Datenstrukturen und Algorithmen
Übung 10, Frühling 2011

5. Mai 2011

Diese Übung muss zu Beginn der Übungsstunde bis spätestens um 16 Uhr 15 am 12. Mai abgegeben werden. Die Abgabe der DA Übungen erfolgt immer in schriftlicher Form auf Papier. Programme müssen zusammen mit der von ihnen erzeugten Ausgabe abgegeben werden. Drucken Sie wenn möglich platzsparend 2 Seiten auf eine A4-Seite aus. Falls Sie mehrere Blätter abgeben heften Sie diese bitte zusammen (Büroklammer, Bostitch, Mäppchen). Alle Teilnehmenden dieser Vorlesung müssen Ihre eigene Übung abgeben (einzeln oder in Zweiergruppen). Vergessen Sie nicht, Ihren Namen und Ihre Matrikelnummer auf Ihrer Abgabe zu vermerken.

Theoretische Aufgaben

1. Nicht jede Greedy-Methode für das Aktivitäten-Auswahl-Problem erzeugt eine maximale Menge paarweise zueinander kompatible Aktivitäten. Geben Sie Beispiele für die folgenden Strategien, die zeigen, dass die Strategien nicht zu optimalen Lösungen führen. Skizzieren Sie Ihre Beispiele grafisch.

   - Man wählt immer die Aktivität mit der geringsten Dauer, die zu den vorher ausgewählten Aktivitäten kompatibel ist.
   - Man wählt immer die kompatible Aktivität, die sich mit den wenigsten anderen noch verbliebenen Aktivitäten überlappt.
   - Man wählt immer die Aktivität mit der frühesten Startzeit, die zu den vorher ausgewählten Aktivitäten kompatibel ist.

   1 Punkt

2. Gegeben sei folgende Zeichenkette bestehend aus den Zeichen a, b, c, d, e, f, g:

   $$ffbgababababcddeabbf$$

   Konstruieren Sie einen Huffman-Code für diese Zeichenkette. Stellen Sie den binären Codierungsbaum dar, und geben Sie für jedes Zeichen den Binärcode. Sie müssen nicht den gesamten Code für die Zeichenkette aufschreiben. **1 Punkt**

3. Wir betrachten das Problem, Wechselgeld für n Rappen zusammenzustellen, indem wir so wenige Münzen wie möglich verwenden. Nehmen Sie an, dass der Nennwert jeder Münze eine ganze Zahl ist.

a) Geben Sie einen Greedy-Algorithmus an, der das Wechselgeld aus Münzen mit Nennwerten von 25, 10, 5, und 1 Rappen zusammenstellt.

Beweisen Sie, dass Ihr Algorithmus zu einer optimalen Lösung führt. Dazu müssen Sie zuerst zeigen, dass das Wechselgeld-Problem *optimale Teilstruktur* hat. Verwenden Sie ein Argument nach dem Muster von "Ausschneiden-und-Einfügen". Als zweites müssen Sie zeigen, dass Ihr Algorithmus die *Gierige-Auswahl-Eigenschaft* hat. Das heisst, dass es immer eine optimale Lösung gibt, welche die gierige Auswahl enthält. Zeigen Sie dies mit einer Fallunterscheidung für jeden Nennwert 25, 10, 5, und 1 Rappen. Tip: Argumentieren Sie mit einem Widerspruch, indem Sie Aussagen nach dem Muster "Falls es eine optimale Lösung ohne die gierige Auswahl geben würde, dann könnte man die Lösung durch die gierige Auswahl verbessern. Also muss jede optimale Lösung die gierige Auswahl enthalten" machen. **2 Punkte**

b) Geben Sie eine Menge von Münznennwerten an, für die ein Greedy-Algorithmus zu keiner optimalen Lösung führt. Ihre Menge sollte 1-Rappen Münzen enthalten, damit es für jeden Wert von n eine Lösung gibt. Geben Sie ein Beispiel, wo der Greedy-Algorithmus versagt.

Geben Sie einen Algorithmus an, der das Wechselgeld für eine beliebige Menge von k verschiedenen Münzwerten und einen beliebigen Betrag von n Rappen erstellt. Nehmen Sie an, dass 1-Rappen Münzen immer in der Menge von Münzwerten enthalten sind. Ihr Algorithmus sollte in Zeit $O(nk)$ ablaufen. Tip: Verwenden Sie dynamische Programmierung. Berechnen Sie eine Tabelle $c[j]$, welche für jeden Betrag j die kleinste Anzahl Münzen enthält. Starten Sie bei $j = 0$ und berechnen Sie $c[j]$ Schritt für Schritt bis $j = n$. Berechnen Sie auch eine Tabelle $denom[j]$, welche den Wert einer Münze angibt, die in der optimalen Lösung für j Rappen vorkommt. Geben Sie schliesslich einen Algorithmus, der mittels der Lösungstabelle $denom[j]$ eine optimale Lösung zurückgibt. **1 Punkt**

Praktische Aufgaben

Implementieren Sie eine Java Klasse *HuffmannCode*, welche folgende Funktionalität anbietet:

1. Eine Methode *void prefixCode (String s)*, welche für eine gegebene Zeichenkette den optimalen Präfix-Code generiert. Der Präfix-Code soll als Binärbaum dargestellt werden. Schreiben Sie dazu eine Klasse *Node* die einen Knoten im Baum darstellt. Verwenden Sie die Klasse *java.util.PriorityQueue* als Prioritätswarteschlange bei der Konstruktion des Codes. **2 Punkte**

2. Eine Methode *void printCode(String s)*, welche die codierte Darstellung einer Zeichenkette als Folge von Nullen und Einsen ausgibt. **2 Punkte**

3. Demonstrieren Sie Ihr Programm anhand der folgenden Eingaben und geben Sie die durchschnittliche Anzahl Bits pro Zeichen an:

   - *In computer science and information theory, Huffman coding is an entropy encoding algorithm used for lossless data compression.*

- *Huffman coding results in a prefix code that expresses the most common characters using shorter strings of bits than are used for less common source symbols.*

1 Punkt