Algorithmen
für
endliche abelsche Gruppen

Diplomarbeit von
Sachar Paulus

angefertigt nach einem Thema von
Prof. Dr. Johannes Buchmann
am Fachbereich 14 Informatik
der Universität des Saarlandes in Saarbrücken.

14. Juli 1993

Hiermit versichere ich an Eides Statt, daß ich bei der Anfertigung dieser Arbeit keine anderen als die angegebenen Quellen benutzt habe.

Saarbrücken, den 14. Juli 1993

Sachar Paulus

Ich möchte mich bei Herrn Prof. Dr. Johannes Buchmann für das überaus interessante und anregende Thema bedanken und bei Christine Hollinger, Volker Müller, Eric Schank und Jörg Zayer für viele hilfreiche Diskussionen und die Korrekturlesungen.

Desweiteren danke ich Jutta Bartholomes, Franz-Dieter Berger, Thomas Denny und Jürgen Loho für die sehr angenehme und fruchtbare Arbeitsatmosphäre.

Mein Dank gebührt nicht zuletzt meinen Eltern, die mir das Studium der Informatik erst ermöglicht haben, und meiner Freundin Sabine, die, besonders in den letzten Wochen, meine Launen und geistige Abwesenheit immer als erste zu spüren bekam.

Inhaltsverzeichnis

Einleitung

Die Gruppe ist ein zentraler, fundamentaler Begriff der Mathematik. Sie stellt quasi die einfachste bekannte Struktur, die gewisse interessante Eigenschaften aufweist, dar. Die Untersuchung dieses Begriffs aus algorithmischer Sicht ist also von grundsätzlichem mathematischen Interesse. Andererseits — und das ist eigentlich natürlich, denn die in der Mathematik entwickelten Strukturen haben auch einen hilfswissenschaftlichen Charakter: man entwirft ein Modell für eine gegebene naturwissenschaftliche Situation und zieht sich auf dieses Modell zurück, um sich nach getaner Reflexion wieder der praktischen Bedeutung zuzuwenden — findet der Begriff der Gruppe in vielen physikalischen Betrachtungen Anwendung, unter anderem in der Quantenphysik; und da ist eine algorithmische Bestimmung der Eigenschaften von Gruppen wünschenswert. In letzter Zeit konnte man aber auch von derartigen Untersuchungen in neuen Bereichen profitieren, besonders in der Informatik, z.B. in der Kryptographie, wo man die Schwierigkeit der Berechnung der Ordnung einer Gruppe zur Sicherheit von Kryptosystemen ausnutzt. Eine komplette algorithmische Untersuchung von Gruppen ist also sowohl von theoretischem, als auch von praktischem großen Interesse.

In dieser Arbeit beschäftigen wir uns mit der algorithmischen Untersuchung von endlichen abelschen Gruppen.

In einem ersten, einführenden Kapitel werden wir die benutzten mathematischen Bezeichnungen und die zur Darstellung von Algorithmen verwendete Technik erklären, sowie einen Überblick über die benötigten Aussagen der Gruppentheorie geben.

Im zweiten Kapitel werden die *algorithmischen Voraussetzungen*, von denen wir ausgehen, geklärt; d.h. wir geben ein Modell an, in dem unsere Aussagen über die Komplexität unserer Algorithmen konsistent sind. Diese Voraussetzungen werden anhand von drei in der Kryptographie verwandten Gruppen erläutert.

Das dritte Kapitel ist ausschließlich der Berechnung der *Ordnung* eines Gruppenelements gewidmet; hierbei werden wir mehrere Versionen des wohlbekannten Baby-Step-Giant-Step-Algorithmus von Shanks unter verschiedenen Voraussetzungen angeben und vergleichen.

Im anschließenden Kapitel führen wir den zentralen Begriff der *minimalen Relation* ein und entwickeln eine schnelle Methode zur Berechnung von minimalen Relationen. Ein Spezialfall der minimalen Relation ist die Berechung des *diskreten Logarithmus*, auf den wir die erarbeiteten Ergebnisse übertragen.

Kennt man die Primfaktorzerlegung der Gruppenordnung, so kann man diese verwenden, um die Berechnung von minimalen Relationen mittels des Chinesischen Restsatzes auf *p-Gruppen* zu reduzieren; allerdings lohnt sich dies nur unter bestimmten Voraussetzungen, wie wir im fünften Kapitel sehen werden.

Im sechsten Kapitel werden wir zeigen, wie man die Struktur und eine Basis einer Gruppe probabilistisch berechnen kann. Hierzu benötigen wir die *Smith-Reduktion*, deren Funktionsweise wir erläutern. Die Berechnung der Gruppenordnung ergibt sich als Spezialfall der vorgestellten Algorithmen.

Eine wichtige Anwendung in endlichen abelschen Gruppen ist die Bestimmung von *n-ten Wurzeln*; die Berechnung von n-ten Wurzeln werden wir untersuchen unter der Voraussetzung, die Struktur einer

Gruppe zu kennen — in diesem Fall kann man deterministisch *alle* n-ten Wurzeln berechnen — und unter der Voraussetzung, nur die Gruppenordnung zu kennen. In letzterem Fall gibt es probabilistische Algorithmen zur Berechnung einer n-ten Wurzel.

Schließlich werden wir die erarbeiteten Ergebnisse in einem abschließenden Kapitel auf die in Kapitel 2 vorgestellten Beispiele übertragen.

Kapitel 1

Grundlagen

In diesem einführenden Kapitel werden wir zuerst die benutzten Bezeichnungen vorstellen, die Formulierung und Darstellung der Algorithmen erklären und die benutzte O-Notation besprechen. Danach werden wir die grundlegenden Begriffe aus der Gruppentheorie wiederholen, einige in dieser Arbeit benötigte Fakten über Gitter anführen und schließlich den Hauptsatz für endliche abelsche Gruppen vorstellen und erläutern.

1.1 Bezeichnungen

Zur Bezeichnung der Zahlbereiche verwenden wir die folgenden Symbole:

$\mathbb{N}$ Menge der natürlichen Zahlen (ohne Null),

$\mathbb{P}$ Menge der Primzahlen,

$\mathbb{Z}$ Menge der ganzen Zahlen,

$\mathbb{Q}$ Menge der rationalen Zahlen,

$\mathbb{R}$ Menge der reellen Zahlen.

Die folgenden Bezeichnungen werden später nicht explizit eingeführt:

- Für $a, b \in \mathbb{Z}$ mit $a \leq b$ bezeichnet $[\![a, b]\!]$ die Menge $\{i : i \in \mathbb{Z}, a \leq i \leq b\}$.
- Für eine endliche Menge M bezeichnet $|M|$ die Anzahl der Elemente dieser Menge; wir sagen auch Mächtigkeit der Menge.
- Für $a \in \mathbb{R}$ bezeichnet $\lfloor a \rfloor$ die größte ganze Zahl x mit $x \leq a$ und $\lceil a \rceil$ die kleinste ganze Zahl y mit $y \geq a$.
- Für zwei Mengen A, B bezeichnet $A \times B$ das kartesische Produkt von A und B.
- Für $a, b \in \mathbb{Z}$ schreiben wir $a|b$ (bzw. $a \nmid b$) wenn a ein (bzw. kein) Teiler von b ist.
- Für $a, b \in \mathbb{Z}$ und $m \in \mathbb{N}$ schreiben wir $a \equiv b \bmod m$, wenn gilt $m|(b - a)$ und bezeichnen mit $a \bmod m$ den Rest der ganzzahligen Division von a durch m.
- Die Wahrscheinlichkeit für das Eintreffen eines Ereignisses E bezeichnen wir mit $p(E)$.

1.2 Die Darstellung der Algorithmen

Zur Darstellung der Algorithmen verwenden wir eine leicht veränderte Form der Struktogramme (Nassi-Shneiderman-Diagramme). In diesen Diagrammen wird der Ablauf der Algorithmen durch die Aneinanderreihung und Ineinanderschachtelung einzelner Blöcke dargestellt.

Der elementare Block besteht aus einer einzigen Anweisung:

Anweisung	(1)

Mehrere, eng zuammenhängende Anweisungen können auch in einem gemeinsamen Block stehen.

Wir verwenden drei Arten von Schleifenkonstrukten: die FOR-, die WHILE- und die REPEAT-Schleife. Sie werden durch die folgenden Blöcke dargestellt:

FOR $z \leftarrow a$ TO e		(1)
	Anweisung	(2)

WHILE Bedingung		(1)
	Anweisung	(2)

	Anweisung	(1)
UNTIL Bedingung		(2)

Hierbei bezeichnet $v \leftarrow a$ die Zuweisung des Wertes des Ausdrucks a an die Variable v. Die Darstellung von Verzweigungen geschieht durch den folgenden Block:

IF Bedingung		(1)
THEN	Anweisung1	(2)
ELSE	Anweisung2	(3)

Diese Darstellung unterscheidet sich grundlegend von der der Nassi-Shneiderman-Diagramme; der Grund ist, daß eine Positionierung der beiden alternativen Anweisungsblöcke nebeneinander in TEX nur unter erheblichem Aufwand realisierbar ist.

Falls der ELSE-Zweig leer ist, so wird dieser weggelassen. Sind die Anweisungen (oder Anweisungsfolgen) innerhalb eines Konstruktes sehr kurz, so fassen wir Konstrukt und inneren Block in einer einzigen Zeile zusammen, etwa

IF Bedingung THEN Anweisung	(1)

oder

FOR $z \leftarrow a$ TO e DO Anweisung	(1)

In parallelen Algorithmen verwenden wir auch das Konstrukt

FORALL $i \in M$ PARDO		(1)
	Anweisung	(2)

Ein Algorithmus wird beendet durch die Anweisung RETURN { *Ergebnis* }; hierbei wird die Kontrolle, unabhängig von der aktuellen Schachtelungstiefe, an den aufrufenden Algorithmus zusammen mit dem Ergebnis der Berechnung zurückgegeben.

1.3 Die O-Notation

Um den Aufwand eines Algorithmus zu beschreiben, verwenden wir folgende Notationen, die wir auf Laufzeitfunktionen und Wahrscheinlichkeiten anwenden:

$$
\begin{aligned}
f = O(g) &\quad\Longleftrightarrow\quad \exists_{n_0, c \in \mathbb{N}} \ \forall_{n \geq n_0} \ f(n) \leq c \cdot g(n) \\
f = o(g) &\quad\Longleftrightarrow\quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \\
f(n) = n^{O(1)} &\quad\Longleftrightarrow\quad \exists_{d \in \mathbb{N}} \ f(n) = O(n^d) \\
f(n) = n^{o(1)} &\quad\Longleftrightarrow\quad \lim_{n \to \infty} \frac{\log f(n)}{\log n} = 0.
\end{aligned}
$$

Die letzte Schreibweise bedeutet anschaulich, daß f bei wachsendem n echt langsamer wächst als n, also *subpolynomiell* in n ist.

1.4 Elementare Begriffe aus der Gruppentheorie

Sei G eine nichtleere Menge und $+ : G \times G \to G$ eine Verknüpfung auf G.

Das Paar $(G, +)$ ist eine **Gruppe**, wenn folgendes gilt:

1. $+$ ist assoziativ
2. es gibt ein neutrales Element $_G \in G$:

$$\forall_{\gamma \in G} \ \gamma +_G =_G +\gamma =_G$$

3. jedes Element in G hat ein Inverses:

$$\forall_{\gamma \in G} \ \exists_{-\gamma \in G} \ \gamma + (-\gamma) = (-\gamma) + \gamma =_G$$

Statt $(G, +)$ schreiben wir nur G und setzen, wenn wir es nicht explizit angeben, die additive Schreibweise der Verknüpfung implizit voraus. Wir schreiben kurz statt $_G$.

- G heißt **abelsch**, wenn die Verknüpfung von G kommutativ ist.
- $U \subseteq G$ heißt **Untergruppe von** G, wenn $(U, +|_{U \times U})$ eine Gruppe ist, wir schreiben $U \leq G$.
- Es gilt: $_G \in U$ für jede Untergruppe $U \leq G$.
- Für $\underbrace{\gamma + \cdots + \gamma}_{n\text{-mal}}$ schreiben wir $n \cdot \gamma \quad (n \in \mathbb{N}, \gamma \in G)$.

Setzen wir für $\gamma \in G$ $0 \cdot \gamma :=$ und $(-1) \cdot \gamma := -\gamma$, so gilt:

$$\forall_{\substack{n_1, n_2 \in \mathbb{Z} \\ \gamma \in G}} \ (n_1 + n_2) \cdot \gamma = n_1 \cdot \gamma + n_2 \cdot \gamma$$

und

$$\underset{\substack{n\in\mathbb{Z}\\ \gamma_1,\gamma_2\in G}}{\forall} \quad n\cdot(\gamma_1+\gamma_2)=n\cdot\gamma_1+n\cdot\gamma_2.$$

Eine Gruppe $(G,+)$ läßt sich somit auch als **$\mathbb{Z}$-Modul** $(G,+,\cdot)$ mit $\cdot:\mathbb{Z}\times G\to G$ skalare Multiplikation, auffassen. Wir schreiben kurz $n\gamma$ für $n\cdot\gamma$.

- Seien $\gamma_1,\dots,\gamma_m\in G$.

  $$<\gamma_1,\dots,\gamma_m>:=\left\{\sum_{i=1}^{m} n_i\gamma_i : (n_1,\dots,n_m)\in\mathbb{Z}^m\right\}$$

  ist eine Untergruppe von G und heißt **die von $\gamma_1,\dots,\gamma_m$ in G erzeugte Untergruppe**, γ_i **erzeugendes Element** oder **Erzeuger**.
  Falls $<\gamma_1,\dots,\gamma_m>=G$, so nennt man $(\gamma_1,\dots,\gamma_m)$ ein **Erzeugendensystem** von G. Eine Gruppe, die von einem einzigen Element erzeugt werden kann, heißt **zyklisch**.

- Die **Ordnung** $\mathrm{ord}(\gamma)$ eines Elements γ ist $\min\{n:n\in\mathbb{N},n\gamma=\}$. Falls G eine endliche Gruppe ist, so nennt man deren Kardinalität $|G|$ auch die **Gruppenordnung**.

- Ist die Kardinalität von G eine Primzahlpotenz, so nennen wir G **p-Gruppe**.

- Es gilt: $|<\gamma>|=\mathrm{ord}(\gamma)$ für $\gamma\in G$. Allgemein gilt für ein beliebiges $U\leq G$:

  $$|G|=|{}^{G}\!/_{U}|\cdot|U|,$$

  wobei

  $${}^{G}\!/_{U}=\{\bar\gamma:\bar\gamma=\{\gamma+\gamma_0:\gamma_0\in U\},\gamma\in G\}$$

  die **Faktorgruppe** "G modulo U" ist.

- Der **Index** einer Untergruppe $U\leq G$ ist die Mächtigkeit der Faktorgruppe G/U.

Wenn nicht eindeutig klar ist, auf welche Gruppe sich ein Begriff bezieht, werden wir ihn mit der Bezeichnung der entsprechenden Gruppe indizieren.

1.5 Gitter

In diesem Abschnitt führen wir den Begriff des Gitters ein und geben Resultate an, die wir im Zusammenhang mit dieser Arbeit benötigen. Sie sind [Dü92] entnommen; wir verzichten auf einen Beweis. Im Folgenden sei $n\in\mathbb{N}$.

1.1. Definition *Eine Untergruppe $\mathcal{L}$ der additiven Gruppe $\mathbb{Z}^n$ heißt* **Gitter im $\mathbb{Z}^n$**. *Die maximale Anzahl linear unabhängiger Vektoren in $\mathcal{L}$ heißt* **Dimension** *des Gitters und wird mit* $\dim\mathcal{L}$ *bezeichnet.*

1.2. Satz *Sei $\mathcal{L}$ ein Gitter im $\mathbb{Z}^n$. Der Index von $\mathcal{L}$ ist genau dann endlich, wenn* $\dim\mathcal{L}=n$ *ist.*

1.3. Definition *Es sei $\mathcal{L}$ ein n-dimensionales Gitter im $\mathbb{Z}^n$. Der Index von $\mathcal{L}$ in $\mathbb{Z}^n$ heißt* **Determinante** *des Gitters $\mathcal{L}$ und wird mit $Det\mathcal{L}$ bezeichnet.*

1.4. Definition *Es seien* $v_1, \ldots, v_m$ *Vektoren im* $\mathbb{Z}^n$. *Dann bezeichnen wir mit*

$$\mathcal{L} =< v_1, \ldots, v_m >= \left\{ \sum_{i=1}^{m} x_i v_i : x_i \in \mathbb{Z}, \quad i = 1, \ldots, m \right\}$$

das **von** $v_1, \ldots, v_m$ **erzeugte** *Gitter im* $\mathbb{Z}^n$.

1.5. Definition *Es sei* $\mathcal{L}$ *ein Gitter im* $\mathbb{Z}^n$, u, $v \in \mathbb{Z}^n$. *Wir schreiben*

$$u = v \bmod \mathcal{L}$$

falls $u - v \in \mathcal{L}$.

Für eine Matrix $A \in \mathbb{Z}^{n \times m}$ bezeichnen wir mit $\mathcal{L}(A)$ das von den Spaltenvektoren der Matrix erzeugte Gitter, wir nennen es kurz das **von** A **erzeugte** Gitter im $\mathbb{Z}^n$.

Ist $A \in \mathbb{Z}^{n \times n}$ regulär, so sind die Spaltenvektoren der Matrix linear unabhängig und somit $\dim \mathcal{L}(A) = n$. In diesem Fall nennen wir A eine **Basismatrix** von $\mathcal{L}(A)$.

1.6. Satz *Ist* $A \in \mathbb{Z}^{n \times n}$ *regulär, so gilt* $|\det A| = Det\mathcal{L}(A)$.

1.6 Der Hauptsatz für endliche abelsche Gruppen

1.7. Satz (Hauptsatz für endliche abelsche Gruppen) *Sei* G *eine endliche abelsche Gruppe. Dann gibt es eine eindeutige Folge von Zahlen* $s_1, \ldots, s_m \in \mathbb{N}$, *für die gilt:*

$$\forall_{i \in [\![1,m]\!]} \; 1 < s_i \qquad \textit{und}$$
$$\forall_{i \in [\![1,m-1]\!]} \; s_i | s_{i+1}$$

so daß

$$G \cong \mathbb{Z}/_{s_1\mathbb{Z}} \times \cdots \times \mathbb{Z}/_{s_m\mathbb{Z}}.$$

D.h. es gibt $\delta_1, \ldots, \delta_m \in G$ *mit*

$$\forall_{i \in [\![1,m]\!]} \; \operatorname{ord} \delta_i = s_i \qquad \textit{und}$$
$$\forall_{\substack{i,j \in [\![1,m]\!] \\ i \neq j}} \; < \delta_i > \cap < \delta_j >= \{\}$$

so daß

$$G =< \delta_1 > \oplus \cdots \oplus < \delta_m > .$$

Den Beweis dieses Satzes findet man z.B. in [Hu74]. Statt

$$\forall_{i \in [\![1,m]\!]} \; 1 < s_i \qquad \text{und}$$
$$\forall_{i \in [\![1,m-1]\!]} \; s_i | s_{i+1}$$

schreiben wir kurz $s_1|\cdots|s_m, \quad s_1 > 1$.

Jede endliche abelsche Gruppe hat also eine gewisse Struktur, die man durch ein Tupel von Zahlen und ein Tupel von Gruppenelementen ausdrücken kann. Die Kenntnis der einer endlichen abelschen Gruppe zugrundeliegenden Struktur ist nicht nur von theoretischem Interesse, sondern wird auch zur algorithmischen Lösung von bestimmten Problemen benötigt. Da wir immer wieder auf den Sachverhalt des obigen Satzes zurückgreifen werden, definieren wir:

1.8. Definition *Sei G eine endliche abelsche Gruppe, $\delta_1, \ldots, \delta_m \in G$ und $s_1, \ldots, s_m \in \mathbb{N}$ mit:*

$$G \cong \bigoplus_{i=1}^{m} < \delta_i > \ \ mit \ \ < \delta_i > \cong \mathbb{Z}/_{s_i\mathbb{Z}}, \quad s_1|\cdots|s_m, \quad s_1 > 1.$$

Die Folge von Zahlen $(s_1, \ldots, s_m)$ nennen wir die **Struktur** *von G, $(\delta_1, \ldots, \delta_m)$ eine* **Basis** *von G.*

Aus der Kenntnis der Struktur einer endlichen abelschen Gruppe folgt sofort die Kenntnis der Gruppenordnung. Es gilt nämlich:

1.9. Korollar *Sei G eine endliche abelsche Gruppe mit Struktur $(s_1, \ldots, s_m)$. Dann gilt:*

$$|G| = \prod_{i=1}^{m} s_i.$$

Beweis: Sei $(\delta_1, \ldots, \delta_m)$ eine Basis von G. Jedes Element $\gamma \in G$ läßt sich eindeutig darstellen durch

$$\gamma = \sum_{i=1}^{m} g_i \delta_i \quad \text{mit} \quad \forall_{i \in [\![1,m]\!]} \quad 0 \le g_i < s_i;$$

dies folgt unmittelbar aus Satz 1.7. Somit gilt:

$$\begin{aligned} |G| &= \left| \left\{ (g_1, \ldots, g_m) : \forall_{i \in [\![1,m]\!]} \quad 0 \le g_i < s_i \right\} \right| \\ &= \prod_{i=1}^{m} s_i. \end{aligned}$$

■

Aus diesem Korollar kann man erkennen, daß es viele verschiedene mögliche Strukturen von Gruppen mit gleicher Mächtigkeit gibt.

1.10. Beispiel Sei G eine endliche abelsche Gruppe, $|G| = 144$. Die möglichen Strukturen von G sind: $(144), (2, 72), (4, 36), (3, 48), (6, 24), (12, 12), (2, 6, 12), (2, 2, 6, 6)$.

Für eine gegebene endliche abelsche Gruppe ist die Struktur eindeutig; es kann jedoch viele verschiedene Basen derselben Gruppe geben. Bestimmen wir die Anzahl der möglichen Basen für eine gegebene endliche abelsche Gruppe.

1.11. Lemma *Sei $n \in \mathbb{N}$. Ein Element $a \in \mathbb{Z}/n\mathbb{Z}$ ist genau dann ein Erzeuger von $\mathbb{Z}/n\mathbb{Z}$, wenn gilt:*

$$\mathrm{ggT}(a, n) = 1.$$

Wir identifizieren hier wie üblich ein Element der Restklassengruppe $\mathbb{Z}/n\mathbb{Z}$ mit dem kanonischen Vertreter dieser Äquivalenzklasse aus dem Intervall $[\![0, n-1]\!]$; siehe [Be90].

Beweis:

$$\begin{aligned} \mathrm{ggT}(a,n) = 1 &\iff \min\{l : l \in \mathbb{N}, l \cdot a = k \cdot n \text{ für ein } k \in \mathbb{N}\} = n \\ &\iff \mathrm{ord}_{\mathbb{Z}/n\mathbb{Z}}(a) = n \\ &\iff \; < a >_{\mathbb{Z}/n\mathbb{Z}} = \mathbb{Z}/_{n\mathbb{Z}}. \end{aligned}$$

■

1.12. Beispiel Sei $G = \mathbb{Z}/6\mathbb{Z} \times \mathbb{Z}/24\mathbb{Z} = \{(a,b) : a \in \mathbb{Z}/6\mathbb{Z}, b \in \mathbb{Z}/24\mathbb{Z}\}$. Die Struktur ist $(6, 24)$, eine Basis ist $((1,0),(0,1))$, eine andere Basis ist $((5,0),(0,13))$ (denn $\mathrm{ggT}(5,6) = 1 \Rightarrow \mathrm{ord}_{\mathbb{Z}/6\mathbb{Z}}\, 5 = 6$, analog für $13 \in \mathbb{Z}/24\mathbb{Z}$, siehe Lemma 1.11).

Wir benötigen im weiteren Verlauf noch

1.13. Lemma *Sei G eine zyklische Gruppe, $n \in \mathbb{N}, n \mid |G|$. Dann gibt es*

1. *n Elemente, deren Ordnung n teilt und*
2. *$\phi(n)$ Elemente mit Ordnung n,*

wobei $\phi(n) = |\{a : a \in [\![0, n-1]\!], \mathrm{ggT}(a,n) = 1\}|$ die Eulersche Phi-Funktion ist.

Beweis: Zu 1.: Sei δ ein erzeugendes Element von G. Betrachte den kanonischen Isomorphismus

$$\begin{aligned} \phi_\delta : G &\longrightarrow \mathbb{Z}/_{|G|\mathbb{Z}} \\ \gamma &\longmapsto g \text{ definiert durch } g\delta = \gamma. \end{aligned}$$

Für

$$\begin{aligned} \psi_n : G &\longrightarrow G \\ \gamma &\longmapsto n\gamma \end{aligned}$$

gilt

$$\mathrm{Kern}\ \psi_n = \{\gamma : \gamma \in G, \mathrm{ord}(\gamma) | n\}.$$

ϕ_d ist bijektiv, also gilt:

$$|\,\mathrm{Kern}\ \psi_n\,| = |\phi_d(\mathrm{Kern}\ \psi_n)|.$$

Da

$$\begin{aligned} \phi_d(\mathrm{Kern}\ \psi_n) &= \left\{ g : g \in \mathbb{Z}/_{|G|\mathbb{Z}}, n \cdot g = k \cdot |G| \text{ für ein } k \in \mathbb{N} \right\} \\ &= \left\{ g : g \in \mathbb{Z}/_{|G|\mathbb{Z}}, g = k \cdot \frac{|G|}{n} \right\} \quad \text{, da } n \mid |G| \\ &= \left\{ k \cdot \frac{|G|}{n} : k \in [\![0, n-1]\!] \right\} \\ &\cong \mathbb{Z}/_{n\mathbb{Z}}, \end{aligned}$$

folgt $\{\gamma \in G : \mathrm{ord}(\gamma) | n\} = n$. Damit ist 1. bewiesen.

Zu 2.:
$$\{\gamma : \gamma \in G, \operatorname{ord}(\gamma) = n\} \subseteq \{\gamma : \gamma \in G, \operatorname{ord}(\gamma) | n\} \cong \mathbb{Z}/_{n\mathbb{Z}},$$

also

$$\begin{aligned}\{\gamma : \gamma \in G, \operatorname{ord}(\gamma) = n\} &\cong \left\{a : a \in \mathbb{Z}/_{n\mathbb{Z}}, \operatorname{ord}(a) = n\right\} \\ &\cong \left\{a : a \in \mathbb{Z}/_{n\mathbb{Z}}, \operatorname{ggT}(a, n) = 1\right\},\end{aligned}$$

nach Lemma 1.11. Somit ist auch 2. bewiesen. ■

Nun können wir folgenden Satz formulieren:

1.14. Satz *Es sei G eine endliche abelsche Gruppe mit Struktur $(s_1, \ldots, s_m)$. Dann gibt es*

$$\prod_{i=1}^{m} \phi(s_i)$$

viele Basen von G.

Beweis: Nach Satz 1.7 gibt es $\delta_1, \ldots, \delta_m \in G$ mit

$$G \cong \bigoplus_{i=1}^{m} < \delta_i > \ \text{mit} \ < \delta_i > \cong \mathbb{Z}/_{s_i\mathbb{Z}}, \quad s_1 | \cdots | s_m, \quad s_1 > 1$$

und nach Lemma 1.11 gibt es $\phi(s_i)$ viele Erzeuger von $\mathbb{Z}/s_i\mathbb{Z}$. Also gilt:

$$\begin{aligned}\left|\left\{(\delta_1, \ldots, \delta_m) : \bigvee_{i \in [\![1,m]\!]} < \delta_i > \cong \mathbb{Z}/_{s_i\mathbb{Z}}\right\}\right| &= \prod_{i=1}^{m} \left|\left\{\delta_i : < \delta_i > \cong \mathbb{Z}/_{s_i\mathbb{Z}}\right\}\right| \\ &= \prod_{i=1}^{m} \phi(s_i).\end{aligned}$$

■

Zum Schluß noch ein

1.15. Lemma *Sei $n \in \mathbb{N}, n = \prod_{i=1}^{m} {p_i}^{e_i}, p_1, \ldots, p_m \in \mathbb{P}$. Dann gilt:*

$$\phi(n) = \prod_{i=1}^{m} (p_i - 1) {p_i}^{e_i - 1} = n \cdot \prod_{i=1}^{m} \left(1 - \frac{1}{p_i}\right).$$

Beweis: Siehe z.B. [Ko87, S. 20]. ■

Kapitel 2

Algorithmische Voraussetzungen

In diesem Kapitel geben wir die Voraussetzungen an, unter denen wir die in dieser Arbeit formulierten Algorithmen und die entsprechenden Sätze auf Gruppen anwenden können und illustrieren unser Modell mit den Beispielen $\mathbb{F}_n^*$, einer elliptischen Kurve und einer Klassengruppe über einem imaginärquadratischen Zahlkörper.

2.1 Ein algorithmisches Modell

Wir betrachten als Modell unendliche Familien $\mathcal{G}$ von endlichen abelschen Gruppen G_i mit aufsteigender Ordnung:

$$\mathcal{G} = \{G_i\}_{i\in\mathbb{N}}, \quad |G_i| \le |G_{i+1}|,$$

für alle $G \in \mathcal{G}$ soll gelten:

1. es gibt eine injektive Darstellungsabbildung $D_G : G \longrightarrow [\![0, n_G]\!]$, $n_G = |G|^{O(1)}$,
2. es gibt eine Abbildung $+_G : [\![0, n_G]\!]^2 \longrightarrow [\![0, n_G]\!]$, für die gilt:
$$\underset{\alpha,\beta\in G}{\forall} \; +_G(D_G(\alpha), D_G(\beta)) = D_G(\alpha+\beta),$$
3. Es gibt eine Abbildung $-_G : [\![0, n_G]\!] \longrightarrow [\![0, n_G]\!]$, für die gilt:
$$\underset{\alpha\in G}{\forall} \; -_G(D_G(\alpha)) = D_G(-\alpha)$$
4. Es gibt eine surjektive Abbildung $\in_{RG} : . \longrightarrow D_G(G)$. Für $\in_{RG}() = \alpha$ schreiben wir auch $\alpha \in_R G$.

Diese vier Bedingungen formulieren die Schnittstelle von einer Realisierung der Programmierumgebung einer endlichen abelschen Gruppe zu den Algorithmen, die wir hier besprechen werden:

- Punkt 1 formuliert die Darstellung der Elemente einer gegebenen endlichen abelschen Gruppe auf einem Rechner, wobei die Größe des Bereichs, in dem alle Darstellungen von Gruppenelementen liegen, polynomiell in der Gruppenordnung sein darf, d.h. die Darstellungen der Elemente dürfen nicht zu weit auseinanderliegen. Dies ist notwendig, damit die Zeitkomplexitäten der Operationen mit Gruppenelementen, die ja mit Operationen von Zahlen in dem Intervall $[\![0, n_G]\!]$ realisiert werden, im wesentlichen Funktionen der Gruppenordnung sind, denn es gilt:
$$n_g = |G|^{O(1)} \quad \longrightarrow \quad \log n_G = O(\log|G|).$$

- Punkt 2 und 3 beschreiben die elementaren Gruppenoperationen, deren Implementation wir im Rahmen der Programmierumgebung einer gegebenen endlichen abelschen Gruppe voraussetzen, d.h. wir setzen voraus, daß wir aus zwei gegebenen Elementen α und β deren Summe $\alpha + \beta$, sowie aus einem gegebenen Element γ dessen additives Inverses $-\gamma$ mit jeweils *einer* elementaren Gruppenaddition bzw. –Inversion (als komplexitätstheoretisches Maß) berechnen können. Die Realisierung der Operationen soll subpolynomiell in der Gruppenordnung viele Bitoperationen benötigen.

- Punkt 4 charakterisiert die "zufällige" Wahl eines Elementes einer gegebenen endlichen abelschen Gruppe. Die Ausgabewahrscheinlichkeiten der probabilistischen Algorithmen $\in_{RG_i}$, die die Ausgabe eines pseudozufälligen Elementes der Gruppen $G_i \in \mathcal{G}$ generiert, soll folgende Bedingung erfüllen:

  Für jede Folge $(\gamma_i)_{i\in\mathbb{N}}$ mit $\gamma_i \in G_i$, $(G_i)_{i\in\mathbb{N}} = \mathcal{G}$ und $p(\gamma_i \in_{RG_i} G_i) < \frac{1}{|G_i|}$ soll gelten:

  $$\exists (g_i)_{i\in\mathbb{N}} : p(\gamma_i \in_{RG_i} G_i) \geq \frac{1}{|G_i|} \cdot \frac{1}{g_i} \quad mit \quad g_i(|G_i|) = |G_i|^{o(1)}.$$

  Interpretation: Bei wachsendem $|G|$ soll sich die Wahrscheinlichkeit der Ausgabe von $\gamma \in G$ der Wahrscheinlichkeit $\frac{1}{n}$ geometrisch annähern.
  Auch die pseudozufällige Wahl eines Gruppenelementes betrachten wir als eine elementare Gruppenoperation.

Wir werden Ergebnisse, die von Ausgabewahrscheinlichkeiten von Gruppenelementen abhängen, in Abhängigkeit von den "unteren Schranken" g_i angeben; wir werden im folgenden diese maximalen Abweichungen "nach unten" von der Gleichverteilung für mehrere Beispiele berechnen. In allgemeinen Sätzen ersetzen wir dann g_i durch $|G|^{o(1)}$.

Ab sofort identifizieren wir ein Gruppenelement mit seiner Darstellung; es ist zu beachten, daß in den theoretischen Überlegungen immer tatsächliche Elemente einer beliebigen Gruppe betrachtet werden, während bei der Besprechung von Algorithmen die Darstellungen von Elementen von Gruppen einer Familie $\mathcal{G}$ manipuliert werden.

Wir werden nun die algorithmische Modellierung einer endlichen abelschen Gruppe an drei Beispielen demonstrieren; die Ergebnisse späterer Kapitel dieser Arbeit können mittels der hier erarbeiteten Ergebnisse auf die behandelten Gruppen angewandt werden.

2.2 Modellierung von $\mathbb{F}_n^*$

Betrachte die multiplikative Gruppe $\mathbb{F}_n^*$:

$$F_n^* = \{i : i \in [\![1, n-1]\!], \mathrm{ggT}(i,n) = 1\}$$

- Wir können eine obere, in der Gruppenordnung $\phi(n)$ polynomielle Begrenzung des Darstellungsintervalls $n_{\mathbb{F}_n^*} := n$ angeben. Die Darstellung der Elemente ist trivial.

- Das neutrale Element $\mathbf{1}$ der Gruppe ist durch $(1 \bmod n)$ dargestellt.

- Zu einem beliebigen von $\mathbf{1}$ verschiedenen Element $a \in \mathbb{F}_n^*$ kann man das (multiplikative) Inverse $\frac{1}{a}$ folgendermaßen berechnen: man berechnet den ggT d von a und n sowie u, v, so daß $d = u \cdot a + v \cdot n$. Wegen $d = 1$ gilt dann $u \cdot a \equiv 1 \bmod n$; also ist $u = \frac{1}{a}$.

- Zu zwei beliebigen Elementen $a, b \in \mathbb{F}_n^*$ ist das Produkt $(a \cdot b \bmod n)$.

- Die pseudozufällige Auswahl eines Elementes aus $\mathbb{F}_n^*$ geschieht folgendermaßen:
  Wir wählen $(\lfloor \log n \rfloor + 1)$–mal mit Hilfe eines Münzwurfprimitivs (MWP) eine 0 oder eine 1. Die durch die Bitfolge binär codierte Zahl ist eine zufällige Zahl aus
  $$[\![0, 2^{\lfloor \log n \rfloor + 1} - 1]\!].$$
  Falls diese Zahl nun nicht in $[\![0, n-1]\!]$ liegt oder nicht prim zu n ist, so beginne das Verfahren von vorne. Nach einer festen Anzahl $k \in \mathbb{N}$ von erfolglosen Versuchen gebe **1** aus.

