

Polynomarithmetik: Multiplikation

Helge Janetzko

Seminar: Computeralgebra

bei Herrn Prof. Dr. Iwanowski

Inhaltsverzeichnis

Abbildungsverzeichnis	II
Formelverzeichnis	II
1 Polynome im Überblick	1
2 Multiplikation: Der Karatsuba-Algorithmus	2
2.1 Karatsuba-Algorithmus für Zahlen	2
2.2 Karatsuba-Algorithmus für Polynome	4
3 Schnelle Multiplikation von Polynomen mit FFT	5
3.1 Stützstellendarstellung von Polynomen	5
3.2 Einheitswurzel.....	7
3.3 Multiplikation durch Anwendung der Fouriertransformation.....	8
3.3.1 Diskrete Fouriertransformation (DFT).....	8
3.3.2 Schnelle Fouriertransformation (FFT)	9
3.3.3 Multiplikation der Polynome.....	11
Literaturverzeichnis	13

Abbildungsverzeichnis

Abbildung 3-1 Potenzen der komplexen primitiven Einheitswurzel	7
Abbildung 3-2 FFT-Algorithmus für ein Polynom f mit $\deg(f) = 7$	10

Formelverzeichnis

Formel 1-1 Standarddarstellung eines Polynoms	1
Formel 1-2 Berechnung des Summenpolynoms	1
Formel 1-3 Formel des Cauchyproduktes	2
Formel 2-1 Multiplikation gemäß des Karatsuba-Algorithmus	3
Formel 3-1 Vektorschreibweise zur Auswertung eines Polynoms an einer Stelle	6
Formel 3-2 Transformation in Stützstellendarstellung	6
Formel 3-3 Komplexe primitive n -te Einheitswurzel	7
Formel 3-4 Eigenschaft 1 der komplexen Einheitswurzel	8
Formel 3-5 Eigenschaft 2 der komplexen Einheitswurzel	8
Formel 3-6 Fouriermatrix für $n = 4$	8
Formel 3-7 Zerlegung des Polynoms gemäß der FFT	9
Formel 3-8 Inverse Fouriermatrix für $n = 4$	11

1 Polynome im Überblick

In der Mathematik ist ein Polynom eine Summe von Vielfachen von Potenzen einer Variablen x . Die Faktoren vor den Potenzen der Variablen x werden als Koeffizienten bezeichnet. Daher spricht man auch von der Koeffizientendarstellung des Polynoms. Liegt das Polynom in der Form

$$a(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 = \sum_{k=0}^n a_k x^k$$

Formel 1-1 Standarddarstellung eines Polynoms

vor, befindet es sich in einer ausmultiplizierten Koeffizientendarstellung. Ist $a_n \neq 0$, ist n der Grad des Polynoms und a_n der führende Koeffizient. Man bezeichnet diese Darstellung auch als Standarddarstellung des Polynoms $a(x)$.

Werden zwei Polynome addiert, ist der Grad des Summenpolynoms kleiner oder gleich dem Maximum des Grades der beiden Summandenpolynome: $\deg(a + b) \leq \max(\deg(a), \deg(b))$ ¹. Es kann dementsprechend kein höherer Grad in dem Summenpolynom vorkommen, als in einem der beiden Summandenpolynomen vorhanden ist. Sollten beide Summandenpolynomen jedoch den gleichen Grad besitzen und der führende Koeffizient des einen Polynoms sich nur durch das Vorzeichen von dem führenden Koeffizient des anderen Polynoms unterscheidet, so würde die Addition beider Koeffizienten Null ergeben. Der Grad des Summenpolynoms hätte sich also verringert.

$$\sum_{k=0}^n a_k x^k + \sum_{k=0}^m b_k x^k = \sum_{k=0}^{\max(m,n)} (a_k + b_k) x^k \quad a_k = 0 \text{ falls } k > n \quad \text{und} \quad b_k = 0 \text{ falls } k > m$$

Formel 1-2 Berechnung des Summenpolynoms

Bei der Multiplikation ergibt sich der Grad des resultierenden Produktpolynoms jedoch aus der Summe der Grade der Faktorpolynome: $\deg(a \cdot b) = \deg(a) + \deg(b)$. Das hat zur Folge, dass bei der Berechnung sehr hohe Potenzen der Variablen x auftreten können.

