DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart
Campus Horb

Algorithmen und Datenstrukturen

Teil 6:
Algorithmen auf Graphen (II) – Färbungen, kürzeste Wege, Euler/Hamilton

DHBW Stuttgart Campus Horb
Fakultät Technik
Studiengang Informatik
Dozent: Olaf Herden
Stand: 05/2020

Gliederung

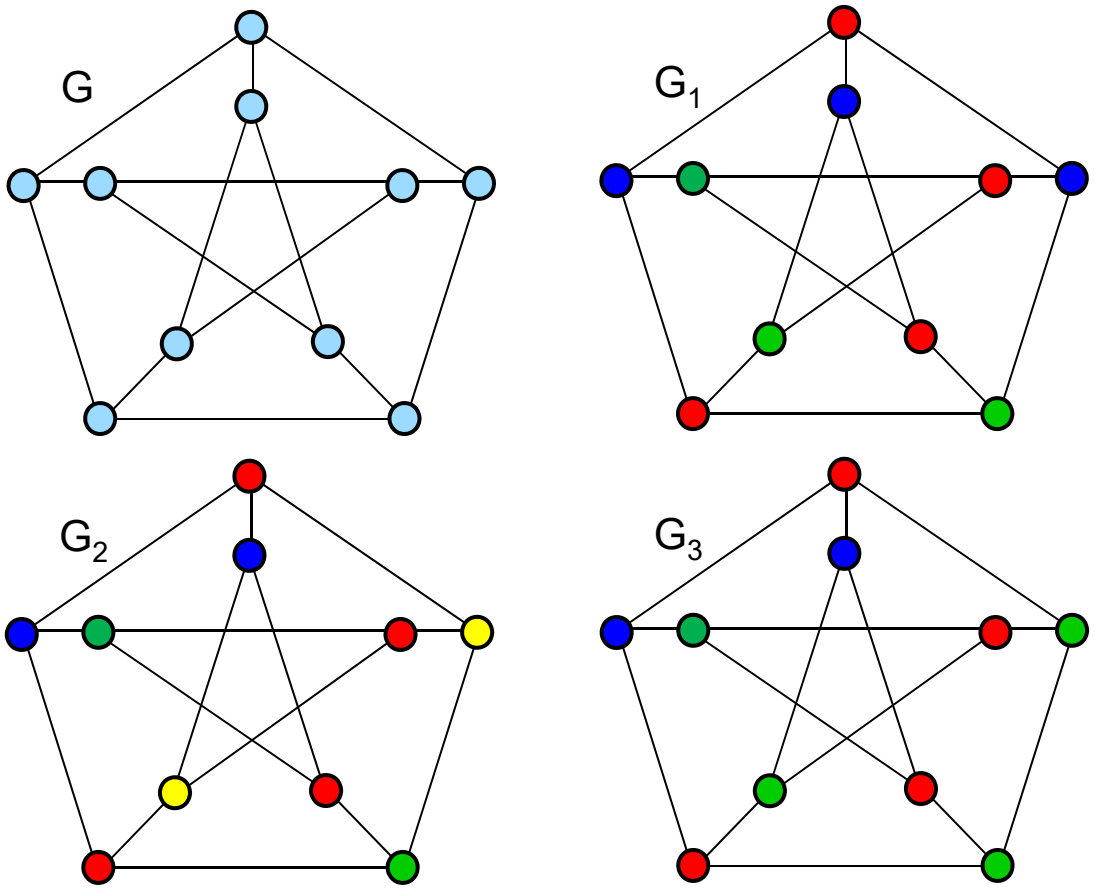
- Färbungen
- Kürzeste Wege
- Euler- und Hamilton-Kreise

Färbungsproblem (I)

- Def.: Färbung eines (ungerichteten) Graphen G:
  - Zuordnung von Farben zu Knoten von G
  - Zwei benachbarte Knoten dürfen nie gleiche Farbe haben
- Positive Ganzzahl m gibt Anzahl Farben an (m-Färbung bzw. m-färbbar))
- Def.:

  Sei G Graph. Chromatische Zahl *chromatic*(G) (oder χ(G)) kleinste Zahl m, für die m-Färbung existiert
- Färbung mit genau *chromatic*(G) Farben heißt minimale Färbung
- Färbung mit |V| Farben heißt triviale Färbung

Färbungsproblem (II): Beispiele



- G_1 und G_2 Färbungen von G, G_3 keine Färbung von G
- G_1 3-Färbung, G_2 4-Färbung
- *chromatic*(G) = 3
- G_1 ist eine minimale Färbung von G

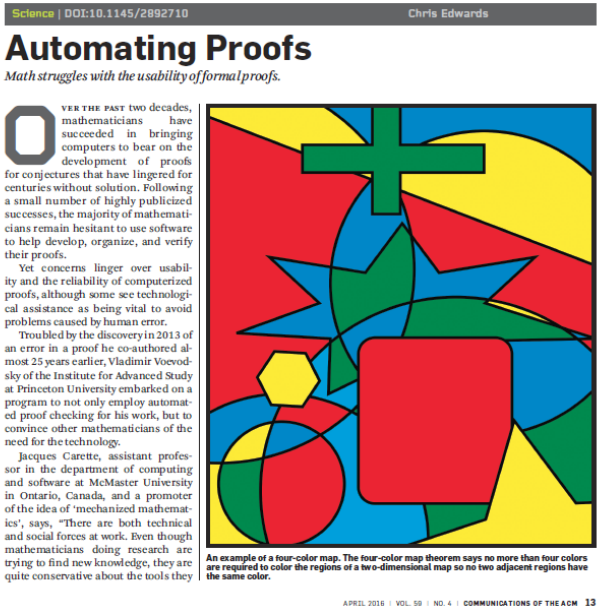
Färbungsproblem (III): Versionen

- Entscheidungsproblem:
  - Gegeben ungerichteter Graph G und positive Ganzzahl m
  - Ist G mit m Farben färbbar?

- Optimierungsproblem:
  - Gegeben ungerichteter Graph G
  - Wie viele Farben werden mindestens für die Färbung benötigt? (= Finde *chromatic*(G))

Färbungsproblem (IV): Geschichte

- Färbungsproblem eines der ältesten bekannten Probleme der Graphentheorie
- Vermutung seit 19. Jhdt.: Jeder planare Graph ist 4-färbbar
- Endgültiger Beweis erst 1976
- Computer(unterstütze) Vorgehensweise [z.B. Ed16]:



Science | DOI:10.1145/2892710 Chris Edwards

Automating Proofs

Math struggles with the usability of formal proofs.

OVER THE PAST two decades, mathematicians have succeeded in bringing computers to bear on the development of proofs for conjectures that have lingered for centuries without solution. Following a small number of highly publicized successes, the majority of mathematicians remain hesitant to use software to help develop, organize, and verify their proofs.

Yet concerns linger over usability and the reliability of computerized proofs, although some see technological assistance as being vital to avoid problems caused by human error.

