



Wintersemester 2017/2018

Lineare und kombinatorische Optimierung

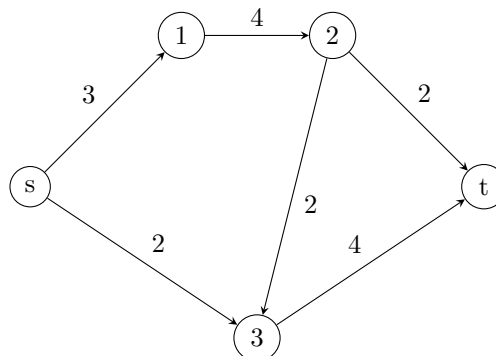
Übungsblatt 7

Gruppenübungen

Aufgabe 1. (Push-Relabel-Algorithmus)

(0 Punkte)

Gegeben sei der folgende Graph. Berechnen Sie mit dem Push-Relabel-Algorithmus von Goldberg und Tarjan einen maximalen Fluss zwischen den Knoten s und t .

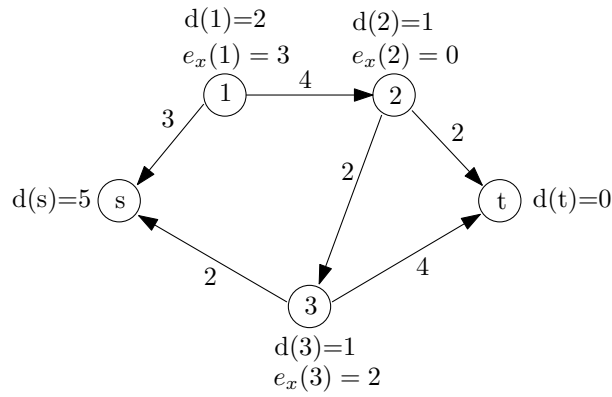


Lösung:

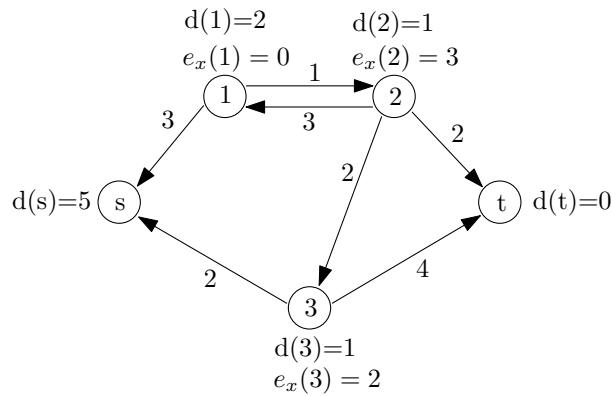
Sei $D = (V, A)$ der gegebene Graph. Wir initialisieren den Fluss x und die Markierung d :

- $x_{s1} = 3, x_{s3} = 2$, für alle anderen Bögen $ij \in A \setminus \{s1, s3\}$ setzen wir $x_{ij} = 0$.
- $d(s) = 5$, für alle anderen Knoten verwenden wir (Backward-)BFS von t aus und erhalten $d(1) = 2, d(2) = 1, d(3) = 1$ und $d(t) = 0$.

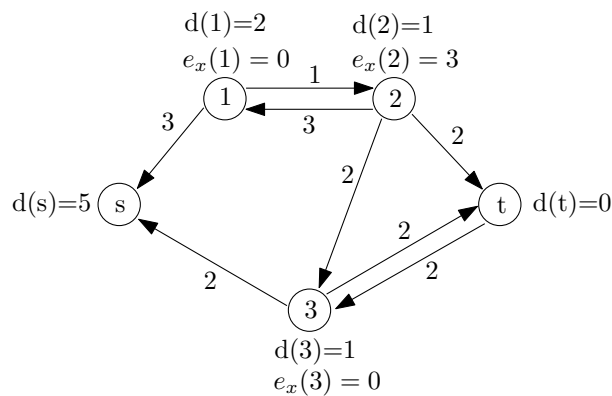
Wir bekommen folgenden Residualgraphen. An den Knoten stehen die Markierung $d(\cdot)$ sowie die Überschusskapazität $e_x(\cdot)$ (soweit vorhanden).



