IDENTIFICATION CODES AND RELATED ALGORITHMS

ABSTRACT

The report covers the topics like why identification codes exist, Importance of concepts like checksum and 5 of the well-known algorithms which play a vital role in the word of data.

Aditya Trivedi (N105)
Kaiwan Vaghchhipawala (N108)
Dev Vora (N110)
Aditya Singh (N115)

INDEX

Identification Codes

What is the purpose of verification code?

A verification code is a security protection method used by form owners to avoid Internet robots from abusing and spamming their forms. There are different verification code types, but the most commonly used is CAPTCHA. This generates a random code within an image next to a text box.

Code Verification:

Code verification is the process used for checking the software code for errors introduced in the coding phase. The objective of code verification process is to check the software code in all aspects. This process includes checking the consistency of user requirements with the design phase. Note that code verification process does not concentrate on proving the correctness of programs. Instead, it verifies whether the software code has been translated according to the requirements of the user.

Techniques:

The code verification techniques are classified into two categories, namely, dynamic and static.

1. The dynamic technique is performed by executing some test data. The outputs of the program are tested to find errors in the software code. This technique follows the conventional approach for testing the software code.
2. The static technique, the program is executed conceptually and without any data. In other words, the static technique does not use any traditional approach as used in the dynamic technique. Some of the commonly used static techniques are code reading, static analysis, symbolic execution, and code inspection and reviews.

Purpose:

The purpose of the verification code is to ensure the security and identity of the users. If not for a verification code, you have no idea how many people had tried to use your identity and even some of them would have been successful in doing so.

Checksum

A check sum or check digit is added to the identification number (usually the last digit). This digit is used to verify the identification number for its legitimacy. Check sum is added to the number to detect any errors made while typing the number into the system. The check sum is calculated with an algorithm. Some most common algorithms are mod 9, mod 10, and mod 11. mod10 algorithm is the most common and is used in most of the identification numbers.

Working:

Checksums work by giving the party on the receiving end information about the transmission to make sure that the full range of data is fully delivered. The checksum value itself is typically a long string of letters and numbers that act as a sort of fingerprint for a file or set of files to indicate the number of bits included in the transmission.

If the checksum value calculated by the end user is even slightly different from the original checksum value of the file, it can alert all parties in the transmission that the file was corrupted or tampered with by a third party. From there, the receiver can investigate what went wrong or try re-downloading the file.

What can cause an inconsistent checksum number:

1. While checksum values that do not match can signal something went wrong during transmission, a few factors can cause this to happen, such as:
2. An interruption in the internet or network connection.
3. Storage or space issues including problems with the hard drive.
4. A corrupted disk or corrupted file.
5. A third party interfering with the transfer of data.

Advantage:

The checksum detects all the errors involving an odd number of bits as well as the error involving an even number of bits.

Disadvantage:

The main problem is that the error goes undetected if one or more bits of a subunit is damaged and the corresponding bit or bits of a subunit are damaged and the corresponding bit or bits of opposite value in second subunit are also damaged. This is because the sum of those columns remains unchanged.

Common types of checksum algorithms:

There are multiple cryptographic hash functions that programmers can use to generate checksum values. A few common ones include:

SHA-0: This hash function was the first of its kind and it was withdrawn shortly after its creation in 1993.

SHA-1: This hash function was no longer considered secure as of 2010.

SHA-2 (224, 256, 384, 512): This family of hash functions relies on sounds and numbers to create a checksum value. The resultant checksums are vulnerable to length extension attacks, which involve a hacker reconstructing the internal state of a file by learning its hash digest.

MD5: This hash function creates a checksum value, but each file will not necessarily have a unique number, so it is open to vulnerabilities if a hacker swaps out a file with the same checksum value.

ISBN Verification Algorithm

On the back of your books, you’ve probably seen a number above the barcode labelled “ISBN” (International Standard Book Number) This is a unique number used by publishers, libraries, and bookstores to identify book titles and editions. The number is less useful to the average book reader, but we can all learn something about a book from the ISBN.

Using ISBN Code:

• Find the ISBN code: The ISBN should also be available on the copyright page. Books published before 2007 were given 10 digit ISBNs. From 2007 on, they have been given 13-digit identifiers.

