INFORME DE ALGORITMO VIGENERE EN ASSEMBLY

Laboratorio No. 2

Andrés Felipe Calvo Ariza

Andrés Guillermo Toloza Guzmán

Ingeniería de sistemas

Universidad de Antioquia

21 de Abril de 2024

Contenido

Decisiones de diseño

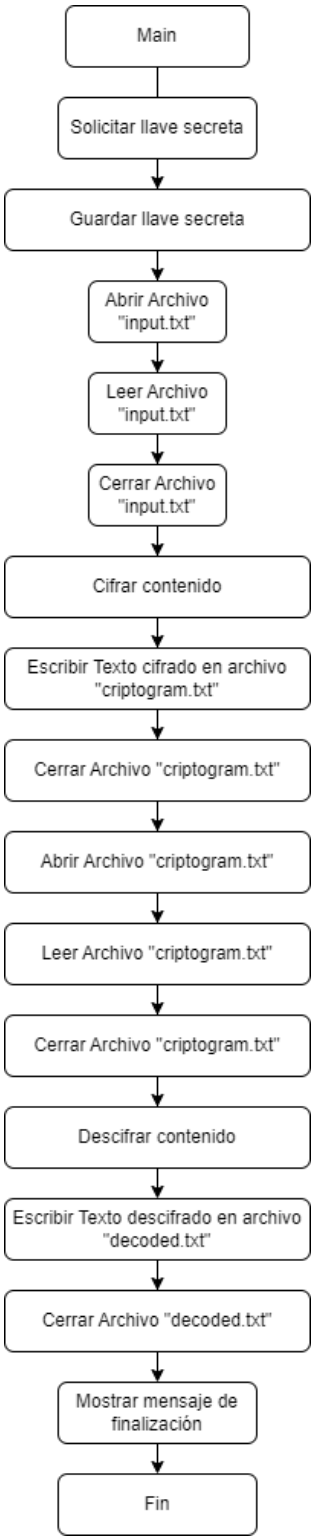
Las decisiones establecidas corresponden a maneras de simplificación y uso eficiente de la memoria para permitir un código mucho más limpio y elegante. Éstas decisiones fueron:

1. **Alfabeto y Rango de Caracteres**:
    - Decisión: Utilizar un alfabeto de letras minúsculas ASCII, limitado entre las posiciones 97 y 122, correspondientes a las letras 'a' a 'z'.
    - Razón: Se opta por restringir el alfabeto a letras minúsculas para simplificar el cifrado y la manipulación de caracteres, centrándose en un conjunto específico de valores ASCII.
2. **Caracteres Especiales**:
    - Decisión: Los caracteres especiales, comprendidos entre las posiciones 32 y 96 de la tabla ASCII, no serán cifrados y se mantendrán sin cambios durante el proceso de cifrado y descifrado.
    - Razón: La exclusión de caracteres especiales simplifica el algoritmo y evita la manipulación innecesaria de estos caracteres, lo que reduce la complejidad y mejora la eficiencia del proceso de cifrado.
3. **Tamaño de la Clave Secreta**:
    - Decisión: Establecer un límite máximo de 20 caracteres para la clave secreta ingresada por el usuario.
    - Razón: Limitar el tamaño de la clave secreta simplifica el proceso de cifrado y descifrado, además de garantizar un manejo adecuado de la memoria y evitar posibles desbordamientos.
4. **Espacio de Buffer para Texto**:
    - Decisión: Asignar un espacio de buffer de 1024 bytes para el texto leído del archivo a cifrar y descifrar.
    - Razón: El tamaño del buffer se selecciona para manejar eficientemente textos de tamaño moderado sin incurrir en un exceso de uso de memoria.
5. **Procedimientos Reutilizables**:
    - Decisión: Implementar procedimientos reutilizables para la apertura, lectura y cierre de archivos, así como el manejo de la bandera para operaciones de cifrado y descifrado.
    - Razón: La reutilización de procedimientos simplifica el desarrollo, reduce la duplicación de código y facilita el mantenimiento del sistema.
6. **Cálculo de Índices**:
    - Decisión: Calcular los índices de cifrado y descifrado restando el valor ASCII del carácter con 97, que representa el valor ASCII de la letra 'a'.

