**Laboratorio #2 – Parte 1**

*Esquemas de detección y corrección*

Escenarios de prueba:

*Algoritmo Hamming(n,m) (Emisor en Ruby, Receptor en Python)*

* 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.
  1. Input: 1011

A black screen with white text

Description automatically generated

* 1. Input: 11001010

A black screen with white text

Description automatically generated

* 1. Input: 11011010101

A computer screen with white text

Description automatically generated

* 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 detecto 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.
  1. Input: 1011, cambiando último bit.

A black screen with white text

Description automatically generated

* 1. Input: 11001010, cambiando bit en posición 9.

A computer screen with white text

Description automatically generated

* 1. Input: 11011010101, cambiando bit en posición 1.

A screenshot of a computer program

Description automatically generated

* 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
  1. Input: 1010, cambiando bits en posiciones 1 y 4.

A black screen with white text

Description automatically generated

* 1. Input: 001100, cambiando bits en posiciones 2 y 3.

A black screen with white text

Description automatically generated

* 1. Input: 11010101, cambiando bits en posiciones 6 y 7.

A black screen with white text

Description automatically generated

*Algoritmo Fletcher Checksum (Emisor en Ruby, Receptor en Python)*

* 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.
  1. Input: 1011

A screen shot of a computer code

Description automatically generated

* 1. Input: 11001010

A computer screen with white text

Description automatically generated

* 1. Input: 11011010101

A computer screen with white text

Description automatically generated

* 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.
  1. Input: 1011, cambiando último bit.

A computer screen with white text

Description automatically generated

* 1. Input: 11001010, cambiando bit en posición 9.

A computer screen shot of a black screen

Description automatically generated

* 1. Input: 11011010101, cambiando bit en posición 1.

A computer screen with white text

Description automatically generated

* 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
  1. Input: 1010, cambiando bits en posiciones 1 y 4.

A computer screen shot of a computer code

Description automatically generated

* 1. Input: 001100, cambiando bits en posiciones 2 y 3.

A computer screen with white text

Description automatically generated

* 1. Input: 11010101, cambiando bits en posiciones 6 y 7.

A computer screen with white text

Description automatically generated

Preguntas:

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

En el caso de Hamming, si es posible que no detecte todos los errores si se cambian dos o más bits. Ya que la version estándar puede corregir un solo error y detectar hasta dos errores, pero no puede corregir más de un error ni detectar tres errores o más.

Ejemplo:

A computer screen with white text

Description automatically generated

El código detectó y corrigió un solo error (en la posición 5). Sin embargo se cambiaron dos bits, en las posiciones 2 y 5, al cambiar dos bits, el código puede fallar en la detección correcta, ya que solo está diseñado para corregir un error y detectar hasta dos errores. Y aún con esos dos cambios se logra engañar al Código para que regrese el mensaje correcto.

Por otro lado, en el algoritmo de Fletcher, teóricamente es posible manipular los bits de tal forma que el algoritmo no detecte el error. Aunque sea más complejo que un chequeo de paridad también tiene limitaciones en cuanto a la detección de ciertos patrones de errores. Existen errores que pueden cambiar el checksum de manera que el receptor no detecte la modificación, especialmente si los cambios afectan al mensaje en un patrón que se alinea con el cálculo del checksum.

Intentamos en múltiples ocasiones modificar el checksum para que poder demostrar esto pero no fue posible, consideramos que esto fue porque engañar al algoritmo requiere manipular los datos de manera que el checksum final se mantenga igual, lo cual es difícil de lograr sin alterar significativamente el mensaje.

* 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 (n,m)

|  |  |
| --- | --- |
| **Ventajas** | Desventajas |
| Puede detectar y corregir errores de un solo bit y detectar errores de dos bits. | La cantidad de bits de paridad necesarios puede ser significativa, aumentando el overhead. |
| Detecta errores complejos en comparación con el bit de paridad. | Más complicado de implementar debido a la necesidad de calcular y ubicar bits de paridad. |
| La redundancia está controlada y depende del tamaño del mensaje. | Limitado a solamente corregir 1 error en muchas ocasiones. |

Fletcher Checksum

|  |  |
| --- | --- |
| **Ventajas** | Desventajas |
| Adecuado para Datos Grandes: Maneja eficientemente errores en mensajes largos. | Mayor Redundancia: Requiere 16 bits para el checksum, lo que aumenta el overhead. |
| Utiliza una suma acumulativa que mejora la detección de errores complejos. | Más complejo de implementar y calcular en comparación con el bit de paridad. |
| Detecta errores de 1 o más bits. | No es capaz de corregir errores. |
|  |  |

Explicaciones:

* Durante el escenario de prueba 3, de modificar más de 1 bit, en el caso del algoritmo de Hamming noté que solamente encontraba el error en el segundo bit que cambié, sin embargo si lograba decodificar correctamente el mensaje.
* En cambio, con el Fletcher, sin importar cuantos cambiara siempre detectaba el error.

Conclusiones:

* Los códigos de Hamming ofrecen una sólida capacidad para corregir errores y detectar errores múltiples, mientras que el checksum de Fletcher es eficaz para la detección de errores pero no corrige errores.
* El bit de paridad es el más simple de implementar y rápido en ejecución, mientras que el código de Hamming y el checksum de Fletcher son más complejos, con el Hamming siendo el más complejo debido a su capacidad de corrección.
* El checksum de Fletcher puede detectar una amplia variedad de errores en comparación con el bit de paridad, pero el código de Hamming es superior en la corrección de errores individuales y en la detección de errores múltiples en comparación con ambos métodos.

Referencias:

GeeksForGeeks .(2024) Hamming Code in Computer Network. Recuperado de: <https://www.geeksforgeeks.org/hamming-code-in-computer-network/>

Nakassis, A. (2010) Fletcher's Error Detection Algorithm: How to implement it efficiently and how to avoid the most common pitfalls. Recuperado de: <https://dl.acm.org/doi/pdf/10.1145/53644.53648>