Algorithms and Probability

Week 2

G08 - mkilic

27.II.2025

Overview

1. Minitest
2. Ein neues DFS - (Woche 1)
3. Kreise
4. Hamilton - DP
5. Inklusion-Exklusion-Prinzip
6. Travelling Salesman Problem
7. Exercises

Roadmap

1. Graphentheorie
   - Zusammenhang
   - Kreise
   - Matchings
   - Färbungen
2. W'keitstheorie
   - Bedingte W'keit
   - Unabhängigkeit
   - (mehrere) Zufallsvariablen
   - Diskrete Verteilungen
   - Abschätzen von W'keiten
   - Randomisierte Algorithmen
3. Algorithmen
   - Lange-Bunte Pfade
   - MaxFlow
   - MinCut
   - Kleinster umschliessender Kreis
   - Konvexe Hülle

Minitest

Passwort: graph

Minitest

Besprechung von Minitest

Kreise

Wir sind interessiert in zwei bestimmten Kreisarten:

- **Eulertour:** Ein geschlossener Weg in $G = (V, E)$, der jede *Kante* genau einmal enthält.

[0] *Engl. Eulerian Circuit,*

Kreise

Wir sind interessiert in zwei bestimmten Kreisarten:

- **Eulertour:** Ein geschlossener Weg in $G = (V, E)$, der jede *Kante* genau einmal enthält.
- **Hamiltonkreis:** Ein Kreis in $G = (V, E)$, der jeden *Knoten* genau einmal enthält.

[0] *Engl. Eulerian Circuit, Hamiltonian Cycle*

Eulertour - Bedingung

Eulertour Bedingung

$G = (V, E)$ enthält Eulertour $\iff$ für alle $v \in V$: $deg(v)$ ist gerade und G zusammenhängend

Eulertour - Bedingung

Eulertour Bedingung

$G = (V, E)$ enthält Eulertour $\iff$ für alle $v \in V$: $deg(v)$ ist gerade und G zusammenhängend

Intuition: So viele Kanten geht hinein wie heraus.

Eulertour - Bedingung

Eulertour Bedingung

$G = (V, E)$ enthält Eulertour $\iff$ für alle $v \in V$: $deg(v)$ ist gerade und G zusammenhängend

Intuition: So viele Kanten geht hinein wie heraus.
Beweis: Algorithmus, der in den zusammenhängenden Graphen, in dem alle Knoten gereade Grad haben, einen Eulertour findet

Eulertour - Algorithmus

1. Wir haben zwei Läufer, Langsamer (L) und Schneller (S). Beide fangen von dem gleichen Startknoten s an. Zunächst beobachte S.

0

Eulertour - Algorithmus

1. Wir haben zwei Läufer, Langsamer (L) und Schneller (S). Beide fangen von dem gleichen Startknoten s an. Zunächst beobachte S.
2. S geht durch den Knoten zufällig. Wann S durch eine Kante läuft, löscht man sie.

[0] *Beweis zur 2 auf übernächster Seite*

Eulertour - Algorithmus

1. Wir haben zwei Läufer, Langsamer (L) und Schneller (S). Beide fangen von dem gleichen Startknoten s an. Zunächst beobachte S.
2. S geht durch den Knoten zufällig. Wann S durch eine Kante läuft, löscht man sie.
3. In dieser Art und Weise hört S diesen `RandomTour` auf, genau an dem Knoten, wo er angefangen hat.

[0] *Beweis zur 2 auf übernächster Seite*

Eulertour - Algorithmus

1. Wir haben zwei Läufer, Langsamer (L) und Schneller (S). Beide fangen von dem gleichen Startknoten s an. Zunächst beobachte S.
2. S geht durch den Knoten zufällig. Wann S durch eine Kante läuft, löscht man sie.
3. In dieser Art und Weise hört S diesen `RandomTour` auf, genau an dem Knoten, wo er angefangen hat.
4. Man hat schon einen Zyklus W gefunden. Falls W ein Eulertour ist sind wir fertig.

[0] *Beweis zur 2 auf übernächster Seite*

Eulertour - Algorithmus

1. Wir haben zwei Läufer, Langsamer (L) und Schneller (S). Beide fangen von dem gleichen Startknoten s an. Zunächst beobachte S.
2. S geht durch den Knoten zufällig. Wann S durch eine Kante läuft, löscht man sie.
3. In dieser Art und Weise hört S diesen `RandomTour` auf, genau an dem Knoten, wo er angefangen hat.
4. Man hat schon einen Zyklus W gefunden. Falls W ein Eulertour ist sind wir fertig.
5. Falls nicht, es muss Knoten $v' \in W$ (auf dem Kreis W) existieren, der noch mindestens eine Kante ausgeht, sonst wäre der Graph nicht zhgd.

[0] *Beweis zur 2 auf übernächster Seite*

Eulertour - Algorithmus

1. Wir haben zwei Läufer, Langsamer (L) und Schneller (S). Beide fangen von dem gleichen Startknoten s an. Zunächst beobachte S.
2. S geht durch den Knoten zufällig. Wann S durch eine Kante läuft, löscht man sie.
3. In dieser Art und Weise hört S diesen `RandomTour` auf, genau an dem Knoten, wo er angefangen hat.
4. Man hat schon einen Zyklus W gefunden. Falls W ein Eulertour ist sind wir fertig.
5. Falls nicht, es muss Knoten $v' \in W$ (auf dem Kreis W) existieren, der noch mindestens eine Kante ausgeht, sonst wäre der Graph nicht zhgd.
6. Jetzt bewegt sich L auf W bis er einen solchen Knoten v' findet.

