

Redes

Laboratorio 2 - Esquemas de detección y corrección

Escenarios de pruebas

Sin errores

Enviar un mensaje al emisor, copiar el mensaje generado por este y proporcionarlo tal cual al receptor, el cual debe mostrar el mensajes originales (ya que ningún bit sufrió un cambio). Realizar esto para tres mensajes distintos con distinta longitud.

Hamming (m,n)

No. Mensaje	Mensaje original	Mensaje con bits de paridad	Output
1	1100	1100001	1100
2	10011	110011111	10011
3	011000	0111001010	011000

No. Mensaje	Emisor	Receptor
1	<pre>-Labatorio-2/Corrección de Errores/emisor.py" Introduce el mensaje: 1001 - Mensaje original: 1001 -Número de bits en la cadena: 4 -Número de bits de paridad: 3 Mensaje con bits de paridad: 10001ppp Mensaje con bits de paridad: 1001100 (base) jime@MacBook-Pro-de-Jime-2 Redes-Labatorio-2 %</pre>	<pre>Ingrese un mensaje en binario: 1100001 Message '1100001' has 4 parity bits No errors found Resultado: 1100 (base) jime@MacBook-Pro-de-Jime-2 Receptor %</pre>
2	<pre>aboratorio-2/Corrección de Errores/emisor.py" Introduce el mensaje: 10011 - Mensaje original: 10011 -Número de bits en la cadena: 5 -Número de bits de paridad: 4 Mensaje con bits de paridad: 1p001p1ppp Mensaje con bits de paridad: 110011111</pre>	<pre>Ingrese un mensaje en binario: 110011111 Message '110011111' has 5 parity bits No errors found Resultado: 10011 (base) jime@MacBook-Pro-de-Jime-2 Receptor %</pre>
3	<pre>aboratorio-2/Corrección de Errores/emisor.py" Introduce el mensaje: 011000 - Mensaje original: 011000 -Número de bits en la cadena: 6 -Número de bits de paridad: 4 Mensaje con bits de paridad: 01p00p0ppp Mensaje con bits de paridad: 0111001010</pre>	<pre>Ingrese un mensaje en binario: 0111001010 Message '0111001010' has 5 parity bits No errors found Resultado: 011000 (base) jime@MacBook-Pro-de-Jime-2 Receptor %</pre>

CRC-32

No. Mensaje	Mensaje original	Mensaje con checksum de 32 bits	Output
1	1100	1100101101011110 0101110010010010 0000	1100
2	10011	1001100011001101 0000001111100100 10010	10011
3	011000	0110001011110010 0110100100101100 110011	011000

No. Mensaje	Emisor	Receptor
1	<pre>PS C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\Emisor> go run . Ingrese un mensaje en binario: 1100 Mensaje codificado con CRC-32: 11000101011110010111001001000000 PS C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\Emisor> [</pre>	<pre>PS C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\receptor> & C:/Python312/python.exe <C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\receptor.py Ingresar un mensaje en binario: 11000101011110010111001001000000 No se detectaron errores. Mensaje original: 1100 PS C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\receptor> [</pre>
2	<pre>PS C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\Emisor> go run . Ingresar un mensaje en binario: 1001 Mensaje codificado con CRC-32: 100110001101010000001111001001001010 PS C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\Emisor> [</pre>	<pre>PS C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\receptor> & C:/Python312/python.exe <C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\receptor.py Ingresar un mensaje en binario: 100110001101010000001111001001001010 No se detectaron errores. Mensaje original: 1001 PS C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\receptor> [</pre>
3	<pre>Ingresar un mensaje en binario: 011000 Mensaje codificado con CRC-32: 011000101111001001100100100100100110011001 PS C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\Emisor> & C:/Python312/python.exe <C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\receptor.py</pre>	<pre>Ingresar un mensaje en binario: 011000101111001001100100100100100110011001 No se detectaron errores. Mensaje original: 011000 PS C:\Users\marka\Coding\UAG\Redes\Laboratorios\Redes-Laboratorio-2\Deteción de errores\receptor> [</pre>

Con un error

Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar un bit cualquiera antes de proporcionarlo al receptor. Si el algoritmo es de detección debe mostrar que se detectó un error y que se descarta el mensaje. Si el algoritmo es de corrección debe corregir el bit, indicar su posición y mostrar el mensaje original. Realizar esto para tres mensajes distintos con distinta longitud.

