Práctica 1

Javier Abad Hernández

In [1]:

```
#-----------------------------------------------------------------------------#
#-------------------------------EJERCICIO 1-----------------------------------#
#-----------------------------------------------------------------------------#

# Primero creamos un cuerpo finito. En este caso de 7 elementos.
# Esto se realiza mediante estos dos comandos.
C1 = GF(7)
C2 = FiniteField(7)

# Despues hemos de representar los elementos. Tanto C1 como C2 dan lo mismo.

for i in C2:
    print (i)

# Vamos a hacer una suma
print("Suma: ",  C1(2), "+", C1(5), "=", C1(2)+C1(5)) #se accede por indice

# Vamos a hacer una multiplicación
print("Multiplicacion: ",  C1(2), "·", C1(5), "=", C1(2)*C1(5)) #se accede por indice
```

```
0
1
2
3
4
5
6
Suma:  2 + 5 = 0
Multiplicacion:  2 · 5 = 3
```

In [2]:

```
# Ahora vamos a crear un cuerpo finito de una potencia de un numero primo de elementos.
# En este caso de 2^5 elementos.
C= GF(2^5, 'a')

# Vamos a representar sus elementos.
for i, x in enumerate(C):
    print("{} {}".format(i,x))

# De nuevo realizamos la suma.
s1= C.random_element()
s2= C.random_element()
print("Suma: ",  s1, "+", s2, "=", s1+s2) #se obtienen los elementos random del codigo

# De nuevo realizamos la multiplicación.
m1= C.random_element()
m2= C.random_element()
print("Multiplicación: ",  m1, "·", m2, "=", m1*m2) #se obtienen los elementos random d
el codigo
```

```
0 0
1 a
2 a^2
3 a^3
4 a^4
5 a^2 + 1
6 a^3 + a
7 a^4 + a^2
8 a^3 + a^2 + 1
9 a^4 + a^3 + a
10 a^4 + 1
11 a^2 + a + 1
12 a^3 + a^2 + a
13 a^4 + a^3 + a^2
14 a^4 + a^3 + a^2 + 1
15 a^4 + a^3 + a^2 + a + 1
16 a^4 + a^3 + a + 1
17 a^4 + a + 1
18 a + 1
19 a^2 + a
20 a^3 + a^2
21 a^4 + a^3
22 a^4 + a^2 + 1
23 a^3 + a^2 + a + 1
24 a^4 + a^3 + a^2 + a
25 a^4 + a^3 + 1
26 a^4 + a^2 + a + 1
27 a^3 + a + 1
28 a^4 + a^2 + a
29 a^3 + 1
30 a^4 + a
31 1
Suma:  a^2 + 1 + a^3 + a^2 + a + 1 = a^3 + a
Multiplicación:  a^4 + a^3 + a^2 · a + 1 = 1
```

In [3]:

```
#---------------------------------------------------------------------------#
#-------------------------------EJERCICIO 2----------------------------------#
#---------------------------------------------------------------------------#


#Defino un cuerpo finito en a
K.<a>=GF(3)
print(K)

#Matriz con 4 vectores que usen un cuerpo finito
G = matrix(GF(3), [[1, 0, 0, 0, 1, 2, 1],
                   [0, 1, 0, 0, 2, 1, 0],
                   [0, 0, 1, 2, 2, 2, 2],
                   [1, 0, 1, 2, 0, 1, 0]])

#Con la matriz defino el codigo
C = LinearCode(G)

##Parametros##
n = C.length() #longitud
k = C.dimension() #dimension
d = C.minimum_distance() #distancia minima

# Tenemos la matriz generadora
G =C.generator_matrix()
# y la de control que es H
H = C.parity_check_matrix()


print ("Matriz generadora: \n",G)
print ("Matriz control: \n",H)

