Universidad del Valle de Guatemala

Redes

Sección ## 20

Carne 21527 Arturo Herberto Argueta Ávila

Carne 20172 Diego Andrés Alonzo Medinilla

Dia | Fecha

Esquemas de detección y corrección. Pt1

Laboratorio # 2
Redes # 20

Escenarios de pruebas

No se cambió la longitud debido a que se implementaron escenarios para Hamming (7,4) de manera que la longitud del mensaje debe ser igual. Aunque sí se soportaría para longitudes más grandes en CRC32.

Escenario 1 (No hay errores)

Caso 1

```
Enter a bit string: 1100
Hamming encoded: 1100001
CRC32 Checksum: 11000111001001001011010111000011001
```

```
Enter a bit message for the Hamming decoding: 1100001
There is no error in message
Enter a bit message for the crc32 detection: 11000111001001001011010111000011001
Correct checksum
```

Caso 2

```
Enter a bit string: 0111
Hamming encoded: 0110011
CRC32 Checksum: 01110011111100101101010110010111 0010
```

```
Enter a bit message for the Hamming decoding: 0110011
There is no error in message
Enter a bit message for the crc32 detection: 01110011111100101101010110010111 0010
Correct checksum
```

Caso 3

```
Enter a bit string: 0010
Hamming encoded: 0011001
CRC32 Checksum: 00101011101001110111111111010000 11100
```

```
Enter a bit message for the Hamming decoding: 0110011
There is no error in message
Enter a bit message for the crc32 detection: 01110011111100101101010110010111 0010
Correct checksum
```

Escenario 2 (Hay errores, pero solo en 1 bit)

No se indicará los mensajes que sufrieron una manipulación porque precisamente ahí se muestran los 2 screenshots para que se pueda ver con detenimiento qué bits sufrieron un error fraudulento. Así mismo, se simuló que sucedería si llegara el mensaje correcto para que se observara la comparación.

Caso 1

```
Enter a bit string: 1010
Hamming encoded: 1010010
CRC32 Checksum: 10100001110010010001110001110000001
```

```
Enter a bit message for the Hamming decoding: 1010011
There is an error in the position 6
Corrected message is: 1010010
Enter a bit message for the crc32 detection: 10100001110010010001110001110000011
Incorrect checksum
```

Caso 2

```
Enter a bit string: 1111
Hamming encoded: 1111000
CRC32 Checksum: 11111001100111001011011001000 1101111
```

```
Enter a bit message for the Hamming decoding: 1111000
There is no error in message
Enter a bit message for the crc32 detection: 11111001100111001011011001000 1101111
Correct checksum
root@80be159ca1da:~/Redes-2024-lab2# /bin/python3 /root/Redes-2024-lab2/receptor.py
Enter a bit message for the Hamming decoding: 1110000
There is an error in the position 3
Corrected message is: 1111000
Enter a bit message for the crc32 detection: 11111001100111001011011001000 1101110
Incorrect checksum
```

Caso 3

```
Enter a bit string: 0101
Hamming encoded: 0101010
CRC32 Checksum: 01011011110110100001100000 0111101010
```

```
Enter a bit message for the Hamming decoding: 0101010
There is no error in message
Enter a bit message for the crc32 detection: 01011011110110100001100000 0111101010
Correct checksum
root@80be159ca1da:~/Redes-2024-lab2# /bin/python3 /root/Redes-2024-lab2/receptor.py
Enter a bit message for the Hamming decoding: 0101110
There is an error in the position 4
Corrected message is: 0101010
Enter a bit message for the crc32 detection: 01011011110110100001100000 0111100010
Incorrect checksum
```

Escenario 3 (Hay errores en más de un bit)

No se indicará los mensajes que sufrieron una manipulación porque precisamente ahí se muestran los 2 screenshots para que se pueda ver con detenimiento qué bits sufrieron un error fraudulento. Así mismo, se simuló que sucedería si llegara el mensaje correcto para que se observara la comparación.

Caso 1

```
Enter a bit string: 0010
Hamming encoded: 0011001
CRC32 Checksum: 00101011101001110111111101000011100
```

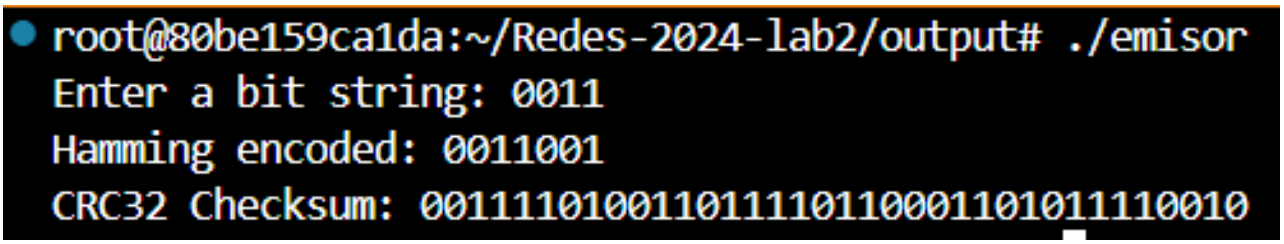
```
Enter a bit message for the Hamming decoding: 0011001
There is no error in message
Enter a bit message for the crc32 detection: 00101011101001110111111101000011100
Correct checksum
root@80be159ca1da:~/Redes-2024-lab2# python3 receptor.py
Enter a bit message for the Hamming decoding: 0011010
There is an error in the position 4
Corrected message is: 0011110
Enter a bit message for the crc32 detection: 00101011101001110111111101000011111
Incorrect checksum
```

