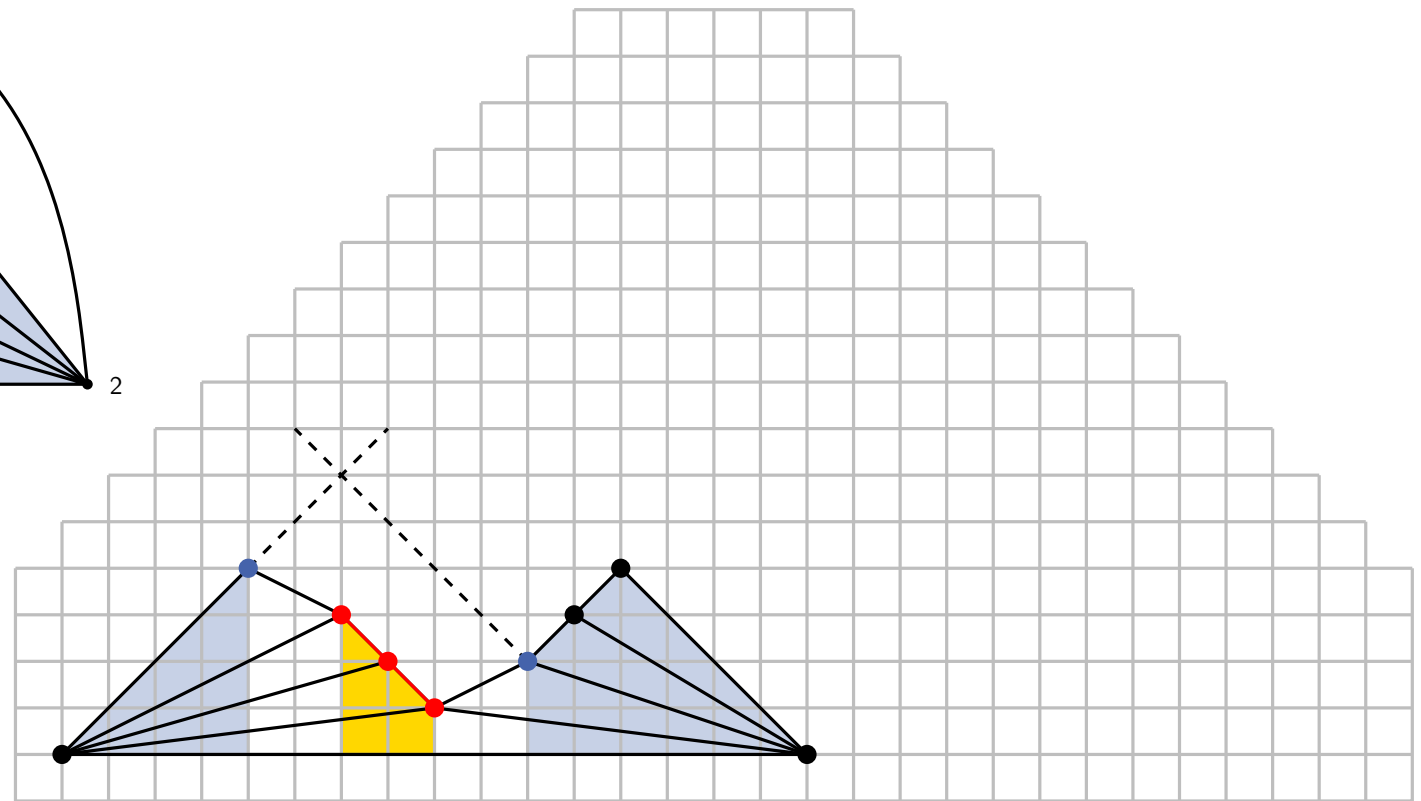
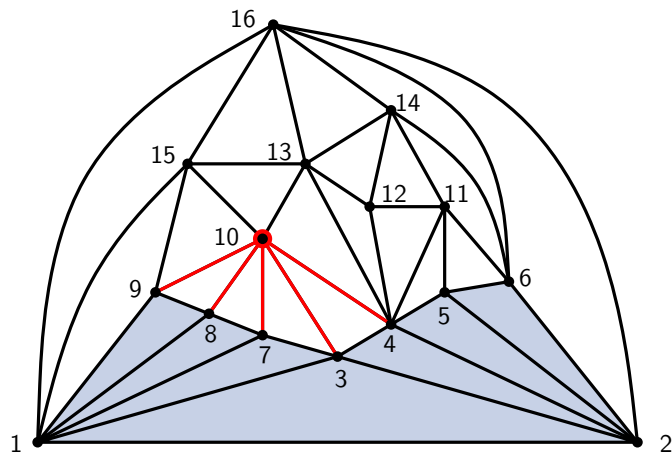
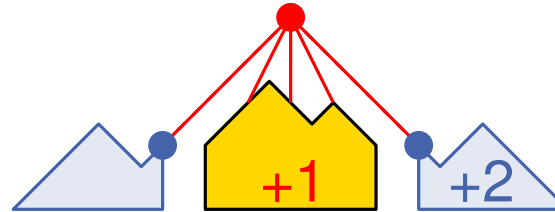
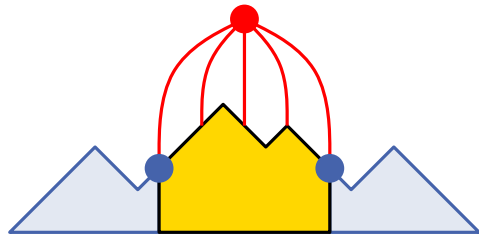
16 - 10

De Fraysseix Pach Pollack (Shift) Algorithm



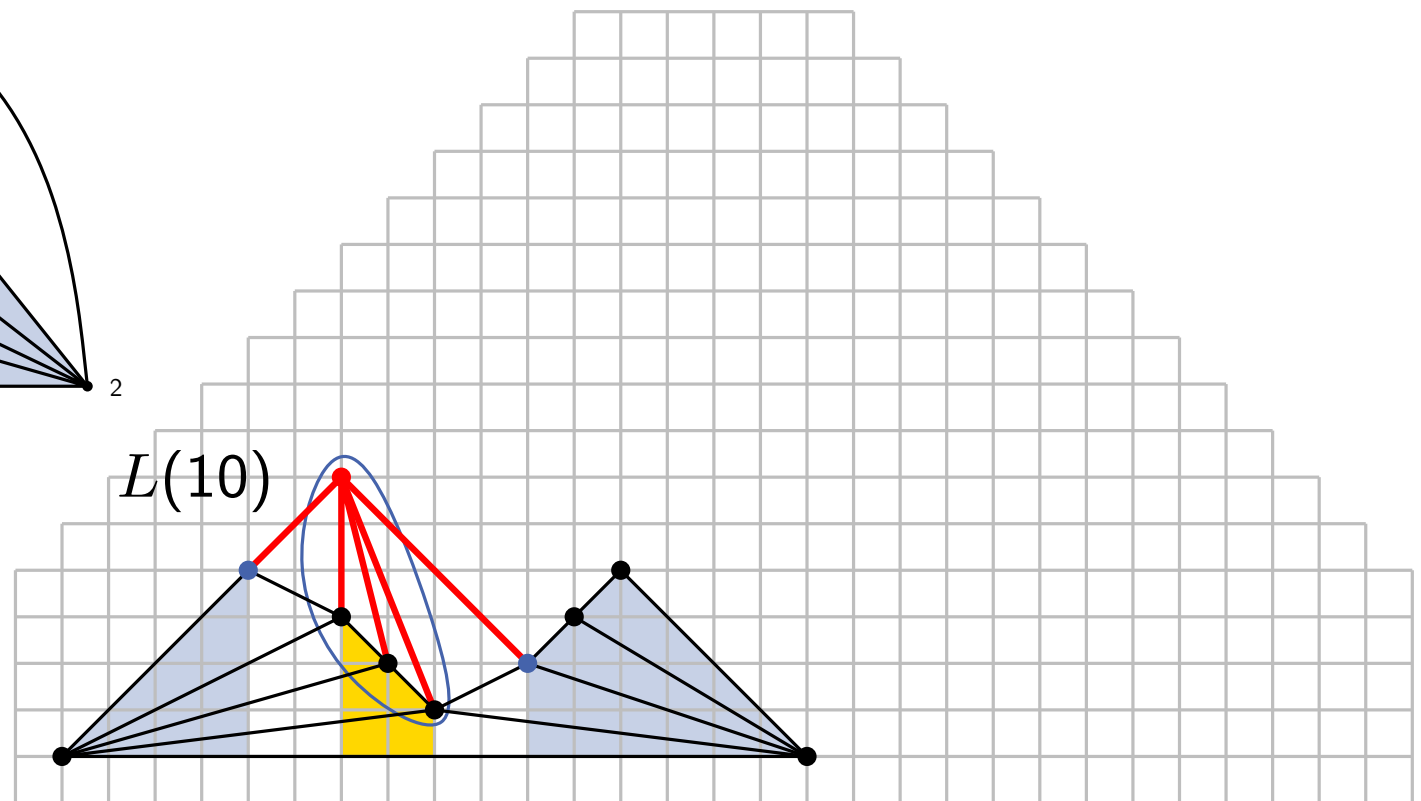
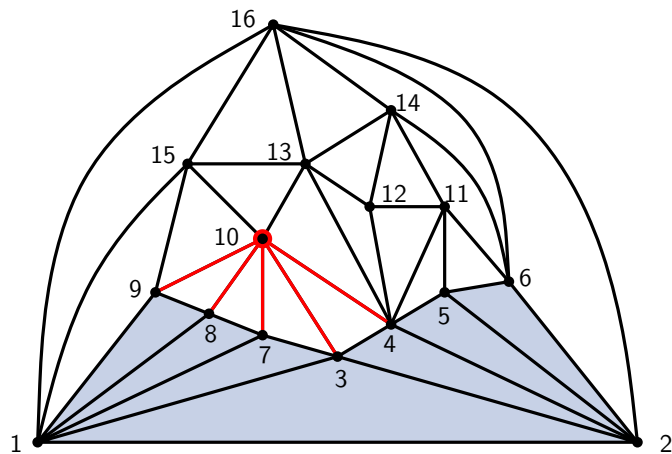
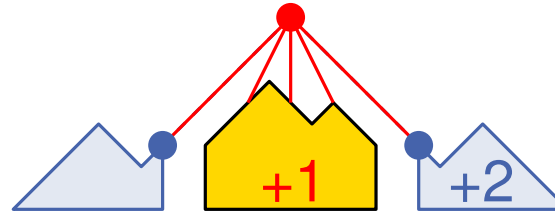
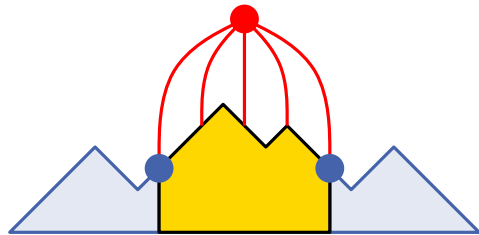
16 - 11

De Fraysseix Pach Pollack (Shift) Algorithm



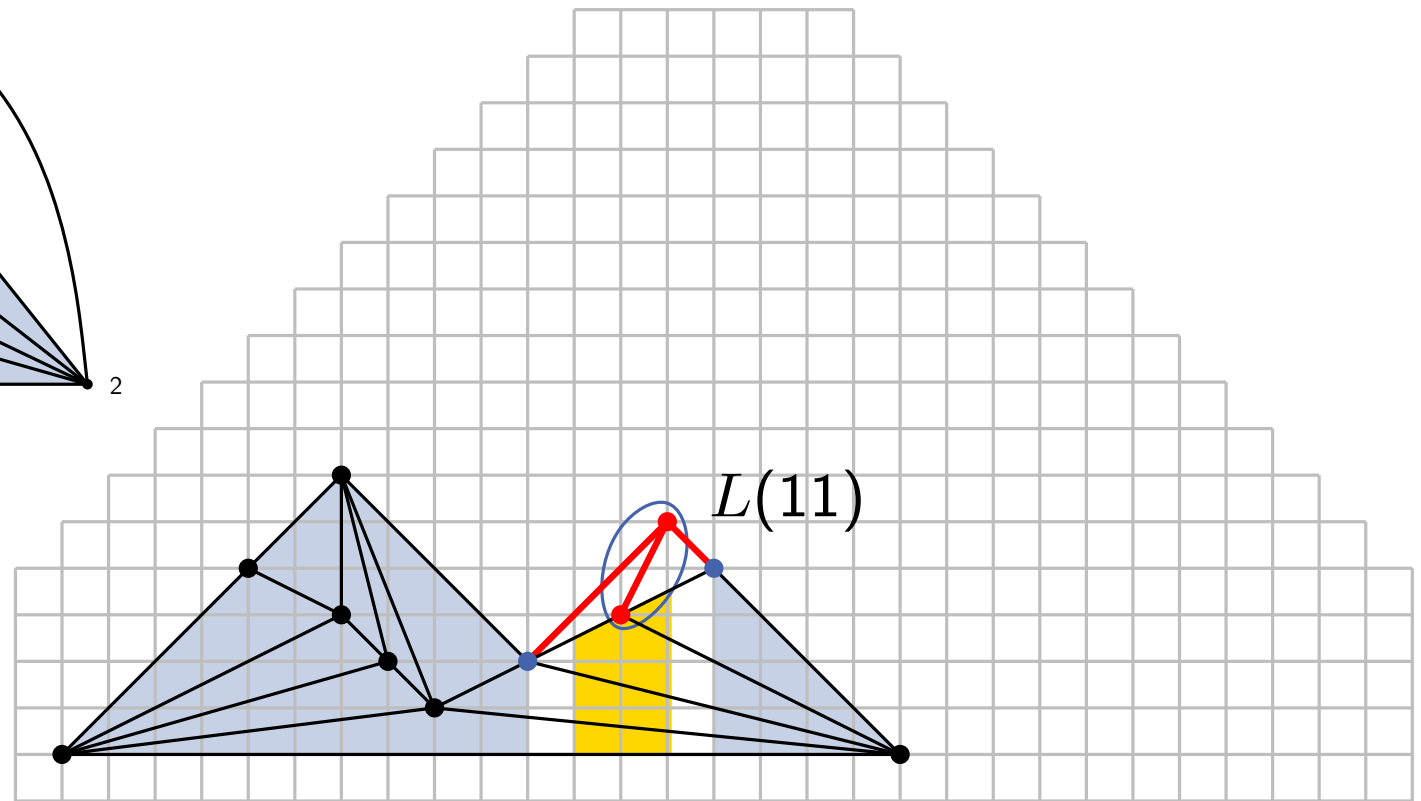
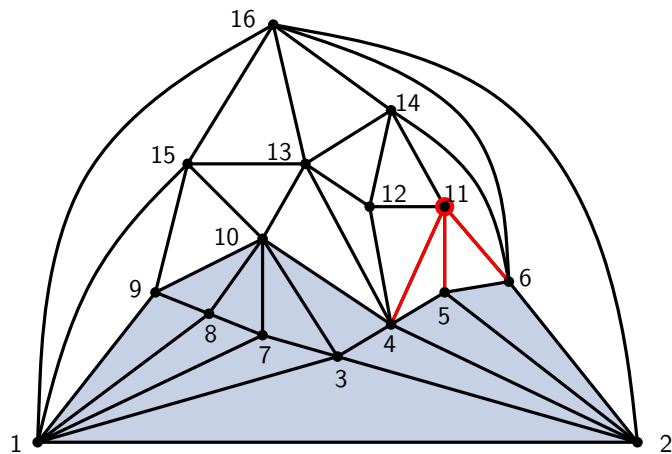
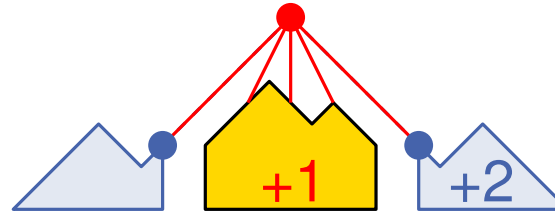
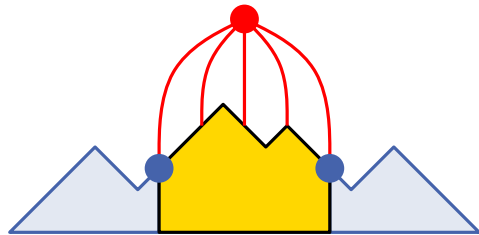
16 - 12

