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
Jörg Derungs

# Basisprüfung D-INFK

# **Datenstrukturen & Algorithmen**

10. März 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.

**Viel Erfolg!**

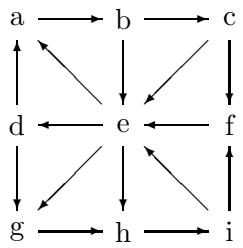
Stud.-Nummer: _____

| Aufgabe | 1 | 2 | 3 | 4 | 5 | **Σ** |
|---|---|---|---|---|---|---|
| Mögl. Punkte | 9 | 7 | 12 | 10 | 13 | 51 |
| 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) 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** b) In der folgenden Tabelle ist ein Heap in der üblichen impliziten Form gespeichert:

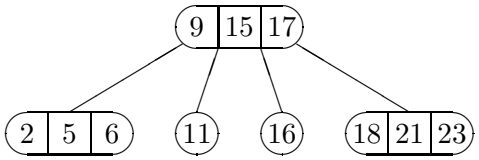
[3, 10, 7, 12, 15, 9, 8, 15, 14, 20]

Wie sieht die Tabelle aus, nachdem zuerst das Element 4 und danach das Element 2 eingefügt und die Heap-Bedingung wieder hergestellt wurde?

**1 P** c) Zeichnen Sie den binären Suchbaum, dessen Postorder-Traversierung die Folge
1, 4, 5, 3, 2, 6, 9, 11, 10, 8, 7, 13, 12
ergibt.

**1 P**

d) Löschen Sie den Schlüssel 11 aus dem untenstehenden B-Baum der Ordnung 4 und fügen Sie danach den Schlüssel 20 ein.




**1 P**

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

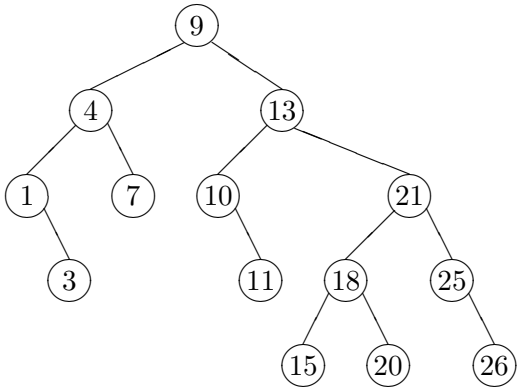
| Schlüssel | a | b | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|---|
| Anzahl Zugriffe | 9 | 5 | 19 | 2 | 3 | 6 | 2 | 12 |

**1 P**

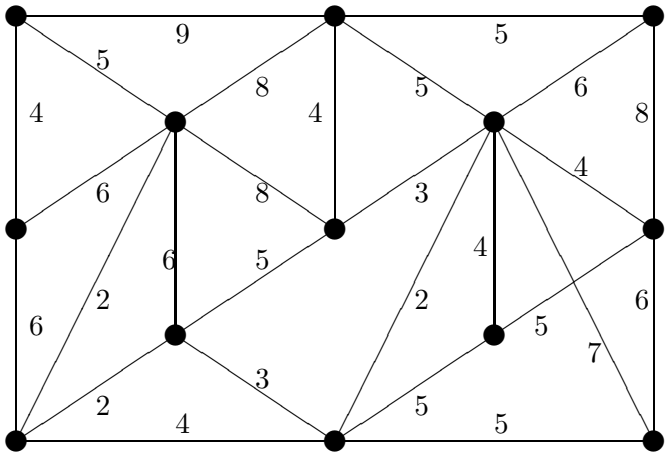
f) Fügen Sie die Schlüssel $67, 18, 35$ in dieser Reihenfolge mittels Double Hashing in die Hashtabelle ein.
Die zu verwendenden Hash-Funktionen sind $h(k) = k \mod 13$ und $h'(k) = (k \mod 11) + 1$.

| | 53 | 119 | 29 | | | 38 | | | 74 | | | 64 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

**1 P** g) Fügen Sie den Schlüssel 17 in den AVL-Baum ein und balancieren Sie ihn wieder.




**1 P** h) Markieren Sie im untenstehenden gewichteten Graphen die Kanten eines minimalen Spannbaums.




**1 P** i) Geben Sie an, wie die nach der Move-To-Front-Regel selbstanordnende Liste aussieht, nachdem auf die Elemente 'H', 'A', 'L', 'L', 'O' in dieser Reihenfolge zugegriffen wurde.

