Diego Perdomo, 20204
Luis Diego Santos, 20226
Laboratorio 2.1

Esquemas de detección y corrección de errores

Descripción

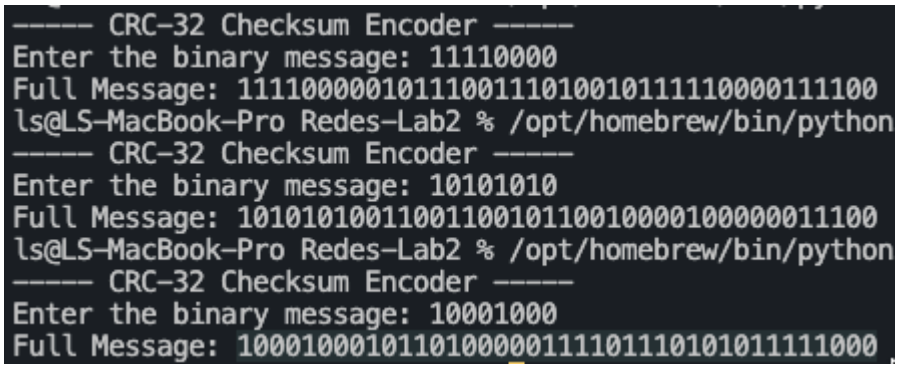
El objetivo de esta práctica es entender, implementar y analizar las ventajas y desventajas de diferentes algoritmos de detección y corrección de errores. Para esto, se nos da una lista de algoritmos de los cuales se eligieron CRC-32 y Hamming. Con estos algoritmos, se implementó un emisor encargado de recibir una trama y codificarla a estándares del algoritmo. También se implementó un receptor que recibe la trama codificada en estándares del algoritmo, con alguno que otro bit erróneo, y este debe (al límite de la capacidad del algoritmo) identificar cuál es el bit erróneo y, en caso de ser un algoritmo de corrección, reemplazarlo por el valor correcto. El emisor y el receptor deben estar implementados en diferentes lenguajes de programación. Después, se lleva a cabo una serie de pruebas para ver el comportamiento de estos algoritmos en presencia de uno o más bits erróneos. Esto es con la finalidad de poder entender y comparar su desempeño y ser capaz de discernir entre las ventajas y desventajas de los diferentes algoritmos y acercamientos.

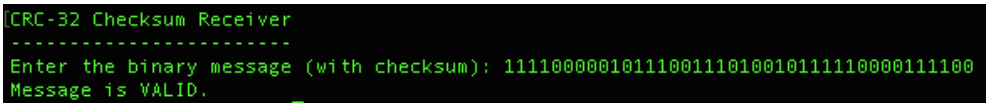
CRC-32 es un algoritmo de detección de errores utilizado comúnmente en comunicación de red y almacenamiento de datos. Consiste en generar un valor de comprobación de longitud fija, típicamente de 32 bits, a partir de los datos que se van a transmitir o almacenar. El remitente agrega este valor de comprobación a los datos y el receptor recalcula el valor de comprobación con los datos recibidos. Si el valor de comprobación calculado coincide con el valor de comprobación transmitido, esto indica que no se han introducido errores durante la transmisión. Si los valores de comprobación no coinciden, el receptor sabe que hay errores presentes y puede solicitar una retransmisión o tomar la acción apropiada. CRC-32 es ampliamente utilizado debido a su eficiencia para detectar una variedad de errores.

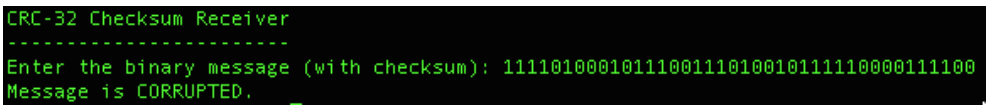
La corrección de errores Hamming es un algoritmo utilizado para detectar y corregir errores en la transmisión de datos. Funciona mediante la adición de bits adicionales (bits de paridad) a los datos, formando una palabra de código. Estos bits de paridad se calculan para garantizar que combinaciones específicas de bits tengan paridad par o impar, dependiendo de

