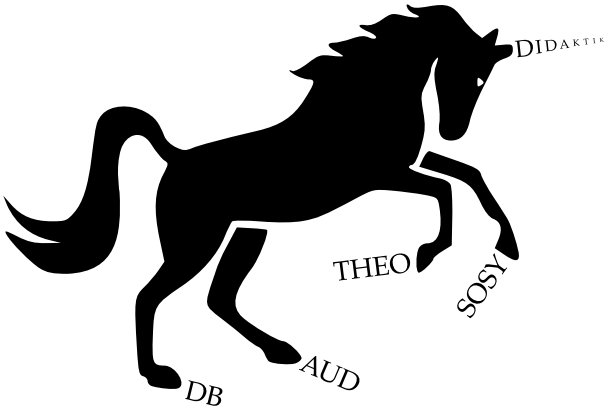
66116
Herbst
2012

Datenbanksysteme / Softwaretechnologie (vertieft)

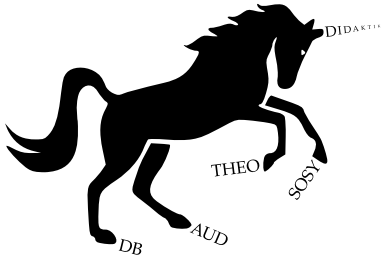
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Thema Nr. 2

Teilaufgabe Nr. 2

1 Objektorientierung [Getränkelieferservice]

Ein Getränkelieferservice verwaltet die Bestellungen verschiedener Kunden. Die folgenden Teilaufgaben sind in einer objektorientierten Programmiersprache zu lösen (die verwendete Sprache ist vorab anzugeben.).

(a) Implementieren Sie eine Klasse `Kasten` zur Beschreibung eines Getränkekastens mit den folgenden Eigenschaften. Entscheiden Sie dabei jeweils ob eine Realisierung als Objekt- oder Klassenfeld sinnvoll ist.

- Es existiert ein einheitliches Kastenpfand in Höhe von 1,50 Euro.
- Für alle Flaschen in einem Kasten gelte ein einheitliches Flaschenpfand, das jedoch von Kasten zu Kasten verschieden sein kann.
- Während das Flaschenpfand für alle Flaschen eines Kastens gleich ist, sind die Einzelpreise der Flaschen je nach Inhalt unterschiedlich. Die Einzelpreise (ohne Flaschenpfand) der im Kasten enthaltenen Flaschen sollen in einem 2-dimensionalen Array abgelegt werden.

Geben Sie für die Klasse `Kasten` einen geeigneten Konstruktor an. Ergänzen Sie in der Klasse `Kasten` eine Objektmethode zur Berechnung des Gesamtpreises des Getränkekastens inklusive Kasten- und Flaschenpfand.

(b) Schreiben Sie eine Klasse `Bestellung`. Jeder Bestellung soll eine eindeutige Bestellnummer zugeordnet werden, die über den Konstruktoraufruf erstellt wird. Außerdem soll zu jeder Bestellung der Name des Kunden gespeichert werden, sowie eine einfach verkettete Liste der bestellten Getränkekästen. Die Klasse Bestellung soll weiterhin eine Methode beinhalten, die den Gesamtpreis der Bestellung ermittelt.

(c) Schreiben Sie ein kleines Testprogramm, das eine Bestellung erstellt, die zwei Getränkekästen umfasst. Der erste Kasten soll ein 1 x 1 Getränkekasten mit einer Flasche zu 0,75 Euro sein, der zweite Kasten soll - wie in Abbildung 1 dargestellt - ein 3 x 3 Getränkekasten mit 3 Flaschen zu 0,7 Euro auf der Diagonalen und 3 weiteren Flaschen zu je 1 Euro sein. Das Flaschenpfand beider Kästen beträgt 0,15 Euro pro Flasche, das Kastenpfand 1,50 Euro. Anschließend soll der Preis der Bestellung berechnet und auf der Standardausgabe ausgegeben werden.

