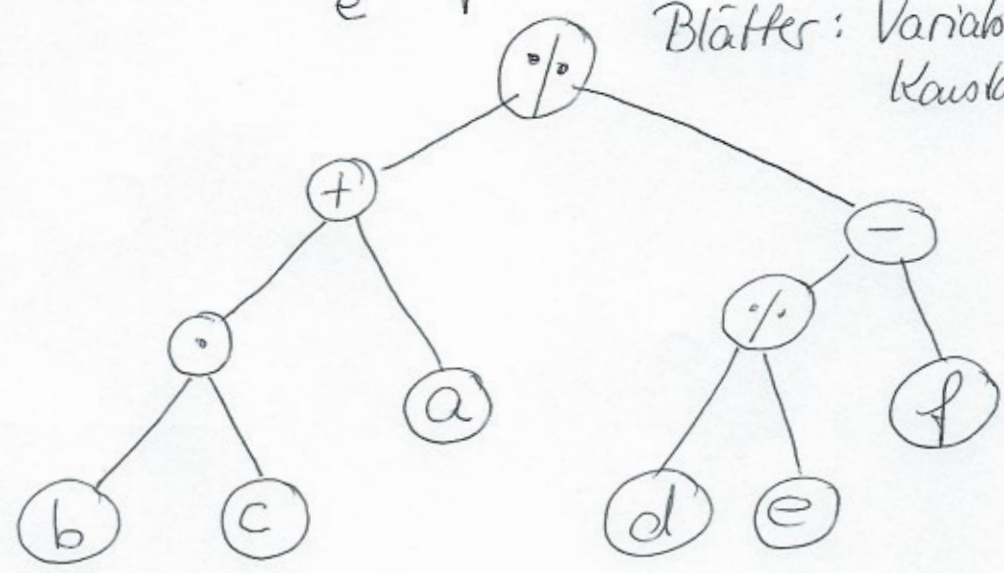


1. Beispiel

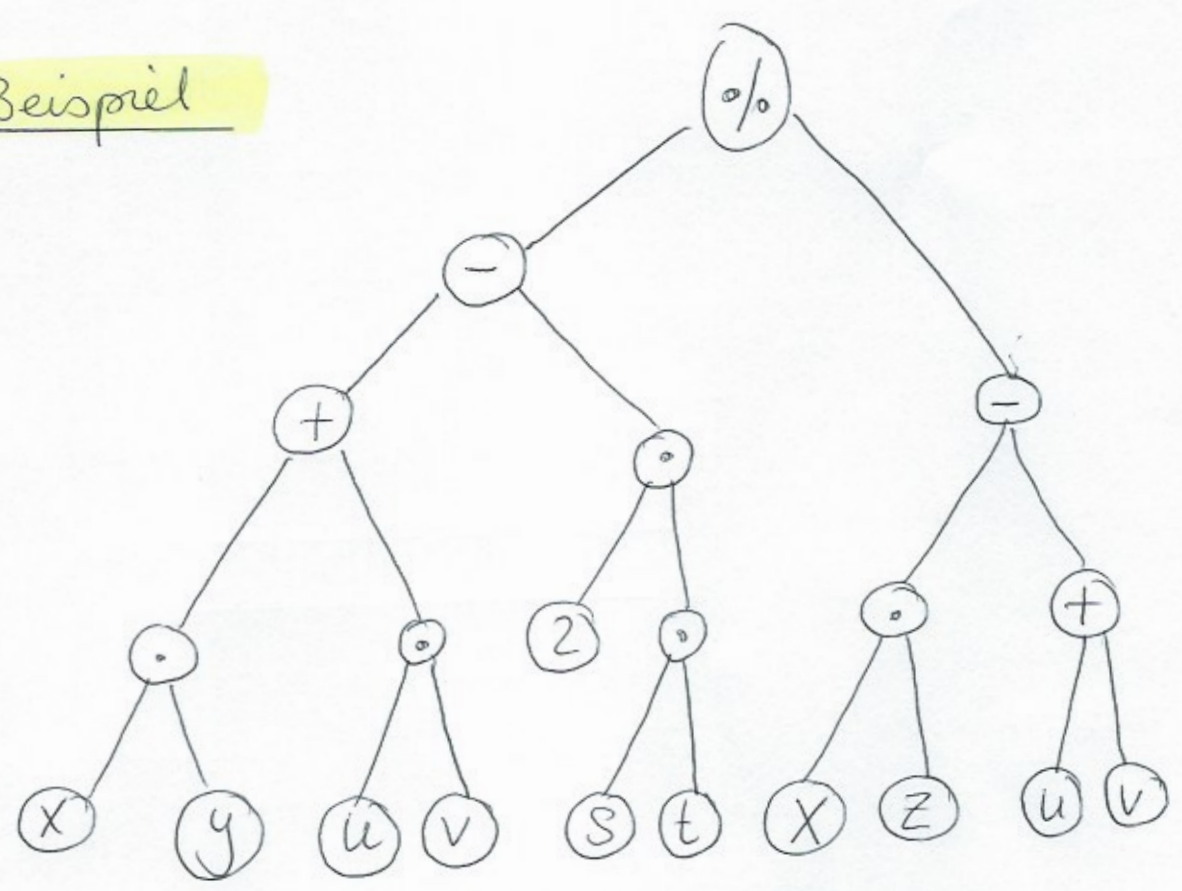
$$\frac{a + b \cdot c}{\frac{d}{e} - f}$$

