Duale Hochschule Baden-Württemberg Mannheim

Studienarbeit

Arithmetik endlicher Körper (speziell in der Charakteristik 2)

Studiengang Informatik

Studienrichtung Angewandte Informatik

Verfasser(in):	Lars Krickl
Matrikelnummer:	2512317
Kurs:	TINF22 AI2
Studiengangsleiter:	Prof. Dr. Holger D. Hofmann
Wissenschaftliche(r) Betreuer(in):	Prof. Dr. Reinhold Hübl
Bearbeitungszeitraum:	15.10.2024 - 15.04.2025

Inhaltsverzeichnis

1 Theorie

x

2 Implementierung

2.1 Polynomdarstellung

Um in der Programmiersprache Python Polynome darzustellen, wird die Datenstruktur Tupel verwendet. In einem Tupel können mehrere Inhalte unveränderlich gespeichert werden. Da in der Charakteristik zwei lediglich Zahlen aus dem Binärsystem Vorfaktoren der verschiedenen Monome sein können, werden die Tupel nur mit Nullen und Einsen befüllt. Dadurch wird bestimmt, ob ein bestimmtes Monom in dem Polynom vorkommt (in diesem Fall mit einer 1 an der entsprechenden Stelle dargestellt) oder ob das Monom nicht in dem Polynom vertreten ist (durch eine 0 dargestellt).

In diesen Tupeln steht das Monom mit dem höchsten Exponenten an der ersten Stelle (entspricht der Stelle ganz links) und die Exponenten nehmen dann schrittweise nach rechts ab, sodass die letzte Stelle (ganz rechts) dem Monom mit dem Exponenten 0 entspricht. Diese Darstellung durch Tupel vereinfacht den Umgang mit Polynomen, da die ausführliche mathematische Schreibweise zusätzliche, unnötige Informationen wie die Variable X oder das $"+"$-Zeichen enthält. Für den Umgang mit Polynomen ist jedoch nur wichtig, welcher Vorfaktor an welcher Stelle des Polynoms steht, und diese beiden Informationen sind in der Schreibweise als binäre Tupel enthalten, wodurch sie sich anbietet für das Arbeiten mit Polynomen.

Um solch ein binäres Tupel in Python wieder in die mathematische Schreibweise eines Polynoms mit Vorfaktor, Variable X und Exponent für jedes Monom darzustellen, wurde die Funktion *"TupleToPolynom"* erstellt. Wie der Name schon sagt, bekommt die Funktion ein Tupel als Funktionsparameter und gibt einen String zurück, der das vollständige Polynom umfasst. Dabei iteriert die Funktion über die einzelnen Elemente des Tupels und bestimmt daraus einen Teilstring. Bei der Erstellung dieser Teilstrings gibt es drei verschiedene Fälle. Bei dem letzten Monom eines Tupels, also dem Monom ganz rechts, wird der Teilstring lediglich eine 1, da das letzte Monom den Exponenten null hat, und alles hoch null gleich eins ergibt. Bei dem vorletzten Monom wird der Teilstring zu einem X, da dieses Monom den Exponenten eins hat. Für alle anderen Fälle, bei denen der Exponent größer als eins ist, wird der Teilstring zu X^n, wobei n dem Exponenten des Monoms entspricht. Im Zuge dieser Funktion wird das Tupel vor der Erstellung des Lösungsstrings einmal umgekehrt, sodass nun das letzte Monom an erster Stelle steht, das vorletzte an zweiter Stelle steht, und so weiter. Dadurch gewinnt man den Vorteil, dass jetzt die Position des Monoms in dem Tupel gleich dem Exponenten des Monoms entspricht. Da bei Tupeln, ähnlich

wie bei Arrays, das Zählen bei null anfängt, steht nach der Umkehrung das erste Monom (was ursprünglich das Letzte war) an der Position null in dem Tupel, was genau dem Exponenten dieses Monoms entspricht. All diese einzelnen Teilstrings werden dann aneinandergereiht, wobei zwischendrin immer ein $"+"$ beigefügt wird. So erhält man schließlich einen String, der das vollständige Polynom, wie man es auch in der mathematischen Schreibweise aufschreiben würde.

Diese Funktion, die aus einem Tupel wieder ein Polynom macht, erfüllt keinen direkten Zweck, der für weitere Operationen wichtig ist, sondern diese Funktion dient lediglich dazu, aus einem Tupel eine für den Anwender lesbare Darstellung zu bieten, da die Tupel, vor allem bei steigender Länge, unübersichtlich werden können.

