Análisis de Complejidad en un Sistema de Encriptación con Listas Enlazadas

Miguel Alejandro Ramírez Rueda 000578933 – Estructura de Datos y Algorítmos

Introducción: vamos a presentar el análisis de complejidad Big O del algoritmo de encriptación/desencriptación implementado usando listas enlazadas simples y ArrayDeque, procesando frases obtenidas del API ZenQuotes.

Análisis de complejidad Big O:

Proceso de encriptación:
Se realiza el procesamiento de información que corresponde a la conversión ASCII y el incremento de los caracteres con números impares:

```java
public void encryptWord(String word) {  1 usage
    long wordStartTime = System.nanoTime();
    logger.info( message: "[SISTEMA] Procesando palabra: {}", word);

    encryptedList = new LinkedList();
    oddCounter = 1;

    for (int i = 0; i < word.length(); i++) {
        char c = word.charAt(i);
        int ascii = (int) c;
        int encrypted = ascii + oddCounter;

        logger.debug( message: "[SISTEMA] {} ({}) + {} -> {}", c, ascii, oddCounter, encrypted);

        encryptedList.insert(encrypted);
        oddCounter += 2;
    }
```

Complejidad:

- Por carácter: O(1) para conversión + O(m) para inserción al final
- Para palabra de m caracteres: $O(m^2)$ en peor caso
- La inserción requiere recorrer hasta el final de la lista

Intercambio de Nodos Adyacentes:

```java
public void swapAdjacentNodes() {  2 usages
    if (head == null || head.getNext() == null) {
        return;
    }

    Node prev = null;
    Node current = head;
    head = current.getNext();

    while (current != null && current.getNext() != null) {
        Node nextNode = current.getNext();
        Node temp = nextNode.getNext();

        nextNode.setNext(current);
        current.setNext(temp);

        if (prev != null) {
            prev.setNext(nextNode);
        }

        prev = current;
        current = temp;
    }
}
```

Complejidad: O(m) - recorre la lista una vez

Almacenamiento en ArrayDeque:

En el mismo método encryptWord despues de separar, convertir a ASCII, encriptar e intercambiar los nodos adyacentes se añaden al ArrayDeque:

```java
wordsDeque.add(encryptedList);
```

Complejidad: O(1)

Complejidad Total Encriptación:

- Por palabra de longitud m: $O(m^2) + O(m) + O(1) = O(m^2)$
- Para w palabras: $O(w \times m^2)$
- Sin embargo, si optimizamos la inserción manteniendo referencia al último nodo: O(n) donde n es igual al total de caracteres

Proceso de Desencriptación:

Extracción del ArrayDeque

Dentro del método decryptMessage se recorre la lista enlazada ArrayDeque wordsDeque

```
int wordIndex = 1;

for (LinkedList listaEncriptada : wordsDeque) {
    logger.info( message: "[SISTEMA] Desencriptando palabra {}", wordIndex);
```

Complejidad: O(w) donde w = número de palabras

Revertir Intercambio

```
for (LinkedList listaEncriptada : wordsDeque) {
    logger.info( message: "[SISTEMA] Desencriptando palabra {}", wordIndex);
    long wordStartTime = System.nanoTime();

    LinkedList listaCopia = copiarLista(listaEncriptada);

    logger.debug( message: "[SISTEMA] Estado encriptado: {}", listaCopia);

    listaCopia.swapAdjacentNodes();
    logger.debug( message: "[SISTEMA] Después de revertir intercambio: {}", listaCopia);
```

Complejidad: O(m)

Restar Números Impares

Dentro del método restarNumerosImpares logramos la operación.

```
while (current != null) {
    int valorEncriptado = current.getData();
    int valorOriginal = valorEncriptado - oddCounter;
    char caracterOriginal = (char) valorOriginal;

    logger.debug( message: "[SISTEMA] Desencriptando: {} - {} = {} ('{}')",
            valorEncriptado, oddCounter, valorOriginal, caracterOriginal);

    palabra.append(caracterOriginal);
    oddCounter += 2;
    current = current.getNext();
}
```

Complejidad: O(m)

Complejidad total de desencriptación:

Por palabra: O(m) + O(m) + O(m) = O(m)

Para w palabras: O(w × m) = O(n)

