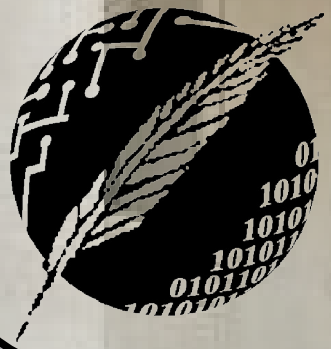
Módulo 07
Detección y Corrección de Errores (Pt. 2)



Organización de Computadoras
Depto. Cs. e Ing. de la Comp.
Universidad Nacional del Sur

Copyright

Contenidos

- Concepto de error
- Mínima distancia de un código
- Mecanismos de detección de errores
- Paridad aplicada en los códigos **VRC** y **LRC**
- Generación y verificación de código **CRC**
- Mecanismos de corrección de errores
- Códigos correctores simples
- Hamming mínima distancia 3 y 4

Código CRC

- El código **CRC** (Cyclic Redundancy Check) se basa en ciertas propiedades matemáticas que satisface el cociente entre polinomios
  - La idea en pocas palabras es agregar al patrón de bits que compone el mensaje a ser enviado un conjunto de bits adicionales de manera tal que el patrón resultante, al ser considerado como un polinomio binario, resulte divisible de manera exacta por un cierto polinomio denominado polinomio generador
  - Esperen, ¡todavía no salgan corriendo!

Código CRC

- Para poder llevar adelante el cociente entre polinomios, se debe interpretar el mensaje original como si se tratara de un polinomio
  - La clave está en pensar los **n** bits que componen al patrón original de bits como los coeficientes binarios de las primeras $n-1$ potencias de x
  - Nótese que el polinomio resultante es de grado a lo sumo $n-1$
  - Por caso, el patrón **010011** denota al polinomio $x^4 + x + 1$, mientras que el patrón **110110** al polinomio $x^5 + x^4 + x^2 + x$

Código CRC

- Sea **M(x)** el polinomio binario representado el mensaje original, sea **G(x)** el polinomio generador que se esté usando y sea **r** su grado
- En este contexto, el mensaje a ser transmitido **T(x)** es $x^r M(x) + R(x)$, donde **R(x)** es el resto de dividir $x^r M(x)$ por **G(x)**
  - **T(x)** resulta divisible de manera exacta por **G(x)** puesto que al tratarse de polinomios binarios, se puede demostrar que $x^r M(x) + R(x)$ equivale a $x^r M(x) - R(x)$

Algoritmo CRC

- Pasos para calcular los bits que deben ser agregados a un cierto mensaje **M(x)**:
  - Primero se añaden **r** bits en **0** a la derecha de **M(x)** (esto es, se añaden tantos ceros como grado tenga el polinomio generador)
  - Luego se divide el polinomio obtenido por el polinomio generador. Esta división se realiza en módulo dos, que es igual que la división binaria, con dos excepciones: no hay carries ni borrows
  - Finalmente, para obtener **T(x)** se suma el resto **R(x)** al polinomio original **M(x)** (ya desplazado en **r** bits)

Ejemplo

- Supongamos que el mensaje que se desea transmitir es **M(x) = 11010110111** y que el polinomio generador que se está usando es **G(x) = 10011**

$$x^r M(x) = 11010110110000$$

$$\begin{array}{r} \oplus 10011 \\ \hline 010011 \\ \oplus 10011 \\ \hline 0000010111 \\ \oplus 10011 \\ \hline 0010000 \\ \oplus 10011 \\ \hline 0001100 \end{array} = R(x)$$

Ejemplo

- Verifiquemos que el mensaje a ser transmitido $T(x) = x^r M(x) + R(x)$ es divisible de manera exacta por $G(x)$:

$$x^r M(x) + R(x) = 1101011011111100$$

$$\begin{array}{r} \oplus 10011 \\ \hline 010011 \\ \oplus 10011 \\ \hline 0000010111 \\ \oplus 10011 \\ \hline 0010011 \\ \oplus 10011 \\ \hline 0000000 \checkmark \end{array}$$

Roles del algoritmo

- El algoritmo **CRC** cumple dos roles:
  - Por un lado permite determinar los bits que se deben agregar al mensaje original
  - A su vez, también permiten verificar si el mensaje recibido contiene o no errores
- Nótese que el receptor puede ir dividiendo el mensaje a medida que va recibiendo los bits
  - Es decir, no hace falta recibirlo en su totalidad para recién ahí comenzar a verificar el **CRC**
  - Este código es muy simple de implementar en **HW**