Troubled by the discovery in 2013 of an error in a proof he co-authored almost 25 years earlier, Vladimir Voevodsky of the Institute for Advanced Study at Princeton University embarked on a program to not only employ automated proof checking for his work, but to convince other mathematicians of the need for the technology.

Jacques Carette, assistant professor in the department of computing and software at McMaster University in Ontario, Canada, and a promoter of the idea of 'mechanized mathematics', says, "There are both technical and social forces at work. Even though mathematicians doing research are trying to find new knowledge, they are quite conservative about the tools they

An example of a four-color map. The four-color map theorem says no more than four colors are required to color the regions of a two-dimensional map so no two adjacent regions have the same color.

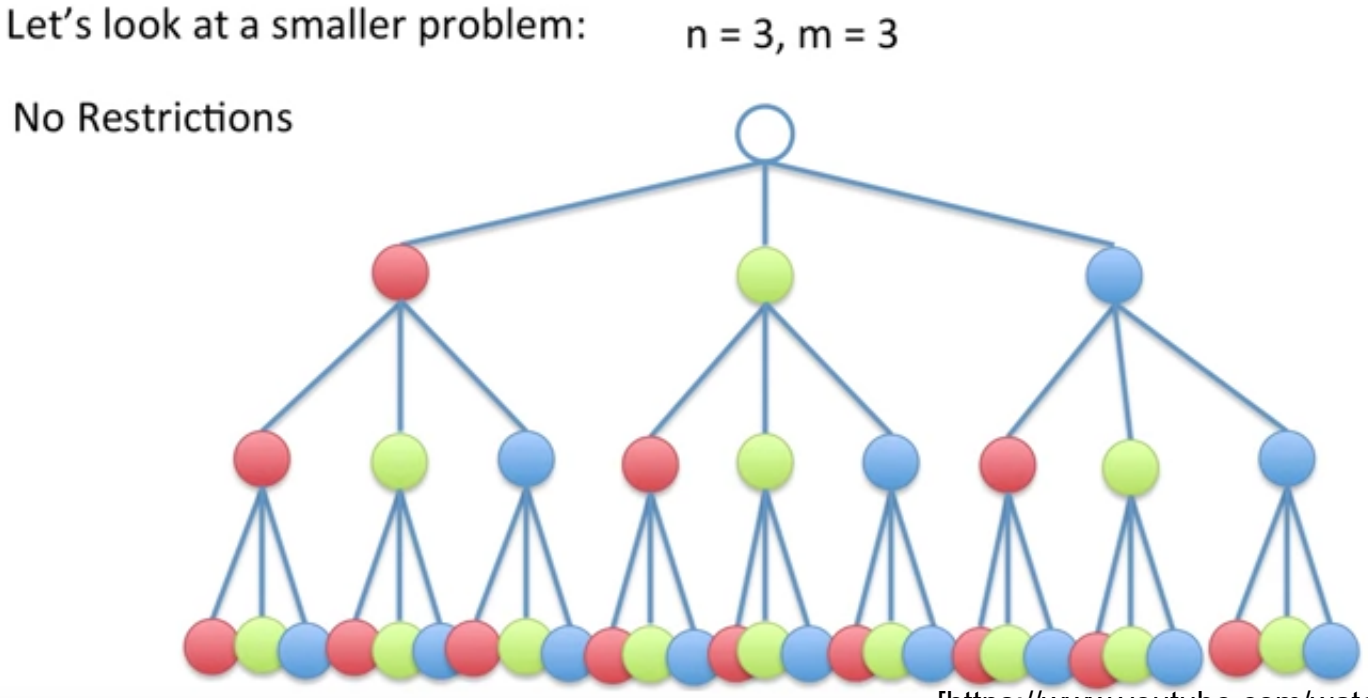
APRIL 2016 | VOL. 59 | NO. 4 | COMMUNICATIONS OF THE ACM 13

[Chris Edwards: Automating Proofs. Commun. ACM 59(4): 13-15 (2016)]

Similar issues of trust greeted the first computerized proof of the theorem that argued only four colors are needed to distinguish between adjacent regions on a 2D map. Forty years ago, working at the University of Illinois at Urbana-Champaign, Kenneth Appel and Wolfgang Haken developed computer programs to demonstrate there were no counterexamples to the theorem. Still, the programs were tedious to check by hand.

Backtracking-Algorithmus (I)

- Prinzip:
  - Systematisches Durchsuchen gesamter Lösungsraum
- Beispiel:
  - Sei n Anzahl der Knoten und m Anzahl der Farben
  - Sei n = 3, m = 3
  - Lösungsraum ohne Restriktionen:



[https://www.youtube.com/watch?v=miCYGGrTwFU]

Backtracking-Algorithmus (II)

- Algorithmus:

```
( 1) graphColour(int k){
( 2)   for c=1 to m{
( 3)     if(isSafe(k,c)){
( 4)       x[k] = c
( 5)       if((k+1)<n){
( 6)         graphColour(k+1)}
( 7)       else{
( 8)         print x[]
( 9)         return
(10)       }
(11)     }
(12)   }
(13) }
```

  - m: Anzahl Farben
  - k: In aktuellem Rekursionslevel zu färbender Knoten
  - x[k]: Feld mit aktuellen Farben für alle Knoten
  - Methode `isSafe` prüft, ob Knoten k Farbe c zugewiesen werden kann

[https://www.youtube.com/watch?v=miCYGGrTwFU]

Backtracking-Algorithmus (III)

- Methode `isSafe`:

```
( 1) boolean isSafe(int k, int c){
( 2)   for i=1 to n{
( 3)     if(areAdjacent(k,i) && x[i]==c){
( 4)       return false;
( 5)     }
( 6)   }
( 7)   return true;
( 8) }
```

[https://www.youtube.com/watch?v=miCYGGrTwFU]

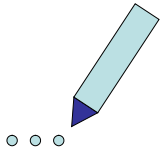
Backtracking-Algorithmus (IV)

- Algorithmus: Siehe Codebeispiel (https://www.geeksforgeeks.org/backttracking-set-5-m-coloring-problem/)
- Experiment 1:
  - Laufzeitmessung für K_n
  - Anzahl Farben n-1

n	Laufzeit [HH:MM:SS]
1..8	0
9	00:00:00,016
10	00:00:00,107
11	00:00:01,164
12	00:00:13,725
13	00:03:00,390
14	>12 Stunden
15	

Backtracking-Algorithmus (V)

- Experiment 2:
  - Laufzeitmessung für K_n
  - Anzahl Farben n
  - Laufzeit für n=100 wenige Millisekunden
  - Warum?



