/ 15 P

Theorieaufgabe T1.

Hinweise:

1) In dieser Aufgabe sollen Sie **nur die Ergebnisse** angeben. Diese können Sie direkt bei den Aufgaben notieren.
2) Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

/ 1 P

a) Gegeben sei eine Menge von Frauen $\mathcal{F} = \{f_1, f_2, f_3, f_4\}$ sowie eine Menge von Männern $\mathcal{M} = \{m_1, m_2, m_3, m_4\}$. Die folgenden beiden Tabellen zeigen jeweils ihre Präferenzen, die von links nach rechts absteigend sortiert sind. Führen Sie den Algorithmus von Gale und Shapley aus, und nehmen Sie an, dass die Frauen die Heiratsanträge machen.

f_1	m_2	m_1	m_3	m_4
f_2	m_1	m_4	m_2	m_3
f_3	m_4	m_2	m_1	m_3
f_4	m_2	m_3	m_4	m_1

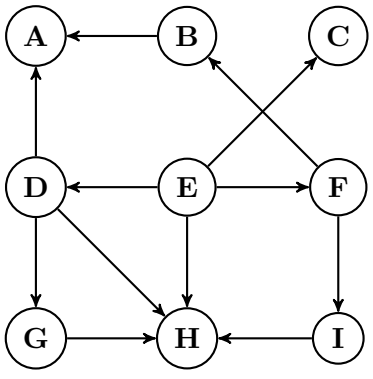
m_1	f_3	f_4	f_2	f_1
m_2	f_3	f_2	f_1	f_4
m_3	f_1	f_2	f_3	f_4
m_4	f_2	f_3	f_1	f_4

Zuordnung:

f_1	m_2	f_2	m_1	f_3	m_4	f_4	m_3

/ 1 P

b) Führen Sie im folgenden Graph eine Breitensuche und eine Tiefensuche aus. Starten Sie dafür in Knoten E und geben sie die Reihenfolgen an, in denen die Knoten besucht werden. Nehmen Sie an, dass die beiden Traversierungen die jeweiligen Nachbarknoten in alphabetischer Reihenfolge abarbeiten.



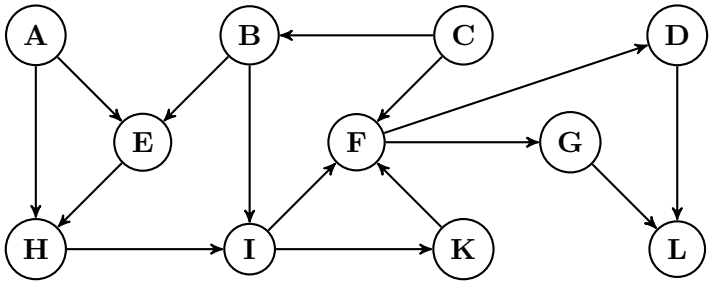
Breitensuche:

E, C, D, F, H, A, G, B, I.

Tiefensuche:

E, C, D, A, G, H, F, B, I.

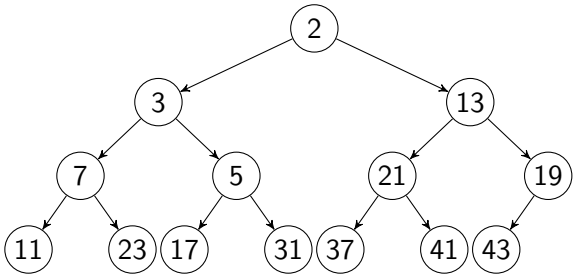
/ 1 P c) Wie viele verschiedene topologische Sortierungen hat der untenstehende Graph?



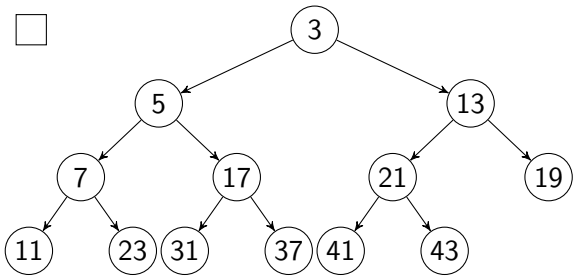
Anzahl topologische Sortierungen:

