
# Pontificia Universidad Javeriana
# Departamento de Ingeniería de Sistemas
# Estructuras de Datos
# Proyecto del curso, 2025-30

## 1 Descripción del problema

En general, la información genética de un organismo se encuentra contenida en su genoma. Para manipular la información del genoma de un organismo cualquiera, se usan secuencias de información conocidas como códigos genéticos. Estos códigos genéticos están compuestos de letras que representan las cuatro bases nitrogenadas del ácido desoxirribonucleico (ADN) o del ácido ribonucleico (ARN). Mientras en el caso del ADN las bases son adenina (`A`), timina (`T`), guanina (`G`) y citosina (`C`); para el ARN las bases son adenina (`A`), uracilo (`U`), guanina (`G`) y citosina (`C`).

Aunque hay muchos avances relacionados con el descubrimiento de los códigos genéticos de los organismos del planeta, mucho trabajo aún queda por hacer. Para guardar la información que se conoce (y señalar la que se desconoce) se usa un formato de códigos genéticos conocido como `FASTA`.

## 2 Descripción del proyecto

El objetivo del presente proyecto es construir un sistema que permita algunas manipulaciones sencillas sobre archivos que contienen códigos genéticos, aplicando los conceptos de Estructuras de Datos vistos a lo largo del semestre.

### 2.1 Formato FASTA

Los archivos en formato `FASTA` (extensión de archivo `.fa`) son basados en texto. Pueden contener uno o varios códigos genéticos que componen un genoma. Cada código genético empieza con una línea de texto que tiene el formato:

> *>descripcion_secuencia*

Debe notarse que siempre empieza con el caracter '>' y justo después se presenta una cadena no vacía que describe la secuencia (nombre de la secuencia). Esta cadena puede contener espacios o símbolos, además de los caracteres usuales. Las siguientes líneas (hasta la siguiente secuencia o hasta el final del archivo) contienen los datos que describen el código genético, utilizando los códigos de las bases ya descritos. Estas líneas están justificadas y todas tienen un ancho constante (que depende de cada secuencia, es decir, no siempre el ancho es el mismo para todas las secuencias), con la posible excepción de la última línea. Un ejemplo de un archivo `FASTA` puede ser:

```
>Full_SEQUENCE
CTCCGGTGAGAAATTTTGGGATGTATCAAATCACGGTCCTACTAC
TTACCGCCAACACGAGCCGGAACCCCCTAGATCAATTCATGCTTT
TCCCTTCACGCGAAGGAGTCGGAAGTGATCTGTATGAAGCTATTA
CCCTAGGTGGCCACACCTAC
>Incomplete_sequence
URDNSHVNKSCUANADDDVMMDVHRSRGNUNTSBTTMCGNBUUBTUMNAYKSTRYMVBWVNSC
SUUDDYVBCGNDRUURUBDHVGTYAVAVVSAKHUTSWCUBUUMSMDVDKBWRHHBHDBDWUDC
CAUWBRNYSTVTBBCKGDCMSNTKBNTRTNYRNWNWNVCSCNDDWHUHWRRTUMWNKHUBDWA
DUNYUVKHNKSDCYAVARTUWDVSTDVVMMVUHVBCDGVBSKKBHKSVHHGKSAADDVBRKSS
UKURTKTNAWYBKVRRBGKGBMGKUCGSMDTDTKCKVUYNVDWRWNBRCGKDWWUNBYMTGBN
YTAHCUTAGNBKWWGNDRKMNCDVDTKNRGVBRVMKGBKUBGBRHVUSTBCGDV
```

El signifcado de cada letra o código se explica en la Tabla 1.

### 2.2 Componentes del sistema

A continuación se describen los componentes individuales que conforman el presente proyecto:

| Código | Significado |
|---|---|
| A | Adenina |
| C | Citosina |
| G | Guanina |
| T | Timina |
| U | Uracilo |
| R | A o G |
| Y | C, T o U |
| K | G, T o U |
| M | A o C |
| S | C o G |
| W | A, T o U |
| B | C, G, T o U |
| D | A, G, T o U |
| H | A, C, T o U |
| V | A, C o G |
| N | A, C, G, T o U |
| X | Máscara |
| - | Espacio de longitud indeterminada |

Table 1: Relación de cada código del formato FASTA con su significado biológico (tomada de https://blast.ncbi.nlm.nih.gov/doc/blast-topics/).

