Computer algebra

—

a beginners course using GAP

Friedrich-Schiller-Universität Jena, WS 2014

Jürgen Müller

Inhalt

1 Erste Schritte

(1.1) Aufgabe: Einloggen.
a) Um die folgenden Aufgaben bearbeiten zu können, müssen Sie sich zunächst auf einem der Rechner einloggen. Starten Sie eine Konsole, einen Text-Editor und einen Browser, die Sie über das Menü finden.

b) Die nachfolgenden Betriebssystemkommandos werden in der Konsole ausgeführt. Hilfen zu den Kommandos erhalten Sie mittels 'man ⟨command⟩'; mit den Pfeiltasten können Sie in der Anzeige scrollen, und sie mit 'q' verlassen. Probieren Sie die Hilfe zu den nachfolgenden Kommandos jeweils aus. Dabei werden Sie sehen, daß die Kommandos jeweils (geschickt gewählte?) Abkürzungen sind.

(1.2) Aufgabe: Verzeichnisse und Dateien.
a) Orientieren Sie sich zunächst über die Verzeichnisstruktur Ihres Rechners; verwenden Sie dazu die Kommandos 'pwd', sowie 'cd' oder 'chdir'. Inhalte von Verzeichnissen können Sie sich mittels 'ls ⟨name⟩' ansehen; was macht 'ls'? Neue Verzeichnisse erzeugen Sie mittels 'mkdir ⟨name⟩'; erstellen Sie eines mit dem Namen 'test'. Was bewirken die Kommandos 'cd test' sowie 'cd ..' und 'cd'? Stellen Sie einen Teil der Verzeichnisstruktur graphisch dar, er sollte insbesondere Ihr eigenes Nutzerverzeichnis (auch 'Persönlicher Ordner' genannt) enthalten.

b) Benutzen Sie das Menü Ihres Editors, um eine Datei 'loeschen' im Verzeichnis 'test' zu öffnen, und die leere Datei zu speichern. Um sicher zu gehen, daß die Datei 'loeschen' wirklich erzeugt wurde, sehen Sie sich den Inhalt des Verzeichnisses 'test' an. Dateien löschen Sie mittels 'rm ⟨name⟩'. Löschen Sie die soeben erzeugte Datei wieder, und vergewissern Sie sich. Was passiert, wenn Sie so versuchen, das Verzeichnis 'test' zu löschen?

c) Öffnen Sie eine Datei 'text' im Verzeichnis 'test', kopieren Sie den bisherigen Inhalt der Konsole hinein, speichern Sie die Datei, und schließen Sie den Editor.

(1.3) Aufgabe: Prozesse.
a) Mittels 'ps' erhalten Sie Informationen über die Prozesse, die gerade auf Ihrem Rechner laufen. Lassen Sie sich sowohl alle, als auch nur Ihre eigenen Prozesse anzeigen.

b) Starten Sie Ihren Editor erneut, und finden Sie seine ProcessId (PID) heraus. Prozesse können Sie mittels 'kill ⟨pid⟩' beenden. Damit können Sie etwa den Editor (unsauber) beenden. Was passiert, wenn Sie eine darin editierte Datei vorher nicht speichern?

(1.4) Aufgabe: GAP.
a) Die nachfolgenden Aufgaben sollen mit dem Computeralgebra-System GAP bearbeitet werden. Dazu starten Sie GAP in einer Konsole mittels des Kommandos 'gap'. Warten Sie das Erscheinen des Prompts 'gap>' ab, dann können Sie mittels Eingaben in die Kommandozeile interaktiv mit GAP rechnen. Mittels

'Ctrl-c' können Sie laufende Rechnungen, und mittels 'quit;' oder 'Ctrl-d' können Sie GAP ganz beenden.

b) Im folgenden werden Sie häufig selbst GAP-Programme schreiben. Um die grundlegenden Elemente von GAP kennenzulernen, sehen Sie sich die browsergestützte Einführung in GAP an, die Sie auf der Internet-Seite 'http://www.gap-system.org/Manuals/doc/tut/chap0.html' finden; dort insbesondere die Abschnitte 'A First Session with GAP' (2.1–2.8), 'Lists and Records' (3.1, 3.4–3.7) und 'Functions' (4.1–4.4). Probieren Sie einige der dort genannten Beispiele aus.

c) Die Benutzung von Computeralgebra-Systemen wie GAP lernt man im übrigen am einfachsten spielerisch, durch Versuch und Irrtum, und es dauert nicht lange, bis es richtig Spaß macht! Falls es Ihnen so geht, und Sie mehr davon wollen, so können Sie GAP (kostenlos und leicht) auf Ihrem eigenen Rechner installieren; siehe dazu die Internet-Seite 'http://www.gap-system.org/gap.html'.