Ejemplo

- Supongamos que se alteran un par de los bits del mensaje transmitido **T(x)**. En este contexto, verifiquemos nuevamente el cociente:

```
1101011011111100 →  1001111011111100
                   ⊕10011
                     0000011011
                        ⊕10011
                          010001
                         ⊕10011
                           00010110
                            ⊕10011
                              001010
```

Polinomio generador

- El polinomio generador a ser usado debe ser elegido con cuidado, ya que la capacidad de detección de errores dependerá del mismo
- Existen varios polinomios muy conocidos:
  - **CRC-4-ITU**: $x^4 + x + 1$
  - **CRC-16-IBM**: $x^{16} + x^{15} + x^2 + 1$
  - **CRC-CCITT**: $x^{16} + x^{12} + x^5 + 1$
  - **CRC-32**: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Códigos cíclicos

- Los polinomios generadores más interesantes dan a lugar a códigos cíclicos
  - Un código se denomina cíclico cuando el mensaje **T(x)** puede ser corrido cíclicamente a derecha o izquierda un número arbitrario de lugares sin perder la propiedad de ser divisible exactamente por **G(x)**
  - Sea **k** la longitud del mensaje original **M(x)** y sea **n** la del mensaje a transmitir **T(x)**. Se puede demonstrar que para obtener un código cíclico para el par **(n, k)** basta con tomar como polinomio generador de grado $n-k$ a alguno de los factores del polinomio x^n+1

Ejemplo

- Se desea obtener un polinomio generador para $k = 11$ y $n = 15$
  - En primer lugar se debe factorizar $x^{15} + 1$:

$$x^{15} + 1 = (x^4 + x + 1) \times (x^4 + x^3 + 1) \times (x^4 + x^3 + x^2 + x + 1) \times (x^2 + x + 1) \times (x + 1)$$

  - Luego, se puede tomar cualquiera de los factores de grado $r = n - k = 4$
  - Por caso, en el último ejemplo desarrollado usamos como polinomio generador $(x^4 + x + 1)$

Ejemplo

- Verifiquemos que al rotar tres lugares a derecha el mensaje **T(x)** del último ejemplo calculado se sigue verificando la propiedad:

```
1101011011111100 >> 3 =  1001101011011111
                        ⊕10011
                         00000010110
                              ⊕10011
                               0010111
                               ⊕10011
                                0010011
                                ⊕10011
                                 00000 ☑
```

Relación entre CRC y paridad

- **CRC** se distingue de los métodos antes vistos de detección de errores ya que a priori parece no depender de la incorporación de bits de paridad entre medio de los bits del mensaje
- No obstante, sigue existiendo un vínculo entre **CRC** y paridad:
  - Agregar un bit de paridad equivale a aplicar el código **CRC** usando $x + 1$ como polinomio generador
  - Verificar esta aseveración calculando el patrón de bits a transmitir para $M(x) = 1101$ y $G(x) = 11$

Capacidad de detección

- Analizar la capacidad de detección de errores del código **CRC** no es del todo trivial, requiere un análisis pormenorizado
- Sea $T(x)$ el patrón de bits enviado y sea $T(x) + E(x)$ el patrón recibido
  - En caso de que $E(x) = 0$ hablamos de una transmisión sin errores
  - Caso contrario, cuando $E(x) \neq 0$, la mayor y menor potencia de x marcarán el comienzo y el fin de la ráfaga en error

Capacidad de detección

- Como por construcción **G(x)** divide a **T(x)**, la capacidad de detección del código **CRC** pivota en que **G(x)** no divida a **E(x)**
  - Por caso, para un error en ráfaga de **p** bits comenzando a partir de la posición **q**, se verifica que $E(x) = x^q(x^{p-1} + \ldots + 1)$
  - Nótese que en el factor de la derecha pueden faltar términos, salvo el primero y el último
- Analicemos por casos lo que sucede para distintos valores de **p**

Capacidad de detección

- Error en ráfaga de longitud **p ≤ r**:
  - En este caso no hay forma de que **G(x)** divida exactamente a **E(x)** puesto que no divide de manera exacta a ninguno de los dos términos que lo componen
  - Es decir, las ráfagas en error de una longitud menor o igual al grado del polinomio generador son siempre detectadas

