

2023_t2_A

⚠ Disclaimer

Alles was hier drinnen steht kann Fehler enthalten! Falls dir etwas auffällt melde dich gerne auf Discord bei mir (@xmozz)

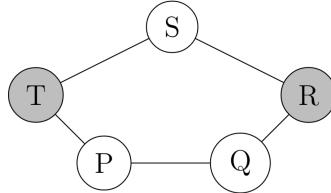
A1: P und NP, Spezialfälle

Stoff: 9. Polynominalzeitreduktionen und 10. NP-Vollständigkeit Spezialfälle

a)

- 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)

- (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?

Im allgemeinen muss ein geeigneter Zertifizierer eine Ja Antwort geben wenn die Antwort des ja/nein Problems von dem Ursprungsproblem ja ist und ein nein wenn's andersherum ist.

Außerdem muss er das Problem polynomiell prüfen können. Und wenn das Zertifikat exponentiell wächst, gibt der Zertifizierer keine Antwort.

(ii)

- (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.

Geeignetes Zertifikat: Menge D die gelöscht werden soll.

Geeigneter Zertifizierer: Algorithmus der die Größe von D prüft und mit k vergleicht und anschließend überprüft ob die Knotengrade der übrigen Knoten kleiner als l sind.

Wenn $|D| < k$ und $\forall i : \deg(v_i) < l$ dann wird es akzeptiert.

(iii)

- (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.

Man kann Vertex Cover auf Deletion-Set reduzieren indem man sich für das Vertex Cover das Deletion Set mit $l = 0$ ausrechnet. Die Knoten die man löschen würde sind dann unser Vertex Cover.

Da man das Reduzieren kann, und daher Deletion Set mindestens genau so schwer ist wie Vertex Cover (NP-Schwer) ist Deletion Set NP-Schwer

b)

- 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)

- (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.

C --> A: n^3

A --> 3-SAT: n

3-SAT --> Deletion-Set: n^3

$$\begin{aligned} \Rightarrow ((n^3)^1)^3 &= n^{3 \cdot 1 \cdot 3} = n^9 \\ \Rightarrow O(n^9) \end{aligned}$$

(ii)

- (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		X	X	X
B		X	X	X
C			X	

- A ist NP-vollständig, weil es in $O(n^3)$ auf 3-SAT reduziert werden kann. Da 3-SAT NP-vollständig ist und A in NP liegt, folgt daraus die NP-Vollständigkeit von A.
- B ist ebenfalls NP-vollständig, weil es eine polynomielle Reduktion ($O(n^3)$) von Deletion Set auf B gibt. Da Deletion Set NP-vollständig ist und die Reduktion polynomial ist, folgt die NP-Vollständigkeit von B.
- C ist NP-schwer, aber nicht unbedingt NP-vollständig. Der Grund ist, dass die Reduktion von Deletion Set auf C exponentiell ($O(4^n)$) ist, was darauf hindeutet, dass C möglicherweise nicht in NP liegt. Da NP-vollständige Probleme auch in NP sein müssen, können wir für C nur die NP-Schwere sicher bestätigen.

A2: Branch-and-Bound

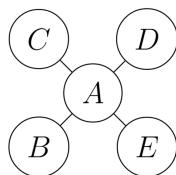
Stoff: 11. Branch and Bound

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)

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.

- {A}
- {B,C}
- {B,D}
- {B,E}
- {C,D}
- {C,E}
- {D,E}

b)

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 Eingabograph 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.

- **Teilproblem 1:** Knoten v wird in die Menge C aufgenommen (also gelöscht).
- **Teilproblem 2:** Einer der $\ell + 1$ Nachbarn von v wird in die Menge C aufgenommen (also gelöscht). Dies wird für jeden Nachbarn von v als separate Option betrachtet (oder es wird ein Nachbar

ausgewählt und der Prozess rekursiv fortgesetzt, bis der Grad von v reduziert ist).

c)

- 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 Eingabographen 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.