• Determine the publisher: One of the most interesting things you can learn about a book with the ISBN is the publisher’s scale of operations. 10 and 13-digit ISBNs have their own ways of identifying the publisher and the title. If the publisher identifier is long, but the title number is only one or two digits, the publisher only plans on releasing a handful of books and the book might even be self-published. Conversely, if the title string is long and the publisher string is short, the book was released by a major publisher.

• Use an ISBN to self-publish: If you plan to sell your manuscript in bookstores, it needs an ISBN, even if you are publishing it yourself. You can purchase an ISBN number at ISBN.org. You will need to purchase an ISBN number for each title you plan to publish and for different editions of the title, including hardback and paperback versions. The more ISBN numbers you purchase at time, the cheaper it will be. Each nation has its own ISBN granting corporation

Step to verify:

Step 1: Find out what kind of ISBN you have. Two kinds of ISBN is now used for a book, 10 or 13 digits. Usually, a 13 digits ISBN is converted from 10 digits one by adding a "978" prefix and alter the last digit, which is often called the check digit.

Step 2 (10 Digit ISBN):

1. In case you have a 10 digits ISBN, for example, the ISBN for the book Gravitation. 0-7167-0344-0. You need to assign the position of each number from left to right.
2. Multiply each number by its position number and then sum up the products.
3. Divide the sum by 11 and find out what is the remainder.
   If the remainder is zero, then it is a valid 10 digit ISBN.
   If the remainder is not zero, then it is not a valid 10 digit ISBN.

Step 3 (13 Digit ISBN):

1. If you have a 13 digits ISBN. 978-0-7167-0344-0. Also assign the position of each number from right to left
2. Multiply each number by an alternating 1 and 3 and then sum up the products.
   Odd number positions by 1.
   Even number positions by 3.
3. Divide the sum by 10 and find out what is the remainder.
   If the remainder is zero, then it is a valid 13-digit ISBN.
   If the remainder is not zero, then it is not a valid 13-digit ISBN.

Code for 10-Digit ISBN:

- **Python Code:**

```
def isValidISBN(isbn):
    if len(isbn) != 10:
        return False

    sum = 0
    for i in range(9):
        sum += int(isbn[i]) * (10 - i)

    if(isbn[9] == 'X'):
        sum+=10
    else:
        sum+=int(isbn[9])*10

    return (sum % 11 == 0)

code = input("Enter the code to be verified: ")
if isValidISBN(code):
    print('Valid')
else:
    print("Invalid")
```

- **Output for the Python code:**

```
D:\DM Python Codes>python "ISBN code
verification.py"
Enter the code to be verified: 007462542X
Valid
```

- **C++ Code:**

```
#include <iostream>
using namespace std;

bool isValidISBN(string isbn)
{
    if (isbn.length() != 10)
        return false;
    int sum = 0;
    for (int i = 0; i < 9; i++)
    {
        int digit = isbn[i] - '0';
        if (digit < 0 || digit > 9)
            return false;
        sum += (digit * (10 - i));
    }
    char end = isbn[9];
    if (end != 'X' && (end < '0' || end > '9'))
        return false;

    if (end == 'X')
        sum += 10;
    else
        sum += (end - '0');
    return (sum % 11 == 0);
}

int main()
{
    string isbn;
    cout << "Enter ISBN Number:";
    cin >> isbn;
    if (isValidISBN(isbn))
        cout << "Valid";
    else
        cout << "Invalid";
    return 0;
}
```

- **Output for the C++ Code:**

```
Enter ISBN Number:007462542X
Valid

...Program finished with exit code 0
Press ENTER to exit console.
```

- **Java Code:**

```
import java.util.*;
public class ISBN
{
    Scanner ob=new Scanner(System.in);
    public int isvalidisbn(String isbn)
    {
        int len=isbn.length();
        if(len!=10)
            return 0;
        double sum=0,digit=0;
        for(int i=0;i<9;i++)
        {
            digit=isbn.charAt(i)-'0';
            if(digit<0 || digit>9)
                return 0;
            sum+=(digit*(10-i));
        }
        char end=isbn.charAt(9);
        if(end!='X' && (end<0 || end>9))
            return 0;
        if(end=='X')
            sum+=10;
        else
            sum+=(end-'0');
        return((sum%11==0)?1:0);
    }
    public void main()
    {
        String isbn;
        System.out.println("Enter ISBN Code :");
        isbn=ob.nextLine();
        if(isvalidisbn(isbn)==1)
            System.out.println("Valid");
        else
            System.out.println("Invalid");
    }
}
```