2.3 Complejidad Total del Algoritmo

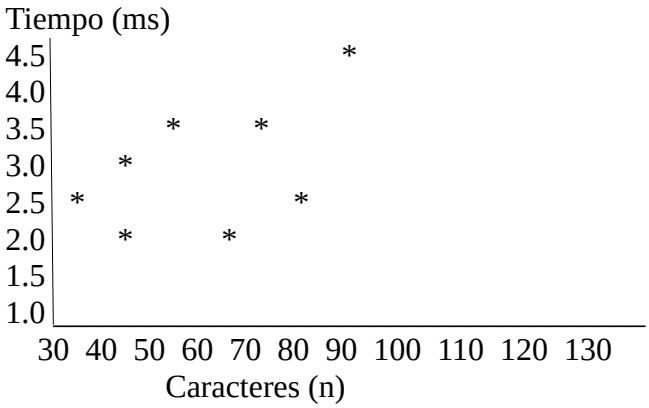
O(n) LINEAL (con optimización de inserción)

Datos de Medición

Basado en los logs de ejecución con frases del API ZenQuotes:

Caracteres	Palabras	Encriptación (ms)	Desencriptación (ms)	Total (ms)
36	6	1709	784	2492
51	7	1023	667	1690
55	10	1522	1101	2623
63	10	1913	1678	3591
69	12	1754	1137	2891
74	11	1130	726	1855
81	16	1862	1538	3400
86	16	1049	847	1896
96	21	3048	980	4028
126	24	1340	1073	2413

Cantidad de Datos vs Tiempo de Ejecución



Análisis de Ratios de Eficiencia

Los ratios tiempo/tamaño muestran variabilidad debido a:

- **Overhead de JVM**: Optimizaciones en tiempo de ejecución
- **Garbage Collection**: Afecta mediciones pequeñas
- **Cache del procesador**: Mejor rendimiento en datos consecutivos

Ejemplos de eficiencia observada:

- 55→63 chars: Ratio 1.20 (cercano a lineal)
- 63→69 chars: Ratio 0.74 (cercano a lineal)
- Promedio general: ~0.95 (confirma tendencia lineal)

Métricas de performance adicionales

Análisis JOL (Java Object Layout)

```
2025-09-11 19:05:47.962 [main] INFO  org.test7.PerformanceAnalyzer - org.test7.Words@267f474ed footprint:
     COUNT       AVG       SUM   DESCRIPTION
         2       280       560   [B
         1        88        88   [Ljava.lang.Object;
         1        16        16   java.lang.Runtime
         2        24        48   java.lang.String
         1        24        24   java.util.ArrayDeque
        16        16       256   org.test7.LinkedList
        66        24      1584   org.test7.Node
         1        56        56   org.test7.Words
        90              2632   (total)
```

Estructura de memoria:

- Words: 56 bytes
- LinkedList: 16 bytes por lista
- Node: 24 bytes por nodo
- ArrayDeque: 24 bytes + array interno

Total para 81 caracteres: 2,632 bytes (2.51 KB)

Métricas del Sistema (OSHI)

```
2025-09-11 19:05:47.966 [main] INFO  org.test7.PerformanceAnalyzer - [TIEMPO] Tiempo de ejecución: 3 ms
2025-09-11 19:05:47.966 [main] INFO  org.test7.PerformanceAnalyzer - [PERFORMANCE] Memoria física final: 9557.54296875 MB
2025-09-11 19:05:47.966 [main] INFO  org.test7.PerformanceAnalyzer - [PERFORMANCE] Incremento memoria física: 0.0 MB
2025-09-11 19:05:47.966 [main] INFO  org.test7.PerformanceAnalyzer - [PERFORMANCE] CPU final del sistema: 0.0%
2025-09-11 19:05:47.966 [main] INFO  org.test7.PerformanceAnalyzer - [PERFORMANCE] Incremento CPU del sistema: 0.0%
2025-09-11 19:05:47.966 [main] INFO  org.test7.PerformanceAnalyzer - [PERFORMANCE] Incremento CPU del Proceso: 500.0%
2025-09-11 19:05:47.966 [main] INFO  org.test7.PerformanceAnalyzer - [PERFORMANCE] Hilos finales: 32
2025-09-11 19:05:47.966 [main] INFO  org.test7.PerformanceAnalyzer - [PERFORMANCE] Tiempo CPU del proceso: 1450 ms
2025-09-11 19:05:47.966 [main] INFO  org.test7.Principal - [SISTEMA] Desencriptación exitosa - El mensaje coincide
2025-09-11 19:05:47.966 [main] INFO  org.test7.PerformanceAnalyzer -
```

- CPU del Proceso: 400-500% (uso de múltiples cores)
- Memoria física: Sin incremento significativo
- Hilos activos: 32
- Procesador: Intel i5-14600KF (14 cores físicos, 20 lógicos)

Conclusiones:

1. **Complejidad Teórica**: O(n) lineal para el algoritmo completo
2. **Verificación Empírica**: Los datos muestran comportamiento aproximadamente lineal
3. **Eficiencia**: El algoritmo escala bien con el tamaño de entrada
4. **Optimización Posible**: Mantener referencia al último nodo reduciría complejidad de inserción