2.1. Lemma *Für oben beschriebenes Verfahren zur pseudozufälligen Erzeugung von einem Element $\gamma \in \mathbb{F}_n^*$ mit $p(\gamma \in_R \mathbb{F}_n^*) < \frac{1}{\phi(n)}$ gilt:*
$$\exists (g_n)_{n \in \mathbb{N}} : p(\gamma \in_R \mathbb{F}_n^*) = \frac{1}{\phi(n)} \cdot \frac{1}{g_n}$$
mit
$$g_n(|F_n^*|) = 1 - \left(1 - \frac{\phi(n)}{2^{\lfloor \log n \rfloor + 1}}\right)^k = \phi(n)^{o(1)}.$$

Beweis: Wir haben ein Münzwurfprimitiv zur Verfügung, d.h. wir haben einen Algorithmus, der uns mit Wahrscheinlichkeit $\frac{1}{2}$ eine 0 bzw. 1 ausgibt. Eine durch eine $\lfloor \log n \rfloor + 1$ lange Bitfolge kodierte Zahl, wobei die Bits mit einem MWP erzeugt wurden, tritt infolgedessen mit Wahrscheinlichkeit
$$\frac{1}{2^{\lfloor \log n \rfloor + 1}}$$
auf. Die Wahrscheinlichkeit, mit diesem Verfahren eine Zahl zu erzeugen, die außerhalb von $[\![0, n-1]\!]$ liegt oder nicht relativ prim zu n ist, beträgt
$$\frac{2^{\lfloor \log n \rfloor + 1} - \phi(n)}{2^{\lfloor \log n \rfloor + 1}} = 1 - \frac{\phi(n)}{2^{\lfloor \log n \rfloor + 1}}.$$
Also beträgt die Wahrscheinlichkeit, in k Versuchen ein Element $\gamma \in \mathbb{F}_n^* \setminus \{\mathbf{1}\}$ zu erzeugen,
$$\begin{aligned}
p(\gamma \in_R \mathbb{F}_n^*) &= \sum_{i=0}^{k-1} \frac{1}{2^{\lfloor \log n \rfloor + 1}} \cdot \left(1 - \frac{\phi(n)}{2^{\lfloor \log n \rfloor + 1}}\right)^i \\
&= \frac{1}{2^{\lfloor \log n \rfloor + 1}} \cdot \frac{\left(1 - \frac{\phi(n)}{2^{\lfloor \log n \rfloor + 1}}\right)^k - 1}{1 - \frac{\phi(n)}{2^{\lfloor \log n \rfloor + 1}}} \\
&= \frac{1}{\phi(n)} \cdot \left(\underbrace{1 - \left(1 - \frac{\phi(n)}{2^{\lfloor \log n \rfloor + 1}}\right)^k}_{=\frac{1}{g}} \right).
\end{aligned}$$
Um nun
$$\lim_{\phi(n) \to \infty} \frac{\log g}{\log \phi(n)}$$
zu untersuchen, betrachte folgende Überlegung: Wie man sich leicht überlegen kann, ist $\frac{\phi(n)}{n}$ am kleinsten, wenn n von der Form
$$n = \prod_{i=1}^{k} p_i$$

ist, wobei $p_1, \ldots, p_k$ die ersten k Primzahlen sind. In diesem Fall gilt nach dem Fundamentalsatz der Arithmetik

$$\phi(n) = \pi(n),$$

wobei $\pi(n) = \{p : p \in [\![2, n]\!], p \text{ prim}\}$. Nach dem Primzahlsatz gilt nun

$$\lim_{n\to\infty} \pi(n) - \left(\frac{n}{\log n}\right) = 0,$$

und somit ist $\frac{c}{\log n}$ mit $c \in \mathbb{R}$ eine untere Schranke für $\frac{\phi(n)}{n}$. Somit gilt:

$$\begin{aligned}
\lim_{n\to\infty} \frac{g}{\log \phi(n)} &\leq \lim_{n\to\infty} \frac{-\log\left(1-\left(1-\frac{\phi(n)}{n}\right)^k\right)}{\log \phi(n)} \\
&\leq \lim_{n\to\infty} \frac{-\log\left(1-\left(1-\frac{1}{\log n}\right)^k\right)}{\log n} \\
&= \lim_{n\to\infty} \frac{-\frac{k\cdot\left(1-\frac{1}{\log n}\right)^{k-1}\cdot\frac{1}{\log^2 n\cdot n}}{1-\left(1-\frac{1}{\log n}\right)^k}}{\frac{1}{n}} \quad \text{mit l'Hopital} \\
&= \lim_{n\to\infty} \frac{-\frac{k}{\log n}}{\frac{1}{\left(1-\frac{1}{\log n}\right)^{k-1}} - 1 + \frac{1}{\log n}} \\
&= \lim_{n\to\infty} \frac{-k}{\frac{\log n}{\left(1-\frac{1}{\log n}\right)^{k-1}} - 1 + 1} \\
&= 0.
\end{aligned}$$

Für $\mathbf{1} \in \mathbb{F}_n^*$ gilt:

$$\begin{aligned}
p(\mathbf{1} \in_R \mathbb{F}_n^*) &= \sum_{i=0}^{k-1} \left(\frac{1}{2^{\lfloor \log n \rfloor + 1}} \cdot \left(1 - \frac{\phi(n)}{2^{\lfloor \log n \rfloor + 1}}\right)^i\right) + \left(1 - \frac{\phi(n)}{2^{\lfloor \log n \rfloor + 1}}\right)^k \\
&> \frac{1}{|\mathbb{F}_n^*|},
\end{aligned}$$

braucht also nicht betrachtet zu werden. Also gilt $g(\phi(n)) = \phi(n)^{o(1)}$. ∎

Fundamentalsatz der Arithmetik und Primteilersatz finden sich z.B. in [Bu88].

2.2. Satz *Für die oben beschriebene Modellierung der Gruppe $\mathbb{F}_n^*$ gilt:*

- *die Berechnung des Inversen eines Elementes benötigt $O(\log^2 \phi(n))$ Bitoperationen,*
- *die Berechnung des Produkts zweier Elemente benötigt $O(\log^2 \phi(n))$ Bitoperationen,*
- *die pseudozufällige Erzeugung eines Elementes benötigt $O(\log^2 \phi(n))$ Bitoperationen und $O(\log n)$ Anwendungen des MWP.*

Beweis: Die Berechnung des Inversen mit dem erweiterten euklidischen Algorithmus benötigt $O(\log^2 n)$ Bitoperationen, ebenso der Test auf Zugehörigkeit zur Gruppe einer Zahl in $[\![0, n-1]\!]$ (ebenfalls euklidischer Algorithmus). Das Produkt zweier Zahlen sowie die Reduktion modulo n benötigt ebenfalls diese Komplexität. ∎

2.3 Die Modellierung einer elliptischen Kurve

Sei K ein Körper, $a, b \in K$ mit $4a^3 + 27b^2 \neq 0$. Die Menge der K-rationalen Punkte einer **elliptischen Kurve** $E(K) = \{(x, y) \mid x, y \in K \text{ mit } y^2 = x^3 + ax + b\} \cup \{(\infty, \infty)\}$ bildet eine abelsche Gruppe. Siehe hierzu [Si86], [Mü91]. Betrachte den Spezialfall $K = \mathbb{F}_p$.

- Eine obere Begrenzung des Darstellungsintervalls ist offensichtlich $p^2 + 1$; da
$$\left| |E| - (p+1) \right| \leq 2\sqrt{p},$$
ist diese Begrenzung sicherlich polynomiell in $|G|$. Anstatt eine lineare Darstellung zu benutzen, bietet es sich hier an, eine (natürlichere) zweidimensionale Darstellung zu wählen.

- Das neutrale Element der Gruppe ist der Punkt $= (\infty, \infty)$.

- Zu einem Punkt $\neq P = (x, y) \in E(K)$ ist das additive Inverse
$$-P = (x, -y).$$
Dies läßt sich in $O(\log p)$ Bitoperationen berechnen.

- Für Punkte P_1 und P_2 mit $P_1 = -P_2$ gilt:
$$P_1 + P_2 = .$$

- Für Punkte $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ mit $P_1 \neq P_2$ gilt $P_1 + P_2 = (x_3, y_3)$ mit
$$\begin{aligned} x_3 &= -x_1 - x_2 + \lambda^2 \\ y_3 &= -y_1 + \lambda(x_1 - x_3), \end{aligned}$$
wobei
$$\lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{falls } P_1 \neq \pm P_2 \\[2ex] \dfrac{3x_1^2 - a}{2y_1} & \text{falls } P_1 = -P_2. \end{cases}$$
Der Aufwand für die Berechnung der Summe von zwei beliebigen Gruppenelementen ist maximal 12 arithmetische Operationen mit Zahlen aus $\mathbb{F}_p$ — davon zwei Berechnungen des multiplikativen Inversen in $\mathbb{F}_p$ —, also $O(\log^2 p)$ Bitoperationen.

- Die pseudozufällige Wahl eines Gruppenelementes sieht folgendermaßen aus: wähle zufällig ein $x \in \mathbb{F}_p$ (wir gehen hier davon aus, eine Zahl aus $[\![0, p-1]\!]$ zufällig wählen zu können) und berechne $x^3 + ax + b$. Ist diese Zahl ein Quadrat in $\mathbb{F}_p$, so berechne die beiden Wurzeln und wähle zufällig einen der beiden Werte, y, aus. Der pseudozufällig erzeugte Punkt ist dann (x, y). Ist $x^3 + ax + b$ kein Quadrat in $\mathbb{F}_p$, so beginne von vorn. Nach einer festen Anzahl t von erfolglosen Versuchen setze $(x, y) := (\infty, \infty)$.

Wir schreiben ab hier kurz E für $E(\mathbb{F}_p)$.

2.3. Lemma *Sei $p \in \mathbb{P}$, E eine elliptische Kurve über $\mathbb{F}_p$. Unter der Annahme, daß man zufällig Zahlen aus $[\![0, p-1]\!]$ wählen kann, gilt für oben beschriebenes Verfahren zur pseudozufälligen Erzeugung von Elementen aus E für alle Punkte $(x, y) \in E$ mit $p((x, y) \in_R E) < \frac{1}{|E|}$:*

$$\exists (g_p)_{p \in \mathbb{P}} : p((x, y) \in_R E) = \frac{1}{|E|} \cdot \frac{1}{g_p} \quad \textit{mit} \quad g_p(|E|) = |E|^{o(1)}.$$

Beweis: Bezeichne $Q(x)$ das Prädikat $x^3 + ax + b$ *ist ein Quadrat in* $\mathbb{F}_p$ *und verschieden von* $0_{\mathbb{F}_p}$, $R(x)$ das Prädikat $x^3 + ax + b = 0_{\mathbb{F}_p}$. Bezeichne

- k die Anzahl der $x \in \mathbb{F}_p$, für die $Q(x)$ gilt,
- c die Anzahl der $x \in \mathbb{F}_p$, für die $R(x)$ gilt und
- l die Anzahl der $x \in \mathbb{F}_p$, für die $x^3 + ax + b$ kein Quadrat in $\mathbb{F}_p$ ist.

Es gilt:

$$k + c + l = p \quad \Longrightarrow \quad l = p - k - c.$$

Nach dem Satz von Hasse (siehe [Si86, S.131ff]) gilt:

$$|E| = p + 1 + (k - l) \quad \text{mit} \quad |k - l| \leq 2\sqrt{p};$$

wir bezeichnen, aus praktischen Gründen, $k - l$ mit z. Es folgt hieraus:

$$p = |E| - 1 - z \quad \text{und} \quad k = \frac{1}{2}\left(|E| - 1 - c\right).$$

Sei $x \in \mathbb{F}_p$. Dann gilt nach unserer Voraussetzung:

$$p(x \in_R \mathbb{F}_p) = p(x \in_R [\![0, p-1]\!]) = \frac{1}{p};$$

somit ist

$$p(Q(x)) = \sum_{Q(x)} p(x \in_R \mathbb{F}_p) = \frac{k}{p}.$$

Aufgrund einer analogen Berechnung gilt:

$$p(R(x)) = \frac{c}{p}.$$

Sei $q := p(Q(x)) + p(R(x)) = \frac{k+c}{p}$.

Betrachten wir nun den Algorithmus zur pseudozufälligen Erzeugung von Punkten auf einer elliptischen Kurve. Ein Versuch ist genau dann erfolgreich, wenn für das in $\mathbb{F}_p$ pseudozufällig gewählte x entweder $Q(x)$ oder $R(x)$ gilt. Hieraus ergibt sich die Ausgabewahrscheinlichkeit des Algorithmus für einen festen Punkt $(x, y) \in E$ (betrachte zuerst den Fall $y \neq 0$):

$$\begin{aligned}
p((x,y) \in_R E, y \neq 0) &= \sum_{i=0}^{t-1} (1-q)^i \cdot \frac{1}{p} \cdot \frac{1}{2} \\
&= \frac{1}{2p}\left(\frac{1 - (1-q)^t}{q}\right) \\
&= \frac{1}{2(k+c)} \cdot \left(1 - (1-q)^t\right) \\
&= \frac{1}{|E|} \cdot \underbrace{\frac{|E| \cdot (1 - (1-q)^t)}{2(k+c)}}_{=\frac{1}{g}}.
\end{aligned}$$

Nun gilt:

$$2(k+c) = 2\left(\frac{1}{2}(|E| - 1 - c) + c\right) = |E| - 1 + c \leq |E| + 2,$$

da c gerade die Anzahl der Nullstellen von $x^3 + ax + b$ in $\mathbb{F}_p$ ist und

$$|E| \cdot \left(1 - (1-q)^t\right) \geq |E| \cdot \left(1 - \left(1 - \frac{1}{4}\right)^t\right),$$

für $p > 16$, denn $|k - l| \leq 2\sqrt{p}$ und somit $k + c \geq \frac{p}{2} - \sqrt{p}$, was größer ist als $\frac{p}{4}$, falls $p > 16$. Also haben wir:

$$\lim_{|E|\to\infty} g(|E|) = \frac{1}{\left(1 - \left(1 - \frac{1}{4}\right)^t\right)} < c \in \mathbb{R},$$

woraus folgt:

$$\lim_{|E|\to\infty} \frac{\log g(|E|)}{\log E|} = 0.$$

Für den Fall $y = 0$ gilt:

$$\begin{aligned} p((x,y) \in_R E, y = 0) &= \sum_{i=0}^{t-1} (1-q)^i \cdot \frac{1}{p} \\ &= \frac{1}{k+c} \cdot \frac{1}{g} > p((x,y) \in_R E, y \neq 0). \end{aligned}$$

nach einer zu obigem Fall analogen Rechnung. Für (∞, ∞) gilt:

$$\begin{aligned} p((\infty,\infty) \in_R E) &= \sum_{i=0}^{t} (1-q)^i \\ &= \frac{p}{k+c} \cdot \left(1 - (1-q)^{t+1}\right) \\ &> \frac{1}{|E|} \end{aligned}$$

und somit gilt die obige Bedingung sowieso. ■

2.4. Bemerkung Wir verwenden eine Voraussetzung, die wir nicht erfüllen können, nämlich

$$p(x \in_R [\![0, p-1]\!]) = \frac{1}{p}$$

für alle $x \in [\![0, p-1]\!]$ (im vorherigen Abschnitt haben wir gezeigt, daß dies auch mit Verwendung eines Münzwurfprimitivs für $p \neq 2^l$, $l \in \mathbb{N}$, nicht möglich ist). Trotzdem liefert Lemma 2.3 eine Aussage über die Güte des Verfahrens zur pseudozufälligen Erzeugung von Punkten einer elliptischen Kurve, die für praktische Überlegungen mit einer angemessenen Rundenzahl k ausreichend ist.

Der Aufwand, ein pseudozufälliges Element zu erzeugen, beträgt $O(\log^2 p)$ arithmetische Operationen mit Zahlen aus $\mathbb{F}_p$, also $O(\log^4 p)$ Bitoperationen und $O(\log p)$ Anwendungen des MWP. Somit gilt:

2.5. Satz *Sei $p \in \mathbb{P}$, E eine elliptische Kurve über $\mathbb{F}_p$. Für die oben beschriebene Modellierung der Gruppe E gilt:*

- *die Berechnung des additiven Inversen eines Elementes benötigt $O(\log^2 p)$ Bitoperationen,*
- *die Berechnung der Summe zweier Elemente benötigt $O(\log^2 p)$ Bitoperationen,*
- *die pseudozufällige Erzeugung eines Elementes benötigt $O(\log^4 p)$ Bitoperationen und $O(1)$ zufällige Erzeugungen einer Zahl aus $[\![0, p-1]\!]$.*

Bei den Komplexitätsbetrachtungen werden wir davon ausgehen, daß die zufällige Erzeugung einer Zahl aus einem Intervall 1. möglich ist und 2. keinen Aufwand kostet.

2.4 Die Modellierung einer Klassengruppe

Als letztes Beispiel wollen wir Klassengruppen über imaginärquadratischen Zahlkörpern betrachten. Betrachte **binäre quadratische Formen**

$$f(X,Y) = aX^2 + bXY + cY^2 \in \mathbb{Z}[X,Y]$$

und deren **Diskriminanten**

$$\Delta = b^2 - 4ac.$$

Wir betrachten nur **positiv definite** (d.h. $\Delta < 0$ und $a > 0$) und **indefinite** (d.h. $\Delta > 0$) binäre quadratische Formen, die außerdem **primitiv** sind (d.h. $\mathrm{ggT}(a,b,c) = 1$) und für deren Diskriminante gilt $\Delta \equiv 0, 1 \bmod 4$, Δ kein Quadrat.

$$\mathrm{QF}(\Delta)$$

bezeichne alle binären quadratischen Formen mit Diskriminante Δ.
Zwei binäre quadratische Formen f, g heißen **äquivalent**, wenn es eine Matrix $A \in \mathbb{Z}^{2\times 2}$ mit $\det A = 1$ gibt, so daß $f((A(X,Y)^T)^T) = g(X,Y)$. Die dadurch definierte Menge der Äquivalenzklassen $\{[f] : f \in \mathrm{QF}(\Delta)\}$ auf $\mathrm{QF}(\Delta)$ ist eine Gruppe, heißt **Klassengruppe** und wird mit

$$\mathrm{Cl}(\Delta)$$

bezeichnet; die Gruppenordnung $|\mathrm{Cl}(\Delta)|$ heißt **Klassenzahl** und wird mit h bezeichnet. Eine binäre quadratische Form heißt **reduziert**, wenn $|b| \leq a \leq c$ und $b > 0$ falls $|b| = a$ oder $a = c$. Für imaginärquadratische Zahlkörper gilt: es gibt genau einen reduzierten Vertreter einer Äquivalenzklasse von binären quadratischen Formen.

- Für genügend großes Δ gilt:

  $$\sqrt{|\Delta|} \leq h \leq 1/3 \cdot \sqrt{|\Delta|} \cdot \log|\Delta|.$$

  Wir haben also eine in der Gruppenordnung polynomielle obere Abschätzung für die Gruppenordnung (siehe [Dü92, S. 39]); auch hier bietet sich eine zweidimensionale Darstellung $[(a,b)]$ der Gruppenelemente durch die reduzierten Vertreter (a,b) der Äquivalenzklassen an (c ist durch a, b und Δ eindeutig definiert).

- Ein Vertreter (a_3, b_3) des Produkts zweier Elemente $[(a_1,b_1)]$ und $[(a_2,b_2)]$ (in einer Klassengruppe schreibt man die Verknüpfung multiplikativ) läßt sich wie folgt berechnen:
  Berechne

  $$d = \mathrm{ggT}\left(a_1, a_2, \frac{b_1+b_2}{2}\right),$$

  sowie v_1, v_2, w aus der Darstellung

  $$d = v_1a_1 + v_2a_2 + w\frac{b_1+b_2}{2};$$

  dann gilt:

  $$a_3 = \frac{a_1a_2}{d^2}, \quad b_3 = v_2b_1\frac{a_2}{d} + v_1b_2\frac{a_1}{d} + w\frac{b_1b_2+\Delta}{2d}.$$

  Schließlich berechne noch den reduzierten Vertreter der Äquivalenzklasse von (a_3, b_3); siehe hierzu [Dü92].

- Das Inverse eines Elementes $[(a,b)]$ ist $[(a,-b)]$.

- Ein Vertreter des neutralen Elements ist $(1,0)$.

- Die Erzeugung eines pseudozufälligen Elementes benötigt etwas mehr Aufwand:
  Sei $p \in \mathbb{P}, \Delta \in \mathbb{N}$. Die Zahl

$$\left(\frac{\Delta}{p}\right) = \begin{cases} 1 & \text{falls } \Delta \text{ Quadrat in } \mathbb{Z}/4p\mathbb{Z} \text{ ist und } p \nmid \Delta, \\ 0 & \text{falls } p|\Delta \text{ und} \\ -1 & \text{sonst} \end{cases}$$

  heißt Kroneckersymbol. Es kann berechnet werden in $O(\log^3 p)$ Bitoperationen. Für $\left(\frac{\Delta}{p}\right) = 1$ kann man

$$b_p = \min\{b : b \in \mathbb{N},\, b^2 \equiv \Delta \bmod (4p)\}$$

  in $O(\log^4 p)$ Bitoperationen unter Zuhilfenahme des Ressol-Algorithmus berechnen. Eine quadratische Form (p, b_p) heißt **Primform** in $\mathrm{QF}(\Delta)$. Da (unter Annahme der Verallgemeinerten Riemannschen Vermutung)

$$\left\{[f_p] : f_p \text{ Primform in } \mathrm{QF}(\Delta), p < 12\log^2|\Delta|\right\} = ([f_{p_1}], \ldots, [f_{p_k}])$$

  ein Erzeugendensystem der Klassengruppe $\mathrm{Cl}(\Delta)$ ist (siehe [Dü92, S.37f]), kann man pseudozufällig Vertreter $f \in \mathrm{QF}(\Delta)$

$$f = \prod_{i=1}^{k} {f_{p_i}}^{e_i}$$

  der Elemente der Klassengruppe durch zufällige Wahl der Exponenten

$$(e_1, \ldots, e_k) \in [\![0, B-1]\!]^k$$

  erzeugen mit $B = \sqrt{|\Delta|} \cdot \log|\Delta|$.

2.6. Satz *Unter der Annahme, daß man zufällig Zahlen aus dem Intervall* $[\![0, B-1]\!]$ *wählen kann und der Verallgemeinerten Riemannschen Vermutung gilt für oben beschriebenes Verfahren zur pseudozufälligen Erzeugung von Elementen einer Klassengruppe* $\mathrm{Cl}(\Delta)$ *für alle Elemente* $[f] \in \mathrm{Cl}(\Delta)$*:*

$$\exists (g_\Delta)_{\Delta \in \mathbb{Z}^-} : ([f] \in_R \mathrm{Cl}(\Delta)) = \frac{1}{|\mathrm{Cl}(\Delta)|} \cdot \frac{1}{g_\Delta} \quad \textit{mit} \quad g_\Delta(h) = h^{o(1)}.$$

Beweis: Betrachte folgende Abbildung:

$$\begin{array}{cccc} \Phi : & \mathbb{Z}^k & \longrightarrow & \mathrm{Cl}(\Delta) \\ & (e_1, \ldots, e_k) & \longmapsto & \left[\prod_{i=1}^{k} {f_{p_i}}^{e_i}\right] \end{array}$$

Unter Annahme der Verallgemeinerten Riemannschen Vermutung ist $([f_{p_1}], \ldots, [f_{p_k}])$ ein Erzeugendensystem von $\mathrm{Cl}(\Delta)$, somit Φ surjektiv und es gilt folglich:

$$\mathbb{Z}^k / \mathrm{Kern}\, \Phi \cong \mathrm{Cl}(\Delta).$$

Kern Φ ist ein Gitter im $\mathbb{Z}^k$, somit gilt:

$$\det(\mathrm{Kern}\, \Phi) = \left| \mathbb{Z}^k / \mathrm{Kern}\, \Phi \right| = |\mathrm{Cl}(\Delta)| = h.$$

Nun gilt folgender

2.7. Satz *Sei Λ ein Gitter im $\mathbb{Z}_k$ mit Determinante h. Dann gilt für alle $w \in \mathbb{Z}^k$ und für alle $B \geq h$:*

$$\frac{1}{h}\left(1-\frac{h-1}{B}\right) \leq \frac{\left|\bar{w}^B\right|}{B^k} \leq \frac{1}{h}\left(1+\frac{h-1}{B}\right),$$

wobei

$$\bar{w}^B = \left\{z : z \in [\![0, B-1]\!]^k, z \equiv w \bmod \Lambda\right\}.$$

Dieser Satz und sein Beweis finden sich (in einer erweiterten Version) in [Le87, 4.].

Hiermit können wir nun unsere Behauptung beweisen:

$$\begin{aligned}
p([f] \in_R \mathrm{Cl}(\Delta)) &= p(\Phi(w) \in_R \mathrm{Cl}(\Delta), \Phi(w) = [f]) \\
&= \sum_{z \in \bar{w}^B} p(z \in_R [\![0, B-1]\!]^k) \\
&= \sum_{z \in \bar{w}^B} \prod_{i=1}^{k} \frac{1}{B} \\
&= \sum_{z \in \bar{w}^B} \frac{1}{B^k}. \\
&= \frac{\left|\bar{w}^B\right|}{B^k}.
\end{aligned}$$

Da gilt

$$\sqrt{|\Delta|} \leq h \leq \sqrt{|\Delta|} \cdot \log|\Delta|,$$

und da zur Anwendung von Satz 2.7 $B \geq h$ gelten muß, setzen wir

$$B = \sqrt{|\Delta|} \cdot \log^2|\Delta|;$$

dann folgt:

$$\left|\frac{h-1}{B}\right| \leq \frac{\sqrt{|\Delta|} \cdot \log|\Delta|}{\sqrt{|\Delta|}\log^2|\Delta|} = \frac{1}{\log|\Delta|}$$

und, mit $g_\Delta = \frac{\log|\Delta|}{\log|\Delta|-1}$ gilt dann für alle $[f] \in \mathrm{Cl}(\Delta)$:

$$p([f] \in_R \mathrm{Cl}(\Delta)) \geq \frac{1}{h} \cdot \frac{1}{g_\Delta} \quad \text{mit} \quad g_\Delta(h) = h^{o(1)},$$

denn es gilt:

$$\begin{aligned}
\lim_{h\to\infty} \frac{\log g_\Delta}{\log h} &= \lim_{h\to\infty} \frac{\log \frac{\log|\Delta|}{\log|\Delta|-1}}{\log h} \\
&= \lim_{h\to\infty} \frac{\log|\Delta| - \log|\Delta| + 1}{\log h} \\
&= \lim_{h\to\infty} \frac{1}{\log h} \\
&= 0.
\end{aligned}$$

Wir haben also unsere Behauptung bewiesen. ■

Für den Aufwand der Modellierung einer Klassengruppe über einem imaginärquadratischen Zahlkörper gilt:

2.8. Satz *Sei* $\Delta \in \mathbb{N}$. *Für die oben beschriebene Modellierung der Gruppe* $\mathrm{Cl}(\Delta)$ *gilt:*

- *die Berechnung eines Vertreters des Inversen eines Elementes benötigt* $O(1)$ *Bitoperationen,*
- *die Berechnung eines Vertreters des Produktes zweier Elemente benötigt* $O(\log^2|\Delta| \cdot \log\log|\Delta| \cdot \log\log\log|\Delta|)$ *Bitoperationen,*
- *die pseudozufällige Erzeugung eines Elementes benötigt* $O(\sqrt{|\Delta|} \cdot \log^5|\Delta| \cdot \log\log|\Delta| \cdot \log\log\log|\Delta|)$ *Bitoperationen, wenn ein Erzeugendensystem der Gruppe aus Primformen vorliegt.*

Beweis: Siehe hierzu [Le87], [Co91, S. 154ff], [Dü92, S. 37ff]. ■

In vielen Algorithmen, die wir besprechen werden, benötigen wir einen Test, ob ein gegebenes Element einer Gruppe das neutrale Element der Gruppe ist. Mit der Berechnung der Addition zweier Gruppenelemente steht uns auch dieser Test zur Verfügung: man addiert das Element zu sich selbst und vergleicht das Ergebnis mit dem betrachteten Element — nur das neutrale Element einer Gruppe erfüllt die Gleichheit. Manchmal benötigen wir auch das neutrale Element selbst; es steht uns trivialerweise dadurch zur Verfügung, daß wir zu einem beliebigen Element das additive Inverse berechnen und hinzuaddieren. Beide Operationen sind in einer konstanten Anzahl von Gruppenoperationen möglich.

Kapitel 3

Die Bestimmung der Ordnung eines Elementes

In diesem Kapitel geht es um die Berechnung der Ordnung eines Elementes einer endlichen abelschen Gruppe; hierzu besprechen wir zuerst die schnelle Multiplikation. Danach stellen wir den Algorithmus von Shanks zur Bestimmung der Ordnung eines Elementes vor und untersuchen einige Varianten, die unter verschiedenen Voraussetzungen funktionieren.

3.1 Schnelle Multiplikation

Sei $G \in \mathcal{G}$, $\gamma \in G$, $n \in \mathbb{N}$. Es kommt oft vor, daß wir $n\gamma$ berechnen wollen. Anstatt naiv $n-1$ Gruppenoperationen auszuführen, können wir die Idee der schnellen Exponentiation verwenden. Schreibe n in binärer Darstellung:

$$n = \sum_{i=0}^{d-1} n_i 2^i \quad \text{mit } d = \lfloor \log n \rfloor + 1 \text{ und } n_1, \ldots, n_{d-1} \in \{0, 1\}.$$

Dann gilt:

$$n\gamma = \sum_{i=0}^{d-1} n_i (2^i \gamma),$$

und die $2^i\gamma$ lassen sich schnell berechnen:

$$\gamma, \gamma + \gamma, (\gamma + \gamma) + (\gamma + \gamma), \ldots$$

d.h. für jede Zweierpotenz benötigen wir nur eine zusätzliche Gruppenoperation, wenn wir die vorherige kennen.

3.1. Beispiel Betrachte

$$\begin{aligned}
&10\gamma = 1(8\gamma) + 0(4\gamma) + 1(2\gamma) + 0(1\gamma) \\
&\gamma \xrightarrow{1} (\gamma + \gamma) \xrightarrow{2} (\gamma + \gamma) + (\gamma + \gamma) \xrightarrow{3} ((\gamma + \gamma) + (\gamma + \gamma)) + ((\gamma + \gamma) + (\gamma + \gamma)) \\
&2\gamma \xrightarrow{4} 2\gamma + 8\gamma,
\end{aligned}$$

also 4 Operationen statt 9.

Wir haben damit folgenden

3.2. Algorithmus (schnelle Multiplikation)

Eingabe: $n \in \mathbb{Z}$, $\gamma \in G$ $(G \in \mathcal{G})$.

Ausgabe: $n\gamma$.

$\mu \leftarrow$		(1)
IF $n < 0$ THEN $n \leftarrow -n$; $\gamma \leftarrow -\gamma$		(2)
WHILE $n \neq 0$		(3)
	IF$(n \bmod 2) = 1$ THEN $\mu \leftarrow \mu + \gamma$	(4)
	$n \leftarrow n \div 2$; $\gamma \leftarrow \gamma + \gamma$	(5)
RETURN $\{\mu\}$		(6)

Zur Korrektheit und Laufzeit des Programms betrachte man folgenden

3.3. Satz *Sei $G \in \mathcal{G}$, $\gamma \in G$, $n \in \mathbb{N}$. Mit schneller Multiplikation kann man $n\gamma$ in*

- $O(\log n)$ *Gruppenoperationen und*
- $O(\log^2 n)$ *Bitoperationen*

berechnen.

Beweis: In Algorithmus 3.2 werden in jedem Schleifendurchlauf

- eine Berechnung $n \bmod 2$ – diese geht in konstanter Anzahl von Bitoperationen,
- eine ganzzahlige Division von $n \div 2$ – diese geht in $O(\log n)$ Bitoperationen und
- maximal 2 Gruppenoperationen

durchgeführt; nach dem i-ten Schleifendurchlauf gilt offensichtlich:

$$\mu^{(i)} = (n \bmod 2^i)\gamma,$$

nach $\lfloor log_2 n \rfloor + 1$ Schleifendurchläufen gilt also:

$$\mu^{(\lfloor log_2 n \rfloor + 1)} = (n \bmod 2^{\lfloor log_2 n \rfloor + 1})\gamma = n\gamma.$$

Daraus folgt die Behauptung. ■

Wenn man das Inverse eines Elementes "schnell" (d.h. erheblich schneller als eine Addition) berechnen kann, besteht eine praktische Verbesserung dieses Algorithmus darin, lange Ketten von Einsen in der Binärzerlegung des Multiplikators n, also viele aufeinanderfolgende Additionen, durch eine Subtraktion zu ersetzen. Beispiel:

$$15\gamma = 2^3\gamma + 2^2\gamma + 2^1\gamma + 2^0\gamma = 2^4\gamma - 2^0\gamma.$$

Obwohl diese Idee keine größenordnungsmäßige Verbesserung bringt, erreicht man damit eine relative Laufzeitverbesserung von z.B. 11 Prozent bei elliptischen Kurven, wenn man die Anzahl der Gruppenoperationen betrachtet. Siehe hierzu [Mü91, S. 52f] und [MoOl90].

3.2 Der Baby-Step-Giant-Step-Algorithmus

Sei G eine endliche abelsche Gruppe. Zur Berechnung der Ordnung eines Elementes $\gamma \in G$ verwende folgende Idee von Shanks [Sh70]:

- $< \gamma >$ ist zyklisch, d.h. $n\gamma = (n \bmod \operatorname{ord}\gamma)\gamma$.
- Berechne
$$, \quad \gamma, \quad 2\gamma, \quad \ldots, \quad (m-1)\gamma,$$
die Menge der **Baby-Steps**, vor (falls eines dieser Elemente, $i\gamma$, schon die Null der Gruppe ist, sind wir fertig und die Ordnung von γ ist i)
- und dann nacheinander
$$m\gamma, \quad 2(m\gamma), \quad \ldots, \quad k(m\gamma),$$
die **Giant-Steps**, bis das gerade berechnete Element $km\gamma$ in der Menge der Baby-Steps liegt.

Für dieses $k \in \mathbb{N}$ gilt dann:

$$\exists_{n \in [\![0, m-1]\!]} \qquad km\gamma = n\gamma;$$

also

$$km\gamma - n\gamma = (km - n)\gamma \Longleftrightarrow \operatorname{ord}\gamma | km - n.$$

Da dieses k das kleinste $k \in \mathbb{N}$ ist, das dies erfüllt und da $n < m < \operatorname{ord}\gamma$, gilt

$$\operatorname{ord}\gamma = km - n.$$

Die Anzahl der Gruppenoperationen zur Berechnung von $\operatorname{ord}\gamma$ beträgt $m + k$, andererseits gilt $\operatorname{ord}\gamma \leq m \cdot k$. Kennt man nun eine obere Schranke L der Ordnung von γ, so nimmt man an, die obere Schranke wäre eine gute Approximation der Ordnung von γ. Unter dieser Annahme minimiert man den Aufwand, wenn man die Anzahl der zu berechnenden Baby- und Giant-Steps nahe bei $\sqrt{L}$ wählt. Das folgende Lemma gibt Auskunft, wie man diese genau wählen soll:

3.4. Lemma *Seien $L, q, r \in \mathbb{N}$. Dann gilt:*

$$\max\left\{g : g = q \cdot \lceil\sqrt{L}\rceil - r, \quad 0 < r \leq \lceil\sqrt{L}\rceil, \quad 1 \leq q \leq \lfloor\sqrt{L}\rfloor + 1\right\} \quad \geq \quad L.$$

Beweis: Sei

$$g_{max} = \max\left\{g : g = q \cdot \lceil\sqrt{L}\rceil - r, \quad 0 < r \leq \lceil\sqrt{L}\rceil, \quad 1 \leq q \leq \lfloor\sqrt{L}\rfloor + 1\right\}.$$

$$g_{max} \quad = \quad \left(\lfloor\sqrt{L}\rfloor + 1\right)\lceil\sqrt{L}\rceil - 1.$$

1. Fall: $\sqrt{L} = \lfloor\sqrt{L}\rfloor = \lceil\sqrt{L}\rceil$. Dann ist:

$$g_{max} = L + \sqrt{L} - 1 \geq L.$$

2. Fall: $\sqrt{L} = \lfloor\sqrt{L}\rfloor + d, \quad d \in (0,1)$. In diesem Fall ist:

$$\begin{aligned} g_{max} &= \left(\sqrt{L} - d + 1\right)\left(\sqrt{L} + (1-d)\right) - 1 \\ &= \left(\sqrt{L} + (1-d)\right)^2 - 1, \end{aligned}$$

und da $g_{max} \in \mathbb{N}$ sowie

$$\left(\sqrt{L} + (1-d)\right)^2 > L \quad \text{für } d \in (0,1),$$

folgt die Behauptung. ■

Wir haben mit Lemma 3.4 unter der Kenntnis einer oberen Schranke L die exakten Werte für die Parameter m und k bestimmt, um möglichst effizient die Baby-Step-Giant-Step-Idee zur Berechnung der Ordnung eines Elementes anzuwenden. Dieses Lemma und die vorhergehende Betrachtung liefern die Korrektheit des folgenden Algorithmus:

3.5. Algorithmus (Baby-Step-Giant-Step mit oberer Schranke)

Eingabe: $\gamma \in G$ $(G \in \mathcal{G})$, $L \in \mathbb{N}$ obere Schranke für $\operatorname{ord}\gamma$.

Ausgabe: $\operatorname{ord}\gamma$.

$m \leftarrow \lceil\sqrt{L}\rceil, \quad \gamma_0 \leftarrow$			(1)
FOR $i \leftarrow 1$ TO m DO		Baby-Steps	(2)
	$\gamma_i \leftarrow \gamma_{i-1} + \gamma$		(3)
	IF $\gamma_i =$ THEN RETURN $\{i\}$		(4)
$\gamma^1 \leftarrow \gamma_m$			(5)
FOR $j \leftarrow 2$ TO $\lfloor\sqrt{L}\rfloor + 1$		Giant-Steps	(6)
	$\gamma^j \leftarrow \gamma^{j-1} + \gamma^1$		(7)
	IF $\gamma^j = \gamma_i$ für ein $i \in [\![0, m-1]\!]$ THEN RETURN $\{j \cdot m - i\}$		(8)

Zur Laufzeit des Algorithmus:

Da wir voraussetzen, daß L eine gute Approximation für die Ordnung von γ ist, beschränken wir uns auf eine *worst-case*-Analyse, die die Anzahl der benötigten Operationen in Abhängigkeit von L aufzeigt; ist L eine sehr grobe Approximation, so terminiert der Algorithmus wesentlich früher. Wir werden später einen Algorithmus kennenlernen, der eine Abschätzung in Abhängigkeit von $\operatorname{ord}\gamma$ erlaubt.

Es werden jeweils $O(\sqrt{L})$ Gruppenoperationen bei den Baby-Steps wie bei den Giant-Steps durchgeführt. Um die Baby-Steps so sortiert in einem Baum abzuspeichern, um nach jedem Giant-Step schnell suchen zu können benötigt man $O(\log\sqrt{L}) = O(\log L)$ Vergleiche von Zahlen aus $[\![0, n_G]\!]$ in jedem Baby-Step. Den Baum der sortiert gespeicherten Baby-Steps zu durchforsten, um sie mit dem aktuellen Giant-Step zu vergleichen, benötigt $O(\log\sqrt{L}) = O(\log L)$ Vergleiche von Zahlen aus $[\![0, n_G]\!]$ in jedem Giant-Step, also insgesamt $O(\sqrt{L} \cdot \log L \cdot \log n_G)$ Bitoperationen. Es folgt

3.6. Satz *Sei $G \in \mathcal{G}$. Dann läßt sich die Ordnung eines Elementes mit Kenntnis einer oberen Schranke L in*

- $O(\sqrt{L})$ *Gruppenoperationen und*
- $O(\sqrt{L} \cdot \log L \cdot \log n_G)$ *Bitoperationen*

berechnen.

3.7. Bemerkung Die Idee der schnellen skalaren Multiplikation, wie sie in Abschnitt 3.1 besprochen wurde, ist hier nicht von Nutzen, da jedes errechnete Gruppenelement entweder zu späteren Vergleichen benötigt oder aber selbst mit früher errechneten Elementen verglichen wird.

3.3 Baby-Step-Giant-Step ohne obere Schranke

Wenn man keine obere Schranke der Ordnung eines Elementes kennt, so wendet man die Baby-Step-Giant-Step-Methode sukzessive auf wachsende $L \in \mathbb{N}$ an, von denen man vermutet, daß sie eine obere Schranke der gesuchten Ordnung bilden. Man erhöht L so lange, bis Algorithmus 3.5 terminiert. In jeder neuen Anwendung von Algorithmus 3.5 wählt man die neue obere Schranke so groß wie möglich, ohne "Information zu verlieren". Dies ist dann der Fall, wenn die neue obere Schranke gerade das Quadrat der alten oberen Schranke ist, denn dann weiß man, daß keiner der Baby-Steps in der neuen Runde den Algorithmus terminieren wird (sonst hätte der Algorithmus ja in der vorherigen Runde schon terminiert).