- **Output for the Java code:**

```
Enter ISBN Code :
007462542X
Valid
```

Luhn's Algorithm

The Luhn algorithm, also known as the modulus 10 or mod 10 algorithm, is a simple checksum formula used to validate a variety of identification numbers, such as credit card numbers, IMEI numbers, Canadian Social Insurance Numbers. The LUHN formula was created in the late 1960s by a group of mathematicians. Shortly thereafter, credit card companies adopted it. Because the algorithm is in the public domain, it can be used by anyone. Most credit cards and many government identification numbers use the algorithm as a simple method of distinguishing valid numbers from mistyped or otherwise incorrect numbers. It was designed to protect against accidental errors, not malicious attacks.

Steps involved in the Luhn algorithm

Let's understand the algorithm with an example: Consider the example of an account number "79927398713".

Step 1: Starting from the rightmost digit, double the value of every second digit,

Step 2: If doubling of a number results in a two digit number i.e greater than 9(e.g., 6 × 2 = 12), then add the digits of the product (e.g., 12: 1 + 2 = 3, 15: 1 + 5 = 6), to get a single digit number.

Step 3: Now take the sum of all the digits. Step 4 – If the total modulo 10 is equal to 0 (if the total ends in zero) then the number is valid according to the Luhn formula; else it is not valid

Code for Luhn’s Algorithm in order to verify Credit card number:

- **Python Code:**

```
def digitSum(num):
    sum=0
    digit=0
    for i in range(16):
        digit=num%10
        num=num//10
        if(i%2==0):
            sum=sum+digit
        else:
            if((digit*2)>8):
                sum+=(digit*2)%10+(digit*2)//10
            else:
                sum+=(digit*2)
    return sum

def checkCard(CardNo):
    if(len(CardNo)!=16):
        print(2)
        return False
    CardNo=int(CardNo)
    if(digitSum(CardNo)%10==0):
        return True
    else:
        print(digitSum(CardNo))
        return False

code=input("Enter Credit Card Code: ")
if (checkCard(code)):
    print("This is a Valid Card.")
else:
    print("This is an Invalid Card.")
```

- **Output for the Python code:**

```
D:\DM Python Codes>python "Luhn Algorithm Credit
Card.py"
Enter Credit Card Code: 2222405343248877
This is a Valid Card.
```

- **C++ Code:**

```
#include <iostream>
using namespace std;
int digitSum(long num)
{
    int sum = 0, digit = 0;
    for (int i = 0; i < 16; i++)
    {
        digit = num % 10;
        num = num / 10;
        if (i % 2 == 0)
            sum += digit;
        else
        {
            if (digit * 2 > 8)
                sum += ((digit * 2) % 10) +
((digit * 2) / 10);
            else
                sum += (digit * 2);
        }
    }
    return sum;
}

int checkCard(long CardNo)
{
    string Num = to_string(CardNo);
    if (Num.length() != 16)
        return false;
    if (digitSum(CardNo) % 10 == 0)
        return true;
    else
    {
        cout << digitSum(CardNo) << endl;
        return false;
    }
}

int main()
{
    long code;
    cout << "Enter Credit card code:";
    cin >> code;
    if (checkCard(code))
    {
        cout << "This is a Valid Card.";
    }
    else
    {
        cout << "This is an Invalid Card.";
    }
    return 0;
}
```

- **Output for the C++ code:**

```
Enter Credit card code:2222405343248877
This is a Valid Card.

...Program finished with exit code 0
Press ENTER to exit console.
```

- **Java Code:**

```
import java.util.*;
class LuhnAlgo
{
    Scanner ob=new Scanner(System.in);
    public long digitsum(long num)
    {
        long sum=0,digit=0;
        for(int i=0;i<16;i++)
        {
            digit=num%10;
            num=(num/10);
            if(i%2==0)
                sum=sum+digit;
            else
            {
                if(digit*2>8)

sum+=((digit*2)%10)+((digit*2)/10);
                else
                    sum=sum+(digit*2);
            }
        }
        return sum;
    }
    public int checkcard(long cardno)
    {
        int len=String.valueOf(cardno).length();
        if(len!=16)
            return 0;
        if(digitsum(cardno)%10==0)
            return 1;
        else
        {
            System.out.println(digitsum(cardno));
            return 0;
        }
    }
    public void main()
    {
        System.out.println("Enter credit card
code : ");
```

```
        long code=ob.nextLong();
        if(checkcard(code)==1)
            System.out.println("This is a valid 
card");
        else
            System.out.println("This is an 
invalid card");
    }
}
```

- **Output for the Java code:**

```
Enter credit card code :
2222405343248877
This is a valid card
```

UPC Verification Algorithm

Perhaps the most well-known, or if not well-known, the most used identification would be the Universal Product Codes, or UPCs. Most of what is sold in stores utilize a UPC, which is the number located either above or below the bar code of the product. This identification number is only 12 numbers, and with the abundance of products in stores, there is a higher potential for collision to occur.

