2WF90: Exercises and programming assignments

Andreas Hülsing, Benne de Weger

Q1 2021

Contents

1 Introduction

The programming assignment for the second part of the course consists of writing code ('from scratch') for arithmetic with polynomials mod p and within finite fields. Your code should deal with various objects, such as integers modulo a number, and elements in a finite field, and with arithmetic in finite fields.

1.1 Role in assessment

This programming assignments contributes 12.5% to your 2WF90 grade.

1.2 General guidelines

The separate document "General Guidelines" also applies to this assignment.

2 Software Assignment on Polynomial and Finite Field Arithmetic

The programming language for this assignment is Python. You are supposed to write a program that reads a list of instructions from a file, executes the instructions, and writes the full set of instructions with added results into an output file (see 2.2). An instruction is a mathematical operation to be executed together with the respective parameters (see 2.3). As for the first assignment you have to accompany your code with a documentation as explained in the "General Guidelines". We now first detail the main objects you have to deal with, then describe the expected interface of your code. Afterwards, we discuss the functionalities you have to implement. We conclude with details on grading. In the appendix you find the ASN.1 specification for input and output files.

2.1 Objects

Here is a description of the main objects you have to deal with. The provided bounds are there to inform you about the magnitude of these values. You do not have to verify that values are within these bounds.

- Polynomials with integer coefficients mod p. You may assume that $p < 100$. You may further assume a maximum degree of $n < 256$.
- Finite fields and elements in a finite field. Given a prime $p < 100$ and an exponent $n < 256$ your program must be able to handle a field with p^n elements, like $\mathbb{Z}/p\mathbb{Z}[X]/(f(X))$ for some irreducible polynomial f of degree n (see the lecture notes for algorithms to produce irreducible polynomials).

You are free of course to implement additional objects if needed and extend the bounds given above as you see fit. In all cases, describe in your accompanying report the limitations of your software.

2.2 Program interface

Your program MUST be delivered as one or more python files. It HAS to contain a file main.py that called with no command line parameters reads a file "input.ops" from the directory it is called from, executes the operations specified in there, and writes a file "output.ops" to the same directory (i.e., you read from and write to directory "."). As for the first software assignment, we will provide an ASN.1 specification for the input/output files and code to support processing JSON files that follow the ASN.1 specification (see below). Note that the format for input and output files is the same. The only difference is that the output files should contain answers. Especially, the output file MUST still contain the parameters of the input file.

Although you MAY allow for optional command line parameters (e.g., change the file name via command line parameter) to ease development, the above interface is mandatory.

The reason is that your code will be tested by a script that expects the above behavior (see section on grading below).

You MAY assume that the file provided as input.ops follows the specification and do not have to write error handling code for this. However, your code MUST be able to handle cases where a specified operation is invalid (e.g. the input asks to apply division with remainder by 0, or the provided modulus p is not prime). In these cases your code MUST write an error message into the answer field and MUST continue with the next operation. Do not abort the script in case of such a functional error.

2.3 Functional Interface

Your code HAS to implement the following instructions. Instructions are provided via the input file. Here we detail the input and output parameters of each operation.

Polynomials are provided as type Poly which is a list of 32 bit signed integers. These integers specify the coefficients of the polynomial starting with the leading non-zero coefficient. For example, the list 1 2 3 4 5 6 represents the polynomial $X^5 + 2X^4 + 3X^3 + 4X^2 + 5X + 6$. We now first define the functionalities from the area of polynomial arithmetic that have to be implemented and afterwards define those for finite field arithmetic.

2.3.1 Polynomial arithmetic

Your code MUST support the following operations for polynomials in $\mathbb{Z}/p\mathbb{Z}[X]$ where p is given per operation as input "mod":

DisplayPoly Display a polynomial in pretty print.

- Input: Integer mod, Poly f
- Output: String answer
- Functionality: Pretty print the polynomial f. This means, all coefficients get reduced modulo p and powers of X are added. E.g.: $X^5 + 2X^4 + 3X^3 + 4X^2 + 5X + 6$.