Begründung: Da wir sowohl beliebige Nachbaren als auch den Knoten selbst entfernen, wir aber einen möglichst kleinen Deletion Set haben wollen ist das eine untere Schranke.

d)

- 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

Q2: Je **größer** die lokalen unteren Schranken sind, desto besser können Teilprobleme abgeschnitten werden, da sie schneller die aktuelle globale obere Schranke übersteigen

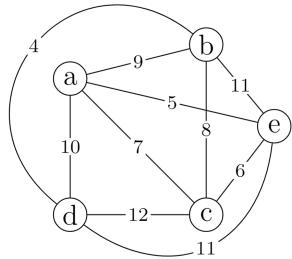
Q3: Nein nur wie schnell man sie findet.

A3: Approximationsalgorithmen

Stoff: 13. Approximation

a)

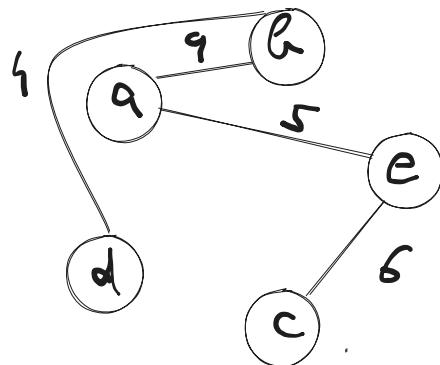
- a) (14 Punkte) Betrachten Sie folgende symmetrische TSP Instanz und wenden Sie die Spanning-Tree-Heuristik darauf an.



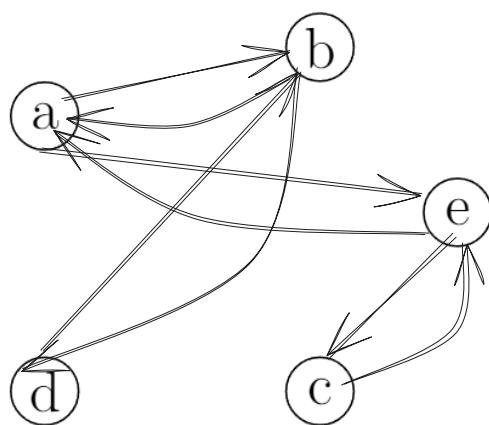
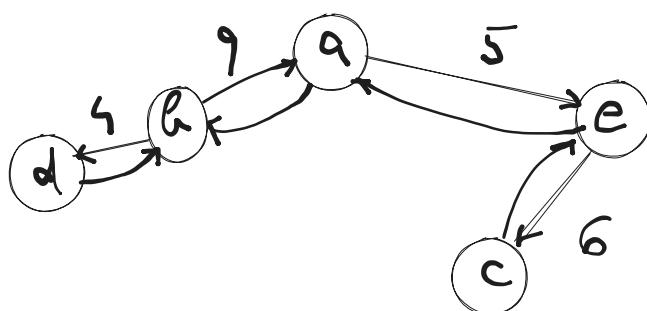
(i)

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

1.) MST



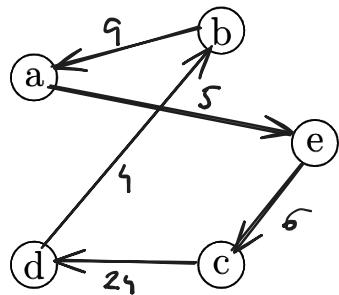
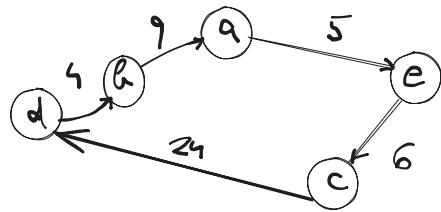
2.) Kanten verdoppeln



Länge: 48

(ii)