De Fraysseix Pach Pollack (Shift) Algorithm



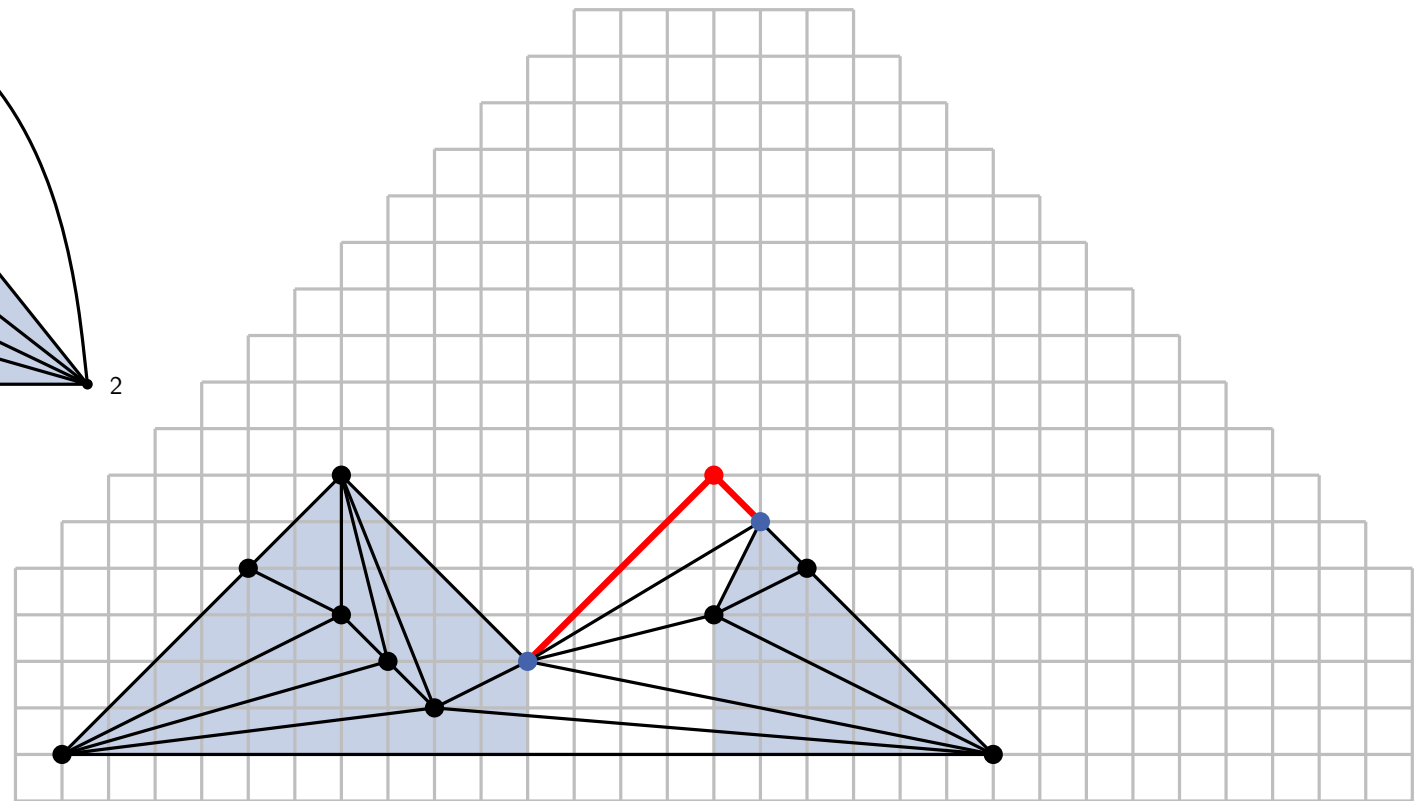
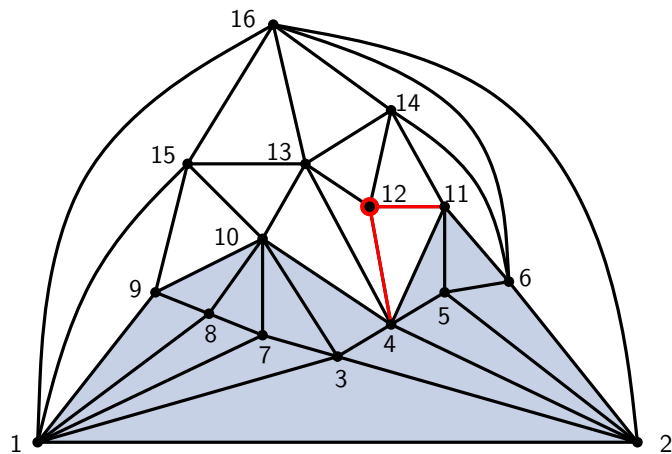
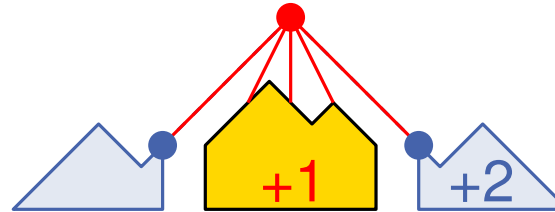
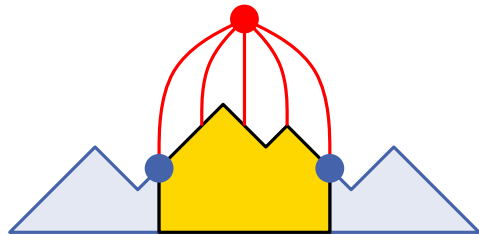
16 - 13

De Fraysseix Pach Pollack (Shift) Algorithm



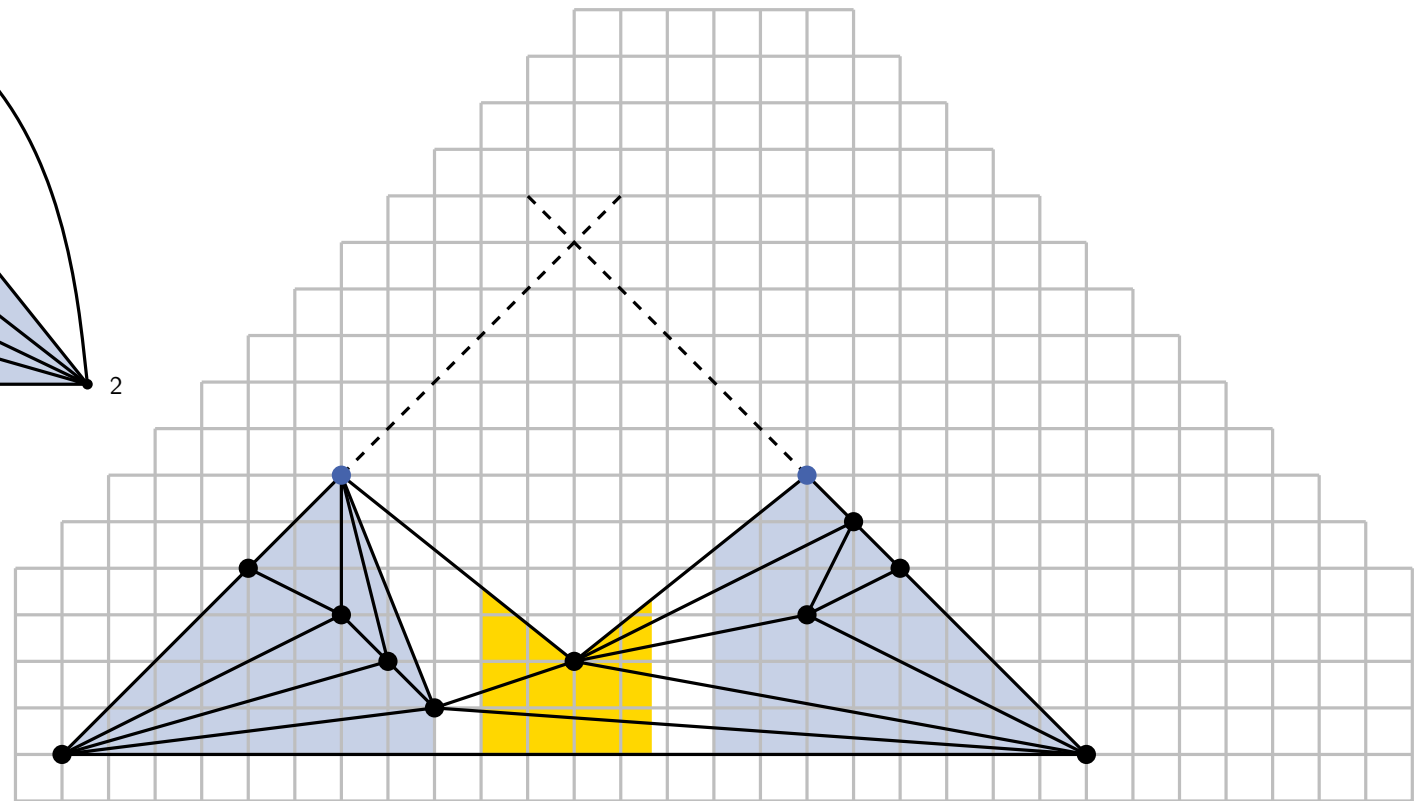
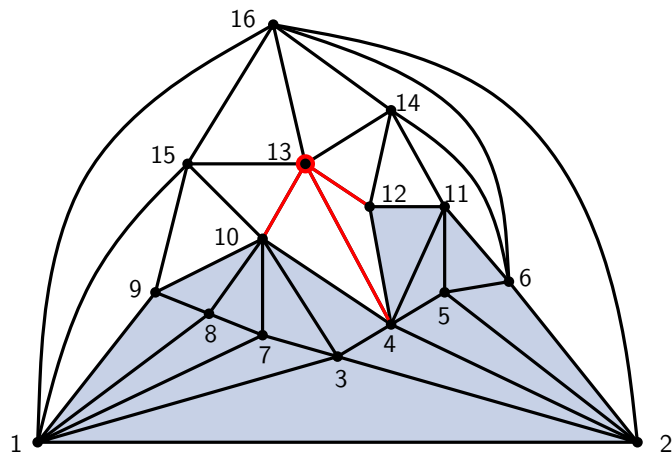
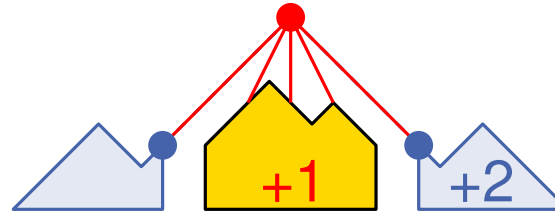
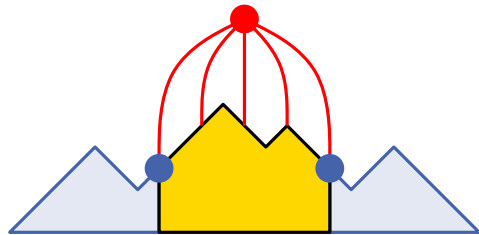
16 - 14

De Fraysseix Pach Pollack (Shift) Algorithm



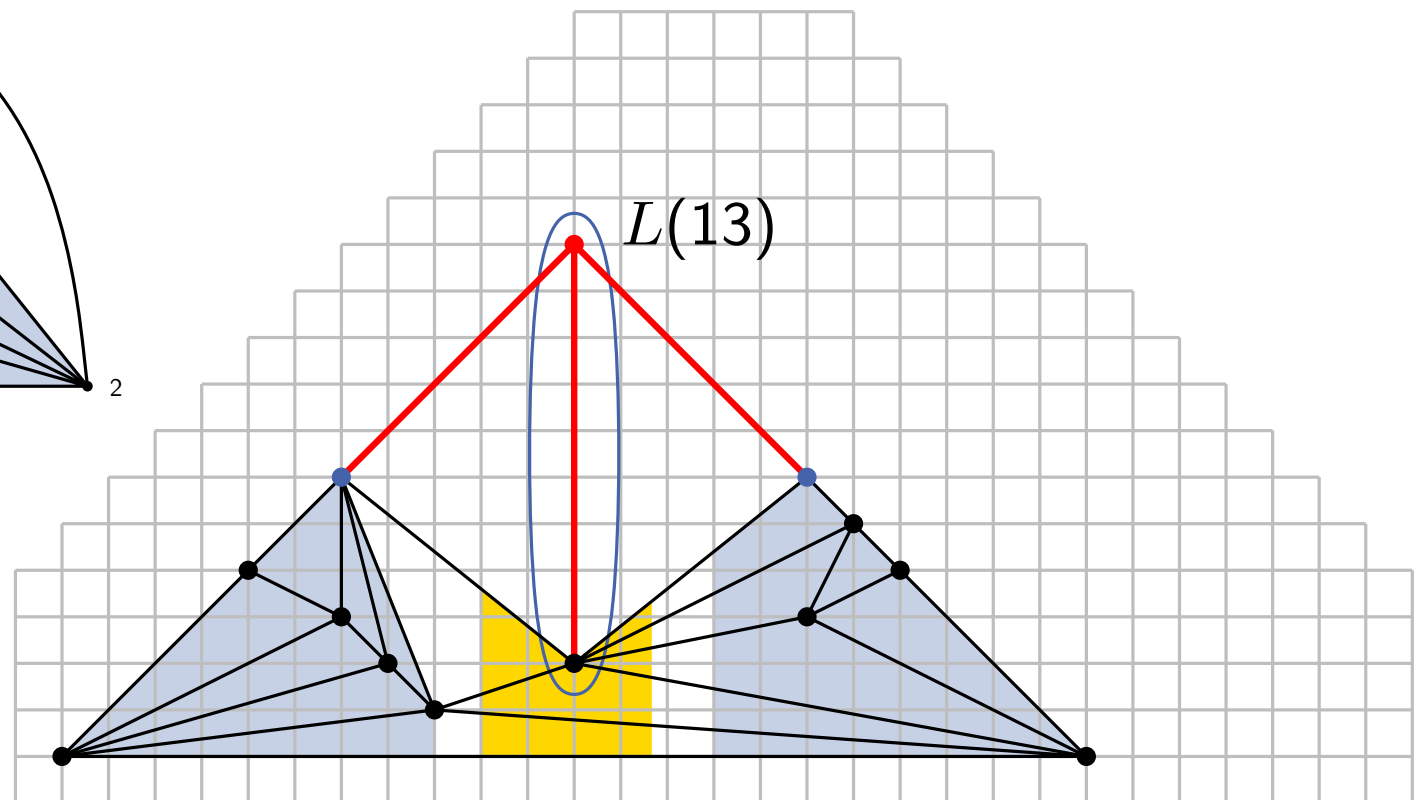
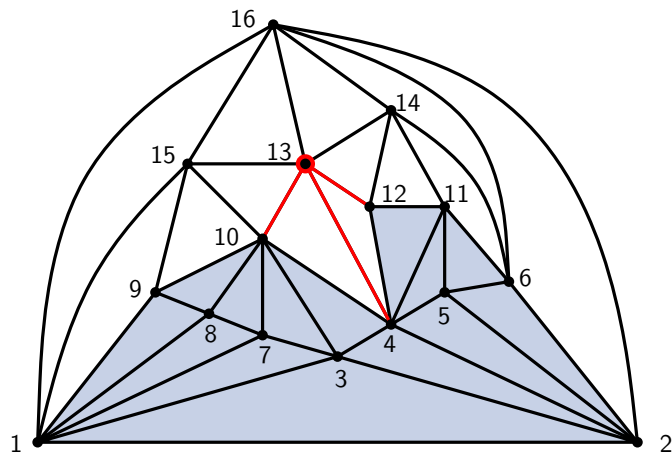
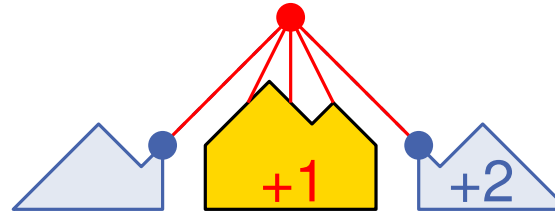
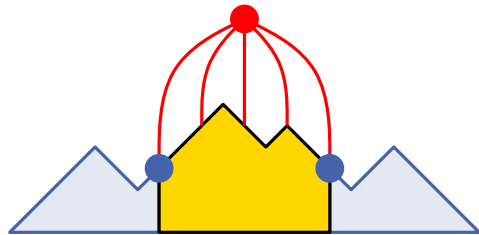
16 - 15

