Technische Universität Braunschweig

Institute of Operating Systems and Computer Networks

Algorithmen und Datenstrukturen II
Große Übung #1

Arne Schmidt

19.04.2016

Organisatorisches

Homepage

Aktuelle Informationen, Hausaufgaben, Slides auf:
`https://www.ibr.cs.tu-bs.de/courses/ss17/aud2/`

Anmeldung

Gruppe	Termin	Raum	Tutor	Teilnehmer
1	Montag, 09:45 - 11:15 (14-tägig)	IZ 305	Mai Hellmann	TBA
2	Mittwoch, 08:00 - 09:30 (14-tägig)	IZ 358	Yannic Lieder	TBA
3	Mittwoch, 09:45 - 11:15 (14-tägig)	IZ 358	Moritz Pfister	TBA
4	Mittwoch, 13:15 - 14:45 (14-tägig)	IZ 358	Micha Horlboge	TBA
5	Freitag, 11:30 - 13:00 (14-tägig)	IZ 358	Dennis Luck	TBA

Anmeldung bis Freitag 23:59 Uhr.

Einteilung der Gruppen: Vorraussichtlich im Laufe des Samstag.

Semesterplan

Semesterplan AuD II SS17

Woche	Vorlesung (Do.)	Gr. Übung (Mi.)	Kl. Übung (Mo./Mi./Fr.)	HA Ausgabe (Mo.)	HA Abgabe (Mo. bis 13:15 Uhr)	HA Rückgabe (in kl UE)
14						
15	1					
16	2	1		HA0*, HA1		
17	3		1			HA0
18	4	2		HA2	HA1**	
19	5		2			HA1
20	6	3		HA3	HA2	
21	Christi Himmelfahrt		3			HA2
22	7	4		HA4	HA3	
23	Exkursionswoche					
24	8	5	4			HA3
25	9			HA5	HA4	
26	10	6	5			HA4
27	11				HA5	
28	12	7	6			HA5
*: Mündliche Aufgaben, Bearbeitung in ersten kleinen Übungen; keine Bewertung						
**: Da der 01.05.2017 ein Feiertag ist, findet die Abgabe des ersten Blattes am 02.05. statt.						

Hausaufgaben

- 6 Blätter
  - 1 unbewertet
  - 5 bewertet
- 45 Punkte pro Blatt, davon maximal 30 erreichbar
  - Es gibt: zwei Theorieaufgaben, eine Programmieraufgabe (Java)
  - Maximal 2 von 3 Aufgaben abgeben
- Studienleistung: mind. 75 Punkte
  - Studienleistung ist **keine** Vorraussetzung, um an der Prüfung teilzunehmen
- Zusammen arbeiten, **aber:** einzeln aufschreiben und abgeben
  - Nicht in Bleistift
  - Nicht in Rot

Klausur

Findet statt am 31.07.2017, zwischen 8:00 und 11:00 Uhr.

Nähere Informationen dazu erfolgen gegen Ende des Semesters über die Mailingliste (und über die Webseite).

Mailingliste

- Anmeldung über Homepage.
- Informationen zur Vorlesung/Übung/Klausur
- Bietet Möglichkeit Fragen zu stellen.
- Hiwis können über die Mailingliste erreicht werden.

Fragen!

Stellt Eure Fragen:

- über die Mailingliste
- in: Vorlesung, Gr. Übung, Kl. Übungen
- Euren Tutoren
- Sprechstunde von Arne Schmidt nach Vereinbarung
- Sprechstunde von Prof. Sándor Fekete (Mi. 13:15 - 14:00 Uhr)

Fragen?

Previously on AuD I

Problem

Allgemeine Fragestellung. Meistens formuliert mit:

Eingabe: Was ist gegeben?

Ausgabe: Was ist gesucht?

Lösung eines Problems:
Angabe eines Algorithmus.

Instanz

Eine Problemstellung mit konkreter Eingabe.

Lösung einer Instanz:
Angabe einer konkreten Ausgabe.

Beispiel - Klausurproblem

Gegeben:
- Maximale Zeit T,
- Zu erreichende Punkte P,
- Aufgaben $A_i = (t_i, p_i)$

Gesucht: Menge S von A_i's mit:

$$\sum_{i \in S} t_i \leq T$$

und

$$\sum_{i \in S} p_i \geq P$$

Lösung: ?? Das seht ihr in der Vorlesung!

Beispiel - Klausurinstanz

Gegeben:
- Maximale Zeit 120,
- zu erreichende Punkte 44,
- Aufgaben A_i:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
t_i	20	32	40	8	16	4	32	40	8	32	28	20	16	20	40	24
p_i	3	3	10	5	2	4	2	9	2	5	3	9	10	3	10	4

Gesucht: Menge S von A_i's mit:

$$\sum_{i \in S} t_i \leq 120$$

und

$$\sum_{i \in S} p_i \geq 44$$

Lösung: $S = \{6, 4, 13, 12, 9, 3, 16\}$ (siehe VL)

