Simulación de envío de mensajes con intercalado a través de un canal con ruido.

Seguridad informática - Universidad de León.

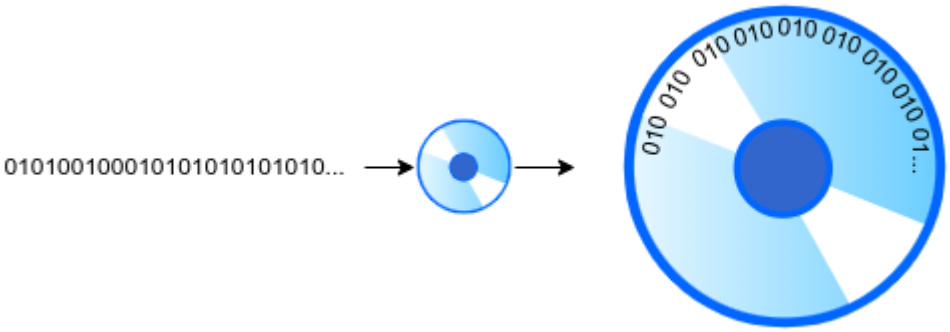
Naamán Huerga Pérez

Índice

Resumen

En este trabajo vamos a explorar la efectividad de la transmisión con intercalado a la hora de enviar un mensaje formado por bits por un canal afectado por errores a ráfagas.

Para visualizar que sería un canal afectado por los llamados errores a ráfagas podemos usar el ejemplo de un soporte físico como un CD o DVD. En un CD o DVD la información la escribimos a lo largo de la circunferencia del medio. Por ejemplo:



Si pensamos en un rallón en el dispositivo que altere los bits de la información, este no alteraría varios bits seguidos, sino que alteraría un bit por cada sector del disco. Esto es útil ya que dependiendo del código corrector que empleemos, podemos llegar a corregir una palabra con uno o dos errores, pero no podremos corregir una palabra entera errónea, como podría pasar si el error afecta a toda la palabra.

Para intentar que esto suceda en todas las comunicaciones que empleemos, nos podemos servir de una técnica de envío llamada ***intercalado de profundidad r***. Si tenemos un código con los parámetros *k, n* y *d*, el código se construye así:

$$a_1 = a_{11} \cdots a_{1k} \rightarrow c_1 = c_{11} \cdots c_{1n}$$

$$a_2 = a_{21} \cdots a_{2k} \rightarrow c_2 = c_{21} \cdots c_{2n}$$

$$\downarrow$$

$$a_r = a_{r1} \cdots a_{rk} \rightarrow c_r = c_{r1} \cdots c_{rn}$$

Y para transmitir estos símbolos, los enviamos por el canal en este orden:

$$c_{11}, c_{21}, c_{31} \cdots c_{r1}, c_{12}, c_{22}, c_{32} \cdots c_{r2} \cdots c_{1n}, c_{2n}, c_{3n} \cdots c_{rn}$$

Es decir, se transmiten por “columnas” según la tabla.

En el caso de que el mensaje a transmitir tenga más de *r* palabras, dividiremos el mensaje en bloques de *r* palabras y transmitiremos bloque por bloque.

Por ejemplo, si tenemos el siguiente mensaje a transmitir con profundidad $r = 3$:

$$c_1 = [0, 0, 0, 1]$$

$$c_2 = [0, 0, 1, 0]$$

$$c_3 = [0, 1, 1, 0]$$

La forma de enviar el mensaje en caso de no haber intercalado sería:

$$mensaje = 000100100110$$

Mientras que usando intercalado con profundidad $r = 3$:

$$mensaje = 000001011100$$

En el caso de que se produjera una ráfaga de errores de longitud 3 en mitad del mensaje, el mensaje enviado por intercalado se podría reconstruir íntegramente, mientras que el enviado de la forma tradicional no. Podemos verlo con un ejemplo representando los errores con el carácter x

$$mensaje_{tradicional} = 0001xxx00110$$

$$mensaje_{intercalado} = 0000xxx11100$$

Y tras reconstruir ambos mensajes:

Mensaje enviado tradicionalmente	**Mensaje enviado con intercalado**
$c_1 = [0, 0, 0, 1]$	$c_1 = [0, 0, x, 1]$
$c_2 = [x, x, x, 0]$	$c_2 = [0, x, 1, 0]$
$c_3 = [0, 1, 1, 0]$	$c_3 = [0, x, 1, 0]$

Podemos comprobar como en el mensaje enviado tradicionalmente no podríamos corregir el error, mientras que en el mensaje enviado con intercalado sí.