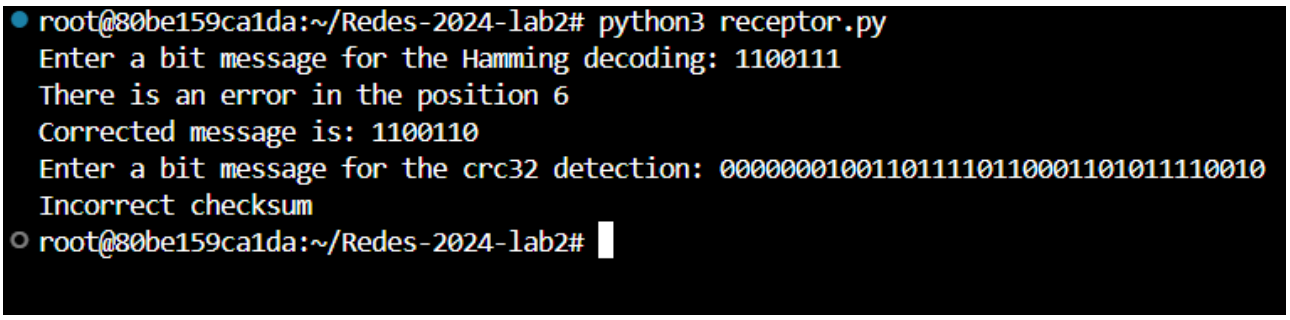
Caso 2

```
root@80be159ca1da:~/Redes-2024-lab2/output# ./emisor
Enter a bit string: 1001
Hamming encoded: 1001011
CRC32 Checksum: 10011111011100010001111111101110111
root@80be159ca1da:~/Redes-2024-lab2/output#
```

```
root@80be159ca1da:~/Redes-2024-lab2# python3 receptor.py
Enter a bit message for the Hamming decoding: 1001011
There is no error in message
Enter a bit message for the crc32 detection: 10011111011100010001111111101110111
Correct checksum
root@80be159ca1da:~/Redes-2024-lab2# python3 receptor.py
Enter a bit message for the Hamming decoding: 1000000
There is an error in the position 0
Corrected message is: 0000000
Enter a bit message for the crc32 detection: 10011111011100010001111111101111000
Incorrect checksum
root@80be159ca1da:~/Redes-2024-lab2#
```

Caso 3





Manipulación de bits ¿indetectable?

Pues, realizamos varias pruebas sin embargo, se nos fue imposible encontrar alguna combinación en la cuál se pudieran manipular bits. Sin embargo, en la teoría es posible ya que podría suceder por ejemplo en el CRC32 que hubiera una colisión de Checksums, de manera que el checksum de un valor justo coincidiera con el valor de otro y justo se modificaran los bits necesarios para eso. Sin embargo, en nuestra implementación no logramos comprobar esto dado que es demasiado improbable y además se necesitaría de mucha simulación para determinar qué bits se necesitarían cambiar para ese polinomio.
Así mismo, para el código de Hamming, sucedió algo similar, con nuestra implementación no logramos comprobar que no pudiera detectar el código erróneo debido a la estructura del algoritmo y lo complicado que es averiguarlo, sin embargo, en teoría sí es posible hacerlo. Es decir, como se vio en los errores de más de un bit, detecta que existe un error lo intenta corregir pero fracasa estrepitosamente en corregir cuándo es de más de un bit, por lo que realmente no es que los errores sean indetectables en este caso, es un error del algoritmo asumir que solo se ha cambiado un bit.

Ventajas y desventajas de cada algoritmo

En el caso de los códigos de Hamming su principal ventaja es su simplicidad y el hecho de que es realmente veloz al computarse. De manera que es un algoritmo casi lineal siempre y cuando se verifique de una forma eficiente y no rústica, esto se puede verificar en el código se computa en 3 lineas en cpp. Así mismo, otra ventaja que posee es el hecho de que empaqueta en piezas muy pequeñas de información, de manera que esto en términos de envío y recepción es ideal ya que no es realmente pesado, esto se pudo notar dado que en comparación con el CRC32 las longitudes generadas eran extremadamente cortas dado que no requiere tanta redundancia, esto producto de basarse en bits de paridad de potencias de 2.
La principal desventaja que posee el algoritmo es precisamente su simplicidad y el hecho de que a pesar de que pueda detectar la posición de los errores, puede que tenga más de un error

entonces al momento de reinterpretar los datos este de malas referencias asumiendo que solamente un bit ha cambiado. En referencia con el otro algoritmo, Viterbi, el Hamming se destaca en esos aspectos mencionados: simplicidad, velocidad, pocos recursos, aunque falla en que es menos robusto. Es precisamente por esto que cada uno es aplicado en campos distintos de manera que el Viterbi se busca en infraestructuras que requieren una mayor seguridad de los datos, mientras que el Hamming es más usado en almacenamiento donde los errores no sean muchos.

Para el CRC32 su principal ventaja es que es robusto dado que utiliza 32 bits de redundancia tal como se pudo observar no se equivocó ni una sola vez en dar un veredicto, así mismo, puede detectar errores complejos como cuando se realizaron cambios de varios bits y determinó que había error. A pesar de que existen colisiones, la probabilidad que ocurran es baja dado que son polinomios de 32 coeficientes. Mientras que para el algoritmo de fletcher es más simple, pero es menos robusto por lo que es mejor el CRC32 sin ninguna duda, además de que está más estandarizado.

En lo personal, creemos que de los 2 algoritmos que programamos el CRC32 es el mejor dado que no realiza veredictos erróneos así como que casi nunca se equivoca solo en casos muy extremos y probablemente se necesitarían de muchos recursos para inducir ese fallo.