**Technische Universität Berlin** **Computer Vision & Remote Sensing**

# Übungsblatt 2 (Block 1)

Prof. Dr. Olaf Hellwich und Mitarbeiter

**KV-Diagramme und Rekursion**

| | |
|---|---|
| **Verfügbar ab:** | **30.04.18** |
| **Abgabe bis:** | **07.-11.05.18** |

**Hinweis**: Aufgrund des JUnit-Tests für die Hausaufgabenabgabe haben alle Studenten bis zum 09.05.18 um 18 Uhr Zeit für die Abgabe der Hausaufgaben. Nähere Einzelheiten zur alternativen Abgabe befinden sich auf ISIS.

## Aufgabe 1: KV-Diagramme(1) — 2 Punkte

1. Gegeben sind die folgenden Karnaugh-Veitch (KV)-Diagramme. Geben Sie unter Benutzung dieser jeweils einen booleschen Ausdruck mit minimaler Anzahl der Terme an[1]. Für $y_0$ soll diese in konjunktiver Normalform (KNF) und für $y_1$ in disjunktiver Normalform (DNF) angegeben werden.

$y_0$ :

| $x_3$ \ $x_1$ \ $x_2$ $x_0$ | | | | |
|---|---|---|---|---|
| | 1 | 1 | 1 | 0 |
| | 1 | 0 | 0 | 0 |
| | 1 | 1 | 0 | 0 |
| | 1 | 1 | 0 | 1 |

$y_1$ :

| $x_3$ \ $x_1$ \ $x_4$ $x_2$ $x_0$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

[1] Teilweise kann es nützlich sein, die KV-Tafel zusammenzuklappen und somit mit einem Term die größtmögliche Zahl an Kästchen abzudecken.

**Musterlösung:**

$y_0 = (\overline{x}_1 + \overline{x}_2)(\overline{x}_0 + \overline{x}_1 + x_3)(\overline{x}_0 + \overline{x}_2 + \overline{x}_3)(x_0 + \overline{x}_2 + x_3)$

$y_1 = \overline{x}_0\overline{x}_2\overline{x}_4 + \overline{x}_2\overline{x}_3\overline{x}_4$ (oder $\overline{x}_1\overline{x}_2\overline{x}_4$) $+ \overline{x}_0\overline{x}_1\overline{x}_4 + x_0x_1\overline{x}_3 + \overline{x}_1\overline{x}_2x_3$

### Aufgabe 2: KV-Diagramme(2) — 2 Punkte

Erstellen Sie für die folgenden Booleschen Ausdrücke jeweils ein KV-Diagramm:

$y_2 : x_0x_2 + \overline{x}_0x_1x_2 + \overline{x}_0x_1\overline{x}_3 + \overline{x}_1\overline{x}_2$

$y_3 : (\overline{x}_0 + \overline{x}_1 + \overline{x}_3 + \overline{x}_4)(\overline{x}_0 + \overline{x}_1 + x_3 + x_4)(x_1 + \overline{x}_2 + \overline{x}_3 + x_4)(x_0 + x_1 + x_2 + \overline{x}_3)(\overline{x}_2 + \overline{x}_4)$

*Hinweis:* Das Erstellen einer Wahrheitstafel ist nicht nötig, das KV-Diagramm kann direkt aus den gegebenen Termen abgeleitet werden.

**Musterlösung:**

$y_2$ :

| $x_3$ \ $x_1$ \ $x_0$ $x_2$ | | | | |
|---|---|---|---|---|
| | 1 | 1 | 1 | 0 |
| | 1 | 0 | 1 | 1 |
| | 0 | 0 | 1 | 1 |
| | 1 | 1 | 1 | 0 |

$y_3$ :

| $x_3$ \ $x_1$ \ $x_4$ $x_2$ $x_0$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

### Aufgabe 3: Rekursion — 6 Punkte

Aus der Mathematik ist Ihnen die Potenzfunktion bekannt. In dieser Aufgabe sollen Sie in der Klasse `Potenz` zur Potenz-Berechnung eine iterative sowie eine rekursive Methode schreiben. Beide Methoden werden in einer weiteren Klasse namens `TestPotenz` in der `main`-Methode getestet.

