Universidad Diego Portales
Escuela de Informática &
Telecomunicaciones

ESTRUCTURAS DE DATOS &
ANÁLISIS DE ALGORITMOS

Laboratorio de Cifrado Vigenère con Clave Extendida y Alfabeto Matricial

---

Autores: Kevin Bello
Sebastian Carrasco

Profesor:
Marcos Fantoval

9 de abril de 2025

Índice

1. Introducción

El presente informe detalla el proceso de diseño e implementación de una clase BigVigenere en Java, la cual permitirá cifrar y descifrar mensajes utilizando esta aproximación innovadora del cifrado de Vigenère con claves de gran tamaño y una representación matricial del alfabeto. A lo largo de este documento, se expondrán los detalles de la construcción de la clase, la lógica implementada en sus métodos, los resultados obtenidos en las pruebas experimentales y las conclusiones derivadas de este ejercicio práctico.

2. Cifrado Vigenere

El cifrado Vigenère es un método clásico de cifrado polialfabético que utiliza una palabra clave para cifrar y descifrar mensajes. A diferencia de los cifrados monoalfabéticos donde cada letra del alfabeto se sustituye siempre por la misma letra, Vigenère emplea una serie de cifrados César diferentes basados en las letras de la palabra clave.

En esencia, cada letra de la clave se utiliza para desplazar una letra del mensaje original en el alfabeto. Si la clave es más corta que el mensaje, se repite. Esto hace que el cifrado Vigenère sea más resistente al criptoanálisis de frecuencia simple en comparación con los cifrados más simples.

En el contexto de este laboratorio, la implementación se complejiza al utilizar una clave numérica de gran tamaño, almacenada en un arreglo, y un alfabeto representado en una matriz bidimensional. Esto busca manejar claves extensas para mayor seguridad en la transmisión de información sensible.

Algunas ventajas de este cifrado en nuestro contexto son:

- Mayor seguridad: Claves muy largas (números grandes en un arreglo) hacen más difícil descifrar el mensaje mediante análisis de frecuencia.
- Manejo de claves extensas: Permite usar claves que no caben en variables normales.
- Alfabeto organizado: La matriz facilita las operaciones de cifrado y descifrado.

Para mas informacion sobre el cifrado Vigenere, como imagenes, o codigo de ejemplo en C, puede visitar la siguiente pagina : Cifrado Vigenere en Wikipedia

3. Metodologia

3.1. Implementacion del Constructor, public BigVigenere

El constructor de la clase BigVigenere es responsable de inicializar los atributos clave para el funcionamiento del cifrado: la clave numérica **key** y la matriz de representación del alfabeto **alphabet**. A continuación, se detalla el proceso de construcción de cada uno de estos atributos.

1. Construcción del atributo **key**:
   El atributo key, de tipo int[ ], está diseñado para almacenar la clave numérica de gran tamaño proporcionada por el usuario. El proceso de construcción de este atributo se realiza en los siguientes pasos:

   - Obtención de la entrada del usuario: Se instancia un objeto de la clase *Scanner* para permitir la lectura de datos desde la entrada estándar del sistema **System.in**.
   - Solicitud de la clave: Se muestra en la consola un mensaje Ïngresar clave: "para solicitar al usuario que introduzca la clave numérica deseada.
   - Lectura de la clave como cadena: La entrada del usuario, que se espera sea una secuencia de dígitos, se lee utilizando el método nextLine() del objeto Scanner y se almacena en una variable de tipo String llamada claveNumerica.
   - Creación del arreglo key: Se crea un nuevo arreglo de enteros (int[]) llamado key. La longitud de este arreglo se determina dinámicamente a partir de la longitud de la cadena claveNumerica ingresada por el usuario.

