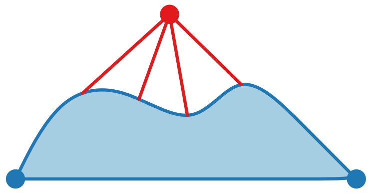
Visualisation of graphs

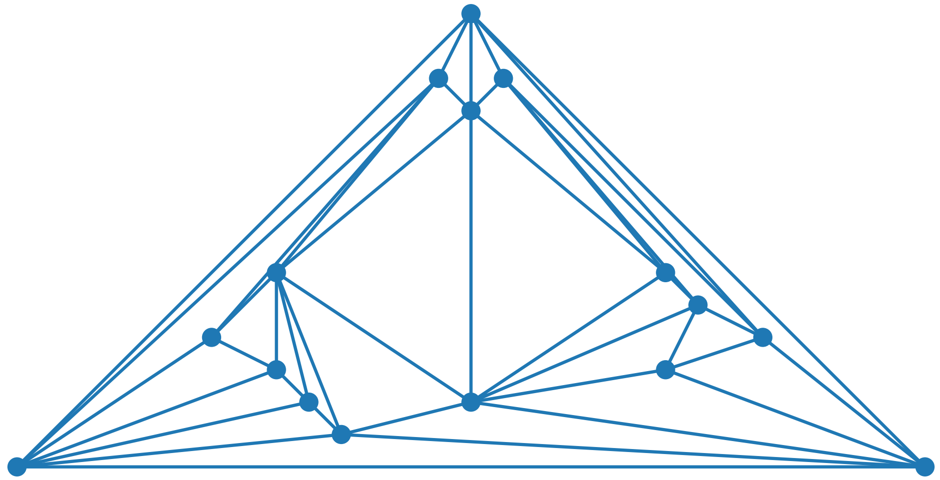
Planar straight-line drawings

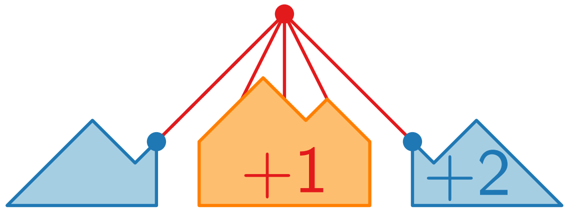
Shift Method

Antonios Symvonis · Chrysanthi Raftopoulou

Fall semester 2020







The original slides of this presentation were created by researchers at Karlsruhe Institute of Technology (KIT), TU Wien, U Wuerzburg, U Konstanz, ...
The original presentation was modified/updated by A. Symvonis and C. Raftopoulou

Planar straight-line drawings

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

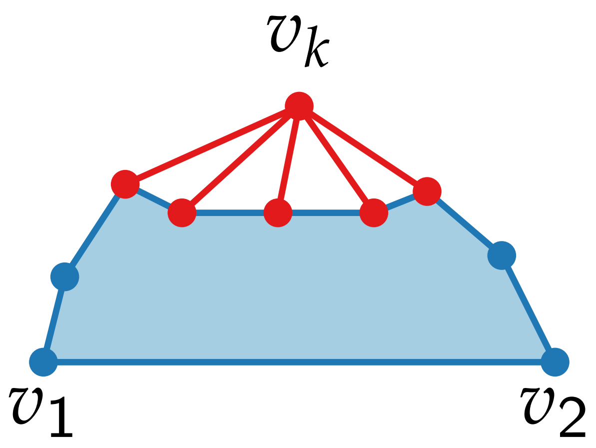
Theorem. [Schnyder '90] Every n-vertex planar graph has a planar straight-line drawing of size $(n-2) \times (n-2)$.

Planar straight-line drawings

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

Idea: Use the canonical order.

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



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

Canonical order – definition

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

Canonical order – definition

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

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

Canonical order – definition

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

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

Canonical order – definition

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

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

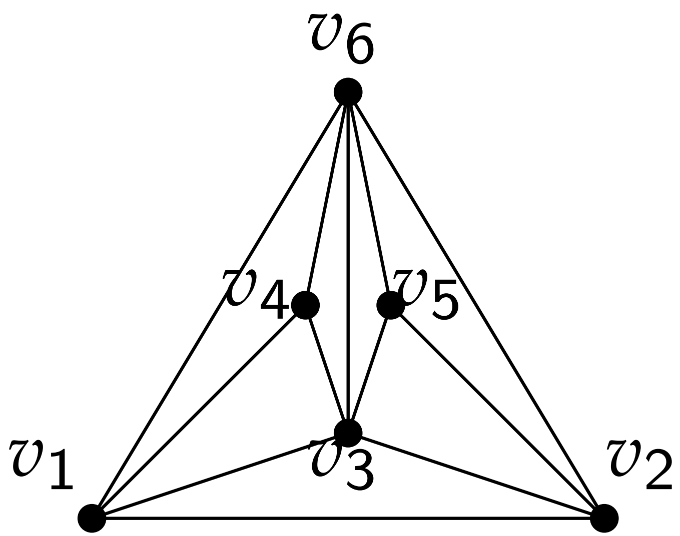
Canonical order – definition

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

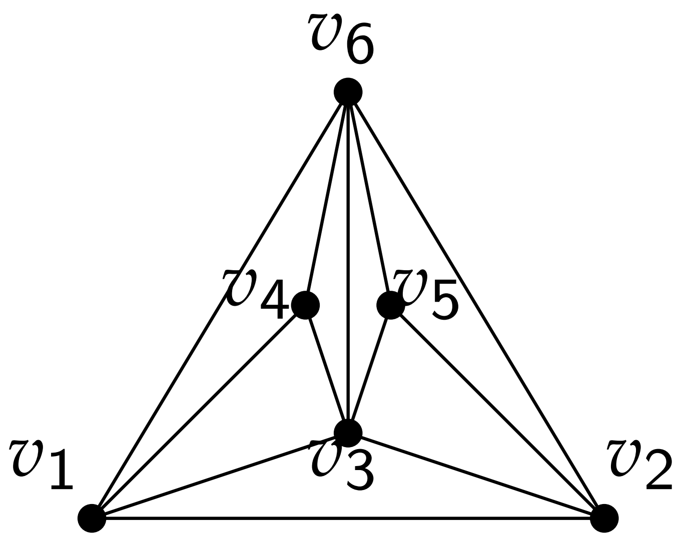
- (C1) Vertices $\{v_1, \ldots v_k\}$ induce a biconnected internally triangulated graph; call it G_k.
- (C2) Edge (v_1, v_2) belongs to the outer face of G_k.
- (C3) If $k < n$ then vertex v_{k+1} lies in the outer face of G_k, and all neighbors of v_{k+1} in G_k appear on the boundary of G_k consecutively.

Lemma.
Every triangulated plane graph has a canonical order.

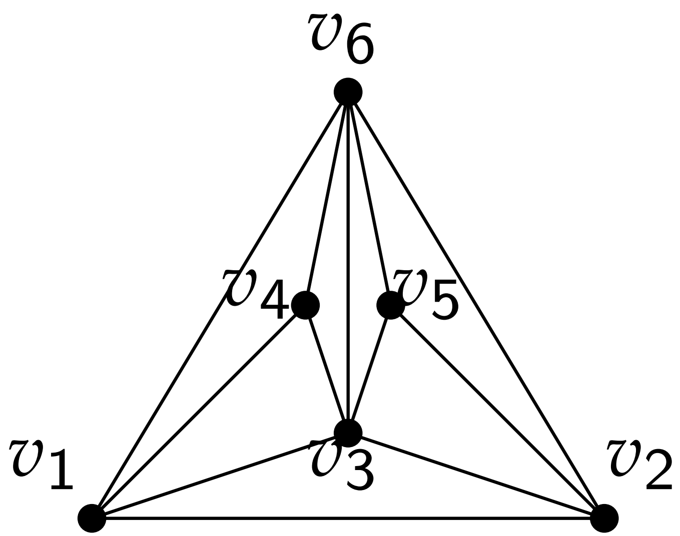
Constraints

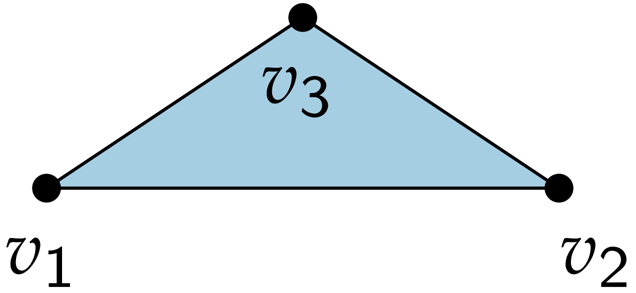


Constraints

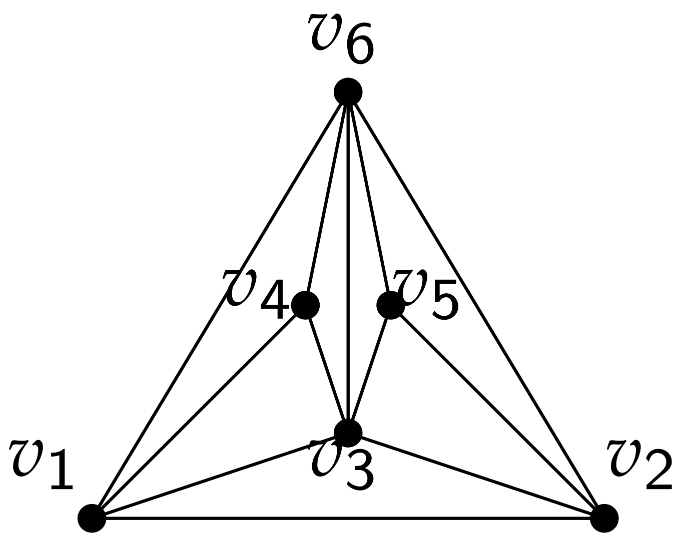


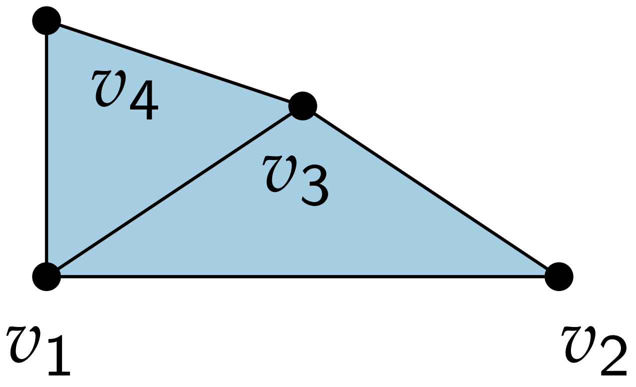
Constraints



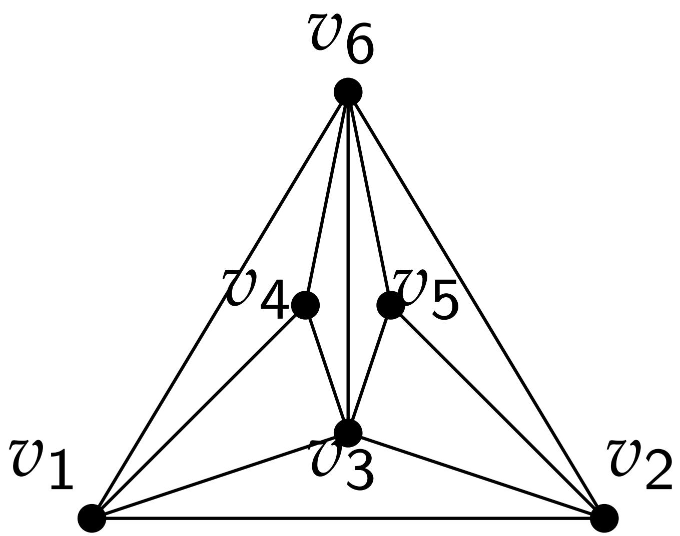


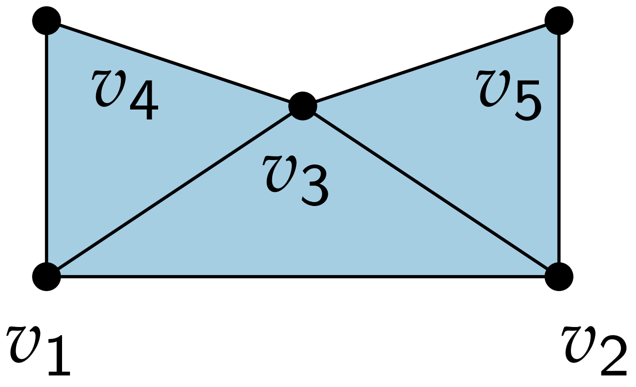
Constraints



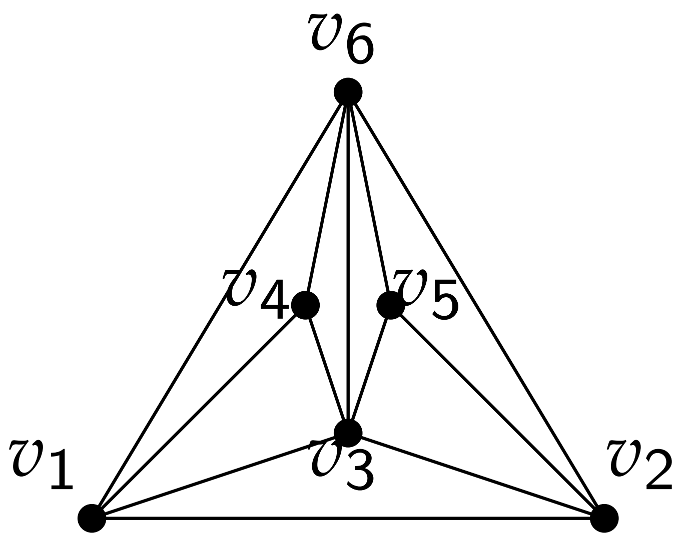


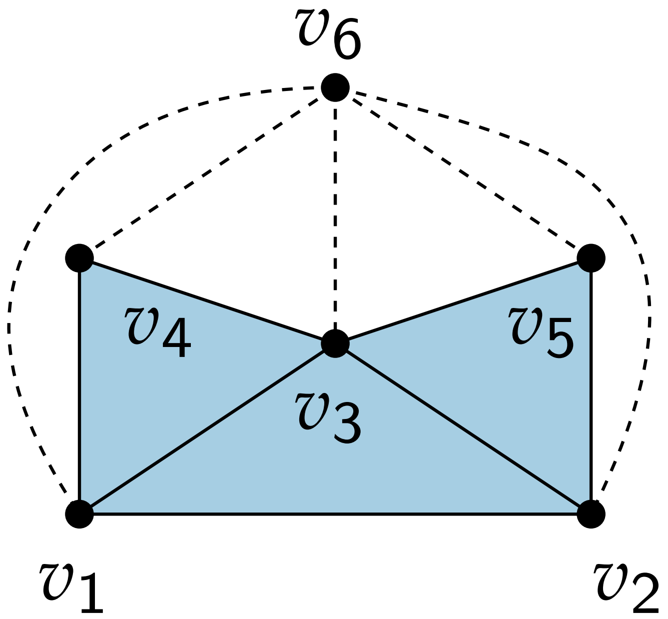
Constraints



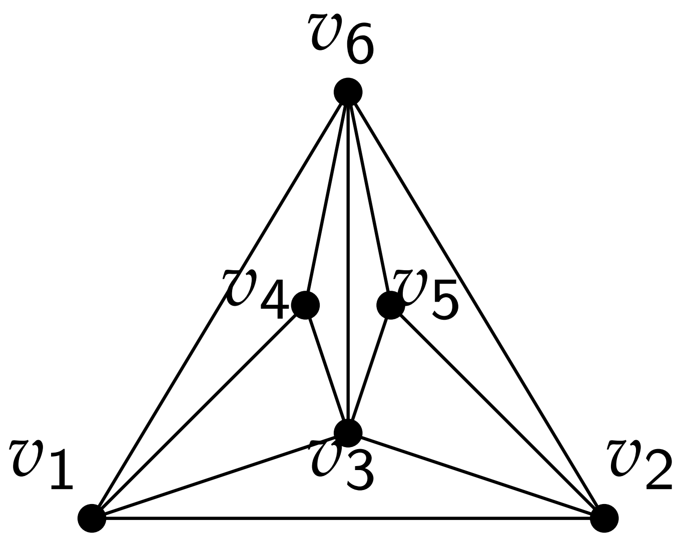


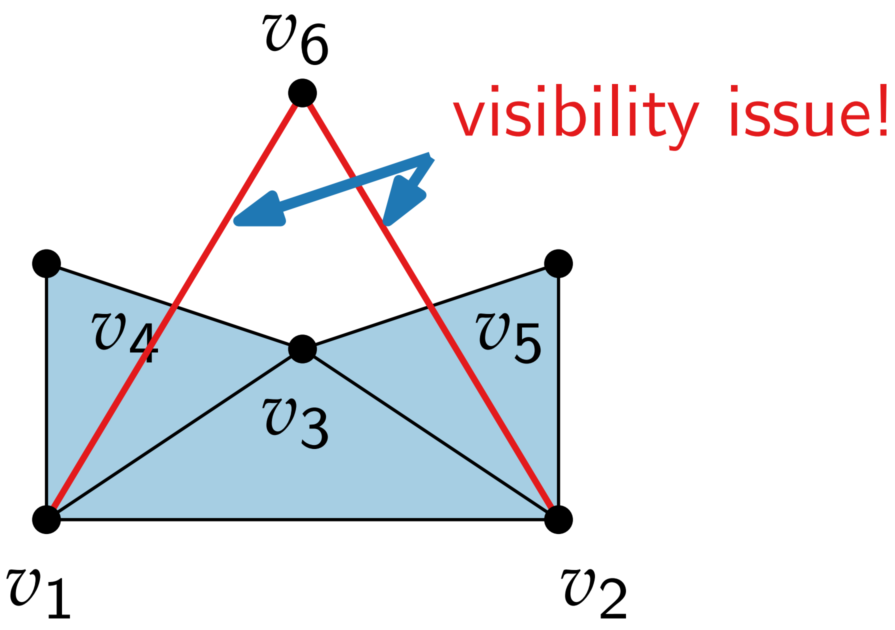
Constraints



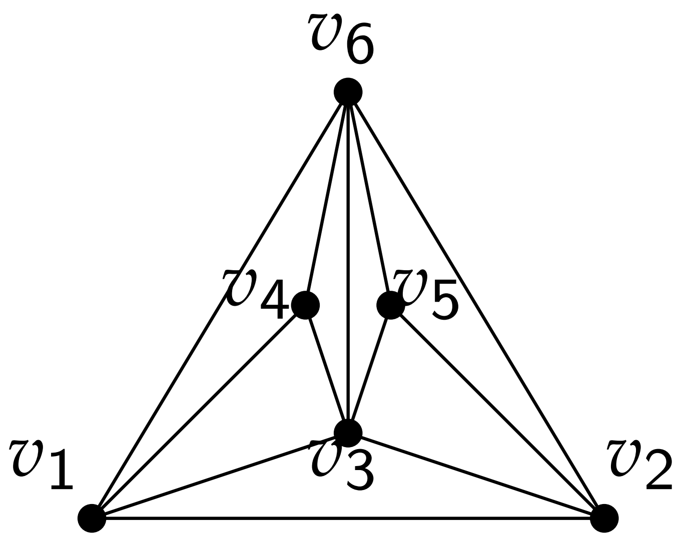


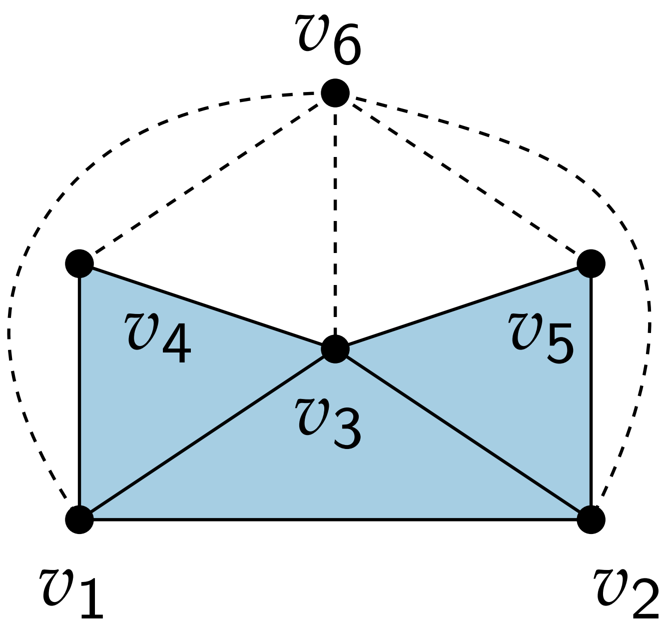
Constraints





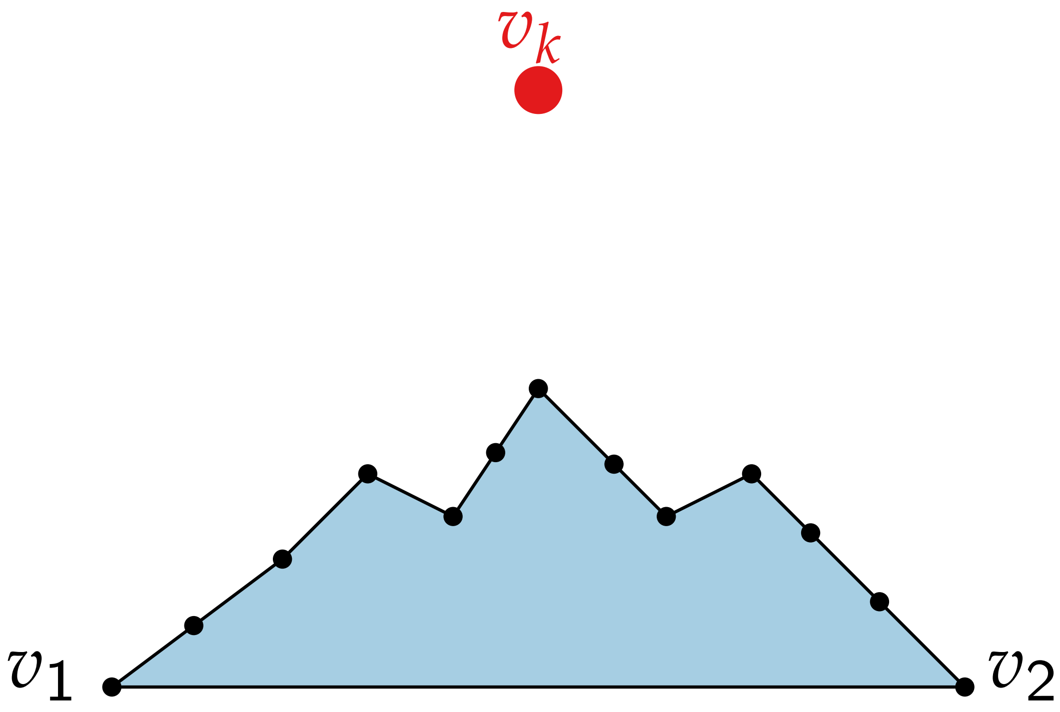
Constraints



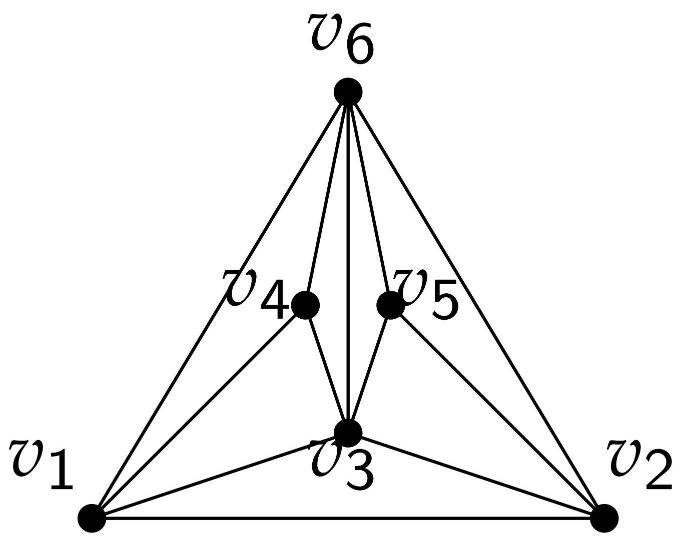


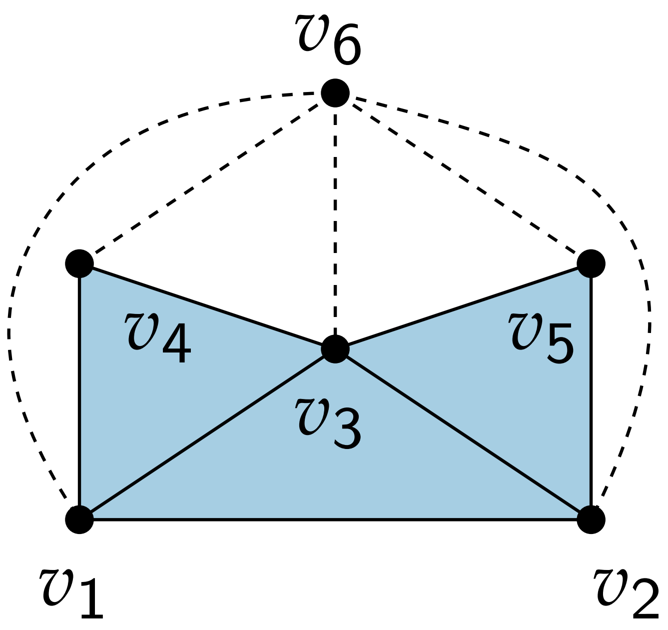
Constraints:

G_{k-1} is drawn such that



Constraints

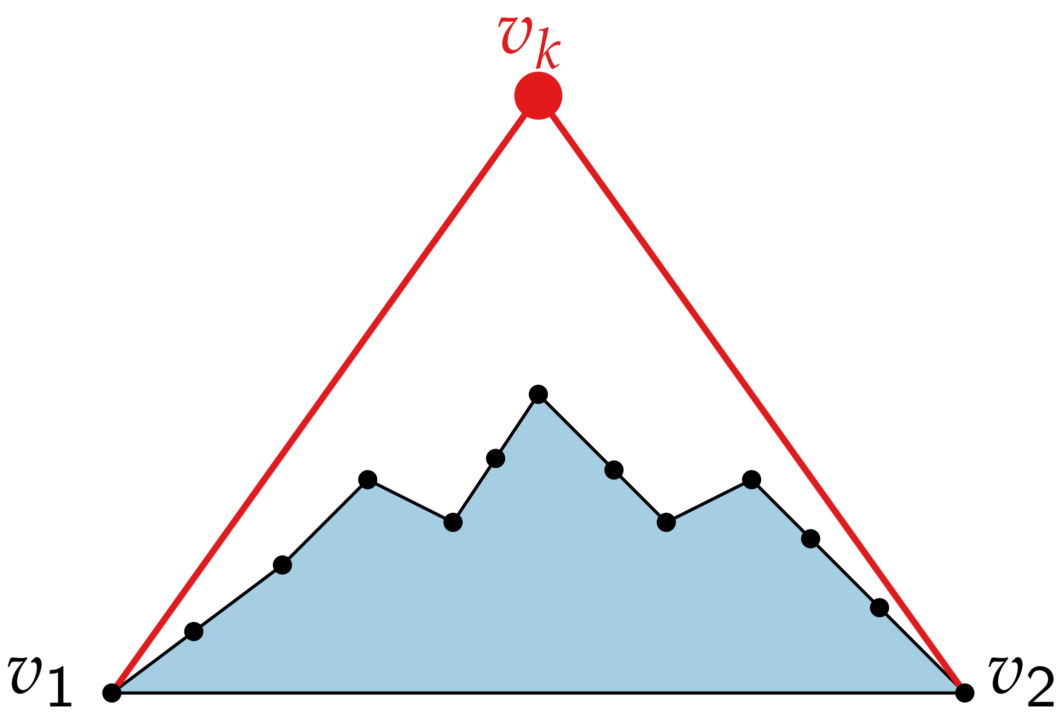




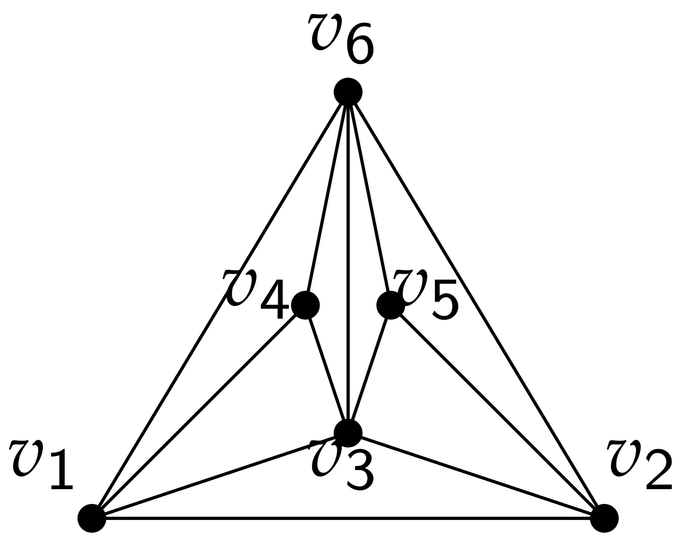
Constraints:

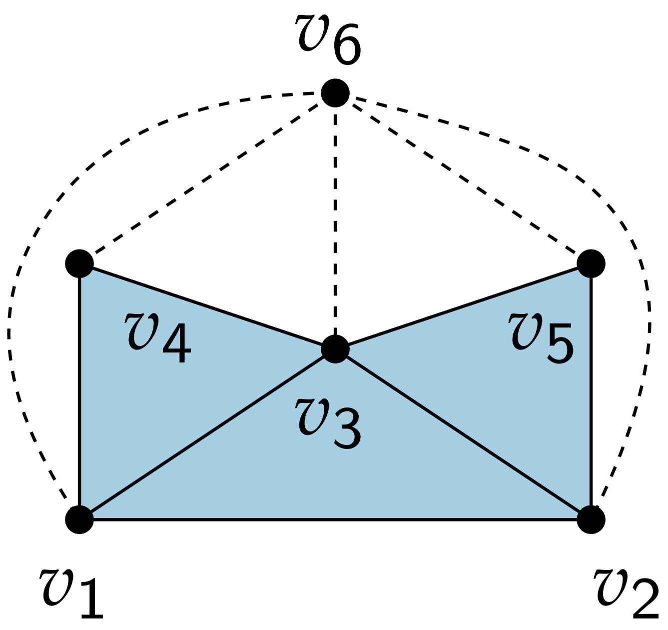
G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,



Constraints

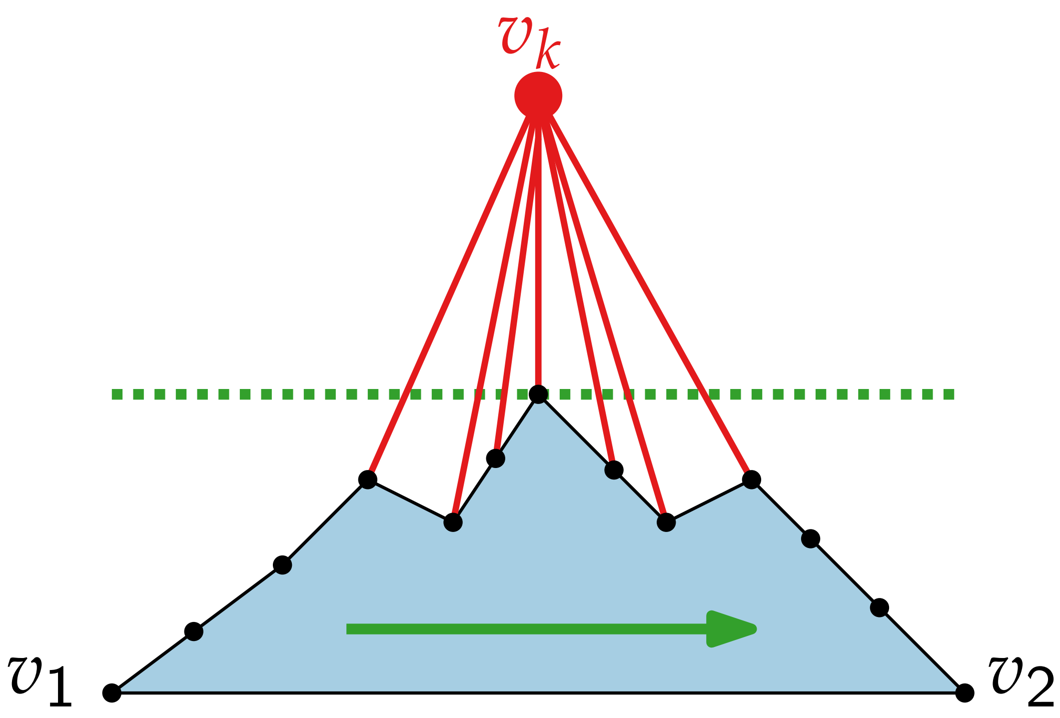




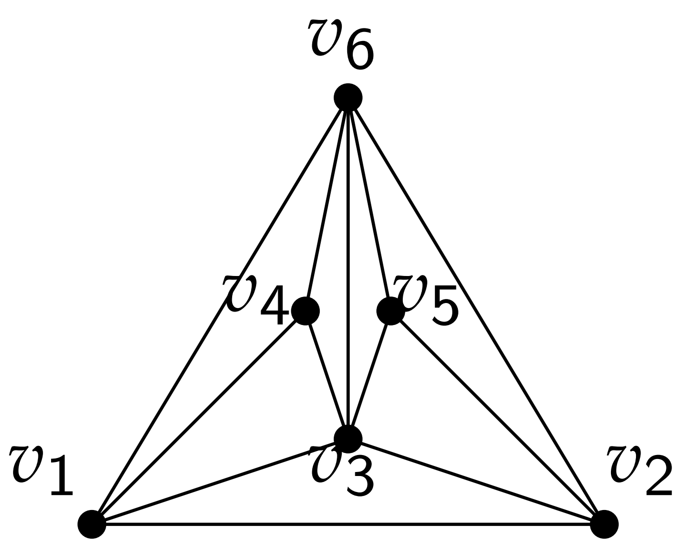
Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- neighbors of v_k on G_{k-1} should be drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.



Constraints

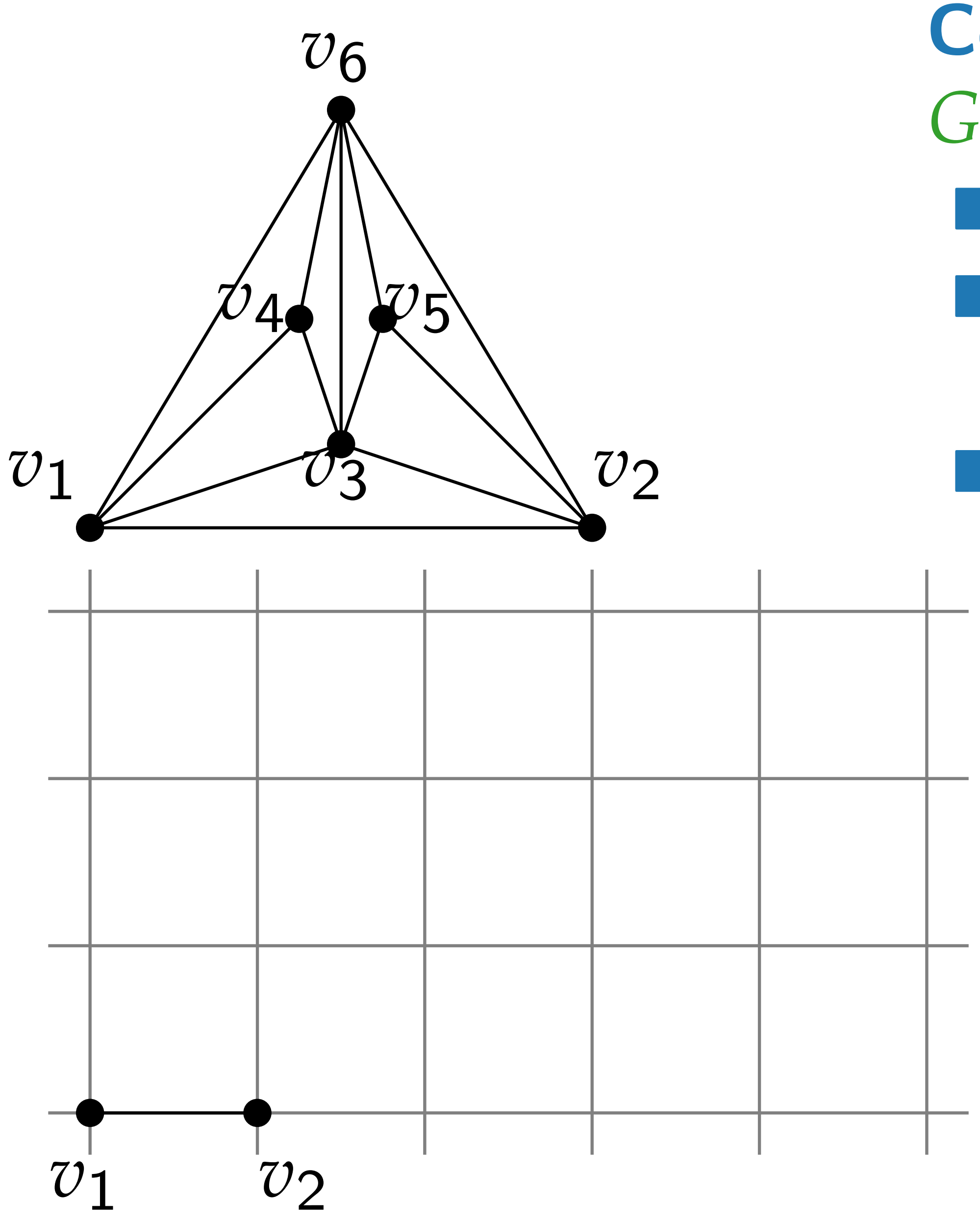


Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

Constraints



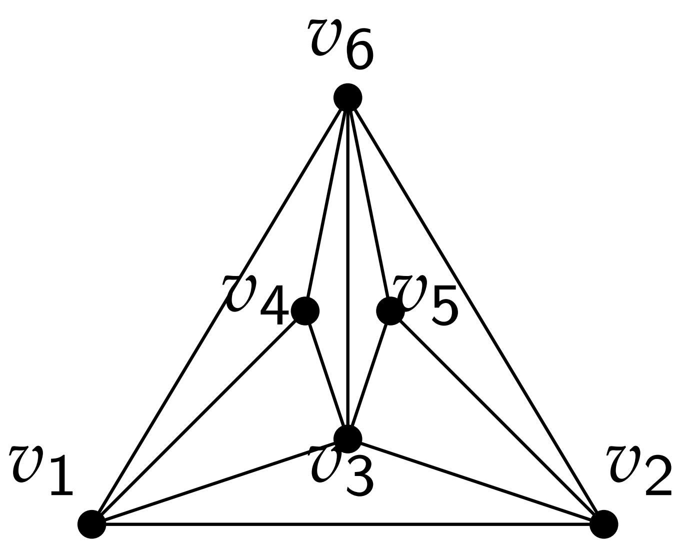
Constraints:

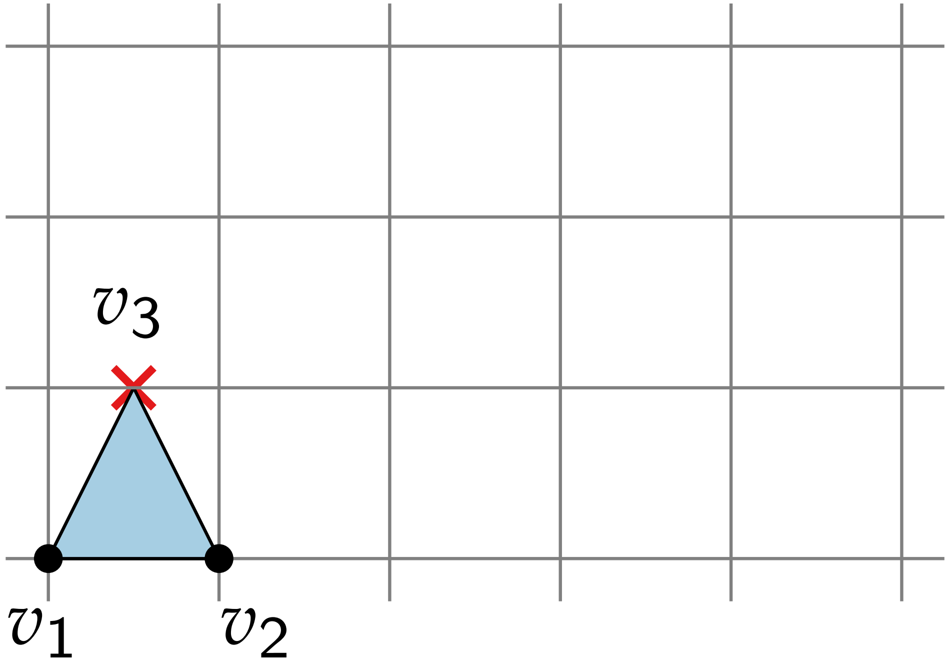
G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

G_2: $v_1 : (0, 0)$, $v_2 : (1, 0)$

Constraints





Constraints:

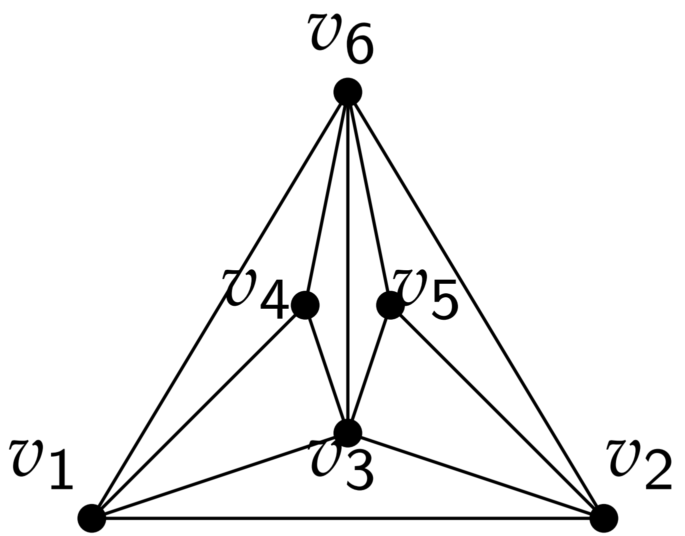
G_{k-1} is drawn such that

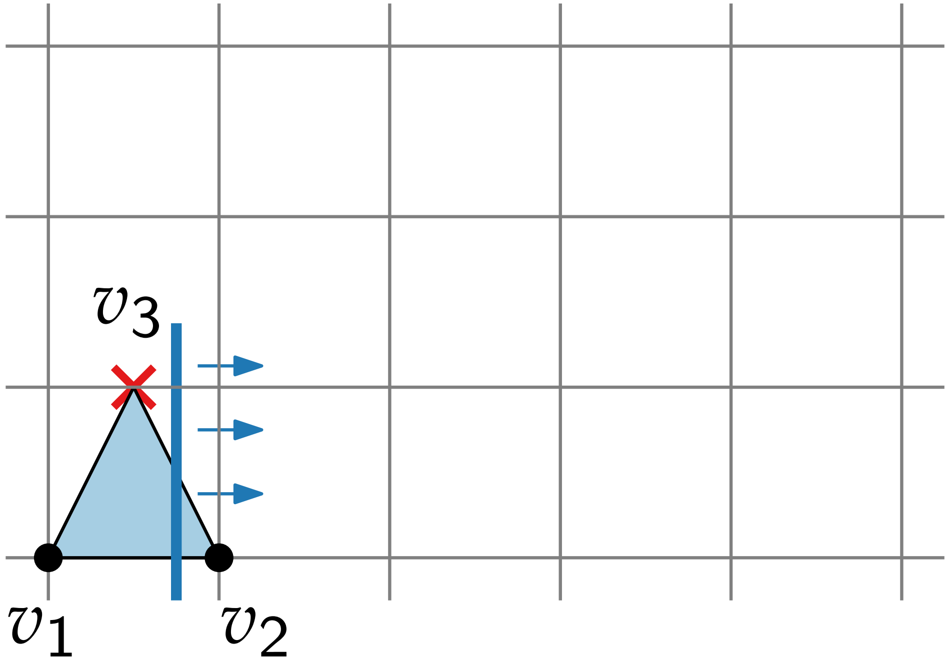
- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

G_2: $v_1 : (0, 0)$, $v_2 : (1, 0)$

- Need to make room for v_3

Constraints





Constraints:

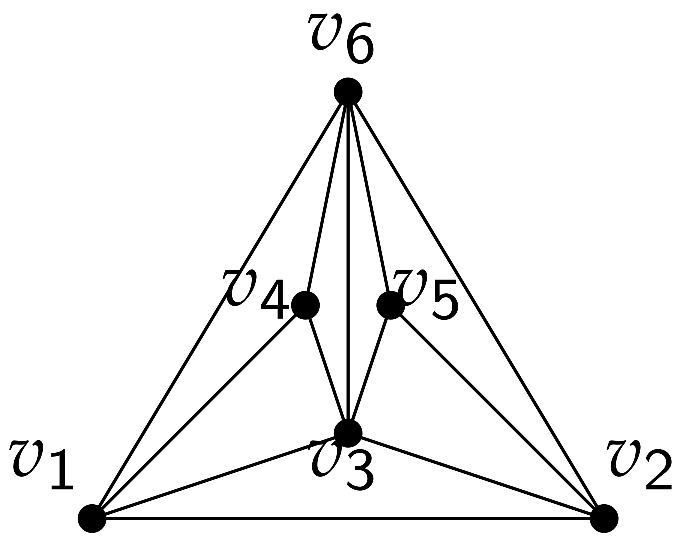
G_{k-1} is drawn such that

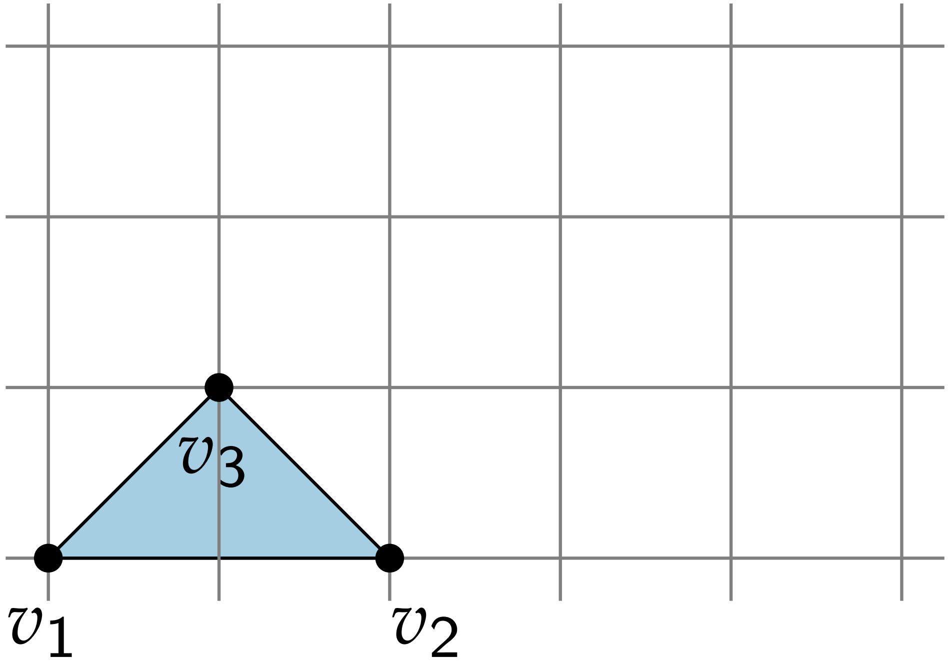
- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

G_2: $v_1 : (0, 0)$, $v_2 : (1, 0)$

- Need to make room for v_3
- **Shift** v_2 to the right

Constraints





Constraints:

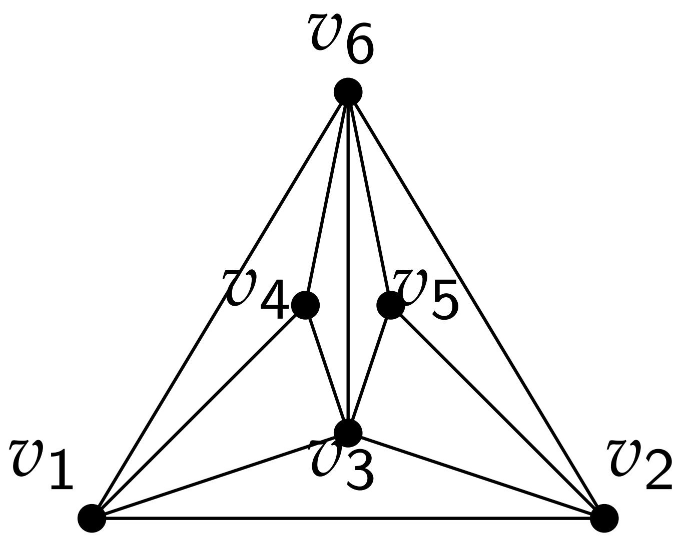
G_{k-1} is drawn such that

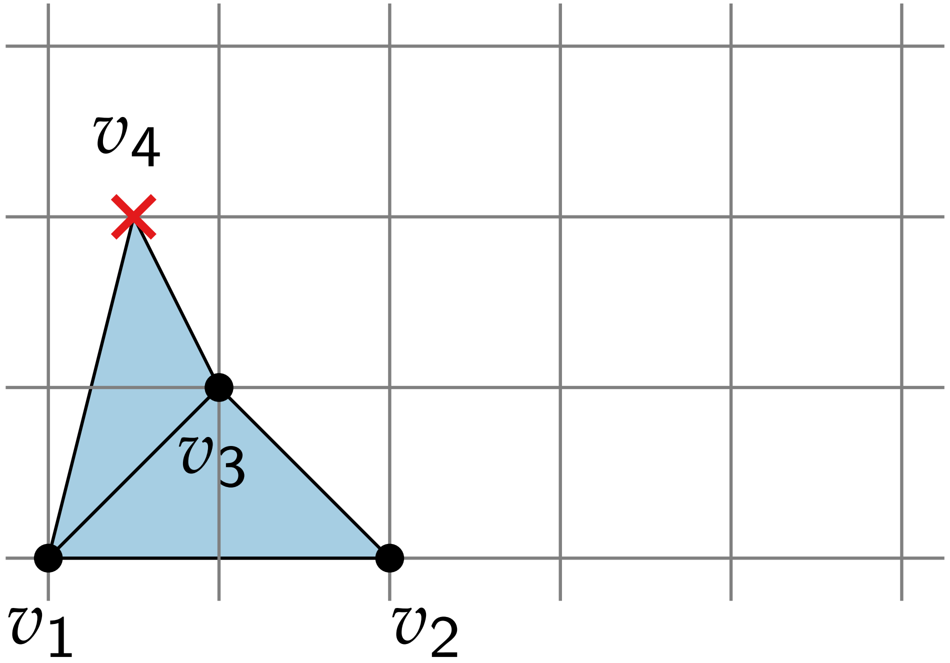
- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

G_2: $v_1 : (0, 0)$, $v_2 : (1, 0)$ ~~(1, 0)~~

G_3: $v_1 : (0, 0)$, $v_2 : (2, 0)$, $v_3 : (1, 1)$

Constraints





Constraints:

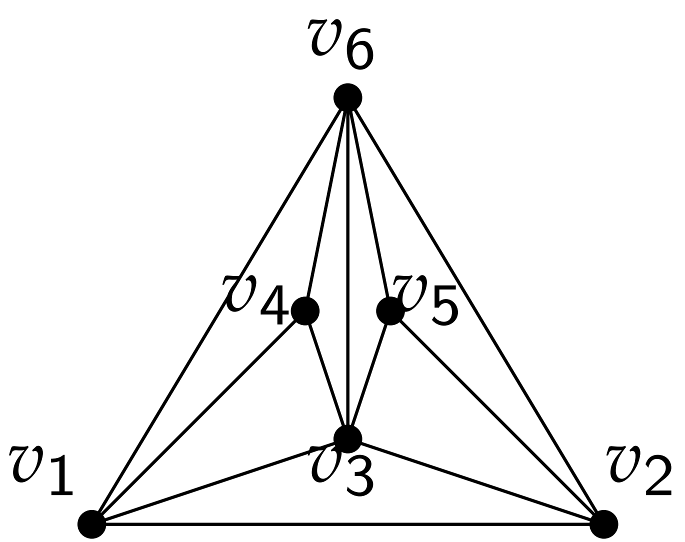
G_{k-1} is drawn such that

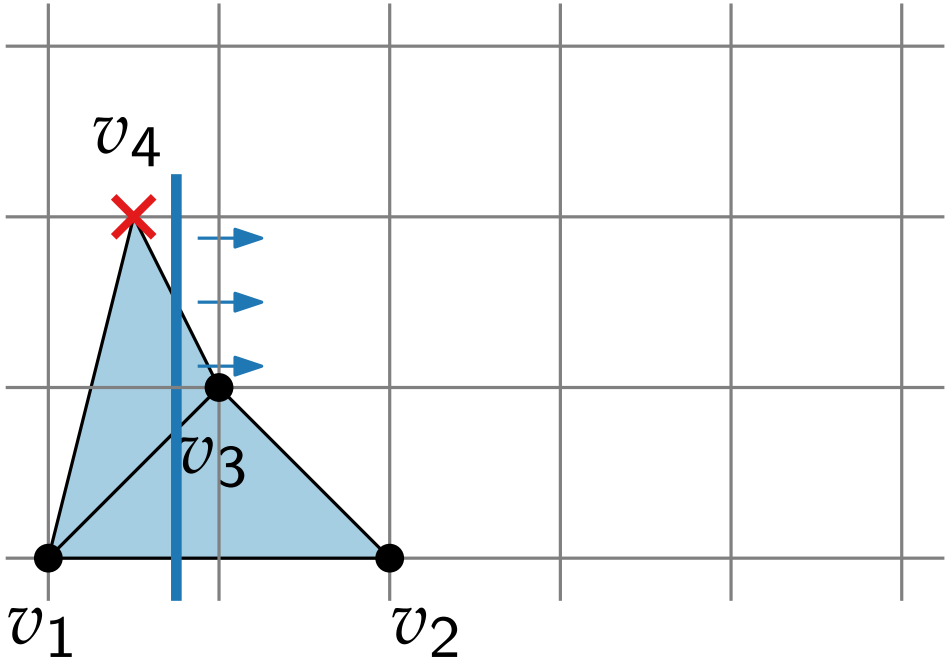
- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

G_2: $v_1 : (0,0)$, $v_2 : (1,0)$

G_3: $v_1 : (0,0)$, $v_2 : (2,0)$, $v_3 : (1,1)$

Constraints





Constraints:

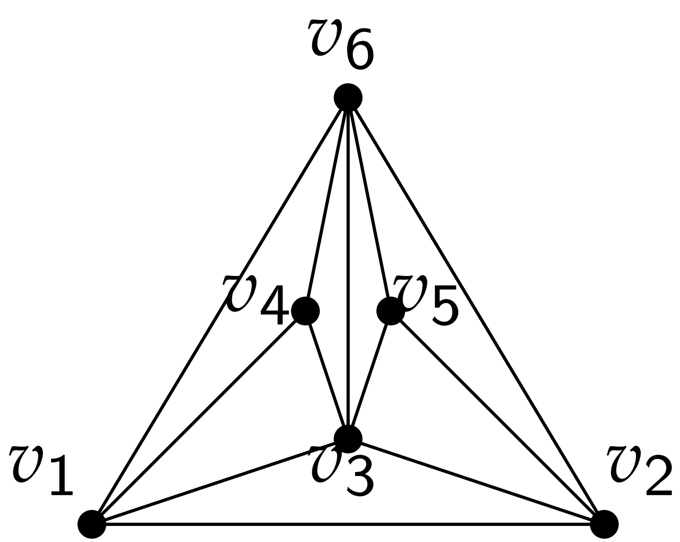
G_{k-1} is drawn such that

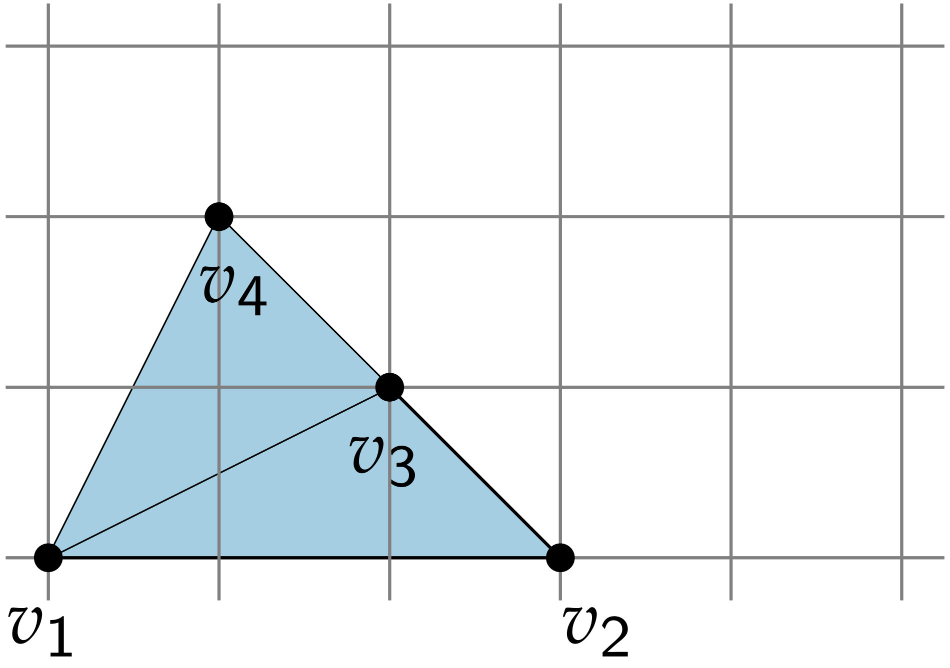
- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

G_2: $v_1 : (0, 0)$, $v_2 : (1, 0)$

G_3: $v_1 : (0, 0)$, $v_2 : (2, 0)$, $v_3 : (1, 1)$

Constraints





Constraints:

G_{k-1} is drawn such that

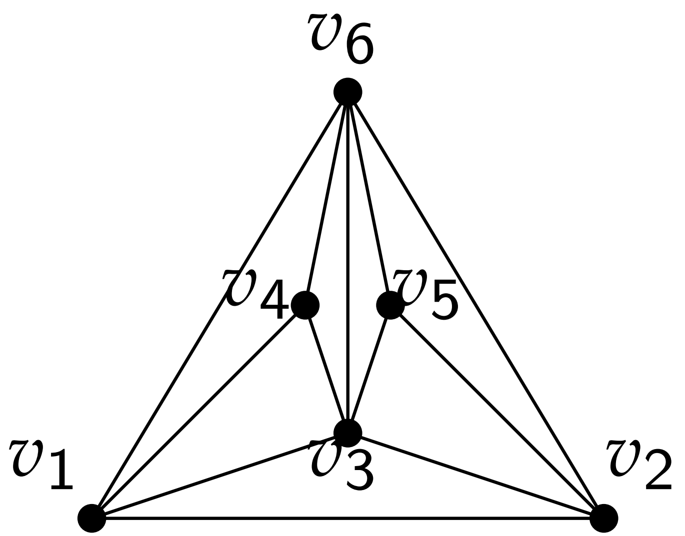
- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

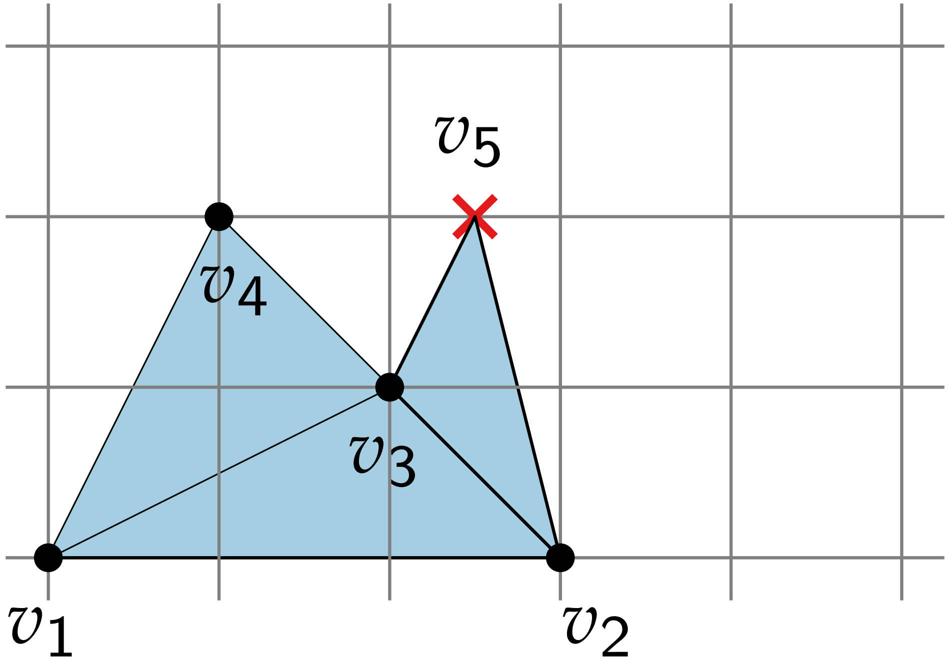
G_2: $v_1 : (0, 0)$, $v_2 :$ ~~$(1, 0)$~~

G_3: $v_1 : (0, 0)$, $v_2 :$ ~~$(2, 0)$~~, $v_3 :$ ~~$(1, 1)$~~

G_4: $v_1 : (0, 0)$, $v_2 : (3, 0)$, $v_3 : (2, 1)$, $v_4 : (1, 2)$

Constraints





Constraints:

G_{k-1} is drawn such that

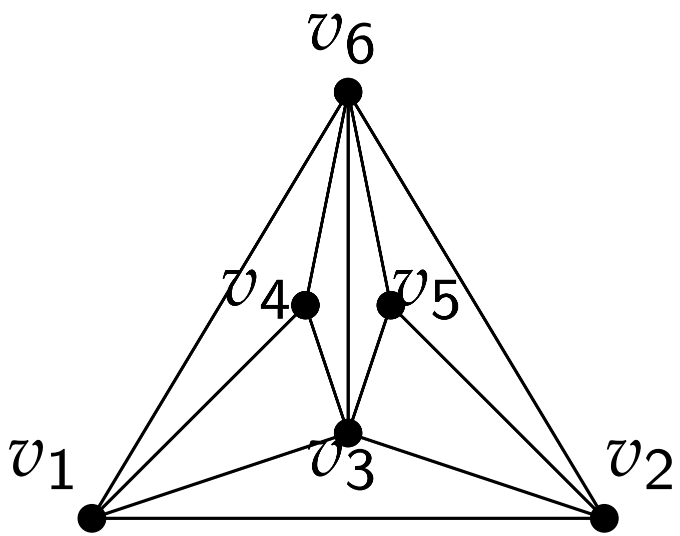
- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

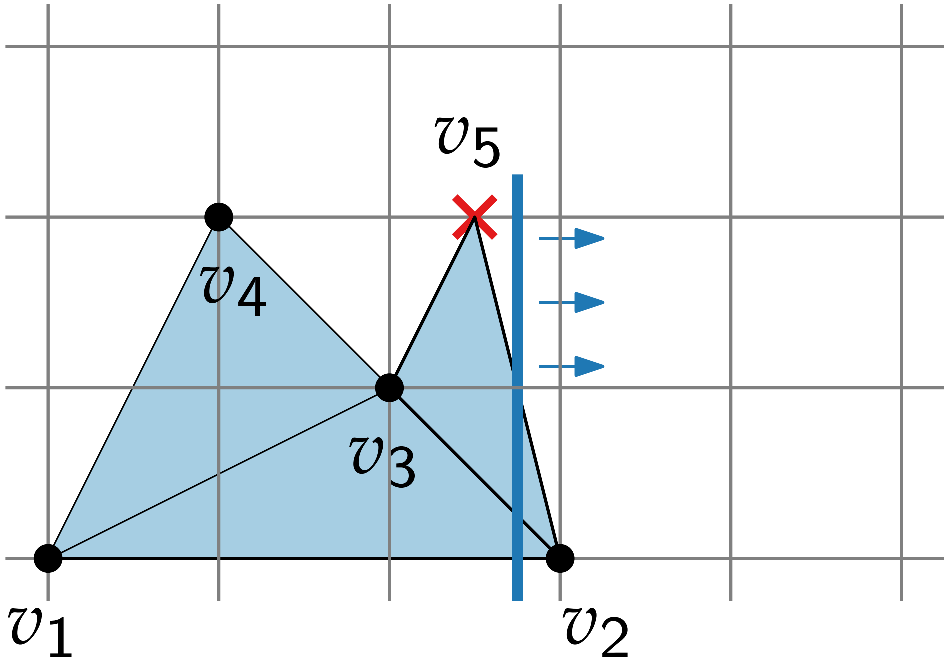
G_2: $v_1 : (0, 0)$, $v_2 :$ ~~$(1, 0)$~~

G_3: $v_1 : (0, 0)$, $v_2 :$ ~~$(2, 0)$~~, $v_3 :$ ~~$(1, 1)$~~

G_4: $v_1 : (0, 0)$, $v_2 : (3, 0)$, $v_3 : (2, 1)$, $v_4 : (1, 2)$

Constraints





Constraints:

G_{k-1} is drawn such that

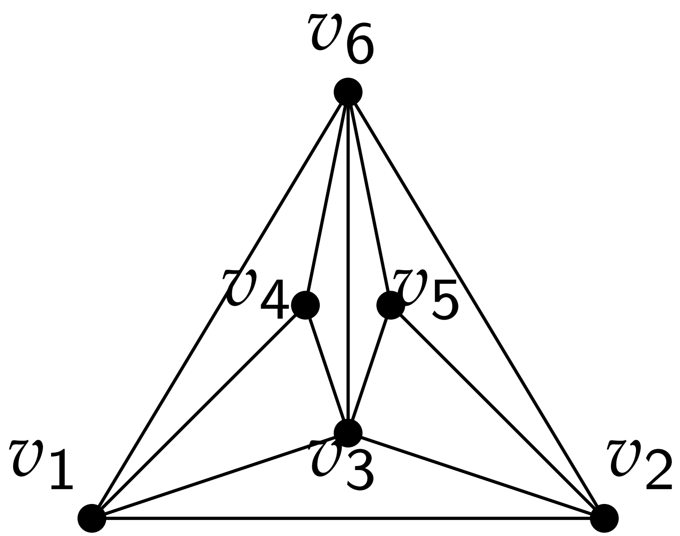
- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

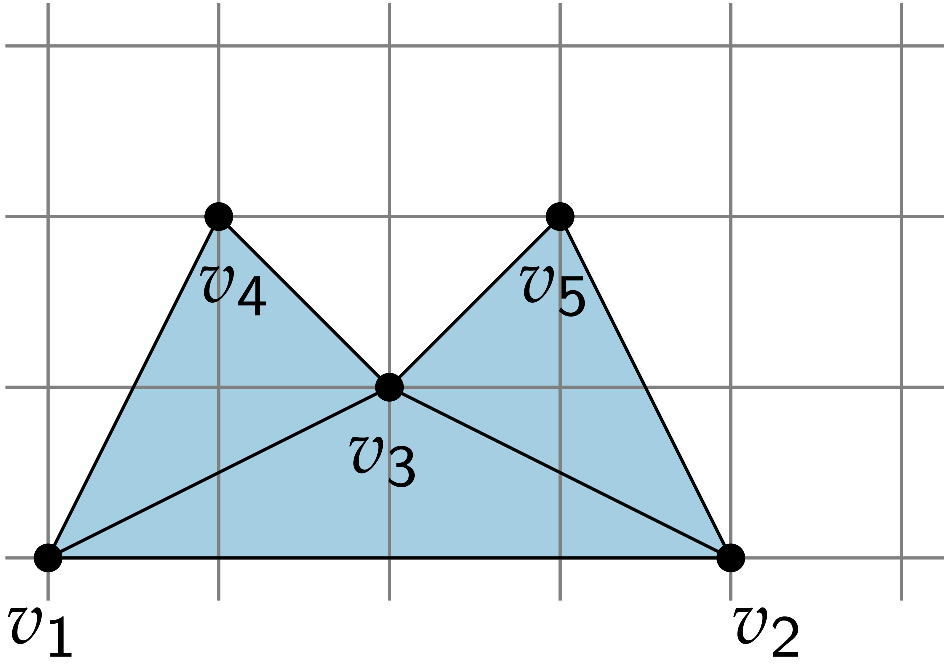
G_2: $v_1 : (0, 0)$, $v_2 :$ ~~$(1, 0)$~~

G_3: $v_1 : (0, 0)$, $v_2 :$ ~~$(2, 0)$~~, $v_3 :$ ~~$(1, 1)$~~

G_4: $v_1 : (0, 0)$, $v_2 : (3, 0)$, $v_3 : (2, 1)$, $v_4 : (1, 2)$

Constraints





Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

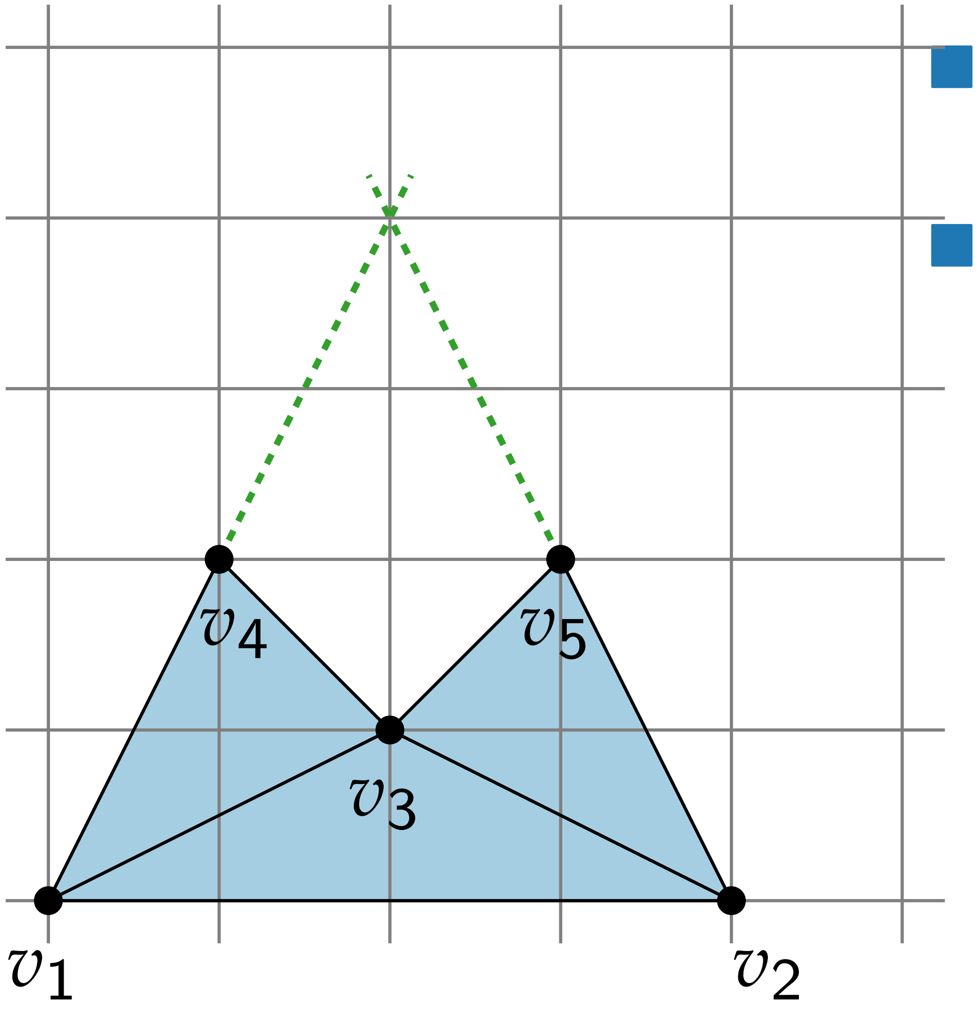
G_2: $v_1 : (0, 0)$, $v_2 :$ ~~$(1, 0)$~~

G_3: $v_1 : (0, 0)$, $v_2 :$ ~~$(2, 0)$~~, $v_3 :$ ~~$(1, 1)$~~

G_4: $v_1 : (0, 0)$, $v_2 : (3, 0)$, $v_3 : (2, 1)$, $v_4 : (1, 2)$

G_5: $v_1 : (0, 0)$, $v_2 :$ ~~$(4, 0)$~~, $v_3 : (2, 1)$, $v_4 : (1, 2)$, $v_5 : (3, 2)$

Constraints



Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

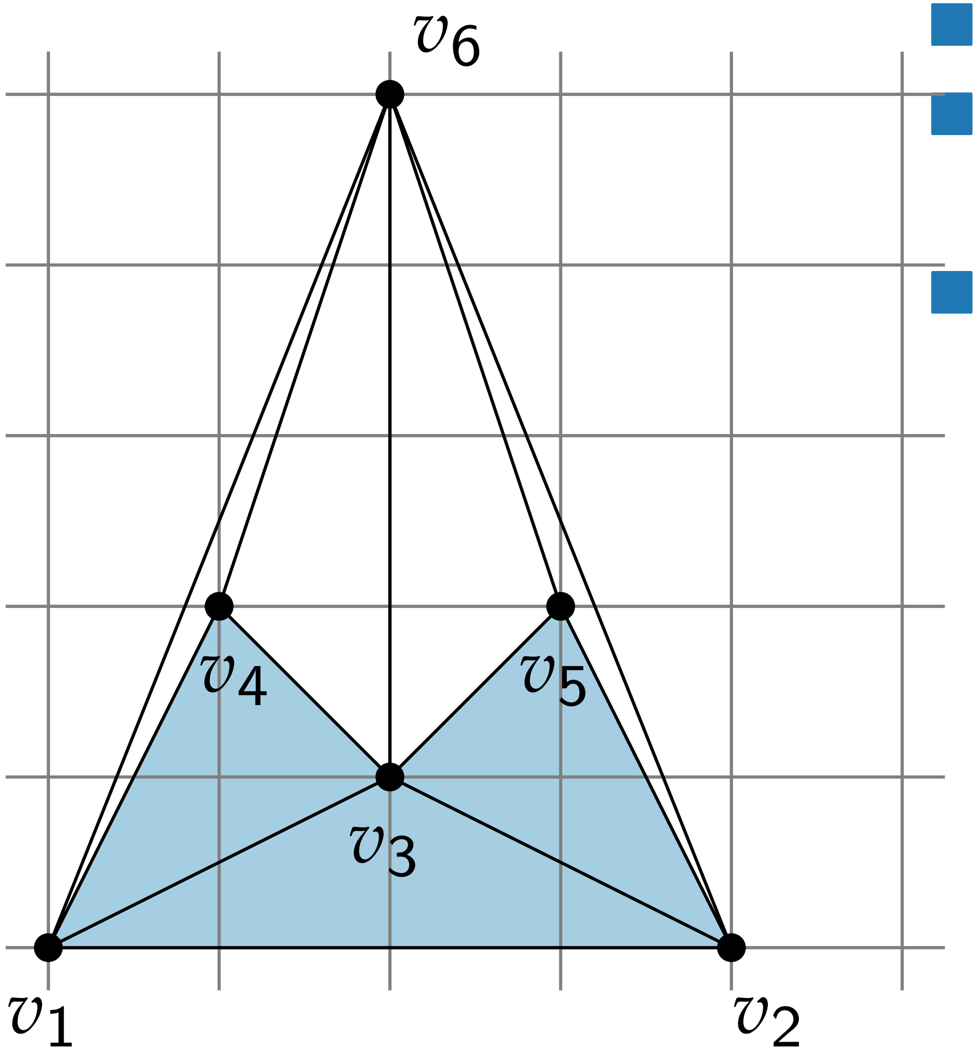
G_2: $v_1 : (0,0)$, $v_2 :$ ~~$(1,0)$~~

G_3: $v_1 : (0,0)$, $v_2 :$ ~~$(2,0)$~~, $v_3 :$ ~~$(1,1)$~~

G_4: $v_1 : (0,0)$, $v_2 : (3,0)$, $v_3 : (2,1)$, $v_4 : (1,2)$

G_5: $v_1 : (0,0)$, $v_2 :$ ~~$(4,0)$~~, $v_3 : (2,1)$, $v_4 : (1,2)$, $v_5 : (3,2)$

Constraints



Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- v_k is placed above its neighbors on G_{k-1}.

G_2: $v_1 : (0,0)$, $v_2 : ~~(1,0)~~$

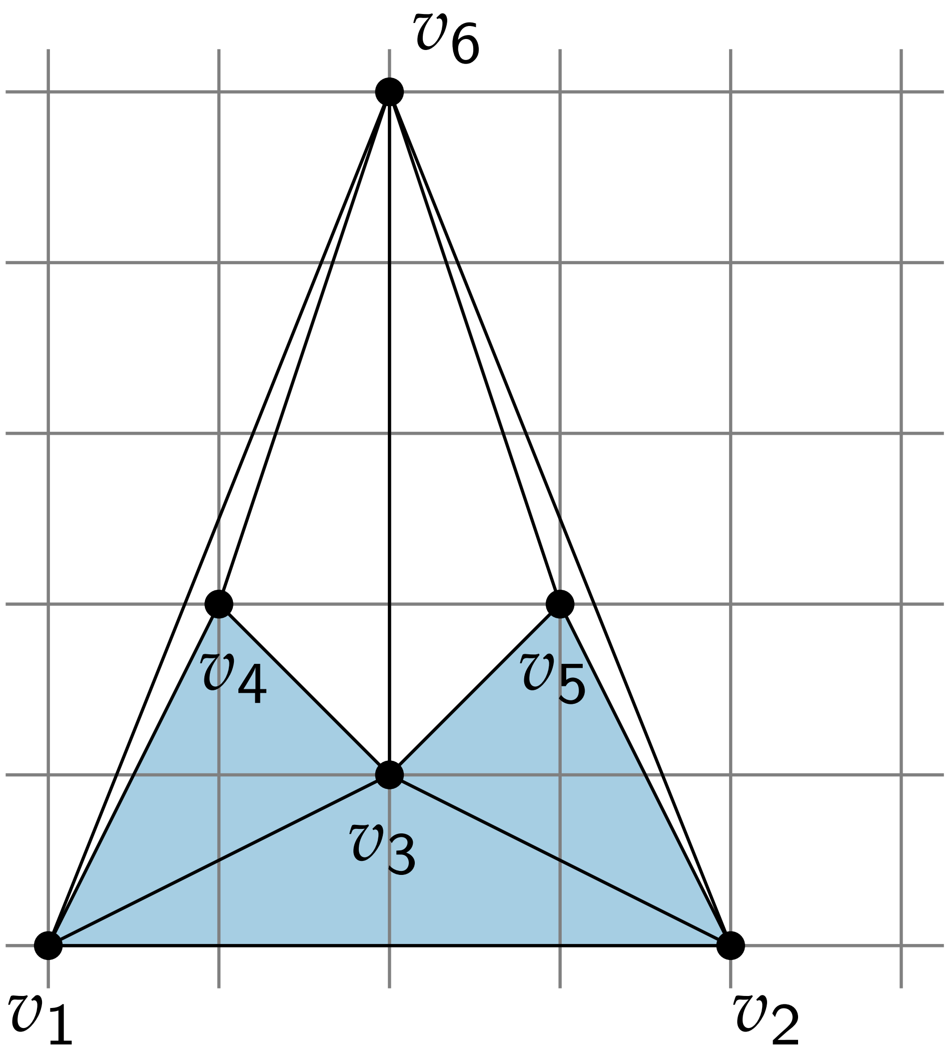
G_3: $v_1 : (0,0)$, $v_2 : ~~(2,0)~~$, $v_3 : ~~(1,1)~~$

G_4: $v_1 : (0,0)$, $v_2 : (3,0)$, $v_3 : (2,1)$, $v_4 : (1,2)$

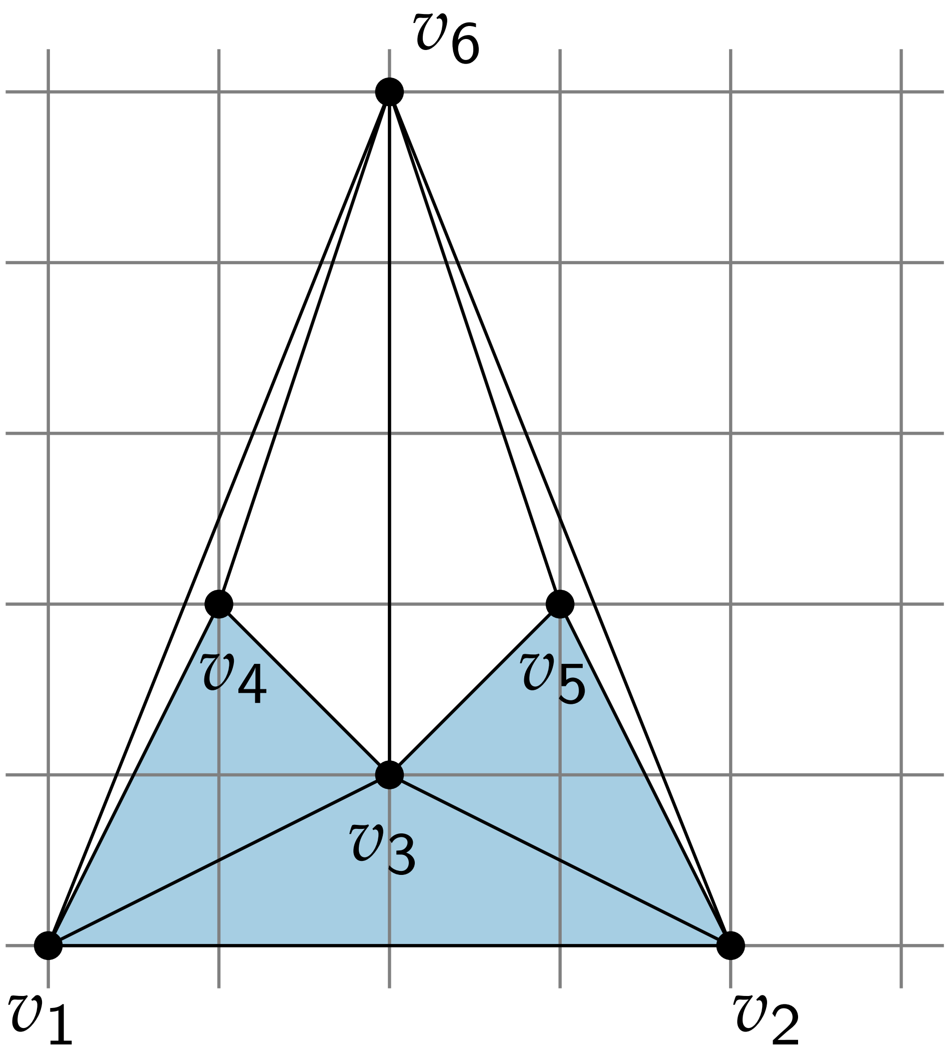
G_5: $v_1 : (0,0)$, $v_2 : ~~(4,0)~~$, $v_3 : (2,1)$, $v_4 : (1,2)$, $v_5 : (3,2)$

G: $v_6 : (2,5)$

Height



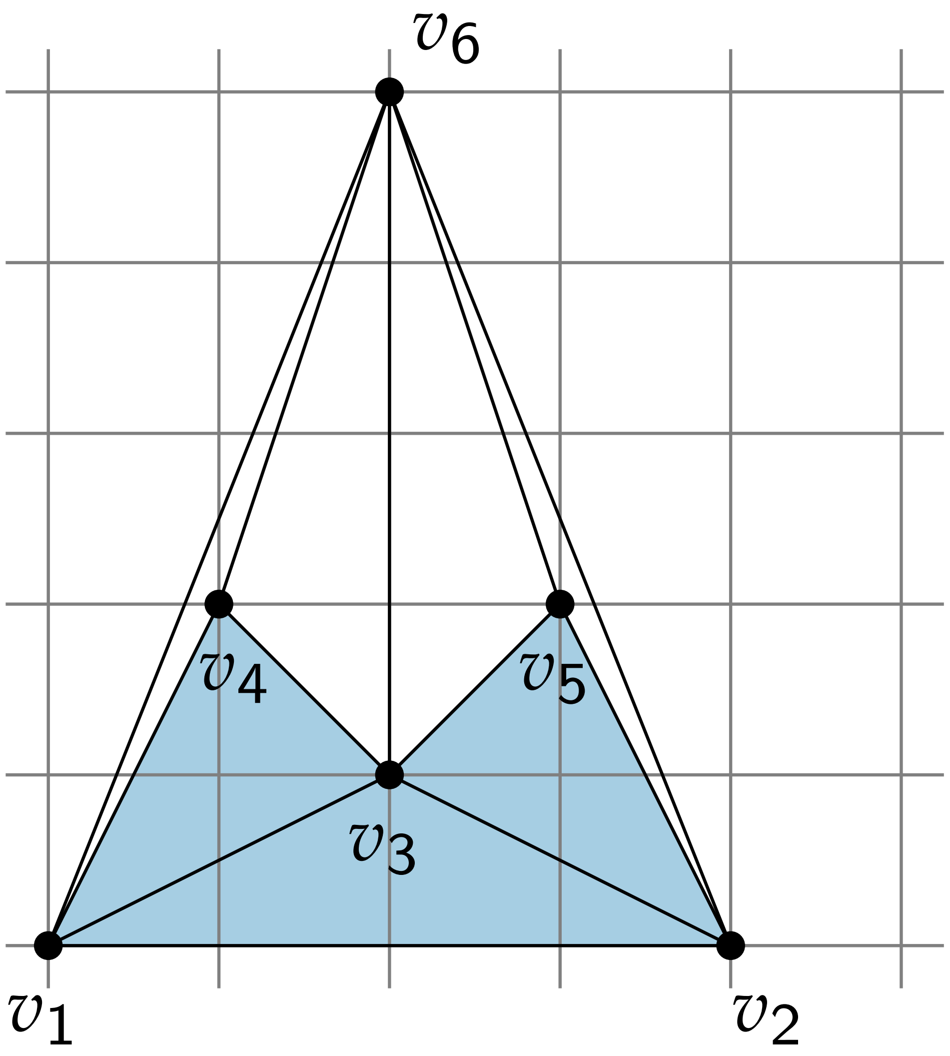
Height



Placement of v_6 depends on

- the slope of (v_1, v_4), (v_2, v_5)
- and the length of (v_1, v_2) (which is at most $n - 2$)

Height

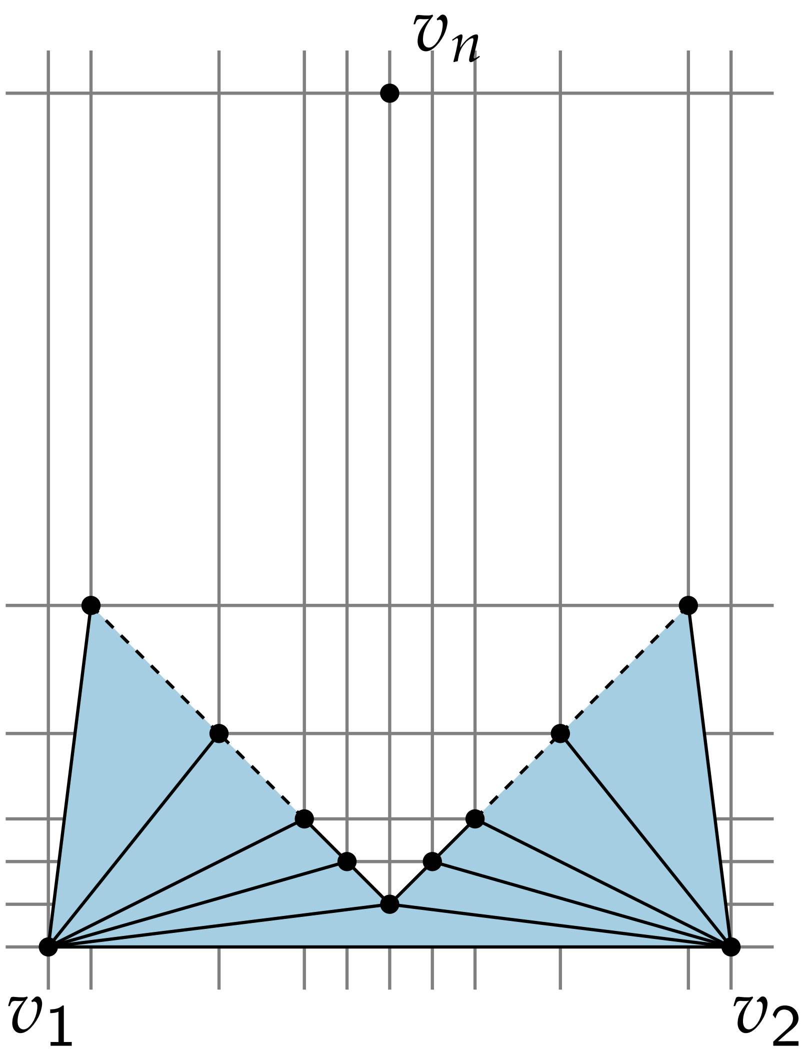


Placement of v_6 depends on

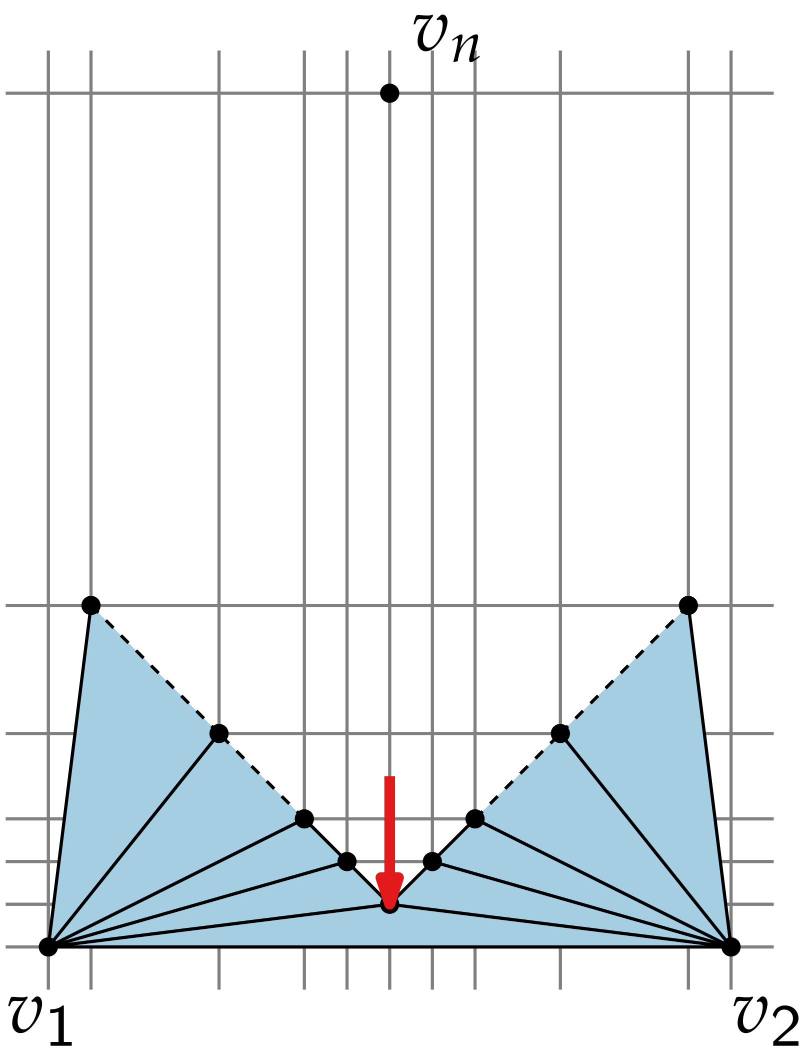
- the slope of (v_1, v_4), (v_2, v_5)
- and the length of (v_1, v_2) (which is at most $n - 2$)

Can the **height** exceed $\mathcal{O}(n)$?

