Dr. Johannes Mayer
Axel Blumenstock
Christian Ehrhardt

Allgemeine Informatik III / Systemnahe Software I (WS 2005/2006)

Abgabetermin: 8. November 2005

Wichtiger Hinweis: Dieses Übungsblatt wurde nachträglich etwas vereinfacht, um ohne Bitoperationen auszukommen. Sollten Sie Ihre Lösung bereits auf Basis der ursprünglichen, in der Übung vorgestellten Fassung erstellt haben, wird diese selbstverständlich genauso akzeptiert.

Ferner sei nochmals ausdrücklich darauf hingewiesen, dass Zusatzaufgaben in der Regel über den Stand der Vorlesung hinausgehen und für Leute gedacht sind, die noch Anregungen für eigene Experimente brauchen. 100 Prozent der Übungspunkte bleiben auch für jene erreichbar, die keine Zusatzaufgaben machen.

Bilder fettarm (10 Punkte)

Paul P. Lanlos sitzt ganz schön in der Patsche. Zuerst hat er sich eine Digitalkamera aufschwatzen lassen, die grade mal Fotos mit 50 Kilopixel zustande bringt. In Schwarzweiß.

Und jetzt, da er seine ersten Urlaubsfotos von der Schwäbischen Alb seiner übergewichtigen Freundin Berta B. Reit schicken will, fällt ihm ein, dass Bilder mit fetten Linien gar nicht so gut sind für sie.

Doch glücklicherweise kennt er Sie! Sie werden ihm nun helfen, die Bilder ein wenig leichtgewichtiger zu machen.

Werfen wir zunächst einen Blick auf die Bilder. Sie liegen im PBM-Format (*portable bitmap*) vor, von dem es wiederum eine ASCII- und eine Raw-Variante gibt.

Die ASCII-Variante sieht folgendermaßen aus: In der ersten Zeile steht lediglich „P1“, in der zweiten die Breite w und Höhe h des Bildes in Pixeln. Danach folgen $h \times w$ Nullen oder Einsen, wobei eine „1“ für ein schwarzes und „0“ für ein weißes Pixel steht. Ein Beispiel:

```
P1
16 4
0 0 1 1 1 1 0 0  0 0 1 1 1 1 0 0
0 1 1 1 1 0 0 0  0 1 1 1 1 0 0 0
1 1 1 1 0 0 0 0  1 1 1 1 0 0 0 0
1 1 1 1 1 1 1 1  1 1 1 0 0 0 0 0
```

Deutlich kompakter kommt die Raw-Variante daher. Sie beginnt mit der Kennung „P4“ und nennt ebenfalls w und h. Nach einem weiteren Newline-Zeichen folgt der Datenblock, in dem kein strukturierender Whitespace mehr erlaubt ist[1]:

```
P4
16 4
<<xxððÿà
```

Wenn wir aus diesem Datenblock Zeichen für Zeichen lesen, erhalten wir als erstes ein `<`, das laut ASCII-Tabelle (siehe `man ascii`) dem Wert $60_d = 3C_h = 00111100_b$ entspricht, also genau obigem Bitmuster. Das dritte Zeichen ist `x` mit dem Wert $120_d = 78_h = 01111000_b$.

(a) Schreiben Sie nun ein Programm, welches ein Bild im PBM-**ASCII**-Format[2] (also mit der Kennung „P1“) über die Standardeingabe (`scanf`) einliest und ein solches über die Standardausgabe ausgibt:

```
a.out < schaf-asc.pbm > ergebnis.pbm
```

Dabei soll jedes schwarze Pixel durch ein weißes ersetzt werden und umgekehrt. Geben Sie zur Kontrolle die Breite und Höhe über `stderr` (!) aus, und melden Sie auf diesem Wege auch, wenn die Eingabedatei nicht dem erwarteten Format entspricht.

Wenn Ihr Programm eine sinnvolle Ausgabe zu erzeugen scheint, können Sie diese auch direkt an das Bildbearbeitungsprogramm *xv* weiterleiten:

```
a.out < schaf.pbm | xv -
```

Einige Testbilder finden Sie auf der Vorlesungshomepage. Sie können davon ausgehen, dass die Breiten aller Testbilder ein Vielfaches von 8 sind.