- Razón: Esta técnica simplifica el proceso de cifrado y descifrado al mapear los caracteres del alfabeto a índices numéricos, permitiendo una implementación más eficiente en MIPS.

Al tomar estas decisiones de diseño, se busca crear un algoritmo de cifrado Vigenère en MIPS que sea eficiente, fácil de mantener y capaz de manejar los requisitos específicos del sistema, como el manejo de caracteres, el tamaño de la clave y el manejo de archivos. Estas decisiones están destinadas a garantizar la funcionalidad adecuada del algoritmo y optimizar su rendimiento en el entorno de MIPS.

Descripción del proceso de diseño

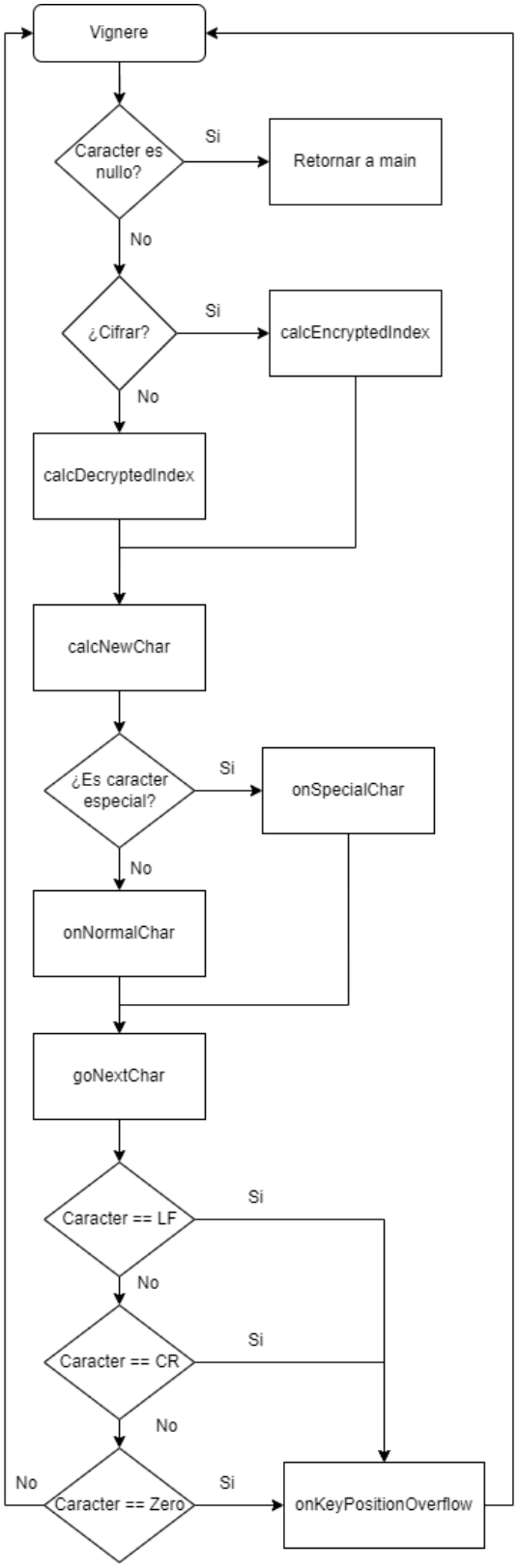


La ejecución del programa consiste en solicitar la llave secreta al usuario con la que se encriptará el archivo input.txt. Luego iteramos sobre cada carácter del archivo, aplicando el cifrado con el algoritmo de vigenere.