innere Knoten: Operatoren

Blätter: Variablen, Konstanten



2. Beispiel



$$\frac{x \cdot y + u \cdot v - 2 \cdot s \cdot t}{x \cdot z - (u + v)}$$

Nachtrag:

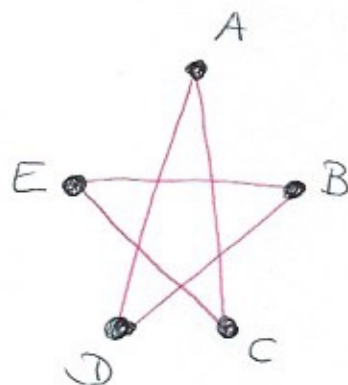
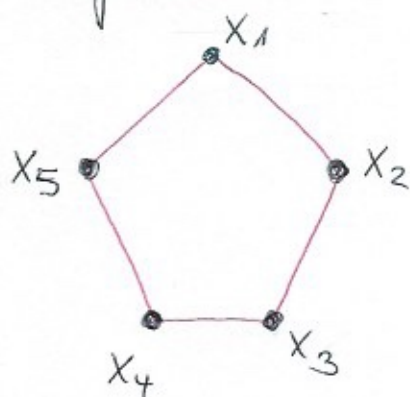
Def: Isomorphe Graphen

Zwei Graphen heißen **isomorph**, wenn es eine **bijektive Abbildung φ** von der Knotenmenge des einen Graphen G in die Knotenmenge des anderen Graphen G' gibt, so dass gilt:

$$\{x_i, x_j\} \text{ Kante von } G \Leftrightarrow \{\varphi(x_i), \varphi(x_j)\} \text{ Kante von } G'$$

Damit werden auch die **Kantenmengen bijektiv aufeinander abgebildet**.

Ein schönes Beispiel für auf den ersten Blick recht unterschiedlich aussehende Graphen finden Sie im Skript von Prof. Koenen:



Die **bijektive Abbildung** kann nur zwischen zwei **gleichmächtigen Knoten- und Kantenmengen** definiert werden. Das ist also das erste, was zu überprüfen ist, wenn man eine solche bijektive Abbildung sucht.

Überprüfen Sie, ob folgende Abbildung die Isomorphie zwischen den beiden Graphen ausdrückt im obigen Beispiel.

$$\begin{aligned} \varphi(x_1) &= A & \varphi(x_2) &= C & \varphi(x_3) &= E \\ \varphi(x_4) &= B & \varphi(x_5) &= D \end{aligned}$$

Die Abbildung der Kanten kann entsprechend definiert werden.