#### 2.2.1 Componente 1: Resumen de la información de un genoma

**Objetivo:** A partir de un archivo FASTA, mostrar en pantalla la información que el usuario requiera. Este componente se implementará utilizando estructuras lineales, y contiene los siguientes comandos:

- **comando:** `cargar` *nombre_archivo*
  **salida en pantalla:**
  (archivo vacío) *nombre_archivo* `no contiene ninguna secuencia.`
  (archivo erróneo) *nombre_archivo* `no se encuentra o no puede leerse.`
  (una sola secuencia) `1 secuencia cargada correctamente desde` *nombre_archivo* `.`
  (varias secuencias) *n* `secuencias cargadas correctamente desde` *nombre_archivo* `.`
  **descripción:** Carga en memoria los datos contenidos en el archivo identificado por *nombre_archivo*, es decir, utiliza adecuadamente las estructuras lineales para cargar la información de los genomas en memoria. Si dentro de la misma sesión de trabajo ya se han cargado otros archivos de secuencias (usando el comando `cargar`), la información debe sobreescribirse en memoria, es decir, no se deben combinar informaciones de secuencias de diferentes archivos.

- **comando:** `listar_secuencias`
  **salida en pantalla:**
  (no hay secuencias cargadas) `No hay secuencias cargadas en memoria.`
  (resultado exitoso) `Hay` *n* `secuencias cargadas en memoria:`
  `Secuencia` *descripcion_secuencia_1* `contiene` *b* `bases.`
  `Secuencia` *descripcion_secuencia_2* `contiene al menos` *b* `bases.`
  `...`
  **descripción:** Imprime la cantidad de secuencias actualmente en memoria, y luego, en n líneas (una para secuencia) la información básica (nombre y cantidad de bases diferentes) de cada secuencia. Si la secuencia es completa (no tiene el código `'-'`) indica la cantidad de bases exactas (`contiene` *b* `bases`); si la secuencia es incompleta (incluye uno o varios códigos `'-'`) indica la cantidad mínima de bases, sin contar el código `'-'` (`contiene al menos` *b* `bases`).

- **comando:** `histograma` *descripcion_secuencia*
  **salida en pantalla:**
  (la secuencia no existe) `Secuencia inválida.`
  (la secuencia existe) `A :` *frecuencia_A* `\n C :` *frecuencia_C* `\n ...`
  **descripción:** Imprime el histograma de una secuencia, en caso de que exista. El histograma se define como el conteo (frecuencia) de cada código en la secuencia. Por cada línea (`'\n'` es el caracter de salto

de línea) se escribe el código y la cantidad de veces que aparece en la secuencia. El ordenamiento del histograma está dado por la Tabla 1. El código `'-'` debe incluirse también como parte del histograma.

- **comando:** `es_subsecuencia` *subsecuencia*
  **salida en pantalla:**
  (no hay secuencias cargadas) `No hay secuencias cargadas en memoria.`
  (la subsecuencia no existe) `La subsecuencia dada no existe dentro de las secuencias cargadas en memoria.`
  (varias subsecuencias) `La subsecuencia dada se repite` *s* `veces dentro de las secuencias cargadas en memoria.`
  **descripción:** Determina si una subsecuencia (secuencia corta de bases) dada por el usuario, existe dentro de las secuencias cargadas en memoria. Si es así, determina la cantidad de veces en las que esta subsecuencia dada se repite. No es necesario indicar el nombre de la secuencia donde se encuentra la repetición (el conteo es general sobre todas las secuencias cargadas en memoria).

