Computer Network Lab

Exp 07: CRC Computation

Dr. Ram P Rustagi
Sem V (2018-H2)
Dept of CSE, KSIT
rprustagi@ksit.edu.in

Ex07 Resources

- References:
  - http://www.ross.net/crc/crcpaper.html
  - http://www.repairfaq.org/filipg/LINK/F_crc_v3.html
  - http://www.ross.net/crc/download/crc_v3.txt
    - contains the program
  - http://srecord.sourceforge.net/crc16-ccitt.html
  - https://www.slideshare.net/sandeep101026/crc-java-code

Exp10 Description

- Program 07 (Java)
    - Write a program for error detecting code using CRC-CCITT.

Cyclic Concepts

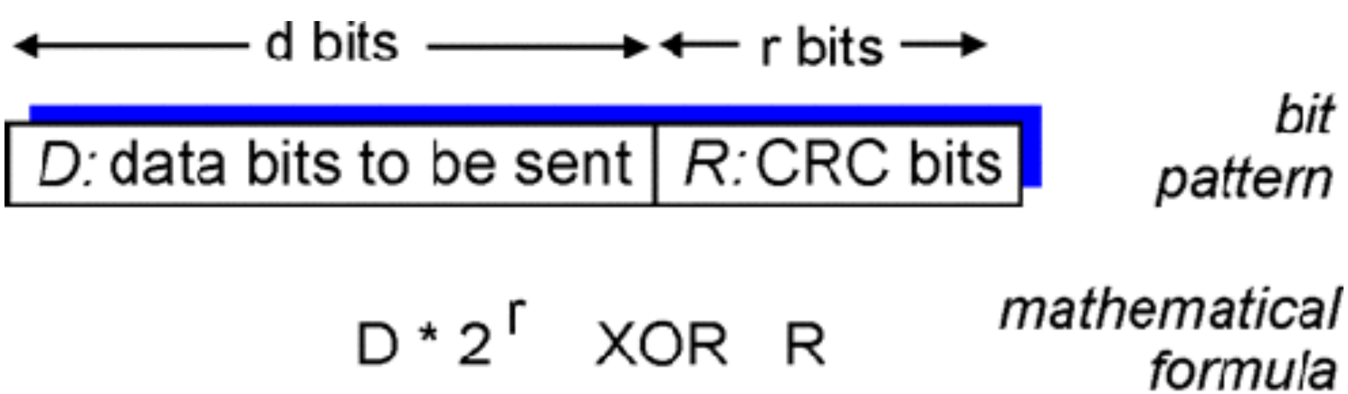
- CRC Codes known as polynomial codes
  - Each bit is taken as coefficient of polynomial
- Using module 2 i.e. bits `0`, `1`
  - Consider when we ignore carries or borrows
  - What would be difference between add & subtract
    - Can be achieved by XOR operation
    - Examples

```
1011 + 0101 = 1110
1011 - 0101 = 1110

1001 + 1101 = 0100
1001 - 1101 = 0100
```

Cyclic redundancy check

- More powerful error-detection coding
- View data bits, D, as a binary number
- Choose `r+1` bit pattern (generator), G
- Goal: choose `r` CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - Receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - Can detect all burst errors less than `r+1` bits
- Widely used in practice (Ethernet, 802.11 WiFi, ATM)



CRC example

Want:

$D \cdot 2^r$ XOR $R = nG$

Equivalently:

$D \cdot 2^r = nG$ XOR R

Equivalently:

if we divide $D \cdot 2^r$ by G, want remainder R to satisfy:

$$\boxed{R = \textit{remainder}\left[\frac{D \cdot 2^r}{G}\right]}$$

Example:

D = 101110

G = 1001

$r = 3$

G D

```
         101011
1001 ) 101110000
       1001
        101
        000
        1010
        1001
         110
         000
         1100
         1001
          1010
          1001
           011
```

R

Note: MSB of Generator is really not used in division as it always cancels out whenever '1' is added to quotient

Cyclic redundancy check

- International CRC Standards defined for
  - `8, 12, 16` and 32 bit generators
  - `16` bit generator (CRC16-CCITT)

    $x^{16} + x^{12} + x^{5} + x^{0}$, i.e.

    `1 0001 0000 0010 0001`

    - Polynomial in actual computation (`0x1021`)
    - Initial value: `0xFFFF`
- CRC Detects bursts errors of less than `r+1` bits
  - Consecutive error of `r` bits or fewer will be detected
  - Under some appropriate assumptions
    - burst of error >`r+1` bits detected
  - Can detect any odd number of bit errors

CRC-16/CCITT

- Various implementations of CRC16/CCITT
  - Xmode:
    - Initial Value: 0x0000
    - Polynomial: 0x1021
  - CCITT:
    - Initial Value: 0xFFFF
    - Polynomial: 0x1021
  - CRC16
    - Polynomial: 0x8005
    -

Exercises

- Given
  - G=`10011` (CRC-4-ITU Standard)
  - D=`1010101010`
- Question:
  - What is the value of `R`
  - Divide `10011` into `1010101010 0000`
    - `R = 0100`

Lab Program:

- Lab program expectation:
  - Provide the input data in ASCII
  - Choose your initial value of CRC appropriately
    - e.g. `0x0000` or `0xFFFF`
  - Optionally provide the polynomial with hex value e.g `0x1021`
- `java Crc16 "abcdefgh"` [`0x1021`]
  - Compare your output with `https://www.lammertbies.nl/comm/info/crc-calculation.html`
- Enhance the program to provide intial value also in command line program.

Program Template:

- Read message from command line input

```
int remainder = ??; // Initial CRC CRC.
int polynomial = ?? // ideally 0x11021
int bitvalue; // one bit of data
int remMsb; // MSB bit of remainder
```

- Check Arguments

```
if ((args.length==0)||(args.length>2)) {
   System.out.println("Usage: Crc16:
<input in hex> [<CRC Poly in hex>]”);
}
```

Program Template:

- Get data and polynomial

```
    byte[] inpdata = ??.getBytes();
    if (args.length > 1) { //CRC polynomial
        polynomial = Integer.decode(args[1]);
    }
```

Program Template:

- Computing CRC

```
   for (byte inpbyte: ??) {
      for (int count = 0; count < 8; count++)
        bitvalue=((inpbyte >>> 7-count)) & 1);
        remMsb=((remainder & 0x8000)>>>15) & 1;
        remainder = remainder << 1;
        //check XOR of data and remainder bit
        if ((??) == 1) {
          // if yes, do XOR CRC with Polynomial
          remainder = remainder ^ polynomial;
        }
      } // end for inpbyte
      remainder = remainder & 0xFFFF; //16 bits
    } // end for input data
S.o.p("CRC value " + Integer.toHexString(??));
```

Summary

- Compute CRC-16
- Use command line arguments for input