Height

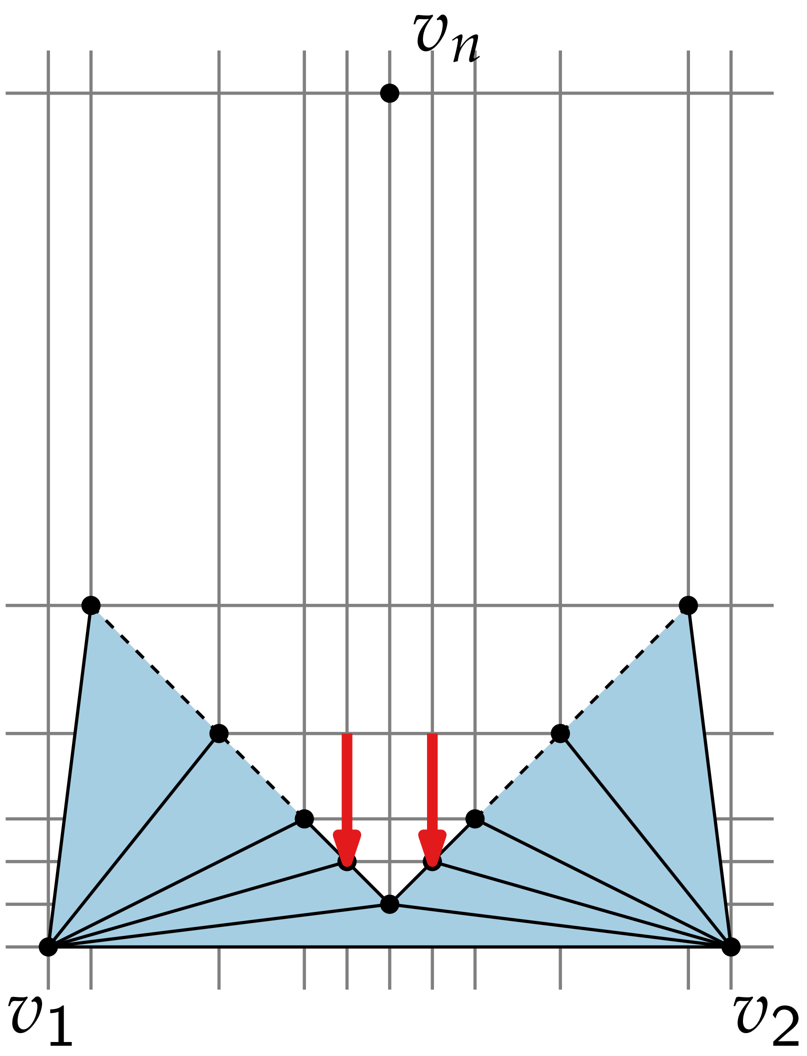


Height



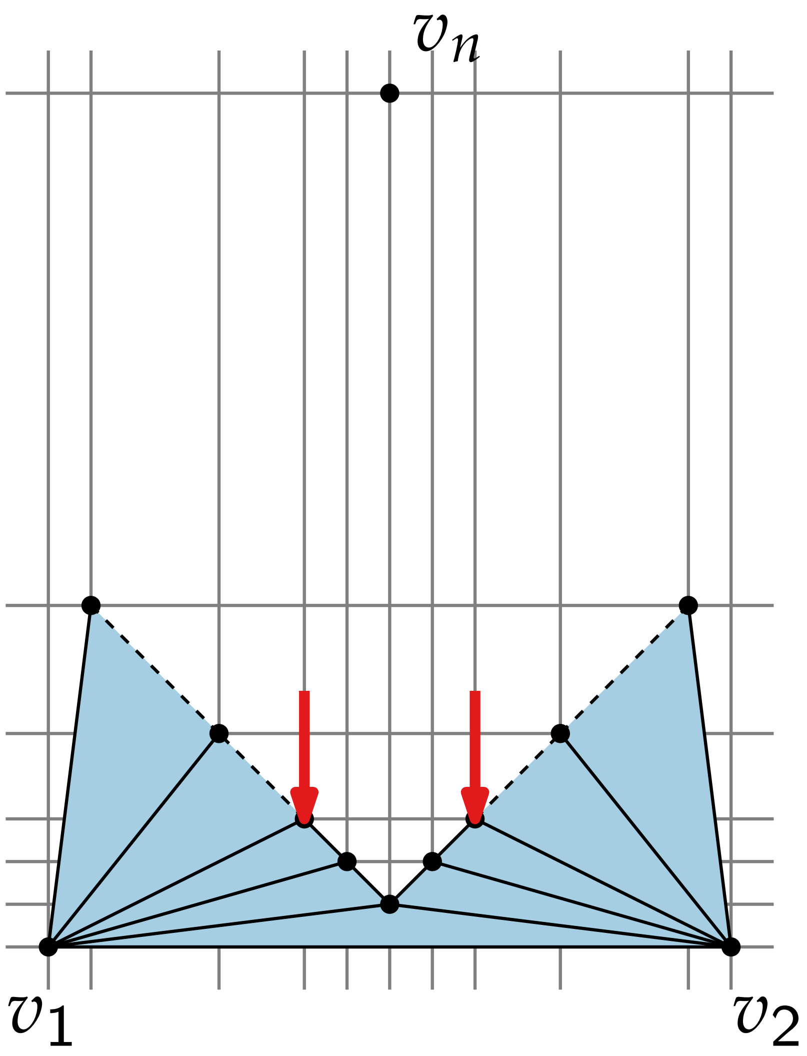
- v_3 at height 1

Height



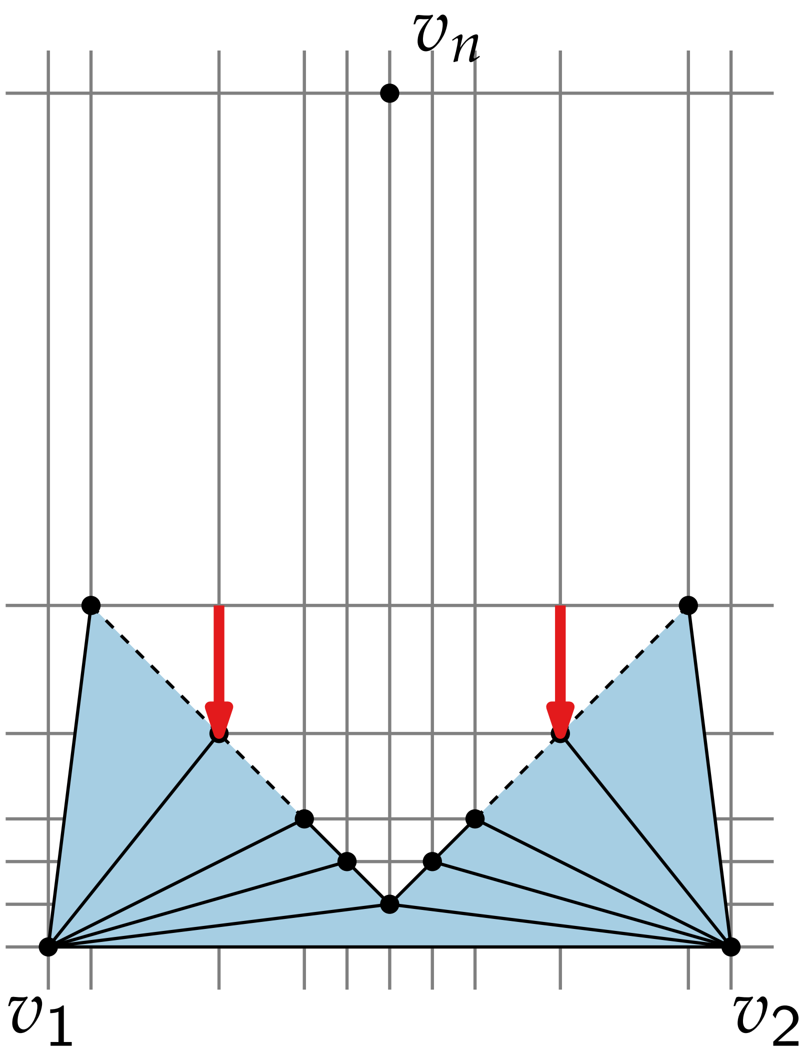
- v_3 at height 1
- v_4, v_5 at height 2

Height



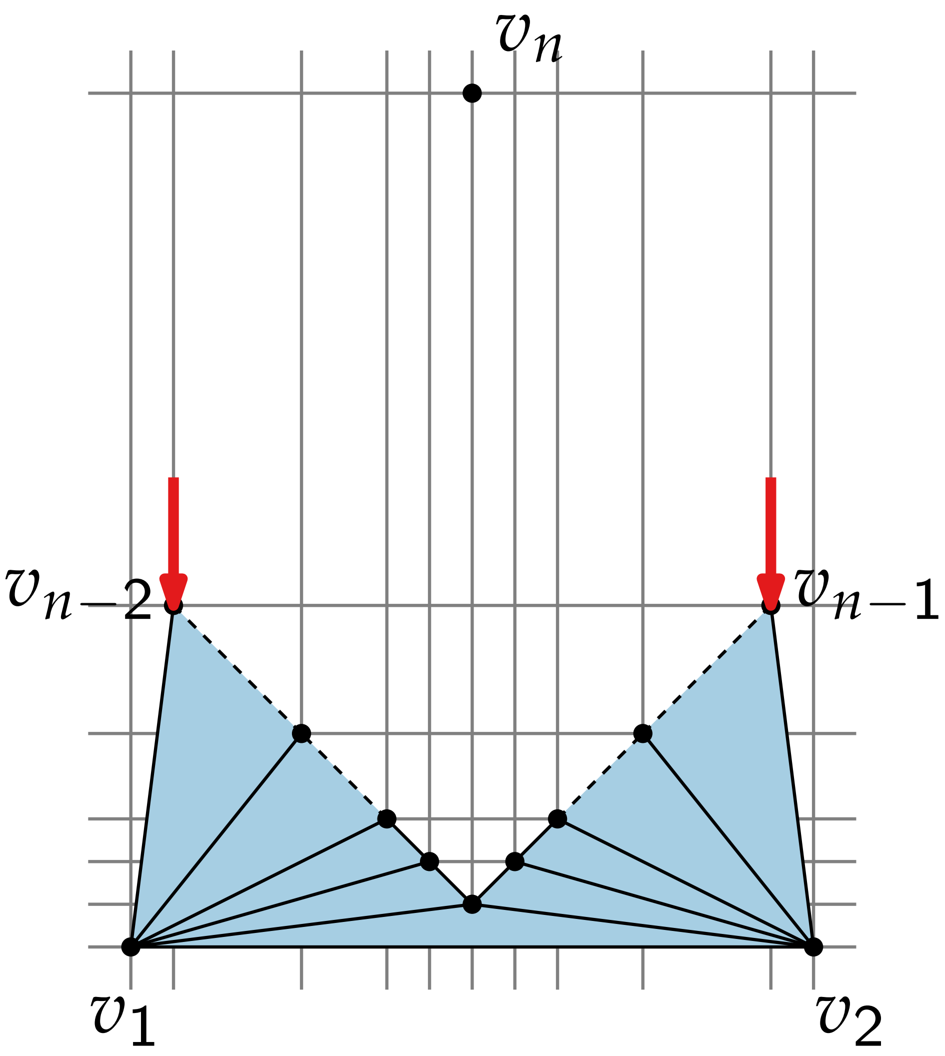
- v_3 at height 1
- v_4, v_5 at height 2
- v_6, v_7 at height 3

Height



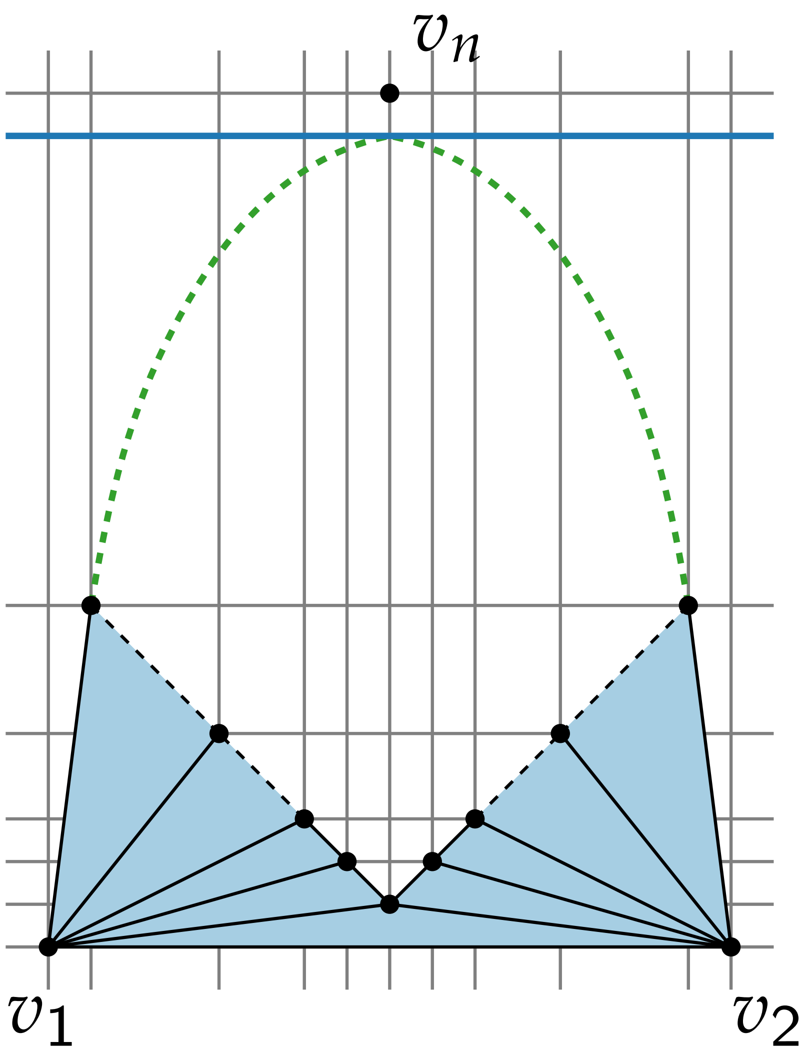
- v_3 at height 1
- v_4, v_5 at height 2
- v_6, v_7 at height 3
- v_{2i}, v_{2i+1} at height i

Height



- v_3 at height 1
- v_4, v_5 at height 2
- v_6, v_7 at height 3
- v_{2i}, v_{2i+1} at height i
- v_{n-2}, v_{n-1} at height $\frac{n-2}{2}$

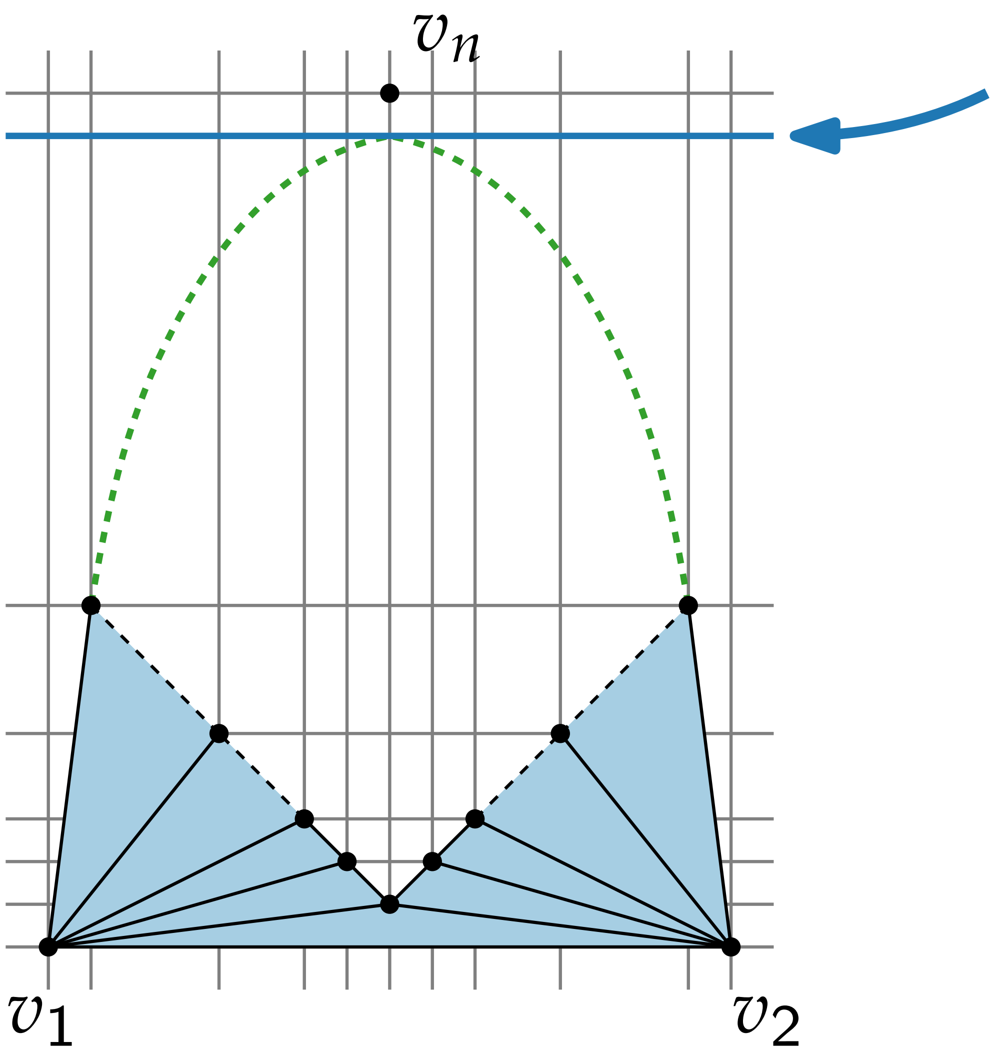
Height



- Slope for $(v_1, v_{n-2}) = \frac{n-2}{2}$
- Slope for $(v_2, v_{n-1}) = -\frac{n-2}{2}$
- Length of $(v_1, v_2) = n - 2$

- v_3 at height 1
- v_4, v_5 at height 2
- v_6, v_7 at height 3
- v_{2i}, v_{2i+1} at height i
- v_{n-2}, v_{n-1} at height $\frac{n-2}{2}$

Height

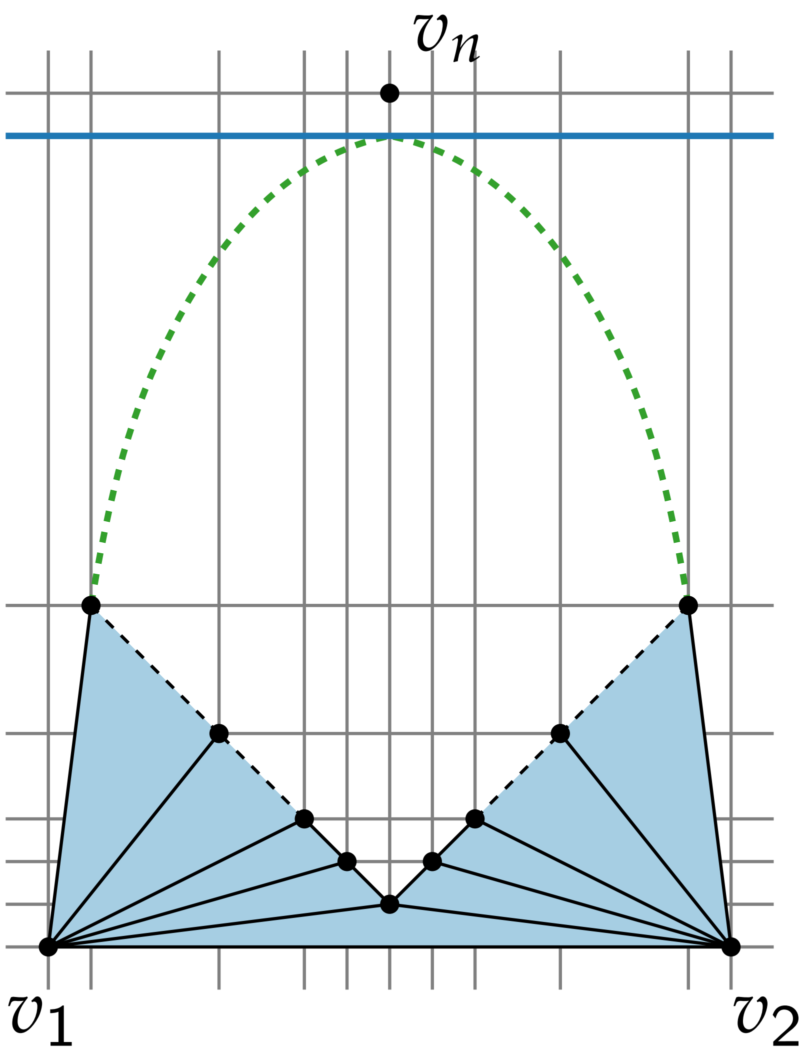


v_n above $\frac{(n-2)^2}{4}$

- Slope for $(v_1, v_{n-2}) = \frac{n-2}{2}$
- Slope for $(v_2, v_{n-1}) = -\frac{n-2}{2}$
- Length of $(v_1, v_2) = n - 2$

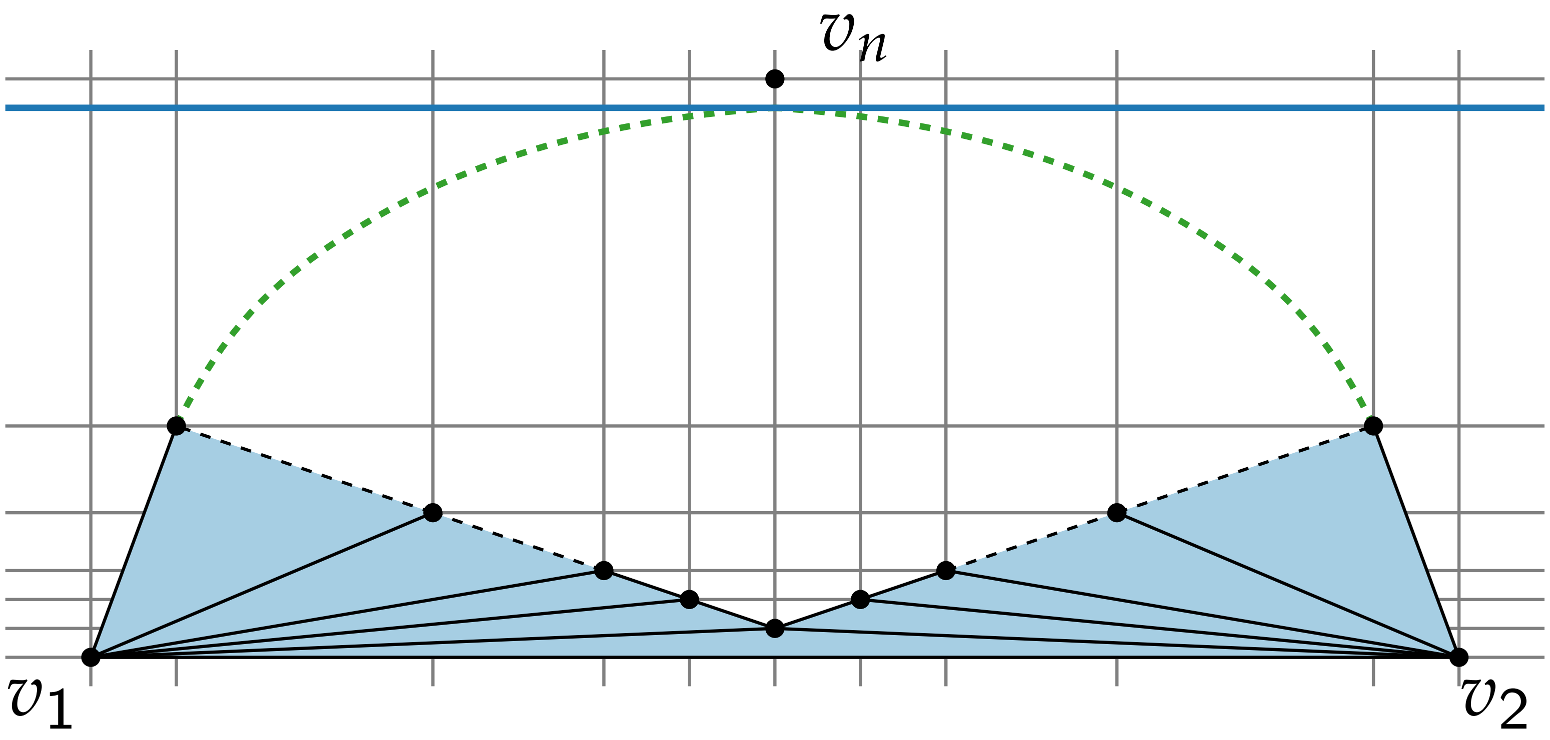
- v_3 at height 1
- v_4, v_5 at height 2
- v_6, v_7 at height 3
- v_{2i}, v_{2i+1} at height i
- v_{n-2}, v_{n-1} at height $\frac{n-2}{2}$

Height

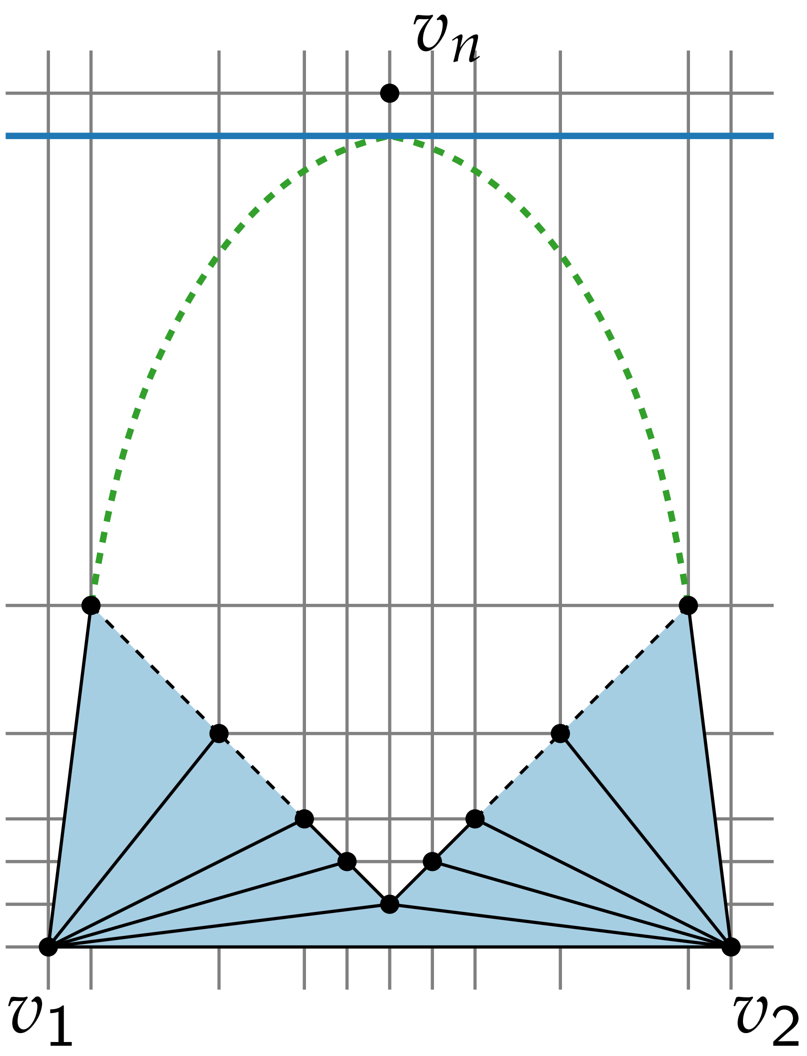


Stretching?

- decrease the height
- increase the width
- vertices on the grid?



Height



Stretching?

- decrease the height
- increase the width
- vertices on the grid?

Shifting

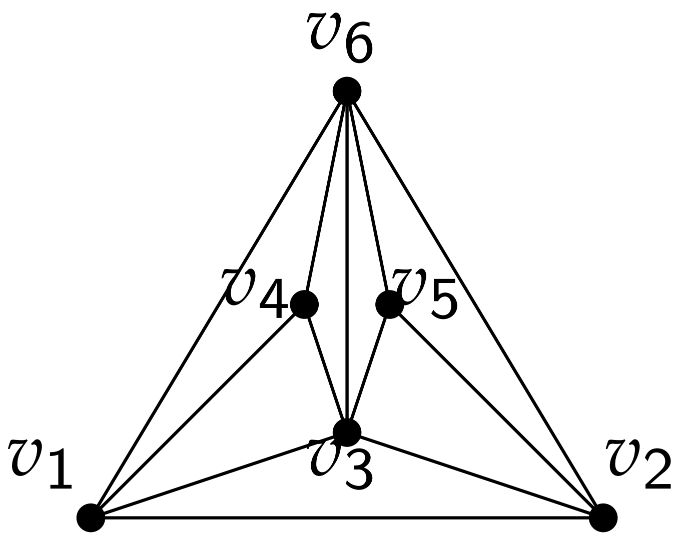
- control slopes
- additional shifting at each step

Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,

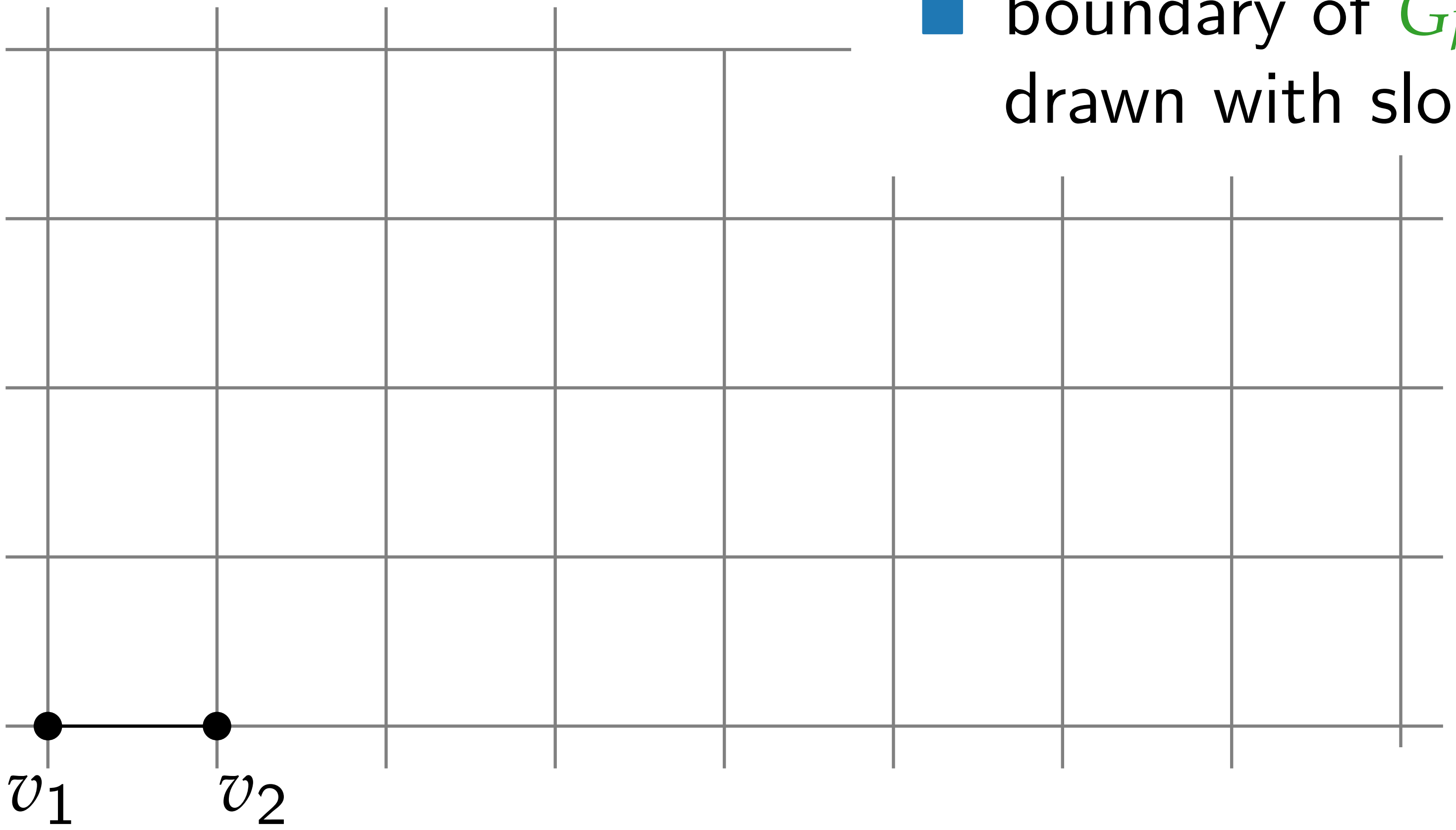


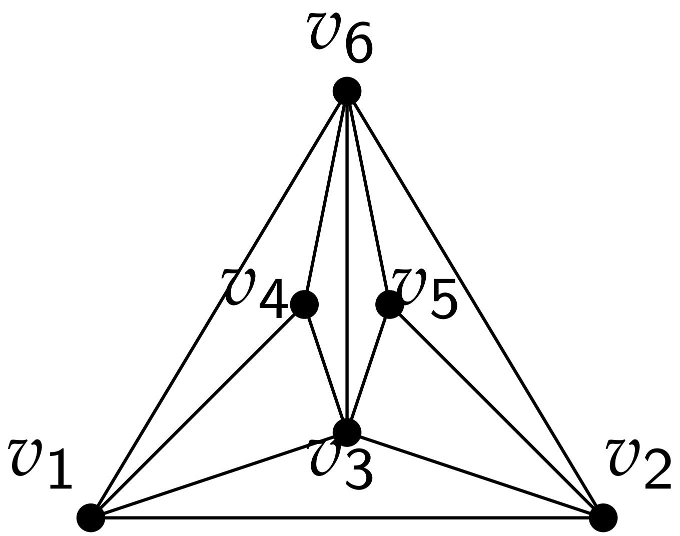
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



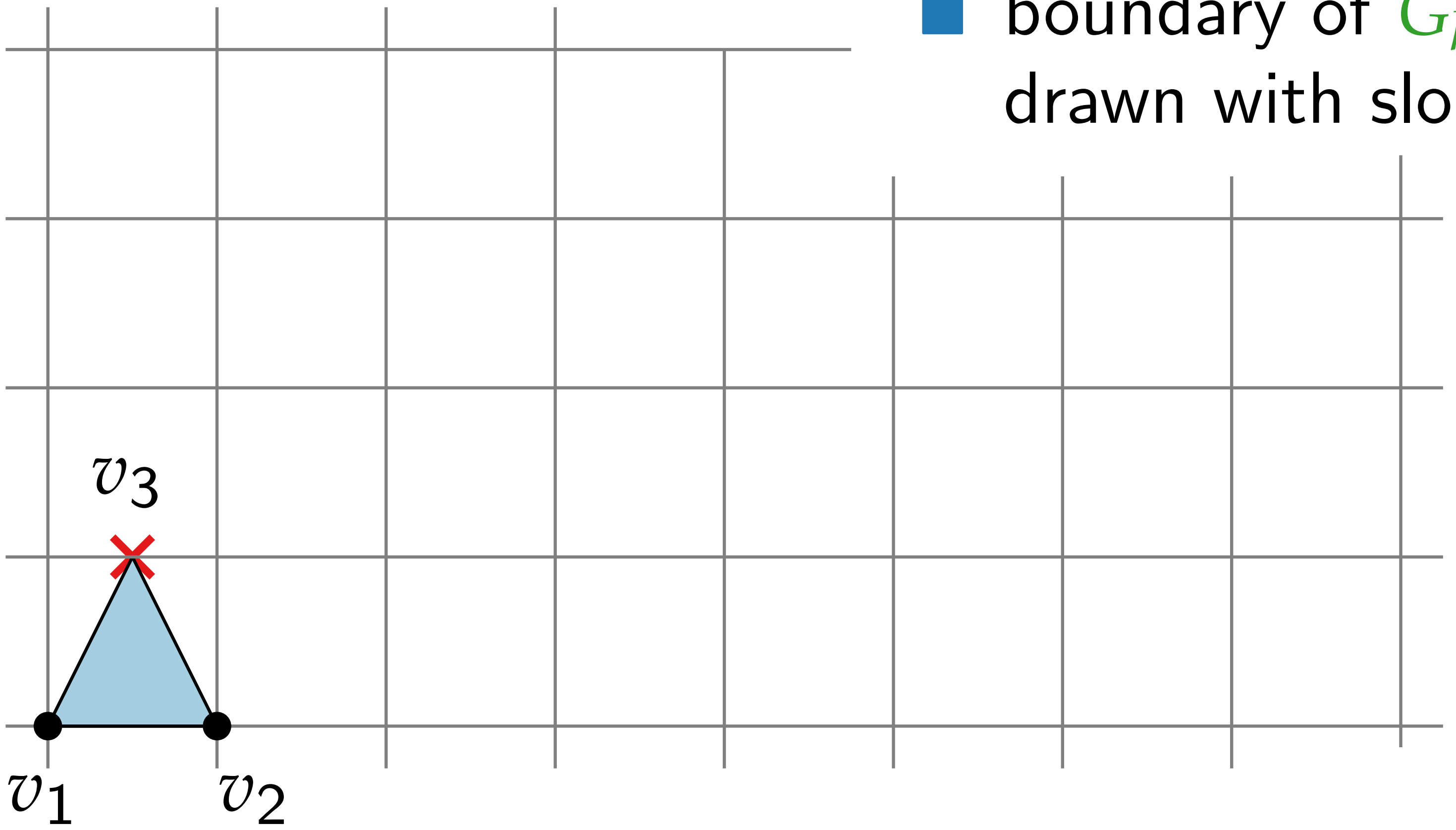


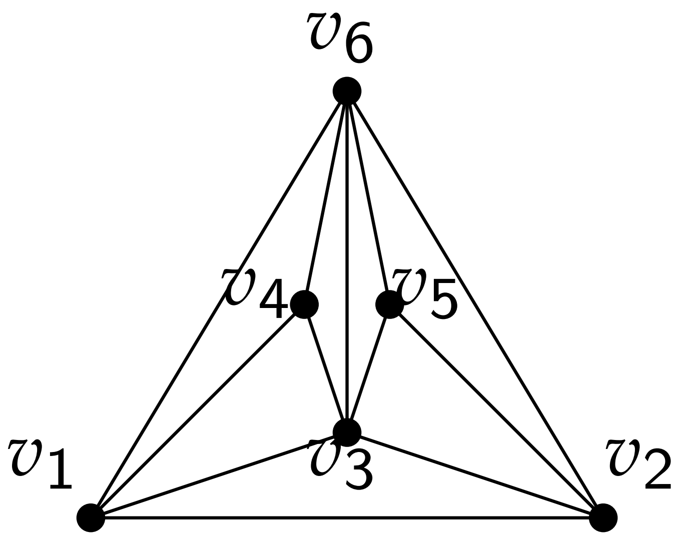
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



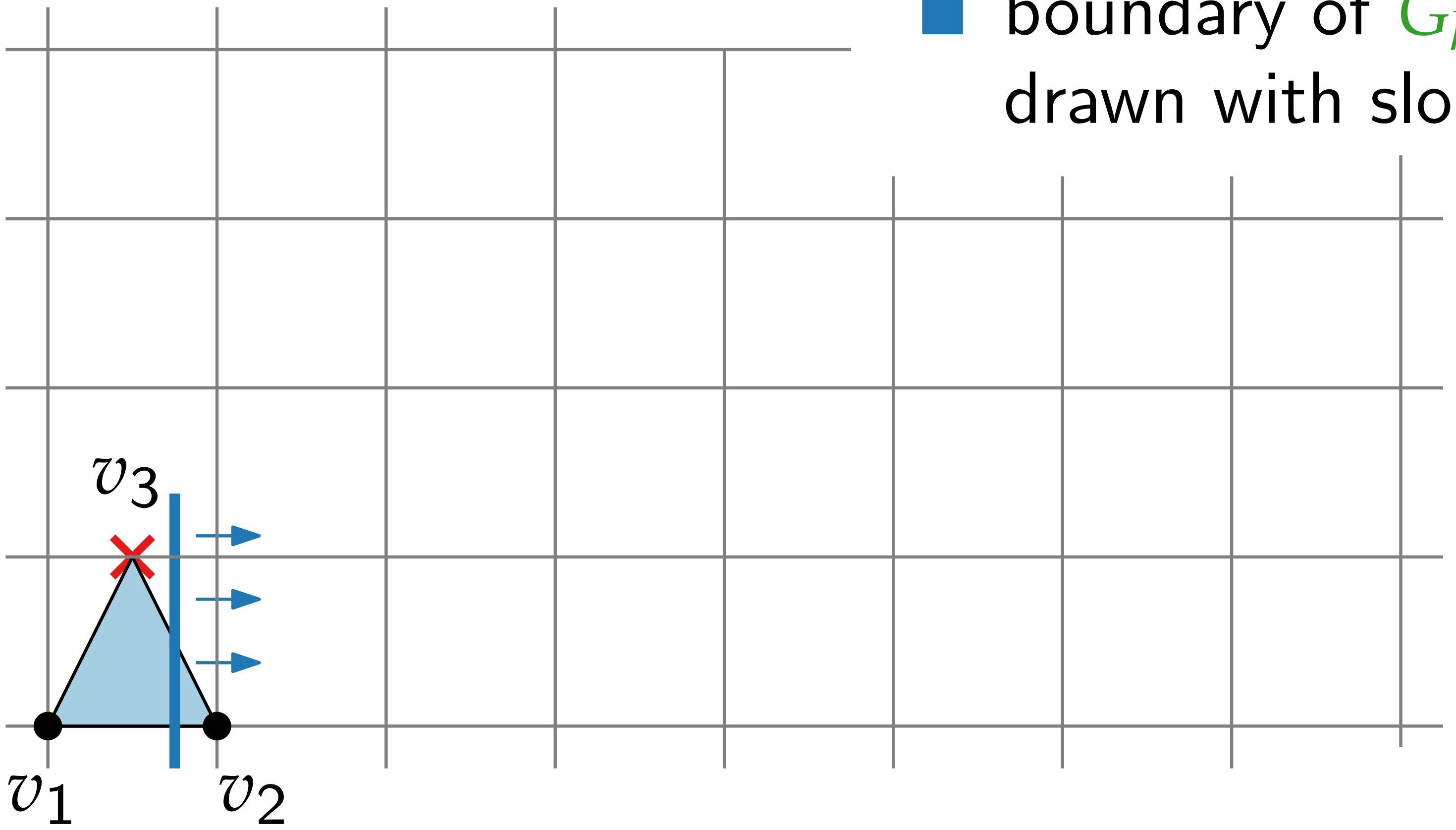


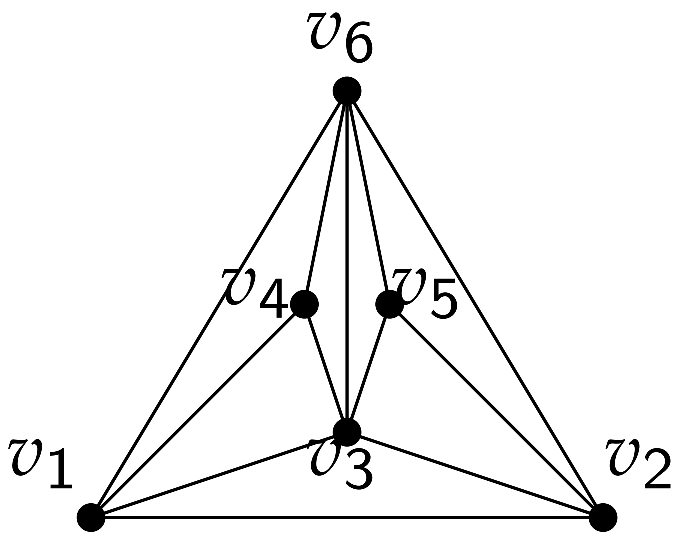
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



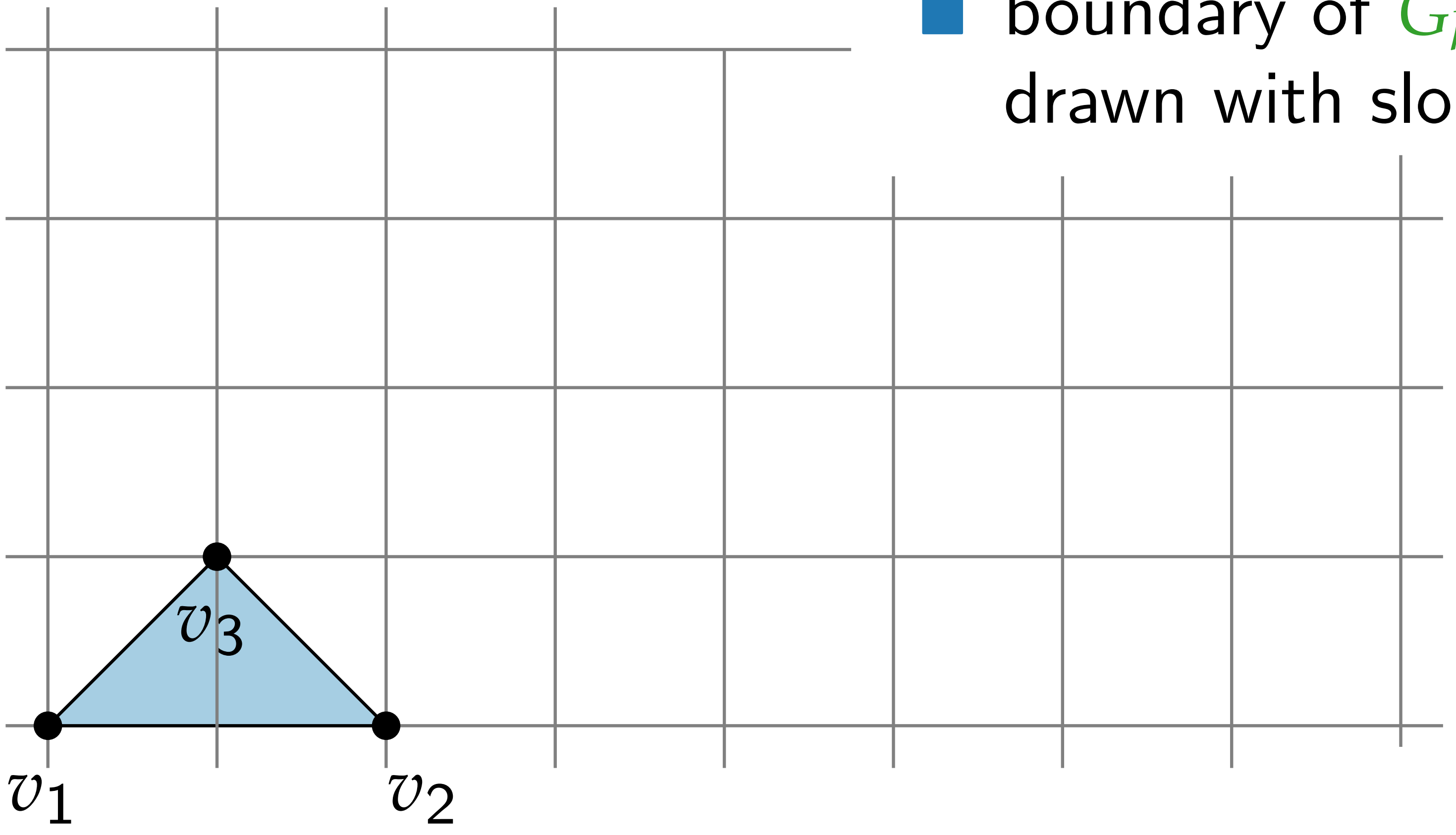


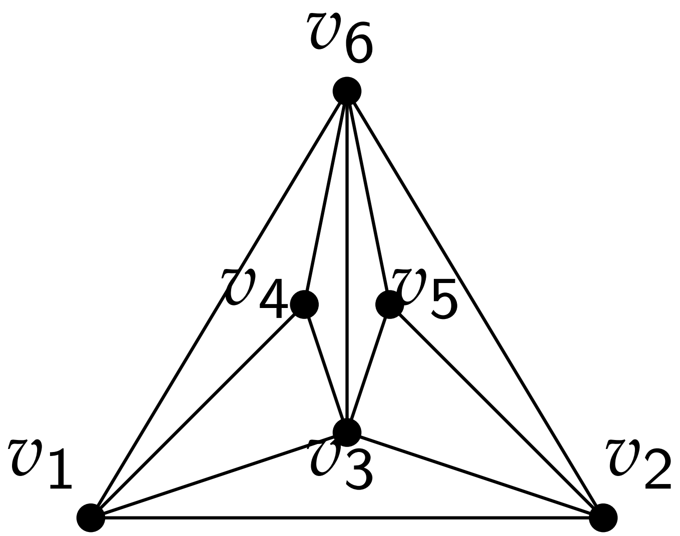
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



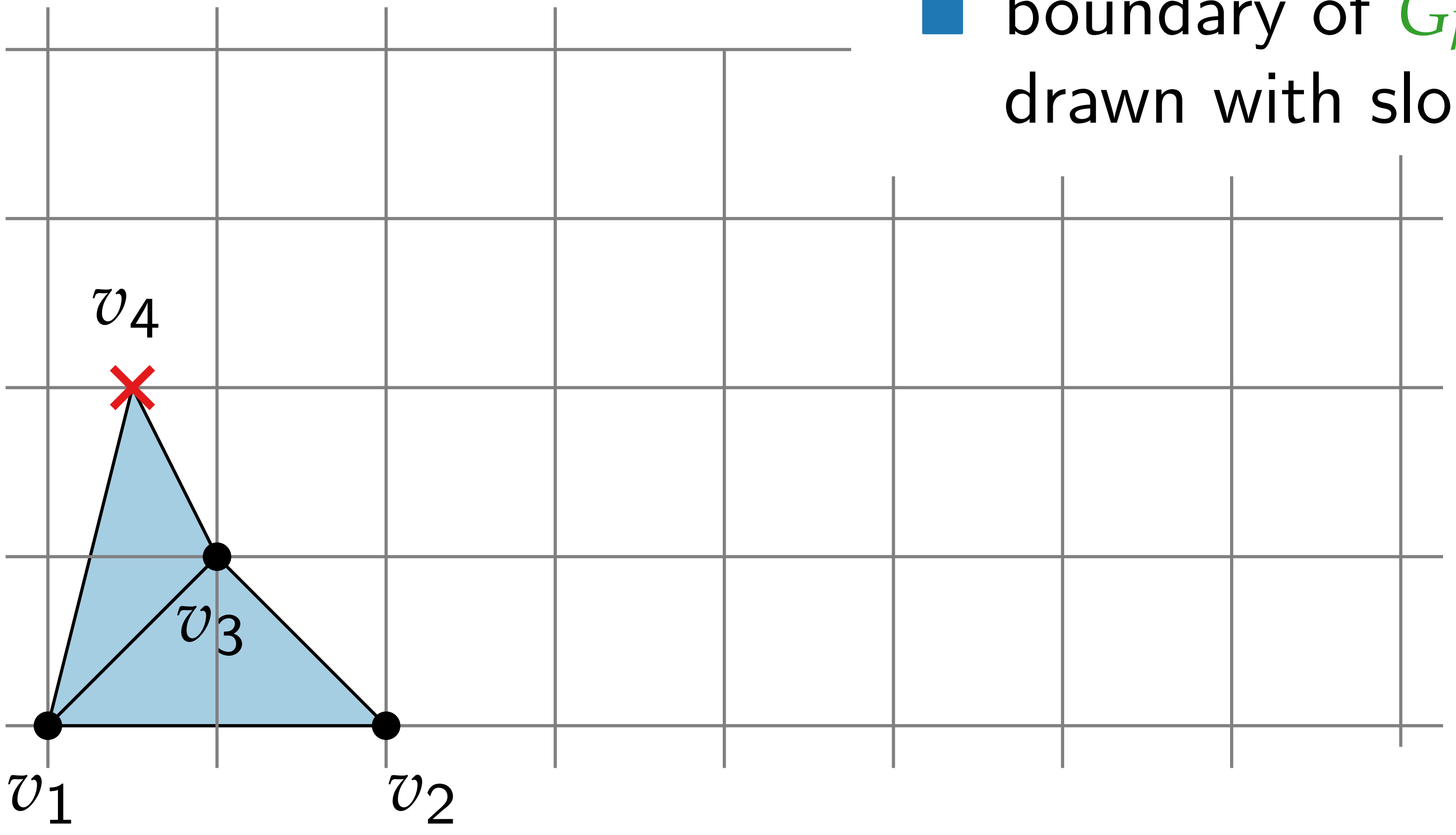


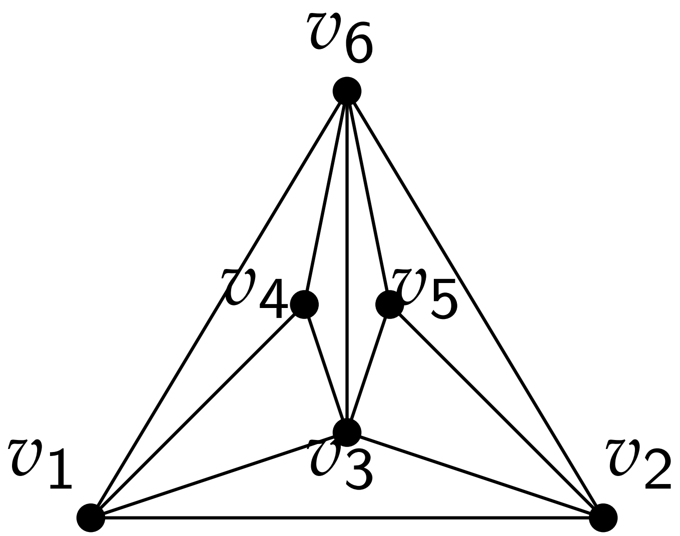
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



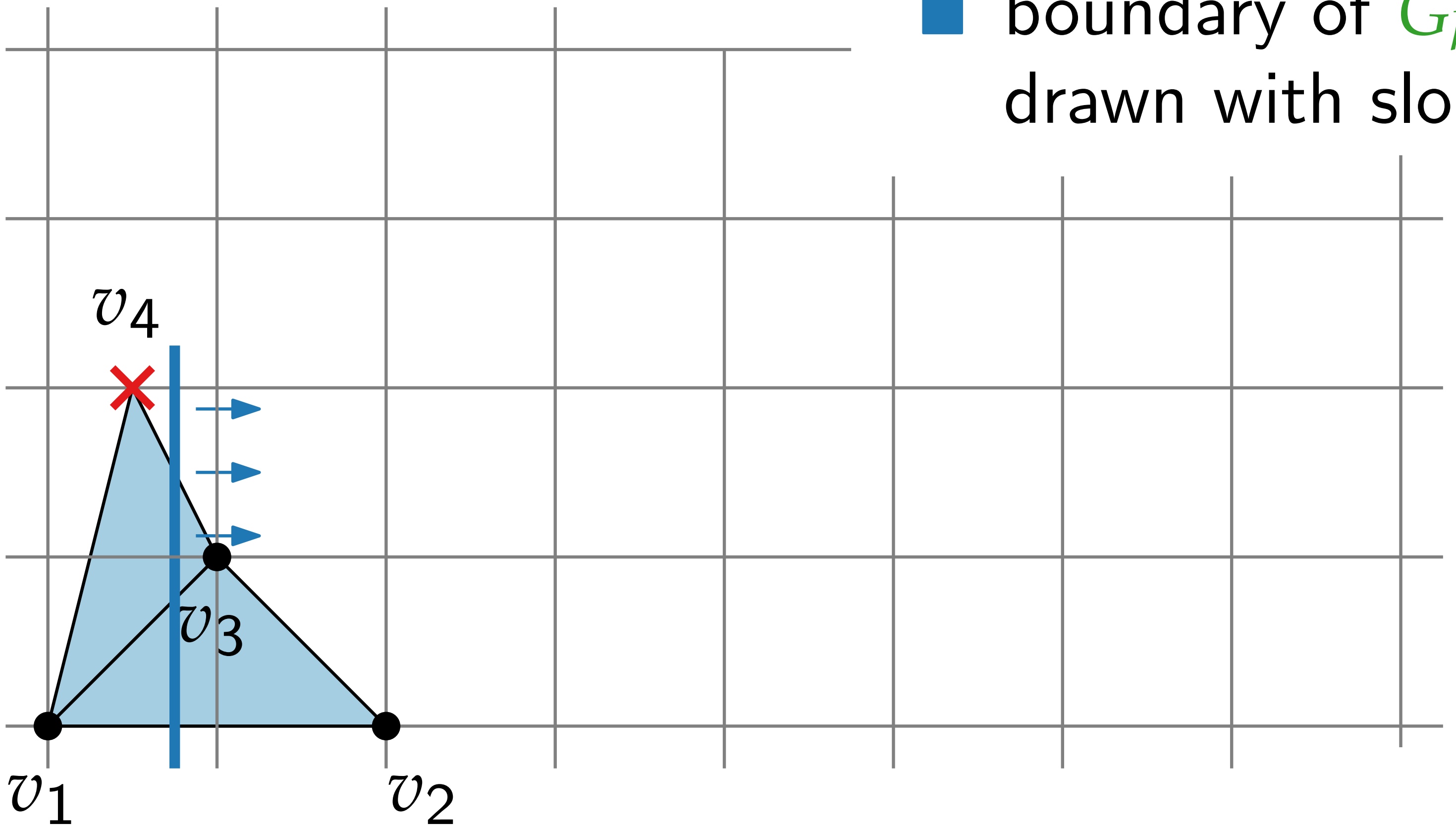


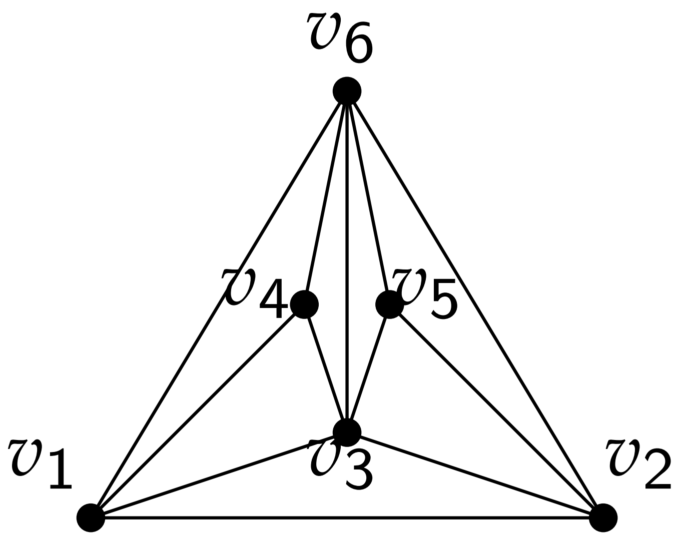
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



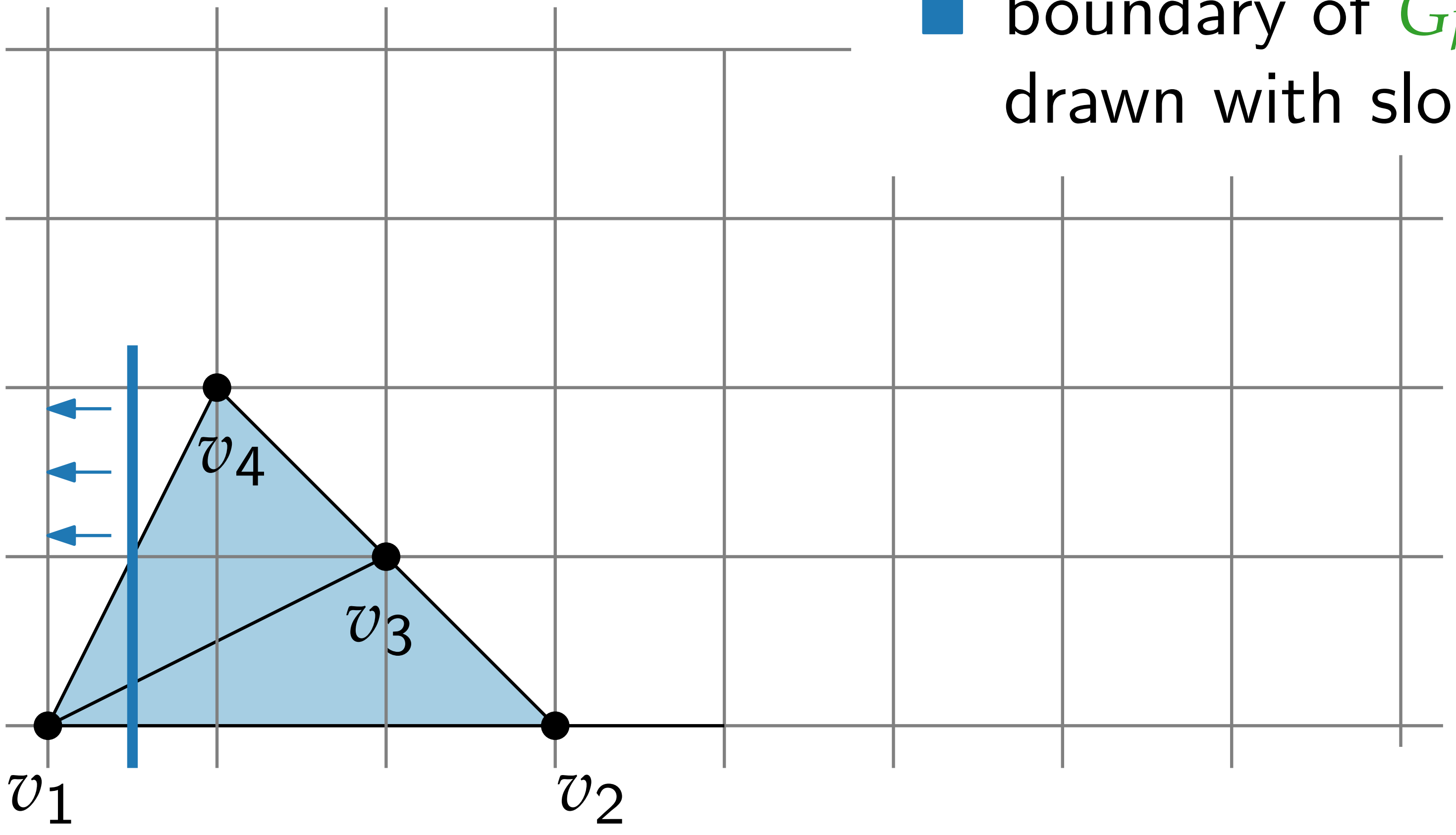


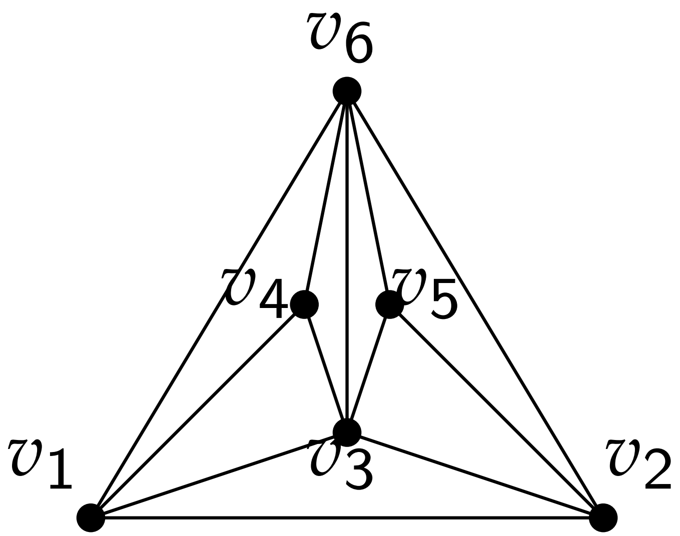
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



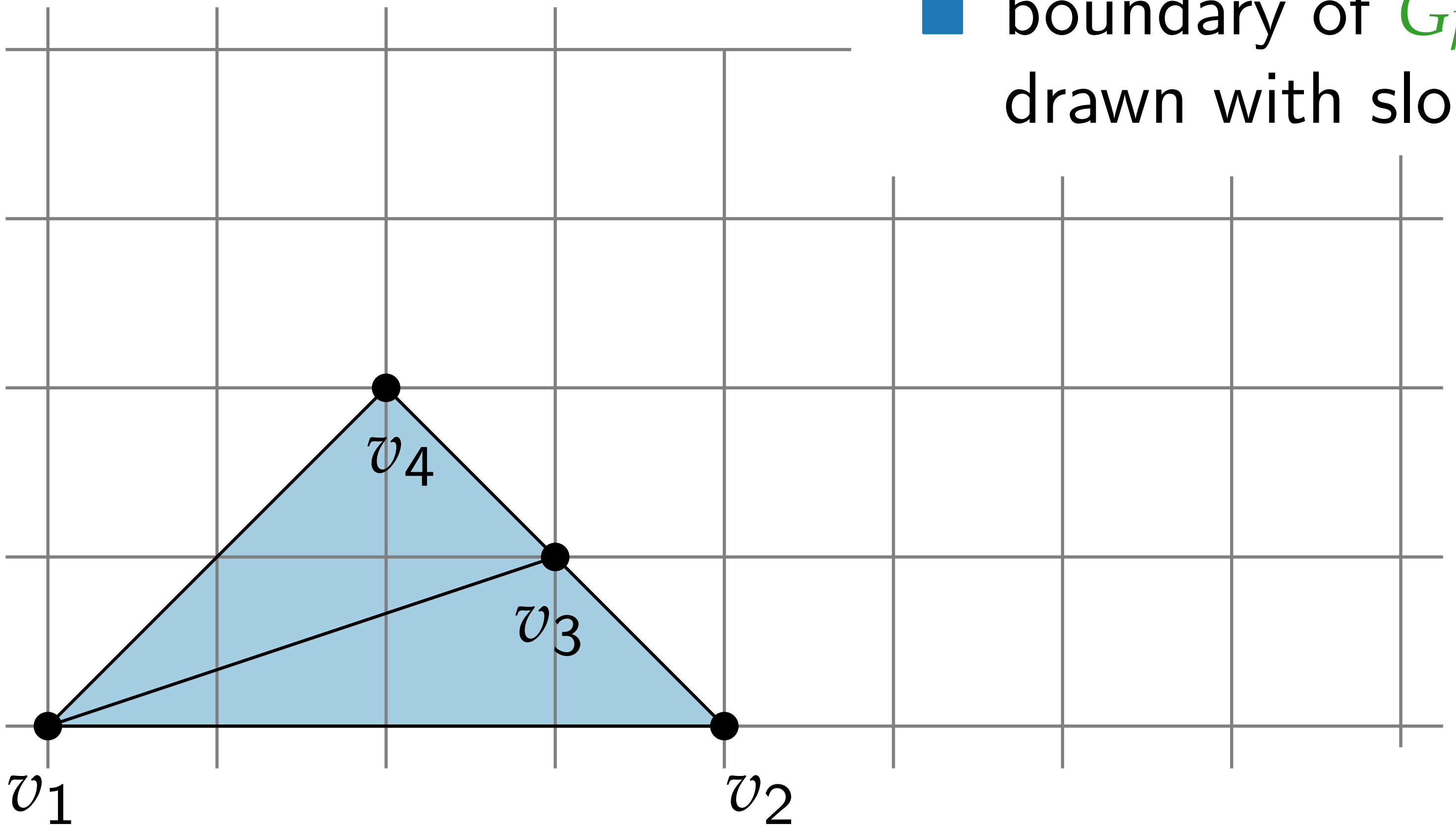


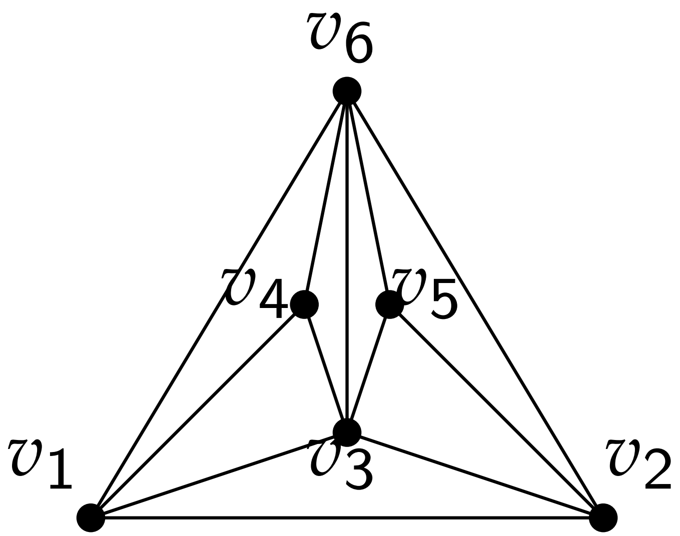
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



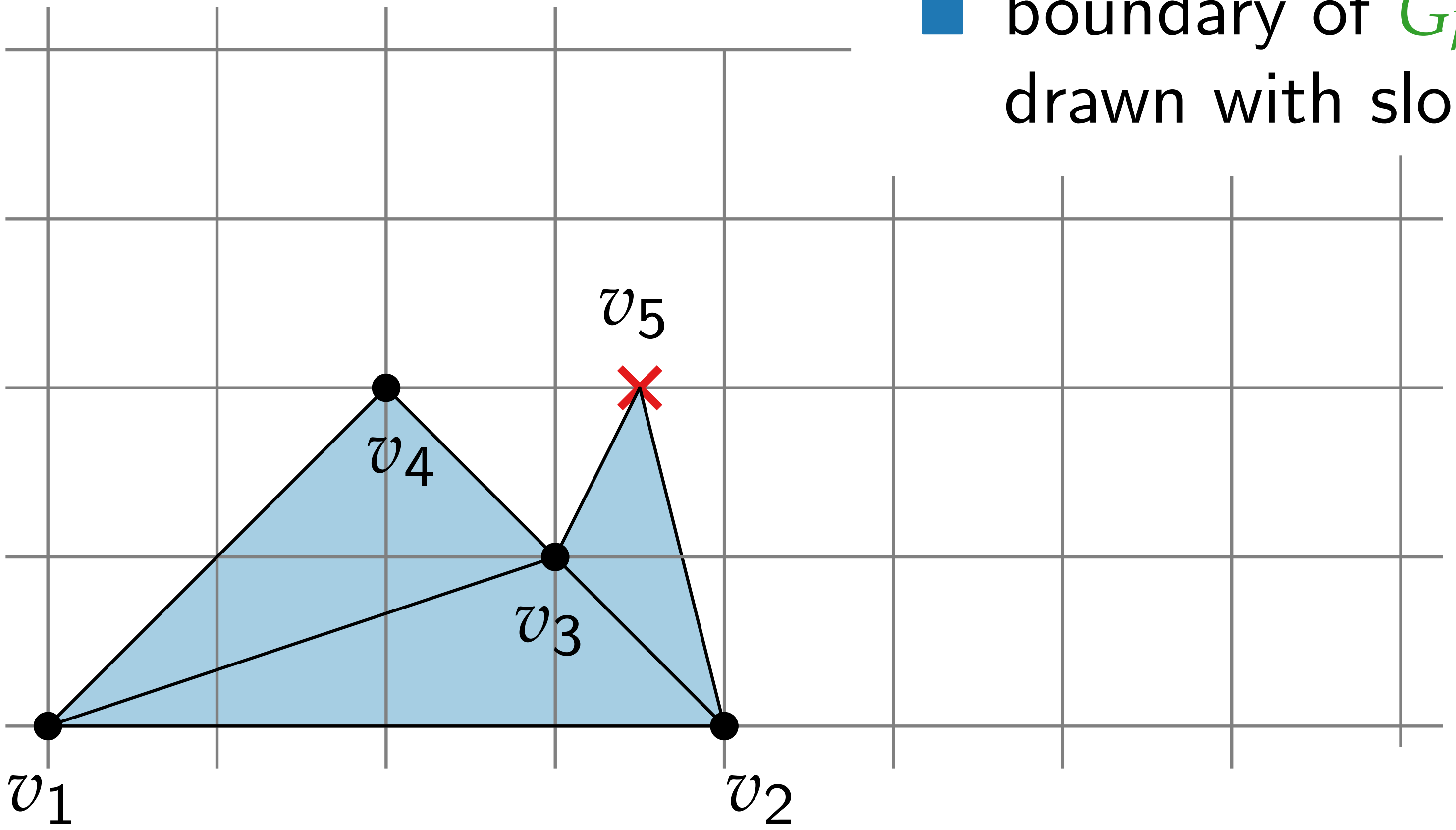


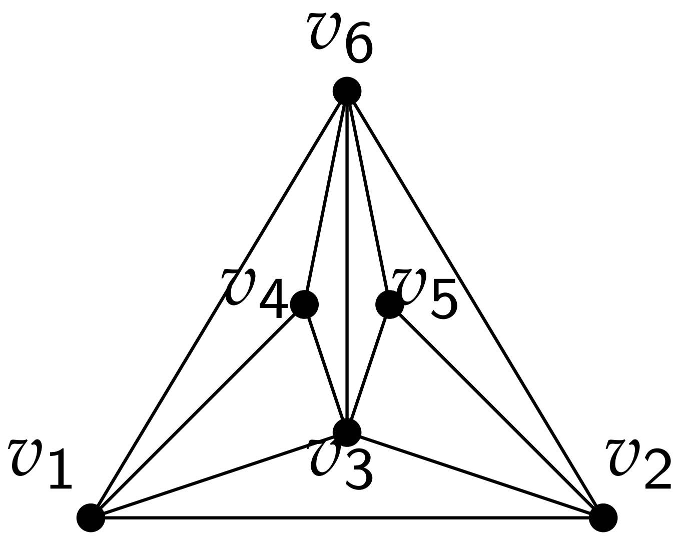
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



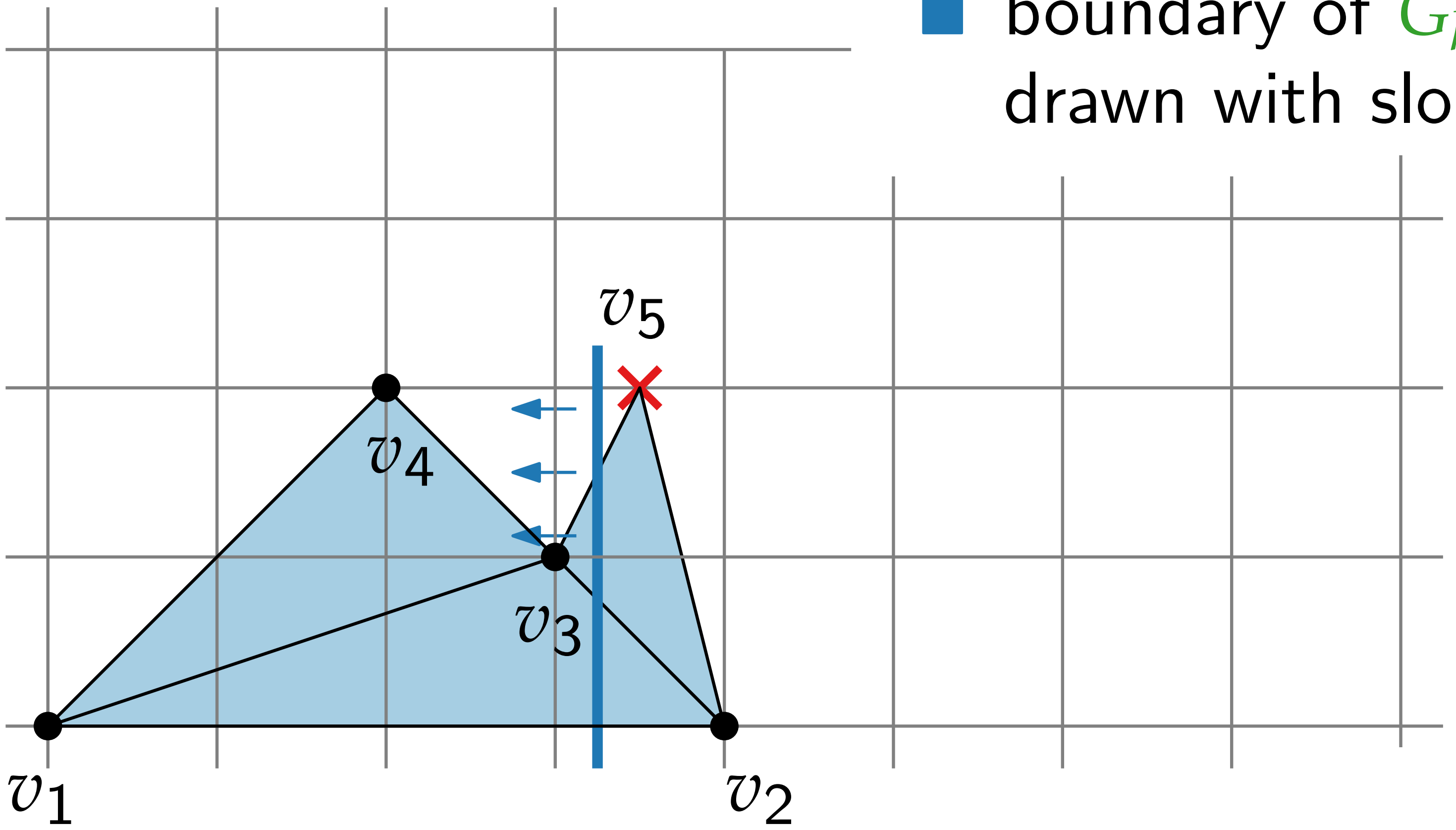


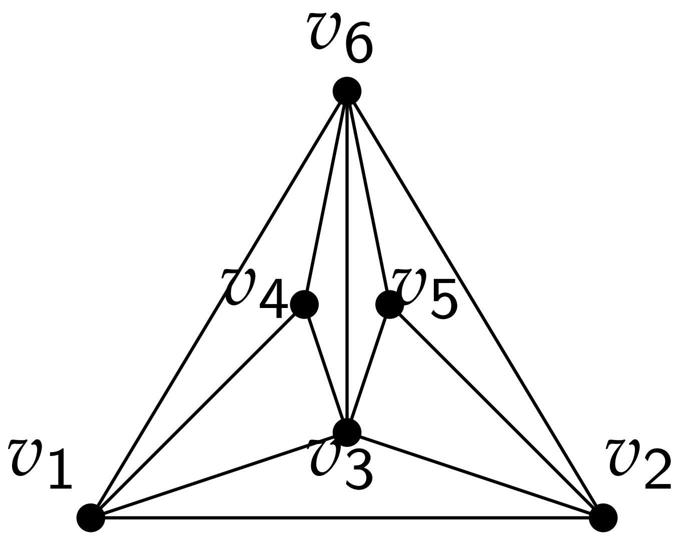
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



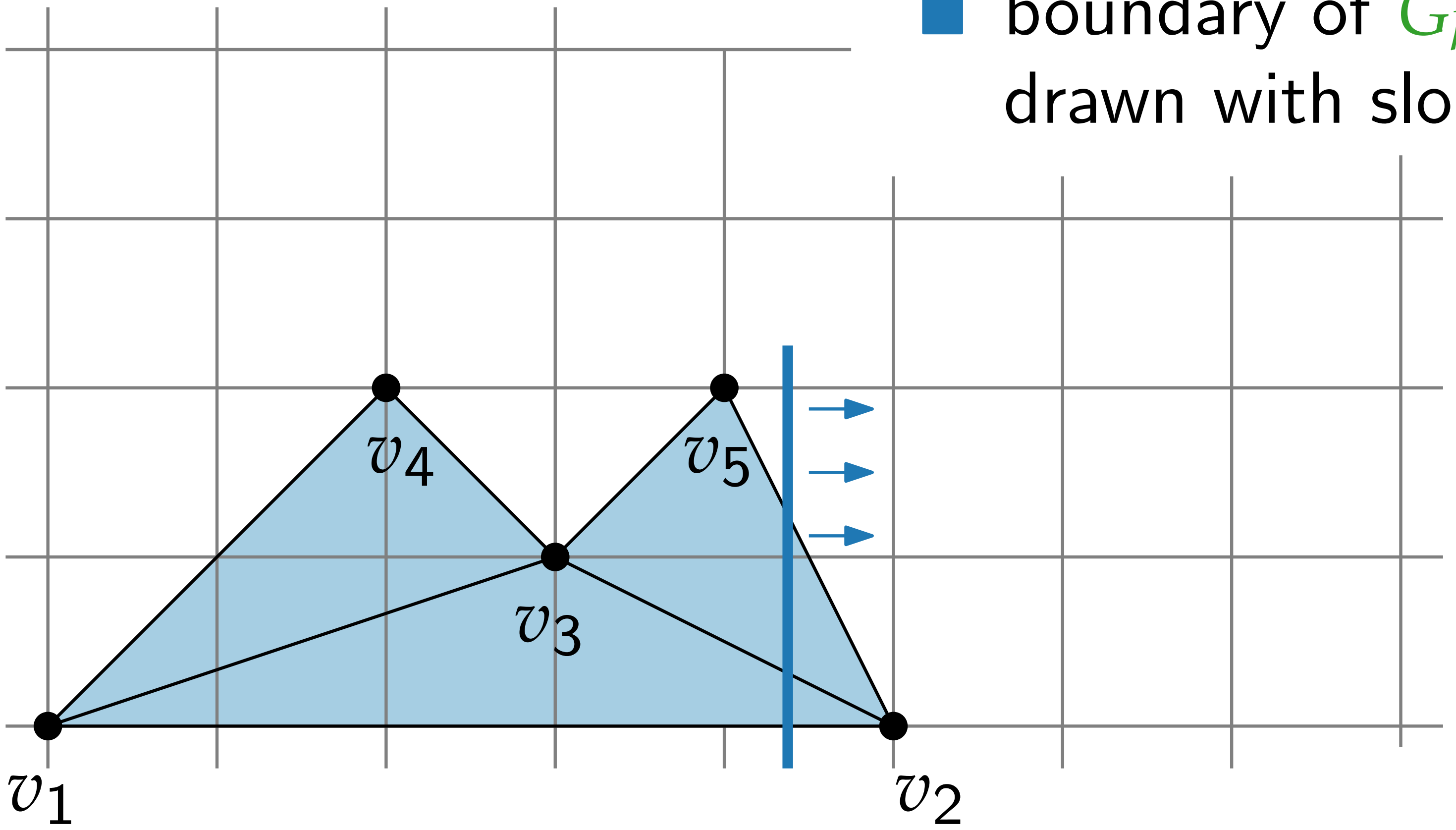


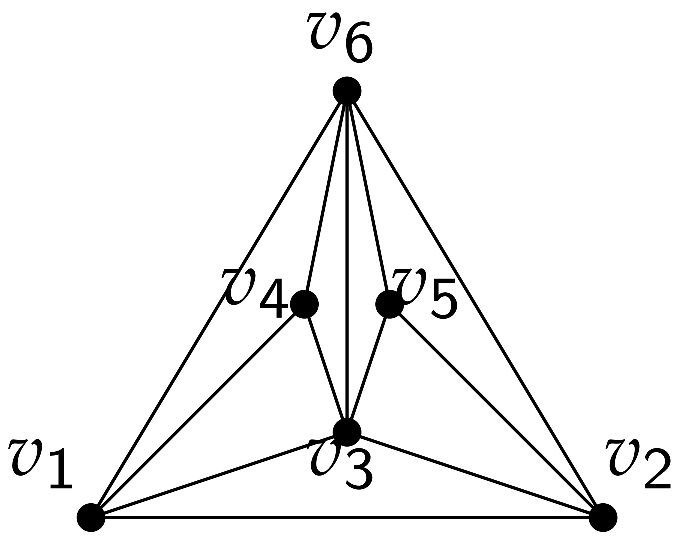
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



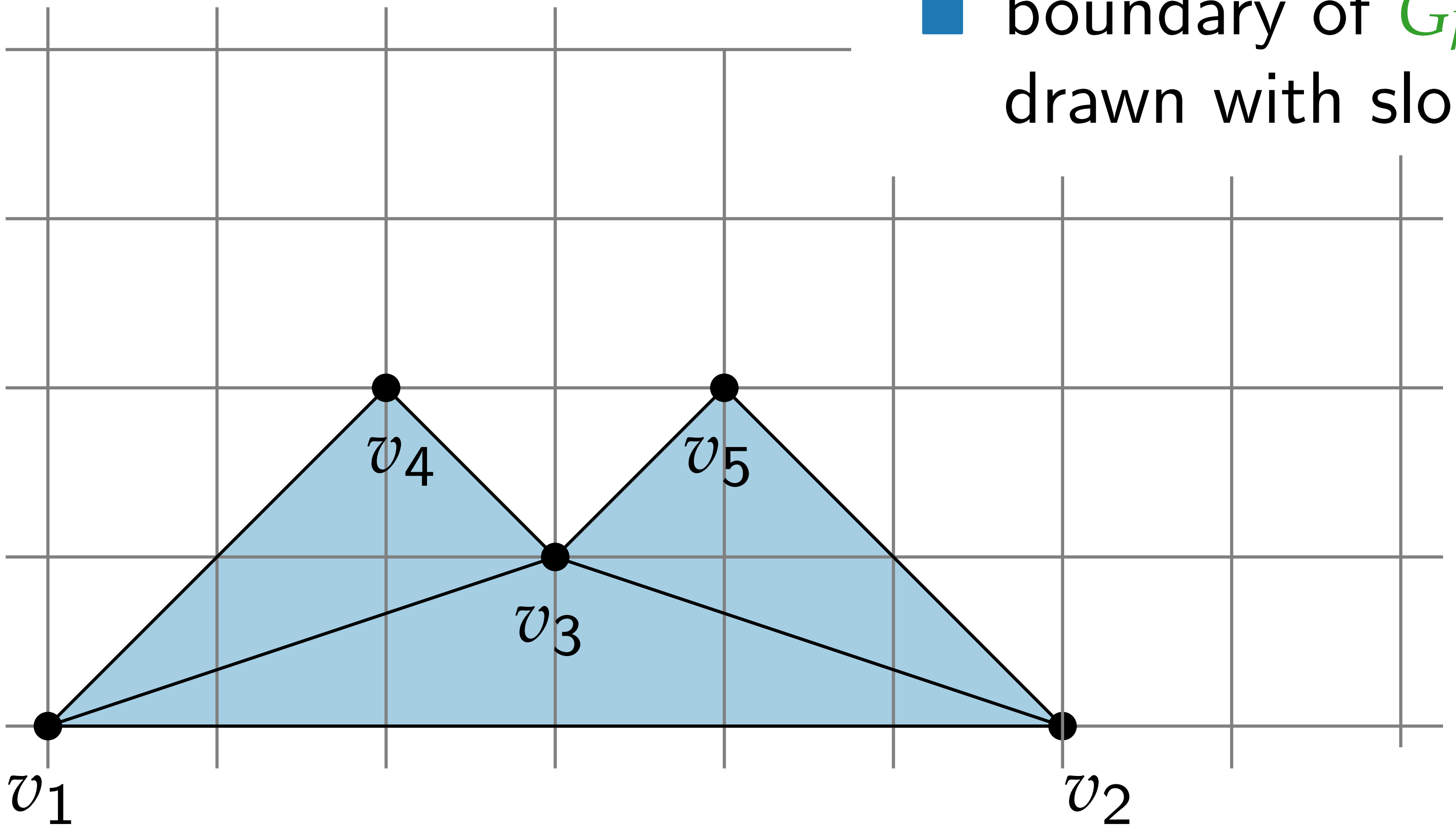


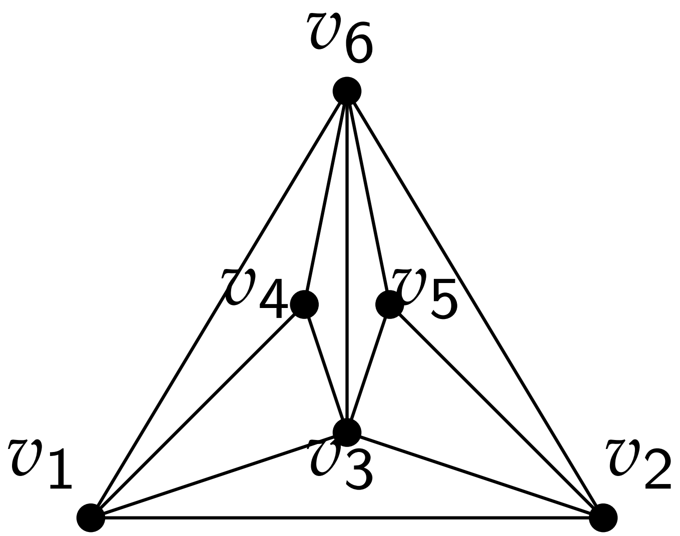
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