Backtracking-Algorithmus (VI)

- Analyse:
  - An jedem inneren Knoten: O(n·m) Zeit zur Bestimmung der Kindknoten für legale Färbung
  - Damit totale Zeit:

$$\sum_{i=0}^{n-1} m^{i+1} \cdot n = \sum_{i=1}^{n} m^{i} \cdot n = \frac{n(m^{n+1} - 2)}{(m-1)} = O(n \cdot m^{n})$$

- Also:
  - Exponentielles Wachstum Zeitkomplexität mit Anzahl Knoten
  - Suche nach Alternativen notwendig

Greedy-Färbe-Algorithmus (I): Prinzip

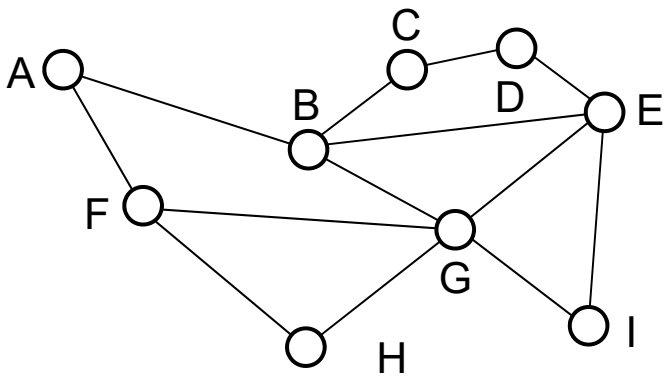
- farben = {1,2,...} stehen zur Verfügung
- Pro Durchgang:
  - Vergabe einer Farbe
  - Behandlung aller Knoten
- Wurde Knoten k gefärbt, dann:
  - Kein Nachbar von k kann aktuelle Farbe bekommen
  - Kennzeichnung entsprechender Knoten mit negativem Wert (-farbe)
- Wurde Knoten k noch nicht gefärbt und ist nicht –farbe, wird k mit aktueller Farbe markiert
- Letzter Punkt: Greedy-Schritt („Färbe alles, was Du kriegen kannst.“)

Greedy-Färbe-Algorithmus (II): Pseudocode

```
( 1) farbe=0;          // Aktuelle Farbe
( 2) z=0;              // Anzahl bereits markierter Knoten
( 3) f[1..n]=0;        // f[i] : Farbe des i-ten Knotens
( 4) while (z < n){
( 5)   farbe++;
( 6)   for „Jeden Knoten i“ {
( 7)     if (f[i]==0 && f[i]!=-farbe){
( 8)       f[i]=farbe;
( 9)       z++;
(10)       for „Jeden Nachbarn von i“{
(11)          if (f[j]==0){
(12)            f[j]=-farbe;
(13)          }
(14)       }
(15)     }
(16)   }
(17)   Setze alle f[i] mit f[i]=-farbe auf f[i]=0;
(18) }
```

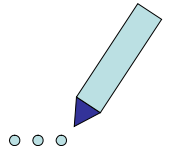
Greedy-Färbe-Algorithmus (III): Beispiel

- Graph



soll mit den Farben = {rot (1), gelb (2), blau(3), grün(4), ... } gefärbt werden

- Knoten sollen dabei in der Reihenfolge ihrer Namen durchlaufen werden



Laufzeit

- Straight-Forward-Implementierung: $O(n^2)$
- Verbesserung:
  - Verwalte für jeden Knoten i kleinste unbenutzte Farbnummer der Nachbarn
  - Kann mit Aufwand O(degree(i)) realisiert werden
  - Idee:
    - Feld `vergeben` der Länge Δ +1 (Δ max. Grad eines Knoten)
    - In `vergeben` werden in jedem Durchgang die schon an Nachbarn vergebenen Farben eingetragen (vergeben[i] = 0, wenn Farbe i schon vergeben, sonst 1)
    - Feld vergeben wird nicht in jedem Durchgang zurückgesetzt (wäre Aufwand $O(\Delta \cdot n)$)
    - Vielmehr: In jedem Durchgang zunächst vergebene Farben markieren und später rückgängig machen (Aufwand O(degree(i))
    - Bestimmung kleinster nicht vergebener Farbnummer (d.h. Suche nach der ersten 0 im Feld vergeben) hat gleichen Aufwand
    - Insgesamt: O (n+m)
- Also: Kleine Änderung am Algorithmus führt zu verbesserter Laufzeit

Verbesserter Färbe-Algorithmus

```
( 1) farbe=0;      // Aktuelle Farbe
( 2) c=0;          // Anzahl benötigter Farben
( 3) f[1..n]=0;    // f[i] : Farbe des i-ten Knotens
( 4) v[1..Δ+1]=0; // v[i] : Farbe i vergeben ja(1), nein(0)
( 5) f[1]=1;
( 6) for „Jeden Knoten i außer 1“{
( 7)   for „Jeden Nachbarn j von i“{
( 8)     if (f[j]>0){
( 9)       v[f[j]]=1;
(10)     }
(11)  }
(12)     farbe=Minimum {i mit v[i]=0}
(13)     f[i]=farbe;
(14)     if (farbe>c){
(15)       c=farbe;
(16)     }
(17)     for „Jeden Nachbarn j von i“{
(18)       if (f[j]>0){
(19)            v[f[j]]=0;
(20)       }
(21)     }
(22) }
```

Fazit

- Keine Rücknahme einmal getroffener Entscheidungen (Greedy-Strategie)
- Lösung:
  - Immer korrekt
  - Nicht immer optimal, d.h. Färbung nicht unbedingt minimal
  - Güte der Lösung stark abhängig von Nummerierung der Knoten
- Es gilt: Für einen Graphen G mit *chromatic*(G) = m existiert mindestens eine Knoten-Nummerierung, so dass Greedy-Algorithmus m-Färbung (d.h. minimale Färbung) liefert

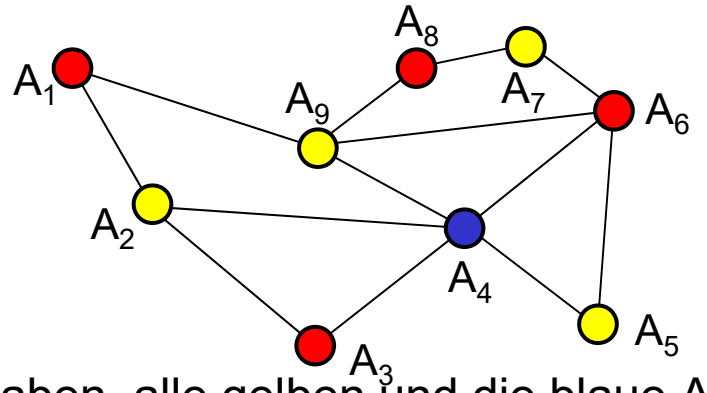
Anwendungen (I)

