CS201A:Mathematics for Computer Science - I
Assignment 4

Anuj Singhal
Roll No. 210166

November 2022

1 [10+3 points]

Write a pseudocode to find an irreducible polynomial $f(x)$, of degree d, over the prime field $\mathbb{F}_p$. The input is d, p.

We use some properties of irreducible polynomials that we have studied in the lectures. It states f is irreducible over $\mathbb{F}_p$ if
1. f divides $(x^{p^d} - x)$ and
2. $gcd(f, x^{p^{d/p_i}} - x) = 1$ for each prime divisor p_i of d
Using the above property of irreducible polynomials, we can deduce the following algorithm which is, even though not very fast but is a major improvement over the brute force algorithm.

Algorithm 1 Algorithm to find an irreducible polynomial over $\mathbb{F}_p$

```
find_irreducible_polynomial(p,d)
prime_divisors ⟵ all the prime factors of d
k ⟵ number of prime divisors
for i = 1 to k do
    n[i] ⟵ d / prime_divisors[i]
end for
for all g in degree d polynomials over F_p do
    isIrreducible ⟵ True
    for i = 1 to k AND isIrreducible = True do
        Compute h = x^(p^(n[i])) − x mod g
        Compute gcd ⟵ gcd(h, g)
        if gcd ≠ 1 then isIrreducible ⟵ False
    end for
    Compute f ⟵ x^(p^d) − x mod g
    if isIrreducible = True AND f = 0 then return g
end for
```

What is the size of your input and output? Is your algorithm fast, or practical?

The size of input is $O(\log(d) + \log(p))$ which is the number of bits we need to input p and d
The size of output is $O(d \log(p))$ because we will need to return a degree d polynomial which has d coefficients each of which will require $\log(p)$ bits.
The algorithm we have given is fast in a sense that checking whether a given polynomial is irreducible is very efficient because of fast algorithm to find gcd and remainder.
However, we are still checking each polynomial of degree d to confirm whether it is irreducible or not makes the algorithm slow.
Though since we have proved this in lectures that the density of irreducible polynomials is similar to density of primes in integers, so we will never need to iterate over all the polynomials before encountering an irreducible.
However as the algorithm takes exponential time in terms of its input, we can not consider it as a fast algorithm because it is not practical to be used for finding irreducible polynomials with a very large degree d
But it is practical to use this for smaller input sizes.

2 [14+3 points]

Write a pseudocode to find a primitive element in a given finite field. Assume that, for prime p, the input is $\mathbb{F}_{p^d} := \mathbb{F}_p[x]/f$, for an irreducible polynomial $f(x)$ of degree d.

The idea is that we will have to iterate through all the polynomials in $\mathbb{F}_p[x]/f$ and check for each element if it is a primitive element until we find a primitive element.
To check if any polynomial in the given field is primitive we can exploit 2 theorems that we proved in lectures, first that the order of any element $g(x)$ in $\mathbb{F}_{p^d}$ is divisible by $p^d - 1$, and second that an element is primitive iff $ord(g) = p^d - 1$
Also we know that any exponent of any polynomial can be computed efficiently by binary exponentiation.
Using all the above results we arrive at the following pseudocode to find a primitive element in $\mathbb{F}_{p^d}$

Algorithm 2 Algorithm to find a primitive element in $\mathbb{F}_{p^d}$

```
find_primitive_element(p,d,f)
divisors ⟵ all the divisors of p^d − 1
remove 1, p^d − 1 from divisors
for all g in F_{p^d} do
    isPrimitive ⟵ True
    for each divisor m in divisors AND isPrimitive = True do
        Compute h = g^m using binary exponentiation
        if h = 1 then isPrimitive ⟵ False
    end for
    if isPrimitive = True then return g
end for
```

What is the size of your input and output? Is your algorithm fast, or practical?

The size of input is $O(d\log(p))$ as we need $\log(p)$ bits to input each of the d coefficients in $f(x)$
The size of output is also same because what we return is a primitive element which might also require roughly the same number of bits as input
The algorithm we have given is exponential time in d and p. Even though we have added various optimizations, the algorithms is still practical for only smaller values of d and p and is impractical for very large values.

Acknowledgements

1. Lecture Notes. Some theorems that I have used in the algorithms for increasing efficiency are not been proved here because these are picked from the lecture notes uploaded on course website.

2. Math.StackExchange. Inspiration has been taken for various methods that could be used for making a fast and more practical algorithm.