Embedding Planar Graphs on the Grid

Marek Chrobak & Thomas H. Payne, 1995

Pascal Hein

22. Juni 2018

Seminar „Algorithms on Sparse Graphs“, RWTH Aachen University

Inhalt

Einleitung

Visualisierung von Graphen

Interne Darstellungen: effizient, aber schwierig vorzustellen

- Adjazenzlisten, -matrix, Kantenmenge, ...

(Planare) Graphen modellieren reale Szenarien!

- Schaltkreise, Karten, Netzwerke, Übergangsdiagramme, Abhängigkeiten, ...

Visualisierung von Graphen

Interne Darstellungen: effizient, aber schwierig vorzustellen

- Adjazenzlisten, -matrix, Kantenmenge, ...

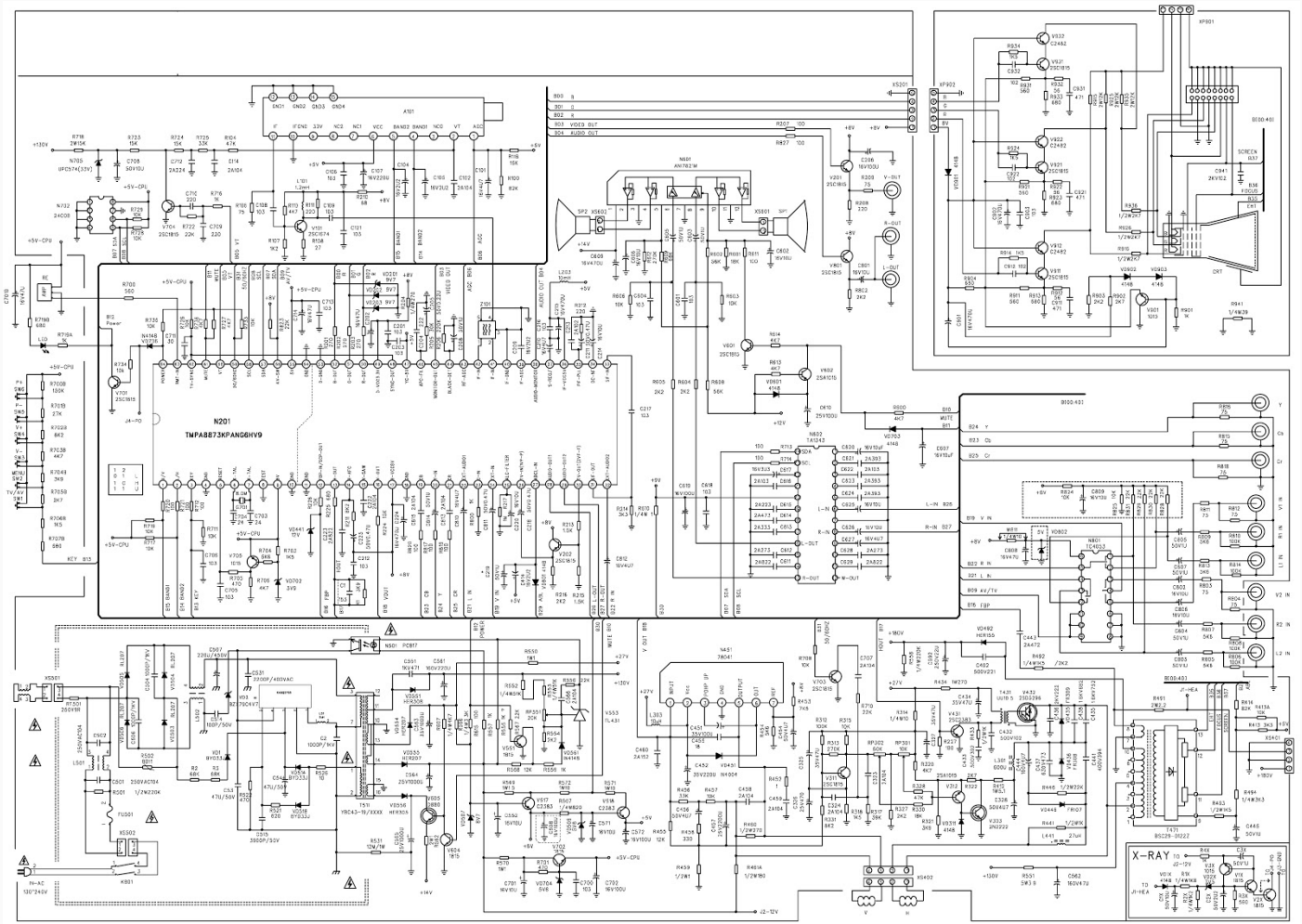
(Planare) Graphen modellieren reale Szenarien!

- Schaltkreise, Karten, Netzwerke, Übergangsdiagramme, Abhängigkeiten, ...

Graph-Visualisierung hilft:

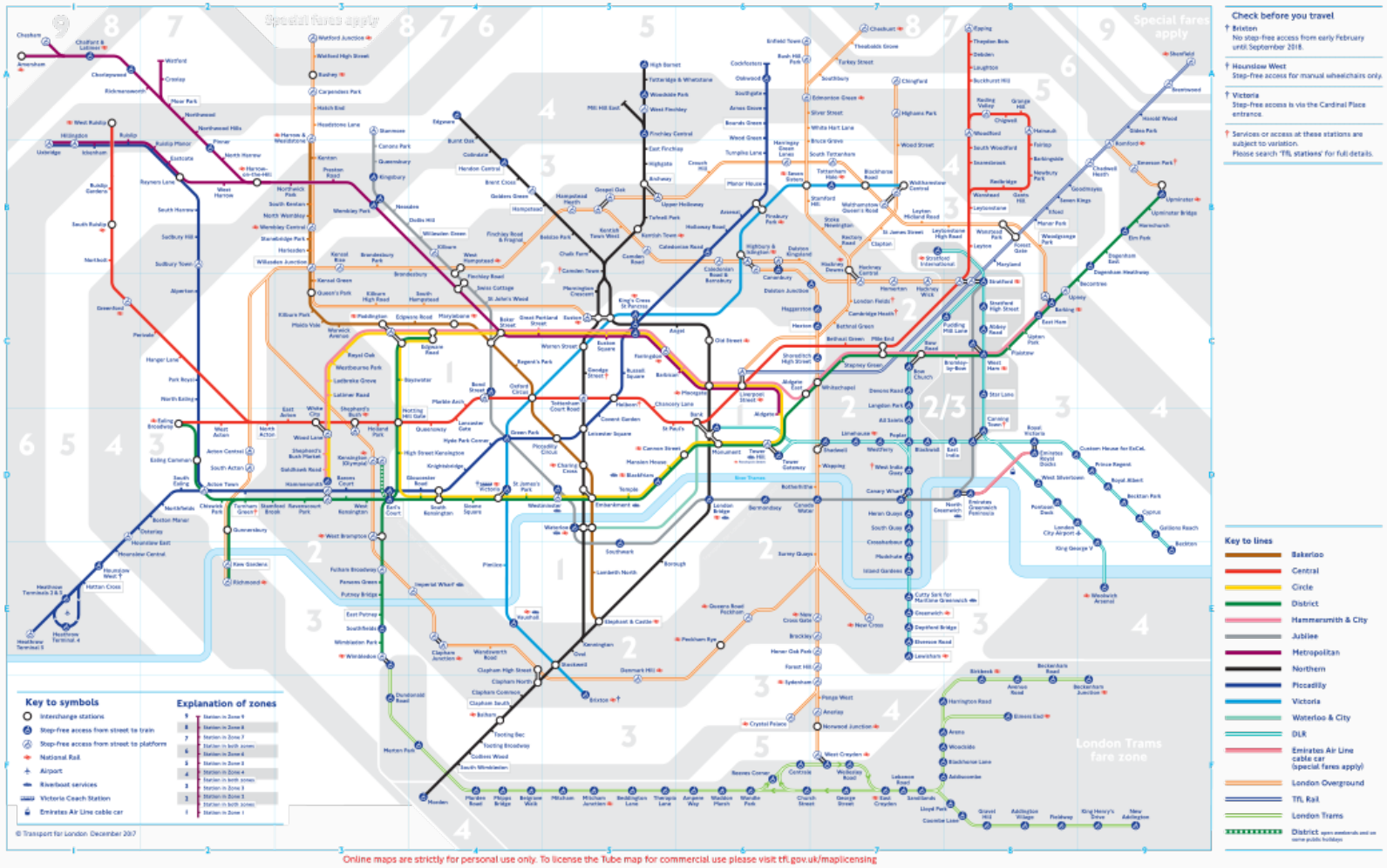
- Graphen als Modelle einzusetzen, Ergebnisse zu verstehen;
- sich Algorithmen vorstellen zu können,
- über Graphen zu sprechen.

Schaltkreise[1]



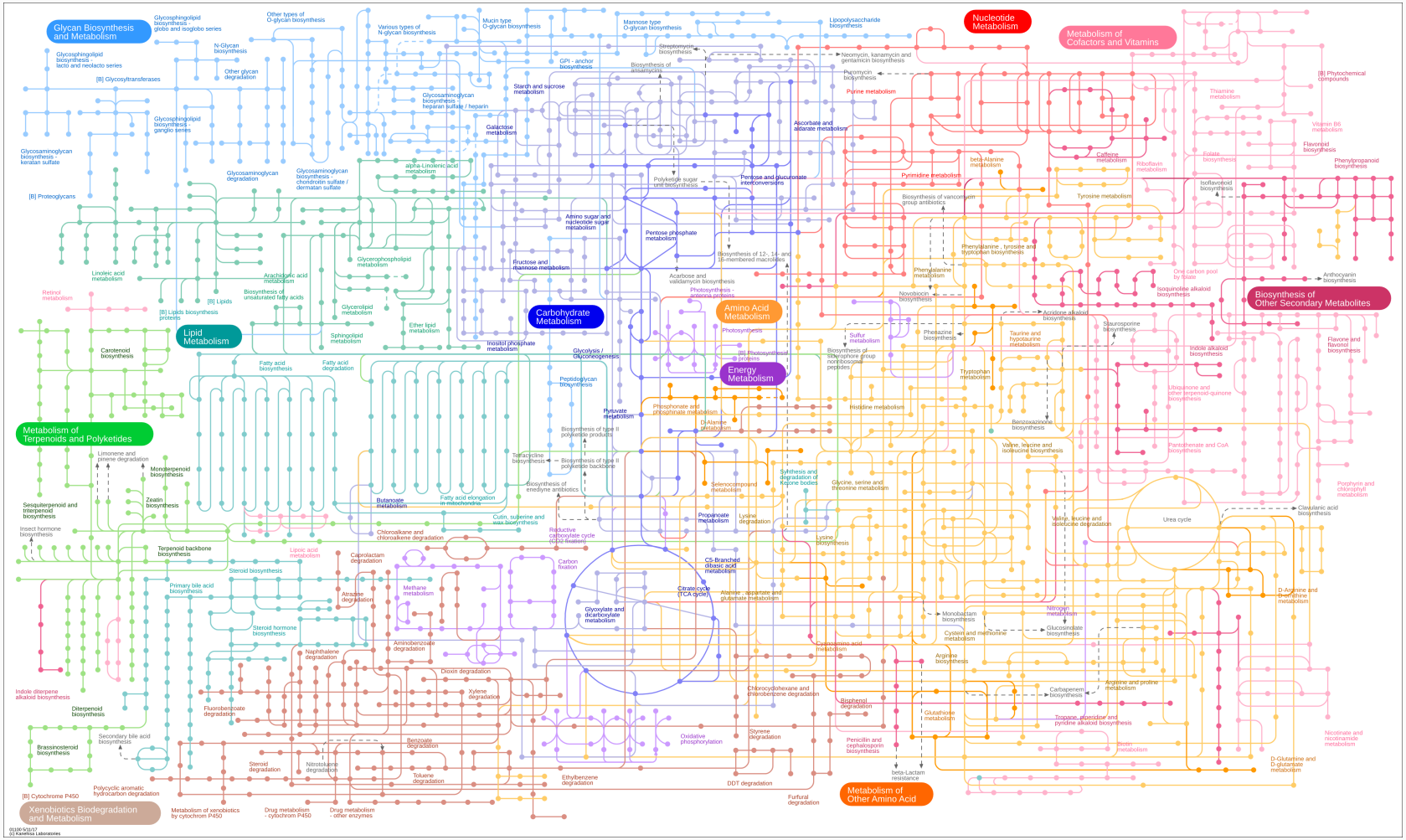
[1] https://electronicshelponline.blogspot.de/2016/01/colour-tv-circuit-diagram.html

Karten[2]



[2]Transport for London, http://content.tfl.gov.uk/large-print-tube-map.pdf

Metabolische Pfade[3]



[3] Yamada *u.a.*, *iPath 3: Interactive Pathway Explorer*, pathways.embl.de/ipath3.cgi

Kriterien für Zeichnungen

„Gut“ hängt von Graph-Typ und Verwendungszweck ab.

- einfacher, korrekter, schneller Algorithmus
- kleine Fläche (bei fester Auflösung)
- wenig Kanten-Kreuzungen
- Form der Kanten
- Anordnung der Knoten, …

Im Allgemeinen: oft (NP-)schwer (z.B. Kanten-Kreuzungen[4])

[4] Garey & Johnson 1983.

Kriterien für Zeichnungen

„Gut“ hängt von Graph-Typ und Verwendungszweck ab.

- einfacher, korrekter, schneller Algorithmus
- kleine Fläche (bei fester Auflösung)
- wenig Kanten-Kreuzungen: keine
- Form der Kanten: geradlinig
- Anordnung der Knoten, …

Im Allgemeinen: oft (NP-)schwer (z.B. Kanten-Kreuzungen[4])

[4] Garey & Johnson 1983.

Ist das möglich?

Satz (Fáry 1948)

Jeder planare Graph kann geradlinig kreuzungsfrei gezeichnet werden.

Beweis (Skizze)

Induktion über die Knotenzahl. Entferne einen Knoten mit $\deg v < 6$ und füge ihn mit geraden Kanten hinzu.

Zeichnungen im Koordinatengitter

Frühe Algorithmen[5][6][7]: hohe Genauigkeit nötig, stark variierende Kantendichte, kompliziert

[5] Tutte 1963.
[6] Chiba, Yamanouchi *u. a.* 1984.
[7] Chiba, Onoguchi *u. a.* 1985.

Zeichnungen im Koordinatengitter

Frühe Algorithmen[5][6][7]: hohe Genauigkeit nötig, stark variierende Kantendichte, kompliziert

Beobachtung (Rosenstiehl & Tarjan 1986)

Auf (kleinen) Koordinatengittern wird wenig Präzision benötigt!
(Aber: ist das immer möglich?)

[5] Tutte 1963.
[6] Chiba, Yamanouchi *u. a.* 1984.
[7] Chiba, Onoguchi *u. a.* 1985.

Zeichnungen im Koordinatengitter

Frühe Algorithmen[5][6][7]: hohe Genauigkeit nötig, stark variierende Kantendichte, kompliziert

Beobachtung (Rosenstiehl & Tarjan 1986)

Auf (kleinen) Koordinatengittern wird wenig Präzision benötigt!
(Aber: ist das immer möglich?)

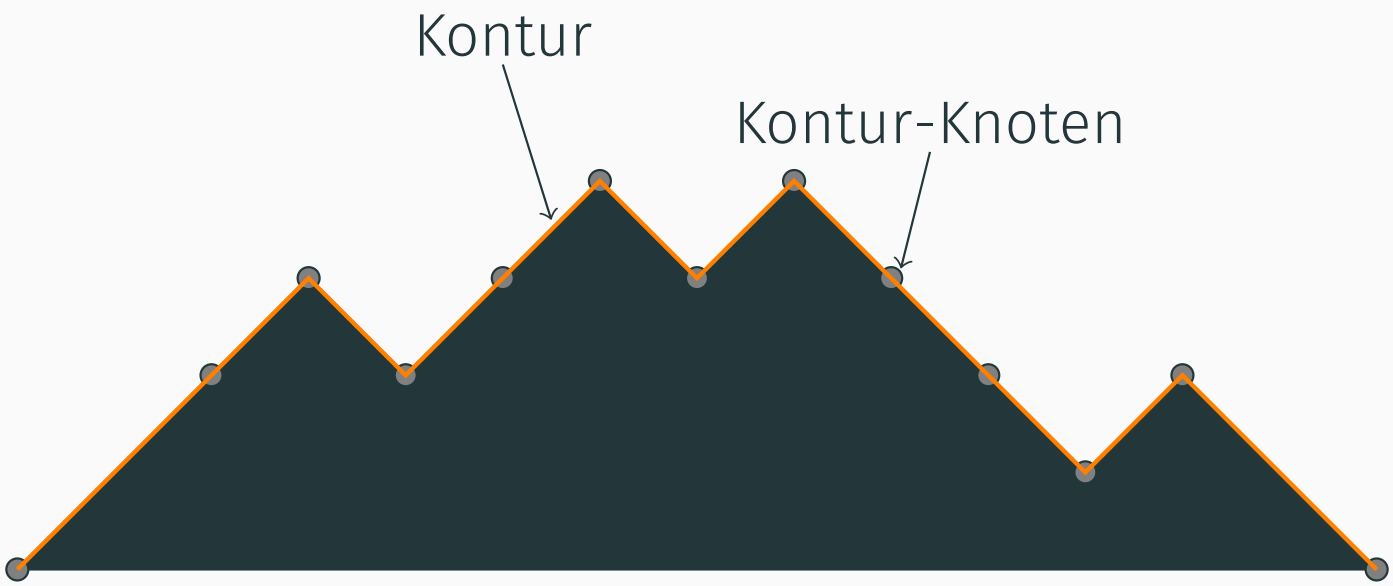
- de Fraysseix *u. a.* 1990: Ja. $\mathcal{O}(n \log n)$; Gitter $(2n-4) \times (n-2)$
- Chrobak & Payne 1995: vereinfachter Algorithmus mit $\mathcal{O}(n)$

[5] Tutte 1963.
[6] Chiba, Yamanouchi *u. a.* 1984.
[7] Chiba, Onoguchi *u. a.* 1985.