Vemos así como enviar un mensaje usando intercalado puede hacer que aumentemos la probabilidad de envíos exitosos en canales con errores.

En el caso de que la longitud de las ráfagas de errores superen la variable r, podemos encontrarnos con limitaciones a la hora de corregir las palabras recibidas. Por ejemplo, si en el mismo ejemplo anterior, la longitud de la ráfaga de error hubiera sido 4, y la capacidad correctora del código hubiera sido 1, observamos como nos quedamos sin poder corregir la segunda palabra:

$$c_1 = [0,\ 0,\ x,\ 1]$$

$$c_2 = [0,\ x,\ x,\ 0]$$

$$c_3 = [0,\ x,\ 1,\ 0]$$

Otro aspecto a considerar es la cola puedan presentar un bloque a la hora de transmitirse si tiene menos de r palabras. Por ejemplo, siendo $r = 3$, si queremos transmitir las siguientes palabras:

$$c_1 = [0,\ 0,\ 0,\ 1]$$

$$c_2 = [0,\ 0, 1,\ 0]$$

Tendríamos que añadir de alguna manera una “cola”. En nuestro caso usaremos una palabra formada por símbolos c. En el caso de que una palabra no ocupe todas las posiciones de longitud n (cola del mensaje codificado lineal, también completamos con c).

$$mensaje_{intercaladoConCola} = 00c00c01c10c$$

También tenemos que tener esto en cuenta a la hora de deshacer el intercalado, considerando los caracteres añadidos como caracteres de cola y omitirlos del mensaje que reconstruyamos.

A continuación, expongo el funcionamiento del algoritmo implementado para simular el envío de un mensaje a través de un canal con ruido, usando la técnica de intercalado para enviarlo y el algoritmo del líder junto con un tablero de errores incompleto para corregir los posibles errores producidos.

Procedimiento

El mensaje que enviemos va a pasar por todas las fases que hemos definido en la asignatura para enviar un mensaje: codificación de la fuente, codificación lineal, transmisión por el canal, introducción de error, corrección de errores, decodificación lineal y decodificación del alfabeto.

En los siguientes apartados entraremos en profundidad en cada fase para indicar que se ha hecho en la simulación y porqué.

Definición de parámetros

El primer paso que tomaremos será definir los parámetros del programa. En concreto, para poder realizar la ejecución, definiremos la matriz G de la forma $(Id|A)$, la matriz H de la forma $(-A^t|Id)$, la profundidad de intercalado r, el alfabeto que usamos y el mensaje que enviaremos a través del canal. Para exponer el ejemplo, usaremos los siguientes parámetros:

```
Matriz_A = [[0, 0, 1],
           [0, 1, 0],
           [1, 0, 0]]


Matriz_G = [[1, 0, 0, 1, 1, 0],
           [0, 1, 0, 1, 0, 1],
           [0, 0, 1, 0, 1, 1]]


Matriz_H = [[1, 1, 0, 1, 0, 0],
           [1, 0, 1, 0, 1, 0],
           [0, 1, 1, 0, 0, 1], ]


profundidad_intercalado = 9
# Mensaje de entrada
mensaje = "ejemplo"
# Definición de alfabeto
alf =
"aábcdeéAÁBCDEÉfghiíjklmnFGHIÍJKLMNoópqrstuúvwxyzOÓPQRSTUÚVWXYZ.,;
¿?¡!"
```

Codificación de la fuente

El siguiente paso a la hora de transmitir un mensaje es codificar la fuente. La codificación de la fuente consiste en transformar los símbolos del alfabeto de origen a símbolos de otro alfabeto, en este caso, binario. Para ello, creamos un diccionario que asocie cada letra del alfabeto a un número binario según su posición en el alfabeto. El número en binario que generemos tiene que tener una longitud adecuada. Esta longitud la sacamos escogiendo la parte entera de la siguiente operación:

$$log_2(longitud_{alfabeto})$$

Entonces, este diccionario nos quedaría así:

$$a_1 \rightarrow a \rightarrow 0000000$$

$$a_2 \rightarrow b \rightarrow 0000001$$

$$\bullet\bullet\bullet$$

$$a_{69} \rightarrow\ ! \rightarrow 1000101$$

Después de realizar este diccionario, traducimos todo el mensaje original de símbolos a cadena binaria:

$mensaje_{original} = hey!$

$mensaje_{codificadoFuente} = 001000100001010101111000101$

Una vez tenemos esta codificación realizada, podemos pasar al siguiente paso: la codificación lineal.

