TU Wien
Institut für Logic and Computation
Algorithms and Complexity Group

186.866 Algorithmen und Datenstrukturen VU 8.0

2. Test, 2023S

30. Juni 2023

Gruppe A

Nachfolgende Angaben gut **leserlich in BLOCKSCHRIFT** machen.

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Sie dürfen die Lösungen nur auf diesen Prüfungsbogen schreiben. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie dokumentenechte Stifte (keine Bleistifte, etc.).

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. Streichen Sie Passagen, die nicht gewertet werden sollen, deutlich durch. Schreiben Sie leserlich, unleserliche Antworten werden nicht gewertet.

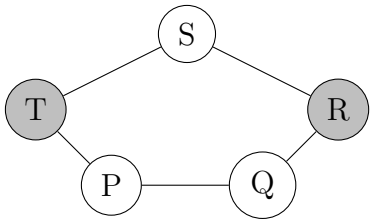
	A1	A2	A3	A4	A5	Summe
Erreichbare Punkte:	20	18	20	20	22	100
Erreichte Punkte:						

Viel Erfolg!

Aufgabe A1: P und NP, Spezialfälle **(_____ / 20 Punkte)**

a) (10 Punkte) Betrachten Sie einen Graphen $G = (V, E)$ und eine nichtnegative ganze Zahl $\ell \in \mathbb{N} \cup \{0\}$. Eine Menge von Knoten $D \subseteq V$ heißt *ℓ-Deletion Set*, wenn nach Löschen aller Knoten in D jeder verbleibende Knoten höchstens ℓ Nachbarn hat. Das heißt, für jeden Knoten $u \in V \setminus D$ gibt es nicht mehr als ℓ Knoten $v \in V \setminus D$, sodass $(u, v) \in E$.

Nachfolgend finden Sie ein Beispiel für ein 1-Deletion Set $D = \{R, T\}$ (grau markiert) in einem Kreis:



Das Deletion Set Problem ist wie folgt definiert:

Deletion Set: Gegeben sei ein Graph $G = (V, E)$ und ganze Zahlen $k > 0$, $\ell \geq 0$. Existiert ein ℓ-Deletion Set $D \subseteq V$, sodass $|D| \leq k$?

Wir wollen zeigen, dass Deletion Set NP-vollständig ist.

(i) Um zu zeigen, dass ein Problem in NP ist, geben wir ein geeignetes Zertifikat und einen passenden Zertifizierer an. Welche Eigenschaften muss ein geeigneter Zertifizierer im Allgemeinen erfüllen?

(ii) Geben Sie ein geeignetes Zertifikat und einen passenden Zertifizierer für Deletion Set an. Argumentieren Sie kurz, dass Ihr Zertifizierer die Eigenschaften aus der vorigen Unteraufgabe erfüllt.

(iii) Zeigen Sie, dass Deletion Set NP-schwer ist, indem Sie eine Reduktion vom Vertex Cover Problem aus der Vorlesung auf Deletion Set angeben.

Hinweis: Es genügt, auf den Fall $\ell = 0$ zu reduzieren. Es gibt eine einfache Reduktion.

b) (10 Punkte) Seien A, B und C Ja/Nein-Probleme und n jeweils die Eingabegröße. Nehmen Sie an, es gibt

- eine Reduktion von Deletion Set nach A in Zeit n,
- eine Reduktion von Deletion Set nach B in Zeit n^3,
- eine Reduktion von Deletion Set nach C in Zeit 4^n,
- eine Reduktion von 3-SAT nach Deletion Set in Zeit n^3,
- eine Reduktion von C nach A in Zeit n^3,
- eine Reduktion von A nach 3-Sat in Zeit n.

Aus Unteraufgabe a) wissen wir, dass Deletion Set NP-vollständig ist.

(i) Geben Sie die engste obere Schranke für die Laufzeit einer Reduktion von C auf Deletion Set in O-Notation an, die sich aus diesen Annahmen ableiten lässt. Begründen Sie Ihre Antwort kurz.

(ii) Was folgt aus den obigen Reduktionen für die Komplexität der Probleme A, B und C? Beantworten Sie die Frage, indem Sie **alle** zutreffenden Felder in der folgenden Tabelle ankreuzen.