6

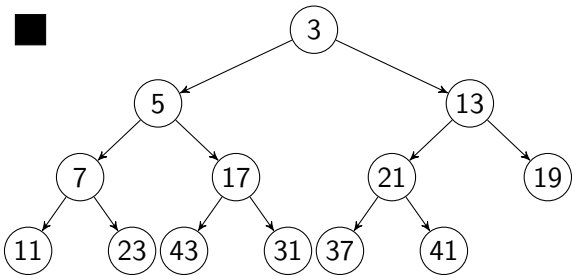
/ 1 P d) Führen Sie eine Extract-Min Operation einschliesslich Wiederherstellung der Heap-Bedingungen auf folgendem Min-Heap aus. Wie sieht der Heap nach der Operation aus? Kreuzen Sie die richtige Antwort an. Sie müssen Ihre Antwort nicht begründen.



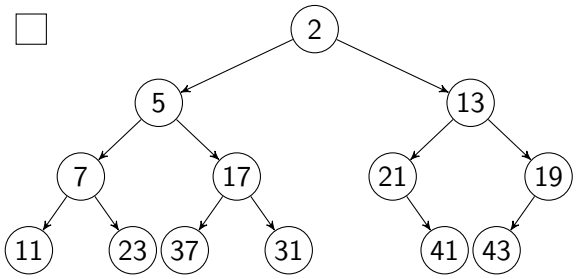
☐



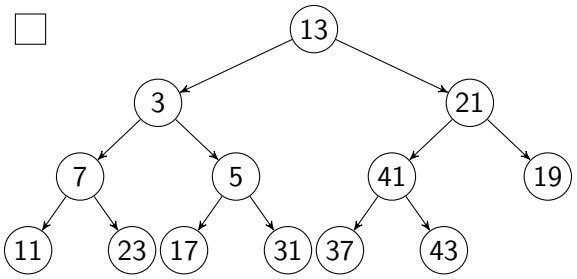
■



☐



☐



/ 1 P

e) Führen Sie auf dem folgenden Array zwei Iterationen des Sortieralgorithmus *Natürliches 2-Wege-Mergesort* aus.

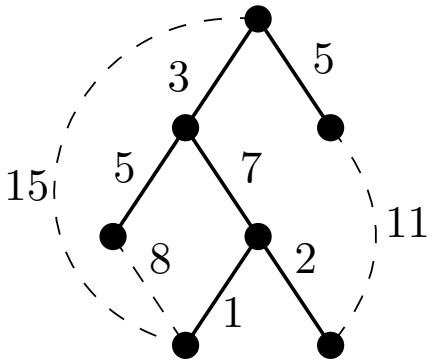
9	32	19	23	72	51	47	64	14	2	67	41	11	31
1	2	3	4	5	6	7	8	9	10	11	12	13	14

9	19	23	32	72	47	51	64	2	14	67	11	31	41
1	2	3	4	5	6	7	8	9	10	11	12	13	14

9	19	23	32	47	51	64	72	2	11	14	31	41	67
1	2	3	4	5	6	7	8	9	10	11	12	13	14

/ 2 P

f) Betrachten Sie jeweils den untenstehenden Graph, in dem ein Minimaler Spannbaum bereits eingezeichnet ist.



i) Was ist der tiefste Wert auf den die Kante mit Gewicht 3 erhöht werden kann, so dass sie nicht mehr in jedem Minimalen Spannbaum dieses Graphen vorkommt?

11

ii) Was ist der höchste Wert auf den die Kante mit Gewicht 11 geändert werden kann, so dass sie in mindestens einem Minimalen Spannbaum dieses Graphen vorkommt?

7

/ 1 P

g) Finden Sie heraus, ob folgende Postorder und Preorder zum gleichen natürlichen Suchbaum gehören.

Postorder: 4, 3, 8, 17, 12, 10, 21, 27, 23, 20, 7

Preorder: 7, 3, 4, 20, 8, 10, 12, 17, 23, 21, 27

Kreuzen Sie die richtige Antwort an. Sie müssen Ihre Antwort nicht begründen.

☐ Ja, gleicher Baum

☒ Nein, verschiedene Bäume

/ 2 P

h) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. Jede korrekte Antwort gibt 0,5 Punkte, für jede falsche Antwort werden 0,5 Punkte abgezogen. Eine fehlende Antwort gibt 0 Punkte. Insgesamt gibt die Aufgabe mindestens 0 Punkte. Sie müssen Ihre Antworten nicht begründen.