- Färbung einer Landkarte (siehe auch Übung)
- Maschinenbelegung:
  - Aufgaben A_1,..., A_n auf verschiedene Maschinen durchführen
  - Durchführung von A_i erfordert jeweils Zeiteinheit T
  - Gleichzeitige Durchführung von Aufgaben, wenn nicht gleiche Maschine benötigt
  - Ziel der Maschinenbelegung:
    - Finden Aufgabenreihenfolge, die diese Einschränkung beachtet und zeitoptimal ist (d.h. möglichst hoher Parallelitätsgrad wird angestrebt)
  - Als Graph:
    - Jeder Knoten entspricht einer Aufgabe
    - Knoten benachbart, wenn Aufgaben nicht gleichzeitig durchführbar (Konfliktgraph)
  - Färbung des Konfliktgraphen liefert Aufgabenreihenfolge (Aufgaben gleicher Farbe zur gleichen Zeit durchführen)
  - Optimale Lösung entspricht minimaler Färbung
  - Benötigte Zeit ist Produkt aus T und der chromatischen Zahl

Anwendungen (II)

- Maschinenbelegung (Beispiel)
- Ausführung von neun Aufgaben $A_1,\ldots, A_9$ auf acht Maschinen $M_1, \ldots, M_8$:

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
A_1,A_2	A_1,A_9	A_4,A_5,A_6	A_2,A_3,A_4	A_4,A_6,A_9	A_8,A_9	A_7,A_8	A_6,A_7



- Alle roten Aufgaben, alle gelben und die blaue Aufgabe können jeweils in einem Zeitslot bearbeitet werden:

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
T_1	A_1	A_1	A_6	A_3	A_6	A_8	A_8	A_6
T_2	A_2	A_9	A_5	A_2	A_9	A_9	A_7	A_7
T_3	--	--	A_4	A_4	A_4	--	--	--

Übersicht

- Färbungen
- Kürzeste Wege
- Euler- und Hamilton-Kreise

Problem

- Gegeben:
  - Kantenbewerteter Graph G = (V, E, g)
  - v_S, $v_Z \in V$ als Start- bzw. Zielknoten
- Def. Kürzester Weg:
  - Weg von v_S nach v_Z mit Kantensumme kleiner gleich jeder andere Weg von v_S nach v_Z
- Ausprägungen:
  - 1-1 : Kürzester Weg von einem Knoten v_S zu einem v_Z
  - 1-many : Kürzester Weg von einem Knoten v_S zu allen $v_Z \in V \setminus \{v_S\}$
  - many-many : Kürzester Weg von allen $v_S \in V$ zu allen $v_Z \in V \setminus \{v_S\}$

Dijkstra-Algorithmus (I): Idee

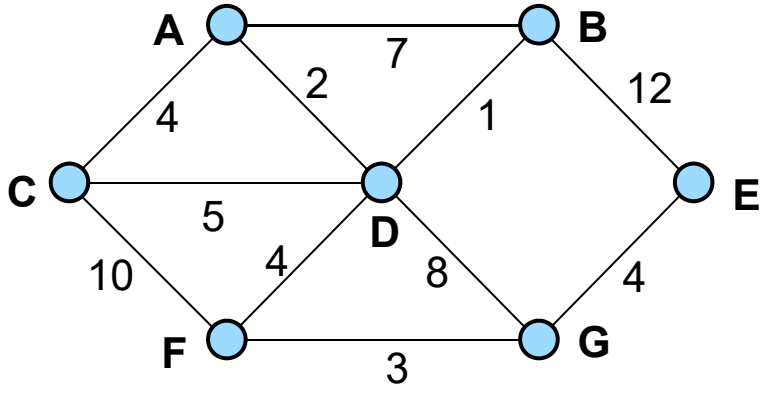
- Gegeben:
  - Kantenbewerteter Graph G = (V, E, g), G gerichtet, zusammenhängend
  - Jede Kantenbewertung sei positiv
  - v_S, $v_Z \in V$ als Start- bzw. Zielknoten
- Prinzip:
  - Aufbau eines Wurzelbaums G‘ = (V‘,E‘,g‘) mit folgenden Eigenschaften:
    - $V' \subseteq V$
    - v_S ist Wurzel von G‘
    - $(v,v') \in E' \Rightarrow (v,v') \in E$ und g‘((v,v‘)) = g((v,v‘))
    - $\forall\ v \in V'$: dist(x,v) stimmt mit dem minimalen Weg von x nach v in G überein

Dijkstra-Algorithmus (II)

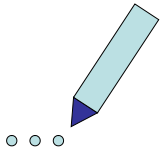
- Ablauf:
  (1) Füge den Startknoten v_S in V‘ ein
  (2) Bestimme die Menge der offenen Kanten (Teilmenge von E, die einen Startknoten v $\in$ E‘ und einen Zielknoten v‘ $\in$ V\V‘ haben)
  (3) Wähle eine offene Kante (v,v‘), für die gilt: dist(v_S,v)+g(v,v‘) ist minimal
  (4) Füge Knoten v‘ zu V‘ und Kante (v,v‘) zu E‘ hinzu
  (5) Wiederhole die Schritte (2), (3) und (4), bis G‘ den gesuchten Knoten v_Z enthält
  (6) Bestimme in G‘ den eindeutigen Weg von v_S nach v_Z
- Aufwand: O(n²)
  - n Durchgänge sind im worst case notwendig
  - In jedem Durchgang sind die offenen Kanten zu bestimmen:
    - Dies sind im worst case n-1 im ersten Durchgang, n-2 im zweiten, n-3 im dritten usw.
    - Auch dies entspricht O(n)

Dijkstra-Algorithmus (III): Beispiel

- Gegeben sei der folgende Graph G:



- Anmerkung:
  - G soll gerichtet sein, die Bewertungen gelten für Hin- und Rückrichtung
  - Einzeichnen von Hin- und Rückkante wäre zu unübersichtlich
  - Annahme ist realistisch (z.B. Entfernungen auf Straßen oder Fahrtzeiten von Zügen)
- Gesucht: Kürzester Weg von C nach E



Dijkstra-Algorithmus (IV): Alle Wege berechnen

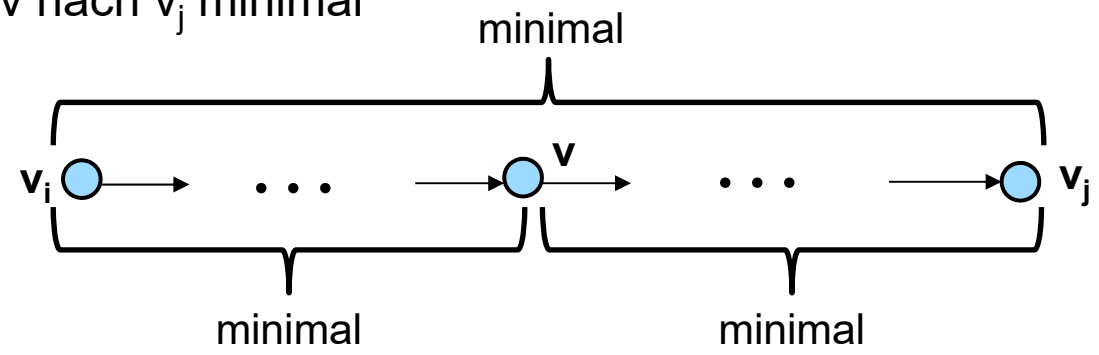
- Alle Wege := Von jedem Knoten v_1 zu jedem Knoten v_2 für alle v_1 , v_2 aus G
- Algorithmus modifizieren:
  - Starte nacheinander mit jedem Knoten als Startknoten
  - Terminierungskriterium: Statt Erreichen eines bestimmten Zielknotens Erreichen jeden Knotens