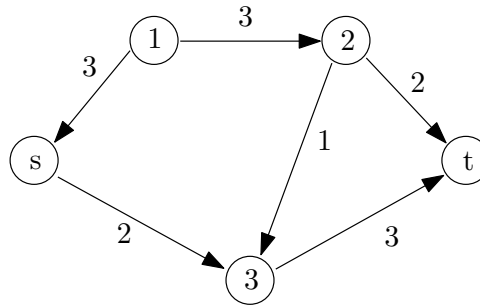
Wir wählen z.B. den aktiven Knoten 1, schieben $\delta = 3$ von Knoten 1 nach Knoten 2 und erhalten $x_{12} = 3$. Der Residualgraph sieht nun folgendermaßen aus:



Wir wählen nun z.B. Knoten 3, schieben $\delta = 2$ von Knoten 3 nach Knoten t und erhalten $x_{3t} = 2$. Der Residualgraph sieht nun folgendermaßen aus:



Wir wählen nun Knoten 2, schieben $\delta = 2$ von Knoten 2 nach Knoten t und erhalten $x_{2t} = 2$. Der Knoten 2 hat immer noch Überschusskapazität $e_x(2) = 1$. Wir wählen also noch mal Knoten 2, relabeln mit $d(2) = 2$ und schieben $\delta = 1$ von Knoten 2 nach Knoten 3. Es ist $x_{23} = 1$. Nun ist Knoten 3 wieder aktiv. Wir wählen also Knoten 3 und schieben ohne zu relabeln Fluss $\delta = 1$ von Knoten 3 nach Knoten t. Es ist $x_{3t} = 3$. Nun gibt es keine aktiven Knoten mehr und wir sind fertig. Der maximale Fluss hat Wert 5:



Aufgabe 2. (Aktualisierung eines Flusses)

(0 Punkte)

Es sei $D = (V, A)$ ein Digraph mit Quelle s und Senke t und ganzzahligen Kapazitäten $c_a > 0$ für alle $a \in A$. Außerdem sei ein ganzzahliger maximaler Fluss x in D gegeben. Nun wird die Kapazität eines einzelnen Bogens

1. um 1 erhöht,
2. um 1 verringert.

Geben Sie einen Algorithmus der Komplexität $\mathcal{O}(|V| + |A|)$ an, der einen maximalen Fluss in dem jeweiligen veränderten Netzwerk bestimmt.

Lösung:

1. Wenn die Kapazität eines Bogens um eine Einheit erhöht wird, braucht man lediglich genau einmal nach einem augmentierenden Weg zu suchen, um den Fluss zu re-maximieren, da sich der Flusswert um höchstens eine Einheit erhöhen kann. Dies ist z.B. mit DFS in $\mathcal{O}(|V| + |A|)$ möglich.
2. Wenn die Kapazität eines Bogens um eine Einheit verringert wird, können 2 Fälle auftreten:
 1. Der Fluss auf dem Bogen bleibt zulässig: In diesem Fall ist nichts zu tun.
 2. Der Fluss auf dem Bogen überschreitet dessen Kapazität um genau eine Einheit: Sei (u, v) der betrachtete Bogen; wir suchen einen Weg P_1 von s nach u und einen Weg P_2 von v nach t . Diese beiden Wege sollen disjunkt sein und nur Bögen beinhalten, in denen ein positiver Fluss fließt (also zwangsweise mindestens eine Einheit). Aufgrund des Satzes über die Flusszerlegung (Lemma 6.3) muss es stets solche Wege geben. Diese Wege können mit DFS (oder BFS) in $\mathcal{O}(|V| + |A|)$ bestimmt werden. Wir reduzieren den Fluss entlang $P_1 - (u, v) - P_2$ um eine Einheit. Dadurch stellen wir wieder einen zulässigen Fluss her. Es könnte nun sein, dass man den Fluss wieder (durch einen anderen Weg) um maximal eine Einheit erhöhen kann. Deswegen suchen wir genau einmal nach einem augmentierenden Weg und augmentieren ggf. den Fluss. Da dies auch in $\mathcal{O}(|V| + |A|)$ möglich ist, erhalten wir die gewünschte Gesamtlaufzeit.

