Proyecto: Estructuras de Datos
(Componente 2)

Grupo #6

Jose David Calderon
Andrés Malagón
Julián Cárdenas

Estructuras de Datos
28 de abril de 2021

Bogotá
Pontificia Universidad Javeriana

Objetivo: El objetivo del proyecto del semestre es construir un sistema que permita algunas manipulaciones sencillas sobre archivos, con formato FASTA y FABIN, que contienen códigos genéticos y estructura binaria.

Descripción general: La idea principal de esta funcionalidad está basada dentro de dos grandes: codificación y decodificación. 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. Es por esto, que a través de este proyecto implementaremos la posibilidad de ejecutar estas dos funciones por medio de un archivo "fabin" (archivo binario), que permite representar la información de una manera diferente a la convencional.

Se presentará la documentación del código asignado para la segunda entrega del proyecto la cual se encarga del segundo componente de funciones las cuales a tintes generales se encargan de todas las acciones de cifrado con los archivos fabin, ya que en estos se encuentra la debida secuencia de las diferentes bases nitrogenadas que conforman un genoma, en este caso específico el genoma humano por lo consiguiente se explicara en tres apartados diferentes el funcionamiento del código, las cuales son:

a. Descripción de entradas, salidas y condiciones para la función principal y las operaciones auxiliares
b. los TAD´s utilizados, debe proveerse una plantilla que lo describa brevemente y la lista de sus operaciones
c. Esquemáticos (diagramas, gráficos, dibujos) que ilustren el funcionamiento general de las operaciones principales.

En consecuencia, con estos tres apartadas daremos la mayor claridad al código posible para que el usuario final posea de una debida información y no se pierda en el manejo del software, esto con el fin de establecer la mejor experiencia hombre-máquina.

2. Entradas

Codificar: Archivo fabin.

Decodificar: Archivo fabin.

3. Salidas:

Codificar: Mensaje de éxito/error.

Decodificar: Mensaje de éxito/error.

4. TAD´s

Se creó un TAD para el uso del mecanismo inicial del código el cual se desarrolló en el archivo principal, por esto se decidió llamarlo "*TAD main*" ya que este contiene las funciones principales del componente, este se sub divide en dos archivos funcionales:

i. Arbol_Huffman.h:
   En este apartado se encuentra el enunciado de todas las funciones necesarias para el componente 1, las cuales están nombradas en la gráfica que se mostrara en este mismo apartado.
ii. Arbol_Huffman.hxx:
   Este archivo contempla todo el código funcional correspondiente para la funcionalidad completa del archivo anterior, cuenta con los enunciados de las funciones y sus respectivos valores de entrada, si los necesita, el código desarrollado y los retornos pertinentes ya sea porque debe retornar una variable o sobrescribirla.
iii. Funciones.h
En este espacio se encuentra enunciados de funciones adicionales a las necesarias para ejecutar las funciones principales del proyecto.
Iv. Funciones.hxx
En este espacio se encuentran desarrolladas dichas funciones.

Se decidió la descomposición del código de esta forma para una mayor organización en términos de trabajo y no sobrecargar al usuario con un solo archivo, esto facilita la integración de partes y será una metodología usada en próximas entregas.

En consecuencia, a lo explicado con anterioridad, se presentan 8 diferentes funciones las cuales son encargadas de la carga del archivo Fasta y la estructuración de los mismos a través de un formato "Fabin", su conteo para saber en específico la cantidad de secuencias en él.

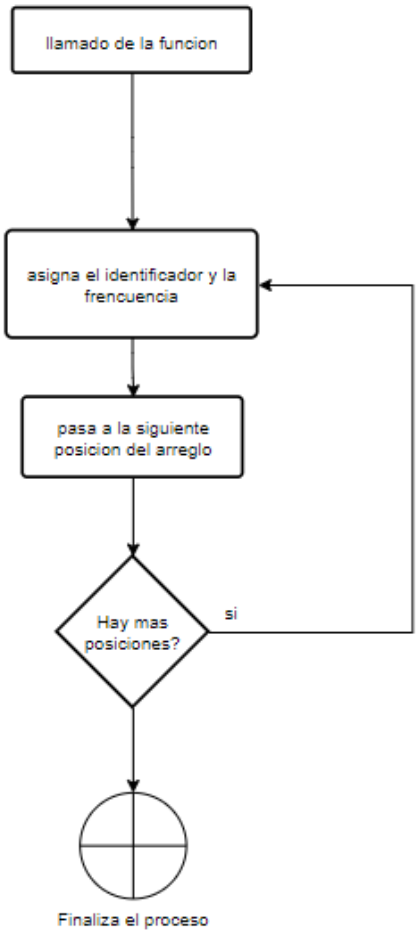
5. Funcionamiento y esquemas

Función Principal: ***main***

Dentro de la función main se encuentra toda la estructura general de nuestro código, se puede decir que define la estructura visual del programa y está en constante comunicación con el usuario a la espera de que este ingrese algún comando que llame a su funcionalidad definida. Una vez el usuario ingresa correctamente su comando, esta evalúa a través de condicionales que función debe llamar, ejecuta la operación y vuelve a la espera. Todo este proceso iterativo termina al recibir el comando Salir, que cancela cualquier operación y termina el programa inmediatamente.

