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

5. Oktober 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. **Keine Angst!** Wir rechnen nicht damit, dass irgendjemanden alles löst!

Viel Erfolg!

Stud.-Nummer: _____

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

AUFGABE 1:

Hinweise:

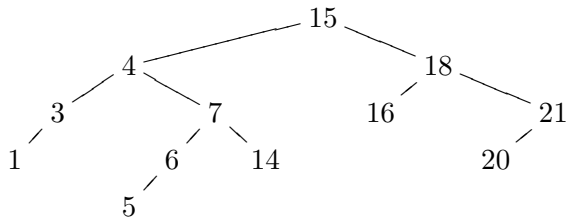
1. In dieser Aufgabe sollen Sie **nur** die Ergebnisse angeben. Diese können Sie direkt bei den Aufgaben notieren.
2. Sofern Sie die Notationen, Algorithmen und Datenstrukturen aus der Vorlesung "Datenstrukturen & Algorithmen" verwenden, sind Erklärungen oder Begründungen nicht notwendig. Falls Sie jedoch 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) Im unten stehenden Array wurden, nachdem der Heap aufgebaut wurde, bereits 3 Iterationen von Heapsort durchgeführt. Jede Iteration besteht aus folgenden Aktionen: das gemäss Heapsort nächste Element an die endgültige Stelle des Array setzen, und die Heap-Bedingung wieder herstellen. Führen Sie eine weitere Iteration von Heapsort durch.

42	41	37	29	27	22	5	10	12	14	17	47	51	59

1 P b) Zeichnen Sie den binären Suchbaum, dessen Pre-Order-Traversierung die Folge $1, 2, 7, 5, 4, 6, 20, 8, 10, 9, 14, 12, 15$ ergibt.

1 P c) Löschen Sie den Schlüssel 20 aus dem AVL-Baum und rebalancieren Sie ihn wieder.



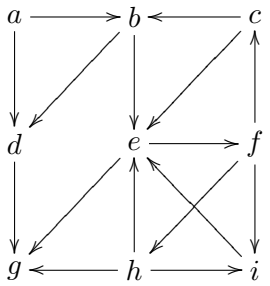
1 P d) Fügen Sie die Schlüssel $31, 33, 34, 49, 19$ in dieser Reihenfolge mittels Double Hashing in die Hashtabelle ein.

Die zu verwendenden Hash-Funktionen sind $h(k) = (k \mod 11)$ und $h'(k) = 1 + (k \mod 9)$.

11	45		47		16				9	
0	1	2	3	4	5	6	7	8	9	10

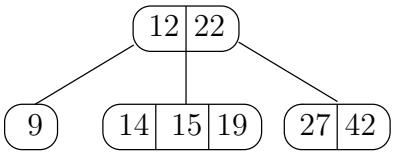
1 P

e) Der folgende gerichtete 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

f) Fügen Sie den Schlüssel 16 in den unten stehenden B-Baum der Ordnung 4 ein und löschen Sie danach den Schlüssel 9, jeweils mit den zugehörigen Strukturänderungen.



Nach Einfügen von 16:	Nach Löschen von 9:

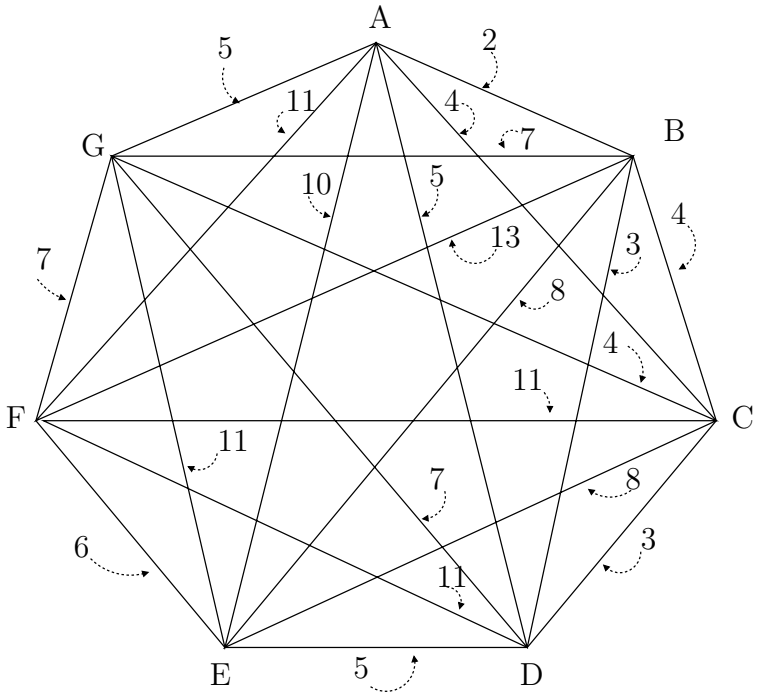
1 P

g) 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	4	10	2	4	5	3	42

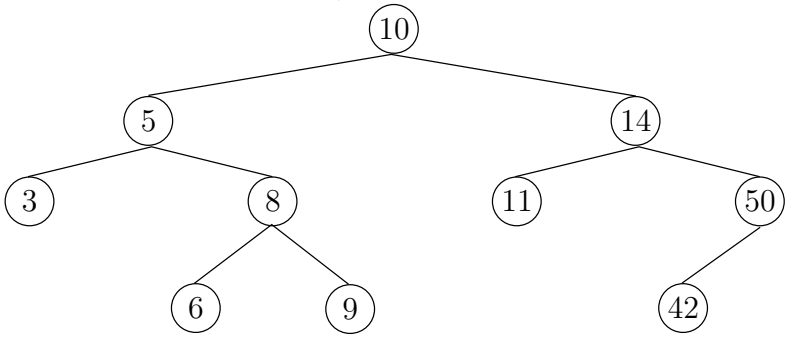
1 P

h) Geben Sie im untenstehenden gewichteten 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. Die Kantengewichte erfüllen die Dreiecksungleichung. Sie müssen diese Schranke nicht beweisen. Beginnen Sie die Rundreise beim Knoten A.



1 P

i) Zeichnen Sie den Splay-Tree, der aus dem unten gezeigten entsteht, nachdem man auf den Knoten 9 zugegriffen hat (und die zugehörigen Splay-Operationen ausgeführt wurden).



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.

- $\sum_{i=1}^{n} \frac{1}{i}$
- $\sum_{i=0}^{n} i$
- $2^{\log_3(n)}$
- $\Pi_{i=1}^{n} \frac{i}{\sqrt{i}}$
- $\frac{n^3}{\sqrt{n}}$
- n^n
- $n \cdot n!$

3 P b) Gegeben ist die folgende Rekursionsgleichung:

$$T(n) := \begin{cases} 4T(\frac{n}{2}) + n + 3 & n > 1 \\ 5 & n = 1 \end{cases}$$

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
   from j := 1 until j > i loop
      j := j + 1
   end
   i := i // 5
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 < (4 * n // 5) then
      from j:=i 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
   from j:=i until j < 2 loop
      j:= j // 5+1;
      from k:=i until k < 5 loop
         k := k // 3
      end
   end
   i:= i + 1
end
```

AUFGABE 3:

Die Zeitschrift YAHG Hardware Guide möchte eine Studie über die jeweilige Laufdauer der Akkus von Notebooks durchführen. Als *Laufdauer* eines Akkus versteht man die Zeit, die vergeht, bis das eingeschaltete Notebook seinen anfangs vollständig geladenen Akku komplett entladen hat. Da der Prozessortakt einen besonderen Einfluss auf die Laufdauer des Akkus hat, sammelt YAHG für eine Menge von verschiedenen Notebooks die Daten über den Prozessortakt (in MHz) und die Laufdauer des Akkus (in Minuten). Auch Notebooks mit gleichgetakteten Prozessoren können unterschiedliche Laufdauer aufweisen, da auch andere (in dieser Studie nicht berücksichtigte) Faktoren die Laufdauer beeinflussen.

Nun möchte YAHG für jeden gegebenen Prozessortakt bestimmen, was die längste und was die kürzeste Laufdauer des Akkus ist.

Die Daten sind in einem Array `nb: ARRAY [ NOTEBOOK ]` gespeichert. Die Klasse `NOTEBOOK` ist unten definiert.

3 P a) Beschreiben Sie kurz in Worten einen Ansatz für einen Algorithmus, der aus einem gegebenen Array `nb` von `NOTEBOOK` möglichst effizient die kleinste und die grösste Laufdauer pro Prozessortakt bestimmt.

Hinweis: Beachten Sie, dass mehrere Notebooks dieselbe Taktrate `clock` oder dieselbe Akku-Dauer `battery` haben können.

5 P b) Schreiben Sie in Eiffel eine möglichst effiziente Funktion, welche die verlangten Werte nach dem in Teilaufgabe a) beschriebenen Algorithmus bestimmt, und das Resultat in einem geeigneten Array zurückgibt.

Hinweise:

- Sie dürfen die in Eiffel vorhandenen Datenstrukturen `STACK[Q]`, `QUEUE[K]`, 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 `NOTEBOOK` beliebig erweitern.

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

