Übungen zur Vorlesung

Algorithmen und Datenstrukturen

Übungsblatt 6

Arbeitsgruppe Kryptographie und Komplexitätstheorie
Prof. Dr. Marc Fischlin
Dr. Christian Janson
Patrick Harasser
Felix Rohrbach

Sommersemester 2019
Veröffentlicht am: 24.05.2019
Abgabe am: 07.06.2019, 12:00 Uhr

P1 (Gruppendiskussion)

Nehmen Sie sich etwas Zeit, um die folgenden Fachbegriffe in einer Kleingruppe zu besprechen, sodass sie anschließend in der Lage sind, die Begriffe dem Rest der Übungsgruppe zu erklären:

(a) Dynamische Programmierung
(b) Greedy-Algorithmen

P2 (Dynamische Programmierung)

Wie Sie in der Vorlesung gelernt haben ist Dynamische Programmierung eine Methode um die Laufzeit mancher Algorithmen zu verbessern, indem Zwischenergebnisse abgespeichert werden (time-memory trade-off).

(a) Nennen Sie die zwei Bedingungen an Problemstellungen, damit die Anwendung von Dynamischer Prgrammierung vorteilhaft sein kann.
(b) Im Folgenden betrachten wir ein konkretes Szenario: Zwei Diebe haben Juwelen gestohlen und wollen ihre gemeinsame Beute nun fair aufteilen. Der Wert der n gestohlenen Juwelen sei gegeben als $w_1, \ldots, w_n$ mit $w_i \leq k, \forall i, k$ mit $w_i \in \mathbb{N}$ und alle Werte verschieden, wobei k eine obere Schranke des Wertes eines Objektes darstellt. Geben Sie einen Algorithmus (in $O(kn^2)$ im Worst-Case) unter Verwendung von Dynamischer Programmierung an, der entscheidet ob eine faire Aufteilung der Objekte möglich ist. Sie können dabei annehmen, dass kein Wert größer als die Hälfte der Summe aller Werte ist und dass mindestens ein Objekt existiert.

Hinweis: Zur Veranschaulichung des Problems finden Sie hier einen naiven Algorithmus (in $O(2^n)$ im Worst-Case), der jede mögliche Aufteilung der Beute prüft. Dabei beinhaltet das Array W die Werte der Objekte:

FairPartition(W)

```
1 : t = Σ_{i=0}^{n-1} W[i]
2 : if t mod 2 ≠ 0 then
3 :     return false
4 : return IsSubsetPossible(W, W.length, t/2)
```

IsSubsetPossible(W, n, s)

```
1 : if s = 0 then
2 :     return true
3 : if n = 0 then
4 :     return false
5 : return IsSubsetPossible(W, n − 1, s)
        ∨ IsSubsetPossible(W, n − 1, s − W[n − 1])
```

P3 (Dynamische Programmierung mit Memoisation)

In dieser Aufgabe betrachten wir Dynamische Programmierung mit Memoisation. Ähnlich zu dem Beispiel mit den Fibonacci-Zahlen aus der Vorlesung betrachten wir eine rekursive Folge über den natürlichen Zahlen mit:

$$\begin{cases} a_n = a_{\lfloor n/2 \rfloor} + a_{\lfloor n/3 \rfloor} & \text{für } n \geq 2 \\ a_0 = a_1 = 1 & \text{sonst.} \end{cases}$$

(a) Berechnen Sie die Werte a_n für $n = 2, \ldots, 6$.
(b) Entwerfen Sie einen Top-Down-Algorithmus (rekursiv, mit zusätzlicher Initialfunktion) mit Memoisation zur Bestimmung des n-ten Wertes dieser Folge und bewerten Sie, ob bei dieser Problemstellung ein Bottom-Up Algorithmus besser geeignet wäre.

P4 (Rucksack-Problem)

Ein Rucksack soll mit n Gegenständen gefüllt werden. Die Gegenstände haben positive Nutzwerte $w_1, \ldots, w_n > 0$, sowie positive Gewichte $g_1, \ldots, g_n > 0$. Im Rucksack kann ein Maximalgewicht $G_{\max}$ transportiert werden. Die Gegenstände können beliebig zerlegt und anteilig mitgenommen werden (z.B. ein halber Regenschirm). Die Gegenstände sollen so ausgewählt werden, dass der Nutzwert maximiert wird. Gesucht ist also ein Vektor $(\alpha_1, \ldots \alpha_n) \in [0, 1]^n$ mit der Eigenschaft, dass $\alpha_1 w_1 + \ldots + \alpha_n w_n$ maximal ist, jedoch $\alpha_1 g_1 + \ldots + \alpha_n g_n \leq G_{\max}$.

(a) Geben Sie einen greedy Algorithmus an, der dieses Problem effizient löst. Sie dürfen Hilfsfunktionen, wie z.B. eine Sortierfunktion, verwenden ohne diese zu beschreiben.
(b) Welche Laufzeit hat der Algorithmus?
(c) Angenommen, die Gegenstände dürfen nicht zerlegt werden, d.h. entwender man packt den ganzen Gegenstand in den Rucksack oder man lässt ihn draußen. Zeigen oder widerlegen Sie, dass ein gieriger Algorithmus wie in a) dieses Problem effizient lösen kann.