(1.5) Aufgabe: Fakultäten.
a) Für $n \in \mathbb{N}$ ist die **Fakultät** definiert als $n! := n \cdot (n-1) \cdots 2 \cdot 1$, und man setzt $0! := 1$. Schreiben Sie Funktionen die auf verschiedene Weisen $n!$ berechnen:
i) Verwenden Sie eine for-Schleife über einen geeigneten range.
ii) Verwenden Sie eine while-Schleife über einen geeigneten absteigenden Zähler.
iii) Verwenden Sie die Beziehung $n! = n \cdot (n-1)!$, für $n \geq 2$, für eine rekursive Funktion, die sich selbst aufruft.

Den Programmcode können Sie am besten in einer Text-Datei schreiben, die typischerweise das Kürzel '.g' oder '.gap' trägt, und per 'Read("⟨name⟩");' in GAP eingelesen wird. Lesen Sie dazu nochmals die Kommentare in der Einführung zu den genannten Elementen der GAP-Programmiersprache.

b) Vergleichen Sie für $n \in \{1, \ldots, 1000\}$ die Ergebnisse Ihrer drei Funktionen mit denen der GAP-Funktion 'Factorial'; hier bieten sich etwa Listenkonstruktionen wie 'List([1..1000],n->Factorial(n));' an.

Vergleichen Sie auch die Laufzeiten Ihrer Funktionen mit denen von 'Factorial', etwa für $n := 2^e$, wobei $e \in \{1, \ldots, 16\}$. Verwenden Sie dazu das GAP-Kommando 'time;'. Um gute Zeitmessungen zu erhalten, führen Sie die jeweilige Operation in einer Schleife hinreichend oft aus. (Wieviel Zeit kostet der Schleifendurchlauf?)

(1.6) Aufgabe: Addition und Multiplikation.
Ermitteln Sie die Laufzeiten von Addition $x+y$ und Multiplikation $x \cdot y$ in GAP, in Abhängigkeit von der Eingabegröße von $x, y \in \mathbb{N}$. Betrachten Sie etwa den Fall $x = y := 2^{10^e + f \cdot 10^{e-1}}$, wobei $e \in \{4, \ldots, 6\}$ und $f \in \{0, \ldots, 5\}$.

Verwenden Sie dazu wiederum das GAP-Kommando 'time;', und führen Sie die jeweilige Operation hinreichend oft aus. Interpretieren Sie die Ergebnisse.

2 Teilbarkeit

(2.1) Aufgabe: Erweiterter Euklidischer Algorithmus.
a) Implementieren Sie den Erweiterten Euklidischen Algorithmus als GAP-Funktion (am besten also in einer '.gap'-Datei). Achten Sie dabei darauf, daß Sie jeweils nur die Variablen halten, die später noch verwendet werden. Außerdem können Sie die Anzahl der nötigen Schleifendurchläufe mittels eines geeigneten 'Print'-Kommandos ausgeben. (Ihre Implementation werden Sie übrigens später weiter verwenden.)

Berechnen Sie mittels Ihrer GAP-Funktion die größten gemeinsamen Teiler für einige der folgenden Beispiele; protokollieren Sie dazu auch die Anzahl der nötigen Schleifendurchläufe. Überprüfen Sie die gefundenen Bézout-Koeffizienten auf Korrektheit, und vergleichen sie Ihre Ergebnisse und Laufzeiten mit denen der GAP-Funktionen 'GcdInt' und 'Gcdex'.

Berechnen Sie die obigen größten gemeinsamen Teiler auch mittels der naiven Methode, die die Faktorisierung der betrachteten Zahlen benutzt; verwenden Sie dazu die GAP-Funktion 'FactorsInt'. Vergleichen Sie Ergebnisse und Laufzeiten mit Ihrer Implementation.

b) Die (aus allerlei Gründen mathematisch interessante) Folge $[F_n \in \mathbb{N}; n \in \mathbb{N}]$ der **Fibonacci-Zahlen** ist rekursiv definiert durch $F_1 := 1$ und $F_2 := 1$, sowie $F_n := F_{n-1} + F_{n-2}$ für $n \geq 3$; in GAP erhalten Sie diese Zahlen bereits vorgefertigt mittels der Funktion 'Fibonacci'.