Idee

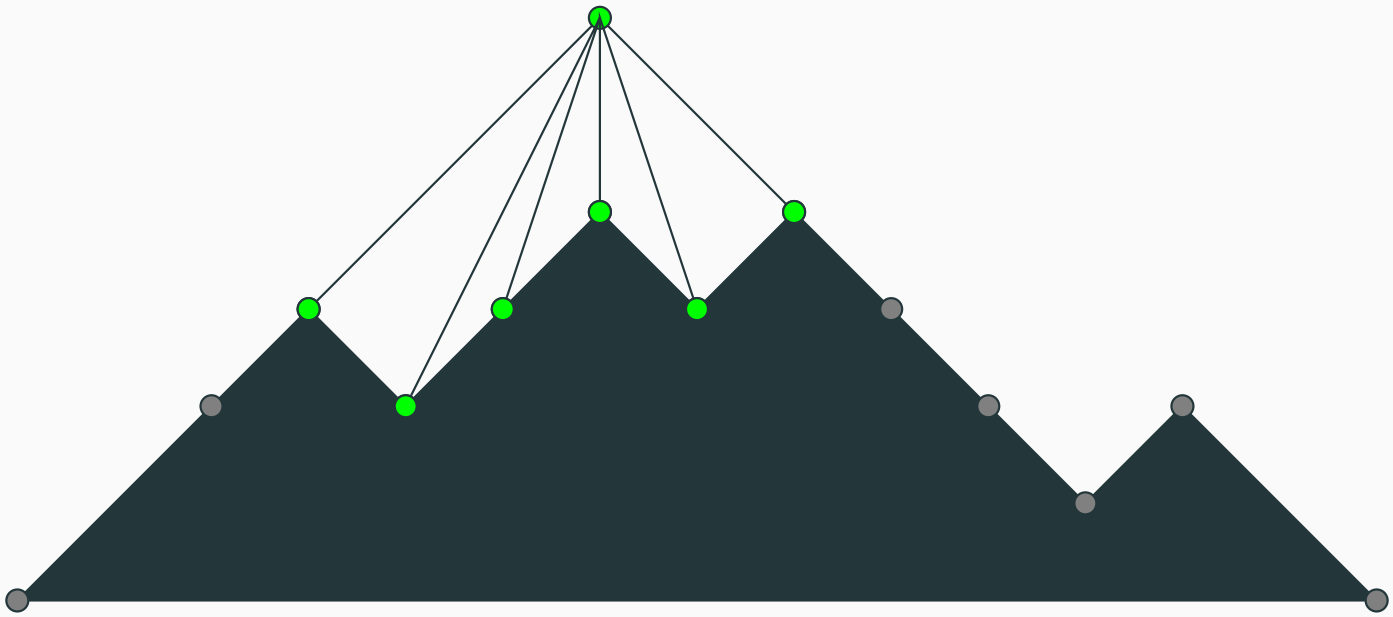
Skizze

- Außenkanten (außer v_1v_2) bilden „Kontur“ mit Steigungen ± 1



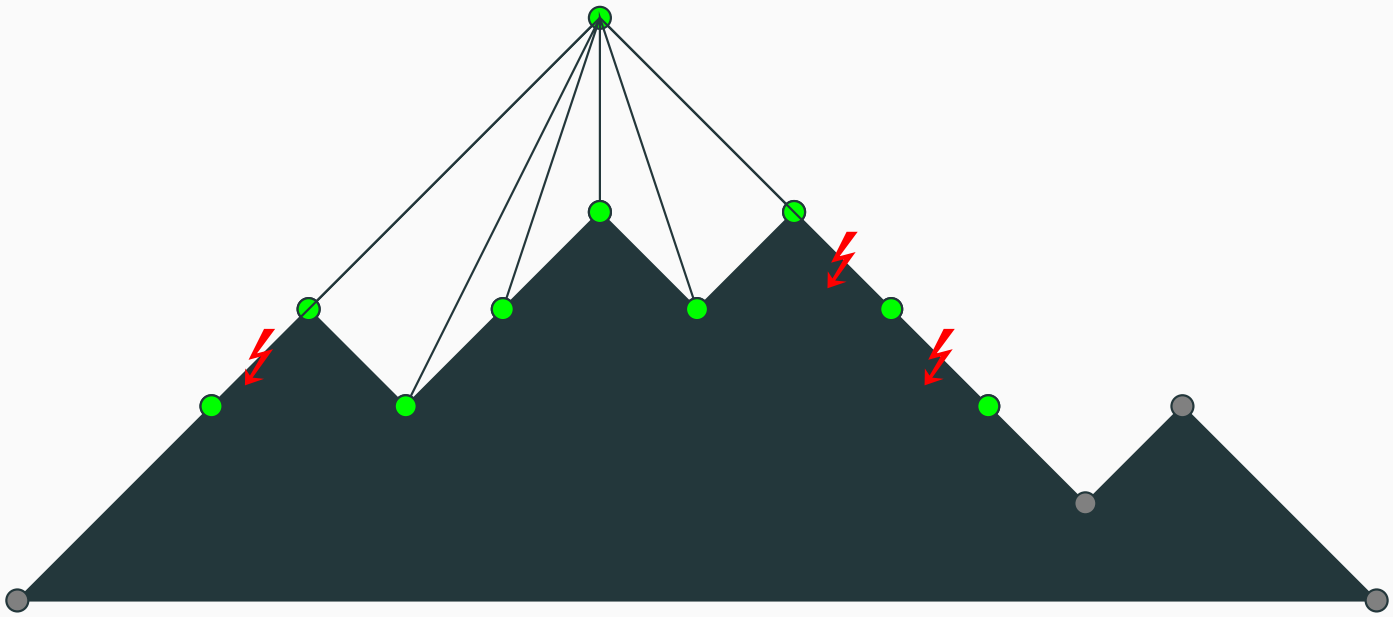
Skizze

- Außenkanten (außer v_1v_2) bilden „Kontur“ mit Steigungen ± 1
- Füge Knoten außen hinzu



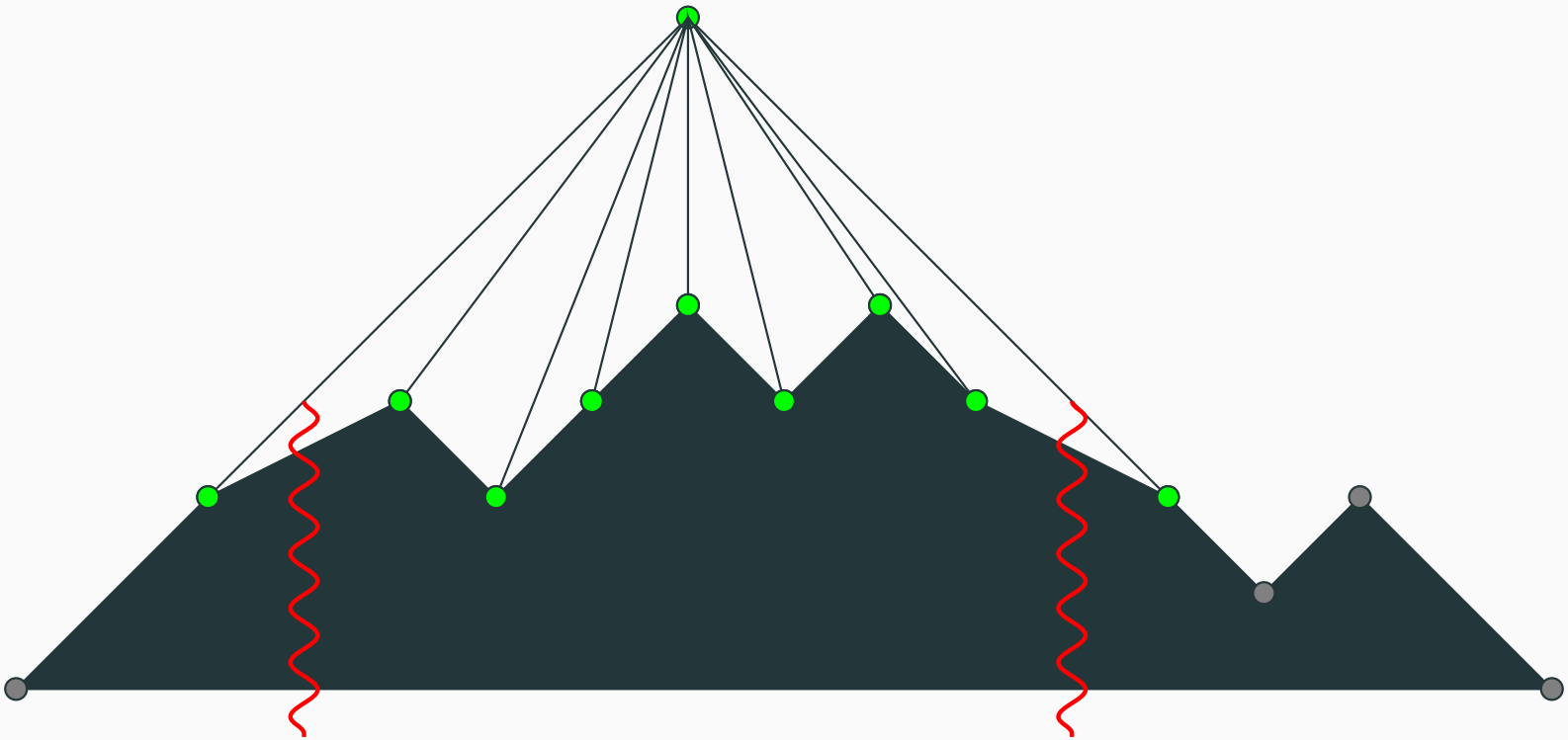
Skizze

- Außenkanten (außer v_1v_2) bilden „Kontur“ mit Steigungen ± 1
- Füge Knoten außen hinzu
- Lücken erzeugen, um Überlappen zu verhindern



Skizze

- Außenkanten (außer v_1v_2) bilden „Kontur“ mit Steigungen ± 1
- Füge Knoten außen hinzu
- Lücken erzeugen, um Überlappen zu verhindern



Triangulierte Graphen

Definition

Ein planarer Graph $G = (V, E)$ heißt *trianguliert*, wenn keine Kante ohne Kreuzung hinzugefügt werden kann.

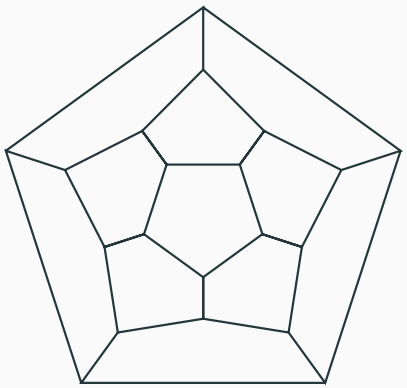
Äquivalent dazu: Jede Fläche ist ein Dreieck.

Triangulierte Graphen

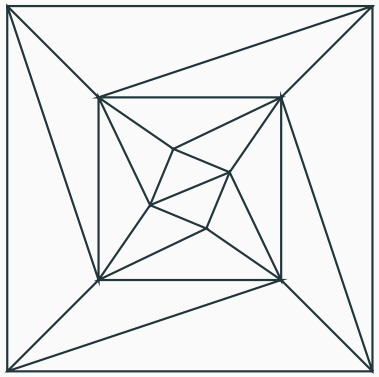
Definition

Ein planarer Graph $G = (V, E)$ heißt *trianguliert*, wenn keine Kante ohne Kreuzung hinzugefügt werden kann.

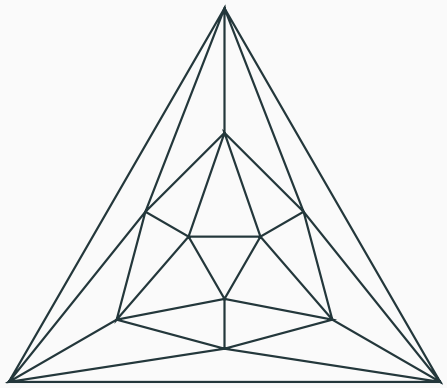
Äquivalent dazu: Jede Fläche ist ein Dreieck.



Dodekaeder: nicht trianguliert



Intern trianguliert



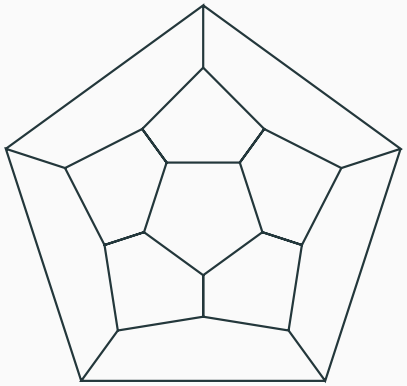
Ikosaeder: trianguliert

Triangulierte Graphen

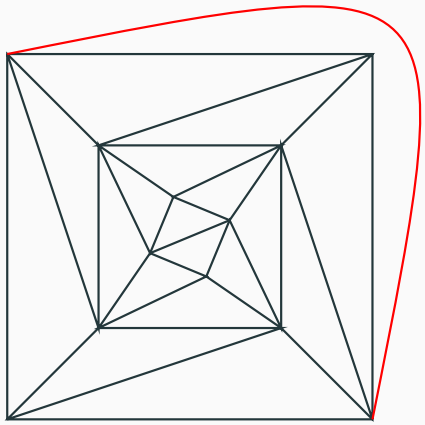
Definition

Ein planarer Graph $G = (V, E)$ heißt *trianguliert*, wenn keine Kante ohne Kreuzung hinzugefügt werden kann.

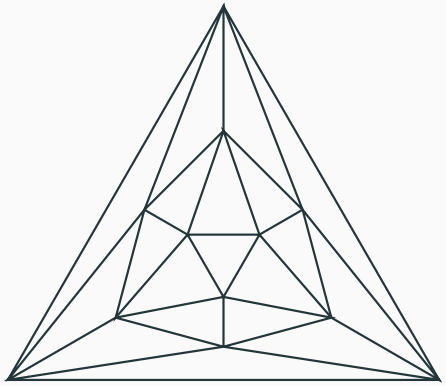
Äquivalent dazu: Jede Fläche ist ein Dreieck.



Dodekaeder: nicht trianguliert



Intern trianguliert



Ikosaeder: trianguliert

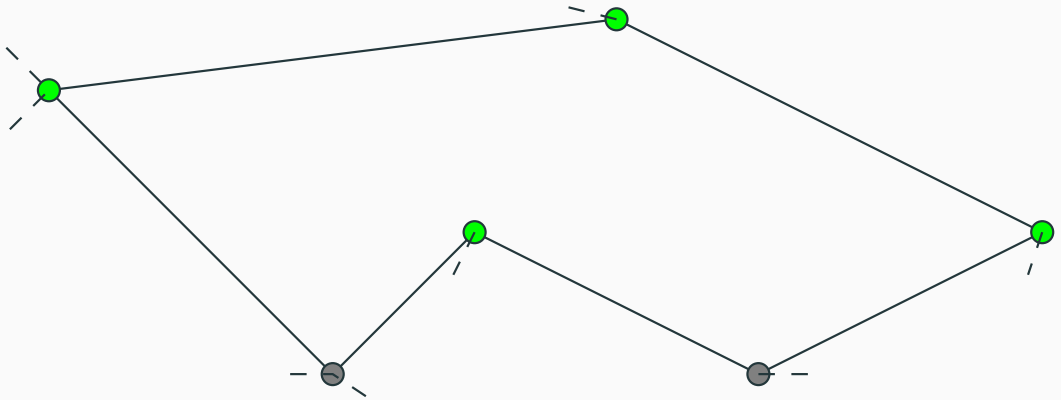
Triangulierung

Satz

Durch das Hinzufügen von Kanten kann jeder planare Graph trianguliert werden.

Beweis

Sind v_1, v_2, v_3, v_4 Knoten einer Seite, dann können v_1v_3 und v_2v_4 nicht gleichzeitig existieren und wir können eine hinzufügen.



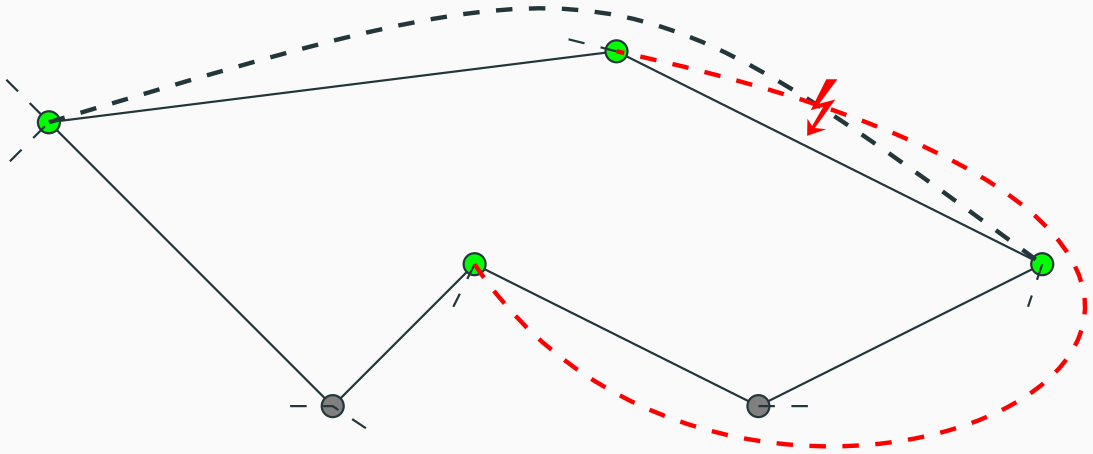
Triangulierung

Satz

Durch das Hinzufügen von Kanten kann jeder planare Graph trianguliert werden.

Beweis

Sind v_1, v_2, v_3, v_4 Knoten einer Seite, dann können v_1v_3 und v_2v_4 nicht gleichzeitig existieren und wir können eine hinzufügen.



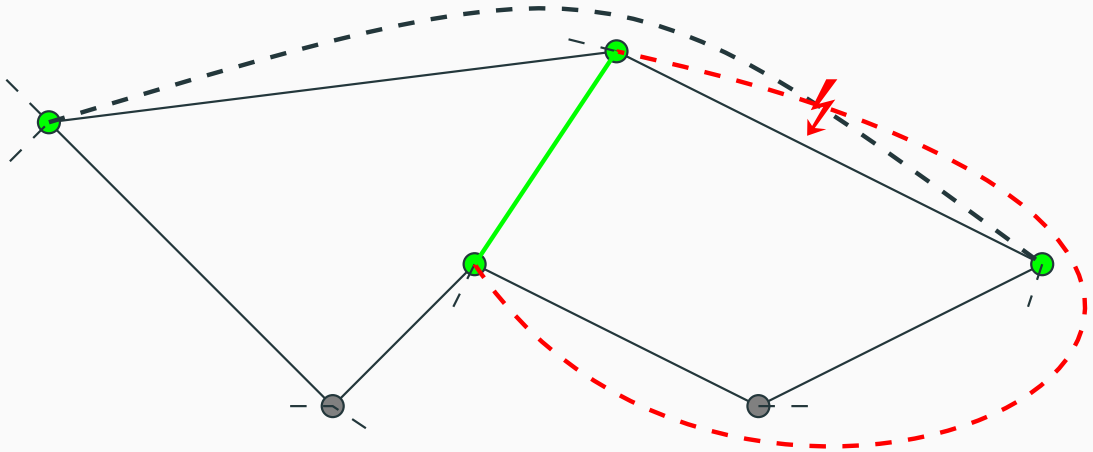
Triangulierung

Satz

Durch das Hinzufügen von Kanten kann jeder planare Graph trianguliert werden.

Beweis

Sind v_1, v_2, v_3, v_4 Knoten einer Seite, dann können v_1v_3 und v_2v_4 nicht gleichzeitig existieren und wir können eine hinzufügen.