[0] *Beweis zur 2 auf übernächster Seite*

Eulertour - Algorithmus

1. Wir haben zwei Läufer, Langsamer (L) und Schneller (S). Beide fangen von dem gleichen Startknoten s an. Zunächst beobachte S.
2. S geht durch den Knoten zufällig. Wann S durch eine Kante läuft, löscht man sie.
3. In dieser Art und Weise hört S diesen `RandomTour` auf, genau an dem Knoten, wo er angefangen hat.
4. Man hat schon einen Zyklus W gefunden. Falls W ein Eulertour ist sind wir fertig.
5. Falls nicht, es muss Knoten $v' \in W$ (auf dem Kreis W) existieren, der noch mindestens eine Kante ausgeht, sonst wäre der Graph nicht zhgd.
6. Jetzt bewegt sich L auf W bis er einen solchen Knoten v' findet.
7. Wenn L, den Knoten v' findet, beginnt S mit noch einem `RandomTour`, diesmal von v' aus.

[0] *Beweis zur 2 auf übernächster Seite*

Eulertour - Algorithmus

1. Wir haben zwei Läufer, Langsamer (L) und Schneller (S). Beide fangen von dem gleichen Startknoten s an. Zunächst beobachte S.
2. S geht durch den Knoten zufällig. Wann S durch eine Kante läuft, löscht man sie.
3. In dieser Art und Weise hört S diesen `RandomTour` auf, genau an dem Knoten, wo er angefangen hat.
4. Man hat schon einen Zyklus W gefunden. Falls W ein Eulertour ist sind wir fertig.
5. Falls nicht, es muss Knoten $v' \in W$ (auf dem Kreis W) existieren, der noch mindestens eine Kante ausgeht, sonst wäre der Graph nicht zhgd.
6. Jetzt bewegt sich L auf W bis er einen solchen Knoten v' findet.
7. Wenn L, den Knoten v' findet, beginnt S mit noch einem `RandomTour`, diesmal von v' aus.
8. Diesen `RandomTour` endet auf v' -wo er angefangen hat- weil der Graph mit entfernten Kanten immer noch nur Knoten mit geradem Grad enthält.

[0] *Beweis zur 2 auf übernächster Seite*

Eulertour - Algorithmus contd.

9. Wir haben jetzt einen anderen Zyklus W', der bei v' anfängt und auf v' endet.

Eulertour - Algorithmus contd.

9. Wir haben jetzt einen anderen Zyklus W', der bei v' anfängt und auf v' endet.
10. Jetzt vereinen wir W und W' zu einem neuen Zyklus indem wir zunächst W bis Knoten v', dann der Zyklus W' und dann den Rest des Zyklus W ablaufen.

Eulertour - Algorithmus contd.

9. Wir haben jetzt einen anderen Zyklus W', der bei v' anfängt und auf v' endet.
10. Jetzt vereinen wir W und W' zu einem neuen Zyklus indem wir zunächst W bis Knoten v', dann der Zyklus W' und dann den Rest des Zyklus W ablaufen.
11. Falls dieser neue Zyklus alle Kanten von G enthält dann ist er Eulertour. Falls nicht wiederholen wir den ganzen Prozess.

Warum kommt schneller Läufer S zurück?

Wiederspruchsbeweis:

- Nehme an, dass der Weg W auf einen Knoten $v \neq u$ endet.

Warum kommt schneller Läufer S zurück?

Wiederspruchsbeweis:

- Nehme an, dass der Weg W auf einen Knoten $v \neq u$ endet.
- Daneben nehmen wir an dass S diesen Knoten zuvor k-mal besucht hatte, für ein $k \in \mathbb{N}_0$

Warum kommt schneller Läufer S zurück?

Wiederspruchsbeweis:

- Nehme an, dass der Weg W auf einen Knoten $v \neq u$ endet.
- Daneben nehmen wir an dass S diesen Knoten zuvor k-mal besucht hatte, für ein $k \in \mathbb{N}_0$
- Dann wurden bei jedem Besuch 2 der zu u inzidenten Kanten gelöscht (Hinein- und Herauslaufen), ausser beim letzten Besuch, wobei nur eine Kante beim Hereinlaufen gelöscht wird.

Warum kommt schneller Läufer S zurück?

Wiederspruchsbeweis:

- Nehme an, dass der Weg W auf einen Knoten $v \neq u$ endet.
- Daneben nehmen wir an dass S diesen Knoten zuvor k-mal besucht hatte, für ein $k \in \mathbb{N}_0$
- Dann wurden bei jedem Besuch 2 der zu u inzidenten Kanten gelöscht (Hinein- und Herauslaufen), ausser beim letzten Besuch, wobei nur eine Kante beim Hereinlaufen gelöscht wird.
- Da der Läufer stoppt, bedeutet es, dass es keine inzidente Kanten mehr gibt. -u ist jetzt isoliert-

Warum kommt schneller Läufer S zurück?

Wiederspruchsbeweis:

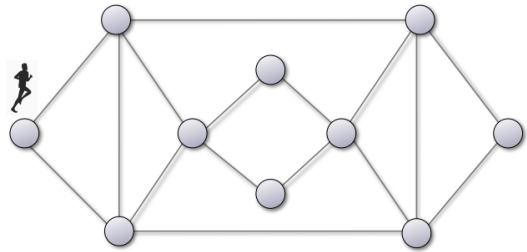
- Nehme an, dass der Weg W auf einen Knoten $v \neq u$ endet.
- Daneben nehmen wir an dass S diesen Knoten zuvor k-mal besucht hatte, für ein $k \in \mathbb{N}_0$
- Dann wurden bei jedem Besuch 2 der zu u inzidenten Kanten gelöscht (Hinein- und Herauslaufen), ausser beim letzten Besuch, wobei nur eine Kante beim Hereinlaufen gelöscht wird.
- Da der Läufer stoppt, bedeutet es, dass es keine inzidente Kanten mehr gibt. -u ist jetzt isoliert-
- Die Summe ergibt $2k + 1$. Wenn man die Kanten von u in W löscht, bleibt ungerade Anzahl von Kanten (insbesondere nicht 0), weil u in G ursprünglich geraden Grad hat.