Al tener todos los caracteres cifrados guardamos el nuevo texto en el archivo criptogram.txt. Con este archivo “criptogram.txt” lo abrimos, procesamos cada carácter y le aplicamos el descifrado.

```
.text
main:

    jal askEncryptionKeyMsg
    jal receiveUserEncryptionKey

    la $a0,fileInput
    jal openFile
    jal readFile
    jal closeFile


    la $a0, fileWords           #DirecciÃ³n de texto a encriptar
    la $a1, keyInput            #DirecciÃ³n de key
    la $a2, alfabeto            #DirecciÃ³n de alfabeto
    la $a3, newFileWords        #DirecciÃ³n de buffer donde estarÃ¡ el texto encriptado
    li $t0, 0                   #Indice actual sobre el contenido del documento
    li $t1, 0                   #Bandera para encriptar
    li $t2, 0                   #Indice actual sobre el key
    li $t6, 26                  #Cantidad de caracteres en el alfabeto
    jal iterateOverFileContent

    #Argumento para escribir codigo encriptado
    la $a0,encodedOutput        # Obtener direcciÃ³n de archivo encriptado de salida
    jal writeText               # Escribir texto encriptado en el archivo
    jal closeFile               # Cerrar archivo


    #Argumentos para leer archivo encriptado
    la $a0,encodedOutput        # Obtener direcciÃ³n de archivo encriptado de salida
    jal openFile                # Abrir archivo
    jal readFile                # Leer texto encriptado del archivo
    jal closeFile


    la $a0, fileWords           #DirecciÃ³n de texto encriptado
    la $a1, keyInput            #DirecciÃ³n de key
    la $a2, alfabeto            #DirecciÃ³n de alfabeto
    la $a3, newFileWords        #DirecciÃ³n de buffer donde estarÃ¡ el texto desencriptado
    li $t0, 0                   #Indice actual sobre el contenido del documento
    li $t1, 1                   #Bandera para descrifrar
    li $t2, 0                   #Indice actual sobre el key
    li $t6, 26                  #Cantidad de caracteres en el alfabeto
    jal iterateOverFileContent


    la $a0,decodedOutput        #DirecciÃ³n de archivo decodificado
    jal writeText               #Escribir texto decodificado
    jal closeFile               #Cerrar el archivo

    li $v0, 4
    la $a0, alertText           #Mostrar mensaje de finalizaciÃ³n exitosa
    syscall


    li $v0, 10                  #Finalizar el programa
    syscall
```

Implementación de cifrado y descifrado



Un aspecto importante es saber cuándo dejaremos de iterar sobre el contenido, esta condición se cumple cuando el carácter actual contenga el valor de vacío.

Para aplicar el cifrado o descifrado lo condensamos en una sola función, a esta función le pasamos una bandera para indicar si vamos a cifrar o descifrar. Dependiendo del valor de esta bandera ejecutaremos un cálculo de índice diferente.

Posterior a esto verificamos si el carácter de entrada fue especial, en caso de ser especial se inserta en el texto tal cual, en caso contrario se calcula el nuevo carácter usando el nuevo índice del paso anterior.

Finalmente tenemos unas validaciones para la llave secreta, si el carácter secreto corresponde a “LF” o “CR” o “Zero” debemos resetear el apuntador al inicio de la llave secreta. Luego de esto pasamos a la siguiente iteración hasta que se cumpla la condición de quiebre.

```
# Argumentos
# $a0 = Direccion sobre el contenido
# $a1 = Direccion sobre la key
# $a2 = Direccion sobre el alfabeto
# $a3 = Direccion sobre el contenido cifrado
# $t0 = Indice actual sobre el contenido del documento
# $t1 = Bandera para indicar si es encriptar (0) , desencriptar(1)
# $t2 = Indice actual sobre el key
# $t3 = caracter del contenido
# $t4 = caracter de la key
# $t5 = Indice cifrado
# $t6 = Longitud del abecedario
iterateOverFileContent:

    #Si el caracter es nulo retornar
    lb $t3, 0($a0)
    beqz $t3,returnFn



    addi $sp,$sp, -4
    sw $a1, 0($sp)                  #Guardar en pila a1

    add $a1, $a1,$t2                #Mover a1 tomando t2 como un offset
    lb $t4, 0($a1)                  #Cargar caracter de la llave secreta

    #Setear valor de a1 de pila
    lw $a1, 0($sp)                  #Resetear valor de a1 con el valor de la pila
    addi $sp,$sp, 4


    #Vignere al caracter
    #Calculo del indice Cifrado
    beq $t1,0,calcEncryptedIndex
    beq $t1,1,calcDecryptedIndex
```

```
calcEncryptedIndex:
    subi $t5,$t3,194
    add $t5,$t5,$t4
    j calcNewChar

calcDecryptedIndex:
    sub $t5,$t3,$t4
    bltz $t5,convertToPositive
    j calcNewChar
```

```
# Argumentos
# $a2 = Direccion sobre el alfabeto
# $a3 = Direccion sobre el nuevo contenido
# $t5 = Indice cifrado
# $t6 = Longitud del abecedario
calcNewChar:

    #Calcular la division para poder tener el residuo en t5

    div $t5,$t6
    mfhi $t5


    la $a2, alfabeto
    add $a2, $a2,$t5
    lb $t5, 0($a2)


    #Ahora t5 es el caracter cifrado

    la $a3, newFileWords
    add $a3, $a3, $t0
    bge $t3,97,onNormalChar
    bge $t3,32,onSpecialChar
```

```
# Argumentos
# $a3 = Direccion sobre el nuevo contenido
# $t5 = Nuevo caracter
onNormalChar:
    sb $t5, ($a3)
    j goNextChar

# Argumentos
# $a3 = Direccion sobre el nuevo contenido
# $t2 = Indice actual sobre el key
# $t3 = caracter del contenido
onSpecialChar:
    sb $t3, ($a3)
    # Cuando es un caracter especial no debemos avanzar en la llave secreta
    # Con esto reseteamos el avance que se realizara en goNextChar
    subi $t2, $t2,1
    j goNextChar
```