# aqui vemos otra manera de calcular la matriz de control que es calculando
# la generadora del codigo dual.
D = C.dual_code()
print ("Matriz control: \n",D.generator_matrix())


# el polinomio de pesos tiene dos variables
polinomioPesos= C.weight_enumerator()
print("Polinomio de pesos: ",polinomioPesos)
#para que sea correcto con y=1
print("Polinomio de pesos con y=1: ", polinomioPesos(y=1))
# para ver los coeficientes del polinomio de pesos
print(C.weight_distribution())
print("\n")

# vemos la relacion que tienen la matriz generadora y la de control
print("Relacion de G y H:")
print(G*(H.transpose()))

## DESCODIFICACION
# Elegimos una palabra del código
pc = C.random_element()
print("\nPalabra cualquiera del código:" ,pc)
# Capacidad correctora del código C
capacidad = math.floor((d-1)/2)
capacidad
# Ahora defino r --> palabra pc introduciendo un error
```

```
error = vector(GF(3),[0,0,2,0,0,0,0])
errores2 = vector(GF(3),[0,0,1,2,0,0,0])
#defino r1 que es la palabra con un error y r2 lo mismo pero con 2 errores
r1 = pc + error
print("Palabra con un error: ",r1)
r2 = pc + errores2
r2
print("Palabra con dos errores: ",r2)

# Decodificamos r1
decodeR = C.decode_to_code(r1)
print("\nr1 decodificado: ",decodeR)
decodeR
# Vemos que r_dec = pc
if decodeR != pc:
    print("Palabras distintas")
else:
    print("Palabras iguales")
# decodificamos r2
decodeR = C.decode_to_code(r2)
print("\nr2 decodificado: ",decodeR)
# Vemos que r_dec2 != pc
if decodeR != pc:
    print("Son palabras distintas")
else:
    print("Son palabras iguales")
```

```
Finite Field of size 3
Matriz generadora:
 [1 0 0 0 1 2 1]
[0 1 0 0 2 1 0]
[0 0 1 2 2 2 2]
Matriz control:
 [1 0 0 0 2 2 2]
[0 1 0 0 1 0 2]
[0 0 1 0 2 2 0]
[0 0 0 1 1 1 0]
Matriz control:
 [1 0 0 0 2 2 2]
[0 1 0 0 1 0 2]
[0 0 1 0 2 2 0]
[0 0 0 1 1 1 0]
Polinomio de pesos:  2*x^7 + 4*x^6*y + 12*x^5*y^2 + 4*x^4*y^3 + 4*x^3*y^4
+ y^7
Polinomio de pesos con y=1:  2*x^7 + 4*x^6 + 12*x^5 + 4*x^4 + 4*x^3 + 1
[1, 0, 0, 4, 4, 12, 4, 2]


Relacion de G y H:
[0 0 0 0]
[0 0 0 0]
[0 0 0 0]

Palabra cualquiera del código: (0, 2, 1, 2, 0, 1, 2)
Palabra con un error:  (0, 2, 0, 2, 0, 1, 2)
Palabra con dos errores:  (0, 2, 2, 1, 0, 1, 2)

r1 decodificado:  (0, 2, 1, 2, 0, 1, 2)
Palabras iguales

r2 decodificado:  (1, 2, 2, 1, 0, 2, 2)
Son palabras distintas
```

En r1 el numero de errores es = 1, que coincide con la capacidad correctora del código, pero con r2 no, ahi el numero de erroes es =2 que es mayor que la capacidad correctora del código por lo que no descodifica correctamente.

In [4]:

```
#----------------------------------------------------------------------------#
#--------------------------------EJERCICIO 3---------------------------------#
#----------------------------------------------------------------------------#
# 3.1
## Creamos el cuerpo para red solomon con su longitud n, dimension k y distancia
## d.
F = GF(11)
n=10
k=5
d=6
t=((d-1)//2) # t es la capacidad correctora.


l0=n-1-t
l1=n-1-t-(k-1)
r=(5,9,0,9,0,1,0,7,0,5)

# creamos la matriz vacia
M=matrix(F,[[0]*((l0+1)+(l1+1))]*n)
print(M, "\n")

# obtenemos la base, y como 2 es elemento primitivo de F 11 lo podemos usar como 2î-1 m
od 11
basex=[]
for i in range (0, 10):
    basex.append((2^(i)).mod(11))

print("La base x es: ",basex, "\n")
print("recibimos", r, "\n")

# resuelvo el sistema de ecuaciones
for i in range(0, n):
    for j in range (0, (l0+1)):
        M[i,j]=basex[i]^j
    for j in range(l0+1, (l0+1+l1+1)):
        M[i,j]=(r[i]*basex[i]^(j-l0-1)).mod(11)
print(M, "\n")

V=M.right_kernel() #soluciones posibles
print(V, "\n")


l = V[1]
print(l, "\n")


# busco q0 y q1 en A[1]
Q0=0
Q1=0
R.<x> = PolynomialRing(GF(11))
for i in range (0,l0+1):
    Q0=Q0+l[i]*x^i
for i in range (l0+1, l0+1+l1+1):
    Q1=Q1+l[i]*x^(i-(l0+1))

print("Q0:", Q0)
print("Q1:", Q1, "\n")

# calculo g
```

```
g=-Q0/Q1
print("g:",g, "\n")


# vamos a ver si g(x) pertenece a pk
descodificado=[]
for i in range(n):
    descodificado.append(g(basex[i]))
print("descodificación: ", descodificado)
print()