$$\sum_{k=0}^n a_k x^k \cdot \sum_{k=0}^m b_k x^k = \sum_{k=0}^{m+n} c_k x^k$$

Die Koeffizienten c_k des Produktpolynoms lassen sich nicht so schnell und einfach berechnen wie die Koeffizienten des Summenpolynoms. Eine Möglichkeit der Berechnung dieser Koeffizienten liefert die Formel des Cauchyproduktes:

¹ Englisch: degree = Grad

$$\sum_{k=0}^n a_k x^k \cdot \sum_{k=0}^m b_k x^k = \sum_{k=0}^{m+n} c_k x^k \quad \text{mit} \quad c_k = \sum_{j=0}^k a_j b_{k-j}$$

und $a_j = 0$ für $j > n$

und $b_j = 0$ für $j > m$

Formel 1-3 Formel des Cauchyproduktes

Grundsätzlich sind alle Additions- und Multiplikationsalgorithmen von Zahlen auch auf Polynome anwendbar. Der einzige Unterschied besteht im Weglassen der Überträge, wie in folgendem Beispiel ersichtlich wird:

Addition von Zahlen:

$$\begin{array}{r} 7 \quad 5 \\ + \quad 1_1 \quad 8 \\ \hline = \quad 9 \quad 3 \end{array}$$

Addition von Polynomen:

$$\begin{array}{r} 7x^1 \quad 5x^0 \\ + \quad 1x^1 \quad 8x^0 \\ \hline = \quad 8x^1 \quad 13x^0 \end{array}$$

Der Vorteil dabei ist, dass die Implementierung von Algorithmen für Polynome einfacher wird als bei Zahlen. Als Nachteil ist jedoch aufzuführen, dass die Koeffizienten beliebig groß werden können, was sich negativ auf die Berechnungszeit auswirken kann.

2 Multiplikation: Der Karatsuba-Algorithmus

Der Karatsuba-Algorithmus² ist ein effizienter Algorithmus zur Multiplikation. Er kann, wie im folgenden beschrieben, sowohl für Zahlen als auch für Polynome verwendet werden. Der Algorithmus arbeitet nach dem Divide-and-Conquer-Prinzip³, indem die Faktoren immer wieder zerlegt werden, so dass man schlussendlich nur noch sehr kleine Ausdrücke multiplizieren muss. Im Gegensatz zur Schulmultiplikation, die eine Komplexität von $O(n^2)$ besitzt, schafft dieser Algorithmus für Zahlen und Polynome gleichermaßen die Multiplikation mit einer Komplexität von nur $O(n^{\log_2 3})$.

2.1 Karatsuba-Algorithmus für Zahlen

Sei n die Länge der beiden Faktoren $x \in \mathbb{Z}$ und $y \in \mathbb{Z}$, die multipliziert werden sollen, und sei n eine Zweierpotenz. Ist einer der beiden Faktoren kleiner als der andere oder keine Zweierpotenz, so wird dieser mit führenden Nullen aufgefüllt, bis beide Faktoren die gleiche Anzahl an Dezimalstellen haben und die Länge einer Zweierpotenz entspricht. Dadurch ändern sich die Werte der Faktoren nicht. Es kann folglich jede Zahl als Zahl mit n Stellen dargestellt werden. Nun werden x und y jeweils in zwei Teile gleicher Länge zerlegt.

² Der Karatsuba-Algorithmus wurde 1960 von Anatolii Alexeevich Karatsuba (*1937) entwickelt.

³ Englisch: divide and conquer = teile und herrsche

$$x = a \cdot 10^{n/2} + b$$

$$y = c \cdot 10^{n/2} + d$$

Daraus ergibt sich folgendes Produkt:

$$\begin{aligned} x \cdot y &= (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d) \\ &= a \cdot c \cdot 10^n + (a \cdot d + b \cdot c) \cdot 10^{n/2} + b \cdot d \end{aligned}$$

