Graphen und diskrete Optimierung

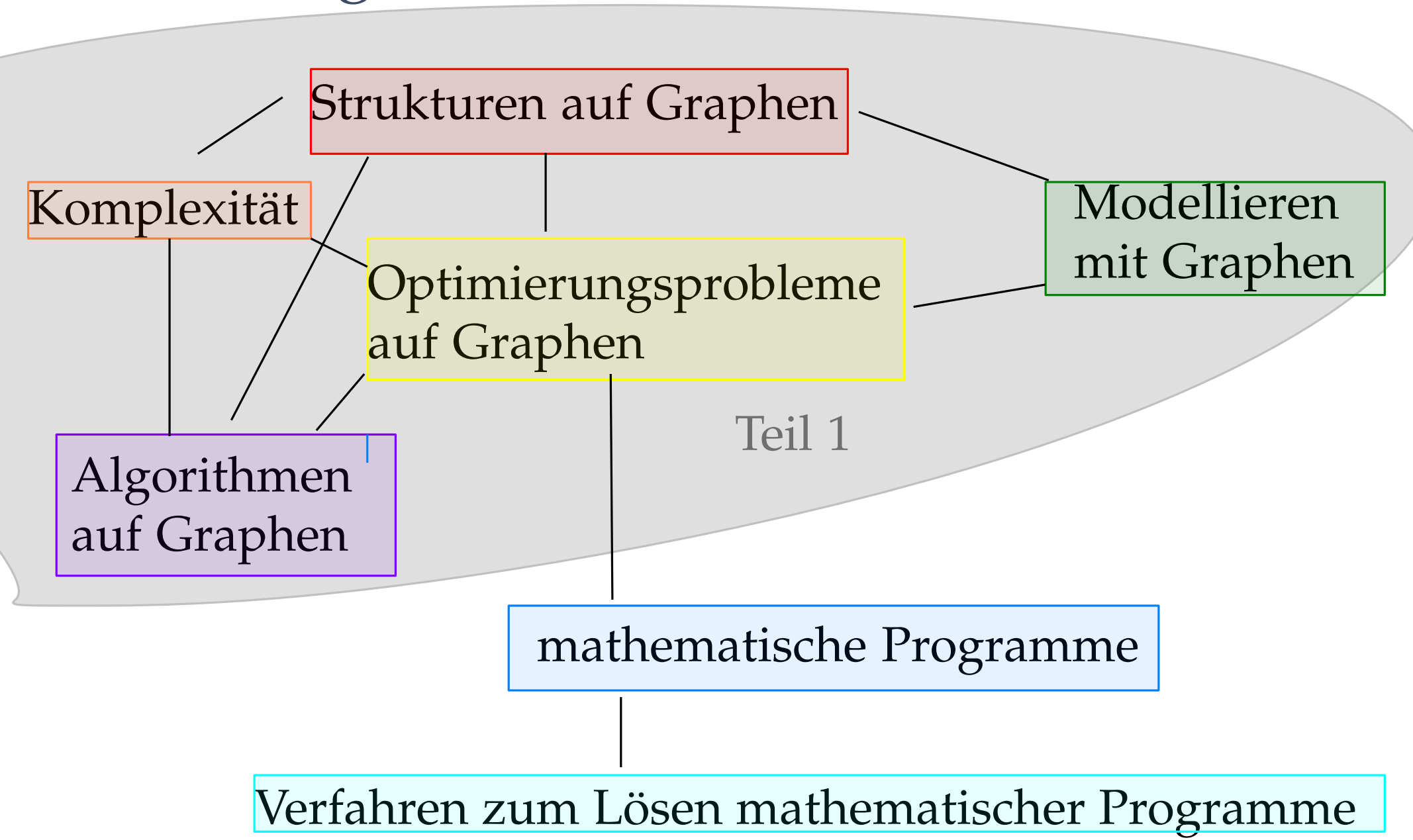
im Bachelorstudiengang 'Informatik und Nachhaltigkeit'

Wege und Touren

Marie Schmidt

26.04.2023

Vorlesungsübersicht

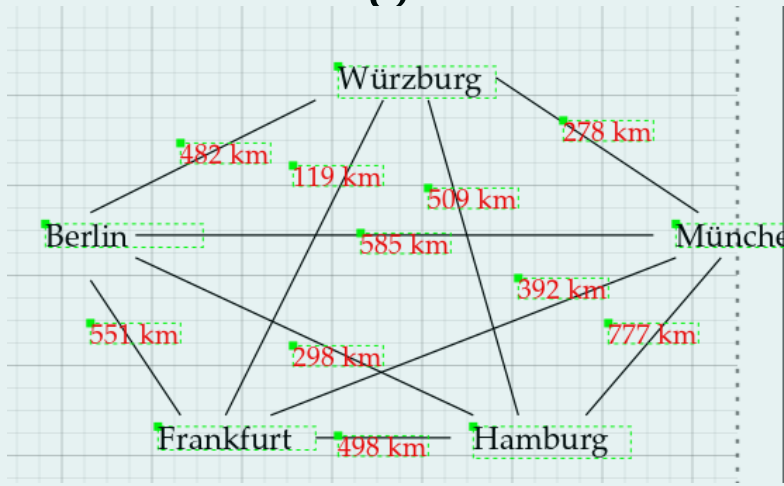


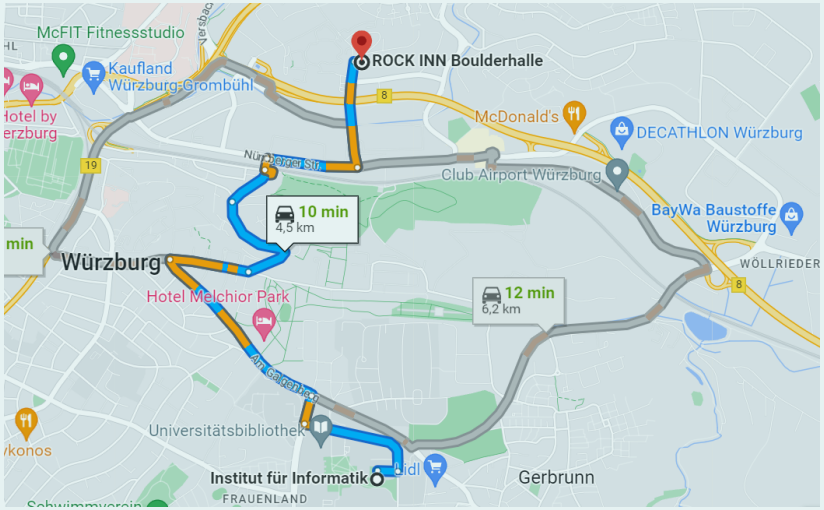
Vorlesungsübersicht

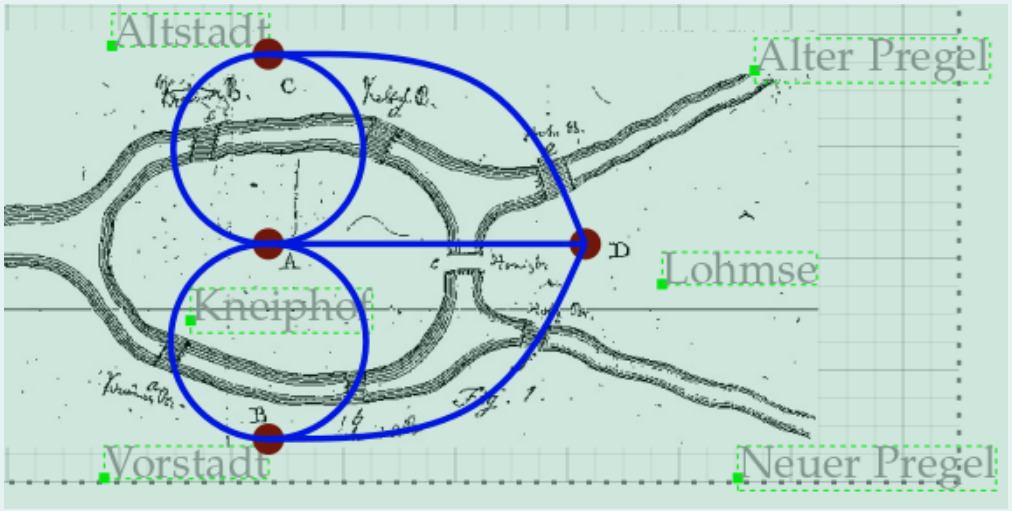
- Einführung: Modellieren auf Graphen ✓
- Teil 1: Graphen und Algorithmen auf Graphen
  - Graphen modellieren räumliche Zusammenhänge
    - Wege: Dijkstra ✓, Bellmann-Ford ✓, multikriterielle Wege
    - Touren
    - Flüsse
  - Graphen modellieren Beziehungen
    - Matchings
    - Färbungsprobleme
    - ...
  - Algorithmen und (Problem-)Komplexität
- Teil 2: Lineare und ganzzahlige Optimierung auf Graphen
  - ...

Worum geht es heute: Wege und Touren

heute: Knoten stehen für Orte, Kanten für Verbindungen zwischen Orten, Kantenlabels für Entfernungen







Agenda

1. Prüfungstermine
2. Wege
3. Touren

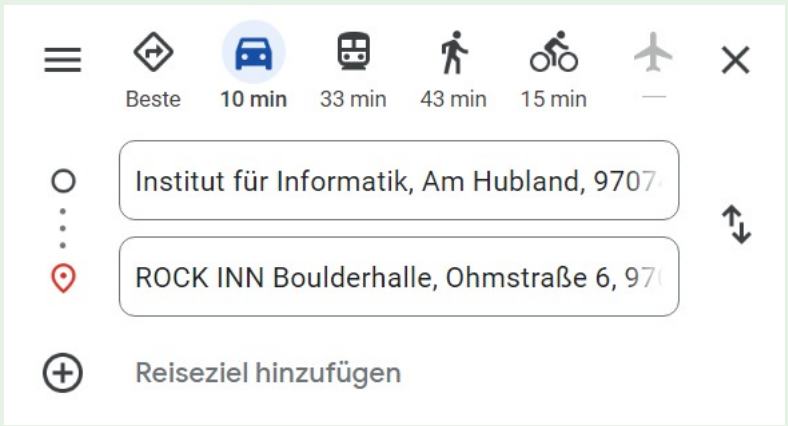
Prüfungstermine

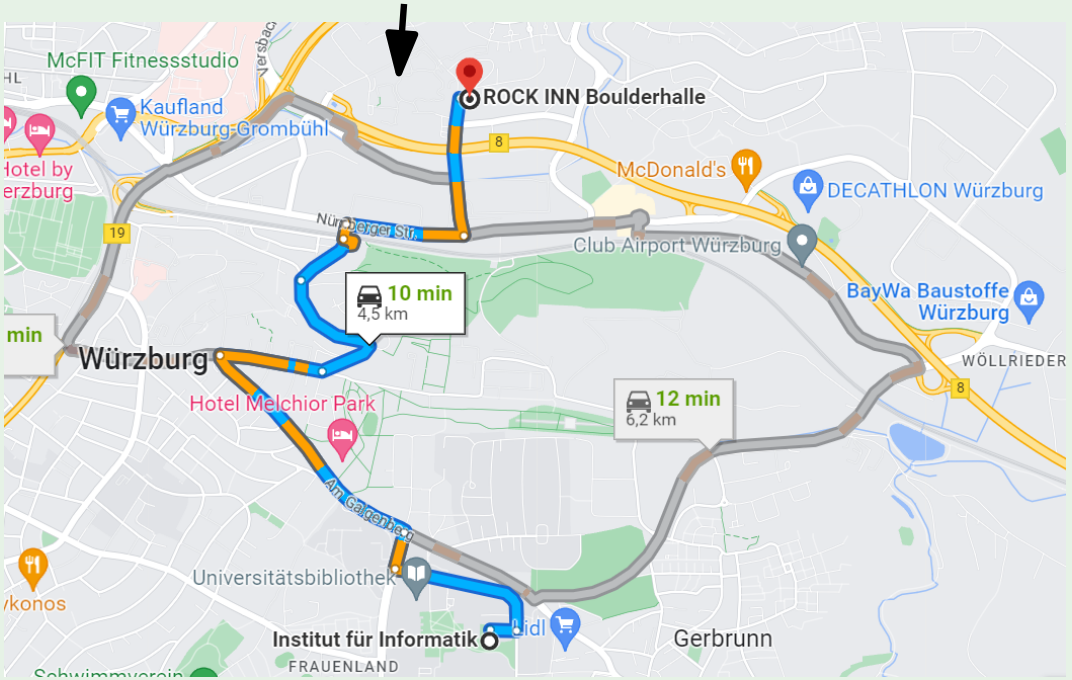
$\rightarrow$ mündliche Prüfungen: Ende erste Semesterferienwoche (26.-28.7.), etwa 20 Minuten

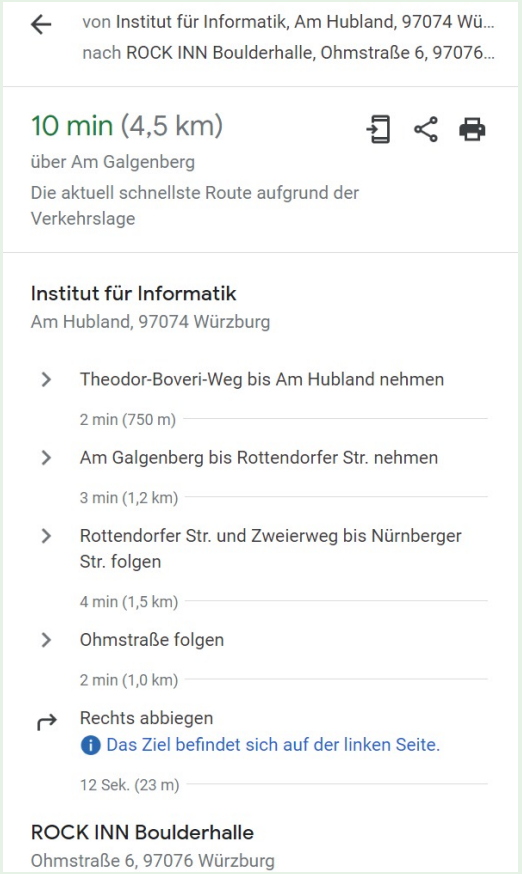
Agenda

1. Prüfungstermine ✓
2. **Wege**
3. Touren

Beispiel Routenplanung revisited

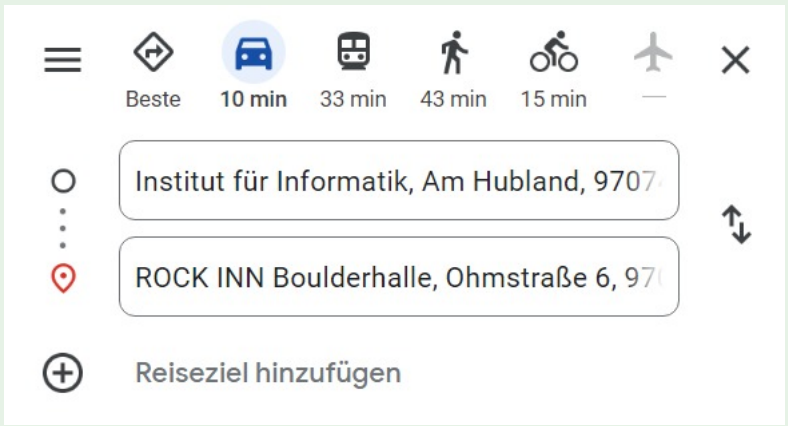


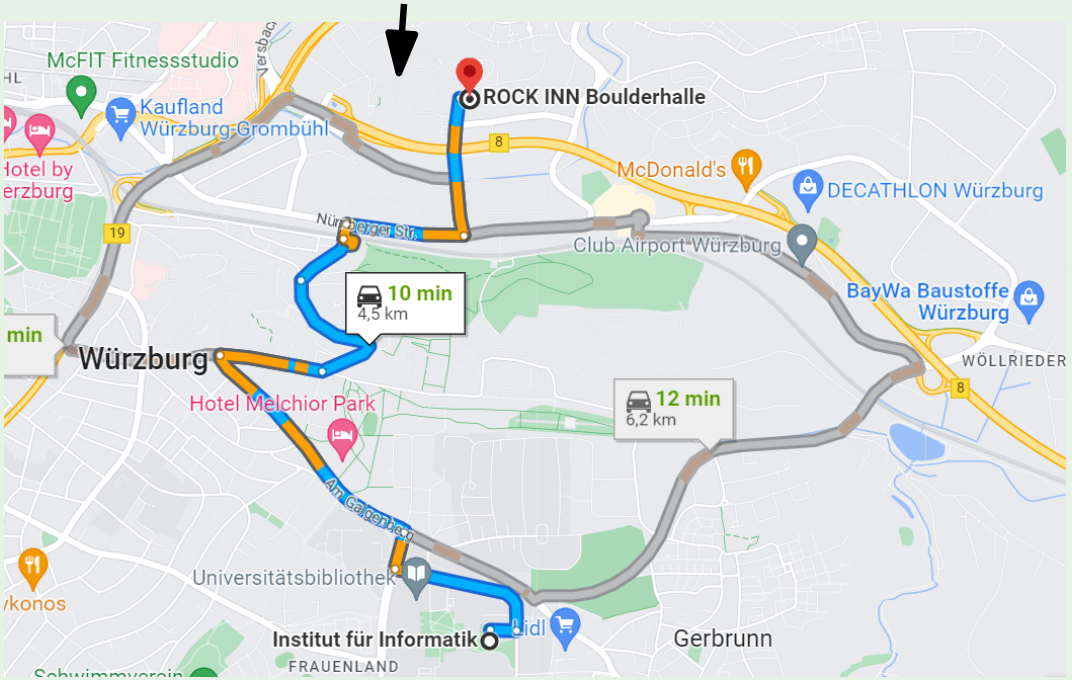


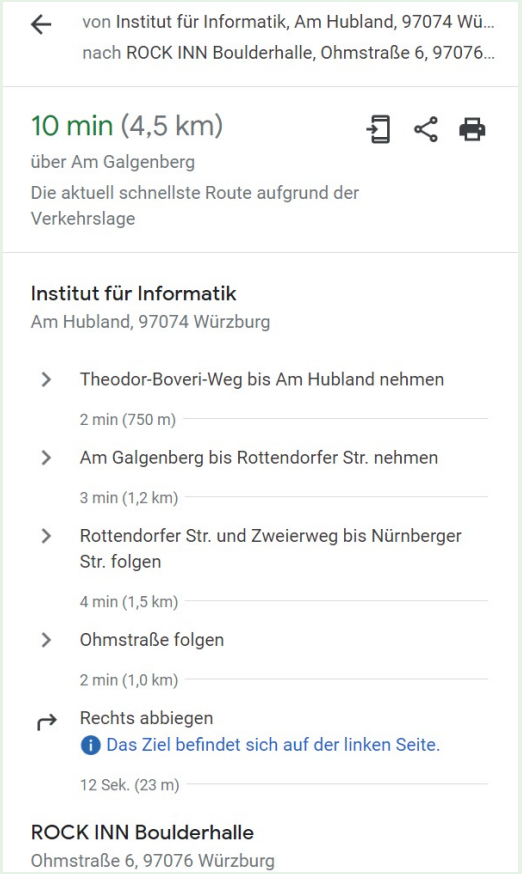


Übung: kürzeste Wege / Dijkstra / Bellmann-Ford

Beispiel Routenplanung revisited



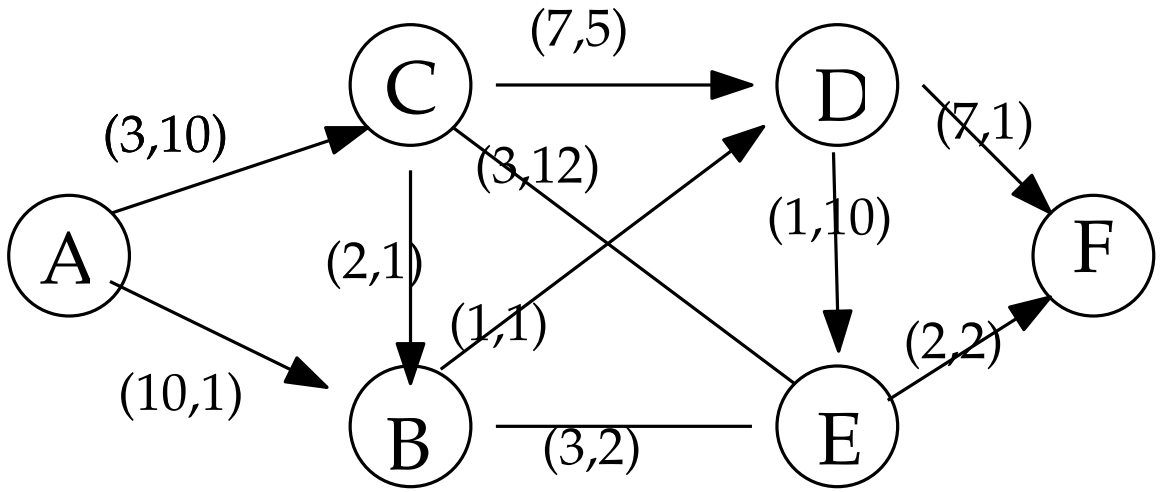




Übung: kürzeste Wege / Dijkstra / Bellmann-Ford

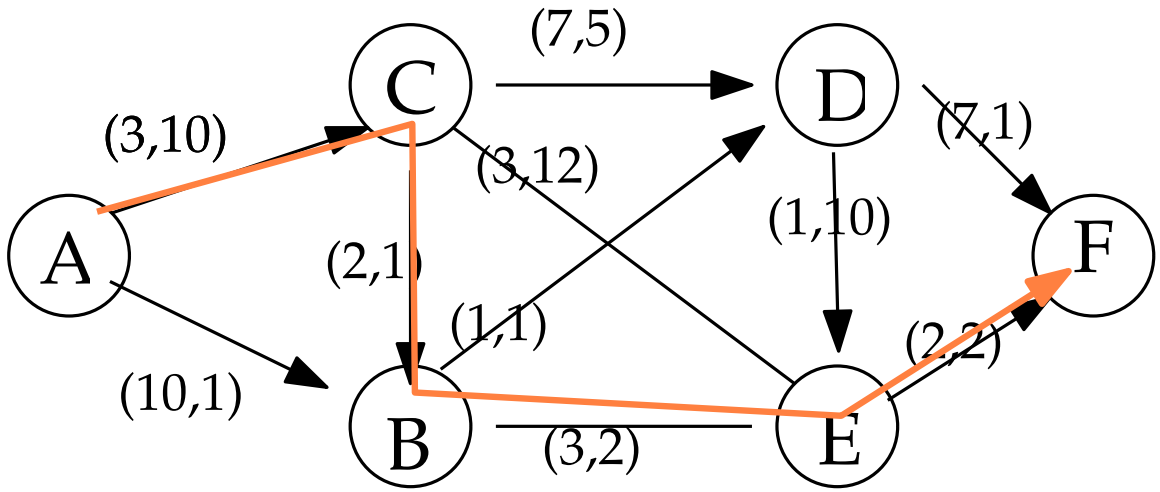
Was, wenn wir mehr als eine Zielfunktion haben?

Multikriterielle kürzeste Wege



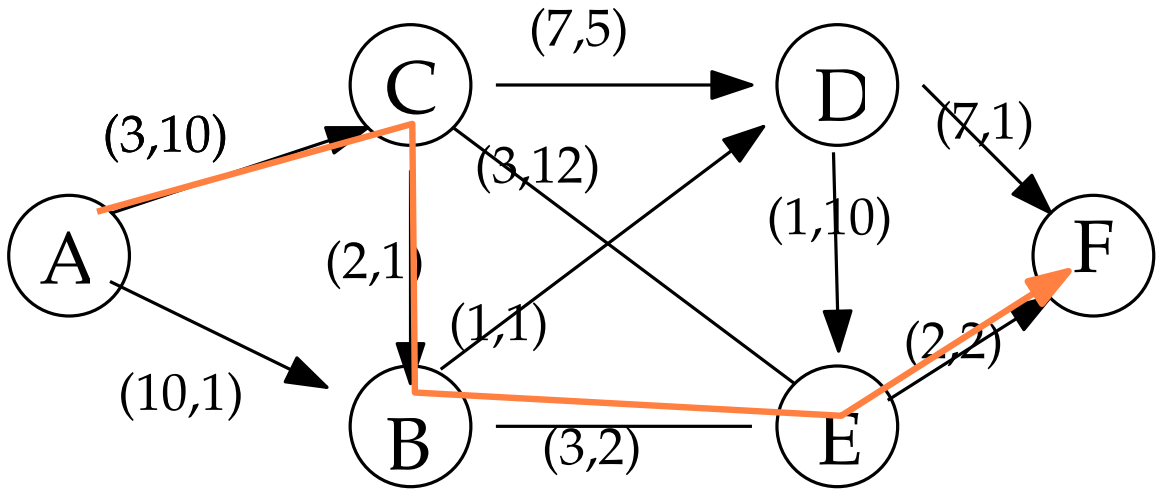
Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: ??

Multikriterielle kürzeste Wege



Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: ??

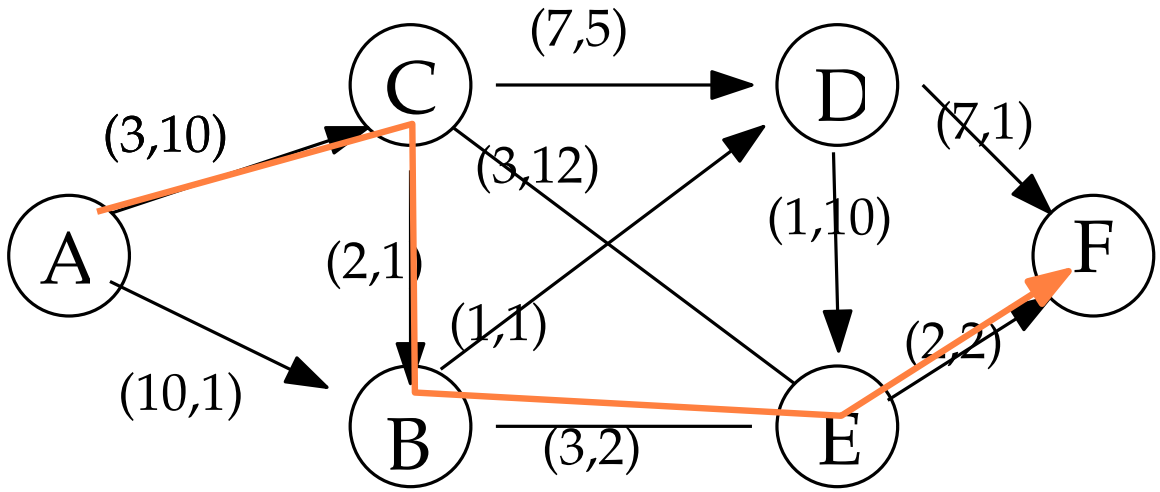
Multikriterielle kürzeste Wege



Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: ??

Weg	Distanz	Kosten
orange		

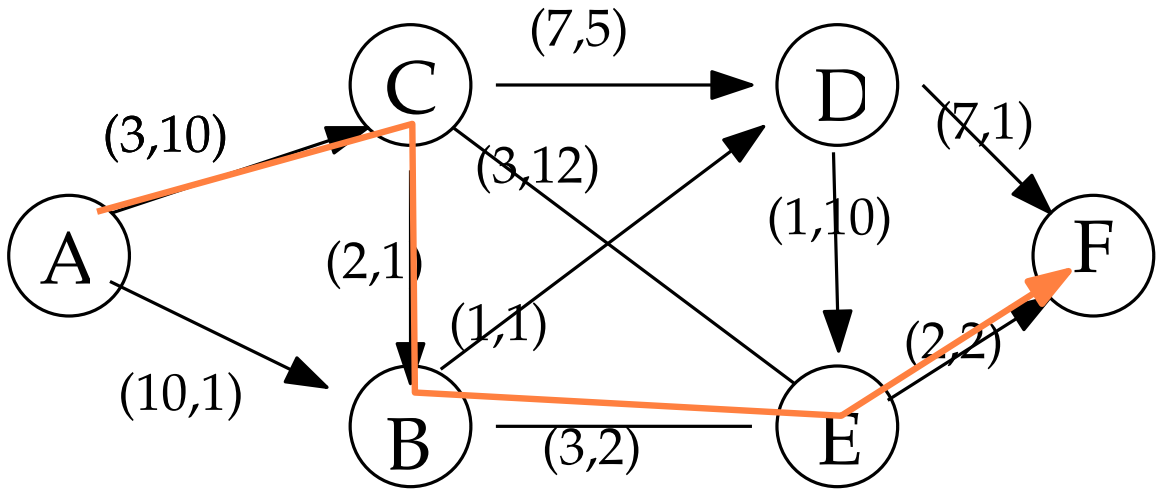
Multikriterielle kürzeste Wege



Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: ??

Weg	Distanz	Kosten
orange	10	

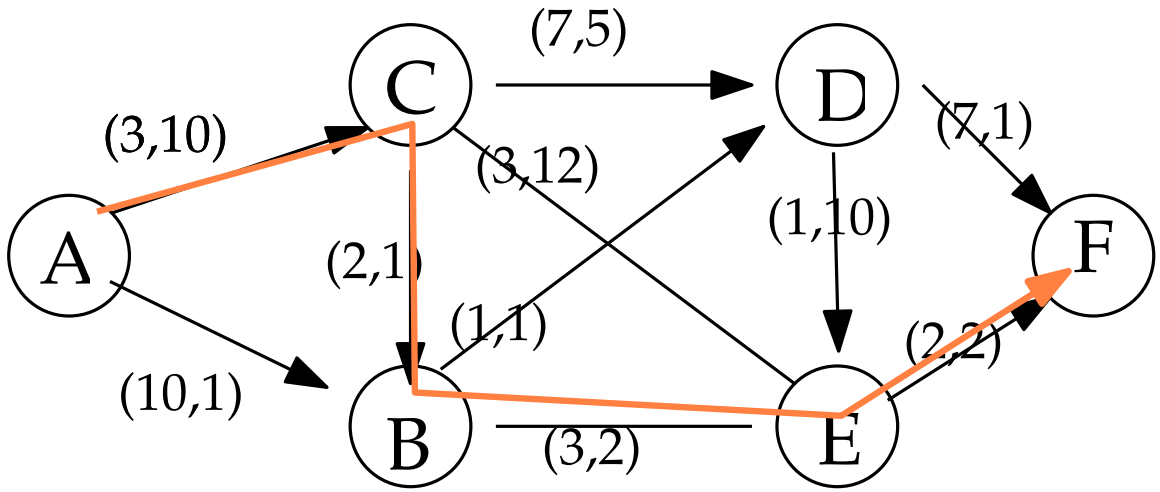
Multikriterielle kürzeste Wege



Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: ??