3.8. Algorithmus (Baby-Step-Giant-Step ohne obere Schranke)

Eingabe: $\gamma \in G$ ($G \in \mathcal{G}$).

Ausgabe: $\operatorname{ord} \gamma$.

Setze L_0 auf einen Anfangswert		(1)
Starte Algorithmus 3.5 mit den Eingaben γ und L_0. Falls dieser erfolgreich terminiert und n zurückgibt, RETURN { n }.		(2)
$L \leftarrow {L_0}^2$		(3)
WHILE TRUE		(4)
	Starte Algorithmus 3.5 mit den Eingaben γ und L, wobei in der Baby-Step-Schleife die berechneten Elemente nicht mehr auf getestet werden und in der vorherigen "Runde" berechnete Gruppenelemente nicht mehr neu berechnet werden. Falls er erfolgreich terminiert und n zurückgibt, RETURN { n }.	(5)
	$L \leftarrow L^2$	(6)

Wie soll man den Anfangswert L_0 der oberen Schranke wählen?

Die Maximalanzahl $T(\gamma)$ der Gruppenoperationen, um die Ordnung eines Gruppenelementes $\gamma \in G$ zu berechnen, ergibt sich wie folgt:

Wir bezeichnen mit "Runde" eine Anwendung von Algorithmus 3.5. Sei $j \in \mathbb{N}$, so daß $\operatorname{ord} \gamma$ in der j-ten Runde berechnet wurde, also

$$ {L_0}^{2^{j-1}} \quad < \quad \operatorname{ord} \gamma \quad \leq \quad {L_0}^{2^j}, $$

wobei die 0-te Runde in Schritt 2 durchgeführt wird. Dann ist

$$ \begin{aligned} T(\gamma) &= \sum_{i=0}^{j} O\left(\sqrt{{L_0}^{2^i}}\right) \\ &= O\left(\sum_{i=0}^{j} {L_0}^{2^{i-1}}\right) \\ &= O\left((j+1) \cdot {L_0}^{2^{j-1}}\right) \\ &= O((j+1) \cdot \operatorname{ord} \gamma). \end{aligned} $$

Man sieht hieraus: es ist erstrebenswert, die Rundenanzahl so klein wie möglich zu halten. Dies entspricht auch der Philosophie der Baby-Step-Giant-Step-Idee: sie erweist sich dann als besonders wirkungsvoll, wenn man eine gute Annäherung an die gesuchte Ordnung kennt. Für den allgemeinen Fall kann man somit keine Aussage über das zu wählende L_0 machen. Allerdings empfiehlt es sich, diese Anfangsschranke so zu wählen, daß alle auftretenden Schranken "rechnerfreundlich" sind; wir setzen infolgedessen L_0 auf 4. Dann gilt:

$$2^{2^j} \quad < \quad \operatorname{ord}\gamma \quad \leq \quad 2^{2^{j+1}};$$

es werden also $\lceil \log\log \operatorname{ord}\gamma \rceil - 1$ viele Runden durchlaufen, in der i-ten Runde werden $O(2^{2^i})$ Gruppenoperationen ausgeführt. Wir haben also den ersten Teil des folgenden Satzes bewiesen:

3.9. Satz *Es sei $G \in \mathcal{G}$,*

$$\begin{array}{rcl} f : \mathbb{N} \setminus \{1\} & \longrightarrow & \mathbb{N} \\ x & \longmapsto & \displaystyle\sum_{i=0}^{\lceil \log\log x \rceil - 1} 2^{2^i}, \\ g : \mathbb{N} \setminus \{1\} & \longrightarrow & \mathbb{N} \\ x & \longmapsto & \displaystyle\sum_{i=0}^{\lceil \log\log x \rceil - 1} 2^{2^i} \cdot 2^i. \end{array}$$

Dann kann man die Ordnung eines beliebigen Elementes $\gamma \in G$ in

- *$O(f(\operatorname{ord}\gamma))$ Gruppenoperationen und*
- *$O(g(\operatorname{ord}\gamma) \cdot \log n_G)$ Bitoperationen*

berechnen.

Teil 2 des Satzes sieht man folgendermaßen ein: in der i-ten Runde, also bei dem Aufruf von Algorithmus 3.5 mit der oberen Schranke $2^{2^{i+1}}$, benötigt man, wie die Analyse von Algorithmus 3.5 zeigt,

$$O\left(\sqrt{2^{2^{i+1}}} \cdot \log \sqrt{2^{2^{i+1}}} \cdot \log n_G\right) = O\left(2^{2^i} \cdot 2^i \cdot \log n_G\right).$$

Hieraus folgt unmittelbar Teil 2 der Behauptung.

Um das Verhalten der Funktionen f und g einzusehen, betrachte das folgende

3.10. Lemma *Es gilt für die oben definierte Funktion f:*

$$\forall_{n \in \mathbb{N} \setminus \{1\}} \quad \sum_{i=1}^{\lceil \log\log n \rceil} n^{1/2^i} \quad \leq \quad f(n) \quad < \quad \sum_{i=0}^{\lceil \log\log n \rceil - 1} n^{1/2^i}.$$

Beweis: Wir zeigen folgende Behauptung durch vollständige Induktion über i; durch Aufsummieren ergibt sich dann das Lemma.

$$\forall_{i \in \mathbb{N} \setminus \{1\}} \quad n^{1/2^i} \quad \leq \quad 2^{2^{\lceil \log\log n \rceil - i}} \quad < \quad n^{1/2^{i-1}}$$

Der Induktionsanfang ($i = 1$) ergibt sich wie folgt:
Es gilt trivialerweise

$$2^{2^{\lceil \log\log n \rceil - 1}} \quad < \quad n \quad \leq \quad 2^{2^{\lceil \log\log n \rceil}};$$

wenn wir überall die Wurzel ziehen, gilt:

$$2^{2^{\lceil \log\log n \rceil - 2}} \quad < \quad \sqrt{n} \quad \leq \quad 2^{2^{\lceil \log\log n \rceil - 1}},$$

woraus die Behauptung für $i = 1$ durch Zusammensetzen der Ungleichungen folgt.

Sei die Behauptung für alle $j \leq i$ bereits bewiesen. Dann gilt:

$$n^{1/2^i} \quad \leq \quad 2^{2^{\lceil \log\log n \rceil - i}} \quad < \quad n^{1/2^{i-1}};$$

durch Wurzelziehen bekommen wir die Induktionsbehauptung:

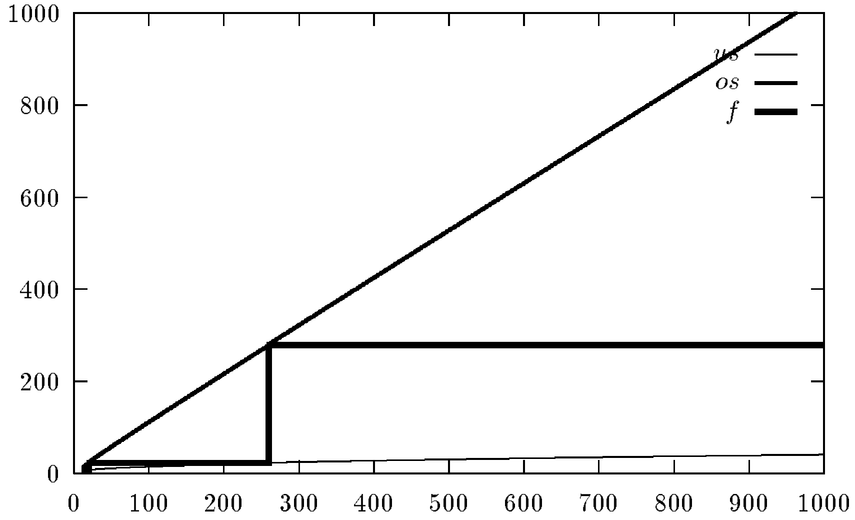
$$n^{1/2^{i+1}} \quad \leq \quad 2^{2^{\lceil \log\log n \rceil - i - 1}} \quad < \quad n^{1/2^i}. \qquad \blacksquare$$

Die angegebenen Schranken für f sind scharf; d.h. es gibt keine besseren Schranken (wie sich aus der Beweisführung leicht ersehen läßt). Man kann hieraus sofort ableiten:

3.11. Korollar *Für oben definierte Funktion f gilt:*

$$\bigvee_{n \in \mathbb{N} \setminus \{1\}} f(n) > \sqrt{n}.$$

Da die scharfen Schranken sehr wenig anschaulich sind, geben wir hier die Graphen der Funktion f und der unteren und oberen Schranken an:



Hierbei bezeichnet *us* die untere, *os* die obere Schranke von f aus obigem Lemma.

Da beim Vergrößern der Schrittweite der von Algorithmus 3.8 benötigte Aufwand einen großen Sprung macht, erscheint es ungünstig, die Schrittweite quadratisch zu vergrößern. Wir geben nun einen Algorithmus an, der die Schrittweite "nur" verdoppelt.

3.12. Algorithmus (Baby-Step-Giant-Step ohne obere Schranke (2))

Eingabe: $\gamma \in G$ $(G \in \mathcal{G})$.

Ausgabe: $\operatorname{ord} \gamma$.

Setze L_0 auf einen Anfangswert	(1)
Starte Algorithmus 3.5 mit den Eingaben γ und L_0. Falls dieser erfolgreich terminiert und n zurückgibt, RETURN { n }.	(2)
$L \leftarrow 2 \cdot L_0$	(3)
WHILE TRUE	(4)
Starte Algorithmus 3.5 mit den Eingaben γ und L, wobei in der Baby-Step-Schleife die berechneten Elemente nicht mehr auf getestet werden und in der vorherigen "Runde" berechnete Gruppenelemente nicht mehr neu berechnet werden. Falls er erfolgreich terminiert und n zurückgibt, RETURN { n }.	(5)
$L \leftarrow 2 \cdot L$	(6)

3.13. Satz *Sei $G \in \mathcal{G}$, $\gamma \in G$. Mit obigem Algorithmus kann man die Ordnung von γ in*

- $O(\sqrt{\operatorname{ord}\gamma})$ *Gruppenoperationen und*
- $O(\sqrt{\operatorname{ord}\gamma} \cdot \log \operatorname{ord}\gamma \cdot \log n_G)$ *Bitoperationen*

berechnen.

Beweis: Wir bezeichnen mit "Runde" wieder einen Aufruf von Algorithmus 3.5. Sei $j \in \mathbb{N}$, so daß $\operatorname{ord}\gamma$ in der j-ten Runde berechnet wird; d.h.

$$L_0 \cdot 2^{j-1} \quad < \quad \operatorname{ord}\gamma \quad \leq \quad L_0 \cdot 2^j,$$

also

$$j = \lceil \log \operatorname{ord}\gamma - \log L_0 \rceil.$$

Sei $T(\gamma)$ die Anzahl der benötigten Gruppenoperationen. In der i–ten Iteration werden $O(\sqrt{L_0 \cdot 2^i})$ Gruppenoperationen benötigt, also gilt:

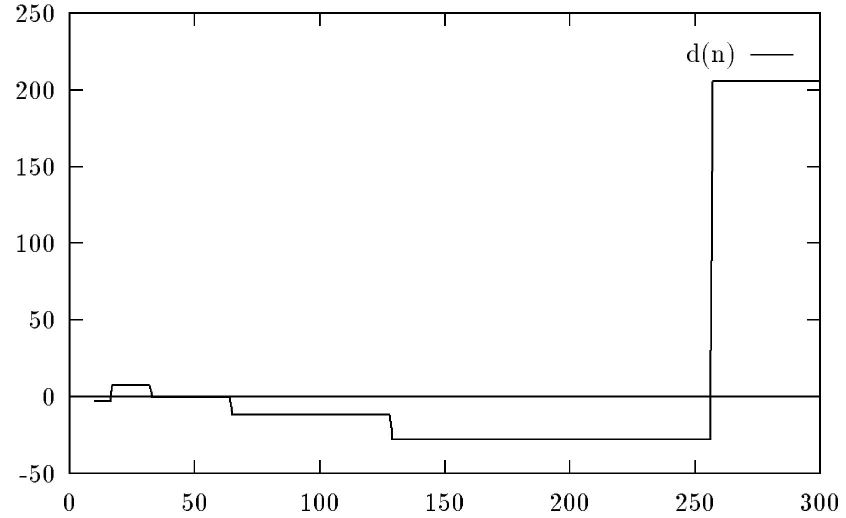
$$\begin{aligned}
T(\gamma) &= \sum_{i=0}^{j} O\left(\sqrt{L_0 \cdot 2^i}\right) \\
&= O\left(\sum_{i=0}^{\lceil \log \operatorname{ord}\gamma - \log L_0 \rceil} \sqrt{L_0} \cdot \left(\sqrt{2}\right)^i\right) \\
&= O\left(\sqrt{L_0} \cdot \frac{\sqrt{2}^{\lceil \log \operatorname{ord}\gamma - \log L_0 \rceil + 1} - 1}{\sqrt{2} - 1}\right) \\
&= O\left(\left(\sqrt{2}\right)^{\log \operatorname{ord}\gamma - \log L_0 + 2}\right) \\
&= O\left(2^{\frac{\log \operatorname{ord}\gamma}{2}}\right) \\
&= O\left(\sqrt{\operatorname{ord}\gamma}\right)
\end{aligned}$$

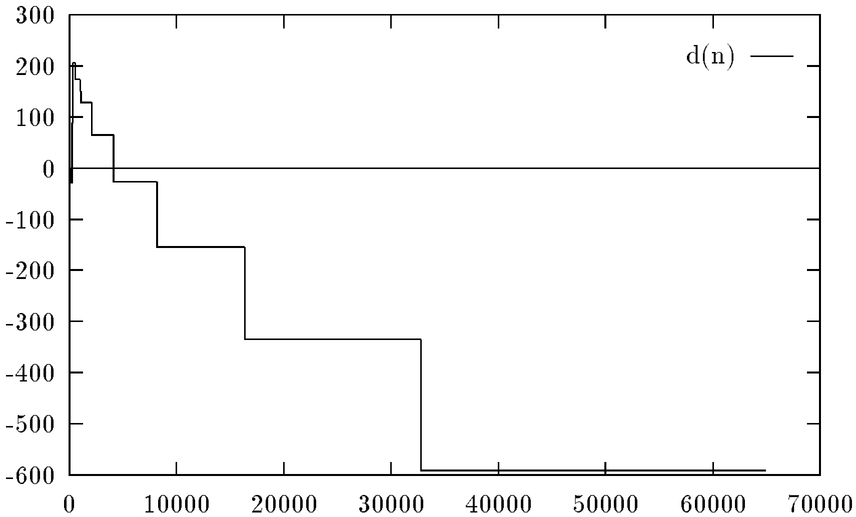
Die Größenordnung der Anzahl an benötigten Bitoperationen ergibt sich mit einer analogen Rechnung (siehe auch den Beweis zu Satz 3.6). ■

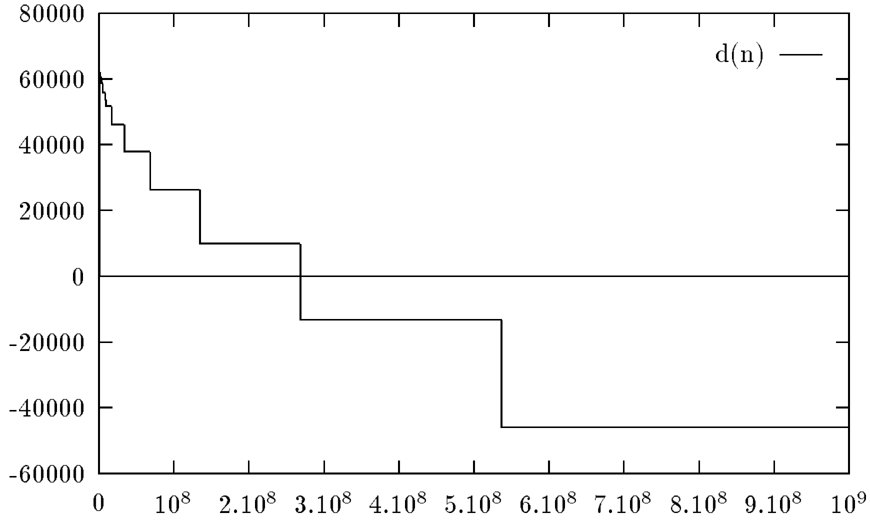
Wir haben mit Korollar 3.11 gezeigt, daß die Anzahl der benötigten Gruppenoperationen für den Baby-Step-Giant-Step-Algorithmus mit Quadrierung der oberen Schranke echt größer als $\sqrt{\operatorname{ord}\gamma}$ ist. Man ist geneigt, aus dieser Betrachtung den Schluß zu ziehen, die Alternative mit *Verdoppeln* sei generell besser. Es zeigt sich aber, daß die beiden Funktionen in einem sehr großen endlichen Bereich so nahe beieinander liegen, daß für die Praxis keine Aussage gemacht werden kann. Zur Illustration dieses Sachverhalts hier einige Ausschnitte des Graphs der Differenz des Aufwandes mit Quadrieren der oberen Schranke und des Aufwandes mit Verdoppeln der oberen Schranke:

$$d(n) = \sum_{i=0}^{\lceil \log\log n\rceil - 1} 2^{2^i} - \sum_{i=0}^{\lceil \log n\rceil - 2} 2 \cdot 2^i;$$

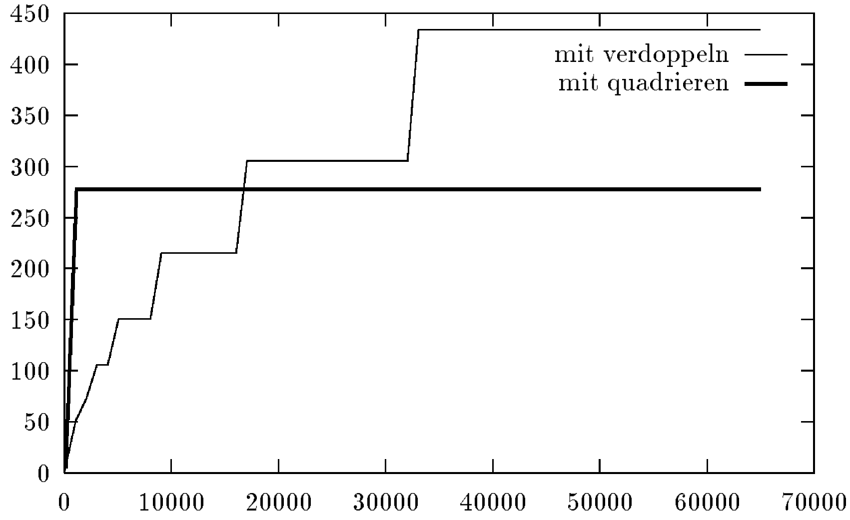
beide Varianten mit Startwert $L_0 = 4$.







Das sprunghafte Verhalten tritt z.B. an der folgenden Stelle auf (am Ende des dargestellten Bereichs macht der Aufwand mit Quadrieren wieder einen großen Sprung nach oben, genauer bei $65536 = 2^{16}$):



Einerseits scheint die Variante mit Verdoppeln öfter mehr Aufwand zu benötigen als die Variante mit Quadrieren; andererseits sieht es so aus, daß wenn die Variante mit Quadrieren mehr Aufwand benötigt, dann benötigt sie *viel mehr* Aufwand.

Es ist in Anbetracht dieser Situation a priori nicht zu entscheiden, welche Variante "besser" ist. Vielmehr müßte man nun die Vor- und Nachteile der beiden Varianten bei einer speziellen Implementation berücksichtigen. Da dies nicht Gegenstand unserer Arbeit ist, verzichten wir hier darauf.

3.4 Baby-Step-Giant-Step mit Intervall

Wenn man nun nicht nur eine obere, sondern auch eine untere Schranke für die Ordnung eines Elementes kennt, so benötigen wir wesentlich weniger Aufwand, um die Ordnung dieses Elementes zu bestimmen.

Wir wenden die Baby-Step-Giant-Step-Idee mit folgender Modifikation an: da wir wissen, daß

$$l \quad \leq \quad \operatorname{ord}\gamma \quad \leq \quad L,$$

durchqueren wir nur das Intervall $[l, L]$ mit Giant-Steps der Größe $O(\sqrt{L-l})$ und berechnen demzufolge nur $O(\sqrt{L-l})$ viele Baby-Steps vor. Wir haben folgenden Algorithmus:

3.14. Algorithmus (Baby-Step-Giant-Step mit Intervall)

Eingabe: $\gamma \in G$ ($G \in \mathcal{G}$), L obere Schranke für $\operatorname{ord}\gamma$, l untere Schranke für $\operatorname{ord}\gamma$.

Ausgabe: $\operatorname{ord}\gamma$.

$m \leftarrow \lceil\sqrt{L-l}\rceil, \quad \gamma_0 \leftarrow$			(1)
FOR $i \leftarrow 1$ TO m		Baby-Steps	(2)
	$\gamma_i \leftarrow \gamma_{i-1} + \gamma$		(3)
	IF $\gamma_i =$ THEN RETURN $\{i\}$		(4)
$\gamma^0 \leftarrow l\gamma$ mit Algorithmus 3.2 (Schnelle Multiplikation)			(5)
IF $\gamma^0 =$ THEN RETURN $\{l\}$			(6)
FOR $j \leftarrow 1$ TO $\lfloor\sqrt{L-l}\rfloor + 1$		Giant-Steps	(7)
	$\gamma^j \leftarrow \gamma^{j-1} + \gamma_m$		(8)
	IF $\gamma^j = \gamma_i$ für ein $i \in [\![0, m-1]\!]$ THEN RETURN$\{l + jm - i\}$		(9)
RETURN {nicht gefunden}			(10)

Die Korrektheit des Algorithmus ist direkt einzusehen, wenn man die Argumente für die Korrektheit des Algorithmus mit oberer Schranke hier analog anwendet. Da man $l\gamma$ berechnen muß, erhöht sich der Aufwand um die notwendigen Operationen für die schnelle Multiplikation, mit der $l\gamma$ berechnet wird. Wir haben folgenden

3.15. Satz *Sei $G \in \mathcal{G}$. Dann läßt sich die Ordnung eines Elementes mit Kenntnis einer oberen Schranke L und einer unteren Schranke l in*

- $O(\sqrt{L-l} + \log l)$ *Gruppenoperationen und*
- $O(\sqrt{L-l} \cdot \log(L-l) \cdot \log n_G + \log^2 l)$ *Bitoperationen*

berechnen.

3.5 Paralleler Baby-Step-Giant-Step

Nun wollen wir noch eine mögliche Parallelisierung des Baby-Step-Giant-Step-Algorithmus betrachten. Sei n die Anzahl der Baby-(bzw. Giant-)Steps. Nehmen wir an, wir hätten beliebig viele Prozessoren zur Verfügung. Eine erste Idee besteht darin, jeden Prozessor einen Baby-Step und einen Giant-Step berechnen zu lassen. Hierzu muß jeder Prozessor zwei schnelle Multiplikationen berechnen – denn er kann ja nicht wie im seqentiellen Algorithmus schon berechnete Werte benutzen – und am Ende seinen Giant-Step mit allen Baby-Steps vergleichen. Also muß jeder Prozessor $O(\log n)$ Gruppenoperationen und $O(\log n)$ Vergleichsoperationen durchführen. Hinzu kommt – wie bei dem sequentiellen Algorithmus – der Aufwand zum Sortieren von n Zahlen durch n Prozessoren; dies geht im besten Fall in $O(\log n)$.

Diese Parallelisierung erreicht keine optimale Beschleunigung; die Arbeit an Gruppenoperationen dieses Algorithmus beträgt $O(\log L \cdot \sqrt{L})$ gegenüber $O(\sqrt{L})$ im Falle des sequentiellen Algorithmus. Allerdings ist diese Version der Parallelisierung optimal beschleunigt bezüglich der Arbeit an Bitoperationen.

Wir können eine optimale Beschleunigung der Arbeit an Gruppenoperationen erreichen, indem wir $O(\frac{\sqrt{L}}{\log L})$ Prozessoren verwenden und jeder von diesen Prozessoren $O(\log L)$ viele aufeinanderfolgende Baby-Steps bzw. Giant-Steps berechnet. Dann muß jeder Prozessor eine schnelle Multiplikation und $O(\log L)$ Gruppenadditionen ausführen.

3.16. Algorithmus (Paralleler Baby-Step-Giant-Step mit oberer Schranke)

Eingabe: $\gamma \in G$ $(G \in \mathcal{G})$, L obere Schranke für $\operatorname{ord}\gamma$.

Ausgabe: $\operatorname{ord}\gamma$.

Anzahl der Prozessoren: $\left\lceil \frac{\lceil\sqrt{L}\rceil}{\log\lceil\sqrt{L}\rceil} \right\rceil$.

$m \leftarrow \lceil\sqrt{L}\rceil$				(1)
FORALL $i = 0$ TO $\lceil\frac{m}{\log m}\rceil - 1$ PARDO			Baby-Steps	(2)
	$\gamma_i \leftarrow i\lceil\log m\rceil\gamma$			(3)
	IF $\gamma_i =$ THEN RETURN $\{i\}$			(4)
	FOR $k \leftarrow 1$ TO $\lceil\log m\rceil - 1$			(5)
		$\gamma_{i+k} \leftarrow \gamma_{i+k-1} + \gamma$		(6)
		IF $\gamma_{i+k} =$ THEN RETURN $\{i+k\}$		(7)
FORALL $j = 1$ TO $\lfloor\frac{m}{\log m}\rfloor$ PARDO			Giant-Steps	(8)
	$\gamma^j \leftarrow j\lfloor\log m\rfloor(m\gamma)$			(9)
	IF $\gamma^j = \gamma_i$ für ein $i \in [\![0, m-1]\!]$ THEN RETURN$\{j\lfloor\log m\rfloor m - i\}$			(10)
	FOR $k \leftarrow 1$ TO $\lfloor\log m\rfloor + 1$			(11)
		$\gamma^{j+k} \leftarrow \gamma^{j+k-1} + m\gamma$		(12)
		IF $\gamma^{j+k} =$ THEN RETURN $\{(j\lfloor\log m\rfloor + k)m - i\}$		(13)

Es gilt:

3.17. Satz *Sei $G \in \mathcal{G}$. Dann läßt sich die Ordnung eines Elementes mit Kenntnis einer oberen Schranke L mit $O(\frac{\sqrt{L}}{\log L})$ Prozessoren in*

- *$O(\log L)$ Zeiteinheiten für Gruppenoperationen und*
- *$O(\log L \cdot \log n_G)$ Zeiteinheiten für Bitoperationen*

berechnen.

Kapitel 4

Die Berechnung von minimalen Relationen

In diesem Kapitel werden wir den für die Lösung von Problemen, die sich in endlichen abelschen Gruppen stellen, wie Berechnung der Gruppenordnung oder der Struktur einer Gruppe, elementaren Begriff der *minimalen Relation* einführen und deren Berechnung unter verschiedenen Voraussetzungen besprechen.

4.1 Minimale Relationen

4.1. Definition *Sei G eine endliche abelsche Gruppe, $\gamma_1, \ldots, \gamma_k \in G$.*
$(x_1, \ldots, x_k) \in \mathbb{Z}^k$ heißt **Relation** *für $(\gamma_1, \ldots, \gamma_k)$, wenn gilt:*

$$\sum_{i=1}^{k} x_i \gamma_i = .$$

Eine Relation $(x_1, \ldots, x_k)$ für $(\gamma_1, \ldots, \gamma_k)$ heißt **minimal***, wenn*

$$x_k = \min \{x : x \in \mathbb{N}, (x_1, \ldots, x_{k-1}, x) \textit{ Relation für } (\gamma_1, \ldots, \gamma_k)\}.$$

Die Menge der Relationen für ein Tupel von Gruppenelementen $(\gamma_1, \ldots, \gamma_k)$ bildet ein Gitter, das **Relationengitter** für $(\gamma_1, \ldots, \gamma_k)$. Das Problem, eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$ zu finden, wird auch **verallgemeinertes Problem des diskreten Logarithmus** genannt. Es kann durchaus mehrere minimale Relationen $(x_1, \ldots, x_k)$ für $(\gamma_1, \ldots, \gamma_k)$ geben; allein x_k ist durch die Minimalitätsbedingung eindeutig bestimmt.

Die Idee, wie man solche minimalen Relationen berechnen kann, ergibt sich aus folgendem

4.2. Lemma *Sei $k \geq 2$, $\gamma_1, \ldots, \gamma_k \in G$, $(x_1, \ldots, x_k)$ minimale Relation für $(\gamma_1, \ldots, \gamma_k)$ und*

$$G_k := < \gamma_1, \ldots, \gamma_{k-1} > .$$

Dann ist x_k die Ordnung von $\overline{\gamma_k}$ als Element der Faktorgruppe G/G_k.

Beweis:

$$x_k \quad = \quad \min \left\{ x : x \in \mathbb{N}, x\gamma_k = -\sum_{i=1}^{k-1} x_i \gamma_i \right\}$$

$$
\begin{aligned}
&= \min\{x : x \in \mathbb{N}, x\gamma_k \in G_k\}\\
&= \min\{x : x \in \mathbb{N}, x\overline{\gamma_k} = \mathbf{0}_{G/G_k}\}\\
&= ord_{G/G_k}\overline{\gamma_k}.
\end{aligned}
$$
■

Wir identifizieren der Einfachheit halber ab sofort $\overline{\gamma_k}$ und den kanonischen Vertreter γ_k, sagen also z.B. "Ordnung von γ_k in G/G_k".

Wir berechnen also eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$, indem wir die Ordnung x_k von γ_k in G/G_k und eine "Linearkombination"

$$\sum_{i=1}^{k-1} x_i\gamma_i = -x_k\gamma_k$$

bestimmen. Haben wir eine obere Schranke L für die Gruppenordnung $|G|$, so können wir minimale Relationen berechnen, indem wir sukzessive für ein festes x_k alle möglichen $(x_1, \ldots, x_{k-1})$ ausprobieren. Dies tun wir — wie bei der Berechnung der Ordnung eines Elementes in G — mit Hilfe der Baby-Step-Giant-Step-Idee.

4.3. Algorithmus (minimale Relation)

Eingabe: $\gamma_1, \ldots, \gamma_k \in G$ $(G \in \mathcal{G})$, L obere Schranke für $|G|$.

Ausgabe: $(x_1, \ldots, x_k)$ minimale Relation für $(\gamma_1, \ldots, \gamma_k)$.

$M \leftarrow \emptyset$		(1)
FOR $i \leftarrow 1$ TO k DO $z_i \leftarrow \lceil\sqrt{L}\rceil$		(2)
FOR $r_k \leftarrow 1$ TO z_k	(Baby-Steps)	(3)
Berechne $M_{r_k} = \left\{\left(\sum_{i=1}^{k} r_i\gamma_i, r_1, \ldots, r_k\right) : 1 \le r_i \le z_i, 1 \le i \le k-1\right\}$, sortiert nach der ersten Komponente in $D(G)$		(4)
IF $\exists(\chi, r_1, \ldots, r_k) \in \mathbb{M}_{r_k} : \chi =$		(5)
THEN RETURN $\{(r_1, \ldots, r_k)\}$		(6)
ELSE $M \leftarrow Merge(M, M_{r_k})$		(7)
FOR $q_k \leftarrow 2$ TO $\lfloor\sqrt{L}\rfloor + 1$	(Giant-Steps)	(8)
FORALL $(q_1, \ldots, q_{k-1}) \in [\![2, \lfloor\sqrt{L}\rfloor + 1]\!]^{k-1}$	(Giant-Steps)	(9)
IF $\exists(\chi, r_1, \ldots, r_k) \in M : \chi = \sum_{i=1}^{k} q_i z_i\gamma_i$		(10)
THEN RETURN $\{ (q_1z_1 - r_1, \ldots, q_kz_k - r_k) \}$		

Der Algorithmus Merge, der hier prozedural angewandt wird, mischt die beiden sortierten Mengen, die ihm übergeben werden, zu einer sortierten Menge.

4.4. Satz *Sei $G \in \mathcal{G}$, L eine obere Schranke für $|G|$, $\gamma_1, \ldots, \gamma_k \in G$. Dann kann man eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$ in*

- $O(k \cdot (\sqrt{L})^k)$ *Gruppenadditionen und*
- $O(k \cdot (\sqrt{L})^k \cdot \log L \cdot \log n_G)$ *Bitoperationen*

berechnen.

Beweis:

1. Zur Korrektheit:

   - Wenn die Ausgabe im ersten Teil (Baby-Steps) erfolgt, so ist das ausgegebene Zahlentupel trivialerweise eine Relation für $(\gamma_1, \ldots, \gamma_k)$. Wenn die Ausgabe im zweiten Teil erfolgt, so gilt

$$\begin{aligned}\sum_{i=1}^{k}(q_i z_i - r_i)\gamma_i &= \sum_{i=1}^{k} q_i z_i \gamma_i - \sum_{i=1}^{k} r_i \gamma_i \\ &= ,\end{aligned}$$

     denn die Gleichheit dieser beiden Summen ist gerade die Abbruchbedingung. Also ist die Ausgabe eine Relation für $(\gamma_1, \ldots, \gamma_k)$.

   - Für einen n-dimensionalen Vektor v bezeichnen wir mit $\Pi_j(v)$ die j-te Komponente von v. Um zu beweisen, daß die Ausgabe eine *minimale* Relation für $(\gamma_1, \ldots, \gamma_k)$ ist, zeigen wir folgende Hilfsbehauptung:

4.5. Behauptung *Hat Algorithmus 4.3 in der Baby-Step-Runde keine minimale Relation gefunden, so gilt für (r, q), $(r', q') \in M \times N_{q_k}$:*

$$\Pi_1(r) = \Pi_1(q) \wedge \Pi_1(r') = \Pi_1(q') \quad \Longrightarrow \quad \Pi_{k+1}(r) = \Pi_{k+1}(r').$$

Beweis: Nehmen wir an, es gelte $\Pi_{k+1}(r) = r_k \neq r'_k = \Pi_{k+1}(r')$; o.B.d.A. sei $r'_k > r_k$. Da die ersten Komponenten der Vektoren r (bzw. r') und q (bzw. q') gleich sind, gilt:

$$\sum_{i=1}^{k}(q'_i z_i - r'_i)\gamma_i == \sum_{i=1}^{k}(q_i z_i - r_i)\gamma_i$$

$$\Longrightarrow \sum_{i=1}^{k}((q'_i - q_i)z_i - (r'_i - r_i))\gamma_i = .$$

Da $q, q' \in N_{q_k}$, gilt $q'_k = q_k$ und somit ist

$$((q'_1 - q_1)z_1 - (r'_1 - r_1), \ldots, r'_k - r_k)$$

eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$. Da $r'_k - r_k < r'_k < z_k$, wäre diese Relation schon in der Baby-Step-Runde gefunden worden; dies widerspricht unserer Voraussetzung. □

Daß eine im Baby-Step-Teil ausgegebene Relation minimal ist, ist trivial, da man x_k dort hochzählt. Betrachten wir wieder eine Ausgabe des Giant-Step-Teils. Nehmen wir an, es gäbe eine minimale Relation $(x'_1, \ldots, x'_k)$ für $(\gamma_1, \ldots, \gamma_k)$ mit $x'_k < x_k$. Dann gibt es r'_k und q'_k mit $x'_k = q'_k z_k - r'_k$. Diese Darstellung ist eindeutig, da $r_k \in [\![1, z_k]\!]$. Analog gibt es eine eindeutige Darstellung $x_k = q_k z_k - r_k$. Nun gilt:

- Der Fall $q'_k > q_k$ widerspricht wegen $x'_k < x_k$ der eindeutigen Darstellung.
- Aus $q'_k = q_k$ folgt $r'_k \neq r_k$; dies widerspricht der obigen Hilfsbehauptung.
- Falls $q'_k < q_k$, dann wäre $(x'_1, \ldots, x'_k)$ in einem früheren Giant-Step ausgegeben worden.

Also ist eine im Giant-Step-Teil ausgegebene Relation ebenfalls minimal.

   - Um einzusehen, daß eine existierende minimale Relation auch tatsächlich ausgegeben wird, so betrachte wieder Lemma 3.4 und beobachte, daß die oberen und unteren Schranken für alle r_i und q_i den geforderten Werten entsprechen.

2. Zur Laufzeit:

   - Betrachte die Baby-Step-Schleife (die Anzahl der Operationen läßt sich für die Giant-Step-Schleife analog abschätzen). Die Berechnung der M_{r_k} kann folgendermaßen effizient realisiert werden: wir berechnen
   $$\gamma_1, 2\gamma_1, \ldots, z_k\gamma_1, \gamma_2, 2\gamma_2, \ldots, z_k\gamma_k$$
   vor; der Aufwand beträgt einmalig
   $$O\left(\prod_{i=1}^{k} z_i\right) = O\left((\sqrt{L})^k\right).$$
   . In jeder Baby-Step-Runde werden dann $k-1$ Gruppenadditionen für jedes Element aus M_{r_k}, also
   $$(k-1) \cdot \prod_{i=1}^{k-1} z_i = O(k \cdot (\sqrt{L})^{k-1})$$
   Gruppenadditionen benötigt. Da es maximal $z_k = O(\sqrt{L})$ Runden gibt, folgt der erste Teil der Behauptung.
   - Das Einsortieren und Mischen (Baby-Step-Teil) wie das Suchen (Giant-Step-Teil) benötigt für n Elemente wieder $O(n \cdot \log n)$ arithmetische Operationen. Die Größe der auftretenden Zahlen ist $O(n_G)$. Die Anzahl der Elemente ist $\prod_{i=1}^{k} z_i = O((\sqrt{L})^k)$; hieraus folgt der zweite Teil der Behauptung: es werden
   $$O((\sqrt{L})^k \cdot \log(\sqrt{L})^k \cdot \log n_G) = O(k \cdot (\sqrt{L})^k \cdot \log L \cdot \log n_G)$$
   Bitoperationen benötigt. ■

Dieses Ergebnis ist unbefriedigend; die Analyse zeigt, daß die Größe der Schranken z_i entscheidend für die Komplexität des Algorithmus ist. Wir zeigen nun, wie wir mit etwas mehr Information beträchtlich bessere Laufzeitabschätzungen mit dem gleichen Algorithmus erreichen.