Nachtrag zu den Adjazenzmatrizen:

Adjazenzmatrix A eines gerichteten Graphen
Mit Hilfe der Potenzen $A^r = A \cdot \dots \cdot A$ lassen sich Aussagen über die Existenz und über die Anzahl von Wegen (Pfeilfolgen) in Digraphen machen.

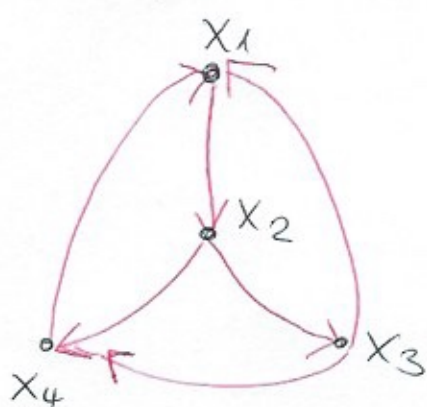
Zur Erinnerung: A Adjazenzmatrix mit $a_{ij} = \begin{cases} 1 & \text{wenn Pfeil von } x_i \text{ nach } x_j \\ 0 & \text{sonst} \end{cases}$

Sei $A^r = A \cdot \dots \cdot A$ r -te Potenz von A

Das Element $a_{ij}^{(r)}$ der Matrix A^r (steht in der i -ten Zeile und der j -ten Spalte)

gibt die Anzahl verschiedener Pfeilfolgen der Länge r von x_i nach x_j im Graphen an. Der Beweis erfolgt mit dem Beweisverfahren der vollständigen Induktion und soll an dieser Stelle nicht gemacht werden.

Bp:



$$\begin{matrix} & x_1 & x_2 & x_3 & x_4 \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

$$A^2 = A \cdot A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

1 Weg der Länge 2 von x_4 nach x_2
 $x_4 \rightarrow x_1 \rightarrow x_2$
2 Wege der Länge 2 von x_2 nach x_1

$x_2 \rightarrow x_3 \rightarrow x_1$
 $x_2 \rightarrow x_4 \rightarrow x_1$

Wenn ein Digraph einen Zyklus enthält, dann werden die Potenzen A^r der Adjazenzmatrix niemals die Nullmatrix werden, da man ja beliebig oft den Zyklus durchlaufen kann.

Ein Digraph mit n Knoten ist genau dann azyklisch (= zyklensfrei), wenn es eine Zahl r mit $1 \leq r \leq n$ gibt mit $A^r \neq 0$ und $A^s = 0$ für alle $s > r$. Dabei ist 0 die Nullmatrix.

Durchlaufen von Graphen

Nachdem wir nun die Graphen mit all den verschiedenen Eigenschaften kennengelernt haben und auch schon an der ein oder anderen Stelle von Durchlaufen von Graphen gesprochen haben, es sei ganz am Anfang des Kapitels Graphentheorie erinnert, an L. Euler und seinem geplanten Weg durch Königsberg. Das Königsberger Brückenproblem ist auch ein "Durchlaufen" von Graphen. Bei zahlreichen anderen Anwendungen muss man von einem bestimmten Knoten zu einem bestimmten anderen Knoten gelangen, entlang einer Kantenfolge (auch Bereich definiert). Über die Adjazenzmatrix und ihre Potenzen kann man Aussagen über die Anzahl und Existenz von Wegen vorgegebener "Länge" machen.

Für viele dieser Anwendungen vor allem in großen, unübersichtlichen Graphen, wird eine Suche mit System gefordert. Aufgabenstellungen wie z.B. bei der Suche nach einem Eulerweg ("jede Kante nur einmal") müssen mit System angegangen werden.

Unnötige Kanten weglassen und dennoch von einem zum anderen oder besser zu allen anderen Knoten kommen, das wird zunächst einmal die Forderung sein.

Dazu benötigen wir noch einmal ein paar Begriffe aus der Graphentheorie.