AddPoly Polynomial addition

- Input: Integer mod, Poly f, Poly g
- Output: String answer, Poly answer-poly
- Functionality: Compute the sum $f + g$ of polynomials f and g in $\mathbb{Z}/p\mathbb{Z}[X]$. Output $f + g$ as pretty print string and as Poly.

SubtractPoly Polynomial subtraction

- Input: Integer mod, Poly f, Poly g
- Output: String answer, Poly answer-poly

- Functionality: Compute the difference $f-g$ of polynomials f and g in $\mathbb{Z}/p\mathbb{Z}[X]$. Output $f-g$ as pretty print string and as Poly.

MultiplyPoly Polynomial multiplication

- Input: Integer mod, Poly f, Poly g
- Output: String answer, Poly answer-poly
- Functionality: Compute the product $f \cdot g$ of polynomials f and g in $\mathbb{Z}/p\mathbb{Z}[X]$. Output $f \cdot g$ as pretty print string and as Poly.

LongDivPoly Polynomial long division.

- Input: Integer mod, Poly f, Poly g
- Output: String answ-q, String answ-r, Poly answ-q-poly, Poly answ-r-poly
- Functionality: Execute a long division to compute polynomials $q, r \in \mathbb{Z}/p\mathbb{Z}[X]$ such that $f = q \cdot g + r$ in $\mathbb{Z}/p\mathbb{Z}[X]$ (see e.g. Algorithm 2.2.2). Output q and r as pretty print string and as Poly, each.

EuclidPoly Extended Euclidean algorithm for polynomials.

- Input: Integer mod, Poly f, Poly g
- Output: String answ-a, String answ-b, String answ-d, Poly answ-a-poly, Poly answ-b-poly, Poly answ-d-poly
- Functionality: Execute the extended Euclidean algorithm to compute polynomials $a, b, d \in \mathbb{Z}/p\mathbb{Z}[X]$ such that $af + bg = d = \gcd(f, g)$in $\mathbb{Z}/p\mathbb{Z}[X]$. Remember, that we defined the gcd to be monic, i.e., the leading coefficient has to be 1 (see Algorithms 2.2.10). Output a, b, and d as pretty print string and as Poly, each.

EqualsPolyMod Equality of polynomials modulo a polynomial.

- Input: Integer mod, Poly f, Poly g, Poly h
- Output: Boolean answer
- Functionality: Return true iff $f \equiv g \mod h$ in $\mathbb{Z}/p\mathbb{Z}[X]$.

Irreducible Check irreducibility.

- Input: Integer mod, Poly f
- Output: Boolean answer
- Functionality: Return true iff f is irreducible in $\mathbb{Z}/p\mathbb{Z}[X]$

FindIrred Find irreducible polynomial.

- Input: Integer mod, Integer, deg

- Output: String answer, Poly answer-poly
- Functionality: Find an irreducible polynomial $f \in \mathbb{Z}/p\mathbb{Z}[X]$ of degree n. Output f as pretty print string and as Poly.

2.3.2 Arithmetic in a finite field

Your code MUST support the following operations for finite fields. All operations take an Integer mod which we will denote p here and a Poly mod-poly which we will denote $q(X)$ below. So, given a prime p and an irreducible polynomial $q(X)$ in $\mathbb{Z}/p\mathbb{Z}[X]$, we consider the field $\mathbb{F} = \mathbb{Z}/p\mathbb{Z}[X]/(q(X))$ in all of the following operations.

Your code MUST implement the following operations:

AddTable Generate addition table.

- Input: Integer mod, Poly mod-poly
- Output: Two-dimensional String array answer, two-dimensional Poly array answer-poly
- Functionality: Generate the full addition table of $\mathbb{F}$. Return it once with elements pretty printed and once with elements as Poly's. In final format, the output is supposed to be a sequence of rows (see examples).

MultTable Generate multiplication table.

- Input: Integer mod, Poly mod-poly
- Output: Two-dimensional String array answer, two-dimensional Poly array answer-poly
- Functionality: Generate the full multiplication table of $\mathbb{F}$. Return it once with elements pretty printed and once with elements as Poly's. In final format, the output is supposed to be a sequence of rows (see examples).

DisplayField Display unique representative.

