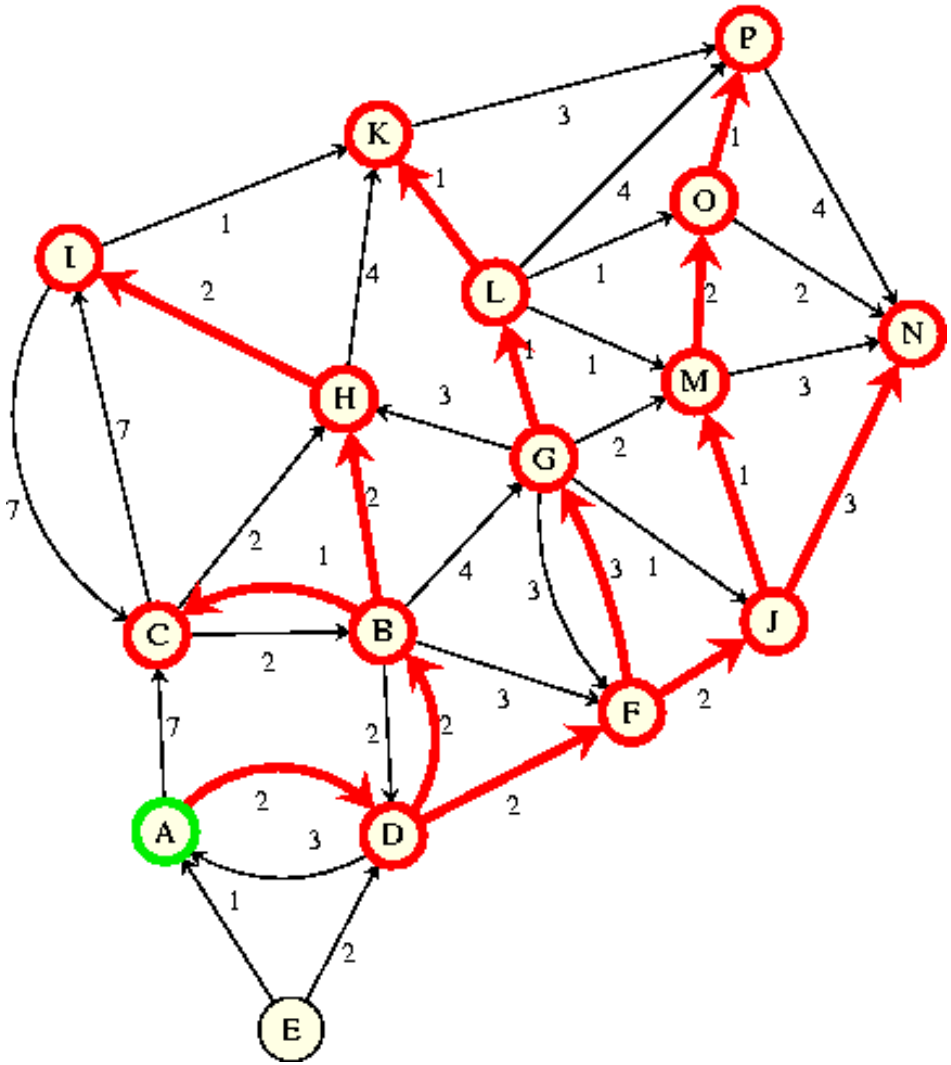
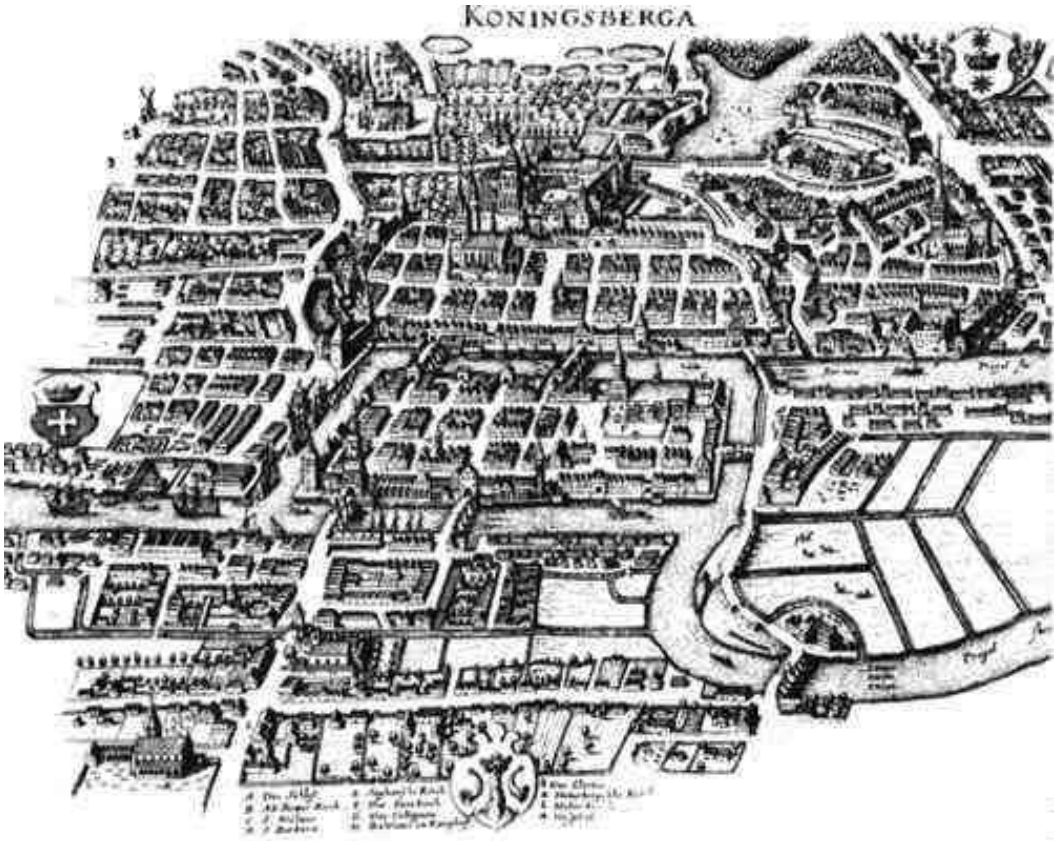
Kapitel 4

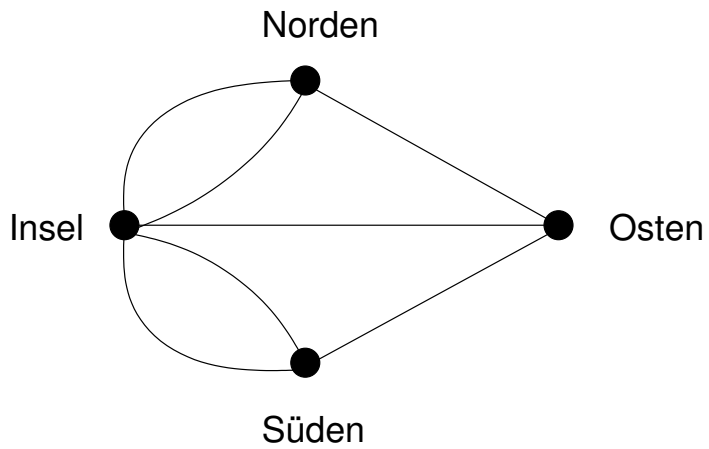
Kreis- und Wegeprobleme



Inhalt

Das Königsberger Brückenproblem



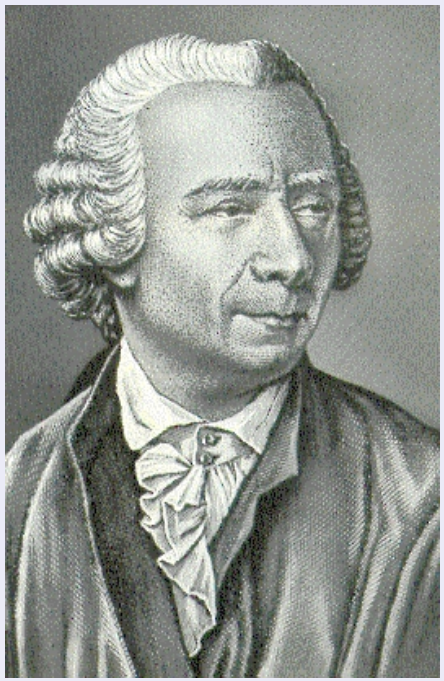


Eulerweg, Eulerkreis

Definition 4.1

Es sei $G = (V, E)$ ein Graph.

- Ein Weg, der jede Kante von G genau einmal enthält, heißt eulerscher Weg von G.
- Ein Kreis, der jede Kante von G genau einmal enthält, heißt eulerscher Kreis von G.
- G heißt eulersch gdw. G einen eulerschen Kreis enthält.