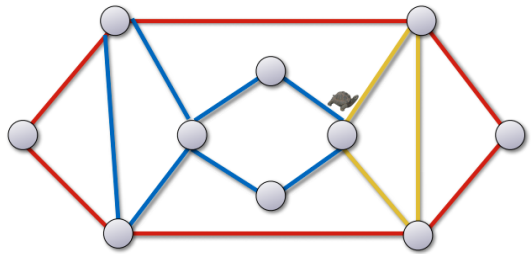
Warum kommt schneller Läufer S zurück?

Wiederspruchsbeweis:

- Nehme an, dass der Weg W auf einen Knoten $v \neq u$ endet.
- Daneben nehmen wir an dass S diesen Knoten zuvor k-mal besucht hatte, für ein $k \in \mathbb{N}_0$
- Dann wurden bei jedem Besuch 2 der zu u inzidenten Kanten gelöscht (Hinein- und Herauslaufen), ausser beim letzten Besuch, wobei nur eine Kante beim Hereinlaufen gelöscht wird.
- Da der Läufer stoppt, bedeutet es, dass es keine inzidente Kanten mehr gibt. -u ist jetzt isoliert-
- Die Summe ergibt $2k + 1$. Wenn man die Kanten von u in W löscht, bleibt ungerade Anzahl von Kanten (insbesondere nicht 0), weil u in G ursprünglich geraden Grad hat.
- Das ist aber ein Widerspruch weil der Läufer S stoppt nur wenn er in einem isolierten Knoten steckenbleibt. $\square$

Graphisches Beispiel aus der Vorlesung





Eulertour - Fazit

Algorithmus für Eulertour (Satz 1.31. im Skript)

Ein zusammenhängender Graph ist genau dann eulersch, wenn der Grad aller Knoten gerade ist.
In einem zusammenhängenden, eulerschen Graphen kann man eine Eulertour in Zeit $O(|E|)$ finden.

Eulertour - Fazit

Algorithmus für Eulertour (Satz 1.31. im Skript)

Ein zusammenhängender Graph ist genau dann eulersch, wenn der Grad aller Knoten gerade ist.
In einem zusammenhängenden, eulerschen Graphen kann man eine Eulertour in Zeit $O(|E|)$ finden.

Laufzeitanalyse: Man kann Kanten -je nach Datenstruktur- in $0(1)$ löschen und zwei Zyklen in $O(1)$ vereinen. Der schnelle Läufer läuft jede Kante nur einmal ab weil man dann die Kante löscht. Der Lauf dauert also insgesamt $O(|E|)$. Der langsame Läufer läuft jede Kante höchstens einmal, was man auch in $O(|E|)$ erledigen kann.

Hamiltonkreis

Hamiltonkreis

Ein Hamiltonkreis in einem Graphen $G = (V, E)$ ist ein Kreis, der alle **Knoten** von V genau einmal durchläuft. Enthält ein Graph einen Hamiltonkreis, so nennt man ihn hamiltonsch.

Hamiltonkreis

Hamiltonkreis

Ein Hamiltonkreis in einem Graphen $G = (V, E)$ ist ein Kreis, der alle **Knoten** von V genau einmal durchläuft. Enthält ein Graph einen Hamiltonkreis, so nennt man ihn hamiltonsch.

- Ein sehr schwierigeres Problem im Vergleich zu Eulertouren.

Hamiltonkreis

Hamiltonkreis

Ein Hamiltonkreis in einem Graphen $G = (V, E)$ ist ein Kreis, der alle **Knoten** von V genau einmal durchläuft. Enthält ein Graph einen Hamiltonkreis, so nennt man ihn hamiltonsch.

- Ein sehr schwierigeres Problem im Vergleich zu Eulertouren. Und zwar NP-vollständig.

Hamiltonkreis

Hamiltonkreis

Ein Hamiltonkreis in einem Graphen $G = (V, E)$ ist ein Kreis, der alle **Knoten** von V genau einmal durchläuft. Enthält ein Graph einen Hamiltonkreis, so nennt man ihn hamiltonsch.

- Ein sehr schwierigeres Problem im Vergleich zu Eulertouren. Und zwar NP-vollständig.
- Kein Polynomzeitalgorithmus, der antwortet die Frage: "Enthält der Graph einen Hamiltonkreis?"

Hamiltonkreis - Algorithmus Ideen

Erste Idee wäre Brute-Force. Wenn man für alle möglichen Permutationen der Knoten versucht, Kanten zu finden um Hamiltonkreise zu erhalten.

Hamiltonkreis - Algorithmus Ideen

Erste Idee wäre Brute-Force. Wenn man für alle möglichen Permutationen der Knoten versucht, Kanten zu finden um Hamiltonkreise zu erhalten.

Davon gibt es $\frac{1}{2}(n-1)!$ viele! D.h. Laufzeit wäre in $O(n!)$. Das möchten wir nicht.

Hamiltonkreis - Algorithmus Ideen

Erste Idee wäre Brute-Force. Wenn man für alle möglichen Permutationen der Knoten versucht, Kanten zu finden um Hamiltonkreise zu erhalten.