De Fraysseix Pach Pollack (Shift) Algorithm



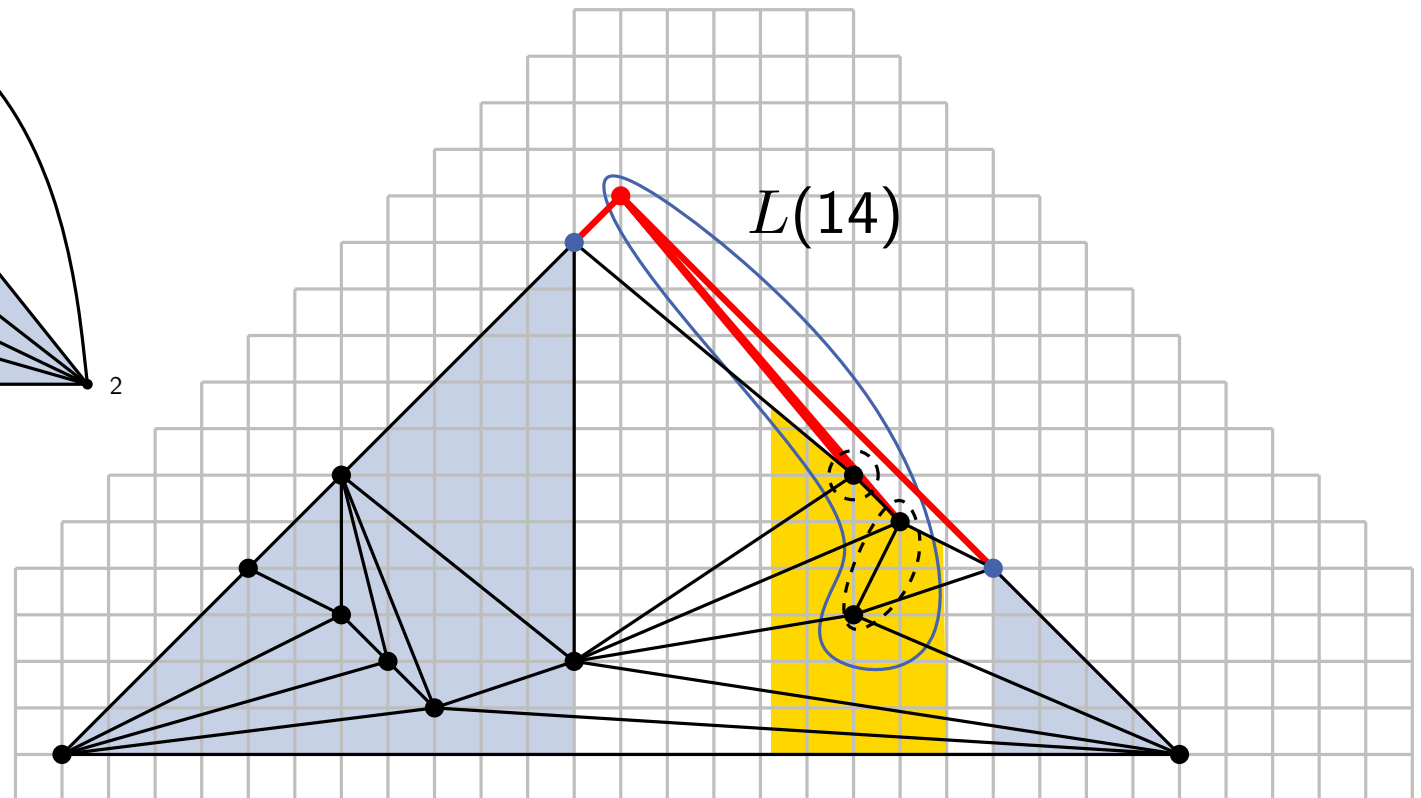
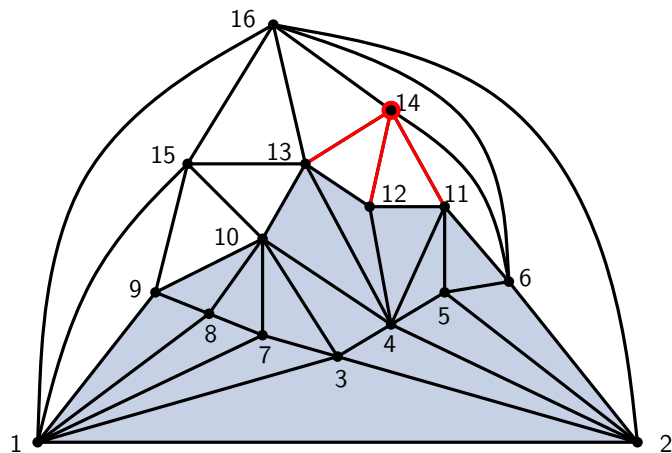
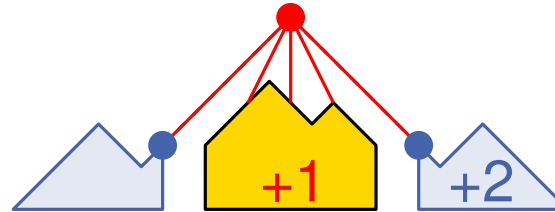
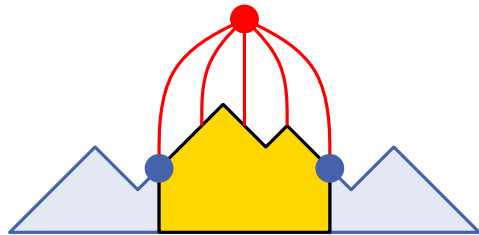
16 - 16

De Fraysseix Pach Pollack (Shift) Algorithm



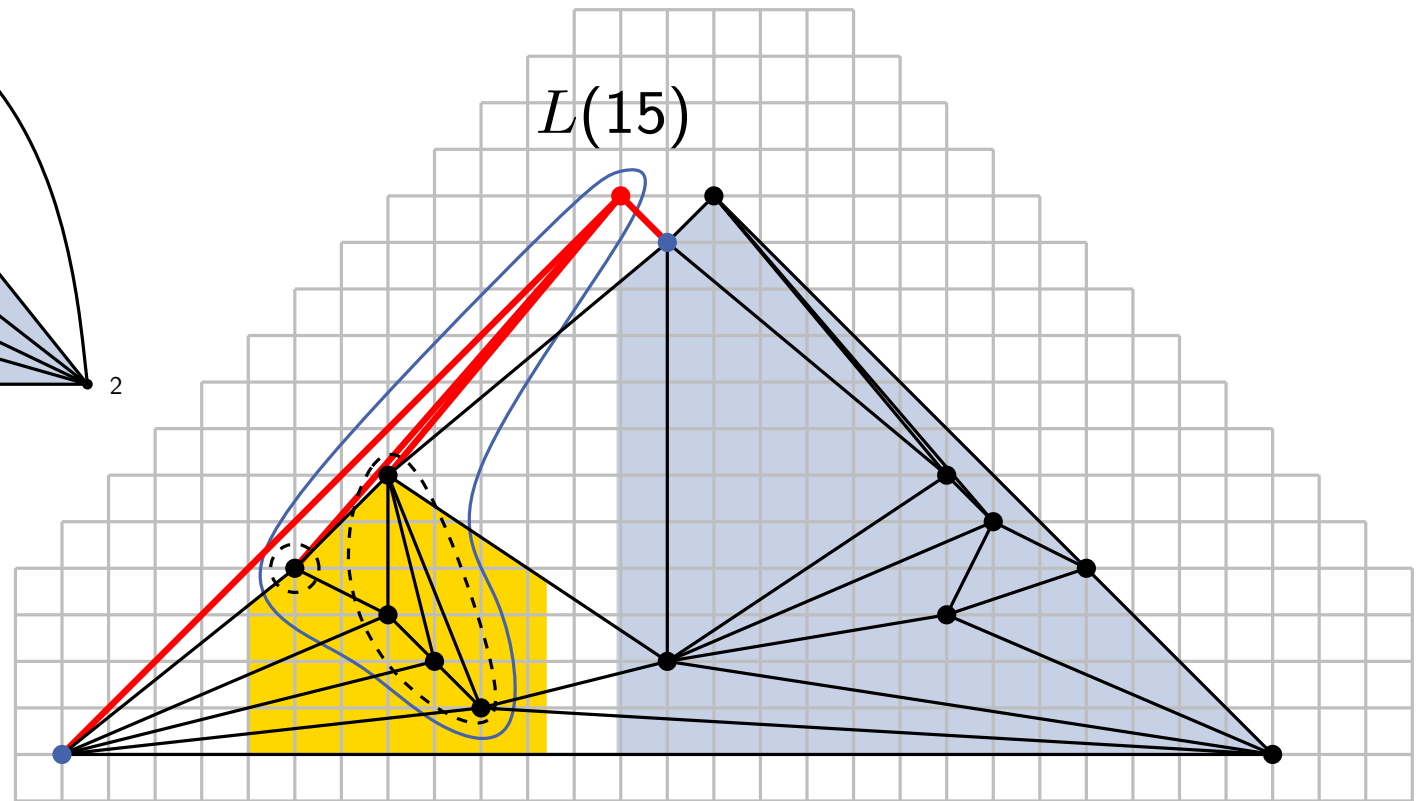
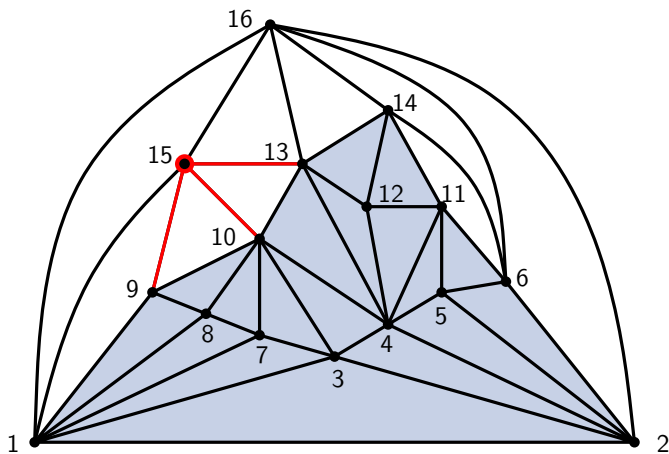
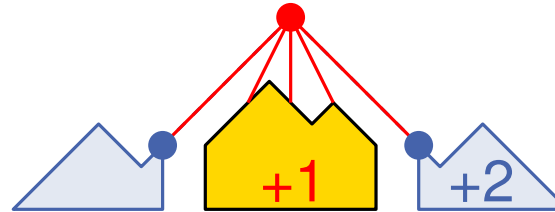
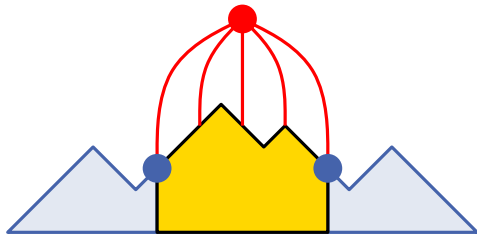
16 - 17

De Fraysseix Pach Pollack (Shift) Algorithm



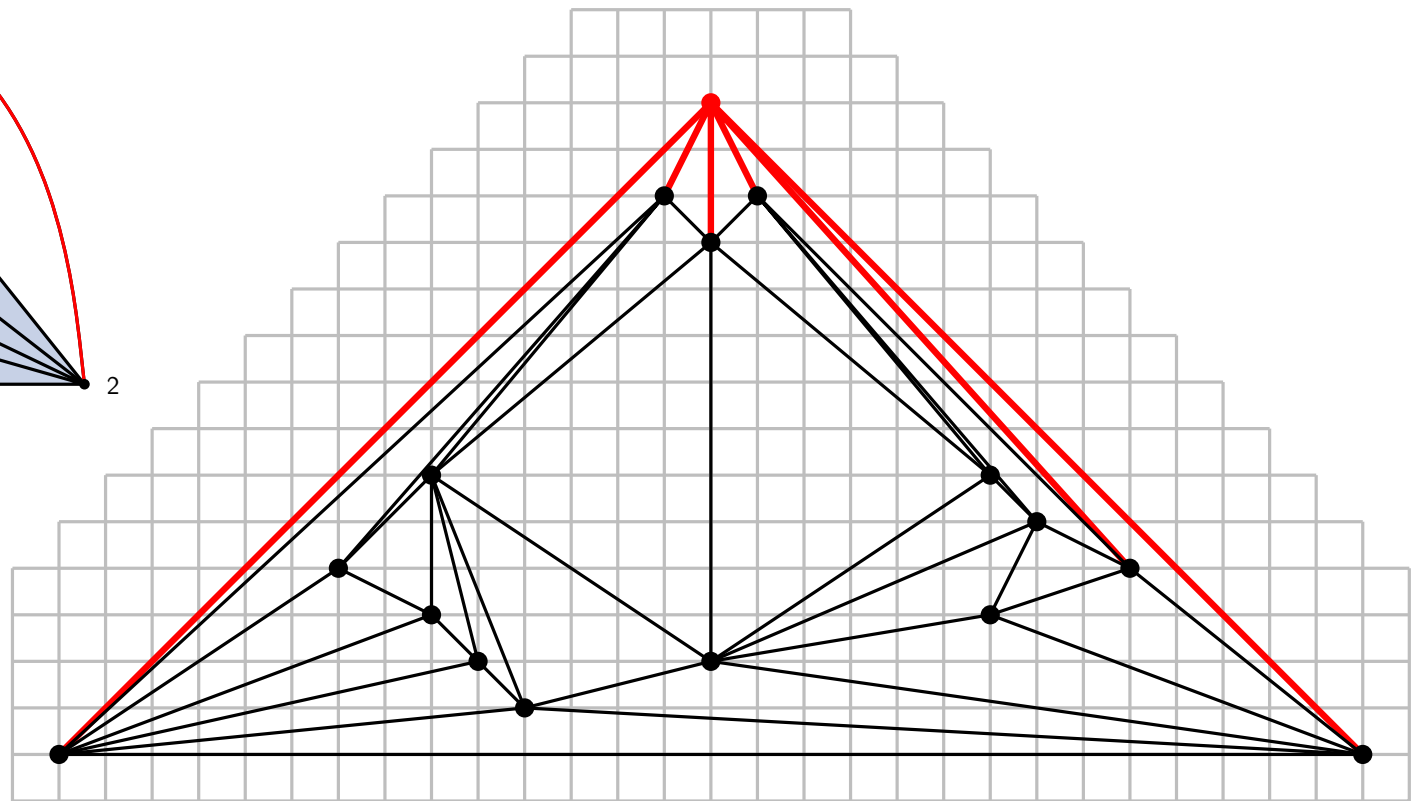
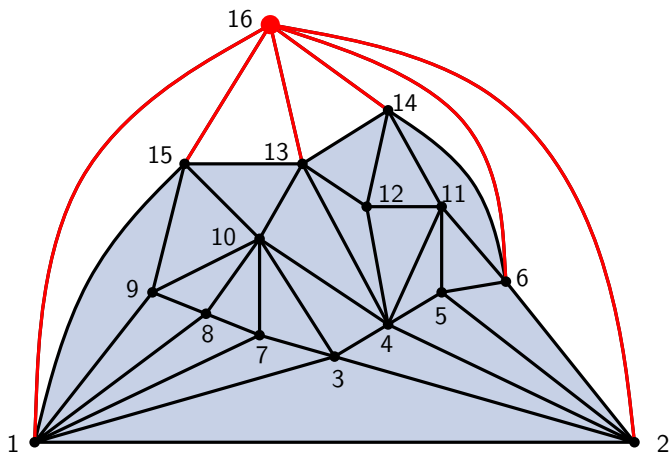
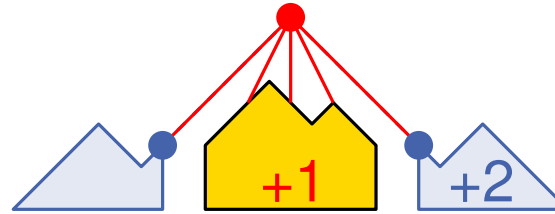
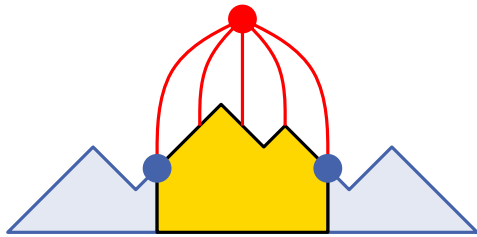
16 - 18