Codificación lineal

La codificación lineal consiste en pasar la cadena binaria que tenemos tras realizar la codificación de la fuente por una operación para generar un código con ciertas propiedades que nos permiten detectar errores y corregirlos. Para generar la nueva cadena de bits que usaremos, tenemos que dividir el mensaje codificado linealmente en trozos de longitud k, siendo k el número de filas de la matriz G.

Una vez dividimos el mensaje, cada trozo tenemos que multiplicarlo por la matriz generadora G. En el caso de que uno de los trozos no tuviera la misma longitud que la

variable k, lo consideraremos como cola, y lo añadiremos al final de la nueva cadena de bits que generemos.

Tras dividir el mensaje codificado por la fuente, multiplicar cada trozo por la matriz G y añadir la cola al final, el mensaje que queda es:

$$mensaje_{codificadoLineal} = 0010110\ 0000010\ 0110001\ 0110101\ 0110110\ 1111000\ 1001100\ 101011$$

Una vez tenemos esta cadena de bits codificada linealmente, podemos proceder a la transmisión por ráfagas.

Transmisión por ráfagas

Para transmitir el mensaje por ráfagas, tenemos que organizar uno o varios bloques de transmisión. Cada bloque de transmisión tendrá r cadenas de longitud n, siendo r la profundidad de intercalado y n el número de columnas de la matriz G. En el ejemplo que hemos propuesto, solo montaremos un bloque de transmisión ya que solo tenemos 8 cadenas de bits tras la codificación lineal, pero debemos de tener en cuenta los siguientes aspectos:

1. Las cadenas de bits de longitud menor que n tendremos que rellenarlas con algo. En esta simulación emplearemos el símbolo *
2. Cuando un bloque no tiene las suficientes cadenas de bits como para rellenarse entero, lo rellenaremos añadiendo cadenas de símbolos *

De esta manera, el bloque de transmisión que nos queda será así:

[0, 0, 1, 0, 1, 1, 0]
[0, 0, 0, 0, 0, 1, 0]
[0, 1, 1, 0, 0, 0, 1]
[0, 1, 1, 0, 1, 0, 1]
[0, 1, 1, 0, 1, 1, 0]
[1, 1, 1, 1, 0, 0, 0]
[1, 0, 0, 1, 1, 0, 0]
[1, 0, 1, 0, 1, 1, *]
[*, * , * , * , * , * , *]

A continuación tenemos que transmitir este bloque por el canal por columnas:

$$mensj_{intercalado} = 00000111*00111100*10111101*00000110*10011011*11001001*0011000**$$

Y una vez tenemos esta cadena de bits, podemos transmitirla por el canal con ruido.

Simulación de ruido en el canal

Al enviar el mensaje por el canal, tenemos que simular ruido por ráfagas. La forma que usaremos para esto será definir una probabilidad de error y una longitud máxima de ráfaga de error por cada bloque de transmisión.

La longitud máxima de ráfaga de error será el producto de la profundidad de intercalado r y la capacidad correctora del código t. En nuestro caso, la capacidad correctora del código es 1, por lo que la longitud máxima de ráfaga de error será 9.

En cuanto a probabilidad de error, usaremos $\frac{1}{20}$.

Recorreremos la cadena de bits obtenida en el paso anterior y a cada bit que recorramos, calcularemos un número aleatorio entre 0 y 1. Si el número aleatorio generado es menor que la probabilidad de error, alteramos los siguientes $longitud\ máxima\ rafaga\ errores$ bits. La forma de alterarlos será cambiar el bit correspondiente a uno aleatorio. Si al alterar un bit detectamos un carácter especial, como el *, no lo alteramos.

Para visualizar mejor la introducción de errores, ilustraremos con una x los lugares donde ocurre un error al ahora de transmitir un bloque de ráfagas:

$$mensj_{intercal.} = 00000111*00111100*10111101*00000110*10011011*11001001*0011000**$$

$$mensj_{errores} = 00000111*00111100*xxxxxxxx*00000110*10011011*11001001*0011000**$$

$$mensj_{errores} = 00000111*00111100*01101100*00000110*10011011*11001001*0011000**$$

Al poner un límite a la longitud máxima de la ráfaga y el número de ráfagas de error por bloque de transmisión nos aseguramos que el código se pueda corregir. Esto no ocurre así en el mundo real, pero en este trabajo solo se está exponiendo una simulación.

Deshacer la transmisión por ráfagas

Para deshacer la transmisión por ráfagas, tendremos que construir una matriz con dimensiones $r \times n$. Introduciremos los bits por columnas y desharemos en intercalado por filas, invirtiendo el proceso de intercalado. A mayores, cuando deshagamos el intercalado, eliminaremos todos los caracteres especiales que hayamos introducido.