Grocery products use the so called UPC system for identifying products. A UPC number consists of 12 digits: $\mathbf{a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}a_{11}a_{12}}$**.**

The first digit a1 identifies the type of product, the next five digits $\mathbf{a_2a_3a_4a_5a_6}$ identify the manufacturer, and the next five digits $\mathbf{a_7a_8a_9a_{10}a_{11}}$ identify the product.

The last digit a12 is the check digit, and it must satisfy:

$$\mathbf{a_1 + a_2 + 3 \cdot a_3 + a_4 + 3 \cdot a_5 + a_6 + 3 \cdot a_7 + a_8 + 3 \cdot a_9 + a_{10} + 3 \cdot a_{11} + a_{12} \equiv 0 \pmod{10}}$$

Code for UPC Verification:

- **Python Code:**

```
def isValidUPC(upc):
    if(len(upc) != 12):
        return False
    upc = int(upc)
    sum = 0
    digit = 0
    num = upc
    for i in range(13):
        digit = num % 10
        if(i % 2 == 0):
            sum = sum+digit*1
        else:
            sum = sum+digit*3
        num = num//10
    if(sum % 10 == upc % 10):
        return True
    else:
        return False


code = input("Enter the code to be verified: ")
if isValidUPC(code):
    print('Valid')
else:
    print("Invalid")
```

- **Output for the Python code:**

```
D:\DM Python Codes>python "UPC code
verification.py"
Enter the code to be verified: 012345543210
Valid
```

- **C++ Code:**

```
#include <iostream>
using namespace std;

int isValidupc(long upc)
{
    int sum = 0, digit = 0;
    long num = upc;

    for (int i = 0; i < 13; i++)
    {
        digit = num % 10;
        if (i % 2 == 0)
            sum += digit * 1;
        else
            sum += digit * 3;
        num = num / 10;
    }
    if (sum % 10 == upc % 10)
        return true;
    else
        return false;
}
int main()
{
    long code;
    cout << "Enter the upc code to be verified: 
";
    cin >> code;
    if (isValidupc(code))
        cout << "Valid";
    else
        cout << "Invalid";
    return 0;
}
```

- **Output for the C++ code:**

```
Enter the upc code to be verified: 012345543210
Valid

...Program finished with exit code 0
Press ENTER to exit console.
```

- **Java Code:**

```
import java.util.*;
public class UPC
{
    Scanner ob=new Scanner(System.in);
    public int isvalidupc(double upc)
    {
        int len=String.valueOf(upc).length();
        if(len!=12)
            return 1;
        double sum=0,digit=0,num=upc;
        for (int i=0;i<13;i++)
        {
            digit=num%10;
            if(i%2==0)
                sum+=digit*1;
            else
                sum+=digit*3;
            num=num/10;
        }
        if(sum%10==upc%10)
            return 1;
        else
            return 0;
    }
    public void main()
    {
        System.out.println("Enter the ups code to 
be verified : ");
        double code=ob.nextDouble();
        if(isvalidupc(code)==1)
            System.out.println("Valid");
        else
            System.out.println("Invalid");
    }
}
```

- **Output for the Java code:**

```
Enter the ups code to be verified :
012345543210
Valid
```

US Postal Money Orders

The Unites States postal uses an identification number system for their postal money orders. These numbers are designed as an 11-digit identification number, which allows for several different combinations to be created.

Thus, if the printed number on the money order is:

$\mathbf{a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}a_{11}}$, then

$\mathbf{a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 + a_9 + a_{10} = a_{11} \pmod 9}$

For example,

The money order on the left has identification number 8431032502 and check digit 1. The sum **8 + 4 + 3 + 1 + 0 + 3 + 2 + 5 + 0 + 2 = 28** and **28 = 3 × 9 + 1** has remainder 1 after dividing by 9. The check digit is 1.

