



Wintersemester 2017/2018

Lineare und kombinatorische Optimierung

Übungsblatt 4

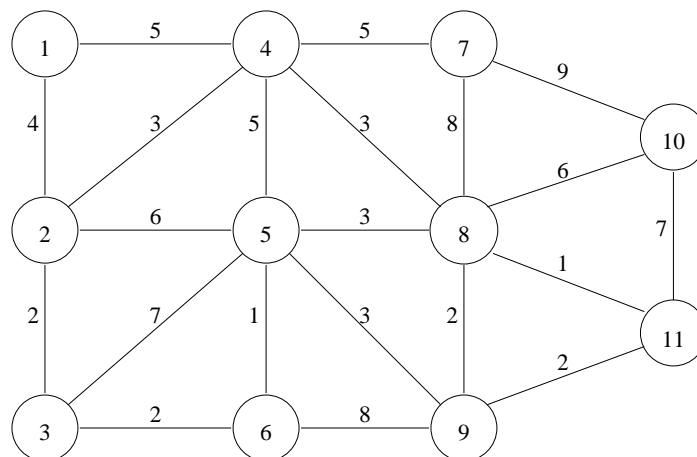
Gruppenübungen

Aufgabe 1. (Minimal aufspannender Baum)

(0 Punkte)

Bestimmen Sie im folgenden Graphen einen minimal aufspannenden Baum

1. mit dem Algorithmus von Kruskal;
2. mit dem Algorithmus von Prim.



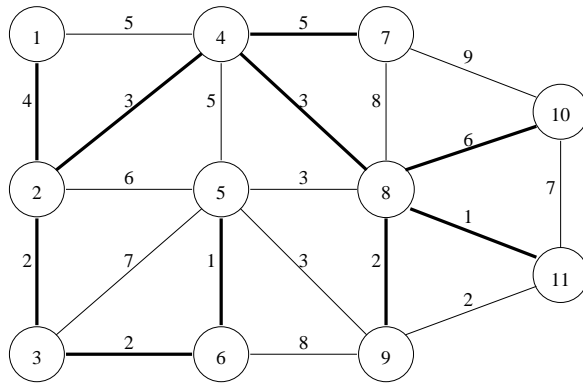
Lösung:

1. Die Kanten werden zunächst nach aufsteigenden Gewichten sortiert. Falls mehrere Kanten dasselbe Gewicht haben, ist die Sortierung dieser Kanten untereinander beliebig. Wir geben *eine* mögliche Sortierung an. In dieser Reihenfolge werden die Kanten, die mit den bisher einbezogenen Kanten keinen Kreis bilden, einbezogen, bis der so entstehende Teilgraph zusammenhängend ist. Dies ist der Fall, wenn $n - 1$ Kanten einbezogen sind.

Sortierung der Kanten: $\{5, 6\}$, $\{8, 11\}$, $\{2, 3\}$, $\{3, 6\}$, $\{8, 9\}$, $\{9, 11\}$, $\{2, 4\}$, $\{4, 8\}$, $\{5, 8\}$, $\{5, 9\}$, $\{1, 2\}$, $\{1, 4\}$, $\{4, 5\}$, $\{4, 7\}$, $\{2, 5\}$, $\{8, 10\}$, $\{3, 5\}$, $\{10, 11\}$, $\{6, 9\}$, $\{7, 8\}$, $\{7, 10\}$.

Da die Reihenfolge und auch die Kantenmenge nicht eindeutig ist, ist auch der damit entstandene Baum nicht eindeutig. Eindeutig ist allerdings der Wert des minimal aufspannenden Baumes. Er beträgt 29. Alle anderen aufspannenden Bäume sind also minimal, wenn sie Gewicht 29 haben.

Es ergibt sich folgender Baum:



2. Wir beginnen mit einem beliebigen Knoten w und setzen $W := \{w\}$, $T := \emptyset$ und $c(T) = 0$. Für jede Kante $e \in \delta(w)$ bestimmen wir c_e und beziehen $\{u, v\}$ mit $c_{uv} = \min\{c_e | e \in \delta(w)\}$ mit $u \in W, v \notin W$ in den Teilgraphen T ein. Dann setzen wir $W = W \cup \{v\}$, $T = T \cup \{u, v\}$ und $c(T) = c(T) + c_{uv}$. In unserem Beispiel beginnen wir mit $w = 7$.

