

Detección y corrección de errores - pt 1

Link de github: <https://github.com/auyjos/Lab2Redes.git>

Languages utilizados:

Emisor: C++

Receptor: Python

Algoritmos utilizados:

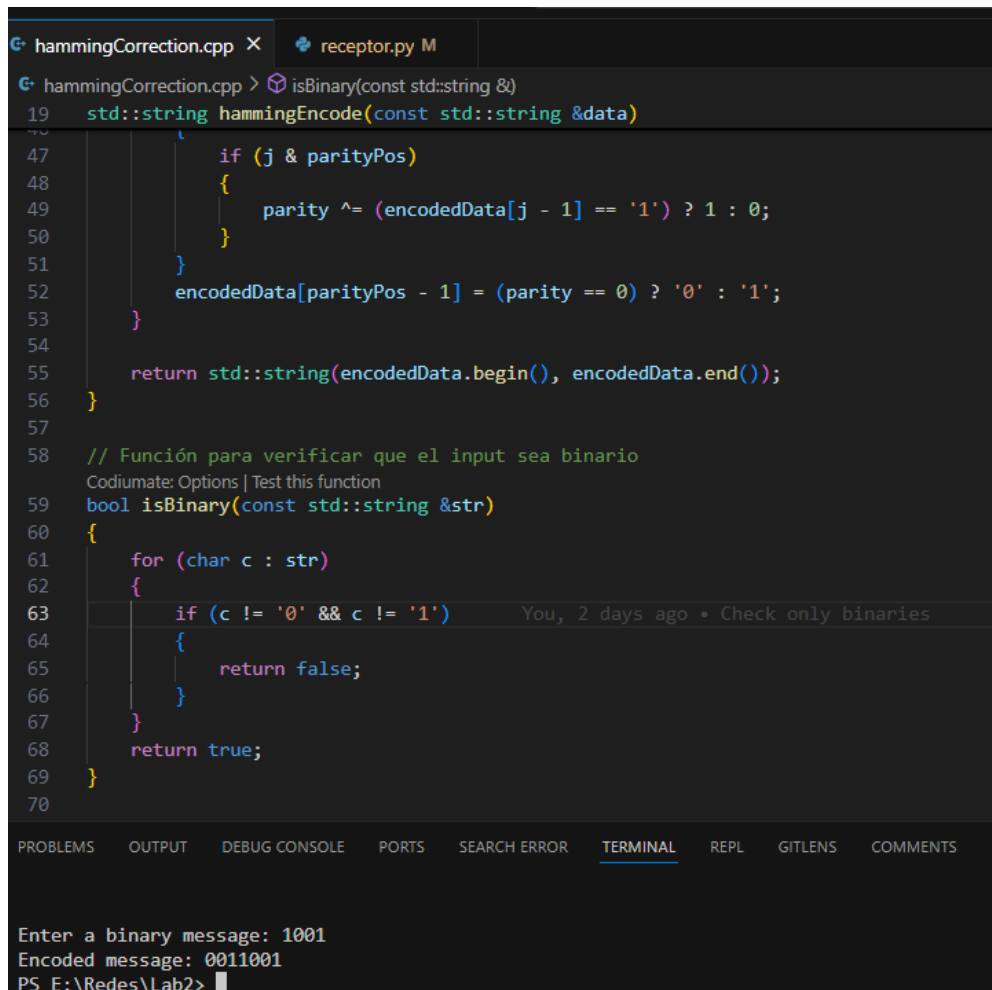
Corrección: Hamming

Detección de errores: checksum

- (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.

Sin errores:

Prueba 1:



```
19  std::string hammingEncode(const std::string &data)
20  {
21      int parityPos = 1;
22      while (parityPos < data.length())
23      {
24          int parity = 0;
25          for (int j = parityPos; j < data.length(); j += 2)
26          {
27              if (j & parityPos)
28              {
29                  parity ^= (encodedData[j - 1] == '1') ? 1 : 0;
30              }
31          }
32          encodedData[parityPos - 1] = (parity == 0) ? '0' : '1';
33          parityPos *= 2;
34      }
35      return std::string(encodedData.begin(), encodedData.end());
36  }
37
38  // Función para verificar que el input sea binario
39  Codiumate: Options | Test this function
40  bool isBinary(const std::string &str)
41  {
42      for (char c : str)
43      {
44          if (c != '0' && c != '1')
45          {
46              return false;
47          }
48      }
49      return true;
50  }
51
52  int main()
53  {
54      std::string message;
55      std::cout << "Enter a binary message: ";
56      getline(cin, message);
57      std::string encodedMessage = hammingEncode(message);
58      std::cout << "Encoded message: " << encodedMessage << endl;
59      return 0;
60  }
```

