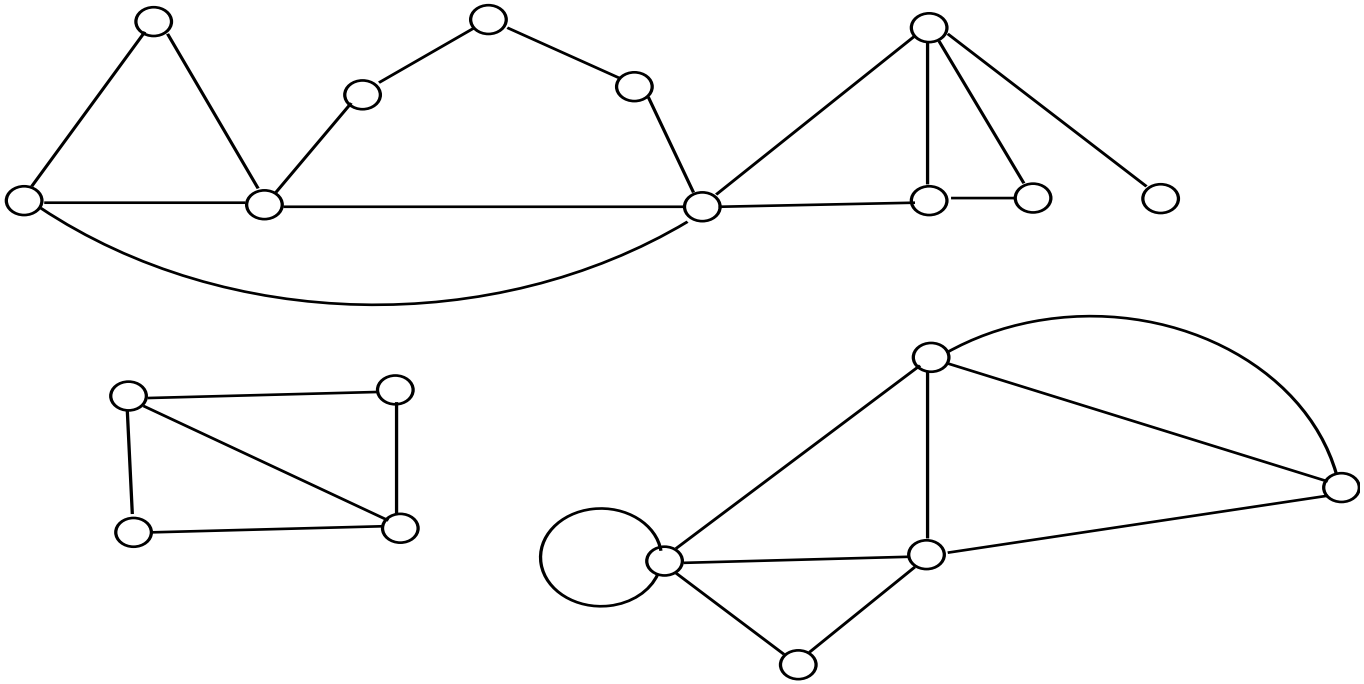
GRUNDBEGRIFFE DER GRAPHENTHEORIE

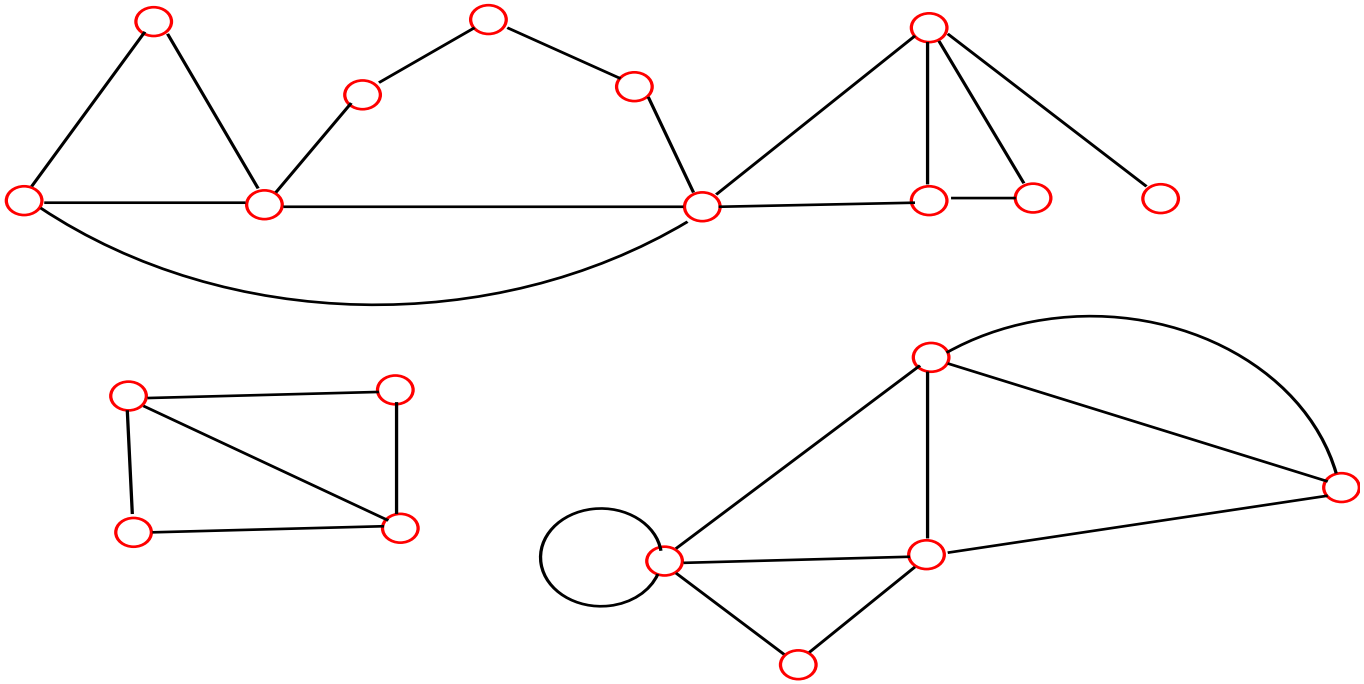
Grundbegriffe der Graphentheorie

Ungerichteter Graph



Grundbegriffe der Graphentheorie

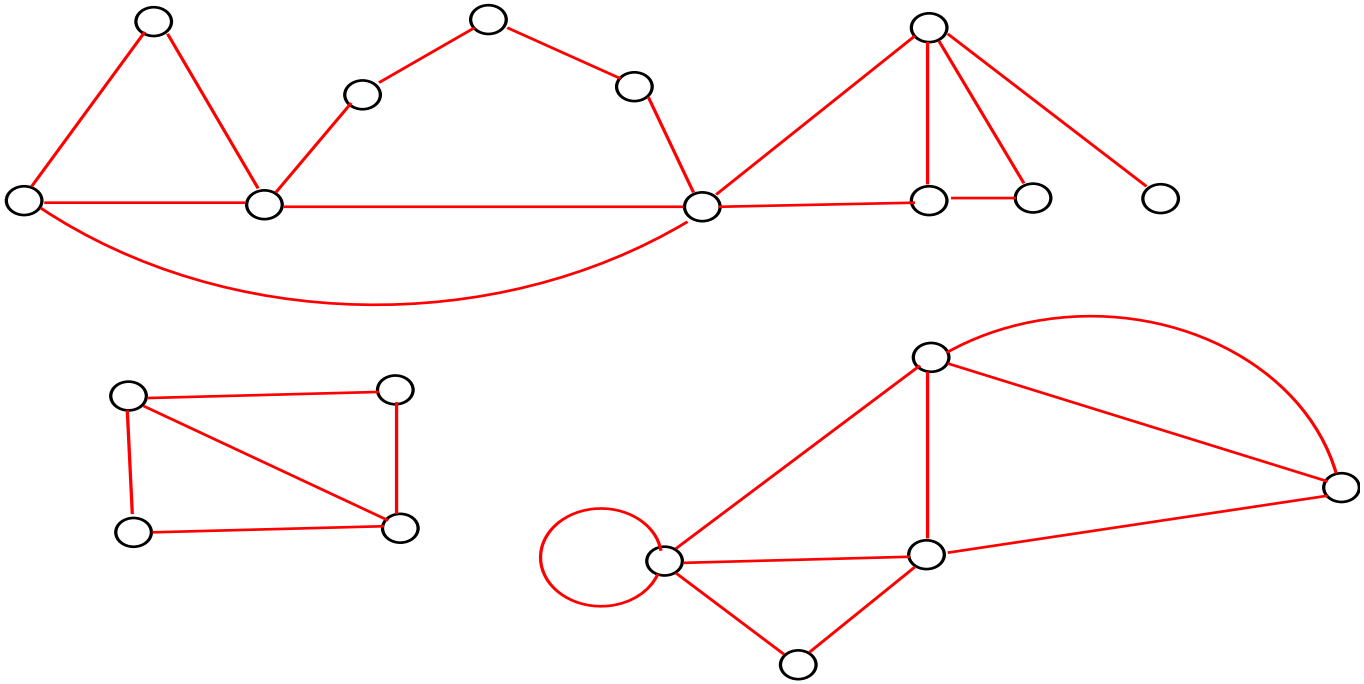
Die Knoten des Graphen



Knotenmenge V, $\alpha_0 := |V|$

Grundbegriffe der Graphentheorie

Die Kanten des Graphen

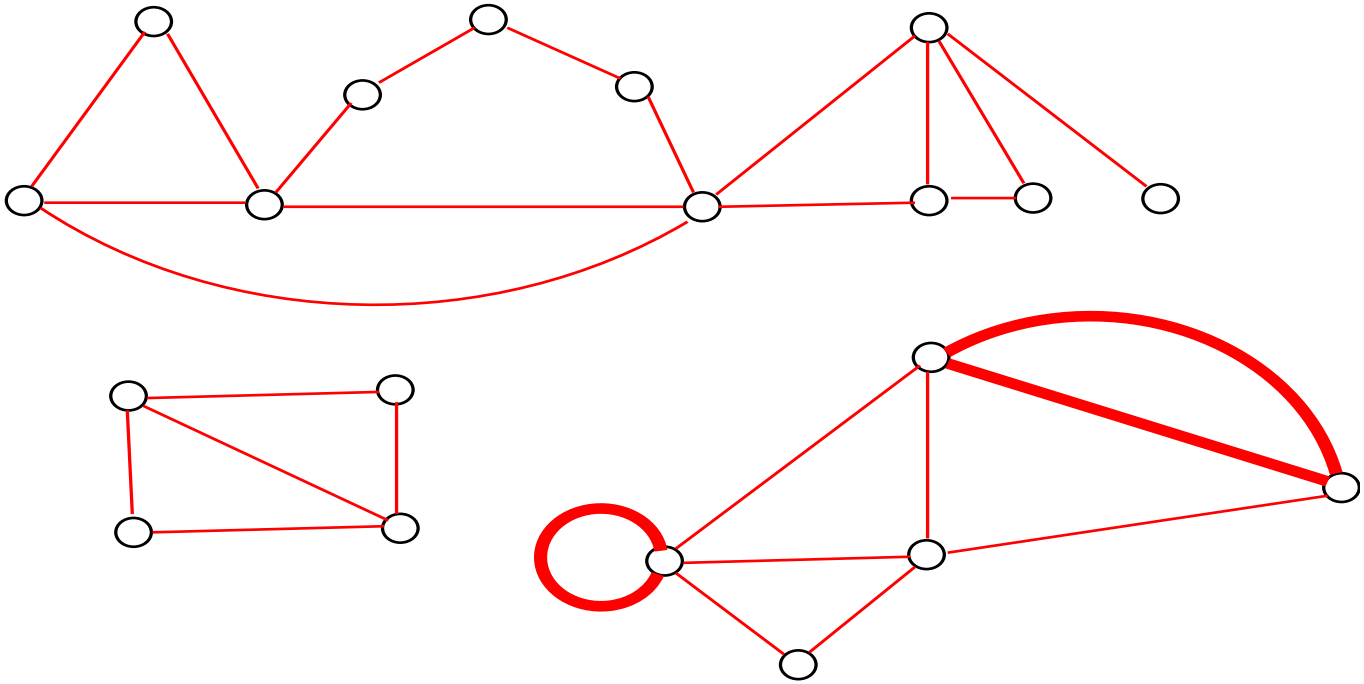


Kantenmenge E, $\alpha_1 := |E|$, $\longrightarrow$ Graph $G = (V, E)$

Dichte $\varepsilon(G) = \frac{|E|}{|V|}$

Grundbegriffe der Graphentheorie

Spezielle Kanten: Schlingen und Mehrfachkanten



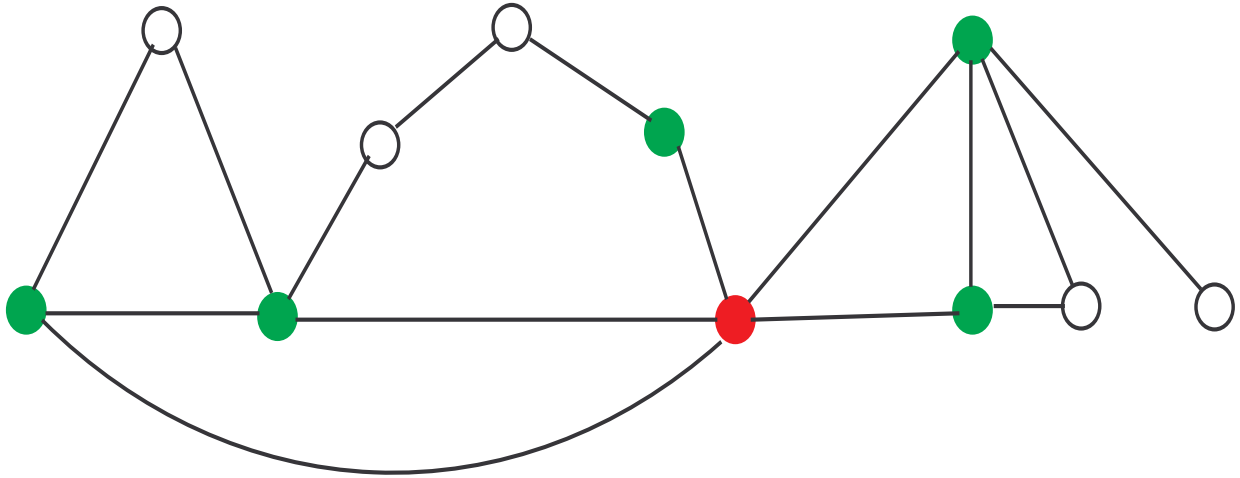
Graphen ohne Schlingen und Mehrfachkanten: **schlichte** Graphen

Grundbegriffe der Graphentheorie

Adjazenz und Inzidenz

Grundbegriffe der Graphentheorie

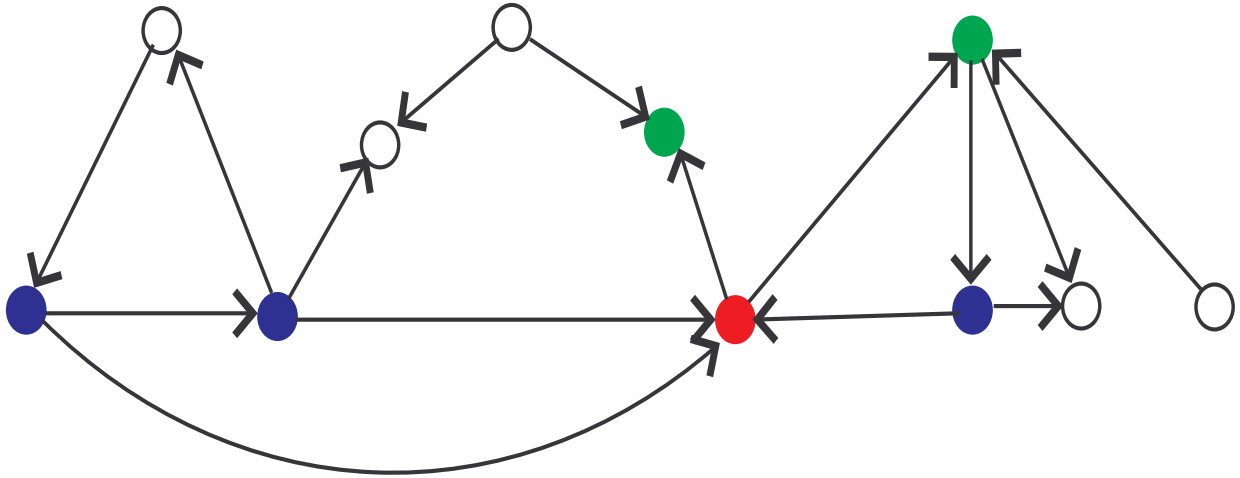
Ein Knoten v und die Menge seiner Nachbarn $\Gamma(v)$
$d(v) = d_G(v) = |\Gamma(v)| =$ der Grad von v



$\delta(G) = \min_{v \in V} d(v), \qquad \Delta(G) = \max_{v \in V} d(v)$

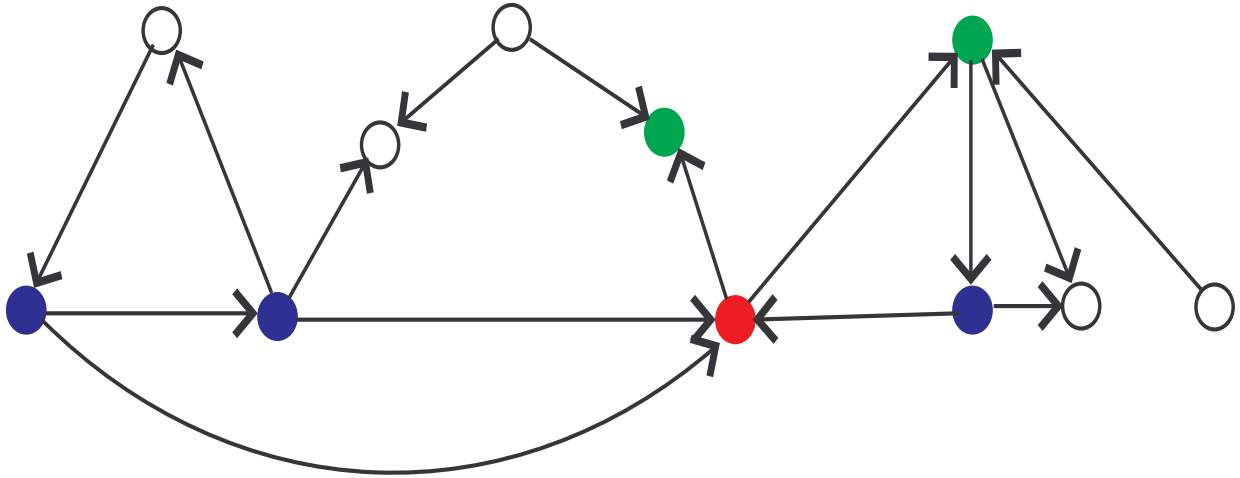
Grundbegriffe der Graphentheorie

Gerichteter Fall: Nachfolger $\Gamma^+(v)$ und Vorgänger $\Gamma^-(v)$
$d^+(v) = |\Gamma^+(v)|$ bzw. $d^-(v) = |\Gamma^-(v)|$: Weggrad bzw. Hingrad von v



Grundbegriffe der Graphentheorie

Gerichteter Fall: Nachfolger $\Gamma^+(v)$ und Vorgänger $\Gamma^-(v)$
$d^+(v) = |\Gamma^+(v)|$ bzw. $d^-(v) = |\Gamma^-(v)|$: Weggrad bzw. Hingrad von v

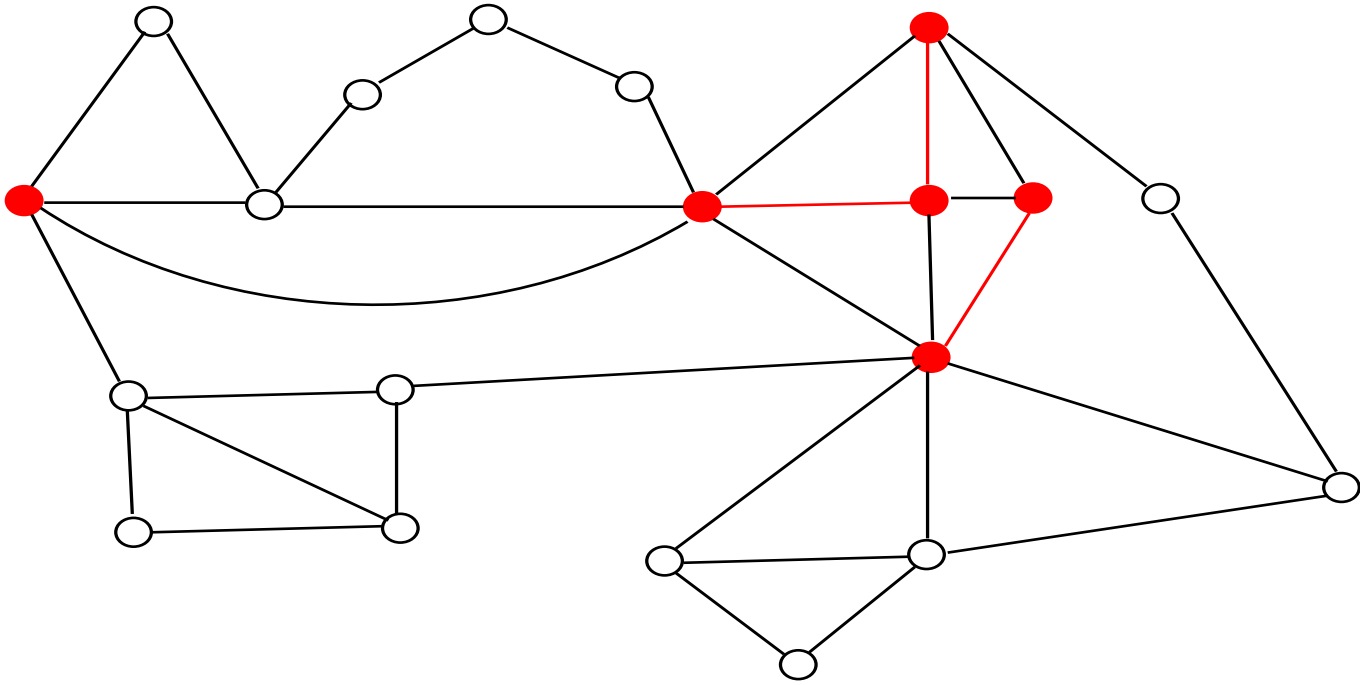


Satz

$$\sum_{x \in V(G)} d(x) = 2|E(G)| \text{ bzw. } \sum_{x \in V(G)} d^+(x) = \sum_{x \in V(G)} d^-(x) = |E(G)|$$

