MA7010 – Number Theory for Cryptography - Assignment 2

Ajeesh Thattukunnel Vijayan

January 11th 2024

1 Answers

1. Lower Range = 600, Upper Range = 750. Consider all the numbers n in your range. Divide the set into two subsets: A - the subset consisting of all n where there is at least one primitive root modulo n; B – the subset consisting of all n where no primitive roots exist modulo n

Answer: The below json snippets show the two sets:

```
{
  "Numbers With Primitve Roots (A)": [
    "601","607","613","614","617","619","622","625",
    "626","631","634","641","643","647","653","659",
    "661","662","673","674","677","683","686","691",
    "694","698","701","706","709","718","719","722",
    "727","729","733","734","739","743","746"
  ]
}

{
  "Numbers Without Primitive Roots (B)": [
    "600","602","603","604","605","606","608","609",
    "610","611","612","615","616","618","620","621",
    "623","624","627","628","629","630","632","633",
    "635","636","637","638","639","640","642","644",
    "645","646","648","649","650","651","652","654",
    "655","656","657","658","660","663","664","665",
    "666","667","668","669","670","671","672","675",
    "676","678","679","680","681","682","684","685",
    "687","688","689","690","692","693","695","696",
    "697","699","700","702","703","704","705","707",
    "708","710","711","712","713","714","715","716",
    "717","720","721","723","724","725","726","728",
    "730","731","732","735","736","737","738","740",
    "741","742","744","745","747","748","749","750"
  ]
}

```

Listing 1: Miller-Rabin failues for numbers between 50 to 100

The Rust code for generating the above result is below:

```
///
/// Returns a vec of primitive roots for the integer
///
```

```
        /// # Arguments
        /// * n: BigInt
        ///
        /// Steps:
        /// This function uses trial and error to find primitive roots
        /// associated to an Integer
        ///
        /// 1. Find all coprime numbers less than `n`
        ///    (coprime_nums_less_than_n)
        /// 2. ϕ(n) = total number of coprimes
        /// 3. Find all the divisors of ϕ(n). Order of an
        ///    element in the Modulo n group will be equal to
        ///    any of the divisor values.
        /// 4. Find the order of each of the coprimes to n one by one
        ///    (skip 1 from the list of
        ///    coprimes as 1 is a trivial root) (`use utils::modular_pow)
        /// 5. if order of a coprime integer equals ϕ(n), that coprime
        ///    is a primitive root
        ///
        /// The above steps are executed aginst all coprimes to n and
        /// returns an integer vector with primitive roots
        ///
        pub fn primitive_roots_trial_n_error(n: &BigInt) -> Vec<BigInt> {
          let mut primitive_roots: Vec<BigInt> = Vec::new();
          let mut has_primitive_roots: bool = false;

          let nums_coprime_n: Vec<BigInt> = coprime_nums_less_than_n(n);
          let phi_n = BigInt::from(nums_coprime_n.len());
          //
          let divisors_phi_n = divisors_of_n(&phi_n);

          for a in nums_coprime_n {
            let mut has_order_phi: bool = true;
            for order in divisors_phi_n.iter() {
              if modular_pow(&a, order, n) == BigInt::one() {
                if *order != phi_n {
                  has_order_phi = false;
                }
              }
            }

            if has_order_phi {
              primitive_roots.push(a);
              has_primitive_roots = true;
              break;
            }
          }

          if has_primitive_roots {
            let orders_coprime_phi_n: Vec<BigInt> =
    coprime_nums_less_than_n(&phi_n);
            // first coprime number is 1 and we are skipping that
            // when calculating power
            for order in orders_coprime_phi_n.iter().skip(1) {
              primitive_roots.push(modular_pow(&primitive_roots[0], order,
     n));
            }
          }

          primitive_roots.sort();

          for (i, num) in primitive_roots.clone().iter().enumerate() {
```

```
            if num == &BigInt::one() {
              primitive_roots.remove(i);
              continue;
            }

            if modular_pow(num, &phi_n, n) != BigInt::one() {
              primitive_roots.remove(i);
            }
          }

          primitive_roots
        }

        ///
        /// Generates a list of integers less than n and co-prime to n.
        ///
        pub fn coprime_nums_less_than_n(n: &BigInt) -> Vec<BigInt> {
          let mut coprimes: Vec<BigInt> = Vec::new();
          let r = range(BigInt::from(1u64), n.clone());

          for num in r {
            if n.gcd_euclid(&num) == BigInt::one() {
              coprimes.push(num)
            }
          }
          coprimes.sort();
          coprimes
        }

        ///
        /// Get list of divisors of a number n > 2
        ///
        pub fn divisors_of_n(n: &BigInt) -> Vec<BigInt> {
          let mut divisors: Vec<BigInt> = Vec::new();
          let mut primes = vec![BigInt::from(2u64)];
          let p_factors_n = n.prime_factors(&mut primes);
          let p_factors_n = p_factors_n
          .iter()
          .map(|(p, _)| p.clone())
          .collect::<Vec<BigInt>>();

          for p in p_factors_n {
            let mut i = 0;
            loop {
              let pow = p.pow(i);
              if n % &pow == BigInt::zero() {
                divisors.push(n / &pow);
                divisors.push(pow);
                i += 1;
              } else {
                break;
              }
            }
          }
          divisors.sort();
          divisors.dedup();
          divisors
        }

```

Listing 2: Primitve Roots Calculation

2. a. Explain why we can always find a primitive root modulo p when p is a prime.