- **comando:** `enmascarar` *subsecuencia*
  **salida en pantalla:**
  (no hay secuencias cargadas) `No hay secuencias cargadas en memoria.`
  (no se enmascararon subsecuencias) `La subsecuencia dada no existe dentro de las secuencias cargadas en memoria, por tanto no se enmascara nada.`
  (varias subsecuencias esmascaradas) *s* `subsecuencias han sido enmascaradas dentro de las secuencias cargadas en memoria.`
  **descripción:** Enmascara una subsecuencia (secuencia corta de bases) dada por el usuario, si existe. Los elementos que pertenecen a la subsecuencia se enmascaran cambiando cada base individual por el código `'X'`. No es necesario indicar el nombre de la secuencia donde se realiza el enmascarado (el proceso es general sobre todas las secuencias cargadas en memoria).

- **comando:** `guardar` *nombre_archivo*
  **salida en pantalla:**
  (no hay secuencias cargadas) `No hay secuencias cargadas en memoria.`
  (escritura exitosa) `Las secuencias han sido guardadas en` *nombre_archivo* `.`
  (problemas en archivo) `Error guardando en` *nombre_archivo* `.`
  **descripción:** Guarda en el archivo *nombre_archivo* las secuencias cargadas en memoria. Se debe tener en cuenta la justificación (de líneas) de cada secuencia inicial, así como las posibles modificaciones que hayan sufrir las secuencias en memoria (después de enmascarar).

- **comando:** `salir`
  **salida en pantalla:**
  (no tiene salida por pantalla)
  **descripción:** Termina la ejecución de la aplicación.

### 2.2.2 Componente 2: compresión y decompresión de archivos `FASTA`

**Objetivo:** Utilizar el algoritmo de codificación de Huffman, y su correspondiente estructura jerárquica (árbol de Huffman), para comprimir y descomprimir archivos `FASTA`.

Un principio muy importante alrededor de la compresión de datos es codificar cada símbolo de un mensaje usando la cantidad mínima posible de bits. Por ejemplo, si un mensaje estuviera escrito en lenguaje `ASCII`, el cual tiene 256 símbolos diferentes, la cantidad mínima de bits por símbolo requerida sería de 8. Otro principio esencial es que, aquellos símbolos que aparecen más frecuentemente en un mensaje, sería útil codificarlos con menos bits que aquellos menos frecuentes, de tal forma que el mensaje comprimido ocupe el menor espacio posible.

La codificación de Huffman[1] tiene en cuenta los dos principios anteriores. Provee una forma de representar cada símbolo de un mensaje con la menor cantidad posible de bits, y al mismo tiempo permite codificar cada símbolo con una cantidad variable de bits, dependiendo de su frecuencia de ocurrencia en el mensaje. Para realizar el proceso de codificación y decodificación, se utiliza un árbol de Huffman. Éste es un árbol binario que representa una codificación de un conjunto de símbolos óptima: el símbolo que tenga una frecuencia más alta (el que más se repita) se representa con un número pequeño de bits. Un símbolo poco frecuente se representa con más bits. Un ejemplo de un árbol de Huffman se muestra en la Figura 1.

En este árbol, cada nodo **hoja** almacena un símbolo del lenguaje utilizado en un mensaje y la cantidad de veces que se encuentra dicho símbolo en el mensaje (valor entre paréntesis), al cual llamaremos "frecuencia".

[1] http://en.wikipedia.org/wiki/Huffman_coding

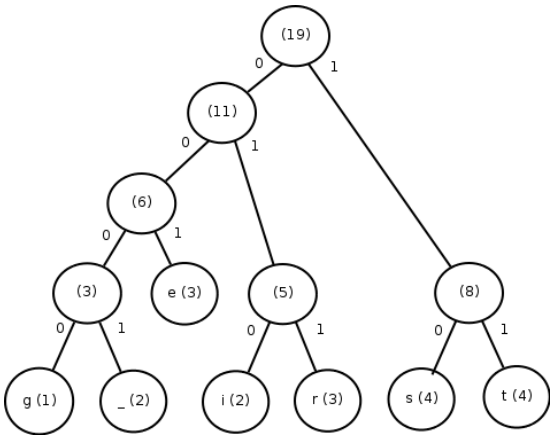



Figure 1: Árbol de Huffman.

Los **nodos intermedios y la raíz** no contienen símbolos, sino sólo un valor correspondiente a la suma de las frecuencias de sus nodos hijos. Note que los símbolos se encuentran exclusivamente en las hojas y cada símbolo aparece una sola vez en todo el árbol (no se repiten en el árbol).

