Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Moritz Laupichler
Nikolai Maas

3. Übungsblatt zu Algorithmen II im WS 2023/2024

https://algo2.iti.kit.edu/AlgorithmenII_WS23.php
{sanders, moritz.laupichler, nikolai.maas}@kit.edu

Musterlösungen

Aufgabe 1 (*Analyse:* preflow-push *Algorithmus (Wiederholung)*)

Sei durch S, T ein minimaler (s, t) Schnitt gegeben. Zeigen oder widerlegen Sie folgende Eigenschaften der Distanzfunktion (Label) nach Ausführung des *preflow-push*-Algorithmus:

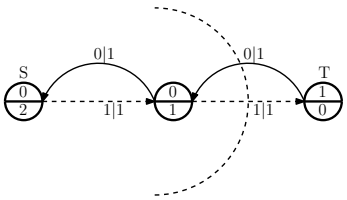
a) $\forall v \in T : d(v) < n$

b) $\forall v \in S : d(v) \geq n$

Musterlösung:

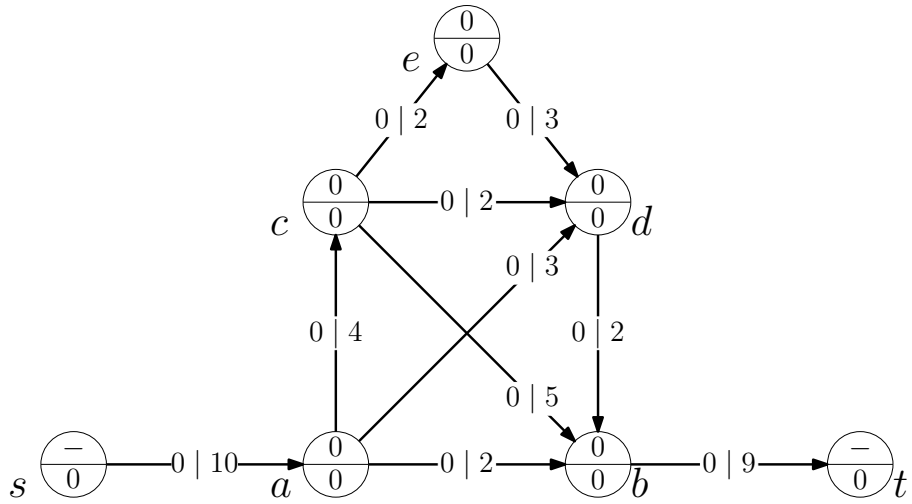
a) Die Aussage ist wahr:
Angenommen es gibt einen Knoten $v \in T$ mit $d(v) \geq n$. Dann existiert im Residualgraph kein Weg von v zu t. Der entsprechende Fluss muss also zurück zu s geschoben werden. Daraus folgt aber auch direkt, dass der gegebene (s, t) Schnitt nicht minimal sein kann, da sonst der zusätzliche Fluss, der über den Schnitt geleitet wurde, zu t gelangen können muss (nach *max-flow-min-cut*-Theorem).

b) Die Aussage ist falsch:
Folgende Abbildung gibt ein Gegenbeispiel an.



Aufgabe 2 (*Rechnen: preflow-push Algorithmus*)

Gegeben sei folgender Flussgraph:



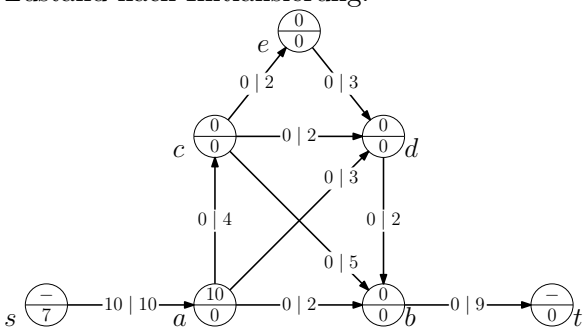
(das Haus vom Nikolaus mit Vorgarten und Hinterhof)

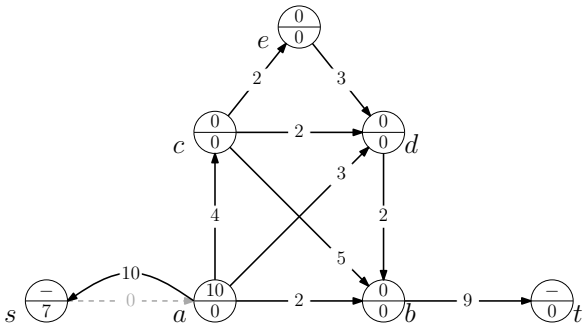
Knotenbeschriftung: Level (unten), Überschuss (oben)
Kantenbeschriftung: Fluss (vorne), Kapazität (hinten)

Bestimmen Sie den maximalen Fluss von s nach t mit dem generischen *preflow-push* Algorithmus.

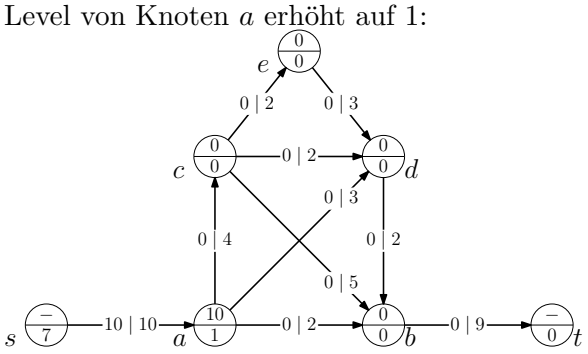
Musterlösung:
Im Folgenden wird der *preflow-push* Algorithmus aus der Vorlesung auf den Flussgraphen angewendet. Aktive Knoten werden zufällig ausgewählt. Es wird bei einem Knoten geblieben bis dessen gesamter Überschuss weggeschoben wurde. Dieser Ablauf ist *nicht* der schnellstmögliche!
Links ist der Zustand des Flussgraphen nach jedem Schritt zu sehen, rechts der des Residualgraphen.

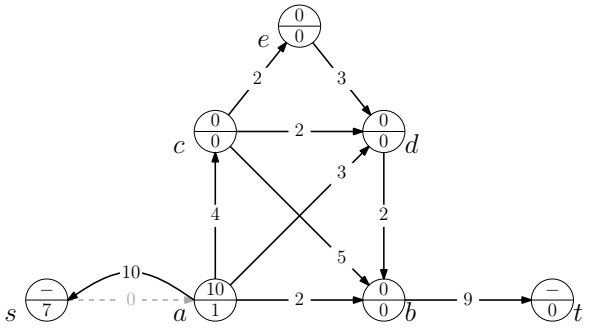
Zustand nach Initialisierung:





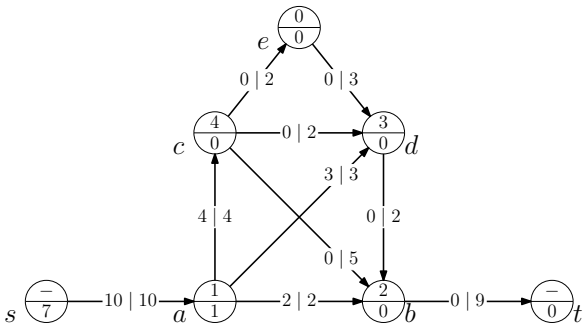
Level von Knoten a erhöht auf 1:

