| evaexam | Algorithmen und Datenstrukturen 2 SS21 | evasys |
|---|---|---|

Institut für Betriebssysteme und Rechnerverbund — Prof. Dr. Sándor Fekete / Matthias Konitzny
Algorithmik — Sommersemester 2021
Technische Universität Braunschweig

Bitte so markieren: ☐ ☒ ☐ ☐ ☐ Bitte verwenden Sie einen Kugelschreiber oder nicht zu starken Filzstift. Dieser Fragebogen wird maschinell erfasst.
Korrektur: ☐ ■ ☐ ☒ ☐ Bitte beachten Sie im Interesse einer optimalen Datenerfassung die links gegebenen Hinweise beim Ausfüllen.

Bitte ausfüllen (Die Angabe des Namens ist freiwillig.):

Vorname: ______________________

Nachname: ______________________

Für die eindeutige Zuordnung der Prüfung übertragen Sie bitte Ihre Prüfungsteilnehmer-ID gewissenhaft in die dafür vorgesehenen Felder. Alle Seiten sind vollständig individualisiert und nicht mit anderen Prüfungen tauschbar.

Prüfungsteilnehmer-ID für den Prüfungsbogen Nr.: 0:

0 ☐ ☐ ☐ ☐ ☐ ☐ ☐
1 ☐ ☐ ☐ ☐ ☐ ☐ ☐
2 ☐ ☐ ☐ ☐ ☐ ☐ ☐
3 ☐ ☐ ☐ ☐ ☐ ☐ ☐
4 ☐ ☐ ☐ ☐ ☐ ☐ ☐
5 ☐ ☐ ☐ ☐ ☐ ☐ ☐
6 ☐ ☐ ☐ ☐ ☐ ☐ ☐
7 ☐ ☐ ☐ ☐ ☐ ☐ ☐
8 ☐ ☐ ☐ ☐ ☐ ☐ ☐
9 ☐ ☐ ☐ ☐ ☐ ☐ ☐

## 1. Greedy Algorithmen - Anwenden (7 Punkte)

Betrachte folgende Instanz für Maximum Knapsack:

| $i$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $z_i$ | 12 | 18 | 9 | 10 | 16 |
| $p_i$ | 12 | 9 | 4 | 11 | 7 |
| | 1 | 2 | 0,.. | 2,.. | |

mit $Z = 31$

Wende den Greedy-Algorithmus für Maximum Knapsack auf diese Instanz an.

1.1 In welcher Reihenfolge werden die Objekte betrachtet?
Trage die Objektnummern in der korrekten Reihenfolge in die Kästchen ein.

4, 1, 2, 3, 5

9

11 + 12 +

1.2 Gib den Gesamtwert, sowie das Gesamtgewicht der gefundenen Lösung an.

Ist die gefundene Lösung optimal? Begründe deine Antwort.

Wert: 27
Gewicht: 31

Ja, da $\text{Lösung}_{\text{Fractional-Greedy}} = \left\lfloor 12 + 11 + \frac{31-10-12}{18} \cdot 9 \right\rfloor = \lfloor 23 + 4{,}5 \rfloor = \lfloor 27{,}5 \rfloor = 27 = \text{Lösung}_{\text{Greedy-Algorithmus}}$

## 2. Greedy Algorithmen - Entwurf (8 Punkte)

Betrachte die fraktionale Variante von Integer Knapsack, d.h. Objekte können beliebig oft und anteilig benutzt werden.

2.1 Gib einen Greedy-Algorithmus an, der dieses Problem in linearer Zeit (O(n)) löst, also ohne Sortieren auskommt.

Begründe kurz die Korrektheit deines Algorithmus und begründe außerdem, warum dein Algorithmus die Zeit einhält.

Input: $z_1, \dots, z_n$ $Z$, $p_1, \dots, p_n$

output: $x_1, \dots, x_n$

Integerknapsack ($z_1, \dots, z_n$, $Z$, $p_1, \dots, p_n$)

$k_{max} := 1$

for $i = 2$ up to $n$

if $\left(\frac{z_{k_{max}}}{p_{k_{max}}} > \frac{z_i}{p_i}\right)$ then

$k_{max} := i$

$x_{k_{max}} = \frac{Z}{z_{k_{max}}}$

$L = x_{k_{max}} \cdot p_{k_{max}}$

return $x_1, \dots, x_{k_{max}}, \dots, x_n$; $L$

O(n)

Korrektheit: der Algorithmus ist korrekt

Angenommen: $k_{max}$, was der Algorithmus gefunden hat, ist kein effizientestes Item.