```
class NOTEBOOK
feature -- Access
  clock: INTEGER
        -- Taktrate des Prozessors in MegaHertz
  battery: INTEGER
        -- Laufdauer des Akkus in Minuten
end -- class NOTEBOOK
```

AUFGABE 4:
MeteoSchweiz hat für eine Periode von n aufeinander folgenden Tagen jeweils die Maximaltemperatur und die Minimaltemperatur gemessen. Nehmen wir an, an jedem Tag sei die Maximaltemperatur später als die Minimaltemperatur erreicht worden. Für zwei (an beliebigen Tagen) gemessene Temperaturwerte sprechen wir dann von einem *Temperaturanstieg*, wenn die später gemessene Temperatur höher ist als die früher gemessene. Jetzt interessieren wir uns für den grössten in der Periode aufgetretenen Temperaturanstieg, allerdings unter einer Bedingung: Die beiden Messungen dürfen nicht mehr als k Tage auseinander liegen, müssen also innerhalb eines $(k+1)$-Tage-Fensters erfolgen.

Betrachten wir zum Beispiel $k = 2$, und eine Periode von 5 aufeinander folgenden Tagen mit jeweiliger Messung von (Minimaltemperatur, Maximaltemperatur): $T_1 = (1, 8)$, $T_2 = (4, 7)$, $T_3 = (3, 10)$, $T_4 = (5, 15)$, $T_5 = (8, 14)$. Der maximale Temperaturanstieg kommt zum Beispiel im 3-Tage-Fenster von Tag 3 bis Tag 5 mit $15 - 3 = 12$ zu Stande.

Die Temperaturen sind als ganzzahlige Werte gegeben. Die Minimaltemperaturen sind als Array `min: ARRAY [INTEGER]`, die Maximaltemperaturen als `max: ARRAY[INTEGER]` gegeben, wobei die Maximal- beziehungsweise Minimaltemperaturen des i-ten Tages in `min.item(i)` bzw. `max.item(i)` gespeichert sind. Beide Arrays haben die gleiche Grösse, und enthalten $n =$`min.count` Messwerte.

3 P a) Beschreiben Sie kurz in Worten einen Algorithmus, der möglichst effizient aus den gegebenen Arrays `min` und `max` von Messwerten den maximalen Temperaturanstieg, der innerhalb eines beliebigen $(k + 1)$-Tage-Fensters erfolgt, berechnet.

1 P b) Geben Sie die Laufzeit Ihres Algorithmus aus a) in Abhängigkeit von n und k an.

3 P c) Schreiben Sie in Pseudocode eine Funktion, die, ausgehend von den Messwerten in `min` und `max`, den maximalen Temperaturanstieg innerhalb eines beliebigen $(k+1)$-Tage-Fensters nach dem in Teilaufgabe a) angegebenen Verfahren berechnet.

3 P d) Die Meteorologen möchten nun wissen, wie gross der maximale Temperatur**unterschied** (egal ob Anstieg oder Abfall, also das Abnehmen der Temperatur von einer Messung zu einer späteren) war, der in dieser Periode von n Tagen innerhalb eines beliebigen $(k + 1)$-Tage-Fensters vorgekommen ist.

Beschreiben Sie kurz in Worten einen Algorithmus, und geben Sie die Laufzeit Ihres Algorithmus in Abhängigkeit von k und n an.

AUFGABE 5:

In einem Naturgarten möchte man einen geraden Weg durch eine grosse Wiese bauen. Der Weg muss vom einen Ende zum anderen gehen, muss also eine exakt festgelegte Gesamtlänge L (ganzzahlig, in cm) haben. Er soll aus Granitplatten gelegt werden, von denen eine grosse Menge M zum Kauf bereitsteht. Alle verfügbaren Granitplatten haben die gleiche Breite, gerade so wie für den Weg gewünscht, haben aber ganz unterschiedliche Einzellängen ℓ_i (ganzzahlig, in cm). Jede Granitplatte hat ihren Preis p_i, unabhängig von ihrer Einzellänge, aufgrund ihrer Schönheit. Wir möchten nun einige der Granitplatten kaufen, so dass exakt die gewünschte Gesamtlänge des Wegs getroffen wird, dass wir aber insgesamt für die gekauften Platten möglichst wenig bezahlen müssen.

Beispiel: Unser Weg ist 1000 cm lang, und es stehen Platten mit (Länge, Preis) von (600,50), (550,60), (350,75), (250,120), (150,140), und (400,320) zur Verfügung. Wir kaufen die erste, vierte und fünfte Platte in dieser Aufzählung.

2 P a) Erstellen Sie ein rekursives Programm in Pseudocode, welches prüft, ob es überhaupt eine Auswahl von Platten aus der Menge M gibt, mit der die Gesamtlänge ℓ exakt getroffen wird.

Geben Sie die Laufzeit des rekursiven Algorithmus an.

5 P b) Entwerfen Sie einen Algorithmus nach dem Muster der dynamischen Programmierung, der den tiefstmöglichen Preis für den Weg (durch Auswahl von Platten aus M) berechnet.

Geben Sie die Laufzeit Ihres Algorithmus an.

1 P c) Beschreiben Sie in Worten, wie durch Rückverfolgung in der Lösungstabelle gemäss b) die Platten für den Weg gefunden werden können.

Geben Sie die Laufzeit für die Rückverfolgung an.

2 P d) Entwerfen Sie einen Algorithmus nach dem Muster der dynamischen Programmierung, der eine preiswerteste Plattenauswahl berechnet für den Fall, dass man nicht nur einen Weg legen will, sondern 2 parallele Wege derselben Länge.

Geben Sie die Laufzeit Ihres Algorithmus an.