Descripción de los procedimientos

Nombre	Entradas	Salidas	Descripción
openFile	$a0 que representa la dirección del archivo a abrir	$s0, donde se guarda la descripción el archivo	Abrir archivo de acuerdo a la dirección que se encuentre en $a0
readFile	$s0 que representa la dirección de archivo que debe leer	$a1, donde se almacenará la info leída del archivo	Leer contenido del archivo que se encuentre e $s0

iterateOverFileContent	- $a0 = Dirección sobre el contenido - $a1 = Dirección sobre la key - $a2 = Dirección sobre el alfabeto - $a3 = Dirección sobre el contenido cifrado - $t0 = Índice actual sobre el contenido del documento - $t1 = Bandera para indicar si es encriptar (0) , desencriptar(1) - $t2 = Índice actual sobre el key - $t3 = carácter del contenido - $t4 = carácter de la key - $t5 = Índice cifrado $t6 = Longitud del abecedario		Función para iterar sobre cada carácter del contenido, recibe una bandera para cifrar o descifrar el contenido. Se llama dos veces en el main, la primera vez para cifrar el archivo y la segunda para descifrarlo. Esta función se llama hasta que no halla un carácter más por iterar

writeText	$a0 que representa la dirección donde escribir y guardar el archivo	$a2 donde escribe el texto dentro del archivo	Escribir texto almacenado en buffer newFileWords dentro del archivo que se encuentre en la dirección $a0
closeFile	$s0 que representa la dirección de archivo que debe cerrar		Cerrar archivo abierto que se encuentre en la dirección $s0
askEncryptionKeyMsg			Solicitar al usuario que ingrese la clave con la que desea encriptar el mensaje
receiveUserEncryptionKey			Obtener el mensaje escrito por el usuario en consola de la clave para encriptar
returnFn			Se encarga de realizar el retorno al valor que se encuentre en $ra
calcEncryptedIndex	- $t3 = carácter del contenido - $t4 = carácter de la key - $t5 = Índice cifrado	-$t5 Nuevo índice para calcular el módulo en la función de calcNewChar	Descripción: Esta función calcula el índice encriptado con la siguiente operación: t5 = (t3 - 97) + (t4 - 97) 97 representa el valor ascii de nuestra primera letra del abecedario Lo cual se puede resumir a: t5 = t3 + t4 - 194
calcDecryptedIndex	- $t3 = carácter del contenido - $t4 = carácter de la key - $t5 = Índice cifrado	-$t5 Nuevo índice para calcular el módulo en la función de calcNewChar	Esta función calcula el índice desencriptado con la siguiente operación: t5 = (t3 - 97) - (t4 - 97) 97 representa el valor ascii de nuestra primera letra del abecedario Lo cual se puede resumir a: t5 = t3 - t4 En caso que t5 resulte negativo lo convertimos a positivo sumando 26 26 representa la longitud de nuestro abecedario

