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
Michel Gatto

Prüfung Datenstrukturen und Algorithmen, D-INFK

Datenstrukturen & Algorithmen

16. März 2006

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** mit blauer oder schwarzer Tinte. Wir werden nur bewerten, was wir lesen können.
- Die Prüfung dauert 120 Minuten.

Viel Erfolg!

Stud.-Nummer: _____

Aufgabe	1	2	3	4	5	**Σ**
Mögl. Punkte	9	7	9	10	10	45
Punkte						

AUFGABE 1:

Hinweise:

1) In dieser Aufgabe sollen Sie **nur** die Ergebnisse angeben. Diese können Sie direkt bei den Aufgaben notieren.

2) Sofern Sie die Notationen, Algorithmen und Datenstrukturen aus der Vorlesung "Datenstrukturen & Algorithmen" 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.

3) Als Ordnung verwenden wir für Buchstaben die alphabetische Reihenfolge, für Zahlen die aufsteigende Anordnung gemäss ihrer Grösse.

1 P a) Führen Sie auf dem gegebenen Array zwei Iterationen des Algorithmus *Sortieren durch Einfügen* aus. Das zu sortierende Array ist durch vorherige Iterationen bereits bis zum Doppelstrich sortiert worden.

1	2	5	12 ‖	17	3	4	18	22	13	6	42

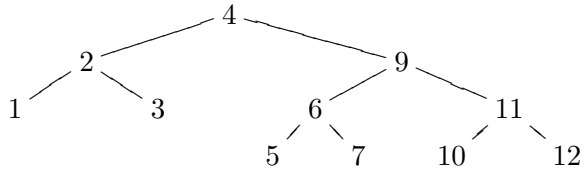
1 P b) Geben Sie an, wieviele Schlüsselvergleiche benötigt werden, um in der gegebenen selbstanordnenden Liste mit der Transpose-Regel auf die Elemente 'H','A','L','L','O' in dieser Reihenfolge zuzugreifen.

A ⟶ L ⟶ G ⟶ O ⟶ R ⟶ I ⟶ T ⟶ H ⟶ M ⟶ U ⟶ S

1 P c) Zeichnen Sie den binären Suchbaum, dessen Post-Order-Traversierung die Folge $1, 5, 6, 13, 12, 16, 18, 17, 21, 24, 23, 19, 14$ ergibt.

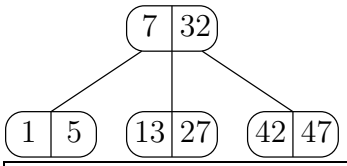
1 P

d) Fügen Sie den Schlüssel 8 in den AVL-Baum ein und, falls nötig, balancieren Sie ihn wieder.



1 P

e) Fügen Sie den Schlüssel 17 in den unten stehenden B-Baum der Ordnung 3 ein und löschen Sie danach den Schlüssel 13.



Nach Einfügen von 17:	Nach Löschen von 13:

1 P

f) 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	1	9	7	2	8	21	15	24

1 P

g) Fügen Sie die Schlüssel $29, 36, 24, 21, 7, 33, 43$ in dieser Reihenfolge mittels Double Hashing in die Hashtabelle ein.

Die zu verwendenden Hash-Funktionen sind $h(k) = (k \mod 7)$ und $h'(k) = 1 + (k \mod 5)$.

0	1	2	3	4	5	6

1 P

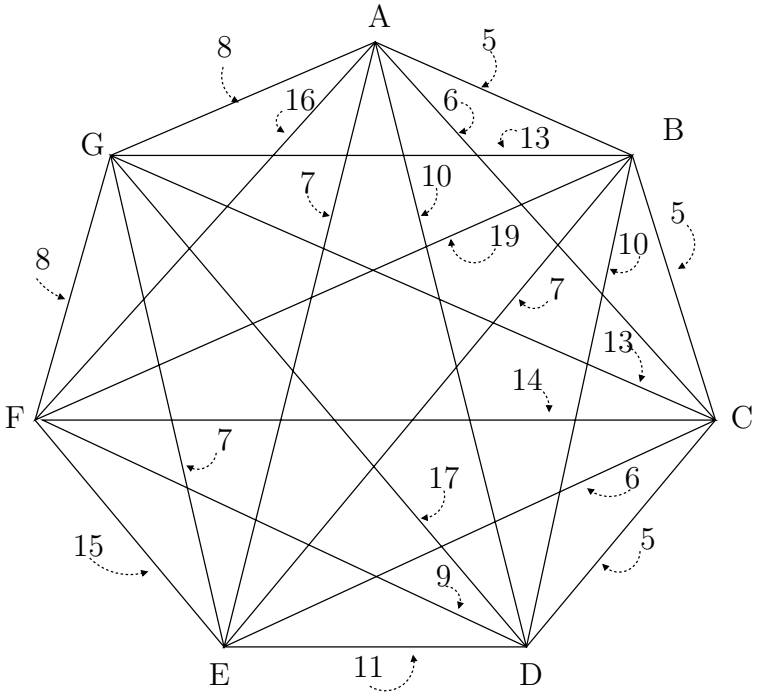
h) In der folgenden Tabelle ist ein Heap in der üblichen impliziten Form gespeichert:

[6, 8, 9, 9, 12, 17, 21, 22, 19, 42, 17, 18]

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

1 P

i) Geben Sie im untenstehenden Graphen *mittels des Approximationsalgorithmus, der auf dem minimalen Spannbaum basiert*, eine Rundreise an, deren Länge höchstens das 2-fache der Länge der optimalen Rundreise ist. Sie müssen diese Schranke nicht beweisen. Beginnen Sie die optimale Rundreise beim Knoten A.



AUFGABE 2:

1 P a) 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.

- $3^{2 \cdot \log_3 n}$
- n^n
- $\Pi_{i=1}^{n}(i \cdot n)$
- $n!$
- $\log n$
- $\frac{n}{\sqrt{n}}$
- n^3

3 P b) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 8T(\frac{n}{2}) + 14 & n > 1 \\ 5 & n = 1 \end{cases}$$

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

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

1 P

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

```
from i:= n until i < 1 loop
   i := i // 2
   from j := 1 until j > i loop
      j := j + 1
   end
end
```

1 P

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

```
from i:= 1 until i > n loop
   if i = n then
      from j:=1 until j > n loop
         j:= j + 1;
      end
   end
   i:= i + 1
end
```

1 P

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

```
from i:= 1 until i > n loop
   if i < n then
      from j:=i until j < 1 loop
         j:= j // 2;
      end
   end
   i:= i + 1
end
```

AUFGABE 3:
Jede Person einer gegebenen Familie ist durch ein Paar (g, w) charakterisiert, wobei g ihre Körpergrösse und w ihr Gewicht ist. Für jede auftretende Körpergrösse möchte man den Gewichts-Median bestimmen, also das mittlere Gewicht in der sortierten Reihenfolge (der Gewichte für diese Körpergrösse).
Zur Erinnerung: Für m gegebene ganze Zahlen ist der Median diejenige Zahl, die an Position $\lceil \frac{m}{2} \rceil$ steht, wenn man die Zahlen in aufsteigend sortierter Reihenfolge hinschreibt. Ein Beispiel: die Zahl 2 ist der Median der Zahlen $1, 2, 3, 4$. Ferner ist 4 der Median der Zahlen $4, 1, 2, 10, 3, 12, 5$.
Unter den Personen $(170, 70), (185, 90), (185, 92), (170, 70), (165, 68), (170, 85)$ haben die Personen $(170, 70)$, $(185, 90)$ und $(165, 68)$ die jeweiligen Gewichts-Mediane.
Die Daten der Personen sind in einem unsortierten Array `pers: ARRAY [ PERSON ]` gespeichert.

```
class PERSON
feature -- Access
  g: INTEGER
        -- g Groesse der Person in cm
  w: INTEGER
        -- w Gewicht der Person in Kg
end -- class PERSON
```

Hinweise:

- Beachten Sie, dass mehrere Personen dasselbe Gewicht und/oder die selbe Grösse haben können.
- Sie dürfen die in Eiffel vorhandenen Datenstrukturen `STACK [S]` , `QUEUE [Q]` , sowie einen effizienten Algorithmus `sort( a: ARRAY [ COMPARABLE ] )` zum Sortieren (in aufsteigender Reihenfolge) benutzen, ohne diese weiter zu implementieren oder zu definieren.
- Sie dürfen Features überschreiben und die Klasse `PERSON` beliebig erweitern.

3 P a) Beschreiben Sie kurz in Worten einen Ansatz für einen Algorithmus, der für eine gegebene Familie von Personen für jede vorhandene Körpergrösse den Gewichts-Median berechnet.

5 P b) Schreiben Sie in Eiffel eine möglichst effiziente Funktion, welche für jede vorhandene Körpergrösse den Gewichts-Median nach dem in Teilaufgabe a) beschriebenen Algorithmus berechnet, und das Resultat als `ARRAY [ PERSON ]` zurückgibt.

