Name: | Vorname: | Matr.–Nr.:

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

SS 2006
29. Juli 2006

Klausur 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)

Beantworten Sie die nachfolgenden Fragen möglichst kurz und präzise:

a) In der Vorlesung haben Sie Nachrichtenwarteschlangen (message queues, siehe Folien 228-229) kennengelernt. Welche der in der Vorlesung genannten Problemklassen kann mit message queues gelöst werden?

b) In der Vorlesung wurde das Konzept der Barrieren (siehe Folie 231) erläutert. Wie kann man dieses Konzept für N Prozesse P_1 bis P_N unter Verwendung von Semaphoren und falls nötig Variablen bei festem N umsetzen?

c) Wieso werden bei der Shannon-Fano-Codierung (siehe Folien 34/35) die Zeichen nach ihrer Häufigkeit sortiert? Würde man die Zeichenmenge in zwei beliebige (unsortierte) Teilmengen einteilen, so könnten die Wahrscheinlichkeiten beider Teilmengen doch näher an 50% liegen und damit das einzelne Bit einen höheren Informationsgehalt transportieren. Begründen Sie die Notwendigkeit der Sortierung mit Hilfe eines Beispiels **oder** einer kurzen Begründung.

d) Vergleichen Sie die Verfahren Longest-Fraction-First (siehe Folie 41) mit dem LZW-Verfahren (siehe Folie 49-51) bei gleich großen Wörterbüchern und geben Sie dabei für jedes Verfahren einen Grund für dessen Benutzung an. Der beim LZW-Verfahren notwendige Rechenaufwand zum Aufbau des Wörterbuches soll dabei nicht berücksichtigt werden.

Aufgabe 2 LZ77: Sliding-Window Algorithmus (10 Punkte)

Der LZ77-Kompressionsalgorithmus (siehe Folien 43-46) oder Sliding-Window-Algorithmus komprimiert Zeichensequenzen und verwendet dabei ein Fenster und einen Puffer (im Gegensatz zum originalen Algorithmus werden hier zur Vereinfachung Fenster und Puffer unbegrenzter Größe verwendet), durch die der Text geschoben und dabei codiert wird. Dabei

- speichert das Textfenster `window` den bereits bearbeiteten Text,
- enthält der Puffer `puffer` die noch nicht codierten Zeichen.

Funktionsweise:

- Suche im Textfenster `window` das Präfix des Puffers mit maximaler Länge, das so genannte maximale Präfix `pre`;
- Ist das maximale Präfix `pre` das leere Wort, so wird das erste Zeichen `z` des Puffers mit (0,0,z) codiert und dieses Zeichen `z` vom Puffer ins Fenster geschoben.
- Hat das maximale Präfix `pre` eine Länge > 0, so wird dieses Präfix mit `(pos, len, next)` codiert, wobei `pos` die Position des Teilstrings im Textfenster, `len` die Länge des maximalen Präfixes und `next` das Zeichen, das im Puffer nach dem Präfix folgt, bezeichnet. Das Präfix `pre` und das Zeichen `next` werden nun in das Textfenster geschoben und der Puffer um das Präfix `pre` und das Zeichen `next` gekürzt.
  Anmerkung: tritt das maximale Präfix `pre` mehrfach im Textfenster `window` auf, so kann zur Vereinfachung ein beliebiges Vorkommen zur Positionsbestimmung verwendet werden.
- Das Verfahren wird solange fortgesetzt bis der Puffer leer ist.

a) Das Wort `MAMAMAMAMIA` soll mit dem LZ77-Algorithmus codiert werden. Geben Sie die Codierung an!

b) Im Weiteren soll der LZ77-Algorithmus in einer funktionalen Sprache umgesetzt werden; die Zeichensequenz sei hierbei als Liste von Zeichen codiert. Folgende Funktionen sind gegeben:

   - Eine Funktion `length ls` zur Bestimmung der Länge von Sequenzen,
     Beispiel: `length [A;B;C] = 3`
   - Eine Funktion `substring sub text`, die die Position eines Teilstrings `sub` in einem Text ermittelt. Die Position wird dabei vom letzten Zeichen des Textes beginnend mit 0 gezählt. Enthält der Text den Teilstring nicht, so wird -1 zurückgegeben.
     Beispiel: `substring [B;C;A] [A;B;C;A] = 2`
     `substring [D] [A;B;C;A] = -1`

   Implementieren Sie nun den LZ77-Algorithmus in folgenden Schritten:

   - Geben Sie eine Funktion `nextpuffer puffer window` an, die für einen Puffer und das Fenster, das maximale Präfix `pre`, die Einfügeposition `pos` und den modifizierten Puffer `modifiedPuffer` (Puffer ohne das Präfix) berechnet und diese drei Elemente als Tupel zurückgibt (Einbettung verwenden!).
     Beispiel:
     `nextpuffer [B;C;A;F;G;H] [D;A;B;C;A;B;C] = ([B;C;A],4,[F;G;H])`
   - Implementieren Sie nun die Funktion `encode`, die eine Zeichensequenz gemäß dem LZ77-Algorithmus codiert (Einbettung verwenden!). Das Ende des Textes soll durch das reservierte Zeichen EOF gekennzeichnet werden.
     Beispiel: `encode [A;B;A;A;A] = [(0,0,A);(0,0,B);(1,1,A);(3,1,EOF)]`