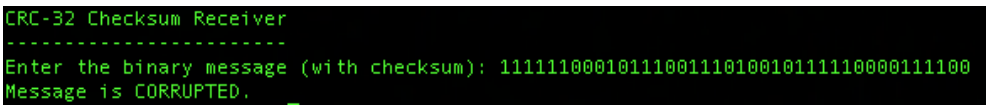
la implementación (en este caso, es par). Esto crea un patrón único para cada palabra de código válida. Cuando el receptor recibe la palabra de código, recalcula los bits de paridad y los compara con los recibidos. Si hay un error de un solo bit, el receptor puede identificar y corregir el bit erróneo al comparar las paridades, pues esta comparación genera un código binario que coincide con la posición absoluta del bit erróneo. Si hay más de un bit con error, el receptor puede detectar la presencia de errores, pero no puede corregirlos ni ubicarlos de manera efectiva.

Resultados con CRC-32

- Emisor (Python)
    - 11110000 - 1111000001011100111010010111110000111100
    - 10101010 - 1010101001100110010110010000100000011100
    - 10001000 - 1000100010110100000111101110101011111000
    - 

- Receptor (C++)
    - 11110000
        - 1111000001011100111010010111110000111100 (Sin modificaciones)
        - 

        - 1111010001011100111010010111110000111100 (Cambio de 1 bit en la posición 6)
        - 

        - 1111110001011100111010010111110000111100 (Cambio de 2 bits en las posiciones 5 y 6)
        - 

- 11111100010111011110100101111110000111100 (Cambio de 3 bits en las posiciones 5, 6, y 16)

```
CRC-32 Checksum Receiver
------------------------
Enter the binary message (with checksum): 11111100010111011110100101111110000111100
Message is CORRUPTED.
```

- 10101010
  - 10101010011001100101100100001000000011100

```
CRC-32 Checksum Receiver
------------------------
Enter the binary message (with checksum): 10101010011001100101100100001000000011100
Message is VALID.
```

  - 10101010011001100101100100001010000011100 (Cambio de 1 bit en la posición 31)

```
CRC-32 Checksum Receiver
------------------------
Enter the binary message (with checksum): 10101010011001100101100100001010000011100
Message is CORRUPTED.
```

  - 01101010011001100101100100001000000011100 (Cambio de 2 bits en las posiciones 1 y 2)

```
CRC-32 Checksum Receiver
------------------------
Enter the binary message (with checksum): 01101010011001100101100100001000000011100
Message is CORRUPTED.
```

  - 01101010011001000101100100001000000011100 (Cambio de 3 bits en las posiciones 1, 2, y 15)

```
CRC-32 Checksum Receiver
------------------------
Enter the binary message (with checksum): 01101010011001000101100100001000000011100
Message is CORRUPTED.
```

- 10001000
  - 10001000101101000001111011101010111111000

```
CRC-32 Checksum Receiver
------------------------
Enter the binary message (with checksum): 10001000101101000001111011101010111111000
Message is VALID.
```

  - 10101000101101000001111011101010111111000 (Cambio de 1 bit en la posición 3)

```
CRC-32 Checksum Receiver
------------------------
Enter the binary message (with checksum): 10101000101101000001111011101010111111000
Message is CORRUPTED.
```

  - 10100000101101000001111011101010111111000 (Cambio de 2 bits en las posiciones 3 y 5)

```
CRC-32 Checksum Receiver
------------------------
Enter the binary message (with checksum): 10100000101101000001111011101010111111000
Message is CORRUPTED.
```

- 10000101101101000001111011101010111111000 (Cambio de 3 bits en las posiciones 5, 6, y 8)

```
CRC-32 Checksum Receiver
------------------------
Enter the binary message (with checksum): 10000101101101000001111011101010111111000
Message is CORRUPTED.
```

Resultados con Hamming

- Emisor (Python)
  - 0101101 - 00001010101
  - 1110010 - 01101101010
  - 0011000 - 10000110000
  -

