BURSA ULUDAĞ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

2020 – 2021 Eğitim Öğretim Yılı Bahar Yarıyılı

Gömülü Sistemler Dersi Projesi

Proje Konusu
CRC Pattern

Mehmet Faruk GÜL - 031790044
031790044@ogr.uludag.edu.tr

Mayıs 2021

İçindekiler

1. Giriş

Raporumuz Gömülü dersi projemizin konusu olan CRC Pattern'in üzerine yazılmıştır. CRC'nin açılımı türkçede Döngüsel Artıklık Kontrolü olarak geçmektedir. CRC'lerin böyle isimlendirilmesinin sebebi konrol değerlerinin bir fazlalık, yani mesajı genişleten bir bilgi olmasından dolayıdır ve algoritması döngüsel kodlara dayanmaktadır. 1961 yılında W. Wesley Peterson tarafından tasarlanmış bir blok kodudur. Genel anlamda telekomünikasyon ağları ve depolama aygıtları aracılığıyla iletilen verilerdeki kazara değişiklikleri tespit etmek için kullanılmaktadırlar.

CRC, iletişim sistemi tarafından kararlaştırılan önceden belirlenmiş bir bölen tarafından gönderilen veri bitlerinin ikili bölünmesini içermektedir. Bölen kısmı polinom kullanımıyla oluşturulmaktadır. Bundan dolayı CRC'ye polinom kod sağlama toplamı da denmektedir. CRC'ler kullanılma sebeplerinde ikili donanıma uygulanma basitliği, matematiksel olarak analiz etme kolaylıkları ve özellikle iletim kanallarındaki gürültünün neden olduğu yaygın hataları tespit etmede iyidirler. Kontrol değerinin sabit bir uzunluğu olduğundan dolayı onu oluşturan işlev bazen bir karma işlevi olarak kullanılmaktadır.

2. Kaynak Araştırması

2.1 Döngüsel Artıklık Denetimi - Cyclic Redundancy Check

CRC (Döngüsel Artıklık Denetimi) döngüsel hata düzeltme kodları teorisine dayanmaktadır. İletişim ağlarında hata tespiti yapılması için sabit uzunluklu kontrol değeri eklenerek mesajları kodlama işlemi yapan döngüsel kodların kullanımı ilk olarak W. Wesley Peterson tarafından 1961 yılında önerilmiştir. Döngüsel kodların uygulanması kolay değildir ama hatalarının tespiti için özellikle uygun olmasının sebebi mesajlardaki hatalı veri sembollerinin bitişik olmasıdır. Bu önemlidir, çünkü patlama hataları güçlü ve optik depolama ekipmanı da dahil olmak üzere birçok iletişim kanalında yaygındır. Tipik olarak, bir N, CRC istenilen uzunlukta bir veri bloğuna tatbik bitlik herhangi bir tek hata fazla olmayan bir süre patlama algılar, n bit algılar tüm uzun hata açılışları fraksiyondur (1-2-N).

Bir CRC kodunun özellikleri, bir polinom üreticisinin tanımlanmasını gerektirir. Bu polinom, iletiyi alan ve bölümün atıldığı ve kalan kısmın sonuç haline geldiği bir polinom uzun bölmede bölen haline gelir. Önemli bir kısım olarak, Polinom katsayıları sonlu alanın aritmetiğine göre hesaplanmalıdır, çünkü bunun sayesinde toplama işlemi her zaman bitsel paralel gerçekleştirilebilir.

Bir CRC n değerli olduğunda bitlik CRC n bit uzunluğundadır. Belirlenmiş bir n değeri için, her biri birbirinden farklı polinom içeren CRC'ler bulunabilir. Böyle bir polinomun en yüksek derecesi n'dir ve toplamda n + 1 tane terimi bulunmaktadır. Uzunluğu ve kodlaması n + 1'dir ve bu kadar bit değeri gerektirmektedir. En basit anlamda hata algılama sistemi olan eşlik biti aslında 1 bitlik bir CRC'dir: üreteç polinomu x + 1'i(iki terim) kullanır ve CRC-1 adını alır.

