Technische Universität Wien
Institut für Logic and Computation
Algorithms and Complexity Group

186.866 Algorithmen und Datenstrukturen VU 8.0

2. Test SS 2018

29. Juni 2018

Gruppe A

Machen Sie die folgenden Angaben bitte in deutlicher **Blockschrift**:

Nachname: Vorname:

Matrikelnummer: Unterschrift:

Sie dürfen Ihre Lösungen nur auf diese Angabeblätter schreiben. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden oder Blätter von der Angabe abzulösen. Benutzen Sie unbedingt dokumentenechte Schreibgeräte (keine Bleistifte)!

Die Verwendung von Taschenrechnern, Mobiltelefonen, Smartwatches, Tablets, Digitalkameras, Skripten, Büchern, Mitschriften, Ausarbeitungen oder vergleichbaren Hilfsmitteln ist unzulässig.

Kennzeichnen Sie bei Ankreuzfragen eindeutig, welche Kästchen Sie kreuzen und streichen Sie klar durch, was nicht gewertet werden soll!

	A1	A2	A3	A4	A5	Summe
Erreichbare Punkte:	10	10	8	10	12	50
Erreichte Punkte:						

Viel Erfolg!

Aufgabe A1: Hashing **(10 Punkte)**

a) (4 Punkte) Fügen Sie die folgenden Zahlen in die jeweiligen Hashtabellen ein, indem Sie die angegebenen Hashfunktionen und Strategien für die Kollisionsbehandlung benutzen.

(i) Einzufügende Zahl: **11**
Kollisionsbehandlung: Quadratisches Sondieren mit $c_1 = 0.5$ und $c_2 = 1.5$

Hashfunktion:

$h'(k) = k \bmod 7$

Hashtabelle:

0	1	2	3	4	5	6
				32		20

(ii) Einzufügende Zahl: **16**
Kollisionsbehandlung: Double Hashing **mit der Verbesserung nach Brent**
Wird ein bereits vorhandenes Element verschoben, so muss die neue Position dieses Elementes eindeutig gekennzeichnet werden.

Hashfunktionen:

$h_1(k) = k \bmod 7$
$h_2(k) = (k \bmod 5) + 1$

Hashtabelle:

0	1	2	3	4	5	6
		23		18		13

b) (2 Punkte) Gegeben ist eine Hashtabelle mit Hashfunktion $h(k) = k \bmod 7$. Als Kollisionsbehandlung wird Lineares Sondieren verwendet.

0	1	2	3	4	5	6
105		121				111

Mit welcher Wahrscheinlichkeit kommt eine zufällige (positive) Zahl an die Position 1?

c) (2 Punkte) Kreuzen Sie alle korrekten Aussagen an:

- ☐ Lineares Sondieren ist als Kollisionsstrategie ineffizient, da es zu primären Häufungen kommen kann.
- ☐ Bei offenen Hashverfahren wird Reorganisation eingesetzt, um den Belegungsfaktor hinreichend klein zu halten.
- ☐ Wird als Kollisionsbehandlung „Verkettung der Überläufer“ verwendet, so wird das Löschen durch eine Markierung als „wieder frei“ realisiert.
- ☐ Hashfunktionen sind im Allgemeinen bijektive Funktionen, die einen Schlüssel auf einen Hashwert abbilden und umgekehrt.

(2 Punkte, wenn alle Kreuze korrekt sind, 1 Punkt bei genau einem Fehler, ansonsten 0 Punkte)

d) (2 Punkte) Gegeben ist eine Hashtabelle mit Double Hashing (ohne Verbesserung nach Brent) als Kollisionsbehandlung. Als Hashfunktionen werden $h_1(k) = k \bmod 7$ und $h_2(k) = 1 + (k \bmod 5)$ gewählt.

