Projekt 3: Multiplikation großer Zahlen

Im Folgenden ist $p \in \mathbb{N}$ immer eine Primzahl. Wir bezeichnen mit $\mathbb{Z}_p$ den Restklassenkörper $\{0, 1, \ldots, p-1\}$ versehen mit der $\mod$ -Arithmetik.

1. Überlegen Sie sich, daß die "normalen" Rechenregeln für Potenzen gelten, d.h. für $0 \neq a$ und $i, j \in \mathbb{Z}$ gilt (man setzt: $a^0 \equiv 1 \mod p$ und a^{-1} bezeichnet die multiplikative Inverse von a)

$$a^{i+j} \equiv a^i a^j \mod p, \qquad a^{ij} = (a^i)^j.$$

a) Für $n \in \mathbb{N}$ heißt $\omega \in \mathbb{Z}_p$ eine *primitive* n-te Wurzel, falls

$$\omega^n \equiv 1 \mod p, \qquad \text{und} \qquad \omega^i \not\equiv 1 \mod p, \qquad i = 1, \ldots, n-1.$$

Zeigen Sie: Ist ω eine primitive n-te Wurzel, dann sind die Zahlen ω^i, $i = 0, \ldots, n-1$, paarweise verschieden.

b) Sei ω eine primitive n-te Wurzel. Zeigen Sie:

$$\sum_{i=0}^{n-1} \omega^{ij} \equiv \begin{cases} 0 & \text{falls } j \not\equiv 0 \mod n \\ n & \text{falls } j \equiv 0 \mod n \end{cases} \tag{1}$$

c) Zeigen Sie: Sei n gerade. Dann gilt: Falls ω eine primitive n-te Wurzel ist, dann ist ω^2 eine primitive $n/2$-te Wurzel.

2. **a)** Sei $n \in \mathbb{N}_0$ und $\omega \in \mathbb{Z}_p$ eine n-te primitive Wurzel. Dann ist die DFT definiert als

$$\mathbb{Z}_p^n \to \mathbb{Z}_p^n, \qquad (y_j)_{j=0}^{n-1} \mapsto \mathcal{F}_n(y) := (\sum_{j=0}^{n-1} \omega^{kj} y_j)_{k=0}^{n-1}.$$

Geben Sie $\mathcal{F}_n^{-1}$ an.

b) Formulieren Sie für Zweierpotenzen $n = 2^q$, $q \in \mathbb{N}_0$ einen FFT-Algorithmus zu schnellen Realisierung von $\mathcal{F}_n$. Formulieren Sie ebenfalls den IFFT-Algorithmus zur schnellen Realisierung von $\mathcal{F}_n^{-1}$.

3. Sei $\mathbb{Z}_{p,per}^n$ der Vektorraum (über $\mathbb{Z}_p$) der n-periodischen Folgen, d.h.

$$\mathbb{Z}_{p,per}^n = \{(\mathfrak{f}_j)_{j\in\mathbb{Z}} \,|\, \mathfrak{f}_j \in \mathbb{Z}_p, \quad \mathfrak{f}_{j+n} = \mathfrak{f}_j \quad \forall j \in \mathbb{Z}\}$$

Auf $\mathbb{Z}_{p,per}^n$ definiert man die DFT wie oben:

$$\mathcal{F}_n(\mathfrak{f}) = (\sum_{j=0}^{n-1} \omega^{kj} \mathfrak{f}_j)_{k\in\mathbb{Z}}$$

Auf $\mathbb{Z}_{p,per}^n$ definieren wir die *Faltung* $* : \mathbb{Z}_{p,per}^n \times \mathbb{Z}_{p,per}^n \to \mathbb{Z}_{p,per}^n$ und das *Produkt*: $\cdot : \mathbb{Z}_{p,per}^n \times \mathbb{Z}_{p,per}^n \to \mathbb{Z}_{p,per}^n$

$$(\mathfrak{f} * \mathfrak{g})_j = \sum_{i=0}^{n-1} \mathfrak{f}_{j-i} \mathfrak{g}_i, \qquad \forall j \in \mathbb{Z}$$

$$(\mathfrak{f} \cdot \mathfrak{g})_j = \mathfrak{f}_j \mathfrak{g}_j \qquad \forall j \in \mathbb{Z}.$$

Zeigen Sie den Faltungssatz:

$$\mathcal{F}_n(\mathfrak{f} * \mathfrak{g}) = \mathcal{F}_n(\mathfrak{f}) \cdot \mathcal{F}_n(\mathfrak{g})$$

4. **a)** Sei $a = \sum_{i=0}^{n/2-1} a_i b^i$ und $c = \sum_{i=0}^{n/2-1} c_i b^i$. Zeigen Sie: das Produkt ac läßt sich darstellen als

$$d = \sum_{i=0}^{n-1} b^i \left(\sum_{j=0}^{n-1} a_{i-j} c_j \right).$$

(Hier wird angenommen, daß $a_i = c_i = 0$ für $i \notin \{0, \ldots, n/2-1\}$.)