4.2 Minimale Relationen mit Vorinformation

Betrachte wieder Lemma 4.2.

Setzen wir $G_1 := \{\}$, dann gilt:

4.6. Lemma *Sei $k \in \mathbb{N}$, $\gamma_1, \ldots, \gamma_k \in G$ und für $i = 1, \ldots, k-1$ sei*
$$y_i := ord_{G/G_i}\gamma_i.$$
Dann gibt es eine minimale Relation $(x_1, \ldots, x_k)$ für $(\gamma_1, \ldots, \gamma_k)$, für die gilt:
$$\forall_{i \in [\![1,k-1]\!]} \quad 0 \le x_i < y_i. \tag{4.1}$$

Beweis: Wir beweisen folgende Hilfsbehauptung:

4.7. Behauptung *Sei $l \in [\![1, k-1]\!]$, $(z_1, \ldots, z_l, x_{l+1}, \ldots, x_k)$ eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$ mit*
$$\forall_{i \in [\![l+1,k-1]\!]} \quad 0 \le x_i < y_i. \tag{4.2}$$
Dann gibt es eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$, für die (4.1) gilt.

Beweis der Hilfsbehauptung durch Induktion über l:

Induktionsanfang $l = 1$: Dividiere ganzzahlig mit Rest $z_1 = q \cdot y_1 + r$, also $q \in \mathbb{Z}$ und $r \in [\![0, y_1 - 1]\!]$. y_1 ist die Ordnung von γ_1 in $G/G_1 = G$, also gilt $y_1\gamma_1 =$ und somit:

$$z_1\gamma_1 = (q \cdot y_1 + r)\gamma_1 = q(y_1\gamma_1) + r\gamma_1 = r\gamma_1,$$

woraus folgt:

$$\begin{aligned} x_k\gamma_k &= -\left(z_1\gamma_1 + \sum_{i=2}^{k-1} x_i\gamma_i\right) \\ &= -\left(r\gamma_1 + \sum_{i=2}^{k-1} x_i\gamma_i\right). \end{aligned}$$

Also ist $(r, x_2, \ldots, x_k)$ eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$, die (4.1) erfüllt.

Induktionsschritt: Nehmen wir an, die Behauptung gelte für alle $i \in [\![1, l-1]\!]$.
Schreibe wieder $z_l = q \cdot y_l + r$ mit $q \in \mathbb{Z}$ und $r[\![0, y_l - 1]\!]$.
y_l ist die Ordnung von γ_l in G/G_l, also gibt es $w_1, \ldots, w_{l-1} \in \mathbb{N}$, so daß

$$(w_1, \ldots, w_{l-1}, y_l) \text{ minimale Relation für } (\gamma_1, \ldots, \gamma_l),$$

woraus folgt:

$$y_l\gamma_l = -\sum_{i=1}^{l-1} w_i\gamma_i.$$

Es gilt also:

$$\begin{aligned} x_k\gamma_k &= -\left(\sum_{i=1}^{l-1} z_i\gamma_i + z_l\gamma_l + \sum_{i=l+1}^{k-1} x_i\gamma_i\right) \\ &= -\left(\sum_{i=1}^{l-1} z_i\gamma_i + (q \cdot y_l + r)\gamma_l + \sum_{i=l+1}^{k-1} x_i\gamma_i\right) \\ &= -\left(\sum_{i=1}^{l-1} z_i\gamma_i + q(y_l\gamma_l) + r\gamma_l + \sum_{i=l+1}^{k-1} x_i\gamma_i\right) \\ &= -\left(\sum_{i=1}^{l-1} z_i\gamma_i + q\left(-\sum_{i=1}^{l-1} w_i\gamma_i\right) + r\gamma_l + \sum_{i=l+1}^{k-1} x_i\gamma_i\right) \\ &= -\left(\sum_{i=1}^{l-1}(z_i - q \cdot w_i)\gamma_i + r\gamma_l + \sum_{i=l+1}^{k-1} x_i\gamma_i\right). \end{aligned}$$

Für die Relation $(z_1 - q \cdot w_1, \ldots, z_{l-1} - q \cdot w_{l-1}, r, x_{l+1}, \ldots, x_k)$ trifft nun (4.2) für $l-1$ zu, also können wir die Induktionsvoraussetzung anwenden. □

Das Lemma folgt nun aus der Hilfsbehauptung für $l = k - 1$. ■

Wir können nun mit dem prinzipiell gleichen Algorithmus unter Kenntnis der Ordnungen von γ_i in G/G_i für $i = 1, \ldots, k-1$ eine viel bessere Laufzeitabschätzung erreichen. Auch für γ_k wird das durchsuchte Intervall wesentlich kleiner, wie die beiden folgenden Lemmata belegen:

4.8. Lemma *Sei G eine endliche abelsche Gruppe, $m \in \mathbb{N}$, $\gamma_1, \ldots, \gamma_m \in G$, sei*

$$G_1 = \{\} \text{ und } G_i = <\gamma_1, \ldots, \gamma_{i-1}> \text{ für } i = 2, \ldots, m$$

sowie

$$y_i = \operatorname{ord}_{G/G_i} \gamma_i \quad \text{für } i = 1, \ldots, m.$$

Dann gilt:

$$|<\gamma_1, \ldots, \gamma_m>| = \prod_{i=1}^{m} y_i.$$

Beweis: Betrachte die Abbildung

$$\begin{array}{rccc} \phi: & \mathbb{Z}^m & \longrightarrow & <\gamma_1, \ldots, \gamma_m> \\ & (n_1, \ldots, n_m) & \longmapsto & \sum_{i=1}^{m} n_i \gamma_i. \end{array}$$

ϕ ist offensichtlich surjektiv. Da $y_i = \operatorname{ord}_{G/G_i} \gamma_i$, gibt es nach Lemma 4.2 und Lemma 4.6 minimale Relationen $(x_{i,1}, \ldots, x_{i,i-1}, y_i)$ für $(\gamma_1, \ldots, \gamma_i)$ und somit

$$\phi(x_{i,1}, \ldots, x_{i,i-1}, y_i, 0, \ldots, 0) =$$

für alle $i = 1, \ldots, m$. Da die m Relationen als Vektoren über $\mathbb{Z}^m$ trivialerweise linear unabhängig sind, gilt Kern $\phi = \mathcal{L}$ mit

$$\mathcal{L} = \left\langle \begin{pmatrix} y_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} x_{2,1} \\ y_2 \\ \vdots \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} x_{m,1} \\ \vdots \\ x_{m,m-1} \\ y_m \end{pmatrix} \right\rangle,$$

also gilt nach dem Homomorphiesatz

$$\mathbb{Z}^m / \mathcal{L} \cong <\gamma_1, \ldots, \gamma_m>$$

und folglich

$$\left| \mathbb{Z}^m / \mathcal{L} \right| = |<\gamma_1, \ldots, \gamma_m>|.$$

Nun gilt nach Satz 1.6

$$\left| \mathbb{Z}^m / \mathcal{L} \right| = \det \mathcal{L} = \begin{vmatrix} y_1 & x_{2,1} & \cdots & x_{m,1} \\ 0 & y_2 & \cdots & x_{m,2} \\ \vdots & 0 & & \vdots \\ \vdots & \vdots & & x_{m,m-1} \\ 0 & 0 & \cdots & y_m \end{vmatrix} = \prod_{i=1}^{m} y_i,$$

womit wir unsere Behauptung bewiesen hätten. ∎

Das im Beweis von Lemma 4.8 aufgetretene Gitter $\mathcal{L}$ ist das **Relationengitter** für $(\gamma_1, \ldots, \gamma_m)$.

4.9. Lemma *Sei G eine endliche abelsche Gruppe, $\gamma_1, \ldots, \gamma_m \in G$, sei*

$$G_1 = \ \{\} \textit{ und } G_i =< \gamma_1, \ldots, \gamma_{i-1} > \textit{ für } i = 2, \ldots, m$$

sowie

$$y_i = \mathrm{ord}_{G/G_i}\, \gamma_i \quad \textit{für } i = 1, \ldots, m-1.$$

Wenn L eine obere Schranke für $|G|$ ist ,so ist

$$\left\lfloor \frac{L}{\prod_{i=1}^{m-1} y_i} \right\rfloor$$

eine obere Abschätzung für $\mathrm{ord}_{G/G_m}\, \gamma_m$.

Beweis: Es gilt:

$$\begin{aligned} L \quad \geq \quad |G| \quad &= \quad \left| {}^{G}\!\big/_{G_m} \right| \cdot |G_m| \\ &= \quad \left| {}^{G}\!\big/_{G_m} \right| \cdot \prod_{i=1}^{m-1} y_i, \end{aligned}$$

woraus folgt:

$$\left| {}^{G}\!\big/_{G_m} \right| \leq \frac{L}{\prod\limits_{i=1}^{m-1} y_i}.$$

Da die Ordnung von G/G_m ganzzahlig ist, muß

$$\left| {}^{G}\!\big/_{G_m} \right| \leq \left\lfloor \frac{L}{\prod\limits_{i=1}^{m-1} y_i} \right\rfloor$$

gelten, womit wir unsere Behauptung bewiesen hätten. ■

Die Größe der für die γ_i zu durchforstenden Intervalle vermindert sich also beträchtlich; wir haben durch Ausnutzung der letzten beiden Lemmata mit $m = k$ folgenden

4.10. Algorithmus (minimale Relation mit Vorinformation)

Eingabe: $\gamma_1, \ldots, \gamma_k \in G$ $(G \in \mathcal{G})$, L obere Schranke für $|G|$, $y_i = \mathrm{ord}_{G/G_i}\, \gamma_i$ $(i = 1, \ldots, k-1)$.

Ausgabe: $(x_1, \ldots, x_k)$ minimale Relation für $(\gamma_1, \ldots, \gamma_k)$.

$M \leftarrow \emptyset$		(1)
FOR $i \leftarrow 1$ TO $k-1$ DO $z_i \leftarrow \lceil \sqrt{y_i} \rceil$		(2)
$z_k \leftarrow \left\lceil \sqrt{\left\lfloor \frac{L}{\prod_{i=1}^{k-1} y_i} \right\rfloor} \right\rceil$		(3)
FOR $r_k \leftarrow 1$ TO z_k	(Baby-Steps)	(4)
Berechne $M_{r_k} = \left\{ \left(\sum_{i=1}^{k} r_i \gamma_i, r_1, \ldots, r_k \right) : 1 \leq r_i \leq z_i, 1 \leq i \leq k-1 \right\}$, sortiert nach der ersten Komponente in $D(G)$		(5)
IF $\exists(\chi, r_1, \ldots, r_k) \in \mathbb{M}_{r_k} : \chi =$		(6)
THEN RETURN $\{(r_1, \ldots, r_k)\}$		(7)
ELSE $M \leftarrow Merge(M, M_{r_k})$		(8)
FOR $q_k \leftarrow 2$ TO $\left\lfloor \sqrt{\left\lfloor \frac{L}{\prod_{i=1}^{k-1} y_i} \right\rfloor} \right\rfloor + 1$	(Giant-Steps)	(9)
FORALL $(q_1, \ldots, q_{k-1}) \in [\![2, \lfloor \sqrt{y_1} \rfloor + 1]\!] \times \cdots \times [\![2, \lfloor \sqrt{y_{k-1}} \rfloor + 1]\!]$	(Giant-Steps)	(10)
IF $\exists(\chi, r_1, \ldots, r_k) \in M : \chi = \sum_{i=1}^{k} q_i z_i \gamma_i$ THEN RETURN $\{\ (q_1 z_1 - r_1, \ldots, q_k z_k - r_k)\ \}$		(11)

Für diesen Algorithmus gilt:

4.11. Satz *Sei $G \in \mathcal{G}$, L eine obere Schranke für $|G|$, $\gamma_1, \ldots, \gamma_k \in G$ und*

$$G_1 = \{\} \text{ und } G_i =< \gamma_1, \ldots, \gamma_{i-1} > \text{ für } i = 2, \ldots, m$$

sowie

$$y_i = \operatorname{ord}_{G/G_i} \gamma_i \text{ für } i = 1, \ldots, m-1.$$

Dann kann man unter Kenntnis der $y_1, \ldots, y_{k-1}$ eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$ in

- $(k \cdot \sqrt{L})$ *Gruppenadditionen und*
- $O(\sqrt{L} \cdot \log L \cdot \log n_G)$ *Bitoperationen*

berechnen.

Beweis: Es gilt:

$$\prod_{i=1}^{k} z_i = \left(\prod_{i=1}^{k-1} O(\sqrt{y_i}) \right) \cdot O\left(\sqrt{\frac{L}{\prod_{i=1}^{k-1} y_i}} \right) = O(\sqrt{L}).$$

Mit dieser Ersetzung im Beweis von Satz 4.4 sowie Lemma 4.8 und Lemma 4.9 ergibt sich die Behauptung. ∎

4.3 Eine iterative Berechnung von minimalen Relationen

Im vorigen Abschnitt haben wir gesehen, wie wir eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$ "schnell" berechnen können, wenn wir die Ordnungen y_i von γ_i in G/G_i kennen. Diese y_i sind nun gerade die jeweils letzten Einträge der minimalen Relationen für $(\gamma_1, \ldots, \gamma_i)$. Wir können also eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$ durch folgenden Algorithmus, der sukzessive minimale Relationen für $\gamma_1, \ldots, \gamma_i$, $i = 1, \ldots, k$ bestimmt, berechnen:

4.12. Algorithmus (Minimale Relationen Iterativ)

Eingabe: $\gamma_1, \ldots, \gamma_k \in G$ $(G \in \mathcal{G})$, L obere Schranke für $|G|$.

Ausgabe: $(x_1, \ldots, x_k)$ minimale Relation für $(\gamma_1, \ldots, \gamma_k)$.

Bestimme $x_1 = \mathrm{ord}_G\, \gamma_1$		(1)
FOR $i \leftarrow 2$ to k		(2)
	$y_{i-1} \leftarrow x_{i-1}$	(3)
	Bestimme $(x_1, \ldots, x_i)$ minimale Relation für $\gamma_1, \ldots, \gamma_i$ mit Hilfe von $(y_1, \ldots, y_{i-1})$ durch Algorithmus 4.10	(4)
RETURN $\{ (x_1, \ldots, x_k) \}$		(5)

4.13. Satz *Sei $G \in \mathcal{G}$, L eine obere Schranke für $|G|$, $\gamma_1, \ldots, \gamma_k \in G$. Dann kann man eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$ in*

- $O(k^2 \cdot \sqrt{L})$ *Gruppenadditionen und*
- $O(k \cdot \sqrt{L} \cdot \log L \cdot \log n_G)$ *Bitoperationen*

berechnen.

Beweis: Die Korrektheit des Algorithmus ergibt sich aus der Konstruktion.
Satz 3.6 und Satz 4.11 besagen, daß für alle $i = 1, \ldots, k$ der Aufwand, minimale Relationen für $\gamma_1, \ldots, \gamma_i$ unter Kenntnis der vorher berechneten $y_1, \ldots, y_{i-1}$

- $O(i \cdot \sqrt{L})$ Gruppenadditionen und
- $O(\sqrt{L} \cdot \log L \cdot \log n_G)$ Bitoperationen

beträgt. Somit ist der Gesamtaufwand:

$$\sum_{i=1}^{k} O(i \cdot \sqrt{L}) = O\left(\sqrt{L} \cdot \frac{k(k+1)}{2}\right) = O(k^2 \cdot \sqrt{L})$$

Gruppenoperationen und

$$\sum_{i=1}^{k} O(\sqrt{L} \cdot \log L \cdot \log n_G) = O(k \cdot \sqrt{L} \cdot \log L \cdot \log n_G)$$

Bitoperationen. ■

4.4 Minimale Relationen ohne obere Schranke

Wie bei der Berechnung der Ordnung eines Elementes betrachten wir die Möglichkeit, daß keine obere Schranke für die Gruppenordnung bekannt ist bzw. wir diese nicht verwenden wollen. Die Grundidee ist analog zur iterativen Berechnung der Ordnung eines Gruppenlelmentes. Wir nehmen — nach der ausführlichen Diskussion im vorigen Kapitel ist dies aus Sicht der Effizienz nicht unbedingt die praktisch beste Version — aus Gründen der Überschaubarkeit der komplexitätstheoretischen Ausdrücke als Grundversion den Baby-Step-Giant-Step-Algorithmus mit der Verdoppelung der oberen Schranke. Wir verändern nun den Algorithmus zur iterativen Berechnung einer minimalen Relation entsprechend.
Die iterative Berechnung einer minimalen Relation für γ_1 besteht gerade im Berechnen der Ordnung von γ_1 mit Algorithmus 3.12; um minimale Relationen für $\gamma_1, \ldots, \gamma_i$ iterativ zu berechnen, müssen wir gewährleisten, daß wir im Falle eines Nichterfolgs das nach $\mathrm{ord}_{G/G_i}\, \gamma_i$ durchsuchte Intervall vergrößern — dies tun wir, wie oben erwähnt, durch Verdoppeln der oberen Schranke. Weiterhin müßten wir noch den Anfangswert L_0 dieser Intervallgrösse festlegen, formulieren den Algorithmus aber ohner diese explizite Angabe. (Wir setzen diesen wieder — wie beim entsprechenden Baby-Step-Giant-Step-Algorithmus — auf 4).

4.14. Algorithmus (Minimale Relationen ohne obere Schranke)

Eingabe: $\gamma_1, \ldots, \gamma_k \in G$, $(G \in \mathcal{G})$.

Ausgabe: $(x_1, \ldots, x_k)$ minimale Relation für $(\gamma_1, \ldots, \gamma_k)$.

Bestimme $y_1 = \mathrm{ord}_G\, \gamma_1$ mit Algorithmus 3.8		(1)
Setze L_O auf einen Anfangswert		(2)
FOR $i \leftarrow 2$ to k		(3)
	$y_{i-1} \leftarrow x_{i-1}$	(4)
$L \leftarrow L_0$		(5)
	Versuche eine minimale Relation $(x_1, \ldots, x_i)$ für $\gamma_1, \ldots, \gamma_i$ mit Hilfe von $(y_1, \ldots, y_{i-1})$ und oberer Schranke L durch Algorithmus 4.10 zu bestimmen	(6)
	$L \leftarrow 2L$	(7)
UNTIL Versuch erfolgreich		(8)

4.15. Satz *Sei* $G \in \mathcal{G}$, $\gamma_1, \ldots, \gamma_k \in G$, $h = |<\gamma_1, \ldots, \gamma_k>|$. *Dann kann man eine minimale Relation für* $(\gamma_1, \ldots, \gamma_k)$ *in*

- $O(k^2 \cdot \sqrt{h})$ *Gruppenadditionen und*
- $O(k \cdot \sqrt{h} \cdot \log h \cdot \log n_G)$ *Bitoperationen*

berechnen.

Beweis: Die Korrektheit des Algorithmus ergibt aus seiner Konstruktion.
Wir benutzen die Bezeichnungen aus Abschnitt 4.2. Der Aufwand für die Berechnung der Ordnung von γ_1 beträgt nach Satz 3.13

- $O(\sqrt{\mathrm{ord}\, \gamma_1}) = O(\sqrt{y_1})$ Gruppenadditionen und
- $O(\sqrt{\mathrm{ord}\, \gamma_1} \cdot \log \mathrm{ord}\, \gamma_1 \cdot \log n_G) = O(\sqrt{y_1} \cdot \log y_1 \cdot n_G)$ Bitoperationen;

betrachten wir nun den Aufwand T_i an Gruppenadditionen in der i-ten Iteration:
Sei j die Anzahl der Anwendungen von Algorithmus 4.10. Dann gilt:

$$\begin{aligned}
T_i &= \sum_{l=0}^{j-1} O\left(i \cdot \left(\prod_{\kappa=1}^{i-1} z_\kappa\right) \cdot L_0 \cdot 2^l\right) \\
&= O\left(i \cdot \left(\prod_{\kappa=1}^{i-1} \sqrt{y_\kappa}\right) \sum_{l=0}^{j-1} L_0 \cdot 2^l\right) \\
&= O\left(i \cdot \left(\prod_{\kappa=1}^{i-1} \sqrt{y_\kappa}\right) \cdot \sqrt{\mathrm{ord}_{G/G_i}\,\gamma_i}\right) \\
&= O\left(i \cdot \left(\prod_{\kappa=1}^{i-1} \sqrt{y_\kappa}\right) \cdot \sqrt{y_i}\right) \\
&= O\left(i \cdot \prod_{\kappa=1}^{i} \sqrt{y_\kappa}\right)
\end{aligned}$$

nach einer zum Beweis von Satz 3.13 analogen Rechnung und so ist der Gesamtaufwand T an Gruppenadditionen für die Berechnung einer minimalen Relation für $\gamma_1, \ldots, \gamma_k$

$$T = O\left(\sum_{i=1}^{k} i \cdot \prod_{\kappa=1}^{i} \sqrt{y_\kappa}\right).$$

Diese Funktion können wir nun grob abschätzen

$$T = O\left(\sum_{i=1}^{k} i \cdot \prod_{\kappa=1}^{k} \sqrt{y_i}\right) = O\left(\sum_{i=1}^{k} i \cdot \sqrt{h}\right) = O(k^2 \cdot \sqrt{h})$$

und erhalten die Behauptung. Der Beweis für die Anzahl der Bitoperationen geht analog. ■

4.5 Parallele Berechnung von minimalen Relationen

Die Übertragung der Idee der Parallelisierung des Baby-Step-Giant-Step-Algorithmus auf die iterative Version der Berechnung von minimalen Relationen in der Weise, wie sie bei den beiden vorherigen Abschnitten funktioniert hat, nämlich nur die jeweils letzten Komponenten aufzuteilen, erreicht nicht die gewünschte Arbeitsoptimalität. Wenn wir allerdings wieder auf den ursprünglichen Algorithmus zur Berechnung von minimalen Relationen zurückgreifen, können wir durch Aufteilung *aller* zu berechnenden Gruppenelemente der Baby-Step- bzw. Giant-Step-Phase auf entsprechend viele Prozessoren mit dem gleichen Trick wie beim Berechnen der Ordnung eines Elementes Arbeitsoptimalität bezüglich der Anzahl an Gruppenoperationen erreichen.

4.16. Algorithmus (Parallele Berechnung einer minimalen Relation)

Eingabe: $\gamma_1, \ldots, \gamma_k \in G$ $(G \in \mathcal{G})$, L obere Schranke für $|G|$

Ausgabe: $(x_1, \ldots, x_k)$ minimale Relation für $(\gamma_1, \ldots, \gamma_k)$

Anzahl der Prozessoren: $\lceil \frac{\lceil \sqrt{L} \rceil^k}{\log \lceil \sqrt{L} \rceil^k} \rceil$.

$M \leftarrow \emptyset$	(1)
FOR $i \leftarrow 1$ TO k DO $z_i \leftarrow \lceil\sqrt{L}\rceil$	(2)
FOR $r_k \leftarrow 1$ TO z_k (Baby-Steps)	(3)
Berechne $M_{r_k} = \left\{\left(\sum_{i=1}^{k} r_i\gamma_i, r_1, \ldots, r_k\right) : 1 \leq r_i \leq z_i, 1 \leq i \leq k-1\right\}$, sortiert nach der ersten Komponente in $D(G)$, gleichmäßig verteilt auf alle Prozessoren	(4)
IF $\exists(\chi, r_1, \ldots, r_k) \in \mathbb{M}_{r_k} : \chi =$	(5)
THEN RETURN $\{(r_1, \ldots, r_k)\}$	(6)
ELSE $M \leftarrow Merge(M, M_{r_k})$	(7)
FOR $q_k \leftarrow 2$ TO $\lfloor\sqrt{L}\rfloor + 1$ (Giant-Steps)	(8)
FORALL $(q_1, \ldots, q_{k-1}) \in [\![2, \lfloor\sqrt{L}\rfloor + 1]\!]^{k-1}$ gleichmäßig auf alle Prozessoren verteilt (Giant-Steps)	(9)
IF $\exists(\chi, r_1, \ldots, r_k) \in M : \chi = \sum_{i=1}^{k} q_i z_i \gamma_i$ THEN RETURN $\{\ (q_1 z_1 - r_1, \ldots, q_k z_k - r_k)\ \}$	(10)

Es gilt der

4.17. Satz *Sei $G \in \mathcal{G}$, $\gamma_1, \ldots, \gamma_k \in G$. Dann läßt sich eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$ mit Kenntnis einer oberen Schranke der Gruppenordnung L mit $O(\frac{\sqrt{L^k}}{\log L})$ Prozessoren in*

- *$O(k \cdot \log L)$ Zeiteinheiten für Gruppenadditionen und*
- *$O(k^2 \cdot \log^2 L \cdot \log n_G)$ Zeiteinheiten für Bitoperationen*

berechnen.

Beweis: Der Gesamtaufwand an Gruppenadditionen beträgt $O\left((\sqrt{L})^k\right)$ (s. Algorithmus 3.5); da die zu berechnenden Tupel gleichmäßig auf alle Prozessoren verteilt sind, muß jeder Prozessor

$$O\left(\log(\sqrt{L})^k\right) = O(k \cdot \log L)$$

Gruppenadditionen durchführen. Analog ergibt sich, mit Betrachtung des Beweises von Algorithmus 3.5, die Behauptung für die Bitoperationen. ■

Ab hier betrachten wir keine parallelen Algorithmen zur Lösung der besprochen Probleme mehr, denn es würde sich dabei nur um eine — wie es hier auch der Fall ist — kanonische Verteilung der Arbeit handeln; auch wenn dabei einige Gesichtspunkte zu beachten sind, so ergeben sich auf diese Weise keine prinzipell *neuen* Algorithmen. Auch kann man die Ergebnisse, die in späteren Kapiteln aufgrund von parallel(isiert)en Algorithmen auftreten, analog zur hier dargelegten Weise herleiten.

4.6 Der diskrete Logarithmus

Die bisher bewiesenen Ergebnisse können ohne Aufwand auf den Spezialfall $k = 2$ angewandt werden. Wir geben hier eine Grundversion der Berechnung an; diese kann, wie in diesem Kapitel beschrieben, unter verschiedenen Voraussetzungen entsprechend angepaßt werden. Wir bekommen somit Komplexitätsaussagen für die Lösung des Problems des *diskreten Logarithmus*:

Gegeben $\alpha, \beta \in G$, gibt es $x \in \mathbb{N}$ so daß $\alpha = x\beta$?

4.18. Algorithmus (Diskreter Logarithmus)

Eingabe: $\alpha, \beta \in G$ ($G \in \mathcal{G}$).

Ausgabe: $x \in \mathbb{N}$ mit $\alpha = x\beta$ oder *nichtlösbar*.

Berechne $y = \operatorname{ord}\beta$ mit Algorithmus 3.5 oder Varianten		(1)
$m \leftarrow \lceil\sqrt{y}\rceil, \quad \gamma_0 \leftarrow \alpha$		(2)
FOR $i \leftarrow 1$ TO $m-1$ DO	Baby-Steps	(3)
IF $\gamma_{i-1} =$ THEN RETURN $\{i-1\}$		(4)
$\gamma_i \leftarrow \gamma_{i-1} - \beta$		(5)
$\gamma^1 \leftarrow -\gamma_i + \alpha$		(6)
FOR $j \leftarrow 2$ TO $\lfloor\sqrt{y}\rfloor + 1$	Giant-Steps	(7)
IF $\gamma^{j-1} = \gamma_i$ für ein $i \in [\![0, m-1]\!]$ THEN RETURN $\{(j-1)\cdot m + i\}$		(8)
$\gamma^j \leftarrow \gamma^{j-1} + \gamma^1$		(9)
RETURN { *keine Lösung* }		(10)

Man kann die Bestimmung der minimalen Relation $(x, 1)$ für (β, α) auf einen leicht modifizierten Baby-Step-Giant-Step-Algorithmus reduzieren, denn die Gleichung ist genau dann lösbar, wenn die Ordnung von α in $G/<\beta>$ genau 1 ist. Somit ergibt sich die Korrektheit des obigen Algorithmus. Der Aufwand für den zweiten Teil Algorithmus 4.18 (d.h. ab Anweisung (2)) entspricht asymptotisch dem Aufwand des Baby-Step-Giant-Step-Algorithmus mit oberer Schranke $\operatorname{ord}\beta$. Mit den verschiedenen Voraussetzungen gilt dann:

4.19. Satz *Sei $G \in \mathcal{G}$, L eine obere Schranke für $|G|$, $\alpha, \beta \in G$. Dann ist der Aufwand, die Lösbarkeit von*

$$\alpha = x\beta,$$

zu entscheiden und gegebenenfalls eine Lösung zu berechnen,

- $O(\sqrt{L})$ *Gruppenoperationen und*
- $O(\sqrt{L} \cdot \log L \cdot \log n_G)$ *Bitoperationen.*

4.20. Satz *Sei $G \in \mathcal{G}$, $\alpha, \beta \in G$, $h = |<\beta>|$. Dann ist der Aufwand, die Lösbarkeit von*

$$\alpha = x\beta,$$

zu entscheiden und gegebenenfalls eine Lösung zu berechnen,

- $O(\sqrt{h})$ *Gruppenoperationen und*
- $O(\sqrt{h} \cdot \log h \cdot \log n_G)$ *Bitoperationen.*

Es zeigt sich, daß eine Lösung des Problems des diskreten Logarithmus ohne Einbeziehen einer Abschätzung der Gruppenordnung komplexitätstheoretisch besser ist; allerdings ist der dabei auftretende tatsächliche Aufwand nicht gering, so daß die Entscheidung wohl zugunsten der Version mit Einbeziehen einer Abschätzung der Gruppenordnung ausfällt.

4.21. Satz *Sei $G \in \mathcal{G}$, L eine obere Schranke für $|G|$, l eine untere Schranke für $|G|$, $\alpha, \beta \in G$. Dann ist der Aufwand, die Lösbarkeit von*

$$\alpha = x\beta,$$

zu entscheiden und gegebenenfalls eine Lösung zu berechnen,

- $O(\sqrt{L-l} + \log l))$ *Gruppenoperationen und*
- $O(\sqrt{L-l} \cdot \log(L-l) \cdot \log n_G + \log^2 l)$ *Bitoperationen.*

4.22. Satz *Sei $G \in \mathcal{G}$, $\alpha, \beta \in G$. Dann kann man die Lösbarkeit von*

$$\alpha = x\beta,$$

unter der Kenntnis einer oberen Schranke der Gruppenordnung L mit $O(\frac{L}{\log L})$ Prozessoren in

- $O(\log L)$ *Zeiteinheiten für Gruppenadditionen und*
- $O(\log L \cdot \log n_G)$ *Zeiteinheiten für Bitoperationen*

entscheiden und gegebenenfalls eine Lösung zu berechnen.

Kapitel 5

Wenn die Zerlegung der Gruppenordnung bekannt ist

In diesem Kapitel wollen wir die Berechnung von minimalen Relationen mit Kenntnis der Primfaktorzerlegung der Gruppenordnung besprechen. Hierzu reduzieren wir das Problem mittels des Chinesischen Restsatzes auf die Berechnung von minimalen Relationen in p-Gruppen und gehen dann ausführlich auf diese Problemstellung ein. Schließlich werden wir die erarbeiteten Ergebnisse zusammensetzen.

5.1 Minimale Relationen und der Chinesische Restsatz

Wir wollen minimale Relationen berechnen unter der Voraussetzung, die Gruppenordnung und dessen Primfaktorzerlegung zu kennen. Wir werden hier die Problemstellung auf den einfacheren Fall der p-Gruppen reduzieren, indem wir den Chinesischen Restsatz anwenden.

5.1. Satz (Chinesischer Restsatz) *Seien $n_1, \ldots, n_m \in \mathbb{N}$ paarweise teilerfremd. Dann sind die simultanen Kongruenzen*

$$x \equiv x_i \bmod n_i \qquad (i = 1, \ldots, n)$$

lösbar und haben modulo $\prod_{i=1}^{m} n_i$ genau eine Lösung x.

Beweis: Siehe z.B. [Ri85, S. 268f]. ■

Für jeden Primteiler p der Gruppenordnung einer gegebenen endlichen abelschen Gruppe G übertragen wir die Berechnung von minimalen Relationen auf eine p-Gruppe, indem wir die betrachteten Elemente mit dem größten Teiler der Gruppenordnung, der prim zu p ist, multiplizieren und so in eine p-Gruppe projizieren. Daß dies sinnvoll ist, d.h. wir danach die minimalen Relationen aus den verschiedenen p-Gruppen zusammensetzen können, begründet das folgende Lemma:

5.2. Lemma *Sei G eine endliche abelsche Gruppe, $\gamma_1, \ldots, \gamma_k \in G$, sei*

$$h = |G| = \prod_{j=1}^{s} {p_j}^{e_j} \quad \text{und} \quad h_j = \frac{h}{{p_j}^{e_j}} \qquad (j = 1, \ldots, s)$$

wobei die p_j paarweise verschiedene Primzahlen sind, und

$$H_j =< h_j\gamma_1, \ldots, h_j\gamma_{k-1} > \qquad (j = 1, \ldots, s)$$

sowie

$$G_k =< \gamma_1, \ldots, \gamma_{k-1} > .$$

Dann gilt:

$$\operatorname{ord}_{G/G_k} \gamma_k = \prod_{j=1}^{s} \operatorname{ord}_{G/H_j} h_j\gamma_k.$$

Beweis: Sei x die Ordnung von γ_k in G/G_k,

$$x = \prod_{j=1}^{s} {p_j}^{g_j},$$

und sei x_j die Ordnung von $h_j\gamma_k$ in G/G_k. Betrachte für $j = 1, \ldots, s$

$${p_j}^{e_j}(h_j\gamma_k) = ({p_j}^{e_j} h_j)\gamma_k = h\gamma_k =\in H_j;$$

$$x_j = \operatorname{ord}_{G/H_j} h_j\gamma_k \quad \Longrightarrow \quad x_j | {p_j}^{e_j} \text{ bzw. } x_j = {p_j}^{f_j}, f_j \leq e_j.$$

Betrachte andererseits für $j = 1, \ldots, s$

$$x(h_j\gamma_k) = h_j(x\gamma_k) \in h_j \cdot G_k = H_j =_{G/H_j};$$

$$x_j = \operatorname{ord}_{G/H_j} h_j\gamma_k \quad \Longrightarrow \quad x_j | x \text{ bzw. } f_j \leq g_j.$$

Betrachte nun für $j = 1, \ldots, s$

$$(x_j h_j)\gamma_k = x_j(h_j\gamma_k) =_{G/H_j} = H_j \leq G_j =_{G/G_k};$$

$$x = \operatorname{ord}_{G/G_k} \gamma_k \quad \Longrightarrow \quad x | x_j h_j = x_j \frac{h}{p_j} e_j$$

$$\Longrightarrow \quad {p_j}^{g_j} | x_j = {p_j}^{f_j} \quad \Longrightarrow g_j \leq f_j.$$

Also gilt $f_j = g_j$, $j = 1, \ldots, s$ und somit

$$x = \prod_{j=1}^{s} x_j.$$

■

Mit diesem Lemma können wir nun die zentrale Behauptung dieses Abschnitts zeigen, die aussagt, warum man, um eine minimale Relation zu berechnen, zuerst die minimalen Relationen in allen "*p*-Untergruppen" berechnen kann:

5.3. Satz *Sei G eine endliche abelsche Gruppe,*

$$h = |G| = \prod_{j=1}^{s} {p_j}^{e_j}, \quad h_j = \frac{h}{{p_j}^{e_j}} \qquad (j = 1, \ldots, s),$$

$\gamma_1, \ldots, \gamma_k \in G$. *Wenn für alle* $j = 1, \ldots, s$ *die* $(x_{1,j}, \ldots, x_{k,j})$ *minimale Relationen für* $(h_j\gamma_1, \ldots, h_j\gamma_k)$ *sind, so ist* $(x_1, \ldots, x_k) \in [\![0, h-1]\!]^k$ *mit*

$$\bigvee_{i \in [\![1,k]\!]} x_i \equiv x_{i,j} \bmod {p_j}^{e_j} \qquad (j = 1, \ldots, s)$$

minimale Relation für $(\gamma_1, \ldots, \gamma_k)$.

Beweis: Es gilt

$$x_i = l_{i,j}{p_j}^{e_j} + x_{i,j} \qquad (i = 1, \ldots, k) \quad (j = 1, \ldots, s)$$

nach Konstruktion.

1. Wir zeigen zuerst, daß $(x_1, \ldots, x_k)$ Relation für $(\gamma_1, \ldots, \gamma_k)$. Für alle $j = 1, \ldots, s$ gilt:

$$\begin{aligned}
& (x_{1,j}, \ldots, x_{k,j}) \text{ Relation für } (h_j\gamma_1, \ldots, h_j\gamma_k) \\
\Longrightarrow \quad & \sum_{i=1}^{k} x_{i,j}(h_j\gamma_i) = \text{ nach Definition} \\
\Longrightarrow \quad & \sum_{i=1}^{k} x_{i,j}(h_j\gamma_i) + \sum_{i=1}^{k} l_{i,j}h\gamma_i =, \text{ denn } h = |G| \\
\Longrightarrow \quad & \sum_{i=1}^{k}(l_{i,j}h + x_{i,j}h_j)\gamma_i = \\
\Longrightarrow \quad & \sum_{i=1}^{k}(l_{i,j}{p_j}^{e_j}h_j + x_{i,j}h_j)\gamma_i =, \text{ denn } h = p_j^{e_j}h_j \\
\Longrightarrow \quad & \sum_{i=1}^{k}(l_{i,j}{p_j}^{e_j} + x_{i,j})h_j\gamma_i = \\
\Longrightarrow \quad & \sum_{i=1}^{k} x_i h_j \gamma_i = \\
\Longrightarrow \quad & h_j \left(\sum_{i=1}^{k} x_i\gamma_i \right) = .
\end{aligned}$$

   Damit haben wir:

$$\Longrightarrow \quad \operatorname{ggT}(h_1, \ldots, h_s) \left(\sum_{i=1}^{k} x_i\gamma_i \right) = .$$

   Nun gilt nach Konstruktion $\operatorname{ggT}(h_1, \ldots, h_k) = 1$, also ist $(x_1, \ldots, x_k)$ Relation für $(\gamma_1, \ldots, \gamma_k)$.

