

**UNIVERSIDAD DEL VALLE DE GUATEMALA**  
Redes  
Sección 21  
Jorge Yass



## **Laboratorio 2**

Esquemas de detección y corrección

Alejandro Ortega 18248  
José Daniel Gomez Cabrera 21429

Github <https://github.com/JDgomez2002/networks-labs.git>

## Hamming

1. 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.

1001

```
def main():
    # print("\nType the transmitter message (4 bits)")
    message = "1001"

Message: 1001
Parity bit p1: 1
Parity bit p2: 0
Parity bit p3: 0
Parity bit p4: 1
Codified bits: 1001100
length: 7

jdgomez@MacbookPro hamming %
```

```
transmitter.py receiver.js
lab2 > hamming > receiver.js > main > message
45
46     function main() {
47         const message = "1001100";
48         // convert the message to an array of integers
49
50 [jdgomez@MacbookPro hamming % node receiver.js
51
52     Received message
53     1001100
54
55     Number of groups: 1
56     Group 1: 1001100 length: 7
57     Calculated parity bits: p1: 1, p2: 0, p3: 0
58     Received parity bits: p1: 1, p2: 0, p3: 0
59     No errors in the data.
60
61     Decoded message
62     1001
63
64 jdgomez@MacbookPro hamming %
```

110110

```
transmitter.py × receiver.js ×
lab2 > hamming > transmitter.py > main
27 def main():
28     # print("\nType the transmitter message (4 bits)")
29     message = "110110"
30
31 jdgom...@MacbookPro hamming % /usr/local/bin/python3 /User...
32                                     Hamming algorithm -
33
34 Message: 11010010
35 Parity bit p1: 0
36 Parity bit p2: 1
37 Parity bit p3: 0
38 Parity bit p4: 1
39 Codified bits: 11001100011001
40 Length: 14
41
42 jdgom...@MacbookPro hamming %
43
44
```

```
transmitter.py × receiver.js ×
lab2 > hamming > receiver.js > main > [e] message
45
46 function main() {
47     const message = "11001100011001";
48     // convert the message to an array of integers
49
50 jdgom...@MacbookPro hamming % node receiver.js
51
52                                     Received message -
53 11001100011001
54
55 Number of groups: 2
56 Group 1: 1100110 length: 7
57 Group 2: 0011001 length: 7
58 Calculated parity bits: p1: 0, p2: 1, p3: 0
59 Received parity bits: p1: 0, p2: 1, p3: 0
60 No errors in the data.
61 Calculated parity bits: p1: 1, p2: 0, p3: 1
62 Received parity bits: p1: 1, p2: 0, p3: 1
63 No errors in the data.
64
65                                     Decoded message -
66 11010010
67 jdgom...@MacbookPro hamming %
68
69
```