De Fraysseix Pach Pollack (Shift) Algorithm



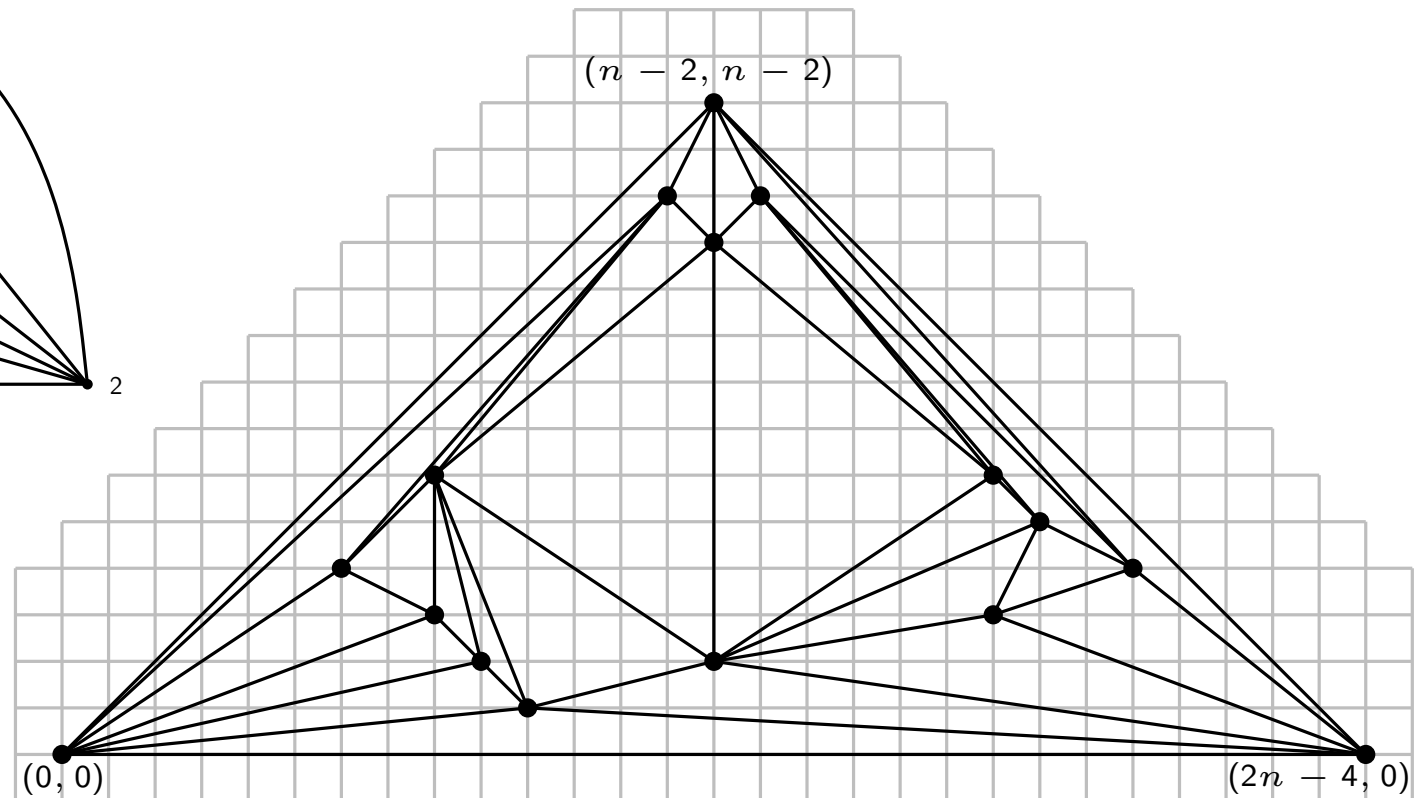
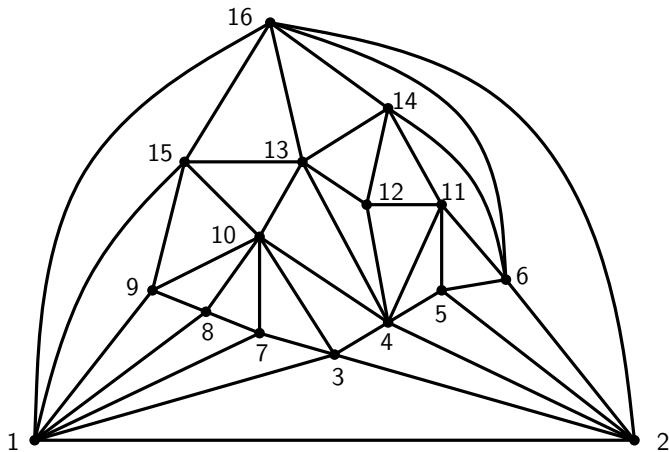
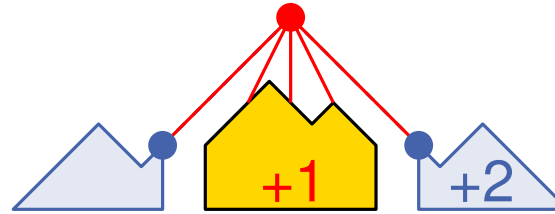
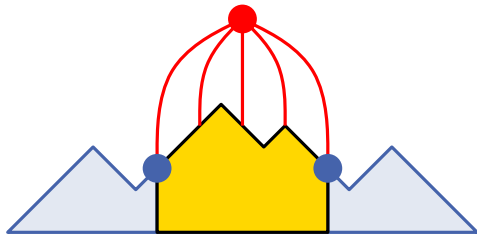
16 - 19

De Fraysseix Pach Pollack (Shift) Algorithm



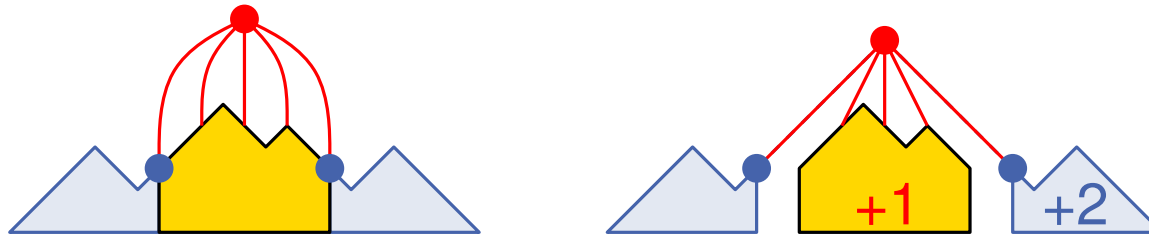
16 - 20

De Fraysseix Pach Pollack (Shift) Algorithm



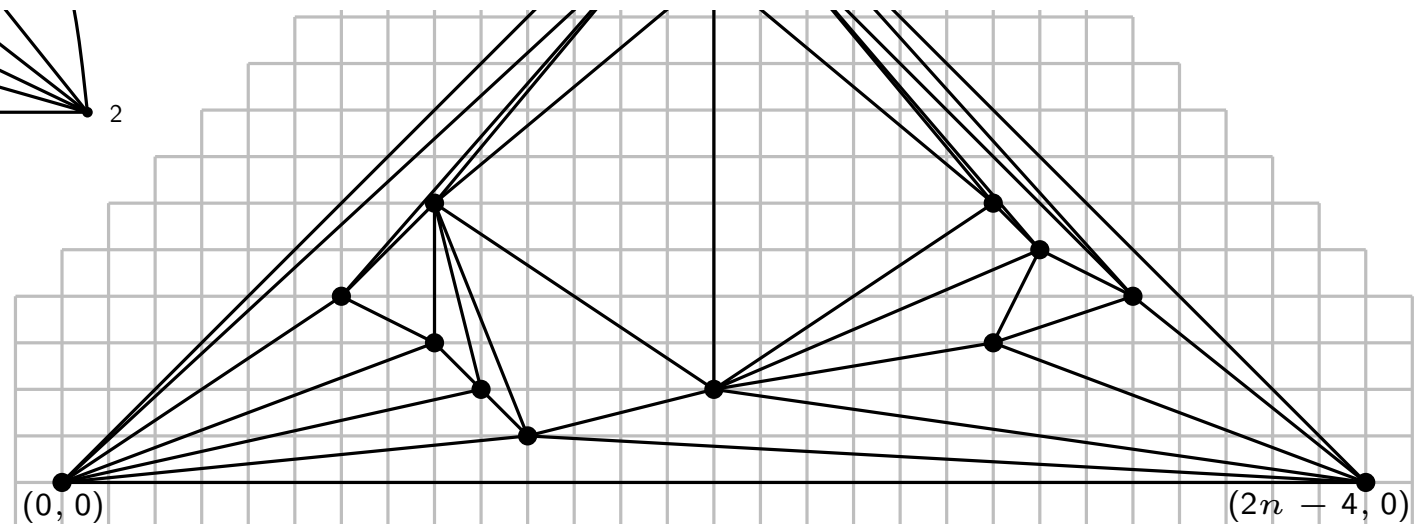
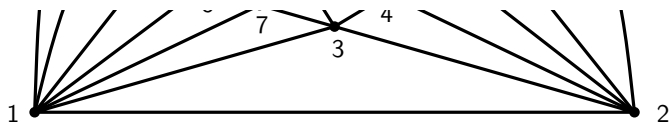
16 - 21

De Fraysseix Pach Pollack (Shift) Algorithm



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

1 min

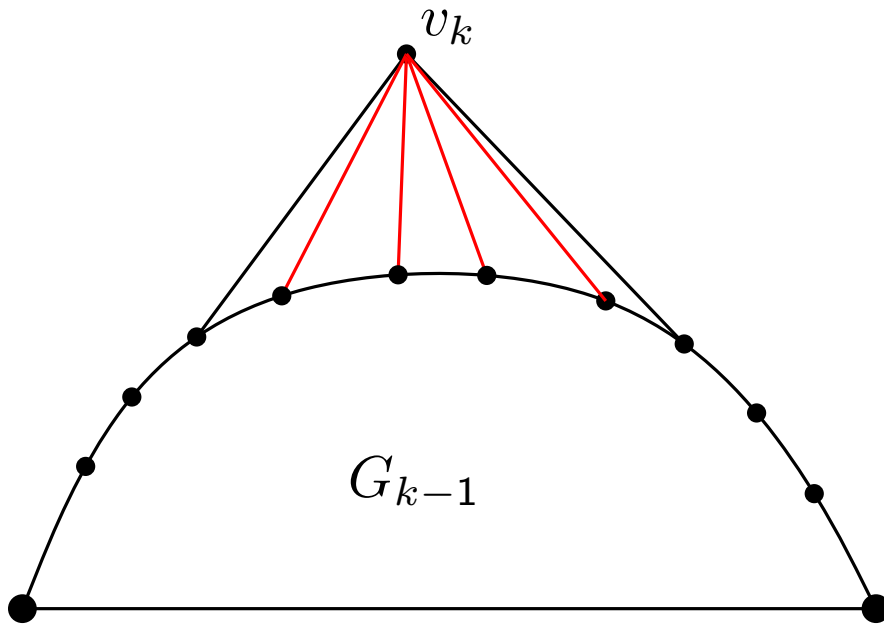


16 - 22

Outline

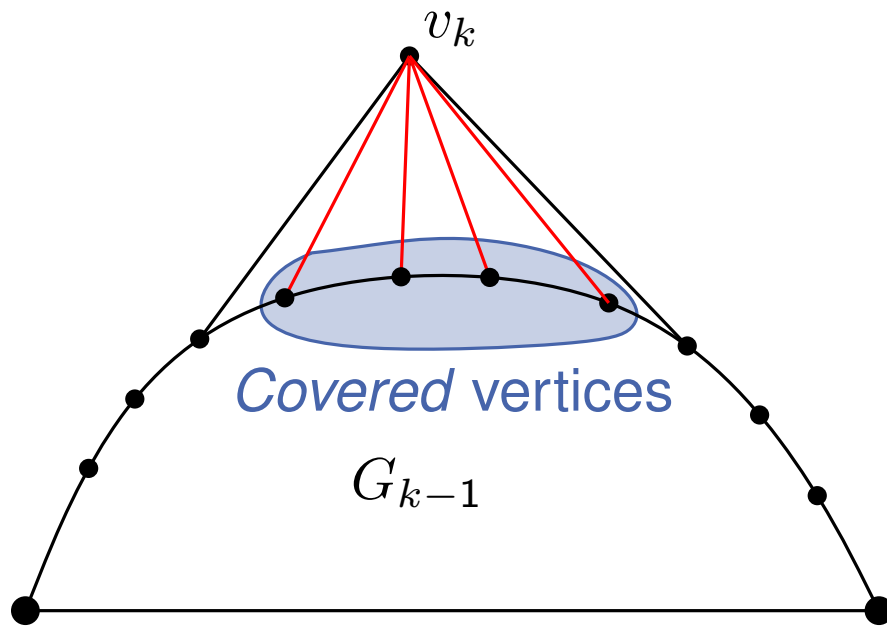
- Canonical ordering. Existence.
- Canonical ordering. Computation.
- Shift algorithm.
- **Proof of planarity.**
- Implementational details.