1. Iteration:

$W = \{7\}$, $T = \emptyset$, $c(T) = 0$.

$\delta(W) = \{\{7, 4\}, \{7, 8\}, \{7, 10\}\}$, mit den Kantengewichten 5, 8 und 9.

$\Rightarrow \{u, v\} = \{7, 4\}$ mit $c_{uv} = 5$.

$\Rightarrow W = \{7, 4\}$, $T = \{\{7, 4\}\}$, $c(T) = 5$.

2. Iteration:

$W = \{7, 4\}$, $T = \{\{7, 4\}\}$, $c(T) = 5$.

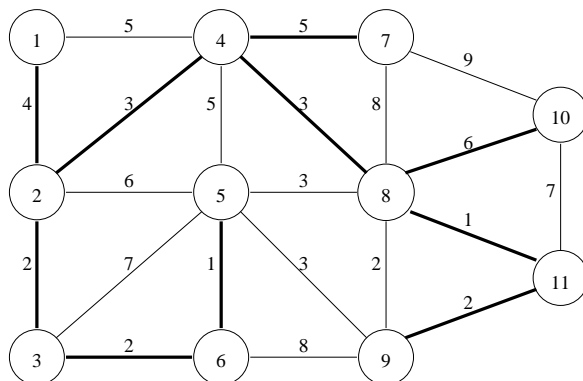
$\delta(W) = \{\{4, 1\}, \{4, 2\}, \{4, 5\}, \{4, 8\}, \{7, 8\}, \{7, 10\}\}$, mit den Kantengewichten 5, 3, 5, 3, 8 und 9.

$\Rightarrow \{u, v\} = \{4, 2\}$ mit $c_{uv} = 3$.

$\Rightarrow W = \{7, 4, 2\}$, $T = \{\{7, 4\}, \{4, 2\}\}$, $c(T) = 8$.

usw.

Es ergibt sich folgender Baum, der wieder nicht eindeutig ist, der aber natürlich ebenfalls Wert 29 hat.



Aufgabe 2. (Baum-Algorithmus)

(0 Punkte)

Geben Sie einen Algorithmus an, der in $O(|V|)$ bestimmt, ob ein Graph $G = (V, E)$ ein Baum ist.

Lösung:

Zuerst werden die Kanten gezählt (ist typischerweise schon in der Adjazenzliste gespeichert). Wenn es mehr als $|V| - 1$ Kanten gibt, terminiert der Algorithmus: der Graph ist kein Baum. Gleiches gilt für den Fall, dass es weniger als $|V| - 1$ Kanten gibt. Danach wird eine Breitensuche (oder Tiefensuche) gestartet, um Zusammenhang zu prüfen. Diese Suche hat eine Komplexität von $O(|V| + |E|)$. Da $|E| = |V| - 1$, ist die Komplexität damit $2|V| - 1 \in O(|V|)$.

Aufgabe 3. (Azyklische Digraphen)

(0 Punkte)

Definition. Sei $D = (V, A)$ ein Digraph. Eine Funktion $f : V \rightarrow \{1, \dots, |V|\}$ heißt *topologische*

Sortierung von V , falls

$$f(u) < f(v) \quad \text{für alle } (u, v) \in A$$

gilt.

Definition. Wir nennen einen Digraph $D = (V, A)$ *azyklisch*, wenn D keine gerichteten Kreise enthält.

Satz. D hat eine topologische Sortierung genau dann, wenn D azyklisch ist.

Um einen kürzesten Weg auf azyklischen Digraphen zu finden, kann der Algorithmus 2 von Moore und Bellmann verwendet werden. Mithilfe des Algorithmus 1 lässt sich feststellen, ob ein Digraph azyklisch ist.

1. Verstehen Sie die beiden Algorithmen 1 und 2.
2. Wenden Sie Algorithmus 1 auf den Beispieldigraphen an, um festzustellen, dass es sich um einen azyklischen Graphen handelt.
3. Berechnen Sie die kürzesten Wege von s zu allen anderen Knoten mithilfe von Algorithmus 2.

