Repetitorium der Informatik

Prüfungsaufgaben und Lösungen 2001 - 2008

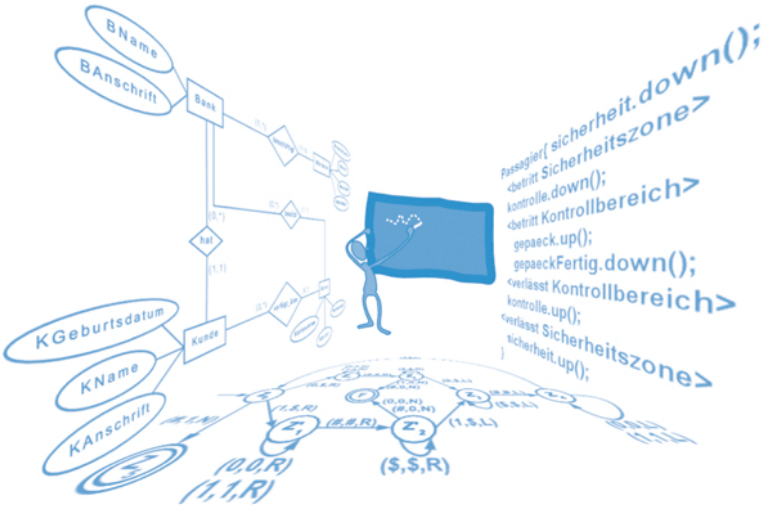
Bearbeitet von
Ulrich Kiesmüller, Sandra Leibinger

1. Auflage 2009. Taschenbuch. 495 S. Paperback
ISBN 978 3 486 58905 4
Format (B x L): 17 x 24 cm
Gewicht: 9320 g

Ulrich Kiesmüller | Sandra Leibinger

Repetitorium der Informatik

Prüfungsaufgaben und Lösungen 2001–2008



Oldenbourg

Repetitorium der Informatik

Prüfungsaufgaben und Lösungen 2001–2008

von
Ulrich Kiesmüller und
Sandra Leibinger

Oldenbourg Verlag München

Sandra Leibinger, geboren 1984 in Duisburg-Rheinhausen. Im September 2003 begann sie das Studium für das Lehramt an Gymnasien in Mathematik und Informatik an der TU Dresden. Im September 2006 wechselte sie an die Friedrich-Alexander-Universität Erlangen-Nürnberg, wo sie ihr Studium dann im Frühjahr 2008 mit dem ersten Staatsexamen abschloss. Ab dem Wintersemester 2006 war sie als studentische Hilfskraft in der Didaktik der Informatik eingesetzt, wo sie insbesondere mit der Betreuung und Staatsexamensvorbereitung der Teilnehmenden am Projekt FLIEG (Flexible Lehrerweiterbildung in Informatik als Erweiterungsfach für Gymnasien) betraut war. Im Schuljahr 2008/2009 hat sie ihr Referendariat am Adam-Kraft-Gymnasium, Schwabach begonnen

Ulrich Kiesmüller, geboren 1964 in München. Von September 1983 bis 1989 studierte er das Lehramt an Gymnasien in Mathematik und Physik an der Julius-Maximilians-Universität Würzburg. Im Herbst 1989 schloss er sein Studium mit dem ersten Staatsexamen ab. Nach seinem Referendariat von Februar 1990 bis 1992 absolvierte er dann das zweite Staatsexamen. Nach absolvieren eines Erweiterungsstudiums für das Fach Informatik an der Friedrich-Alexander-Universität Erlangen-Nürnberg legte er im Herbst 2003 das Staatsexamen für Informatik ab. Seit dem ist er als wissenschaftlicher Mitarbeiter in der Didaktik der Informatik bei Prof. Dr. Torsten Brinda an der Friedrich-Alexander-Universität Erlangen-Nürnberg tätig. Dort zählt neben einem Promotionsvorhaben zu seinen Aufgaben insbesondere die Betreuung und Staatsexamensvorbereitung der Teilnehmenden am Projekt FLIEG (Flexible Lehrerweiterbildung in Informatik als Erweiterungsfach für Gymnasien).

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0
oldenbourg.de

Lektorat: Dr. Margit Roth
Herstellung: Anna Grosser
Coverentwurf: Kochan & Partner, München
Gedruckt auf säure- und chlorfreiem Papier
Druck: Grafik + Druck, München
Bindung: Thomas Buchbinderei GmbH, Augsburg

ISBN 978-3-486-58905-4

Vorwort

Mit der Einführung des achtstufigen Gymnasiums in Bayern im Jahr 2004 und dem damit verbundenen neuen Lehrplan wurde die Informatik als Pflichtfach ab bereits der 6. Jahrgangsstufe eingeführt. Um die möglichst baldige Versorgung der Schulen mit genügend Informatiklehrkräften zu erreichen, wurde von Prof. Dr. Peter Hubwieser (TU München) und Prof. Dr. Torsten Brinda (Friedrich-Alexander-Universität Erlangen-Nürnberg) das Projekt FLIEG (**F**lexible **L**ehrerweiterbildung für **I**nformatik als **E**rweiterungsfach an **G**ymnasien) ins Leben gerufen. Hierbei werden Gymnasiallehrer, die bereits in zwei Fächern das Staatsexamen absolviert haben, parallel zum laufenden Unterrichtsbetrieb innerhalb von 2,5 bis 4 Jahren durch betreutes Tele-Learning auf das Staatsexamen in Informatik als Erweiterungsfach an Gymnasien vorbereitet. Die ersten Absolventen traten im Herbst 2008 zur Examensprüfung an.
Die Autoren dieses Buches waren an der Friedrich-Alexander-Universität Erlangen-Nürnberg in der Didaktik der Informatik damit betraut, die am FLIEG-Projekt Teilnehmenden zu betreuen und auf das Staatsexamen vorzubereiten. Hierbei wurden die in diesem Buch zusammengestellten Lösungen zu den Staatsexamensaufgaben der Jahrgänge 2001 bis 2008 entwickelt. Wir danken den Teilnehmenden sehr für die konstruktive Mitarbeit bei der Erstellung der Lösungen und die kritische Kontrolle sowie die eingebrachten Verbesserungsvorschläge.

Die Einteilung der Staatsexamensaufgaben in die Themenbereiche „Theoretische Informatik“, „Algorithmen und Datenstrukturen“, „Objektorientierte Modellierung“, „Datenbanken“ und „Betriebssysteme“ spiegeln sich in der Kapiteleinteilung dieses Buches wieder. Innerhalb dieser Kapitel werden die Aufgaben (Angabentexte sind grau unterlegt) jeweils chronologisch aufgeführt. Am Seitenrand sind den Stoffumfang der Aufgabe charakterisierende Stichworte angegeben. Erscheint ein bestimmter Aufgabentyp zum ersten Mal, so werden zusätzlich zur Lösung ausführliche theoretische Grundlagen vermittelt. Später auftretende ähnliche Aufgabenlösungen werden dann knapper gehalten. An manchen Stellen werden Hinweise auf hilfreiche Literatur (✎ bzw. ✐) bzw. Webseiten (☞ bzw. ☜) gegeben. Die Programme sind in den jeweils in der Aufgabenstellung vorgegebenen Programmiersprachen verfasst. Ist dort nichts näheres spezifiziert, so werden für objektorientierte Probleme Java, für funktionale Programme Haskell und für imperative Programme historisch bedingt Pascal eingesetzt. Durch Kommentierung des Quellcodes werden die Strukturen und Denkweisen der Programme näher erläutert.

Der Wortlaut der Aufgabenstellungen wurde unverändert übernommen, lediglich hinsichtlich des Layouts wurden kleinere Änderungen vorgenommen. Die Angabentexte sind auch zu finden unter:

http://ddi.informatik.uni-erlangen.de/Service/st_ex/index.xml ☜

(Teil-)Aufgaben, die den Themenbereichen „Rechnernetze“ und „Rechnerarchitektur“ entstammen, wurden in diesem Buch nicht behandelt, da diese Themenbereiche nach der aktuellen Prüfungsordnung nicht mehr zum Stoffumfang der schriftlichen Prüfungen gehören.