2.2 Hilfsfunktionen

Im Laufe der Implementierung werden für einige Operationen Hilfsfunktionen benötigt. Diese sind in der Datei *ÄuxiliaryFunctions.ipynb"* gesammelt, da sie später von verschiedenen Stellen aus benötigt werden. Im Folgenden werden diese Funktionen genauer erläutert.

2.2.1 Wegschneiden von Nullen

Um führende Nullen von Polynomen, bzw. von binären Tupeln wegzuschneiden, wurde die Funktion *"cut_zeros_left"* implementiert, die dafür sorgt, dass das gegebene Tupel mit der ersten Eins, die in dem Tupel vorhanden ist, beginnt und alle vorherigen Stellen abgeschnitten werden. Zudem gibt es noch die Funktion *"cut_zeros_right"*, die die gleiche Funktionalität beschreibt, allerdings von der anderen Seite, d.h. diese Funktion schneidet führende Nullen von rechts weg.

Um diese führenden Nullen abzuschneiden, wird über das binäre Tupel iteriert und überprüft, ob die erste Stelle (bei der Funktion *"cut_zeros_left"*), bzw. die letzte Stelle (bei der Funktion *"cut_zeros_right"*) eine Null ist. Sollte diese Bedingung wahr sein, wird die erste, bzw. letzte Stelle abgeschnitten. Dieser Vorgang wird so lange wiederholt, bis die erste, bzw. letzte Stelle eine Eins ist.

Für den Sonderfall, dass ein leeres Tupel in die Funktion gegeben wird, muss eine zusätzliche Bedingung eingeführt werden, da in einem leeren Tupel keine Eins gefunden werden kann, und somit eine Endlosschleife entstehen würde. Dieser Fall

kann einfach abgefangen werden, indem in der Schleifenbedingung geprüft wird, ob das Tupel leer ist, und die Schleife nur durchlaufen wird, sollte das Tupel nicht leer sein.

2.2.2 Erste Einser-Stelle finden

Die Funktion *"find_pos_of_first_one"* dient dazu, die erste Stelle eines Polynoms, von links aus, zu finden, die mit einer Eins befüllt ist. Diese Funktion wird im späteren Verlauf benötigt, um den Grad einer Funktion ermitteln zu können.

Um diese Stelle bestimmen zu können, wird über die Stellen des gegebenen Tupels iteriert. Sobald eine Eins gefunden wurde, wird der Index zurückgegeben, bei dem sich die Iteration zu dem Zeitpunkt befindet. Dieser Index stellt dann die Position der ersten Eins des Polynoms dar.

Sollte keine Eins gefunden werden, wird von der Funktion "Noneßurückgegeben. Dieses Ergebnis bedeutet dann, dass das Tupel leer sein muss, da es lediglich aus Nullen besteht.

2.2.3 Grad eines Polynoms bestimmen

Der Grad eines Polynoms wird bestimmt durch den höchsten Exponenten, der in dem jeweiligen Polynom zu finden ist. Um nun, mittels der Funktion "find_degree", den Grad eines Polynoms bestimmen zu können, muss zunächst die erste Stelle des Polynoms ermittelt werden, die eine Eins enthält. Dies ist durch die eben aufgeführte Hilfsfunktion *"find_pos_of_first_one"* möglich.

Anhand dieser Stelle, welche das Monom mit dem höchsten Exponent darstellt, kann nun der Grad des Polynoms ermittelt werden. Dazu muss von der Gesamtlänge des Polynoms die Zahl, der Index der ersten Eins abgezogen werden. Dadurch erhält man die Länge des Polynoms, wenn man von der ersten Eins aus anfängt zu zählen. Von dieser Länge muss nun noch eins abgezogen werden, da das letzte Monom den Exponenten Null hat.

Als Ergebnis erhält man den höchsten Exponenten, der in dem Polynom zu finden ist, und dies entspricht dem Grad des Polynoms.

2.2.4 Polynom auf Nullstellen prüfen

Mittel der Funktion *"has_root"* ist es möglich, ein Polynom darauf zu prüfen, ob es eine Nullstelle hat. Ein Polynom hat genau dann eine Nullstelle, wenn man in dem Polynom für die Variable X etwas einsetzen kann, wodurch das Polynom zu Null wird. In der Charakteristik Zwei gibt es hierbei lediglich die beiden Möglichkeiten Null und Eins, die in das Polynom eingesetzt werden können.