Este árbol se utiliza para codificar y decodificar los símbolos de un mensaje. Cada arista entre un nodo padre y sus hijos se etiqueta con un "0" para el hijo izquierdo y con un "1" para el hijo derecho. Cada camino entre el nodo raíz y las hojas se puede representar como la concatenación de las etiquetas de las aristas que lo conforman. Esta representación corresponde a la codificación para cada uno de los símbolos. En este caso, por ejemplo, el símbolo que más se repite es la "t" (4 veces), y su codificación es "11", el cual corresponde a las etiquetas de las aristas del camino desde la raíz hasta el nodo con el símbolo "t"

Otros ejemplos de codificación son:

- La "s" se codifica como "10".
- El "_" se codifica como "0001".
- La "r" se codifica como "011".

Por ejemplo, la siguiente matriz de datos (imagen donde cada fila empieza y termina con el símbolo '+'):

```
+-----------------------------------+
+ ______________________________   +
+ < Soy capaz de hacer el codigo! > +
+ ------------------------------   +
+         \   ^__^                  +
+          \  (oo)\_______          +
+             (__)\       )\/\      +
+                 ||----w |         +
+                 ||     ||         +
+-----------------------------------+
```

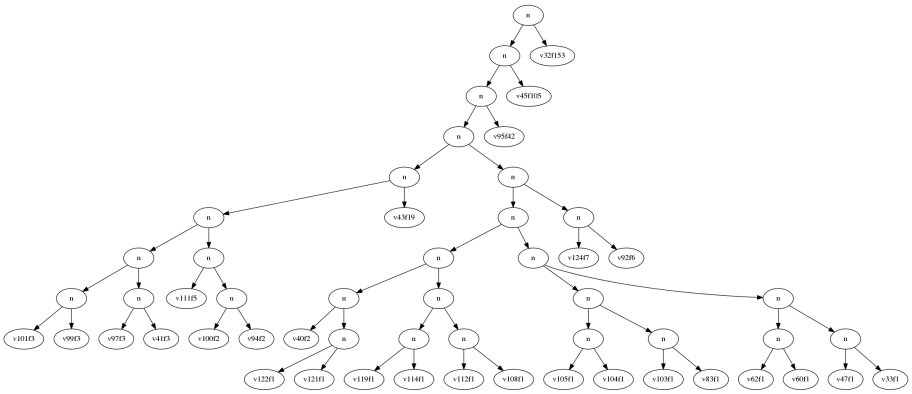
Tiene las siguientes características:

- Ancho de la imagen (W) = 36.
- Alto de la imagen (H) = 10.
- Valor (intensidad) máximo posible de la imagen (M) = 255.
- Los caracteres (que hacen las veces de intensidades) y sus frecuencias son:

| ASCII | carácter | frecuencia |
|---|---|---|
| 32 | | 153 |
| 33 | ! | 1 |
| 40 | ( | 22 |
| 41 | ) | 3 |
| 43 | + | 20 |
| 45 | - | 105 |
| 47 | / | 1 |
| 60 | < | 1 |
| 62 | > | 1 |
| 83 | S | 1 |
| 92 | \ | 6 |
| 94 | ^ | 2 |
| 95 | _ | 42 |
| 97 | a | 3 |

| ASCII | carácter | frecuencia |
|---|---|---|
| 99 | c | 3 |
| 100 | d | 2 |
| 101 | e | 3 |
| 103 | g | 1 |
| 104 | h | 1 |
| 105 | i | 1 |
| 108 | l | 1 |
| 111 | o | 5 |
| 112 | p | 1 |
| 114 | r | 1 |
| 119 | w | 1 |
| 121 | y | 1 |
| 122 | z | 1 |
| 124 | \| | 7 |

- El árbol asociado es (las hojas tiene asociado un valor "vXfY", donde "X" es el código del símbolo y "Y" es su frecuencia):