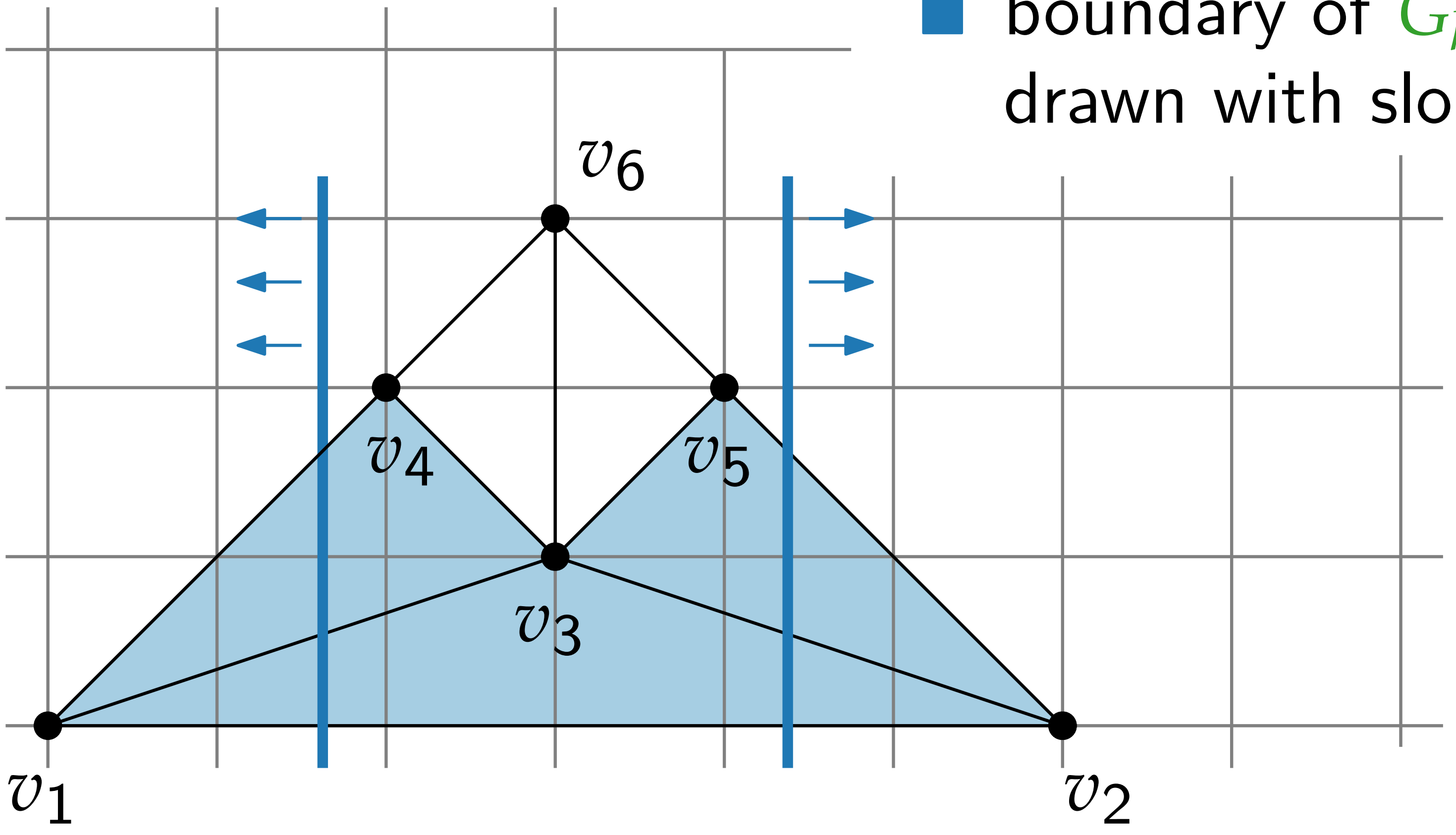


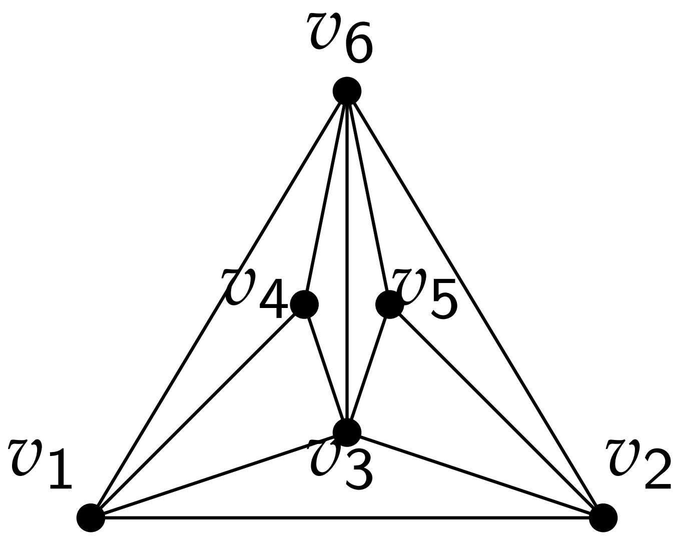
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



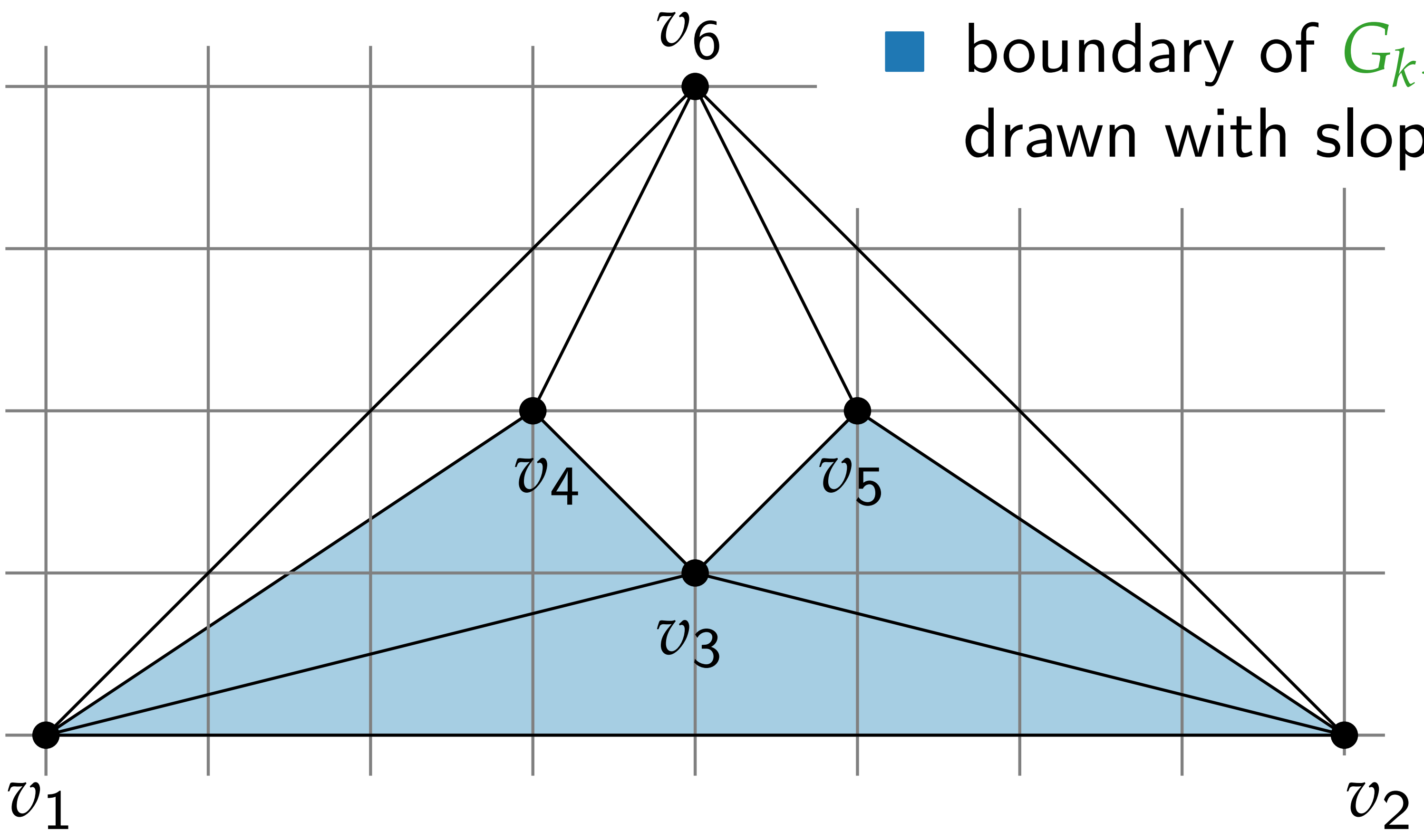


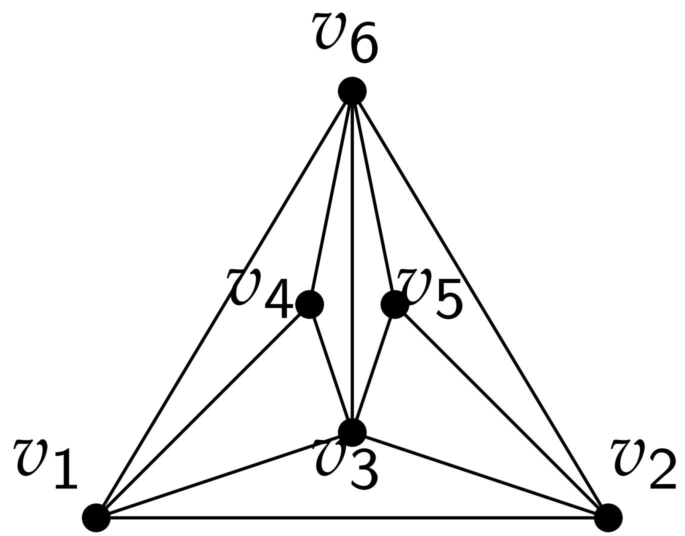
Constraints

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,





Constraints

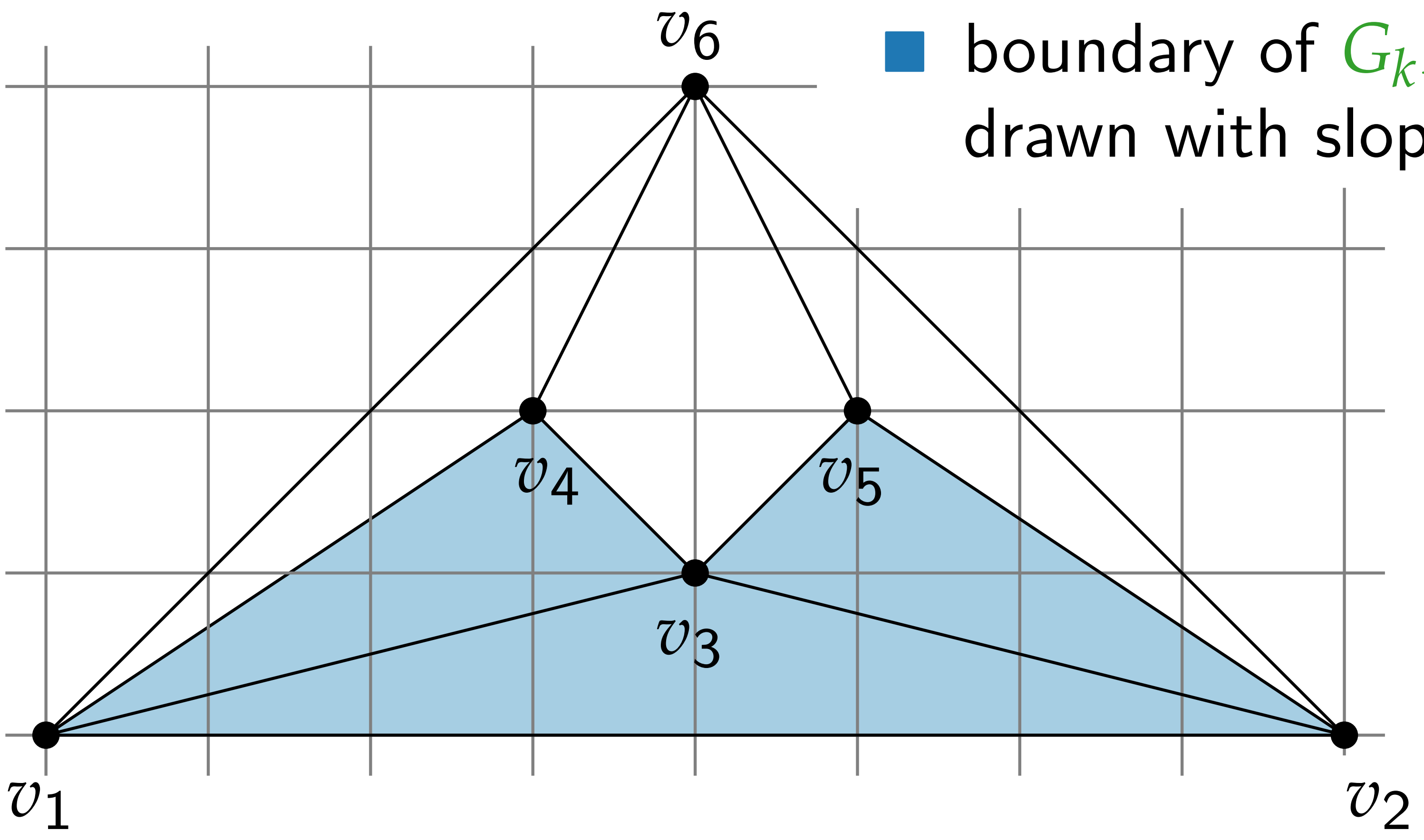
Remarks:

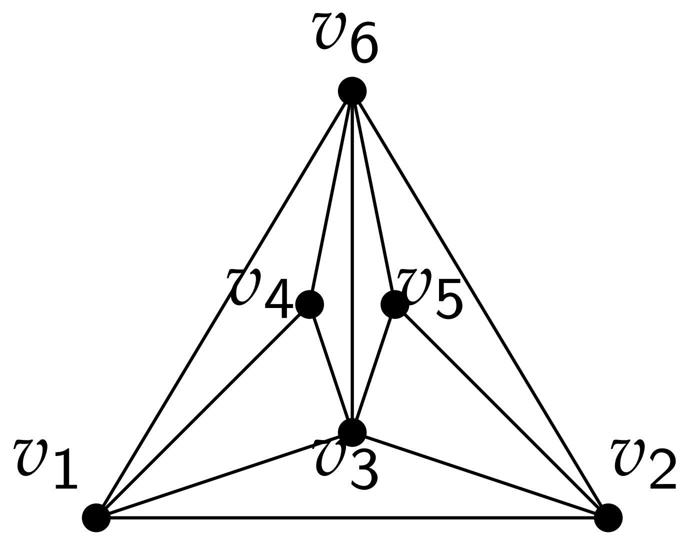
- 2 shifts per step
- width $< 2n$
- height $< n$

Constraints:

G_{k-1} is drawn such that

- v_1 is leftmost vertex, v_2 is rightmost vertex,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn x-monotone,
- boundary of G_{k-1} (minus edge (v_1, v_2)) is drawn with slope ± 1,



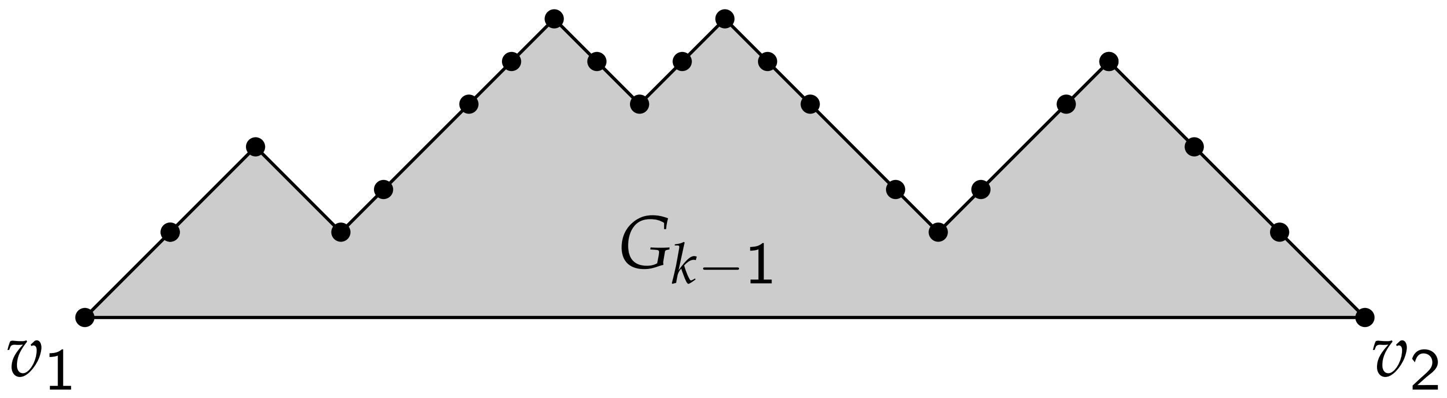


Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

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

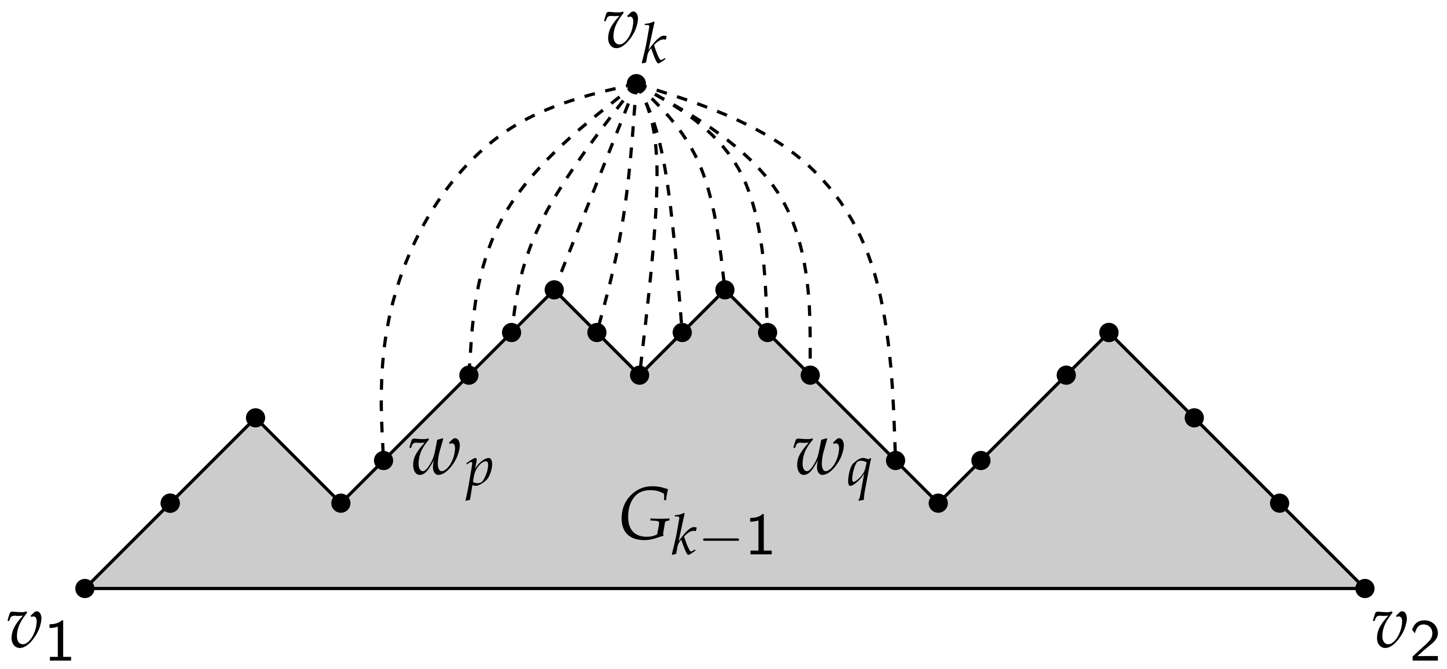


Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

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

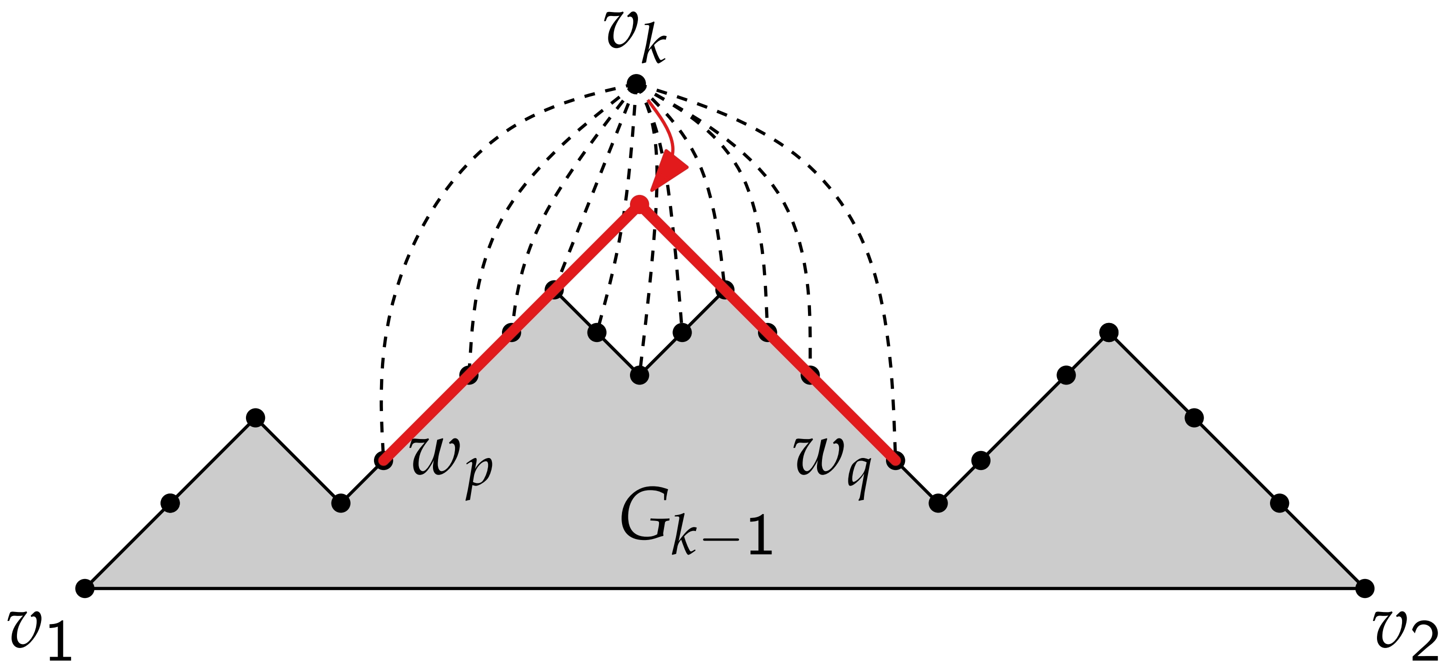


Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

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

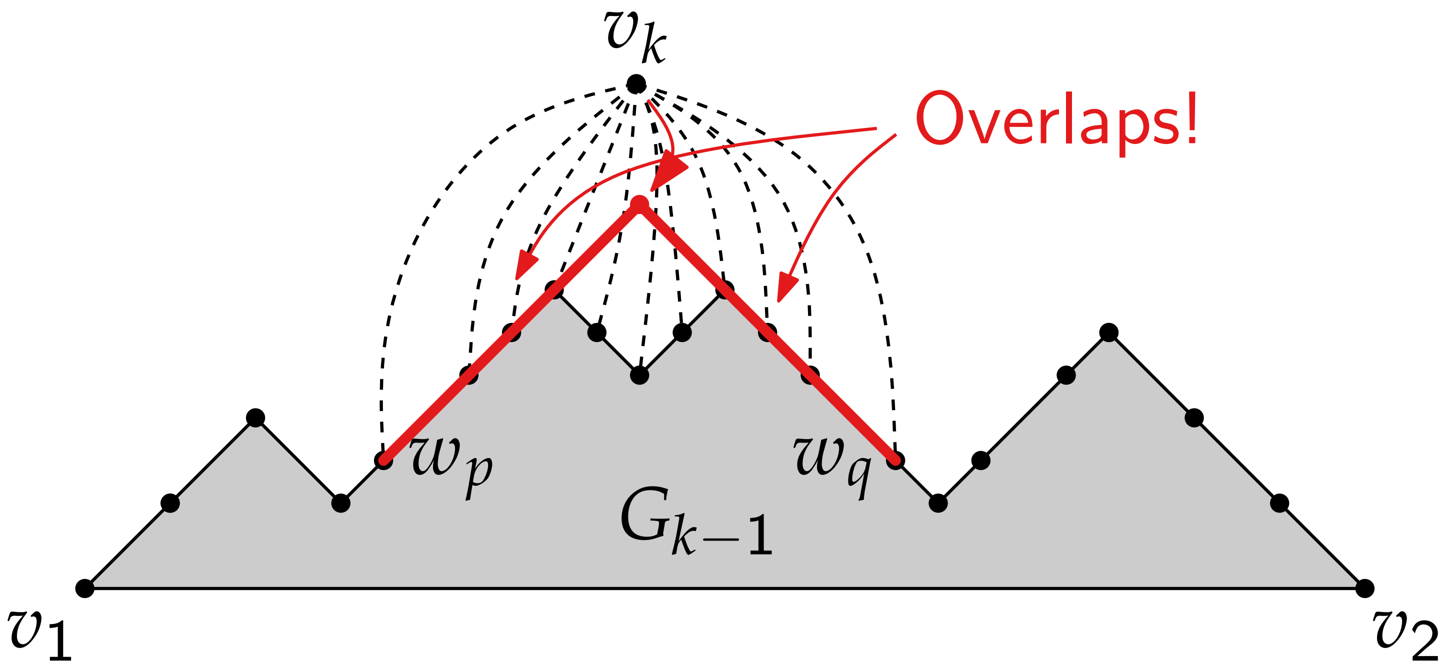


Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

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

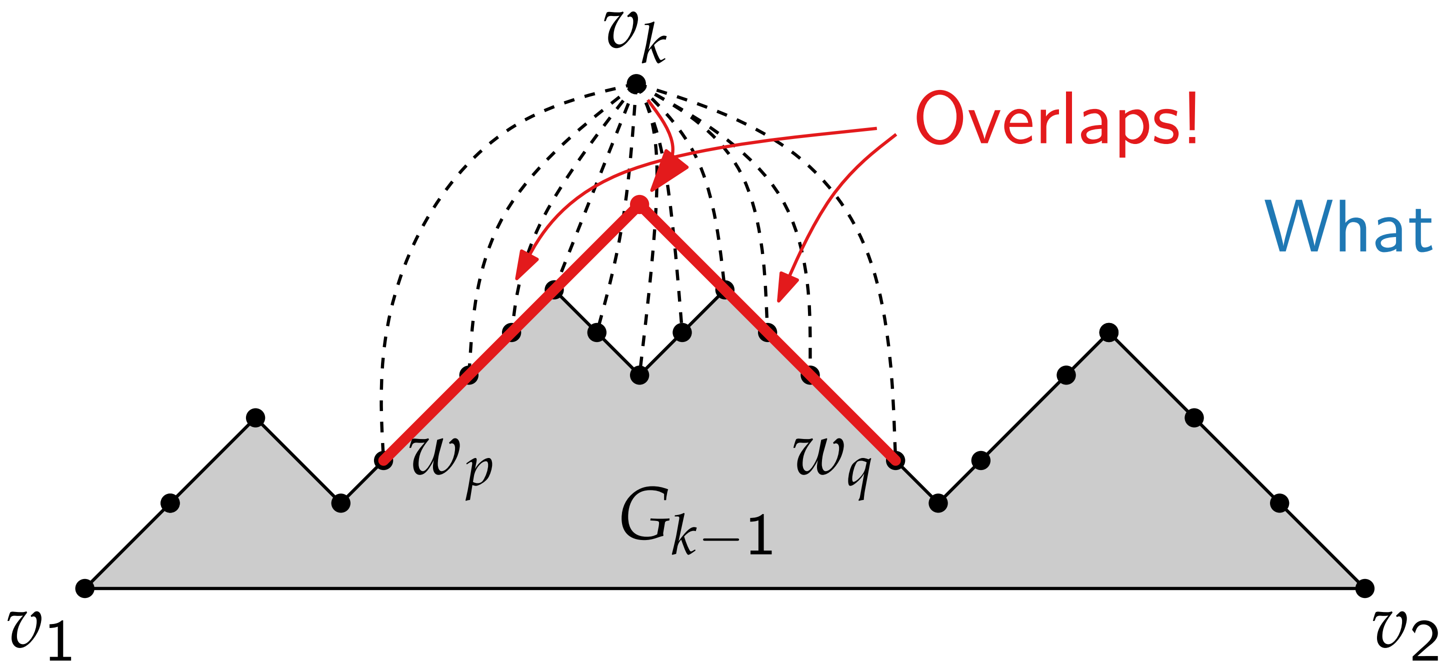


Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

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



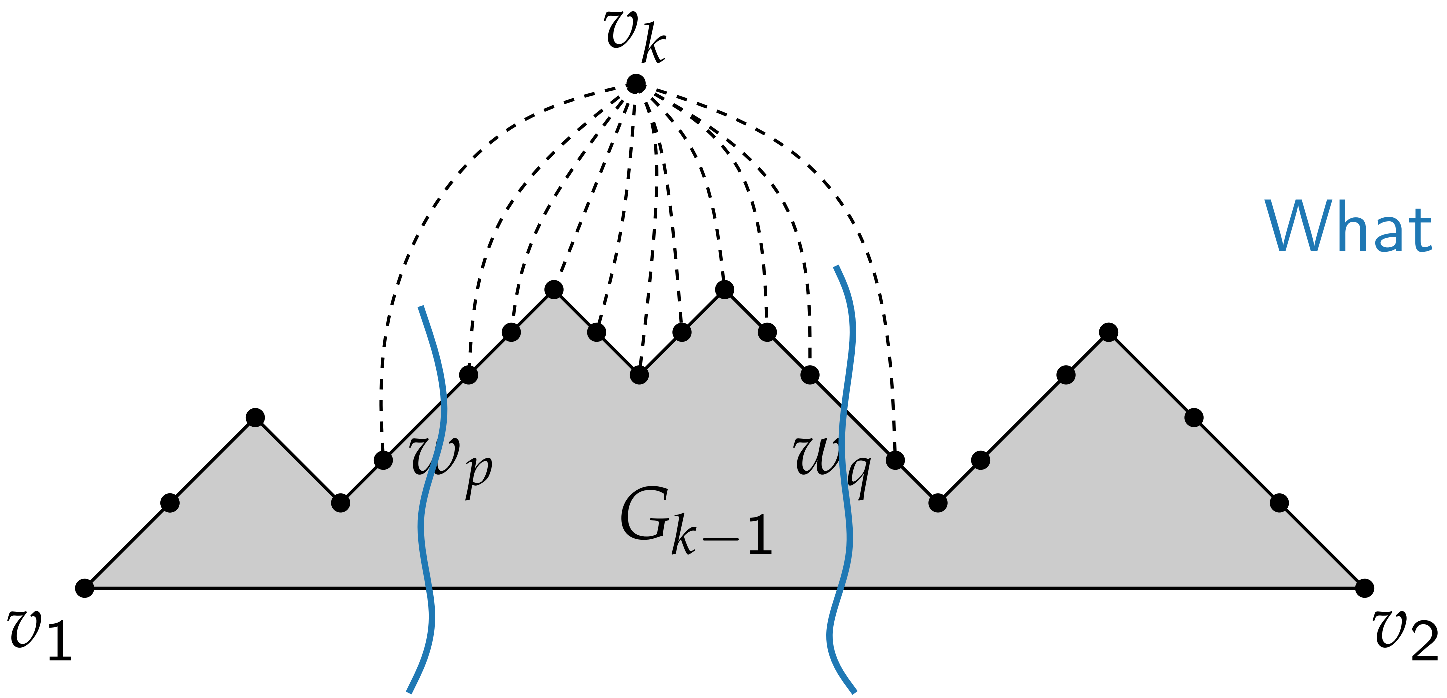
What is the solution?

Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

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



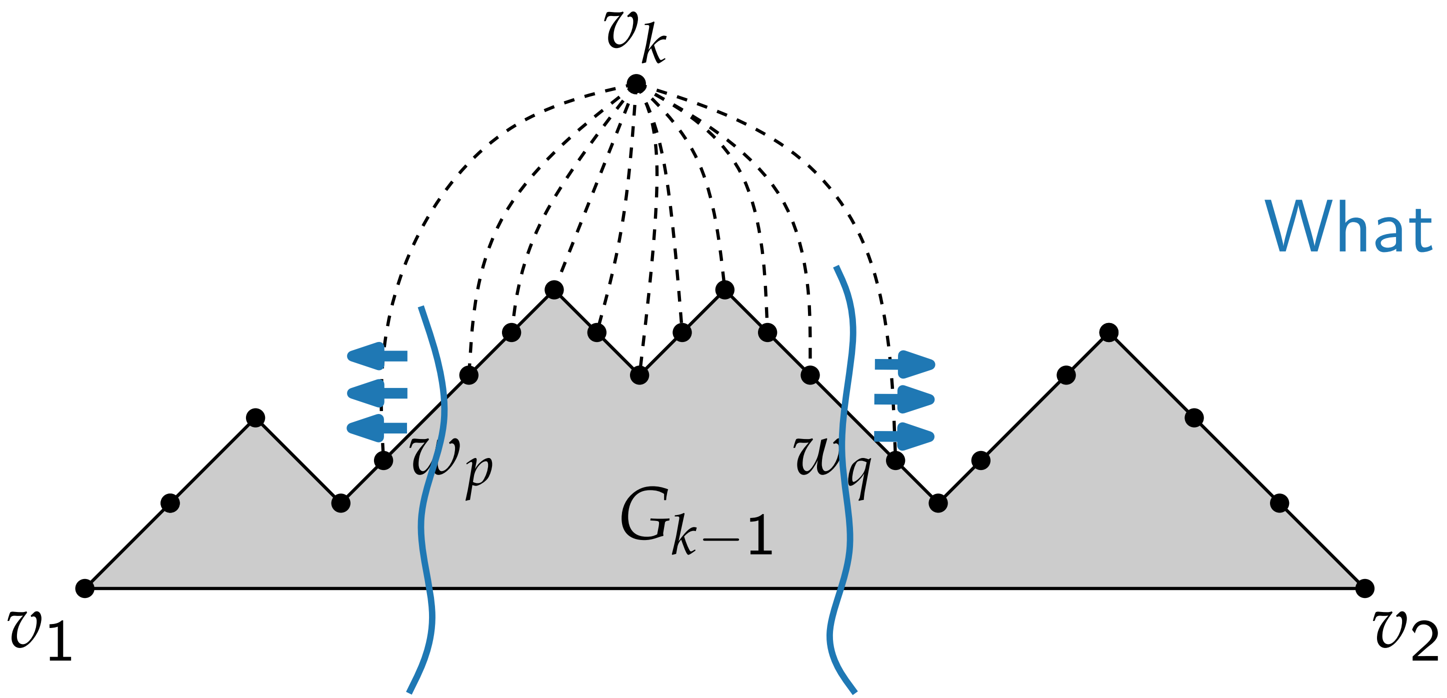
What is the solution?

Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

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



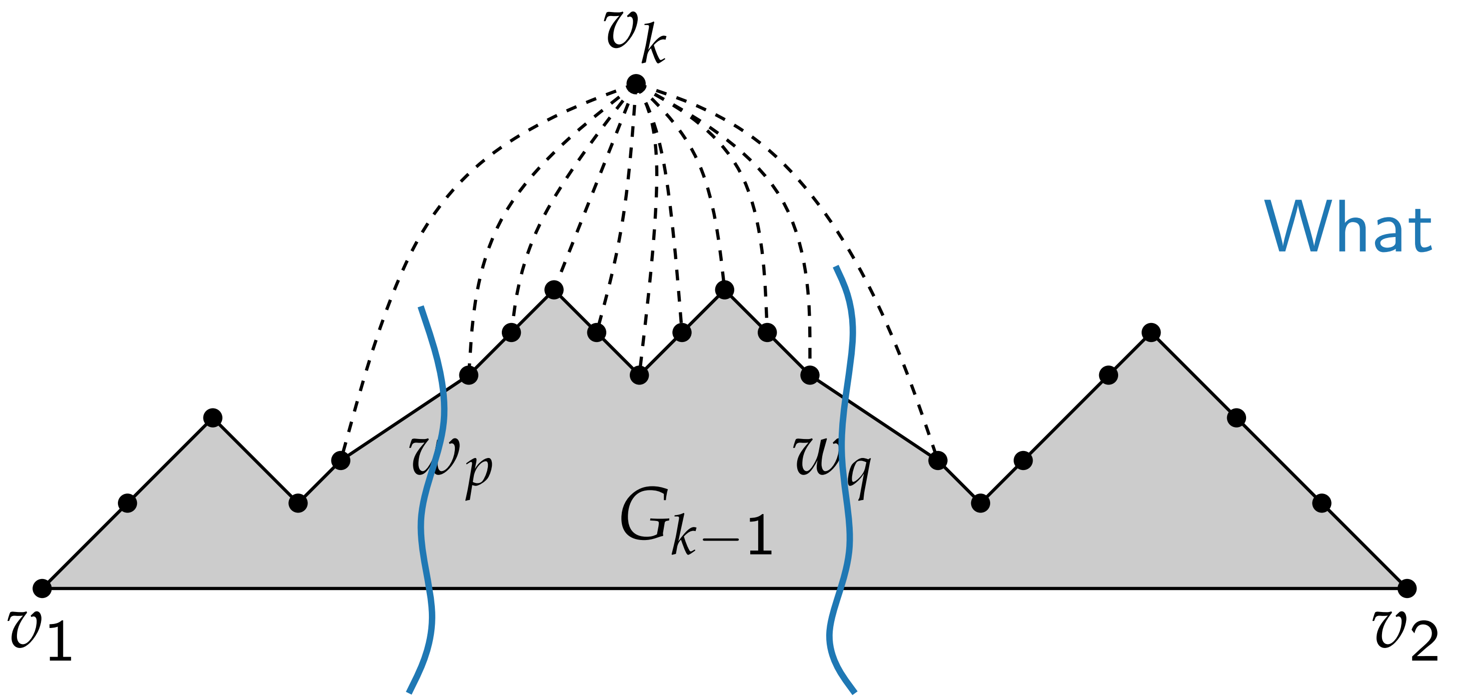
What is the solution?

Shift method

Algorithm invariants/constraints:

G_{k-1} is drawn such that

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



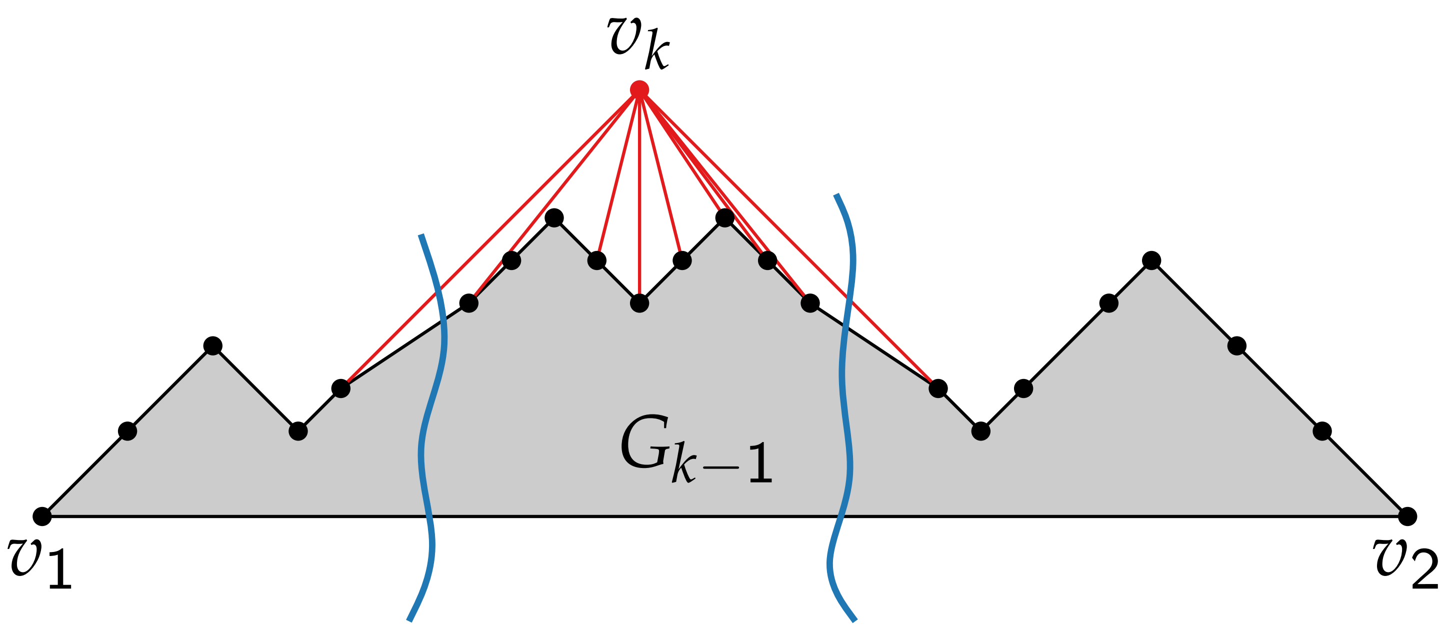
What is the solution?

Shift method

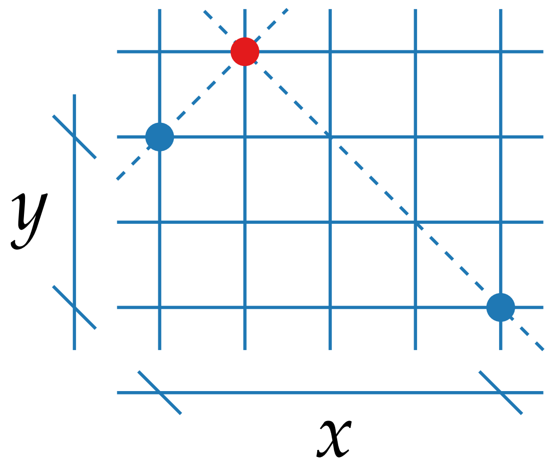
Algorithm invariants/constraints:

G_{k-1} is drawn such that

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



Shift method

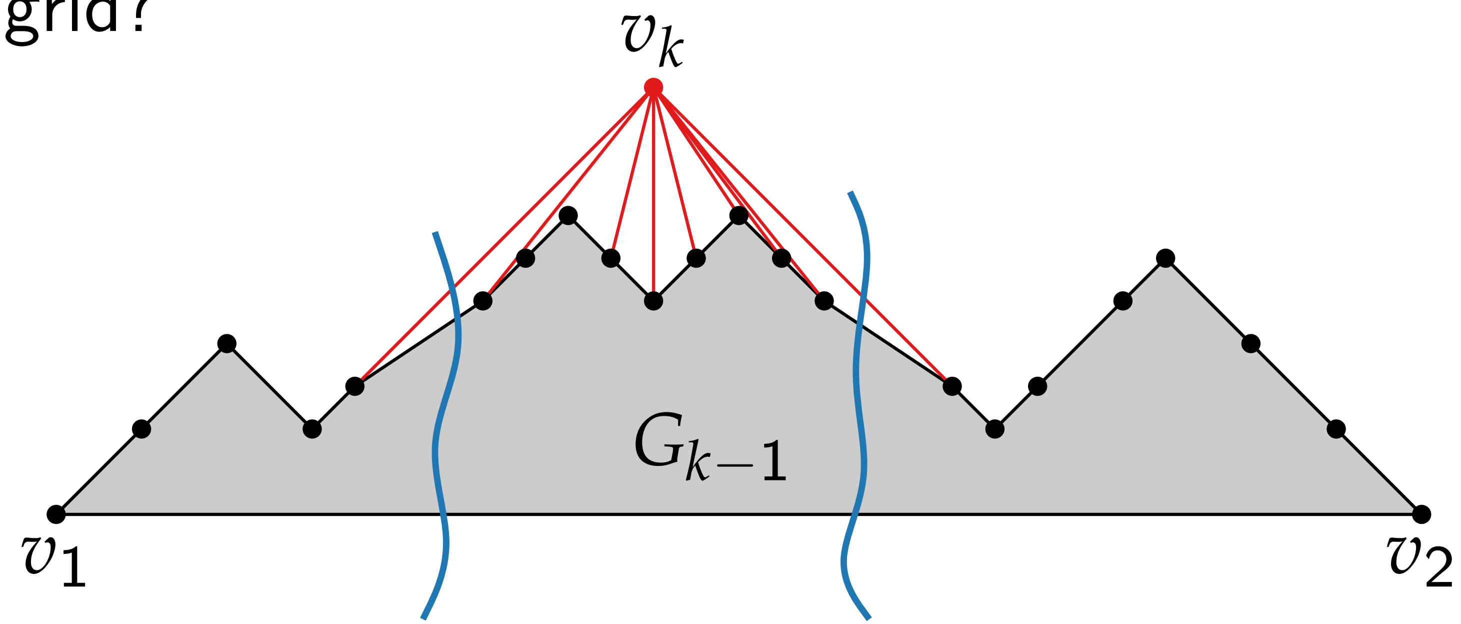


Algorithm invariants/constraints:

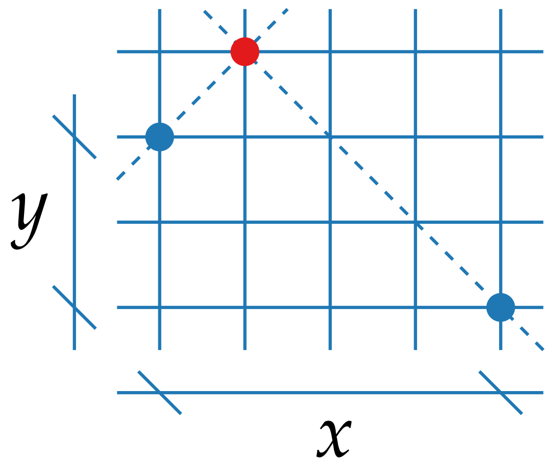
G_{k-1} is drawn such that

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

- Why is v_k on grid?



Shift method

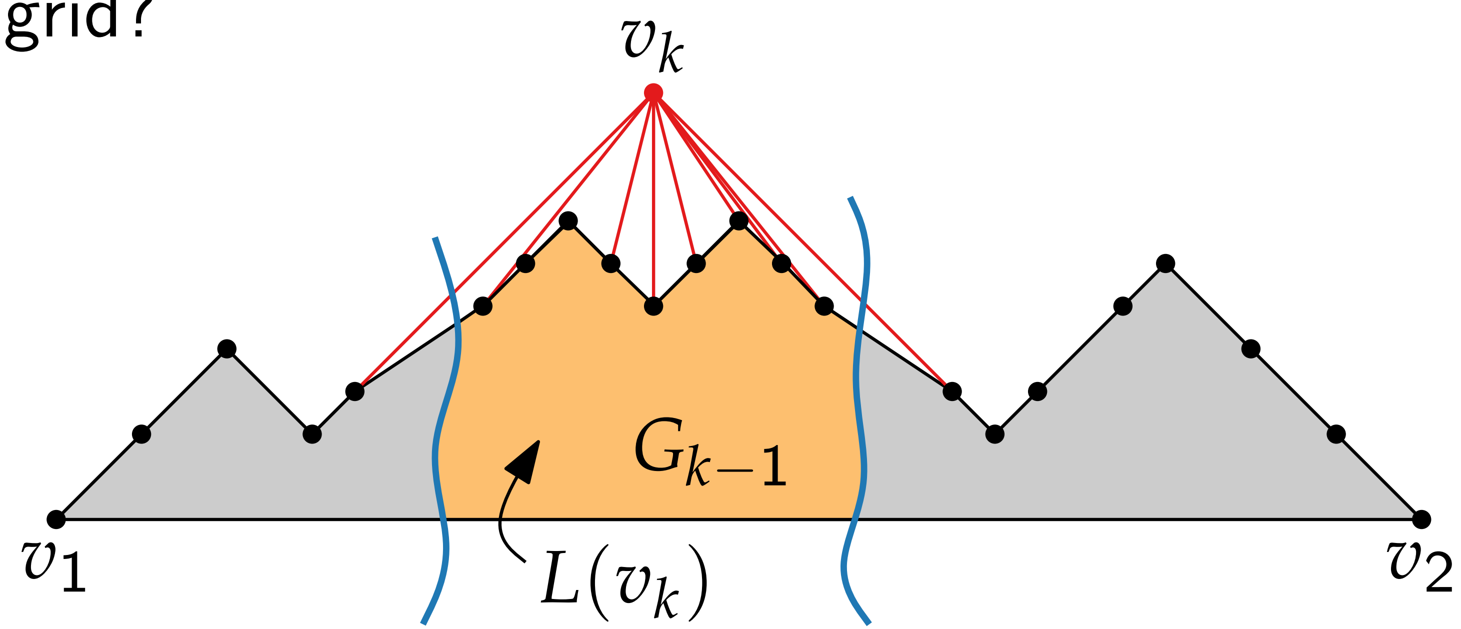


Algorithm invariants/constraints:

G_{k-1} is drawn such that

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

- Why is v_k on grid?



Shift method

Lemma.
Every two vertices on the outerface of G_{k-1} have even Manhattan distance.

Shift method

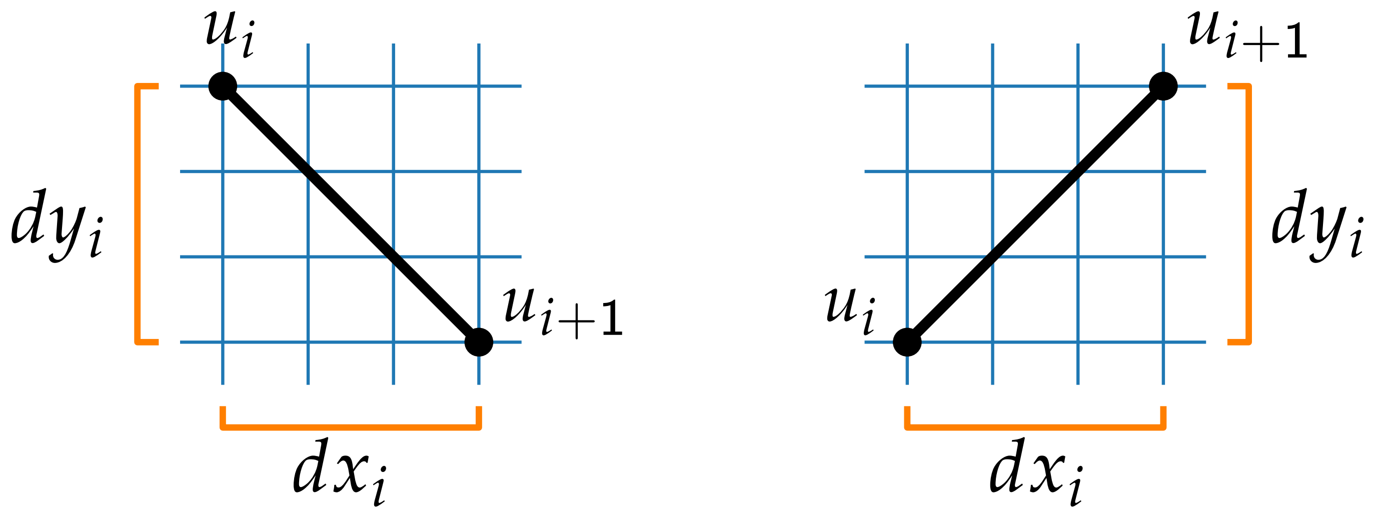
Lemma.
Every two vertices on the outerface of G_{k-1} have even Manhattan distance.

- u_i and u_{i+1} consecutive on the outerface of G_{k-1}

Shift method

Lemma.
Every two vertices on the outerface of G_{k-1} have even Manhattan distance.

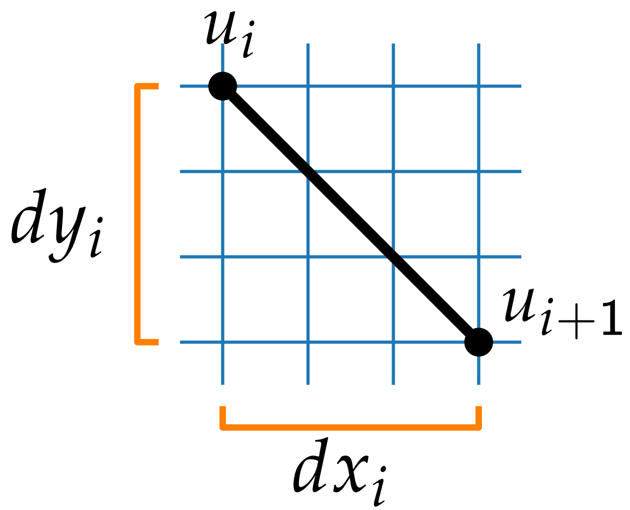
- u_i and u_{i+1} consecutive on the outerface of G_{k-1}

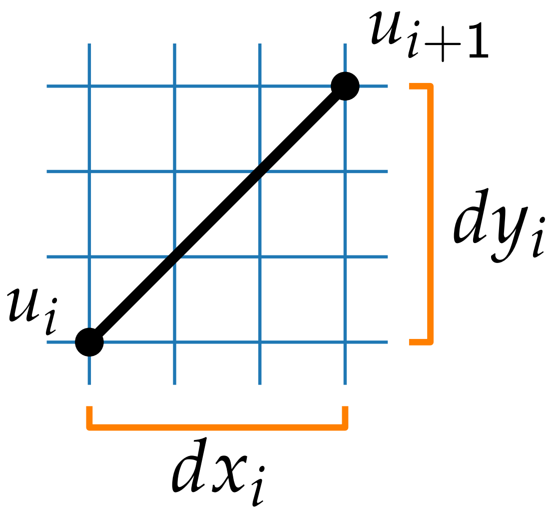


Shift method

Lemma.
Every two vertices on the outerface of G_{k-1} have even Manhattan distance.

- u_i and u_{i+1} consecutive on the outerface of G_{k-1}





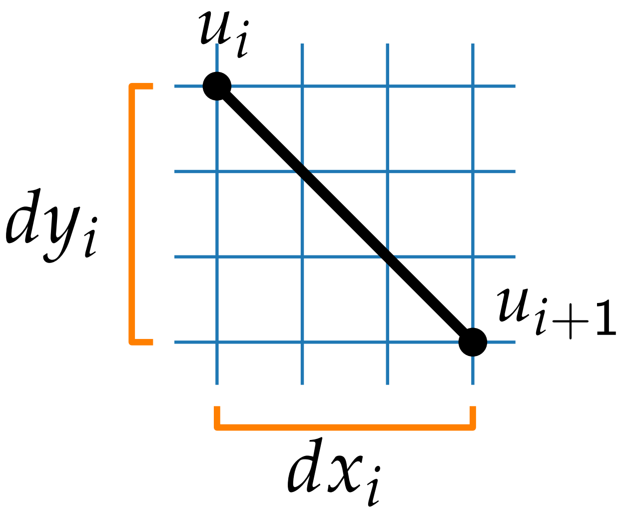
$d(u_i, u_{i+1}) = |dx_i| + |dy_i|$ even

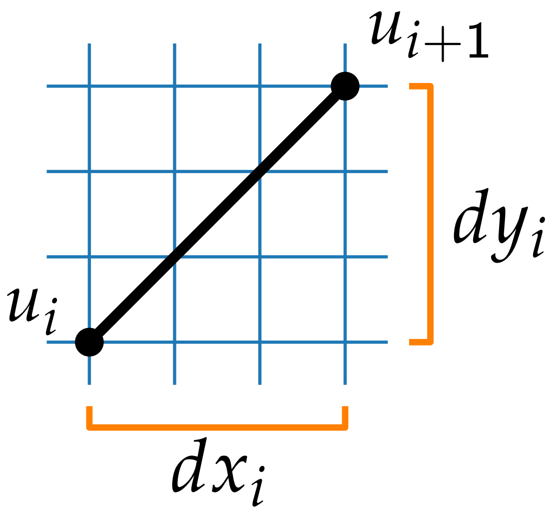
$|dx_i| \pm |dy_i|$ even

Shift method

Lemma.
Every two vertices on the outerface of G_{k-1} have even Manhattan distance.

- u_i and u_{i+1} consecutive on the outerface of G_{k-1}





$d(u_i, u_{i+1}) = |dx_i| + |dy_i|$ even

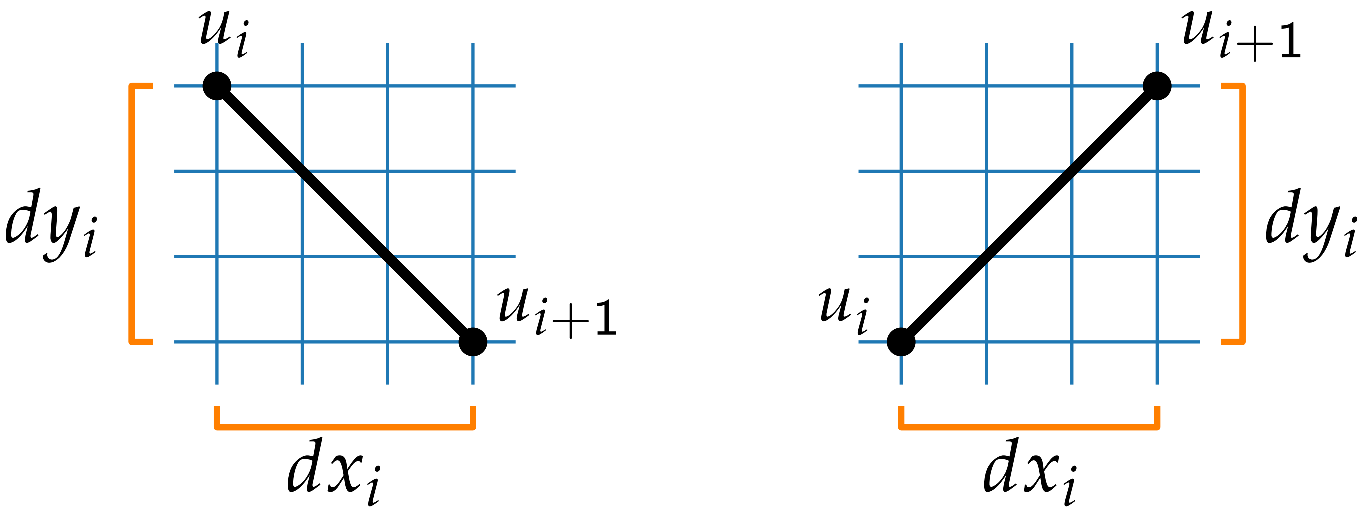
$|dx_i| \pm |dy_i|$ even

- u_i, $u_{i+\ell}$ on the outerface of G_{k-1}

Shift method

Lemma.
Every two vertices on the outerface of G_{k-1} have even Manhattan distance.

- u_i and u_{i+1} consecutive on the outerface of G_{k-1}



$d(u_i, u_{i+1}) = |dx_i| + |dy_i|$ even

$|dx_i| \pm |dy_i|$ even

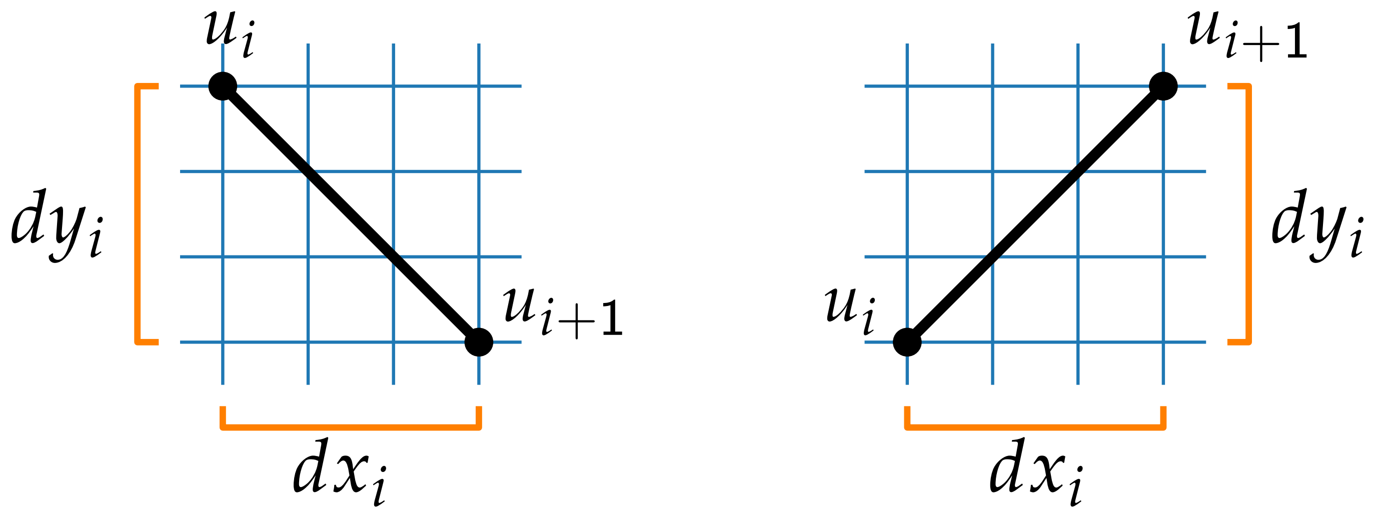
- u_i, $u_{i+\ell}$ on the outerface of G_{k-1}

$$d(u_i, u_\ell) = \sum_{j=i}^{\ell-1} |dx_j| + \lambda_j |dy_j|, \lambda_j = \pm 1 \quad \text{even}$$

Shift method

Lemma.
Every two vertices on the outerface of G_{k-1} have even Manhattan distance.

- u_i and u_{i+1} consecutive on the outerface of G_{k-1}

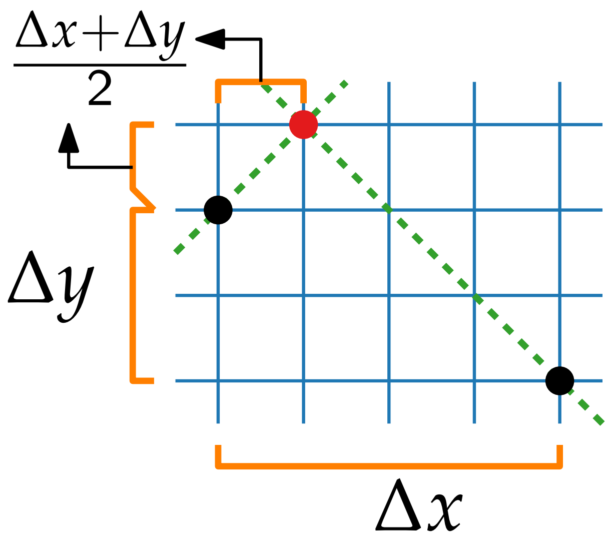


$d(u_i, u_{i+1}) = |dx_i| + |dy_i|$ even

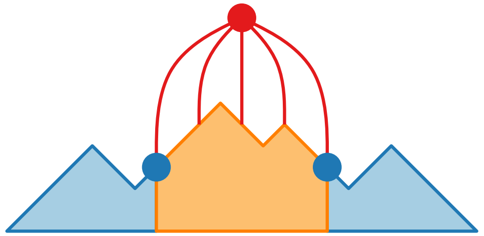
$|dx_i| \pm |dy_i|$ even

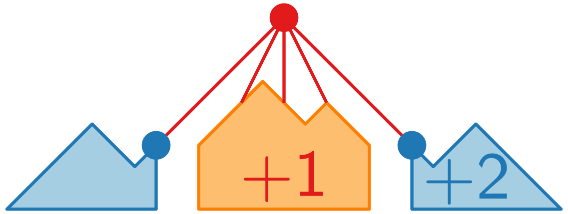
- u_i, $u_{i+\ell}$ on the outerface of G_{k-1}

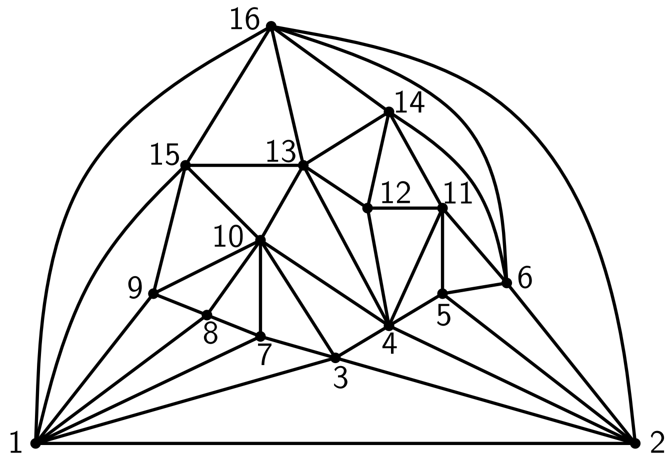
$$d(u_i, u_\ell) = \sum_{j=i}^{\ell-1} |dx_j| + \lambda_j |dy_j|, \lambda_j = \pm 1 \quad \text{even}$$