Die Null stellt genau dann eine Nullstelle dar, wenn das kleinste Monom den Vorfaktor Null hat. Denn in allen anderen Monomen des Polynoms wird eine Null eingesetzt, wodurch sich all diese zu Null addieren. Dadurch muss nur das niedrigste Monom betrachtet werden, da dies durch den Exponenten von Null nicht von der für die Variable eingesetzten Zahl abhängt. Demnach gilt: Wenn die letzte Stelle des Tupels eine Eins ist, stellt die Null keine Nullstelle dar, und wenn die letzte Stelle des Tupels eine Null ist, ist die Null eine Nullstelle des Polynoms.

Für die Eins hingegen müssen andere Bedingungen überprüft werden. Hierfür ist insbesondere die folgende Rechenregel wichtig, die aufgrund des Binärsystems gilt: $1 + 1 = 0$. Daraus ergibt sich, dass, wenn die Summe der Einser gerade ist, das Ergebnis Null wird, da sich immer zwei Einser gegenseitig auslöschen. Dadurch gilt, dass die Eins eine Nullstelle ist, sollten die Anzahl der Einser im binären Tupel gerade sein. Somit ist bei ungerader Anzahl der Einser im Tupel die Eins keine Nullstelle.

In der Implementierung wurden die eben aufgeführten Bedingungen für Nullstellen folgendermaßen umgesetzt. Da für die Null lediglich die letzte Stelle entscheidend ist, ob sie eine Nullstelle für das Polynom ist, wird überprüft, ob die letzte Stelle eine Null ist. Sollte diese Bedingung stimmen, gibt die Funktion *‘True‘* zurück, da in diesem Fall eine Nullstelle vorliegt. Um die Eins zu überprüfen, wird stellenweise über das binäre Tupel iteriert und die Anzahl der vorkommenden Einsen gezählt. Sollte diese Anzahl gerade sein, gibt es eine Nullstelle und somit gibt die Funktion *‘True‘* zurück. Sollten diese beiden aufgeführten Bedingungen nicht eingetreten sein, hat das Polynom keine Nullstelle und die Funktion gibt *‘False‘* zurück.

2.3 Arithmetische Operationen

Da die Charakteristik Zwei auf dem Binärsystem basiert, verhalten sich die arithmetischen Operationen innerhalb von Körpern der Charakteristik Zwei analog zu den Berechnungen im Binärsystem. Im Folgenden wird die Implementierung der arithmetischen Rechenoperationen für binäre Tupel, welche Polynome der Charakteristik

Zwei darstellen, genauer erläutert.

2.3.1 Addition von Polynomen

Bei der Addition von Polynomen in Körpern der Charakteristik zwei gelten die Rechenregeln wie bei der Addition im Binärsystem. Diese lauten folgendermaßen. $0 + 0 = 0$; $0 + 1 = 1$; $1 + 0 = 1$; $1 + 1 = 0$ (mit Übertrag 1).

Um nun diese Addition in Python zu implementieren, werden der Funktion *ÄddPolynoms"* zunächst zwei Polynome übergeben, die in Form von binären Tupeln dargestellt werden. Um diese beiden binären Tupel zu addieren, wird über die Stellen der Tupel iteriert, und jede Stelle einzeln, nach den oben gegebenen Rechenregeln, binär addiert. Die Ergebnisse werden dann schrittweise dem Lösungspolynom hinzugefügt.

Bevor die iterative Addition der Stellen der Polynome durchgeführt werden kann, muss jedoch sichergestellt werden, dass auch jeweils die passenden Stellen miteinander addiert werden. Als Voraussetzung dafür, dass immer die passenden Stellen miteinander addiert werden, gilt, dass die beiden Polynome, beziehungsweise die beiden Tupel, gleich viele Stellen haben. Um sicherzustellen, dass die beiden Tupel gleich viele Stellen haben, wird, bevor der iterativen Addition geprüft, welches Tupel das längere der beiden ist, und im Anschluss wird das kürzere Tupel von links aus mit Nullen ergänzt, bis es so viele Stellen wie das jeweils andere hat. Diese Nullen dienen lediglich zur richtigen Zuordnung der Stellen bei der Addition. Zudem ändern sie nichts an den Rechnungen, da die Zahl Null das neutrale Element in der Addition darstellt. Demnach wird durch diese Auffüllung mit Nullen das Ergebnis der Addition nicht beeinflusst.