Berechnen Sie $\mathrm{ggT}(F_m, F_n)$ für einige $m, n \in \{1, \ldots, 10^9\}$. (Die Obergrenze ist kein Schreibfehler! Erst für große Zahlen wird der Euklidische Algorithmus richtig beeindruckend.)

Versuchen Sie, für die obigen Beispiele Gesetzmäßigkeiten zu entdecken, was die Anzahl der Schleifendurchläufe, sowie die gefundenen größten gemeinsamen Teiler angeht. Können Sie sie auch formal beweisen? (Computeralgebra hat in der Tat auch einen experimentellen Charakter: Durch Untersuchung vieler Beispiele kann man womöglich Muster erkennen...)

(2.2) Aufgabe: Binärer Euklidischer Algorithmus [Stein, 1967].
a) Es seien $m, n \in \mathbb{N}$. Man zeige: Es gilt

$$\mathrm{ggT}(m,n) = \begin{cases} 2 \cdot \mathrm{ggT}(\frac{m}{2}, \frac{n}{2}), & \text{falls } m \text{ und } n \text{ gerade sind,} \\ \mathrm{ggT}(\frac{m}{2}, n), & \text{falls } m \text{ gerade und } n \text{ ungerade ist,} \\ \mathrm{ggT}(\frac{m-n}{2}, n), & \text{falls } m \text{ und } n \text{ ungerade sind.} \end{cases}$$

b) Entwickeln Sie aus den obigen Relationen einen (rekursiven?) Algorithmus zur Berechnung größter gemeinsamer Teiler und implementieren Sie ihn als GAP-Funktion. Vergleichen Sie die Ergebnisse und Laufzeiten mit Ihrer Implementation des Euklidischen Algorithmus; verwenden Sie dazu insbesondere die Beispiele aus Aufgabe (2.1).

(2.3) Aufgabe: Sieb des Erathostenes.
a) Implementieren Sie das Sieb des Erathostenes als GAP-Funktion, die für $n \in \mathbb{N}$ die (aufsteigend sortierte) Liste $[p \in \mathcal{P}; p \leq n]$ zurückgibt.

Versuchen Sie, eine möglichst effiziente Funktion zu schreiben. Zugelassen sind dabei nur Elemente der GAP-Sprache, aber keine vorgefertigten Funktionen, und alle Rechnungen müssen im Bereich der ganzen Zahlen ablaufen.

Computeralgebra hat manchmal auch Wettbewerbscharakter: Wir werden Ihre Implementationen miteinander vergleichen... Dazu noch ein Hinweis: Ihre Funktion sollte auch noch für $n := 10^8$ in handhabbarer Zeit ein Ergebnis liefern.

b) Benutzen Sie Ihre Implementation des Sieb des Erathostenes, um die Funktion $\pi(n) := |\{p \in \mathcal{P}; p \leq n\}|$, für $n \in \mathbb{N}$, in GAP zu implementieren; beachten Sie die GAP-Funktion 'Length'.

Vergleichen Sie die $\pi(n)$ mit $\lfloor \frac{n}{\ln(n)} \rfloor$, etwa für $n = 10^e$ mit $e \in \{1, \ldots, 8\}$; beachten Sie die GAP-Funktionen 'Float' und 'Log'. Was beobachten Sie?

3 Modulare Arithmetik

(3.1) Aufgabe: Modulare Arithmetik.
a) Implementieren Sie GAP-Funktionen zur Addition und Multiplikation in $\mathbb{Z}_n$, für $n \in \mathbb{N}$, sowie zur Berechnung von Inversen in $\mathbb{Z}_n^*$. Verwenden Sie dazu Ihre Implementation des Erweiterten Euklidischen Algorithmus.

Bestimmen Sie als Anwendung alle $a \in \{0, \ldots, 999\}$, so daß $67a$ in Dezimaldarstellung die drei letzten Ziffern 123 hat. Was passiert, wenn Sie stattdessen $68a$ und/oder 124 betrachten?

b) Untersuchen Sie die Werte der Eulerschen φ-Funktion für einige $n \in \mathbb{N}$, und versuchen Sie, Gesetzmäßigkeiten zu entdecken. Verwenden Sie wiederum Ihre Implementation des Euklidischen Algorithmus, und vergleichen Sie die Laufzeiten mit denen der GAP-Funktion 'Phi'.