2.1.1 Uygulama

CRC özelliğine sahip elektronik bir cihaz, gönderilecek veya depolanacak her bir veri bloğu için CRC olarak bilinen kısa, belirli uzunluklu ikili bir dizi hesabı yapar ve bunu verilere aktararak bir kod sözcüğü oluşturur.

Bir kod sözcüğü okunduğunda ya da alındığında, cihaz ya kontrol değerini veri bloğundan yeni hesaplanmış olan değer ile karşılaştırır ya da eşdeğer olarak tüm kod sözcüğü üzerinde CRC algoritmasını çalıştırır ve sonuçta elde edilen kontrol değerini beklenen bir kalıntı sabiti ile karşılaştırma yapar.

CRC değerleri eşleşmezse, blok bir veri hatası içerir. Cihaz, bloğun yeniden okunması veya tekrar gönderilmesini istemesi gibi düzeltici eylemler gerçekleştirebilir. Aksi takdirde, verilerin kusursuz olduğu varsayılır (hala hataları bulunamamış olabilir; bu hata kontrolünün doğasında vardır).

2.1.2 Veri Bütünlüğü

CRC'ler, mesajların bütünlüğü için hızlı ve mantıksal olarak geçebilecek iletişim kanallarındaki ortak hatalara karşı korunmak için özellikle tasarlanmıştır. Ancak, verilerdeki kasıtlı değişim için uygun değildir.

2.1.3 Hesaplama

Bir n- bitlik ikili CRC'yi hesaplamak için, bir satırdaki girdiyi temsil eden bitleri sıralayın ve CRC'nin bölenini (" polinom " olarak adlandırılır) temsil eden (n + 1) - bit modelini satırın sol ucunun altına yerleştirin. Sıradaki örnekte 3 bitlik bir CRC ile 14 bitlik mesajı kodlayacağız. Polinom, katsayılar ikili olarak yazılır; 3. derece polinomun 4 katsayısı vardır (1x3+0x2+x+1). Bu durumda, katsayılar 1, 0, 1 ve 1'dir. Hesaplamanın sonucu 3 bit uzunluğunda olacaktır, bu sebeple 3 bitlik CRC olarak değerlendirilecektir. Ama polinomu belirtmek için 4 bite ihtiyacımız bulunmaktadır.

Kodlanacak mesajımız: 11010011101100

Bu, ilk olarak CRC'nin n bit uzunluğuna denk gelen sıfırlarla doldurulur. Bu, karşımıza çıkan kod sözcüğünün düzgün bir biçimde olması için yapılır. Sırada 3 bitlik bir CRC değerini hesaplanmasına bakalım:

```
11010011101100 000 <--- input right padded by 3 bits
1011               <--- divisor (4 bits) = x³ + x + 1
------------------
01100011101100 000 <--- result
```

Algoritma her adımda bir bölümün doğrudan bölümünü etkiler. Bu tekrarlamanın sonucu, polinom böleninin üzerindeki bitlerle bitsel XOR değeri uygulanması sonucudur. Bölenin üstünde olmayan bitler doğrudan aşağıya kopyalanır. Bölen ise sonradan girişte kalan en çok 1 bit ile hizalanacak şekilde sağa kaydırılır ve bölen giriş satırının sağ tarafına ulaşıncaya kadar işlem tekrarlanır. Tüm hesaplama altta bulunmaktadır:

```
11010011101100 000 <--- input right padded by 3 bits
1011               <--- divisor
01100011101100 000 <--- result (note the first four bits are the XOR with the divisor beneath, the rest of the bits are
unchanged)
 1011              <--- divisor ...
00111011101100 000
  1011
00010111101100 000
   1011
00000001101100 000 <--- note that the divisor moves over to align with the next 1 in the dividend (since quotient for that
step was zero)
       1011            (in other words, it doesn't necessarily move one bit per iteration)
00000000110100 000
        1011
00000000011000 000
         1011
00000000001110 000
          1011
00000000000101 000
           101 1
-----------------
00000000000000 100 <--- remainder (3 bits).  Division algorithm stops here as dividend is equal to zero.
```