Aufgabe 3. (Konvexität)

(0 Punkte)

Gegeben sei ein Netzwerk $(D = (V, A), c, s, t)$. Zeigen Sie, dass die Menge der zulässigen Flüsse

$$\{x : A \rightarrow \mathbb{R} : x \text{ ist Fluss in } D\}$$

konvex ist.

Lösung:

Wir müssen zeigen, dass wenn x, y zulässige Flüsse sind, dann ist auch $\lambda x + (1 - \lambda)y$ für $\lambda \in [0, 1]$ ein zulässiger Fluss.

Seien also $x, y : A \rightarrow \mathbb{R}$ zulässige Flüsse in D . Also

- $0 \leq x_a, y_a \leq c_a$ für alle $a \in A$

- $\sum_{a \in \delta^-(v)} x_a = \sum_{a \in \delta^+(v)} x_a \quad \forall v \in V \setminus \{s, t\}$ (Analog für y .)

Für $\lambda \in [0, 1]$ definieren wir eine neue Funktion

$$z := \lambda x + (1 - \lambda)y : A \rightarrow \mathbb{R} \text{ durch } z_a = \lambda x_a + (1 - \lambda)y_a$$

. Wir zeigen, dass auch z ein zulässiger Fluss ist:

- $\lambda x_a + (1 - \lambda)y_a \leq \lambda c_a + (1 - \lambda)c_a = c_a$, also $z_a \leq c_a$ für alle $a \in A$.
- $\lambda x_a + (1 - \lambda)y_a \geq \lambda \cdot 0 + (1 - \lambda) \cdot 0 = 0$, also $z_a \geq 0$ für alle $a \in A$.
-

$$\begin{aligned} \sum_{a \in \delta^-(v)} z_a &= \sum_{a \in \delta^-(v)} (\lambda x_a + (1 - \lambda)y_a) \\ &= \lambda \sum_{a \in \delta^-(v)} x_a + (1 - \lambda) \sum_{a \in \delta^-(v)} y_a \\ &= \lambda \sum_{a \in \delta^+(v)} x_a + (1 - \lambda) \sum_{a \in \delta^+(v)} y_a \\ &= \sum_{a \in \delta^+(v)} (\lambda x_a + (1 - \lambda)y_a) \\ &= \sum_{a \in \delta^+(v)} z_a \end{aligned}$$

Hausübungen

Aufgabe 4. (Eindeutigkeit von maximalen Flüssen und minimalen Schnitten)

(1+1+1 Punkte)

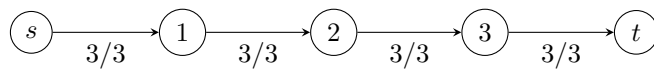
Gegeben sei ein Netzwerk $N = (D, c, s, t)$. Sind die folgenden Aussagen wahr oder falsch? Begründen Sie Ihre Antwort oder finden Sie ein Gegenbeispiel.

1. Ist der maximale Fluss in N eindeutig, existiert auch ein eindeutiger minimaler (s, t) -Schnitt in N .
2. Ist der minimale (s, t) -Schnitt in N eindeutig, existiert auch ein eindeutiger maximaler Fluss in N .
3. Sei x ein maximaler Fluss und $\delta^+(W)$ ein minimaler Schnitt in N . Dann ist der Fluss auf den Bögen des Schnittes eindeutig, d. h. x_e ist eindeutig für alle $e \in \delta^+(W)$.