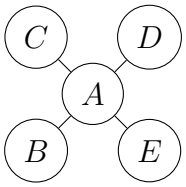
	in P	in NP	NP-schwer	NP-vollständig
A				
B				
C				

Aufgabe A2: Branch-and-Bound **(_____ / 18 Punkte)**

Für die folgenden Teilaufgaben betrachten wir das Minimierungsproblem Min Deletion Set. Erinnern Sie sich an die folgende Definition aus Aufgabe 1: Eine Menge von Knoten $D \subseteq V$ heißt *ℓ-Deletion Set*, wenn nach Löschen aller Knoten in D jeder verbleibende Knoten höchstens ℓ Nachbarn hat.

Min Deletion Set: Gegeben: Ein Graph $G = (V, E)$ und eine Zahl $\ell \in \mathbb{N}$. Gesucht: Eine kleinstmögliche Knotenteilmenge $D \subseteq V$, die ein ℓ-Deletion Set für G ist.

a) (4 Punkte) Betrachten Sie einen Stern mit 4 Blättern, also den folgenden Graphen:



Nehmen Sie an, dass $\ell = 2$ gilt (nur für diese Teilaufgabe). Beschreiben Sie **alle** ℓ-Deletion Sets für den Stern mit 4 Blättern, die inklusionsminimal sind. Das heißt, alle ℓ-Deletion Sets D, sodass jede echte Teilmenge $D' \subset D$ *kein* ℓ-Deletion Set ist. Es ist keine Begründung nötig.

b) (4 Punkte) Ein Branch-and-Bound Algorithmus soll Min Deletion Set lösen. Ein Teilproblem wird dabei durch ein Tupel (H, C) repräsentiert, in dem C eine Menge von Knoten ist, die schon aus dem Eingabegraph G gelöscht wurden und H der verbleibende Graph nach Löschung der Knoten in C. Sei nun (H, C) ein Teilproblem bei dem H einen Knoten v enthält, der $\ell + 1$ Nachbarn in H hat. Beschreiben Sie, wie der Algorithmus das Teilproblem (H, C) in kleinere Teilprobleme zerteilen kann. Eine kurze und klare Beschreibung der Idee ohne Begründung genügt.

Hinweis: Betrachten Sie v als Zentrum eines Sterns mit $\ell + 1$ Blättern.

c) (6 Punkte) Betrachten Sie den folgenden Algorithmus. Er bekommt als Eingabe eine Instanz (G, ℓ) von Min Deletion Set und gibt eine natürliche Zahl zurück.

Function `MDS-A`(G, ℓ)**:**
  $A \leftarrow 0$
  while *Es existiert ein Knoten v mit mindestens $\ell + 1$ Nachbarn in G*
    Entferne v und $\ell + 1$ beliebige Nachbarn von v aus G
    $A \leftarrow A + 1$
  return A

(i) Genau eine der folgenden beiden Aussagen ist richtig. Kreuzen Sie die richtige Aussage an:

`MDS-A` berechnet für die Größe eines kleinstmöglichen ℓ-Deletion Sets für den Eingabegraphen G eine

☐ untere Schranke ☐ obere Schranke.

(ii) Begründen Sie Ihre Antwort zu (i). Das heißt, geben Sie einen Beweis für die richtige Aussage oder ein Gegenbeispiel für die falsche Aussage an.

d) (4 Punkte) Welche Aussagen für Branch-and-Bound Verfahren sind korrekt? Kreuzen Sie Zutreffendes an.

(+1 Punkt für jede richtige, -1 Punkt für jede falsche und 0 Punkte für keine Antwort, keine negativen Gesamtpunkte auf diese Unteraufgabe)

(Q1) Die Wahl der Heuristiken für U' und L' ist entscheidend für die Effizienz.

☐ Wahr ☐ Falsch

(Q2) Branch-and-Bound für ein Minimierungsproblem funktioniert im Allgemeinen umso besser, je kleiner die lokalen unteren Schranken sind.

☐ Wahr ☐ Falsch

(Q3) Die Reihenfolge der Auswahl des nächsten Teilproblems ist für die Optimalität der am Ende zurückgegebenen Lösung von Bedeutung.

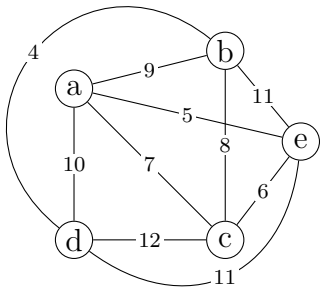
☐ Wahr ☐ Falsch

(Q4) Bei einem Maximierungsproblem kann die Branch-and-Bound Suche für eine Teilinstanz abgebrochen werden, wenn die globale untere Schranke größer als die lokale obere Schranke ist.