2. Construcción de la matriz **alphabet**.

   - El atributo **alphabet**, de tipo **char[][]**, es una matriz bidimensional de 64x64 que representa el alfabeto extendido utilizado en el cifrado Vigenère. Su construcción se realiza en los siguientes pasos:
   - Alfabeto Base (Chars): Se define un arreglo con los 64 caracteres alfanuméricos (a-z, ñ, A-Z, Ñ, 0-9) en un orden específico.
   - Inicialización: Se crea una matriz vacía de 64x64 para almacenar el alfabeto desplazado.
   - Generación Desplazada: Se recorre cada celda de la matriz. El carácter asignado a cada celda [i][j] se toma del alfabeto base Chars en una posición calculada como (i + j) % 64. Esto provoca que cada fila de la matriz sea una versión del alfabeto base rotada una posición a la izquierda respecto a la fila anterior, creando el alfabeto Vigenère necesario para el cifrado.

3.1.1. Codigo De el Constructor:

```
public BigVigenere() {

    ///aca le pedimoss al usuario que ingrese una clave
       numerica para despues meterla en un arr de enteros
    ///para despues usarla y moverse por el alfabeto largo
    Scanner sc = new Scanner(System.in);
    System.out.print("Ingresar clave numerica: ");
    String claveNumerica = sc.nextLine();
    this.key = new int[claveNumerica.length()];
    for (int i = 0; i < claveNumerica.length(); i++) {
        key[i] = Character.getNumericValue(claveNumerica.
           charAt(i));

    }
    this.Chars = new char[]{
            'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
            'k', 'l', 'm', 'n', ' ', 'o', 'p', 'q', 'r', 's'
               ,
            't', 'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C',
            'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
            'N', ' ', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V'
               ,
            'W', 'X', 'Y', 'Z', '0', '1', '2', '3', '4', '5',
            '6', '7', '8', '9'
    };
    this.alphabet=new char[this.Chars.length][this.Chars.
       length];
    ///aca se ajusta la matriz para que en cada fila
       horizontal, se vayan corriendo los Chars +1
    for (int i = 0; i < this.Chars.length; i++) {
        for (int j = 0; j < this.Chars.length; j++) {
            alphabet[i][j] = this.Chars[(i + j) % this.Chars.
               length];
        }
    }
}
```

Listing 1: Constructor BigVigenere()

4. Metodos

4.1. public String encrypt(String message)

El método encrypt cifra un mensaje carácter por carácter, simulando el cifrado Vigenère con una clave numérica y un alfabeto matricial:

- Inicialización: Se crea un StringBuilder llamado MensajeCifrado. Esta estructura permite construir la cadena cifrada de manera eficiente al evitar la creación de múltiples objetos String en cada iteración.
- Iteración del Mensaje: El código recorre cada carácter del message de entrada utilizando un bucle for. La variable i rastrea la posición del carácter actual.
- Obtención del Carácter: En cada paso, se extrae el carácter actual del mensaje (message.charAt(i)) y se almacena en la variable c.
- Búsqueda en el Alfabeto Base: Se inicializa una variable coord en -1. Luego, se realiza una búsqueda lineal a través del alfabeto base (this.Chars) para encontrar la posición del carácter c. Si se encuentra, su índice se guarda en coord y el bucle de búsqueda se interrumpe.

1. Cifrado (si el carácter está en el alfabeto):
    - Si coord no es -1 (el carácter fue encontrado en el alfabeto): Se determina la letra de la clave a utilizar para este carácter del mensaje. Esto se hace accediendo al arreglo this.key en la posición i % this.key.length. El uso del operador módulo (%) asegura que la clave se repita si es más corta que el mensaje.
    - Se realiza la sustitución Vigenère. El carácter cifrado se obtiene de la matriz this.alphabet en la fila correspondiente al valor de la clave (clave) y la columna correspondiente a la posición del carácter del mensaje en el alfabeto base (coord).
    - El carácter cifrado resultante se añade al StringBuilder MensajeCifrado.
2. Mantenimiento de Caracteres No Encontrados:
    - El carácter cifrado resultante se añade al StringBuilder MensajeCifrado.
    - Si coord es -1 (el carácter del mensaje no está en el alfabeto base), este carácter se añade directamente al MensajeCifrado sin cifrar. Esto permite que el método maneje mensajes que contengan caracteres fuera del alfabeto definido.
    - Retorno del Mensaje Cifrado: Finalmente, después de procesar todos los caracteres del mensaje, el contenido del StringBuilder MensajeCifrado se convierte a un String utilizando toString() y se devuelve como el resultado del cifrado.