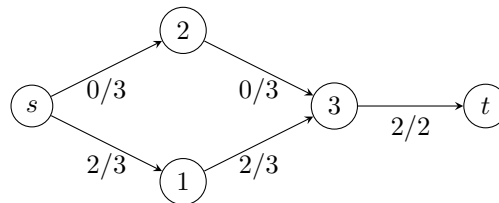
Lösung:

Die ersten beiden Aussagen sind falsch. Gegenbeispiele:

1. Folgendes Netzwerk hat einen maximalen Fluss, aber mehrere minimale Schnitte:



2. Folgendes Netzwerk hat einen minimalen Schnitt, aber mehrere maximale Flüsse:



3. Die dritte Aussage ist richtig. Nach Lemma 6.4 und Korollar 6.5 gilt

$$x(\delta^+(W)) - x(\delta^-(W)) = c(\delta^+(W)).$$

Da $0 \leq x \leq c$ folgt daraus, dass

$$x(\delta^-(W)) = 0 \text{ und } x(\delta^+(W)) = c(\delta^+(W)).$$

Also $x_e = c_e$ für alle $e \in \delta^+(W)$.

Aufgabe 5. (Arbeitsteilung)

(4 Punkte)

Wir betrachten ein Computersystem mit zwei Prozessoren, die unterschiedlich sein können. Wir wollen ein großes Programm auf diesem Computer laufen lassen. Dieses enthält verschiedene Module, die während der Ausführung interagieren. Die Kosten, die bei der Ausführung jedes Moduls auf den Prozessoren entstehen, seien vorher bekannt und können für die beiden Prozessoren unterschiedlich sein (durch Unterschiede im Speicher, Geschwindigkeit, ...). Wir notieren diese mit α_i bzw. β_i für Modul i und Prozessor 1 bzw. 2. Die Zuweisung verschiedener Module zu verschiedenen Prozessoren kann zu weiteren Kosten führen, da Interprozessor-Kommunikation notwendig werden kann. Seien c_{ij} die Kosten, die entstehen, wenn Modul i und j auf verschiedenen Prozessoren laufen.

Finden Sie auf graphentheoretischem Weg eine Zuweisung der Module zu den beiden Prozessoren, so dass die Gesamtkosten aus Rechenkosten und Interaktionskosten minimal sind. Die folgende Tabelle gibt Beispieldaten. Bestimmen Sie eine kostenminimale Zuweisung.

i	1	2	3	4
α_i	6	5	10	4
β_i	4	10	3	8

c_{ij}	1	2	3	4
1	0	5	0	0
2	5	0	6	2
3	0	6	0	1
4	0	2	1	0

Tipp: Sie können CATBox verwenden um einen maximalen Fluss bzw. minimalen Schnitt zu berechnen.

Lösung:

Wir formulieren das Problem als ein min-cut Problem auf einem ungerichteten Graphen. Wir definieren eine Quelle s , die Prozessor 1 repräsentiert, und einen Zielknoten t für Prozessor 2, außerdem Knoten i für jedes Modul i . Für jedes Modul i führen wir eine Kante (s, i) mit Kapazität α_i und eine Kante (i, t) mit Kapazität β_i ein. Außerdem führen wir eine Kante (i, j) mit Kapazität c_{ij} ein, wenn Modul i mit Modul j während der Programmausführung interagiert. Ein (s, t) -Schnitt in diesem Netzwerk entspricht einer kostenminimalen Zuweisung. Für die gegebenen Daten ist dies Modul 1, 2 und 3 auf Prozessor 2 und Modul 4 auf Prozessor 1.

ANMERKUNG: Der minimale Schnitt trennt die Knotenmengen $S = \{s, 1, 2, 3\}$ und $\bar{S} = \{t, 4\}$. Dies bedeutet nun aber, dass es günstig ist, Modul 1, 2 und 3 auf Prozessor 2, von t repräsentiert, laufen zu lassen, da die kostengünstigen Kanten, die geschnitten werden, gerade die Kosten angeben, die nötig sind um die Module auf diesen Prozessoren laufen zu lassen.

Der von CATBox berechnete minimale Schnitt ist in Abbildung 1 dargestellt.