Grundbegriffe der Graphentheorie

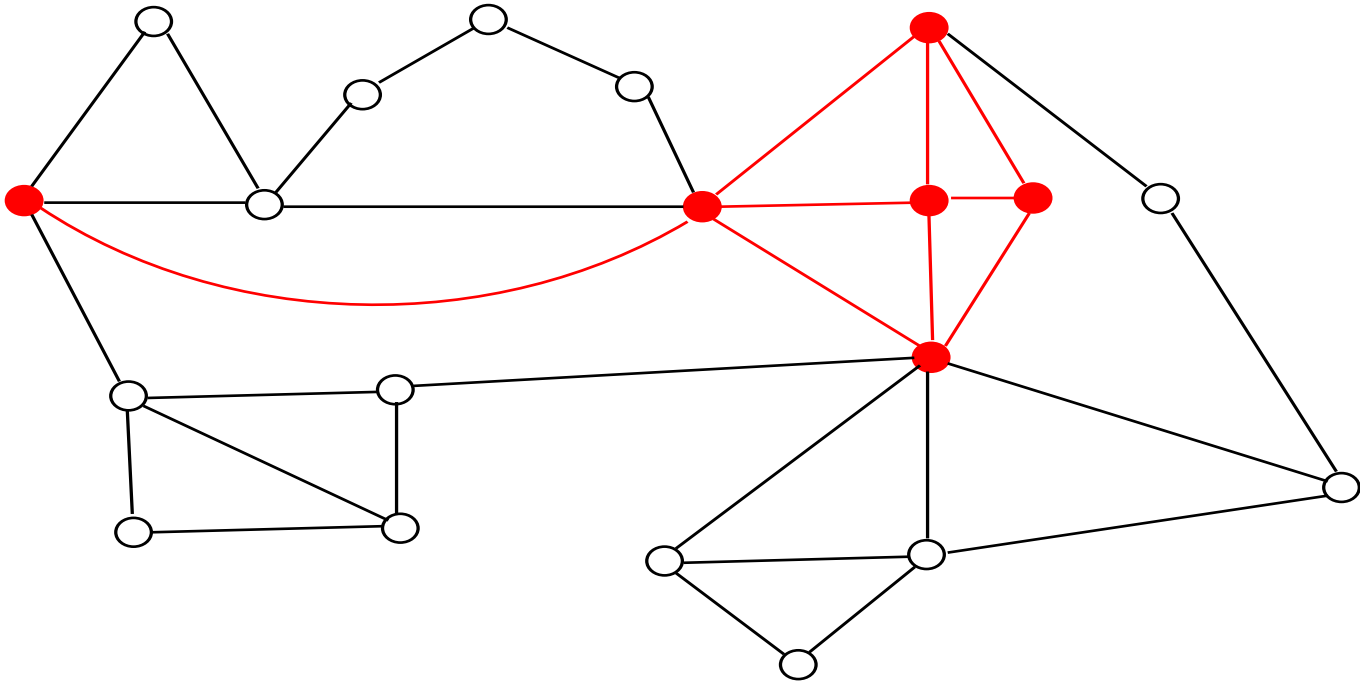
Ein Graph G und einer seiner Teilgraphen, G'



$$G' = (V', E'), \qquad V' \subseteq V, \quad E' \subseteq E$$

Grundbegriffe der Graphentheorie

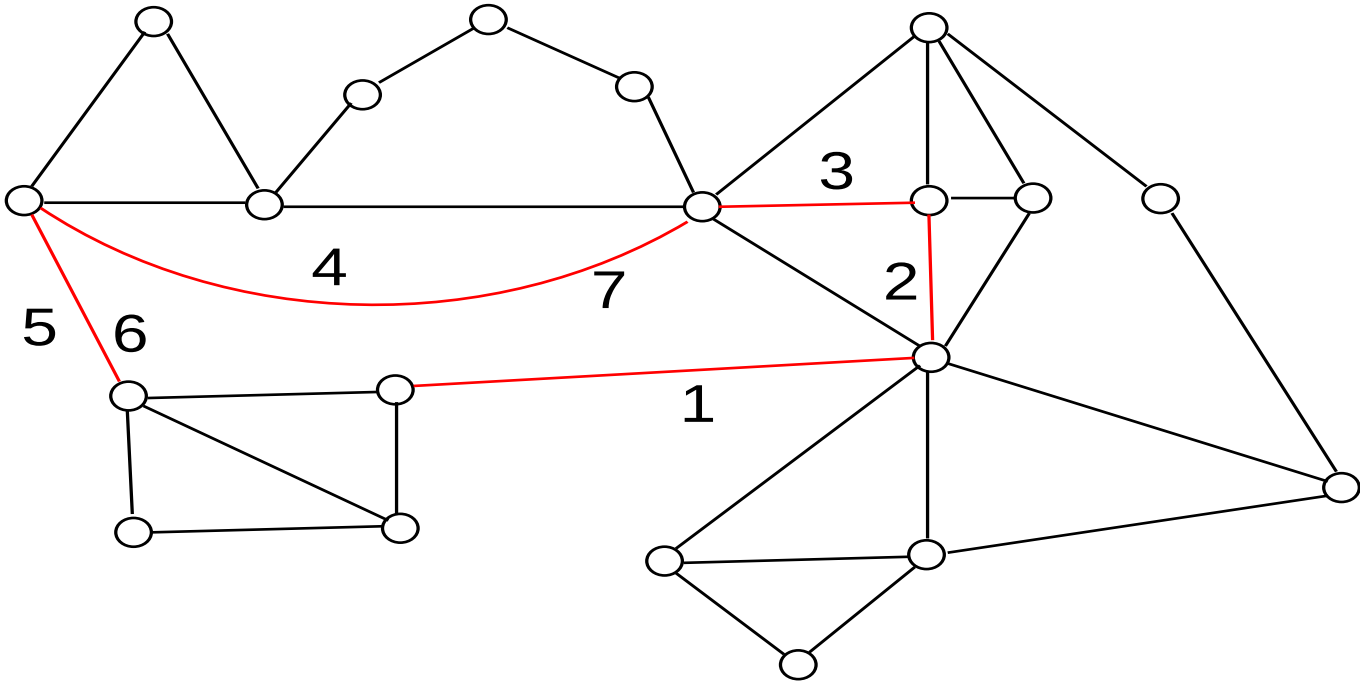
Induzierte Teilgraphen von $G = (V, E)$: $G[V_0]$ durch ihre Knotenmenge $V_0 \subseteq V$ bestimmt



Kantenmenge maximal bzgl. der Inklusion

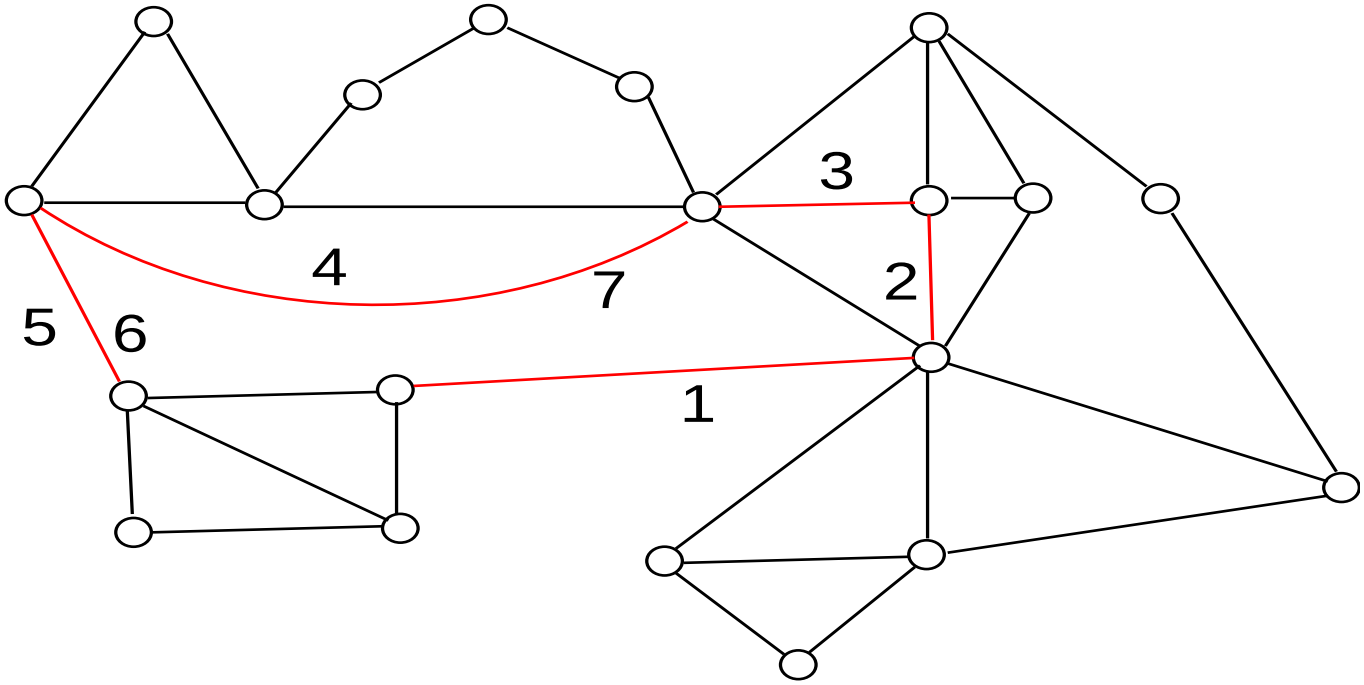
Grundbegriffe der Graphentheorie

Kantenfolgen



Grundbegriffe der Graphentheorie

Kantenfolgen



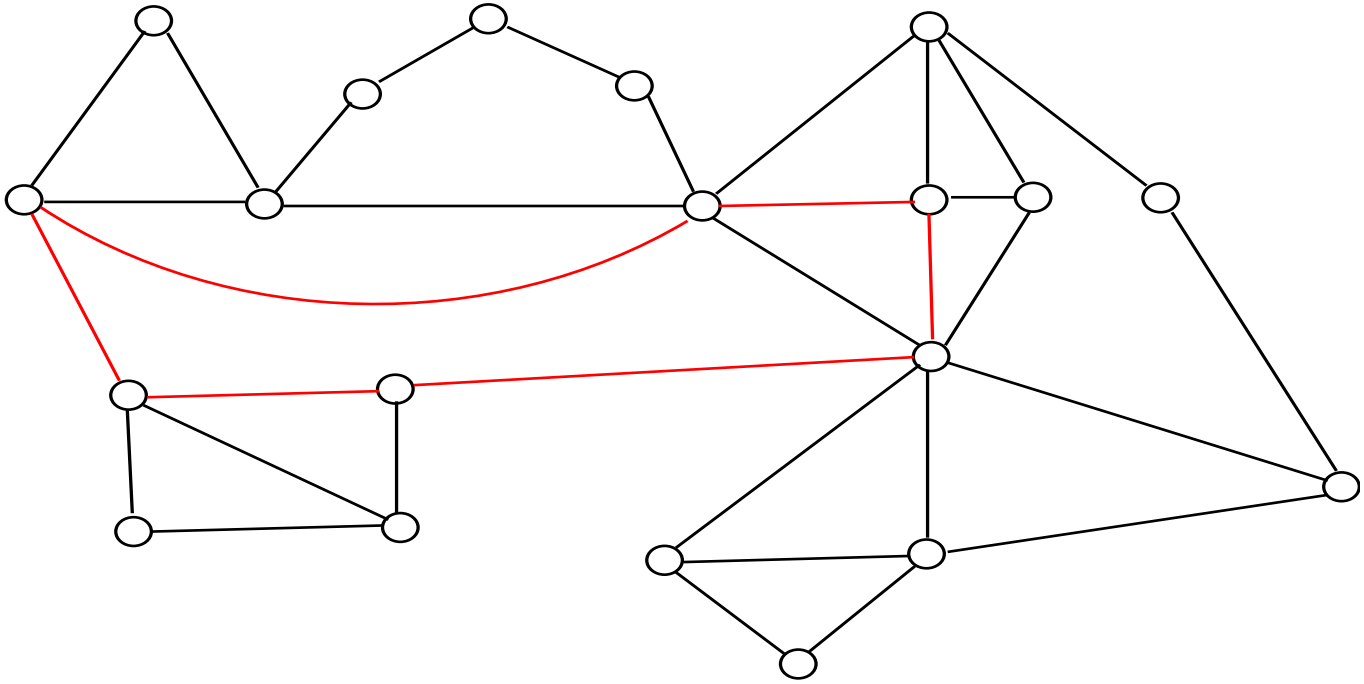
Kantenzug: Kantenfolge, in der keine Kante mehrfach auftritt

Weg: Kantenfolge, in der kein Knoten mehrfach auftritt

Grundbegriffe der Graphentheorie

Spezielle Kantenfolgen: Kreis (in gerichteten Graphen: Zyklus)

. . . ist ein Kantenzug, in dem alle Knoten mit Ausnahme von Anfangs- und Endknoten verschieden sind.



Grundbegriffe der Graphentheorie

Satz *Falls eine Kantenfolge von v nach w existiert, so gibt es auch einen Weg von v nach w.*

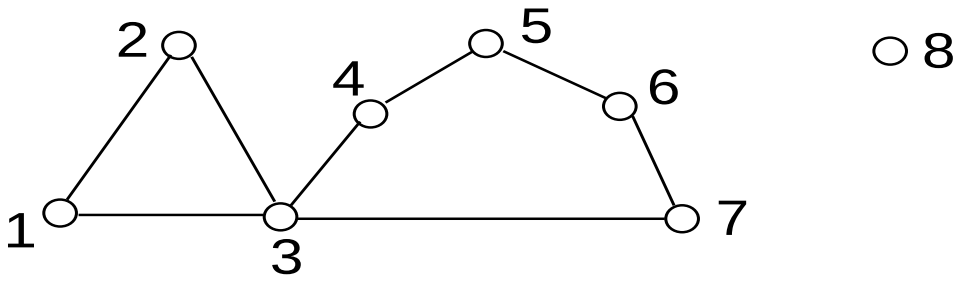
Satz *Falls in einem ungerichteten Graphen 2 verschiedene Wege von v nach w existieren, dann gibt es einen Kreis (positiver Länge).*

Falls in einem gerichteten Graphen eine geschlossene Kantenfolge positiver Länge existiert, dann gibt es einen Zyklus (positiver Länge).

Grundbegriffe der Graphentheorie

Die Adjazenzmatrix von $G = (V, E)$:

$V = \{v_1 \ldots, v_n\}$, $A = (a_{ij})_{i,j=1,\ldots,n}$ mit $a_{ij} = \begin{cases} 1, & \text{falls } (v_i, v_j) \in E, \\ 0 & \text{sonst.} \end{cases}$

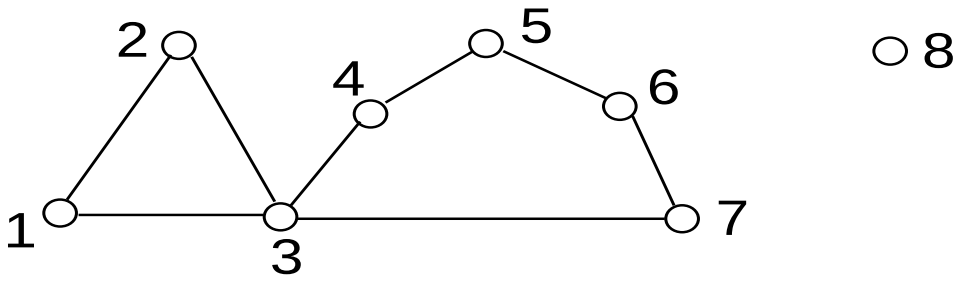


$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Grundbegriffe der Graphentheorie

Die Adjazenzmatrix von $G = (V, E)$:

$V = \{v_1 \ldots, v_n\}$, $A = (a_{ij})_{i,j=1,\ldots,n}$ mit $a_{ij} = \begin{cases} 1, & \text{falls } (v_i, v_j) \in E, \\ 0 & \text{sonst.} \end{cases}$



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Es gilt: $d(v_i) = \sum_{j=1}^{n} a_{ij} = \sum_{j=1}^{n} a_{ji}$

Grundbegriffe der Graphentheorie

Erreichbarkeitsrelation (ungerichteter Fall): $v \sim w$ genau dann, wenn eine (möglicherweise leere) Kantenfolge von v nach w existiert.

Erreichbarkeitsrelation (gerichteter Fall): $v \sim w$ genau dann, wenn sowohl eine (möglicherweise leere) Kantenfolge von v nach w als auch eine (möglicherweise leere) Kantenfolge von w nach v existiert.

Grundbegriffe der Graphentheorie

Erreichbarkeitsrelation (ungerichteter Fall): $v \sim w$ genau dann, wenn eine (möglicherweise leere) Kantenfolge von v nach w existiert.

Erreichbarkeitsrelation (gerichteter Fall): $v \sim w$ genau dann, wenn sowohl eine (möglicherweise leere) Kantenfolge von v nach w als auch eine (möglicherweise leere) Kantenfolge von w nach v existiert.

Matrix der Erreichbarkeitsrelation (ungerichteter Fall): Sei $|V| = n$ und $|E| = m$.

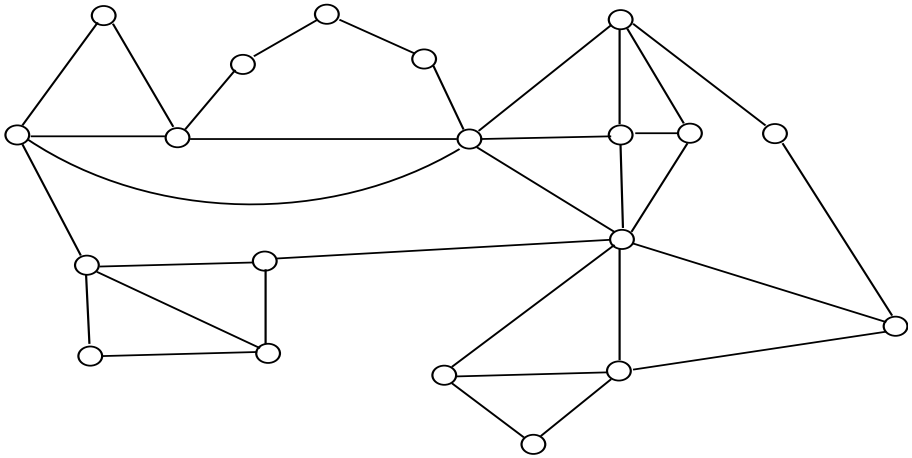
$$M = (m_{i,j})_{i,j=1,\ldots,n}, \text{ wobei } m_{i,j} = \mathrm{sgn}(c_{i,j})$$