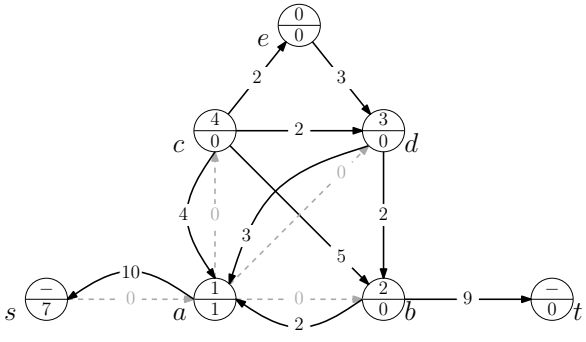




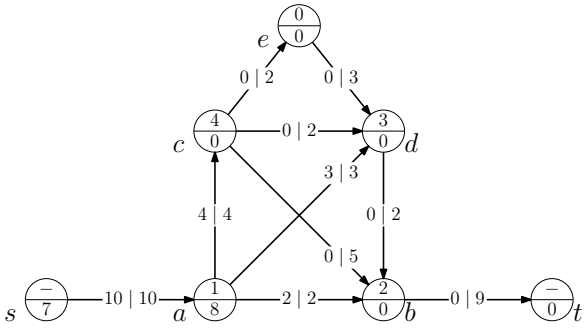
Musterlösung:

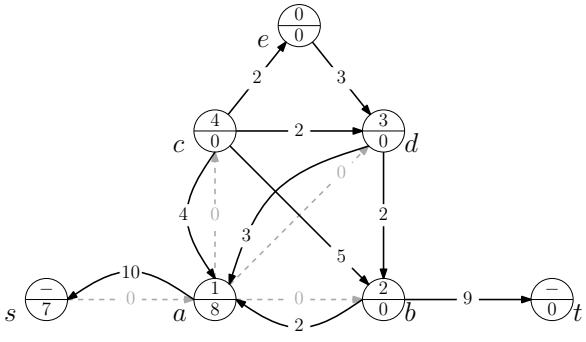
Fluss von a nach b, c und d geschoben:



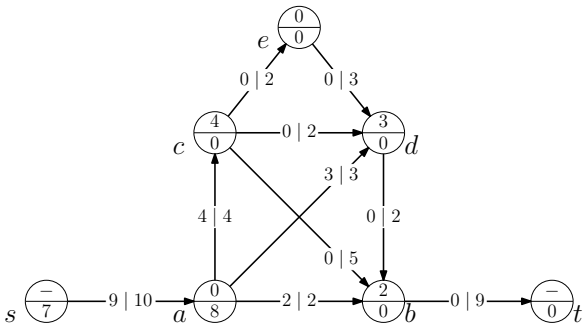


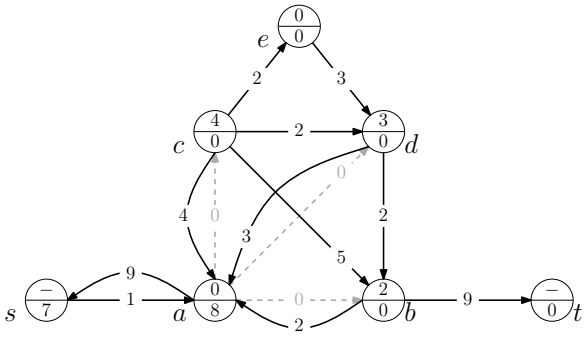
Level von Knoten a erhöht auf 8:



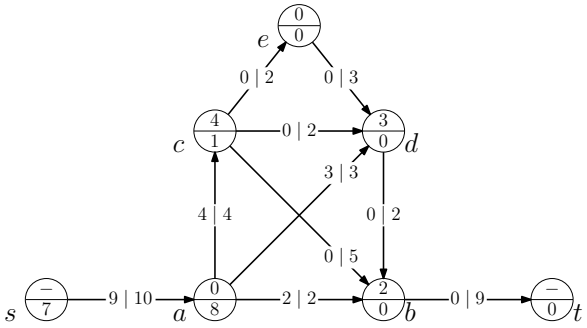


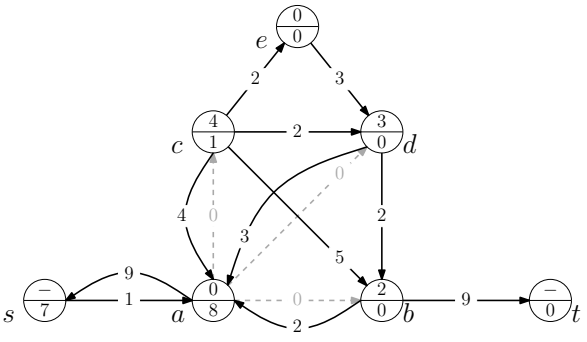
Fluss von a nach s geschoben:





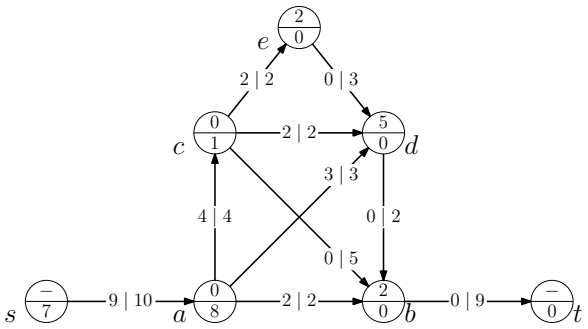
Level von Knoten c erhöht auf 1:

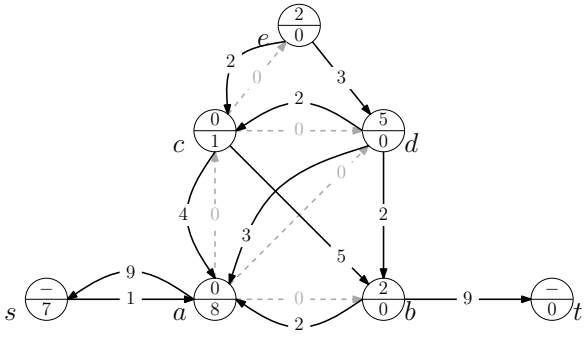




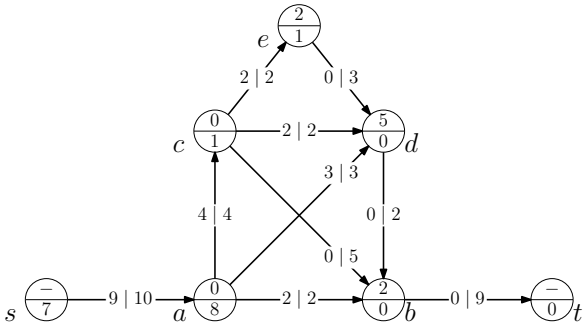
Musterlösung:

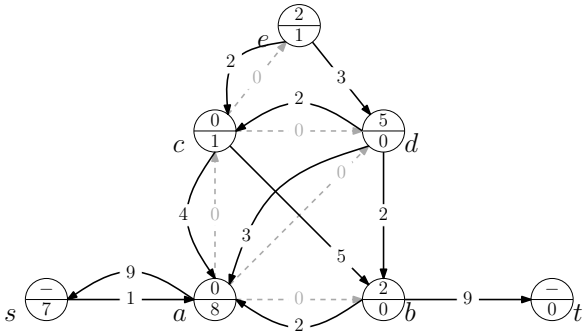
Fluss von c nach d und e geschoben:



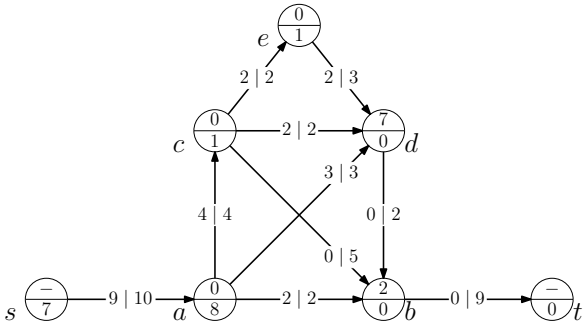


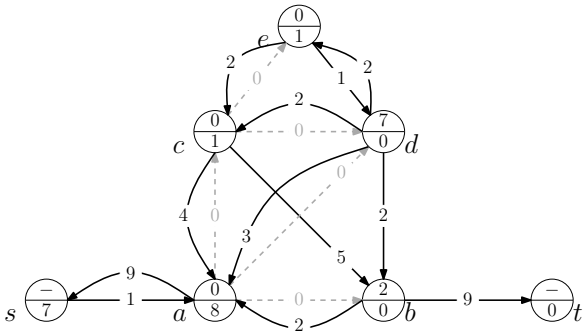
Level von Knoten e erhöht auf 1:



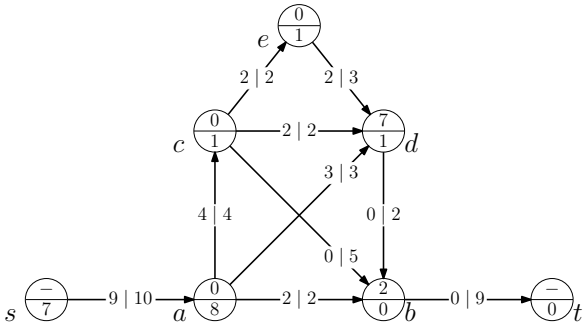


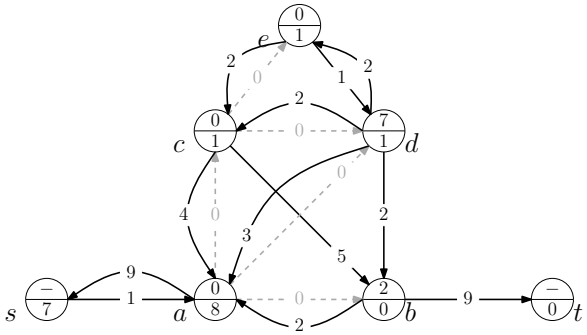
Fluss von e nach d geschoben:





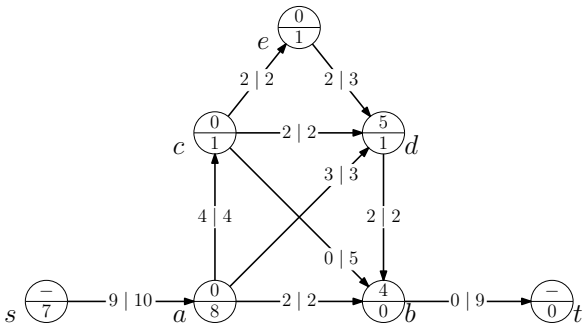
Level von Knoten d erhöht auf 1:

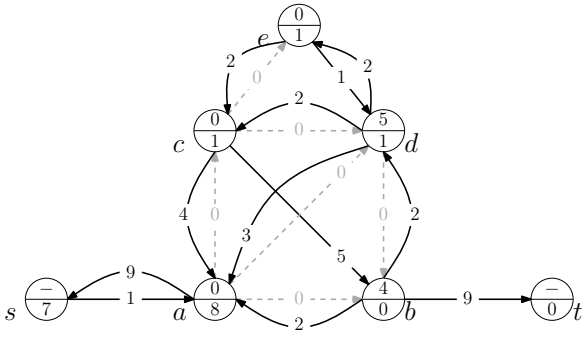




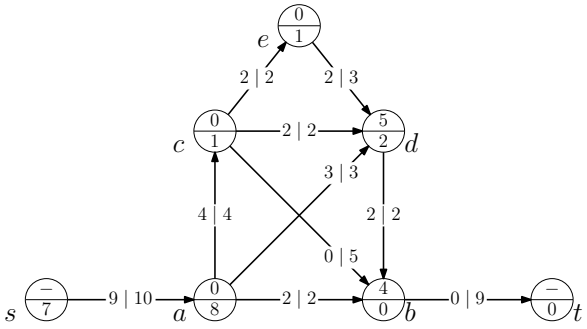
Musterlösung:

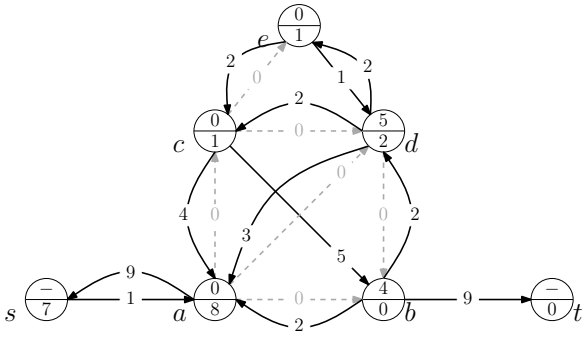
Fluss von d nach b geschoben:



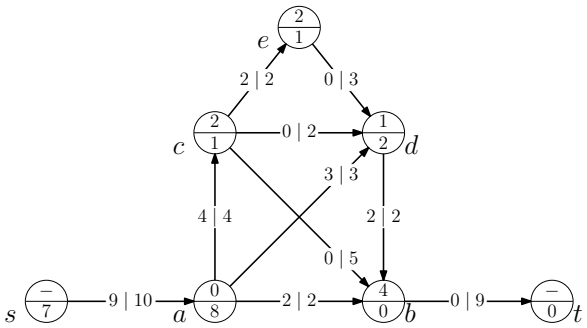


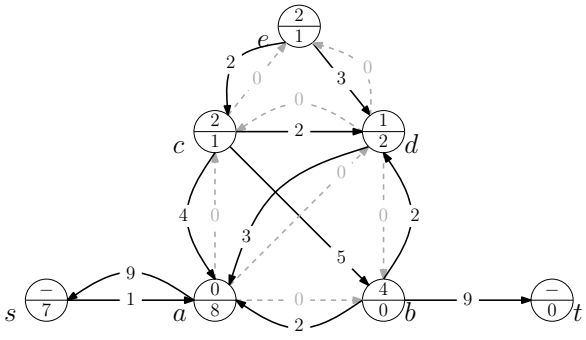
Level von Knoten d erhöht auf 2:



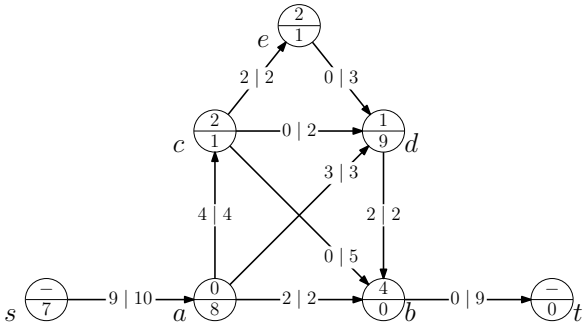


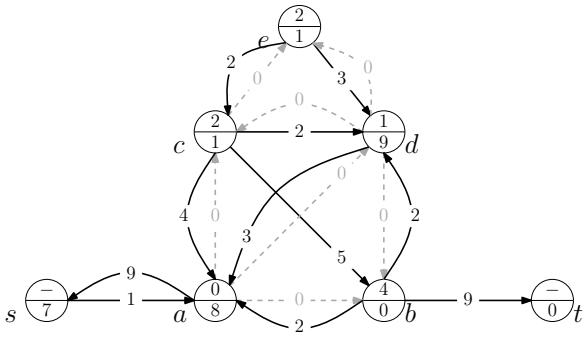
Fluss von d nach c und e geschoben:





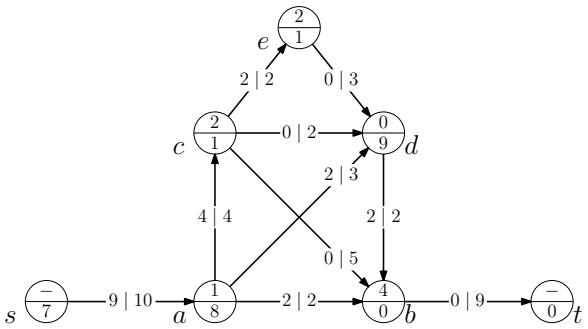
Level von Knoten d erhöht auf 9:

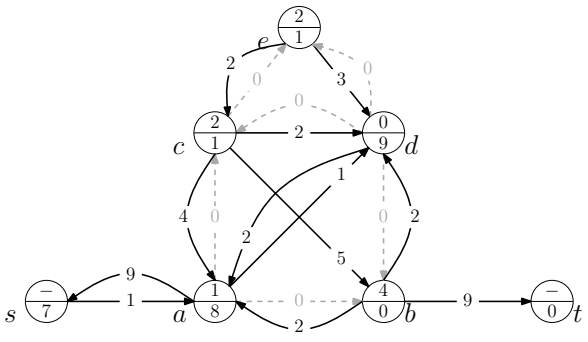




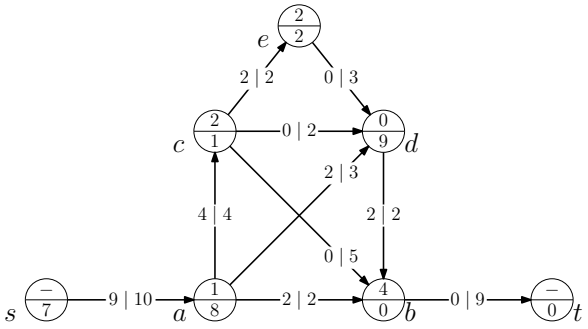
Musterlösung:

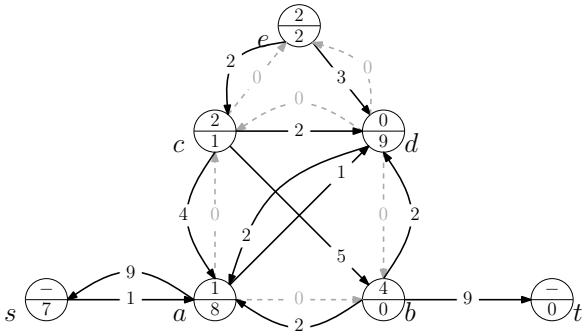
Fluss von d nach a geschoben:



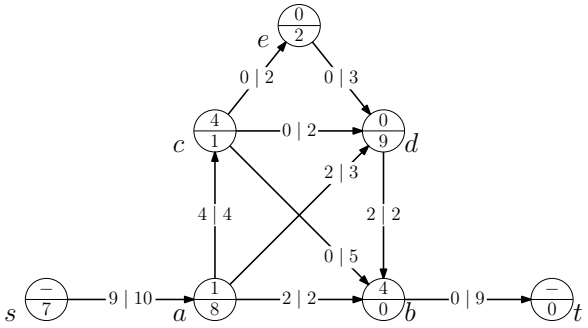


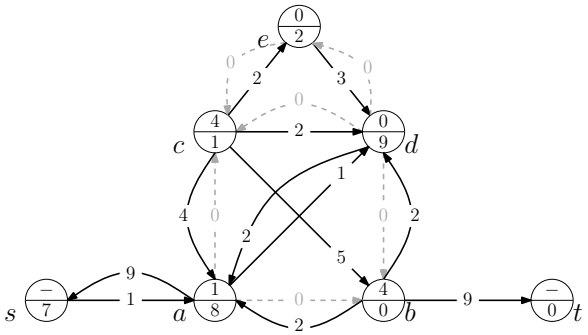
Level von Knoten e erhöht auf 2:



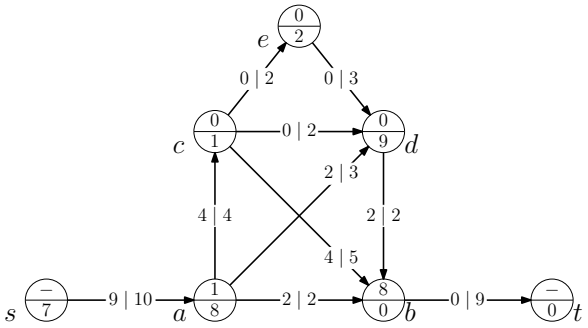


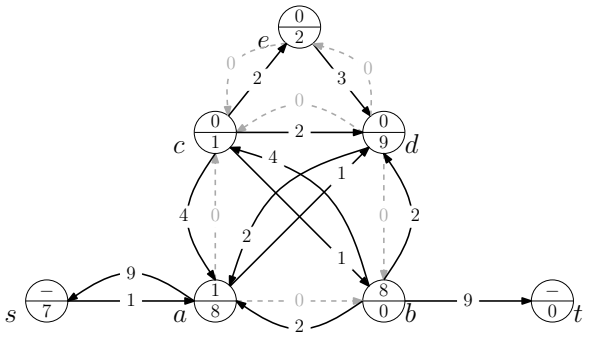
Fluss von e nach c geschoben:





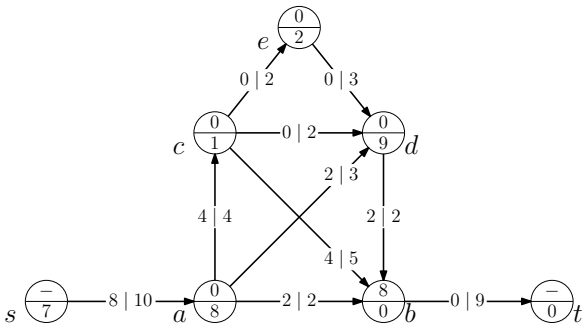
Fluss von c nach b geschoben:

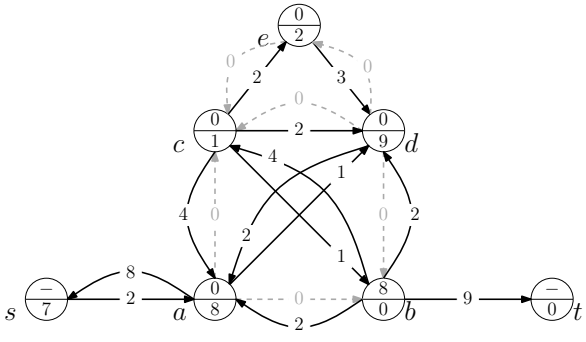




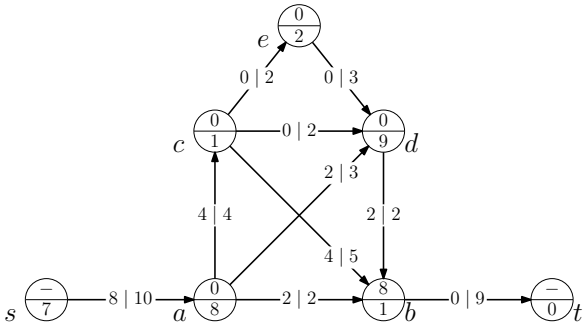
Musterlösung:

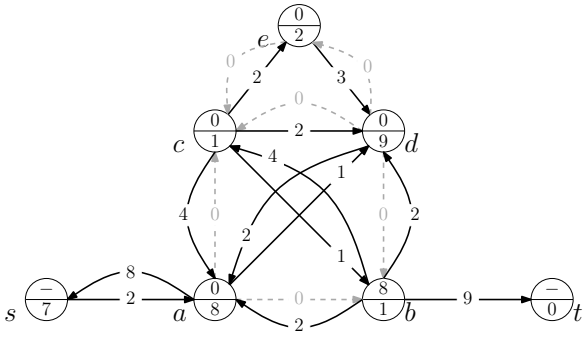
Fluss von a nach s geschoben:



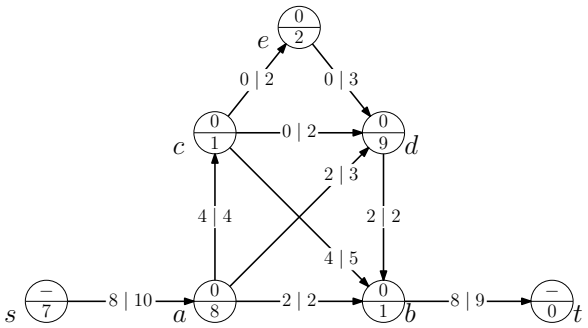


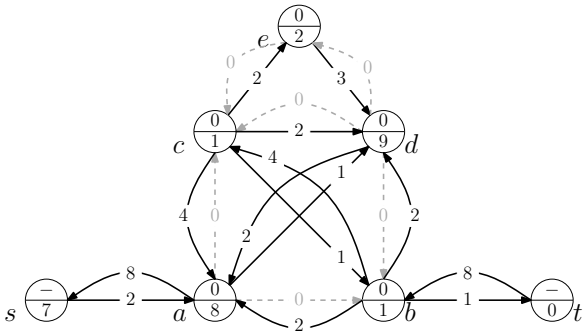
Level von Knoten b erhöht auf 1:





Fluss von b nach t geschoben:





Fertig!

Aufgabe 3 (*Analyse: Matchings*)

a) Zeigen oder widerlegen Sie. Existiert in einem bipartiten Matching ein maximaler alternierender Pfad, dessen erste und letzte Kante nicht Teil des Matchings sind, so hat das zugehörige Flussnetzwerk einen augmentierenden Pfad.

b) Gegeben ein bipartiter Graph $G = (L \cup R, E)$ mit $|L| = |R| = n$. Bezeichne $neigh(S) \subseteq R$ die Nachbarn der Knoten aus $S \subseteq L$. Zeigen Sie, gdw. ein perfektes Matching für G existiert, gilt für alle $S \subseteq L$: $|neigh(S)| \geq |S|$.

c) An einem Tanzkurs wollen n Männer und n Frauen teilnehmen. Jeder Teilnehmer kennt genau k Teilnehmer des anderen Geschlechts (alle Bekanntschaften sind beidseitig). Ist es möglich eine Paarung zu finden, in der jeder mit einem Bekannten tanzt?

Der Tanzkurs dauert genau k Wochen. Ist es möglich für jede Woche Paarungen zu bilden, so dass immer zwei Bekannte zusammen tanzen, sich aber keine Paarung wiederholt?

Musterlösung:

a) *(Wiederholung der Modellierung zur Referenz)*
Seien L, R die zu matchenden Mengen und E die möglichen Matchings.

Im zugehörigen Flussnetzwerk sei o.b.d.A. Quelle s mit jedem Knoten aus L und alle Knoten aus R mit Senke t verbunden. Die möglichen Matchings E seien als Kanten von L nach R gerichtet. Alle Kanten haben Gewicht 1. Eine Kante aus E ist gematcht, wenn ein Fluss über sie fließt.

Gegeben ein beliebiges Matching, so enthält der Residualgraph des Flussnetzwerks nur Kanten von L nach R die nicht Teil des Matchings sind, und nur Kanten von R nach L die Teil des Matchings sind.

(Beweis)
Existiere nach Aufgabenstellung ein Matching mit einem maximalen alternierenden Pfad mit nicht gematchten Kanten am Anfang und Ende. Dann startet dieser Pfad im Residualgraphen des Flussnetzwerks bei einem Knoten $l \in L$ und endet bei einem Knoten $r \in R$. Knoten l und r sind nicht benutzt (kein Fluss über Kante (s, l) und (r, t)), da sonst der alternierende Pfad nicht maximal wäre. Damit existiert ein augmentierender Pfad von s über den alternierenden Pfad nach t mit Fluss 1.

b) Die zu beweisende Aussage ist unter dem Namen *Halls Hochzeitstheorem* bekannt.

$\Rightarrow$: (G hat perfektes Matching $\rightarrow$ f.a. $S \subseteq L : |neigh(S)| \geq |S|$)

In einem perfekten Matching ist jeder Knoten aus L mit einem Knoten aus R gematcht. Betrachte beliebiges $S \subseteq L$. Würde gelten $|neigh(S)| < |S|$, könnte ein Knoten aus S nicht gematcht werden (*pigeon hole principle*). Widerspruch!

$\Leftarrow$: (f.a. $S \subseteq L : |neigh(S)| \geq |S| \rightarrow G$ hat perfektes Matching)

Bezeichne $neigh_{G'}(S)$ die Nachbarn der Knoten aus $S \subseteq L'$ in einem bel. bipartiten Graphen $G' = (L' \cup R', E')$. Beweis durch Induktion über n.

Sei $n = 1$. Es exisitiert trivialerweise ein perfektes Matching. Sei $n > 1$. Wähle $l \in L$ und $r \in neigh(\{l\})$ beliebig. Falls die Voraussetzung auch für $G' := (L' \cup R', E') := G \setminus \{l, r\}$ erfüllt ist, existiert ein perfektes Matching für G' (nach Induktion) und zusammen mit (l, r) ein perfektes Matching für G.