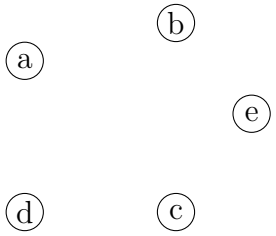
☐ Wahr ☐ Falsch

Aufgabe A3: Approximationsalgorithmen **(_____ / 20 Punkte)**

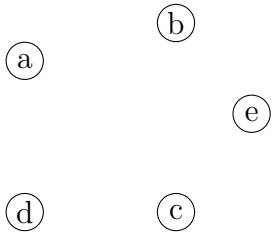
a) (14 Punkte) Betrachten Sie folgende symmetrische TSP Instanz und wenden Sie die Spanning-Tree-Heuristik darauf an.



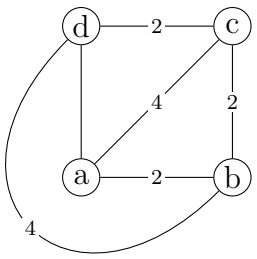
(i) Zeichnen Sie eine Eulertour, die von der Spanning-Tree-Heuristik berechnet wird, ein. Wie ist die Länge dieser Eulertour?



(ii) Leiten Sie von der Eulertour eine Lösung für das TSP ab und zeichnen Sie diese ein. Wie lang ist diese Tour?



(iii) Ergänzen Sie das fehlende Kantengewicht in der folgenden symmetrischen TSP Instanz, sodass die Spanning-Tree Heuristik die Gütegarantie von 2 **nicht** erfüllt.



b) (3 Punkte) Betrachten Sie den aus der Vorlesung bekannten 2-Approximationsalgorithmus für das CENTER SELECTION Problem. Sei x eine Probleminstanz mit optimalem Lösungswert $C_{\text{OPT}}(x)$ und $C_{\text{A}}(x)$ der Wert der von dem Algorithmus berechneten Lösung. Welche der folgenden Aussagen treffen in jedem Fall zu? Kreuzen Sie Zutreffendes an.

☐ $C_{\text{A}}(x) \leq \frac{C_{\text{OPT}}(x)}{2}$

☐ $C_{\text{A}}(x) > 2 \cdot C_{\text{OPT}}(x)$

☐ $C_{\text{A}}(x) \leq 2 \cdot C_{\text{OPT}}(x)$

☐ $C_{\text{OPT}}(x) \leq C_{\text{A}}(x)$

c) (3 Punkte) Betrachten Sie den aus der Übung bekannten $(1-1/k)$-Approximationsalgorithmus für das MAX-K-PART Problem und fixieren Sie den Wert des Parameters k auf $k = 3$. Sei x eine Probleminstanz mit optimalem Lösungswert $C_{\text{OPT}}(x)$ und $C_{\text{B}}(x)$ der Wert der von dem Algorithmus berechneten Lösung. Welche der folgenden Aussagen treffen in jedem Fall zu? Kreuzen Sie Zutreffendes an.

☐ $C_{\text{B}}(x) \leq \frac{C_{\text{OPT}}(x)}{3}$

☐ $C_{\text{B}}(x) \geq 2/3 \cdot C_{\text{OPT}}(x)$

☐ $C_{\text{B}}(x) \leq 2/3 \cdot C_{\text{OPT}}(x)$

☐ $C_{\text{OPT}}(x) - C_{\text{B}}(x) \geq 0$

Aufgabe A4: Heuristische Verfahren **(_____ / 20 Punkte)**

Wir betrachten das Optimierungsproblem MINIMUM SET COVER, bei dem für eine Menge U von Elementen und eine Menge $\mathcal{S} = \{S_1, \ldots, S_m\}$ von Teilmengen von U ein minimales Set Cover $\mathcal{C}$ gefunden werden muss.