Burada en solda bulunan bölen biti etkileşime girdiği her giriş bitini sıfırladığından dolayı bu işlem son bulduğunda giriş satırında bulunan sıfır olmayan tek bit, satırın sağ ucundaki n bittir. Bu n bit, bölme adımının geri kalanıdır ve CRC fonksiyonunun değeri olacaktır. Alınan bir mesajın geçerliliğini öğrenmek için, yukarıdaki hesaplama tekrar yapılarak, bu sefer sıfır yerine kontrol değeri eklenerek

kolaylıkla doğrulanabilmektedir. Saptanabilir hata yoksa kalan sıfıra eşit olmalıdır ve mesajın geçerliği olduğu sonucuna varılır.

```
11010011101100 100 <--- input with check value
1011               <--- divisor
01100011101100 100 <--- result
 1011              <--- divisor ...
00111011101100 100


......

00000000001110 100
          1011
00000000000101 100
           101 1
------------------
00000000000000 000 <--- remainder
```

2.2 Döngüsel Artıklık Kontrolü (CRC)

CRC ile alınan bir değere bölünecek bir oluşturucu polinomumuz var. Sıfırın kalanını alırsak, hata olmadığını belirleyebiliriz. Daha sonra bir modulo-2 bölmesinden gerekli kalanı hesaplamalı ve bölmeyi gerçekleştirdiğimizde kalanın sıfır olması için bunu verilere eklemeliyiz.

Basit bir örnek vermek gerekirse, 32 var ve bunu 9'a bölünebilir hale getiriyoruz, '320' yapmak için '0' ekliyoruz ve şimdi 35 kalan 4 vermek için 9'a bölüyoruz. O halde 324 yapmak için '4' ekleyelim. Şimdi alındığında 9'a bölüyoruz ve eğer cevap sıfırsa hata yok ve son rakamı görmezden gelebiliriz. Buna bir örnek göstermek gerekirse:

İkili biçim: 1001100, 110111'e bölünür

x^5 + x^4 + x^2 + x + 1

İkili biçim (sıfır eklendi): 100110000000 bölü 110111
Sonuç 1111010
Kalan 00110

Çalışma:

```
     1111010
------------
100110000000
110111
------
 10001000000
 110111
```

```
------
 1010100000
 110111
 ------
  111010000
  110111
  ------
   01101000
   000000
   ------
    1101000
    110111
    ------
     000110
     000000
     ------
      00110
```

İletilen değer: 100110000110

Bu örneğimizde tam olarak olayın nasıl olduğunu anlatmış bulunmaktayız. Kodlama kısmında akış bu şekilde işlemektedir.

2.3 CRC Oluşturma ve Kontrol Etme

Bu uygulama notu, Döngüsel Artıklık Tespiti (CRC) teorisini ve uygulamasını kontrol edin. CRC, bir mesajdaki hataları tespit etmek için kullanılır. İki uygulama gösterilmektedir:

• Tablo tabanlı CRC hesaplaması
• Döngü odaklı CRC hesaplaması

CRC, iletilen mesajlardaki veya depolanan verilerdeki hataları tespit etmek için yaygın bir yöntemdir. CRC çok güçlüdür ve kolayca uygulanarak veri güvenliğini sağlar.

2.3.1 Operasyon Teorisi

Bir CRC hesaplamasının teorisi açıktır. Veriler, CRC algoritması tarafından ikili sayı olarak ele alınır. Bu sayı, polinom adı verilen başka bir ikili sayıya bölünür. Bölümün geri kalanı, iletilen mesaja eklenen CRC sağlama toplamıdır. Alıcı, mesajı (hesaplanan CRC dahil), kullanılan vericiyle aynı polinomla böler. Bu bölmenin sonucu sıfır ise, iletim başarılı olmuştur. Bununla birlikte, sonuç sıfıra eşit değilse, iletim sırasında bir hata meydana geldi. CRC-16 polinomu Denklem 1'de gösterilmektedir. Polinom, ikili bir değere çevrilebilir, çünkü bölen, ikili katsayıları olan bir polinom olarak görülür. Örneğin, CRC-16 polinomu 1000000000000101b'ye çevrilir. X2 veya x15 gibi tüm katsayılar, ikili değerde mantıksal 1 ile temsil edilir. Bölüm, Modül 2 aritmetiğini kullanır. Modül 2 hesaplaması basitçe iki sayının XOR'lanmasıyla gerçekleştirilir.

