60004200132

Ayush Jain

CN-Experiment No. 3

Aim: To implement CRC and hamming codes as error detection and correction codes.

Theory:

1) **CRC:** The Cyclic Redundancy Checks (CRC) is the most powerful method for Error-Detection and Correction. It is given as a kbit message and the transmitter creates an (n – k) bit sequence called frame check sequence. The out coming frame, including n bits, is precisely divisible by some fixed number. Modulo 2 Arithmetic is used in this binary addition with no carries, just like the XOR operation.

   Redundancy means duplicacy. The redundancy bits used by CRC are changed by splitting the data unit by a fixed divisor. The remainder is CRC.

2) **Hamming Code:** Hamming code is a popular error detection and error correction method in data communication. Hamming code can only detect 2 bit error and correct a single bit error which means it is unable to correct burst errors if may occur while transmission of data.

   Hamming code uses redundant bits (extra bits) which are calculated according to the below formula:-

   $2r \geq m+r+1$

   Where r is the number of redundant bits required and m is the number of data bits.

   - R is calculated by putting r = 1, 2, 3 … until the above equation becomes true.
   - R1 bit is appended at position 20
   - R2 bit is appended at position 21
   - R3 bit is appended at position 22 and so on.

   These redundant bits are then added to the original data for the calculation of error at receiver's end.

At receiver's end with the help of even parity (generally) the erroneous bit position is identified and since data is in binary we take complement of the erroneous bit position to correct received data.

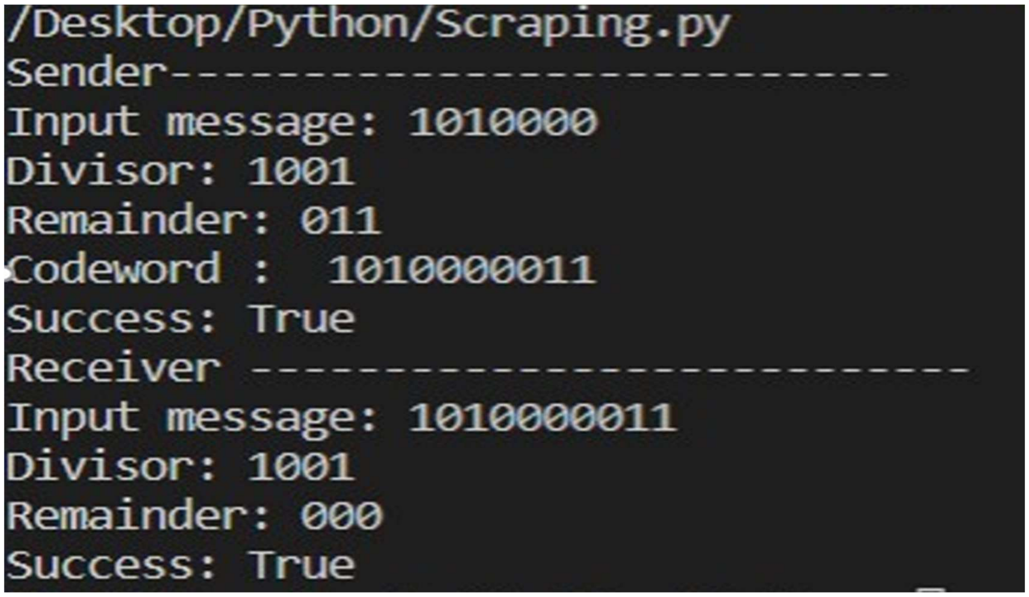
Respective index parity is calculated for r1, r2, r3, r4 and so on.