Algorithmus 1 Topologische Sortierung

Eingabe: Ein Digraph $D = (V, A)$.

Ausgabe: Eine topologische Sortierung $f : V \rightarrow \{1, \dots, |V|\}$ oder die Meldung, dass D einen gerichteten Kreis enthält.

1: Setze

$$\text{indeg}(v) = |\delta^{\text{in}}(v)| \quad \text{für alle } v \in V,$$

$$L = \{v \in V : \text{indeg}(v) = 0\},$$

next = 0.

2: **while** $L \neq \emptyset$ **do**

3: Wähle ein $u \in L$.

4: Setze $L \leftarrow L \setminus \{u\}$.

5: Setze next \leftarrow next + 1.

6: Setze $f(u) \leftarrow$ next.

7: **for all** $(u, v) \in \delta^{\text{out}}(u)$ **do**

8: Setze $\text{indeg}(v) \leftarrow \text{indeg}(v) - 1$.

9: **if** $\text{indeg}(v) = 0$ **then**

10: Setze $L \leftarrow L \cup \{v\}$.

11: **if** next < $|V|$ **then**

12: **return** “ D enthält einen Kreis.”

13: **else**

14: **return** topologische Sortierung f .

Algorithmus 2 von Moore und Bellmann für azyklische Digraphen

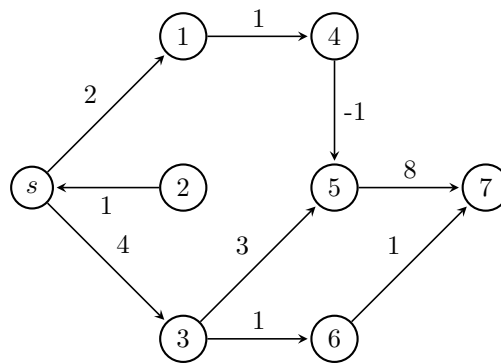
Eingabe: Ein azyklischer Digraph $D = (V, A)$ mit Bogengewichten $c_a, a \in A$ (möglicherweise auch negativ). Ein Startknoten $s \in V$.

Ausgabe: Kürzeste gerichtete Wege von s nach v für alle $v \in V$ und deren Längen.

- 1: Sortiere die Knoten in V topologisch o.B.d.A. $1, \dots, n$ mit $i < j$ für alle $(i, j) \in A$.
- 2: Setze

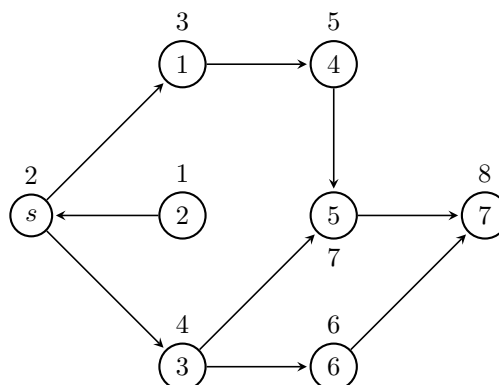
$$d(v) = \begin{cases} 0, & \text{falls } v = s, \\ +\infty, & \text{sonst,} \end{cases}$$
$$\text{pred}(v) = \begin{cases} s, & \text{falls } v = s, \\ \text{ungesetzt,} & \text{sonst.} \end{cases}$$

- 3: **for** $u = s : n$ **do**
 - 4: **for all** $(u, v) \in \delta^{\text{out}}(u)$ **do**
 - 5: **if** $d(v) > d(u) + c_{uv}$ **then**
 - 6: Setze $d(v) \leftarrow d(u) + c_{uv}$.
 - 7: Setze $\text{pred}(v) \leftarrow u$.
 - 8: **if** $d(v) < \infty$ **then**
 - 9: $d(v)$ enthält die Länge eines kürzesten Weges von s nach v .
 - 10: $\text{pred}(\cdot)$ die Vorgänger auf den kürzesten Wegen.
 - 11: **else if** $d(v) = \infty$ **then**
 - 12: Es existiert kein Weg von s nach v .
-



Lösung:

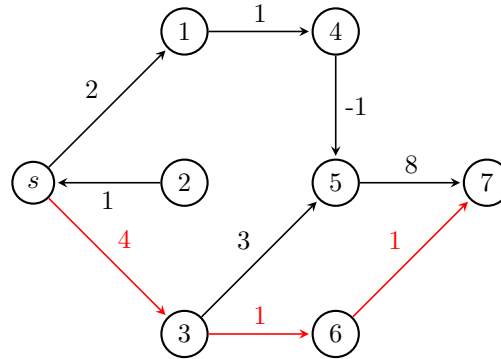
2. Mithilfe von Algorithmus 1 wird die folgende topologische Sortierung erreicht:



3. Der Algorithmus 2 liefert folgendes Resultat:

i	$\text{pred}(i)$	i	$d(i)$
s	s	s	0
1	s	1	2
2	ungesetzt	2	∞
3	s	3	4
4	1	4	3
5	4	5	2
6	3	6	5
7	6	7	6

Beispielsweise wäre dies der kürzeste Weg von s nach 7 mit Gewicht 6:



Hausübungen

Aufgabe 4. (Charakterisierung von Bäumen)

(4 Punkte)

Beweisen Sie, dass für einen Graphen $G = (V, E)$ mit $n \geq 2$ Knoten die folgenden Aussagen äquivalent sind:

- G ist ein Baum, d.h., G ist zusammenhängend und enthält keinen Kreis.
- Für je zwei Knoten u und v aus V gibt es genau einen u - v -Weg in G .
- G ist minimal zusammenhängend (d.h., G ist zusammenhängend und $G \setminus \{e\}$ ist nicht zusammenhängend für alle $e \in E$).
- G enthält keinen Kreis und bei Hinzufügen einer Kante wird genau ein Kreis erzeugt.
- G ist zusammenhängend und enthält $n - 1$ Kanten.

Lösung:

(a) \Rightarrow (b): Seien $u, v \in V$. Da G zusammenhängend ist, gibt es mindestens einen Weg $p = (v = v_1, v_2, \dots, v_k = u)$ in G . Nehmen wir an, es gibt einen anderen Weg $q = (v = w_1, w_2, \dots, w_l = u)$ in G . Sei i der kleinste Index mit $v_i \neq w_i$ und j der kleinste Index mit $v_j = w_j$ und $j > i$. Beachte, $i > 1$ (denn $v_1 = w_1$) und j existiert, da $v_k = w_l$. Dann bildet $(v_{i-1}, v_i, \dots, v_j = w_j, w_{j-1}, \dots, w_{i-1} = v_{i-1})$ einen Kreis in G . Widerspruch.

(a) \Rightarrow (c): Sei $e = uv$ eine beliebige Kante in G . Wenn $G \setminus \{e\}$ zusammenhängend ist, dann gibt es einen Weg zwischen u und v , der e nicht enthält. Wenn wir nun diesen Weg in G betrachten und die Kante e dranhängen, dann erhalten wir einen Kreis in G und das ist ein Widerspruch.

(a) \Rightarrow (d): Nehmen wir an, wir möchten eine Kante e zwischen den Knoten $u, v \in V$ hinzufügen. Da G zusammenhängend ist, gibt es schon einen Weg zwischen u und v und, wenn wir nun die Kante e hinzufügen, entsteht ein Kreis in G . Da es genau einen Weg zwischen u und v in G gibt, entsteht auch genau ein Kreis, wenn wir e hinzufügen.

(a) \Rightarrow (e): Wir beweisen mit Induktion über n . Wenn $n = 1$ ist, dann haben wir nur einen einzelnen Knoten und $n - 1 = 0$ Kanten. Nehmen wir nun an, die Aussage gilt für Bäume mit weniger als n Knoten. Sei T ein Baum mit $n > 1$ Knoten und sei v ein Blatt von T . Wir entfernen v von T und erhalten einen

Baum T' mit $n - 1$ Knoten. Tatsächlich ist T' zusammenhängend (da wir ein Blatt entfernt haben) und hat keinen Kreis (da T schon keinen Kreis hatte). Per Induktionsannahme gibt es genau $n - 2$ Kanten in T' . T hat genau eine Kante mehr als T' , denn beim Entfernen eines Blattes entfernt man genau eine Kante. Daraus folgt, dass genau $n - 1$ Kanten in T enthalten sind.