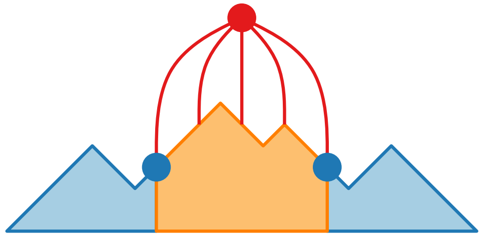
Shift method – example

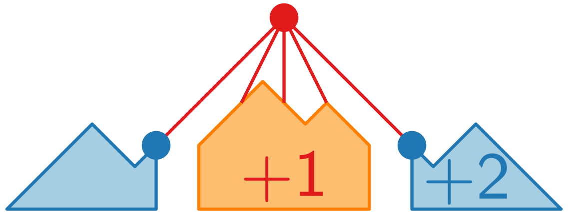


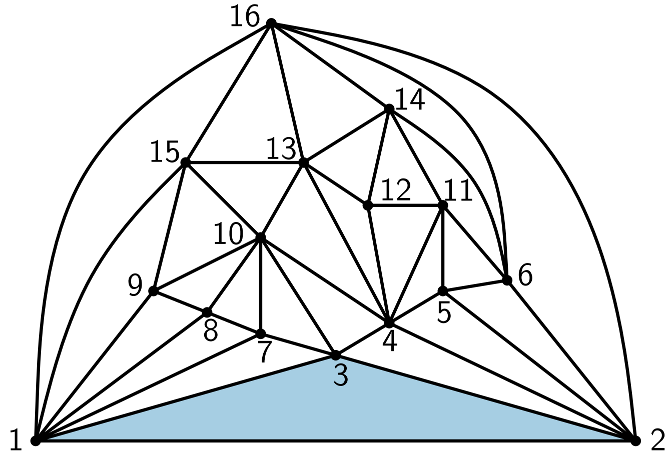


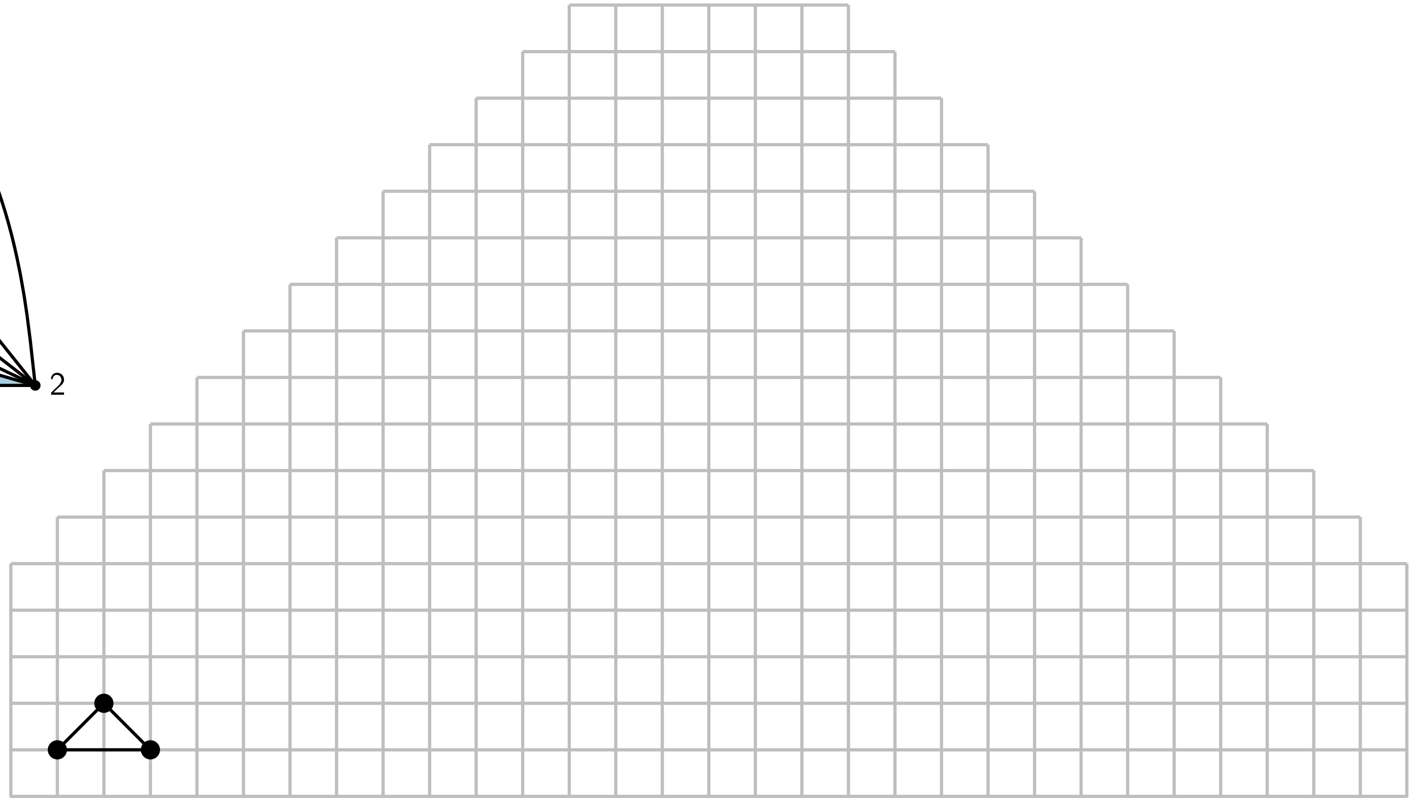


Shift method – example

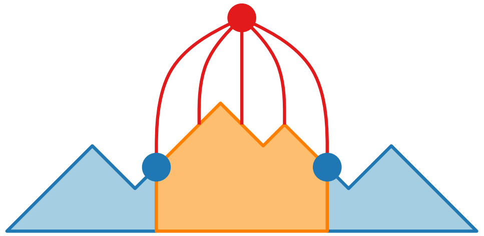


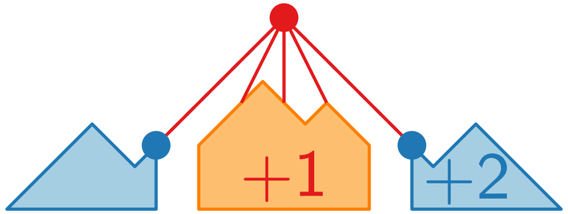


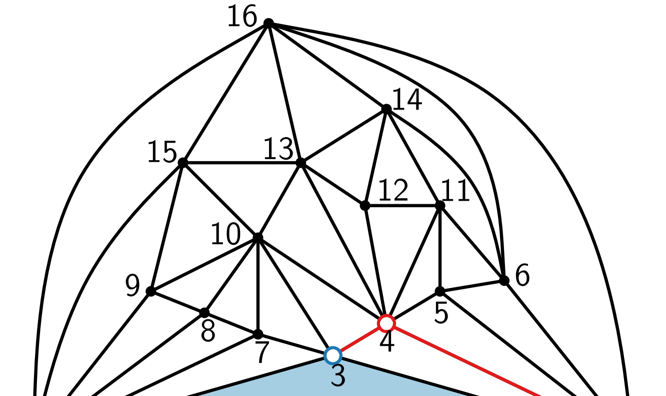


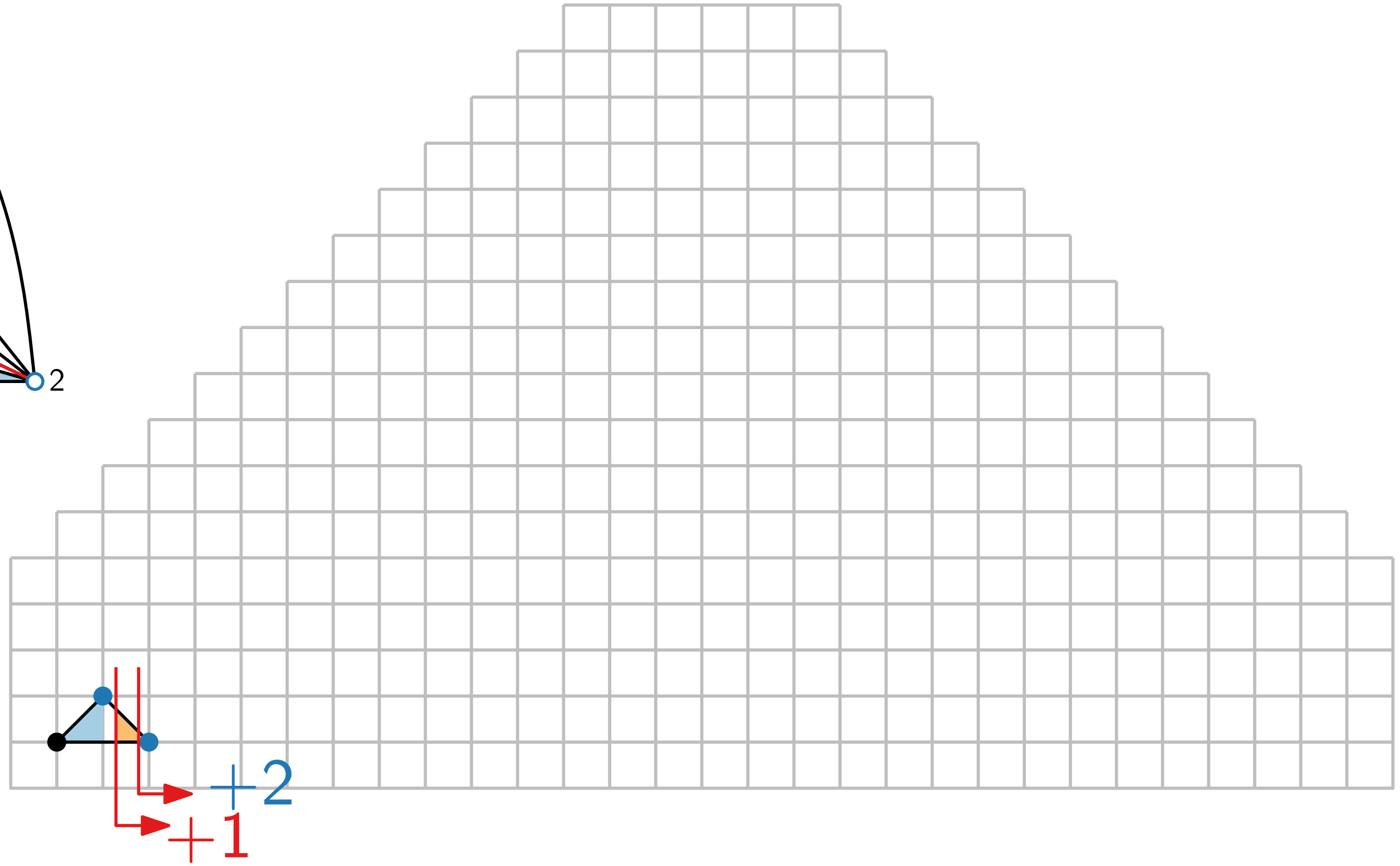


Shift method – example

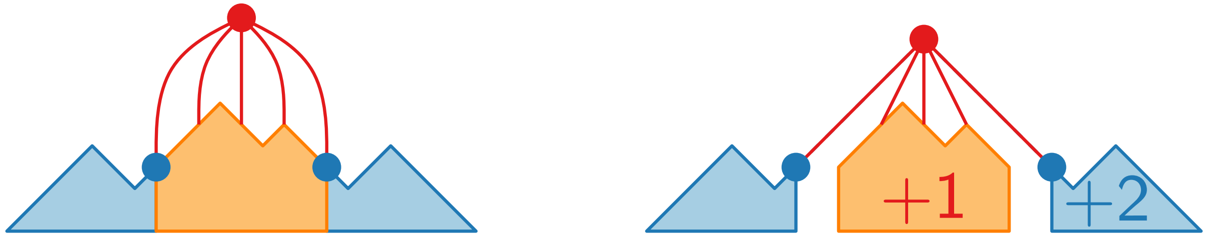


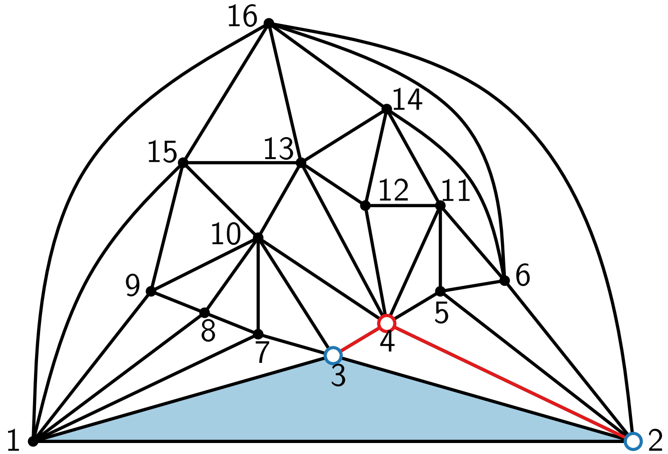


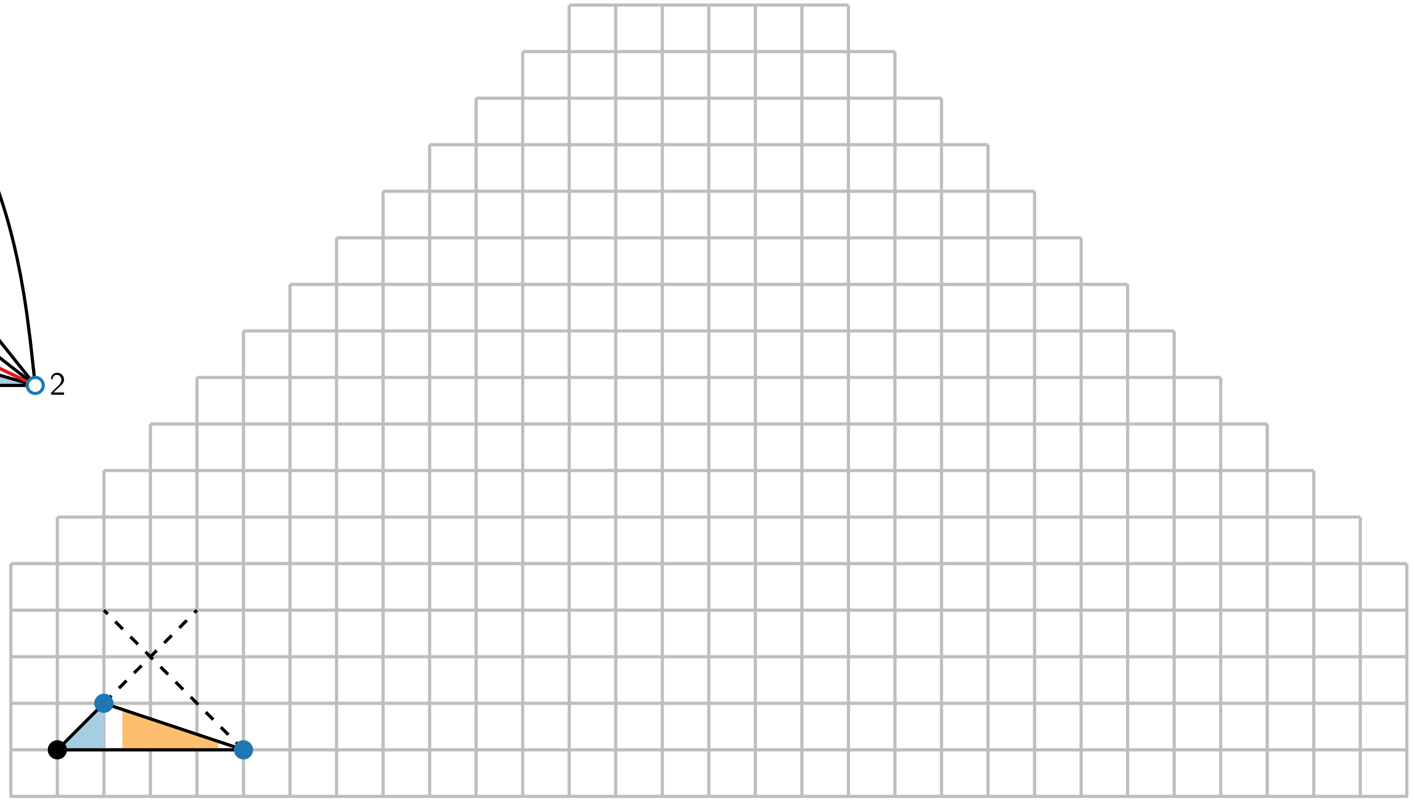




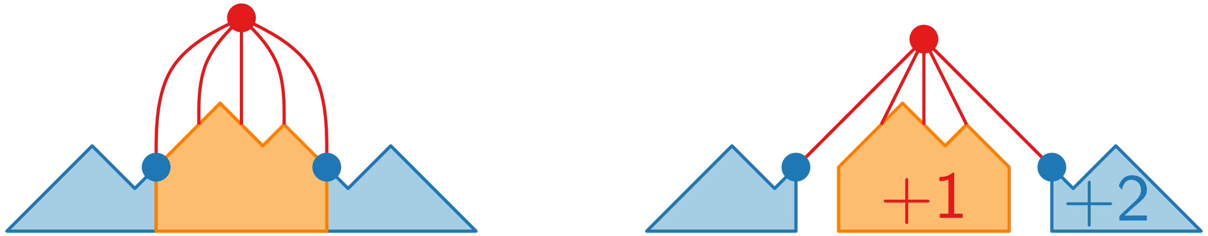
Shift method – example

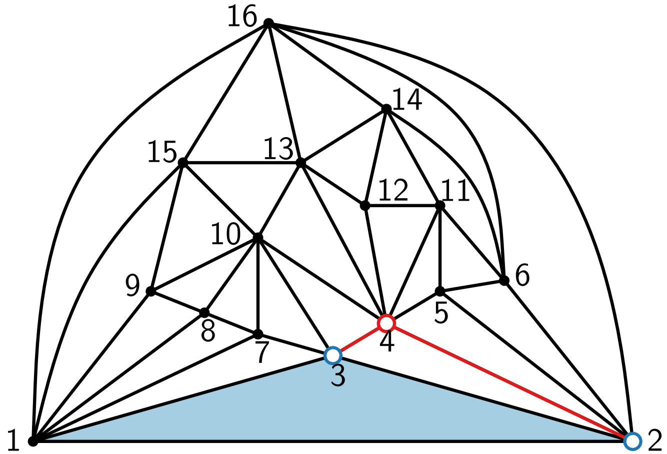


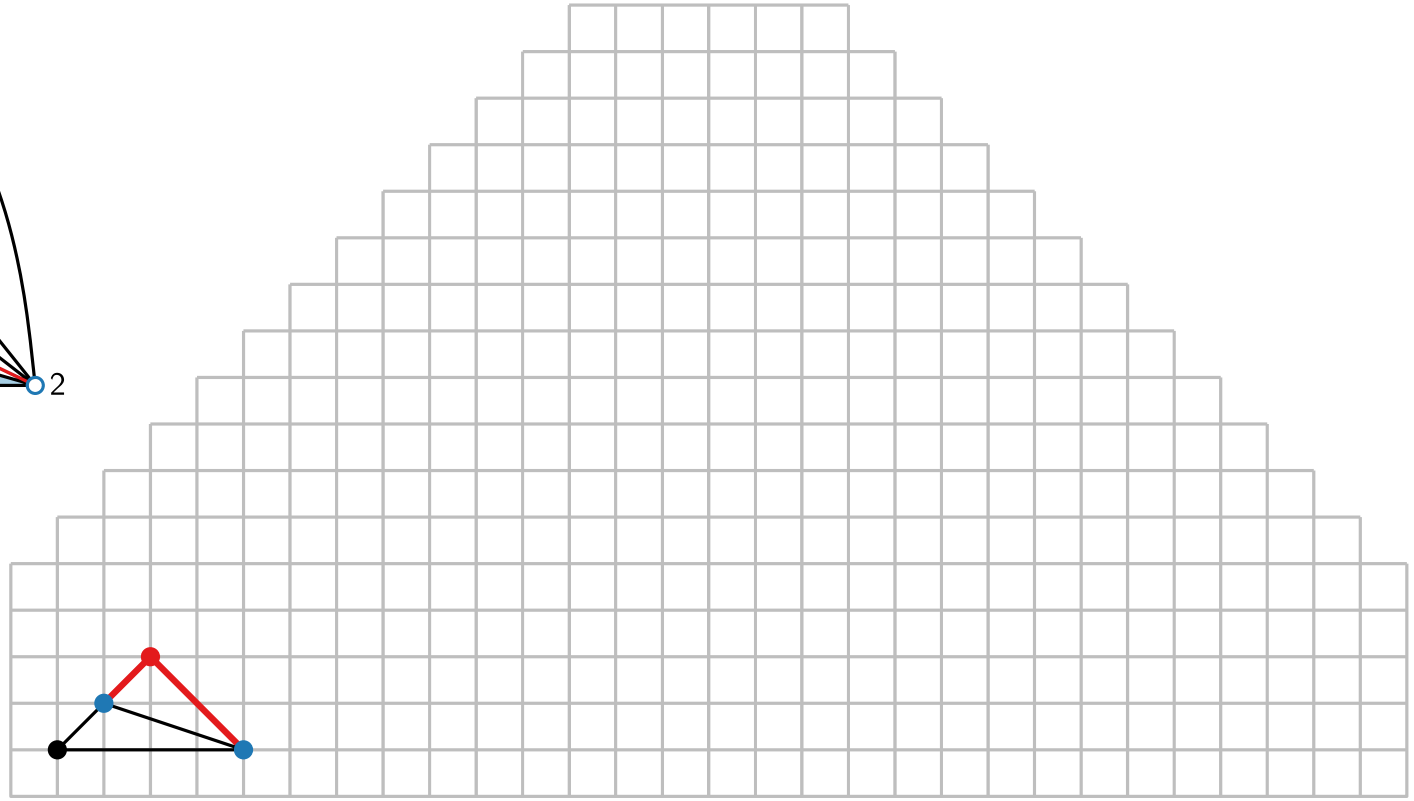




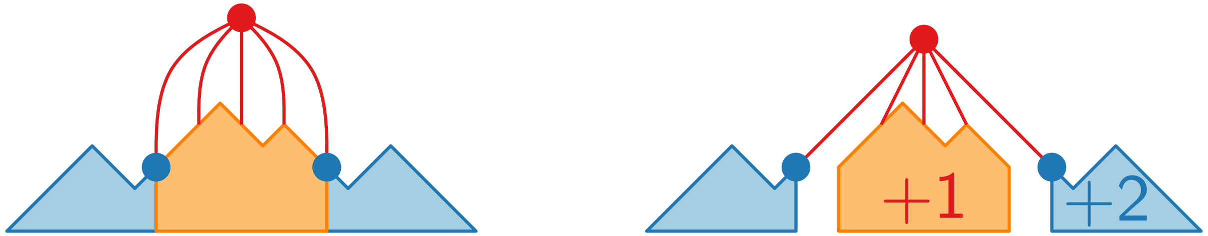
Shift method – example

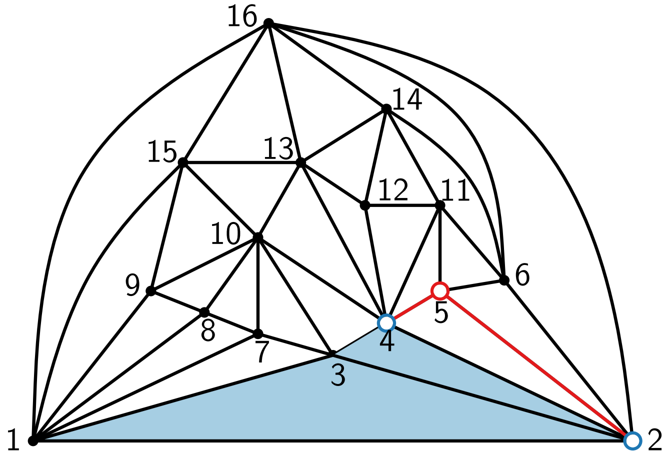


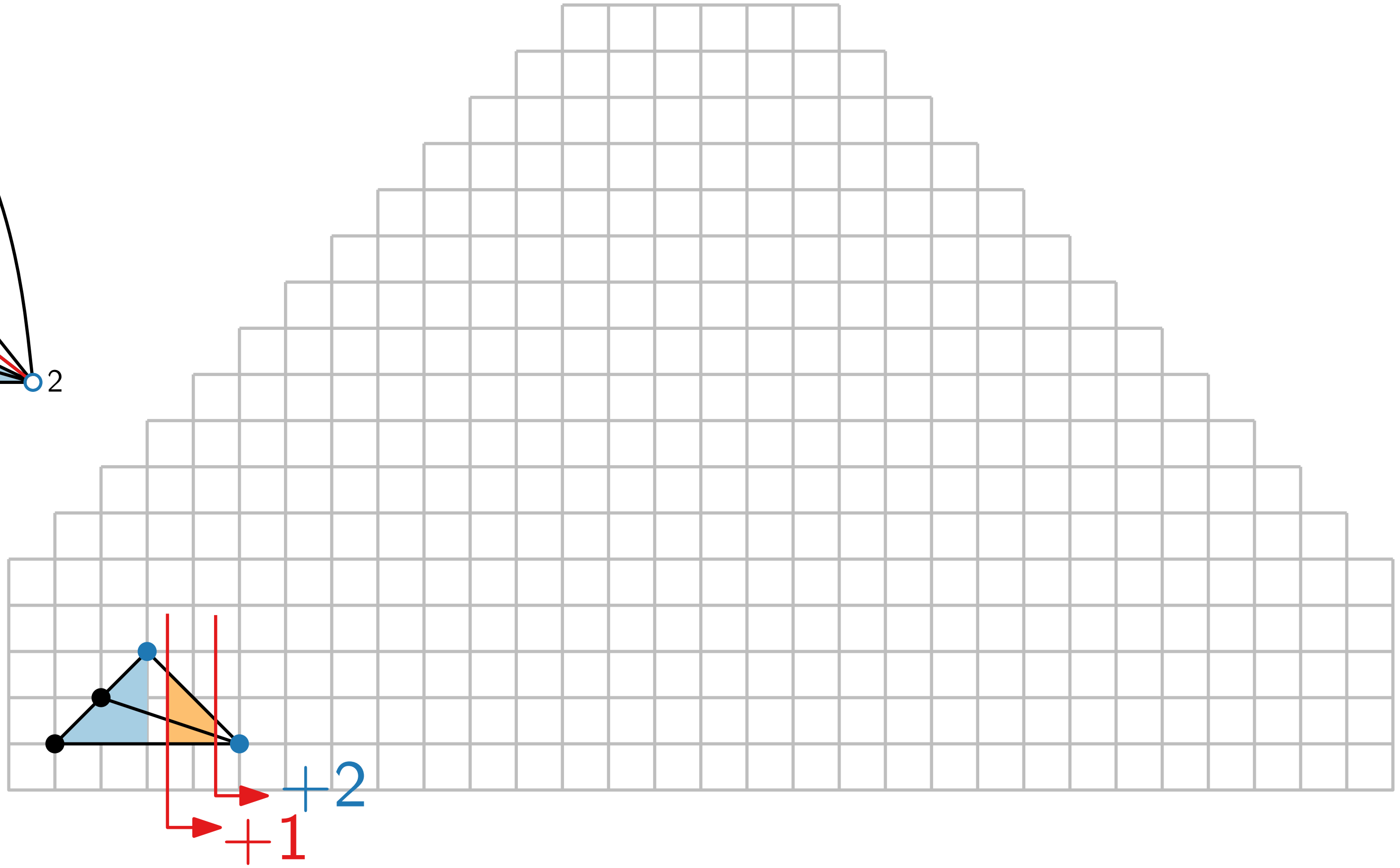




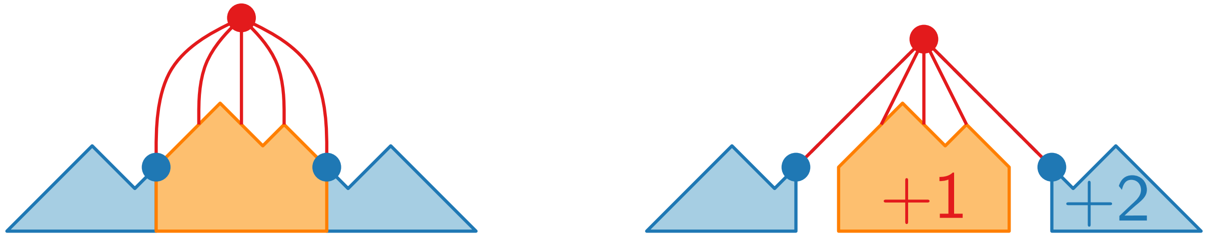
Shift method – example

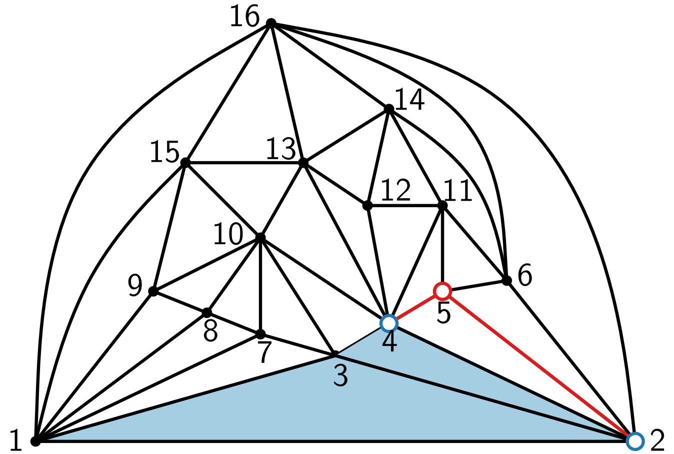


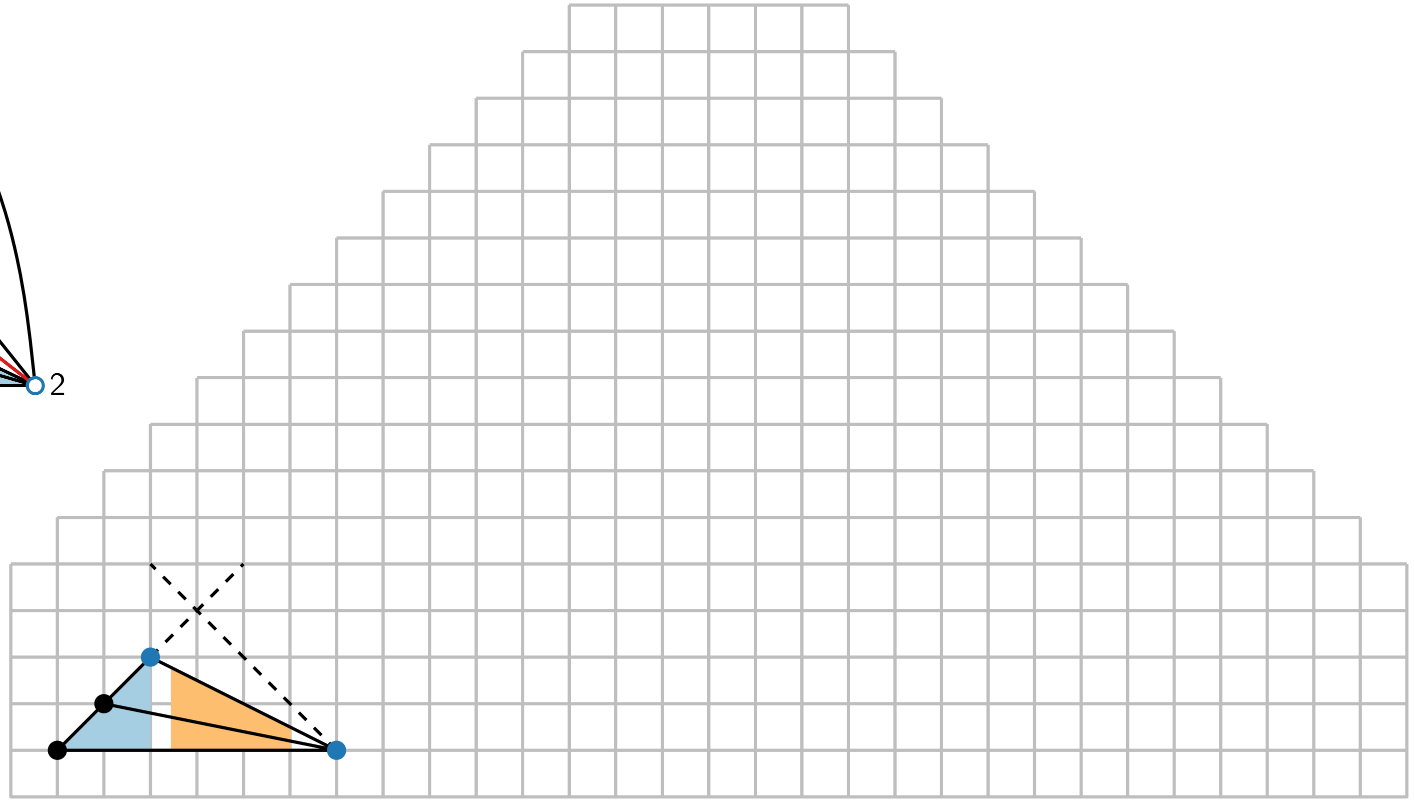




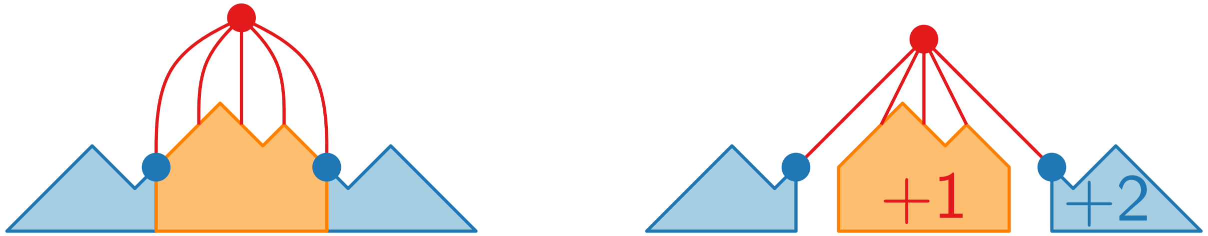
Shift method – example

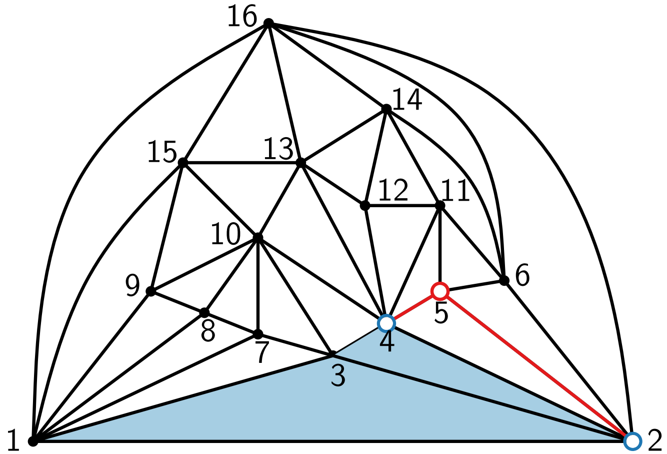


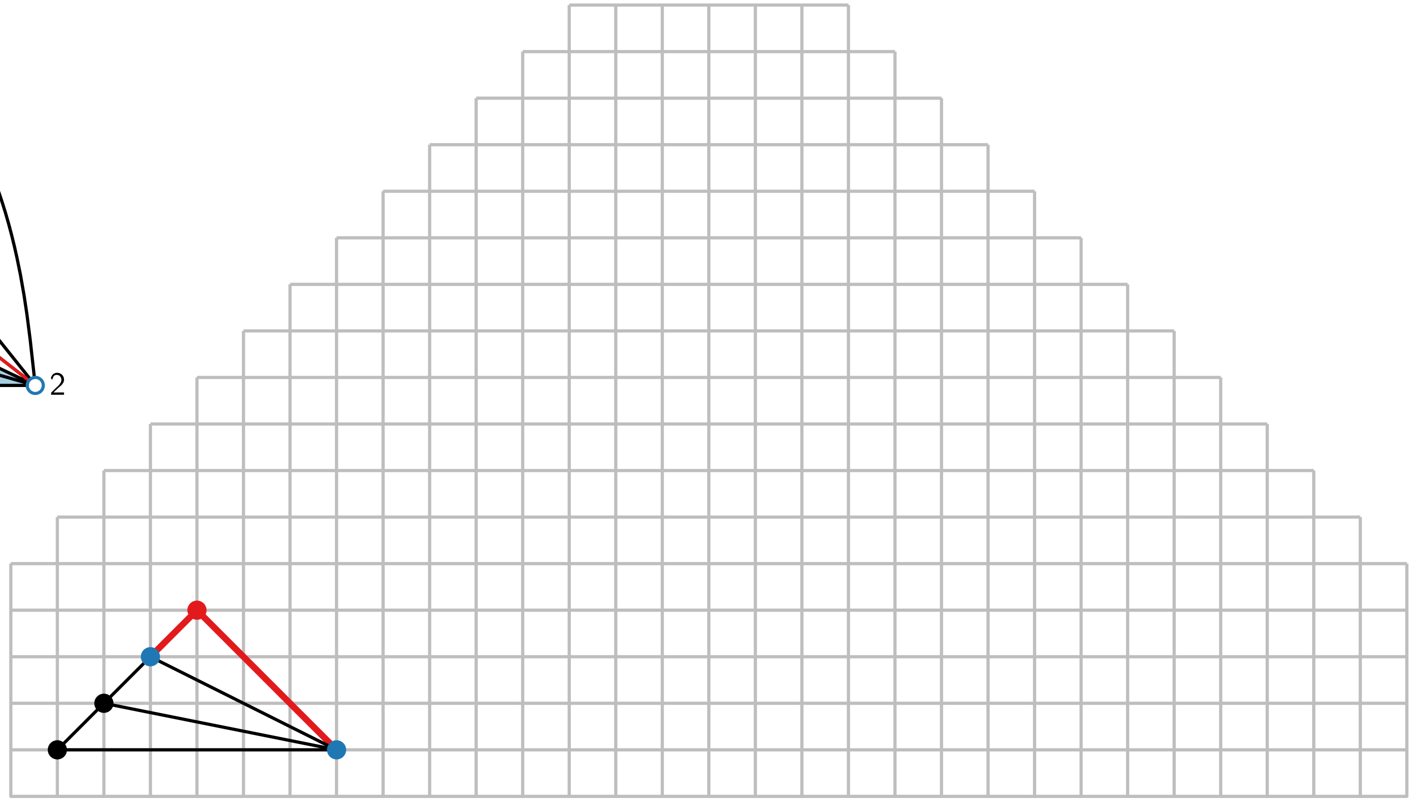




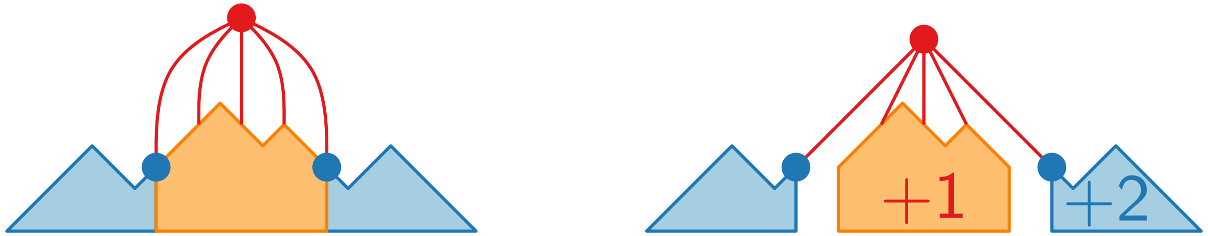
Shift method – example

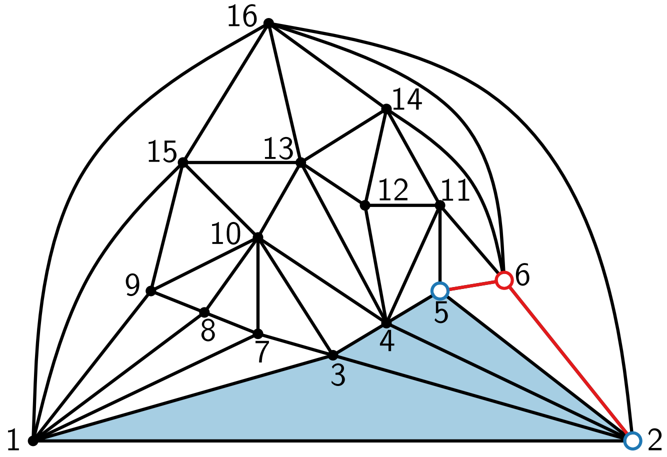


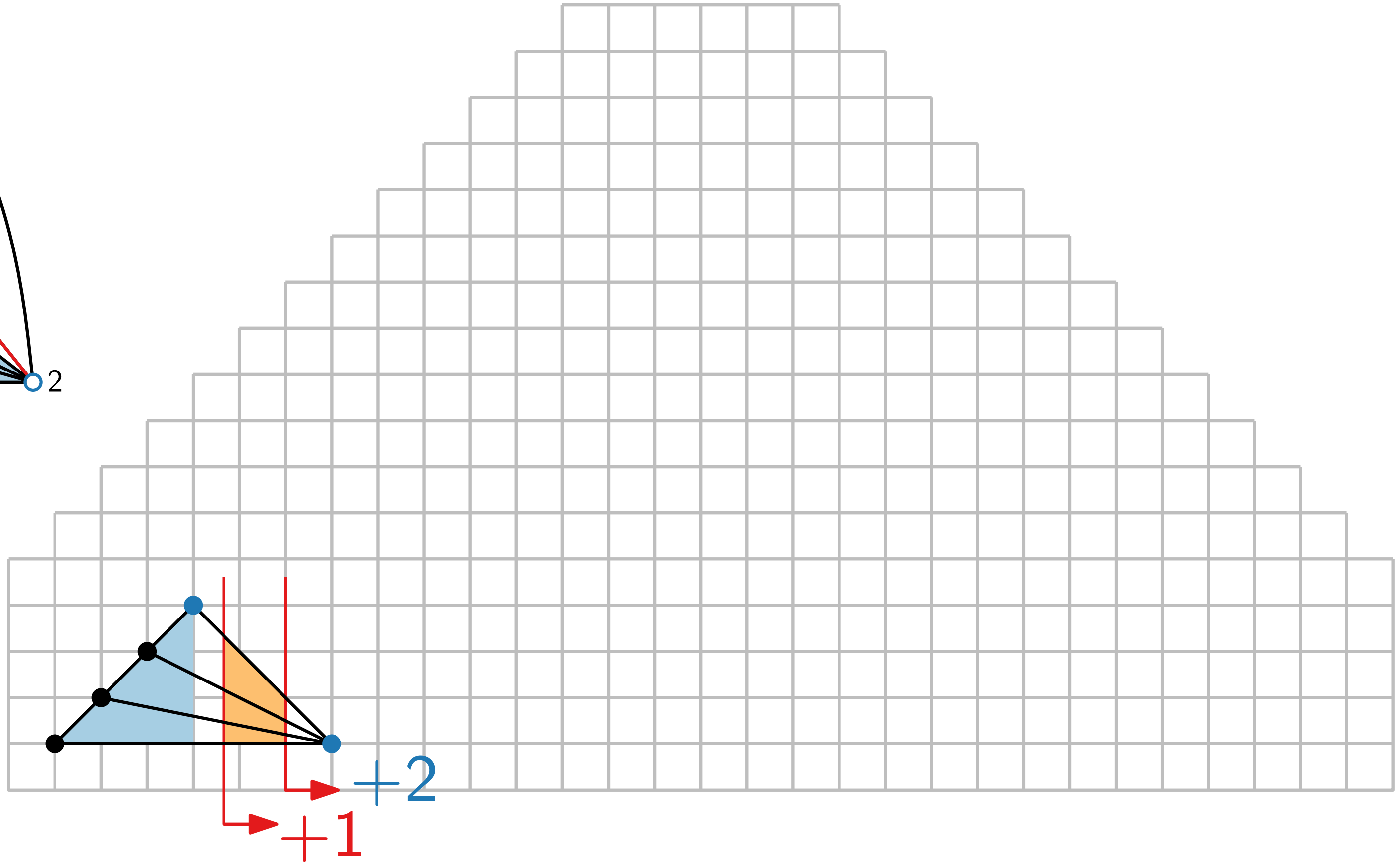




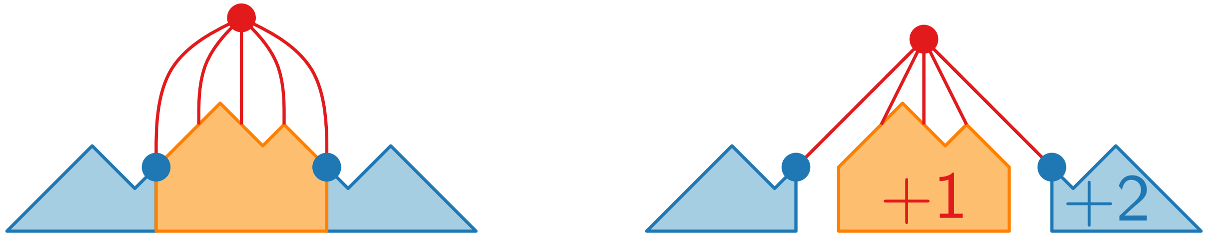
Shift method – example

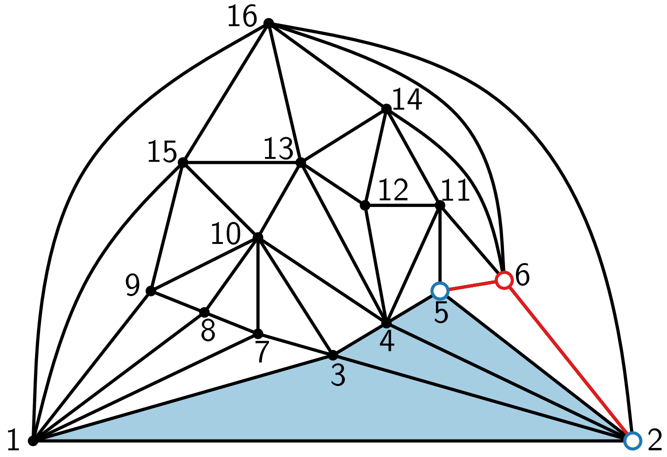


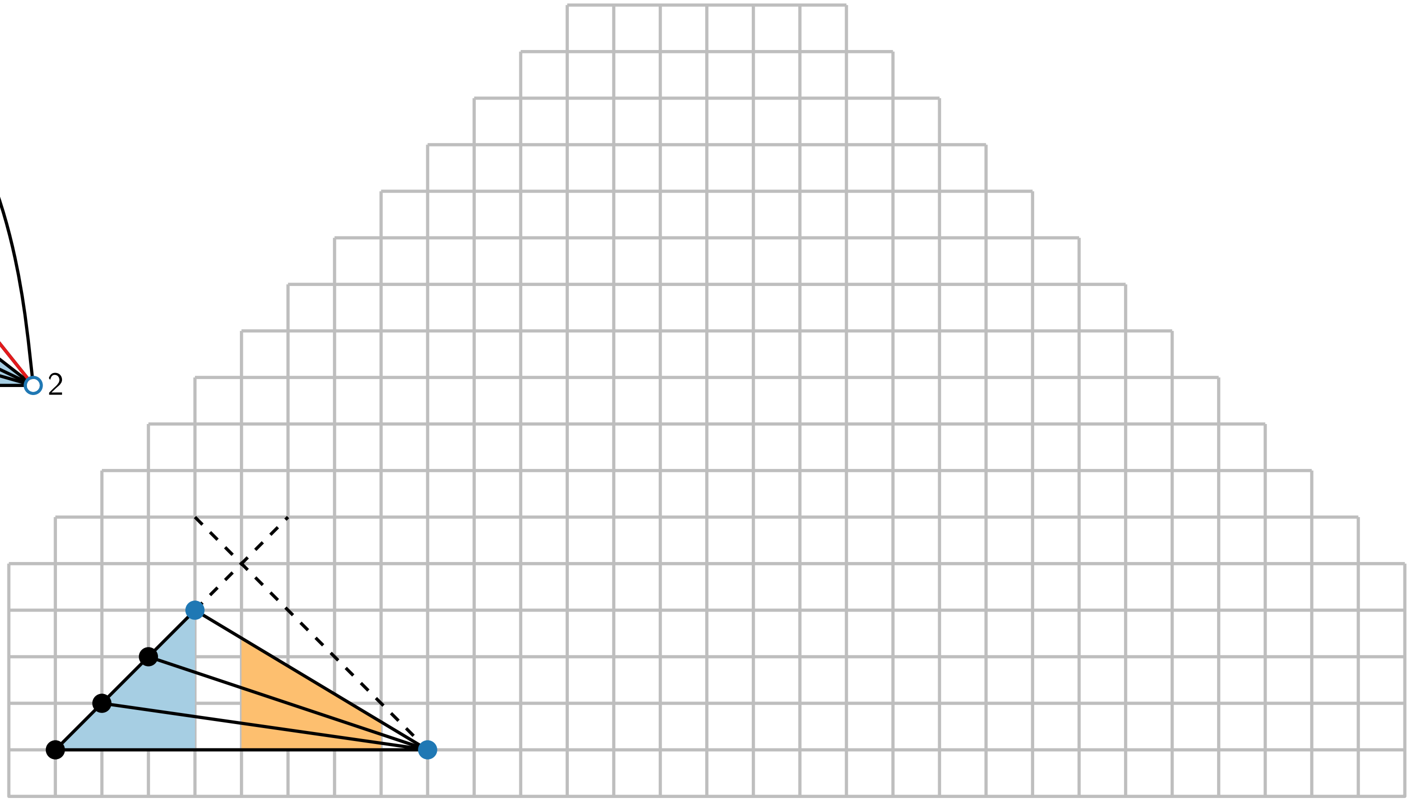




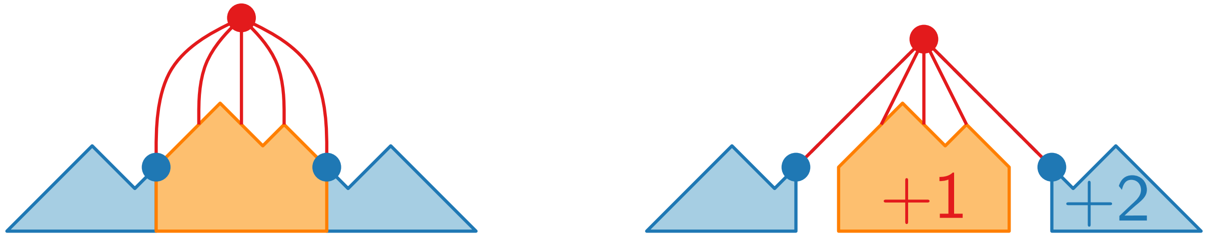
Shift method – example

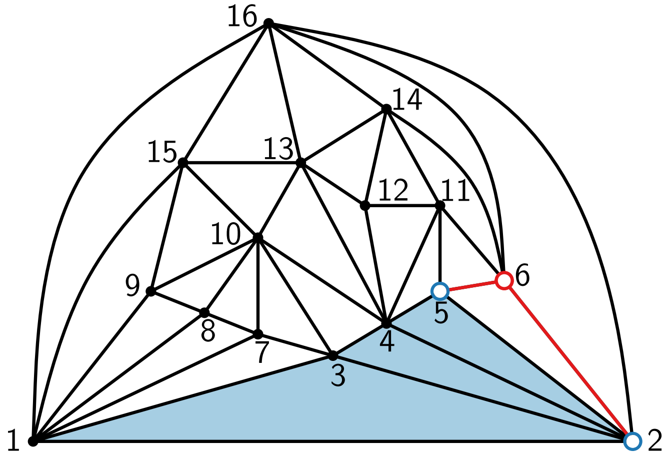


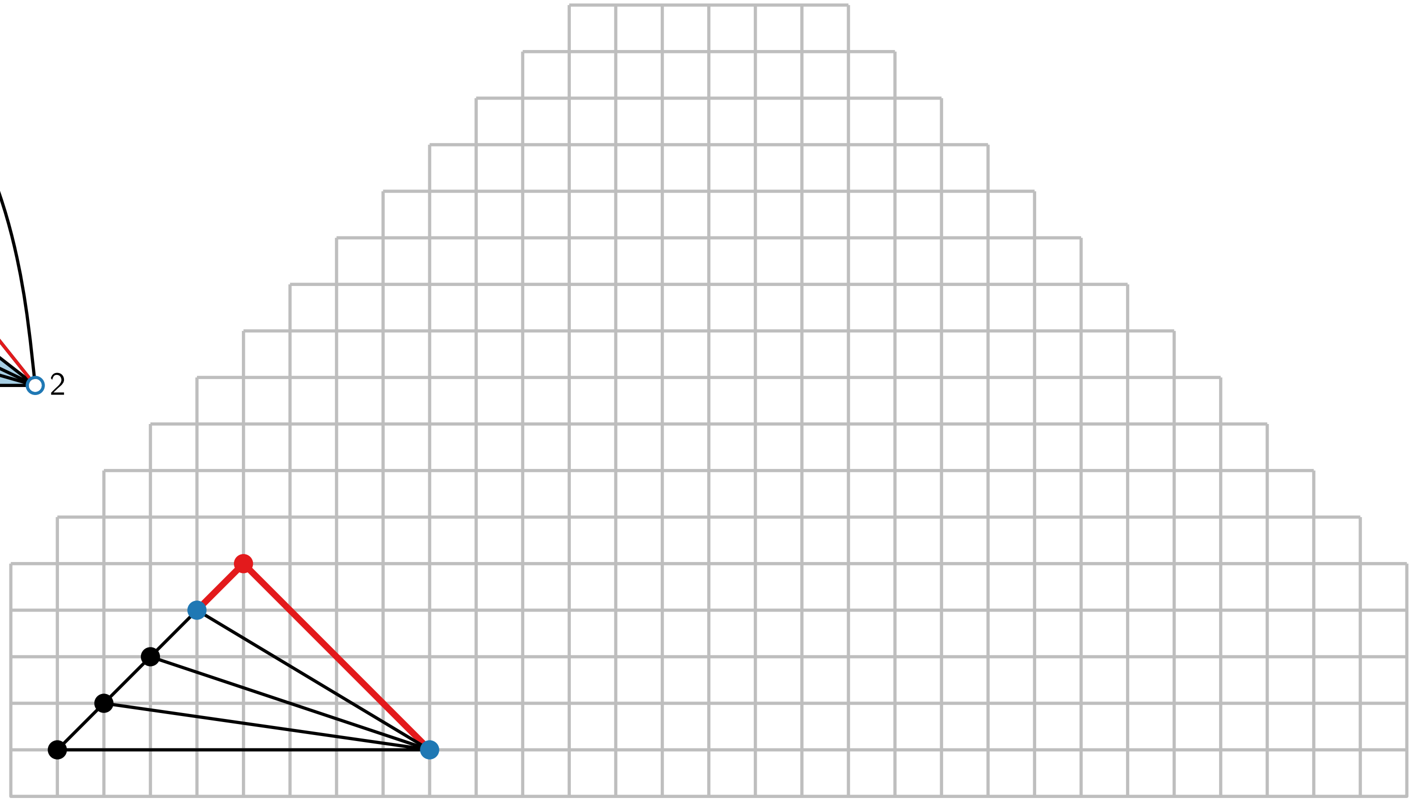




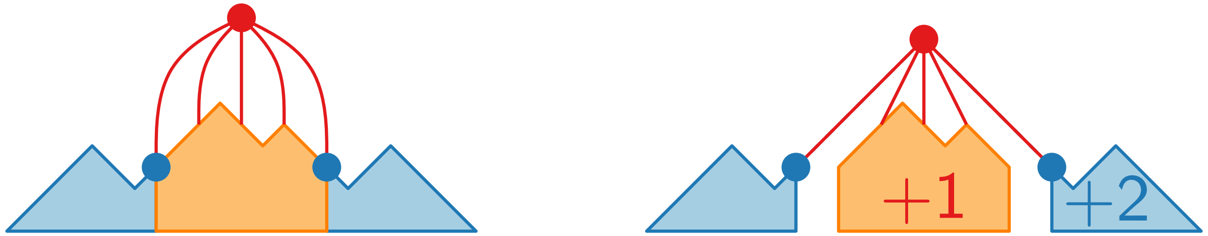
Shift method – example

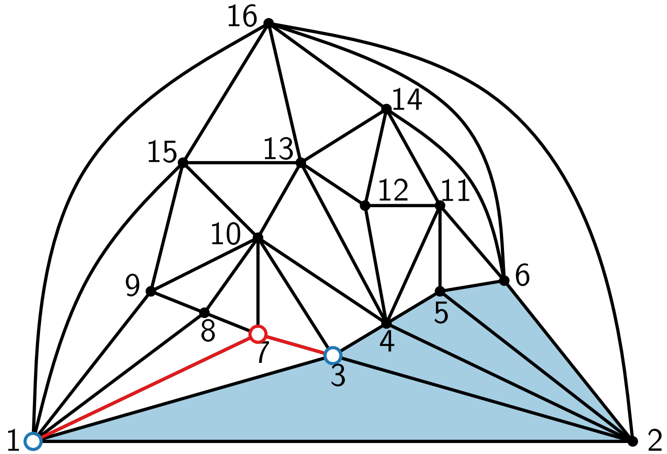


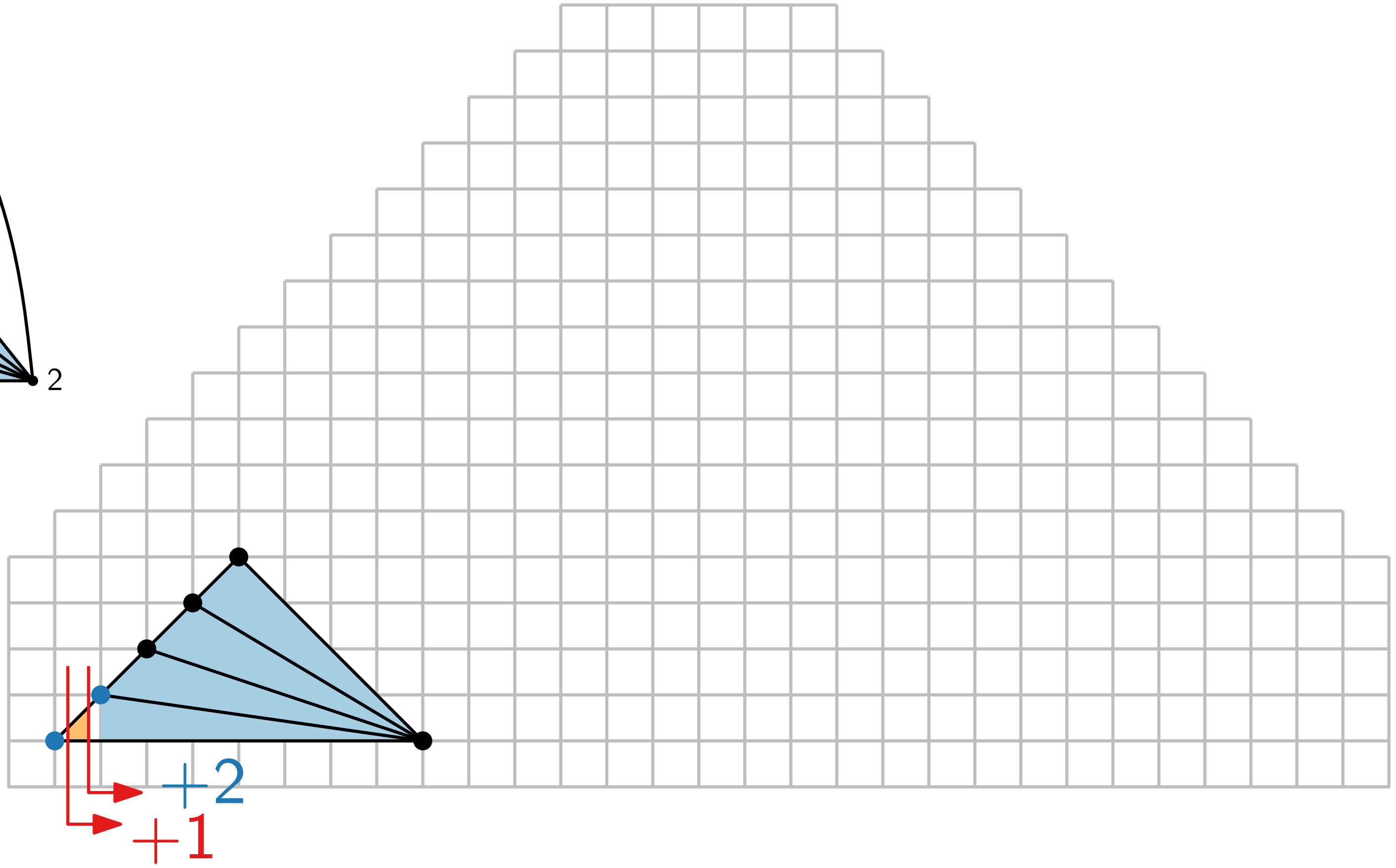




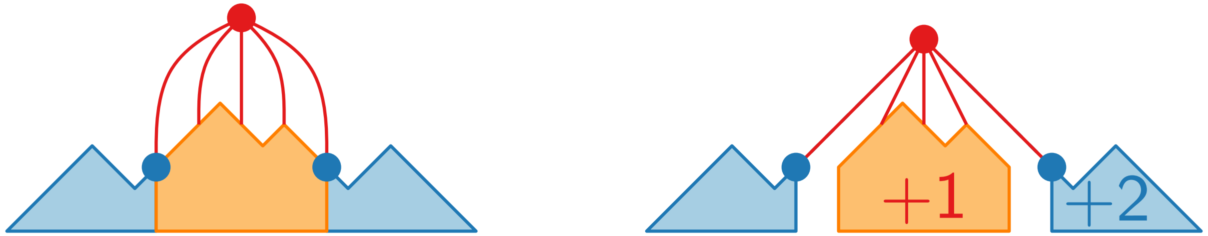
Shift method – example

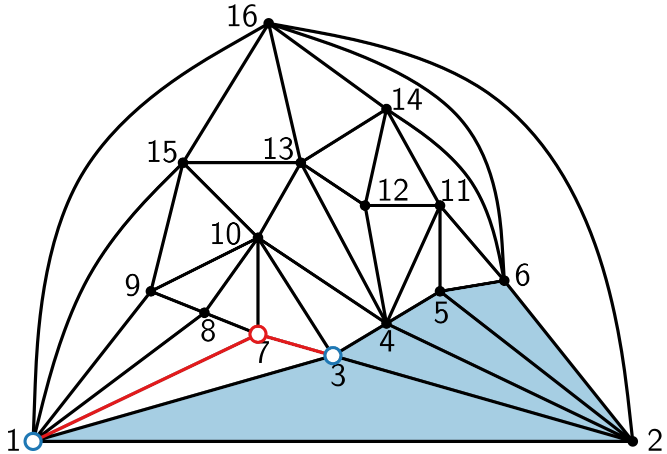


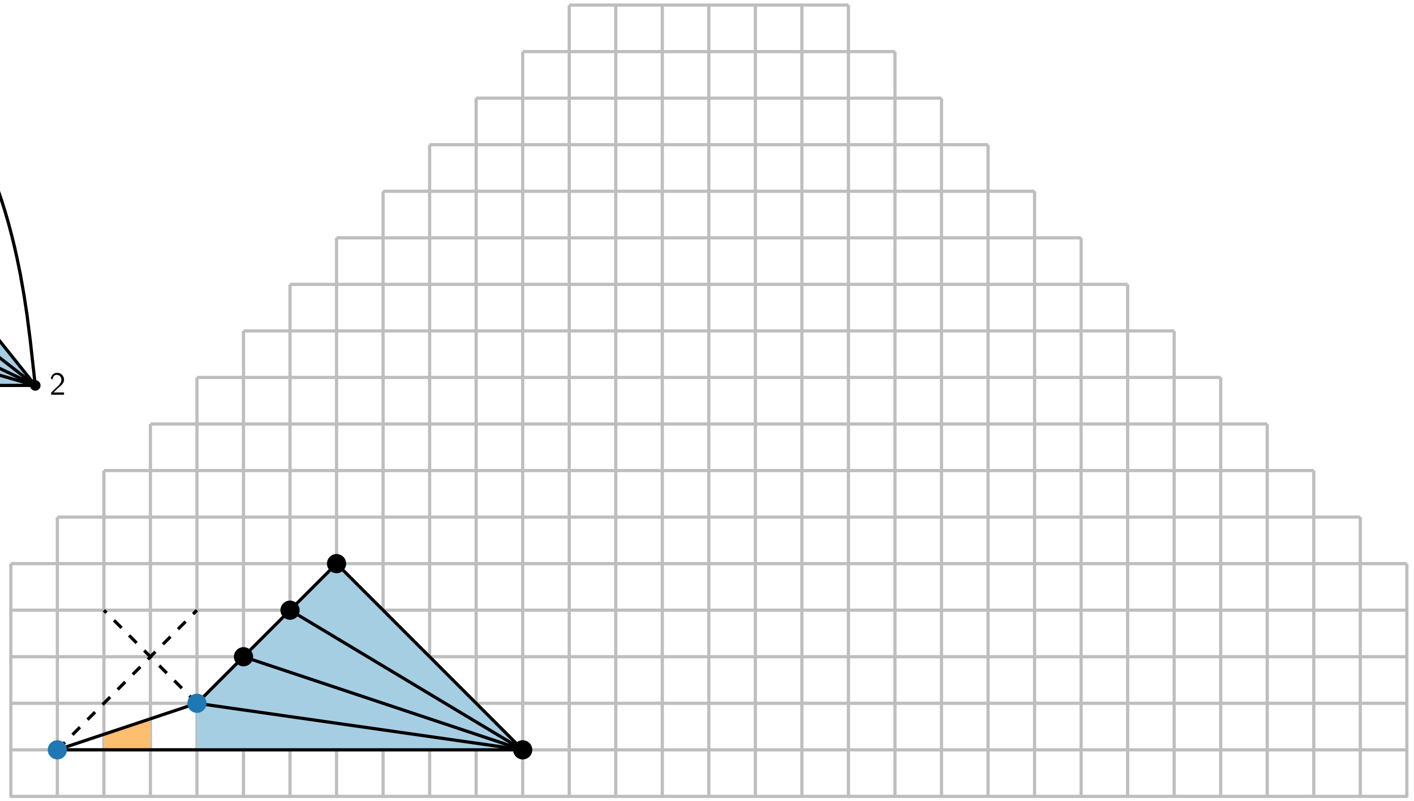




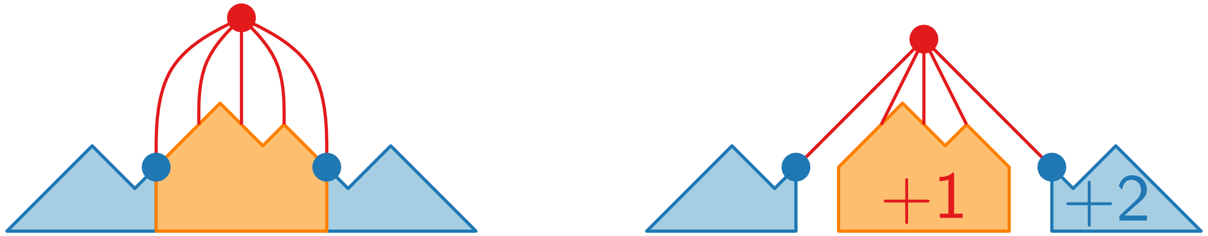
Shift method – example

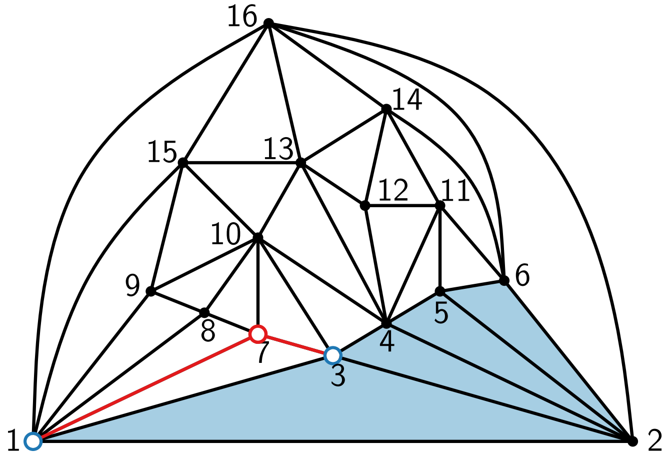


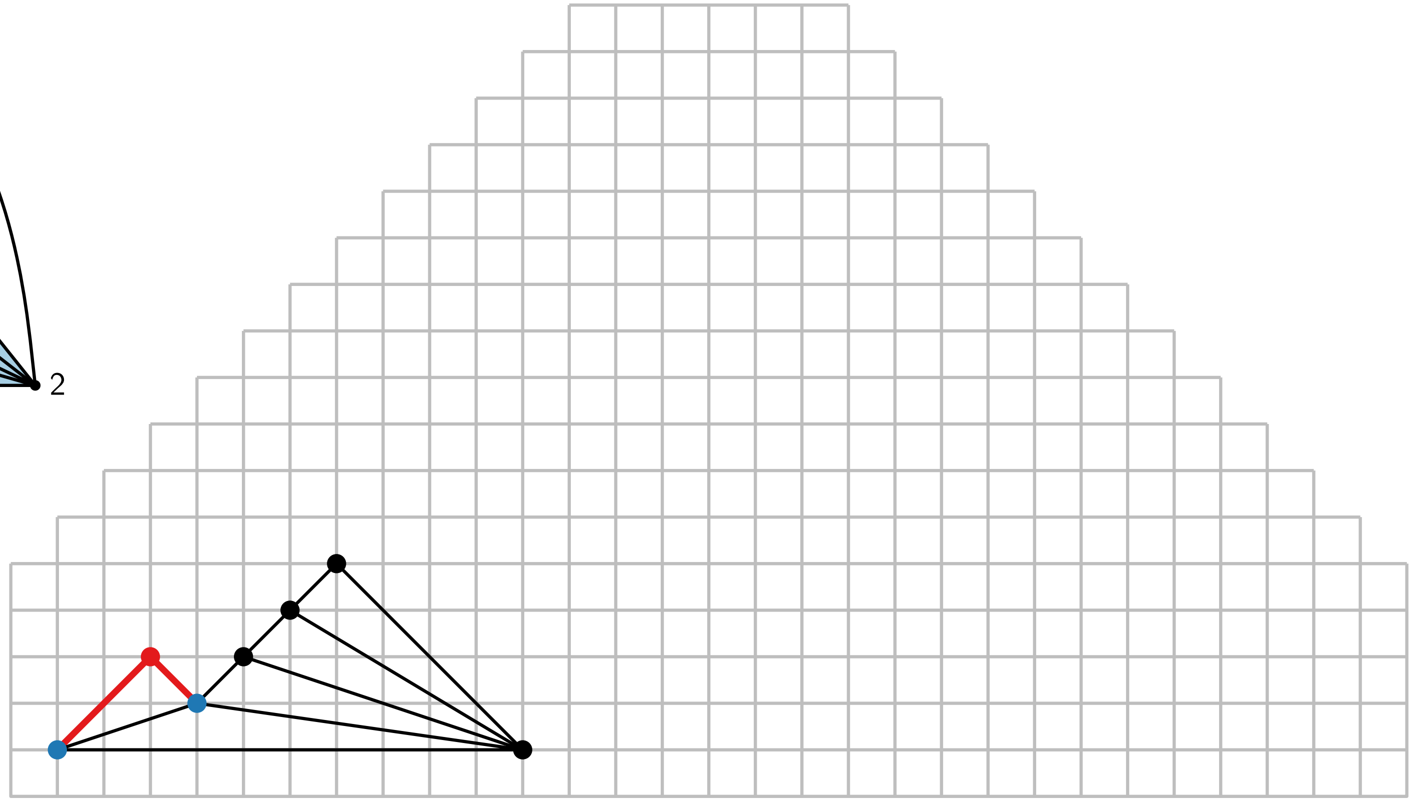




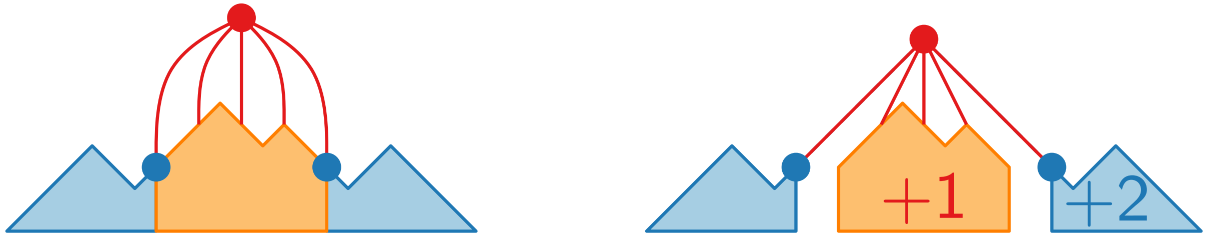
Shift method – example

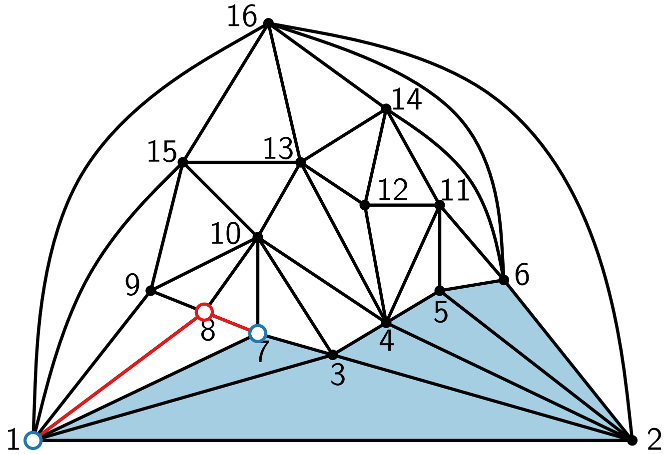


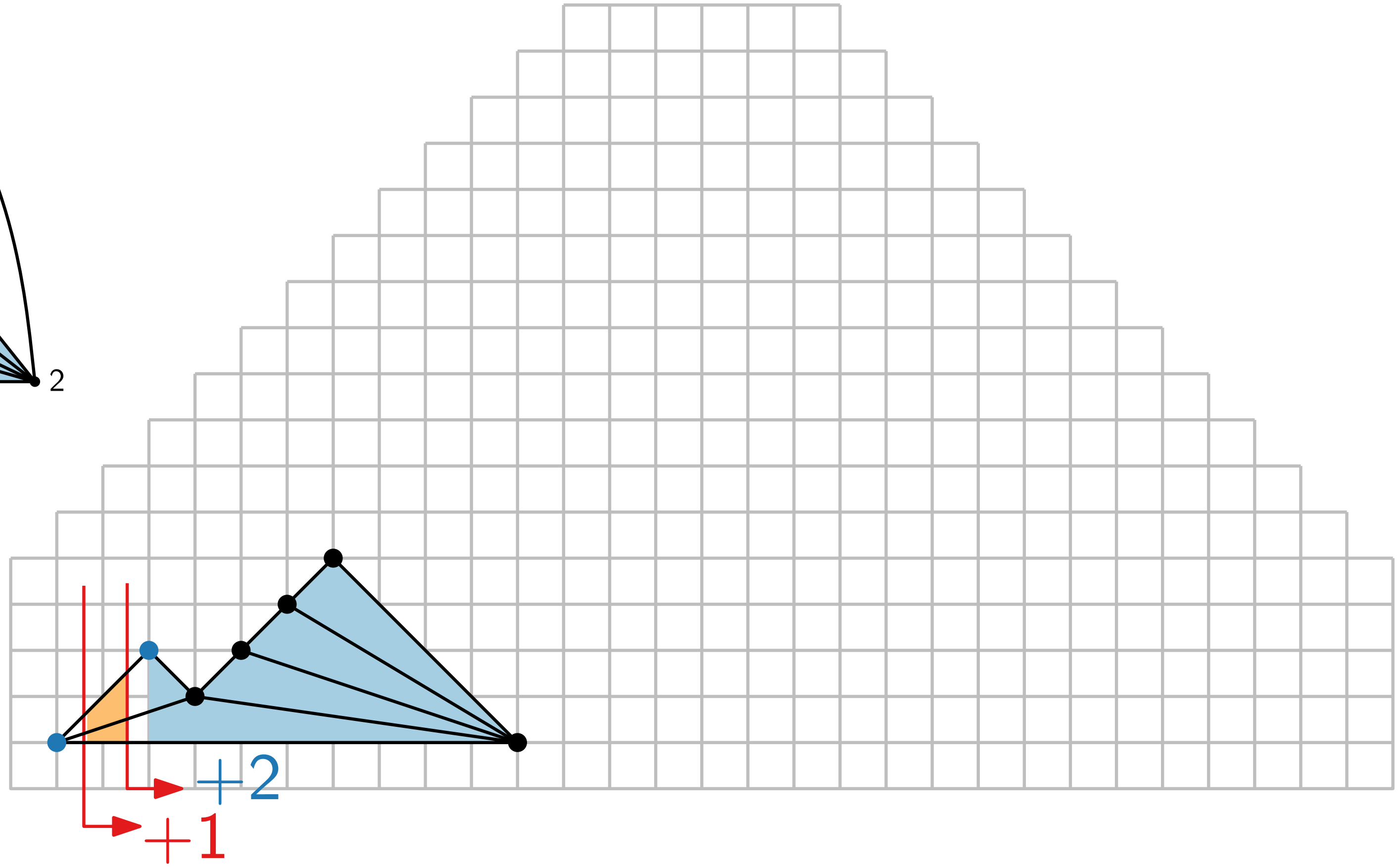




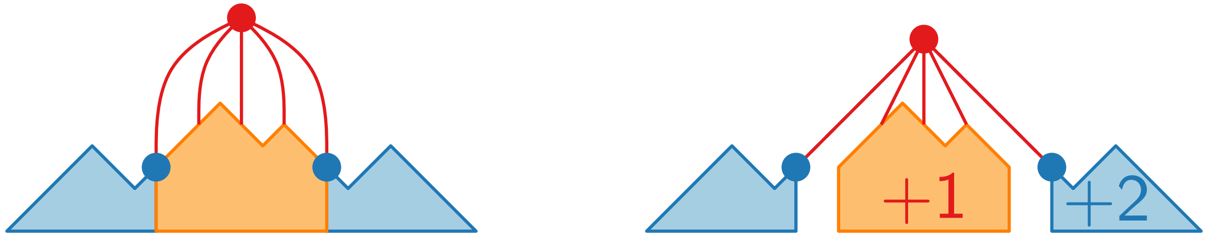
Shift method – example

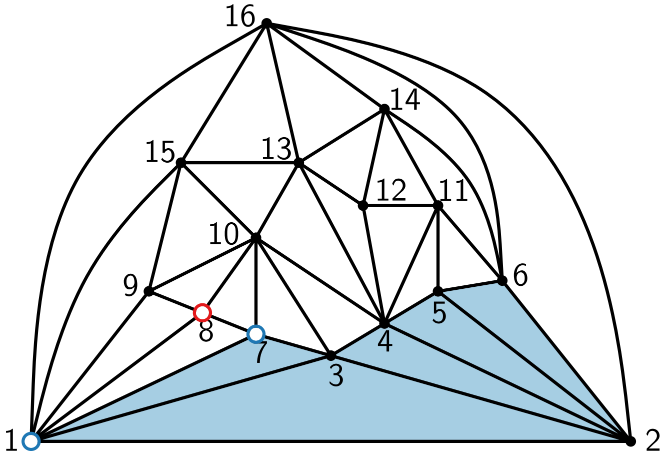


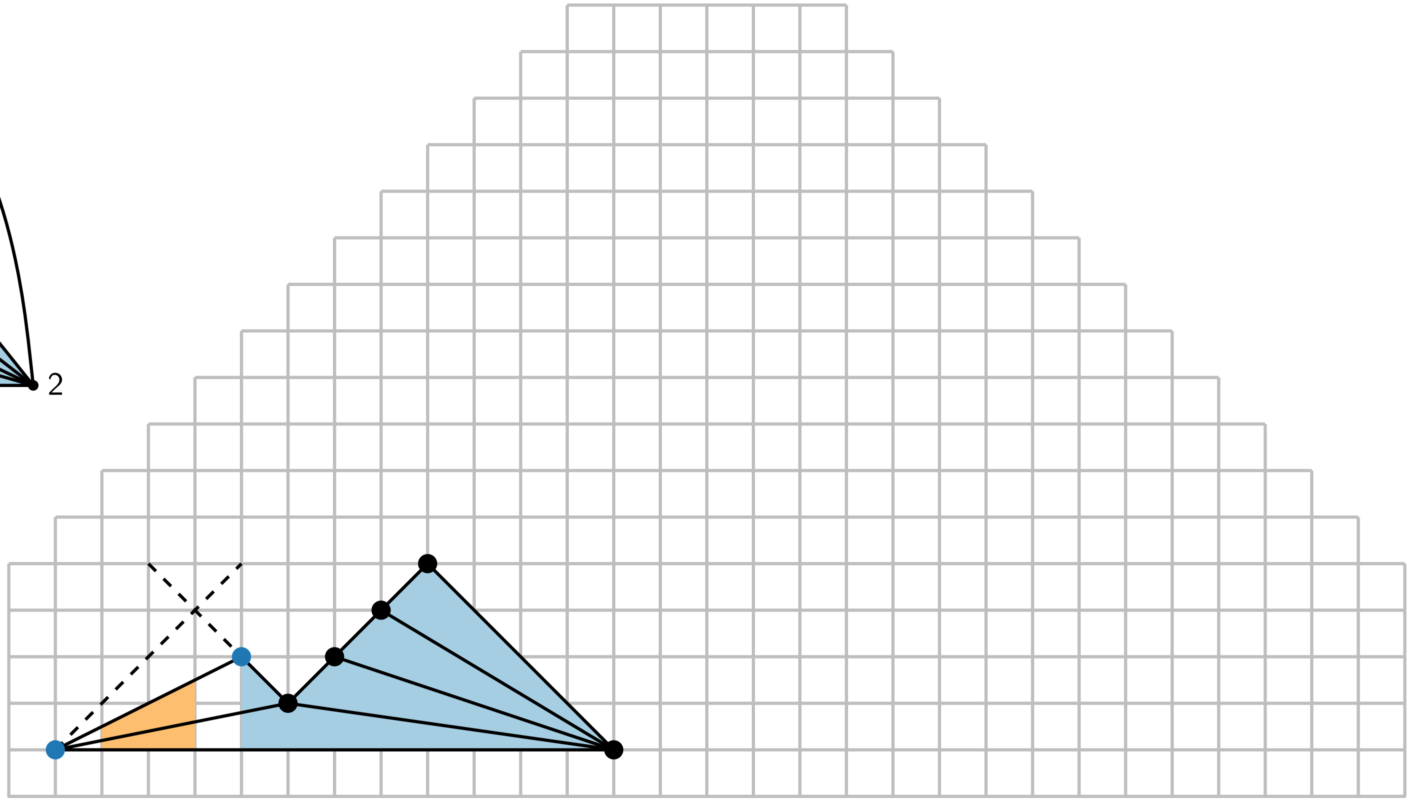




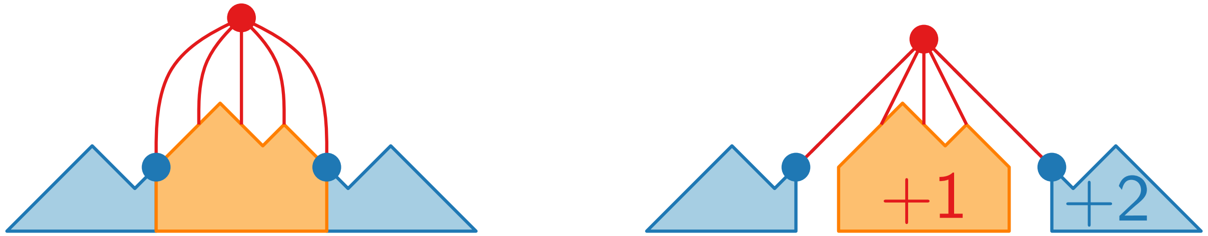
Shift method – example

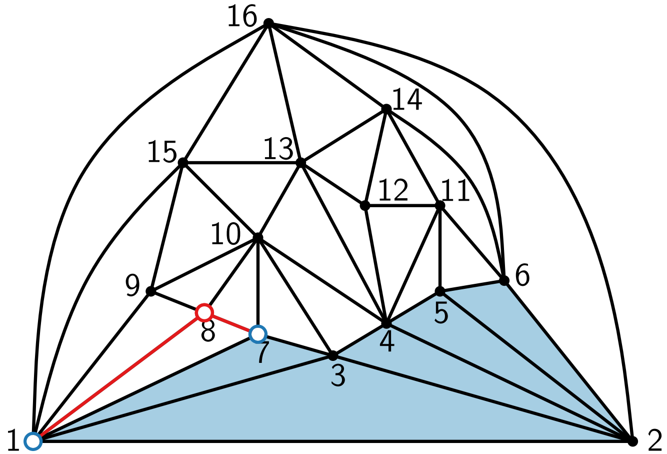


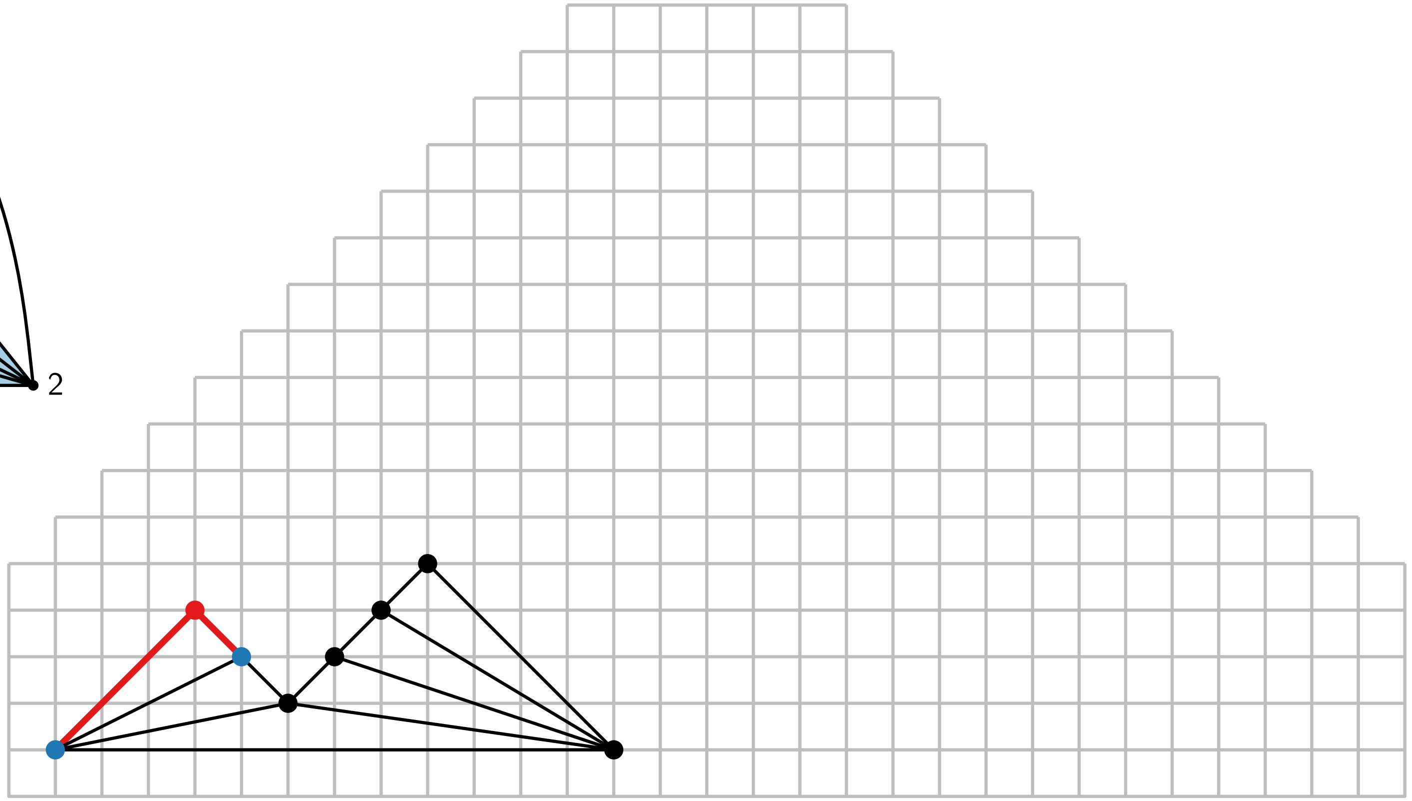




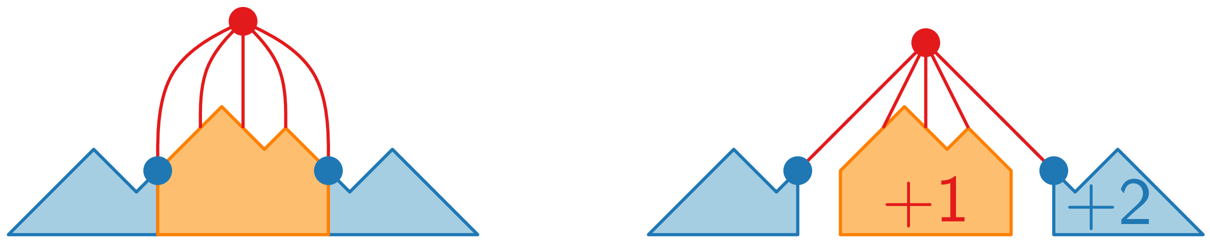
Shift method – example

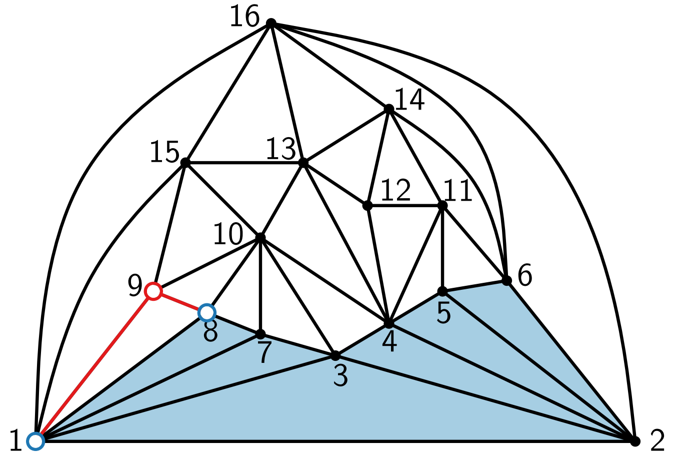


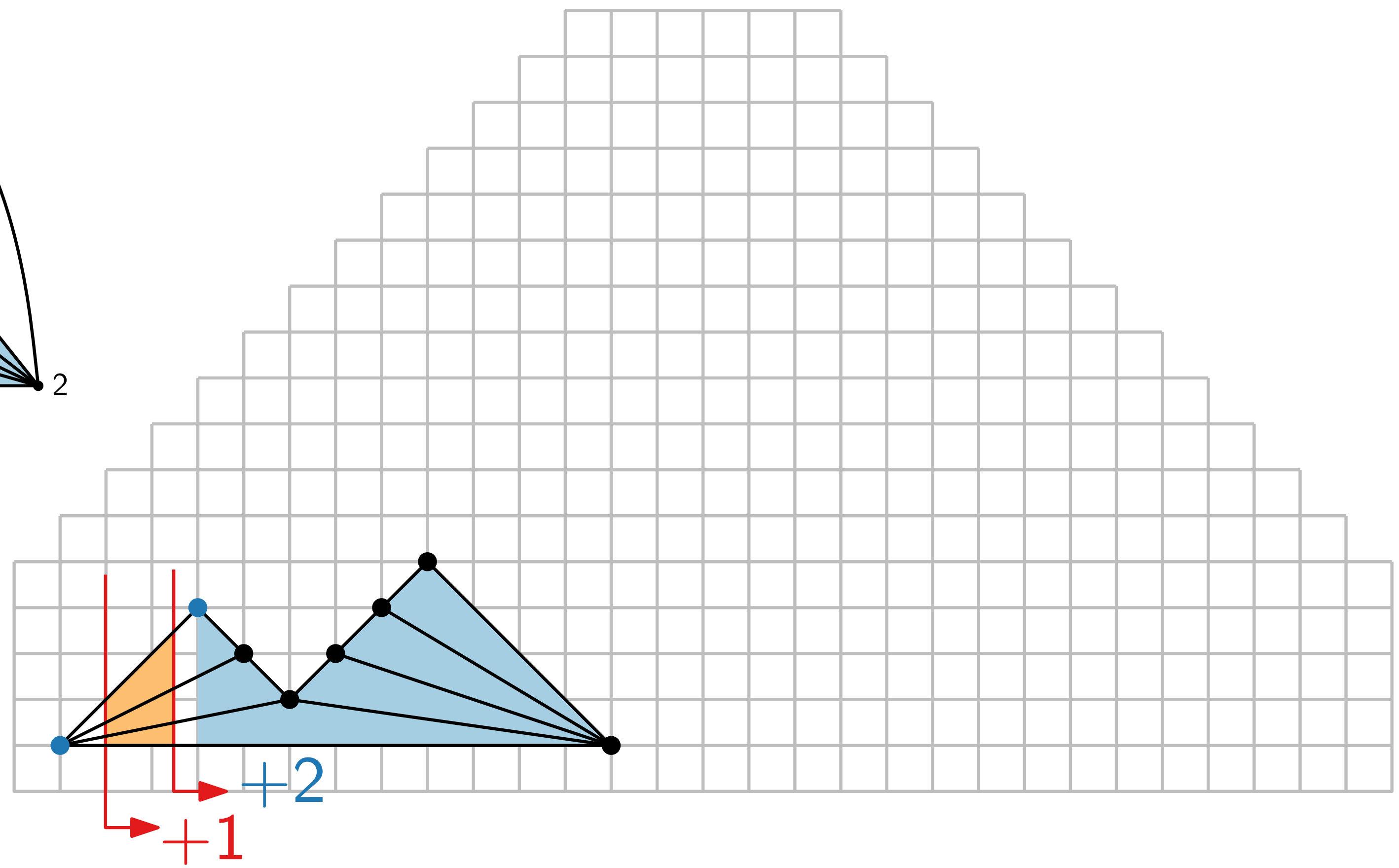




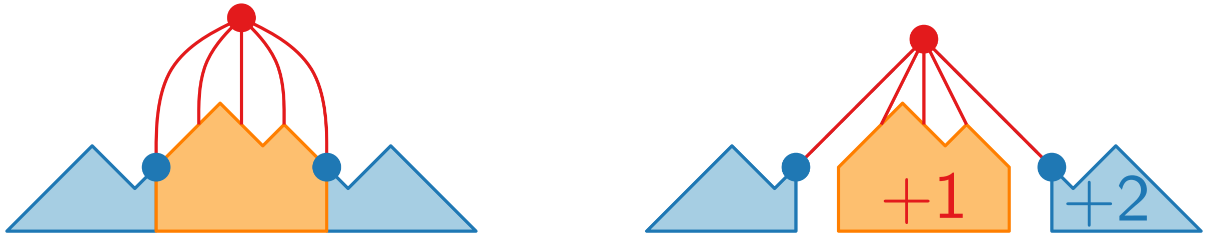
Shift method – example

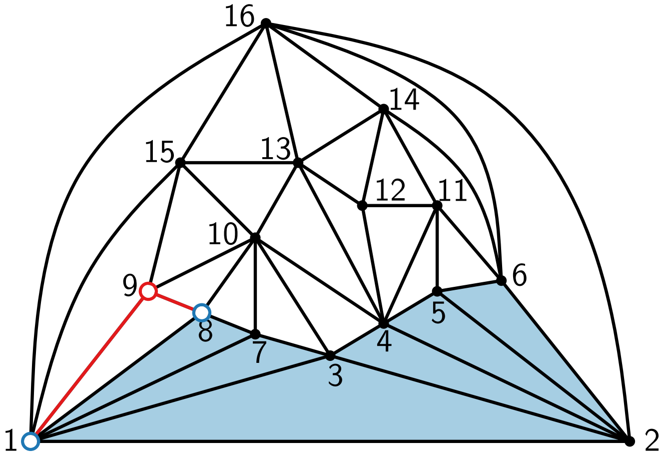