Diese Zerlegung wird fortgesetzt, bis die Faktoren nur noch die Länge $n = 1$ besitzen. In diesem Ausdruck sind jedoch vier Produkte vorhanden, wodurch man die gleiche Komplexität wie bei der Schulmultiplikation erhalten würde. Um nun aber die genannte Effizienz zu erreichen, muss die Identität ausgenutzt werden, indem man folgenden Teilausdruck umschreibt:

$$a \cdot d + b \cdot c = a \cdot c + b \cdot d + (a - b) \cdot (d - c)$$

Auf den ersten Blick wirkt der neue Teilausdruck umständlicher, da statt zwei nun drei Multiplikationen vorhanden sind. Setzt man diesen neuen Teilausdruck nun wieder in die gesamte Formel ein, so wird schnell ersichtlich, dass zwar mehr Produkte als vorher vorhanden sind, es sich aber nur um drei voneinander verschiedene handelt. Letztlich müssen also nur noch drei Produkte berechnet werden.

$$\begin{aligned} x \cdot y &= a \cdot c \cdot 10^n + (a \cdot c + b \cdot d + (a - b) \cdot (d - c)) \cdot 10^{n/2} + b \cdot d \\ &\Rightarrow p1 = a \cdot c; \quad p2 = b \cdot d; \quad p3 = (a - b) \cdot (d - c) \\ &\Rightarrow x \cdot y = p1 \cdot 10^n + (p1 + p2 + p3) \cdot 10^{n/2} + p2 \end{aligned}$$

Formel 2-1 Multiplikation gemäß des Karatsuba-Algorithmus

Unter Verwendung dieser Formel kann folgender rekursiver Algorithmus aufgestellt werden:

```
long multiply( long x, long y ) {
    long a, b, c, d, n, p1, p2, p3;
    int n;

    // Maximum der Anzahl der Stellen
    n = max. Stellen von x und y, auf nächste 2er-Potenz erhöht;

    // wenn mehr als eine Stelle, Karatsuba
    if( n > 1 ) {

        // Zerlegung von x und y
        b = x mod 10^(n/2);
        a = (x - b) / 10^(n/2);
        d = y mod 10^(n/2);
        c = ( y - d ) / 10^(n/2);

        // die drei Produkte durch rekursive Aufrufe
        p1 = multiply( a, c );
        p2 = multiply( b, d );
        p3 = multiply( a - b, d - c );
    }
}
```

```

    // Resultat berechnen
    return p1 * 10^n + ( p1 + p2 + p3 ) * 10^(n/2) + p2;
}

return x * y;
}

```

2.2 Karatsuba-Algorithmus für Polynome

Der Karatsuba-Algorithmus für Polynome arbeitet nach genau dem gleichen Prinzip, wie der für Zahlen. Die Polynome werden hier jedoch als Koeffizientenlisten dargestellt. Bei der Zerlegung der Polynome wird die Liste also in die linke und in die rechte Hälfte aufgeteilt. Die Zerlegung wird rekursiv fortgesetzt, bis die Listen nur noch ein Element enthalten. Der führende Koeffizient steht in der hier dargestellten Implementierung am Listenanfang. Daher werden, falls ein Polynom weniger Koeffizienten hat als das andere oder die Anzahl an Koeffizienten keiner Zweierpotenz entspricht, die zusätzlichen Nullen an den Listenanfang gehängt. Eine Multiplikation mit 10^n entspricht somit dem Anhängen von n Nullen an das Ende der Koeffizientenliste. Des Weiteren werden alle Rechenoperationen ohne Überträge ausgeführt, d.h. bei Addition und Subtraktion werden die einzelnen Listenelemente jeweils unabhängig voneinander berechnet. Damit ergibt sich der folgende rekursive Algorithmus:

```

list multiply ( list x, list y ) {
    list a, b, c, d, p1, p2, p3, midterm, tab, result;
    int n;

    n = max. Anzahl Koeffizienten von x und
        y, auf nächste 2er-Potenz erhöht;

    if( n > 1 ) {

        // Listen mit Nullen bis n auffüllen
        while( x.size() < n ) x.addFirst( 0 );
        while( y.size() < n ) y.addFirst( 0 );

        // Zerlegung der Koeffizienten-Listen
        a = subList( x, 0, n/2 - 1 );
        b = subList( x, n/2, n );
        c = subList( y, 0, n/2 - 1 );
        d = subList( y, n/2, n );

        // die drei Produkte
        p1 = multiply( a, c );
        p2 = multiply( b, d );
        p3 = multiply( subtract(a, b), subtract(d, c) );

        //---- Resultat berechnen ----
        // Liste der Länge n/2, nur mit Nullen
    }
}