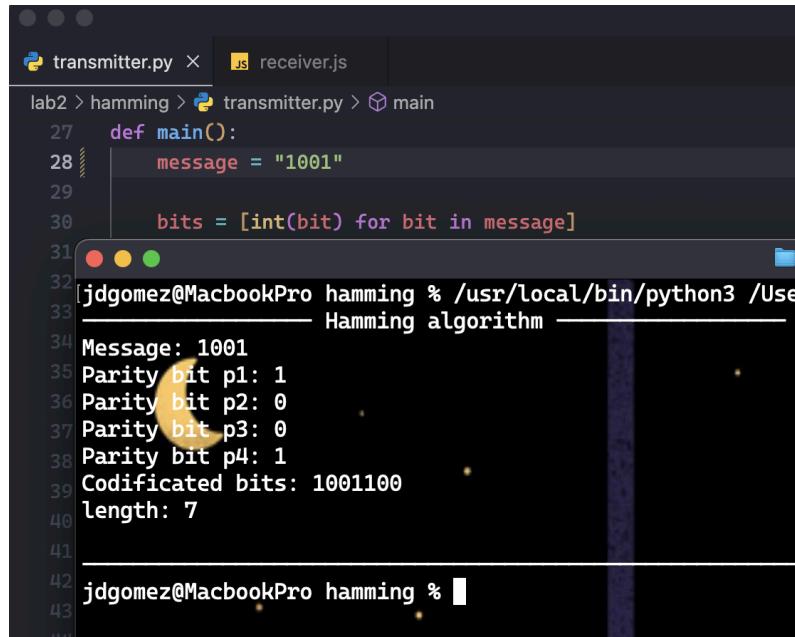
1001100111

```
transmitter.py receiver.js
lab2 > hamming > transmitter.py > main
26
27     def main():
28         message = "1001100111"
29
30     jdgom...@MacbookPro hamming % /usr/local/bin/python3 /User
31     _____ Hamming algorithm _____
32     Message: 100110010011
33     Parity bit p1: 1
34     Parity bit p2: 0
35     Parity bit p3: 0
36     Parity bit p4: 1
37     Codified bits: 100110010011000011110
38     length: 21
39
40
41 jdgom...@MacbookPro hamming %
```

```
transmitter.py receiver.js
lab2 > hamming > receiver.js > main
46     function main() {
47         const message = "100110010011000011110";
48         // convert the message to an array of integers
49
50     jdgom...@MacbookPro hamming % node receiver.js
51
52     _____ Received message _____
53     100110010011000011110
54
55     Calculated parity bits: p1: 1, p2: 0, p3: 0
56     Received parity bits: p1: 1, p2: 0, p3: 0
57     No errors in the data.
58     Calculated parity bits: p1: 1, p2: 0, p3: 0
59     Received parity bits: p1: 1, p2: 0, p3: 0
60     No errors in the data.
61     Calculated parity bits: p1: 1, p2: 1, p3: 0
62     Received parity bits: p1: 1, p2: 1, p3: 0
63     No errors in the data.
64
65     _____ Decoded message _____
66     100110010011
67
68 jdgom...@MacbookPro hamming %
```

- 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.

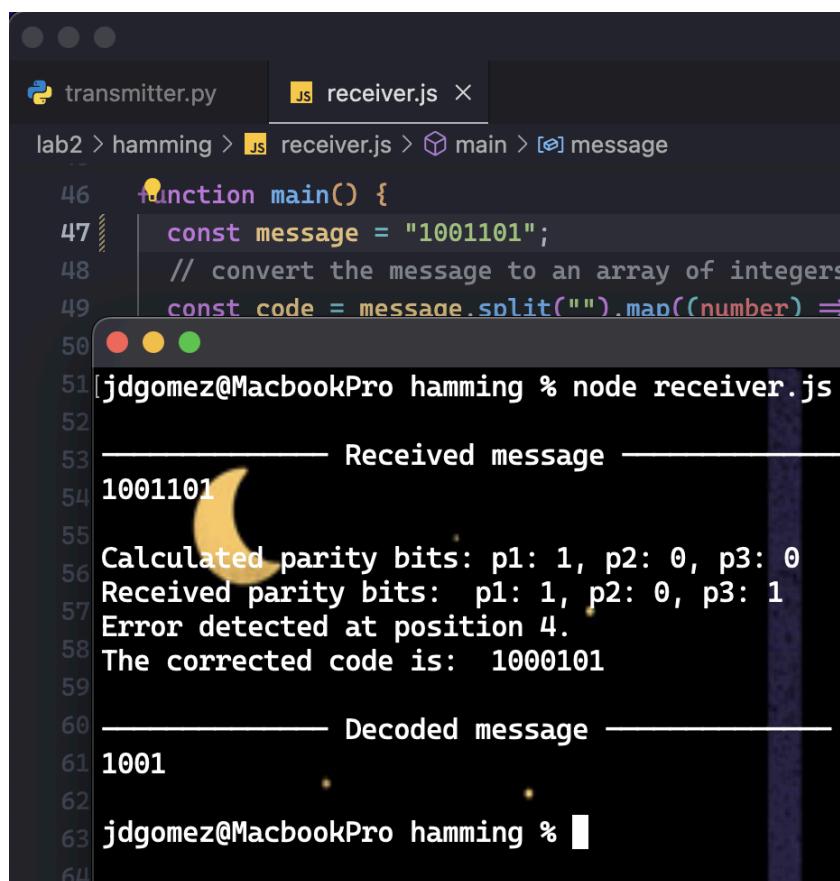
1001



```

transmitter.py × receiver.js
lab2 > hamming > transmitter.py > main
27 def main():
28     message = "1001"
29
30     bits = [int(bit) for bit in message]
31
32 [jdgomez@MacbookPro hamming % /usr/local/bin/python3 /User
33                                     Hamming algorithm —————
34 Message: 1001
35 Parity bit p1: 1
36 Parity bit p2: 0
37 Parity bit p3: 0
38 Parity bit p4: 1
39 Codified bits: 1001100
40 length: 7
41
42 [jdgomez@MacbookPro hamming % ]
43