- Input: Integer mod, Poly mod-poly, Poly a
- Output: String answer, Poly answer-poly
- Functionality: Output a representative of the field element of F that represents a, this means the unique polynomial f of degree less than the degree of $q(x)$ that is congruent to a. Output f as pretty print string and as Poly.

AddField Finite field addition.

- Input: Integer mod, Poly mod-poly, Poly a, Poly b
- Output: String answer, Poly answer-poly

- Functionality: Compute the element $f = a + b$ in $\mathbb{F}$. Output f as pretty print string and as Poly.

SubtractField Finite field subtraction.

- Input: Integer mod, Poly mod-poly, Poly a, Poly b
- Output: String answer, Poly answer-poly
- Functionality: Compute the element $f = a - b$ in $\mathbb{F}$. Output f as pretty print string and as Poly.

MultiplyField Finite field multiplication.

- Input: Integer mod, Poly mod-poly, Poly a, Poly b
- Output: String answer, Poly answer-poly
- Functionality: Compute the element $f = a \cdot b$ in $\mathbb{F}$. Output f as pretty print string and as Poly.

InverseField Computing multiplicative inverses.

- Input: Integer mod, Poly mod-poly, Poly a
- Output: String answer, Poly answer-poly
- Functionality: Compute the multiplicative inverse of a, i.e., the element $f \in \mathbb{F}$ such that $a \cdot f = 1$ (use for instance Algorithm 4.1.5, which is based on the Extended Euclidean algorithm). Output f as pretty print string and as Poly.

DivisionField Finite field division.

- Input: Integer mod, Poly mod-poly, Poly a, Poly b
- Output: String answer, Poly answer-poly
- Functionality: Compute the element $f = a/b$ in $\mathbb{F}$. Output f as pretty print string and as Poly.

EqualsField Finite field comparison.

- Input: Integer mod, Poly mod-poly, Poly a, Poly b
- Output: Boolean answer
- Functionality: Test if $a = b$ in $\mathbb{F}$. Output true iff $a = b$ in $\mathbb{F}$.

Primitive Check primitivity.

- Input: Integer mod, Poly mod-poly, Poly a
- Output: Boolean answer

- Functionality: Verify if a is a primitive element in $\mathbb{F}$. Output true iff a is primitive in $\mathbb{F}$.

FindPrim Find a primitive element.

- Input: Integer mod, Poly mod-poly
- Output: String answer, Poly answer-poly
- Functionality: Compute a primitive element f in $\mathbb{F}$ (see Algorithm 4.4.4). Output f as pretty print string and as Poly.

2.4 Grading

The reason we are so strict about the interface of your program is that your submission for this assignment will be tested partially by means of automated software. Our program does the following repeatedly:

- Construct a file called input.ops in the folder containing your file. This file contains input for a single command to be executed (see the example file for example inputs);
- Run your code using a command line prompt executing "python3 main.py";
- If your code is still running after some amount of time (typically 5 seconds), terminate the code;
- Read the file output.ops (to be produced by your program), and test the correctness of your output.

This means that your software should typically be able to produce a correct output to a given input in less than 5 seconds, and write it to an output file. The new output file should only contain this output; i.e., make sure to clear / overwrite any earlier output files (do not open the file in append mode).

Tests range from simple (add these two polynomials of degree 3, find an irreducible polynomial of degree 3 with coefficients mod 7) to tricky (modulus not prime, zero divisors, special cases) or computationally difficult (produce an irreducible polynomial of degree 10 with coefficients modulo 11, test primitivity in a field with $11^{10} - 1$ elements). All of these tests are in principle doable in 5 seconds, but you might have to think on how to make your program efficient enough to do so. More than half of the tests will be simple.

3 Appendix

Here the asn.1 format and an example file are provided (both also provided as separate files on Canvas).

