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
Beat Gfeller

Prüfung
Datenstrukturen und Algorithmen
D-INFK

20. August 2008

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 irgendjemand alles löst! Sie brauchen bei weitem nicht alle Punkte, um die Bestnote zu erreichen.

Viel Erfolg!

Stud.-Nummer: _____

Aufgabe	1	2	3	4	5	**Σ**
Mögl. Punkte	9	7	9	9	6	40
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) Zeichnen Sie die Liste, welche entsteht, wenn die Move-to-Front-Regel benutzt wird, und auf folgende Elemente zugegriffen wird (in dieser Reihenfolge): Q, F, X, Q, Z, V, F.

Liste vor den Zugriffen: $\mathrm{X} \rightarrow \mathrm{V} \rightarrow \mathrm{Z} \rightarrow \mathrm{F} \rightarrow \mathrm{B} \rightarrow \mathrm{Q}$

Liste nach den Zugriffen:

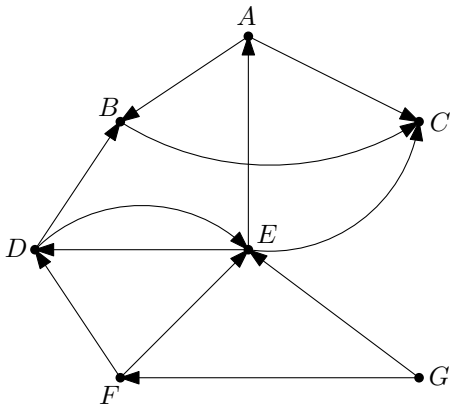
1 P b) Führen Sie auf dem folgenden Array die ersten zwei Schritte des Sortierverfahrens *Natürliches 2-Wege-Mergesort* durch.

7	6	8	17	2	20	5	14	3	1

nach Schritt 1:

nach Schritt 2:

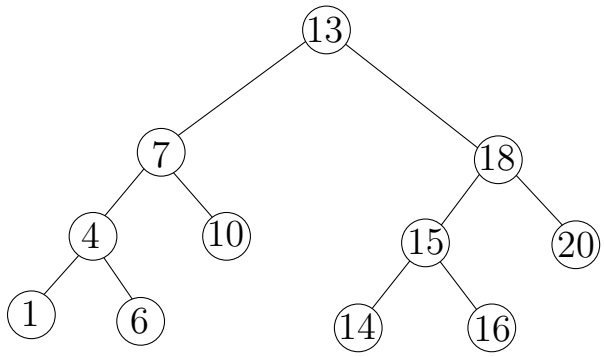
1 P c) Der folgende gerichtete Graph wird mit Tiefensuche traversiert. Die Suche startet beim Knoten G. Geben Sie eine Reihenfolge an, in der die Knoten erreicht werden können.



1 P d) Fügen Sie die Schlüssel $19, 14, 30, 10, 12, 99$ in dieser Reihenfolge mittels Offenem Hashing in die folgende Hashtabelle ein (die bereits ein paar Schlüssel enthält), und benutzen Sie dabei Double Hashing. Die zu verwendende Hash-Funktion ist $h(k) = k \mod 11$, und für das Sondieren die Hashfunktion $h'(k) = 1 + (k \mod 9)$.

11			3		16				20	0
0	1	2	3	4	5	6	7	8	9	10

1 P e) Fügen Sie in den untenstehenden AVL-Baum den Schlüssel 17 ein, und löschen Sie *danach* den Schlüssel 13 daraus.



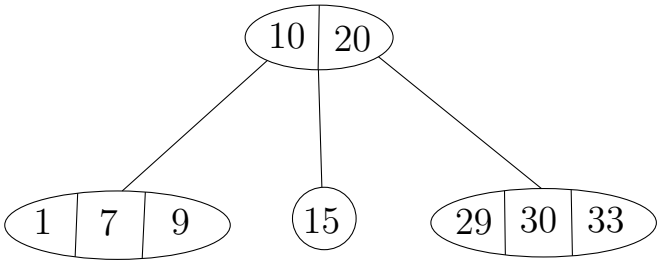
1 P f) Zeichnen Sie den binären Suchbaum, dessen Postorder-Traversierung die Folge 9, 5, 12, 14, 19, 18, 27, 21, 20, 10 ergibt.

1 P g) Das untenstehende Array enthält die Elemente eines Min-Heaps in der üblichen Form gespeichert. Wie sieht das Array aus, nachdem das Minimum entfernt wurde und die Heap-Bedingung wieder hergestellt wurde?