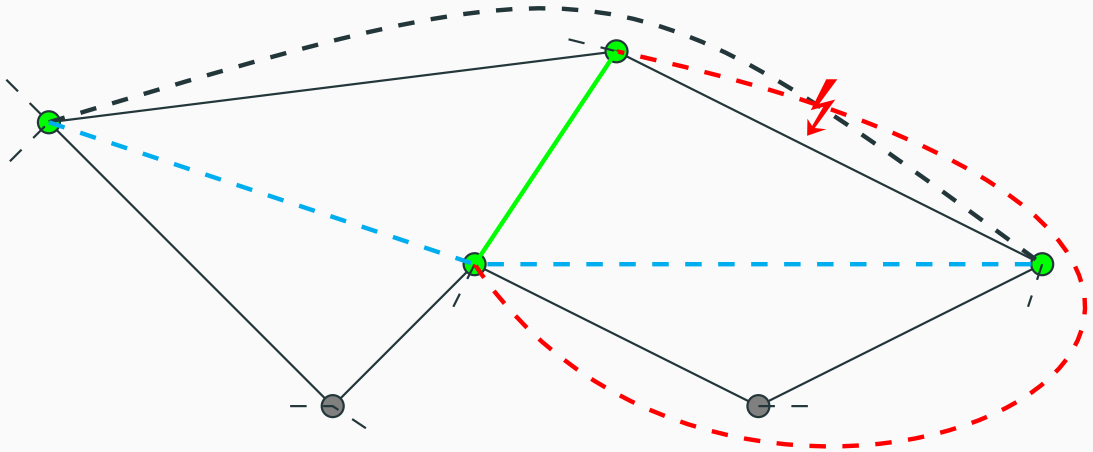
Triangulierung

Satz

Durch das Hinzufügen von Kanten kann jeder planare Graph trianguliert werden.

Beweis

Sind v_1, v_2, v_3, v_4 Knoten einer Seite, dann können v_1v_3 und v_2v_4 nicht gleichzeitig existieren und wir können eine hinzufügen.

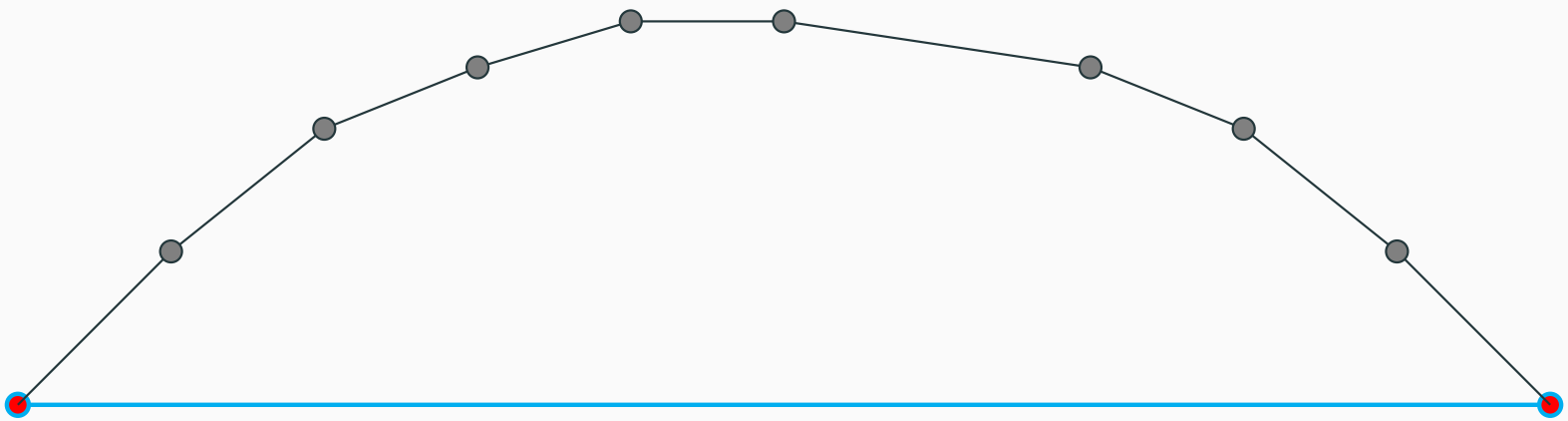


Knoten ohne Sehne

Satz

Jeder planare Graph hat einen Knoten $v \notin \{v_1, v_2\}$ auf der Kontur, an dem keine Sehne endet.

(Sehne: Kante zu Kontur-Knoten, die keine Außenkante ist)

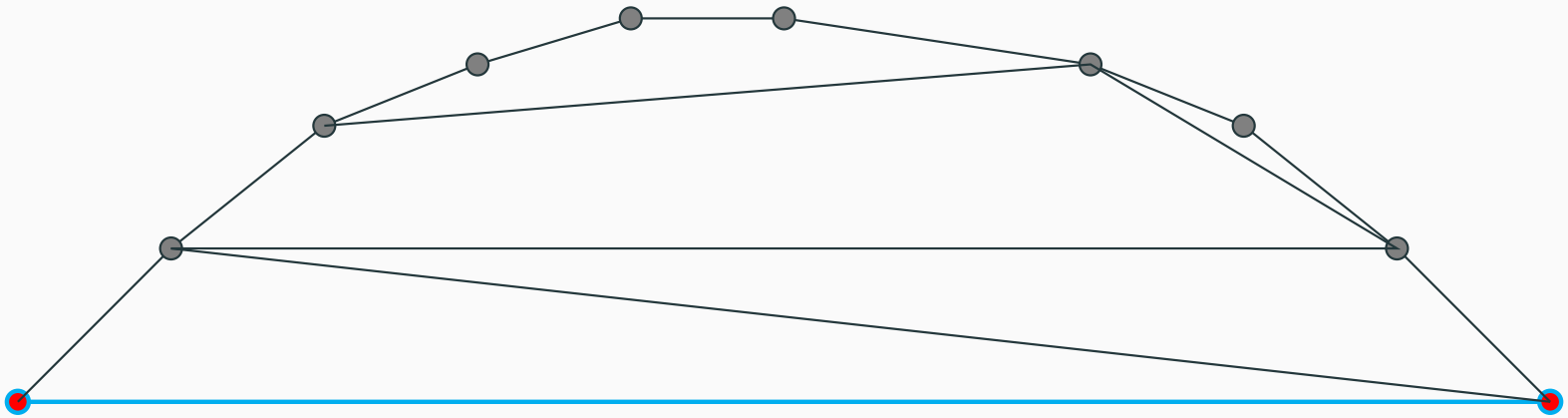


Knoten ohne Sehne

Satz

Jeder planare Graph hat einen Knoten $v \notin \{v_1, v_2\}$ auf der Kontur, an dem keine Sehne endet.

(Sehne: Kante zu Kontur-Knoten, die keine Außenkante ist)

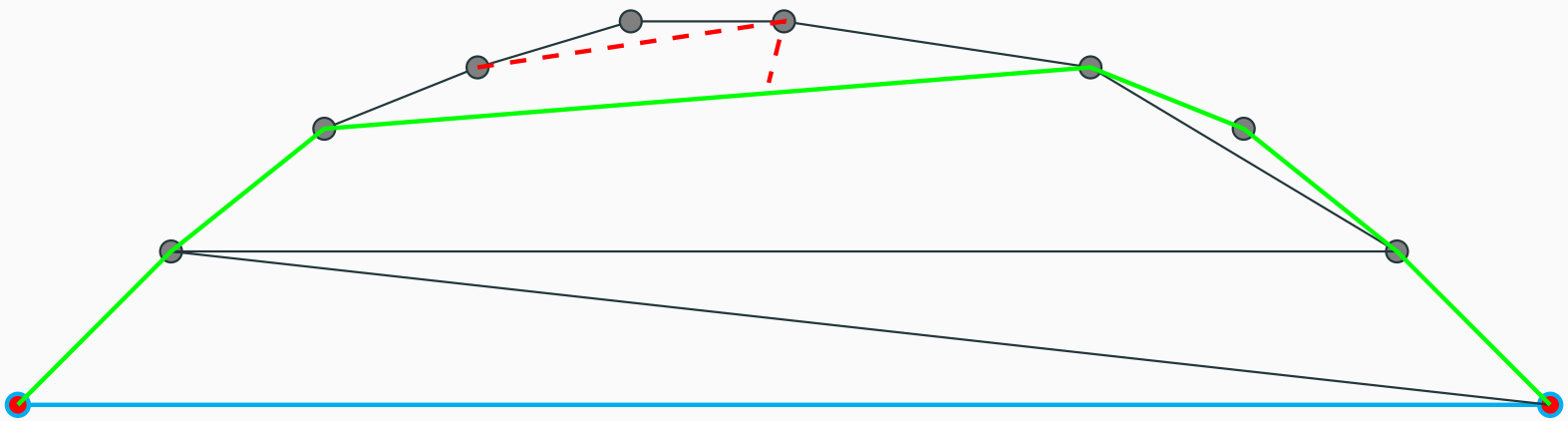


Knoten ohne Sehne

Satz

Jeder planare Graph hat einen Knoten $v \notin \{v_1, v_2\}$ auf der Kontur, an dem keine Sehne endet.

(Sehne: Kante zu Kontur-Knoten, die keine Außenkante ist)



Knoten entfernen

Sei $G = (V, E)$ planar trianguliert.

Ordne die Knoten $v_n, \ldots, v_1$ so, dass:

1. Die Außenfläche von $G_n = G$ ist (v_1, v_2, v_n);

Knoten entfernen

Sei $G = (V, E)$ planar trianguliert.

Ordne die Knoten $v_n, \ldots, v_1$ so, dass:

1. Die Außenfläche von $G_n = G$ ist (v_1, v_2, v_n);
2. $v_k \notin \{v_1, v_2\}$ ist ein Kontur-Knoten von G_k ohne Sehne;
3. G_{k-1} entsteht durch Entfernen von v_k aus G_k.

Knoten entfernen

Sei $G = (V, E)$ planar trianguliert.

Ordne die Knoten $v_n, \ldots, v_1$ so, dass:

1. Die Außenfläche von $G_n = G$ ist (v_1, v_2, v_n);
2. $v_k \notin \{v_1, v_2\}$ ist ein Kontur-Knoten von G_k ohne Sehne;
3. G_{k-1} entsteht durch Entfernen von v_k aus G_k.

Beobachtung

$G_n = G$ ist intern trianguliert, 2-zusammenhängend, und $v_1v_2 \in E$ ist Außenkante.

Rückwärts-Induktion über $k = n, \ldots, 3$.

Eigenschaften der Sortierung I

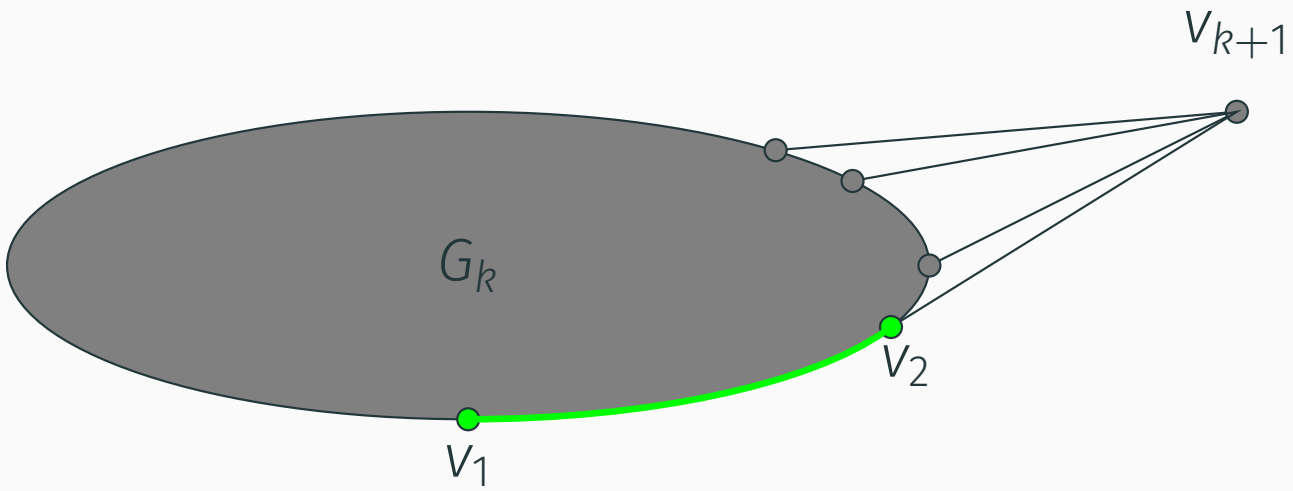
Satz

Für alle k mit $3 \leq k < n$...

liegt v_{k+1} in der Außenfläche von G_k,

ist v_1v_2 Außenkante von G_k,

und ist G_k intern trianguliert.

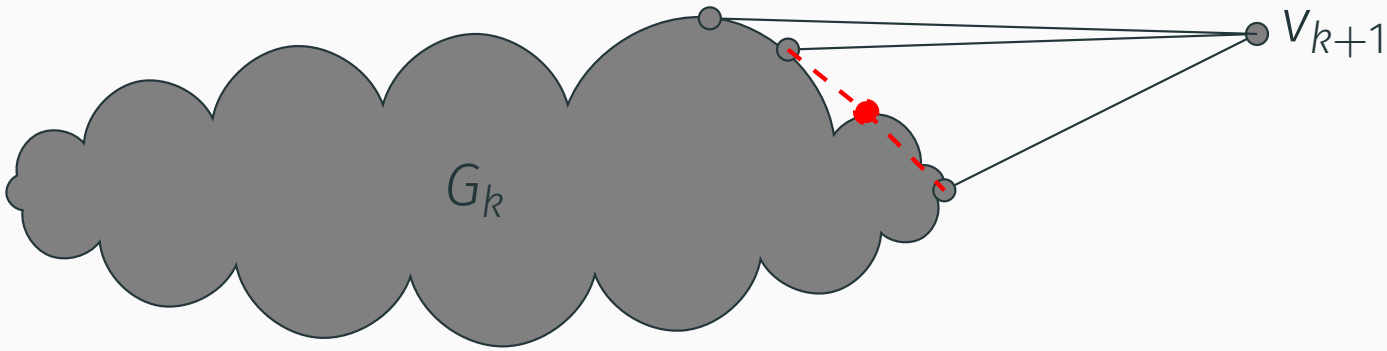


Eigenschaften der Sortierung II

Satz

Für alle k mit $3 \leq k < n$...

sind die Nachbarn von v_{k+1} in G_k aufeinanderfolgend auf der Kontur.

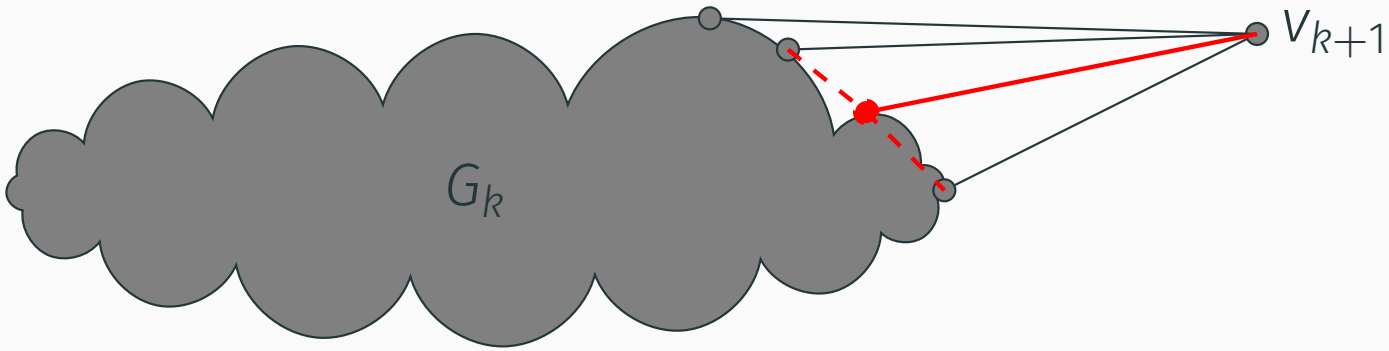


Eigenschaften der Sortierung II

Satz

Für alle k mit $3 \leq k < n$...

sind die Nachbarn von v_{k+1} in G_k aufeinanderfolgend auf der Kontur.

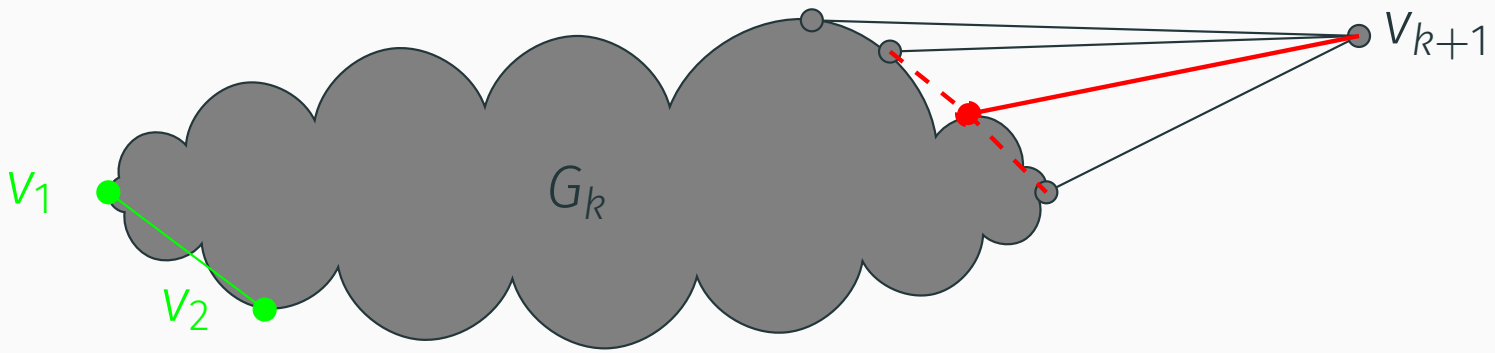


Eigenschaften der Sortierung II

Satz

Für alle k mit $3 \leq k < n$...

sind die Nachbarn von v_{k+1} in G_k aufeinanderfolgend auf der Kontur.



Eigenschaften der Sortierung III

Satz

Für alle k mit $3 \leq k < n$...

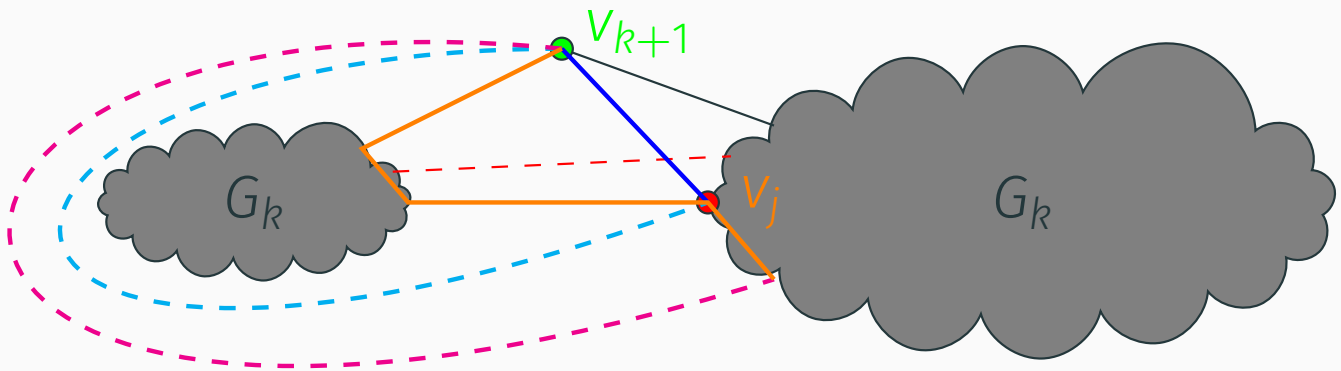
ist G_k 2-zusammenhängend.

Eigenschaften der Sortierung III

Satz

Für alle k mit $3 \leq k < n$...

ist G_k 2-zusammenhängend.