# ahora si se ha superado la capacidad correctora del codigo.
distancia=0
errores=[]
for i in range(len(descodificado)):
    if descodificado[i] != r[i]:
        distancia += 1
        errores.append(i+1)
#print("la distancia es: ", distancia)

if(distancia > t):
    print("La decodificacion no es correcta, distancia>capacidad")
else:
    print("Se han corregido",distancia, "errores en las posiciones ", errores)
```

```
[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 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 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 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 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0]

La base x es:  [1, 2, 4, 8, 5, 10, 9, 7, 3, 6]

recibimos (5, 9, 0, 9, 0, 1, 0, 7, 0, 5)

[ 1  1  1  1  1  1  1  1  5  5  5  5]
[ 1  2  4  8  5 10  9  7  9  7  3  6]
[ 1  4  5  9  3  1  4  5  0  0  0  0]
[ 1  8  9  6  4 10  3  2  9  6  4 10]
[ 1  5  3  4  9  1  5  3  0  0  0  0]
[ 1 10  1 10  1 10  1 10  1 10  1 10]
[ 1  9  4  3  5  1  9  4  0  0  0  0]
[ 1  7  5  2  3 10  4  6  7  5  2  3]
[ 1  3  9  5  4  1  3  9  0  0  0  0]
[ 1  6  3  7  9 10  5  8  5  8  4  2]

Vector space of degree 12 and dimension 2 over Finite Field of size 11
Basis matrix:
[ 1  0  8 10 10  9 10  2 10  1  3  9]
[ 0  1  3  6  6  6  5  3  0 10  9  8]

(1, 0, 8, 10, 10, 9, 10, 2, 10, 1, 3, 9)

Q0: 2*x^7 + 10*x^6 + 9*x^5 + 10*x^4 + 10*x^3 + 8*x^2 + 1
Q1: 9*x^3 + 3*x^2 + x + 10

g: x^4 + x^3 + x^2 + x + 1

descodificación:  [5, 9, 0, 6, 0, 1, 0, 7, 0, 4]

Se han corregido 2 errores en las posiciones  [4, 10]
```

In [5]:

```
#--------------------------------------------------------------------------------#
#--------------------------------EJERCICIO 3------------------------------------#
#--------------------------------------------------------------------------------#
# 3.2
## Creamos el cuerpo para red solomon con su longitud n, dimension k y distancia
## d.

def ejercicio3_2(n,k,tamGF,r):
    F=GF(tamGF)
    d=n+1-k
    t=((d-1)//2) # t es la capacidad correctora.

    l0=n-1-t
    l1=n-1-t-(k-1)

    M=matrix(F,[[0]*((l0+1)+(l1+1))]*n)
    print(M, "\n")


    basex=[]
    for i in range (0, 10):
        basex.append((2^(i)).mod(tamGF))

    print("La base x es: ",basex, "\n")
    print("recibimos", r, "\n")

    for i in range(0, n):
        for j in range (0, (l0+1)):
            M[i,j]=basex[i]^j
        for j in range(l0+1, (l0+1+l1+1)):
            M[i,j]=(r[i]*basex[i]^(j-l0-1)).mod(tamGF)
    print(M, "\n")

    V=M.right_kernel() #soluciones posibles
    print(V, "\n")


    l = V[1]
    print(l, "\n")



    Q0=0
    Q1=0
    R.<x> = PolynomialRing(GF(tamGF))
    for i in range (0,l0+1):
        Q0=Q0+l[i]*x^i
    for i in range (l0+1, l0+1+l1+1):
        Q1=Q1+l[i]*x^(i-(l0+1))

    print("Q0:", Q0)
    print("Q1:", Q1, "\n")

    g=-Q0/Q1
    print("g:",g, "\n")


    # vamos a ver si g(x) pertenece a pk
    descodificado=[]
```

```
    for i in range(n):
        descodificado.append(g(basex[i]))
    print("descodificación: ", descodificado)
    print()