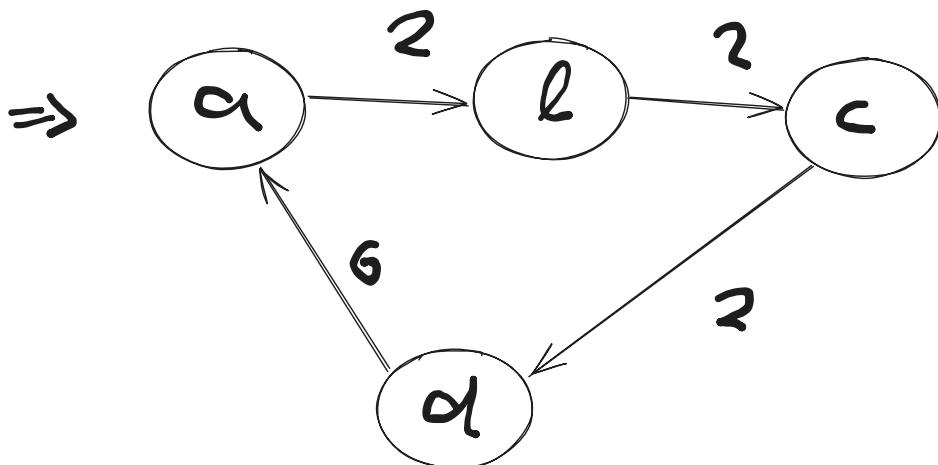
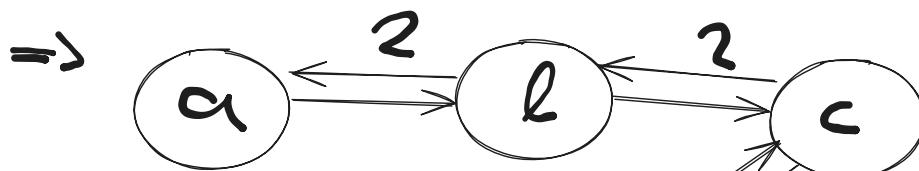
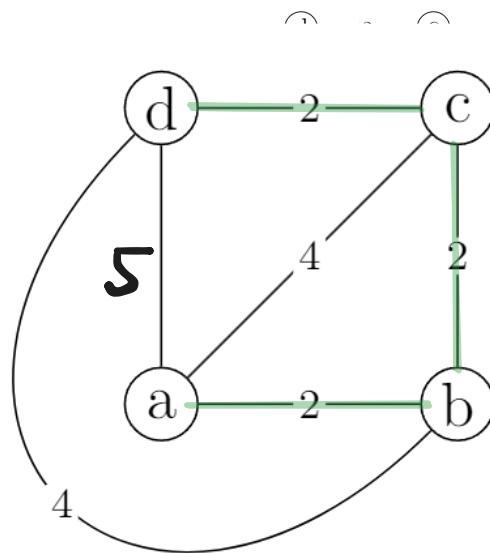
- (ii) Leiten Sie von der Eulertour eine Lösung für das TSP ab und zeichnen Sie diese ein. Wie lang ist diese Tour?



Länge h P

(iii)

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



$$\Rightarrow L = \underline{\underline{12}}$$

Die Kante **(b,d) = 5** ist **ein** möglicher Wert, der die Bedingung erfüllt, dass die Spanning-Tree-Heuristik die Gütegarantie von 2 **nicht** erfüllt.

Die Begründung, warum 5 funktioniert (oder jeder andere Wert, der die Dreiecksungleichung verletzt):

Die Spanning-Tree-Heuristik für TSP hat eine **Gütegarantie von 2** (d.h. die gefundene Tour ist höchstens doppelt so lang wie die optimale Tour) **NUR DANN**, wenn die Instanz des TSP die Dreiecksungleichung erfüllt.

Die Dreiecksungleichung besagt, dass für jede Dreiergruppe von Knoten u,v,w in einem Graphen die Summe der Gewichte zweier Seiten immer größer oder gleich der dritten Seite sein muss:
 $W(u,v) + W(v,w) \geq W(u,w)$.

Wenn diese Bedingung nicht erfüllt ist, d.h., wenn es eine Kante gibt, deren Gewicht **größer** ist als die Summe der Gewichte zweier anderer Kanten, die einen Umweg zwischen den gleichen beiden Endpunkten bilden ($W(u,w) > W(u,v) + W(v,w)$), dann ist die Dreiecksungleichung verletzt.

In unserem Graphen mit dem fehlenden Gewicht (b,d):

Betrachten wir die Knoten b,c,d. Die Kanten sind $W(b,c)=2$ und $W(c,d)=2$. Die Summe dieser beiden Kanten ist $W(b,c) + W(c,d) = 2 + 2 = 4$.

Wenn wir nun die Kante **W(b,d) auf 5 setzen**: Dann ist $W(b,d)=5$. Vergleichen wir dies mit dem Umweg über c: $5 > 4$.

