Number Theory: Applications of Number Theory in Cryptography

SRIKAR R

SUBMITTED TO

Dr. Sudev N.K

CHRIST

(DEEMED TO BE UNIVERSITY)

BANGALORE · INDIA

B Sc. Economics, Mathematics, Statistics

Department of Mathematics

Deanery of Sciences

(April 2022)

Table of Contents

Introduction

Number theory is a branch of mathematics that deals with integers and is one of the oldest branches in mathematics. Number theory is said to be an infinite field and has always puzzled mathematicians and has kept the world of mathematics going with several unsolved theorems and conjectures. The best part of the field is that it can be understood by most people and does not deal with extreme mathematics that needs several pre-requisites to be understood.

It was after the late 20^{th} century when mathematicians discovered the applicability of number theory in the real world. Until then and most part of it is still considered to be one of the purest branches of mathematics. The development of computers and technology has made it possible to see the several applications of number theory in our fast-paced world. Several theorems and conjectures were only proved using the power of computers such as determining primes, finding factorials of large numbers and testing problems that could never be solved before.

The applications of number theory are mostly seen in cryptography, encryption, decryption and hashing. With the help of these basic mathematical algorithms, we have been able to build programs and software that ensure security of a user, encode messages and ensure privacy of information, generate random identification numbers to ensure safety and so on. We have also seen how concepts such as Euclid's division algorithm have made it possible to convert and interact with computers through binary system.

This paper mostly covers the applications of 5 different topics where the application of number theory is evidently seen. They are: 1. Barcodes 2. Pseudorandom numbers 3. Hashing functions 4. Parity check bits 5. Cryptography.

Application of Number Theory

1. Barcodes

A barcode is an image that consists of different lines in black that represents information usually digits, letters and symbols. One must use a barcode scanner that is specifically designed to read the barcode.

There are several systems of barcodes that use different methods to store information. The most famous barcode is the UPC code (Universal Product code). It is one of the oldest form of barcodes and was made by George J. Laurer in 1973.It is now mandatory for every product to have a UPC. Since the barcode stores all information about the product, each product usually has a unique barcode. The information is encoded in a numeric format such that it follows certain rules for one to create a barcode.

Most barcodes display a twelve-digit number, usually printed underneath as a backup for possible complications. Here are what the numbers represent:
First Number: Product Type. The product type is typically denoted by 0,1,6,7 or 8.
Following 5 Numbers: The Manufacturer Code. The five numbers are a unique code that identifies the manufacturer or distributor of the product.
Following 5 Numbers on the Right: Product Code. This part of the code is unique to the individual product.
Final Number: Check Digit (a Self-Policing System). The final digit of a barcode number is a computer check digit which makes sure the barcode is correctly composed.

Now for the computer to accept the barcode number in UPC format, it must follow the below algorithm:

1) Sum up the numbers in the odd position and multiply it by 3
2) Sum up the numbers in the even position and add this value to the value above4
3) If the sum is divisible by 10, it is accepted by the computer else it is rejected.

Consider the barcode: 0 36000 26150 9

Here the 12th digit (9) is the check digit and helps prevent human made errors when the UPC is keyed in by hand. Now, when we sum the odd places before the check digit and multiply it by 3, we get 50 which is divisible by 10 and hence accepted by the computer. We can create a program to check whether the barcode is accepted or not by the computer:

```
def barcode(n):
    sumOdd = 0
    sumEven = 0

    num = str(n)

    for i in range(len(num)):
        if(i % 2 == 0):
            sumOdd = sumOdd+int(num[i])
        else:
            sumEven = sumEven+int(num[i])
    if (sumOdd+sumEven*3)%10==0:
        print("Accepted by computer")
    else:
        print("Not accepted by computer")
```

We define a function called barcode and initialize two variables called sumOdd and sumEven in order to calculate the values of odd-positioned numbers and even-positioned numbers. In order to do so, we iterate through the index positions of the number to check whether it is odd or even.

If it is evenly positioned it must be divisible by 2 else it is odd positioned. We usually multiply 3 with the odd-positioned values but since python does not accept 0 as the leading value, we are forced to multiply with the even places. If the sum of this value is divisible by 10, it is then accepted by the computer.

```
barcode(36000261509)
```

```
Accepted by computer
```

We observe that this barcode number is accepted by the computer when we sum the odd places before the check digit and multiply it by 3, we get 50 which is divisible by 10.

2. Pseudorandom numbers

Pseudo-random numbers are a set of values that are randomly generated knowing the starting point or the end point and is typically repeated over and over above or below a certain value. Since it is repeated over and over, the process tends to repeat certain sequences and hence it known as pseudorandom number.

