

Datenstrukturen und Algorithmen

Cedric Aehi 17-103-235

Nicolas Müller 17-122-094

Datenstrukturen und Algorithmen

Übung 10, Frühling 2018

3. Mai 2018

Abgabe: Diese Übung muss zu Beginn der Vorlesungsstunde bis spätestens um 14 Uhr 15 am 9. 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). *Der gesamme Sourcecode muss außerdem elektronisch über Ilias abgegeben werden.*

Die Übung sollte vorzugsweise in Zweiergruppen bearbeitet werden, kann aber auch einzeln abgegeben werden. Vergessen Sie nicht, Ihren Namen und Ihre Matrikelnummer auf Ihrer Abgabe zu vermerken. Jede Übungsserie gibt 10 Punkte. Im Durchschnitt müssen Sie 7 von 10 Punkten erreichen, um die Testatbedingungen zu erfüllen.

Theoretische Aufgaben

1. Nicht jede Greedy-Methode für das Aktivitäten-Auswahl-Problem erzeugt eine maximale Menge paarweise zueinander kompatibler 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, d, o, p, q, r :

$opabqprqpdbqpdopq$

Konstruieren Sie einen Huffman-Code für diese Zeichenkette. Stellen Sie den binären Codierungsbaum dar und geben Sie für jedes Zeichen den Binärkode an. Wie lang ist die Huffman-codierte Zeichenkette? Sie müssen nicht den gesamten Code für die Zeichenkette aufschreiben.

1 Punkt

3. In dieser Aufgabe betrachten wir das Problem, Wechselgeld für n Rappen zusammenzustellen, indem wir so wenige Münzen wie möglich verwenden.

- a) Entwerfen Sie einen Greedy-Algorithmus, der das Wechselgeld aus Münzen mit Nennwerten aus der Menge $N = \{25, 10, 5, 1\}$ Rappen zusammenstellt und beweisen Sie, dass Ihr Algorithmus zu einer optimalen Lösung führt. Der Beweis soll die folgenden Schritte beinhalten:
 - Zeigen Sie, dass das Wechselgeld-Problem eine *optimale Teilstruktur* hat.
 - Zeigen Sie, dass Ihr Algorithmus die *Gierige-Auswahl-Eigenschaft* hat. Das bedeutet, dass es immer eine optimale Lösung gibt die durch Ihre *gierige* Auswahlvorschrift gefunden werden kann (die lokale optimale Wahl führt zu einer globalen optimalen Lösung).

1 Punkt

- b) Geben Sie eine Menge N von Münzenwerten an, für die Ihr Greedy-Algorithmus nicht zur optimalen Lösung führt. Ihre Menge sollte 1-Rappen Münzen enthalten, damit es für jeden Wert von n eine Lösung gibt. Nehmen Sie an, dass der Nennwert jeder Münze eine ganze Zahl ist. Geben Sie ein Beispiel an, wo der Greedy-Algorithmus versagt. **1 Punkt**
- c) Geben Sie einen auf dynamischer Programmierung basierenden Algorithmus an, der das Wechselgeld für eine beliebige Menge N von k verschiedenen Münzwerten und einen beliebigen Betrag von n Rappen erstellt, wobei in N immer 1-Rappen Münzen enthalten sind. Ihr Algorithmus sollte in Zeit $O(nk)$ ablaufen. *Tipp: Bauen sie eine Tabelle $c[j]$ auf, welche für jeden Betrag j die optimale Anzahl Münzen enthält. Bauen Sie parallel dazu eine Tabelle $denom[j]$ auf, welche den Wert einer beliebigen Münze angibt, die in der optimalen Lösung für j Rappen vorkommt. Im letzten Schritt soll der Algorithmus mittels der Lösungstabelle $denom[j]$ eine optimale Lösung konstruieren.* **1 Punkt**

Praktische Aufgaben

Implementieren Sie eine Java Klasse *HuffmanCode*, welche folgende Funktionalität bietet:

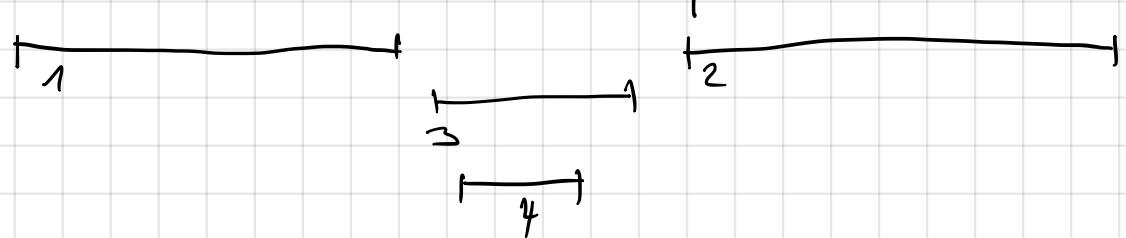
1. Eine Methode `void prefixCode(String s)`, welche für eine gegebene Zeichenkette einen optimalen Präfix-Code generiert. Der Präfix-Code soll als Binärbaum repräsentiert werden. Schreiben Sie dazu eine Klasse `Node` die einen Knoten im Baum darstellt. Verwenden Sie die Klasse `java.util.PriorityQueue` als Prioritätswarteschlange für die 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:
 - *Jobs launched into a sermon about how the Macintosh and its software would be so easy to use that there would be no manuals.*

