Wiederholungsklausur Informatik II
Sommersemester 2002 (27.09.2002)

Vorname	Klaus
Nachname	Wachtler
Matrikelnummer	12345678
Prüfer	Klaus Wachtler

Die Antworten meiner Musterlösung sind in blau gehalten, während Punktzahlen und Anmerkungen zur Bewertung in rot eingetragen sind.

Allgemeine Hinweise

Fragen mit freien Antworten

Die meisten Fragen sind durch freien (aber bitte lesbaren) Text im entsprechenden Feld zu beantworten. Der vorgesehene Platz ist **NICHT** bindend: Einerseits kann man manche Fragen auch sinnvoll beantworten, ohne den ganzen Platz zu benötigen; andererseits kann man auch auf einem zusätzlichen Blatt fortfahren, wenn man einen Roman schreiben will.

Jedes zusätzliche Blatt bitte mit Name und Matrikelnummer versehen sowie der Nummer der zugehörigen Frage, und spätestens bei der Abgabe mit den ausgeteilten Blättern zusammenheften.

Im Notfall fragen

Wenn eine Frage unklar oder falsch formuliert ist, dann lieber sofort fragen, anstatt mit meinen Fehlern Zeit zu verschwenden.

Wer eine Wissenslücke hat, die man schnell beheben könnte, und wegen dieser Lücke den Rest einer Aufgabe nicht lösen kann, kann sich einen Hinweis geben lassen, der mit dem Anschreiben angemessener Strafpunkte einhergeht. Damit kann er dann hoffentlich den Rest der Aufgabe lösen, und vermeidet dadurch Folgefehler.

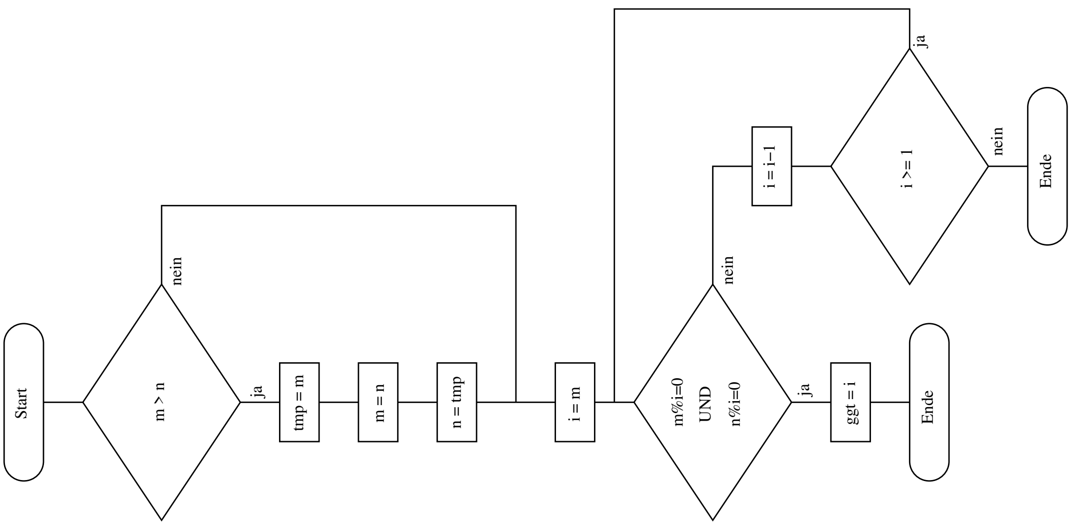
Und jetzt bitte beruhigen, tief Luft holen, und viel Glück!

Prüfungsfragen

1. Geben Sie für den folgenden Algorithmus einen Flußplan an:

```
eigener ggT:
+-------------------------------------+
|               m>n                   |
|                |                    |
+--- ja  --------+--- nein ----------+
| Vertausche m,n |                    |
+-------------------------------------+
|  für i = m ... 1  (-1)              |
|   +---------------------------------+
|   |  m/i ohne Rest teilbar          |
|   |      UND                        |
|   |  n/i ohne Rest teilbar          |
|   |          |                      |
|   +-- nein --+-- ja ----------------+
|   |          |                      |
|   |          | ggT = i              |
|   |          +----------------------+
|   |          | <- eigener ggT       |
+---+----------+----------------------+
```

Hier mein Vorschlag:



Summe: 4

In einer sortierten, doppelt verketteten Liste soll ein Element eingefügt werden. Wie ist vorzugehen (in Worten)?