Für jeden AVL-Baum gibt es eine Einfügereihenfolge seiner Schlüssel, die zu diesem Baum führt, *ohne* dass eine einzige Rotation stattfindet.	☒ Wahr	☐ Falsch
Es gibt einen AVL-Baum, bei dem *alle* seine inneren Knoten nicht perfekt balanciert sind (d.h. einen Balancierungsfaktor ungleich Null haben).	☒ Wahr	☐ Falsch
Ein natürlicher Suchbaum der Höhe h hat *nie* mehr als 2^{h-1} Knoten.	☐ Wahr	☒ Falsch
Sei $T(n)$ die Anzahl verschiedener natürlicher Suchbäume, die genau die Schlüssel $\{1, 2, 3, \ldots, n\}$ verwalten. Dann gilt $T(0) = 1$, $T(1) = 1$ und $T(n) = \sum_{k=1}^{n}(T(k-1) \cdot T(n-k))$ für $n \geq 2$.	☒ Wahr	☐ Falsch

/ 1 P

i) Geben Sie für das folgende Codefragment die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ in Θ-Notation an. Halten Sie sich so knapp wie möglich. Die Funktion f läuft in $\Theta(1)$ Zeit. Sie müssen Ihre Antwort nicht begründen.

```
1 for(int i = n; i ≥ 1; i = i/2 ) {
2         for(int j = 1; j ≤ i; j = 2*j ) {
3                 f();
4         }
5 }
```

Anzahl der Funktionsaufrufe in möglichst knapper Θ-Notation:

$\Theta(\log^2 n)$

/ 1 P

j) Geben Sie für das folgende Codefragment die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ in Θ-Notation an. Halten Sie sich so knapp wie möglich. Die Funktion f läuft in $\Theta(1)$ Zeit. Sie müssen Ihre Antwort nicht begründen.

```
1 for(int i = 1; i*i ≤ n; i = i+1 ) {
2         for(int j = n; j ≥ 1; j = j/4 ) {
3                 for(int k = 1; k ≤ j; k = k+1 ) {
4                         f();
5                 }
6         }
7 }
```

Anzahl der Funktionsaufrufe in möglichst knapper Θ-Notation:

$\Theta(n^{3/2})$

/ 1 P

k) Geben Sie für die untenstehenden Funktionen eine **Reihenfolge** an, so dass folgendes gilt: Wenn eine Funktion f links von einer Funktion g steht, dann gilt $f \leq \mathcal{O}(g)$.

Beispiel: Die drei Funktionen n^3, n^7, n^9 sind bereits in der entsprechenden Reihenfolge, da $n^3 \leq \mathcal{O}(n^7)$ und $n^7 \leq \mathcal{O}(n^9)$ gilt.

$$n^{\frac{8}{7}},\ 24^n,\ \frac{\sqrt{n}}{\log^2 n},\ \sqrt{n!},\ \log^{100} n,\ n^{\frac{7}{8}},\ \log n!,\ 5^{2n}$$

Lösung: $\log^{100} n$, $\frac{\sqrt{n}}{\log^2 n}$, $n^{\frac{7}{8}}$, $\log n!$, $n^{\frac{8}{7}}$, 24^n, 5^{2n}, $\sqrt{n!}$.

/ 2 P

l) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} n + 2 \cdot T(n/5) & n > 1 \\ 1 & n = 1 \end{cases}$$

Sie können annehmen, dass n eine Potenz von 5 ist. Benutzen Sie also $n = 5^k$ oder $k = \log_5(n)$. Zeigen Sie mit vollständiger Induktion, dass

$$T(n) = T(5^k) = \frac{5^{k+1} - 2^{k+1}}{3}$$

die dazugehörige geschlossene Form ist.

Induktionsbeweis:

Induktionsanfang ($k = 0$)*:* Es gilt $T(5^0) = 1 = \frac{3}{3} = \frac{5-2}{3} = \frac{5^{0+1}-2^{0+1}}{3}$.

Induktionshypothese: Für ein $k \in \mathbb{N}_0$ sei $T(5^k) = \frac{5^{k+1}-2^{k+1}}{3}$.

