ALTE ALGOBAT PRÜFUNGEN

2016 - 2019

TU Wien
Institut für Logic and Computation
Algorithms and Complexity Group

186.866 Algorithmen und Datenstrukturen VU 8.0

2. Test, 2019S

28. Juni 2019

Gruppe A

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

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie dokumentenechte Schreibgeräte (keine Bleistifte!).

Die Verwendung von Taschenrechnern, Mobiltelefonen, 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. Unleserliche Antworten werden nicht gewertet.

	A1	A2	A3	A4	A5	Summe
Erreichbare Punkte:	21	21	20	25	13	100
Erreichte Punkte:						

Viel Erfolg!

Dort wurden C und A vertauscht

Rep 2020

Aufgabe A1: P-NP und Polynomialzeitreduktionen **(21 Punkte)**

a) (12 Punkte) Seien A, B, C Ja/Nein-Probleme und n die Eingabegröße. Nehmen Sie an, es gibt

- eine Reduktion von B nach A in Zeit $O(n^2)$,
- eine Reduktion von A nach SAT in Zeit $O(n^3)$,
- eine Reduktion von SAT nach A in Zeit $O(n)$,
- eine Reduktion von A nach C in Zeit $O(3^{2n})$.

$B \leq_P A \equiv_P SAT$

NP-Reduktion C

(i) Kreuzen Sie in den folgenden Tabellen jeweils die zutreffenden Felder an:
(je korrekter Zeile 1 Punkt, keine Minuspunkte)

C ist ...	Ja	Nein	Keine Aussage möglich
...in NP			X
...NP-schwer			X

A ist ...	Ja	Nein	Keine Aussage möglich
...in NP	X		
...NP-schwer	X		

$A \equiv_P SAT \in NP\text{-}C$

B ist ...	Ja	Nein	Keine Aussage möglich
...in NP	X		
...NP-schwer			X

(ii) Welche der obigen Probleme A, B, C und SAT wären sicher in polynomieller Zeit lösbar, falls P = NP gelten sollte.

A, SAT, B

(iii) Nehmen Sie nun an, C kann für Eingaben der Größe m in Zeit $O(m^2)$ gelöst werden. Welche engste obere Schranke können Sie aus den gegebenen Informationen für die Worst-Case Laufzeit eines optimalen Algorithmus für A schließen?

$A \leq_{NP} C$ in $O(3^{2n})$

C lässt sich in $O(m^2)$ lösen

Obere Schranke für A:

$O((3^{2n})^2) = O(3^{4n}) = O(81^n)$

Red 2026

b) (9 Punkte) Im Folgenden sind verschiedene Probleme mit verschiedenen Zertifikaten gegeben. Welche Zertifikate sind (gemeinsam mit einem geeigneten Zertifizierer) geeignet, um zu zeigen, dass das gegebene Problem in NP ist? Kreuzen Sie Zutreffendes an. Sie dürfen für diese Frage P≠NP annehmen.

Es kann mehr als eine richtige Antwort geben.

(je Unteraufgabe: alles korrekt: 3 Punkte, ein Fehler: 1 Punkt, sonst / kein Kreuz: 0 Punkte)

i) **Problem:** Gegeben eine Menge U und eine Menge $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ von Teilmengen von U. Gibt es ein Set Cover von U mit $\leq k$ Mengen? ∈ NP-C

Zertifikat:

- ☐ ~~Ein leerer String.~~
- ☒ Ein Set Cover mit $\leq k$ Mengen. ✓
- ☐ ~~Eine Zahl m, so dass $m \leq k$ und es ein Set Cover mit m Mengen gibt.~~
- ☐ ~~Keines der Zertifikate ist geeignet.~~

ii) **Problem:** Gegeben sei ein Graph G mit gewichteten Kanten. Gibt es einen Spanning Tree von G mit Gewicht $\leq k$? ∈ P

wir können MSP berechnen

Zertifikat:

- ☒ ~~Ein leerer String.~~
- ☒ Ein Spanning Tree von G mit Gewicht $\leq k$. ✓
- ☐ ~~Alle möglichen Teilmengen von Kanten.~~ nicht polynomiell beschränkt
- ☐ ~~Keines der Zertifikate ist geeignet.~~

iii) **Problem:** Gegeben sei ein Graph G. Hat das *größte* Independent Set von G *genau* k Knoten? ∉ NP-C Optimierungs-Problem!

Zertifikat:

- ☐ ~~Die Größe l des größten Independent Sets.~~
- ☐ ~~Ein Independent Set mit $> k$ Knoten.~~
- ☐ Ein Independent Set mit k Knoten. ✓
- ☒ Keines der Zertifikate ist geeignet.

sagt nicht aus ob es maximal ist oder nicht.

Voraussetzungen:

Zertifikat: Größe muss polynomiell beschränkt sein

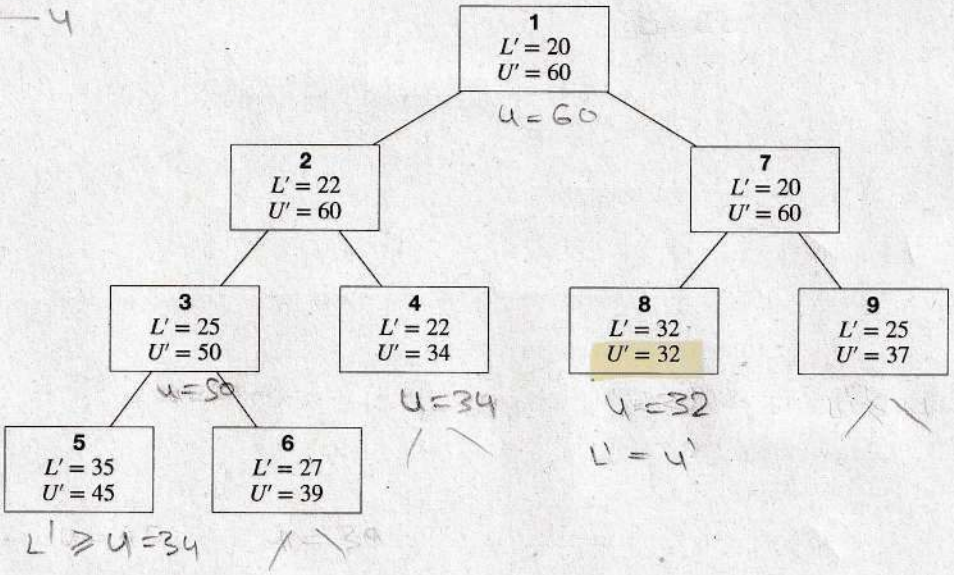
Zertifizierer: Muss in P-Laufzeit, Ja-Instanzen verifizieren können

Rep 2020

Aufgabe A2: Branch-and-Bound und Heuristische Verfahren (21 Punkte)

a) (7 Punkte) Branch-and-Bound wird zur Lösung eines *Minimierungsproblems* verwendet und erzeugt den unten abgebildeten Suchbaum. Jeder Knoten entspricht einer Teilinstanz mit einer lokalen unteren Schranke L' und einer lokalen oberen Schranke U'.

U'
U
L'



Abbruch wenn:
$L' \geq U$
$L' = U'$

Geben Sie die Menge S der Nummern jener Teilinstanzen (repräsentiert durch Blattknoten) an, die *nicht* mehr weiter aufgespalten werden müssen.

S: 5, 8

Was ist die globale obere Schranke U?

U: 32

b) (2 Punkte) Was ist die Idee / das Prinzip der Lokalen Suche?

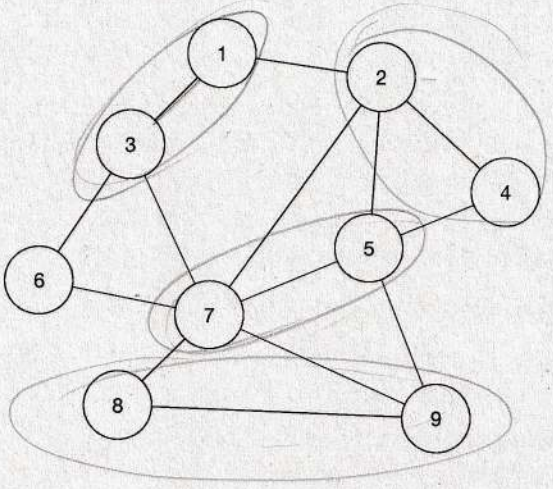
Verbessern einer bereits vorhandenen Lösung durch iteratives Berechnen von Nachbarschafts lösungen

c) (2 Punkte) Nennen Sie einen Unterschied zwischen Heuristischen Verfahren und Approximationsalgorithmen, den Sie aus der Vorlesung kennen.

Kein Gütekriterium bei heuristischen Verfahren

~~Rep 2020~~

d) (7 Punkte) Auf dem folgenden Graphen wird eine obere und eine untere Schranke für die Größe eines kleinsten Vertex Covers berechnet.

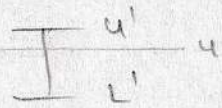


(i) Wenden Sie die Greedy-Heuristik, die jeweils einen Knoten mit höchstem Grad wählt, an und geben Sie die dabei ausgewählte Menge an Knoten an.

Ausgewählte Knoten: 7, 5, 2, 3, 8

(ii) Berechnen Sie ein nicht erweiterbares Matching und zeichnen Sie das Matching im Graphen ein.

(iii) Für die Größe eines kleinsten Vertex Covers ist das eben berechnete nicht erweiterbare Matching ...



☒ ... eine obere Schranke.

greedy → ☐ ... eine untere Schranke.

(korrekt: 1 Punkte, inkorrekt: -1 Punkt, nicht beantwortet: 0 Punkte)

e) (3 Punkte) In welchen Fällen kann die Branch-and-Bound Suche für eine Teilinstanz bei einem Maximierungsproblem abbrechen? Kreuzen Sie Zutreffendes an.

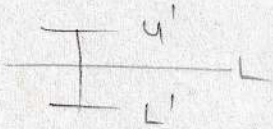
(alles korrekt: 3 Punkte, ein Fehler: 1 Punkt, sonst / kein Kreuz: 0 Punkte)

$L > u'$ ☒ Globale untere Schranke ist größer als lokale obere Schranke.

$L' > L$ ☐ Lokale untere Schranke ist größer als globale untere Schranke:

$L' = L$ ☐ Lokale obere Schranke entspricht lokaler unterer Schranke.

$L' < L$ ☐ Lokale untere Schranke ist kleiner als globale untere Schranke.



Abbruch wenn:

$u' \leq L$

$u' = l'$

Aufgabe A3: NP-Vollständigkeit Spezialfälle **(20 Punkte)**

a) (12 Punkte) *Gewichtetes Vertex Cover auf Bäumen* kann, analog zu gewichtetem Independent Set auf Bäumen, mittels dynamischer Programmierung gelöst werden.

Kreuzen Sie im nachfolgenden Pseudocode jene Codezeilen an, die ausgeführt werden müssen, um eine funktionierende Implementierung von folgendem Algorithmus für gewichtetes Vertex Cover auf Bäumen zu erhalten: der Algorithmus erhält einen Baum $T = (V, E)$, bei dem jedem Knoten $v \in V$ ein positives Gewicht w_v zugewiesen ist und berechnet das minimale Gewicht eines Vertex Covers für T.

Je Block (grau hinterlegte Box) ist genau eine Auswahl korrekt.

(je Block: korrekt: 3 Punkte, falsch / mehrere Kreuze: -1 Punkt, kein Kreuz 0 Punkte. Zumindest 0 Punkte für diese Unteraufgabe)

`Min-Weight-Vertex-Cover-In-A-Tree`(T):
Wähle eine Wurzel r aus
foreach *Knoten u von T* **do**
$A_{\text{out}} \leftarrow \infty$; $A_{\text{in}} \leftarrow \infty$

☐ **foreach** *Knoten u von T in Postorder* **do**

☐ **foreach** *Knoten u von T in Preorder* **do**

☐ **foreach** *Knoten u von T in Inorder* **do**

if *u ist ein Blatt* **then**

☐ $\begin{cases} A_{\text{in}}[u] \leftarrow 0 \\ A_{\text{out}}[u] \leftarrow w_u \end{cases}$

☐ $\begin{cases} A_{\text{in}}[u] \leftarrow 1 \\ A_{\text{out}}[u] \leftarrow 0 \end{cases}$

☐ $\begin{cases} A_{\text{in}}[u] \leftarrow w_u \\ A_{\text{out}}[u] \leftarrow 0 \end{cases}$

else

☐ $\begin{cases} A_{\text{in}}[u] \leftarrow 1 + \sum_{v \in \text{Nachfolger}(u)} A_{\text{in}}[v] \\ A_{\text{out}}[u] \leftarrow w_u + \sum_{v \in \text{Nachfolger}(u)} \min\{A_{\text{in}}[v], A_{\text{out}}[v]\} \end{cases}$

☐ $\begin{cases} A_{\text{in}}[u] \leftarrow w_u + \sum_{v \in \text{Nachfolger}(u)} \min\{A_{\text{in}}[v], A_{\text{out}}[v]\} \\ A_{\text{out}}[u] \leftarrow \sum_{v \in \text{Nachfolger}(u)} A_{\text{in}}[v] \end{cases}$

☐ $\begin{cases} A_{\text{in}}[u] \leftarrow w_u + \sum_{v \in \text{Nachfolger}(u)} \max\{A_{\text{in}}[v], A_{\text{out}}[v]\} \\ A_{\text{out}}[u] \leftarrow \sum_{v \in \text{Nachfolger}(u)} A_{\text{in}}[v] \end{cases}$

☐ **return** $\min\{A_{\text{in}}[r], A_{\text{out}}[r]\}$
☐ **return** $A_{\text{in}}[r] + A_{\text{out}}[r]$
☐ **return** $\max\{A_{\text{in}}[r], A_{\text{out}}[r]\}$

Rep 2020

b) (8 Punkte) Nehmen Sie an, dass gewichtetes Vertex Cover auf Bäumen in quadratischer Zeit gelöst werden kann. $O(n^2)$

Kreuzen Sie an, ob die folgenden (voneinander unabhängigen) Aussagen wahr oder falsch sind.

(je Unteraufgabe: korrekt: 2 Punkte, falsch: -2 Punkte, kein Kreuz: 0 Punkte. Zumindest 0 Punkte für diese Unteraufgabe)

(i) Daraus folgt, dass gewichtetes Vertex Cover auf *Graphen* in quadratischer Zeit gelöst werden kann.

☐ Wahr ☒ Falsch

(ii) Daraus folgt, dass *ungewichtetes* Vertex Cover auf Bäumen in quadratischer Zeit gelöst werden kann.

☒ Wahr ☐ Falsch

(iii) Daraus folgt, dass gewichtetes Vertex Cover auf *Pfaden* in quadratischer Zeit gelöst werden kann.

nicht azyklisch

☐ Wahr ☒ Falsch

(iv) Daraus folgt, dass gewichtetes Vertex Cover auf *Binärbäumen* in quadratischer Zeit gelöst werden kann.

☒ Wahr ☐ Falsch

Aufgabe A4: Dynamisches Programmieren **(25 Punkte)**

a) (9 Punkte) Betrachten Sie das aus der Vorlesung bekannte Rucksackproblem. Gegeben sei eine Instanz mit einer Kapazität $G = 5$ und den folgenden fünf Gegenständen, die jeweils nur einmal vorhanden sind:

	Gegenstand				
	A	B	C	D	E
Wert	1	3	2	4	6
Gewicht	2	1	1	3	2

Vervollständigen Sie die untenstehende Tabelle, sodass der Eintrag in Zeile i und Spalte g gerade dem Wert $OPT(i, g)$ entspricht, der sich mit den ersten i Gegenständen und einem Rucksack der Kapazität g erreichen lässt.

Gegenstände \ Kapazität	0	1	2	3	4	5
$\emptyset$	0	0	0	0	0	0
$\{A\}$	0	0	1			
$\{A, B\}$	0	3				
$\{A, B, C\}$	0					
$\{A, B, C, D\}$	0					
$\{A, B, C, D, E\}$	0					

Was ist der optimale *Wert* einer Lösung für einen Rucksack mit Kapazität $G = 3$?

Wert: ____

Angenommen der Gegenstand E ist nicht mehr verfügbar. Welche Gegenstände entsprechen nun einer optimalen Lösung bei Kapazität $G = 5$?

Gegenstände: ____

Rep 2020

b) (9 Punkte) Sei nun eine weitere Instanz des Rucksackproblems gegeben mit Kapazität $G = 6$ und den folgenden sechs Gegenständen.

	Gegenstand					
	A	B	C	D	E	F
Wert	2	5	3	1	6	9
Gewicht	2	1	1	2	4	3

Die Wertetabelle M nach Ausführung des Dynamischen Programms lautet wie folgt:

Gegenstände \ Kapazität	0	1	2	3	4	5	6
$\emptyset$	0	0	0	0	0	0	0
$\{A\}$	0	0	2	2	2	2	2
$\{A, B\}$	0	5	5	7	7	7	7
$\{A, B, C\}$	0	5	8	8	10	10	10
$\{A, B, C, D\}$	0	5	8	8	10	10	11
$\{A, B, C, D, E\}$	0	5	8	8	10	11	14
$\{A, B, C, D, E, F\}$	0	5	8	9	14	17	17

(i) Markieren Sie in der Tabelle all jene Felder, 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.

$S = \{$ F, C, B $\}$

F + C + B = 9 + 3 + 5 = 17 ✓

c) (7 Punkte) Gegeben ist ein gerichteter Graph $G = (V, E)$ mit $|V| = n$ Knoten. Die Kanten von G sind mit Kantengewichten $c_{vw} \in \mathbb{R}$ für alle Kanten $(v, w) \in E$ gewichtet, sodass es keine negative Kreise gibt.

(i) Ergänzen Sie Bellmans Rekursionsgleichungen für die Länge $OPT(i, v)$ eines kürzesten v-t Pfades in G mit höchstens i Kanten, wobei $v \in V$ ein beliebiger Knoten ist und $t \in V$ ein fester Zielknoten.

$OPT(0, t) =$ [ ]

$OPT(0, v) =$ [ ] für $v \neq t$

$OPT(i, v) =$ [ ] für $i \geq 1$

(ii) Geben Sie den kleinsten Wert für die Kantenanzahl i an, sodass für jeden Graphen G, der die gegebenen Voraussetzungen erfüllt, Folgendes gilt:

$$OPT(i, v) = OPT(i + 1, v) \text{ für alle Knoten } v \in V.$$

Der kleinste Wert ist $i =$ [ ].

(iii) Geben Sie für Ihre Wahl der kleinsten Kantenanzahl i (aus Unterpunkt (ii)) eine kurze Begründung in 1–2 Sätzen an.

Rep 2020

$\frac{C_A(x)}{C_{OPT}(x)} \leq \varepsilon$

Aufgabe A5: Approximationsalgorithmen **(13 Punkte)**

a) (6 Punkte) Angenommen A ist ein ε-Approximationsalgorithmus für ein Minimierungsproblem (d.h. $\varepsilon \geq 1$). Sei x eine Probleminstanz mit optimalem Lösungswert $C_{OPT}(x)$ und $C_A(x)$ der Wert der von A berechneten Lösung. Welche der folgenden Aussagen treffen in jedem Fall zu? Kreuzen Sie Zutreffendes an.

- [x] $C_{OPT}(x) \leq C_A(x)$
- [ ] $C_A(x) > C_{OPT}(x)$
- [x] $C_A(x) \leq \varepsilon \cdot C_{OPT}(x)$
- [ ] $C_A \leq \frac{C_{OPT}(x)}{\varepsilon}$

Sei B ein Approximationsalgorithmus für das gleiche Problem mit Gütegarantie $\varepsilon/2$ und $C_B(x)$ der von B berechnete Lösungswert für die Instanz x. Welche der folgenden Aussagen sind in jedem Fall korrekt? Kreuzen Sie Zutreffendes an.

$\frac{C_B(x)}{C_{OPT}(x)} \leq \frac{1}{2}\varepsilon$

- [ ] $\frac{C_A(x)}{C_B(x)} \geq 2$
- [ ] $\frac{C_B(x)}{C_A(x)} = \frac{1}{2}$

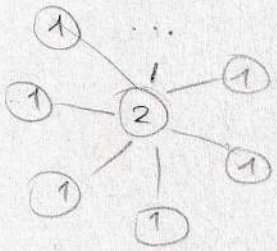
$C_A \leq \varepsilon C_{OPT}$
$C_B \leq \frac{1}{2}\varepsilon C_{OPT}$
$C_A + C_B \leq \frac{3}{2}\varepsilon C_{OPT}$

- [x] $C_B(x) \leq \varepsilon \cdot C_{OPT}(x)$
- [x] $C_A(x) + C_B(x) \leq 3\varepsilon/2 \cdot C_{OPT}(x)$

(je Block: alles korrekt: 3 Punkte, ein Fehler: 1 Punkte, sonst / kein Kreuz: 0 Punkte)

b) (7 Punkte) Beim Gewichteten Independent Set Problem wird in einem Eingabegraphen nach einem Independent Set mit maximalem Gewicht gesucht (jedem Knoten ist ein Gewicht zugeordnet, das Gewicht eines Independent Set ist die Summe der Gewichte der darin enthaltenen Knoten). Zeigen Sie, dass ein Greedy-Algorithmus für Gewichtetes Independent Set, der wiederholt einen Knoten mit maximalem Gewicht zu einem bestehenden Independent Set hinzufügt, jede konstante Gütegarantie $\varepsilon \leq 1$ verletzt.

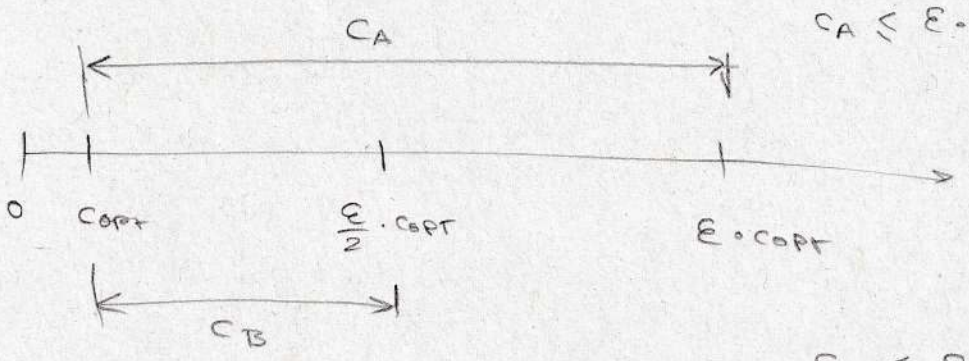
Greedy unterscheidet nicht zwischen OPT_{IN} und OPT_{OUT}.
Daraus folgt:



$\frac{C_A(x)}{C_{OPT}(x)} = \frac{1}{n} \leq 1$ ✓

wobei $n \geq 2$

Greedy → 2
OPT → n-1



$C_A \leq \varepsilon \cdot C_{OPT}$

$C_B \leq \frac{\varepsilon}{2} \cdot C_{OPT}$

Weiters wissen wir:

$C_A \leq \varepsilon \cdot C_{OPT}$

$2 \cdot C_B \leq \varepsilon C_{OPT}$

$C_A + 2C_B \leq 2\varepsilon C_{OPT}$

$C_A \leq \varepsilon C_{OPT}$

$C_B \leq \frac{1}{2}\varepsilon C_{OPT}$

$C_A + C_B \leq \varepsilon \cdot C_{OPT} + \frac{1}{2}\varepsilon C_{OPT}$

$C_A + C_B \leq \frac{3}{2}\varepsilon C_{OPT}$

Daraus folgt

$C_B \leq \frac{1}{2} C_A$

$\frac{C_B}{C_A} \leq \frac{1}{2}$

$\frac{C_A}{C_B} \leq 2$

TU Wien
Institut für Logic and Computation
Algorithms and Complexity Group

186.866 Algorithmen und Datenstrukturen VU 8.0

1. Test, 2019S

14. Mai 2019

Gruppe A

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

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie dokumentenechte Schreibgeräte (keine Bleistifte!).

Die Verwendung von Taschenrechnern, Mobiltelefonen, 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. Unleserliche Antworten werden nicht gewertet.

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

Viel Erfolg!

Rep 2020

Aufgabe A1: Algorithmenanalyse **(22 Punkte)**

a) (12 Punkte) Tragen Sie für die Codestücke `FunktionA` und `FunktionB` die Laufzeit und den Rückgabewert (z) in Abhängigkeit von n, jeweils in Θ-Notation in die nachfolgende Tabelle ein.

```
FunktionA(n):
    x ← n^2
    z ← 1
    while x > 1
        x ← x/3
        z ← 3z
    return z
```

```
FunktionB(n):
    j ← 0
    z ← 1
    for i ← 1 bis 3n
        if i mod 3 = 0 then
            j ← j + i
    j ← (2/3)j − n
    while j > 0
        z ← nz
        j ← j − 1
    return z
```

	FunktionA	FunktionB
Laufzeit	$\Theta(\log_3 n)$	$\Theta(n^2)$
Rückgabewert (z)	$\Theta(n^2)$	$\Theta(n^{2n})$

Aufgabe A-1)

Funktion A(n)

$x = n^2$
$z = 1$
} $\Theta(1)$

while $x > 1$

$x = \frac{x}{3}$

$z = 3z$

return z

$\rightarrow$ Abbruchsbedingung: $x \leq 1$

Iterationen

$x \cdot \frac{1}{3}^k \leq 1$

$n^2 \cdot \frac{1}{3}^k \leq 1$

$n^2 \cdot \frac{1}{3^k} \leq 1$

$n^2 \leq 3^k$

$\log_3(n^2) \leq \log_3(3^k)$

$2 \log_3(n) \leq k \underbrace{\log_3(3)}_{1}$

$2 \log_3(n) \leq k$

$z = \underbrace{z}_{1} \cdot 3^{2 \log_3(n^2)}$ (x)

$z = 3^{2 \cdot 2 \log_3(n)}$

$z = 81^{\log_3(n)}$

Rückgabe:

$\Theta(81^{\log_3 n})$

Laufzeit:

$\Theta(1 + 2 \log_3(n)) = \Theta(\log_3(n))$

Alternativer Rechenweg:

$z = 1 \cdot 3^{\log_3(n^2)}$

$z = n^2$

Bei Exponenten sind konstante Faktoren relevant, es ist wichtig, dass man $3^{\log_3(n^2)}$ schreibt und nicht $3^{\log(n^2)}$!

Funktion B(n):

```
j = 0
z = 1
for i = 1 bis 3n
    if i mod 3 = 0
        j = j + i
```

$O(n)$

```
j = (2/3) j - n
```

```
while j > 0
    z = n z
    j = j - 1
return z
```

$O(n^2)$

Laufzeit $O(n^2)$

Wie oft teilt 3 die Zahl $3n$? → n mal

$j = 0$

$j = 0 + 3$

$j = 0 + 3 + 6$

$j = 0 + 3 + 6 + 9 \ldots \longrightarrow \sum_{i=1}^{n} 3i = 3 \sum_{i=1}^{n} i$

$= 3 \frac{n(n+2)}{2}$

$\frac{2}{3} \frac{3}{2} (n^2 + n) - n = n^2$

Abbruch wenn $j \leq 0$

genau n^2 Iterationen

$z = z \cdot n^{(n^2)}$

(z mit 1 unterklammert)

$z = n^{2n}$; Rückgabewert

$$2 \cdot \sqrt{\frac{n+1}{4}} \leq c_n \qquad c = \frac{1}{2}$$

$$2 \cdot \sqrt{\frac{n+1}{4}} \leq \frac{1}{2} n \quad | \cdot 2$$

$$4 \cdot \sqrt{\frac{n+1}{4}} \leq n \quad \rightarrow \quad 4 \sqrt{(n+1) \cdot \frac{1}{4}} \leq n \quad = \frac{4}{2} \sqrt{n+1} \leq n$$

$$2 \cdot \sqrt{n+1} \leq n \quad |^2 \qquad = 2 \sqrt{n+1} \leq n$$

$$4(n+1) \leq n^2$$

$$4n + 4 \leq n^2$$

$$0 \leq n^2 - 4n - 4 \longrightarrow \text{Quadratische Gleichung}$$

$a = 1$
$b = -4$
$c = -4$

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \rightarrow \frac{4 \pm \sqrt{16 - 4 \cdot 1 \cdot (-4)}}{2} =$$

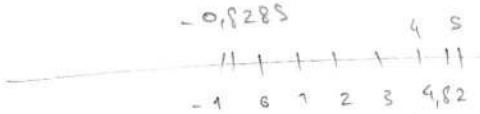
$$= \frac{4 \pm \sqrt{16+16}}{2} =$$

$$= \frac{4 \pm \sqrt{32}}{2} =$$

$$= \frac{4 \pm 5{,}657}{2} =$$

$x_1 = 4{,}8285$

$x_2 = -0{,}8285$



daraus folgt:

$u_6 = 5$

Da Funktion eine Parabel hat die nach oben geöffnet!

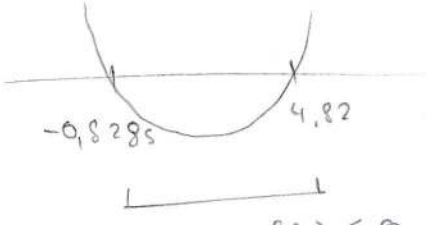
Probe:

$0 \leq 25 - 20 - 4 = 1$

Probe:

$2 \cdot \sqrt{\frac{6}{4}} \leq \frac{1}{2} \cdot 5$

$4{,}89 \leq 5$ ✓



Dadurch, dass ein c und ein n_0 existieren, die diese Beziehung erlauben,

$$2 \cdot \sqrt{\frac{n+1}{4}} \leq \frac{1}{2} \cdot n \quad \longrightarrow \quad \left| \frac{a_n}{b_n} \right| \leq c$$

a_n c b_n

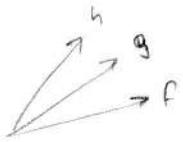
wenn $n_0 \rightarrow \leq n$

$$\downarrow$$

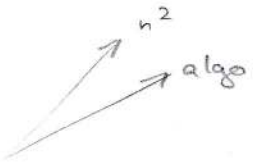
$$a_n \in \mathcal{O}(b_n)$$

c)

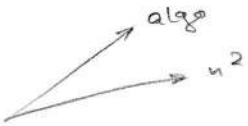
i) $f \in \mathcal{O}(g)$, $g \in \Omega(h)$ $\Rightarrow$ $f \in \Theta(h)$ falsche Schlussfolgerung



ii) Wenn worst case eines Algorithmus



iii)



Rep 2020

b) (4 Punkte) Sei $c = \frac{1}{2}$. Finden Sie das kleinste $n_0 \in \mathbb{N}^+$, sodass für alle $n \geq n_0$ gilt

$$2 \cdot \sqrt{\frac{n+1}{4}} \leq cn.$$

Kleinstes $n_0 =$ 5

Was haben Sie somit gezeigt? Kreuzen Sie Zutreffendes an:

- ☐ $2 \cdot \sqrt{\frac{n+1}{4}}$ ist in $\Omega(n)$
- ☒ $2 \cdot \sqrt{\frac{n+1}{4}}$ ist in $O(n)$
- ☐ $2 \cdot \sqrt{\frac{n+1}{4}}$ ist in $\Theta(n)$
- ☐ keine der zuvor genannten Aussagen

$$2 \cdot \sqrt{\frac{n+1}{4}} \leq \frac{1}{2} n$$

$$4 \cdot \sqrt{(n+1) \cdot \frac{1}{4}}^2 \leq n$$

$$\frac{4}{2} \sqrt{n+1} \leq n$$

$$2 \sqrt{n+1} \leq n$$

c) (6 Punkte) Kreuzen Sie bei folgenden Aussagen an, ob diese wahr oder falsch sind.
(korrektes Kreuz: +2 Punkte, inkorrektes Kreuz: -2 Punkte, kein Kreuz: 0 Punkte. Minimum für diese Unteraufgabe: 0 Punkte)

(i) Wenn $f = O(g)$ und $g = \Omega(h)$ gilt, dann gilt auch $f = \Theta(h)$.
☐ Wahr ☒ Falsch

(ii) Wenn die Worst-Case Laufzeit eines Algorithmus über alle Eingaben der Eingabegröße n in $O(n^2)$ liegt, so kann seine asymptotische Laufzeit nicht in $\Omega(n^3)$ liegen.
☒ Wahr ☐ Falsch

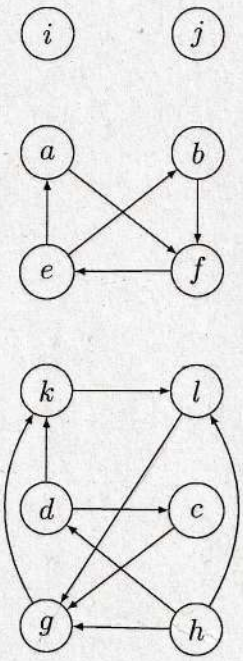
(iii) Wenn die Worst-Case Laufzeit eines Algorithmus über alle Eingaben der Eingabegröße n in $\Omega(n^2)$ liegt, so kann seine asymptotische Laufzeit nicht in $\Omega(n^3)$ liegen.
☐ Wahr ☒ Falsch

Rep 2020

Aufgabe A2: Graphen **(18 Punkte)**

a) (5 Punkte) Gegeben ist der nachfolgende Graph $H = (V_H, E_H)$ mit 12 Knoten. Verwenden Sie die aus der Vorlesung bekannten Definitionen und kreuzen Sie Zutreffendes in der untenstehenden Tabelle an. Betrachten Sie für jede Zeile der Tabelle den gerichteten Teilgraphen $G = (V_G, E_G)$ von H mit der in der Tabelle gegebenen Knotenmenge V_G und $E_G = \{(a, b) \mid a, b \in V_G, (a, b) \in E_H\}$.