```

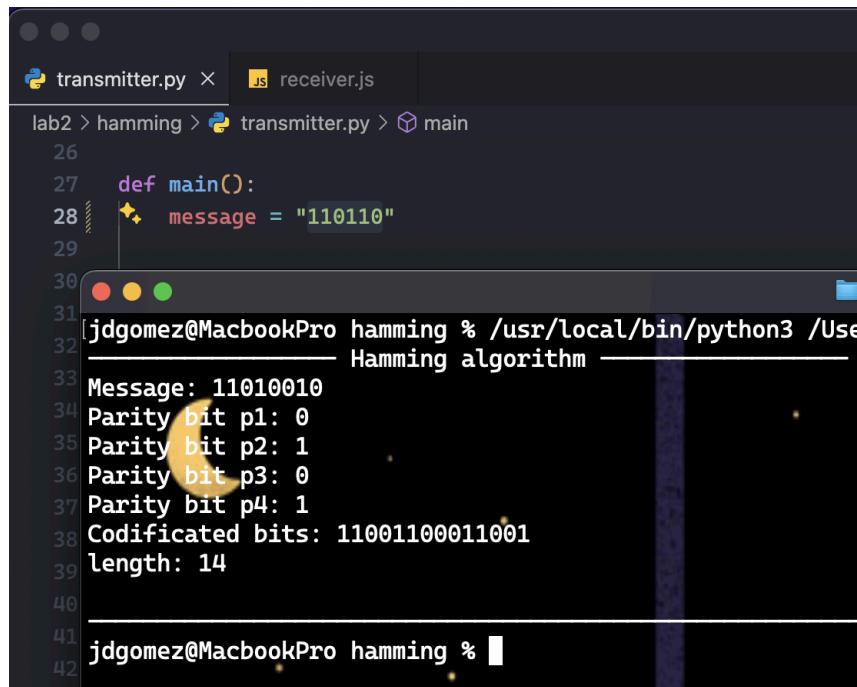


```

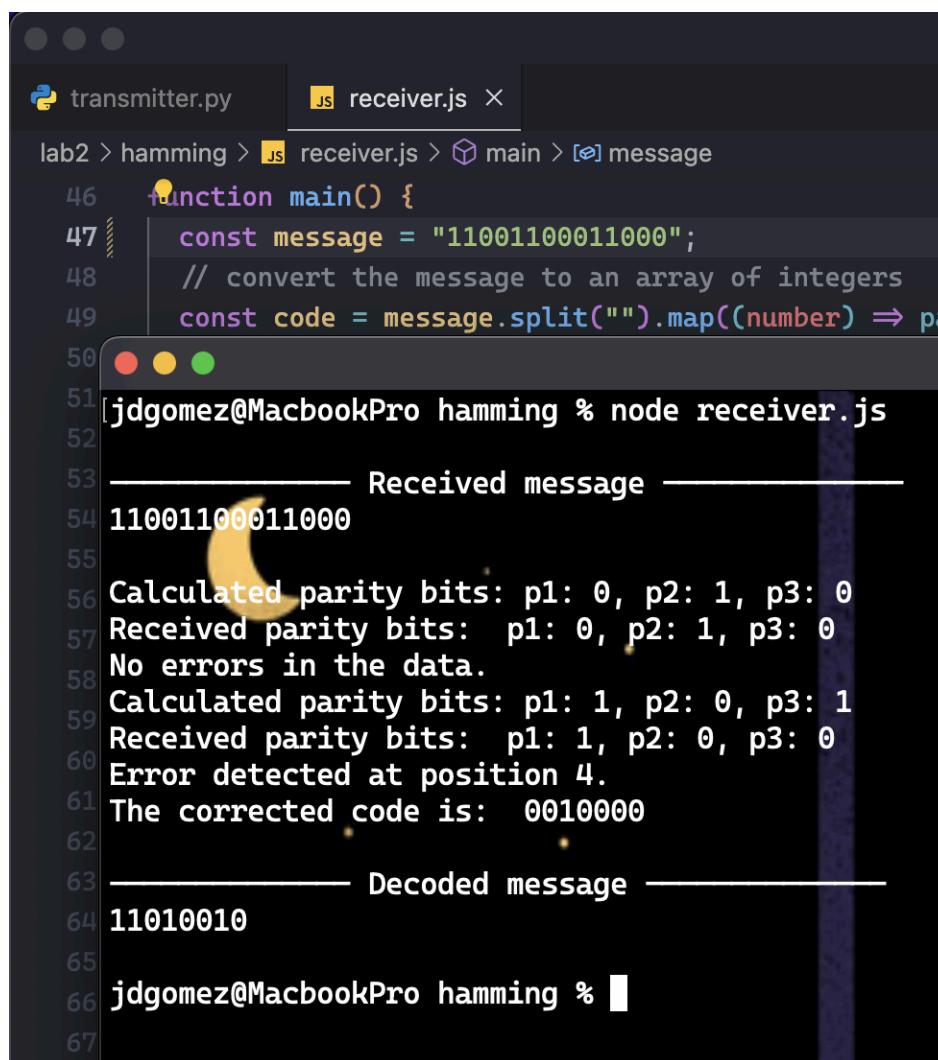
transmitter.py × receiver.js ×
lab2 > hamming > receiver.js > main > [?] message
46 function main() {
47     const message = "1001101";
48     // convert the message to an array of integers
49     const code = message.split("").map((number) =>
50
51 [jdgomez@MacbookPro hamming % node receiver.js
52
53                                     Received message —————
54 1001101
55
56 Calculated parity bits: p1: 1, p2: 0, p3: 0
57 Received parity bits: p1: 1, p2: 0, p3: 1
58 Error detected at position 4.
59 The corrected code is: 1000101
60
61                                     Decoded message —————
62 1001
63
64 [jdgomez@MacbookPro hamming % ]

```

110110



```
transmitter.py > receiver.js
lab2 > hamming > transmitter.py > main
26
27     def main():
28         message = "110110"
29
30
31 jdgomez@MacbookPro hamming % /usr/local/bin/python3 /Use
32                                     Hamming algorithm
33 Message: 11010010
34 Parity bit p1: 0
35 Parity bit p2: 1
36 Parity bit p3: 0
37 Parity bit p4: 1
38 Codified bits: 11001100011001
39 length: 14
40
41 jdgomez@MacbookPro hamming %
42
```



```
transmitter.py > receiver.js > main > message
46     function main() {
47         const message = "11001100011000";
48         // convert the message to an array of integers
49         const code = message.split("").map((number) => pa
50
51 jdgomez@MacbookPro hamming % node receiver.js
52
53                                     Received message
54 11001100011000
55
56 Calculated parity bits: p1: 0, p2: 1, p3: 0
57 Received parity bits: p1: 0, p2: 1, p3: 0
58 No errors in the data.
59 Calculated parity bits: p1: 1, p2: 0, p3: 1
60 Received parity bits: p1: 1, p2: 0, p3: 0
61 Error detected at position 4.
62 The corrected code is: 0010000
63
64                                     Decoded message
65 11010010
66 jdgomez@MacbookPro hamming %
67
```