(3.2) Aufgabe: Potenzieren.
a) Implementieren Sie eine GAP-Funktion zur Berechnung von $a^e \in \mathbb{Z}_n$, wobei $n \in \mathbb{N}$, $a \in \mathbb{Z}_n$ und $e \in \mathbb{N}$, die 'Repeated Squaring' benutzt. Achten Sie dabei wiederum darauf, welche Variablen benötigt werden, und daß die Binärdarstellung des Exponenten e nicht explizit berechnet wird.

Vergleichen Sie, für einige selbstgewählte (große) Beispiele, die Ergebnisse und Laufzeiten Ihrer Implementation mit denen des naiven GAP-Kommandos '$(a\hat{}e)$ mod n' und der GAP-Funktion 'PowerModInt'.

b) Für $n \in \mathbb{N}_0$ sei $F_n := 2^{2^n} + 1 \in \mathbb{N}$ die n-te **Fermat-Zahl**. Es fällt auf, daß $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65537$ sämtlich Primzahlen sind, was [Fermat, 1640] veranlaßte, zu vermuten, daß alle Fermat-Zahlen unzerlegbar sind. Aber nach [Euler, 1732] ist $F_5 = 4\,294\,967\,297$ zerlegbar:

Zeigen Sie, daß F_5 den Primteiler 641 hat; rechnen Sie dazu, mittels Ihrer GAP-Funktionen zum modularen Addieren, Multiplizieren und Potenzieren, in $\mathbb{Z}_{641}$.

c) Verwenden Sie Ihre Implementation des Siebs des Erathostenes, um einen Primteiler von F_6 zu finden.

(3.3) Aufgabe: Primitivwurzeln.
Implementieren Sie eine GAP-Funktion, die für eine Primzahl $n \in \mathbb{N}$ jeweils die kleinste Primitivwurzel in $\mathbb{Z}_n$ bestimmt. (Zur Überprüfung können Sie Ihre Ergebnisse mit denen der GAP-Funktion 'PrimitiveRootMod' vergleichen.)

Bestimmen Sie jeweils die kleinste Primitivwurzel für die Primzahlen $n \leq 10^4$; verwenden Sie dazu Ihre Implementation des Siebs des Erathostenes. Können Sie eine Systematik finden?

(3.4) Aufgabe: RSA-Cryptosystem mit Signatur.
a) Wählen Sie einen RSA-Modulus $n := pq < 10^{100}$, wobei $p \neq q$ Primzahlen sind; richten Sie es so ein, daß n nicht in handhabbarer Zeit mittels der GAP-Funktionen 'FactorsInt' oder 'FactInt' faktorisiert werden kann. (Diese Funktionen unterscheiden sich in den benutzten Algorithmen, wobei die zweite das gleichnamige Zusatzpaket benötigt, also je nach GAP-Installation nicht zur Verfügung steht.) Wir hatten ja früher schon gesehen, daß es reichlich große Primzahlen gibt, diese sind also leicht zu finden; beachten Sie dazu die GAP-Funktion 'NextPrimeInt'.

Wählen Sie weiter einen öffentlichen RSA-Schlüssel $e \in \mathbb{Z}_n$, und bestimmen Sie einen zugehörigen geheimen Schlüssel $d \in \mathbb{Z}_n$; richten Sie es so ein, daß d nicht leicht aus n und e allein berechnet werden kann.

b) Ziel ist es nun, eine verschlüsselte Nachricht zu senden, und diese gleichzeitig zu authentifizieren. Führen Sie dazu folgendes **Signatur-Protokoll** aus. Wie kann aus den gesendeten Informationen die Nachricht zurückgewonnen werden? Wieso ist es gleichzeitig eine Signatur?

• Wählen Sie die Nachricht $a := m^i \in \mathbb{N}$, wobei $m \in \mathbb{N}$ Ihre Matrikelnummer und $i := \lfloor \log_m(n) \rfloor \in \mathbb{N}$ sind. Wieso ist diese Nachricht im gegebenen Kontext sinnvoll? (Einmal abgesehen davon, daß die Nachricht ja eigentlich nicht öffentlich bekannt, sondern nur für den Empfänger zu entschlüsseln sein soll, aber hier dient das dazu, die Korrektheit Ihrer RSA-Implementation zu überprüfen.)