Weg	Distanz	Kosten
orange	10	

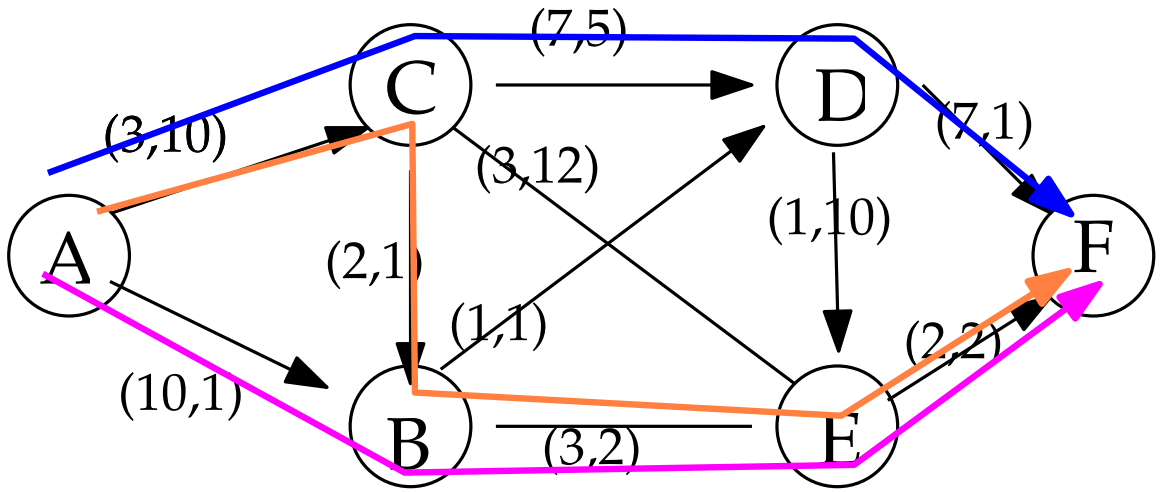
Multikriterielle kürzeste Wege



Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: ??

Weg	Distanz	Kosten
orange	10	15

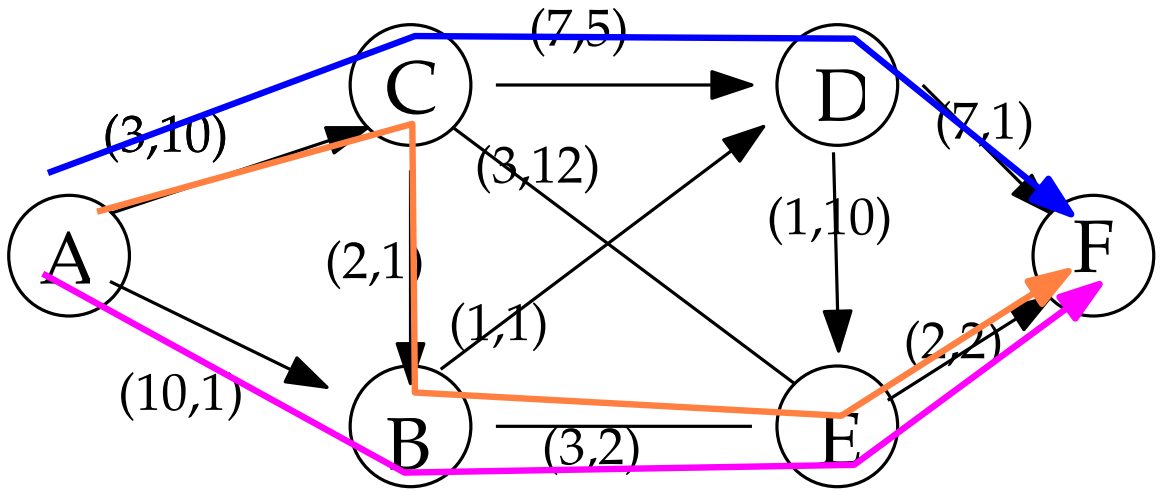
Multikriterielle kürzeste Wege



Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: ??

Weg	Distanz	Kosten
orange	10	15
blau		
pink		

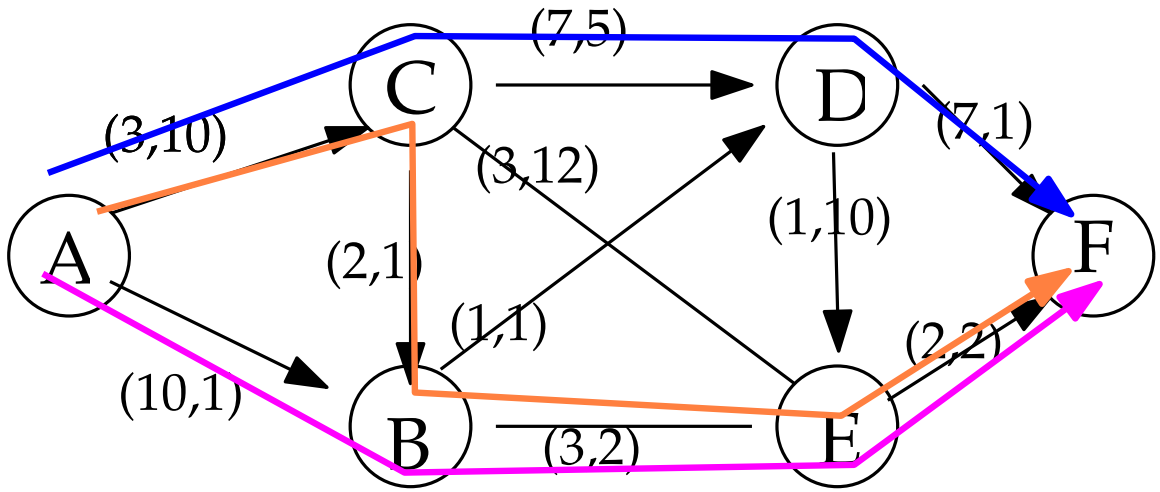
Multikriterielle kürzeste Wege



Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: ??

Weg	Distanz	Kosten
orange	10	15
blau	17	16
pink	15	5

Multikriterielle kürzeste Wege

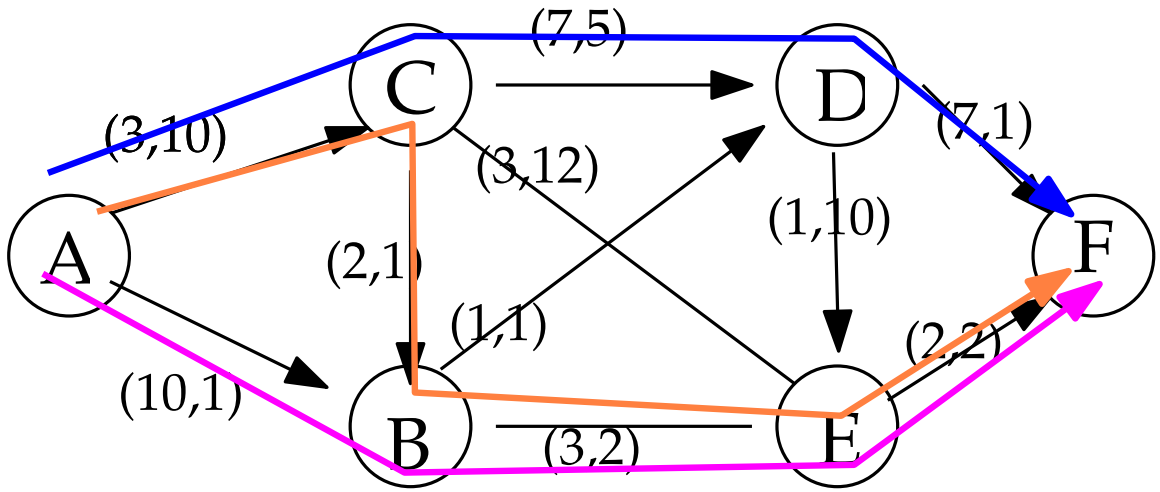


Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: ??

Weg	Distanz	Kosten
orange	10	15
blau	17	16
pink	15	5

Wir sagen: Weg W **dominiert** Weg W' ($W \leqq W'$) , wenn W in keinem Kriterium schlechter ist als W' und in mindestens einem besser.

Multikriterielle kürzeste Wege

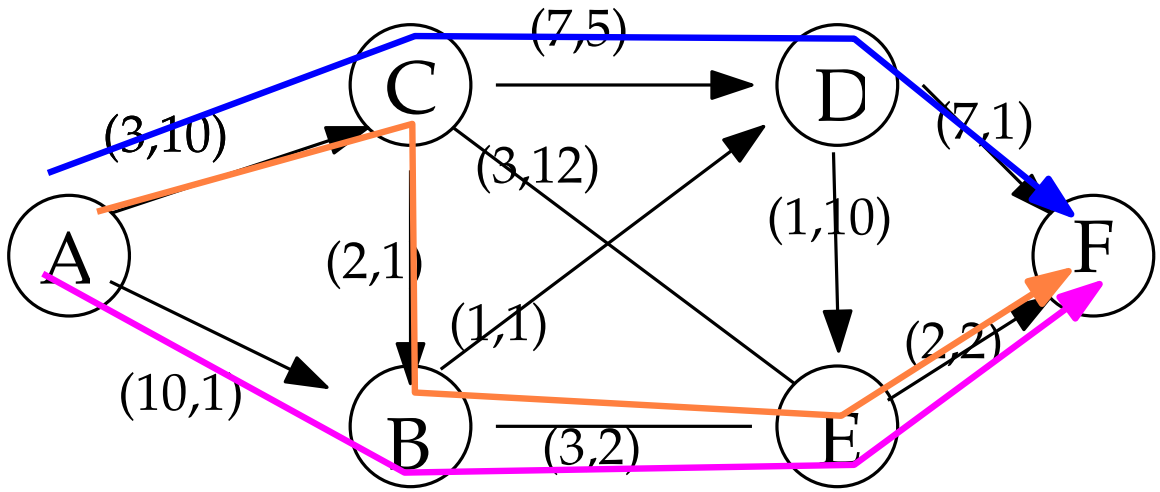


Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: ??

Weg	Distanz	Kosten
orange	10	15
blau	17	16
pink	15	5

Wir sagen: Weg W **dominiert** Weg W' ($W \leqq W'$) , wenn W in keinem Kriterium schlechter ist als W' und in mindestens einem besser.

Multikriterielle kürzeste Wege

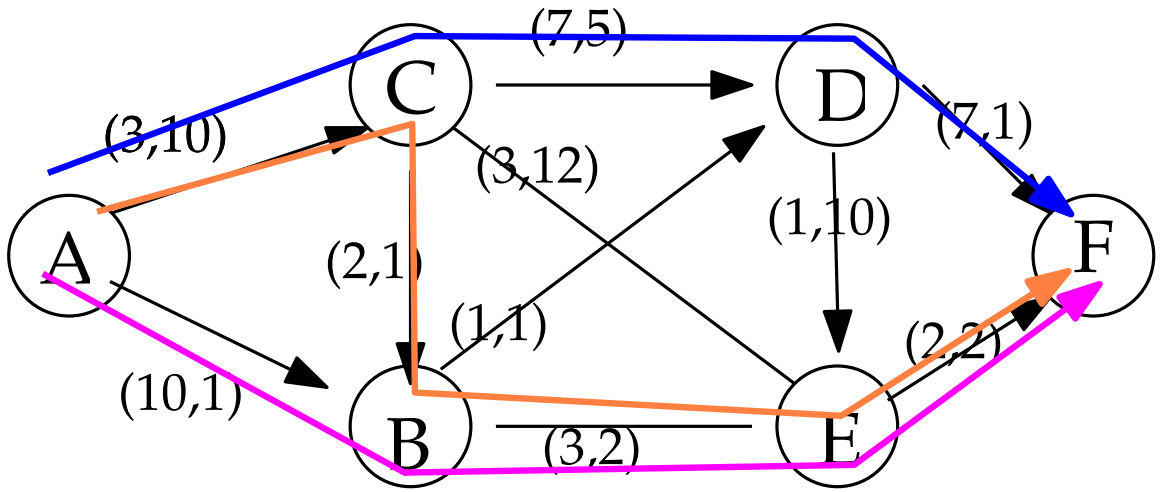


Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: Menge alles **nichtdominierten** s-t-Wege

Weg	Distanz	Kosten
orange	10	15
blau	17	16
pink	15	5

Wir sagen: Weg W **dominiert** Weg W' ($W \leqq W'$) , wenn W in keinem Kriterium schlechter ist als W' und in mindestens einem besser.

Multikriterielle kürzeste Wege



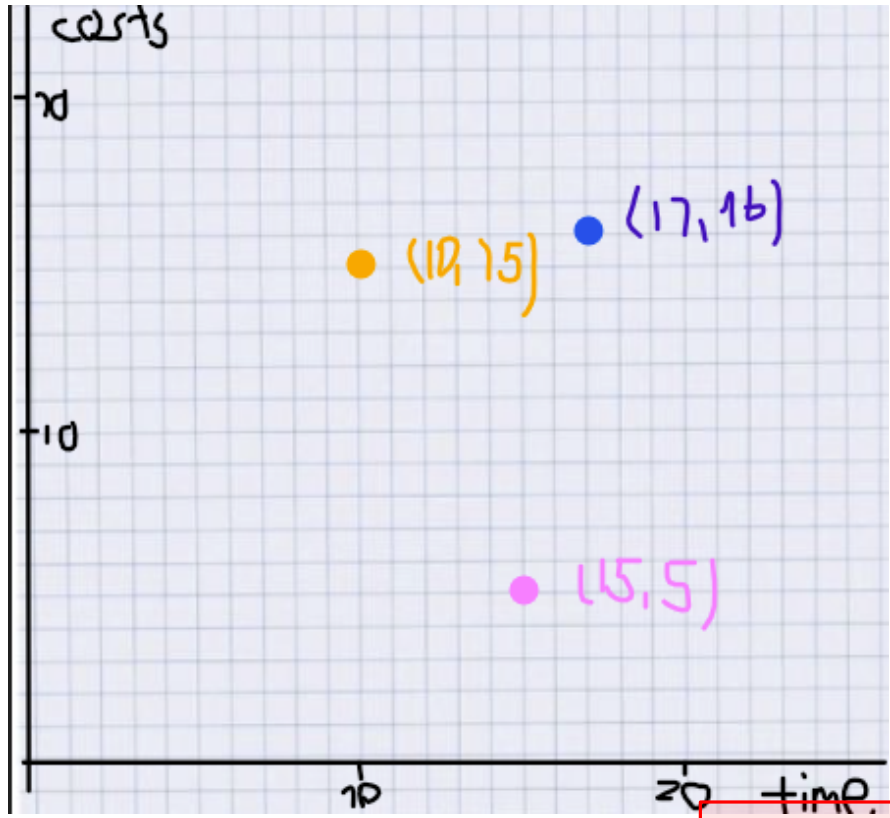
nichtdominierte Lösungen aka **Pareto-Lösungen**: zulässige Lösungen eines multikriteriellen Optimierungsproblems, die durch keine andere zulässige Lösung dominiert werden

Multikriterielles kürzeste-Wege Problem
Gegeben: ein (gerichteter oder ungerichteter) Graph mit m Kantenlabels auf jeder Kante, Startknoten s und Zielknoten t
Gesucht: Menge alles **nichtdominierten** s-t-Wege

Weg	Distanz	Kosten
orange	10	15
blau	17	16
pink	15	5

Wir sagen: Weg W **dominiert** Weg W' ($W \leqq W'$) , wenn W in keinem Kriterium schlechter ist als W' und in mindestens einem besser.

Multikriterielle kürzeste Wege

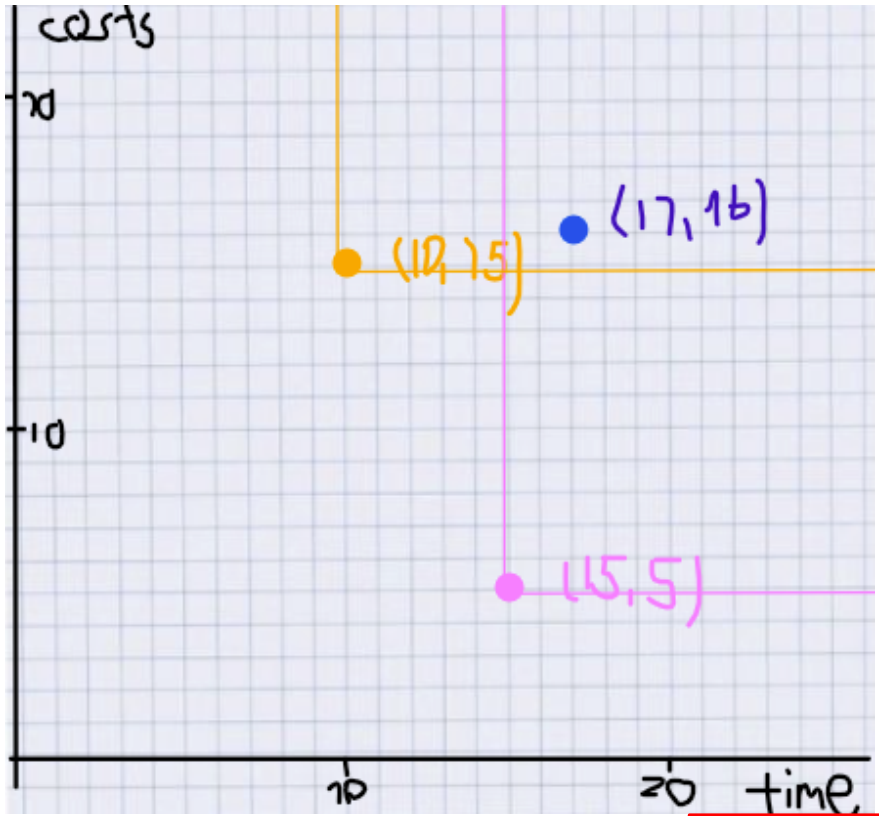


Dominanz dargestellt im Zielfunktionsraum

Weg	Distanz	Kosten
orange	10	15
blau	17	16
pink	15	5

Weg W **dominiert** Weg W', wenn W in keinem Kriterium schlechter ist als W' und in mindestens einem besser.

Multikriterielle kürzeste Wege



Dominanz dargestellt im Zielfunktionsraum

Weg	Distanz	Kosten
orange	10	15
blau	17	16
pink	15	5

Weg W **dominiert** Weg W', wenn W in keinem Kriterium schlechter ist als W' und in mindestens einem besser.

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Input: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit m positiven Kantenlabels w_e^i für $i = 1, \ldots, m$, $e \in E$, Startknoten s, Zielknoten t

Output: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)

Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ for $i = 1..m$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$

while L_{open} nicht leer **do**

  wähle $l \in L_{open}$,

  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$

  for $v \in Adj(l.knoten)$ **do**

    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für alle $i = 1..m$, $l'.\pi = l$

    Dominanzcheck$(l', L_{open}, L_{closed})$

  end for

end while

return Labels $l \in L_{closed}$ mit $l.knoten = t$

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Input: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit m positiven Kantenlabels w_e^i für $i = 1, \ldots, m$, $e \in E$, Startknoten s, Zielknoten t

Output: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)

Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ for $i = 1..m$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$

while L_{open} nicht leer **do**

  wähle $l \in L_{open}$,

  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$

  for $v \in Adj(l.knoten)$ **do**

    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für alle $i = 1..m$, $l'.\pi = l$

    Dominanzcheck$(l', L_{open}, L_{closed})$

  end for

end while

return Labels $l \in L_{closed}$ mit $l.knoten = t$

Label l repräsentiert einen Weg $p(l)$ von s zu einem Knoten u.
l setzt sich zusammen aus:
- $l.knoten = u$ - Endknoten des Weges
- d_i für $i = 1..m$: 'Länge' des Weges bzgl Kantenlabels w_i
- (Pointer auf) Vorgängerlabel $l.\pi$

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Input: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit m positiven Kantenlabels w_e^i für $i = 1, \ldots, m$, $e \in E$, Startknoten s, Zielknoten t

Output: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)

Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ for $i = 1..m$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$

while L_{open} nicht leer **do**

  wähle $l \in L_{open}$,

  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$

  for $v \in Adj(l.knoten)$ **do**

    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für alle $i = 1..m$, $l'.\pi = l$

    Dominanzcheck$(l', L_{open}, L_{closed})$

  end for

end while

return Labels $l \in L_{closed}$ mit $l.knoten = t$

Clevere Wahl von l?

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Input: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit m positiven Kantenlabels w_e^i für $i = 1, \ldots, m$, $e \in E$, Startknoten s, Zielknoten t

Output: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)

Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ for $i = 1..m$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$

while L_{open} nicht leer **do**

  wähle $l \in L_{open}$,

  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$

  for $v \in Adj(l.knoten)$ **do**

    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für alle $i = 1..m$, $l'.\pi = l$

    Dominanzcheck$(l', L_{open}, L_{closed})$

  end for

end while

return Labels $l \in L_{closed}$ mit $l.knoten = t$

Clevere Wahl von l?

Wie machen wir **Dominanzcheck**?

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Input: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit m positiven Kantenlabels w_e^i für $i = 1, \ldots, m$, $e \in E$, Startknoten s, Zielknoten t

Output: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)

Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ for $i = 1..m$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$

while L_{open} nicht leer **do**

  wähle $l \in L_{open}$,

  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$

  for $v \in Adj(l.knoten)$ **do**

    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für alle $i = 1..m$, $l'.\pi = l$

    Dominanzcheck$(l', L_{open}, L_{closed})$

  end for

end while

return Labels $l \in L_{closed}$ mit $l.knoten = t$

Clevere Wahl von l?

Wie machen wir **Dominanzcheck**?

Erinnert Sie der Algorithmus an etwas?

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Input: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit m positiven Kantenlabels w_e^i für $i = 1, \ldots, m$, $e \in E$, Startknoten s, Zielknoten t

Output: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)

Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ for $i = 1..m$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$

while L_{open} nicht leer **do**

  wähle $l \in L_{open}$,

  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$

  for $v \in Adj(l.knoten)$ **do**

    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für alle $i = 1..m$, $l'.\pi = l$

    Dominanzcheck$(l', L_{open}, L_{closed})$

  end for

end while

return Labels $l \in L_{closed}$ mit $l.knoten = t$

Clevere Wahl von l?
Wie machen wir **Dominanzcheck**?
Erinnert Sie der Algorithmus an etwas?
Laufzeit?

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Input: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit m positiven Kantenlabels w_e^i für $i = 1, \ldots, m$, $e \in E$, Startknoten s, Zielknoten t

Output: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)

Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ for $i = 1..m$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$

while L_{open} nicht leer **do**

  wähle $l \in L_{open}$,

  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$

  for $v \in Adj(l.knoten)$ **do**

    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für alle $i = 1..m$, $l'.\pi = l$

    Dominanzcheck$(l', L_{open}, L_{closed})$

  end for

end while

return Labels $l \in L_{closed}$ mit $l.knoten = t$

Clevere Wahl von l?

Wie machen wir **Dominanzcheck**?

Erinnert Sie der Algorithmus an etwas?

Laufzeit?

Was passiert ohne **Dominanzcheck?**

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Input: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit m positiven Kantenlabels w_e^i für $i = 1, \ldots, m$, $e \in E$, Startknoten s, Zielknoten t

Output: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)

Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ for $i = 1..m$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$

while L_{open} nicht leer **do**

  wähle $l \in L_{open}$,

  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$

  for $v \in Adj(l.knoten)$ **do**

    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für alle $i = 1..m$, $l'.\pi = l$

    Dominanzcheck$(l', L_{open}, L_{closed})$

  end for

end while

return Labels $l \in L_{closed}$ mit $l.knoten = t$

Clevere Wahl von l?
Wie machen wir **Dominanzcheck**?
Erinnert Sie der Algorithmus an etwas?
Laufzeit?
Was passiert ohne **Dominanzcheck?**

Mehr dazu auf dem aktuellen Übungsblatt

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Satz: Wenn **Dominanzfunktion**$(l, L_{open}, L_{closed})$ in jedem Schritt (genau) alle dominierten Label in der Menge , $L_{open} \cup L_{closed}$ findet und aus L_{open} bzw L_{closed} entfernt, dann ist der Algorithmus korrekt.

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Satz: Wenn **Dominanzfunktion**$(l, L_{open}, L_{closed})$ in jedem Schritt (genau) alle dominierten Label in der Menge , $L_{open} \cup L_{closed}$ findet und aus L_{open} bzw L_{closed} entfernt, dann ist der Algorithmus korrekt.

Beweis: 1. 1-zu-1-Korrespondenz *s-l.knoten*-Weg p und Label l mit Länge $l - d_i$ bzgl Kantenlabel w_i (Rekonstruktion von Weg durch Vorgängerpointer π).

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Satz: Wenn **Dominanzfunktion**$(l, L_{open}, L_{closed})$ in jedem Schritt (genau) alle dominierten Label in der Menge , $L_{open} \cup L_{closed}$ findet und aus L_{open} bzw L_{closed} entfernt, dann ist der Algorithmus korrekt.

Beweis: 1. 1-zu-1-Korrespondenz *s-l.knoten*-Weg p und Label l mit Länge $l - d_i$ bzgl Kantenlabel w_i (Rekonstruktion von Weg durch Vorgängerpointer π).

2. Angenommen $\exists$ nicht-dominierter Weg *s-t*-Weg p, für den nach Ablauf des Algos kein Label l in L_{closed} enthalten ist.
Sei l_1 das erste Vorgängerlabel von l, das nicht in L_{closed} enthalten ist.
Sei p_1 der zu l_1 gehörige *s-l.knoten*-Weg und p_2 der zweite Teil des Weges p (also $p = p_1 \circ p_2$).
Entweder l_1 wurde nicht erstellt ($\rightarrow$ Widerspruch zu $l_1.\pi \in L_{closed}$)
oder l_1 wurde durch anderes Label l' dominiert
d.h. $p' \leqq p_1$
$\Rightarrow$ Der Pfad $p' \circ p_2$ dominiert den Pfad $p_1 \circ p_2 \Rightarrow$ Widerspruch zu l ist nichtdominiert

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Satz: Wenn **Dominanzfunktion**$(l, L_{open}, L_{closed})$ in jedem Schritt (genau) alle dominierten Label in der Menge , $L_{open} \cup L_{closed}$ findet und aus L_{open} bzw L_{closed} entfernt, dann ist der Algorithmus korrekt.

Beweis: 1. 1-zu-1-Korrespondenz s-$l.knoten$-Weg p und Label l mit Länge $l - d_i$ bzgl Kantenlabel w_i (Rekonstruktion von Weg durch Vorgängerpointer π).

2. Nach Ablauf des Algorithmus ist für jeden nichtdominierten s-t-Weg ein Label in L_{closed}.

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Satz: Wenn **Dominanzfunktion**$(l, L_{open}, L_{closed})$ in jedem Schritt (genau) alle dominierten Label in der Menge , $L_{open} \cup L_{closed}$ findet und aus L_{open} bzw L_{closed} entfernt, dann ist der Algorithmus korrekt.

Beweis: 1. 1-zu-1-Korrespondenz s-$l.knoten$-Weg p und Label l mit Länge $l - d_i$ bzgl Kantenlabel w_i (Rekonstruktion von Weg durch Vorgängerpointer π).

2. Nach Ablauf des Algorithmus ist für jeden nichtdominierten s-t-Weg ein Label in L_{closed}.

3. Angenommen nichtdominiertes Label l in L_{closed}.
Widerspruch zur Korrektheit der Dominanzfunktion

Agenda

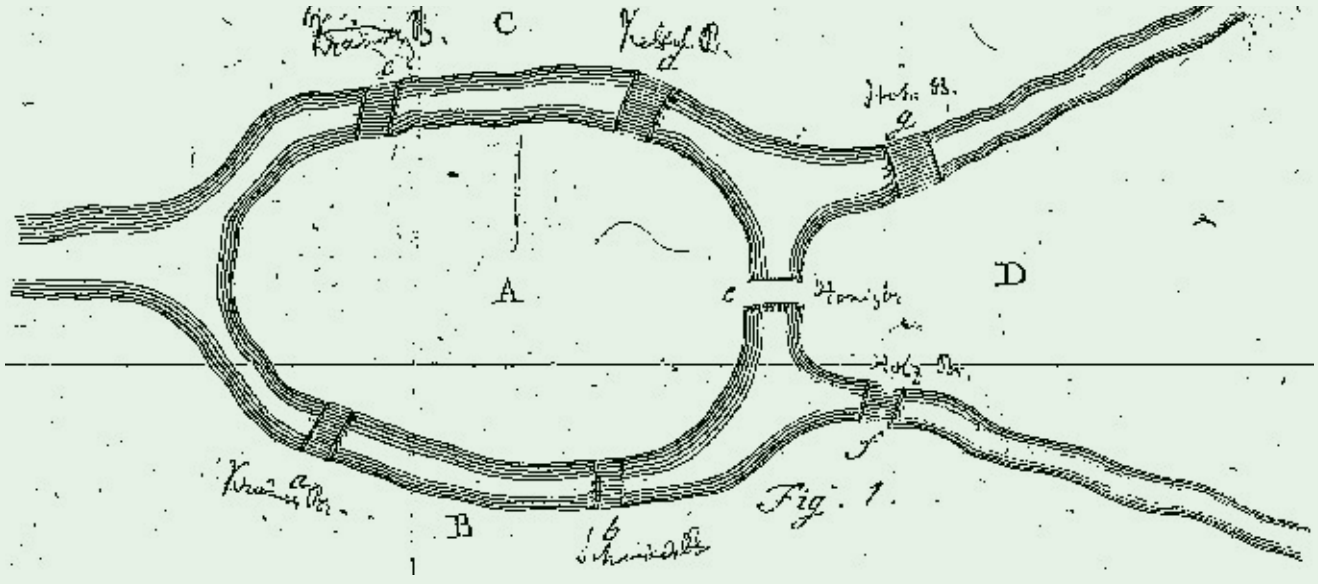
1. Prüfungstermine ✓
2. Wege ✓
3. **Touren**

Das allererste Tourenplanungsproblem

Basel 1707 – St. Petersburg 1783

Königsberger Brückenproblem [Euler, 1741]

Euler möchte einen Spaziergang durch Königsberg machen, auf der er jedes Stadtviertel (getrennt durch die Flüsse) besucht, aber keine Brücke zweimal benutzt...