1,0	1,0	0,7
1,0	0,7	0
0,7	0	0

```java
/**
 * „Implementieren Sie eine Klasse Kasten zur Beschreibung eines Getränkekastens
 * mit den folgenden Eigenschaften. Entscheiden Sie dabei jeweils ob eine
 * Realisierung als Objekt- oder Klassenfeld sinnvoll ist."
 */
public class Kasten {
  /**
   * Wir verwenden static, also ein sogenanntes Klassenfeld, da das Kastenpfand
   * für alle Kästen gleich ist: „Es existiert ein einheitliches Kastenpfand in
   * Höhe von 1,50 Euro."
   */
  static double kastenPfad = 1.5;

  /**
   * Wir verwenden ein Objektfeld, d.h. ein nicht statisches Feld: „Für alle
   * Flaschen in einem Kasten gelte ein einheitliches Flaschenpfand, das jedoch
   * von Kasten zu Kasten verschieden sein kann."
   */
  double flaschenPfad;

  /**
   * „Während das Flaschenpfand für alle Flaschen eines Kastens gleich ist, sind
   * die Einzelpreise der Flaschen je nach Inhalt unterschiedlich. Die
   * Einzelpreise (ohne Flaschenpfand) der im Kasten enthaltenen Flaschen sollen
   * in einem 2-dimensionalen Array abgelegt werden."
   */
  double[][] flaschen;

  /**
   * „sowie eine einfach verkettete Liste der bestellten Getränkekästen. "
   */
  Kasten nächsterKasten = null;

  /**
   * „Geben Sie für die Klasse Kasten einen geeigneten Konstruktor an."
   *
   * @param flaschen     Die Belegung des Kasten mit Flaschen als
   *                     zweidimensionales Feld der Flaschenpreise ohne
   *                     Flaschenpfand.
   * @param flaschenPfad Die Höhe des Flaschenpfads, dass für alle Flaschen in
   *                     diesem Kasten gleich ist.
   */
  public Kasten(double[][] flaschen, double flaschenPfad) {
    this.flaschen = flaschen;
    this.flaschenPfad = flaschenPfad;
  }

  /**
   * „Ergänzen Sie in der Klasse Kasten eine Objektmethode zur Berechnung des
   * Gesamtpreises des Getränkekastens inklusive Kasten- und Flaschenpfand."
   *
   * @return Der Gesamtpreis des Getränkekastens inklusive Kasten- und
   *         Flaschenpfand.
   */
```

```
  double berechneGesamtPreis() {
    double gesamtPreis = kastenPfad;
    for (int i = 0; i < flaschen.length; i++) {
      double[] reihe = flaschen[i];
      for (int j = 0; j < reihe.length; j++) {
        double flaschenPreis = flaschen[i][j];
        // Nur im Kasten vorhandene Flaschen kosten auch Flaschenpfand.
        if (flaschenPreis > 0)
          gesamtPreis += flaschenPfad + flaschen[i][j];
      }
    }
    return gesamtPreis;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2012/herbst/getraenke/Kasten.java`

```
import java.text.DecimalFormat;

/**
 * „Schreiben Sie eine Klasse Bestellung"
 */
public class Bestellung {
  /**
   * „Jeder Bestellung soll eine eindeutige Bestellnummer zugeordnet werden, die
   * über den Konstruktoraufruf erstellt wird."
   */
  int bestellNummer;

  /**
   * „Außerdem soll zu jeder Bestellung der Name des Kunden gespeichert werden."
   */
  String kundenName;

  /**
   * „sowie eine einfach verkettete Liste der bestellten Getränkekästen. "
   */
  Kasten kästen = null;

  /**
   * „Jeder Bestellung soll eine eindeutige Bestellnummer zugeordnet werden, die
   * über den Konstruktoraufruf erstellt wird. Außerdem soll zu jeder Bestellung
   * der Name des Kunden gespeichert werden."
   *
   * @param bestellNummer Die Nummer der Getränkebestellung.
   * @param kundenName    Der Name des/der KundenIn.
   */
  public Bestellung(int bestellNummer, String kundenName) {
    this.bestellNummer = bestellNummer;
    this.kundenName = kundenName;
  }