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

**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.

- $\sum_{i=1}^{n} i$
- $2^n$
- $n \cdot \log\left(n^5\right)$
- $\sqrt{n}$
- $\prod_{i=1}^{n} i$
- $\frac{n+1}{n-1}$

**3 P** b) Gegeben ist die folgende Rekursionsgleichung:

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

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

*Hinweise:*

i) Sie können annehmen, dass $n$ eine Potenz von 2 ist.

ii) $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

**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
      from k := i + j until k > n loop
         if i * i + j * j = k * k then
            t := t + 1
         end
         k := k + 1
      end
      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 until i > n loop
   from j := i until j = 0 or else a.item (j) <= a.item (i) loop
      j := j // 2
   end
   if j > 0 then
      t := t + a.item (j)
   end
   i := i + 1
end
```

*Hinweis*: `a.item` hat konstante Laufzeit.

**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
      t := t + 3
      if i = j then
         from k := 1 until k > n loop
            t := t + 1
            k := k + 1
         end
      end
      j := j + 1
   end
   i := i + 1
end
```

**AUFGABE 3:**

Das Ziel der Alchemisten ist es, den Stein der Weisen herzustellen. Dazu braucht der geübte Alchemist verschiedene Ingredienzen, wovon er einige bereits im Lager hat und andere aus wieder anderen Ingredienzen herstellen muss.

*Beispiel:*
Zur Herstellung des Steins der Weisen braucht der Alchemist ein Drachenauge, Sonnenstrahlen, Katzengold und Blei. Um Katzengold herzustellen benötigt er Katzenhaare, Mäusekot und Kupfer, für das Drachenauge muss er Katzengold, Ameisenschweiss und Edelweiss verarbeiten. Sonnenstrahlen und Blei hat er auf Lager.

Jede Substanz ist zusammen mit der Liste der Ingredienzen, aus denen diese Substanz hergestellt werden kann, in einem Objekt der Klasse

```
class SUBSTANZ
feature -- Access
   name: STRING
         -- Name der Substanz
   ingredienzen: ARRAY [SUBSTANZ]
         -- Liste der zur Herstellung noetigen Ingredienzen; kann leer sein
   vorhanden: BOOLEAN
         -- Ist die Substanz auf Lager?
end -- class SUBSTANZ
```

gespeichert. Sie dürfen diese Klasse nach Ihren Bedürfnissen erweitern.
Das Rezept für den Stein der Weisen ist in `stein: SUBSTANZ` gespeichert. Zusätzlich finden Sie alle Substanzen in `substanzen: ARRAY [SUBSTANZ]`.

*Hinweise:*

i) Von jeder Substanz im Vorrat ist beliebig viel vorhanden.

ii) Für jede Substanz gibt es nur ein einziges Rezept, um diese herzustellen.

iii) Sie dürfen die Klassen `STACK [G]` und `QUEUE [G]` benutzen, ohne sie zu definieren.

**2 P** a) Beschreiben Sie kurz in Worten einen Ansatz für einen Algorithmus, der berechnet, ob der Stein der Weisen mit den vorrätigen Ingredienzen hergestellt werden kann.

**6 P** b) Schreiben Sie in Eiffel eine möglichst effiziente Funktion, welche berechnet, ob der Stein der Weisen mit den vorrätigen Ingredienzen hergestellt werden kann, und das Resultat als BOOLEAN zurückgibt.

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

bitte wenden

**3 P**

d) Beschreiben Sie in Worten, wie Ihr Algorithmus erweitert werden muss, wenn die Vorräte beschränkt sind und die Rezepte Mengenangaben enthalten.

```
class SUBSTANZ
feature -- Access
   name: STRING
         -- Name der Substanz
   ingredienzen: ARRAY [TUPLE [SUBSTANZ, INTEGER]]
         -- Liste der zur Herstellung noetigen Ingredienzen mit
         -- Mengenangabe; kann leer sein
   vorhanden: INTEGER
         -- Anzahl vorraetiger Einheiten dieser Substanz
end -- class SUBSTANZ
```