(b) \Rightarrow (a): Da es zwischen je 2 Knoten in G einen Weg gibt, ist G zusammenhängend. Wenn G einen Kreis C mit $v, u \in C$ hätte, dann gäbe es 2 verschiedene Wege zwischen u und v , was im Widerspruch zur Annahme steht.

(c) \Rightarrow (a): Wir müssen nur zeigen, dass G kreisfrei ist. Wenn es in G einen Kreis gäbe, könnten wir daraus eine beliebige Kante entfernen und der Graph bliebe weiterhin zusammenhängend. Das ist ein Widerspruch zum minimalen Zusammenhang von G .

(d) \Rightarrow (a): Wir müssen zeigen, dass G zusammenhängend ist. Wenn dem nicht so wäre, dann würde G aus mindestens 2 Zusammenhangskomponenten K_1 und K_2 bestehen. Wir könnten dann zwischen einem beliebigen Knoten von K_1 und einem beliebigen Knoten von K_2 eine Kante hinzufügen, ohne dass ein Kreis entsteht. Widerspruch.

(e) \Rightarrow (a): Nehmen wir an, es gibt einen Kreis C in G . Wenn wir eine beliebige Kante $e \in C$ aus G entfernen, bleibt G zusammenhängend, da man jeden Weg über e auch über den Rest von C laufen kann. Auf diese Weise können wir alle Kreise aus G entfernen und erhalten einen zusammenhängenden kreisfreien Graphen G' , d.h. G' ist ein Baum. Dann muss G' aber $n - 1$ Kanten enthalten; da aber G schon $n - 1$ Kanten hatte und G' mindestens eine Kante weniger als G hat, führt das zu einem Widerspruch. Also war G schon zu Beginn kreisfrei.

Aufgabe 5. (Vom Verhältnis von Bäumen und Wäldern)

(2+2 Punkte)

Definition. Sei $G = (V, E)$ ein Graph mit Gewichten $w : E \rightarrow \mathbb{R}$. Die Aufgabe, einen aufspannenden Baum von G mit minimalem Gewicht zu finden, oder zu entscheiden, dass G keinen Spannbaum hat, wird als *minimales Spannbaumproblem* bezeichnet.

Definition. Sei $G = (V, E)$ ein Graph mit Gewichten $w : E \rightarrow \mathbb{R}$. Die Aufgabe, einen Wald in G mit maximalem Gewicht zu finden, wird als *maximales gewichtetes Waldproblem* bezeichnet.

Die Abbildungen 1 und 2 zeigen beispielhaft einen Graphen G und den zugehörigen Wald maximalen Gewichts.

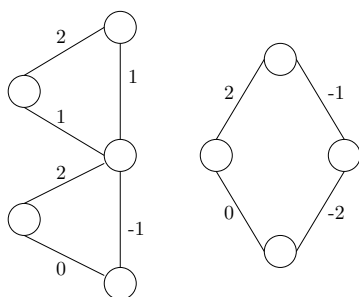


Abbildung 1: Graph $G = (V, E)$

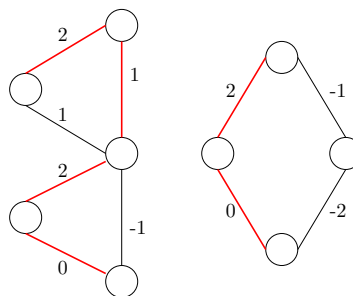


Abbildung 2: Der maximal gewichtete Wald entspricht den roten Kanten.

Zeigen Sie, dass das maximal gewichtete Waldproblem und das minimale Spannbaumproblem äquivalent sind.¹ Das heißt,

1. zeigen Sie, dass man mit einem Algorithmus für minimal aufspannende Bäume auch einen Wald maximalen Gewichts bestimmen kann;
2. zeigen Sie, dass man mit einem Algorithmus zur Bestimmung maximaler Wälder auch einen minimal aufspannenden Baum berechnen kann.

¹Damit ist nicht gemeint, dass die Lösungen der beiden Probleme auf jeder Instanz identisch sind.

Lösung:

1. Gegeben ist ein Algorithmus zur Bestimmung eines minimal spannenden Baumes. Wir benutzen diesen um einen Wald maximalen Gewichts zu bestimmen.