Program :

1) CRC:

```
def crc(msg, div, code='000'):
        msg = msg + code

        msg = list(msg)     div = list(div)     for i in
    range(len(msg)-len(code)):         if msg[i] == '1':
    for j in range(len(div)):                msg[i+j] =
    str((int(msg[i+j])+int(div[j]))%2)

        return ''.join(msg[-len(code):])

    print('Sender---------------------------')
    div = '1001'
    msg = '1010000'

    print('Input message:', msg)
    print('Divisor:', div)

    code = crc(msg, div)

    print('Remainder:', code) codeword = msg +
    code print("Codeword : ",codeword)
    print('Success:', crc(msg, div, code) ==
    '000')
    print('Receiver ---------------------------')

    print('Input message:', codeword)
    print('Divisor:', div)

    code = crc(codeword, div)

    print('Remainder:', code)

    print('Success:', crc(codeword, div, code) == '000')
```

Output:



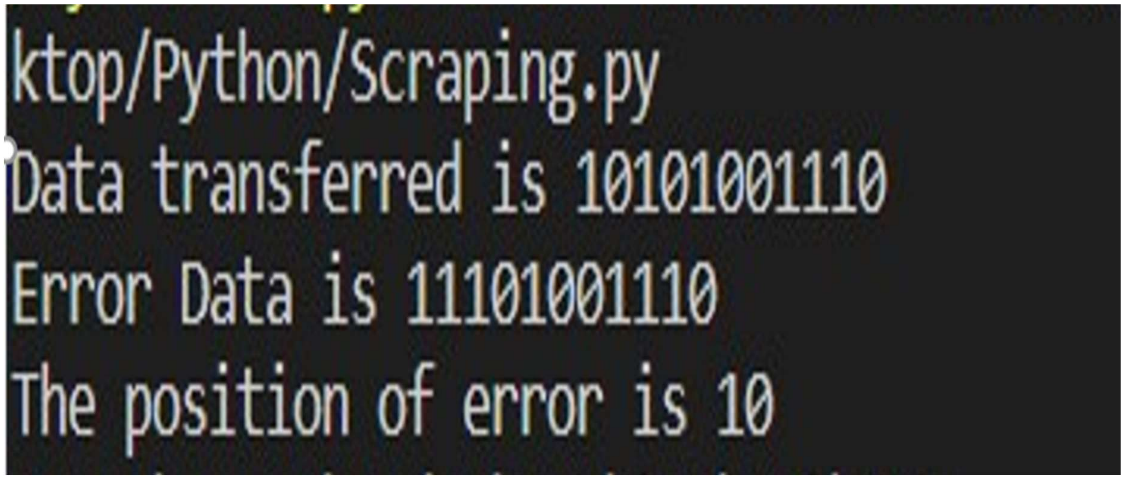
```
/Desktop/Python/Scraping.py
Sender-----------------------------
Input message: 1010000
Divisor: 1001
Remainder: 011
Codeword :  1010000011
Success: True
Receiver ------------------------------
Input message: 1010000011
Divisor: 1001
Remainder: 000
Success: True
```

2) Hamming Code:

```
def calcRedundantBits(m):
for i in range(m):
                    if(2**i >= m + i + 1):
                              return i
def posRedundantBits(data, r):
          j = 0
k = 1
          m = len(data)   res = ''
for i in range(1, m + r+1):
if(i == 2**j):                      res =
res + '0'
                              j += 1
else:
                              res = res + data[-1 * k]
                    k += 1
          return res[::-1] def
calcParityBits(arr, r):  n =
len(arr)          for i in
range(r):
                  val = 0
for j in range(1, n + 1):



                           if(j & (2**i) == (2**i)):
                  val = val ^ int(arr[-1 * j])
                  arr = arr[:n-(2**i)] + str(val) + arr[n-
(2**i)+1:]        return arr def detectError(arr, nr):
          n = len(arr)
res = 0           for i in
range(nr):
                  val = 0                        for j in range(1, n
+ 1):                      if(j & (2**i) == (2**i)):
                  val = val ^ int(arr[-1 * j])
res = res + val*(10**i)
          return int(str(res), 2) data =
'1011001' m = len(data) for i in
range(m):                  if(2**i >= m
+ i + 1):
                           r = i
          break
arr = posRedundantBits(data, r) arr
= calcParityBits(arr, r) print("Data
transferred is " + arr) arr =
'11101001110'
```

```
print("Error Data is " + arr) correction
= detectError(arr, r)
print("The position of error is " + str(correction))
```

Output:



```
ktop/Python/Scraping.py
Data transferred is 10101001110
Error Data is 11101001110
The position of error is 10
```

Conclusion: Successfully implemented CRC and hamming codes as error detection and correction codes.