- *An academic career in which a person is forced to produce scientific writings in great amounts creates a danger of intellectual superficiality, Einstein said.*

1 Punkt

Vergessen Sie nicht Ihren Sourcecode innerhalb der Deadline über die Ilias Aufgabenseite einzureichen.

1) ;)

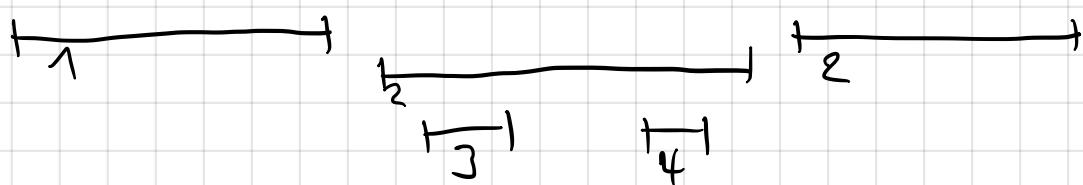


Nun wird neben den bereits gewählten Aktivitäten

1 & 2 die 4. anstatt 3. ausgewählt (da kürzer).

Diese ist weniger gut als die 3.

∴)



Hier werden 3 und 4 ausgewählt, obwohl 2 besser wäre. Zuerst wird 3 oder 4 gewählt, da diese (im Gegensatz zu 2) sich nur mit einer überlappen. Dann fällt 2 weg.

∴)

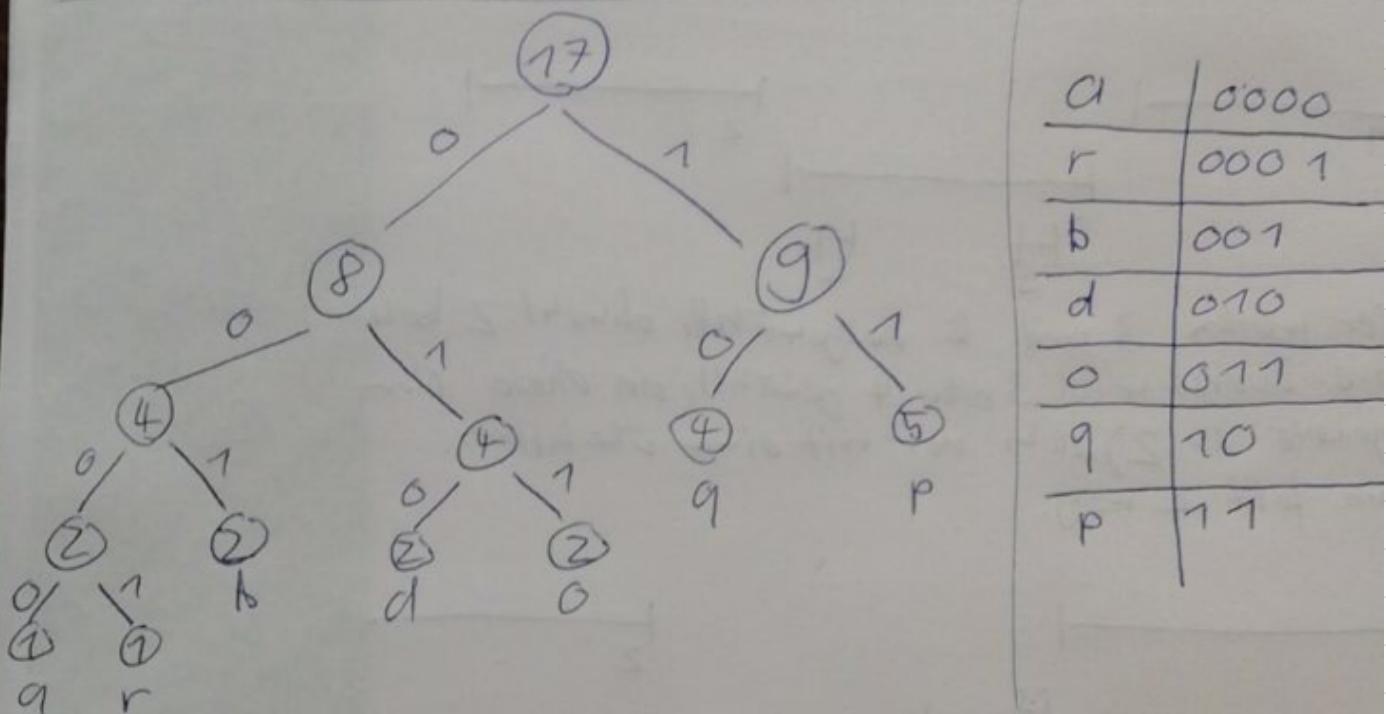
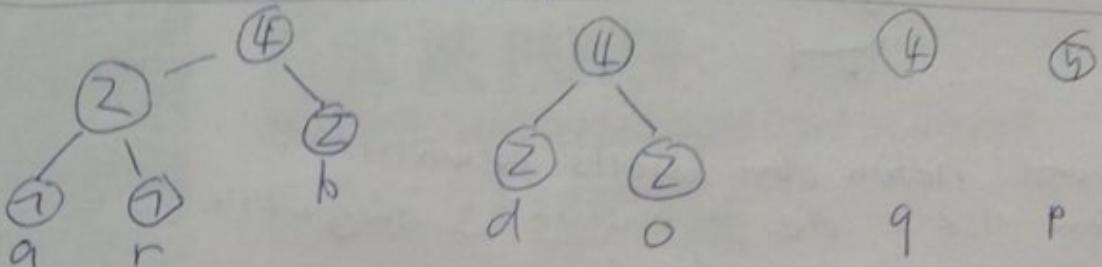


