Universidad Diego Portales
Escuela de Informática &
Telecomunicaciones

Estructuras de Datos & Análisis de Algoritmos

Laboratorio n°1

Autores:
Allen Mora
Gabriel Varas

Profesor:
Marcos Fantóval

07/04/2025

Índice

1. Implementación

En este laboratorio se realizará la creación e implementación de un algoritmo para proteger mensajes dentro de una red segura con cifrado Vigenere, esto para una empresa de telecomunicaciones la cual busca que su información confidencial sea protegida debido a la sensibilidad de su información.

Esto se logra mediante la implementación de un código en Java el cual contiene el método BigVigenere,la cual posee las siguientes atributos:

- Key o clave: 64
- Matriz 64x64

```
public static class BigVignere {
    int[] key;
    char[][] alphabet;
```

Figura 1: Classe BigVigenere

2. Métodos

Dentro de la clase BigVigenere se utilizaron 11 métodos distintos cada uno con un propósito distinto para la correcta implementación de la encriptación del mensaje, estos métodos fueron los siguientes:

- **void generateAlphabet()**: Como su nombre lo indica, este método se encarga de generar la matriz de Vigenère, incluyendo caracteres desde a-z, A-Z y 0-9. A través de esta matriz se encriptarán los mensajes.

```
private void generateAlphabet() {  2 usages
    alphabet = new char[chars.length()][chars.length()];
    for (int i = 0; i < chars.length(); i++) {
        for (int j = 0; j < chars.length(); j++) {
            int index = i + j;
            if (index >= chars.length()) {
                index -= chars.length();
            }
            alphabet[i][j] = chars.charAt(index);
        }
    }
}
```

Figura 2: Método gerateAlphabet()

- **void setKey(String key)**: Este método asigna la llave de encriptación al arreglo de enteros `key`. Recibe como parámetro una llave de tipo `String`, y mediante un ciclo `for` se recorre para extraer cada carácter y asignarle su posición correspondiente en la llave.

```
private void setKey(String key) {  4 usages
    this.key = new int[key.length()];
    for (int i = 0; i < key.length(); i++) {
        this.key[i] = key.charAt(i);
    }
}
```

Figura 3: Método setKey(String Key)

- **BigVignere(String key)**: Este método es uno de los constructores de la clase `BigVignere`. Incluye un parámetro de tipo `String` que se asigna como llave de encriptación con el método `setKey()`, y además genera la matriz de encriptación con el método `generateAlphabet()`.

```
public BigVignere(String key) {  1 usage
    setKey(key);
    generateAlphabet();
}
```

Figura 4: Método BigVignere(String Key)(Constructor n°1)

- **BigVignere()**: Este es el segundo constructor de la clase `BigVignere`. A diferencia del anterior, este método no tiene parámetros, y la clave se asigna mediante una entrada por parte del usuario. El algoritmo solicita la clave de encriptación con la clase `Scanner` y posteriormente la asigna con el método `setKey()`. La matriz, al igual que en el otro constructor, se genera con el método `generateAlphabet()`.

```
public BigVignere() {  no usages
    Scanner scanner = new Scanner(System.in);
    System.out.print("Ingrese la clave: ");
    String key = scanner.nextLine();
    setKey(key);
    scanner.close();
    generateAlphabet();
}
```

Figura 5: Método BigVignere()(Constructor n°2)

- **int getCharIndex(int character)**: Este método entrega la posición de un carácter en el alfabeto. Si no es capaz de encontrar el carácter, el método retorna -1. Para lograrlo, recorre la primera columna de la matriz de arriba hacia abajo.

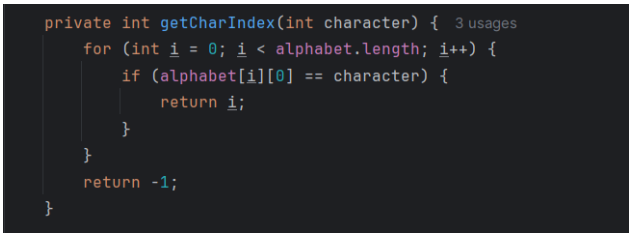

Figura 6: Método getCharIndex(int character)