D.h. es existiert ein effizienteres Item $J$, also $\frac{z_J}{p_J} < \frac{z_{k_{max}}}{p_{k_{max}}}$. Nach Algorithmus $k_{max} := J$ und der Output würde anders als was angenommen

↳ $k_{max}$ muss das effizienteste Item sein

Laufzeit O(n), da die meisten Zeit in for-Schleife von 2 bis n liegt.

## 3. Dynamic Programming - Anwendung (9 Punkte)

Betrachte folgende Instanz für Subset Sum.

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $z_i$ | 7 | 3 | 6 | 7 | 4 | 5 |

und $Z = 15$.

Wende das dynamische Programm für Subset Sum auf diese Instanz an. Zeichne dir dafür eine Tabelle der folgenden Form. Der Eintrag in Zeile i und Spalte x entspricht dem Wert S(x,i)

| $i \backslash x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 5 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 6 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

3.1 Wie viele Einsen stehen in der ersten Zeile (i = 0)?
- [ ] 0
- [x] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5

3.2 Wie viele Einsen stehen in der zweiten Zeile (i = 1)?
- [ ] 0
- [ ] 1
- [x] 2
- [ ] 3
- [ ] 4
- [ ] 5

3.3 Wie viele Einsen stehen in der dritten Zeile (i = 2)?
- [ ] 0
- [ ] 1
- [ ] 2
- [x] 3
- [ ] 4
- [ ] 5

3.4 Wie viele Einsen stehen in der vierten Zeile (i = 3)?
- [ ] 4
- [ ] 5
- [ ] 6
- [x] 7
- [ ] 8
- [ ] 9

3.5 Wie viele Einsen stehen in der fünften Zeile (i = 4)?
- [ ] 4
- [ ] 5
- [ ] 6
- [ ] 7
- [x] 8
- [ ] 9

3.6 Wie viele Einsen stehen in der sechsten Zeile (i = 5)?
- [x] 10
- [ ] 11
- [ ] 12
- [ ] 13
- [ ] 14
- [ ] 15

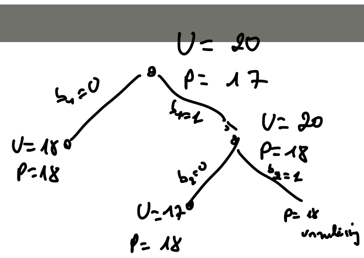
3.7 Wie viele Einsen stehen in der siebten Zeile (i = 6)?
- [ ] 10
- [ ] 11
- [ ] 12
- [ ] 13
- [x] 14
- [ ] 15

3.8 Ist die gegebene Instanz von Subset Sum lösbar? Begründe deine Antwort.

Ja, da in der letzten Spalte der Tabelle steht 1, d.h. 15 kann von 6 Items erzeugt werden

unten rechts

## 4. Branch-and-Bound - Anwendung (7 Punkte)




Betrachte folgende Instanz von Maximum Knapsack.

| $i$ | 1 | 2 | 3 |
|---|---|---|---|
| $z_i$ | 10 | 18 | 7 |
| $p_i$ | 12 | 18 | 5 |

mit $Z = 18$

Wende den Branch-and-Bound-Algorithmus für Maximum Knapsack auf diese Instanz an. Zeichne dir dazu den Entscheidungsbaum auf. Beachte dazu folgende Dinge:
- Beschrifte Kanten mit der Auswahl, die getroffen wurde.
- Beschrifte Knoten mit den aktuell besten Schranken.
- Kennzeichne Knoten, falls die aktuelle Auswahl unzulässig ist.

**4.1** Welche obere Schranke wird gefunden, sofern noch kein Element fixiert wurde?

- [ ] 17
- [ ] 18
- [ ] 19
- [x] 20
- [ ] 21
- [ ] 22

**4.2** Welche untere Schranke wird gefunden, sofern noch kein Element fixiert wurde?

- [ ] 15
- [ ] 16
- [x] 17
- [ ] 18
- [ ] 19
- [ ] 20

**4.3** Wie viele Knoten werden im Entscheidungsbaum besucht?

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [x] 5
- [ ] 6
- [ ] 7
- [ ] 8
- [ ] 9

**4.4** Wie viele der besuchten Knoten enthalten eine unzulässige Auswahl?

- [ ] 0
- [x] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5

**4.5** Bei wie vielen der besuchten, zulässigen Knoten ist die untere Schranke mindestens so groß wie die obere Schranke?