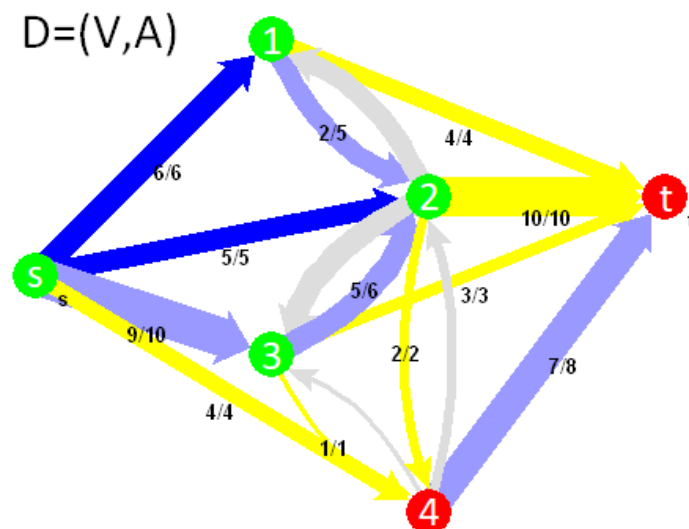
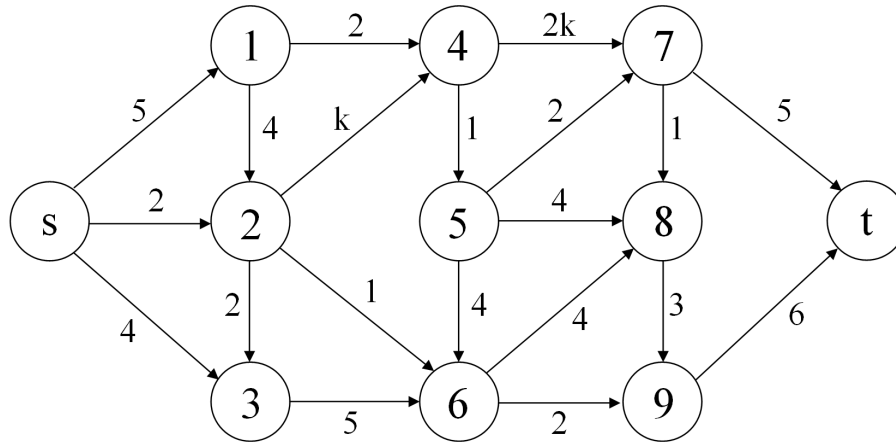


Abbildung 1: Von CATBox berechneter minimaler Schnitt (gelb)

Aufgabe 6. (Minimale Schnitte)

(1+3+1 Punkte)

Es sei die folgende Graphenfamilie $(D_k = (V, A_k))_{k \in \mathbb{N}_0}$ gegeben:



1. Zeigen Sie: Für kein $k \in \mathbb{N}_0$ ist $\delta(\{s, 1, 2, 3, 5, 6\})$ ein minimaler (s, t) -Schnitt.
2. Geben Sie (in Abhängigkeit von $k \in \mathbb{N}_0$) einen minimalen (s, t) -Schnitt in D_k und den maximalen Flusswert eines (s, t) -Flusses an.
Benutzen Sie CATBox zum Finden der Schnitte und berechnen Sie jeweils mindestens einen Schnitt mit dem Algorithmus von Ford-Fulkerson bzw. mit dem Push-Relabel-Algorithmus.
3. Zeigen Sie, dass der (s, t) -Schnitt in D_2 , den Sie in der 2. Teilaufgabe angegeben haben, minimal ist.

Lösung:

1. Wegen $\delta(\{s, 1, 2, 3, 6\}) = \{(1, 4), (2, 4), (6, 8), (6, 9)\} \subsetneq \{(1, 4), (2, 4), (5, 7), (5, 8), (6, 8), (6, 9)\} = \delta(\{s, 1, 2, 3, 5, 6\})$ und der Nichtnegativität der Bogenkapazitäten gilt:

$$\sum_{a \in \delta(\{s, 1, 2, 3, 5, 6\})} c_a \geq \sum_{a \in \delta(\{s, 1, 2, 3, 6\})} c_a$$

Daher ist $\delta(\{s, 1, 2, 3, 5, 6\})$ kein minimaler (s, t) -Schnitt.