Das allererste Tourenplanungsproblem

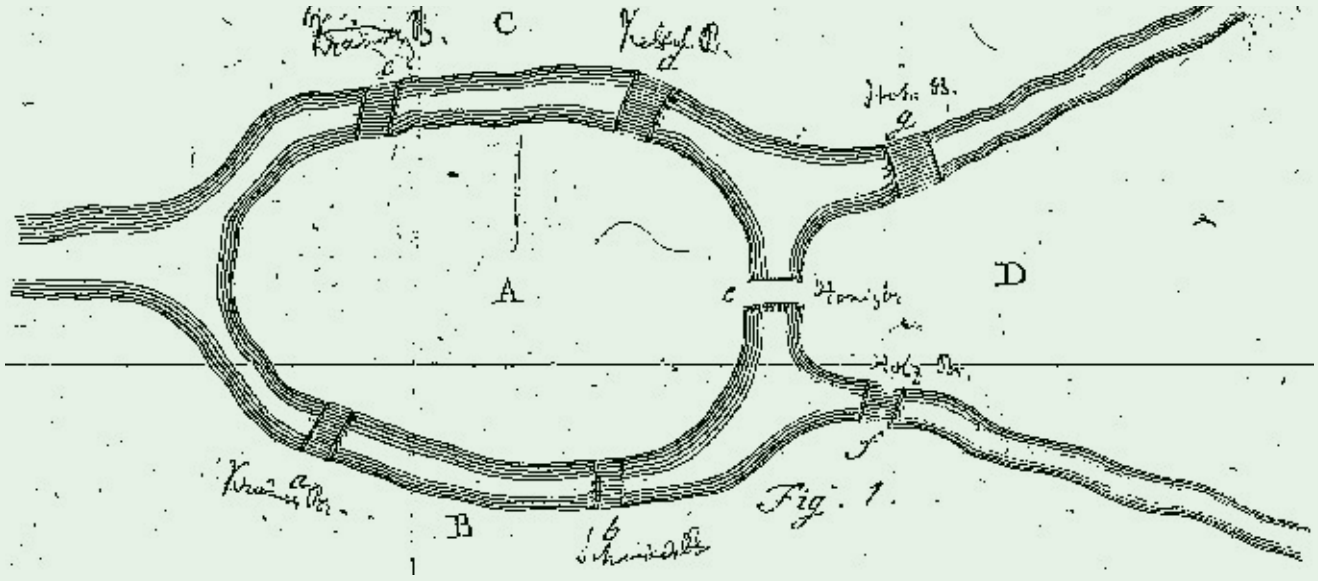
Basel 1707 – St. Petersburg 1783

Königsberger Brückenproblem [Euler, 1741]

Euler möchte einen Spaziergang durch Königsberg machen, auf der er jedes Stadtviertel (getrennt durch die Flüsse) besucht, aber keine Brücke zweimal benutzt...

...und begründet damit die Graphentheorie!

Wie können wir das Königsberger Brückenproble als graphen-theoretisches Problem modellieren?



Das allererste Tourenplanungsproblem

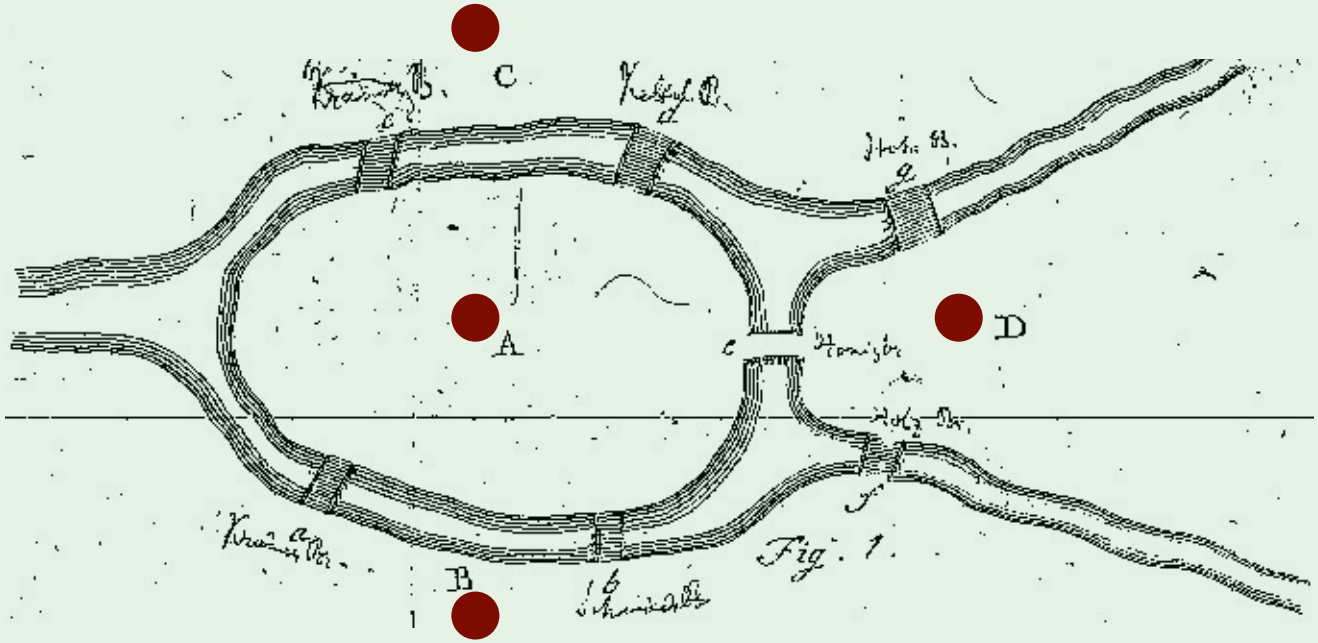
Basel 1707 – St. Petersburg 1783

Königsberger Brückenproblem [Euler, 1741]

Euler möchte einen Spaziergang durch Königsberg machen, auf der er jedes Stadtviertel (getrennt durch die Flüsse) besucht, aber keine Brücke zweimal benutzt...

...und begründet damit die Graphentheorie!

Wie können wir das Königsberger Brückenproble als graphen-theoretisches Problem modellieren?



Das allererste Tourenplanungsproblem

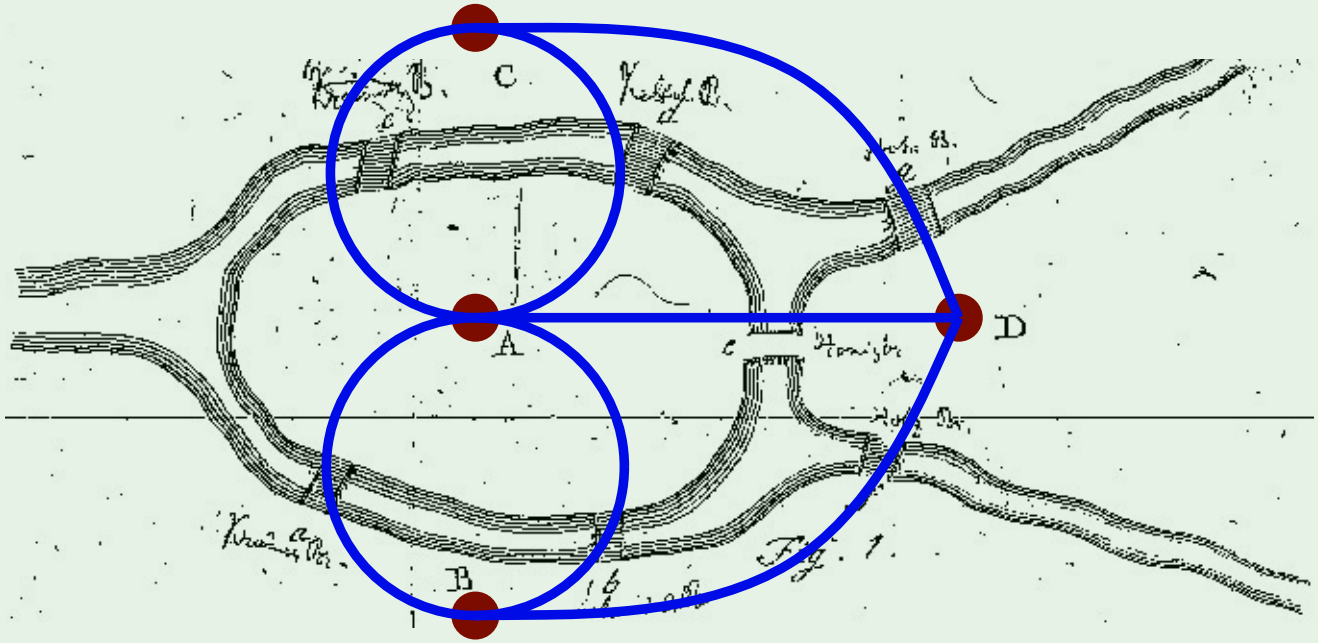
Basel 1707 – St. Petersburg 1783

Königsberger Brückenproblem [Euler, 1741]

Euler möchte einen Spaziergang durch Königsberg machen, auf der er jedes Stadtviertel (getrennt durch die Flüsse) besucht, aber keine Brücke zweimal benutzt...

...und begründet damit die Graphentheorie!

Wie können wir das Königsberger Brückenproble als graphen-theoretisches Problem modellieren?



Das allererste Tourenplanungsproblem

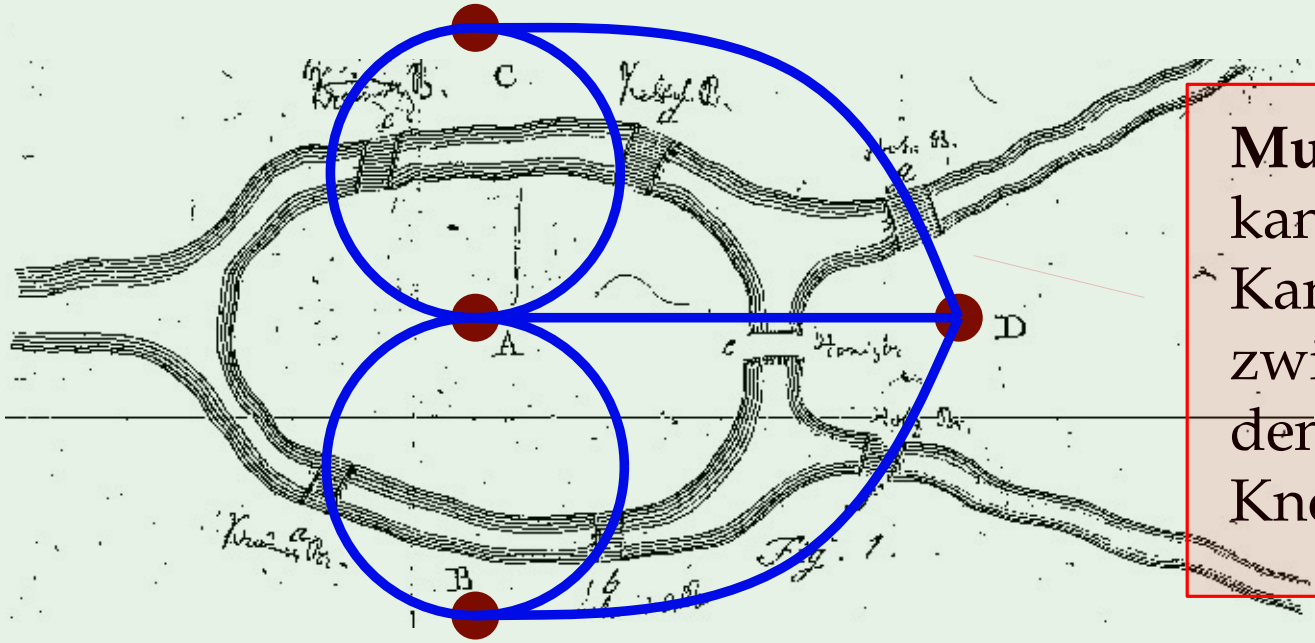
Basel 1707 – St. Petersburg 1783

Königsberger Brückenproblem [Euler, 1741]

Euler möchte einen Spaziergang durch Königsberg machen, auf der er jedes Stadtviertel (getrennt durch die Flüsse) besucht, aber keine Brücke zweimal benutzt...

...und begründet damit die Graphentheorie!

Wie können wir das Königsberger Brückenproble als graphen-theoretisches Problem modellieren?



Multigraph: kann mehrere Kanten zwischen denselben Knoten haben

Eulertouren

Ein **Kreis** in einem Graphen ist ein geschlossener Weg (geschlossen heißt: der Anfangsknoten des Weges ist gleich dem Endknoten).

Ein(e) **Eulerkreis/Eulertour** in einem Graph ist ein *Kreis*, der jede Kante genau einmal benutzt.
Ein Graph, der einen Eulerkreis enthält, heißt **eulersch**.
Ein **Eulerweg** ist ein *Weg*, der jede Kante genau einmal benutzt.

Eulertouren

Ein **Kreis** in einem Graphen ist ein geschlossener Weg (geschlossen heißt: der Anfangsknoten des Weges ist gleich dem Endknoten).

Ein(e) **Eulerkreis/Eulertour** in einem Graph ist ein *Kreis*, der jede Kante genau einmal benutzt.
Ein Graph, der einen Eulerkreis enthält, heißt **eulersch**.
Ein **Eulerweg** ist ein *Weg*, der jede Kante genau einmal benutzt.

Gibt es in jedem Graph einen Eulerkreis?

Eulertouren

Ein **Kreis** in einem Graphen ist ein geschlossener Weg (geschlossen heißt: der Anfangsknoten des Weges ist gleich dem Endknoten).

Ein(e) **Eulerkreis/Eulertour** in einem Graph ist ein *Kreis*, der jede Kante genau einmal benutzt.
Ein Graph, der einen Eulerkreis enthält, heißt **eulersch**.
Ein **Eulerweg** ist ein *Weg*, der jede Kante genau einmal benutzt.

Gibt es in jedem Graph einen Eulerkreis?

Gibt es in jedem Graph einen Eulerweg?

Eulertouren

Ein **Kreis** in einem Graphen ist ein geschlossener Weg (geschlossen heißt: der Anfangsknoten des Weges ist gleich dem Endknoten).

Ein(e) **Eulerkreis/Eulertour** in einem Graph ist ein *Kreis*, der jede Kante genau einmal benutzt.
Ein Graph, der einen Eulerkreis enthält, heißt **eulersch**.
Ein **Eulerweg** ist ein *Weg*, der jede Kante genau einmal benutzt.

Gibt es in jedem Graph einen Eulerkreis?

Gibt es in jedem Graph einen Eulerweg?

Woran erkenne ich, dass es in einem Graph einen Eulerkreis/weg gibt?

Eulertouren

Ein **Kreis** in einem Graphen ist ein geschlossener Weg (geschlossen heißt: der Anfangsknoten des Weges ist gleich dem Endknoten).

Ein(e) **Eulerkreis/Eulertour** in einem Graph ist ein *Kreis*, der jede Kante genau einmal benutzt.
Ein Graph, der einen Eulerkreis enthält, heißt **eulersch**.
Ein **Eulerweg** ist ein *Weg*, der jede Kante genau einmal benutzt.

Gibt es in jedem Graph einen Eulerkreis?

Gibt es in jedem Graph einen Eulerweg?

Woran erkenne ich, dass es in einem Graph einen Eulerkreis/weg gibt?

Ist der Eulerkreis/weg (wenn es einen gibt) eindeutig?

Eulertouren

Ein **Kreis** in einem Graphen ist ein geschlossener Weg (geschlossen heißt: der Anfangsknoten des Weges ist gleich dem Endknoten).

Ein(e) **Eulerkreis/Eulertour** in einem Graph ist ein *Kreis*, der jede Kante genau einmal benutzt.
Ein Graph, der einen Eulerkreis enthält, heißt **eulersch**.
Ein **Eulerweg** ist ein *Weg*, der jede Kante genau einmal benutzt.

Gibt es in jedem Graph einen Eulerkreis?

Gibt es in jedem Graph einen Eulerweg?

Woran erkenne ich, dass es in einem Graph einen Eulerkreis/weg gibt?

Ist der Eulerkreis/weg (wenn es einen gibt) eindeutig?

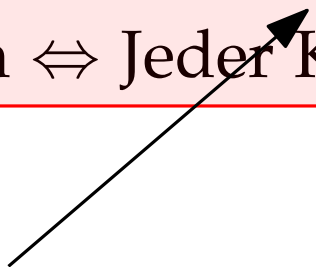
Wie kann ich einen Eulerkreis/weg bestimmen?

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.



Eine **Schlinge** ist eine Kante $\{v, v\}$ (Startknoten = Endknoten).
Ein Graph mit Schlingen hat genau dann einen Eulergraph, wenn er nach Entfernen der Schlingen einen Eulergraph hat.

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

Beweis:

Satz von Euler

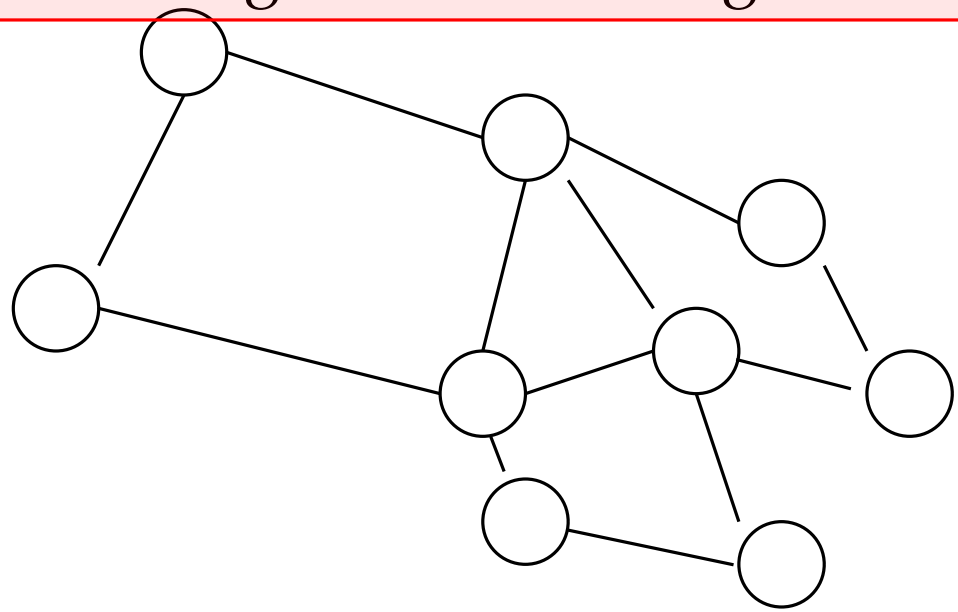
Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

Beweis:

"$\Rightarrow$"(Hinrichtung):

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.



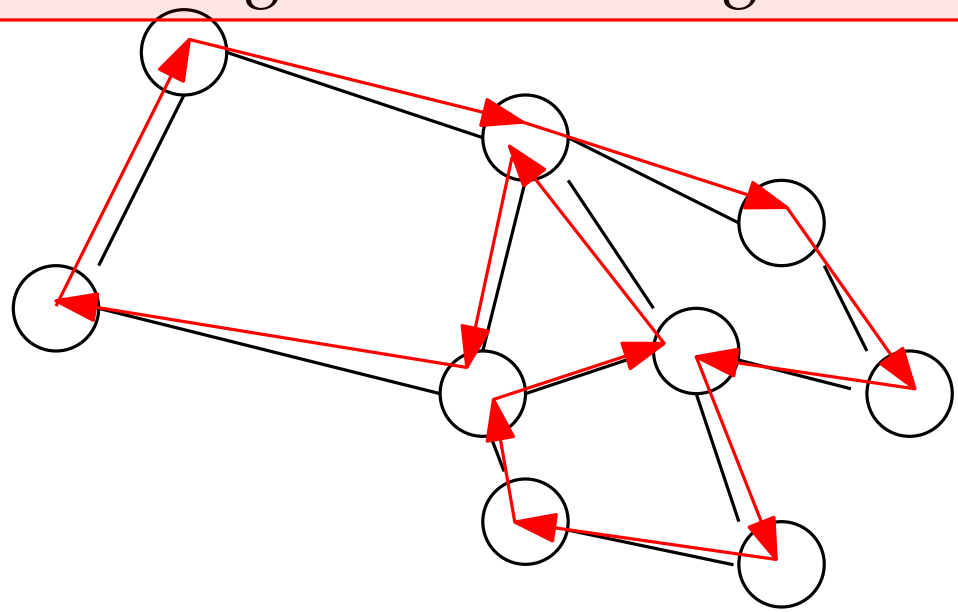
Beweis:

"$\Rightarrow$"(Hinrichtung):

Angenommen G ist eulersch.

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.



Beweis:

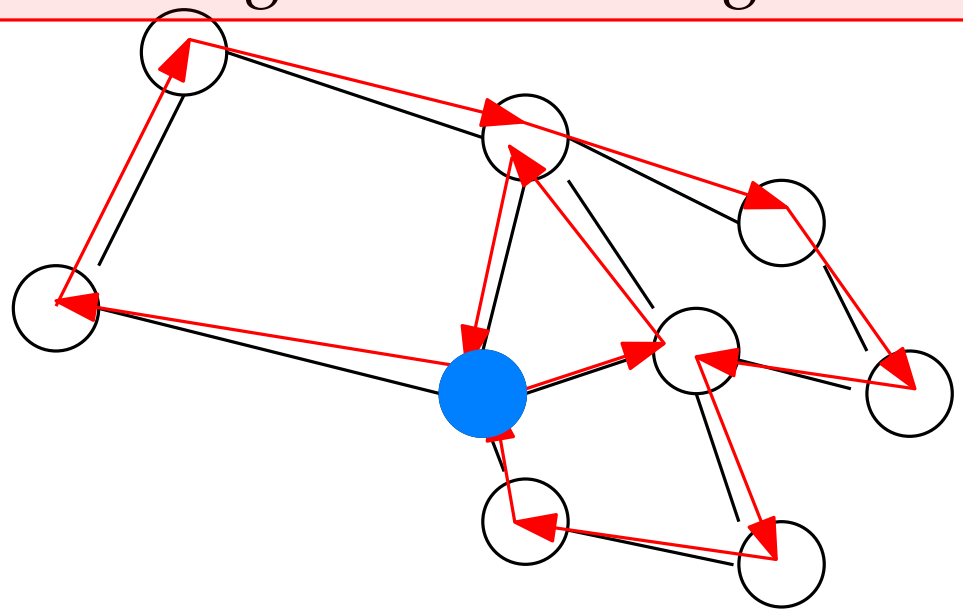
"$\Rightarrow$"(Hinrichtung):

Angenommen G ist eulersch.

Sei K ein Eulerkreis in G.

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.



Beweis:

"$\Rightarrow$"(Hinrichtung):

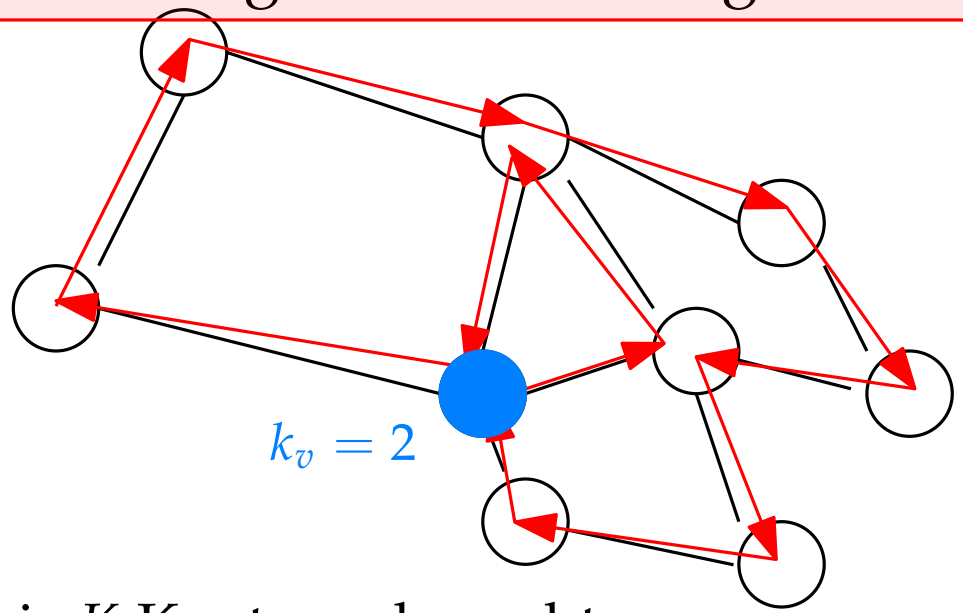
Angenommen G ist eulersch.

Sei K ein Eulerkreis in G.

Sei v ein Knoten in V.

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.



Beweis:

"$\Rightarrow$"(Hinrichtung):

Angenommen G ist eulersch.

Sei K ein Eulerkreis in G.

Sei v ein Knoten in V.

Sei k_v die Anzahl der Male, die Kreis K Knoten v besucht.

Da jede Kante E genau einmal in K benutzt wird, ist $deg(v) = 2 \cdot k_v$.

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

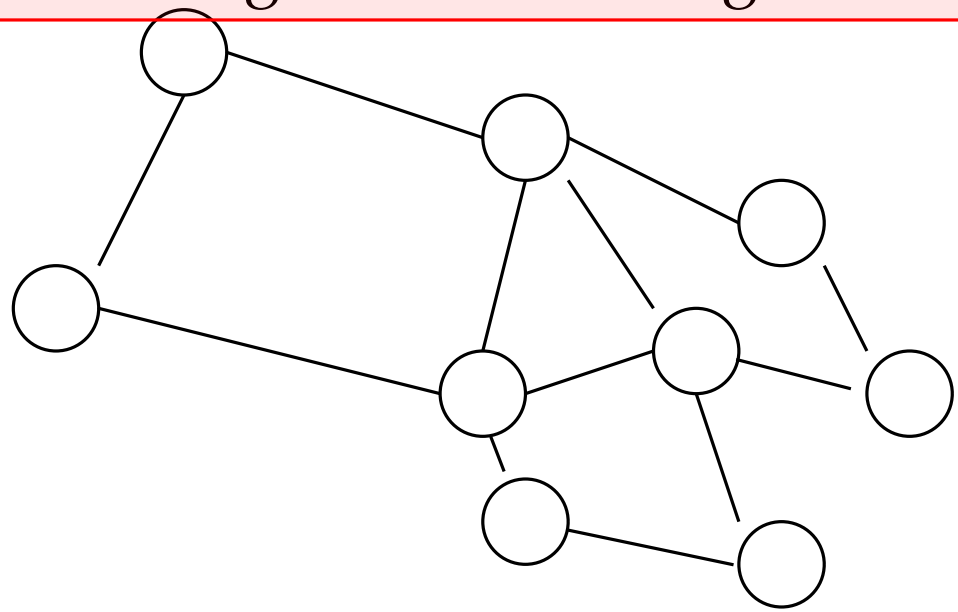
"$\Leftarrow$"(Rückrichtung):

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

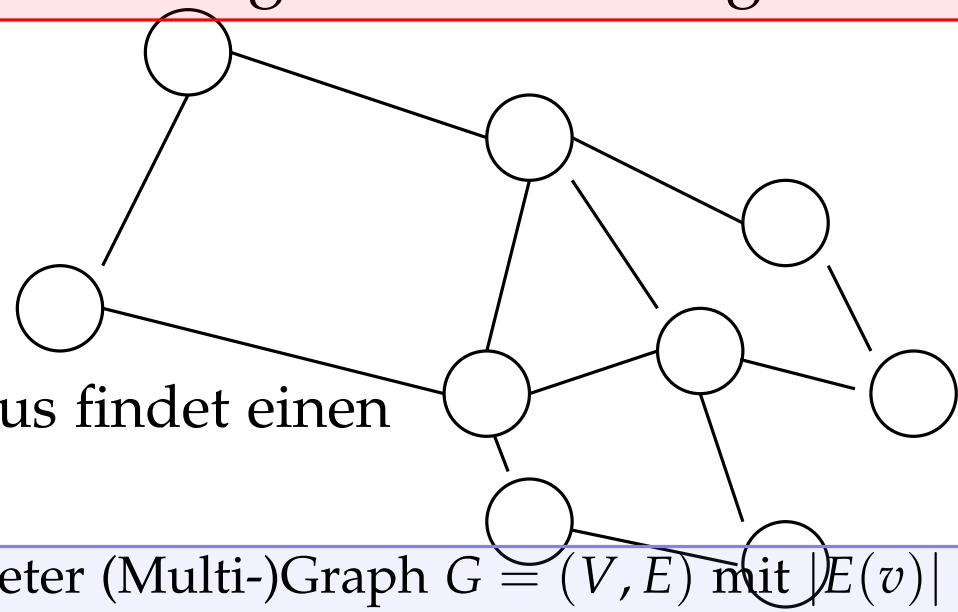
"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.