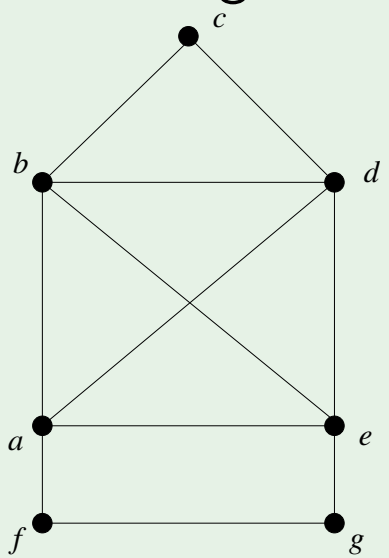


Leonhard Euler

Beispiel 4.2

Der Graph des Königsberger Brückenproblems scheint nicht eulersch zu sein, er scheint auch keinen eulerschen Weg zu enthalten.

Das "Haus des Nikolaus" enthält einen eulerschen Weg.



Das "Haus des Nikolaus mit Keller" ist eulersch:

Weitere Graphen: ✎.

Charakterisierung von eulerschen Graphen

Satz 4.3 (Euler 1736)

Es sei $G = (V, E)$ ein Graph.

G hat einen eulerschen Weg gdw.

- *G bis auf isolierte Knoten zusammenhängend ist und*
- *für die Zahl u der Knoten mit ungeradem Grad gilt: $u = 0$ oder $u = 2$.*

Die Existenz eines Eulerkreises ist äquivalent mit $u = 0$.

Beweis zu Satz 4.3

Beweis.

1. "$\Longrightarrow$": G habe einen eulerschen Kreis $K = (v_0, v_1, \ldots, v_{m-1}, v_0)$.
   - Dann ist G bis auf isolierte Knoten zusammenhängend und
   - tritt der Knoten v in der Folge $v_0, v_1, \ldots, v_{m-1}$ genau t-mal auf, so gilt $deg(v) = 2t$, d.h. v hat geraden Grad.
2. "$\Longleftarrow$": Beweis durch vollständige Induktion über die Zahl der Knoten.

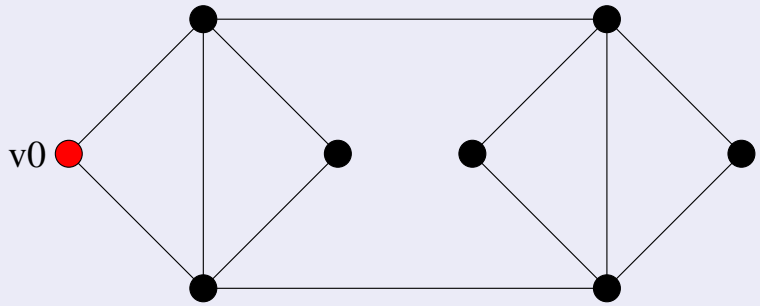
Induktionsanfang: Der Graph $G = (\{v_0\}, \{\})$ ist eulersch, denn (v_0) ist ein eulerscher Kreis.

Induktionsannahme: Für Graphen mit höchsten n Knoten gelte die Behauptung.

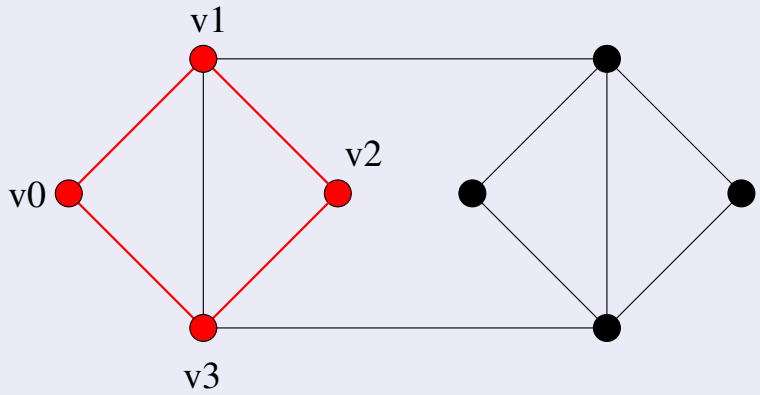
Fortsetzung Beweis.

Induktionsschritt: Sei $G = (V, E)$ ein zusammenhängender Graph mit $n + 1$ Knoten und und alle Knoten haben geraden Grad.

Wähle einen beliebigen Knoten $v_0 \in V$.



Wähle solange dies möglich ist Knoten $v_1, v_2, \ldots$, so dass $(v_0, \ldots, v_i)$ jeweils ein Weg in G ist.

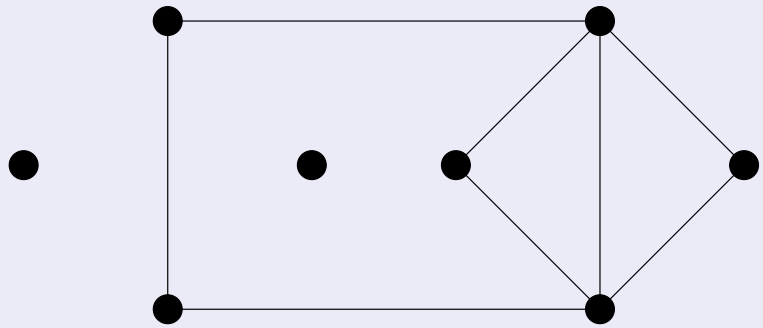


Unter den gegebenen Voraussetzungen entsteht so automatisch ein Kreis K. Sei E_k die Menge der Kanten in K.

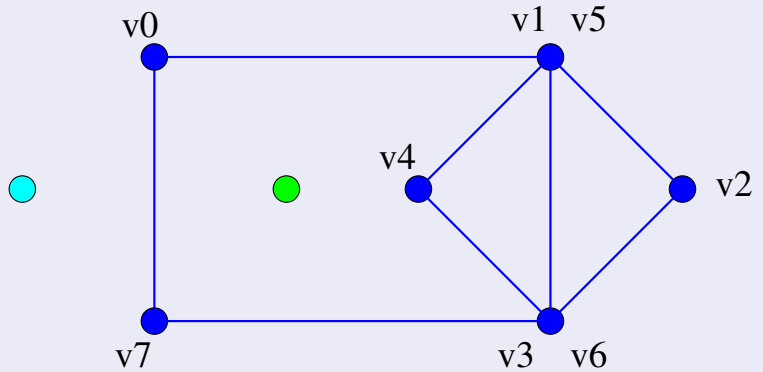
Fortsetzung Beweis.

Wenn K alle Kanten aus E enthält, so ist K ein Eulerkreis.

Ansonsten bildet man den Restgraphen $G' = (V, E \setminus E_K)$.

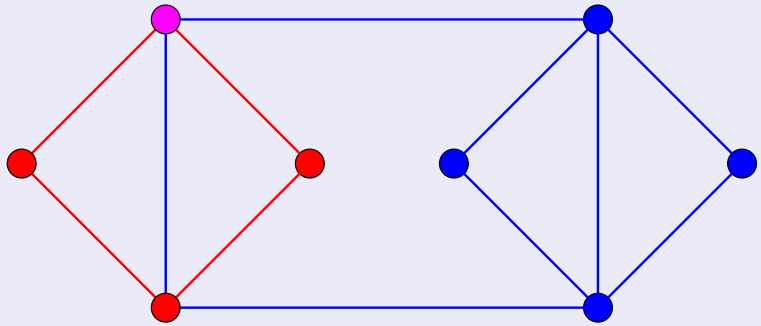


Für die Komponenten des Restgraphen gilt die Induktionsannahme.

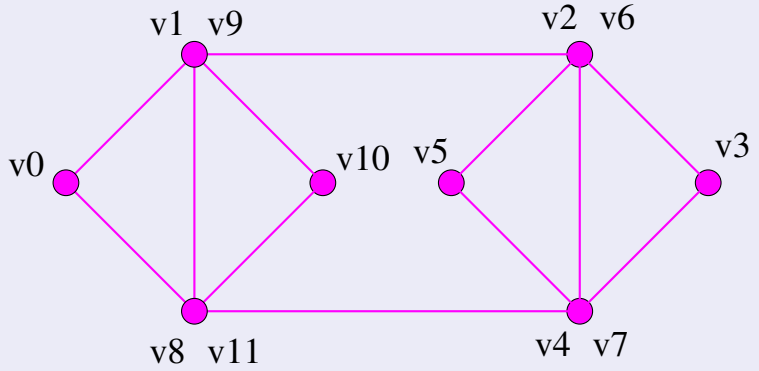


Fortsetzung Beweis.

Nun fügt man die Kreise an Knoten zusammen, die in K und einer Komponente des Restgraphen enthalten sind.