calcNewChar	- $a2 = Dirección sobre el alfabeto - $a3 = Dirección sobre el nuevo contenido - $t5 = Índice cifrado - $t6 = Longitud del abecedario		Se encarga de calcular el nuevo valor que se debe colocar en el texto cifrado o descifrado teniendo en cuenta el carácter actual de la key y del texto del usuario
onNormalChar	- $a3 = Dirección sobre el nuevo contenido - $t5 = Nuevo carácter		Se encarga agregar el carácter calculado dentro del buffer o texto donde estará el valor cifrado o descifrado correspondiente
onSpecialChar	- $a3 = Dirección sobre el nuevo contenido - $t2 = Índice actual sobre el key - $t3 = carácter del contenido		Realiza la misma función de onNormalChar con la diferencia que este procedimiento sucederá siempre y cuando sea un carácter entre 32 y 96 del código ascii y por ende, no se aumentará la dirección que estamos recorriendo en la key de cifrado
goNextChar	- $a0 = Dirección sobre el contenido - $a1 = Dirección sobre la key - $t0 = Índice actual sobre el contenido del - $t2 = Índice actual sobre el key - $t4 = carácter de la key		Se encarga de avanzar en las direcciones y guardando en pila la dirección sin avanzar para comprobar en caso de que haya overflow de la llave , reiniciar el apuntador o dirección a su posición inicial de la key

onKeyPositionOverflow	- $t2 = Índice actual sobre el key	$t2 = 0	Se encarga de establecer en la posición 0 o inicial el contador $t2 para empezar desde el inicio de la key debido a overflow.
convertToPositive	- $t5 = Índice cifrado - 26 = Longitud del abecedario	-$t5 Índice cifrado positivo	

Observaciones

1. Cuando se quiera calcular el módulo, se debe tener en cuenta que MIPS no tiene una operación para el módulo, por lo que debemos realizar la división y luego buscar el residuo usando mfhi
2. Algunos lenguajes de programación manejan el módulo como una operación que puede retornar números negativos, para nuestro caso era indispensable que el valor fuera positivo, por ello nos aseguramos que el dividendo (El valor de $t5) fuera positivo antes de hacer la división.

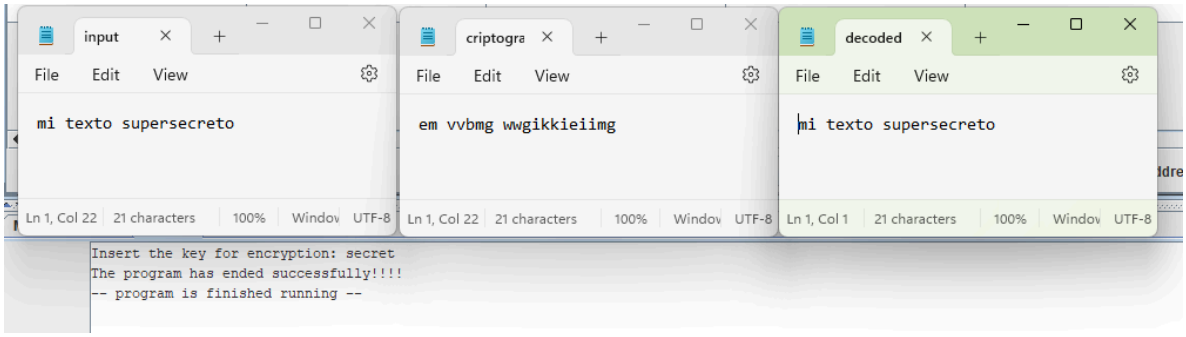
3. Al inicio del diseño, consideramos implementar un bucle while para validar la cantidad de caracteres en cada buffer. Sin embargo, en el diseño final decidimos descartar esta opción debido a que asumimos que al avanzar en la dirección de memoria, si el contador de posición resultaba en un byte de valor 0, indicaría que habíamos completado el recorrido de todo el texto o la clave. Esta decisión simplifica el código y mejora la eficiencia del algoritmo, ya que evita la necesidad de realizar una comparación explícita para determinar la longitud del texto o la clave en cada iteración del bucle.