3.1 operations.asn

```
Poly DEFINITIONS ::= BEGIN

Exercises ::= SET {
exercises SEQUENCE OF Exercise
}

    Poly ::= SEQUENCE OF INTEGER

    Row ::= SEQUENCE OF IA5String

    RowPoly ::= SEQUENCE OF Poly

    DisplayPoly ::= SET {
        mod INTEGER,
        f Poly,
        answer IA5String
    }

    AddPoly ::= SET {
        mod INTEGER,
        f Poly,
        g Poly,
        answer IA5String,
        answer-poly Poly
    }

    SubtractPoly ::= SET {
        mod INTEGER,
        f Poly,
        g Poly,
        answer IA5String,
        answer-poly Poly
    }

    MultiplyPoly ::= SET {
        mod INTEGER,
        f Poly,
        g Poly,
        answer IA5String,
        answer-poly Poly
    }
```

```
LongDivPoly ::= SET {
    mod INTEGER,
    f Poly,
    g Poly,
    answ-q IA5String,
    answ-r IA5String,
    answ-q-poly Poly,
    answ-r-poly Poly
}

EuclidPoly ::= SET {
    mod INTEGER,
    f Poly,
    g Poly,
    answ-a IA5String,
    answ-b IA5String,
    answ-d IA5String,
    answ-a-poly Poly,
    answ-b-poly Poly,
    answ-d-poly Poly
}

EqualsPolyMod ::= SET {
    mod INTEGER,
    f Poly,
    g Poly,
    h Poly,
    answer BOOLEAN
}

Irreducible ::= SET {
    mod INTEGER,
    f Poly,
    answer BOOLEAN
}

FindIrred ::= SET {
    mod INTEGER,
    deg INTEGER,
    answer IA5String,
    answer-poly Poly
}
```

```
AddTable ::= SET {
    mod INTEGER,
    mod-poly Poly,
    answer SEQUENCE OF Row
    answer-poly SEQUENCE OF RowPoly
}

MultTable ::= SET {
    mod INTEGER,
    mod-poly Poly,
    answer SEQUENCE OF Row
    answer-poly SEQUENCE OF RowPoly
}

DisplayField ::= SET {
    mod INTEGER,
    mod-poly Poly,
    a Poly,
    answer IA5String,
    answer-poly Poly
}

AddField ::= SET {
    mod INTEGER,
    mod-poly Poly,
    a Poly,
    b Poly,
    answer IA5String,
    answer-poly Poly
}

SubtractField ::= SET {
    mod INTEGER,
    mod-poly Poly,
    a Poly,
    b Poly,
    answer IA5String,
    answer-poly Poly
}

MultiplyField ::= SET {
    mod INTEGER,
    mod-poly Poly,
    a Poly,
    b Poly,
    answer IA5String,
    answer-poly Poly
}

InverseField ::= SET {
    mod INTEGER,
    mod-poly Poly,
    a Poly,
    answer IA5String,
    answer-poly Poly
}

DivisionField ::= SET {
    mod INTEGER,
    mod-poly Poly,
    a Poly,
    b Poly,
    answer IA5String,
    answer-poly Poly
}

EqualsField ::= SET {
    mod INTEGER,
    mod-poly Poly,
    a Poly,
    b Poly,
    answer BOOLEAN
}

Primitive ::= SET {
    mod INTEGER,
    mod-poly Poly,
    a Poly,
    answer BOOLEAN
}

FindPrim ::= SET {
    mod INTEGER,
    mod-poly Poly,
    answer IA5String,
        answer-poly Poly
    }

    Exercise ::= CHOICE {
        display-poly DisplayPoly,
        add-poly AddPoly,
        subtract-poly SubtractPoly,
        multiply-poly MultiplyPoly,
        long-div-poly LongDivPoly,
        euclid-poly EuclidPoly,
        equals-poly-mod EqualsPolyMod,
        irreducible Irreducible,
        find-irred FindIrred,
        add-table AddTable,
        mult-table MultTable,
        display-field DisplayField,
        add-field AddField,
        subtract-field SubtractField,
        multiply-field MultiplyField,
        inverse-field InverseField,
        division-field DivisionField,
        equals-field EqualsField,
        primitive Primitive,
        find-prim FindPrim
    }

END
```

