PROJECT IN ADVANCED ALGEBRA 83-804, 2024

Given a prime number p, let $k = \mathbb{F}_p$ be the corresponding prime field. The aim of this project is to implement finite fields of cardinality p^n for any natural number n. Let l be an n-dimensional extension of k by an irreducible polynomial $f(x) \in k[x]$ of degree n.

Note. We have that $l \cong k[x]/\langle f(x) \rangle$. Elements of l can be represented as vectors of length n, that corresponds to polynomials in $k[x]$ of degree less than n.
That is, an element $\alpha \in l$ can be given as $[a_0, a_1, \ldots, a_{n-1}]$ or as $a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$ for some coefficients $a_i \in k$. In particular, the cardinality of l is p^n, and assume it is not too large.

Directions

(1) Choose a programming language in which you are comfortable writing code, that supports OOP (object oriented programming) or an equivalent for classes.
For the different functions and methods you implement below, make sure to add test cases to verify correctness.

(2) If your programming language supports (or it has a library) for basic operations over k, you can use it. Please do not wrap an existing library for l.
Otherwise, you should implement basic arithmetic over k.
That is, create a class with a name like `PrimeFieldElement`. Its constructor should get two integers (a, p) as arguments that represent the element $a \in k$. Use operator overloading for addition, subtraction, multiplication, division and inversion modulo p. The only tricky part is inversion modulo p, which requires the extended Euclidean algorithm. If $\gcd(a, p) = p$, then $a \equiv 0 \pmod{p}$, and of course a does not have an inverse in k. If $\gcd(a, p) = 1$, then by the extened Euclidean alogirthm we have $s, t \in \mathbb{Z}$ such that $sa + tp = 1$, and then $a^{-1} \equiv s \pmod{p}$.

(3) Create two classes: The first class can be named `FiniteField` that represents l, whose constructor gets the parameters $(p, f(x))$. You can assume that p is indeed prime, and $f(x)$ is indeed irreducible, or for degrees 2 and 3 check it by not finding any root in k.
The second class will be `FiniteFieldElement` that represents $\alpha \in l$, whose constructor gets a vector of length n with the coefficients $[a_0, a_1, \ldots, a_{n-1}]$. Remember to make sure it is somehow linked to l.

(4) Given a non-zero $\alpha \in l$, realize it as an element of $\mathbf{GL}_n(k)$. In other words, find the image of an embedding $\varphi\colon l^\times \hookrightarrow \mathbf{GL}_n(k)$. This is where you can use the library for operations of scalars in k, or your own implementation. Denote $G = \mathrm{Im}(\varphi)$, which is isomorphic to the multiplicative group $l^\times$.
Hint: The element $1 \in l$ should be sent to the identity element, and an element of the prime subfield k should be sent to a scalar matrix.
One way this can be implemented is to recall that $\{1, x, \ldots, x^{n-1}\}$ is a basis for l over k. Then you will need to compute how does the coefficients vector of $x^i \cdot \alpha$ looks like. It helps that $x^i = x^{i-1} \cdot x$, so we can use induction, and reduce monomials with high degree (i.e. more than $n-1$) modulo $f(x)$.

(5) Implement pretty-printing of instances of `FiniteFieldElement` as polynomials, matrices and vectors. Using operator overloading, also implement basic arithmetic. That is, implement addition, subtraction, multiplication and division. Remember to raise an exception (or equivalent in your chosen programming language) when dividing by zero. For multiplication and division, it can be very useful to use the above matrix realization, if you can easily multiply and invert matrices. Equivalently, implement the extended Euclidean algorithm for polynomials in $k[x]$ with $f(x)$.

(6) Implement an exponentiation method for elements of k and l, using exponentiation by squaring. Remember to support any integer as the exponent, e.g. -1 or 0. This is more efficient than naive exponentiation.

(7) Implement a method for `FiniteFieldElement` to return its multiplicative order. That is, for $\alpha \in l^\times$ return $o(\alpha)$ and raise an exception for the zero element.

(8) Implement a method for `FiniteField` that finds a generator γ of the multiplicative group $l^\times$, which we know is cyclic.
Hint: Finding such an element deterministically can be difficult when p^n is large. In our case naive approaches can work, and there should be quite a lot of generators in $l^\times$.

(9) Assume n is even. Using the BSGS algorithm, write a function that given $h, \gamma \in l^\times$ determines if $h = \gamma^x$ for some integer $0 \leq x < p^n$, and return x. Otherwise, throw an exception. This algorithm can be implemented as follows:
Giant step: Set $m = p^{n/2}$, and insert elements of the type $h \cdot \gamma^{-im}$, for $0 \leq i < m$, into a hash table of size m, indexed by the lower $n/2$ entries. Try to reduce unnecessary computations.
Baby step: Run over successive powers γ^j and compare each with the elements sharing the same $n/2$ lower entries in the table. Once you get a match: $h \cdot \gamma^{-im} = \gamma^j$ return $x = j + im$.

Good luck!