Name: | Vorname: | Matr.–Nr.:

Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

SS 2006
12. Oktober 2006

Wiederholungsklausur zu Einführung in die Informatik II

Hinweis: Bei dieser Klausur sind insgesamt **50 Punkte** erreichbar. Die Berechnung der Note basiert jedoch auf maximal **40 Punkten**. Sie können sich also ein Punktedefizit von 10 Punkten erlauben, ohne dass dieses in die Notenfindung eingeht!

Aufgabe 1 Transferfragen (10 Punkte)

Beurteilen Sie folgende Aussagen jeweils mit **wahr** oder **falsch** - eine Begründung ist **nicht** nötig. Jede richtige Antwort gibt einen Punkt, jede falsche Antwort einen Punkt Abzug, fehlende Antworten werden nicht gewertet. Werden insgesamt mehr Fragen falsch als richtig beantwortet, so wird die gesamte Aufgabe mit 0 Punkten bewertet.

a) Umso höher die Entropie ist, umso besser kann ein Komprimierungsverfahren arbeiten.

b) Ein optimaler Komprimierungsalgorithmus maximiert die Summe der Informationsgehalte aller Zeichen.

c) Der Aufwand zum Knacken des RSA-Verfahrens steigt linear mit der Länge des verwendeten Schlüssels.

d) Das Man-in-the-Middle-Attack Problem kann durch Verwendung eines TTP-Dienstes (Third-Trusted-Party) gelöst werden.

e) Eine parallele Ausführung ist auch immer nebenläufig.

f) Leser-Schreiber-Probleme können durch statisches Scheduling vermieden werden.

g) Die Verwendung von Threads beschleunigt grundsätzlich die Ausführung eines Programms.

h) Windows XP unterstützt kein präemptives Scheduling.

i) Mit Semaphoren kann man Warteschlangen realisieren.

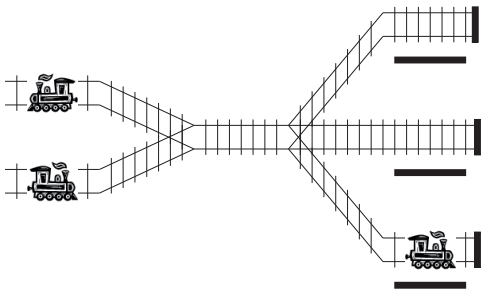
j) Mit Warteschlangen kann man Semaphoren realisieren.

Aufgabe 2 Barrieren (10 Punkte)

In der Vorlesung haben Sie das Konzept der Barrieren (siehe Folie Seite 229). Implementieren Sie ein Klasse Barrier die dieses Konzept umsetzt in Java. Die Klasse soll

- Einen Konstruktor haben, der als Anzahl die an der Barriere beteiligten Prozesse übergeben bekommt.
- Eine Methode `getWaitingProcessCount()`, die die Anzahl der Prozesse zurückgibt, die die Barriere schon erreicht haben.
- Eine Methode `setReached()`, die von den beteiligten Prozessen aufgerufen werden kann und die Ausführung des Prozesses solange verzögert, bis alle anderen Prozesse ebenfalls die Barriere erreicht haben (Implementierung ohne Busy Waiting).

Aufgabe 3 Züge (10 Punkte)



Der dargestellte Sackbahnhof verfügt über drei Bahnsteige. Wegen Bauarbeiten müssen alle ein- und abfahrenden Züge eine gemeinsame Engstelle passieren, die nur von einem Zug gleichzeitig befahren werden kann. Ein ein- und ausfahrender Zug wird durch einen Thread `Train` repräsentiert. Sobald ein Zug auf einen Bahnsteig eingefahren ist, wird ein neuer Schaffner-Thread `Conductor` erzeugt, der nach einer Wartezeit den Zug weiterfahren läßt. Die nachfolgenden Aufgaben sollen in der Programmiersprache Java gelöst werden.

Verwenden Sie zur Umsetzung der Teilaufgaben a) und c) die Klasse `Semaphore`.

Die Synchronisation der Züge außerhalb des Bahnhofes und der Engstelle ist nicht Teil dieser Aufgabe.

a) Implementieren Sie die Methode `arrive()` der Klasse `Train`, die das verklemmungsfreie Einfahren des Zuges in den Bahnhof realisiert. Stellen Sie dabei sicher, dass sich immer nur maximal ein Zug in der Engstelle und höchstens drei Züge im Bahnhof befinden (Sie müssen sich nicht um die Zuweisung der Züge auf die einzelnen Gleise kümmern).

b) Implementieren Sie einen Thread `Conductor`, der eine Wartezeit von 3-5 Sekunden besitzt und danach die Ausführung beendet (zur Simulation des Ein- und Ausstiegs der Passagiere). Erweitern Sie zusätzlich Ihre Klasse `Train` um eine Methode `waitInStation()`, die eine Instanz der Klasse `Conductor` erzeugt und bis zur Beendigung der Ausführung dieses Threads verzögert wird.

c) Implementieren Sie die Methode `depart()` der Klasse `Train`, die das Ausfahren eines Zuges aus dem Bahnhof unter den angegebenen Rahmenbedingungen umsetzt.