Induktions-Schritt ($k \to k+1$)*:*

$$\begin{aligned} T(5^{k+1}) &= 5^{k+1} + 2 \cdot T(5^k) \\ &\overset{I.H.}{=} 5^{k+1} + 2 \cdot \frac{5^{k+1} - 2^{k+1}}{3} \\ &= \frac{3 \cdot 5^{k+1}}{3} + \frac{2 \cdot 5^{k+1}}{3} - \frac{2^{k+2}}{3} \\ &= \frac{5^{k+2} - 2^{k+2}}{3} \end{aligned}$$

/ 15 P

Theorieaufgabe T2.

Zürich führt ab Sommer 2047 ein Road Pricing ein. Dabei wird jeder Strasse eine Gebühr zugeordnet. Es ist bekannt, dass jeweils am Morgen alle Autos von s nach t fahren möchten. Mit dem neuen Pricing werden selbstverständlich alle Autos entlang eines günstigsten Wegs fahren. Um sich für das neue Verkehrsaufkommen zu wappnen, möchte die Stadt Zürich herausfinden, welche Kreuzungen überhaupt befahren werden für ein gegebenes Pricing.

Das Strassennetz von Zürich ist gegeben durch n Kreuzungen und m Strassen. Jede Strasse e ist gegeben als ein Tupel (u, v, g_{uv}), definiert durch die beiden Kreuzungen u und v welche e verbindet, und durch die Gebühr von g_{uv} Franken welche *bei jeder Benützung* erhoben wird ($g_{uv} \in \mathbb{N}$).

Modellieren Sie die nachfolgenden drei Fragestellungen jeweils als graphentheoretisches Problem. Geben Sie dazu an, welche Knoten und Kanten definiert werden und welche Gewichte den Kanten zugeordnet werden. Nennen Sie einen möglichst effizienten Algorithmus zur Lösung des jeweiligen graphentheoretischen Problems und bestimmen Sie seine Laufzeit in Abhängigkeit von n und m.

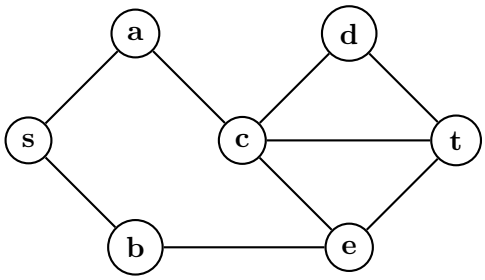
Sie dürfen jeweils davon ausgehen, dass mindestens ein Weg *von s nach t* existiert (für alle anderen Paare von Kreuzungen u, v gibt es hingegen nicht notwendigerweise einen Weg von u nach v).

/ 5 P

a) Nehmen Sie in dieser Teilaufgabe an, dass jede Strasse, gegeben durch $e = (u, v, g_{uv})$, in *beide Richtungen* befahren werden darf. Nehmen Sie ausserdem an, dass die Gebühr für jede Strasse gleich 1 ist.

Für *jede* Kreuzung soll bestimmt werden, ob sie auf einem *günstigsten* Weg von s nach t liegt, oder nicht.

Beispiel. JA: s, a, b, c, e, t NEIN: d

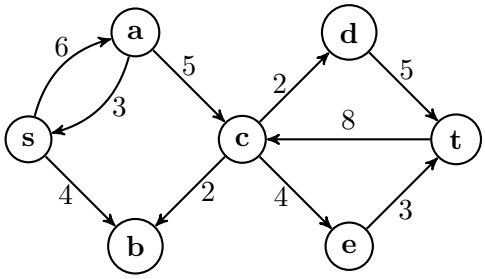


/ 5 P

b) Nehmen Sie in dieser Teilaufgabe an, dass jede Strasse, gegeben durch $e = (u, v, g_{uv})$, nur *von u nach v* befahren werden darf.

Für *jede* Kreuzung soll bestimmt werden, ob sie auf einem *günstigsten* Weg von s nach t liegt, oder nicht.