4.1.1. Codigo del Metodo Encrypt:

```
1  public String encrypt(String message) {
2          /// trabajamos con la clase StringBuilder para
               modificar el string mas facil
3
4          StringBuilder MensajeCifrado = new StringBuilder();
5          for (int i = 0; i < message.length(); i++) {
6              char c = message.charAt(i);
7      /// vamos a recorrer el alfabeto con el caracter "c"
          entonces inicializamos las coordenadas en -1-1 para
          actualizar luego
8      ///(con este tip podemos ver mas facilmente un error si
          esque llega a haber alguno en la busqueda osea coords
          -1-1=error)
9          int coord=-1;
10     for (int j = 0; j < this.Chars.length; j++) {
11         if (this.Chars[j] == c) {
12             coord=j;
13             break;
14         }
15      }
16         if (coord != -1) {
17            ///encontrar la posiicion de la  clave
18          int clave=this.key[i%this.key.length];
19         char CharCifrado = this.alphabet[clave][coord];
20         MensajeCifrado.append(CharCifrado);
21      }
22             else { //si el caracter no se encuentra, solo
                   incluirlo tal cual
23                 MensajeCifrado.append(c);
24      }
25         }
26         return MensajeCifrado.toString();
27     }
```

Listing 2: Metodo encrypt

4.2. public String decrypt(String encryptedMessage)

El método decrypt toma un encryptedMessage y busca revertir el proceso de cifrado Vigenère utilizando la misma clave y la matriz de alfabeto:

- Preparación: Se inicializa un StringBuilder llamado MensajeDecifrado. Esta herramienta permite construir eficientemente el mensaje original a medida que se descifran los caracteres.
- Iteración del Mensaje Cifrado: Se recorre cada carácter del encryptedMessage utilizando un bucle for. La variable i indica la posición del carácter cifrado actual.
- Obtención de la Clave Correspondiente: Para cada carácter cifrado, se determina el valor de la clave que se utilizó para cifrarlo. Esto se hace accediendo al arreglo this.key en la posición i % this.key.length. El operador módulo asegura que la clave se aplique de forma repetitiva si el mensaje cifrado es más largo que la clave.
- Búsqueda en la Fila del Alfabeto Cifrado: Se realiza una búsqueda dentro de la fila de la matriz this.alphabet que corresponde al valor de la clave actual (this.alphabet[clave]). El objetivo es encontrar la posición del carácter cifrado (c). Se utiliza un bucle for que itera a través de las columnas (j) de esta fila.

  1. Identificación del Carácter Original:

     - Si el carácter cifrado c se encuentra en la fila de la matriz (this.alphabet[clave][j] == c), el índice de la columna j representa la posición del carácter original dentro del alfabeto base (this.Chars). Por lo tanto, el carácter original es this.Chars[j], y se añade al MensajeDecifrado. Después de encontrar el carácter original, se interrumpe la búsqueda en la fila (break).

  2. Manejo de Caracteres No Cifrados:

     - Si después de recorrer toda la fila de la matriz correspondiente a la clave, no se encuentra el carácter cifrado (coordAnterior permanece en su valor inicial de -1), se asume que este carácter no fue cifrado (quizás no estaba en el alfabeto base original). En este caso, el carácter cifrado original c se añade directamente al MensajeDecifrado.