und

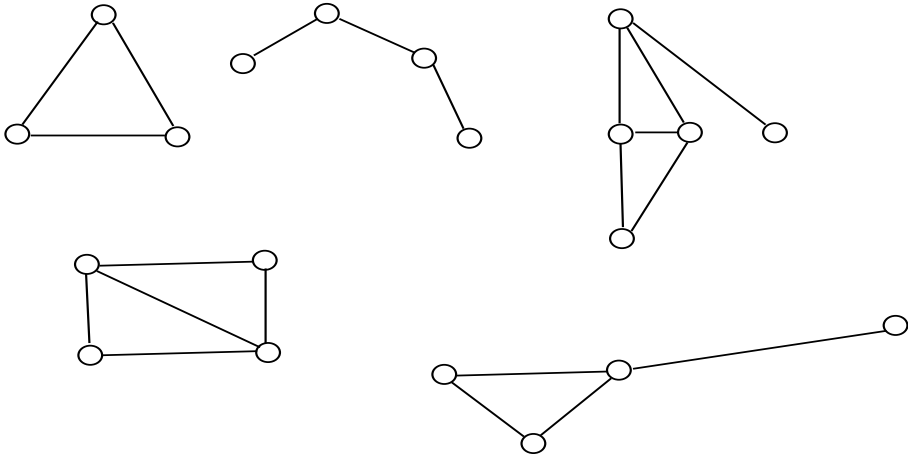
$$C = \sum_{k=0}^{\min(m,n-1)} A^k.$$

Grundbegriffe der Graphentheorie

zusammenhängender Graph

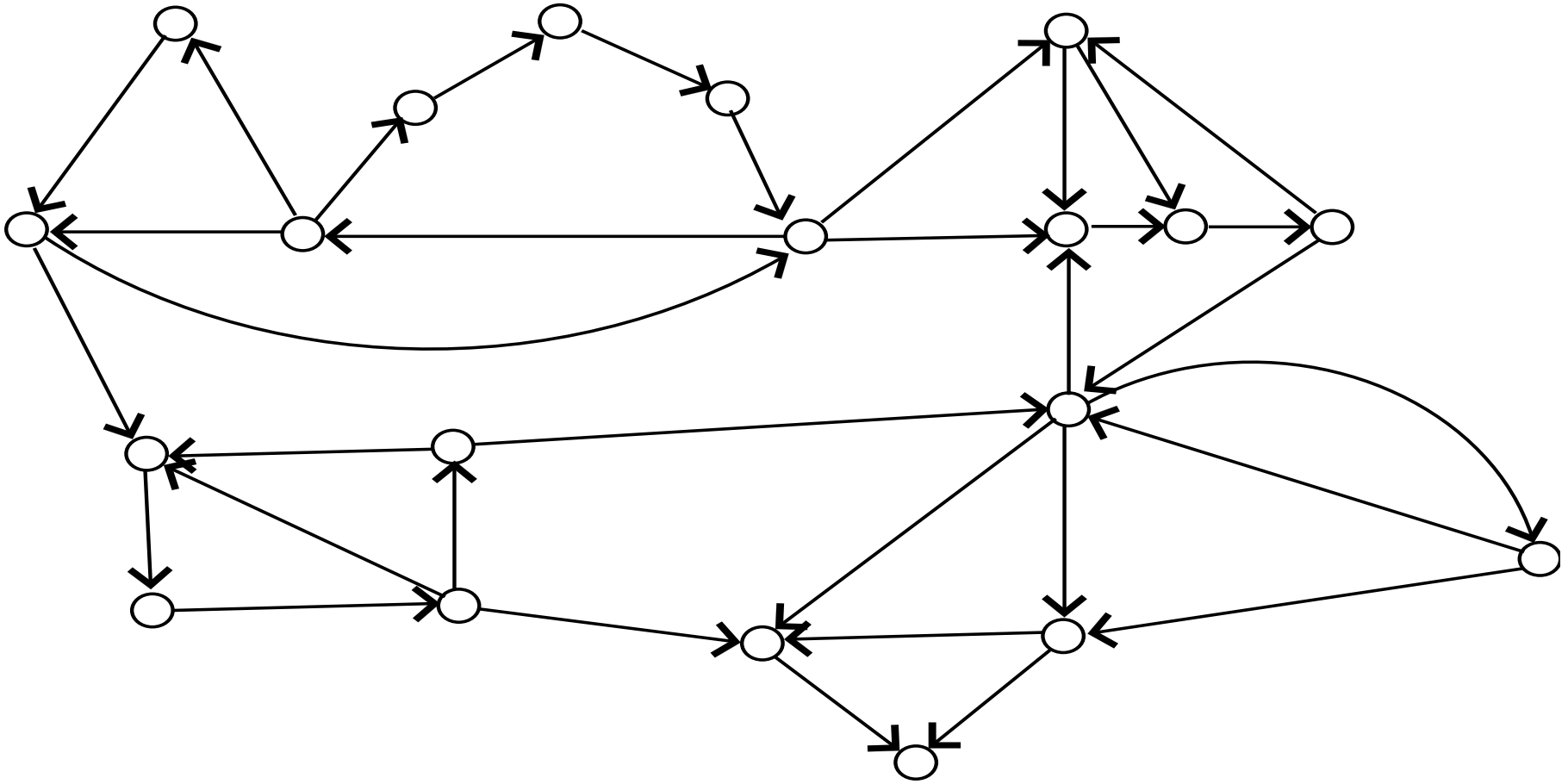


nicht zusammenhängender Graph



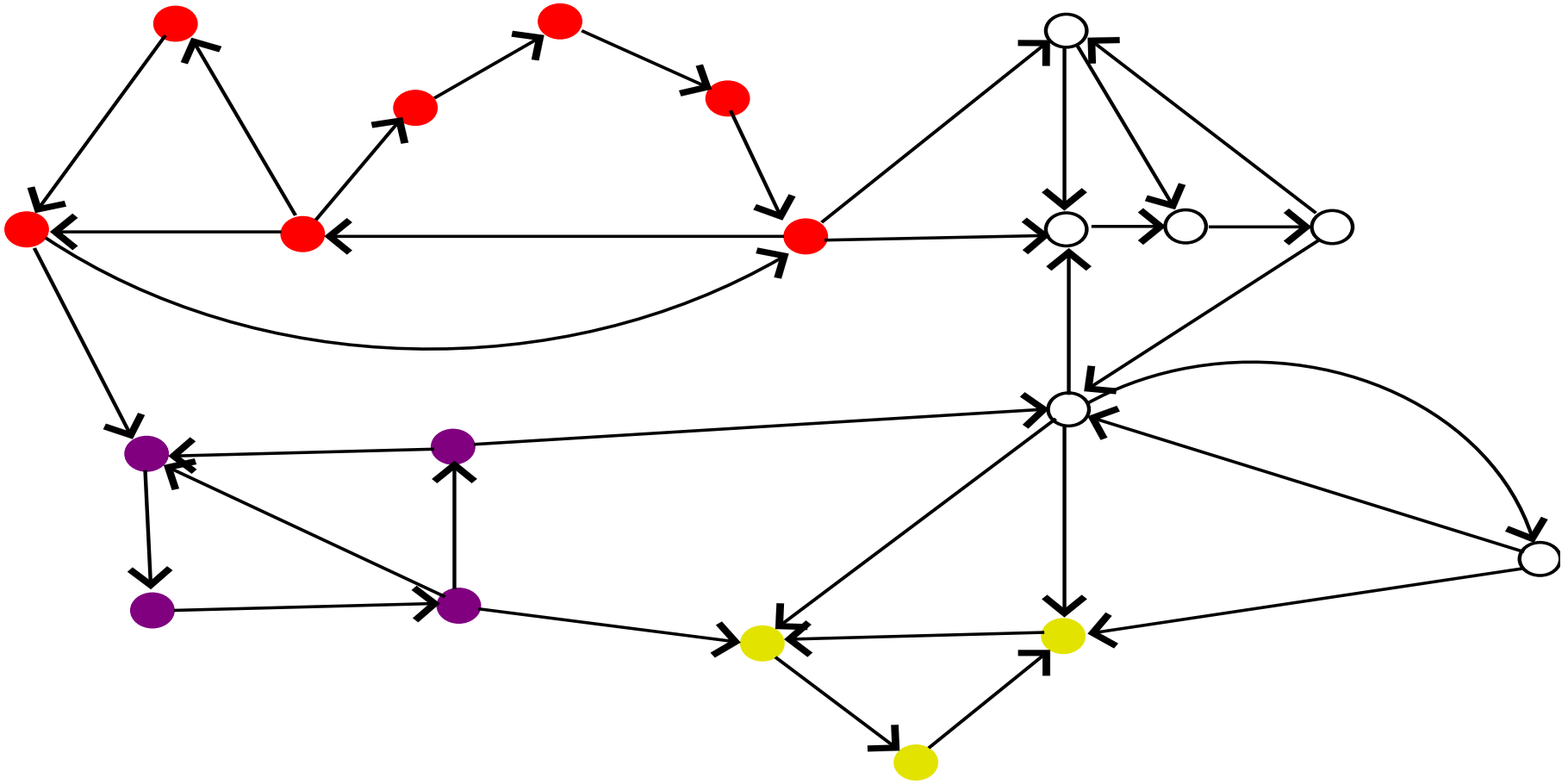
Grundbegriffe der Graphentheorie

Ein schwach, aber nicht stark zusammenhängender Graph



Grundbegriffe der Graphentheorie

Die starken Zusammenhangskomponenten

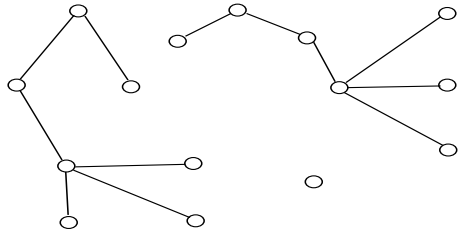


Grundbegriffe der Graphentheorie

Bäume und Wälder

Ein schlichter ungerichteter Graph, der keine Kreise positiver Länge besitzt, heißt *Wald*.

Ein zusammenhängender Wald heißt *Baum*

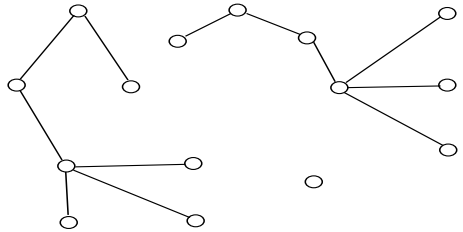


Grundbegriffe der Graphentheorie

Bäume und Wälder

Ein schlichter ungerichteter Graph, der keine Kreise positiver Länge besitzt, heißt *Wald*.

Ein zusammenhängender Wald heißt *Baum*



Satz *Für einen Baum $T = (V, E)$ gilt: Für je zwei Knoten $v, w \in V$ gibt es genau einen Weg $W(v, w)$, der v und w verbindet.*

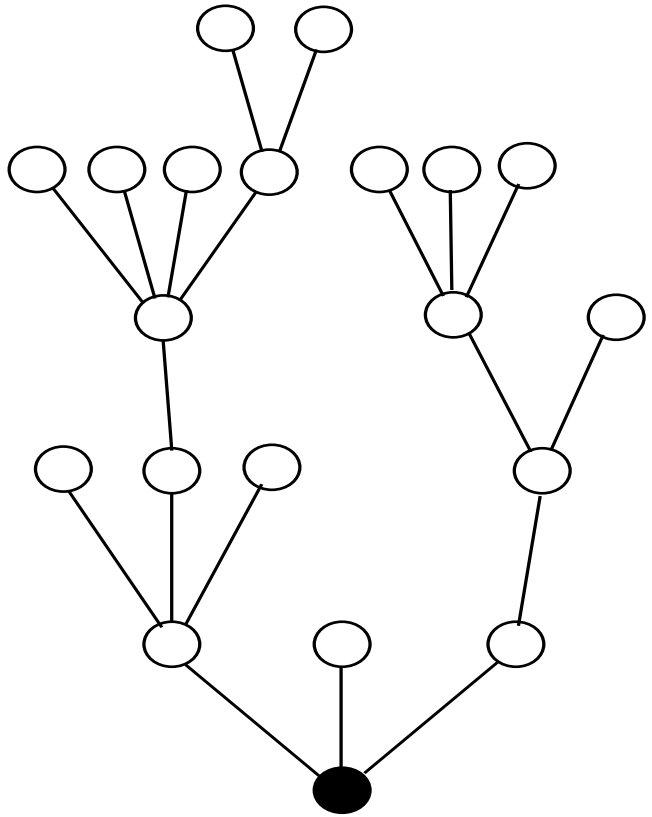
Die Länge des Weges $W(v, w)$ wird mit $d_T(v, w)$ bezeichnet und heißt *Abstand* zwischen v und w

Grundbegriffe der Graphentheorie

Spezielle Klassen von Bäumen: Wurzelbäume, ebene Wurzelbäume, Binärbäume, . . .

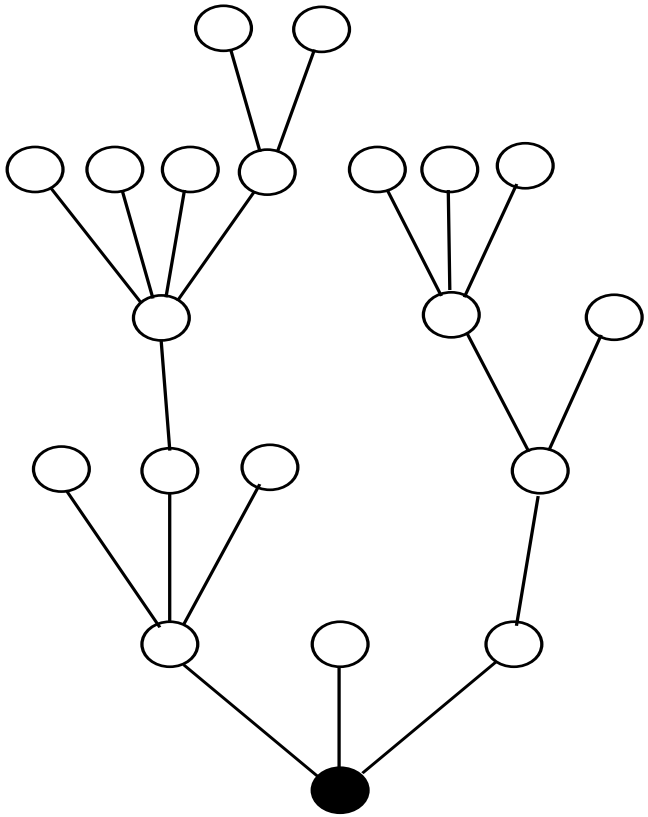
Grundbegriffe der Graphentheorie

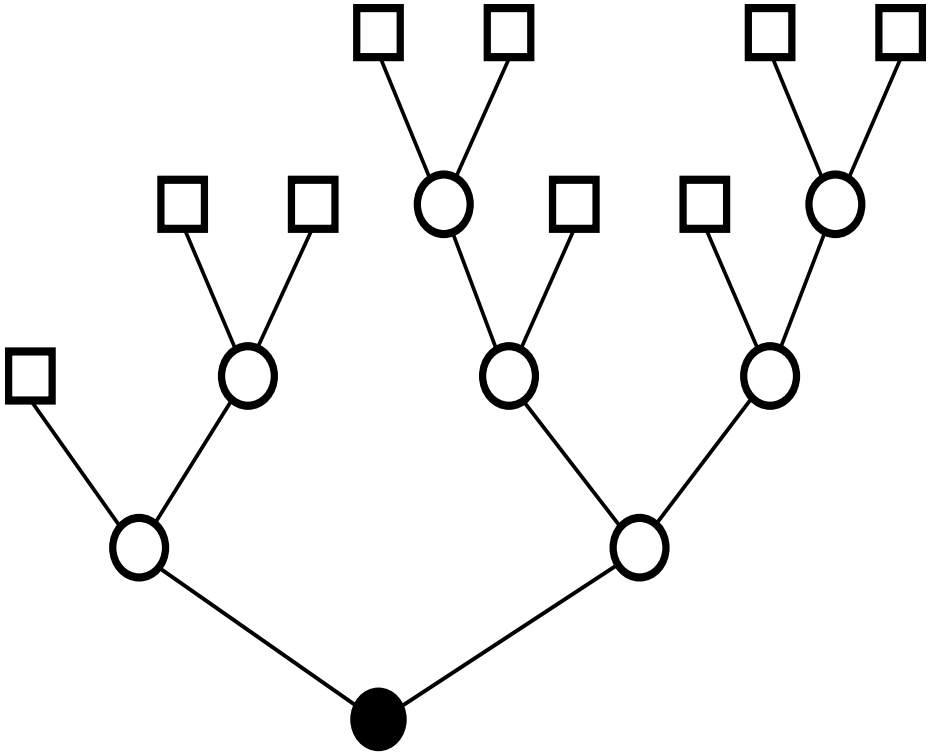
Spezielle Klassen von Bäumen: Wurzelbäume, ebene Wurzelbäume, Binärbäume, . . .



Grundbegriffe der Graphentheorie

Spezielle Klassen von Bäumen: Wurzelbäume, ebene Wurzelbäume, Binärbäume, . . .





Grundbegriffe der Graphentheorie

Ein Knoten v heißt *Endknoten*, wenn $d(v) = 1$.

Satz *Ein Baum mit mindestens zwei Knoten besitzt mindestens zwei Endknoten.*

Grundbegriffe der Graphentheorie

Ein Knoten v heißt *Endknoten*, wenn $d(v) = 1$.

Satz *Ein Baum mit mindestens zwei Knoten besitzt mindestens zwei Endknoten.*

Beweis: Man betrachte einen längsten Weg

$$v - v_1 - v_2 - v_3 - \cdots\cdots - v_k - w.$$

Dann müssen v und w Endknoten sein.

Grundbegriffe der Graphentheorie

Satz *Für einen Baum T gilt $\alpha_0(T) = \alpha_1(T) + 1$.*

Für einen Wald W mit k Zusammenhangskomponenten gilt $\alpha_0(W) = \alpha_1(W) + k$

Grundbegriffe der Graphentheorie

Satz *Für einen Baum T gilt $\alpha_0(T) = \alpha_1(T) + 1$.*

Für einen Wald W mit k Zusammenhangskomponenten gilt $\alpha_0(W) = \alpha_1(W) + k$

Beweis: vollständige Induktion nach $n = \alpha_0(T)$.

Induktionsanfang: $\alpha_0(T) = 1$, $\alpha_1(T) = 0$.

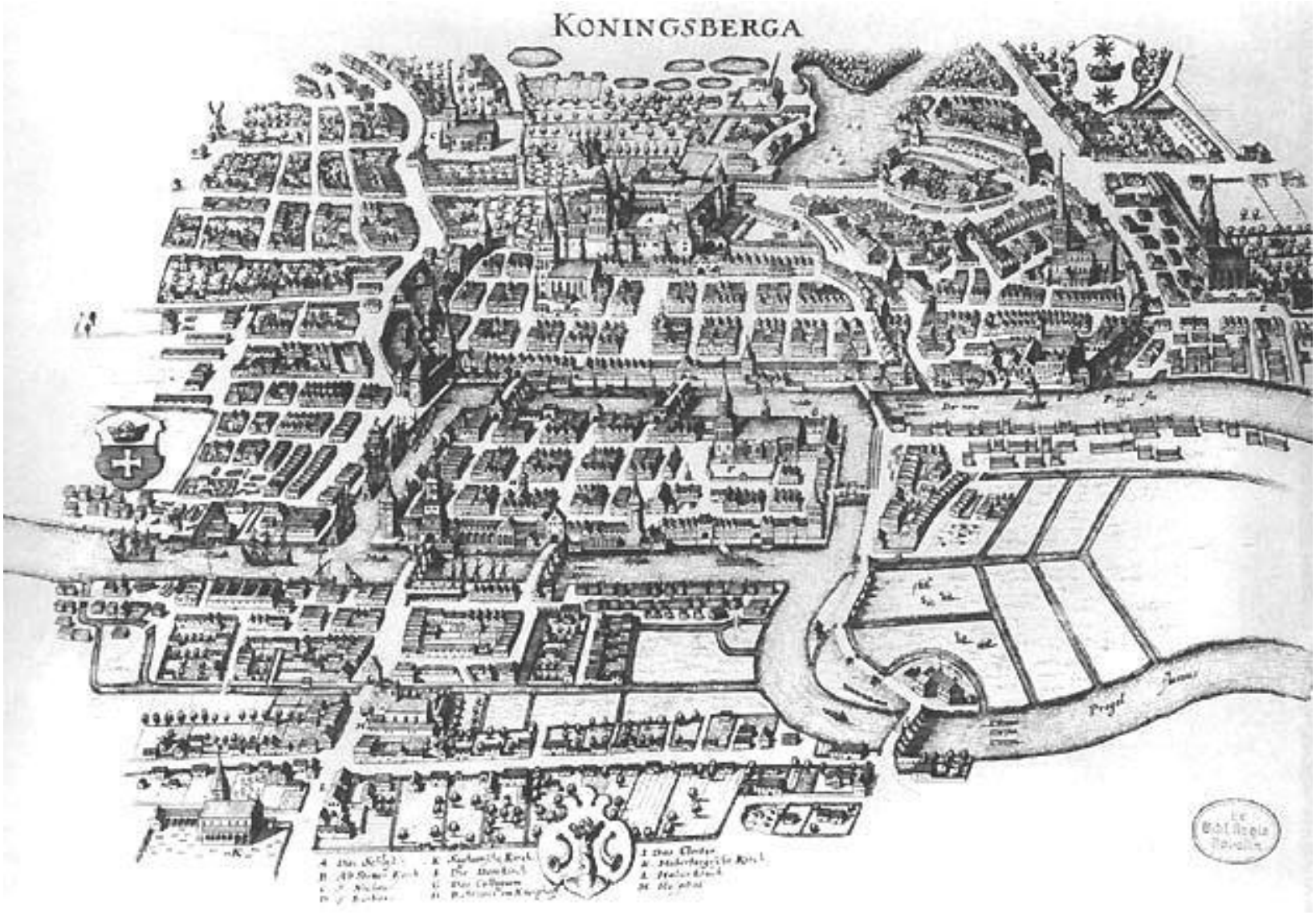
Man betrachte einen Baum $T = (V, E)$ mit $n + 1$ Knoten. Dann gibt es einen Endknoten v mit inzidenter Kante e. Sei $T' = (V \setminus \{v\}, E \setminus \{e\}$.

Wegen $\alpha_0(T') = n$ kann die Induktionsvoraussetzung angewendet werden.

EULERSCHE LINIEN

Eulersche Linien

Das Königsberger Brückenproblem:



Eulersche Linien

Eine geschlossene Kantenfolge (d.h. Startknoten = Zielknoten), die jede Kante von $G = (V, E)$ genau einmal enthält, heißt *Eulersche Linie*. Ein Graph, der eine Eulersche Linie besitzt, heißt *Eulerscher Graph*.

Variante: Unter einer offenen Eulerschen Linie versteht man eine offene Kantenfolge (Startknoten $\neq$ Zielknoten), die jede Kante von $G = (V, E)$ genau einmal enthält.

Eulersche Linien

Eine geschlossene Kantenfolge (d.h. Startknoten = Zielknoten), die jede Kante von $G = (V, E)$ genau einmal enthält, heißt *Eulersche Linie*. Ein Graph, der eine Eulersche Linie besitzt, heißt *Eulerscher Graph*.

Variante: Unter einer offenen Eulerschen Linie versteht man eine offene Kantenfolge (Startknoten $\neq$ Zielknoten), die jede Kante von $G = (V, E)$ genau einmal enthält.

Satz *Ein ungerichteter zusammenhängender Graph ist genau dann Eulersch, wenn alle seine Knoten geraden Grad haben.*

Ein ungerichteter zusammenhängender Graph besitzt genau dann eine offene Eulersche Linie, wenn er genau zwei Knoten ungeraden Grades hat.

Eulersche Linien

Eine geschlossene Kantenfolge (d.h. Startknoten = Zielknoten), die jede Kante von $G = (V, E)$ genau einmal enthält, heißt *Eulersche Linie*. Ein Graph, der eine Eulersche Linie besitzt, heißt *Eulerscher Graph*.

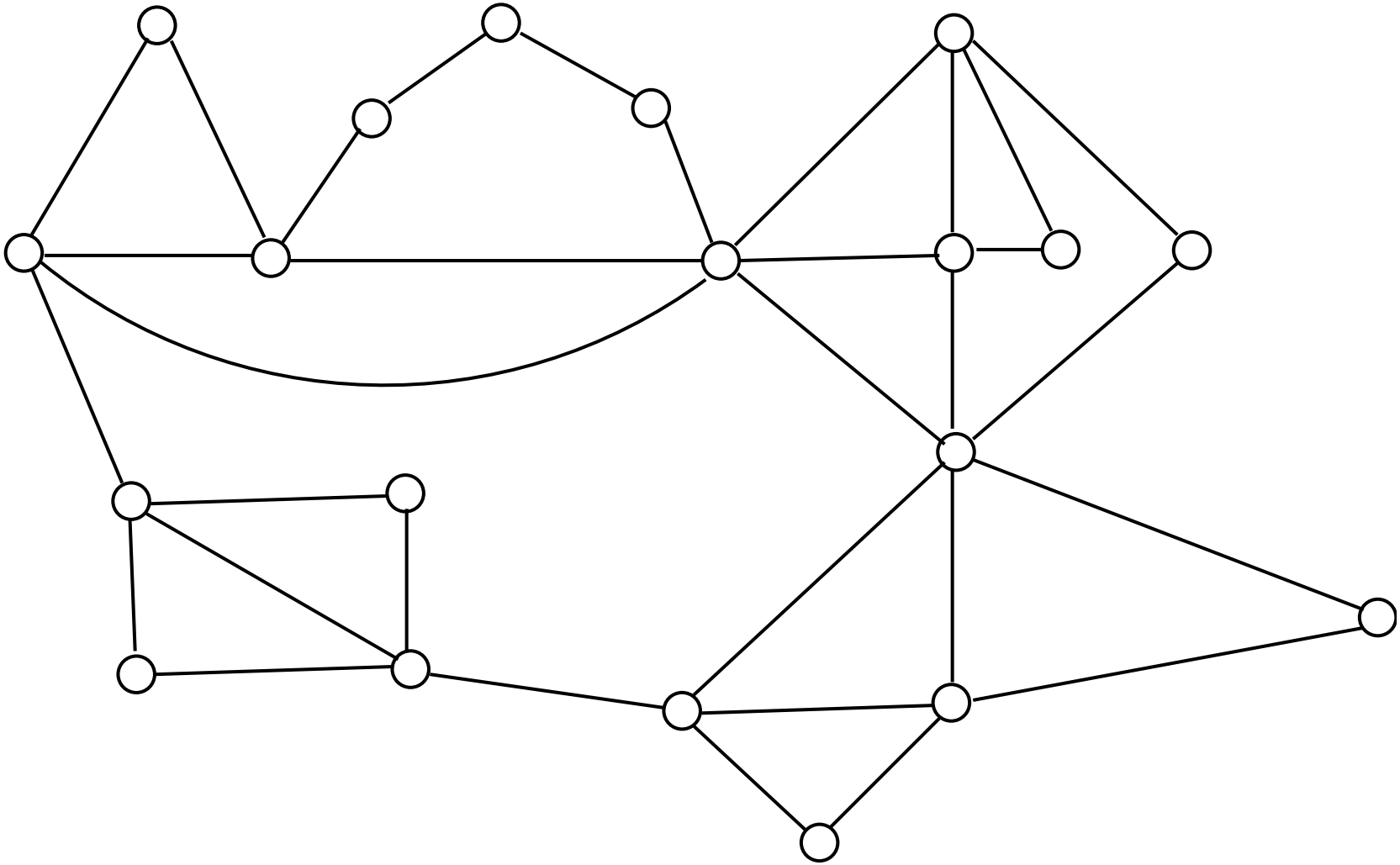
Variante: Unter einer offenen Eulerschen Linie versteht man eine offene Kantenfolge (Startknoten $\neq$ Zielknoten), die jede Kante von $G = (V, E)$ genau einmal enthält.