La cadena de bits obtenida tras deshacer el intercalado es la siguiente:

$$mensaje_{trasIntercalado} = 000011000100100110001010010101101101111000100110010011$$

Corrección de ruido

Una vez obtenido el mensaje con el intercalado deshecho, tenemos que detectar y corregir los errores. Para ello, usamos el algoritmo del líder.

El primer paso es generar todos los errores patrón posibles. Para ello, generamos todas las cadenas de bits posibles de longitud n. En nuestro caso, como operamos con un código binario, serían 2^n posibilidades:

$$e_p^1 = 000000, e_p^2 = 000001, e_p^3 = 000010, \quad \cdots \quad , e_p^{62} = 111110, e_p^{63} = 111111$$

Una vez tenemos todos los errores patrones generados, generamos sus síndromes trasponiendo cada error patrón y multiplicándose por la matriz de control H. Tras esto, por cada síndrome generado, nos quedamos únicamente con el error patrón de menor peso que corresponda a ese síndrome, obteniendo así un tablero de errores patrón incompleto. En el ejemplo que estamos explicando, el tablero incompleto se vería así:

Error patrón	Síndrome
000000	000
000001	001
000010	010
001000	011
000100	100
010000	101
100000	110
001100	111

Con este tablero generado, únicamente tenemos que recorrer la cadena de bits recibida, dividirla en trozos de longitud n, calcular sus síndromes multiplicando la matriz de control H por el vector columna del trozo en cuestión y en el caso de que ese producto no de un síndrome que se corresponda con un vector columna lleno de 0, restamos el error patrón que tenga el mismo síndrome que el trozo.

Una vez hayamos recorrido y corregido todos los trozos, estamos preparados para realizar la decodificación lineal del mensaje.

Decodificación lineal

Para decodificar linealmente, tendríamos que calcular la aplicación inversa a la aplicación de codificación que hayamos usado. En nuestro caso, como hemos usado una matriz generadora G de la forma $(Id|A)$, podemos simplemente dividir el mensaje a decodificar linealmente en trozos de longitud n, y quedarnos únicamente con los k primeros bits de cada trozo, conservando la cola final si hubiera. En el ejemplo propuesto, la cadena de bits ya corregida que recibimos del paso anterior es la siguiente:

$$mensj_{codLineal} = 00101100000010011000101101010110110111100010011001010 11$$

Conservando solo los k primeros bits de cada trozo de longitud n, podemos construir un mensaje al que solo habría que decodificarle la fuente, que será el paso siguiente:

$$mensj_{codFuente} = 0010001000010101011111000101$$

Decodificación de la fuente

Para decodificar la fuente, simplemente hay que referirse al diccionario que construimos en el primer paso que relaciona cadenas de bits de longitud 7 con letras del alfabeto.

Dividiremos el mensaje en trozos de longitud 7 (puesto que es la longitud mínima binaria que hemos calculado anteriormente) y traduciremos cada cadena a bits a la letra que corresponda del alfabeto:

Cadena de bits	Símbolo correspondiente
0010001	h
0000101	e
0101111	y
1000101	!

Y como podemos observar, tras todos los pasos, introducir intercalado y corregir errores del canal, el mensaje obtenido es el mismo que hemos mandado

Ejemplo 2

Dado que se ha desarrollado un programa que realiza todas estas simulaciones de forma automática, podemos probar el mismo procedimiento con otro mensaje distinto para ilustrar que el método funciona independientemente del ejemplo.

Usaremos las mismas matrices G y H, y usaremos la profundidad de intercalado 9. Enviaremos el mensaje “Lana”:

Alfabeto: aábcdeé AÁBCDEÉfghiíjklmnFGHIÍJKLMNoópqrstuúvwxyzOÓPQRSTUÚVWXYZ.,;¿?¡!
Longitud del alfabeto: 70
Longitud mínima binaria: 7
Mensaje codificado fuente: 010000000000000110000000000
Longitud de la cola: 1
El mensaje codificado lineal que vamos a mandar es:
0101010 0000000 0000000 0000000 0001111 0000000 0000000 000000
Array de bloques de transmisión:

```
[[['0', '1', '0', '1', '0', '1', '0'],
  ['0', '0', '0', '0', '0', '0', '0'],
  ['0', '0', '0', '0', '0', '0', '0'],
  ['0', '0', '0', '0', '0', '0', '0'],
  ['0', '0', '0', '1', '1', '1', '1'],
  ['0', '0', '0', '0', '0', '0', '0'],
  ['0', '0', '0', '0', '0', '0', '0'],
  ['0', '0', '0', '0', '0', '0', '*'],
  ['*', '*', '*', '*', '*', '*', '*']]]
```