Man läuft entlang K bis zu solch einem Knoten, dann entlang des Kreises des Restgraphen und anschließend wieder entlang K.



Berechnung eines Eulerkreises

Algorithmus 4.4 (Hierholzer 1873)

Es sei $G = (V, E)$ ein bis auf isolierte Knoten zusammenhängender Graph, der nur Knoten mit geradem Grad aufweist.

1. *Wähle einen beliebigen Knoten $v_0 \in V$.*

   Wähle solange dies möglich ist Knoten $v_1, v_2, \ldots$, so dass $(v_0, \ldots, v_i)$ jeweils ein Weg in G ist.

   Unter den gegebenen Voraussetzungen entsteht so automatisch ein Kreis K. Setze $w' := v_0$.
2. *Prüfe, ob K ein eulerscher Kreis ist. Wenn ja, dann STOP, ansonsten gehe zu 3.*

Fortsetzung Algorithmus.

3. *Setze $K' := K$.*

   Laufe ab w' entlang K' und wähle einen in K' enthaltenen Knoten w, der mit einer nicht in K' enthaltenen Kante inzident ist.

   Konstruiere wie unter 1. ausgehend von w einen Kreis K'', der keine Kanten von K' enthält.

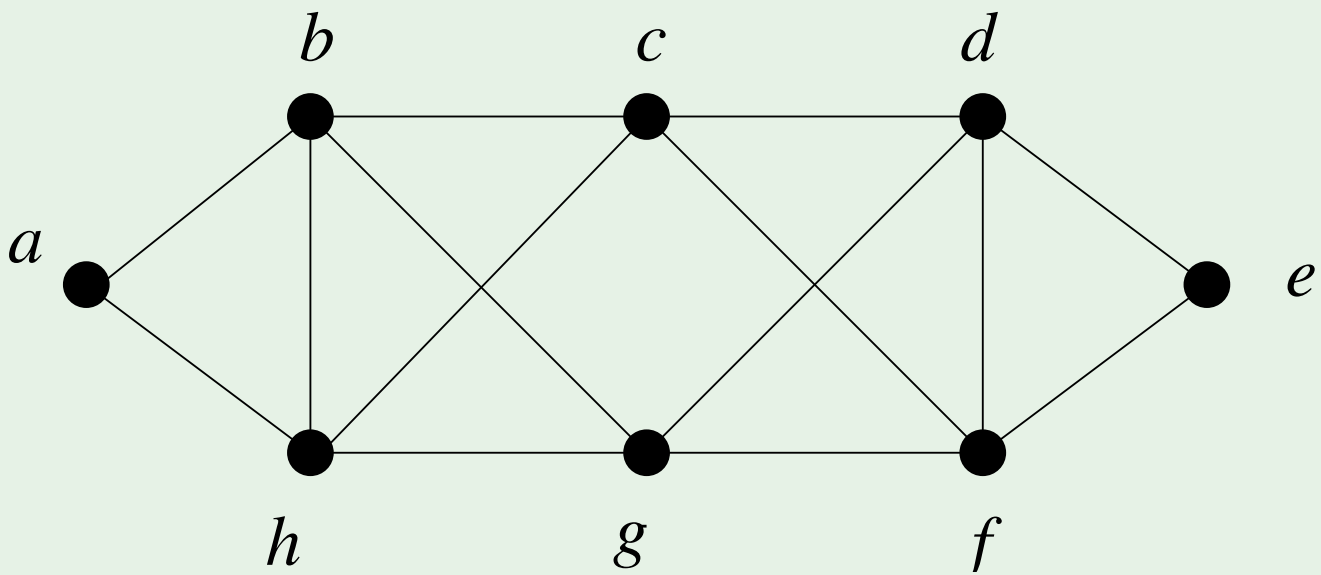
   Füge K'' in den Kreis K' an der Stelle w ein. Setze $w' := w$ und $K := K'$.

   Gehe zu 2.

Beispiel zum Hierholzer-Algorithmus

Beispiel 4.5

Wir demonstrieren den Algorithmus von Hierholzer an dem folgenden Graphen:



Tafel ✎.

Eigenschaften des Algorithmus von Hierholzer

Satz 4.6

Algorithmus 4.4 ist korrekt, d.h. bei erfüllten Voraussetzungen wird ein eulerscher Kreis konstruiert.

Bemerkung: Wir können Algorithmus 4.4 auch zur Berechnung eines Eulerweges benutzen.

Satz 4.7

Die Zeitkomplexität von Algorithmus 4.4 beträgt $O(|V| + |E|)$.

Beweis.

Siehe Übungen.

Anwendung von Eulerwegen

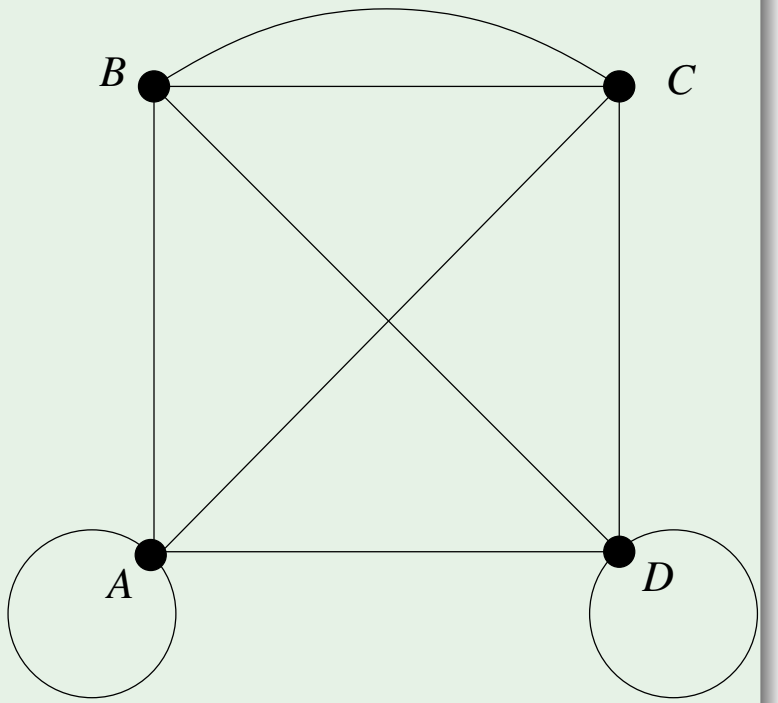
- Dominospiel: Gegeben sind eine Menge S von Spielsteinen, die auf jeder Seite mit einem Symbol markiert sind.
- Einen Spielstein $[A : B]$ kann man sowohl in dieser Form als auch als $[B : A]$ verwenden.
- Man darf zwei Spielsteine aneinander legen, wenn die sich berührenden Hälften das gleiche Symbol aufweisen.
- Kann man die Steine einer Menge S zu einer ununterbrochenen Kette zusammenlegen?

Beispiel 4.8

Gegeben ist die folgende Menge an Spielsteinen:

$[A : B]$ $[A : D]$ $[B : C]$ $[C : D]$ $[A : A]$
$[D : D]$ $[B : C]$ $[A : C]$ $[B : D]$

Zugehöriger Graph:



- Jede Kante entspricht einem Spielstein.
- Eulerweg entspricht einer ununterbrochenen Dominokette.

Anwendung in der Praxis: Berechnung von Prozessketten mit möglichst wenigen Unterbrechungen.