Satz *Ein ungerichteter zusammenhängender Graph ist genau dann Eulersch, wenn alle seine Knoten geraden Grad haben.*

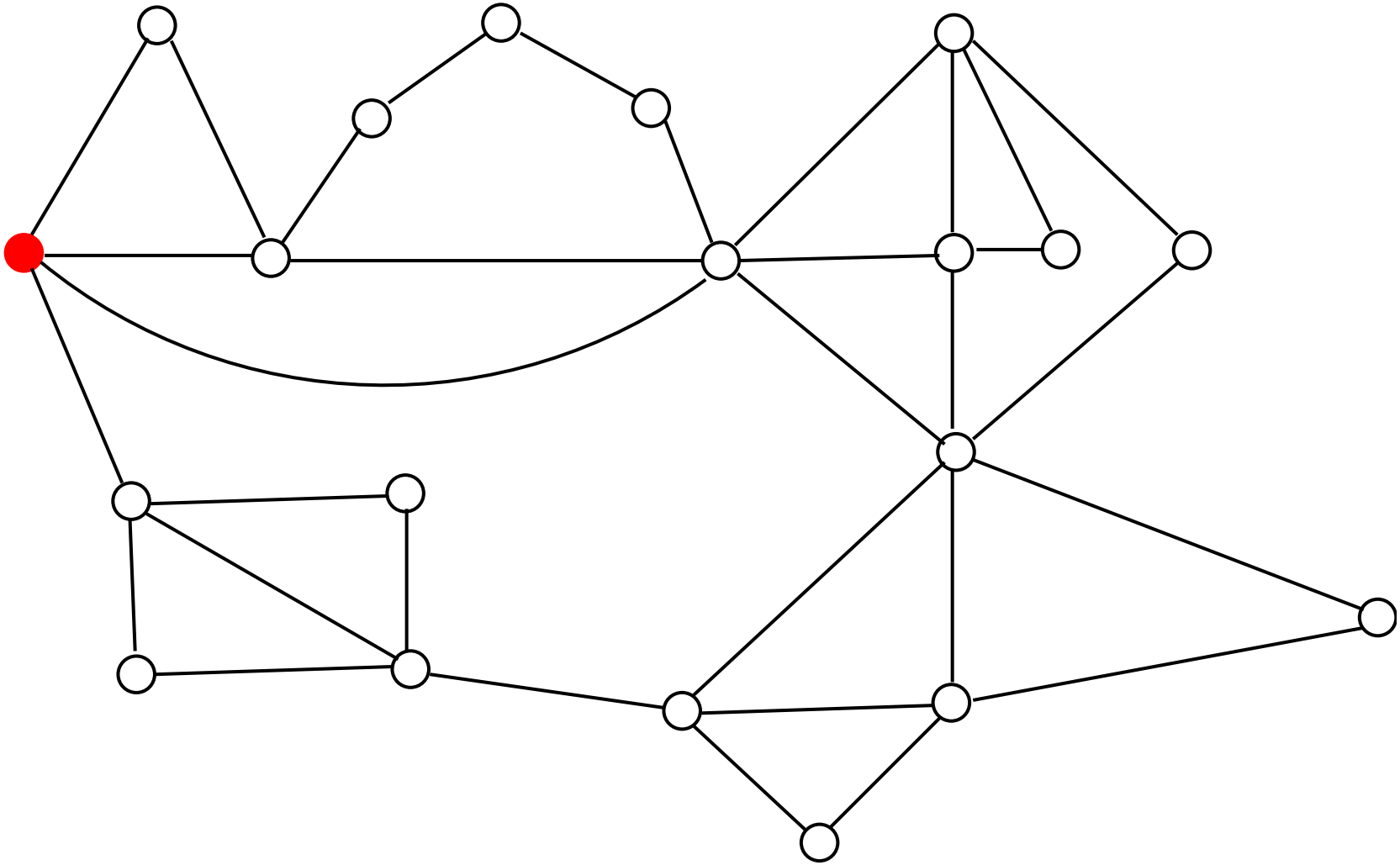
Ein ungerichteter zusammenhängender Graph besitzt genau dann eine offene Eulersche Linie, wenn er genau zwei Knoten ungeraden Grades hat.

Beweis: Induktion nach der Anzahl der Kanten.

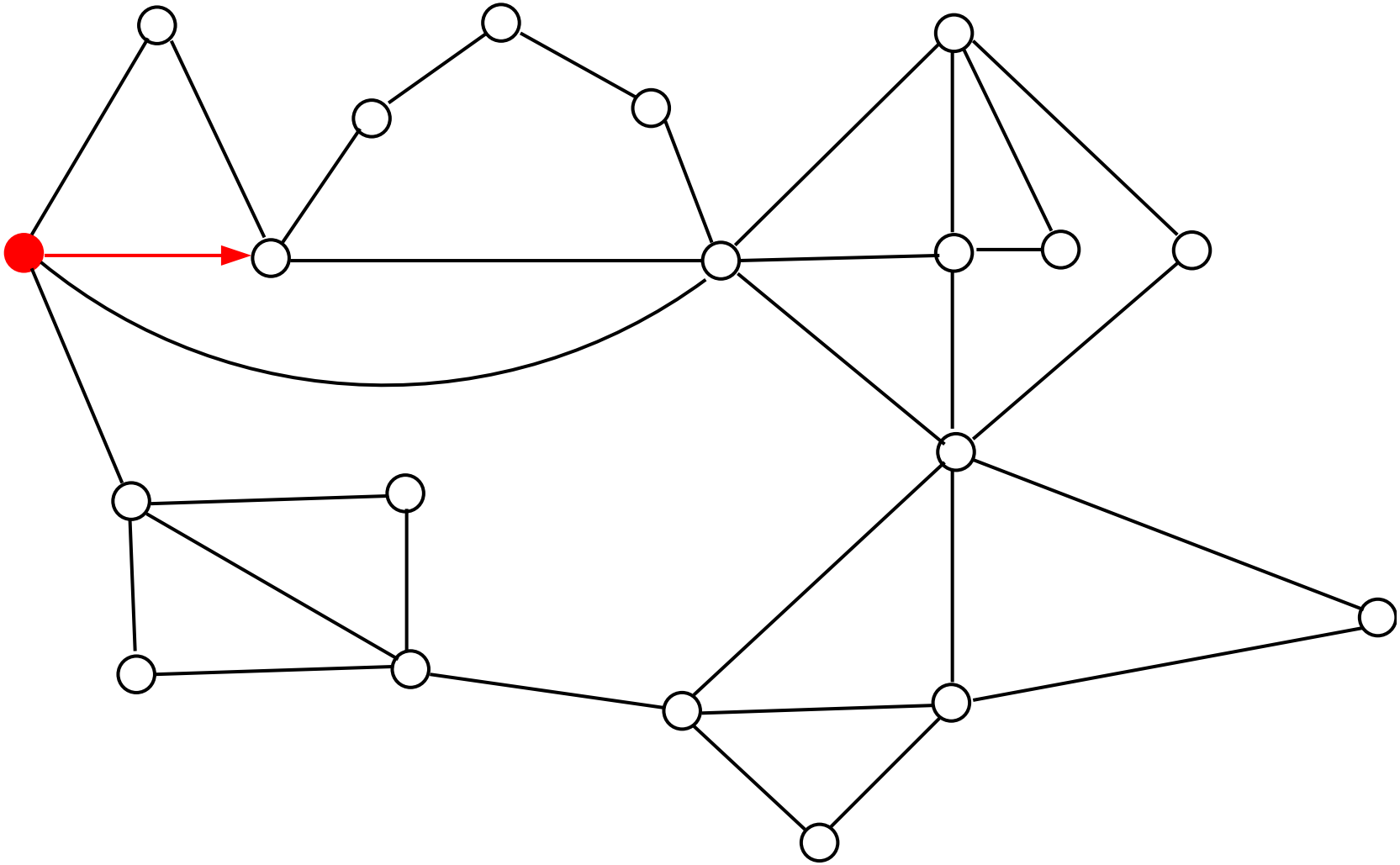
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



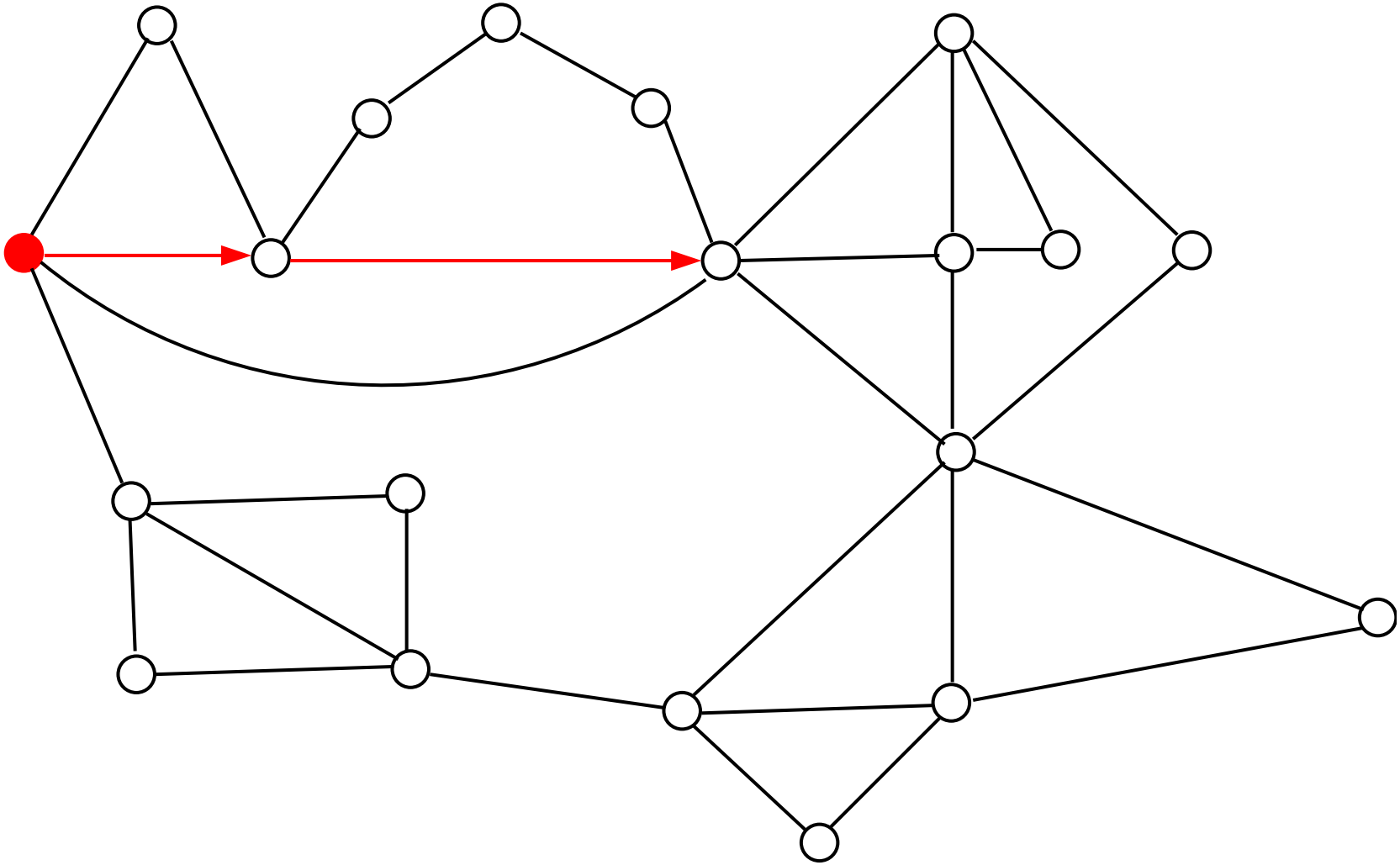
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



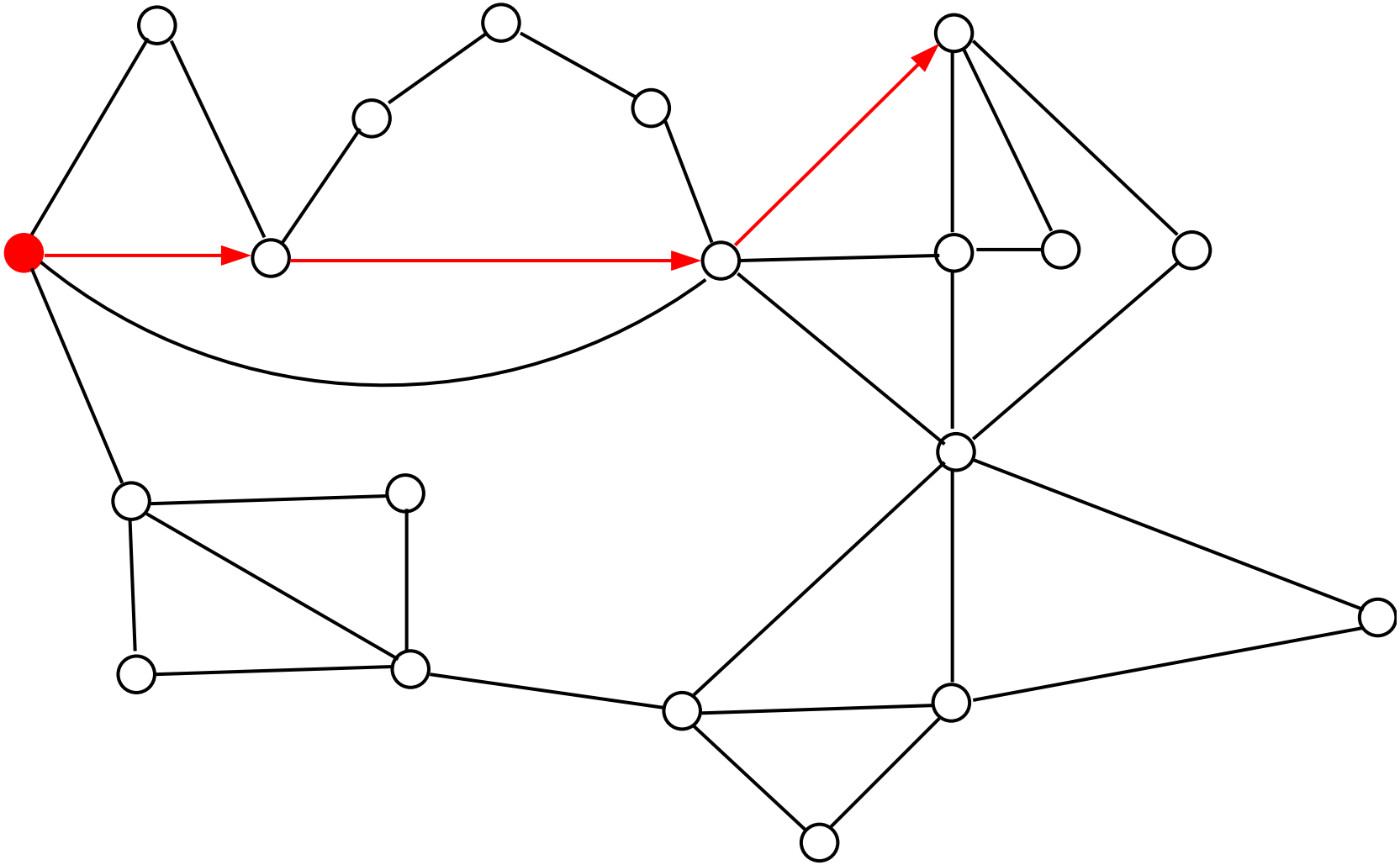
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



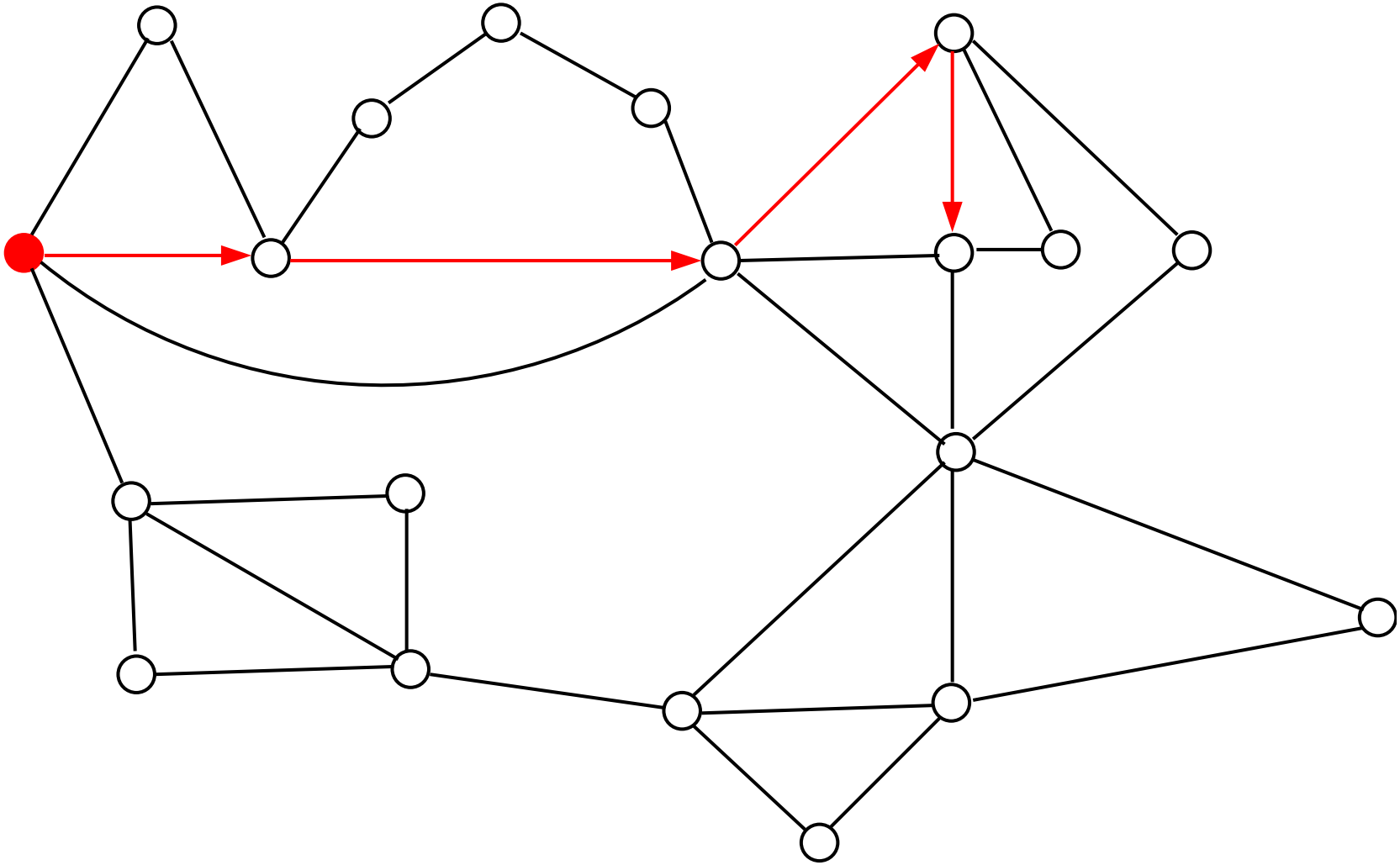
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



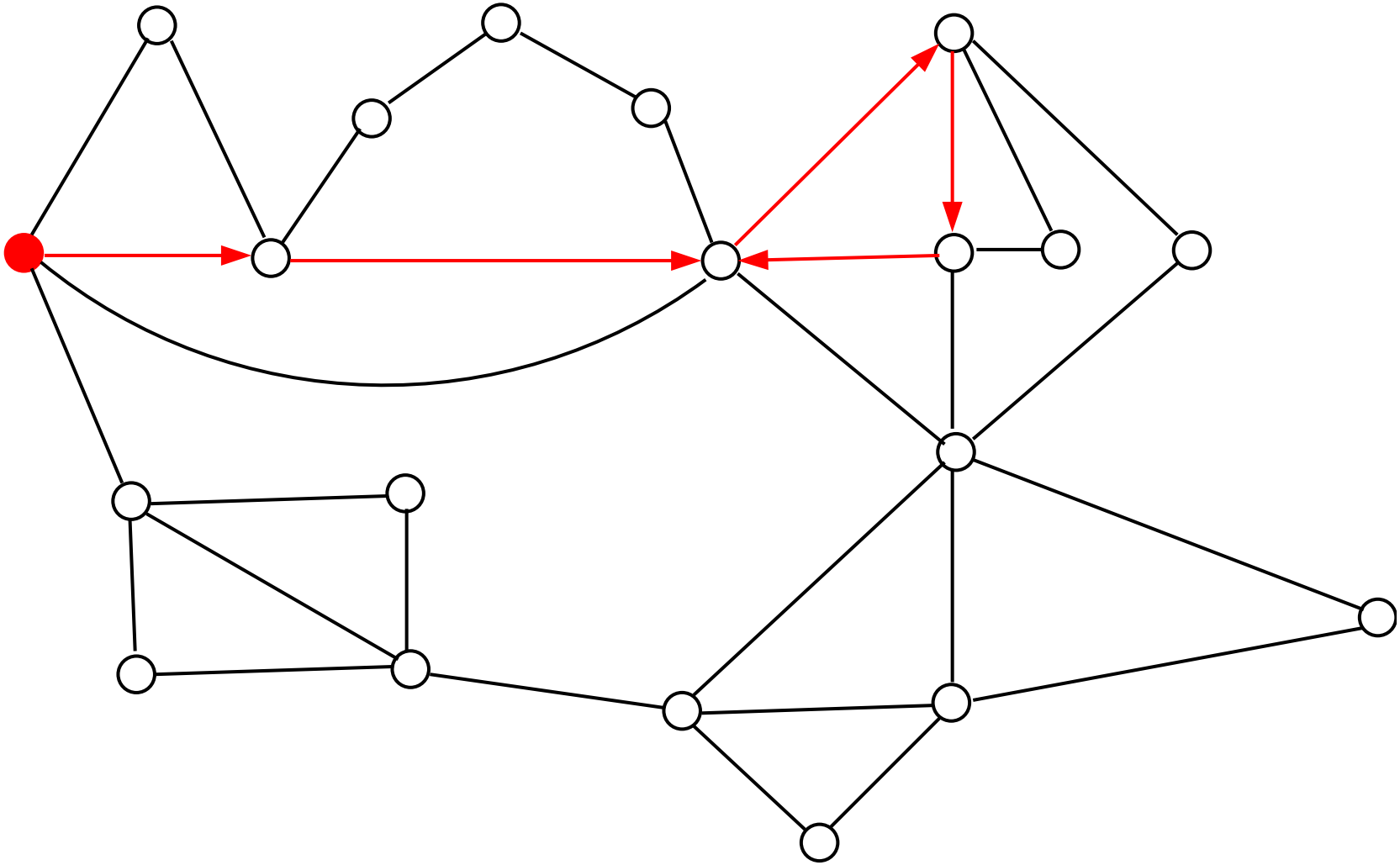
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