Enter a binary message: 1001
Encoded message: 0011001
PS E:\Redes\Lab2>

```

PS E:\Redes\Lab2> python receptor.py
Enter the received encoded message: 0011001
Calculated Fletcher checksum: 12850
Decoded message: 1001
Checksum verification passed. Message is intact.
PS E:\Redes\Lab2>

```

Prueba 2:

The screenshot shows a code editor with two tabs: `hammingCorrection.cpp` and `receptor.py`. The `hammingCorrection.cpp` tab is active, displaying C++ code for Hamming encoding and binary verification. The code includes a function `hammingEncode` that calculates parity and encodes data, and a function `isBinary` that checks if a string contains only '0' and '1' characters. Below the code editor, a terminal window shows the execution of the program. It prompts the user to enter a binary message, which is then encoded and decoded, with the Fletcher checksum calculated and verified.

```

hammingCorrection.cpp X receptor.py M
hammingCorrection.cpp > isBinary(const std::string &)
19  std::string hammingEncode(const std::string &data)
20  {
47      if (j & parityPos)
48      {
49          parity ^= (encodedData[j - 1] == '1') ? 1 : 0;
50      }
51  }
52  encodedData[parityPos - 1] = (parity == 0) ? '0' : '1';
53  }
54
55  return std::string(encodedData.begin(), encodedData.end());
56  }
57
58  // Función para verificar que el input sea binario
59  Codiumate: Options | Test this function
60  bool isBinary(const std::string &str)
61  {
62      for (char c : str)
63      {
64          if (c != '0' && c != '1')
65          {
66              return false;
67          }
68      }
69      return true;
70  }
71
PROBLEMS OUTPUT DEBUG CONSOLE PORTS SEARCH ERROR TERMINAL REPL GITLENS COMMENTS
Enter a binary message: 1001001
Encoded message: 11110011001
PS E:\Redes\Lab2>

```

```

PS E:\Redes\Lab2> python receptor.py
Enter the received encoded message: 11110011001
Calculated Fletcher checksum: 5140
Decoded message: 1001001
Checksum verification passed. Message is intact.

```

Prueba 3:

```

hammingCorrection.cpp X  receptor.py M
hammingCorrection.cpp > isBinary(const std::string &)
19  std::string hammingEncode(const std::string &data)
20  {
21      // ...
22      if (j & parityPos)
23      {
24          parity ^= (encodedData[j - 1] == '1') ? 1 : 0;
25      }
26      encodedData[parityPos - 1] = (parity == 0) ? '0' : '1';
27  }
28  return std::string(encodedData.begin(), encodedData.end());
29  }
30
31  // Función para verificar que el input sea binario
32  Codiumate: Options | Test this function
33  bool isBinary(const std::string &str)
34  {
35      for (char c : str)
36      {
37          if (c != '0' && c != '1')
38          {
39              return false;
40          }
41      }
42      return true;
43  }
44
45  PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  SEARCH ERROR  TERMINAL  REPL  GITLENS  COMMENTS
46
47  Enter a binary message: 110011001100
48  Encoded message: 01111000110011000
49  PS E:\Redes\Lab2>

```

```

PS E:\Redes\Lab2> python receptor.py
Enter the received encoded message: 01111000110011000
Calculated Fletcher checksum: 35397
Decoded message: 110011001100
Checksum verification passed. Message is intact.
PS E:\Redes\Lab2>

```

- (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.

Original:

```
Enter a binary message: 1010
Encoded message: 1011010
PS E:\Redes\Lab2> █
```

Error 1:

```
PS E:\Redes\Lab2> python receptor.py
Enter the received encoded message: 1011011
Calculated Fletcher checksum: 46774
Error detected at position: 7
Corrected encoded message: 1011010
Decoded message: 1010
Checksum verification passed. Message is intact.
PS E:\Redes\Lab2> █
```

Original:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  SEARCH ERROR  TE
Enter a binary message: 100100
Encoded message: 0011001000
PS E:\Redes\Lab2> █
```

Error 2:

```
PS E:\Redes\Lab2> python receptor.py
Enter the received encoded message: 0011001001
Calculated Fletcher checksum: 29298
Error detected at position: 2
Corrected encoded message: 0111001001
Decoded message: 100101
Checksum verification passed. Message is intact.
PS E:\Redes\Lab2> █
```

Original:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  SEA
Enter a binary message: 100010001
Encoded message: 1111000010001
PS E:\Redes\Lab2> █
```

Error 3:

```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS SEARCH ERROR TERMINAL REPL GITLENS COMMENTS
PS E:\Redes\Lab2> python receptor.py
Enter the received encoded message: 1111000010000
Calculated Fletcher checksum: 29041
Error detected at position: 5
Corrected encoded message: 1111100010000
Decoded message: 110010000
Checksum verification passed. Message is intact.
PS E:\Redes\Lab2> █
```

- (dos+ errores): Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar dos o más bits cualesquiera antes de proporcionar 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.