0	1	2	3	4	5	6
4	8	15		22	12	

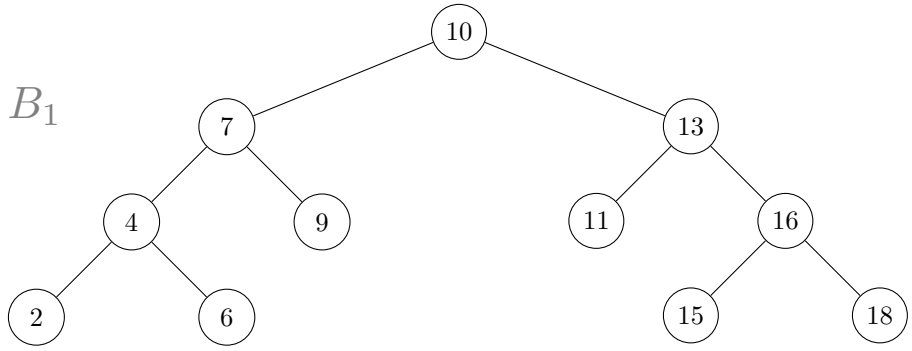
Welche dieser Eingabefolgen führen beim Einfügen in eine anfangs leere Tabelle zu diesem Ergebnis?

- ☐ $(12, 22, 8, 15, 4)$
- ☐ $(8, 22, 15, 12, 4)$
- ☐ $(15, 22, 8, 12, 4)$
- ☐ $(12, 8, 22, 15, 4)$

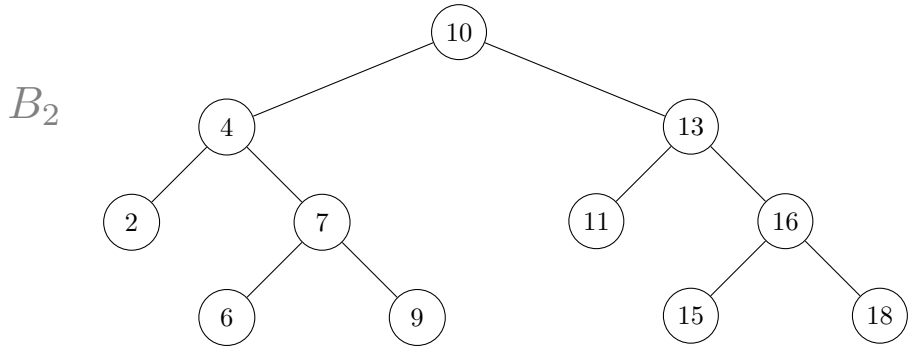
(2 Punkte, wenn alle Kreuze korrekt sind, 1 Punkt bei genau einem Fehler, ansonsten 0 Punkte)

Aufgabe A2: Suchbäume **(10 Punkte)**

a) (6 Punkte) Gegeben ist der folgende AVL-Baum B_1:



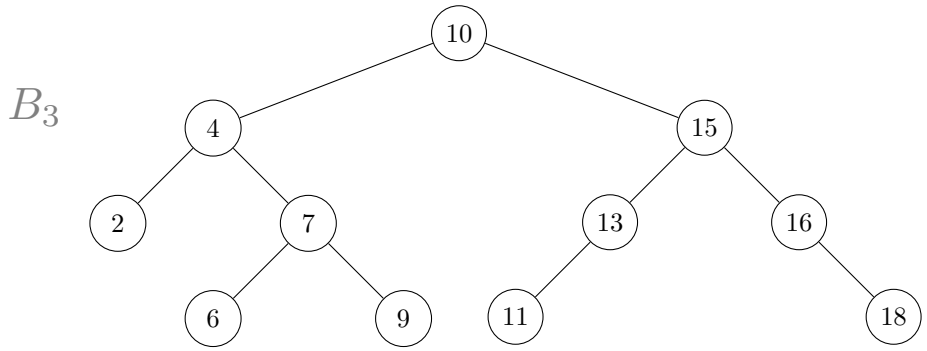
Es wird nun eine ganze Zahl $x \in \{0, \ldots, 20\}$, die noch nicht in B_1 enthalten ist, in B_1 eingefügt. Danach wird B_1 rebalanciert und x wird wieder gelöscht. Dadurch entsteht der folgende AVL-Baum B_2:



Geben Sie alle möglichen Werte für x an, die zu B_2 geführt haben könnten:

Mögliche Werte für x: ____

Nun wird eine ganze Zahl $y \in \{0, \ldots, 20\}$, die noch nicht in B_2 enthalten ist, in B_2 eingefügt. Danach wird B_2 rebalanciert und y wird wieder gelöscht. Dadurch entsteht der folgende AVL-Baum B_3:



Geben Sie alle möglichen Werte für y an, die zu B_3 geführt haben könnten:

Mögliche Werte für y: ____

b) (4 Punkte) Bei einer **In-Order**-Traversierung eines binären Baumes ergibt sich folgende Knotenliste:

$$1, 5, 7, 2, 8, 3, 11$$

Bei einer **Pre-Order**-Traversierung des gleichen Baumes ergibt sich:

$$8, 5, 1, 2, 7, 3, 11$$

Zeichnen Sie den zugrundeliegenden Baum und tragen Sie die entsprechenden Knotenwerte ein.

Aufgabe A3: Reduktionen und NP **(8 Punkte)**

a) (4 Punkte) Seien A, B, C, D Ja/Nein-Probleme in NP und n die Eingabegröße. Angenommen, es gibt

- eine Reduktion von A auf B mit Laufzeit $O(n^2)$,
- eine Reduktion von B auf C mit Laufzeit $O(n^3)$,
- sowie eine Reduktion von C auf D mit Laufzeit $O(n^2 \cdot \log(n))$.

Beantworten Sie dazu die folgenden (voneinander unabhängigen) Fragen:

(i) Geben Sie die engste obere Schranke für die Laufzeit einer Reduktion von A nach D in O-Notation an, die sich aus diesen Annahmen ableiten lässt.

(ii) Nehmen Sie an, C kann in Polynomialzeit gelöst werden. Was folgt daraus für die Komplexität der übrigen Probleme (A, B, D)?

(iii) Nehmen Sie an, dass B NP-schwer ist. Was folgt daraus für die Komplexität der Probleme A, B, C, D?

b) (2 Punkte) Kreuzen Sie alle korrekten Aussagen an:

- ☐ Ein NP-vollständiges Problem ist genau dann in Polynomialzeit lösbar, wenn P=NP.
- ☐ Jedes NP-schwere Problem ist NP-vollständig.
- ☐ Wenn Problem A NP-schwer ist und Problem B in Polynomialzeit auf A reduziert werden kann, dann ist Problem B ebenfalls NP-schwer.
- ☐ Wenn SAT in Linearzeit gelöst werden kann, sind alle NP-vollständigen Probleme in Polynomialzeit lösbar.

(2 Punkte, wenn alle Kreuze korrekt sind, 1 Punkt bei genau einem Fehler, ansonsten 0 Punkte)

c) (2 Punkte) Geben Sie den Namen von vier aus der Vorlesung bekannten NP-vollständigen Problemen an.

Aufgabe A4: Dynamische Programmierung **(10 Punkte)**

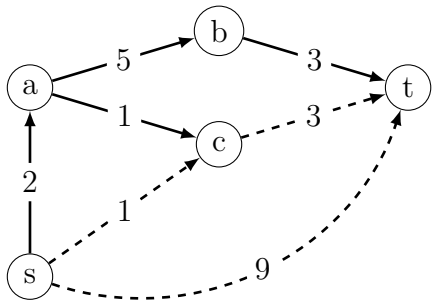
Wir definieren folgendes EXTENDED SHORTEST PATH Problem: Gegeben ist ein gerichteter Graph $G = (V, E)$ mit einem positiven Kantengewicht c_{uv} für jede Kante $(u, v) \in E$ und zwei speziellen Knoten s und t. Die Kantenmenge E besteht aus roten Kanten E_r und schwarzen Kanten E_s, d.h. $E = E_r \cup E_s$ und $E_r \cap E_s = \emptyset$. Ein *guter* Pfad ist ein Pfad, welcher maximal eine rote Kante benutzt. Aufgabe ist es, den kürzesten guten Pfad von s nach t zu finden.