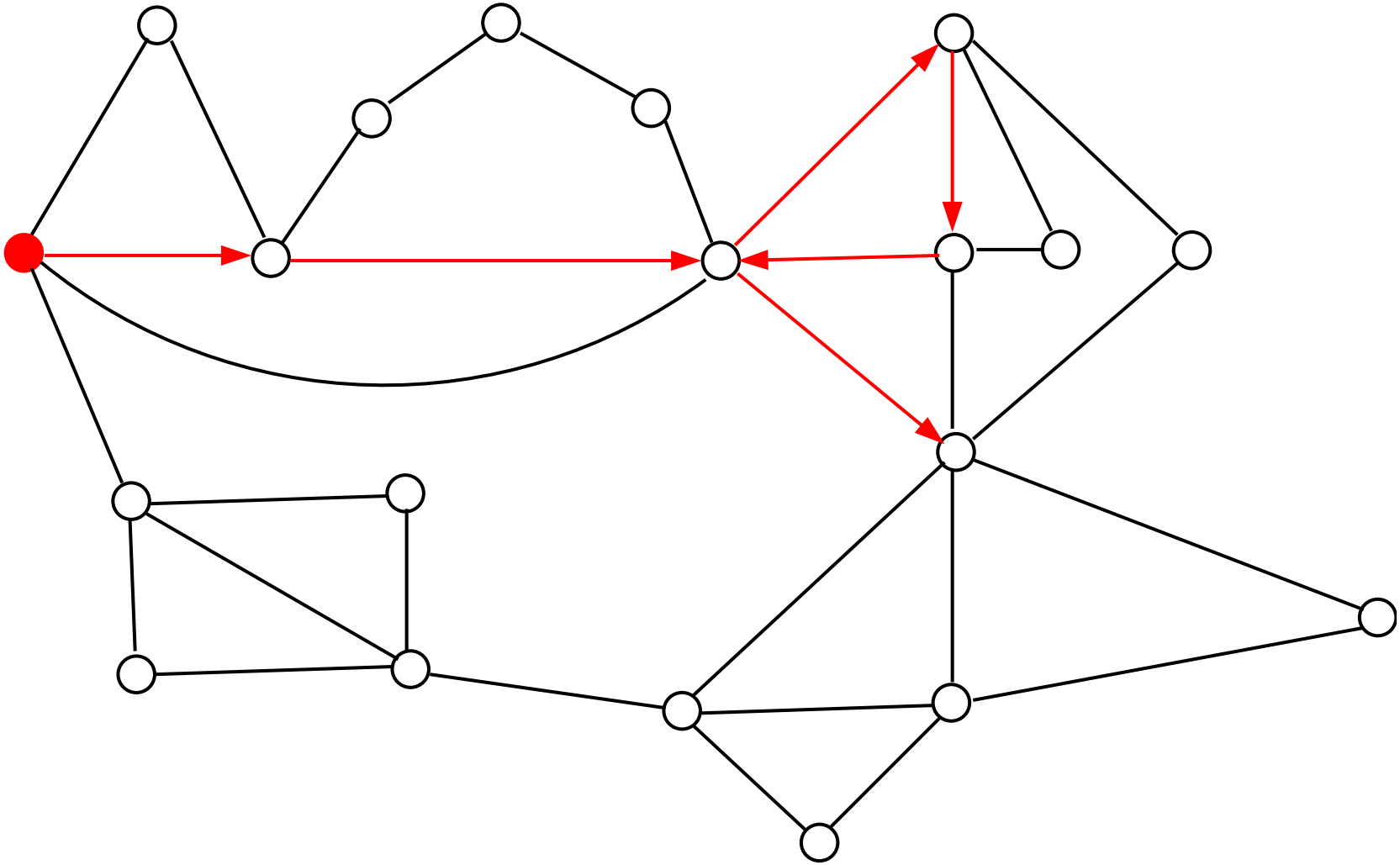
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



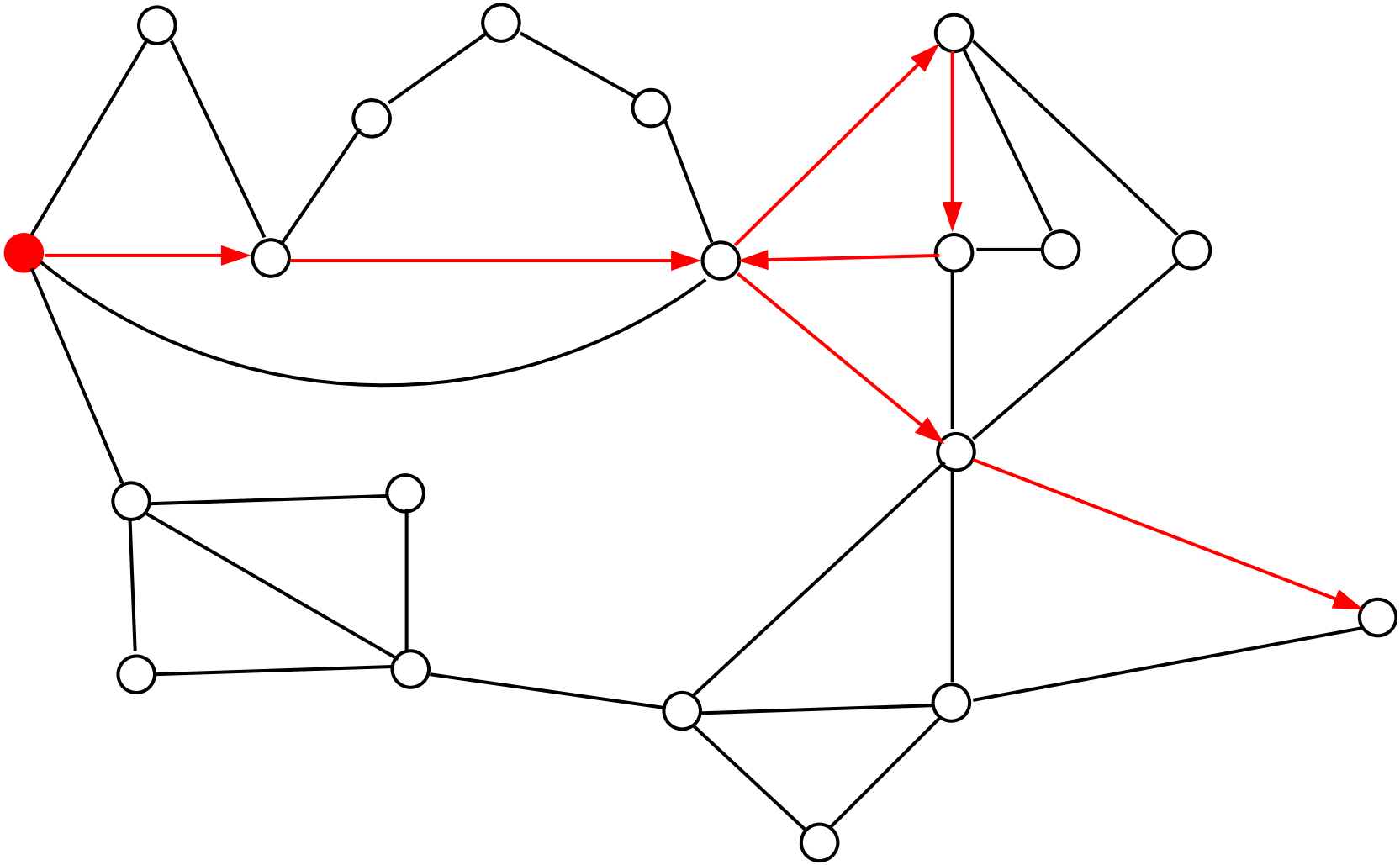
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



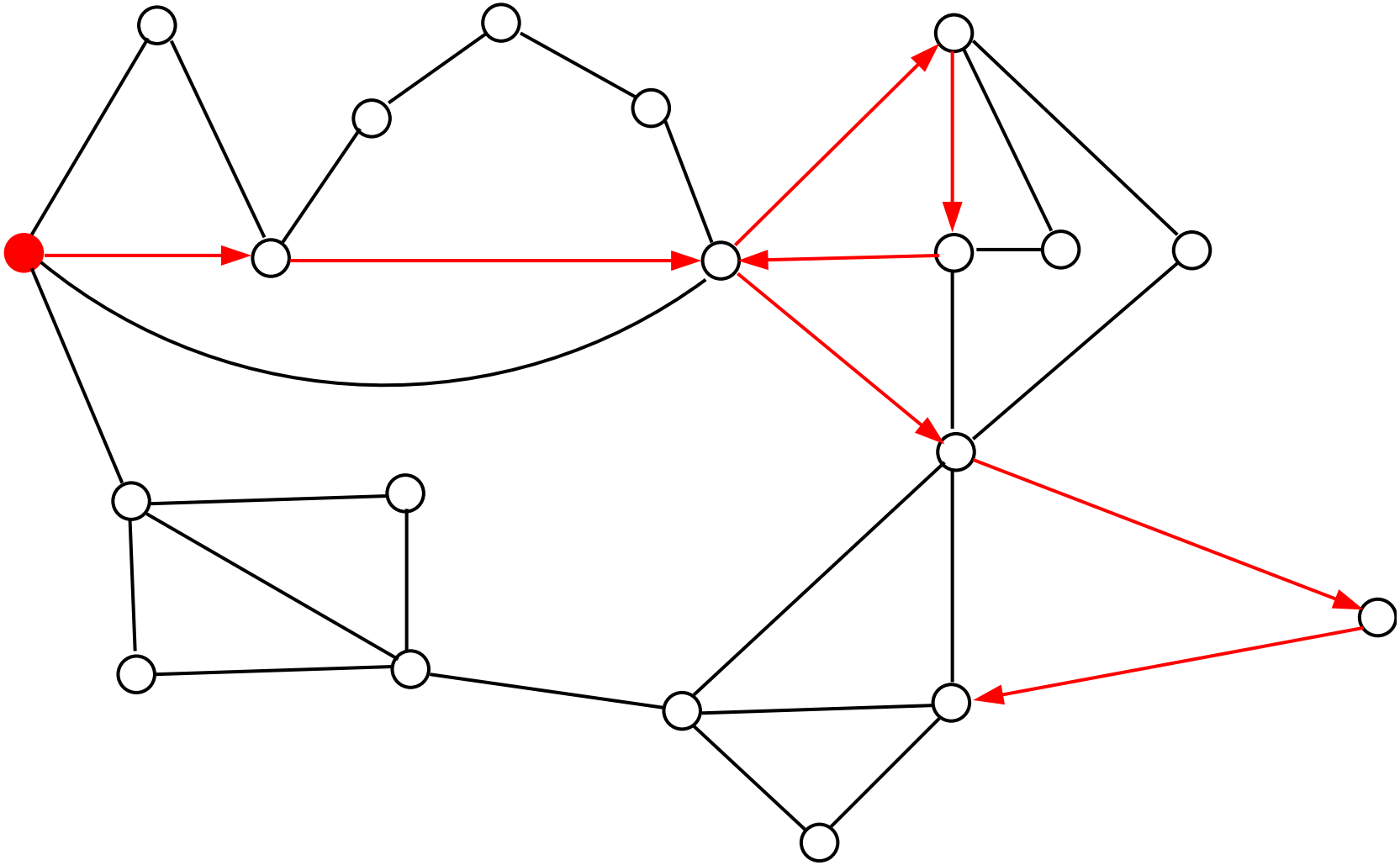
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



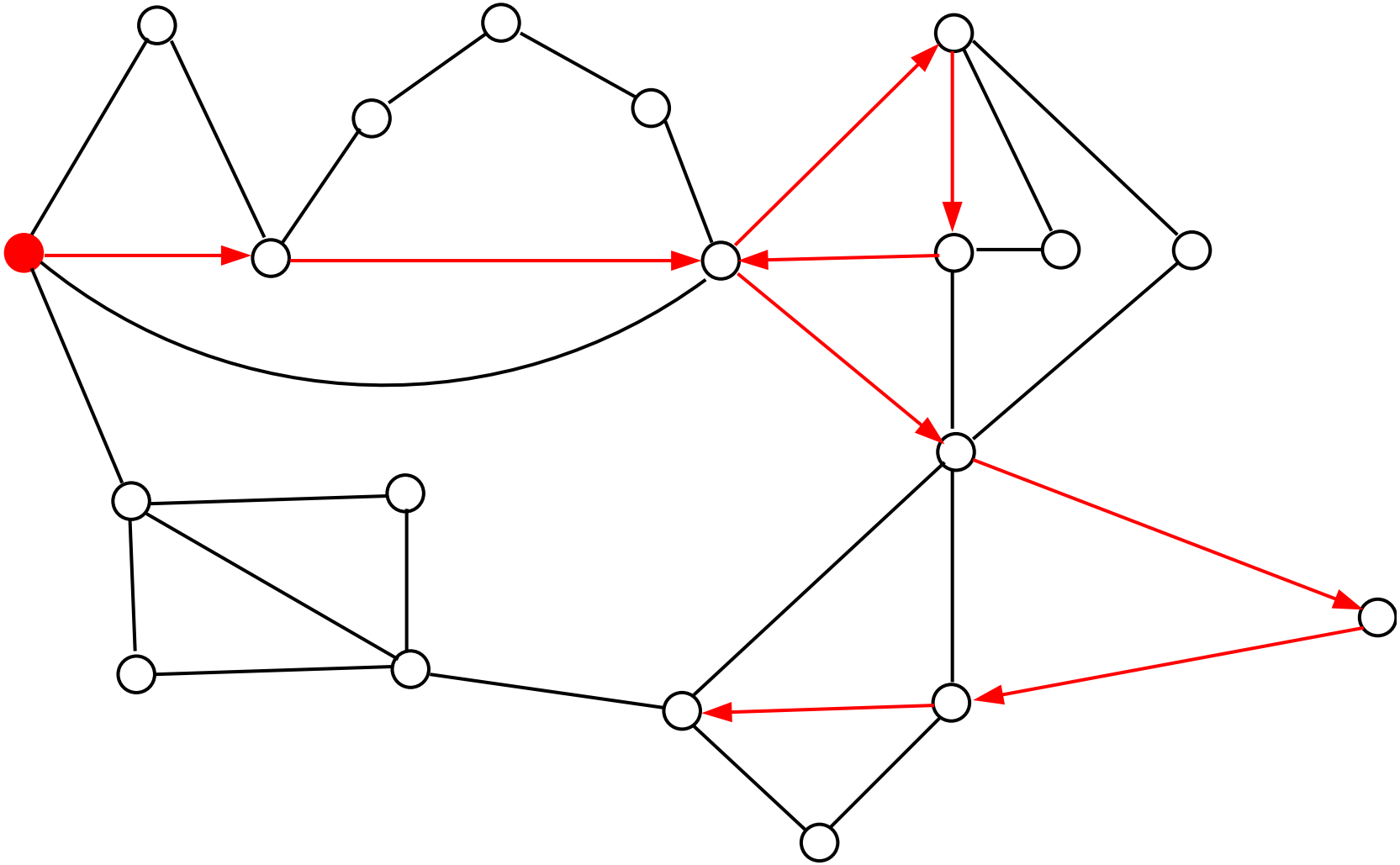
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



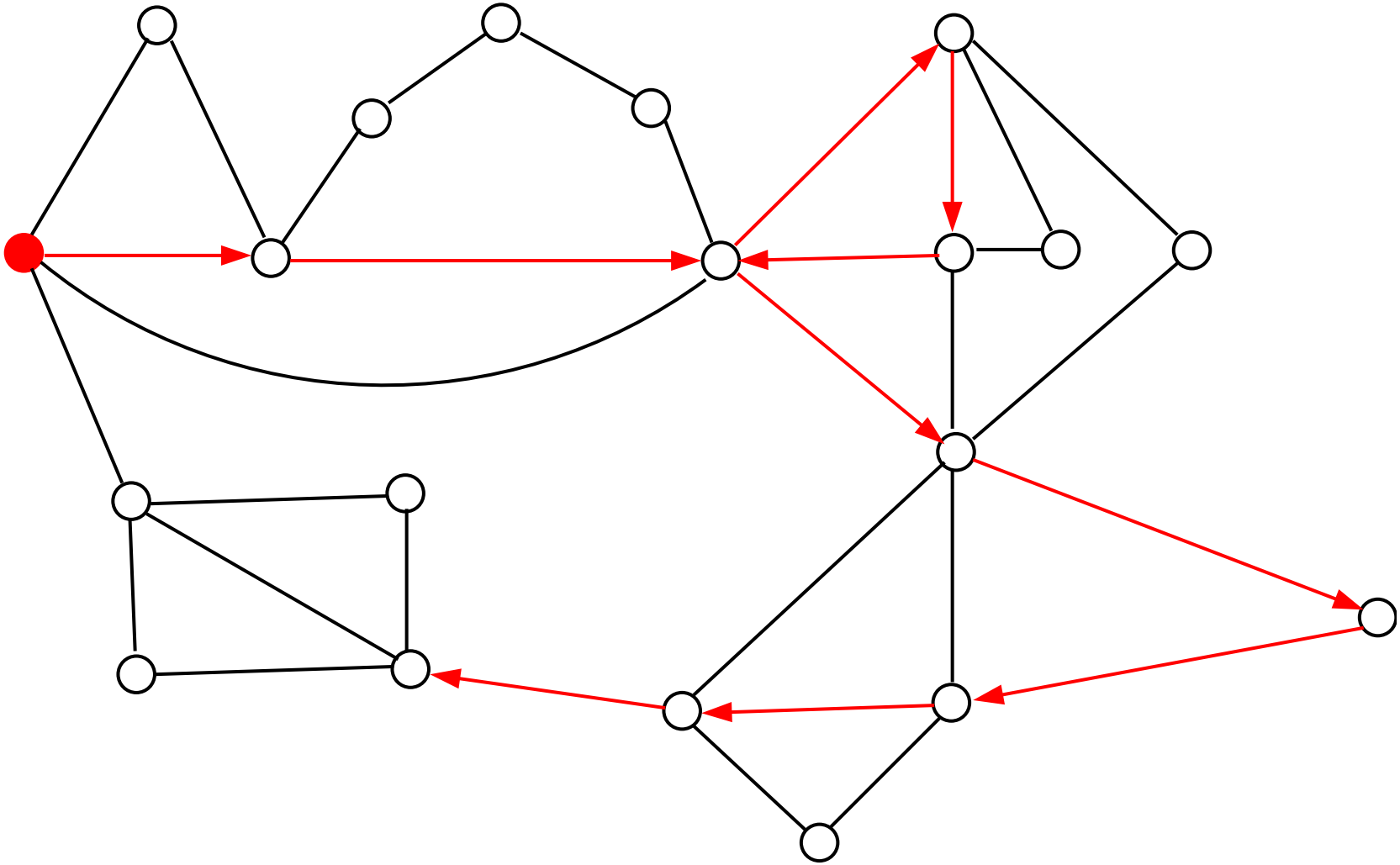
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



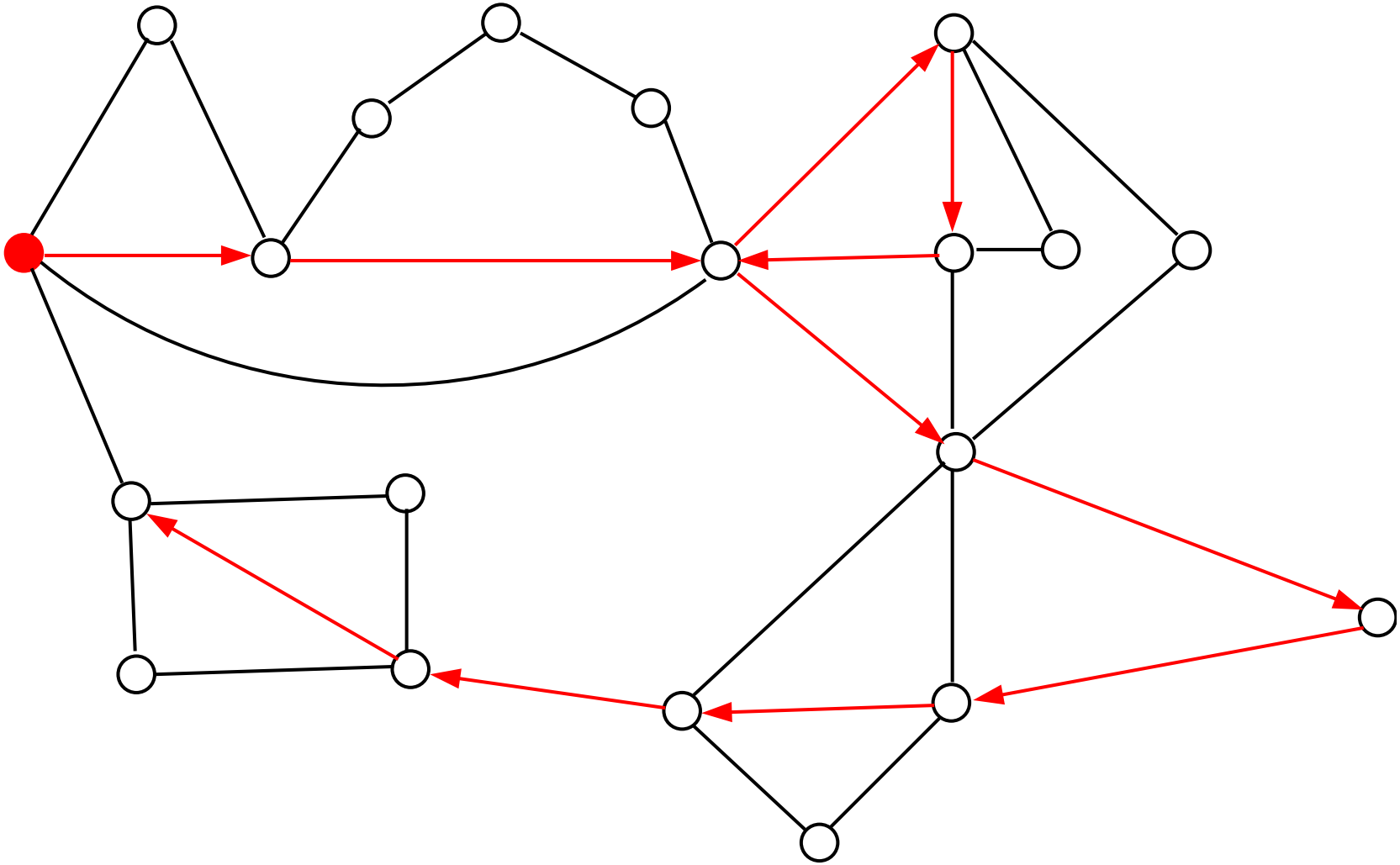
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