"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

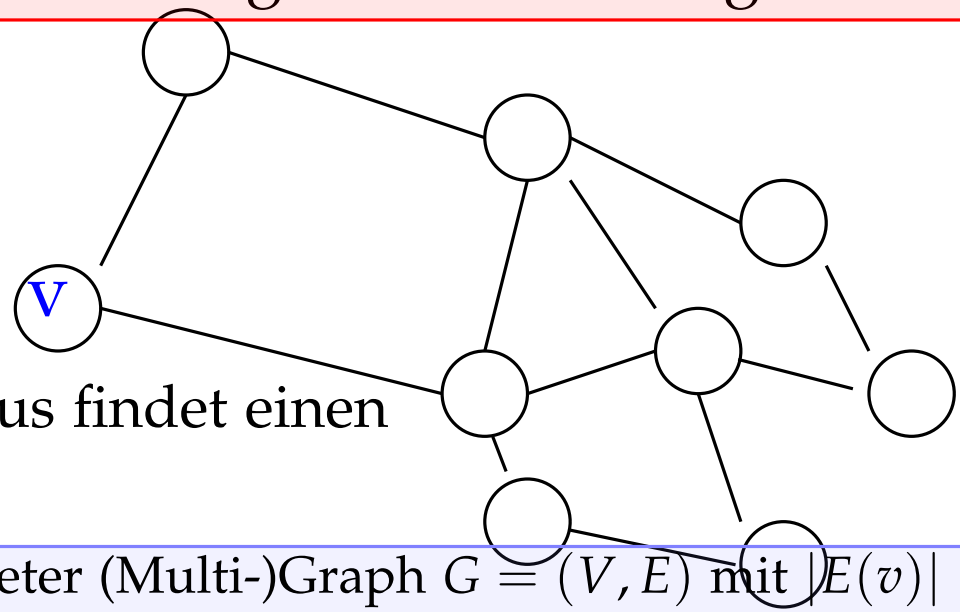
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

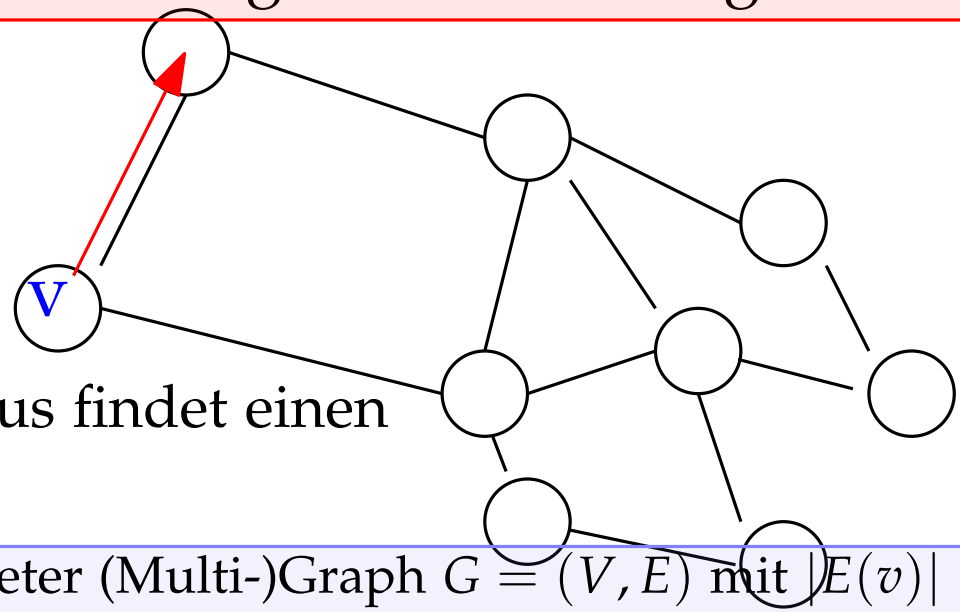
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

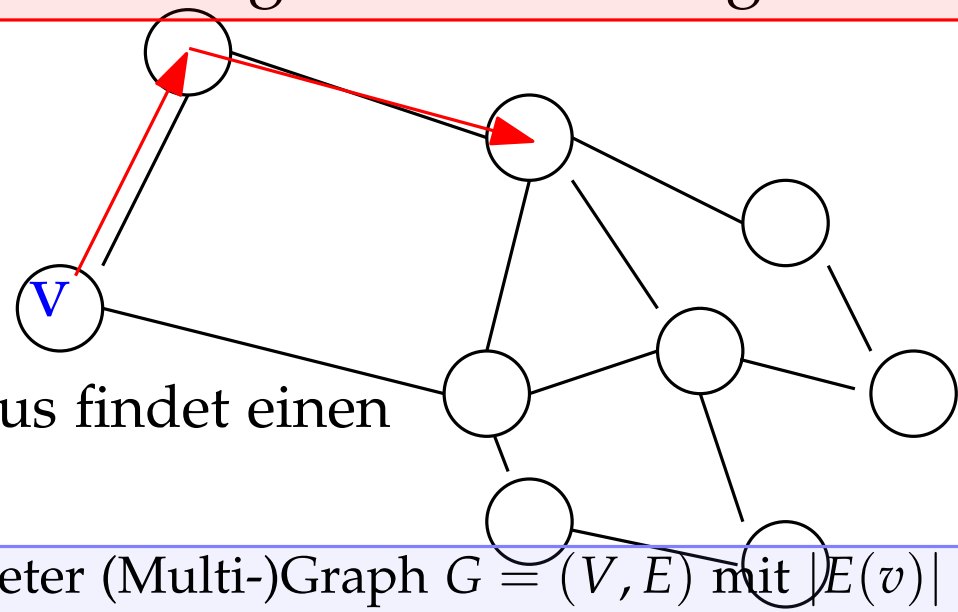
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

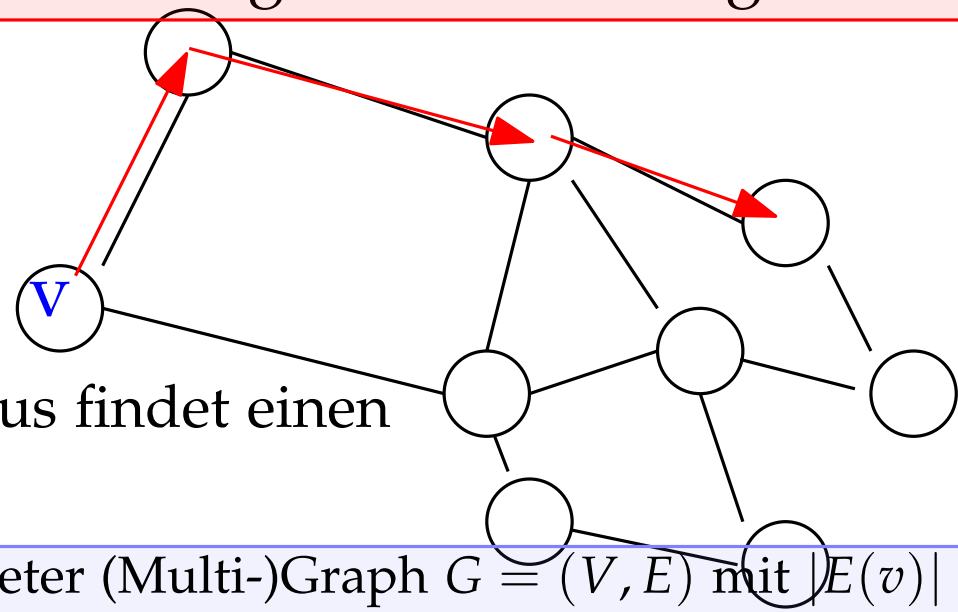
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

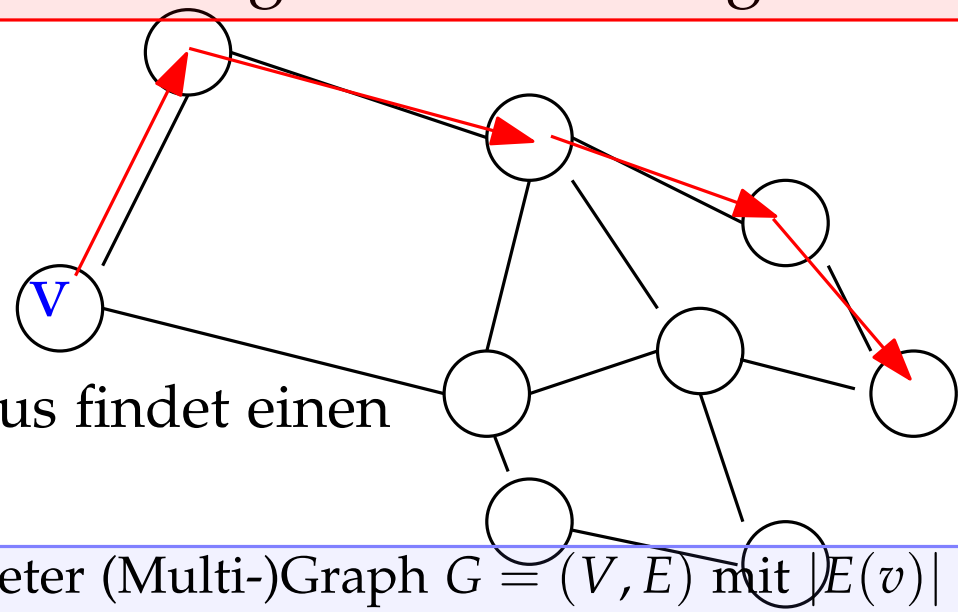
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

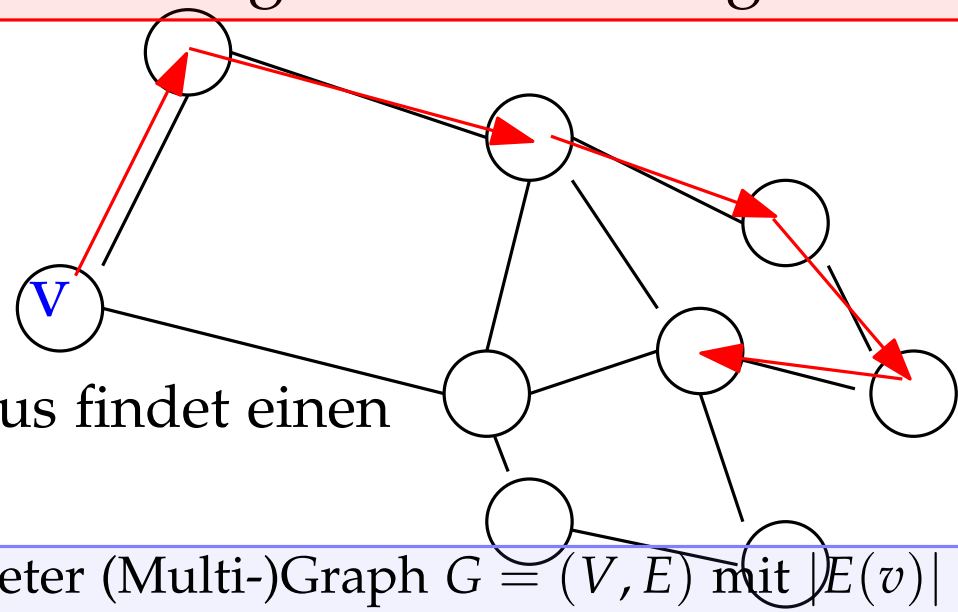
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

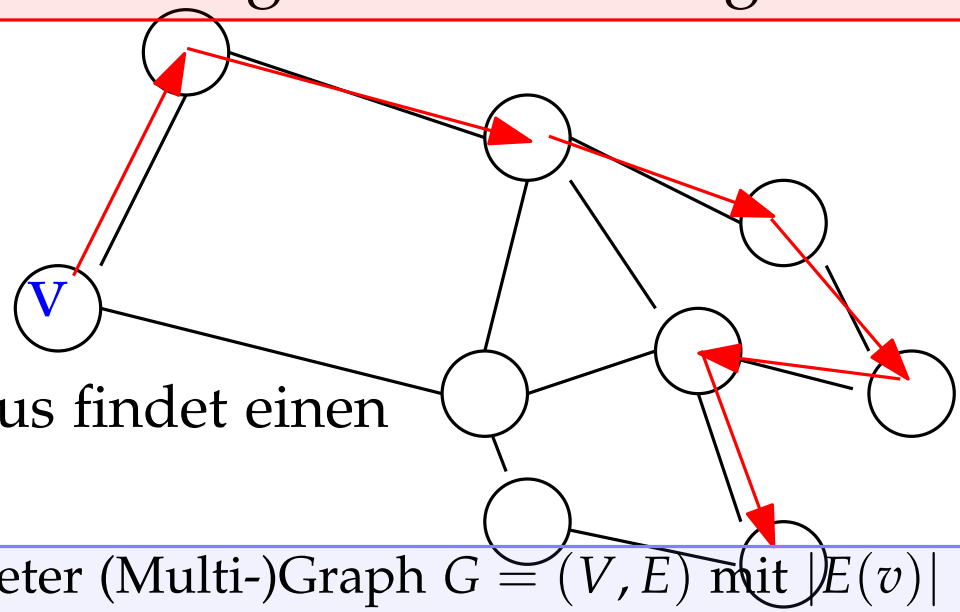
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

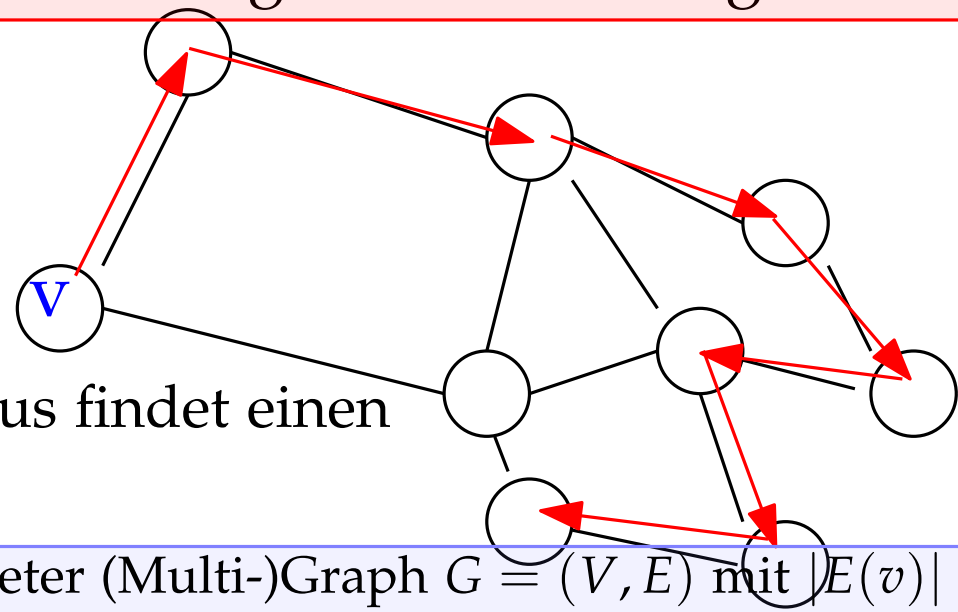
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

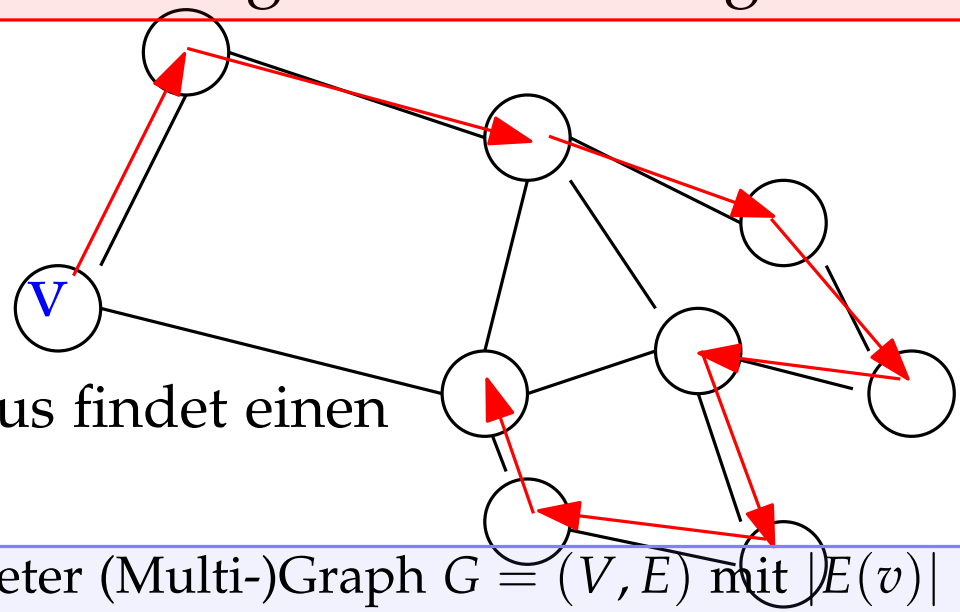
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

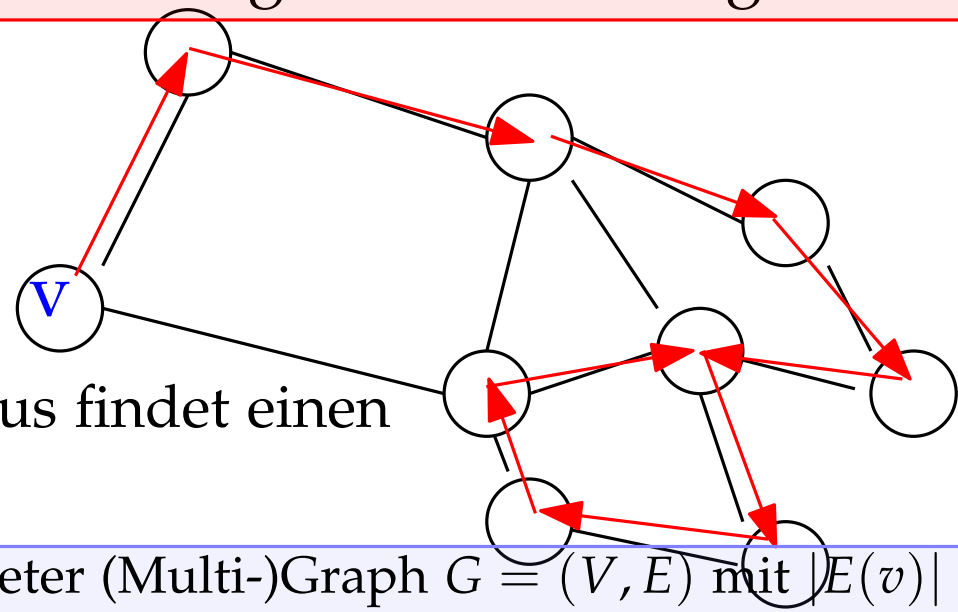
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

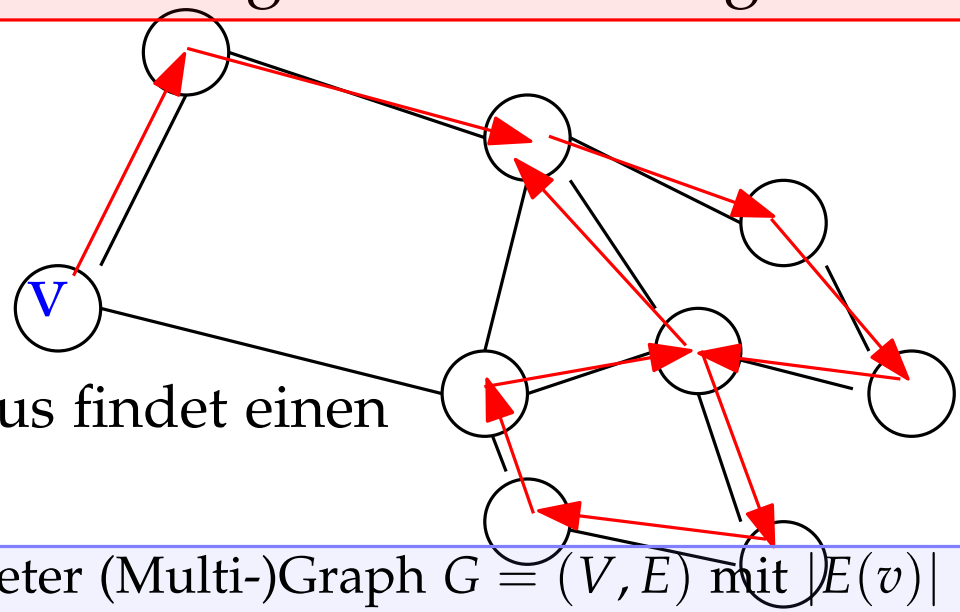
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

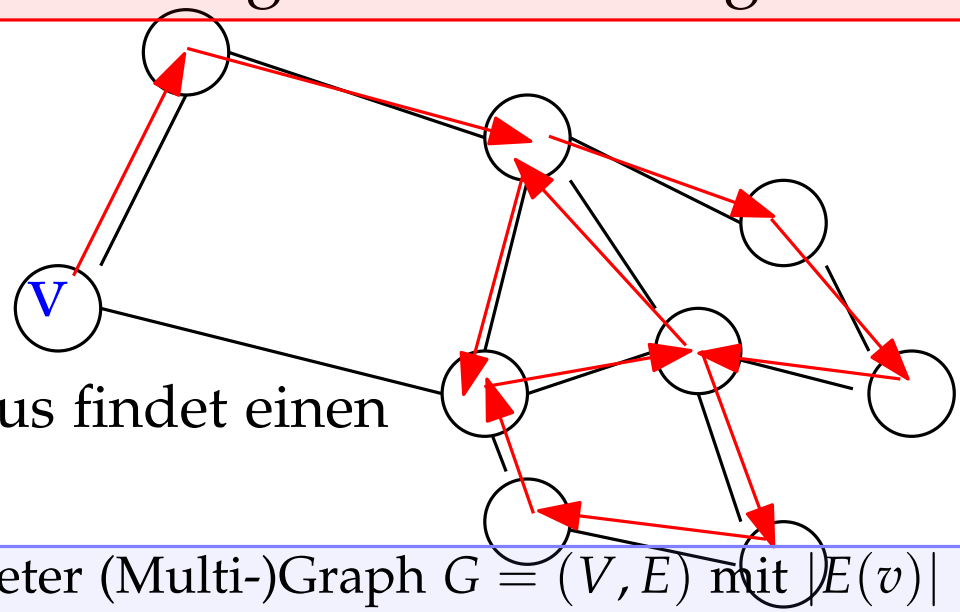
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

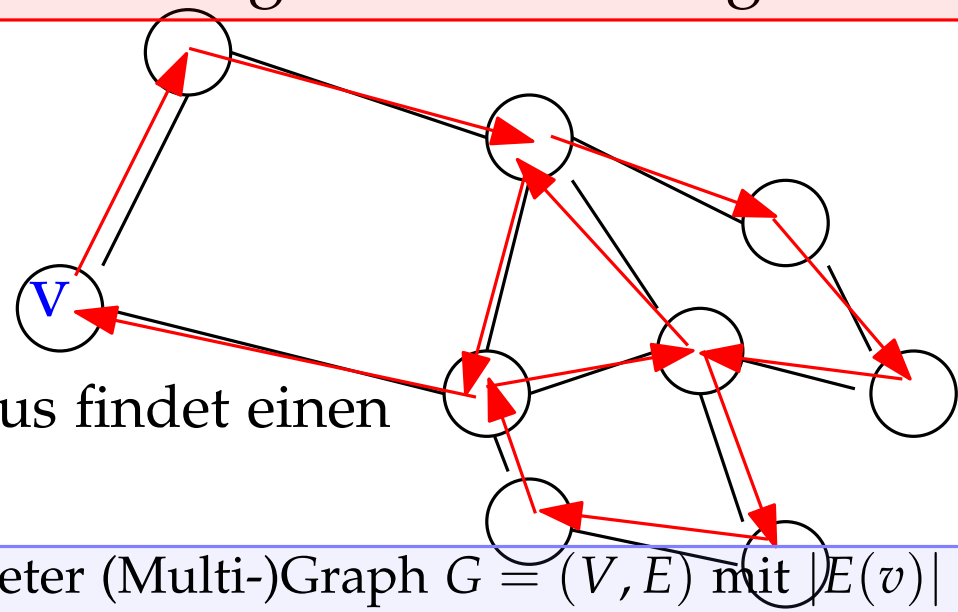
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

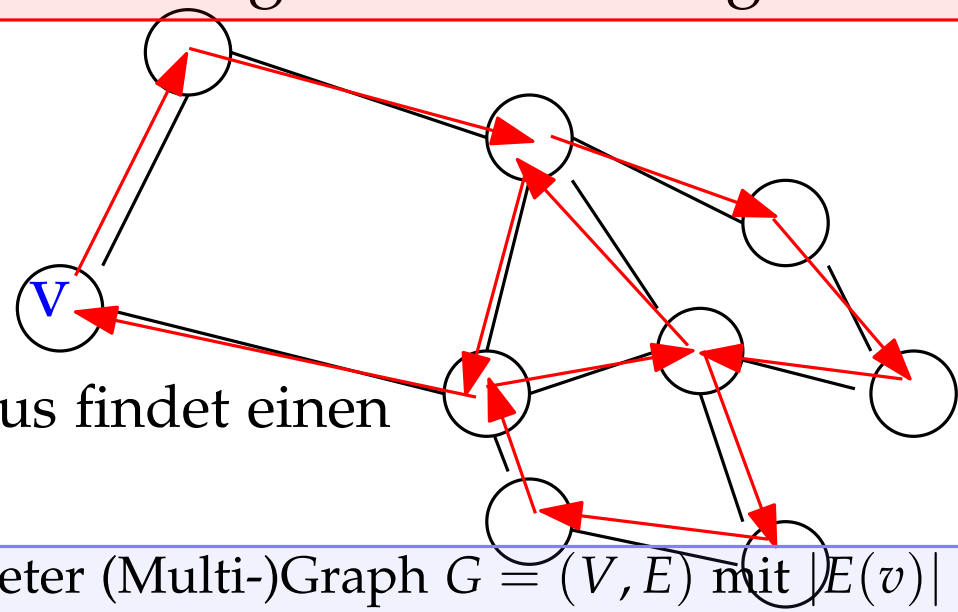
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

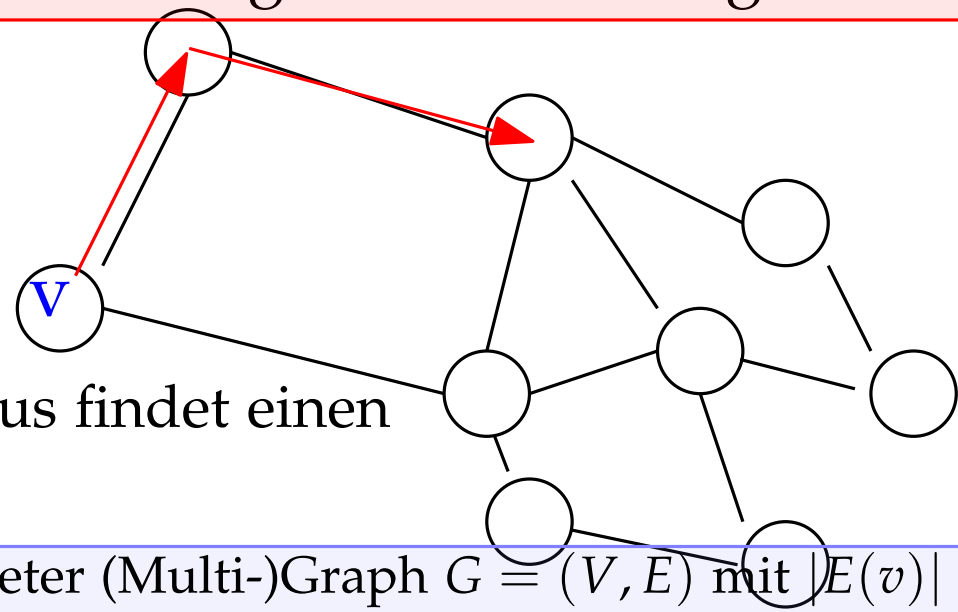
Funktioniert das immer?

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

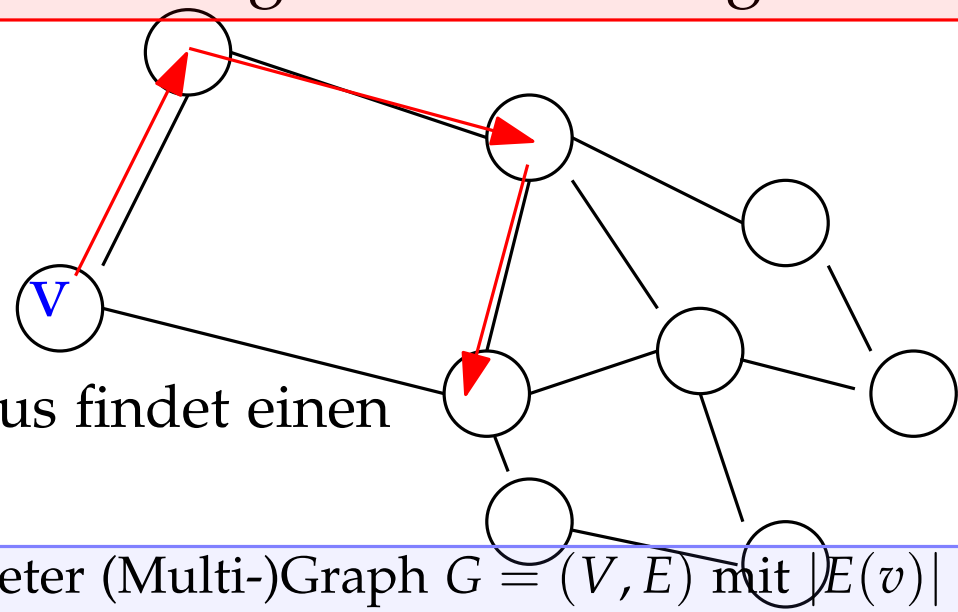
Funktioniert das immer?