Insbesondere bei Modellierungsaufgaben (Entity-Relation-Ship Modelle, objektorientierte Klassendiagramme), aber auch in anderen Bereichen wie die Implementierung von Algorithmen oder die Ableitung von Worten bei gegebenen Grammatiken und endlichen Automaten, existieren verschiedene Lösungsmöglichkeiten für einige der gestellten Aufgaben; die dargestellten Wege sind als Vorschläge und nicht als einzig gültige Musterlösung zu verstehen. In einigen Fällen bieten wir mehrere Alternativlösungen an.

Seitenzahlen, die im Stichwortverzeichnis durch Fettdruck hervorgehoben sind, verweisen auf besonders ausführliche Lösungen mit Hintergrundinformationen zum jeweiligen Thema.

Dieses *Repetitorium der Informatik* soll dazu dienen, den grundlegenden Lehrstoff der oben genannten Teilgebiete der Informatik an Hand der thematisch und chronologisch geordneten Lösungen der bayerischen Staatsexamensprüfungsaufgaben der Jahrgänge 2001-2008 zu lernen und zu wiederholen. Es richtet sich somit in erster Linie an Studierende der Informatik für das Lehramt an Gymnasien als Vorbereitung für das erste Staatsexamen, eignet sich jedoch gleichermaßen als studienbegleitende Lektüre für alle Informatikstudierenden sowie für den begleitenden Einsatz im Unterricht der Mittel- und Oberstufe an der Schule.

Erlangen, im November 2008 S. Leibinger, U. Kiesmüller

Inhaltsverzeichnis

Herbst 06 - Thema 2

Aufgabe 1

kontextfreie Grammatik, reguläre Grammatik, Ableitungsregel, Produktion

Gegeben seien ein Variablenalphabet $V = \{A, B\}$ und ein Terminalalphabet $T = \{a, b\}$. Es sei $G_1 = (V, T, P_1, A)$ die kontextfreie Grammatik mit den Produktionen

$$P_1 = \{A \to aAb|bB, B \to bB|aB|\lambda\}$$

und $G_2 = (V, T, P_2, A)$ die kontextfreie Grammatik mit den Produktionen

$$P_2 = \{A \to aAb|Ba, B \to bB|aB|\lambda\},$$

wobei λ für das leere Wort steht.

a) Welches sind die von G_1 bzw. G_2 generierten Sprachen $L(G_1)$ bzw. $L(G_2)$? Geben Sie Beschreibungen von $L(G_1)$ und $L(G_2)$, die nicht auf die Grammatiken G_1 bzw. G_2 Bezug nehmen! Beweisen Sie Ihre Behauptungen!

b) Zeigen Sie, dass die kontextfreien Sprachen $L(G_1)$ und $L(G_2)$ nicht regulär sind!

c) Welches sind die Sprachen $L(G_1) \cup L(G_2)$ und $L(G_1) \cap L(G_2)$? Welche dieser Sprachen ist regulär?

a) Bei $L(G_1)$ entstehen Worte der Form $a^n b(a, b)^* b^n$. Am Anfang des Wortes stehen somit genauso viele a wie am Ende b. Nach den a folgt zwangsläufig ein b und dann beliebige Kombinationen von a und b.

Ableitungsregeln:

1) $A \longrightarrow aAb$
2) $A \longrightarrow bB$
3) $B \longrightarrow bB$
4) $B \longrightarrow aB$
5) $B \longrightarrow \lambda$

Beweis:
Durch n-maliges Anwenden von Regel 1) entsteht $a^n Ab^n$ ($n \in \mathbb{N}_0$, da die Regel auch gar nicht angewendet werden darf). Mit Regel 2) wird das einzelne b abgeleitet. Die Regeln 3) und 4) können beliebig oft angewendet werden (auch abwechselnd) und produzieren $(a, b)^*$. Mit Regel 5) wird das Wort abgeschlossen.

Die Worte von $L(G_2)$ werden durch $a^n(a, b)^* ab^n$ beschrieben. Auch hier stehen vorne genauso viele a wie hinten b, vor den b muss ein a stehen und zwischen den a^n und ab^n dürfen beliebige Kombinationen von a und b stehen.

Ableitungsregeln:

1) $A \longrightarrow aAb$
2) $A \longrightarrow Ba$
3) $B \longrightarrow bB$
4) $B \longrightarrow aB$
5) $B \longrightarrow \lambda$

Beweis:
Wiederum produziert Regel 1) $a^n Ab^n$ und die Regeln 3) bis 5) $(a,b)^*$. Durch Regel 2) wird ein einzelnes a vor den b^n eingefügt.

b) In beiden Fällen verletzt $A \longrightarrow aAb$ (Regel 1) die Bedingungen für eine reguläre Grammatik. Da es nicht möglich ist, diese Regel in mehrere äquivalente reguläre Regeln umzuformen, ist es auch nicht möglich eine reguläre Grammatik für $L(G_1)$ bzw. $L(G_2)$ anzugeben.

c) $L(G_1) \cap L(G_2)$ sind alle Wörter, bei denen am Ende genauso viele b stehen wie a am Anfang - jeweils getrennt von einem a bzw b. Sie haben also die Form $a^n b(a,b)^* ab^n$. Auch hier lässt sich die Grammatikregel $A \longrightarrow aAb$ nicht vermeiden,
$L(G_1) \cap L(G_2)$ ist also nicht regulär.
$L(G_1) \cup L(G_2)$ enthält alle Wörter der Form $a^n(a,b)^+ b^n$ und ist ebenfalls nicht regulär.

Aufgabe 2

Zahldarstellung, reguläre Sprache

Es sei $\Sigma = \{0,1\}$. Das Alphabet Σ_2 bestehe aus allen Paaren von Elementen aus Σ, geschrieben als Spaltenvektoren der Länge 2 über Σ, also

$$\Sigma_2 = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

Ein Wort $w = w_1 w_2 ... w_n \in \Sigma_2^n$ mit $w_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, $1 \leq i \leq n$, kann aufgefasst werden als ein Paar $(x,y) \in \Sigma^n \times \Sigma^n$ mit $x = x_1 x_2 ... x_n, y_1 y_2 ... y_n$, d. h.

$$w = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} ... \begin{pmatrix} x_n \\ y_n \end{pmatrix},$$

wobei λ für das leere Wort steht.

a) Jedes Wort $a = a_1 a_2 ... a_{n-1} \in \Sigma^n$ stellt die natürliche Zahl $bin(a) = a_1 \cdot 2^{n-1} + a_2 \cdot 2^{n-2} + ... a_{n-1} \cdot 2 + a_n$ dar (binäre Zahldarstellung). Die Sprache des Größenvergleichs ist

$$LEQ := \left\{ w = \begin{pmatrix} x \\ y \end{pmatrix} \in \Sigma_2^*; bin(x) \leq bin(y) \right\}.$$

Es gilt also beispielsweise

$$\begin{pmatrix} 0101 \\ 0110 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \in LEQ,$$
$$\begin{pmatrix} 0110 \\ 0101 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \notin LEQ$$

Zeigen Sie, dass die Sprache LEQ regulär ist!