Laufzeiten / Komplexität

Meistens ist die absolute Laufzeit eines Algorithmus weniger interessant.
Wichtig ist die asympotische Laufzeit.
Beispielsweise kann man

$$25n^2 + 12n \log n - 8n$$

nach oben und unten abschätzen mit

$$17n^2 \leq 25n^2 + 12n \log n - 8n \leq 37n^2$$

Da Konstanten für asympotisches Wachstum uninteressant sind, sagt man "Die Funktion liegt in $\Theta(n^2)$":

$$25n^2 + 12n \log n - 8n \in \Theta(n^2)$$

Laufzeiten / Komplexität

Manchmal finden sich nur Abschätzungen nach oben bzw. nach unten. Dann ist zum Beispiel

$$25n^2 + 12n \log n - 8n \in O(n^2)$$

und

$$25n^2 + 12n \log n - 8n \in \Omega(n^2)$$

Diese beiden Notationen werden verwendet, um auszudrücken: Eine Funktion $f(n)$ braucht **maximal** $c \cdot g(n)$ lange (sprich $f(n) \in O(g(n))$). Eine Funktion $f(n)$ braucht **mindestens** $c \cdot g(n)$ lange (sprich $f(n) \in \Omega(g(n))$).

Sortieren

Aufgabe: Sortiere ein Array von n Zahlen in aufsteigender Reihenfolge.
Lösungen:

1. **Bubblesort**: Lasse Zahlen wie Blasen aufsteigen. Große Blasen verdrängen kleine Blasen.
2. **Quicksort**: Nimm ein Pivotelement p und teile die Zahlen in die Bereiche $< p$, $= p$ und $> p$. Wiederhole Prozedur auf erstem und drittem Bereich.
3. **Mergesort**: Teile das Array in der Mitte und sortiere die linke und rechte Hälfte mit Mergesort. Sind beide Hälften sortiert, kombiniere beide Hälften.

Sortieren - Laufzeit

Algorithmus	Best-Case	Average-Case	Worst-Case
Bubblesort	$O(n)$	$O(n^2)$	$O(n^2)$
Quicksort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Mergesort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Alle Algorithmen nutzen Vergleiche. Man weiß:

Theorem

Jeder vergleichsbasierte Sortieralgorithmus benötigt $\Omega(n \log n)$ Vergleiche.

Hausaufgaben
(Programmierteil)

```java
/**
 * Here are the actual algorithms!
 */
public class Knapsack1234567 {
        // Parses input. Do not edit.
        public static KnapsackInstance readInstance() {
                Scanner scanner = new Scanner(System.in);
                KnapsackInstance kp = new KnapsackInstance(scanner.nextInt());

                while (scanner.hasNextLong()) {
                        long capacity = scanner.nextLong();

                        if (scanner.hasNextLong()) {
                                long weight = scanner.nextLong();
                                long value = scanner.nextLong();
                                kp.append(weight, value);
                        } else {
                                kp.setCapacity(capacity);
                        }
                }

                kp.sort();
                return kp;
        }
```

"1234567" muss durch eure Matrikelnummer ersetzt werden!
(Auch der Dateiname muss dann angepasst werden.)

```java
public static KnapsackSolution bruteForce(KnapsackInstance kp) {
        return null; //TODO
}

public static KnapsackSolution fractional(KnapsackInstance kp) {
        return null; //TODO
}

public static KnapsackSolution greedy(KnapsackInstance kp) {
        return null; //TODO
}

public static KnapsackSolution dynamicProgramming(KnapsackInstance kp) {
        return null; //TODO
}

public static KnapsackSolution branchAndBound(KnapsackInstance kp) {
        return null; //TODO
}


public static KnapsackSolution greedyK(KnapsackInstance kp, int k) {
        return null; //TODO
}
```

Dies ist der Teil, der von euch über das Semester ausgefüllt werden soll.

```
        public static void main(String[] args) {
                KnapsackInstance kp = readInstance();
                System.out.println(kp);
//              System.out.println(fractional(kp));
//              System.out.println(bruteForce(kp));
//              System.out.println(branchAndBound(kp));
//              System.out.println(dynamicProgramming(kp));
//              System.out.println(greedyK(kp, 5));
        }
}
```

Um das Ergebnis einer Methode zu sehen, entfernt die Kommentierung der entsprechenden Zeile.
Das erste println gibt euch die Instanzdaten auf der Konsole aus.

Hinweis: Der Wert einer Lösung ist immer auf die nächste kleinere ganze Zahl gerundet!

Hausaufgaben

(Das Partitionsproblem)

Das Partitionsproblem - Partition

Das Problem wird noch in der 3. Vorlesung behandelt. Um die Hausaufgaben zu bearbeiten, seht ihr es jetzt schon:

Gegeben: n Objekte $1, \ldots, n$ mit jeweils Größe z_i.

Gesucht: Teilmenge $S \subseteq \{1, \ldots, n\}$ mit

$$\sum_{i \in S} z_i = \sum_{i \notin S} z_i$$

Anders ausgedrückt:
Teile n Zahlen in zwei Mengen, deren Gewicht gleichgroß ist.
Ein Beispiel:
7, 5, 9, 4, 6, 8, 3 lässt sich aufteilen in {7,6,8} und {5,9,4,3}
Leider klappt das Aufteilen nicht immer so einfach!

Fragen?