Def.: (aufspannender Baum, Gerüst)

Ein zusammenhängender Teilgraph eines zusammenhängenden Graphen mit minimaler Kantenanzahl heißt aufspannender Baum, Gerüst oder Spannbau.

Bem: die Knotenanzahl ist bei einem aufspannenden Baum unverändert, es werden nur Kanten weggelassen (s. Def. Teilgraph Blatt 82)

Bem: Ein aufspannender Baum enthält keinen Kreis bzw. Zyklus und ist auf alle Fälle ein Baum. Er ist auf alle Fälle zusammenhängend. Im englischen Sprachraum wird dafür der Begriff spanning tree verwendet.

Aufgabe: Wie viele Spannbäume gibt es in einem vollständigen Graphen mit 4 Knoten.

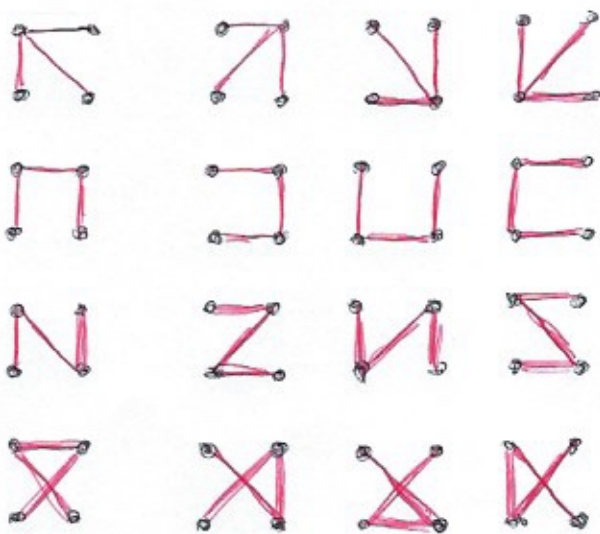


Cayley-Formel:

Ein vollständiger Graph mit n Knoten hat $\frac{n(n-1)}{2}$ Kanten.

aufspannende Bäume

hier $n=4$ also $4^{n-2} = 16$



Der Beweis der Cayley-Formel soll hier nicht erbracht werden.

Wie im Eingangsbeispiel an der Karte mit den Städten rund um Köln zu sehen war, waren die Kanten, die Straßenverbindungen repräsentierten, bewertet. Einen bewerteten Graphen haben wir auf Blatt (84) definiert.

Wenn man aufspannende Bäume bewerteter Graphen betrachtet, dann werden besonders die von Interesse sein, die z.B. in der Summe das kürzeste Streckennetz, die geringsten Kosten usw. widerspiegeln.

Das führt zu folgender Definition:

Def.: Minimal aufspannender Baum (MST)

T sei ein aufspannender Baum eines minimum spanning tree bewerteten Graphen G . T heißt minimal aufspannender Baum, wenn die Summe seiner Kantenbewertungen minimal ist.

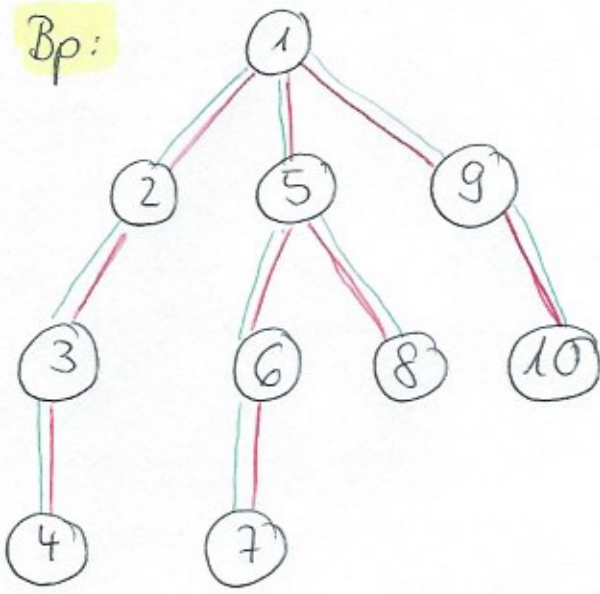
Zunächst einmal soll unabhängig von bewerteten Kanten untersucht werden, mit welchen Suchalgorithmen man mit System in großen, unübersichtlichen Graphen, Wege zu finden, jeden Knoten mindestens einmal zu besuchen.