1 P c) Geben Sie die Laufzeit Ihrer Funktion in Abhängigkeit von `n = pers.count` an.

AUFGABE 4:
Ein Reisebüro bietet eine Vielfalt an Destinationen für Urlaubsreisen an, mit Flug ab Zürich. Es stellt sich heraus, dass viele Kunden die Destination so wählen, dass sie mindestens k Kilometer von Zürich entfernt ist, und unter allen solchen Reisezielen die Reise so billig wie möglich ist.
Es entsteht deshalb der Bedarf, die Informationen über die Reisen in einer Datenstruktur zu organisieren, so dass die oft vorkommenden Abfragen nach dem minimalen Preis einer Destination, die mindestens eine Distanz von k Kilometern ab Zürich haben, möglichst effizient erfolgen können.
Das Reisebüro erhält die Daten aller Reisen am Anfang jedes Jahrs, und diese Daten bleiben für das ganze Jahr unverändert. Die verschiedenen Destinationen sind als Paar (d, c) gegeben, wobei d die Distanz in Kilometern ab Zürich ist, und c die Gesamtkosten der Reise für die Destination.

```
class DESTINATION
feature -- Access
  d: INTEGER
        -- d Distanz der Destination in km
  c: INTEGER
        -- c Kosten der Reise mit dieser Destination
end -- class DESTINATION
```

Hinweise: Sie können annehmen, dass keine zwei Destinationen dieselbe Distanz von Zürich haben, und dass es pro Destination genau ein Reiseangebot gibt (die d-Werte sind also eindeutig).
Sie dürfen die Klasse `DESTINATION` beliebig erweitern.

3 P a) Beschreiben Sie eine Datenstruktur für eine fixe Menge von Destinationen, mit der man so effizient wie möglich
`min_price(k : INTEGER ): INTEGER` berechnen kann, als den Preis der billigsten Reise mit Mindestdistanz k ab Zürich.

Hinweise:
1) Die Daten können geeignet vorverarbeitet werden, um anschliessende Abfragen zu unterstützen.
2) Sie können bekannte Datenstrukturen so erweitern, dass sie die gegebenen Anforderungen erfüllen.

2 P b) Beschreiben Sie, wie die Operation `min_price(k : INTEGER ) : INTEGER` mit Ihrer in a) definierten Datenstruktur erfolgt, und implementieren Sie die Operation in einem Eiffel-ähnlichen Pseudocode. Geben Sie die Laufzeit von `min_price` an.

5 P c) In einer schwierigeren Problemvariante kann sich das Angebot an Destinationen ändern. Deshalb möchte man zusätzlich ermöglichen, dass neue Destinationen eingefügt und vorhandene Destinationen gelöscht werden können. Beschreiben Sie eine Datenstruktur, die es erlaubt, so effizient wie möglich diese zusätzlichen Operationen und die Operation
`min_price(k : INTEGER ) : INTEGER`
durchzuführen. Beschreiben Sie möglichst genau, wie diese Operationen mit der von Ihnen entworfenen Datenstruktur erfolgen.

Für die Destinationen gelten dieselben Restriktionen wie für den statischen Fall; insbesondere ist jede Distanz in der Menge der vorhandenen Destinationen eindeutig.

AUFGABE 5:

Zwei Numismatiker beschliessen, gemeinsam eine Sammlung von M Münzen zu kaufen. Der ganzzahlige Kaufpreis von K Franken setzt sich zusammen aus der Summe der ganzzahligen Werte aller Münzen: die i-te Münze kostet k_i Franken ($i \in \{1, \cdots, M\}$), und $K = \sum_{i=1}^{M} k_i$.

Die zwei Numismatiker beteiligen sich unterschiedlich beim Kauf: der Erste trägt mit F_1 Franken zum Kauf bei, der Zweite bezahlt den Rest von $F_2 = K - F_1$ Franken, wobei $F_1 < F_2$.

Nun möchten die Numismatiker die Münzensammlung so in zwei Mengen teilen, dass jeder eine Menge bekommt, die genau dem Wert des bezahlten Betrags entspricht.

2 P a) Erstellen Sie ein rekursives Programm in Pseudocode, welches eine Aufteilung der Münzen auf beide Numismatiker berechnet, falls es sie gibt, und das sonst die entsprechende Meldung liefert. Brechen Sie die Rekursion ab, sobald es offensichtlich ist, dass keine Lösung erzeugt werden kann.

Geben Sie die Laufzeit des rekursiven Algorithmus an.

5 P b) Entwerfen Sie einen Algorithmus für obiges Problem nach dem Muster der dynamischen Programmierung.

Geben Sie die Laufzeit Ihres Algorithmus an.

1 P c) Beschreiben Sie in Worten, wie durch Rückverfolgung in der Lösungstabelle gemäss b) die richtigen Münzen für jede Menge gefunden werden können.

Geben Sie die Laufzeit für die Rückverfolgung an.

2 P d) Bei teureren Sammlungen beteiligen sich nicht zwei Numismatiker am Kauf, sondern eine beliebige natürliche Zahl k von Numismatikern, $k \geq 2$, mit Kaufpreis-Beteiligung $F_i, i \in \{1, \cdots, k\}$, wobei $\sum_{i=1}^{k} F_i = K$. Entwerfen Sie einen Algorithmus nach dem Muster der dynamischen Programmierung für die Münzenaufteilung in diesen Fall.

Geben Sie die Laufzeit Ihres Algorithmus an.