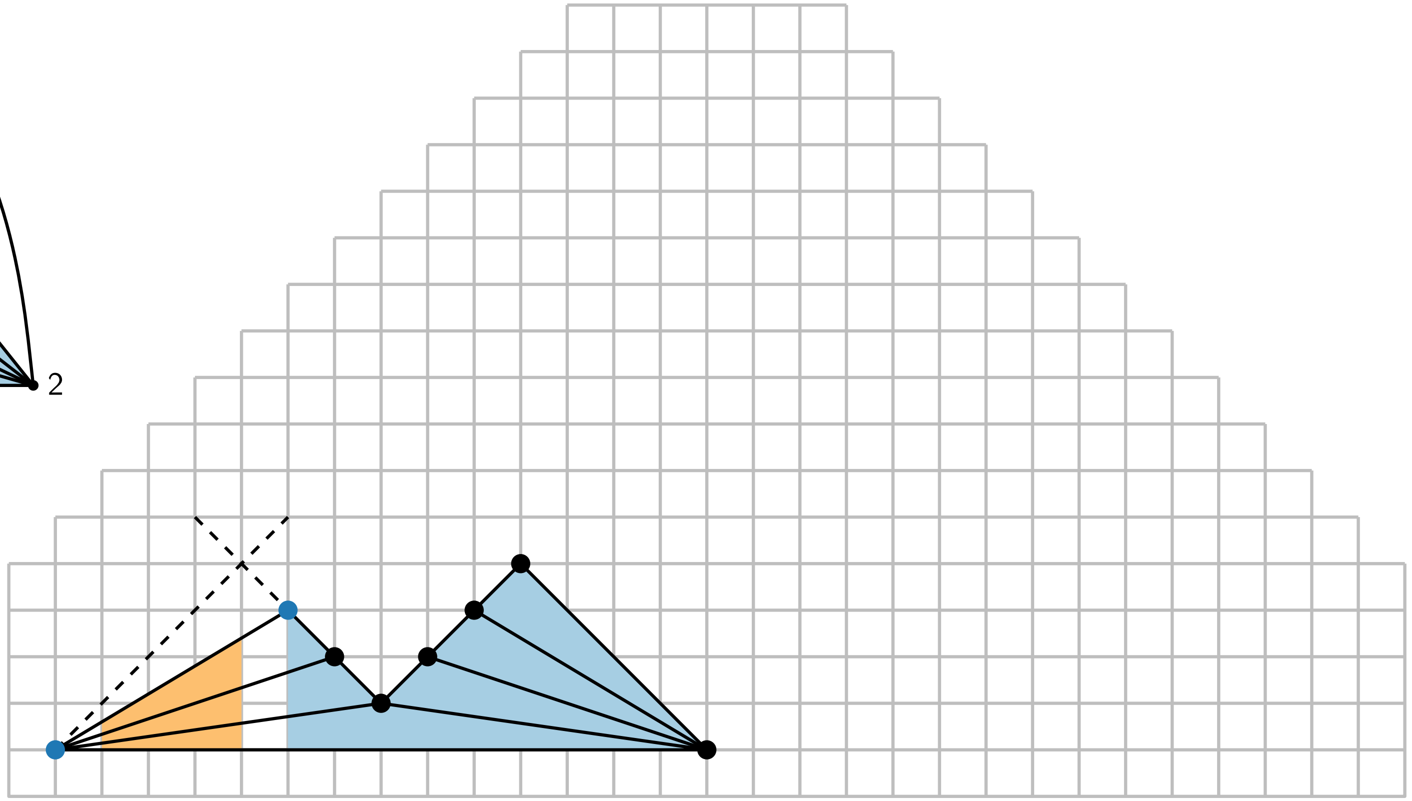




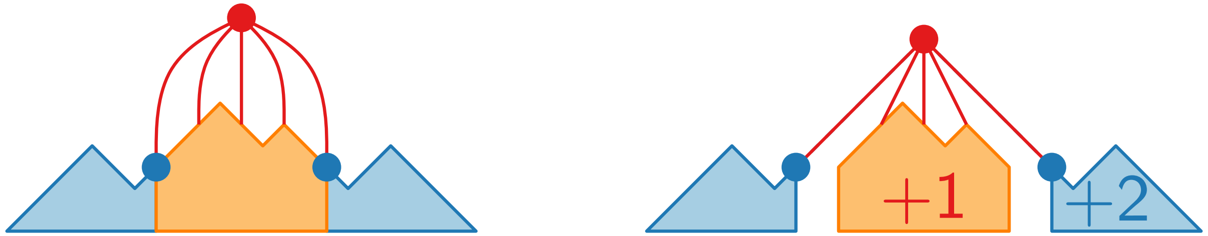
Shift method – example

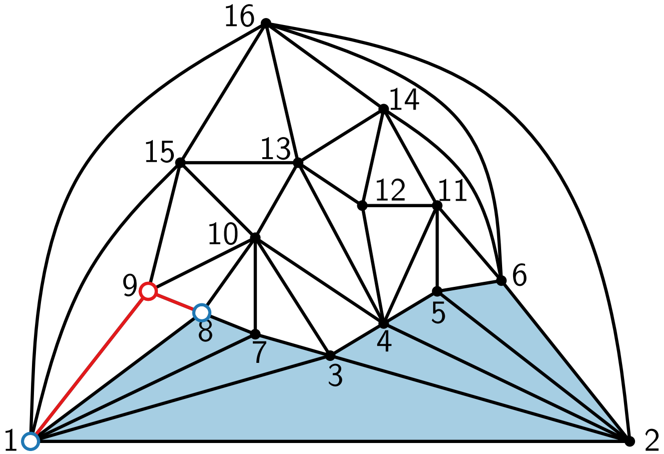


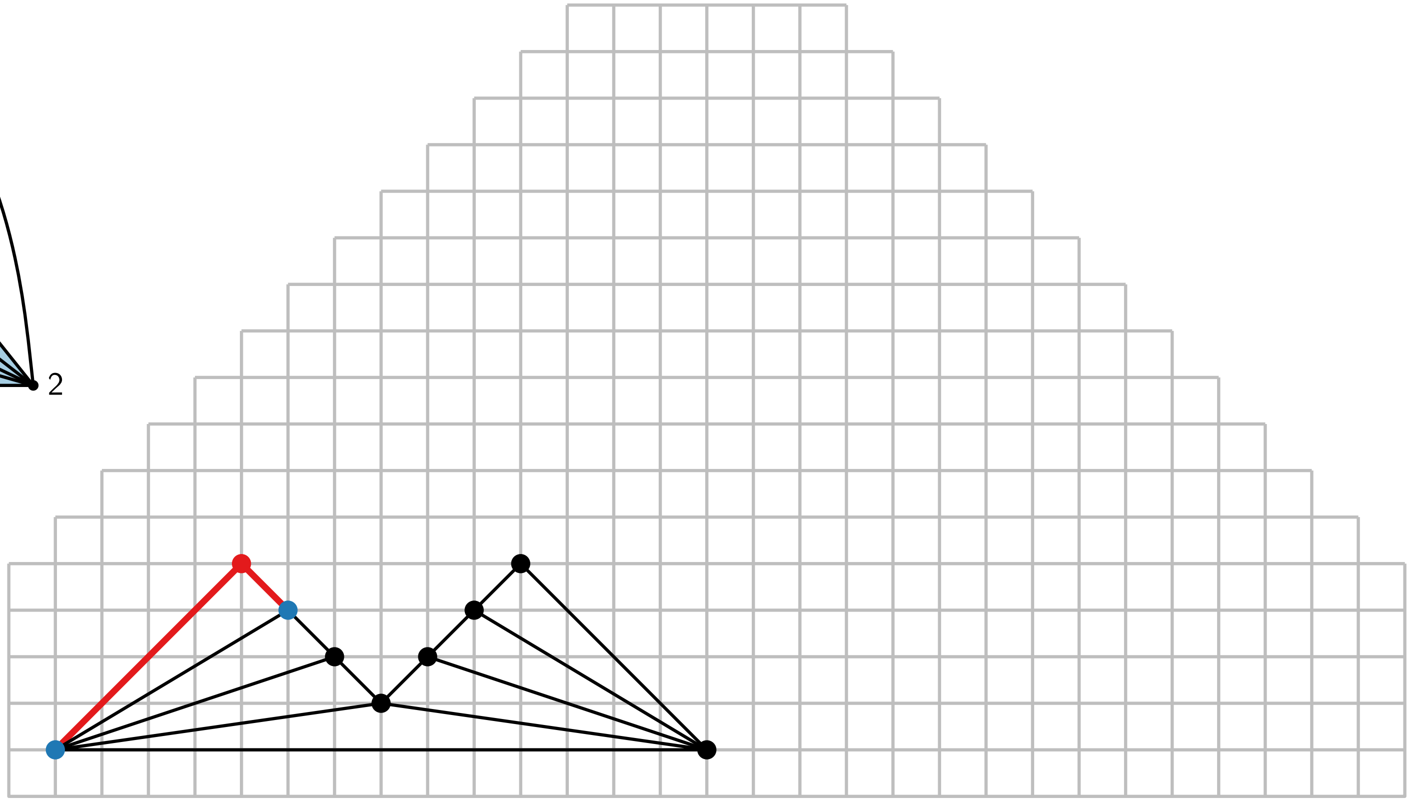




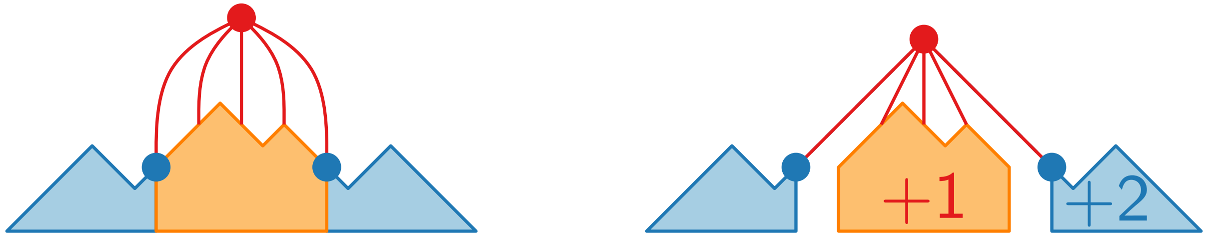
Shift method – example

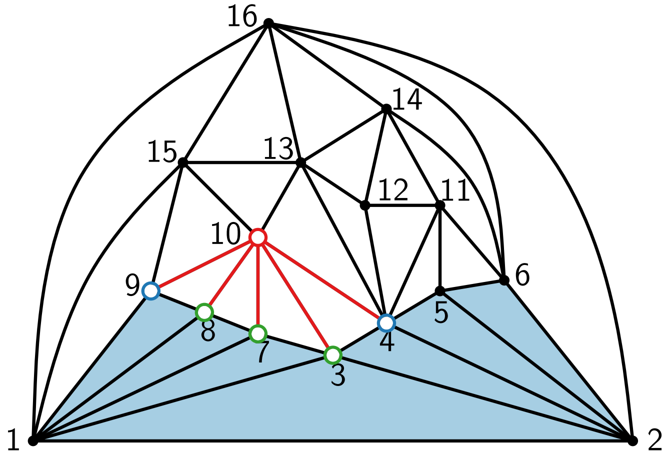


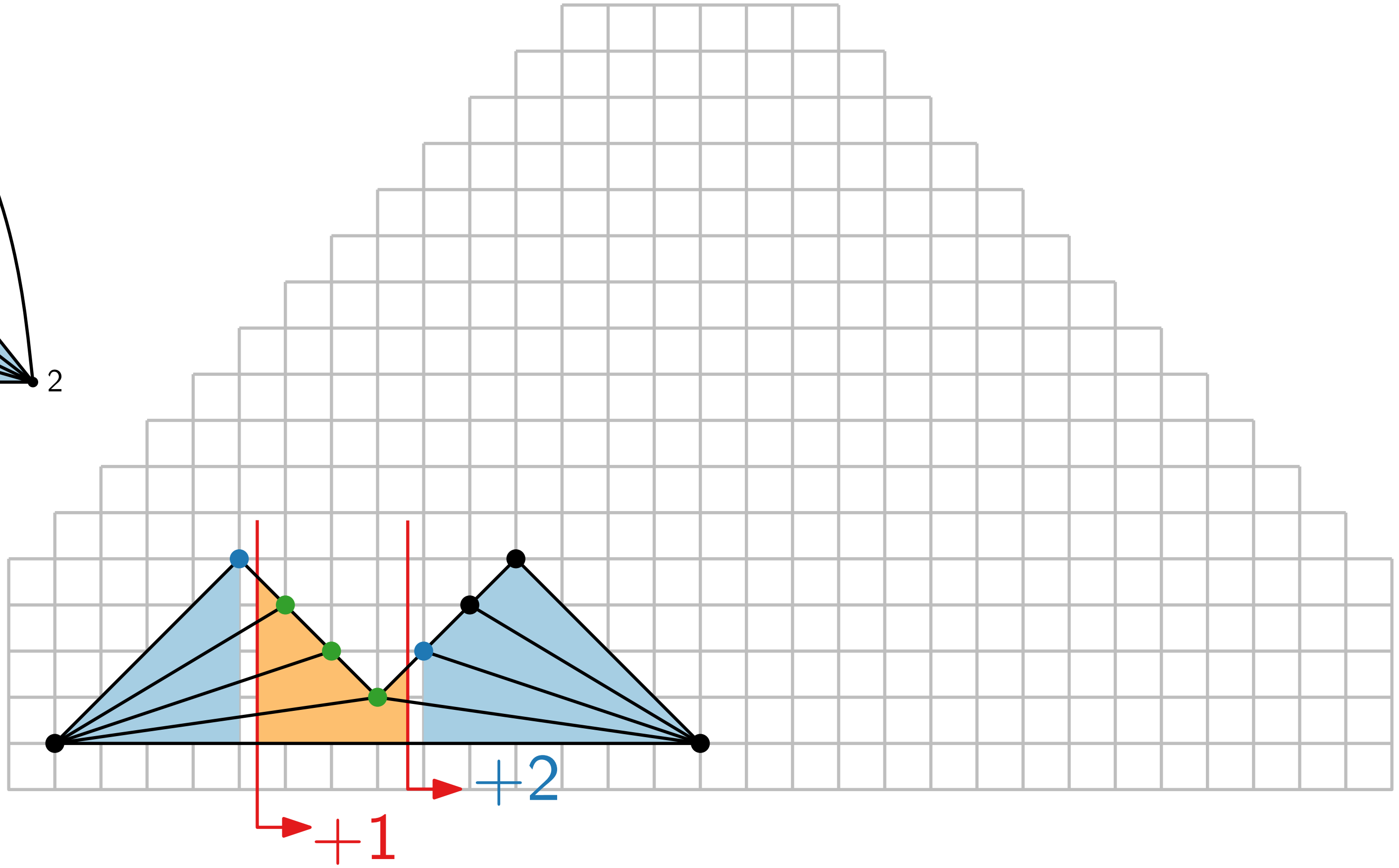




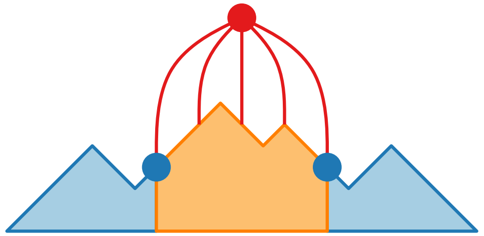
Shift method – example

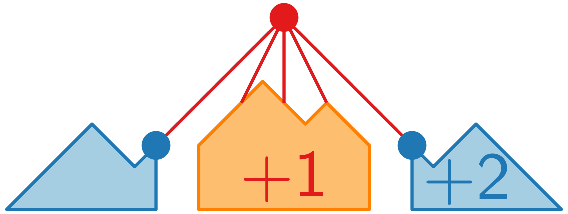


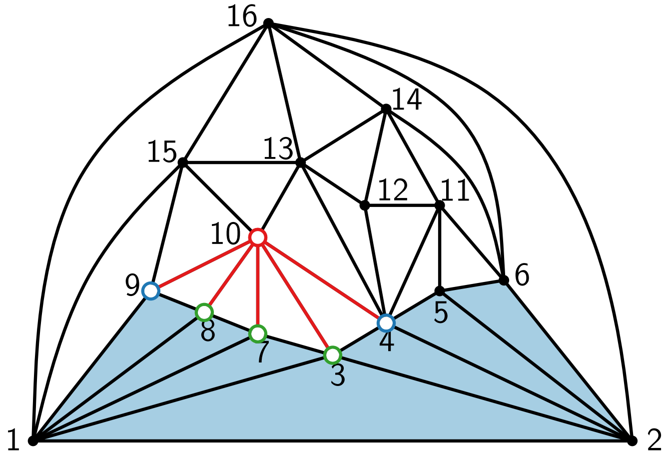


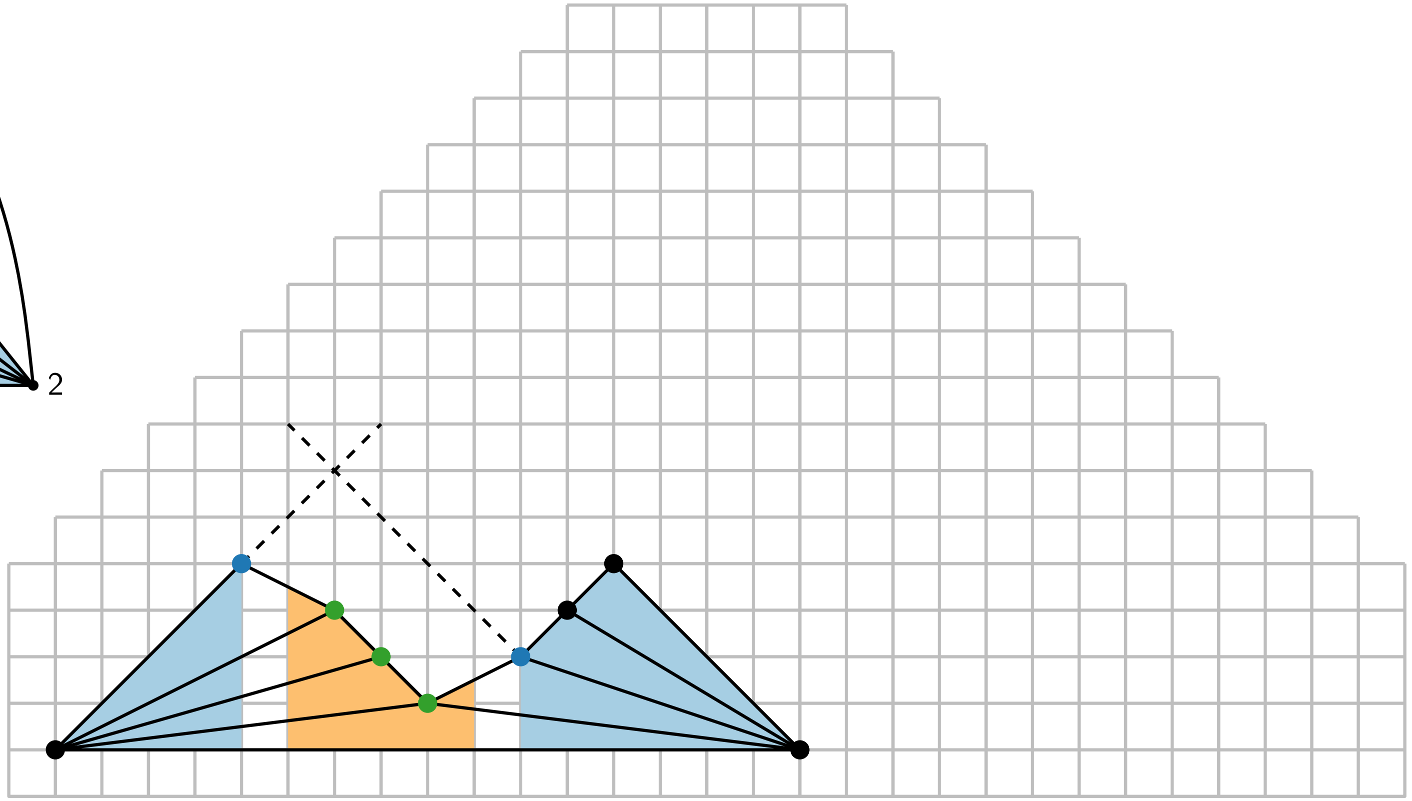


Shift method – example

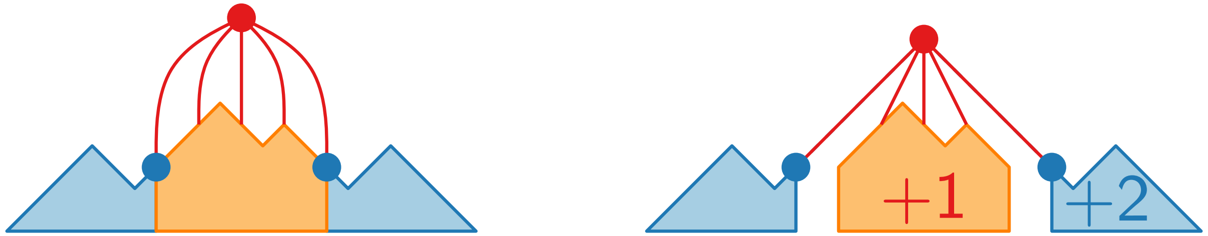


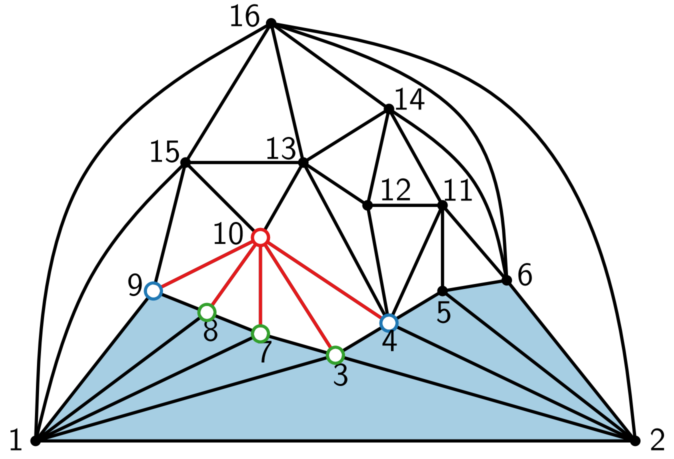


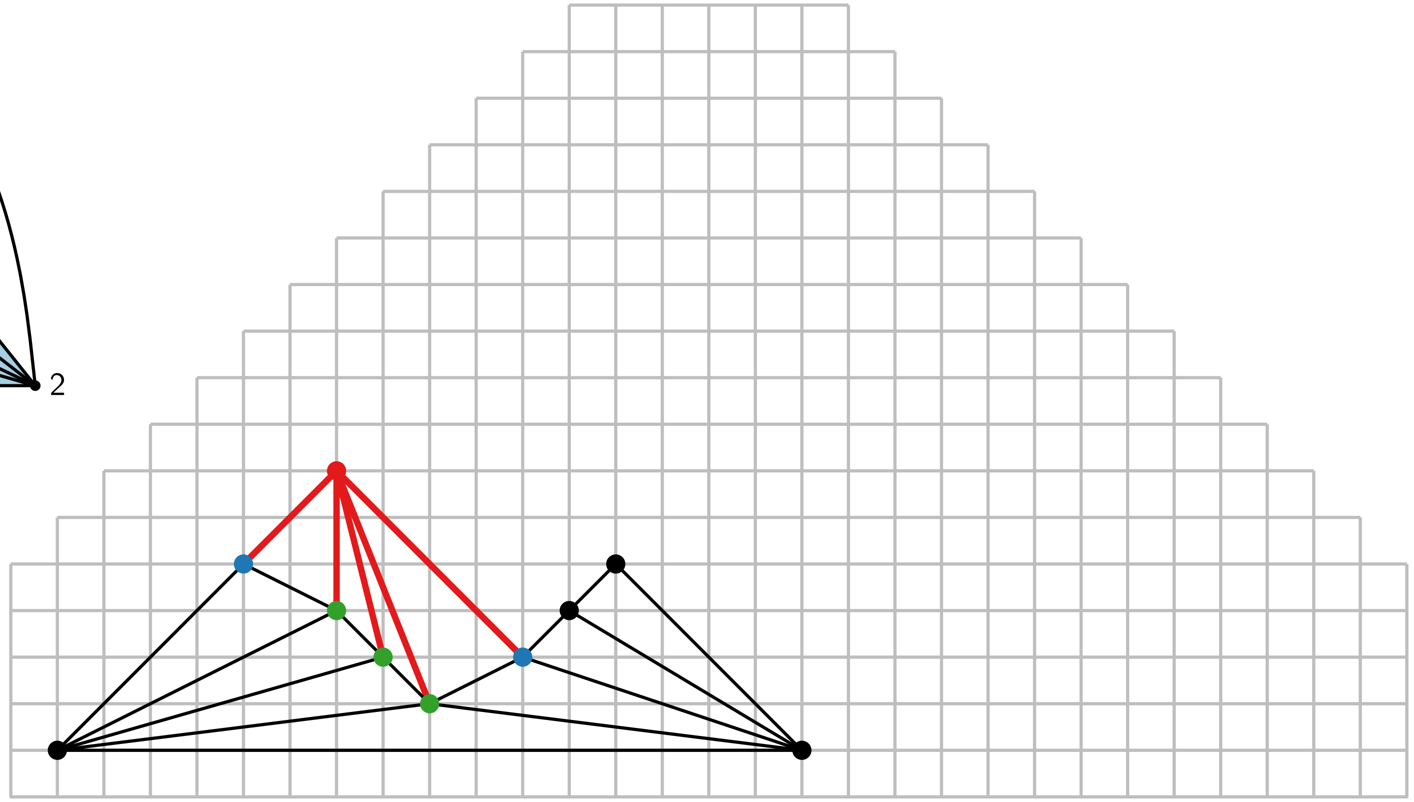




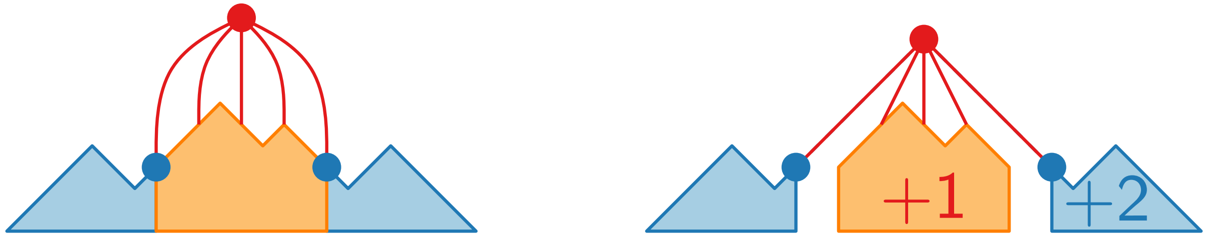
Shift method – example

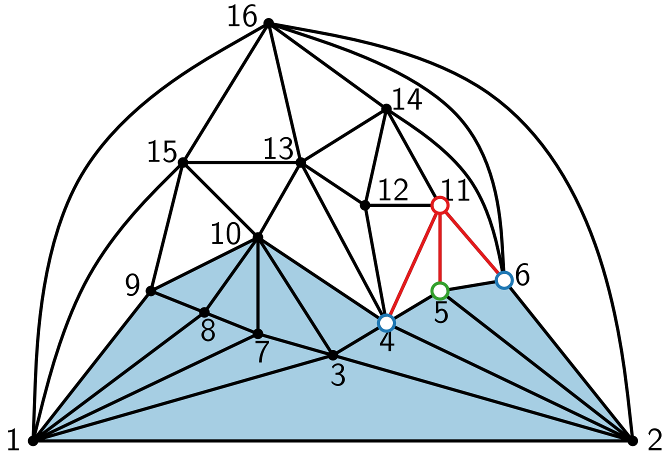


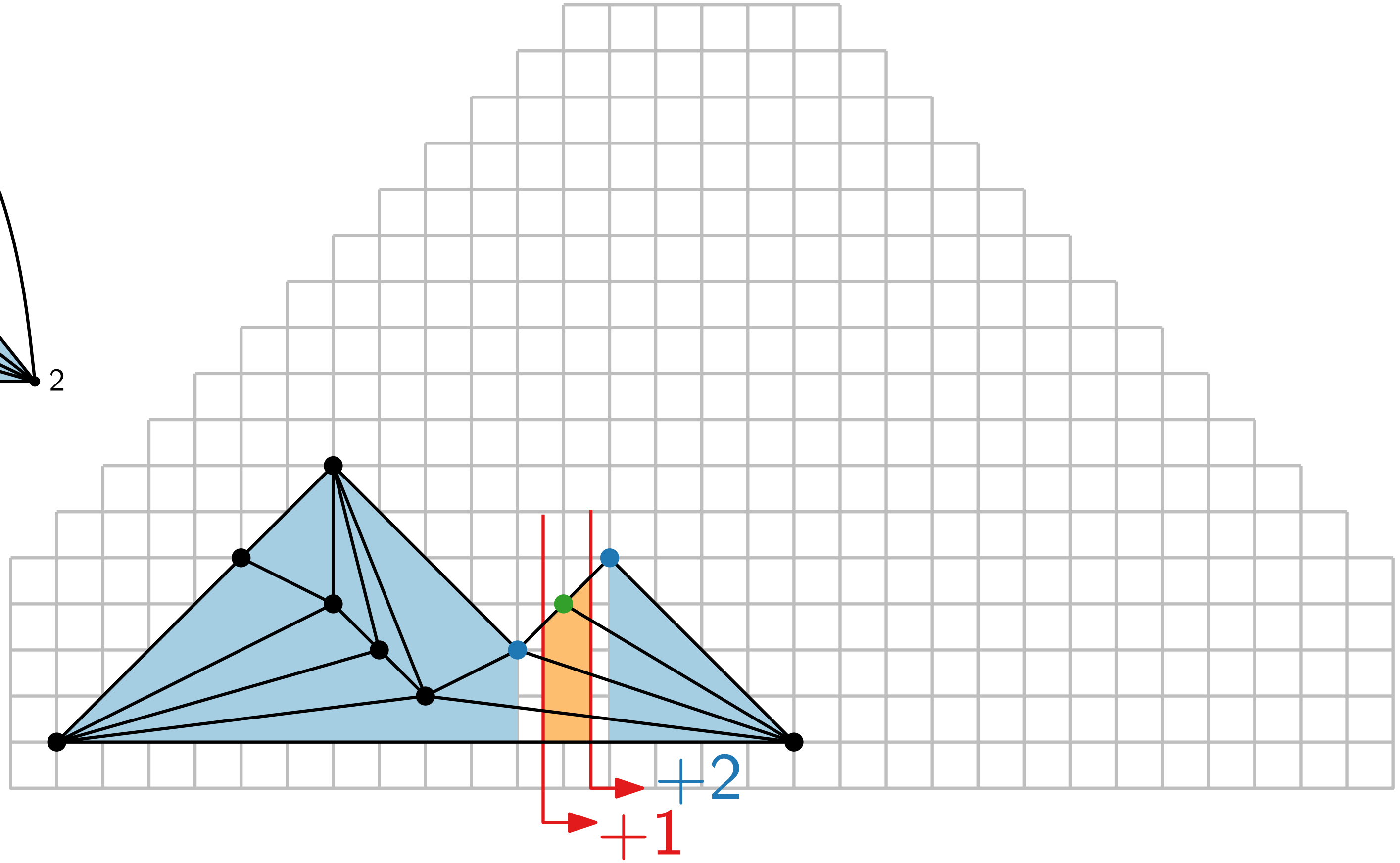




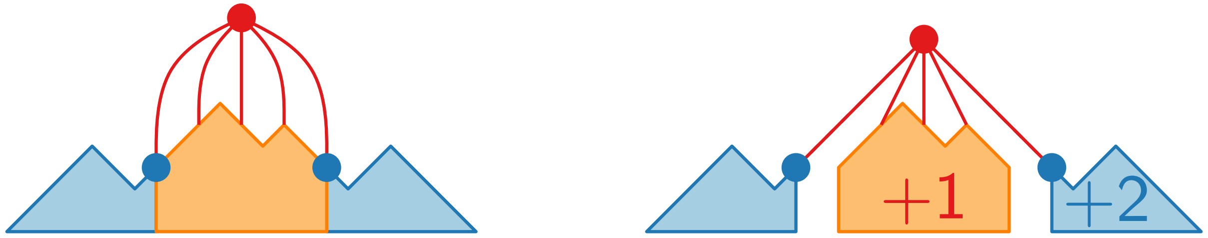
Shift method – example

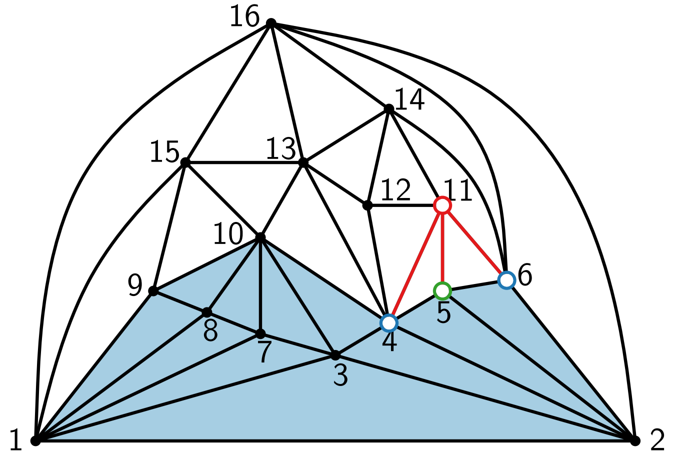


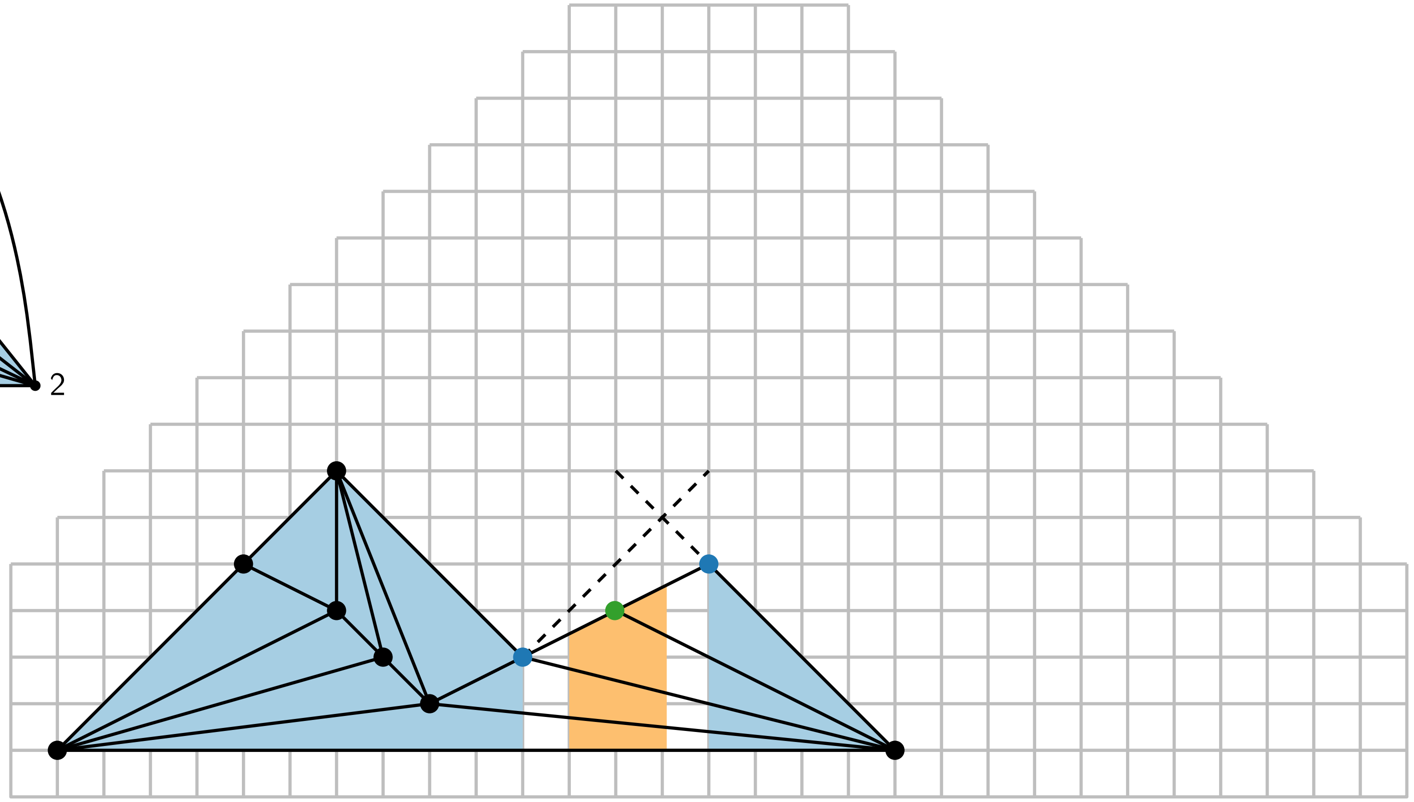




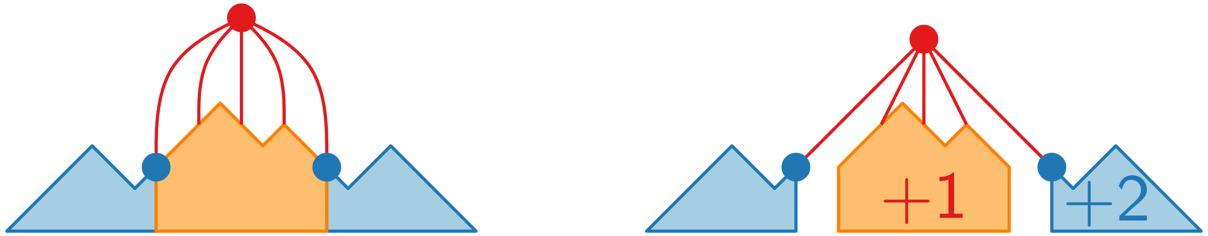
Shift method – example

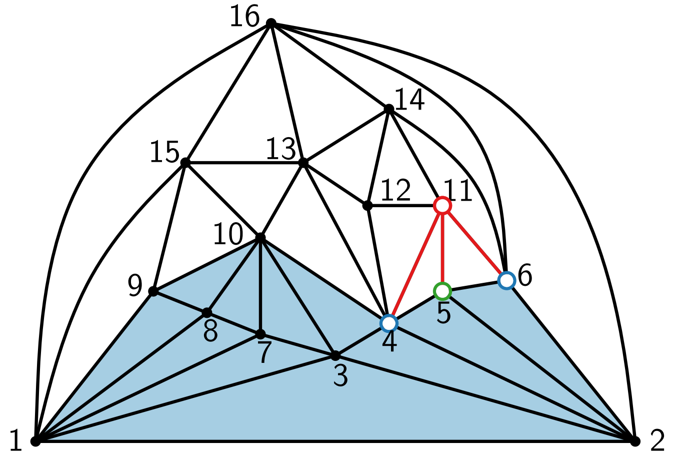


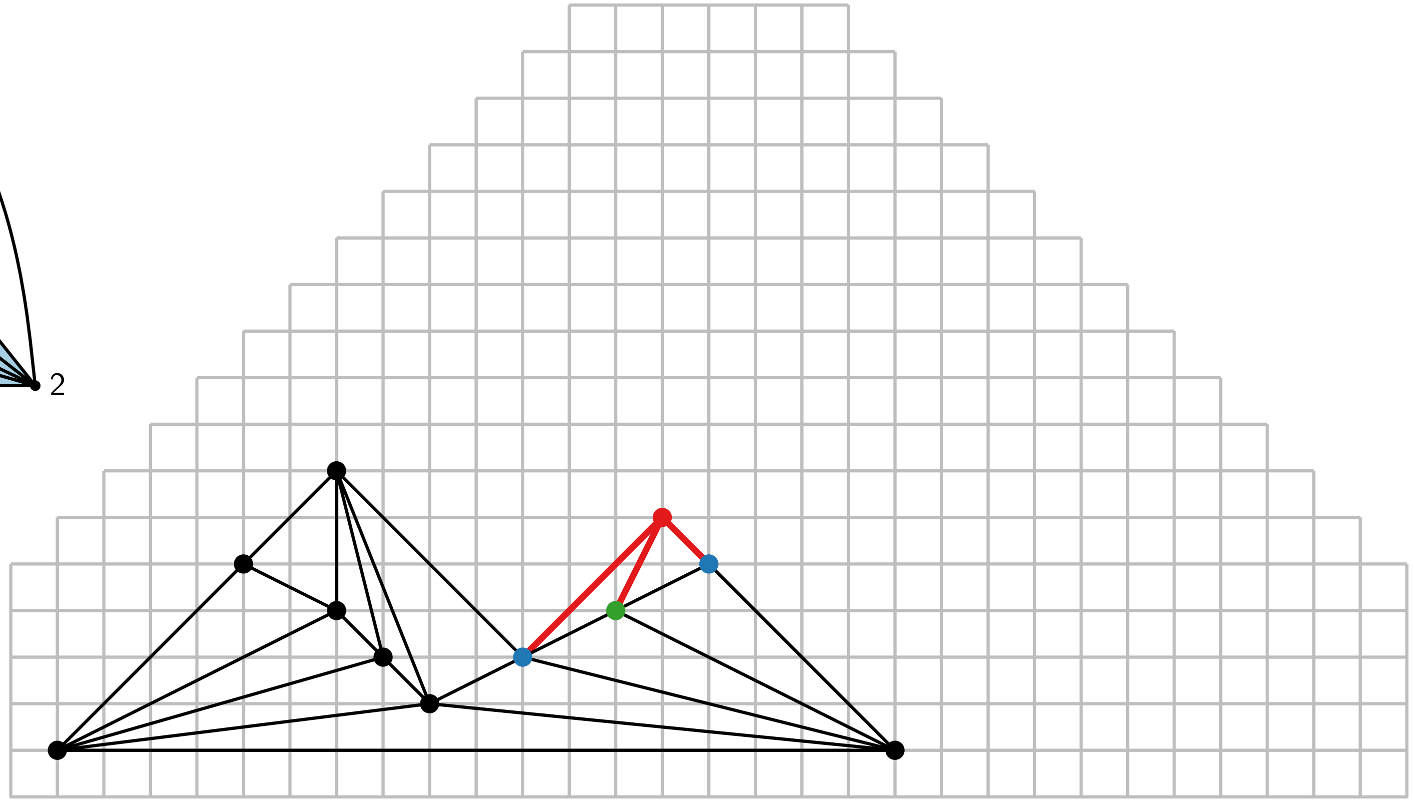




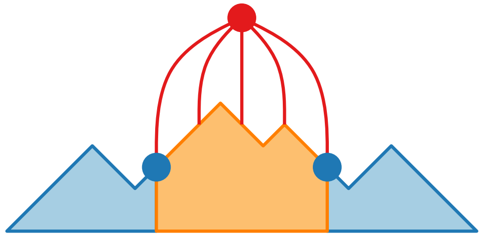
Shift method – example

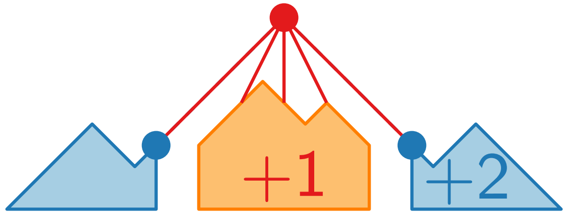


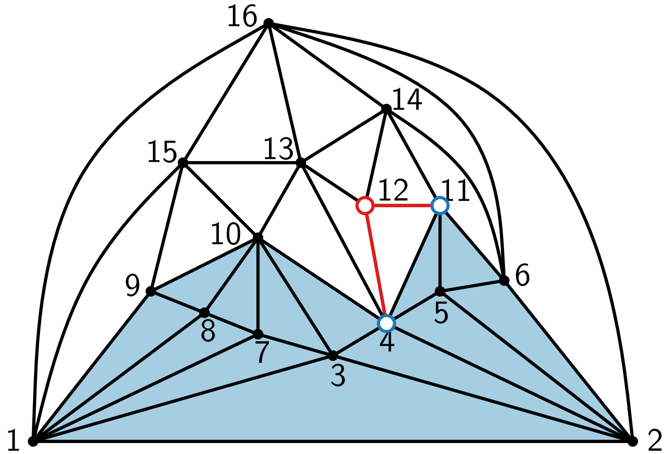


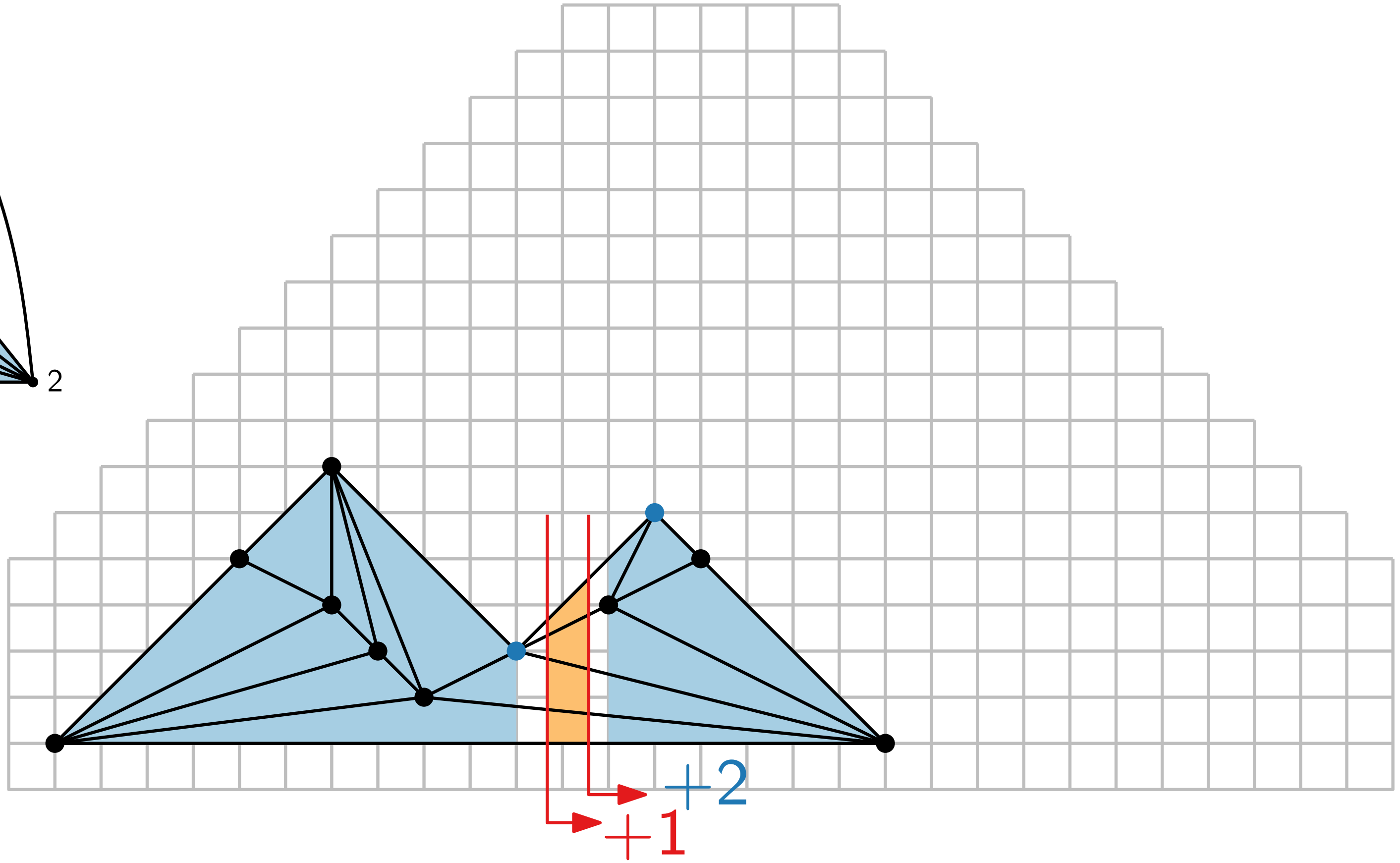


Shift method – example

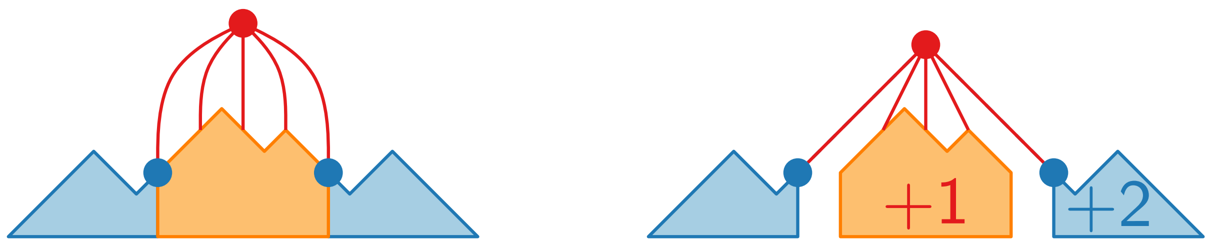


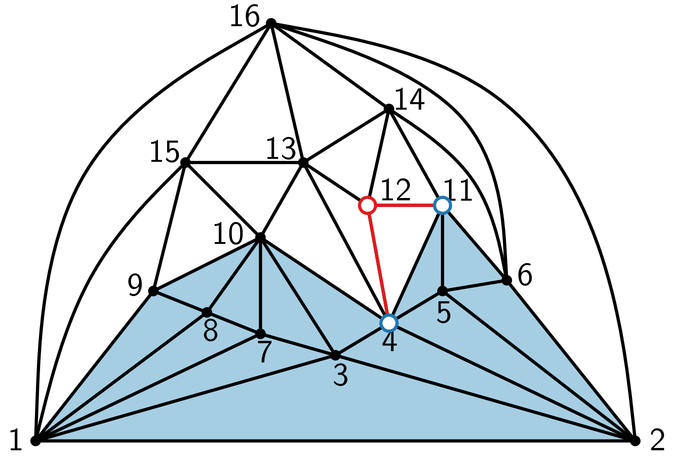


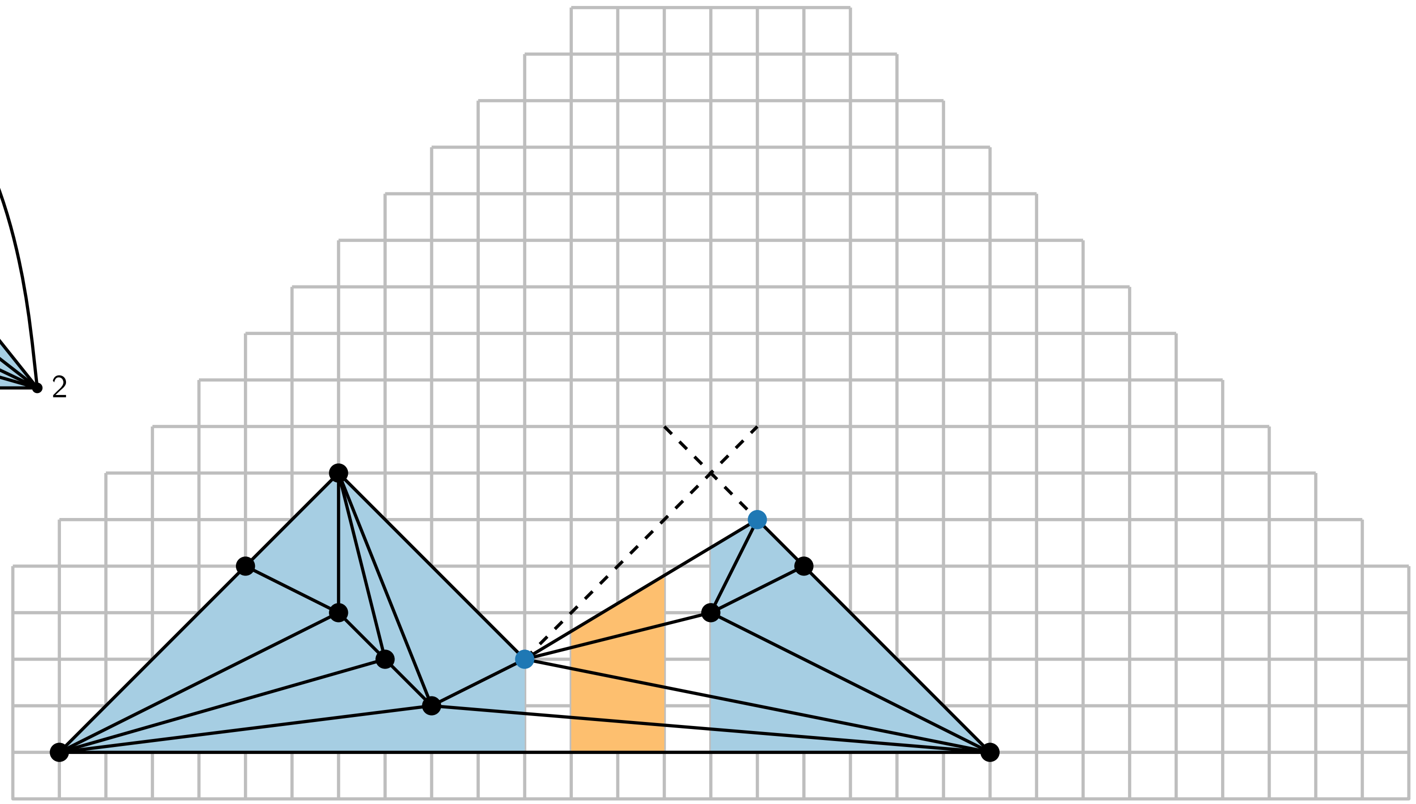




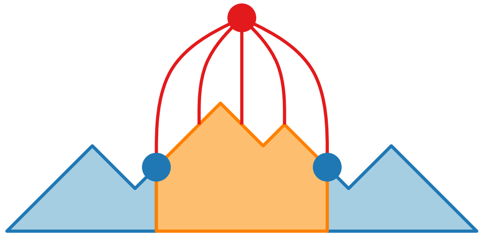
Shift method – example

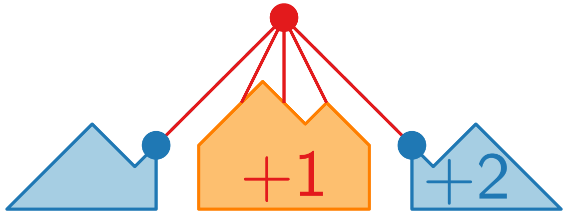


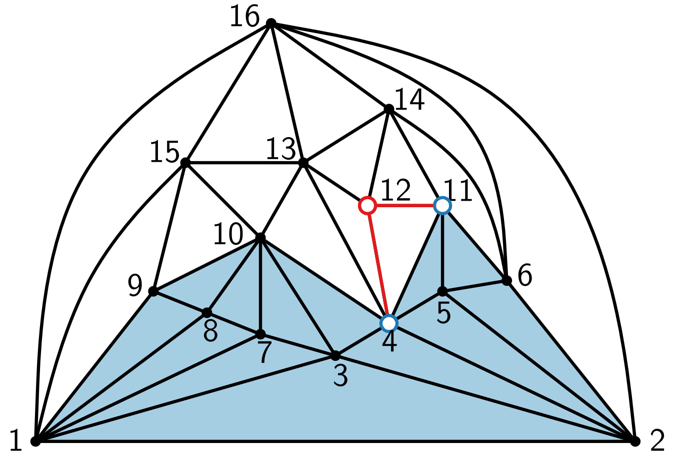


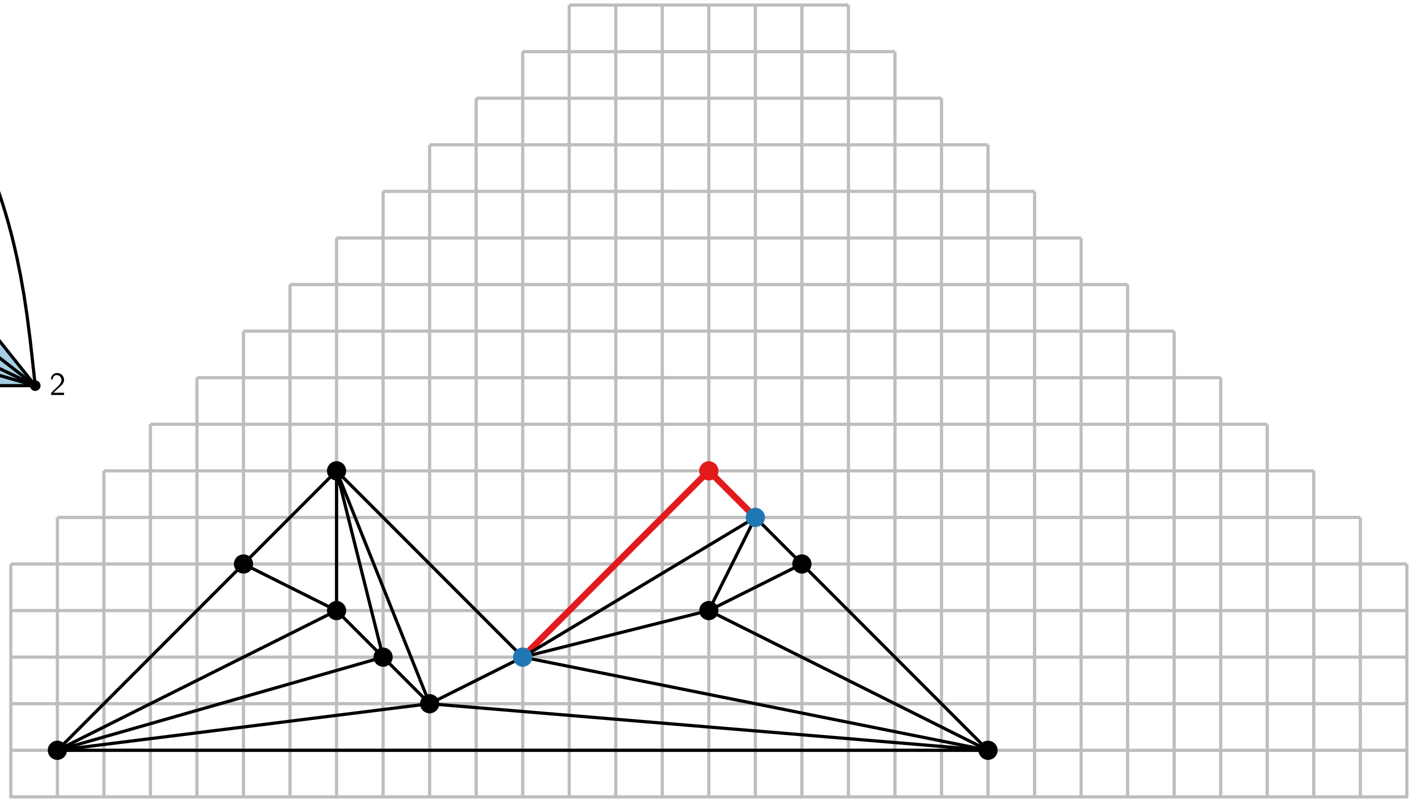


Shift method – example

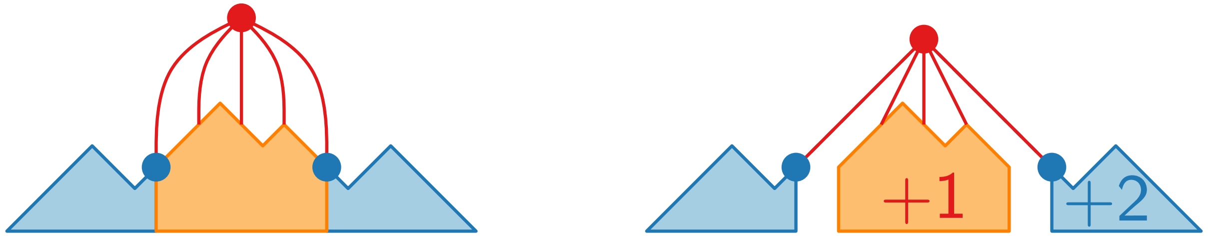


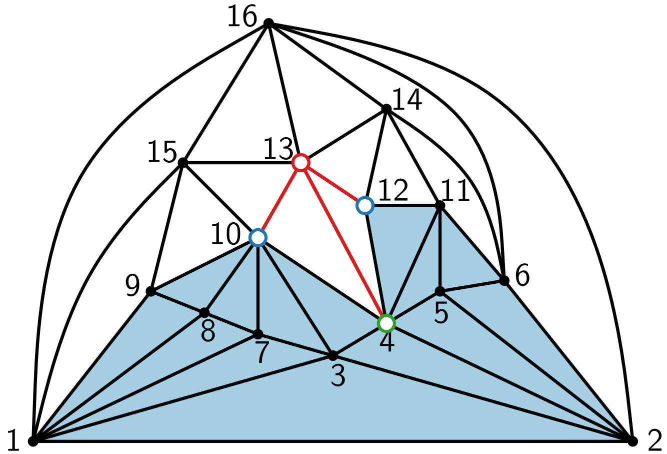


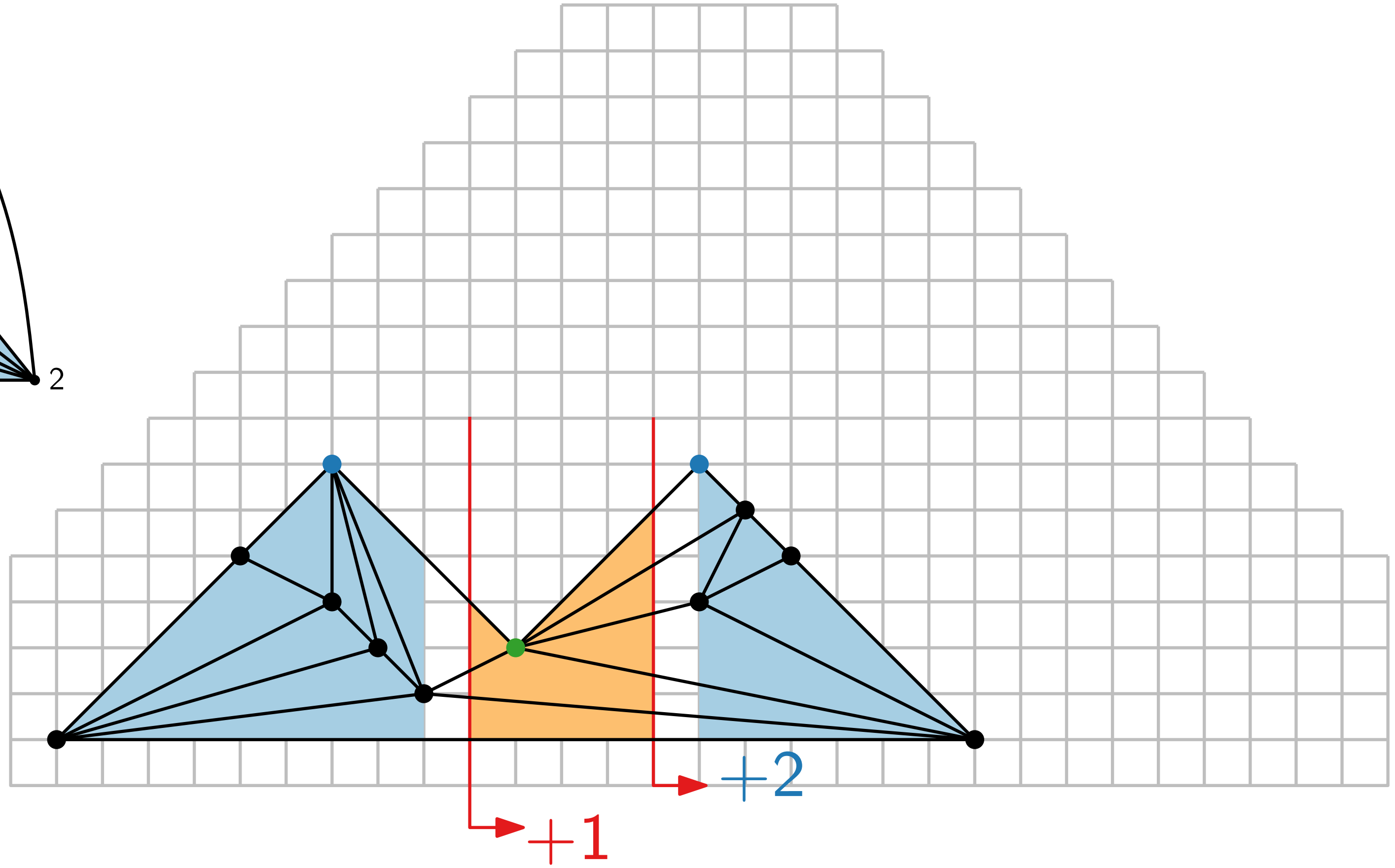




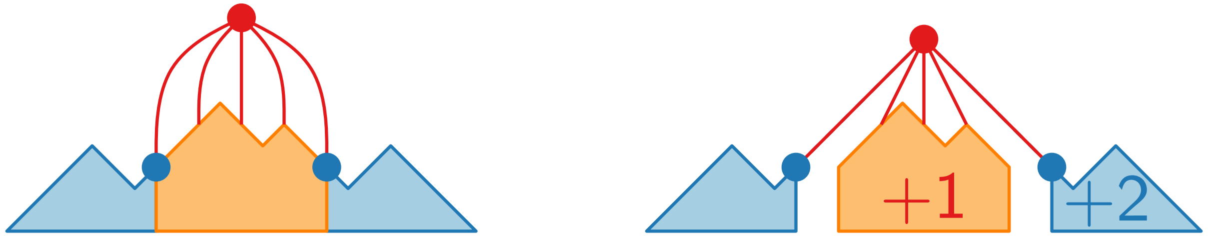
Shift method – example

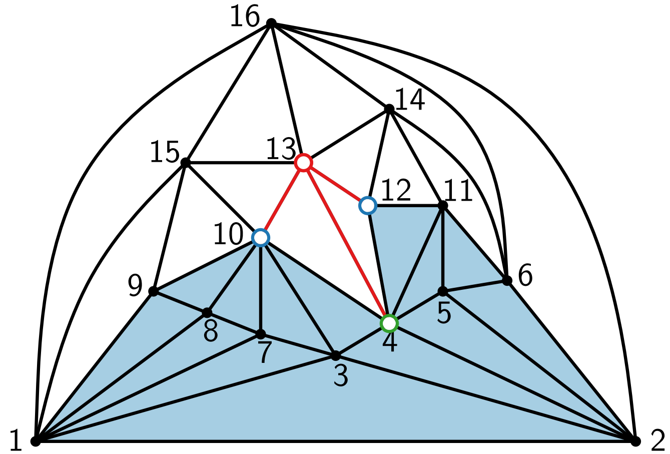


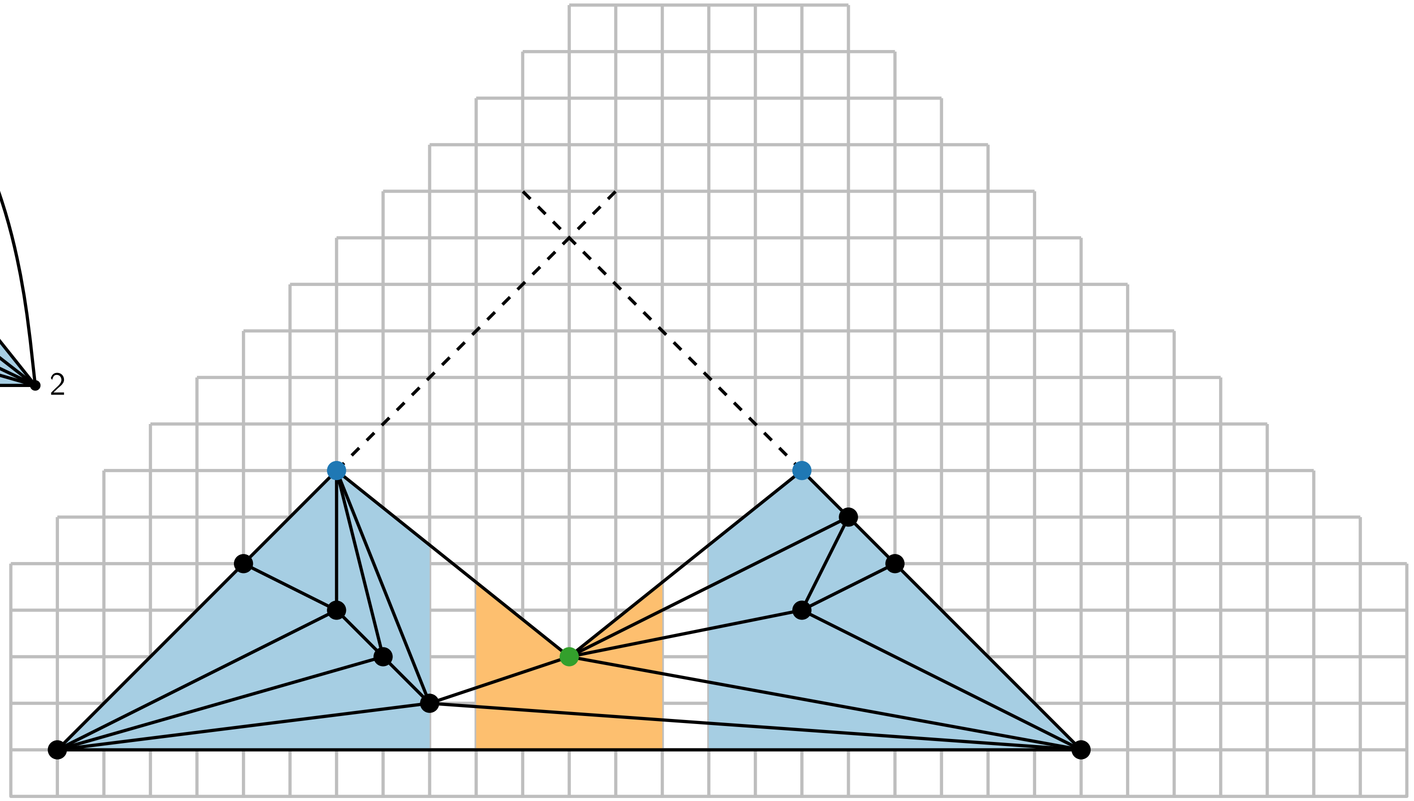




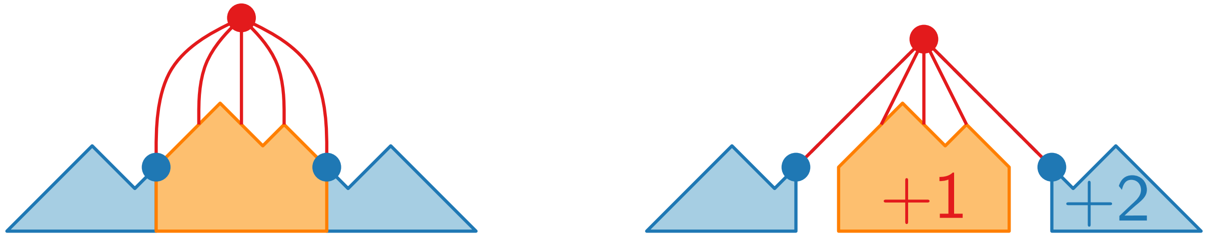
Shift method – example

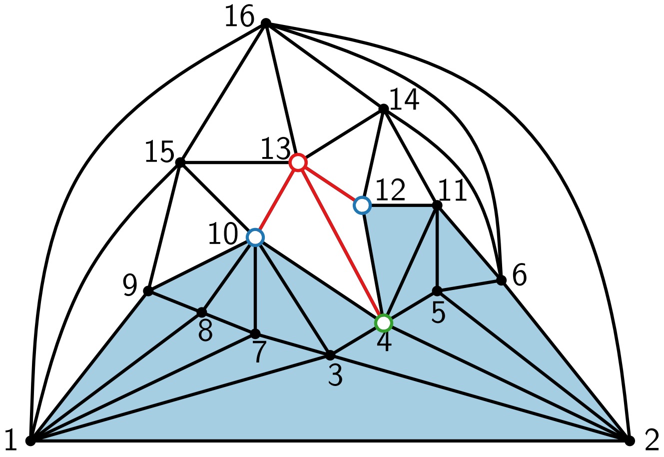


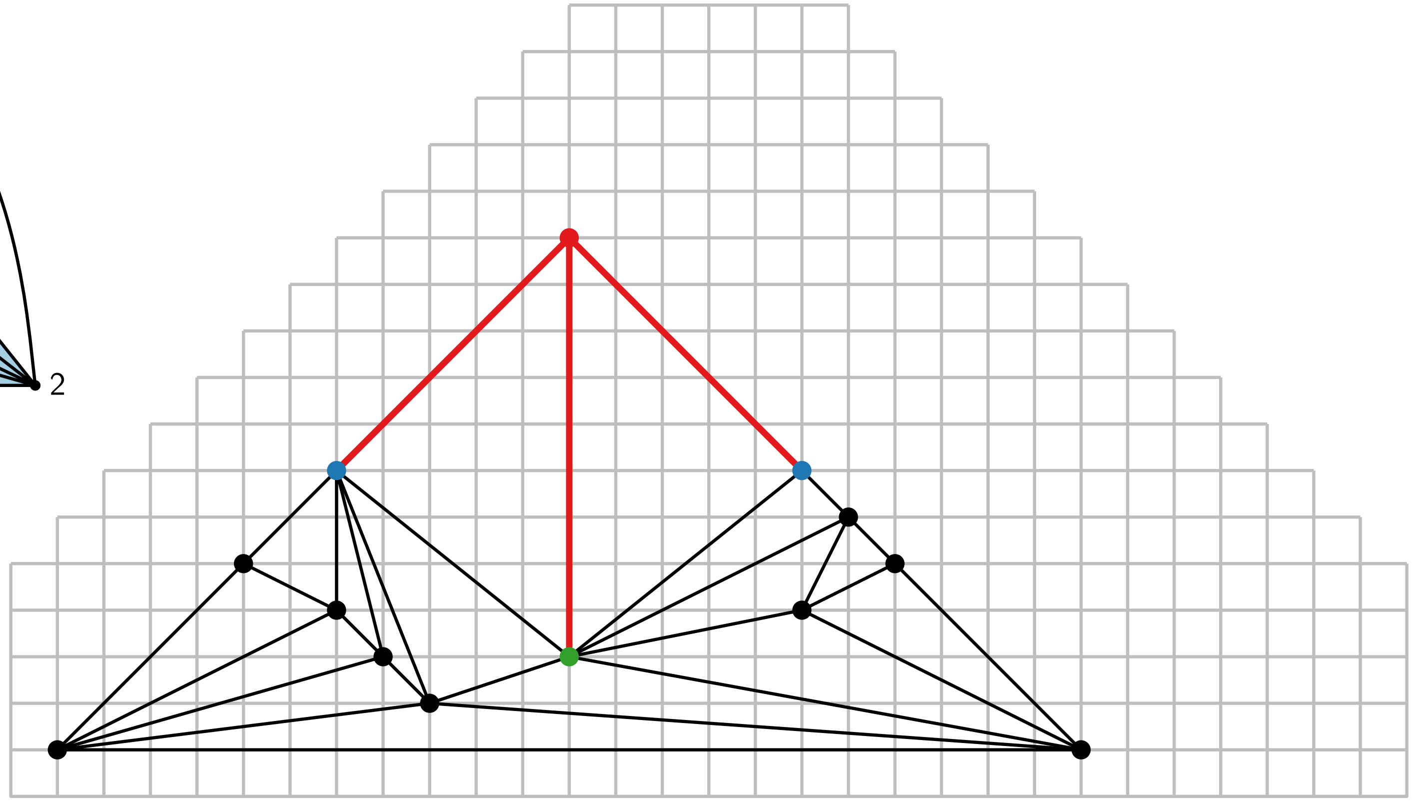




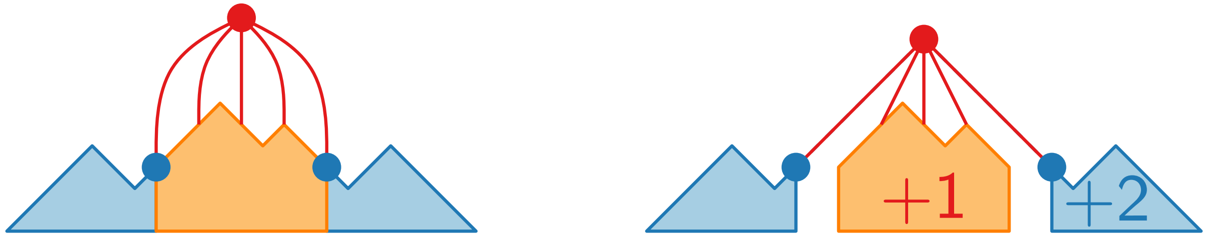
Shift method – example

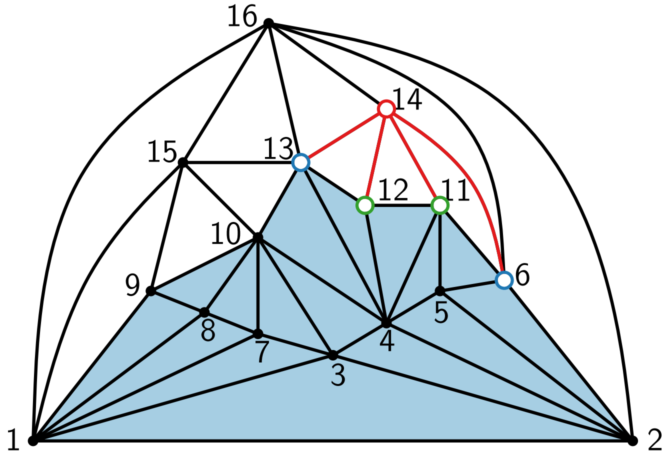


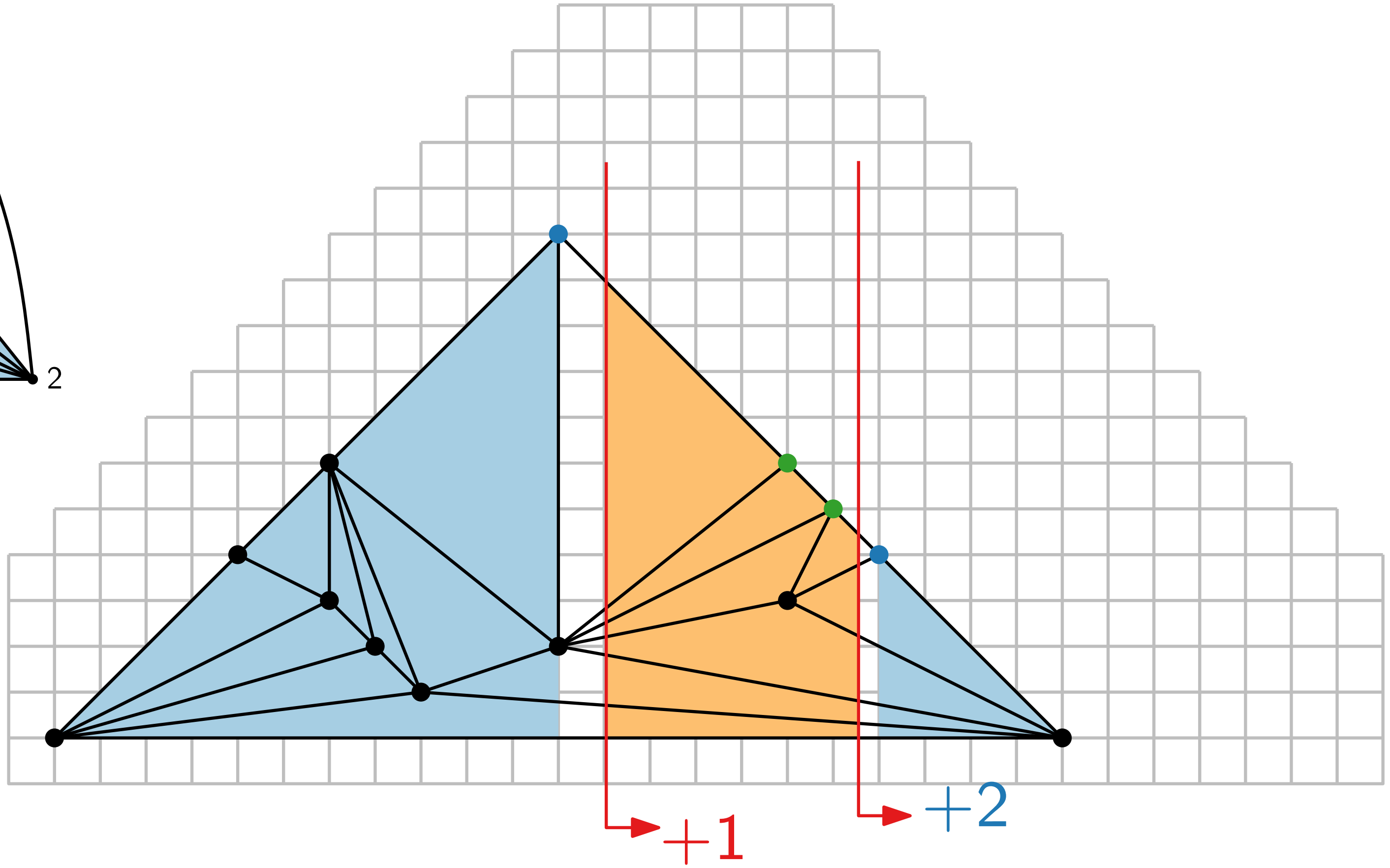




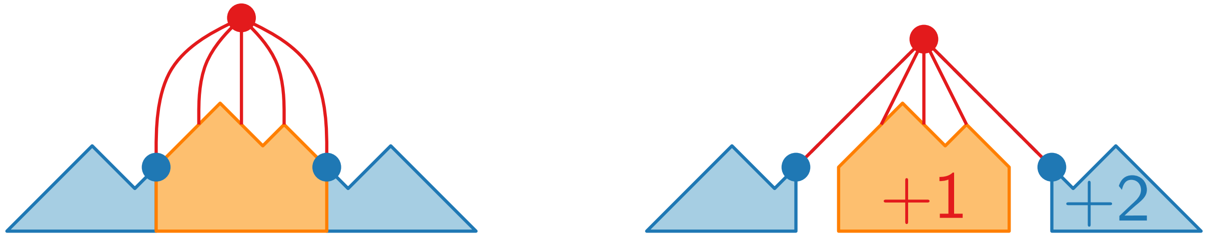
Shift method – example

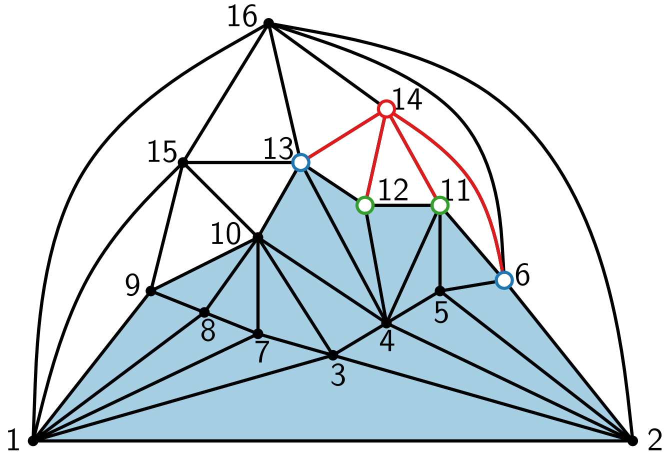


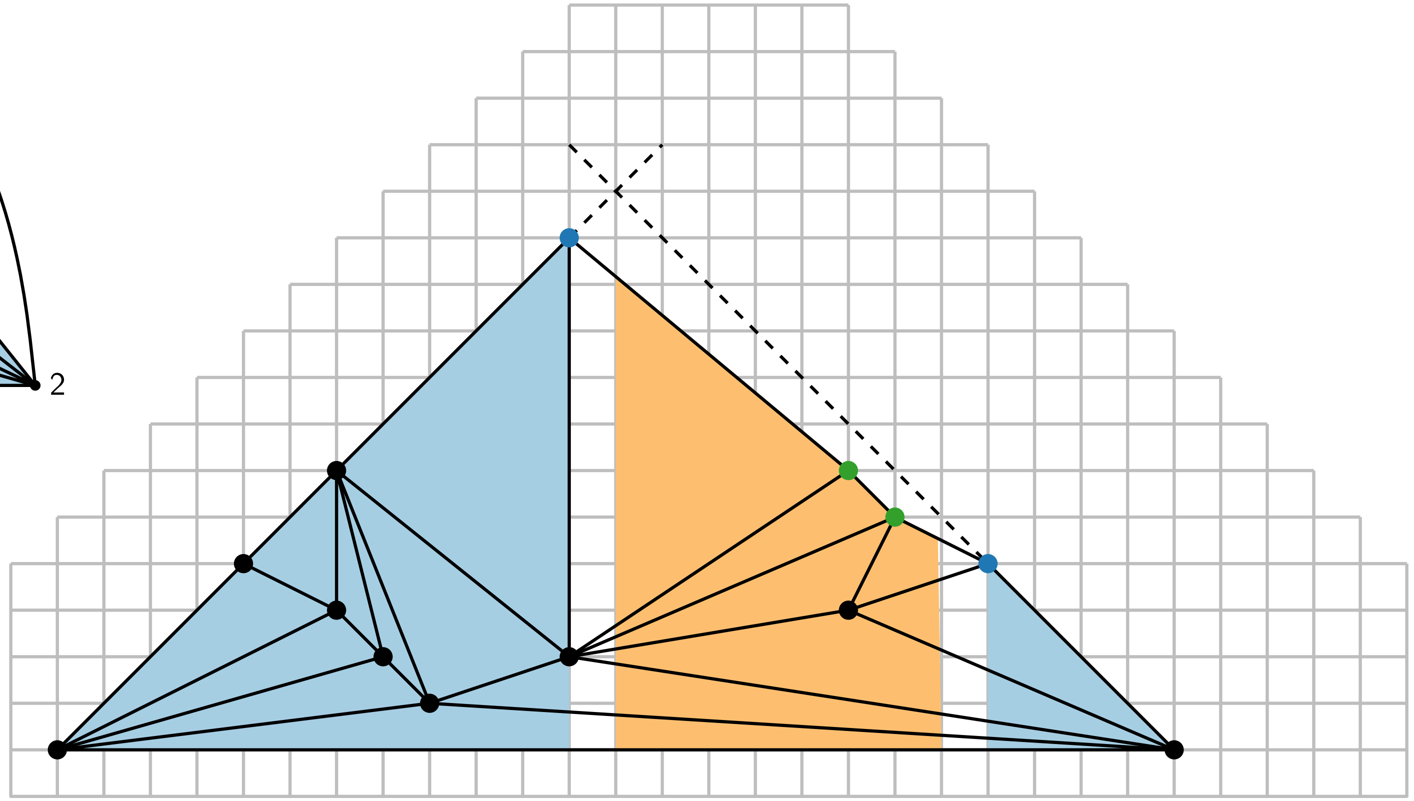




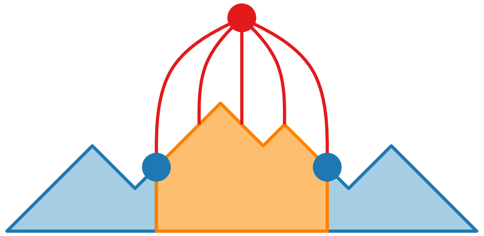
Shift method – example

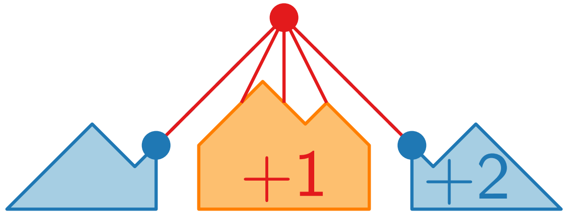


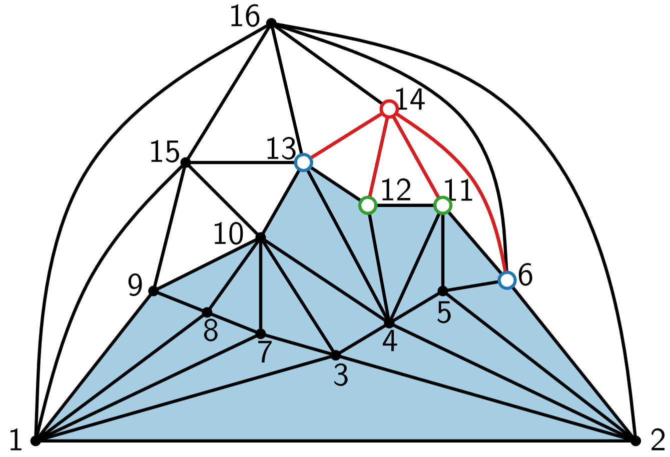


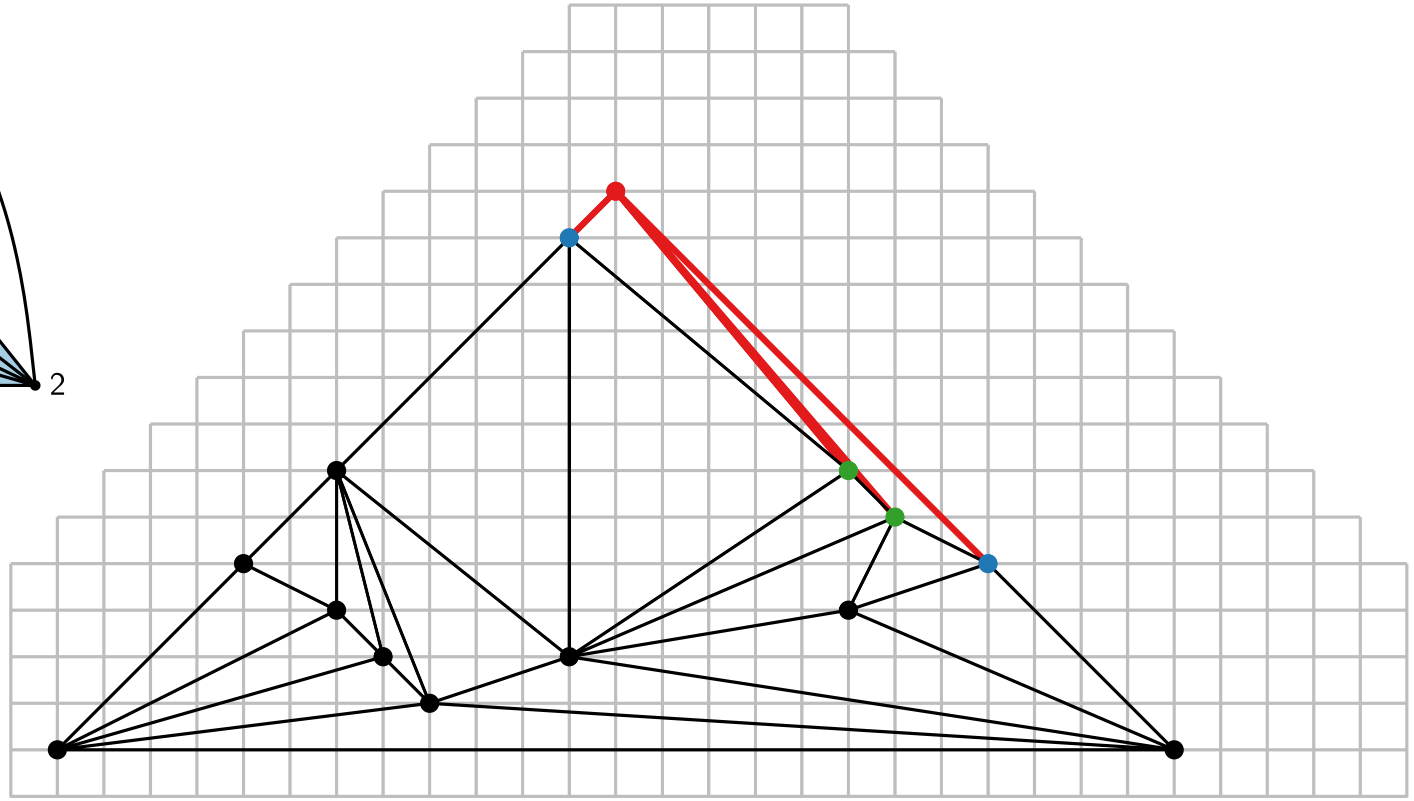


Shift method – example

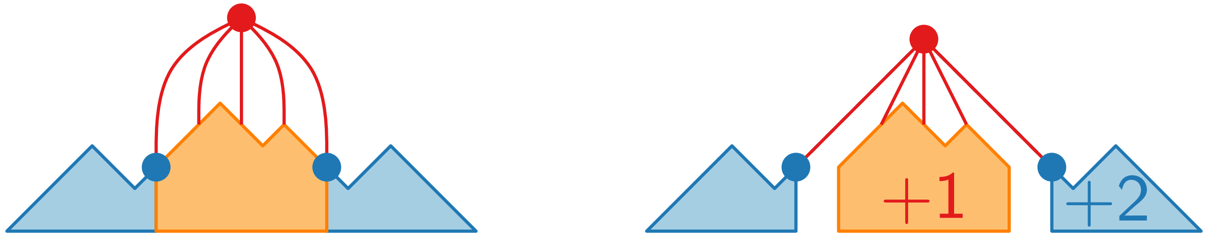


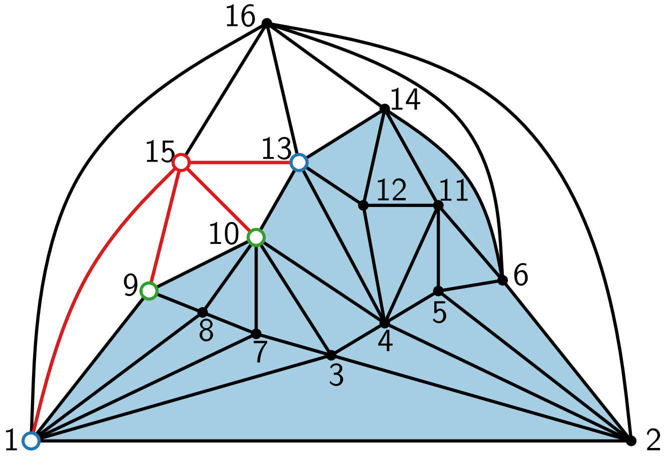