Ejemplo

- Retomando el ejemplo anterior, toda ráfaga en error de longitud **4** o menor será detectada. Por caso, para $E(x) = x^{10}(x^3 + 1)$ vimos que $T(x) + E(x)$ no resulta divisible por $G(x)$:

```
1101011011111100 → 1001111011111100
                  ⊕10011
                   0000011011
                     ⊕10011
                      010001
                     ⊕10011
                       00010110
                        ⊕10011
                         001010✗
```

Capacidad de detección

- Error en ráfaga de longitud **p = r + 1**:
  - En este caso, como estamos calculando el cociente entre polinomio de igual grado, **G(x)** dividirá a **E(x)** únicamente cuando **G(x)** coincida con el factor de la derecha de **E(x)**
  - En este escenario, ¿cuál será la probabilidad de no detectar el error? Pues bien, para que coincidan los **r + 1** en error, basta con que coincidan los **r − 1** bits internos (ya que en el polinomio generador, al igual que en **E(x)**, el primer y el último bit deben ser **1**)
  - Es decir, **CRC** falla en sólo **1** de las 2^{r-1} posibilidades

Ejemplo

- Analicemos el caso en que no se detecta la ráfaga en error de longitud **5**. Por caso, considerando que $E(x) = x^5(x^4 + x + 1)$:

$$
\begin{array}{l}
11010110111100 \rightarrow\ 11010010011100 \\
\quad \oplus 10011 \\
\quad\ 010010 \\
\quad \oplus 10011 \\
\quad\ 000011000 \\
\quad\quad \oplus 10011 \\
\quad\quad\ 010111 \\
\quad\quad \oplus 10011 \\
\quad\quad\ 0010011 \\
\quad\quad \oplus 10011 \\
\quad\quad\ 0000000 \checkmark
\end{array}
$$

Capacidad de detección

- Error en ráfaga de longitud **p > r + 1**:
  - En este caso la situación es análoga, debemos analizar bajo qué condiciones **G(x)** divide exactamente al término de la derecha de **E(x)**
  - Este análisis se reduce a considerar qué sucede con el último paso de la división: para que el resto sea **0** el último resto parcial debe coincidir con **G(x)**
  - Esto equivale a que coincidan sólo los primeros **r** bits de **G(x)** con ese último resto, ya que el bit más significativo de **G(x)** es necesariamente **1**

Ejemplo

- Analicemos el caso en que no se detecta una ráfaga en error de longitud mayor a **5**. Por caso, sea $E(x) = x(x^{12} + x^9 + x^2 + 1)$:

$$
\begin{array}{l}
1101011011111100 \rightarrow\ 1001111011110110 \\
\quad \oplus 10011 \\
\quad\ \ 0000011011 \\
\quad\quad \oplus 10011 \\
\quad\quad\ \ 010001 \\
\quad\quad\ \oplus 10011 \\
\quad\quad\quad\ 00010011 \\
\quad\quad\quad \oplus 10011 \\
\quad\quad\quad\ \ 000000 \checkmark
\end{array}
$$

Capacidad de detección

- En síntesis, la capacidad de detección de ráfagas en error de un código **CRC** que haga uso de un polinomio generador de grado **r** es:
  - Detecta con una probabilidad del **100%** a las ráfagas en error de longitud $p \leq r$
  - Detecta con una probabilidad de $1 - 2^{-(r-1)}$ a las ráfagas en error de longitud $p = r + 1$
  - Detecta con una probabilidad de $1 - 2^{-r}$ a las ráfagas en error de cualquier otra longitud

Ejemplo

- Consideremos la capacidad de detección de los códigos **CRC-12** y **CRC-CCITT**:
  - **CRC-12** detecta el **100%** de las ráfagas en error de longitud **12** o menor, pero también detecta el **99.91%** de las ráfagas de longitud **13** y el **99.96%** de las restantes ráfagas en error
  - Pero adoptando **CRC-CCITT** en vez de **CRC-12**, la situación mejora notablemente, ahora se detecta el **100%** de la ráfagas en error de longitud **16** o menor, pero también el **99.994%** de las ráfagas de longitud **17** y el **99.997%** de las restantes

Funciones hash

- Si bien la capacidad de detección del código **CRC** es directamente proporcional al grado del polinomio generador que se esté usando, no se ha popularizado el uso de polinomios de grado mayor a 32
  - El uso de polinomios generadores de grado tan alto degrada el desempeño
- De hacer falta un mayor niveles de detección se puede hacer uso de funciones hash tales como **MD5** o **SHA-1**

¿Preguntas?