1001100111

```
transmitter.py x receiver.js
lab2 > hamming > transmitter.py > main
27 def main():
28     message = "1001100111"
29
30     bits = [int(bit) for bit in message]
31
32 jdgomez@MacbookPro hamming % /usr/local/bin/python3 /Use
33                                     Hamming algorithm
34 Message: 100110010011
35 Parity bit p1: 1
36 Parity bit p2: 0
37 Parity bit p3: 0
38 Parity bit p4: 1
39 Codified bits: 100110010011000011110
40 length: 21
41
42 jdgomez@MacbookPro hamming %
43
```

```
transmitter.py x receiver.js x
lab2 > hamming > receiver.js > main > [o] message
46 function main() {
47     const message = "100110010011000011010";
48     // convert the message to an array of integers
49     const code = message.split("").map((number) =>
50
51 jdgomez@MacbookPro hamming % node receiver.js
52
53                                     Received message
54 100110010011000011010
55
56 Calculated parity bits: p1: 1, p2: 0, p3: 0
57 Received parity bits: p1: 1, p2: 0, p3: 0
58 No errors in the data.
59 Calculated parity bits: p1: 1, p2: 0, p3: 0
60 Received parity bits: p1: 1, p2: 0, p3: 0
61 No errors in the data.
62 Calculated parity bits: p1: 1, p2: 0, p3: 1
63 Received parity bits: p1: 1, p2: 1, p3: 0
64 Error detected at position 6.
65 The corrected code is: 0011000
66
67                                     Decoded message
68 100110010010
69 jdgomez@MacbookPro hamming %
```

3. 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 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.

1001

```
transmitter.py > receiver.js
lab2 > hamming > transmitter.py > main
27 def main():
28     message = "1001"
29
30     bits = [int(bit) for bit in message]
31
32 [● ● ●] jdgomez@MacbookPro hamming % /usr/local/bin/python3 /User:
33                                     Hamming algorithm ——————
34 Message: 1001
35 Parity bit p1: 1
36 Parity bit p2: 0
37 Parity bit p3: 0
38 Parity bit p4: 1
39 Codified bits: 1001100
40 length: 7
41
42 jdgomez@MacbookPro hamming %
```

```
transmitter.py receiver.js

lab2 > hamming > receiver.js > main > message

46 function main() {
47     const message = "1001101";
48     // convert the message to an array of integers
49     const code = message.split("").map((number) =>
50
51 jdgomez@MacbookPro hamming % node receiver.js
52
53 ━━━━━━ Received message ━━━━━━
54 1001101
55
56 Calculated parity bits: p1: 1, p2: 0, p3: 0
57 Received parity bits: p1: 1, p2: 0, p3: 1
58 Error detected at position 4.
59 The corrected code is: 1000101
60
61 ━━━━━━ Decoded message ━━━━━━
62 1001
63
64 jdgomez@MacbookPro hamming %
```