Proof of Planarity



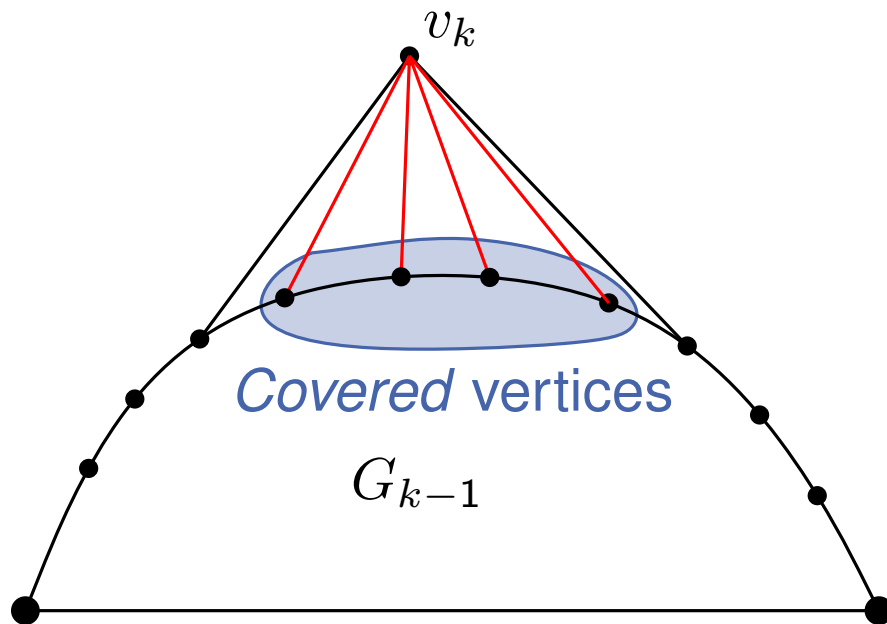
18 - 1

Proof of Planarity



18 - 2

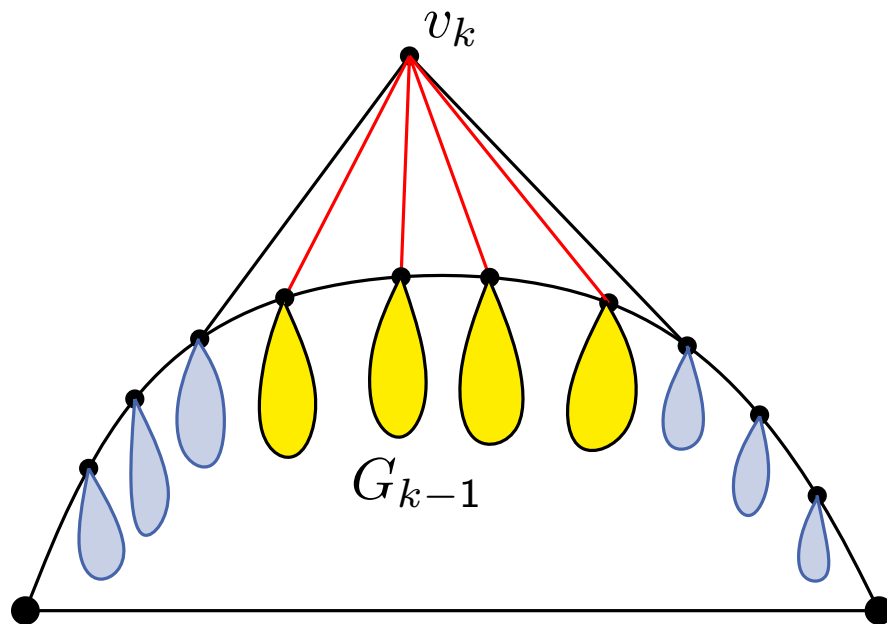
Proof of Planarity



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

18 - 3

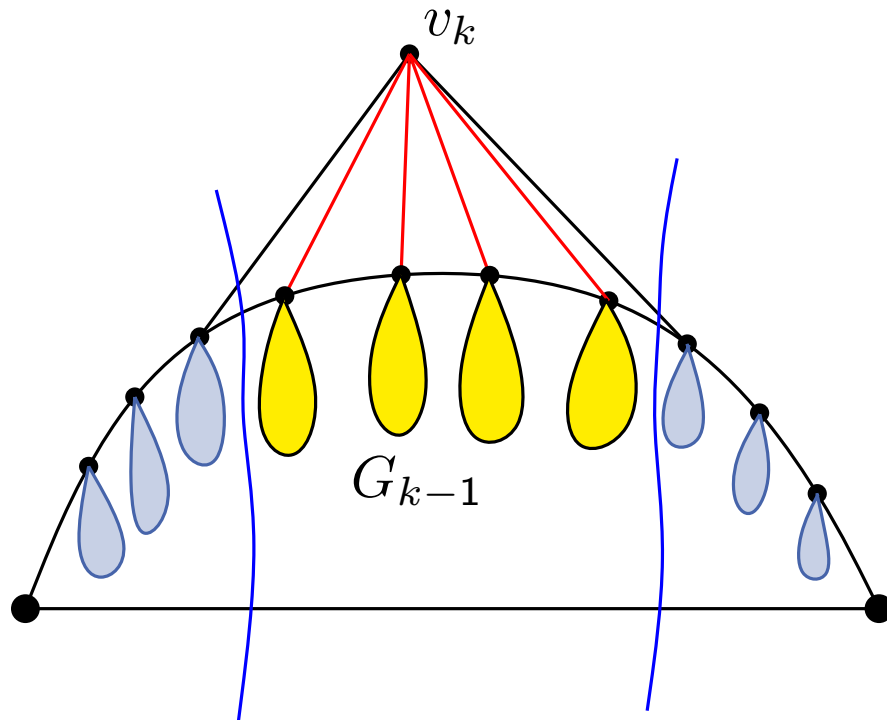
Proof of Planarity



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

18 - 4

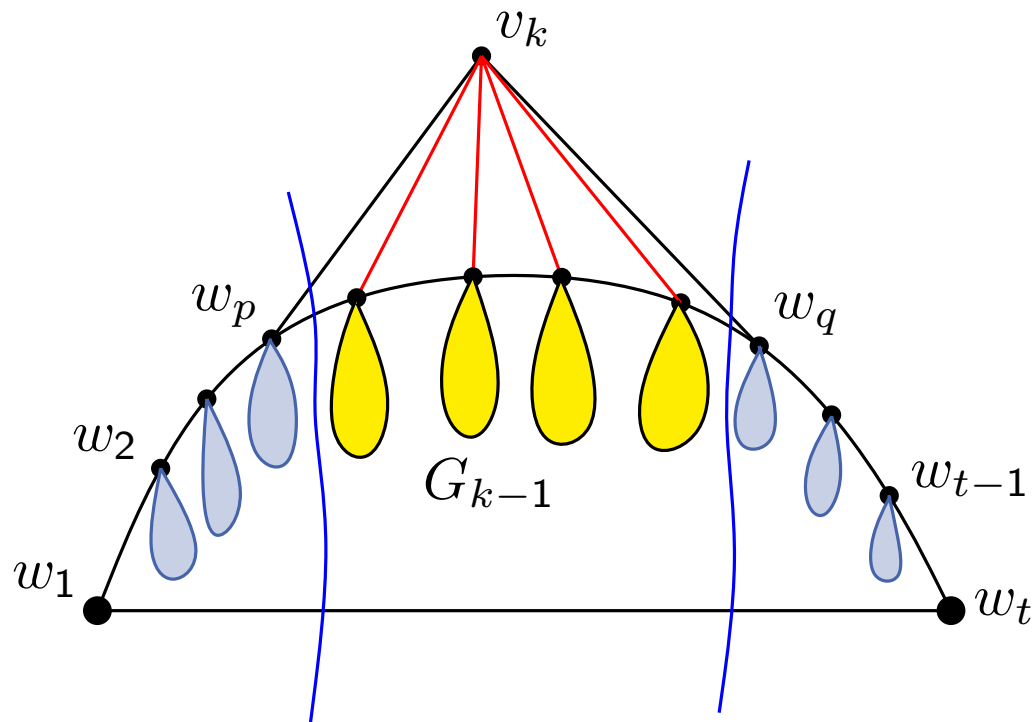
Proof of Planarity



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

18 - 5

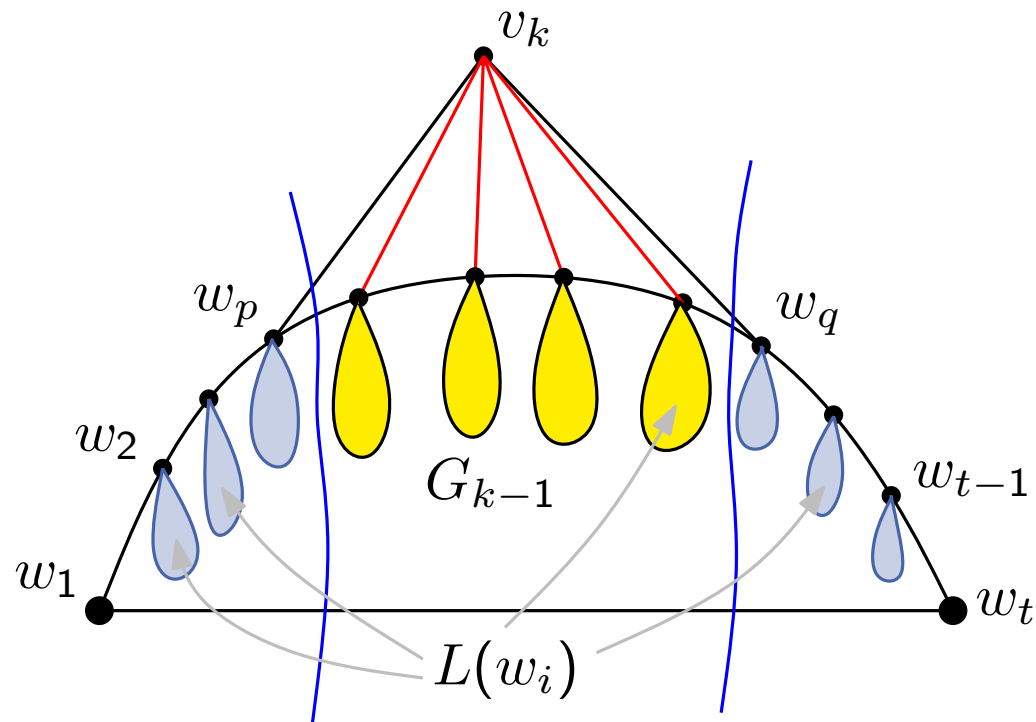
Proof of Planarity



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

18 - 6

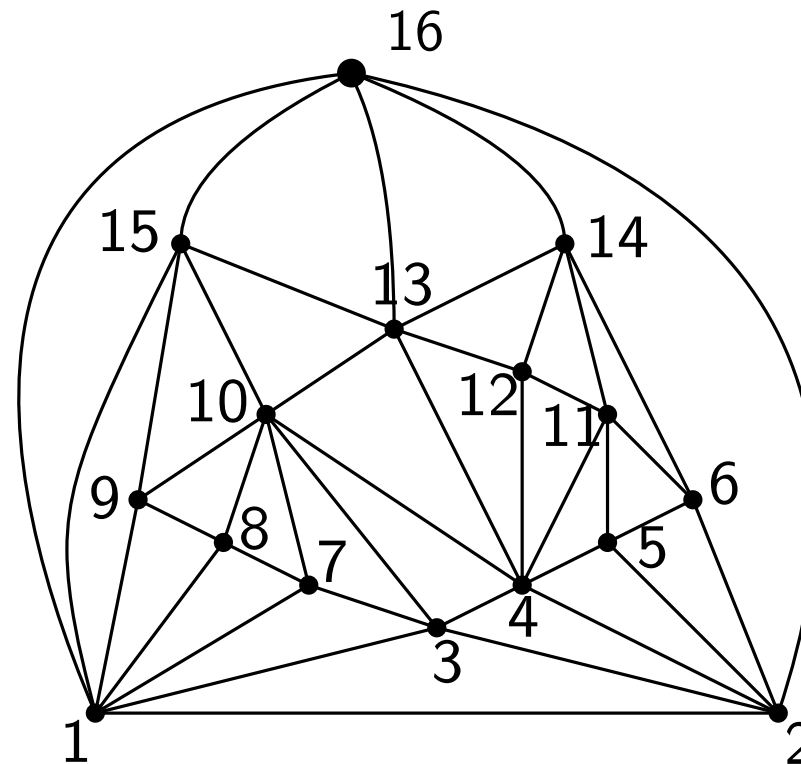
Proof of Planarity



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

18 - 7

Proof of Planarity

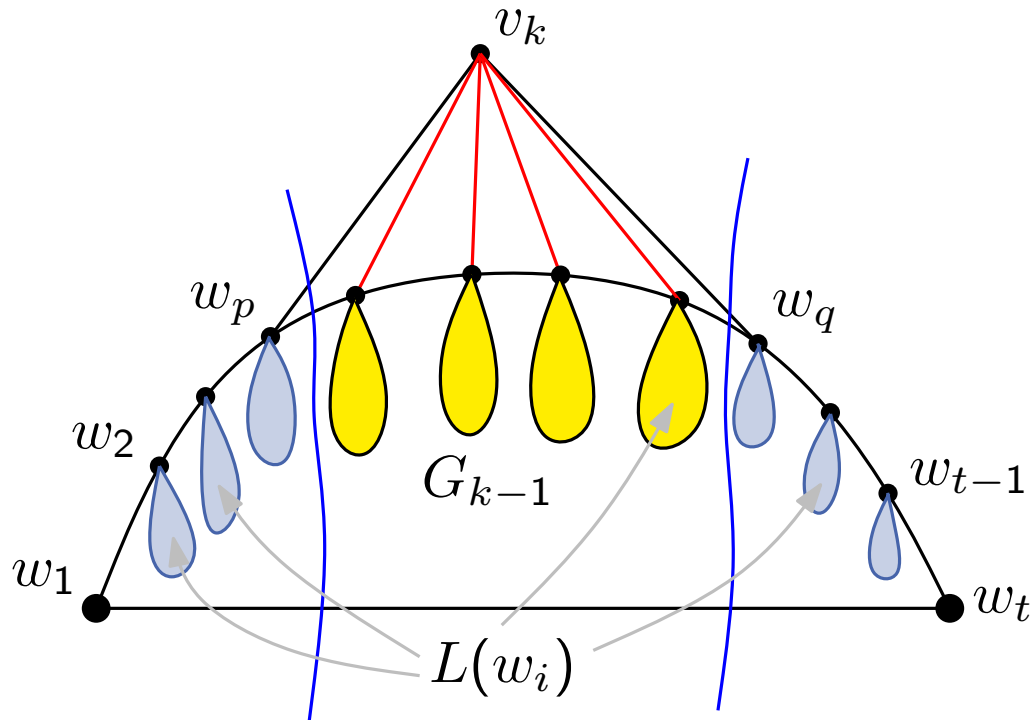


Work with your neighbour(s) and then share

5 min

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

19



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

Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$, 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.

Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$, 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.

Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$, 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, \dots, G_n .

Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$, 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, \dots, G_n .
- Assume that this is true for G_{k-1} .

Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$, 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, \dots, G_n .
- Assume that this is true for G_{k-1} .
- Let $w_1, \dots, w_p, v_k, w_q, \dots, w_t$ be the boundary of G_k .

Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$, 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, \dots, G_n .
- Assume that this is true for G_{k-1} .
- Let $w_1, \dots, w_p, v_k, w_q, \dots, w_t$ be the boundary of G_k .
- Let $\delta_1 \leq \dots \leq \delta_p \leq \delta \leq \delta_q \leq \dots \leq \delta_t$.

Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$, 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, \dots, G_n .
- Assume that this is true for G_{k-1} .
- Let $w_1, \dots, w_p, v_k, w_q, \dots, w_t$ be the boundary of G_k .
- Let $\delta_1 \leq \dots \leq \delta_p \leq \delta \leq \delta_q \leq \dots \leq \delta_t$.
- 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 $q \leq i \leq t$.

Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$, 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, \dots, G_n .
- Assume that this is true for G_{k-1} .
- Let $w_1, \dots, w_p, v_k, w_q, \dots, w_t$ be the boundary of G_k .
- Let $\delta_1 \leq \dots \leq \delta_p \leq \delta \leq \delta_q \leq \dots \leq \delta_t$.
- 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 $w_1 \dots, w_t$ by $\delta'_1 \dots \delta'_t$, respectively.

Lemma

Let $0 < \delta_1 \leq \delta_2 \leq \dots \leq \delta_t \in \mathbb{N}$, 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, \dots, G_n .
- Assume that this is true for G_{k-1} .
- Let $w_1, \dots, w_p, v_k, w_q, \dots, w_t$ be the boundary of G_k .
- Let $\delta_1 \leq \dots \leq \delta_p \leq \delta \leq \delta_q \leq \dots \leq \delta_t$.
- 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, \dots, w_t by $\delta'_1, \dots, \delta'_t$, respectively.
- We can complete the drawing by placing v_k , v_k is moved with $L(w_{p+1}), \dots, L(w_{q-1})$ by δ .

