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

6. Oktober 2005

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	9	10	44
Punkte						

AUFGABE 1:

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 "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.
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 einen Aufteilungsschritt des Sortieralgorithmus Quicksort durch. Benutzen Sie als Pivot das am rechten Ende stehende Element im Array.

59	89	12	67	1	4	17	85	95	13	72	6	42

1 P

b) Geben Sie die kürzeste Folge von Zugriffen an, die, ausgehend von der Liste

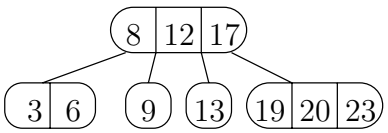
$$A \longrightarrow L \longrightarrow G \longrightarrow O \longrightarrow R \longrightarrow I \longrightarrow T \longrightarrow H \longrightarrow M \longrightarrow U \longrightarrow S$$

mittels der Move-To-Front-Regel folgende Liste ergibt:

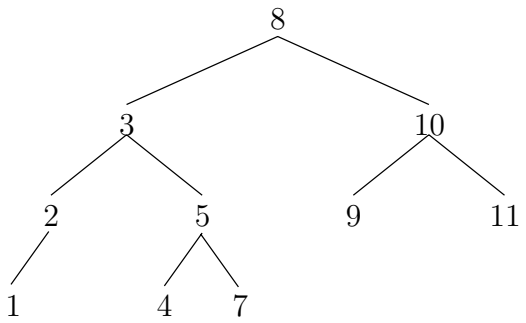
$$L \longrightarrow O \longrightarrow G \longrightarrow A \longrightarrow R \longrightarrow I \longrightarrow T \longrightarrow H \longrightarrow M \longrightarrow U \longrightarrow S$$

1 P

c) Fügen Sie den Schlüssel 25 in den untenstehenden B-Baum der Ordnung 4 ein und löschen Sie danach den Schlüssel 13.

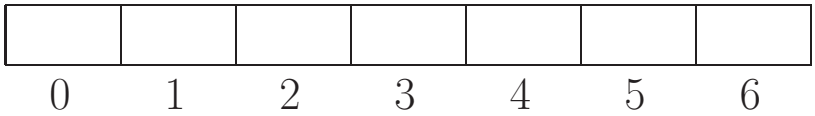


1 P d) Fügen Sie den Schlüssel 6 in den AVL-Baum ein und rebalancieren Sie ihn wieder.



1 P e) Fügen Sie die Schlüssel $45, 36, 10, 16, 43, 3, 48$ mittels einem offenen Hashverfahren mit quadratischem Sondieren in die Hashtabelle ein.

Die zu verwendende Hash-Funkion ist $h(k) = k \mod 7$.

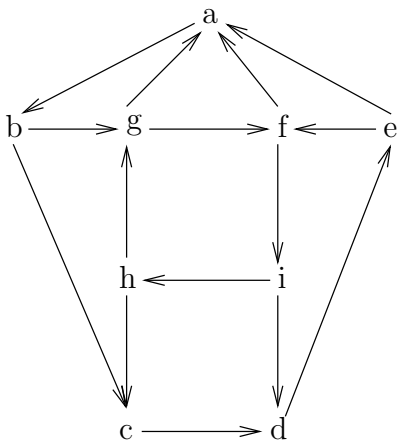


1 P f) Gegeben sind acht Schlüssel mit der relativen Anzahl der Zugriffe. Erstellen Sie mit Hilfe des Huffman-Algorithmus einen optimalen Codierungsbaum (Trie).

Schlüssel	a	b	c	d	e	f	g	h
Anzahl Zugriffe	1	3	5	1	8	2	3	22

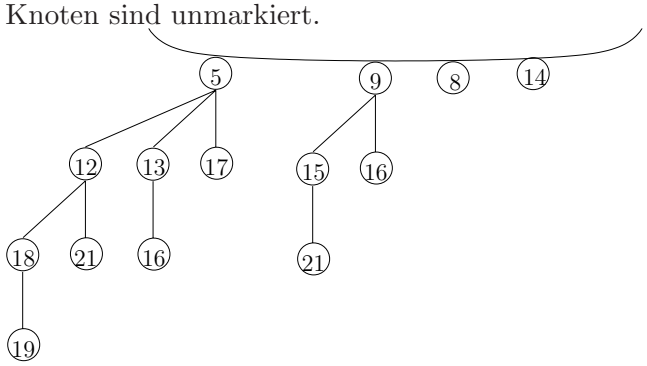
1 P

g) Der folgende gerichtete Graph wird mit Breitensuche traversiert. Die Suche startet beim Knoten `a`. Geben Sie eine Reihenfolge an, in der die Knoten erreicht werden können.