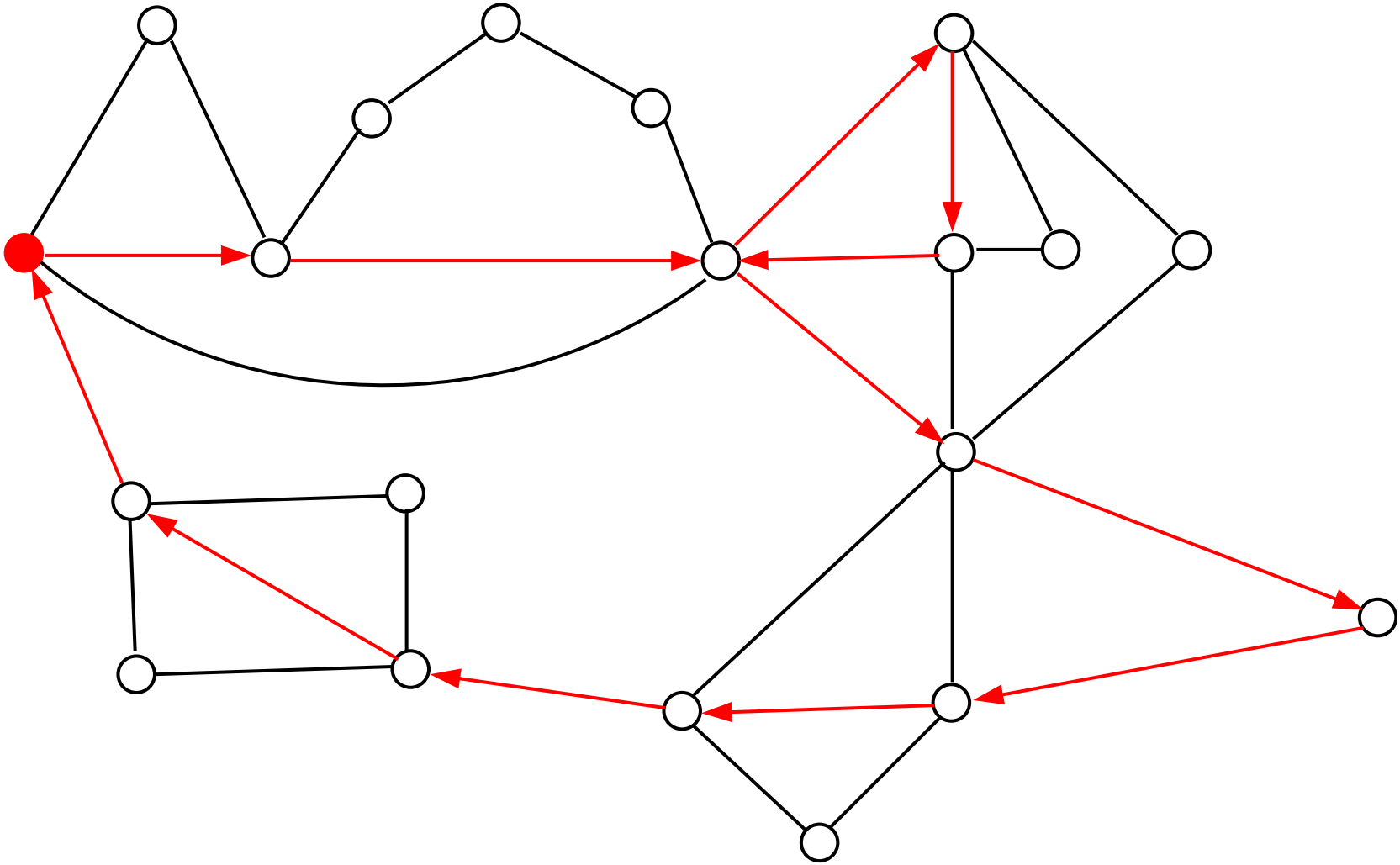
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



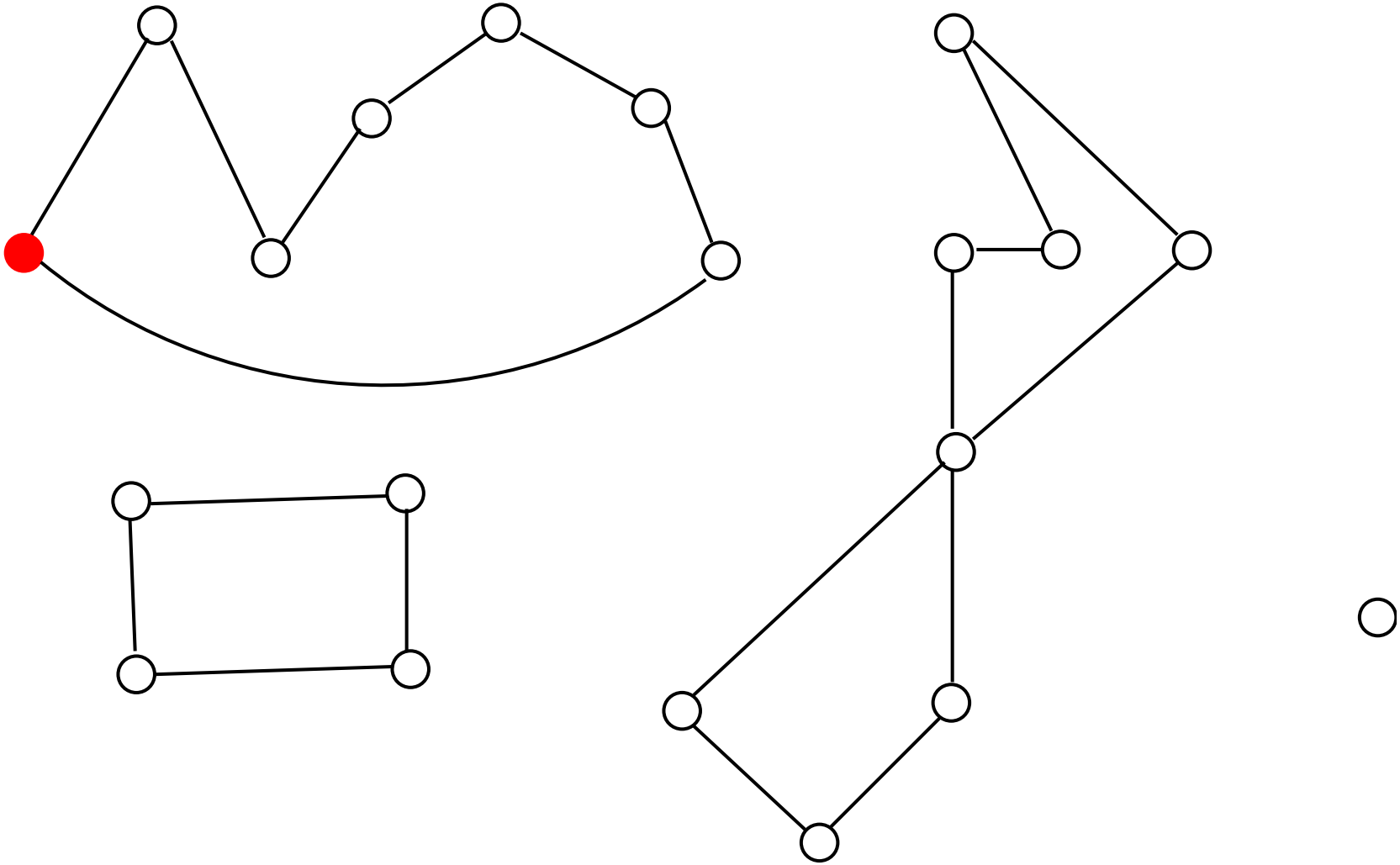
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



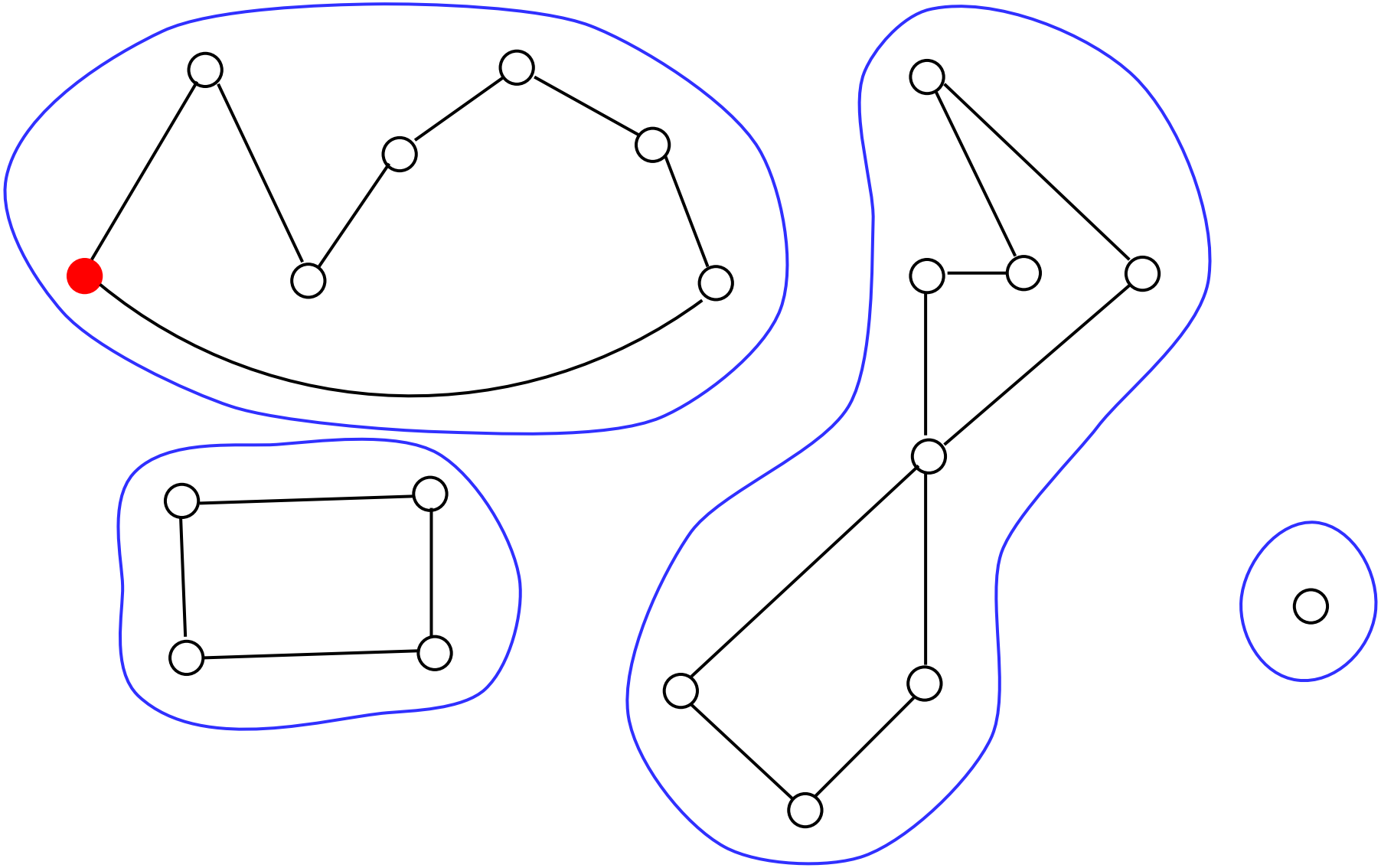
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



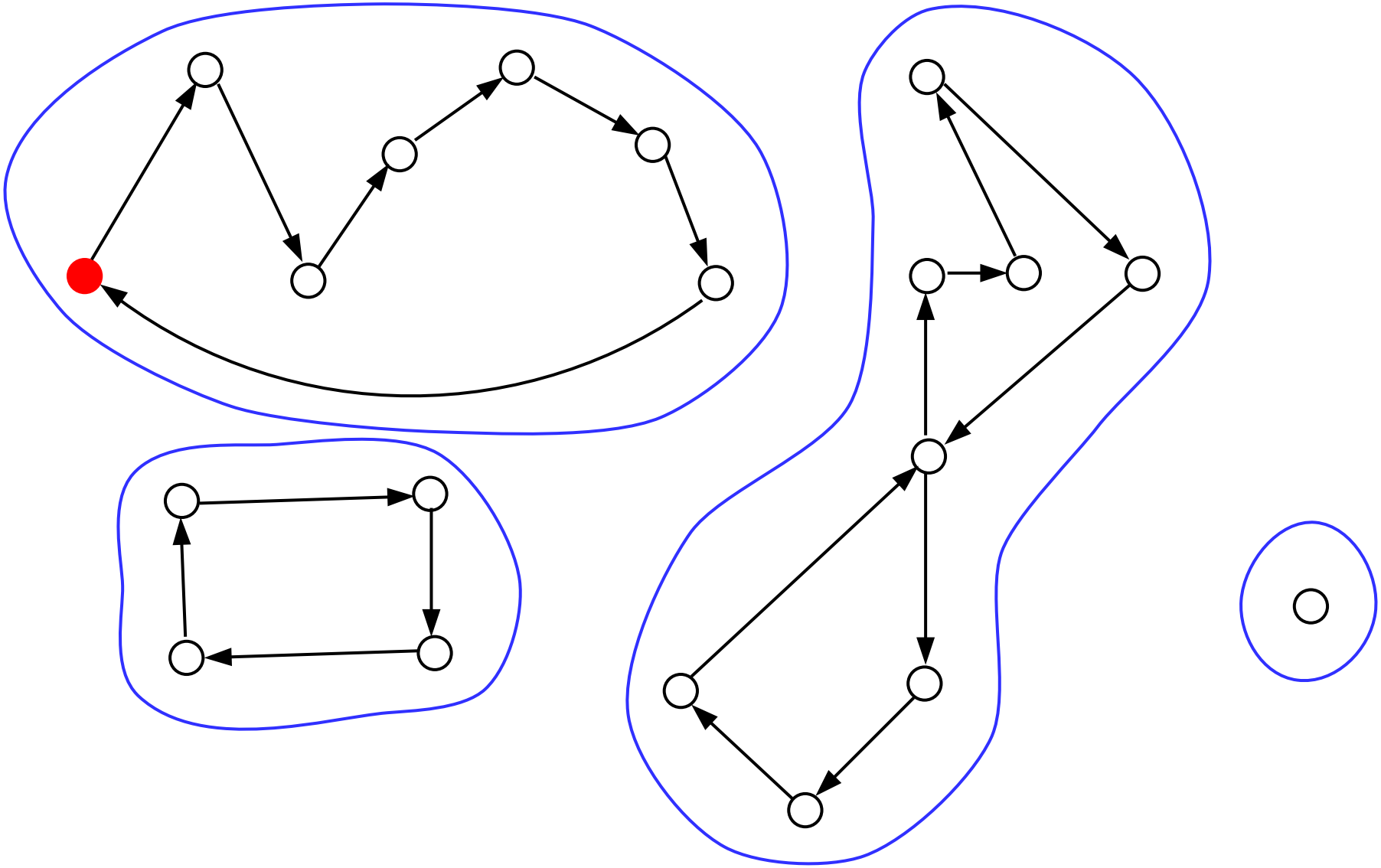
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



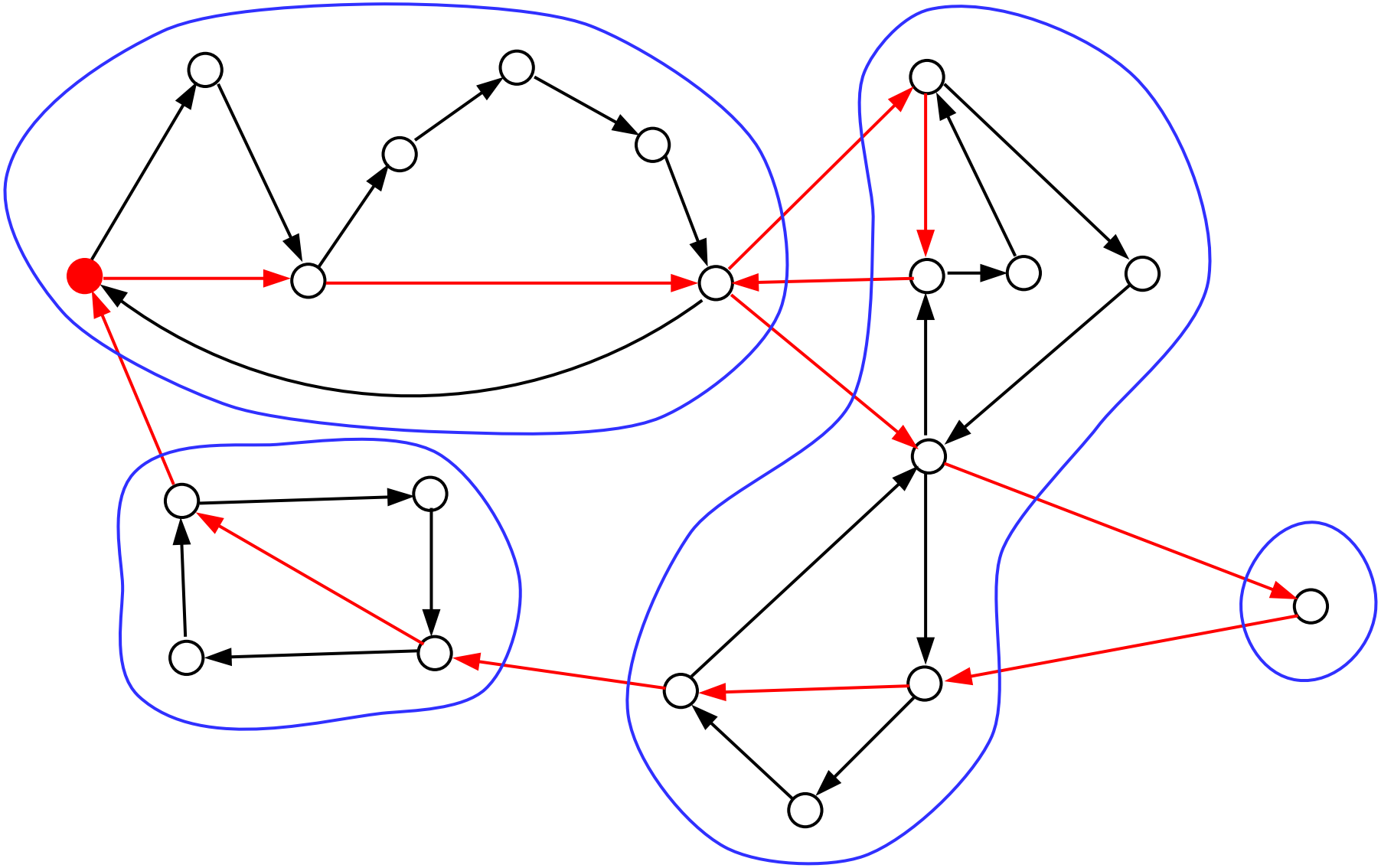
Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



Beweisidee zum Kriterium für die Existenz einer geschlossenen Eulerschen Linie



Eulersche Linien in gerichteten Graphen

Satz *Ein gerichteter, schwach zusammenhängender Graph $G = (V, E)$ ist genau dann Eulersch, wenn für alle seine Knoten Weggrad und Hingrad übereinstimmen, d.h.*

$$\forall\, x \in V :\ d^+(x) = d^-(x).$$

Ein gerichteter, schwach zusammenhängender Graph $G = (V, E)$ besitzt genau dann eine offene Eulersche Linie, wenn es Knoten $x, y \in V$ gibt, sodass

$$\begin{aligned} &d^+(x) = d^-(x) + 1, \\ &d^+(y) = d^-(y) - 1, \\ &\forall\, z \in V \setminus \{x, y\} :\ d^+(z) = d^-(z). \end{aligned}$$

Hamiltonsche Linien

Eine geschlossene Kantenfolge eines Graphen heißt *Hamiltonsche Linie*, wenn sie alle Knoten genau einmal durchläuft.

Ein Graph, der eine Hamiltonsche Linie besitzt, heißt *Hamiltonscher Graph*.

Hamiltonsche Linien

Eine geschlossene Kantenfolge eines Graphen heißt *Hamiltonsche Linie*, wenn sie alle Knoten genau einmal durchläuft.

Ein Graph, der eine Hamiltonsche Linie besitzt, heißt *Hamiltonscher Graph*.

Satz (Satz von Dirac) *Jeder Graph mit n Knoten, in dem jeder Knoten mindestens den Grad $n/2$ hat, ist Hamiltonsch.*

Satz (Satz von Ore) *Jeder Graph mit n Knoten, in dem die Summe der Knotengrade von je zwei nicht-adjazenten Knoten mindestens n ist, ist Hamiltonsch.*

PLANARE GRAPHEN

Planare Graphen

Zwei Graphen $G = (V_G, E_G)$ und $H = (V_H, E_H)$ heißen *isomorph*, in Zeichen $G \cong H$, wenn es eine bijektive Abbildung $f : V_G \to V_H$ gibt, die die Adjazenz erhält, d.h.

$$\forall\, x, y \in V_G :\ xy \in E_G \Longrightarrow f(x)f(y) \in E_H.$$

Planare Graphen

Zwei Graphen $G = (V_G, E_G)$ und $H = (V_H, E_H)$ heißen *isomorph*, in Zeichen $G \cong H$, wenn es eine bijektive Abbildung $f : V_G \to V_H$ gibt, die die Adjazenz erhält, d.h.