  /**
   * „Die Klasse Bestellung soll weiterhin eine Methode beinhalten, die den
```

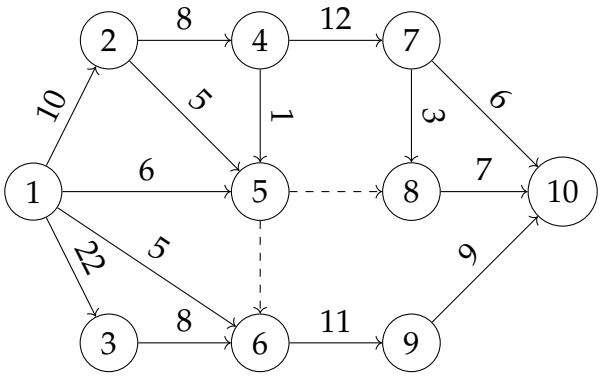
```java
 * Gesamtpreis der Bestellung ermittelt."
 *
 * @return Der Gesamtpreis der Getränkebestellung.
 */
double berechneGesamtPreis() {
  double gesamtPreis = 0;
  Kasten kasten = kästen;
  while (kasten != null) {
    gesamtPreis += kasten.berechneGesamtPreis();
    kasten = kasten.nächsterKasten;
  }
  return gesamtPreis;
}

/**
 * Nicht verlangt. Könnte auch in die Test-Methode geschrieben werden.
 *
 * @param flaschen     Die Belegung des Kasten mit Flaschen als
 *                     zweidimensionales Feld der Flaschenpreise ohne
 *                     Flaschenpfand.
 * @param flaschenPfad Die Höhe des Flaschenpfads, dass für alle Flaschen in
 *                     diesem Kasten gleich ist.
 */
void bestelleKasten(double[][] flaschen, double flaschenPfad) {
  Kasten bestellterKasten = new Kasten(flaschen, flaschenPfad);
  if (kästen == null) {
    kästen = bestellterKasten;
    return;
  }
  Kasten kasten = kästen;

  Kasten letzterKasten = null;
  while (kasten != null) {
    letzterKasten = kasten;
    kasten = kasten.nächsterKasten;
  }
  letzterKasten.nächsterKasten = bestellterKasten;
}

/**
 * Kleines Schmankerl. Nicht verlangt. Damit wir nicht 9.899999999999999 als
 * Aufgabe bekommen.
 *
 * @param preis Ein Preis als Gleitkommazahl.
 *
 * @return Der Preis als Text mit zwei Stellen nach dem Komma.
 */
static String runde(double preis) {
  DecimalFormat df = new DecimalFormat("#.##");
  df.setMinimumFractionDigits(2);
  return df.format(preis);
}

/**
```

```java
    * Die main-Methode soll hier als Testmethode verwendet werden:
    *
    * „Schreiben Sie ein kleines Testprogramm, das eine Bestellung erstellt, die
    * zwei Getränkekästen umfasst. Der erste Kasten soll ein 1 x 1 Getränkekasten
    * mit einer Flasche zu 0,75 Euro sein, der zweite Kasten soll - wie in
    * Abbildung 1 dargestellt - ein 3 x 3 Getränkekasten mit 3 Flaschen zu 0,7 Euro
    * auf der Diagonalen und 3 weiteren Flaschen zu je 1 Euro sein. Das
    * Flaschenpfand beider Kästen beträgt 0,15 Euro pro Flasche, das Kastenpfand
    * 1,50 Euro. Anschließend soll der Preis der Bestellung berechnet und auf der
    * Standardausgabe ausgegeben werden."
    *
    * @param args Kommandozeilenargumente, die uns nicht zu interessieren brauchen.
    */
   public static void main(String[] args) {
     Bestellung bestellung = new Bestellung(1, "Hermine Bschlangaul");

     // Müsste eigentlich nicht mehr gesetzt werden, da wir es schon in der
     // Klassendefinition gesetzt haben.
     Kasten.kastenPfad = 1.50;

     bestellung.bestelleKasten(new double[][] { { 0.75 } }, 0.15);
     bestellung.bestelleKasten(new double[][] { { 1.0, 1.0, 0.7 }, { 1.0, 0.7, 0 },
     ↪  { 0.7, 0, 0 } }, 0.15);

     // Oder kürzer
     // bestellung.bestelleKasten(new double[][] { { 1, 1, .7 }, { 1, .7 }, { .7 }
     ↪  }, .15);

     // Gegenrechnung:
     // 1 x 0.75 = 0.75
     // 3 x 1.00 = 3.00
     // 3 x 0.70 = 2.10
     // 7 x 0.15 = 1.05 (Flaschenpfad)
     // 3 x 1.50 = 3.00 (Kastenpfand)
     // ----
     // 9.90
     System.out.println("Der Gesamtpreis der Getränkebestellung beträgt: " +
     ↪  runde(bestellung.berechneGesamtPreis()) + " €");
   }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2012/herbst/getraenke/Bestellung.java

Aufgabe 2 [Gantt und CPM]

Die unten stehende Abbildung stellt ein CPM-Netzwerk dar. Die Ereignisse sind fortlaufend nummeriert (Nummer im Inneren der Kreise) und tragen keine Namen. Gestrichelte Linien stellen Pseudo-Aktivitäten mit einer Dauer von 0 dar.