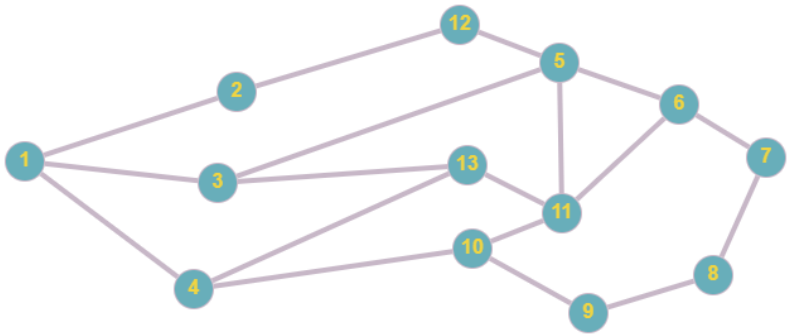
Davon gibt es $\frac{1}{2}(n-1)!$ viele![1] D.h. Laufzeit wäre in $O(n!)$. Das möchten wir nicht.

Eine andere Idee wäre dynamische Programmierung.

[1] fix a starting point otherwise all starting points would yield equivalent cycles, then fix a direction otherwise you would go through the cycle in boht ways and count them as different occurances

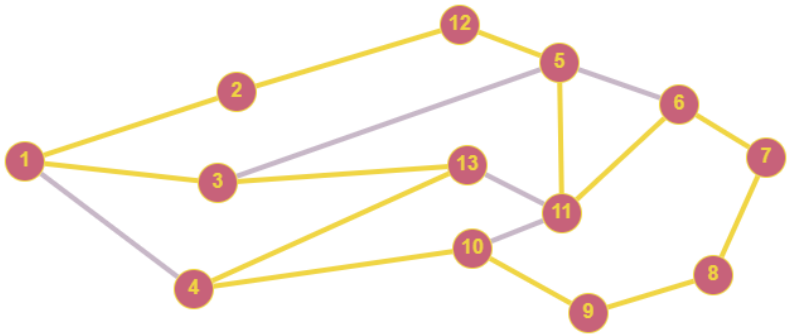
Hamiltonkreis - DP

Sei $G = (V, E)$ wie folgt:



Hamiltonkreis - DP

Spoilers, G enthält einen Hamiltonkreis:



Hamilton - DP

Die Idee unserer Algorithmus ist einfach:

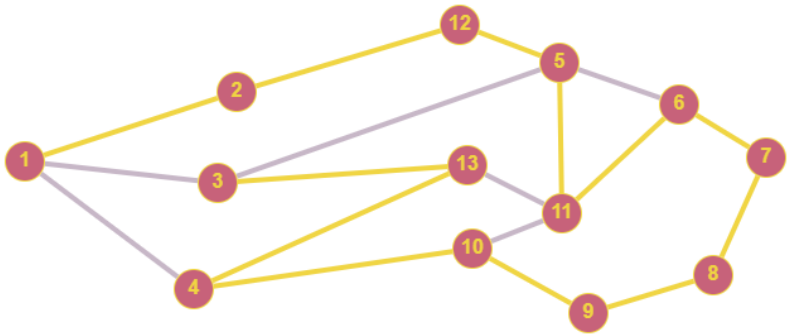
- Betrachte $N(\mathbf{1})$ [2] wobei $\mathbf{1}$ unserer Startkonten ist.
- Falls $x \in N(\mathbf{1})$ und es existiert einen $\mathbf{1}$-x Pfad, der alle Knoten des Graphen enthält[3], können wir diesen Pfad mit der Kante $\{\mathbf{1}, x\}$ schliessen und in einen Hamiltonkreis umwandeln.

[2] Erinnerung Nachbarschaft: $N(v) = \{u \in V \mid \{v, u\} \in E\}$

[3] Ein solcher Pfad heisst Hamiltonpfad *en. Hamiltonian Path*

Hamilton - DP

Hier ist der entschprechende Hamiltonpfad für G:



Hamilton - DP - Einträge und Bedingung

DP-Einträge und Bedingung

Für $G = (V, E)$ sei $V = [n]$. Für alle Teilmengen $S \subseteq [n]$ mit $\mathbf{1} \in S$ und für alle $x \in S$ mit $x \neq \mathbf{1}$ definieren wir:

$$P_{S,x} := \begin{cases} 1, & \text{es gibt in G einen } \mathbf{1}\text{-x-Pfad der genau die Knoten aus } S \text{ enthält} \\ 0, & \textit{sonst} \end{cases}$$

Dann gilt:

$$\text{G enthält einen Hamiltonkreis} \iff \exists x \in N(\mathbf{1}) \text{ mit } P_{[n],x} = 1$$

Hamilton - DP - Tabelle

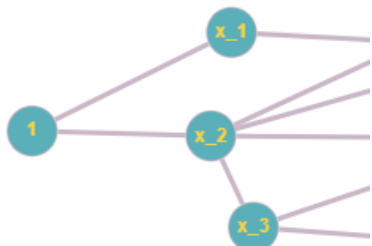
Grösse[4] der DP-Tabelle: $(n-1) \times 2^{n-1} = O(n \cdot 2^n)$

	$\{\mathbf{1}, x_1\}$	$\{\mathbf{1}, x_2\}$	$\{\mathbf{1}, x_3\}$	$\{\mathbf{1}, x_1, x_2\}$	$\{\mathbf{1}, x_2, x_3\}$	$\cdots$
x_1	$P_{\{\mathbf{1},x_1\},x_1}$	$P_{\{\mathbf{1},x_2\},x_1}$	$P_{\{\mathbf{1},x_3\},x_1}$	$P_{\{\mathbf{1},x_1,x_2\},x_1}$	$P_{\{\mathbf{1},x_2,x_3\},x_1}$	$\cdots$
x_2	$P_{\{\mathbf{1},x_1\},x_2}$	$P_{\{\mathbf{1},x_2\},x_2}$	$P_{\{\mathbf{1},x_3\},x_2}$	$P_{\{\mathbf{1},x_1,x_2\},x_2}$	$P_{\{\mathbf{1},x_2,x_3\},x_2}$	$\cdots$
x_3	$P_{\{\mathbf{1},x_1\},x_3}$	$P_{\{\mathbf{1},x_2\},x_3}$	$P_{\{\mathbf{1},x_3\},x_3}$	$P_{\{\mathbf{1},x_1,x_2\},x_3}$	$P_{\{\mathbf{1},x_2,x_3\},x_3}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

[4] $V = [n]$ also $|V| = n$

Hamilton - DP - Tabelle Bsp.