Schließlich wird am Ende das Lösungspolynom in Form eines binären Tupels zurückgegeben.

2.3.2 Multiplikation von Polynomen

Das Multiplizieren von zwei Polynomen funktioniert so, dass man die beiden Polynome mithilfe des Distributivgesetzes ausmultipliziert. Das bedeutet, dass jedes Monom des ersten Polynoms mit jedem Monom des zweiten Polynoms multipliziert wird.

Bei diesem Ausmultiplizieren der einzelnen Monome kann es dazu kommen, dass mehrere Monome die gleiche Potenz haben. Diese müssen dann noch miteinander addiert werden. Da diese Polynome in Körpern der Charakteristik zwei sind, muss hierbei lediglich geprüft werden, ob die Anzahl Monome mit der gleichen Potenz gerade oder ungerade ist.

Die Multiplikation von zwei Polynomen wird durch die Funktion *"MultiplyPolynoms"* ermöglicht. Hierbei wird zu Beginn ein Lösungsarray definiert und alle Stellen mit Null initialisiert. Die Länge dieses Lösungsarrays wird von den Längen der beiden Faktoren bestimmt. Und zwar bekommt das Lösungsarray die Länge der Summe der beiden Faktoren. Davon muss jedoch noch eins abgezogen werden, da die Multiplikation mit dem letzten Monom keine zusätzliche Stelle liefert.

Wie bereits erwähnt, muss jedes Monom des einen Polynoms mit jedem Monom des anderen Polynoms multipliziert werden. Daraus folgt, dass für die Implementierung der Multiplikation von Polynomen zwei ineinander geschachtelte Schleifen benötigt werden, die jeweils über die Länge der beiden Polynome iterieren.

Innerhalb dieser beiden Schleifen wird dann die Multiplikation der einzelnen Monome, sowie die Addition von Monomen gleicher Potenzen durchgeführt. Dabei wird anhand der Schleifenindizes die aktuelle Stelle des Lösungspolynoms ermittelt, und der Inhalt dieser aktuellen Stelle wird mit dem Produkt der Multiplikation der einzelnen Monome mittels einer *'XOR'*-Anweisung verknüpft. Die *'XOR'*-Verknüpfung entspricht in der Charakteristik zwei der Addition, wodurch die Multiplikation und die Addition zusammen in einer Anweisung durchgeführt werden können.

Im Anschluss wird das Lösungsarray noch zu einem Tupel umgewandelt. Für die Zwischenspeicherung der Lösung wurde ein Array benutzt, da Tupel in Python ïmmutableßind, das heißt, man kann den Inhalt nicht mehr ändern. Von diesem Tupel werden anschließend noch, mittels der Hilfsfunktion *"cut_zeros_left"*, die führenden Nullen von links abgeschnitten, da dies überflüssige Stellen sind, und dieses Tupel wird dann von der Funktion zurückgegeben.

2.3.3 Reduktion eines Produkts

Wenn man sich in einem Körper, hier der Charakteristik zwei, befindet, gibt es eine definierende Relation, die diesen Körper definiert. Dafür gilt, dass alle Elemente, die in diesem Körper sind, einen geringeren Grad als das Minimalpolynom, bzw. als die

definierende Relation, haben müssen.

Da es bei dem eben aufgeführten Verfahren der Multiplikation von Polynomen dazu kommen kann, dass die Stellenanzahl, und damit der Grad des Lösungspolynoms nach der Multiplikation größer wird als der Grad des Minimalpolynoms zur zugehörigen definierenden Relation, muss nach der Berechnung der Multiplikation noch eine Reduktion des Produkts mithilfe der definierenden Relation des Körpers durchgeführt werden.

Diese Reduktion funktioniert folgendermaßen. Zunächst werden der Funktion *"reduce_product"* zwei Polynome übergeben. Zum einen das Polynom *"F"*, das reduziert werden soll, und zum anderen das Minimalpolynom *"M"*, das für die definierende Relation steht.

Als erstes wird mittels der Hilfsfunktion *"cut_zeros_left"* überprüft, ob das zu reduzierende Polynom *"F"* leer ist. In diesem Fall gibt die Funktion ein leeres Tupel zurück, da ein leeres Polynom bereits in reduzierter Form vorliegt.