Nachbarn von v_{k+1} sind aufeinanderfolgend, also müsste jede Kante zu Trennern eine Sehne sein. Nach Wahl von v_{k+1} ist G_k 2-zusammenhängend.

Kanonische Sortierung

Definition

Eine Sortierung $v_n, \ldots, v_1$ heißt *kanonisch*, wenn (v_1, v_2, v_n) die Außenfläche von G ist, und für $3 \leq k < n$ gilt:

Kanonische Sortierung

Definition

Eine Sortierung $v_n, \ldots, v_1$ heißt *kanonisch*, wenn (v_1, v_2, v_n) die Außenfläche von G ist, und für $3 \leq k < n$ gilt:

1. $G_k = G|_{v_1,\ldots,v_k}$ ist 2-zusammenhängend und intern trianguliert,

Kanonische Sortierung

Definition

Eine Sortierung $v_n, \ldots, v_1$ heißt *kanonisch*, wenn (v_1, v_2, v_n) die Außenfläche von G ist, und für $3 \leq k < n$ gilt:

1. $G_k = G|_{v_1,\ldots,v_k}$ ist 2-zusammenhängend und intern trianguliert,
2. v_1v_2 ist Außenkante in G_k,

Kanonische Sortierung

Definition

Eine Sortierung $v_n, \ldots, v_1$ heißt *kanonisch*, wenn (v_1, v_2, v_n) die Außenfläche von G ist, und für $3 \leq k < n$ gilt:

1. $G_k = G|_{v_1,\ldots,v_k}$ ist 2-zusammenhängend und intern trianguliert,
2. v_1v_2 ist Außenkante in G_k,
3. v_{k+1} ist in der Außenfläche von G_k und seine Nachbarn sind aufeinanderfolgend auf der Kontur.

Kanonische Sortierung

Definition

Eine Sortierung $v_n, \ldots, v_1$ heißt *kanonisch*, wenn (v_1, v_2, v_n) die Außenfläche von G ist, und für $3 \leq k < n$ gilt:

1. $G_k = G|_{v_1,\ldots,v_k}$ ist 2-zusammenhängend und intern trianguliert,
2. v_1v_2 ist Außenkante in G_k,
3. v_{k+1} ist in der Außenfläche von G_k und seine Nachbarn sind aufeinanderfolgend auf der Kontur.

Korollar

Jeder triangulierte Graph hat eine kanonische Sortierung.

Wir können sie in Linearzeit berechnen (Kant 1993).

Algorithmus

Zeichnen des Graphen

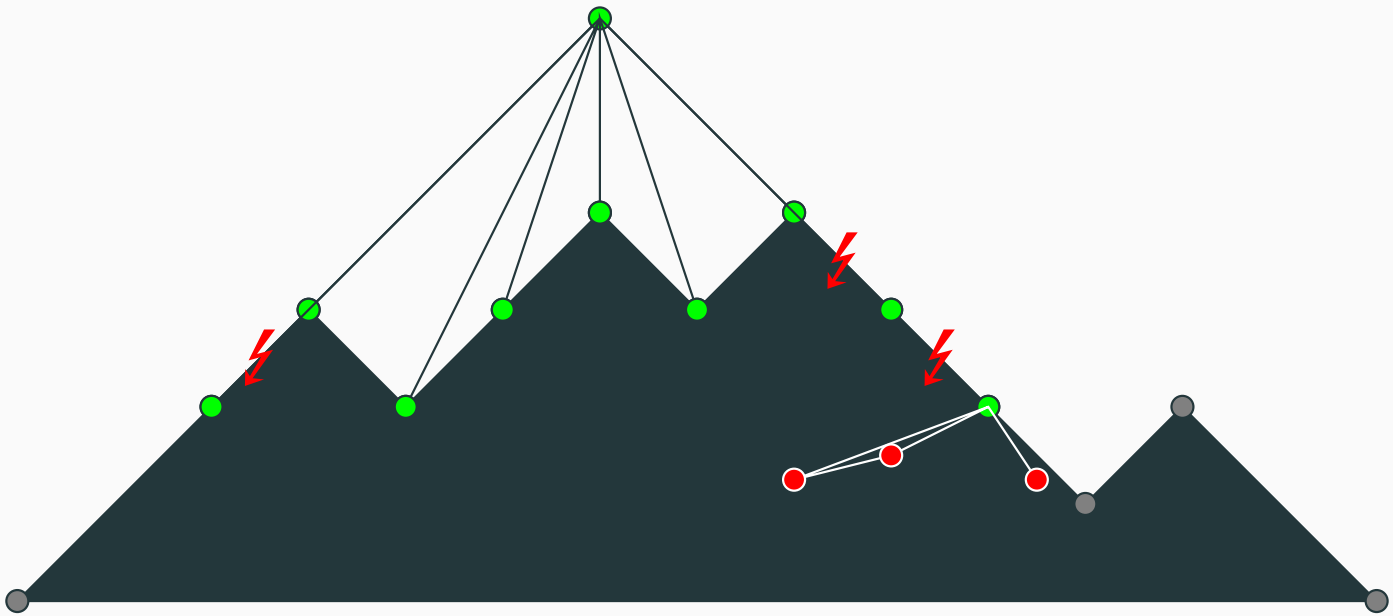
Grundoperation

Füge Knoten „über“ dem Graphen in kanonischer Sortierung ein.

Zeichnen des Graphen

Grundoperation

Füge Knoten „über“ dem Graphen in kanonischer Sortierung ein.

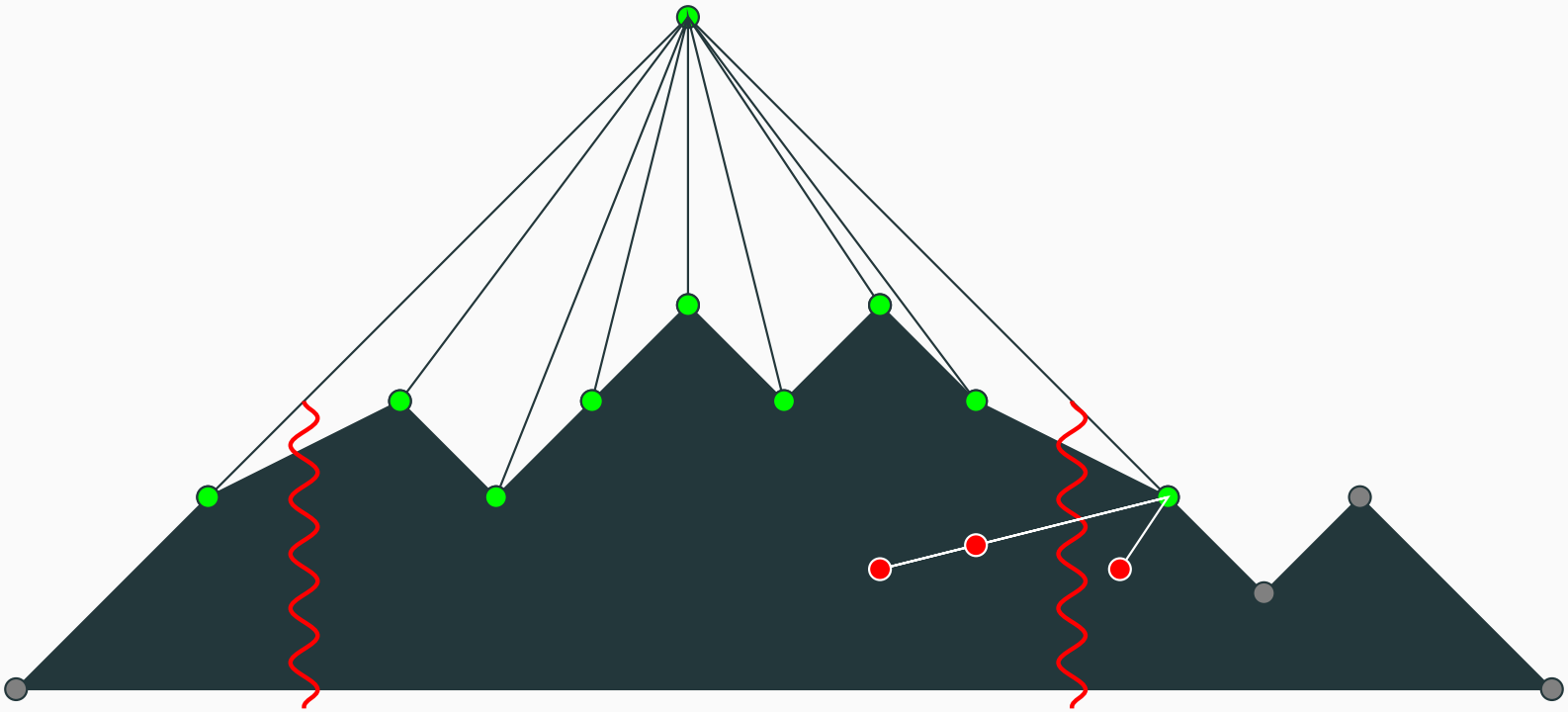


Zeichnen des Graphen

Grundoperation

Füge Knoten „über“ dem Graphen in kanonischer Sortierung ein.

Erzeuge Lücken neben dem linken und rechten Nachbarn.



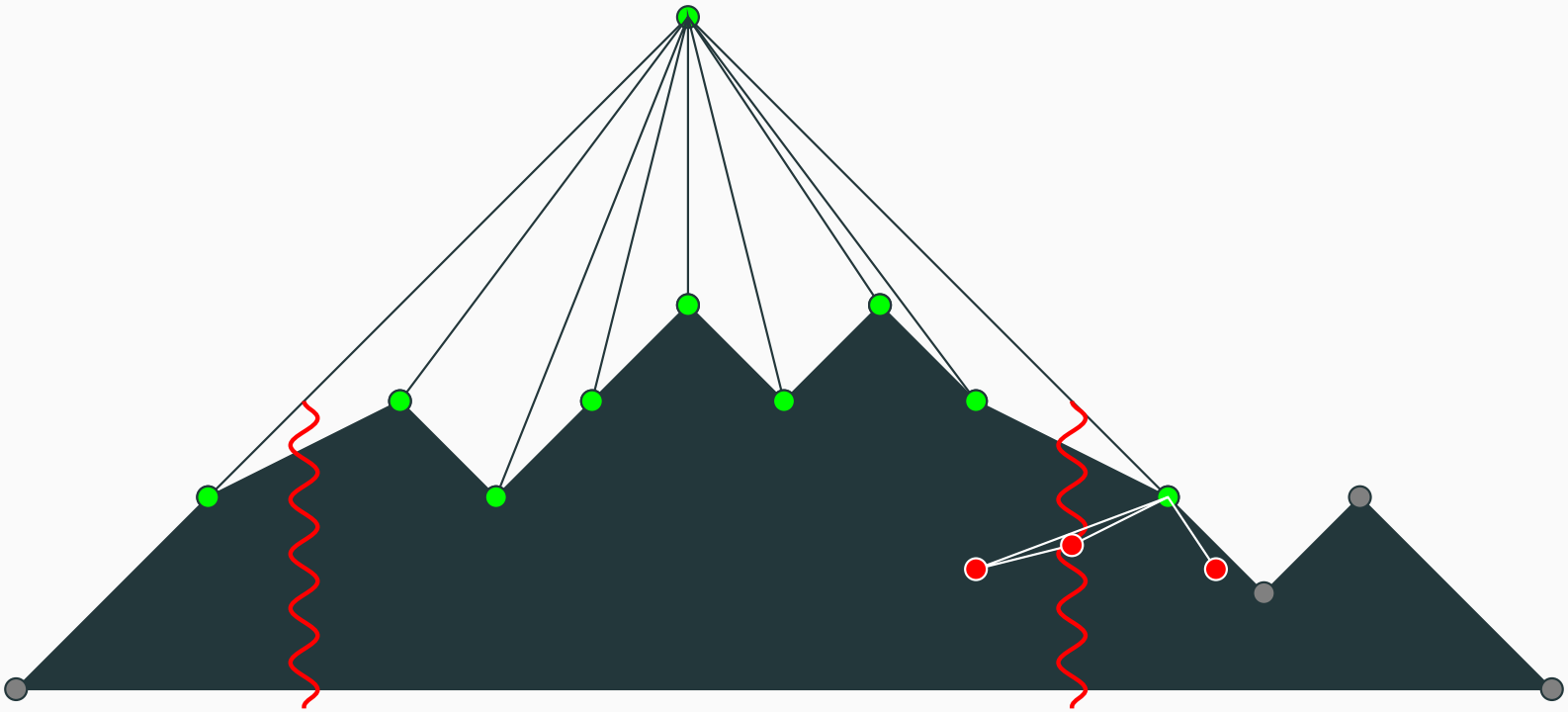
Zeichnen des Graphen

Grundoperation

Füge Knoten „über“ dem Graphen in kanonischer Sortierung ein.

Erzeuge Lücken neben dem linken und rechten Nachbarn.

Speichere, welche Knoten zusammen bewegt werden müssen.



Zeichnen des Graphen

Grundoperation

Füge Knoten „über“ dem Graphen in kanonischer Sortierung ein.

Erzeuge Lücken neben dem linken und rechten Nachbarn.

Speichere, welche Knoten zusammen bewegt werden müssen.

Definition

$L(v)$ sind die mit v bewegten Knoten.

Wir erhalten es durch $L(v) = \{v\} \cup \bigcup_{i=p+1}^{q-1} L(w_i)$ wobei $w_{p+1}, \ldots, w_{q-1}$ die von v verdeckten Knoten sind.

Invarianten

In G_k:

- $v_1 = (0, 0)$; $v_2 = (2k - 4, 0)$.

Invarianten

In G_k:

- $v_1 = (0, 0)$; $v_2 = (2k - 4, 0)$.
- Kontur-Knoten $v_1 = w_1, w_2 \ldots, w_m = v_2$ haben aufsteigende x-Koordinaten.
- Jede Kante der Kontur hat Steigung $+1$ or -1.

Invarianten

In G_k:

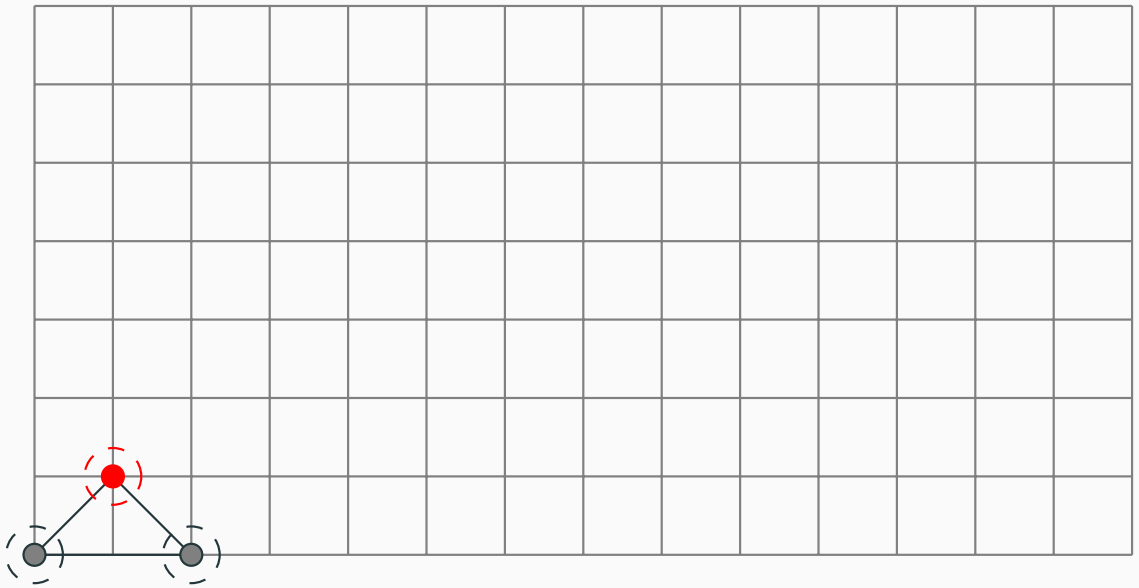
- $v_1 = (0, 0)$; $v_2 = (2k - 4, 0)$.
- Kontur-Knoten $v_1 = w_1, w_2 \ldots, w_m = v_2$ haben aufsteigende x-Koordinaten.
- Jede Kante der Kontur hat Steigung $+1$ or -1.

G_n ist ein rechtwinkliges Dreieck, also auf einem $(2n - 4) \times (n - 2)$-Gitter.

Die Mengen $L(w_i)$

Beobachtung

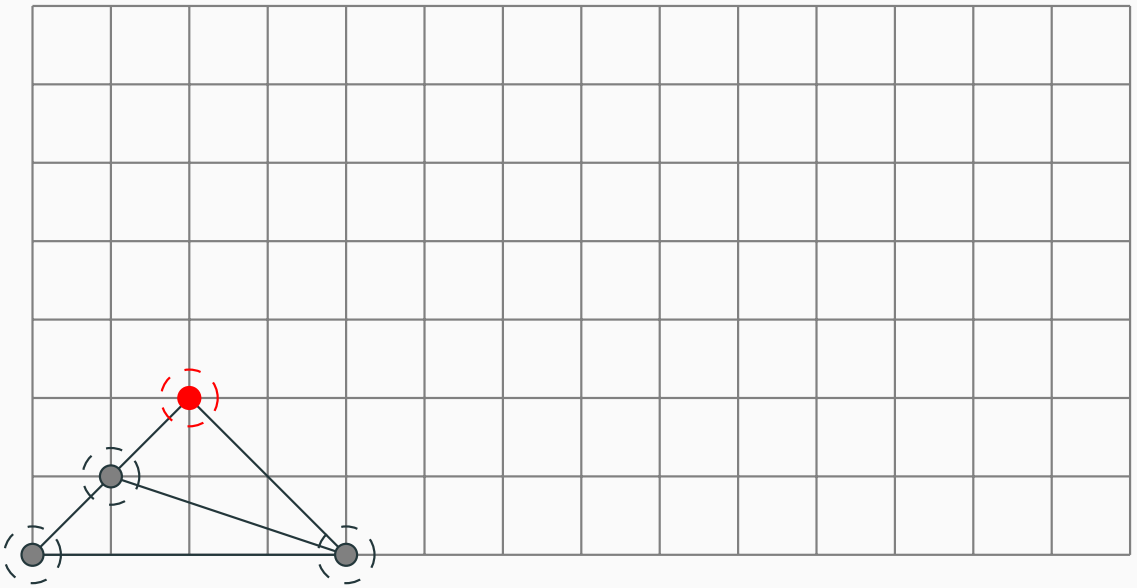
$L(w_i)$ der Kontur-Knoten von G_k sind eine Partition von $v_1, \ldots, v_k$.
Jedes $L(w_i)$ ist ein Baum mit Wurzel w_i.