3.2 output.ops

The output file as it should look after receiving the same file with blank answer entries. **Note, this example does not contain the “-poly” answer entries as they make things unreadable. Those are included in the actual file on Canvas**.

```
{
  "exercises": [
    {
      "display-poly": {
        "mod": 7,
        "f": [
          1,
          2,
          6
    ],
    "answer": "X^2+2X+6"
  }
},
{
  "display-poly": {
    "mod": 5,
    "f": [
      1,
      2,
      6
    ],
    "answer": "X^2+2X+1"
  }
},
{
  "display-poly": {
    "mod": 7,
    "f": [
      1,
      2,
      0
    ],
    "answer": "X^2+2X"
  }
},
{
  "display-poly": {
    "mod": 7,
    "f": [
      1,
      2,
      7
    ],
    "answer": "X^2+2X"
  }
},
{
  "display-poly": {
    "mod": 7,
    "f": [
      0,
      1,
      2,
      0
    ],
    "answer": "X^2+2X"
  }
},
{
  "display-poly": {
    "mod": 7,
    "f": [
      -1,
      0,
      1,
      3
    ],
    "answer": "6X^3+X+3"
  }
},
{
  "display-poly": {
    "mod": 7,
    "f": [
      0,
      1,
      10,
      -1,
      0,
      2,
      3
    ],
    "answer": "X^5+3X^4+6X^3+2X+3"
  }
},
{
  "display-poly": {
    "mod": 7,
    "f": [
      0
    ],
    "answer": "0"
  }
},
{
  "display-poly": {
    "mod": 7,
    "f": [
      0,
      0
    ],
    "answer": "0"
  }
},
{
  "add-poly": {
    "mod": 7,
    "f": [
      5,
      2,
      3
    ],
    "g": [
      2,
      3,
      4,
      0
    ],
    "answer": "2X^3+X^2+6X+3"
  }
},
{
  "subtract-poly": {
    "mod": 7,
    "f": [
      1,
      2,
      3
    ],
    "g": [
      2,
      3,
      4,
      0
    ],
    "answer": "5X^3+5X^2+5X+3"
  }
},
{
  "multiply-poly": {
    "mod": 7,
    "f": [
      6
    ],
    "g": [
      5
    ],
    "answer": "2"
  }
},
{
  "multiply-poly": {
    "mod": 7,
    "f": [
      27
    ],
    "g": [
      33
    ],
    "answer": "2"
  }
},
{
  "multiply-poly": {
    "mod": 7,
    "f": [
      1,
      1,
      1
    ],
    "g": [
      1,
      -1
    ],
    "answer": "X^3+6"
  }
},
{
  "long-div-poly": {
    "mod": 7,
    "f": [
      6
    ],
    "g": [
      5
    ],
    "answ-q": "4",
    "answ-r": "0"
  }
},
{
  "long-div-poly": {
    "mod": 7,
    "f": [
      1,
      1,
      1
    ],
    "g": [
      2,
      -2
    ],
    "answ-q": "4X+1",
    "answ-r": "3"
  }
},
{
  "long-div-poly": {
    "mod": 7,
    "f": [
      1,
      1,
      1
    ],
    "g": [
      0
    ],
    "answ-q": "ERROR",
    "answ-r": "ERROR"
  }
},
{
  "euclid-poly": {
    "mod": 7,
    "f": [
      1,
      1,
      1
    ],
    "g": [
      2,
      -2
    ],
    "answ-a": "5",
    "answ-b": "X+2",
    "answ-d": "1"
  }
},
{
  "euclid-poly": {
    "mod": 7,
    "f": [
      1,
      0,
      1
    ],
    "g": [
      1,
      0,
      0,
      1
    ],
    "answ-a": "3X^2+3X+4",
    "answ-b": "4X+4",
    "answ-d": "1"
  }
},
{
  "euclid-poly": {
    "mod": 2,
    "f": [
      1,
      0,
      1
    ],
    "g": [
      1,
        0,
        0,
        1
      ],
      "answ-a": "X",
      "answ-b": "1",
      "answ-d": "X+1"
    }
  },
  {
    "euclid-poly": {
      "mod": 7,
      "f": [
        1,
        1,
        1
      ],
      "g": [
        0
      ],
      "answ-a": 1,
      "answ-b": 0,
      "answ-d": "X^2+X+1"
    }
  },
  {
    "euclid-poly": {
      "mod": 7,
      "f": [
        2,
        2,
        2
      ],
      "g": [
        0
      ],