Danach werden die Grade der beiden Polynome mittels der Hilfsfunktion *"find_degree"* ermittelt. Sollte der Grad des Polynoms *"F"* kleiner als der Grad des Polynoms *"M"* sein, bedeutet das, dass das Polynom *"F"* nicht reduziert werden muss. Demnach kann das Polynom *"F"* unverändert zurückgegeben werden.

Sollte der Grad des Polynoms *"F"* jedoch gleich oder höher als der Grad des Polynoms *"M"* sein, bedeutet das für das Polynom *"F"*, dass es reduziert werden muss. Dazu werden zuerst alle führenden Nullen von *"F"* und *"M"* weggeschnitten und im Anschluss werden die beiden Polynome umgedreht. Durch diese Umkehrung der Polynome entsprechen nun die Stellen der Polynome den Exponenten der jeweiligen Monome.

Nun wird das Polynom *"F"* stellenweise mit der Relation reduziert, wobei pro Stelle, die von *"F"* weggeschnitten wird, der Grad von *"F"* um eins verringert wird. Dieser Vorgang wird so lange wiederholt, bis der Grad von *"F"* kleiner als der Grad des Minimalpolynoms *"M"* ist. Im Zuge dieser stellenweisen Reduktion wird zuerst der höchste Exponent des Polynoms *"F"* ermittelt, indem von der Länge des Polynoms eins abgezogen wird. Von diesem höchsten Exponenten des Polynoms wird nun der Grad des Minimalpolynoms *"M"* abgezogen. Dieser Wert beschreibt nun, welcher Faktor nach dem Schritt der Vereinfachung übrig bleibt. Beispielsweise ist der Grad der Relation acht und der höchste Exponent des Polynoms *"F"* ist 13. Dann wird

dieser Exponent 13 mittels der Potenzregel für gleiche Basen aufgeteilt in $X^8 * X^5$. Hierbei kann nun das Monom mit Exponent acht durch die Relation ersetzt werden.

Im weiteren Verlauf der Funktion wird nun die letzte Stelle, entsprechend dem Monom mit dem höchsten Exponenten, des Polynoms *"F"* abgeschnitten, da diese Stelle in dem aktuellen Schritt der Vereinfachung wegfällt. Anschließend wird ein temporäres Polynom erstellt, welches beschreibt, was zu *"F"* addiert werden muss. Dieses temporäre Polynom stellt den aktuellen Schritt der Vereinfachung dar, der eben beispielsweise anhand des Exponenten 13 erklärt wurde. Dazu werden dem temporären Polynom von links so viele Nullen angefügt, wie hoch der Faktor nach dem Schritt der Vereinfachung übrig bleibt. Anschließend wird das Minimalpolynom rechts an dem temporären Polynom angefügt, wobei die letzte Stelle weggelassen wird. Um zurück zu dem eben aufgeführten Beispiel zurückzukommen, entsprechen die Nullen dem Monom mit Exponent fünf, und das Minimalpolynom ohne den höchsten Exponenten stellt die Relation dar.

Anschließend wird das Polynom *"F"* mit dem eben aufgeführten temporären Polynom addiert, und somit wird ein Schritt der Vereinfachung durchgeführt. Dann werden noch alle führenden Nullen von rechts weggeschnitten. Dieser Vorgang wird so lange wiederholt, bis der Grad von *"F"* kleiner als der Grad des Minimalpolynoms *"M"* ist.

Ist diese Bedingung nun erreicht, ist die Reduktion des Polynoms *"F"* mittels der definierenden Relation des Körpers, bzw. mittels des Minimalpolynoms, vollendet. Das Ergebnis wird nun noch um führende Nullen ergänzt, bis es die maximale Länge hat, die in dem jeweiligen Körper möglich ist. Außerdem wird das Ergebnis noch umgekehrt, sodass nun das Monom mit dem höchsten Exponenten wieder an erster Stelle steht.

2.3.4 Alternativer Algorithmus für die Multiplikation

2.3.5 Polynomdivision

2.4 x

2.4.1 Euklidischer Algorithmus

2.4.2 Vergleich der beiden Multiplikations-Algorithmen

3 Beispielaufgaben

x

Literatur

[1] a. c, x. Zugriff am xx. xxxxxxx 2025.

[2] test. test. *test*, test(test):test, test.