Die Mengen $L(w_i)$

Beobachtung

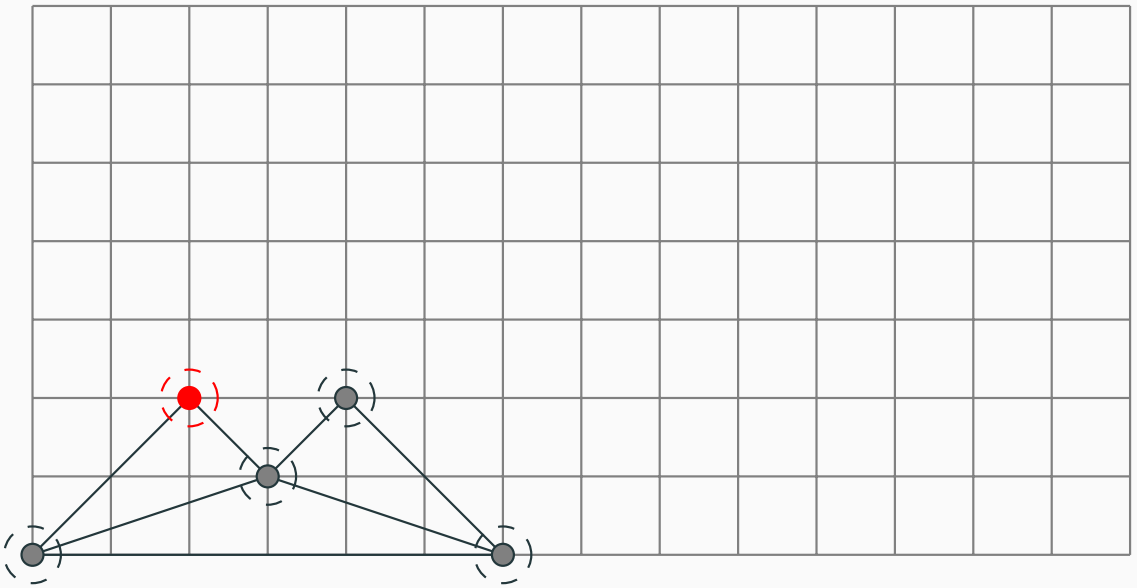
$L(w_i)$ der Kontur-Knoten von G_k sind eine Partition von $v_1, \ldots, v_k$.
Jedes $L(w_i)$ ist ein Baum mit Wurzel w_i.



Die Mengen $L(w_i)$

Beobachtung

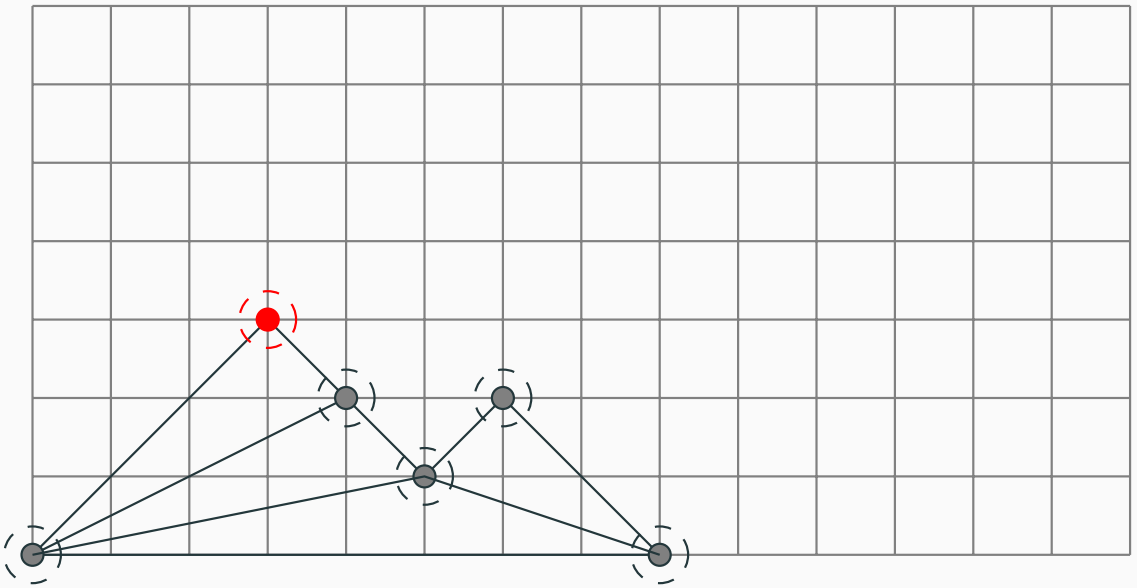
$L(w_i)$ der Kontur-Knoten von G_k sind eine Partition von $v_1, \ldots, v_k$.
Jedes $L(w_i)$ ist ein Baum mit Wurzel w_i.



Die Mengen $L(w_i)$

Beobachtung

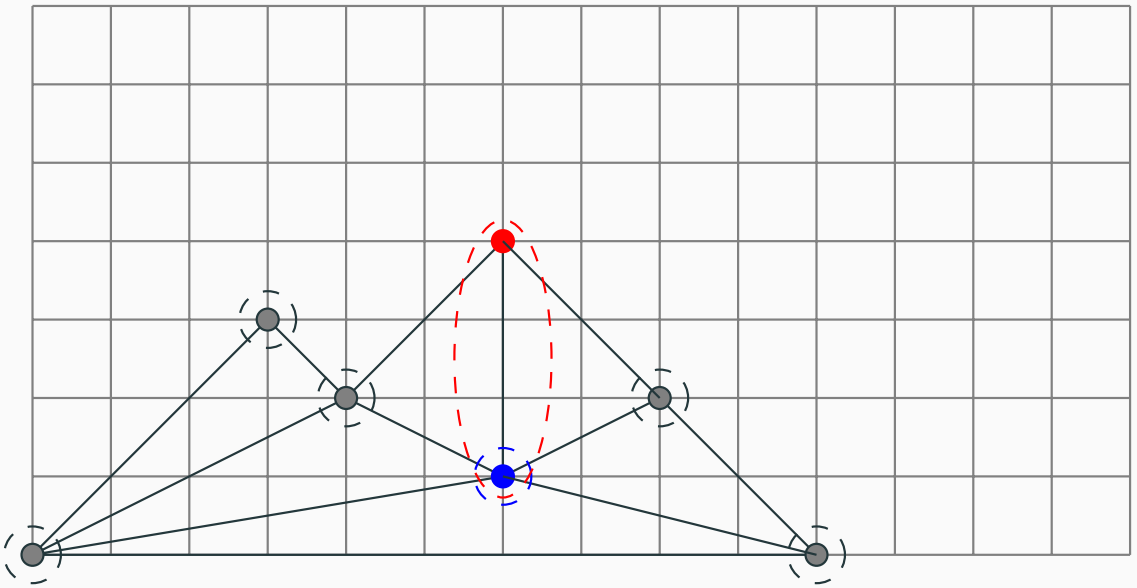
$L(w_i)$ der Kontur-Knoten von G_k sind eine Partition von $v_1, \ldots, v_k$.
Jedes $L(w_i)$ ist ein Baum mit Wurzel w_i.

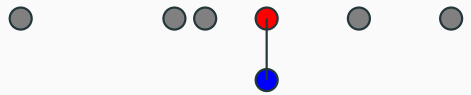


Die Mengen $L(w_i)$

Beobachtung

$L(w_i)$ der Kontur-Knoten von G_k sind eine Partition von $v_1, \ldots, v_k$.
Jedes $L(w_i)$ ist ein Baum mit Wurzel w_i.

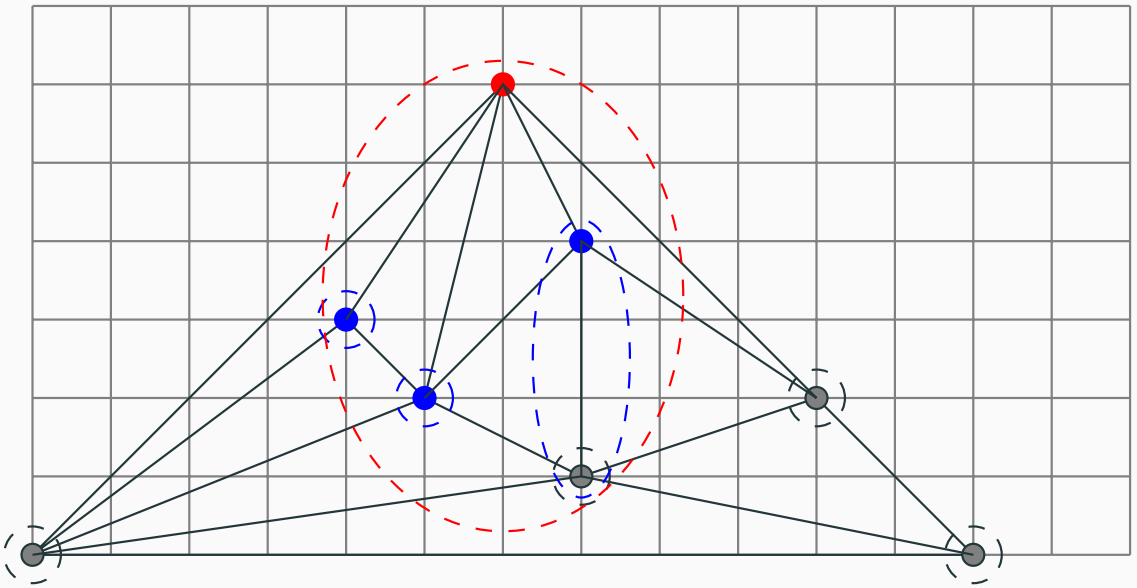


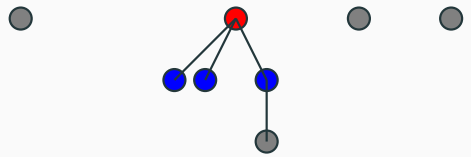


Die Mengen $L(w_i)$

Beobachtung

$L(w_i)$ der Kontur-Knoten von G_k sind eine Partition von $v_1, \ldots, v_k$.
Jedes $L(w_i)$ ist ein Baum mit Wurzel w_i.

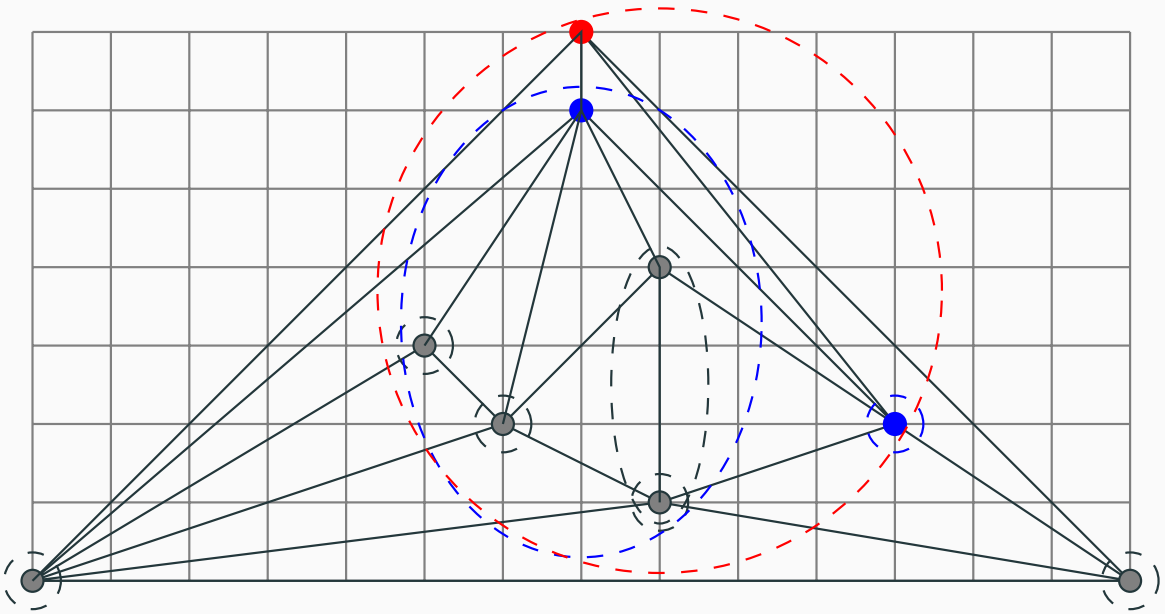


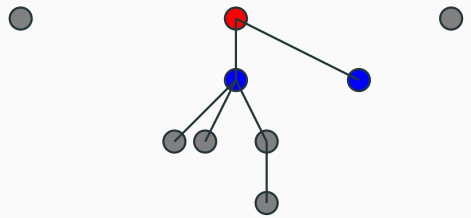


Die Mengen $L(w_i)$

Beobachtung

$L(w_i)$ der Kontur-Knoten von G_k sind eine Partition von $v_1, \ldots, v_k$.
Jedes $L(w_i)$ ist ein Baum mit Wurzel w_i.





Gesamter Algorithmus

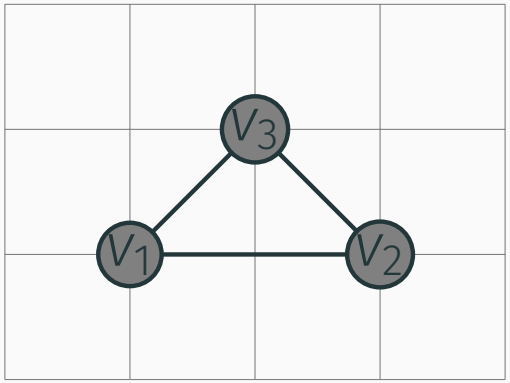
procedure ZEICHNE(G)

// $v_1, \ldots, v_n$: kanonische Sortierung von G

$v_1 = (0,0), v_2 = (2,0), v_3 = (1,1)$

$L(v_1) = \{v_1\}, L(v_2) = \{v_2\}, L(v_3) = \{v_3\}$

for $3 \leq k < n$ **do**



end for

end procedure

Gesamter Algorithmus

procedure ZEICHNE(G)

// $v_1, \ldots, v_n$: kanonische Sortierung von G

$v_1 = (0,0), v_2 = (2,0), v_3 = (1,1)$

$L(v_1) = \{v_1\}, L(v_2) = \{v_2\}, L(v_3) = \{v_3\}$

for $3 \leq k < n$ **do**

// $w_1, \ldots, w_m$: Kontur von G_k

// $w_p, \ldots, w_q$: Nachbarn von v_{k+1} in G_k

end for

end procedure

Gesamter Algorithmus

procedure ZEICHNE(G)

// $v_1, \ldots, v_n$: kanonische Sortierung von G

$v_1 = (0,0), v_2 = (2,0), v_3 = (1,1)$

$L(v_1) = \{v_1\}, L(v_2) = \{v_2\}, L(v_3) = \{v_3\}$

for $3 \leq k < n$ **do**

// $w_1, \ldots, w_m$: Kontur von G_k

// $w_p, \ldots, w_q$: Nachbarn von v_{k+1} in G_k

Verschiebe $\bigcup_{i=p+1}^{q-1} L(w_i)$ um 1

Verschiebe $\bigcup_{i=q}^{m} L(w_i)$ um 2

end for

end procedure

Gesamter Algorithmus

procedure ZEICHNE(G)

// $v_1, \ldots, v_n$: kanonische Sortierung von G

$v_1 = (0,0), v_2 = (2,0), v_3 = (1,1)$

$L(v_1) = \{v_1\}, L(v_2) = \{v_2\}, L(v_3) = \{v_3\}$

for $3 \leq k < n$ **do**

// $w_1, \ldots, w_m$: Kontur von G_k

// $w_p, \ldots, w_q$: Nachbarn von v_{k+1} in G_k

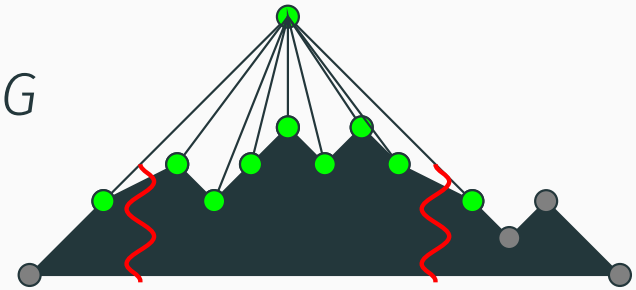
Verschiebe $\bigcup_{i=p+1}^{q-1} L(w_i)$ um 1

Verschiebe $\bigcup_{i=q}^{m} L(w_i)$ um 2

Füge v_{k+1} so ein, dass $w_p v_{k+1}$, $v_{k+1} w_q$ Steigung $+1$, -1 haben

end for

end procedure



Gesamter Algorithmus

procedure ZEICHNE(G)

// $v_1, \ldots, v_n$: kanonische Sortierung von G

$v_1 = (0,0), v_2 = (2,0), v_3 = (1,1)$

$L(v_1) = \{v_1\}, L(v_2) = \{v_2\}, L(v_3) = \{v_3\}$

for $3 \leq k < n$ **do**

// $w_1, \ldots, w_m$: Kontur von G_k

// $w_p, \ldots, w_q$: Nachbarn von v_{k+1} in G_k

Verschiebe $\bigcup_{i=p+1}^{q-1} L(w_i)$ um 1

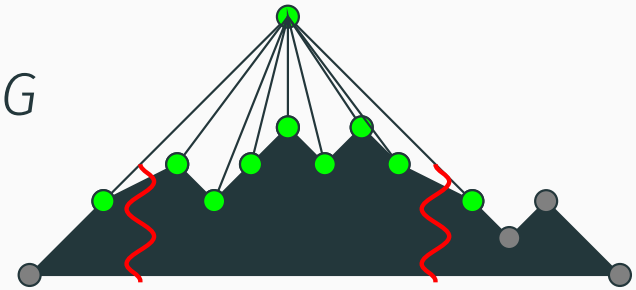
Verschiebe $\bigcup_{i=q}^{m} L(w_i)$ um 2

Füge v_{k+1} so ein, dass $w_p v_{k+1}$, $v_{k+1} w_q$ Steigung $+1$, -1 haben

Setze $L(v_{k+1}) = \{v_{k+1}\} \cup \bigcup_{i=p+1}^{q-1} L(w_i)$

end for

end procedure



Gesamter Algorithmus

procedure ZEICHNE(G)

// $v_1, \ldots, v_n$: kanonische Sortierung von G

$v_1 = (0,0), v_2 = (2,0), v_3 = (1,1)$

$L(v_1) = \{v_1\}, L(v_2) = \{v_2\}, L(v_3) = \{v_3\}$

for $3 \leq k < n$ **do**

// $w_1, \ldots, w_m$: Kontur von G_k

// $w_p, \ldots, w_q$: Nachbarn von v_{k+1} in G_k

Verschiebe $\bigcup_{i=p+1}^{q-1} L(w_i)$ um 1

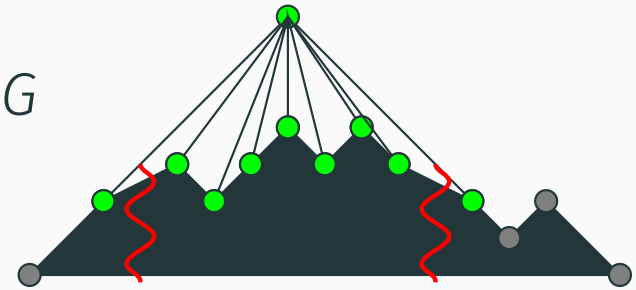
Verschiebe $\bigcup_{i=q}^{m} L(w_i)$ um 2

Füge v_{k+1} so ein, dass $w_p v_{k+1}$, $v_{k+1} w_q$ Steigung $+1$, -1 haben

Setze $L(v_{k+1}) = \{v_{k+1}\} \cup \bigcup_{i=p+1}^{q-1} L(w_i)$

end for

end procedure



Koordinaten von v_{k+1}

Satz

v_{k+1} wird an einem Gitterpunkt eingebettet.