110110

```
transmitter.py receiver.js

lab2 > hamming > transmitter.py > main

26
27 def main():
28     message = "110110"
29
30 jdgomez@MacbookPro hamming % /usr/local/bin/python3 /Use
31                                     Hamming algorithm
32
33 Message: 11010010
34 Parity bit p1: 0
35 Parity bit p2: 1
36 Parity bit p3: 0
37 Parity bit p4: 1
38 Codified bits: 110011100011001
39 length: 14
40
41 jdgomez@MacbookPro hamming %
```

```
transmitter.py receiver.js
lab2 > hamming > receiver.js > main > message
46  function main() {
47    const message = "11001100011000";
48    // convert the message to an array of integers
49    const code = message.split("").map((number) => pa
50
51 [jdgomez@MacbookPro hamming % node receiver.js
52
53 _____ Received message _____
54 11001100011000
55
56 Calculated parity bits: p1: 0, p2: 1, p3: 0
57 Received parity bits: p1: 0, p2: 1, p3: 0
58 No errors in the data.
59 Calculated parity bits: p1: 1, p2: 0, p3: 1
60 Received parity bits: p1: 1, p2: 0, p3: 0
61 Error detected at position 4.
62 The corrected code is: 0010000
63 _____ Decoded message _____
64 11010010
65
66 jdgomez@MacbookPro hamming %
67
```

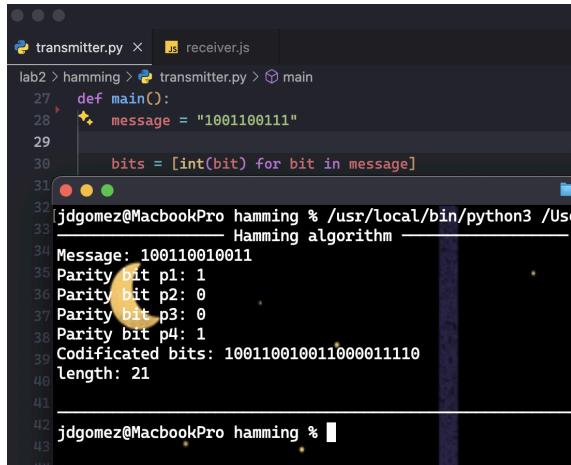
1001100111

```
transmitter.py receiver.js
lab2 > hamming > transmitter.py > main
27 def main():
28   message = "1001100111"
29
30   bits = [int(bit) for bit in message]
31
32 [jdgomez@MacbookPro hamming % /usr/local/bin/python3 /Use
33                                     Hamming algorithm -
34 Message: 100110010011
35 Parity bit p1: 1
36 Parity bit p2: 0
37 Parity bit p3: 0
38 Parity bit p4: 1
39 Codified bits: 100110010011000011110
40 length: 21
41
42 jdgomez@MacbookPro hamming %
43
```

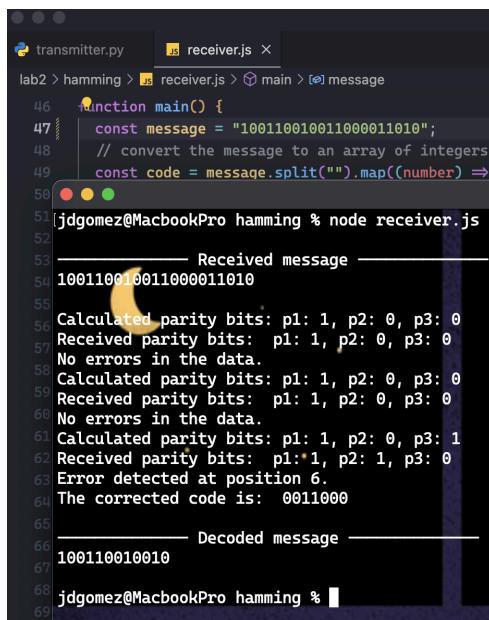
```
transmitter.py receiver.js ×
lab2 > hamming > receiver.js > main > [o] message
46 function main() {
47     const message = "100110010011000011010";
48     // convert the message to an array of integers
49     const code = message.split("").map((number) =>
50
51 jdomez@MacbookPro hamming % node receiver.js
52
53 ━━━━━━ Received message ━━━━━━
54 100110010011000011010
55
56 Calculated parity bits: p1: 1, p2: 0, p3: 0
57 Received parity bits: p1: 1, p2: 0, p3: 0
58 No errors in the data.
59 Calculated parity bits: p1: 1, p2: 0, p3: 0
60 Received parity bits: p1: 1, p2: 0, p3: 0
61 No errors in the data.
62 Calculated parity bits: p1: 1, p2: 0, p3: 1
63 Received parity bits: p1: 1, p2: 1, p3: 0
64 Error detected at position 6.
65 The corrected code is: 0011000
66
67 ━━━━━━ Decoded message ━━━━━━
68 100110010010
69 jdomez@MacbookPro hamming %
```

4. ¿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.