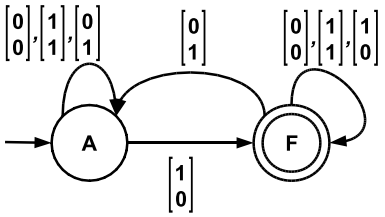
Hinweis:
Sie können - falls Ihnen das hilfreich erscheint - hier die Tatsache verwenden, dass eine Sprache L genau dann regulär ist, wenn die gespiegelte Sprache $L^R = \{w^R; w \in L\}$ regulär ist. Dabei ist $(w_1 w_2 ... w_{n-1} w_n)^R = w_n w_{n-1} ... w_2 w_1$.

b) Zeigen Sie, dass die Sprache

$$\left\{ w = \begin{pmatrix} x \\ y \end{pmatrix} \in \Sigma_2^*; y = x^R \right\}.$$

nicht regulär ist!

a) Betrachten Sie die komplementäre Sprache, die nur Wörter mit $bin(x) > bin(y)$ enthält. Konstruieren Sie dann einen deterministischen endlichen Automaten, der die Worte jeweils von hinten abarbeitet, also zur gespiegelten Sprache gehört.



$$M = (\{S, F\}, \Sigma_2, \delta, S, \{F\})$$

$$\delta(S, \begin{bmatrix}1\\0\end{bmatrix}) = F \qquad \delta(F, \begin{bmatrix}1\\0\end{bmatrix}) = F$$

$$\delta(S, \begin{bmatrix}0\\1\end{bmatrix}) = S \qquad \delta(F, \begin{bmatrix}0\\1\end{bmatrix}) = S$$

$$\delta(S, \begin{bmatrix}0\\0\end{bmatrix}) = S \qquad \delta(F, \begin{bmatrix}0\\0\end{bmatrix}) = F$$

$$\delta(S, \begin{bmatrix}1\\1\end{bmatrix}) = S \qquad \delta(F, \begin{bmatrix}1\\1\end{bmatrix}) = F$$

Damit ist gezeigt, dass $\bar{L}^R$ regulär ist, was dann auch für die Sprache L (die regulären Sprachen sind unter Komplementbildung abgeschlossen, s. [THEO] 1.2.6) gilt.

b) Das klassische Werkzeug für den Nachweis der Nicht-Regularität ist das Pumping-Lemma:

Sei L eine reguläre Sprache. Dann gibt es eine Zahl n, so dass sich alle Wörter $x \in L$ mit $\mid x \mid \geq n$ zerlegen lassen in $x = uvw$, so dass folgende Eigenschaften erfüllt sind:

1. $\mid v \mid \geq 1$
2. $\mid uv \mid \leq n$
3. $\forall_{i \in \mathbb{N}_0} uv^i w \in L$

Kann man also nachweisen, dass eine Zerlegung mit den genannten Eigenschaften nicht existiert, so steht damit fest, dass die zu Grunde liegende Sprache nicht regulär ist.

Es sei $L_2 = \{z = \begin{bmatrix}x\\y\end{bmatrix} \in \Sigma_2^* \; ; \; y = x^R\}$.

Wähle $z \in L, z = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ x_n & x_{n-1} & \cdots & x_1 \end{bmatrix}$

1. Fall: $|\ z\ |$ ungerade, $k = \frac{n+1}{2}$

z lässt sich zerlegen in

$$z = uvw = \underbrace{\begin{bmatrix} x_1 & x_2 & \cdots & x_{k-1} \\ x_n & x_{n-1} & \cdots & x_{k+1} \end{bmatrix}}_{u} \underbrace{\begin{bmatrix} x_k \\ x_k \end{bmatrix}}_{v} \underbrace{\begin{bmatrix} x_{k+1} & \cdots & x_{n-1} & x_n \\ x_{k-1} & \cdots & x_2 & x_1 \end{bmatrix}}_{w}$$

Das Aufpumpen von z ergibt $z' = uv^iw$
$\Rightarrow x' = u_x x_k^i w_x\ ,\ y' = u_y x_k^i w_y$
$u_x = w_y^R\ ,\ u_y = w_x^R$ und $v_x = v_y$
$y' = x'^R \Rightarrow z' \in L$

2. Fall: $|\ z\ |$ gerade, $k = \frac{n}{2}$

z lässt sich zerlegen in

$$z = \underbrace{\begin{bmatrix} x_1 & x_2 & \cdots & x_{k-1} \\ x_n & x_{n-1} & \cdots & x_{k+2} \end{bmatrix}}_{u} \underbrace{\begin{bmatrix} x_k & x_{k+1} \\ x_{k+1} & x_k \end{bmatrix}}_{v} \underbrace{\begin{bmatrix} x_{k+2} & \cdots & x_{n-1} & x_n \\ x_{k-1} & \cdots & x_2 & x_1 \end{bmatrix}}_{w}$$

Das Aufpumpen von z ergibt wiederum $z' = uv^iw$
$\Rightarrow x' = u_x(x_k x_{k+1})^i w_x\ ,\ y' = u_y(x_{k+1}x_k)^i w_y$
$u_x = w_y^R\ ,\ u_y = w_x^R$ und $v_x = v_y^R$
$y' = x'^R \Rightarrow z' \in L$

Dies führt also nicht zum gewünschten Widerspruch und damit liefert das Pumping-Lemma in diesem Fall leider keinen Beweis für die Nicht-Regularität. Denn aus der Regularität folgt zwar nach dem Pumping-Lemma die Existenz einer Zerlegung, die umgekehrte Folgerung stimmt aber nicht.

Es wird hier somit ein anderes Werkzeug z. B. die Argumentation über endliche Automaten benötigt:
Angenommen L_2 wäre regulär. Dann gäbe es einen endlichen Automaten, der alle $z \in L$ akzeptiert. Der Automat müsste $\begin{bmatrix} x_1 \\ x_n \end{bmatrix}$ mit $\begin{bmatrix} x_n \\ x_1 \end{bmatrix}$ vergleichen; dies ist aber bei endlichen Automaten nicht möglich, da der Automat nicht die Länge des Wortes kennt und das Wort nur linear abarbeiten kann.
Da es nicht möglich ist, einen endlichen Automaten zu konstruieren, ist L_2 nicht regulär.

die niemand (auch nicht in Mischung) verkostet hat. Die vergiftete Flasche kann aus der Kombination der gestorbenen Vorkoster jeweils eindeutig identifiziert werden.

Variante 2

	1	2	3	4	5	6	7	8
Vorkoster 1	■	■	■	■				
Vorkoster 2	■	■			■	■		
Vorkoster 3	■		■		■		■	

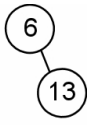
Es werden nur drei Vorkoster eingesetzt und jeder trinkt eine Mischung aus vier Flaschen. Auch hier kann wieder in Abhängigkeit der Toten eindeutig auf die vergiftete Flasche geschlossen werden.

Herbst 06 - Thema 1

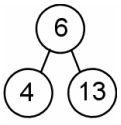
Aufgabe 4

AVL-Baum

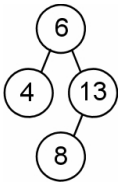
a) Gegeben sei die folgende Folge ganzer Zahlen: 6, 13, 4, 8, 11, 9, 10.
- Fügen Sie obige Zahlen der Reihe nach in einen anfangs leeren AVL-Baum ein und stellen Sie den Baum nach jedem Einfügeschritt dar.
- Löschen Sie das Wurzelelement des entstandenen AVL-Baums und stellen Sie die AVL-Eigenschaft wieder her!



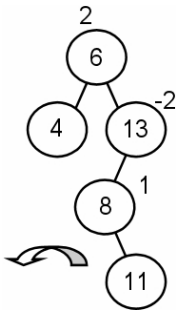
einfügen der 6 als Wurzel; rechts anhängen der 13, da $13 > 6$



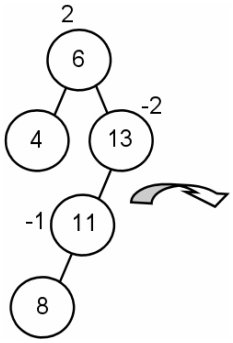
links anhängen der 4, da $4 < 6$



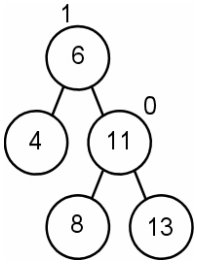
$8 > 6$, deshalb rechter Teilbaum; $8 < 13$, also links anhängen



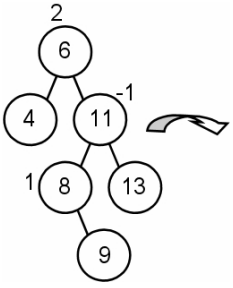
$11 > 6, 11 < 13, 11 > 8$
deshalb an linken Teil des rechten Teilbaums 11 anhängen; Balancierung verletzt, deshalb Linksrotation



