Software CRC

Gomez Lacchini, Lihuel Joaquin

Aritmética modular

5 mod 2 = 1

Propiedad:

(a*b) mod c = ((a mod c) * (b mod c)) mod c

Ejemplo con a= 5, b=3 y c=2

- (5*3)mod 2 = 15 mod 2 = 1
- ((5 mod 2)* (3 mod 2)) mod 2 = (1*1) mod 2 = 1

Ejemplo con mas factores:

(7 * 8 * 4 * 2) mod 3

448 mod 3 = 1

((7 mod 3)*(8 mod 3)*(4 mod 3)*(2 mod 3)) mod 3

(1 * 2 * 1 * 2) mod 3

4 mod 3 = 1

Aritmética modular

5 mod 2 = 1

Propiedad:

(a*b) mod c = ((a mod c) * (b mod c)) mod c

Ejemplo con a= 5, b=3 y c=2

- (5*3)mod 2 = 15 mod 2 = 1
- ((5 mod 2)* (3 mod 2)) mod 2 = (1*1) mod 2 = 1

Entonces:

(a*b) mod c = ((a mod c) * b) mod c

((5 mod 2) * 3) mod 2 = ((1)*3) mod 2 = 3 mod 2 = 1

Ejemplo con mas factores:

(7 * 8 * 4 * 2) mod 3
448 mod 3 = 1

((7 mod 3)*(8 mod 3)*(4 mod 3)*(2 mod 3)) mod 3
(1 * 2 * 1 * 2) mod 3
4 mod 3 = 1

((((7 mod 3)*8) mod 3)*4 mod 3) *2 mod 3
(((1 * 8) mod 3)*4 mod 3) *2 mod 3
((8 mod 3)*4 mod 3) *2 mod 3
((2)*4 mod 3) *2 mod 3
(8 mod 3) *2 mod 3
(2) *2 mod 3
4 mod 3
1

Trabaja para una empresa que utiliza muchos ordenadores personales.
Su jefe, el Dr. Penny Pincher, ha quería vincular las computadoras durante algún tiempo, pero no ha estado dispuesto a gastar dinero en las tarjetas Ethernet que ha recomendado. Usted, sin saberlo, ha señalado que cada uno de los Las PC provienen del proveedor con un puerto serie asíncrono sin costo adicional. Dr. Pincher, de Por supuesto, reconoce su oportunidad y le asigna la tarea de escribir el software necesario para permitir comunicación entre PC. Ha leído un poco sobre comunicaciones y sabe que cada transmisión está sujeta a errores y que la solución típica a este problema es agregar alguna información de verificación de errores al final de cada mensaje. Esta información permite al programa receptor detectar cuándo se ha producido un error de transmisión. ocurrido (en la mayoría de los casos). Entonces, vaya a la biblioteca, tome prestado el libro más grande sobre comunicaciones que puede encontrar y pasar su fin de semana (horas extras no pagadas) leyendo sobre la verificación de errores. Finalmente, decide que CRC (verificación de redundancia cíclica) es la mejor verificación de errores para su situación y escriba una nota al Dr. Pincher detallando el mecanismo de verificación de errores propuesto que se indica a continuación.

Generación de CRC

El mensaje que se va a transmitir se ve como un número binario positivo largo. El primer byte del mensaje se trata como el byte más significativo del número binario. El segundo byte es el siguiente más significativo, etc. Este número binario se llamará "m" (para el mensaje). En lugar de transmitir "m", transmitirá un mensaje, "m2", que consta de "m" seguido por un valor CRC de dos bytes. El valor CRC se elige de modo que "m2" cuando se divide por un cierto valor de 16 bits "g" deja un resto de 0. Esto facilita que el programa receptor determine si el mensaje ha sido dañado por errores de transmisión. Simplemente divide cualquier mensaje recibido por "g". Si el resto de la división es cero, se supone que no se ha producido ningún error. Observa que la mayoría de los valores sugeridos de "g" en el libro son extraños, pero no ve cualquier otra similitud, por lo que selecciona el valor 34943 para "g" (el valor del generador). Debe idear un algoritmo para calcular el valor CRC correspondiente a cualquier mensaje que podría ser enviado.

- Para probar este algoritmo, escribirá un programa que lee líneas de la entrada estándar y escribe en la salida estándar.