3	8	4	10	16	6	7	27	14	20
1	2	3	4	5	6	7	8	9	10

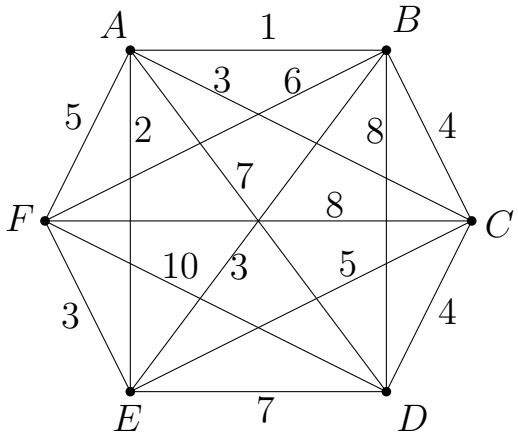
1 P

h) Gegeben sei der folgende B-Baum der Ordnung 4. Fügen Sie in diesen zuerst den Schlüssel 2 ein, und löschen Sie danach den Schlüssel 10 aus dem B-Baum.



1 P

i) 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 einer optimalen Rundreise ist. Die Kantengewichte erfüllen die Dreiecksungleichung. Sie müssen die Approximations-Schranke nicht beweisen. Beginnen Sie die 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.

- $2^{\log_2 n}$
- n^2
- $\frac{\log^5 n}{\sqrt{n}}$
- $\sqrt{n^3}$
- $\log(n^8)$
- $\prod_{i=1}^{n} i^2$

3 P b) Gegeben ist die folgende Rekursionsgleichung:

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

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

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

```
from
  i := 1
  j := 1
until i > n
loop
  from
  until j > i + 5
  loop
    j := j + 1
  end
  i := 2*i
end
```

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

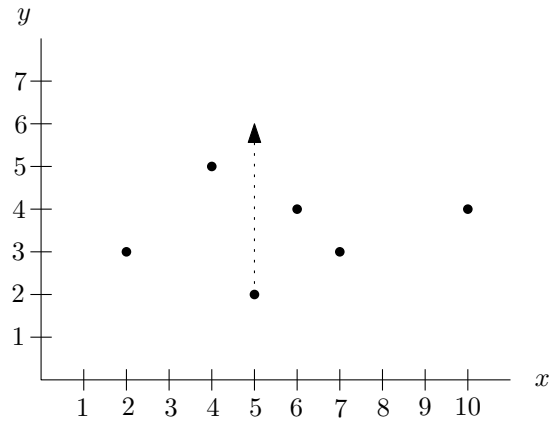
```
from
  i := 1
  s := 1
until i > n
loop
  if i > s then
    from
      j := 1
    until j > s
    loop
      j := j + 1
    end
    s := 2 * s
  end
  i := i + 1
end
```

Aufgabe 3:

In dieser Aufgabe soll eine Datenstruktur entworfen werden. Das zu lösende Problem ist das folgende: Gegeben sind n Punkte $P_1, P_2, \ldots, P_n$ mit ganzzahligen Koordinaten $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. Die x_i sind alle verschieden und unveränderlich. Die Punkte können jedoch auf der y-Achse verschoben werden, d.h. die Werte y_i können durch Operationen verändert werden.
Die zu entwerfende Datenstruktur soll die folgenden Operationen effizient unterstützen:

- `RangeMax(`$x_{\min}$`, `$x_{\max}$`)`:
  Liefert die grösste y-Koordinate eines Punktes im Intervall $[x_{\min}, x_{\max}]$ zurück.
- `Update(`x_i`, `y_{new}`)`:
  Setzt die y-Koordinate von Punkt (x_i, y_i) auf den Wert y_{new}.

Im untenstehenden Beispiel würde also `RangeMax(2, 7)` zuerst den Wert 5 liefern. Wenn danach jedoch die Operation `Update(5, 6)` ausgeführt wird, liefert `RangeMax(2, 7)` den Wert 6.



3 P a) Entwerfen und beschreiben Sie eine Datenstruktur für das obige Problem, sowie eine nützliche Invariante, welche für die in Ihrer Datenstruktur gespeicherten Werte gelten soll (unabhängig davon, wieviele `Update`-Operationen schon ausgeführt wurden).

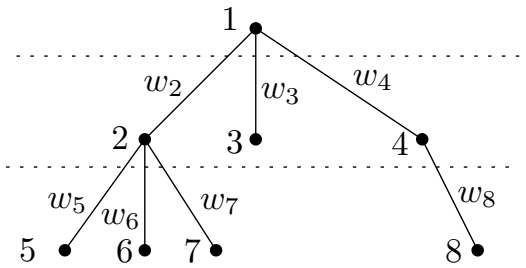
3 P b) Beschreiben Sie, wie die in a) entworfene Datenstruktur benutzt werden kann, um die Operation `RangeMax(`$x_{\min}$`, `$x_{\max}$`)` effizient zu implementieren. Geben Sie zudem die Laufzeit dieser Operation im schlechtesten Fall an.

3 P c) Beschreiben Sie, wie die Datenstruktur aktualisiert wird, wenn die Operation `Update(`x_i`, `y_{new}`)` ausgeführt wird.