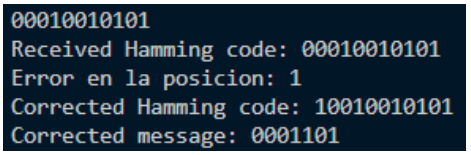
```
PS C:\Users\DIEGO\Documents\GitHub\Redes-Lab2>  c:; cd
s-Lab2'; & 'C:\Users\DIEGO\AppData\Local\Programs\Pyth
GO\.vscode\extensions\ms-python.python-2023.12.0\pytho
HammingS.py'
Ingrese la trama: 0101101
Trama original:  0101101
Trama en código de hamming:  00001010101
PS C:\Users\DIEGO\Documents\GitHub\Redes-Lab2>  c:; cd
s-Lab2'; & 'C:\Users\DIEGO\AppData\Local\Programs\Pyth
GO\.vscode\extensions\ms-python.python-2023.12.0\pytho
.\debugpy\launcher' '59868' '--' 'c:\Users\DIEGO\Docum
HammingS.py'
Ingrese la trama: 1110010
Trama original:  1110010
Trama en código de hamming:  01101101010
PS C:\Users\DIEGO\Documents\GitHub\Redes-Lab2>  c:; cd
s-Lab2'; & 'C:\Users\DIEGO\AppData\Local\Programs\Pyth
GO\.vscode\extensions\ms-python.python-2023.12.0\pytho
.\debugpy\launcher' '59879' '--' 'c:\Users\DIEGO\Docum
HammingS.py'
Ingrese la trama: 0011000
Trama original:  0011000
Trama en código de hamming:  10000110000
PS C:\Users\DIEGO\Documents\GitHub\Redes-Lab2>
```

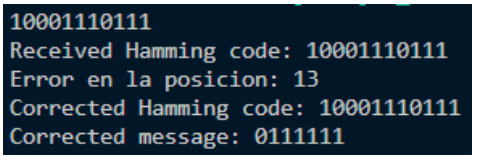
- Receptor (Java)
  - 00001010101 - 0101101
    - 00001010101 - Original
      -

```
00001010101
Received Hamming code: 00001010101
No hay error
Mensaje: 0101101
```

    - **1**0001010101 - Un bit cambiado
      -

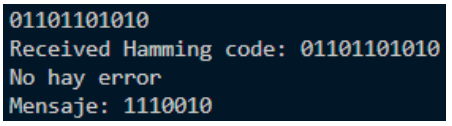
```
10001010101
Received Hamming code: 10001010101
Error en la posicion: 1
Corrected Hamming code: 00001010101
Corrected message: 0101101
```

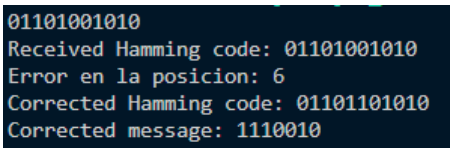
- 000**10**010101 - Al menos dos bits
  - 

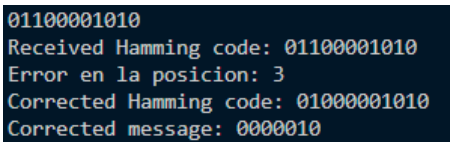
  - Hamming es un algoritmo de corrección con soporte para sólo un bit, por lo que al ingresar más de un bit erróneo, el algoritmo es incapaz de corregir el código recibido y descifrar la trama.
- **1**0001**1**101**1**1 - Indescifrable para Hamming
  - 

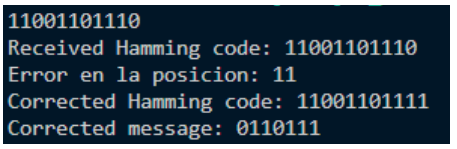
  - Hamming es un algoritmo de corrección con soporte para sólo un bit, por lo que al ingresar más de un bit erróneo, el algoritmo es incapaz de corregir el código recibido y descifrar la trama.