Sei der Folgende ein Abschnitt aus einem Graphen $G = (V, E)$. Wir füllen jetzt die Tabelle für diesen Knoten:



Hamilton - DP - Initialisierung

Für $|S| = 2$ e.g. $\{\mathbf{1}, x\}$ gilt $P_{S,x} = 1 \iff \{\mathbf{1}, x\} \in E$. Also falls es eine Kante zwischen $\mathbf{1}$ und x in E gibt, haben wir den entschprechenden Eintrag gleich 1. Sonst haben wir 0. Damit machen wir die Initialisierung.

	$\{\mathbf{1}, x_1\}$	$\{\mathbf{1}, x_2\}$	$\{\mathbf{1}, x_3\}$	$\{\mathbf{1}, x_1, x_2\}$	$\{\mathbf{1}, x_2, x_3\}$	$\cdots$
x_1	1	0	0	$P_{\{\mathbf{1},x_1,x_2\},x_1}$	$P_{\{\mathbf{1},x_2,x_3\},x_1}$	$\cdots$
x_2	0	1	0	$P_{\{\mathbf{1},x_1,x_2\},x_2}$	$P_{\{\mathbf{1},x_2,x_3\},x_2}$	$\cdots$
x_3	0	0	0	$P_{\{\mathbf{1},x_1,x_2\},x_3}$	$P_{\{\mathbf{1},x_2,x_3\},x_3}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

Hamilton - DP - Rekursion

- Wie gehen wir von Grösse s auf $s+1$? Die Idee ist ähnlich wie Hamiltonkreis auf Hamiltonpfad.
- Wenn $P_{S,x} = 1$, dann muss es ein **1**-x-Pfad geben (oben), der alle Knoten in S benutzt. Lassen wir die letzte Kante weg (unten), dann haben wir einen **1**-x'-Pfad, wobei x' ein Nachbarknoten von x ist.
- Also können wir $P_{S,x}$ für $|S| = s+1$ damit berechnen, indem wir über die $P_{S\setminus\{x\},x'}$ für jeder $x' \in N(x)$ iterieren. (Die Menge $S \setminus \{x\}$ und alle Nachbarknoten $x' \in N(x)$)
- Falls wir einen solchen Pfad finden, also falls $P_{S\setminus\{x\},x'} = 1$ für ein $x' \in N(x)$ dann ist $P_{S,x} = 1$.

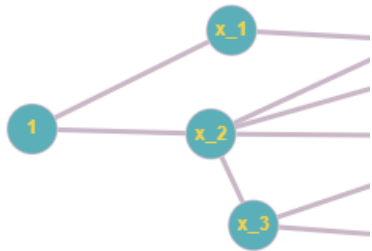
Hamilton - DP - Rekursion

Hamilton-DP-Rekursion

$$P_{S,x} = \max\{P_{S\setminus\{x\},x'} \mid x' \in S \cup N(x), x' \neq 1\}$$

Hamilton - DP - Tabelle Bsp.

Zur Erinnerung:



Hamilton - DP - Tabelle Füllen

Rekursion: $P_{S,x} = \max\{P_{S\setminus\{x\},x'} \mid x' \in S \cup N(x), x' \neq 1\}$

	$\{\mathbf{1}, x_1\}$	$\{\mathbf{1}, x_2\}$	$\{\mathbf{1}, x_3\}$	$\{\mathbf{1}, x_1, x_2\}$	$\{\mathbf{1}, x_2, x_3\}$	$\cdots$
x_1	1	0	0	$P_{\{\mathbf{1},x_1,x_2\},x_1}$	$P_{\{\mathbf{1},x_2,x_3\},x_1}$	$\cdots$
x_2	0	1	0	$P_{\{\mathbf{1},x_1,x_2\},x_2}$	$P_{\{\mathbf{1},x_2,x_3\},x_2}$	$\cdots$
x_3	0	0	0	$P_{\{\mathbf{1},x_1,x_2\},x_3}$	$P_{\{\mathbf{1},x_2,x_3\},x_3}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

Hamilton - DP - Tabelle Füllen

Rekursion: $P_{S,x} = \max\{P_{S\setminus\{x\},x'} \mid x' \in S \cup N(x), x' \neq 1\}$

	$\{\mathbf{1}, x_1\}$	$\{\mathbf{1}, x_2\}$	$\{\mathbf{1}, x_3\}$	$\{\mathbf{1}, x_1, x_2\}$	$\{\mathbf{1}, x_2, x_3\}$	$\cdots$
x_1	1	0	0	0	$P_{\{\mathbf{1},x_2,x_3\},x_1}$	$\cdots$
x_2	0	1	0	$P_{\{\mathbf{1},x_1,x_2\},x_2}$	$P_{\{\mathbf{1},x_2,x_3\},x_2}$	$\cdots$
x_3	0	0	0	$P_{\{\mathbf{1},x_1,x_2\},x_3}$	$P_{\{\mathbf{1},x_2,x_3\},x_3}$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

Hamilton - DP - Tabelle Füllen

Rekursion: $P_{S,x} = \max\{P_{S\setminus\{x\},x'} \mid x' \in S \cup N(x), x' \neq 1\}$