Code for USPMO Verification:

- **Python Code:**

```
def digitSum(num):
    digit = 0
    sum = 0
    num = num//10
    for i in range(10):
        digit = num % 10
        num = num//10
        sum = sum+digit
    return sum

def isValidUSPMO(uspmo):
    if (len(uspmo) != 11):
        return False
    uspmo = int(uspmo)
    if (digitSum(uspmo) % 9 == uspmo % 10):
        return True
    else:
        return False

def errorRectifier(num):
    num = int(num)
    print("Possible Valid Code can be (Only if
the Last Digit is entered incorrectly):",
          (num//10)*10 + (digitSum(num) % 9))

code = input("Enter a US Postal Money Order Code:
")
if isValidUSPMO(code):
    print("Valid")
else:
    print("Invalid ")
    errorRectifier(code)
```

- **Output for the Python code:**

```
D:\DM Python Codes>python "USPMO.py"
Enter a US Postal Money Order Code: 58312044178
Valid
```

- **C++ Code:**

```
#include <iostream>
using namespace std;

int digitSum(long num)
{
    int digit = 0, sum = 0;
    num = num / 10;
    for (int i = 0; i < 10; i++)
    {
        digit = num % 10;
        num = num / 10;
        sum += digit;
    }
    return sum;
}

int isValidUSPMO(long uspmo)
{
    string uspmo_str = to_string(uspmo);
    if (uspmo_str.length() != 11)
        return false;
    if (digitSum(uspmo) % 9 == uspmo % 10)
        return true;
    else
        return false;
}

int errorRectifier(long num)
{
    cout << "Possible Valid Code can be (Only if
the Last Digit is entered incorrectly): "
         << (num / 10) * 10 + (digitSum(num) % 9)
<< endl;
    return 0;
}

int main()
{
    long code;
    cout << "Enter a US Postal Money Order Code:
";
    cin >> code;
    if (isValidUSPMO(code))
        cout << "Valid" << endl;
    else
    {
        cout << "Invalid" << endl;
        errorRectifier(code);
    }
    return 0;
}
```

- **Output for the C++ code:**

```
Enter a US Postal Money Order Code: 58312044178
Valid


...Program finished with exit code 0
Press ENTER to exit console.
```

- **Java Code:**

```
import java.util.*;
public class USPMO
{
    Scanner ob=new Scanner(System.in);
    public long digitsum(long num)
    {
        long sum=0,digit=0;
        num=num/10;
        for(int i=0;i<10;i++)
        {
            digit=num%10;
            num=num/10;
            sum+=digit;
        }
        return sum;
    }

    public int isvaliduspmo(long uspmo)
    {
        int len=String.valueOf(uspmo).length();
        if(len!=11)
            return 0;
        if(digitsum(uspmo)%9==uspmo%10)
            return 1;
        else
            return 0;
    }

    public void main()
    {
        System.out.println("Enter a US postal money
order code : ");
        long code=ob.nextLong();
        if(isvaliduspmo(code)==1)
            System.out.println("Valid code");
        else
        {
            System.out.println("Invalid code");
            System.out.println("Possible valid code
can be (Only if the last digit is entered
incorrectly) :
"+(((code/10)*10)+(digitsum(code)%9))+"\n");
        }
    }
}
```

- **Output for the Java code:**

```
Enter a US postal money order code :
58312044178
Valid code
```

Verhoeff's Algorithm

The Verhoeff algorithm is a checksum formula for error detection developed by the Dutch mathematician Jacobus Verhoeff and first published in 1969. It was the first decimal check digit algorithm which detects all single-digit errors, and all transposition errors involving two adjacent digits, which was at the time thought impossible with such a code.

Steps:

- Write routines, methods, procedures etc. in your language to generate Verhoeff checksum digit for non-negative integers of any length and to validate the result. A combined routine is also acceptable.
- The more mathematically minded may prefer to generate the 3 tables required from the description provided rather than to hard-code them.
- Write your routines in such a way that they can optionally display digit by digit calculations as in the Wikipedia example.
- Use your routines to calculate check digits for the integers: 236, 12345 and 123456789012 and then validate them. Also attempt to validate the same integers if the check digits in all cases were 9 rather than what they actually are.
- Display digit by digit calculations for the first two integers but not for the third.