**AUFGABE 4:**

In einem fensterbasierten System wie MS Windows oder gnome passiert es oft, dass sich rechteckige Fenster öffnen. Dabei kommt es vor, dass einige Fenster durch die neu geöffneten Fenster partiell oder auch vollständig überdeckt werden. Zweck dieser Aufgabe ist es, eine Menge von Fenstern in zwei Mengen aufzuteilen, nämlich die der Fenster, die man vollständig sieht (die also von keinem anderen Fenster überdeckt werden), sowie die der Fenster, die man nicht vollständig sieht.
Jedes Fenster wird als Rechteck beschrieben:

```
class WINDOW
feature -- Access
   x: INTEGER
         -- x Koordinate der oberen linken Ecke des Fenster
   y: INTEGER
         -- y Koordinate der oberen linken Ecke des Fenster
   b: INTEGER
         -- Breite des Fensters in x-Richtung
   h: INTEGER
         -- Hoehe des Fensters in y-Richtung
   z: INTEGER
         -- Position in der Reihenfolge des Oeffnens
end -- class WINDOW
```

Die Zahl `z` gibt die Position in der Reihenfolge des Öffnens der Fenster an: ist z.B. für zwei Fenster `i` und `j`, die sich überschneiden, `j.z > i.z`, so liegt `j` oben und verdeckt (einen Teil von) `i`. `z` kann also als dritte Koordinate angesehen werden, und der Benutzer schaut aus `z` = $\infty$ in Richtung `z` = `0`.
Die Fenster sind in `windows: ARRAY[WINDOW]` gespeichert, und müssen in die zwei Arrays `unverdeckt: ARRAY [WINDOW]` und `ueberdeckt: ARRAY [WINDOW]` aufgeteilt werden (wobei `unverdeckt` die vollständig sichtbaren Fenster enthält, und `ueberdeckt` die nur teilweise oder gar nicht sichtbaren Fenster).

*Hinweise:*

i) Um Fallunterscheidungen zu vermeiden können Sie annehmen, dass keine zwei Fenster dieselbe x- oder y-Koordinate haben.

ii) Die Fenster sind im Array nach aufsteigender x-Koordinate sortiert.

iii) Für diese Aufgabe dürfen Sie die in der Vorlesung behandelten Datenstrukturen sowie die Prozedur `sort (a: ARRAY [ANY])` verwenden, ohne sie weiter zu definieren.

**3 P** a) Beschreiben Sie in zwei bis drei kurzen Sätzen einen Ansatz für einen Algorithmus, der eine gegebene Menge von Fenster in vollständig sichtbare und nicht vollständig sichtbare Fenster unterteilt.

**6 P** b) Formulieren Sie in Pseudocode einen möglichst effizienten Algorithmus, der die Aufteilung durchführt.

**1 P** c) Geben Sie die Laufzeit Ihres Algorithmus in Abhängigkeit von der Anzahl der Fenster an und begründen Sie diese.

**AUFGABE 5:**

Herr Neureich will alle Besucher mit seiner Büchersammlung beeindrucken. Deshalb hat er über der Garderobe ein Bücherbrett angebracht, auf dem er einen möglichst hohen Wert an Büchern ausstellen will.

Die Breite des Bücherbretts in Millimetern ist `B: INTEGER`.

Die Bücher sind in einem Array `buecher: ARRAY [TUPLE [INTEGER, REAL]]` gespeichert, wobei `buecher.item (i).integer_item (1)` die Breite des i-ten Buches in Millimeter angibt und `buecher.item (i).real_item (2)` seinen Wert.

**3 P** a) Erstellen Sie ein rekursives Programm in Pseudocode, welches den maximalen Wert der Bücher berechnet, die auf dem Bücherbrett Platz finden.

Geben Sie die Laufzeit des rekursiven Algorithmus an.

**5 P** b) Entwerfen Sie in Pseudocode einen Algorithmus nach dem Muster der dynamischen Programmierung, der den maximalen Wert der Bücher berechnet.

Geben Sie die Laufzeit Ihres Algorithmus an.

**2 P** c) Beschreiben Sie in Worten, wie durch Rückverfolgung die richtigen Bücher für das Brett gefunden werden können.

Geben Sie die Laufzeit für die Rückverfolgung an.

**3 P** d) Entwerfen Sie in Pseudocode einen Algorithmus nach dem Muster der dynamischen Programmierung, der den maximalen Wert berechnet für den Fall, dass die Garderobe durch ein ganzes Bücherregal mit `K: INTEGER` Bücherbretter der Breite `B` ersetzt wird.

Geben Sie die Laufzeit Ihres Algorithmus an.