Ansonsten existiert $S \subseteq L \setminus \{l\}$ mit $|neigh_{G'}(S)| < |S|$ und $|neigh_G(S)| \geq |S|$.
Also $|S| \leq |neigh_G(S)| \leq |neigh_{G'}(S)| + 1 < |S| + 1$, und damit $|neigh(S)| = |S|$. Teile G in zwei disjunkte Teilgraphen G_1, bestehend aus S und $neigh(S)$, und G_2, bestehend aus $L \setminus S$ und $R \setminus neigh(S)$, auf. Finde für jeden ein perfektes Matching:

G1: Da für alle $S' \subseteq S \subseteq L$ gilt, $neigh(S') \subseteq neigh(S)$, folgt direkt $|neigh(S')| \geq |S'|$. Damit folgt, S und $neigh(S)$ besitzen ein perfektes Matching (nach Induktion).

G2: Für $L \setminus S$ und $R \setminus neigh(S)$ ist zunächst die Voraussetung zu prüfen: Wähle $T \subseteq (L \setminus S)$ beliebig. Dann gilt $|neigh_{G_2}(T)| = |neigh_{\text{in } G}(T \cup S)| - |neigh_G(S)| \geq |T \cup S| - |S| = |T|$. Die Voraussetzung ist erfüllt und damit existiert auch in diesem Teilgraphen ein perfektes Matching (nach Induktion) und damit für G.

c) Angenommen es gäbe kein perfektes Matching. Dann existiert nach Teilaufgabe b) eine Teilmenge M an Männern, die weniger als $|M|$ Frauen kennt. Dies ist nicht möglich, da jeder Mann genau k Bekanntschaften hat und jede der $|M|$ Frauen maximal k Bekanntschaften in M hat: Man kann $k \cdot |M|$ 'ausgehende' Bekanntschaften der Männer nicht auf weniger als $k \cdot |M|$ 'eingehende' Bekanntschaften der Frauen aufteilen.

Per Konstruktion sind k verschiedene Paarungen mit jeweils anderen Bekannten möglich: Nach der ersten Woche streiche die Bekanntschaften, die schon miteinander getanzt haben. Danach hat jeder nur noch $k - 1$ Bekanntschaften. Für diese existiert ein perfektes Matching. Fahre in den folgenden Wochen entsprechend fort.

Aufgabe 4 (*Schlechter Zufall*)

Gegeben sei ein randomisierter Algorithmus badBit, der keine Eingabe liest und als Ausgabe zufällig mit Wahrscheinlichkeit p die Zahl 0 und mit Wahrscheinlichkeit $q = 1 - p$ die Zahl 1 liefert. Es sei $0 < p < 1$; der konkrete Wert von p sei aber unbekannt. Der Algorithmus gebe bei mehrfachem Aufruf unabhängige Ergebnisse aus.
Entwerfen Sie einen randomisierten Algorithmus fairBit, der keine Eingabe liest und als Ausgabe immer zufällig eine der Zahlen 0 und 1 mit Wahrscheinlichkeit 1/2 liefert.
Was können Sie über die Laufzeit Ihres Algorithmus sagen?

Musterlösung:
Algorithmusidee:

```
1: function FAIRBIT
2:     repeat
3:         x ← badBit()
4:         y ← badBit()
5:     until x ≠ y
6:     return x
7: end function
```

Korrektheit:
$\mathbb{P}(x = 1 | x \neq y) = \frac{\mathbb{P}(x=1 \ \wedge \ y=0)}{\mathbb{P}(x \neq y)} = \frac{pq}{2pq} = \frac{1}{2}$

Laufzeit:
Im Folgenden benutzen wir folgende Überlegung (für $z \in \mathbb{R}$ mit $0 < |z| < 1$):

$$\text{sei} \qquad R = \sum_{i=1}^{\infty} i \cdot z^{i-1} = \sum_{i=0}^{\infty} (i+1) \cdot z^i$$

$$\text{dann} \qquad Rz = \sum_{i=1}^{\infty} i \cdot z^i$$

$$\text{also} \qquad R(1-z) = R - Rz = \sum_{i=0}^{\infty} z^i = \frac{1}{1-z}$$

$$\text{also} \qquad R = \frac{1}{(1-z)^2}$$

Nun ist im Algorithmus die Wahrscheinlichkeit für $x = y$ gleich $z = p^2 + q^2$, also $1 - z = 2pq$. Wegen $0 < p < 1$ ist $0 < |z| < 1$. Daher ergibt sich für den Erwartungswert der Laufzeit:

$$\sum_{i=1}^{\infty} i \cdot 2 \cdot (p^2 + q^2)^{i-1} 2pq = 4pq \sum_{i=1}^{\infty} i \cdot z^{i-1} = 4pq \frac{1}{4p^2q^2} = \frac{1}{pq}$$

Aufgabe 5 (*Kleinaufgaben: Laufzeiten*)

Sei $T(n, \varepsilon)$ die Laufzeit eines Approximationsalgorithmus und $g(n, \varepsilon)$ seine Approximationsgarantie. Geben Sie für die folgenden Fälle an, ob der Algorithmus ein PTAS, FPTAS oder keines von beiden ist. Begründen Sie Ihre Antwort jeweils kurz.

- $T_1(n, \varepsilon) = \frac{1}{\varepsilon} \cdot (4n^3 + n^2)$, $g_1(n, \varepsilon) = (1 - \varepsilon)$
- $T_2(n, \varepsilon) = \frac{1}{\varepsilon} \cdot n^2$, $g_2(n, \varepsilon) = (1 + 2\varepsilon)$
- $T_3(n, \varepsilon) = \sqrt{n} + n^{\frac{3}{2}}$, $g_3(n, \varepsilon) = 2 + \frac{1}{n}$
- $T_4(n, \varepsilon) = n \cdot \log \frac{1}{\varepsilon}$, $g_4(n, \varepsilon) = (1 - \varepsilon)$
- $T_5(n, \varepsilon) = \varepsilon + e^{\log n} + n^5$, $g_5(n, \varepsilon) = (1 + \varepsilon)$
- $T_6(n, \varepsilon) = n^{\frac{1}{\varepsilon}} + n^5$, $g_6(n, \varepsilon) = (2 + \varepsilon)$

Anmerkung: Für $g_6(n, \epsilon)$ können Sie annehmen, dass der Approximationsalgorithmus mit beliebigem $\epsilon \in (-1, 0)$ spezifiziert werden kann.