	$\{\mathbf{1}, x_1\}$	$\{\mathbf{1}, x_2\}$	$\{\mathbf{1}, x_3\}$	$\{\mathbf{1}, x_1, x_2\}$	$\{\mathbf{1}, x_2, x_3\}$	$\cdots$
x_1	1	0	0	0	$P_{\{\mathbf{1},x_2,x_3\},x_1}$	$\cdots$
x_2	0	1	0	$P_{\{\mathbf{1},x_1,x_2\},x_2}$	$P_{\{\mathbf{1},x_2,x_3\},x_2}$	$\cdots$
x_3	0	0	0	$P_{\{\mathbf{1},x_1,x_2\},x_3}$	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

Hamilton - DP - Schluss

Satz 1.34 vom Skript

Algorithmus `Hamiltonkreis` ist korrekt und benötigt Spicher $O(n \cdot 2^n)$ und Laufzeit $O(n^2 \cdot 2^n)$, wobei $n = |V|$

Hamilton - DP - Schluss

Satz 1.34 vom Skript

Algorithmus `Hamiltonkreis` ist korrekt und benötigt Spicher $O(n \cdot 2^n)$ und Laufzeit $O(n^2 \cdot 2^n)$, wobei $n = |V|$

Wir können Speicherbedarf mithilfe von Inklusion-Exklusion-Prinzip von $O(n \cdot 2^n)$ auf $O(n^2)$ verbessern.

Daneben können wir die Laufzeit mithilfe von Strassen von $O(n^2 \cdot 2^n)$ auf $O(n^{2.81} \log n \cdot 2^n)$ verbessern.

Hamilton Einige Sätze

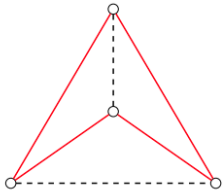
Bipartit Hamiltonsch

Lemma 1.38. Ist $G = (A \uplus B, E)$ ein bipartiter Graph mit $|A| \neq |B|$, so kann G keinen Hamiltonkreis enthalten.

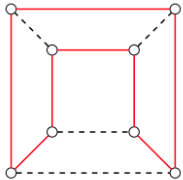
Satz von Dirac

Wenn $G = (V, E)$ ein Graph mit $|V| \geq 3$ Knoten ist, in dem jeder Knoten mindestens $|V|/2$ Nachbarn hat, dann ist G hamiltonsch.

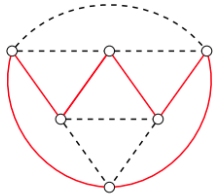
Hamiltonsche Graphen



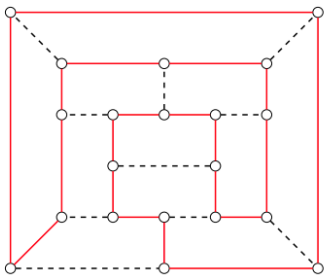
Tetraeder



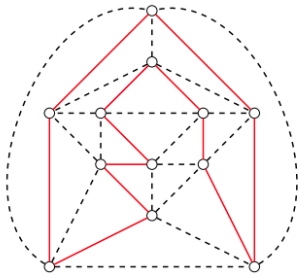
Würfel



Oktaeder



Dodekaeder



Ikosaeder

Inklusion-Exklusion-Prinzip

Satz 1.35. *(Siebformel, Prinzip der Inklusion/Exklusion)* Für endliche Mengen $A_1, \ldots, A_n$ $(n \geq 2)$ gilt:

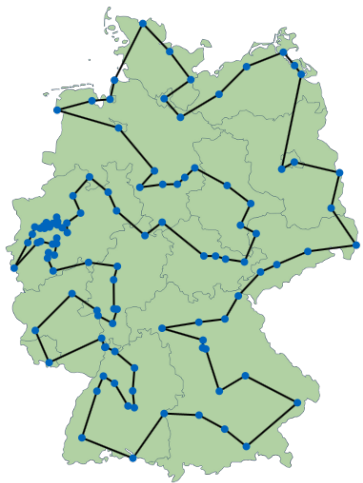
$$\begin{aligned}\left|\bigcup_{i=1}^{n} A_i\right| &= \sum_{l=1}^{n} (-1)^{l+1} \sum_{1 \leq i_1 < \cdots < i_l \leq n} |A_{i_1} \cap \cdots \cap A_{i_l}| \\ &= \sum_{i=1}^{n} |A_i| - \sum_{1 \leq i_1 < i_2 \leq n} |A_{i_1} \cap A_{i_2}| + \sum_{1 \leq i_1 < i_2 < i_3 \leq n} |A_{i_1} \cap A_{i_2} \cap A_{i_3}| \\ &\quad - \cdots + (-1)^{n+1} \cdot |A_1 \cap \cdots \cap A_n|.\end{aligned}$$

Traveling Salesman Problem

Gegeben:

vollständiger Graph K_n

Längefunktion $\ell : \binom{[n]}{2} \to \mathbb{R}$ die jeder Kante des Graphen eine Länge zuordnet



Gesucht:

ein Hamiltonkreis C in K_n mit:

$$\min_{C} \sum_{e \in E(C)} \ell(e)$$

α-Approximationsalgorithmus

Optimale Lösung

Wir bezeichnen die Länge einer optimalen Lösung wie folgt:

$$opt(K_n, \ell) := \min\{\sum_{e \in C} \ell(e) \mid C \text{ ist ein Hamiltonkreis in } K_n\}$$

α-Approximationsalgorithmus

Entschprechend spricht man von einem α-Approximationsalgorithmus, wenn der Algorithmus immer einen Hamiltonkreis C findet mit

$$\sum_{e \in C} \ell(e) \leq \alpha \cdot opt(K_n, \ell)$$

Hamilotnkreis auf TSP (Polynomzeitreduktion)

Für ein $G = (V, E)$ kann man folgendes Problem definieren:

Sei $K_n = ([n], \binom{[n]}{2})$ Definiere Längefunktion $\ell : \binom{[n]}{2} \to \{0, 1\}$ so dass

$$\ell(\{x, y\}) = \begin{cases} 0, & \text{falls } \{x, y\} \in E \\ 1, & \text{sonst} \end{cases}$$