Dies ist eine Verletzung der Dreiecksungleichung. Da die Dreiecksungleichung verletzt ist, ist die Voraussetzung für die Gütegarantie von 2 der Spanning-Tree-Heuristik nicht gegeben. **Daher ist die Garantie von 2 nicht erfüllt.**

Es geht hier nicht darum, dass die tatsächliche Heuristiklösung plötzlich mehr als doppelt so lang ist wie die optimale Lösung (was in kleinen Graphen schwer zu zeigen ist, da die Anzahl der Touren begrenzt ist und die Heuristik oft zufällig eine gute Tour findet). Es geht darum, dass die **theoretische Garantie** von 2 nur unter der Bedingung der Dreiecksungleichung gilt. Wenn diese Bedingung nicht erfüllt ist, kann man nicht mehr garantieren, dass das Ergebnis höchstens zweimal so lang ist.

Deshalb ist **5** (oder jeder andere Wert größer als 4 für (b,d)) eine korrekte Antwort

b) und c)

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 $C_A(x)$ der Wert der von dem Algorithmus berechneten Lösung. Welche der folgenden Aussagen treffen in jedem Fall zu? Kreuzen Sie Zutreffendes an.

- $C_A(x) \leq \frac{C_{\text{OPT}}(x)}{2}$
- $C_A(x) > 2 \cdot C_{\text{OPT}}(x)$
- $C_A(x) \leq 2 \cdot C_{\text{OPT}}(x)$
- $C_{\text{OPT}}(x) \leq C_A(x)$

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 $C_B(x)$ der Wert der von dem Algorithmus berechneten Lösung. Welche der folgenden Aussagen treffen in jedem Fall zu? Kreuzen Sie Zutreffendes an.

- $C_B(x) \leq \frac{C_{\text{OPT}}(x)}{3}$
- $C_B(x) \geq 2/3 \cdot C_{\text{OPT}}(x)$
- $C_B(x) \leq 2/3 \cdot C_{\text{OPT}}(x)$
- $C_{\text{OPT}}(x) - C_B(x) \geq 0$

A4: Heuristische Verfahren

Stoff: 14. Heuristiken und Lokale Suche

Wir betrachten das Optimierungsproblem MINIMUM SET COVER, bei dem für eine Menge U von Elementen und eine Menge $\mathcal{S} = \{S_1, \dots, S_m\}$ 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)

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

```
ConstructSetCover( $U, \mathcal{S} = \{S_1, \dots, S_m\}$ ):
     $\mathcal{C} \leftarrow \emptyset$ 
     $W \leftarrow U$ 
    while  $W \neq \emptyset$ 
         $T \leftarrow \emptyset$ 
        for  $i = 1, \dots, m$ 
            if  $|S_i \cap W| > |T \cap W|$  then
                 $T \leftarrow S_i$ 
        if  $T = \emptyset$  then
            return  $\mathcal{C}$ 
         $\mathcal{C} \leftarrow \mathcal{C} \cup \{T\}$ 
         $W \leftarrow W \setminus T$ 
    return  $\mathcal{C}$ 
```

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	$\{\{2, 3, 4\}\}$	$\{1, 5, 6, 7\}$
2	$\{\{2, 3, 4\}, \{5, 6\}\}$	$\{1, 7\}$
3	$\{\{2, 3, 4\}, \{5, 6\}, \{1, 2\}\}$	$\{7\}$
4	$\{\{2, 3, 4\}, \{5, 6\}, \{1, 2\}, \{7\}\}$	\emptyset
5		

Der Ablauf des `ConstructSetCover`-Algorithmus und die Entscheidungen für T basieren darauf, welches Set S_i die größte Anzahl an noch unüberdeckten Elementen ($|S_i \cap W|$) enthält.