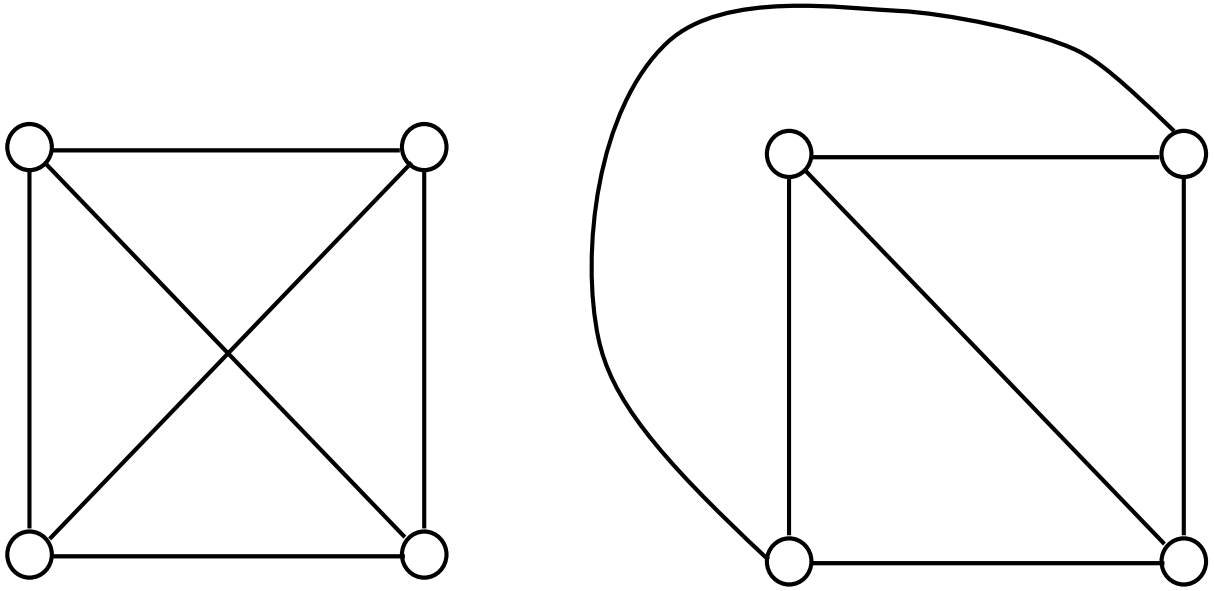
$$\forall\, x, y \in V_G :\ xy \in E_G \Longrightarrow f(x)f(y) \in E_H.$$

Ein Graph $G = (V, E)$ heißt *ebener Graph*, wenn $V \subseteq \mathbb{R}^2$, jede Kante eine einfache Kurve (zB ein Polygonzug) ist, die zwei Knoten verbindet, und keine zwei Kanten sich überschneiden.

Ein Graph G heißt *planarer Graph*, wenn es einen ebenen Graphen H gibt, sodass $G \cong H$.

Planare Graphen

Beispiel eines planaren Graphen:



Die Knoten und Kanten eines ebenen Graphen G umschließen Gebiete im $\mathbb{R}^2$, welche *Gebiete* von G genannt werden. Deren Anzahl wird mit $\alpha_2(G)$ bezeichnet.

Planare Graphen

Satz (Eulersche Polyederformel) *In einem planaren Graphen G gilt die folgende Gleichung:* $\alpha_0(G) - \alpha_1(G) + \alpha_2(G) = 2$

Beweis: Wenn $\alpha_2(G) > 1$, Kanten entfernen, sodass immer zwei Gebiete zusammenfallen.

Nach $\alpha_2(G) - 1$ Schritten: Graph G', $\alpha_2(G') = 1$, also ist G' ein Baum und daher gilt $\alpha_0(G') = \alpha_1(G') - 1$.

$$\Longrightarrow \qquad \underbrace{\alpha_0(G) - 1}_{\text{übrige Kanten}} + \underbrace{\alpha_2(G) - 1}_{\text{entfernte Kanten}} = \alpha_1(G)$$

Planare Graphen

Satz (Eulersche Polyederformel) *In einem planaren Graphen G gilt die folgende Gleichung:* $\alpha_0(G) - \alpha_1(G) + \alpha_2(G) = 2$

Beweis: Wenn $\alpha_2(G) > 1$, Kanten entfernen, sodass immer zwei Gebiete zusammenfallen.

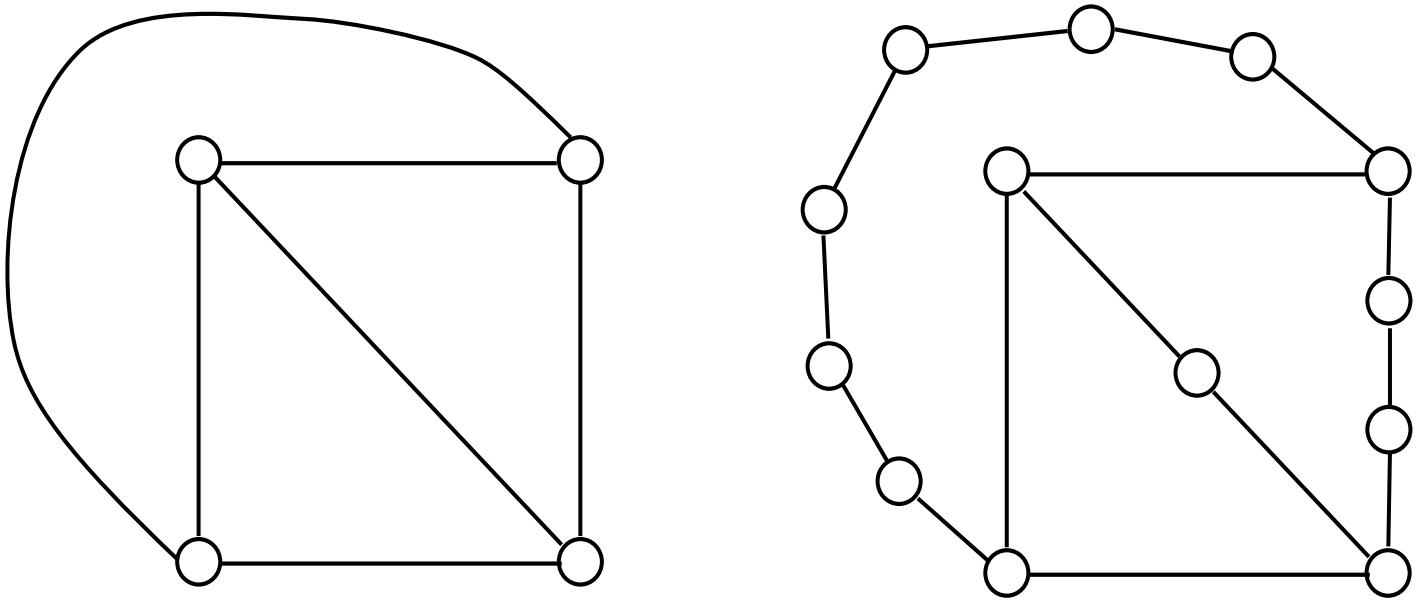
Nach $\alpha_2(G) - 1$ Schritten: Graph G', $\alpha_2(G') = 1$, also ist G' ein Baum und daher gilt $\alpha_0(G') = \alpha_1(G') - 1$.

$$\Longrightarrow \qquad \underbrace{\alpha_0(G) - 1}_{\text{übrige Kanten}} + \underbrace{\alpha_2(G) - 1}_{\text{entfernte Kanten}} = \alpha_1(G)$$

Folgerung: In einem planaren Graphen gilt $\alpha_1 \leq 3\alpha_0 - 6$. Daher ist der K_5 nicht planar.

Planare Graphen

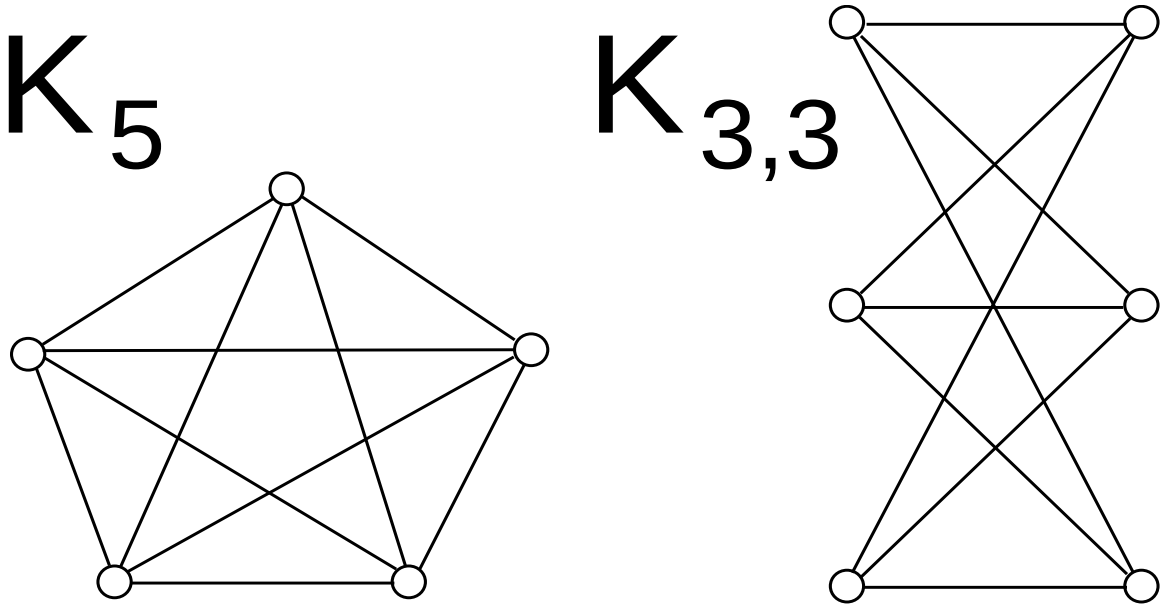
Ein Graph G' heißt Unterteilung von G, falls



Ein Graph H heißt topologischer Minor eines Graphen G, falls es eine Unterteilung H' von H gibt, sodass H' ein Teilgraph von G ist.

Planare Graphen

Satz (Satz von Kuratowski) *Ein Graph G ist genau dann planar, wenn weder K_5 noch $K_{3,3}$ topologische Minoren von G sind.*



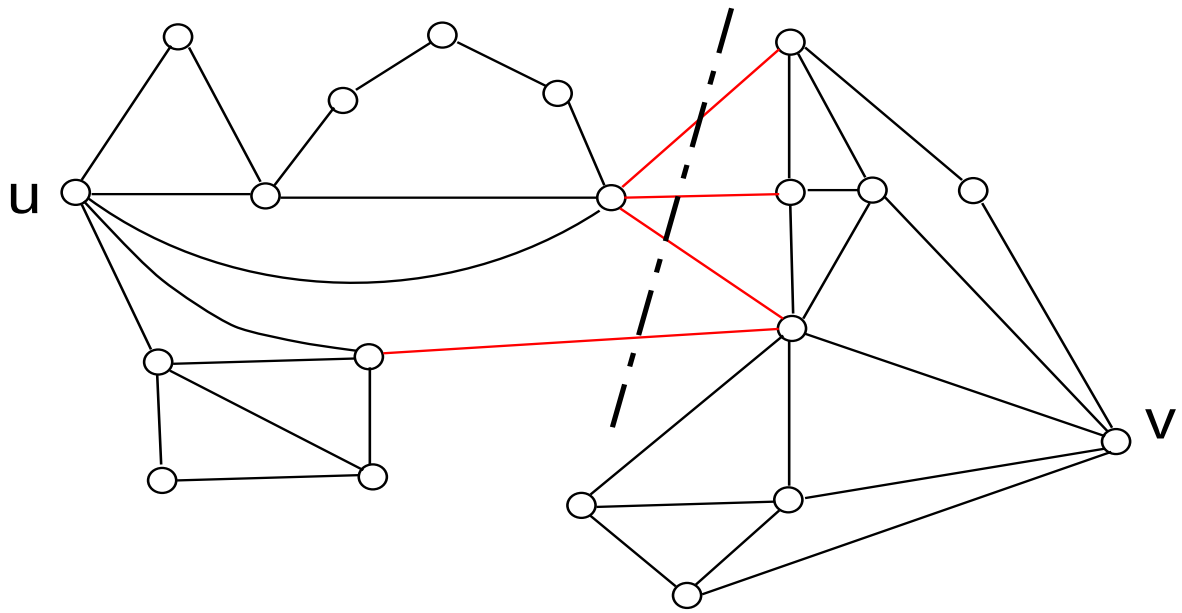
SCHNITTE UND DER SATZ VON MENGER

Schnitte und der Satz von Menger

Sei $G = (V, E)$ ein ungerichteter Graph, zwei Knoten $u, v \in V$ und eine Zerlegung $V = U \cup (V \setminus U)$ derart, dass $u \in U$ und $v \notin U$.

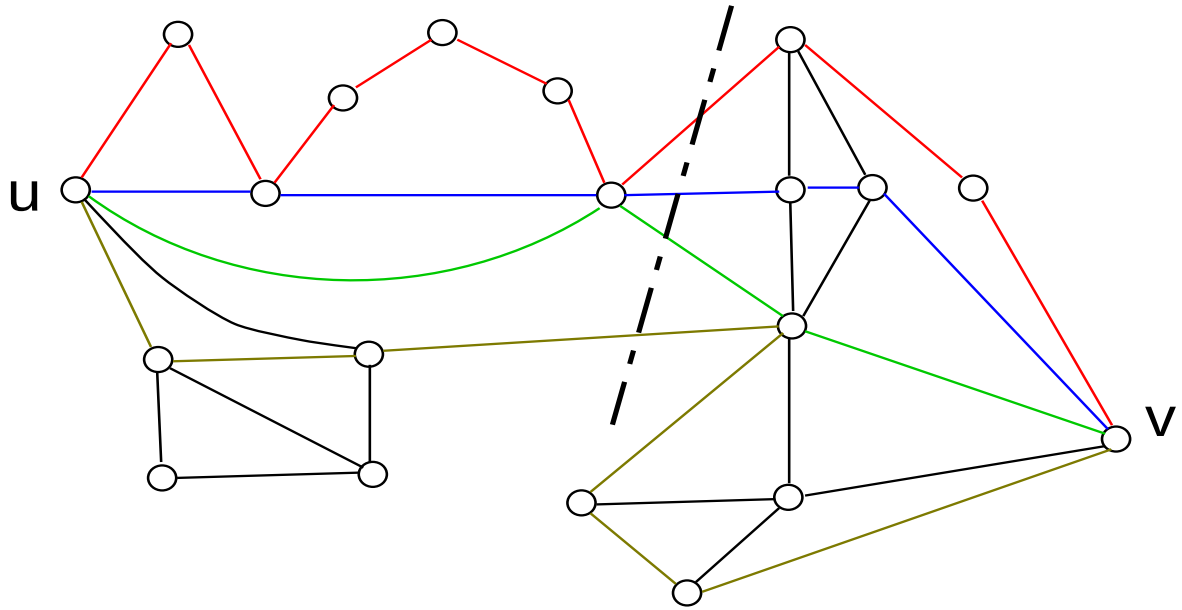
Ein *Schnitt*, der u und v trennt, ist definiert durch

$$S(U, V \setminus U) = \{xy \in E \mid x \in U, y \in V \setminus U\}$$



Schnitte und der Satz von Menger

Satz (Satz von Menger) *Sei $G = (V, E)$ ein ungerichteter Graph und $u, v \in V$. Dann gilt: Die maximale Anzahl kantendisjunkter Wege von u nach v ist gleich der Kantenanzahl eines minimalen Schnitts, der u und v trennt.*



NETZWERKE

Netzwerke

Ein *Netzwerk* $G = (V, E, w)$ ist ein Graph (V, E) mit einer Bewertungsfunktion $w : E \to \mathbb{R}$.

Die gewichtete Adjazenzmatrix eines Netzwerks ist

$$A_w(G) = (w(v_i, v_j))_{1 \leq i,j \leq |V|}.$$

Netzwerke

Ein *Spannbaum* (spannender Baum) eines zusammenhängenden Graphen $G = (V, E)$ ist ein Teilgraph T von G mit folgenden Eigenschaften: $V(T) = V(G)$, $E(T) \subseteq E(G)$.

Ein *Gerüst* eines Graphen ist ein Teilgraph T von G mit folgenden Eigenschaften: $V(T) = V(G)$, $E(T) \subseteq E(G)$, T hat die selben Zusammenhangskomponenten wie G.

Ein Gerüst $T = (V, E')$ eines Netzwerks $G = (V, E, w)$ heißt *minimal*, wenn

$$w(T) := \sum_{e \in E(T)} w(e)$$

minimal ist.

Netzwerke

In einem (gerichteten oder ungerichteten) Netzwerk $G = (V, E, w)$ sei für $x, y \in V$

$$M_{x,y} := \{|W| \,:\, W = (V_W, E_W) \text{ ist ein Weg von } x \text{ nach } y\},$$

wobei

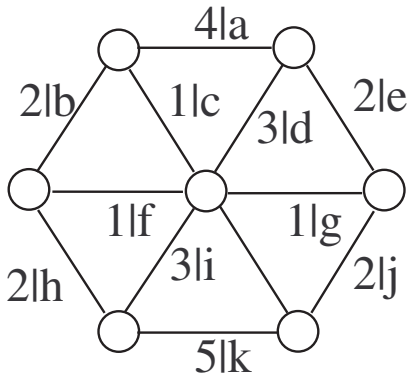
$$|W| = \sum_{e \in E_W} w(e).$$

Dann definieren wir den *Abstand* von x zu y durch

$$d(x,y) = \begin{cases} \min M_{x,y} & \text{falls } M_{x,y} \neq \emptyset, \\ \infty & \text{falls } M_{x,y} = \emptyset. \end{cases}$$

DER ALGORITHMUS VON KRUSKAL

Der Algorithmus von Kruskal

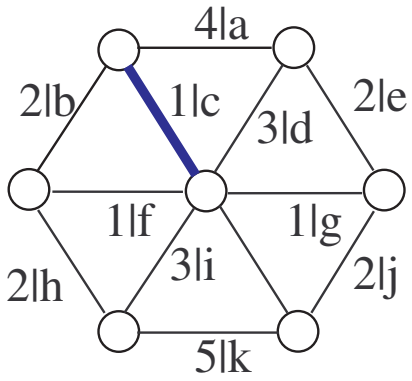


1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



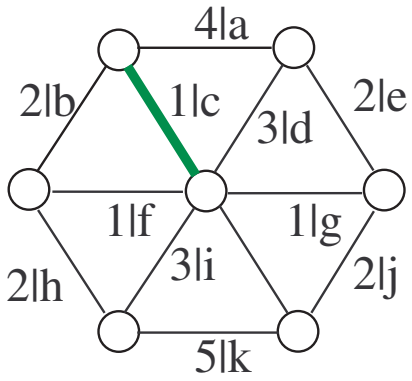
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \emptyset$
$j = 1$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



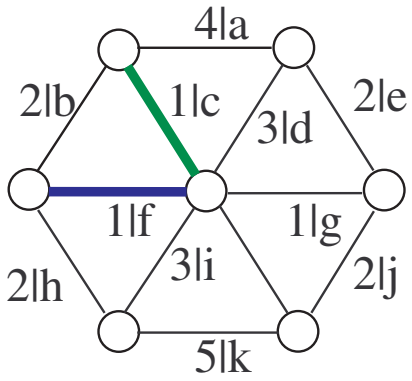
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c\}$
$j = 1$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



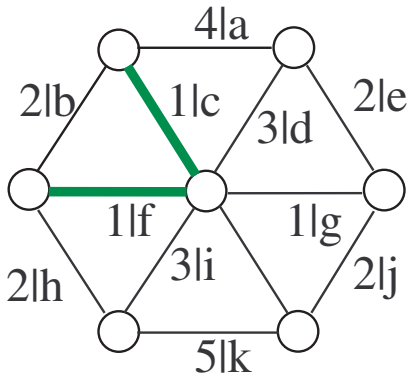
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c\}$
$j = 2$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



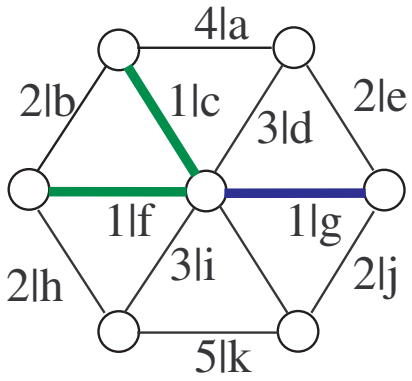
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f\}$
$j = 2$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



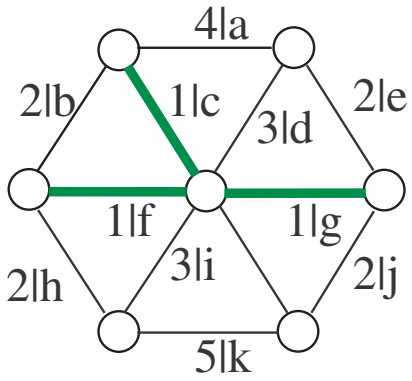
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f\}$
$j = 3$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



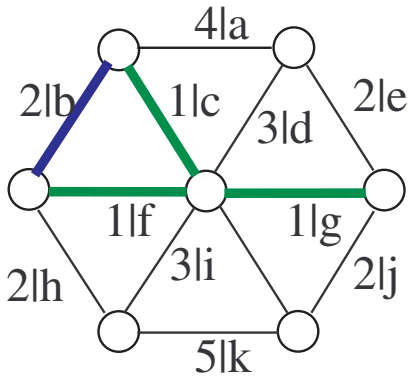
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f, g\}$
$j = 3$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



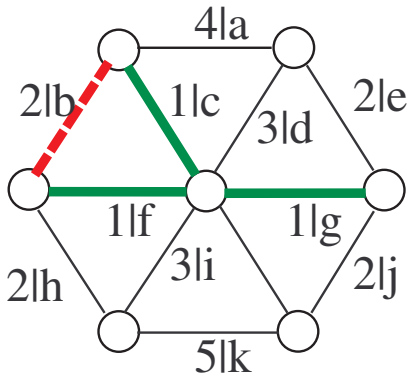
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f, g\}$
$j = 4$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



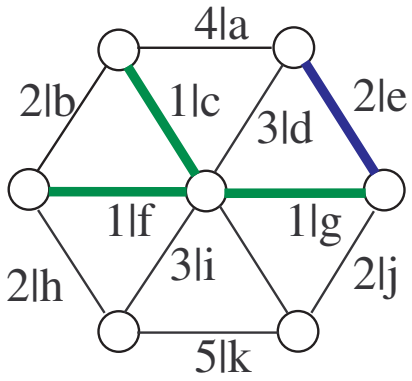
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f, g\}$
$j = 4$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



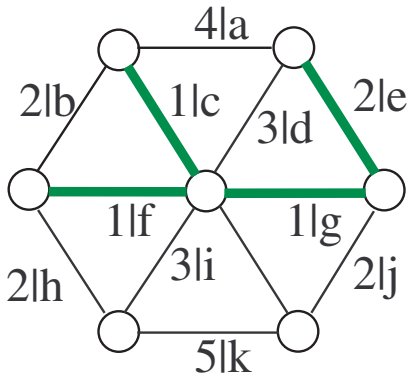
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f, g\}$
$j = 5$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