Hamilotnkreis auf TSP (Polynomzeitreduktion)

Für ein $G = (V, E)$ kann man folgendes Problem definieren:

Sei $K_n = ([n], \binom{[n]}{2})$ Definiere Längefunktion $\ell : \binom{[n]}{2} \to \{0, 1\}$ so dass

$$\ell(\{x, y\}) = \begin{cases} 0, & \text{falls } \{x, y\} \in E \\ 1, & \text{sonst} \end{cases}$$

- Stellt euch vor, wir hätten eine Blackbox, das TSP-Problem in Polynomialzeit löst.
- Wir könnten dann auch die oben definierte Instanz von TSP-Problem in Polynomialzeit lösen.
- Die Lösung dieser oben definierten Instanz von TSP beantwortet aber die frage: "Hat G einen Hamiltonkreis?"
- Also wenn wir so eine Blackbox hätten, könnten wir auch Hamiltonkreis Problem in Polynomialzeit lösen.

Kein α-Approximationsalgorithmus mit $\alpha > 1$

Satz 1.42

Gibt es für ein $\alpha > 1$ einen α-Approximationsalgorithmus für das Travelling Salesman Problem mit Laufzeit $O(f(n))$, so gibt es auch einen Algorithmus, der für alle Graphen auf n Knoten in Laufzeit $O(f(n))$ entscheidet, ob sie hamiltonsch sind.

Kein α-Approximationsalgorithmus mit $\alpha > 1$

Satz 1.42

Gibt es für ein $\alpha > 1$ einen α-Approximationsalgorithmus für das Travelling Salesman Problem mit Laufzeit $O(f(n))$, so gibt es auch einen Algorithmus, der für alle Graphen auf n Knoten in Laufzeit $O(f(n))$ entscheidet, ob sie hamiltonsch sind.

Beweisidee: $opt(K_n, \ell) = 0$ für ℓ wie wir sie oben definiert haben. Eine α-Approximation würde (wegen $0 \cdot \alpha = 0$) immer die exakte Lösung finden, also einen Hamiltonkreis.

Metrisches Traveling Salesman Problem

Jetzt fügen wir eine natürliche Annahme zu originalem TSP.

Gegeben:

vollständiger Graph K_n

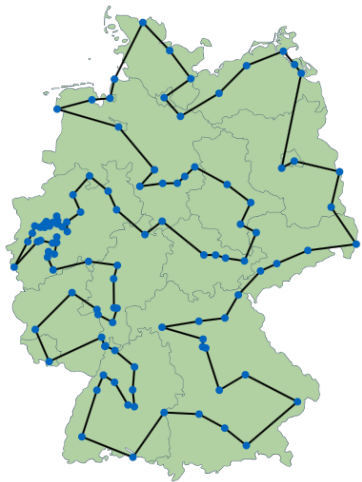
Längefunktion $\ell : \binom{[n]}{2} \to \mathbb{R}$

mit

$$\ell(\{x,z\}) \leq \ell(\{x,y\}) + \ell(\{y,z\})$$

für alle $x, y, z \in [n]$

Dreiecksungleichung

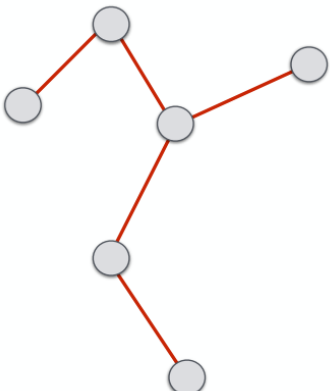


Gesucht:

ein Hamiltonkreis C in K_n mit:

$$\min_{C} \sum_{e \in E(C)} \ell(e)$$

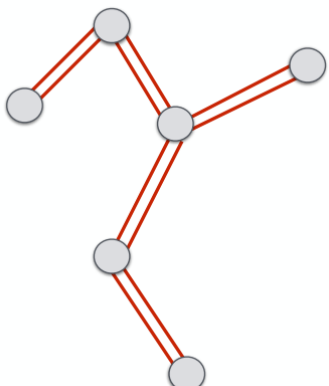
Metrisches TSP - Algorithmus



1. Bestimme MST $\mathbf{T}$, *es gilt*

$$\ell(T) \leq opt(K_n, \ell)$$

Metrisches TSP - Algorithmus



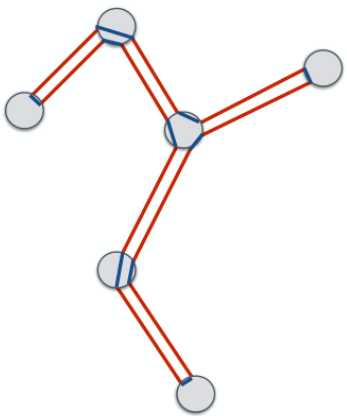
1. Bestimme MST **T**, *es gilt*

$$\ell(T) \leq opt(K_n, \ell)$$

2. verdopple alle Kanten von **T** *es gilt*

$$2\ell(T) \leq 2opt(K_n, \ell)$$

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt*

$$\ell(T) \leq opt(K_n, \ell)$$

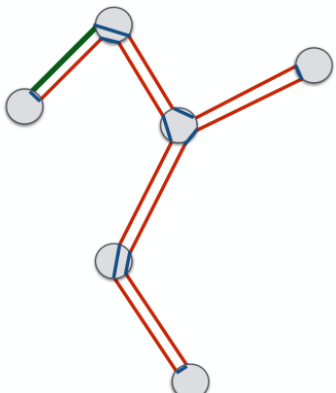
2. verdopple alle Kanten von **T** *es gilt*