Outline

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

Algorithm Shift

Let v_1, \dots, 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 $w_1 = v_1, w_2, \dots, w_{t-1}, w_t = v_2$ denote the boundary of $G_{i-1};$
and let w_p, \dots, 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 $P(w_q);$

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

Algorithm Shift

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

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 $P(w_q);$

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



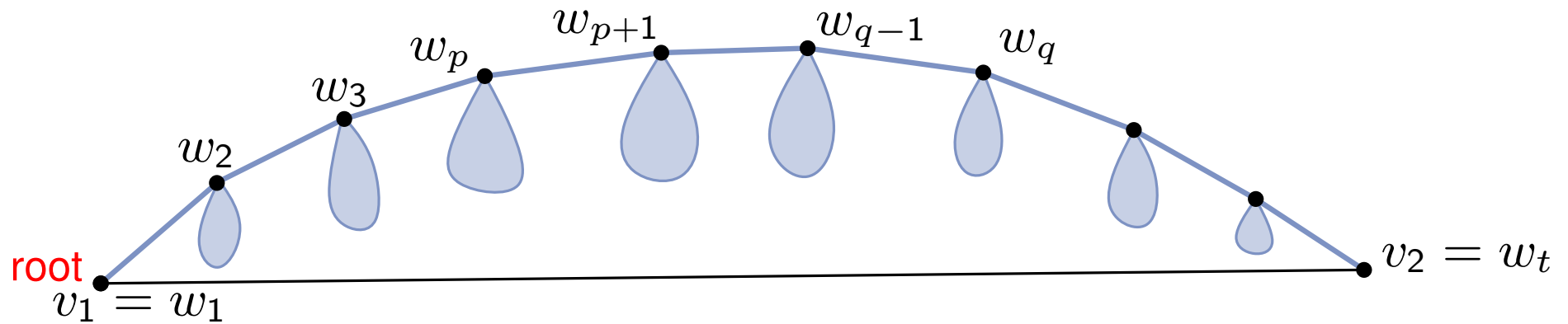
**Take a minute to think
about the time complexity of
the algorithm.**

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

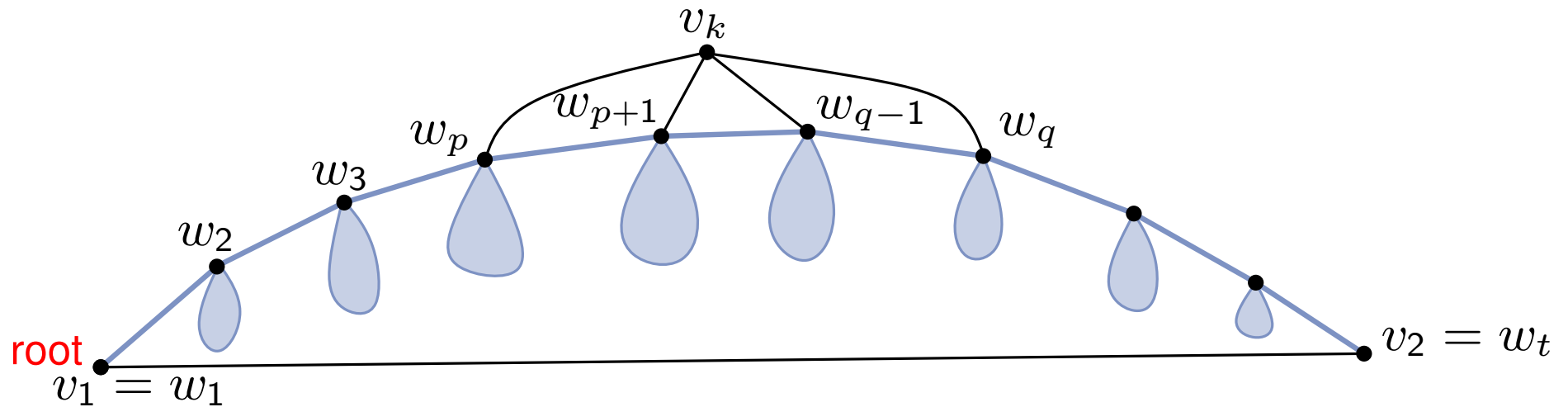


24 - 1

Implementation Details

relative x -distance tree

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

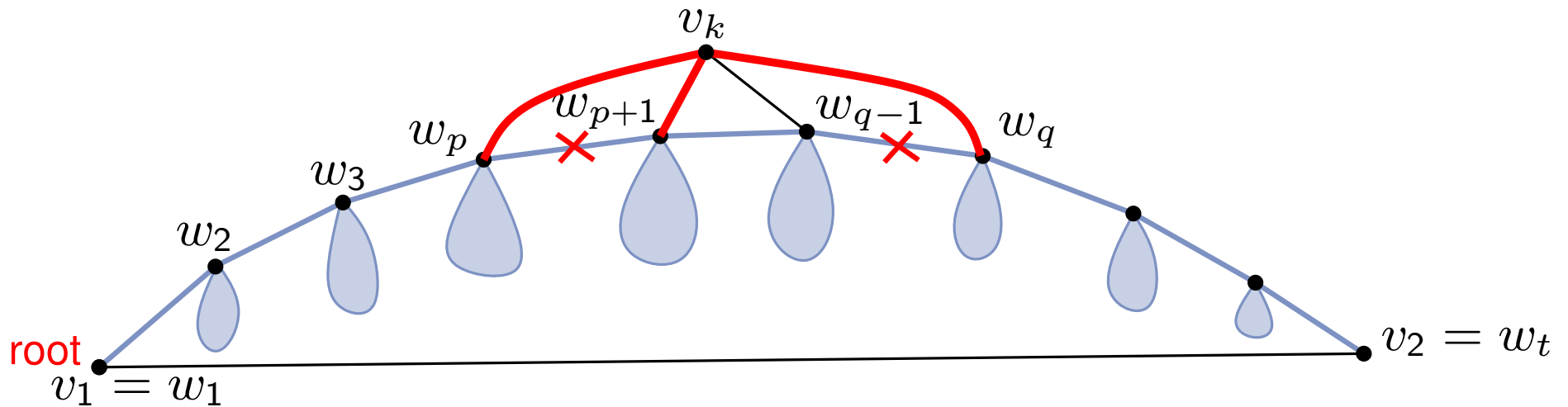


24 - 2

Implementation Details

relative x -distance tree

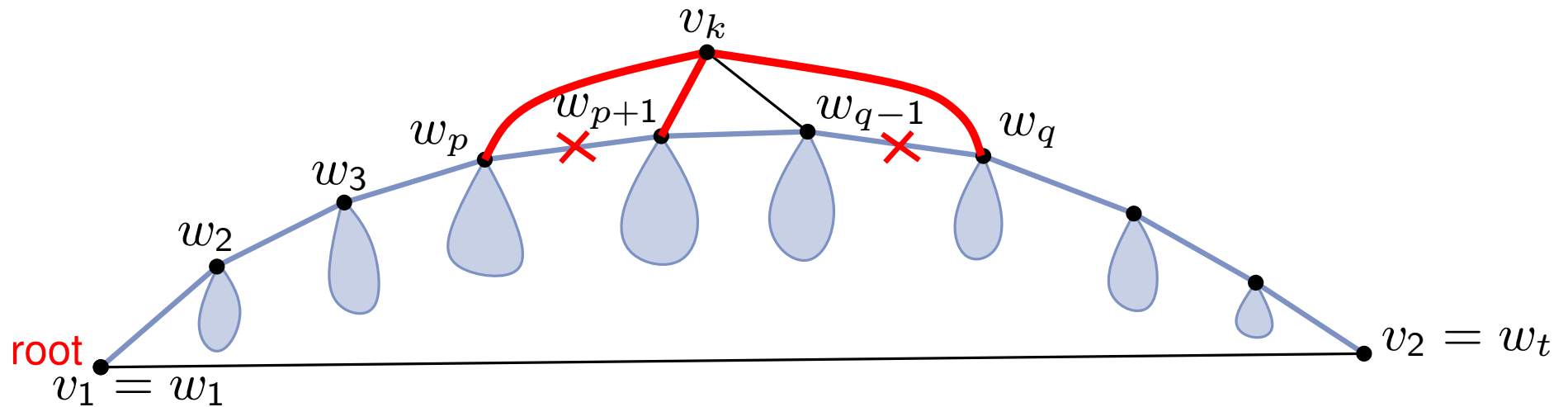
For each vertex store the x -offset from its parent and the y -coordinate



24 - 3

relative x -distance tree

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



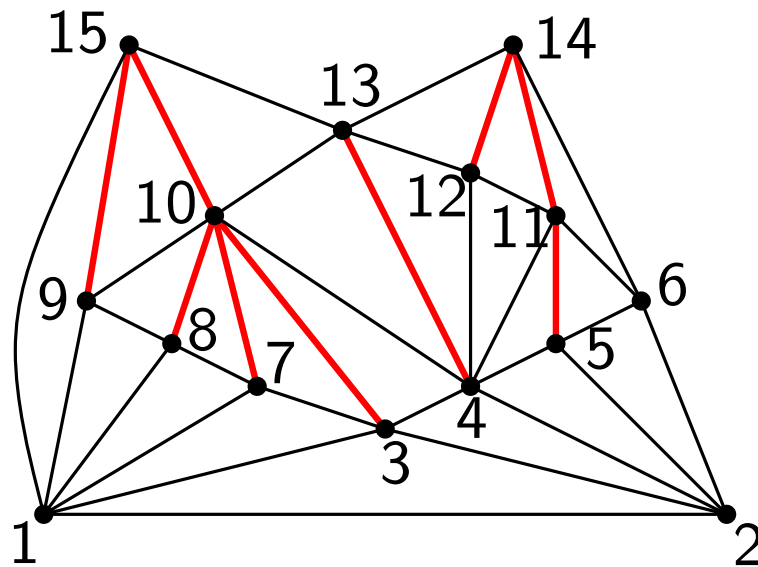
- $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$ (1)

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

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

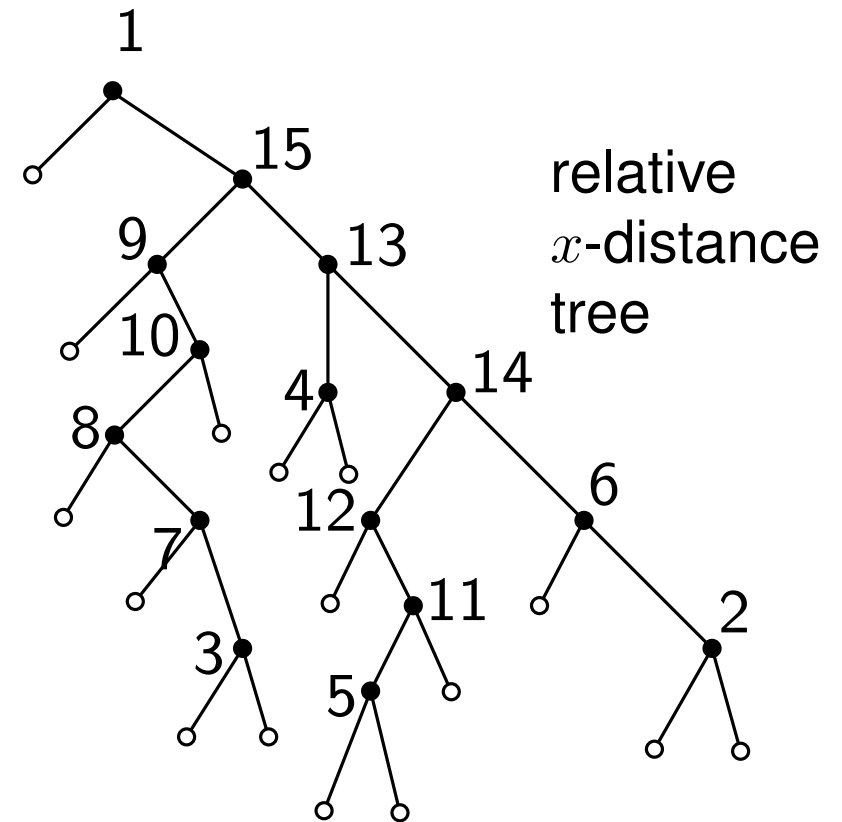
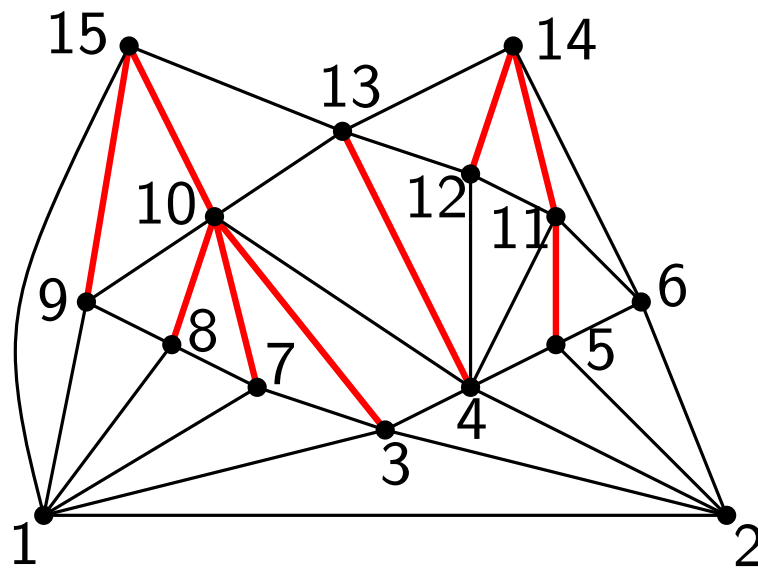
24 - 4

Implementation Details



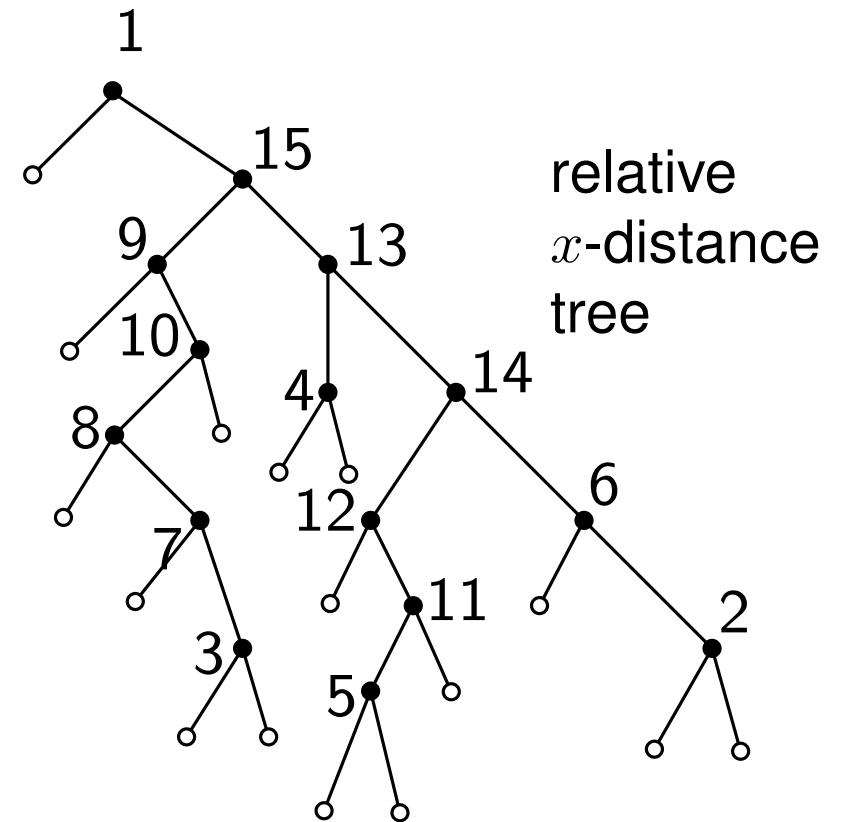
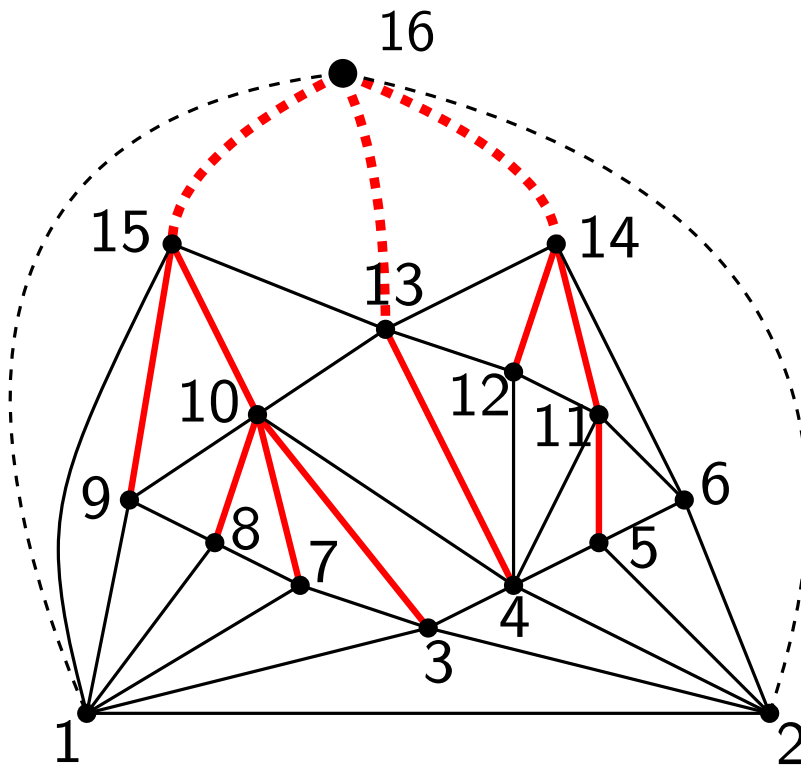
25 - 1