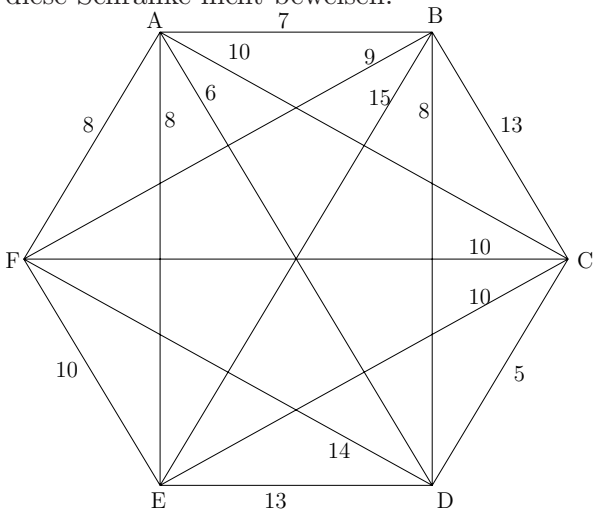
1 P

h) Führen Sie auf dem unten stehenden Fibonacci-Heap die Operation Delete-Min aus. Alle Knoten sind unmarkiert.



1 P

i) Geben Sie im untenstehenden Graphen *mittels dem Algorithmus von Christofides* eine Rundreise an, deren Länge höchstens das $\frac{3}{2}$-fache der Länge der Optimalen Tour ist. Sie müssen diese Schranke nicht beweisen.



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.

- 10^{20}
- $\frac{1}{n!}$
- $n^3 + 170n^2 + 2$
- $\frac{n^3}{\log n}$
- $\frac{n^3}{\sqrt{n}}$
- $\frac{1}{2}\sum_{i=0}^{n} i$

3 P b) Gegeben ist die folgende Rekursionsgleichung:

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

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 3 ist. (2) $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

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

AUFGABE 3:

Gegeben ist eine Menge von paarweise verschiedenen Punkten (x, y) in der Ebene. Ein Punkt P_1 *dominiert* einen anderen Punkt P_2, wenn keine Koordinate von P_1 grösser als die entsprechende Koordinate von P_2 und eine davon echt kleiner ist. Formal dominiert P_1 den Punkt P_2, wenn entweder $x_1 < x_2$ und $y_1 \leq y_2$ oder wenn $x_1 \leq x_2$ und $y_1 < y_2$.
Die Dominanz ist in Abbildung 1 gezeigt. Die Menge der nicht dominierten Punkte nennt man *Pareto-Front*. Diese ist in Abbildung 2 dargestellt. In dieser Aufgabe soll die Pareto-Front einer Menge von Punkten berechnet werden.

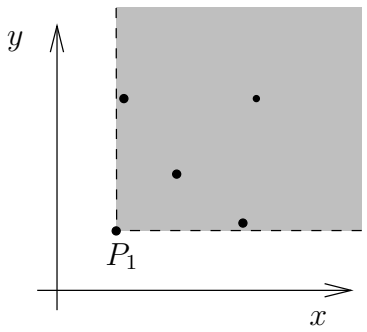


Abbildung 1: P_1 dominiert die Punkte in der schattierten Region.

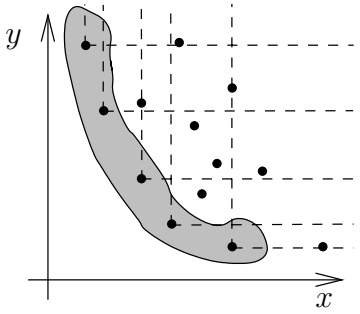


Abbildung 2: Die Pareto-Front der gezeichneten Punkte ist die Menge der Punkte in der schattierten Region.

Die Punkte sind in einem Array `pc: ARRAY [ POINT ]` gespeichert.

```
class POINT
feature -- Access
  x: INTEGER
        -- x Koordinate des Punktes
  y: INTEGER
        -- y Koordinate des Punktes
end -- class POINT
```

Hinweise:

- Beachten Sie, dass mehrere Punkte dieselbe x oder y Koordinate haben können.
- 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 `POINT` beliebig erweitern.

3 P a) Beschreiben Sie kurz in Worten einen Ansatz für einen Algorithmus, der aus einer gegebenen Menge von Punkten möglichst effizient die Pareto-Front berechnet.