Gegeben ist ein Graph $G = (V, E)$ mit Kantengewichten c_e .

Alternative 1: Wir konstruieren uns den vollständigen Graphen auf V ($K_n = (V, E_n)$) mit den Kantengewichten c'_e wie folgt:

$$\begin{aligned} c'_e &= -c_e & \text{für } e \in E \text{ mit } c_e \geq 0 \\ c'_e &= M & \text{für } e \in E_n \setminus \{e \in E \mid c_e \geq 0\} \end{aligned}$$

mit $M = n(\max\{|c_e|, e \in E\} + 1)$.

Sei B ein minimal aufspannender Baum in K_n bzgl. der Gewichtsfunktion c' , dann ist $F = B \setminus \{e \in B \text{ mit } c'_e = M\}$ ein Wald maximalen Gewichts in G .

Da F ein Teilgraph des Baumes B ist, ist F ein Wald. Aus $c_{e_1} < c_{e_2}$ genau dann, wenn $c'_{e_1} = -c_{e_1} > -c_{e_2} = c'_{e_2}$ folgt, dass F ein maximaler Wald ist, wenn B ein minimal aufspannender Baum ist.

Alternative 2: Anstelle von K_n können wir auch einen Graphen $G' = (V, E')$ mit Bewertung c' konstruieren durch

- Entferne alle Kanten e aus G mit $c_e < 0$.
- Setze $c'_e = -c_e$ für alle verbleibenden Kanten.
- Füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass g' zusammenhängend ist.

Sei B ein minimal aufspannender Baum in G' bzgl. c' . Dann ist $F = B \setminus K$ ein Wald maximalen Gewichts in G .

2. Gegeben ist ein Algorithmus zur Bestimmung eines Wald maximalen Gewichts. Wir benutzen diesen um einen minimal spannenden Baum zu bestimmen.

Gegeben ist ein Graph $G = (V, E)$ mit Kantengewichten c_e . Wir setzen $M = \max\{|c_e|, e \in E\} + 1$ und $c'_e = M - c_e$.

Bestimme in G einen maximalen Wald W bzgl. der Gewichtsfunktion c' .

Falls G zusammenhängend ist, ist W ein aufspannender Baum. Denn: Angenommen W sei kein aufspannender Baum, dann gibt es eine Kante $e \in E$ so, dass $W' = W \cup \{e\}$ ein Baum ist und $c'(W') > c'(W)$ gilt, da $c'_e > 0$ ist. Da der Algorithmus den maximalen Wald bestimmt, hätte er in diesem Fall W' und nicht W bestimmt. Demnach ist W also ein aufspannender Baum.

Falls W nicht zusammenhängend ist, so ist auch G nicht zusammenhängend und es gibt keinen aufspannenden Baum.

Wegen $c_{e_1} > c_{e_2}$ genau dann, wenn $c'_{e_1} = M - c_{e_1} < M - c_{e_2} = c'_{e_2}$, ist W minimal.

Aufgabe 6. (Knobelaufgabe)

(4 Punkte)

Ein Mann soll einen Wolf, eine Ziege und einen Korb Kohl über einen Fluss transportieren. Er hat jedoch nur ein Boot mit zwei Plätzen zur Verfügung. Ist es möglich, alle drei sicher auf das andere Ufer zu bringen?

Beachte, sowohl der Wolf und die Ziege als auch die Ziege und der Korb Kohl dürfen nie allein auf einer Seite des Flusses sein.

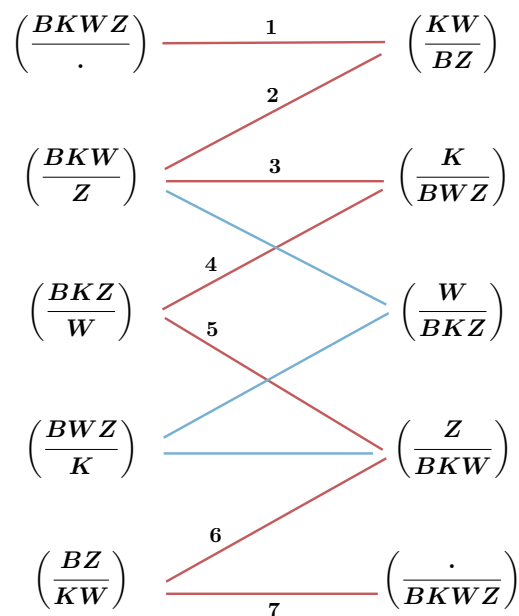
Formuliere dieses Problem als ein Kürzeste-Wege-Problem. Falls es eine Lösung gibt, wie oft muss der Mann den Fluss überqueren?²