Floyd-Algorithmus (I): Idee

- Berechnung kürzester Distanz zwischen allen Knotenpaaren (v_i, v_j)
- Annahmen:
  - Jede Kante ist mit Distanzwert Wert dist >=0 markiert
  - Für alle i: dist (v_i, v_i) = 0
  - Existiert keine Kante (v_i, v_j) => dist (v_i, v_j) = ∞
  - Rechenregeln:
    - x + ∞ := ∞
    - ∞ + ∞ := ∞
    - min(x, ∞) := x
- Definiere Distanzmatrix D:
  - Hauptdiagonale in D ist 0
  - Bei direkter Kante trage dist (v_i, v_j) ein
  - Alle anderen Distanzen werden als ∞ angenommen

Floyd-Algorithmus (II): Prinzip (I)

- Prinzip: Dynamische Programmierung
  - Optimierungsproblem in Teilprobleme aufteilen
  - Systematische Speicherung von Zwischenresultaten
- Anwendung:
  - Führt kürzester Weg von v_i nach v_j durch v, dann sind auch Teilpfade v_i nach v und v nach v_j minimal

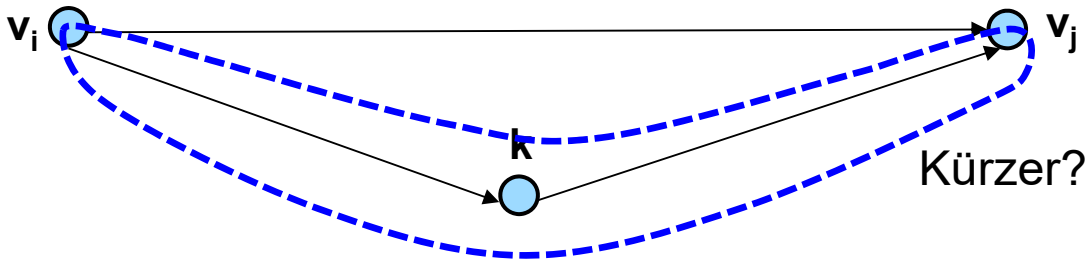


  - Kennt man alle kürzesten Wege zwischen allen Knotenpaaren mit Index kleiner k und sucht kürzeste Wege über alle Knotenpaare mit Index höchstens k, dann gibt es für Pfad von v_i nach v_j zwei Möglichkeiten:
    - Geht über k, dann Zusammensetzung aus bekannten Pfaden v_i nach k und k nach v_j
    - Geht nicht über k, dann ist es der bekannte Pfad von v_i nach v_j

Floyd-Algorithmus (III): Prinzip (II)

- Mit anderen Worten:

  In jeder Iteration wird Distanz von v_i nach v_j genommen und überprüft, ob Weg über v_k eventuell kürzer ist



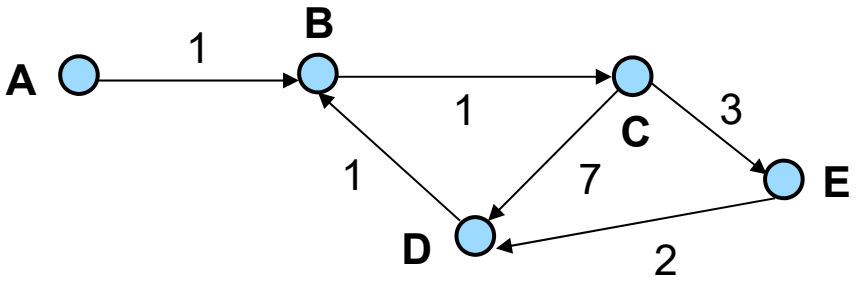
- Folgender Kernalgorithmus:

```
(1) for all k ∈ nodes(G)
(2)   for all i ∈ nodes(G)
(3)     for all j ∈ nodes(G)
(4)         D[i,j] = min(D[i,j],(D[i,k]+D[k,j]));
```

- Komplexität: $O(n^3)$

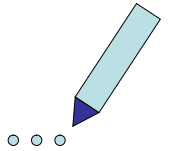
Floyd-Algorithmus (IV): Beispiel

- Finde alle kürzesten Wege in folgendem Graphen:



- Hinweis:
  - Protokollieren Sie die Schritte am besten in einer Tabelle

k	i	j	D[i,j]	D[i,k]+D[k,j]	Minimum	Änderung in D
A	A	A	0	0 + 0 = 0	0	---
A	A	B	1	0 + 1 = 1	1	---
A	A	C	∞	0 + ∞ = ∞	∞	---
…	…	…	…	…	…	…



Übersicht

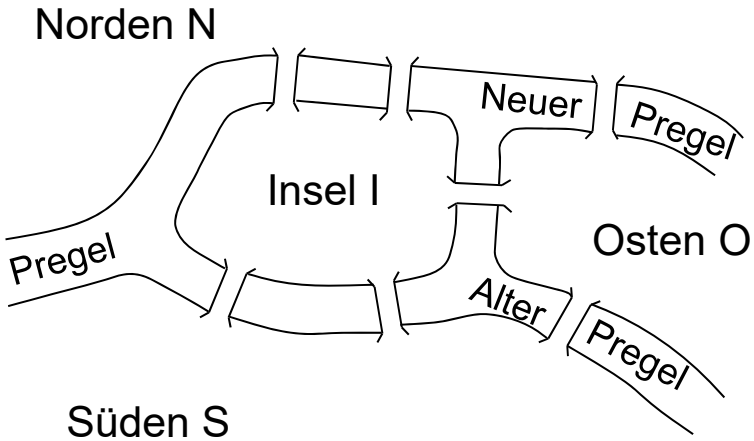
- Färbungen
- Kürzeste Wege
- Euler- und Hamilton-Kreise

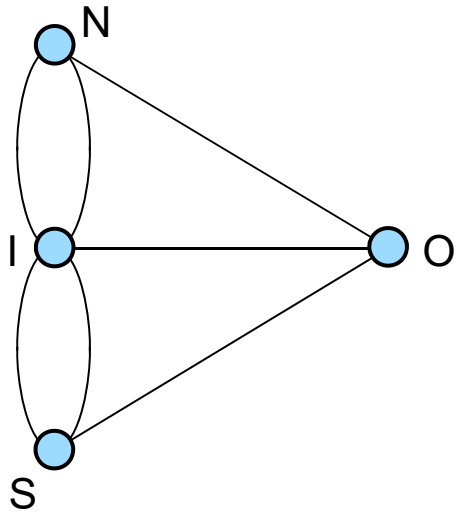
Königsberger Brückenproblem (I)