Pseudorandom numbers are essential to computer applications such as video games and cyber-security. For example, in gaming, there could be random obstacles or objects that are generated from time and again in certain levels of the game or for example we could generate random cards in games that include playing with cards.

In computer security, pseudo randomness is important in encryption algorithms, which create codes that must not be predicted or guessed. There are pseudorandom number generators (PRNG) which are programmes or functions which uses concepts of math to generate randomness.

The process to generate a pseudorandom number are just:

1) Set a seed or a starting point and generate random numbers.
2) If the seed value changes, the generated numbers also change. Since a single seed value could always generate a repeated number, there is no real true randomness

Consider a random card generator which generates any number of cards that the user wants. We create a list that contains the four suits.

```
import random as random

C=['Heart','Diamond','Clubs','Spade']
```

We then set an initial value and an end value for numbers to be generated from (1-13) where 1 represents Ace and 11,12 and 13 represent Jack, Queen and King respectively.

```
#1

n=abs(int(input("Enter how many random cards you want to draw")))
M=[]
for i in range(1,n+1):
    A={random.choice(C):[random.randrange(1,13)]}
    M.append(A)
print(M)
```

```
Enter how many random cards you want to draw5
[{'Clubs': [2]}, {'Diamond': [3]}, {'Diamond': [6]}, {'Diamond': [10]}, {'Spade': [8]}]
```

Here we generate 5 random cards which are randomly suit list.
We then run the same code again and get the following output:

```
#2

n=abs(int(input("Enter how many random cards you want to draw")))
M=[]
for i in range(1,n+1):
    A={random.choice(C):[random.randrange(1,13)]}
    M.append(A)
print(M)
```

```
Enter how many random cards you want to draw5
[{'Clubs': [9]}, {'Diamond': [6]}, {'Clubs': [8]}, {'Diamond': [1]}, {'Spade': [2]}]
```

We observe that 6 of Diamonds is repeated again which confirms that they are pseudo-random numbers.

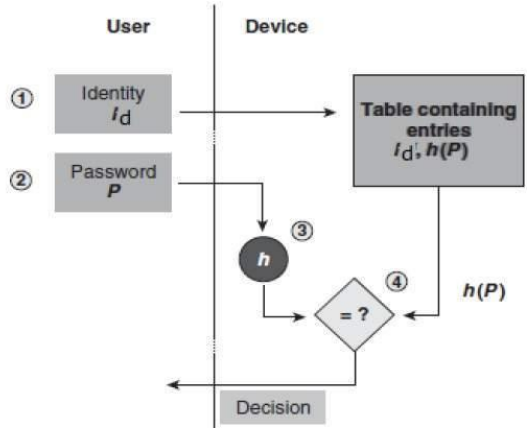
3. Hashing functions

A hash function uses a group of characters known as a key and maps it to a certain value which is usually smaller than the original value. Hashing is usually used for indexing and locating items in databases because it is easier to find the shorter hash value than the longer string.

For example, big phone numbers are usually converted into small integer values while storing in a database. The mapped integer is used as an index in the hash table.

Usually a good hash table, will have a table position equally likely for each key which means that it will not repeat values most often. For example, taking the first three digits of a phone number is a bad hash function as most numbers will start from them whereas taking the last three digits is a ‘better’ way to hash it.

The main application of hash function is password storage. Hash functions provide protection to password storage. Instead of storing password in clear, mostly all logon processes store the hash values of passwords in the file. The process of login is depicted in the following illustration:



MD5 (message-digest algorithm) was one of the most famous hash functions that was used. MD5 is based on a hash function that verifies that a file you sent matches the file received by the person you sent it to. Previously, MD5 was used for data encryption, but now it's used primarily for authentication.

MD5 runs entire files through a complex mathematical function to create a hash that can be matched with an original file. The MD5 hashing algorithm converts data into a string of 32 characters. For example, the word "frog" always generates this hash: 938c2cc0dcc05f2b68c4287040cfcf71.

An MD5 hash is 16 bytes. Each MD5 hash looks like 32 numbers and letters, but each digit is in hexadecimal and represents four bits. Since a single character represents eight bits (to form a byte), the total bit count of an MD5 hash is 128 bits. Two hexadecimal characters form a byte, so 32 hexadecimal characters equal 16 bytes.

In order to check create a program that converts information into MD5, we use the hashlib library to do so.

We take the value from the user and convert it into a MD5 hash value.

