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.*

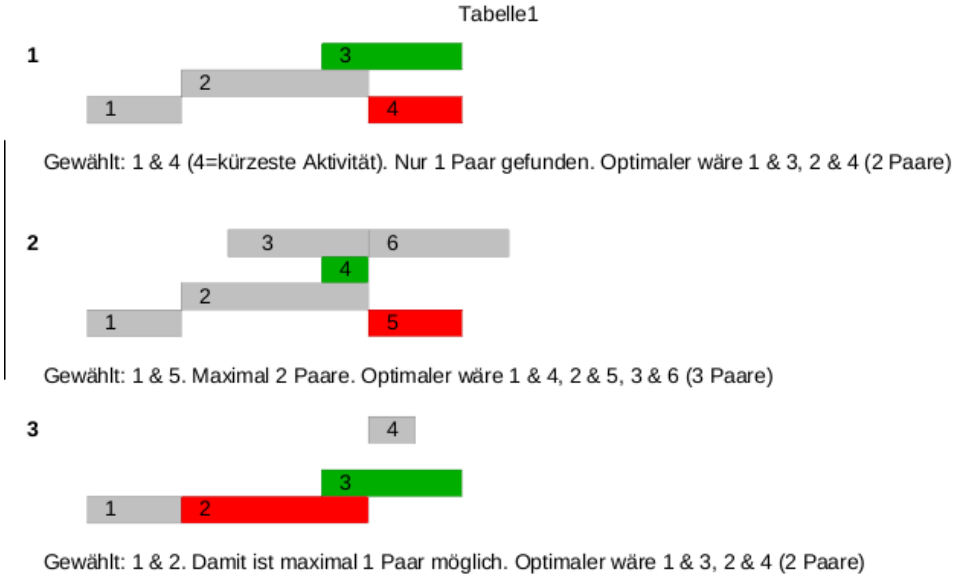
- *Huffman coding results in a prefix code that expresses the most common characters using shorter strings of bits than are used for less common source symbols.*

1 Punkt

Theoretische Aufgaben

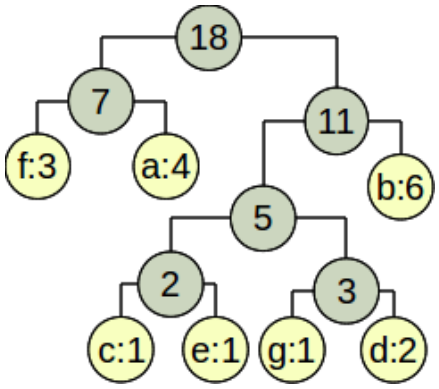
Aufgabe 1

Gesucht: *Maximale Menge paarweise zueinander kompatible Aktivitäten.*
Unterste Zeile: Ganze Auswahl. Rot: (nicht optimale) Auswahl der nächsten Aktivität. Grün: optimale Auswahl.



Aufgabe 2

Häufigkeit: `a:4, b:6, c:1, d:2, e:1, f:3, g:1`.
Sortiert: `c:1, e:1, g:1, d:2, f:3, a:4, b:6`.



Aufgabe 3

- a) Greedy-Algorithmus: Nimm die grösstmögliche Münze, die kleiner oder gleich gross ist wie der Gesamtbetrag.

Listing 1: Aufgabe 3

```
int[] chooseCoins(int n)
int[] coins = {25, 10, 5, 1}
int[] coinsChosen = {}
int moneyLeft = n

while moneyLeft > 0
   for i = 0 to coins.size
      if (coins[i] >= moneyLeft)
         coinsChosen.add(coins[i])
         moneyLeft -= coins[i]
         break

return coinsChosen
```

Optimale Teilstruktur
Das Wechselgeld-Problem hat in dieser Form eine optimale Teilstruktur. Wird von einem anfänglichen Betrag n der bestmögliche Betrag abgezogen, dann bleibt ein optimaler Teilbetrag, von dem wieder ein bester Betrag abgezogen werden kann. Die Teilbeträge sind alle für sich optimal, da mit dem Greedy-Algorithmus der beste Betrag gewählt wird.