• Verschlüsseln Sie $a \in \mathbb{Z}_n$ zunächst mittels Ihres geheimen Schlüssels d, danach verschlüsseln Sie das Ergebnis $b \in \mathbb{Z}_n \subseteq \mathbb{Z}_N$ mit unserem unten genannten öffentlichen Schlüssel E. Senden Sie uns das Ergebnis $c \in \mathbb{Z}_N$ (in Ihrer E-Mail-Abgabe), zusammen mit Ihrem Modulus n und Ihrem öffentlichen Schlüssel e. (Wenn Sie gemeinschaftlich abgeben, so führen Sie das für alle Ihre Matrikelnummern aus; dabei brauchen Sie natürlich n und e nur einmal zu wählen.)

Hier ist unser öffentlicher Schlüssel: Es seien $E := 2^{16} + 1 = 65537$ und N die folgende 101-stellige Zahl:

164201572649746711693859559321745062306256253080626
966918555050734106921134058705057028940135621323 61

4 Primzahltests

(4.1) Aufgabe: Pseudo-Zufallszahlen.
a) Implementieren Sie eine GAP-Funktion, die für $n \in \mathbb{N}$ sukzessive eine Folge $[x_1, x_2, \ldots]$ von **Pseudo-Zufallszahlen** in $\mathbb{Z}_n$ generiert, und bei jedem Aufruf das jeweils nächste Folgenglied zurückgibt.

Überlegen Sie sich dazu, welche Variablen lokal und welche global gehalten werden sollten, und verwenden Sie die folgende Rekursion: Für Parameter $a \in \mathbb{Z}_n^*$ und $b \in \mathbb{Z}_n$, und Startwert $x_0 \in \mathbb{Z}_n$, sei $x_i := ax_{i-1} + b \in \mathbb{Z}_n$, für $i \in \mathbb{N}$, gegeben.

b) Untersuchen Sie, für einige selbstgewählte Beispiele, ob die so gefundenen Folgen hinreichend 'zufällig' sind. Wie groß sind etwa die Periodenlängen?

(4.2) Aufgabe: Lucas-Test.
a) Implementieren Sie eine GAP-Funktion, die den den Lucas-Test für $n \in \mathbb{N}$ ausführt, und für unzerlegbares n ein Lucas-Zertifikat zurückgibt. Verwenden Sie dazu Ihre Implementation des modularen Potenzierens. Suchen Sie nach Primitivwurzeln, indem Sie zufällig Elemente von $\mathbb{Z}_n^*$ mittels Ihres Pseudo-Zufallszahlen-Generators auswählen. (Wie stellen Sie sicher, daß die betrachteten Elemente wirklich in $\mathbb{Z}_n^*$ liegen?)

Zur Faktorisierung von $n-1$ gehen Sie wie folgt vor: Implementieren Sie eine GAP-Funktion, die die 'kleinen' Primteiler von $n \in \mathbb{N}$ mittels Probedivision findet; verwenden Sie dazu Ihre Implementation des Sieb des Erathostenes, etwa mit der Schranke 10^6. Wir können also jetzt annehmen, daß n keine 'kleinen' Primteiler hat.

Nun brauchen wir einen schnellen Test, um zu entscheiden, ob n sicher zerlegbar oder wahrscheinlich unzerlegbar ist, und im ersten Fall brauchen wir einen guten Faktorisierungsalgorithmus. (Später werden das der Miller-Rabin-Test bzw. etwa die ρ-Methode sein.) Weil uns diese im Moment noch nicht zur Verfügung stehen, verwenden Sie hilfsweise die GAP-Funktion 'FactorsInt'. Diese führt einen randomisierten Primzahltest durch, so daß die gefundenen Teiler mittels des Lucas-Tests rekursiv als unzerlegbar verifiziert werden müssen.