    # ahora si se ha superado la capacidad correctora del codigo.
    distancia=0
    errores=[]
    for i in range(len(descodificado)):
        if descodificado[i] != r[i]:
            distancia += 1
            errores.append(i+1)
    #print("la distancia es: ", distancia)

    if(distancia > t):
        print("La decodificacion no es correcta, distancia>capacidad")
    else:
        print("Se han corregido",distancia, "errores en las posiciones ", errores)



# Paso como argumentos, n, k, el tamaño del cuerpo finito, y la palabra.
ejercicio3_2(10, 5, 11, (5,9,0,9,0,1,0,7,0,5))

print("\n\n################## OTRA PRUEBA ################\n\n")

ejercicio3_2(3,1, 5, (0,0,1))

#LOS COMENTARIOS QUE FALTAN SON LOS MISMOS QUE EN EL APARTADO 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 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 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 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 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0]

La base x es:  [1, 2, 4, 8, 5, 10, 9, 7, 3, 6]

recibimos (5, 9, 0, 9, 0, 1, 0, 7, 0, 5)

[ 1  1  1  1  1  1  1  1  5  5  5  5]
[ 1  2  4  8  5 10  9  7  9  7  3  6]
[ 1  4  5  9  3  1  4  5  0  0  0  0]
[ 1  8  9  6  4 10  3  2  9  6  4 10]
[ 1  5  3  4  9  1  5  3  0  0  0  0]
[ 1 10  1 10  1 10  1 10  1 10  1 10]
[ 1  9  4  3  5  1  9  4  0  0  0  0]
[ 1  7  5  2  3 10  4  6  7  5  2  3]
[ 1  3  9  5  4  1  3  9  0  0  0  0]
[ 1  6  3  7  9 10  5  8  5  8  4  2]

Vector space of degree 12 and dimension 2 over Finite Field of size 11
Basis matrix:
[ 1  0  8 10 10  9 10  2 10  1  3  9]
[ 0  1  3  6  6  6  5  3  0 10  9  8]

(1, 0, 8, 10, 10, 9, 10, 2, 10, 1, 3, 9)

Q0: 2*x^7 + 10*x^6 + 9*x^5 + 10*x^4 + 10*x^3 + 8*x^2 + 1
Q1: 9*x^3 + 3*x^2 + x + 10

g: x^4 + x^3 + x^2 + x + 1

descodificación:  [5, 9, 0, 6, 0, 1, 0, 7, 0, 4]

Se han corregido 2 errores en las posiciones  [4, 10]


################# OTRA PRUEBA ################


[0 0 0 0]
[0 0 0 0]
[0 0 0 0]

La base x es:  [1, 2, 4, 3, 1, 2, 4, 3, 1, 2]

recibimos (0, 0, 1)

[1 1 0 0]
[1 2 0 0]
[1 4 1 4]

Vector space of degree 4 and dimension 1 over Finite Field of size 5
Basis matrix:
[0 0 1 1]
```

```
(0, 0, 1, 1)

Q0: 0
Q1: x + 1

g: 0

descodificación:  [0, 0, 0]

Se han corregido 1 errores en las posiciones  [3]
```

In [6]:

```
# 3.3
##Parametros##
n = 15
k = 3
d = n - k + 1
l = 2 # tamaño de lista
t = 6 #tau

# se define el GF y la "a" para referenciar a 'a'(alpha)
F = GF(16,'a')
a = F.0

base = [a^0,a^1,a^2,a^3,a^4,a^5,a^6,a^7,a^8,a^9,a^10,a^11,a^12,a^13,a^14]
r = [1,1,1,1,0,0,0,0,0,0,0,0,0,0,0] #Palabra recibida

#Matriz generadora inicializada a 0
G = matrix(F, [[0] * n] * n)
for i in range(n):
    for j in range(n):
        G[i,j] = base[i]^j