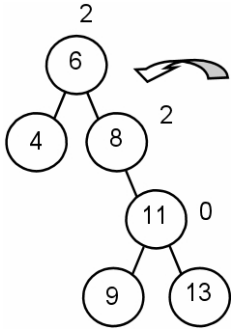
Balancierung weiter verletzt (-2); deshalb Rechtsrotation



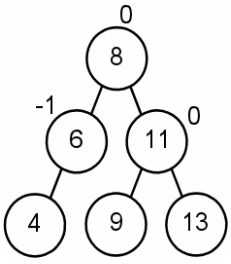
Balancierung wieder hergestellt



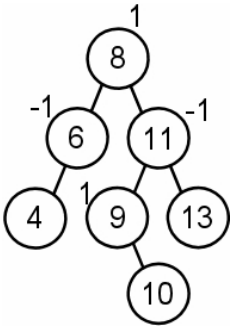
$9 > 8$, deshalb einfügen rechts unter der 8; Balancierung verletzt, deshalb Rechtsrotation



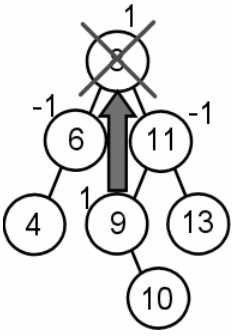
Balancierung weiter verletzt (2); deshalb Linksrotation



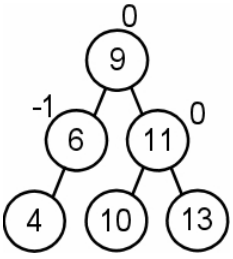
Balancierung wieder hergestellt



9 einfügen
Endergebnis Teil 1 mit korrekter Balancierung



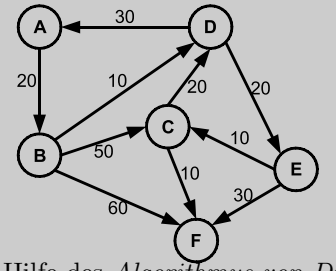
kleinster Knoten des rechten Teilbaums nach oben



Endergebnis

Algorithmus von Dijkstra, Tiefendurchlauf

b) Gegeben sei der folgende gerichtete und gewichtete Graph:



- Bestimmen Sie mit Hilfe des *Algorithmus von Dijkstra* die kürzesten Wege vom Knoten A zu allen anderen Knoten! Geben Sie dabei nach jedem Verarbeitungsschritt den Zustand der Hilfsdatenstruktur an.
- Skizzieren Sie einen Algorithmus zum Tiefendurchlauf von gerichteten Graphen, wobei jede Kante nur einmal verwendet werden darf!

b)

S	Heap		A	B	C	D	E	F
$\{\}$	$\{A,B,D,E,C,F\}$	dis	0	∞	∞	∞	∞	∞
		from	null	null	null	null	null	null
$\{A\}$	$\{B,D,E,C,F\}$	dis	0	20	∞	∞	∞	∞
		from	null	A	null	null	null	null
$\{A,B\}$	$\{D,C,F,E\}$	dis	0	20	70	30	∞	80
		from	null	A	B	B	null	B
$\{A,B,D\}$	$\{E,C,F\}$	dis	0	20	70	30	50	80
		from	null	A	B	B	D	B
$\{A,B,D,E\}$	$\{C,F\}$	dis	0	20	60	30	50	80
		from	null	A	E	B	D	B
$\{A,B,D,E,C\}$	$\{F\}$	dis	0	20	60	30	50	70
		from	null	A	E	B	D	C
$\{A,B,D,E,C,F\}$	$\{\}$	dis	0	20	60	30	50	70
		from	null	A	E	B	D	C

Prinzipiell wird beim Tiefendurchlauf nur irgendein noch nicht besuchter Nachbarknoten im nächsten Schritt abgearbeitet. Verwendet man immer den Nachbarknoten mit dem geringsten Abstand zum Startknoten, so resultiert der Algorithmus von Djikstra. Es sei V die Menge aller Knoten, D[v] der Abstand (Distanz) eines Knotens v zum Startknoten u. *(s. [DUD], Stichworte „Djikstra-Algorithmus“ bzw. „Tiefensuche“)*

```
setze D[v] = ∞ für alle Knoten v;
D[u] = 0, S := {u}
while V ≠ ∅ do
```

```
deletemin(V, v); //entferne den Knoten mit minimalem Abstand
S:= S ∪ {u};
for all Nachbarknoten w von v do
    h:= D[v] + d(v, w) //neuer Abstand
    if (h < D[w]) //nur wenn Abstand kleiner ist
       then D[w] := h;
```

c) Ein wesentlicher Nachteil der Standardimplementierung des QUICKSORT-Algorithmus ist dessen rekursiver Aufruf.

Quicksort-Algorithmus

- Implementieren Sie den Algorithmus QUICKSORT **ohne** den rekursiven Prozeduraufruf!

c)
```
public class Quicksort{
// Implementierung eines Stacks:
int pos;
int [] stack;
public boolean isempty() {return( (pos == 0)); }
//Prüfung, ob Stapel leer ist
public void push(int i) {stack[++pos] = i; }
//Ablegen eines Wertes
public int top() {return( stack[pos] ); }
//Auslesen des obersten Wertes
public void pop() { if (pos > 0) pos--; }
//Entfernen des obersten Wertes

// Zu sortierende Liste
int [] A;
int N;

Quicksort() {
     stack = new int[50];
     pos = 0;
     A = new int[10];
     A[0] = 5; A[1] = 60; A[2] = 23; A[3] = 90;
     A[4] = 24; A[5] = 3;
     N = 6;
     //Länge der Liste - könnte auch als eigene Methode
     //implementiert werden
}

void Go(){
     for(int i = 0; i < N; i++) System.out.print(A[i] + „,  \);
     System.out.println();
     //Ausgabe der unsortierten Liste
     Sort();
```

```
      for(int i = 0; i < N; i++) System.out.print(A[i] + „,  \);
      System.out.println();
      //Ausgabe der sortierten Liste
}

void Sort(){
      int i, j, links, rechts, x, w;

      links = 0;
      //Index des ersten Elements
      rechts = N-1;
      //Index des letzten Elements

      push (links);
      push (rechts);
      //Auf dem Stapel werden jeweils die Indizes der Enden
      //der gerade aktuellen (Teil-)Liste abgelegt.

      while (!isempty()){
            if (rechts > links){
                  x = A[(links + rechts)/2];
                  //Wert des mittleren Elements wird
                  //als Pivotelement verwendet
                  i = links;
                  j = rechts;
      //In der folgenden while-Schleife werden alle Elemente,
      //die größer sind als das Pivotelement, in die rechte
      //Teilliste sortiert, alle anderen in die linke Teilliste.
                  while (i<j){
                        while (A[i]<x) { i++; }
                        while (A[j]>x) { j--; }
                        if(i<=j){
                                w = A[i];
                                A[i] = A[j];
                                A[j] = w;
                                i = i + 1;
                                j = j - 1;
                        }
                  }
      //Nun wird die Liste in Teillisten zerlegt.
                  if ((i-links) >= (rechts-i)){
                        push (links);
                        push (i - 1);
                        links = i ;
                  }
                  else{
                        push (i + 1);
                        push (rechts);
                        rechts = i - 1;
```

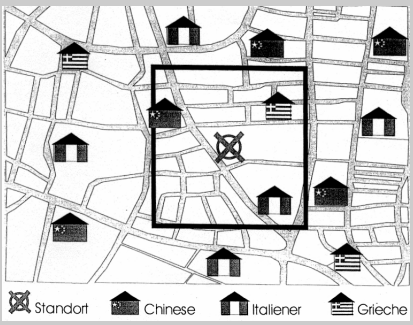
```
                }
            }
            else{ rechts = top();
                pop();
                links = top();
                pop();
}}  }   }
```

Herbst 06 - Thema 2

Aufgabe 4

Algorithmus, Liste, Laufzeit

Ein Navigationssystem soll folgenden Service anbieten. Ausgehend von einem aktuellen Standort eines Anfragenden sollen alle Restaurants einer bestimmten Küchenrichtung (z. B. italienisch, chinesisch) ausgegeben werden, die sich innerhalb eines quadratischen Bereichs einer anzugebenden Größe um diesen befinden.