Mensaje que transmitiremos por el canal:
00000000*10000000*00000000*10001000*00001000*10001000*0000100**
Mensaje tras introducir error:
00000000*10000000*01101100*10001000*00001000*10001000*0000100**
Deshacemos las ráfagas y obtenemos este mensaje:
010101000100000010000000000000111110010000000000000000
Recibimos mensaje y corregimos ruido:
Generamos el tablero de errores patron incompleto con sus síndromes:
000000:000 , 000001:001, 000010:010, 001000:011, 000100:100, 010000:101, 100000:110, 001100:111
Corregimos mensaje:
010101000000000000000000000000111100000000000000000000

Comenzamos la decodificación lineal:

Dividimos el mensaje en trozos de longitud 6:

[['0', '1', '0', '1', '0', '1'],
['0', '0', '0', '0', '0', '0'],
['0', '0', '0', '0', '0', '0'],
['0', '0', '0', '0', '0', '0'],
['0', '0', '0', '0', '0', '0'],
['0', '1', '1', '1', '1', '0'],
['0', '0', '0', '0', '0', '0'],
['0', '0', '0', '0', '0', '0'],
['0', '0', '0', '0', '0', '0'],
['0']]

Como G es de forma estándar, **sacamos los 3 primeros bits de cada trozo:**

010000000000000011000000000

Tras decodificar la fuente, el mensaje final es: Lana

Conclusiones y vías futuras

Como hemos podido observar, hemos realizado con éxito una transmisión de un mensaje por un canal utilizando intercalado. Esto ha hecho que pese a tener una ŕafaga de 9 errores en el canal de transmisión, podamos corregir el mensaje entero y recibirlo de forma correcta. De no haber hecho la transmisión con intercalado y haberse producido una ráfaga de 9 errores, hubiera sido imposible reconstruir el mensaje original.

Determinamos así que enviar los mensajes por un canal usando una técnica de intercalado es una forma más fiable para enviar mensajes cifrados.

Las posibles vías futuras que tiene este proyecto es la simulación realista de los errores en un canal. Hasta ahora, hemos simulado los errores en el canal con unas condiciones que nos permiten asegurar que conseguiremos reconstruir el mensaje original. Sin embargo, en las transmisiones de verdad, no siempre se tienen que cumplir estas condiciones. Sería interesante estudiar en profundidad cómo se realizan las transmisiones por canales reales para poder simular a la perfección estas transmisiones, y ver qué porcentaje de éxito conseguimos transmitiendo mensajes usando esta técnica, así como comparar esta técnica con otras técnicas para mitigar errores, como el intercalado cruzado.

Referencias

Pisabarro Manteca, Maria Jesús. “Introducción a los Códigos Correctores de Errores: Códigos Lineales.” 2020.