Answer: (Not complete)

Theorem 1 (Euler's Theorem) *Suppose that* $m \geq 1$ *and* $(a, m) = 1$*, then* $a^{\phi(m)} = 1($ mod $m)$*, where* $\phi(m)$ *is Euler's Totient function which yields the number of integers less than* m *and relatively prime to* m*.*

A special case occurs when m is a prime number, which is called Fermat's Little theorem. When m is a prime, the number of integers less than m and relatively prime to m equal $m - 1$. i.e., $\phi(m) = m - 1$.

b. Express the number of primitive roots that exist modulo p using the Euler Totient function and show that your answer correctly predicts the number of primitive roots for all primes in your given range.

Answer: The number of primitive roots associated with an integer n is given by $\phi(\phi(n))$ When n is a prime, namely p, $\phi(\phi(p)) = \phi(p - 1)$. The below table verifies this value against the number calculated using trial and error for all primes in the range $600 \leq p \leq 750$.

Prime	Primitive Roots Count - Trial and Error	$\phi(p-1)$
601	160	160
607	200	200
613	192	192
617	240	240
619	204	204
631	144	144
641	256	256
643	212	212
647	288	288
653	324	324
659	276	276
661	160	160
673	192	192
677	312	312
683	300	300
691	176	176
701	240	240
709	232	232
719	358	358
727	220	220
733	240	240
739	240	240
743	312	312

Table 1: Primitive Roots Count

The Rust code for generating the above result is below. It calls the function listed in code 2.

```
1
2                       PrimitiveRootsCommands::Ass2Question2b(r) => {
3           let start = r.start;
4           let end = r.end;
5
6           let mut result: Vec<HashMap<String, String>> = Vec::new();
7           let (primes_in_range, _) =
    find_primes_in_range_trial_division_parallel(start, end);
8           for p in primes_in_range.iter() {
```

```
            let primitive_roots = primitive_roots_trial_n_error(p);
            let phi_phi_n = euler_totient_phi(&(p - BigInt::one()));
            let mut item: HashMap<String, String> = HashMap::new();
            item.insert("Prime".to_string(), p.to_string());
            item.insert("Euler_Totient(p-1)".to_string(), phi_phi_n.
to_string());
            item.insert(
            "Prim Roots Count - Trial and Error".to_string(),
            primitive_roots.len().to_string(),
            );
            result.push(item);
          }
          println!("{}", serde_json::to_string_pretty(&result).unwrap
())
        }

```

Listing 3: Primitive Roots - Euler's Totient Function Verification

We can use the below command to see the above result:

```
.\target\release\nt-assignments.exe primitive-roots ass2-question2b -s 600 -e 750
```

Listing 4: Verify Primitive Roots Counting using Totient Function

c. For the same range as Question 1 use the command ifactors in Maple to find the set C whose elements consist of numbers of the form $p^k (p > 2, k \geq 1)$ or $2p^k (p > 2, k \geq 1)$

Number	Form	Number	Form
601	601^1	674	$2^1 \times 337^1$
607	607^1	677	677^1
613	613^1	683	683^1
614	$2^1 \times 307^1$	686	$2^1 \times 7^3$
617	617^1	691	691^1
619	619^1	694	$2^1 \times 347^1$
622	$2^1 \times 311^1$	698	$2^1 \times 349^1$
625	5^4	701	701^1
626	$2^1 \times 313^1$	706	$2^1 \times 353^1$
631	631^1	709	709^1
634	$2^1 \times 317^1$	718	$2^1 \times 359^1$
641	641^1	719	719^1
643	643^1	722	$2^1 \times 19^2$
647	647^1	727	727^1
653	653^1	729	3^6
659	659^1	733	733^1
661	661^1	734	$2^1 \times 367^1$
662	$2^1 \times 331^1$	739	739^1
673	673^1	743	743^1
746	$2^1 \times 373^1$	-	-

Table 2: Numbers of the form p^k, $2p^k$

d. Hence form a conjecture about when primitive roots do and don't exist

3. Suppose n has the form $n = pq$ where p and q are different primes both > 2.

(a) What is $\phi(n)$ in terms of p and q?

(b) Suppose a is relatively prime to pq. Explain why

i. $a^{p-1} \equiv 1 \mod p$

ii. $a^{q-1} \equiv 1 \mod q$

iii. $m = lcm(p-1, q-1)$ is less than $(p-1)(q-1)$

iv. $a^m \equiv 1 \mod (p-1)(q-1)$

(c) Hence explain why numbers of the form n have no primitive roots. check it out

(d) Show that all numbers of the form $n = pq$ (p and q both odd primes) in your range are included in set B.

4. Use the BabyStepsGiantSteps algorithm to find discrete logarithms x of b mod n for the primitive root a for each of the two examples assigned to you in the table below. Verify that your answer is correct by calculating $a^x \mod m$ by hand using the method of modular exponentiation.

5. Use the Pohlig Helmann algorithm to find in the cyclic group of order n with the generating element a for both the examples assigned to you below. Verify your answer in Maple.

6. Use the Pollard Rho method to verify your answer to the first example you were allocated in Question 4.

Name	b	n	a	Method
Ajeesh	47	71	21	BabyStepGiantStep
Ajeesh	24	53	26	BabyStepGiantStep
Ajeesh	x^{41}	343	x^{11}	Pohlig Hellmen
Ajeesh	x^{157}	3267	x^{13}	Pohlig Hellmen

Table 3: List of composite numbers of the form P.Q

References

[1] C R Jordan & D A Jordan *MODULAR MATHEMATICS Groups* .

[2] Dr. Ben Fairbairn *GROUP THEORY Solutions to Exercises.*

[3] *https://github.com/Ssophoclis/AKS-algorithm/blob/master/AKS.py*