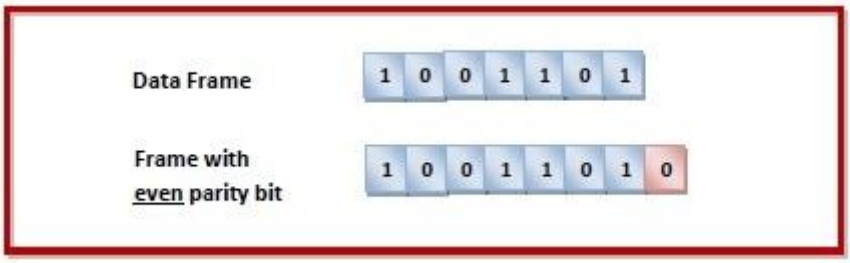
```
import hashlib

result = hashlib.md5(b'Number theory')


print("The byte equivalent of hash is : ", end ="")
print(result.digest())
```

```
The byte equivalent of hash is : b'\x18\xc5\xc2\xc2\xea\xe0\x15&\xd6\xd3^\x15\xc6\x87q\xa2'
```

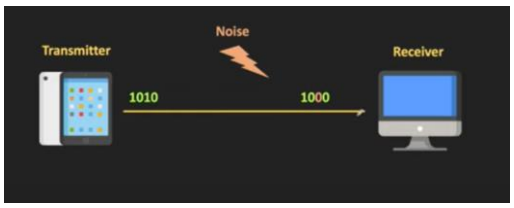
-

4. Parity Check Bits

A parity bit, or check bit, is a bit added to a string of binary code. Parity bits are a simple form of error detecting code. Accordingly, there are two variants of parity bits: even parity bit and odd parity bit. In the case of even parity, for a given set of bits, the occurrences of bits whose value is 1 are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1s in the whole set (including the parity bit) an even number. For example, consider the number below:



Suppose, some information is sent from the transmitter to the receiver in binary form and one one's (highlighted below) gets changed to a zero which completely changes the information.



By adding a parity bit (which cannot change), it bit ensures that the total number of 1-bits in the string is even or odd. If the information while sending had even number of 1's but the parity bit shows a 0, this means that there is an error and it needs to be corrected

Although parity bit is a great way to detect code, it has its limitations. For example, if two of the values get changed, then the parity bit is of no use as the number of ones could increase or decrease. We can observe from the below picture that the two of the 0's have been changed to 1 but there is no error detected since the count of one's still remains even.

Consider the below barcode that is inputted by the user. We then iterate through the binary number and count the number of ones in it and append it to a list. Then we check if there are odd or even number of ones in that number. If there are even, we add a 0 at the end of the number else we add a one.

```
n=int(input("Enter in binary form: "))
n1=str(n)
ones=[]
for i in n1:
    if i=='1':
        ones.append(i)
if len(ones)%2==0:
    n1=n1+'0'
    print("its an even parity")
else:
    n1=n1+'1'
    print("its an odd parity")
```

```
Enter in binary form: 1110001
its an even parity
```

Here the number is an even parity as there are even number of ones in the number.

Say, there was an error and one of the zeros got changed to a one. In order to detect the error, we count the number of ones and check if it matches with the parity bit. If there is an error, we display the error message or else we say that there is no error.

```
n_1=str(1110011)
print(n_1)
T=[]
for j in n_1:
    if j=='1':
        T.append(i)
if n1[-1]=='1':
    if len(T)%2!=0:
        print("there is an error")
    else:
        print("No error")
if n1[-1]=='0':
    if len(T)%2==0:
        print("No error")
    else:
        print("there is an error")
```

```
1110011
there is an error
```

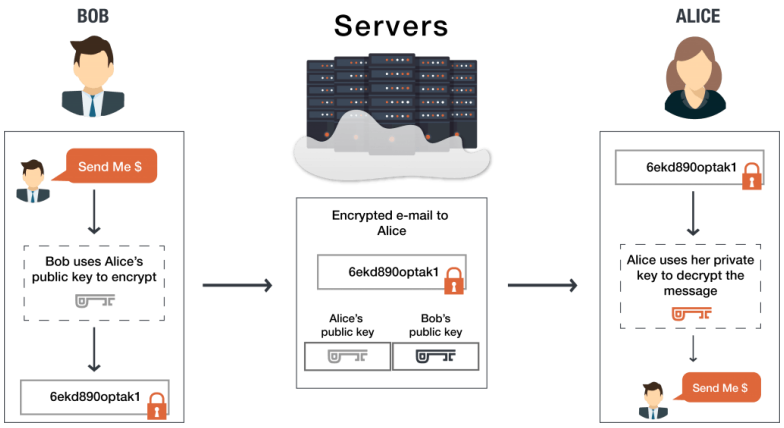
5. Cryptography

Cryptography is the study of secure communications techniques that allow only the sender and intended recipient of a message to view its contents. The term is derived from the Greek word kryptos, which means hidden. It is closely associated to encryption, which is the act of scrambling ordinary text into what's known as ciphertext and then back again upon arrival.