EXAMPLE 1: MODULO-2 CALCULATION

```
        1 0 0 1 1 0 0 1 0 1
XOR     0 1 0 0 1 1 0 1 1 1
=       1 1 0 1 0 1 0 0 1 0
```

XOR-Function

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Modül 2

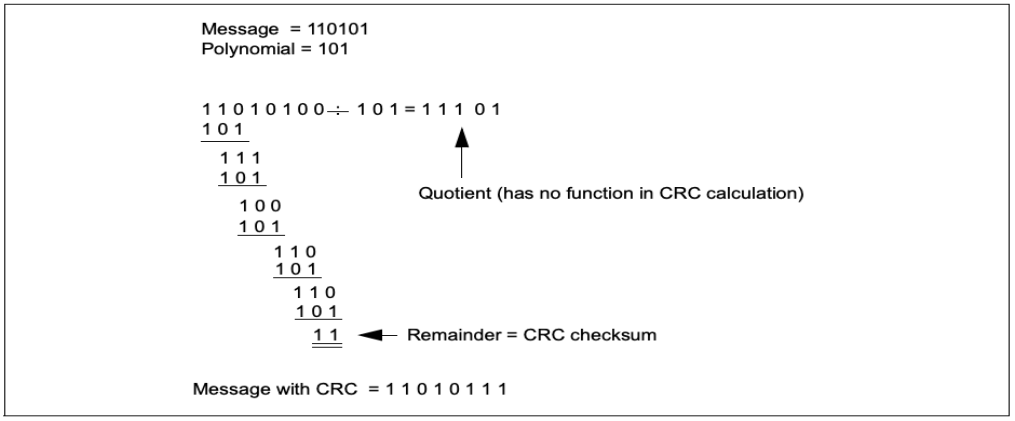
EQUATION 1: THE CRC-16 POLYNOMIAL

$$P(x) = x^{16}+x^{15}+x^{2}+1$$

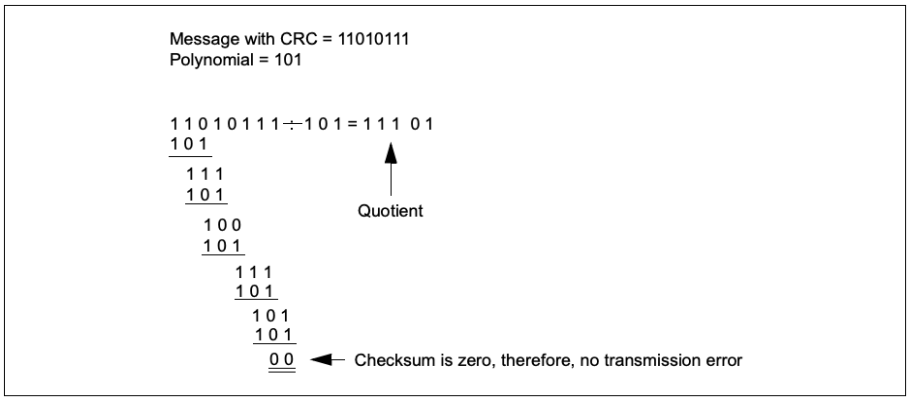
Denklem 1

2.3.2 Örnek Hesaplama

Bu örnek hesaplamada, mesaj iki bayt uzunluğundadır. Genel olarak, mesaj bayt cinsinden herhangi bir uzunluğa sahip olabilir. CRC değeri 1'i hesaplamaya başlamadan önce, mesajın n-bit ile artırılması gerekir, burada n, polinomun uzunluğudur. CRC-16 polinomunun uzunluğu 16 bittir, bu nedenle 16 bitin orijinal mesaja kadar artırılması gerekir. Bu örnek hesaplamada, polinomun uzunluğu 3 bittir, bu nedenle mesajın sonunda üç sıfır ile genişletilmesi gerekir. Bir CRC için örnek bir hesaplama Örnek 1'de gösterilmektedir. Ters hesaplama Örnek 2'de gösterilmektedir.