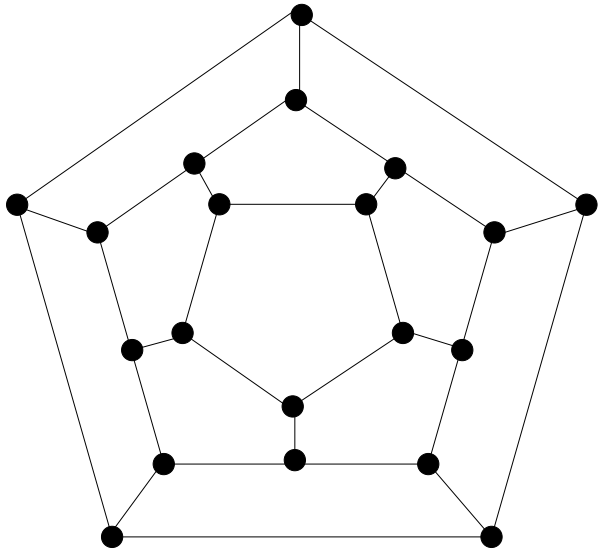
Hamiltonsche Graphen

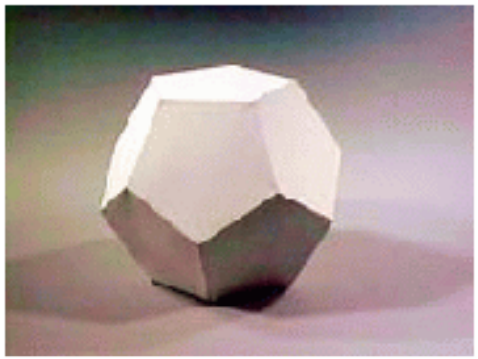
Definition 4.9

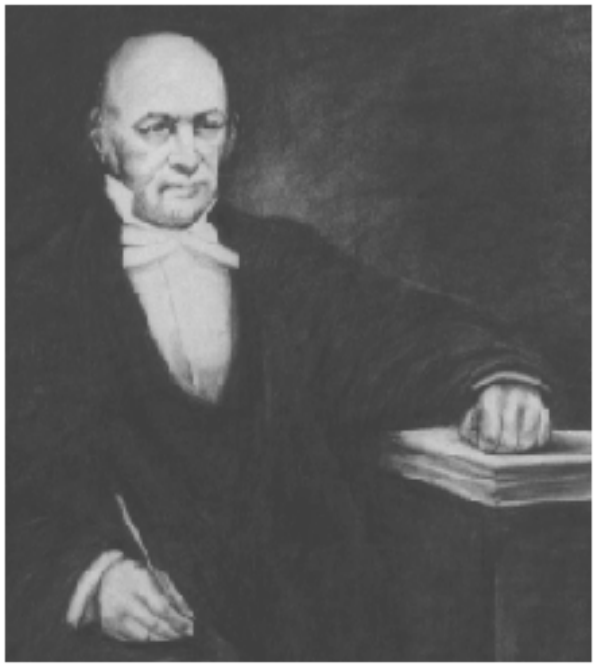
Es sei $G = (V, E)$ ein Graph.

- Ein Weg, der jeden Knoten von G genau einmal enthält, heißt hamiltonscher Weg.
- Ein Kreis, der jeden Knoten von G genau einmal enthält, heißt hamiltonscher Kreis.
- G heißt hamiltonsch gdw. G einen hamiltonschen Kreis enthält.

- Die Bezeichnung “hamiltonsch” geht auf Sir William Rowan Hamilton (1805 – 1865) zurück, der 1859 das Spiel “around the world” erfand.
- Die Punkte eines Dodekaeders stellten Städte dar.
- Die Aufgabe des Spiels bestand darin, entlang der Kanten des Dodekaeders eine Rundreise zu unternehmen, bei der man jede Stadt genau einmal besucht.







Sir William Rowan Hamilton

Around the World

Charakterisierung von hamiltonschen Graphen

Satz 4.10

Es sei $G = (V, E)$ ein Graph mit $n := |V| \geq 3$. Gilt:

$$\forall v \in V : deg(v) \geq n/2,$$

dann ist G hamiltonsch.

Beweis.

Wir erzeugen den Graph G' aus G, indem wir

- k neue Knoten in G einfügen und
- jeden neuen Knoten mit allen Knoten von G über neue Kanten verbinden.

Wie viele solcher neuer Knoten brauchen wir mindestens, damit G' hamiltonsch wird?

Fortsetzung Beweis.

Mit $k = n$ gelingt es uns immer, G' hamiltonsch werden zu lassen.

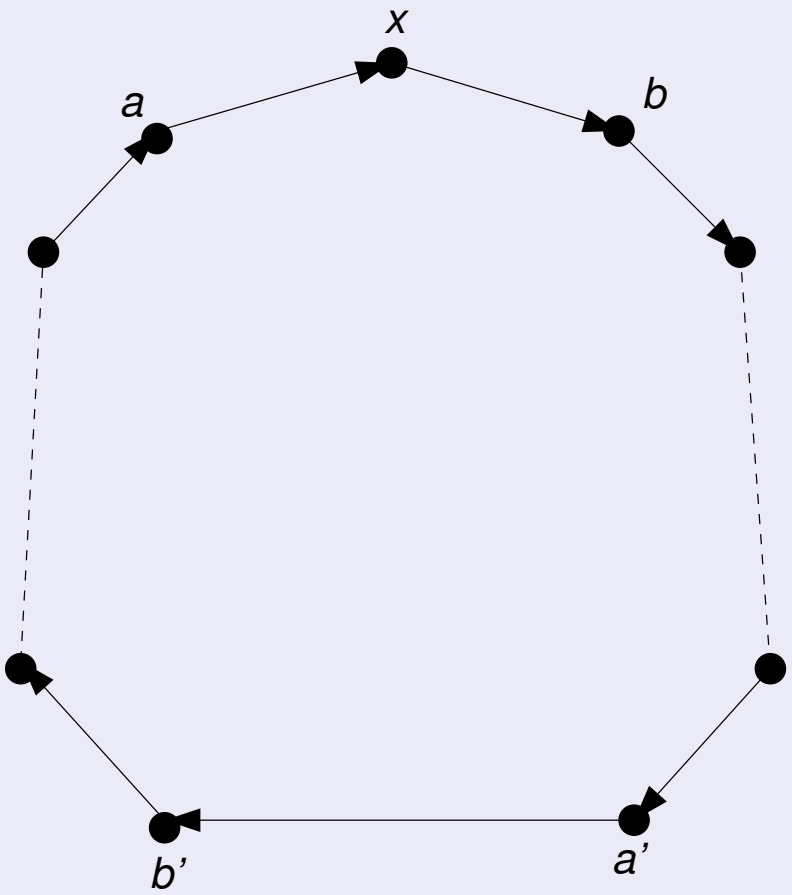
Wir wählen k so klein wie möglich ist und nehmen an, dass $k > 0$ gilt. Hieraus konstruieren wir einen Widerspruch.

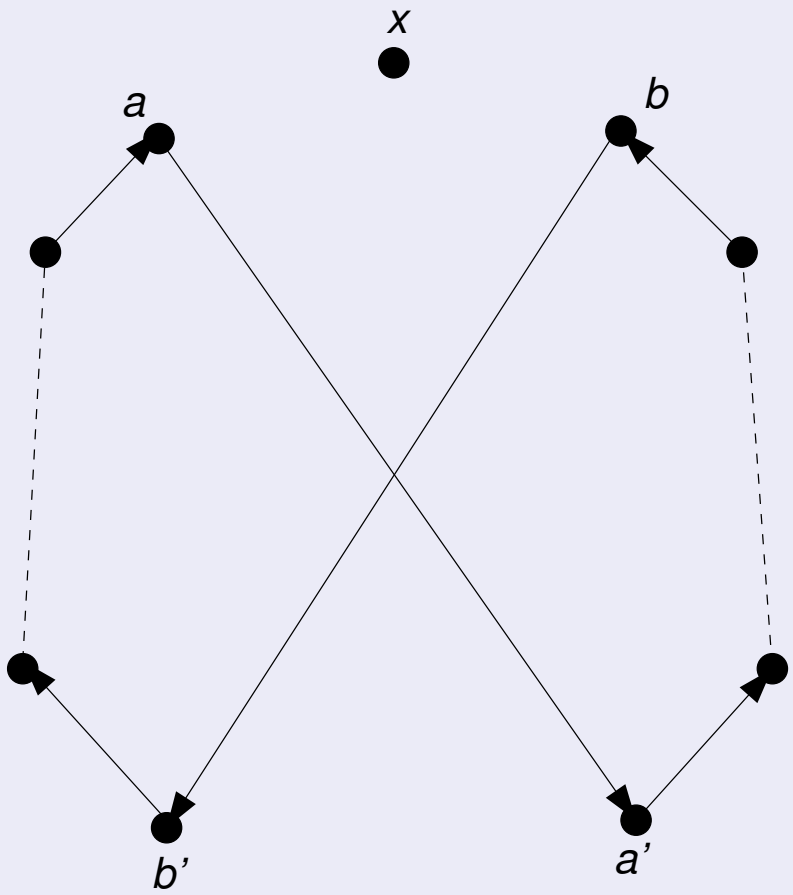
Es sei $K = (a, x, b, \ldots, a)$ ein hamiltonscher Kreis von G', a, b seien Knoten von G, x sei ein neuer Knoten. Dann gilt:

1. a und b sind nicht benachbart, denn sonst wäre x für den hamiltonschen Kreis überflüssig.
2. In K kann ein zu b benachbarter Knoten b' nie direkt auf einen zu a benachbarten Knoten a' folgen.

Fortsetzung Beweis.

Beweisskizze zu (2):





Fortsetzung Beweis.

Wir definieren:

A := Menge der Nachbarn von a in G'

B := Menge der Nachbarn von b in G'

F := Menge der Knoten von G', die in K direkt auf einen Knoten aus A f

Wegen (i) und (ii) gilt: $B \cap F = \emptyset$.
Weiterhin gilt: $|B| \geq \frac{n}{2} + k$ und $|F| = |A| = \frac{n}{2} + k$.