2. Für $k = 0$ beträgt der maximale Flusswert 6 und $\delta(\{s, 1, 2, 3, 4, 6, 8\})$ ist der minimale (s, t) -Schnitt (vgl. Abbildung 2). Achtung: Beachten Sie, dass die Nummerierungen der Knoten in den folgenden Abbildungen um 1 zur Angabe verschoben sind.

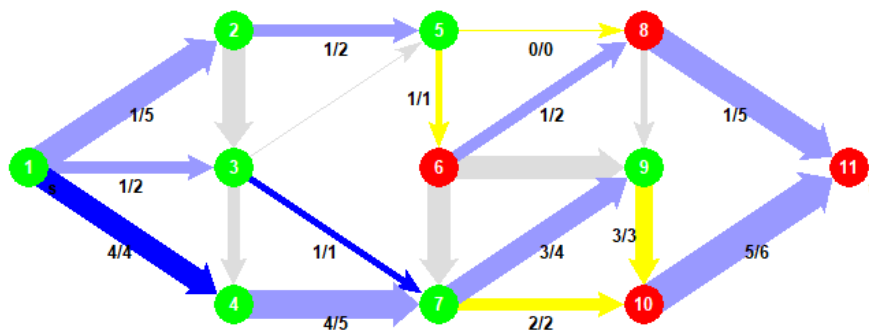


Abbildung 2: Minimaler Schnitt (gelb) in D_0

Für $k \in \{1, 2, 3\}$ ist $\delta(\{s, 1, 2, 3, 6, 8\})$ ein minimaler (s, t) -Schnitt und der maximale Flusswert beträgt $7 + k$ (siehe Abbildung 3).

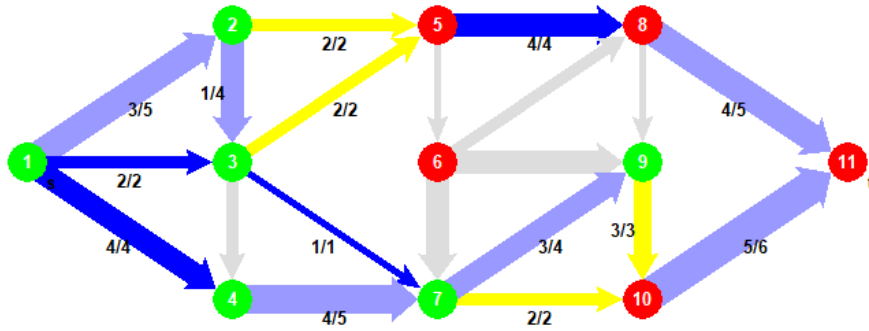


Abbildung 3: Minimaler Schnitt in D_2

Für $k = 4$ ist $\delta(\{s, 1, 2, 3, 4, 5, 6, 7, 8\})$ der minimale (s, t) -Schnitt (vgl. Abbildung 4) mit Wert 10 (unabhängig von k) und, da sich die Bogenkapazitäten für $k > 4$ nur vergrößern, ist dieser Schnitt minimal für alle $k \geq 4$. Der maximale Flusswert beträgt 10.

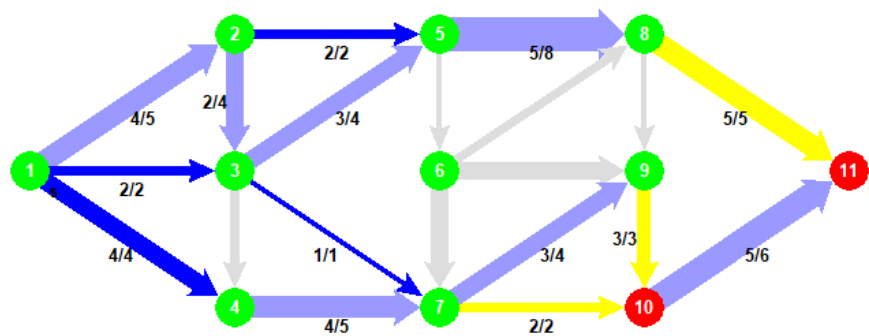


Abbildung 4: Minimaler Schnitt in D_4

3. Nach der zweiten Teilaufgabe existiert ein (s, t) -Schnitt mit Wert 9. Wie man in 3 sehen kann, gibt es auch einen (s, t) -Fluss in D_2 mit Wert 9. Nach Korollar 6.5 gilt dann für den Wert eines maximalen (s, t) -Flusses x :