#Definimos el codigo lineal
# En esta parte es donde no he sabido muy bien seguir ni que hacer, realmente
# he intentado adaptar lo que tenia que no funcionaba con lo que colgó Diego en
# el campus pero tampoco lo he hecho funcionar, así que a partir de aqui lo que
# está es con esa modificación y dejé comentada la parte que cambie por lo nuevo
# DE TODAS FORMAS ESTÁ MAL. Asi que lo que nos devuelve es incorrecto.
# porque trate de mezclar dos cosas distintas realizadas en distintos momentos y
# ya no me acordaba bien de como iba
C = LinearCode(G)

S.<x>=F[]
R.<x,y>=F[]

x,y = PolynomialRing(F,2,['x','y']).gens()
Qxy = 0

#implemento el sumatorio
for j in range(l+1):

    ##primero calculo la matriz diagonal
    D = matrix(F, [[0] * n] * n)
    for i in range(n):
        D[i,i] = r[i]^j
    print(D)
    print()

    ## despues la segunda matriz
    lj = n-t-1-j*(k-1)
    M = matrix(F, [[0] * lj] * n)
    for i in range(lj):
        for j in range(n):
            M[j,i] = base[j]^i
    print(M)
    print()

    #cmultiplico las matrices una vez obtenidas
    X = D * M
    print(X)
```

```
    print()
    # esta es la solucion
    A = X.right_kernel()
    print(A)
    print()

    # tenemos que hayar Qj(x) y Q(x,y)
    # si existe una solución existiran los polinomios.
    if (len(A) > 1):

        Qj = 0
        var = (A[1])
        for k in range(lj):
            Qj = Qj + var[k] * (x^k)

        #Qxy
        Qxy = Qxy + Qj * y^j

##fin for

print("Q(x,y):", Qxy)
print()

##3)Procedemos a encontrar los factores de Q(x,y) de la forma (y-f(x)) con gradof(x) <
 k##
lista_factores=list(Qxy.factor())
poly=[]
for i in lista_factores:
    f=y-i[0]
    if f in S and f.degree()<k: #[0] porque devuelve una 2-upla con la multiplicidad
        f.change_ring(S) #puede que no sea necesario
        poly.append(f)
"""
if (Qxy != 0):
    Factor = factor(Qxy)

print(Factor)
print()

lista_factores = list(Factor)
"""
print(lista_factores)
lista_candid = []
for i in range(len(lista_factores)):
        grados = lista_factores[i][0].degrees()
        if grados[1] == 1 and grados[0] < k:
            lista_candid.append((lista_factores[i][0]-y)) #Multiplicar por el coeficien
te de y * -1


print("lista candidatos:", lista_candid)