```

```

tab = new list();
for( int i = 0; i < n/2; ++i )
    tab.addFirst( 0 );

// p1 + p2 + p3
midterm = plus( p1, p2, p3 );

// p1 * 10^n
p1.append( tab );
p1.append( tab );

// midterm * 10^(n/2)
midterm.append( tab );

return plus( p1, midterm, p2 );
//-----
}

result = new list();
result.addFirst( x.getFirst() * y.getFirst() );
return result;
}

```

3 Schnelle Multiplikation von Polynomen mit FFT

Die Multiplikation von Polynomen mit der schnellen Fouriertransformation (FFT)⁴ ist signifikant schneller als der Karatsuba-Algorithmus. Mit diesem Verfahren kann die Multiplikation mit einer Komplexität von nur $O(n \log_2 n)$ durchgeführt werden. Im Folgenden werden vorerst einige zum Verständnis des Algorithmus notwendige Grundlagen beschrieben, bevor darauf aufbauend die Multiplikation mit diesem Verfahren erläutert wird.

3.1 Stützstellendarstellung von Polynomen

Bisher wurden die Polynome in der Koeffizientendarstellung vorgestellt. Eine weitere Darstellung von Polynomen ist die Stützstellendarstellung. Ein Polynom a mit $\deg(a) = n - 1$ wird eindeutig durch n Werte y_0, \dots, y_{n-1} an n verschiedenen vorgegebenen Stellen x_0, \dots, x_{n-1} dargestellt. Die Stellen x_0, \dots, x_{n-1} werden Stützstellen genannt. Ist $\deg(a) = 1$, so handelt es sich um eine Gerade. Um diese eindeutig darzustellen, werden also zwei Punkte bzw. zwei Stützstellen mit den zugehörigen Werten benötigt. Ist $\deg(a) = 2$, handelt es sich um eine Parabel. Hier werden drei Punkte zur eindeutigen Darstellung gebraucht. Zur eindeutigen Darstellung wird folglich pro Koeffizient eine Stützstelle benötigt.

⁴ Englisch: Fast Fourier Transform = schnelle Fouriertransformation

Der FFT-Algorithmus wurde 1965 von James Cooley und John W. Tukey veröffentlicht.

Soll ein Polynom an der Stelle x_0 ausgewertet werden, also der zugehörige Wert für die Stützstelle x_0 berechnet werden, lässt sich dies als Vektormultiplikation aufschreiben. Die Koeffizienten a_0, \dots, a_{n-1} bilden einen Zeilenvektor und die Potenzen der Stützstelle x_0^0, \dots, x_0^{n-1} einen Spaltenvektor.

$$y_0 = (a_0 \quad \dots \quad a_{n-1}) \cdot \begin{pmatrix} x_0^0 \\ \vdots \\ x_0^{n-1} \end{pmatrix}$$

Formel 3-1 Vektorschreibweise zur Auswertung eines Polynoms an einer Stelle

Soll das Polynom an n Stellen ausgewertet werden, ergibt sich eine Vektor-Matrix-Multiplikation. Jede Spalte der Matrix T enthält die Potenzen jeweils einer Stützstelle. Das Ergebnis der Multiplikation ist ein Zeilenvektor, welcher die zu den Stützstellen zugehörigen Werte y_0, \dots, y_{n-1} enthält.

$$(y_0 \quad \dots \quad y_{n-1}) = (a_0 \quad \dots \quad a_{n-1}) \cdot \begin{pmatrix} x_0^0 & \dots & x_{n-1}^0 \\ \vdots & & \vdots \\ x_0^{n-1} & \dots & x_{n-1}^{n-1} \end{pmatrix}$$

$$(y_0 \quad \dots \quad y_{n-1}) = (a_0 \quad \dots \quad a_{n-1}) \cdot T$$

Formel 3-2 Transformation in Stützstellendarstellung

Die Matrix T wird Transformationsmatrix genannt und dient zur Transformation eines Polynoms aus der Koeffizientendarstellung in die Stützstellendarstellung. Soll ein Polynom aus der Stützstellendarstellung in die Koeffizientendarstellung zurück transformiert werden, so wird die inverse Transformationsmatrix T^{-1} benötigt.