- **String encrypt(String message)**: Este es el método encargado de encriptar mensajes. Para su correcto funcionamiento, el mensaje encriptado se maneja con un `StringBuilder`, lo que facilita la concatenación de caracteres en cada ciclo. Se utiliza una variable entera `keyIndex`, que funciona como índice alternativo al ciclo `for` y sirve para omitir los caracteres vacíos del mensaje. El funcionamiento del algoritmo es el siguiente:
  - 1. Se crea un ciclo `for` desde el índice 0 hasta la longitud del mensaje.
  - 2. Se obtiene el carácter en la posición actual del ciclo con el método `charAt(i)`.
  - 3. Se verifica si el carácter es un espacio vacío ' '. En caso afirmativo, se agrega directamente al mensaje final y se continúa con la siguiente iteración.
  - 4. Se obtiene la posición del carácter en la matriz `alphabet`. Si la posición es -1 (es decir, no se encuentra en la matriz), se agrega un '?' al mensaje final.
  - 5. Se obtiene el carácter correspondiente de la llave, reutilizando la clave en caso de ser más corta que el mensaje, y omitiendo espacios en blanco.
  - 6. Se obtiene el índice en la matriz `alphabet` del carácter correspondiente de la llave.
  - 7. Se agrega el carácter encriptado usando la posición del carácter del mensaje original como fila y el índice de la llave como columna.
  - 8. Se incrementa en 1 la variable `keyIndex` para mantener la sincronización de la clave solo en los caracteres no vacíos.
  - 9. Finalmente, se retorna el mensaje encriptado.

```java
public String encrypt(String message) {  2 usages
    StringBuilder encryptedMessage = new StringBuilder();
    int keyIndex = 0;
    for (int i = 0; i < message.length(); i++) {
        char messageChar = message.charAt(i);
        if (messageChar == ' ') {
            encryptedMessage.append(' ');
            continue;
        }
        int messagePos = getCharIndex(messageChar);
        if (messagePos == -1) {
            encryptedMessage.append('?');
            continue;
        }
        int keyChar = key[keyIndex % key.length];
        int keyCol = getCharIndex(keyChar);
        encryptedMessage.append(alphabet[messagePos][keyCol]);
        keyIndex++;
    }
    return encryptedMessage.toString();
}
```

Figura 7: Método encyrpt(String message)

- **String decrypt(String message)**: Este es el método para desencriptar un mensaje que haya sido cifrado con el método Vigenère y una clave conocida. Su funcionamiento es muy similar al método de encriptación, pero aplicado de manera inversa. Sigue los siguientes pasos:
  - 1. Se crea un ciclo `for` desde el índice 0 hasta la longitud del mensaje.
  - 2. Se obtiene el carácter encriptado en la posición actual con `charAt(i)`.
  - 3. Se evalúan los distintos valores del carácter. Si es un símbolo desconocido '?' o un espacio en blanco ' ', se agrega directamente al mensaje desencriptado con el método `append()` del `StringBuilder`.
  - 4. Se obtiene el carácter correspondiente de la clave en la posición actual.
  - 5. Se obtiene la posición del carácter de la clave en la matriz `alphabet`.
  - 6. Se inicializa una variable `int originalPos = -1` para guardar la posición de la columna del carácter original.
  - 7. Se inicia un segundo ciclo `for` desde 0 hasta la longitud de la matriz `alphabet`. En cada iteración, se verifica si el carácter cifrado coincide con el carácter en la fila correspondiente a la clave y columna del segundo `for`. Si coincide, se guarda el índice de esa columna en `originalPos`.
  - 8. Finalizado el segundo ciclo, se utiliza `originalPos` para obtener el carácter original desde la primera columna de la matriz.
  - 9. Se incrementa en uno el índice alternativo de la clave.
  - 10. Finalmente, se retorna el mensaje desencriptado.

```java
public String decrypt(String message) {  2 usages
    StringBuilder decryptedMessage = new StringBuilder();
    int keyIndex = 0;
    for (int i = 0; i < message.length(); i++) {
        char messageChar = message.charAt(i);
        if (messageChar == ' ') {
            decryptedMessage.append(' ');
            continue;
        } else if (messageChar == '?') {
            decryptedMessage.append('?');
            continue;
        }
        int keyChar = key[keyIndex % key.length];
        int keyRow = getCharIndex(keyChar);
        int originalPos = -1;
        for (int j = 0; j < alphabet.length; j++) {
            if (alphabet[keyRow][j] == messageChar) {
                originalPos = j;
                break;
            }
        }
        decryptedMessage.append(chars.charAt(originalPos));
        keyIndex++;
    }
    return decryptedMessage.toString();
}
```

Figura 8: Método decrypt(String message)

- **char search(int position)**: Este método es cual realiza la búsqueda del carácter recorriendo la matriz Vignere de izquierda a derecha y de arriba hacia abajo para encontrar la posición buscada.

```java
public char search(int position) {  1 usage
    for (int i = 0; i < alphabet.length; i++) {
        for (int j = 0; j < alphabet.length; j++) {
            if (position == i * alphabet.length + j) {
                return alphabet[i][j];
            }
        }
    }
    return '?';
}
```

Figura 9: Método search(int position)