# DE AQUI HACIA ABAJO ES COMO CREO QUE SE HACE PERO REALMENTE NO LO PUEDO DEMOSTRAR,
# YA QUE NO FUNCIONA POR LO ANTERIOR, TAMBIEN LO HE INTENTADO ADAPTAR A LO DEL CORREO
# POR LO QUE ES PROBABLE QUE ESTÉ AUN PEOR.
polinomio= (a^2 + 1) * x^2 + (a^3 + a)*x + 1
print(polinomio(x = 1))
print()

Factores = matrix(F,[[0] * n] * len(lista_candid))
print(Factores)
```

```
print()

for i in range(len(lista_candid)):
    polinomio= lista_candid[i]
    for j in range(n):
        Factores[i,j] = polinomio(x=base[j])

#Como output tendremos la lista formada por todos los fatores anteriores que verifiquen
--> d( (f(x1),...f(xn)),(r1,...,rn))<=tau
output = []

for i in range(len(lista_candid)):
    dist = 0
    polinomio= Factores[i]
    for j in range(n):
        if polinomio[j] != r[j]:
            dist += 1
    if dist <= t:
        output.append(polinomio)

print("Lista de palabras con distancia d < tau:", output)
```

```
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]

[                1                 1                 1                 1
1                 1                 1                 1]
[                1                 a               a^2               a^3
a + 1           a^2 + a         a^3 + a^2       a^3 + a + 1]
[                1               a^2             a + 1         a^3 + a^2
a^2 + 1       a^2 + a + 1 a^3 + a^2 + a + 1           a^3 + 1]
[                1               a^3         a^3 + a^2           a^3 + a a
^3 + a^2 + a + 1                 1               a^3         a^3 + a^2]
[                1             a + 1           a^2 + 1 a^3 + a^2 + a + 1
a           a^2 + a           a^3 + a     a^3 + a^2 + 1]
[                1           a^2 + a       a^2 + a + 1                 1
a^2 + a       a^2 + a + 1                 1           a^2 + a]
[                1         a^3 + a^2 a^3 + a^2 + a + 1               a^3
a^3 + a                 1         a^3 + a^2 a^3 + a^2 + a + 1]
[                1       a^3 + a + 1           a^3 + 1         a^3 + a^2
a^3 + a^2 + 1           a^2 + a a^3 + a^2 + a + 1             a + 1]
[                1           a^2 + 1                 a           a^3 + a
a^2       a^2 + a + 1               a^3     a^3 + a^2 + a]
[                1           a^3 + a               a^3 a^3 + a^2 + a + 1
a^3 + a^2                 1           a^3 + a               a^3]
[                1       a^2 + a + 1           a^2 + a                 1
a^2 + a + 1           a^2 + a                 1       a^2 + a + 1]
[                1     a^3 + a^2 + a       a^3 + a + 1               a^3
a^3 + 1       a^2 + a + 1         a^3 + a^2               a^2]
[                1 a^3 + a^2 + a + 1           a^3 + a         a^3 + a^2
a^3                 1 a^3 + a^2 + a + 1           a^3 + a]
[                1     a^3 + a^2 + 1     a^3 + a^2 + a           a^3 + a
a^3 + a + 1           a^2 + a               a^3                 a]
[                1           a^3 + 1     a^3 + a^2 + 1 a^3 + a^2 + a + 1
a^3 + a^2 + a       a^2 + a + 1           a^3 + a           a^2 + 1]

[                1                 1                 1                 1
1                 1                 1                 1]
[                1                 a               a^2               a^3
a + 1           a^2 + a         a^3 + a^2       a^3 + a + 1]
[                1               a^2             a + 1         a^3 + a^2
a^2 + 1       a^2 + a + 1 a^3 + a^2 + a + 1           a^3 + 1]
[                1               a^3         a^3 + a^2           a^3 + a a
^3 + a^2 + a + 1                 1               a^3         a^3 + a^2]
[                1             a + 1           a^2 + 1 a^3 + a^2 + a + 1
a           a^2 + a           a^3 + a     a^3 + a^2 + 1]
[                1           a^2 + a       a^2 + a + 1                 1
a^2 + a       a^2 + a + 1                 1           a^2 + a]
[                1         a^3 + a^2 a^3 + a^2 + a + 1               a^3
a^3 + a                 1         a^3 + a^2 a^3 + a^2 + a + 1]
```

```
[                1       a^3 + a + 1           a^3 + 1         a^3 + a^2
a^3 + a^2 + 1           a^2 + a a^3 + a^2 + a + 1            a + 1]
[                1           a^2 + 1                 a           a^3 + a
a^2       a^2 + a + 1              a^3     a^3 + a^2 + a]
[                1           a^3 + a               a^3 a^3 + a^2 + a + 1
a^3 + a^2               1          a^3 + a              a^3]
[                1       a^2 + a + 1           a^2 + a                 1
a^2 + a + 1          a^2 + a                 1      a^2 + a + 1]