Aufgabe 4:

In dieser Aufgabe geht es um Bäume, deren Kanten Gewichte haben. Für einen gegebenen Baum soll eine Teilmenge M aller Kanten des Baumes ausgewählt werden, so dass keine zwei ausgewählten Kanten einen gemeinsamen Knoten als Endpunkt haben (d.h. keine zwei Kanten dürfen sich berühren). Sie erinnern sich: Eine solche Teilmenge nennt man ein *Matching*. Das Ziel ist nun, ein Matching zu finden, bei dem die Summe der Gewichte aller ausgewählten Kanten grösstmöglich ist.
Die Knoten sind wie bei einem Heap in Ebenen gegliedert, und von oben nach unten und von links nach rechts nummeriert (siehe Bild). Die Nummer einer Kante sei die Nummer ihres unteren Endpunkts im Baum. Das Gewicht der Kante i heisse w_i.

Der Baum ist wie folgt in einem Array T gegeben: T enthält n Elemente, nummeriert von 1 bis n. Element i steht an Position i und beschreibt das Intervall der Nummern der Kinder des Knotens i. Daher ist $T[i].min$ das Kind von Knoten i mit der kleinsten Nummer, und $T[i].max$ das mit der grössten. Hat Knoten i keine Kinder, so ist $T[i].min = 0, T[i].max = -1$.
Das untenstehende Bild zeigt einen Beispielbaum mit der zugehörigen Repräsentation als Array. Die Gewichte aller Kanten sind durch ein zweites Array W gegeben, welches an Position i das Gewicht der Kante mit Nummer i enthält, d.h. $W[i] = w_i$.



	1	2	3	4	5	6	7	8
T:	2,4	5,7	0,-1	8,8	0,-1	0,-1	0,-1	0,-1

2 P a) Entwerfen Sie einen möglichst effizienten Algorithmus, der für einen gegebenen Baum berechnet, wie hoch das maximal mögliche Gewicht eines Matchings auf diesem Baum ist. Beschreiben Sie den Algorithmus kurz in Worten, und geben Sie dessen Laufzeit im schlechtesten Fall an.

6 P b) Schreiben Sie ein feature in Eiffel, welches den in a) beschriebenen Algorithmus realisiert. Anstelle von Eiffel können Sie auch C++ oder Java verwenden.

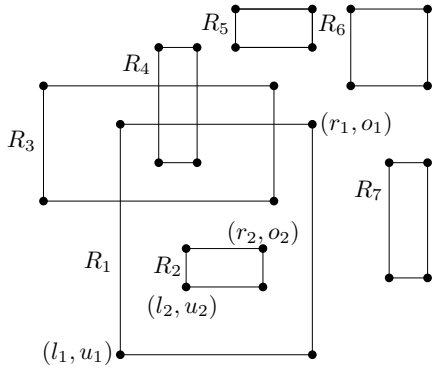
Die Arrays T und W seien definiert als `T: ARRAY[INTERVALL]` und `W: ARRAY[INTEGER]`, wobei die Klasse `INTERVALL` wie folgt definiert ist:

```
class INTERVALL
feature
    min: INTEGER
    max: INTEGER
end -- class INTERVALL
```

1 P c) Beschreiben Sie, wie man mit Hilfe der Lösung aus Teilaufgabe a) ein Matching finden kann, welches das maximal mögliche Gewicht hat.

Aufgabe 5:

Gegeben sei eine Menge von n orthogonalen Rechtecken $R_1, \ldots, R_n$. Jedes Rechteck R_i ist gegeben durch die Koordinaten (l_i, u_i) des unteren linken Eckpunkts sowie die Koordinaten (r_i, o_i) des oberen rechten Eckpunkts. Wir sagen, dass Rechteck R_i das Rechteck R_j ganz enthält, wenn alle der folgenden Bedingungen gelten: $l_i < l_j$, $u_i < u_j$, $r_i > r_j$, $o_i > o_j$. In der untenstehenden Abbildung ist beispielsweise das Rechteck R_2 im Reckteck R_1 ganz enthalten.



Nun möchte man feststellen, ob es in der gegebenen Menge ein Rechteck gibt, welches ganz in einem anderen enthalten ist.
Hinweis: Natürlich könnte man einfach jedes Paar von Rechtecken prüfen, und so die Antwort in $O(n^2)$ Zeit bestimmen. Wir erwarten hier daher eine effizientere Lösung.

5 P a) Entwerfen Sie einen möglichst effizienten Algorithmus, welcher für eine gegebene Menge von Rechtecken eine Nachricht darüber ausgibt, ob es ein Rechteck gibt, das ganz in einem anderen Rechteck enthalten ist. Beschreiben Sie Ihren Algorithmus in Worten und/oder Pseudocode.

1 P b) Welche Laufzeit hat Ihr Verfahren im schlechtesten Fall?