b) Überlegen Sie sich Bedingungen an $b = 2^m$ und $n = 2^q$, so daß die Faltungsprodukte

$$\sum_{j=0}^{n-1} a_{i-j} c_j, \qquad i = 0, \ldots, n-1 \tag{2}$$

die Bedingung

$$0 \leq \sum_{j=0}^{i} a_{i-j} c_j < p, \qquad 0 \leq i \leq n-1.$$

erfüllen. Hierbei nehmen Sie an, daß a_i, $c_i \in \{0, \ldots, b-1\}$ für $i = 0, \ldots, n-1$ gilt. Schließen Sie, daß sich die Faltungsprodukte (2) mit FFT-Techniken in $\mathbb{Z}_p$ berechnen lassen.

c) Zu gegebenem p können Sie die Multiplikation zweier Zahlen a, c bezüglich verschiedener Basen b (und damit entsprechend verschiedener Werte für n) realisieren. Wie müssen Sie b und n wählen, damit bei *festem* p möglichst große Zahlen multipliziert werden können?

5. Implementieren Sie die Multiplikation großer ganzer Zahlen mittels der oben gemachten Überlegungen. Prüfen Sie hierzu zuerst, daß Ihr Computer den Datentyp `unsigned int` mit mindestens 32 bit darstellt. Es werden bei der Multiplikation auch Zwischenergebnisse auftreten, die 64 bit benötigen. Testen Sie deshalb, ob der Datentyp `unsigned long int` bzw. `unsigned long long int` 64 bit (d.h. 8 Byte) zur Verfügung gestellt wird[1]

Unter `http://www.math.tuwien.ac.at/~melenk/teach/numerik_WS0809/projekte` finden Sie ein kleines `C++`-Programm (mit Hinweisen, wie es in ein `C`-Programm einzubinden ist), welches Dezimalzahlen aus einer Datei einliest und als Vektor von Ziffern bezüglich einer gewählten Basis b umwandelt.

Für die Realisierung benötigen Sie eine Primzahl p und die n-te Einheitswurzel ω. Verwenden Sie hierzu:

$$\begin{aligned}
n_{max} &:= 134.217.728 = 2^{27} \\
m_{max} &:= 15 \\
p &:= 15 \cdot 2^{27} + 1 = 2.013.265.921 \\
\omega_{max} &:= 440.564.289 = n_{max}\text{-te Wurzel; gegeben durch } \omega_{max} = 31^{15} \mod p \\
2^{-1} &= 1.006.632.961 \mod p
\end{aligned}$$

Sollten Sie keine 64 bit-Zahlen vom typ `long int` oder `long long int` haben, dann verwenden Sie folgende Werte:

$$\begin{aligned}
n_{max} &:= 1.024 = 2^{10} \\
m_{max} &:= 7 \\
p &:= 25 \cdot 2^{10} + 1 = 25.601 \\
\omega_{max} &:= 12.725 = n_{max}\text{-te Wurzel; gegeben durch } \omega_{max} = 3^{25} \mod p \\
2^{-1} &= 12.801 \mod p
\end{aligned}$$

[1] dies Testen Sie mit einem kleinen `C`-Programm und Befehlen von der Art `sizeof(long long int)`.

a) Schreiben Sie die Routine, die zu gegebenem $n = 2^q \leq n_{max}$ eine primitive n-te Wurzel ω von 1 bestimmt und alle weiteren Potenzen von ω, die die FFT und die IFFT benötigen.

b) Schreiben Sie Programme für die FFT, die IFFT sowie für die effiziente Faltung von Folgen auf $\mathbb{Z}^n_{p,per}$. Hierbei werde pro Folgenglieder eine Variable vom Typ `unsigned int` verwendet.

c) Schreiben Sie ein Programm, welches die Multiplikation von zwei großen ganzen Zahlen realisiert. Diese Zahlen seien in der Basis b darstellt und mögen jeweils $N/2$ Ziffern haben (N ist natürlich[2] eine Zweierpotenz).

Gehen Sie in zwei Schritten vor: Berechnen Sie zuerst die Faltungsprodukte. Anschließend wollen Sie das Produkt ac wieder in der Basis b darstellen. Schreiben Sie hierzu eine Routine, die aus der Folge $(\sum_{j=0}^{n-1} a_{i-j}b_j)_{i=0}^{n-1}$ eine neue Folge $(d_i)_{i=0}^{n-1}$ erzeugt, so daß $0 \leq d_i < b$ für alle i und

$$\sum_{i=0}^{n-1} b^i \sum_{j=0}^{n-1} a_{i-j}c_j = \sum_{i=0}^{n-1} b^i d_i.$$

6. Machen Sie Zeitmessungen für Ihr Programm mit wachsendem n (und festem b). Sehen Sie das erwartete Verhalten?

[2]das ist so in der Informatik