- **void reEncrypt()**: Este método le pregunta al usuario cual es el mensaje que desea encriptar y cual será la clave de encriptación, para luego imprimir el mensaje cifrado, esto lo hace llamando a los métodos setKey, encrypt y decrypt los cuales fueron explicados anteriormente.

```
public void reEncrypt() {  no usages
    Scanner scanner = new Scanner(System.in);
    System.out.print("Ingrese el mensaje encriptado:");
    String encryptedMessage = scanner.nextLine();
    String decryptedMessage = decrypt(encryptedMessage);
    System.out.print("Ingrese la nueva clave de encriptacion: ");
    String newKey = scanner.nextLine();
    setKey(newKey);
    String newEncryptedMessage = encrypt(decryptedMessage);
    System.out.println("El mensaje encriptado con la nueva clave es: " + newEncryptedMessage);
    scanner.close();
}
```

Figura 10: Método reEncrypt()

- **char optimalSearch(int position)**: Este método realiza la busqueda de un carácter especifico dentro de la matriz de una manera más eficiente que el método search, esto lo logra realizando operaciones lineales sobre la variable 'position' para así obtener de manera directa la fila y la columna del carácter deseado, las operaciones que realiza son:
  - Para la fila se realiza 'posición / alphabet,length',lo que entrega un valor entero el cual seria el valor de la fila del carácter.
  - Para la columna se realiza 'posición % alphabet.length',la cual también entrega un valor entero que esta vez será el valor de la columna del carácter.

  Para al final retornar el carácter deseado.

```
public char optimalSearch(int position) {  1 usage
    int row = position / alphabet.length;
    int col = position % alphabet.length;
    return alphabet[row][col];
}
```

Figura 11: Método optimalSearch

- **Void startTest()**: Este método es quien realiza las pruebas de encriptación de los mensajes, esto lo logra siguiendo los siguientes pasos para cada una de las 5 claves de encriptación mediante un ciclo for:
  - 1. Guarda el tiempo de inicio en nano segundos para medir cuanto demora la prueba.
  - 2. Obtiene la clave que se probará y su longitud.
  - 3. Configura la clave de encriptacion que se usara en los métodos encrypt y decrypt.
  - 4. Encripta un mensaje de prueba con la clave que se configuro en el paso 3.
  - 5. Desencripta el mensaje para verificar que funciona de manera correcta.
  - 6. Calcula cuanto tiempo se demoro en encriptar y desencriptar el mensaje.
  - 7. Luego muestra cuanto tiempo se tardo en hacer la prueba en mili segundos.
  - 8. Por ultimo cierra el ciclo for e imprime una linea separadora formada por múltiples '-'.

```
public void startTest() {  1 usage
    System.out.println("Iniciando test de encriptacion.\n");
    for (int i = 0; i < testPasswords.length; i++) {
        long startTime = System.nanoTime();
        String pass = testPasswords[i];
        System.out.print("Iniciando test para clave de " + pass.length() + " caracteres.\n");
        setKey(pass);
        String encryptedMessage = encrypt(testMessage);
        decrypt(encryptedMessage);

        long endTime = System.nanoTime();
        long duration = endTime - startTime;

        System.out.println("Duracion del test de encriptacion: " + duration / 1000000 + " milisegundos.\n");
    }
    System.out.println("-------------------------------------------------");
}
```

Figura 12: Método startTest

3. Experimentación y Resultados

Para probar el código escrito se hicieron 7 pruebas, cada una de las pruebas se hizo para un caso específico con un mensaje de una cantidad de caracteres específicos, las cantidades de caracteres probadas fueron : 10, 50, 100, 500, 1000, 5000 y 10000 caracteres respectivamente.

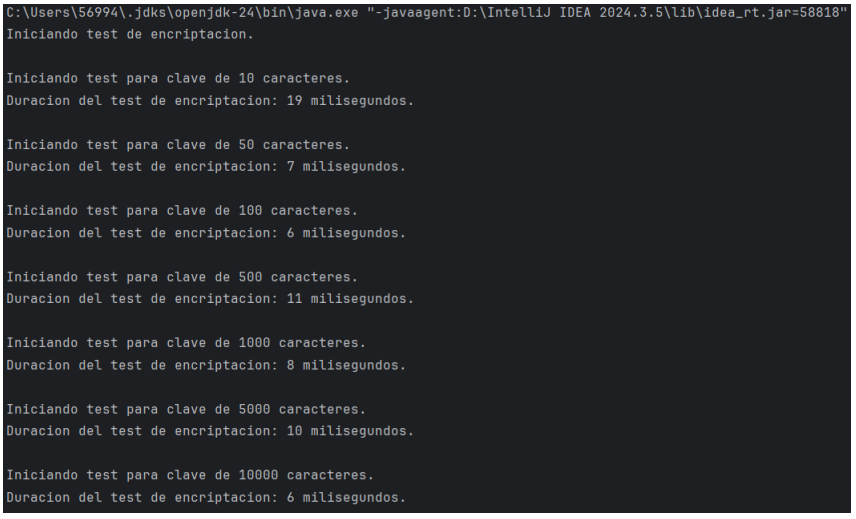

Figura 13: Pruebas claves

Se puede apreciar que mientras más aumenta de caracteres la clave, el programa demora menos. Como grupo, creemos que esto se debe al caché del programa, al este funcionar en base a iteraciones. El programa accedería al método de encriptación con una mayor rapidez.