Input

Cada línea de entrada no contendrá más de 1024 caracteres ASCII (cada línea contiene todos los caracteres hasta, pero sin incluir el carácter de final de línea) como entrada. La entrada termina con una línea que contiene un "#" en la columna 1.

Output

Para cada línea de entrada calcula el valor CRC para el mensaje contenido en la línea y escribe el valor numérico de los bytes CRC (en notación hexadecimal) en una línea de salida. Tenga en cuenta que cada CRC impreso debe estar en el rango de 0 a 34942 (decimal).

Sample Input

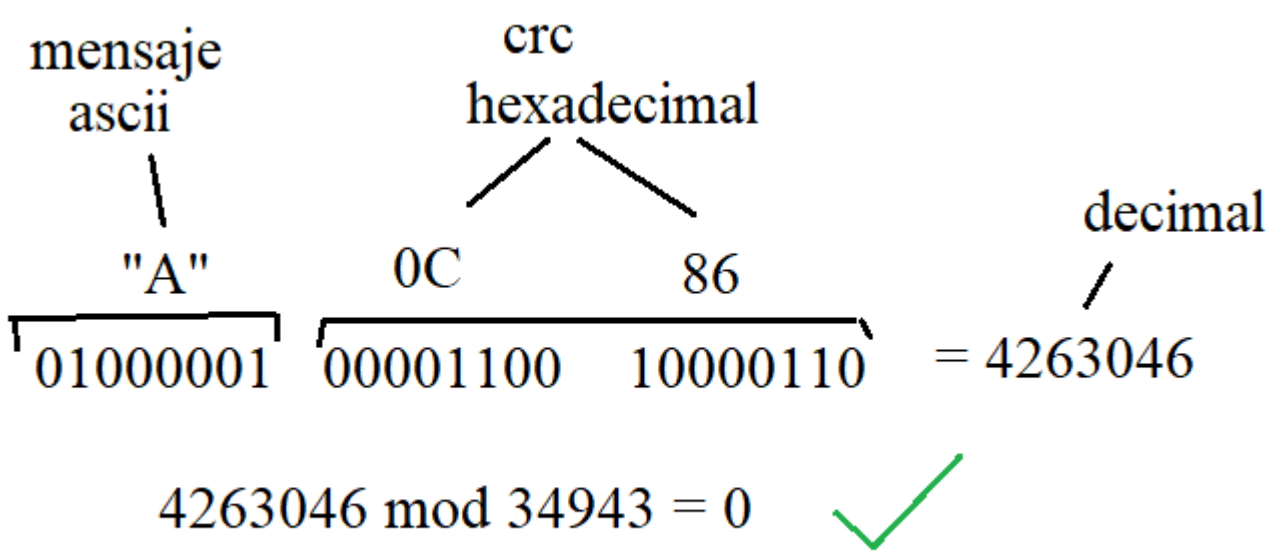
```
this is a test

A
#
```

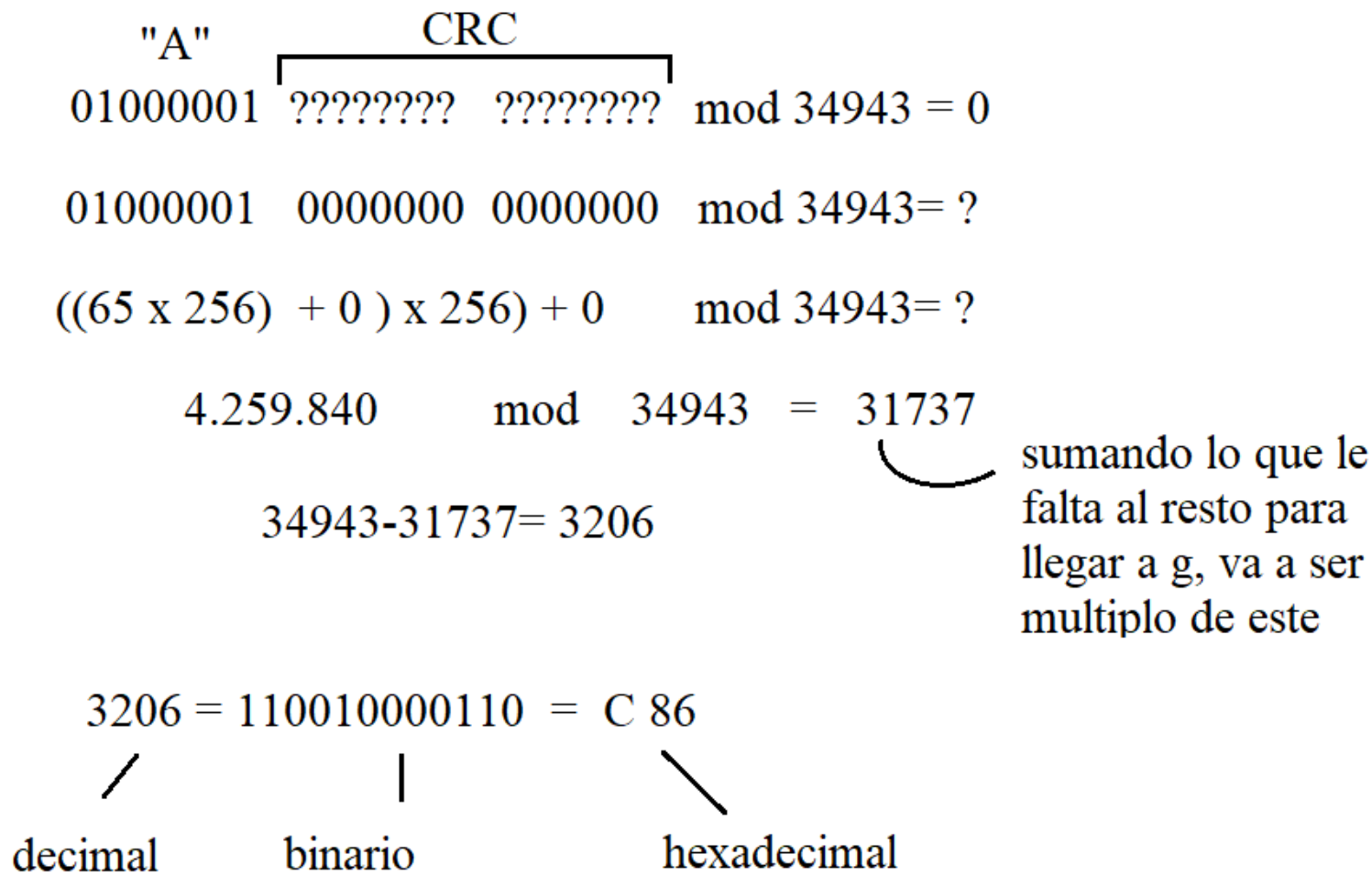
Sample Output

```
77 FD
00 00
0C 86
```

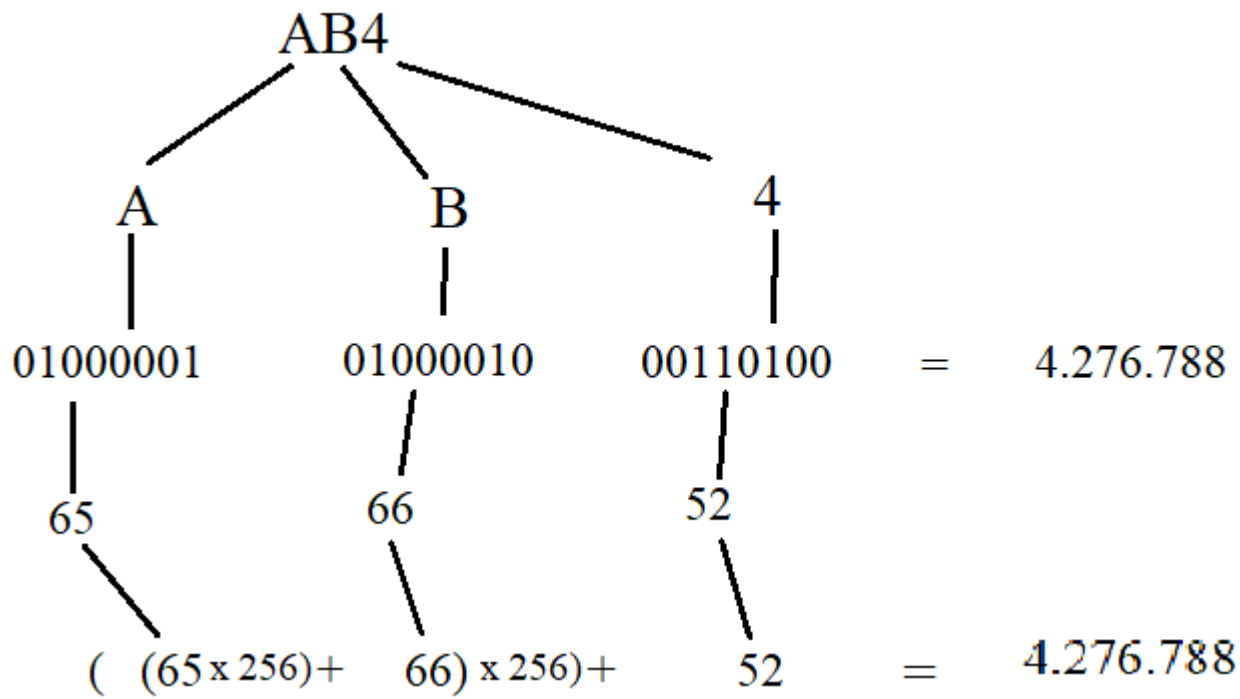
AB4 ⟶ ?