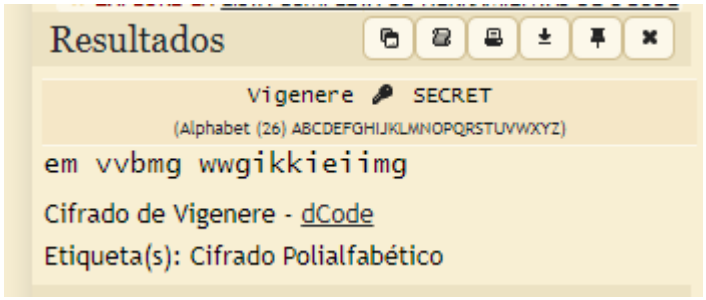
4. En nuestro diseño, empleamos el puntero de pila (SP) y la pila para almacenar temporalmente valores antes de sumarlos como direcciones, evitando así desbordamientos (overflow) de memoria. Además, para optimizar el uso de recursos, no utilizamos todos los registros temporales (T) disponibles en la CPU, asignando solo los necesarios para las operaciones específicas del algoritmo de cifrado Vigenère en MIPS. Esto garantiza una implementación eficiente y robusta del algoritmo.

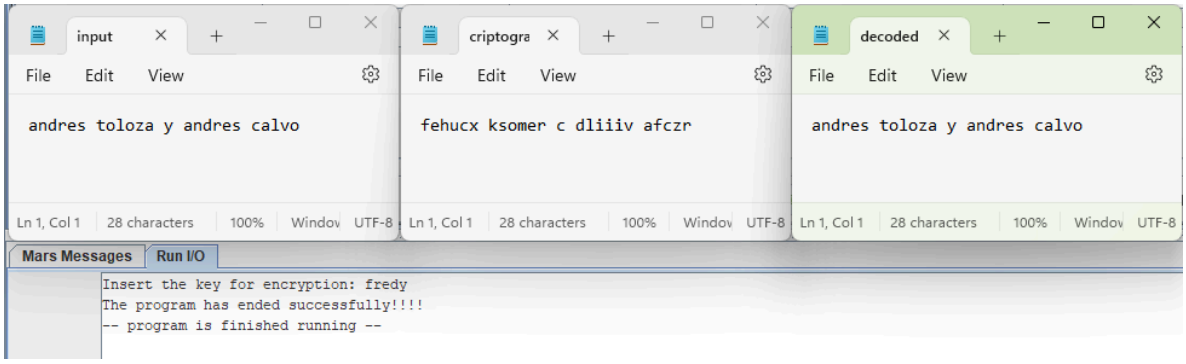
5. Tener muy presente el espacio máximo en memoria definido para los archivos de entrada y salida (1024 bytes). En caso de querer utilizar este algoritmo para archivos mayores a este buffer, se puede aumentar el buffer a 50mb y agregar un procesamiento por bloques, procesando cada 50mb del archivo y concatenarlo con el archivo final. Esto permitirá procesar archivos de cualquier tamaño sin sobrecargar la RAM.

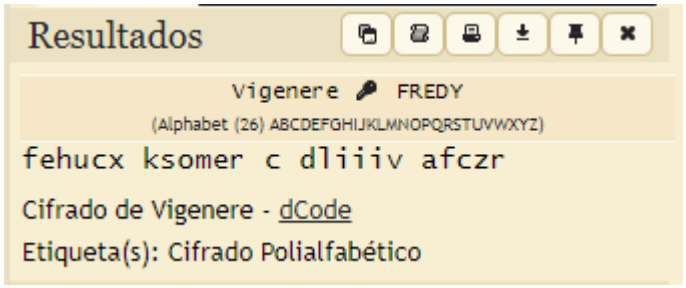
Ejecuciones

Validaciones realizadas con esta webapp: https://www.dcode.fr/cifrado-vigenere









Conclusiones

A lo largo del proceso, hemos adquirido un mayor entendimiento sobre cómo se aplican los principios criptográficos básicos para asegurar la confidencialidad de la información. Además, hemos experimentado con los desafíos únicos que presenta la programación en un lenguaje de bajo nivel como el ensamblador MIPS, lo cual ha fortalecido nuestras habilidades técnicas y de resolución de problemas.

Al realizar este laboratorio, hemos reforzado la importancia de la planificación cuidadosa y la meticulosidad en la codificación, así como la necesidad de realizar pruebas exhaustivas para garantizar la precisión y fiabilidad del código implementado. Además, hemos explorado cómo la optimización del rendimiento puede ser crucial en entornos con recursos limitados, como es el caso de la programación en dispositivos embebidos o sistemas embebidos.

Enlace a vídeo

https://youtu.be/NZKG26bH0AQ