Aufgabe 4 Kompressionsalgorithmen **(10 Punkte)**

a) In Vorlesung und Übung wurde der Lempel-Ziv-Welch Algorithmus behandelt (Folien zur Vorlesung s. 50 ff. und Übungsblatt 3). Der Algorihmus erzeugt bei der Ausführung ein Wörterbuch das zur Kompression der nachfolgenden Daten eingesetzt wird. Wir nehmen dabei an, dass das Wörterbuch bereits mit den Buchstaben des Alphabeths gefüllt ist (1="A" bis 26="Z"). Die wesentlichen Strukturen sind ein Puffer p und ein Zeichenleser z. Es soll nun mit Hilfe des Verfahrens der String "MAMAMAMAMIA" komprimiert werden. Zunächst ist der Puffer leer und der Zeichenleser holt das erste Zeichen. Falls die Konkatenation des Puffers mit diesem Zeichen ($p \circ z$) bereits im Wörterbuch eingetragen ist, so wird das Zeichen an den Puffer angehängt und das nächste Zeichen eingelesen (s.u. Tabelle: Übergang Zeile 1 zu 2). Ist ($p \circ z$) noch nicht im Wörterbuch, so wird ein neuer Eintrag vorgenommen, die Wörterbuchkodierung von p ausgegeben und der Puffer geleert und mit z initialisiert (s. u. Tabelle: Übergang Zeile 2 zu 3). Sind keine weiteren Eingabezeichen mehr vorhanden, so wird die Kodierung des Puffers ausgegeben und das Verfahren beendet. Geben Sie nun die Kodierung des gesamten Strings an und ergänzen sie dabei die angegebene Tabelle:

p	z	**Neueintrag**	**Kodierung**
""	'M'		
"M"	'A'	27="MA"	13
"A"	'M'	28="AM"	1

b) Geben Sie nun für denselben String "MAMAMAMAMIA" eine möglichst kompakte Lauflängenkodierung der Form $[n\{x\}]*$ an (anstelle von Wiederholungen von Einzelzeichen können hier im Gegensatz zur Vorlesung auch Wiederholungen längere Sequenzen codiert werden). Dabei ist n die Anzahl der Wiederholungen der nachfolgenden Zeichenfolge x, $[expr]*$ ist der reguläre Ausdruck für beliebige Wiederholungen von $expr$.
"Möglichst kompakt" bedeutet, dass der komprimierte String möglichst kurz ist, wobei jedes Zeichen (auch Ziffern und Klammern) mitgezählt werden.
(Unsinniges) Beispiel: 1{MAMAMAMAMIA}

c) Geben Sie wieder für den String "MAMAMAMAMIA" eine "Longest Fragment First"-Kompression an, wobei folgendes Lexikon verwendet werden soll: $L = \{M, A, I, MA, MAM, MAMI\}$, d.h. der String "MI" hätte die Kodierung "0,2".

d) Nehmen Sie zu den Algorithmen kritisch Stellung (ca. fünf Sätze).

Aufgabe 5 RSA-Verschlüsselung: Schnelles Potenzieren (2+4 Punkte)

Im Rahmen des RSA-Algorithmus werden Algorithmen zur schnellen Potenzierung benötigt (siehe Folien 126-133). Ein mögliches Verfahren ist durch folgende Funktion gegeben (es gilt

nur für n > 0):

```
let potenz (x, n) =
        let rec pot (x, p, n) = match n with
                | 1 -> p*x
                | n when (n mod 2 = 1) -> pot (x * x, p * x, (
                   n-1)/2)
                | n -> pot (x*x, p, n/2)
        in pot (x, 1, n);;
```

a) Geben Sie in einer funktionalen Sprache eine Funktion `M(n)` an, die für das angegebene Verfahren die Anzahl der Multiplikationen berechnet.

b) Beweisen Sie: Mit dem angegebenen Verfahren läßt sich eine Potenz mit positivem Exponenten durch höchstens $1 + 2\log_2 n$ Multiplikationen berechnen.

Aufgabe 6 RSA-Verschlüsselung (4 Punkte)

Wie muss die asymmetrische RSA-Verschlüsselung (siehe Folien 126-133) eingesetzt werden, damit Sender und Empfänger einer Nachricht eindeutig identifiziert werden, also nur der Empfänger die Nachricht lesen kann und darüber hinaus sicher sein kann, dass sie vom Sender ist? Die Schlüssel von Sender und Empfänger sind dabei Beiden bekannt. Begründen Sie Ihre Antwort mathematisch!