$$(y_0 \quad \dots \quad y_{n-1}) \cdot T^{-1} = (a_0 \quad \dots \quad a_{n-1})$$

Der Vorteil der Stützstellendarstellung ist, dass die Multiplikation von zwei Polynomen in dieser Darstellung eine Komplexität von $O(n)$ aufweist. Voraussetzung ist, dass bei beiden Faktorpolynomen die gleichen Stützstellen verwendet werden. Außerdem muss die Anzahl der Stützstellen sich am Grad des Produktpolynoms orientieren, da bei einer Multiplikation zweier Polynome eine Addition der Grade stattfindet und somit das Produktpolynom einen höheren Grad aufweist, als die Faktorpolynome. Das hat zur Folge, dass zur eindeutigen Darstellung des Produktpolynoms auch mehr Stützstellen benötigt werden.

Die Multiplikation kann so zwar effizient ausgeführt werden, die Transformation zwischen den verschiedenen Darstellungsformen weist jedoch eine Komplexität von $O(n^2)$ auf. Das ist darauf zurückzuführen, dass die Transformationsmatrix eine $n \times n$ -Matrix ist. Im folgenden wird

beschrieben, wie die Transformation beschleunigt werden kann, indem Potenzen der komplexen primitiven n -ten Einheitswurzel als Stützstellen verwendet werden.

3.2 Einheitswurzel

Sei R ein kommutativer Ring mit Einselement 1 und $n \geq 1$ eine natürliche Zahl. $\zeta \in R$ heißt n -te Einheitswurzel, wenn

$$\zeta^n = 1$$

und primitive n -te Einheitswurzel, wenn zusätzlich folgende Eigenschaft gilt:

$$\zeta^k \neq 1 \quad \text{für} \quad 0 < k < n$$

Bei den Reellen Zahlen ist keine primitive n -te Einheitswurzel für ein beliebiges n vorhanden. Anders ist das bei den komplexen Zahlen. Für jedes $n \geq 1$ gibt es auch die primitive n -te Einheitswurzel.

$$\zeta = e^{\frac{2\pi i}{n}}, \quad \text{denn} \quad \zeta^n = e^{\frac{2\pi i n}{n}} = e^{2\pi i} = \cos(2\pi) + i \cdot \sin(2\pi) = 1 + i \cdot 0 = 1$$

Formel 3-3 Komplexe primitive n -te Einheitswurzel

Die Punkte ζ^k für $0 < k \leq n$ bilden ein regelmäßiges n -Eck auf dem Einheitskreis der Gaußschen Zahlenebene.

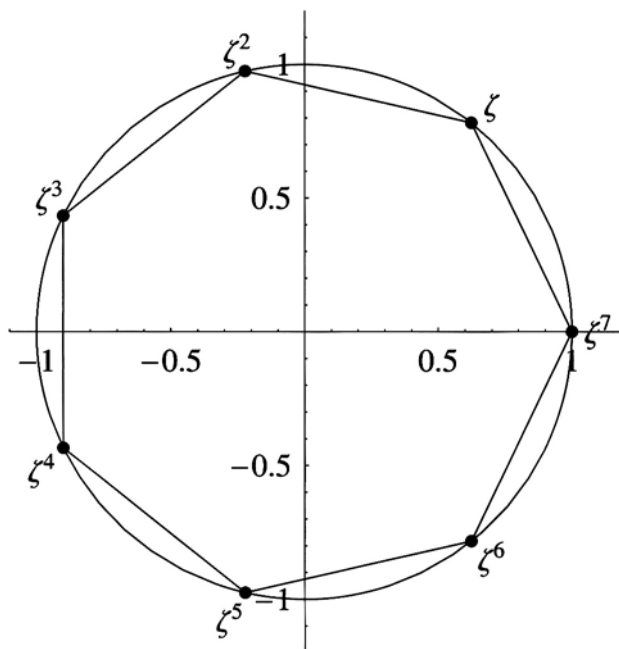


Abbildung 3-1 Potenzen der komplexen primitiven Einheitswurzel

An Abbildung 3-1 lässt sich leicht die Symmetrie an den Potenzen der komplexen primitiven n -ten Einheitswurzel erkennen. Aufgrund dieser Symmetrie sind ein paar besondere Eigenschaften

vorhanden. Zwei dieser Eigenschaften sollen aufgeführt werden, da sie für das Verständnis der FFT essenziell sind.