Man kann zeigen, dass jeder Kontur-Knoten Koordinaten mit gerader Summe hat.

Korrektheit I

Satz (de Fraysseix *u. a.* 1990)

G_k sei mit dem Algorithmus eingebettet und habe Kontur $w_1, \ldots, w_m$; $0 \leq \rho_1 \leq \cdots \leq \rho_m$ seien ganze Zahlen.

Korrektheit I

Satz (de Fraysseix *u. a.* 1990)

G_k sei mit dem Algorithmus eingebettet und habe Kontur $w_1, \ldots, w_m$; $0 \leq \rho_1 \leq \cdots \leq \rho_m$ seien ganze Zahlen.

Dann bleibt G_k planar, wenn man $L(w_i)$ um ρ_i verschiebt.

Korrektheit I

Satz (de Fraysseix *u. a.* 1990)

G_k sei mit dem Algorithmus eingebettet und habe Kontur $w_1, \ldots, w_m$; $0 \leq \rho_1 \leq \cdots \leq \rho_m$ seien ganze Zahlen.

Dann bleibt G_k planar, wenn man $L(w_i)$ um ρ_i verschiebt.

Beweis (Induktion; Skizze)

Verschiebe Kontur $w_1, \ldots, w_p, \quad v_{k+1}, \quad w_q, \ldots, w_m$ von G_{k+1} um $\rho_1, \ldots, \rho_p, \quad \rho, \quad \rho_q, \ldots, \rho_m$.

Die Kontur-Knoten von G_k werden um

$$\rho_1, \ldots, \rho_p, \quad \underbrace{\rho + 1, \ldots, \rho + 1,}_{\text{verdeckte Knoten in } L(v_{k+1})} \quad \rho_q + 2, \ldots, \rho_m + 2 \text{ verschoben.}$$

Korrektheit I

Satz (de Fraysseix *u. a.* 1990)

G_k sei mit dem Algorithmus eingebettet und habe Kontur $w_1, \ldots, w_m$; $0 \leq \rho_1 \leq \cdots \leq \rho_m$ seien ganze Zahlen.

Dann bleibt G_k planar, wenn man $L(w_i)$ um ρ_i verschiebt.

Beweis (Induktion; Skizze)

Verschiebe Kontur $w_1, \ldots, w_p, \quad v_{k+1}, \quad w_q, \ldots, w_m$ von G_{k+1} um $\rho_1, \ldots, \rho_p, \quad \rho, \quad \rho_q, \ldots, \rho_m$.

Die Kontur-Knoten von G_k werden um

$$\rho_1, \ldots, \rho_p, \quad \underbrace{\rho + 1, \ldots, \rho + 1,}_{\text{verdeckte Knoten in } L(v_{k+1})} \quad \rho_q + 2, \ldots, \rho_m + 2 \text{ verschoben.}$$

Da v_{k+1} sich genau wie $w_{p+1}, \ldots, w_{q-1}$ verschiebt, bleibt G_{k+1} planar.

Korrektheit II

Korollar

Alle $G_3, \ldots, G_n = G$ sind planar eingebettet.

Beweis

Korrektheit II

Korollar

Alle $G_3, \ldots, G_n = G$ sind planar eingebettet.

Beweis

Verschieben der Knoten in G_k erzeugt keine neuen Kreuzungen.

Korrektheit II

Korollar

Alle $G_3, \ldots, G_n = G$ sind planar eingebettet.

Beweis

Verschieben der Knoten in G_k erzeugt keine neuen Kreuzungen.

Einfügen von v_{k+1} kann keine Kreuzung erzeugen, da alle neuen Kanten zwischen w_p und w_q steiler sind als Steigung ± 1.

Verbesserung der Laufzeit

Laufzeit

Pseudocode: $\mathcal{O}(n^2)$. Geht es schneller?

Laufzeit

Pseudocode: $\mathcal{O}(n^2)$. Geht es schneller?

- Planares Einbetten: $\mathcal{O}(n)$ (vgl. Hopcroft & Tarjan 1974)

Laufzeit

Pseudocode: $\mathcal{O}(n^2)$. Geht es schneller?

- Planares Einbetten: $\mathcal{O}(n)$ (vgl. Hopcroft & Tarjan 1974)
- Eingebetteten Graph triangulieren und kanonische Sortierung berechnen: $\mathcal{O}(n)$ (vgl. Kant 1993)

Laufzeit

Pseudocode: $\mathcal{O}(n^2)$. Geht es schneller?

- Planares Einbetten: $\mathcal{O}(n)$ (vgl. Hopcroft & Tarjan 1974)
- Eingebetteten Graph triangulieren und kanonische Sortierung berechnen: $\mathcal{O}(n)$ (vgl. Kant 1993)

Idee

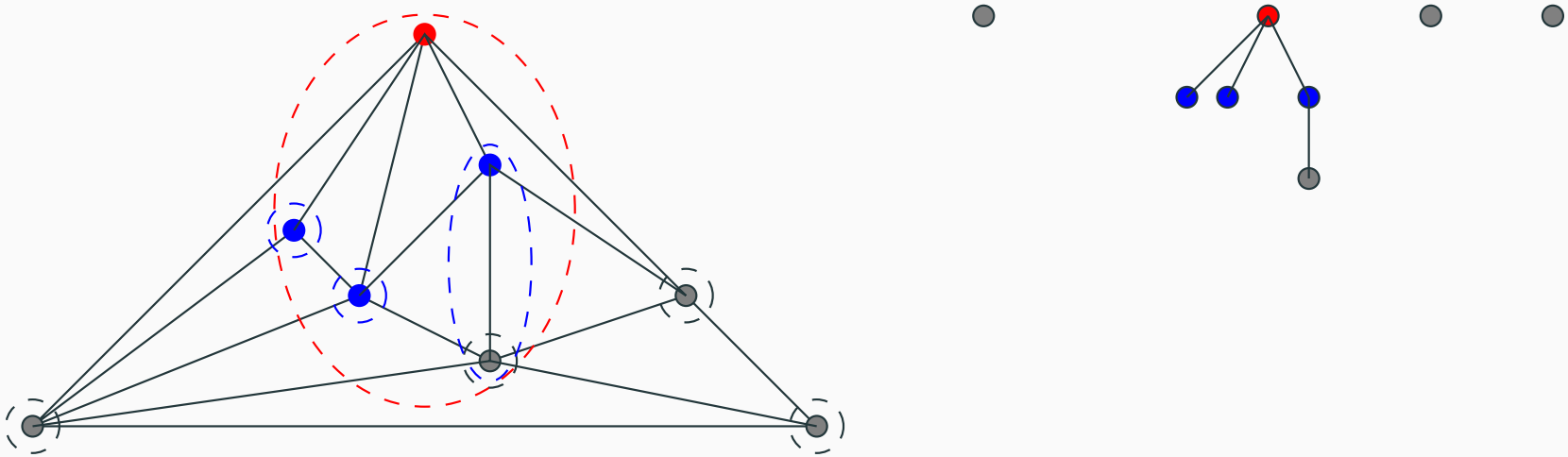
Wir müssen nur die x-Koordinaten effizient berechnen.

Berechne Koordinaten nicht jedes Mal neu:

Speichere nur den Abstand von einem Referenzpunkt.

Wald der $L(w_i)$

$L(w_i)$ der Kontur-Knoten in G_k bilden einen Wald aus Bäumen mit Wurzel w_i.



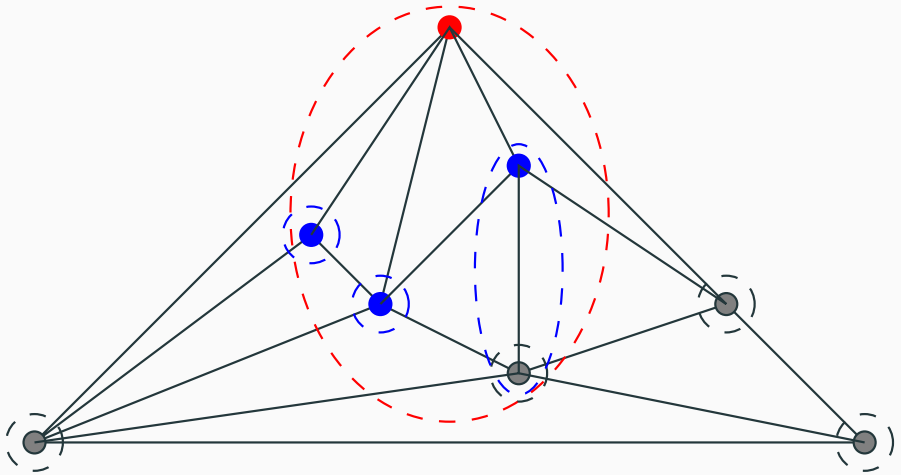
Wald der $L(w_i)$

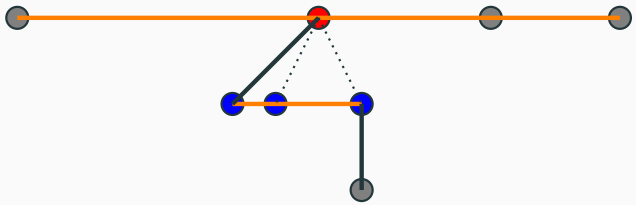
$L(w_i)$ der Kontur-Knoten in G_k bilden einen Wald aus Bäumen mit Wurzel w_i.

Speichere Knoten in einem Binärbaum.

Ein Knoten v enthält den x-Abstand $\Delta x(v)$ zum Elternknoten p.

p ist der Knoten, der v verdeckt, oder der linke „Nachbar“ von v.





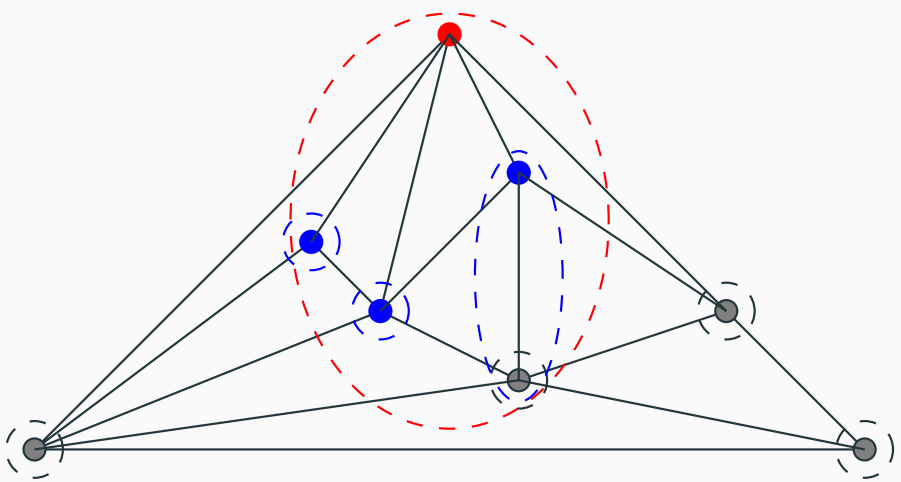
Wald der $L(w_i)$

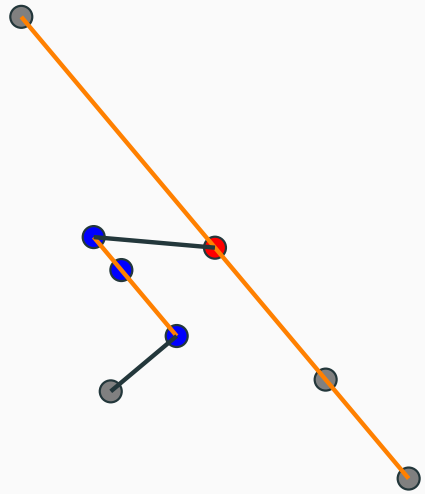
$L(w_i)$ der Kontur-Knoten in G_k bilden einen Wald aus Bäumen mit Wurzel w_i.

Speichere Knoten in einem Binärbaum.

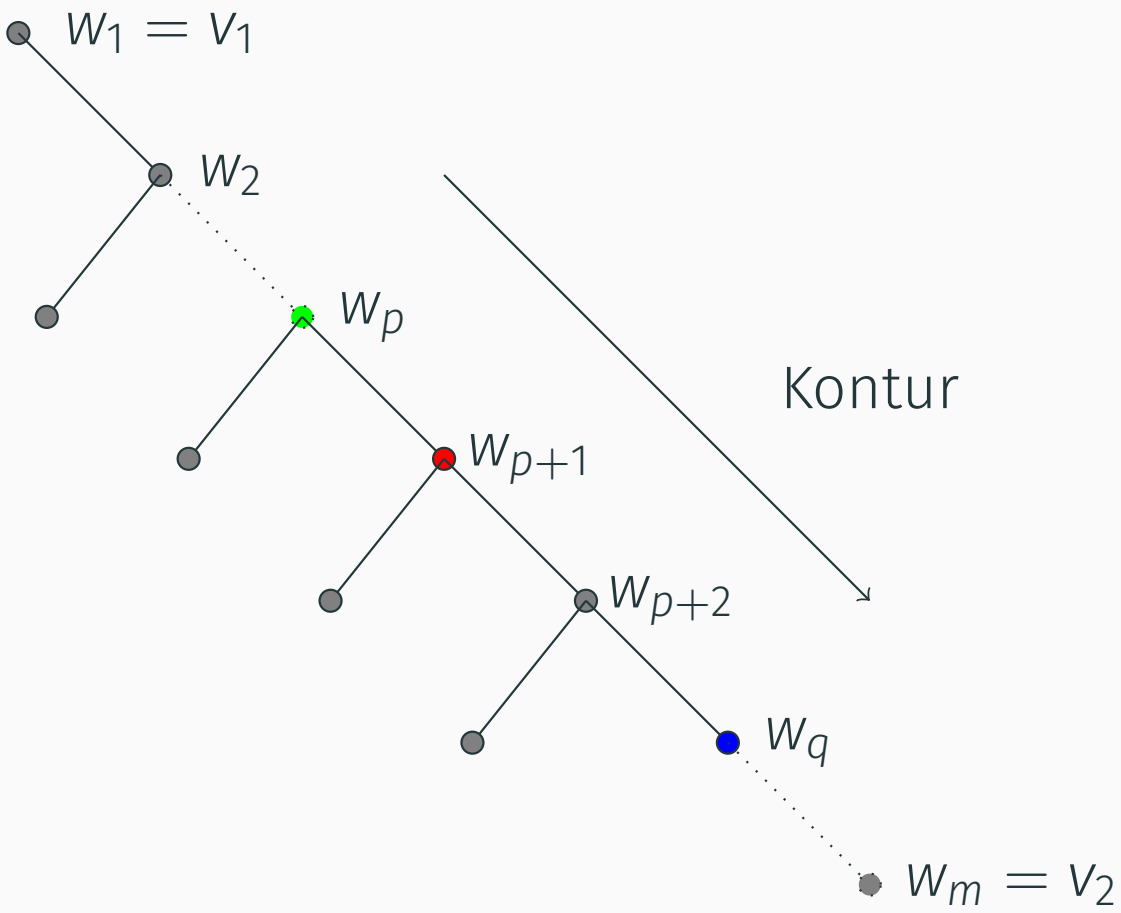
Ein Knoten v enthält den x-Abstand $\Delta x(v)$ zum Elternknoten p.

p ist der Knoten, der v verdeckt, oder der linke „Nachbar“ von v.