Gierige-Auswahl-Eigenschaft
Für jeden Betrag gibt es eine optimale lösung, welche die gierige Auswahl enthält:
z.B. für Beträge ab 25 Rappen muss immer 25 Rappen enthalten sein. Wenn z.B. 40 Rappen in {10, 10, 10, 10} aufgeteilt wird, kann man die Beträge ersetzen mit {25, 10, 5}, um eine verbesserte Lösung zu enthalten.
Beispiel für Nennwert 10: 12 Rappen kann in {5, 5, 1, 1} unterteilt werden. {5, 5} kann jedoch durch {10} ersetzt werden (die optimale Lösung).
Beispiel für Nennwert 5: 8 Rappen kann man in {1, 1, 1, 1, 1, 1, 1, 1} aufteilen, aber au mit der gierigen Auswahl in {5, 1, 1, 1}.
Beispiel für Nennwer 1: 2 Rappen können nur in {1, 1} aufgeteilt werden, welches bereits die optimale Lösung ist und auch die gierige Auswahl enthält.

- b) Eine Menge mit {25, 10, 1} Rappen kann mit dem Greedy-Algorithmus zu einer suboptimalen Lösung führen. z.B. 30 Rappen wird in {25, 1, 1, 1, 1, 1} aufgeteilt, statt in optimal 3 Münzstücke {10, 10, 10}.

Listing 2: Aufgabe 3

```
int[] chooseCoinsDP(int n, int[] k)
int[] denom

c[0] = 0

//loop to n (possible money values)
for j = 1 to n

   //loop through DP-table
   for i = 0 to j

      //loop through coins available
      for i2 = 0 to k.size
        if c[i]+k[i2] == j
```

```
28                denom.add(k[i])
29                break
30
31    return denom
```

Aufruf, um Münzwert zu finden:

Listing 3: Aufgabe 3

```
32  //call, denom was calculated before
33  int[] getCoins(int n)
34  int[] coinsChosen
35  moneyLeft = n
36
37  while moneyLeft > 0
38      coinsChosen.add(denom[n])
39      moneyLeft - denom[n]
40
41  return coinsChosen
```