2. Es gilt (mit den Bezeichnungen von Lemma 5.2) für $j = 1, \ldots, s$

$$x_{k,j} = \operatorname{ord}_{G/H_j} h_i\gamma_k = {p_i}^{f_i}$$

   und nach Lemma 5.2 existiert

$$x = \prod_{i=1}^{s} x_{k,j} = \operatorname{ord}_{G/G_k} \gamma_k;$$

   für dieses x gilt:

$$x \equiv x_{k,j} \bmod {p_j}^{e_j} \quad (j = 1, \ldots, s),$$

   denn

$$x_{k,l} \equiv 0 \bmod {p_j}^{e_j} \qquad \text{für alle } l \neq j.$$

   Da die durch die Kongruenzen definierten x_i laut Satz 5.1 modulo h eindeutig sind, gilt $x_k = x$ und somit ist $(x_1, \ldots, x_k)$ *minimale* Relation für $(\gamma_1, \ldots, \gamma_k)$. ■

5.2 Der Chinesische Rest-Algorithmus

Um die modulo h eindeutig bestimmten Lösungen der simultanen Kongruenzen zu berechnen, benötigt man eine Erweiterung des euklidischen Algorithmus:

5.4. Algorithmus (erweiterter euklidischer Algorithmus)

Eingabe: $a,\ b \in \mathbb{N}$.

Ausgabe: $d, s, t \in \mathbb{Z}$ mit $\operatorname{ggT}(a,b) = d$ und $sa + tb = d$.

$(u_1, u_2, u_3) \leftarrow (1, 0, a)$; $(v_1, v_2, v_3) \leftarrow (0, 1, b)$		(1)
WHILE $v_3 > 0$		(2)
	$q \leftarrow \lfloor \frac{u_3}{v_3} \rfloor$	(3)
	$(t_1, t_2, t_3) \leftarrow (u_1, u_2, u_3) - q(v_1, v_2, v_3)$; $(u_1, u_2, u_3) \leftarrow (v_1, v_2 v_3)$; $(v_1, v_2, u_3) \leftarrow (t_1, t_2, t_3)$	(4)
RETURN $\{\ u_1, u_2, u_3\}$		(5)

Zur Korrektheit dieses Algorithmus siehe [Pe91].

5.5. Satz *Sei T die Anzahl an notwendigen Bitoperationen, um den erweiterten euklidischen Algorithmus auf $a,\ b \in \mathbb{N}$ anzuwenden, $a > b$, $n = \log_2 a$, $m = \log_2 b$, $k = \log_2(\operatorname{ggT}(a,b))$. Dann gilt:*

$$T = O\left(\max\{m, n-k\} \cdot (n-k+1)\right).$$

Beweis: Wir bezeichnen den Wert der Variablen x nach dem i-ten Schleifendurchlauf mit $x^{(i)}$ und mit d den ggT von a und b. Wir verwenden folgende Ergebnisse aus [Kn73, 4.3]:

- Die Division mit Rest $u_3^{(i)} = q^{(i+1)} v_3^{(i)} + t_3^{(i+1)}$ kann in $O(\log v_3^{(i)}(\log q^{(i+1)} + 1))$ Bitoperationen berechnet werden.
- Das Produkt $q^{(i+1)} v_j^{(i)}$, und somit die Anweisungen $t_j^{(i+1)} \leftarrow u_j^{(i)} - q^{(i+1)} v_j^{(i)}$ können in $O(\log q^{(i+1)} \log |v_j^{(i)}|)$ Bitoperationen berechnet werden.

Sei l die Anzahl der Schleifendurchläufe. Dann ist $v_3^{(l)} = 0$ und es gilt:

$$T = \sum_{i=0}^{l-1} \Big(O(\log v_3^{(i)}(\log q^{(i+1)} + 1)) + O(\log q^{(i+1)} \log |v_1^{(i)}|) + O(\log q^{(i+1)} \log |v_2^{(i)}|) \Big)$$

Nun gilt:

$$|v_1^{(i)}| < \frac{b}{d} \quad \text{und} \quad |v_2^{(i)}| < \frac{a}{d},$$

wie folgende Überlegung zeigt: man kann durch Induktion zeigen, daß

- die Folgen $|v_1^{(i)}|$ und $|v_2^{(i)}|$ monoton wachsend sind,
- für alle i gilt:
$$v_1^{(i)} \cdot v_2^{(i-1)} + v_1^{(i-1)} \cdot v_2^{(i)} = (-1)^i,$$
woraus sofort folgt, daß $v_1^{(i)}$ und $v_2^{(i)}$ teilerfremd sind und

- natürlich die Schleifeninvariante aus dem Korrektheitsbeweis gilt:
$$v_1^{(i)} \cdot a + v_2^{(i)} \cdot b = v_3^{(i)}.$$

Da $v_3^{(l)} = 0$, gilt:
$$v_1^{(l)} \cdot a + v_2^{(l)} \cdot b = 0 \quad \Longrightarrow \quad v_1^{(l)} \cdot a = -v_2^{(l)} \cdot b \quad \Longrightarrow \quad v_1^{(l)} \cdot \frac{a}{d} = -v_2^{(l)} \cdot \frac{b}{d},$$
woraus folgt
$$\frac{b}{d} = k \cdot v_1^{(l)} \quad \text{und} \quad \frac{a}{d} = k \cdot v_2^{(l)} \quad \text{mit } k \in \mathbb{Z}$$
und da $v_1^{(i)}$ und $v_2^{(i)}$ teilerfremd sind, muß
$$|v_1^{(l)}| = \frac{b}{d} \quad \text{und} \quad |v_2^{(l)}| = \frac{a}{d}$$
gelten. Somit hätten wir unsere Teilbehauptung gezeigt. Da die $v_3^{(i)}$ eine absteigende Folge bilden, gilt:
$$\begin{aligned}
T &= \sum_{i=0}^{l-1} \Big(O(\log v_3^{(i)}(\log q^{(i+1)} + 1)) + O(\log q^{(i+1)} \log |v_1^{(i)}|) + O(\log q^{(i+1)} \log |v_2^{(i)}|) \Big) \\
&= \sum_{i=0}^{l-1} \Big(O(\log v_3^{(0)} \cdot \log q^{(i+1)}) + O(\log q^{(i+1)} \log |v_1^{(l)}|) + O(\log q^{(i+1)} \log |v_2^{(l)}|) \Big) \\
&= O\left(\log \max\{v_3^{(0)}, |v_1^{(l)}|, |v_2^{(l)}|\}\right) \sum_{i=0}^{l-1} O(\log q^{(i+1)}) \\
&= O\left(\log \max\{b, \frac{b}{d}, \frac{a}{d}\}\right) \sum_{i=0}^{l-1} O(\log q^{(i+1)}) \\
&= O\left(\log \max\{b, \frac{a}{d}\}\right) \sum_{i=0}^{l-1} O(\log q^{(i+1)}) + 1.
\end{aligned}$$

Da $q^{(i+1)} = \lfloor \frac{v_3^{(i-1)}}{v_3^{(i)}} \rfloor$, gilt:
$$\begin{aligned}
\sum_{i=0}^{l-1} O(\log q^{(i+1)}) &= O\left(\log \prod_{i=0}^{l-1} q^{(i+1)}\right) \\
&= O\left(\log \prod_{i=1}^{l-1} \frac{v_3^{(i-1)}}{v_3^{(i)}}\right) \\
&= O\left(\log \frac{v_3^{(0)}}{v_3^{(l-1)}}\right) = O(\log a - \log d).
\end{aligned}$$

Somit hätten wir unsere Behauptung gezeigt. ■

Es folgt sofort

5.6. Korollar *Sei T die Anzahl an notwendigen Bitoperationen, um den erweiterten euklidischen Algorithmus auf $a, b \in \mathbb{N}$ anzuwenden, $a > b$, $n = \log_2 a$, $m = \log_2 b$, $k = \log_2(\mathrm{ggT}(a, b))$. Dann gilt:*
$$T = O(\log^2 a).$$

Es gibt komplexitätstheoretisch bessere Algorithmen zur Bestimmung des ggT zweier Zahlen inklusive "Darstellung" (Algorithmus von *Lehmer* mit Komplexität $O(n \cdot \log^5 n \cdot \log\log n)$ Bitoperationen, wobei n die Länge der größeren Zahl ist, theoretisch verbessert von *Schönhage* zu $O(n \cdot \log^2 n \cdot \log\log n)$), jedoch werden diese nach meiner praktischen Erfahrung (siehe [WePa91]) erst für Zahlen der Größenordnung 10^{90} attraktiv. Die tatsächliche Laufzeit des (erweiterten) euklidischen Algorithmus läßt sich in der Praxis jedoch wesentlich verbessern. Man beobachtet, daß in vielen Iterationen des Algorithmus der auftretende Quotient q den Wert 1 oder 2 hat, man also durch 2 Subtraktionen statt 1 Division den Aufwand wesentlich verringern kann. Praktische Ergebnisse findet man auch [Be92].

Wir werden die Anwendung dieses Algorithmus in algorithmischen Beschreibungen in Form einer Funktion schreiben, etwa:

$$(d, u, v) \leftarrow ERWEUKLALG(a, b),$$

wobei $d = \mathrm{ggT}(a, b)$ und u, v die Koeffizienten aus der Darstellung

$$a \cdot u + b \cdot v = d$$

sind. Nun formulieren wir den *Chinesischen Rest-Algorithmus*, den Algorithmus zur Berechnung der Lösung der simultanen Kongruenzen aus dem Chinesischen Restsatz.

5.7. Algorithmus (Chinesischer Rest-Algorithmus)

Eingabe: $n_1, \ldots, n_m \in \mathbb{Z}$ paarweise teilerfremd, $x_1, \ldots, x_m \in \mathbb{Z}$.

Ausgabe: $x \in [\![0, \left(\prod_{i=1}^{m} n_i\right) - 1]\!]$ mit $x \equiv x_i \bmod n_i \quad (i = 1, \ldots, m)$.

$n \leftarrow n_1$; $x \leftarrow x_1 \bmod n_1$		(1)
FOR $i \leftarrow 2$ TO m		(2)
	$(1, u, v) \leftarrow ERWEUKLALG(n, n_i)$	(3)
	$x \leftarrow unx_i + vn_ix$; $n \leftarrow nn_i$; $x \leftarrow x \bmod n$	(4)

Zur Korrektheit betrachte [Dü92, S.34].

Für die Komplexität dieses Algorithmus gilt:

5.8. Satz *Seien $n_1, \ldots, n_m \in \mathbb{N}$ paarweise teilerfremd, $n = \prod_{i=1}^{m} n_i$ und $x_1, \ldots, x_m \in \mathbb{Z}$, $x_1, \ldots, x_m < n$, sei T die Anzahl der benötigten Bitoperationen um die Lösung der simultanen Kongruenzen*

$$x \equiv x_i \bmod n_i \qquad (i = 1, \ldots, n)$$

zu berechnen. Dann gilt:

$$T = O(m \cdot \log^2 n).$$

Beweis: Sei $T_{Ausdruck}$ die Anzahl der Bitoperationen, die notwendig sind, um den Ausdruck (oder den Algorithmus) *Ausdruck* zu berechnen, $x^{(i)}$ der Inhalt der Variablen x am Anfang des $(i-1)$-ten Schleifendurchlaufs. Wie wir bei der Komplexitätsbetrachtung des euklidischen Algorithmus gesehen haben, gilt

$$T_{ggT(n^{(i)}, n_i)} = O(\max\{\log n_i, \log n^{(i)} - \log 1\} \cdot (\log n^{(i)} - \log 1 + 1) = O(\log^2 n^{(i)})$$

und

$$|u^{(i)}| < n_i \quad \text{und} \quad |v^{(i)}| < n_i.$$

Also gilt:

$$\begin{aligned}
T &= \sum_{i=2}^{m} \left(T_{ggT(n^{(i)},n_i)} + T_{u^{(i)}\cdot n^{(i)}\cdot x_i} + T_{v^{(i)}\cdot n_i\cdot x^{(i)}} + T_{n^{(i)}\cdot n_i} + T_{x^{(i)} \bmod n^{(i+1)}}\right) \\
&= \sum_{i=2}^{n} \Big(O(\log^2 n^{(i)})\Big) \\
&\quad +O(\log n_i \cdot \log n^{(i)}) + O((\log n_i + \log n^{(i)} + 1)\log n^{(i)}) \\
&\quad +O(\log n_i \cdot \log n^{(i)}) + O((\log n_i + \log n^{(i)} + 1)\log n^{(i)}) \\
&\quad +O(\log n_i \cdot \log n^{(i)}) \\
&\quad +O((\log n^{(i)} + \log n_i + 1) \cdot \log n^{(i)}) \\
&= \sum_{i=2}^{m} \left(\log n^{(i)}\right)^2 \\
&= \sum_{i=2}^{m} \left(\log \prod_{j=1}^{i-1} n_j\right)^2 .
\end{aligned}$$

Es ist hier zu bemerken, daß der größte Aufwand in jeder Runde nicht durch den Aufruf des erweiterten euklidischen Algorithmus gegeben ist, sondern durch die Multiplikation von $v^{(i)} \cdot n_i$ mit $x^{(i)}$, Zahlen der Größenordnung $O(\log n^{(i)} + \log n_i)$ und $O(\log n^{(i)})$. Diesen großen Aufwand umgeht man auch nicht, wenn man (siehe [Pe91]) alle Koeffizienten in

$$u_{i,j} n_i + v_{i,j} n_j = 1, \quad i \neq j$$

vorberechnet und dann x mit Hilfe dieser Zahlen zusammensetzt.

Um nun die Behauptung einzusehen, genügt es zu bemerken, daß

$$\forall_{i\in [\![2,m]\!]} \prod_{j=1}^{i-1} n_j \leq n.$$

■

Diese Abschätzung ist besonders grob, wenn alle n_i ungefähr gleich groß sind. Man kann auch folgende Abschätzung zeigen:

Bezeichne $n_{max} = \max\{n_i : i = 1, \ldots, m\}$*. Dann gilt*

$$T = O(m^3 \cdot \log^2 n_{max}).$$

Allerdings ist diese Abschätzung zu unhandlich; wir benutzen im folgenden, mit dem Wissen der Grobheit dieser Abschätzung, die erste der beiden Abschätzungen.

Wir haben nun die Voraussetzungen geklärt, unter denen man die Berechnung von minimalen Relationen in einer beliebigen endlichen abelschen Gruppe reduzieren kann auf die Berechnung von minimalen Relationen in p-Gruppen.

5.3 Minimale Relationen in p-Gruppen

Ab hier betrachten wir Gruppen G mit Ordnung $h = p^l$. Es gilt:

$$\forall_{\gamma_i \in G} \quad \operatorname{ord}\gamma_i = p^{l_i}, \quad l_i \leq l \quad \text{denn} \quad \operatorname{ord}\gamma_i | h = p^l.$$

Wir möchten wieder zu $\gamma_1, \ldots, \gamma_k \in G$ eine minimale Relation $(x_1, \ldots, x_k)$ bestimmen. Hierzu gehen wir prinzipiell wie im vorigen Abschnitt vor: wir versuchen für jede mögliche Ordnung x_k von $\gamma_k \in G/G_k$ (mit den Bezeichnungen des vorigen Kapitels) eine "Linearkombination" für $x_k\gamma_k \in G_k$ unter Kenntnis der Ordnungen von γ_i in G/G_i zu bestimmen. In p-Gruppen ist zu erwarten, daß dies viel schneller gehen kann, da alle auftretenden Ordnungen Potenzen von p sein müssen.

Zuerst bestimmen wir die Ordnung von einem Element einer p-Gruppe:

5.9. Algorithmus (Ordnung in p-Gruppe)
Eingabe: $\gamma \in G$ $(G \in \mathcal{G})$, $|G| = p^l$.
Ausgabe: $i \in \mathbb{N}$ mit $\operatorname{ord}\gamma = p^i$.

$i \leftarrow 0$	(1)
WHILE $\gamma \neq$ DO $i \leftarrow i+1$; $\gamma \leftarrow p \cdot \gamma$ mit Algorithmus 3.2	(2)
RETURN $\{ i \}$.	(3)

5.10. Satz *Sei $G \in \mathcal{G}$, $|G| = p^l$. Dann läßt sich die Ordnung p^i eines Elementes $\gamma \in G$ in*

- $O(i \cdot \log p)$ *Gruppenadditionen und*
- $O(i \cdot \log^2 i)$ *Bitoperationen*

berechnen.

Beweis: Trivial: Algorithmus 3.2 wird i Mal angewandt. ■

Eine Realisierung der Baby-Step-Giant-Step-Idee (mit den Potenzen) ist hier nicht möglich.

Der Berechnung einer minimalen Relation liegt folgende Idee zugrunde (hierbei bezeichne wieder $G_k =< \gamma_1, \ldots, \gamma_{k-1} >$):

> *Probiere für $j = 0, \ldots, l_k$:*
>
> > *Ist $p^j\gamma_k \in G_k$, d.h. führt die Berechnung einer Linearkombination für $p^j\gamma_k$ in G_k zu einem Ergebnis, ist man fertig.*

Wenn die Ordnungen p^{l_i} der γ_i in G bekannt sind, wird die Berechnung einer Linearkombination innerhalb der obigen Schleife auf Elemente der Ordnung

$$\leq p^{\max\{l_i : 1 \leq i \leq k-1\}} =: p^s$$

reduziert. Dieses Wissen benötigt man, um die folgende Idee zu realisieren:

Sei $\mu \in G_k$ mit $\operatorname{ord}_{G_k} \mu = p^s$. Dann gilt:

$$\mu = \sum_{i=1}^{k-1} y_i\gamma_i = \sum_{i=1}^{k-1} \left(\sum_{j=0}^{s-1} z_{i,j}p^j \right) \gamma_i,$$

mit $z_{i,j} = 0$ für $j \geq l_i$, denn $\operatorname{ord}_G \gamma_i = p^{l_i}$. Wir können dies auch anders aufschreiben (die Indexverwaltung wird im weiteren Verlauf dadurch wesentlich einfacher):

$$\mu = \sum_{i=1}^{k-1} \left(\sum_{j=1}^{s} y_{i,j}p^{l_i - j} \right) \gamma_i$$

mit $y_{i,j} = 0$ für $j > l_i$, denn sonst wäre μ nicht definiert. Wenn wir die Matrix

$$Z = (z_{i,j})_{\substack{i=1,\ldots,k-1 \\ j=0,\ldots,s-1}}$$

mit der Matrix

$$Y = (y_{i,s-j})_{\substack{i=1,\ldots,s \\ j=s,\ldots,1}}$$

(aus Gründen der Anschaulichkeit betrachten wir die Matrix vertikal gespiegelt) vergleichen, so können wir folgendes sagen:

- in beiden Matrizen steht in der i-ten Zeile die p-adische Darstellung von y_i,
- in der Matrix Z stehen am *Ende* der i-ten Zeile $s - l_i$ viele Nullen,
- in der Matrix Y stehen am *Anfang* der i-ten Zeile $s - l_i$ viele Nullen,
- in der Matrix Z stehen in der j-ten Spalte die Koeffizienten von p^{j-1} in den p-adischen Darstellungen und
- in der Matrix Y stehen in der j-ten Spalte die Koeffizienten von $p^{l_i-s-1+j}$ in den p-adischen Darstellungen.

Man kann sich die Indextransformation so vorstellen, daß die Zeilen der Matrix so lange nach rechts geschoben wurden, bis ein von Null verschiedenes Element ganz rechts steht. Der entstehende freie Platz an Anfang der Zeilen wird mit Nullen gefüllt; er ist irrelevant für die Berechnung der Linearkombinationen. Die neue Darstellung hat eine interessante Eigenschaft: die *Ordnung* der Elemente $y_{i,j}p^{l_i-j}\gamma_i$ ist gleich p^j. Dies benutzen wir, um die $y_{i,j}$ zu berechnen, indem wir sukzessive die von Null verschiedenen Einträge der Spalten der Matrix Y durch Projektion in eine Untergruppe der Ordnung p bestimmen.

Projiziere μ nun in eine Untergruppe der Ordnung p:

$$\begin{aligned}
p^{s-1}\mu &= \sum_{i=1}^{k-1}\left(\sum_{j=1}^{s} y_{i,j}p^{s-1+l_i-j}\right)\gamma_i \\
&= \sum_{i=1}^{k-1}\left(\sum_{j=1}^{s-1} y_{i,j}p^{l_i+\overbrace{(s-1-j)}^{\geq 0}} + y_{i,s}p^{l_i-1}\right)\gamma_i \\
&= \sum_{i=1}^{k-1} y_{i,s}\left(p^{l_i-1}\gamma_i\right) \quad \text{mit} \quad y_{i,s} = 0 \text{ für } l_i < s \\
&= \sum_{i:l_i\geq s} y_{i,s}\left(p^{l_i-1}\gamma_i\right);
\end{aligned}$$

die nach der Projektion "übriggebliebenen" $y_{i,s}$ entsprechen gerade den von Null verschiedenen Elementen der Matrix Y in der ersten Spalte.

Für diese $y_{i,s}$ gilt nun:

$$\forall_{i:l_i\geq s} \; 0 \leq y_{i,s} < p, \quad \text{denn} \quad \operatorname{ord}_G p^{l_i-1}\gamma_i = p.$$

Die Bestimmung der Indizes i mit $l_i \geq s$ erfolgt durch Vergleiche der bekannten Zahlen l_i und s. Numeriere diese Indizes mit $i_1, \ldots, i_m$. Die $y_{i,s} \neq 0$ können dann mit einer Baby-Step-Giant-Idee, die ähnlich zur Berechnung von minimalen Relationen ist, berechnet werden. Bezeichne der Übersichtlichkeit halber $p^{l_i-1}\gamma_i$ mit ϵ_i und $p^s\mu$ mit ϵ. Zur Erinnerung: es muß keine Linearkombination für dieses ϵ geben!

5.11. Algorithmus (LK-p-BSGS)

Eingabe: $\epsilon, \epsilon_1, \ldots, \epsilon_m \in G$ $(G \in \mathcal{G})$, $p \in \mathbb{P}$ mit $\operatorname{ord} \epsilon_i = p$ $(1 \leq i \leq m)$.

Ausgabe: Entweder $z_1, \ldots, z_m$ mit $\epsilon = \sum_{i=1}^{m} z_i \epsilon_i$ oder *nicht gefunden*.

$M \leftarrow \emptyset$	(1)
FOR $i \leftarrow 1$ TO m DO $z_i \leftarrow \lceil \sqrt{p} \rceil$	(2)
Berechne $M_{r_m} = \left\{ \left(\epsilon - \sum_{i=1}^{m} r_i \epsilon_i, r_1, \ldots, r_m \right) : 0 \leq r_i \leq z_i - 1, 1 \leq i \leq m \right\}$, sortiert nach der ersten Komponente in $D(G)$	(3)
IF $\exists (\chi, r_1, \ldots, r_m) \in \mathbb{M}_{r_m} : \chi =$	(4)
THEN RETURN $\{(r_1, \ldots, r_m)\}$	(5)
ELSE $M \leftarrow Merge(M, M_{r_m})$	(6)
FORALL $(q_1, \ldots, q_m) \in [\![1, \lfloor \sqrt{p} \rfloor]\!]^m$ (Giant-Steps)	(7)
IF $\exists (\chi, r_1, \ldots, r_m) \in M : \chi = \sum_{i=1}^{m} q_i z_i \gamma_i$ THEN RETURN $\{ (q_1 z_1 + r_1, \ldots, q_m z_m + r_m) \}$	(8)
RETURN { *nicht gefunden* }	(9)

5.12. Satz *Sei $G \in \mathcal{G}$, $\epsilon, \epsilon_1, \ldots, \epsilon_m \in G$, $p \in \mathbb{P}$ und $\operatorname{ord} \epsilon_i = p$, $(i = 1, \ldots, m)$. Algorithmus LK-p benötigt zur Bestimmung der Existenz und gegebenenfalls der Berechnung einer Linearkombination der Form*

$$\epsilon = \sum_{i=1}^{m} z_i \epsilon_i$$

- $O\left(\left(\sqrt{p} \right)^m \right)$ *Gruppenadditionen und*
- $O\left(m \cdot \left(\sqrt{p} \right)^m \cdot \log p \cdot \log n_G \right)$ *Bitoperationen.*

Beweis: Betrachte die Analogie zu Algorithmus 4.3; es gibt 3 Modifikationen zu diesem Algorithmus:

- Das r_m (in Algorithmus 4.3 r_k) ist "eingereiht"; d.h. man sucht nicht nach der Ordnung des Elementes ϵ_m in einer Faktorgruppe, sondern einfach eine Linearkombination von ϵ_i mit

$$\epsilon = \sum_{i=1}^{m} z_i \epsilon_i$$

  und so braucht man nicht nach einem minimalen r_m zu suchen. Es entfällt also die äußere Schleife der Baby-Steps (und der Giant-Steps). Die Anzahl der zu berechnenden Gruppenelemente und somit der durchzuführenden Gruppenoperationen bleibt allerdings gleich.

- Die erste Komponente der Elemente von M wird ersetzt durch $\epsilon - \sum_{i=1}^{m}$, denn man sucht ja eine Linearkombination von ϵ. Demzufolge wird in den Giant-Steps auch $(q_1 z_1 + r_1, \ldots, q_m z_m + r_m)$ anstatt $(q_1 z_1 - r_1, \ldots, q_m z_m - r_m)$ wie in Algorithmus 4.3 ausgegeben und die Grenzen der in Frage kommenden Koeffizienten sind entsprechend geändert; der Aufwand bleibt gleich.

- Die obere Schranke L wird ersetzt durch p, denn man weiß ja, daß die ϵ_i Ordnung p haben und so sind nur Koeffizienten $< p$ interessant.

Ansonsten gelten die Argumente, die im Beweis von Satz 4.4 dargelegt wurden; daraus folgt die Behauptung. ■

Wir können nun noch die Bestimmung der Indizes mit $l_i \geq s$ vorschalten und somit einen Algorithmus angeben, der die ganze erste Spalte der Matrix Y berechnet:

5.13. Algorithmus (LK-p)

Eingabe: $\epsilon, \epsilon_1, \ldots, \epsilon_{k-1} \in G$ $(G \in \mathcal{G})$, $p \in \mathbb{P}$ mit $\operatorname{ord} \epsilon_i = p$ $(1 \leq i \leq k-1)$, $l_1, \ldots, l_{k-1}, s \in \mathbb{N}$.

Ausgabe: Entweder $z_1, \ldots, z_{k-1}$ mit $\epsilon = \sum\limits_{i: l_i \geq s} z_i \epsilon_i$ oder *nicht gefunden*.

Bestimme die Indizes $i_1, \ldots, i_m$ mit $l_{i_j} \geq s$ $(j = 1, \ldots, m)$	(1)
Starte Algorithmus 5.11 mit den Eingaben $(\epsilon, \epsilon_{i_1}, \ldots, \epsilon_{i_m}, p)$.	(2)

Die Bestimmung der Indizes kostet $O(k \cdot \log \max\{l_i : 1 \leq i \leq k-1\}) = O(k \cdot \log n_G)$, also ist die Komplexität dieses Algorithmus nicht größer als die von Algorithmus 5.11.

Wir haben jetzt die erste Zeile der Matrix Y "gefüllt". Mit der folgenden Überlegung lassen sich nun die Einträge der zweiten Spalte bestimmen. Wir nennen das bisher betrachtete μ ab jetzt μ_s. Wir definieren

$$\mu_{s-1} := \mu_s - \sum_{i=1}^{k-1} y_{i,s} p^{l_i - s} \gamma_i;$$

— hier benötigen wir, daß die nicht zur Rechnung beitragenden Einträge in der ersten Spalte der Matrix Y Null sind — dieses μ_{s-1} hat nun Ordnung p^{s-1}, denn:

$$\begin{aligned} \mu_{s-1} &= \sum_{i=1}^{k-1} \left(\sum_{j=1}^{s} y_{i,j} p^{l_i - j} \right) \gamma_i - \sum_{i=1}^{k-1} y_{i,s} p^{l_i - s} \gamma_i \\ &= \sum_{i=1}^{k-1} \left(\sum_{j=1}^{s-1} y_{i,j} p^{l_i - j} \right) \gamma_i \end{aligned}$$

und es gilt:

$$p^{s-1} \mu_{s-1} = \sum_{i=1}^{k-1} \left(\sum_{j=1}^{s-1} y_{i,j} p^{l_i + \overbrace{(s-1-j)}^{\geq 0}} \right) \gamma_i = .$$

Wir können nun also μ_{s-1} durch Multiplizieren mit p^{s-2} in eine Untergruppe der Ordnung p jagen und mit Algorithmus 5.11 die $y_{i,s-1}$ berechnen.

Dieses Verfahren können wir nun iterieren: sei

$$\mu_t = \sum_{i=1}^{k-1} \left(\sum_{j=1}^{t} y_{i,j} p^{l_i - j} \right) \gamma_i .$$

Dann gilt

$$p^t \mu_t =$$

und

$$p^{t-1} \mu_t = \sum_{i=1}^{k-1} y_{i,t} \left(p^{l_i - 1 \gamma_i} \right)$$

sowie

$$\mu_{t-1} = \mu_t - \sum_{i=1}^{k-1} y_{i,t} p^{l_i - t} \gamma_i,$$

wobei die $y_{i,t}$ entweder Null sind (falls $l_i < t$) oder durch Algorithmus 5.11 berechnet werden. Nach s Iterationen haben wir alle $y_{i,j}$ bestimmt. Wir haben also:

5.14. Algorithmus (LK-p^s)

Eingabe: $\mu, \gamma_1, \ldots, \gamma_{k-1} \in G$ ($G \in \mathcal{G}$), $p \in \mathbb{P}$, $l_1, \ldots, l_{k-1} \in \mathbb{N}$ mit

$$\mathrm{ord}_G\, \gamma_i = p^{l_i} \quad (i = 1, \ldots, k-1).$$

Ausgabe: Entweder $y_1, \ldots, y_{k-1} \in \mathbb{N}$ mit $\mu = \sum_{i=1}^{k-1} y_i \gamma_i$ oder *nicht gefunden.*

$s \leftarrow \max\{l_i : 1 \leq i \leq k-1\}$	(1)
FOR $i \leftarrow 1$ TO $k-1$ DO $y_{i,1} \leftarrow 0, \ldots, y_{i,s} \leftarrow 0$	(2)
FOR $i \leftarrow 1$ TO $k-1$ DO $\epsilon_i \leftarrow p^{l_i - 1}\gamma_i$; Berechne dabei wie in Algorithmus 5.9 alle $p^i\gamma_i$ und speichere sie ab	(3)
FOR $t \leftarrow s$ DOWNTO 1	(4)
$\epsilon \leftarrow p^{t-1}\mu$ mit Algorithmus 3.2	(5)

$(y_{1,t}, \ldots, y_{k-1,t}) \leftarrow LK - p(\epsilon, \epsilon_1, \ldots, \epsilon_{k-1}, p, l_1, \ldots, l_{k-1}, t)$;
    falls Ausgabe von Algorithmus 5.13 "*nicht gefunden*",
    RETURN { *nicht gefunden* }
	(6)

$\mu \leftarrow \mu - \sum_{i=1}^{k-1} y_{i,t} p^{l_i - t}\gamma_i$	(7)
FOR $i \leftarrow 1$ TO $k-1$	(8)
$y_i \leftarrow 0$; $j \leftarrow 1$	(9)
WHILE ($j \leq \min\{l_i, s\}$ DO $y_i \leftarrow py_i + y_{i,j}$; $j \leftarrow j+1$	(10)
RETURN { $(y_1, \ldots, y_k)$ }	(11)

Die Korrektheit des Algorithmus folgt aus der Konstruktion. Es gilt:

5.15. Satz *Sei $G \in \mathcal{G}$ p-Gruppe, $\gamma_1, \ldots, \gamma_{k-1} \in G$, $\mu \in G$, p^s die maximale Ordnung in $< \gamma_1, \ldots, \gamma_{k-1} >$. Dann benötigt Algorithmus 5.14 zur Bestimmung der Existenz einer Linearkombination*

$$\mu = \sum_{i=1}^{k-1} y_i \gamma_i$$

- $O(s \cdot (\sqrt{p})^{k-1} + s^2 \cdot \log p + s \cdot (k-1) \cdot \log p)$ *Gruppenadditionen und*
- $O(s \cdot (k-1) \cdot (\sqrt{p})^{k-1} \cdot \log p \cdot \log n_G)$ *Bitoperationen.*

Beweis: Wir geben die benötigten Operationen der einzelnen Schritte an; die Behauptung ergibt sich dann durch Majorisieren bzw. Addieren der Komplexitäten.

- Der Verwaltungsaufwand von Schritt (1) und (2) ist, im Vergleich zum restlichen Aufwand, offensichtlich vernachlässigbar.

- Schritt (3) benötigt — analog zu Algorithmus 5.9 — $O((l_i - 1) \cdot \log p)$ Gruppenadditionen für jedes $i = 1, \ldots, k-1$; also ist der Aufwand für diesen Schritt $O((k-1) \cdot s \cdot \log p)$ Gruppenadditionen.
- Schritt (5) benötigt mit schneller Multiplikation $O((t-1) \cdot \log p)$ Gruppenadditionen.
- Schritt (6) ist die Anwendung von Algorithmus 5.13, benötigt also
  - $O((\sqrt{p})^{k-1})$ Gruppenadditionen und
  - $O((\sqrt{p})^{k-1} \cdot (k-1) \cdot \log p \cdot \log n_G)$ Bitoperationen.
- Schritt (7) kann mit den vorberechneten $p^i \gamma_i$ und schneller Multiplikation in $O((k-1) \cdot \log p)$ Gruppenadditionen realisiert werden.
- Schritt (4) benötigt also insgesamt
  - $O(s \cdot (\sqrt{p})^{k-1} + s^2 \cdot \log p + s \cdot (k-1) \cdot \log p)$ Gruppenadditionen und
  - $O((s \cdot \sqrt{p})^{k-1} \cdot (k-1) \cdot \log p \cdot \log n_G)$ Bitoperationen.
- Schritt (8) ist ein geschicktes Hornerschema; dies kann man in $O(s \cdot (k-1) \cdot \log^2 p)$ Bitoperationen realisiert werden (es gilt sicher: $\log p = O(\log n_G)$). ■

Jetzt können wir minimale Relationen in p-Gruppen berechnen. Wir probieren sukzessive, eine Linearkombination für $p^i \gamma_k$ in $< \gamma_1, \ldots, \gamma_{k-1} >$ zu berechnen. Falls dies nicht erfolgreich ist, so ist $\mathrm{ord}_{G/G_k} \gamma_k > p^i$ und wir wiederholen den Versuch mit $i+1$. Falls die Berechnung einer Linearkombination erfolgreich ist, sind wir fertig. Betrachte nun noch

5.16. Lemma *Sei G eine p-Gruppe, $G_k \leq G$, $\gamma_k \in G \backslash G_k$, $\mathrm{ord}_G \gamma_k = p^{l_k}$, $\mathrm{ord}_{G/G_k} \gamma_k = p^x$ und $\mathrm{ord}_{G_k}(p^x \gamma_k) = p^z$. Dann gilt:*

$$0 \leq z \leq s \quad \Longrightarrow \quad \max\{0, l_k - s\} \leq x \leq l_k.$$

Beweis: Es gilt für alle endlichen Gruppen G mit Untergruppe G_k und $\gamma_k \in G$:

$$\begin{aligned} \mathrm{ord}_G \gamma_k &= \mathrm{ord}_{G/G_k} \gamma_k \cdot \mathrm{ord}_G \left(\left(\mathrm{ord}_{G/G_k} \gamma_k \right) \gamma_k \right) \\ &= \mathrm{ord}_{G/G_k} \gamma_k \cdot \mathrm{ord}_{\mathbf{G_k}} \left(\left(\mathrm{ord}_{G/G_k} \gamma_k \right) \gamma_k \right). \end{aligned}$$

Daraus folgt:

$$p^{l_k} = p^z \left(p^x \gamma_k \right),$$

also

$$l_k = z + x \quad \Longleftrightarrow \quad x = l_k - z.$$

Daraus folgt unmittelbar die Behauptung. ■

Dieses Lemma gibt uns die Begründung für die Schleifengrenzen in dem folgenden Algorithmus:

5.17. Algorithmus (Minimale Relationen in p-Gruppen)

Eingabe: $\gamma_1, \ldots, \gamma_k \in G$ $(G \in \mathcal{G})$, $p \in \mathbb{P}$.

Ausgabe: $(x_1, \ldots, x_k)$ minimale Relation für $(\gamma_1, \ldots, \gamma_k)$.

FOR $j \leftarrow 1$ TO k DO Berechne l_j so daß $p^{l_j} = \mathrm{ord}_G \gamma_j$ mit Algorithmus 5.9		(1)
FOR $i \leftarrow \max\{0, l_k - max\{l_i : 1 \leq i \leq k-1\}\}$ TO l_k		(2)
	Starte Algorithmus 5.14 mit den Eingaben $(p^i \gamma_k, \gamma_1, \ldots, \gamma_{k-1}, p, l_1, \ldots, l_{k-1})$; falls die Berechnung einer Linearkombination $(y_1, \ldots, y_{k-1})$ erfolgreich ist, RETURN $\{ (y_1, \ldots, y_{k-1}, p^i) \}$	(3)

5.18. Satz *Sei $G \in \mathcal{G}$ p-Gruppe, $\gamma_1, \ldots, \gamma_k \in G$, $s \in \mathbb{N}$ minimal, so daß $p^s \gamma_i =$ für alle $i = 1, \ldots, k$. Dann kann man eine minimale Relation für $(\gamma_1, \ldots, \gamma_k)$ in*

- $O(s^2 \cdot (\sqrt{p})^{k-1} + s^3 \cdot \log p + s^2 \cdot k \cdot \log p)$ *Gruppenadditionen und*
- $O(s^2 \cdot (k-1) \cdot (\sqrt{p})^{k-1} \cdot \log p \cdot \log n_G)$ *Bitoperationen*

berechnen.

Beweis: Die Korrektheit des Algorithmus geht aus der Konstruktion hervor.
Schritt (1) benötigt k Anwendungen von Algorithmus 5.9; jede gefundene Ordnung ist $\leq p^s$, also ist der Aufwand insgesamt $O(s \cdot k \cdot \log p)$. In Schritt (2) ist die Anzahl der Iterationen sicher $\leq s$, somit ergibt sich mit Satz 5.15 die Behauptung. ∎

Eine iterative Berechnung der minimalen Relationen ist hier nicht sinnvoll; erstens würde der Aufwand beträchtlich erhöht, um die Ordnungen der γ_i in den Faktorgruppen G/G_i zu kennen; zweitens bringt diese Kenntnis bei der Realisierung der Berechnung einer minimalen Relation in einer p-Gruppe nichts, da man die *tatsächliche* Ordnung eines jeden γ_i benötigt, um die Projektion in eine Gruppe der Ordnung p vorzunehmen. Auch ist die Kenntnis der Gruppenordnung hier nicht nützlich; die Ideen des letzten Kapitels bringen in p-Gruppen keine Verbesserung.