Gehen Sie vereinfachend davon aus, dass sowohl der Standort, als auch aller Restaurants jeweils mit ihren (x,y)-Koordinaten vorliegen, wobei $0 \leq x \leq xmax$ und $0 \leq y \leq ymax$ gelten soll. Verwenden Sie zur Formulierung von Algorithmen bzw. Datentypen eine gängige höhere Programmiersprache oder einen entsprechenden Pseudocode! Erläutern Sie Ihre Lösung ausgiebig durch Kommentare!

a) Geben Sie einen geeigneten Datentyp zur Verwaltung der Restaurants an! Zusätzlich zur Lage ((x,y)-Koordinaten) soll der Name, die Adresse, die Telefonnummer und die Küchenrichtung angegeben werden!

b) Geben Sie einen Algorithmus an, der als Eingabe einen Standort in (x,y)-Koordinaten, eine Bereichsgröße und eine bevorzugte Küchenrichtung erhält und der als Ergebnis eine Datenstruktur liefert, die alle Restaurants dieser Richtung innerhalb eines achsenparallelen, quadratischen Bereichs um den Standort enthält!
   Lösungshinweis:
   Eine mögliche Strategie besteht darin, zunächst nur die Restaurants mit passender x-Koordinate zu identifizieren und aus diesen diejenigen mit passender y-Koordinate auszuwählen.

c) Geben Sie die Laufzeit Ihres Verfahrens in O(n)-Notation an und begründen Sie Ihr Ergebnis!

Best-Fit:

Segment:	1000KB	1MB	200KB	110KB	4MB
1000KB	0KB				
900KB		124KB			
200KB			0KB		
100KB				10KB	
110KB		14KB			
2MB					2MB

Worst-Fit:

Segment:	1000KB	1MB	200KB	110KB	4MB
1000KB					3096KB
900KB					2196KB
200KB					1996KB
100KB					1896KB
110KB					1786KB
2MB	kein ausreichend großes Segment mehr				

c) Worst-Fit schneidet am schlechtesten ab, weil die letzte Speicheranforderung nicht mehr erfüllbar ist.
First-Fit ist in der Fragmentierung ein kleines bisschen effizienter, weil nur ein unbrauchbares Speichersegment übrig bleibt, während es bei Best-Fit zwei sind.

Herbst 06 - Thema 1

Aufgabe 1

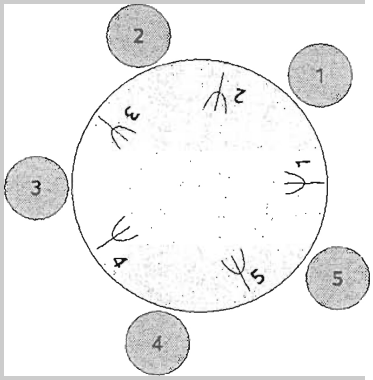
Verklemmung,

1.1 Definieren Sie den Begriff Verklemmung!

1.2 Welche Bedingungen müssen erfüllt sein, damit eine Verklemmung entsteht?

1.3 Das Philosophenproblem

Fünf Philosophen sitzen an einem runden Tisch, in dessen Mitte ein immer voller Teller mit Nudeln steht. Rechts und links von jedem Philosophen liegt jeweils eine Gabel. Zum Essen benötigt ein Philosoph zwei Gabeln. Die Philosophen sind nur mit zwei Dingen beschäftigt: Denken und Essen. Ist einer hungrig, so greift er zuerst nach der linken und dann nach der rechten Gabel und fängt an zu essen. Ist eine Gabel belegt, so wartet er, bis sie wieder frei ist. Hat der Philosoph sich satt gegessen, so legt er die Gabeln wieder an ihren Platz und fährt fort mit Denken.



Philosophenproblem, ganzzahliger Semaphor

a) Betrachten Sie unten stehenden Pseudocode, der einen einzelnen Philosophen simuliert! Erläutern Sie in einem Satz, wie es in dieser Lösung zu einer Verklemmung kommen könnte!

```
N=5; Procedure phil(i:INTEGER)
       WHILE (TRUE) DO
         think;                 //Philosoph denkt nach
         take_fork(i);          //nimm die linke Gabel
         take_form((i+1)%N);    //nimm die rechte Gabel,
                                  % ist Modulo-Operator
         eat;                   //essen
         put_fork(i);           //lege linke Gabel zurück
         put_fork((i+1)%N);     //lege rechte Gabel zurück
       END;
END phil;
```

b) Neben einer Verklemmung könnte noch eine weiteres Problem auftauchen. Erläutern Sie kurz welches!

1.1 Mit *Deadlock* oder *Verklemmung* bezeichnet man in der Informatik einen Zustand, bei dem ein oder mehrere Prozesse auf Betriebsmittel warten, die dem Prozess selbst oder einem anderen beteiligten Prozess zugeteilt sind. *(vgl. [TAN])*

1.2 *(siehe Herbst 01 Thema 1 Aufgabe 2.2 auf Seite 230)*

1.3 a) Wenn die Prozesse aller fünf Philosophen gleichzeitig starten, dann nimmt jeder Philosoph zuerst die linke Gabel auf, kann dann aber nicht mehr die rechte Gabel nehmen, da diese bereits sein Nachbar hat. Damit liegt eine Verklemmung vor.

b) Es handelt sich hierbei um das Fairness-Problem, d. h. es ist nicht garantiert, dass jeder Philosoph zum Essen kommt. Es kann vorkommen,

dass immer der selbe Philosoph warten muss und er dadurch aushungern kann.

1.4 An einem Flughafen gibt es eine Lagerhalle, an der Transportflugzeuge ihre Ware abliefern und aufnehmen können. Lieferanten bringen jeweils zwei Kisten, während Abholer jeweils nur eine Kiste abholen. Zum Be- und Entladen fliegen die Flugzeuge zuerst über eine Start- und Landebahn ein und fahren dann vor die Lagerhalle. Dort steht ein Kran, der die Flugzeuge be- und entlädt. Danach fliegen die Flugzeuge wieder von der gleichen Start- und Landebahn weg. Beachten Sie folgende Bedingungen:

- Die Start- und Landebahn kann jeweils nur von einem Flugzeug befahren werden.
- Vor der Lagerhalle ist unbegrenzt Platz, so dass dort mehrere Lieferanten und Abholer stehen können.
- Die Lagerhalle hat eine Kapazität von höchstens 30 Kisten.
- Lieferanten dürfen nur zur Lagerhalle fahren, wenn die Lagerhalle noch für alle Kisten Kapazitäten hat. Ansonsten müssen sie nach dem Landen warten.
- Abholer dürfen nur zur Lagerhalle fahren, wenn noch mindestens eine Kiste in der Lagerhalle ist. Ansonsten müssen sie nach dem Landen warten.
- Es gibt nur einen Kran, der Flugzeuge nacheinander be- und entladen kann.
- Der Kran kann erst zu einem Flugzeug fahren, wenn es zur Lagerhalle gefahren ist.
- Ein Flugzeug darf erst zur Startbahn fahren, nachdem der Kran vom Flugzeug abgefahren ist.
- Die Lagerhalle ist zu Beginn leer.

1.5 Die Prozesse sehen folgendermaßen aus:

```
Lieferant                          Abholer                          Kran
{                                  {                                {
 while (TRUE)                       while (TRUE)                     while (TRUE)
 {                                  {                                {
  <auf Start-/Landebahn landen>      <auf Start-/Landebahn landen>    <Anfahrt zu Flugzeug>
  <zur Lagerhalle fahren>            <zur Lagerhalle fahren>          <be-/entlade Flugzeug>
  <2 Pakete abliefern>               <1 Paket holen>                  <Abfahrt von Flugzeug>
                                                                     }
  <auf Start-/Landebahn starten>     <auf Start-/Landebahn starten> }
 }                                   }
}                                   }
```

Vervollständigen Sie mit Hilfe von Semaphoren die obigen Prozesse so, dass es zu keiner Verklemmung kommen kann! Verwenden Sie ganzzahlige Semaphore und geben Sie zu den verwendeten Semaphoren die Startwerte an! Sperrphasen sind möglichst kurz zu halten.