Man hat zwei Möglichkeiten, nach Auswahl eines Startknotens vorzugehen:

1) Tiefensuche

Startknoten festlegen, als besucht markieren
Suche einen Nachfolgerknoten, markieren,
wieder Nachfolger suchen. Wenn kein Nachfolger
mehr vorhanden, Rückkehr zum zuletzt besuchten
Knoten, der noch nicht alle Nachfolger markiert hat.

Bp:



Tiefensuche : Startknoten ①

Nachfolger ②

Nachfolger ③

Nachfolger ④ kein Nachfolger
zurück nach ①

Nachfolger ⑤

Nachfolger ⑥

Nachfolger ⑦ kein Nachfolger
zurück nach ⑤

Nachfolger ⑧ kein Nachfolger
zurück nach ①

Nachfolger ⑨

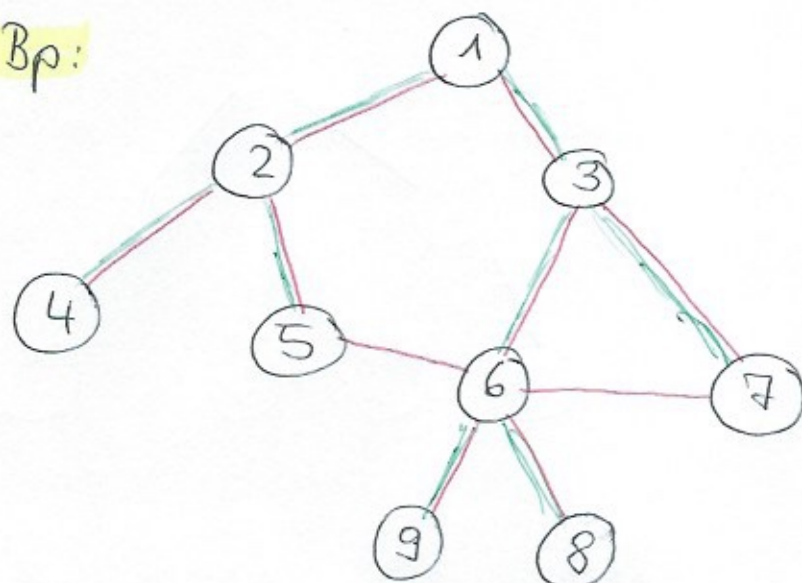
Nachfolger ⑩ kein Nachfolger

alle Knoten besucht

2) Breitensuche

Startknoten festlegen, als besucht markieren
Alle Nachfolgerknoten markieren ("Breite")
von diesen wieder alle Nachfolgerknoten, bis alle
Knoten markiert sind.

Bp:



Breitensuche:

Startknoten ①

Markiere ② und ③

Markiere bei ② ④ und ⑤

bei ③ ⑥ und ⑦

Markiere bei ⑥ ⑨ und ⑧

alle Knoten besucht

Frage: Wie findet man einen minimal aufspannenden Baum?

Mit dem Algorithmus von Kruskal hat man eine klar formulierte, relativ einfache Lösung.

Joseph Kruskal (1928 - 2010) veröffentlichte diesen Algorithmus 1956.