Code for verification of Aadhar card number using Verhoeff’s Algorithm:

- **Python Code:**

```
d = [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
     [1, 2, 3, 4, 0, 6, 7, 8, 9, 5],
     [2, 3, 4, 0, 1, 7, 8, 9, 5, 6],
     [3, 4, 0, 1, 2, 8, 9, 5, 6, 7],
     [4, 0, 1, 2, 3, 9, 5, 6, 7, 8],
     [5, 9, 8, 7, 6, 0, 4, 3, 2, 1],
     [6, 5, 9, 8, 7, 1, 0, 4, 3, 2],
     [7, 6, 5, 9, 8, 2, 1, 0, 4, 3],
     [8, 7, 6, 5, 9, 3, 2, 1, 0, 4],
     [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]]

p = [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
     [1, 5, 7, 6, 2, 8, 3, 0, 9, 4],
     [5, 8, 0, 3, 7, 9, 6, 1, 4, 2],
     [8, 9, 1, 6, 0, 4, 3, 5, 2, 7],
     [9, 4, 5, 3, 1, 2, 6, 8, 7, 0],
     [4, 2, 8, 6, 5, 7, 3, 9, 0, 1],
     [2, 7, 9, 3, 8, 0, 6, 4, 1, 5],
     [7, 0, 4, 6, 9, 1, 3, 2, 5, 8]]

inv = [0, 4, 3, 2, 1, 5, 6, 7, 8, 9]


def isValidAadhar(aadhar_num):

    if len(aadhar_num) != 12:
        return False

    aadhar_num = int(aadhar_num)

    c = 0
    digit = 0
    p_i_ni = 0
    for i in range(12):
        digit = aadhar_num % 10
        aadhar_num //= 10
        p_i_ni = p[i % 8][digit]
        c = d[c][p_i_ni]
    if (c == 0):
        return True
    else:
        return False

code = input("Enter Aadhaar Number to be
verified:")
if isValidAadhar(code):
    print("Valid")
else:
    print("Invalid")
```

- **Output for the Python code:**

```
D:\DM Python Codes>python "Verhoeff algorithm.py"
Enter Aadhaar Number to be verified:397788000234
Valid
```

- **C++ Code:**

```
#include <iostream>
using namespace std;

int isValidAadhar(long aadhar_no){
    int d[10][10] = {
        {0, 1, 2, 3, 4, 5, 6, 7, 8, 9},
        {1, 2, 3, 4, 0, 6, 7, 8, 9, 5},
        {2, 3, 4, 0, 1, 7, 8, 9, 5, 6},
        {3, 4, 0, 1, 2, 8, 9, 5, 6, 7},
        {4, 0, 1, 2, 3, 9, 5, 6, 7, 8},
        {5, 9, 8, 7, 6, 0, 4, 3, 2, 1},
        {6, 5, 9, 8, 7, 1, 0, 4, 3, 2},
        {7, 6, 5, 9, 8, 2, 1, 0, 4, 3},
        {8, 7, 6, 5, 9, 3, 2, 1, 0, 4},
        {9, 8, 7, 6, 5, 4, 3, 2, 1, 0},
    };
    int p[8][10] = {
        {0, 1, 2, 3, 4, 5, 6, 7, 8, 9},
        {1, 5, 7, 6, 2, 8, 3, 0, 9, 4},
        {5, 8, 0, 3, 7, 9, 6, 1, 4, 2},
        {8, 9, 1, 6, 0, 4, 3, 5, 2, 7},
        {9, 4, 5, 3, 1, 2, 6, 8, 7, 0},
        {4, 2, 8, 6, 5, 7, 3, 9, 0, 1},
        {2, 7, 9, 3, 8, 0, 6, 4, 1, 5},
        {7, 0, 4, 6, 9, 1, 3, 2, 5, 8},
    };
    int inv[10] = {0, 4, 3, 2, 1, 5, 6, 7, 8, 9};
    string aadhar_str = to_string(aadhar_no);
    if (aadhar_str.length() != 12)
        return false;
    int c = 0, digit = 0, p_i_ni = 0;
    for (int i = 0; i < 12; i++){
        digit = aadhar_no % 10;
        aadhar_no = aadhar_no / 10;
        p_i_ni = p[i % 8][digit];
        c = d[c][p_i_ni];
    }

    if (c == 0)
        return true;
    else
        return false;
}

int main(){
    long code;
    cout << "Enter Aadhaar Number to be
verified:";
    cin >> code;
    if (isValidAadhar(code))
        cout << "Valid" << endl;
    else
        cout << "Invalid" << endl;
    return 0;
}
```

