UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Guatemala 25 de Julio de 2024
Jimena Hernandez - 21199
Mark Albrand - 21004

Redes

Laboratorio 2 - Esquemas de detección y corrección

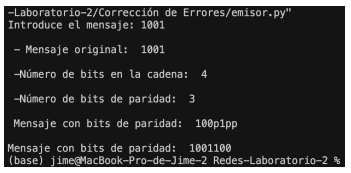
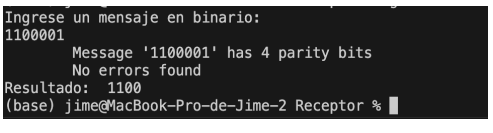
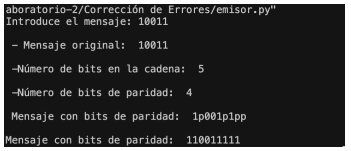
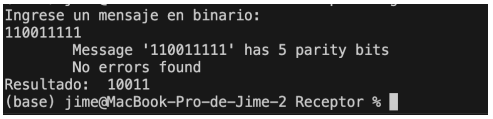
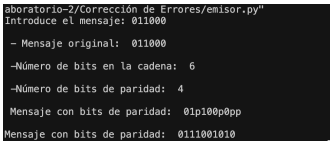
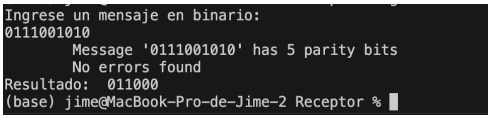
Escenarios de pruebas

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.

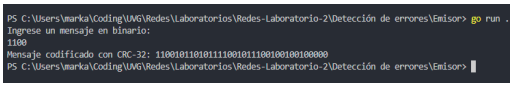
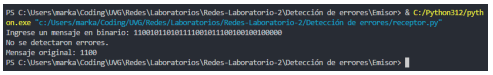
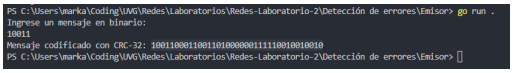
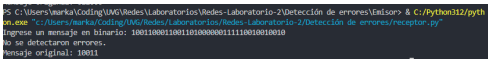
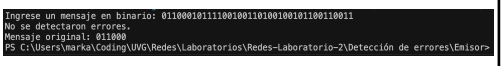
Hamming (m,n)

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		
2		
3		

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

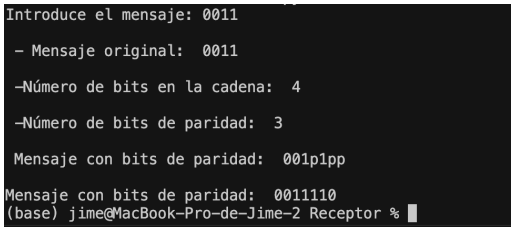
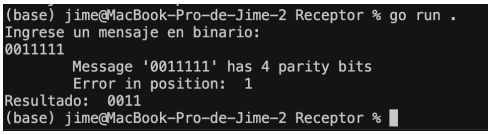
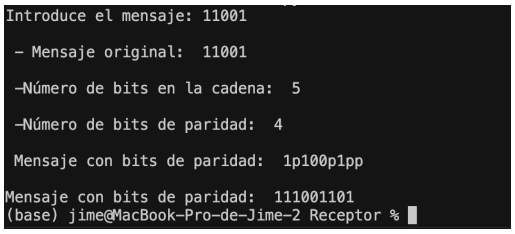
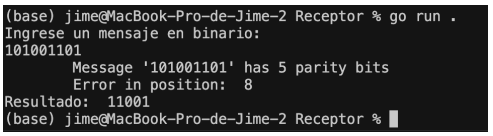
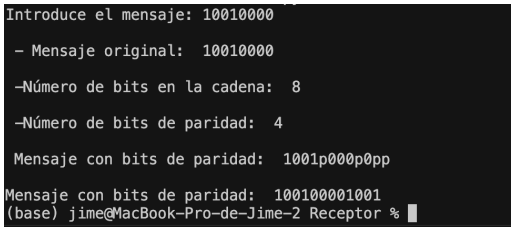
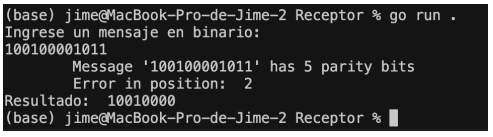
No. Mensaje	Emisor	Receptor
1		
2		
3		

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.

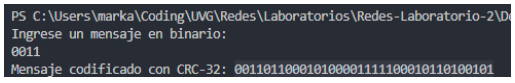
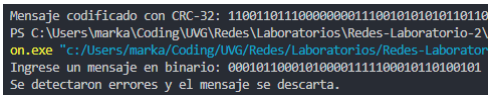
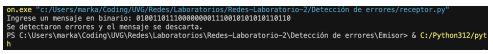
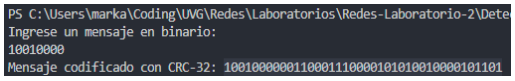
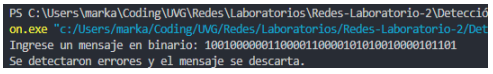
Hamming (m,n)

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		
5		
6		

CRC-32

No. Mensaje	Mensaje original	Mensaje con checksum de 32 bits	Cambio (añadiendo error)	Output
4	0011	00110110001010000011111000101101001010	00010110001010000011111000101101001010	Se detectaron errores y el mensaje se descarta.
5	11001	11001101110000000011100101010110110	01001101110000000011100101010110110	Se detectaron errores y el mensaje se descarta.
6	10010000	10010000001100011100001010100100001011011	10010000001100001100001010100100001011011	Se detectaron errores y el mensaje se descarta.