Implementation Details



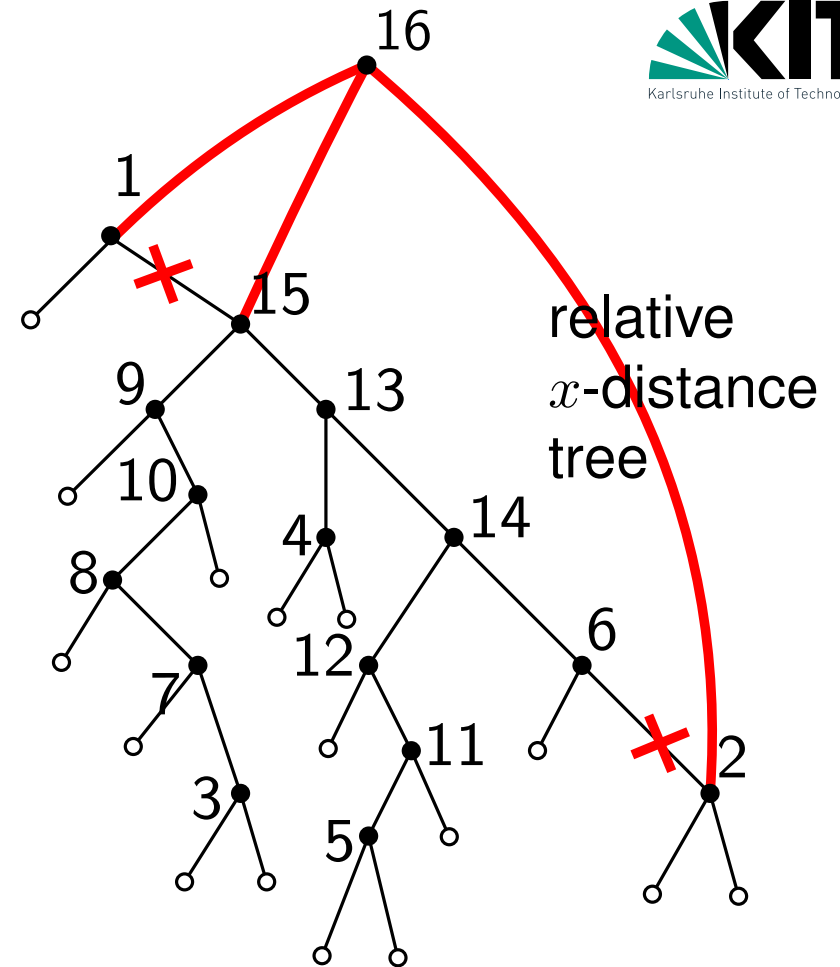
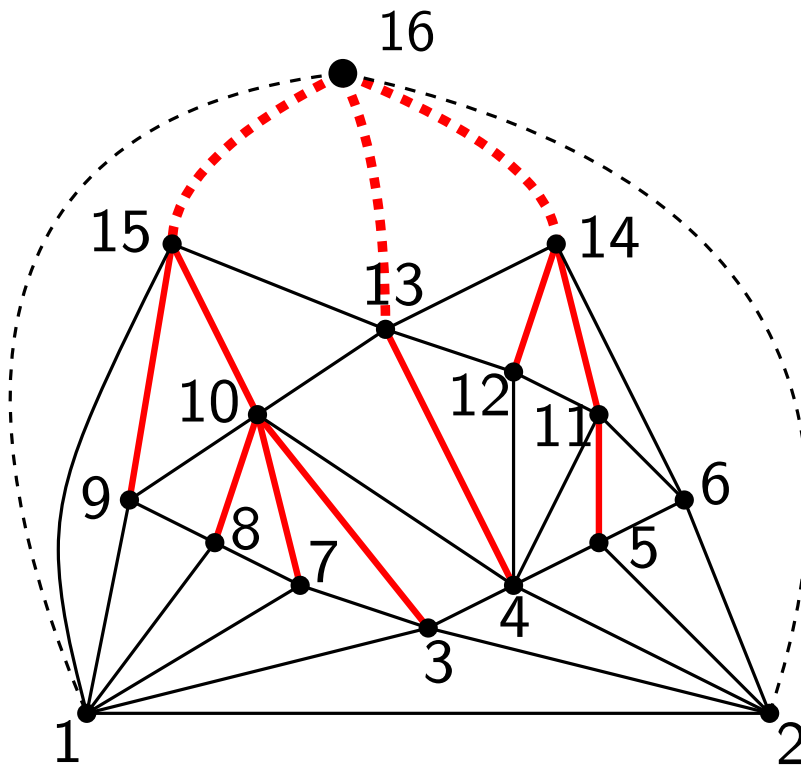
25 - 2

Implementation Details



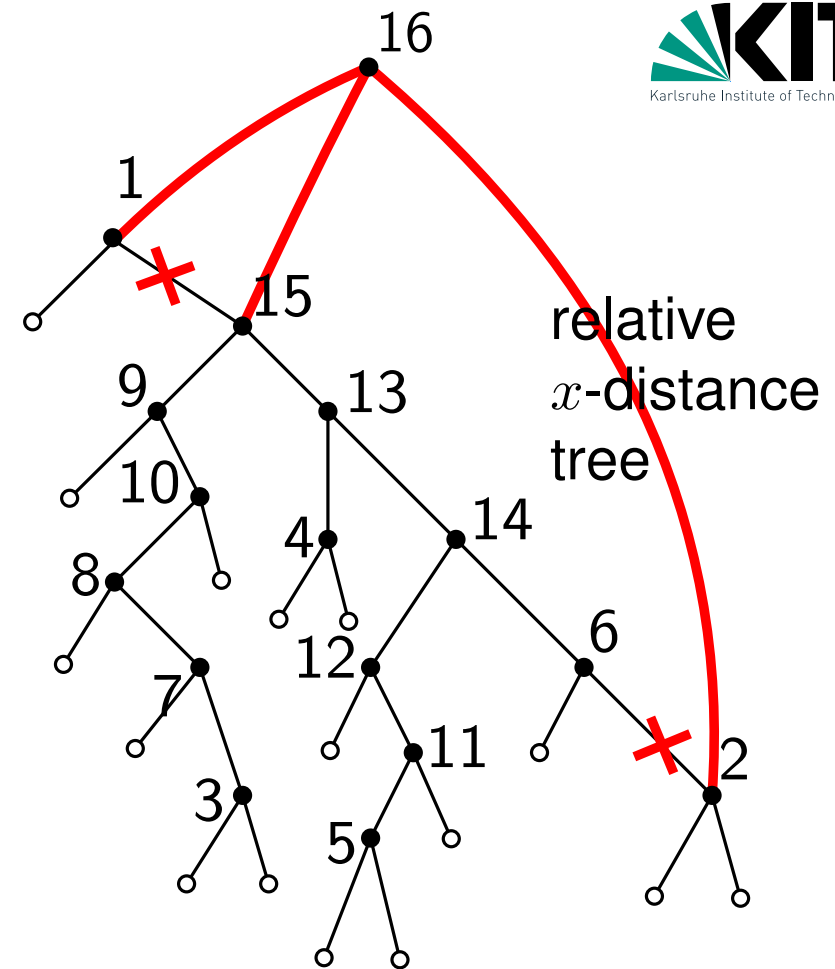
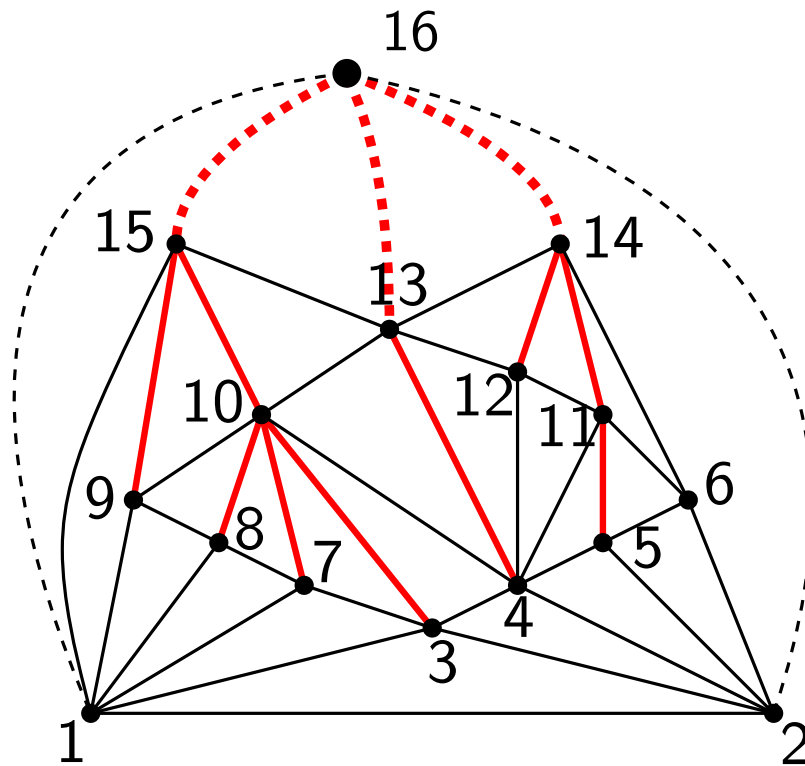
25 - 3

Implementation Details



25 - 4

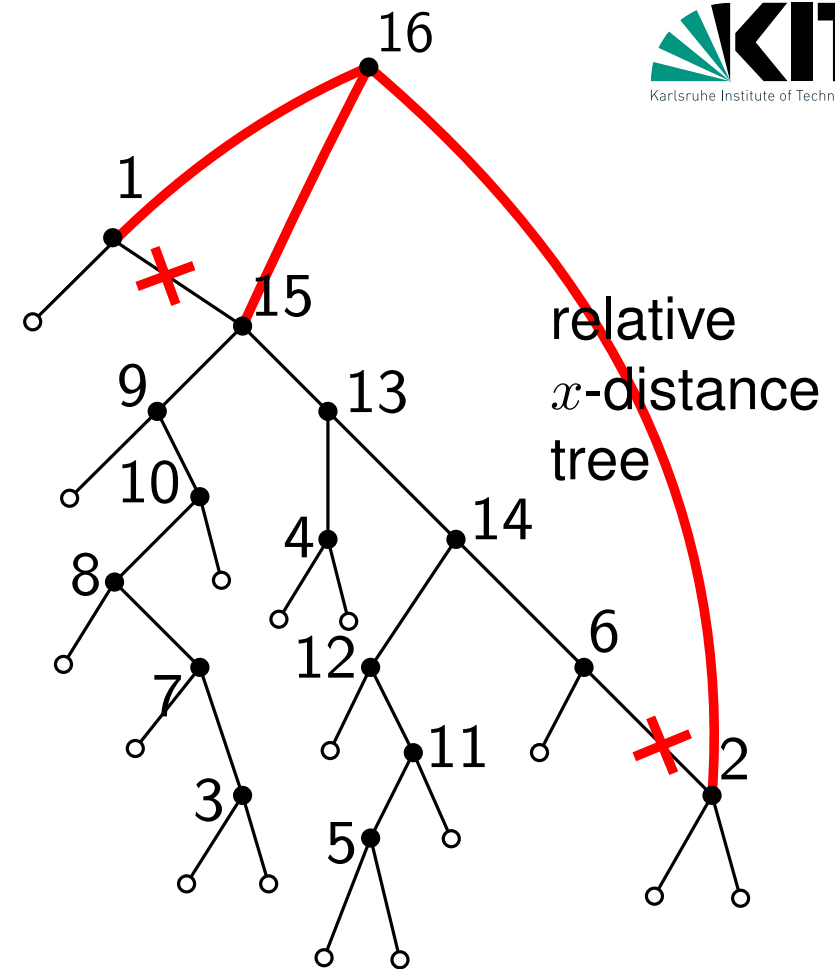
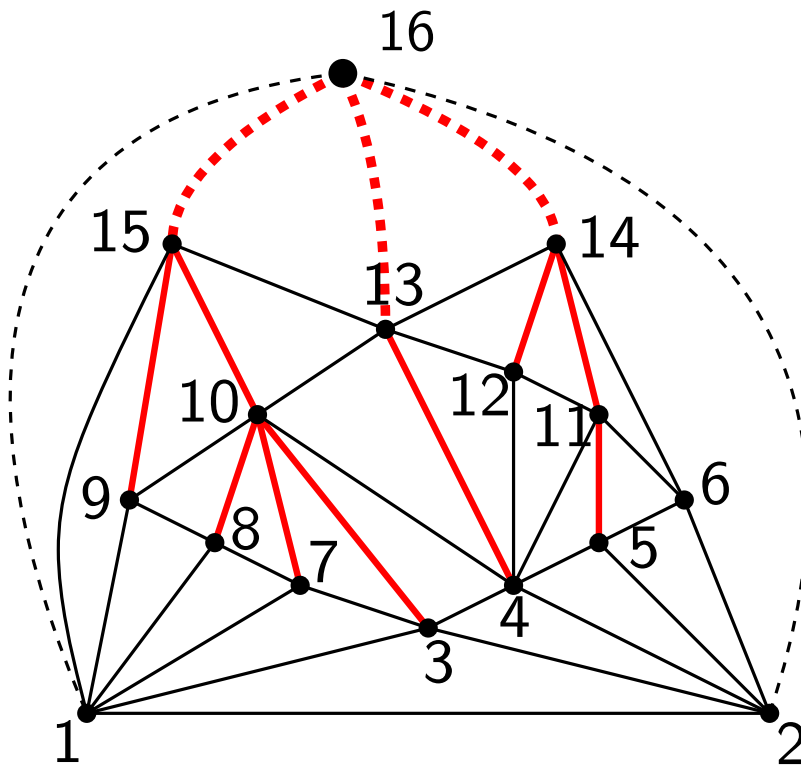
Implementation Details



- In the binary tree at each vertex we keep its relative x -distance from its parent and its y -coordinate

25 - 5

Implementation Details

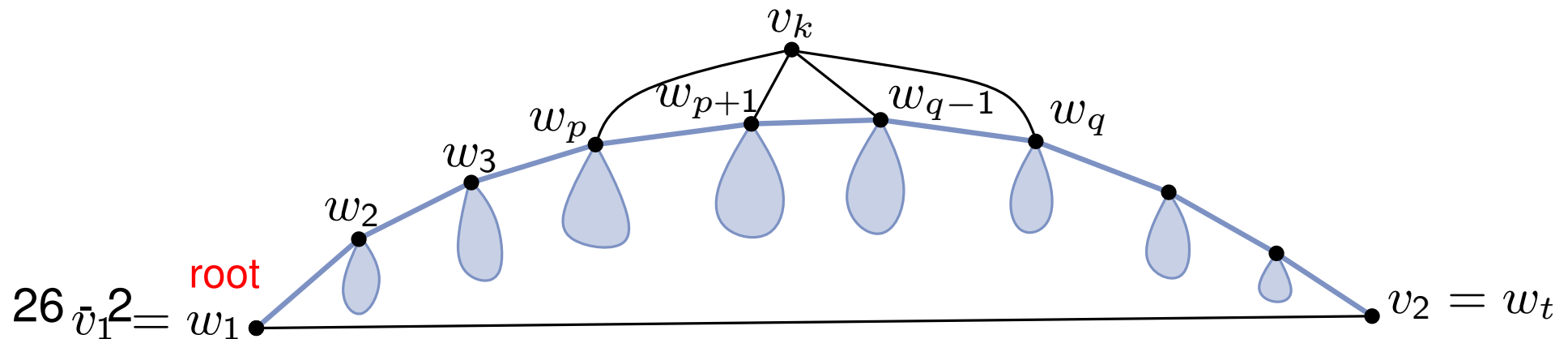


- In the binary tree at each vertex we keep its relative x -distance from its parent and its y -coordinate
- If we know the y -coordinates of w_1 and w_2 and the difference $x(w_1) - x(w_2)$, we can compute the difference $x(v_{16}) - x(w_1)$