Musterlösung:

a) Ein Approximationsalgorithmus wird als PTAS bezeichnet, wenn seine Laufzeit $T(n, \varepsilon)$ polynomiell in n ist und sich seine Approximationsgarantie beliebig nahe der 1 nähern kann. Für ein FPTAS muss zusätzlich $T(n, \varepsilon)$ polynomiell in $\frac{1}{\varepsilon}$ sein. Mit dieser Definition ergibt sich für die angegebenen Algorithmen:

- $T_1(n, \varepsilon) = \frac{1}{\varepsilon} \cdot (4n^3 + n^2)$, $\quad g_1(n, \varepsilon) = (1 - \varepsilon)$
  Bei dem angegebenen Algorithmus handelt es sich um ein FPTAS. $T_1(n, \varepsilon)$ hängt polynomiell von n und $\frac{1}{\varepsilon}$ ab und die Approximationsgarantie kann beliebig nahe an die 1 herankommen.
- $T_2(n, \varepsilon) = \frac{1}{\varepsilon} \cdot n^2$, $\quad g_2(n, \varepsilon) = (1 + 2\varepsilon)$
  Bei dem angegebenen Algorithmus handelt es sich um ein FPTAS. Es gilt die gleiche Begründung wie bei $T_1(n, \varepsilon)$ und $g_1(n, \varepsilon)$. Will man die klassische Approximationsgarantie ohne den Faktor 2 sehen, substituiert man einfach $\delta = 2\varepsilon$.
- $T_3(n, \varepsilon) = \sqrt{n} + n^{\frac{3}{2}}$, $\quad g_3(n, \varepsilon) = 2 + \frac{1}{n}$
  Bei dem angegebenen Algorithmus handelt es sich weder um ein FPTAS noch um ein PTAS. Die Approximationsgarantie hängt von n ab und kann nicht beliebig nahe an 1 herankommen.
- $T_4(n, \varepsilon) = n \cdot \log \frac{1}{\varepsilon}$, $\quad g_4(n, \varepsilon) = (1 - \varepsilon)$
  Bei dem angegebenen Algorithmus handelt es sich um ein FPTAS. Die Approximationsgarantie kann sich beliebig der 1 nähern. $T_4(n, \varepsilon)$ hängt polynomiell von n und auch von $\frac{1}{\varepsilon}$ (da $\log x = O(n) = O(poly(n))$).
- $T_5(n, \varepsilon) = \varepsilon + e^{\log n} + n^5$, $\quad g_5(n, \varepsilon) = (1 + \varepsilon)$
  Bei dem angegebenen Algorithmus handelt es sich um ein FPTAS. $T_5(n, \varepsilon)$ ist polynomiell in n ($e^{\log n} = n$) und $\frac{1}{\varepsilon}$ ($\varepsilon = \frac{1}{\varepsilon}^{-1}$). Außerdem kann sich die Approximationsgarantie beliebig der 1 nähern.
- $T_6(n, \varepsilon) = n^{\frac{1}{\varepsilon}} + n^5$, $\quad g_6(n, \varepsilon) = (2 + \varepsilon)$
  Bei dem angegebenen Algorithmus handelt es sich um ein FPTAS. Die Approximationsgarantie kann sich beliebig der 1 nähern. $T_6(n, \varepsilon)$ lässt sich für $\varepsilon < 0$ abschätzen durch $\leq n^{-1} + n^5$ und ist damit sogar unabhängig von $\frac{1}{\varepsilon}$.

Aufgabe 6 (*Analyse+Rechnen: Vertex-Cover*)

Gegeben sei folgender Algorithmus zur Berechnung eines *vertex cover* C für einen Graph $G = (V, E)$:

1. Initialisiere die Ergebnismenge $C = \varnothing$ als leere Menge.
2. Wähle Kante $(u, v) \in E$ beliebig.
3. Füge u, v zu C hinzu.
4. Entferne u, v und alle inzidenten Kanten aus G.
5. Wiederhole solange G noch Kanten hat

Nach Abschluss des Algorithmus ist $C \subseteq V$ ein *vertex cover*, d.h. für jede Kante $(u, v) \in E$ ist einer ihrer beiden Knoten in C. Falls o.b.d.A. $u \in C$ sagt man auch Knoten u *überdeckt* Kante (u, v).

a) Zeigen Sie, dass der angegebene Algorithmus ein korrektes *vertex cover* berechnet.

b) Geben Sie ein Beispiel an, in dem der Algorithmus ein minimales *vertex cover* liefert.

c) Geben Sie ein Beispiel an, in dem der Algorithmus kein minimales *vertex cover* liefert.

d) Zeigen oder widerlegen Sie, dass der Algorithmus eine 2-Approximation für *vertex cover* berechnet, d.h. dass er höchstens doppelt soviele Knoten auswählt als minimal nötig.

Betrachten Sie abschließend diesen alternativen Algorithmus zur Bestimmung einer 2-Approximation von *vertex cover*:

1. Initialisiere die Ergebnismenge $C = \varnothing$ als leere Menge.
2. Wähle Knoten $u \in V$ mit minimalem Grad.
3. Füge u zu C hinzu.
4. Entferne u und alle inzidenten Kanten aus G
5. Wiederhole solange G noch Kanten hat

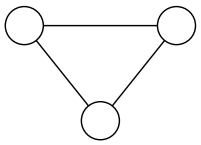
Der Algorithmus berechnet offenbar –mit ähnlichen Argumenten wie in Teilaufgabe (a)– ein *vertex cover*. Es bleibt folgende Frage zu beantworten:

e) Zeigen oder widerlegen Sie, dass der Algorithmus eine 2-Approximation für *vertex cover* berechnet, d.h. dass er höchstens doppelt soviele Knoten auswählt als minimal nötig.

Musterlösung:

a) Nach Abschluss enthält der Graph keine Kanten mehr. Da eine Kante nur entfernt wird, wenn einer ihrer Knoten in C aufgenommen wurde, ist folglich jede Kante $e \in E$ von einem Knoten überdeckt. Damit ist C ein *vertex cover*.

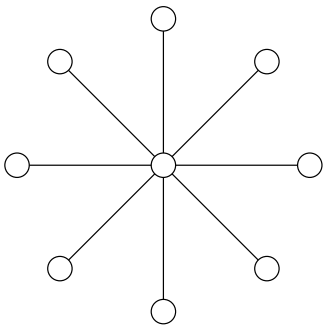
b) In folgendem Graphen werden immer genau zwei Knoten ausgewählt. Dies ist optimal, da einzelner Knoten nur zwei der drei Kanten überdecken.



c) In folgendem Graphen werden immer zwei Knoten ausgewählt. Das ist nicht optimal, da ein einzelner Knoten genügen würde.

d) Sei A die Menge der in Schritt 2 ausgewählten Kanten. Es gilt $|C| = 2|A|$, da beide Knoten jeder ausgewählten Kante zu C hinzugefügt und anschließend zusammen mit allen inzidenten Kanten aus G entfernt werden, so dass sie nicht noch einmal ausgewählt werden können. Damit folgt auch, dass keine zwei Kanten aus A einen Knoten gemeinsam haben können. Sei nun C^* ein minimales *vertex cover*. C^* enthält nach Definition mindestes einen Knoten jeder Kante, also insbesondere einen Knoten jeder Kante aus A. Da keine zwei Kanten aus A vom gleichen Knoten aus C^* überdeckt werden können, gilt $|C^*| \geq |A|$. Es folgt $|C| = 2|A| \leq 2|C^*|$. Die Lösung C des angegebenen Algorithmus ist also maximal doppelt so groß wie die optimale Lösung C^*. Damit berechnet der Algorithmus eine 2-Approximation.

e) Der Algorithmus berechnet keine 2-Approximation. Folgender Graph ist ein Gegenbeispiel:



Ein optimales *vertex cover* benötigt nur den Knoten in der Mitte. Der angegebene Algorithmus markiert allerdings $n - 1$ Knoten (entweder alle Randknoten, oder den mittleren Knoten und alle Randknoten bis auf einen).