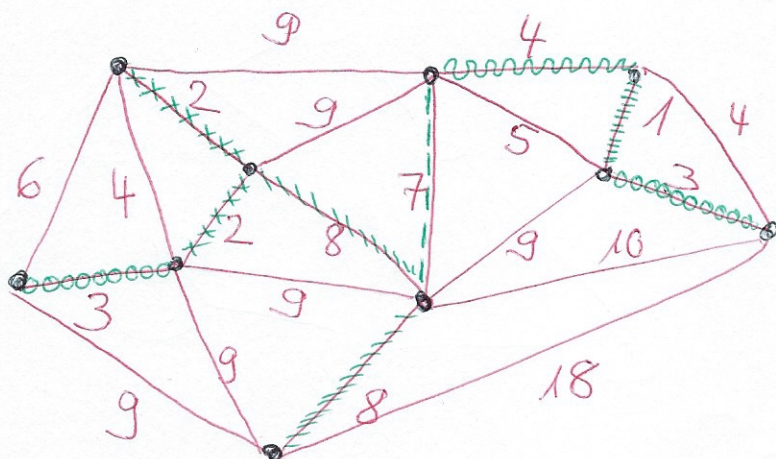
Der Algorithmus ist anzuwenden auf einen ungerichteten bewerteten Graphen.

- 1) Suche in G (bewertet) die Kante mit dem kleinsten Gewicht, falls die Kante mit den schon markierten einen Kreis bildet, verwerfen, ansonsten hinzufügen
- 2) Dieser Schritt wird so oft wiederholt, bis nichts mehr hinzugefügt werden kann.

Bp.:

111

1



Algorithmus von Kruskal:

- 1) Kante mit dem kleinsten Gewicht : 1 ~~~~~
- 2) Kante mit dem kleinsten Gewicht : 2, 2 xxxx
- 3) Kante mit dem kleinsten Gewicht : 3, 3 ooooo
- 4) Kante mit dem kleinsten Gewicht : 4 ~~~~~
die anderen beiden gehen nicht (Zyklus)
- 5) Kante mit dem kleinsten Gewicht : 5 (Zyklus)
- 6) Kante mit dem kleinsten Gewicht : 6 (Zyklus)
- 7) Kante mit dem kleinsten Gewicht : 7 ----
- 8) Kante mit dem kleinsten Gewicht : 8, 8 -----

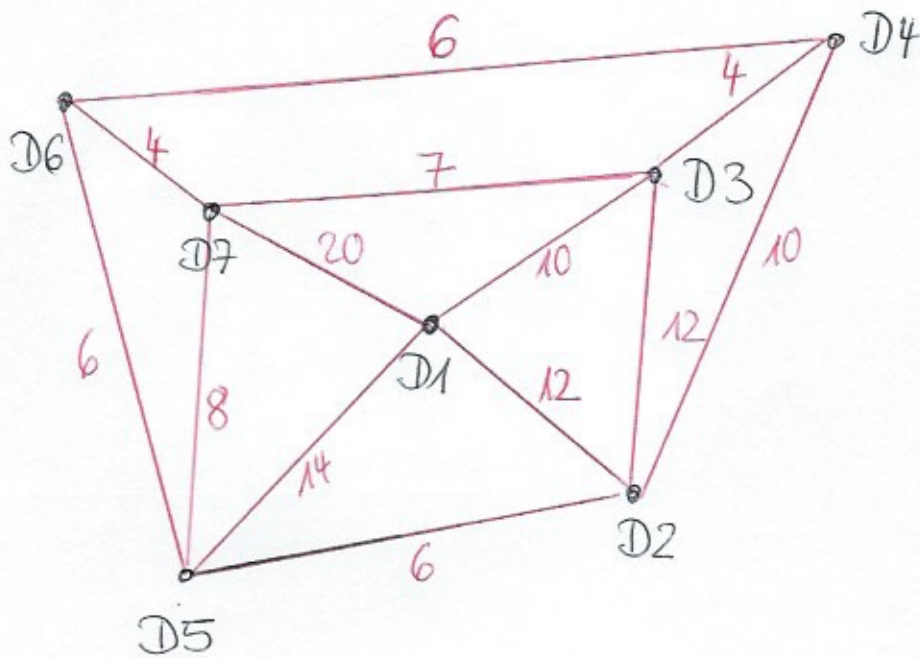
Alle weiteren in Frage kommenden Kanten bilden Kreise
Durch Addition der markierten Kantenwerte erhält
man das Minimum: 38

Sieben Dörfer sollen mittels Straßen verbunden
werden. Für diesen Bau soll möglichst wenig Geld
ausgegeben werden. Zeichnen Sie den Graphen und
ermitteln Sie die minimalen Kosten.