Aufgabe 3 RSA (10 Punkte)

Ein berühmtes asymmetrisches Verfahren in der Kryptologie ist der RSA-Algorithmus (siehe Folien 126-133). Hierzu werden große Primzahlen (p, q) und ihr Produkt ($N = p \cdot q$) eingesetzt. Mit dem öffentlichen Schlüssel N sowie einer weiteren aus p und q ableitbaren Zahl e kann ein verschlüsselter Text C erzeugt werden:

$$C = M^e \bmod N$$

a) Zur effizienten Berechnung dieser Formel ist folgende Umformung nützlich. Zeigen Sie dass diese zulässig ist:

$$x^n \bmod y = \left(x \cdot \left(x^{n-1} \bmod y\right)\right) \bmod y$$

Hilfestellung:

$$x^{n-1} = r + m \cdot y$$

b) Die folgende Implementierung von RSA kann wegen der Beschränkung von Integer-Werten (max. $2^{31} - 1$) so nicht eingesetzt werden. Verbessern Sie diese Version unter Verwendung der Erkenntnisse aus Teilaufgabe a. Implementieren Sie Ihre Lösung in Java.

```
public static String encrypt(String text, int n, int e) {
  String encrypted = "";
  for (int i = 0; i < text.length(); i++){
    int m = text.charAt(i);
    int c = (int) Math.pow(m, e) % n;
    encrypted += (char) c;
  }
  return encrypted;
}
```

c) Ist es möglich die oben angegebene Codierung mit einfachen Mitteln zu entschlüsseln? Wie kann dies verhindert werden? Eventuell erforderliche Änderungen des Codes müssen nicht im Rahmen dieser Aufgabe implementiert werden.

Aufgabe 4 Prozesse (10 Punkte)

In einem Kochbuch findet sich folgendes Rezept für das Backen eines Kuchens: als Zutaten sollen nur Eier, Mehl, Milch und Zucker verwendet werden. Zunächst werden die Eier aufgeschlagen und das Eiweiß vom Eigelb getrennt. Das Mehl muss noch gesiebt werden, bevor es verwendet werden kann. Das Eigelb wird mit Milch und Mehl vermischt und in eine Form gegeben. Das Eiweiß wird steif geschlagen, mit Zucker vermengt und als Überzug auf den Kuchen in der Form gegossen. Danach wird das Ganze noch gebacken. Das Rezept soll dabei in folgende Aktionen zerlegt werden können:

a_0 : `Eier aufschlagen und Eiweiss vom Eigelb trennen`

a_1 : `Mehl sieben`

a_2 : `Eiweiss schlagen und mit Zucker vermischen`

a_3 : `Eigelb, Milch und Mehl vermischen und in die Form geben`

a_4 : `Den Eiweissueberzug auf den Kuchen giessen und backen`

Das Backen des Kuchens soll nun als Menge von gültigen Prozessen $P = (E, <, \alpha)$ dargestellt werden. Dazu wird für jede der o.g. Aktionen jeweils ein Ereignis e_i, das die Aktion auslöst, definiert.
Es gilt also $E = \{e_0, \ldots, e_4\}$ mit $\alpha(e_i) = a_i$

a) Zur vollständigen Definition der Prozessmenge benötigt man noch die Kausalitätsrelation $< \subseteq E \times E$, d.h. eine Festlegung welches Ereignis unbedingt vor einem anderen stattfinden muss. Geben Sie diese an.

b) Zeichnen Sie das entsprechende Aktionsdiagramm des Kuchenrezepts (Hinweis: Ereignisse sollen durch Knoten und die Kausalitätsrelation als gerichtetete Kanten, d.h. Pfeile zwischen den Knoten, dargestellt werden).

c) Geben Sie die transitive Hülle der Kausalitätsrelation an.

d) Geben Sie das kleinste Präfix P_3 an (formale Definition des entsprechenden Prozesses).

e) Geben Sie drei verschiedene vollständige Sequenzialisierungen des Kuchenrezepts an.

Aufgabe 5 Leser-Schreiber-Problem **(10 Punkte)**

Lösen Sie das Leser-Schreiber-Problem mit Schreiberpriorität (z.B. Folien 233-235) in Pseudocode mit Hilfe von Semaphoren. Verwenden Sie folgende Notation:

```
Semaphor mySem=1;//Dekl. eines Semaphores (initialer Wert 1)
int      myInt=5; //Dekl. einer Variablen (initialer Wert 5)
down(mySem);   //Erniedrigung eines Semaphors (pot.Blockade)
up(mySem);     //Erhöhung eines Semaphors
```

a) Geben Sie die benötigten globalen Semaphoren und Variablen inklusive Initialisierung an.

b) Ergänzen Sie folgenden Code mit Operationen auf den von Ihnen eingeführten Semaphoren und Variablen, so dass der Code für einen Leserprozess geeignet ist:

```
...
x=readData();
...
```

c) Ergänzen Sie folgenden Code mit Operationen auf den von Ihnen eingeführten Semaphoren und Variablen, so dass der Code für einen Schreiberprozess geeignet ist:

```
...
writeData(y);
...
```