- [ ] 0
- [ ] 1
- [x] 2
- [ ] 3
- [ ] 4
- [ ] 5

$p \geq U$

**4.6** Welche Objekte sind in der Lösung enthalten?

- [ ] 1
- [x] 2
- [ ] 3

**4.7** Welchen Wert hat die zurückgegebene Lösung?

- [ ] 5
- [ ] 10
- [ ] 17
- [x] 18
- [ ] 23
- [ ] 30

## 5. Approximation - Modellierung (13 Punkte)

Betrachte das folgende Problem

**Gegeben:** Eine Punktmenge mit $n$ Punkten $\mathcal{P} = \{p_1, p_2, \ldots p_n\} \in \mathbb{R}^2$.

**Gesucht:** Ein Punktpaar $(p_x, p_y)$ aus $\mathcal{P}$ mit Distanz $d(p_x, p_y)$ maximal.

**Hinweis**: Zur Berechnung der Distanz d nutzen wir die Manhattan-Metrik.

Die Manhattan-Metrik berechnet den Abstand zwischen zwei Punkten als Summe der absoluten Differenzen der einzelnen Koordinaten der Punkte. So ist beispielsweise der Abstand zwischen den Punkten $p_1$=(1,2) und $p_2$=(4,6) nach der Manhatten-Metrik $d(p_1,p_2)$ = |1-4| + |2-6| = 7.

5.1 In welcher Komplexitätsklasse liegt das gegebene Problem?

- [x] P
- [x] NP
- [ ] Keine der beiden

5.2 Begründe kurz deine Auswahl zu 5.1.

Das Problem kann immer und optimal in polynomieller Zeit gelöst werden und auch in polynomieller Zeit verifiziert werden, also in $O(n^2)$, da es $\frac{n!}{(n-2)!\,2!} = n.(n-1)$ Vergleiche gibt, sodass die Paare mit größten Distanz gefunden wird.

5.3 Beschreibe einen (1/2)-Approximationsalgorithmus, der das Problem in linearer Zeit (O(n)) löst.

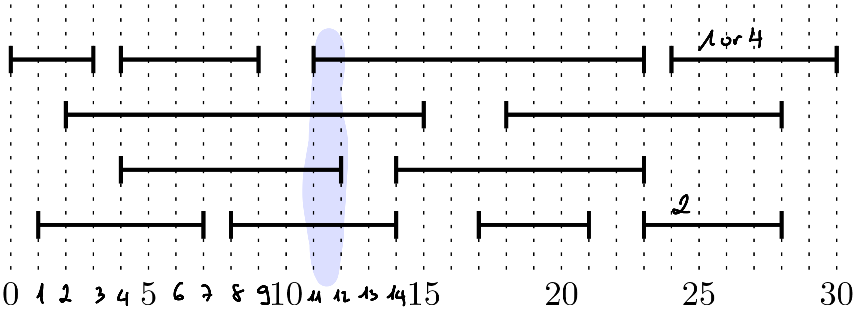
Begründe kurz, warum dein Algorithmus eine (1/2)-Approximation ist.
(Hinweis: Betrachte dazu Instanzen, welche durch ein Quadrat so umschlossen werden können, dass auf jeder Seite des Quadrates mindestens ein Punkt liegt.)

## 6. Hörsaal-Belegung (15 Punkte)

Betrachte das Problem Hörsaal-Belegung:

**Gegeben:** Intervalle $I_1,\ldots, I_n$
**Gesucht:** Größte Teilmenge S von $I_1,\ldots, I_n$, sodass S nur disjunkte Intervalle enthält.

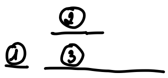
Betrachte nun folgende Instanz.




6.1 Wie viele Intervalle sind in einer optimalen Lösung dieser Hörsaal-Problem-Instanz enthalten?

- ☐ 1
- ☐ 2
- ☐ 3
- ☒ 4
- ☐ 5
- ☐ 6

6.2 Zeige: Es gibt Instanzen, für die das Problem nicht optimal gelöst werden kann, indem sukzessive das kürzeste noch nicht überdeckte Intervall aufgenommen wird




Optimale Lösung: 1 + 3
Greedy Lösung nach: 1 + 2
kürzesten noch nicht überdeckten Intervall

} => Greedy Lösung ist nicht optimal

6.3 Beschreibe kurz, wie das Problem optimal in O(n log n) Zeit gelöst werden kann.

-. Sortiert die Intervalle nach frühestem Ende → O(nlogn)
- man benutzt Greedy Algorithmus, um alle Intervalle durchzulaufen und die Intervalle mit frühestem Ende sowie ohne Überlappung mit vorherigen Intervallen bevorzugterweise aufzunehmen → O(n)

O(n) + O(nlogn) → O(nlogn), da nlogn wächst schneller als n

---

Betrachte nun eine Variante des Problems: Anstatt eine größte Teilmenge von disjunkten Intervallen auszuwählen, wollen wir nun alle Intervalle auf so wenig Räume wie möglich verteilen.

