Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Jörg Derungs

Institut für Computersysteme
Jürg Gutknecht
Raphael Güntensperger

1. Vordiplom D-INFK

Prüfung Informatik I + II

11. März 2004

Name, Vorname: _____

Stud.-Nummer: _____

Ich bestätige mit meiner Unterschrift, dass ich diese Prüfung unter regulären Bedingungen ablegen konnte und dass ich die untenstehenden Hinweise gelesen und verstanden habe.

Unterschrift: _____

Hinweise:

- Ausser einem Wörterbuch dürfen Sie keine Hilfsmittel verwenden.
- Bitte schreiben Sie Ihre StudentInnen-Nummer auf **jedes** Blatt.
- Melden Sie sich bitte **sofort**, wenn Sie sich während der Prüfung in irgendeiner Weise bei der Arbeit gestört fühlen.
- Bitte verwenden Sie für jede Aufgabe ein neues Blatt. Pro Aufgabe kann nur eine Lösung angegeben werden. Ungültige Lösungsversuche müssen klar durchgestrichen werden.
- Bitte schreiben Sie **lesbar**! Wir werden nur bewerten, was wir lesen können.

Viel Erfolg!

Stud.-Nummer: _____

Informatik I:

Aufgabe	1	2	3	4	**Σ**
Mögl. Punkte	8	10	8	10	36
Punkte					

Informatik II:

Aufgabe	5	6	7	8	**Σ**
Mögl. Punkte	10	7	10	9	36
Punkte					

8 P **AUFGABE 1** *(Informatik I)*:

In dieser Aufgabe geht es um die Programmierung eines Marsroboters. Wir nehmen an, dass die auszukundschaftende Fläche mit einem feinmaschigen Gitter überzogen ist, und dass sich der Roboter in jedem Zeitschritt horizontal, vertikal oder diagonal von einem Gitterpunkt zum nächsten bewegen kann. Die Aufgabe besteht nun in der Entwicklung eines Oberonprogrammes, welches den Roboter auf möglichst geradlinigem Weg von der aktuellen Position `(0, 0)` zum Felsbrocken bei `(a, b)` steuert, und zwar so, dass sich der Roboter stets möglichst nahe an der exakten Verbindungsgeraden befindet.
Vorgehen: Zur Vereinfachung sollen Sie annehmen, dass `(a, b)` im ersten Oktanten liegt, d. h. dass `a >= b` und `b >= 0` gilt. Dann muss der Roboter bei jedem Schritt von `x` zu `x + 1` im wesentlichen nur entscheiden, ob er `y` belassen oder um `1` erhöhen soll. Erstellen Sie zuerst eine Fassung des Programmes, in welcher die Entscheidung abstrakt programmiert ist. Verwenden Sie dabei eine vorgegebene Prozedur

```
PROCEDURE Move (dx, dy: INTEGER);
```

welche den Roboter anweist sich um `(dx, dy)` zu verschieben. Programmieren Sie danach die Entscheidung aus.

10 P **AUFGABE 2** *(Informatik I)*:

Gegeben sei eine lange, ungeordnete Liste von Namen

```
TYPE Name = ARRAY NameLen OF CHAR;
VAR names: ARRAY NofNames OF Name;
```

Konzipieren und erstellen Sie ein Oberonprogramm, welches die Namensliste `names` effizient nach einer vorgegebenen Menge

```
VAR samples: ARRAY NofSamples OF Name;
```

einiger Musternamen durchsucht, d. h. welches die Nummern (Indizes) derjenigen Namen der Liste ausgibt, welche gleich einem der Musternamen sind.

a) Erstellen Sie eine Prozedur

```
PROCEDURE Equal(i, j : INTEGER) : BOOLEAN;
```

welche genau dann `TRUE` zurückliefert, wenn die Namen `samples[i]` und `names[j]` identisch sind.

b) Erstellen Sie das Durchsuchsprogramm unter Verwendung von Hashing. Nehmen Sie dabei an, dass eine Prozedur

```
PROCEDURE HashCode (VAR name: ARRAY OF CHAR): INTEGER;
```

zur Berechnung des Hashcodes eines Namens bereits vorliegt. Nehmen Sie ausserdem an, dass innerhalb der Samplemenge keine Kollisionen auftreten.