      "answ-a": 4,
      "answ-b": 0,
      "answ-d": "X^2+X+1"
    }
  },
  {
    "equals-poly-mod": {
```

```
    "mod": 7,
    "f": [
      1,
      1,
      1
    ],
    "g": [
      10
    ],
    "h": [
      1,
      -1
    ],
    "answer": true
  }
},
{
  "equals-poly-mod": {
    "mod": 5,
    "f": [
      1,
      1,
      1
    ],
    "g": [
      10
    ],
    "h": [
      1,
      -1
    ],
    "answer": false
  }
},
{
  "equals-poly-mod": {
    "mod": 7,
    "f": [
      1,
      1,
      1
    ],
    "g": [
      3
    ],
    "h": [
      0
    ],
    "answer": false
  }
},
{
  "irreducible": {
    "mod": 2,
    "f": [
      1,
      1,
      1
    ],
    "answer": true
  }
},
{
  "irreducible": {
    "mod": 3,
    "f": [
      1,
      1,
      1
    ],
    "answer": false
  }
},
{
  "find-irred": {
    "mod": 2,
    "deg": "3",
    "answer": "X^3+X+1"
  }
},
{
  "find-irred": {
    "mod": 2,
    "deg": "3",
    "answer": "X^3+X^2+1"
  }
```

```
},
{
  "find-irred": {
    "mod": 2,
    "deg": "4",
    "answer": "X^4+X+1"
  }
},
{
  "add-table": {
    "mod": 2,
    "mod-poly": [
      1,
      1,
      1
    ],
    "answer": [
      [
        "0",
        "1",
        "X",
        "X+1"
      ],
      [
        "1",
        "0",
        "X+1",
        "X"
      ],
      [
        "X",
        "X+1",
        "0",
        "1"
      ],
      [
        "X+1",
        "X",
        "1",
        "0"
      ]
    ]
  }
```

```
},
{
  "mult-table": {
    "mod": 2,
    "mod-poly": [
      1,
      1,
      1
    ],
    "answer": [
      [
        "0",
        "0",
        "0",
        "0"
      ],
      [
        "0",
        "1",
        "X",
        "X+1"
      ],
      [
        "0",
        "X",
        "X+1",
        "1"
      ],
      [
        "0",
        "X+1",
        "1",
        "X"
      ]
    ]
  }
},
{
  "add-table": {
    "mod": 7,
    "mod-poly": [
      1,
      0
```

```
],
"answer": [
  [
    "0",
    "1",
    "2",
    "3",
    "4",
    "5",
    "6"
  ],
  [
    "1",
    "2",
    "3",
    "4",
    "5",
    "6",
    "0"
  ],
  [
    "2",
    "3",
    "4",
    "5",
    "6",
    "0",
    "1"
  ],
  [
    "3",
    "4",
    "5",
    "6",
    "0",
    "1",
    "2"
  ],
  [
    "4",
    "5",
    "6",
    "0",
          "1",
          "2",
          "3"
        ],
        [
          "5",
          "6",
          "0",
          "1",
          "2",
          "3",
          "4"
        ],
        [
          "6",
          "0",
          "1",
          "2",
          "3",
          "4",
          "5"
        ]
      ]
    }
  },
  {
    "mult-table": {
      "mod": 7,
      "mod-poly": [
        1,
        0
      ],
      "answer": [
        [
          "0",
          "0",
          "0",
          "0",
          "0",
          "0",
          "0"
        ],
        [
    "0",
    "1",
    "2",
    "3",
    "4",
    "5",
    "6"
  ],
  [
    "0",
    "2",
    "4",
    "6",
    "1",
    "3",
    "5"
  ],
  [
    "0",
    "3",
    "6",
    "2",
    "5",
    "1",
    "4"
  ],
  [
    "0",
    "4",
    "1",
    "5",
    "2",
    "6",
    "3"
  ],
  [
    "0",
    "5",
    "3",
    "1",
    "6",
    "4",
    "2"
```

```
      ],
      [
        "0",
        "6",
        "5",
        "4",
        "3",
        "2",
        "1"
      ]
    ]
  }
},
{
  "display-field": {
    "mod": 5,
    "mod-poly": [
      1,
      0,
      2
    ],
    "a": [
      1,
      1
    ],