H1 (Huffman-Codes) (1+2+1+2+5+4 Punkte)

In dieser Übung sollen Sie lernen, einen Greedy-Algorithmus am Beispiel von Huffman-Codes zu implementieren. **Bitte lesen Sie sich zunächst die allgemeinen Hinweise für die praktischen Übungen auf Moodle durch!** Laden Sie sich anschließend das vorgegebene Framework herunter, um die Aufgabe implementieren zu können. Bitte entfernen Sie vor der Abgabe alle eigenen `System.out.println`s, damit Ihr Tutor die Ausgabe der Tests nicht suchen muss!

Huffman-Codes geben einen variabel-langen Bezeichner für Zeichen aus einem Alphabet an. Dabei richtet sich die Länge des Bezeichners nach der Frequenz, mit der das Zeichen vorkommt – um so öfter das Zeichen vorkommt, um so kleiner ist der Bezeichner. Die Bezeichner können dabei durch einen Baum dargestellt werden, dessen Blätter die Zeichen des Alphabets sind. Der Bezeichner ist jetzt definiert durch den Pfad von der Wurzel zum Blatt: Falls wir im Pfad als nächstes den linken Kindknoten auswählen, steht im Bezeichner an dieser Stelle eine 0, wenn wir den rechten Kindknoten auswählen, eine 1.

Der Huffman-Algorithmus zur Erstellung eines solchen Baums funktioniert wie folgt: Wir nehmen eine Priority-Queue (eine Queue, die intern die Elemente nach einem Schlüssel sortiert) und fügen jedes Zeichen des Alphabets als einen (unverbundenen) Baum-Knoten mit der Frequenz des Zeichens als Schlüssel in die Priority-Queue ein. Anschließend entfernen wir die zwei zu diesem Zeitpunkt kleinsten Knoten aus der Priority-Queue aus, erstellen einen neuen Knoten, der die beiden entfernten Knoten als Kinder hat, und fügen diesen neuen Knoten mit der Summe der Schlüssel der Kinder wieder in die Priority-Queue ein. Nach $n-1$ Wiederholungen ist nur noch ein Element in der Priority-Queue – dieses stellt dann die Wurzel unseres Baums da. Genauere Informationen zu Huffman-Bäumen finden Sie auch im Buch "Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms" ab Seite 428.

Sie sollen nun Huffman-Codes verwenden, um Daten zu komprimieren. In unserem Fall besteht damit das Alphabet aus allen möglichen Werten, die ein Byte annehmen kann (-128 bis 127). Implementieren Sie dazu im Framework die folgenden Punkte:

(a) Implementieren Sie die Methode `buildFrequencyTable` in der Klasse `HuffmanCodes`, die die relative Häufigkeit in den Daten für jedes Byte in der Liste `frequencyTable` speichert.

(b) Implementieren Sie die Methode `buildHuffmanTree` in der Klasse `HuffmanCodes`, die wie oben beschrieben aus der Häufigkeit der Bytes einem Huffman-Baum baut. Speichern Sie die Wurzel dieses Baums in `huffmanTreeRoot`. Für die Priority-Queue können Sie die Java-Klasse PriorityQueue verwenden.

(c) Implementieren Sie die Methode `buildHuffmanTable` in der Klasse `HuffmanCodes`, die ein Array mit den Bezeichnern für alle Bytes aufbaut (wir könnten auch jedes Mal den Baum nach dem Byte durchsuchen, um den Bezeichner zu erhalten, aber dies wäre deutlich langsamer). Das Ergebnis soll in `codeTable` gespeichert werden.

(d) Implementieren Sie die Methoden `compress` und `decompress` in der Klasse `HuffmanCodes`, die den gegebenen Eingabe-Stream komprimiert im Ausgabe-Stream speichert. Sie dürfen dazu `codeTable` und den Huffman-Baum verwenden.

(e) Implementieren Sie die Methoden `saveHuffmanTree` und `loadHuffmanTree` in der Klasse `HuffmanCodes`, die den Huffman-Baum im Ausgabe-Stream speichern bzw. aus dem Eingabe-Stream wieder auslesen. Sie sollen sich hierbei selbst eine effiziente Speicherform für den Baum überlegen. Sie sollte jedoch nicht mehr als 400 Bytes lang sein.

(f) Implementieren Sie die Methoden `compress` und `decompress` in der Klasse `DataCompressor`. Der komprimierte Stream sollte dabei das folgende Format haben: Erst kommen die drei Byte "AUD", um anzuzeigen, dass diese Daten von unserem Algorithmus komprimiert wurden. Als nächstes folgt die Repräsentation des Huffman-Baums. Anschließend wird die Anzahl der komprimierten Bytes als Integer gespeichert und zuletzt folgen die komprimierten Daten. Beim dekomprimieren der Daten soll zunächst auf die ersten drei Zeichen "AUD" geprüft werden und anschließend die Daten eingelesen werden. Wenn ein Fehler in den übergebenen Daten enthalten ist, soll eine `EncodingException` geworfen werden.