- **Output for the C++ code:**

```
Enter Aadhaar Number to be verified:397788000234
Valid


...Program finished with exit code 0
Press ENTER to exit console.
```

- **Java Code:**

```
import java.util.*;
public class Verhoeff
{
    Scanner ob=new Scanner(System.in);

    public int isvalidaadhar(long aadhar_no)
    {
        long d[][]={
            {0,1,2,3,4,5,6,7,8,9},
            {1,2,3,4,0,6,7,8,9,5},
            {2,3,4,0,1,7,8,9,5,6},
            {3,4,0,1,2,8,9,5,6,7},
            {4,0,1,2,3,9,5,6,7,8},
            {5,9,8,7,6,0,4,3,2,1},
            {6,5,9,8,7,1,0,4,3,2},
            {7,6,5,9,8,2,1,0,4,3},
            {8,7,6,5,9,3,2,1,0,4},
            {9,8,7,6,5,4,3,2,1,0}
        };
        long p[][]={
            {0,1,2,3,4,5,6,7,8,9},
            {1,5,7,6,2,8,3,0,9,4},
            {5,8,0,3,7,9,6,1,4,2},
            {8,9,1,6,0,4,3,5,2,7},
            {9,4,5,3,1,2,6,8,7,0},
            {4,2,8,6,5,7,3,9,0,1},
            {2,7,9,3,8,0,6,4,1,5},
            {7,0,4,6,9,1,3,2,5,8}
        };
        long inv[]={0,4,3,2,1,5,6,7,8,9};
        int
len=String.valueOf(aadhar_no).length();
        if(len!=12)
            return 0;
        long c=0,digit=0,temp=0;
        for(int i=0;i<12;i++)
        {
            digit=aadhar_no%10;
            aadhar_no=aadhar_no/10;
            temp=p[i%8][(int)digit];
            c=d[(int)c][(int)temp];
        }
        if(c==0)
            return 1;
        else
            return 0;
    }

    public void main() {
        System.out.println("Enter Aadhar number
to be verified : ");
        long code=ob.nextLong();
        if(isvalidaadhar(code)==1)
            System.out.println("Valid");
        else
            System.out.println("Invalid");
    }
}
```

- **Output for the Java code:**

```
Enter Aadhar number to be verified :
397788000234
Valid
```

Conclusion

While there remains to be a host of potential fallout regarding check digit schemes, the fact remains that this method for identifying products is fairly sound. While there are some cracks within catching every error, using modular arithmetic for the check digit schemes provides us with a simple and efficient strategy. As discussed at the beginning of the paper, it is important to have efficiency when gathering data on large scales, such as the ones presented here. Perhaps other areas of mathematics could prove better in terms of error prevention rate and overall efficiency numbers; but the math laid out in this paper provides sufficient evidence to warrant continuation with check digit schemes. Maybe it will cost us an extra 10 dollars at the grocery store or getting placed on the wrong light; but the overall success of the check digit schemes is understood and will continue to be largely utilized for the foreseeable future. ISBNs (International Standard Book Numbers) and ISSNs (International Standard Serial Numbers) are unique bibliographic reference numbers used to identify individual books, journals, multimedia, and other published materials. Luhn algorithm, also known as modulus 10 or mod 10 algorithm, is a simple checksum process for validating various identification numbers such as credit card numbers, Canadian social security's numbers. This algorithm is designed to protect again mistyped or accidental error rather than malicious attacks. VerhoeffCheckDigit class implements the algorithm. Static methods are provided to calculate and verify check digits.

The AppendCheckDigit method calculates the Verhoeff check digit for a given number provided as input, then returns the input with the check digit appended at the end. Four overloads of this method allow for different data types as input and return value.

The CalculateCheckDigit method calculates and returns the Verhoeff check digit for a given number provided as input. Four overloads of this method provide for different data types as input. The return value is an int for each overload.

The Check method verifies the check digit for a number. Eight overloads of this method provide for different data types as input, and whether the check digit is included as part of the input number or as a separate number. The return value is a boon for each overload: (true if the check digit is valid, false otherwise). Use one of the first four overloads if the check digit is the last digit in the input. Use one of the last four overloads if the check digit is separate to the input.