Lösung:

Wir bilden einen Zustandsgraphen und suchen dann einen kürzesten Weg vom Anfangszustand zum Endzustand:

Sei $S = \{M, W, Z, K\}$, wobei M, W, Z und K jeweils den Mann, den Wolf, die Ziege und den Kohl repräsentieren sollen. Ein Zustand für dieses Problem soll ein Paar (X, Y) sein, wobei $\{X, Y\}$ eine Partition von S ist. X soll angeben, was noch auf der Ausgangsseite, und Y , was bereits auf der Zielseite des Flusses ist. Einen Zustand nennen wir zulässig, falls weder der Wolf und die Ziege noch die Ziege und der Kohl auf einer Seite allein sind. Wir konstruieren einen gerichteten Graphen $G = (Z, E)$, wobei Z die Menge aller zulässigen Zustände darstellen soll. Es soll eine gerichtete Kante von einem Zustand Z_1 zu einem Zustand Z_2 geben, falls der Mann mit einer Bootfahrt den Zustand Z_1 in den Zustand Z_2 überführen kann. Das Gewicht jeder Kante setzen wir gleich 1. Der Anfangszustand ist (S, \emptyset) und der Endzustand ist (\emptyset, S) . Wir können jetzt einfach einen kürzesten Weg in G vom Anfangszustand zum Endzustand suchen. In unserem Fall hat der kürzeste Weg die Länge 7, so dass sieben Überquerungen notwendig sind, um Wolf, Ziege und Kohl überzusetzen.

Folgender Graph gibt alle Möglichkeiten an:



Ein kürzester Weg vom linken oberen zum rechten unteren Knoten in diesem Graphen entspricht folgender Lösung:

1. Der Bauer rudert mit der Ziege rüber.
2. Der Bauer kommt alleine zurück.
3. Der Bauer nimmt den Wolf mit rüber.
4. Der Bauer kommt mit der Ziege zurück.
5. Der Bauer nimmt den Kohl mit.
6. Der Bauer fährt alleine zurück.
7. Der Bauer nimmt die Ziege mit. Jetzt sind alle auf der anderen Seite des Flusses.

Aufgabe 7. (Das Prinzip von Inklusion und Exklusion)

(2 Punkte)

²Motivation: <https://www.youtube.com/watch?v=bY8phNV5RU0>

Das *Prinzip von Inklusion und Exklusion*³ findet in der Kombinatorischen Optimierung Anwendung und dient zur Bestimmung der Mächtigkeit einer Menge.

Lösen Sie mit dem Prinzip von Inklusion und Exklusion das folgende Problem: Ein Club hat 27 Mitglieder, die entweder männlich oder weiblich, entweder Erwachsene oder Jugendliche, bzw. entweder Vollmitglied oder Teilmitglied sind. Genau 14 Mitglieder sind männlich, genau 20 sind erwachsen und genau 14 sind Vollmitglied. Genau 10 sind männliche Erwachsene, genau 6 männliche Vollmitglieder gibt es und genau 13 erwachsene Vollmitglieder. Ferner sind genau 5 der männlichen Erwachsenen Vollmitglied. Wieviele weibliche Jugendliche im Club sind Teilmitglied?

Lösung:

Sei C die Menge der Mitglieder des Clubs, M die Menge der Männer, E die Menge der Erwachsenen und V die Menge der Vollmitglieder. Laut Aufgabenstellung gilt $|C| = 27$, $|M| = 14$, $|E| = 20$, $|V| = 14$, $|M \cap E| = 10$, $|M \cap V| = 6$, $|E \cap V| = 13$ und $|M \cap E \cap V| = 5$. Gesucht ist $|C \setminus (M \cup E \cup V)|$. Dies ist nach dem Prinzip von Inklusion und Exklusion gleich

$$\begin{aligned}|C \setminus (M \cup E \cup V)| &= |C| - |M| - |E| - |V| + |M \cap E| + |M \cap V| + |E \cap V| - |M \cap E \cap V| \\&= 27 - 14 - 20 - 14 + 10 + 6 + 13 - 5 \\&= 3.\end{aligned}$$

Aufgabe 8. (Programmieraufgabe: Kruskal)

(3 Punkte)

Laden Sie die Datei `ProgAufgabe04.py` aus Studon herunter und fügen Sie sie in PyCharm ein.