Wenn n gerade ist, so gilt folgende Eigenschaft:

$$\zeta^{\frac{n}{2}+k} = -\zeta^k$$

Formel 3-4 Eigenschaft 1 der komplexen Einheitswurzel

Die Eigenschaft in Formel 3-4 lässt sich mit Abbildung 3-1 leicht veranschaulichen. Erhöht man den Exponenten von ζ , so bewegt man sich gegen den Uhrzeigersinn auf dem Einheitskreis der Gaußschen Zahlenebene. Ist der Exponent $n/2$, so ist das Ergebnis -1 . Man kann also alle Potenzen von ζ erhalten, indem man den Exponenten nur bis $n/2$ erhöht und dafür das Ergebnis sowohl positiv als auch negativ (mit -1 multipliziert) verwendet.

Die zweite wesentliche Eigenschaft ist, dass das Quadrat der komplexen primitive n -ten Einheitswurzel die komplexe primitive $n/2$ -te Einheitswurzel ist.

$$\left(e^{\frac{2\pi i}{n}} \right)^2 = e^{\frac{2\pi i \cdot 2}{n}} = e^{\frac{2\pi i \cdot 2}{2 \cdot n/2}} = e^{\frac{2\pi i}{n/2}}$$

Formel 3-5 Eigenschaft 2 der komplexen Einheitswurzel

3.3 Multiplikation durch Anwendung der Fouriertransformation

3.3.1 Diskrete Fouriertransformation (DFT)

Von der diskreten Fouriertransformation (DFT) spricht man bei der Transformation des Polynoms a mit $\deg(a) = n - 1$ in Stützstellendarstellung mit

$$\zeta^k \quad \text{für} \quad 0 \leq k < n \quad \text{mit} \quad \zeta = e^{\frac{2\pi i}{n}}$$

als Stützstellen. Es werden also Potenzen der komplexen primitive n -ten Einheitswurzel als Stützstellen verwendet. Die Stützstellen liegen daher in äquidistanten Abständen auf dem Einheitskreis der Gaußschen Zahlenebene. Die daraus resultierende symmetrische Transformationsmatrix wird Fouriermatrix F genannt.

$$\begin{pmatrix} x_0^0 & \dots & x_{n-1}^0 \\ \vdots & & \\ x_0^{n-1} & \dots & x_{n-1}^{n-1} \end{pmatrix} \xrightarrow{n=4} \begin{pmatrix} \zeta^{0 \cdot 0} & \zeta^{1 \cdot 0} & \zeta^{2 \cdot 0} & \zeta^{3 \cdot 0} \\ \zeta^{0 \cdot 1} & \zeta^{1 \cdot 1} & \zeta^{2 \cdot 1} & \zeta^{3 \cdot 1} \\ \zeta^{0 \cdot 2} & \zeta^{1 \cdot 2} & \zeta^{2 \cdot 2} & \zeta^{3 \cdot 2} \\ \zeta^{0 \cdot 3} & \zeta^{1 \cdot 3} & \zeta^{2 \cdot 3} & \zeta^{3 \cdot 3} \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

Formel 3-6 Fouriermatrix für $n = 4$

Daraus ergibt sich, dass ein Element von F an der Stelle ij die Potenz von ζ ist, dessen Exponent das Produkt von i und j ist.

$$F_{ij} = \zeta^{i \cdot j}$$

3.3.2 Schnelle Fouriertransformation (FFT)

Die FFT ist ein effizienter Algorithmus für die DFT und arbeitet wie der Karatsuba-Algorithmus nach dem Divide-and-Conquer-Prinzip. Die Zerlegung des Polynoms ist jedoch eine andere. Das zu transformierende Polynom wird in ein Polynom zerlegt, welches die Koeffizienten mit geradem Koeffizientenindex enthält und ein Polynom, welches die Koeffizienten mit ungeradem Koeffizientenindex enthält. Gemäß folgender Formel können die beiden Polynome wieder zusammengesetzt werden.