El objetivo del componente 2 es implementar un árbol de Huffman para codificar y decodificar archivos `FASTA`. En este contexto, el "mensaje" sería el conjunto de genomas contenidos en un archivo `FASTA` y los "símbolos" del mensaje serían los diferentes códigos de las bases del genoma. En el proceso de codificación, se tomarán las secuencias del genoma almacenadas en memoria y se generará un archivo binario con la correspondiente secuencia de "0"s y "1"s (separada en paquetes de 8 bits) que codifican cada secuencia. El archivo binario (con extensión de archivo `.fabin`) tiene la siguiente estructura:

$$n\ c_1\ f_1 \cdots c_n\ f_n\ n_s\ l_1 s_{11} \cdots\ s_{1l_1} \cdots\ l_{n_s} s_{n_s 1} \cdots\ s_{n_s l_{n_s}} w_1 x_1 binary\_code_1 \cdots w_{n_s} x_{n_s} binary\_code_{n_s}$$

donde:

- $n$ es un número entero de 2 bytes que representa la cantidad de bases diferentes presentes en las secuencia cargadas en ese momento en memoria.
- $c_i$ y $f_i$ son dos números enteros, de 1 y 8 bytes respectivamente, que representan un código de la base de genoma y su frecuencia asociada (cuántas veces aparece la base en todas las secuencias actualmente en memoria).
- $n_s$ es un número entero de 4 bytes que representa la cantidad de secuencias que hay actualmente en memoria.
- $l_i$ es un número entero de 2 bytes que representa el tamaño del nombre de la $i$-ésima secuencia.
- $s_{ij}$ es el caracter que se encuentra en la $j$-ésima posición del nombre de la $i$-ésima secuencia.
- $w_i$ es un número entero de 8 bytes que representa la longitud de la $i$-ésima secuencia.
- $x_i$ es un número entero de 2 bytes que representa la justificación (ancho de línea) de la $i$-ésima secuencia.
- $binary\_code_i$ es la secuencia binaria (unos y ceros) que representa la codificación de la $i$-ésima secuencia. Note que si la secuencia no es múltiplo de 8, se debe completar con los "0" necesarios al final de la secuencia.

En el proceso de decodificación, se tomará un archivo binario de extensión `.fabin` con las secuencias codificadas (siguiendo la estructura presentada anteriormente) y se decodificará, cargando en memoria las secuencias del genoma con sus nombres y bases constitutivas. De esta forma, el componente completo se implementará con los siguientes comandos:

- **comando:** `codificar` *nombre_archivo.fabin*
  **salida en pantalla:**
  (no hay secuencias cargadas) `No hay secuencias cargadas en memoria.`
  (mensaje de error) `No se pueden guardar las secuencias cargadas en` *nombre_archivo.fabin* `.`
  (codificación exitosa) `Secuencias codificadas y almacenadas en` *nombre_archivo.fabin* `.`
  **descripción:** El comando debe generar el archivo binario con la correspondiente codificación de Huffman en el formato descrito más arriba, almacenándolo en disco bajo el nombre *nombre_archivo.fabin*. Si no hay secuencias cargadas en memoria, o si el archivo no puede escribirse correctamente, el comando debe mostrar el mensaje correspondiente.

- **comando:** `decodificar` *nombre_archivo.fabin*
  **salida en pantalla:**
  (mensaje de error) `No se pueden cargar las secuencias desde` *nombre_archivo.fabin* `.`
  (decodificación exitosa) `Secuencias decodificadas desde` *nombre_archivo.fabin* `y cargadas en memoria.`
  **descripción:** El comando debe cargar en memoria las secuencias contenidas en el archivo binario *nombre_archivo.fabin*, que contiene una codificación Huffman de un conjunto de secuencias en el formato descrito más arriba. Si dentro de la misma sesión de trabajo ya se han cargado otros archivos de secuencias (usando el comando `cargar`), la información debe sobreescribirse en memoria, es decir, no se deben combinar informaciones de secuencias de diferentes archivos. Si por alguna razón no es posible cargar la información de codificación, o no es posible realizar el proceso de decodificación, el comando debe mostrar el mensaje de error.