Hamming (m,n)

No. Mensaje	Mensaje original	Mensaje con bits de paridad	Cambio (añadiendo error)	Output
4	0011	0011110	0011111	0011
5	11001	111001101	101001101	11001
6	10010000	100100001001	100100001011	10010000

No. Mensaje	Emisor	Receptor
4	Introduce el mensaje: 0011 - Mensaje original: 0011 -Número de bits en la cadena: 4 -Número de bits de paridad: 3 Mensaje con bits de paridad: 0011110 Mensaje con bits de paridad: 0011110 (base) jime@MacBook-Pro-de-Jime-2 Receptor %	(base) jime@MacBook-Pro-de-Jime-2 Receptor % go run . Ingrese un mensaje en binario: 0011111 Message '0011111' has 4 parity bits Error in position: 1 Resultado: 0011 (base) jime@MacBook-Pro-de-Jime-2 Receptor %
5	Introduce el mensaje: 11001 - Mensaje original: 11001 -Número de bits en la cadena: 5 -Número de bits de paridad: 4 Mensaje con bits de paridad: 1p100p1pp Mensaje con bits de paridad: 111001101 (base) jime@MacBook-Pro-de-Jime-2 Receptor %	(base) jime@MacBook-Pro-de-Jime-2 Receptor % go run . Ingrese un mensaje en binario: 101001101 Message '101001101' has 5 parity bits Error in position: 8 Resultado: 11001 (base) jime@MacBook-Pro-de-Jime-2 Receptor %
6	Introduce el mensaje: 10010000 - Mensaje original: 10010000 -Número de bits en la cadena: 8 -Número de bits de paridad: 4 Mensaje con bits de paridad: 1001p000p0pp Mensaje con bits de paridad: 100100001001 (base) jime@MacBook-Pro-de-Jime-2 Receptor %	(base) jime@MacBook-Pro-de-Jime-2 Receptor % go run . Ingrese un mensaje en binario: 100100001011 Message '100100001011' has 5 parity bits Error in position: 2 Resultado: 10010000 (base) jime@MacBook-Pro-de-Jime-2 Receptor %

CRC-32

No. Mensaje	Mensaje original	Mensaje con checksum de 32 bits	Cambio (añadiendo error)	Output
4	0011	0011011000101 0000111110001 0110100101	0001011000101 0000111110001 0110100101	Se detectaron errores y el mensaje se descarta.
5	11001	1100110111000 0000011100101 01010110110	0100110111000 0000011100101 01010110110	Se detectaron errores y el mensaje se descarta.
6	10010000	1001000000110 0011100001010 1001000010110 1	1001000000110 0001100001010 1001000010110 1	Se detectaron errores y el mensaje se descarta.

No. Mensaje	Emisor	Receptor
4	PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2\Detec Ingrese un mensaje en binario: 0011 Mensaje codificado con CRC-32: 00110110001010000111100010110100101	Mensaje codificado con CRC-32: 11001101110000000111001010101011010 PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2\Detec on.exe "c:/Users/marka/Coding/UVG/Redes/Laboratorios/Redes-Laborato Ingresar un mensaje en binario: 00010110001010000111100010110100101 Se detectaron errores y el mensaje se descarta.
5	Ingresar un mensaje en binario: 11001 Mensaje codificado con CRC-32: 11001101110000000111001010101011010 PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2\Detec on de errores\Emisor> & C:/Python312/pyt	on.exe "c:/Users/marka/Coding/UVG/Redes/Laboratorios/Redes-Laboratorio-2\Detec on de errores\Receptor> & C:/Python312/pyt Ingresar un mensaje en binario: 00010110001010000111100010110100101 Se detectaron errores y el mensaje se descarta.
6	PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2\Detec Ingresar un mensaje en binario: 10010000 Mensaje codificado con CRC-32: 1001000000110001110000101010010000101101	PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2\Detec on.exe "c:/Users/marka/Coding/UVG/Redes/Laboratorios/Redes-Laboratorio-2\Det Ingresar un mensaje en binario: 1001000000110000110000101010010000101101 Se detectaron errores y el mensaje se descarta.

Con varios errores

Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar dos o más bits cualesquiera antes de proporcionarlo al receptor. Si el algoritmo es de detección debe mostrar que se detectó un error y que se descarta el mensaje. Si el algoritmo es de corrección y puede corregir más de un error, debe corregir los bits, indicar su posición y mostrar el mensaje original. Realizar esto para tres mensajes distintos con distinta longitud.