¿Como se puede llegar a eso?



AB4 ⟶ ?



01000001 01000010 00110100 ???????? ???????? mod 34943 = 0

A B 4 CRC

01000001 01000010 00110100 ???????? ???????? mod 34943 = 0

((((65x256) + 66) x256) + 52) x256) + ??) x256) +??) mod 34943 = 0

((((65x256) + 66) x256) + 52) x256) +0) x256) + 0 = 280.283.578.368

01000001 01000010 00110100 00000000 00000000 = 280.283.578.368

((((65x256) + 66) x256) + 52) x256) +0) x256) + 0 = 280.283.578.368

280.283.578.368 % 34943 = 8.021.165,2796...

280.283.578.368 mod 34943 = 9773

34943 - 9773 = 25170 ⟶ CRC

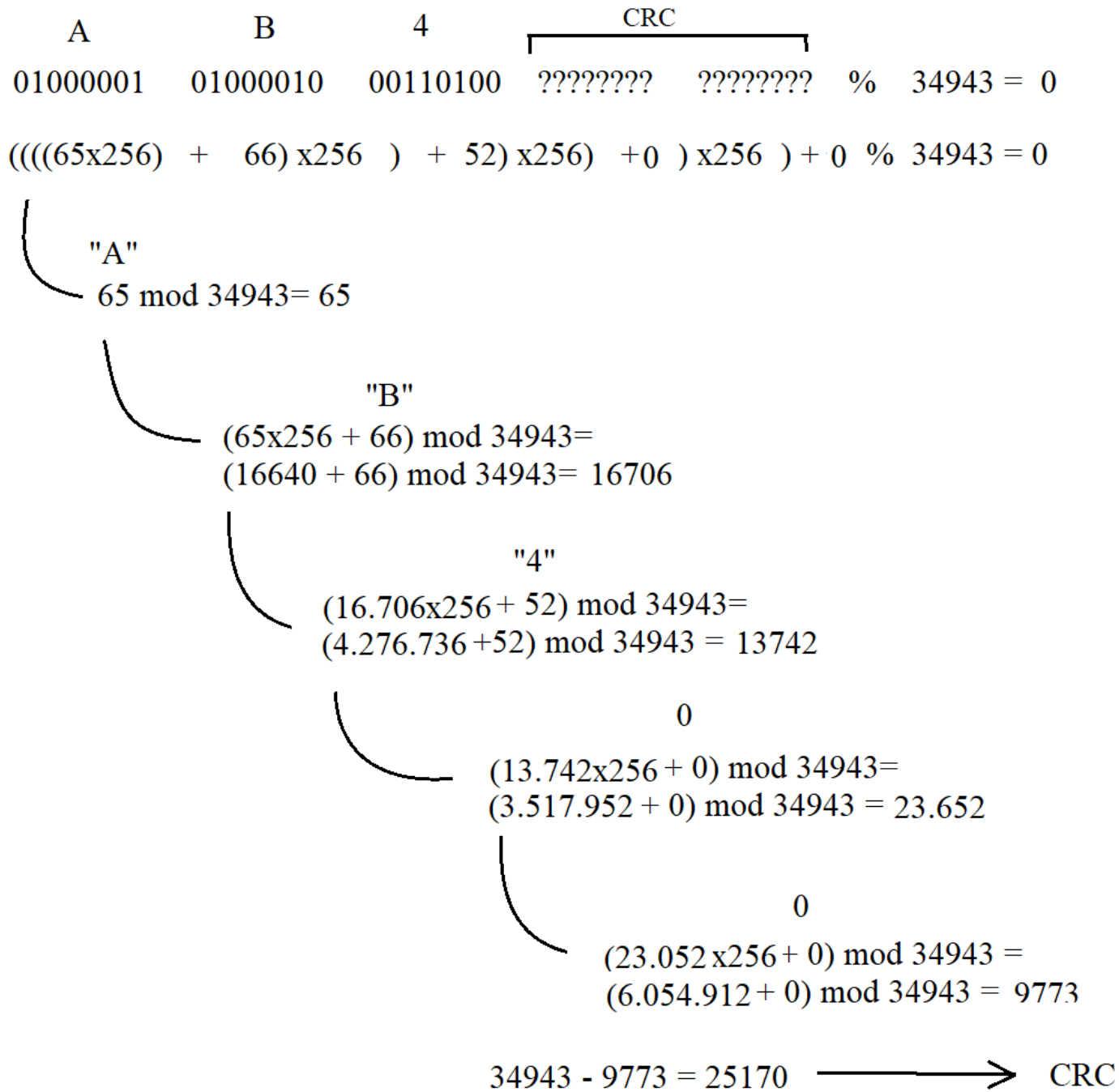
Demostración

280.283.578.368+25.170 = 280.283.603.538

280.283.603.538 % 34943 = 8.021.166

280.283.603.538 mod 34943 = 0

¿Cómo hacer que los números no se vuelvan exponencialmente mas grandes con mensajes mas largos?



```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    #Calculo CRC


    #Lo imprimo en Hexadecimal


    mensaje = input() #Leo otro mensaje
```

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #Lo imprimo en Hexadecimal

    mensaje = input() #Leo otro mensaje
```

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

AB4

resto=0

caracter A

resto = ((0*256) + 65) mod 34943

resto = 65 mod 34943

resto = 65

"A"

65 mod 34943= 65

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

AB4

resto = 65

caracter A B

resto = ((65*256)+66 mod 34943

resto = (16640+66) mod 34943

resto = 16706

"B"

(65x256 + 66) mod 34943=

(16640 + 66) mod 34943= 16706

```
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

resto = 16706

caracter ~~A~~ ~~B~~ 4

resto = ((16706*256) + 52) mod 34943

resto = (4376736+52) mod 34943

resto = 13742

"4"