In dieser Aufgabe wird das Problem EXTENDED SHORTEST PATH mittels dynamischer Programmierung durch eine Erweiterung des aus der Vorlesung bekannten Bellman-Ford Algorithmus gelöst.

Dabei werden die folgenden beiden Arrays dynamisch berechnet:

- `NoRed`, welches die Längen der Pfade von allen Knoten $x \in V$ zu t beinhaltet, die keine roten Kanten enthalten.
- `Good`, welches die Längen der Pfade von allen Knoten $x \in V$ zu t beinhaltet, mit höchstens einer roten Kante.

a) (4 Punkte) Gegeben sei die folgende Probleminstanz. Die roten Kanten sind strichliert gezeichnet.



(i) Vervollständigen Sie die beiden Arrays `NoRed` und `Good` für diese Instanz. Genau wie in der Vorlesung repräsentieren die Spalten 0-4 die Längen der Pfade mit der entsprechenden Anzahl an Kanten.

`NoRed`	0	1	2	3	4
t	0	0	0	0	0
a	∞	∞			
b	∞	3			
c	∞	∞			
s	∞	∞			

`Good`	0	1	2	3	4
t	0	0	0	0	0
a	∞	∞			
b	∞	3			
c	∞	3			
s	∞	9			

(ii) Welche Länge hat der kürzeste gute Pfad von s nach t in dieser Instanz? Wie kommen Sie auf Basis der Arrays zum Ergebnis?

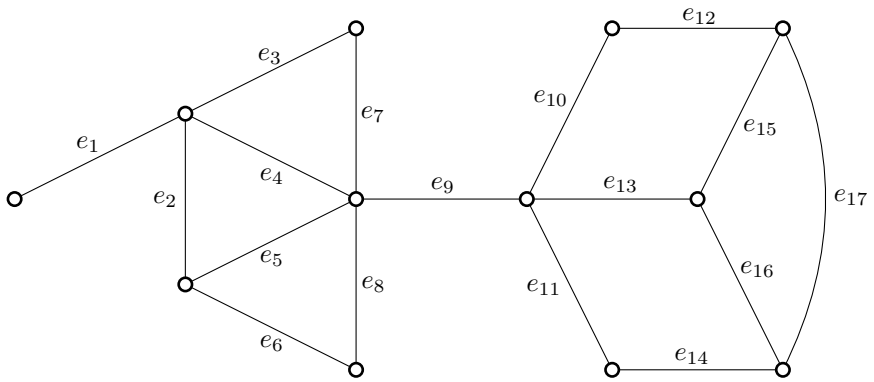
b) (6 Punkte) Nun ist ein Algorithmus für das Problem Extended Shortest Path zu finden. Kreuzen Sie je grau markiertem Block eine Zeile an, sodass die Arrays richtig berechnet werden und das Problem korrekt gelöst wird.

(richtiges Kreuz +2 Punkte; falsches Kreuz/mehrere Kreuze im Block -2 Punkte; kein Kreuz 0 Punkte; negative Summe wird auf 0 gesetzt)

```
Ext-Short-Path(G,s,t):
foreach node v ∈ V
        NoRed[0,v] ← ∞
NoRed[0,t] ← 0
for i ← 1 bis n − 1
        foreach Knoten v ∈ V
                NoRed[i,v] ← NoRed[i − 1,v]
    ☐   foreach schwarze Kante (v,w) ∈ E_s
    ☐   foreach Kante (v,w) ∈ E
    ☐   foreach rote Kante (v,w) ∈ E_r
                NoRed[i,v] ← min{NoRed[i,v], c_vw + NoRed[i − 1,w]}
foreach node v ∈ V
        Good[0,v] ← ∞
Good[0,t] ← 0
for i ← 1 bis n − 1
        foreach Knoten v ∈ V
                Good[i,v] ← Good[i − 1,v]
        foreach schwarze Kante (v,w) ∈ E_s
            ☐ Good[i,v] ← min{Good[i,v], c_vw + NoRed[i − 1,w]}
            ☐ Good[i,v] ← min{Good[i,v], c_vw + Good[i − 1,w]}
            ☐ Good[i,v] ← min{Good[i,v], NoRed[i,w]}
        foreach rote Kante (v,w) ∈ E_r
            ☐ Good[i,v] ← c_vw + NoRed[i − 1,w]
            ☐ Good[i,v] ← min{Good[i,v], c_vw + NoRed[i − 1,w]}
            ☐ Good[i,v] ← min{Good[i,v], c_vw + Good[i − 1,w]}
return Good[n − 1, s]
```