Hamming (m,n)

No. Mensaje	Mensaje original	Mensaje con bits de paridad	Cambio (añadiendo 2 errores)	Output
7	1010	1010010	1010001	1011
8	101110	1011111010	0011111011	001110
9	100100100	1001000100001	0111000100001	011000100

No. Mensaje	Emisor	Receptor
7	<pre>Introduce el mensaje: 1010 - Mensaje original: 1010 -Número de bits en la cadena: 4 -Número de bits de paridad: 3 Mensaje con bits de paridad: 1010p0pp Mensaje con bits de paridad: 1010010</pre>	<pre>(base) jime@MacBook-Pro-de-Jime-2 Receptor % go run . Ingresé un mensaje en binario: 1010001 Message '1010001' has 4 parity bits Error in position: 3 Resultado: 1011 (base) jime@MacBook-Pro-de-Jime-2 Receptor %</pre>
8	<pre>2/Corrección de Errores/emisor.py" Introduce el mensaje: 101110 - Mensaje original: 101110 -Número de bits en la cadena: 6 -Número de bits de paridad: 4 Mensaje con bits de paridad: 10p111p0pp Mensaje con bits de paridad: 101111010</pre>	<pre>Ingresé un mensaje en binario: 0011111011 Message '0011111011' has 5 parity bits Error in position: 11 Error in parity bits, can't correct Resultado: 001110</pre>
9	<pre>2/Corrección de Errores/emisor.py" Introduce el mensaje: 100100100 - Mensaje original: 100100100 -Número de bits en la cadena: 9 -Número de bits de paridad: 4 Mensaje con bits de paridad: 10010p010p0pp Mensaje con bits de paridad: 1001000100001</pre>	<pre>(base) jime@MacBook-Pro-de-Jime-2 Receptor % go run . Ingresé un mensaje en binario: 0111000100001 Message '0111000100001' has 5 parity bits Error in position: 10 Resultado: 011000100 (base) jime@MacBook-Pro-de-Jime-2 Receptor %</pre>

CRC-32

No. Mensaje	Mensaje original	Mensaje con checksum de 32 bits	Cambio (añadiendo 2 errores)	Output
7	1010	1010101011010 0111100100100 1001010110	1011101011010 0111100100100 1001110110	Se detectaron errores y el mensaje se descarta.
8	101110	1011100101001 0011111111110 1010010100000	1011100101001 0011111111110 1010010101111	Se detectaron errores y el mensaje se descarta.
9	100100100	1001001001011 0000111101110 0101011110001 11	1001001001011 0011111101110 0101011110001 11	Se detectaron errores y el mensaje se descarta.

No. Mensaje	Emisor	Receptor
7	PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2> Ingrese un mensaje en binario: 1010 Mensaje codificado con CRC-32: 10101010101001111001001001010110	PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2> on.exe "c:/Users/marka/Coding/UVG/Redes/Laboratorios/Redes-Laboratorio-2/De Ingrese un mensaje en binario: 101110101101001111110010010010110110 Se detectaron errores y el mensaje se descarta.
8	Ingrese un mensaje en binario: 101110 Mensaje codificado con CRC-32: 10111001010010011111111101010010000 PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2\De h	on.exe "c:/Users/marka/Coding/UVG/Redes/Laboratorios/Redes-Laboratorio-2/De Ingrese un mensaje en binario: 10111001010010011111111101010010111 Se detectaron errores y el mensaje se descarta. PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2\De
9	PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2> Detección de errores % cd Emisor % go run . Ingrese un mensaje en binario: 100100100 Mensaje codificado con CRC-32: 10010010010110000111101110010101111000111	PS C:\Users\marka\Coding\UVG\Redes\Laboratorios\Redes-Laboratorio-2> on.exe "c:/Users/marka/Coding/UVG/Redes/Laboratorios/Redes-Laboratorio-2/De Ingrese un mensaje en binario: 10010010010110000111101110010101111000111 Se detectaron errores y el mensaje se descarta.

Prueba de tiempo para emisores

(Hamming) Tiempo de ejecución: 0.000269 s

(CRC-32) Tiempo de ejecución de MakeCrcTable: 0.000012792s

```
Introduce el mensaje: 1100
- Mensaje original: 1100
-Número de bits en la cadena: 4
-Número de bits de paridad: 3
Mensaje con bits de paridad: 110p0pp
Mensaje con bits de paridad: 1100001
Tiempo de ejecución: 0.000269 segundos

(base) jime@MacBook-Pro-de-Jime-2 Redes-Laboratorio-2 % cd Detección\ de\ er  

(base) jime@MacBook-Pro-de-Jime-2 Detección de errores % cd Emisor
(base) jime@MacBook-Pro-de-Jime-2 Emisor % go run .
Ingrese un mensaje en binario:
1100
Tiempo de ejecución de MakeCrcTable: 12.792µs
Mensaje codificado con CRC-32: 11001011010111001011100100100000
(base) jime@MacBook-Pro-de-Jime-2 Emisor %
```

Preguntas adicionales

- ¿Es posible manipular los bits de tal forma que el algoritmo seleccionado no sea capaz de detectar el error? ¿Por qué sí o por qué no? En caso afirmativo, demuestrelo con su implementación.