Dies ergibt schließlich ein Pratt-Zertifikat für die Unzerlegbarkeit von $n \in \mathbb{N}$, bestehend aus einem Lucas-Zertifikat für n, sowie rekursiv aus Pratt-Zertifikaten für die Primteiler von $n-1$. Für die gesicherten 'kleinen' Primzahlen $\leq 10^6$ sind dabei natürlich keine Zertifikate mehr erforderlich.

b) Geben Sie Pratt-Zertifikate für die Primteiler der Fermat-Zahlen F_5 und F_6 an. (Zum Finden der Primteiler reicht Probedivision mit gesicherten 'kleinen' Primzahlen aus, allerdings nicht zum Beweis, daß die gefundenen Faktoren wirklich unzerlegbar sind.)

c) Berechnen Sie Pratt-Zertifikate für die in Ihrem RSA-Modulus verwendeten Primzahlen. (Auch die GAP-Funktion IsPrimeInt führt für große Zahlen nur einen randomisierten Primzahltest durch, so daß Sie erst jetzt sicher sein können, daß die gewählten Zahlen unzerlegbar sind.)

(4.3) Aufgabe: Fermat-Test.
a) Implementieren Sie eine GAP-Funktion, die den randomisierten Fermat-Test für $n \in \mathbb{N}$ und Fehlerwahrscheinlichkeit $0 < \epsilon < 1$ ausführt, und für zerlegbares n einen Fermat-Zeugen zurückgibt. Verwenden Sie dazu Ihre Implementation des modularen Potenzierens und Ihren Pseudo-Zufallszahlen-Generator.

b) Wenden Sie den Fermat-Test auf die Zahlen $n \in \{10 \cdot 10^6, \ldots, 11 \cdot 10^6\}$ an, jeweils mit Fehlerwahrscheinlichkeit $\epsilon := \frac{1}{2^k}$, für $k \in \{1, \ldots, 10\}$. Wieviele wahrscheinliche Primzahlen finden Sie jeweils? Wieviele davon sind Pseudo-Primzahlen (zu welchen Basen)? Wieviele davon sind Carmichael-Zahlen? (Zur Verifikation dürfen Sie hier die GAP-Funktion 'IsPrimeInt', oder Ihre Implementation des Lucas-Tests benutzen.)

c) Welche der Zahlen $10^{200} + 349$ und $10^{200} + 357$ sind unzerlegbar?

d) Für $n \in \mathbb{N}_0$ sei weiter $F_n := 2^{2^n} + 1 \in \mathbb{N}$ die n-te Fermat-Zahl. **Pepin's Test [1877]** besagt, daß für $n \geq 1$ die Basis $3 \in \mathbb{Z}_{F_n}$ entweder ein Lucas-Zeuge für die Unzerlegbarkeit oder ein Fermat-Zeuge für die Zerlegbarkeit ist. Benutzen Sie Ihre Implementation des Fermat-Tests, um damit die Zerlegbarkeit von F_n für $n \in \{5, 6, \ldots\}$ zu zeigen. Für welche n ist das in handhabbarer Zeit machbar?

(4.4) Aufgabe: Miller-Rabin-Test.
a) Implementieren Sie eine GAP-Funktion, die den randomisierten Miller-Rabin-Test für ungerades $n \in \mathbb{N}$ und Fehlerwahrscheinlichkeit $0 < \epsilon < 1$ ausführt, und für zerlegbares n einen starken Zeugen zurückgibt. Verwenden Sie dazu Ihre Implementation der modularen Potenzierens und kleine prime Basen.

b) Wenden Sie den Miller-Rabin-Test auf die Zahlen $n \in \{10 \cdot 10^6, \ldots, 11 \cdot 10^6\}$ an, jeweils mit Fehlerwahrscheinlichkeit $\epsilon := \frac{1}{2^k}$, für $k \in \{1, \ldots, 10\}$, und vergleichen Sie die Ergebnisse und Laufzeiten mit Ihrer Implementation des Fermat-Tests. (Beachten Sie dabei die unterschiedliche Häufigkeit von Zeugen in den beiden Tests.)

Wieviele wahrscheinliche Primzahlen finden Sie jeweils? Wieviele davon sind starke Pseudoprimzahlen (zu welchen Basen)? Wieviele davon sind Carmichael-Zahlen? (Zur Verifikation dürfen Sie wieder die GAP-Funktion 'IsPrimeInt' oder Ihre Implementation des Lucas-Tests benutzen.)

Bestimmen Sie für $n \leq 11 \cdot 10^6$ alle starken Pseudoprimzahlen zur Basis 2 bzw. zu den Basen $\{2, 3\}$. Können Sie auch eine starke Pseudoprimzahl zu den Basen $\{2, 3, 5\}$ bzw. $\{2, 3, 5, 7\}$ finden?

c) Welche der Zahlen $10^{200} + 349$ und $10^{200} + 357$ sind unzerlegbar?

d) Ist 1 195 068 768 795 265 792 518 361 315 725 116 351 898 245 581 zerlegbar? In diesem Falle versuchen Sie, die Faktorisierung zu berechnen.