### 2.2.3 Componente 3: Relaciones entre bases de las secuencias

**Objetivo:** A partir de las secuencias de un genoma, utilizar representaciones en grafos de las secuencias para obtener información adicional sobre la distribución de las bases.

En el archivo `FASTA` las secuencias se presentan en bloques de bases con una indentación específica. De esta forma, cada secuencia puede interpretarse como una matriz de bases, organizadas en filas y columnas. Cada fila corresponde a una línea del texto de la secuencia, y cada columna almacena una única base, de forma que hay tantas columnas como el número de indentación. Para el archivo `FASTA` de ejemplo presentado anteriormente, la secuencia `Full_SEQUENCE` puede verse como una matriz de 4 filas y 45 columnas, mientras que la secuencia `Incomplete_sequence` puede corresponder a una matriz de 6 filas y 63 columnas. Cada base se ubica entonces en una fila `i` y una columna `j` de la secuencia.

Esta configuración facilita la construcción de un grafo sobre cada secuencia. Para esto, cada una de las diferentes bases corresponde a un vértice del grafo, y estos vértices se interconectan teniendo en cuenta los 4 vecinos (superior, inferior, izquierdo, derecho) más cercanos a cada base. Para una base ubicada en la posición `[i,j]`, estos vecinos estarían ubicados así:

- superior: en la misma posición de la fila anterior, es decir `[i-1,j]`
- inferior: en la misma posición de la fila siguiente, es decir `[i+1,j]`
- izquierdo: sobre la misma fila, en la siguiente posición, es decir `[i,j+1]`
- derecho: sobre la misma fila, en la posición anterior, es decir `[i,j-1]`

La indexación en filas y columnas siempre arranca desde `0`, es decir, la esquina superior izquierda de la matriz corresponde a la posición `[0,0]`.La siguiente figura ilustra las conexiones en el grafo para una base dada, de acuerdo a la descripción anterior.

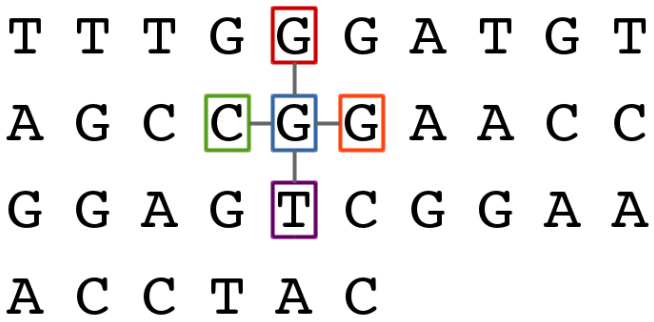



Figure 2: Ubicación de los vecinos para una base dada (azul): sobre la misma fila, a la derecha (naranja) y a la izquierda (verde); en la misma posición de la fila anterior (rojo), y en la misma posición de la fila siguiente (morado).

Interconectando todas las bases en la secuencia, se construye entonces el grafo que representa la secuencia, el cual se ilustra en la siguiente figura.

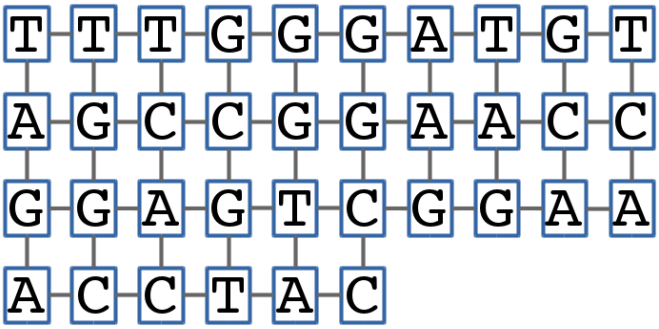



Figure 3: Construcción del grafo que representa una secuencia de bases.

