udp

Laboratorio 1

Protegiendo Mensajes en una Red Segura con Cifrado Vigenère

Autores: Ailyn Bravo, Benjamín Herrera
Fecha: Abril 2025

1. Implementación

El núcleo del proyecto es la clase BigVigenere, cuyo constructor principal (public BigVigenere()) solicita una key al usuario y la convierte en un arreglo de índices según la posición de cada carácter en una cadena base. Esta cadena contiene 64 caracteres: letras mayúsculas, minúsculas, dígitos del 0 al 9 y los caracteres ñ y Ñ. La clave se almacena como un arreglo de enteros para facilitar el desplazamiento en la matriz alfabética.

La matriz alphabet, de 64x64, se construye en el método buildAlphabet(). Esta matriz es una tabla Vigenère extendida que define los desplazamientos para cada combinación de caracteres posibles, manteniendo la lógica de rotación modular con (fila + columna) 'modulo' 64.

El segundo constructor (BigVigenere(String numericKey)) permite la creación directa de un objeto con una clave ya conocida, útil para pruebas automatizadas.

2. Métodos

1. public String encrypt(String message): Este método cifra un mensaje recorriéndolo carácter por carácter. Para cada carácter, encuentra su índice en chars y lo desplaza utilizando la clave. Si encuentra un espacio, lo conserva. Si el carácter no pertenece al alfabeto, lo deja intacto.
2. public String decrypt(String encryptedMessage): Recorre la frase cifrada e identifica en qué fila se encuentra el carácter cifrado, buscando en la columna correspondiente al valor de la clave. Una vez encontrado, se recupera el carácter original desde chars.
3. public void reEncrypt(): Desencripta una frase dada y luego permite reencriptarla con una nueva clave ingresada por el usuario. Utiliza los métodos decrypt y encrypt.
4. public char search(int position): Busca un carácter en la matriz alphabet recorriéndola fila por fila hasta alcanzar la posición indicada. Tiene una complejidad de tiempo O(n), ya que la búsqueda es lineal.
5. public char optimalSearch(int position): Realiza un acceso directo a la posición de la matriz utilizando la fórmula: fila = position / 64 y columna = position 'modulo' 64. Esto permite una búsqueda de tiempo constante O(1), mucho más eficiente que search.

3. Experimentación y Resultados

Al realizar el experimento de *encrypt* y *decrypt* de la forma correcta, podemos ver cómo en *encrypt* colocamos una clave y una frase, y esta nos retorna letras al azar, que es la frase encriptada. Por otro lado, tenemos *decrypt*, en el cual llamamos a la función, y nos pide la frase encriptada y la clave para retornarnos la frase inicial.

```
Cifrado Vigenere
A) Encrypt
B) Decrypt
C) reEncrypt
D) Buscar caracter
E) Buscar caracter optimo
F) Salir
Opcion: a
Ingresa la key: 23dj49567co
Introduzca una frase: Hola Mundo
Frase encriptada: xfoj Jnhñq
Process finished with exit code 0
```

Figura 1: Resultado del proceso de encriptado

```
Cifrado Vigenere
A) Encrypt
B) Decrypt
C) reEncrypt
D) Buscar caracter
E) Buscar caracter optimo
F) Salir
Opcion: b
Introduzca la frase: xfoj Jnhñq
Ingresa la key: 23dj49567co
Frase desencriptada: Hola Mundo
Process finished with exit code 0
```

Figura 2: Resultado del proceso de desencriptado

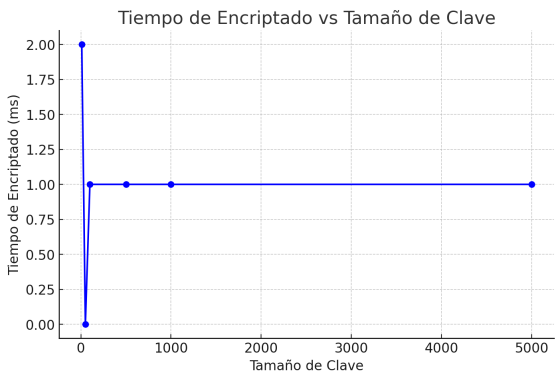

Figura 3: Gráfico de tiempo de encriptado

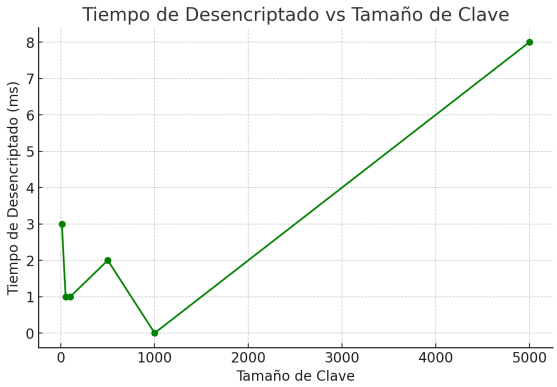

Figura 4: Gráfico de tiempo de desencriptado

Con estos graficos podemos observar que cuando se encripta, entre mayor sera el tamaño, menos es el tiempo de ejecucion, mientras tanto cuando se cifra, mientras mayor sea el tamaño, mayor sera el tiempo.

Ambos algoritmos Encrypt y Decrypt, tienen un Big(O)n, en mensajes es Big tamaño O(1), la busqueda de char, tiene O(n) El algoritmo es eficiente, incluso con claves muy grandes, por el uso de operaciones simples como módulo y acceso directo en arreglos. Esto lo hace adecuado para aplicaciones donde se requiere velocidad y seguridad en el cifrado de mensajes sin afectar el rendimiento.

4. Conclusiones

Durante el desarrollo del laboratorio, una de las principales dificultades fue construir correctamente la matriz alfabética. Esta matriz incluye letras mayúsculas, minúsculas y números. Esto llevó a la necesidad de ajustar

el conjunto de caracteres base y asegurarse de que la matriz resultante tuviera un tamaño uniforme de 64 por 64, para que el cifrado funcionara de manera correcta. Para lograr esto, implementamos un método que construye la matriz a partir de rotaciones modulares, garantizando que no haya repeticiones ni errores de indexación.

La optimización en la búsqueda de posiciones se logró gracias al método optimalSearch, que utiliza operaciones aritméticas directas para acceder a cualquier posición de la matriz sin tener que recorrerla completamente, a diferencia del método search. Esto representa una mejora notable en términos de eficiencia, ya que disminiye la complejidad de la búsqueda de O(n) a O(1), permitiendo respuestas inmediatas sin importar la posición solicitada.

Sobre el tamaño de la clave utilizar, se observó que las claves más largas dan un mayor nivel de seguridad, ya que disminuyen la repetición del patrón de cifrado. Sin embargo, esto también conlleva un mayor consumo. Por lo tanto, se recomienda optar por claves de tamaño media a larga que contengan una combinación de letras, números y caracteres especiales presentes en el conjunto base del cifrado.

Como recomendaciones para crear la clave, se sugiere evitar claves demasiado cortas o repetitivas, asegurarse de que todos los caracteres estén incluidos en el conjunto definido y si es posible, generar claves al azar o desordenadas para aumentar la fuerza del sistema frente a posibles intentos de descifrado.