     - Retorno del Mensaje Decifrado: Una vez que se han procesado todos los caracteres del mensaje cifrado, el contenido del StringBuilder MensajeDecifrado se convierte a un String y se devuelve como el mensaje original descifrado

4.2.1. Codigo del Metodo Decrypt:

```
public String decrypt(String encryptedMessage) {
        StringBuilder MensajeDecifrado = new StringBuilder();
        for (int i = 0; i < encryptedMessage.length(); i++) {
            char c = encryptedMessage.charAt(i);
            int coordAnterior=-1;
            int clave=this.key[i%this.key.length];

            for (int j = 0; j < this.Chars.length; j++) {
                if (this.alphabet[clave][j] == c) {
                    coordAnterior=j;
                    break;
                }
            }
                if (coordAnterior != -1) {
                char cAnterior=this.Chars[coordAnterior];
                MensajeDecifrado.append(cAnterior);
                }
                else {
                    MensajeDecifrado.append(c);
                }
            }
        return MensajeDecifrado.toString();
    }
```

Listing 3: Metodo Decrypt

4.3. public void reEncrypt()

- El método reEncrypt() permite volver a cifrar un mensaje. Primero, solicita al usuario un mensaje ya cifrado y lo descifra utilizando la clave y el alfabeto actuales de la instancia BigVigenere. El resultado de este descifrado se guarda para el siguiente paso.
- Luego, el método pide al usuario que ingrese una nueva clave numérica. Con esta nueva clave, se crea una nueva instancia de la clase BigVigenere. Es importante recordar que el constructor espera un arreglo de enteros como clave, por lo que la cadena ingresada debe ser procesada. Esta nueva instancia tendrá su propia clave y matriz de alfabeto, independientes de la original.
- Finalmente, el mensaje que fue descifrado en el primer paso se encripta nuevamente. Esta encriptación se realiza utilizando el método encrypt() de la nueva instancia de BigVigenere, empleando la nueva clave y el nuevo alfabeto. El resultado de esta segunda encriptación se muestra al usuario en la consola.

4.3.1. Codigo del Metodo reEncrypt:

```
1  public void reEncrypt() {
2          /// reEcntrypt por lo que nos piden es mas sencillo
               ya que es llamar a metodos que
3          /// ya implementamos mas arriba en un orden en
               especifico
4          Scanner sc = new Scanner(System.in);
5  //pedimos un mensaje encriptado
6          System.out.print("ingresa mensaje cifrado: ");
7          String MensajeCifrado = sc.nextLine();
8  //se desencripta
9          String MensajeDecifrado = decrypt(MensajeCifrado);
10 //pedimos una clave nueva para encriptarlo
11         System.out.println("ingresar nueva clave numErica: ")
               ;
12         String Clave = sc.nextLine();
13 //generamos una instancia para encriptar
14         BigVigenere cifrar= new BigVigenere(Clave);
15 //se encripta con la nueva clave
16         String NuevoMnsjCifrado = cifrar.encrypt(
               MensajeDecifrado);
17         //se imprime mensaje encriptado con la nueva clave
18         System.out.println(NuevoMnsjCifrado);
19
20     }
```

Listing 4: Metodo reEncrypt

4.4. public char search(int position)

- El método search(int position) tiene como objetivo traducir una posición lineal dada (position) a las coordenadas bidimensionales correspondientes dentro de la matriz alphabet y luego devolver el carácter que se encuentra en esa ubicación, Para lograr esto, el método primero determina las dimensiones de la matriz alphabet. Obtiene el número total de filas y el número de columnas (asumiendo que todas las filas tienen la misma longitud).
- A continuación, para calcular el índice de la fila (Filacalculada), se realiza una división entera de la position proporcionada por el número de columnas. El resultado de esta división entera indica cuántas filas completas se han -ecorrido.[en] la representación lineal de la matriz.
- Para calcular el índice de la columna (Columnacalculada), se utiliza el operador módulo (%) entre la position y el número de columnas. El residuo de esta operación indica la posición dentro de la fila a la que corresponde la position lineal.
- Finalmente, con los índices de fila (Filacalculada) y columna (Columnacalculada) calculados, el método accede a la matriz alphabet en esa posición específica y devuelve el carácter que se encuentra allí. Este método esencialmente proporciona una forma de acceder a la matriz bidimensional como si sus elementos estuvieran dispuestos en una única secuencia lineal.