1.4
```
var landebahn,nochFreiePlaetze,belegtePlaetze,kran:semaphore;
                                  //Definition der Semaphoren
    landebahn := 1;               //die Landebahn ist zu Beginn leer
```

```
    nochFreiePlaetze := 30;
    belegtePlaetze := 0;
    kran := 1;              //der Kran ist frei

Lieferant
{ while (TRUE)
{   down(landebahn);        //exklusiver Zugriff auf die
                            //Start-/Landebahn
    <landen> ;
    up(landebahn);          //Start-/Landebahn freigeben
    nochFreiePlaetze = nochFreiePlaetze - 2;
    <zur Lagerhalle fahren>;//nur wenn noch 2 Plätze frei
    down(kran);             //exklusive Belegung des Krans
    Kran();
    <zwei Pakete abliefern>;
    up(kran);               //Kran freigeben
    belegtePlaetze = belegtePlaetze + 2;
                            //Sperrzeit für Kran gering
    <zur Start-/Landebahn fahren>;
    down(landebahn);        //exklusiver Zugriff auf die
                            //Start-/Landebahn
    <starten>;
    up(landebahn);          //Start-/Landebahn freigeben
}   }

Abholer
{ while (TRUE)
{   down(landebahn);        //exklusiver Zugriff auf die
                            //Start-/Landebahn
    <landen>;
    up(landebahn);
    down(belegtePlaetze);   //nur zur Halle fahren,
                            //wenn noch mind. 1 Kiste da ist
    <zur Lagerhalle fahren>;
    down(kran);             //exklusive Belegung des Krans
    Kran();
    <ein Paket abholen>;
    up(kran);               //Kran freigeben
    up(nochFreiePlaetze);
    <zur Start-/Landebahn fahren>;
    down(landebahn);        //exklusiver Zugriff auf die
                            //Start-/Landebahn
    <starten>;
    up(landebahn);          //Start-/Landebahn freigeben
}   }
Kran
{ while (TRUE)
```

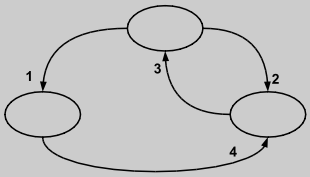
```
{   <Anfahrt zu Flugzeug>;
    <be-/entlade>;
    <Abfahrt von Flugzeug>;
}   }
```

Aufgabe 2

Programm, Prozess, Prozesszustand, Zustandsdiagramm, Schedulingverfahren, Round-Robin, First-come, First-served, priority scheduling

2.1 Erklären Sie kurz den Unterschied zwischen einem Programm und einem Prozess!

2.2 Nennen Sie die Zustände, die ein Prozess aus Betriebssystemsicht einnehmen kann! Übertragen Sie dafür rechts abgebildetes Zustandsdiagramm auf Ihr Arbeitsblatt! Tragen Sie die Zustände in die Kreise ein und benennen Sie die Zustandsübergänge 1 bis 4!



2.3 Erläutern Sie kurz die Vorgehensweisen der folgenden Scheduling-Algorithmen:

- Round Robin
- First-come, First-served
- priority scheduling (statisch)

Gehen Sie auch auf die Kriterien Fairness, Echtzeitfähigkeit und unterbrechend ein!

2.4 Fünf Stapelverarbeitungsaufgaben A bis E kommen in einem Rechenzentrum an. Die folgende Tabelle zeigt für jede Stapelverarbeitungsaufgabe die Ankunftszeit, die Laufzeit sowie die Priorität. Die Priorität wächst mit steigenden Zahlen.

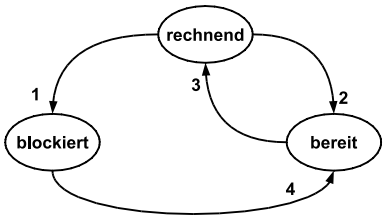
	Ankunftszeit	Laufzeit	Priorität
A	0ms	10ms	3
B	1ms	6ms	5
C	2ms	2ms	2
D	3ms	4ms	1
E	4ms	8ms	4

a) Bestimmen Sie für jeden der in Aufgabe 2.3 genannten Scheduling-Algorithmen die durchschnittliche Prozessdurchlaufzeit (Verweilzeit)! Vernachlässigen Sie dabei den Overhead des Prozesswechsels! Die Zeitscheibe für Round Robin betrage 2ms.

b) Welchen Algorithmus würden Sie auswählen? Begründen Sie kurz Ihre Entscheidung!

2.1 *(siehe Frühjahr 07 Thema 2 Aufgabe 1 auf Seite 304)*

2.2 a) Prozess blockiert z. B. wegen Eingabe
b) Scheduler wählt anderen Prozess
c) Scheduler wählt diesen Prozess
d) Eingabe vorhanden, Blockade aufgehoben

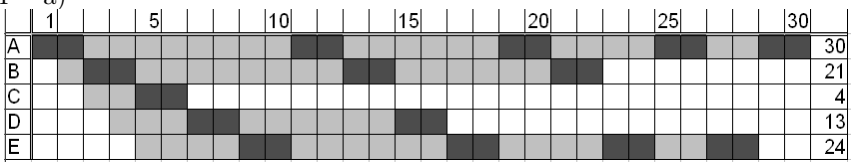


2.3 *Round Robin:* Dabei werden alle rechenbereiten Jobs in einer Warteschlange angeordnet. Jeweils der vorderste Job wird aus der Schlange genommen, bekommt für eine bestimmte, kurze Zeitspanne den Prozessor zugeteilt und wird dann, falls er mehr Zeit benötigt, erneut hinten an die Warteschlange angestellt. Neu hinzukommende Jobs werden ebenfalls an das Ende der Schlange gestellt. Die Zeitspanne, oder auch Zeitscheibe genannt, ist immer gleich groß, typischerweise in Größenordnungen von 10 bis 50 Millisekunden. Round Robin ist fair, unterbrechend und im Allgemeinen nicht echtzeitfähig.

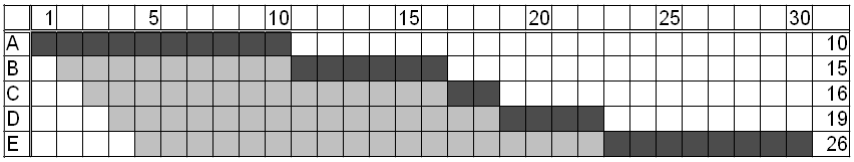
First-come, First-served: Hierbei werden alle Prozesse in der Reihenfolge ihres Eingangs bearbeitet. Dabei werden einzelne Prozesse immer komplett verarbeitet, bevor der nächste Prozess an die Reihe kommt. Diese Strategie erzielt eine gute Auslastung bezüglich der CPU, allerdings nicht bezüglich der Ressourcen, die längere Zeit für eine Anforderung benötigen können, wie z. B. Ein-/Ausgabe oder Massenspeicher. FCFS ist nicht fair, arbeitet ohne Priorität, und ist weder unterbrechend noch echtzeitfähig.

priority scheduling (statisch): Bei dieser Strategie wird jedem Prozess eine Priorität zugeordnet. Die Abarbeitung erfolgt dann in der Reihenfolge der Prioritäten. Priority scheduling ist echtzeitfähig, die Fairness wird durch die Prioritäten geregelt. Wir gehen im folgenden von der unterbrechenden Variante dieser Strategie aus - je nach Literaturquelle ist auch eine nicht-unterbrechende Variante möglich.