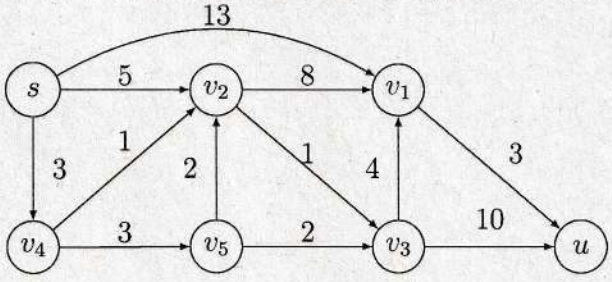
(korrekte Zeile: +1 Punkt, inkorrekte Zeile: -1 Punkt, keine Antwort (Zeile ohne Kreuz): 0 Punkte. Minimum für diese Unteraufgabe: 0 Punkte)



V_G	G ist schwach zusammenhängend	G ist schwache Zusammenhangskomp. von H	G ist stark zusammenhängend	G ist starke Zusammenhangskomp. von H	keine der zuvor genannten Aussagen trifft zu
$\{c, l\}$	☐	☐	☐	☐	☒
$\{a, e, f\}$	☒	☐	☒	☐	☐
$\{c, d, g, h, k, l\}$	☒	☒	☐	☐	☐
$\{g, k, l\}$	☒	☐	☒	☒	☐
$\{i\}$	☒	☒	☒	☒	☐

✓

b) (10 Punkte) Wenden Sie den aus der Vorlesung bekannten *Algorithmus von Dijkstra in der Implementierung mit einer Liste*, zum Finden eines kürzesten Pfades von s nach u auf dem gegebenen Graphen an. Dokumentieren Sie für jeden Schritt, bei dem sich die Menge Discovered ändert (diese enthält die bereits untersuchten Knoten), für jeden Knoten $d(\cdot)$ und die Mengen Discovered und L. Extrahieren Sie anschließend aus dem Ablauf einen kürzesten Pfad, sowie seine Länge.



1. Discovered = {
   L = {

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$							

2. Discovered = {
   L = {

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$							

3. Discovered = {
   L = {

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$							

4. Discovered = {
   L = {

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$							

5. Discovered = {
   L = {

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$							

6. Discovered = {
   L = {

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$							

7. Discovered = {
   L = {

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$							

8. Discovered = {
   L = {

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$							

Pfad:

Gewicht:

c) (3 Punkte) Zeichnen Sie einen Min-Heap zur Menge $\{12, 9, 2, 45, 5\}$.

Aufgabe A3: Greedy **(20 Punkte)**

Im Folgenden sind einige Fragen zu dem Themenbereich Greedy-Algorithmen gegeben. Beantworten Sie jeweils die gestellte Ja/Nein-Frage. Für jede Antwort ist zusätzlich eine (informelle) Begründung bzw. ein konkretes Gegenbeispiel notwendig.

a) Gegeben ist eine Währung deren Münzen eine Stückelung von $1, 3, 6$ und 13 haben. Liefert der Greedy-Algorithmus zum Geldwechseln aus der Vorlesung bei dieser Stückelung immer ein optimales Ergebnis?

☐ Ja ☐ Nein

Begründung/Gegenbeispiel:

b) Sei G ein gewichter Graph, bei dem alle Kanten unterschiedliche Gewichte haben. Wenn Sie den Algorithmus von Kruskal und den Algorithmus von Prim verwenden um einen Minimum Spanning Tree (MST) von G zu berechnen, finden dann beide Algorithmen immer denselben MST?

☐ Ja ☐ Nein

Begründung/Gegenbeispiel:

c) Sei T ein Spannbaum in einem gewichteten Graphen G. Wir nennen die Kanten in T mit dem höchsten Gewicht die Bottleneck-Kanten von T. Wir sagen T ist ein Minimum Bottleneck Spanning Tree (MBST) falls es keinen Spanning Tree von G gibt, dessen Bottleneck-Kanten ein geringeres Gewicht als die Bottleneck-Kanten von T haben. Ist jeder MBST auch ein minimaler Spannbaum?

☐ Ja ☐ Nein

Begründung/Gegenbeispiel:

d) Sei G ein Graph, bei dem jedem Knoten v eine reele Zahl $w(v)$ als Gewicht zugeordnet ist. Für eine Menge von Knoten S schreiben wir $w(S)$ für die Summe $\Sigma_{v \in S} w(v)$ der Gewichte der Knoten in S. Eine *überdeckende Knotenmenge* von G ist eine Menge S von Knoten, so dass jede Kante von G zu einem Knoten in S inzident ist. Eine überdeckende Knotenmenge S heißt minimal, wenn für jede andere überdeckende Knotenmenge S^* stets $w(S) \leq w(S^*)$ gilt. Gegeben ist der folgende Greedy-Algorithmus:

```
Greedy(n):
    Input: Graph G = (V, E) mit gewichteten Knoten
    S ← ∅
    S_E ← ∅
    while S_E ≠ E
        Wähle einen Knoten v ∈ V \ S mit minimalem Gewicht.
        Füge v zu S hinzu.
        Füge alle Kanten die indizent zu v sind zu S_E hinzu.
```

Findet dieser Algorithmus immer eine minimale überdeckende Knotenmenge?

☐ Ja ☐ Nein

Begründung/Gegenbeispiel:

Rep 2020

Aufgabe A4: Hashing und Bäume **(20 Punkte)**

a) (12 Punkte) Im Folgenden geht es um die Best-Case- und Worst-Case-Laufzeiten unterschiedlicher Operationen für Hashing mit Verkettung von Überläufern, offenes Hashing und bei binären Suchbäumen in Abhängigkeit der n bereits eingefügten Elemente.

(korrekte Zeile: +1 Punkt, inkorrekte Zeile: -1 Punkt, leere bzw. unvollständige Zeile: 0 Punkte. Minimum für diese Unteraufgabe: 0 Punkte)

(i) AVL-Bäume

Operation	Best-Case	Worst-Case
Erfolgreiches Einfügen	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$
Erfolgreiches Suchen	$\Theta(1)$	$\Theta(\log_2 n)$
Erfolgloses Suchen	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$

(ii) Offenes Hashing mit linearem Sondieren

Operation	Best-Case	Worst-Case
Erfolgreiches Einfügen	$\Theta(1)$	$\Theta(n)$
Erfolgreiches Suchen	$\Theta(1)$	$\Theta(n)$
Erfolgloses Suchen	$\Theta(1)$	$\Theta(n)$

als „frei" markiert

(iii) Hashing mit Verkettung der Überläufer

Operation	Best-Case	Worst-Case
Erfolgreiches Einfügen	$\Theta(1)$	$\Theta(1)$
Erfolgreiches Suchen	$\Theta(1)$	$\Theta(n)$
Erfolgloses Suchen	$\Theta(1)$	$\Theta(n)$

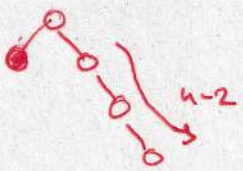
oben einfügen

längst mögliche Kette

(iv) Binäre Suchbäume ohne Höhenbalancierung

Operation	Best-Case	Worst-Case
Erfolgreiches Einfügen	$\Theta(1)$	$\Theta(n)$
Erfolgreiches Suchen	$\Theta(1)$	$\Theta(n)$
Erfolgloses Suchen	$\Theta(1)$	$\Theta(n)$

gleich bei Wurzel

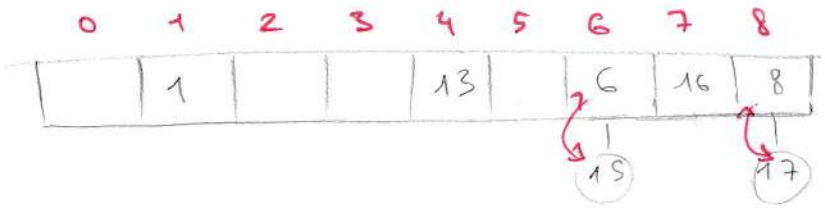


Aufgabe A4) b)

Zahlen in vorgegebener Reihenfolge hinzufügen: [16, 8, 17, 6, 1, 13, 15]

$m = 9$

Verkettung der Überläufer mit $h(k) = k \bmod 9$



Ø Anzahl der Vergleiche für erfolgreiche Suche:

Elemente in Tabelle: 7

16	8	17	6	1	13	15	
1	2	~~2~~ 1	2	1	1	~~4~~ 1	→ Σ = 9

Durchschnitt: $\frac{9}{7}$

$m = 9$

$h_1(k) = k \bmod m$

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

[16, 8, 17, 6, 1, 13, 15]

→ Double-Hashing:

$h(i,k) = (h_1(k) + i \cdot h_2(k)) \bmod m$

$h(0,16) = 7$

$h(0,8) = 8$

$h(0,17) = 8$ → besetzt, $h(1,17) = 8 + 1 + 2 = 11 \equiv 2$

$h(0,6) = 6$

$h(0,1) = 1$

$h(0,13) = 4$

$h(0,15) = 6$ → besetzt, $h(1,15) = 6 + 1 = 7$

0	1	2	3	4	5	6	7	8
15	1	17	~~13~~	13		6	16	8

besetzt

$h(2,15) = 6 + 2 \cdot 1 = 8$

$h(3,15) = 6 + 3 = 9 \equiv 0$

Ø Anzahl von Vergleichen für erfolgreiche Suche:

Elemente: 6 7

$\sum$ Vergleiche: $1 + 1 + 2 + 1 + 1 + 1 + 4 = 11$

$\frac{11}{7}$

Dez 2020

b) (8 Punkte) Gegeben sind folgende natürliche Zahlen:

$$[16, 8, 17, 6, 1, 13, 15]$$

Fügen Sie diese in der vorgegebenen Reihenfolge jeweils in folgende Varianten von anfangs leeren Hashtabellen der Größe $m = 9$ ein. Berechnen Sie dann die durchschnittliche Anzahl von Schlüsselvergleichen bei einer erfolgreichen Suche eines Elements, wobei jedes dieser Elemente mit gleicher Wahrscheinlichkeit gesucht wird. Für das Sondieren gilt $i = 0, 1, \ldots, m - 1$.

(je korrekt befüllter Hashtabelle: 2 Punkte, je korr. Anzahl Vergleiche: 2 Punkte)

(i) Verkettung der Überläufer mit $h(k) = k \bmod m$.

Durchschnittliche Anzahl von Vergleichen bei erfolgreicher Suche: $T =$ 9/7

Hashtabelle:

0	1	2	3	4	5	6	7	8
	1		·	13		15→6	16	17→8

wichtig!
15 wird an den Anfang der Liste angehängt

(ii) Double-Hashing mit $h(k) = (h_1(k) + ih_2(k)) \bmod m$, $h_1(k) = k \bmod m$ und $h_2(k) = 1 + (k \bmod 5)$.

Durchschnittliche Anzahl von Vergleichen bei erfolgreicher Suche: $T =$ 11/7

Hashtabelle:

0	1	2	3	4	5	6	7	8
15	1	17		13		6	16	8

Rep 2020

Aufgabe A5: Sortieralgorithmen **(20 Punkte)**

a) (4 Punkte) Können die folgenden Arrays als Run in einem Ablauf eines Timsort Algorithmus mit $Min\_Run = 6$ vorkommen? Kreuzen Sie Zutreffendes an.

		Ja	Nein	
✓	[2, 2, 2, 2, 2, 2, 2]	☒	☐	
✗	[2, 4, 7, 6, 8, 11, 14]	☐	☒	← hat 7 Elemente
✗	[7, 7, 6, 6, 5, 5, 4]	☐	☒	
✓	[2, 6, 9, 11, 18, 15]	☒	☐	✓

wir nehmen aber 6 Elemente →

(korrektes Kreuz: +1 Punkt, inkorrektes Kreuz: -1 Punkt, kein Kreuz: 0 Punkte. Minimum für diese Unteraufgabe: 0 Punkte)

b) (16 Punkte) Dual Pivot Quicksort ist eine Variante von Quicksort, die in der Praxis effizienter ist als ein normaler Quicksort. Dual Pivot Quicksort funktioniert im Wesentlichen wie Quicksort, mit den folgenden Änderungen:

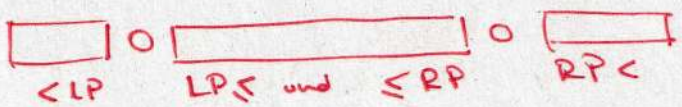
left pivot LP
right pivot RP

Bei Dual Pivot Quicksort werden zwei Elemente als Pivot gewählt. Wir nennen das kleinere Pivot Element linkes Pivot (LP) und das größere Pivot Element rechtes Pivot (RP). Das Array wird dann in drei Teile partitioniert: Die Elemente die kleiner sind als LP, die Elemente die größer oder gleich LP aber kleiner oder gleich RP sind, und die Elemente die größer als RP sind. Danach wird Dual Pivot Quicksort rekursiv auf jede der drei Partitionen aufgerufen.

Kreuzen Sie im Pseudocode auf der nächsten Seite die Codezeilen an, die ausgeführt werden müssen, um eine funktionierende Implementierung von Dual Pivot Quicksort zu erhalten. Je Block (grau hinterlegte Box) ist genau eine Auswahl korrekt.

(richtiges Kreuz +4 Punkte, falsches Kreuz/mehrere Kreuze im Block -4 Punkte, kein Kreuz 0 Punkte. Minimum für diese Unteraufgabe: 0 Punkte)

Partitionierung:



Rep 2020

DualPivotQuicksort(A,left,right):

// Array A, wird von Element left bis Element right sortiert

if $right - left \geq 1$ **then**

// Erstes (=left) und letztes (=right) Element als Pivot

$p \leftarrow \min\{A[\text{left}], A[\text{right}]\}$ // p ist das kleinere Pivotelement

$q \leftarrow \max\{A[\text{left}], A[\text{right}]\}$ // q ist das größere Pivotelement

$l \leftarrow \text{left} + 1$ // l begrenzt die linke Partition

$g \leftarrow \text{right} - 1$ // g begrenzt die rechte Partition

$k \leftarrow l$ // k ist die Laufvariable der Partitionierung

☒ **while** $k \leq g$

☐ **while** $l \leq g$

☐ **while** $k \leq q$

if $A[k] < p$ **then**

Tausche $A[k]$ und $A[l]$

$l \leftarrow l + 1$

else if $A[k] > q$ **then**

while $A[g] > q$ *und* $k < g$

☒ $g \leftarrow g - 1$

☐ $k \leftarrow k + 1$

☐ $l \leftarrow l + 1$

Tausche $A[k]$ und $A[g]$

$g \leftarrow g - 1$

if $A[k] < p$ **then**

Tausche $A[k]$ und $A[l]$

$l \leftarrow l + 1$

else

☐ $g \leftarrow g - 1$

☒ $k \leftarrow k + 1$

☐ $l \leftarrow l + 1$

$l \leftarrow l - 1$

$g \leftarrow g + 1$

$A[\text{left}] \leftarrow A[l]$

$A[l] \leftarrow p$ // p wird an die finale Position gesetzt

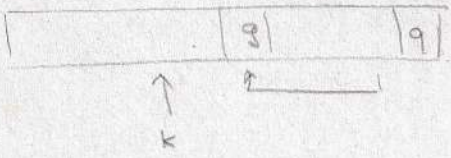
$A[\text{right}] \leftarrow A[g]$

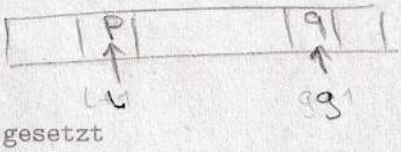
$A[g] \leftarrow q$ // q wird an die finale Position gesetzt

☐ { **call** DualPivotQuicksort(A, left $+ 1$, l)
call DualPivotQuicksort(A, $l + 1$, g)
call DualPivotQuicksort(A, $g + 1$, right $- 1$) }

☐ { **call** DualPivotQuicksort(A, left, $p - 1$)
call DualPivotQuicksort(A, $p + 1$, $q - 1$)
call DualPivotQuicksort(A, $q + 1$, right) }

☒ { **call** DualPivotQuicksort(A, left, $l - 1$)
call DualPivotQuicksort(A, $l + 1$, $g - 1$)
call DualPivotQuicksort(A, $g + 1$, right) }





p = LP
q = RP
A[k] < p
A[k] > q
p ≤ A[k] ≤ q
Pivot an richtige Stelle gesetzt
left
l
alle Werte < p
swap: l-1 und left
Setze p auf l
g
right
alle Werte > q
swap: g+1 und right
Setze q auf r
p und q sind values und keine Indexes

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!

Rep 1 - 19

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

Quadratisches Sondieren

$h''(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$

$h''(k,i) = (h'(k) + 0{,}5i + 1{,}5i^2) \bmod m$

(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?

Aufgabe A4: Hashing

a) $m = 7$
$c_1 = 0{,}5$
$c_2 = 1{,}5$

$h'(11,0) = (4 + 0 + 0) \bmod 7 = 4$ (besetzt)

$h'(11,1) = (4 + 0{,}5 + 1{,}5) \bmod 7 = 6$ (besetzt)

$h'(11,2) = (4 + 0{,}5 \cdot 2 + 1{,}5 \cdot 4) \bmod 7 =$

$(4 + 1 + 6) \bmod 7 = 4$ (besetzt)

$h'(11,3) = (4 + 0{,}5 \cdot 3 + 1{,}5 \cdot 3^2) \bmod 7 =$

$(4 + 1{,}5 + 13{,}5) \bmod 7 = 5$ ✓ frei

b) Double Hashing mit
Verbesserung von Brent

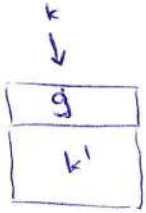
$h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m \qquad m = 7$

$h_1(k) = k \bmod 7$

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

Verbesserung von Brent

Wir wollen k an Stelle g einfügen aber es ist mit k' besetzt



So lange k nicht platziert wurde:

$j = (j + h_2(k)) \bmod m$

$j' = (j + h_2(k')) \bmod m$

wie $h(k,1)$ mit $h_1(k) = j$

wenn (j frei ∨ j' besetzt)
probiere k auf j zu setzen ($g = j$)

wenn (j besetzt ∧ j' frei)
Setze k auf g
Setze k' auf j'

Anwendung

$h(k,0) = h(16,0) = 2$ (besetzt)

$j = (2 + h_2(16)) \bmod m = (2+2) \bmod 7 = 4$ (besetzt)

$j' = (2 + h_2(23)) \bmod m = (2+4) \bmod 7 = 6$ (besetzt)

$g = 4$

$j = (4 + h_2(16)) \bmod m = (4+2) \bmod 7 = 6$ (besetzt)

$j' = (4 + h_2(18)) \bmod m = (4+4) \bmod 7 = 1$ (frei)

$k \to g$ (4)
$k' \to j'$ (1)

16 auf ~~15~~ 4, 18 auf 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	

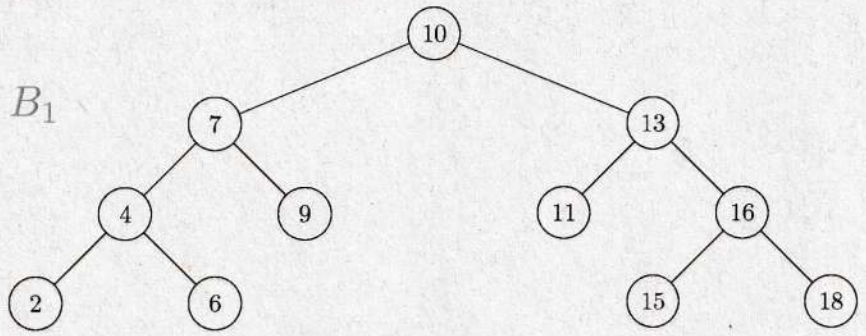
Welche dieser Eingabefolgen führen beim Einfügen in eine anfangs leere Tabelle zu diesem Ergebnis?

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

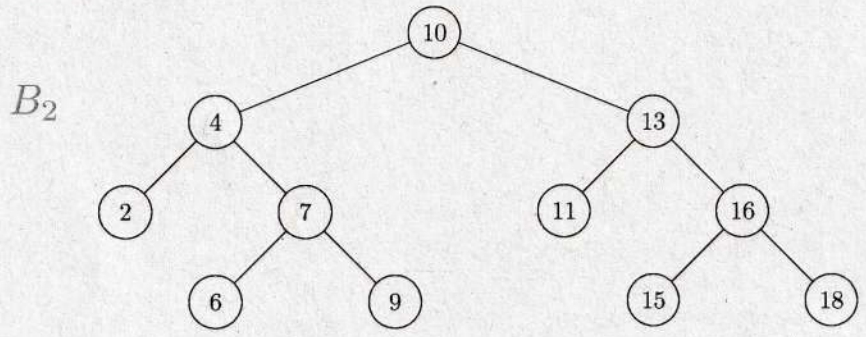
(2 Punkte, wenn alle Kreuze korrekt sind, 1 Punkt bei genau einem Fehler, ansonsten 0 Punkte)

Aufgabe A2: Suchbäume **(10 Punkte)**

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



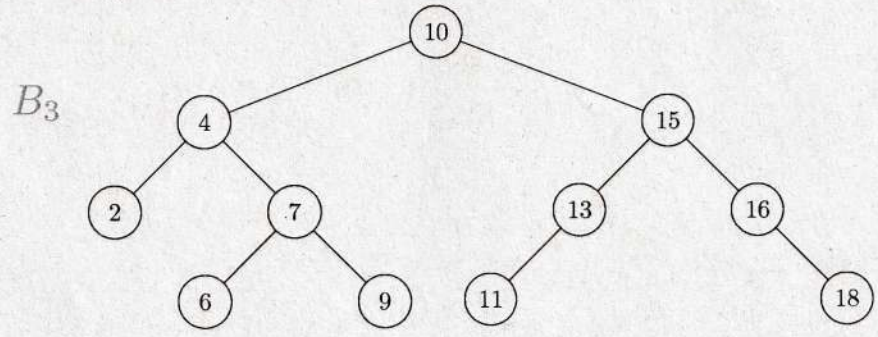
Es wird nun eine ganze Zahl $x \in \{0, \ldots, 20\}$, die noch nicht in B_1 enthalten ist, in B_1 eingefügt. Danach wird B_1 rebalanciert und x wird wieder gelöscht. Dadurch entsteht der folgende AVL-Baum B_2:



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.

Rep 2,19

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

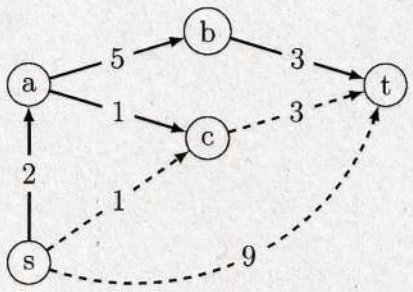
Wir definieren folgendes EXTENDED SHORTEST PATH Problem: Gegeben ist ein gerichteter Graph $G = (V, E)$ mit einem positiven Kantengewicht c_{uv} für jede Kante $(u, v) \in E$ und zwei speziellen Knoten s und t. Die Kantenmenge E besteht aus roten Kanten E_r und schwarzen Kanten E_s, d.h. $E = E_r \cup E_s$ 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	∞	∞	8	8	8
b	∞	3	3	3	3
c	∞	∞	∞	∞	∞
s	∞	∞	∞	10	10

Good	0	1	2	3	4
t	0	0	0	0	0
a	∞	∞	4	4	4
b	∞	3	3	3	3
c	∞	3	3	3	3
s	∞	9	9	6	6

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

wenn Good < NoRed
dann Good.

Sonst
NoRed

Rep 2, 19

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]
```

Schritte

NoRed

Good

Case 1: Noch keine roten Kanten benutzt:

a) this, rote Kante + NoRed-Nachbar

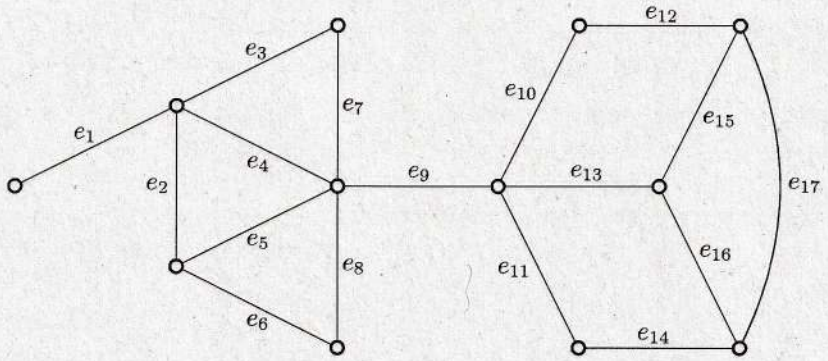
b) this, schwarze Kante + Good-Nachbar

Case 2: Bereits rote Kanten benutzt

wird Ausgeschlossen!
Vorgänger haben wir rot benutzt wenn sie good-Nachbe aufrufen!

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.

$C_A = C_{OPT}$

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

$C_A = 2 \cdot C_{OPT}$

Rep 2 - 2019

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	5	7	12	6

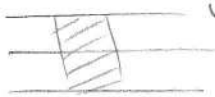
$\frac{W}{G}$

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, B, D, A

9	6	5	3	4
3	72	35	18	20
$\frac{3}{9}$	12	7	6	5

Berechnung der Schranken:



← Dualheuristik: Greedy + Restkapazität · $\frac{\text{Wert}}{\text{Gewichts}}$ Quotient

← Greedy

Depth first Strategie:

Zuerst möglichst weit weg vom Startknoten wie bei DFS

---

① $L' = 72 + 18 = 90 \longrightarrow L = 90$

$U' = 72 + 4 \cdot 7 = 72 + 28 = 100$

(keine Teilschränkungen)

c nehmen

② $L' = 72 + 18 = 90$

$U' = 100$

(mit C)

B nehmen

③ $L' = /$

$U' = /$

(mit B, C)

$g = 11 > 10$

STOP

B nicht nehmen

④ $L' = 72 + 18 = 90$

$U' = 72 + 18 + 1 \cdot 5 = 95$

(mit C, ohne B)

mit D

⑤ $L' = 72 + 18 = 90$

$U' = 72 + 18 + 1 \cdot 5 = 95$

(mit C, D, ohne B)

mit A

⑥ $L' = /$

$U' = /$

(mit C D A, ohne B)

$g = 6 + 3 + 4 > 10$

STOP

ohne A

⑦ $L' = 72 + 18 = 90$

$U' = 90$

(mit C D, ohne B A)

$U' = L'$ – STOP

ohne D

⑧ $L' = 72 + 20 = 92 \longrightarrow L = 92$

$U' = 72 + 20 = 92$

(mit C, ohne B, D)

$U' = L$ — STOP

c nicht nehmen

⑨ $L' = 35 + 18 = 53$

$U' = 35 + 18 + 2 \cdot 5 = 63$

(ohne C)

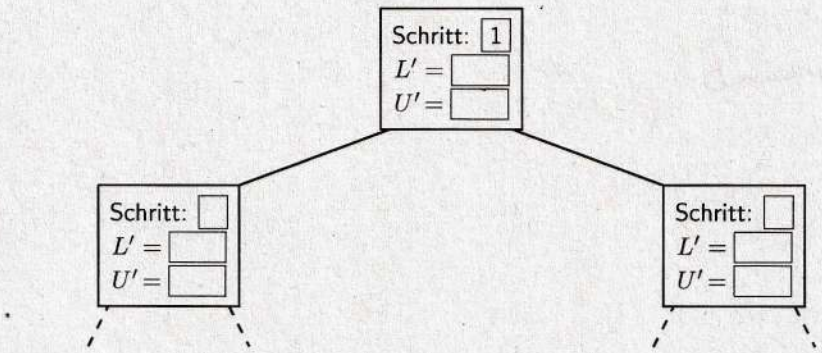
$U' < L$ — STOP

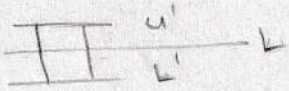
$63 < 90$

Klr 2 - 2019

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.

L = 92

{c, A}

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

186.866 Algorithmen und Datenstrukturen VU 8.0

1. Test SS 2018

26. April 2018

Gruppe A

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

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

Kennzeichnen Sie bei Ankreuzfragen eindeutig, welche Kästchen Sie kreuzen!

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

Viel Erfolg!

Rev 1 - 19

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

a) (3 Punkte) Geben Sie für das Codestück `FunktionA` die **Laufzeit** in Abhängigkeit von n in Θ-Notation an.

```
FunktionA(n):
Input: n ≥ 1
z ← 0
for j ← 1, ..., n
    z ← z + j + 1
    k ← 0
    while k ≤ n
        k ← k + j
while z > 0
    z ← ⌊z/2⌋
```

Laufzeit: $\Theta($ n^2 $)$

Hinweis: Es gilt $\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$.

b) (3 Punkte) Geben Sie für das Codestück `FunktionB` den **Rückgabewert** (z) in Abhängigkeit von n in Θ-Notation an.