5 Faktorisierung

(5.1) Aufgabe: Die ρ-Methode.
a) Implementieren Sie eine GAP-Funktion, die für $n \in \mathbb{N}$ die ρ-Methode zur Berechnung eines nichttrivialen Teilers ausführt. Verwenden Sie dabei quadratische Funktionen zur Iteration, variieren Sie diese und den Startwert durch Parameter, und benutzen Sie 'Cycle Detection'.

Geben Sie auch die Anzahl der nötigen Schleifendurchläufe aus. Untersuchen Sie, anhand der folgenden Beispiele, wie diese Anzahl von der Wahl der Parameter abhängt, und wie sich diese Anzahl zur 'Quadratwurzel-Heuristik' verhält.

b) Für $n \in \mathbb{N}_0$ sei wieder $F_n := 2^{2^n} + 1 \in \mathbb{N}$ die n-te Fermat-Zahl; Sie haben ja bereits verifiziert, daß F_n für alle handhabbaren $n \geq 5$ zerlegbar ist. Berechnen Sie für $n \in \{5, \ldots, 12\}$ alle Primteiler $\leq 10^{16}$ von F_n. Versuchen Sie, Primteiler weiterer F_n zu finden, etwa für $n \leq 18$.

Wieviele Schleifendurchläufe sind nötig, um hinreichend sicher zu sein, alle Primteiler bis zu einer gegebenen Schranke gefunden zu haben? Verifizieren Sie (mittels Ihres Lucas-Tests), daß die gefundenen Teiler wirklich unzerlegbar sind, und untersuchen Sie (mittels Ihres Miller-Rabin-Tests) die verbleibenden Kofaktoren von F_n auf Unzerlegbarkeit.

(5.2) Aufgabe: Die $(p-1)$-Methode.
a) Implementieren Sie eine GAP-Funktion, die für $n \in \mathbb{N}$ die $(p-1)$-Methode zur Berechnung eines nichttrivialen Teilers ausführt; behandeln Sie dabei die Glattheitsschranke als Parameter. Wie werden die nötigen logarithmischen Schranken bestimmt, und wie werden kleinste gemeinsame Vielfache berechnet? Sie können auch eine Variante implementieren, die die Primzahlen unterhalb der Glattheitsschranke benutzt; verwenden Sie dazu Ihre Implementation des Siebs des Erathostenes.

b) Versuchen Sie, die oben gefundenen Primteiler von F_n, für $n \in \{5, \ldots, 12\}$, wiederzuentdecken. (Wie können Ihnen die Ergebnisse der Lucas-Tests dabei helfen?) Untersuchen Sie, wie der Erfolg der Methode von der Wahl der Glattheitsschranke abhängt.

(5.3) Aufgabe: Die Dixon-Methode.
a) Implementieren Sie eine GAP-Funktion, die für ungerades $n \in \mathbb{N}$ die Dixon-Methode zur Berechnung eines nichttrivialen Teilers ausführt; verwenden Sie dabei die Größe der Faktorbasis als Parameter. Gehen Sie wie folgt vor:

i) Zur Bestimmung einer Faktorbasis verwenden Sie Ihre Implementation des Siebs des Erathostenes, und lassen Sie (abhängig von n) nur Primzahlen zu, die die Euler-Bedingung erfüllen.

ii) Sammeln Sie die ganzzahligen Exponentenvektoren in einer Liste von Listen auf. Um sie als Matrix über $\mathbb{Z}_2$ aufzufassen, gehen Sie wie folgt vor: In GAP werden endliche Körper mit dem Kommando 'GF(⟨size⟩)' erzeugt, das Einselement darin jeweils mit der Funktion 'One'. Durch Multiplikation Ihrer Liste von Listen mit dem Einselement von $\mathbb{Z}_2$ werden die ganzzahligen Einträge zu Elementen von $\mathbb{Z}_2$, und mit dem GAP-Kommando 'ConvertToMatrixRep' erhalten Sie schließlich eine Matrix über $\mathbb{Z}_2$.