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

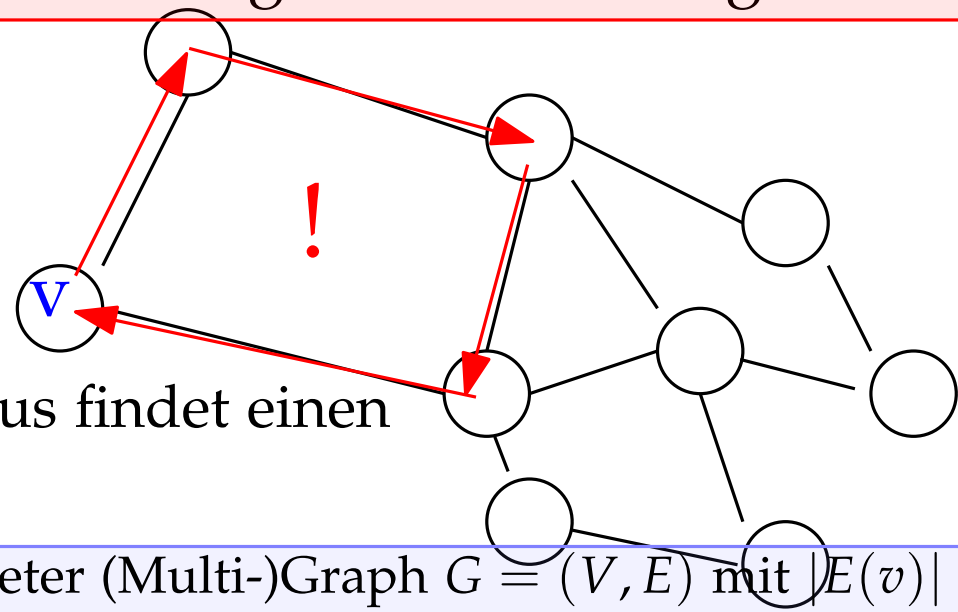
Funktioniert das immer?

Satz von Euler

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender (Multi-)Graph ohne Schlingen. Dann gilt:
G is eulersch $\Leftrightarrow$ Jeder Knoten in V hat geraden Knotengrad.

"$\Leftarrow$"(Rückrichtung):

Sei G ein Graph, in dem alle Knoten geraden Knotengrad haben.



Behauptung: Folgender Algorithmus findet einen Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$
Output: Eulerkreis K in G
Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K = K \circ (\{v, w\}, w)$. Setze $v = w$
end while
return K

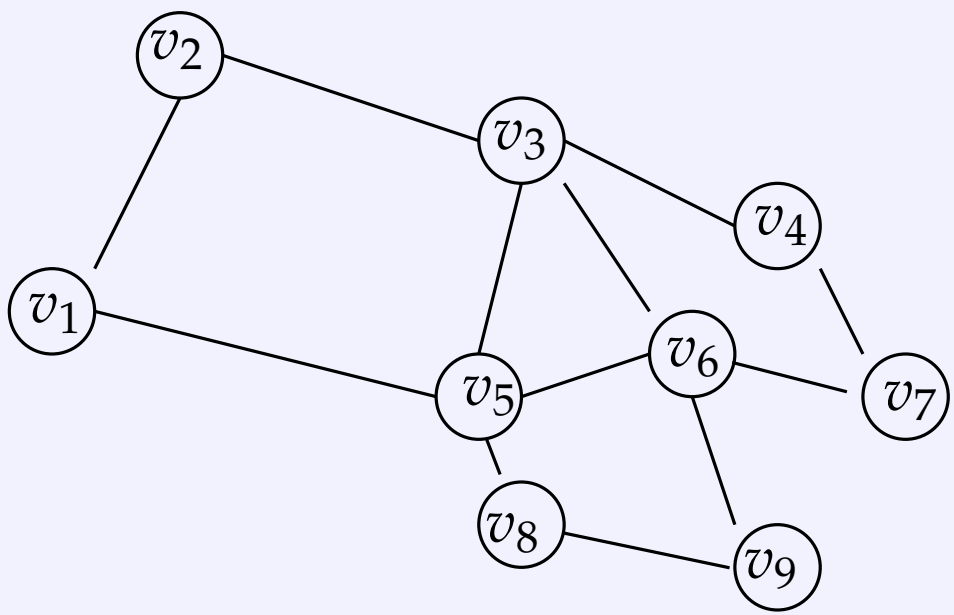
Funktioniert das immer?

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K

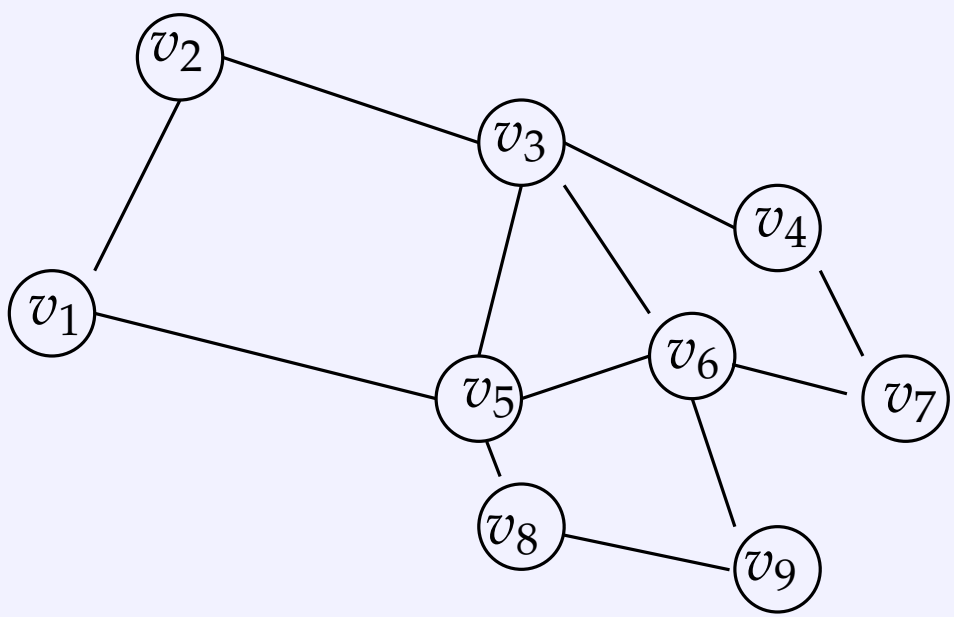


Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
Teile Kreis K in v in zwei Wege K^1 und K^2
Setze $K' = (v)$
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
$K' = K' \circ (\{v, w\}, w)$.
Setze $v = w$
end while
$K = K_1 \circ K' \circ K_2$
end while
return K



$K^1 = (v_1)$

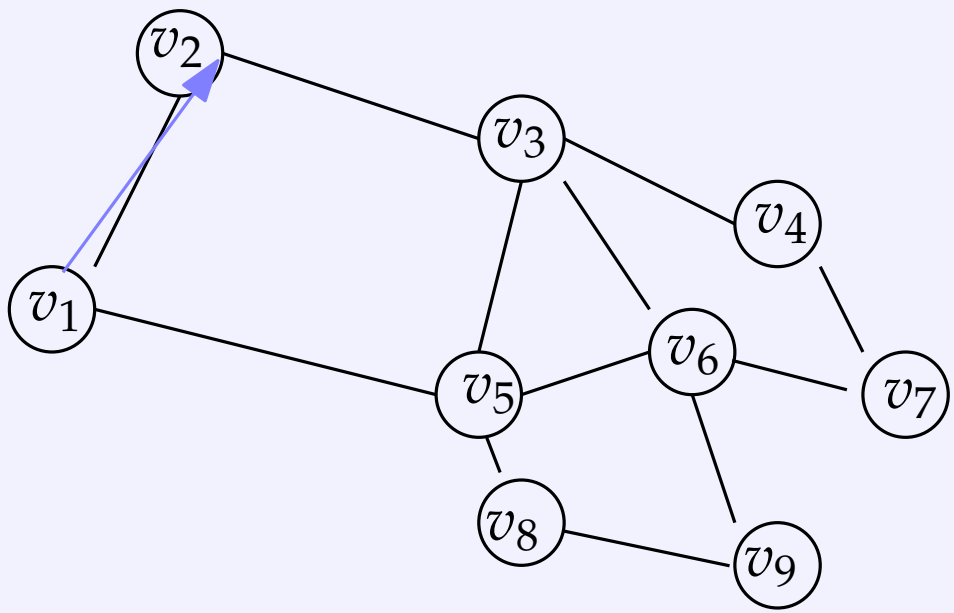
$K^2 = (v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



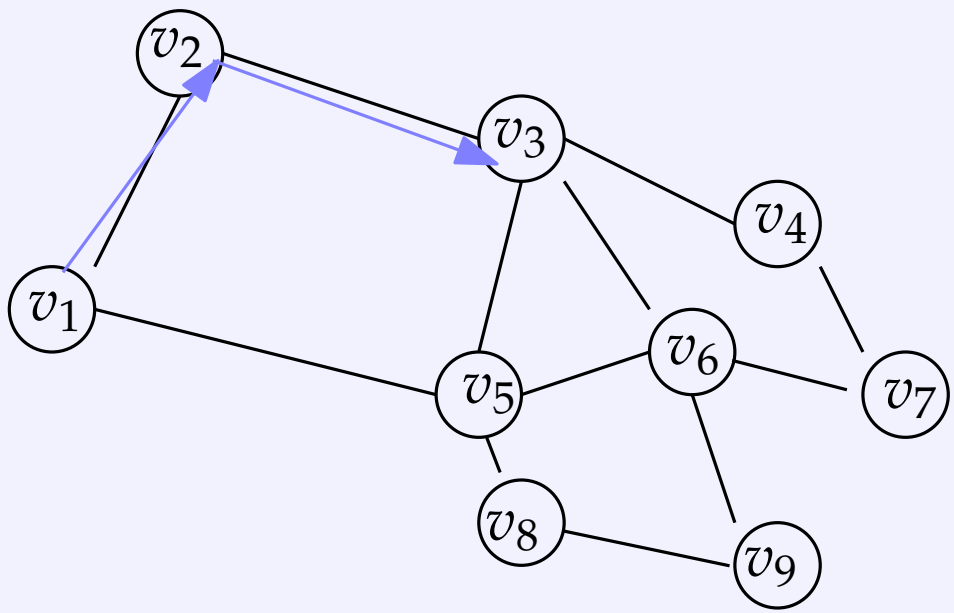
$K^1 = (v_1)$
$K' = (v_1, \{v_1, v_2\}, v_2)$
$K^2 = (v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



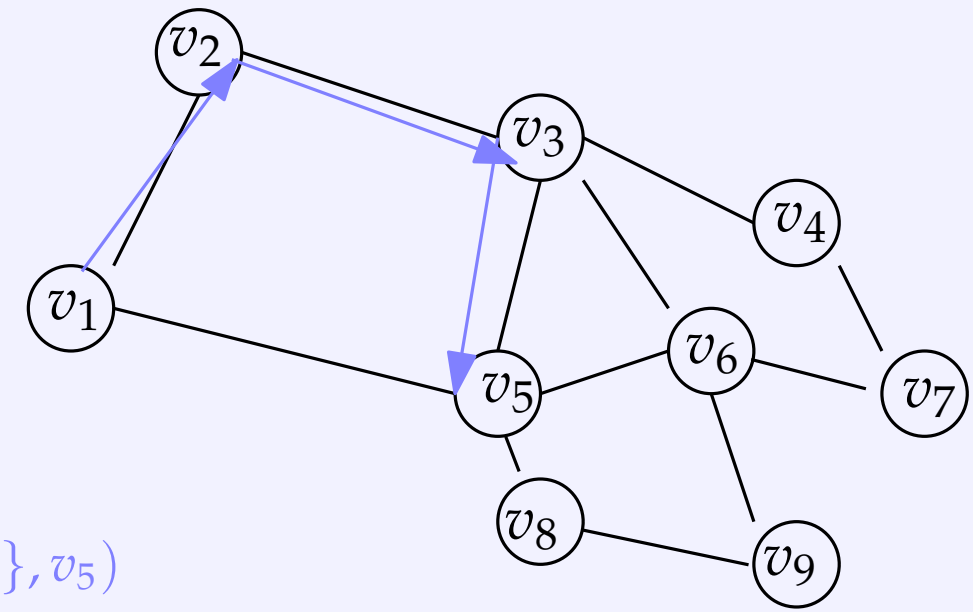
$K^1 = (v_1)$
$K' = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3)$
$K^2 = (v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



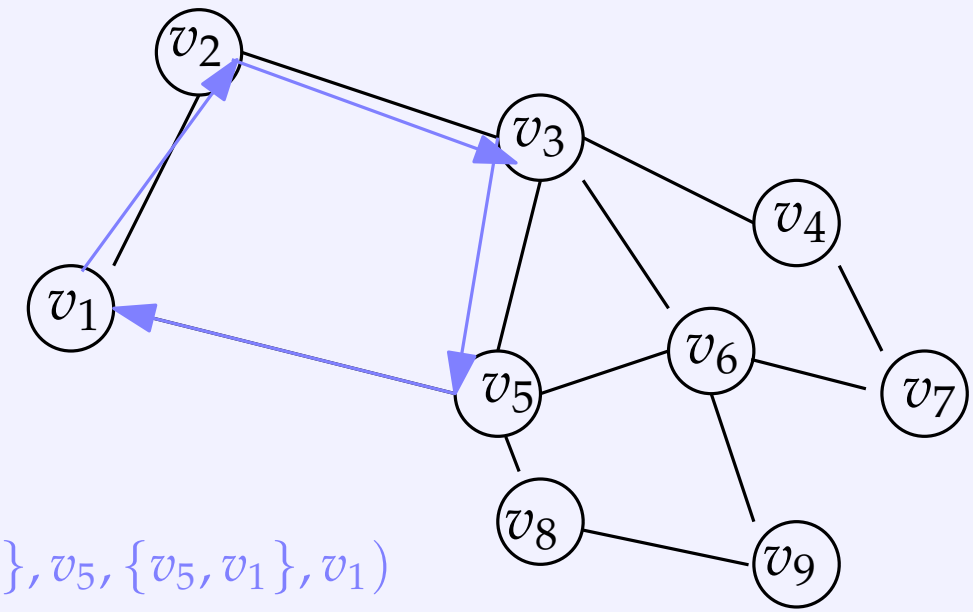
$K^1 = (v_1)$
$K' = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5)$
$K^2 = (v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



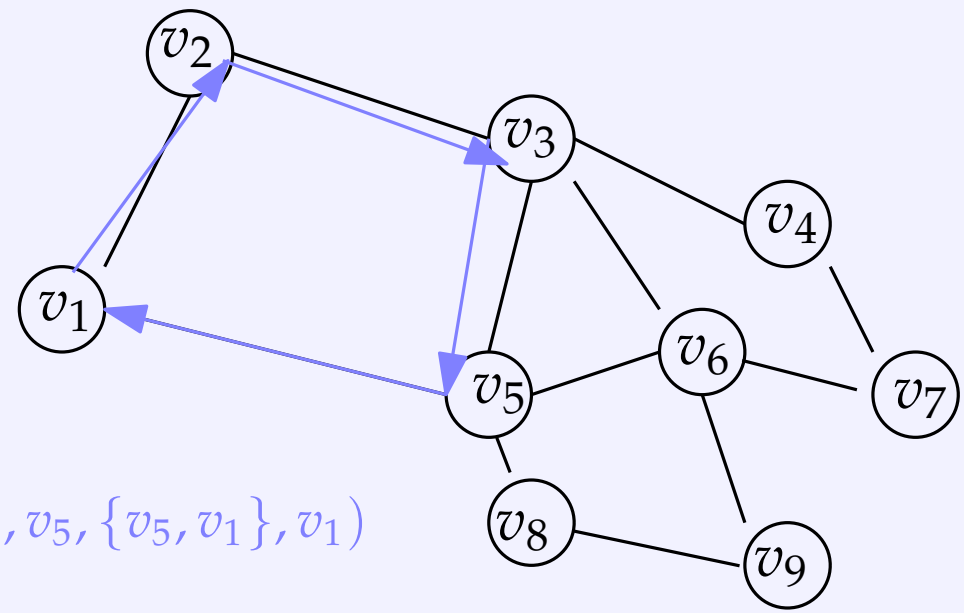
$K^1 = (v_1)$
$K' = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5, \{v_5, v_1\}, v_1)$
$K^2 = (v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



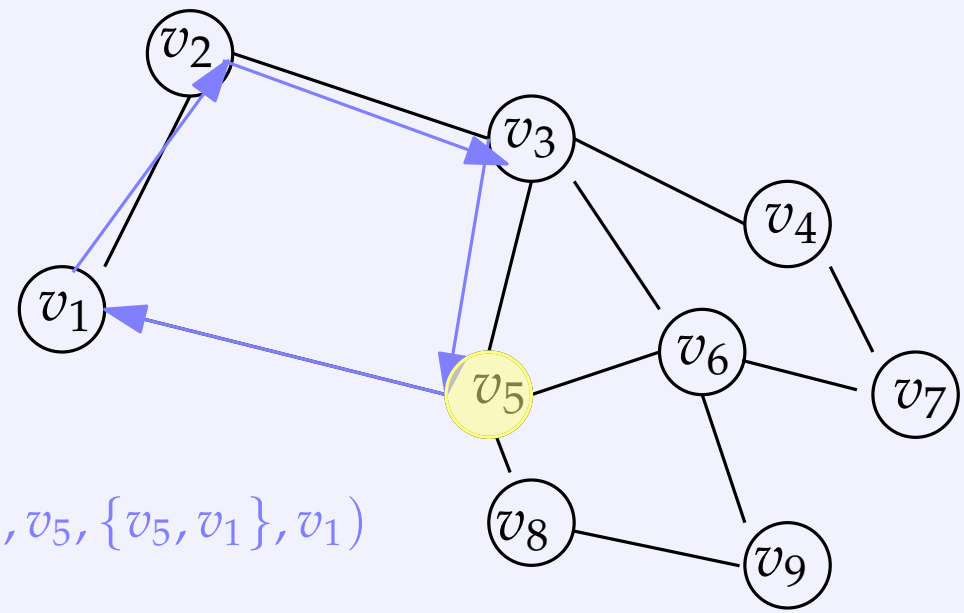
$K = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5, \{v_5, v_1\}, v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



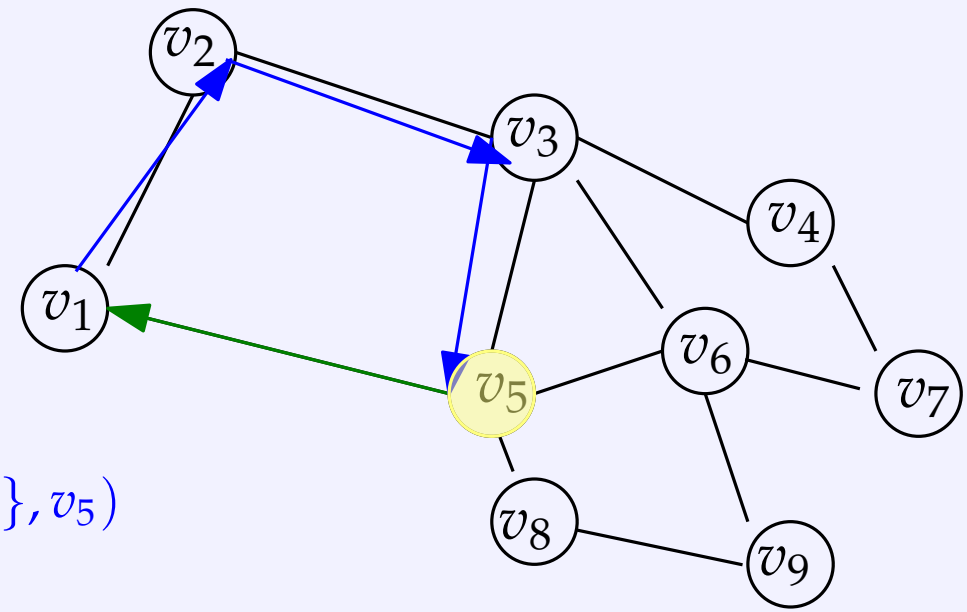
$K = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5, \{v_5, v_1\}, v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



$K_1 = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5)$

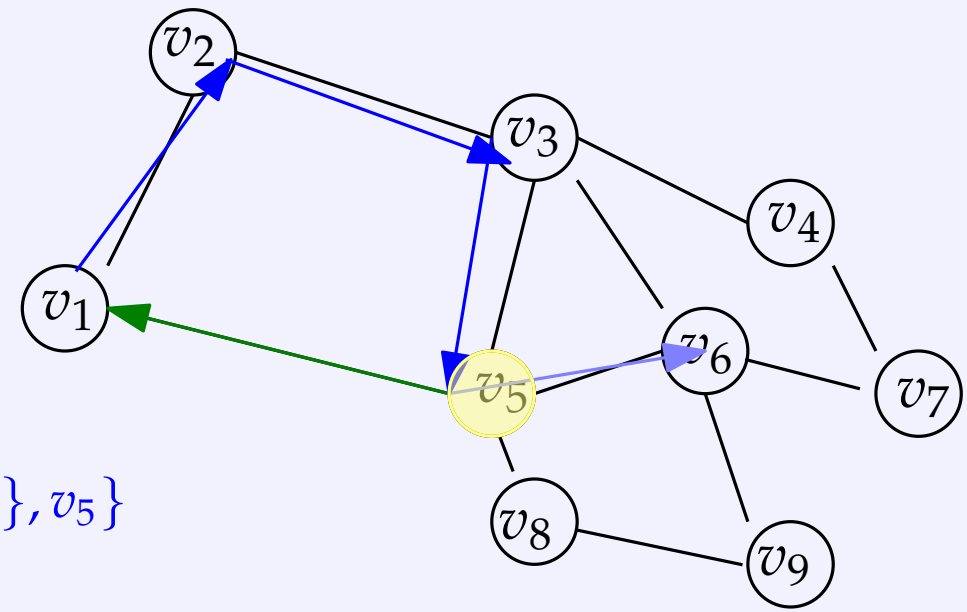
$K_2 = (v_5, \{v_5, v_1\}, v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



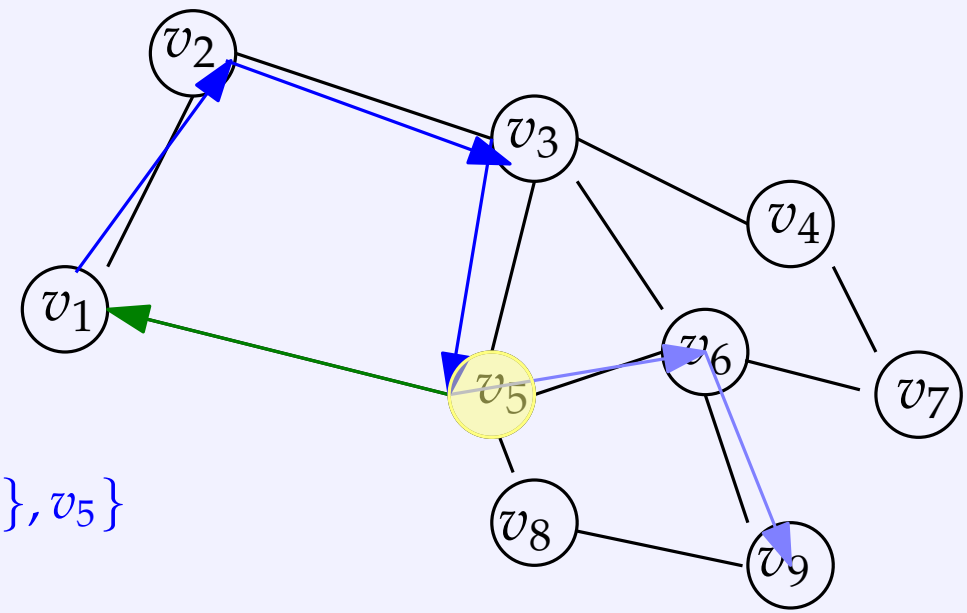
$K_1 = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5)$
$K' = (v_5, \{v_5, v_6\}, v_6)$
$K' = (v_5, \{v_5, v_1\}, v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



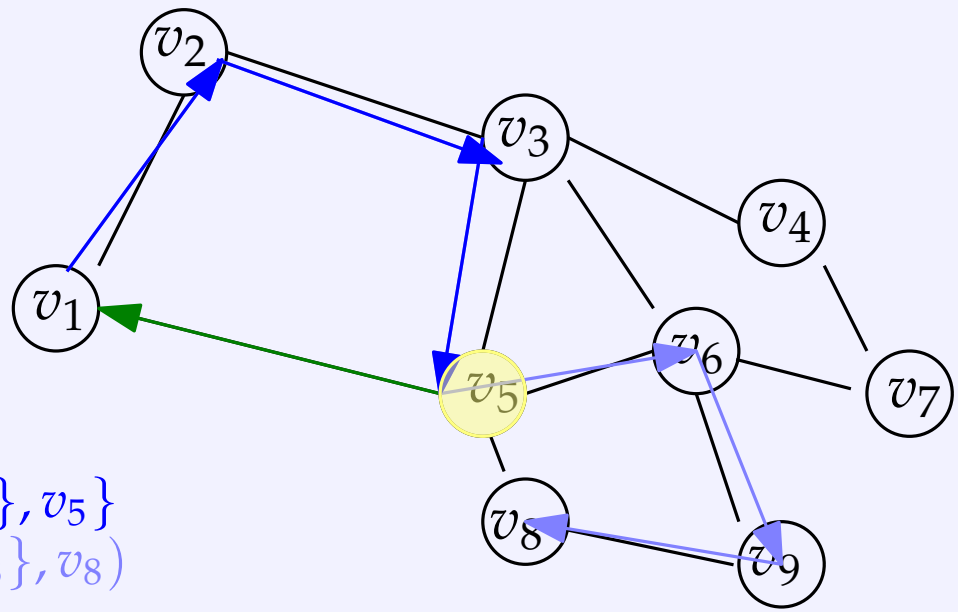
$K_1 = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5)$
$K' = (v_5, \{v_5, v_6, \}, v_6, \{v_6, v_9\}, v_9)$
$K_2 = (v_5, \{v_5, v_1\}, v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



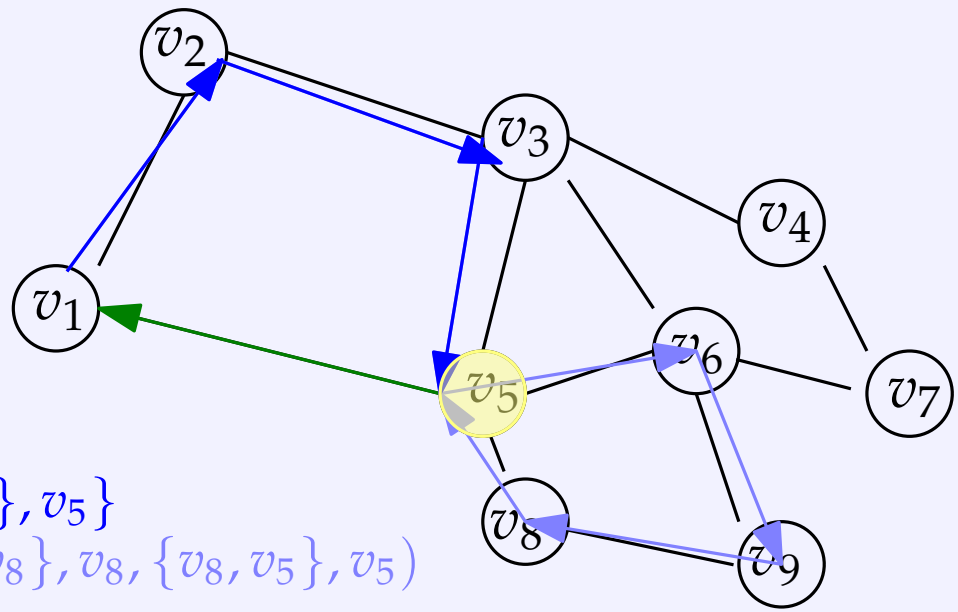
$K_1 = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5)$
$K' = (v_5, \{v_5, v_6, \}, v_6, \{v_6, v_9\}, v_9, \{v_9, v_8\}, v_8)$
$K_2 = (v_5, \{v_5, v_1\}, v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



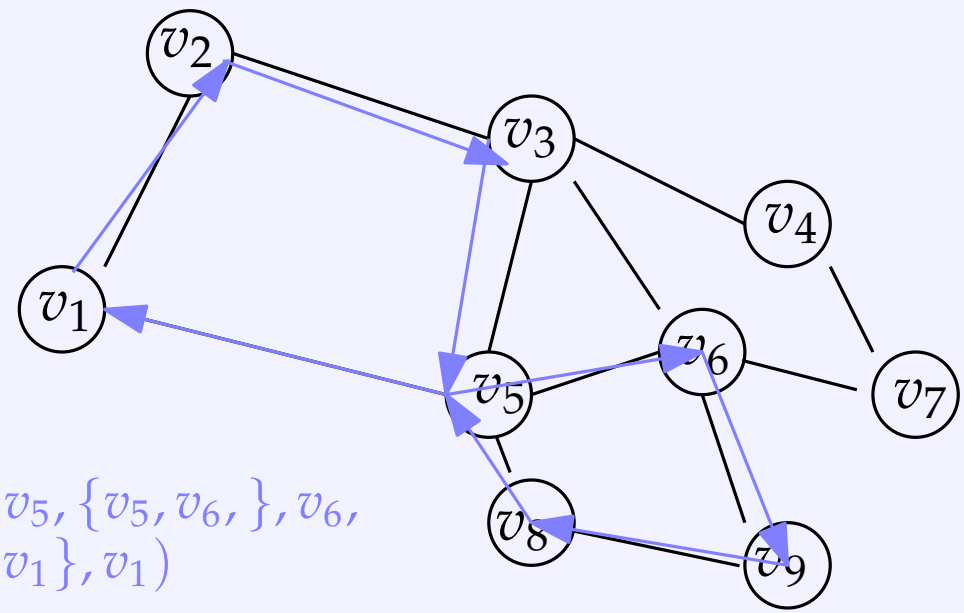
$K_1 = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5\}$
$K' = (, v_5, \{v_5, v_6, \}, v_6, \{v_6, v_9\}, v_9, \{v_9, v_8\}, v_8, \{v_8, v_5\}, v_5)$
$K_2 = (v_5, \{v_5, v_1\}, v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