Konsequenz: $|B \cup F| = |B| + |F| \geq n + 2k$.
Da G' nur $n + k$ Knoten hat, ist dies ein Widerspruch für $k > 0$.

Bemerkung: Das Entscheidungsproblem, ob ein Gaph G hamiltonsch ist (HC), ist $\mathcal{NP}$-vollständig.

Crashkurs $\mathcal{NP}$-Vollständigkeit

Die Klasse $\mathcal{P}$:

- Ein Entscheidungsproblem heißt polynomiell lösbar, wenn es einen Algorithmus zur Lösung des Problems gibt, dessen Worst-Case-Laufzeit durch ein Polynom in der Länge der Eingabe beschränkt ist.
- Die Klasse $\mathcal{P}$ ist die Klasse der polynomiell lösbaren Probleme.
- Beispiele: Ist der Graph G zusammenhängend? Hat der Graph G einen eulerschen Kreis?

Die Klasse $\mathcal{NP}$

- Die Klasse $\mathcal{NP}$ ist die Klasse der Entscheidungsprobleme, für die ein Lösungsvorschlag in polynomieller Rechenzeit überprüft werden kann.
- Beispiel: Überprüfung, ob eine Knotenfolge eines Graphen einen hamiltonschen Kreis bildet.
- Beispiel: Überprüfung, ob eine TSP-Tour eine Länge $\leq l$ hat.

Beziehung zwischen $\mathcal{P}$ und $\mathcal{NP}$

Es gilt: $\mathcal{P} \subseteq \mathcal{NP}$

Gilt sogar $\mathcal{P} = \mathcal{NP}$?

Man weiß es nicht! Man vermutet, dass dies nicht der Fall ist. Stand heute kann man aber

$$\text{weder } \mathcal{P} = \mathcal{NP} \text{ noch } \mathcal{P} \neq \mathcal{NP}$$

beweisen.

- ☞ Die Lösung des $\mathcal{P} = \mathcal{NP}$ Problems gehört zu den wichtigsten offenen Problemen der Mathematik/Informatik.

$\mathcal{NP}$-Vollständigkeit

- Ein Problem Π aus $\mathcal{NP}$ ist $\mathcal{NP}$-vollständig, wenn aus seiner polynomiellen Lösbarkeit $\mathcal{P} = \mathcal{NP}$ folgt, d.h. alle Probleme in $\mathcal{NP}$ polynomiell lösbar wären.
- Die $\mathcal{NP}$-vollständigen Probleme können anschaulich als die "schwersten" Probleme in der Klasse $\mathcal{NP}$ angesehen werden.

Wie kann man zeigen, dass ein Problem Π $\mathcal{NP}$-vollständig ist?

- Man nimmt sich ein Problem Π', von dem man weiß, dass es $\mathcal{NP}$-vollständig ist und zeigt, dass Π "nicht leichter" ist.
- Genauer: Man reduziert Π' auf Π, d.h. man zeigt, dass man Π' in polynomieller Zeit lösen könnte, wenn man Π in polynomieller Zeit lösen kann.
- Schreibweise:

$$\Pi' \propto \Pi$$

Welche $\mathcal{NP}$-vollständigen Probleme kennt man denn?

Satz 4.11 (Satz von Cook (1971))

Das Erfüllbarkeitsproblem der Aussagenlogik (SAT) ist $\mathcal{NP}$-vollständig.

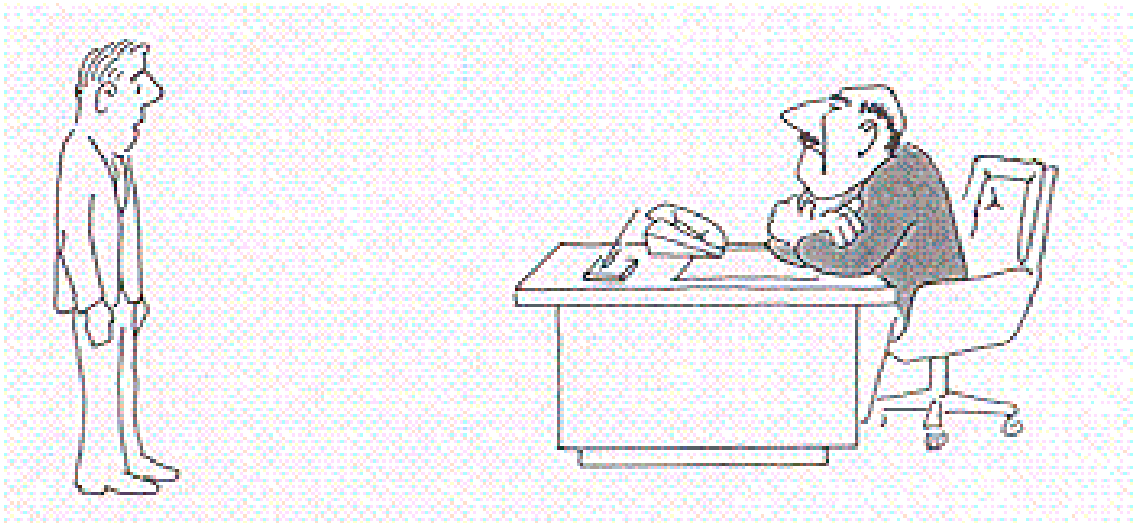
Das Erfüllbarkeitsproblem der Aussagenlogik fragt, ob eine aussagenlogische Formel erfüllbar ist.

Karps 21 $\mathcal{NP}$-vollständige Probleme [1972]: Richard Karp griff die Idee von Cook auf und zeigte für 21 verschiedene kombinatorische und graphentheoretische Probleme, dass sie $\mathcal{NP}$-vollständig sind, darunter auch das Hamiltonkreisproblem (HC).

Seitdem konnte für viele weitere Probleme gezeigt werden, dass sie $\mathcal{NP}$-vollständig sind.

Was bringt einem das in der Praxis?

Stellen Sie sich vor, Ihr Chef gibt Ihnen den Auftrag, für ein wichtiges Problem innerhalb eines Projekts einen effizienten Algorithmus zu programmieren, Ihnen gelingt die Konstruktion solch eines Algorithmus aber nicht.



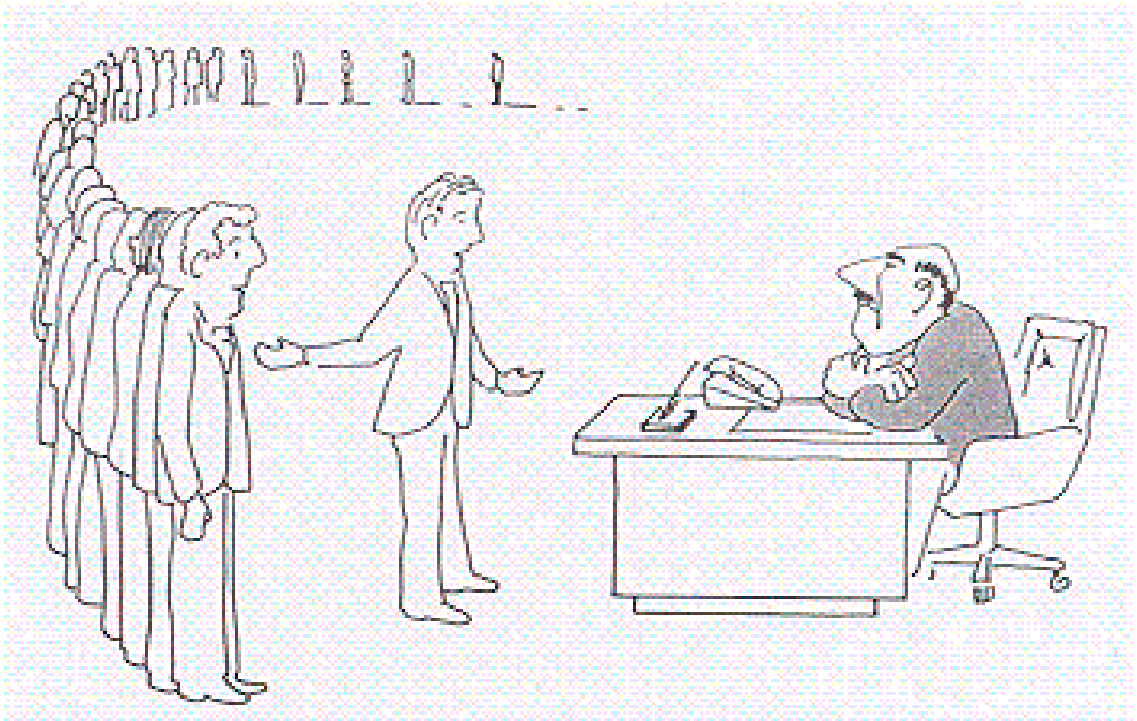
I can't find an efficient algorithm, I guess I'm just too dumb.