○ 01101101010 - 1110010

- 01101101010 - Original
  - 

- 01101**0**01010 - Un bit cambiado
  - 

- 0110**0**01010 - Al menos dos bits
  - 

- **1**1**0**01101**1**10 - Indescifrable para Hamming
  - 

○ 10000110000 - 0011000

- 10000110000 - Original

```
10000110000
Received Hamming code: 10000110000
No hay error
Mensaje: 0011000
```

- 10000110010 - Un bit cambiado

```
10000110010
Received Hamming code: 10000110010
Error en la posicion: 10
Corrected Hamming code: 10000110000
Corrected message: 0011000
```

- 10000110110 - Al menos dos bits

```
10000110110
Received Hamming code: 10000110110
Error en la posicion: 3
Corrected Hamming code: 10100110110
Corrected message: 1011110
```

- 11100110001 - Indescifrable para Hamming

```
11100110001
Received Hamming code: 11100110001
Error en la posicion: 10
Corrected Hamming code: 11100110011
Corrected message: 1011011
```

Discusión

En las pruebas realizadas con el algoritmo de detección de errores CRC-32, se pudo observar que este presenta un alto nivel de efectividad en la detección de errores en las cadenas de bits utilizadas. Para cada una de las tres cadenas generadas por el programa sender, escrito en Python, se observó que el programa receiver, escrito en C++, identificó correctamente cuáles eran las cadenas válidas y cuáles eran las que habían sido modificadas. Cada cadena se probó con tres escenarios diferentes de pérdida de información; modificación de uno, dos, y tres bits. El algoritmo de CRC-32 identificar los errores en todos los casos, incluso en los que otros algoritmos hubieran fallado.

Tomando como ejemplo la tercera prueba de la segunda cadena de CRC-32, en la cual dos bits, un 0 y un 1, cambiaron de valor, un algoritmo más sencillo como un bit de paridad no hubiera identificado que la cadena tenía un error. Esto se debe a que el bit de paridad únicamente toma en cuenta la cantidad de bits 1 que una cadena tiene, y debido a que la cantidad de 1s se mantiene igual por el cambio de dos bits, el bit de paridad no hubiera detectado el error. (IBM, 2021) Por el otro lado, CRC-32 utiliza una función matemática que toma en cuenta varios aspectos de la cadena para crear una cadena de bits que se utiliza para validar la cadena original.

Aunque este algoritmo de detección de errores presentó muy buenos resultados en las pruebas realizadas, aún tiene ciertas limitaciones que se deben tomar en cuenta al momento de decidir implementarlo. La primera de estas limitaciones es que es un algoritmo muy predecible, lo cual lo hace susceptible a ataques deliberados de modificación de bits. Debido a que se conoce la forma en la que el algoritmo CRC-32 calculó el checksum, es posible modificar la cadena de tal manera que el mensaje original sea irreconocible y el error sea indetectable. (Embedded Office, 2023) Adicionalmente, también es importante considerar que es únicamente un algoritmo de detección de errores, lo cual significa que el mensaje tiene que ser reenviado en su totalidad ya que no hay manera de corregir el error del lado del receptor.

Por otro lado, en el caso del algoritmo de codificación y corrección de Hamming, nos topamos con un acercamiento más proactivo pero mucho más limitado. Una de las primeras diferencias que nos topamos es que, a diferencia de CRC-32, Hamming utiliza una cantidad de bits redundantes en función de la trama enviada originalmente. Esto permite una mayor adaptabilidad al error, en caso de existir, en diferentes tramas de diferentes tamaños mientras que utiliza de manera eficiente el espacio para almacenar la trama codificada (*Error detecting and error correcting codes*, 1950). Otro punto fuerte del algoritmo de Hamming es su capacidad de ubicar y corregir el error en la cadena recibida de una manera rápida y eficiente, pues la cadena diferencial entre los bits de paridad recibidos y calculados nos anexa a la posición del bit erróneo.

Teniendo en cuenta lo eficiente que es Hamming, cabe resaltar que tiene una limitación muy fuerte en cuanto a su capacidad de corrección. Mientras exista un sólo error en la trama recibida, este algoritmo será capaz de corregirlo usando el índice generado con la comparación de bits de paridad generados y recibidos. Pero, si existe más de un bit corrupto en la cadena, Hamming se vuelve incapaz de corregir el error en esta, y su capacidad se limita a indicar que existe un error; ni siquiera es seguro que se ubique este de manera correcta (Tanenbaum et al., 2012). Incluso, en los casos más extremos, cabe la posibilidad de ni siquiera detectar la presencia de un error en la cadena debido a la posibilidad de que los bits erróneos converjan en una cadena válida según el cálculo de paridad.