(a) je nach Sprache ist Platz für das neue Element zu beschaffen (new, malloc() etc.)

1

(b) es ist die Stelle zu finden, an der eingefügt werden soll (je nachdem, wie die Liste sortiert ist).

1

(c) je nach gefundener Position sind folgende Fälle zu unterscheiden:

   i. Falls die Liste bisher leer war: Das neue Element wird zum Listenanfang und -ende; sein Vorgänger und sein Nachfolger werden als ungültig markiert.

   ii. Falls am Anfang eingefügt werden soll: Der Nachfolger des neuen Elements wird auf den bisherigen Listenanfang gesetzt, und der Vorgänger des bisherigen Listenanfangs (bisher: keiner) wird auf das neue Element gesetzt; weiterhin wird das neue Element zukünftig als Listenanfang behandelt und sein Vorgänger als ungültig markiert

2

   iii. Falls am Ende eingefügt werden soll: Das bisherige Listenende wird beim neuen Element als Vorgänger eingetragen, umgekehrt sein Nachfolger auf das neue Element gesetzt. Der Nachfolger des neuen Elements wird als ungültig markiert. Das neue Element wird zum Listenende (falls das als solches gespeichert wird).

2

   iv. Beim Einfügen in der bisherigen Liste werden Anfang und Ende der Liste nicht verändert; man benötigt die beiden Elemente der Liste, zwischen denen eingefügt werden soll (i.F. A und B genannt). Der Nachfolger von A wird auf das neue Element gesetzt, und A wird zum Vorgänger des neuen Elements. Der Vorgänger von B wird ebenfalls auf das neue Element gesetzt, und der Nachfolger des neuen Elements wird zu B gesetzt.

4

Summe: 10

Welche Laufzeitordnung hat ihre Lösung aus der vorigen Aufgabe?

Das eigentliche Einfügen ist von der Ordnung O(1), weil es unabhängig von der Problemgröße ist.
Allerdings muß vorher die Suche stattfinden, die wiederum O(n) hat. Insgesamt also O(n). 2

2

Summe: 10

Bitte geben Sie für die beiden vorigen Aufgaben ein Nassi-Shneiderman-Diagramm an:

```
+-----------------------------------------------------------------------+
|                                                                       |
| Platz für N beschaffen                                                |
|                                                                       |
+-----------------------------------------------------------------------+
|                                                                       |
|       Liste leer (Anfang=NULL)?                                       |
|                 |                                                     |
+---- ja ---------+------------------ nein -----------------------------+
|                 |                                                     |
| N.next = NULL   | Vorgänger A suchen (hinter dem eingefügt wird)      |
|                 |                                                     |
+-----------------+-----------------------------------------------------+
|                 |                                                     |
| N.prev = NULL   |              A = NULL?                              |
|                 |                 |                                   |
+-----------------+----- ja --------+-------------- nein ---------------+
|                 |                 |                                   |
| Anfang = N      | N.next = Anfang |             A = Ende?             |
|                 |                 |                 |                 |
+-----------------+-----------------+----- ja --------+---- nein -------+
|                 |                 |                 |                 |
| Ende = N        | Anfang.prev = N | N.prev = A      | N.prev = A      |
|                 |                 |                 |                 |
+-----------------+-----------------+-----------------+-----------------+
|                 |                 |                 |                 |
|                 | N.prev = NULL   | A.next = N      | N.next = A.next |
|                 |                 |                 |                 |
|                 +-----------------+-----------------+-----------------+
|                 |                 |                 |                 |
|                 | Anfang = N      | N.next = NULL   | A.next.prev = N |
|                 |                 |                 |                 |
|                 +-----------------+-----------------+-----------------+
|                 |                 |                 |                 |
|                 |                 | Ende = N        | A.next = N      |
|                 |                 |                 |                 |
+-----------------+-----------------+-----------------+-----------------+
```

je nach vorhergehenden Antworten entsprechend bewertet

Summe: 8

Angenommen, Sie sollen ein Programm erstellen, mit dem unter anderem Artikel verwaltet werden sollen. Jeder Artikel besteht aus einer eindeutigen Nummer, einer Bezeichnung (ebenfalls eindeutig), sowie einigen anderen Werten (Einkaufspreis, Lagerbestand, etc.).