Sie wären natürlich aus dem Schneider, wenn Sie zeigen könnten, dass solch ein Algorithmus nicht existiert.

I can't find an efficient algorithm, because no such algorithm is possible.

Unglücklicherweise gelingt Ihnen dies aber auch nicht.

Wenn Sie aber nachweisen, dass das betreffen Problem $\mathcal{NP}$-vollständig ist, haben Sie gegenüber Ihrem Chef gute Argumente.



I can't find an efficient algorithm, but neither can all these famous people.

- Gelingt der Nachweis der $\mathcal{NP}$-Vollständigkeit, so zeigt dies, dass man nicht zu dumm ist, ein polynomielles Lösungsverfahren zu finden.
- Vermutlich existiert keines. Zumindest hat noch nie jemand ein solches gefunden und es macht keinen Sinn, nach einem solchen zu suchen.
- Man kann für "große" Probleme keine (optimalen) Lösungen erwarten.
- Anwendung heuristischer statt exakter Verfahren

Traveling Salesman Problem

Definition 4.12

Gegeben sei ein vollständiger Graph $G = (V, E)$ mit einer Kostenfunktion $c : E \longrightarrow \mathbb{N}$ auf den Kanten.

Die Enscheidungsvariante des Traveling Salesman Problems (TSP) lautet: Existiert ein hamiltonscher Kreis (eine TSP-Tour) K in G, für den die Summe der Kantengewichte $c(K) \leq k$ ist.

Bemerkungen:

- Die Entscheidungsvariante des TSP ist $\mathcal{NP}$-vollständig.
- Dies kann durch $HC \propto TSP$ bewiesen werden.

Skizze für $HC \propto TSP$

- Es sei $G = (V, E)$ ein Graph. Wir wollen entscheiden, ob G hamiltonsch ist.
- Es sei $G' = (V, E')$ ein vollständiger Graph für die Knotenmenge von G. Die Kostenfunktion $c : E' \longrightarrow \mathbb{N}$ sei

$$c(e) = \begin{cases} 1 & \text{für } e \in E, \text{ also die Kanten aus } G \\ 2 & \text{sonst} \end{cases}$$

- Dann hat G genau dann einen hamiltonschen Kreis, wenn in G' eine TSP-Tour der Länge $|V|$ existiert.
- Konsequenz: Würde es einen polynomiellen Algorithmus für das TSP-Problem geben, wäre auch das HC-Problem polynomiell lösbar.

TSP als Optimierungsproblem

- Die Optimierungsvariante des TSP lautet: Finde einen hamiltonschen Kreis mit minimalem Gewicht.
- Durch die $\mathcal{NP}$-Vollständigkeit des zugehörigen Entscheidungsproblems kann es optimal nur für "kleine" n gelöst werden.
- Für "große" n müssen Heuristiken zur Berechnung einer möglichst guten Lösung angewendet werden.
- Theoretisch interessant sind Heuristiken mit Gütegarantie (siehe folgendes Kapitel).

Abstände in Graphen

Definition 4.13

Es sei $G = (V, E)$ ein Graph. Der Abstand $d(v, w)$ zweier Knoten $v, w \in V$ ist die minimale Länge eines Weges von v nach w. Falls es keinen solchen Weg gibt, setzen wir $d(v, w) = \infty$.

Bemerkung:

- Die Länge eines Kantenzugs bzw. Weges ist definiert als die Anzahl der Kanten, die in diesem enthalten sind.
- Insbesondere gilt also $d(v, v) = 0$ für alle $v \in V$.

Berechnung von Abständen

Algorithmus 4.14

Es sei $G = (V, E)$ ein durch seine Adjazenzlisten A_v gegebener Graph und v_0 sei ein Knoten von G. Es werden die Abstände $d(v)$ von v_0 zu v für alle $v \in V$ berechnet.

$d(v_0) := 0; V(0) := \{v_0\}; k := 1;$
for all $v \in V, v \neq v_0$ *do* $d(v) := \infty;$
while $V(k-1) \neq \emptyset$ *do*
    $V(k) := \emptyset$
    for all $v \in \bigcup\{A_u | u \in V(k-1)\}$ *do*
        if $d(v) = \infty$ *then*
            $d(v) := k; V(k) := V(k) \cup \{v\}$
        end
    end
    $k := k + 1;$
end

Erläuterungen zu Algorithmus 4.14

- Ausgehend von $V(0) = \{v_0\}$ werden sukzessive die Mengen $V(1), V(2), \ldots$ berechnet.
- $V(k)$ besteht aus den Knoten, die mit Knoten aus $V(k-1)$ adjazent sind, aber nicht in $V(0) \cup \ldots \cup V(k-1)$ liegen.
- Mit Induktion zeigt man leicht, dass jedes $V(k)$ genau aus den Knoten besteht, die von v_0 den Abstand k haben.
- Dieses Vorgehen entspricht einer Breitensuche.
- Die Zeitkomplexität beträgt $O(|V| + |E|)$.

Abstände in Netzwerken

Wir ordnen nun jeder Kante eines Graphen eine Länge zu. Typisches Beispiel sind Straßennetze mit Entfernungsangaben.

Definition 4.15

Es sei $G = (V, E)$ ein Graph sowie $w : E \longrightarrow \mathbb{R}$ eine Bewertung der Kanten mit reellen Zahlen.

Das Paar (G, w) heißt Netzwerk. Für jede Kante $e \in E$ heißt $w(e)$ die Länge oder das Gewicht von e.

Für einen Kantenzug $K = (v_0, \ldots, v_k)$ ist $w(K) := \sum_{i=1}^{k} w(\{v_{i-1}, v_i\})$ die Länge von K.

Abstände in Netzwerken (2)

- Bei den Längen können auch negative Werte sinnvoll sein (z.B. Ertrag, Aufwand).
- Bei der Definition des Abstandes können solche negativen Längen aber zu Problemen führen.
- Wir setzen daher voraus, dass die Längen nicht negativ sind.

Definition 4.16

Es sei (G, w) ein Netzwerk, $G = (V, E)$ sowie $w(e) \geq 0$ für alle $e \in E$.

Der Abstand $d(v, w)$ zweier Knoten $v, w \in V$ in einem Netzwerk ist definiert als das Minimum der Längen aller Wege von v nach w. Falls es keinen solchen Weg gibt, setzen wir $d(v, w) = \infty$.

Abstände in Netzwerken (3)

Beispiel 4.17

In einem Netzwerk sei jeder Kante e eine Ausfallwahrscheinlichkeit $p(e)$ zugeordnet.

Setzt man voraus, dass Fehler unabhängig voneinander auftreten, dann ist

$$(1 - p(\{v_0, v_1\})) \cdots (1 - p(\{v_{k-1}, v_k\}))$$

die Wahrscheinlichkeit für das Funktionieren eines Kantenzuges $(v_0, \ldots, v_k)$.

Dieser Wert ist maximal, wenn $\sum_{i=1}^{k} \log(1 - p(\{v_{i-1}, v_i\}))$ maximal ist.

Setzt man $w(e) := -\log(1 - p(e))$, dann ist die zuverlässigste Verbindung zwischen zwei Knoten v und w äquivalent zu einem kürzesten Weg zwischen v und w.

Kürzeste Wege in Netzwerken

Algorithmus 4.18 (Dijkstra)

Es sei (G, w) ein Netzwerk mit $G = (V, E)$ und einer nichtnegativen Längenfunktion w auf E sowie $v_0 \in V$. Es werden alle Abstände $d(v)$ von v_0 zu einem Knoten $v \in V$ berechnet.

1. *Setze $d(v_0) := 0, d(v) := \infty$ für alle $v \in V \setminus \{v_0\}, U := V$.*
2. *Falls $U = \emptyset$, dann STOP. Sonst weiter mit 3.*
3. *Finde ein $u \in U$, für das $d(u)$ minimal ist.*
4. *Für alle $v \in U$ mit $\{u, v\} \in E$ setze*

$$d(v) := \min\{d(v), d(u) + w(\{u, v\})\}.$$

5. *Setze $U := U \setminus \{u\}$. Gehe zu 2.*

Eigenschaften des Dijkstra-Algorithmus

Bemerkungen:

- Der Algorithmus von Dijkstra kann analog für gerichtete Graphen verwendet werden.
- Der Algorithmus kann leicht so erweitert werden, dass nicht nur die Abstände $d(v_0, v)$ sondern auch die kürzesten Wege berechnet werden.