Praktische Aufgaben

Aufgabe 1, 2 3

Siehe Anhang `output.txt`, `HuffmannCode.java` und `Node.java`.

```java
import java.util.PriorityQueue;


public class HuffmannCode {

	public Node root;

	public static void main(String[] args) {
		HuffmannCode encoder = new HuffmannCode();
		String s1 = "In computer science and information theory, Huffmann coding is an entropy encoding algorithm used for lossless data compression.";
		String s2 = "Huffmann 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.";
		encoder.printCode(s1);
		encoder.printCode(s2);
	}

	void prefix (String s) {
		//build priority queue
		PriorityQueue<Node> Q = new PriorityQueue<Node>();

		char a = '0';
		int dist = 0;
		//count freq for each sign
		while (s.length()>0) {
			a = s.charAt(0);
			Node z = new Node(a, countOccurences(s, a));
			//System.out.printf("Neuer Knoten: "+a+"\n");
			Q.add(z);

			//add to distinct set
			dist ++;

			//remove char from String
			s = removeChar(s, a);
		}

		//build tree
		for (int i=1; i<dist; i++) {
			Node x = Q.poll();
			Node y = Q.poll();
			Node z = new Node('0', x.freq+y.freq);
			z.left = x;
			z.right = y;
			Q.add(z);
		}
		//last element is the root
		root = Q.poll();
	}

	void printCode (String s) {
		//Eingabestring
		System.out.println("\nEingabestring: "+s);

		//construct
		prefix(s);

		//set codes
		setCodes(root, "");

		//traverse tree
		String code = "";
		for (int i=0; i<s.length(); i++) {
			code += traverse(root, s.charAt(i));
		}

		System.out.println(code);
	}

	private String setCodes(Node n, String code) {
		if (n.left!=null) setCodes(n.left, code+"0");
		if (n.right!=null) setCodes(n.right, code+"1");
```

```
                if (n.left==null && n.right==null) {
                        n.code=code;
                        System.out.println("Keine Kinder vorhanden. Setze code: "+code+" für "+n.name);
                }

                return null;
        }

        private String traverse(Node n, char c) {
                //if found, return
                if (n.name==c) {
                        //System.out.println(c+" gefunden!");
                        return n.code;
                }

                if (n.left==null && n.right==null) return null;

                String l=traverse(n.left, c), r=traverse(n.right, c);
                if (l!=null)
                        return l;
                if (r!=null)
                        return r;

                return null;
        }

        public static int countOccurences(String s, char b) {
           int counter=0;
           for (int i=0;i<s.length();i++) {
                if (s.charAt(i) == b) counter++;
           }
           return counter;
        }

        public static String removeChar(String s, char c) {
                    String r = "";

                    for (int i=0; i<s.length(); i++) {
                       if (s.charAt(i) != c) r += s.charAt(i);
                    }

                    return r;
                }

}
```

```java
public class Node implements Comparable<Node> {
	public Node left;
	public Node right;
	public int freq;
	public char name;
	public String code;

	@Override
	public int compareTo(Node n) {
		return this.freq - n.freq;
	}

	public Node(char n, int f) {
		this.name = n;
		this.freq = f;
	}
}
```

```
Eingabestring: In computer science and information theory, Huffmann coding is an entropy encoding
algorithm used for lossless data compression.
Keine Kinder vorhanden. Setze code: 00000 für g
Keine Kinder vorhanden. Setze code: 00001 für p
Keine Kinder vorhanden. Setze code: 00010 für l
Keine Kinder vorhanden. Setze code: 00011 für u
Keine Kinder vorhanden. Setze code: 001 für n
Keine Kinder vorhanden. Setze code: 0100 für t
Keine Kinder vorhanden. Setze code: 0101 für r
Keine Kinder vorhanden. Setze code: 0110 für a
Keine Kinder vorhanden. Setze code: 01110 für f
Keine Kinder vorhanden. Setze code: 0111100 für I
Keine Kinder vorhanden. Setze code: 0111101 für ,
Keine Kinder vorhanden. Setze code: 0111110 für H
Keine Kinder vorhanden. Setze code: 0111111 für .
Keine Kinder vorhanden. Setze code: 1000 für i
Keine Kinder vorhanden. Setze code: 1001 für s
Keine Kinder vorhanden. Setze code: 101 für
Keine Kinder vorhanden. Setze code: 110000 für h
Keine Kinder vorhanden. Setze code: 110001 für y
Keine Kinder vorhanden. Setze code: 11001 für m
Keine Kinder vorhanden. Setze code: 1101 für e
Keine Kinder vorhanden. Setze code: 11100 für d
Keine Kinder vorhanden. Setze code: 11101 für c
Keine Kinder vorhanden. Setze code: 1111 für o
01111000011011110111111100100001000110100110101011011001111011000110100111101110110101100011111001
01100000101110111101011100101100100100011110011010100110000110111110101110001011110110101111110000
11011100111011001011000100110111101111111100100000100000101100010011010110001101110100101000010111
11000011100011011101001111011111111001000000100000101011000010000000111101011000010011000011001101 0
00111001110111100101011101111010110100010111110011001000101101100110011011110001100100011010111110
1111111001000010101110110011001100011110010111111

Eingabestring: Huffmann 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.
Keine Kinder vorhanden. Setze code: 0000 für c
Keine Kinder vorhanden. Setze code: 0001 für m
Keine Kinder vorhanden. Setze code: 0010 für a
Keine Kinder vorhanden. Setze code: 00110 für g
Keine Kinder vorhanden. Setze code: 001110 für x
Keine Kinder vorhanden. Setze code: 001111 für b
Keine Kinder vorhanden. Setze code: 010 für s
Keine Kinder vorhanden. Setze code: 0110 für n
Keine Kinder vorhanden. Setze code: 0111 für t
Keine Kinder vorhanden. Setze code: 10000 für u
Keine Kinder vorhanden. Setze code: 100010 für p
Keine Kinder vorhanden. Setze code: 100011 für d
Keine Kinder vorhanden. Setze code: 10010 für h
Keine Kinder vorhanden. Setze code: 10011 für f
Keine Kinder vorhanden. Setze code: 1010 für r
Keine Kinder vorhanden. Setze code: 101100 für l
Keine Kinder vorhanden. Setze code: 1011010 für .
Keine Kinder vorhanden. Setze code: 10110110 für H
Keine Kinder vorhanden. Setze code: 10110111 für y
Keine Kinder vorhanden. Setze code: 10111 für i
Keine Kinder vorhanden. Setze code: 110 für
Keine Kinder vorhanden. Setze code: 1110 für o
Keine Kinder vorhanden. Setze code: 1111 für e
1011011010000100111001100010010011001101100000111010001110111011000110110101011110101000010110001 1
10101101011101101100010110100010101011111001110111001110110000011101000111111100111100100010011111
1011110011101000101010111101001011110101100111100101111110000111100100111110000011100001000111100 1
1011000000100100010101000100000011111111010010110100000101011101100011011001010010111010100111111 1
01011001001111010101110110001100101101110100111100011111011101110101100111100100010011011000101010
1111110100000101111100011110100111101010110101100111101001011000001110000100011110011011001011101
000010100000111111001010110111000100111111101011000101011010
```