(a) Berechnen Sie die früheste Zeit für jedes Ereignis, wobei angenommen wird, dass das Projekt zum Zeitpunkt 0 startet!

Lösungsvorschlag

— Wir führen eine Vorwärtsterminierung durch und addieren die Dauern. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Maximum aus. **Erläuterungen:** i: Ereignis i; FZ_i: Frühester Zeitpunkt, zu dem Ereignis i eintreten kann.

i	Nebenrechnung	FZ_i
1		0
2		10
3		22
4		18
5	$\max(15_2, 6_1, 19_4)$	19
6	$\max(5_1, 30_6, 19_5)$	30
7		30
8	$\max(33_7, 19_5)$	33
9		41
10	$\max(36_7, 40_8, 50_9)$	50

(b) Setzen Sie anschließend beim letzten Ereignis die späteste Zeit gleich der frühesten Zeit und berechnen Sie die spätesten Zeiten!

Lösungsvorschlag

— Wir führen eine Rückwärtsterminierung durch und subtrahieren die Dauern vom letzten Ereignis aus. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Minimum aus. **Erläuterungen:** i: Ereignis i; SZ_i: Spätester Zeitpunkt, zu dem Ereignis i eintreten kann.

i	Nebenrechnung	SZ_i
10	siehe FZ_{10}	50
9		41
8		43
7	$\min(44_{10}, 40_8)$	40
6		30
5	$\min(30_6, 43_8)$	30
4	$\min(29_5, 28_7)$	28
3		22
2	$\min(20_4, 25_5)$	20
1	$\min(10_2, 24_5, 0_3, 25_6)$	0

(c) Berechnen Sie nun für jedes Ereignis die Pufferzeiten!

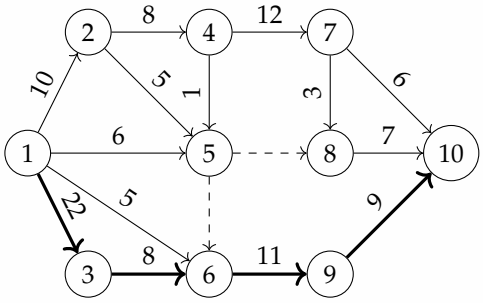
Lösungsvorschlag

i	1	2	3	4	5	6	7	8	9	10
FZ_i	0	10	22	18	19	30	30	33	41	50
SZ_i	0	20	22	28	30	30	40	43	41	50
GP	0	10	0	10	11	0	10	10	0	0

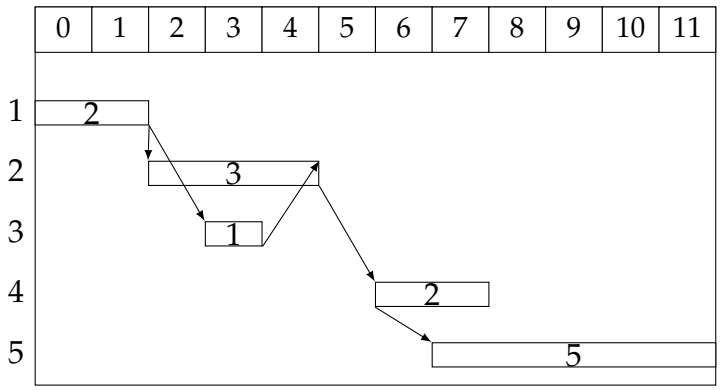
(d) Bestimmen Sie den kritischen Pfad!

Lösungsvorschlag

$1 \to 3 \to 6 \to 9 \to 10$



(e) Konvertieren Sie das Gantt-Diagramm aus Abbildung 3 in ein CPM-Netzwerk!



Lösungsvorschlag