$$\begin{aligned} a(x) &= a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} \\ a^{[1]}(x) &= a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1} \\ a^{[2]}(x) &= a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1} \\ \Rightarrow a(x) &= a^{[1]}(x^2) + x \cdot a^{[2]}(x^2) \end{aligned}$$

Formel 3-7 Zerlegung des Polynoms gemäß der FFT

Die Zerlegung wird solange rekursiv fortgesetzt, bis die zerlegten Polynome nur noch aus einem Koeffizienten bestehen. Als Vorbedingung für diese Zerlegung lässt sich somit feststellen, dass die Anzahl der Koeffizienten eine Zweierpotenz sein muss, was durch zufügen von Nullen bei jedem Polynom möglich ist. Folgender Algorithmus ergibt sich.

```
// Führender Koeffizient am Listenende !
// Voraussetzung: Koeffizientenanzahl ist 2er-Potenz
list fft ( list l ) {
    complex zeta, c1, c2;
    list
        evenList = new list(), oddList = new list(),
        a, b, tab1 = new list(), tab2 = new list();

    n = Grad von l auf nächste 2er-Potenz erhöht;

    if( n > 1 ) {

        // primitive n-te Einheitswurzel
        zeta = e^( 2*π*i / n );

        // Zerlegung der Koeffizientenliste
        for( int j = 0; j < n-1; j+=2 )
            evenList.addLast( l.get( j ) );
        for( int j = 1; j < n ; j+=2 )
            oddList.addLast( l.get( j ) );
    }
}
```

```

a = fft( evenList );
b = fft( oddList );

// Werte an den Stellen zeta^k und zeta^(k+n/2) bestimmen
for( int k = 0; k < n/2; ++k )
{
    c1 = a.get( k ) + zeta^k * b.get( k );
    c2 = a.get( k ) + zeta^k * -b.get( k );

    tab1.addLast( c1 );
    tab2.addLast( c2 );
}

// Liste tab2 an Liste tab1 hängen
tab1.addAll( tab2 );
return tab1;
}

return l;
}

```

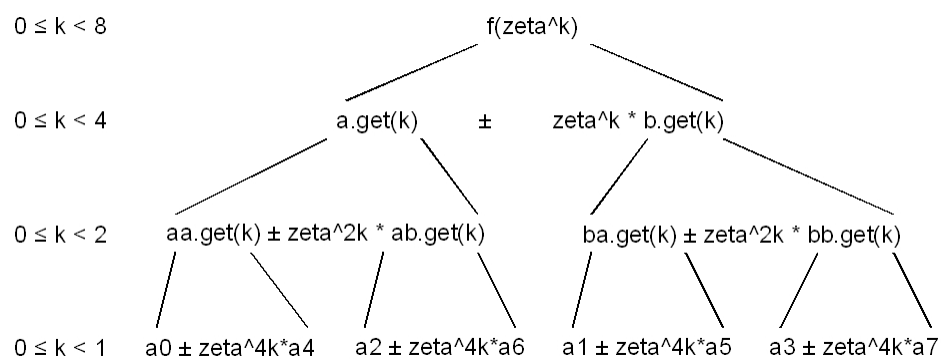


Abbildung 3-2 FFT-Algorithmus für ein Polynom f mit $\deg(f) = 7$

In Abbildung 3-2 wird deutlich, wie der FFT-Algorithmus das Polynom gemäß Formel 3-7 zerlegt und die Resultate berechnet. Das Polynom f wird in ein Polynom a mit den Koeffizienten mit geradem Koeffizientenindex und in ein Polynom b mit den Koeffizienten mit ungeradem Koeffizientenindex zerlegt. In jeder Rekursionsstufe wird in Abbildung 3-2 die Stützstelle quadriert. In dem FFT-Algorithmus geschieht dies implizit durch Ausnutzung von Formel 3-5, da sich die Anzahl der Koeffizienten bei jedem rekursiven Aufruf halbiert. Das Teilpolynom mit den Koeffizienten mit ungeradem Koeffizientenindex kann durch Verwendung von Formel 3-4 jeweils zweimal verwendet werden, indem es einmal mit positivem und einmal mit negativem Vorzeichen verwendet wird.