Anmerkung: Wie aus der Vorlesung bekannt ist $\mathcal{C} \subseteq \mathcal{S}$ ein Set Cover von $\mathcal{S}$ genau dann wenn $\bigcup_{S_i \in \mathcal{C}} S_i = U$. Weiters ist $\mathcal{C}$ ein *minimales* Set Cover von $\mathcal{S}$ wenn es kein anderes Set Cover $\mathcal{C}' \subseteq \mathcal{S}$ von $\mathcal{S}$ gibt, sodass $|\mathcal{C}'| < |\mathcal{C}|$.

a) (8 Punkte) Gegeben ist eine Konstruktionsheuristik für MINIMUM SET COVER:

```
ConstructSetCover(U, S = {S_1, ..., S_m}):
    C ← ∅
    W ← U
    while W ≠ ∅
        T ← ∅
        for i = 1, ..., m
            if |S_i ∩ W| > |T ∩ W| then
                T ← S_i
        if T = ∅ then
            return C
        C ← C ∪ {T}
        W ← W \ T
    return C
```

Führen Sie `ConstructSetCover` mit folgendem Input aus:

$$U = \{1, 2, 3, 4, 5, 6, 7\},\ \mathcal{S} = \{S_1, S_2, S_3, S_4, S_5\},\ \text{wobei}$$
$$S_1 = \{1, 2\},\ S_2 = \{2, 3, 4\},\ S_3 = \{5, 6\},\ S_4 = \{7\},\ S_5 = \{4, 5\}.$$

Geben Sie $\mathcal{C}$ und W nach jedem Durchlauf der while-Schleife an. Sollten Sie eine oder mehrere Zeilen nicht benötigen, lassen Sie diese frei.

Durchlauf	$\mathcal{C}$	W
1		
2		
3		
4		
5		

b) (8 Punkte) Gegeben ist eine Nachbarschaftsstruktur N für Minimum Set Cover:

$\mathcal{C}' \in N(\mathcal{C})$ genau dann wenn $\mathcal{C}'$ ein Set Cover von $\mathcal{S}$ ist und ausgehend von $\mathcal{C}$ durch eine der beiden Operationen erzeugt werden kann:

- entferne eine einzige Menge aus $\mathcal{C}$, also $\mathcal{C}' = \mathcal{C} \setminus \{S_i\}$ für $S_i \in \mathcal{C}$;
- entferne zwei unterschiedliche Mengen aus $\mathcal{C}$ und füge eine Menge von $\mathcal{S} \setminus \mathcal{C}$ hinzu, also $\mathcal{C}' = (\mathcal{C} \setminus \{S_i, S_j\}) \cup \{S_\ell\}$ für $S_i, S_j \in \mathcal{C}$, wobei $i \neq j$, und $S_\ell \in \mathcal{S} \setminus \mathcal{C}$.

Weiters ist folgende Instanz von Minimum Set Cover gegeben:

$$U = \{1, 2, 3, 4, 5, 6\},\ \mathcal{S} = \{S_1, S_2, S_3, S_4, S_5, S_6\},\ \text{wobei}$$
$$S_1 = \{1\},\ S_2 = \{2, 3\},\ S_3 = \{4\},\ S_4 = \{4, 5, 6\},\ S_5 = \{6\},\ S_6 = \{1, 2, 3\}.$$

(i) Bestimmen Sie $N(\mathcal{C})$ für $\mathcal{C} = \mathcal{S} \setminus \{S_6\}$.

(ii) Wenden Sie lokale Suche mit Nachbarschaftsstruktur N und Ausgangslösung $\mathcal{C} = \mathcal{S} \setminus \{S_6\}$ auf die obige Instanz an, bis ein lokales Optimum erreicht wird. Geben Sie die Zwischenschritte an, also jedes Set Cover welches im Laufe der lokalen Suche ausgewählt wird.

c) (4 Punkte) Im Folgenden bezeichnet `ConstructSetCover` die Konstruktionsheuristik aus Unteraufgabe a) und N die Nachbarschaftsstruktur aus Unteraufgabe b).

Aussage: Seien $\mathcal{S}$ und U ein beliebiger Input für Minimum Set Cover. Wenn ein Set Cover $\mathcal{C} \subseteq \mathcal{S}$ durch `ConstructSetCover` konstruiert wurde, dann kann $\mathcal{C}$ nicht mehr mittels N verbessert werden, es gilt also immer $N(\mathcal{C}) = \emptyset$.

Widerlegen Sie die obige Aussage durch ein entsprechendes Gegenbeispiel. Geben Sie dazu einen Input für Minimum Set Cover an und argumentieren Sie, dass dieser Input ein Gegenbeispiel darstellt.