When transmitting electronic data, the most common use of cryptography is to encrypt and decrypt email and other plain-text messages. The simplest method uses the symmetric or "secret key" system. Here, data is encrypted using a secret key, and then both the encoded message and secret key are sent to the recipient for decryption. Every user has two keys: one public and one private. Senders request the public key of their intended recipient, encrypt the message and send it along. When the message arrives, only the recipient's private key will decode it — meaning theft is of no use without the corresponding private key.

An End-to-end encryption provides the gold-standard for protecting communication. In an end-to-end encrypted system, the only people who can access the data are the sender and the intended recipients. In true end-to-end encryption, encryption occurs at the device level. That is, messages and files are encrypted before they leave the phone or computer and isn't decrypted until it reaches its destination.

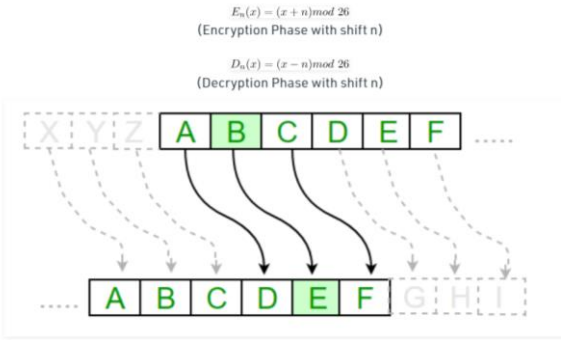
Let's say Alice and Bob create accounts on the system. The end-to-end encrypted system provides each with a public-private key pair, whereby their public keys are stored on the server and their private keys are stored on their device. Alice wants to send Bob an encrypted message. She uses Bob's public key to encrypt her message to him. Then, when Bob receives the message, he uses his private key on his device to decrypt the message from Alice.



When Bob wants to reply, he simply repeats the process, encrypting his message to Alice using Alice's public key description of end-to-end encryption.

Now illustrating the above example using a simple cryptography method knows as Caesar Cipher. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet.

For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials. Thus, to cipher a given text we need an integer value, known as shift which indicates the number of positions each letter of the text has been moved down. The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1, Z = 25. Encryption of a letter by a shift n can be described mathematically as.



The steps to create a Caesar's cipher are:
1) Traverse the given text one character at a time.
2) For each character, transform the given character as per the rule, depending on whether we're encrypting or decrypting the text.
3) Return the new string generated

To decrypt, we must write another function decrypt similar to encrypt, that'll apply the given shift in the opposite direction to decrypt the original text.

Encryption

```
def encrypt(word,k):
    answer=""

    for i in range(len(word)):
        char=word[i]

        if char.isupper():
            answer=answer+chr((ord(char) + k-65) % 26 + 65)
        else:
            answer=answer+chr((ord(char) + k- 97) % 26 + 97)
    return answer
```

```
encrypt("NUMBERTHEORY",4)
```

```
'RYQFIVXLISVC'
```

We find the order of each letter in the given word and shift to the corresponding letter. Here k is the number of letters we have to skip to get the desired letter. We return the letter in uppercase if the original letter was in uppercase else, we return it as lower case.

Similarly, we move in the opposite direction in order to decode.

Decryption

```
def decrypt(word,k):
    answer=""

    for i in range(len(word)):
        char=word[i]

        if char.isupper():
            answer=answer+chr((ord(char) - k-65) % 26 + 65)
        else:
            answer=answer+chr((ord(char) - k- 97) % 26 + 97)
    return answer
```

```
decrypt("RYQFIVXLISVC",4)
```

```
'NUMBERTHEORY'
```

Conclusion

We can see that the world of number theory is applied almost everywhere especially in the field of technology. From the above examples, it can be observed that number theory is used mostly in cyber-security and error detection. In the current tech world, where we are very dependent on our smart-devices for almost everything from personal to financial, privacy of an individual or organization is always at threat. With the help of number theory, we are able to solve modern problems which almost seemed like a struggle during the age of the analogue. Number theory which was once accepted as a pure field has several applications in the real world. This only suggests that there is a lot we don't know in the world of mathematics and how it can be applied in reality.

Reference

[1] https://www.gcsu.edu/sites/default/files/documents/2021-06/shores.pdf

[2] https://www.barcoding.co.uk/how-do-barcodes-work/#:~:text=How%20Do%20Barcodes%20Work%20in,absorbed%20while%20the%20rest%20reflects.

[3] https://prezi.com/6qbj5jzoh9tp/the-math-behind-bar-codes/

[4] https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/

[5] https://www.geeksforgeeks.org/md5-hash-python/?ref=rp

[6] https://blogs.ucl.ac.uk/infosec/2017/03/12/applications-of-cryptography/