```
FunktionB(n):
Input: n ≥ 1
i ← 1
a ← 1
repeat
    i = i + 1
    a = a · 2
until i ≥ log2 n
z ← 0
for j ← 0, ..., a
    z ← z + j + n
return z
```

Rückgabewert (z): $\Theta($ n^2 $)$

Aufgabe A1)

```
Funktion A (n)        Input: n ≥ 1
z = 0
for j ← 1 bis n                          Θ(n)
    z = z + j + 1
    k = 0
    while k ≤ n                          O(n log n)
        k = k + j
while z > 0                              Θ(log n)
    z = ⌊z/2⌋
```

for (i = 1 ; i ≤ n ; i++) ⟶ n Iterationen

$z = \sum_{j=1}^{n} j + n \cdot 1$ $\quad \left(\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \right)$

$z = \frac{n(n+1)}{2} + n = \frac{n^2+n+2n}{2} = \frac{n^2+3n}{2}$

Abbruchbedingung:

$k > n$

$k + i \cdot j > n$

↑ Iterationen

$k = 0$

$i = \frac{n}{j}$

↑ j steigt immer um 1 bis es n erreicht

Aussagesatz:

$i = \frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \dots + \frac{n}{n}$

$\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$

Deshalb

$n \cdot \sum_{i=1}^{n} \frac{1}{i} = \Theta(n \log n)$

$\lfloor z \cdot \left(\frac{1}{2}\right)^k \rfloor = 0$

Nach k Iterationen ist $z = 0$ (Abbruchbedingung)

$\lfloor z \cdot \left(\frac{1}{2}\right)^k \rfloor = 0$

$\log(z) + k \log\left(\frac{1}{2}\right) = 0$

$k \log\left(\frac{1}{2}\right) = -\log(z)$

$k = -\frac{\log(z)}{\log\left(\frac{1}{2}\right)}$

⟶ Nach $\Theta(\log n)$ Iterationen

Gesamtlaufzeit

$\Theta\left(n \cdot n \log(n) + \log(n)\right) = \Theta\left(n^2 \log(n)\right) = \Theta(n^2)$

Lösung laut Löser aber $\Theta(n \log n + \log n)$

Funktion B(n): $n \geq 1$ Rückgabewert z

```
i = 1
a = 1
repeat
  i = i + 1
  a = a · 2
until i ≥ log2 n
z = 0
for j ← 0 bis a
  z = z + j + n
return z
```

do-while Schleife:

Am Anfang $i = 2$, $a = 2$

Abbruchbedingung: $i = \log_2 n$

$i + k \cdot 1 = \log_2 n$

$i = 2$

$2 + k = \log_2 n$

$k = \log_2 n - 2$ Iterationen

Das bedeutet:

$a = a \cdot \underbrace{2^{\log_2(n)-1}}_{\Theta(n)}$ $\quad i = \log_2(n-2) \cdot 1 + 2$

$a = \Theta(2n)$

$2n+1$ Iterationen

$$z = \underbrace{\sum_{j=0}^{2n+1} j}_{\frac{(2n+1)(2n+2)}{2}} + (2n+1) \cdot n$$

$$z = \Theta(n^2 + n^2) = \Theta(n^2)$$

c) (4 Punkte)

(i) Vervollständigen Sie die folgende Definition:

Eine Funktion $T(n)$ ist in $O(f(n))$ wenn

(ii) Gegeben ist die folgende Funktion:

$$g(n) = \begin{cases} 134 & \text{falls } n = 5 \\ 2n^2 + \frac{n^2}{25} + 10 & \text{sonst} \end{cases}$$

Sind die nachfolgenden Abschätzungen wahr, wenn man die gegebenen Werte für n_0 und c in der jeweiligen Definition der asymptotischen Notation einsetzt? Kreuzen Sie die zutreffende Antwort an.

	n_0	c	Richtig	Falsch
g ist in $O(n^2)$	10	3	☐	☐
g ist in $O(n^3)$	4	1	☐	☐
g ist in $\Omega(1)$	6	100	☐	☐

(richtiges Kreuz +1 Punkt; falsches Kreuz -1 Punkt; kein Kreuz 0 Punkte; negative Summe wird auf 0 gesetzt)

Rep 2020

Aufgabe A2: Greedy-Algorithmen **(8 Punkte)**

a) (6 Punkte) Im Folgenden definieren wir ein Scheduling Problem. Gegeben ist eine Menge von Jobs $J = \{1, \ldots, n\}$, die alle abgearbeitet werden müssen. Für jeden Job, $j \in J$, ist eine Bearbeitungsdauer $p_j > 0$, und ein Gewicht $w_j > 0$ spezifiziert.

Wie beim Interval Scheduling, können Jobs nicht unterbrochen werden und es kann immer nur ein Job zu einem Zeitpunkt ausgeführt werden.

Ziel ist es, allen Jobs, $j \in J$, eine Startzeit s_j zuzuweisen, sodass die Summe der gewichteten Endzeiten $\sum_{j \in J} w_j f_j$ minimal ist, wobei die Endzeit eines Jobs, $j \in J$, durch $f_j = s_j + p_j$ gegeben ist.

Ein Algorithmus führt die Jobs unmittelbar aufeinanderfolgend aus. Zur Bestimmung der Reihenfolge sind zwei Strategien gegeben. Beurteilen Sie, ob das jeweilige Kriterium immer zu einer optimalen Lösung führt. Geben Sie ein Gegenbeispiel an, wenn Optimalität nicht garantiert ist.

(i) Ordne die Prozesse aufsteigend gemäß ihrer Bearbeitungsdauer.

Optimal: ☐ Ja ☒ Nein

Gegenbeispiel:

(ii) Ordne die Prozesse absteigend gemäß ihres Gewichtes.

Optimal: ☐ Ja ☒ Nein

Gegenbeispiel:

Hinweis: Spezifizieren Sie ggf. möglichst kleine Gegenspiele in Form konkreter Mengen von Paaren: $\{(p_1, w_1), (p_2, w_2), \ldots\}$. Eine Begründung ist nicht erforderlich.

Aufgabe 2)

Jobs $J = \{1, \dots, n\}$

p_j Bearbeitungsdauer

w_j Gewicht

s_j Startzeit

f_j Endzeit (Startzeit + Bearbeitungsdauer)

Ablauf:

Man soll eine gute Sortierung des Jobs finden, sodass sie ununterbrochen hintereinander von einem Automaten abgearbeitet werden können und

$\sum w_j \cdot f_j$ minimiert wird.

Wichtig:

w_j und p_j stehen fest.
Man bestimmt mit der Sortierung nur s_j

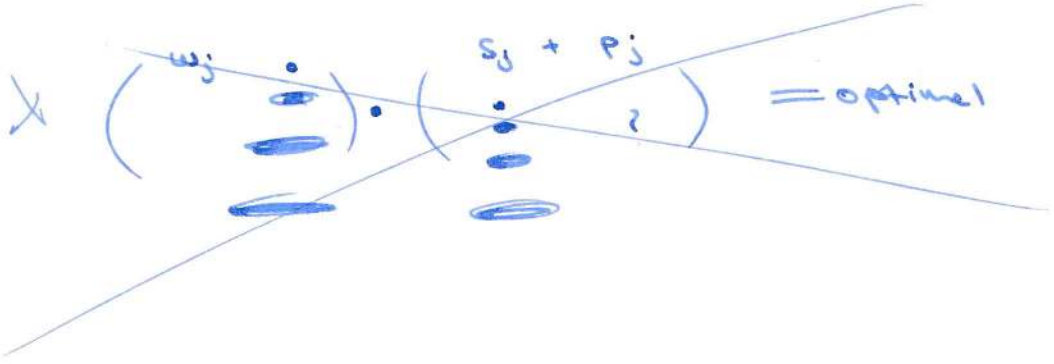
Strategie 1:

- Sortiere aufsteigend nach p_j

Strategie 2:

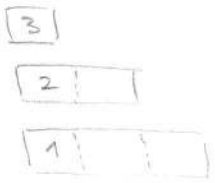
- Sortiere absteigend nach w_j

~~Strategie 2 ist optimal, weil $f_j = s_j + p_j$ und die Multiplikation fällt wesentlich mehr ins Gewicht als die addition deshalb~~



Gegenbeispiel für Strategie 2:

$\{(1,3), (2,2), (3,1)\}$



Paare als (p_j, w_j)

→ Aufsteigende Sortierung gemäß Bearbeitungsdauer

$p_1 = 1$ $p_2 = 2$ $p_3 = 3$

$w_1 = 3$ $w_2 = 2$ $w_3 = 1$

$s_1 = 0$ $s_2 = 1$ $s_3 = 3$

$f_1 = s_1 + p_1 = 1$ $f_2 = 3$ $f_3 = 6$

s_1 s_2 s_3

Summe

$3 \cdot (0+1) +$

$2 \cdot (1+2) +$

$1 \cdot (3+3) +$

$3 + 4 + 6 = 13$

Absteigende Sortierung gemäß Gewicht

$1 \cdot (0+3)$

$2 \cdot (3+2)$

$3 \cdot (5+1)$

$3 + 10 + 18 = 31$

Gegenbeispiel für Strategie 1:

$\{(2,1), (3,2)\}$

→ $\Sigma = 12$ 1. Strategie

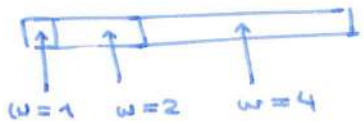
→ $\Sigma = 11$ 2. Strategie

i) Gegenbeispiel:

Wir haben pro Glied in der Summe 2 Faktoren: $\boxed{f_j \cdot w_j}$

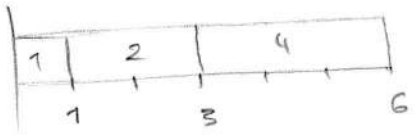
optimalerweise sollten große f kleine w bekommen und kleine f große w um jedes Glied der Summe zu minimieren.

Wenn aber die Prozesse nach aufsteigender Dauer sortiert werden:



Dann [große f · große w]!

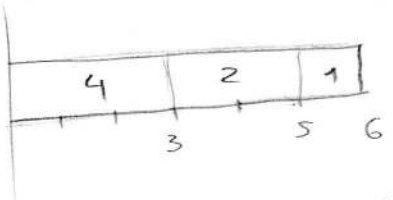
Beispiel:



$\Sigma = 1 + 6 + 24 = 31$

$J_1 \begin{cases} w_1 = 1 \\ s_1 = 0 \\ p_1 = 1 \\ f_1 = 1 \end{cases}$ $J_2 \begin{cases} w_2 = 2 \\ s_2 = 1 \\ p_2 = 2 \\ f_2 = 3 \end{cases}$ $J_3 \begin{cases} w_3 = 4 \\ s_3 = 3 \\ p_3 = 3 \\ f_3 = 6 \end{cases}$

Optimale Lösung wäre aber:

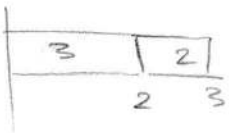


$\Sigma = 12 + 10 + 6 = 28$

ii) Gegenbeispiel

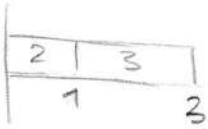
Gewichte sind absteigend

~~Reihenfolge der sind unbearbeitet auch absteigend~~



$\Sigma = 6 + 6 = 12$

Korrekt wäre aber:



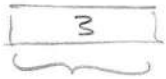
$\Sigma = 4 + 2 = 11$

f = Startzeit + Bearbeitung

klein für große Gewichte

Im Gegenbeispiel groß für große Gewichte

$J_1 = \begin{cases} w_1 = 2 \\ p_1 = 1 \end{cases}$



$J_2 = \begin{cases} w_2 = 3 \\ p_2 = 2 \end{cases}$

b) (2 Punkte) Betrachten Sie die folgenden Behauptungen in Bezug auf minimale Spannbäume für schlichte ungerichtete zusammenhängende Graphen und kreuzen Sie die korrekten an.

- ☐ Der minimale Spannbaum eines Graphen enthält zumindest eine Kante minimalen Gewichtes innerhalb jeder Kantenschnittmenge.
- ☐ Das Kreislemma besagt, dass die günstigste Kante jedes in einem Graph enthaltenen Kreises Teil des minimalen Spannbaumes dieses Graphen sein muss.
- ☐ Der Algorithmus von Prim und der Algorithmus von Kruskal berechnen gültige minimale Spannbäume und retournieren somit zwangsläufig denselben Spannbaum.
- ☐ Der Algorithmus von Kruskal ist wegen seiner Laufzeitkomplexität auf dünnen Graphen gegenüber dem Algorithmus von Prim zu bevorzugen.

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

Rep 2020

Aufgabe A3: Dijkstra und Heap **(12 Punkte)**

Gegeben ist die folgende Beschreibung einer Ausführung des Algorithmus von Dijkstra zum Finden eines kürzesten Pfades auf einem gerichteten Graphen in der Implementierung mit einer Liste.

1. Discovered $= \emptyset$, $L = \{s, v_1, v_2, v_3, v_4, u\}$

Vertex	s	v_1	v_2	v_3	v_4	u
$d(\cdot)$	0	∞	∞	∞	∞	∞

2. Discovered $= \{s\}$, $L = \{v_1, v_2, v_3, v_4, u\}$

Vertex	s	v_1	v_2	v_3	v_4	u
$d(\cdot)$	0	∞	5	∞	3	∞

3. Discovered $= \{s, v_4\}$, $L = \{v_1, v_2, v_3, u\}$

Vertex	s	v_1	v_2	v_3	v_4	u
$d(\cdot)$	0	∞	4	∞	3	∞

4. Discovered $= \{s, v_4, v_2\}$, $L = \{v_1, v_3, u\}$

Vertex	s	v_1	v_2	v_3	v_4	u
$d(\cdot)$	0	12	4	5	3	∞

5. Discovered $= \{s, v_4, v_2, v_3\}$, $L = \{v_1, u\}$

Vertex	s	v_1	v_2	v_3	v_4	u
$d(\cdot)$	0	9	4	5	3	15

6. Discovered $= \{s, v_4, v_2, v_3, v_1\}$, $L = \{u\}$

Vertex	s	v_1	v_2	v_3	v_4	u
$d(\cdot)$	0	9	4	5	3	12

7. Discovered $= \{s, v_4, v_2, v_3, v_1, u\}$, $L = \emptyset$

Vertex	s	v_1	v_2	v_3	v_4	u
$d(\cdot)$	0	9	4	5	3	12

a) (2 Punkte) Extrahieren Sie aus diesem Ablauf den kürzesten Pfad von s nach u sowie seine Länge. (Geben Sie den Pfad als Liste von Knoten an).

Pfad: s, v_4, v_2, v_3, v_1, u

Länge: 12

Aufgabe A3)

Dijkstra - Algorithmus

(ineffiziente Version)

```
Dijkstra(G, s)
foreach v ∈ V:                          ┐
    Discovered[v] = false               │ Initialisierung
d[s] = 0                                │
foreach v ∈ V\{s}                       │
    d[v] = ∞                            │
L ← V                                   ┘

while L ist nicht leer
    wähle u ∈ L mit min d[u]
    lösche u aus L
    Discovered[u] = true
    foreach (u,v) ∈ E
        if !Discovered[v]
            d[v] = min(d[v], d[u] + l_e)
```

s	v_1	v_2	v_3	v_4	4
0	∞	∞	∞	∞	∞
0	∞	5	∞	3	∞
0	∞	4	∞	3	∞
0	12	4	5	3	∞
0	9	4	5	3	15
0	9	4	5	3	12
0	9	4	5	3	12

● = updated
● = currently chosen

Discovered $D = \emptyset$
Not visited $L = \{s, v_1, v_2, v_3, v_4, 4\}$

smallest $d[u] : u \in L \rightarrow (s)$

$D = \{s\}$
$L = \{v_1, v_2, v_3, v_4, 4\}$

smallest $d[u] : (v_4)$

$D = \{s, v_4\}$
$L = \{v_1, v_2, v_3, 4\}$

smallest $d[u] : (v_2)$

$D = \{s, v_4, v_2\}$
$L = \{v_1, v_3, 4\}$

smallest $d[u] : v_3$

$D = \{s, v_4, v_2, v_3\}$
$L = \{v_1, 4\}$ → smallest : v_1

$D = \{s, v_4, v_2, v_3, v_1\}$
$L = \{4\}$ → smallest : 4

$D = V$ (alle knoten)
$L = \emptyset$

s updated v_2 and v_4 → prev $[v_2] = s$ prev $[v_4] = s$
v_4 updated v_2 → prev $[v_2] = v_4$
v_2 updated v_1 and v_3 → prev $[v_1] = v_2$ prev $[v_3] = v_2$
v_3 updated v_1 and 4 → prev $[v_1] = v_3$ prev $[4] = v_3$
v_1 updated 4 → prev $[4] = v_1$

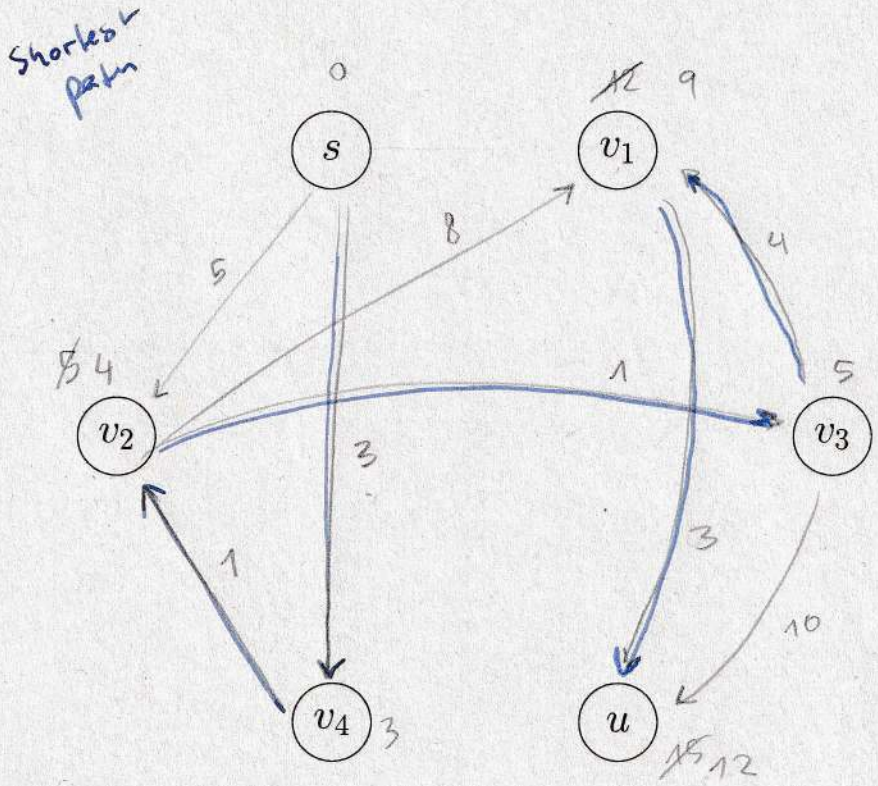
Backtracking: $4 \leftarrow v_1 \leftarrow v_3 \leftarrow v_2 \leftarrow v_4 \leftarrow s$

Komplexitätsvergleich:

MIN-HEAP		ARRAY
$O(1)$	finden eines minim. Elements	$O(1) \leftarrow A[0]$ ~~$O(\log_2 n)$ wenn sortiert sonst $O(n)$~~
$O(1) + \underbrace{O(\log n)}_{\text{heapify-up}}$	Einfügen eines beliebigen Elements	$O(1) + O(n) + O(n \log n)$ (Array neu erstellen und alle Elemente übertragen) + sortieren
„init-A" $O(n)$	Erstellen aus unsortiertem Array	$O(1) + O(n \log n)$ (Referenz übernehmen) + sortieren
Mit inorder Traversal alle Nodes durchgehen in $O(n)$	Finden des maximalen Elements	$O(1) \leftarrow A[n-1]$

Rep 2020

b) (4 Punkte) Zeichnen Sie die Kanten und Kantengewichte eines minimalen (so wenige Kanten wie möglich) Graphen, der zu einem solchen Ablauf führt, in der folgenden Grafik ein.



c) (2 Punkte) Geben Sie für die folgenden Operationen an, ob sie im Worst-Case asymptotisch in Bezug auf die Anzahl der Elemente schneller in einem Min-Heap, in einem sortierten Array oder in beiden gleich schnell ausgeführt werden können.

(i) Finden des minimalen Elements:

☐ Min-Heap ☐ sortiertes Array ☒ asymptotisch gleich schnell

(ii) Einfügen eines beliebigen Elements:

☒ Min-Heap ☐ sortiertes Array ☐ asymptotisch gleich schnell

(iii) Erstellen aus einem unsortierten Array:

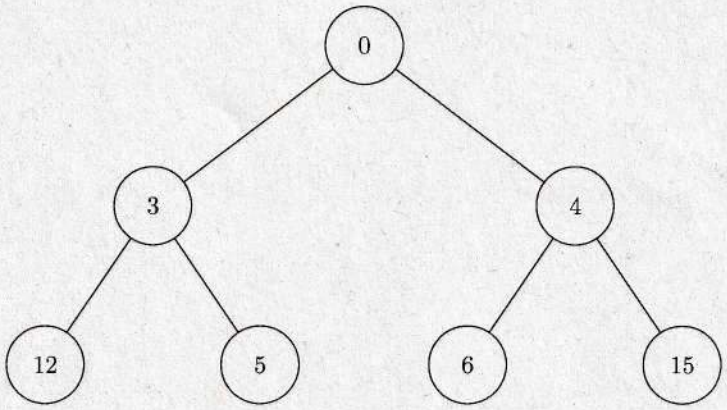
☒ Min-Heap ☐ sortiertes Array ☐ asymptotisch gleich schnell

(iv) Finden des maximalen Elements:

☐ Min-Heap ☒ sortiertes Array ☐ asymptotisch gleich schnell

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

d) (4 Punkte) Gegeben ist der folgende Min-Heap H:



Hinweis: Gehen Sie bei den Teilaufgaben (i) und (ii) jeweils von diesem Heap aus.

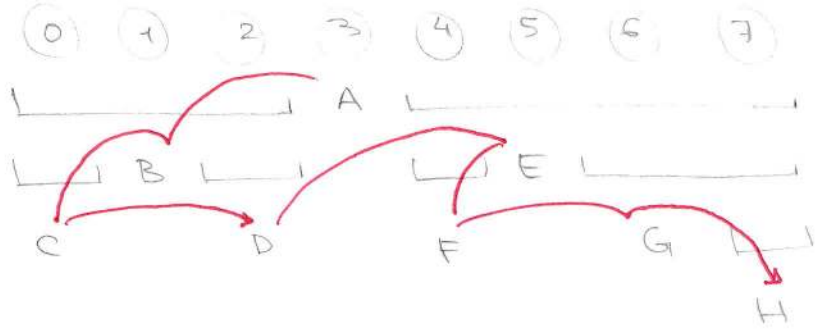
(i) Zeichnen Sie den Heap H, wie er nach dem Einfügen des Element 1 aussieht.

(ii) Zeichnen Sie den Heap H, wie er nach dem Löschen des Element 0 aussieht.

Worst Case verursachen bei Quicksort mit $n=7$, Pivot $\lfloor \frac{n}{2} \rfloor$

kleinste Zahl = A

größte Zahl = H



Erstes Element $\lfloor \frac{n}{2} \rfloor$: $\lfloor \frac{7}{2} \rfloor = 3 \leftarrow A$ $e = 3$ (erstes Element)

Linke Hälfte $\lfloor \frac{e-1}{2} \rfloor$: $\lfloor \frac{3-1}{2} \rfloor = 1 \leftarrow B$

Rechte Hälfte $\lfloor \frac{n-e}{2} \rfloor + e$: $\lfloor \frac{7-3}{2} \rfloor + 3 = 5 \leftarrow E$

usw...

$\lfloor \frac{7}{2} \rfloor = 3$ (A)

(B) $\lfloor \frac{3-1}{2} \rfloor = 1$ $\lfloor \frac{7-3}{2} \rfloor + 3 = 5$ (E)

(C) $\lfloor \frac{1-1}{2} \rfloor = 0$ $\lfloor \frac{3-1}{2} \rfloor + 2 = 2$ (D) $= 4$ (F) $\lfloor \frac{7-5}{2} \rfloor + 3$ $\lfloor \frac{7-5}{2} \rfloor + 5 = 6$ (G)

$= 7$ (H)

Rep 1 – 19

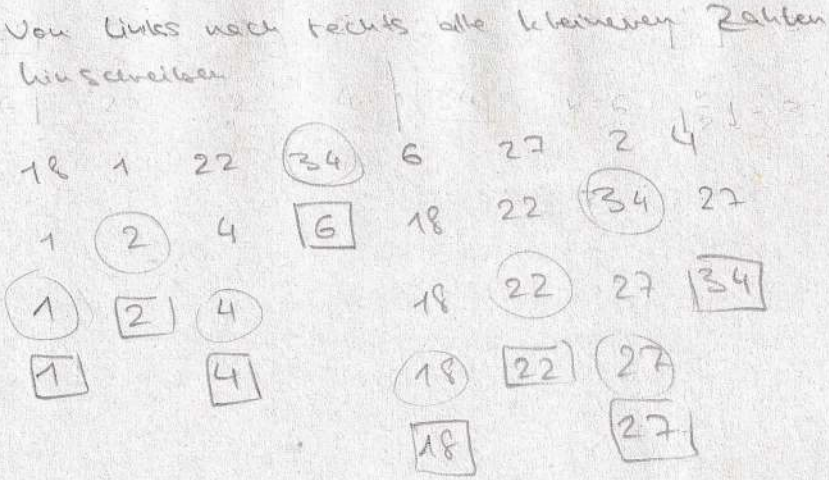
Aufgabe A4: Divide-and-Conquer **(10 Punkte)**

a) (6 Punkte) Führen Sie den Quicksort-Algorithmus auf folgendes Array (Index startet bei 0) aus, um es aufsteigend zu sortieren:

$$[\,18,\ 1,\ 22,\ 34,\ 6,\ 27,\ 2,\ 4\,]$$

Als Pivotelement wird jeweils das Element an der Position $\lfloor \frac{n}{2} \rfloor$ gewählt. Geben Sie die Zwischenschritte nach jedem Aufteilen an und markieren Sie jeweils das als nächstes gewählte Pivotelement.

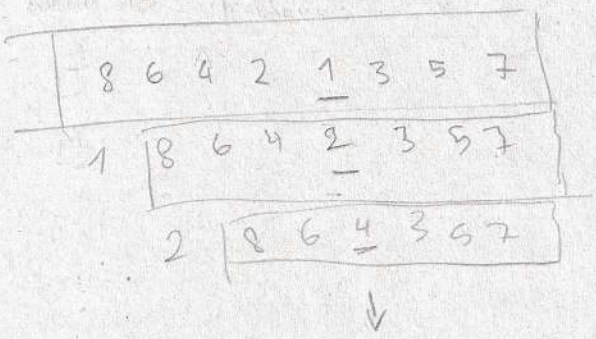
Der Schritt des Aufteilens ist so implementiert, dass die Elemente einer Subfolge immer in der gleichen Reihenfolge angeordnet werden wie in der Originalfolge.



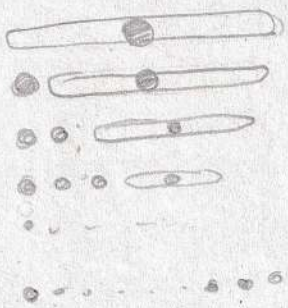
b) (4 Punkte) Geben Sie ein Eingabearray mit 8 unterschiedlichen Elementen an, sodass der Quicksort-Algorithmus aus Teilaufgabe a) Worst-Case-Laufzeit hat.

Worst-case = maximale # an Ebenen also maximale Rekursionstiefe

Lösung



wenn bei $\lfloor \frac{n}{2} \rfloor$ immer die kleinste Zahl steht

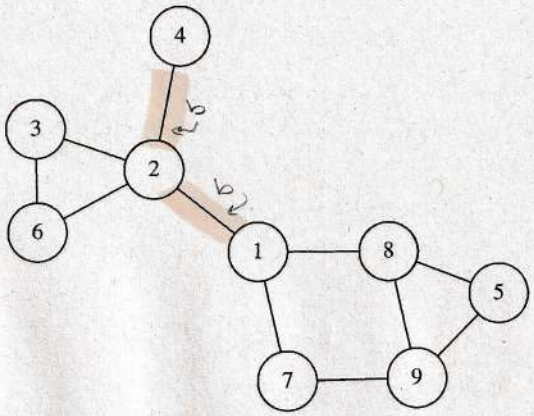


↑ maximale Verschiebungs-Anzahl

Rep 2020

Aufgabe A5: Graphen **(10 Punkte)**

Sei G der folgende ungerichtete schlichte zusammenhängende Graph mit Knoten V und Kanten E:



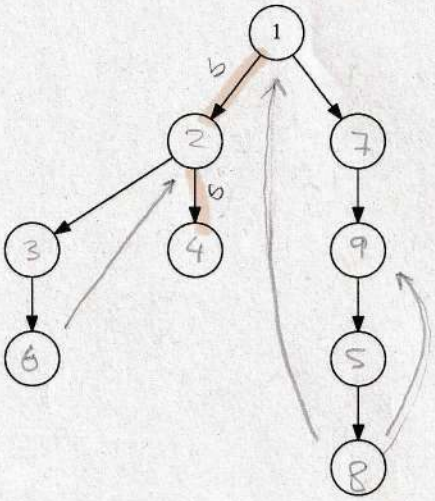
Macht das Entfernen einer Kante $b \in E$ den Graphen unzusammenhängend, so nennen wir b eine **Brücke**.

a) (2 Punkte) Markieren Sie im oben abgebildeten Graphen alle Brücken.

b) (2 Punkte) Wir führen nun im Graphen eine Tiefensuche durch. Aus der Entdeckungsreihenfolge der Knoten entsteht ein Baum. Eine Kante $v \to w$ in diesem Baum drückt aus, dass w von v aus entdeckt wurde. Wir bezeichnen v als **Elternknoten** von w.

Wird bei der Tiefensuche durch eine Kante r ein bereits entdeckter Knoten geringerer Tiefe gefunden, der nicht der Elternknoten ist, so nennen wir r eine **Rückwärtskante**. Das heißt, sie beginnt unten und zeigt auf einen Knoten weiter oben im Baum.

Vervollständigen Sie den unten dargestellten Tiefensuchbaum für den oben abgebildeten Graphen. Lässt die Suche eine Auswahl zwischen Knoten zu, wählen Sie den numerisch kleinsten. Ergänzen Sie dabei die Rückwärtskanten.



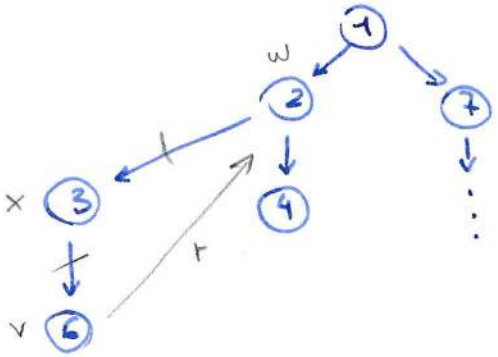
Rückwärtskanten + DFS-Graph!

Brücken aus ursprünglichem Graphen sind hier weiterhin Brücken (wenn Graph jetzt ungerichtet wäre)

Aufgabe A5, Graphen) c)

(Ausschnitt aus Code)

```
foreach (v, w) ∈ R
    while v ≠ w
        x = Eltern [v]      ← bezieht sich nicht nur auf R
        B = B \ (x, v)      ← DFS Baum
        v = x
return B                    ← Aus DFS-BAUM werden Brücken ermittelt
```

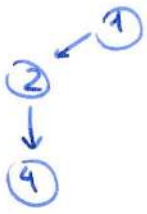


1. $v = 6$, $w = 2$, $x = 3$
   In B gibt es eine gerichtete Kante von x nach v
   $v \neq w \rightarrow v = x$, $x = \text{Eltern}[v]$

2. $v = 3$, $w = 2$, $x = 2$
   In B gibt es eine gerichtete Kante von x nach v
   $v \neq w \rightarrow v = x$, $x = \text{Eltern}[v]$

3. $v = 2$, $w = 2$, $x = 1$ → $v = w$, Abbruch

übriger Graph B:



Rep 2020

c) (6 Punkte) Es gilt nun einen effizienten Algorithmus zur Bestimmung aller Brücken eines beliebigen ungerichteten schlichten zusammenhängenden Graphen zu finden.

Kreuzen Sie im folgenden Pseudocode die Codezeilen an, die ausgeführt werden müssen, um die Brücken zu finden. Je Block ist genau eine Zeile richtig.

Hinweis: Teilaufgabe b) gibt bereits einen guten Hinweis auf die Idee des gesuchten Algorithmus. Betrachten Sie dabei die Rückwärtskanten und Brücken im Baum.

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

```
// Mengen R und B, sowie Arrays Tiefe und Eltern sind global.