Aufgabe A5: Approximation und Branch-and-Bound **(12 Punkte)**

a) (4 Punkte) Betrachten Sie den 2-Approximationsalgorithmus für das Problem *Kleinstes Vertex Cover* aus der Vorlesung. Das Ergebnis des Algorithmus hängt von der Reihenfolge ab, in der die Kanten betrachtet werden.



Geben Sie für den abgebildeten Graphen eine gültige Kantenreihenfolge an, so dass die Lösung optimal ist.

Geben Sie außerdem eine Kantenreihenfolge an, so dass die Lösung doppelt so viele Knoten enthält wie ein kleinstes Vertex Cover.

b) (2 Punkte) Betrachten Sie die folgende Instanz des Rucksackproblems. Die Gewichte und Werte der Gegenstände sind in der Tabelle angegeben. Die Kapazität des Rucksacks beträgt $G = 10$.

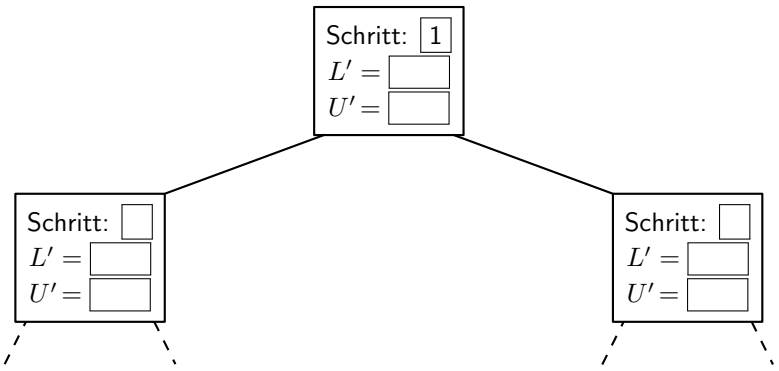
Gegenstand	A	B	C	D
Gewicht	4	5	6	3
Wert	20	35	72	18
Verhältnis				

Berechnen Sie die Wert-Gewichts-Verhältnisse aller Gegenstände und tragen Sie diese in obiger Tabelle ein. Geben Sie die Reihenfolge an, in der die Gegenstände betrachtet werden, wenn Sie den Branch-and-Bound Algorithmus der Vorlesung anwenden.

Reihenfolge: | |

c) (5 Punkte) Wenden Sie den verbesserten Branch-and-Bound-Algorithmus aus der Vorlesung auf die Instanz an. Nutzen Sie dabei die Depth-first Strategie zur Auswahl des nächsten Teilproblems. Ergänzen Sie zur Lösung der Aufgabe den untenstehenden begonnenen B&B Baum.

Geben Sie in jedem Knoten an, in welchem Schritt er besucht wird und welchen Wert die zugehörigen unteren und oberen Schranken haben, bzw. markieren Sie, wenn es keine gültige Lösung geben kann. Geben Sie an den Kanten klar an, welche Branching-Entscheidung getroffen wird. Erweitern Sie den Baum nach Bedarf und zeichnen Sie die zusätzlich benötigten Kanten und Knoten ein.



d) (1 Punkt) Geben Sie die optimale Lösung und den zugehörigen Lösungswert an.