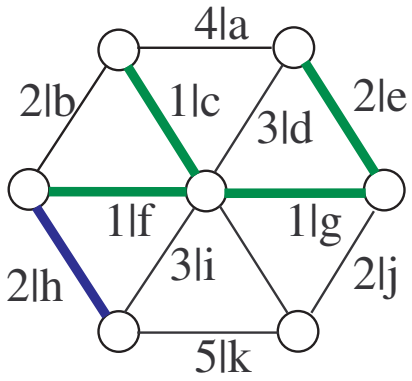
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f, g, e\}$
$j = 5$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



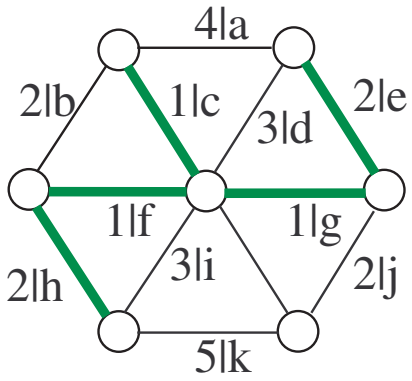
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f, g, e\}$
$j = 6$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



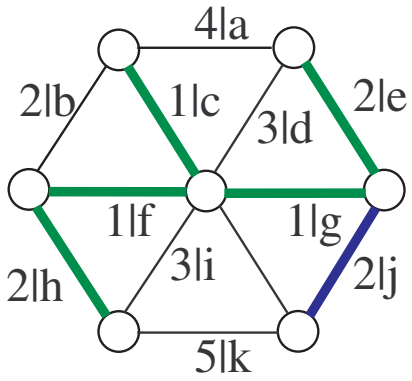
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f, g, e, h\}$
$j = 6$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



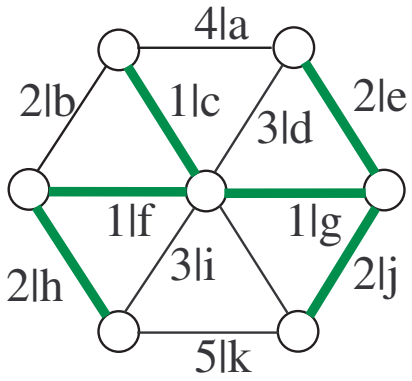
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f, g, e, h\}$
$j = 7$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



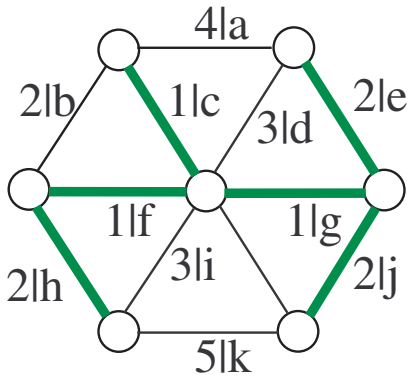
$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f, g, e, h, j\}$
$j = 7$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

Der Algorithmus von Kruskal



$E = \{c, f, g, b, e, h, j, d, i, a, k\}$
$E' = \{c, f, g, e, h, j\}$
$j = 7$
$\boxed{|E'| = 6 \rightsquigarrow ENDE}$

1. Kanten nach steigendem Gewicht sortieren; $E' := \emptyset$; $j := 1$;

2. `if` $(V, E' \cup \{e_j\})$ kreisfrei `then` $E' := E' \cup \{e_j\}$: `end`;

3. `If` $(|E'| = |V| - 1$ `or` $j = m)$ `then END`
   `else` $j := j + 1$; `goto` 2;
   `end`;

DER ALGORITHMUS VON DIJKSTRA

Der Algorithmus von Dijkstra

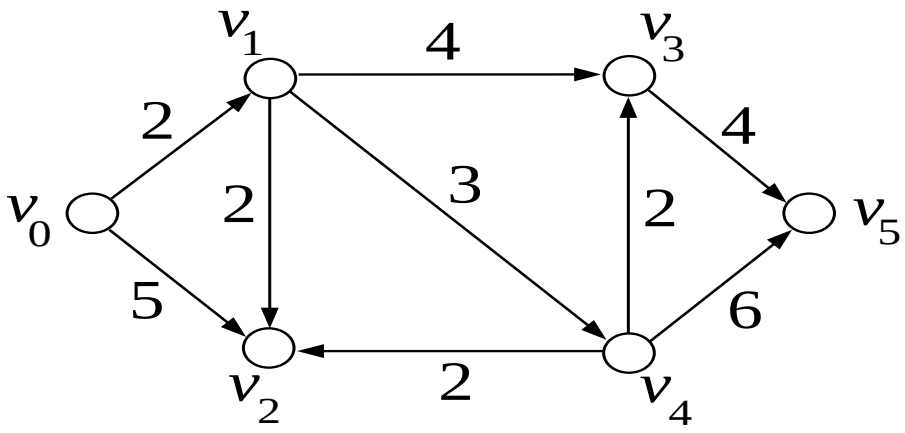
1. $l(v_0) := 0$; `for` $v \in V \setminus \{v_0\}$ `do` $l(v) := \infty$ `end`; $U := \{v_0\}$; $u := v_0$;

2. `for` $v \in V \setminus U$ `do`:
   `if` $(u,v) \in E$ `and` $l(v) > l(u) + w(u,v)$ `then`
   $p(v) := u$;
   $l(v) := l(u) + w(u,v)$;
   `end if`;

3. $m := \min_{v \in V \setminus U} l(v)$, wähle Knoten $z \in V \setminus U$ mit $l(z) = m$;
   $U := U \cup \{z\}$;
   $u := z$;

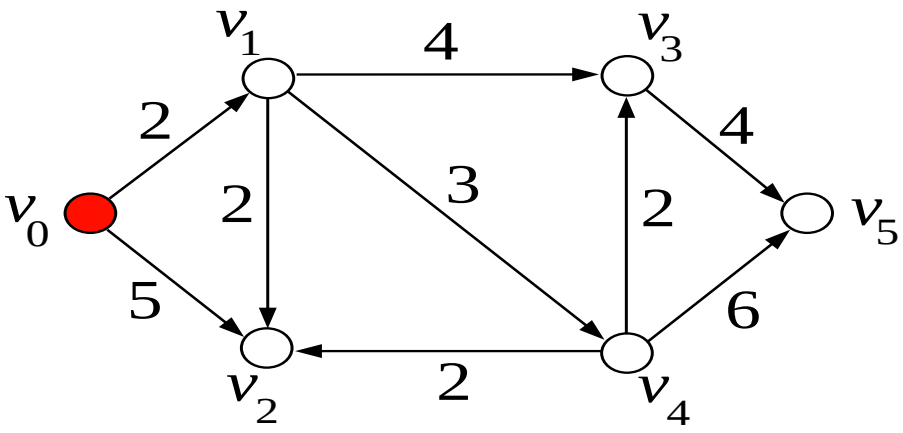
4. `if` $U = V$ `or` $\forall v \in V \setminus U : l(v) = \infty$ `then END`
   `else goto 2`;

Der Algorithmus von Dijkstra



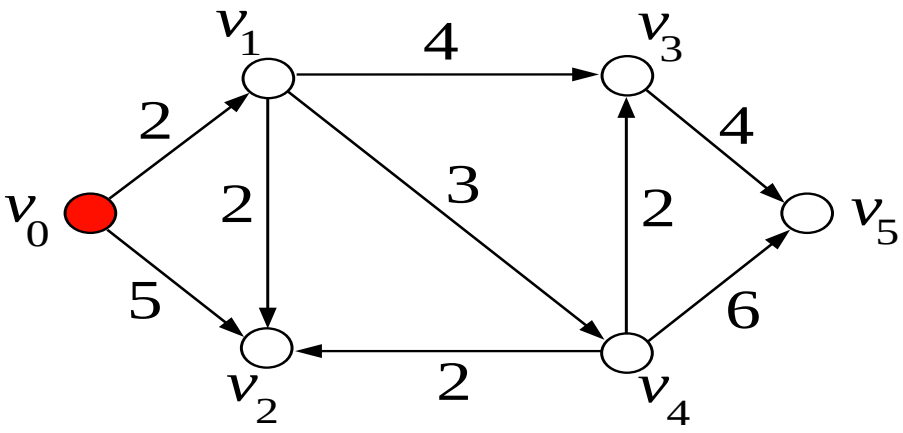
	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	0	∞	∞	∞	∞	∞		

Der Algorithmus von Dijkstra



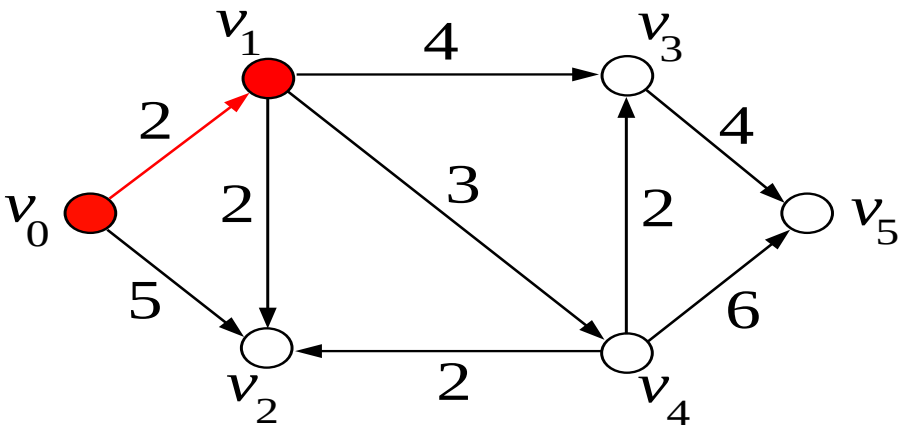
	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	

Der Algorithmus von Dijkstra



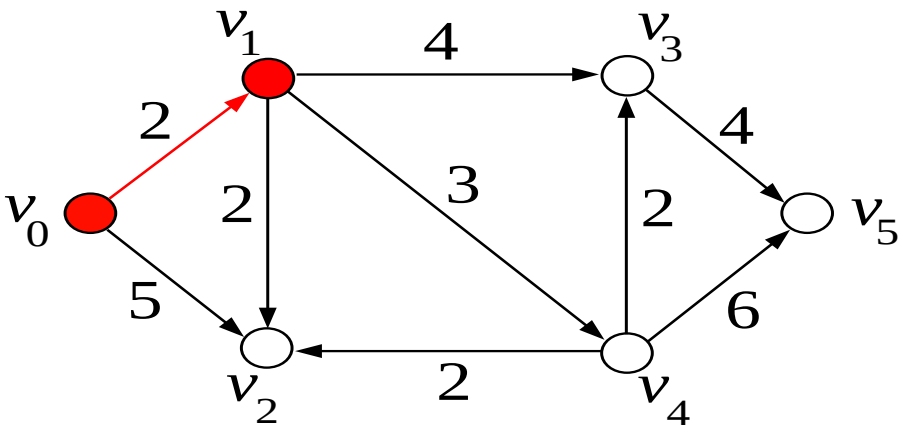
	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	
1		$2/v_0$	$5/v_0$	∞	∞	∞		

Der Algorithmus von Dijkstra



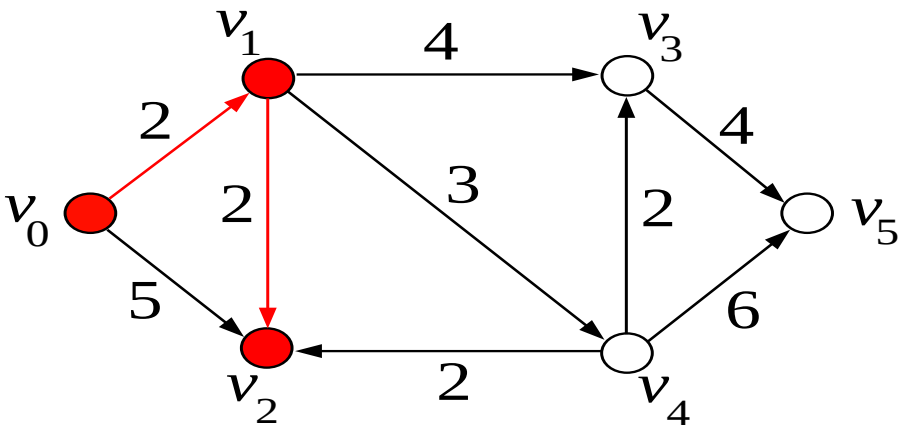
	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	
1		$\boxed{2/v_0}$	$5/v_0$	∞	∞	∞	v_1	v_0

Der Algorithmus von Dijkstra



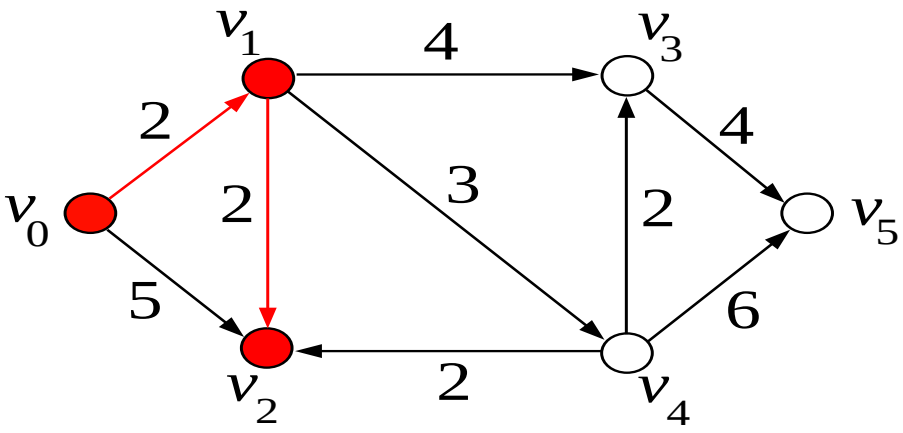
	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	
1		$\boxed{2/v_0}$	$5/v_0$	∞	∞	∞	v_1	v_0
2			$4/v_1$	$6/v_1$	$5/v_1$	∞		

Der Algorithmus von Dijkstra



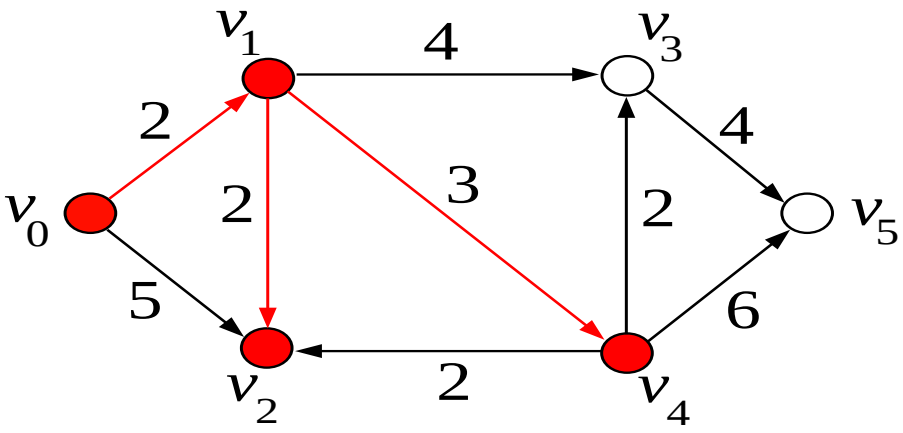
	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	
1		$\boxed{2/v_0}$	$5/v_0$	∞	∞	∞	v_1	v_0
2			$\boxed{4/v_1}$	$6/v_1$	$5/v_1$	∞	v_2	v_1

Der Algorithmus von Dijkstra



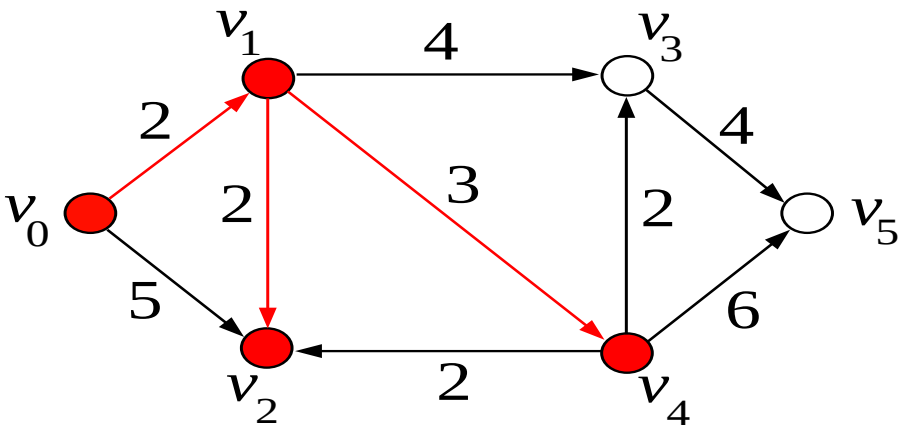
	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	
1		$\boxed{2/v_0}$	$5/v_0$	∞	∞	∞	v_1	v_0
2			$\boxed{4/v_1}$	$6/v_1$	$5/v_1$	∞	v_2	v_1
3				$6/v_1$	$5/v_1$	∞		

Der Algorithmus von Dijkstra



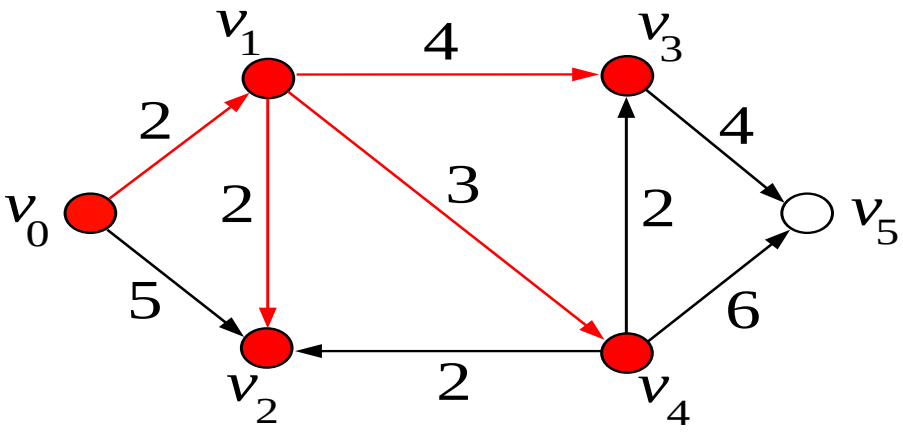
	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	
1		$\boxed{2/v_0}$	$5/v_0$	∞	∞	∞	v_1	v_0
2			$\boxed{4/v_1}$	$6/v_1$	$5/v_1$	∞	v_2	v_1
3				$6/v_1$	$\boxed{5/v_1}$	∞	v_4	v_1

Der Algorithmus von Dijkstra



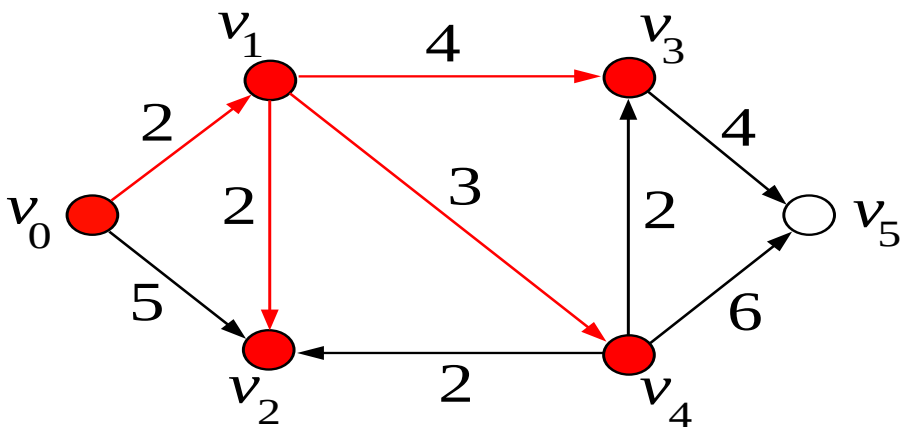
	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	
1		$\boxed{2/v_0}$	$5/v_0$	∞	∞	∞	v_1	v_0
2			$\boxed{4/v_1}$	$6/v_1$	$5/v_1$	∞	v_2	v_1
3				$6/v_1$	$\boxed{5/v_1}$	∞	v_4	v_1
4				$6/v_1$		$11/v_4$		

Der Algorithmus von Dijkstra



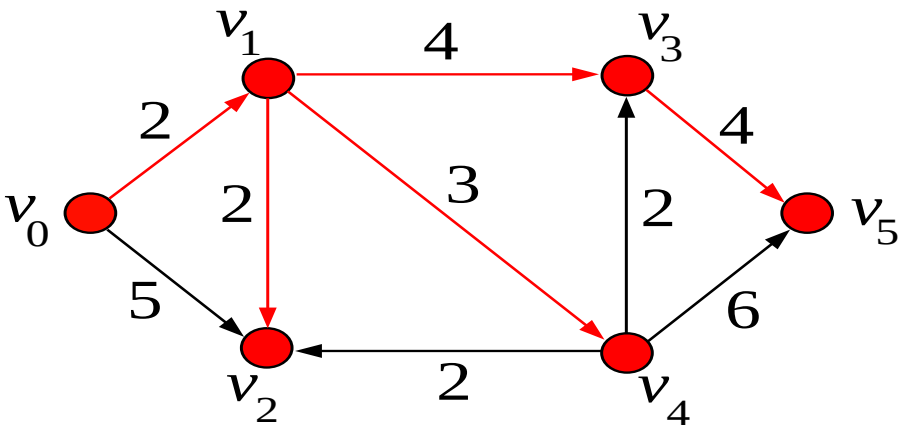
	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	
1		$\boxed{2/v_0}$	$5/v_0$	∞	∞	∞	v_1	v_0
2			$\boxed{4/v_1}$	$6/v_1$	$5/v_1$	∞	v_2	v_1
3				$6/v_1$	$\boxed{5/v_1}$	∞	v_4	v_1
4				$\boxed{6/v_1}$		$11/v_4$	v_3	v_1

Der Algorithmus von Dijkstra



	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	
1		$\boxed{2/v_0}$	$5/v_0$	∞	∞	∞	v_1	v_0
2			$\boxed{4/v_1}$	$6/v_1$	$5/v_1$	∞	v_2	v_1
3				$6/v_1$	$\boxed{5/v_1}$	∞	v_4	v_1
4				$\boxed{6/v_1}$		$11/v_4$	v_3	v_1
5						$10/v_3$		

Der Algorithmus von Dijkstra



	v_0	v_1	v_2	v_3	v_4	v_5	Auswahl	Vorgänger
0	$\boxed{0}$	∞	∞	∞	∞	∞	v_0	
1		$\boxed{2/v_0}$	$5/v_0$	∞	∞	∞	v_1	v_0
2			$\boxed{4/v_1}$	$6/v_1$	$5/v_1$	∞	v_2	v_1
3				$6/v_1$	$\boxed{5/v_1}$	∞	v_4	v_1
4				$\boxed{6/v_1}$		$11/v_4$	v_3	v_1
5						$\boxed{10/v_3}$	v_5	v_3