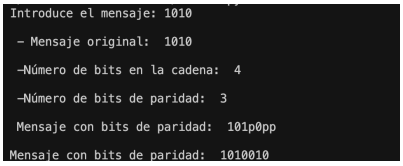
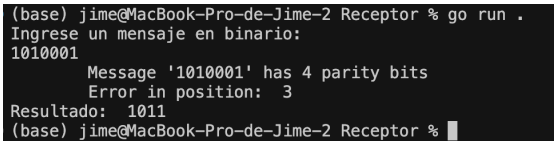
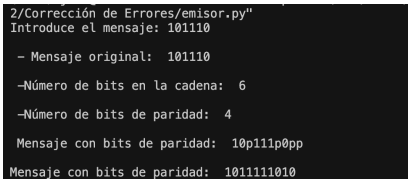
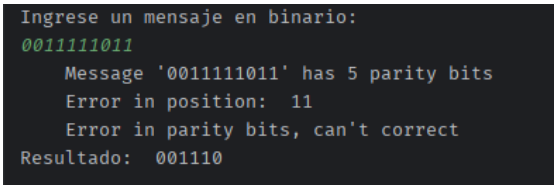
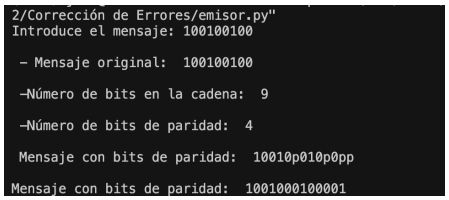
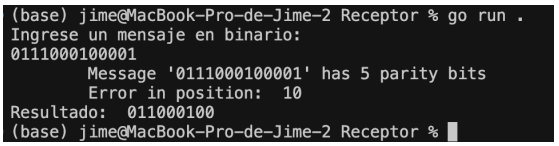
No. Mensaje	Emisor	Receptor
4		
5		
6		

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 detecto 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.

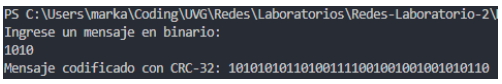
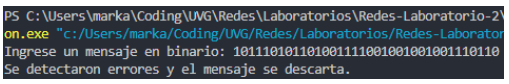
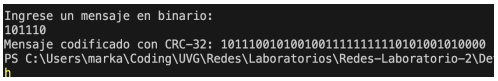
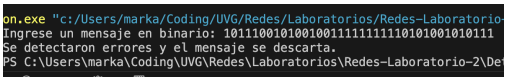
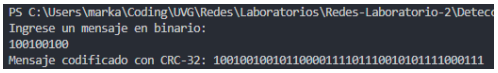
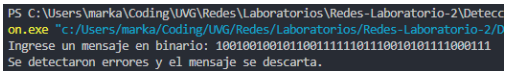
Hamming (m,n)

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		
8		
9		

CRC-32

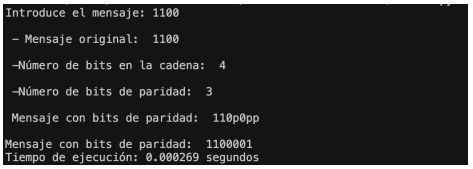
No. Mensaje	Mensaje original	Mensaje con checksum de 32 bits	Cambio (añadiendo 2 errores)	Output
7	1010	10101010110100111100100100100101010110	10111010110100111100100100100111010110	Se detectaron errores y el mensaje se descarta.
8	101110	10111001010010011111111110101001010000	10111001010010011111111110101001010111	Se detectaron errores y el mensaje se descarta.
9	100100100	10010010010110000111101110010101111000111	10010010010110011111101110010101111000111	Se detectaron errores y el mensaje se descarta.

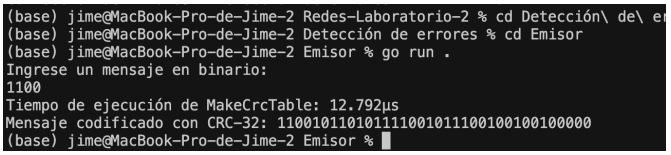
No. Mensaje	Emisor	Receptor
7		
8		
9		

Prueba de tiempo para emisores

(Hamming) Tiempo de ejecución: 0.000269 s

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





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 bruta[1]. Por ejemplo, en la cadena de entrada “1100” se encontró una colisión con la cadena “klfR8”, ya que ambas resultan en el checksum “10110101111001011100100100100000”.

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 dificultará 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/