(16.706x256 + 52) mod 34943=

(4.276.736 +52) mod 34943 = 13742

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

AB4

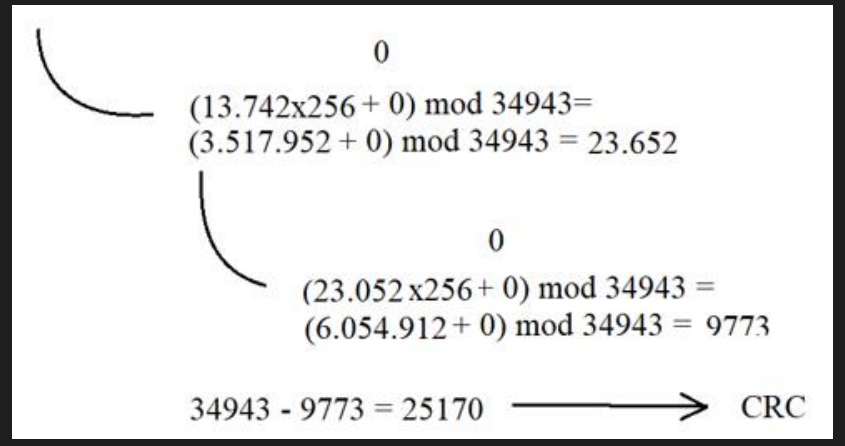
resto = 13742

resto= (13742x256x256) mod 34943

resto = 9773

CRCdecimal= (34943-9773) mod 34943

CRCdecimal = 25170



```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

AB4

CRCdecimal = 25170

CRChex= 0x6252

CRChex= 6252

CRChex= 6252

Output= 62 52

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = "A"

resto = 0

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = "A"

resto = 0

A

resto= ((0*256) +65) mod 34943

resto = 65 mod 34943 = 65

resto = (65*256*256) mod 34943

resto = 31737

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = "A"

resto = 0

A

resto= ((0*256) +65) mod 34943

resto = 65 mod 34943 = 65

resto = (65*256*256) mod 34943

resto = 31737

CRCdecimal = (34943-31737)mod 34943

CRCdecimal = 3206

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = "A"

resto = 0

A

resto= ((0*256) +65) mod 34943

resto = 65 mod 34943 = 65

resto = (65*256*256) mod 34943

resto = 31737

CRCdecimal = (34943-31737)mod 34943

CRCdecimal = 3206

CRChex= 0xc96

CRChex= c96

CRChex = C96

CRChex = 0C96

Output = 0C 96

```
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = ""

resto = 0

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = ""

resto = 0

resto = (0*256*256) mod 34943

resto = 0 mod 34943 = 0

CRCdecimal = (34943-0) mod 34943

CRCdecimal = 0

Si CRCdecimal en vez de (g-resto)%g hubiera sido solamente g-resto, hubiese quedado igual a 34943 que es incorrecto

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = ""

resto = 0

resto = (0*256*256) mod 34943

resto = 0 mod 34943 = 0

CRCdecimal = (34943-0) mod 34943

CRCdecimal = 0

CRChex = 0x0

CRChex= 0

CRChex = 0

CRChex = 0000

Output = 00 00

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = "tron"

resto = 0

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = "tron"

resto = 0

t

resto = ((0*256) + 116) mod 34943

resto = 116

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = "tron"

resto = 116

t r

resto = ((116*256)+114) mod 34943

resto = (29696+114) mod 34943

resto = 29810

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = "tron"

resto = 29810

o

resto = ((29810*256)+111) mod 34943

resto = (7631360+111)mod 34943

resto = 13897

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = "tron"

resto = 13897 t r o n

resto = ((13897*256)+110) mod 34943

resto = (3557632+110)mod 34943

resto = 28499

resto= (28499*256*256)mod 34943

resto = 1867710464 mod 34943

resto = 7114

CRCdecimal = (34943-7114)mod 34943

CRCdecimal=27829

```python
g= 34943
mensaje = input()
while mensaje != "#": #Termina cuando entra #
    resto=0
    #Calculo el resto del valor del mensaje dividido g
    for caracter in mensaje:
        resto= ( (resto*256) + ord(caracter) )% g
    #Calculo el resto del valor del mensaje con dos bytes vacios dividido g
    resto= (resto*256*256)%g
    #En CRCdecimal calculo cuanto falta para llegar a un multiplo de g (g-resto)
    #El %g es por si resto da 0, que devuelva 0 y no g
    CRCdecimal=(g-resto)%g

    #En CRCdecimal ya tengo el resultado del CRC de ese mensaje, ahora
    #hay que presentarlo como pide el output
    #Se convierte a decimal:
    CRChex= hex(CRCdecimal)
    #Al convertir a decimal se genera en modo 0xaaa, entonces le quito el "0x":
    CRChex=CRChex[2:(len(CRChex)+1)]
    #Convierto las letras a mayuscula:
    CRChex=CRChex.upper()
    #Si es menor de 4 digitos, entonces lo relleno con ceros delante:
    while len(CRChex)!=4:
        CRChex="0"+CRChex
    #Al imprimir pongo un espacio entre el primer byte y el segundo:
    print(CRChex[:2]+" "+CRChex[2:])

    mensaje = input() #Leo otro mensaje
```

mensaje = "tron"

resto = 13897

t r o n

resto = ((13897*256)+110) mod 34943

resto = (3557632+110)mod 34943

resto = 28499

resto= (28499*256*256)mod 34943

resto = 1867710464 mod 34943

resto = 7114

CRCdecimal = (34943-7114)mod 34943

CRCdecimal=27829

CRChex=0x6cb5

CRChex=6cb5

CRChex=6CB5

Output = 6C B5