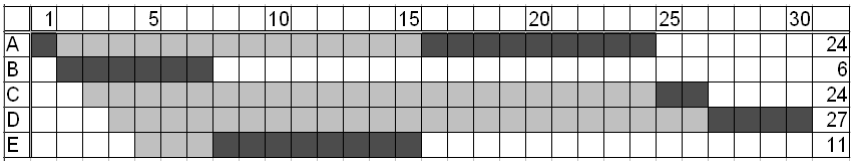
2.4 a)



Round Robin: $(28 + 21 + 4 + 15 + 26)/5 = 94/5 = 18,8$



First-come, First-served: $(10 + 15 + 16 + 19 + 26)/5 = 86/5 = 17,2$



priority scheduling: $(24 + 6 + 24 + 27 + 11)/5 = 92/5 = 18,4$

b) Für Stapelverarbeitung sind keine unterbrechenden Verfahren geeignet, so dass FCFS ausgewählt wird. Ginge man beim priority scheduling von der nicht-unterbrechenden Variante aus, müsste als zweites Argument die durchschnittliche Verweilzeit (die möglichst kurz sein sollte) hinzugezogen werden.

Aufgabe 3

dynamische Segmentierung, Best Fit, First Fit, Worst Fit

3.1 Eine Arbeitsspeicherverwaltung wende dynamische Segmentierung an und habe in der Freispeicherliste noch drei Segmente von 700, 450 und 300 Speichereinheiten zur Verfügung (Reihenfolge wie Aufzählung). Es treffen nun nacheinander Anforderungen von 100, 300, 400, 300 und 250 Speichereinheiten an. Wie würden mit dem first fit- und dem best fit-Verfahren die Speichereinheiten auf die Segmente verteilt werden? Stellen Sie Ihr Ergebnis in einer zu unten stehender äquivalenten Tabelle dar, vergleichen Sie die Ergebnisse und erläutern Sie die Unterschiede! Geben Sie hierbei jeweils die noch freien Speichergrößen an!

Anforderung	**First Fit**			**Best Fit**		
Segment	Segm. 1	Segm. 2	Segm. 3	Segm. 1	Segm. 2	Segm. 3
Initial	700	450	300	700	450	300
100						
300						
400						
300						
250						

3.2 Als weitere Möglichkeit gibt es das Verfahren worst fit. Erläutern Sie kurz die Verfahrensweise, Vor- und Nachteile!

3.1 First Fit verwendet die erste ausreichend große Lücke und ist damit schneller als Best Fit, das die kleinste, gerade noch ausreichende Lücke sucht. Best Fit testet also immer alle Segmente. Außerdem liefert Best Fit in der Praxis

schlechtere Ergebnisse, da sehr viele kleine, unbrauchbare Lücken entstehen, also Speicherplatz verschwendet wird.

Anforderung	**First Fit**			**Best Fit**		
Segment	Seg. 1	Seg. 2	Seg. 3	Seg. 1	Seg. 2	Seg. 3
Initial	700	450	300	700	450	300
100	600	450	300	700	450	200
300	300	450	300	700	150	200
400	300	50	300	300	150	200
300	0	50	300	0	150	200
250	0	50	50	kein ausreichend großes Segment mehr		

3.2 Worst Fit verwendet immer die größte Lücke, um möglichst große Restlücken zu lassen. Auch Worst Fit liefert in der Praxis keine guten Ergebnisse (es sind schneller als bei anderen Verfahren nur noch Reste übrig, die zu klein sind für große Teile) und ist ebenfalls nicht so schnell wie First Fit.

Herbst 06 - Thema 2

Aufgabe 1

Speicherverwaltung, Seitenadressierung, logische, physi(kali-)sche Adresse, Segmentierung, ganzzahliger Semaphor, Prozesszustand

a) Skizzieren Sie (grafisch) die Abbildung einer logischen Adresse in eine physikalische Adresse in einem System mit Segmentierung!

b) Was muss das Betriebssystem tun, wenn aufgrund von Hauptspeichermangel ein Segment ausgelagert werden soll? Beschreiben Sie den Ablauf und welche Änderungen in welchen der in Teilaufgabe a) beschriebenen Datenstrukturen vorgenommen werden müssen!

c) Was passiert, wenn der Prozess nach dem Auslagern erneut auf die Daten des Segments zugreift? Beantworten Sie in Ihrer Beschreibung vor allem die Fragen:
   - Welche Einheit des Systems erkennt, dass das Segment nicht vorhanden ist?
   - Woran wird das erkannt?
   - Welche Aktivitäten finden daraufhin im Betriebssystem oder in der Anwendung statt?
   - Welche Datenstrukturen werden in diesem Zusammenhang wie modifiziert?
   - Welche Prozesszustände nimmt der betroffene Prozess in welcher Phase ein?

d) Ein Segment soll von zwei Prozessen gemeinsam genutzt werden. Was muss das Betriebssystem hierfür tun?

e) Nennen Sie drei wesentliche Unterschiede zwischen Segmentierung und Seitenadressierung!

f)
```
SELECT   Tour.TNR, Fahrer, COUNT(SendungsID)
FROM     Tour, Ladeliste
WHERE    Tour.TNR = Ladeliste.TNR
GROUP BY Tour.TNR;
```

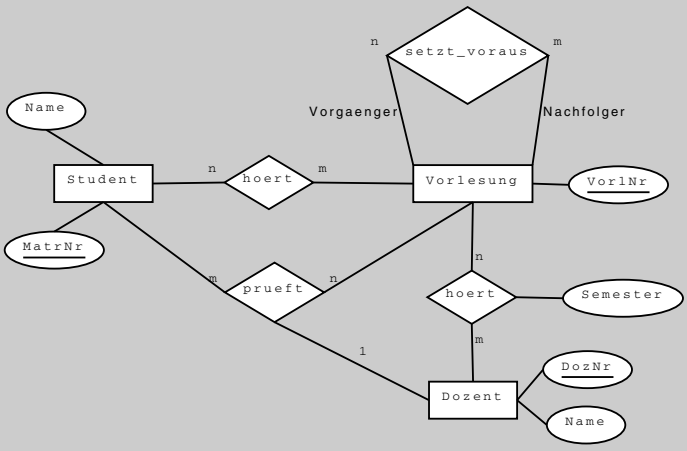
g)
```
SELECT SID, Name, Vorname
FROM   Sendung, Postkunde
WHERE  KNR = AbsenderNR
       AND AbsenderNR = EmpfängerNR;
```

Herbst 07 - Thema 2

Aufgabe 1

ER-Modell, relationales Schema, (Fremd-) Schlüssel, schwache Entität, SQL

Gegeben sei folgender Ausschnitt aus dem ER-Modell einer Universitätsdatenbank:



a) Übersetzen Sie das ER-Modell in das Relationenmodell!

b) Geben Sie für die Tabellen `prüft`, `liest` und `hört` CREATE TABLE-Statements an, welche auch die nötigen Schlüssel- sowie Fremdschlüsselbedingungen enthalten!

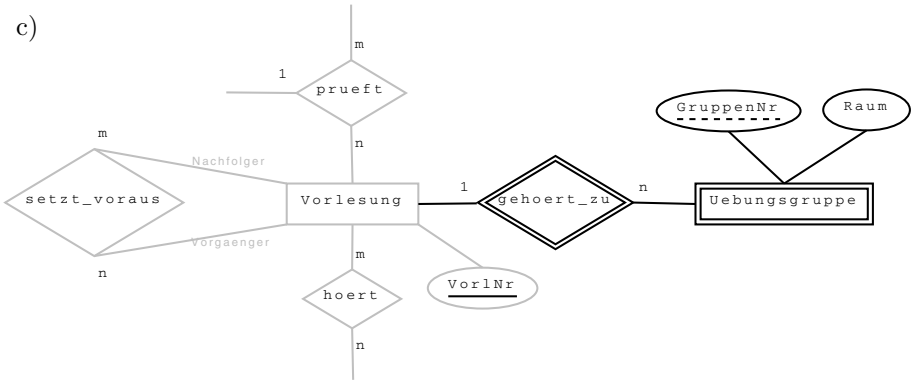
c) Ergänzen Sie das ER-Modell um einen schwachen Entity-Typen `Übungsgruppe` mit den Attributen `Gruppennummer` und `Raum`, der in Verbindung zur Tabelle `Vorlesung` stehen soll!

