Praktische Informatik 1 WS 98/99

Prof. Bernd Krieg-Brückner
Axel Lankenau
Shi Hui
Kolyang
Christoph Lüth
Lutz Zegartowski

2. Übungsblatt

Ausgabe: 05.11.1998

Bearbeitungszeit: zwei Praktikumstermine

---

1 *Primzahlen* *Wichtung: 15%*

Implementiere eine Funktion `is_prime :: Int-> bool`, die `True` liefert, wenn ihr Argument eine Primzahl ist, und `False` wenn nicht.

2 *Größter Gemeinsamer Teiler* *Wichtung: 25%*

Euklids Methode, den größten gemeinsamen Teiler (ggT) zu berechnen, haben wir bereits kennengelernt. Leider ist diese Methode etwas ineffizient, weshalb wir eine etwas schnellere Methode implementieren wollen. Diese beruht auf der Annahme, daß das Halbieren einer Zahl eine "schnelle" Operation ist, gegeben durch die Operation *halb*, die für gerade x definiert ist als $2 * halb(x) = x$. Seien zwei Zahlen $a, b \in \mathbb{N}$ gegeben so daß $a \geq b$, dann haben wir für folgende Fälle:

- Wenn a und b gerade sind, dann ist $ggT(a, b) = 2 * ggT(halb(a), halb(b))$;
- Wenn a ungerade und b gerade ist, dann ist $ggT(a, b) = ggT(a, halb(b))$
- Wenn a und b ungerade sind, dann ist $ggT(a, b) = ggT(halb(a - b), b)$

1. Die Operation *halb* ist nur für gerade Zahlen definiert. Warum tritt in den drei Gleichungen oben trotzdem kein undefinierter Fall auf? Darauf aufbauend begründe die Gültigkeit der drei Gleichungen.
2. Basierend auf der Fallunterscheidung und den drei Gleichungen, implementiere eine Funktion `fast_gcd :: Int-> Int`, die den größten gemeinsamen Teiler berechnet. Dafür muß *halb* natürlich auch implementiert werden. Was ist die Abbruchbedingung?

3 *Fibonacci* *Wichtung: 30%*

Der frühe Informatik-Pionier Leonardo "Fibonacci" von Pisa (ca. 1200) war ein begeisterter Kaninchenzüchter. Er erfand die *Fibonacci-Zahlen*, einen Klassiker der Informatik, die auf folgenden Annahmen basierend die Entwicklung einer idealisierten Kaninchenpopulation beschrieben:

- Zu jedem Zeitpunkt i gibt es A_i alte Kaninchenpaare, und J_i junge Kaninchenpaare.
- Zu jedem Zeitpunkt i erzeugt jedes alte Paar ein junges Paar, und jedes junge Paar wird zu einem alten Paar.

Die Gesamtzahl der Kaninchen F_i zum Zeitpunkt i ist dann immer $F_i = A_i + J_i$. Ferner sterben Kaninchen nicht. Die Zucht wird begonnen mit einem jungen Kaninchenpaar; vorher haben wir gar keine Kaninchen.

Gerüchten zufolge gibt es für die Fibonacci-Zahlen eine sogenannte geschlossene (i.e. nichtrekursive) Darstellung:

$$F'_k = \frac{\phi^k - \hat{\phi}^k}{\sqrt{5}} \tag{1}$$

Hierbei sind ϕ und $\hat{\phi}$ der sogenannte Goldene Schnitt und dessen Konjugat:

$$\phi \stackrel{def}{=} \frac{1+\sqrt{5}}{2} \qquad \hat{\phi} \stackrel{def}{=} \frac{1-\sqrt{5}}{2}$$

1. Leite aus den Annahmen und der Definition von F_i eine rekursive Formel für F_i her. Implementiere diese Definition als Funktion `fibrec :: Int-> Int`, die für $i \in \mathbb{N}$ die Fibonacci-Zahl F_i berechnet.

2. Aufwandsabschätzung für `fibrec`: für $n \in \mathbb{N}$, wie oft ruft `fibrec` sich rekursiv auf? Gib eine grobe Abschätzung an.

3. Definiere eine Funktion `fibit :: Int-> Int`, die für $i \in \mathbb{N}$ die Fibonacci-Zahl F'_k nach Gleichung 1 berechnet.

4. Liefert Gleichung 1 wirklich die Fibonacci-Zahlen? Schreibe ein Funktion `check :: Int-> bool`, die für $n \in \mathbb{N}$ prüft, ob für alle $i < n$, $F_i = F'_i$ gilt.

   Bis zu welchen Zahlen kann die Prüfung realistisch durchgeführt werden?

$\boxed{!}$ *Zusatzaufgabe*
Beweise (durch natürliche Induktion), daß $F'_n = F_n$.

$\boxed{4}$ *Da hört's auf...?* *Wichtung: 30%*

Der als besonders grausam geltende Sultan von Terminatien pflegt zum Tode verurteilte Gefangene anstelle sie kurzerhand den Krokodilen vorzuwerfen (wie dies seine Vorväter taten) einer perfiden Prozedur auszusetzen: die Gefangengen dürfen sich aus einer Schüssel eine Handvoll Reis ziehen. Danach beginnt folgende Prozedur:

- haben sie eine gerade Anzahl Reiskörner in der Hand, dürfen sie genau die Hälfte weglegen;
- haben sie eine ungerade Anzahl n, bekommen sie noch einmal $2n + 1$ Reiskörner hinzu;

Diese Prozedur zieht sich solange hin, bis entweder nur noch ein Reiskorn übrig ist, dann ist der Gefangene frei, oder bis der Gefangene an hohem Alter verstirbt. Merkwürdigerweise kamen bisher alle Gefangenen frei, bis auf einen Mathematiker, der behauptet, er müsse sowieso freigelassen werden, da es egal sei, wieviele Körner Reis er ziehe, irgendwann würde sowieso immer nur noch ein Korn Reis übrig bleiben. Leider grübelt er seit 13 Jahren im Gefängnis des Sultans an einem Beweis seiner Behauptung.

1. Implementiere eine Funktion `free :: Int-> Bool`, die testet, ob der Gefangenene freigelassen wird, wenn er n Reiskörner gezogen hat.

2. Erweitere diese Funktion zu einer Funktion `when_free: Int-> Int`, die berechnet wie oft der Gefangene bei n Reiskörnern die Prozedur oben durchlaufen muß.

3. Sollte der Mathematiker sich eine Handvoll Reis greifen und anfangen zu zählen, oder lieber an dem Beweis weiterarbeiten? Begründe Deine Antwort; ggf. implementiere weitere Hilfsfunktionen.