Gehen Sie dabei wie folgt vor:

1. Erstellen Sie die Klasse `Potenz`.
   a) Erstellen Sie die Methode `potenzIterativ`. Diese soll

- öffentlich sichtbar sein,
- eine statische Methode sein,
- zwei Zahlen übergeben bekommen: $a$ (die Basis als Fließkommazahl `double`) und $n$ (den Exponenten als Ganzzahl `int`),
- prüfen, ob Basis oder Exponent negativ sind und in diesem Fall -1 zurückgeben. Ansonsten wird unter Verwendung wiederholter Multiplikation die Potenz berechnet und zurückgegeben.

b) Erstellen Sie die Methode `potenzRekursiv`. Diese soll

- öffentlich sichtbar sein,
- eine Objektmethode (also nicht statisch) sein,
- zwei Zahlen übergeben bekommen: $a$ (die Basis als Fließkommazahl `double`) und $n$ (den Exponenten als Ganzzahl `int`),
- prüfen, ob Basis oder Exponent negativ sind und in diesem Fall -1 zurückgeben. Ansonsten wird unter Verwendung wiederholter Multiplikation die Potenz berechnet und zurückgegeben.
- Hinweis: Überlegen Sie sich, welche Abbruchbedingung für die Rekursion am besten geeignet ist.
- Abgaben, die mehr als zwei Methodenparameter nutzen, führen zu **Punktabzug**.
- Zusatzaufgabe: Verwenden Sie für die bisherige Methode `potenzRekursiv` eine baumartige Rekursion und nennen Sie die zusätzliche Methode `potenzRekursivBaum`. Veranschaulichen lässt sich dies an folgendem Beispiel: $5^7 = 5^3 \cdot 5^3 \cdot 5$. Überlegen Sie sich selbst, wie sich hier die Methode verändern müsste.

2. Erstellen Sie die Klasse `TestPotenz`

a) Erstellen Sie die `main`-Methode. Diese soll

- die Methode `Math.pow`[2] mit den Parametern $a = 2.2$, $n = 10$ sowie $a = 3$, $n = 0$ aufrufen und die Berechnungsergebnisse in Form eines Satzes auf der Konsole ausgeben.[3]
- die Methode `potenzIterativ` mit den Parametern $a = 2.2$, $n = 10$ sowie $a = 3$, $n = 0$ in geeigneter Weise aufrufen und die Berechnungsergebnisse als Satz auf der Konsole ausgeben.
- die Methode `potenzRekursiv` mit den Parametern $a = 2.2$, $n = 10$ sowie $a = 3$, $n = 0$ in geeigneter Weise aufrufen und die Berechnungsergebnisse als Satz auf der Konsole ausgeben.

Anmerkungen:

1. Überlegen Sie sich einen sinnvollen Rückgabetyp für jede Methode.
2. Die $n$-te Potenz der Zahl $a$ lässt sich auf die Multiplikation zurückführen: $a^n = \underbrace{a \cdot a \cdot \ldots \cdot a}_{n \text{ mal}}$.
3. Das neutrale Element der Multiplikation ist 1 und für alle ganzen Zahlen $x$ gilt: $x^0 = 1$.
4. Geben Sie bitte, wie immer, nur jeweils eine Implementierung pro Methode ab.

**Musterlösung:**

```
1  public class Potenz {
2     /**
3       * Berechnet die Potenz iterativ.
4       * Statisch, Aufruf von aussen erfolgt i.d.R. mit
5       * vorangestelltem Klassenbezeichner "Potenz"
```

[2] Diese Methode wird bereits von Java zur Verfügung gestellt und muss nicht implementiert werden. Die Ergebnisse aus den anderen Methoden (`potenzIterativ` und `potenzRekursiv`) können Sie mit den Ergebnissen dieser Methode vergleichen.