In dieser Programmieraufgabe werden Sie den Algorithmus von Kruskal implementieren, wofür wir jeweils eine Abwandlung von Tiefensuche und Quicksort vom letzten Blatt benötigen.

1. Zuerst passen Sie Quicksort auf das Sortieren von Listen von Graph-Kanten nach deren Gewicht an:
 - Wir verwenden nun Graphen mit gewichteten Kanten (wie man solche erstellt entnehmen sie dem Test-Code der Datei `ProgAufgabe04.py`). Sei `graph` ein `networkx`-Graph. Während Sie mit `graph.edges()` eine Liste der Kanten eines Graphen erhalten haben, erhalten Sie mit `E = graph.edges(data=True)` eine Liste `E` der Kanten eines Graphen inklusive ihrer Attribute. Für eine Kante `e` in `E` sind dabei `e[0]` und `e[1]` die Endknoten und `e[2]` ist das Dictionary, das die Attribut-Werte von `e` enthält. Mit `e[2]['weight']` erhalten sie das Gewicht der Kante. Insgesamt erhalten sie mit `E[i][2]['weight']` das Gewicht der `i`-ten Kante in der Liste `E`.
 - Nutzen Sie dies, um die zwei fehlenden Stellen in der Methode `quicksort_edges(edges, l, r)` zu ergänzen. Nun werden nicht mehr `edges[i]` und `edges[j]` verglichen, sondern die Gewichte der `i`-ten und `j`-ten Kante.
2. Nun benötigen wir auf Tiefensuche basierende Methoden, um alle Knoten zu berechnen, die von einem gegebenen Knoten aus (direkt oder indirekt) erreichbar sind.
 - Implementieren Sie zunächst die Methode `checkNeighbor(G, w, list)`, die überprüft, ob `w` schon in `list` enthalten ist. Ist dies nicht der Fall, wird `w` in `list` aufgenommen (mittels `list.append()`) und `checkNeighbor` bei allen Nachbarn von `w` aufgerufen.
 - Implementieren Sie nun die Methode `accessible_nodes(G, v)`, die eine Liste aller von aus `v` direkt oder indirekt erreichbaren Knoten ausgibt (mit `list = []` initialisieren sie eine leere Liste).
3. Nun implementieren Sie den Algorithmus von Kruskal in der Methode `kruskal(graph)`. Diese erhält als Eingabe einen gewichteten Graphen `graph` (der nicht verändert werden darf!) und gibt einen neuen Graphen `tree` aus, der ein minimaler Spannbaum von `graph` ist:
 - Neuen Graphen erstellen.
 - Liste der gewichteten Kanten des gegebenen Graphen erstellen.

³https://de.wikipedia.org/wiki/Prinzip_von_Inklusion_und_Exklusion

- Liste sortieren (`quicksort_edges`).
- Liste durchiterieren: Falls einer der Endknoten der Kante noch nicht in `tree` ist oder keine Verbindung zwischen den Endknoten in `tree` besteht (`accessible_nodes`), Kante samt Endknoten hinzufügen.

HINWEIS: Mit `tree = nx.Graph()` erstellen sie einen neuen Graphen, mit `tree.add_node` bzw. `tree.add_edge` fügen sie Knoten und Kanten hinzu. Falls ein Knoten bereits vorhanden ist, wirft `add_node` keinen Fehler, sondern tut einfach nichts).

Lösung:

Die Musterlösung der Programmieraufgabe finden Sie im StudOn in der Datei `ProgAufgabe04_Loesung.py`.

Gruppe 1: Abgabe der Hausaufgaben am 21.11.2017 in der Übung.
 Gruppe 2+3: Abgabe der Hausaufgaben am 22.11.2017 in der jeweiligen Übung.