Aus der Linearen Algebra ist bekannt, wie man den Kern einer Matrix über einem Körper berechnet (Gauß-Algorithmus). Dazu dient die GAP-Funktion 'NullspaceMat', die wir übernehmen, weil wir uns hier nicht mit algorithmischer linearer Algebra befassen wollen.

iii) Implementieren Sie zwei Varianten:

(P) Suchen Sie b-Zahlen systematisch in $\mathcal{I} := \{\lfloor\sqrt{n}\rfloor - c, \ldots, \lfloor\sqrt{n}\rfloor + c\}$, von der Mitte des Intervalls aus. (Hier dürfen Sie die GAP-Funktionen 'Sqrt' und 'Float' verwenden.) Dabei kann c variabel sein, und die Suche endet, sobald eine Relation gefunden wird; falls dies zu einer trivialen Faktorisierung führt, so kann etwa die zuletzt gefundene b-Zahl ignoriert und die Suche fortgesetzt werden.

(Q) Suchen Sie b-Zahlen mit dem eigentlichen Quadratischen Sieb, indem Sie für die Primzahlen p in der gewählten Faktorbasis auf einer Liste mit den Werten $f(x)$ in Schritten der Länge p entlanggehen, und nur für $x \in \mathcal{I}$ mit $f(x) = 0 \in \mathbb{Z}_p$ die p-Anteile aus $f(x)$ herausdividieren. Dabei wird die Länge c des Siebintervalls als fester Parameter behandelt.

b) Untersuchen Sie, anhand der folgenden Beispiele, die Häufigkeit von b-Zahlen in Abhängigkeit von b und vom Abstand von der Mitte des Intervalls $\mathcal{I}$. Wie sollten b und c in Abhängigkeit von n gewählt werden? Wie muß dabei c in Abhängigkeit von b variiert werden, und wie verändern sich damit die Laufzeiten? Wie verhalten sich gute Wahlen von b sich zu $\exp(\frac{1}{2}\sqrt{\ln(n)\cdot\ln(\ln(n))})$? Vergleichen Sie die Laufzeiten der Varianten (P) und (Q), und vergleichen Sie das Verhalten der Dixon-Methode mit dem der ρ-Methode.

Hier sind nun einige Beispiele: (Weitere sind weiter unten zu finden.)

i) Für $n \in \mathbb{N}_0$ sei wieder $F_n := 2^{2^n} + 1 \in \mathbb{N}$ die n-te Fermat-Zahl. Faktorisieren Sie erneut F_n für $n \in \{5, 6, 7\}$, und versuchen Sie es für F_8. (Beachten Sie, wieviele Dezimalstellen diese Zahlen jeweils haben.)

ii) Faktorisieren Sie die folgenden Zahlen (vom RSA-Modulus-Typ):

$$\begin{aligned}
&4370161634111152737068 17\\
&2999541446900512353141818303\\
&4341853589553787843317594387 3373\\
&1992674160280932501870083141868 16507\\
&1483757509910600906323875397001481989077
\end{aligned}$$

(5.4) Aufgabe: Faktorisierung.
Nachdem nun auch Faktorisierungsalgorithmen zur Verfügung stehen, können Sie Ihre Implementation des Lucas-Tests zu einem eigenständigen Programmpaket aufbereiten, das in der Lage ist, Zahlen $n \in \mathbb{N}$ faktorisieren, und Pratt-Zertifikate für die Primteiler von n anzugeben. Behandeln Sie dabei die zerlegbaren Primzahlpotenzen gesondert.

Ihre Implementation sollte für beliebige Zahlen $n \leq 10^{35}$, und für weit größere Zahlen mit 'kleinen' Primteilern, handhabbare Laufzeit haben. (Ich finde es übrigens beeindruckend, daß man mit ein wenig raffinierter Mathematik auf den heute üblichen Rechnern so weit kommt.) Hier sind noch ein paar Beispiele:

Für $p \in \mathbb{N}$ sei $M_p := 2^p - 1 \in \mathbb{N}$ die p-te **Mersenne-Zahl**; für zerlegbares p ist M_p (offenbar) zerlegbar, aber man vermutet, daß es für unzerlegbares p unendlich viele unzerlegbare Mersenne-Zahlen gibt. Untersuchen Sie die Zahlen M_p für Primzahlen $p \leq 200$: Welche davon sind wahrscheinlich unzerlegbar? Faktorisieren Sie möglichst viele der anderen.