Hier wird 3 anstelle von 4 gewählt, da diese früher beginnt. 4 wäre aber besser.

2.

opabqprqrpdqpdopq

a	b	d	o	p	q	r
1	2	2	3	5	4	1
✓	✓	✓	✓	✓	✓	✓



a	b	d	o	p	q	r
---	---	---	---	---	---	---

Länge: 1 · 4 2 · 3 2 · 3 2 · 3 5 · 2 4 · 2 1 · 4
 4 + 6 + 6 + 6 + 10 + 8 + 4

= 44 bits lang.

3) a) "Pseudopseudo"-Code:

coinproblem (int n)

Array coins with {25, 10, 5, 1}

moneyLeft = n;

while moneyLeft > 0

for i=0 to coins.size

if (coins[i] ≤ moneyLeft)

choose coin at coins[i];

moneyLeft -= coins[i];

break;

return coins which were chosen

Optimale Teilstruktur:

Annahme dass die optimale Lösung für den Betrag

n enthalte k Münzen, also

$$z(n)=k$$

Zudem gilt eine Münze mit Geldwert c ist

in dieser Lösung enthalten, also

$$k-1 \text{ Münzen mit } z(n-c) = k-1$$

Falls es eine bessere Lösung für $z(n-c)$ geben würde, dann würde es auch eine bessere Lösung für $z(n)$ geben. Widerspruch \rightarrow

Gierige Auswahl:

$1 \leq n < 5$	$c=1$ enthalten in optimaler Lösung
$5 \leq n < 10$	$c=5$ enthalten (ganz sicher)
$10 \leq n < 25$	$c=10$ enthalten
$25 \leq n$	$c=25$ enthalten

$\underbrace{\{1, \dots, 1\}}_{25\text{mal}} \underbrace{\{5, \dots, 5\}}_{5\text{mal}} \underbrace{\{10, 10, 5\}}_{3\text{mal}} \underbrace{\{25\}}_{1\text{mal}} + \{\text{Rest}\}$
 $\rightarrow 25\text{er Münze ist optimal}$

b) $N = \{25, 10, 1\}$ $n = 30$

der Algorithmus wählt hier $\{25, 1, 1, 1, 1, 1\}$, statt der optimalen $\{10, 10, 10\}$

c) Denonfunction (`int n, int[] array`)

```
int[] denom = new Array();
c[] = new Table();
for i = 1 to n
    for j = 1 to i
        for k = 0 to array.size
            if c[j] + array[k] == i
                denom.add(array[j]);
                break;
return denom;
```

coin Problem (int n)

moneyLeft = n ;

while (n > moneyLeft)

add (denom [n]) to the chosen coins ;

moneyLeft - denom [n] ;