Con esto, notamos que la comparación entre estos dos acercamientos se reduce a una comparación entre robustez y eficiencia. El uso de estos dependerá más del contexto en el que se desea utilizar. Si buscamos una garantía más alta en cuanto a la fidelidad del mensaje y la capacidad de detección de errores, CRC-32 es el acercamiento más apropiado. Por otro lado, si el contexto es más dinámico y se necesita algo más eficiente y que asegure que la información se estará transmitiendo con una fidelidad al límite de la capacidad del algoritmo, Hamming tiene más sentido.

Comentario

En la actualidad, tomamos por asegurado que la información que transmitimos por redes como el WiFi o el internet llegará a su destino sin ningún problema. Sin embargo, es inevitable que se den errores en esta transmisión de información al tomar en cuenta la cantidad de bits de la cual se está hablando. Tomando como referencia una película de 1GB, lo cual corresponde a 8,000,000,000 bits, un porcentaje de error de transmisión de 0.1% aún corresponde a un error de 8 millones de bits. Al considerar este aspecto de las redes que utilizamos todos los días, se hace evidente la importancia de los algoritmos de detección y corrección de errores. Algoritmos como CRC-32 y Hamming ayudan a asegurar que nuestra información llega a su destino sin problema y sin que se den pérdidas de información.

El algoritmo de detección de errores CRC-32 puede no resultar tan útil como Hamming por el simple hecho de que no puede corregir estos errores, pero aún resulta extremadamente útil para determinar la integridad de la información que se está mandando. Esto puede parecer irrelevante, pero al momento que se está mandando información delicada o muy importante, se hace evidente que la pérdida de un solo bit de información puede ser catastrófica. Aunque en un mundo ideal todas las redes de comunicación deberían utilizar algoritmos de corrección de errores para asegurar nuestra información, un algoritmo más sencillo como CRC-32 es lo mínimo que se debería utilizar para garantizar la integridad de toda la información que transmitimos.

En contraste a un acercamiento más cauteloso como lo es CRC-32, Hamming se acerca más a la cultura tan veloz en la que nos encontramos hoy en día. Para comunicaciones de día a día y manejo de información no crítica, el interés se inclina más en lo rápido que la obtenemos y no tanto en la calidad de esta. Uno de los puntos que más respalda este acercamiento es la mayor capacidad de medios que tenemos a nuestra disposición, lo que nos

permite alcanzar más fidelidad en cuanto a la información que se envía y recibe. Con esto, algoritmos como Hamming que priorizan la rápida detección y corrección de errores se vuelven el acercamiento más lógico, siempre y cuando los datos que se estén manejando no sean críticos.

Conclusiones

- A pesar de sus limitaciones, el algoritmo CRC-32 es un algoritmo no muy complejo que es altamente eficiente en detectar errores de uno o varios bits en una cadena.
- Al momento de elegir un algoritmo de detección o corrección de errores, es importante tomar en cuenta sus limitaciones para elegir el que mejor se adapte a las necesidades de la implementación.
- Uno de los criterios más importantes para tener en cuenta al momento de elegir el acercamiento a implementar debe radicar en la delicadeza y criticalidad de los datos que se estarán enviando y recibiendo; si la data no es sensible, tiene más sentido priorizar un acercamiento ágil con un algoritmo como Hamming.

Referencias

- IBM. (2021) ***Bits de paridad***. Recuperado de https://www.ibm.com/docs/es/aix/7.2?topic=parameters-parity-bits
- Embedded Office. (2023) ***When using CRC32 Checksum: Exploring Its Importance and Limitations in Embedded Systems***. Recuperado de: https://www.linkedin.com/pulse/when-using-crc32-checksum-exploring-its-importance-limitations
- *Error detecting and error correcting codes*. (1950, 1 abril). Nokia Bell Labs Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/6772729
- Tanenbaum, A. S., Wetherall, D. J., & Elizondo, A. V. R. (2012). *Redes de computadoras*.