Wir geben hier nun noch ein Resultat an, das das obige Ergebnis für das Problem des disrekten Logarithmus angibt:

5.19. Satz *Sei $G \in \mathcal{G}$ p-Gruppe, $\alpha, \beta \in G$, $\operatorname{ord} alpha = p^a$ und $\operatorname{ord} \beta = p^b$; sei $s = \max\{a, b\}$. Dann kann man die Lösbarkeit von*

$$\alpha = x\beta,$$

in

- $O(s \cdot \sqrt{p} + s^2 \cdot \log p)$ *Gruppenadditionen und*
- $O(s \cdot \sqrt{p} \cdot \log p \cdot n_G)$ *Bitoperationen*

entscheiden und gegebenenfalls eine Lösung bestimmen.

Beweis: Betrachte Algorithmus 4.18. Die Bestimmung der Ordnung von β kann mit Algorithmus 5.9 vollzogen werden; der zweite Teil des Algorithmus kann mit Algorithmus 5.14 berechnet werden; die Behauptung folgt aus Satz 5.10 und Satz 5.15 durch majorisieren. ∎

5.4 Zusammensetzen der Ergebnisse

Wir können nun in einer beliebigen endlichen abelschen Gruppe minimale Relationen mit Kenntnis der Primfaktorzerlgung der Gruppenordnung berechnen, indem wir die Elemente $\gamma_1, \ldots, \gamma_k$ sukzessive in die maximale p-Gruppe für jeden Printeiler p der Gruppenordnung transportieren, dort minimale Relationen berechnen und dann die Ergebnisse mit den Chinesischen Rest-Algorithmus zusammensetzen.

5.20. Algorithmus (Minimale Relationen modular)

Eingabe: $\gamma_1, \ldots, \gamma_k \in G$, $|G| = h = \prod_{i=1}^{r} p_i^{e_i}$, $h_i = \frac{h}{p_i^{e_i}}$.

Ausgabe: $(x_1, \ldots, x_k)$ minimale Relation für $(\gamma_1, \cdots \gamma_k)$.

FOR $j \leftarrow 1$ TO r DOa $(\gamma_{1,j},\ldots,\gamma_{k,j}) \leftarrow (h_j\gamma_1,\ldots,h_j\gamma_k)$		(1)
FOR $j \leftarrow 1$ TO r		(2)
	Berechne $(x_{1,j},\ldots,x_{k,j})$ minimale Relation für $(\gamma_{1,j},\ldots,\gamma_{k,j})$ mit Algorithmus 5.17 und den Eingaben $(\gamma_{1,j},\ldots,\gamma_{k,j},p_j)$	(3)
FOR $i \leftarrow 1$ TO k		(4)
	Berechne die Lösung x_i der simultanen Kongruenzen $$x_i \equiv x_{i,j} \bmod p_j^{e_j} \quad (j=1,\ldots,r)$$ mit Algorithmus 5.7 und den Eingaben $(p_1^{e_1},\ldots,p_s^{e_s},x_{i,1},\ldots,x_{i,s})$	(5)
RETURN $\{\ (x_1,\ldots,x_k)\ \}$		(6)

Die Korrektheit des Algorithmus ergibt sich aus Satz 5.3. Wir haben somit:

5.21. Satz *Sei $G \in \mathcal{G}$, $\gamma_1,\ldots,\gamma_k \in G$, $h=|G|=\prod_{j=1}^r p_j^{e_j}$, $e=\max\{e_j : 1\le j\le r\}$. Dann läßt sich unter der Kenntnis der Primfaktorzerlegung der Gruppenordnung eine minimale Relation für $(\gamma_1,\ldots,\gamma_k)$ in*

- $O\left(\sum_{j=1}^r \left(e_j^2\cdot(\sqrt{p_j})^{k-1}\right)+e^2\cdot\log h+e\cdot k\cdot\log h\right)$ *Gruppenadditionen und*
- $O\left(\sum_{j=1}^r \left(e_j^2\cdot(k-1)\cdot(\sqrt{p_j})^{k-1}\cdot\log p_j\cdot\log n_G\right)+s\cdot\log^2 h\right)$ *Bitoperationen*

berechnen.

Beweis: Schritt (1) bewältigt für $j=1,\ldots,s$ jeweils k schnelle Multiplikationen mit h_j; der Aufwand ist also $O\left(\sum_{j=1}^s k\cdot\log h\right)$ Gruppenadditionen. Dies wird durch die Komplexität von Schritt (3) majorisiert: der Aufwand für Schritt (3) beträgt nämlich

$$\begin{aligned}
&\sum_{j=1}^s O\left(e_j^2\cdot(\sqrt{p_j})^{k-1}+e_j^3\cdot\log p_j+e_j^2\cdot k\cdot\log p_j\right)\\
=\;& O\left(\sum_{j=1}^s\left(e_j^2\cdot(\sqrt{p_j})^{k-1}\right)+e^2\cdot\sum_{j=1}^s e_j\cdot\log p_j+e\cdot k\cdot\sum_{j=1}^s e_j\cdot\log p_j\right)\\
=\;& O\left(\sum_{j=1}^s\left(e_j^2\cdot(\sqrt{p_j})^{k-1}\right)+e^2\cdot\prod_{j=1}^s\log p_j^{e_j}+e\cdot k\cdot\prod_{j=1}^s\log p_j^{e_j}\right)\\
=\;& O\left(\sum_{j=1}^s\left(e_j^2\cdot(\sqrt{p_j})^{k-1}\right)+e^2\cdot\log h+e\cdot k\cdot\log h\right)
\end{aligned}$$

Gruppenadditionen; der Aufwand an Bitoperationen ergibt sich direkt durch Summation und folgende Tatsache: die Zusammensetzung mit dem Chinesischen Rest-Algorithmus in Schritt (4) kostet aufgrund von Satz 5.8 $O(s\cdot\log^2 h)$ Bitoperationen. ■

Interpretation: Die Idee, durch die Kenntnis der Primfaktorzerlegung in *jedem* Fall schneller minimale Relationen berechnen zu können, ist sehr verlockend. Allerdings zeigt die obige Komplexitätsaussage, daß diese Version nicht in jedem Fall vorzuziehen ist; reduziert sie sich doch für den Fall einer Primgruppe auf Algorithmus 4.3 zur Bestimmung einer minimalen Relation aus dem vorherigen Kapitel mit in der Gruppenordnung polynomiellen Aufwand. Dieser Algorithmus kann seine Stärke nur dann ausspielen, wenn die Gruppenordnung in viele kleine Primfaktoren mit hohen Exponenten zerfällt; dann ist die Laufzeit logarithmisch in der Gruppenordnung. Ist allerdings ein großer Primfaktor dabei, so wird die Laufzeit polynomiell von diesem Faktor dominiert ($O(p^{k/2})$). Der Aufwand für die iterative Berechnung einer minimalen Relation ist hingegen unabhängig von der Primfaktorzerlegung der Gruppenordnung ($O(|G|^{1/2})$). Es ist also abzuwägen, für welchen Algorithmus man sich entscheidet.

Kapitel 6

Bestimmung der Struktur einer Gruppe

In diesem Kapitel werden wir erläutern, wie man die Struktur und eine Basis einer endlichen abelschen Gruppe berechnen kann. Hierzu benötigen wir die *Smith-Reduktion*, die wir vorstellen werden. Weiterhin zeigen wir, daß wir die Struktur einer Gruppe berechnen können, wenn wir "genügend" minimale Relationen zur Verfügung haben. Im Anschluß daran werden wir Algorithmen angeben, die unter verschiedenen Voraussetzungen die Struktur der betrachteten Gruppe berechnen und schließlich die Berechnung der Gruppenordnung als Spezialfall der behandelten Algorithmen ernten.

Ein deterministisches Verfahren, die Ordnung einer beliebigen Gruppe mit den Operationen unserer algorithmischen Umgebung zu berechnen, ist bisher nicht bekannt. Hierzu müßte man auf systematische Weise Elemente der betrachteten Gruppe erzeugen können; diese Systematik müßte unabhängig von der Gruppenstruktur sein. Für spezielle Gruppen ist dies wohl möglich, z.B. wenn die Darstellungen der Gruppenelemente in einem geordneten Bereich liegen; dies ist aber im allgemeinen nicht der Fall. Wir nehmen an, daß wir keine Information dieser Art über die Darstellung der Gruppenelemente zur Verfügung haben.

6.1 Die Smith-Normalform und die Struktur einer Gruppe

Wir geben hier zuerst einige Fakten über die Smith-Normalform einer Matrix und deren Zusammenhang mit Gittern an.

6.1. Definition *Eine Matrix $S = (s_{i,j}) \in \mathbb{Z}^{n \times n}$ hat* **Smith-Normalform**, *wenn gilt:*

- $s_{i,j} = 0$ *für alle* $i, j \in [\![1, n]\!]$ *mit* $i \neq j$,
- $s_{i,i} > 0$ *für alle* $i \in [\![1, n]\!]$ *und*
- $s_{i,i} | s_{i+1,i+1}$ *für alle* $i \in [\![1, n-1]\!]$.

6.2. Definition *Eine Matrix $U \in \mathbb{Z}^{n \times n}$ heißt* **unimodular**, *wenn $|\det U| = 1$. Die Matrix $A \in \mathbb{Z}^{n \times n}$ geht durch* **unimodulare Transformationen** *in die Matrix $B \in \mathbb{Z}^{n \times n}$ über, wenn es eine unimodulare Matrix U gibt mit $B = U \cdot A$.*

Die folgenden beiden Sätze sind [Dü92] entnommen:

6.3. Satz *Zu jeder regulären Matrix* $A \in \mathbb{Z}^{n\times n}$ *gibt es unimodulare Matrizen* $U, V \in \mathbb{Z}^{n\times n}$, *so daß* $S = U \cdot A \cdot V$ *Smith-Normalform hat.*

6.4. Satz *Sei* $A \in \mathbb{Z}^{n\times n}$ *regulär und seien* $U, V \in \mathbb{Z}^{n\times n}$ *unimodular, so daß* $S = U \cdot A \cdot V$ *Smith-Normalform hat. Seien* $s_1, \ldots, s_n$ *die Diagonaleinträge von* S. *Dann gilt:*

$$\mathbb{Z}^n / \mathcal{L}(A) \cong \mathbb{Z}/s_1\mathbb{Z} \times \cdots \times \mathbb{Z}/s_n\mathbb{Z}.$$

Wir stellen nun den Zusammenhang her zwischen minimalen Relationen und der Struktur einer Gruppe:

6.5. Lemma *Sei* G *eine endliche abelsche Gruppe,* $G = <\gamma_1, \ldots, \gamma_n>$ *und sei* $A \in \mathbb{Z}^{n\times n}$ *die Matrix, deren Spaltenvektoren*

$$(x_{1,i}, \ldots, x_{i-1,i}, y_i, 0, \ldots, 0)^T$$

minimale Relationen $(x_{1,i}, \ldots, x_{i-1,i}, y_i)$ *für* $(\gamma_1, \ldots, \gamma_i)$ *enthalten. Dann gilt:*

$$\mathbb{Z}^n / \mathcal{L}(A) \cong G.$$

Dieser Zusammenhang wurde bereits im Beweis von Lemma 4.8 gezeigt. Es folgt sofort:

6.6. Korollar *Sei* G *eine endliche abelsche Gruppe,* $G = <\gamma_1, \ldots, \gamma_n>$ *und sei* $A \in \mathbb{Z}^{n\times n}$ *die Matrix, deren Spaltenvektoren*

$$(x_{1,i}, \ldots, x_{i-1,i}, y_i, 0, \ldots, 0)^T$$

minimale Relationen $(x_{1,i}, \ldots, x_{i-1,i}, y_i)$ *für* $(\gamma_1, \ldots, \gamma_i)$ *enthalten. Seien* $U, V \in \mathbb{Z}^{n\times n}$ *unimodular, so daß* $S = U \cdot A \cdot V$ *Smith-Normalform hat und seien* $s_1, \ldots, s_n$ *die Diagonaleinträge von* S. *Dann gilt:*

$$G \cong \mathbb{Z}/s_i\mathbb{Z} \times \cdots \times \mathbb{Z}/s_n\mathbb{Z},$$

wobei $i = \min\{j : 1 \le j \le n,\ s_j > 1\}$.

Da die Struktur einer Gruppe eindeutig ist, muß $(s_i, \ldots, s_n)$ die Struktur von G sein. Man berechnet also die Struktur einer endlichen abelschen Gruppe dadurch, daß man versucht, $\gamma_1, \ldots, \gamma_n \in G$ zu bekommen mit

$$G = <\gamma_1, \ldots, \gamma_n>,$$

sukzessive minimale Relationen für $<\gamma_1, \ldots, \gamma_i>$ berechnet und für die so entstehende Matrix die Smith-Normalform berechnet.

6.2 Die Smith-Normalform und eine Basis einer Gruppe

Wir wollen nicht nur die Struktur, sondern auch eine Basis einer endlichen abelschen Gruppe berechnen. Auch hier gibt es einen natürlichen Zusammenhang:

6.7. Lemma *Sei* G *eine endliche abelsche Gruppe,* $G = <\gamma_1, \ldots, \gamma_n>$ *und sei* $A \in \mathbb{Z}^{n\times n}$ *die Matrix, deren Spaltenvektoren* $(x_{1,i}, \ldots, x_{i-1,i}, y_i, 0, \ldots, 0)^T$ *minimale Relationen* $(x_{1,i}, \ldots, x_{i-1,i}, y_i)$ *für* $(\gamma_1, \ldots, \gamma_i)$ *enthalten. Seien* $U, V \in \mathbb{Z}^{n\times n}$ *unimodular, so daß* $S = U \cdot A \cdot V$ *Smith-Normalform hat und seien* $s_1, \ldots, s_n$ *die Diagonaleinträge von* S. *Seien* $\delta_1, \ldots, \delta_n \in G$ *so daß*

$$(\delta_1, \ldots, \delta_n) = (\gamma_1, \ldots, \gamma_n) \cdot U^{-1}.$$

Dann ist $(\delta_i, \ldots, \delta_n)$ *mit* $i = \min\{j : 1 \le j \le n,\ s_j > 1\}$ *eine Basis von* G.

Beweis: Zeige zuerst $G =< \delta_1, \dots, \delta_n >$.
$< \delta_1, \dots, \delta_n > \subseteq G$ klar, denn

$$(\delta_1, \dots, \delta_n) = (\gamma_1, \dots, \gamma_n) \cdot U^{-1} \in G^n.$$

Sei andererseits $x \in G$,

$$x = \sum_{i=1}^{n} x_i \gamma_i.$$

Dann ist (da U unimodular, also invertierbar):

$$\begin{aligned} x &= (x_1, \dots, x_n) \cdot (\gamma_1, \dots, \gamma_n)^T \\ &= (x_1, \dots, x_n) \cdot ((\delta_1, \dots, \delta_n) \cdot U)^T \\ &= \underbrace{(x_1, \dots, x_n) \cdot U^T}_{\in \mathbb{Z}^n} \cdot (\delta_1, \dots, \delta_n)^T \in < \delta_1, \dots, \delta_n > . \end{aligned}$$

Betrachte nun die Abbildung $\phi : G =< \delta_1, \dots, \delta_n > \longrightarrow \mathbb{Z}/\mathcal{L}(S)$ definiert durch

$$\phi\left(\sum_{j=1}^{n} x_i \delta_i\right) = (x_1 \bmod s_1, \dots, x_n \bmod s_n).$$

ϕ ist trivialerweise Homomorphismus. Wir zeigen zuerst

$$\forall_{j \in [\![1,n]\!]} \quad s_j \delta_j =$$

Dies ist äquivalent zu

$$(\delta_1, \dots, \delta_n) \cdot S = (0, \dots, 0),$$

und es gilt:

$$\begin{aligned} (\delta_1, \dots, \delta_n) \cdot S &= (\gamma_1, \dots, \gamma_n) \cdot U^{-1} \cdot (U \cdot A \cdot V) \\ &= ((\gamma_1, \dots, \gamma_n) \cdot A) \cdot V \\ &= (0, \dots, 0) \cdot V \\ &= (0, \dots, 0). \end{aligned}$$

Also haben wir

$$\begin{aligned} \operatorname{Kern} \phi &= \left\{ x : x = \sum_{i=1}^{n} x_i \delta_i \text{ mit } s_j | x_j \text{ für alle } j = 1, \dots, n \right\} \\ &= \{\}. \end{aligned}$$

Da sowieso $G \cong \mathbb{Z}^n/\mathcal{L}(A) = \mathbb{Z}^n/\mathcal{L}(S)$, ist folglich ϕ Isomorphismus. Also gilt für $i \in [\![1, n]\!]$:

$$< \delta_i > \quad \cong \quad < \phi(\delta_i) > \quad = \quad < (0, \dots, 0, \overbrace{1}^{i}, 0, \dots, 0) > \quad \cong \quad \mathbb{Z}/_{s_i \mathbb{Z}}$$

und für $i, j \in [\![1, n]\!]$, $i \neq j$:

$$\begin{aligned} & x \in < \delta_i > \cap < \delta_j > \\ \Longrightarrow \quad & \phi(x) \in < (0, \dots, 0, \overbrace{1}^{i}, 0, \dots, 0) > \cap < (0, \dots, 0, \overbrace{1}^{j}, 0, \dots, 0) > \\ \Longrightarrow \quad & \phi(x) = (0, \dots, 0) \\ \Longrightarrow \quad & x = \end{aligned}$$

und somit hätten wir unsere Behauptung gezeigt. ■

Wir können nun einen Algorithmus angeben, der mit der unimodularen Transformationsmatrix U, die bei der Smith-Reduktion mitberechnet werden kann, aus einem Erzeugendensystem einer Gruppe eine Basis berechnet. Für die Matrix M bezeichne $m_{i,j}$ den Eintrag in der i-ten Zeile und der j-ten Spalte.

6.8. Algorithmus (Basis einer Gruppe)

Eingabe: $\gamma_1, \ldots, \gamma_n \in G$ $(G \in \mathcal{G})$, $A, U, S \in \mathbb{Z}^{n \times n}$ mit $S = U \cdot A \cdot V$ in Smith-Normalform, wobei in der i-ten Spalte

$$(x_{1,i}, \ldots, x_{i-1,i}, y_i, 0, \ldots, 0)^T$$

von A eine minimale Relation $(x_{1,i}, \ldots, x_{i-1,i}, y_i)$ für $(\gamma_1, \ldots, \gamma_i)$ enthalten ist.

Ausgabe: $\delta_1, \ldots, \delta_m$ Basis von $< \gamma_1, \ldots, \gamma_n >$.

$D \leftarrow U^{-1}$	(1)
$i \leftarrow \min\{j : s_{j,j} \neq 1\}$	(2)
FOR $j \leftarrow i$ TO n	(3)
$(z_1, \ldots, z_n) \leftarrow (d_{1,j}, \ldots, d_{n,j})$	(4)
FOR $k \leftarrow n$ DOWNTO 1	(5)
Berechne q, r in der Division mit Rest $z_k = q \cdot a_{k,k} + r$	(6)
$(z_1, \ldots, z_k) \leftarrow (z_1 + q \cdot a_{1,k}, \ldots, z_{k-1} + q \cdot a_{k-1,k}, r)$	(7)
$\delta_j \leftarrow \sum_{l=1}^{n} z_l \gamma_l$	(8)
RETURN $\{ \delta_i, \ldots, \delta_n \}$	(9)

Zur Korrektheit:

1. Man bestimmt natürlich nur die δ_j, deren Ordnung > 1 ist; daher beginnt man erst mit $\min\{j : s_j \neq 1\}$.
2. Die Berechnung der Linearkombination
$$\delta_j = \sum_{l=1}^{n} d_{l,j} \gamma_l \quad \text{denn} \ ,$$
die sich aus $(\delta_1, \ldots, \delta_n) = (\gamma_1, \ldots, \gamma_n) \cdot U^{-1}$ ergibt, wird in Schritt (4) bis Schritt (8) so ausgeführt, daß man möglichst wenig Gruppenoperationen durchführen muß. Man geht dabei vor wie im Beweis von Lemma 4.6.
3. Die tatsächliche Berechnung
$$\sum_{l=1}^{n} z_l \gamma_l$$
wird mit n schnellen Multiplikationen ausgeführt.

6.9. Satz *Sei $G \in \mathcal{G}$, $\gamma_1, \ldots, \gamma_n \in G$, $h = | < \gamma_1, \ldots, \gamma_n > |$. Dann kann man eine Basis $(\delta_1, \ldots, \delta_m)$ von $< \gamma_1, \ldots, \gamma_n >$ unter der Kenntnis der wie oben beschriebenen Matrix A, deren Smith-Normalform S und der unimodularen Transformationsmatrix U mit $S = U \cdot A \cdot V$ in*

- $O(s \cdot (\log h + n))$ *Gruppenadditionen und*

- $O(n^3)$ *arithmetischen Operationen*

berechnen.

Beweis: Für jedes δ_j muß man

$$\begin{aligned}\sum_{l=1}^{n} O(\log z_i) + (n-1) &= \sum_{l=1}^{n} O(\log a_{l,l}) + (n-1)\\ &= O\left(\log\left(\prod_{l=1}^{n} a_{l,l}\right)\right) + O(n)\\ &= O(\log h + n)\end{aligned}$$

viele Gruppenoperationen ausführen; der Aufwand an arithmetischen Operationen wird durch die Inversenberechnung der Matrix U bestimmt. ■

6.3 Die Smith-Reduktion

Den Vorgang, zu einer gegebenen regulären Matrix $A \in \mathbb{Z}^{n\times n}$ die zugehörige Matrix S in Smith-Normalform zu berechnen, bezeichnet man mit **Smith-Reduktion**. Wir werden hier ein Verfahren angeben, das die beiden unimodularen Transformationsmatrizen U und V mit $S = U \cdot A \cdot V$ mit ausgibt.

Ein elementarer Schritt der Smith-Reduktion besteht darin, in der ersten Zeile rechts vom Diagonalelement Nullen zu erzeugen, ohne das von der Matrix erzeugte Gitter zu verändern; die Transformationsmatrix dieses elementaren Schrittes muß also unimodular sein. Es bezeichne $a_{i,j}$ das Matrixelement in der i-ten Zeile und j-ten Spalte von A und I die Einheitsmatrix.

6.10. Algorithmus (UNIMOD-ZT)
Eingabe: $A \in \mathbb{Z}^{n\times n}$ regulär.
Ausgabe: $B \in \mathbb{Z}^{n\times n}$ mit $\mathcal{L}(A) = \mathcal{L}(B)$ und $b_{1,1} > 0$ sowie $b_{1,j} = 0$ für $j = 1, \ldots, n$.

IF $a_{1,1} = 0$ THEN vertausche Spalte 1 mit einer Spalte, dessen oberstes Element $\neq 0$ ist (eine solche Spalte existiert, sonst wäre A nicht regulär). (1)

FOR $j \leftarrow 2$ TO n (2)

  $(r, u, v) \leftarrow \mathit{ERWEUKLALG}(a_{1,1}, a_{1,j})$ (3)

  (4)

$$A \leftarrow A \cdot \begin{array}{r} \\ \\ \\ \\ j\{ \\ \\ \\ \end{array} \underbrace{\left(\begin{array}{ccccccc} u & 0 & \cdots & 0 & \overbrace{-\frac{a_{1,j}}{r}}^{j} & 0 & \cdots \\ 0 & 1 & & & 0 & & \\ \vdots & & \ddots & & \vdots & & \\ 0 & & & 1 & 0 & & \\ v & 0 & \cdots & 0 & \frac{a_{1,1}}{r} & 0 & \cdots \\ 0 & & & & 0 & 1 & \\ \vdots & & & & \vdots & & \ddots \end{array}\right)}_{= ZT(A,u,v,j)}$$

Sei $x^{(j)}$ der Wert der Variablen x nach dem j-ten Schleifendurchlauf. $ZT(A,u,v,j)$ ist unimodular, also ist die Folge der Transformationen unimodular. Es gilt für alle $j = 1,\ldots,n$:

$$a_{1,j}^{(j)} = \frac{a_{1,1}^{(j-1)}}{r} a_{1,j}^{(j-1)} - \frac{a_{1,j}^{(j-1)}}{r} a_{1,1}^{(j-1)} = 0$$

sowie

$$a_{1,1}^{(j)} = u^{(j)} a_{1,1}^{(j-1)} + v^{(j)} a_{1,j}^{(j-1)} = r^{(j)} > 0.$$

Daraus folgt die Korrektheit von Algorithmus 6.10.

6.11. Satz *Sei $A \in \mathbb{Z}^{n\times n}$ regulär. Algorithmus 6.10 benötigt, angewandt auf A,*

- *$O(n)$ Anwendungen des erweiterten Euklidischen Algorithmus und*
- *$O(n^2)$ arithmetische Operationen.*

Beweis: Der erste Teil der Behauptung ist trivial. Zum zweiten Teil: die Multiplikation von A mit $ZT(A,u,v,j)$ ist de facto keine Matrixmultiplikation; die tatsächliche Berechnung begrenzt sich auf

$$(A_1, A_j) \leftarrow (uA_1 + vA_j, \frac{a_{1,1}}{r} A_j - \frac{a_{1,j}}{r} A_1),$$

wobei A_i der i-te Spaltenvektor von A ist. Der Aufwand für jede dieser Zuweisungen von Spaltenvektoren beträgt $O(n)$ arithmetische Operationen. ■

Analog kann man einen Algorithmus UNIMOD-ST angeben, der alle Elemente der ersten Spalte von A bis auf das erste Element durch unimodulare Transformationen in der oben beschriebenen Weise "annulliert"; dabei wird in Schritt (4) folgende Matrix von *links* an A heranmultipliziert:

$$\begin{array}{cc}
 & \begin{array}{ccccccc} \phantom{-\frac{a_{j,1}}{r}} & & & & \overbrace{}^{j} & & \end{array} \\
j\{ & \left(\begin{array}{ccccccc}
p & 0 & \cdots & 0 & q & 0 & \cdots \\
0 & 1 & & & 0 & & \\
\vdots & & \ddots & & \vdots & & \\
0 & & & 1 & 0 & & \\
-\frac{a_{j,1}}{r} & 0 & \cdots & 0 & \frac{a_{1,1}}{r} & 0 & \cdots \\
0 & & & & 0 & 1 & \\
\vdots & & & & \vdots & & \ddots
\end{array}\right)
\end{array}$$

Diese Matrix bezeichnen wir mit $ST(A,u,v,j)$.

Wenn wir nun A in eine Matrix der Gestalt

$$\begin{pmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \cdots & * \end{pmatrix} \tag{6.1}$$

transformieren wollen, müssen wir möglicherweise mehrere Male sukzessive die beiden obigen Algorithmen auf A anwenden (was wir nun einen *Schritt* nennen), denn z.B. UNIMOD-ST kann, nach einer vorherigen Anwendung von UNIMOD-ZT, Einträge der ersten Zeile wieder mit von Null verschiedenen Elementen ersetzen. Das passiert aber nur dann, wenn durch die Anwendung einer der beiden Algorithmen $a_{1,1}$ tatsächlich verändert wird (wenn $r = a_{1,1}$, so gilt $ZT(Au,v,j) = ST(A,u,v,j) = I$), also mindestens halbiert wird (der "neue" Eintrag teilt immer den "alten"). Also ist das oberste Diagonalelement nach spätestens $\log a_{1,1}$ Schritten gleich $\mathrm{ggT}(a_{1,1}, \ldots, a_{1,n}, a_{2,1}, \ldots, a_{n,1})$ und wird in den beiden anschließenden Anwendungen von UNIMOD-ZT und UNIMOD-ST nicht mehr verändert; A hat dann die gewünschte Gestalt.

Wir geben nun den eben informal beschriebenen Algorithmus an, der mit unimodularen Transformationen $\mathrm{ggT}(a_{1,1}, \ldots, a_{1,n}, a_{2,1}, \ldots, a_{n,1})$ in das Element $a_{1,1}$ schreibt und alle anderen Einträge der ersten Zeile und der ersten Spalte auf Null setzt.

6.12. Algorithmus (UNIMOD-1,1)

Eingabe: $A \in \mathbb{Z}^{n \times n}$.

Ausgabe: $B, U, V \in \mathbb{Z}^{n \times n}$, U, V unimodular mit $B = U \cdot A \cdot V$ und B hat die Gestalt 6.1.

$U \leftarrow I;\ V \leftarrow I$		
Falls $a_{1,1} = 0$ (und $a_{j,1} \neq 0$) tausche die erste mit der j-ten Zeile in A und ändere V entsprechend		
WHILE A nicht die gewünschte Form hat		
	FOR $j \leftarrow 2$ TO n	
		$(r, u, v) \leftarrow \mathit{ERWEUKLALG}(a_{1,1}, a_{1,j})$
		$A \leftarrow A \cdot ZT(A, u, v, j); V \leftarrow V \cdot ZT(A, u, v, j)$
	FOR $j \leftarrow 2$ TO n	
		$(r, u, v) \leftarrow \mathit{ERWEUKLALG}(a_{1,1}, a_{j,1})$
		$A \leftarrow ST(A, u, v, j) \cdot A; U \leftarrow ST(A, u, v, j) \cdot U$
RETURN $\{ A, U, V \}$		

6.13. Satz *Sei $A \in \mathbb{Z}^{n \times n}$ regulär, $a_{1,1} \neq 0$. Dann benötigt Algorithmus 6.12, angewandt auf A,*

- *$O(n \cdot \log |a_{1,1}|)$ Anwendungen des euklidischen Algorithmus und*
- *$O(n^2 \cdot \log |a_{1,1}|)$ arithmetische Operationen.*

Beweis: Mit Satz 6.11 und obiger Betrachtung über die Anzahl der Anwendungen ergibt sich sofort die Behauptung. ■

Nach der Anwendung dieses Algorithmus überprüft man, ob $a_{1,1}$ alle restlichen Einträge der Matrix teilt. Wenn ja, dann wendet man den Algorithmus auf die rechte untere $(n-1) \times (n-1)$-Matrix an. Dann gilt: $a_{1,1} | a_{2,2}$. Wenn $a_{1,1}$ z.B. ein Element der j-ten Zeile nicht teilt, so addiere die j-te Zeile zu der ersten hinzu (das erzeugte Gitter bleibt unverändert, also ist diese Transformation unimodular) und wende wieder UNIMOD-1,1 an. Danach teilt $a_{1,1}$ alle Elemente der Zeile j. So sieht man: nach spätestens n Anwendungen von UNIMOD-1,1 hat die Matrix A die gewünschte Gestalt und $a_{1,1}$ teilt alle restlichen Matrixelemente.

Wenn wir nun dieses Verfahren sukzessive für jedes Diagonalelement wiederholen, erhalten wir eine Diagonalmatrix, die durch unimodulare Transformationen aus A hervorgegangen ist und dessen Diagonaleinträge sich sukzessive teilen. Wir berechnen also mit dem folgenden Algorithmus die Smith-Normalform von A, wobei wir einen leicht modifizierten Algorithmus UNIMOD-i,i verwenden, der die rechte untere $i \times i$-Teilmatrix wie oben gesehen bearbeitet, aber die durch natürliche Einbettung entstehenden $n \times n$-Matrizen zurückgibt. Es bezeichne $Z(i,j)$ die Einheitsmatirx mit einer zusätzlichen 1 an der Stele (i,j).

6.14. Algorithmus (Smith-Reduktion)

Eingabe: $A \in \mathbb{Z}^{n \times n}$.

Ausgabe: $S, U, V \in \mathbb{Z}^{n \times n}$ mit S in Smith-Normalform und U, V unimodular, so daß $S = U \cdot A \cdot V$.

$U \leftarrow I$; $V \leftarrow I$			
FOR $i \leftarrow 1$ TO $n-1$			
	$(A, X, Y) \leftarrow$ UNIMOD-i,i(A)		
	$U \leftarrow X \cdot U$; $V \leftarrow V \cdot Y$		
	FOR $j \leftarrow i+1$ TO n		
		IF es gibt ein Matrixelement $a_{j,k}$ mit $j, k > i$ und $a_{i,i} \nmid a_{j,k}$	
		THEN	$A \leftarrow Z(i,j) \cdot A; U \leftarrow Z(i,j) \cdot U$, wobei $Z(i,j)$ die Einheitsmatrix mit einer zusätzlichen 1 an der Stelle (i,j) ist
			$(A, X, Y) \leftarrow$ UNIMOD-i,i(A)
			$U \leftarrow X \cdot U$; $V \leftarrow V \cdot Y$
RETURN $\{ S, U, V \}$			

6.15. Satz *Sei $A \in \mathbb{Z}^{n \times n}$ regulär. Dann benötigt Algorithmus 6.14*

- *$O(n^2 \log d)$ Anwendungen des erweiterten euklidischen Algorithmus und*
- *$O(n^3 \log d)$ arithmetische Operationen,*

um die Smith-Normalform von A zu berechnen, wobei d eine obere Schranke für die Beträge der in der Diagonalen während der Berechnung auftretenden Zahlen ist.

Beweis: Der Übersichtlichkeit halber haben wir Algorithmus 6.12 prozedural behandelt; für die Komplexitätsanalyse betrachten wir jedoch die tatsächliche Anzahl der Anwendungen von UNIMOD-ZT und UNIMOD-ST, denn der — so scheint es — zusätzliche Aufwand, um zu erreichen, daß das gerade betrachtete Diagonalelement alle Elemente der entsprechenden Untermatrix teilen soll, wird schon in der Abschätzung für Algorithmus 6.12 abgefangen: Im schlechtesten Fall wird UNIMOD-ZT bzw. UNIMOD-ST $\log a_{i,i}$ mal für jedes i angewandt; dann ist $a_{i,i} = 1$ und teilt somit sowieso alle restlichen Matrixeinträge. Wenn d eine obere Schranke für die auftretenden $a_{i,i}$ ist, so folgt unmittelbar die Behauptung. ∎

Die Größe der auftretenden Zahlen ist nicht nur theoretisch schwer vorherzusehen, sondern stellt auch praktisch ein großes Problem dar. Allerdings kann man mit dem prinzipiell gleichen Verfahren die Smith-Normalform einer regulären Matrix berechnen und dabei eine obere Beschränkung der Größe der auftretenden Zahlen (die Determinante der Matrix) benutzen. Der Schlüssel zu diesem abgewandelten Verfahren liefert folgender

6.16. Satz *Es seien $A \in \mathbb{Z}^{n \times n}$ regulär, d die Determinante von A, $s_1, \ldots, s_n$ die Diagonaleinträge der Smith-Normalform von A,*

$$d_1 = d \quad \textit{und} \quad d_{i+1} = \frac{d_i}{s_i} \quad (i = 1, \ldots, n-1)$$

und $t_1, \ldots, t_n$ die Diagonaleinträge der Matrix, die durch die unimodularen Transformationen, die bei der Smith-Reduktion gemacht werden, gefolgt von Reduktion modulo d, aus A entstanden ist. Dann gilt:

$$\underset{i \in [\![1,n]\!]}{\forall} \quad s_i = \mathrm{ggT}(d_i, t_i).$$

Beweis: Dieses Reduktionsprinzip wurde von Domich, Kannan und Trotter für die Berechnung der *Hermite-Normalform* einer Matrix vorgestellt und auf die Smith-Normalform einer Matrix von Hafner und McCurley übertragen. Siehe zum Beweis [Dü92, S.27ff], [HaMcCu] und [DoKaTr87]. ∎

Damit ergibt sich der folgende Algorithmus:

6.17. Algorithmus (Modulare Smith-Reduktion)

Eingabe: $A \in \mathbb{Z}^{n \times n}$ regulär, $d = \det A$.

Ausgabe: Die Diagonaleinträge $s_1, \ldots, s_n$ der Smith-Normalform von A.

```
FOR i ← 1 TO n − 1
    Berechne A mit UNIMOD-i,i(A); berechne jede arithmetische Operation modulo d.
    FOR j ← i + 1 TO n
        IF es gibt ein Matrixelement a_{j,k} mit j, k > i und a_{i,i} ∤ a_{j,k}
        THEN A ← Z(i, j) · A
             Berechne A durch UNIMOD-i,i(A); berechne jede arithmetische Operation modulo d
d_1 ← d
FOR i ← 1 TO n
    s_i ← ggT(d_i, a_{i,i})
    d_{i+1} ← d_i / s_i
RETURN { s_1, ..., s_n }
```

Die Anzahl der Anwendungen des euklidischen Algorithmus mit Zahlen $\leq d$ im zweiten Teil des Algorithmus beträgt n; ansonsten ist der Aufwand dieses Algorithmus gegeben wie bei Algorithmus 6.14. Wir haben also:

6.18. Satz *Sei $A \in \mathbb{Z}^{n \times n}$ regulär, $d = \det A$. Dann benötigt Algorithmus 6.17*

- *$O(n^2 \log d)$ Anwendungen des erweiterten euklidischen Algorithmus und*
- *$O(n^3 \log d)$ arithmetische Operationen*

mit Zahlen $\leq d$ um die Smith-Normalform von A zu berechnen.

Es folgt

6.19. Satz *Sei $A \in \mathbb{Z}^{n \times n}$ regulär, $d = \det A$. Dann kann die Smith-Normalform von A in*

- *$O(n^3 \log^3 d)$ Bitoperationen*

berechnet werden.

Beweis: Der erweiterte euklidische Algorithmus angewandt auf Zahlen $\leq d$ kann in $O(\log^2 d)$ Bitoperationen realisiert werden; jede arithmetische Operation zweier Zahlen $\leq d$ kann ebenfalls in $O(\log^2 d)$ Bitoperationen realisiert werden. Daraus folgt die Behautpung. ■

Nachteilig ist allerdings, daß man mit der hier vorgestellten modularen Smith-Reduktion keine Basis berechnen kann. Die Frage, ob dieses Verfahren so erweiterbar ist, daß die Transformationmatrizen U und V mit berechnet werden können, sprengt allerdings den Rahmen unserer Arbeit.