return the chosen coins ;

```
1 public class Node implements Comparable<Node> {
2
3     public char name;
4     public int frequency;
5     public Node left;
6     public Node right;
7     public String code;
8
9     public Node(char i, int m){
10         this.name = i;
11         this.frequency = m;
12     }
13     @Override
14     public int compareTo(Node n){
15         return this.frequency - n.frequency;
16     }
17 }
18
```

```

1 import java.util.ArrayList;
2 import java.util.PriorityQueue;
3
4
5 public class HuffmanCode {
6
7     public Node root;
8     private static int numberofCaracters = 0;
9     static ArrayList<String> array = new ArrayList();
10
11    void prefixCode (String s) {
12
13        PriorityQueue<Node> Queue = new PriorityQueue<Node>();
14
15        char c;
16        int distinct = 0;
17
18        while (s.length()>0) {
19            c = s.charAt(0);
20            String tempS = "" + c;
21            int replace = s.length() - s.replace(tempS, "").length();
22            Node nodi = new Node(c, replace);
23            Queue.add(nodi);
24
25            distinct++;
26
27            s = s.replace(tempS, "");
28        }
29        buildHuffmanTree(Queue,distinct);
30
31        root = Queue.poll();
32    }
33
34    private void buildHuffmanTree(PriorityQueue<Node> Queue, int distinct){
35        for (int i=1; i<distinct; i++) {
36            Node x = Queue.poll();
37            Node y = Queue.poll();
38            Node nodi = new Node('$', x.frequency+y.frequency);
39            nodi.left = x;
40            nodi.right = y;
41            Queue.add(nodi);
42        }
43    }
44
45    void printCode (String s) {
46
47        numberofCaracters = 0;
48
49        System.out.println("Input-String: \n" + s);
50
51        prefixCode(s);
52
53        setHuffmanCodes(root, "");
54
55        String c = "";
56        for (int i=0; i<s.length(); i++) {
57            c += traverseTree(root, s.charAt(i));
58        }
59        System.out.println("Huffman coded: \n" + c + "\n");
60    }
61
62    private String setHuffmanCodes(Node node, String code) {
63        if(node.left!=null){
64            setHuffmanCodes(node.left, code + "0");
65        }
66        if(node.right!=null){
67            setHuffmanCodes(node.right, code + "1");
68        }
69        if(node.left==null && node.right==null) {
70            node.code=code;
71        }
72        if(node.name != '$') {
73            array.add(node.code);
74            numberofCaracters++;
75        }
76        return null;

```

```
77     }
78
79     private String traverseTree(Node n, char c) {
80         if (n.name==c) {
81             return n.code;
82         }
83         if (n.left==null && n.right==null)
84             return null;
85
86         String left = traverseTree(n.left, c);
87         String right = traverseTree(n.right, c);
88
89         if (left!=null)
90             return left;
91         if (right!=null)
92             return right;
93         return null;
94     }
95
96     public static void main(String[] args) {
97         HuffmanCode huffman = new HuffmanCode();
98         String s1 = "Jobs launched into a sermon about how the Macintosh and its
software would be so easy to use that there would be no manuals." + "\n";
99         String s2 = "An academic career in which a person is forced to produce
scientific writings in great amounts creates a danger of intellectual superficiality
, Einstein said." + "\n";
100        huffman.printCode(s1);
101        System.out.println("Average of bits per caracter: \n" + calculateAverage() +
"\n\n");
102        array.clear();
103        huffman.printCode(s2);
104        System.out.println("Average of bits per caracter: \n" + calculateAverage() +
"\n\n");
105    }
106
107    public static int calculateAverage(){
108        Integer sum = 0;
109        if(!array.isEmpty()) {
110            for (String a : array) {
111                sum += a.length();
112            }
113            return sum / array.size();
114        }
115        return sum;
116    }
117
118 }
```

Output of program:

Input-String:

Jobs launched into a sermon about how the Macintosh and its software would be so easy to use that there would be no manuals.

Huffman coded:

```
1000010010110101000100101110111101111000101110110110011000110001111000100  
0111000101011011011110000001001110011101011001011111100001111001001101001100  
11110110100101110011101000100110001111000101010111100011100111100110001100110  
0101000101001011100011011110101111101000110101011111001010011001011011011  
0100101001000110111101010100011000100011111010110100110011110111011101100001  
1001111011011111010001101011111001010011001011011010001110100011101000100000111  
001111111111010010101000111100011
```

Average of bits per caracter:

5

Input-String:

An academic career in which a person is forced to produce scientific writings in great amounts creates a danger of intellectual superficiality, Einstein said.

Huffman coded:

```
00110001001110111010111010110000001110010101011010111010000000001000110010  
1001110001101001001010100010011101110101111000100001101111010011100100110  
11011111011110100010100001011011001111111011010111110001111010110111111010000  
1100110100100001001011101011111001010101100011011000010011101010011011100110  
110010100111010111010000001110011111011100011101111011111100101110110110101  
0000001110011100001101101110101101110001000110111101111101100101  
0010111000010100101000101001111111111000101110011011111101111000100011110  
010101001011100010101001110011001000011000100010101001011001110000101001110  
011011100101011000111100011111
```

Average of bits per caracter:

5