Brücken(Graph G, Startknoten s):
    R ← ∅                          // Rückwärtskanten
    B ← ∅                          // Baumkanten von DFS
    foreach v ∈ V do
        Tiefe[v] ← −1
        Eltern[v] ← null
    call Tiefensuche(G, s, 0)      <— Tiefe[s] = 0
    ☒ foreach (v, w) ∈ R do
    ☐ foreach (v, w) ∈ B do
    ☐ foreach (v, w) ∈ E do
        while v ≠ w
            x ← Eltern[v]          <— bezieht sich auf B
            ☐ B ← B ∪ {(x, v)}
            ☐ B ← B ∩ {(x, v)}
            ☒ B ← B \ {(x, v)}
            v ← x
    return B

Tiefensuche(Graph G, Knoten v, Tiefe d):
    Tiefe[v] ← d                   Tiefe[v] = d
    // für jeden an v adjazente Knoten w
    foreach (v, w) ∈ E do
        if Tiefe[w] = −1 then
            Eltern[w] ← v
            B ← B ∪ {(v, w)}
            call Tiefensuche(G, w, d + 1)
        else if (Eltern[v] ≠ w) and (d > Tiefe[w]) then
            ☒ R ← R ∪ {(v, w)}
            ☐ R ← R ∪ {(w, Eltern[v])}
            ☐ R ← R ∪ {(v, Eltern[w])}
```

DFS Baum

r

Eltern

Suche fortsetzen mit Tiefe++

Bereits erkundet

Tiefer als w

nicht Eltern von v

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.813 Algorithmen und Datenstrukturen 1 VU 6.0

Nachtragstest SS 2017

05. Oktober 2017

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	20	18	12	50
Erreichte Punkte:				

Viel Erfolg!

Aufgabe A1: Algorithmenanalyse **(20 Punkte)**

a) (4 Punkte) Tragen Sie für das Codestück `FunktionA` die Laufzeit in Abhängigkeit von n in Θ-Notation in die nachfolgende Tabelle ein.

	`FunktionA`
Laufzeit	

```
FunktionA(n):
for i ← 1 bis ⌈n/2⌉
    for j ← ⌊n/2 + 1⌋ bis n
        x ← 1
        while x ≤ n
            x ← x + x
```

b) (4 Punkte) Tragen Sie für das Codestück `FunktionB` den Rückgabewert (z) in Abhängigkeit von n in Θ-Notation in die nachfolgende Tabelle ein.

	`FunktionB`
Rückgabewert (z)	

```
FunktionB(n):
x ← n
z ← n
while x > 1
    for j ← 1 bis 3n
        z ← z * 2
    x ← x − 1
    z ← z − 1
return z
```

c) (4 Punkte) Tragen Sie für das Codestück `FunktionC` die Laufzeit in Abhängigkeit von n in Θ-Notation in die nachfolgende Tabelle ein.

	`FunktionC`
Laufzeit	

```
FunktionC(n):
x ← 1
y ← 50
while x < 2^n
    for z ← 1 bis y
        x ← zx
```

d) (4 Punkte) Beantworten Sie die nachfolgenden Fragen und begründen Sie jeweils Ihre Antwort in **wenigen** Worten!

- Gilt $2^{n+1} = \Theta(2^n)$?

- Gilt $n^{n+1} = \Theta(n^n)$?

e) (4 Punkte) Sie haben ein Array A von beliebigen ganzen Zahlen gegeben. Darin soll das minimale Element gefunden werden. Nachfolgend sind verschiedene Methoden für das Suchen beschrieben. Geben Sie die Worst-Case-Komplexität jeder Methode in Θ-Notation an. Gehen Sie dabei von den vorgestellten Implementierungen in der Vorlesung aus.

Methode	$\Theta(\cdot)$
Lineare Suche	
Anwenden von Quicksort (auf A aufsteigend sortierend) und danach Zugriff auf das erste Element	
Erstellen eines Min-Heaps und Zugriff auf die Wurzel	
Erstellen eines binären Suchbaums (Elemente aus A werden dabei der Reihe nach eingefügt) und ermitteln des minimalen Elements	

Rep 1 - 2018

Aufgabe A2: Greedy und Divide and Conquer **(18 Punkte)**

a) (12 Punkte) Sie haben folgendes Problem gegeben: In einer Liste L sind die Größen von n Dateien $F_1, F_2, \ldots, F_n$ gespeichert. Dabei entspricht L_i der Größe von Datei F_i. Die Aufgabe ist nun, alle Dateien mit minimalem Aufwand zu einer großen Datei zu verschmelzen. Werden zwei Dateien F_i und F_j verschmolzen, dann ist der Aufwand dafür (und die Größe der dabei neu erzeugten Datei) $L_i + L_j$.

I) Sie haben folgenden Algorithmus gegeben:

`GreedyMerge()`:
while *L enthält noch mehr als einen Eintrag*
Wähle die ersten beiden Einträge L_1 und L_2 der Liste L
Verschmelze die zu L_1 und L_2 gehörenden Dateien zu einer neuen Datei M
Lösche L_1 und L_2 aus L
Füge die Größe von M am Anfang der Liste L wieder ein

Dieser Algorithmus liefert keine optimale Lösung, d.h. die Summe aller Aufwände ist nicht minimal. Zeigen Sie das mit einem Beispiel!

Gegenbeispiel

$L = 3, 2, 1$ $F_1 = 3$ $F_2 = 2$ $F_3 = 1$

1. Schritt $\Theta(5)$
$3+2 = M$
$L' = 5, 1$

Insgesamt: $\Theta(11)$

2. Schritt $\Theta(6)$
$5+1 = M$
$L'' = 6$

Verbessert

$L = 3, 2, 1$

$\Theta(3)$
$M = 2+1$
$L' = 3, 3$

$\Theta(6)$
$M = 3+3$
$L'' = 6$

Insgesamt eine Verbesserung
$\Theta(9)$

Aufgabe A2)

L speichert n Dateigrößen F_i von Datei i

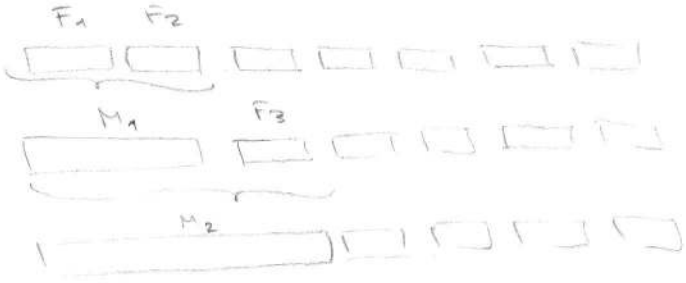
$L[i] = F_i$

Ziel: Alle n Dateien mit je Größe = Aufwand F_i zu mergen

Merge $F_i, F_j = O(F_i + F_j)$

Greedy-Merge:

```
While Queue not empty
    L1 = pop.queue
    L2 = pop.queue
    M  = merge(L1, L2)
    push.queue(M)
```



Beispiel

$F_1 = c_1$ $F_2 = c_2$ ... $F_n = c_n$

$M_1 = c_1 + c_2$

$M_2 = c_1 + c_2 + c_3$

$M_3 = c_1 + c_2 + c_3 + c_4$

$\vdots$

$M_n = \sum_{k=1}^{n+1} c_k$

Problem:

Angenommen $F_1 = c$

Dann wird F_1 (als Inhalt von M) n-1 weitere Male gemerged.

Algorithmus ist auch nicht greedy weil Liste nicht nach größe sortiert ist.

Lösung:

Divide and Conquer Algorithmus

1 Layer:
Alle Elemente nur 1x effektiv gemerged

Somit wird ein Element maximal $\lceil \log_2(n) \rceil$-Mal zur Laufzeit Last

Layer-Tiefe

$\lceil \log_2(6) \rceil = 3$

II) Sie haben weiters folgenden Algorithmus gegeben:

```
PairMerge():
```
while *L enthält noch mehr als einen Eintrag*
Verschmelze jeweils Paare von Dateien (d.h. F_1 und F_2, F_3 und F_4 usw.)
Erzeuge eine neue Liste L mit den neuen $\left\lceil \frac{|L|}{2} \right\rceil$ Dateigrößen

Divide and Conquer ←

Hinweise:

- Alle Paare werden immer nebeneinander verschmolzen, d.h. man kann nicht beim Verschmelzen eines Paares auf das Ergebnis des vorherigen Paares zugreifen.
- Bei einer ungeraden Anzahl an Dateien wird eine einzige Datei nicht verschmolzen. Ihre Größe verändert sich nicht, sie wird aber danach wieder berücksichtigt.

Liefert dieser Algorithmus eine optimale Lösung, d.h. ist die Summe aller Aufwände minimal? Wenn nein, dann geben Sie ein Gegenbeispiel an!

Nicht immer optimal:

Angenommen $L = 1, 2, 3$ dann worst-case siehe letztes Beispiel: $O(11)$

III) Wie müsste man den Algorithmus `GreedyMerge` anpassen, damit er eine optimale Lösung liefert?

Liste müsste aufsteigend sortiert sein

(trotzdem divide and conquer im best case besser)

b) (6 Punkte)

Führen Sie Mergesort auf dem Array $[5, 4, 3, 2, 1]$ aus und geben Sie den Ablauf des Aufrufes in Form eines Ablaufdiagrammes (wie in Vorlesung und in Übung bereits verwendet) an.

Aufgabe A3: Bäume und Graphen (12 Punkte)

a) (6 Punkte)

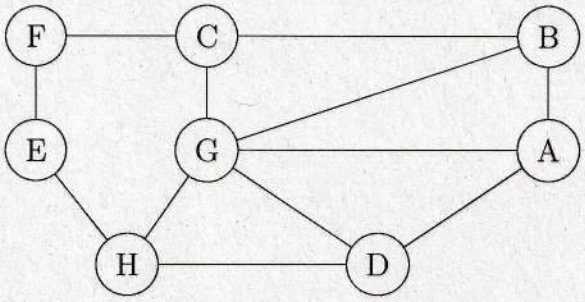
Sie haben folgende Durchmusterungen eines binären Baumes gegeben:

Preorder: 10, 20, 5, 15, 50, 30, 40
Inorder: 5, 20, 15, 50, 10, 40, 30

Rekonstruieren Sie den originalen Baum!

b) (6 Punkte)

Gegeben ist der folgende ungerichtete Graph G:



Auf dem gegebenen Graphen G wird die aus dem Skriptum bekannte **Tiefensuche** durchgeführt. Welche der folgenden Listen von besuchten Knoten können dabei in genau dieser Reihenfolge entstehen, wenn die Nachbarn eines Knotens in lexikographischer bzw. in beliebiger Reihenfolge abgearbeitet werden? Kreuzen Sie Zutreffendes an.

Reihenfolge	lexikograph.	beliebig	nicht möglich
A B C F E H D G			
A B C D E F G H			
A G H D E F C B			

Jede Zeile wird nur dann gewertet, wenn sie vollständig richtig ist.

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.815 Algorithmen und Datenstrukturen 2 VU 3.0

1. Test SS 2017

29. Juni 2017

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

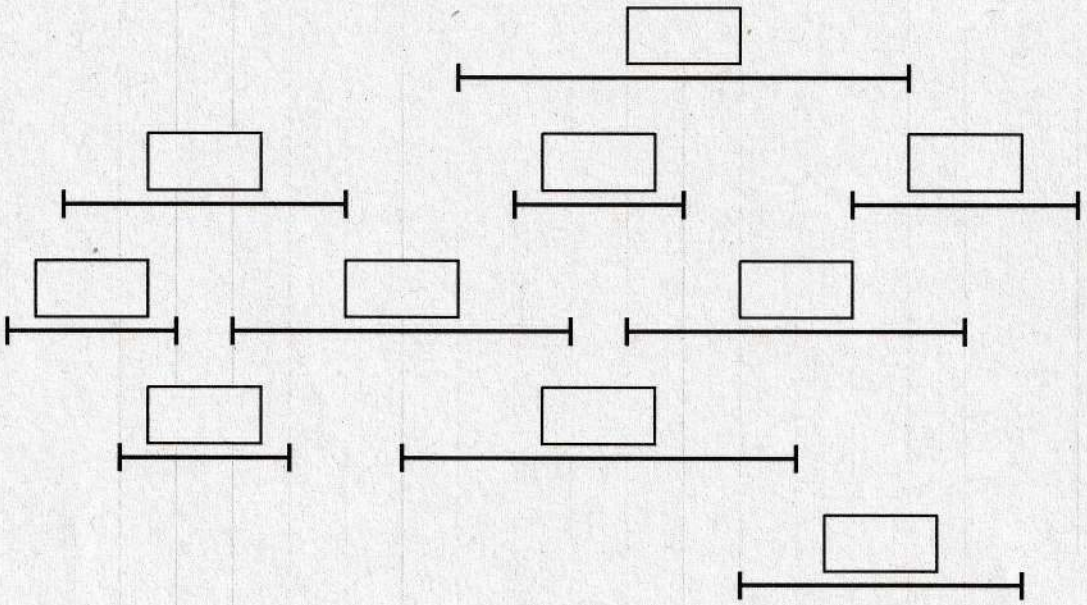
	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	16	16	18	50
Erreichte Punkte:				

Viel Erfolg!

Aufgabe A1: Reduktionen und NP-Spezialfälle (16 Punkte)

a) (8 Punkte)

Gegeben sei die untenstehende Intervallrepräsentation eines Graphen $G = (V, E)$. Führen Sie den aus der Vorlesung bekannten Algorithmus zum Färben von Intervallgraphen mit einer minimalen Anzahl von Farben $\{0, 1, 2, \dots\}$ aus. Schreiben Sie an jedes Intervall dessen Farbe.



Rep 2 – 2018

b) (8 Punkte)

Seien A,B,C Ja/Nein-Probleme in NP und n die Eingabegröße. Nehmen Sie an, es gibt

- eine Reduktion von A nach B in Zeit $O(n^3)$,
- eine Reduktion von B nach C in Zeit $O(n!)$,
- eine Reduktion von C nach A in Zeit $O(n^2)$.

$A \leq_P B$
$B \leq_{NP} C$
$C \leq_P A$
$\rightarrow C \leq_P A \leq_P B$

1) Nehmen Sie an, es existiert ein Algorithmus, der Problem B in Zeit $O(n^4)$ löst. Was können Sie dann über die anderen Probleme aussagen? Kreuzen Sie die richtigen Antworten an.

$B \in O(n^4)$

$O((n^3)^4) \rightarrow$
$O(((n^2)^3)^4) \rightarrow$

B in $O(n^4)$	in P	NP-vollständig	keine Aussage möglich
A ist	✓		
C ist	✓		

2) Nehmen Sie an, A sei NP-vollständig. Was können Sie dann über die anderen Probleme aussagen? Kreuzen Sie die richtigen Antworten an.

$A \leq_P B$

A NP-vollständig	in P	NP-vollständig	keine Aussage möglich
B ist		✓	
C ist			✓

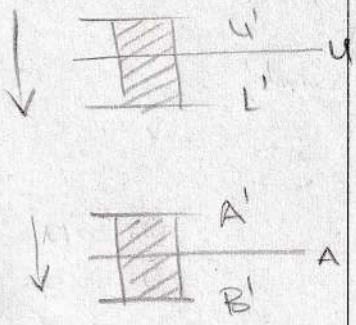
$A \in NP - C$
$B \in NP$
$C \in NP$

Rep 2 - 2018

Aufgabe A2: Branch-and-Bound **(16 Punkte)**

a) (8 Punkte)

Betrachten Sie folgenden Pseudocode eines generischen Branch-and-Bound-Algorithmus für ein **Minimierungsproblem**.




```
Branch-and-Bound-Min(I):
A ← Wert einer initialen heuristischen Lösung
Π ← {I}
while ∃I' ∈ Π
    Entferne I' aus Π
    Berechne für I' lokale [ untere ] Schranke B'
    if A [ > ] B'
        A' ← Wert einer heuristischen Lösung für I'
        if A' entspricht einer gültigen Lösung für I'
            if A' < A
                [ A ≤ A' ]
        if B' [ < ] A
            Partitioniere I' in Teilinstanzen I_1, ..., I_k
            Π ← Π ∪ {I_1, ..., I_k}
return beste gefundene Lösung mit Wert A
```

Füllen Sie die vier Lücken im Pseudocode so aus, dass ein korrekter Branch-and-Bound-Algorithmus entsteht.

b) (2 Punkte)

Beschreiben Sie in 1–2 Sätzen die Best-First-Strategie zur Auswahl des nächsten zu bearbeitenden Teilproblems in einem Branch-and-Bound Algorithmus.

Best first:
Abhängig von Dualheuristik (Best-Case-Schranke)
wird die nächste Node gewählt
(die mit der besten „best case-Schranke")

Rep 2 - 2018

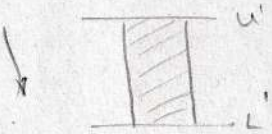
c) (6 Punkte)

Ist der folgende Algorithmus geeignet, um eine untere Schranke in einem Branch-and-Bound-Algorithmus zum Finden eines minimalen Vertex Cover in einem Graphen $G = (V, E)$ zu berechnen? Begründen Sie Ihre Antwort (falls Ihre Antwort 'JA' ist, begründen Sie warum die Schranke *immer* gilt, falls Ihre Antwort 'NEIN' ist, erklären Sie warum die Schranke *nicht immer* gilt).

```
Maybe-Lower-Bound(G = (V, E)):
bound ← 0
while E ≠ ∅
    (u, v) ← beliebige Kante aus E
    if ∃w ∈ V mit (u, w), (v, w) ∈ E
        bound ← bound+3
        entferne u, v, w aus V und alle zu u, v, w inzidenten Kanten aus E
    else
        bound ← bound+1
        entferne u, v aus V und alle zu u, v inzidenten Kanten aus E
return bound
```

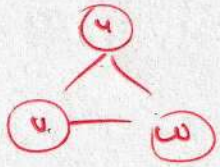
Eigenschaften der unteren Schranke:

- muss nicht gültige Lösung liefern sondern nur „best-case" darstellen



- Echte Lösung darf nicht kleiner als L' sein

Gegen-Beispiel



Tatsächliche MIN-VC-Lösung: 2

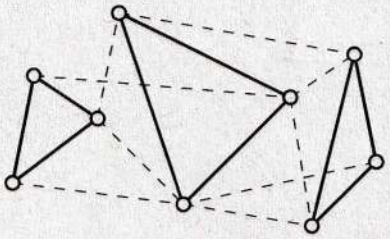
Best-case Einschätzung: 3

Rep 2-2018

Aufgabe A3: Approximation **(18 Punkte)**

Betrachten Sie das aus der Vorlesung bekannte Minimierungsproblem KLEINSTES VERTEX COVER für einen Graphen $G = (V, E)$ mit $|V| = 3n$ Knoten. Nehmen Sie an, G lässt sich in ein perfektes Dreiecksmatching zerteilen. Das bedeutet, es gibt eine Menge $T = \{\Delta_1, \ldots, \Delta_n\}$ von n knotendisjunkten Dreiecken, wobei jedes $\Delta_i = (V_i, E_i)$ ein Teilgraph von G mit drei Knoten und drei Kanten ist, d.h. $V_i = \{v_1^i, v_2^i, v_3^i\}$ und $E_i = \{(v_1^i, v_2^i), (v_2^i, v_3^i), (v_3^i, v_1^i)\}$. Weiters gilt $V = \bigcup_{i=1}^{n} V_i$.

Die Abbildung zeigt ein Beispiel eines perfekten Dreiecksmatchings (dicke Kanten) in einem Graphen mit 9 Knoten.



a) (3 Punkte)

Der vollständige Graph K_n mit n Knoten ist ein Graph, in dem alle Knoten paarweise durch eine Kante verbunden sind. Lässt sich der vollständige Graph K_{12} in ein perfektes Dreiecksmatching zerteilen? Begründen Sie kurz Ihre Antwort.

b) (3 Punkte)

Wie viele Knoten enthält ein kleinstes Vertex Cover in dem Graphen K_{12}?

Aufgabe A3)

a) Perfektes „Dreiecks-Matching" vom vollständigen Graph $K_{12} \leftarrow |V|$

Möglich da alle Knoten verbunden sind!

12 : 3 = 4

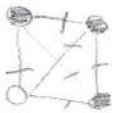
Dreiecke Disjunkte Mengen

b) Kleinstes VC in K_{12}

Beispiel: K_3

$|VC| = 2$

K_4



$|VC| = 3$

...

K_{12}

$|VC| = 12 - 1 = 11$

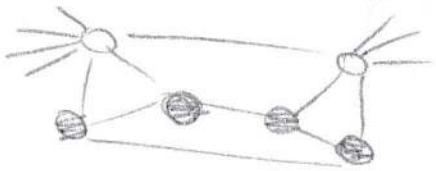
Begründung:

(Es gibt im Vollständigen Graph K_n genau $\frac{n \cdot (n-1)}{2}$ Kanten)

Angenommen es gibt 2 Knoten die nicht Element vom VC Set sind, dann gibt es zwischen ihnen immer eine Kante

Deshalb müssen n-1 Knoten gedeckt werden.

c) (Ich finde die vorgezeigte Lösung ungültig, da $\Omega(2n)$ keine Schranke sein kann)



Lösung

$G = (V, E)$

$|V| = 3n$

$|VC(G)| = \Omega(2n) = c_{OPT}$

$$\frac{c_A}{c_{OPT}} = \frac{|V|}{\Omega(2n)} = \frac{3n}{\Omega(2n)} \leq \frac{3/2}{} = \varepsilon$$

↑

kann größer sein, deshalb $\leq$

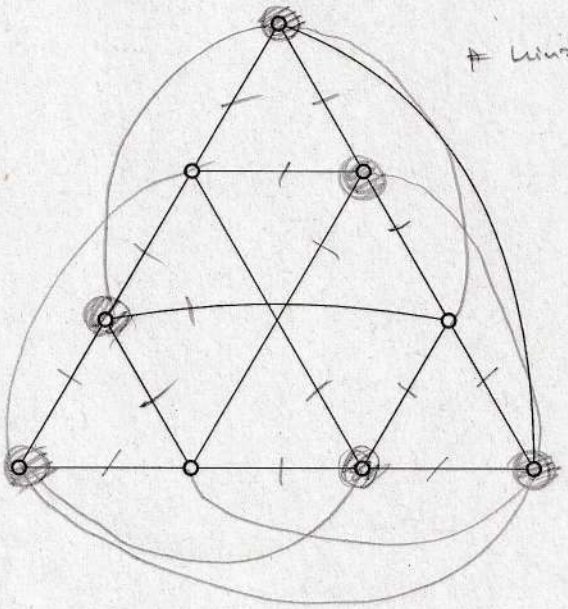
$$\frac{c_A}{c_{OPT}} \leq \varepsilon$$

Rep 2 - 2018

c) (6 Punkte)

schlicht = keine Schlingen ∨ Mehrfachkanten

Unten sehen Sie einen Graphen mit 9 Knoten und 16 Kanten. Können Sie 7 Kanten hinzufügen, so dass der entstehende Graph einfach ist und ein Vertex Cover der Größe 6 enthält? Zeichnen Sie, falls möglich, die 7 Kanten ein und markieren Sie ein Vertex Cover der Größe 6. Andernfalls zeichnen Sie weniger als 7 Kanten ein und markieren Sie ein Vertex Cover der Größe 6.



hinzugefügte Kanten = 7

✓ gültiges VC mit 6 Knoten

d) (6 Punkte)

Die komplette Knotenmenge V ist nach Definition immer ein Vertex Cover. Welche bestmögliche Gütegarantie erfüllt die Menge V für das Problem KLEINSTES VERTEX COVER für einen Graphen G, der sich in ein perfektes Dreiecksmatching zerlegen lässt? Beweisen Sie Ihre Antwort. Betrachten Sie dazu die Lösungsgröße c_{opt} eines kleinsten Vertex Covers von G und vergleichen Sie c_{opt} mit $|V|$.

↳ Alle Knoten im Graph sind gültiges Vertexcover

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.813 Algorithmen und Datenstrukturen 1 VU 6.0

1.Übungstest SS 2017

27. April 2017

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	18	12	20	50
Erreichte Punkte:				

Viel Erfolg!

Rep1 2018

Aufgabe A1: Algorithmenanalyse **(18 Punkte)**

a) (10 Punkte) Tragen Sie für das Codestück `FunktionA` die **Laufzeit** in Abhängigkeit von n in Θ-Notation in die nachfolgende Tabelle ein. Tragen Sie für das Codestück `FunktionB` den **Rückgabewert** (z) in Abhängigkeit von n in Θ-Notation in die nachfolgende Tabelle ein.

	`FunktionA`
Laufzeit	

	`FunktionB`
Rückgabewert (z)	

```
FunktionA(n):
a ← 2
b ← 2n²
while b > 1
    a ← a + 1
    c ← n
    while c ≥ n
        b ← b − a
        a ← ⌊a/2⌋
        c ← ⌊2n/c⌋
```

```
FunktionB(n):
x ← 1
for j ← 1 bis ⌊log₅ n⌋
    x ← 4 · x
    z ← j
    x ← ⌊x/2⌋
for j ← 1 bis ⌊log₂ x⌋
    z ← z − 1
return z
```

Rep 2020

b) (8 Punkte) Gegeben sei die folgende Funktion $f\colon \mathbb{N}^+ \to \mathbb{R}$

$$f(n) = \begin{cases} \dfrac{1}{2\log(n)} - 2n + 2^n & \text{wenn } n > 42 \text{ und ungerade} \\ \left(\dfrac{7}{2}\right)^n + \dfrac{3}{4}n^2 - \log(n) & \text{wenn } n \leq 42 \\ n^{-3} \cdot 3^n - (n+1)^3 + \log(2^n) & \text{sonst} \end{cases}$$

Kreuzen Sie **in der folgenden Tabelle** die zutreffenden Felder an:

$f(n)$ ist	$O(.)$	$\Omega(.)$	$\Theta(.)$	keines
e^n				X
n^2		X		
3^n	X			
5		X		

Jede Zeile wird nur dann gewertet, wenn sie vollständig richtig ist.

Aufgabe A4)

```
Funktion A(n)
a = 2
b = 2n^2
while b > 1
    a++
    c = n
    while c ≥ n
        b = b - a
        a = ⌊a/2⌋
        c = ⌊2n/c⌋
```

Innere Schleife: Abbruch wenn $c < n$

c immer auf n gesetzt

jede Iteration:

$c = n$

$c = \lfloor \frac{2n}{c} \rfloor = \lfloor \frac{2n}{n} \rfloor = 2$ <

$c = \lfloor \frac{2n}{c} \rfloor = \lfloor \frac{2n}{2} \rfloor = n$ <

$c = \lfloor \frac{2n}{c} \rfloor = \lfloor \frac{2n}{n} \rfloor = 2$ <

Endlosschleife, wenn $2 \geq n$

Nur 1 Iteration wenn $2 < n$

Äußere Schleife: Abbruch wenn $b = 1$

$a = 2$

$b = 2n^2$ ← case 1: $2 \geq n$ ∞ Iterationen

case 2: $2 < n$ 1 Iteration

case 1: n ist $\{2, 1, 0\}$

keine Lösung

→ case 2: n ist $\{3, 4, 5 \ldots \infty\}$

$b = 2n^2 > 1$

Und nur eine Iteration der inneren Schleife, somit:

→ Funktion A(n):

```
a = 2
b = 2n^2
while b > 1
    a++
    b = b - a
    a = ⌊a/2⌋
```

```
a = 2
b = 2n^2
while b > 1
    a = 3
    b = 2n^2 - 3
    a = 1
    ---
    a = 2
    b = 2n^2 - 3 - 2
    a = 1
    ---
    a = 2
    b = 2n^2 - 3 - 2 - 2
    a = 1
```

$b - 2 \cdot k = 1$ k = Iterationen

$2n^2 - 2k = 1$

$n^2 - k = \frac{1}{2}$

$-k = \frac{1}{2} - n^2$

$k = n^2 - \frac{1}{2}$

Conclusion:

Wenn $n \geq 3$ dann:

innere Schleife $\Theta(1)$

äußere Schleife $\Theta(n^2 - \frac{1}{2}) = \Theta(n^2)$] $\Theta(n^2)$

Wenn $n < 3$ dann:

innere Schleife wird endlos

äußere Schleife damit auch.

Funktion B(n):

$x = 1$

for $j \leftarrow 1$ bis $\lfloor \log_5 n \rfloor$ ← $5^{\log_5 n} = n$

$x = 4x$

$z = j$ ← $x = \lfloor \frac{4x}{2} \rfloor = 2x$

$x = \lfloor \frac{x}{2} \rfloor$ ←

} x bei Ausgabe irrelevant

for $j \leftarrow 1$ bis $\lfloor \log_2 n \rfloor$ ← $2^{\log_2 n} = 2$

$z--$

return z

Wenn man Code vereinfacht:

for $j \leftarrow 1$ bis $\lfloor \log_5 n \rfloor$

$z = j$ → z wird nicht aufsummiert, also nur letzte Zuordnung:

$z = \lfloor \log_5 n \rfloor$

for $j \leftarrow 1$ bis $\lfloor \log_2 n \rfloor$

$z--$

return z

→ for $(j = 1;\ j \leq \lfloor \log_2 n \rfloor;\ j++)$

Abbruch bei $j > \lfloor \log_2 n \rfloor$

$j + k \cdot 1 > \log_2 n$

$1 + k \cdot 1 > \log_2 n$

$k > \log_2 n - 1$ Iterationen

$\Theta(\log n)$

$z = z - 1 \cdot (\lfloor \log_2 n \rfloor - 1)$

$z = \lfloor \log_5 n \rfloor - \lfloor \log_2 n \rfloor - 1$

Aufgabe A-1)

```
Funktion A(n)
a = 2
b = 2n^2
while b > 1
    a++
    c = n
    while c >= n
        b = b - a
        a = ⌊a/2⌋
        c = ⌊2n/c⌋
```

Innere Schleife: Abbruch wenn $c < n$

c immer auf n gesetzt

jede Iteration:

$c = n$

$c = \lfloor \frac{2n}{c} \rfloor = \lfloor \frac{2n}{n} \rfloor = 2$

$c = \lfloor \frac{2n}{c} \rfloor = \lfloor \frac{2n}{2} \rfloor = n$

$c = \lfloor \frac{2n}{c} \rfloor = \lfloor \frac{2n}{n} \rfloor = 2$

Endlosschleife, wenn $2 \geq n$

Nur 1 Iteration wenn $2 < n$

äußere Schleife: Abbruch wenn $b = 1$

$a = 2$

$b = 2n^2$

case 1: $2 \geq n$ ∞ Iterationen

case 2: $2 < n$ 1 Iteration

case 1: n ist $\{2, 1, 0\}$

keine Lösung

→ case 2: n ist $\{3, 4, 5 \ldots \infty\}$

$b = 2n^2 > 1$

Und nur eine Iteration der inneren Schleife, somit:

→ Funktion A(n):

```
a = 2
b = 2n^2
while b > 1
    a++
    b = b - a
    a = ⌊a/2⌋
```

```
a = 2
b = 2n^2
while b > 1
    a = 3
    b = 2n^2 - 3
    a = 1
-----------
a = 2
b = 2n^2 - 3 - 2
a = 1
-----------
a = 2
b = 2n^2 - 3 - 2 - 2
a = 1
```

$b - 2 \cdot k = 1$ k = Iterationen

$2n^2 - 2k = 1$

$n^2 - k = \frac{1}{2}$

$-k = \frac{1}{2} - n^2$

$k = n^2 \cdot \frac{1}{2}$

Conclusion:

Wenn $n \geq 3$ dann:

innere Schleife $\Theta(1)$

äußere Schleife $\Theta\left(n^2 - \frac{1}{2}\right) = \Theta(n^2)$] $\Theta(n^2)$

Wenn $n < 3$ dann:

innere Schleife wird endlos

äußere Schleife damit auch.

Funktion B(n):

```
x = 1
for j ← 1 bis ⌊log5 n⌋        ← 5^(log5 n) = n
    x = 4 x                   ← x = ⌊4x/2⌋ = 2x
    z = j
    x = ⌊x/2⌋
for j ← 1 bis ⌊log2 n⌋        ← 2^(log2 n) = 2
    z--
return z
```

} x bei Ausgabe irrelevant

Wenn man Code vereinfacht:

```
for j ← 1 bis ⌊log5 n⌋
    z = j
for j ← 1 bis ⌊log2 n⌋
    z--
return z
```

→ z wird nicht aufsummiert, also nur letzte Zuordnung:

$z = \lfloor \log_5 n \rfloor$

→ for ($j = 1$; $j \leq \lfloor \log_2 n \rfloor$; j++)

Abbruch bei $j > \lfloor \log_2 n \rfloor$

$j + k \cdot 1 > \log_2 n$

$1 + k \cdot 1 > \log_2 n$

$k > \log_2 n - 1$ Iterationen

$\Theta(\log n)$

$z = z - 1 \cdot (\lfloor \log_2 n \rfloor - 1)$

$z = \lfloor \log_5 n \rfloor - \lfloor \log_2 n \rfloor - 1$

Landau-Symbole, Beispiele

1) $a_n = n^2 + 30n + 42 \in O(n^2)$

$$\left| \frac{n^2 + 30n + 42}{n^2} \right| = 1 + \underbrace{\frac{30n}{n^2} + \frac{42}{n^2}}_{\text{konvergiert gegen 0 und kann höchstens 30 bzw. 42 sein}} \leq 1 + 30 + 42 \leq 100$$

2) $a_n = 42n^2 \in O(n^2)$

$$\left| \frac{42n^2}{n^2} \right| = 42$$

3) $n^2 \in O(2^n)$

$$\left| \frac{n^2}{2^n} \right| \to 0 \quad \text{konvergiert gegen 0}$$

4) (Klausur Nachtragstest 2017, Wiederholung verfügbar)
Aufgabe A-1) b): ordnen Sie nach der Dominanz:

$4n \log n \leftarrow n \cdot (\log n^4) \in O(n \log n)$

$n^3 \leftarrow n^3 - n^2 - n \in O(n^3)$

$n \log 4 \leftarrow \log(4^n) \in O(n)$

$\log 4$ ist const

$\left(\frac{1}{5000}\right)^n$

$n^{\frac{n}{2}} \leftarrow \sqrt{n^n}$

$\left(\frac{1}{4}\right)^n$

kleiner werdende Zahl hoch n | kleine Zahl hoch n

$$\left(\frac{1}{4}\right)^n << \left(\frac{1}{5000}\right)^n <<$$

linear | $n \log n$ | poly

$$\log(4^n) << n(\log n^4) << n^3 - n^2 - n$$

$$<< \text{exponentiell } \sqrt{n^n}$$

Aufgabe A1) a) (Sommersemester, Nachtragstest → nicht verfügbar)

Laufzeit von Funktion A:

```
Funktion A(n)
a = n/10
if a ≤ 10
    return
else
    Funktion A(a)
```

Äquivalent zu:

```
a = n/10
while (a > 10)
    a = a/10
return
```

Abbruch wenn $a = 10$:

$a = \frac{n}{10}$

$a \cdot \left(\frac{1}{10}\right)^k = 10$

$\left(\frac{1}{10}\right)^k = \frac{10}{a}$

$k \log\left(\frac{1}{10}\right) = \log\left(\frac{10}{a}\right)$

$k = \frac{\log\left(\frac{10}{a}\right)}{\log\left(\frac{1}{10}\right)}$

Wobei $a = \frac{n}{10}$

$k = \frac{\log\left(\frac{10}{1} \cdot \frac{10}{n}\right)}{\log\left(\frac{1}{10}\right)}$

ist $\Theta(\log n)$

(Eigentlich $\log_{10} n$)

Rückgabewert von Funktion B:

```
Funktion B(n)
n = 2n
x = 1
a = 1
for j = 1 bis n
    x = x + a x
    z = j
    a = -a
while j ← 1 ≤ x
    z ← z + 1
return z
```

$n = 2n$ dadurch:

$1 + k \cdot 1 = 2n$

$k = 2n - 1$ Iterationen

mit Sicherheit $x = 0$

$z = j$ in letzter Iteration

also $z = n\ (2n)$

$a = a \cdot (-1)^{2n-1} = -a$

Am Ende der for-Schleife

$j = \text{false}$ weil $1 \neq 0$

$z = 2n$

↳ fällt weg

Aufgabe A1) b)

$f : \mathbb{N}^+ \to \mathbb{R}$

$$f(n) = \begin{cases} \frac{1}{2 \cdot \log(n)} - 2n + 2^n & \text{wenn } n > 42 \text{ und } 2 \nmid n \\ \left(\frac{7}{2}\right)^n + \frac{3n^2}{4} - \log(n) & \text{wenn } n \leq 42 \\ n^{-3} \cdot 3^n - (n+1)^3 + \log(2^n) & \text{sonst} \end{cases}$$

Bei Asymptotischer Laufzeitanalyse nur $\lim_{n \to \infty}$ beobachtet, deshalb 2. Zeile irrelevant.

(kein Grenzwert)

Landau-Symbole

groß O für $n \to \infty$ $a_n \in O(b_n)$ wenn $\exists C > 0$, sodass

$\left|\frac{a_n}{b_n}\right| \leq C$ für $n \geq N$ $(n \geq 0)$

Omega Ω für $n \to \infty$ $a_n \in \Omega(b_n)$ wenn $\exists C > 0$, sodass

$\left|\frac{b_n}{a_n}\right| \geq C$ für $n \geq N$ $(n \geq 0)$

Theta Θ für $n \to \infty$ $a_n \in \Theta(b_n)$ wenn $\exists c_1 > 0 \wedge \exists c_2 > 0$, sodass

$c_1 |b_n| \leq |a_n| \leq c_2 |b_n|$ für $n \geq N$

> Also zugleich

$a_n \in O(b_n)$ und $b_n \in O(a_n)$

Das bedeutet

$a_n \in \Theta(b_n) \iff b_n \in \Theta(a_n)$

Dominanz

$O(1) \subseteq O(\log n) \subseteq O(n) \subseteq O(n \log n) \subseteq O(n^2) \subseteq O(2^n) \subseteq O(n!) \subseteq O(n^n)$

Also vereinfacht:

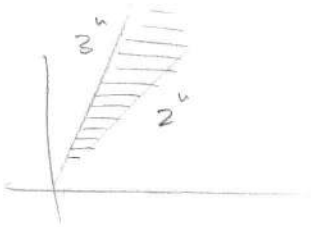
dominiert

$$f(n) = \begin{cases} \frac{1}{2\log(n)} - 2n + 2^n & \text{wenn } n > 42 \wedge 2 \nmid n \\ \frac{1}{n^3} \cdot 3^n + \underbrace{(n+1)^3}_{O(n^3)} + \underbrace{\log(2^n)}_{n\log(2)} & \text{sonst} \\ & (\text{wenn } n > 42 \wedge 2 \nmid n) \end{cases}$$

$n\log(2) \in O(n)$

$$O(n) \ll O(n^3) \ll O\left(\frac{1}{n^3} \cdot 3^n\right)$$

$f(n)$



Bei Tabelle:

$e^n = (2{,}7\ldots)^n$ wäre einerseits untere, andererseits obere Schranke → keine Schranke

n^2 ——— Untere Schranke

3^n ——— obere Schranke

5 ——— Untere Schranke

Aufgabe A2)a)

	1	2	3	4	5
V	E	B	A	C	D
W	A	D	C	B	E
X	A	C	D	B	E
Y	E	C	D	A	B
Z	C	B	A	D	E

	1	2	3	4	5
A	Z	X	Y	V	W
B	X	Y	V	Z	W
C	W	V	X	Z	Y
D	Z	X	Y	W	V
E	Y	X	V	W	Z

1. Beispiel

Matchings

V - B → instabil: siehe Z - E

W - C ✓

X - D ✓

Y - A → instabil: A steht weiter oben auf der Liste von X, Z

Z - E → instabil: E steht weiter oben bei V, Y

2. Beispiel

V - B ✓

W - A → instabil: A steht weiter oben bei X

X - D ✓

Y - E ✓

Z - C → instabil: C steht weiter oben bei X
(aber A steht noch höher als C bei X)

3. Beispiel

V - B ✓

W - D ✓

X - C ✓ Stabil

Y - E ✓

Z - A ✓

Rep 1 2018

Aufgabe A2: Stable-Matching, Starker Zusammenhang **(12 Punkte)**

a) (8 Punkte)

	1.	2.	3.	4.	5.
V	E	B	A	C	D
W	A	D	C	B	E
X	A	C	D	B	E
Y	E	C	D	A	B
Z	C	B	A	D	E

	1.	2.	3.	4.	5.
A	Z	X	Y	V	W
B	X	Y	V	Z	W
C	W	V	X	Z	Y
D	Z	X	Y	W	V
E	Y	X	V	W	Z

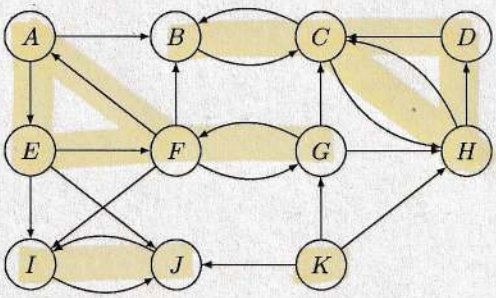
Jede Zeile der nachfolgenden Tabelle enthält ein Matching. Stellen Sie jeweils fest, ob dieses in Bezug auf obige Präferenzlisten stabil oder instabil ist. Wenn ein Matching instabil ist, dann zeigen sie das, indem Sie ein instabiles Paar angeben.

Matching	stabil	instabil	Instabiles Paar
V-B, W-C, X-D, Y-A, Z-E		X	V-E
V-B, W-A, X-D, Y-E, Z-C		X	X-A, X-C
V-B, W-D, X-C, Y-E, Z-A	X		

Rep 1 2018

b) (4 Punkte) Partitionieren Sie den folgenden Graphen in maximale Teilgraphen, sodass jeder Teilgraph stark zusammenhängend ist. Tragen sie zusammengehörige Knoten jeweils in eine Zeile der nachfolgenden Tabelle ein.

(Wenn Sie nicht alle Zeilen benötigen, lassen sie die übrigen einfach frei.)



Teilgraph	Enthaltene Knoten
1	K
2	I, J
3	A, E, F, G, H
4	B, C, D, H
5	
6	

Rep 2018 – Teil 1

Aufgabe A3: Graphen **(20 Punkte)**

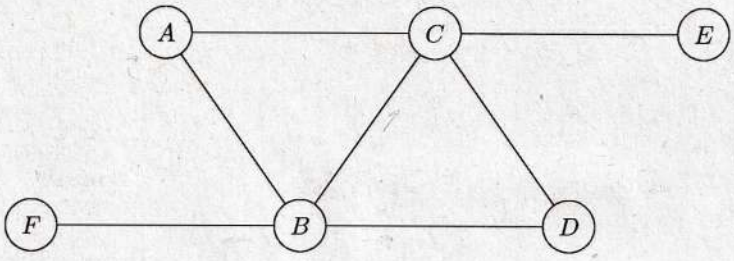
a) (12 Punkte) Gegeben ist der folgende Algorithmus für schlichte, ungerichtete, zusammenhängende Graphen $G = (V, E)$:

Hinweis: Das Array `Markiert` ist global.

```
WasBinIch(G):
X ← ∅
foreach Knoten v ∈ V
    Nachbar[u] ← false für alle Knoten u ∈ V
    Markiert[u] ← false für alle Knoten u ∈ V
    foreach Kante (v, u) inzident zu v
        Nachbar[u] ← true
        q ← u
    Hilfsfunktion(G − {v},q)
    foreach Knoten u ∈ V
        if Nachbar[u] und !Markiert[u]
            X ← X ∪ {v}
return X
```

```
Hilfsfunktion(G,q):
Markiert[q] ← true
foreach Kante (q, b) inzident zu q
    if !Markiert[b]
        Hilfsfunktion(G,b)
```

Weiters ist ein Graph $G = (V, E)$ gegeben:



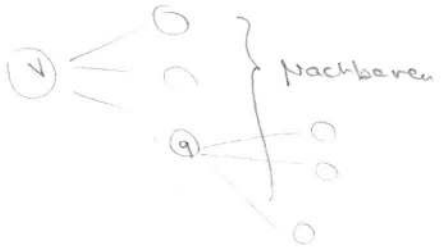
- (3 Punkte) Wenden Sie den Algorithmus *WasBinIch* auf den gegebenen Graphen G an. Geben Sie die vom Algorithmus retournierte Knotenmenge X an.

$X = \{C, B\}$

```
WasBinIch (G):
X = ∅

foreach v∈V
    Nachbar [u] = false  für ∀ Knoten
    Merkiert [u] = false für ∀ Knoten

    foreach Nachbar u von v
        Nachbar [u] = true
        q = u

    Hilfsfunktion (G\{v}, q)

    foreach Knoten u∈V
        if Nachbar ∧ !markiert
            X ← X ∪ {v}

return X
```

reset

Markiert alle „Nachbaren von v"
Speichert nur letzten Eintrag als q



Markiert alle Knoten die ein Weg zu q haben \ {v} als markiert"

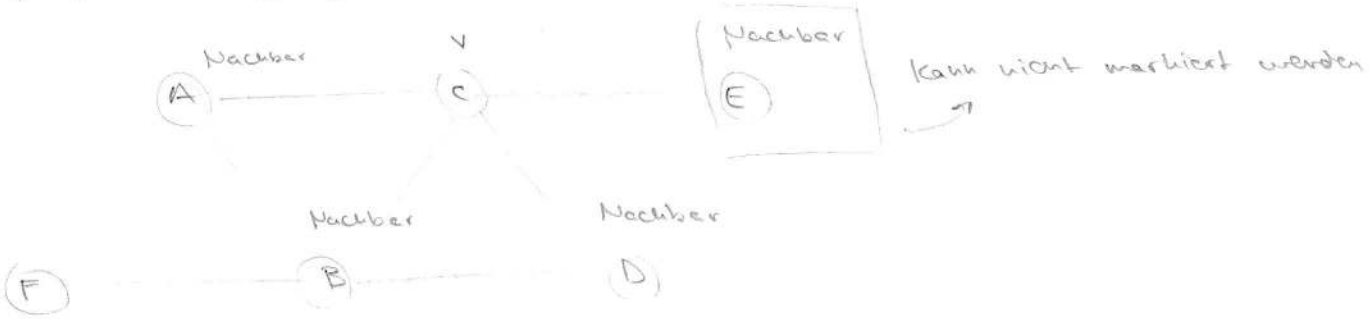
Wenn es nicht erreichbare Nachbaren gibt, nehme v (Anfangsknoten) in die Lösungsmenge