Beispiel. JA: s, a, c, d, e, t NEIN: b

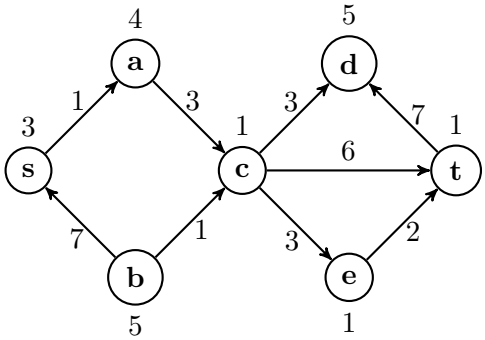


/ 5 P

c) Zürich beschliesst, zusätzlich zu den Strassen auch das Befahren einer Kreuzung gebührenpflichtig zu machen. Für jede Benützung einer Kreuzung v werden b_v Franken verrechnet. Nehmen Sie an, dass jede Strasse, gegeben durch $e = (u, v, g_{uv})$, nur *von u nach v* befahren werden darf.

Für *jede* Kreuzung soll bestimmt werden, ob sie auf einem *günstigsten* Weg von s nach t liegt, oder nicht.

Beispiel. JA: s, a, c, e, t NEIN: b, d



Teilaufgabe a)

- Graphentheoretische Modellierung inklusive Kantengewichte (in Worten):

  Ungewichteter, ungerichteter Graph $G = (V, E)$ mit:
  - je einem Knoten u für jede Kreuzung u.
  - je einer Kante $\{u, v\}$ für jede Strasse (u, v, g_{uv}).

- Anzahl der Knoten und Kanten in Abhängigkeit von n, m (in möglichst knapper Θ-Notation):

  $|V| \in \Theta(n), \quad |E| \in \Theta(m)$

- Möglichst effizienter Algorithmus zur Klassifikation der n Kreuzungen:

  Mit einer BFS von s aus speichern wir für jeden Knoten u die Distanz $d(s, u)$.
  Mit einer BFS von t aus speichern wir für jeden Knoten u die Distanz $d(u, t)$.

  Nun testen wir für jeden Knoten u, ob $d(s, u) + d(u, t) == d(s, t)$ ist.
  Falls ja, wird die Kreuzung u befahren, ansonsten nicht.

- Laufzeit in Abhängigkeit von n, m (in möglichst knapper Θ-Notation) inklusive Begründung:

  Zweimal BFS: $\Theta(|V| + |E|) = \Theta(n + m)$.
  Kreuzungen klassifizieren: $\Theta(|V|) = \Theta(n)$.

  Insgesamt: $\Theta(n + m)$.

Teilaufgabe b)

- Graphentheoretische Modellierung inklusive Kantengewichte (in Worten):

  Gewichteter, gerichteter Graph $G = (V, E, w)$ mit:
  - je einem Knoten u für jede Kreuzung u.
  - je einer Kante (u, v) mit Gewicht $w(u, v) = g_{uv}$ für jede Strasse (u, v, g_{uv}).

  Denselben Graph mit umgekehrten Kanten, das heisst $G^T = (V^T, E^T, w^T)$ mit:
  - je einem Knoten u für jede Kreuzung u.
  - je einer Kante (v, u) mit Gewicht $w^T(v, u) = g_{uv}$ für jede Strasse (u, v, g_{uv}).

- Anzahl der Knoten und Kanten in Abhängigkeit von n, m (in möglichst knapper Θ-Notation):

  $|V| \in \Theta(n), \quad |E| \in \Theta(m)$.
  $|V^T| \in \Theta(n), \ |E^T| \in \Theta(m)$.

- Möglichst effizienter Algorithmus zur Klassifikation der n Kreuzungen:

  Mit Dijkstra in G von s aus speichern wir für jedes $u \in V$ die Distanz $d(s, u)$.
  Mit Dijkstra in G^T von t aus speichern wir für jedes $u \in V^T$ die Distanz $d^T(t, u)$.

  Nun testen wir für jede Kreuzung u, ob für die entsprechenden Knoten u in V, V^T gilt, dass $d(s, u) + d^T(t, u) == d(s, t)$ ist.
  Falls ja, wird die Kreuzung u befahren, ansonsten nicht.

- Laufzeit in Abhängigkeit von n, m (in möglichst knapper Θ-Notation) inklusive Begründung:

  Zweimal Dijkstra (mit Fibonacci-Heaps):
  $\Theta(|V| \log |V| + |E| + |V^T| \log |V^T| + |E^T|)) = \Theta(n \log n + m)$.
  Kreuzungen klassifizieren: $\Theta(|V|) = \Theta(n)$.