Si, es posible por medio de manipular tantos bits que ya no pueda reconocer el error.



```
lab2 > hamming > transmitter.py > main
27 def main():
28     message = "100110010011"
29
30     bits = [int(bit) for bit in message]
31
32 jdgomez@MacbookPro hamming % /usr/local/bin/python3 /Use
33                                         Hamming algorithm
34 Message: 100110010011
35 Parity bit p1: 1
36 Parity bit p2: 0
37 Parity bit p3: 0
38 Parity bit p4: 1
39 Codified bits: 100110010011000011110
40 length: 21
41
42 jdgomez@MacbookPro hamming %
```



```
lab2 > hamming > receiver.js > main > message
46 function main() {
47     const message = "100110010011000011010";
48     // convert the message to an array of integers
49     const code = message.split("").map((number) =>
50
51 jdgomez@MacbookPro hamming % node receiver.js
52
53             Received message
54 100110010011000011010
55
56 Calculated parity bits: p1: 1, p2: 0, p3: 0
57 Received parity bits: p1: 1, p2: 0, p3: 0
58 No errors in the data.
59 Calculated parity bits: p1: 1, p2: 0, p3: 0
60 Received parity bits: p1: 1, p2: 0, p3: 0
61 No errors in the data.
62 Calculated parity bits: p1: 1, p2: 0, p3: 1
63 Received parity bits: p1: 1, p2: 1, p3: 0
64 Error detected at position 6.
65 The corrected code is: 0011000
66
67             Decoded message
68 100110010010
69 jdgomez@MacbookPro hamming %
```

5. En base a las pruebas que realizó, ¿qué ventajas y desventajas posee cada algoritmo con respecto al otro?

- Consideramos que el algoritmo de Hamming puede tener la ventaja al momento de reconocer errores sencillos o de pocos bits.
- Sin embargo, para disturbios con mayor número de bits, podría tener problemas en reconocer los errores de forma efectiva.
- Otros algoritmos de reconocimiento y corrección con mejor implementación podrían evitar este tipo de problemas.

- Se tendría que debuggear los casos específicos de cada error para poder encontrar si es posible manejarlo y/o solucionarlo de alguna otra manera.

## Fletcher16

1. 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.

### 1001

Emisor:

```
thon.exe "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/  
Ingrese un mensaje en binario: 1001  
Calculando checksum...  
Listo!  
Mensaje con checksum: 10010000100100001001  
○ PS C:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02>
```

Receptor:

```
[Running] node "c:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02\receiver.js"  
Mensaje con checksum: 10010000100100001001  
No se detectaron errores. Mensaje original: 1001  
  
[Done] exited with code=0 in 0.173 seconds
```

### 110110

Emisor

```
thon.exe "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/transmitter.py"  
Ingrese un mensaje en binario: 110110  
Calculando checksum...  
Listo!  
Mensaje con checksum: 1101100011011000110110  
○ PS C:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02>
```

Receptor:

```
[Running] node "c:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02\receiver.js"  
Mensaje con checksum: 1101100011011000110110  
No se detectaron errores. Mensaje original: 110110  
  
[Done] exited with code=0 in 0.128 seconds
```

### 1001100111

Emisor:

```
thon.exe "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/transmitter.py"  
Ingrese un mensaje en binario: 1001100111  
Calculando checksum...  
Listo!  
Mensaje con checksum: 10011001110110101101101001  
○ PS C:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02>
```

Receptor:

```
[Running] node "c:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02\receiver.js"  
Mensaje con checksum: 10011001110110101101101001  
No se detectaron errores. Mensaje original: 1001100111  
  
[Done] exited with code=0 in 0.155 seconds
```

2. 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.