```
Hilfsfunktion (G, q)
Markiert [q] = true
foreach Nachbar b von q
    if (b != merkiert)
        Hilfsfunktion (G, b)
```

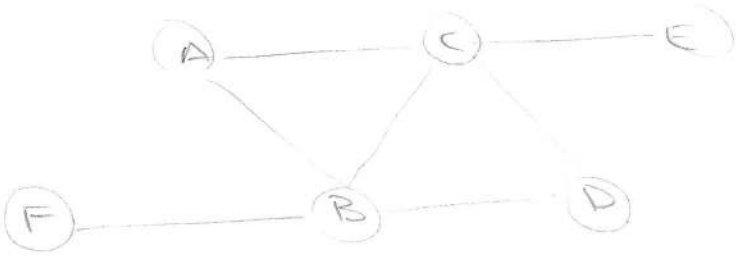
→ markiert Input

→ merkiert Nachbaren vom Input wenn noch nicht markiert.

Beispiele aus Graph:



Lösungsmenge X:



$X = \{C, B\}$

- (2 Punkte) Welchen, aus der Vorlesung bekannten, Algorithmus verwendet *WasBinIch*?

Hilfsfunktion = DFS Depth first search

- (3 Punkte) Beschreiben Sie **kurz** die Funktion des Algorithmus und charakterisieren Sie die Knoten, die in der retournierten Menge X enthalten sind.

Artikulations / Gelenkspunkte:
Graph zerfällt in mehrere Partitionen, wenn diese entfernt werden

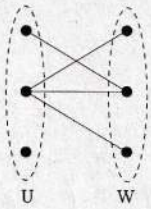
- (4 Punkte) Geben Sie die Laufzeit von *WasBinIch* in Θ-Notation in Abhängigkeit der Anzahl der Knoten n und der Kanten m des Eingabegraphen an. Begründen Sie **kurz**, wie sich diese Laufzeit ergibt.

$\Theta(n^2 + n \cdot m)$

weil n. Tiefensuche mit DFS = $O(n+m)$

$n(n+m) = n^2 + nm$

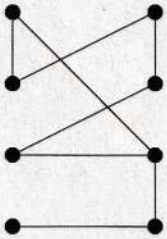
b) (8 Punkte) Ein ungerichteter Graph $G = (V, E)$ heißt *bipartit*, wenn man die Menge V in zwei disjunkte Teilmengen U und W so aufspalten kann, dass für alle Kanten $(u, w) \in E$ gilt: $u \in U$ und $w \in W$. Die untenstehende Abbildung stellt einen bipartiten Graphen dar. Die Teilmengen U und W sind gekennzeichnet.



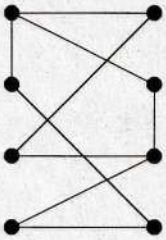
Schreiben Sie **unter** jeden der nachfolgenden Graphen „**Ja**“, wenn er bipartit ist und „**Nein**“ sonst. Beweisen Sie diese Angabe folgendermaßen:

- Ist ein Graph **nicht bipartit**, dann markieren Sie einen minimalen (bzgl. der Anzahl der enthaltenen Knoten und Kanten) Teilgraph der nicht bipartit ist.
- Ist ein Graph **bipartit**, dann teilen Sie die Knoten in zwei disjunkte Teilmengen (gemäß obiger Beschreibung, z.B. durch Färben oder Einkreisen).

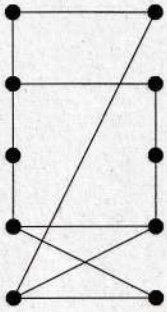
A



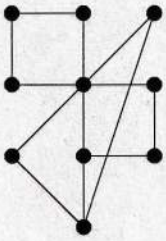
B



C



D



Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.813 Algorithmen und Datenstrukturen 1 VU 6.0

2. Test SS 2017

1. Juni 2017

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	16	18	16	50
Erreichte Punkte:				

Viel Erfolg!

Aufgabe A1: Hashtabellen **(16 Punkte)**

Fügen Sie die folgenden Zahlen in die jeweiligen Hashtabellen ein, indem Sie die angegebenen Hashfunktionen und Strategien für die Kollisionsbehandlung benutzen.

a) (4 Punkte)

Einzufügende Zahl: 16
Kollisionsbehandlung: Quadratisches Sondieren mit $c_1 = 1$ und $c_2 = 1$

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

Hashtabelle:

0	1	2	3	4	5	6
		23		32	40	

b) (4 Punkte)

Einzufügende Zahl: 16
Kollisionsbehandlung: Double Hashing **ohne** die Verbesserung nach Brent

Hashfunktionen:
$h_1(k) = k \bmod 7$
$h_2(k) = (k \bmod 6) + 1$

Hashtabelle:

0	1	2	3	4	5	6
		23		32	40	

c) (4 Punkte)

Einzufügende Zahl: 79
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:
$h_1(k) = k \bmod 7$
$h_2(k) = (k \bmod 6) + 1$

Hashtabelle:

0	1	2	3	4	5	6
		44		32	40	

d) (4 Punkte)

Gegeben sind im Folgenden mehrere Hashverfahren. Geben Sie zu jedem an, ob es gut funktionieren würde, oder ob es zu Problemen kommen könnte. Begründen Sie ihre Antworten.

I) (2 Punkte)

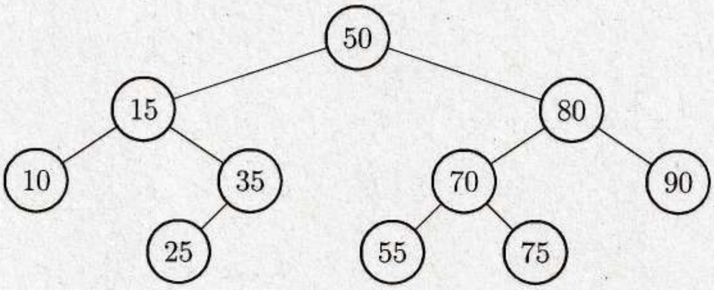
Multiplikationsmethode
Tabellengröße $m = 29$
Faktor $A = 3.25$

II) (2 Punkte)

Divisions-Rest-Methode mit Double Hashing
Tabellengröße $m = 40$
$h_1(k) = k \bmod 40$
$h_2(k) = (k \bmod 36) + 1$

Aufgabe A2: Bäume **(18 Punkte)**

a) (10 Punkte) Gegeben sei folgender AVL-Baum:

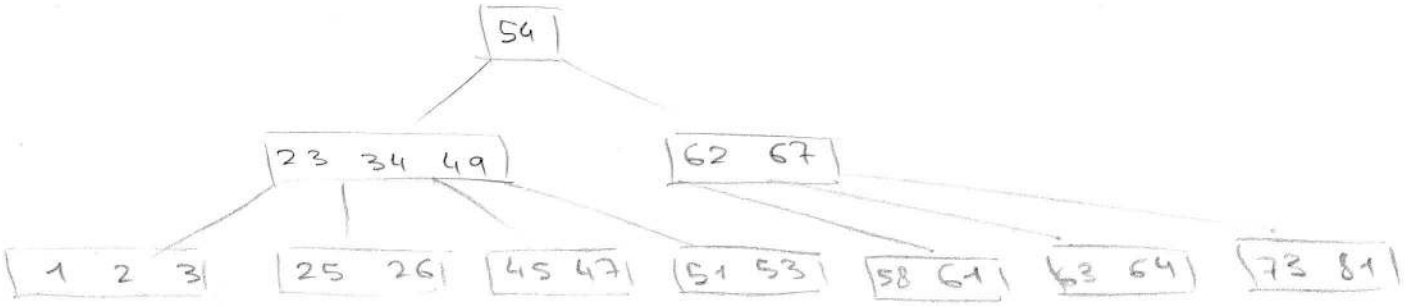


Teilen Sie alle noch nicht vorhandenen ganzen Zahlen aus dem Intervall [0, 100] in Teilintervalle auf, sodass jede Zahl aus demselben Teilintervall an der gleichen Position in dem gegebenen Baum eingefügt werden würde. Tragen Sie nun diese Intervalle zusammen mit der entsprechenden Einfügestelle sowie die Art der Reorganisation, welche nach dem Einfügen einer Zahl in einem solchen Intervall ausgeführt werden muss, in die folgende Tabelle ein. Die erste Zeile in der Tabelle ist bereits exemplarisch ausgefüllt.

Intervall	Einfügestelle	Art der Reorganisation		
		keine	einfache Rotation	doppelte Rotation
0–9	links von 10	✓		

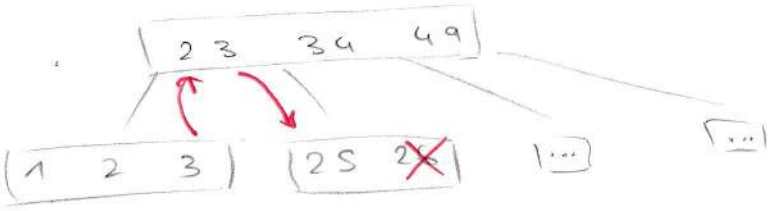
Ordnung 5 — max 5 Kinder, min $\lceil \frac{5}{2} \rceil = 3$ Kinder
— max 4 keys, min $\lceil \frac{4}{2} \rceil = 2$ keys

Aufgabe 2:
Beispiel b)

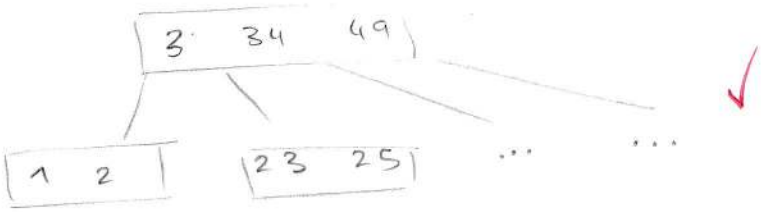


t.lösche(26):

- 26 ist in der untersten Ebene
- Wenn man 26 entfernt hat die Node < 2 keys und muss deshalb von Nachbaren ausleihen
- rechter Nachbar kann nicht ausleihen
- Linker Nachbar kann ausleihen



1. 3 kommt zu elterm Node
2. 23 kommt zum child Node



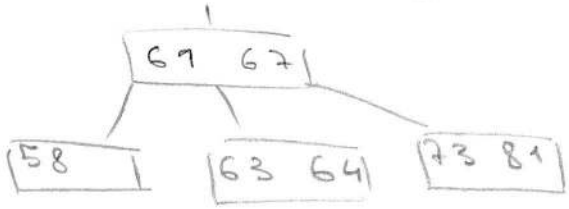
t. Lösche (62)

- 62 ist nicht in der untersten Ebene
- ohne 62 hätte Node < 2 keys
- deshalb muss nachdem 62 entfernt wurde ein Ersatzschlüssel hin

  option 1: 61

  option 2: 63

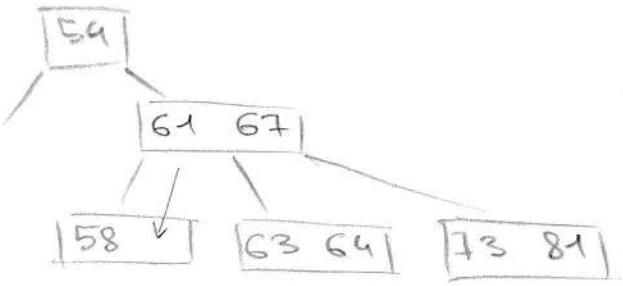
- wir entscheiden uns für 61



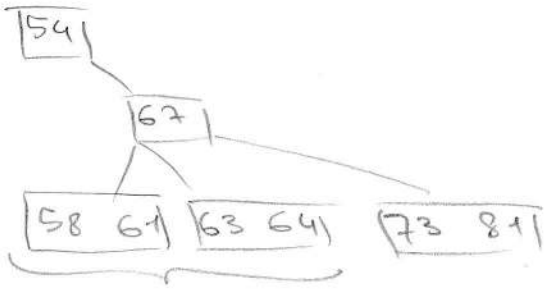
Jetzt muss ident zum vorherigen Beispiel ein Nachbar geliehen werden, aber die Nachbaren haben nicht genug keys zum verschenken.

- wir müssen vom Elternknoten leihen

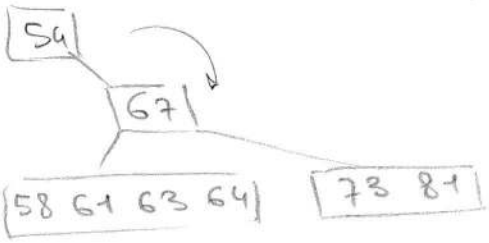
[Abbruch des Streams]



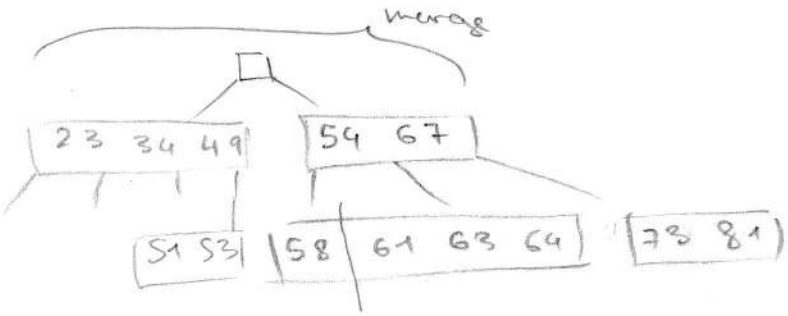
Eltern borgen key

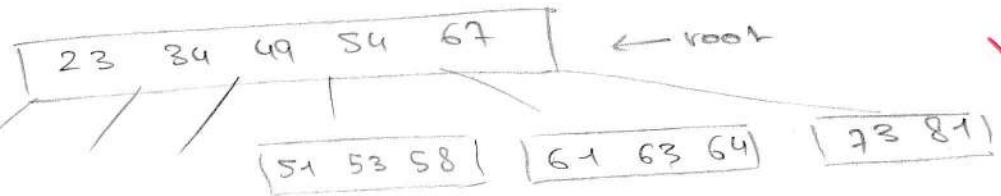


Merge!



Eltern borgen key





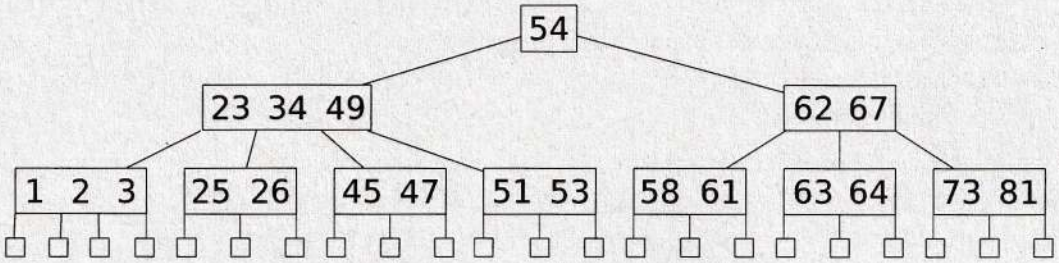
kontrolliert mit
guerley trees, stimmt

Rep 1 - 2019

b) (8 Punkte) Gegeben sei der nachfolgende **B-Baum der Ordnung 5**.

- Entfernen Sie aus dem angegebenen Baum den Schlüssel 26.
- Entfernen Sie aus dem angegebenen Baum den Schlüssel 62.

Geben Sie dabei alle Zwischenschritte an! Sie können sich bei der Angabe der Zwischenschritte auf den Teilbaum beschränken, der sich geändert hat.



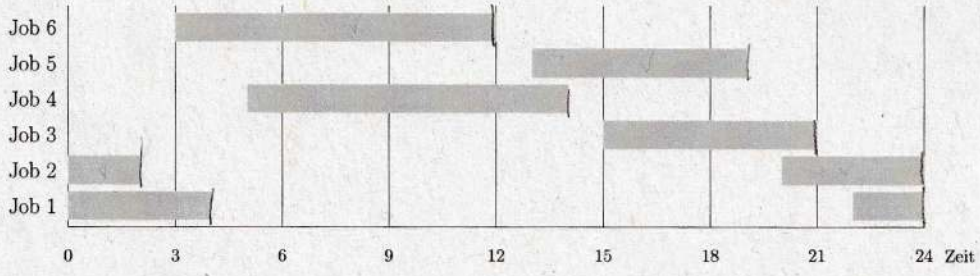
Rep 2018 – Teil 1

Aufgabe A3: Greedy **(16 Punkte)**

Betrachten Sie die folgende Modifikation des Interval-Scheduling-Problems. Ein Prozessor operiert 24 Stunden am Tag, jeden Tag. Leute stellen Anfragen, um tägliche Jobs (Programme) auf dem Prozessor auszuführen. Jeder solcher Jobs hat eine Startzeit und eine Endzeit (welche z.B. in Sekunden nach Mitternacht angegeben werden kann). Wird der Job zum Ausführen akzeptiert, dann muss er täglich von der angegeben Startzeit bis zur angegebenen Endzeit ausgeführt werden. Beachten Sie hier, dass ein Job auch vor Mitternacht beginnen und erst am nächsten Tag enden kann. Weiterhin darf jeder Job höchstens 24 Stunden dauern.

Entwerfen Sie einen Algorithmus mit Laufzeit $O(n^2 \log n)$, welcher ähnlich zu dem aus der Vorlesung bekannten Greedy-Algorithmus für das normale Interval-Scheduling-Problem, eine größenmässig maximale Teilmenge von Jobs findet, welche kontinuierlich (über mehrere Tage hinweg) ohne Überschneidungen ausgeführt werden kann.

Die folgende Abbildung zeigt eine mögliche Instanz dieses Problems mit 6 Jobs. Die Beispielinstanz hat zwei optimale Lösungen: die Teilmenge die aus den Jobs 1, 3 und 4 besteht sowie die Teilmenge die aus den Jobs 2, 5 und 6 besteht.



Wir sagen ein Job ist "einfach", falls er vollständig innerhalb eines Tages ausgeführt werden kann. Im obigen Beispiel sind die Jobs 3–6 einfache Jobs, die Jobs 1 und 2 hingegen nicht.

a) (1 Punkt) Geben Sie ein einfaches Kriterium an, welches es Ihnen erlaubt zu entscheiden, ob ein Job einfach ist:

Startzeit < Endzeit

Schwer: Startzeit > Endzeit

b) (2 Punkte) Wir sagen eine Instanz des modifizierten Interval-Scheduling-Problems ist einfach, falls sie nur aus einfachen Jobs besteht. Stimmt es, dass der Ihnen aus der Vorlesung bekannte Greedy-Algorithmus für das Interval-Scheduling-Problem, die richtige Lösung für jede einfache Instanz liefert? Begründen Sie Ihre Antwort!

Ja! Dann sie überprüft, ob

max Endzeit (L) < Startzeit neuer Job

c) (1 Punkt) Wieviele nicht einfache Jobs kann eine optimale Lösung des modifizierten Interval-Scheduling-Problems maximal enthalten?

1x weil sie sich sonst zu Mitternacht überschneiden

d) (12 Punkte) Schreiben Sie nun in detailiertem Pseudocode eine Funktion

`Menge findeMaximaleMenge( Menge einfJobs, Menge schwJobs )`,

welche eine optimale Lösung für das modifizierte Interval-Scheduling-Problem zurückgibt und eine Laufzeit von höchstens $O(n^2 \log n)$ für eine gegebene Menge von n Jobs aufweist. Die beiden Parameter `einfJobs` und `schwJobs` der Funktion enthalten dabei die Menge aller einfachen bzw. die Menge aller nicht einfachen Jobs der Instanz.

Sie können dafür die Ihnen aus der Vorlesung bekannte Funktion, nachfolgend als `Greedy` bezeichnet, in der Form `Menge Greedy(Menge J)` als gegeben voraussetzen, welche für eine gegebene Menge J von Jobs, eine optimale Teilmenge für das (unmodifizierte) Interval-Scheduling-Problem zurückgibt und eine Laufzeit von $O(|J| \log |J|)$ aufweist.

Die Datenstruktur `Menge` einer Menge m unterstützt die folgenden Operationen:

- `m.size()` gibt die Größe der Menge m zurück.
- `m.add( Job j )` fügt den Job j zur Menge m hinzu.

Desweiteren können Sie alle Elemente der Menge m mit der Anweisung `foreach j ∈ m` durchlaufen. Auf die Startzeit und Endzeit eines Jobs j können Sie mit `j.start` und `j.end` zugreifen.

1. Idee: Man überprüft

X max Endzeit (Lösungsmenge) < Startzeit neuer Job UND
min Startzeit (Lösungsmenge) > Endzeit neuer Job

Problem

→ Start zu früh, beansprucht zu viel Zeit

2. Idee: Anwendung des einfachen Greedy-Algorithmus mit höchstens einem schweren Job

✓ (siehe Rückseite)

Konzept:

Case 1: Einfache Jobs besser als gefundene schwere Jobs

Case 2: Gefundene schwere Jobs besser als einfache

Case 3: Keine schweren Jobs gefunden → bleibt 0
(weil sie zu viel Platz einnehmen → Greedy = 0)

Menge findeMaximaleMenge (einfJobs, schwJobs)

nur einfach = |Greedy (einfJobs)|

mit schwer = 0

i' = 0

for each $j_i \in$ schwJobs $\quad O(n^2 \log n)$

firstpart = [0; j_i.end] Job

lastpart = [j_i.start; 24] Job

mit j_i = |Greedy (einfJobs ∪ firstpart ∪ lastpart)|

if (mit $j_i \geq$ mit schwer)] update max wenn gefunden

mit schwer = mit j_i

i' = i von j_i

if (nur einfach > mit schwer)

return einfJobs

else

return einfJobs ∪ {$j_{i'}$}

↑ von oben

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.815 Algorithmen und Datenstrukturen 2 VU 3.0

Nachtragstest SS 2016

gelöst

5. Oktober 2016

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	15	21	14	50
Erreichte Punkte:				

Viel Erfolg!

~~A1, A2, A3~~

A2 – A3 im Rep SS17 besprochen

i=4 1:>1 max{0, π[0]+4} 1:>2 max{6, π[2-1]+4} 1:>3 max{8, π[3-1]+4}
1:>4 max{10, π[4-1]+4} 1:>5 max{14, π[5-1]+4} 1:>6 max{16, π[6-1]+4}
1:>7 max{18, π[7-1]+4, 1:>8 max{18, π[8-1]+4}

Aufgabe A1: Dynamische Programmierung **(15 Punkte)**

Gegeben sei folgende Instanz des Rucksackproblems mit Kapazität $G = 8$ und Gegenständen $\{A, B, C, D\}$ mit den in der Tabelle genannten Gewichten beziehungsweise Werten.

i=3 3:>3 max{6, π[3-3]+8}
3:>4 max{10, π[4-3]+8}
3:>5 max{10, π[5-3]+8}
3:>6 max{16, π[6-3]+8}
3:>7 max{16, π[7-3]+8}
3:>8 max{16, π[8-3]+8}

i=2 2:>2 max{0, π[2-2]+6}
2:>3 max{0, π[3-2]+6}
2:>4 max{10, π[4-2]+6}
2:>5 max{10, π[5-2]+6}
2:>6 max{10, π[6-2]+6}

Gegenstand	A	B	C	D
Gewicht	4	2	3	1
Wert	10	6	8	4

a) (10 Punkte)

Wenden Sie das in der Vorlesung vorgestellte dynamische Programm für das Rucksackproblem auf die gegebenen Gegenstände an. Tragen Sie in den **weißen** Zellen der unten stehenden Tabelle die Werte aller Teillösungen ein, die der Algorithmus berechnet.

G+1, g

i=1, g=0 4>0
4:>4 max{0, π[4-4]+4}
4:>5 max{0, π[5-4]+4}

i

n+1

	0	1	2	3	4	5	6	7	8	
$\emptyset$	0	0	0	0	0	0	0	0	0	$w_0=0$ $p_0=0$
$\{A\}$	0	0	0	0	10	10	10	10	10	$w_1=10$ $p_1=4$
$\{A, B\}$	0	0	6	6	10	10	16	16	16	$w_2=6$ $p_2=2$
$\{A, B, C\}$	0	0	6	8	10	14	16	18	18	$w_3=8$, $p_3=3$
$\{A, B, C, D\}$	0	4	6	10	12	14	18	20	22	$w_4=4$, $p_4=1$

b) (3 Punkte)

In der Vorlesung wurde auch der Algorithmus „Find-Solution“ zum Finden der Lösung besprochen. Markieren Sie in den **grauen** Zeilen genau diejenigen Zellen, die der Algorithmus „Find-Solution“ während seiner Ausführung ausliest.

c) (2 Punkte)

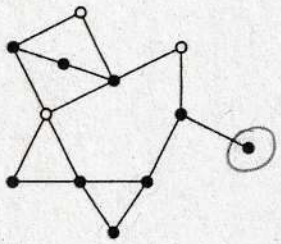
Geben Sie die Menge der für die optimale Lösung ausgewählten Gegenstände an.

Gegenstände 1, 3 und 4 werden ausgewählt

Find-Solution: i=5, k=8 22≠18, A=4, k=8-1=7, i=4,
i=4 k=7 18≠16, A=3 k=7-3=4, i=3
i=3 k=4 10=10 i=2
i=2 k=4 10≠0 A=1, k=4-4=0 i=2

Aufgabe A2: Lokale Suche, Branch and Bound, Approximation (21 Punkte)

Betrachten Sie den folgenden Graphen G und das durch die schwarzen Knoten markierte Vertex Cover W.



a) (3 Punkte)

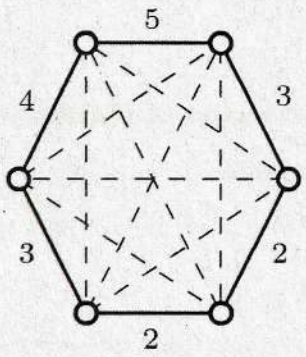
Führen Sie einen Schritt der Lokalen Suche mittels Nachbarschaftsstruktur für VERTEX COVER aus, wie in der Vorlesung vorgestellt. Markieren Sie eindeutig welcher Knoten dabei aus W entfernt wird.

b) (3 Punkte)

Wie viele verschieden Möglichkeiten gibt es einen Knoten aus W zu entfernen?

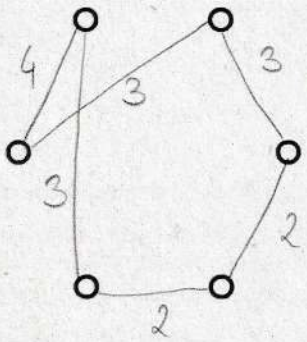
4

Betrachten Sie die folgende Instanz des TSP Problems und die durch die dicken äußeren Kanten markierte Tour. Alle gestrichelten Kanten haben das Gewicht 3.



c) (5 Punkte)

Führen Sie einen Schritt der 2-Opt Lokalen Suche durch, der die größtmögliche Verbesserung erzielt. Zeichnen Sie die resultierende Tour in die folgende Abbildung ein.



d) (2 Punkte)

Welches Gewicht hat Ihre oben eingezeichnete Tour? 17

4+3+3+3+2+2 = 17

e) (4 Punkte)

Welche der folgenden Aussagen über Branch-and-Bound Algorithmen sind korrekt?

Kreuzen Sie genau diejenigen Antworten an, die eine wahre Aussage darstellen. Bei einem Fehler (falsches oder fehlendes Kreuz) wird noch 1 Punkt vergeben, bei zwei oder mehr Fehlern werden 0 Punkte für diese Aufgabe vergeben.

- [x] In einem Maximierungsproblem kann man die Betrachtung eines Teilproblems abbrechen, wenn dessen lokale obere Schranke kleiner ist als die aktuelle globale untere Schranke.
- [x] Der Wert einer gültigen Lösung in einem Minimierungsproblem ist eine globale obere Schranke.
- [ ] Die asymptotische worst-case Laufzeit eines Branch-and-Bound Algorithmus ist immer polynomiell in der Eingabegröße. Kann sehr groß werden
  worst-case kann exponentiell sein
- [ ] Wenn in einem Minimierungsproblem eine gültige Lösung gefunden wird, deren Wert kleiner ist als die bisherige globale obere Schranke, hat man eine optimale Lösung gefunden und kann das Verfahren abbrechen.

f) (4 Punkte)

Welche der folgenden Aussagen über Approximationsalgorithmen sind korrekt?

Kreuzen Sie genau diejenigen Antworten an, die eine wahre Aussage darstellen. Bei einem Fehler (falsches oder fehlendes Kreuz) wird noch 1 Punkt vergeben, bei zwei oder mehr Fehlern werden 0 Punkte für diese Aufgabe vergeben.

AD2-04 S4

- [ ] Algorithmus A ist ein ε-approximativer Algorithmus für ein Maximierungsproblem wenn für ihn gilt, dass er für jede Probleminstanz eine Lösung mit einem Wert zurück liefert, der den optimalen Lösungswert um den Faktor $1/\varepsilon$ nicht unterschreitet. $0 \leq \varepsilon \leq 1$
- [x] Algorithmus A ist ein ε-approximativer Algorithmus für ein Maximierungsproblem wenn für ihn gilt, dass er für jede Probleminstanz eine Lösung mit einem Wert zurück liefert, der den optimalen Lösungswert um den Faktor ε nicht unterschreitet.

- [ ] Für das symmetrische Traveling Salesman Problem mit beliebiger Kostenmatrix besitzt die Spanning-Tree-Heuristik eine Gütegarantie von 2.
  Gibt hier Gütegarantie von 2 für das metrische, aber nicht symmetrische TSP!

AD2-04 S9

- [x] Für das Problem ein kleinstes Vertex Cover für einen gegebenen Graphen zu finden gibt es einen Approximationsalgorithmus mit Gütegarantie 3.
  Es gibt einen mit 2 und deshalb auch mit 3. Hier ist nach oben eine minimale Güte garantiert

Aufgabe A3: Interval Scheduling **(14 Punkte)**

Betrachten Sie folgende Problemdefinition und die darunter angegebene Instanz.

Problem (gewichtetes Interval Scheduling mit Strafkosten). *Gegeben sei eine Menge von n Jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ mit folgenden Eigenschaften:*

- *Job J_i startet zum Zeitpunkt s_i und endet zum Zeitpunkt $f_i > s_i$.*
- *Job J_i hat den Wert $w_i > 0$, wenn er ausgeführt wird, und die Strafkosten $t_i \geq 0$, wenn er nicht ausgeführt wird.*
- *Zwei Jobs J_i und J_k sind* kompatibel, *wenn sie sich nicht überlappen, also $f_i \leq s_k$ oder $f_k \leq s_i$ gilt.*

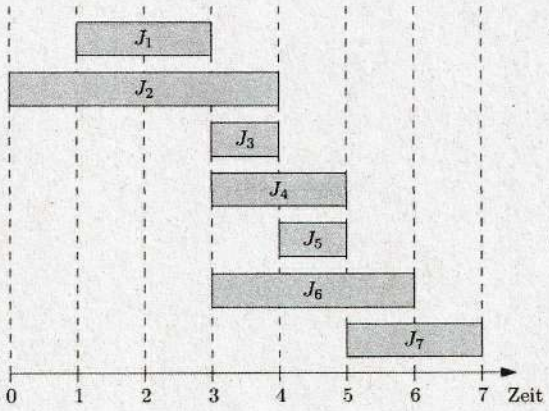
Gesucht ist eine Teilmenge $S \subset \mathcal{J}$ von paarweise kompatiblen Jobs, mit maximalem Gesamtgewinn

$$G(S) = \sum_{J_i \in S} w_i - \sum_{J_i \in \mathcal{J} \setminus S} t_i,$$

also dem Wert der ausgeführten Jobs minus den Strafkosten der nicht ausgeführten Jobs.

Hinweis: Wenn alle Strafkosten $t_i = 0$ $(i = 1, \ldots, n)$ sind, erhält man genau das aus der Vorlesung bekannte Problem *gewichtetes Interval Scheduling.*

Job	Startzeit	Ende	Wert	Strafkosten
J_1	$s_1 = 1$	$f_1 = 3$	$w_1 = 7$	$t_1 = 10$
J_2	$s_2 = 0$	$f_2 = 4$	$w_2 = 10$	$t_2 = 0$
J_3	$s_3 = 3$	$f_3 = 4$	$w_3 = 2$	$t_3 = 0$
J_4	$s_4 = 3$	$f_4 = 5$	$w_4 = 6$	$t_4 = 5$
J_5	$s_5 = 4$	$f_5 = 5$	$w_5 = 5$	$t_5 = 2$
J_6	$s_6 = 3$	$f_6 = 6$	$w_6 = 12$	$t_6 = 7$
J_7	$s_7 = 5$	$f_7 = 7$	$w_7 = 5$	$t_7 = 5$



J1 & J2 sind nicht kompatibel weil sie sich überlappen

J1 & J3 sind kompatibel

a) (2 Punkte) Erklärung auf nächster Seite

Berechnen Sie für die oben angegebene Instanz, deren Jobs nicht-absteigend nach dem Ende f_j ($j = 1, \ldots, 7$) sortiert sind, die Werte $p(j)$ für $j = 1, \ldots, 7$. Der Wert $p(j)$ gibt – wie in der Vorlesung – den größten Index $i < j$ an, so dass Job J_i kompatibel zu J_j ist. Füllen Sie die Tabelle aus. Existiert kein solcher Job J_i sei der Wert als $p(j) = 0$ definiert.

$p(1)$	$p(2)$	$p(3)$	$p(4)$	$p(5)$	$p(6)$	$p(7)$
0	0	1	1	3	1	5

b) (6 Punkte)

Ergänzen Sie die beiden fehlenden Zeilen in nachfolgendem Pseudocode, so dass nachdem der Algorithmus ausgeführt wurde, der Eintrag $M[j]$ für $j = 1, \ldots, n$ den Gesamtgewinn eines optimalen Schedules der Jobs $J_1, J_2, \ldots, J_j$ enthält. Orientieren Sie sich an dem Bottom-Up Verfahren für *gewichtetes Interval Scheduling* aus der Vorlesung. Zur Berechnung von $M[j]$ können Sie auf bereits berechnete Werte des Arrays M sowie auf die Werte w_i und Strafkosten t_i aller Jobs J_i zugreifen.

`weightedSchedulingWithPenalties`(Jobs $J_1, \ldots, J_n$):

initialisiere $M \leftarrow$ Array der Länge $n + 1$
$M[0] \leftarrow 0$
berechne Werte $p(1), \ldots, p(n)$ `// wie in Aufgabe (a)`
for $j = 1$ **to** n
  if $p(j) = j - 1$
    $M[j] \leftarrow$ $w_j + M[j-1]$
  else
    $M[j] \leftarrow$ $\max\left(M[p[j]] + w_j - \sum_{k=p(j)+1}^{j-1} t_k \;,\; M[j-1] - t_j \right)$

Wert von dem Job der kompatibel ist + Wert davon – die Jobs die rausfallen

Derjenige Wert ist optimal und wird nicht genommen

return M

c) (6 Punkte)

Füllen Sie die in Aufgabe (b) definierte Tabelle M für die oben angegebene Instanz mit den richtigen Werten aus **und** geben Sie den optimalen Schedule mit seinem Gesamtgewinn an.

$M[1]$	$M[2]$	$M[3]$	$M[4]$	$M[5]$	$M[6]$	$M[7]$
7	7	9	13	11	12	9

Erklärung für a)

→ Wir schauen uns die einzelnen Intervalle an. J_1 & J_2 haben der Wert 0, weil hier kein Job darauf ist

→ J_3 ist kompatibel mit J_1, deshalb wird 1 (Index) in die Tabelle eingeschrieben

→ Selbes gilt für J_4

→ J_5 J_1, J_2 & J_3 sind kompatibel mit 5 Wir nehmen den größten Index, deshalb schreiben wir 3 in die Tabelle

→ etc.

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.815 Algorithmen und Datenstrukturen 2 VU 3.0

Nachtragstest SS 2016

5. Oktober 2016

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	15	21	14	50
Erreichte Punkte:				

Viel Erfolg!

Rucksackproblem:

$$OPT(i,g) = \begin{cases} 0 & \text{wenn } i=0 \\ OPT(i-1,g) & \text{wenn } g_i > g \\ \max\left(\underbrace{OPT(i-1,g)}_{OPT_{OUT}}, \underbrace{w_i + OPT(i-1, g-g_i)}_{OPT_{IN}} \right) & \end{cases}$$

Backtracking

$A = \emptyset$

while $i>0$ and $G>0$

if $M[i,G] \neq M[i-1,G]$

$A = A \cup \{i\}$

$G = G - g_i$

$i = i-1$

return A

Vorw gelöst

Aufgabe A1: Dynamische Programmierung **(15 Punkte)**

Gegeben sei folgende Instanz des Rucksackproblems mit Kapazität $G = 8$ und Gegenständen $\{A, B, C, D\}$ mit den in der Tabelle genannten Gewichten beziehungsweise Werten.

Gegenstand	A	B	C	D
Gewicht	4	2	3	1
Wert	10	6	8	4

a) (10 Punkte)

Wenden Sie das in der Vorlesung vorgestellte dynamische Programm für das Rucksackproblem auf die gegebenen Gegenstände an. Tragen Sie in den **weißen** Zellen der unten stehenden Tabelle die Werte aller Teillösungen ein, die der Algorithmus berechnet.

	0	1	2	3	4	5	6	7	8
$\emptyset$	0	0	0	0	0	0	0	0	0
$\{A\}$	0	0	0	0	10	10	10	10	10
$\{A, B\}$	0	0	6	6	10	10	16	16	16
$\{A, B, C\}$	0	0	6	8	10	14	16	18	18
$\{A, B, C, D\}$	0	4	10	10	12	14	18	20	22

← Gewicht

Gegenstände ↓

$i=1$ $w_1=10$ $g_1=4$

$i=2$ $w_2=6$ $g_2=2$

$i=3$ $w_3=8$ $g_3=3$

$i=4$ $w_4=4$ $g_4=1$

$w_0=0$ $g_0=0$

b) (3 Punkte)

In der Vorlesung wurde auch der Algorithmus „Find-Solution“ zum Finden der Lösung besprochen. Markieren Sie in den **grauen** Zeilen genau diejenigen Zellen, die der Algorithmus „Find-Solution“ während seiner Ausführung ausliest.

Back-Tracking

c) (2 Punkte)

Geben Sie die Menge der für die optimale Lösung ausgewählten Gegenstände an.

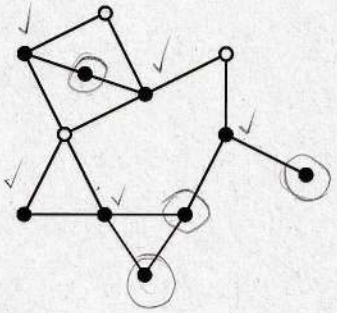
Gegenstände

$A = \{4, 3, 1\} \longrightarrow \{D, C, A\}$

Rep 2-2019

Aufgabe A2: Lokale Suche, Branch and Bound, Approximation (21 Punkte)

Betrachten Sie den folgenden Graphen G und das durch die schwarzen Knoten markierte Vertex Cover W.



a) (3 Punkte)

Führen Sie einen Schritt der Lokalen Suche mittels Nachbarschaftsstruktur für VERTEX COVER aus, wie in der Vorlesung vorgestellt. Markieren Sie eindeutig welcher Knoten dabei aus W entfernt wird.

b) (3 Punkte)

Wie viele verschieden Möglichkeiten gibt es einen Knoten aus W zu entfernen?

4

Alle Nachbaren in LS1 definieren

LS1

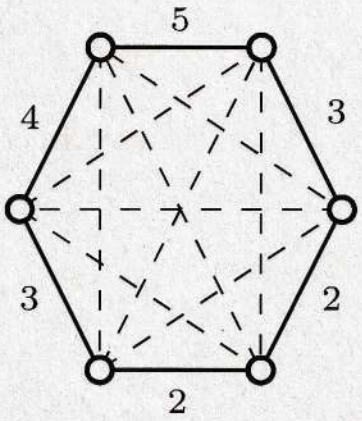
Lokale Suche: remove 1

LS2

Lokale Suche: add 1, remove 2

Rep 2 - 2019

Betrachten Sie die folgende Instanz des TSP Problems und die durch die dicken äußeren Kanten markierte Tour. Alle gestrichelten Kanten haben das Gewicht 3.

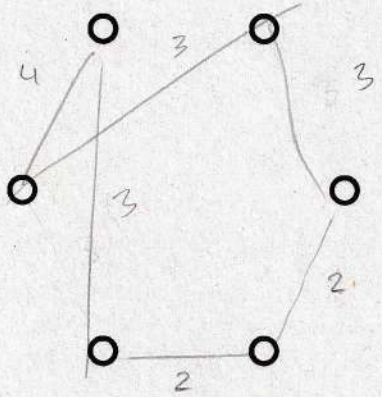


$\sum = 19$

c) (5 Punkte)

2-exchange

Führen Sie einen Schritt der 2-Opt Lokalen Suche durch, der die größtmögliche Verbesserung erzielt. Zeichnen Sie die resultierende Tour in die folgende Abbildung ein.



d) (2 Punkte)

Welches Gewicht hat Ihre oben eingezeichnete Tour? 17

2-exchange

e) (4 Punkte)

Welche der folgenden Aussagen über Branch-and-Bound Algorithmen sind korrekt?

Kreuzen Sie genau diejenigen Antworten an, die eine wahre Aussage darstellen. Bei einem Fehler (falsches oder fehlendes Kreuz) wird noch 1 Punkt vergeben, bei zwei oder mehr Fehlern werden 0 Punkte für diese Aufgabe vergeben.

- [x] In einem Maximierungsproblem kann man die Betrachtung eines Teilproblems abbrechen, wenn dessen lokale obere Schranke kleiner ist als die aktuelle globale untere Schranke.
- [x] Der Wert einer gültigen Lösung in einem Minimierungsproblem ist eine globale obere Schranke.
- [ ] Die asymptotische worst-case Laufzeit eines Branch-and-Bound Algorithmus ist immer polynomiell in der Eingabegröße.
- [ ] Wenn in einem Minimierungsproblem eine gültige Lösung gefunden wird, deren Wert kleiner ist als die bisherige globale obere Schranke, hat man eine optimale Lösung gefunden und kann das Verfahren abbrechen.

f) (4 Punkte)

Welche der folgenden Aussagen über Approximationsalgorithmen sind korrekt?

Kreuzen Sie genau diejenigen Antworten an, die eine wahre Aussage darstellen. Bei einem Fehler (falsches oder fehlendes Kreuz) wird noch 1 Punkt vergeben, bei zwei oder mehr Fehlern werden 0 Punkte für diese Aufgabe vergeben.

- [ ] Algorithmus A ist ein ε-approximativer Algorithmus für ein Maximierungsproblem wenn für ihn gilt, dass er für jede Probleminstanz eine Lösung mit einem Wert zurück liefert, der den optimalen Lösungswert um den Faktor $1/\varepsilon$ nicht unterschreitet.
- [x] Algorithmus A ist ein ε-approximativer Algorithmus für ein Maximierungsproblem wenn für ihn gilt, dass er für jede Probleminstanz eine Lösung mit einem Wert zurück liefert, der den optimalen Lösungswert um den Faktor ε nicht unterschreitet.
- [ ] Für das symmetrische Traveling Salesman Problem mit beliebiger Kostenmatrix besitzt die Spanning-Tree-Heuristik eine Gütegarantie von 2.
- [x] Für das Problem ein kleinstes Vertex Cover für einen gegebenen Graphen zu finden gibt es einen Approximationsalgorithmus mit Gütegarantie 3.

muss metrisch sein

$\frac{c_A}{c_{OPT}} \leq \varepsilon \quad c_A \leq \varepsilon \cdot c_{OPT}$

$\frac{c_{OPT}}{c_A} \leq \frac{1}{\varepsilon}$

$c_{OPT} \leq \frac{1}{\varepsilon} \cdot c_A$

Rep 2, 2018

Aufgabe A3: Interval Scheduling **(14 Punkte)**

Betrachten Sie folgende Problemdefinition und die darunter angegebene Instanz.

Problem (gewichtetes Interval Scheduling mit Strafkosten). *Gegeben sei eine Menge von n Jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ mit folgenden Eigenschaften:*

- *Job J_i startet zum Zeitpunkt s_i und endet zum Zeitpunkt $f_i > s_i$.*
- *Job J_i hat den Wert $w_i > 0$, wenn er ausgeführt wird, und die Strafkosten $t_i \geq 0$, wenn er nicht ausgeführt wird.*
- *Zwei Jobs J_i und J_k sind* kompatibel, *wenn sie sich nicht überlappen, also $f_i \leq s_k$ oder $f_k \leq s_i$ gilt.*

Gesucht ist eine Teilmenge $S \subset \mathcal{J}$ von paarweise kompatiblen Jobs, mit maximalem Gesamtgewinn

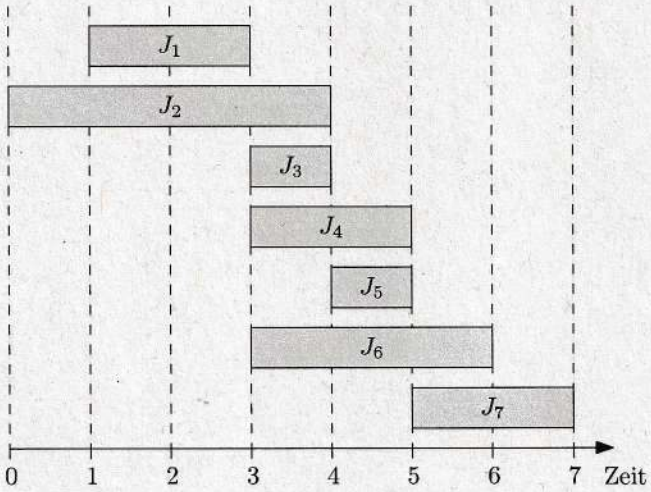
$$G(S) = \sum_{J_i \in S} w_i - \sum_{J_i \in \mathcal{J} \setminus S} t_i,$$

also dem Wert der ausgeführten Jobs minus den Strafkosten der nicht ausgeführten Jobs.

Hinweis: Wenn alle Strafkosten $t_i = 0$ $(i = 1, \ldots, n)$ sind, erhält man genau das aus der Vorlesung bekannte Problem *gewichtetes Interval Scheduling.*

Job	Startzeit	Ende	Wert	Strafkosten
J_1	$s_1 = 1$	$f_1 = 3$	$w_1 = 7$	$t_1 = 10$
J_2	$s_2 = 0$	$f_2 = 4$	$w_2 = 10$	$t_2 = 0$
J_3	$s_3 = 3$	$f_3 = 4$	$w_3 = 2$	$t_3 = 0$
J_4	$s_4 = 3$	$f_4 = 5$	$w_4 = 6$	$t_4 = 5$
J_5	$s_5 = 4$	$f_5 = 5$	$w_5 = 5$	$t_5 = 2$
J_6	$s_6 = 3$	$f_6 = 6$	$w_6 = 12$	$t_6 = 7$
J_7	$s_7 = 5$	$f_7 = 7$	$w_7 = 5$	$t_7 = 5$

Vorgänger
p(1) = 0
p(2) = 0
p(3) = 1
p(4) = 1
p(5) = 3
p(6) = 1
p(7) = 5



Ursprünglicher Algorithmus aus VO:

$$OPT(j) = \begin{cases} 0 & \text{wenn } j = 0 \\ \max\left(OPT_{IN}(j), OPT_{OUT}(j)\right) & \text{sonst} \end{cases}$$

$OPT_{IN}(j) \rightarrow w_j + OPT(p(j))$

$OPT_{OUT}(j) \rightarrow OPT(j-1)$

In diesem Fall ist OPT aber auch zusätzlich die negative Summe aller nicht gewählten Werte.

$$M[j] = \begin{cases} w_j + M[j-1] & \text{wenn } p(j) = j-1 \\ \max\left\{ w_j + M[p(j)] - \sum_{k=p(j)+1}^{j-1} t_k \;,\; M[j-1] - t_j \right\} & \text{sonst} \end{cases}$$

Berechnung:

j	p(j)	w_j	t_j	M[j]
1	0	7	10	$p(1) = 1-1$ deshalb $w_1 + 0 = 7$
2	0	10	0	$\max\{w_2 + M[0] - t_1, M[1] - t_2\} = 7$
3	1	2	0	$\max\{w_3 + M[1] - t_2, M[2] - t_3\} = 9$
4	1	6	5	$\max\{w_4 + M[1] - t_2 - t_3, M[3] - t_4\} = 13$
5	3	5	2	$\max\{w_5 + M[3] - t_4, M[4] - t_5\} = 11$
6	1	12	7	$\max\{w_6 + M[1] - t_2 - t_3 - t_4 - t_5, M[5] - t_6\} = 12$
7	5	5	5	$\max\{w_7 + M[5] - t_6, M[6] - t_7\} = 9$

$M[2] = \max\{10 + 7 - 10, 7 - 0\} = 7$

$M[3] = \max\{2 + 7 - 0, 7 - 0\} = 9$

$M[4] = \max\{6 + 7 - 0 - 0, 9 - 5\} = \cancel{11}\ 13$

Alternativer Lösungsansatz:

$$G(S) = \sum_{j \in S} w_j - \sum_{j \in J \setminus S} t_j =$$

Belohnung — Strafe

Umformulierung:

$$= \sum_{j \in S} w_j - \left(\sum_{j \in S} t_j - \sum_{j \in S} t_j \right) =$$

$$= \sum_{j \in S} w_j - \sum_{j \in S} t_j + \sum_{j \in S} t_j =$$

$$= \sum_{j \in S} w_j + \sum_{j \in S} t_j - \sum_{j \in S} t_j =$$

$$= \sum_{j \in S} w_j + t_j - \sum_{j \in S} t_j$$

$w_j' = w_j + t_j$ bleibt immer konstant

a) (2 Punkte)

Berechnen Sie für die oben angegebene Instanz, deren Jobs nicht-absteigend nach dem Ende f_j ($j = 1, \ldots, 7$) sortiert sind, die Werte $p(j)$ für $j = 1, \ldots, 7$. Der Wert $p(j)$ gibt – wie in der Vorlesung – den größten Index $i < j$ an, so dass Job J_i kompatibel zu J_j ist. Füllen Sie die Tabelle aus. Existiert kein solcher Job J_i sei der Wert als $p(j) = 0$ definiert.

$p(1)$	$p(2)$	$p(3)$	$p(4)$	$p(5)$	$p(6)$	$p(7)$
0	0	1	1	3	1	5

b) (6 Punkte)

Ergänzen Sie die beiden fehlenden Zeilen in nachfolgendem Pseudocode, so dass nachdem der Algorithmus ausgeführt wurde, der Eintrag $M[j]$ für $j = 1, \ldots, n$ den Gesamtgewinn eines optimalen Schedules der Jobs $J_1, J_2, \ldots, J_j$ enthält. Orientieren Sie sich an dem Bottom-Up Verfahren für *gewichtetes Interval Scheduling* aus der Vorlesung. Zur Berechnung von $M[j]$ können Sie auf bereits berechnete Werte des Arrays M sowie auf die Werte w_i und Strafkosten t_i aller Jobs J_i zugreifen.

`weightedSchedulingWithPenalties`(Jobs $J_1, \ldots, J_n$):

initialisiere $M \leftarrow$ Array der Länge $n + 1$
$M[0] \leftarrow 0$
berechne Werte $p(1), \ldots, p(n)$ `// wie in Aufgabe (a)`
for $j = 1$ **to** n
  if $p(j) = j - 1$
    $M[j] \leftarrow$ $w_j + M[j-1]$ keine Strafe
  else
    $M[j] \leftarrow$ $\max\{ w_j + M[p(j)] - \sum_{k=p(j)+1}^{j-1} t_k \text{ (OPT}_{IN}\text{, Strafe = alle dazwischen)},\ M[j-1] - t_j \}$
    OPT$_{OUT}$, Strafe $= t_j$
return M

c) (6 Punkte)

Füllen Sie die in Aufgabe (b) definierte Tabelle M für die oben angegebene Instanz mit den richtigen Werten aus **und** geben Sie den optimalen Schedule mit seinem Gesamtgewinn an.

$M[1]$	$M[2]$	$M[3]$	$M[4]$	$M[5]$	$M[6]$	$M[7]$
7	7	9	13	11	12	9

Rechnung

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.815 Algorithmen und Datenstrukturen 2 VU 3.0

1.Übungstest SS 2016

gelöst

23. Juni 2016

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	17	20	13	50
Erreichte Punkte:				

Viel Erfolg!

A1 - A2 in Rep SS17 besprochen

Aufgabe A1: Dynamische Programmierung **(17 Punkte)**

Das Minimum-Weight Vertex Cover Problem sei wie folgt definiert.

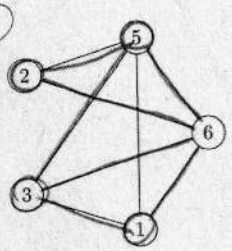
Problem (Minimum-Weight Vertex Cover). *Gegeben sei ein schlichter Graph $G = (V, E)$ mit positiven Knotengewichten $w_v \in \mathbb{R}^+$ für alle Knoten $v \in V$. Gesucht ist ein Vertex Cover $V' \subset V$ mit minimalem Gewicht, d.h. für jede Kante $(u, v) \in E$ gilt $\{u, v\} \cap V' \neq \emptyset$ und $\sum_{v \in V'} w_v$ ist minimal über alle möglichen Vertex Cover von G.*

a) (8 Punkte)

Markieren Sie in den beiden angegebenen Graphen jeweils ein Vertex Cover mit minimalem Gewicht. Die Werte in den Knoten entsprechen dabei den Knotengewichten.

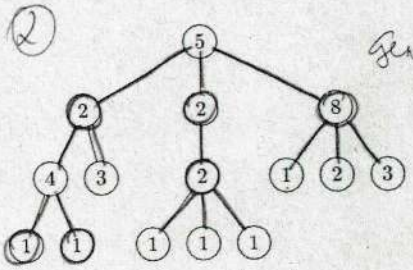
(1)

Gewicht 11



(2)

Gewicht 16



(1) → Bei einem MVC würden wir Knoten 5, 6, 1 auswählen
→ Wir fangen wir mit den geringsten Gewicht an, also 1. Danach 2, 3 und 5. Alle Kanten sind abgedeckt. Wir haben ein Gewicht von 11
(1+2+3+5=11)

(2) → Wir verwenden den Alg. aus der VO (Weighted-Independent-Set-In-A-Tree).
→ linker Teilbaum: Wir wählen 1 & 1 und 2
→ rechter Teilbaum: Wählen 8
→ mittlerer Teilbaum: 2 & 2
→ Gewicht: 2+1+1+2+2+8=16

b) (9 Punkte)

Betrachten Sie nun den Fall, dass der gegebene Graph ein Baum T ist. Im Folgenden ist ein Gerüst für einen Algorithmus in Pseudocode gegeben, der das Minimum-Weight Vertex Cover Problem auf Bäumen mittels dynamischer Programmierung lösen soll.

Die Struktur entspricht dem aus der Vorlesung bekannten Algorithmus zur Berechnung eines Maximum-Weight Independent Set auf Bäumen. Für jeden Knoten $u \in V$ (mit Gewicht w_u) sollen die Tabelleneinträge $M_{\text{in}}[u]$ und $M_{\text{out}}[u]$ ausgefüllt werden. Am Ende enthält $M_{\text{in}}[u]$ den besten Wert für eine Lösung des Teilbaums T_u mit u als Wurzel, so dass u Teil der Lösung ist. Äquivalent enthält $M_{out}[u]$ den besten Wert einer Lösung für T_u, aber unter der Annahme, dass u *nicht* Teil der Lösung ist.

Am Ende soll der Algorithmus das minimale Gewicht eines Vertex Cover von T retournieren. Ergänzen sie die markierten Zeilen so, dass der Algorithmus korrekt ist. Definieren Sie (falls nötig) eigene Notation.

`Minimum-Weight-Vertex-Cover-In-A-Tree`(Baum $T = (V, E)$):

Wähle eine Wurzel $r \in V$ aus
foreach *Knoten u in T in Postorder* **do**
  if *u ist ein Blatt*
    $M_{\text{in}}[u] \leftarrow$ w_u
    $M_{\text{out}}[u] \leftarrow$ 0
  else `// u ist kein Blatt`
    $M_{\text{in}}[u] \leftarrow$ $w_u + \sum_{v \in N(u)} \min(M_{in}[v], M_{out}[v])$
    $M_{\text{out}}[u] \leftarrow$ $\sum_{v \in N(u)} M_{in}[v]$
return $\min(M_{in}[r], M_{out}[r])$

M_{in} ... wird übernommen
M_{out} ... wird nicht übernommen
$N(u)$ ... Nachfolger(u)

b) (9 Punkte)

Betrachten Sie nun den Fall, dass der gegebene Graph ein Baum T ist. Im Folgenden ist ein Gerüst für einen Algorithmus in Pseudocode gegeben, der das Minimum-Weight Vertex Cover Problem auf Bäumen mittels dynamischer Programmierung lösen soll.

Die Struktur entspricht dem aus der Vorlesung bekannten Algorithmus zur Berechnung eines Maximum-Weight Independent Set auf Bäumen. Für jeden Knoten $u \in V$ (mit Gewicht w_u) sollen die Tabelleneinträge $M_{\text{in}}[u]$ und $M_{\text{out}}[u]$ ausgefüllt werden. Am Ende enthält $M_{\text{in}}[u]$ den besten Wert für eine Lösung des Teilbaums T_u mit u als Wurzel, so dass u Teil der Lösung ist. Äquivalent enthält $M_{out}[u]$ den besten Wert einer Lösung für T_u, aber unter der Annahme, dass u *nicht* Teil der Lösung ist.

Am Ende soll der Algorithmus das minimale Gewicht eines Vertex Cover von T retournieren. Ergänzen sie die markierten Zeilen so, dass der Algorithmus korrekt ist. Definieren Sie (falls nötig) eigene Notation.

`Minimum-Weight-Vertex-Cover-In-A-Tree`(Baum $T = (V, E)$):

Wähle eine Wurzel $r \in V$ aus
foreach *Knoten u in T in Postorder* **do**
  if *u ist ein Blatt*
    $M_{\text{in}}[u] \leftarrow$ w_u
    $M_{\text{out}}[u] \leftarrow$ 0
  else `// u ist kein Blatt`
    $M_{\text{in}}[u] \leftarrow$ $w_u + \sum_{v \in N(u)} \min\left(M_{in}[v], M_{out}[v]\right)$
    $M_{\text{out}}[u] \leftarrow$ $\sum_{v \in N(u)} M_{in}[v]$
return $\min\left(M_{in}[r], M_{out}[r]\right)$

M_{in} ... wird übernommen
M_{out} ... wird nicht übernommen
$N(u)$ ... Nachfolger(u)

Aufgabe A2: Approximation und Polynomialzeitreduktion **(20 Punkte)**

Betrachten Sie erneut das Minimum-Weight Vertex Cover Problem aus Aufgabe 1.

Problem (Minimum-Weight Vertex Cover). *Gegeben sei ein schlichter Graph $G = (V, E)$ mit positiven Knotengewichten $w_v \in \mathbb{R}^+$ für alle Knoten $v \in V$. Gesucht ist ein Vertex Cover $V' \subset V$ mit minimalem Gewicht, d.h. für jede Kante $(u, v) \in E$ gilt $\{u, v\} \cap V' \neq \emptyset$ und $\sum_{v \in V'} w_v$ ist minimal über alle möglichen Vertex Cover von G.*

a) (10 Punkte)

Nehmen Sie nun zusätzlich an, dass in $G = (V, E)$ alle Knotengewichte w_v für $v \in V$ nur Werte aus der Menge $\{1, 2, 3\}$ sind. Angenommen Sie wenden den unten angegebenen Approximationsalgorithmus `Approx-Vertex-Cover` aus der Vorlesung unverändert auf den gewichteten Graphen G an. Besitzt dieser Algorithmus dann immer noch eine konstante Gütegarantie für das Minimum-Weight Vertex Cover Problem auf G? Begründen Sie Ihre Antwort und geben Sie ggf. die erzielbare Gütegarantie an.