    "answer": "X+1"
  }
},
{
  "display-field": {
    "mod": 5,
    "mod-poly": [
      1,
      0,
      2
    ],
    "a": [
      1,
      0,
      0
    ],
    "answer": "3"
  }
```

```
},
{
  "display-field": {
    "mod": 7,
    "mod-poly": [
      2,
      -2
    ],
    "a": [
      1,
      1,
      1
    ],
    "answer": "3"
  }
},
{
  "add-field": {
    "mod": 2,
    "mod-poly": [
      1,
      1,
      1
    ],
    "a": [
      1,
      1
    ],
    "b": [
      1,
      0
    ],
    "answer": "1"
  }
},
{
  "add-field": {
    "mod": 7,
    "mod-poly": [
      2,
      -2
    ],
    "a": [
        1,
        1,
        1
      ],
      "b": [
        2
      ],
      "answer": "5"
    }
  },
  {
    "subtract-field": {
      "mod": 3,
      "mod-poly": [
        1,
        0,
        2,
        1
      ],
      "a": [
        1,
        1,
        2
      ],
      "b": [
        2,
        0,
        1
      ],
      "answer": "2X^2+X+1"
    }
  },
  {
    "multiply-field": {
      "mod": 3,
      "mod-poly": [
        1,
        0,
        2,
        1
      ],
      "a": [
        1,
          1
        ],
        "b": [
          1,
          2
        ],
        "answer": "X^2+2"
      }
    },
    {
      "multiply-field": {
        "mod": 3,
        "mod-poly": [
          1,
          0,
          2,
          1
        ],
        "a": [
          1,
          0,
          0
        ],
        "b": [
          1,
          0
        ],
        "answer": "X+2"
      }
    },
    {
      "inverse-field": {
        "mod": 2,
        "mod-poly": [
          1,
          1,
          1
        ],
        "a": [
          1,
          0
        ],
        "answer": "X+1"
    }
  },
  {
    "inverse-field": {
      "mod": 2,
      "mod-poly": [
        1,
        1,
        0
      ],
      "a": [
        1,
        0
      ],
      "answer": "ERROR"
    }
  },
  {
    "division-field": {
      "mod": 2,
      "mod-poly": [
        1,
        1,
        1
      ],
      "a": [
        1,
        0
      ],
      "b": [
        1,
        0
      ],
      "answer": "1"
    }
  },
  {
    "division-field": {
      "mod": 2,
      "mod-poly": [
        1,
        1,
        1
    ],
    "a": [
      1
    ],
    "b": [
      1,
      0
    ],
    "answer": "X+1"
  }
},
{
  "division-field": {
    "mod": 2,
    "mod-poly": [
      1,
      1,
      1
    ],
    "a": [
      1
    ],
    "b": [
      0
    ],
    "answer": "ERROR"
  }
},
{
  "equals-field": {
    "mod": 5,
    "mod-poly": [
      1,
      0,
      2
    ],
    "a": [
      1,
      0,
      0
    ],
    "b": [
      3
    ],
    "answer": true
  }
},
{
  "primitive": {
    "mod": 7,
    "mod-poly": [
      1,
      0,
      0,
      2
    ],
    "a": [
      1,
      0
    ],
    "answer": false
  }
},
{
  "primitive": {
    "mod": 7,
    "mod-poly": [
      1,
      0,
      0,
      2
    ],
    "a": [
      1,
      0,
      1
    ],
    "answer": true
  }
},
{
  "find-prim": {
    "mod": 7,
    "mod-poly": [
      1,
      0,
          6
        ],
        "answer": "ERROR"
      }
    },
    {
      "find-prim": {
        "mod": 7,
        "mod-poly": [
          1,
          0,
          1
        ],
        "answer": "2X+6"
      }
    },
    {
      "find-prim": {
        "mod": 7,
        "mod-poly": [
          1,
          0,
          1
        ],
        "answer": "X+2"
      }
    }
  ]
}
```