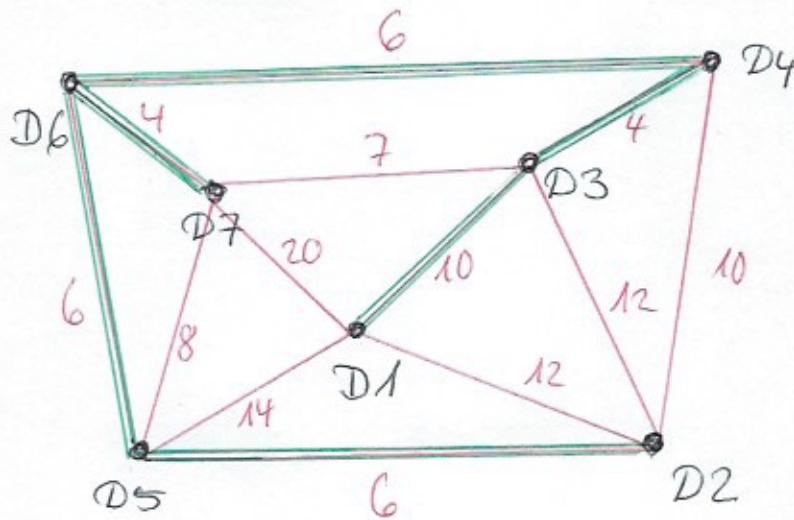
2

	D1	D2	D3	D4	D5	D6	D7
D1	0	12	10	0	14	0	20
D2	12	0	12	10	6	0	0
D3	10	12	0	4	0	0	7
D4	0	10	4	0	0	6	0
D5	14	6	0	0	0	6	8
D6	0	0	0	6	6	0	4
D7	20	0	7	0	8	4	0

Der (kreuzungsfrei) gezeichnete Graph:



- 1) Man wählt zuerst die Kanten $D7, D6$ und $D3, D4$ mit den Kantenbewertungen 4 aus
- 2) Man wählt nun $D6, D5$ und $D4, D6$ und $D5, D2$ Kantenbewertungen 6 aus
- 3) Die Kante mit der Bewertung 7 kann nicht gewählt werden, da es einen Kreis geben würde.
- 4) Die Kante mit der Bewertung 8 kann auch nicht gewählt werden, da es einen Kreis geben würde
- 5) Man wählt $D3, D1$ mit Bewertung 10



Die Summe der Kantenbewertungen beträgt: 36

Bestimmung kürzester und längster Wege in Graphen

Geht man von den bereits vorgestellten Anwendungsbeispielen aus, so wird es häufig wichtig sein, z. B. den kürzesten Transportweg herauszufinden, der in einem Graphen, der alle möglichen Transportwege zwischen Lieferformen aufzeigt, vorhanden ist. Dabei sollen die entsprechenden Weglängen (als Kantenmarkierung im bewerteten Graphen) berücksichtigt werden.

Das Problem: "Kürzeste Wege in Graphen", ist eines der grundlegenden algorithmischen Problemstellungen in der Graphentheorie, das in vielen Anwendungsbereichen auftritt und häufig als Teilproblem in anderen Problemstellungen auftritt. Viele Probleme lassen sich auch direkt als "Problem kürzester Wege" umformulieren. Dabei gibt es eine Reihe von Varianten. Wir wollen hier folgende Problemstellungen ansprechen:

- Aufsuchen der kürzesten Wege von einem vorgegebenen Knoten aus zu allen übrigen Knoten des Graphen:
 "single source shortest paths"
- Aufsuchen der kürzesten Wege zwischen je zwei Knoten in einem Graphen:
 "all pairs shortest paths"

Anderer Fragestellungen wären auch die Frage nach kürzesten Wegen in Digraphen (gerichteten Graphen), hier muss jeweils die "Durchlaufrichtung" berücksichtigt werden.

