Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Proyecto del curso, 2020-10

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

2.1. Formato FASTA

Los archivos en formato FASTA (extensión .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. Las siguientes líneas (hasta la siguiente secuencia o hasta el final del archivo) contienen los datos que describen el código genético. Estas líneas están justificadas y todas tienen un ancho constante, 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 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

Cuadro 1: Relación de cada código del formato FASTA con su significado biológico (tomada de http://www.ncbi.nlm.nih.gov/blast/fasta.shtml).

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 con los siguientes comandos:

- **comando:** `cargar` *nombre_archivo*
  salida en pantalla:
  (Archivo vacío) "*nombre_archivo*" no contiene ninguna secuencia.
  (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*.

- **comando:** `conteo`
  salida en pantalla:
  (No hay secuencias cargadas) No hay secuencias cargadas en memoria.
  (Una sola secuencia) 1 secuencia en memoria.
  (Varias secuencias) *n* secuencias en memoria.
  descripción: Imprime por pantalla la cantidad de secuencias cargadas en memoria.

- **comando:** `listar_secuencias`
  salida en pantalla:
  (No hay secuencias cargadas) No hay secuencias cargadas en memoria.
  (Secuencia completa de archivo no vacío) Secuencia "*descripcion_secuencia*" contiene *b* bases.
  (Secuencia incompleta de archivo no vacío) Secuencia "*descripcion_secuencia*" contiene al menos *b* bases.
  descripción: Imprime en n líneas (una para secuencia) la información básica (cantidad de bases) de cada secuencia. Si la secuencia es completa (i.e. no tiene el código '-') imprime el segundo mensaje, si no, el tercero.

- **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.

- **comando:** `es_subsecuencia` *secuencia*
  salida en pantalla:
  (No hay secuencias cargadas) `No hay secuencias cargadas en memoria.`
  (La secuencia no existe) `La secuencia dada no existe.`
  (Varias secuencias) `La secuencia dada se repite` *s* `veces.`
  descripción: Determina si una secuencia, dada por el usuario, existe dentro de las secuencias cargadas. Si es así, determina la cantidad de veces en las que esta secuencia dada se repite.

- **comando:** `enmascarar` *secuencia*
  salida en pantalla:
  (No hay secuencias cargadas) `No hay secuencias cargadas en memoria.`
  (No se enmascararon subsecuencias) `La secuencia dada no existe, por tanto no se enmascara nada.`
  (Una subsecuencia enmascarada) `1 secuencia ha sido enmascarada.`
  (Varias subsecuencias esmascaradas) *s* `secuencias han sido enmascaradas.`
  descripción: Enmascara una secuencia dada por el usuario, si existe. Los elementos que pertenecen a la subsecuencia se enmascaran cambiando cada código por el código 'X'.

- **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) del archivo inicial.

- **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 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 (i.e. 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". 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:

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

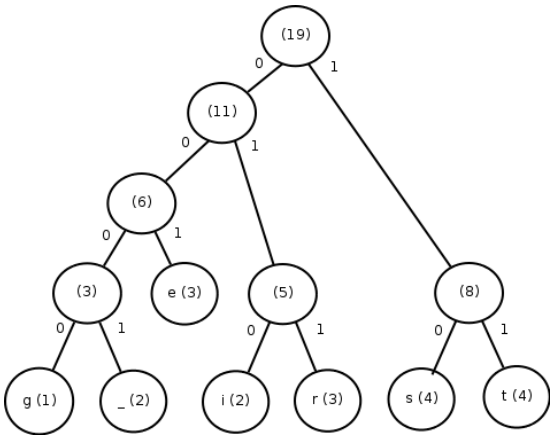


Figura 1: Árbol de Huffman.