Original:

```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS SEARCH ERROR TERMINAL REPL GITLENS COMMENTS
Enter a binary message: 1001
Encoded message: 0011001
PS E:\Redes\Lab2> █
```

Con dos bits cambiados:

```
PS E:\Redes\Lab2> python receptor.py
Enter the received encoded message: 0001000
Calculated Fletcher checksum: 4112
Error detected at position: 4
Corrected encoded message: 0000000
Decoded message: 0000
Checksum verification passed. Message is intact.
PS E:\Redes\Lab2> █
```

Original:

```
Enter a binary message: 1001000
Encoded message: 00110010000
PS E:\Redes\Lab2>
```

Dos bits cambiados:

```
PS E:\Redes\Lab2> python receptor.py
Enter the received encoded message: 00010010010
Calculated Fletcher checksum: 21074
Error detected at position: 1
Corrected encoded message: 10010010010
Decoded message: 0001010
Checksum verification passed. Message is intact.
PS E:\Redes\Lab2>
```

Original:

```
Enter a binary message: 1000110001
Encoded message: 01110001110001
PS E:\Redes\Lab2>
```

Dos bits cambiados:

```
PS E:\Redes\Lab2> python receptor.py
Enter the received encoded message: 01100001110000
Calculated Fletcher checksum: 8738
Error detected at position: 2
Corrected encoded message: 00100001110000
Decoded message: 1000110000
Checksum verification passed. Message is intact.
PS E:\Redes\Lab2>
```

- ¿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, demuéstrelo con su implementación.

Sí, es posible manipular los bits de tal forma que el algoritmo de Hamming y el checksum de Fletcher no sean capaces de detectar el error. Esto se debe a las limitaciones inherentes de cada algoritmo. El código de Hamming puede corregir errores de un solo bit y detectar errores de dos bits. Sin embargo, no puede corregir errores de dos bits y no puede detectar errores de más de dos bits. El checksum de Fletcher es eficaz para detectar errores de bits individuales y pequeños errores de múltiples bits, pero no es infalible contra ciertos patrones de errores. Esto se puede observar al cambiar dos bits en los mensajes, ya que ambos algoritmos pueden localizar el error en un cierto punto pero no en todos.

- 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.

Código de Hamming

Ventajas:

Las ventajas del código de Hamming no solo detecta errores de un solo bit, sino que también puede corregirlos, lo que lo hace útil en aplicaciones donde la corrección automática es crítica, también puede detectar errores de dos bits, lo que proporciona una capa adicional de seguridad en comparación con otros algoritmos que solo detectan errores de un bit. Aunque requiere la adición de bits de paridad, el overhead es relativamente bajo comparado con otros métodos de corrección de errores.

Desventajas:

La implementación del código de Hamming es más compleja que la de otros métodos simples como el bit de paridad aunque puede detectar los errores de dos bits, puede corregirlos y no puede detectar errores de más de dos bits, lo que limita su efectividad en canales con altas tasas de error. También requiere la adición de múltiples bits de paridad, lo que aumenta la redundancia en comparación con métodos más simples.

Fletcher Checksum

Ventajas:

La implementación del checksum de Fletcher es relativamente simple y directa, lo que facilita su uso en sistemas con recursos limitados. La computación del checksum es rápida, ya que solo implica sumas y operaciones modulares, sin operaciones complejas como las necesarias en el código de Hamming. Es más efectivo que los checksums simples en la detección de múltiples errores de bits cercanos entre sí.

Desventajas:

Las desventajas de checksum es que no es infalible y puede fallar en detectar ciertos patrones de errores múltiples. A diferencia del código de Hamming, el checksum de Fletcher solo detecta errores pero no proporciona un mecanismo para corregirlos. Aunque menos que el código de Hamming, aún introduce overhead adicional en la transmisión.

Comparación Basada en Pruebas

Para evaluar la efectividad y limitaciones de cada algoritmo, se realizaron las siguientes pruebas:

Prueba de Errores de Un Bit:

Hamming: Detectó y corrigió el error de un bit, mostrando la posición y corrigiendo el mensaje.

Fletcher: Detectó el error al verificar el checksum, indicando que había una discrepancia.

Prueba de Errores de Dos Bits:

Hamming: Detectó la presencia de errores, pero no pudo corregirlos. Indicó que el mensaje tenía errores sin capacidad de corrección.

Fletcher: Detectó la discrepancia en el checksum, indicando la presencia de errores.