[                1     a^3 + a^2 + a       a^3 + a + 1               a^3
a^3 + 1       a^2 + a + 1         a^3 + a^2               a^2]
[                1 a^3 + a^2 + a + 1           a^3 + a         a^3 + a^2
a^3                1 a^3 + a^2 + a + 1           a^3 + a]
[                1     a^3 + a^2 + 1     a^3 + a^2 + a           a^3 + a
a^3 + a + 1          a^2 + a              a^3                 a]
[                1           a^3 + 1     a^3 + a^2 + 1 a^3 + a^2 + a + 1
a^3 + a^2 + a       a^2 + a + 1          a^3 + a           a^2 + 1]

Vector space of degree 8 and dimension 0 over Finite Field in a of size 2^
4
Basis matrix:
[]

[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 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 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 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 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 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 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 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 0 0 0 0 0 0 0 0 0 0 0 0 0]

[                1                 1                 1                 1
1                 1]
[                1                 a               a^2               a^3
a + 1           a^2 + a]
[                1               a^2             a + 1         a^3 + a^2
a^2 + 1       a^2 + a + 1]
[                1               a^3         a^3 + a^2           a^3 + a a
^3 + a^2 + a + 1                 1]
[                1             a + 1           a^2 + 1 a^3 + a^2 + a + 1
a           a^2 + a]
[                1           a^2 + a       a^2 + a + 1                 1
a^2 + a       a^2 + a + 1]
[                1         a^3 + a^2 a^3 + a^2 + a + 1               a^3
a^3 + a                 1]
[                1       a^3 + a + 1           a^3 + 1         a^3 + a^2
a^3 + a^2 + 1           a^2 + a]
[                1           a^2 + 1                 a           a^3 + a
a^2       a^2 + a + 1]
[                1           a^3 + a               a^3 a^3 + a^2 + a + 1
a^3 + a^2                 1]
[                1       a^2 + a + 1           a^2 + a                 1
a^2 + a + 1          a^2 + a]
[                1     a^3 + a^2 + a       a^3 + a + 1               a^3
```

```
a^3 + 1       a^2 + a + 1]
[             1 a^3 + a^2 + a + 1          a^3 + a        a^3 + a^2
a^3              1]
[             1    a^3 + a^2 + 1     a^3 + a^2 + a          a^3 + a
a^3 + a + 1          a^2 + a]
[             1          a^3 + 1     a^3 + a^2 + 1 a^3 + a^2 + a + 1
a^3 + a^2 + a       a^2 + a + 1]

[             1                 1                 1                 1
1                 1]
[             1                 a               a^2               a^3
a + 1           a^2 + a]
[             1               a^2             a + 1         a^3 + a^2
a^2 + 1       a^2 + a + 1]
[             1               a^3         a^3 + a^2           a^3 + a a
^3 + a^2 + a + 1                 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                 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                 0                 0
0                 0]
[             0                 0                 0                 0
0                 0]
[             0                 0                 0                 0
0                 0]

Vector space of degree 6 and dimension 2 over Finite Field in a of size 2^
4
Basis matrix:
[           1             0       a^2 + 1   a^2 + a + 1   a^3 + a + 1
a^3]
[           0             1       a^3 + a a^3 + a^2 + 1     a^3 + a^2
a^3 + a]

[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 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 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 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 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 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 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 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 0 0 0 0 0 0 0 0 0 0 0 0]

[]

[]

Vector space of degree 0 and dimension 0 over Finite Field in a of size 2^
4
Basis matrix:
[]

Q(x,y): (a + 1)*x^5*y^14 + (a^2 + 1)*x^4*y^14 + (a^3 + a^2 + a)*x^3*y^14 +
(a^3 + a)*x^2*y^14 + (a)*y^14

[(x + (a^2), 1), (x + (a^3), 1), (x + (a), 1), (x + 1, 1), (x + (a^3 + a^
2), 1), (y, 14)]
lista candidatos: [0]
(a^3 + a^2 + a)

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Lista de palabras con distancia d < tau: [(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0)]
```

In [7]:

```
# 3.4