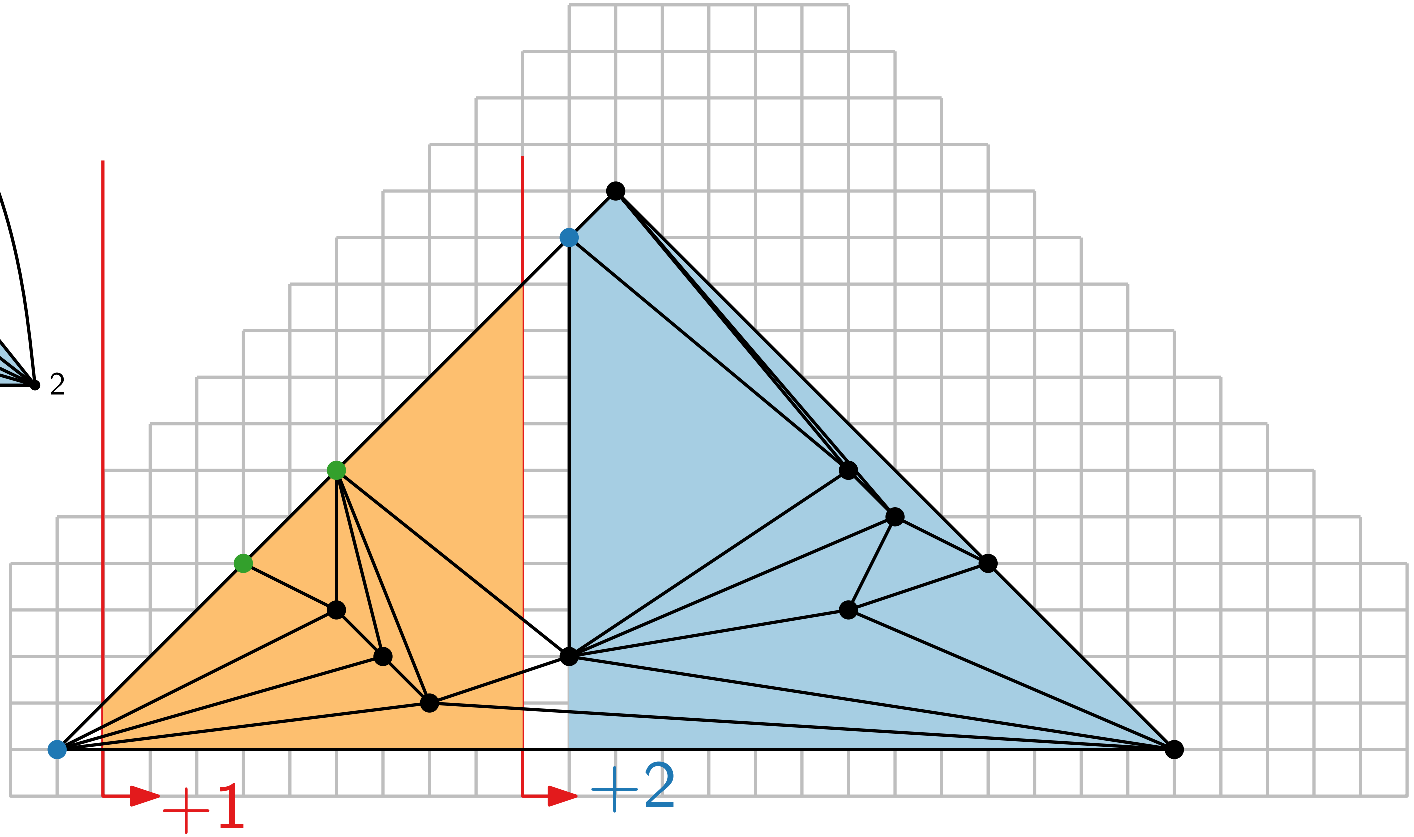




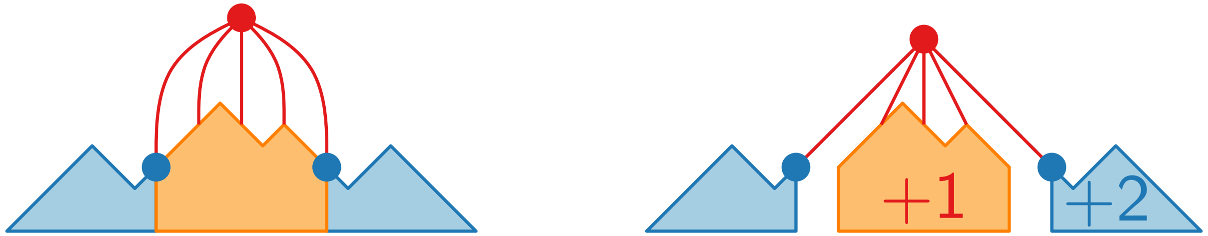
Shift method – example

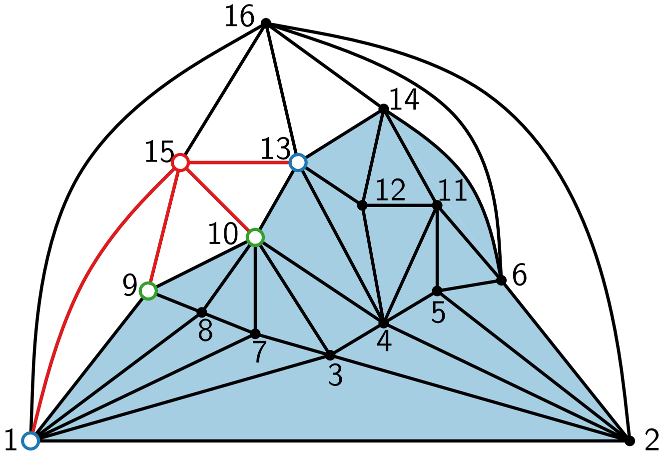


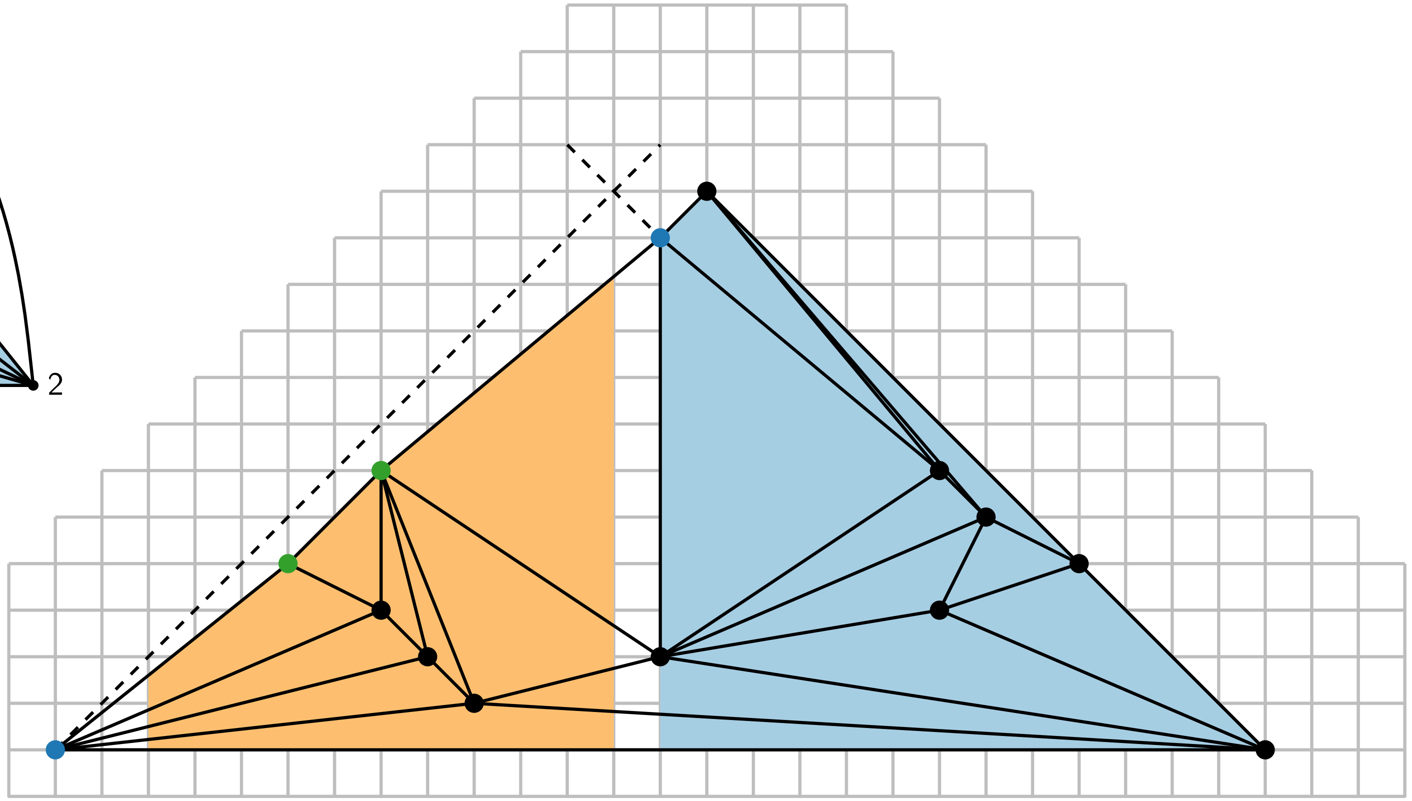




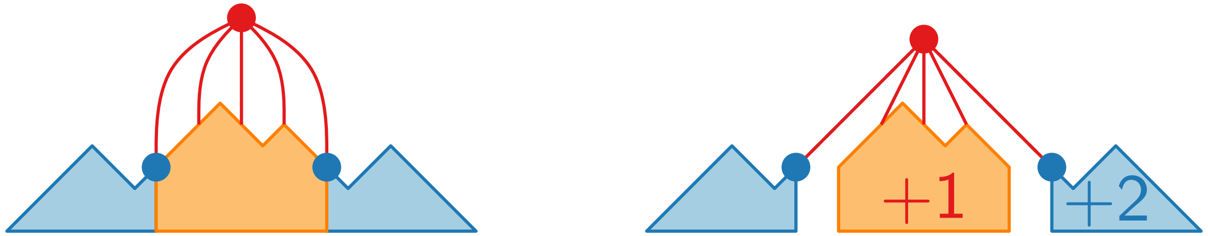
Shift method – example

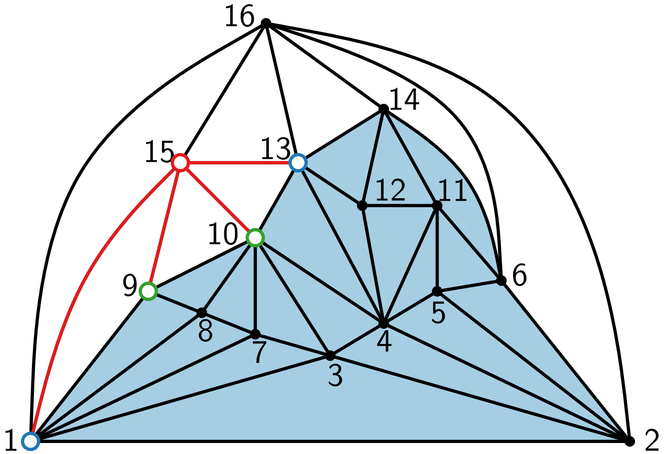


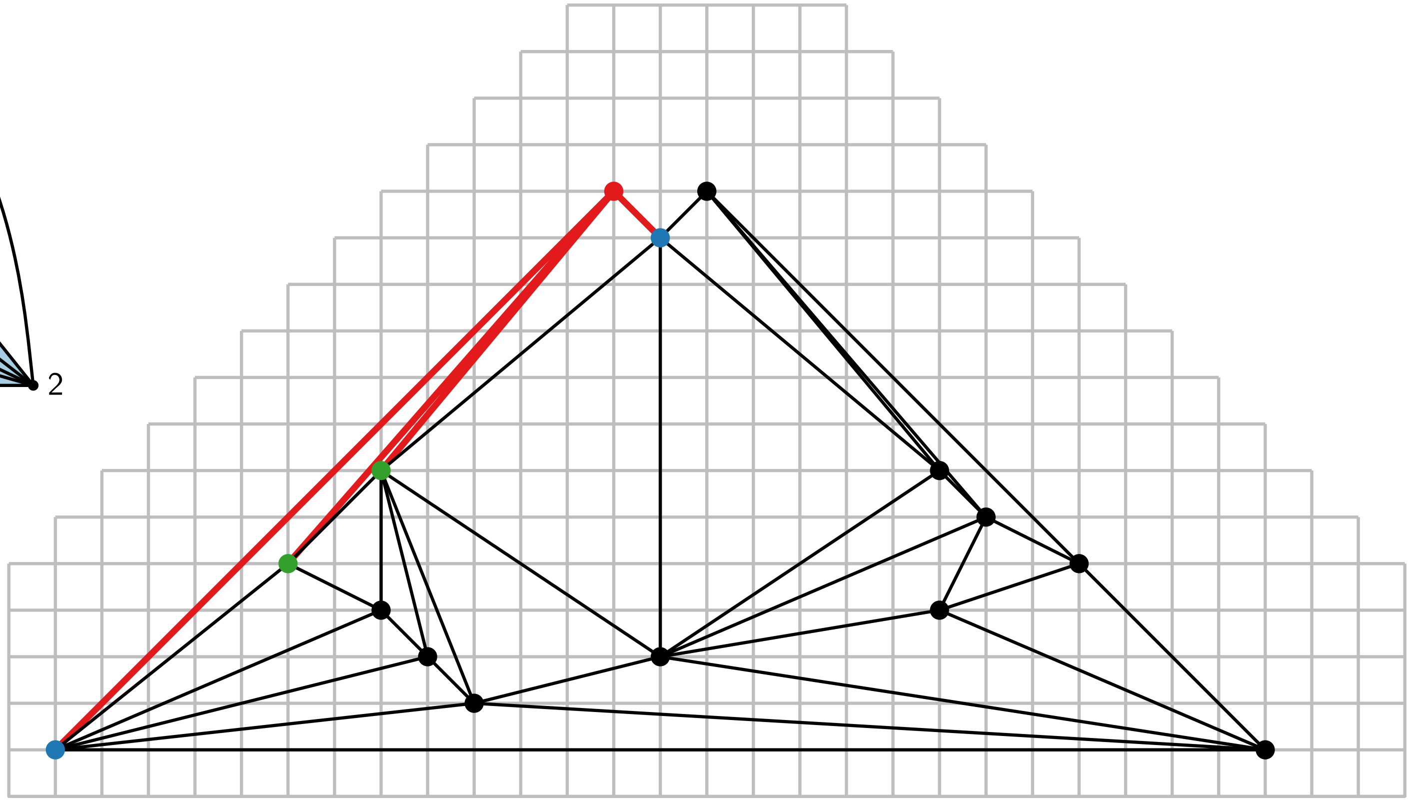




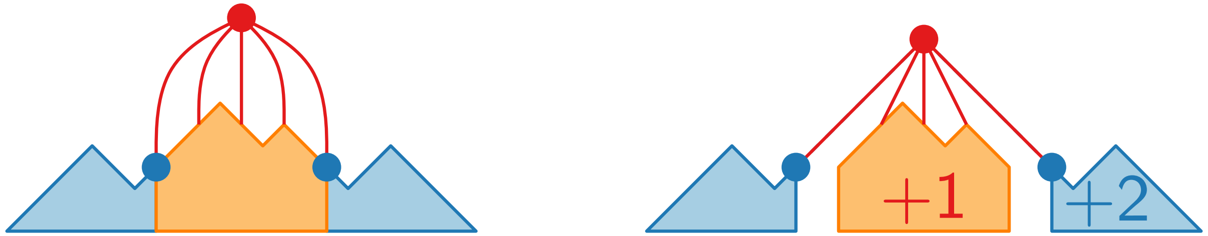
Shift method – example

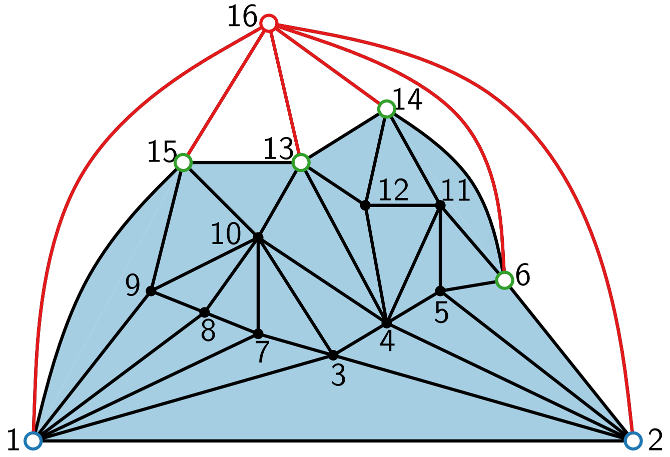


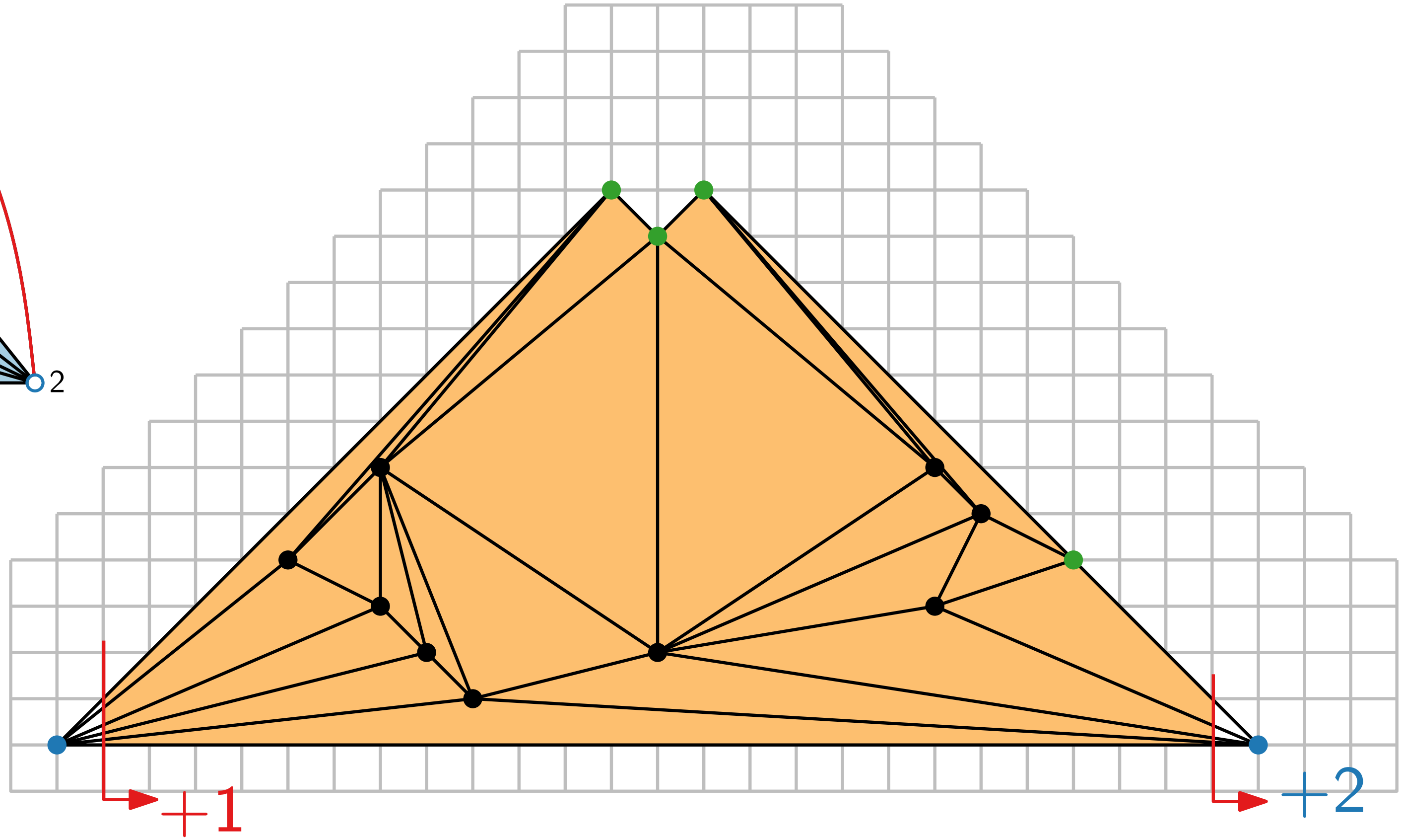




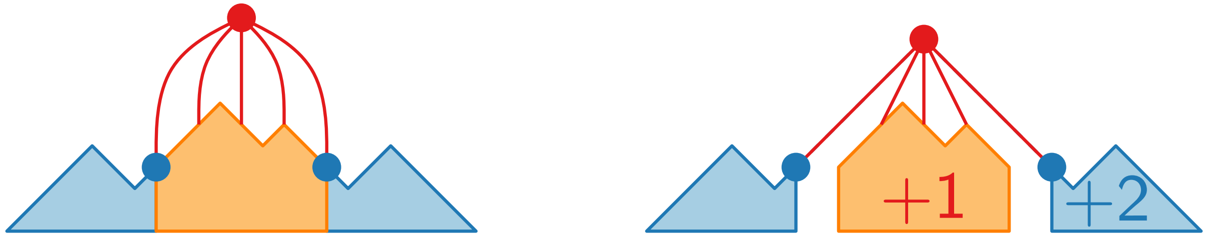
Shift method – example

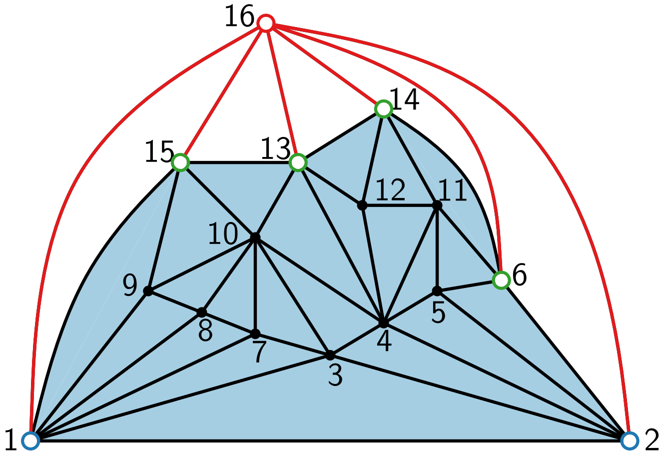


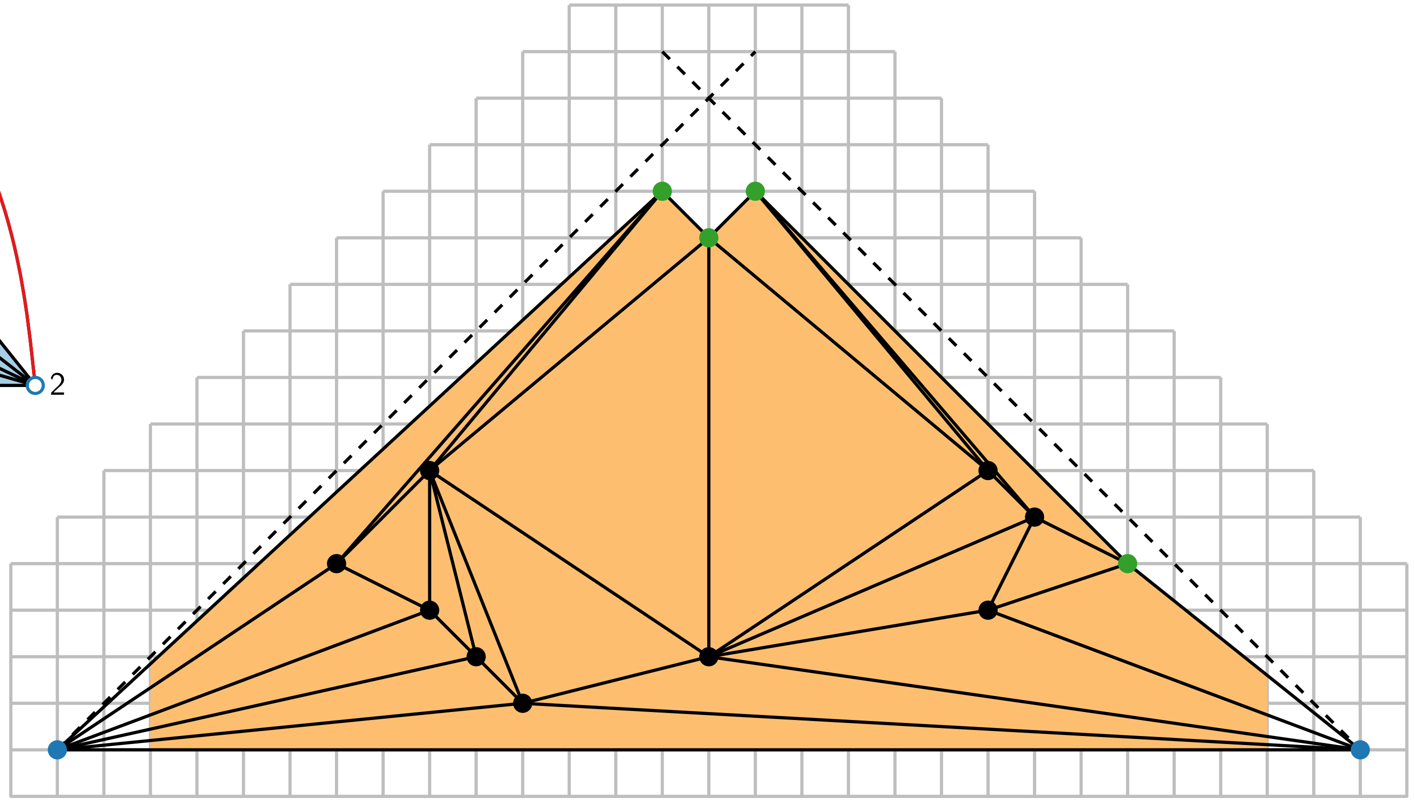




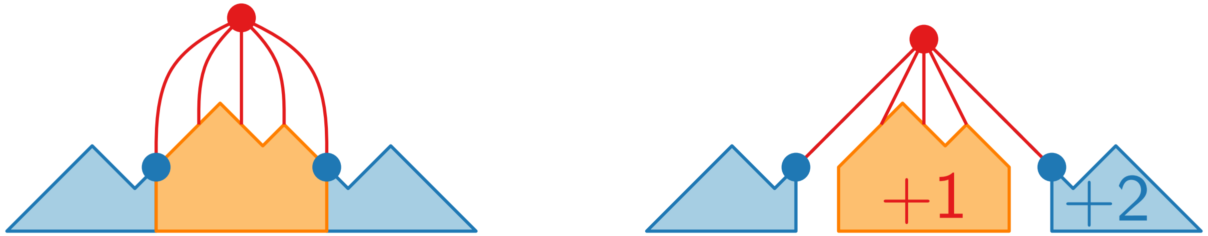
Shift method – example

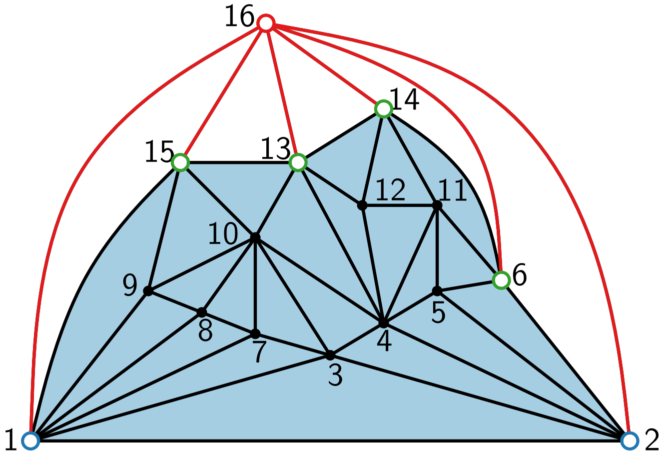


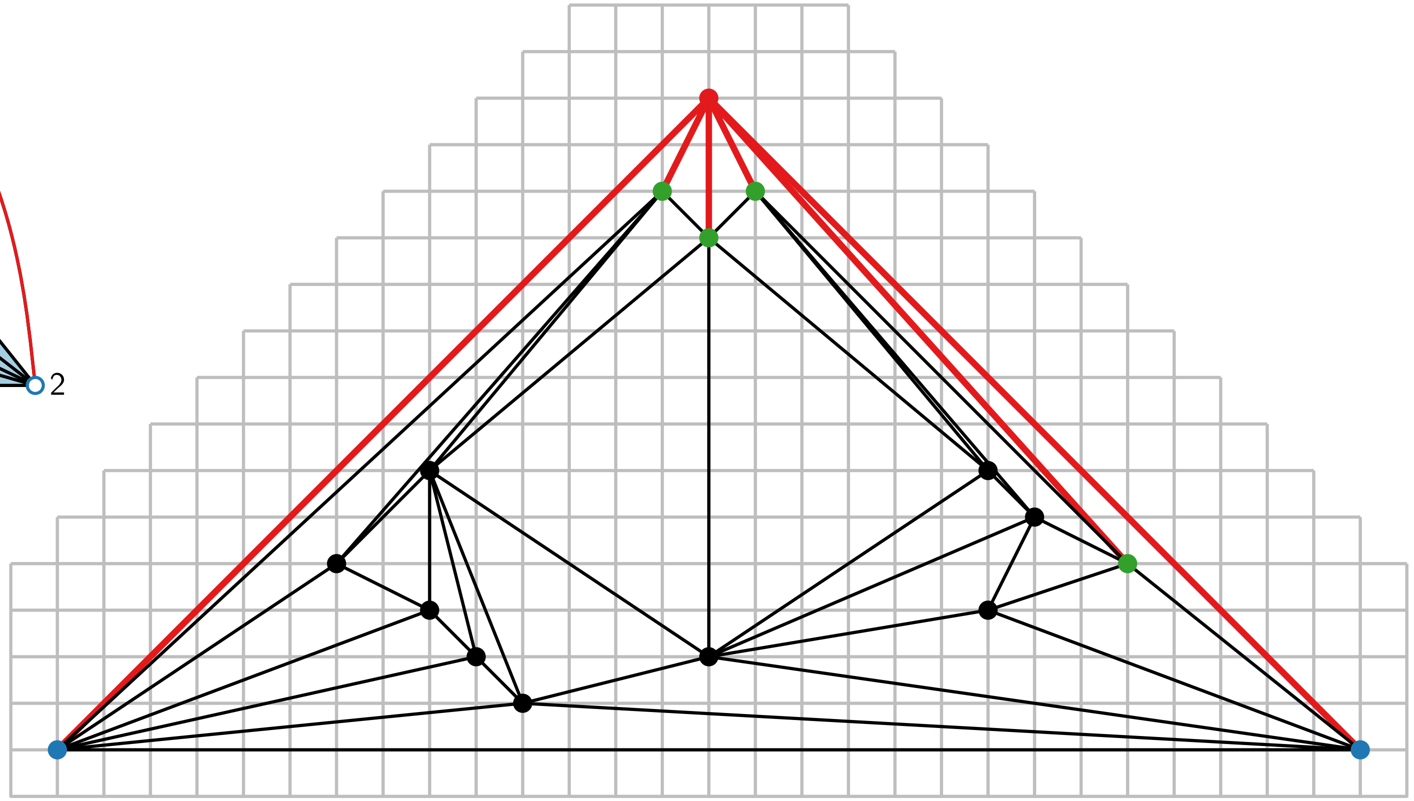




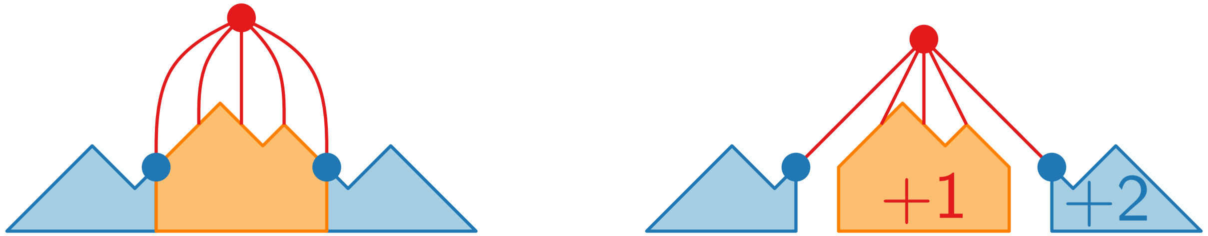
Shift method – example

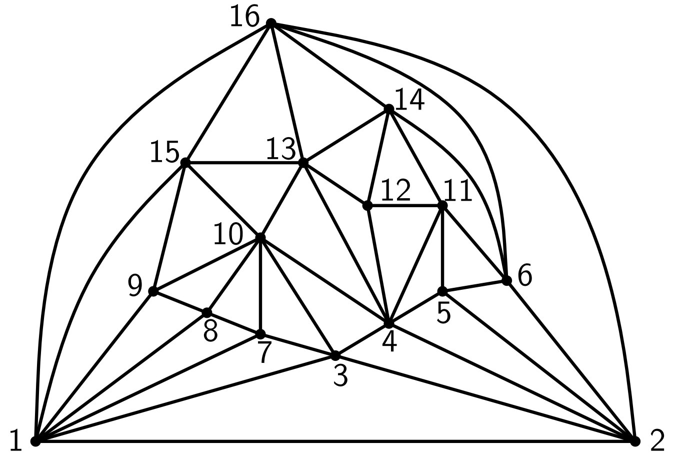


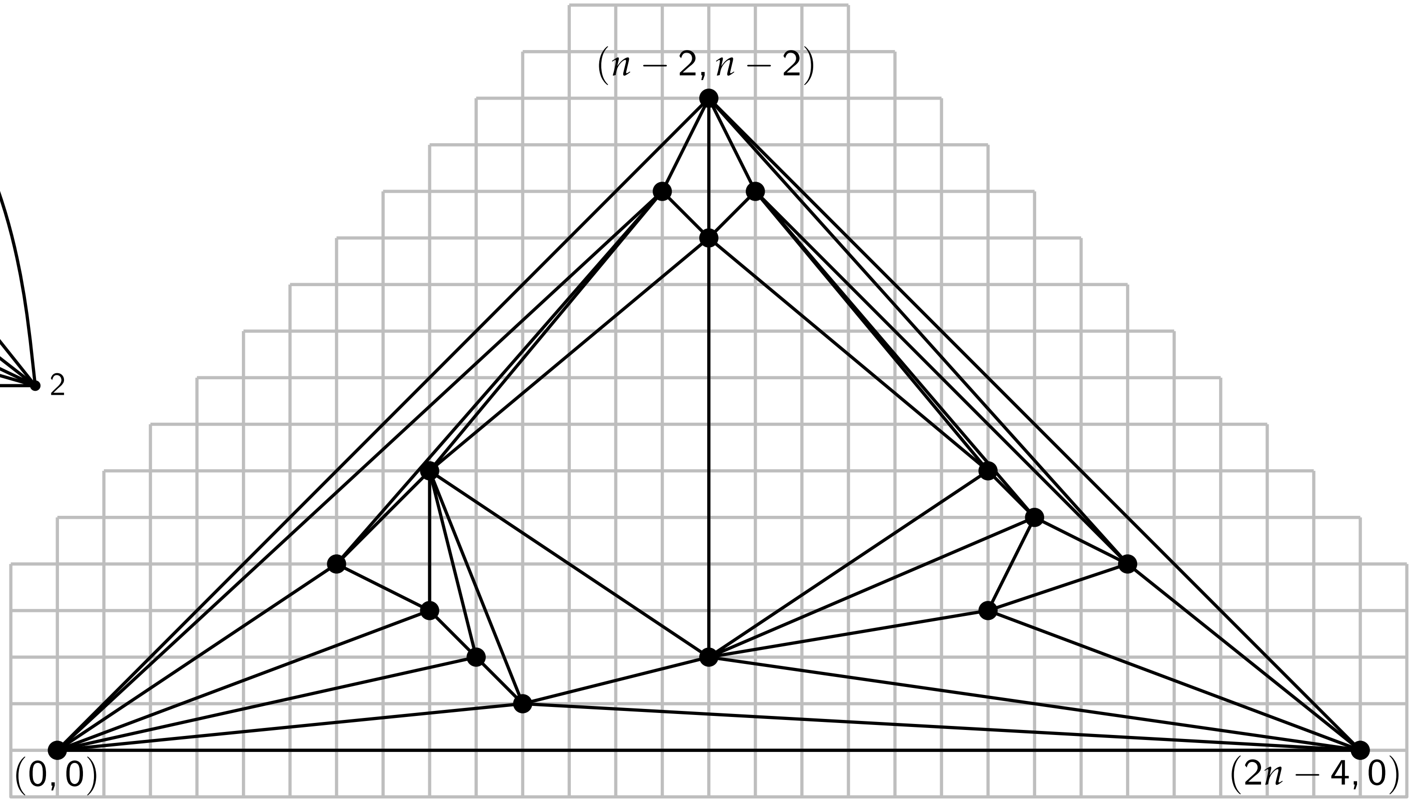




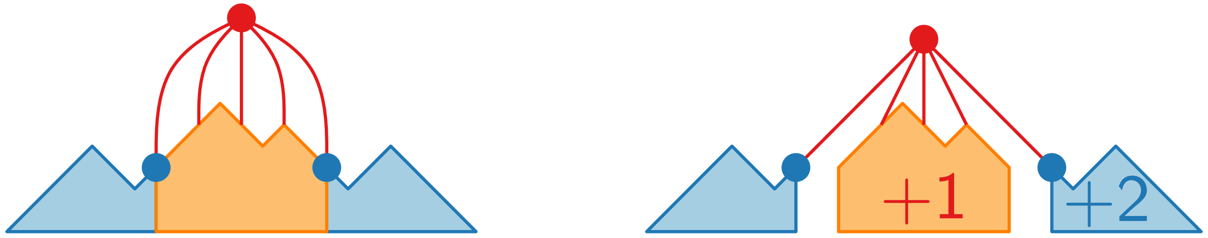
Shift method – example

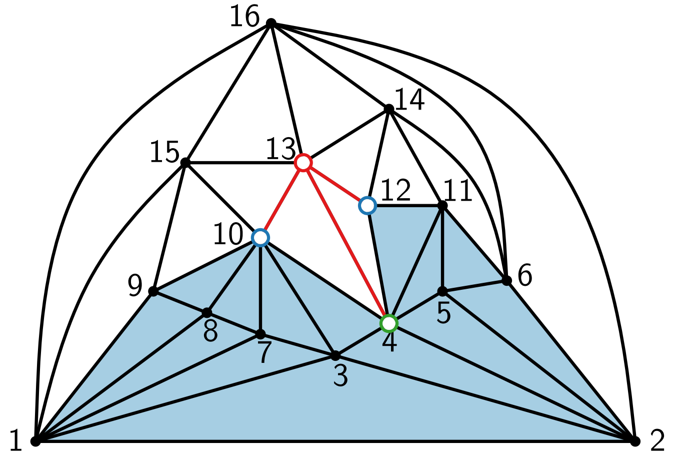


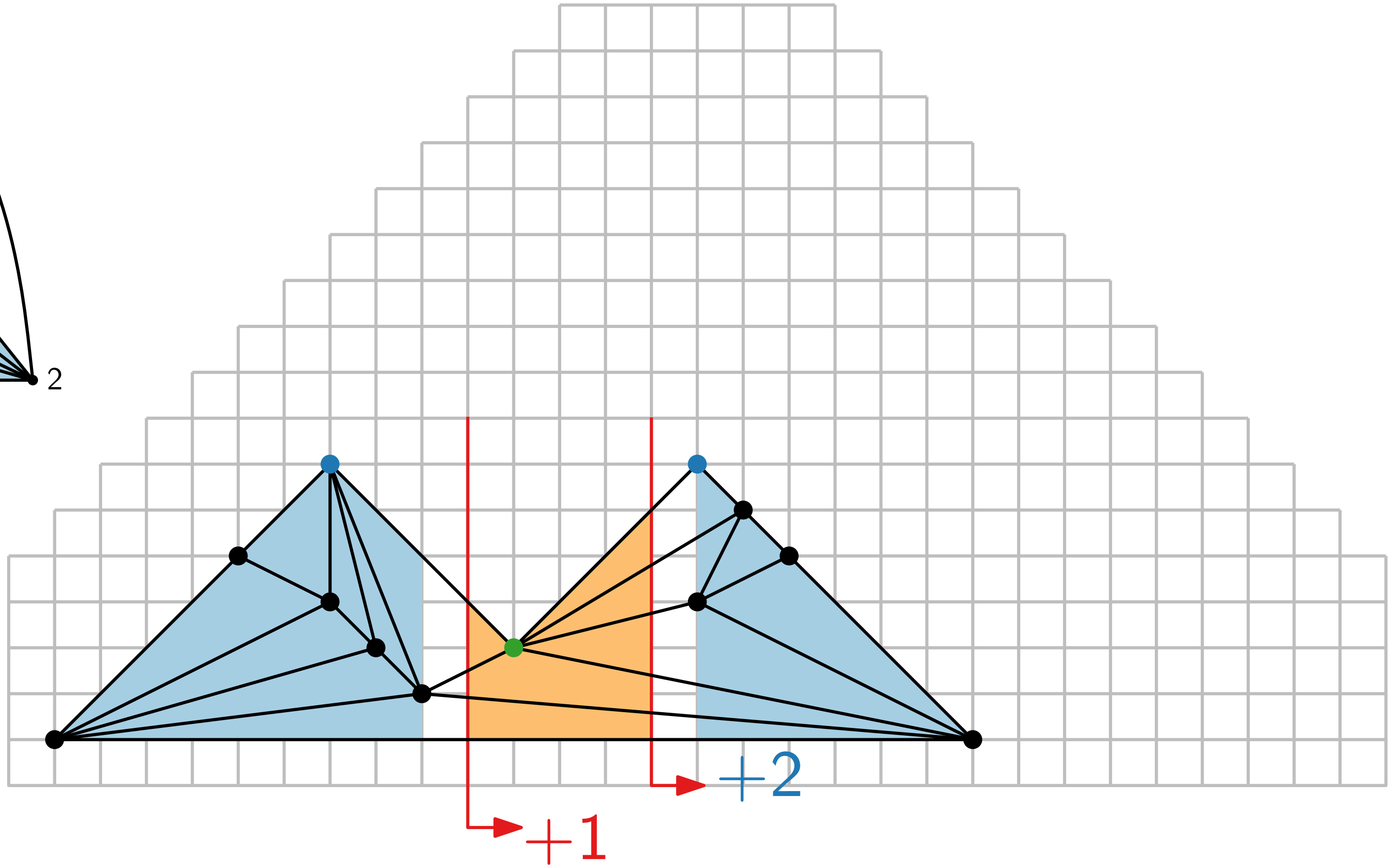




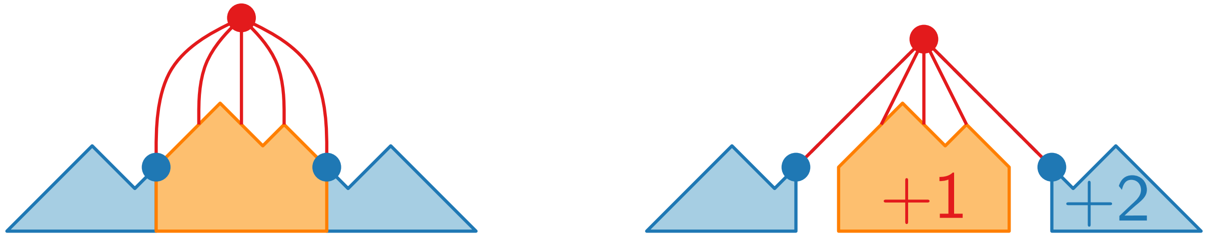
Shift method – example

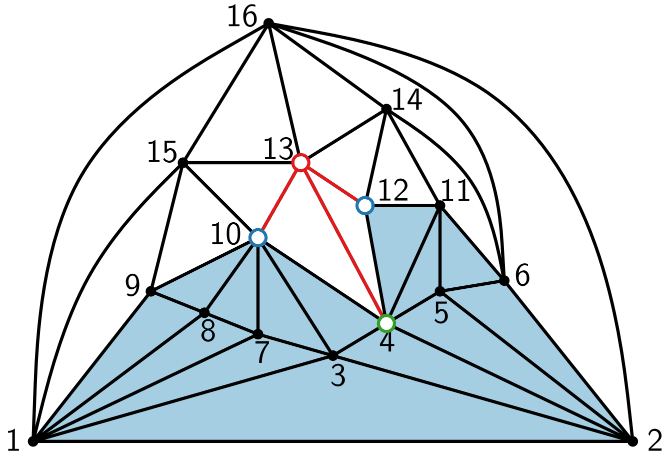


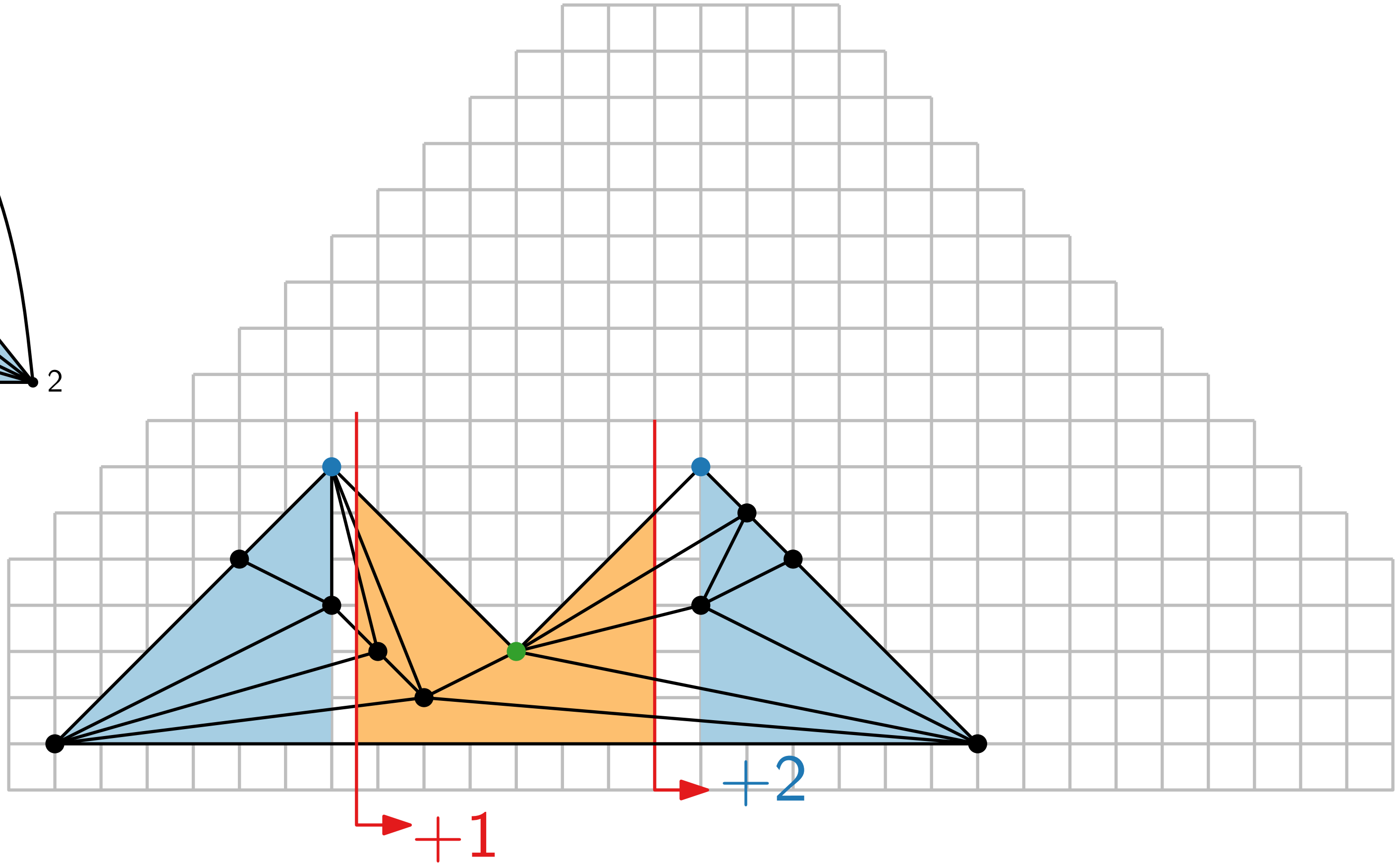




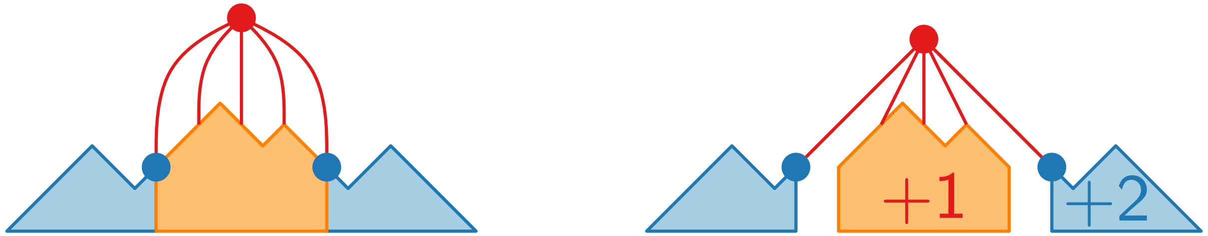
Shift method – example



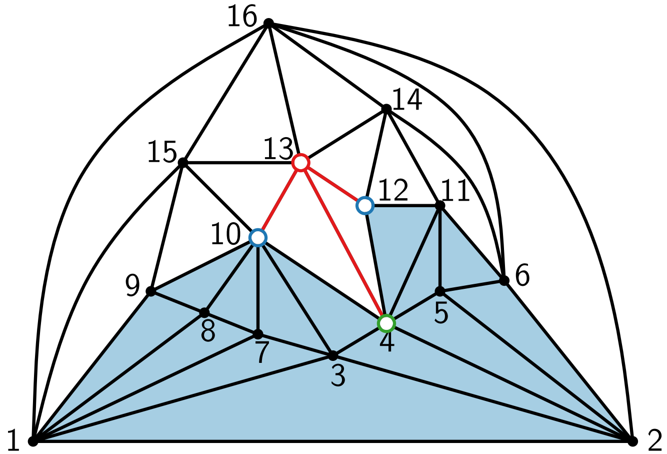


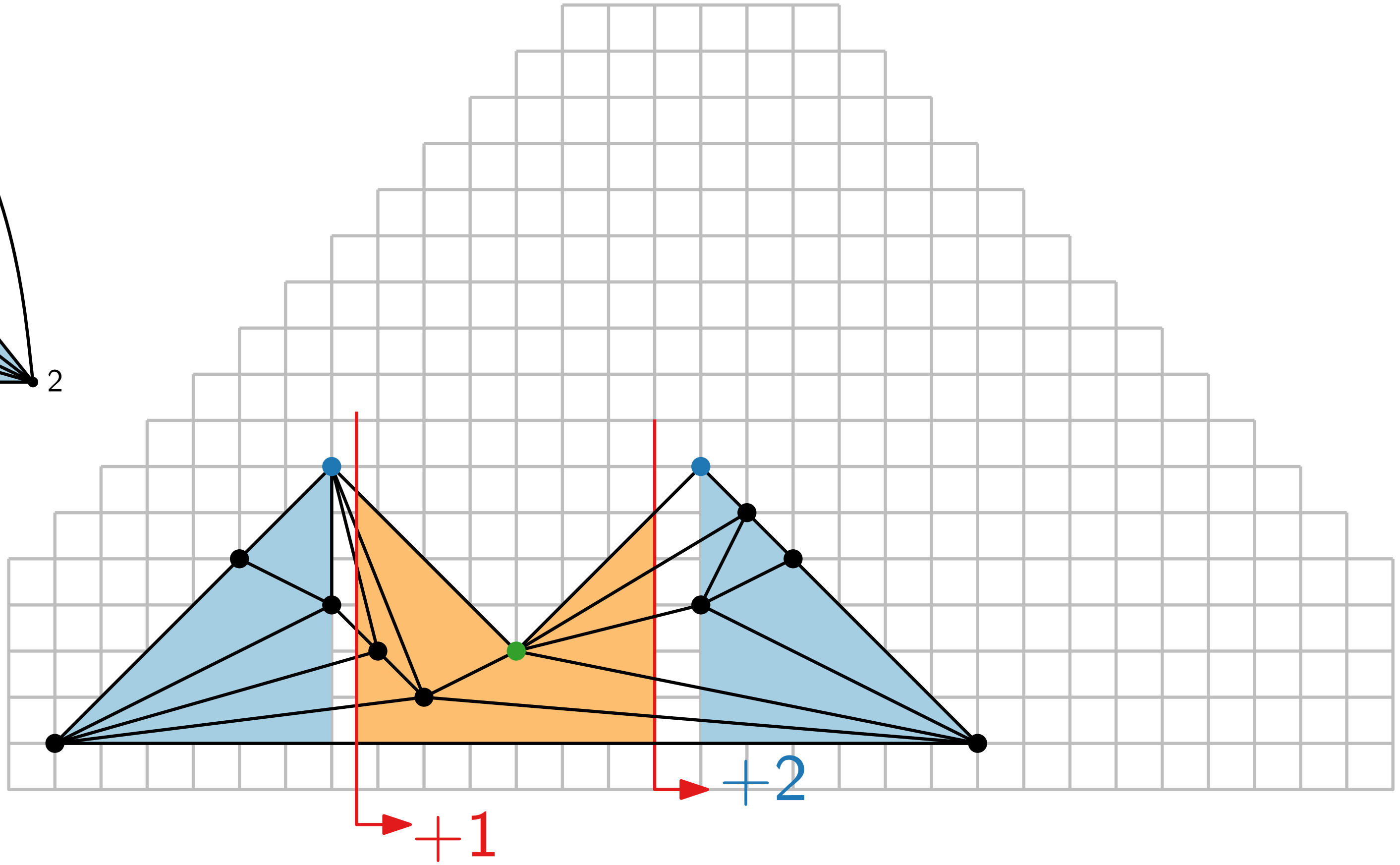


Shift method – example

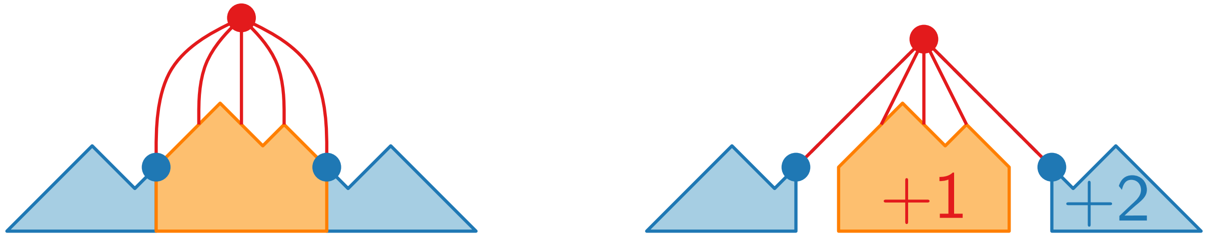


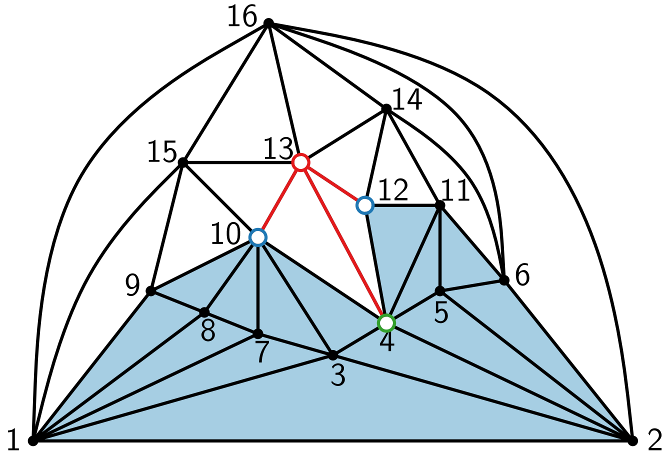
Which internal nodes are shifted?

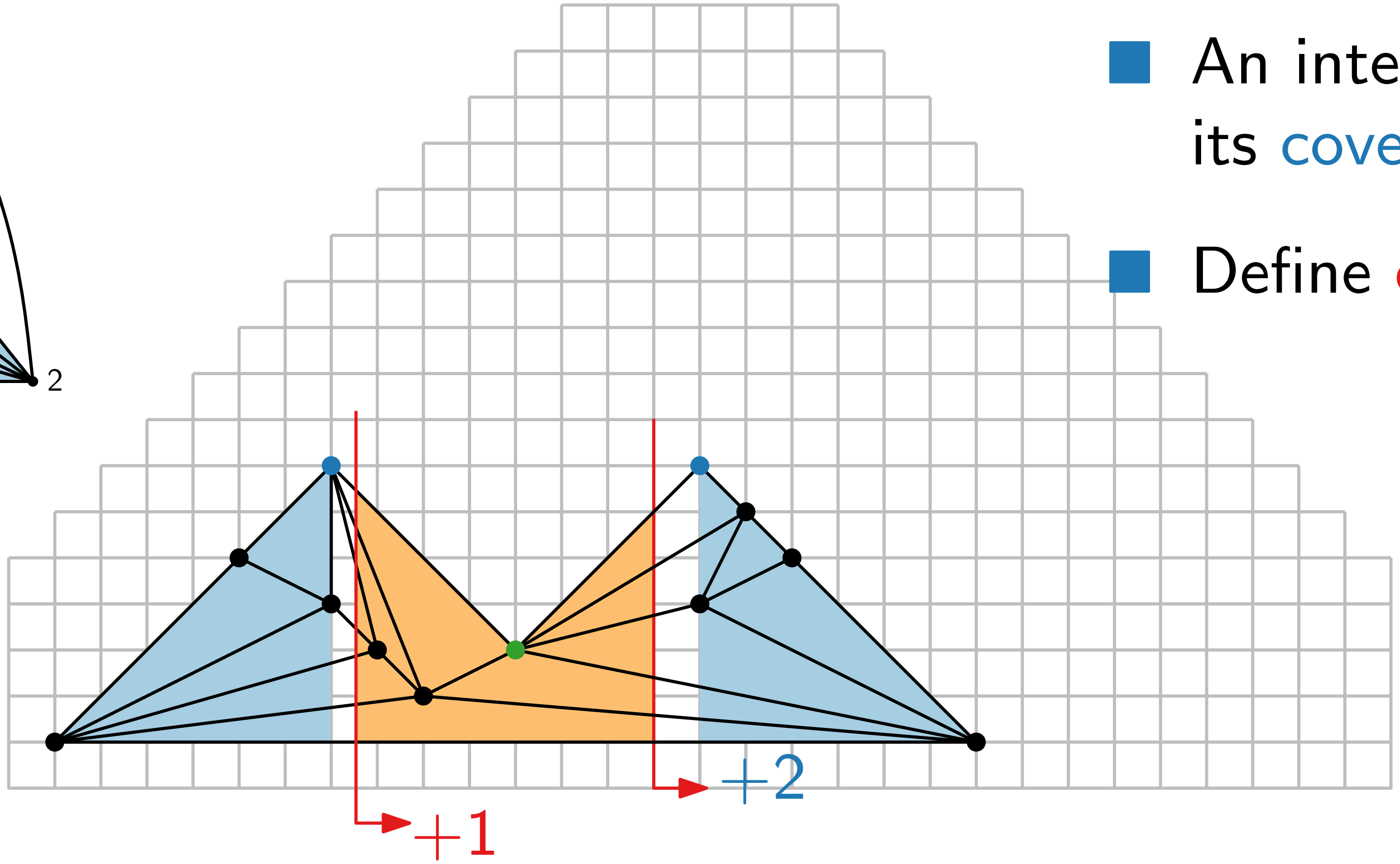




Shift method – example



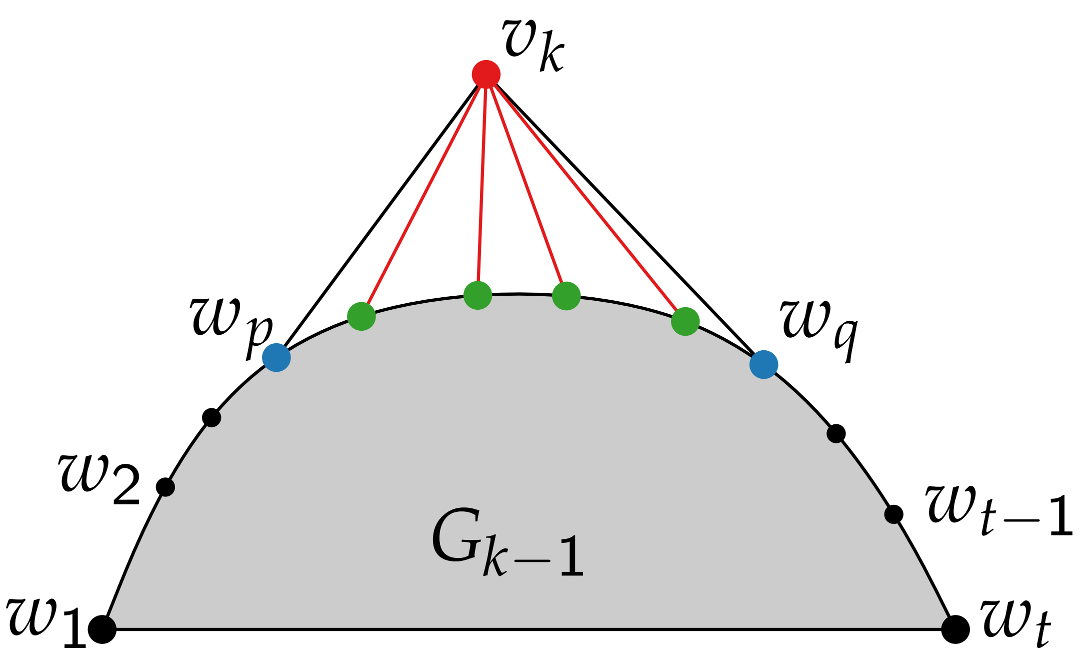




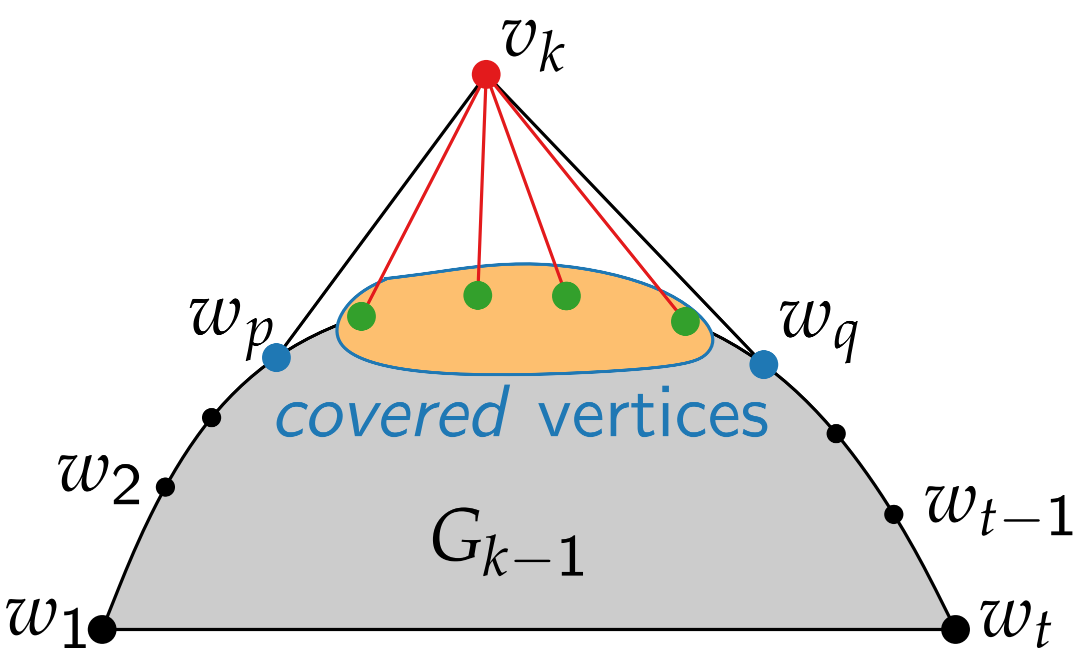
Which internal nodes are shifted?

- An internal node shifts with its covering outer vertex
- Define covering

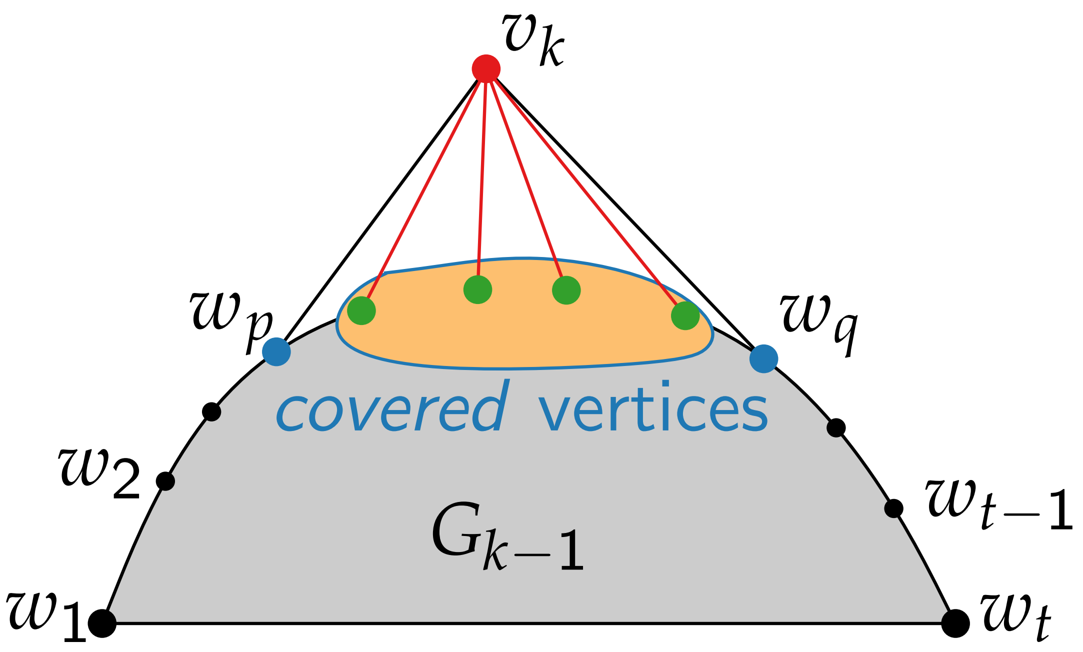
Shift method – dominating



Shift method – dominating



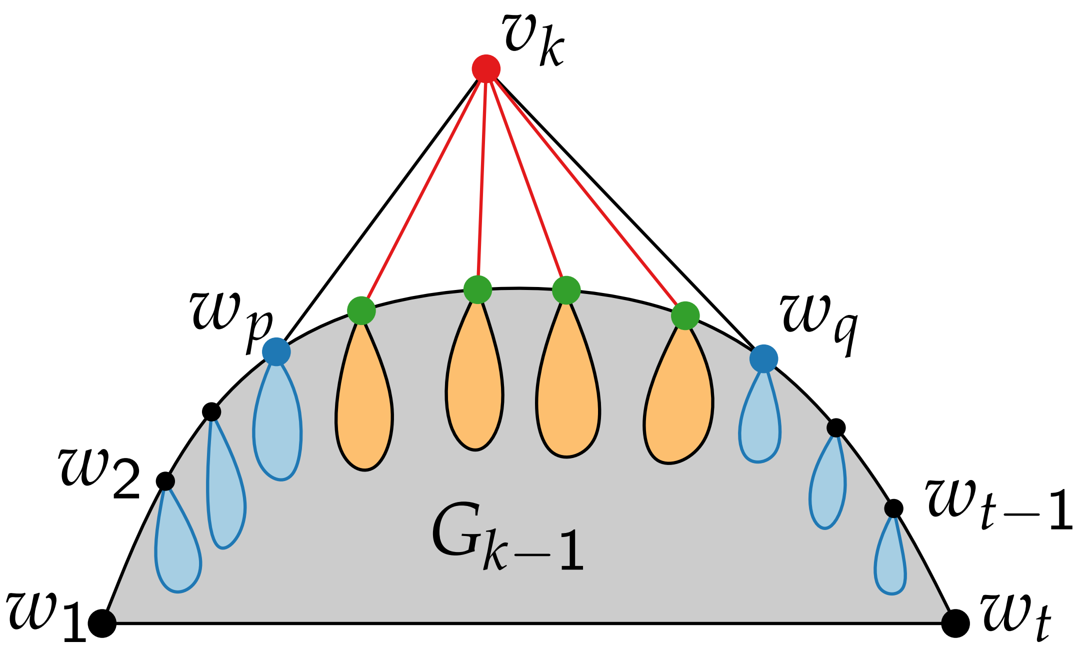
Shift method – dominating



Observations.

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

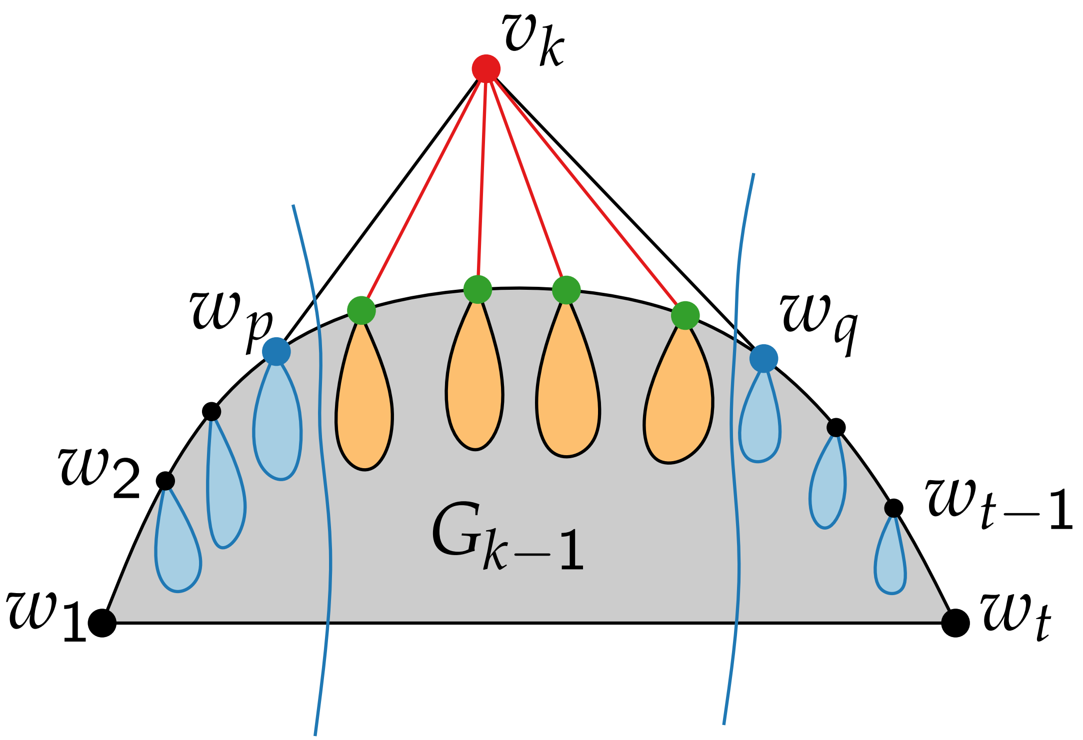
Shift method – dominating



Observations.

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

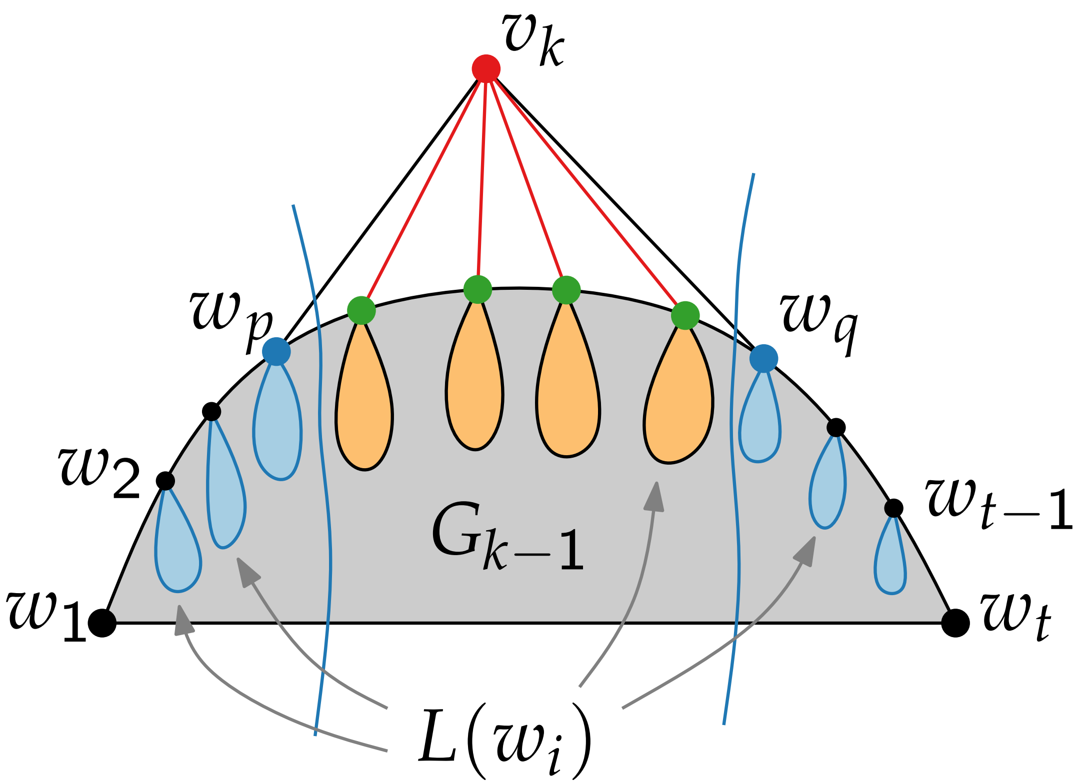
Shift method – dominating



Observations.

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

Shift method – dominating

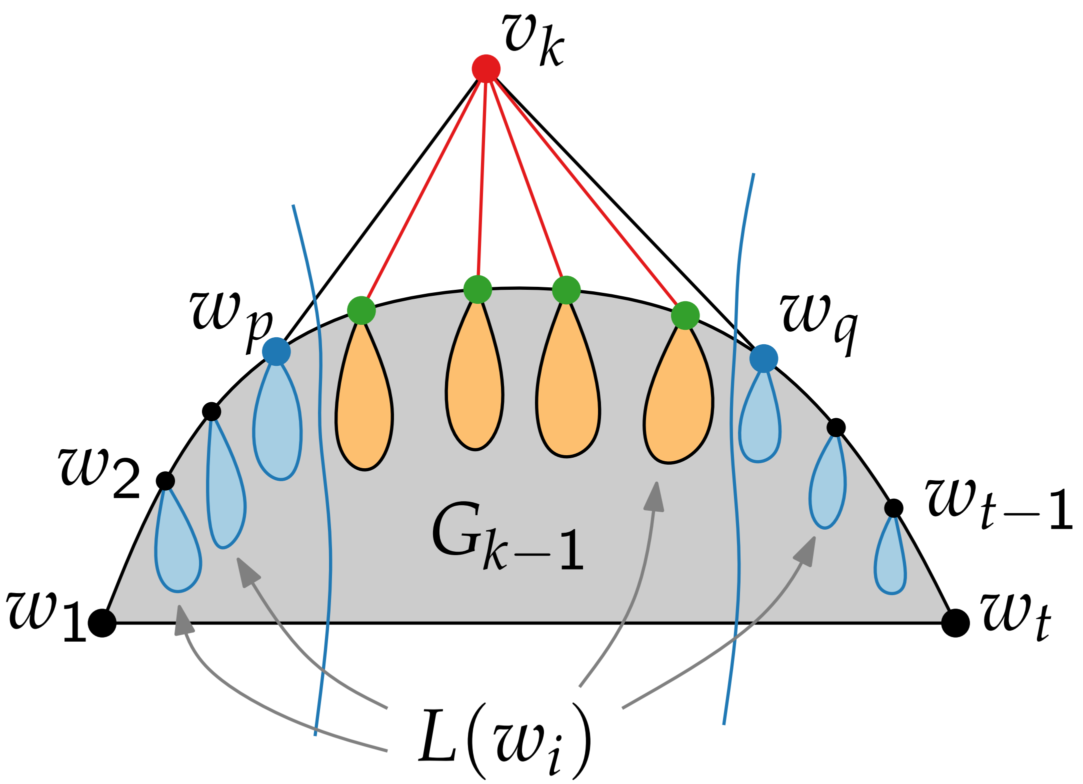


Observations.

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

Shift method – dominating

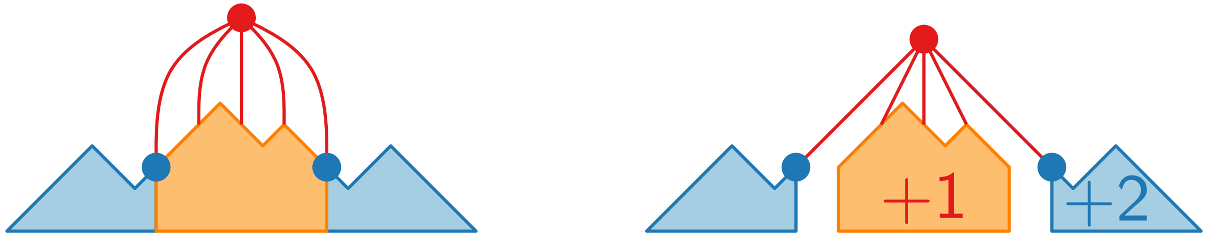
Definition.
$L(w_i)$ is the set of vertices covered by w_i
$L(w_i)$ is the subtree of the covering tree rooted at w_i

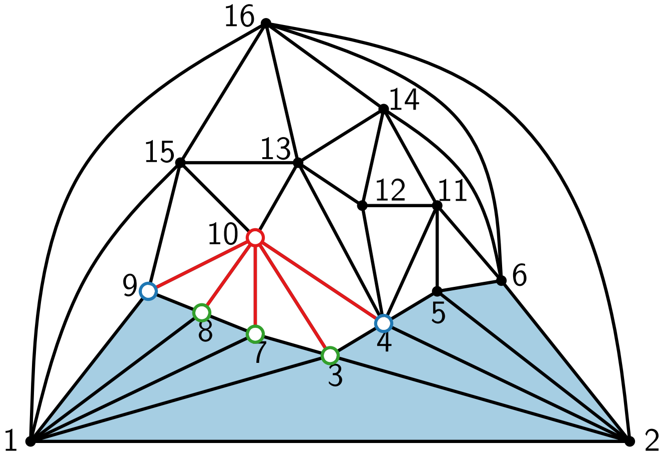


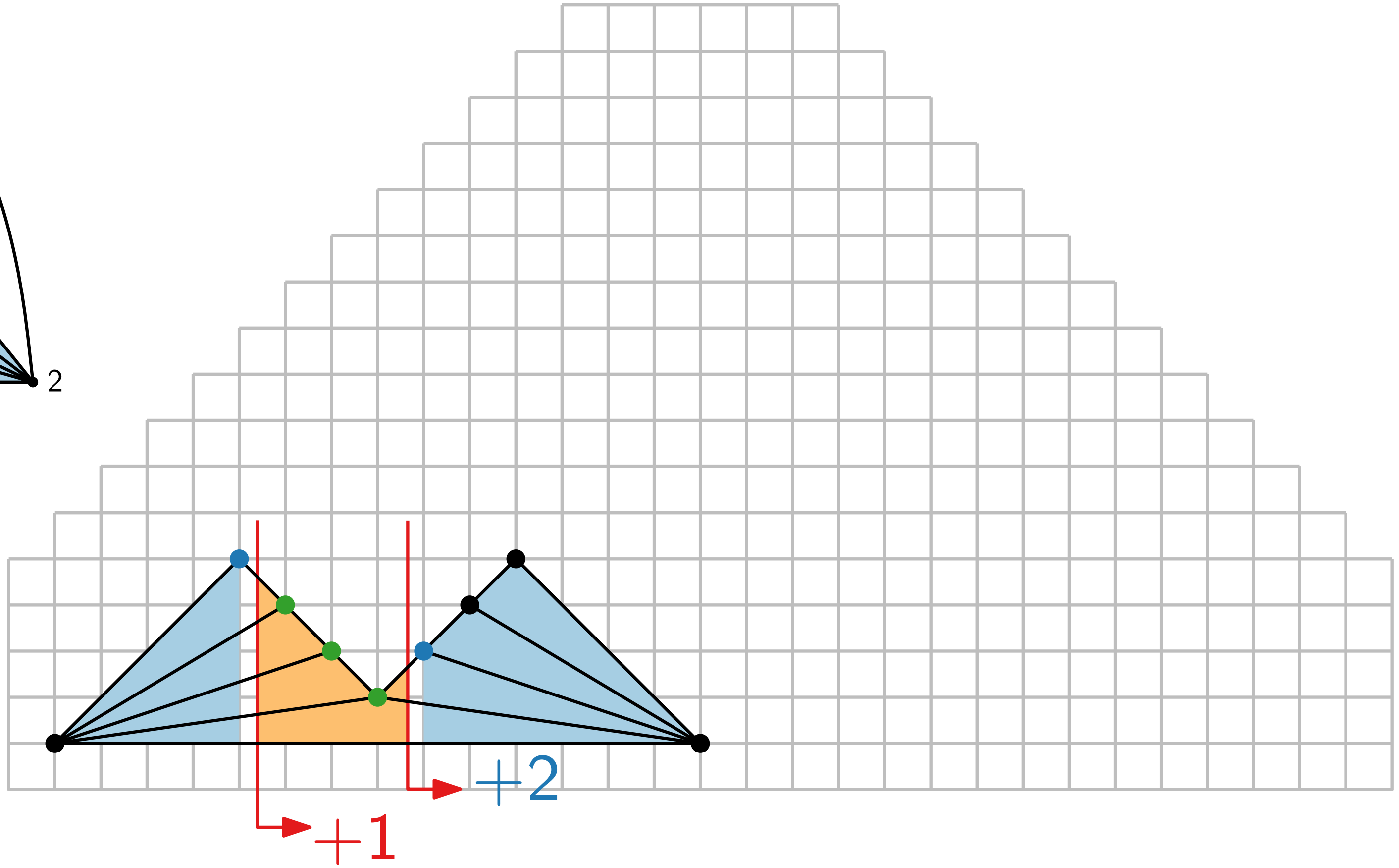
Observations.

