UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Informática II
Semestre 2025 – 2
Desafío 1
Septiembre 14, de 2025

Este desafío simula un escenario de ingeniería inversa, en el que, con información limitada y sin conocer directamente el tipo de algoritmo, ni los parámetros utilizados, se debe reconstruir la información original utilizando lógica, deducción y habilidades de programación.

Objetivos

- Desarrollar la capacidad de solución de problemas en los estudiantes, enfrentándolos a problemáticas cercanas a situaciones reales.
- Evaluar si el estudiante ha adquirido las destrezas y conocimientos fundamentales de la programación en C++, tales como el uso de estructuras de control (secuenciales, iterativas y condicionales), manejo de tipos de datos, operaciones a nivel de bits, punteros, arreglos, memoria dinámica y funciones.

El objetivo principal de esta actividad es poner a prueba sus habilidades en el análisis de problemas y en el dominio del lenguaje C++. Si ha seguido un proceso disciplinado de aprendizaje a lo largo del semestre, esta es una excelente oportunidad para demostrarlo. Podrá proponer una solución efectiva y obtener un resultado satisfactorio. En caso contrario, esta actividad le permitirá identificar sus debilidades y tomar acciones correctivas con miras a prepararse mejor para enfrentar desafíos similares, como los planteados en la sección " Consideraciones Iniciales".

Se recomienda valorar adecuadamente la verdadera complejidad del problema planteado. No se rinda antes de intentarlo ni descarte posibles enfoques sin analizarlos. Descubrirá que, aunque al principio el reto pueda parecer difícil, ya ha desarrollado las herramientas necesarias para enfrentarlo con éxito. Si dedica el tiempo adecuado al análisis del problema, el proceso de codificación será mucho más fluido.

Esperamos que disfrute del desafío propuesto. Le sugerimos leer detenidamente todo el documento antes de comenzar, y asegurarse de entender completamente las instrucciones antes de abordar esta actividad evaluativa.

Fue revisado por los profesores Aníbal Guerra y Augusto Salazar.

I. Consideraciones iniciales

A. Método de compresión RLE (Run-Length Encoding)

Es un algoritmo de compresión sin pérdida muy simple, que funciona mejor cuando los datos tienen muchas repeticiones consecutivas.
La idea es reemplazar una secuencia de símbolos repetidos (un *run*) por:

- El símbolo
- La longitud de la secuencia (cuántas veces aparece seguido).

Por ejemplo, AAAABBBCCDAA, se transforma en 4A3B2C1D2A.

Paso a paso de su funcionamiento:

1. Recorrido secuencial: Se toma el mensaje desde el inicio y se van leyendo los caracteres uno a uno.
2. Contar repeticiones: Cada vez que se detecta que un símbolo se repite consecutivamente, se cuenta cuántas veces aparece seguido.
3. Codificación del par (longitud, símbolo): Una vez que cambia el símbolo, se guarda la pareja (longitud, símbolo) o en notación compacta (longitud + símbolo)
4. Repetir hasta el final: Se continúa hasta procesar toda la secuencia.

Ejemplo detallado

Mensaje original: WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWW

1. Primer bloque: 11 veces W → 11W
2. Luego: 1 vez B → 1B
3. Luego: 12 veces W → 12W
4. Luego: 3 veces B → 3B
5. Luego: 14 veces W → 14W

Resultado comprimido: 11W1B12W3B14W

Mecanismo de descompresión

Para reconstruir el mensaje original:

- Se leen los números (longitud).
- Se repite el símbolo indicado esa cantidad de veces.

Por ejemplo, el texto comprimido: 4A3B2C1D2A, resulta en AAAABBBCCDAA después de la descompresión.

B. Método de compresión Lempel-Ziv (LZ)

Los algoritmos de la familia LZ (como LZ77, LZ78, LZW, etc.) buscan reducir la redundancia de un texto reemplazando repeticiones por referencias más cortas.

Un texto largo suele tener subcadenas repetidas. En el método LZ, en lugar de guardar la misma secuencia varias veces, guardamos un puntero a donde ya apareció antes. Por ejemplo, el texto: ABABABAB, en lugar de guardar ABABABAB, podemos guardar “AB" literal y luego ABAB se representa como "lo mismo que antes".

Compresión LZ78 (versión básica)

A diferencia de versiones previas, en el LZ78 usamos un diccionario dinámico para comprimir. Cada vez que encontramos una subcadena nueva, la añadimos al diccionario con un número. Codificamos como: (índice_diccionario, siguiente_caracter). Donde:

- índice_diccionario: apunta a un prefijo ya visto.
- siguiente_caracter: el nuevo carácter que completa la nueva frase.

Ejemplo de compresión usando LZ78:

Texto: ABAABABA

Proceso:

1. A → (0, 'A'), guardo "A" en el diccionario.
2. B → (0, 'B'), guardo "B".
3. AA → (1, 'A'), guardo "AA".
4. BA → (2, 'A'), guardo "BA".