8 P **AUFGABE 3** *(Informatik I)*:

Jetzt geht es um die Erstellung eines Oberonprogrammes zum Testen von (zusammenhängenden) binären Datenstrukturen im Hinblick auf die Baumeigenschaft. Genauer gesagt soll eine Oberonprozedur

```
PROCEDURE IsTree (root: Node): BOOLEAN;
```

erstellt werden, welche genau dann `TRUE` retourniert, wenn die bei `root` verwurzelte Datenstruktur ein Baum ist (Baumeigenschaft!), wobei der Typ `Node` wie folgt deklariert sei:

```
TYPE Node = POINTER TO RECORD left, right: Node END;
```

Zur Vereinfachung der Aufgabe sei es erlaubt, den Typ `Node` nach Bedarf zu erweitern.

10 P **AUFGABE 4** *(Informatik I)*:

Wir betrachten eine Menge von Gegenständen, deren Kaufpreis durch die Reihe

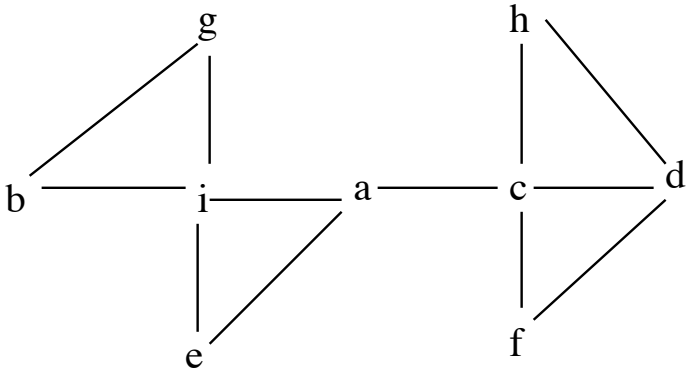
```
p: ARRAY NofObjects OF REAL;
```

gegeben sei. Die Aufgabe besteht nun in der Konzipierung und Entwicklung eines schnellen rekursiven Programmes, welches das Kaufpreis-Array in Gruppen von maximal 10 Elementen unterteilt und zwar so, dass sämtliche Preise einer jeden Gruppe nicht kleiner als sämtliche Preise der linken Nachbargruppe und nicht grösser als sämtliche Preise der rechten Nachbargruppe sind. Beschreiben Sie Ihre Idee und erstellen Sie dann ein entsprechendes Oberonprogramm, welches die Gruppeneinteilung vornimmt und die Gruppeninhalte ausdruckt.

AUFGABE 5 *(Informatik II)*:

In dieser Aufgabe sollen Sie **nur** die Ergebnisse angeben. Diese können Sie direkt bei den Aufgaben notieren. Sofern Sie die Notationen, Algorithmen und Datenstrukturen aus der Vorlesung Informatik II verwenden, sind Erklärungen oder Begründungen nicht notwendig. Falls Sie jedoch andere Methoden benutzen, müssen Sie diese **kurz** soweit erklären, dass Ihre Ergebnisse verständlich und nachvollziehbar sind.
Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

1 P a) Der folgende ungerichtete Graph wird mit Tiefensuche traversiert. Die Suche startet beim Knoten 'a'. Geben Sie eine Reihenfolge an, in der die Knoten erreicht werden können.



1 P b) Geben Sie für die untenstehenden Funktionen eine **Reihenfolge** an, so dass Folgendes gilt: Wenn Funktion f links von Funktion g steht, so gilt $f \in O(g)$.

Beispiel: Die drei Funktionen n^3, n^7, n^9 sind bereits in der entsprechenden Reihenfolge, da $n^3 \in O(n^7)$ und $n^7 \in O(n^9)$ gilt.

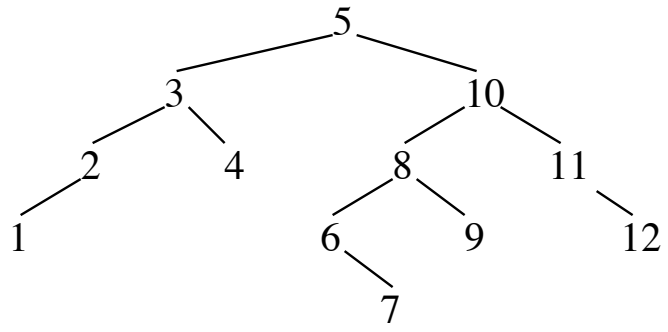
- $7n^2 + 3n$
- $\frac{n^3}{7} - 7n^2$
- 2^n
- $\frac{n^3}{\log n}$
- $n!$
- $n^2 \log n$
- $2^{\sqrt{n}}$

1 P c) Markieren Sie in der folgenden Liste genau die Aussagen, die korrekt sind:

- ☐ $n! \in O(n^n)$
- ☐ $5^{\log_3 n} \in O(5^{\log_5 n})$
- ☐ $\frac{n}{\log n} \in \Theta(n)$
- ☐ $n^2 + 3n \log n \in \Omega(n\sqrt{n})$