- Leonhard Euler (1707 - 1782)
- Bedeutender schweizer Mathematiker
- Arbeitete lange in Königsberg
- Stellte sich die Frage, ob es in Königsberg einen Weg gibt, so dass jede Brücke nur genau einmal überquert wird und man zum Ausgangspunkt zurückkehrt?
- Fragestellung hat große historische Bedeutung
- Gilt als Begründung der modernen Graphentheorie

Königsberger Brückenproblem (II)

- Modellierung als Graph:
  - Jeder Stadtteil Knoten
  - Jeder Weg über eine Brücke Kante





- Frage: Gibt es zu jedem Knoten einen Kantenzug, der jede Kante genau einmal enthält und wieder am Ausgangsknoten ankommt?
- Ein solcher Kantenzug wird als Eulerscher Kreis bezeichnet
- Graph mit dieser Eigenschaft heißt eulersch

Test auf Eulerschen Kreis

- Euler fand zwar nicht in Königsberg einen solchen Weg, aber folgenden allgemeinen Satz: Graph G ist genau dann eulersch, wenn der Grad eines jeden Knoten von G gerade ist.
- Gegeben:
  - Ungerichteter Graph G = (V,E)
  - Feld f der Dimension |V|, das die Gradzahlen der Knoten enthält
- Algorithmus:

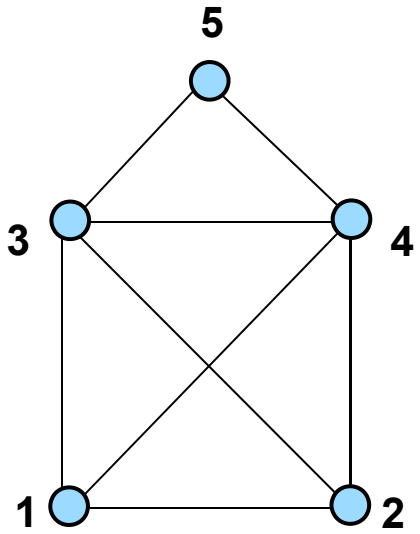
  (1) Setze f[i]=0 für alle i $\in$ {1, ,,,, n}

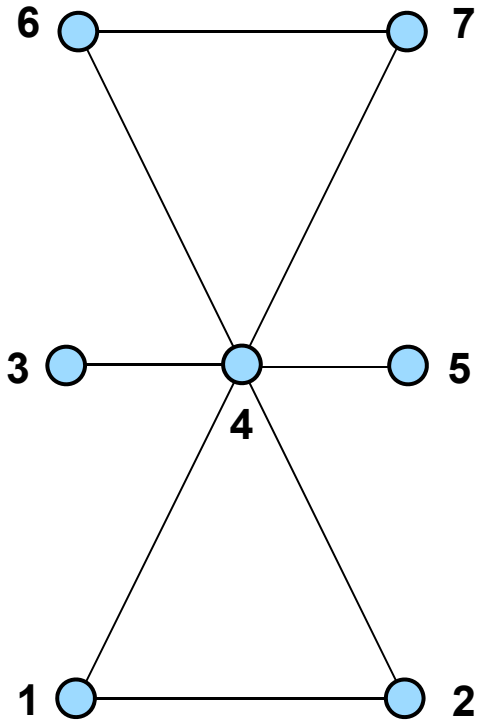
  (2) Gehe alle (v,v‘) $\in$ E durch:
    - Erhöhe f[v] und f[v‘] jeweils um 1

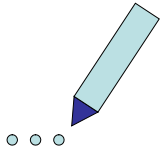
  (3) Wenn f[i] ist gerade für alle i $\in$ {1, ,,,, n} $\Rightarrow$ G eulersch
- Komplexität: O(n+m)
  - In Schritt (1) und (3) sind n Schritte notwendig
  - In Schritt (2) sind m Schritte notwendig

Eulerweg

- Eulerweg (synonym: offener Eulerzug) Kantenfolge, bei der jede Kante genau einmal durchlaufen wird
- Start- und Endknoten nicht identisch
- Graph mit Eulerweg heißt semi-eulersch
- Beispiele:

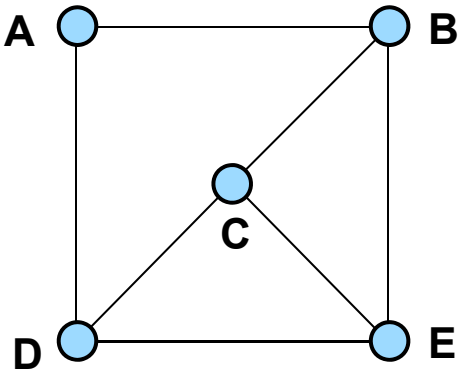






Hamiltonsches Problem (I)

- Sir William Hamilton (1805-1865) formulierte 1859 ein sehr ähnliches Problem
- Gibt es für einen vorgegebenen Graphen einen geschlossenen Weg, der jeden Knoten genau einmal besucht?
- Ein solcher Weg wird Hamiltonscher Kreis genannt
- Beispiel:



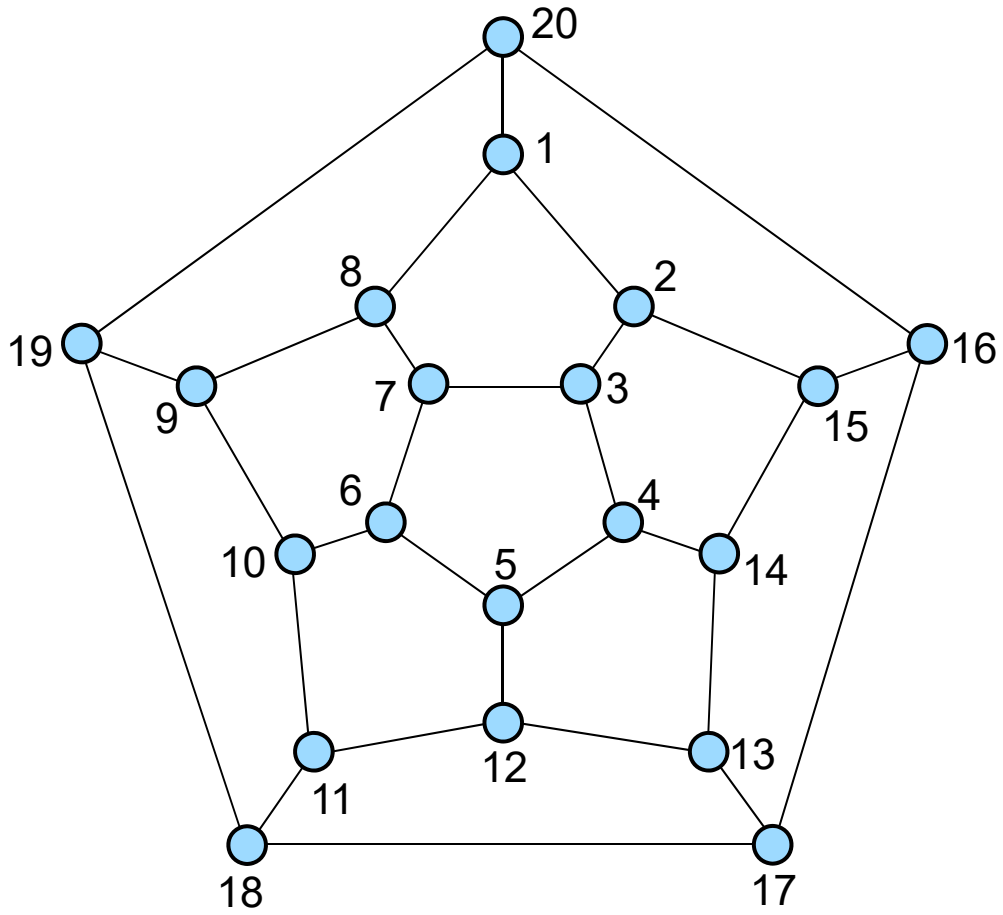
  - (A,B,C,E,D,A) ist ein Hamiltonscher Kreis
- Problem klingt dem Eulerschen Problem recht ähnlich
- Kann daraus die gleiche (gute) Laufzeitkomplexität geschlossen werden?
- Leider nein, d.h. es gibt keine allgemeingültige Aussage, die über einen Graphen und das Enthalten eines Hamiltonschen Kreises etwas sagt