- 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 '+'):

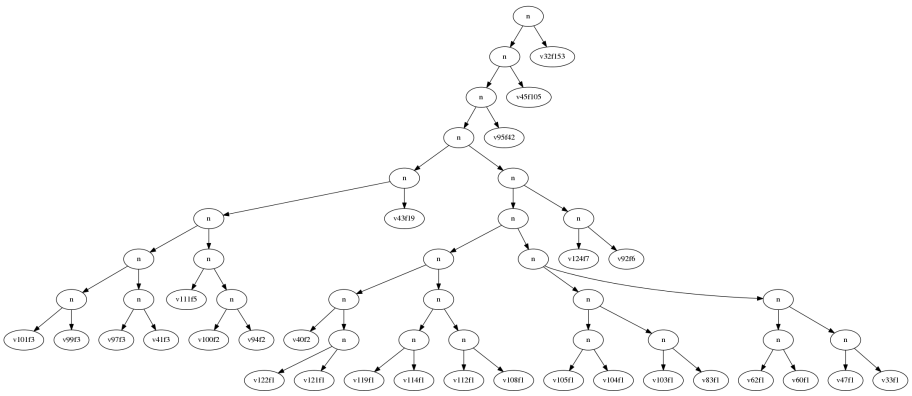
```
+-----------------------------------+
+ _______________________________   +
+ < Soy capaz de hacer el codigo! > +
+ -------------------------------   +
+         \   ^__^                  +
+          \  (oo)\_______          +
+             (__)\       )\/\      +
+                 ||----w |   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

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 = 32, caracter = ' ', frecuencia = 153`
  - `ASCII = 33, caracter = '!', frecuencia = 1`
  - `ASCII = 40, caracter = '(', frecuencia = 2`
  - `ASCII = 41, caracter = ')', frecuencia = 3`
  - `ASCII = 43, caracter = '+', frecuencia = 20`
  - `ASCII = 45, caracter = '-', frecuencia = 105`
  - `ASCII = 47, caracter = '/', frecuencia = 1`
  - `ASCII = 60, caracter = '<', frecuencia = 1`
  - `ASCII = 62, caracter = '>', frecuencia = 1`
  - `ASCII = 83, caracter = 'S', frecuencia = 1`
  - `ASCII = 92, caracter = '\', frecuencia = 6`
  - `ASCII = 94, caracter = '^', frecuencia = 2`
  - `ASCII = 95, caracter = '_', frecuencia = 42`

- `ASCII = 97, caracter = 'a', frecuencia = 3`
- `ASCII = 99, caracter = 'c', frecuencia = 3`
- `ASCII = 100, caracter = 'd', frecuencia = 2`
- `ASCII = 101, caracter = 'e', frecuencia = 3`
- `ASCII = 103, caracter = 'g', frecuencia = 1`
- `ASCII = 104, caracter = 'h', frecuencia = 1`
- `ASCII = 105, caracter = 'i', frecuencia = 1`
- `ASCII = 108, caracter = 'l', frecuencia = 1`
- `ASCII = 111, caracter = 'o', frecuencia = 5`
- `ASCII = 112, caracter = 'p', frecuencia = 1`
- `ASCII = 114, caracter = 'r', frecuencia = 1`
- `ASCII = 119, caracter = 'w', frecuencia = 1`
- `ASCII = 121, caracter = 'y', frecuencia = 1`
- `ASCII = 122, caracter = 'z', frecuencia = 1`
- `ASCII = 124, caracter = '|', frecuencia = 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. Dado que la secuencia para codificar un mensaje completo es una secuencia de "0"s y "1"s, este mensaje se puede separar en paquetes de 8 bits y guardar en un archivo binario. El archivo binario (de extension "fabin", para el presente proyecto) 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 entero de 1 y 8 bytes, respectivamente, que representan un código de la base de genoma y su frecuencia asociada (i.e. cuántas veces aparece en todas las secuencias.
- n_s es un número entero de 4 bytes que representa la cantidad de secuencias que hay en el archivo.
- 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 un número entero de 8 bytes que representa la longitud de la i-ésima secuencia.
- x_i un número entero de 2 bytes que representa la indentación de la i-ésima secuencia.

- $binary\_code_i$ es la secuencia binaria que representa la i-ésima secuencia. Note que si la secuencia no es múltiplo de 8, se debe completar con los "0" necesarios.

Los comandos que usted debe desarrollar son:

- **comando:** `codificar` *`nombre_archivo.fabin`*
  salida en pantalla:
  (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`*.

- **comando:** `decodificar` *`nombre_archivo.fabin`*
  salida en pantalla:
  (mensaje de error) `No se pueden cargar las secuencias en` *`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.

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 siguiente figura ilustra las conexiones en el grafo para una base dada, de acuerdo a la descripción anterior.

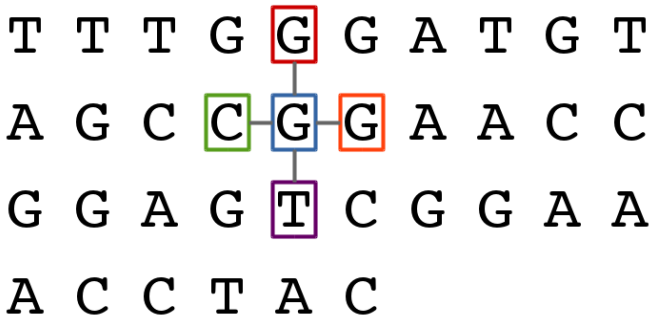