6.4 Betrachte nun noch einmal die gegebene Instanz. Auf wie viele Räume können die Intervalle minimal verteilt werden? Begründe, warum es nicht weniger Räume sein können.

4, da im Zeitraum von 11 bis 12 Uhr (markierter Feld) gibt es 4 Intervalle, die sich gegenseitig überlappen → von 11 bis 12 Uhr braucht man mindestens 4 Räume

=> 4 ist die möglichstkleine Anzahl an benötigten Räumen.

## 6. Hörsaal-Belegung (15 Punkte) [Fortsetzung]

6.5 Beschreibe kurz, wie sich dieses Problem optimal in O(n log n) Zeit lösen lässt.

(+) Sortiert Intervalle nach dem frühesten Start → $O(n\log n)$

(+) man benutzt Greedy Algorithmus, um alle sortierten Intervalle (sowohl Startpunkt als auch Endpunkt zu betrachten) durchzulaufen:

(+) Startpunkt einer Intervalle: → einen schon vergebenen Raum zu geben
↳ falls alle vergebenen Räume schon besetzt sind, einen ganz neuen Raum zugeben

(+) Endpunkt einer Intervalle → diesen Raum freizugeben

→ $O(n)$

→ $O(n) + O(n\log n)$ → $O(n\log n)$ Zeit ist benötigt

## 7. Dynamische Programmierung - Modellierung (10 Punkte)

Gegeben sei ein Holzbrett H der Länge L und eine Preisliste $p_1,\ldots,p_n$, wobei $p_i$ den Preis für ein Holzbrett der Länge i für i = 1,...,n angibt. Wir wollen H in kleinere Stücke teilen, um maximalen Profit zu erhalten. Jedes Stück kann dabei beliebig oft verkauft werden.

Betrachte nun folgende Preisliste.

2

| $i$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $p_i$ | 1 | 4 | 6 | 9 | 12 |
| | 1 | 0,5 | 0,5 | $0{,}4\overline{4}$ | $0{,}41\overline{6}$ |

7.1 Gib den maximalen Profit für ein Holzbrett der Länge i an:

i=0: 0,0 i=1: 1,0 i=2: 4,0 i=3: 6,0 i=4: 9,0 i=5: 12

7.2 Gib den maximalen Profit für ein Holzbrett der Länge i an:

i=6: 13 i=7: 16 i=8: 18 i=9: 21

7.3 Welche Variante von den aus der Vorlesung bekannten Knapsack-Problemen steckt hinter dem Holzbrett-Problem? Begründe deine Antwort.

Integer Knapsack, wobei jedes Objekt (positiv) ganzzahlig genommen wird. ($x_i \in \mathbb{N}$)
da das Kosten entspricht der Länge i, der Wert entspricht dem Profit $p_i$. Ein Holzbrett kann nur entsprechend in kleinen Stücken der Länge 1,2,3,4,5 verteilt werden (nicht zB in Stücken der Länge 2,5 verteilt) => Ein Objekt kann nur integer genommen werden.

## 8. Hashing - Anwendung (7 Punkte)

Betrachte ein anfangs leeres Array A der Größe 11 mit Speicherzellen A[0],...,A[10]. In diesem Array führen wir Hashing mit offener Adressierung mit der folgenden Hashfunktion durch:

$$t(x,i) = 3x^2 + 4 + h(x) \cdot i \bmod 11 \qquad \text{mit} \qquad h(x) = (3x \bmod 10) + 1$$

Dabei ist x ein Schlüssel und i die Nummer des Versuchs, x in eine unbesetzte Speicherzelle des Arrays einzufügen, beginnend mit i=0. Berechne zu jedem der folgenden Schlüssel die Position, die er in A bekommt:

$$\underline{1, 10, 4, 2, 6}$$

8.1 Gib die Position in der Hashtabelle an, die die Schlüssel nach dem Einfügen erhalten haben.

1: 7 10: 8 4: 0 2: 5 6: 2

$h(1) = 4$

$\hookrightarrow t(1,0) = 3 \cdot 1^2 + 4 \bmod 11 = 7$

$h(10) = 1$

$\hookrightarrow t(10,0) = 3 \cdot 10^2 + 4 \bmod 11$
$= 304 \bmod 11 = 7$

$\hookrightarrow t(10,1) = 304 + 1 \bmod 11$
$= 8$

$h(4) = 3$

$\hookrightarrow t(4,0) = (3 \cdot 4^2 + 4) \bmod 11$
$= 8$

$\hookrightarrow t(4,1) = (3 \cdot 4^2 + 4 + 3 \cdot 1) \bmod 11$
$= 0$

$h(2) = 7$

$\hookrightarrow t(2,0) = (3 \cdot 2^2 + 4) \bmod 11 = 5$

$h(6) = (3 \cdot 6 \bmod 10) + 1 = 9$

$\hookrightarrow t(6,0) = (3 \cdot 6^2 + 4) \bmod 11 = 2$

## 9. Hashing - Wissen (6 Punkte)

9.1 Begründe kurz, warum beim Hashing mit offener Adressierung Schlüssel nicht einfach gelöscht werden dürfen.

Beim löschen beendet sich der Search-Vorgang früher als gewünscht

9.2 Beschreibe kurz, wie das Problem kurzfristig umgangen werden kann.

Die Plätze in der Hashtabelle werden als „schon besetzt", „noch nie besetzt", „wieder frei" markiert und die Suche wird nur beim „noch nie besetzt" abgebrochen

9.3 Warum ist die Lösung auf Dauer nicht geeignet?

Im Laufe der Zeit gibt es keine Plätze mit der Markierung „noch nie besetzt"

## 10. Kurzfragen (18 Punkte)

Kreuze an, welche Aussagen korrekt sind. Es gibt nur Punkte für vollständig korrekt angekreuzte Teilaufgaben. (Hinweis: In jeder Teilaufgabe ist immer mindestens eine Aussage korrekt.)

10.1 Welche Aussagen zu Fractional Knapsack sind korrekt?
- [ ] Die optimale Lösung hat immer einen größeren Wert als die Lösung derselben Instanz als ganzzahlige Variante (Maximum Knapsack). kann gleich sein
- [ ] Die Laufzeit des Algorithmus aus der Vorlesung hängt von der Größenschranke Z ab. nur $O(n \log n)$
- [x] Es existiert ein Greedy-Algorithmus, der das Problem optimal löst.

10.2 Das dynamische Programm für Subset Sum...
- [ ] ...besitzt polynomielle Laufzeit. $n2^n$
- [x] ...findet eine optimale Lösung.
- [x] ...kann verwendet werden, um Partition zu lösen.

10.3 Der Branch-and-Bound-Algorithmus für Maximum Knapsack...
- [ ] ...nutzt Fractional Knapsack als untere Schranke.
- [x] ...terminiert für bestimmte Instanzen bereits im Wurzelknoten. U=P
- [x] ...findet eine optimale Lösung.

10.4 Es gibt eine...
- [x] (1/2)-Approximation für Maximum Knapsack. Greedy 2 + Approximationsalgorithmus.
- [x] 1-Approximation für Fractional Knapsack (als Optimierungsproblem).
- [ ] (1/2)-Approximation für Minimum Vertex Cover.

2-Approximation

?? 10.5 Zur Bestimmung einer oberen Schranke eines Maximierungsproblems nutzen wir...
- [ ] ...einen Spezialfall.
- [x] ...eine Relaxierung.
- [ ] ...eine vollständige Enumeration.

đi qua hết tất cả trường hợp

10.6 Für jedes Problem der Klasse P...
- [x] ...existiert ein effizienter Algorithmus, der immer eine optimale Lösung zurück gibt.
- [ ] ...existiert eine Reduktion auf 3-SAT.
- [ ] ...können wir eine Lösung in polynomieller Zeit auf Zulässigkeit prüfen.

---

10.7 Nenne ein Problem aus der Klasse NP und begründe kurz, warum es in NP liegt.

Subset Sum: NP, da man mit polynomieller Zeit eine Lösung verifizieren kann, ob sie gültig ist, indem man den Wert aller Objekte der Lösungen aufsummiert und checkt ob die Summe dem erwünschten Wert gleicht

Z

O(n) Zeit

10.8 Wie zeigt man, dass ein Problem NP-vollständig ist?

man zeigt das Problem liegt sowohl in NP als auch in NP-schwer

NP: beweist alle Lösungen können in polynomieller Zeit verifiziert werden.

NP-hard: ⊕ Finden ein NP-hard Problem I'
⊕ Reduzieren aus I' auf I (das betrachtete Problem)
⊕ Beweis, dass I' genau dann eine Lösung hat, wenn I ein Lösung hat
⊕ Beweis die polynomieller Zeit der Transformation.