Örnek 1: CRC oluşturmak için hesaplama



Örnek 2: Bir CRC Hatası İçin Bir Mesajı Kontrol Etme

3. Araştırma Sonuçları

Yapılan araştırmalar sonucunda CRC hakkında genel bilgiler aktarılmış bulunmaktadır. Bu bilgiler sonucunda CRC, iletilen mesajlardaki veya depolanan verilerdeki hataları tespit etmek için yaygın bir yöntem olduğu sonucuna varıldı. CRC'nin çok güçlü olduğu ve kolayca uygulanarak veri güvenliğini sağladığını örnekler ile tespit edildi.

Bu algoritmanın nasıl uygulandığı hakkında genel bilgilere sahip olduk. Şimdi bu bilgileri kullanarak python dilinde nasıl yazılabileceğini detaylıca inceleyelim.

İlk başta kodumuza başlarken kütüphanelerimizi çağırmamız gerekmektedir.

```
import struct
import sys
import re
```

Python'daki struct modülü, dizeler ve sayılar gibi yerel Python veri türlerini bir bayt dizisine dönüştürmek için kullanılır ve bunun tersi de geçerlidir. Python sys modülü, Python Çalışma Zamanı Ortamının farklı bölümlerini işlemek için kullanılan işlevler ve değişkenler sağlar. Re modülümüzü ekledik çünkü Python'daki normal ifade veya RegEx, RE (RE'ler, regexes veya regex kalıbı) re modülü aracılığıyla içe aktarılır olarak belirtilir.

```
15  deger1="1001100"
16  deger2="1101"
```

Burada ilk değerimiz input değerimiz, diğeri divisor değerimizdir.

```
24  def polinomuGoster(a):
25      str1 = ""
26      bitsayi = len(a)
27      for x in range (0,bitsayi-2):
28          if (a[x] == '1'):
29              if (len(str1)==0):
30                  str1 +="x**"+str(bitsayi-x-1)
31              else:
32                  str1 +="+x**"+str(bitsayi-x-1)
33
34      if (a[bitsayi-2] == '1'):
35          if (len(str1)==0):
36              str1 +="x"
37          else:
38              str1 +="+x"
39
40      if (a[bitsayi-1] == '1'):
41          str1 +="+1"
42
43      print str1
```

Bu fonksiyonun amacı değer1 ve değer2 değerlerini polinom şeklinde gösterebilmektir. Bu iki değer içinde sonradan kullanılacaktır.

```
44  def ListeyeEkle(x):
45      l = []
46      for i in range (0,len(x)):
47          l.append(int(x[i]))
48      return (l)
```

Bu fonksiyonumuz değerleri bir listeye eklemektedir. Bunun için bir dizi ve diziye ekleme işlemi yapan append() fonksiyonunu kullanmaktadır.

```
52  def toString(x):
53      str1 =""
54      for i in range (0,len(x)):
55          str1+=str(x[i])
56      return (str1)
```

Bu fonksiyon buna parametre olarak gönderilen değeri string'e çeviriyor ve bize return değeri olarak sunmaktadır.

```
61  def bolmeIslemi(deger1,deger2):
62      a = ListeyeEkle(deger1)
63      b = ListeyeEkle(deger2)
64      calisma=toString(deger1)+"\n"
65
66      res=""
67      boslukEkle=""
```

Şimdi asıl önemli kısma girmekteyiz. Burada bölme işlemini yapacağız. Verileri ilk başta listeye aktarıyoruz ve deger1 değerini working adlı parametreye string değer olarak gönderiyoruz.

```
75      while len(b) <= len(a) and a:
76              if a[0] == 1:
77                  del a[0]
78                  for j in range(len(b)-1):
79                          a[j] ^= b[j+1
80                  if (len(a)>0):
81                  calisma +=boslukEkle+
82                  calisma +=boslukEkle+
83                  boslukEkle+=" "
84                  calisma +=boslukEkle+toString(a)+"\n"
85                  res+= "1"
86
87          else:
88                  del a[0]
89              calisma +=boslukEkle+"0" * (len(b))+"\n"
90              calisma +=boslukEkle+"-" * (len(b))+"\n"
91              boslukEkle+=" "
92              calisma +=boslukEkle+toString(a)+"\n"
93
94                  res+="0"
```

```
def toString(x):
    str1 =""
    for i in range (0,len(x)):
toString: toString
```