## 1001

Emisor:

```
thon.exe "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/transmitter.py"
Ingrese un mensaje en binario: 1001
Calculando checksum...
Listo!
Mensaje con checksum: 10010000100100001001
PS C:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02>
```

Receptor:

```
[Running] node "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/receiver.js"
Mensaje con checksum: 10110000100100001001
Se detectaron errores. El mensaje se descarta por detectar errores.

[Done] exited with code=0 in 0.135 seconds
```

## 110110

Emisor:

```
thon.exe "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/transmitter.py"
Ingrese un mensaje en binario: 110110
Calculando checksum...
Listo!
Mensaje con checksum: 1101100011011000110110
PS C:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02>
```

Receptor:

```
[Running] node "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/receiver.js"
Mensaje con checksum: 1111100011011000110110
Se detectaron errores. El mensaje se descarta por detectar errores.

[Done] exited with code=0 in 0.136 seconds
```

## 1001100111

Emisor:

```
thon.exe "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/transmitter.py"
Ingrese un mensaje en binario: 1001100111
Calculando checksum...
Listo!
Mensaje con checksum: 10011001110110101101101001
PS C:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02>
```

Receptor:

```
[Running] node "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/receiver.js"
Mensaje con checksum: 11011001110110101101101001
Se detectaron errores. El mensaje se descarta por detectar errores.

[Done] exited with code=0 in 0.126 seconds
```

3. 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 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.

## 1001

Emisor:

```
thon.exe "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/transmitter.py"
Ingrese un mensaje en binario: 1001
Calculando checksum...
Listo!
Mensaje con checksum: 10010000100100001001
PS C:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02>
```

Receptor:

```
[Running] node "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/receiver.js"
Mensaje con checksum: 11010000100100001001
Se detectaron errores. El mensaje se descarta por detectar errores.

[Done] exited with code=0 in 0.126 seconds
```

## 110110

Emisor:

```
thon.exe "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/transmitter.py"
Ingrese un mensaje en binario: 110110
Calculando checksum...
Listo!
Mensaje con checksum: 1101100011011000110110
PS C:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02>
```

Receptor:

```
[Running] node "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/receiver.js"
Mensaje con checksum: 0011100011011000110110
Se detectaron errores. El mensaje se descarta por detectar errores.

[Done] exited with code=0 in 0.132 seconds
```

## 1001100111

Emisor:

```
thon.exe "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/transmitter.py"
Ingrese un mensaje en binario: 1001100111
Calculando checksum...
Listo!
Mensaje con checksum: 1001100111011010110110101001
PS C:\Users\Alejandro\Documents\UVG\Ingeniería en Ciencias de la Computación\8vo Ciclo\Redes\Lab02>
```

Receptor:

```
[Running] node "c:/Users/Alejandro/Documents/UVG/Ingeniería en Ciencias de la Computación/8vo Ciclo/Redes/Lab02/receiver.js"
Mensaje con checksum: 1111110111011010110110101001
Se detectaron errores. El mensaje se descarta por detectar errores.

[Done] exited with code=0 in 0.135 seconds
```

4. ¿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.

Si es posible, si se llegara a alterar alguno de los bits que corresponden al *checksum*, la validación en el receptor resultará en un error y se descartará el mensaje.

5. En base a las pruebas que realizó, ¿qué ventajas y desventajas posee cada algoritmo con respecto al otro?

- La implementación es mucho más sencilla comparada con otros algoritmos.
- El algoritmo sólo es capaz de detectar errores, pero no de corregirlos.
- Es vulnerable a interferencia/ruido, especialmente si se afectan los bits correspondientes al *checksum*.