$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

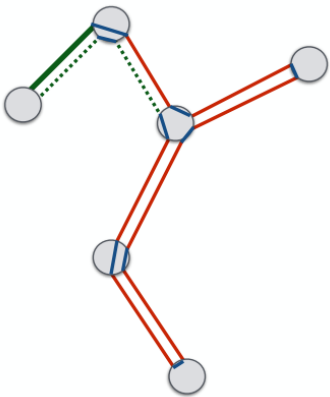
$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**)

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

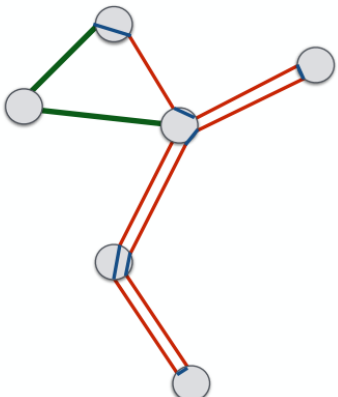
$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**)

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

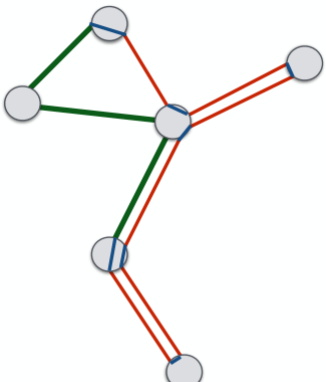
$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**)

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

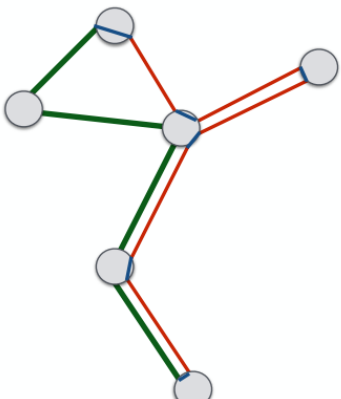
$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**)

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

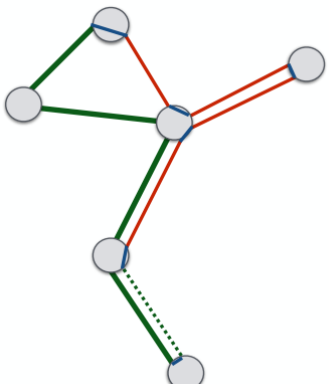
$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**)

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

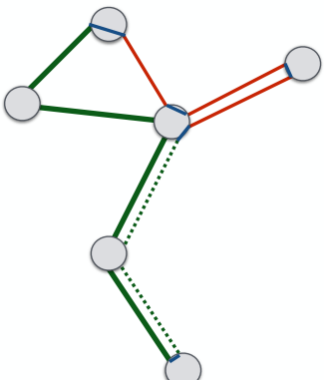
$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**)

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

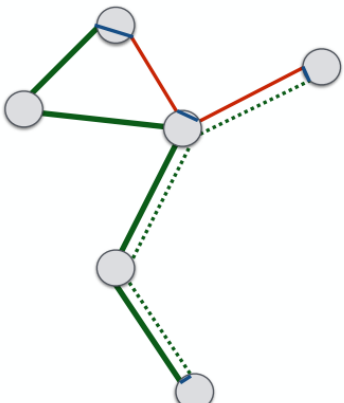
$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**)

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

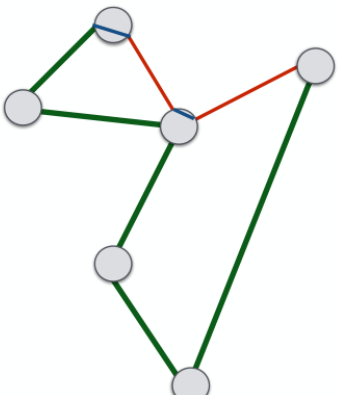
$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**)

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

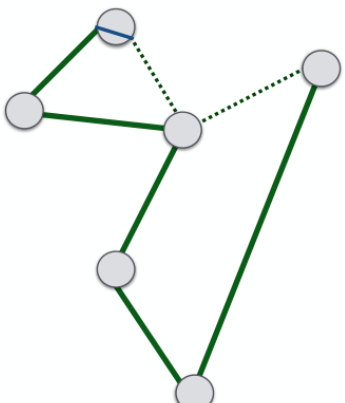
$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**)

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

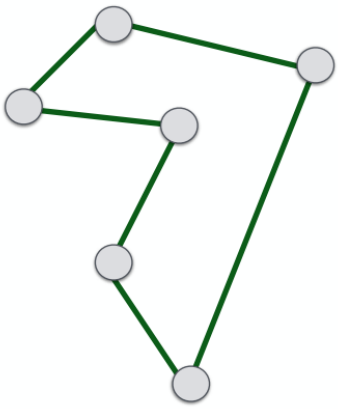
$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**)

Metrisches TSP - Algorithmus



1. Bestimme MST **T**, *es gilt* $\ell(T) \leq opt(K_n, \ell)$
2. verdopple alle Kanten von **T** *es gilt*

$$2\ell(T) \leq 2opt(K_n, \ell)$$

3. bestimme *Eulertour* **W** *es gilt*

$$\ell(W) = 2\ell(T) \leq 2opt(K_n, \ell)$$

4. Durchlaufe **W** mit Abkürzungen, so dass jeder Knoten nur einmal besucht wird (das ergibt Hamiltonkreis **C**) *es gilt*

$$\ell(C) \leq \ell(W) \underset{\Delta ungl.}{=} 2\ell(T) \leq 2opt(K_n, \ell)$$

Metrisches TSP - Fazit

Satz 1.43.

Für das Metrische Travelling Salesman Problem gibt es einen 2-Approximationsalgorithmus mit Laufzeit $O(n^2)$

[4] $O(n^2)$ wegen MST Berechnung mit z.B. Algorithmus von Prim

The End

♠