Die eindeutigen Artikelnummern werden nicht fortlaufend vergeben, so daß in ihrem Wertebereich auch große Lücken auftreten können.

Bei Programmstart sollen alle Artikel aus einer Datenbank oder Datei gelesen werden, und während des Programmlaufs im Arbeitsspeicher liegen. Es muß damit gerechnet werden, daß sich die Gesamtzahl der Artikel während des Programmlaufs stark ändern kann (aber immer genug Arbeitsspeicher zur Verfügung steht, notfalls virtueller).

Notwendige Operationen sind:

- Einfügen (alle bei Programmstart, dann nur noch gelegentlich)
- Suchen (sehr häufig)
- Ändern eines Artikels
- Löschen (sehr selten)

(a) Angenommen, die Suche soll nur anhand der eindeutigen Nummer erfolgen.

1. Welche Datenstruktur würden Sie für Darstellung der Artikel im Programm wählen?

   Eine Hashtabelle, die bei Überschreiten eines bestimmten Füllgrades vergößert wird.
   Die Hashtabelle wird über eine Hashfunktion anhand der Nummer indiziert, und enthält die Artikel als Einträge (bei Programmstart kann gleich die Tabelle in voller Länge angelegt werden; da nur gelegentliches Einfügen stattfindet, ist ein aufwendiges späteres Verlängern der Tabelle nicht oder nur selten nötig; alle anderen Operationen haben die Ordnung $O(1)$).
   Kollisionen können über lineares Sondieren aufgelöst werden; alternativ wäre auch offenes Hashing möglich.

Andere Lösungen möglich; je uneffektiver, desto weniger Punkte

Summe: 10

je nach vorhergehenden Antworten entsprechend bewertet

Summe: 3

ii. Bewerten Sie die Speichereffizienz Ihrer Lösung:

je nach Füllgrad der Tabelle ist ein Teil (10-30%?) ungenutzt; ansonsten skaliert der Speicherverbrauch linear mit der Anzahl der Einträge.

iii. Welche Operationen sind nötig, um einen Artikel zu suchen, welche Laufzeitordnung können Sie dafür angeben?

anhand der Nummer einen Hashwert bestimmen (O(1)), an dieser Stelle in der Hashtabelle den Artikel vermuten; falls hier der falsche steht je nach Kollisionsbehandlung linear sondieren oder (bei offenem Hashing) in Liste suchen; bei mäßigem Füllgrad insgesamt (O(1))

Summe: 4

iv. Welche Operationen sind nötig, um einen Artikel einzufügen, welche Laufzeitordnung können Sie dafür angeben?

Falls zulässiger Füllgrad überschritten wird: vergrößern;
in jedem Fall anhand der Nummer einen Hashwert bestimmen;
falls hier belegt: lineares Sondieren bis freier Platz gefunden ist; dort eintragen
Laufzeit bei mäßigem Füllgrad insgesamt (O(1))

je nach vorhergehenden Antworten entsprechend, z.B. bei offenem Hashing: nicht sondieren, sondern in Liste eintragen

Summe: 4

v. Welche Operationen sind nötig, um einen Artikel zu löschen, welche Laufzeitordnung können Sie dafür angeben?

anhand der Nummer einen Hashwert bestimmen;
falls hier das falsche Element steht: lineares Sondieren bis richtiges gefunden und dort als gelöscht markieren; wenn bis zum ersten freien und nicht gelöschten Eintrag nichts gefunden wird: Abbruch
(alternativ bei offenem Hashing: anhand des Hashwerts Liste finden, dort löschen; als gelöscht markieren ist überflüssig)
Laufzeit bei mäßigem Füllgrad insgesamt (O(1))

Summe: 4

(b) Angenommen, die Suche soll sowohl anhand der eindeutigen Nummer als auch anhand der Bezeichnung des Artikels möglich sein.

1. Welche Datenstruktur würden Sie jetzt für die Darstellung der Artikel im Programm wählen?

Andere Lösungen möglich; je uneffektiver, desto weniger Punkte

Die Artikel stehen in einem Feld mit exakt der nötigen Größe in beliebiger Reihenfolge. Gelöschte Artikel werden hier als gelöscht markiert. Beim Einfügen neuer Elemente wird das Feld verlängert (da dies nach dem anfänglichen Füllen nur selten vorkommt, lohnt es sich nicht, linear nach gelöschten Einträgen zu suchen, und diese wieder zu verwenden).
Für jedes Suchkriterium (also hier für die Nummer und für die Bezeichnung) wird je eine Hashtabelle gebaut, die einen Hashwert des Suchkriteriums auf die Position des Artikels im Feld abbildet. Jede dieser Hashtabellen soll beispielsweise mit linearer Sondierung arbeiten, und bei Bedarf vergrößert werden; alternativ auch hier offenes Hashing möglich.