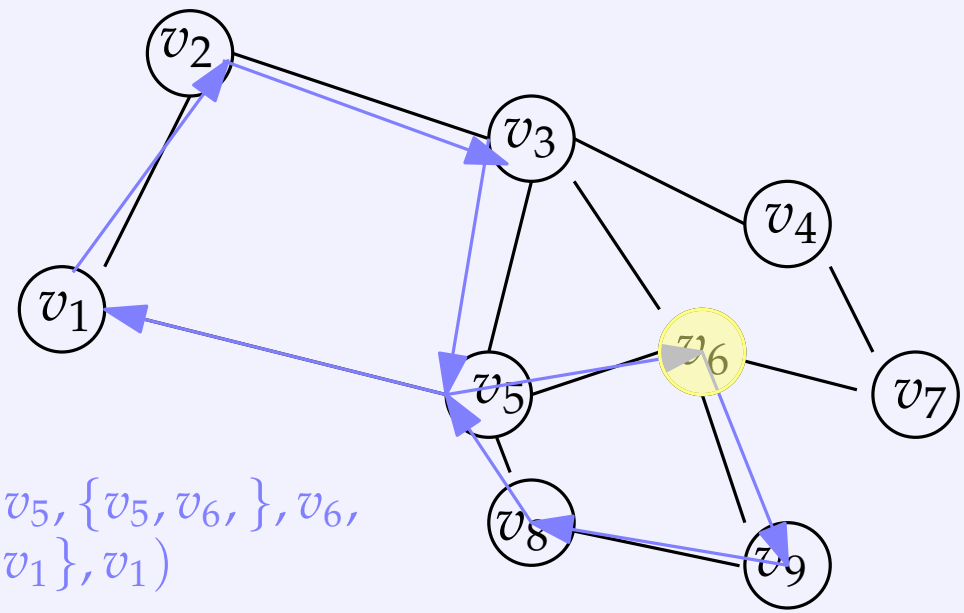
$K = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3 \{v_3, v_5\}, v_5, \{v_5, v_6, \}, v_6, \{v_6, v_9\}, v_9, \{v_9, v_8\}, v_8, \{v_8, v_5\}, v_5, \{v_5, v_1\}, v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



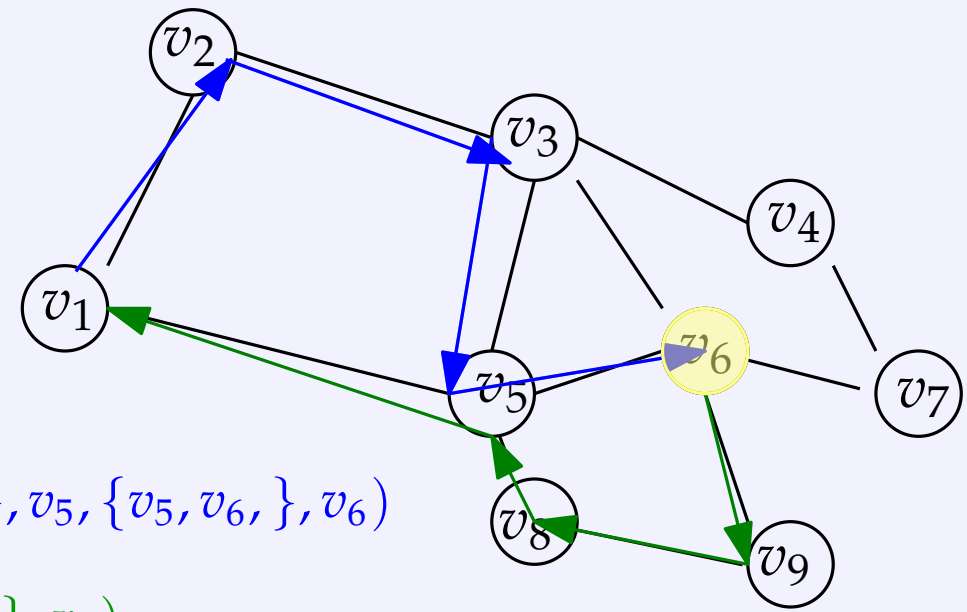
$K = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3 \{v_3, v_5\}, v_5, \{v_5, v_6, \}, v_6,$
$\{v_6, v_9\}, v_9, \{v_9, v_8\}, v_8, \{v_8, v_5\}, v_5, \{v_5, v_1\}, v_1)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
Teile Kreis K in v in zwei Wege K^1 und K^2
Setze $K' = (v)$
while v hat unbenutzte inzidente Kanten **do**
Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
$K' = K' \circ (\{v, w\}, w)$.
Setze $v = w$
end while
$K = K_1 \circ K' \circ K_2$
end while
return K



$K_1 = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3\{v_3, v_5\}, v_5, \{v_5, v_6, \}, v_6)$

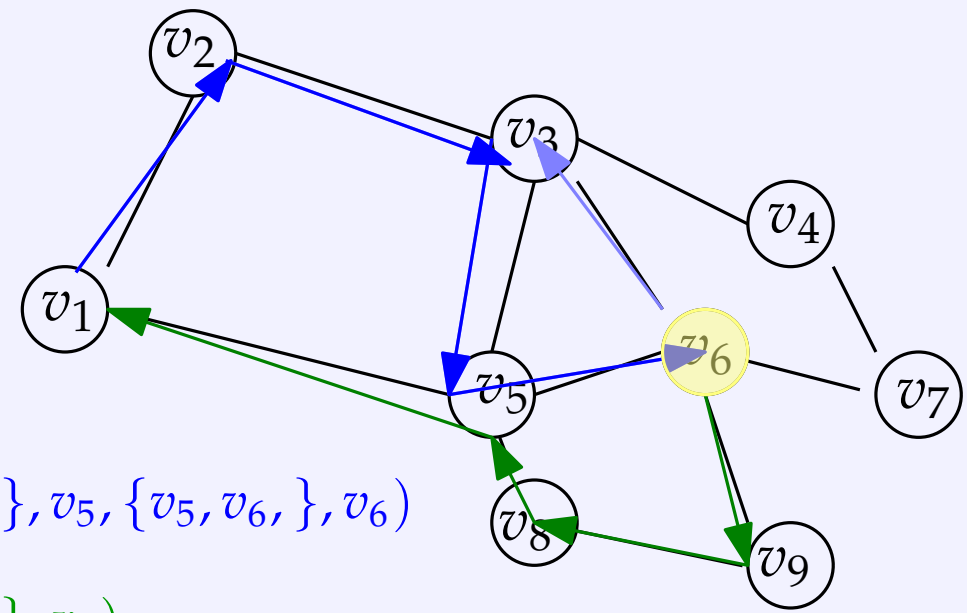
$K_2 = (v_6, \{v_6, v_9\}, v_9, \{v_9, v_8\}, v_8, \{v_8, v_5\}, v_5)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



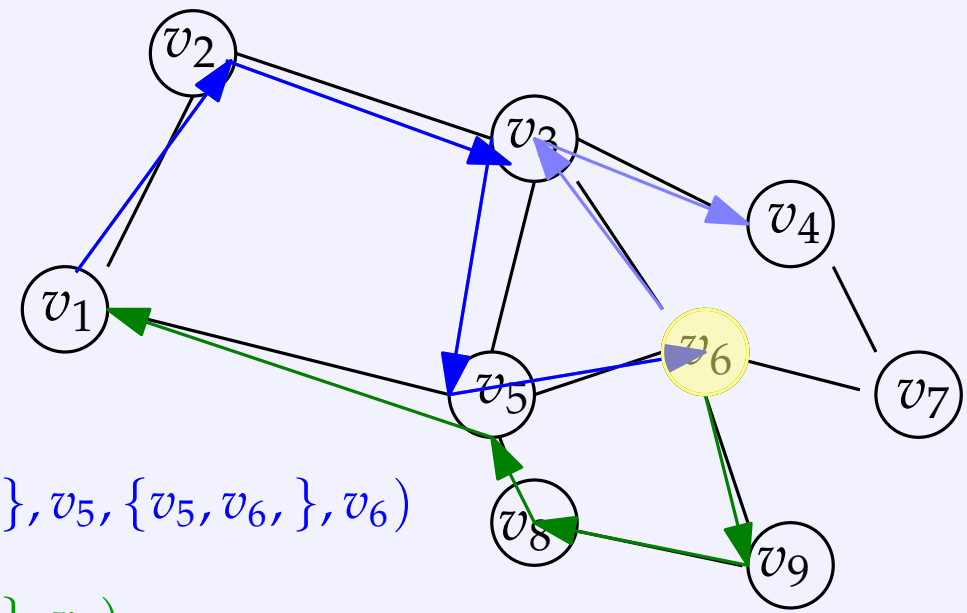
$K_1 = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5, \{v_5, v_6, \}, v_6)$
$K' = (v_6, \{v_6, v_3\}, v_3)$
$K_2 = (v_6, \{v_6, v_9\}, v_9, \{v_9, v_8\}, v_8, \{v_8, v_5\}, v_5)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



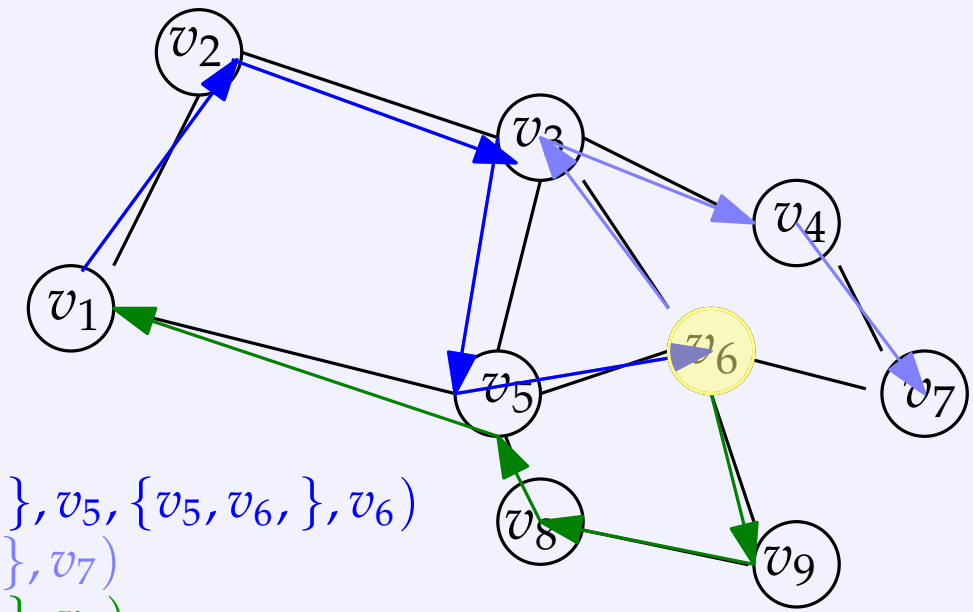
$K_1 = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5, \{v_5, v_6, \}, v_6)$
$K' = (v_6, \{v_6, v_3\}, v_3, \{v_3, v_4\}, v_4)$
$K_2 = (v_6, \{v_6, v_9\}, v_9, \{v_9, v_8\}, v_8, \{v_8, v_5\}, v_5)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze
    $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



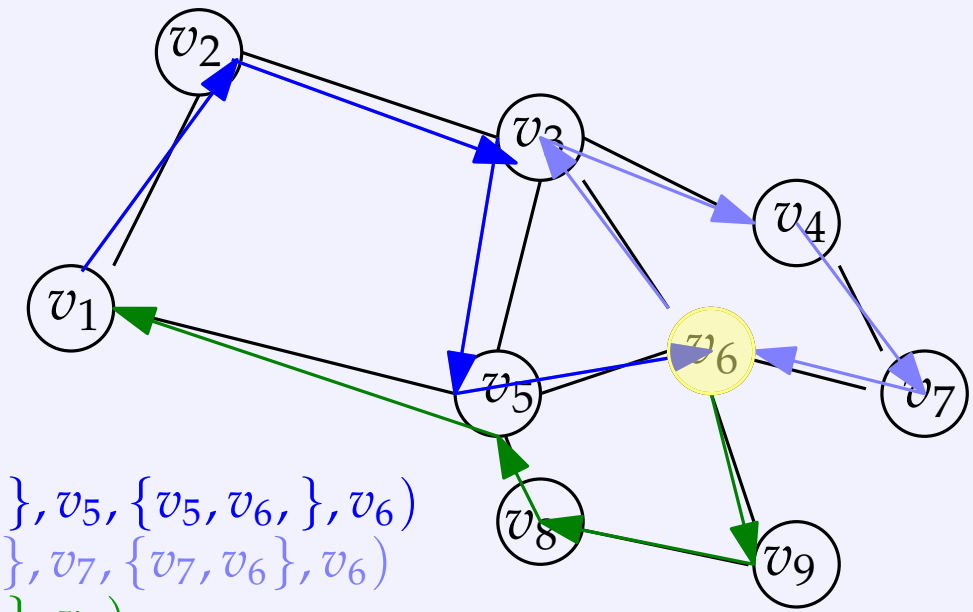
$K_1 = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5, \{v_5, v_6, \}, v_6)$
$K' = (v_6, \{v_6, v_3\}, v_3, \{v_3, v_4\}, v_4, \{v_4, v_7\}, v_7)$
$K_2 = (v_6, \{v_6, v_9\}, v_9, \{v_9, v_8\}, v_8, \{v_8, v_5\}, v_5)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



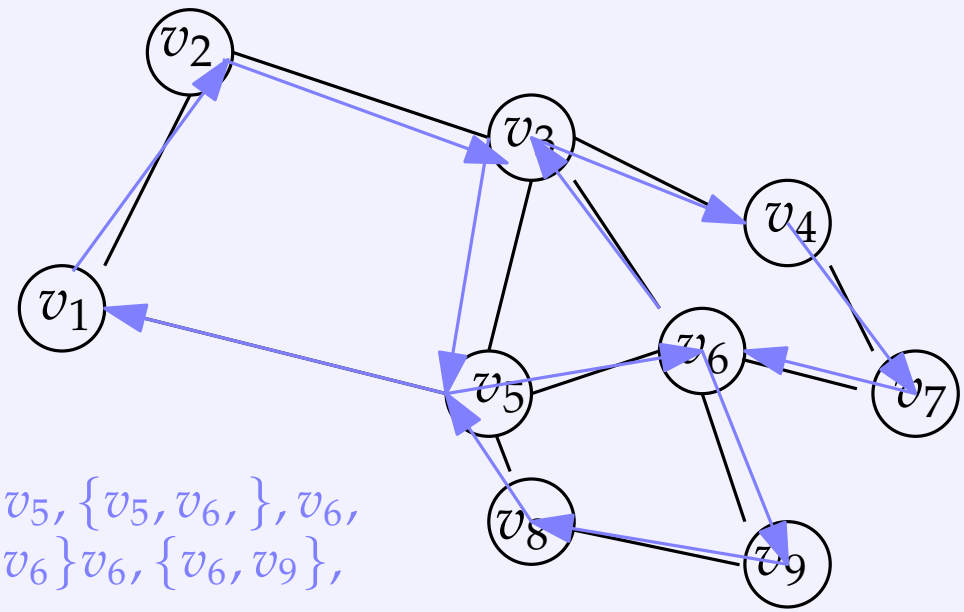
$K_1 = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \{v_3, v_5\}, v_5, \{v_5, v_6, \}, v_6)$
$K' = (v_6, \{v_6, v_3\}, v_3, \{v_3, v_4\}, v_4, \{v_4, v_7\}, v_7, \{v_7, v_6\}, v_6)$
$K_2 = (v_6, \{v_6, v_9\}, v_9, \{v_9, v_8\}, v_8, \{v_8, v_5\}, v_5)$

Satz von Euler - Algorithmus von Hierholzer

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$ mit $|E(v)|$ gerade für alle $v \in V$

Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K



$K = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3 \{v_3, v_5\}, v_5, \{v_5, v_6, \}, v_6, \{v_6, v_3\}, v_3, \{v_3, v_4\}, v_4, \{v_4, v_7\}, v_7, \{v_7, v_6\} v_6, \{v_6, v_9\}, v_9, \{v_9, v_8\}, v_8, \{v_8, v_5\}, v_5)$

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K

Zeige per Induktion: Nach dem i-ten Ausführen der äußeren While-Schleife ist $K_i := K$ ein Kreis, der keine Kante doppelt enthält.

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K

Zeige per Induktion: Nach dem i-ten Ausführen der äußeren While-Schleife ist $K_i := K$ ein Kreis, der keine Kante doppelt enthält.
I.A.: Vor dem ersten Ausführen der äußeren While-Schleife ist $K_0 = (v)$. ✓

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

Zeige per Induktion: Nach dem i-ten Ausführen der äußeren While-Schleife ist $K_i := K$ ein Kreis, der keine Kante doppelt enthält.
I.B.:Angenomen K_{i-1} ist ein Kreis, der keine Kante doppelt enthält.

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

Zeige per Induktion: Nach dem i-ten Ausführen der äußeren While-Schleife ist $K_i := K$ ein Kreis, der keine Kante doppelt enthält.
I.B.:Angenomen K_{i-1} ist ein Kreis, der keine Kante doppelt enthält.
Dann ist für alle $v \in G$ $|E_{K_i^1}(v)| + |E_{K_i^2}(v)| = |E_{K_{i-1}}(v)|$ gerade.

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

Zeige per Induktion: Nach dem i-ten Ausführen der äußeren While-Schleife ist $K_i := K$ ein Kreis, der keine Kante doppelt enthält.
I.B.:K_{i-1} Kreis ohne doppelte Kante $\Rightarrow \forall v \in G \; |E_{K_i^1}(v)| + |E_{K_i^2}(v)|$ gerade.

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

Zeige per Induktion: Nach dem i-ten Ausführen der äußeren While-Schleife ist $K_i := K$ ein Kreis, der keine Kante doppelt enthält.
I.B.:K_{i-1} Kreis ohne doppelte Kante $\Rightarrow \forall v \in G \; |E_{K_i^1}(v)| + |E_{K_i^2}(v)|$ gerade.

I.S.: Nach Konstruktion enthält Pfad $K_i' = (w^1, \ldots, w^k)$ keine Kante doppelt und keine, die schon in $E_{K_i^1}$ oder $E_{K_i^2}$ ist.

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

Zeige per Induktion: Nach dem i-ten Ausführen der äußeren While-Schleife ist $K_i := K$ ein Kreis, der keine Kante doppelt enthält.
I.B.:K_{i-1} Kreis ohne doppelte Kante $\Rightarrow \forall v \in G \; |E_{K_i^1}(v)| + |E_{K_i^2}(v)|$ gerade.

I.S.: Pfad $K_i' = (w^1, \ldots, w^k)$ enthält keine Kante doppelt und keine aus $E_{K_i^1} \cup E_{K_i^2}$.✓

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K

Zeige per Induktion: Nach dem i-ten Ausführen der äußeren While-Schleife ist $K_i := K$ ein Kreis, der keine Kante doppelt enthält.
I.B.:K_{i-1} Kreis ohne doppelte Kante $\Rightarrow \forall v \in G \; |E_{K_i^1}(v)| + |E_{K_i^2}(v)|$ gerade.

I.S.: Angenomen Pfad $K_i' = (w^1, \ldots, w^k)$ ist kein Kreis.

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

Zeige per Induktion: Nach dem i-ten Ausführen der äußeren While-Schleife ist $K_i := K$ ein Kreis, der keine Kante doppelt enthält.
I.B.:K_{i-1} Kreis ohne doppelte Kante $\Rightarrow \forall v \in G \; |E_{K_i^1}(v)| + |E_{K_i^2}(v)|$ gerade.

I.S.: Angenomen Pfad $K_i' = (w^1, \ldots, w^k)$ ist kein Kreis.$\Rightarrow |E_{K_i'}(w^k)|$ ungerade .

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

I.B.:K_{i-1} Kreis ohne doppelte Kante $\Rightarrow \forall v \in G \; |E_{K_i^1}(v)| + |E_{K_i^2}(v)|$ gerade.

I.S.: Angenomen Pfad $K_i' = (w^1, \ldots, w^k)$ kein Kreis $\Rightarrow |E_{K_i'}(w^k)|$ ungerade .

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

I.B.:K_{i-1} Kreis ohne doppelte Kante $\Rightarrow \forall v \in G \; |E_{K_i^1}(v)| + |E_{K_i^2}(v)|$ gerade.

I.S.: Angenomen Pfad $K_i' = (w^1, \ldots, w^k)$ kein Kreis $\Rightarrow |E_{K_i'}(w^k)|$ ungerade .

Die Anzahl der benutzten, zu w^k inzidenten Kanten
$|E_{K_i'}(w^k)| + |E_{K_i^1}(w^k)| + |E_{K_i^2}(w^k)|$ ist also auch ungerade.

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

I.B.:K_{i-1} Kreis, für alle $v \in G$ $|E_{K_i^1}(v)| + |E_{K_i^2}(v)| = |E_{K_{i-1}}(v)|$ gerade.

I.S.: Angenomen Pfad $K_i' = (w^1, \ldots, w^k)$ kein Kreis $\Rightarrow \ldots$
$\Rightarrow |E_{K_i'}(w^k)| + |E_{K_i^1}(w^k)| + |E_{K_i^2}(w^k)|$ ungerade.

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G

Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
while noch nicht alle Kanten benutzt **do**
  Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
  Teile Kreis K in v in zwei Wege K^1 und K^2
  Setze $K' = (v)$
  while v hat unbenutzte inzidente Kanten **do**
    Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
    Setze $v = w$
  end while
  $K = K_1 \circ K' \circ K_2$
end while
return K

I.B.:K_{i-1} Kreis, für alle $v \in G$ $|E_{K_i^1}(v)| + |E_{K_i^2}(v)| = |E_{K_{i-1}}(v)|$ gerade.

I.S.: Angenomen Pfad $K_i' = (w^1, \ldots, w^k)$ kein Kreis $\Rightarrow \ldots$
$\Rightarrow |E_{K_i'}(w^k)| + |E_{K_i^1}(w^k)| + |E_{K_i^2}(w^k)|$ ungerade.

$\Rightarrow$ Die Anzahl der unbenutzten, zu w^k inzidenten Kanten
$|E(w^k)| - (|E_{K_i'}(w^k)| + |E_{K_i^1}(w^k)| + |E_{K_i^2}(w^k)|)$ ist ungerade (da $|E(w^k)|$ gerade).

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

I.B.:K_{i-1} Kreis, für alle $v \in G$ $|E_{K_i^1}(v)| + |E_{K_i^2}(v)| = |E_{K_{i-1}}(v)|$ gerade.

I.S.: Angenomen Pfad $K_i' = (w^1, \ldots, w^k)$ kein Kreis $\Rightarrow \ldots$

$\Rightarrow$ Die Anzahl der unbenutzten, zu w^k inzidenten Kanten ist ungerade.

$\Rightarrow$ Die Anzahl der unbenutzten, zu w^k inzidenten Kanten ist nicht 0.

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

I.B.:K_{i-1} Kreis, für alle $v \in G$ $|E_{K_i^1}(v)| + |E_{K_i^2}(v)| = |E_{K_{i-1}}(v)|$ gerade.

I.S.: Angenomen Pfad $K_i' = (w^1, \ldots, w^k)$ kein Kreis $\Rightarrow \ldots$

$\Rightarrow$ Die Anzahl der unbenutzten, zu w^k inzidenten Kanten ist ungerade.

$\Rightarrow$ Die Anzahl der unbenutzten, zu w^k inzidenten Kanten ist nicht 0.

Widerspruch zu 'w^k hat keine unbenutzen inzidenten Kanten' (Abbruchbedingung der inneren While-Scheife).

Satz von Euler - Beweis Korrektheit Algorithmus

"$\Leftarrow$"(Rückrichtung):
Sei G ein (Multi-)-Graph, in dem alle Knoten geraden Knotengrad haben.

Behauptung: Algorithmus von Hierholzer findet Eulerkreis in G:

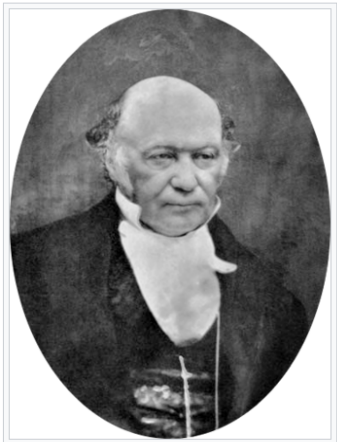
Input: zusammenhängender, ungerichteter (Multi-)Graph $G = (V, E)$
Output: Eulerkreis K in G
  Wähle beliebigen Knoten $v \in V$. Setze $K = (v)$.
  while noch nicht alle Kanten benutzt **do**
    Wähle Knoten v in K, der noch unbenutzte inzidente Kanten hat
    Teile Kreis K in v in zwei Wege K^1 und K^2
    Setze $K' = (v)$
    while v hat unbenutzte inzidente Kanten **do**
      Wähle unbenutzte Kante $\{v, w\} \in E(v)$ und setze $K' = K' \circ (\{v, w\}, w)$.
      Setze $v = w$
    end while
    $K = K_1 \circ K' \circ K_2$
  end while
  return K

Laufzeit?

Wo benutzen wir Tourenplanung?

Hamiltonsche Kreise

Ein **Hamiltonscher Kreis** in einem Graph ist ein *Kreis*, der jeden Knoten genau einmal besucht.
Ein Graph, der einen Hamiltonschen Kreis enthält, heißt **Hamiltonsch**.
Ein **Hamilonscher Weg/Pfad** ist ein *Weg*, der jeden Knoten genau einmal besucht.

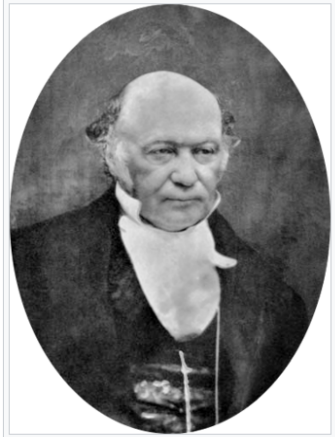


William Rowan Hamilton

*4. August 1805 in Dublin;
† 2. September 1865 in Dunsink bei Dublin

Hamiltonsche Kreise

Ein **Hamiltonscher Kreis** in einem Graph ist ein *Kreis*, der jeden Knoten genau einmal besucht.
Ein Graph, der einen Hamiltonschen Kreis enthält, heißt **Hamiltonsch**.
Ein **Hamilonscher Weg/Pfad** ist ein *Weg*, der jeden Knoten genau einmal besucht.

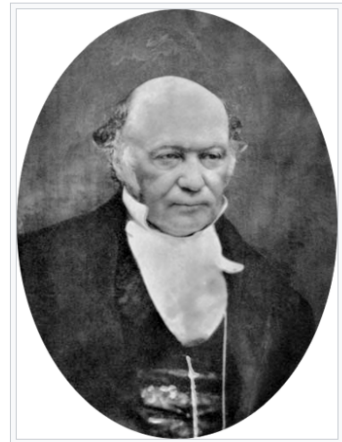

William Rowan Hamilton

*4. August 1805 in Dublin;
† 2. September 1865 in Dunsink bei Dublin

Bemerkung: Das impliziert, das jede Kante höchstens einmal benutzt wird (sonst würden Start- und Endknoten ja mehrmals im Kreis enthalten sein).

Hamiltonsche Kreise

Ein **Hamiltonscher Kreis** in einem Graph ist ein *Kreis*, der jeden Knoten genau einmal besucht.
Ein Graph, der einen Hamiltonschen Kreis enthält, heißt **Hamiltonsch**.
Ein **Hamilonscher Weg/Pfad** ist ein *Weg*, der jeden Knoten genau einmal besucht.


William Rowan Hamilton

*4. August 1805 in Dublin;
† 2. September 1865 in Dunsink bei Dublin

Bemerkung: Das impliziert, das jede Kante höchstens einmal benutzt wird (sonst würden Start- und Endknoten ja mehrmals im Kreis enthalten sein).

Gibt es in jedem Graph einen Hamiltonschen Kreis?

Gibt es in jedem Graph einen Hamiltonschen Weg?

Ist der Hamiltonsche Kreis/Weg (wenn es einen gibt) eindeutig?

Woran erkenne ich, dass es in einem Graph einen Hamiltonschen Kreis/Weg gibt?

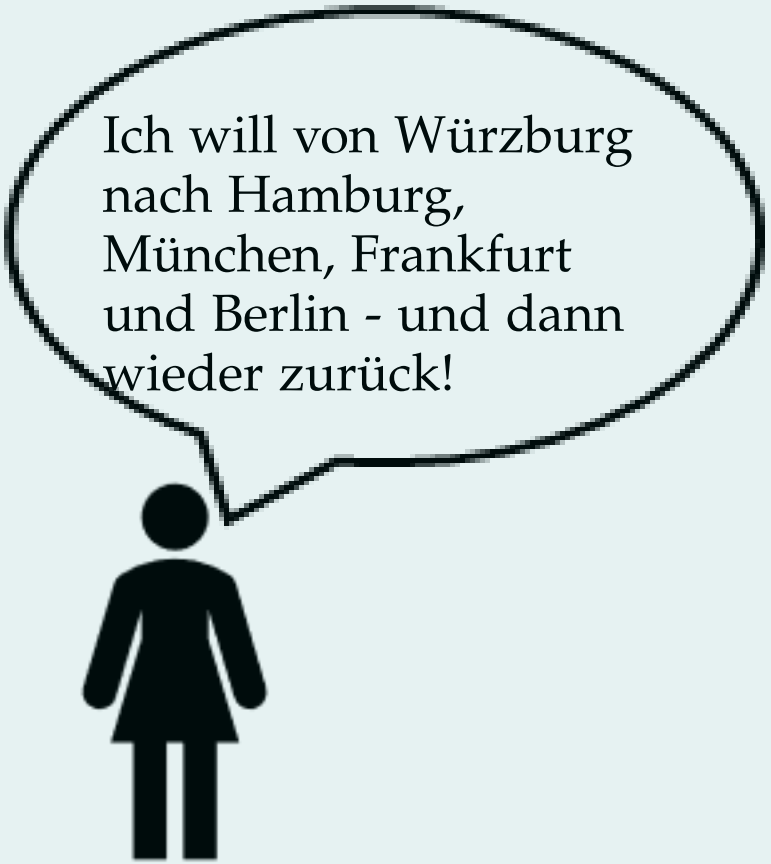
Wie kann ich einen Hamiltonschen Kreis/Weg bestimmen?