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

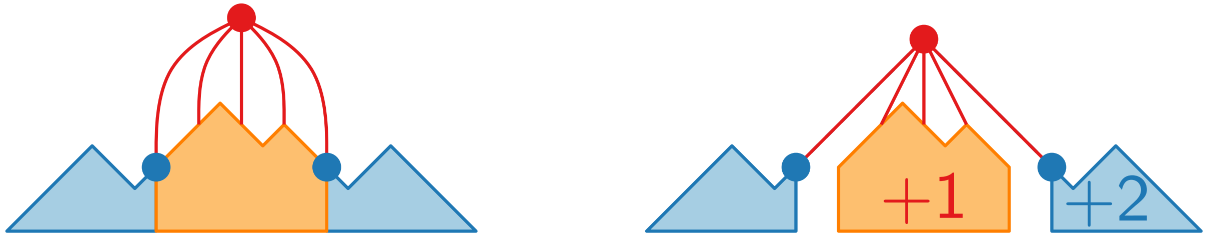
Shift method – example

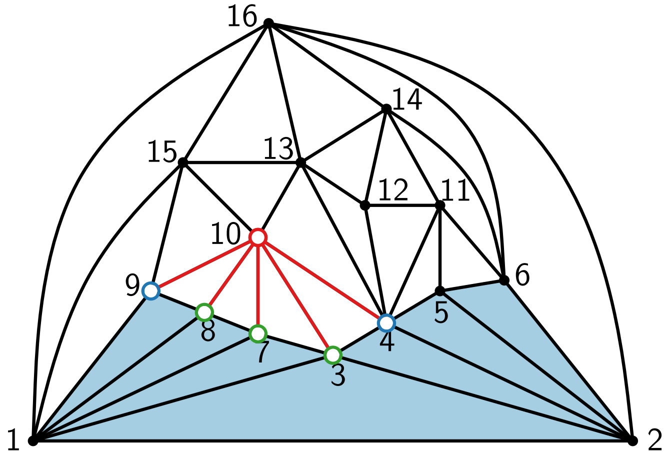


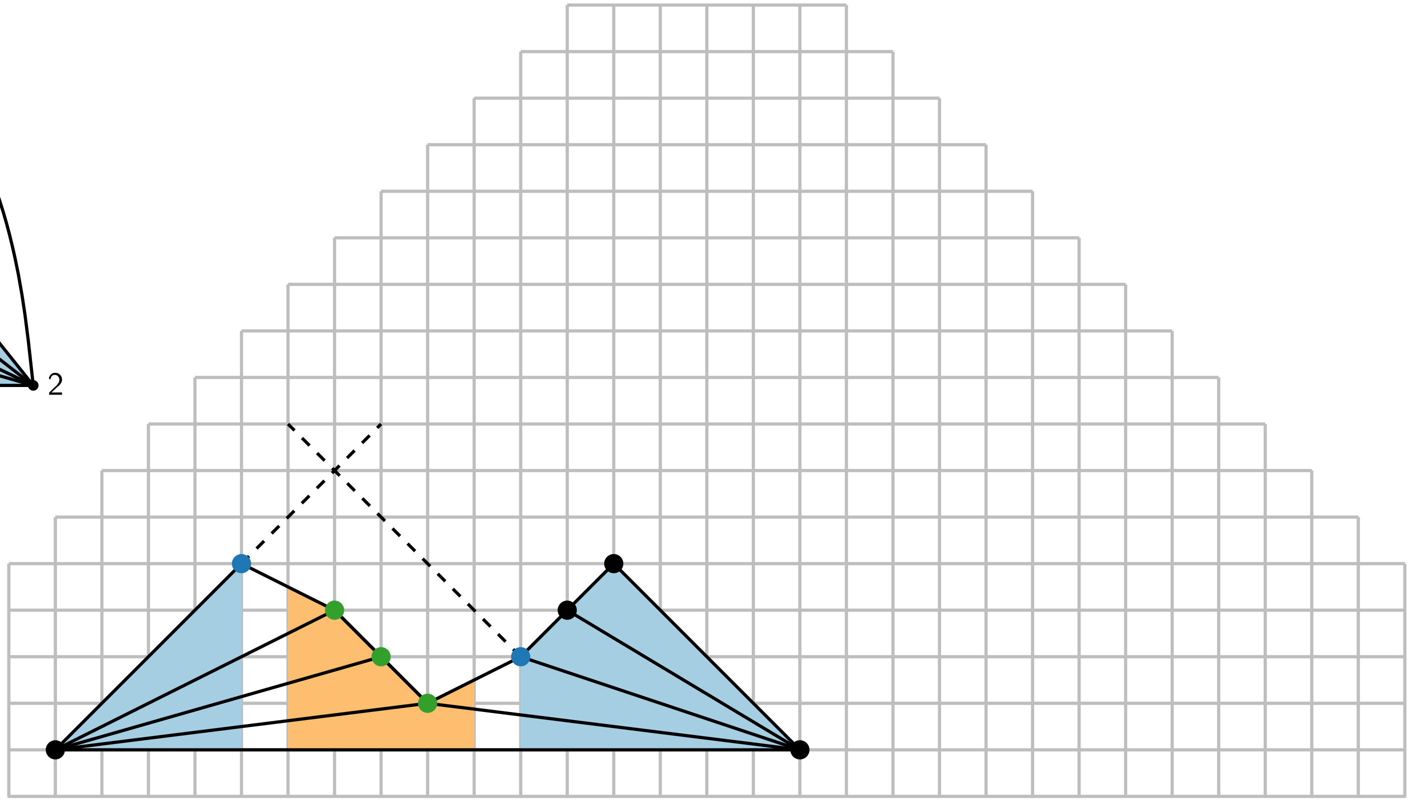




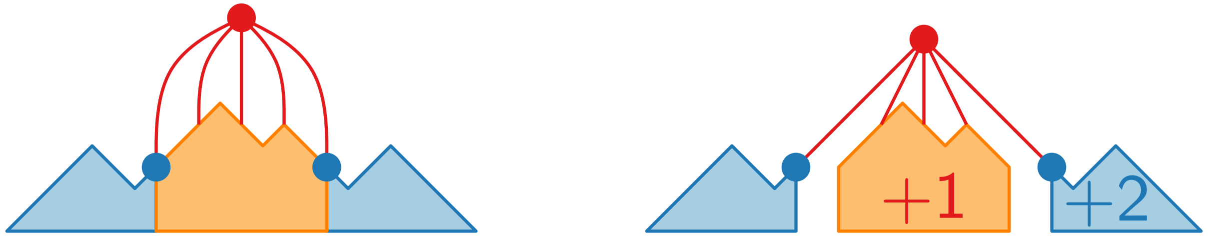
Shift method – example

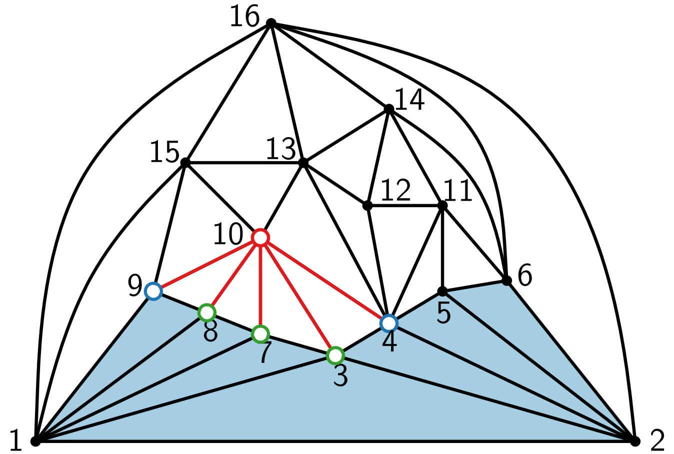


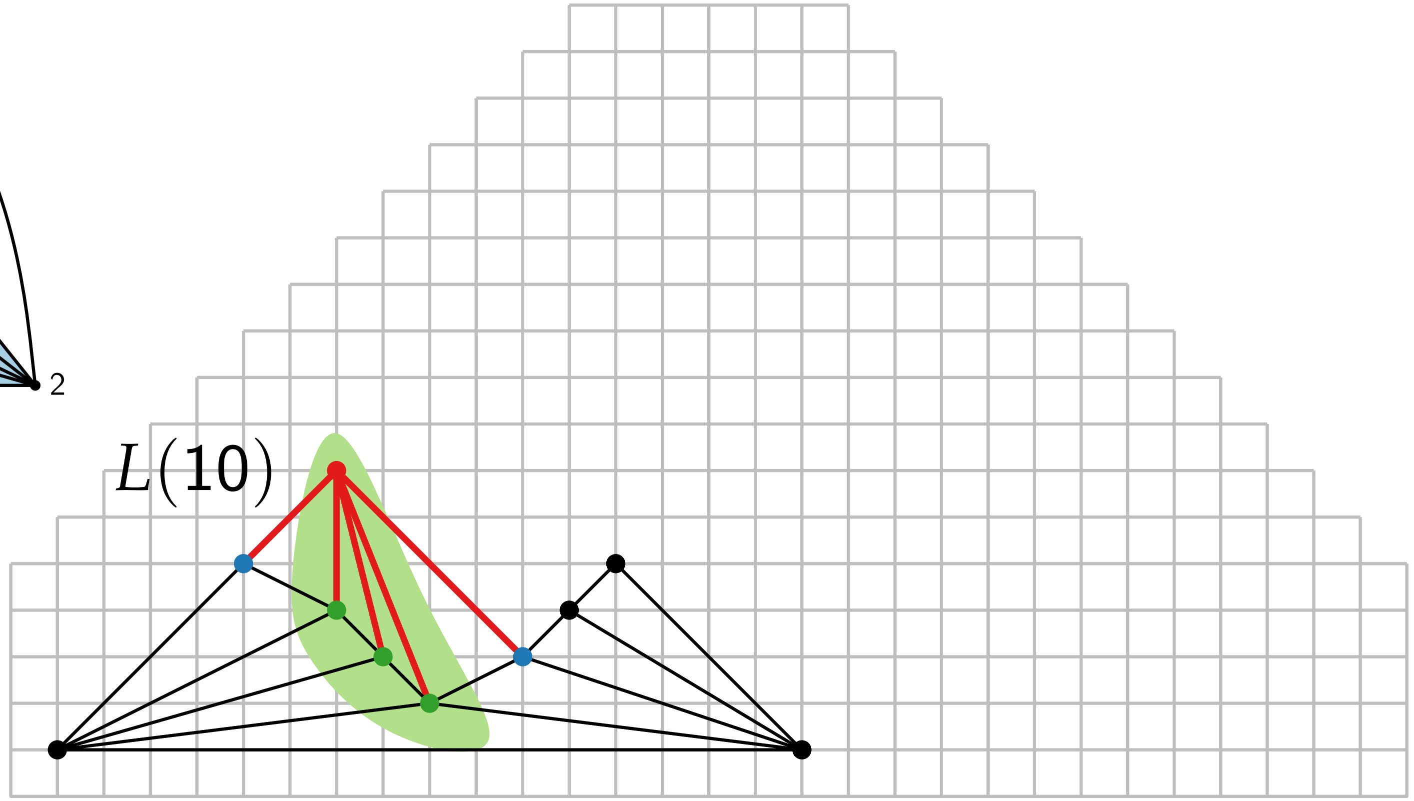




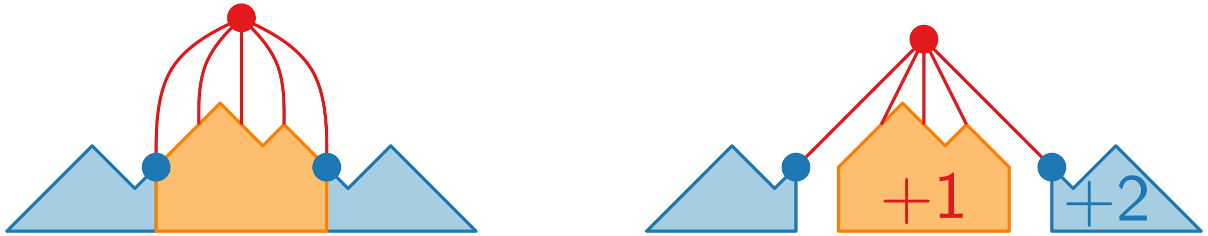
Shift method – example

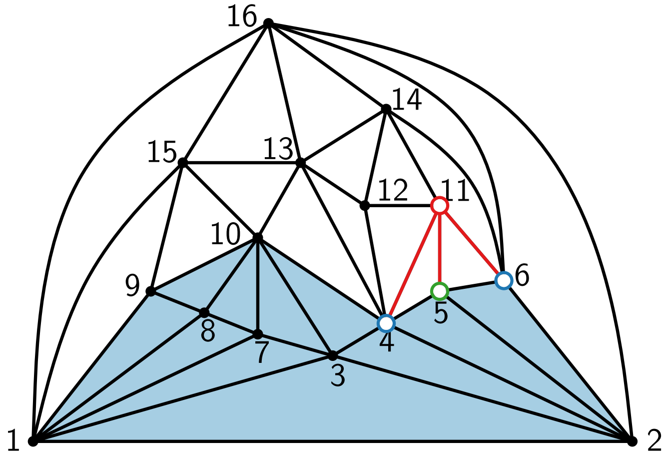


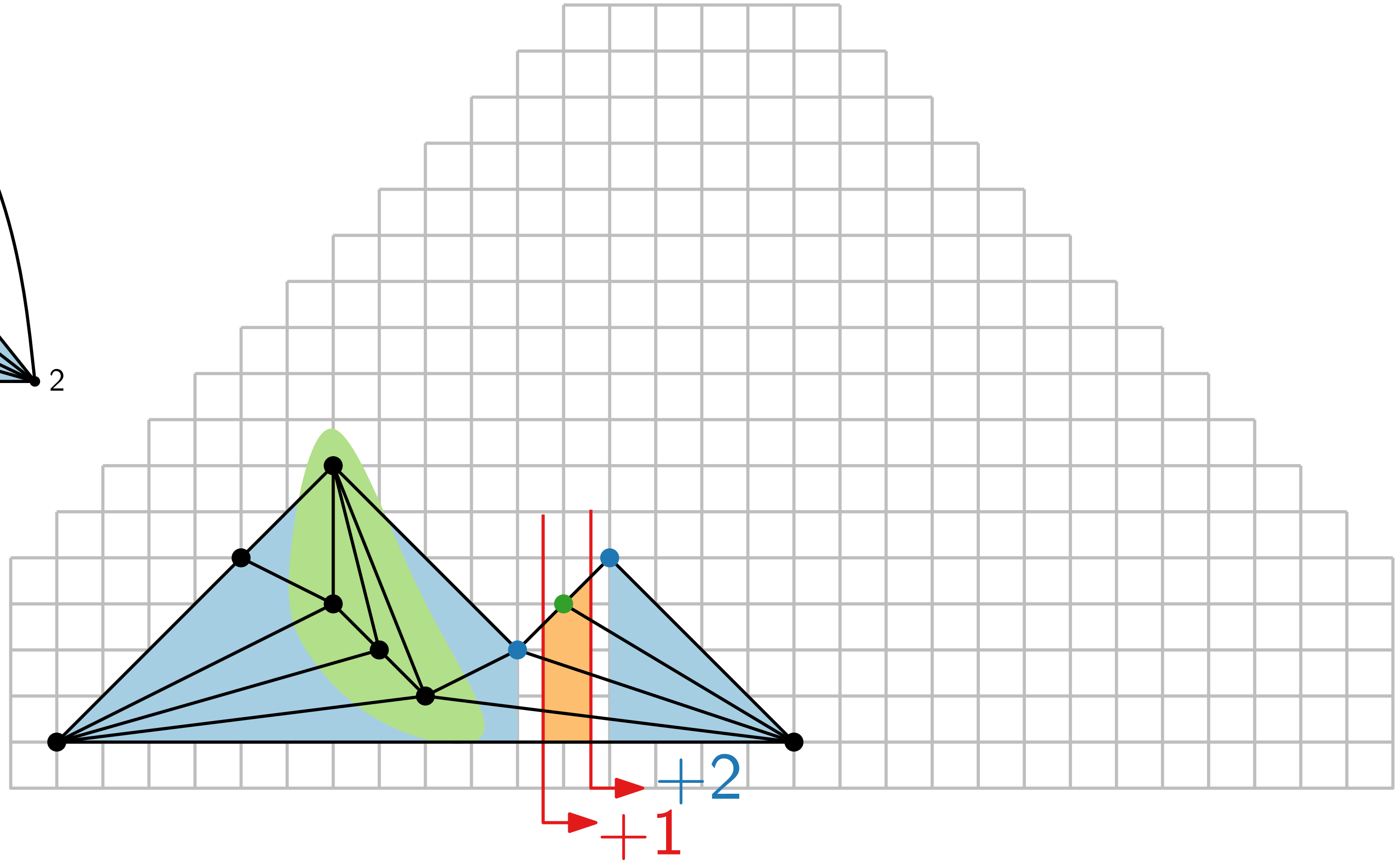




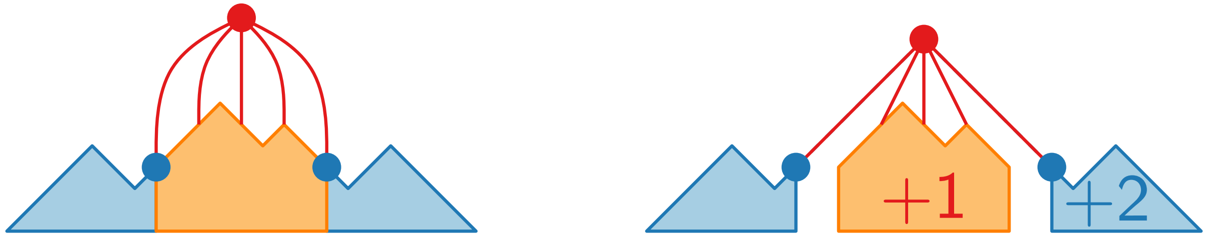
Shift method – example

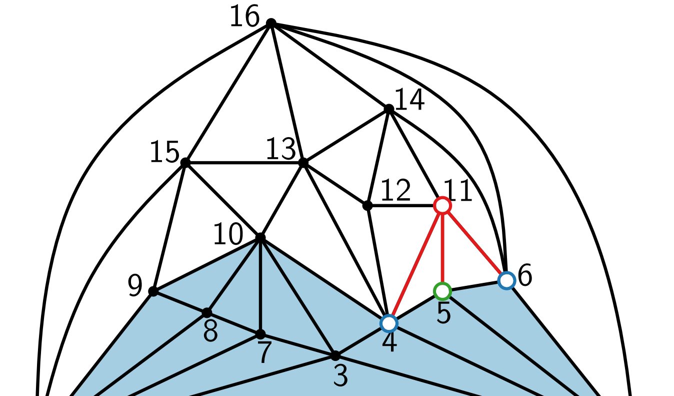


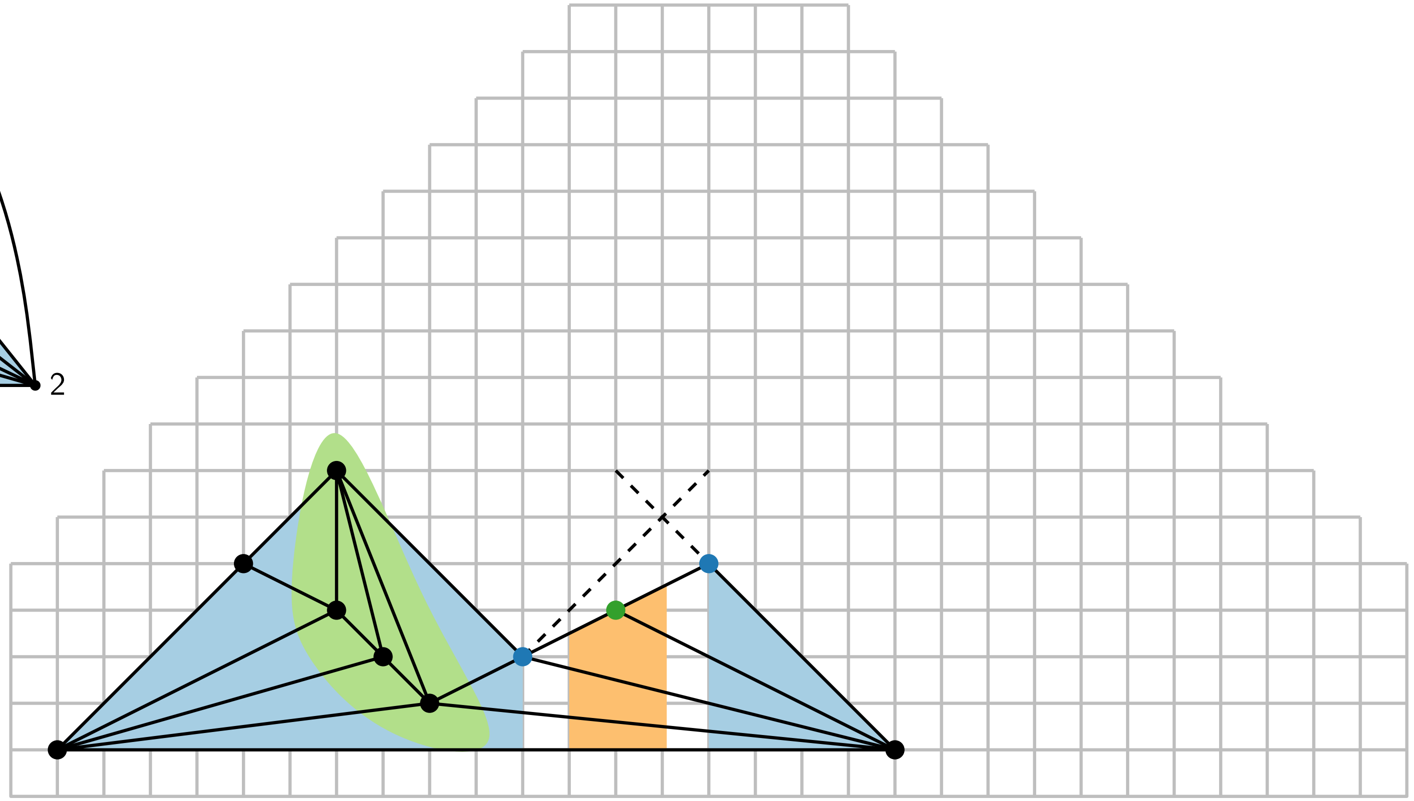




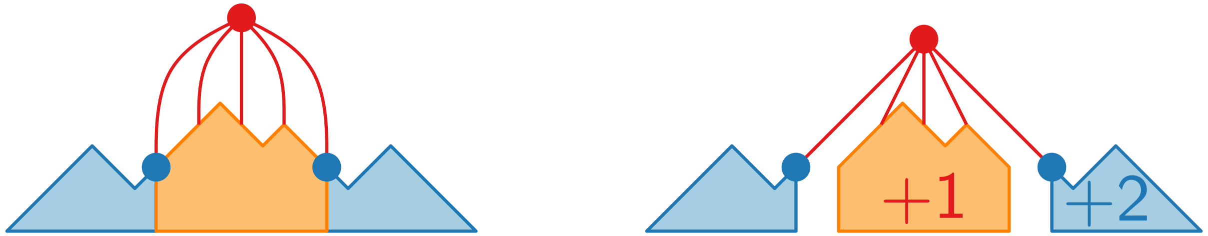
Shift method – example

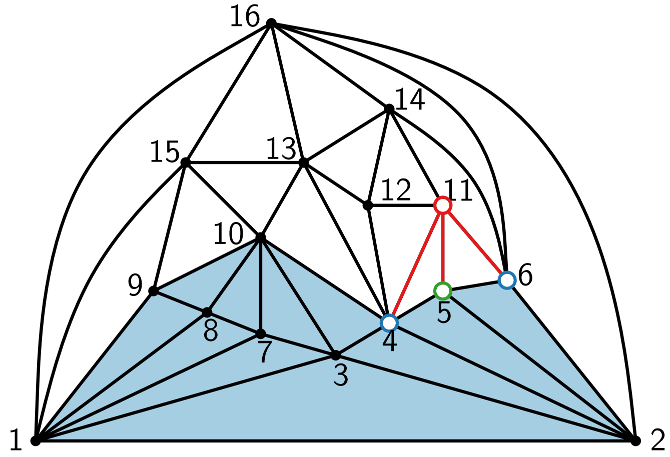


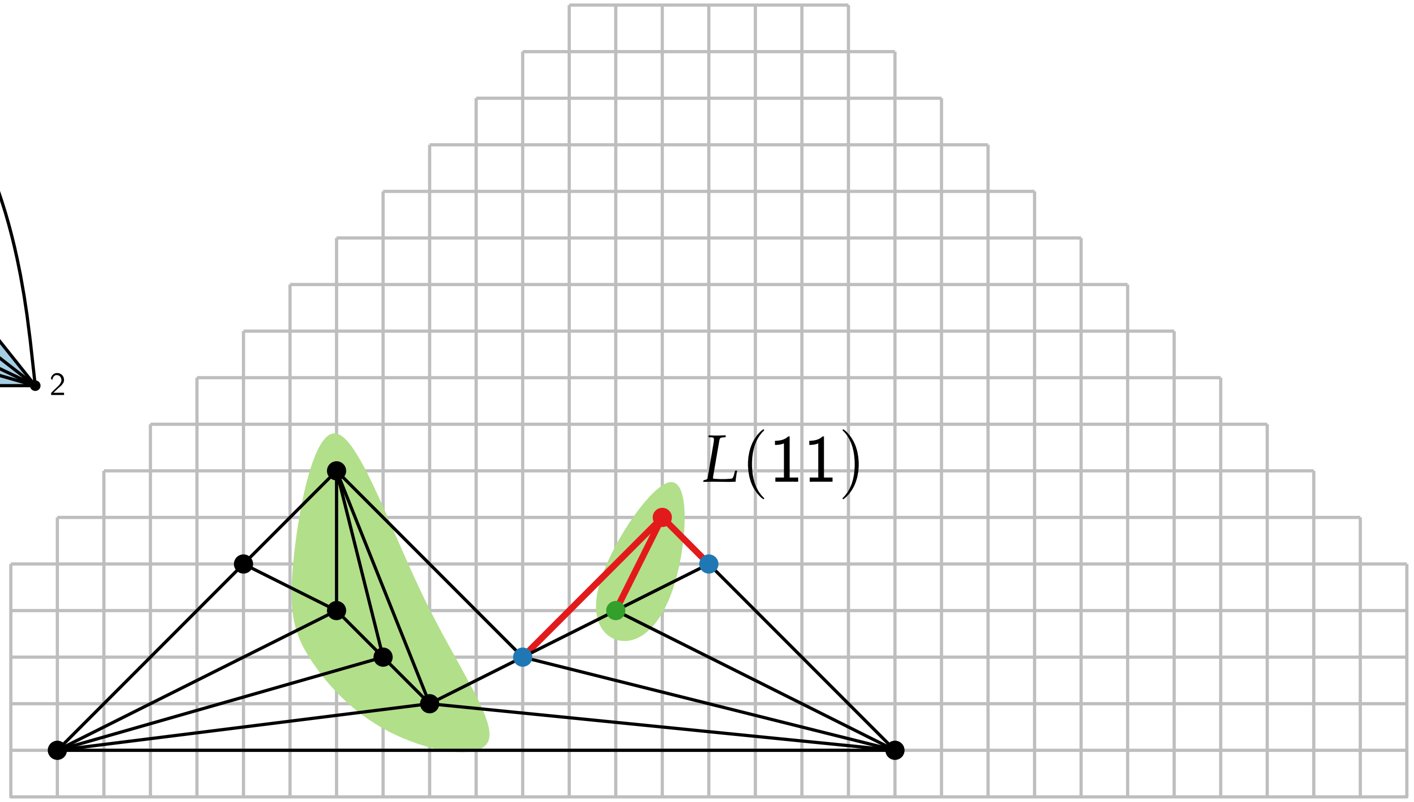




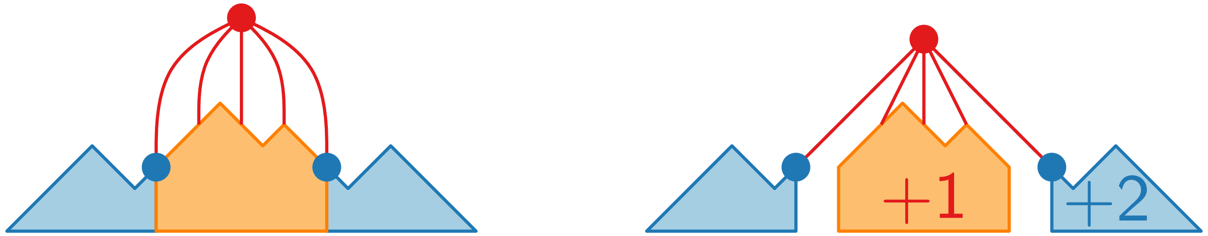
Shift method – example

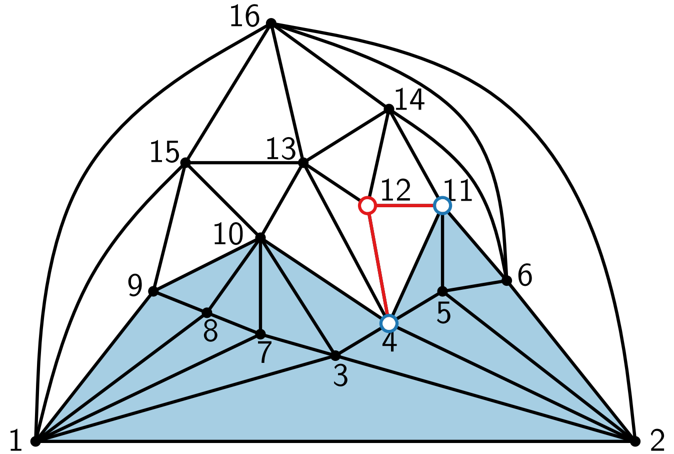


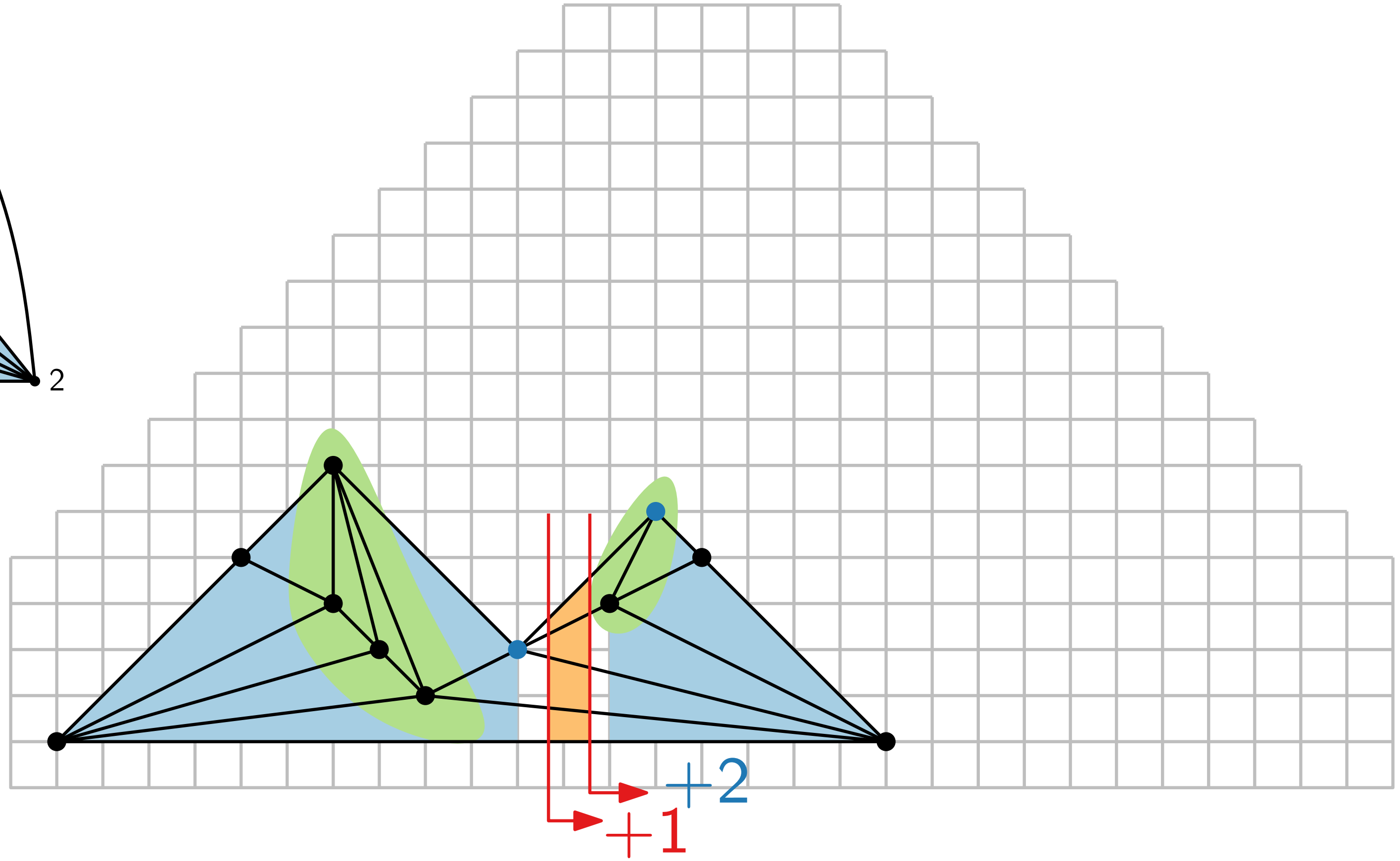




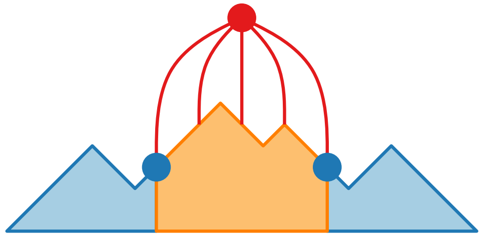
Shift method – example

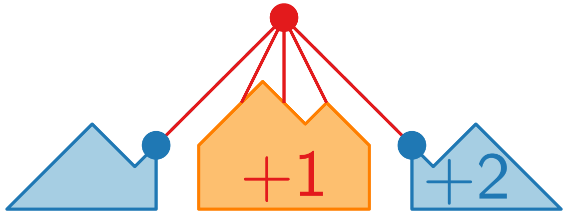


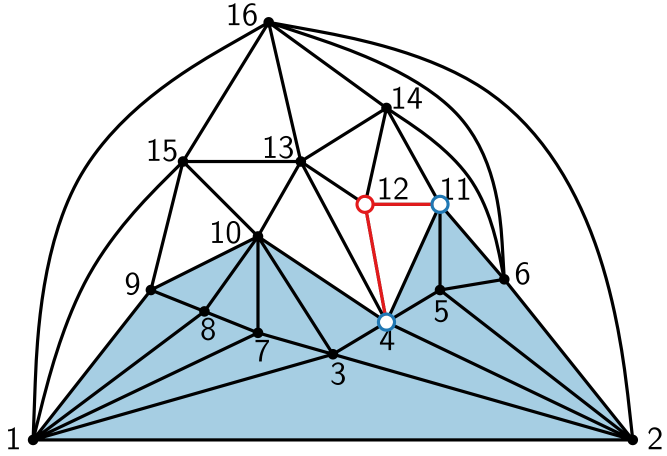


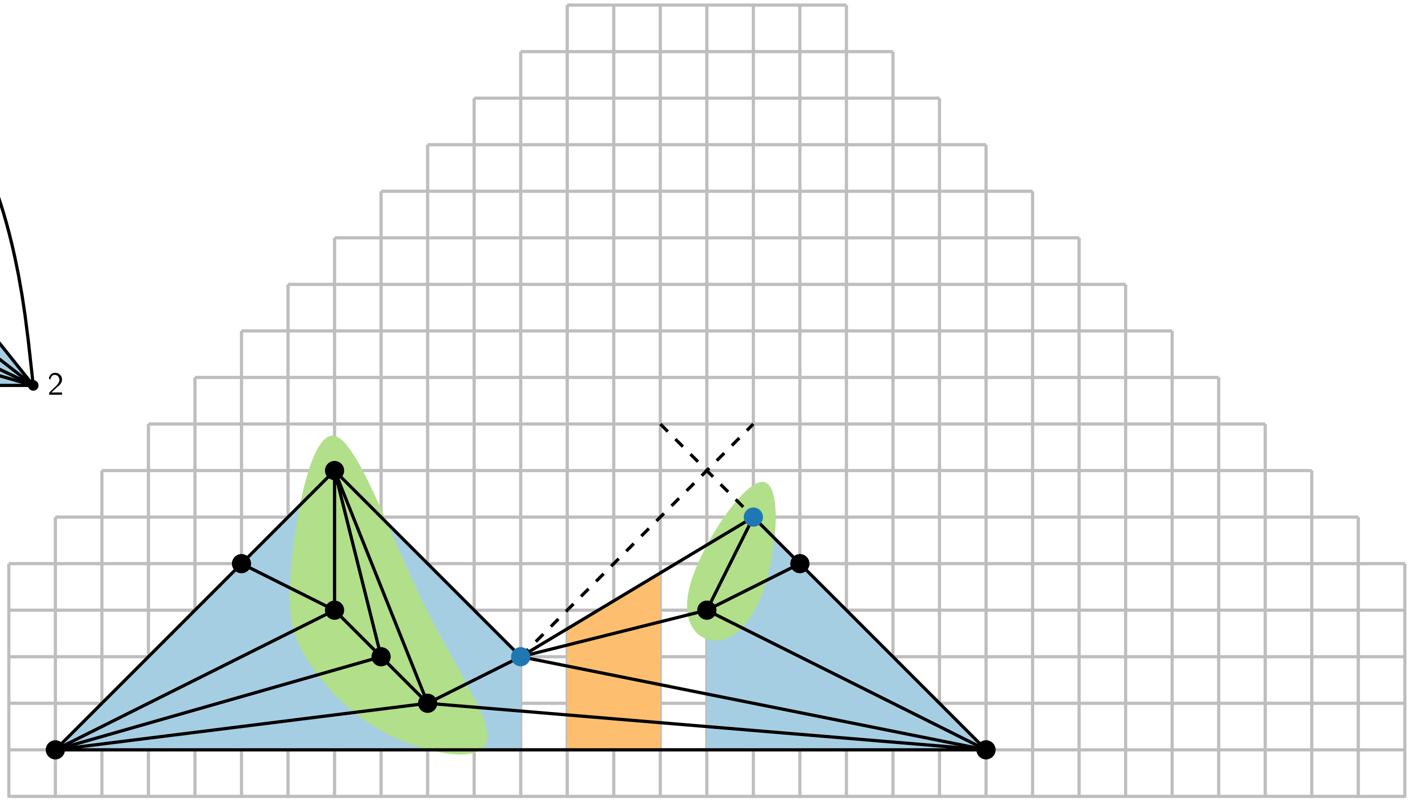


Shift method – example

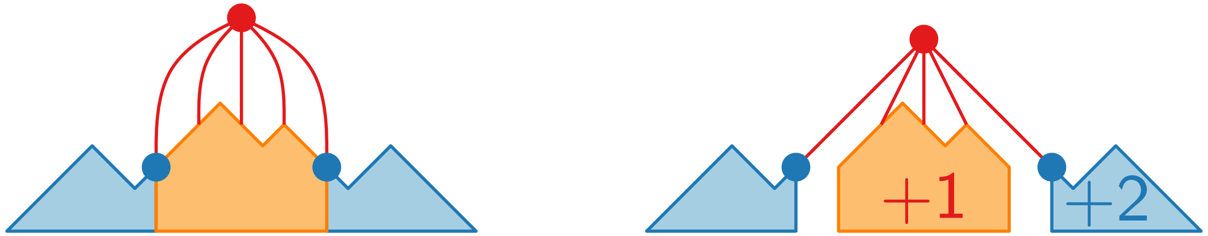


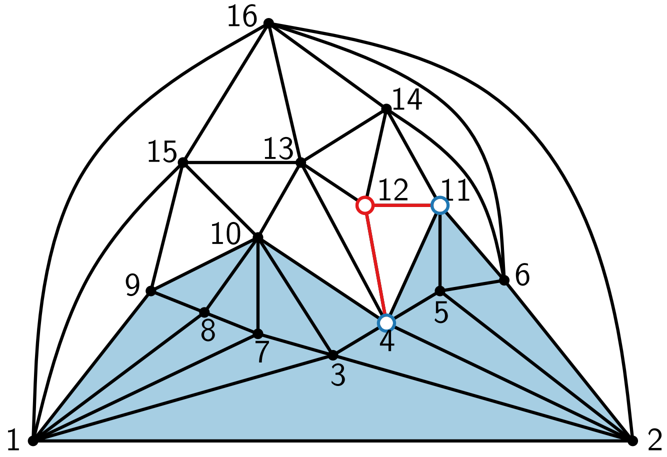


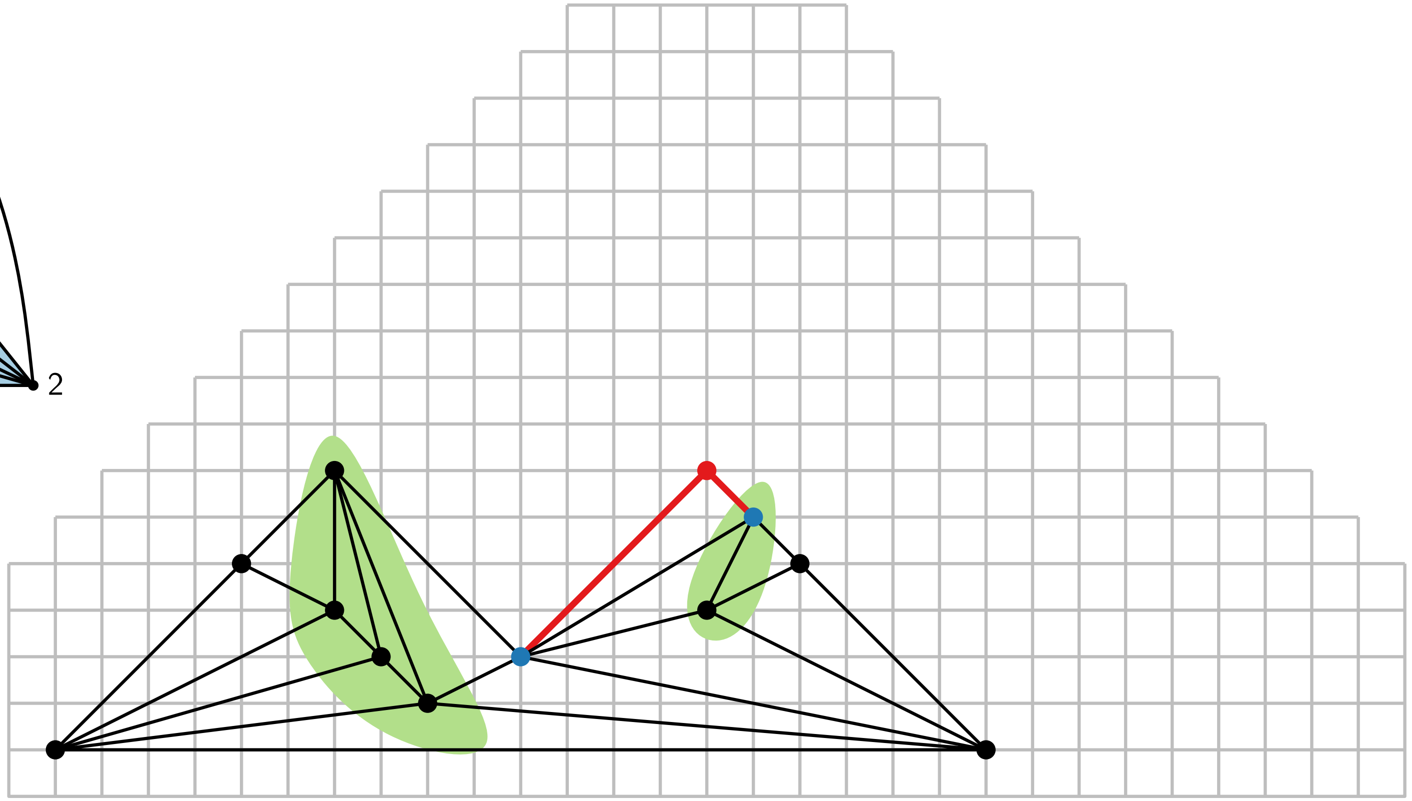




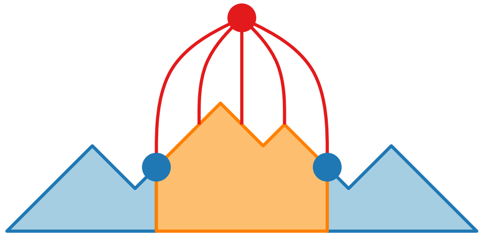
Shift method – example

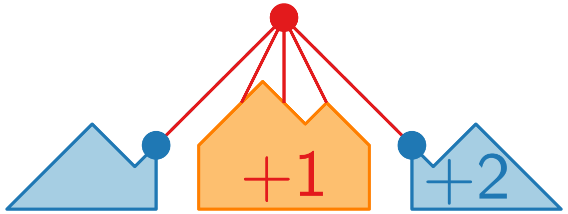


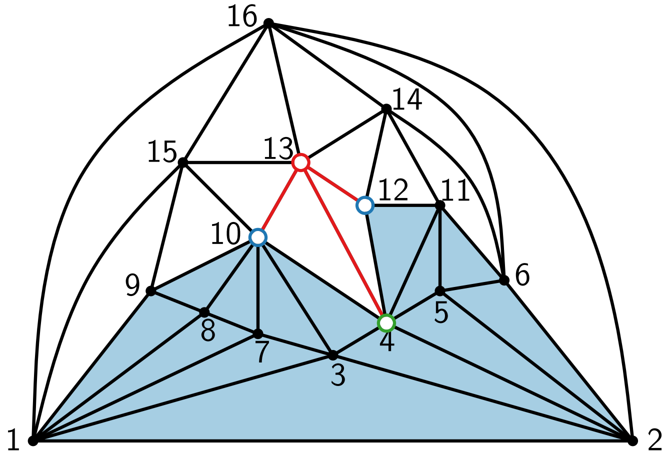


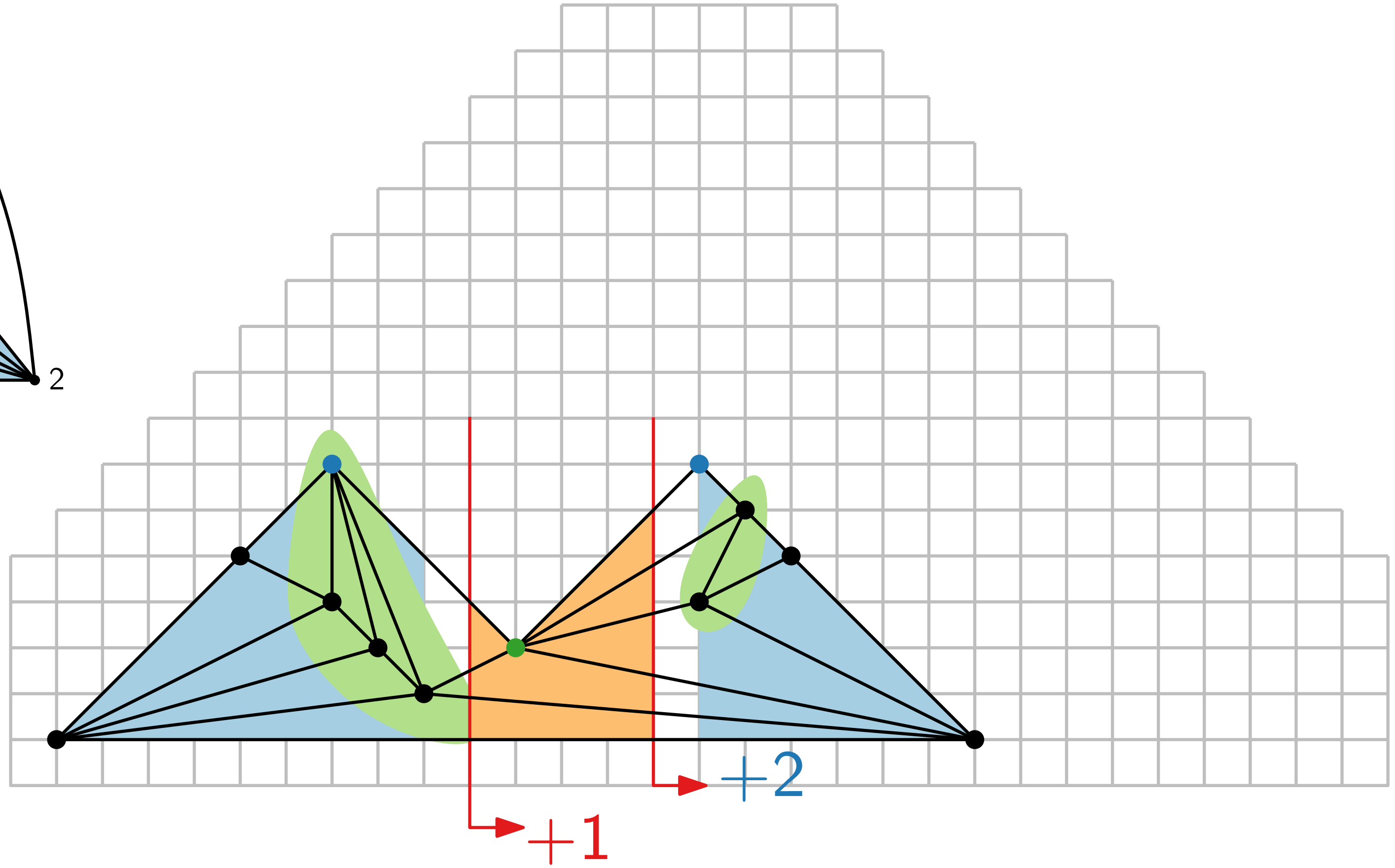


Shift method – example

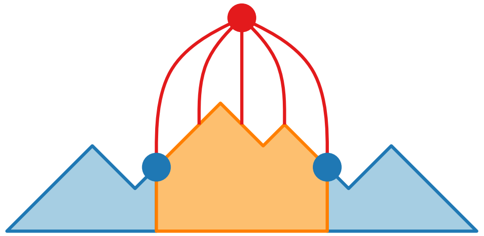


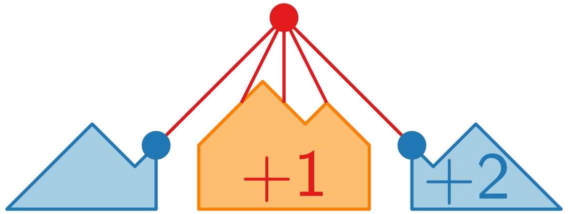


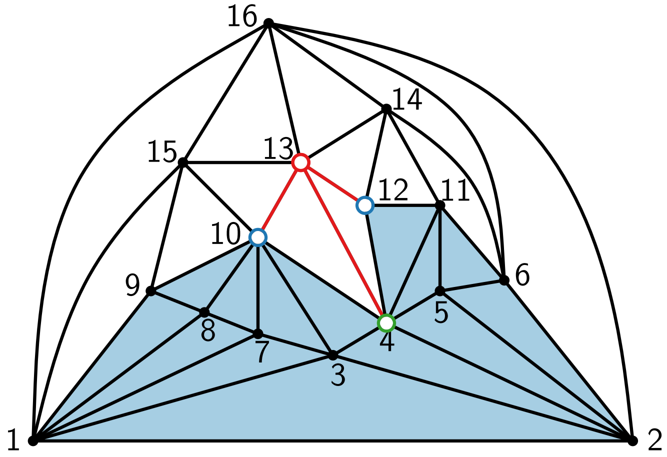


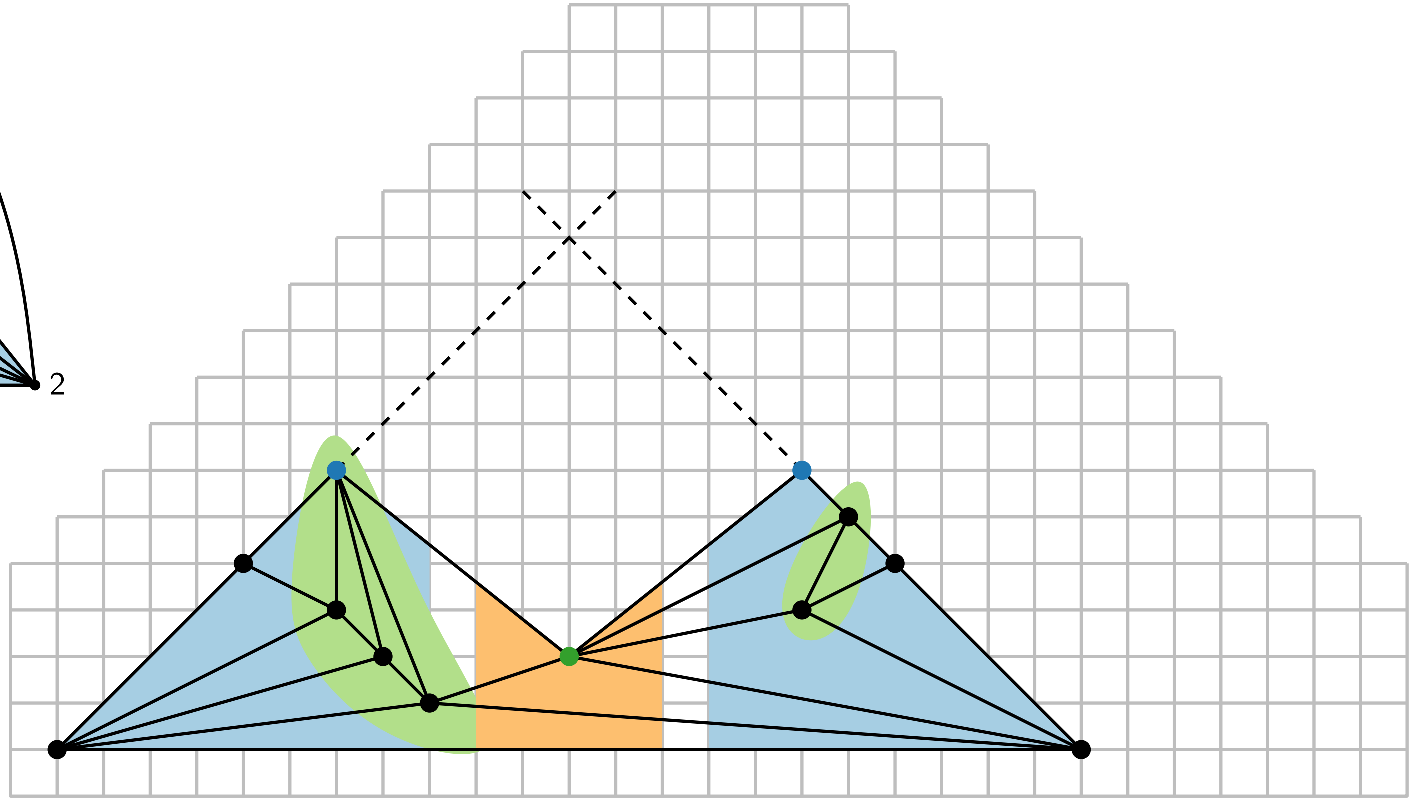


Shift method – example

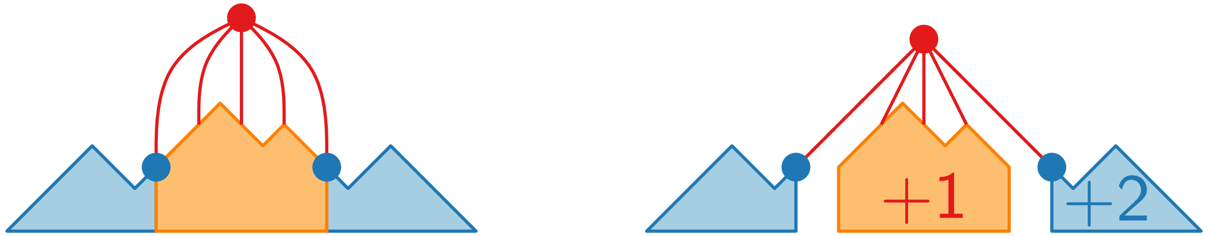


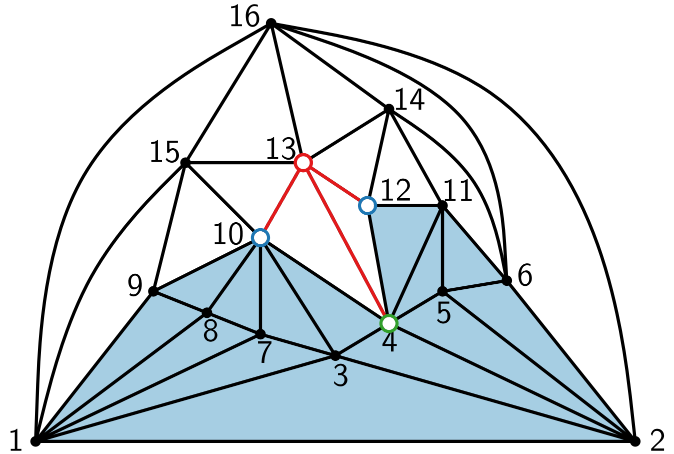


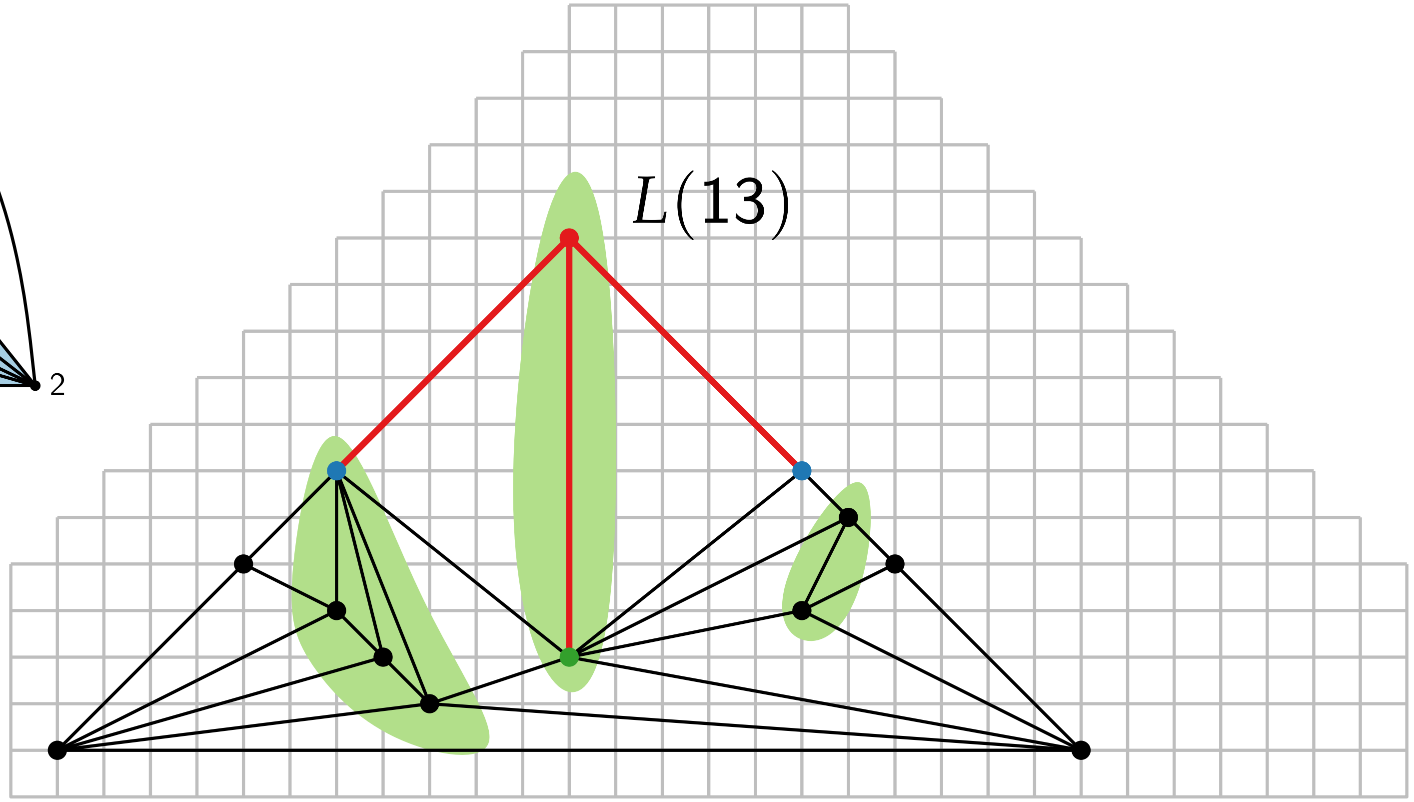




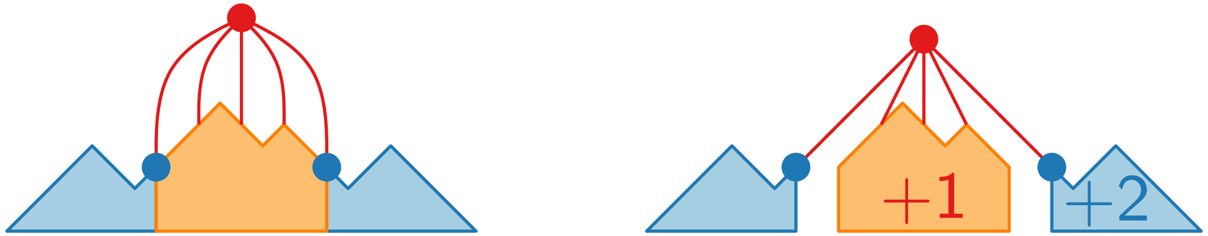
Shift method – example

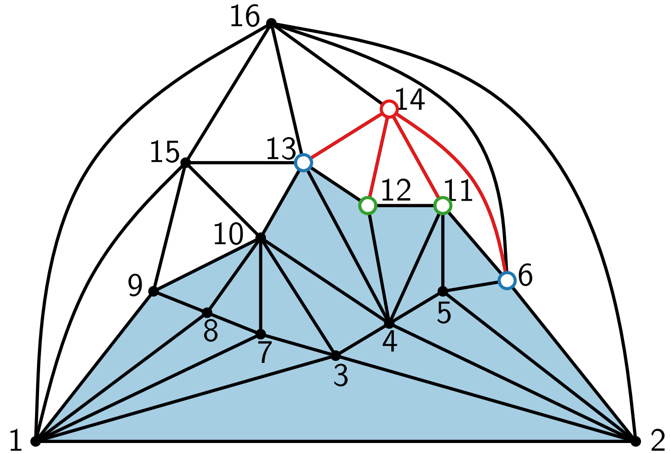


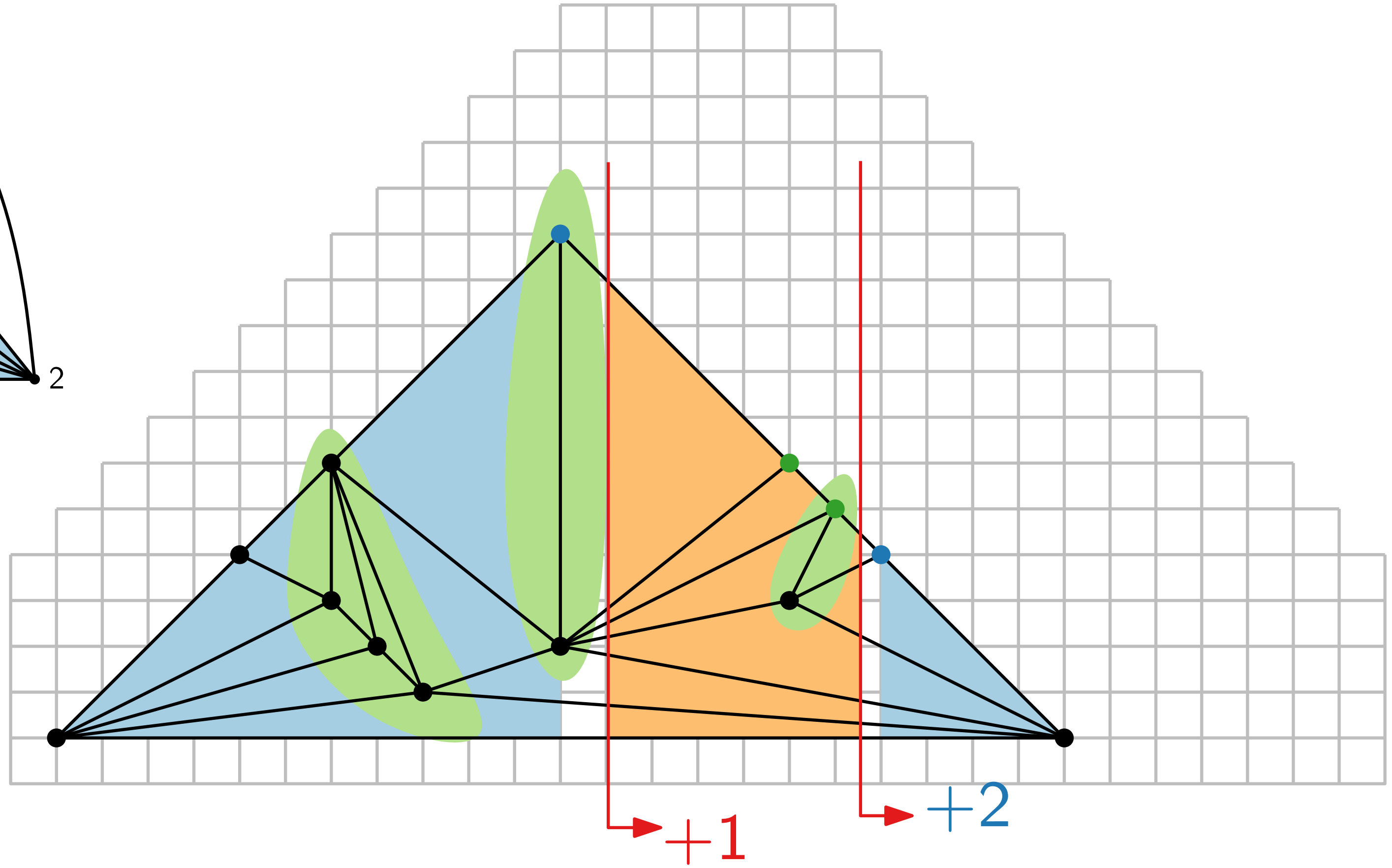




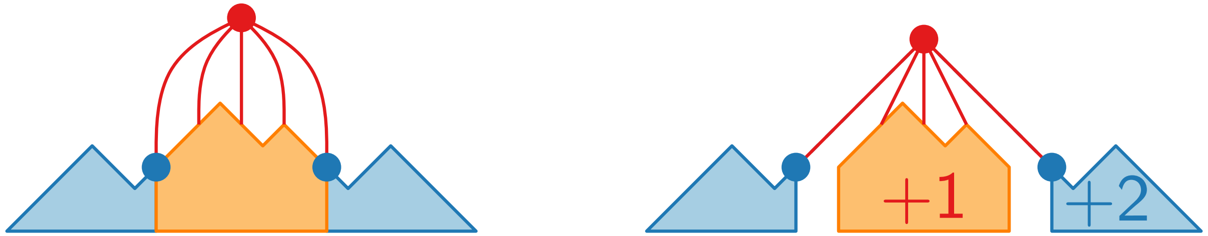
Shift method – example

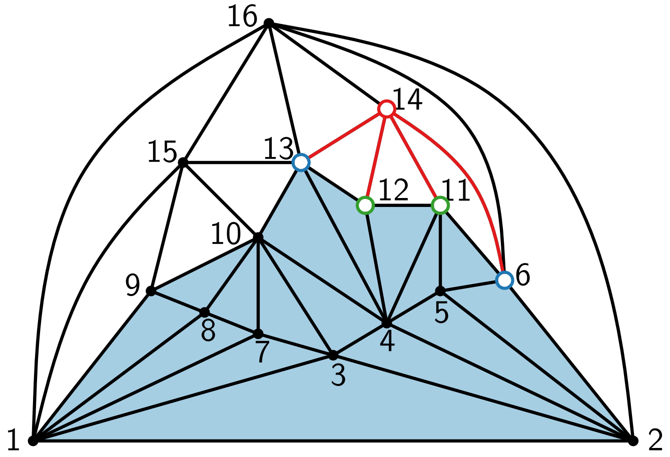


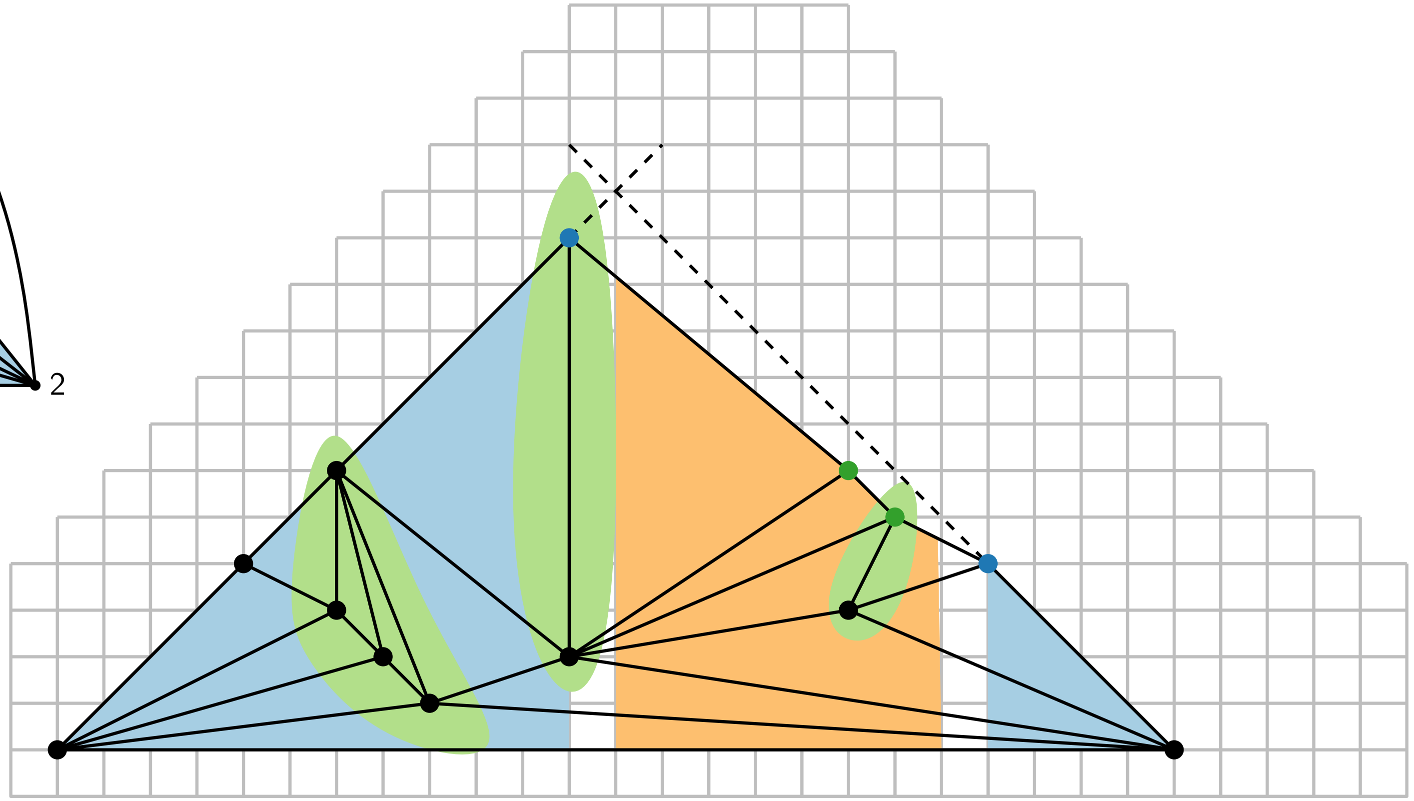




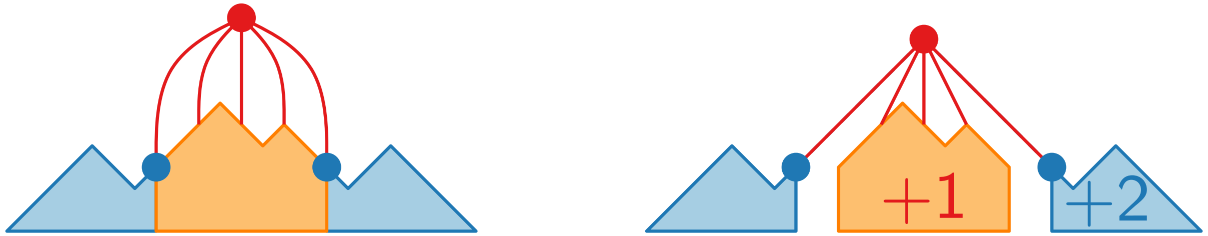
Shift method – example

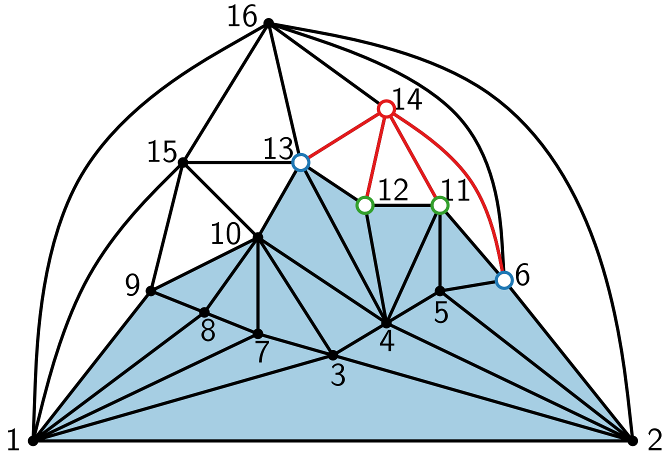


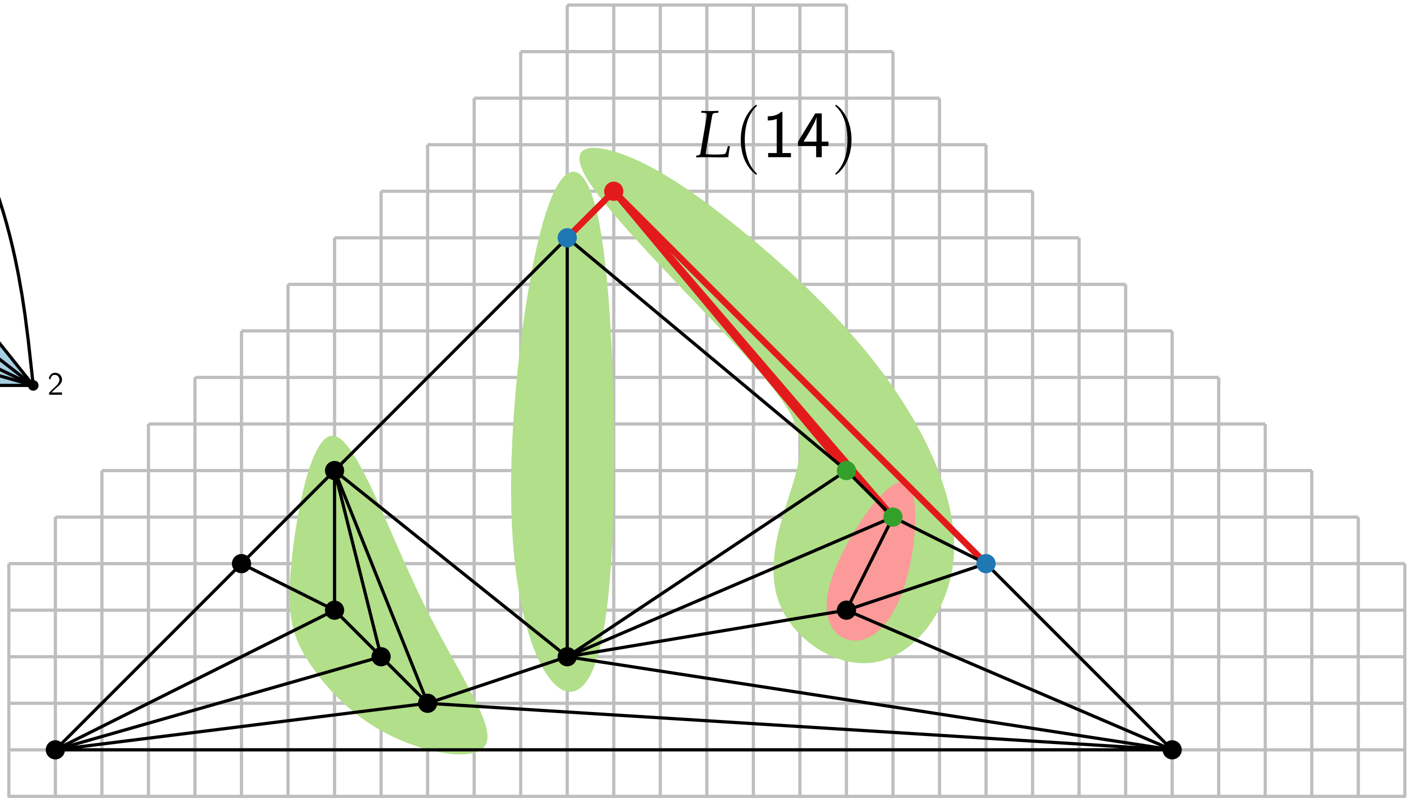




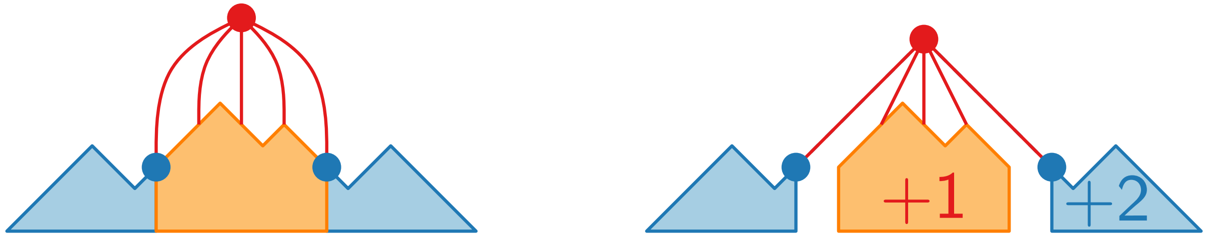
Shift method – example

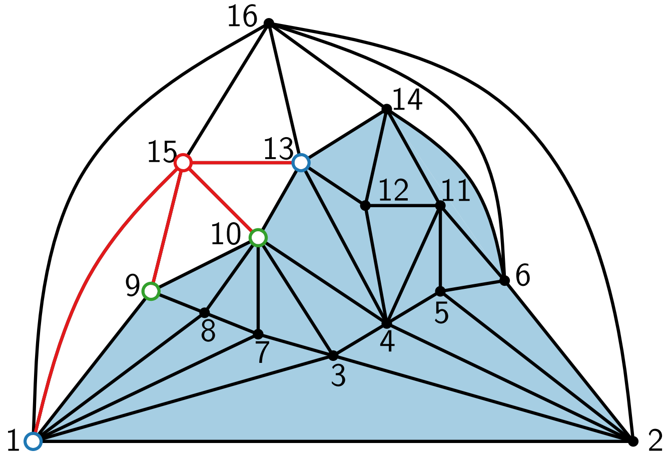


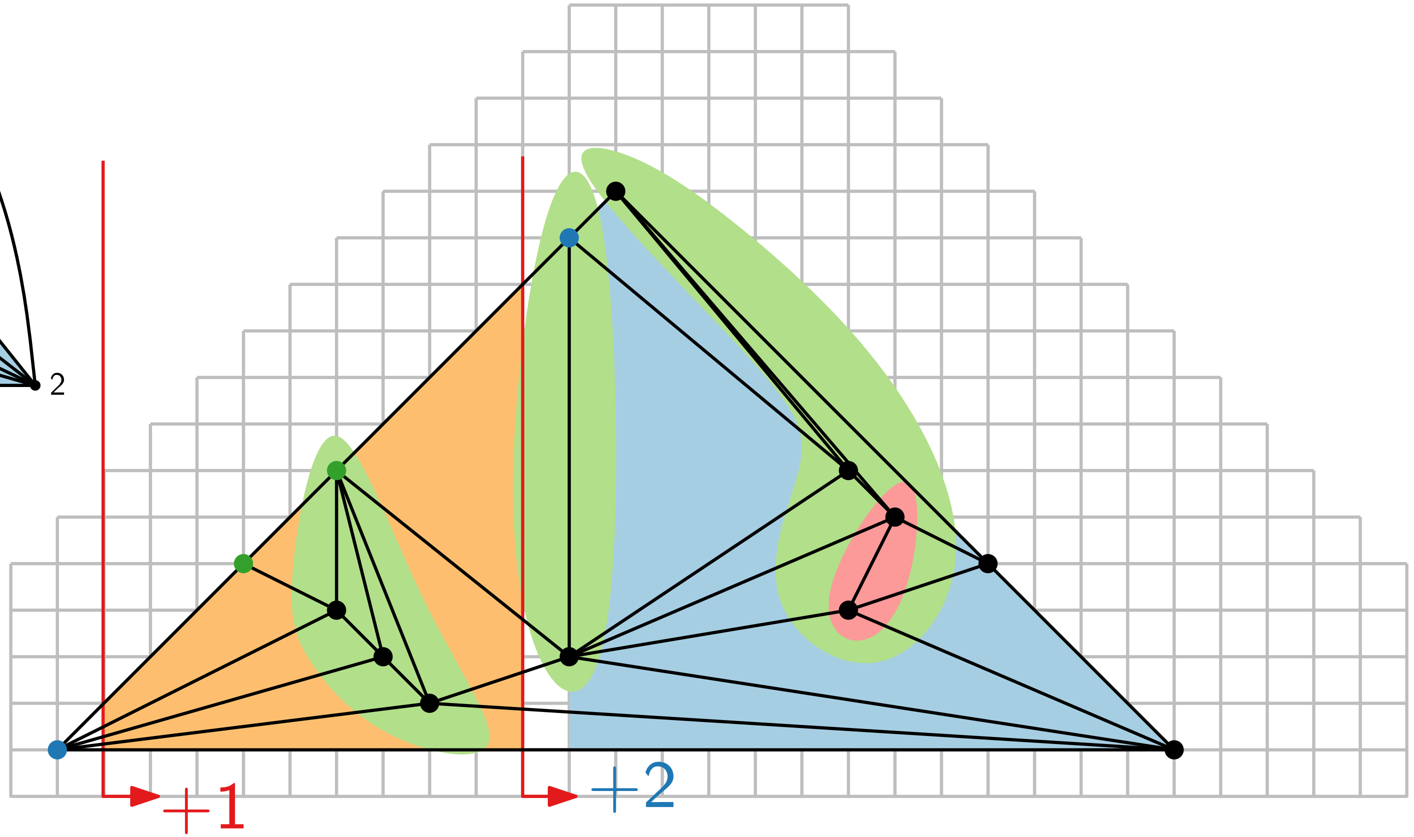




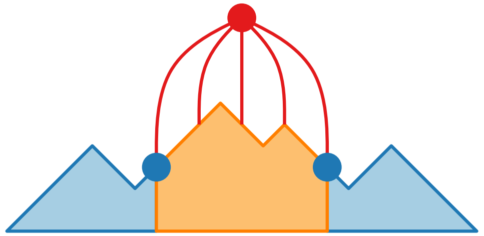
Shift method – example

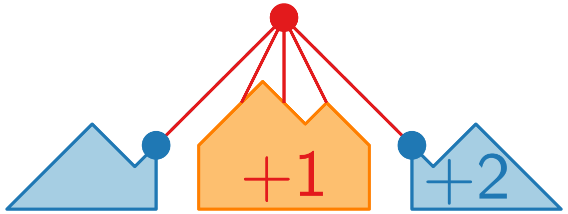


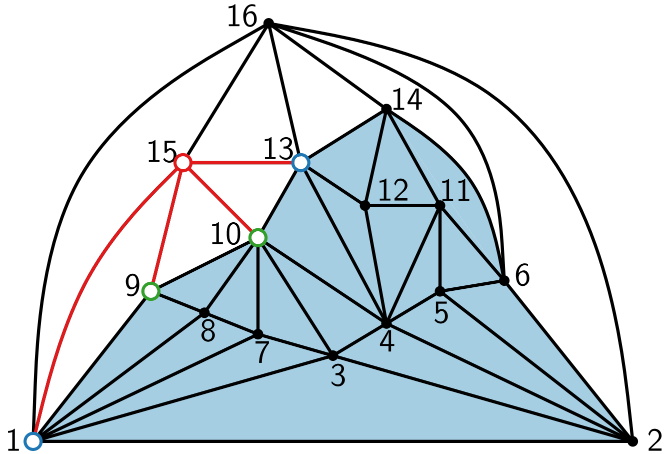


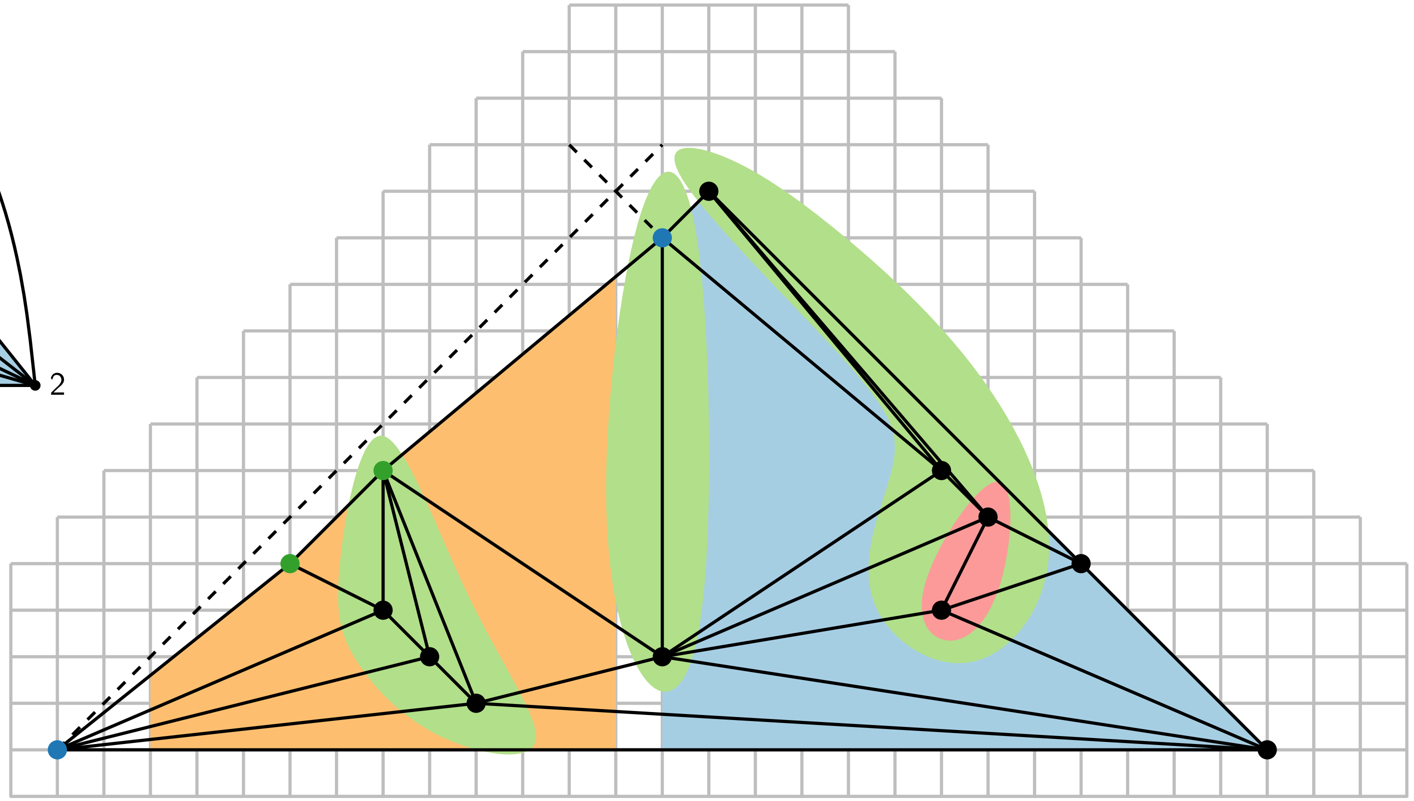


Shift method – example

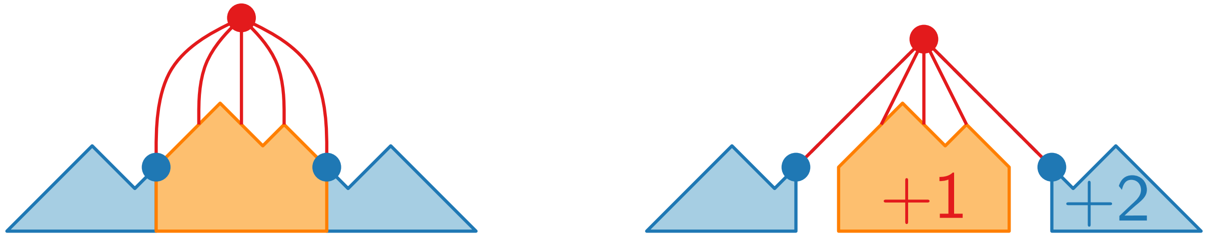


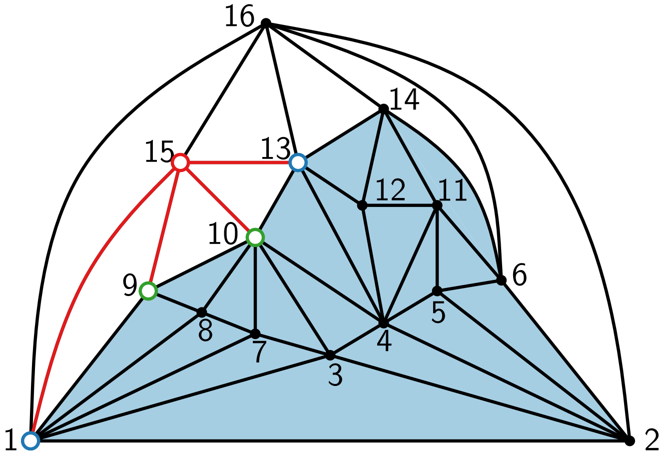


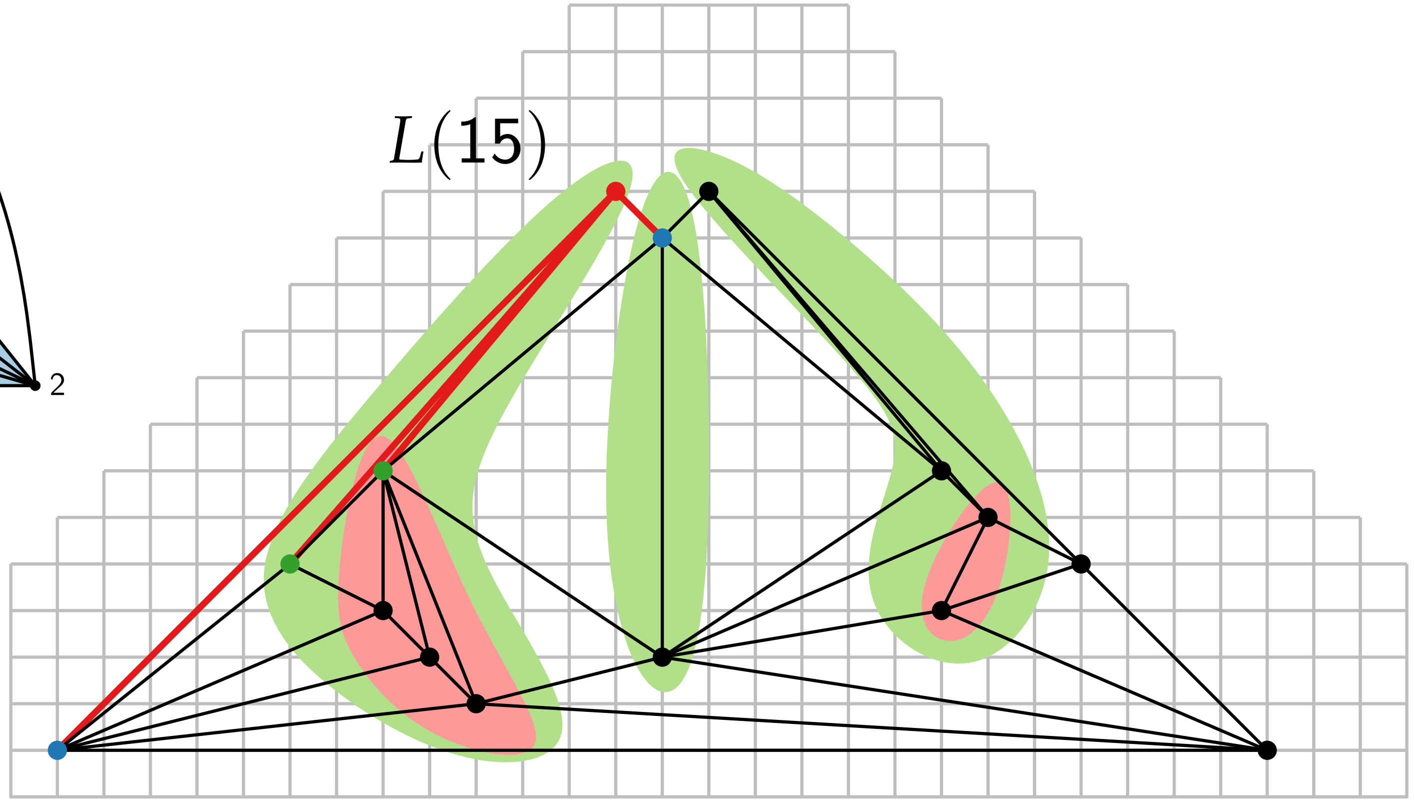




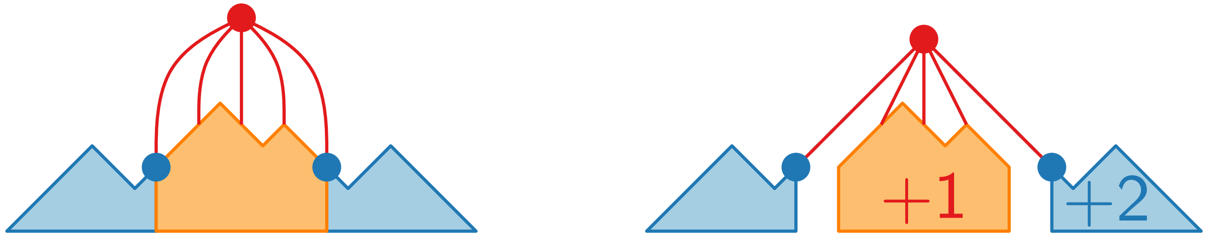
Shift method – example

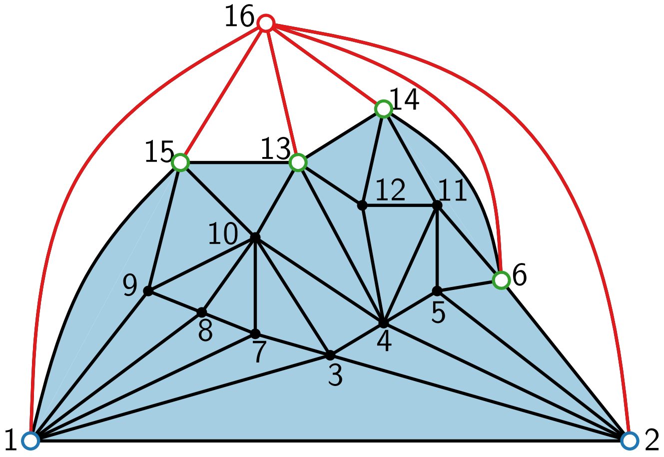


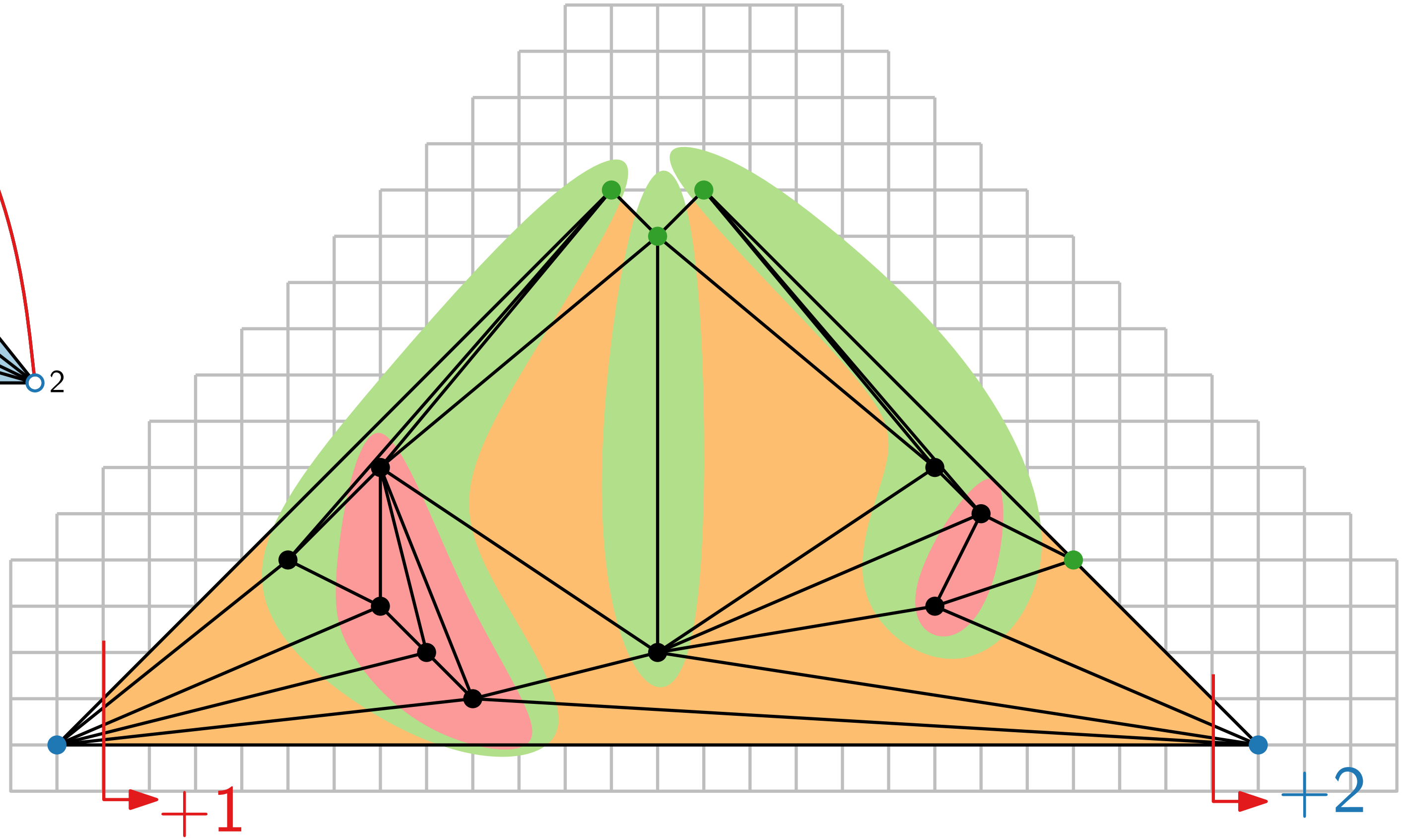




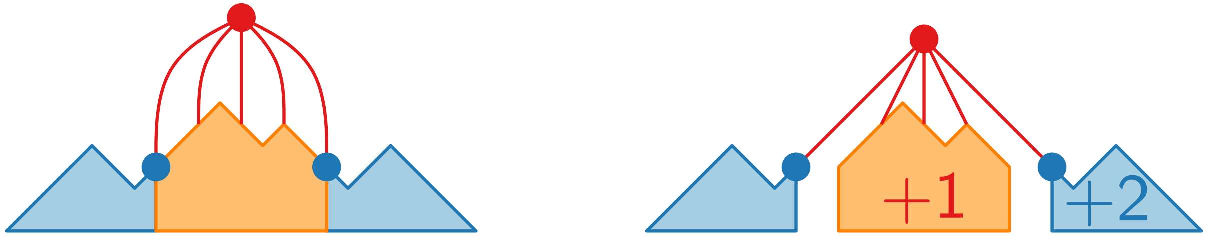
Shift method – example

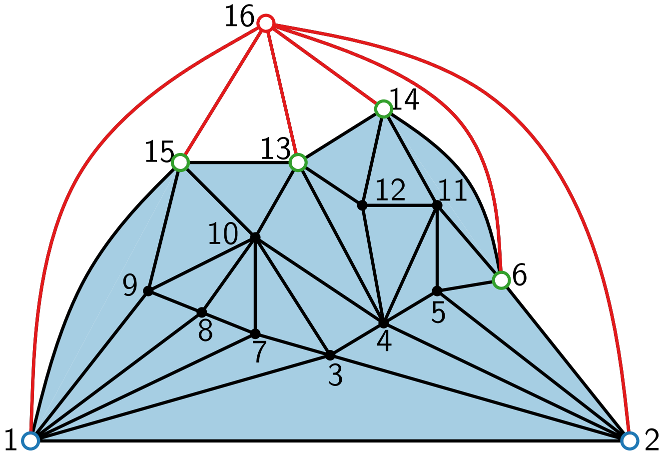


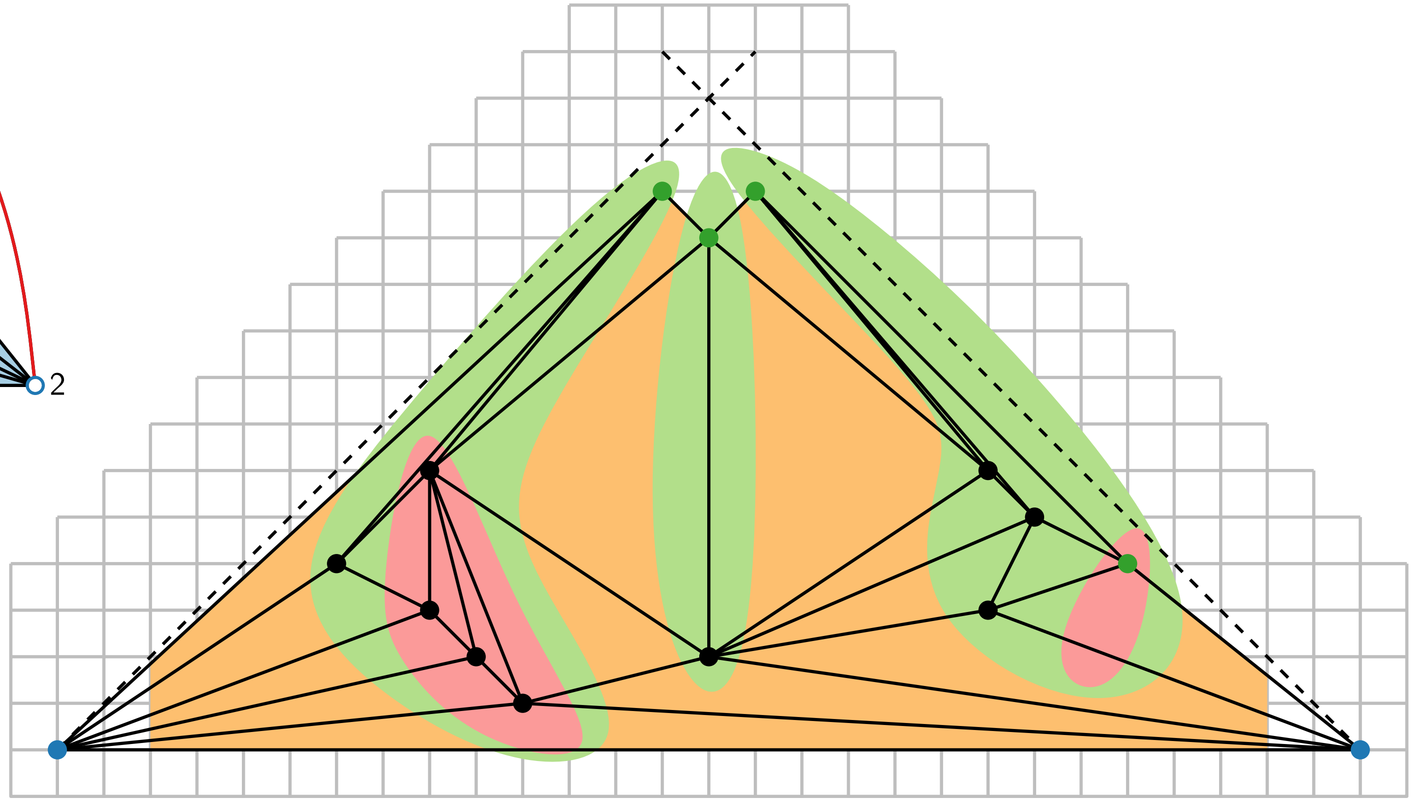




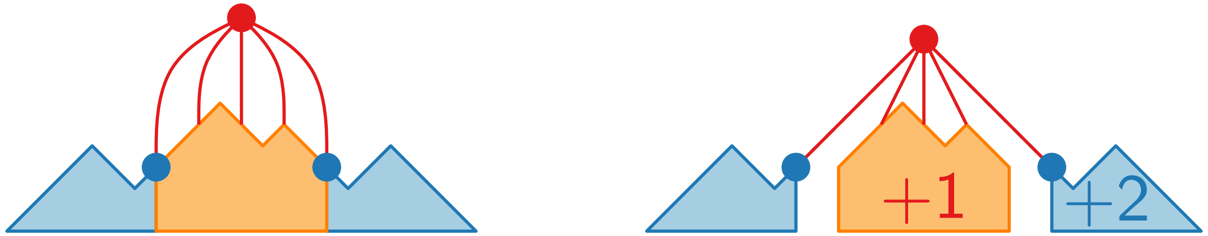
Shift method – example

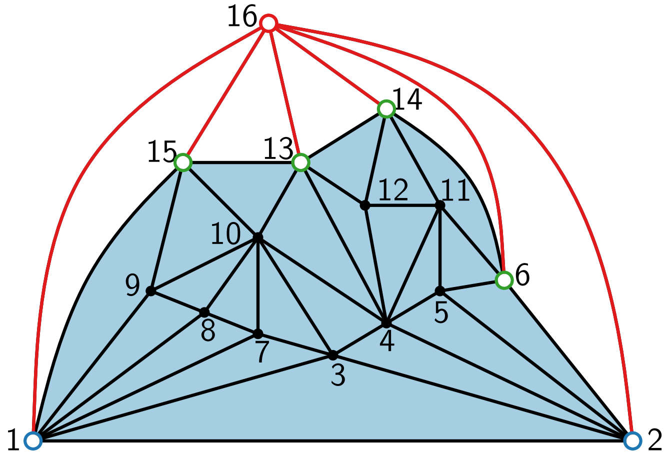


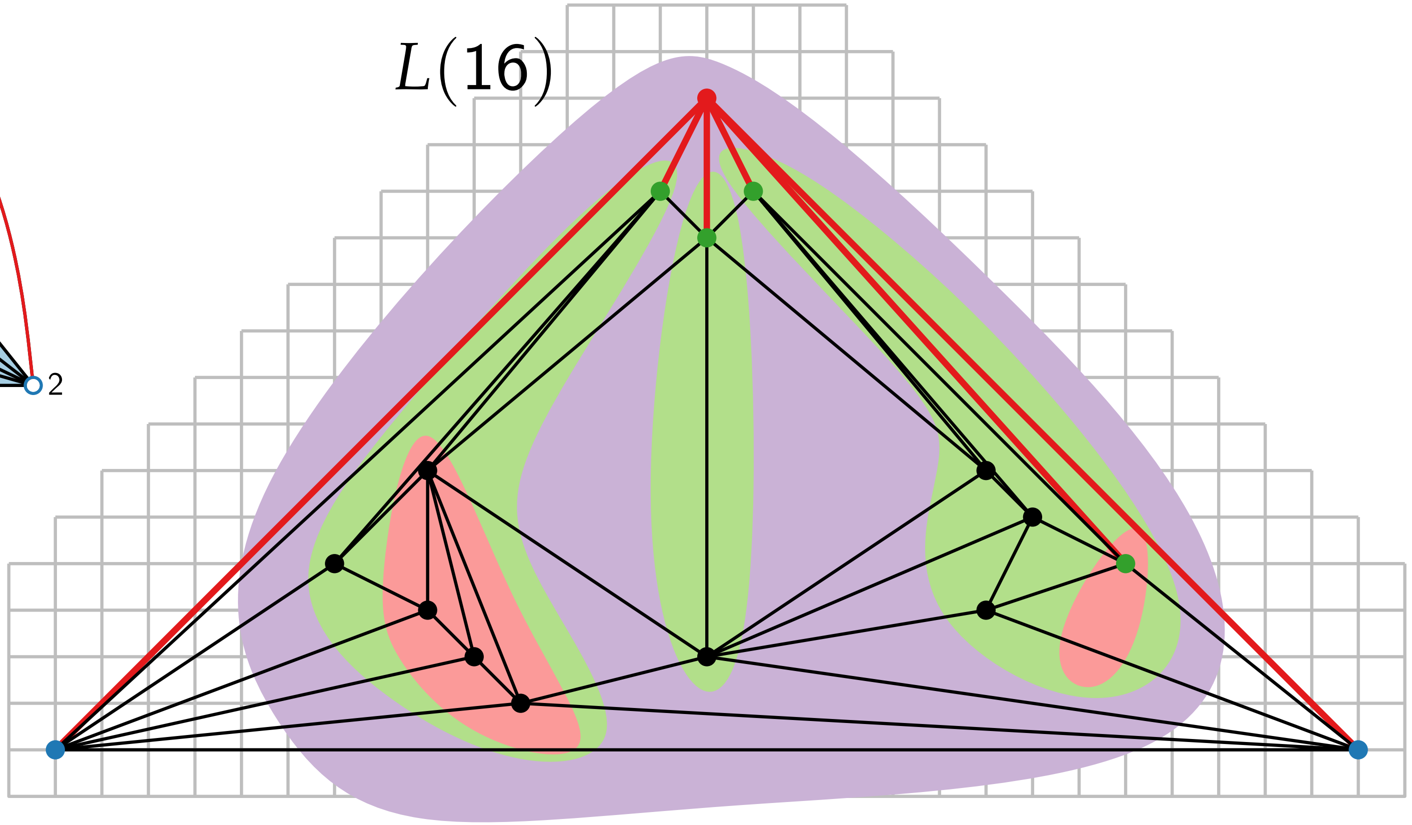




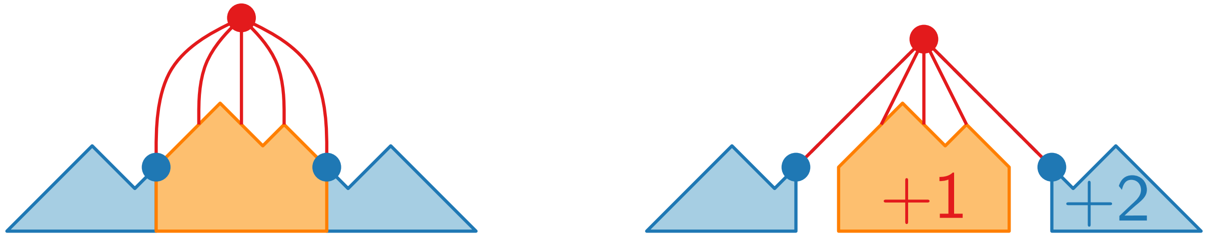
Shift method – example

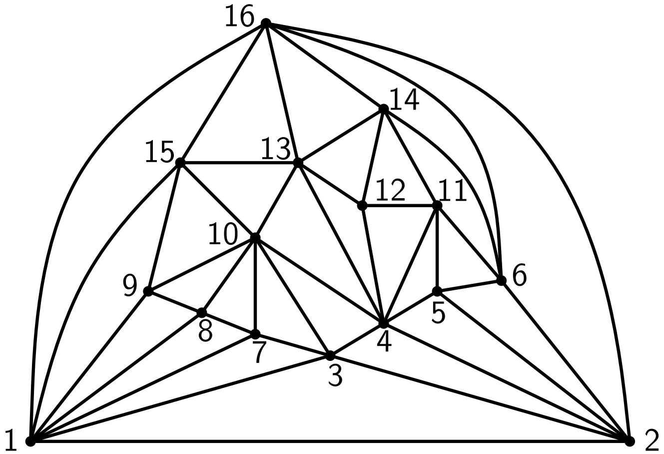


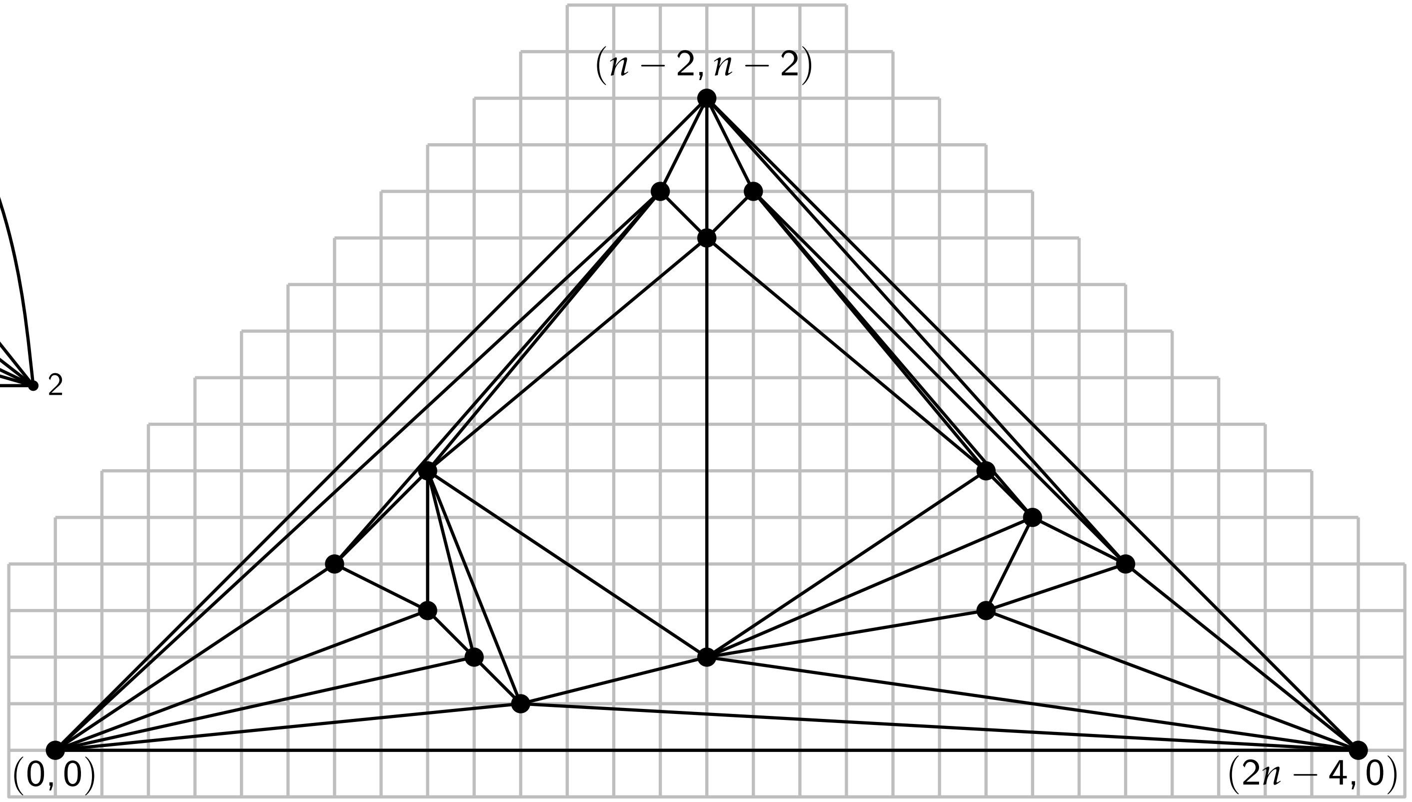




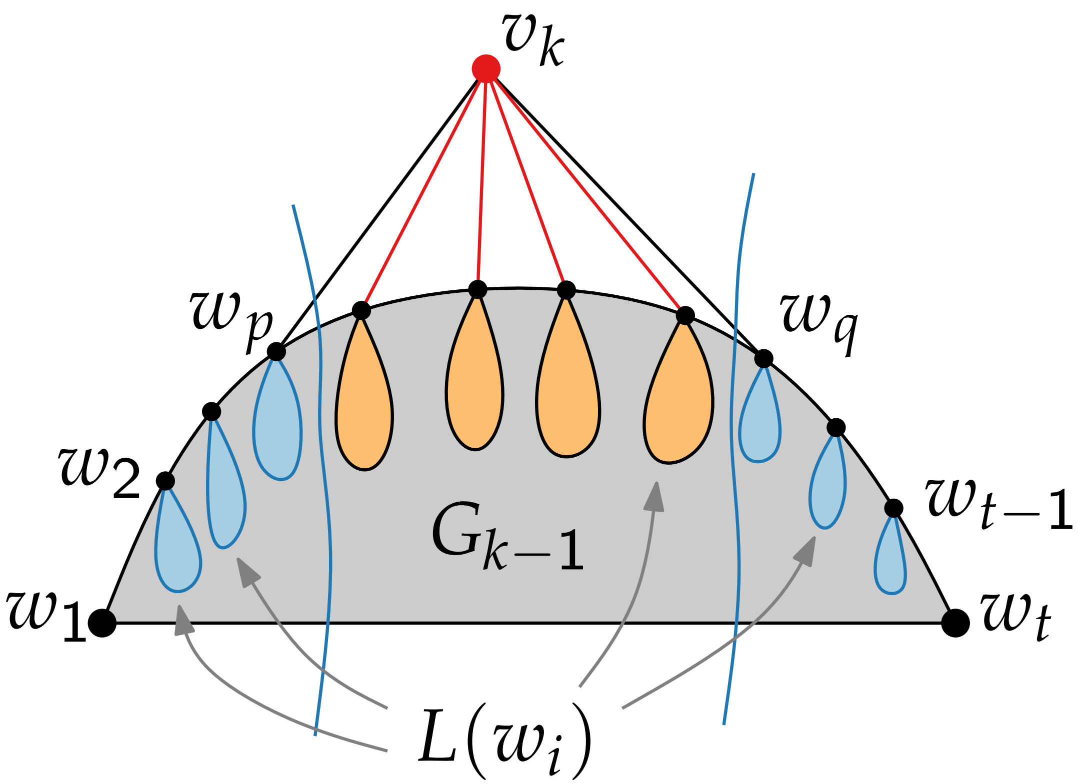
Shift method – example







Shift method – planarity

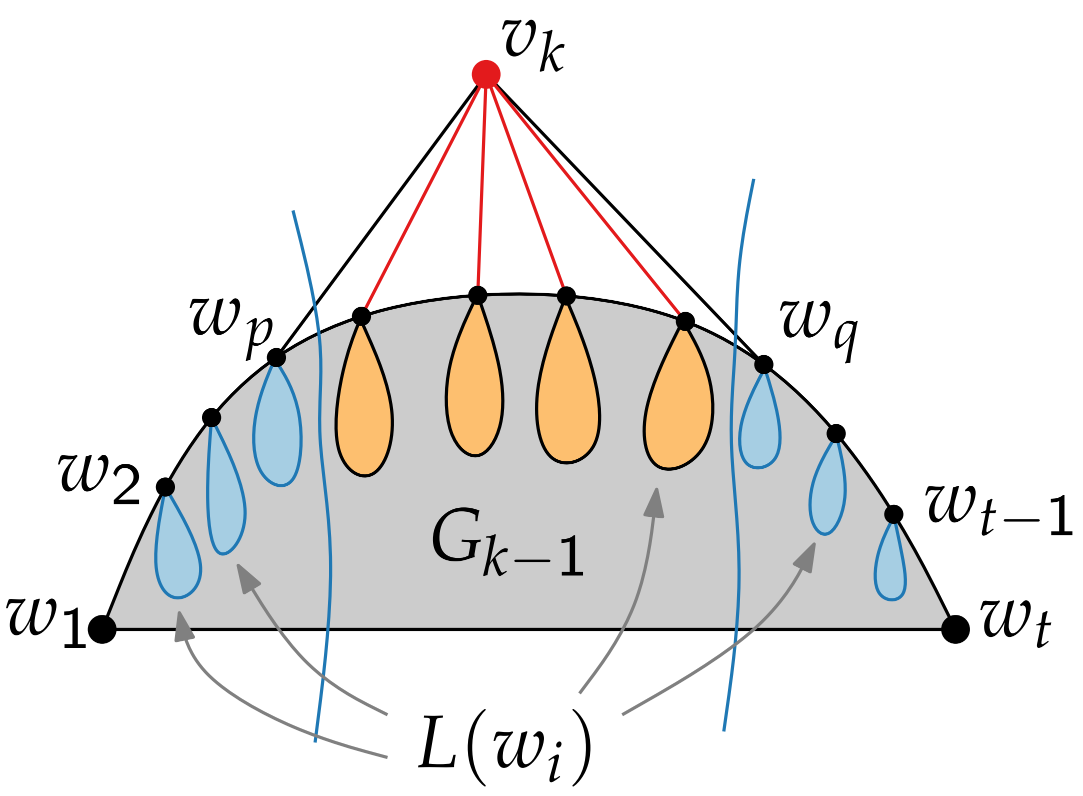


Observations.

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

Shift method – planarity

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



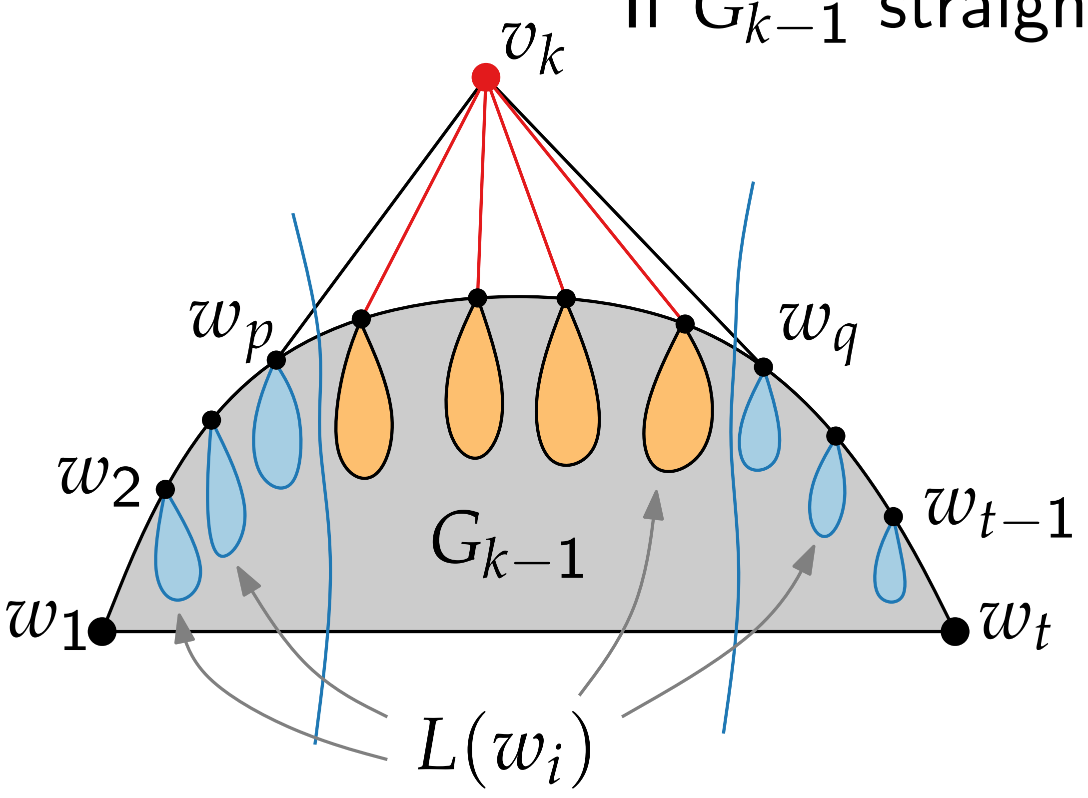
Observations.

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

Shift method – planarity

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

Proof by induction:
If G_{k-1} straight-line planar, then also G_k.



Observations.

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

Shift method – pseudocode

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

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

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

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

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

Shift method – pseudocode

Let $v_1, \ldots, v_n$ be a canonical order of G
for $i = 1$ to 3 **do**
  $L(v_i) \leftarrow \{v_i\}$
$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$
for $k = 4$ to n **do**
  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 neighbours of v_k
  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**
    $x(v) \leftarrow x(v) + 2$
  $P(v_k) \leftarrow$ intersection of $+1/-1$ edges from $P(w_p)$ and $P(w_q)$
  $L(v_k) \leftarrow \cup_{j=p+1}^{q-k} L(w_j) \cup \{v_k\}$

Shift method – pseudocode

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

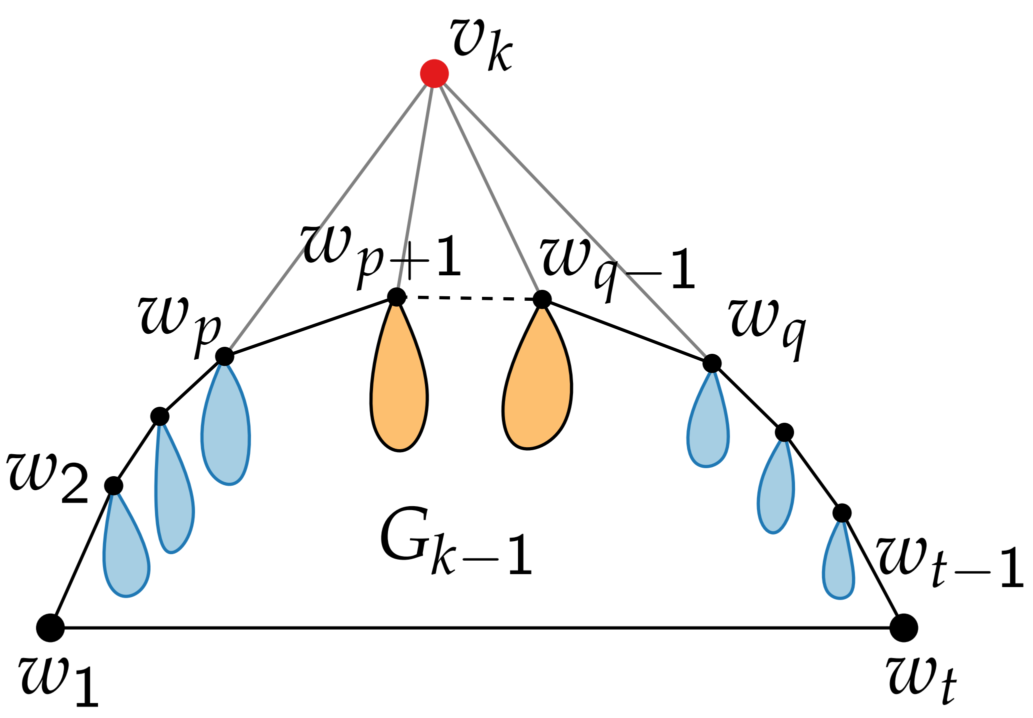
$P(v_1) \leftarrow (0,0); P(v_2) \leftarrow (2,0), P(v_3) \leftarrow (1,1)$
for $k = 4$ to n **do**
  Let $w_1 = v_1, w_2, \ldots, w_{t-1}, w_t = v_2$ denote the boundary of G_{k-1}
  and let $w_p, \ldots, w_q$ be the neighbours of v_k
  for $\forall v \in \cup_{j=p+1}^{q-1} L(w_j)$ **do**
    $x(v)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

- Runtime $\mathcal{O}(n^2)$
- Can we do better?

Shift method – linear time implementation

Idea:

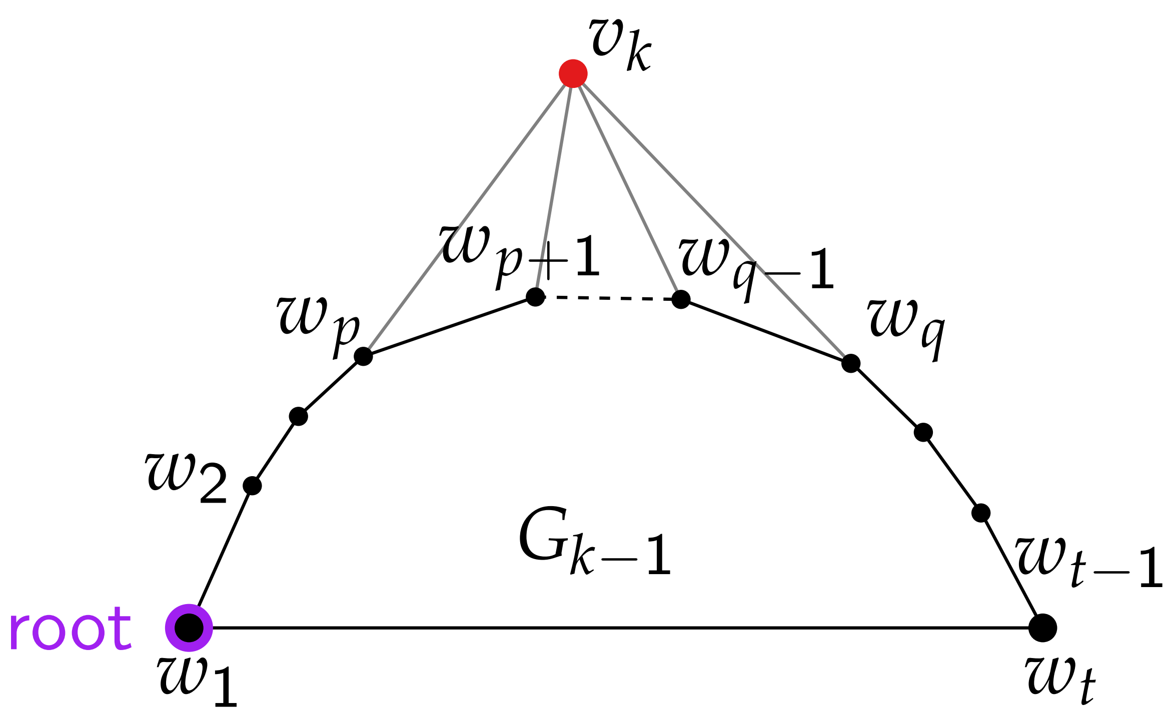
- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1



Shift method – linear time implementation

Idea:

- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1



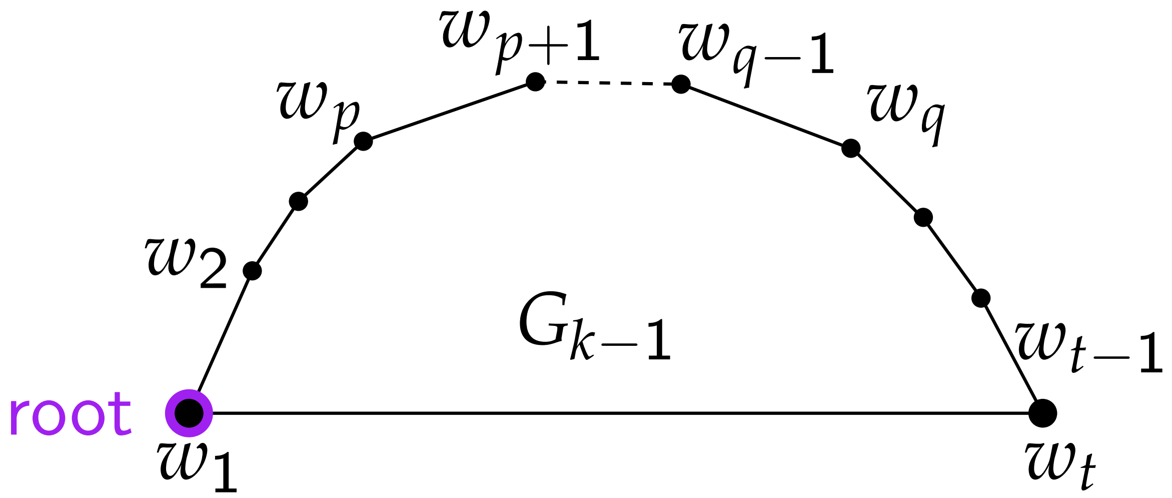
Shift method – linear time implementation

Idea:

- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1

Outerface of G_{k-1}

- at w_i store $\Delta x(w_i) = x(w_i) - x(w_{i-1})$



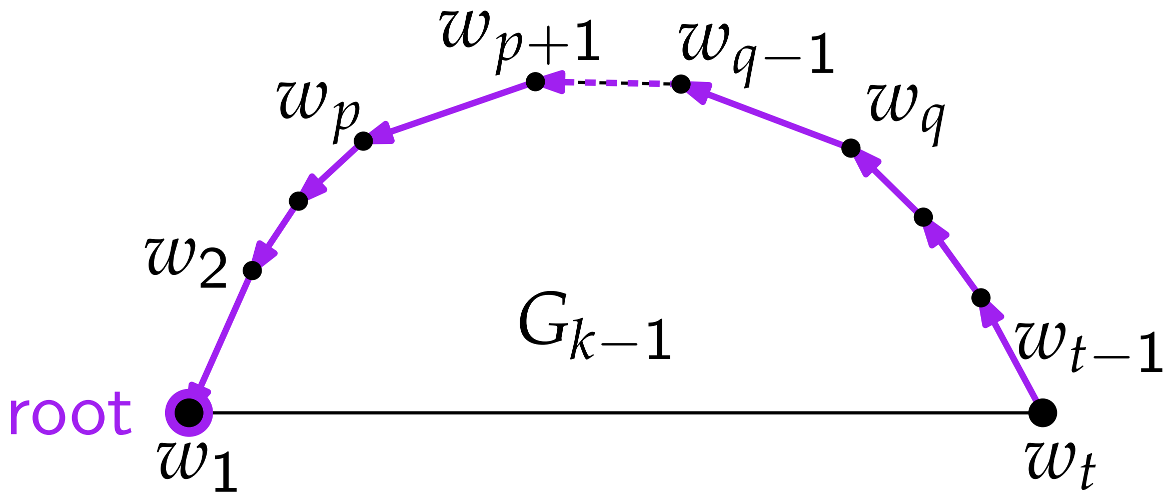
Shift method – linear time implementation

Idea:

- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1

Outerface of G_{k-1}

- at w_i store $\Delta x(w_i) = x(w_i) - x(w_{i-1})$



Shift method – linear time implementation

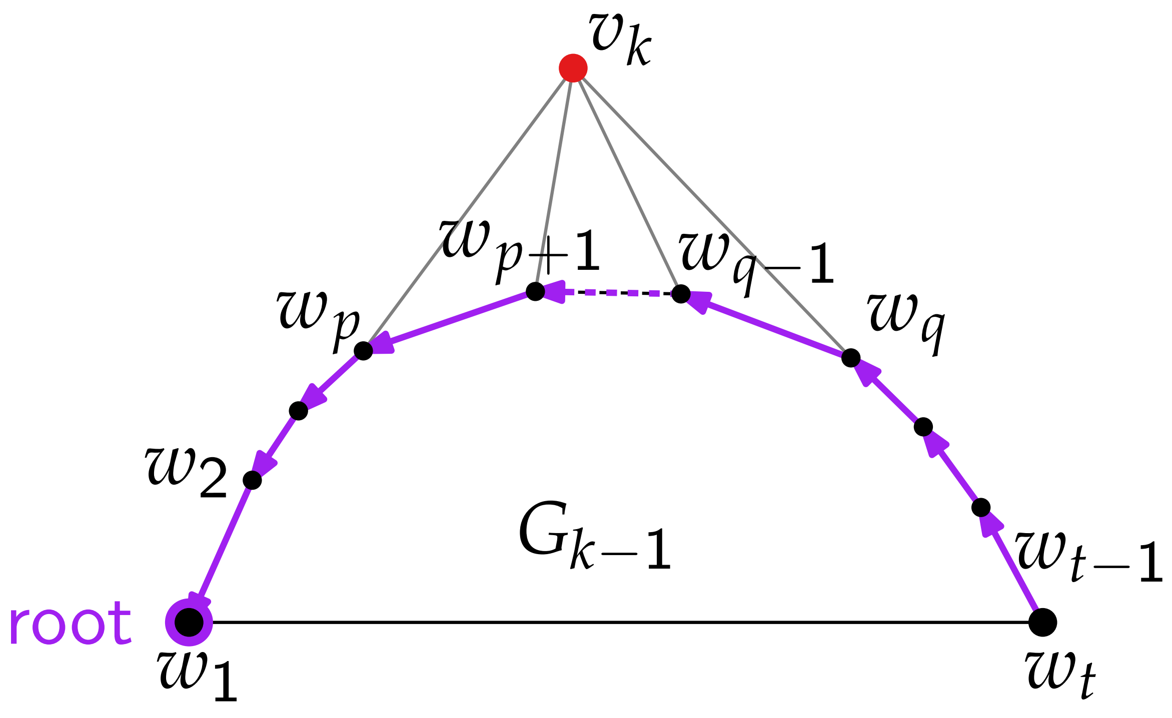
Idea:

- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1

Outerface of G_{k-1}

- at w_i store $\Delta x(w_i) = x(w_i) - x(w_{i-1})$

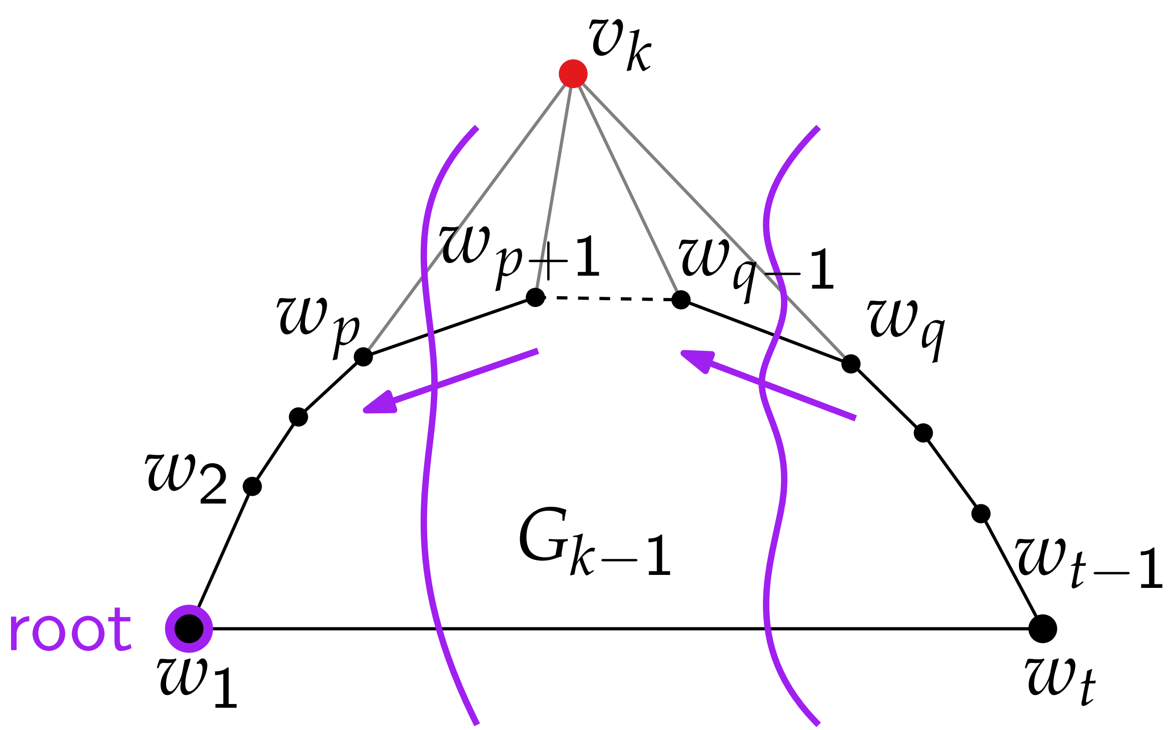
Adding v_k



Shift method – linear time implementation

Idea:

- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1



Outerface of G_{k-1}

- at w_i store $\Delta x(w_i) = x(w_i) - x(w_{i-1})$

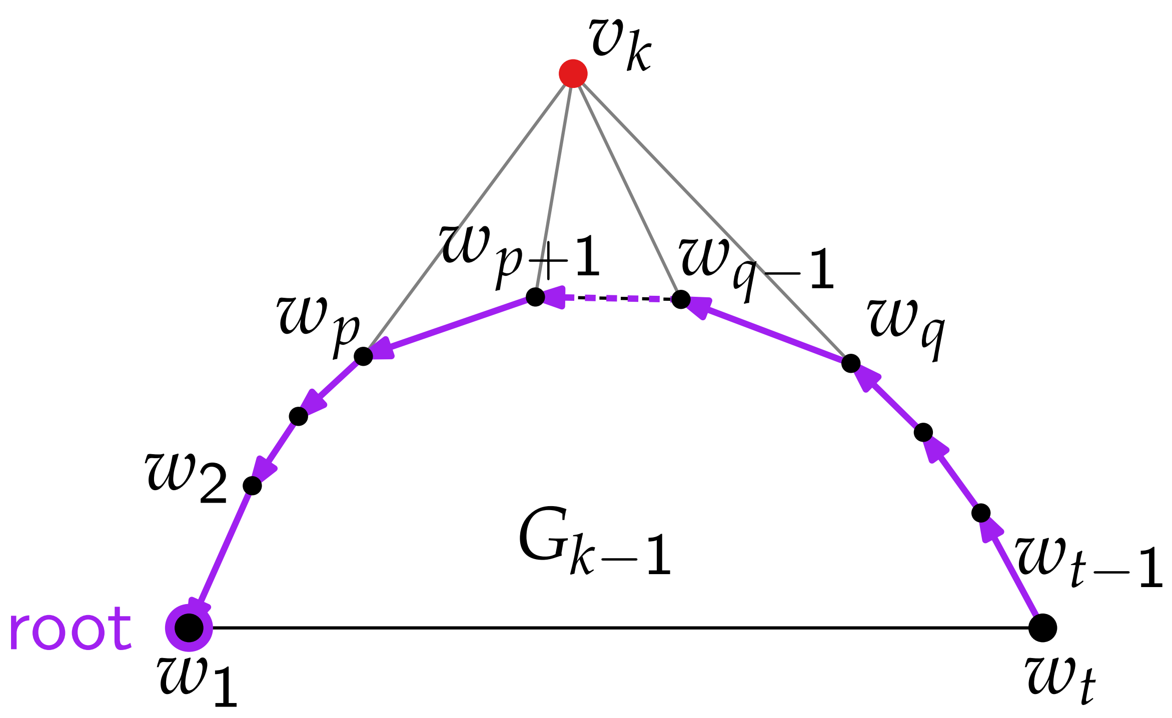
Adding v_k

- Shifting is performed by increasing $\Delta x(w_{p+1})$ and $\Delta x(w_q)$

Shift method – linear time implementation

Idea:

- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1



Outerface of G_{k-1}

- at w_i store $\Delta x(w_i) = x(w_i) - x(w_{i-1})$

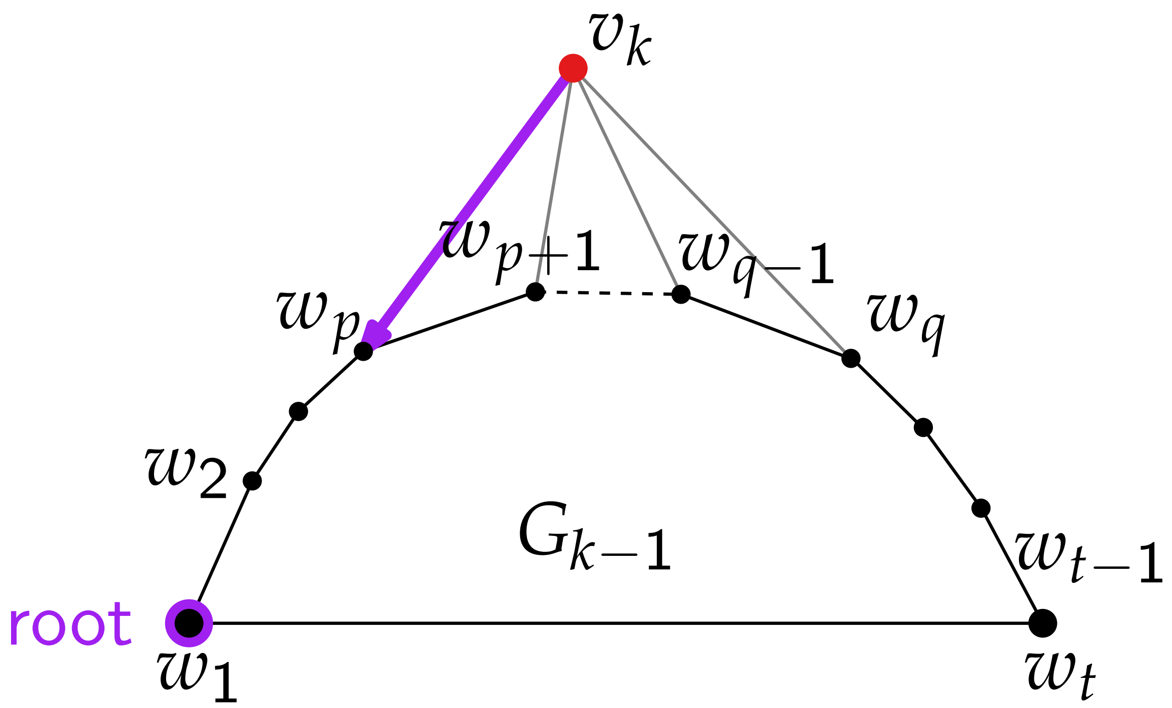
Adding v_k

- Shifting is performed by increasing $\Delta x(w_{p+1})$ and $\Delta x(w_q)$
- $x(v_k)$ depends on $x(w_p)$ and $x(w_q)$

Shift method – linear time implementation

Idea:

- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1



Outerface of G_{k-1}