Considerando que debe mantenerse el mensaje y no detectar errores, para CRC-32, con la implementación realizada y los ejemplos proporcionados, no es posible de realizarse. Esto se debe a que este algoritmo es determinístico, por lo que no es posible generar un segundo checksum que sea válido para un mismo mensaje (*Faster CRC32-C Computation In MySQL 8.0.27*, s. f.). Si lo que se desea es cambiar el mensaje, pero manteniendo el mismo checksum, un método de realizar esto sería explotando la vulnerabilidad de *Hash Collision*, el cuál significa que es posible que dos inputs diferentes resulten en el mismo checksum generado (Wikipedia contributors, 2024). Esto se puede demostrar con fuerza bruta¹. Por ejemplo, en la cadena de entrada “1100” se encontró una colisión con la cadena “kIfR8”, ya que ambas resultan en el checksum “10110101110010111001001001000000”.

Con el algoritmo de Hamming, cambiando el mensaje, pero manteniendo el mensaje sin errores, la estrategia que se podría tomar es buscar las combinaciones que den 0 para que el mensaje se muestre sin errores a pesar de que este está cambiado, por ejemplo si se ingresa 1100 esto generaría 1100001 y si al receptor se le enviará 0011110 es decir los bits al contrario, el mensaje no sería el mismo pero no se mostrarían errores en este caso mostraría 0011. En el caso de mantener el mismo mensaje manipulando los bits sería complicado ya que esta manipulación mostraría que se generan errores en el mensaje original.

- En base a las pruebas que realizó, ¿qué ventajas y desventajas posee cada algoritmo con respecto a los otros dos? Tome en cuenta complejidad, velocidad, redundancia (overhead), etc.

Hamming

Para nosotros fue más sencillo entender este algoritmo en comparación al CRC-32, ya que las operaciones eran considerablemente más sencillas que las del otro algoritmo. Además, el poder corregir un bit de error es una gran ventaja en comparación al CRC-32.

Se nos dificultó más las verificaciones de este algoritmo en comparación al segundo, ya que la cantidad de bits de paridad depende de la longitud del mensaje, por lo que en cadenas más largas nos fue más complicado visualizar y encontrar los bits de paridad para poder verificar nuestros resultados.

Además, aunque no fue algo tan perceptible para nosotros, nos dimos cuenta que este algoritmo es alrededor de 20 veces más lento que el CRC-32, con tiempos de ejecución de

¹ El código de ejemplo se adjunta en el repositorio.

0.000269 s. Esta diferencia se le puede atribuir al propio algoritmo como tal, ya que su tiempo de ejecución teórico es de $O(n \cdot (m + n))$ mientras que el del segundo algoritmo es de $O(n)$.

CRC-32

En la implementación de CRC-32 nos dimos cuenta, que este algoritmo era mejor realizarlo de manera que usará código ascii, lo que haría que funcionara con las cadenas de binarios basándonos en el ejemplo de The LXP32 Processor, esto hizo que se nos dificultaría mucho más la comprensión del algoritmo ya que era necesario saber la variante de polinomio que se observa a continuación.

$$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1.$$

Consideramos que esto es una desventaja comparado a Hamming que es más fácil de comprender y que no requiere de polinomios. Otra desventaja es que con este algoritmo lo único que logramos hacer fue descartar el mensaje en el caso de que hubiesen errores, lo que consideramos que no es práctico.

Entre las ventajas que encontramos es la velocidad ya que el emisor tardó únicamente 0.000012792s a comparación de Hamming que fue más tardado con 0.000269 s. También observamos que con este algoritmo al tener un número fijo (checksum) podemos evitar el overhead en comparación con Hamming ya que el número de bits puede ser mayor a este número.

Referencias

Wikipedia contributors. (2024, 20 junio). Hash collision. Wikipedia. https://en.wikipedia.org/wiki/Hash_collision

Faster CRC32-C computation in MySQL 8.0.27. (s. f.). Oracle Blogs. Recuperado 23 de julio de 2024, de <https://blogs.oracle.com/mysql/post/faster-crc32-c-computation-in-mysql-8027>

A simple example: CRC32 calculation – The LXP32 Processor. (s. f.). <https://lxp32.github.io/docs/a-simple-example-crc32-calculation/>