Informe Tercera Entrega

Cristóbal Castrillón Balcázar   
Nicolás Pérez Fonseca

# Acta de evaluación de la entrega anterior

A nivel de funcionalidad el programa cuenta con las implementaciones completas de la primera y segunda entrega, por tanto, cuenta con los comandos exigidos por el enunciado.

Como incumplimientos a los lineamientos del enunciado, hubo dos en la entrega previa. Si bien el algoritmo de compresión de Huffman se aplicaba, el resultado se escribía como caracteres (tipo de dato «char») y no como su representación binaria; por lo tanto, el tamaño final del archivo en muchos casos se veía incrementado, producto de este error. Por lo anterior, los comandos de compresión y descompresión tuvieron que ser ajustados para cumplir con una efectiva compresión de las secuencias cargadas en memoria.

## Ajustes realizados

Ambas inconsistencias halladas en la sustentación de la segunda entrega fueron resueltas.

La implementación presentada en la segunda entrega adolecía de guardar en memoria archivos de texto en representación binaria (o y 1) de caracteres, por lo que cada bit del archivo en su representación física era de un byte y, debido a esto, la supuesta compresión no era tal. Por lo tanto, el algoritmo de la implementación se ajustó para guardar la cadena de caracteres original en su representación binaria, escribiéndose en disco con esta especificación y efectivamente aplicándose una compresión a las secuencias almacenadas en la memoria utilizada por el programa.

Debido a la falla previamente mencionada, el algoritmo de descompresión tuvo que ser ajustado para abrir ficheros binarios y no de texto como se presentó en un comienzo, aplicar la descompresión a estos archivos para posteriormente cargarlos en memoria.

De acuerdo con los ajustes mencionados se considera la segunda entrega completada en su totalidad.

## Conclusiones acta de la entrega anterior

Se concluye que la entrega anterior se llevó a cabo con un resultado mejorable, ya que la implementación de los comandos de descompresión y compresión no se desempañaban como el docente requería. Funcionalmente, el archivo no comprimía los datos codificados mediante el algoritmo de Huffman. Por último, podemos decir que el algoritmo fue asimilado, pero no aplicado correctamente al requerimiento.

# Documentación

## Vista General

El proyecto se compone de seis archivos. Tres de estos, corresponden a archivos fuente (su extensión es ‘.cpp’), y tres son archivos de encabezado (extención ‘.hpp’). El apartado *Arquitectura del Proyecto* ilustra la arquitectura del proyecto y los tipos abstractos de datos (TADs) —que en este caso son *structs*— utilizados para implementar las funcionalidades requeridas, y sus relaciones.

### Arquitectura del Proyecto

### Archivos de Encabezado

Los archivos de extensión ‘.hpp’ son usados para almacenar las definiciones de las estructuras usadas en el programa.

* **ArchivoFASTA.hpp:** Este archivo contiene la definición de la estructura que almacena toda información provista por el archivo FASTA, y define las funciones de las cuales hace uso el programa para proveer las funcionalidades requeridas (Figura 2).
* **Secuencia.hpp:** Este archivo contiene la definición de la estructura que almacena la información de una secuencia provista por el archivoFASTA.
* **Base.hpp:** Este archivo contiene la definición de la estructura que almacena la información correspondiente a una base en un archivo FASTA, a saber, la letra (el carácter que representa), y la cantidad de veces que se encuentra en una secuencia.
* **HuffmanNode.hpp:** Este archivo contiene la definición de la estructura del nodo utilizado en el árbol de Huffman para la compresión de la escritura del archivo binario de secuencias.
* **HuffmanTree.hpp:** Este archivo contiene la definición de la estructura del árbol de Huffman utilizado en la compresión de la escritura del archivo binario de secuencias.
* **Grafo.hpp:**
* **NodoGrafo.hpp:**

### Archivos Fuente

Los archivos de extensión ‘.cpp’ son usados para almacenar las implementaciones de las estructuras definidas en los archivos de encabezado que carecen de ellas.