Anexo: código fuente

El código fuente del proyecto se puede obtener de forma sencilla en este enlace, sin embargo, lo adjunto aquí por conveniencia:

```
from cmath import sin
import copy
import pprint
import math
import numpy as np
from pandas import array

#set the random seed to 123
np.random.seed(1234)

################################################
# DATOS DE ENTRADA PARA REALIZAR LA SIMULACIÓN #
################################################

# defininir los datos de entrada
Matriz_A = [[0, 0, 1],
            [0, 1, 0],
            [1, 0, 0]]

# Errores q admite = 1
Matriz_G = [[1, 0, 0, 1, 1, 0],
            [0, 1, 0, 1, 0, 1],
            [0, 0, 1, 0, 1, 1]]

n_columnas_G = len(Matriz_G[0])
n_filas_G = len(Matriz_G)

Matriz_H = [[1, 1, 0, 1, 0, 0],
            [1, 0, 1, 0, 1, 0],
            [0, 1, 1, 0, 0, 1], ]
n_filas_H = len(Matriz_H)
n_columnas_H = len(Matriz_H[0])
mgh = np.array(Matriz_H)

profundidad_intercalado = 5

# Mensaje de entrada
mensaje = "Lana"
# Definición de alfabeto
alf = "aábcdeé
AÁBCDEÉfghiíjklmnFGHIÍJKLMNoópqrstuúvwxyzOÓPQRSTUÚVWXYZ.,;¿?¡!"
```

```
# la longitud binaria de este alfabeto es 7
longitud_minima_binaria = math.ceil(math.log(len(alf), 2))
print("Alfabeto: ", alf)
print("longitud del alfabeto:", len(alf))
print("longitud minima binaria:", longitud_minima_binaria)

# Codificación de la fuente
diccionario_decodificacion = dict()
posicion = 0
for palabra in alf:
      diccionario_decodificacion[palabra] = bin(
      posicion)[2:].zfill(longitud_minima_binaria)
      posicion += 1

#print("diccionario de decodificación:", diccionario_decodificacion)

mensaje_codificado_fuente = ""
for letra in mensaje:
      mensaje_codificado_fuente += diccionario_decodificacion[letra]

print("Longitud del mensaje codificado fuente:",
len(mensaje_codificado_fuente))
print("Numero de caracteres del mensaje original:",
      len(mensaje_codificado_fuente)/longitud_minima_binaria)
print("Mensaje codificado fuente:", mensaje_codificado_fuente)


# Creamos el string que contendrá el mensaje codificado linealmente
mensaje_codificado_fuente_dividido =
[mensaje_codificado_fuente[i:i+n_filas_G]
                                    for i in range(0,
len(mensaje_codificado_fuente), n_filas_G)]
mensaje_codificado_lineal = ""
for chunk in mensaje_codificado_fuente_dividido:
      if(len(chunk) == n_filas_G):
      #print("chunk:", chunk)
      # create an array with chunk
      chunk_array = np.array(list(chunk))
      # convert chunk_array to int
      chunk_array = chunk_array.astype(int)
      # multiply chunk_array by matriz_G
      mga = np.array(Matriz_G)
      chunk_array_multiplicado = np.dot(chunk_array, mga)
      # get the module 2 of chunk_array_multiplicado
      chunk_array_multiplicado = np.mod(chunk_array_multiplicado, 2)
      # add to mensaje_codificado_lineal all the elements of
```

```
chunk_array_multiplicado as strings
      for i in range(0, len(chunk_array_multiplicado)):
            mensaje_codificado_lineal +=
str(chunk_array_multiplicado[i])
      else:
      # Cola
      print("Longitud de la cola:", len(chunk))
      mensaje_codificado_lineal += chunk

print("Longitud del mensaje codificado lineal:",
len(mensaje_codificado_lineal))
print("La profundidad del intercalado es: " +
str(profundidad_intercalado))

print("El mensaje codificado lineal que vamos a mandar es:")
for i in range(0, len(mensaje_codificado_lineal)):
      if(i % longitud_minima_binaria == 0 and i != 0):
      print(" ", end="")
      print(mensaje_codificado_lineal[i], end="")
print()


# Transmisión de lista por ráfagas
# Creamos un array de bloques de transmisión rxn simbolos

array_intercalado = []
array_intercalado =
[mensaje_codificado_lineal[i:i+longitud_minima_binaria]
                  for i in range(0, len(mensaje_codificado_lineal),
longitud_minima_binaria)]
# complete the last line of array_intercalado with "*" if it is not
complete
if(len(array_intercalado[len(array_intercalado)-1]) <
longitud_minima_binaria):
      for i in range(len(array_intercalado[len(array_intercalado)-1]),
longitud_minima_binaria):
      array_intercalado[len(array_intercalado)-1] += "*"

# add len(array_intercalado) % profundidad_intercalado  rows of "*" to
array_intercalado
for i in range(0, profundidad_intercalado - (len(array_intercalado) %
profundidad_intercalado)):
      array_intercalado.append("*"*longitud_minima_binaria)

# add a line of "-" to array_intercalado between each
profundidad_intercalado rows
```

```
puntos_de_corte = []
for i in range(0, len(array_intercalado)):
      if(i % (profundidad_intercalado) == 0):
      puntos_de_corte.append(i)

array_strings_bloques_transmision = []
for i in range(0, len(puntos_de_corte)):
      if(i == len(puntos_de_corte)-1):
      array_strings_bloques_transmision.append(

array_intercalado[puntos_de_corte[i]:len(array_intercalado)])