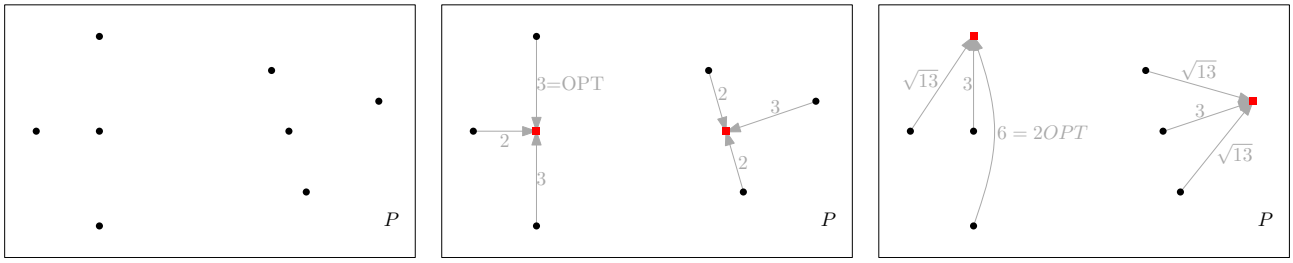
Aufgabe 7 (*Analyse: Metrisches k-Zentren Problem (*)*)

Gegeben sei eine Menge an Punkten $P \subset \mathbb{R}^2$ in der Ebene sowie eine Zahl $k > 0$. Gesucht ist eine k-elementige Teilmenge $K \subset P$ dieser Punkte, genannt Zentren, so dass für jeden Punkt $p \in P$ der maximale Abstand zu seinem nächstgelegenen Zentrum minimal ist.

Es existiert folgender *greedy* Algorithmus, der eine 2-Approximation des Problems berechnet:

1. Wähle beliebigen Punkt aus P als erstes Zentrum
2. Wähle Punkt aus P als nächstes Zentrum mit größter Entfernung zu allen bisherigen Zentren (d.h. der den maximalen kürzesten Abstand zu einem Zentrum besitzt)
3. Wiederhole bis k Zentren gewählt worden sind

Das folgende Beispiel veranschaulicht die Problemstellung für $k = 2$:



Links ist eine Punktmenge P abgebildet. In der Mitte ist eine optimale Lösung zu sehen. Die roten Quadrate sind die ausgewählten Zentren. Die Kanten geben das nächstgelegene Zentrum für jeden Knoten sowie den Abstand an. Rechts ist eine weitere aber suboptimale Lösung aufgezeigt.

Zunächst einige allgemeine Fragen zu diesem Algorithmus:

a) Beschreiben Sie in Worten, welche Bedeutung OPT sowie die Aussage eine Lösung sei eine 2-Approximation des metrischen k-Zentren Problems, haben.

b) Handelt es sich bei dem angegebenen Algorithmus um ein PTAS, ein FPTAS oder um keines von beiden? Begründen Sie kurz.

Im Folgenden soll gezeigt werdem, dass der Algorithmus tatsächlich eine 2-Approximation berechnet. Dafür sind zunächst einige Vorüberlegungen nötig.

c) Zeigen Sie, bei einer Auswahl von $k + 1$ Punkten aus P existieren immer mindestens 2 Punkte, die das gleiche nächstgelegene Zentrum haben.

d) Gegeben eine optimale Lösung, wie groß kann der Abstand zwischen zwei Punkten maximal sein, wenn diese das gleiche nächstgelegene Zentrum besitzen?

e) In einer Lösung des *greedy* Algorithmus sei der maximale Abstand eines Punktes $p \notin K$ zu seinem nächstgelegenen Zentrum $> l$. Zeigen Sie, dass l eine untere Schranke für den Abstand zwischen je zwei der Zentren $k_i,\ k_j \in K,\ i \neq j$ der Lösung darstellt.

f) Zeigen Sie mit obigen Aussagen, dass der angegebene *greedy* Algorithmus eine 2-Approximation für das Problem berechnet. Nehmen Sie dazu an, in der Lösung des Algorithmus sei der maximale

Abstand eines Punktes $p \notin K$ zu seinem nächstgelegenen Zentrum $> 2 \cdot OPT$, und führen Sie diese Aussage zum Widerspruch.

Hinweis: Machen Sie zunächst eine Aussage über die paarweisen Abstände von $k+1$ speziell gewählten Punkten. Verwenden Sie anschließend einen Vergleich zu Abständen in der optimalen Lösung, um zum Widerspruch zu gelangen.

Musterlösung:

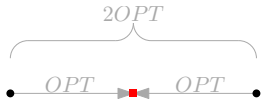
a) Im metrischen k-Zentren Problem charakterisiert OPT den maximalen Abstand eines Punktes zu seinem nächstgelegenen Zentrum. Eine 2-Approximation bedeutet, dass der Abstand eines Punktes zu seinem nächstgelegenen Zentrum höchstens doppelt so groß ist wie OPT (das rechte Bild in der Aufgabenstellung beschreibt eine 2-Approximation).

b) Der beschriebene *greedy* Algorithmus ist weder ein PTAS noch ein FPTAS, da sich seine Approximationsgüte nicht beliebig der 1 annähern lässt (bei polynomieller Laufzeit ist der Algorithmus immerhin in APX).

c) Angenommen k Punkte haben paarweise verschiedene nächstgelegene Zentren. Damit ist jeder Punkt einem anderen Zentrum zugeordnet und alle Zentren sind verwendet. Ein weiterer Punkt hätte auf alle Fälle ein schon verwendetes Zentrum als nächstgelegenes Zentrum. Wäre dies nicht der Fall, so gäbe es mehr als k Zentren oder die bisherigen Punkte hätten nicht alle paarweise verschiedene nächstgelegene Zentren.

Dieses Prinzip wird auch *pigeon hole principle* genannt.

d) Wie in der Abbildung zu sehen, können sich beide Punkte auf entgegengesetzten Seiten des Zentrums befinden mit maximalem Abstand OPT. Damit ist ihr Abstand zueinander $2 \cdot OPT$.



(Hinweis: Für diese Aussage wurde der metrische Raum benötigt!)

e) Nach Voraussetzung ist Abstand $d(p,k) > l$ f.a. $k \in K$ und damit auch f.a. $k \in K/\{k_j\}$. Angenommen es gelte für einen Abstand $d(k_i, k_j) \leq l$. O.b.d.A. werde k_i vor k_j als Zentrum ausgewählt. Dann würde im weitern Verlauf des Algorithmus p anstatt k_j als Zentrum gewählt werden, da p den größeren minimalen Abstand zu den bisherigen Zentren hat. Da aber k_j ein Zentrum ist, muss $d(k_i, k_j) > l$ gelten.

f) Angenommen, in der Lösung des Algorithmus sei der maximale Abstand eines Punktes $p \notin K$ zu seinem nächstgelegenen Zentrum $> 2 \cdot OPT$. Nach Teilaufgabe (e) wäre der Abstand zwischen allen Zentren $> 2 \cdot OPT$. Das würde bedeuten, es gäbe $k+1$ Punkte mit einem paarweisen Abstand $> 2 \cdot OPT$ (Punkt p sowie die k Zentren). Wähle zwei dieser Punkte, die in einer optimalen Lösung das gleiche nächste Zentrum haben. Teilaufgabe (c) belegt die Existenz dieser Punkte. Nach Teilaufgabe (d) hätten sie allerdings einen Abstand $\leq 2 \cdot OPT$. Widerspruch zu der Aussage, dass alle diese Punkte einen paarweisen Abstand $> 2 \cdot OPT$ besitzen.