6.4 Die Berechnung der Struktur

Wir können nun einen Algorithmus angeben, der deterministisch die Struktur einer Gruppe, die durch $\gamma_1, \ldots, \gamma_n$ gegeben ist, berechnen. Wir berechnen sukzessive minimale Relationen für $\gamma_1, \ldots, \gamma_i$, schreiben diese in eine Matrix und berechnen die Smith-Normalform der Matrix.

6.20. Algorithmus (Struktur)

Eingabe: $\gamma_1, \ldots, \gamma_n \in G$ $(G \in \mathcal{G})$.

Ausgabe: $(s_1, \ldots, s_m)$, die Struktur von $< \gamma_1, \ldots, \gamma_n >$.

Berechne iterativ minimale Relationen für $\gamma_1, \ldots, \gamma_i$, $(i = 1, \ldots, n)$ mit *einer* Anwendung von Algorithmus 4.10 und schreibe diese spaltenweise in eine Matrix A, so daß $(a_{1,i}, \ldots, a_{i,i})$ minimale Relation für $(\gamma_1, \ldots, \gamma_i)$ und $a_{j,i} = 0$ falls $j > i$.	(1)
$d \leftarrow \prod_{i=1}^{n} a_{i,i}$	(2)
Berechne die Smith-Normalform der Matrix A mit Algorithmus 6.17 und den Eingaben (A, d).	(3)
RETURN { $(s_i, \ldots, s_n)$ }, wobei $i = \min\{j : 1 \leq j \leq n \text{ mit } s_j \neq 1\}$	(4)

6.21. Satz *Sei $G \in \mathcal{G}$, $\gamma_1, \ldots, \gamma_n \in G$, sei $h = | < \gamma_1, \ldots, \gamma_n > |$. Dann kann die Struktur der Gruppe $< \gamma_1, \ldots, \gamma_n >$ in*

- *$O(n^2 \cdot \sqrt{h})$ Gruppenadditionen und*
- *$O(n \cdot \sqrt{h} \cdot \log h \cdot \log n_G + n^3 \log^3 h)$ Bitoperationen*

berechnet werden.

Beweis: Durch Zusammensetzen der Ergebnisse von Satz 4.15 und Satz 6.19 ergibt sich die Behauptung; d ist hier die Ordnung der betrachteten Gruppe (s. Lemma 4.8). ■

Im allgemeinen ist man aber nicht an der Struktur einer durch ein Tupel von erzeugenden Elementen gegebenen Gruppe interessiert, sondern an der Struktur der Gruppe G, in der man rechnet. Wir gehen nun davon aus, daß wir eine obere Schranke L der Gruppenordnung der Form

$$\frac{1}{2}L < |G| \leq L$$

kennen. Die Berechnung der Gruppenstruktur geht nun folgendermaßen: wir erzeugen solange zufällig Elemente $\gamma_1, \ldots, \gamma_n$, bis die Ordnung der von diesen Elementen erzeugten Gruppe $> \frac{1}{2}L$ ist. Denn es gilt:

6.22. Lemma *Sei G eine endliche abelsche Gruppe, $L \in \mathbb{N}$ mit*

$$\frac{1}{2}L < |G| \leq L$$

und $\gamma_1, \ldots, \gamma_n \in G$. Dann gilt:

$$| < \gamma_1, \ldots, \gamma_n > | > \frac{1}{2}L \quad \Longrightarrow \quad < \gamma_1, \ldots, \gamma_n >= G.$$

Beweis: Bezeichne $G_n =< \gamma_1, \ldots, \gamma_n >$. Ist $G_n \neq G$, so gilt:

$$L \geq |G| = |G_n| \cdot \left| {}^{G}\!/_{G_n} \right| \geq |G_n| \cdot 2,$$

woraus $|G_n| \leq \frac{1}{2}L$ folgt. Daraus erhält man die Behauptung. ∎

Dann können wir wie oben die Smith-Reduktion auf die Relationenmatrix anwenden und sind fertig. Die Berechnung der Ordnung wird mit der Berechung der minimalen Relationen gekoppelt und wir haben folgenden Algorithmus:

6.23. Algorithmus (Struktur1)

Eingabe: $L \in \mathbb{N}$ mit $\frac{1}{2}L < |G| \leq L$, $(G \in \mathcal{G})$.

Ausgabe: $(s_1, \ldots, s_m)$, die Struktur von G.

$l \leftarrow 1$; $k \leftarrow 0$	(1)
WHILE $l \leq \frac{L}{2}$	(2)
$x \leftarrow 1$	(3)
WHILE $x = 1$	(4)
$\gamma \in_R G$	(5)
Berechne minimale Relationen $(x_1, \ldots, x_k, x)$ mit Vorinformation $(x_{1,1}, \ldots, x_{k,k})$ für $(\gamma_1, \ldots, \gamma_k, \gamma)$ mit Algorithmus 4.10	(6)
$\gamma_{k+1} \leftarrow \gamma$; $l \leftarrow l \cdot x$; $(x_{k+1,1}, \ldots, x_{k+1,k+1}) \leftarrow (x_1, \ldots, x_k, x)$; $k \leftarrow k+1$	(7)
$A \leftarrow (x_{i,j})_{i,j=1,\ldots,k}$, $a_{i,j} = 0$ für $j < i$	(8)
Starte Algorithmus 6.17 mit Eingaben (A, l)	(9)
RETURN $\{ (s_i, \ldots, s_n) \}$, wobei $i = \min\{j : 1 \leq j \leq n \text{ mit } s_j \neq 1\}$	(10)

Bei der Erzeugung eines zufälligen Elementes berücksichtigen wir natürlich nur solche, die die betrachtete Gruppe vergrößern. Also lassen wir solche γ, die eine minimale Relation mit einer 1 zur Folge haben, wieder fallen, denn dann liegt γ ja gerade in $\gamma_1, \ldots, \gamma_k$. Dies kann dazu führen, daß der Algorithmus nicht terminiert. Allerdings ist, falls er terminiert, das Ergebnis das geforderte. Ein solcher Algorithmus wird **Las-Vegas-Algorithmus** genannt. Berechnen wir nun die erwartete Anzahl an Versuchen, bis wir ein *nicht redundantes* γ bekommen.

6.24. Lemma *Sei $G \in \mathcal{G}$, G_k eine echte Untergruppe von G und $g \in \mathbb{N}$, so daß für alle $\gamma \in G$*

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Dann ist die erwartete Anzahl von Versuchen, ein Element $\gamma \in G \setminus G_k$ zu wählen, $\leq 2 \cdot g$.

Beweis: Die Wahrscheinlichkeit dafür, ein Element $\gamma \in G \setminus G_k$ zu wählen, beträgt

$$p(\gamma \in G \setminus G_k) = \sum_{\delta \in G \setminus G_k} p(\delta \in_R G) \geq \sum_{\delta \in G \setminus G_k} \frac{1}{|G|} \cdot \frac{1}{g} = \frac{1}{g} \cdot \frac{|G| - |G_k|}{|G|}.$$

Nun ist G_k eine echte Untergruppe von G, also ist $|G| - |G_k| \geq \frac{|G|}{2}$. Somit ist

$$p(\gamma \in G \setminus G_k) \leq \frac{1}{2} \cdot \frac{1}{g}.$$

Daraus folgt die Behauptung. ∎

Wir können weiter feststellen:

6.25. Lemma *Sei G eine endliche abelsche Gruppe, die Wahl eines Elementes von G unterliege der Gleichverteilung. Dann ist die erwartete Anzahl von "Ziehungen" von Gruppenelementen $\gamma_1, \ldots, \gamma_k$, bis $G = <\gamma_1, \ldots, \gamma_k>$*

$$O(\log |G| \cdot g).$$

Beweis: Im schlechtesten Fall vergrößert sich die betrachtete Gruppe bei jeder "Ziehung" eines nichtredundanten Gruppenelementes nur um den Faktor 2; man benötigt also $O(\log |G|)$ Ziehungen von nichtredundanten Elementen. Mit obigem Lemma ergibt sich die Behauptung. ∎

Hiermit können wir nun eine obere Schranke für die erwartete Laufzeit von Algorithmus 6.23 angeben:

6.26. Satz *Sei $G \in \mathcal{G}$, $L \in \mathbb{N}$ mit $\frac{L}{2} < |G| \leq L$ und $g \in \mathbb{N}$, so daß für alle $\gamma \in G$*

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Dann kann man die Berechnung der Struktur von G nach

- *$O(\log |G| \cdot g)$ pseudozufälligen Erzeugungen von Elementen aus G,*
- *$O(\sqrt{|G|} \cdot \log^2 |G| \cdot g^2)$ Gruppenadditionen und*
- *$O(\sqrt{|G|} \cdot \log^2 |G| \cdot \log n_G \cdot g + \log^6 |G| \cdot g^3)$ Bitoperationen*

erwarten.

Beweis: Der Unterschied zwischen Algorithmus 6.23 und Algorithmus 6.20 besteht in der Ziehung von Elementen aus der betrachteten Gruppe und in der maximalen Anzahl der Elemente, für die man minimale Relationen mit Vorinformation berechnet. Diese Anzahl (in Satz 6.21 mit n bezeichnet) ist hier gleich der Anzahl an Ziehungen von Elementen aus der Gruppe und beträgt

$$O(\log |G| \cdot g).$$

Mit Satz 6.21 ergibt sich die Behauptung. ∎

6.5 Strukturberechnung ohne obere Schranke für die Gruppenordnung

In den meisten Fällen kennen wir entweder gar keine obere Schranke L oder zumindest keine, die der obigen Bedingung

$$\frac{L}{2} < |G| \leq L$$

genügt. In diesem Fall können wir folgende Idee verwenden: wenn wir viele Male hintereinander ein redundantes Element der Gruppe "ziehen", so gehen wir davon aus, daß wir schon die gesamte Gruppe erzeugen können. Es gilt:

6.27. Lemma *Sei $G \in \mathcal{G}$, $G_n \leq G$ und $g \in \mathbb{N}$, so daß für alle $\gamma \in G$*

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Wenn k-mal hintereinander das Ergebnis γ der pseudozufälligen Erzeugung eines Gruppenelementes schon in der von den zuvor gezogenen Elementen aufgespannten Untergruppe G_n liegt, so gilt:

$$p(G_n = G) \geq 1 - \left(\frac{2g-1}{2g}\right)^k.$$

Beweis: Bezeichne γ_i das i-te gezogene Element. Dann ist die Wahrscheinlichkeit, k Mal hintereinander ein redundantes Element zu ziehen

$$\begin{aligned}
\prod_{i=1}^{k} p(\gamma_i \in G_n) &= \prod_{i=1}^{k} 1 - p(\gamma \in G \setminus G_n) \\
&\leq \prod_{i=1}^{k} \left(1 - \frac{1}{2} \cdot \frac{1}{g}\right) \\
&= \left(\frac{2g-1}{2g}\right)^k.
\end{aligned}$$

Daraus ergibt sich die Behauptung. ∎

Wenn wir nun einen Algorithmus angeben wollen, der die Struktur einer Gruppe berechnet, so können wir eine Genauigkeitsschranke $c > 0$ angeben, so daß die Ausgabe mit Wahrscheinlichkeit $1 - c$ richtig ist. Allerdings benötigt man dann $g \in \mathbb{N}$, so daß für alle $\gamma \in G$

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

6.28. Lemma *Sei $G \in \mathcal{G}$ und $g \in \mathbb{N}$, so daß für alle $\gamma \in G$*

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Wählt man in Lemma 6.27

$$k > \frac{\log c}{\log\left(\frac{2g-1}{2g}\right)},$$

so gilt

$$p(G_n = G) > 1 - c.$$

Beweis: Bricht der Algorithmus nach k erfolglosen Schritten ab, so ist die Wahrscheinlichkeit, ein richtiges Ergebnis zu haben:

$$1 - \left(\frac{2g-1}{2g}\right)^k.$$

Soll nun diese Wahrscheinlichkeit größer als $1 - c$ sein, so muß die obere Schranke für k gelten. ■

Bei dieser Lösung, im Gegensatz zu der Variante mit der vorgegebenen oberen Schranke ist ein Terminieren hier gewährleistet. Man kann dafür in diesem Fall eine Wahrscheinlichkeit vorgeben, mit der das ausgegebene Ergebnis falsch sein darf und so die Rundenanzahl vermindern; ein solcher Algorithmus heißt **Monte-Carlo-Algorithmus**.

6.29. Algorithmus (Struktur2)

Eingabe: $c > 0$ Genauigkeitsschranke, $g \in \mathbb{N}$, so daß für alle $\gamma \in G$

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Ausgabe: $s_1, \ldots, s_m$ mit $p\left((s_1, \ldots, s_m) \text{ Struktur von } G\right) > 1 - c$.

$k \leftarrow 0$; $s \leftarrow 0$	(1)
WHILE $s < \lceil -\log c \rceil$	(2)
— $x \leftarrow 1$	(3)
— WHILE $s < \frac{\log c}{\log \frac{2g-1}{2g}}$ AND $x = 1$	(4)
— — $s \leftarrow s + 1$	(5)
— — $\gamma \in_R G$	(6)
— — $(x_1, \ldots, x_k, x) \leftarrow MINREL(\gamma_1, \ldots, \gamma_k, \gamma)$	(7)
— $\gamma_{k+1} \leftarrow \gamma$; $l \leftarrow l \cdot x$; $(x_{k+1,1}, \ldots, x_{k+1,k+1}) \leftarrow (x_1, \ldots, x_k, x)$; $k \leftarrow k + 1$	(8)
— IF $x \neq 1$ THEN $s \leftarrow 0$	(9)
$A \leftarrow (x_{i,j})_{i,j=1,\ldots,k}$, $a_{i,j} = 0$ für $j < i$	(10)
Starte Algorithmus 6.17 mit Eingaben (A, l)	(11)
RETURN $\{ (s_i, \ldots, s_n) \}$, wobei $i = \min\{j : 1 \leq j \leq n \text{ mit } s_j \neq 1\}$	(12)

Es folgt

6.30. Satz *Sei $G \in \mathcal{G}$, $g \in \mathbb{N}$, so daß für alle $\gamma \in G$*

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Dann kann man die Struktur von G mit

- $O(\frac{\log c}{\log \frac{2g-1}{2g}})$ *pseudozufälligen Erzeugungen von Elementen aus G,*
- $O\left(\sqrt{|G|} \cdot \log^2 |G| \cdot \left(\frac{\log c}{\log \frac{2g-1}{2g}}\right)^2\right)$ *Gruppenadditionen und*

- $O\left(\left(\sqrt{|G|} \cdot \log^2 |G| \cdot \log n_G \cdot \frac{\log c}{\log \frac{2g-1}{2g}} + \left(\frac{\log c}{\log \frac{2g-1}{2g}}\right)^3 \log^3 |G|\right)\right.$ *Bitoperationen*

mit Wahrscheinlichkeit $1-c$ *berechnen.*

Beweis: Der einzige Unterschied zwischen Algorithmus 6.29 und Algorithmus 6.23 besteht in der nun festen Anzahl der maximalen Schleifendurchläufe. Mit Satz 6.21 und der in Lemma 6.28 bestimmten Anzahl der maximalen erfolglosen Ziehungen der inneren Schleife pro erfolgreiche Ziehung sowie der Maximalanzahl der erfolgreichen Ziehungen ($\log |G|$, denn mit jeder erfolgreichen Ziehung wird die Ordnung der aufgespannten Gruppe mindestens verdoppelt) ergibt sich die Behauptung. ■

6.6 Die Berechnung der Gruppenordnung

Ist man nur an der Gruppenordnung interessiert, so genügt es, zufällig Elemente zu erzeugen, minimale Relationen zu berechnen und entweder — wie oben — zu warten, bis das Produkt der Diagonaleinträge der konstruierten Matrix größer als eine bestimmte Schranke ist oder nach einer durch eine vorgegebene Genauigkeit gegebene Anzahl von Schritten abzubrechen und zu behaupten, das Produkt der Diagonaleinträge sei die Gruppenordnung. Die Algorithmen entsprechen den schon besprochenen Algorithmen Struktur1 und Struktur2; mit der Ausnahme, daß keine Smith-Reduktion berechnet wird, sondern x, die Ordnung der von den zufällig gewählten $\gamma_1, \ldots, \gamma_k$ aufgespannten Gruppe, ausgegeben wird.

Für die Bestimmung der Ordnung einer Gruppe haben wir die folgenden Komplexitätsaussagen:

6.31. Satz *Sei* $G \in \mathcal{G}$, $L \in \mathbb{N}$ *mit* $\frac{L}{2} < |G| \leq L$ *und* $g \in \mathbb{N}$, *so daß für alle* $\gamma \in G$

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Dann kann man die Berechnung der Gruppenordnung von G *nach*

- $O(\log |G| \cdot g)$ *pseudozufälligen Erzeugungen von Elementen aus* G,
- $O(\sqrt{|G|} \cdot \log^2 |G| \cdot g)$ *Gruppenadditionen und*
- $O(\sqrt{|G|} \cdot \log^2 |G| \cdot \log n_G \cdot g)$ *Bitoperationen*

erwarten.

und

6.32. Satz *Sei* $G \in \mathcal{G}$ *und* $g \in \mathbb{N}$, *so daß für alle* $\gamma \in G$

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Dann kann man die Gruppenordnung von G *in*

- $O(\frac{\log c}{\frac{2g-1}{2g}})$ *pseudozufälligen Erzeugungen von Elementen aus* G,
- $O\left(\sqrt{|G|} \cdot \log^2 |G| \cdot \left(\frac{\log c}{\frac{2g-1}{2g}}\right)^2\right)$ *Gruppenadditionen und*

- $O\left(\sqrt{|G|} \cdot \log^2|G| \cdot \log n_G \cdot \frac{\log c}{\frac{2g-1}{2g}}\right)$ *Bitoperationen*

mit Wahrscheinlichkeit $1-c$ *berechnen.*

Die Aussagen ergeben sich aus den Sätzen zu den Algorithmen Struktur1 und Struktur2, indem wir den Aufwand für die Smith-Reduktion herausnehmen.

Falls wir ein Intervall $[\![l, L]\!]$ für die Gruppenordnung kennen, so können wir die Anzahl der Runden durch einen kleinen Trick deutlich minimieren:

6.33. Lemma *Seien* $l, L \in \mathbb{N}$, $l < L$. $x \in \mathbb{N}$. *Dann gibt es genau dann höchstens ein Vielfaches von* x *in* $[\![l, L]\!]$, *wenn* $x > L - l$.

Beweis: Wenn es zwei aufeinanderfolgende Vielfache $k \cdot x$ und $(k+1) \cdot x$ in $[\![l, L]\!]$ gibt, so gilt $(k+1) \cdot x - k \cdot x \leq L - l$. Die andere Richtung ist trivial. ■

Somit prüfen wir in jeder Runde, ob $x > L - l$ ist und bestimmen am Schluß $z = x \cdot \left\lfloor \frac{L}{x} \right\rfloor$, die Gruppenordnung.

6.34. Algorithmus (Gruppenordnung mit Intervall)

Eingabe: $L, l \in \mathbb{N}$ mit $l \leq |G| \leq L$.

Ausgabe: h, die Gruppenordnung.

$h \leftarrow 1$; $k \leftarrow 0$	(1)
WHILE $h \leq L - l$	(2)
$x \leftarrow 1$	(3)
WHILE $x = 1$	(4)
$\gamma \in_R G$	(5)
Berechne minimale Relationen $(x_1, \ldots, x_k, x)$ mit Vorinformation $(x_{1,1}, \ldots, x_{k,k})$ für $(\gamma_1, \ldots, \gamma_k, \gamma)$ mit Algorithmus 4.10	(6)
$\gamma_{k+1} \leftarrow \gamma$; $l \leftarrow l \cdot x$; $(x_{k+1,1}, \ldots, x_{k+1,k+1}) \leftarrow (x_1, \ldots, x_k, x)$; $k \leftarrow k+1$	(7)
RETURN $\left\{h \cdot \left\lfloor \frac{L}{h} \right\rfloor\right\}$	(8)

Wir erhalten:

6.35. Satz *Sei* $G \in \mathcal{G}$, $L, l \in \mathbb{N}$ *mit* $l \leq |G| \leq L$ *und* $g \in \mathbb{N}$, *so daß für alle* $\gamma \in G$

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Dann kann man die Berechnung der Gruppenordnung von G *nach*

- $O(\log(L-l) \cdot g)$ *pseudozufälligen Erzeugungen von Elementen aus* G,
- $O(\sqrt{L-l} \cdot \log^2(L-l) \cdot g)$ *Gruppenadditionen und*
- $O(\sqrt{L-l} \cdot \log^2(L-l) \cdot \log n_G \cdot g + \log L \cdot \log \frac{L}{L-l})$ *Bitoperationen*

erwarten.

Beweis: Der einzige Unterschied zu Algorithmus 6.23 besteht – abgesehen von der nicht vorhandenen Smith-Reduktion und der dafür eingesetzten Berechnung der Gruppenordnung – in der Größe der maximal aufgespannten Gruppe, nämlich $O(L-l)$. Demzufolge sind die Komplexitäten, die von den Gruppenoperationen herkommen, Funktion von $L-l$.
Der zusätzliche Aufwand für die Berechnung der Gruppenordnung in Schritt (8) beträgt

$$O(\log x \cdot \log \frac{L}{x}) = O(\log L \cdot \log \frac{L}{L-l}),$$

wobei x die Ordnung der bis dahin aufgespannten Untergruppe ist, sowohl für die ganzzahlige Division als auch für die Multiplikation. Da kommt der zusätzliche Aufwand bei den Bitoperationen her. ∎

Kapitel 7

Berechnung von n-ten Wurzeln in einer Gruppe

In diesem Kapitel werden wir sehen, wie man unter zwei verschiedenen Voraussetzungen (unter der Kenntnis der Struktur und einer Basis der Gruppe sowie der Kenntnis der Gruppenordnung alleine) **n-te Wurzeln** in einer Gruppe berechnen kann. Hierzu betrachten wir unter anderem spezieller die Situation in p-Gruppen.

7.1 Problemstellung

7.1. Definition *Sei G eine endliche abelsche Gruppe, $\alpha \in G$ und $n \in \mathbb{N}$. Ein Element $\gamma \in G$ mit*

$$n\gamma = \alpha$$

heißt **n-te Wurzel** *von α.*

Bei einer gegebenen Gruppe $G \in \mathcal{G}$ und gegebenem $\alpha \in G$ und $n \in \mathbb{N}$ interessieren wir uns

1. für die *Existenz* einer n-ten Wurzel von α,
2. für die Berechnung *einer* n-ten Wurzel von α und
3. für die Berechnung *aller* n-ten Wurzeln von α;

unter der Voraussetzung, die Struktur und eine Basis der Gruppe seien bekannt. In diesem Fall reduziert sich das Problem auf die Berechnung von n-ten Wurzeln in zyklischen Gruppen:

7.2. Lemma *Sei G eine endliche abelsche Gruppe, $(s_1, \ldots, s_m)$ die Struktur von G, $(\delta_1, \ldots, \delta_m)$ eine Basis von G, $\alpha = \sum_{i=1}^{m} a_i\delta_i \in G$ und $n \in \mathbb{N}$. Dann gilt:*

1. *Es existiert genau dann ein $\gamma \in G$ mit $n\gamma = \alpha$, wenn für alle $i = 1, \ldots, m$ $g_i \in [\![0, s_i - 1]\!]$ existiert mit*
$$n \cdot g_i \equiv a_i \mod s_i.$$
2. *Unter dieser Voraussetzung gilt:* $\gamma = \sum_{i=1}^{m} g_i\delta_i\,.$

Beweis: Die Behauptung folgt sofort aus Satz 1.7. ■

Zur Berechnung der a_i bei bekanntem α benötigt man die Struktur und eine Basis der Gruppe, denn die Berechnung einer minimalen Relation für $(\delta_1, \ldots, \delta_m, \alpha)$ liefert gerade $(a_1, \ldots, a_m, 1)$. Die Anzahl der betrachteten n schränkt sich ebenfalls ein, denn es gilt:

7.3. Lemma *Sei G eine zyklische Gruppe der Ordnung s, $n \in \mathbb{N}$. Dann gilt:*

$$\forall_{\alpha \in G} \; n\alpha = (n \bmod s)\,\alpha.$$

Beweis: Sei $n = q \cdot s + r$. Dann ist

$$n\alpha = (q \cdot s + r)\,\alpha = q\,(s\alpha) + r\alpha = r\alpha.$$

Daraus folgt die Behauptung. ■

Um eine beliebige n-te Wurzel in einer zyklischen Gruppe der Ordnung s zu bestimmen, genügt es also, eine $(n \bmod s)$-te Wurzel zu berechnen.

7.2 n-te Wurzeln in zyklischen Gruppen

In diesem Kapitel werden wir folgende Frage beantworten: gegeben eine zyklische Gruppe G mit Ordnung s, ein Element $\alpha \in G$ und eine Zahl $n \in [\![0, s-1]\!]$, wie berechnet man eine n-te Wurzel von α? Da wir davon ausgehen, ein Basiselement zu kennen, reduziert sich nach Lemma 7.2 die Betrachtung auf die Gruppe $\mathbb{Z}/s\mathbb{Z}$.

7.4. Satz *Seien $n, a \in [\![0, s-1]\!]$. Die Gleichung*

$$n \cdot x \equiv a \bmod s$$

ist genau dann lösbar, wenn

$$\mathrm{ggT}(n, s) \mid a$$

und hat dann $\mathrm{ggT}(n, s)$ viele Lösungen x_i, für die gilt:

$$x_i \equiv x_j \bmod \frac{s}{\mathrm{ggT}(n, s)} \quad \text{für alle } i, j = 0, \ldots, \mathrm{ggT}(n, s) - 1.$$

Beweis: Bezeichne $d = \mathrm{ggT}(n, s)$ und $\overline{x}$ die Äquivalenzklasse von x in $\mathbb{Z}/s\mathbb{Z}$. Betrachte folgende Abbildung:

$$\begin{aligned} \tau : \mathbb{Z}/s\mathbb{Z} &\longrightarrow \mathbb{Z}/s\mathbb{Z} \\ \overline{x} &\longmapsto n\overline{x}. \end{aligned}$$

Die Gleichung

$$n \cdot x \equiv a \bmod s$$

ist genau dann lösbar, wenn $\overline{a} \in \mathrm{Im}(\tau)$; in diesem Fall gibt es $|\,\mathrm{Kern}\,\tau|$ viele verschiedene Lösungen. Es gilt:

$$\begin{aligned}
\mathrm{Kern}\,\tau &= \left\{\overline{x} : n\overline{x} =_{\mathbb{Z}/s\mathbb{Z}}\right\} \\
&= \{\overline{x} : \mathrm{ord}\,\overline{x} \,|\, n\} \\
&= \{\overline{x} : \mathrm{ord}\,\overline{x} \,|\, d\} \quad \text{da } \mathrm{ord}\,\overline{x} \,|\, s \\
&= \left\{\overline{x} : \overline{x} = i \cdot \overline{\frac{s}{d}}, i = 0, \ldots, d-1\right\};
\end{aligned}$$

Falls die Gleichung lösbar ist, gibt es also d verschiedene Lösungen, für die gilt:

$$x_i \equiv x_j \bmod \frac{s}{\mathrm{ggT}(n,s)} \qquad \text{für alle } i, j = 0, \ldots, \mathrm{ggT}(n,s) - 1.$$

Sei $n' = n/d$. Betrachte nun

$$\begin{aligned}
\mathrm{Im}\tau &= \left\{n\overline{x} : \overline{x} \in {}^{\mathbb{Z}}\!/_{s\mathbb{Z}}\right\} \\
&= \left\{n'd\,\overline{x} : \frac{s}{d}d\,\overline{x} =_{ZpZs}\right\} \\
&= \left\{n'\overline{y} : \frac{s}{d}\overline{y} =_{ZpZs}\right\} \\
&= \left\{\overline{y} : \frac{s}{d}\overline{y} =_{ZpZs}\right\},
\end{aligned}$$

denn die obige Betrachtung für die Abbildung

$$\begin{aligned}
\sigma : {}^{\mathbb{Z}}\!/_{s\mathbb{Z}} &\longrightarrow {}^{\mathbb{Z}}\!/_{s\mathbb{Z}} \\
\overline{x} &\longmapsto n'\overline{x}
\end{aligned}$$

liefert $\mathrm{Kern}\,\sigma = \{_{\mathbb{Z}/s\mathbb{Z}}\}$ (da $\mathrm{ggT}(n', s) = 1$), und somit ist σ bijektiv. Also gilt:

$$\begin{aligned}
\mathrm{Im}\tau &= \left\{\overline{y} : \frac{s}{d}\overline{y} =_{\mathbb{Z}/s\mathbb{Z}}\right\} \\
&= \left\{\overline{y} : \overline{y} = i \cdot \overline{d}, i = 0, \ldots, \frac{s}{d} - 1\right\};
\end{aligned}$$

womit wir auch den ersten Teil unserer Behauptung bewiesen hätten. ■

Wir haben bis jetzt nur eine existentielle Aussage über die Lösung unseres Problems angegeben; nun interessiert uns die Konstruktion einer Lösung. Hierzu dient folgende Idee:

7.5. Lemma *Sei $s \in \mathbb{N}$, $n, a \in [\![0, s-1]\!]$; bezeichne $d = \mathrm{ggT}(n, s)$ und $n' = \frac{n}{d}$. Es gilt $\mathrm{ggT}(n', s) = 1$ und somit gibt es $n_0 \in [\![0, s-1]\!]$ mit*

$$n_0 \cdot n' \equiv 1 \bmod s.$$

Wenn $d \,|\, a$, dann ist $x_0 = n_0 \cdot \frac{a}{d}$ eine Lösung der Gleichung

$$n \cdot x \equiv a \bmod s.$$

Beweis:

$$n \cdot \left(n_0 \cdot \frac{a}{d}\right) = d \cdot n' \cdot n_0 \cdot \frac{a}{d} \equiv d \cdot \frac{a}{d} = a \quad (\bmod s).$$ ■

Mit dieser Idee können wir nun einen Algorithmus angeben, der n-te Wurzeln in zyklischen Gruppen bestimmt, falls sie existieren:

7.6. Algorithmus (Wurzeln in zyklischen Gruppen)

Eingabe: $\alpha, \delta \in G$ $(G \in \mathcal{G})$, $s, \in \mathbb{N}$ mit $\alpha \in <\delta>$ und ord $\delta = s$, $n \in [\![0, s-1]\!]$.

Ausgabe: *Es gibt keine n-te Wurzel* oder $\gamma \in <\delta>$ mit $n\gamma = \alpha$.

Berechne eine minimale Relation $(a, 1)$ für (δ, α)		(1)
$d \leftarrow EUKLALG(n, s)$		(2)
$(a', r) \leftarrow (a \div d, a \bmod d)$		(3)
IF $r \neq 0$		(4)
THEN	RETURN { Es gibt keine n-te Wurzel von a }	(5)
ELSE	$(1, n_0, *) \leftarrow ERWEUKLALG(\frac{n}{d}, s)$	(6)
	return $\{ n_0 \cdot a'\delta \}$	(7)

Schritt (1) benötigt man, um das Problem in die Gruppe $\mathbb{Z}/s\mathbb{Z}$ zu transportieren. Entsprechend muß man am Schluß die Lösung wieder in die betrachtete Gruppe bringen ((7)). Ansonsten geht die Korrektheit aus Satz 7.4 und Lemma 7.5 hervor. Es gilt:

7.7. Satz *Sei $G \in \mathcal{G}$ eine zyklische Gruppe mit Ordnung s. Dann kann man unter Kenntnis eines erzeugenden Elementes und der Ordnung in*

- $O(\sqrt{s})$ *Gruppenadditionen und*
- $O(\sqrt{s} \cdot \log s \cdot \log n_G)$ *Bitoperationen*

entscheiden, ob ein Element $\alpha \in G$ eine n-te Wurzel in G hat und sie gegebenenfalls berechnen.

Beweis: Der Aufwand von Schritt (2) beträgt sicherlich $O(\log^2 s)$ Bitoperationen (da $n < s$). Auch Schritt (3) hat diese Komplexität. Wie wir in Kapitel 5 gesehen haben, ist auch der Aufwand von Schritt (6)) dadurch beschränkt. Schritt (7) kann mit schneller Multiplikation realisiert werden und benötigt $O(\log s)$ Gruppenadditionen und $O(\log^2 s)$ Bitoperationen. Alle diese Berechnungen werden von der Komplexität der Berechnung des Diskreten Logarithmus a majorisiert. So ergibt sich die Behauptung ■

Wie man nun *alle* n-ten Wurzeln berechnen kann, geht aus Satz 7.4 hervor:

7.8. Algorithmus (alle n-ten Wurzeln)

Eingabe: $\alpha, \delta \in G$ $(G \in \mathcal{G})$, $s, \in \mathbb{N}$ mit $\alpha \in <\delta>$ und ord $\delta = s$, $n \in [\![0, s-1]\!]$.

Ausgabe: *Es gibt keine n-te Wurzel* oder alle $\gamma \in <\delta>$ mit $n\gamma = \alpha$.

Berechne eine minimale Relation $(a, 1)$ für (δ, α)		(1)
$d \leftarrow EUKLALG(n, s)$		(2)
$(a', r) \leftarrow (a \div d, a \bmod d)$		(3)
IF $r \neq 0$		(4)
THEN	return { Es gibt keine n-te Wurzel von a }	(5)
ELSE	$(1, n_0, *) \leftarrow ERWEUKLALG(\frac{n}{d}, s)$	(6)
	$\gamma_0 \leftarrow n_0 \cdot a'\delta$; $\epsilon \leftarrow \frac{s}{d}\delta$	(7)
	FOR $i \leftarrow 1$ TO $d-1$ DO $\gamma_i \leftarrow \gamma_{i-1} + \epsilon$	(8)
	return $\{ \gamma_0, \ldots, \gamma_{d-1} \}$	(9)

Wenn wir nun den trivialen Fall $n = 0$ bzw. $n = s$ außer Acht lassen, so ist $d = \mathrm{ggT}(n, s) < \sqrt{s}$ und so ist die Anzahl der Gruppenperationen in Schritt (8) $O(\sqrt{s})$; Schritt (7) kann mit schneller Multiplikation realisiert werden und so wird auch der zusätzliche Aufwand für *alle* n-ten Wurzeln zu berechnen, vom Aufwand in Schritt (1) majorisiert. Wir haben:

7.9. Satz *Sei $G \in \mathcal{G}$ eine zyklische Gruppe mit Ordnung s. Dann kann man unter Kenntnis eines erzeugenden Elementes und der Ordnung s in*

- *$O(\sqrt{s})$ Gruppenadditionen und*
- *$O(\sqrt{s} \cdot \log s \cdot \log n_G)$ Bitoperationen*

entscheiden, ob ein Element $\alpha \in G$ eine n-te Wurzel in G hat und gegebenenfalls alle n-ten Wurzeln berechnen.

7.3 Wurzeln in beliebigen Gruppen

Wir verallgemeinern nun das im vorigen Abschnitt besprochene Verfahren auf eine beliebige endliche abelsche Gruppe. Die Grundidee wurde bereits vorgestellt: zuerst eine minimale Relation berechnen und dann in jeder zyklischen Restklassengruppe, die zu der von einem Basiselement erzeugten Untergruppe isomorph ist, die Lösbarkeit bzw. eine Lösung bestimmen. Der Algorithmus sieht dann folgendermaßen aus:

7.10. Algorithmus (alle n-ten Wurzeln)

Eingabe: $\alpha \in G$, $(s_1, \ldots, s_m)$ Struktur von G ($G \in \mathcal{G}$), $(\delta_1, \ldots, \delta_m)$ eine Basis von G und $n \in [\![0, s_m - 1]\!]$.

Ausgabe: *Es gibt keine n-te Wurzel* oder alle $\gamma \in G$ mit $n\gamma = \alpha$.

Berechne eine minimale Relation $(a_1, \ldots, a_m, 1)$ mit Vorinformation $(s_1, \ldots, s_m)$ für $(\delta_1, \ldots, \delta_m, \alpha)$ mit Algorithmus 4.10.			(1)
FOR $i \leftarrow 1$ TO m			(2)
	$d_i \leftarrow \mathit{EUKLALG}(n, s_i)$		(3)
	$(a_i', r_i) \leftarrow (a_i \div d_i, a_i \bmod d_i)$		(4)
IF $(r_1, \ldots, r_m) \neq (0, \ldots, 0)$			(5)
THEN	return { Es gibt keine n-te Wurzel von a }		(6)
ELSE	FOR $i \leftarrow 1$ TO m		(7)
		$(1, n_{i,0}, *) \leftarrow \mathit{ERWEUKLALG}(\frac{n}{d_i}, s_i)$	(8)
		$\gamma_{i,0} \leftarrow (n_{i,0} a_i' \bmod s_i)\delta_i$; $\epsilon_i \leftarrow \frac{s_i}{d_i}\delta_i$	(9)
	Berechne alle $\gamma_{i_1,\ldots,i_m} = \sum_{j=1}^{m} \gamma_{j,0} + i_j \epsilon_j$		(10)
	return { $\gamma_{i_1,\ldots,i_m} : 0 \leq i_j < d_j, 1 \leq j \leq m$ }		(11)

Es folgt

7.11. Satz *Sei $G \in \mathcal{G}$. Dann kann man unter Kenntnis der Struktur $(s_1, \ldots, s_m)$ und einer Basis von G in*

- $O(m \cdot \sqrt{|G|})$ *Gruppenadditionen und*
- $O(\sqrt{|G|} \cdot \log |G| \cdot \log n_G)$ *Bitoperationen*

entscheiden, ob ein Element $\alpha \in G$ *eine* n*-te Wurzel in* G *hat und gegebenenfalls die* t *vielen* n*-ten Wurzeln in zusätzlich*

- $O(t + \log |G|)$ *Gruppenadditionen und*
- $O(m \cdot \log^2 s_m)$ *Bitoperationen*

berechnen.