5 P b) Schreiben Sie in Eiffel eine möglichst effiziente Funktion, welche die Pareto-Front nach dem in Teilaufgabe a) beschriebenen Algorithmus berechnet, und das Resultat als `ARRAY [ POINT ]` zurückgibt.

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

AUFGABE 4:

Wir möchten mit einer gegebenen Menge von Vorlesungen $V = \{v_1, \cdots, v_n\}$ einen Stundenplan erzeugen, so dass so viele Vorlesungen wie möglich stattfinden können. Für diese Vorlesungen steht ein einziger Hörsaal zur Verfügung. Jede Vorlesung hat eine Startzeit s_i und eine Endzeit e_i, ab der der Hörsaal besetzt ist bzw. wieder zur Verfügung steht, falls die Vorlesung tatsächlich stattfindet, wobei $0 \leq s_i < e_i < \infty$. Eine im Stundenplan geplante Vorlesung v_i besetzt somit den Hörsaal im Zeitintervall $[s_i, e_i)$ (auf der rechten Seite offen). Keine zwei Vorlesungen besetzen das genau gleiche Intervall.
Zwei Vorlesungen v_i, v_j nennt man *kompatibel*, falls sich ihre Intervalle $[s_i, e_i)$ und $[s_j, e_j)$ nicht überlappen.
Wir möchten den Stundenplan so erstellen, dass möglichst viele zueinander kompatible Vorlesungen stattfinden.

Beispiel: Bei einer Vorlesungsmenge $V = \{v_1, v_2, v_3\}$, mit $s_1 = 0, e_1 = 3; s_2 = 1, e_2 = 4;\ s_3 = 3$, $e_3 = 6$, bilden die Vorlesungen $\{v_1, v_3\}$ die maximale Menge von kompatiblen Vorlesungen.

Hinweise:

- Man beachte, dass zwei Vorlesungen v_k und v_ℓ auch dann noch zueinander kompatibel sind, wenn die Endzeit der einen gleich der Startzeit der anderen ist, d.h. falls $e_k = s_\ell$.
- Mehrere Vorlesungen können die gleiche Startzeit oder die gleiche Endzeit haben.

3 P a) Beschreiben Sie kurz in Worten einen Ansatz für einen Algorithmus, der aus einer gegebenen Menge von Vorlesungen V möglichst effizient eine maximale Menge von zueinander kompatiblen Vorlesungen berechnet.

5 P b) Schreiben Sie in Pseudocode eine möglichst effiziente Funktion, die, ausgehend von der Menge V, eine maximale Menge von zueinander kompatiblen Vorlesungen nach dem in Teilaufgabe a) angegebenen Algorithmus berechnet.

1 P c) Geben Sie die Laufzeit der in b) implementierten Funktion an.

AUFGABE 5:

In einem Naturgarten möchte man zwei gerade und zu einander parallele Wege durch eine grosse Wiese bauen. Beide Wege sollen dieselbe Breite haben und mit Steinplatten bedeckt werden. Die zu Verfügung stehenden N Platten haben alle bereits die korrekte Breite, unterscheiden sich allerdings in ihrer Länge. Die Wege sollen so gebaut werden, dass alle Platten benutzt werden, und dass beide Wege möglichst gleich lang sind. Man möchte also den Unterschied in der Länge zwischen den zwei Wegen minimieren.

Die ganzzahligen Längen der Platten sind in einem Array `lengths : ARRAY [ INTEGER ]` gespeichert.

2 P a) Erstellen Sie ein rekursives Programm in Pseudocode, welches die Zusammensetzung der Platten für jeden Weg berechnet, so, dass die Längendifferenz der Wege minimal ist. Brechen Sie die Rekursion ab, sobald es offensichtlich ist, dass keine optimale Lösung mehr erzeugt werden kann.

Geben Sie die Laufzeit des rekursiven Algorithmus an.

5 P b) Entwerfen Sie einen Algorithmus nach dem Muster der dynamischen Programmierung, der die Gesamtlänge der Platten für jeden Weg bei minimalem Längenunterschied berechnet.

Geben Sie die Laufzeit Ihres Algorithmus an.

1 P c) Beschreiben Sie in Worten, wie durch Rückverfolgung in der Lösungstabelle gemäss b) die richtigen Platten für jeden 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 den minimalen Längenunterschied berechnet für den Fall, dass man nicht zwei parallele Wege bauen will, sondern k parallele Wege, für festes, gegebenes k. Auch hier muss der maximale Längenunterschied minimal sein, und es müssen alle Platten benutzt werden.

Geben Sie die Laufzeit Ihres Algorithmus an.