a) Student(MatrNr: Int, Name: String)
Vorlesung(VorlNr: Int)
Dozent(DozNr: Int, Name: String)
hoert(*MatrNr: Int, VorlNr: Int*)
liest(*DozNr: Int, VorlNr: Int*, Semester: String)
setzt_voraus(*Vorgaenger: Int, Nachfolger: Int*)
prueft(*MatrNr: Int, VorlNr: Int, DozNr: Int*)

b) Viele Aufgabensteller wollen an dieser Stelle erst einmal die Erzeugung der von den explizit angefragten Tabellen benötigten grundlegenden Tabellen sehen. Deshalb hier zuerst die Erzeugung von *Student, Vorlesung, Dozent*.

```
CREATE TABLE Student (
             MatrNr Integer PRIMARY KEY,
             Name Varchar(50));

CREATE TABLE Vorlesung (
             VorlNr Integer PRIMARY KEY);

CREATE TABLE Dozent (
             DozNr Integer PRIMARY KEY,
             Name Varchar(50));

CREATE TABLE prueft (
             MatrNr Integer REFERENCES Student(MatrNr),
             VorlNr Integer REFERENCES Vorlesung(VorlNr),
             DozNr Integer REFERENCES Dozent(DozNr),
             PRIMARY KEY (MatrNr, VorlNr));

CREATE TABLE liest (
             DozNr Integer REFERENCES Dozent(DozNr),
             VorlNr Integer REFERENCES Vorlesung(VorlNr),
             Semester Varchar(25),
             PRIMARY KEY (DozNr, VorlNr));

CREATE TABLE hoert (
             MatrNr Integer REFERENCES Student(MatrNr),
             VorlNr Integer REFERENCES Vorlesung(VorlNr),
             PRIMARY KEY (DozNr, MatrNr));
```

c)



Aufgabe 2

SQL-Anfrage

Gegeben seien die folgenden Tabellen:

- Student(MatNr, Name, Alter, Fachbereich, Fachsemester).
- hört(MatNr, Vorlesung, Semester, Note).

Erstellen Sie in SQL folgende Anfragen:

a) Bestimmen Sie alle Studierende, die älter sind als 23 Jahre!

b) Bestimmen Sie das durchschnittliche Fachsemester der Studierenden des Fachbereichs „`Informatik`“!

c) Bestimmen Sie für jeden Studierenden - gegeben durch die Matrikelnummer - und jedes Semester die Durchschnittsnote für alle gehörten Vorlesungen!

d) Bestimmen Sie für jeden Studierenden - gegeben durch den Namen - und jedes Semester die Anzahl der Vorlesungen, die er in diesem Semester hört, vorausgesetzt, dass der Studierende im betrachteten Semester mindestens fünf Vorlesungen hört!

e) Bestimmen Sie alle Studierenden - gegeben durch den Namen - die dieselbe Vorlesung (mindestens) in zwei verschiedenen Semestern gehört haben!

a)
```
SELECT *
FROM   Student
WHERE  Alter > 23;
```

b)
```
SELECT AVG(Fachsemester)
FROM   Student
WHERE  Fachbereich = 'Informatik';
```

c)
```
SELECT   MatNr, Semester, AVG(Note)
FROM     hoert
GROUP BY Semester, MatNr;
```

d)
```
SELECT   Student.MatNr, Name, Semester, COUNT(Vorlesung)
FROM     Student, hört
WHERE    Student.MatNr = hoert.MatNr
GROUP BY Semester, Student.Matnr
HAVING   COUNT(Vorlesung)≥5;
```

e)
```
SELECT   DISTINCT s.MatNr, Name
FROM     Student s, hoert h1, hoert h2
WHERE    h1.Vorl = h2.Vorl AND s.MatNr = h1.MatNr
         AND h1.MatNr = h2.MatNr AND h1.Semester <> h2.Semester;
```

Aufgabe 3

Relationenschema, funktionale Abhängigkeit, Attributhülle, Syntheseal-gorithmus, 3NF, BCNF

Gegeben sei das Relationenschema $R = (U, F)$ mit der Attributmenge $U = \{A, B, C, D, E\}$ und folgender Menge F von funktionalen Abhängigkeiten:

$$F = \{A \to B, AC \to BD, BC \to A\}$$

a) Bestimmen Sie die Attributhüllen $\{A\}_F^+$ und $\{B, C\}_F^+$.

b) Geben Sie alle Schlüssel für R an!

c) Zerlegen Sie R mittels des Synthesealgorithmus in ein 3NF-Datenbankschema!

d) Ist die Zerlegung von R in die beiden Relationenschemata $R_i = (U_i, F_i)$, $i = 1, 2$, mit $U_1 = \{A, C, D\}$ und $U_2 = \{A, B, C\}$ und entsprechenden Mengen $F_i = \Pi_{v_i}(F^+)$ von funktionalen Abhängigkeiten verlustfrei? Ist sie in BCNF?

a) Bestimmung von $\{A\}_F^+$:
Überprüfe FD $A \to B$: $\Rightarrow \{A\}_F^+ = \{A, B\}$
Überprüfe die FDs $AC \to BD$ und $BC \to A$:
die linke Seite ist jeweils nicht in $\{A\}_F^+$ enthalten, es ergibt sich also keine Änderung mehr an $\{A\}_F^+$.
$\Longrightarrow \{A\}_F^+ = \{A, B\}$

Bestimmung von $\{B, C\}_F^+$:

$BC \to A$	$\{B, C\}_F^+ = \{A, B, C\}$
$A \to B$	keine Änderung an $\{B, C\}_F^+$
$AC \to BD$	$\{B, C\}_F^+ = \{A, B, C, D\}$

b) Da wegen der zweiten FD $\{A, C\}_F^+ = \{A, B, C, D\}$ gilt, bis auf E also jedes Attribut von U erreicht wird, sind die Schlüsselkandidaten $\{A, C, E\}$ und $\{B, C, E\}$.

c) Zuerst muss die minimale Überdeckung bestimmt werden:

1. Zerlege F in einfache FDs:
   $F = \{A \to B, AC \to B, AC \to D, BC \to A\}$
2. Eliminiere redundante Attribute:
   Betrachte FD $AC \to B$: C redundant?
   $\{A\}_F^+ = \{A, B\} \Rightarrow C$ redundant.
   $F' = \{A \to B, AC \to D, BC \to A\}$
   $\{B\}_F^+ = \{B\}, \{C\}_F^+ = \{C\}$ und $D \notin \{A\}_F^+$
   Es gibt also keine weiteren redundaten Attribute.
3. Eliminiere redundante FDs:
   Auf der rechten Seite aller FDs kommen die Attribute nur jeweils einmal vor, so dass es keine redundanten FDs geben kann.

Jetzt kann der Synthesealgorithmus angewendet werden:

1. Erstelle für jede FD eine Relation:
   $$R_1(A, B)$$
   $$R_2(A, C, D)$$
   $$R_3(B, C, A)$$
2. Überprüfe, ob eine der Relationen bereits einen Schlüsselkandidaten enthält:
   Keine Relation enthält einen Schlüsselkandidaten, es ist also noch eine zusätzliche Relation $R_4(A, C, E)$ (oder $R_4(B, C, E)$) notwendig.
3. Überprüfe, ob $R = R_1 \cup R_2 \cup R_3 \cup R_4$. Dies ist erfüllt, es werden keine weiteren Relationen benötigt.
4. Die Relationen R_1 und R_3 können nun zusammengefasst werden:
   $$R_{13}(A, B, C)$$
   $$R_2(A, C, D)$$
   $$R_4(A, C, E)$$

d) *Eine Relation ist in Boyce-Codd Normalform, wenn kein Attribut funktional abhängig von einer Attributgruppe ohne Schlüsseleigenschaft ist.*
$F_1 = \{AC \to D\}$: AC ist einziger Schlüssel von $R_1 \Rightarrow R_1$ ist in BCNF.
$F_2 = \{A \to B, BC \to A\}$ ist nicht in BCNF, da A kein Schlüssel ist.
Die Zerlegung ist also abhängigkeitserhaltend, da $F_1 \cup F_2 = F'$; aber sie ist nicht verlustfrei, da $E \notin U_1 \cup U_2$ ist.