Summe: 14

ii. Bewerten Sie die Speichereffizienz Ihrer Lösung:

je nach Füllgrad der Tabellen für die Suchkriterien ist darin ein Teil (10-30%?) ungenutzt; ansonsten skaliert der Speicherverbrauch dieser Tabellen linear mit der Anzahl der Einträge. Der so ungenutzte Platz ist nicht kritisch, weil diese Einträge nicht Platz für den gesamten Artikel bieten, sondern nur Tabellenpositionen.
Die eigentliche Tabelle mit den Artikeln ist anfangs ohne Verschnitt vollständig gefüllt; nur beim Löschen von Artikeln bleiben Lücken. Da dies selten vorkommt, ist das unerheblich.

je nach vorhergehenden Antworten entsprechend bewertet

Summe: 5

iii. Welche Operationen sind nötig, um einen Artikel zu suchen, welche Laufzeitordnung können Sie dafür angeben?

anhand der Nummer oder der Bezeichnung einen Hashwert bestimmen (O(1)), daraus aus der zugehörigen Hastabelle die Position des Artikels in der Artikeltabelle bestimmen; je nach Kollisionen sondieren; bei mäßigem Füllgrad insgesamt (O(1))

Summe: 4

iv. Welche Operationen sind nötig, um einen Artikel einzufügen, welche Laufzeitordnung können Sie dafür angeben?

Feld mit den Artikeln um eins verlängern, dort den Artikel eintragen
für beide Hashtabellen: falls zulässiger Füllgrad überschritten wird: vergrößern;
in jedem Fall anhand der Nummer und der Bezeichnung für die jeweils zugehörige Tabelle einen Hashwert bestimmen;
falls hier belegt: jeweils lineares Sondieren bis freier Platz gefunden ist;
dort den Index des neuen Elements im Artikelfeld eintragen (jeweils in beiden Tabellen)
Laufzeit bei mäßigem Füllgrad insgesamt O(1), falls nicht verlöngert werden muß; sonst O(n), weil die Hashtabellen neu aufgebaut werden müssen.

Summe: 4

v. Welche Operationen sind nötig, um einen Artikel zu löschen, welche Laufzeitordnung können Sie dafür angeben?

anhand der Nummer oder Bezeichnung einen Hashwert bestimmen;
falls in zugehöriger Tabelle hier das falsche Element steht: lineares Sondieren bis richtiges gefunden (wenn bis zum ersten freien und nicht gelöschten Eintrag nichts gefunden wird: Abbruch) und im Artikelfeld dort als gelöscht markieren sowie in beiden Hashtabellen ebenfalls als gelöscht markieren;
Laufzeit bei mäßigem Füllgrad insgesamt O(1)

Summe: 4

Wie wird in einem *Binären Suchbaum* nach einem Element gesucht?

(a) Falls Baum leer: nicht gefunden
(b) falls gesuchtes Element kleiner als Wurzel: rekursive Suche im linken Teilbaum, sonst im rechten.

Summe: 6

Wie wird in einem *Binären Suchbaum* ein Element gelöscht?

(a) Suche nach dem zu löschenden Element (siehe letzte Aufgabe) *1*
(b) Falls das Element keine Kinder hat: Bei Vorgänger im Baum den entsprechenden linken oder rechten Nachfolger löschen. *2*
(c) Falls das Element ein Kind hat: Bei Vorgänger im Baum den entsprechenden linken oder rechten Nachfolger auf das Kind setzen *3*
(d) Falls das Element zwei Kinder hat: entweder das rechte Kind anstelle des zu löschenden Elements bei dessen Vorgänger einhängen, und das linke Kind des zu löschenden Elements soweit links wie möglich unter das rechte Kind hängen, oder alternativ das linke Kind beim Vorgänger einhängen, und dafür darin so weit rechts wie möglich das bisherige rechte Kind einhängen. *4*

Summe: 10