Resultan pares de índices y letras, en vez del texto plano.

Salida paso para del compresor implementado

Se usan dos bytes para representar el prefijo dado que 256 posiciones en el diccionario pueden no ser suficientes. Por el contrario, con dos bytes se podrían representar hasta 65535 diferentes entradas en el diccionario.

La siguiente tabla muestra un ejemplo de la salida paso a paso para una entrada: ABAB.

Paso	Buffer	Diccionario	Prefijo	Nuevo carácter	Salida (bytes)
1	A	{}	0	'A'	0x00 0x00 'A'
2	B	{"A"}	0	'B'	0x00 0x00 'B'
3	A	{"A","B"}	1	'B'	0x00 0x01 'B'
4	B	{"A","B","AB"}	2	'A'	0x00 0x02 'A'

A continuación, se ponen algunos ejemplos del resultado de la representación usando dos bytes para los prefijos de acuerdo con su valor:

Prefijo	Binario 16 bits	Byte alto	Byte bajo	Hexa byte alto	Hexa byte bajo
1	00000000 00000001	00000000	00000001	0x00	0x01
300	00000001 00101100	00000001	00101100	0x01	0x2C
1025	00000100 00000001	00000100	00000001	0x04	0x01
65535	11111111 11111111	11111111	11111111	0xFF	0xFF

Descompresión LZ78

El descompresor recibe la secuencia de pares (índice, c) y reconstruye el texto original creando el mismo diccionario que usó el compresor.

Algoritmo en alto nivel

1. Inicializa el diccionario vacío.
2. Para cada par (i, c) en la entrada:
    - Si i = 0, el prefijo es vacío.
    - Si i > 0, busca la cadena almacenada en el diccionario en la posición i.
    - Reconstruye la cadena S = prefijo + c.
    - Escribe S en la salida.
    - Inserta S en el diccionario como nueva entrada.

Ejemplo práctico

Texto original: ABAABABA

Compresión LZ78 produce: (0, A), (0, B), (1, A), (2, A), (3, B)

Descompresión

Inicio: diccionario vacío.

1. (0, A) → prefijo vacío + 'A' = A.
    - Salida: A
    - Diccionario: ["A"]
2. (0, B) → "" + 'B' = B.
    - Salida: AB
    - Diccionario: ["A", "B"]
3. (1, A) → "A" + 'A' = AA.
    - Salida: ABAA
    - Diccionario: ["A", "B", "AA"]
4. (2, A) → "B" + 'A' = BA.
    - Salida: ABAABA
    - Diccionario: ["A", "B", "AA", "BA"]
5. (2, A) → "B" + 'A' = BA.
    - Salida: ABAABABA
    - Diccionario: ["A", "B", "AA", "BA"]

Texto reconstruido: ABAABABA

Texto reconstruido: ABAABABA

La implementación que vamos a utilizar se apega a la versión del LZ78 descrita en el video del enlace **(**https://www.youtube.com/watch?v=1CLphtG10yA**)**

C. Encriptación de mensajes usando rotación de bits y operación XOR

Este método combina dos operaciones simples a nivel de bits para transformar un mensaje encriptado:

1. Rotación de bits hacia la izquierda (o derecha) un número fijo de posiciones.
2. Operación XOR con una clave de un byte.

El secreto está en que ambas operaciones son reversibles, de modo que con los parámetros correctos se puede recuperar el mensaje original.

Rotación de bits

Consiste en mover los bits de un byte hacia la izquierda (o derecha). A diferencia de un corrimiento, los bits que salen por un extremo vuelven a entrar por el otro.

Ejemplo (rotación a la izquierda 3 posiciones):

- Original: 01000001
- Rotación: 00001010

Así, el valor se transforma de manera predecible, y puede revertirse aplicando la rotación en el sentido contrario.

Operación XOR

La operación XOR opera bit a bit de la siguiente manera:

- 0 XOR 0 = 0
- 0 XOR 1 = 1
- 1 XOR 0 = 1
- 1 XOR 1 = 0

La propiedad importante es: si aplicas XOR con la misma clave dos veces, recuperas el valor original.

Ejemplo ilustrativo

- Mensaje original: "A" (01000001 en binario)
- Clave: 01011010 (0x5A)
- Rotación: 3 bits a la izquierda

Encriptación:

1. Rotación izquierda 3:
2. 01000001 → 00001010
3. XOR con clave:
4. 00001010 XOR 01011010 = 01010000

Desencriptación:

1. XOR con clave:
2. 01010000 XOR 01011010 = 00001010
3. Rotación derecha 3:
4. 00001010 → 01000001 ("A")

II. Requerimientos de la solución a desarrollar

A la empresa Informa2, llega un cliente que dispone de un mensaje original en texto plano que ha sido sometido a un proceso de compresión y posteriormente a un proceso de encriptación.