      else:
      array_strings_bloques_transmision.append(
            array_intercalado[puntos_de_corte[i]:puntos_de_corte[i+1]])

abt = []
for i in range(0, len(array_strings_bloques_transmision)):
      bloque = []
      for j in range(len(array_strings_bloques_transmision[i])):
      linea_bloque = []
      for k in range(0, len(array_strings_bloques_transmision[i][j])):

linea_bloque.append(array_strings_bloques_transmision[i][j][k])
      bloque.append(linea_bloque)
      abt.append(bloque)
print("Array de bloques de transmisión:")
pprint.pprint(abt)

# Como lo vamos a transmitir:
print("Mensaje que transmitiremos por el canal: ")
for i in range(0, len(abt)):
      for column in zip(*abt[i]):
      for element in column:
            print(element, end="")
print()

# extra
hay_cola = (abt[-1][-1][-1] == "*")

# TODO: introducir ruido random

prob_error = 1/20
ha_habido_error = False

capacidad_correctora = 1
max_long_rafaga = profundidad_intercalado * capacidad_correctora
```

```
num_errores = 0


for i in range(0, len(abt)):
      for k in range(0, len(abt[i][j])):
      for j in range(0, len(abt[i])):
            if(np.random.random() < prob_error):
                  prob_error = 1
                  ha_habido_error = True
                  if(num_errores < max_long_rafaga):
                  if(abt[i][j][k] != "*"):
                        abt[i][j][k] = str(np.random.randint(0, 2))
                        #abt[i][j][k] = "x"
                  num_errores += 1
      num_errores = 0
      prob_error = 1/20

print("Mensaje tras introducir error: ")
for i in range(0, len(abt)):
      for column in zip(*abt[i]):
      for element in column:
            print(element, end="")
print()
#print("¿Ha habido algún error al transmitir el mensaje? " +
str(ha_habido_error))

# Deshacemos las rafagas

mensaje_codificado_y_enviado = []
for i in range(0, len(abt)):
      for j in range(0, len(abt[i])):
      for k in range(0, len(abt[i][j])):
            if(abt[i][j][k] != "*"):
                  mensaje_codificado_y_enviado.append(abt[i][j][k])
print("Deshacemos las ráfagas y obtenemos este mensaje:")
for i in mensaje_codificado_y_enviado:
      print(i, end="")
print()
#print(mensaje_codificado_y_enviado)

# Corregir ruido
print("Recibimos mensaje y corregimos ruido: ")

# GENERAMOS TABLERO DE ERRORES PATRON
array_patron = []
for i in range(0, 2**n_columnas_H):
```

```
        array_patron.append(bin(i)[2:].zfill(n_columnas_H))
# print("Array de patrones:")
# #print(array_patron)
# for i in array_patron:
#     print(f"{array_patron.index(i)}:{i}")

# for each string in array_patron convert it to a column matrix
errores_patron = dict()
for i in range(0, len(array_patron)):
        error_matrix = np.array(list(array_patron[i]), dtype=int).T
        sindrome_error = np.dot(mgh, error_matrix)
        sindrome_error = np.mod(sindrome_error, 2)
        # get a string of the sindrome error
        sindrome_error_string = ""
        for j in range(0, len(sindrome_error)):
        sindrome_error_string += str(sindrome_error[j])
        # add to errores_patron array_patron[1] as key and the
sindrome_error_string as value
        # check if errores_patron has the key
        if(sindrome_error_string not in errores_patron):
        errores_patron[sindrome_error_string] = (array_patron[i])
        else:
        # count the number of 1s in errores_patron[sindrome_error_string]
        count_patron = 0
        for j in range(0, len(errores_patron[sindrome_error_string])):
                if(errores_patron[sindrome_error_string][j] == "1"):
                        count_patron += 1
        # count the number of 1s in array_patron[i]
        count_candidato = 0
        for j in range(0, len(array_patron[i])):
                if(array_patron[i][j] == "1"):
                        count_candidato += 1
        # if count_patron > count_candidato, replace the value of
errores_patron[sindrome_error_string] with array_patron[i]
        if(count_patron > count_candidato):
                errores_patron[sindrome_error_string] = (array_patron[i])


print("Tablero de errores patron incompleto con sus síndromes:")
#print(errores_patron)
for i in errores_patron:
        print(f"{errores_patron.get(i)}:{i}")

mensaje_corregido = []
```

```
for i in range(0, len(mensaje_codificado_y_enviado), n_columnas_H):
      # divide mensaje_codificado_y_enviado in chunks of n_columnas_H
      chunk = mensaje_codificado_y_enviado[i:i+n_columnas_H]
      # make a string with the elements of chunk
      chunk_string = ""
      for j in range(0, len(chunk)):
      chunk_string += chunk[j]
      # make a column matrix of the chunk
      chunk_matrix_col = np.array([chunk]).T
      # make chunk_matrix_col a int matrix
      chunk_matrix_col = chunk_matrix_col.astype(int)
      # if chunk_matrix_col has not n_columnas_H rows, do nothing