Das Handlungsreisendenproblem

(*Eine* Variante des) **Handlungsreisendenproblem** (Englisch: **Traveling Salesperson Problem**, kurz **TSP**):
Gegeben: Eine Menge von Orten O und paarweise Entfernungen zwischen den Orten
Gesucht: Die kürzeste Rundtour, die jeden Ort genau einmal besucht.

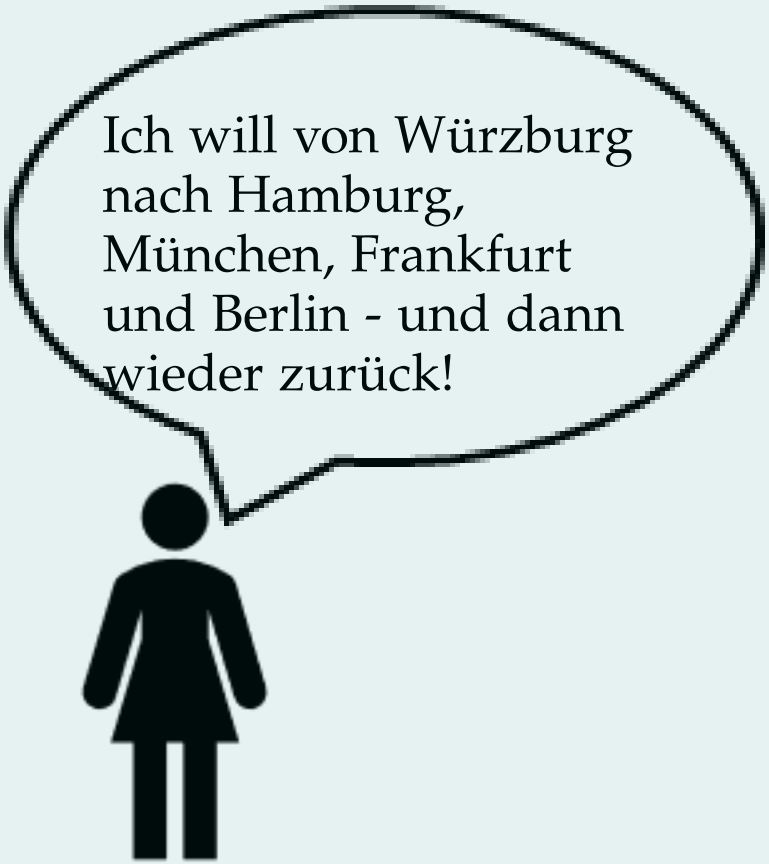
Das Handlungsreisendenproblem

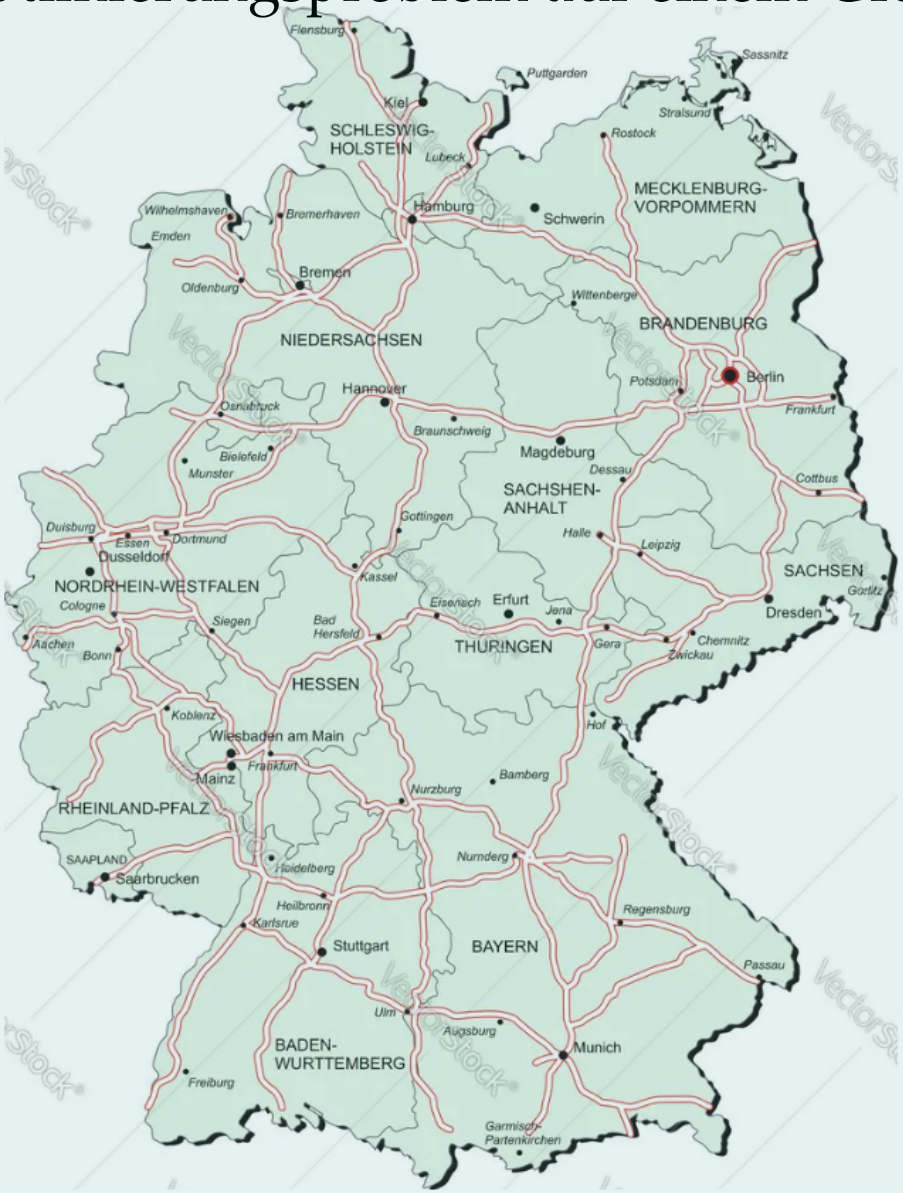
TSP als Optimierungsproblem auf einem Graph:



Das Handlungsreisendenproblem

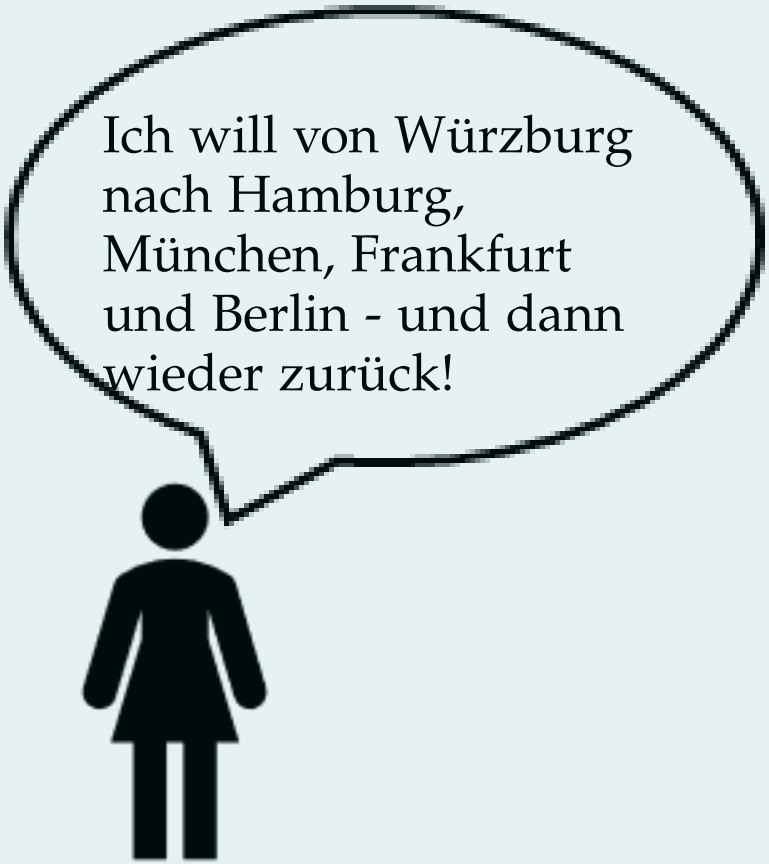
TSP als Optimierungsproblem auf einem Graph:

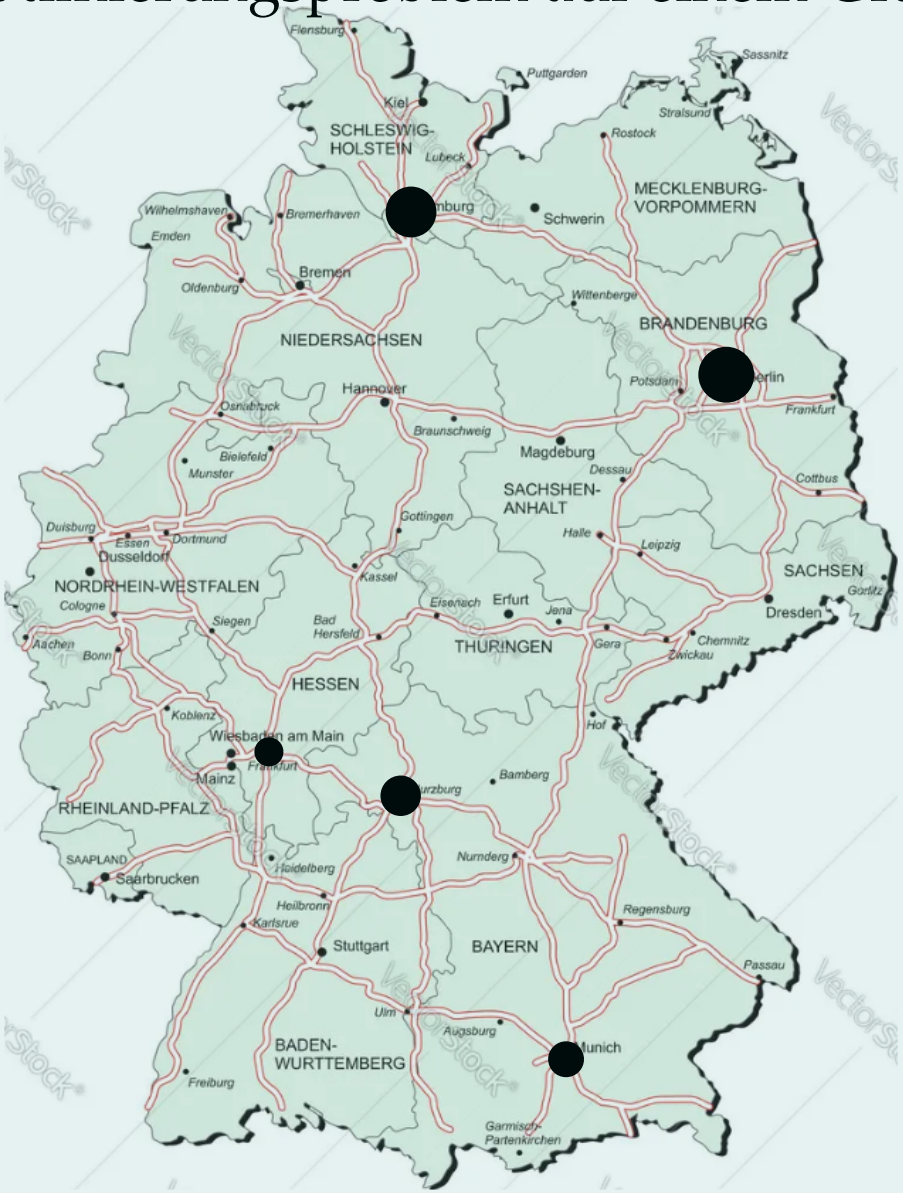




Das Handlungsreisendenproblem

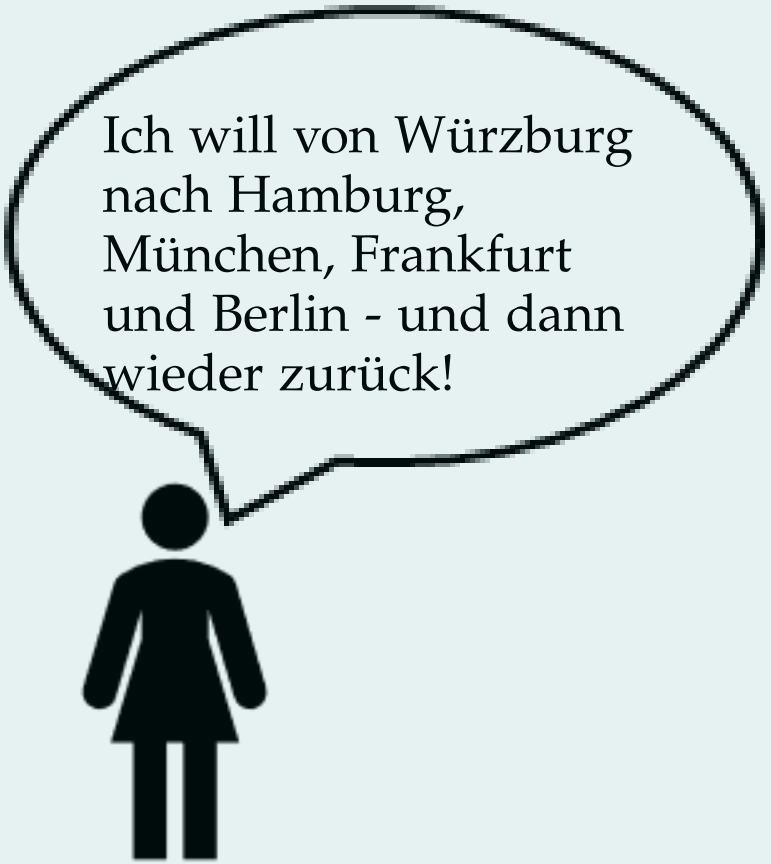
TSP als Optimierungsproblem auf einem Graph:

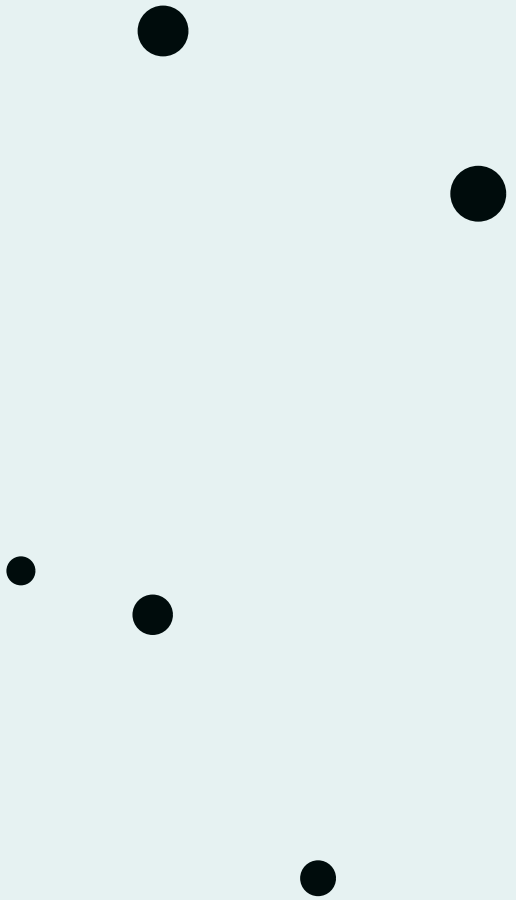




Das Handlungsreisendenproblem

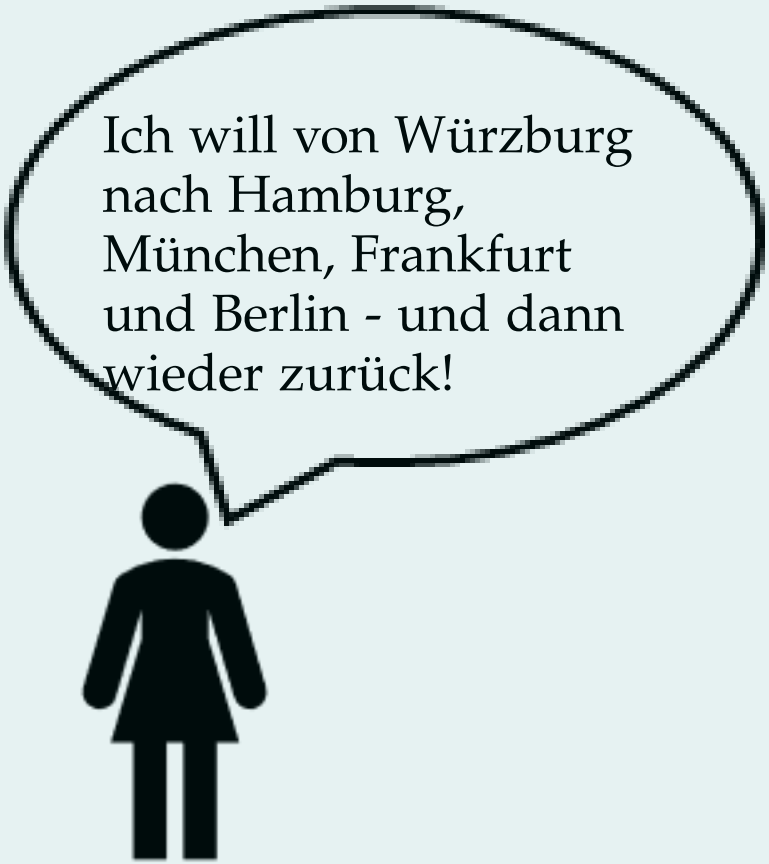
TSP als Optimierungsproblem auf einem Graph:

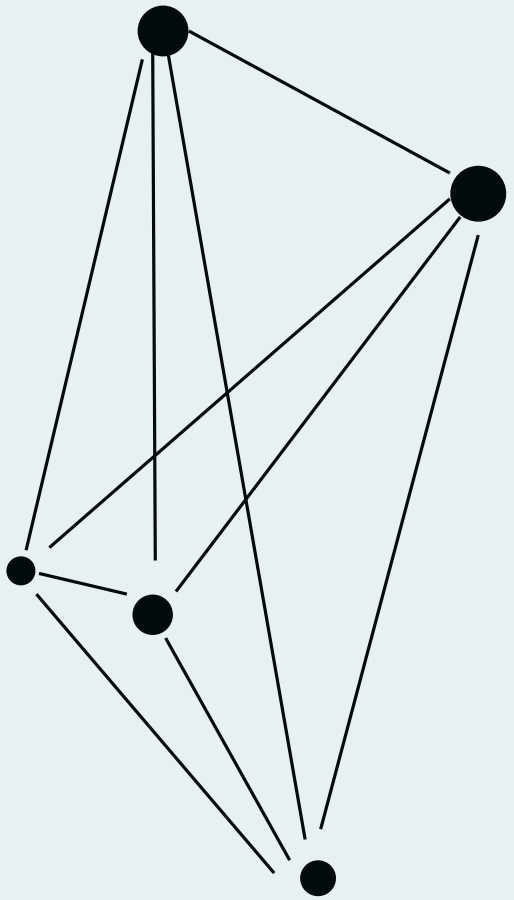




Das Handlungsreisendenproblem

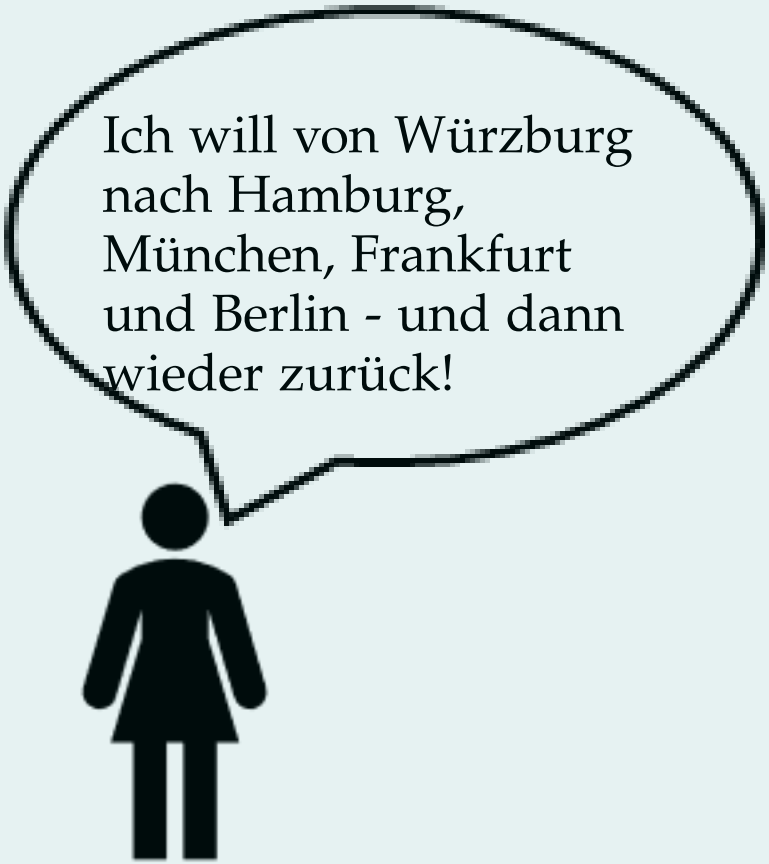
TSP als Optimierungsproblem auf einem Graph:

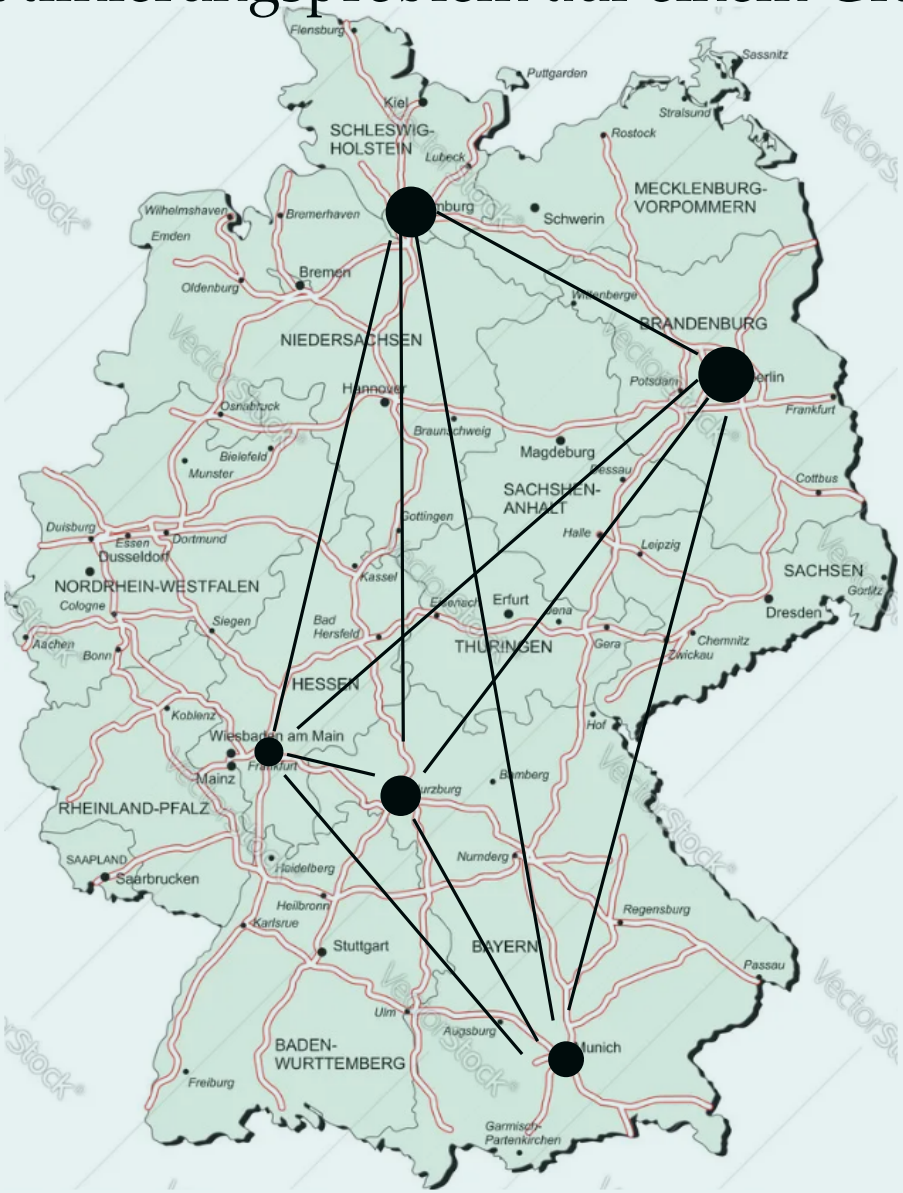




Das Handlungsreisendenproblem

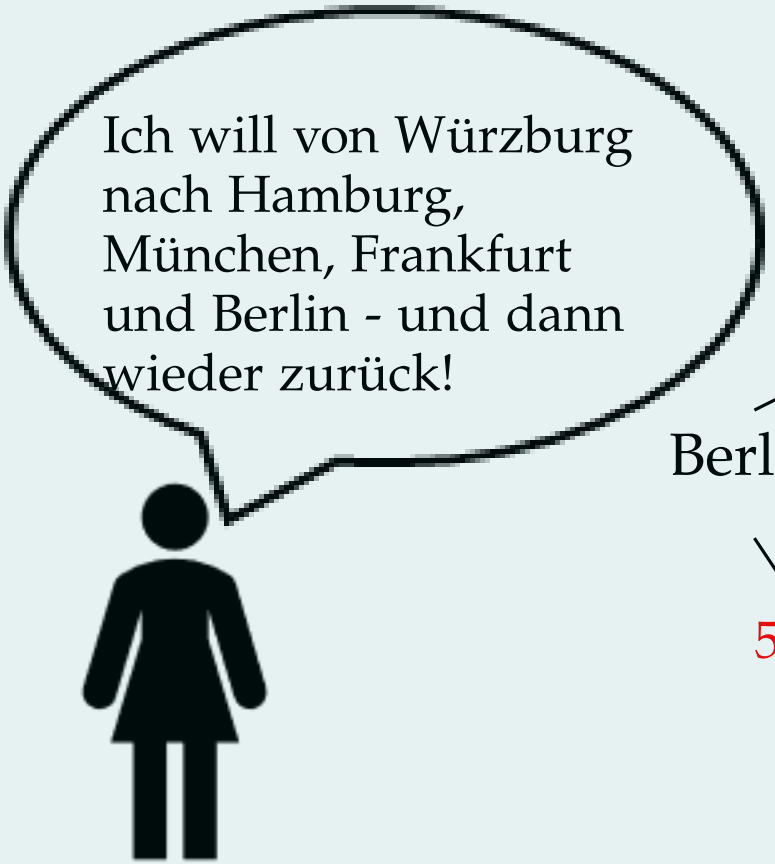
TSP als Optimierungsproblem auf einem Graph:

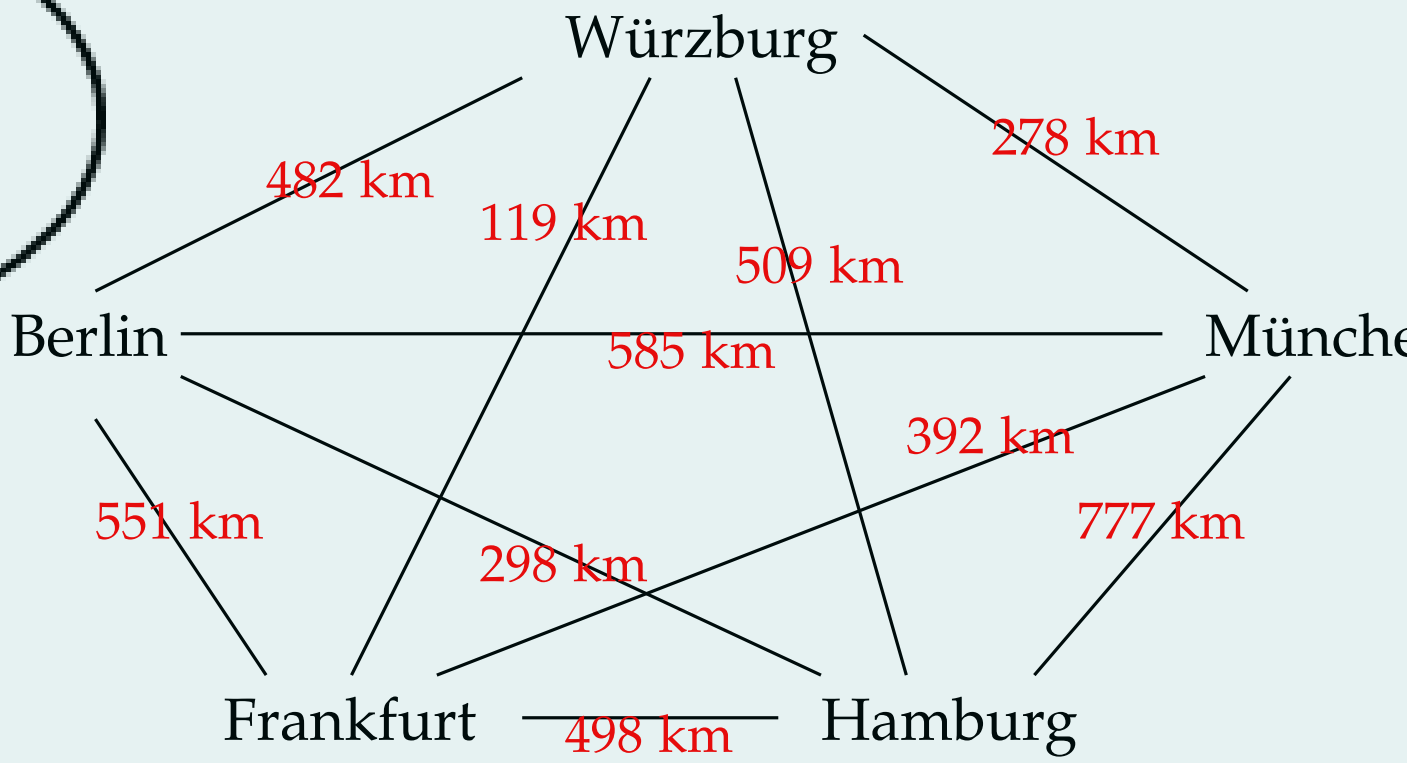




Das Handlungsreisendenproblem

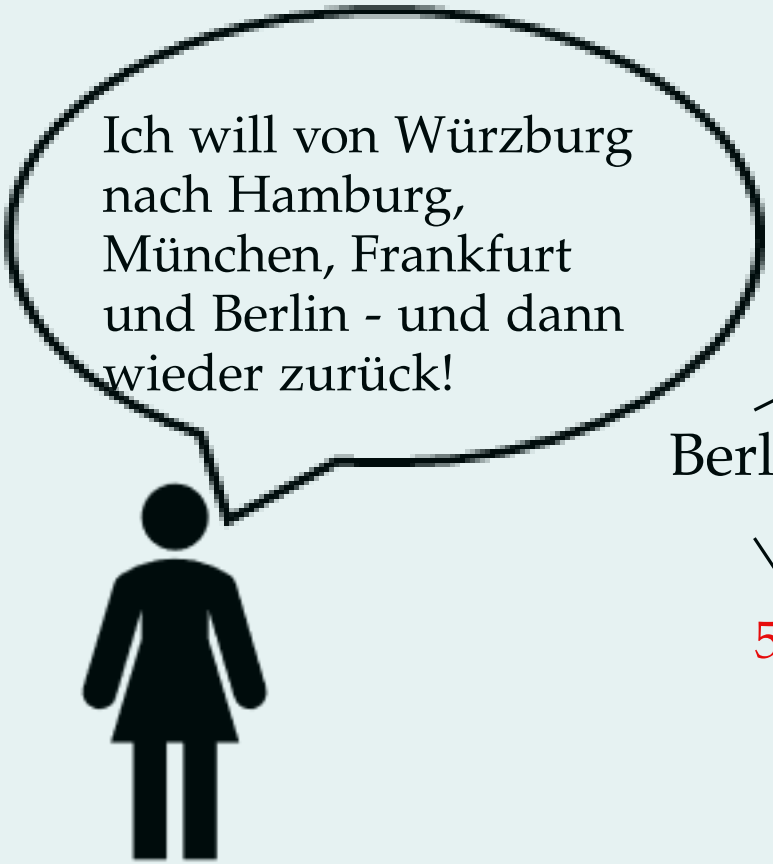
TSP als Optimierungsproblem auf einem Graph:

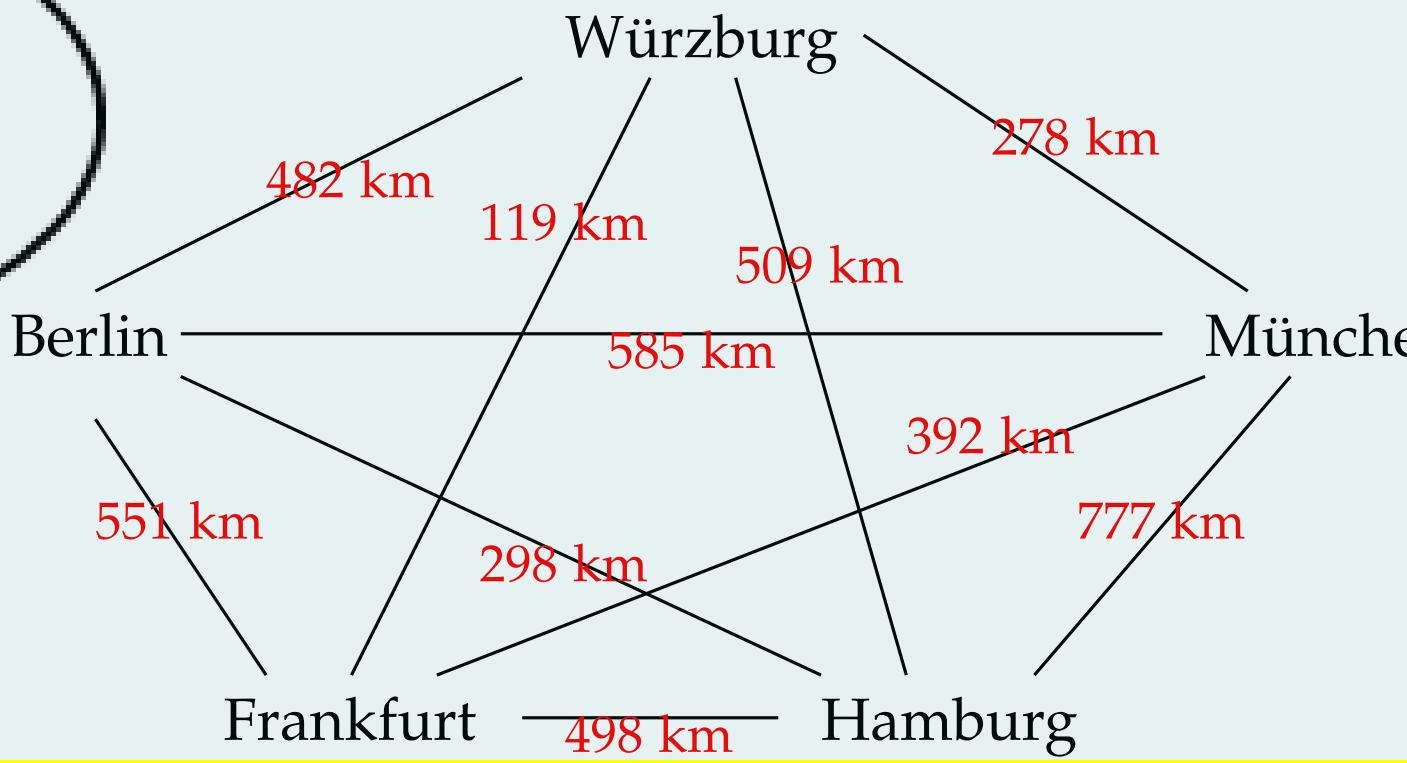




Das Handlungsreisendenproblem

TSP als Optimierungsproblem auf einem Graph:

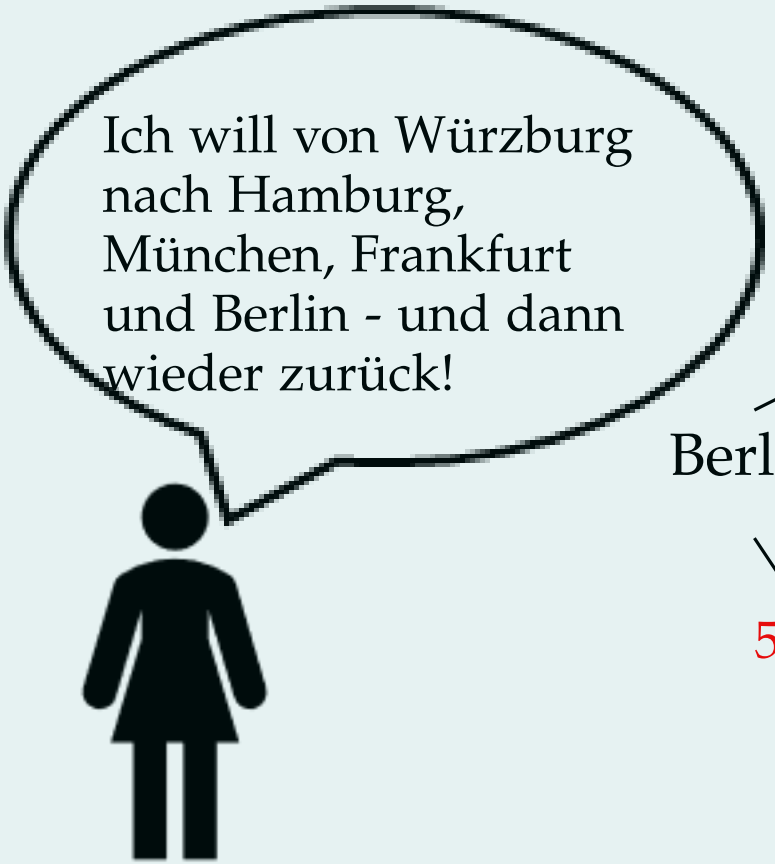


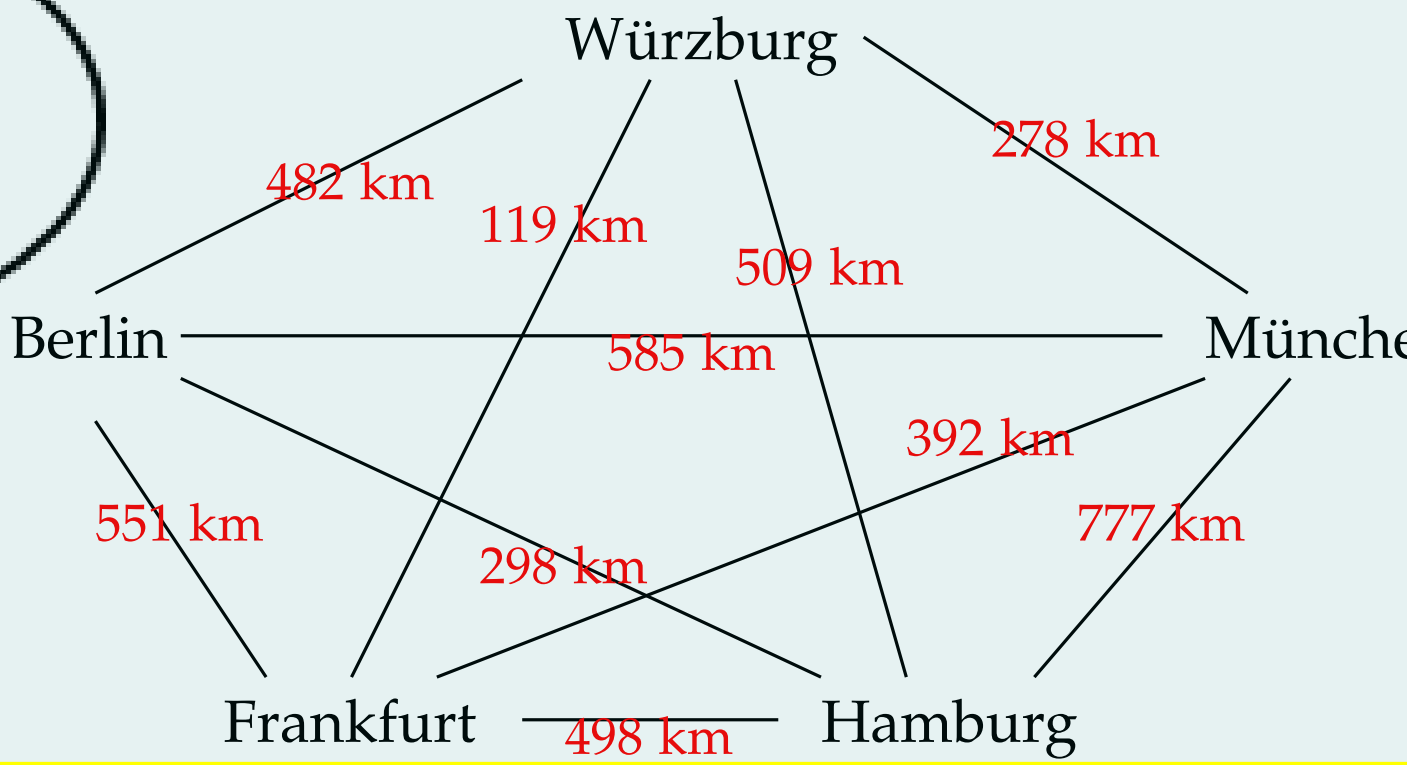


TSP auf Graph (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Das Handlungsreisendenproblem

TSP als Optimierungsproblem auf einem Graph:



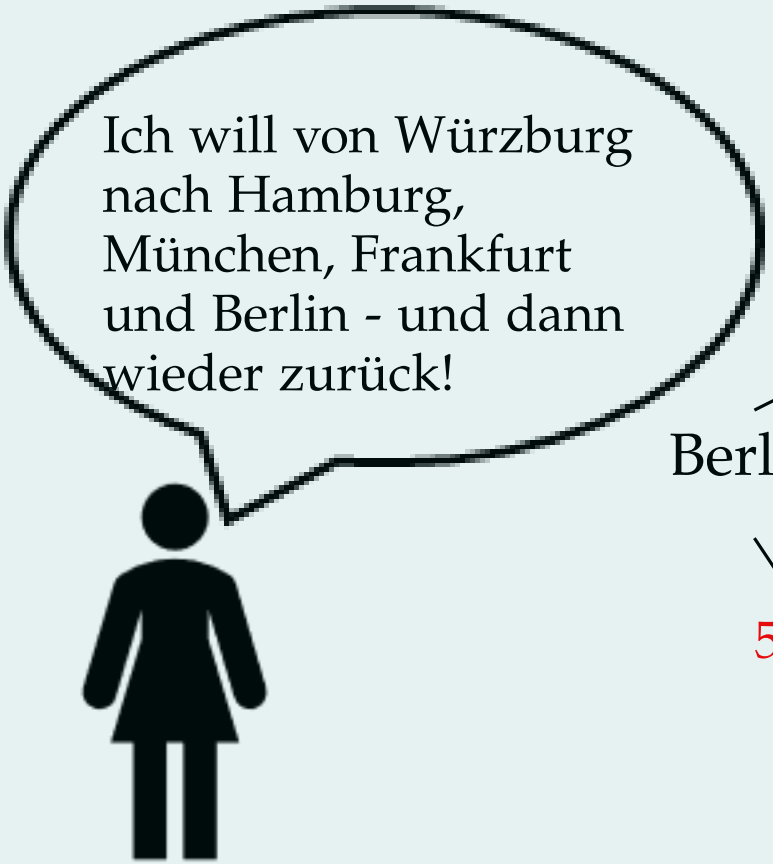


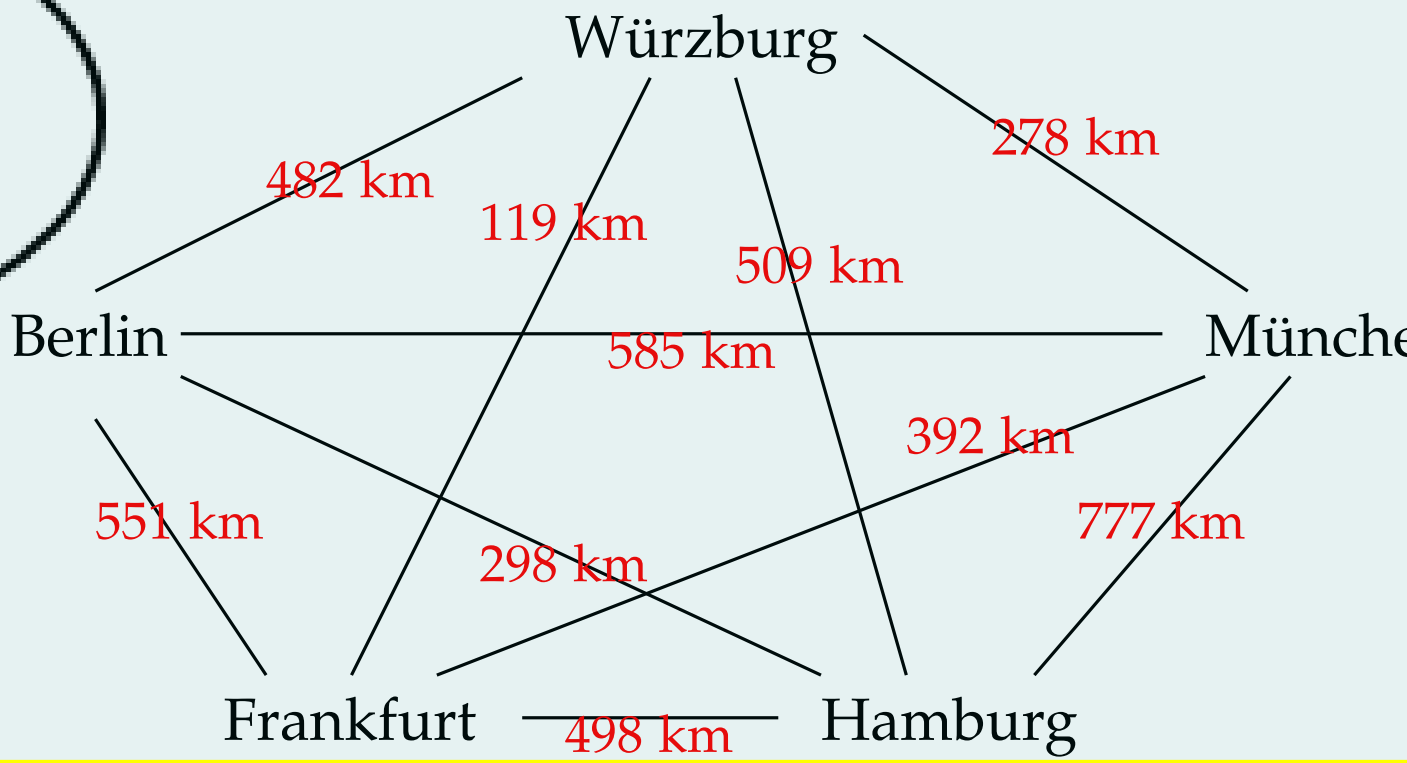
TSP auf Graph (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Ein ungerichteter Graph $G = (V, E)$ heißt **vollständig**, wenn für alle $v, w \in V$ mit $v \neq w$ gilt $\{v, w\} \in E$.

Das Handlungsreisendenproblem

TSP als Optimierungsproblem auf einem Graph:

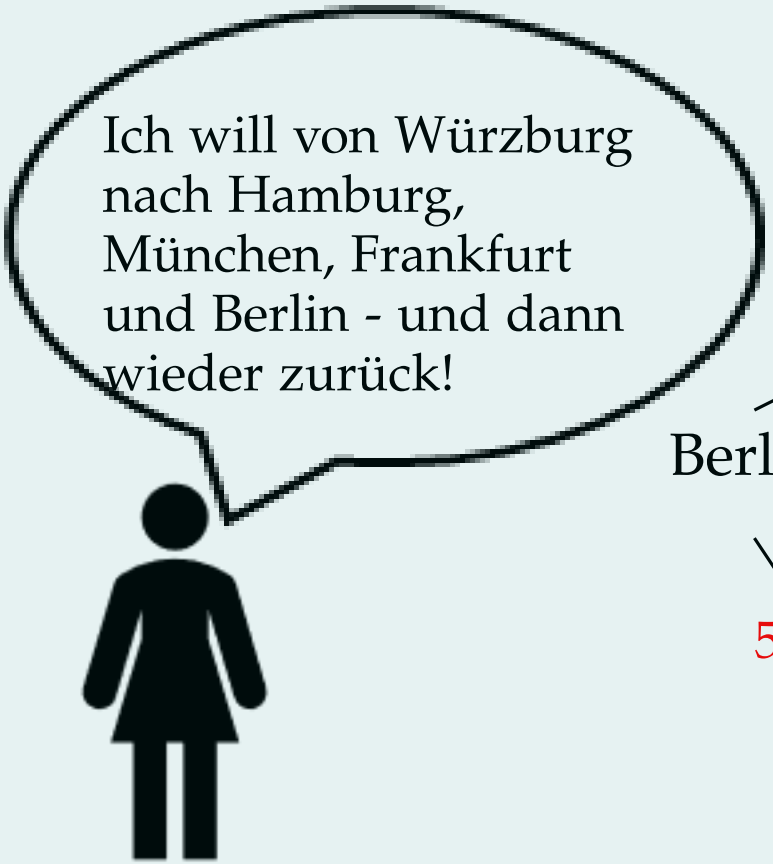


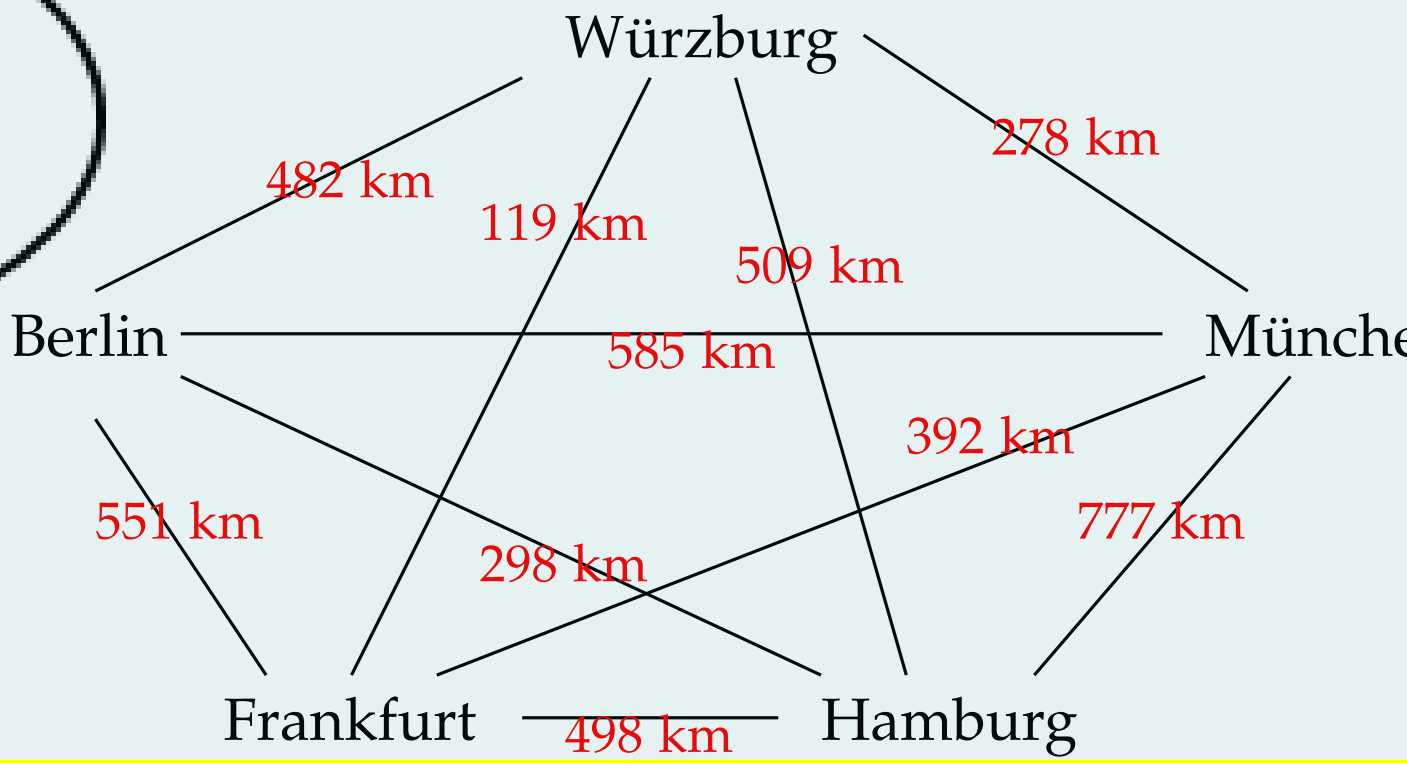


TSP auf Graph (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Das Handlungsreisendenproblem

TSP als Optimierungsproblem auf einem Graph:



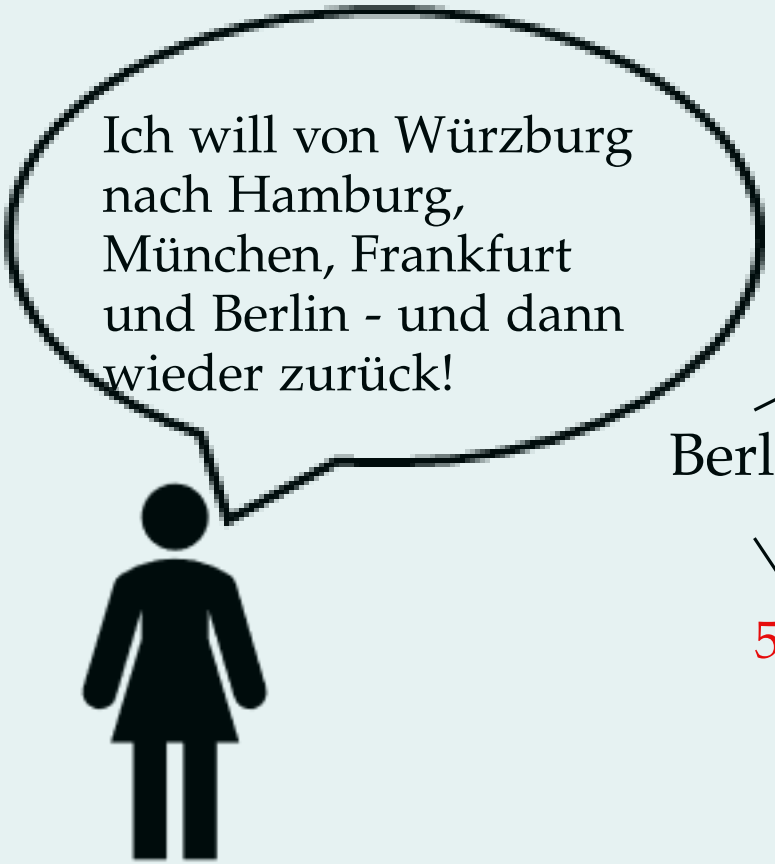


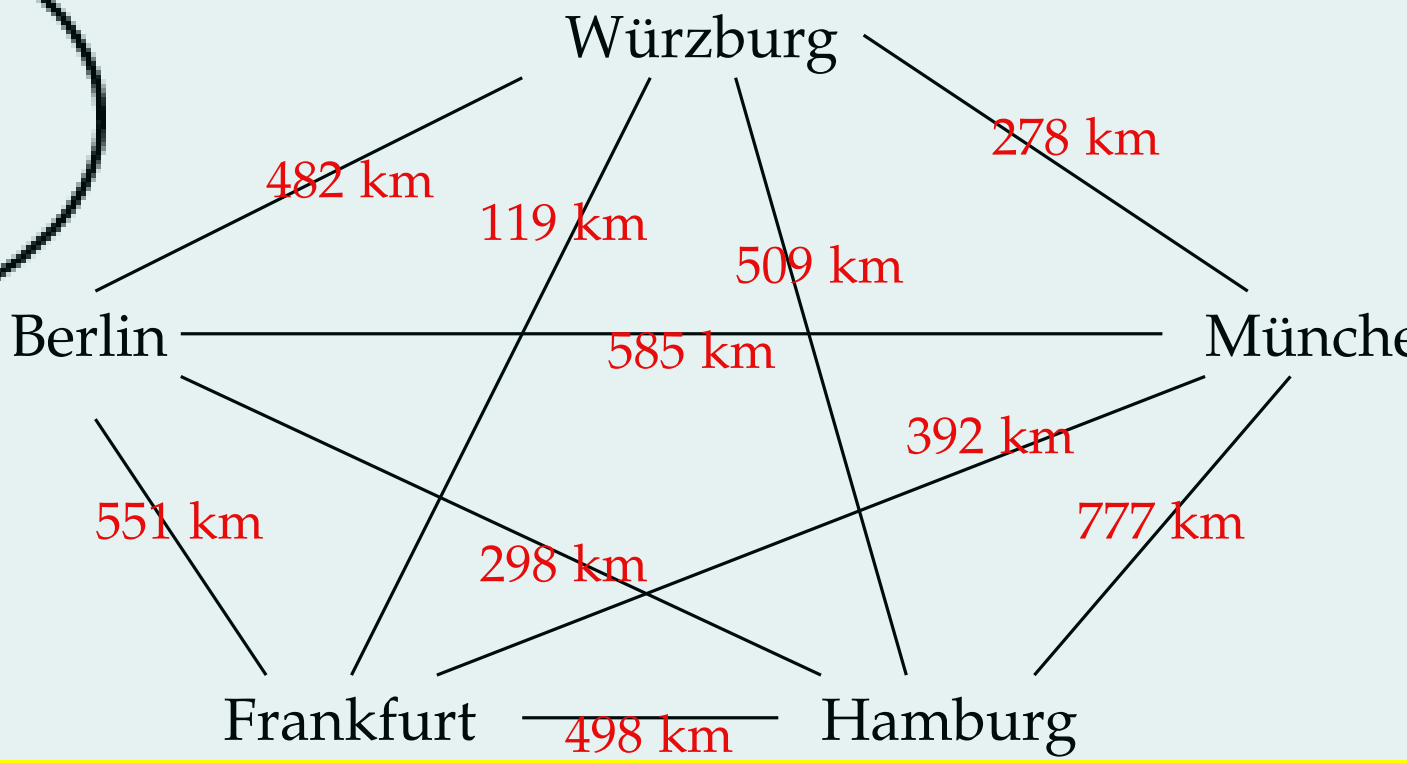
TSP auf Graph (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Algorithmus um optimale TSP-Tour zu finden?

Das Handlungsreisendenproblem

TSP als Optimierungsproblem auf einem Graph:





TSP auf Graph (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Zusammenhang Hamiltonscher Kreis?

Stichworte heute

Konzepte (auf Graphen): Pareto/nichtdominierte Wege, Multigraph, zusammenhängend, vollständig, Eulerkreis, Eulerweg, eulerscher Graph, Satz von Euler, Hamiltonscher Kreis/Weg/Graph

Optimierungsprobleme: *multikriterielle Optimierung*, *Dominanz*, *nichtdominierte/Pareto Lösungen*, multikriterielle kürzeste Wege, Handlungsreisendenproblem (TSP)

Modellierung: Routenplanung, Königsberger Brückenproblem, Handlungsreisendenproblem, . . . , *Graphen stellen räumliche Zusammenhänge dar*

Algorithmen: multikriterielles Label-Setting, Finden eines Eulerkreises, Cheapest Insertion^{Uebungsblatt}, 2-OPT^{Uebungsblatt}