1. **Compresión**: El mensaje fue comprimido utilizando RLE o LZ78.
2. **Encriptación**: Una vez comprimido, el mensaje resultante fue encriptado mediante la aplicación de dos operaciones consecutivas:
    - Una rotación a la izquierda de cada byte en una cantidad n de bits (siendo $0 < n < 8$).
    - Una operación XOR con una clave de un solo byte K.

El resultado de estas transformaciones es el mensaje comprimido y encriptado, que es el único disponible.

Sin embargo, se conoce un fragmento del mensaje original en texto plano. Este fragmento servirá como pista para determinar:

- ¿Qué método de compresión se utilizó (RLE o LZ78)?
- El valor de la rotación de bits n.
- El valor de la clave K usada en la operación XOR.

A partir del mensaje comprimido y encriptado, y del fragmento conocido del mensaje original, usted deberá diseñar e implementar un programa que sea capaz de:

1. [20%] Identificar el método de compresión utilizado y los parámetros de encriptación (n y K).
2. [20%] Desencriptar el mensaje aplicando las operaciones inversas en el orden correcto.
3. [60%] Descomprimir el mensaje para obtener el texto original completo.

Para lograr lo anterior, se debe realizar la implementación propia de los algoritmos de compresión y descompresión RLE y LZ78. De igual forma, deberá implementar los métodos de encriptación y desencriptación con rotación de bits + XOR.

Entradas

- Cadena que contiene el mensaje comprimido y encriptado.
- Fragmento conocido del mensaje original (texto plano).}

Nota: El texto incluye únicamente los caracteres de la A a la Z (mayúsculas y minúsculas) sin: tildes, signos de puntuación, espacios en blancos, ni la Ñ. Puede incluir los dígitos del 0 al 9.

Salidas

- Mensaje original completo reconstruido.
- Método y parámetros utilizados en los procesos de compresión y encriptación.

Requisitos mínimos

A continuación, se describen los requisitos que se deben cumplir. El incumplimiento de cualquiera de ellos implica que su nota sea cero.

1. Genere un informe en donde se detalle el desarrollo del proyecto, explique entre otras cosas:
   a. Análisis del problema y consideraciones para la alternativa de solución propuesta.
   b. Esquema donde describa las tareas que usted definió en el desarrollo de los algoritmos.
   c. Algoritmos implementados.
   d. Problemas de desarrollo que afrontó.
   e. Evolución de la solución y consideraciones para tener en cuenta en la implementación.
2. La solución debe ser implementada en lenguaje C++ en el *framework* Qt.
3. La selección del tipo de variables debe hacerse responsable, pensando en uso eficiente de memoria. Las decisiones tomadas deben estar bien argumentadas.
4. No se pueden usar objetos tipo string.
5. La implementación debe incluir el uso de punteros, arreglos y memoria dinámica.
6. En la implementación no se pueden usar estructuras ni STL.
7. Se debe crear un repositorio público en el cual se van a poder cargar todos los archivos relacionados a la solución planteada por usted (informe, código fuente y otros anexos).
8. Una vez cumplida la fecha de entrega no se podrá hacer modificación alguna al repositorio.
9. Se deben hacer *commits* de forma regular (repartidos durante el periodo de tiempo propuesto para las entregas) de tal forma que se evidencie la evolución de la propuesta de solución y su implementación.
10. Se debe adjuntar un enlace de *youtube* a un video que debe incluir lo siguiente:
    a. Presentación de la solución planteada. Análisis realizado y explicación de la arquitectura del sistema (3 minutos máximo).
    b. **Demostración de funcionamiento del sistema**. Explicar cómo funciona: ejemplos demostrativos (3 minutos máximo).
    c. Explicación del código fuente. Tenga en cuenta que debe justificar la elección de las variables y estructuras de control usados. Por qué eligió uno u otro tipo de variable o estructura de control en cada caso particular y que ventaja ofrecen estos en comparación de otras que también podrían haber sido usados (5 minutos máximo).
    d. La duración total del video no debe exceder 11 minutos ni ser inferior a 5 minutos.
    e. Asegúrese que el video tenga buen sonido y que se puede visualizar con resolución suficiente para apreciar bien los componentes presentados.
    f. El video no debe ser generado por alguna herramienta de IA.
    g. En la elaboración del video deben participar todos los miembros del equipo. La colaboración de cada uno debe ser explícita (a través del audio descriptivo y/o video donde aparezcan).
11. El plazo de entrega se divide en dos momentos:
    a. El día 19 de septiembre (11:59 pm), para adjuntar la evidencia del proceso de análisis y diseño de la solución. (Informe preliminar: Contextualización, Análisis, Diseño)
    b. El día 26 de septiembre (11:59 pm), para adjuntar la evidencia del proceso de implementación.
12. En Ude@ se deben adjuntar dos enlaces: uno al repositorio y otro al video, nada más.
13. Para la evaluación del desafío se realizará una sustentación oral en un horario concertado con el profesor. La asistencia a la sustentación es obligatoria.