$$\text{val}(x) \leq 9$$

Ebenso gilt

$$9 \leq \text{val}(x)$$

aufgrund der Maximalität von x .

Mit dem Max-Flow-Min-Cut-Theorem folgt die Minimalität von $\delta(\{s, 1, 2, 3, 6, 8\})$.

Aufgabe 7. (Schnitte im Residualgraphen)

(2 Punkte)

Es sei $N = (D = (V, A), c, s, t)$ ein Flussnetz und x ein beliebiger Fluss in N . Ferner sei $W \subset V$ eine Knotenmenge mit $s \in W$ und $t \notin W$. Zeigen Sie, dass zwischen der Kapazität $c(\delta^+(W))$ des Schnittes in D und der Kapazität $c_x(\delta^+(W))$ des Schnittes im Residualgraphen $G(x)$ folgender Zusammenhang besteht:

$$c_x(\delta^+(W)) = c(\delta^+(W)) - \text{val}(x).$$

Lösung:

$$\begin{aligned}
c_x(\delta^+(W)) &= \sum_{(i,j) \in A(x), i \in W, j \notin W} c_x(i, j) \\
&= \sum_{(i,j) \in A, i \in W, j \notin W} (c(i, j) - x(i, j)) + \sum_{(i,j) \in A, i \notin W, j \in W} x(i, j) \\
&= \underbrace{\sum_{(i,j) \in A, i \in W, j \notin W} c(i, j)}_{c(\delta^+(W))} - \underbrace{\left(\sum_{(i,j) \in A, i \in W, j \notin W} x(i, j) - \sum_{(i,j) \in A, i \notin W, j \in W} x(i, j) \right)}_{\text{val}(x) \quad (\text{vgl. Lemma 6.4})} \\
&= c(\delta^+(W)) - \text{val}(x).
\end{aligned}$$

Aufgabe 8. (Programmieraufgabe: Push-Relabel-Algorithmus)

(1+1+1+1+3 Punkte)

Laden Sie die Datei `ProgAufgabe07.py` aus Studon herunter und fügen Sie sie in ihr PyCharm ein.

In dieser Aufgabe implementieren Sie den Push-Relabel-Algorithmus zum Finden eines maximalen s-t-Flusses. Im ersten Teil implementieren Sie vier Hilfsmethoden für den Algorithmus, im zweiten Teil implementieren Sie den Algorithmus mithilfe der vier Methoden aus Teil 1.

Im Folgenden ist G der Graph, x ein Dictionary, das jeder Kante (v, w) den Fluss von v nach w zuordnet, Gx der Residualgraph $G(x)$, $label$ ein Dictionary, das jedem Knoten ein Label (eine natürliche Zahl) zuordnet und $excess$ ein Dictionary, das jedem Knoten den Überschuss in diesem Knoten, das heißt den Fluss in den Knoten hinein minus den Fluss aus dem Knoten heraus zuordnet.

Hinweis: In dieser Aufgabe wird die Implementierung des Residualgraphen aus der Lösung von Programmieraufgabe 6 verwendet. In der Angabe von Aufgabe 6 stehen alle nötigen Informationen zur Gestalt des konstruierten Dualgraphen. Die Implementierung ist bereits in der Datei `ProgAufgabe07.py` enthalten.

1. Die Hilfsmethoden:

- Implementieren Sie die Methode `find_active_node(G, s, t, excess)`. Die Methode überprüft, ob es einen Knoten v gibt, der
 - weder s noch t ist und
 - Überschuss ungleich 0 hat.