```
Approx-Vertex-Cover(Graph G = (V, E)):
    C ← ∅
    while E ≠ ∅
        wähle eine beliebige Kante (u, v) ∈ E
        C ← C ∪ {u, v}
        entferne aus E alle Kanten, die inzident zu u oder v sind
    return C
```

→ ja, es gibt eine Gütegarantie. Diese ist ≤ 4

→ 

→ Dies ist der Extremfall. Wir haben einen Knoten mit Gewicht 1 und einen mit 3. (1+3=4). Der Alg nimmt beide Knoten und wir haben ein Gewicht von 4

→ Wenn man sich alle Fälle anschaut, sieht man dass 1 & 3 der worst case ist.

b) (4 Punkte)

Welche der folgenden Aussagen über NP-Vollständigkeit sind korrekt?

Kreuzen Sie genau diejenigen Antworten an, die eine wahre Aussage darstellen. Bei einem Fehler (falsches oder fehlendes Kreuz) wird noch 1 Punkt vergeben, bei zwei oder mehr Fehlern werden 0 Punkte für diese Aufgabe vergeben.

- [x] Wenn es einen polynomiellen Algorithmus für INDEPENDENT SET gibt, gilt P=NP. Independent Set ist ein schweres Problem und kann in polynomieller Zeit gelöst werden
- [ ] Wenn P ≠ NP gilt, dann gibt es keinen polynomiellen Algorithmus für INDEPENDENT SET auf Bäumen. Es gibt einen Algorithmus, deshalb falsch.
- [x] Für $k = 12$ kann man in Zeit $O(n)$ bestimmen, ob ein schlichter Graph G mit n Knoten ein Vertex Cover der Größe $\leq k$ besitzt. $2^k n$. k ist eine Konstante
- [x] Wenn P ≠ NP gilt, dann existiert kein polynomieller Algorithmus für das Problem SET COVER. Kann in O(n) gelöst werden

Kapitel Spezialfall S12 S5

Kapitel Polynomialreduktion

c) (4 Punkte)

Seien X, Y und Z drei Ja/Nein-Probleme in NP. Weiters seien R_{XY} ein Reduktionsalgorithmus von X auf Y mit Laufzeit $O(n^3)$ und R_{YZ} ein Reduktionsalgorithmus von Y auf Z mit Laufzeit $O(n^2)$.

Welche der folgenden Aussagen sind korrekt?

Kreuzen Sie genau diejenigen Antworten an, die eine wahre Aussage darstellen. Bei einem Fehler (falsches oder fehlendes Kreuz) wird noch 1 Punkt vergeben, bei zwei oder mehr Fehlern werden 0 Punkte für diese Aufgabe vergeben.

- [x] Wenn Z effizient lösbar ist, dann ist auch X effizient lösbar. wegen der Transitivität X auf Y und Y auf Z und das in polynomieller Zeit
- [ ] Wenn X effizient lösbar ist, dann ist auch Z effizient lösbar.
- [ ] Wenn Y NP-vollständig ist, dann ist auch X NP-vollständig. Geht in die andere Richtung → deshalb falsch
- [x] Wenn Y NP-vollständig ist, dann ist auch Z NP-vollständig.

Es können keine Aussagen über X gemacht werden

d) (2 Punkte)

Folien AD2-01 S14

Seien X, Y und Z wieder drei Ja/Nein-Probleme in NP. Weiters seien R_{XY} ein Reduktionsalgorithmus von X auf Y mit Laufzeit $O(n^3)$ und R_{YZ} ein Reduktionsalgorithmus von Y auf Z mit Laufzeit $O(n^2)$.

Nehmen Sie an, Sie kennen einen Algorithmus, der eine Instanz von Z der Größe n in Zeit $O(n^{1.5})$ löst. In welcher asymptotischen Laufzeit können Sie dann eine Instanz von X der Größe n lösen?

$$O\left(\left(n^3\right)^2\right)^{1,5} = O(n^9)$$

3·2·1,5 = 9

$X \leq_P Y \leq_P Z$

1. Reduktion: $O(n^3)$
2. Reduktion: $O(n^2)$
3. Reduktion: $O(n^{1,5})$

Aufgabe A3: Branch and Bound **(13 Punkte)**

Betrachten Sie die folgende Instanz des Rucksackproblems. Die Gewichte und Werte der Gegenstände sind in der Tabelle angegeben. Die Kapazität des Rucksacks beträgt $G = 20$.

Gegenstand	A	B	C	D
Gewicht g_i	8	7	10	3
Wert w_i	64	49	120	18
Verhältnis	8	7	12	6

a) (2 Punkte)

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.

Verhältnis = w_i / g_i.

Sortierung (absteigend): C(12), A(8), B(7), D(6)

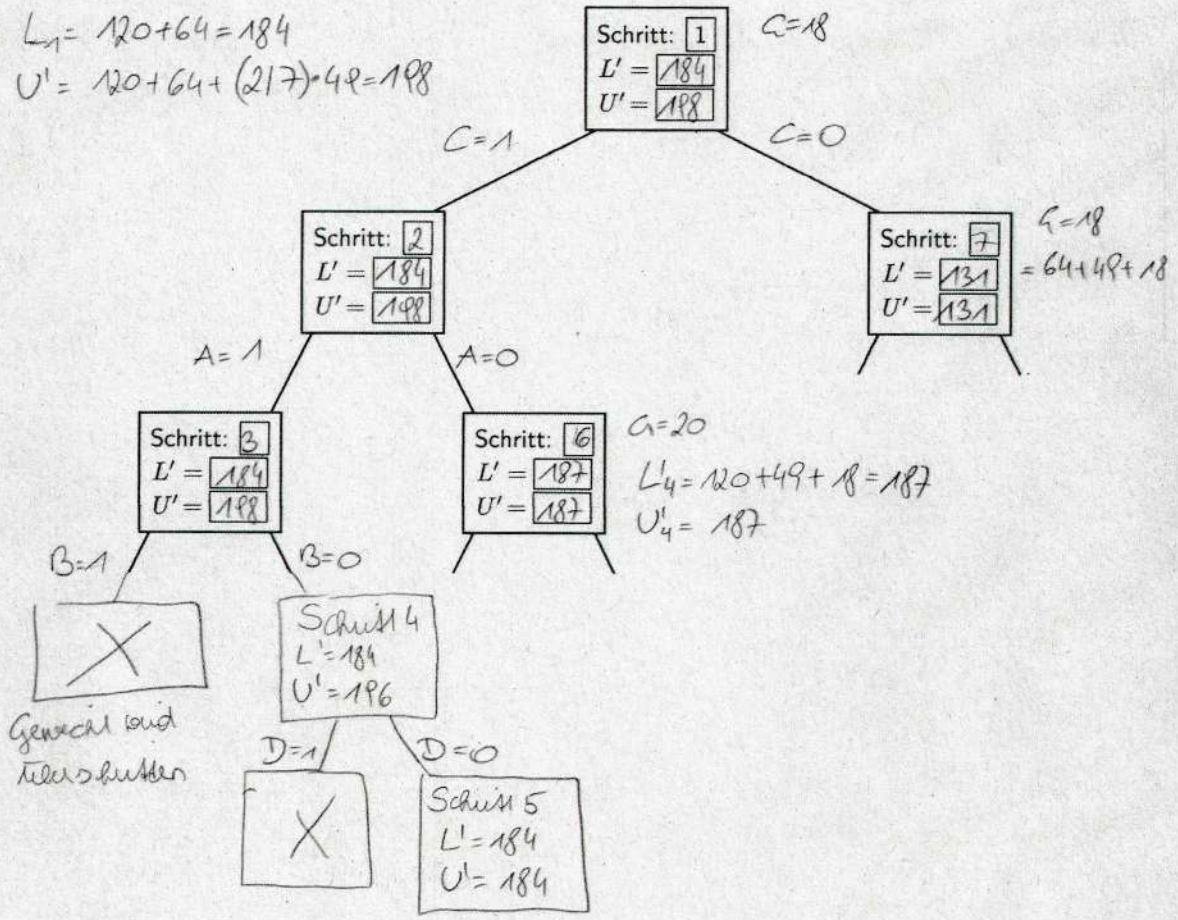
b) (9 Punkte)

Wenden Sie den verbesserten Branch-and-Bound-Algorithmus[1] aus der Vorlesung auf die Instanz an. Nutzen Sie dabei die Depth-first Strategie zur Auswahl des nächsten Teilproblems. Verwenden Sie zur Lösung der Aufgabe den leeren Branch-and-Bound Baum auf der nächsten Seite.

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. Beachten Sie, dass der Baum nach Bedarf erweitert werden muss und zeichnen Sie die zusätzlich benötigten Kanten und Knoten.

[1] Zur Erinnerung: Die untere Schranke wird durch einen Greedy-Algorithmus berechnet, für die obere Schranke können Gegenstände auch partiell eingepackt werden. Die Reihenfolge, in der die Gegenstände betrachtet werden, ergibt sich aus den Wert-Gewichts-Verhältnissen (s. Aufgabe 3(a)).

C(12), A(8), B(7), D(6) G=20



U_4'= 120+64+(213)·18=196

c) (2 Punkte)

Geben Sie die optimale Lösung und den zugehörigen Lösungswert an.

optimale Lsg. Gegenstand: C, B, D

Wert: 187

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.815 Algorithmen und Datenstrukturen 2 VU 3.0

1.Übungstest SS 2016

23. Juni 2016

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	17	20	13	50
Erreichte Punkte:				

Viel Erfolg!

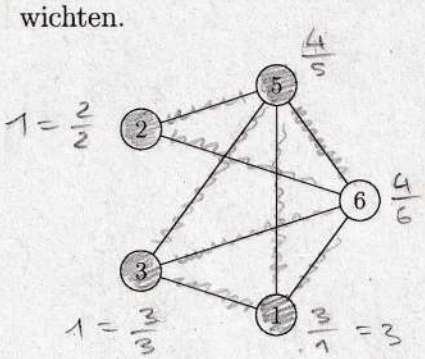
Aufgabe A1: Dynamische Programmierung (17 Punkte)

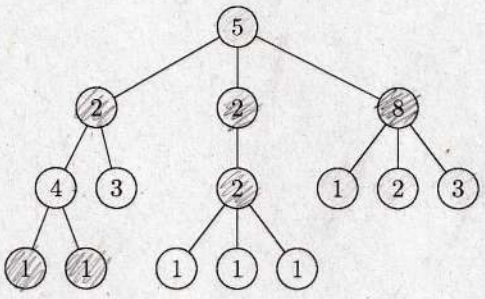
Das Minimum-Weight Vertex Cover Problem sei wie folgt definiert.

Problem (Minimum-Weight Vertex Cover). *Gegeben sei ein schlichter Graph $G = (V, E)$ mit positiven Knotengewichten $w_v \in \mathbb{R}^+$ für alle Knoten $v \in V$. Gesucht ist ein Vertex Cover $V' \subset V$ mit minimalem Gewicht, d.h. für jede Kante $(u, v) \in E$ gilt $\{u, v\} \cap V' \neq \emptyset$ und $\sum_{v \in V'} w_v$ ist minimal über alle möglichen Vertex Cover von G.*

a) (8 Punkte)

Markieren Sie in den beiden angegebenen Graphen jeweils ein Vertex Cover mit minimalem Gewicht. Die Werte in den Knoten entsprechen dabei den Knotengewichten.





Greedy:
nach Preis-Leistung:

Knotengrad / Wert

Dynamische Programmierung

min (Blatt nehmen + für Eltern rekursiv entscheiden, Eltern nehmen)

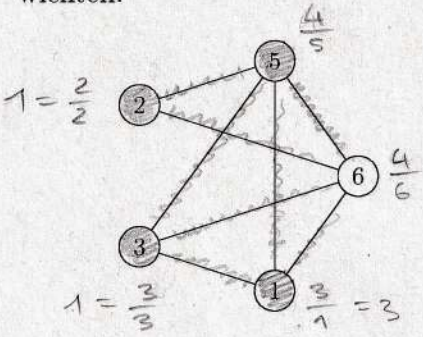
Aufgabe A1: Dynamische Programmierung (17 Punkte)

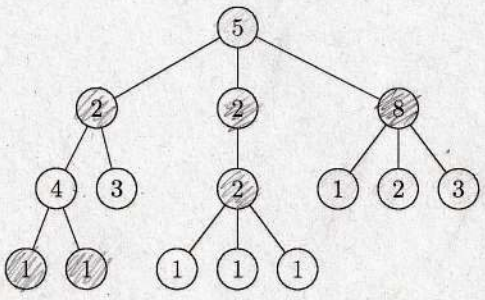
Das Minimum-Weight Vertex Cover Problem sei wie folgt definiert.

Problem (Minimum-Weight Vertex Cover). *Gegeben sei ein schlichter Graph $G = (V, E)$ mit positiven Knotengewichten $w_v \in \mathbb{R}^+$ für alle Knoten $v \in V$. Gesucht ist ein Vertex Cover $V' \subset V$ mit minimalem Gewicht, d.h. für jede Kante $(u, v) \in E$ gilt $\{u, v\} \cap V' \neq \emptyset$ und $\sum_{v \in V'} w_v$ ist minimal über alle möglichen Vertex Cover von G.*

a) (8 Punkte)

Markieren Sie in den beiden angegebenen Graphen jeweils ein Vertex Cover mit minimalem Gewicht. Die Werte in den Knoten entsprechen dabei den Knotengewichten.





Greedy:
nach Preis-Leistung:

Knotengrad / Gewicht

Dynamische Programmierung

min (Blatt nehmen + für Eltern rekursiv entscheiden,

Eltern nehmen)

b) (9 Punkte)

Betrachten Sie nun den Fall, dass der gegebene Graph ein Baum T ist. Im Folgenden ist ein Gerüst für einen Algorithmus in Pseudocode gegeben, der das Minimum-Weight Vertex Cover Problem auf Bäumen mittels dynamischer Programmierung lösen soll.

Die Struktur entspricht dem aus der Vorlesung bekannten Algorithmus zur Berechnung eines Maximum-Weight Independent Set auf Bäumen. Für jeden Knoten $u \in V$ (mit Gewicht w_u) sollen die Tabelleneinträge $M_{\text{in}}[u]$ und $M_{\text{out}}[u]$ ausgefüllt werden. Am Ende enthält $M_{\text{in}}[u]$ den besten Wert für eine Lösung des Teilbaums T_u mit u als Wurzel, so dass u Teil der Lösung ist. Äquivalent enthält $M_{out}[u]$ den besten Wert einer Lösung für T_u, aber unter der Annahme, dass u *nicht* Teil der Lösung ist.

Am Ende soll der Algorithmus das minimale Gewicht eines Vertex Cover von T retournieren. Ergänzen sie die markierten Zeilen so, dass der Algorithmus korrekt ist. Definieren Sie (falls nötig) eigene Notation.

`Minimum-Weight-Vertex-Cover-In-A-Tree`(Baum $T = (V, E)$):

Wähle eine Wurzel $r \in V$ aus
foreach *Knoten u in T in Postorder* **do**
  if *u ist ein Blatt*
    $M_{\text{in}}[u] \leftarrow$ w_u
    $M_{\text{out}}[u] \leftarrow$ 0
  else `// u ist kein Blatt`
    $M_{\text{in}}[u] \leftarrow$ $w_u + \sum_{u.\text{child} \in V} \min(M_{\text{IN}}[v], M_{\text{OUT}}[v])$
    $M_{\text{out}}[u] \leftarrow$ $\sum_{u.\text{child} \in V} M_{\text{IN}}[v]$
return $\min(M_{\text{IN}}[r], M_{\text{OUT}}[r])$

Rep 2 – 2019

Aufgabe A2: Approximation und Polynomialzeitreduktion (20 Punkte)

Betrachten Sie erneut das Minimum-Weight Vertex Cover Problem aus Aufgabe 1.

Problem (Minimum-Weight Vertex Cover). *Gegeben sei ein schlichter Graph $G = (V, E)$ mit positiven Knotengewichten $w_v \in \mathbb{R}^+$ für alle Knoten $v \in V$. Gesucht ist ein Vertex Cover $V' \subset V$ mit minimalem Gewicht, d.h. für jede Kante $(u, v) \in E$ gilt $\{u, v\} \cap V' \neq \emptyset$ und $\sum_{v \in V'} w_v$ ist minimal über alle möglichen Vertex Cover von G.*

a) (10 Punkte)

Nehmen Sie nun zusätzlich an, dass in $G = (V, E)$ alle Knotengewichte w_v für $v \in V$ nur Werte aus der Menge $\{1, 2, 3\}$ sind. Angenommen Sie wenden den unten angegebenen Approximationsalgorithmus `Approx-Vertex-Cover` aus der Vorlesung unverändert auf den gewichteten Graphen G an. Besitzt dieser Algorithmus dann immer noch eine konstante Gütegarantie für das Minimum-Weight Vertex Cover Problem auf G? Begründen Sie Ihre Antwort und geben Sie ggf. die erzielbare Gütegarantie an.

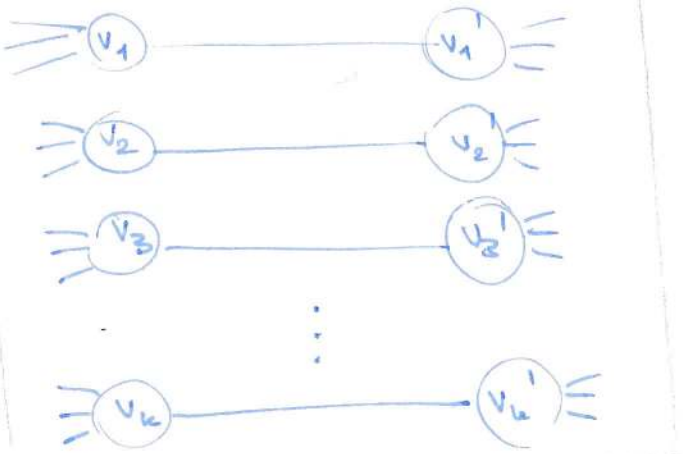
```
Approx-Vertex-Cover(Graph G = (V, E)):
    C ← ∅
    while E ≠ ∅
        wähle eine beliebige Kante (u, v) ∈ E
        C ← C ∪ {u, v}
        entferne aus E alle Kanten, die inzident zu u oder v sind
    return C
```

Ohne Gewichtung der Knoten

$$\frac{c_A}{c_{OPT}} \leq 2 = \varepsilon$$

Aufgabe A2)

Wir nehmen beliebigen Graph an:



Approximationsalgorithmus nimmt beide Seiten

$$A = \sum_{i=1}^{k} w(v_i) + w(v_i')$$

Optimaler Algorithmus nimmt kleinere Seite

$$OPT = \sum_{i=1}^{k} \min\left(w(v_i), w(v_i')\right)$$

↓

Daraus folgt:

Obere Schranke für OPT:

$$OPT \geq \sum_{i=1}^{k} \min\left(3, 1\right) = 1 \cdot k$$

Untere Schranke für Approx:

$$A \leq \sum_{i=1}^{k} 3 + 1 = 4k$$

↓

$$\frac{A}{OPT} \leq 4 = \varepsilon$$

Gütegarantie von $\varepsilon = 4$ auch beweisbar:

$$\frac{\sum a_i}{\sum b_i} \leq \max_i \frac{a_i}{b_i}$$

(kompliziertes)

Intuitives Argument:

Unabhängig von Knoten-Gewichtung ist $\frac{A}{OPT} \leq 2$.

Mit Gewichtung lässt es sich behaupten, dass alle Knoten mindestens ein Gewicht von 1 haben und es eine allgemeine untere Schranke von $1 \cdot n$ gibt.

Weiters eine obere Schranke von $3 \cdot n$

$\rightarrow 3n \geq A \geq OPT \geq n$

Aufgabe A2) b)

☑ Independent Set ist NP-C ⇒ (zugleich ∈NP-Hard und ∈NP)

↑ Alle NP lassen sich darauf reduzieren

↑ Lässt sich in P verifizieren

Das bedeutet:
Wenn P = NP-C dann
NP ≤p NP-C = P

also:
P = NP

☐ Bäume sind ein Spezialfall

☑ Bei Vertex Covers gilt:

Wenn k Teil der Problem-Instanz ist: Input (G,k) → Normales NP-Vollständiges Vertex Cover

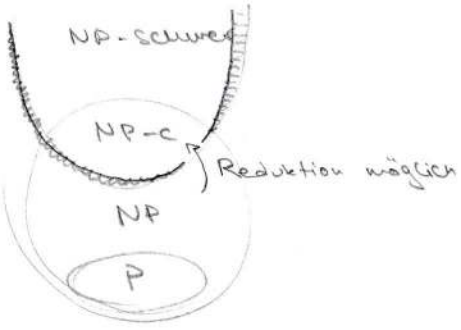
JA/NEIN-PROBLEM: Input (G,k) → Output (JA xor NEIN)

Wenn k fix vorgegeben wird und nicht Teil der Probleminstanz ist, dann ist es ein Spezialfall, dass sich in $O(2^k \cdot k \cdot (n-1))$ lösen lässt → $O(n)$ weil k = const

2^k weil rekursiv berechnet

Verifikation: Weil jeder der k Knoten der Menge höchstens n-1 Kanten haben kann

☑ Ja weil



c) $X \leq_P Y \leq_P Z$ ← polynomielle Reduktion

☑ Alle anderen müssen P oder leichter sein

☐

☐ muss nicht sein

☑ Z ist mindestens so schwer wie Y

d)

$X \leq_p Y \leq_p Z$

$X \in NP$) R_{XY} $O(n^3)$
$Y \in NP$ ↙
$Z \in NP$) R_{YZ} $O(n^2)$

Angenommen $Z \in O(n^{1,5})$

Eine Reduktion von X auf Z:

Eingabegröße von X: n_0 $\longrightarrow O\left(\left(\left((n_0)^3\right)^2\right)^{1,5}\right) = O(n_0^9)$

Rep 2 – 2018

b) (4 Punkte)

Welche der folgenden Aussagen über NP-Vollständigkeit sind korrekt?

Kreuzen Sie genau diejenigen Antworten an, die eine wahre Aussage darstellen. Bei einem Fehler (falsches oder fehlendes Kreuz) wird noch 1 Punkt vergeben, bei zwei oder mehr Fehlern werden 0 Punkte für diese Aufgabe vergeben.

- ☐ Wenn es einen polynomiellen Algorithmus für INDEPENDENT SET gibt, gilt P=NP.
- ☐ Wenn P $\neq$ NP gilt, dann gibt es keinen polynomiellen Algorithmus für INDEPENDENT SET auf Bäumen.
- ☐ Für $k = 12$ kann man in Zeit $O(n)$ bestimmen, ob ein schlichter Graph G mit n Knoten ein Vertex Cover der Größe $\leq k$ besitzt.
- ☐ Wenn P $\neq$ NP gilt, dann existiert kein polynomieller Algorithmus für das Problem SET COVER.

c) (4 Punkte)

Seien X, Y und Z drei Ja/Nein-Probleme in NP. Weiters seien R_{XY} ein Reduktionsalgorithmus von X auf Y mit Laufzeit $O(n^3)$ und R_{YZ} ein Reduktionsalgorithmus von Y auf Z mit Laufzeit $O(n^2)$.

Welche der folgenden Aussagen sind korrekt?

Kreuzen Sie genau diejenigen Antworten an, die eine wahre Aussage darstellen. Bei einem Fehler (falsches oder fehlendes Kreuz) wird noch 1 Punkt vergeben, bei zwei oder mehr Fehlern werden 0 Punkte für diese Aufgabe vergeben.

- ☐ Wenn Z effizient lösbar ist, dann ist auch X effizient lösbar.
- ☐ Wenn X effizient lösbar ist, dann ist auch Z effizient lösbar.
- ☐ Wenn Y NP-vollständig ist, dann ist auch X NP-vollständig.
- ☐ Wenn Y NP-vollständig ist, dann ist auch Z NP-vollständig.

d) (2 Punkte)

Seien X, Y und Z wieder drei Ja/Nein-Probleme in NP. Weiters seien R_{XY} ein Reduktionsalgorithmus von X auf Y mit Laufzeit $O(n^3)$ und R_{YZ} ein Reduktionsalgorithmus von Y auf Z mit Laufzeit $O(n^2)$.

Nehmen Sie an, Sie kennen einen Algorithmus, der eine Instanz von Z der Größe n in Zeit $O(n^{1.5})$ löst. In welcher asymptotischen Laufzeit können Sie dann eine Instanz von X der Größe n lösen?

Aufgabe A3: Branch and Bound **(13 Punkte)**

Betrachten Sie die folgende Instanz des Rucksackproblems. Die Gewichte und Werte der Gegenstände sind in der Tabelle angegeben. Die Kapazität des Rucksacks beträgt $G = 20$.

Gegenstand	A	B	C	D
Gewicht	8	7	10	3
Wert	64	49	120	18
Verhältnis				

a) (2 Punkte)

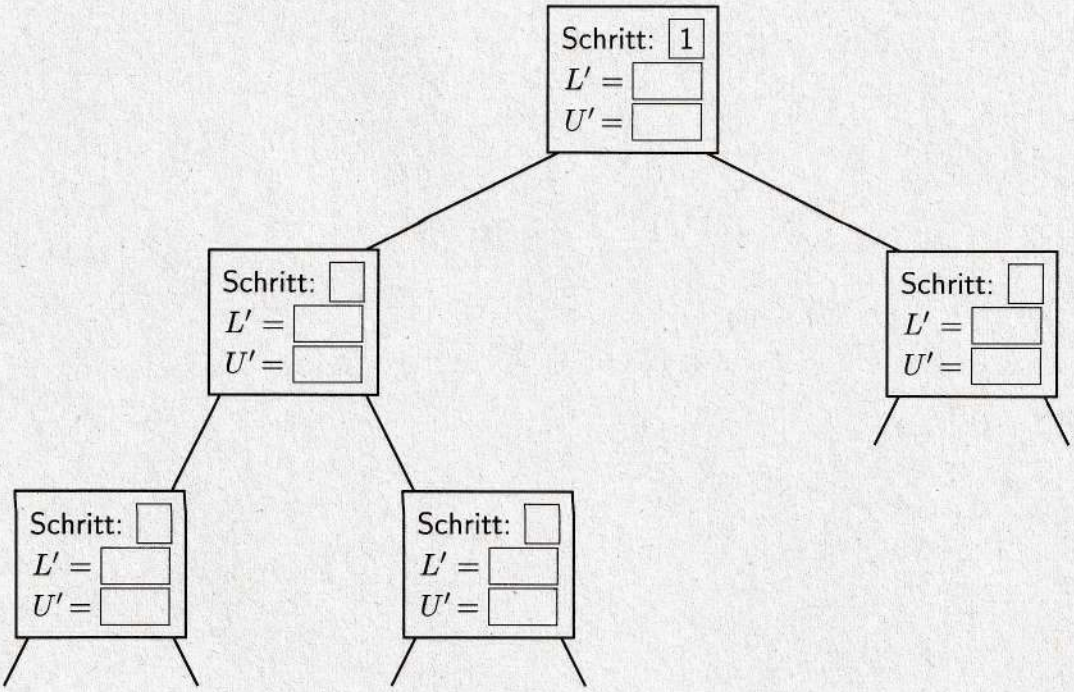
Berechnen Sie die Wert-Gewichts-Verhältnisse aller Gegenstände und tragen Sie diese in obiger Tabelle ein. Geben Sie die Reihenfolge an, in der die Gegenstände betrachtet werden, wenn Sie den Branch-and-Bound Algorithmus der Vorlesung anwenden.

b) (9 Punkte)

Wenden Sie den verbesserten Branch-and-Bound-Algorithmus[1] aus der Vorlesung auf die Instanz an. Nutzen Sie dabei die Depth-first Strategie zur Auswahl des nächsten Teilproblems. Verwenden Sie zur Lösung der Aufgabe den leeren Branch-and-Bound Baum auf der nächsten Seite.

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. Beachten Sie, dass der Baum nach Bedarf erweitert werden muss und zeichnen Sie die zusätzlich benötigten Kanten und Knoten.

[1] Zur Erinnerung: Die untere Schranke wird durch einen Greedy-Algorithmus berechnet, für die obere Schranke können Gegenstände auch partiell eingepackt werden. Die Reihenfolge, in der die Gegenstände betrachtet werden, ergibt sich aus den Wert-Gewichts-Verhältnissen (s. Aufgabe 3(a)).



c) (2 Punkte)

Geben Sie die optimale Lösung und den zugehörigen Lösungswert an.

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.813 Algorithmen und Datenstrukturen 1 VU 6.0

2.Übungstest SS 2016

2. Juni 2016

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname:		Vorname:	
Matrikelnummer:		Unterschrift:	

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	16	22	12	50
Erreichte Punkte:				

Viel Erfolg!

Rep 2 – 2018

Aufgabe A1: Hashtabellen **(16 Punkte)**

Fügen Sie die folgenden Zahlen in die jeweiligen Hashtabellen ein, indem Sie die angegebenen Hashfunktionen und Strategien für die Kollisionsbehandlung benutzen.

a) (4 Punkte)

Einzufügende Zahl: 26
Kollisionsbehandlung: Double Hashing **ohne** die Verbesserung nach Brent

Hashfunktionen:

$h_1(k) = k \bmod 11$
$h_2(k) = (k \bmod 8) + 1$

Hashtabelle:

0	1	2	3	4	5	6	7	8	9	10
33				37		17	84	30		

b) (4 Punkte)

Einzufügende Zahl: 26
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:

$h_1(k) = k \bmod 11$
$h_2(k) = (k \bmod 5) + 1$

Hashtabelle:

0	1	2	3	4	5	6	7	8	9	10
33				37		17	84	30		

Aufgabe A-1)

a) $h_1(k) = k \bmod 11$

$h_2(k) = (k \bmod 8) + 1$ ⟶ Double Hashing $m = 11$

$h(k,i) = (h_1(k) + i\,h_2(k)) \bmod 11$

$h_1(26) = 4$

$h_2(26) = 3$

$h(26,0) = 4$ besetzt

$h(26,1) = 7$ besetzt

$h(26,2) = 10$ frei

b) $h_1(k) = k \bmod 11$

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

$h(k,i) = (h_1(k) + i\,h_2(k)) \bmod 11$

Verbesserung nach Brent

1. Zuerst überprüfen ob j frei ist

$h(26,0) = 4$ besetzt

$q = 4$

$k' = 37$

2a) j ist frei

k einfach in $h(k, i+1)$ einsetzen

2b) j ist nicht frei

checken ob j' frei ist und sich verschieben lässt, sonst 2a

$h_2(26) = 2$ $j = 6$ besetzt

$h_2(37) = 3$ $j = 7$ besetzt

↓

$h(26,1) = 6$ besetzt.

$q = 6$

$k' = 17$

$j = (q + h_2(k)) \bmod 11$

$j' = (q + h_2(k')) \bmod 11$

$h_2(26) = 2$ $j = 8$ besetzt

$h_2(17) = 3$ $j' = 9$ frei

k' (17) wird auf j' (9) verschoben

k ersetzt k' an Stelle q (6)

c)

I Multiplikationsmethode $h(k) = \lfloor m \cdot (\lceil k \cdot A \rceil - \lfloor k \cdot A \rfloor) \rfloor$

A sollte eine irrationale Zahl sein, optimalerweise der goldene Schnitt

Hier: $A = \frac{\pi}{2}$ → geeignet ✓

II Multiplikationsmethode

$A = 2$ → ungeeignet, da nicht irrational ×

Es entstehen sehr viele Kollisionen

III Divisionsrest-Methode

m = Potenz von 2 → ungeeignet, da keine Primzahl ×

IV Divisionsrest-Methode

mit double-Hashing

$h(i,k) = (h_1(k) + i \cdot h_2(k)) \bmod m$

↑	↑	↑
$23 \in \mathbb{P}$	$6 \notin \mathbb{P}$	$27 \notin \mathbb{P}$
✓	×	×

$h_1(k) = k \bmod 23$ → da $23 < 27$ können bei $h(k,0)$ nicht alle Indizes belegt werden

$h_2(k) = (k \bmod 6) + 1$ → 6 teilt 23 nicht ✓ → keine ggT

Hash kann nicht 0 ergeben ✓

Rep 2 - 2018

c) (8 Punkte)

Gegeben sind im Folgenden mehrere Hashverfahren. Geben Sie zu jedem an, ob es gut funktionieren würde, oder ob es zu Problemen kommen könnte. Begründen Sie ihre Antworten.

I) (2 Punkte)

Multiplikationsmethode
Tabellengröße $m = 2^{10}$
Faktor $A = \frac{\pi}{2}$

II) (2 Punkte)

Multiplikationsmethode
Tabellengröße $m = 13$
Faktor $A = 2$

III) (2 Punkte)

Divisions-Rest-Methode
Tabellengröße $m = 2^5$
$h(k) = (k + 1) \mod m$

IV) (2 Punkte)

Divisions-Rest-Methode mit Double Hashing
Tabellengröße $m = 27$
$h_1(k) = k \mod 23$
$h_2(k) = (k \mod 6) + 1$

Aufgabe A2: Bäume **(22 Punkte)**

a) (12 Punkte)

Gegeben sei der Wurzelknoten `root` eines (vermeintlichen) B-Baumes. Schreiben Sie eine rekursive Funktion `int checkBTree(node, leftBound, rightBound)` in detailliertem Pseudocode, die überprüft ob der Baum tatsächlich ein gültiger B-Baum ist. Die Funktion soll die Höhe des Baumes zurückgeben oder -1 falls es sich um keinen gültigen B-Baum handelt.

Sie können davon ausgehen, dass jeder Knoten für sich genommen ein korrekter Knoten ist, d.h. die Anzahl der Schlüssel/Kinder, sowie die Ordnung der Schlüssel innerhalb eines Knotens ist korrekt.

Die Parameter der Funktion sind wie folgt definiert:

- `node` ist ein Knoten des B-Baumes,
- `leftBound` und `rightBound` geben das geschlossene Intervall an, in dem sich die Schlüssel des übergebenen Knotens befinden dürfen.

Die Datenstruktur eines B-Baum-Knotens r ist wie folgt definiert:

- `r.key[x]`, $1 \leq x$ — Ein Array das die Schlüssel des Knotens speichert.
- `r.child[x]`, $0 \leq x$ — Ein Array das Verweise auf die Kindknoten speichert.
- Blätter sind Knoten mit `r.key.size() == r.child.size() == 0`.

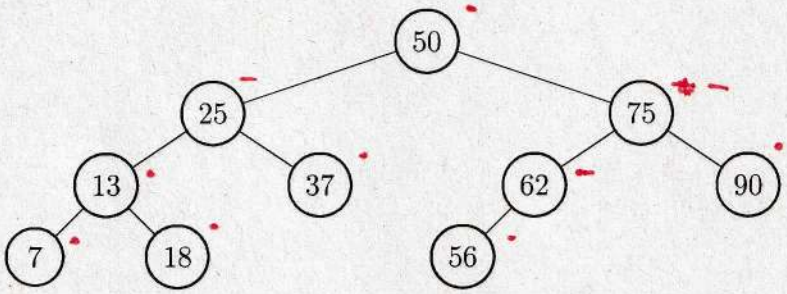
Arrays erlauben Ihnen ausschließlich die folgenden Operationen:

- `a[x]` — Indexzugriff auf das Element an Position x des Arrays a.
- `a.size()` — Gibt die Anzahl an gespeicherten Elementen im Array a zurück.

Sie können davon ausgehen, dass der B-Baum **paarweise verschiedene** ganzzahlige Schlüssel aus $[0, 100]$ speichert. Die Funktion wird mit `checkBTree(root, 0, 100)` aufgerufen.

Rep2,2018

b) (10 Punkte) Gegeben sei folgender AVL-Baum:

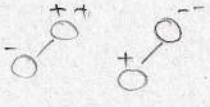


Teilen Sie alle noch nicht vorhandenen ganzen Zahlen aus dem Intervall $[0, 100]$ in Teilintervalle auf, sodass jede Zahl aus demselben Teilintervall an der gleichen Position in dem gegebenen Baum eingefügt werden würde. Tragen Sie nun diese Intervalle zusammen mit der entsprechenden Einfügestelle sowie die Art der Reorganisation, welche nach dem Einfügen einer Zahl in einem solchen Intervall ausgeführt werden muss, in die folgende Tabelle ein. Die erste Zeile in der Tabelle ist bereits exemplarisch ausgefüllt.

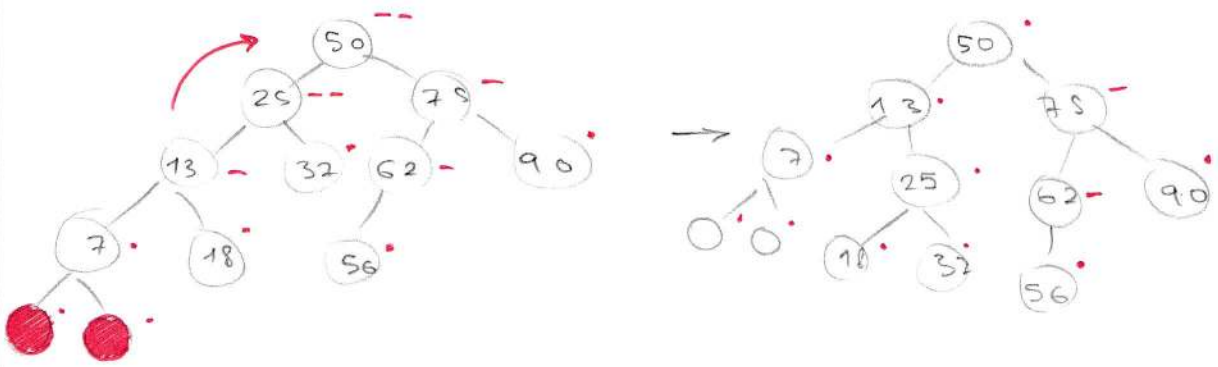
Intervall	Einfügestelle	Art der Reorganisation		
		keine	einfache Rotation	doppelte Rotation
0–6	links von 7		✓	
8–12	rechts von 7		✓	
14–17	links von 18			✓
19–24	rechts von 18			✓
26–36	links von 37	X		
38–49	rechts von 37	X		
51–55	links von 56		✓	
57–61	rechts von 56			✓
63–74	rechts von 62	X		
76–89	links von 90	X		
91–100	rechts von 90	X		

Einfache Rot: Gleiche Vorzeichen

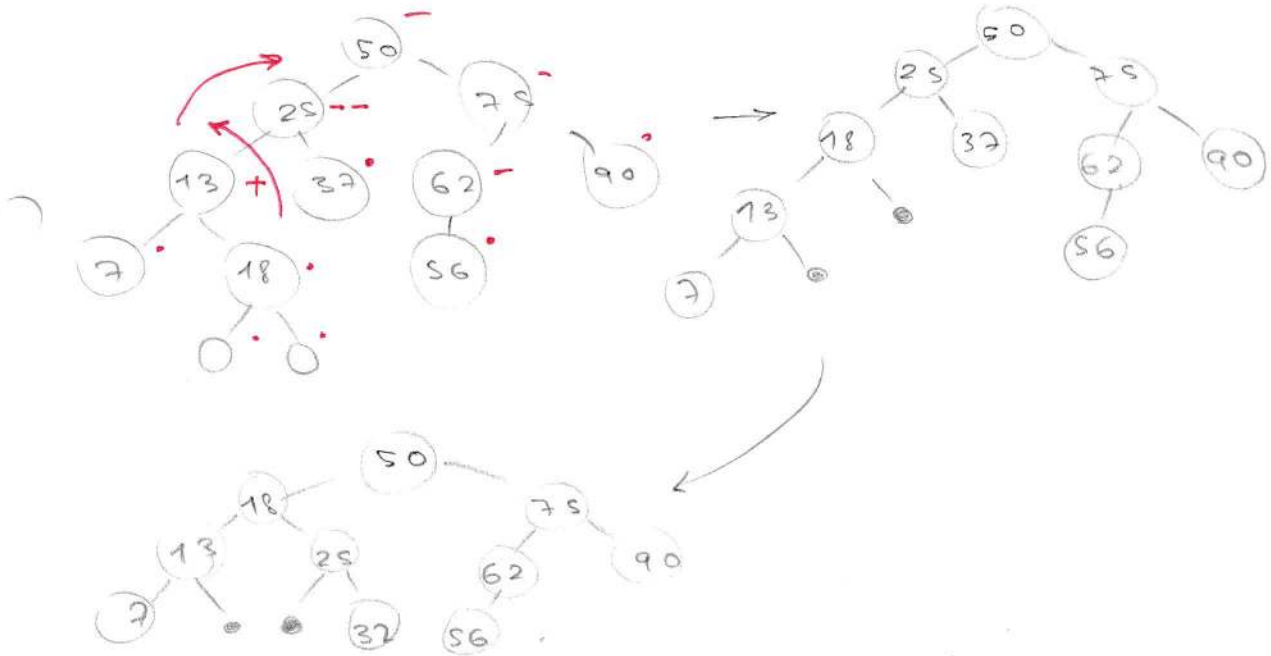
Doppelte Rotation: Eltern und Kind verschiedene Vorzeichen bei Balance



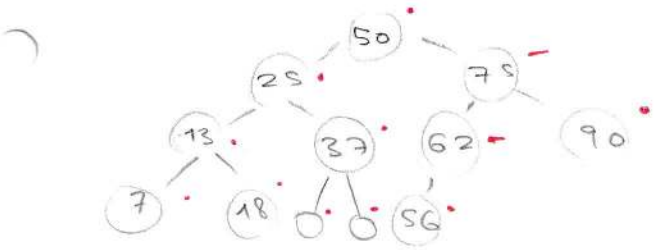
Links und rechts von 7



Links und rechts von 18

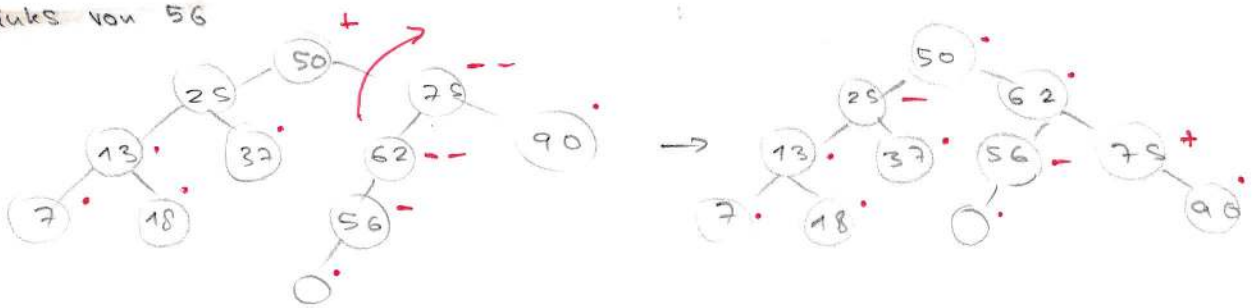


Links und rechts von 37

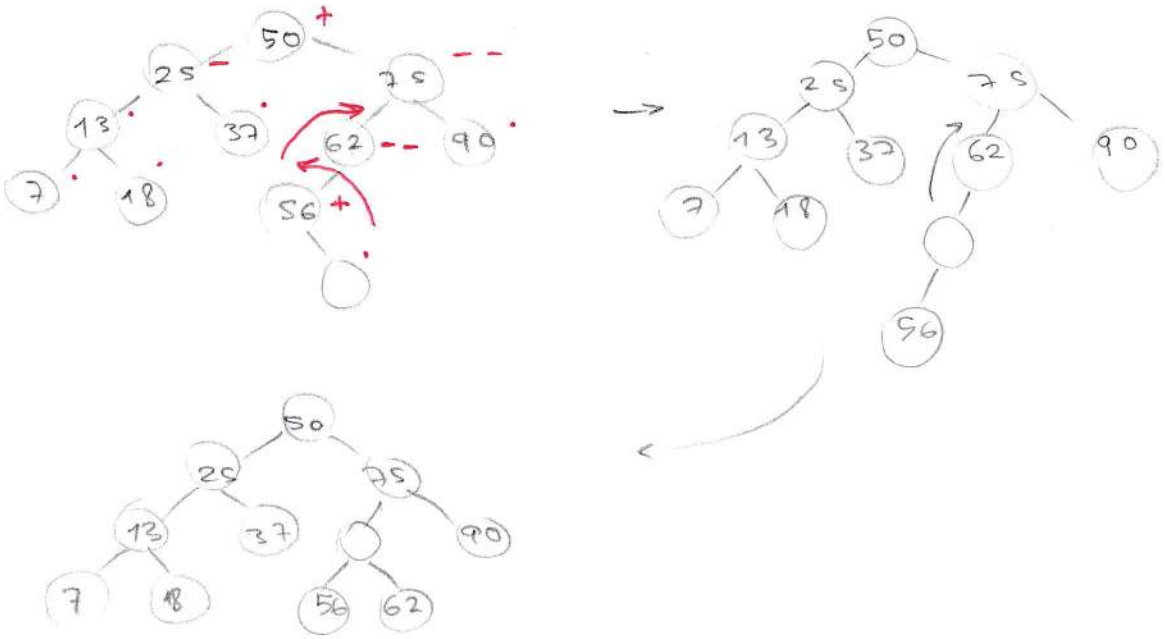


keine Rotation notwendig

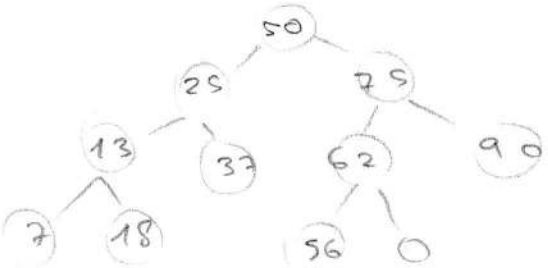
Links von 56



Rechts von 56

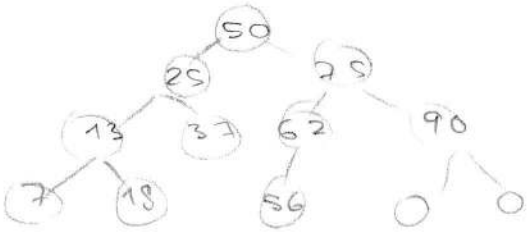


Rechts von 63



keine Rotation

Links und Rechts von 90

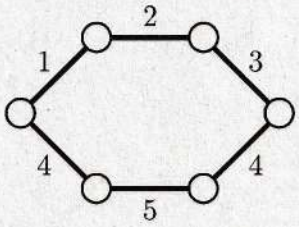


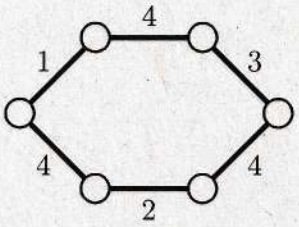
keine Rotation

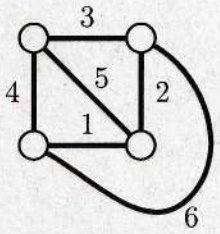
Aufgabe A3: Greedy Algorithmen **(12 Punkte)**

a) (6 Punkte)

Geben Sie zu jedem der folgenden drei gewichteten Graphen an, wieviele minimale Spannbäume der Graph hat.







Anzahl: **Anzahl**: **Anzahl**:

b) (6 Punkte)

Betrachten Sie das sogenannte k-Minimum Spanning Tree (k-MST) Problem, welches wie folgt definiert ist.

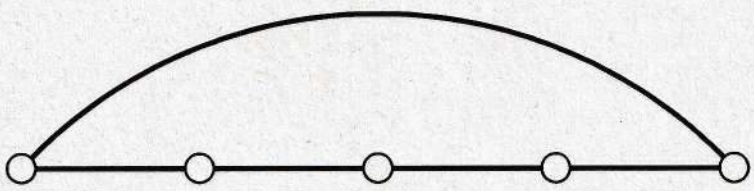
Eine Instanz des k-MST Problems ist gegeben durch einen kantengewichteten zusammenhängenden Graph $G = (V, E)$ mit Kantengewichten $c(e) : E \to \mathbb{R}$ und eine natürliche Zahl k mit $k > 2$. Gesucht ist ein zusammenhängender Teilbaum von G mit genau k Knoten und minimalem Kantengewicht.

Was würde passieren, wenn Sie die bekannten Algorithmen von Prim bzw. Kruskal auf das Problem anwenden und beide Algorithmen nach $k-1$ hinzugefügten Kanten abbrechen? Zeigen Sie, dass beide Algorithmen das falsche bzw. kein optimales Ergebnis liefern, indem Sie die folgenden Gegenbeispiele vervollständigen.

I) (3 Punkte)

Geben Sie für den folgenden Graphen Kantengewichte an, sodass der modifizierte Algorithmus von Kruskal für $k = 4$ ein falsches Ergebnis liefert.

II) (3 Punkte) Geben Sie für den folgenden Graphen Kantengewichte und einen Startknoten an, sodass der modifizierte Algorithmus von Prim für $k = 4$ beginnend mit dem angegebenen Startknoten ein falsches Ergebnis liefert.



Technische Universität Wien
Institut für Computergraphik und Algorithmen
Algorithms and Complexity Group

186.813 Algorithmen und Datenstrukturen 1 VU 6.0

1.Übungstest SS 2016

28. April 2016

Machen Sie die folgenden Angaben bitte in deutlicher Blockschrift:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Legen Sie während der Prüfung Ihren Ausweis für Studierende vor sich auf das Pult.

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie bitte dokumentenechte Schreibgeräte (keine Bleistifte!).

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

	A1:	A2:	A3:	Summe:
Erreichbare Punkte:	20	20	10	50
Erreichte Punkte:				

Viel Erfolg!

(Rep 1 2018)

Aufgabe A1: Algorithmenanalyse **(20 Punkte)**

a) (10 Punkte) Tragen Sie für die Codestücke `FunktionA` und `FunktionB` jeweils die Laufzeit und den Rückgabewert (z) in Abhängigkeit von n in Θ-Notation in die nachfolgende Tabelle ein.

	`FunktionA`	`FunktionB`
Laufzeit	n	n^2
Rückgabewert (z)	n	n^3

`FunktionA(`n`)`:
$x \leftarrow 50000$
while $x > 1$
  for $j \leftarrow 1$ bis $\lfloor \frac{n}{50} \rfloor$
    $z \leftarrow 2j$
  $x \leftarrow \frac{x}{5}$
return z

`FunktionB(`n`)`:
$x \leftarrow 1$
$y \leftarrow 0$
$z \leftarrow 0$
while $x < n^2$
  while $y \leq x$
    $y \leftarrow y + 1$
    $z \leftarrow z + n$
  $x \leftarrow x + 1$
return z

Lösung: $z = 2 \cdot \lfloor \frac{n}{50} \rfloor = \Theta(n)$

$z = n^2 \cdot n = \Theta(n^3)$

Aufgabe A1:

Funktion A(n)