Andererseits kann auch die Frage nach den längsten Wegen in einem Graphen gestellt werden, z. B. zeitlich gesehen in einem Projektablauf oder man fragt nach der Länge von Dienst- und Lustauswegen.

Bem.: Die Bestimmung längster Wege ist nur in einem gerichteten Graphen ohne Schleifen (Schlingen) und Zyklen sinnvoll, da man, wie leicht einzusehen, durch wiederholtes Durchlaufen einer Schleife oder eines Kreises immer einen beliebig langen Weg finden kann. Man kann zeigen, dass die Bestimmung längster Wege in einem Graphen mit den Algorithmen für kürzeste Wege erfolgen kann. Dazu ist eine leichte Modifizierung notwendig, auf die hier nicht eingegangen wird.

Bem.: Die Bestimmung von kürzesten (oder auch längsten) Wegen in unbewerteten Graphen, bei dem die Länge eines Weges durch die Anzahl der durchlaufenen Kanten gemessen wird, kann immer auf die Ermittlung kürzester oder längster Wege in einem bewerteten Graphen mit Kantenbewertung 1 zurückgeführt werden.

Gegenstand der weiteren Vorlesung soll die Bestimmung der kürzesten Wege in einem Graphen sein, was für eine Einführung in das Gebiet der Graphentheorie auch ausreichend ist.

Markierungsalgorithmus zur Bestimmung kürzester Wege in Graphen : Algorithmus von Dijkstra

Vorbem.: Der Algorithmus von Dijkstra gehört zu den sogenannten Greedy-Algorithmen. Greedy-Algorithmen bilden in der Informatik eine spezielle Klasse von Algorithmen. Greedy ist das englische Wort für gierig, diese Algorithmen zeichnen sich dadurch aus, dass sie schrittweise immer denjenigen nachfolgenden Zustand auswählen, der in dem betreffenden Schritt das beste Ergebnis liefert, auf dem dann aufgebaut wird.

(115)

Im Jahr 1959 veröffentlichte Edsger W. Dijkstra (1930-2002) diesen Algorithmus.

Der Algorithmus von Dijkstra liefert eine Lösung zum Problem: Auffinden kürzester Wege von einem Knoten aus zu allen übrigen (single source shortest paths)

Wann ist dieses Problem überhaupt lösbar? Wenn ein zusammenhängender, bewerteter, schlichter Graph vorliegt, in dem die Kantenbewertungen alle nicht negativ sind, dann ist der Algorithmus anzuwenden. Wenn der Graph aus mehreren Zusammenhangskomponenten besteht, dann liefert der Algorithmus keine Lösung.

Grundidee: man wähle immer diejenige Kante aus, die vom Startknoten aus immer den kürzesten Abschnitt (Summe der Kantenbewertungen bis zum betreffenden Schritt minimal) tieft.

Wie das nun genau durchzuführen ist, soll im folgenden kurz theoretisch beschrieben werden und danach an einem konkreten Beispiel durchgeführt werden.

Allgemeine Kurzbeschreibung des Algorithmus:

Gegeben: bewerteter Graph $G = (M, K, v, f)$
mit $f(k) \geq 0$ für alle $k \in K$ (alle Kanten mit positiver Bewertung)

Bezeichnung: $D(x_i)$: Bewertung der Distanz vom Startknoten x_0 zum Knoten x_i

(1) Markiere x_0 (Startknoten) mit $D(x_0) = 0$
(x_0 hat zu sich selbst die Entfernung 0)

Setze $i = 1$, $M_1 := \{x_0\}$ weiter mit (2)

(2) Sei M^* die Menge aller Knoten x_s der Kanten (x_r, x_s) , deren Knoten x_r in M_i liegt, d. h.

$M^* = \{x_s \in M \setminus M_i \mid \text{es gibt eine Kante } (x_r, x_s) \text{ mit } x_r \in M_i\}$

Falls $M^* = \emptyset$ weiter mit (6), falls $M^* \neq \emptyset$ weiter mit (3)