  Insgesamt: $\Theta(n \log n + m)$.

  Ohne Fibonacci-Heaps: $\Theta(n + m \log m)$.

Teilaufgabe c)

- Graphentheoretische Modellierung inklusive Kantengewichte (in Worten):

  Gewichteter, gerichteter Graph $G = (V, E, w)$ mit:
  - je zwei Knoten u_{in}, u_{out} für jede Kreuzung u.
  - je einer Kante (u_{in}, u_{out}) mit Gewicht $w(u_{in}, u_{out}) = b_u$ für jede Kreuzung u.
  - je einer Kante (u_{out}, v_{in}) mit Gewicht $w(u_{out}, v_{in}) = g_{uv}$ für jede Strasse (u, v, g_{uv}).

  Denselben Graph mit umgekehrten Kanten, das heisst $G^T = (V^T, E^T, w^T)$ mit:
  - je zwei Knoten u_{in}, u_{out} für jede Kreuzung u.
  - je einer Kante (u_{out}, u_{in}) mit Gewicht $w^T(u_{out}, u_{in}) = b_u$ für jede Kreuzung u.
  - je einer Kante (v_{in}, u_{out}) mit Gewicht $w^T(v_{in}, u_{out}) = g_{uv}$ für jede Strasse (u, v, g_{uv}).

- Anzahl der Knoten und Kanten in Abhängigkeit von n, m (in möglichst knapper Θ-Notation):

  $|V| \in \Theta(n), \quad |E| \in \Theta(n + m)$**.**
  $|V^T| \in \Theta(n), \ |E^T| \in \Theta(n + m)$**.**

- Möglichst effizienter Algorithmus zur Klassifikation der n Kreuzungen:

  Mit Dijkstra in G von s_{in} aus speichern wir für jedes $u \in V$ die Distanz $d(s_{in}, u)$.
  Mit Dijkstra in G^T von t_{out} aus speichern wir für jedes $u \in V^T$ die Distanz $d^T(t_{out}, u)$.

  Nun testen wir für jede Kreuzung u, ob für die entsprechenden Knoten u_{in} in V, V^T gilt, dass $d(s_{in}, u_{in}) + d^T(t_{out}, u_{in}) == d(s_{in}, t_{out})$ ist.
  Falls ja, wird die Kreuzung u befahren, ansonsten nicht.

- Laufzeit in Abhängigkeit von n, m (in möglichst knapper Θ-Notation) inklusive Begründung:

  Zweimal Dijkstra (mit Fibonacci-Heaps):
  $\Theta(|V| \log |V| + |E| + |V^T| \log |V^T| + |E^T|)) = \Theta(n \log n + m)$**.**
  Kreuzungen klassifizieren: $\Theta(|V|) = \Theta(n)$**.**

  Insgesamt: $\Theta(n \log n + m)$**.**

  Ohne Fibonacci-Heaps: $\Theta((n + m) \log(n + m))$**.**

/ 10 P

Theorieaufgabe T3.

In einem weit entfernten Alpenland möchte das Militär seine veraltete Kommunikationsstruktur austauschen. Um das Parlament von diesem teuren Vorhaben zu überzeugen, möchte die Führung einige metrische Angaben über das aktuelle Netzwerk vorlegen. Leider fehlen zur Berechnung dieser Angaben die nötigen IT-Spezialisten. Deshalb werden Sie als Berater angeheuert.

Das Netzwerk ist gegeben als *gerichteter, gewichteter Graph* $G = (V, E, w)$, wobei $w\colon E \to \mathbb{R}$ die Kantengewichtsfunktion ist. Das Netzwerk hat $n = |V|$ Knoten und $m = |E|$ gerichtete Kanten. Für beliebige Knoten $u \neq v$ bezeichnet $d(u, v)$ die Länge eines kürzesten Pfads *von u nach v*. Gesucht sind möglichst effiziente Algorithmen zur Berechnung des *Minimums* über alle Knotenpaare, $\min_{u \neq v} d(u, v)$, und des *Maximums* über alle Knotenpaare, $\max_{u \neq v} d(u, v)$.

Sie erhalten das Netzwerk G als *Adjazenzliste*.