Hamiltonsches Problem (II)

- Algorithmus zum Prüfen auf einen Hamiltonschen Kreis muss also alle Möglichkeiten durchprobieren
- Gegeben: Ungerichteter Graph G
- Algorithmus:
  - Für alle Permutationen von $v_1, ..., v_n \in V$:
    - Wenn $\forall\ i \in \{1, ..., n\} : \{v_i, v_{i+1}\} \in E \Rightarrow$ G hat Hamiltonschen Kreis
  - Trifft die Bedingung für keine Permutation zu $\Rightarrow$ G hat keinen Hamiltonschen Kreis
- Komplexität: O(n!)
  - Es gibt $n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 1 = n!$ mögliche Permutationen der Knoten
  - Dies entspricht größenordnungsmäßig $O(n^n)$ (Stirlingsche Formel)
- Fazit: Obwohl Eulerproblem und Hamiltonsches Problem auf den ersten Blick recht ähnlich sind, unterscheiden sie sich in der Laufzeitkomplexität erheblich!

Beispiel Hamiltonscher Kreis

- Der folgende Graph enthält einen Hamiltonschen Kreis, wenn man die Knoten in der Reihenfolge der Nummerierung durchläuft



Zusammenfassung

- Färbungen:
  - Färbung, Chromatische Zahl
  - Backtracking-Algorithmus
  - Problem NP-vollständig
  - Greedy-Algorithmus

- Kürzeste Wege:
  - Algorithmus von Dijkstra
  - Floyd-Algorithmus

- Euler- und Hamilton-Kreise:
  - Euler- und Hamilton-Graphen
  - Ähnliche Probleme, verschiedene Komplexitäten

Übungen (I)

- **Aufgabe 1**

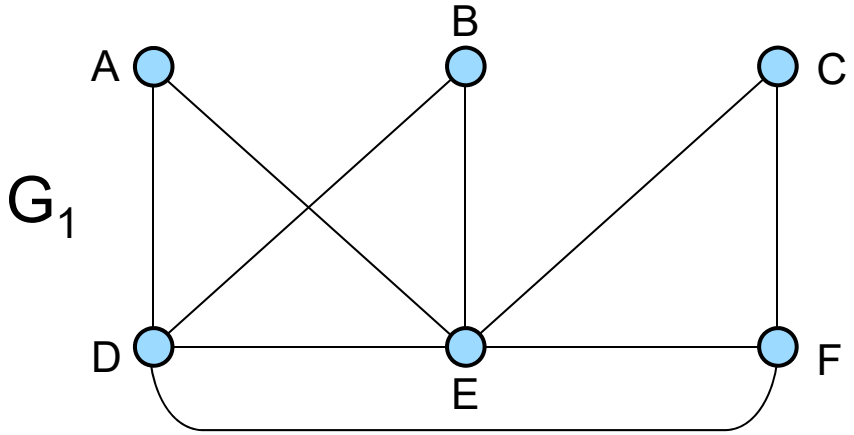
  Finden Sie im folgenden Rätsel fünf Begriffe aus diesem Vorlesungsabschnitt!

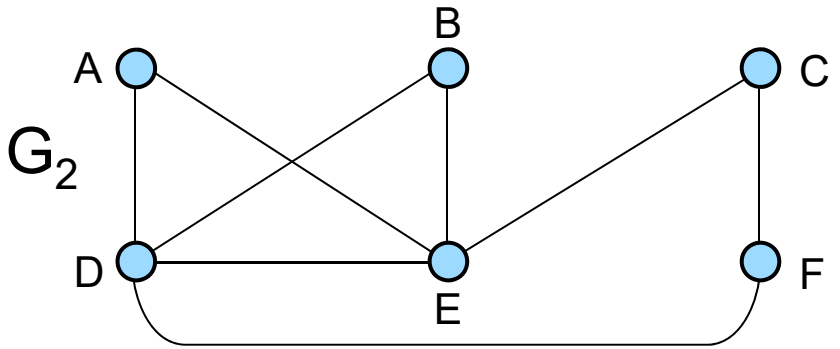
```
C S A N X M G K Y V E G Y J I Q K J R S
D T H D D Q B R N D J K M E Y T F R N I
J Q G Y L M B A Q O U K Z R F A S G S E
C X I N L I R O U B N U U O Z S I A N R
P L T H U T V H R L U N C H H C J Q V K
E I H R S B L J Z N J H J F B D J T D R
I U M K Y B R E T C Y U P B W X T D E E
P Y J L H A Z E H C S I T A M O R H C L
S I B L L R D M A S D M T M I N T C X U
D U H X I D P D O F Z O U B H L A Z Y E
O P O L N R Y V I U E Z Q G F P O J H D
X X K A N G X Y Y Y I L R V B Z Z M Q R
X Y U T A X M J K F D Y A F L O Y D J N
V A W R F P K F L F L N Q M U N Y G L Q
U J H P Q C U M F W Z H P L I L W L R Z
Y M Y A P P Q Y W R X Y S Z R N T B V X
B F H D X N I F F H F I Z W X G I K B A
Y I U Y F U X A S H N F K S D Y U M Y L
E X N F Q G N L O Z X P K Q V N X Q L A
O N U H Z A D A Y U D I H A V L K K N Y
```

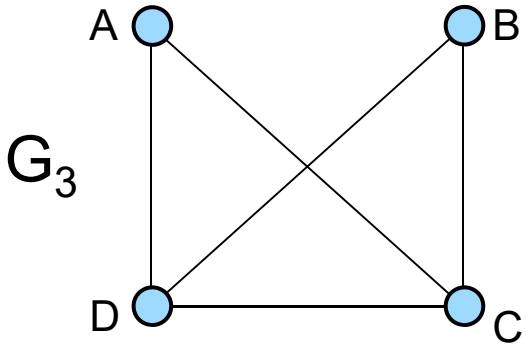
Übungen (II)

- **Aufgabe 2**

  Geben Sie für die folgenden Graphen an, ob diese einen Eulerschen und/oder einen Hamiltonschen Kreis besitzen! Geben Sie diesen auch jeweils an!