Aufgabe A5: Dynamisches Programmieren **(_____ / 22 Punkte)**

a) (14 Punkte) Im Teilsummenproblem ist eine Menge A von positiven ganzen Zahlen und eine positive ganze Zahl *sum* gegeben. Gefragt ist nun, ob eine Teilmenge von A existiert, deren Summe exakt *sum* ergibt.

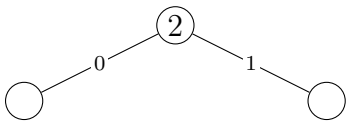
Der unten abgebildete Algorithmus in Pseudocode (`isSubsetSum`) stellt einen Ansatz dar, um das Teilsummenproblem rekursiv zu lösen. Dabei wird angenommen, dass die Menge A als Array gegeben ist und n Elemente enthält. Der Rückgabewert ist genau dann *True*, wenn die Summe der Elemente einer Teilmenge von A exakt *sum* entspricht.

```
Function isSubsetSum(A, n, sum)
    if sum = 0 then
        return True
    if n = 0 then
        return False

    if sum < A[n − 1] then
        return isSubsetSum(A, n − 1, sum)
    else
        included ← isSubsetSum(A, n − 1, sum − A[n − 1])
        excluded ← isSubsetSum(A, n − 1, sum)
        return (included ∨ excluded)
```

(i) Gegeben sind ein Array $A = [3, 4, 8, 2]$ und $sum = 7$.

Vervollständigen Sie für diese Werte den unten angegebenen Rekursionsbaum von `isSubsetSum` bis zur benötigten Tiefe. Tragen Sie den Wert des im Aufruf betrachteten Elements in den Knoten ein. Tragen Sie *True* oder *False*, je nach Rückgabewert, in die Blattknoten ein. Kennzeichnen Sie Kanten mit 1, wenn ein Element Teil der Summe ist oder mit 0, falls ein Element nicht Teil der Summe ist.



(ii) Das Teilsummenproblem kann auch mit einem iterativen Dynamischen Programm gelöst werden. Dabei wird ein zweidimensionales Array *store* verwendet, das den Rückgabewert speichert. Dabei wird *store* bottom-up gefüllt. Vervollständigen Sie folgendes Dynamisches Programm:

```
Function isSubsetSum(A, n, sum)
    // Initialisierung eines leeren zweidimensionalen Arrays
    store ← new Array(n + 1, sum + 1)

    // Basisfälle
    for i = 0, ..., n − 1
        for j = 0, ..., sum
            if i = [          ] then
                store[i][j] = [          ]
            if j = [          ] then
                store[i][j] = [          ]

    // Iteratives Befüllen der Tabelle
    for i = 0, ..., n − 1
        for j = 0, ..., sum
            if j < A[i] then
                store[i][j] = [                    ]
            else
                store[i][j] = [                    ]

    return store[n][sum]
```

Unteraufgabe b): siehe nächste Seite.

b) (8 Punkte) Gegeben sei eine Instanz des aus der Vorlesung bekannten Rucksackproblems mit Kapazität $G = 7$ und den folgenden sechs Gegenständen.

	Gegenstand					
	o_1	o_2	o_3	o_4	o_5	o_6
Wert	2	4	6	3	3	7
Gewicht	2	1	4	1	2	3

Die Wertetabelle M nach Ausführung des Dynamischen Programms lautet:

Gegenstände \ Kapazität	0	1	2	3	4	5	6	7
$\emptyset$	0	0	0	0	0	0	0	0
$\{o_1\}$	0	0	2	2	2	2	2	2
$\{o_1, o_2\}$	0	4	4	6	6	6	6	6
$\{o_1, o_2, o_3\}$	0	4	4	6	6	10	10	12
$\{o_1, o_2, o_3, o_4\}$	0	4	7	7	9	10	13	13
$\{o_1, o_2, o_3, o_4, o_5\}$	0	4	7	7	10	10	13	13
$\{o_1, o_2, o_3, o_4, o_5, o_6\}$	0	4	7	7	11	14	14	17

(i) Kreisen Sie in der Tabelle all jene Felder ein, die der Algorithmus `Find-Solution(M)` aus der Vorlesung bei der Berechnung der Lösungsmenge S ausliest und verwendet.

(ii) Geben Sie die Menge der Gegenstände in der Lösungsmenge S an.

$S = \{$ [ ] $\}$