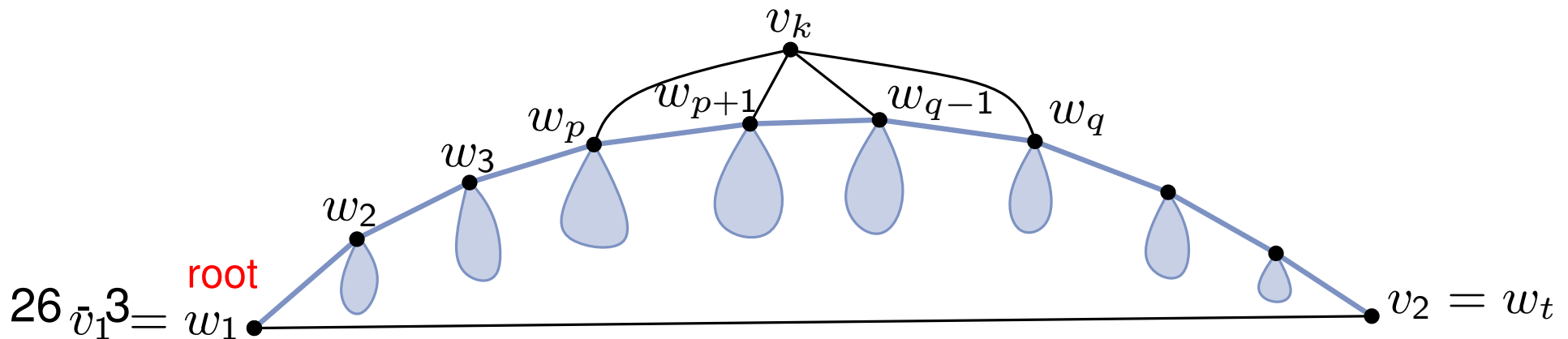
- In the binary tree at each vertex we keep its relative x -distance from its parent and its y -coordinate
- If we know the y -coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$

Implementation Details

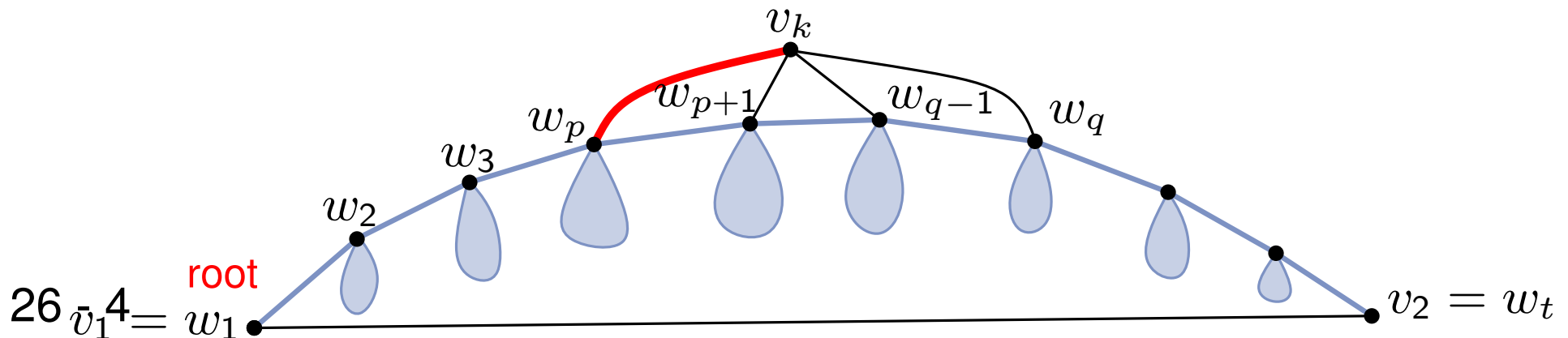
- In the binary tree at each vertex we keep its relative x -distance from its parent and its y -coordinate
- If we know the y -coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$



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

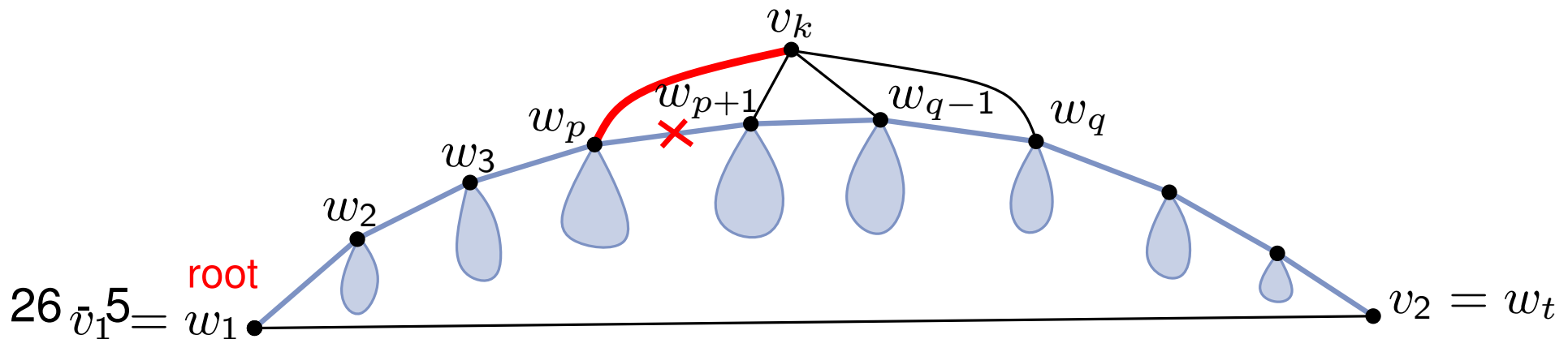


- In the binary tree at each vertex we keep its relative x -distance from its parent and its y -coordinate
- If we know the y -coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \dots + \Delta_x(w_q)$,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

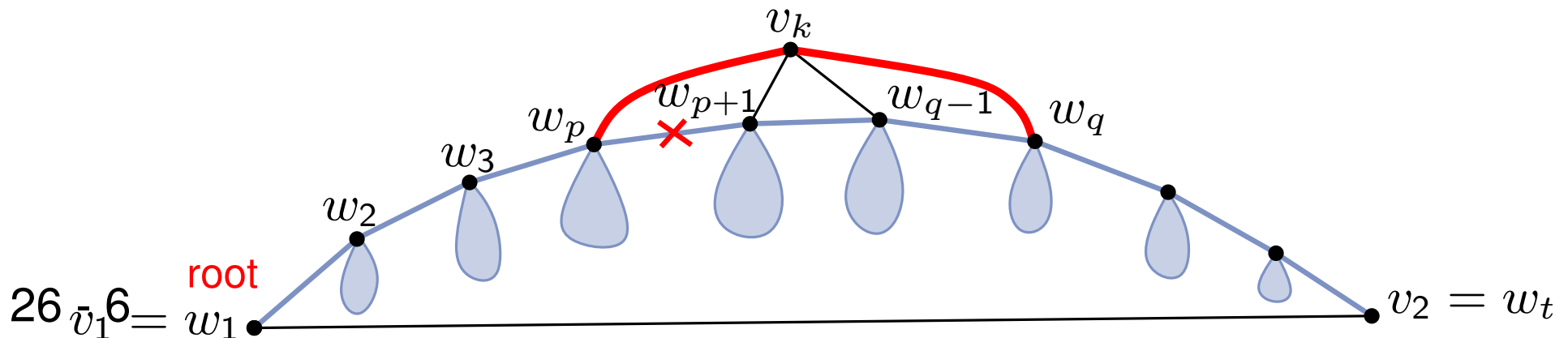


Implementation Details

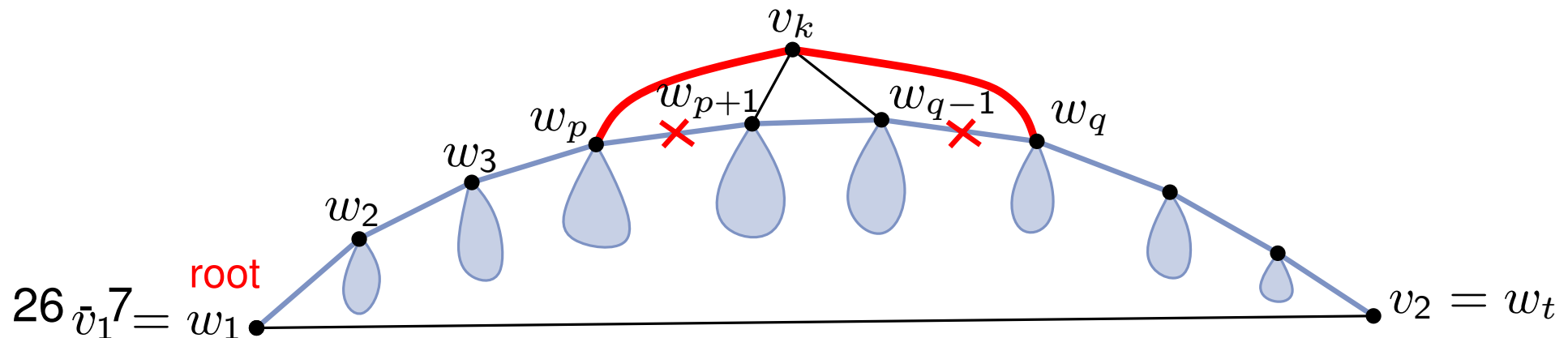
- In the binary tree at each vertex we keep its relative x -distance from its parent and its y -coordinate
- If we know the y -coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \dots + \Delta_x(w_q)$,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



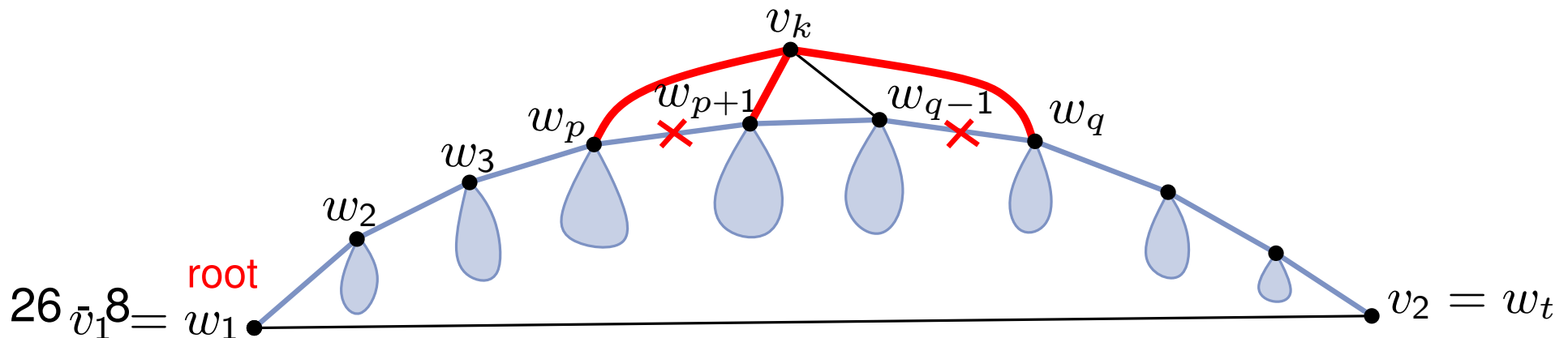
- In the binary tree at each vertex we keep its relative x -distance from its parent and its y -coordinate
- If we know the y -coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \dots + \Delta_x(w_q)$,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



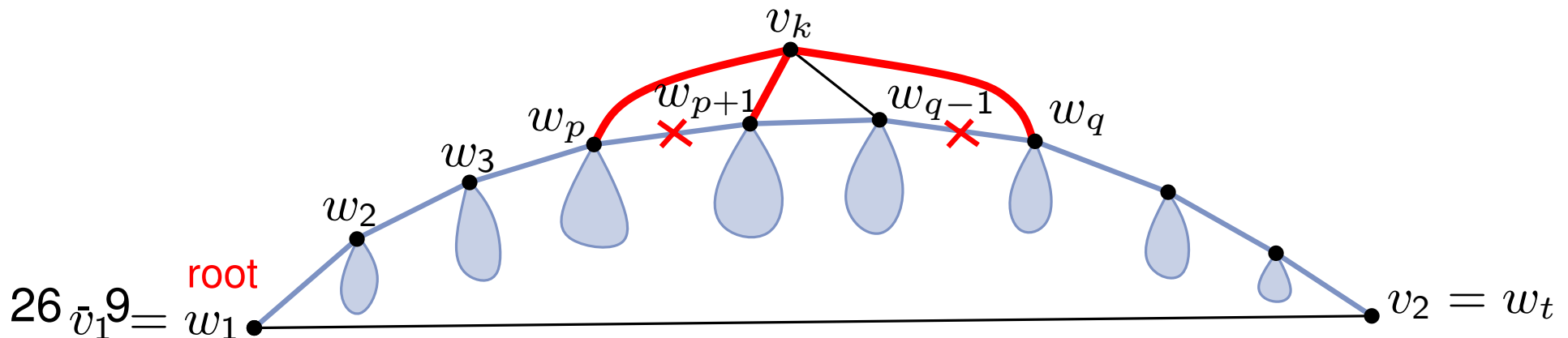
- In the binary tree at each vertex we keep its relative x -distance from its parent and its y -coordinate
- If we know the y -coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \dots + \Delta_x(w_q)$,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



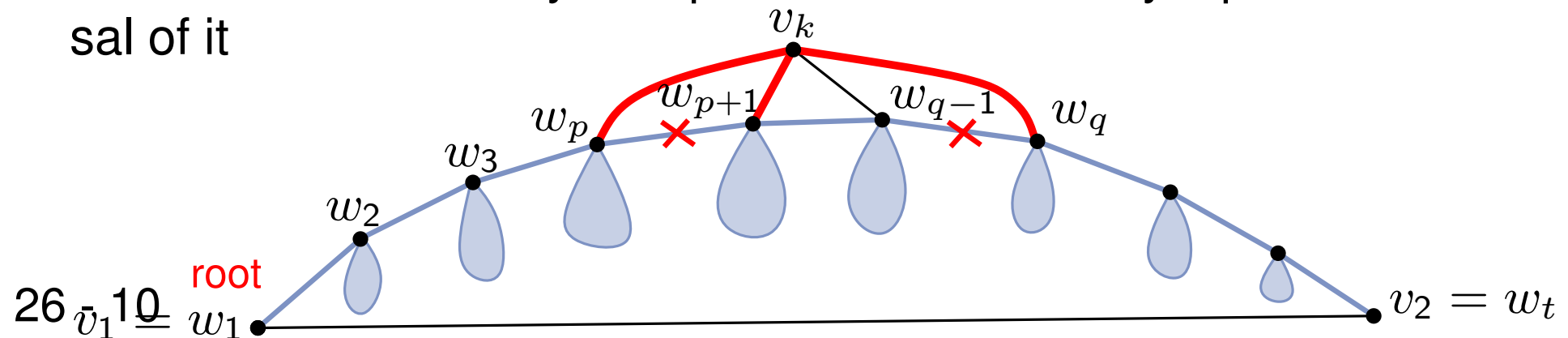
- In the binary tree at each vertex we keep its relative x -distance from its parent and its y -coordinate
- If we know the y -coordinates of w_p and w_q and the difference $x(w_p) - x(w_q)$, we can compute the difference $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \dots + \Delta_x(w_q)$,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



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



- 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 $x(v_k) - x(w_p)$
- $\Delta_x(w_p, w_q) = \Delta_x(w_{p+1}) + \dots + \Delta_x(w_q)$, 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)
- When the tree is ready, compute x-coordinates by a pre-order traversal of it



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.

27 - 1

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

27 - 2