Cada conexión entre bases vecinas tiene asignado un peso que se calcula teniendo en cuenta el código `ASCII` de la letra que representa cada base. Así, el peso de la arista que conecta el vértice en la posición `[i,j]` con el vértice en la posición `[x,y]` se calcula de acuerdo a la ecuación

$$C_{ij-xy} = \frac{1}{1 + |ASCII_{ij} - ASCII_{xy}|}$$

donde $ASCII_{ij}$ y $ASCII_{xy}$ corresponden al valor del código `ASCII` para la base en `[i,j]` y en `[x,y]`, respectivamente. Por ejemplo, para la base `'G'` (`ASCII` 71) en la Figura 2, el peso de su conexión con la base `'C'` (`ASCII` 67) ubicada a su izquierda sería:

$$C_{1,4-1,3} = \frac{1}{1 + |71 - 67|} = \frac{1}{1+4} = \frac{1}{5} = 0,2$$

Teniendo en cuenta el grafo construído sobre cada secuencia del genoma, los comandos que se deben desarrollar son:

- **comando:** `ruta_mas_corta` *`descripcion_secuencia i j x y`*
  **salida en pantalla:**
  (la secuencia no existe) `La secuencia` *`descripcion_secuencia`* `no existe.`
  (posición de base origen inválida) `La base en la posición [`*`i`*`,`*`j`*`] no existe.`
  (posición de base destino inválida) `La base en la posición [`*`x`*`,`*`y`*`] no existe.`
  (la secuencia existe) `Para la secuencia` *`descripcion_secuencia`*`, la ruta más corta entre la base` *`N`* `en [`*`i`*`,`*`j`*`] y la base` *`M`* `en [`*`x`*`,`*`y`*`] es: .... El costo total de la ruta es: ...`
  **descripción:** El comando debe imprimir en pantalla la secuencia de vértices (bases) del grafo que describen la ruta más corta entre la base ubicada en la posición `[`*`i`*`,`*`j`*`]` de la matriz de la secuencia

*descripcion_secuencia* y la base ubicada en la posición [*x*,*y*] de la misma matriz. Así mismo, debe imprimir el costo total de la ruta, teniendo en cuenta el peso que tiene cada conexión entre bases.

- **comando:** base_remota *descripcion_secuencia i j*
  **salida en pantalla:**
  (la secuencia no existe) La secuencia *descripcion_secuencia* no existe.
  (posición de base inválida) La base en la posición [*i*,*j*] no existe.
  (la base existe) Para la secuencia *descripcion_secuencia*, la base remota está ubicada en [*a*,*b*], y la ruta entre la base en [*i*,*j*] y la base remota en [*a*,*b*] es: .... El costo total de la ruta es: ...
  **descripción:** Para la base ubicada en la posición [*i*,*j*] de la matriz de la secuencia *descripcion_secuencia*, el comando busca la ubicación de la misma base (misma letra) más lejana dentro de la matriz. Para esta base remota, el comando debe imprimir en pantalla su ubicación, la secuencia de vértices (bases) que describen la ruta entre la base origen y la base remota, y el costo total de la ruta, teniendo en cuenta el peso que tiene cada conexión entre bases.

### 2.3 Interacción con el sistema

La interfaz del sistema a construir debe ser una consola interactiva donde los comandos correspondientes a los componentes serán tecleados por el usuario, de la misma forma que se trabaja en la terminal o consola del sistema operativo. El indicador de línea de comando debe ser el caracter $. Se debe incluir el comando `ayuda` para indicar una lista de los comandos disponibles en el momento. Así mismo, para cada comando se debe incluir una ayuda de uso que indique la forma correcta de hacer el llamado, es decir, el comando `ayuda comando` debe existir.

Cada comando debe presentar en pantalla los mensajes de resultado (éxito o error) especificados antes, además de otros mensajes necesarios que permitan al usuario saber, por un lado, cuando terminó el comando su procesamiento, y por el otro lado, el resultado de ese procesamiento. Los comandos de los diferentes componentes deben ensamblarse en un único sistema (es decir, funcionan todos dentro de un mismo programa, no como programas independientes por componentes).

## 3 Evaluación

Las entregas se harán a través de la correspondiente asignación de BrightSpace, hasta la media noche del día anterior al indicado para la sustentación de la entrega. Se debe entregar un archivo comprimido (único formato aceptado: `.zip`) que contenga dentro de un mismo directorio (sin estructura de carpetas interna) los documentos (único formato aceptado: `.pdf`) y el código fuente (`.h`, `.hxx`, `.cxx`, `.cpp`). Si la entrega contiene archivos en cualquier otro formato, o no cumple con las indicaciones anteriores, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de 0 (cero) sobre 5 (cinco).