# NO SE HACERLO, YA QUE TIENE QUE VER CON EL ANTERIOR, SE QUE TIENES QUE BUSCAR
# SI LAS PALABRAS ESTAN EL EL CODIGO RED SOLOMON Y LUEGO JUGAR CON LAS PROBABILIDADES
# DEL ESTILO AL EJERCICIO 4, CON ITERACIONES Y DANDOSE EL CASO DE QUE ESTE EN LA DE MAS
# DE UN ELEMENTO QUE SUME 1 AL CONTADOR, PERO NO SE REALIZARLO.
```

In [8]:

```
#-----------------------------------------------------------------------------#
#-------------------------------EJERCICIO 4-----------------------------------#
#-----------------------------------------------------------------------------#

#######################################################################################
######################################
# ESTA VERSION CREO QUE ES MALA Y QUE ME HE COMPLICADO MUCHISIMO LA VIDA PARA HACERLO A
SÍ, HAY MANERAS MAS FACILES, AQUI SE #
# CALCULA TODO DIRECTAMENTE, AMBOS APARTADOS INCLUIDOS. Y DE TODAS FORMAS DIRECTAMENTE
 CREO QUE ESTA MAL ####################

def calculo_probabilidad(C, probabilidad_error):
    seed(0)
    iteraciones=10000
    veces_mal=0
    H=C.parity_check_matrix()

    for i in range (iteraciones):
        r=matrix(GF(2), [[0]*C.length()]*1)
        for i in range (C.length()):
            p=random()
            if(p<=probabilidad_error):
                r[0,i]=(C.random_element()[i]+1)
            else:
                r[0,i]=(C.random_element()[i])
        dist=0
        for z in range(C.length()):
            if(C.random_element()[z]!=r[0][z]):
                dist+=1
        if(dist!=0):
            esta=H*r.transpose()
            detectado=False
            for j in range(C.length()-C.dimension()):
                if(esta[j][0]!=0):
                    detectado=True
            if(detectado):
                veces_mal=veces_mal+1
    return (veces_mal)/iteraciones



G = matrix(GF(2), [[1, 1, 0, 0, 0, 0],
                   [0, 1, 1, 0, 0, 0],
                   [1, 1, 1, 1, 1, 1]])

C = LinearCode(G)

print(calculo_probabilidad(C,1/11))
print(calculo_probabilidad(C,1/11)*1.0)
```

```
4301/5000
0.864400000000000
```

In [27]:

```
#---------------------------------------------------------------------------#
#-------------------------------EJERCICIO 4 VERSION BUENA---------------------#
#---------------------------------------------------------------------------#
# 4.1 ESTE PRIMER APARTADO ESTA BIEN
import numpy as np

def versionEasi(e,p):
    for i in range (len(e)):
        if random()<p:
            if e[i]==0:
                e[i]=1
            else:
                e[i]=0
    return e


#---------------------------------------------------------------------------#
#4.2 LO HICE DE DOS MANERAS, LA PRIMERA QUE ES LA COMENTADA ES INCORRECTA Y NO
#FUNCIONA, ME SACA PROBABILIDADES CUANTO MENOS EXTRAÑAS, LA SEGUNDA ESTA BIEN
iteraciones = 10000
#veces_mal = 0
veces_bien = 0
G = matrix(GF(2), [[1, 1, 0, 0, 0, 0],
                   [0, 1, 1, 0, 0, 0],
                   [1, 1, 1, 1, 1, 1]])

C = LinearCode(G)
c1 = C.random_element()

H=C.parity_check_matrix()
p=0.5
seed(0)
"""
for i in range(iteraciones):
    igual = False
    while(not igual):
        c1 = C.random_element()
        e = copy(c1)
        comprobacion = vector((np.transpose(e) * H)[0])
        if comprobacion.is_zero():
            igual = True
            veces_bien +=1

    r = versionEasi(e, p)
    segundaComprobacion = vector((np.transpose(r) * H)[0])
    if segundaComprobacion.is_zero():
        veces_mal += 1
    else:
        veces_bien +=1



#Ahora dividimos cont, con el numero de errores, por las iteraciones para obtener la pr
obabilidad final
calculo_probabilidad = veces_mal/iteraciones
calculo_probabilidad2 = veces_bien/iteraciones
"""
```

```
veces_bien = 0

for i in range(iteraciones):
    e = copy(c1)
    r = versionEasi(e,p)
    decodificada = C.decode_to_code(r)

    if decodificada == c1 :
        veces_bien = veces_bien + 1
    else:
        continue
calculo_probabilidad2 = veces_bien/iteraciones

print("Probabilidad de decodificacion correcta, de forma experimental:", calculo_probab
ilidad2*1.0)

print()
```

```
Probabilidad de decodificacion correcta, de forma experimental: 0.13180000
0000000
```

In [ ]:

```
#Vemos que queda similar, bastante cercana la primera version y la segunda, la
#que creo que es la correcta vaya.
```

In [ ]:

```
4.3
### al realizar las pruebas, puedes comprobar que cuanto mayor sea la probabilidad
### menores codificaciones correctas habrá, esto se reduce de una manera considerable
### para por cada 0,1 que se aumente de probabilidad. con 0,1 obtengo sobre un 80&
### con 0,2 cerca de un 60%, con 0,3 bajo a 0.38 y con 0.5 un 12%
```