4.4.1. Codigo del Metodo search:

```
1 public char search(int position) {
2         int filas = this.alphabet.length;
3         int columnas = alphabet[0].length;
4         int Filacalculada= position/columnas;
5         int Columnacalculada= position%columnas;
6         return this.alphabet[Filacalculada][Columnacalculada
             ];
7
8     }
```

Listing 5: Metodo search

4.5. public char optimalSearch(int position)

Inicialmente, se planeó implementar un método adicional denominado **optimal-Search(int position)** con el objetivo de optimizar la búsqueda de un carácter dentro de la matriz,sin embargo, no se identificó una estrategia de búsqueda significativamente mejor, Por lo tanto, la funcionalidad de acceso a la matriz alphabet basada en una posición lineal se implementó directamente en el método **search(int position)**.

5. Experimentación y Resultados.

Para esta sección se realizaron los experimentos solicitados con ayuda de la libreria Random para generar los diferentes parametros.

Primeramente se utilizo una funcion Main modificada para generar aleatoriamente un mismo mensaje de almenos 10.0000 caracteres el cual se puso a prueba con los metodos requeridos. De la misma manera la funcion main genera aleatoriamente las diferentes claves de diferentes medidas depende cual sea el caso.

```
public static void main(String[] args) throws IOException {
        int[] keySizes = {10, 50, 100, 500, 1000, 5000};
        int messageLength = 10000;
        String characters = "
           abcdefghijklmn opqrstuvwxyzABCDEFGHIJKLMN OPQRST
        UVWXYZ0123456789";
    // Generar mensaje base de 10.000 caracteres aleatorios
        Random rand = new Random();
        StringBuilder messageBuilder = new StringBuilder();
        for (int i = 0; i < messageLength; i++) {
            messageBuilder.append(characters.charAt(rand.
               nextInt(characters.length())));
        }
        String mensaje = messageBuilder.toString();

        for (int size : keySizes) {
        // Generar clave num rica aleatoria del tama o dado
            StringBuilder claveNumerica = new StringBuilder()
               ;
            for (int i = 0; i < size; i++) {
                claveNumerica.append(rand.nextInt(10)); //
                   d gitos del 0 al 9
            }
            BigVigenere bigVigenere = new BigVigenere(
               claveNumerica.toString());

        //medicion de tiempo
            long startEncrypt = System.nanoTime();
            String mensajeCifrado = bigVigenere.encrypt(
               mensaje);
            long endEncrypt = System.nanoTime();
            long tiempoCifrado = endEncrypt - startEncrypt
               ;....//se repite el proceso de medicion de
               tiempo
```

Listing 6: Main de experimentacion

Los valores del tiempo de ejecucion (en nanosegundos) para los metodos de cifrar y decifrar con claves de diferente tamaño y mensaje de almenos 10.000 caracteres son los siguientes:

```
Clave de largo 10 => Cifrado: 2292200 ns | Descifrado: 3085000 ns
Clave de largo 50 => Cifrado: 495300 ns | Descifrado: 472100 ns
Clave de largo 100 => Cifrado: 411000 ns | Descifrado: 1291100 ns
Clave de largo 500 => Cifrado: 275600 ns | Descifrado: 1086200 ns
Clave de largo 1000 => Cifrado: 266400 ns | Descifrado: 948000 ns
Clave de largo 5000 => Cifrado: 279800 ns | Descifrado: 886000 ns
```

Figura 1: Resultados de tiempo cifrado y decifrado.

Con la recopilacion de estos datos podemos llevarlo al siguiente grafico:

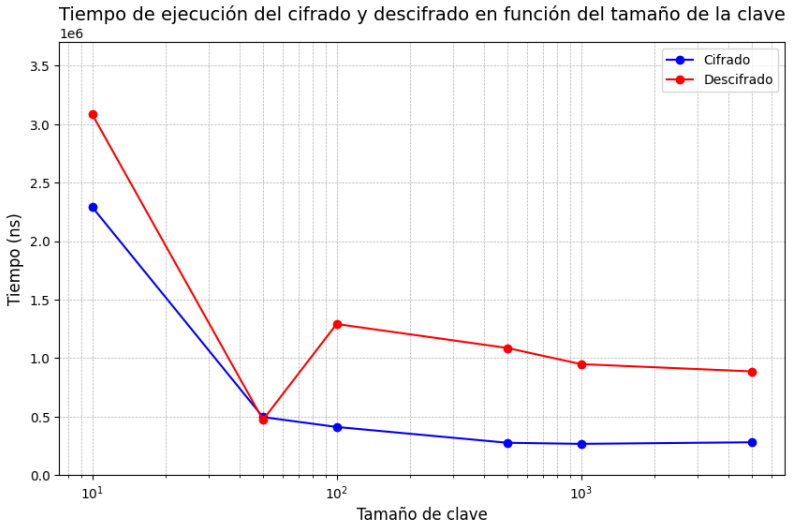

Figura 2: Nanosegundos en relacion a tamaño de la clave

Como se puede apreciar, antintuitivamente el programa parece trabajar mas rapido a mayor cantidad de datos de entrada. Esto puede ser que a medida que aumentan los datos, el comportamiento de la CPU y la forma en que se accede a la memoria pueden hacer que el tiempo de ejecución disminuya, pero esto es más una coincidencia o un efecto de optimización de la caché que una mejora real en el algoritmo.

Aunque el tamaño de la clave afecta el proceso de cifrado (al repetir la clave a lo largo del mensaje), en términos de eficiencia, el impacto no es significativo. El programa sigue siendo bastante eficiente, ya que la complejidad del algoritmo de cifrado no depende directamente del tamaño de la clave, sino del tamaño del mensaje. Es decir, mientras más largo sea el mensaje, mayor será el tiempo de cifrado, pero el tamaño

de la clave solo tiene un efecto indirecto en la cantidad de veces que se repite el ciclo de búsqueda en la matriz.

La complejidad temporal del cifrado y descifrado es O(n), donde "n.es el tamaño del mensaje a cifrar/descifrar. Cada carácter del mensaje requiere buscar su posición en la matriz de caracteres y aplicar la clave correspondiente, lo que resulta en un tiempo de ejecución lineal respecto al tamaño del mensaje. Aunque la longitud de la clave afecta el número de ciclos (se repite para cada carácter), esta variación no cambia la complejidad fundamental, ya que el tiempo sigue siendo proporcional al tamaño del mensaje y no al tamaño de la clave.

6. Conclusión

Durante el desarrollo del laboratorio, una de las dificultades principales fue la implementación de los dos métodos de búsqueda: search y optimalSearch. Aunque se intentó mejorar la eficiencia del segundo método, no encontramos una manera significativa de optimizarlo respecto al primero, por lo que ambos métodos resultaron ser prácticamente iguales. Esto nos llevó a utilizar solo el segundo método para la búsqueda de posiciones.

En el diseño de la clave para el algoritmo de Vigenère, es importante considerar tanto la seguridad como la eficiencia. En nuestras pruebas, se observó que el tiempo de ejecución disminuía a medida que aumentaba el tamaño de la clave, lo cual fue un comportamiento inesperado. Esto se puede atribuir a la forma en que el algoritmo maneja la búsqueda y el cifrado, donde un tamaño mayor de clave, al ser más çomplejo", puede mejorar la distribución de los índices en la matriz, reduciendo el tiempo de búsqueda y aumentando la eficiencia en ciertos casos.

Sin embargo, es importante señalar que este comportamiento no es común en todos los algoritmos y puede depender de factores específicos del entorno de ejecución y de la implementación del algoritmo. En este contexto, la clave no debe ser excesivamente grande para evitar tiempos de procesamiento innecesarios, pero sí debe ser lo suficientemente larga para asegurar una distribución eficaz de los caracteres en el cifrado.

7. Anexos

Para acceder al código completo de este proyecto, puede visitar el siguiente enlace:
`https://github.com/Kevinb1103/Lab1-EDA/blob/Codigo/src/BigVigenere.java`