$G_4 = C_6$

$$A(G_5) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Übungen (III)

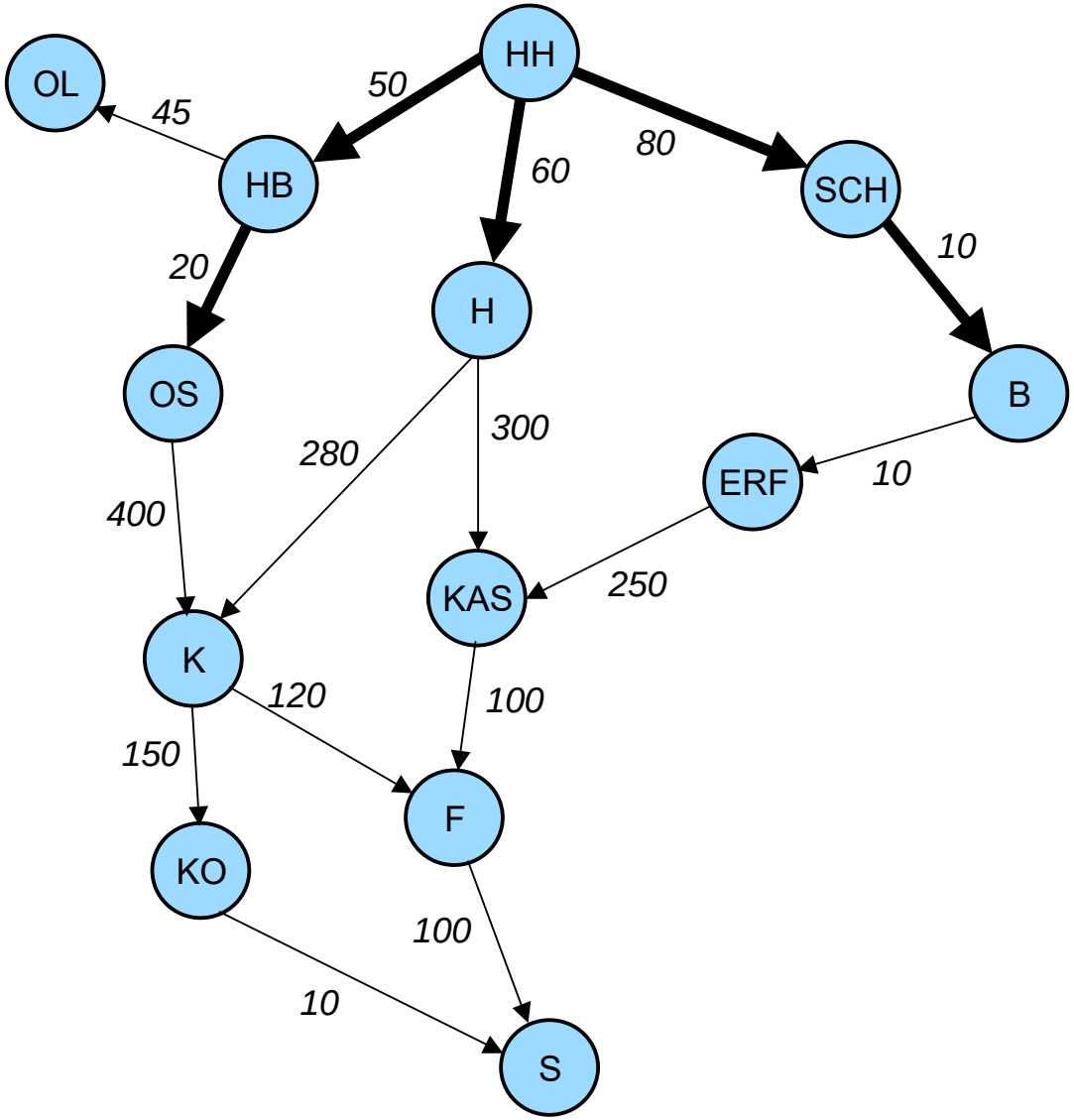
- **Aufgabe 3**

  Modellieren Sie die Karte der Bundesländer als Graphen und wenden Sie den Greedy-Färbealgorithmus an! Versuchen Sie durch Wahl des Startknotens und Auswahl des nächsten zu färbenden Knotens verschiedene Färbungen zu bekommen!

- **Aufgabe 4**

  Auf den folgenden Graphen ist der Algorithmus von Dijkstra zur Ermittlung des kürzesten Weges von Hamburg (HH) nach Stuttgart (S) angewendet worden. Die fett markierten Kanten sind vom Algorithmus bereits markiert worden.

Übungen (IV)

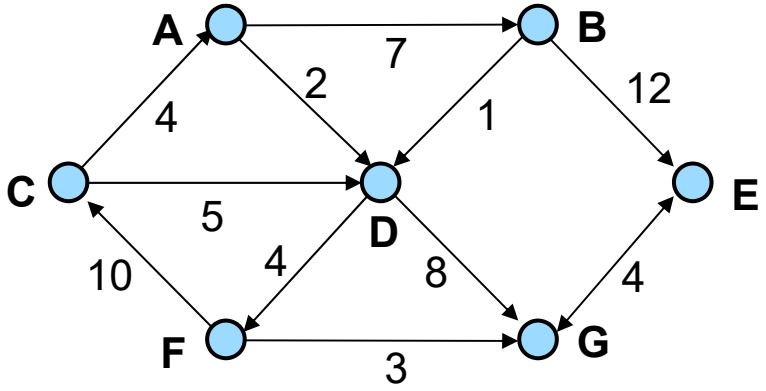

OL
HH
HB
H
SCH
OS
B
ERF
KAS
K
F
KO
S
45
50
60
80
20
10
280
300
400
10
250
120
100
150
100
10

Übungen (V)

a) Geben Sie die Reihenfolge an, in der die Kanten ausgewählt worden sind!
b) Erläutern Sie die Auswahl der nächsten beiden Kanten ausführlich!
c) Geben Sie die Knotenfolge an, die schließlich als Ergebnis des Algorithmus ausgegeben wird!

- **Aufgabe 5**

Gegeben sei der folgende Graph:



Ermitteln Sie mit dem Floyd-Algorithmus die Distanzmatrix!

Übungen (VI)

- **Aufgabe 6**

  a) Geben Sie für die beiden Beispielgraphen zum Eulerweg jeweils alle möglichen Eulerwege an!

  b) Besitzt der folgende Graph einen Eulerweg?