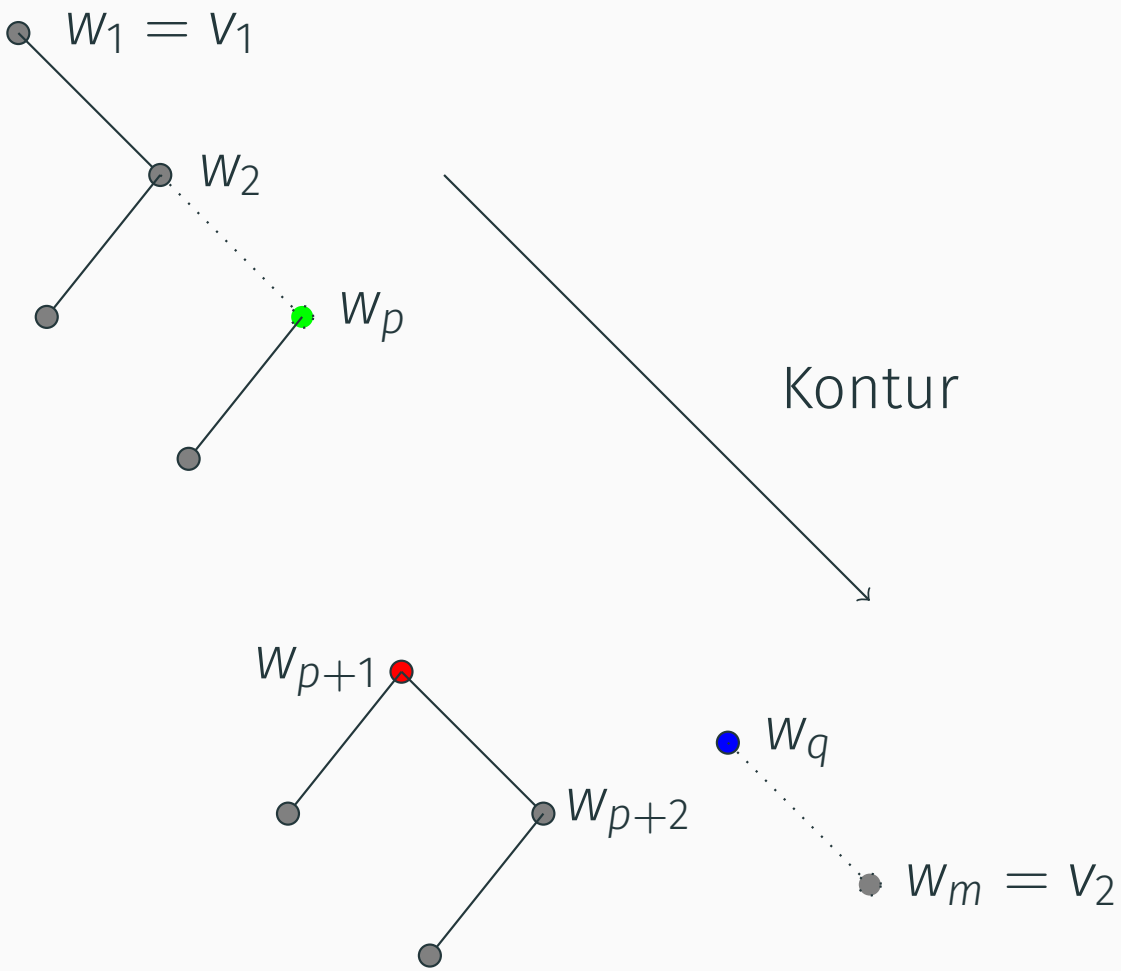




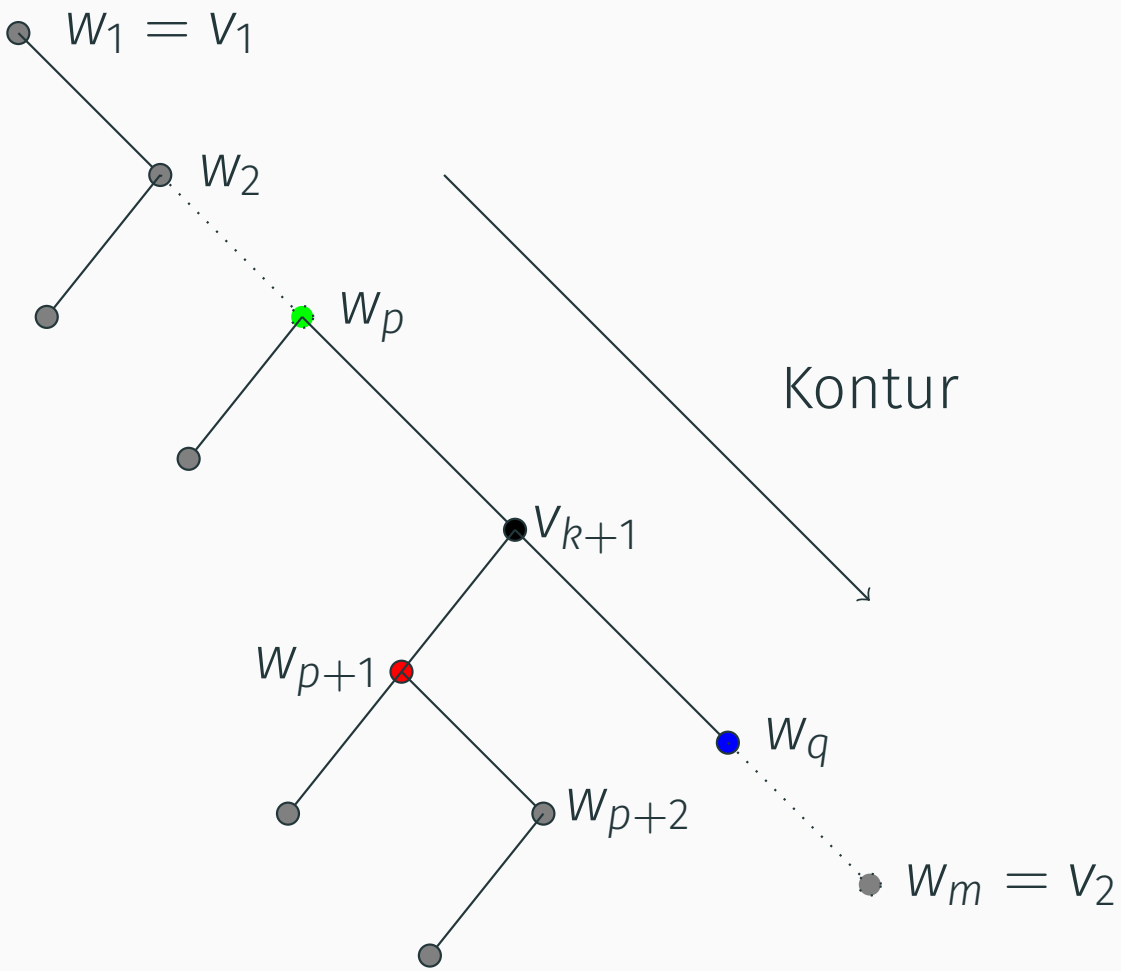
Einfügen im Binärbaum



Einfügen im Binärbaum



Einfügen im Binärbaum



Verschieben im Binärbaum

Beobachtung

Der linke Teilbaum eines Knotens v entspricht $L(v)$.

$\Delta x(w_i)$ verschiebt alle $L(w_i), \ldots, L(w_m)$.

Wir können Lücken erzeugen, indem wir $\Delta x(w_{p+1})$ und $\Delta x(w_q)$ erhöhen.

Verschieben im Binärbaum

Beobachtung

Der linke Teilbaum eines Knotens v entspricht $L(v)$.

$\Delta x(w_i)$ verschiebt alle $L(w_i), \ldots, L(w_m)$.

Wir können Lücken erzeugen, indem wir $\Delta x(w_{p+1})$ und $\Delta x(w_q)$ erhöhen.

Wenn der Elternknoten wechselt, muss der Abstand neu berechnet werden! (w_{p+1}, w_q)

Verschieben im Binärbaum

Beobachtung

Der linke Teilbaum eines Knotens v entspricht $L(v)$.

$\Delta x(w_i)$ verschiebt alle $L(w_i), \ldots, L(w_m)$.

Wir können Lücken erzeugen, indem wir $\Delta x(w_{p+1})$ und $\Delta x(w_q)$ erhöhen.

Wenn der Elternknoten wechselt, muss der Abstand neu berechnet werden! (w_{p+1}, w_q)

Am Ende summieren wir rekursiv x-Koordinaten auf (1 Aufruf pro Knoten).

Korrektheit klar, da die selben Positionen berechnet werden.

Abstandsberechnung

Wie können wir v_{k+1} nur mit Abständen einfügen?

Abstandsberechnung

Wie können wir v_{k+1} nur mit Abständen einfügen?

Nimm an, dass $x(w_p) = 0$; mit y-Koordinaten und $d = x(w_q) - x(w_p)$ berechnen wir $\Delta x(v_{k+1})$ (relativ zu w_p) und $y(v_{k+1})$.

Abstandsberechnung

Wie können wir v_{k+1} nur mit Abständen einfügen?

Nimm an, dass $x(w_p) = 0$; mit y-Koordinaten und $d = x(w_q) - x(w_p)$ berechnen wir $\Delta x(v_{k+1})$ (relativ zu w_p) und $y(v_{k+1})$.

$d = \Delta x(w_{p+1}) + \cdots + \Delta x(w_q)$: höchstens $\mathbf{deg}(v_{k+1})$ Summanden.

Das ist möglich, da $\sum_{v \in V} \mathbf{deg}(v) \in \mathcal{O}(|E|) \subseteq \mathcal{O}(n)$.

Struktur des Algorithmus

procedure ZEICHNE(G)

G einbetten und triangulieren

$v_1, \ldots, v_n :=$ kanonische Sortierung von G

Initialisiere Binärbaum für G_3

Struktur des Algorithmus

procedure ZEICHNE(G)

G einbetten und triangulieren

$v_1, \ldots, v_n :=$ kanonische Sortierung von G

Initialisiere Binärbaum für G_3

for jeden weiteren Knoten v_{k+1} **do**

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

Berechne $\Delta x(v_{k+1})$, $y(v_{k+1})$ und füge in den Baum ein

Aktualisiere Elternknoten, Abstand für w_{p+1}, w_q

end for

Struktur des Algorithmus

procedure ZEICHNE(G)
    G einbetten und triangulieren
    $v_1, \ldots, v_n :=$ kanonische Sortierung von G
    Initialisiere Binärbaum für G_3
    for jeden weiteren Knoten v_{k+1} **do**
        Erhöhe $\Delta x(w_{p+1})$, $\Delta x(w_q)$ um 1
        Berechne $\Delta x(v_{k+1})$, $y(v_{k+1})$ und füge in den Baum ein
        Aktualisiere Elternknoten, Abstand für w_{p+1}, w_q
    end for
    Summiere Abstände rekursiv auf
    Entferne Kanten der Triangulierung
end procedure

Zusammenfassung

Zusammenfassung

- Geradlinige, planare Zeichnung eines Graphen in Linearzeit
- Koordinatengitter: $(2n - 4) \times (n - 2)$

Zusammenfassung

- Geradlinige, planare Zeichnung eines Graphen in Linearzeit
- Koordinatengitter: $(2n - 4) \times (n - 2)$
- In kanonischer Sortierung werden Lücken geschaffen und Knoten hinzugefügt

Zusammenfassung

- Geradlinige, planare Zeichnung eines Graphen in Linearzeit
- Koordinatengitter: $(2n - 4) \times (n - 2)$
- In kanonischer Sortierung werden Lücken geschaffen und Knoten hinzugefügt
- Effiziente Berechnung der x-Koordinaten durch Abstände und Binärbaum

Verbesserungen

Variante des Algorithmus erzeugt $(n-2) \times (n-2)$-Gitter durch größere Steigungen als 1 (asymmetrisch)

[8] Tutte 1960.
[9] Chiba, Yamanouchi *u. a.* 1984.
[10] Chrobak & Kant 1997.
[11] De Fraysseix *u. a.* 1990.

Verbesserungen

Variante des Algorithmus erzeugt $(n-2) \times (n-2)$-Gitter durch größere Steigungen als 1 (asymmetrisch)

Konvexe Zeichenalgorithmen erzeugen oft bessere Zeichnungen, wenn viele Kanten entfernt werden müssen (Triangulierung)

- Für 3-zusammenhängende Graphen möglich[8]; in Linearzeit[9]
- $(n-2) \times (n-2)$-Gitter[10], ähnliche Idee (kanonische Zerlegung)

[8] Tutte 1960.
[9] Chiba, Yamanouchi *u. a.* 1984.
[10] Chrobak & Kant 1997.
[11] De Fraysseix *u. a.* 1990.

Verbesserungen

Variante des Algorithmus erzeugt $(n-2) \times (n-2)$-Gitter durch größere Steigungen als 1 (asymmetrisch)

Konvexe Zeichenalgorithmen erzeugen oft bessere Zeichnungen, wenn viele Kanten entfernt werden müssen (Triangulierung)

- Für 3-zusammenhängende Graphen möglich[8]; in Linearzeit[9]
- $(n-2) \times (n-2)$-Gitter[10], ähnliche Idee (kanonische Zerlegung)

Untere Schranke[11]: $(\frac{2}{3}n - 1) \times (\frac{2}{3}n - 1)$-Gitter

[8] Tutte 1960.
[9] Chiba, Yamanouchi *u. a.* 1984.
[10] Chrobak & Kant 1997.
[11] De Fraysseix *u. a.* 1990.

Vielen Dank!

Anhang

Vollständiger Algorithmus I

Nach Berechnung der Abstände summieren wir x-Koordinaten auf.

Vollständiger Algorithmus I

Nach Berechnung der Abstände summieren wir x-Koordinaten auf.

procedure EINSAMMELN(v: Knoten, $x(p)$: x-Koordinate des Elternknotens)

$x(v) = x(p) + \Delta x(v)$

EINSAMMELN(v.left, $x(v)$)

EINSAMMELN(v.right, $x(v)$)

end procedure

Höchstens ein Aufruf pro Knoten, also $\mathcal{O}(n)$.

Vollständiger Algorithmus II

procedure ZEICHNE(G)

G einbetten und triangulieren

$v_1, \ldots, v_n =:$ kanonische Sortierung von G

Vollständiger Algorithmus II

procedure ZEICHNE(G)

G einbetten und triangulieren

$v_1, \ldots, v_n =:$ kanonische Sortierung von G

▷ Initialisiere G_3 und Binärbaum

$\Delta x(v_1) \leftarrow 0;\ y(v_1) \leftarrow 0;\ v_1.\text{left} \leftarrow \emptyset;\ v_1.\text{right} \leftarrow v_3$

$\Delta x(v_2) \leftarrow 1;\ y(v_2) \leftarrow 0;\ v_2.\text{left} \leftarrow \emptyset;\ v_2.\text{right} \leftarrow \emptyset$

$\Delta x(v_3) \leftarrow 1;\ y(v_3) \leftarrow 1;\ v_3.\text{left} \leftarrow \emptyset;\ v_3.\text{right} \leftarrow v_2$

Vollständiger Algorithmus II

procedure ZEICHNE(G)

G einbetten und triangulieren

$v_1, \ldots, v_n =:$ kanonische Sortierung von G

▷ Initialisiere G_3 und Binärbaum

$\Delta x(v_1) \leftarrow 0;\ y(v_1) \leftarrow 0;\ v_1.\text{left} \leftarrow \emptyset;\ v_1.\text{right} \leftarrow v_3$

$\Delta x(v_2) \leftarrow 1;\ y(v_2) \leftarrow 0;\ v_2.\text{left} \leftarrow \emptyset;\ v_2.\text{right} \leftarrow \emptyset$

$\Delta x(v_3) \leftarrow 1;\ y(v_3) \leftarrow 1;\ v_3.\text{left} \leftarrow \emptyset;\ v_3.\text{right} \leftarrow v_2$

for $3 \leq k < n$ **do**

▷ Kontur von G_k ist $w_1, \ldots, w_m$

Vollständiger Algorithmus II

procedure ZEICHNE(G)

G einbetten und triangulieren

$v_1, \ldots, v_n =:$ kanonische Sortierung von G

▷ Initialisiere G_3 und Binärbaum

$\Delta x(v_1) \leftarrow 0;\ y(v_1) \leftarrow 0;\ v_1.\text{left} \leftarrow \emptyset;\ v_1.\text{right} \leftarrow v_3$

$\Delta x(v_2) \leftarrow 1;\ y(v_2) \leftarrow 0;\ v_2.\text{left} \leftarrow \emptyset;\ v_2.\text{right} \leftarrow \emptyset$

$\Delta x(v_3) \leftarrow 1;\ y(v_3) \leftarrow 1;\ v_3.\text{left} \leftarrow \emptyset;\ v_3.\text{right} \leftarrow v_2$

for $3 \leq k < n$ **do**

▷ Kontur von G_k ist $w_1, \ldots, w_m$

▷ Erzeuge Lücken für v_{k+1} (Nachbarn in G_k: $w_p, \ldots, w_q$)

$\Delta x(w_{p+1}) \leftarrow \Delta x(w_{p+1}) + 1$

$\Delta x(w_q) \leftarrow \Delta x(w_q) + 1$

Vollständiger Algorithmus III

▷ Berechne Koordinaten für v_{k+1}

$d \leftarrow \Delta x(w_{p+1}) + \cdots + \Delta x(w_q)$

$\Delta x(v_{k+1}) \leftarrow \frac{1}{2}(y(w_q) - y(w_p) + d)$

$y(v_{k+1}) \leftarrow \frac{1}{2}(y(w_p) + y(w_q) + d)$

Vollständiger Algorithmus III

▷ Berechne Koordinaten für v_{k+1}

$d \leftarrow \Delta x(w_{p+1}) + \cdots + \Delta x(w_q)$

$\Delta x(v_{k+1}) \leftarrow \frac{1}{2}(y(w_q) - y(w_p) + d)$

$y(v_{k+1}) \leftarrow \frac{1}{2}(y(w_p) + y(w_q) + d)$

▷ Aktualisiere Binärbaum

$w_p.\text{right} \leftarrow v_k$

$v_k.\text{right} \leftarrow w_q$

$\Delta x(w_q) \leftarrow d - \Delta x(v_k)$ ▷ Elternknoten von w_q ändert sich

Vollständiger Algorithmus IV

if $p + 1 < q$ **then** ▷ Bearbeite verdeckte Knoten

Vollständiger Algorithmus IV

if $p + 1 < q$ **then** ▷ Bearbeite verdeckte Knoten

v_k.left $\leftarrow w_{p+1}$

w_{q-1}.right $\leftarrow \emptyset$

$\Delta x(w_{p+1}) = \Delta x(w_{p+1}) - \Delta x(v_k)$ ▷ Neuer Elternknoten

Vollständiger Algorithmus IV

if $p + 1 < q$ **then** ▷ Bearbeite verdeckte Knoten

  v_k.left $\leftarrow w_{p+1}$

  w_{q-1}.right $\leftarrow \emptyset$

  $\Delta x(w_{p+1}) = \Delta x(w_{p+1}) - \Delta x(v_k)$ ▷ Neuer Elternknoten

else

  v_k.left $= \emptyset$

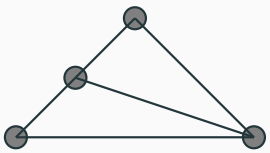
end if

Vollständiger Algorithmus IV

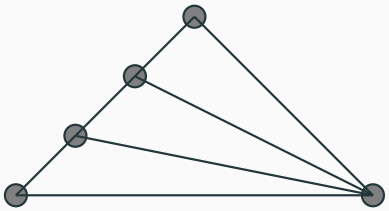
if $p + 1 < q$ **then** ▷ Bearbeite verdeckte Knoten
  v_k.left $\leftarrow w_{p+1}$
  w_{q-1}.right $\leftarrow \emptyset$
  $\Delta x(w_{p+1}) = \Delta x(w_{p+1}) - \Delta x(v_k)$ ▷ Neuer Elternknoten
else
  v_k.left $= \emptyset$
end if
end for
EINSAMMELN(v_1, 0) ▷ Berechne endgültige x-Koordinaten
Entferne Kanten der Triangulierung
end procedure