Burada kaynak taramasında anlatmış olduğumuz bölme işlemini yapmaktayız. Yaptığımız işlemlerin böyle bu şekilde karmaşık olmasının sebebi bölme olayını terminalde tamamen gösterilebilmesini sağlamaktır. Çıktı kısmında örneğini

görebileceksiniz. While döngüsünde a değerimiz b değerimizden büyük oldukça devam etmektedir ve küçük olduğu durumda ise bölme işlemimiz son bulmaktadır.

```
 99      print "Çıktı: \t",res
100      print "Kalan: \t",toString(a)
101
102      print "Çalışma: \t\n\n",res.rjust(len(deger1)),"\n",
103      print "-" * (len(deger1)),"\n",calisma
104
105      return toString(a)
```

Burada çıktılarımızı göstermekteyiz. İlk başta result değerimiz, sonra kalan değerimiz bulunmaktadır. working kısmında bölüm olayının nasıl yapıldığı bulunmaktadır.

```
109  print "İkili Form:",deger1,", Bölen sayısı: ",deger2
110  print ""
111  polinomuGoster(deger1)
112  polinomuGoster(deger2)
```

Burada artık main işlemler yapılmaktadır diyebiliriz. Değerleri gösterme işlemi yapıldıktan sonra polinom halinde gösterimler yapılmaktadır.

```
114  # Burada 0 değerleri deger1'e eklenmektedir.
115  strSifir=""
116  strSifir = strSifir.zfill(len(deger2)-1)
117  deger3=deger1+strSifir
```

Burada 0 değerleri deger1'e eklenmektedir.

```
119  # Burada 0 eklenmiş hali gösterilmektedir.
120  print ""
121  print "İkili form (0'lar eklendi): ",deger3,"Bölen sayı: ",deger2
```

0 eklenmiş hali gösterilmektedir.

```
127  res=bolmeIslemi(deger3,deger2)
128  print "Gönderilen değer:",deger1+res
```

Bu kısımda ise bizim için önemli olan bölüm fonksiyonu çalışmıştır ve fonksiyonun çalışması bittikten sonra çıktı değerini bize sunmaktadır.

```
131    print "Şimdi Doğru gönderim hatalı mı diye bakacağız şimdi."
132
133    #Hatalı mı değil mi diye kontrol ediyoruz burada
134    yeniSayi = deger1+res
135    yeniSayi = yeniSayi
136
137    #Tekrardan bir bölme işlemi yapılması gerekiyor yeni sayıyla
138    #Bu şekilde çıktıdan anlamış olacağız hatalı mı değil mi diye.
139    res= bolmeIslemi(yeniSayi, deger2)
140    print "Hata kodumuz:", res
141
142    found = re.search("1", res)
143  v if(found):
144        print "Hatalı gönderim yapıldı!"
145  v else:
146        print "Doğru gönderim yapıldı!"
147
148    # Bu şekilde çalışmalarını bitirmektedir. Bu kod python 2.7 ile sorunsuz bir
149    # şekilde çalışmaktadır.
```