```
x = 50,000
while (x > 1)
    for j ← 1 bis ⌊n/50⌋
        z = 2j
    x = x/5
return z
```

→ Äquivalent zu

```
for (j=1 ; j ≤ ⌊n/50⌋ ; j++)
```

Ausgabe z:

weil z nicht gespeichert wird: letzte Iteration: $z = 2 \cdot \lfloor \frac{n}{50} \rfloor$

Laufzeit

While Loop:

Abbruch wenn $x = 1$

Wie oft muss 50,000 durch 5 dividiert werden damit $= 1$?

50.000
10.000
2.000
400
80
16
3,2 ↓ (6x)

$50000 \cdot \frac{1}{5}^k \in 1$

$\frac{1}{5}^k \leq \frac{1}{50000}$

$k \ln\left(\frac{1}{5}\right) = \ln\left(\frac{1}{50000}\right)$

$k = \frac{\ln\left(\frac{1}{50000}\right)}{\ln\left(\frac{1}{5}\right)}$

$k = 6,722706$ ✓

Siehe Differenzengleichungen:

Lineares Wachstum

$y_{n+1} = y_n \pm k \quad \Rightarrow y_0 + n \cdot k = y_n$

Exponentielles Wachstum

$y_{n+1} = y_n \cdot k \quad \Rightarrow y_0 \cdot k^n = y_n$

y_0 = irgendein Wert an

$y_1 = y_0 \cdot k$

$y_2 = y_1 \cdot k$

$y_3 = y_2 \cdot k$

↓

$y_1 = m \cdot k$

$y_2 = (m \cdot k) \cdot k$

$y_3 = ((m \cdot k) \cdot k) \cdot k = m \cdot k^3$

↓

$y_n = m \cdot k^n$

Aufgabe A4:

Funktion A(n)

```
x = 50.000
while (x > 1)
    for j ← 1 bis n/⌊50⌋
        z = 2j
    x = x/5
return z
```

→ Äquivalent zu

```
for (j=1 ; j ≤ n/⌊50⌋ ; j++)
```

Ausgabe z:

Weil z nicht gespeichert wird: letzte Iteration: $z = 2 \cdot \lfloor \frac{n}{50} \rfloor$

Laufzeit

While Loop:

Abbruch wenn $x = 1$

Wie oft muss 50,000 durch 5 dividiert werden damit $= 1$?

50,000
10 000
2 000
400
80
16
3,2 ↓

(6x)

$50000 \cdot \frac{1}{5}^k \leq 1$

$\frac{1}{5}^k \leq \frac{1}{50000}$

$k \ln\left(\frac{1}{5}\right) = \ln\left(\frac{1}{50\,000}\right)$

$k = \frac{\ln\left(\frac{1}{50\,000}\right)}{\ln\left(\frac{1}{5}\right)}$

$k = 6,722706$ ✓

Siehe Differenzvergleichungen:

Lineares Wachstum

$y_{n+1} = y_n \pm k \quad \Rightarrow y_0 + n \cdot k = y_n$

Exponentielles Wachstum

$y_{n+1} = y_n \cdot k \quad \Rightarrow y_0 \cdot k^n = y_n$

y_0 = irgendein Wert m

$y_1 = y_0 \cdot k$

$y_2 = y_1 \cdot k$

$y_3 = y_2 \cdot k$

↓

$y_1 = m \cdot k$

$y_2 = (m \cdot k) \cdot k$

$y_3 = ((m \cdot k) \cdot k) \cdot k = m \cdot k^3$

↓

$y_n = m \cdot k^n$

innere for-loop: $\Theta\left(\left\lfloor \frac{n}{50} \right\rfloor\right)$

Somit insgesamt:

```
x = 50 000                    Θ(1)
while x > 1                   ┐
    for j: 1 bis ⌊n/50⌋        ├ Θ(n/50)   ┐
        z = 2j                 Θ(1)        ├ Θ(6)
    x = x/5                    Θ(1)        ┘
return z
```

Somit insgesamt: $\Theta\left(\left\lfloor \frac{n}{50} \right\rfloor \cdot 6\right)$

Bestimmung der Iterationszyklen einer Schleife:

```
for (x=0; x<n; x++)
```

bezeichnungsweise

```
x = 0
while (x < n)
    x++
```

→ Schleifenabbruch wenn $x = n$

$x + k \cdot 1 = n$

$x = 0$ weil immer $x = x + 1$

$0 + k \cdot 1 = n$

$k = n$

```
for (x = 1; x < n; x++)
```

bzw

```
x = 1
while (x < n)
    x++
```

→ Abbruch wenn $x = n$

$x + k \cdot 1 = n$

$1 + k \cdot 1 = n$

$k \cdot 1 = n - 1$

$k = n - 1$

```
for (x = 1; x < n^2; x++)
```

bzw

```
x = 1
while (x < n^2)
    x++
```

→ Abbruch wenn $x = n^2$

$x + k \cdot 1 = n^2$

$1 + k \cdot 1 = n^2$

$k \cdot 1 = n^2 - 1$

$k = n^2 - 1$

Funktion B(n):

```
x = 1
y = 0
z = 0
while (x < n^2)
    while (y ≤ x)
        y = y + 1
        z = z + n
    x = x + 1
return z
```

$\Theta(1)$

$O(2)$

$\Theta(n^2-1)$

Laufzeit:

Äußere While-Schleife

Abbrechen wenn $x = n^2$ $\longrightarrow$ $n^2 - 1$ Iterationen

Innere while-Schleife

Abbrechen wenn $y > x$ (y außen gespeichert)

Anfangsstand: $x = 1$ $y = 0$

innere $y = 1$, $y = 2$

äußere $x = 2$

innere $y = 3$

äußere $x = 3$

innere $y = 4$

Die innere Schleife braucht nur 1-2 Iterationen bis $y > x$

also $O(2)$

Rückgabewert:

z wird $O((n^2-1) \cdot 2)$ Mal um n erhöht:

$\underbrace{((n^2-1) \cdot 2)}_{k} \cdot n = (2n^2 - 2) \cdot n = 2n^3 - 2n = \Theta(n^3)$

b) (6 Punkte) Beantworten Sie die nachfolgenden Fragen und begründen Sie jeweils Ihre Antwort in **wenigen** Worten!

- Wenn die Worst-Case-Laufzeit eines Algorithmus in $\Theta(n^2)$ liegt, ist es dann möglich, dass seine Laufzeit für **manche** Instanzen in $O(n)$ liegt?

- Wenn die Worst-Case-Laufzeit eines Algorithmus in $\Theta(n^2)$ liegt, ist es dann möglich, dass seine Laufzeit für **alle** Instanzen in $O(n)$ liegt?

c) (4 Punkte) Ordnen Sie folgende Funktionen nach Dominanz, beginnend mit der asymptotisch am schwächsten wachsenden. Es genügt die Funktionen zu reihen, ein Beweis der Gültigkeit der Relationen ist nicht erforderlich.

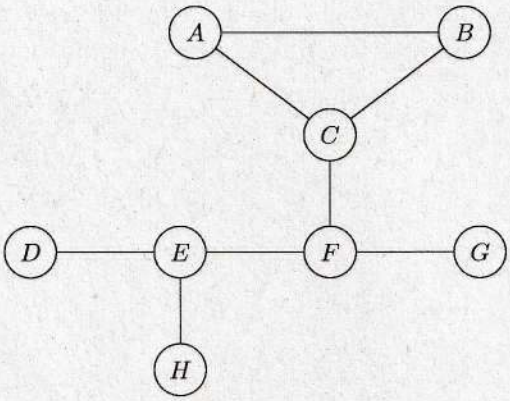
$$\log\left(n^{15}\right), \quad \left(\frac{3}{2}\right)^n, \quad n - n^3 + 7n^5, \quad \left(\frac{1}{3}\right)^n, \quad \sqrt{n^8}, \quad n^2 \left(\log n\right)^2$$

Rep 2018 - Teil 1

Aufgabe A2: Graphen **(20 Punkte)**

a) (12 Punkte) Betrachten Sie den nachfolgenden Graphen $G = (V, E)$. Gehen Sie alle Knoten $v \in V$ durch und überlegen Sie jeweils, ob es möglich ist, dass sowohl die Breiten- als auch die Tiefensuche (jeweils gemäß dem Pseudocode aus den Vorlesungsfolien) mit v als Startknoten den Graphen in derselben Reihenfolge abarbeiten.

- Ist das für den jeweils betrachteten Knoten v **möglich**, dann geben Sie eine passende Abarbeitungsreihenfolge an.
- Ist das für den jeweils betrachteten Knoten v **nicht möglich**, dann soll dies stattdessen durch Aufteilen der Knoten in zwei Mengen X und Y ($X \cup Y = V, X \cap Y = \emptyset$) bewiesen werden:
  - Für die Breitensuche soll gelten, dass immer alle Knoten aus X vor allen Knoten aus Y berücksichtigt werden.
  - Für die Tiefensuche muss es aber immer ein Knotenpaar $x \in X$ und $y \in Y$ geben, sodass y vor x besucht wird.



Knoten	Abarbeitungsreihenfolge	X	Y
A			
B			
C			
D			
E			
F			
G			
H			

Graph Traversal

BFS Breitensuche → nach Levels

DFS Tiefensuche → nach Tiefe

a) Ziel ist es eine Reihenfolge zu finden nach der sowohl BFS als auch DFS traversieren würden.

A: A, B, C, F, G, E, H, D

B: B, A, C, F, G, E, H, D

C: C, B, A, F, G, E, H, D

D: D, E, H, F, G, C, A, B

E: E, D, H, F, G, C, A, B

H: H, E, D, F, G, C, A, B

(Wir dürfen entscheiden welche Kante wir zuerst nehmen)

Problematisch bei (G):

Deshalb 2 Knotendisjunkte Mengen bilden: X und Y

BFS: Zuerst X vollständig abarbeiten, dann Y

TFS: Ein Knotenpaar yY sodass y vor X

Partitionierung:

$X = \{G, F, E, C\}$

↓

DFS würde nach E D, H nehmen

Ein Knotenpaar ($y = D$, $x = E$)

$y = \{A, B, D, H\}$

Problematisch bei (F):

Partitionierung:

$X = \{F, G, C, E\}$

$y = \{A, B, D, H\}$

b) Topologische Sortierung

$\{D, E\}, B, \{A, F\}, C, \{G, H\}$

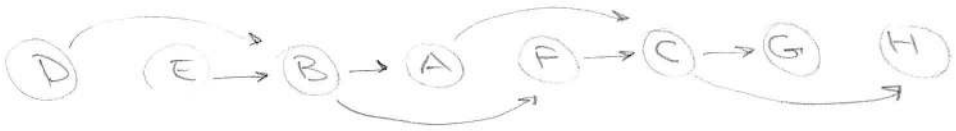
↑

[Es gibt 2 Quellen: $deg(v) = 0$
3 Knotenpaare dürfen aber vertauscht werden. Dadurch in Summe:
2^3 Möglichkeiten]

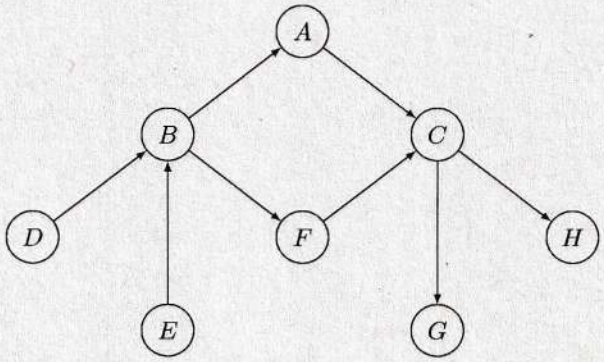
D	A	G
E	F	H
0	0	0

$\rightarrow 2^3$

Beispiel:



b) (8 Punkte) Gegeben sei der folgende gerichtete Graph:



- Finden Sie für diesen Graphen eine topologische Sortierung. (2 Punkte)

- Wie viele unterschiedliche topologische Sortierungen gibt es für diesen Graphen? (3 Punkte)

- Zeichnen Sie **eine** zusätzliche Kante ein, sodass keine gültige topologische Sortierung mehr möglich ist. Begründen Sie in einem Satz, warum das so ist. (3 Punkte)

Topologische Sortierungen sind nur in einem DAG (directed, acyclic graph) möglich.

Zusätzliche Kante müsste Kreis bilden.

Ein Kreis würde keine topologische Ordnung mehr zulassen weil dann:

- $\exists$ Kante von (i) nach (j) obwohl $j < i$ in Ordnung

Beispiel: (F) → (E)

Aufgabe A3: Sortieren **(10 Punkte)**

a) (2 Punkte) Das Array $A = [15, 18, 6, 2, 9, 11]$ wird mittels Selection-Sort gemäß dem Pseudocode aus den Vorlesungsfolien sortiert. Kreuzen Sie in nachfolgender Liste jene Arrays an, die nach einer oder mehreren Iterationen der äußersten Schleife entstehen können.

- ☐ $A = [2, 6, 18, 15, 9, 11]$
- ☐ $A = [6, 15, 18, 2, 9, 11]$
- ☐ $A = [11, 15, 18, 6, 2, 9]$
- ☐ $A = [2, 6, 9, 11, 18, 15]$

b) (8 Punkte) Im Folgenden seien vier Arrays A, B, C, D mit jeweils n Elementen gegeben:

- A: aufsteigend sortiert
- B: absteigend sortiert
- C: jedes Element an einer ungeraden Position ist kleiner als jedes Element an einer geraden Position (z.B. $5, 4, 6, 2, 8, 1, 7, 3$)
- D: ausgehend von einem aufsteigend sortierten Array wird jedes Element an gerader Position mit dem direkt darauffolgenden Element ungerader Position vertauscht (z.B. $2, 1, 4, 3, 6, 5, 8, 7$)

Geben Sie für jedes Array die Laufzeit von Insertion-Sort (gemäß dem Pseudocode aus den Vorlesungsfolien) in Θ-Notation an:

Array	$\Theta(\cdot)$
A	
B	
C	
D	

Nehmen Sie an, dass Arrays immer mit 0 beginnend indiziert sind, 0 gilt als gerade.