Animation

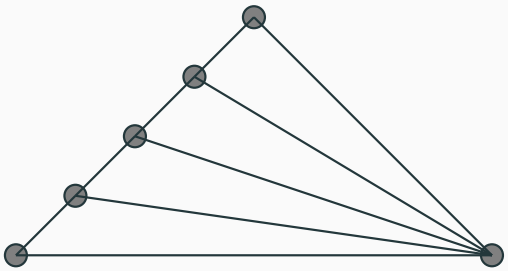
Animation



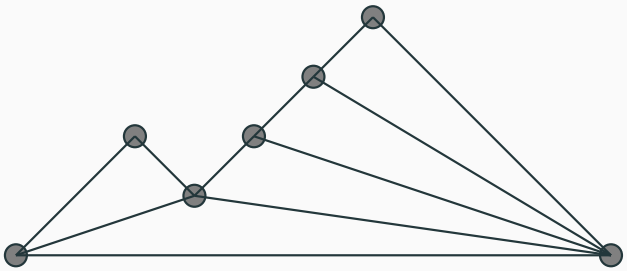
Animation



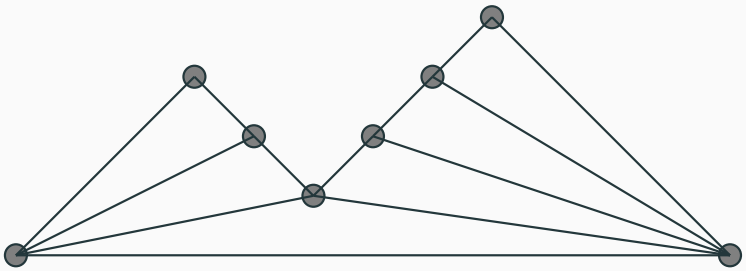
Animation



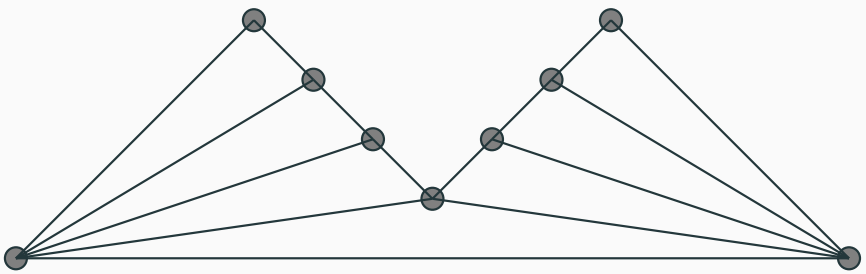
Animation



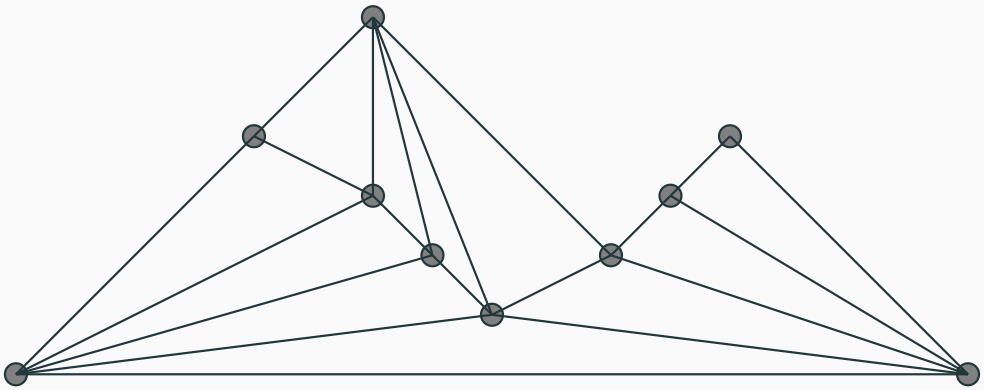
Animation



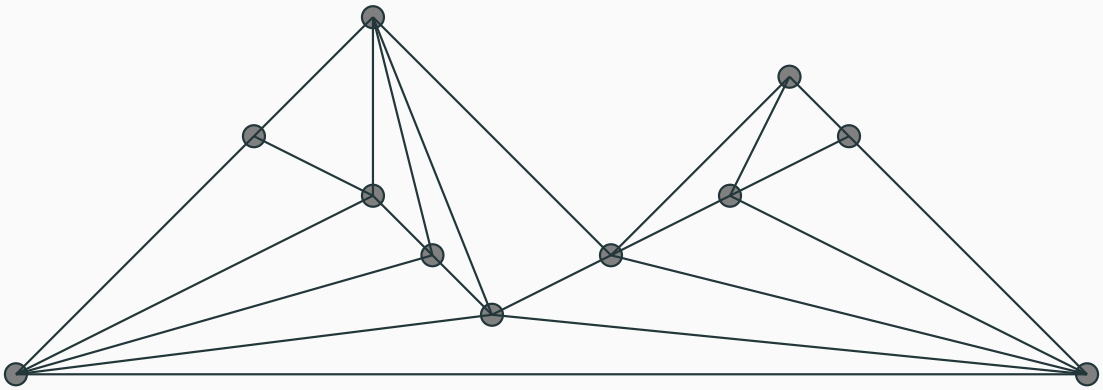
Animation



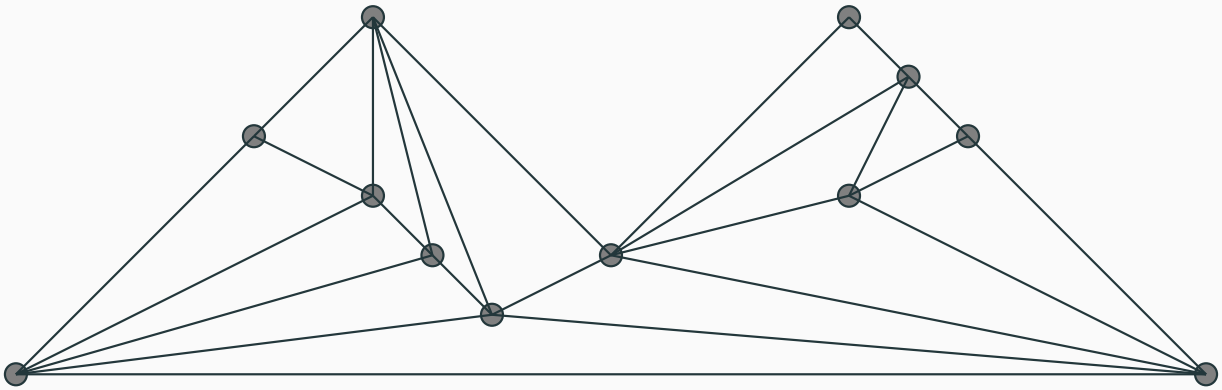
Animation



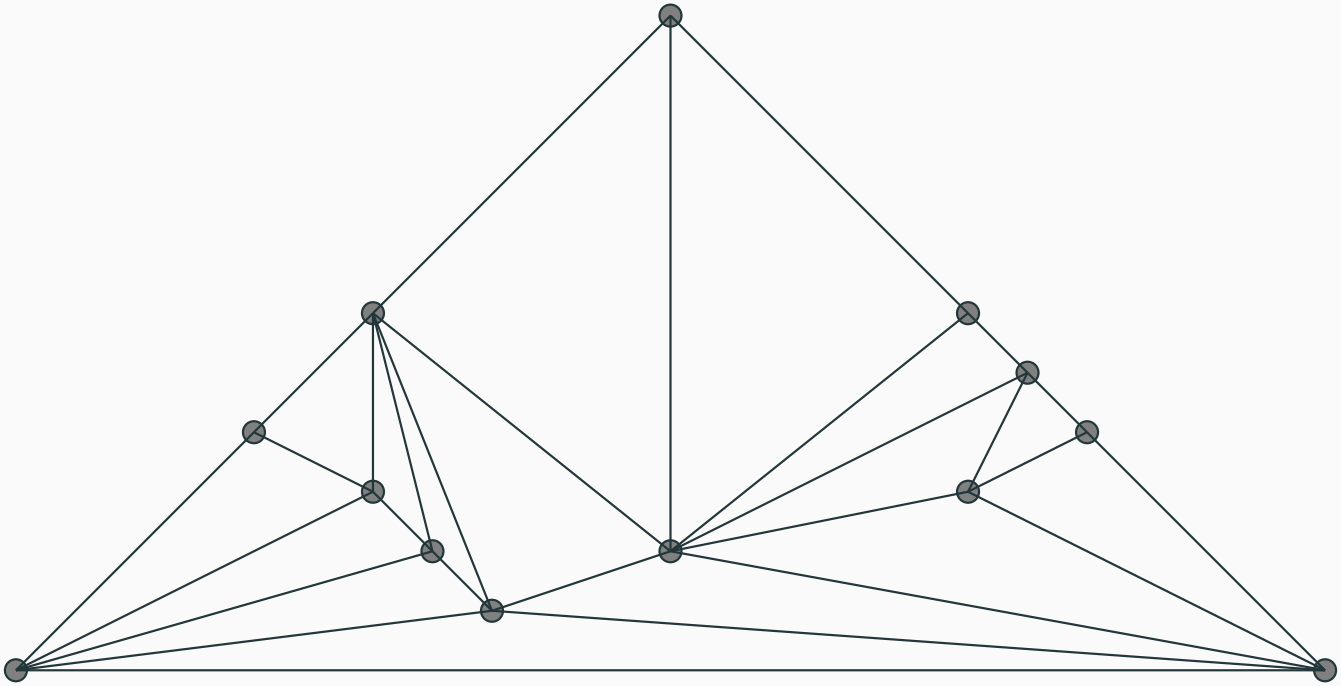
Animation



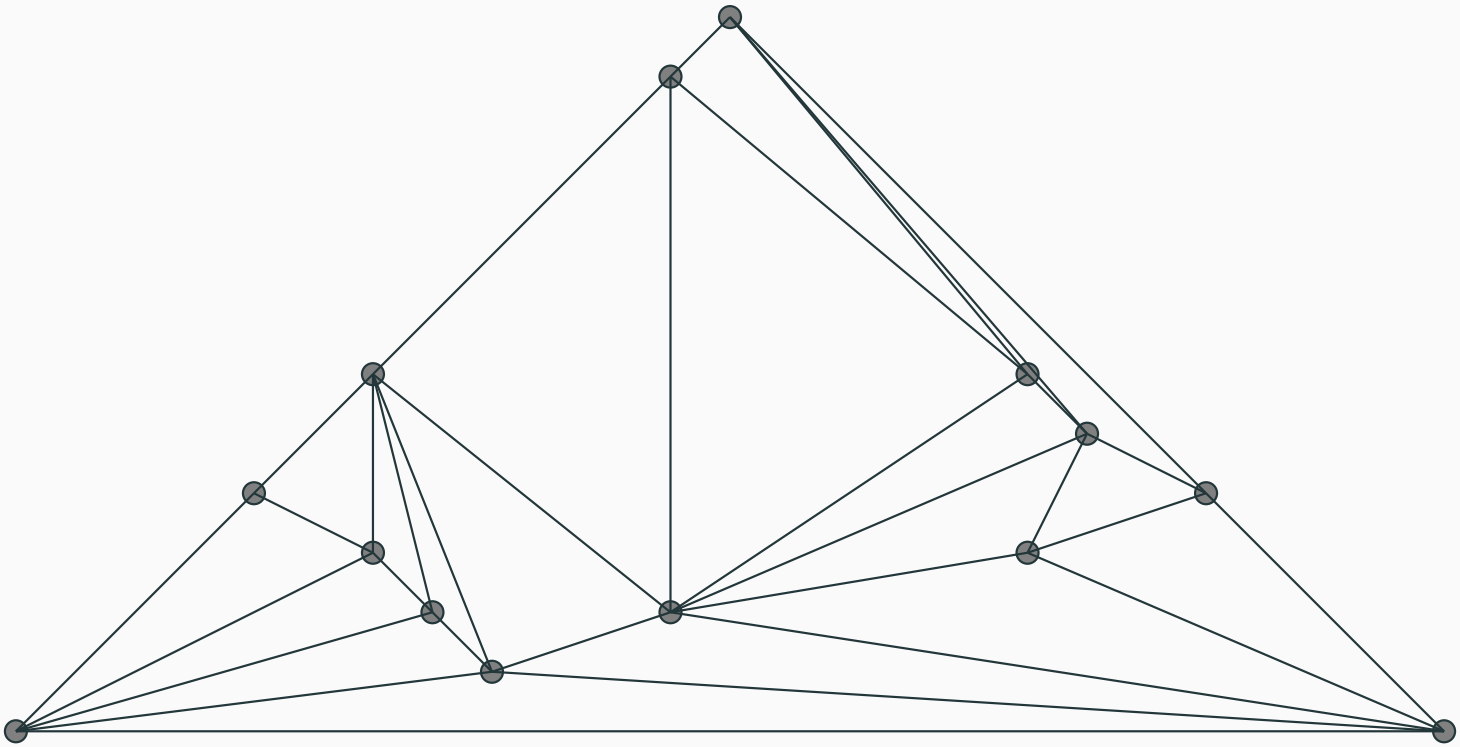
Animation



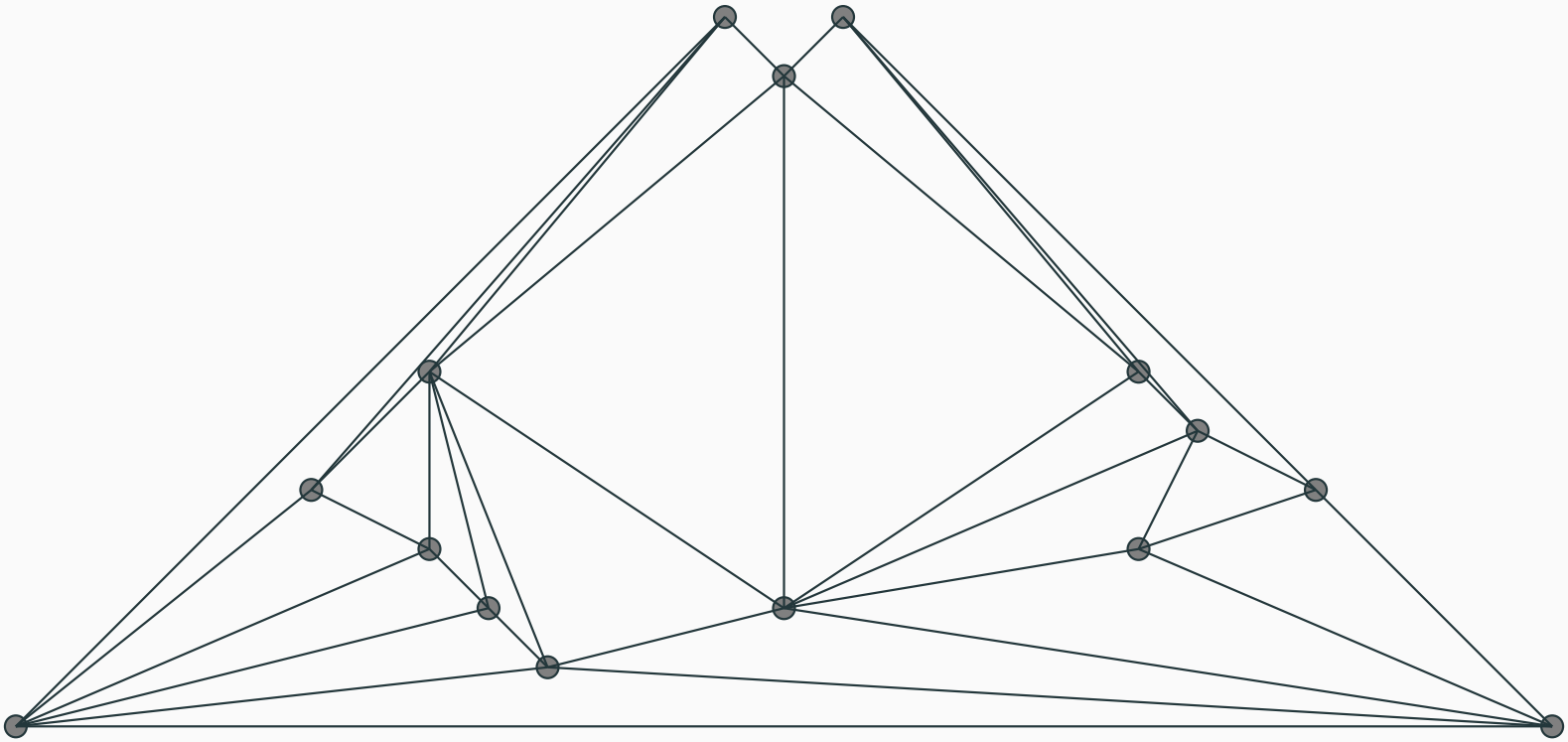
Animation



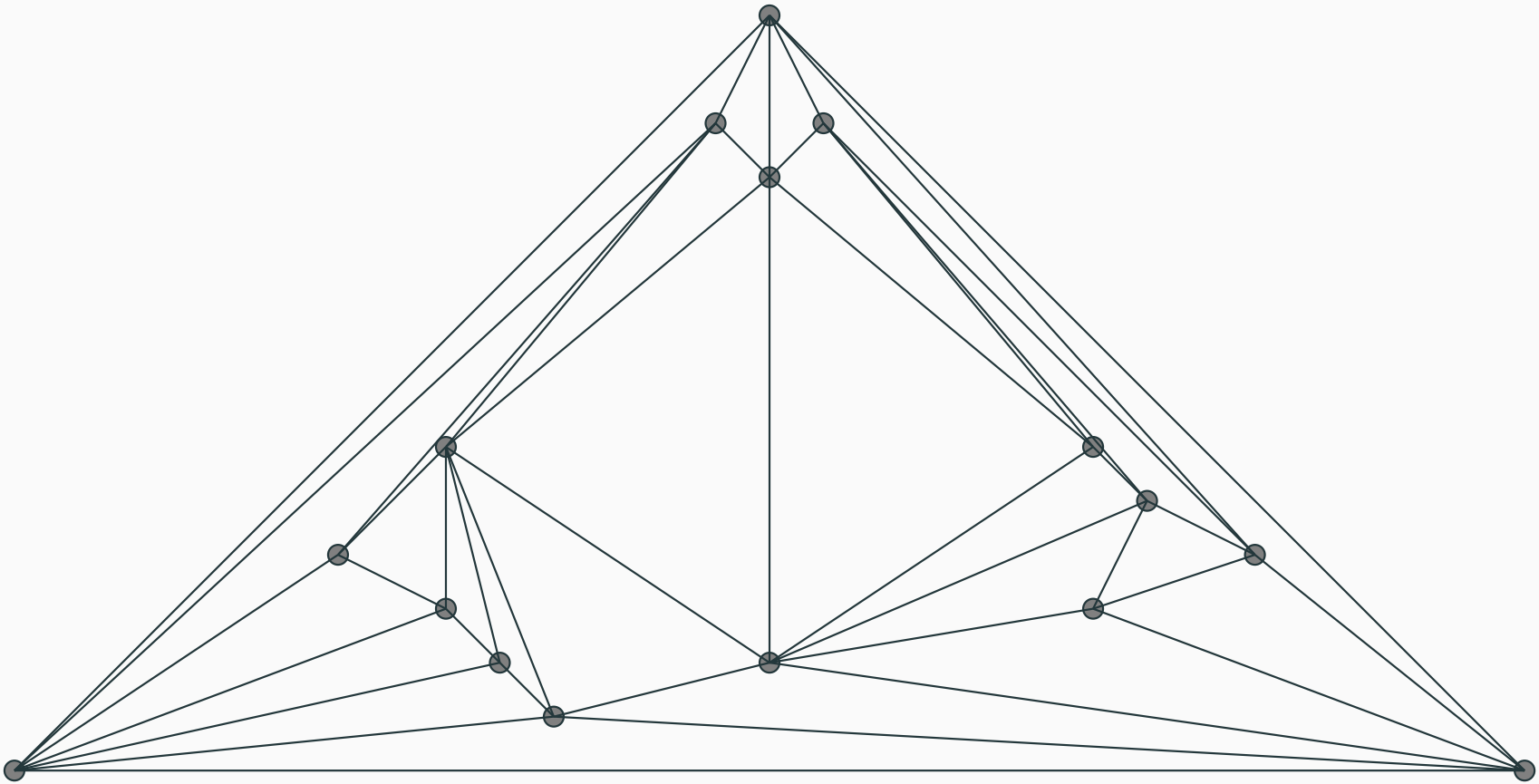
Animation



Animation



Animation



Literatur

1. Biedl, T. & Kant, G. *A better heuristic for orthogonal graph drawings.* in *European Symposium on Algorithms* (1994), 24–35.
2. Chiba, N., Onoguchi, K. & Nishizeki, T. Drawing plane graphs nicely. *Acta Informatica* **22,** 187–201 (1985).
3. Chiba, N., Yamanouchi, T. & Nishizeki, T. Linear algorithms for convex drawings of planar graphs. *Progress in graph theory,* 153–173 (1984).
4. Chrobak, M. & Kant, G. Convex grid drawings of 3-connected planar graphs. *International Journal of Computational Geometry & Applications* **7,** 211–223 (1997).

Literatur

5. Chrobak, M. & Payne, T. H. A linear-time algorithm for drawing a planar graph on a grid. *Information Processing Letters* **54,** 241–246 (1995).

6. De Fraysseix, H., Pach, J. & Pollack, R. How to draw a planar graph on a grid. *Combinatorica* **10,** 41–51 (1990).

7. Fáry, I. On straight line representation of planar graphs. *Acta Scientiarium Mathematicarum Szeged* **11,** 229–233 (1948).

8. Garey, M. R. & Johnson, D. S. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods* **4,** 312–316 (1983).

9. Hopcroft, J. & Tarjan, R. E. Efficient planarity testing. *Journal of the ACM (JACM)* **21,** 549–568 (1974).

Literatur

10. Kant, G. *Algorithms for drawing planar graphs.* Diss. (1993).
11. Rosenstiehl, P. & Tarjan, R. E. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete & Computational Geometry* **1,** 343–353 (1986).
12. Tamassia, R. & Tollis, I. G. A unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry* **1,** 321–341 (1986).
13. Tutte, W. T. Convex representations of graphs. *Proceedings of the London Mathematical Society* **3,** 304–320 (1960).
14. Tutte, W. T. How to draw a graph. *Proceedings of the London Mathematical Society* **3,** 743–767 (1963).