Gibt es einen solche Knoten, gibt die Methode ihn zurück, ansonsten gibt sie `None` zurück.

- Implementieren sie die Methode `find_feasible_edge(Gx, label, v)`. Diese überprüft, ob es im Residualgraphen eine ausgehende Kante (v, w) von v gibt, für die `label[v]=label[w]+1` gilt. Gibt es eine solche Kante, gibt die Methode das Tripel `e, residual, forward` aus, wobei `e = (v, w)` die gefundene Kante und `residual` und `forward` die beiden Kantenattribute der Kante im Residualgraphen sind (vgl. Programmieraufgabe 6). Gibt es keine solche Kante, gibt die Methode das Tripel `None, 0, 0` zurück.
- Implementieren Sie die Methode `push(x, excess, e, push_value, forward)`. Dabei ist `e=(v, w)` eine Kante im Residualgraphen, zusammen mit ihren Kantenattributen `residual` und `forward`. Die Methode erhöht den effektiven Fluss von v nach w um `push_value` (pusht `push_value` von v nach w). Dabei ist Folgendes zu beachten:
 - Ist `forward==True`, so ist `e` ein Vorwärtsbogen in Gx , das heißt `e` entspricht der Kante (v, w) in G und die Methode `push` erhöht den Fluss von v nach w um `push_value`.
 - Ist `forward==False`, so ist `e` ein Rückwärtsbogen in Gx , das heißt `e` entspricht der Kante (w, v) in G und die Methode `push` verringert den Fluss von w nach v um `push_value`.

Wichtig: In beiden Fällen müssen anschließend die Werte `excess[v]` bzw. `excess[w]` um `push_value` angehoben, bzw. reduziert werden.

- Implementieren Sie die Methode `relabel(Gx, label, v)`. Die Methode sucht unter an allen Knoten w , für die es im Residualgraphen eine Kante von v nach w (nicht andersrum) gibt, nach demjenigen w mit dem kleinsten `label[w]` und setzt das Label von v auf `label[w]+1`.

2. Implementieren Sie die Methode `push_relabel(G,s,t)`, die mit dem Push-Relabel-Algorithmus einen maximalen Fluss von `s` nach `t` berechnet. Gehen Sie wie folgt vor:
Initialisieren Sie die 3 Dictionaries `label`, `x` und `excess`:

- `label[s]` ist gleich Knotenanzahl von `G`, alle anderen Labels sind zunächst 0.
- `x[e]` ist gleich Kapazität von `e` für jede aus `s` ausgehende Kante `e`. Alle anderen Flusswerte `x` sind zunächst 0 (dass dieser "Fluss" nicht zulässig ist, macht nichts).
- Setzen Sie `excess` gemäß dem zuvor definierten Pseudofluss (in einem echten Fluss wäre `excess` gleich 0 in allen Knoten außer `s` und `t`).

Der Algorithmus geht nun wie folgt vor:

Wiederhole, solange `find_active_node` einen aktiven Knoten `v` findet:

- Suche mittels `find_feasible_edge` eine aus `v` ausgehende Residualkante (`e,residual,forward`).
- Gibt es `e`, so schiebe (mittels `push`) den Wert `push_value=min(residual,excess[v])` entlang der Kante `e`.
- Gibt es `e` nicht, so wende `relabel` auf `v` an.

Gibt es keinen aktiven Knoten mehr, dann gibt die Methode das Tupel `x,flow_value` zurück, wobei `flow_value` der Wert des Flusses ist.

Tipp: Am Ende kann `flow_value` aus einem geeigneten Eintrag von `excess` ausgelesen werden.

Lösung:

Die Musterlösung der Programmieraufgabe finden Sie im StudOn in der Datei `ProgAufgabe07_Loesung.py`.

Gruppe 1: Abgabe der Hausaufgaben am 12.12.2017 in der Übung.
Gruppe 2+3: Abgabe der Hausaufgaben am 13.12.2017 in der jeweiligen Übung.