### 3.1 (10%) Entrega 0: semana 3

La entrega inicial corresponderá únicamente a la interfaz de usuario necesaria para interactuar con el sistema. De esta forma, se verificará el indicador de línea del comando, y que el sistema realice la validación de los comandos permitidos y sus parámetros (en cantidad y formato, de ser necesario). Revisar en particular el numeral 2.3.

### 3.2 (30%) Entrega 1: semana 6

Componente 1 completo y funcional, implementado a partir de estructuras lineales. Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (40%) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares (comandos). Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones (comandos) principales. El diseño debe ser totalmente coherente con la implementación en código fuente.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `enmascarar`. El plan de pruebas debe ser totalmente coherente con el diseño y la implementación en código fuente.

- (30%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0 como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando. La implementación en código fuente debe ser totalmente coherente con el diseño presentado.
- (10%) Sustentación (individual) con participación de todos los miembros del grupo.

## 3.3 (30%) Entrega 2: semana 12

Componentes 1 y 2 completos y funcionales, implementados a partir de estructuras lineales y jerárquicas (árboles). Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (10%) Completar la funcionalidad que aún no haya sido desarrollada de la primera entrega. Se debe generar un acta de evaluación de la entrega anterior (incluirla al principio del documento de diseño) que detalle los comentarios textuales (literales) hechos a la entrega y la forma en la que se corrigieron, arreglaron o completaron para la segunda entrega.
- (25%) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares (comandos). Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones (comandos) principales. El diseño debe ser totalmente coherente con la implementación en código fuente.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `decodificar`. El plan de pruebas debe ser totalmente coherente con el diseño y la implementación en código fuente.
- (25%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0 como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando. La implementación en código fuente debe ser totalmente coherente con el diseño presentado.
- (20%) Sustentación (individual) con participación de todos los miembros del grupo.

## 3.4 (30%) Entrega 3: semana 18

Componentes 1, 2 y 3 completos y funcionales, implementados a prtir de estructuras lineales, jerárquicas (árboles) y no lineales (grafos). Esta entrega tendrá una sustentación (del proyecto completo) durante el último día de clase de la semana 18, y se compone de:

- (10%) Completar la funcionalidad que aún no haya sido desarrollada de la primera y segunda entregas. Se debe generar un acta de evaluación de las entregas anteriores (incluirla al principio del documento de diseño) que detalle los comentarios textuales (literales) hechos a las entregas y la forma en la que se corrigieron, arreglaron o completaron para la tercera entrega.
- (25%) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares (comandos). Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones (comandos) principales. El diseño debe ser totalmente coherente con la implementación en código fuente.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `base_remota`. El plan de pruebas debe ser totalmente coherente con el diseño y la implementación en código fuente.
- (25%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0 como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando. La implementación en código fuente debe ser totalmente coherente con el diseño presentado.
- (20%) Sustentación (individual) con participación de todos los miembros del grupo.

## 3.5 Recomendaciones: Uso de herramientas de IA generativa

Para la implementación de los componentes de este proyecto, se espera que los estudiantes puedan evidenciar el uso y aplicación de los conceptos y técnicas vistos en clase de forma propia. Sin embargo, existe la posibilidad de que puedan apoyarse en herramientas de IA generativa, bajo las siguientes condiciones:

1. Las partes creadas con apoyo de herramientas de IA generativa deben identificarse claramente a través de comentarios, de forma que pueda saberse claramente cuáles líneas de código o párrafos redactados han sido generados automáticamente, incluyendo además información sobre el prompt utillizado en la IA generativa para obtenerlas. En casos en que se utilice un esqueleto o borrador base dado por la IA generativa que luego sea modificado o editado, esta situación también debería comentarse claramente.

2. La cantidad de contenido generado con apoyo de las herramientas de IA generativa no debería superar un tercio del contenido total generado en cada entrega, para garantizar la apropiación y aplicación de los conceptos por parte de los estudiantes.