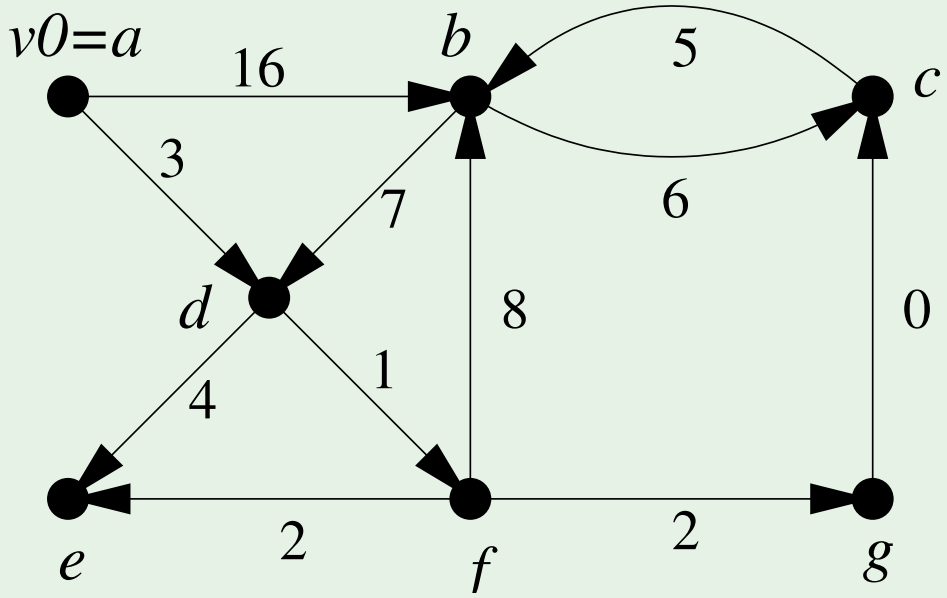
Satz 4.19

Nach der Terminierung von Algorithmus 4.18 gilt $d(v) = d(v_0, v)$ für alle $v \in V$.

Die Zeitkomplexität des Algorithmus ist $O(|V|^2)$.

Beispiel 4.20

Wir betrachten das folgende gerichtete Netzwerk:

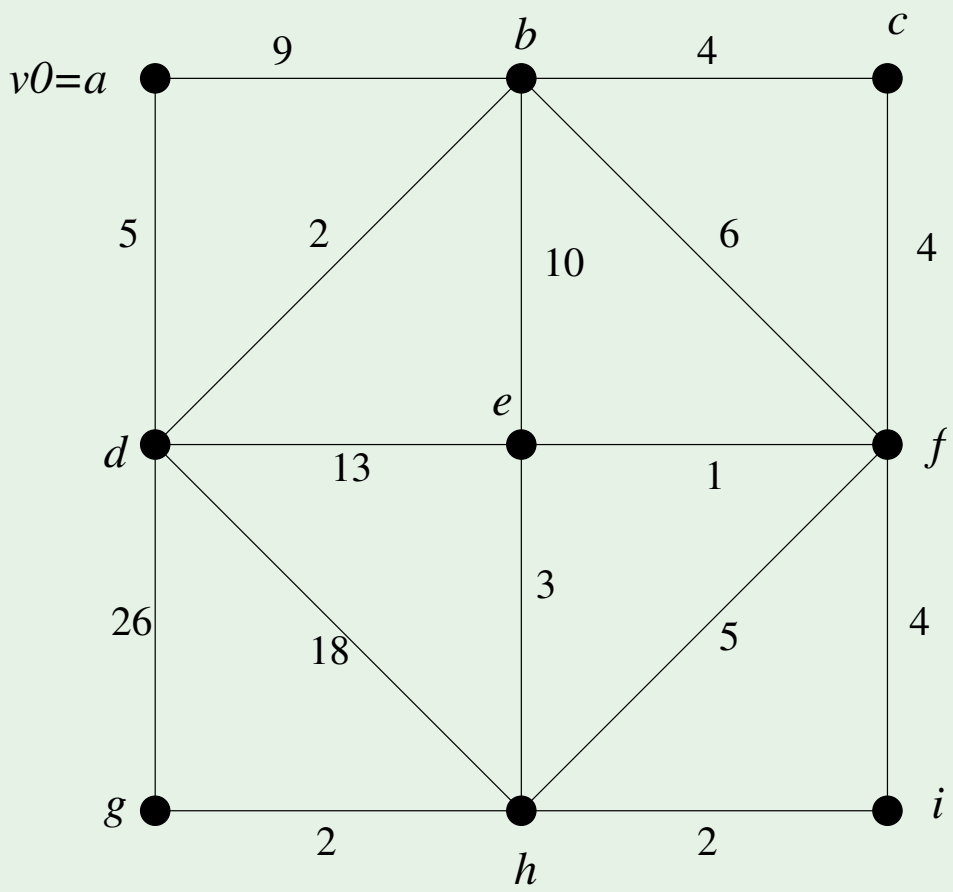


Algorithmus 4.18 liefert hierfür die folgenden Abstände:

v	a	d	f	e	g	c	b
$d(v)$	0	3	4	6	6	6	11

Beispiel 4.21

Wir betrachten das folgende nicht gerichtete Netzwerk:



Bemerkungen zum Dijkstra-Algorithmus

- In einem Netzwerk mit negativen Katenlängen werden durch den Dijkstra-Algorithmus u. U. falsche Abstände ermittelt
- und zwar selbst dann, wenn keine Kreise negativer Länge existieren.
- Durch die Wahl einer geeigneten Datenstruktur für die Selektion des Knotens u (Schritt 3) kann die Laufzeit erheblich verbessert werden.
- Für die Berechnung von Routen in Straßennetzen benötigt man optimierte Varianten des Dijkstra-Algorithmus.

Kürzeste Wege in DAGs

Da in DAGs keine Kreise auftreten, sind negative Kantenlängen unproblematisch.

Satz 4.22

Gegeben sei ein gerichtetes Netzwerk (G, w) mit $G = (V, A)$. Wir lassen auch $w(e) < 0$ zu. $L = (v_1, \ldots, v_n)$ sei eine topologische Sortierung von V. Dann ergibt sich die Länge $d(v_1, v_i)$ eines kürzesten Weges von v_1 zu einem Knoten v_i durch die folgende Rekursion für $d(v_i)$:

1. $d(v_1) = 0$
2. *Für $i > 1$:*

$$d(v_i) = \min_{\{1 \leq j < i | (v_j, v_i) \in A\}} w(v_j, v_i) + d(v_j)$$

Beispiel 4.23

Tafel ✎.

Kürzeste Wege in DAGs (2)

- Wenn man in Satz 4.22 min durch max ersetzt, dann erhält man die Länge eines längsten Weges.
- Das allgemeine Entscheidungsproblem, ob in einem (ungerichteten) Netzwerk ein (einfacher) Weg der Länge $\geq l$ existiert, ist dagegen $\mathcal{NP}$-vollständig.
- Die in Satz 4.22 angewendete Vorgehensweise, eine optimale Lösung durch eine bestmögliche Kombination der Lösungen zu Subproblemen zu finden, nennt man dynamisches Programmieren.
- Satz 4.22 läßt sich in einen Algorithmus mit Zeitkomplexität $O(|V| + |A|)$ umsetzen.

Beweis.

Mit Induktion über i. Tafel ✎.

Kürzeste Wege in DAGs (3)

- Mit dem Verfahren aus Satz 4.22 können nicht nur kürzeste und längste Wege sondern auch Anzahlen von Wegen berechnet werden.
- Z.B. die Anzahl $\#(v_i)$ der Wege von v_1 nach v_i.
- Hierfür muss der Minimum-Operator durch die Summenbildung

$$\#(v_i) = \sum_{\{1 \leq j < i \mid (v_j, v_i) \in A\}} \#(v_j)$$

ersetzt und die Initialisierung $\#(v_1) = 1$ verwendet werden.

Anwendungen

Für optimale Wege in DAGs gibt es eine Fülle von Anwendungen. Wir betrachten zwei:

- Optimale Einteilung in Blöcke (z.B. in der Textverarbeitung)
- Projektplanung

Blocksatzbildung

- Gegeben ist eine Folge $F = (1, \ldots, n)$ von n Gegenständen.
- Jeder Gegenstand hat eine Länge $l(i)$.
- Die Gegenstände sollen in Blöcke eingeteilt werden, wobei die Idealgröße eines Blockes B ist.
- Die Gesamtlänge der Gegenstände in einem Block darf B nicht übersteigen.
- Die Gegenstände dürfen nicht umsortiert werden. Ein Block B_j ergibt sich also durch einen Index i_j mit
  - $i_{j-1} + 1$ ist das erste Element im Block j und
  - i_j ist das letzte Element im Block B_j.
- Für die Abweichung der Länge

$$L_j := \sum_{r=i_{j-1}+1}^{i_j} l(r)$$

eines Blockes B_j von B wird eine Bewertung definiert, die monoton in der Größe der Abweichung ist, z.B. $(B - L_j)^2$.