      if(chunk_matrix_col.shape[0] != n_columnas_H):
      for i in range(0, len(chunk)):
            mensaje_corregido.append(chunk[i])

      else:
      # multiply chunk_matrix_col by mgh
      sindrome_palabra = np.dot(mgh, chunk_matrix_col)
      # get chunk_matrix_col_multiplicado in module 2
      sindrome_palabra = np.mod(sindrome_palabra, 2)
      # if chunk_matrix_col_multiplicado is all 0, add chunk to
mensaje_corregido
      # sum all elements of sinrom_palabra
      suma_sindrome_palabra = 0
      for i in range(0, len(sindrome_palabra)):
            suma_sindrome_palabra += sindrome_palabra[i]
      if(suma_sindrome_palabra == 0):
            for i in range(0, len(chunk)):
                  mensaje_corregido.append(chunk[i])
      else:
            # Error aquí
            # get a string of the sindrome_palabra
            sindrome_palabra_string = ""
            for j in range(0, len(sindrome_palabra)):
                  for i in range(0, len(sindrome_palabra[j])):
                  sindrome_palabra_string += str(sindrome_palabra[j][i])
            # check if errores_patron has the key
            if(sindrome_palabra_string in errores_patron):
                  error_patron = errores_patron[sindrome_palabra_string]
                  # substract error_patron to mensaje_corregido
                  palabra_corregida = ""
                  # for each position in chunk_string, subtract the
value of error_patron at the same position
                  for i in range(0, len(chunk_string)):
                  value_chunk = int(chunk_string[i])
```

```
                value_error_patron = int(error_patron[i])
                value_position = np.mod(
                      value_chunk - value_error_patron, 2)
                palabra_corregida += str(value_position)
                # add palabra_corregida to mensaje_corregido
                for i in range(0, len(palabra_corregida)):
                mensaje_corregido.append(palabra_corregida[i])

print("Mensaje corregido: ")
for i in mensaje_corregido:
      print(i, end="")
print()

# Decodificación lineal
print("Comenzamos la decodifición lineal")

# COMENZAMOS PASO 3 - DECODIFICACION LINEAL (SIN RUIDO)
# Dividimos en trozos iguales que el numero de columnas de G
mensaje_codificado_lineal_dividido = []
palabra_codificada_lineal = []
columna = 0
contador = 0
for i in range(0, len(mensaje_corregido)):
      palabra_codificada_lineal.append(mensaje_corregido[i])
      contador += 1
      if(contador == n_columnas_G):
      contador = 0

mensaje_codificado_lineal_dividido.append(palabra_codificada_lineal)
      palabra_codificada_lineal = []
mensaje_codificado_lineal_dividido.append(palabra_codificada_lineal)


# Ahora tenemos en mensaje_codificado_dividido cada fila con 6 bits
print(f"Dividimos el mensaje en trozos de longitud {n_columnas_G}")
pprint.pprint(mensaje_codificado_lineal_dividido)
# print(len(mensaje_codificado))

# Ahora sacamos los n_filas_G primeros bits (porque G es de la forma
ID|A)
mensaje_codificado_binario = []
contador = 0
for fila in mensaje_codificado_lineal_dividido:
      if(len(fila) < n_columnas_G):
      for i in fila:
            mensaje_codificado_binario.append(i)
```

```
      else:
      for i in range(0, n_filas_G):
            mensaje_codificado_binario.append(fila[i])
print(
      f"Como G es de forma estándar, sacamos los {n_filas_G} primeros
bits de cada trozo")
#print(mensaje_codificado_binario)
for i in mensaje_codificado_binario:
      print(i, end="")
print()

# Ahora sacamos el diccionario para decodificar segund alf
diccionario_decodificacion = dict()
posicion = 0
for palabra in alf:
      diccionario_decodificacion[palabra] = '{0:07b}'.format(posicion)
      posicion += 1

# print("Usaremos el siguiente diccionario de decodificación:")
# pprint.pprint(diccionario_decodificacion)

mensaje_codificado_binario_dividido = []
palabra_codificada_binario = []
columna = 0
contador = 0
for i in range(0, len(mensaje_codificado_binario)):
      palabra_codificada_binario.append(mensaje_codificado_binario[i])
      contador += 1
      if(contador == longitud_minima_binaria):
      contador = 0

mensaje_codificado_binario_dividido.append(palabra_codificada_binario)
      palabra_codificada_binario = []

# pprint.pprint(mensaje_codificado_binario_dividido)

diccionario_invertido = {v: k for k, v in
diccionario_decodificacion.items()}
# AHORA POR CADA POSICION SACAR NUMERO
palabra_en_binario = str()
mensaje_final = str()
contador = 0
# pprint.pprint(mensaje_codificado_binario_dividido)
for fila in mensaje_codificado_binario_dividido:
      for numero in fila:
      palabra_en_binario += str(numero)
```

```
        mensaje_final += str(diccionario_invertido[palabra_en_binario])

        palabra_en_binario = str()

print("Tras decodificar la fuente, el mensaje final es: ",
mensaje_final)
```