ALGEBRA FOR SECURITY DOCUMENTATION

JEROEN UBBINK & RENÉ WOUTERS

October 27, 2015

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

This is the report accompanying the code submission for 2WF70, Algebra for Security. This document will serve as a user guide and feature documentation of the Polynomial calculator program, outlining the the operations the program is capable of performing, and how it does so.

2 METHODS

2.1 Input Requirements

The requirements for the program were as follows:

- Polynomial Objects
  1. Input two polynomials with integer coefficients modulo a prime, with output of the sum, returning the sum, product, and difference of the two entered polynomials.
  2. Input a polynomial with integer coefficients modulo a prime, and a scalar coefficient, returning the polynomial after being scaled.
  3. Input two polynomials with integer coefficients modulo a prime, return the quotient and remainder through long division.
  4. For coefficients modulo a prime, implement the (Extended) Euclidean Algorithm, returning
  5. For input of three polynomials, return true if the first two polynomials are equal to each other modulo the third polynomial.
- Finite Field Objects
  1. Upon input of a prime number and an irreducible polynomial $q(X)$, return the addition and multiplication table of $\mathbb{Z}/p\mathbb{Z}[X]/(q(X))$.
  2. Upon input of two field elements a and b, return the sum, the product, and the quotient ab^{-1} in a field.
  3. Upon input of a field and field element, return the primitivity of the element.
  4. Upon input of a field, return the primitive elements of the field.
  5. Upon input of a polynomial modulo a prime, test the irreducibility, and produce the irreducible polynomials of the prescribed degree.

Our implementation currently implements all of the Polynomial object methods, and none of the Finite Field Object methods.

2.2 Overview of Program

Our program is split into three classes, the main class, which parses input. The polynomial class, which defines the polynomial class of objects, and the finite field class, which defined the finite field class of objects.

PARSER Our parser class has a number of methods, but generally works through parsing the input stream, and depending on the input selectors, selects one of the ten cases, and then calls a method in order to parse the remainder of the input stream to convert the text input into polynomial objects if necessary. It also parses the input stream to integers.//

The input strings required for each operation are as follows:
Polynomial objects:

1. poly basic polynomial1 polynomial2 prime, where the first two words are strings, the next two strings are polynomial objects, and the prime is an integer.
2. poly scalar polynomial1 coefficient prime, where the first two words are strings, the next string is a polynomial object, and the next two strings are integers.
3. poly division polynomial1 polynomial2 prime, where the first two words are strings, the next two strings are polynomial objects, and the final string is an integer.
4. poly GCD polynomial1 polynomial2 prime, where the first two words are strings, the next two strings are polynomial objects, and the final string is an integer.
5. poly polymod polynomial1 polynomial2 polynomial3, where the first two words are strings, and the three polynomial objects are the polynomials that you would like to compare to each other.

It is important to note here that strings must be entered *exactly* as above (including capitals), meaning the first two words in the input. The polynomials must always have a term included, with an example as follows: $-2X^-3-1X^0$. As seen, it is allowed to have negative coefficients and exponents. Finally, for input of all integers, negatives are allowed.

Finite Field inputs have not currently been implemented, and thus return an error message if the first string is FF, which has been chosen as the selector for that set of inputs.

POLYNOMIAL The Polynomial class defines the object used to hold a polynomial, which is done through a HashMap, which stores the set of exponents as an index, which maps to the coefficients. It also has an integer value $modulus$, which holds the modulo for the polynomial, which if uninitialized is the maximum value -1 of an integer in java. The coefficients have mod $modulus$ applied to them after every operation that changes their value.

This class also has all of the methods to perform operations on polynomial objects, with standard get and set methods for the "terms" Hashmap and the "modulus" integers.

The methods in order to do Long Division(Algorithm 1.2.6), Greatest Common Divisor (Algorithm 1.2.10), and extended GCD(Algorithm 1.2.11) operations are from the book, modified to use for each loops to iterate through the HashMaps in order to get the values to perform the computations.

The methods in order to do addition, subtraction, and multiplication were done through for each loops (nested for each loops for multiplication) to iterate through the terms in order to do the operations. Currently, the implementation does not remove terms if the coefficient is zero.

FINITE FIELDS Currently does not do anything.

2.3 Proof of Function

BASIC METHODS poly basic polynomial1 polynomial2 prime, where the first two words are strings, the next two strings are polynomial objects, and the prime is an integer.

SCALAR MULTIPLICATION poly scalar polynomial1 coefficient prime, where the first two words are strings, the next string is a polynomial object, and the next two strings are integers.

LONG DIVISION poly division polynomial1 polynomial2 prime, where the first two words are strings, the next two strings are polynomial objects, and the final string is an integer.

(EXTENDED) EUCLID'S ALGORITHM poly GCD polynomial1 polynomial2 prime, where the first two words are strings, the next two strings are polynomial objects, and the final string is an integer.

TWO POLYNOMIALS MODULO ANOTHER poly polymod polynomial1 polynomial2 polynomial3, where the first two words are strings, and the three polynomial objects are the polynomials that you would like to compare to each other.