4. Conclusiones

¿Cuáles dificultades se presentaron y cómo se resolvió?

La principal dificultad que se presento a la hora de realizar el código fue la parte de encriptación y desencriptación, puesto que es un método abstracto y difícil de encontrar una implementación adecuada utilizando código. Por otra parte, el método de búsqueda optimizada requirió tiempo para encontrar y visualizar una forma de obtener el mismo resultado utilizando una fracción de tiempo que toma el método original.

¿Cómo se logró optimizar la búsqueda de las posiciones?

Para optimizar la búsqueda de posiciones de caracteres específicos se implementó el método optimalSearch(descrito anteriormente) el cual logra ahorrar tiempo y recursos gracias a que este no recorre una matriz, si no que solo trabaja con operaciones lineales, por lo que agiliza la búsqueda.

```java
public char optimalSearch(int position) {  1 usage
    int row = position / alphabet.length;
    int col = position % alphabet.length;
    return alphabet[row][col];
}
```

Figura 14: Método optimalSearch

Respecto a la notación Big O el método search presenta un O(n^2) puesto que como se mencionó antes, este método funciona recorriendo la matriz, en cambio el método optimalSearch presenta una notación O(1) ya que este solo realiza operaciones aritméticas básicas y luego obtiene el carácter en una matriz.

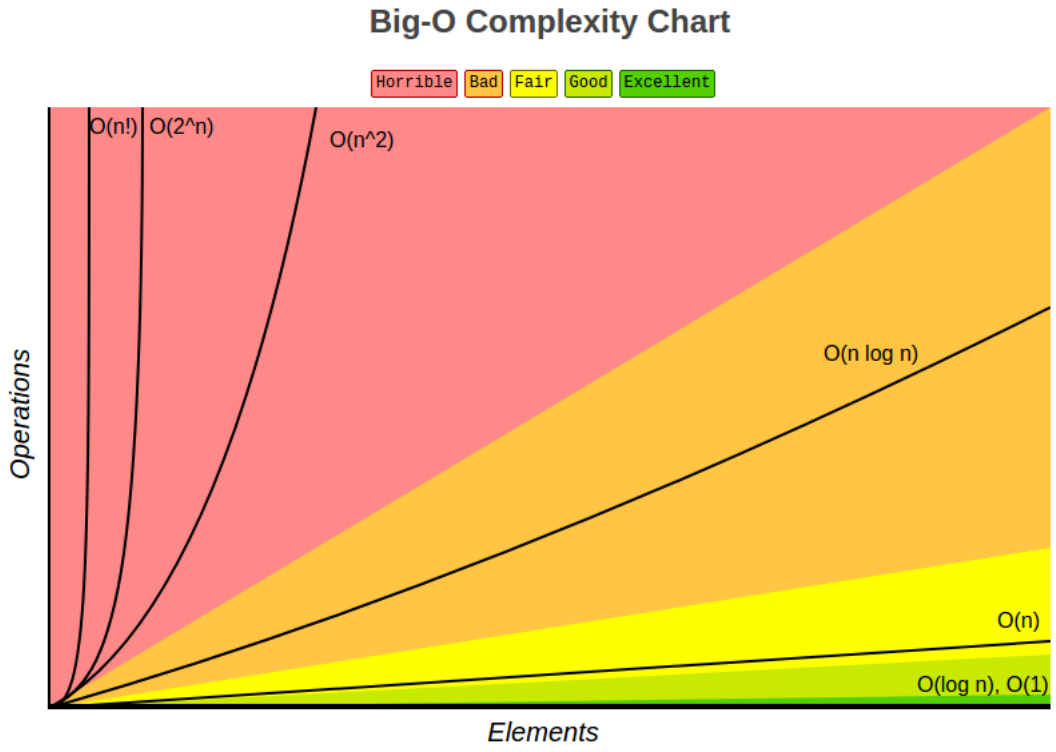

Figura 15: Comparación Big O

En relación con el tamaño de la clave a utilizar, ¿qué recomendaciones entregaría para la creación de la clave?

Se recomienda utilizar una clave igual o más larga que el mensaje que se desea enviar. Si es posible utilizar caracteres aleatorios sin un patrón fijo que sean difíciles de descifrar.

https://github.com/AllenMora/Lab-1-EDA