Zur Berechnung der Komplexität der FFT muss die Komplexität für eine Rekursionsstufe und die Anzahl der Rekursionsstufen betrachtet werden. Pro Rekursionsstufe gibt es eine Komplexität von $O(n)$. Die Anzahl der Rekursionsstufen ergibt sich aus der Häufigkeit der Zerlegung des Polynoms. O.B.d.A. sei $n = 2^p$ die Anzahl der Koeffizienten des Polynoms. Da

sich bei jeder Zerlegung die Anzahl der Koeffizienten halbiert, ist $p = \log_2 n$ die Anzahl der Rekursionsstufen. Daraus ergibt sich eine gesamte Komplexität für den Algorithmus von $O(n \cdot p) = O(n \log_2 n)$.

Soll das Polynom von der Stützstellendarstellung in die Koeffizientendarstellung zurück transformiert werden, so kann der FFT-Algorithmus auf die gleiche Weise verwendet werden. Der einzige Unterschied ist, dass statt der Fouriermatrix F die inverse Fouriermatrix F^{-1} verwendet werden muss. Hier muss anstelle der primitiven n -ten Einheitswurzel jedoch nur die inverse primitive n -te Einheitswurzel verwendet werden, welche bei den komplexen Zahlen die konjugiert komplexe primitive n -te Einheitswurzel ist. Außerdem müssen die Elemente durch n dividiert werden.

$$F^{-1} \stackrel{n=4}{=} \frac{1}{4} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}, \quad F^{-1}_{ij} = \frac{\zeta^{-i \cdot j}}{n}$$

Formel 3-8 Inverse Fouriermatrix für $n = 4$

Konkret bedeutet das für den Algorithmus, dass die Exponenten der primitiven Einheitswurzel mit -1 multipliziert werden müssen und das Ergebnis am Ende durch n dividiert werden muss. Alles andere bleibt identisch.

3.3.3 Multiplikation der Polynome

Sollen nun zwei Polynome durch Anwendung der FFT multipliziert werden, müssen beide als erstes durch Anhängen von Nullen so viele Koeffizienten erhalten, wie das Produktpolynom haben wird. Anschließend wird auf beide Faktorpolynome die FFT angewendet. Die nun in der Stützstellendarstellung vorliegenden Polynome werden multipliziert, was nur einer einfachen Multiplikation der zusammengehörigen Werte entspricht. Auf das resultierende Polynom wird die FFT invers angewendet. Man erhält das Produktpolynom in der Koeffizientendarstellung. Da die aufwändigsten Rechenschritte eine Komplexität von $O(n \log_2 n)$ aufweisen, beträgt die gesamte Komplexität der Multiplikation zweier Polynome durch Anwendung der FFT auch nur $O(n \log_2 n)$. Folgender Algorithmus zur Multiplikation lässt sich aufstellen.

```
// Führender Koeffizient am Listenende !
list multiply( list l1, list l2 ) {
    list result = new list();

    // Grad der Multiplikation ist n+m
    int m = l1.size();
```

```

int n    = l2.size();
int num = (n + m) auf nächste 2er-Potenz erhöht;
while( l1.size() < num ) l1.addLast( 0 );
while( l2.size() < num ) l2.addLast( 0 );

// Fouriertransformation, O(n·logn)
l1 = fft( l1 );
l2 = fft( l2 );

// Multiplikation der Werte, O(n)
for( int i = 0; i < l1.size(); ++i )
    result.addLast( l1.get( i ) * l2.get( i ) );

// Inverse Fouriertransformation, O(n·logn)
result = ifft( result );

// Koeffizienten durch n dividieren
for( int i = 0; i < result.size(); ++i )
    result.set( i, result.get( i ) / n );

return result;
}

```

Literaturverzeichnis

- [1] Koepf, Wolfram:
Computeralgebra – Eine algorithmisch orientierte Einführung. Heidelberg, 2006

- [2] Lang, H.W.:
FH Flensburg, Internet <http://www.inf.fh-flensburg.de/lang/algorithmen/fft/index.htm>

- [3] Mathematik Online, Projekt der Universitäten Stuttgart und Ulm, 2006,
Internet <http://mo.mathematik.uni-stuttgart.de/kurse/kurs19/kapitel2.html>