Initialisierung:

- $C = \emptyset$ (Die Menge der ausgewählten Sets ist leer)
- $W = U$ (Die Menge der noch zu überdeckenden Elemente ist anfangs das gesamte Universum)

Schleifen-Durchläufe:

1. Durchlauf 1:

- $W = \{1, 2, 3, 4, 5, 6, 7\}$
- $S1 = \{1, 2\} : |S1 \cap W| = 2$
- $S2 = \{2, 3, 4\} : |S2 \cap W| = 3$ (maximal bisher)
- $S3 = \{5, 6\} : |S3 \cap W| = 2$
- $S4 = \{7\} : |S4 \cap W| = 1$
- $S5 = \{4, 5\} : |S5 \cap W| = 2$
- **Entscheidung:** T wird $S2$ gewählt, da $S2$ mit 3 Elementen die meisten unüberdeckten Elemente aus W abdeckt.
- **Update:** $C = \{S2\}, W = \{1, 5, 6, 7\}$

2. Durchlauf 2:

- $W = \{1, 5, 6, 7\}$
- $S1 = \{1, 2\} : |S1 \cap W| = 1$
- $S2 = \{2, 3, 4\} : |S2 \cap W| = 0$ (bereits überdeckt)
- $S3 = \{5, 6\} : |S3 \cap W| = 2$ (maximal bisher)
- $S4 = \{7\} : |S4 \cap W| = 1$
- $S5 = \{4, 5\} : |S5 \cap W| = 1$
- **Entscheidung:** T wird $S3$ gewählt, da $S3$ mit 2 Elementen die meisten unüberdeckten Elemente aus W abdeckt.
- **Update:** $C = \{S2, S3\}, W = \{1, 7\}$

3. Durchlauf 3:

- $W = \{1, 7\}$
- $S1 = \{1, 2\} : |S1 \cap W| = 1$ (maximal bisher)
- $S2 = \{2, 3, 4\} : |S2 \cap W| = 0$
- $S3 = \{5, 6\} : |S3 \cap W| = 0$
- $S4 = \{7\} : |S4 \cap W| = 1$
- $S5 = \{4, 5\} : |S5 \cap W| = 0$
- **Entscheidung:** T wird $S1$ gewählt. Obwohl $S4$ auch 1 Element abdeckt, wird $S1$ gewählt, da es in der Schleife früher betrachtet wird und $|S1 \cap W|$ denselben Maximalwert hat. (Die genaue Auswahl bei Gleichstand hängt von der Implementierung ab, hier nehmen wir die erste gefundene Maximierung.)
- **Update:** $C = \{S2, S3, S1\}, W = \{7\}$

4. Durchlauf 4:

- $W = \{7\}$
- $S1 = \{1, 2\} : |S1 \cap W| = 0$
- $S2 = \{2, 3, 4\} : |S2 \cap W| = 0$
- $S3 = \{5, 6\} : |S3 \cap W| = 0$
- $S4 = \{7\} : |S4 \cap W| = 1$ (maximal bisher)
- $S5 = \{4, 5\} : |S5 \cap W| = 0$
- **Entscheidung:** T wird $S4$ gewählt, da $S4$ das einzige verbleibende unüberdeckte Element abdeckt.
- **Update:** $C = \{S2, S3, S1, S4\}, W = \emptyset$

Schleifenende:

- `w` ist leer, der Algorithmus terminiert.
- Die endgültige Lösung ist $C = \{\{2, 3, 4\}, \{5,$

b)

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)

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

Ausgangsmenge:

$$\mathcal{C} = \mathcal{S} \setminus \{S_6\} = \{S_1, S_2, S_3, S_4, S_5\}$$

Prüfe: $\bigcup \mathcal{C} = \{1, 2, 3, 4, 5, 6\} = U$

Nachbarschaftsstruktur $N(\mathcal{C})$:

1. Operation: Entferne eine Menge S_i aus \mathcal{C} , sofern $\mathcal{C}' = \mathcal{C} \setminus \{S_i\}$ ein Set Cover ist.

Gültige:

- $\mathcal{C}'_1 = \{S_1, S_2, S_4, S_5\}$ (ohne S_3)
- $\mathcal{C}'_2 = \{S_1, S_2, S_3, S_4\}$ (ohne S_5)

2. Operation: Entferne zwei Mengen $S_i, S_j \in \mathcal{C}$, füge $S_\ell \in \mathcal{S} \setminus \mathcal{C} = \{S_6\}$ hinzu. $\mathcal{C}' = (\mathcal{C} \setminus \{S_i, S_j\}) \cup \{S_6\}$

Gültige:

- $\mathcal{C}'_3 = \{S_3, S_4, S_5, S_6\}$
- $\mathcal{C}'_4 = \{S_2, S_4, S_5, S_6\}$
- $\mathcal{C}'_5 = \{S_2, S_3, S_5, S_6\}$
- $\mathcal{C}'_6 = \{S_1, S_4, S_5, S_6\}$
- $\mathcal{C}'_7 = \{S_1, S_3, S_4, S_6\}$

- $C'_8 = \{S_1, S_2, S_4, S_6\}$

Gesamte Nachbarschaft $N(C)$:

$$\begin{aligned} N(C) = & \{\{S_1, S_2, S_4, S_5\}, \\ & \{S_1, S_2, S_3, S_4\}, \\ & \{S_3, S_4, S_5, S_6\}, \\ & \{S_2, S_4, S_5, S_6\}, \\ & \{S_2, S_3, S_5, S_6\}, \\ & \{S_1, S_4, S_5, S_6\}, \\ & \{S_1, S_3, S_4, S_6\}, \\ & \{S_1, S_2, S_4, S_6\}\} \end{aligned}$$

(ii)

- (ii) Wenden Sie lokale Suche mit Nachbarschaftsstruktur N und Ausgangslösung $C = \mathcal{S} \setminus \{S_6\}$ 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.

1. $C_0 = \{S_1, S_2, S_3, S_4, S_5\} \rightarrow |C| = 5$
2. $C_1 = \{S_1, S_2, S_3, S_4\} \rightarrow |C| = 4$
3. $C_2 = \{S_1, S_2, S_4\} \rightarrow |C| = 3$
4. $C_3 = \{S_4, S_6\} \rightarrow |C| = 2 \leftarrow \text{lokales Optimum}$

c)

- 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 $C \subseteq \mathcal{S}$ durch **ConstructSetCover** konstruiert wurde, dann kann C nicht mehr mittels N verbessert werden, es gilt also immer $N(C) = \emptyset$.

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.

$$U = \{1, 2, 3, 4, 5\}$$

$$\begin{aligned} S = & \{ \\ S_1 = & \{1, 2\}, \\ S_2 = & \{2, 3\}, \\ S_3 = & \{3, 4\}, \\ \} \end{aligned}$$

ConstructSetCover gibt: $C = \{S_1, S_3, S_4\}$ (3 Mengen)

Aber: $C' = \{S_1, S_3\} \in N(C)$ ist ebenfalls Set Cover mit 2 Mengen

- ⇒ Die Lösung kann durch Nachbarschaft verbessert werden
- ⇒ Aussage in (c) ist falsch

A5: Dynamisches Programmieren

Stoff: 12. Dynamische Programmierung

a)

- 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

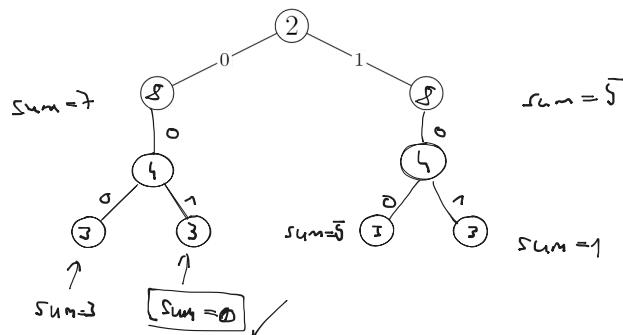
  if  $sum < A[n - 1]$  then
    return isSubsetSum( $A$ ,  $n - 1$ ,  $sum$ )
  else
    included  $\leftarrow$  isSubsetSum( $A$ ,  $n - 1$ ,  $sum - A[n - 1]$ )
    excluded  $\leftarrow$  isSubsetSum( $A$ ,  $n - 1$ ,  $sum$ )
    return (included  $\vee$  excluded)

```

(i)

- (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.

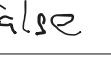


(ii)

- (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:

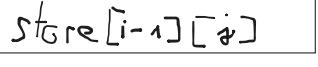
```

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*]

b)

- 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:

	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	7	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.

$$S = \{ \quad \textcircled{2}, \textcircled{4}, \textcircled{5}, \textcircled{6} \quad \}$$

Bin mir nicht sicher, bitte falls falsch ist melden!

Appendix