- at w_i store $\Delta x(w_i) = x(w_i) - x(w_{i-1})$

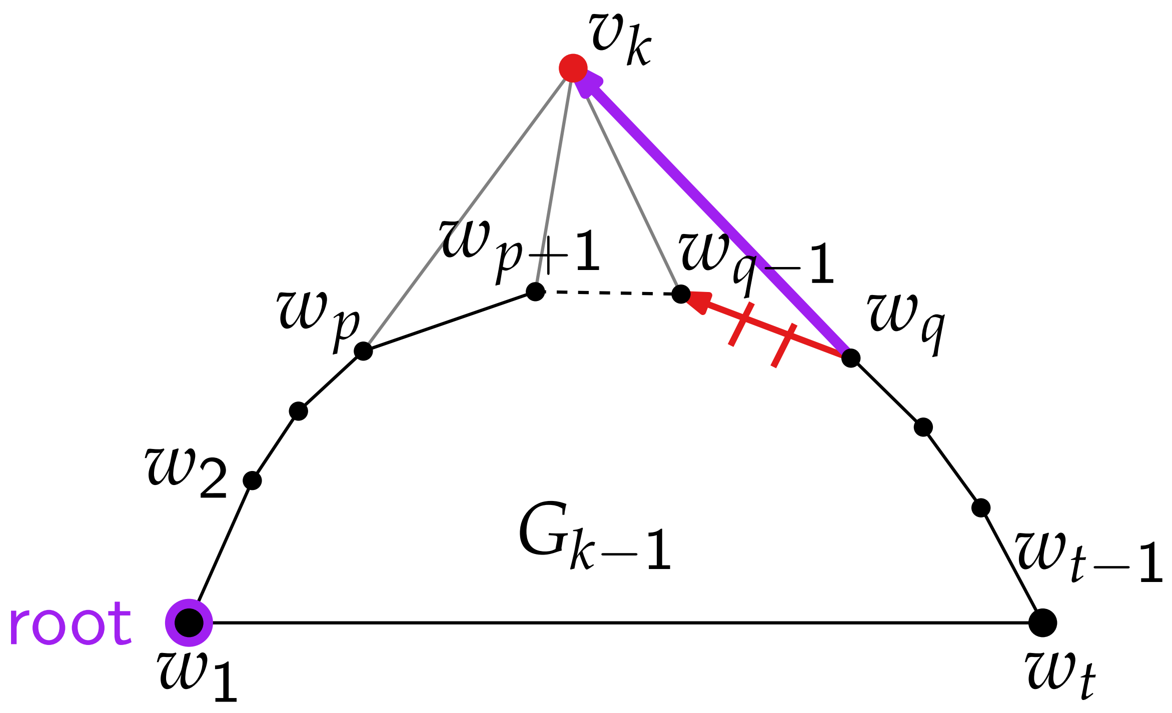
Adding v_k

- Shifting is performed by increasing $\Delta x(w_{p+1})$ and $\Delta x(w_q)$
- $x(v_k)$ depends on $x(w_p)$ and $x(w_q)$
- $x(v_k)$ as x difference from w_p

Shift method – linear time implementation

Idea:

- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1



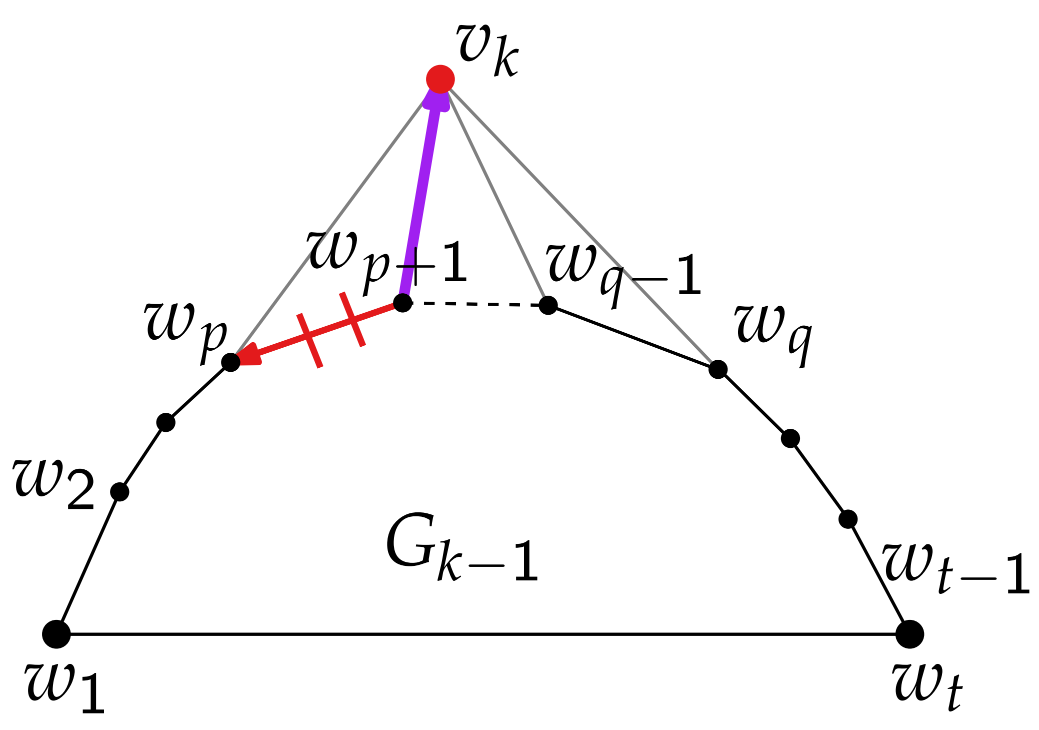
Outerface of G_{k-1}

- at w_i store $\Delta x(w_i) = x(w_i) - x(w_{i-1})$

Adding v_k

- Shifting is performed by increasing $\Delta x(w_{p+1})$ and $\Delta x(w_q)$
- $x(v_k)$ depends on $x(w_p)$ and $x(w_q)$
- $x(v_k)$ as x difference from w_p
- $x(w_q)$ as x difference from v_k

Shift method – linear time implementation



Idea:

- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1

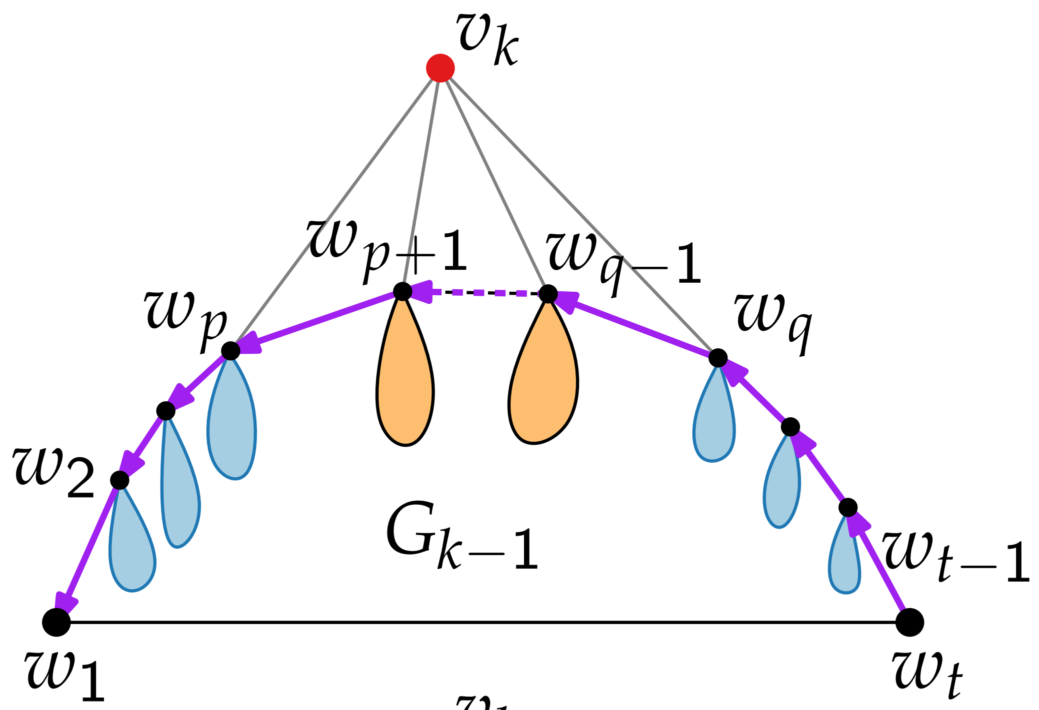
Outerface of G_{k-1}

- at w_i store $\Delta x(w_i) = x(w_i) - x(w_{i-1})$

Adding v_k

- Shifting is performed by increasing $\Delta x(w_{p+1})$ and $\Delta x(w_q)$
- $x(v_k)$ depends on $x(w_p)$ and $x(w_q)$
- $x(v_k)$ as x difference from w_p
- $x(w_q)$ as x difference from v_k
- w_{p+1} covered by v_k
  $\rightarrow x(w_{p+1})$ as x difference from $x(v_k)$

Shift method – linear time implementation



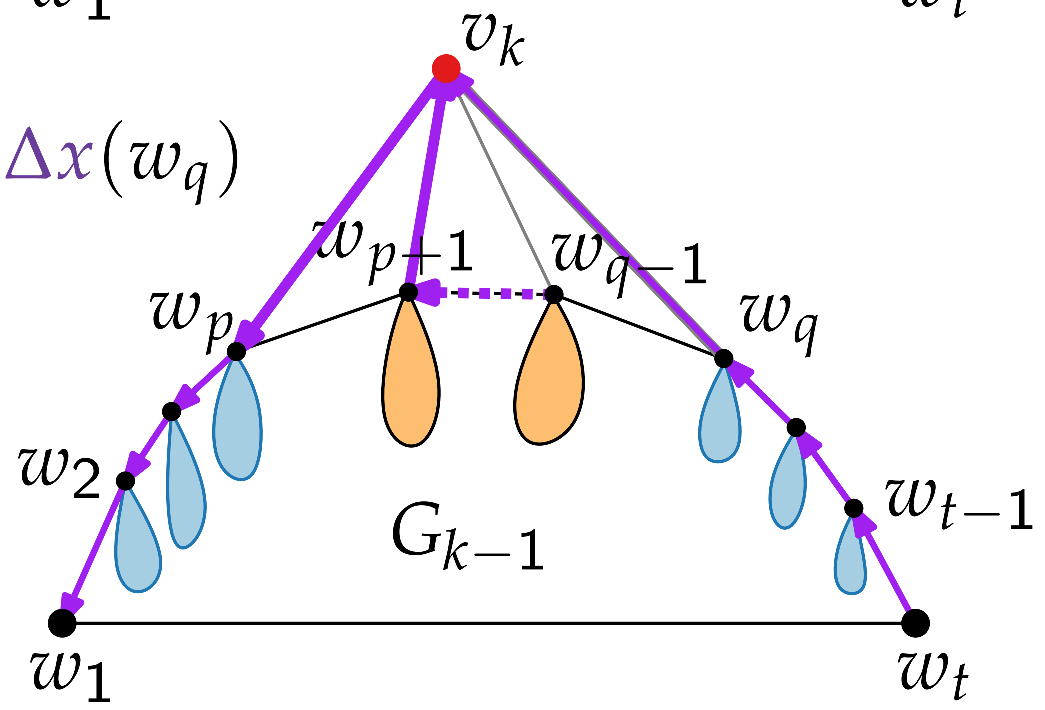
Idea:

- Instead of storing explicit x-coordinates, we store x differences.
- We need a spanning tree rooted at v_1

Outerface of G_{k-1}

- at w_i store $\Delta x(w_i) = x(w_i) - x(w_{i-1})$

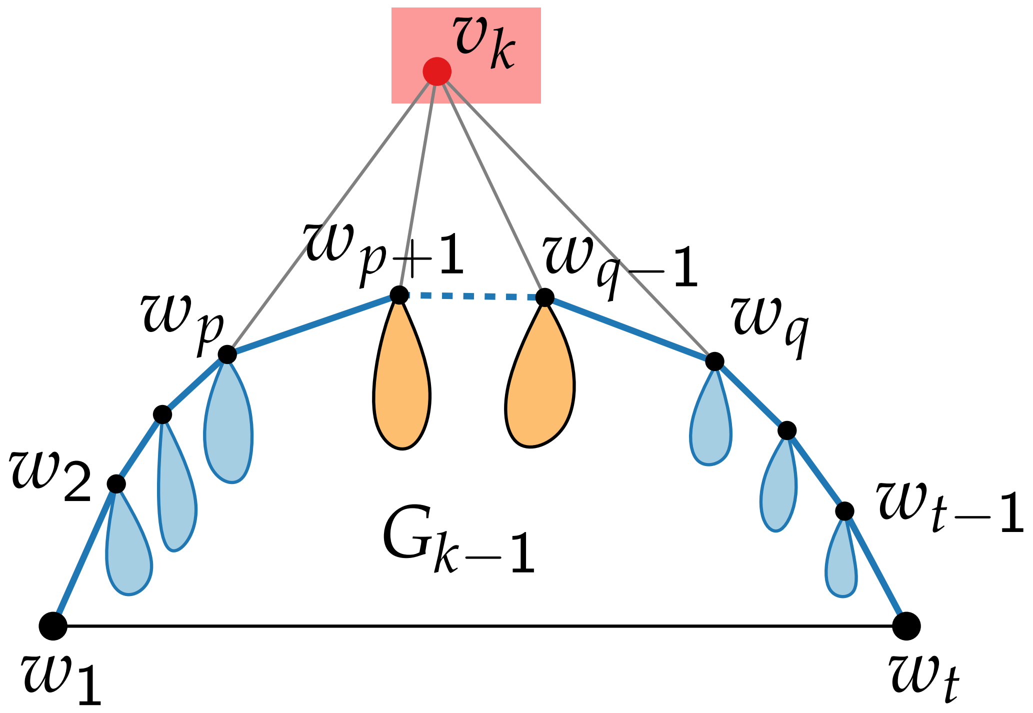
Adding v_k



- Shifting is performed by increasing $\Delta x(w_{p+1})$ and $\Delta x(w_q)$
- $x(v_k)$ depends on $x(w_p)$ and $x(w_q)$
- $x(v_k)$ as x difference from w_p
- $x(w_q)$ as x difference from v_k
- w_{p+1} covered by v_k
  $\rightarrow x(w_{p+1})$ as x difference from $x(v_k)$

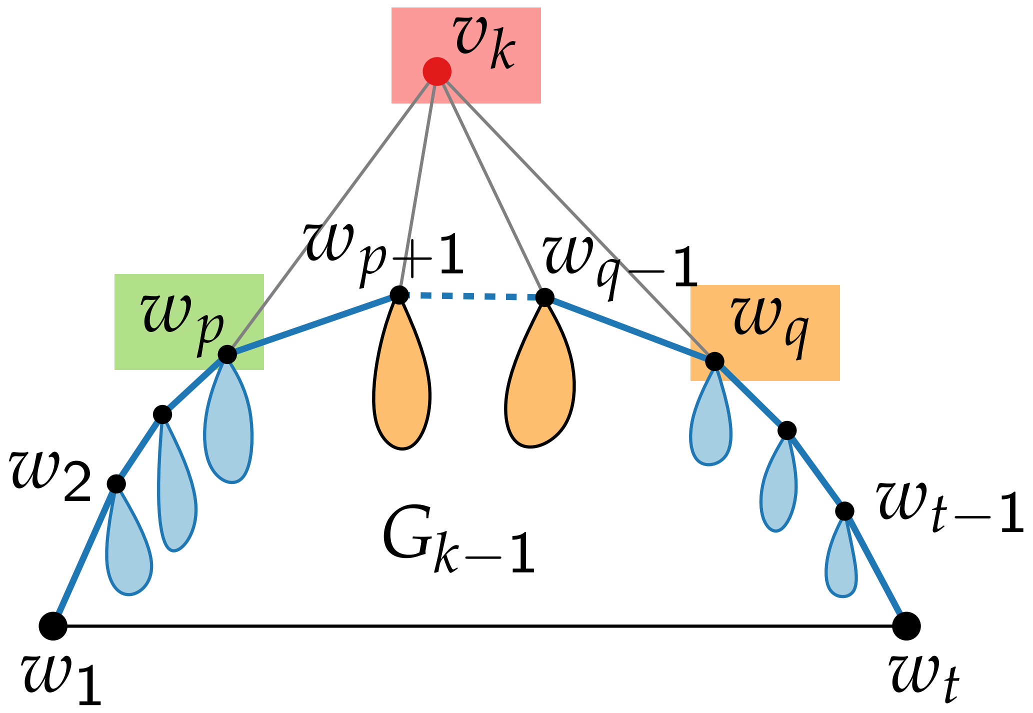
Shift method – linear time implementation

- **Step 1.** compute $x(v_k)$ and $y(v_k)$



Shift method – linear time implementation

- **Step 1.** compute$x(v_k)$ and $y(v_k)$

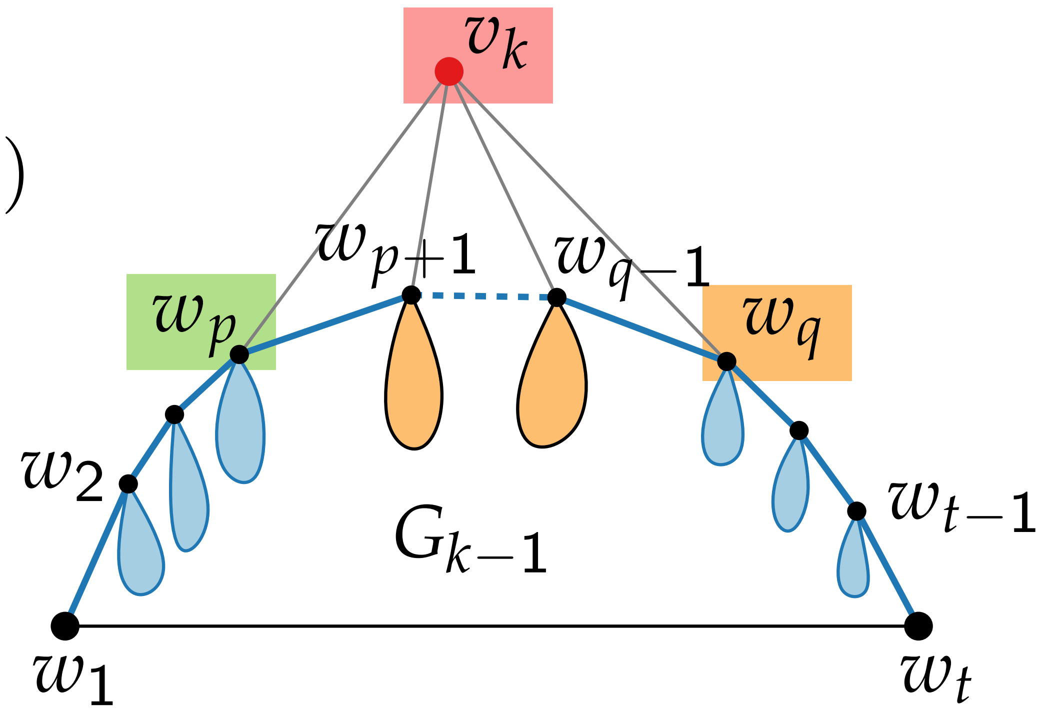


(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

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

Shift method – linear time implementation

- **Step 1.** compute$x(v_k)$ and $y(v_k)$
- **Step 1 revised.** compute $x(v_k) - x(w_p)$ and $y(v_k)$

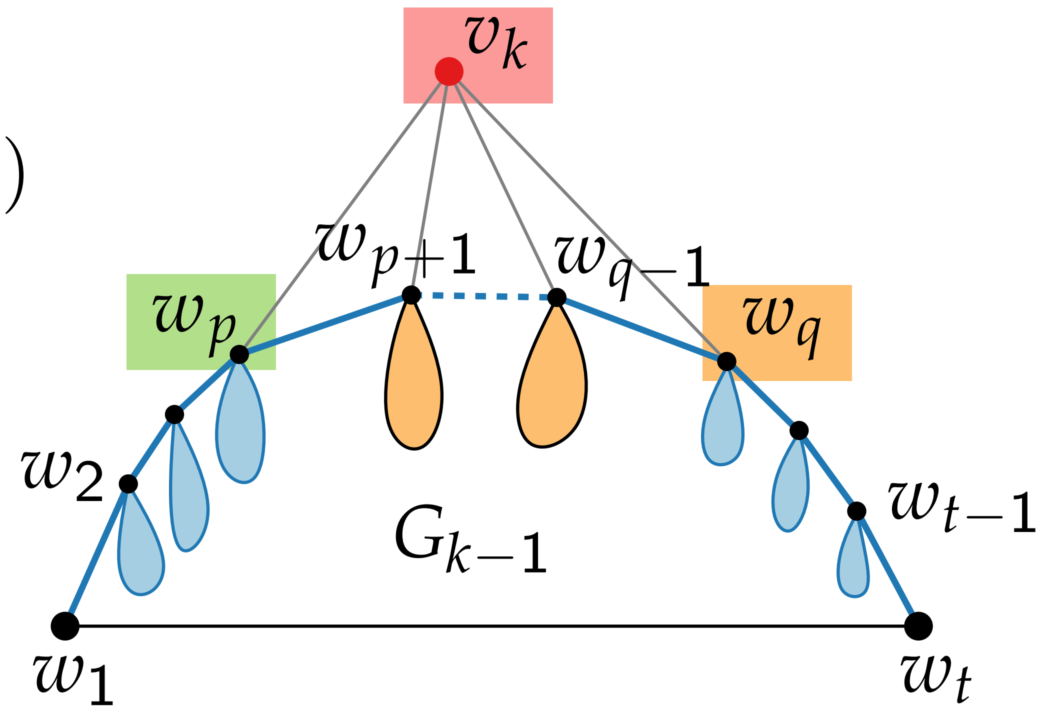


(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

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

Shift method – linear time implementation

- **Step 1.** compute$x(v_k)$ and $y(v_k)$
- **Step 1 revised.** compute $x(v_k) - x(w_p)$ and $y(v_k)$



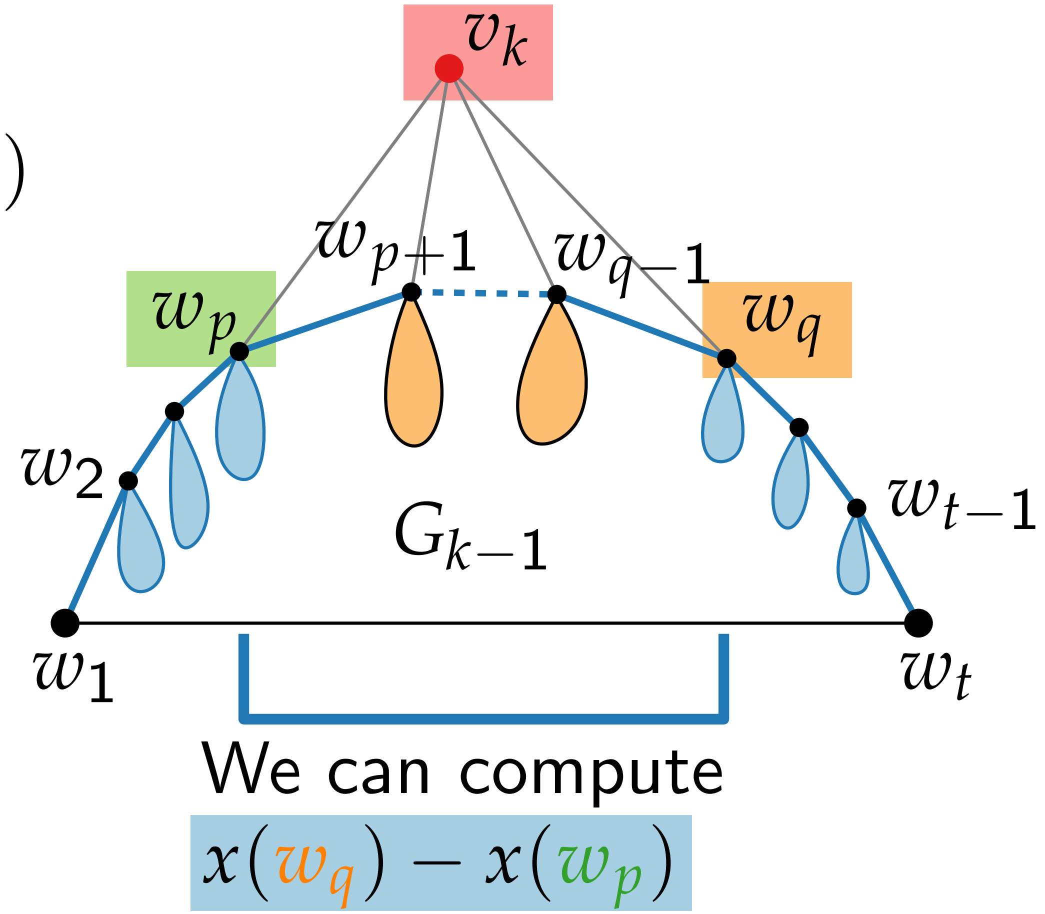
(1) $x(v_k) = \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))$

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

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

Shift method – linear time implementation

- **Step 1.** compute$x(v_k)$ and $y(v_k)$
- **Step 1 revised.** compute $x(v_k) - x(w_p)$ and $y(v_k)$



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

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

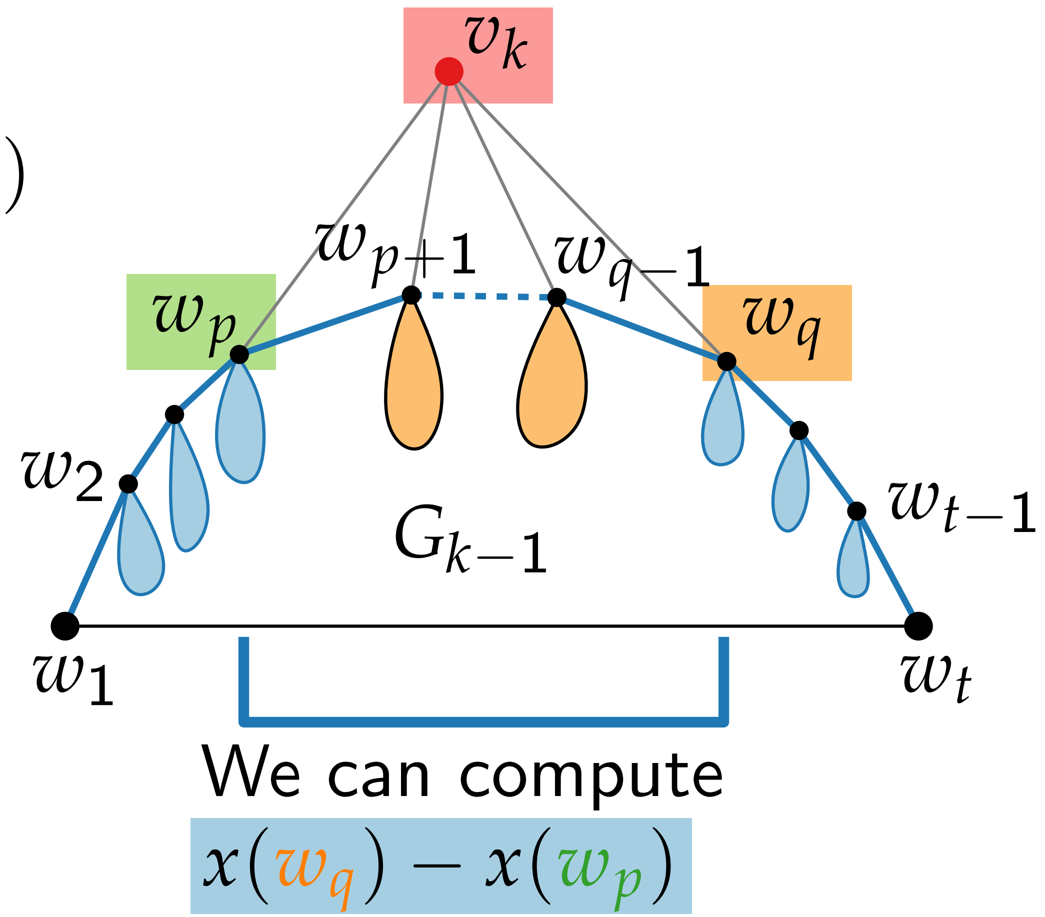
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift method – linear time implementation

- **Step 1.** compute$x(v_k)$ and $y(v_k)$
- **Step 1 revised.** compute $x(v_k) - x(w_p)$ and $y(v_k)$

Step 2- Calculations.

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



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

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

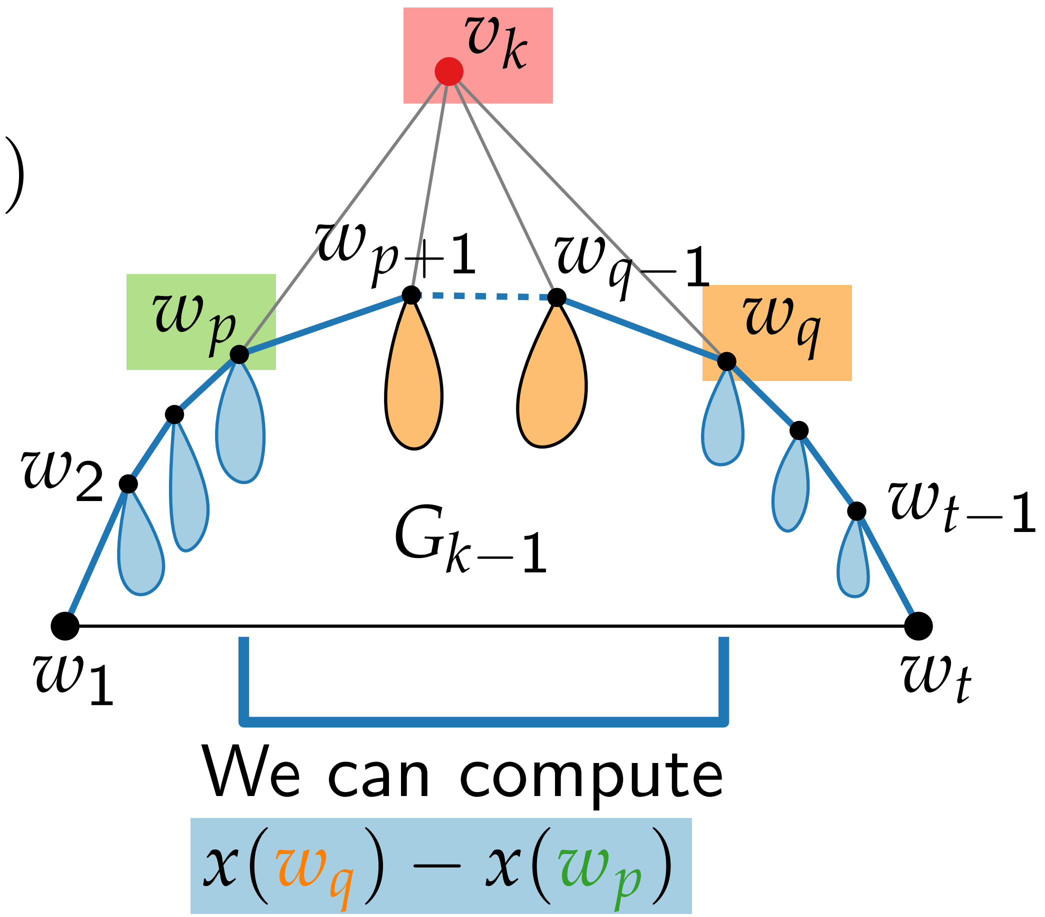
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift method – linear time implementation

- **Step 1.** compute$x(v_k)$ and $y(v_k)$
- **Step 1 revised.** compute $x(v_k) - x(w_p)$ and $y(v_k)$

Step 2- Calculations.

- $\Delta x(w_{p+1})$++, $\Delta x(w_q)$++
- $x(w_q) - x(w_p) = \Delta x(w_{p+1}) + \ldots + \Delta x(w_q)$



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

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

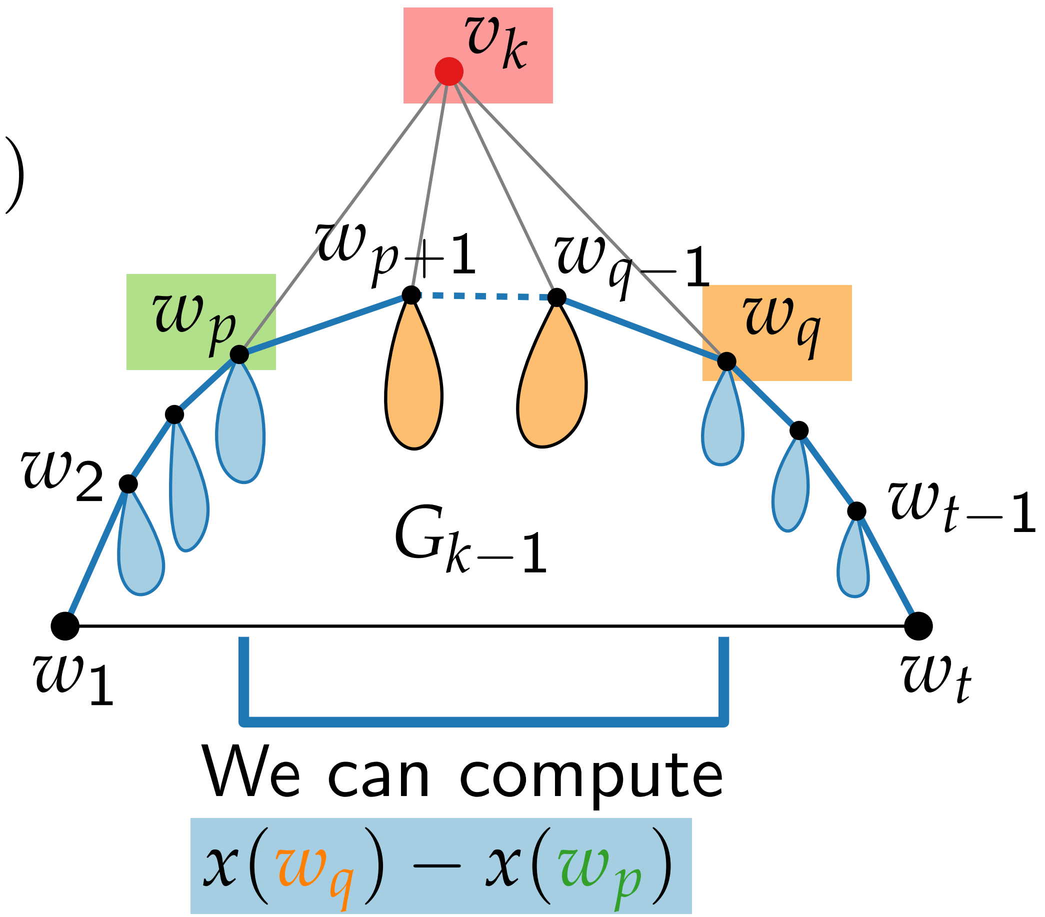
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift method – linear time implementation

- **Step 1.** compute $x(v_k)$ and $y(v_k)$
- **Step 1 revised.** compute $x(v_k) - x(w_p)$ and $y(v_k)$

Step 2- Calculations.

- $\Delta x(w_{p+1})$++, $\Delta x(w_q)$++
- $x(w_q) - x(w_p) = \Delta x(w_{p+1}) + \ldots + \Delta x(w_q)$
- $\Delta x(v_k)$ by (3)



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

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

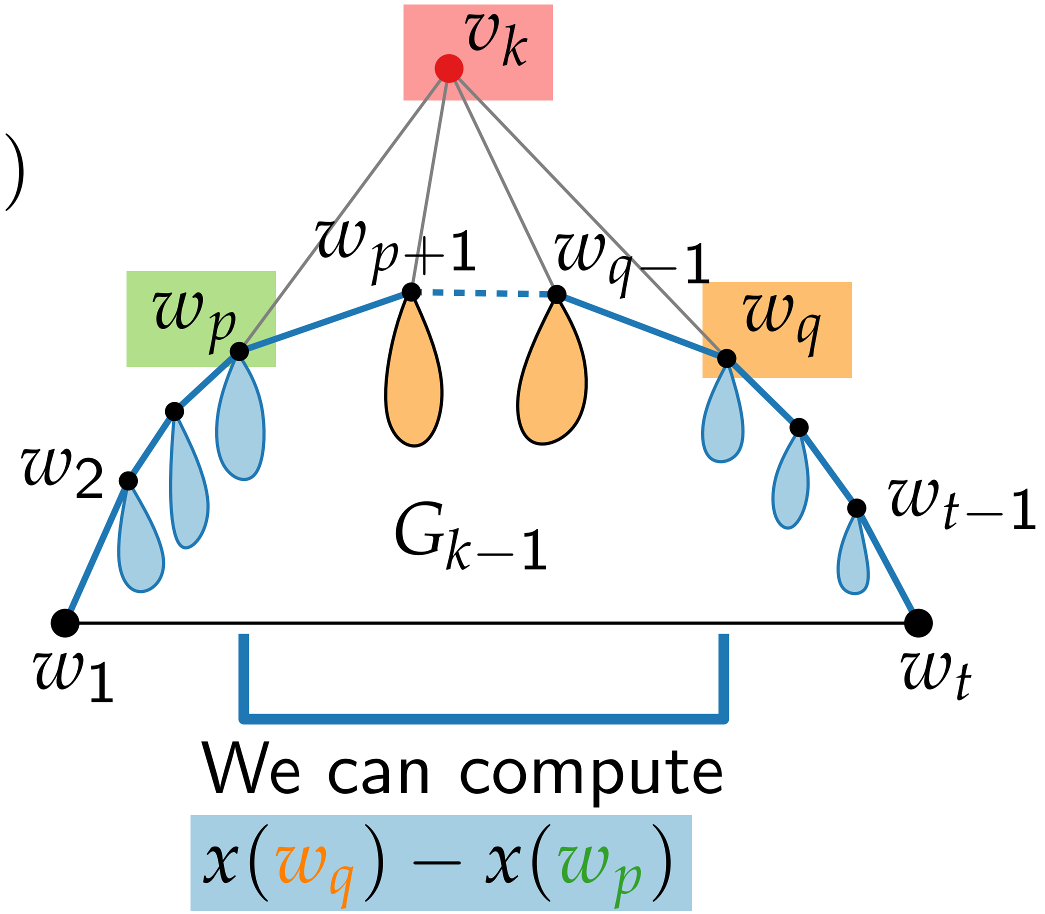
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift method – linear time implementation

- **Step 1.** compute$x(v_k)$ and $y(v_k)$
- **Step 1 revised.** compute $x(v_k) - x(w_p)$ and $y(v_k)$

Step 2- Calculations.

- $\Delta x(w_{p+1})$++, $\Delta x(w_q)$++
- $x(w_q) - x(w_p) = \Delta x(w_{p+1}) + \ldots + \Delta x(w_q)$
- $\Delta x(v_k)$ by (3)
- $\Delta x(w_q) = x(w_q) - x(w_p) - \Delta x(v_k)$



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

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

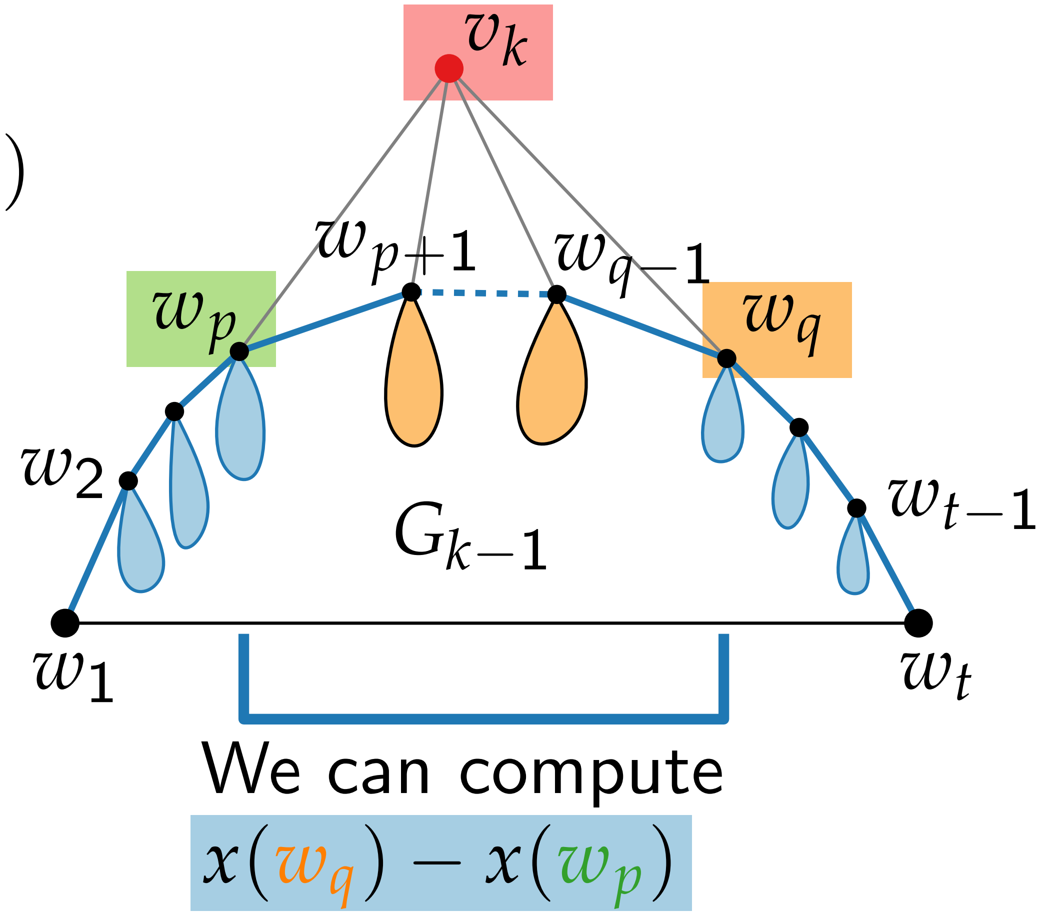
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift method – linear time implementation

- **Step 1.** compute$x(v_k)$ and $y(v_k)$
- **Step 1 revised.** compute $x(v_k) - x(w_p)$ and $y(v_k)$

Step 2- Calculations.

- $\Delta x(w_{p+1})$++, $\Delta x(w_q)$++
- $x(w_q) - x(w_p) = \Delta x(w_{p+1}) + \ldots + \Delta x(w_q)$
- $\Delta x(v_k)$ by (3)
- $\Delta x(w_q) = x(w_q) - x(w_p) - \Delta x(v_k)$
- $\Delta x(w_{p+1}) = \Delta x(w_{p+1}) - \Delta x(v_k)$



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

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

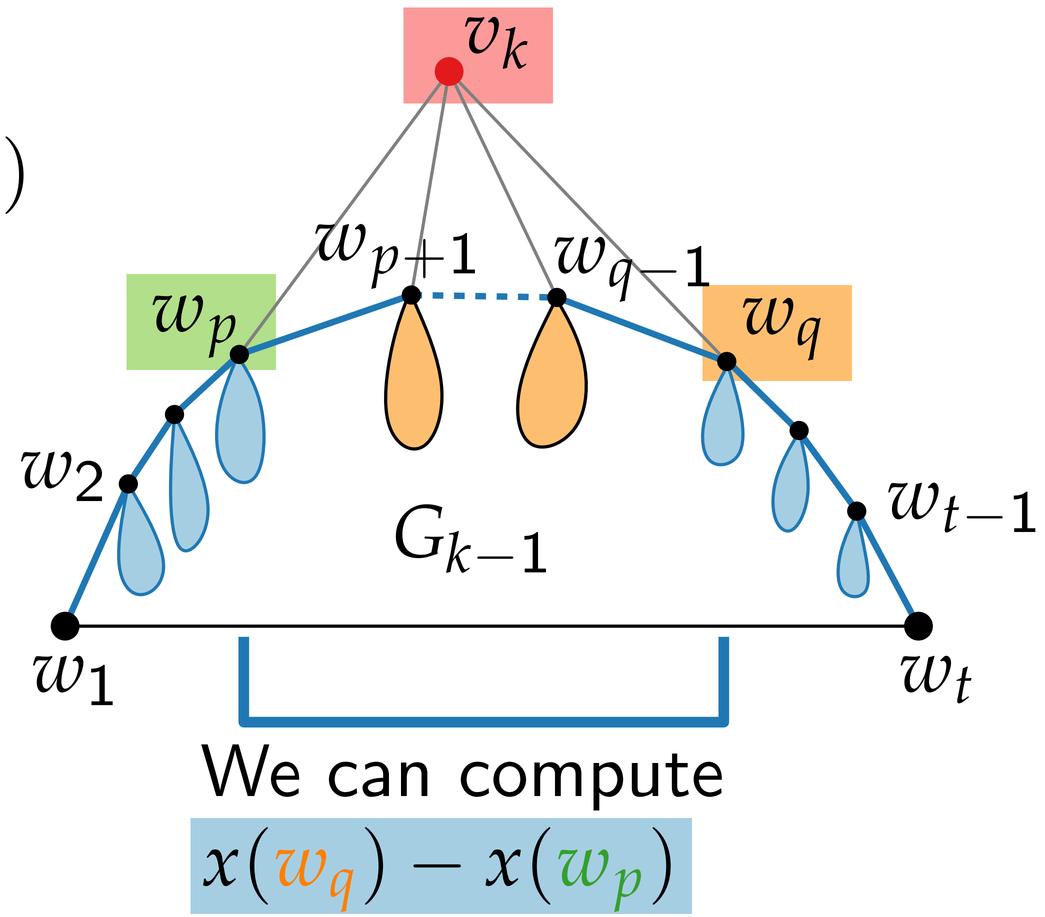
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift method – linear time implementation

- **Step 1.** compute$x(v_k)$ and $y(v_k)$
- **Step 1 revised.** compute $x(v_k) - x(w_p)$ and $y(v_k)$

Step 2- Calculations.

- $\Delta x(w_{p+1})$++, $\Delta x(w_q)$++
- $x(w_q) - x(w_p) = \Delta x(w_{p+1}) + \ldots + \Delta x(w_q)$
- $\Delta x(v_k)$ by (3)
- $\Delta x(w_q) = x(w_q) - x(w_p) - \Delta x(v_k)$
- $\Delta x(w_{p+1}) = \Delta x(w_{p+1}) - \Delta x(v_k)$
- $y(v_k)$ by (2)



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

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

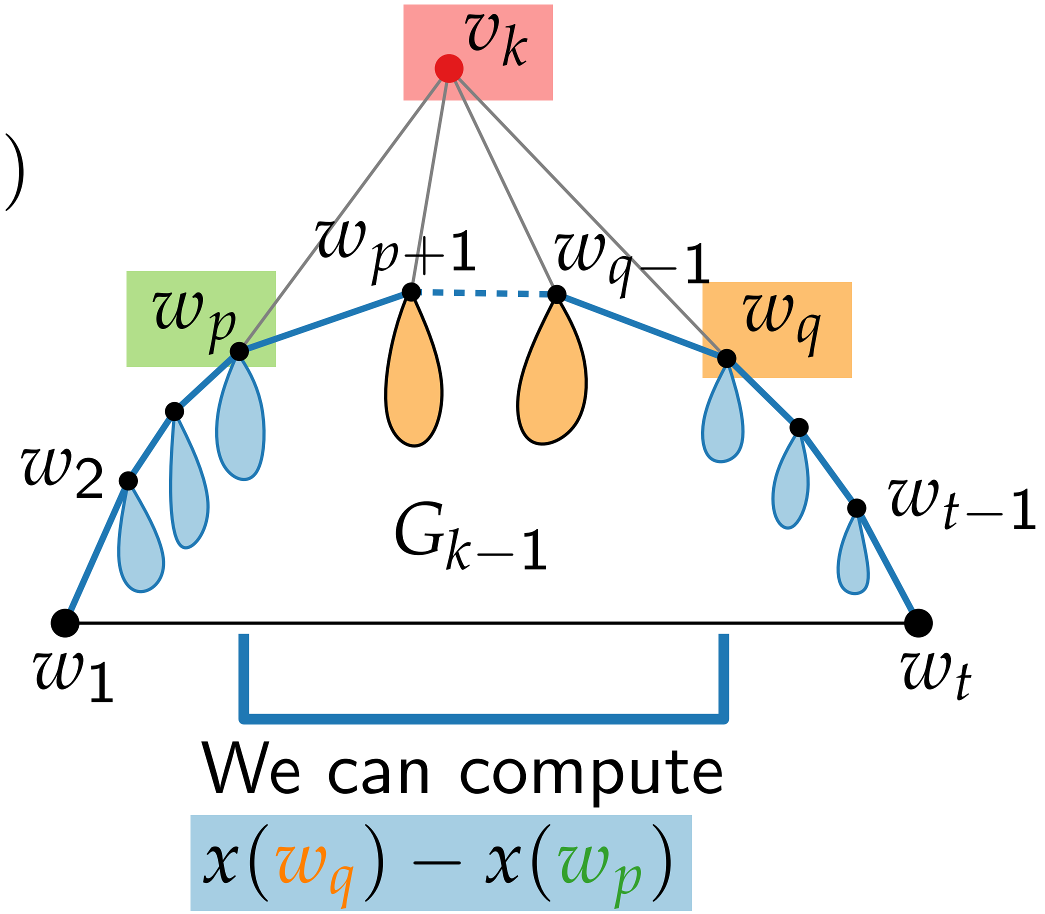
(3) $x(v_k) - x(w_p) = \frac{1}{2}(x(w_q) - x(w_p) + y(w_q) - y(w_p))$

Shift method – linear time implementation

- **Step 1.** compute$x(v_k)$ and $y(v_k)$
- **Step 1 revised.** compute $x(v_k) - x(w_p)$ and $y(v_k)$

Step 2- Calculations.

- $\Delta x(w_{p+1})$++, $\Delta x(w_q)$++
- $x(w_q) - x(w_p) = \Delta x(w_{p+1}) + \ldots + \Delta x(w_q)$
- $\Delta x(v_k)$ by (3)
- $\Delta x(w_q) = x(w_q) - x(w_p) - \Delta x(v_k)$
- $\Delta x(w_{p+1}) = \Delta x(w_{p+1}) - \Delta x(v_k)$
- $y(v_k)$ by (2)



After v_n, use preorder traversal to compute x-coordinates

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

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

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

Literature

- [dFPP90] de Fraysseix, Pach, Pollack *"How to draw a planar graph on a grid"*, Combinatorica, 1990