/ 3 P

a) Zusätzlich zum Netzwerk erhalten Sie zwei Hinweise: (i) es gibt $m = \Theta(n^{1.5})$ Kanten und (ii) alle Kanten $(u, v) \in E$ haben nicht-negatives Gewicht $w(u, v) \geq 0$.

Beschreiben Sie einen möglichst effizienten Algorithmus, der den Wert $\min\limits_{u \neq v} d(u, v)$ berechnet. Bestimmen Sie die Laufzeit in Abhängigkeit von n.

Entlang eines beliebigen kürzesten Wegs $\langle v_0, v_1, \ldots, v_\ell \rangle$ gilt unter Annahme nicht-negativer Kantengewichte für jede Kante: $w(v_i, v_{i+1}) \leq \sum_j w(v_j, v_{j+1}) = d(v_0, v_\ell)$.

Der Wert $\min_{u \neq v} d(u, v)$ wird deshalb sicherlich durch zwei benachbarte Knoten angenommen. Es genügt deshalb, den Graph einzulesen, alle Kanten durchzugehen und sich das kleinste Kantengewicht zu merken.

Laufzeit: $\mathcal{O}(n + m) = \mathcal{O}(n^{1.5})$.

/ 3 P

b) Sie erhalten die selben Hinweise wie in Teilaufgabe a): (i) es gibt $m = \Theta(n^{1.5})$ Kanten und (ii) alle Kanten $(u, v) \in E$ haben nicht-negatives Gewicht $w(u, v) \geq 0$.

Beschreiben Sie einen möglichst effizienten Algorithmus, der den Wert $\max\limits_{u \neq v} d(u, v)$ berechnet. Bestimmen Sie die Laufzeit in Abhängigkeit von n.

Wir berechnen die kürzesten Wege zwischen allen Paaren von Knoten. Dazu starten wir von jedem Knoten x aus genau einmal den Algorithmus von Dijkstra (damit kennen wir für alle Knoten y die Distanzen $d(x, y)$).

Schlussendlich iterieren wir einmal über alle Knotenpaare und merken uns die grösste Distanz.

Laufzeit (mit Fibonacci-Heaps): $\mathcal{O}(n \cdot (n \log n + m) + n^2) = \mathcal{O}(n^{2.5})$.
Laufzeit (ohne Fibonacci-Heaps): $\mathcal{O}(n \cdot (n + m \log m) + n^2) = \mathcal{O}(n^{2.5} \log n)$.

Die Skeptische Partei bezweifelt die präsentierten Ergebnisse. Deren Parlamentsfraktion vermutet, dass das Netzwerk auch negative Kantengewichte enthält.

/ 4 P

c) Nun erhalten Sie also *nur noch* den ersten Hinweis: (*i*) $m = \Theta(n^{1.5})$.

Beschreiben Sie einen möglichst effizienten Algorithmus, welcher
– abbricht, falls das Netzwerk einen Kreis mit negativem Gewicht enthält,
– andernfalls die Werte $\min_{u \neq v} d(u, v)$ und $\max_{u \neq v} d(u, v)$ berechnet.

Bestimmen Sie die Laufzeit Ihres neuen Algorithmus in Abhängigkeit von n.

Wir berechnen die kürzesten Wege zwischen allen Paaren von Knoten. Dazu starten verwenden wir den Algorithmus von Johnson.

Dieser verwendet in einem ersten Schritt als Subroutine Bellman-Ford (und bricht ab, falls ein Kreis mit negativem Gewicht gefunden wird) – in einem zweiten Schritt berechnet der Algorithmus von Johnson dann die kürzesten Wege für alle Knotenpaare (mit n Aufrufen von Dijkstra auf einem modifizierten Graphen).

Schlussendlich iterieren wir einmal über alle Knotenpaare und merken uns die kleinste und die grösste Distanz.

Laufzeit (mit Fibonacci-Heaps): $\mathcal{O}(n \cdot m + n \cdot (n \log n + m) + n^2) = \mathcal{O}(n^{2.5})$**.**
Laufzeit (ohne Fibonacci-Heaps): $\mathcal{O}(n \cdot m + n \cdot (n + m \log m) + n^2) = \mathcal{O}(n^{2.5} \log n)$**.**