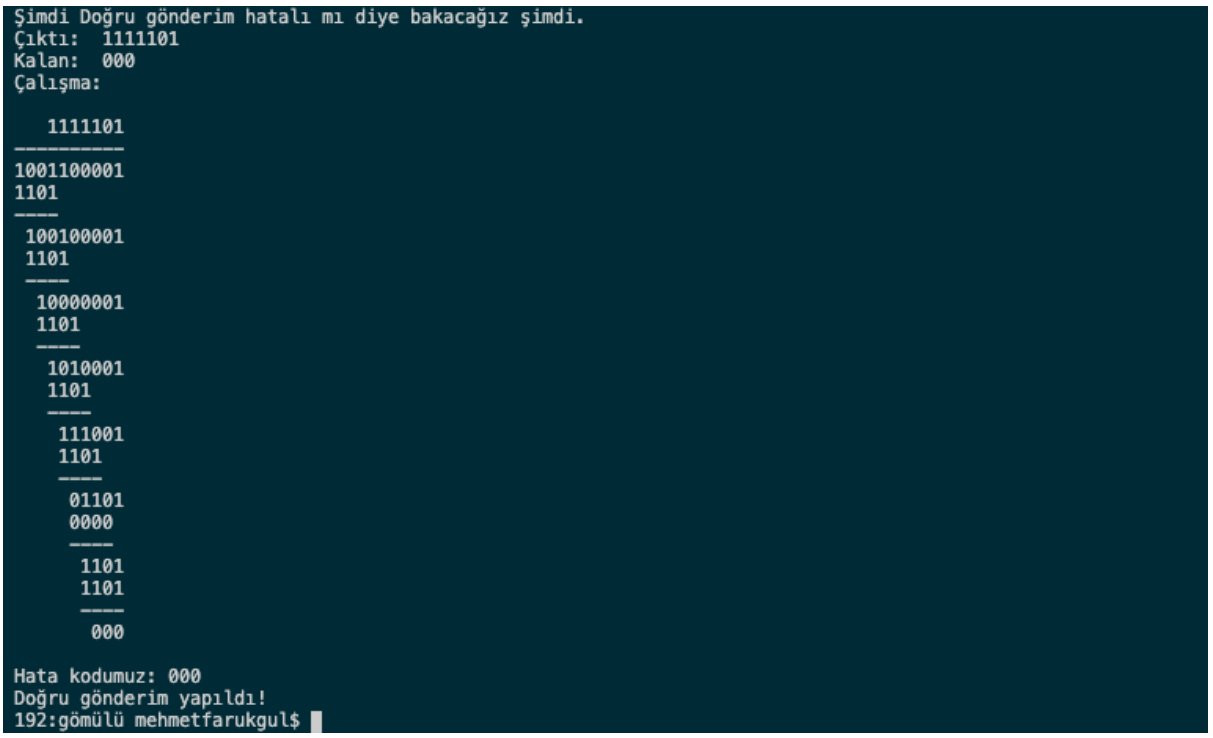
Burada tekrardan yeni sayımız ile bölme işlemi yapıyoruz ve çıktımıza bakıyoruz. Eğer çıktımızda 1 yok ise ise hatalı değildir ama var ise hatalıdır. Bu şekilde mesajımızın doğru gönderilip gönderilmediğini anlamış oluyoruz.

Şimdi çıktıya göz atalım.

```
192:gömülü mehmetfarukgul$ python crc.py
İkili Form: 1001100 , Bölen sayısı:  1101

x**6+x**3+x**2
x**3+x**2+1

İkili form (0'lar eklendi):  1001100000 Bölen sayı:  1101
Çıktı:  1111101
Kalan:  001
Çalışma:

   1111101
  ----------
1001100000
1101
----
 100100000
 1101
 ----
  10000000
  1101
  ----
   1010000
   1101
   ----
    111000
    1101
    ----
     01100
     0000
     ----
      1100
      1101
      ----
       001

Gönderilen değer: 1001100001
```

CRC python kodu genel anlamda bu şekilde çalışmaktadır. İlk başta değerleri alıp polinom hale çevirili bir şekilde bize göstermektedir. Bunu yaptıktan sonra bölme

işlemlerini yapıp, bize nasıl yapıldığını terminalde gösterip son çıktıyı da beraberinde sunmaktadır.



Devamında ise mesajımızın hatalı yada doğru gönderildiğine dair bilgi alabilmemiz için yeni değer ile bölme fonksiyonumuz tekrardan çalışmaktadır ve sonucumuz gösterilmektedir.

4. Kaynakça

URL 1. https://en.wikipedia.org/wiki/Cyclic_redundancy_check, (01.05.2021)
URL 2. https://www.tutorialspoint.com/what-is-algorithm-for-computing-the-crc, (25.04.2021)
URL 3. https://www.gatevidyalay.com/cyclic-redundancy-check-crc-error-detection/, (25.04.2021)
Thomas Schmidt, Microchip Technology Inc., 2000, “CRC Generating and Checking”.