Beweis: Die Korrektheit folgt aus der Konstruktion. Betrachten wir die Laufzeit von jedem Schritt:

- Der Aufwand für Schritt (1) beträgt nach Satz 4.11
  - $O(m \cdot \sqrt{|G|})$ Gruppenadditionen und
  - $O(\sqrt{|G|} \cdot \log |G| \cdot \log n_G)$ Bitoperationen.
- Schritt (2) hat Komplexität
$$\sum_{i=1}^{m} O(\log n \cdot \log s_i) = O\left(\log n \cdot \sum_{i=1}^{m} \log s_i\right) = O(\log n \cdot \log |G|),$$
denn der Aufwand für den (einfachen) euklidischen Algorithmus beträgt
$$O((\max\{\log n, \log s_i\} - \log d_i + 1)(\log \min\{\log n, \log s_i\})) = O(\log n \cdot \log s_i)$$
und der Aufwand für die ganzzahlige Division ist
$$O(\log(a_i \div d_i) \cdot \log d_i) = O(\log s_i \cdot \log n);$$
da $n < s_m < n_G$, haben wir den ersten Teil der Behauptung bewiesen, denn bei Anweisung (5) ist die Lösbarkeit entschieden.
- Schritt (7) benötigt
$$O\left(\sum_{i=1}^{m} \log(n_{i,0} a_i' \bmod s_i) \cdot \log \frac{s_i}{d_i}\right) = O\left(\sum_{i=1}^{m} \log s_i\right) = O(\log |G|)$$
Gruppenoperationen (man kann Schritt (9) mit schneller Multiplikation ausführen) und
$$O\left(\sum_{i=1}^{m} \log^2 \max\{\frac{n}{d_i}, s_i\}\right) = O(m \cdot \log^2 s_m)$$
Bitoperationen (denn $n < s_m$) wegen der Komplexität des euklidischen Algorithmus.
- Schritt (10) benötigt für jede Lösung, aufgrund der Vorberechnung in Schritt (9), O(1) Gruppenoperation. Damit ergibt sich der zweite Teil der Behauptung. ■

Mit der Kenntnis der Struktur einer Gruppe ist zwar prinzipiell die Primfaktorzerlegung der Gruppenordnung noch nicht bekannt, aber da sich ja die s_i sukzessive teilen, ist die Primfaktorzerlegung im allgemeinen schnell bestimmt und man kann den in bestimmten Fällen wesentlich schnelleren Algorithmus zur Bestimmung einer minimalen Relation verwenden, der die Kenntnis der Zerlegung voraussetzt. Es zeigt sich hier, daß, wenn man die Berechnung *aller* n-ten Wurzeln vernachlässigt, der restliche Aufwand immer noch von dem zur Berechnung der minimalen Relation majorisiert wird.

7.12. Satz *Sei $G \in \mathcal{G}$. Dann kann man unter Kenntnis der Struktur $(s_1, \ldots, s_m)$, der Primfaktorzerlegung $\prod_{i=1}^{r} p_i^{e_i}$ der Gruppenordnung und einer Basis von G in*

- $O\left(\sum_{j=1}^{r} \left(e_j^2 \cdot (\sqrt{p_j})^{k-1}\right) + e^2 \cdot \log|G| + e \cdot k \cdot \log|G|\right)$ *Gruppenadditionen und*
- $O\left(\sum_{j=1}^{r} \left(e_j^2 \cdot (k-1) \cdot (\sqrt{p_j})^{k-1} \cdot \log p_j \cdot \log n_G\right) + s \cdot \log^2|G|\right)$ *Bitoperationen*

entscheiden, ob ein Element $\alpha \in G$ eine n-te Wurzel in G hat und gegebenenfalls alle t n-ten Wurzeln in zusätzlich

- $O(t + \log|G|)$ *Gruppenadditionen und*
- $O(m \cdot \log^2 s_m)$ *Bitoperationen*

berechnen.

7.4 Wenn die Struktur nicht bekannt ist

Für den Rest dieses Kapitels wollen wir demonstrieren, wie man nur mit Kenntnis der Gruppenordnung n-te Wurzeln ziehen kann. Dies geschieht dadurch, daß wir sukzessive für jeden Primteiler p von n eine p-te Wurzel zu ziehen versuchen. Hierzu reduzieren wir das Problem zuerst darauf, in p-Gruppen p-te Wurzeln zu bestimmen. Der Grund für diese Annahme liefert folgendes

7.13. Lemma *Sei G eine endliche abelsche Gruppe mit Ordnung h, p eine Primzahl und $h_0 \in \mathbb{N}$, so daß*

$$h = p^l \cdot h_0 \quad \textit{und} \quad \mathrm{ggT}(h_0, p) = 1.$$

Sei $\alpha \in G$. Dann gibt es $\chi \in G$ mit

$$\mathrm{ord}\,(p\chi - \alpha) \quad | \quad p^l.$$

Beweis: Es gilt $\mathrm{ggT}(h_0, p) = 1$; also können wir mit dem erweiterten euklidischen Algorithmus $h_0' \in [\![0, p-1]\!]$ berechnen mit

$$h_0' \cdot h_0 \equiv -1 \bmod p.$$

Also teilt p $h_0 \cdot h_0' + 1$ und

$$\chi := \frac{h_0 \cdot h_0' + 1}{p} \alpha \in G$$

ist wohldefiniert. Für $\gamma := p\chi - \alpha$ gilt:

$$\mathrm{ord}\,\gamma = \mathrm{ord}\,(p\chi - \alpha) = \mathrm{ord}\,((h_0 \cdot h_0' + 1)\alpha - \alpha) = \mathrm{ord}(h_0 h_0' \alpha);$$

durch die Multiplikation mit h_0 wird $h_0'\alpha$ (ähnlich wie bei der modularen Berechnung von minimalen Relationen) in eine p-Gruppe transportiert, denn

$$h = p^l \cdot h_0 \quad \text{und} \quad \mathrm{ggT}(h_0, p) = 1.$$

■

Wir haben im Beweis auch gesehen, wie man χ berechnen kann und somit das Problem algorithmisch darauf reduziert, p-te Wurzeln in p-Gruppen zu bestimmen. Insbesondere können wir mit dieser Idee *deterministisch* n-te Wurzeln in Gruppen ziehen, deren Ordnung relativ prim zu n ist:

7.14. Algorithmus (n-te Wurzel, n prim zu $|G|$)

Eingabe: $\alpha \in G$ ($G \in \mathcal{G}$), n mit $n \nmid h = |G|$.

Ausgabe: $\chi \in G$, $p\chi = \alpha$.

$(1, h', *) \leftarrow ERWEUKLALG(h, n)$	(1)
return $\left\{\left(\left(\frac{-h' \cdot h+1}{n}\right) \bmod \|G\|\right) \alpha\right\}$	(2)

7.15. Satz *Sei $G \in \mathcal{G}$, $\alpha \in G$, $n \in \mathbb{N}$, $\mathrm{ggT}(n, |G|) = 1$. Dann kann man eine n-te Wurzel von α in*

- *$O(\log |G|)$ Gruppenadditionen und*
- *$O(\log^2 |G|)$ Bitoperationen*

berechnen.

Beweis: Zur Korrektheit des obigen Algorithmus beachte man, daß sich die Überlegung im Beweis von obigem Lemma auf n relativ prim zu $|G|$ übertragen läßt und zusätzlich $p\chi - \alpha =$ gilt.
Zur Laufzeit ist zu bemerken, daß der zweite Schritt mit schneller Multiplikation realisiert werden kann. ∎

7.5 p-te Wurzeln in p-Gruppen

Wir wollen nun zeigen, wie man auch p-te Wurzeln in p-Gruppen ziehen kann; allerdings gibt es hierfür nur probabilistische Verfahren. Die Grundidee zur Lösung dieses Problems illustriert folgendes

7.16. Lemma *Sei G eine p-Gruppe, $\gamma, \gamma_1 \in G \setminus \{\}$. Falls es eine minimale Relation (x_0, x_1) für (γ, γ_1) gibt mit $x_1 > 1$ und $p \nmid x_0$, so gibt es $z \in \mathbb{N}$ mit*

$$\gamma = z(\gamma + \gamma_1) \quad \text{und} \quad p \mid z.$$

Beweis: Sei $x = p^k$ ($k \in \mathbb{N}$), die Ordnung von γ in G. Wenn es eine minimale Relation (x_0, x_1) für (γ, γ_1) gibt mit $x_1 > 1$, so ist $x_1 = p^l$, ($l \in \mathbb{N}$), denn x_1 ist die Ordnung von γ_1 als Element der Faktorgruppe $G/ < \gamma >$. Nun teilt p nicht x_0, also gilt

$$\mathrm{ggT}(x_1 - x_0, x) = 1$$

und folglich gibt es z_1 und z_2 mit

$$z_1(x_1 - x_0) + z_2 x = 1.$$

Mit $z = z_1 \cdot x_1$ folgt nun unsere Behauptung, denn p teilt z, denn $z = z_1 x_1 = z_1 p^l$ und

$$\begin{aligned}
z(\gamma + \gamma_1) &= z_1 x_1 \gamma + z_1 x_1 \gamma_1 \\
&= z_1 x_1 \gamma + z_1(-x_0\gamma) \quad \text{wegen der minimalen Relation } (x_0, x_1) \\
&= z_1(x_1 - x_0)\gamma \\
&= (1 - z_2 x)\gamma \\
&= \gamma - z_2(x\gamma) \\
&= \gamma - z_2 \\
&= \gamma.
\end{aligned}$$

∎

Aufgrund von Lemma 7.16 können wir nun p-te Wurzeln in p-Gruppen berechnen, indem wir zufällig ein Element aus der Gruppe wählen, eine minimale Relation berechnen und darauf hoffen, daß diese minimale Relation die gewünschte Gestalt hat. Der Algorithmus hat natürlich nur dann eine Ausgabe, wenn es tatsächlich eine p-te Wurzel von γ in G gibt.

7.17. Algorithmus (p-te Wurzeln in p-Gruppen)

Eingabe: $\gamma \in G$ p-Gruppe ($G \in \mathcal{G}$), p; γ hat p-te Wurzel in G.

Ausgabe: $\epsilon \in G$ mit $p\epsilon = \gamma$.

$x \leftarrow \operatorname{ord}\gamma$; $(x_0, x_1) \leftarrow (0,1)$		(1)
WHILE $x_1 = 1$ OR $p \mid x_0$		(2)
	$\gamma_1 \in_R G$	(3)
	Berechne minimale Relation (x_0, x_1) für (γ, γ_1)	(4)
$(d, z_1, z_2) \leftarrow \mathit{ERWEUKLALG}((x_1 - x_0), x)$		(5)
return $\{ \frac{z_1 \cdot x_1}{p}(\gamma + \gamma_1) \}$		(6)

Um eine Aussage über die erwartete Laufzeit machen zu können, müssen wir erst präzisieren, unter welchen Voraussetzungen — die Struktur der Gruppe und die Wahl von γ betreffend — der Algorithmus erfolgreich ist.

7.18. Lemma *Sei G eine endliche abelsche Gruppe mit Struktur $(p^{l_1}, \ldots, p^{l_r})$ und einer Basis $(\delta_1, \ldots, \delta_r)$, $\gamma, \gamma_1 \in G$ mit*

$$\gamma = \sum_{i=1}^{r} x_i \delta_i \quad \textit{und} \quad \gamma_1 = \sum_{i=1}^{r} y_i \delta_i,$$

wobei $0 \le x_i, y_i < p^{l_i}$ für $i = 1, \ldots, r$. Seien weiterhin für $i = 1, \ldots, r$

$$x_i = p^{t_i} x_i' \quad \textit{und} \quad y_i = p^{s_i} y_i',$$

wobei $p \nmid x_i', y_i'$. Dann gibt es genau dann eine minimale Relation (z_0, z_1) für (γ, γ_1) mit $z_1 > 1$ und $p \nmid z_0$, wenn es $k \in \mathbb{N}$ gibt mit

$$\left(\mathop{\forall}_{\substack{i \in [\![1,r]\!] \\ x_i' > 0}} s_i = t_i - k \right) \quad \textit{und} \quad \left(\mathop{\forall}_{\substack{i \in [\![1,r]\!] \\ x_i' = 0}} s_i \le l_i - k \right).$$

Beweis: Da z_1 die Ordnung von γ_1 als Element der Faktorgruppe $G/<\gamma>$ eine p-Potenz ist, gilt:

$$z_1 > 1 \quad \Longleftrightarrow \quad z_1 = p^k \text{ mit } k \in \mathbb{N}.$$

Also finden wir genau dann eine minimale Relation (z_0, z_1) für (γ, γ_1) mit den obigen Bedingungen, wenn es ein $k \in \mathbb{N}$ gibt mit

$$p^k \gamma_1 = -z_0 \gamma, \quad \text{mit} \quad p \nmid z_0.$$

Dies ist, mit den obigen Darstellungen, äquivalent zu:

$$p^k \sum_{i=1}^{r} p^{s_i} y_i' \delta_i = -z_0 \sum_{i=1}^{r} p^{t_i} x_i' \delta_i \quad \text{mit} \quad p \nmid z_0.$$

bzw. zu:

$$\forall_{i=1,\dots,r} \quad p^k p^{s_i} y_i' \equiv -z_0 p^{t_i} x_i' \bmod p^{l_i} \quad \text{mit} \quad p \nmid z_0.$$

Die zusätzliche Bedingung $p \nmid x_0$ ist äquivalent zu:

$$(1) \quad p^k p^{s_i} y_i' \equiv 0 \bmod p^{l_i} \quad \Longleftrightarrow x_i' = 0$$

und

$$(2) \quad p^k p^{s_i} \equiv p^{t_i} \bmod p^{l_i} \quad \Longleftrightarrow x_i' > 0,$$

Daraus folgt unmittelbar unsere Behauptung. ■

Somit können wir nun die Wahrscheinlichkeit für die Wahl eines günstigen γ_1 bestimmen.

7.19. Lemma *Sei G eine endliche abelsche p-Gruppe mit Struktur $(p^{l_1}, \dots, p^{l_r})$, $\gamma = \sum_{i=1}^{r} p^{t_i} x_i' \delta_i \in G$ so daß es eine p-te Wurzel von γ in G gibt und $g \in \mathbb{N}$, so daß für alle $\gamma \in G$*

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Dann ist die Wahrscheinlichkeit, ein Element $\gamma_1 \in G$ zu wählen, so daß es eine minimale Relation (z_0, z_1) für (γ, γ_1) gibt mit $z_1 > 1$ und $p \nmid x_o$

$$P := \left(\frac{p-1}{p}\right)^s \cdot \frac{p^{\sum_{i:x_i \neq 0} (l_i - t_i)}}{g} \cdot \frac{p^{(t_{i_{min}}+1)r} - 1}{p^r - 1}, \tag{7.1}$$

wobei s die Anzahl die Indizes i, für die $x_i \neq 0$ und i_{min} der Index, für den $t_{i_{min}} = \min t_i$ ist.

Beweis: Die Wahrscheinlichkeit, ein solches $\gamma_1 \in G$ zu wählen, beträgt (mit den Bezeichnungen des vorhergehenden Lemmas)

$$\sum_{\substack{\gamma_1 : (\forall i, x_i = 0 : s_i < l_i - k) \\ \text{und } (\forall i, x_i \neq 0 : s_i = t_i - k)}} p(\gamma_1 \in_R G)$$

$$= \frac{1}{g} \cdot \frac{|\{\gamma_1 : \forall i, x_i = 0 : s_i < l_i - k \text{ und } \forall i, x_i \neq 0 : s_i = t_i - k\}|}{|G|}.$$

Wir bestimmen also jetzt

$$N := |\{\gamma_1 : \forall i, x_i = 0 : s_i < l_i - k \text{ und } \forall i, x_i \neq 0 : s_i = t_i - k\}|.$$

Betrachte zuerst eine einzelne "Komponente": sei i ein Index, für den $x_i = 0$ gilt. Dann ist die Anzahl der $p^{s_i} y_i' \in \mathbb{Z}/p^{l_i}\mathbb{Z}$ für die $s_i > l_i - k$ gleich der Anzahl der $y_i \in \mathbb{Z}/p^{l_i}\mathbb{Z}$, deren Ordnung $\leq p^k$ ist, also p^k. Ist andererseits j ein Index, für den $x_j \neq 0$ ist, so ist die Anzahl der $p^{s_i} y_i' \in \mathbb{Z}/p^{l_i}\mathbb{Z}$ für die $s_i = t_i - k$ gleich der Anzahl der $y_i \in \mathbb{Z}/p^{l_i - s_i}\mathbb{Z}$, die relativ prim zu p sind, also $\phi(p^{l_i - s_i}) = \phi(p^{l_i - t_i + k})$.
Wenn wir nun die Anzahl aller γ_1 bestimmen wollen, so müssen wir die Anzahl der Möglichkeiten in jeder einzelnen Komponente miteinander multiplizieren und über alle möglichen k summieren. k kann aber höchstens $t_{i_{min}} = \min t_i$ sein, denn sonst ist s_i negativ, was nicht sein kann. Also haben wir:

$$N = \sum_{k=1}^{t_{i_{min}}} \left(\prod_{i:x_i \neq 0} \phi\left(p^{l_i - t_i + k}\right) \right) \cdot \left(\prod_{i:x_i = 0} p^k \right);$$

sei s nun die Anzahl der Indizes, für die $x_i \neq 0$. Dann haben wir

$$\begin{aligned} N &= \sum_{k=1}^{t_{i_{min}}} \left(\prod_{i:x_i \neq 0} (p^{l_i - t_i + k - 1}(p-1)) \right) \cdot \left(\prod_{i:x_i = 0} p^k \right) \\ &= (p-1)^s \sum_{k-1}^{t_{i_{min}}} p^{k(r-s)} \prod_{i:x_i \neq 0} p^{l_i - t_i + k - 1} \\ &= (p-1)^s \sum_{k-1}^{t_{i_{min}}} p^{k(r-s)+s(k-1)+\sum_{i:x_i \neq 0}(l_i - t_i)} \\ &= \frac{(p-1)^s}{p^s} \sum_{k-1}^{t_{i_{min}}} (p^r)^k \cdot p^{\sum_{i:x_i \neq 0}(l_i - t_i)} \\ &= \frac{(p-1)^s}{p^s} \cdot p^{\sum_{i:x_i \neq 0}(l_i - t_i)} \cdot \frac{p^{(t_{i_{min}}+1)r} - 1}{p^r - 1}. \end{aligned}$$

Daraus folgt die Behauptung. ■

Diese Wahrscheinlichkeit sieht sehr kompliziert aus; in der Tat hängt sie von der Struktur der Gruppe und dem vorliegenden γ ab. Da es allerdings unwahrscheinlich ist, daß ein Eintrag x_i von γ in einer Komponente 0 ist (nämlich $\frac{1}{p^{l_i}}$) und andererseits $t_{i_{min}} = 1$ sehr wahrscheinlich ist, verallgemeinert sich der Ausdruck wesentlich und wir haben unter diesen Annahmen nach einiger Rechnung die Wahrscheinlichkeit

$$\frac{(p-1)^r}{p^r} \cdot \frac{p^r + 1}{p^{\sum_{i=1}^{r} t_i}} \cdot \frac{1}{g}.$$

Zwei Beobachtungen können wir über diese Wahrscheinlichkeit machen: es gibt einen von γ unabhängigen Anteil, der nur von der Struktur der Gruppe abhängt (den linken) und einen, der dokumentiert, daß es umso schwieriger ist, eine Wurzel für ein Element der Gruppe zu finden, desto unterschiedlicher "groß" die t_i sind. Sind die t_i groß, aber nahe beieinander, so ist $t_{i_{min}}$ ebenfalls groß und der zweite Ausdruck wird von dem dritten etwa ausgeglichen. Ist allerdings $t_{i_{min}}$ klein und andere t_i trotzdem groß, wird die Wahrscheinlichkeit immer kleiner.

Nun können wir die erwartete Laufzeit von Algorithmus 7.17 angeben:

7.20. Satz *Sei $G \in \mathcal{G}$ p-Gruppe, $\gamma \in G$, so daß es eine p-te Wurzel von γ in G gibt. Dann kann man die Berechnung einer p-ten Wurzel von γ nach*

- $\frac{1}{P}$ *pseudozufälligen Erzeugungen von Elementen von G,*
- $O(\frac{1}{P} \cdot \sqrt{|G|})$ *Gruppenadditionen und*
- $O(\frac{1}{P} \cdot \sqrt{|G|} \cdot \log|G| \cdot \log n_G)$ *Bitoperationen*

erwarten, wobei P die in 7.1 angegebene Wahrscheinlichkeit ist.

Beweis: Am Anfang muß man einmal die Ordnung von γ bestimmen; dies wird aber durch den Aufwand, eine minimale Relation für (γ, γ_1) zu berechnen, majorisiert. Der Faktor $\frac{1}{P}$ ergibt sich als der Erwartungswert der benötigten Versuche, um ein geeignetes γ_1 zu bekommen. Am Schluß muß man noch eine GGT-Berechnung und eine schnelle Multiplikation machen, der Aufwand von beiden wird durch wiederum durch den Aufwand der Minimalen-relationen-Berechnung majorisiert. Somit ergibt sich die Behauptung. ■

Der eben besprochene Algorithmus war ein Las-Vegas-Algorithmus, denn wenn er terminiert, gibt er ein richtiges Ergebnis aus; dabei benötigte er nicht die Kennnis der Gruppenordnung. Nun wollen wir noch einen Monte-Carlo-Algorithmus angeben, der nach einer bestimmten Anzahl von erfolglosen Versuchen, ein geeignetes γ_1 zu bestimmen, behauptet, es gebe keine p-te Wurzel von γ in G. Hierzu schätzen wir obiges P nach unten ab (sei $|G| = p^d$):

$$\begin{aligned} P &= \left(\frac{p-1}{p}\right)^s \cdot \frac{p^{\sum_{i:x_i\neq 0}(l_i-t_i)}}{g} \cdot \frac{p^{(t_{i_{min}}+1)r}-1}{p^r-1} \\ &\geq \left(\frac{p-1}{p}\right)^r \cdot \frac{1}{p^d} \cdot (p^r+1) \cdot \frac{1}{g} \\ &\geq (p-1)^r \cdot \frac{1}{p^d} \cdot \frac{1}{g} \\ &\geq \frac{p-1}{p^d} \cdot \frac{1}{g}. \end{aligned}$$

Dies ist eine äußerst grobe Abschätzung, doch sie hängt nur noch von p und d ab, was wir beide als bekannt voraussetzen. Um nun einen Algorithmus angeben zu können, der in Abhängigkeit einer vorgegebenen Genauigkeit ein Ergebnis ausgibt, benötigen wir folgendes

7.21. Lemma *Man betrachte ein Problem, von dem man nicht weiß, ob es eine Lösung hat. Man breche einen probabilistischen Algorithmus, der mit Wahrscheinlichkeit W eine Lösung für das Problem liefert, nach k erfolglosen Versuchen ab und behaupte, es gebe keine Lösung für das Problem. Dann ist die Wahrscheinlichkeit, mit dieser Behauptung richtig zu liegen,*

$$1-(1-w)^k.$$

Beweis: Man irrt sich genau dann, wenn es eine Lösung für das Problem gibt, der probabilistische Algorithmus aber in k Versuchen keine Lösung berechnet hat. Daraus ergibt sich die Behauptung. ■

Hat man also eine Genauigkeitsschranke $c > 0$, so daß man ein richtiges Ergebnis mit Wahrscheinlichkeit $1-c$ haben möchte, so muß die Anzahl der erfolglosen Versuche

$$\leq \left\lceil \frac{\log c}{\log(1-W)} \right\rceil$$

sein, denn

$$c > (1-W)^k \Leftrightarrow \log c > k \cdot \log(1-W) \Leftrightarrow \frac{\log c}{\log(1-W)} < k.$$

Angewandt auf unsere Situation bedeutet dies, daß die Anzahl der benötigten Schritte, um eine Genauigkeit $c \cdot |G|^{o(1)}$ zu erreichen,

$$\lceil \frac{\log c}{\log(1-\frac{p-1}{p^d\cdot g})} \rceil$$

beträgt. Somit haben wir folgenden Algorithmus:

7.22. Algorithmus (p-te Wurzeln in p-Gruppen)

Eingabe: $\gamma \in G$ p-Gruppe ($G \in \mathcal{G}$), p, d mit $|G| = p^d$, $c > 0$.

Ausgabe: $\epsilon \in G$ mit $p\epsilon = \gamma$ oder *es gibt keine p-te Wurzel von γ in G.*

$x \leftarrow \operatorname{ord}\gamma$; $(x_0, x_1) \leftarrow (0,1)$; $k \leftarrow 0$		
WHILE $(x_1 = 1$ OR $p \mid x_0)$ AND $k < \lceil \frac{\log c}{\log(1-\frac{p-1}{p^d \cdot g})} \rceil$		
	$\gamma_1 \in_R G$	
	Berechne minimale Relation (x_0, x_1) für (γ, γ_1)	
	$k \leftarrow k+1$	
IF $k < \boxed{\frac{\log c}{\log(1-\frac{p-1}{p^d \cdot g})}}$		
THEN	$(d, z_1, z_2) \leftarrow ERWEUKLALG((x_1 - x_0), x)$	
	return $\{ \frac{z_1 \cdot x_1}{p}(\gamma + \gamma_1) \}$	
ELSE	return $\{$ *es gibt keine p-te Wurzel in G* $\}$	

Es folgt

7.23. Satz *Sei $G \in \mathcal{G}$ p-Gruppe, $\gamma \in G$ und $g \in \mathbb{N}$, so daß für alle $\gamma \in G$*

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Dann kann man in

- $\frac{\log c}{\log(1-\frac{p-1}{|G| \cdot g})}$ *pseudozufälligen Erzeugungen von Elementen von G,*
- $O(\frac{\log c}{\log(1-\frac{p-1}{|G| \cdot g})} \cdot \sqrt{|G|})$ *Gruppenadditionen und*
- $O(\frac{\log c}{\log(1-\frac{p-1}{|G| \cdot g})} \cdot \sqrt{|G|} \cdot \log|G| \cdot \log n_G)$ *Bitoperationen*

mit Wahrscheinlichkeit $> 1 - c$ entscheiden, ob γ eine p-te Wurzel in G hat und gegebenenfalls eine berechnen.

Beweis: Der einzige Unterschied zwischen Algorithmus 7.22 und Algorithmus 7.17 besteht in der maximalen Anzahl von Versuchen, ein nicht redundantes Element zu bestimmen; diese haben wir oben bestimmt. Mit der entsprechenden Änderung in Satz 7.20 ergibt sich die Behauptung. ∎

7.6 n-te Wurzeln in beliebigen Gruppen

Nun können wir die n-te Wurzel eines Elements γ, falls es existiert, in beliebigen Gruppen berechnen, wenn die Gruppenordnung bekannt ist (wir verfolgen ab hier nur die Monte-Carlo-Alternative der Berechnung der p-ten Wurzel in p-Gruppen; die Las-Vegas-Version ist analog erweiterbar). Hierzu berechnet man den ggT d von n und der Gruppenordnung; eine $\frac{d}{n}$-te Wurzel kann dann deterministisch mit Algorithmus 7.14 bestimmt werden. Ausgehend von diesem Gruppenelement versucht man dann sukzessive, p-te Wurzeln zu bestimmen für jeden Primteiler p von d. Hierbei reduziert man dabei jedesmal das Problem mit einem zu Algorithmus 7.14 analogen auf die Suche einer p-ten Wurzel in einer p-Gruppe.

7.24. Algorithmus (n**-te Wurzeln beliebig**)

Eingabe: $n \in \mathbb{N}$, $\alpha \in G$ $(G \in \mathcal{G})$, $h = |G|$, $c > 0$.

Ausgabe: $\chi \in G$ mit $n\chi = \alpha$ oder *es gibt keine* n*-te Wurzel von* α *in* G.

$d \leftarrow \mathrm{ggT}(n, h)$				(1)
$n' \leftarrow \frac{n}{d}$				(2)
$(1, h', *) \leftarrow \mathit{ERWEUKLALG}(h, n')$				(3)
$\alpha \leftarrow \left(\frac{-h \cdot h' + 1}{n'}\right) \alpha$				(4)
FORALL $p \mid d$, p prim				(5)
	Bestimme l, h_0 mit $h = p^l \cdot h_0$, $p \nmid h_0$			(6)
	$(1, h_0', *) \leftarrow \mathit{ERWEUKLALG}(h_0, p)$			(7)
	$\alpha \leftarrow \left(\frac{-h_0' \cdot h_0 + 1}{p}\right) \alpha$			(8)
	$x \leftarrow \mathrm{ord}\, \alpha$; $(x_0, x_1) \leftarrow (0, 1)$; $k \leftarrow 0$			(9)
	WHILE $(x_1 = 1$ OR $p \mid x_0)$ AND $k < \lceil \frac{\log c}{\log(1 - \frac{p-1}{p^l})} \rceil$			(10)
		$\gamma \in_R G$		(11)
		$\gamma_1 \leftarrow h_0 \gamma$		(12)
		Berechne minimale Relation (x_0, x_1) für (α, γ_1)		(13)
	IF $k < \lceil \frac{\log c}{\log(1 - \frac{p-1}{p^l})} \rceil$			(14)
	THEN	$(d, z_1, z_2) \leftarrow \mathit{ERWEUKLALG}((x_1 - x_0), x)$		(15)
		$\alpha \leftarrow \frac{z_1 \cdot x_1}{p}(\alpha + \gamma_1)$		(16)
	ELSE	return { *es gibt keine* p*-te Wurzel in* G }		(17)
	$d \leftarrow \frac{d}{n}$			(18)

Der Algorithmus ergibt sich durch Zusammensetzung von Algorithmus 7.14 und Algorithmus 7.17 für jeden Primteiler p von n. Zu beachten ist dabei, die daß die pseudozufällig erzeugten Elemente, die sich in der gesamten Gruppe befinden können, durch Die Multiplikation mit h_0 in die gerade betrachtete p-Gruppe transportiert werden; diese Multiplikationen können mit schneller Multiplikation durchgeführt werden. Es gilt:

7.25. Satz *Sei* $G \in \mathcal{G}$, $\alpha \in G$, $n \in \mathbb{N}$, $d = \mathrm{ggT}(n, |G|)$, $d = \prod_{i=1}^{r} p_i^{e_i}$, $|G| = p_i^{l_i} \cdot m_i$ *mit* $p_i \nmid m_i$ *für* $i = 1, \ldots, r$ *und* $g \in \mathbb{N}$, *so daß für alle* $\gamma \in G$

$$p(\gamma \in_R G) \geq \frac{1}{|G|} \cdot \frac{1}{g}.$$

Dann kann man in

- $\log c \cdot \sum_{i=1}^{r} \frac{1}{\log(1 - \frac{p_i - 1}{p_i^{l_i} \cdot g})}$ *pseudozufälligen Erzeugungen von Elementen von* G,

- $O\left(\log c \cdot \left(\sum_{i=1}^{r} \frac{1}{\log(1 - \frac{p_i - 1}{p_i^{l_i} \cdot g})} \cdot \sqrt{p_i^{l_i}}\right)\right)$ *Gruppenadditionen und*

- $O\left(\log c \cdot \log n_G \cdot \left(\sum_{i=1}^{r} \frac{1}{\log(1-\frac{p_i-1}{p_i^{l_i}\cdot g})} \cdot \sqrt{p_i^{l_i}} \cdot \log p_i^{l_i}\right)\right)$ *Bitoperationen*

mit Wahrscheinlichkeit $> 1 - c$ *entscheiden, ob* α *eine* n*-te Wurzel in* G *hat und gegebenenfalls eine berechnen.*

Die Aussage ergibt sich durch Zusammensetzen der Ergebnisse für die einzelnen Runden des Algorithmus zur Berechnung einer p-ten Wurzel in einer p-Gruppe.

Kapitel 8

Ergebnisse

In diesem abschließenden Kapitel wollen wir für die in Kapitel 2 eingeführten Beispiele die Ergebnisse unserer Arbeit demonstrieren. Dabei haben wir keinerlei Wissen über die spezielle Struktur der betrachteten Gruppe benutzt. Spezielle Algorithmen können dies natürlich ausnutzen; so kann man hier nicht erwarten, eine Verbesserung der optimalen Laufzeit zu erhalten. Allerdings geben diese Ergbnisse ein Gefühl für die Güte der besprochenen Algorithmen.

In der multiplikativen Gruppe $\mathbb{F}_n^*$ ist man oft an Wurzeln interessiert; die Gruppenstruktur interessiert nur wenig und die Gruppenordnung ist bekannt. Hier können wir also den zweiten Teil des 7. Kapitels anwenden und mit dem Aufwand für Gruppenoperationen in $\mathbb{F}_n^*$ zusammensetzen. So erhalten wir:

8.1. Satz *Sei $a \in \mathbb{F}_n^*$, $\phi(n) = \prod_{i=1}^{r} p_i^{l_i}$. Dann kann man die Berechnung einer m-ten Wurzel von a nach*

- $O\left(\sum_{i=1}^{r} \frac{1}{P_i}\sqrt{p_i^{l_i}}\right) \cdot \log^2 \phi(n)$ *Bitoperationen*

erwarten, wobei die P_i die in Lemma 7.19 definierten, von der Gruppe abhängigen Wahrscheinlichkeiten sind.

Bei elliptischen Kurven möchte man die Punkteanzahl (die Gruppenordnung) und den Typ (die Struktur) wissen. Wir setzen die bewiesenen Ergebnisse aus Kapitel 6 und den Aufwand für die Gruppenoperationen zusammen und erhalten:

8.2. Satz *Sei E eine elliptische Kurve über $\mathbb{F}_p$, $p > 16$, $t \in \mathbb{N}$. Dann kann man die Berechnung der Punkteanzahl der Kurve nach*

- $O\left(p^{\frac{1}{4}} \cdot \log^5 p \cdot \frac{1}{1-\left(1-\frac{1}{4}\right)^t} + t \cdot \log^2 p\right)$ *Bitoperationen*

und die Berechnung des Typs nach

- $O\left(\sqrt{p} \cdot \log^5 p \cdot \frac{1}{1-\left(1-\frac{1}{4}\right)^t} + \left(\frac{1}{1-\left(1-\frac{1}{4}\right)^t}\right)^3 \cdot \log^8 p + t \cdot \log^2 p\right)$ *Bitoperationen*

erwarten.

Für Klassengruppen über imaginärquadratischen Zahlkörpern will man die Klassenzahl, also die Ordnung der Gruppe berechnen. Hier gilt:

8.3. Satz *Sei $\Delta < 0$. Dann kann man die Berechnung der Klassenzahl $h(\Delta)$ nach*

- $O\left(|\Delta|^{\frac{1}{4}} \cdot \frac{\log^{5.5}|\Delta|}{\log|\Delta|-1} \cdot \log\log|\Delta| \cdot \log\log\log|\Delta|\right)$ *Bitoperationen*

erwarten.

Für alle hier angegebenen Sätze gilt: wir nehmen den entsprechenden Satz, den wir in dieser Arbeit für endliche abelsche Gruppen bewiesen haben, setzen den Aufwand für Gruppenoperationen und die "untere Schranke" g der geometrischen Abweichung von der Gleichverteilung der Ausgabewahrscheinlichkeit des Algorithmus, der Elemente aus der Gruppe pseudozufällig erzeugt, dort ein und verwenden die Tatsache, daß wir ein Intervall für die Gruppenordnung der Größe $O(\sqrt{|G|})$ kennen.

Allgemein gilt folgender

8.4. Satz *Für alle $G \in \mathcal{G}$ gilt: die Ordnung und die Struktur von G können in*

- $O\left(|G|^{\frac{1}{2}+o(1)}\right)$ *Bitoperationen*

probabilistisch berechnet werden.

Beweis: Für alle $G \in \mathcal{G}$ gilt:

- der Aufwand, Gruppenoperationen zu realisieren, ist subpolynomiell in der Gruppenordnung und
- die maximale geometrische Abweichung der Wahrscheinlichkeit der Erzeugung eines Elementes der Gruppe von der Gleichverteilung ist ebenfalls subpolynomiell.

Durch Einsetzen in Satz 6.26 ergibt sich die Behauptung. ■

Literaturverzeichnis

[Be90] R. Berger, Skript zur Vorlesung *Algebra I*,WS 90/91.

[Be92] F.-D. Berger, *Faktorisieren mit elliptischen Kurven*, Diplomarbeit, Universität des Saarlandes, voraussichtlich 1992.

[Bu88] P. Bundschuh, *Einführung in die Zahlentheorie*, Springer Verlag, New York, 1988.

[Co91] H. Cohen, *A course in computational algebraic number theory*, Buch-Manuskript, 1991.

[Co74] G. E. Collins, *The Computing Time of the Euclidean Algorithm*, SIAM J. Comput., Vol. 3, No. 1, März 1974, S. 1-10.

[DoKaTr87] P.D. Domich, R. Kannan und L.E. Trotter, *Hermite Normal Form Computation using Modulo Determinant Arithmetic*, Math. Op. Res. 12 (1987), S. 50–59.

[Dü92] S. Düllmann, *Ein Algorithmus zur Bestimmung der Klassengruppe positiv definiter binärer quadratischer Formen*, Dissertation, Universität des Saarlandes, 1992.

[HaMcCu] J.L. Hafner und K.S. McCurley, *A rigorous subexponential algorithm for computation of class groups*, Journal AMS, to appear.

[Hu74] T.W. Hungerford, *Algebra*, Springer Verlag, New York, 1974.

[Kn73] D. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Reading, Massachusets, 1973.

[Ko87] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer Verlag, New York, 1987.

[Le87] A. K. Lenstra, *Fast and rigorous factorization under the generalized Riemann hypothesis*, Technical Report 87-007, University of Chicago, 1987

[MoOl90] F. Morain, J. Olivos, *Speeding up the Computations on an Elliptic Curve using Addition-Subtraction Chains*, Thèse, Université Claude Bernard, Lyon, 1990.

[Mü91] V. Müller, *Die Berechnung der Punkteanzahl von elliptischen Kurven über endlichen Primkörpern*, Diplomarbeit, Universität des Saarlandes, 1991.

[Pe91] A. Pethö, Skript zur Vorlesung *Computer Algebra*, Universität des Saarlandes, WS 90/91.

[Pr78] K. Prachar, *Primzahlverteilung*, Springer Verlag New York, 1978.

[Ri85] H. Riesel, *Prime Numbers and Computer Methods for Factorisation*, Progress in Mathematics, Birkhäuser, Boston, 1985.

[Sh70] D. Shanks, *Class Number, A Theory of Factorization and Genera,* Proceedings Symposium Pure Mathematics, Vol. 20, American Mathematical Society, 1970, S. 415–440

[Si86] J. Silverman, *The Arithmetic of Elliptic Curves*, Graduate Texts in Mathematics 106, Springer Verlag, New York, 1986.

[WePa91] D. Weber, S. Paulus, *Berechnung des GGT nach Lehmer und nach Stein*, Studienarbeit zum Fortgeschrittenenpraktikum, Universität des Saarlandes, 1991.