* **ArchivoFASTA.cpp:** Almacena las implementaciones de las funciones de la *struct* ArchivoFASTA. Se implementan las funciones de los comandos que recibe el programa desde el main.cpp.
* **Secuencia.cpp:** Almacena la implementación de las bases y las funciones asociadas al tipo abstracto de dato Secuencia.
* **main.cpp**: Es el archivo principal del programa, y almacena la entrada de comandos, interfaz de interacción entre el usuario del programa y las funcionalidades de este.

## Precondiciones y postcondiciones

En esta sección se enuncian tanto las precondiciones, como las postcondiciones de cada una de las funciones contenidas en los archivos fuente, exceptuando la función/archivo principal.

|  |  |  |  |
| --- | --- | --- | --- |
| **Ubicación** | **Función** | **Precondición** | **Postcondición** |
| ArchivoFASTA | getLineaDescriptiva | int | string |
| setLineaDescriptiva | string | N/A |
| getSecuencia | string | int |
| setSecuencia | string | N/A |
| cargarArchivo | string | bool |
| conteoBasesXSecuencia | N/A | N/A |
| conteoSecuencias | N/A | int |
| histograma | string | N/A |
| listarSecuencias | N/A | N/A |
| subsecuencia | string | N/A |
| enmascarar | string | N/A |
| guardar | string | bool |
| ayuda | N/A | N/A |
| codificar | string | HuffmanNode\* |
| decodificar | string | bool |
| armarMatrizSecuencia | vector | N/A |
| getIndiceSecuencia | String | int |
| ruta\_mas\_corta | string | N/A |
| base\_remota | string | N/A |
| HuffmanTree | codificar | HuffmanNode\*, string, map | N/A |
| decodificar | HuffmanNode\*, int string | string |
| getNode | char, int, HuffmanNode\*, HuffmanNode\* | HuffmanNode\* |
| Secuencia | setSecuencia | string | N/A |
| appendSecuencia | string | N/A |
| getBases | N/A | vector |
| setBases | vector | N/A |
| setIncompleta | N/A | bool |
| getIncompleta | N/A | N/A |
| construirGrafo | N/A | N/A |
| setMatriz | vector | N/A |
| getMatriz | N/A | vector |
| calcularPeso | Int | float |
| conteoBases | N/A | N/A |
| rutaMasCorta | Int | N/A |
| Base | getLetraBase | N/A | char |
| getCantidadBase | N/A | int |
| setLetraBase | char | N/A |
| setCantidad | int | N/A |
| Grafo | shortestPath | NodoGrafo | N/A |

## Complejidad

En esta sección se enuncia la complejidad algorítmica asociada a cada una de las funciones correspondientes a los respectivos comandos.

* **Cargar:** Complejidad de tiempo lineal (O(n)) debido a que entre mayor sea la longitud del documento tanto en extensión de líneas como en la longitud de las mismas m ́as operaciones son necesarias para procesar la información.
* **Conteo:** Complejidad de tiempo lineal (O(n)) debido a que, entre mayor sea el volumen de información almacenado, la operación se repite de acuerdo con el número de datos a contar.
* **Histograma:** Complejidad (O(n^3)) debido a la cantidad de bucles anidados
* **Listas Secuencias:** Complejidad (O(n^2)) debido a la cantidad de bucles anidados.
* **Es Subsecuencia:** Se mantiene lineal debido al que, de igual manera que las anteriores, trabaja con un volumen de datos que depende de la cantidad de secuencias almacenadas en el aplicativo. Hace uso de funciones de STL como std::string::find(), que, de acuerdo con la documentación oficial, es de complejidad lineal.
* **Enmascarar:** Complejidad lineal debido a que depende del volumen de datos a procesar. Hace uso de funciones de STL: std::string::find() y std::string::replace(), ambas de acuerdo con la documentación oficial son de complejidad lineal.
* **Guardar:** Escribe líneas de texto en un archivo especificado, y la operación aumenta de acuerdo con el volumen de datos a escribir, por lo tanto, es lineal, ya que la operación es de tipo (O(n))
* **Codificar:**
* **Decodificar:**
* **Ruta Más Corta:**
* **Base Remota:**