Figura 2: Ubicación de los vecinos para una base dada (azul): sobre la misma fila, a la derecha (naranja) y a la izquierda (verde); en la misma posición de la fila anterior (rojo), y en la misma posición de la fila siguiente (morado).

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

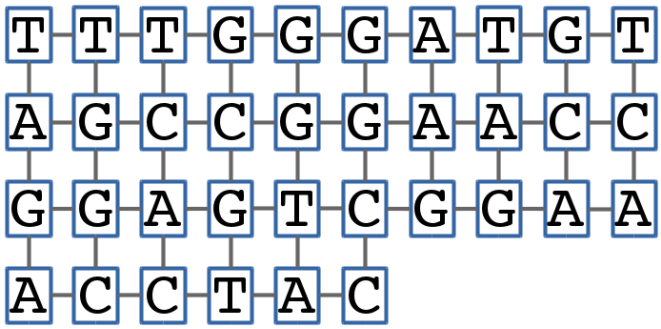


Figura 3: Construcción del grafo que representa una secuencia de bases.

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.

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 dada 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) `La ruta más corta entre la base en [`*`i,j`*`] y la base en [`*`x,y`*`] es: .... El costo total de la ruta es: ...`
  descripción: El comando debe imprimir en pantalla la secuencia de vértices 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 dada no existe.`
  (Posición de base inválida) `La base en la posición [`*`i,j`*`] no existe.`
  (La base existe) `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 que describen la ruta entre la base origen y la base remota, y el costo total de la ruta.

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 mensajes de éxito o error 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 en la correspondiente actividad de Uvirtual, hasta la media noche del día anterior al indicado para la entrega. Se debe entregar un archivo comprimido (únicos formatos aceptados: .zip, .tar, .tar.gz, .tar.bz2, .tgz) que contenga dentro de un mismo directorio (sin estructura de carpetas interna) los documentos (único formato aceptado: .pdf) y el código fuente (.h, .hxx, .hpp, .c, .cxx, .cpp). Si la entrega contiene archivos en cualquier otro formato, 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: viernes 21 de febrero de 2020

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.

3.2. (30 %) Entrega 1: miércoles 11 de marzo de 2020

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

- (30 %) 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. 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 principales.
- (55 %) Código fuente compilable en el compilador gnu-g++ v4.0.0 (mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (15 %) Sustentación con participación de todos los miembros del grupo.

3.3. (30 %) Entrega 2: miércoles 29 de abril de 2020

Componentes 1 y 2 completos y funcionales. Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (15 %) Completar la funcionalidad que aún no haya sido desarrollada de la primera entrega. Se debe generar un acta de evaluación de la entrega anterior que detalle los elementos faltantes y cómo se 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. 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 principales.
- (45 %) Código fuente compilable en el compilador gnu-g++ v4.0.0 (mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (15 %) Sustentación con participación de todos los miembros del grupo.

3.4. (30 %) Entrega 3: viernes 29 de mayo de 2020

Componentes 1, 2 y 3 completos y funcionales. Esta entrega tendrá una sustentación (del proyecto completo) durante el viernes 29 de mayo de 2020 entre las 8am y las 12m, y se compone de:

- (15 %) Completar la funcionalidad que aún no haya sido desarrollada de la primera y segunda entregas. Se debe generar un acta de evaluación de la entrega anterior que detalle los elementos faltantes y cómo se 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. 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 principales.
- (45 %) Código fuente compilable en el compilador gnu-g++ v4.0.0 (mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (15 %) Sustentación con participación de todos los miembros del grupo.