- Gesucht ist eine Blockung, die eine möglichst kleine Bewertung hat, also eine Anzahl k an Blöcken und eine Folge $0 = i_0 < i_1 < i_2 < \cdots < i_k = n$ mit

$$\sum_{j=1}^{k} (B - L_j)^2 \longrightarrow \min$$

unter den Bedingungen

$$L_j = \sum_{r=i_{j-1}+1}^{i_j} l(r) \leq B \text{ für } j = 1, \ldots, k$$

Beispiel 4.24

- Länge der Gegenstände: $9, 1, 5, 5, 1, 10$
- Blockgröße: 10
- "gierige" Einteilung: $[9, 1], [5, 5], [1], [10]$, Bewertung: 81
- andere Einteilung: $[9], [1, 5], [5, 1], [10]$, Bewertung: 33

Modellierung des Problems als Wegeproblem

- $V := \{0, \ldots, n\}$
- $E := \{(i,j) | i < j \text{ und } \sum_{r=i+1}^{j} l(r) \leq B\}$
- Gewichte: $w(i,j) := (B - \sum_{r=i+1}^{j} l(r))^2$
- Jeder Weg von 0 nach n definiert dann eine zulässige Einteilung in Blöcke, wobei die Bewertung dieser Blockung gleich der Länge des Weges ist.
- Also: Finde einen Weg minimaler Länge von 0 nach n.

☞ Auf diesem Prinzip basiert die Formatierung in TeX.

Projektplanung mit Netzplantechnik

- Bei der Durchführung umfangreicher Projekte ist es erforderlich, dass einzelne Teilaufgaben (Jobs) zeitlich genau aufeinander abgestimmt werden.
- Hierfür werden häufig Methoden der Netzplantechnik eingesetzt.
- Die bekannteste Methode der Netzplantechnik heißt CPM (Critical Path Method).
- J sei die Menge der Jobs eines Projekts. Jeder Job $j \in J$ hat eine zugeordnete Dauer $t(j)$ sowie eine Menge $P(j) \subseteq J$ von Vorgängern.
- Ein Job j kann erst dann gestartet werden, wenn alle Jobs aus $P(j)$ beendet wurden.
- Wir gehen davon aus, dass Jobs parallel bearbeitet werden können.
- Man möchte z.B. wissen:
  - Ab welchem Zeitpunkt kann mit einem Job begonnen werden?
  - Wie lange wird ein Projekt dauern?
  - Welche Jobs sind besonders kritisch in bezug auf die Gesamtdauer eines Projekts?

Beispielprojekt

Beispiel 4.25

Zusammenbau eines Fahrrads:

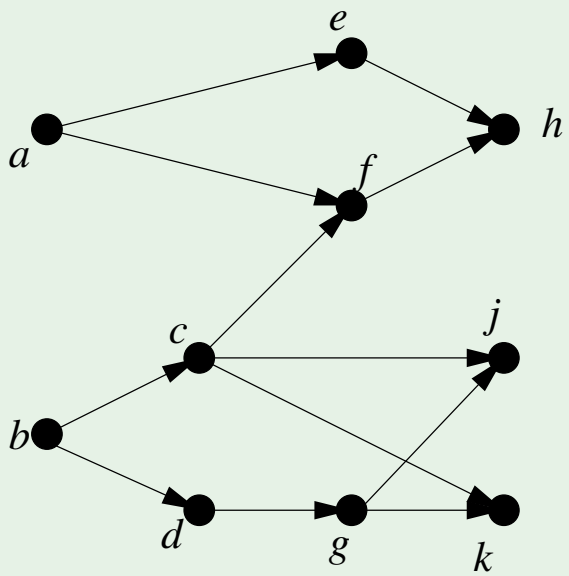
Job	Beschreibung	Dauer	Vorg.
a	Rahmen vorbereiten (Gabel, Schutzbleche, usw.)	6	–
b	Kettenführung anbringen	2	–
c	Gangschaltung anbringen	3	b
d	Kettenblatt an Kurbel montieren	4	b
e	Vorderrad montieren und anpassen	6	a
f	Hinterrad montieren und anpassen	6	a,c
g	Kurbel am Rahmen anbringen	3	d
h	Endmontage (Lenker, Sattel, usw.)	12	e,f
j	Linkel Pedal anbringen	3	c,g
k	Rechtes Pedal anbringen	3	c,g

Modellierung

- Die Abhängigkeiten der Jobs untereinander können wir in Form eines DAGs repräsentieren.
- Die Knoten stellen hierbei die Jobs dar.
- Die Kanten stellen Vorgängerbeziehungen zwischen den Jobs dar.

Beispiel 4.26

Zusammenbau eines Fahrrads:



Fortsetzung Beispiel.

- Die Jobs a und b können offensichtlich sofort begonnen werden.
- Der Job f kann begonnen werden, wenn a beendet ist und wenn c beendet ist.
- a ist nach einer Dauer von 6 beendet, c nach einer Dauer von 5, da c erst gestartet werden kann, wenn b beendet ist.
- Somit kann f frühestens zum Zeitpunkt 6 begonnen werden.
- Allgemein: Der frühste Starttermin eines Jobs ergibt sich durch einen Weg zu diesem Job mit maximaler Gesamtdauer.
- Hier liegen die Bewertungen aber auf den Knoten.

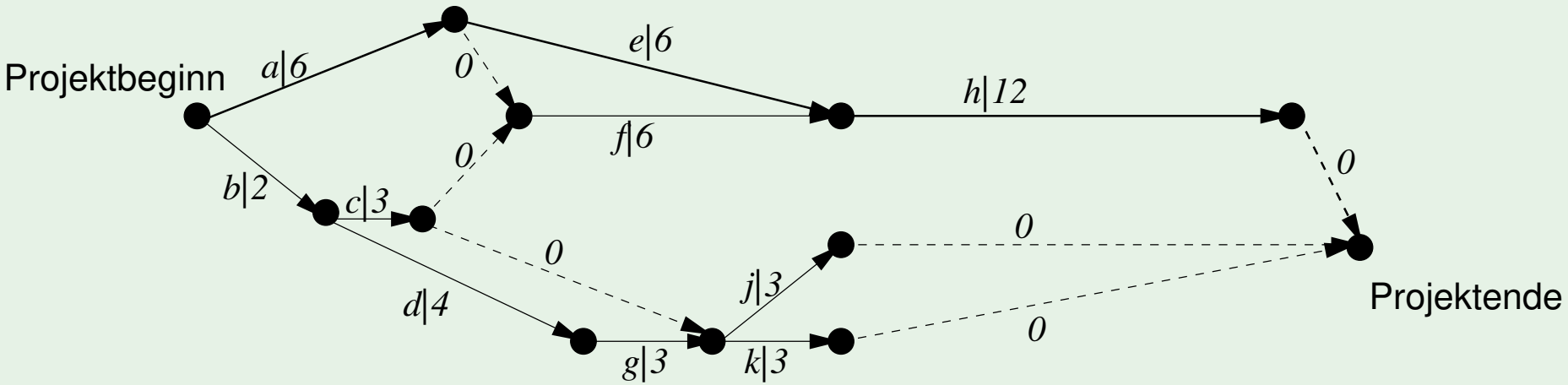
Critical-Path-Method

Die Critical-Path-Method (CPM) stellt eine Modellierung dar, bei der die Jobs die Kanten eines DAGs sind.

Die Knoten können als Fertigstellungsereignisse angesehen werden.

Die Kanten a, e, h stellen einen längsten Weg vom Projektbeginn zum -ende dar (kritische Pfad).

Beispiel 4.27



Die Gesamtlaufzeit des Beispielprojekts ist also 24.

Projektplanung mit CPM

Für die Projektplanung sind u.a. die folgenden interessant:

- Projektdauer D: Länge eines längsten Weges (kritischer Pfad) vom Projektbeginn bis zum Projektende.
- Starttermin für Jobs j: Länge eines längsten Weges bis zum Startknoten von j.
- Zeitkritische Jobs: Alle Jobs, die auf einem kritischen Pfad liegen.
- Pufferzeit für Job j: $D-$Länge eines längsten Weges, der Kante j enthält.
- Und warum verzögern sich Projekte trotz ausgefeilter Planungsmethoden? Siehe Übungen.

Zusammenfassung des Kapitels

- Charakterisierung und Berechnung von Eulerwegen (Hierholzer-Algorithmus)
- Hamiltonsche Wege und Kreise (Berechnung ist schwer)
- Dijkstra-Algorithmus zur Ermittlung kürzester Wege
- Dynamische Programmierung zur Berechnung kürzester Wege in DAGs
- Anwendungen: Blocksatzbildung und Netzplantechnik