[3] Beispielaufruf: `double potenz1 = Math.pow(2.2, 3);`

```
6       * mit Punktoperator.
7       */
8     public static double potenzIterativ(double a, int n) {
9        if ((a < 0) || (n < 0)) { // pruefe Parameter
10          System.out.println("Fehler: a oder n negativ");
11          return -1;
12       }
13       double potenz = 1; // neutrales Element; fuer n=0 muss 1
             herauskommen
14       for (int i=0; i<n; i++) { // fuer n > 0 wird die
             Schleifenbedingung n-mal erfuellt
15          potenz = potenz * a;
16       }
17       return potenz;
18    }
19
20    /**
21      * Berechnet die Potenz rekursiv.
22      * Aufruf erfolgt auf einem Objekt der Potenz-Klasse.
23      */
24    public double potenzRekursiv(double a, int n) {
25       if ((a < 0) || (n < 0)) { // pruefe Parameter
26          System.out.println("Fehler: a oder n negativ");
27          return -1;
28       } else if (n == 0) { // Abbruchbedingung der Rekursion
29          return 1;
30       } else { // zerlege in a * Restproblem
31          return a * potenzRekursiv(a, n-1);
32       }
33    }
34
35    /**
36      * Berechnet die Potenz rekursiv und mit baumartiger Rekursion.
37      * (Alternative zur Methode berechneRekursiv.)
38      * Aufruf erfolgt auf einem Objekt der Potenz-Klasse.
39      */
40    public static double potenzRekursivBaum(double a, int n) {
41       if ((a < 0) || (n < 0)) { // pruefe Parameter
42          System.out.println("Fehler: a oder n negativ");
43          return -1;
44       } else if (n == 0) { // Abbruchbedingung der Rekursion
45          return 1;
46       } else {
47          // rekursiver Funktionsaufruf durch baumartige Rekursion
48          if (n % 2 == 0) { // gerader Exponent, Halbierung in der Mitte
49             return potenzRekursivBaum(a, n/2) * potenzRekursivBaum(a, n
                   /2);
50             // Alternativ: Die baumartige Rekursion wird vereinfacht zu
                   einer linearen Rekursion, um die Laufzeit zu optimieren
51             // float res = potenzRekursivBaum(a, n/2);
52             // return res * res;
53          } else { // ungerader Exponent, Halbierung in der Mitte, a
                bleibt 1x seperat
54             return a * potenzRekursivBaum(a, n/2) * potenzRekursivBaum(a
                   , n/2);
55             // Alternativ: Die baumartige Rekursion wird vereinfacht zu
                   einer linearen Rekursion, um die Laufzeit zu optimieren
```

```
56              // float res = potenzRekursivBaum(a, n/2);
57              // return a * res * res;
58          }
59      }
60  }
61  }
```

```
1   public class TestPotenz {
2       /**
3        * Hauptprogramm.
4        * Testet die Java-Implementierung sowie eigene Implementierungen
           der Potenzfunktion.
5        */
6       public static void main(String[] args) {
7           // Testwerte, a=Basis, e=Exponent
8           double a1 = 2.2;
9           int e1 = 10;
10          double a2 = 3;
11          int e2 = 0;
12
13          // Ergebnisvariablen
14          double p1;
15          double p2;
16
17          // teste Java-Methode
18          p1 = Math.pow(a1, e1);
19          p2 = Math.pow(a2, e2);
20          System.out.println("Math.pow: Das Ergebnis von " + a1 + " hoch " +
                e1 + " ist " + p1 + ".");
21          System.out.println("Math.pow: Das Ergebnis von " + a2 + " hoch " +
                e2 + " ist " + p2 + ".");
22          System.out.println("
                ------------------------------------------------------");
23
24          // teste iterative Implementierung
25          Potenz potenzklassenobjekt = new Potenz();
26          p1 = Potenz.potenzIterativ(a1, e1);
27          p2 = Potenz.potenzIterativ(a2, e2);
28          System.out.println("potenzIterativ: Das Ergebnis von " + a1 + "
                hoch " + e1 + " ist " + p1 + ".");
29          System.out.println("potenzIterativ: Das Ergebnis von " + a2 + "
                hoch " + e2 + " ist " + p2 + ".");
30          System.out.println("
                ------------------------------------------------------");
31
32          // teste rekursive (sowie statische) Implementierung
33          p1 = potenzklassenobjekt.potenzRekursiv(a1, e1);
34          p2 = potenzklassenobjekt.potenzRekursiv(a2, e2);
35          System.out.println("potenzRekursiv: Das Ergebnis von " + a1 + "
                hoch " + e1 + " ist " + p1 + ".");
36          System.out.println("potenzRekursiv: Das Ergebnis von " + a2 + "
                hoch " + e2 + " ist " + p2 + ".");
37          System.out.println("
                ------------------------------------------------------");
38
39          // teste rekursive (sowie statische) baumartige Implementierung
40          p1 = potenzklassenobjekt.potenzRekursivBaum(a1, e1);
```

```
41          p2 = potenzklassenobjekt.potenzRekursivBaum(a2, e2);
42          System.out.println("potenzRekursivBaum: Das Ergebnis von " + a1 +
                " hoch " + e1 + " ist " + p1 + ".");
43          System.out.println("potenzRekursivBaum: Das Ergebnis von " + a2 +
                " hoch " + e2 + " ist " + p2 + ".");
44      }
45  }
```