(b) (3 Zusatzpunkte) Hier nun wollen wir das Bild „leichter“ machen. Wandeln Sie dazu Ihr Programm so ab, dass es nur mehr die *Umrisse* der Figur im Eingabebild ausgibt. Es soll genügen, dass Sie die das Bitmuster zeilenweise durchgehen. Ändert sich dabei die Farbe des Pixels, geben Sie ein schwarzes Pixel aus, ansonsten ein weißes. (Tip: Hier kann Ihnen der Exclusiv-ODER-Operator `^` nützlich sein. Damit das Ausgabebild die Breite des Eingabebildes behält, vergleichen Sie das erste Pixel einer Zeile mit „weiß“.)

(c) (3 Zusatzpunkte) Da wir das Eingabebild zeilenweise abarbeiten, werden die Umrisse horizontaler Strukturen nicht erkannt. Lösen Sie dieses Problem, indem Sie das aktuelle Pixel auch mit dem in der Zeile darüber vergleichen und genau dann schwarz ausgeben, wenn sich das aktuelle Pixel von seinem linken oder seinem darüberliegenden Pixel unterscheidet, oder von beiden.

[1] In dieser Darstellung wird Latin-1 Codierung vorausgesetzt
[2] Die ursprüngliche Fassung dieses Übungsblattes verlangte hier das PBM-Raw-Format.

Top Secret (10 Punkte)

Paul P. Lanlos ist begeistert von Ihrem Programm. Darum erhalten Sie auch gleich einen Folgeauftrag: Weil ihm das Verschicken der Bilder an seine Freundin Berta so peinlich ist, sollen Sie die Nachrichten verschlüsseln. Zu diesem Zweck implementieren Sie ein vereinfachtes RSA (man weiß ja nie).

RSA in einem Satz erklärt: Sind drei Zahlen (ein Verschlüsselungsexponent e, ein Entschlüsselungsexponent d sowie ein Modul m) geeignet gewählt, so lässt sich eine Botschaft in Form einer Ganzzahl $b \in \{0 \dots m-1\}$ mit der Abbildung $c = b^e \mod m$ chiffrieren und mit $b = c^d \mod m$ wieder dechiffrieren.

Normalerweise ist m so groß, dass die gesamte Nachricht in b untergebracht werden kann. Wir wollen jedes Zeichen einzeln codieren, das heißt, wir wählen für b einfach das jeweils eingelesene Zeichen und wenden die Chiffriervorschrift darauf an.[3]

(a) Implementieren Sie zunächst eine Funktion

```
int modPow(int a, int x, int m)
```

welche $a^x \mod m$ berechnet. Beachten Sie, dass a^x sehr groß werden kann, so dass es in keinem der Standard-Datentypen (wie double, long long) mehr repräsentiert werden kann. Nutzen Sie stattdessen die Binärdarstellung des Exponenten sowie die Beziehungen

$$\begin{aligned} a^{2x} \mod m &= (a^x \mod m)^2 \mod m \\ a^{2x+1} \mod m &= ((a^x \mod m)^2 \cdot a) \mod m \end{aligned}$$

Schreiben Sie eine kurze `main()`, um Ihre Funktion zu testen. Beispielsweise sollten Sie $65^{343} \mod 28583 = 11520$ erhalten. Weitere Fälle können Sie mit dem Kommandozeilenrechner *bc* überprüfen.

(b) Verwenden Sie als Parameter fest $e = 343$, $d = 24007$ und $m = 28583$ und implementieren Sie ein kleines Programm, welches die Standardeingabe zeichenweise liest und für jedes Zeichen b den Wert $c = b^e \mod m$ als Dezimalzahl ausgibt. Beispielsweise soll für die Eingabe „Aha“ die Ausgabe „11520 25377 4219“ erzeugt werden. Schreiben Sie ein weiteres kleines Programm, das eine derartige Folge von Dezimalzahlen einliest und wieder zum Klartext dechiffriert.

(c) Was hat Paul wohl seiner Berta geschrieben?

```
24027 11520 5356 14399 1054 24725
20253 268 24027 1054 17220 14399
26786 1496 20253 12261 12261 12560
1054 25489 11931 19835 22816
```

Viel Erfolg!

[3]Sollten Sie tatsächlich einmal ein reales Kryptosystem implementieren müssen, nehmen Sie von der zeichenweisen Codierung natürlich Abstand, da sich so erzeugte Texte sehr leicht dechiffrieren lassen.