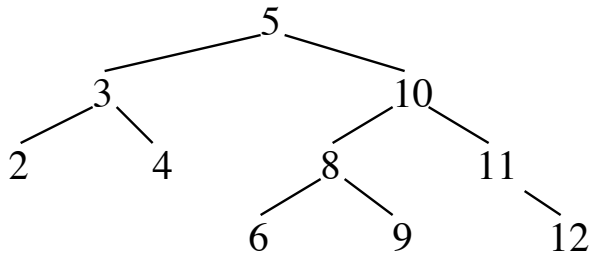
1 P

d) Geben Sie die Knoten des folgenden Suchbaumes in ihrer Postorder-Reihenfolge an.



1 P

e) Fügen Sie den Schlüssel 7 in den AVL-Baum ein und balancieren Sie ihn wieder.



1 P

f) In der folgenden Tabelle ist ein Heap in der üblichen impliziten Form gespeichert:

[2,7,20,15,10,25,22,16,21,11]

Wie sieht die Tabelle aus, nachdem das kleinste Element gelöscht und die Heap-Bedingung wieder hergestellt wurde?

1 P

g) Beim Einfügen eines Elements in eine Datenstruktur unterscheiden wir zwischen Vergleichen, die benötigt werden, um die richtige Stelle für das neue Element zu finden, und Schreiboperationen, die benötigt werden, um die Datenstruktur so abzuändern, dass sie das neue Element enthält.

Geben Sie die Anzahl Vergleiche und die Anzahl Schreiboperationen in Theta-Notation für das möglichst effiziente Einfügen eines Elements in folgende Datenstrukturen (welche bereits `n` Elemente enthalten) an:

	Vergleiche	Schreib-Operationen
sortierter Array		
sortierte lineare Liste		
AVL-Baum		
Heap		

1 P

h) Fügen Sie die Schlüssel $17, 8, 19, 10, 12$ mittels Double Hashing in die Hashtabelle ein.
Die zu verwendenden Hash-Funktionen sind $h(k) := k \mod 7$ und $h'(k) := (k \mod 5) + 1$.

0	1	2	3	4	5	6

1 P

i) Gegeben sind acht Schlüssel mit der relativen Anzahl der Zugriffe. Erstellen Sie mit Hilfe des Huffman-Algorithmus einen optimalen Codierungsbaum (Trie).

Schlüssel	a	b	c	d	e	f	g	h
Anzahl Zugriffe	5	8	9	5	8	3	7	3

1 P

j) Geben Sie an, wie die nach der Move-To-Front-Regel selbstanordnende Liste aussieht, nachdem auf die Elemente 'T', 'E', 'S', 'T' in dieser Reihenfolge zugegriffen wurde.

E $\longrightarrow$ T $\longrightarrow$ W $\longrightarrow$ A $\longrightarrow$ S

AUFGABE 6 *(Informatik II)*:

2 P a) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 9T(\frac{n}{3}) - 1 & n > 1 \\ \frac{1}{4} & n = 1 \end{cases}$$

Beweisen Sie mit vollständiger Induktion, dass $\frac{n^2+1}{8}$ eine geschlossene Formel für $T(n)$ ist.

Hinweis: Sie können annehmen, dass n eine Potenz von 3 ist.

3 P b) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 9T(\frac{n}{3}) + n & n > 1 \\ \frac{1}{2} & n = 1 \end{cases}$$

Geben Sie eine geschlossene (d.h. nicht-rekursive) Formel für $T(n)$ an und beweisen Sie diese mit vollständiger Induktion.

Hinweise: (1) Sie können annehmen, dass n eine Potenz von 3 ist. (2) $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

1 P c) Geben Sie eine möglichst genaue asymptotische Laufzeit in Abhängigkeit von n für folgenden Algorithmus an (in Oh-Notation):

```
FOR i := 0 TO n-1 DO
    FOR j := 0 TO n-1 DO
        a[i, j] := 1
    END
END;
FOR i := 0 TO n-1 DO
    a[i, i] := 0
END
```

1 P d) Geben Sie die asymptotische Laufzeit in Abhängigkeit von n für folgenden Algorithmus in Theta-Notation an:

```
i := 0; j := n;
WHILE j - i > 1 DO
    m := (i + j) DIV 2;
    IF a[m] <= x THEN
        i := m
    ELSE
        j := m
    END
END
```

AUFGABE 7 *(Informatik II)*:

Damit die Programmfenster auf einem Bildschirm richtig überlappend dargestellt werden können, wird neben der Grösse und Position auch für jedes Fenster eine Tiefe gespeichert. Je kleiner diese Zahl, desto näher beim Betrachter (weiter oben auf dem Stapel) ist das Fenster. So verdeckt z.B. ein Fenster mit Tiefe 3 eines mit Tiefe 7, falls sie sich überlappen. Keine zwei der insgesamt `N` Fenster haben die gleiche Tiefe. Der Punkt `(0,0)` befindet sich in der linken unteren Ecke des Bildschirms. Der Typ

```
Window: RECORD
    x, y: INTEGER;  (* linke untere Ecke *)
    w, h: INTEGER;  (* Breite und Hoehe *)
    depth: INTEGER  (* Tiefe *)
END;
```

beschreibt ein Fenster. Die Fenster sind in einem Array

```
windows: ARRAY N OF Window;
```

gespeichert.
Für die Teilaufgaben a) und b) erstrecken sich alle Fenster vom oberen bis zum unteren Bildschirmrand, d.h. für jedes Window gilt: `y` ist `0` und `h` ist die Höhe des Bildschirms.
Hinweis: Sie dürfen die Datentypen `AVLTree`, `Heap`, `List`, `SortedList`, `Stack` und `Queue` verwenden, ohne sie zu implementieren.

2 P a) Beschreiben Sie eine Datenstruktur, deren Speicheraufwand unabhängig von der Grösse des Bildschirms ist, die es erlaubt, möglichst effizient das oberste Fenster für eine gegebene Cursorposition zu bestimmen.

Geben Sie den asymptotischen Speicheraufwand für die Datenstruktur an, und bestimmen Sie den asymptotischen Aufwand zur Bestimmung des obersten Fensters für einen gegebenen Bildschirmpunkt.

Beachten Sie, dass `y` und `h` für diese Teilaufgabe nicht benötigt werden!

4 P b) Schreiben Sie eine Prozedur in Pseudocode, welche möglichst effizient aus dem Array `windows` die oben beschriebene Datenstruktur erzeugt. Geben Sie die asymptotische Laufzeit Ihrer Prozedur an.

Beachten Sie, dass `y` und `h` für diese Teilaufgabe nicht benötigt werden!

2 P c) Erweitern Sie Ihre Datenstruktur, so dass überlappende Fenster mit verschiedenen y-Koordinaten und Höhen gespeichert werden können.

Geben Sie den asymptotischen Speicheraufwand für die Datenstruktur an, und bestimmen Sie den asymptotischen Aufwand zur Bestimmung des obersten Fensters für einen gegebenen Bildschirmpunkt.

2 P d) Beschreiben Sie in Worten, wie die Prozedur von Teilaufgabe b) erweitert werden muss, um die Fenster aus `windows` in Ihrer Datenstruktur zu speichern. Geben Sie die asymptotische Laufzeit der erweiterten Prozedur an.

AUFGABE 8 *(Informatik II)*:

Für das Wochenende nach der letzten Prüfung planen Sie eine dreitägige Skitour, um den Kopf durchzulüften.
Sie planen diese Tour sehr gründlich und berechnen, dass Sie unterwegs mindestens `K` Kilokalorien benötigen, damit Sie nicht verhungern. Natürlich wollen Sie nur so viel Gewicht tragen wie nötig. Deshalb stellen Sie eine Liste aller Lebensmittel zusammen, die Sie zuhause haben, um zu entscheiden, welche davon Sie mitnehmen.
Die Lebensmittel sind in einem Array gespeichert:

```
food: ARRAY N OF RECORD
        cal: INTEGER;       (* Kalorien *)
        weight: INTEGER     (* Gewicht *)
      END;
```

3 P a) Schreiben Sie einen rekursiven Algorithmus, der bestimmt, wieviel Gewicht Sie mitnehmen müssen, um auf mindestens `K` Kilokalorien zu kommen.

Geben Sie die asymptotische Laufzeit Ihres Algorithmus in Abhängigkeit von `N` an.

4 P b) Entwerfen Sie einen Algorithmus nach dem Muster der dynamischen Programmierung, der bestimmt, wieviel Gewicht Sie mitnehmen müssen, um auf mindestens `K` Kilokalorien zu kommen.

Geben Sie die asymptotische Laufzeit Ihres Algorithmus in Abhängigkeit von `N` an.

2 P c) Beschreiben Sie in Worten, wie aus der Lösung gemäss der dynamischen Programmierung die entsprechenden Lebensmittel durch Rückverfolgen gefunden werden können.