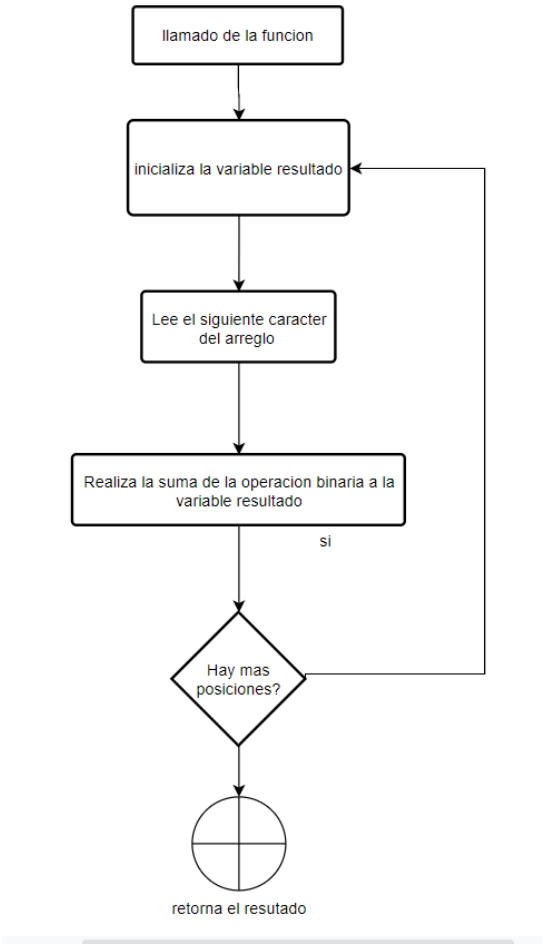
Función: ***create_node_array***

Esta función inicializa el vector de nodos creado en un principio dentro de la clase ArbolHuffman. Simplemente es llamada con el propósito de inicializar valores a cada uno de los 128 espacios dentro del arreglo y prevenir excepciones futuras mientras este es manipulado.



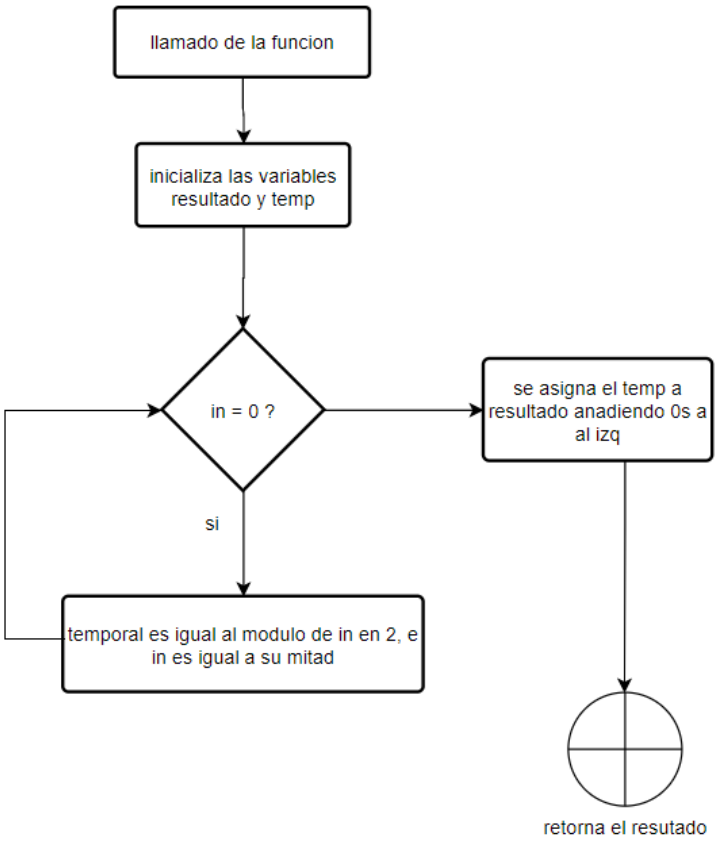
Función: ***Binary_to_decimal:***

La función recibe como parámetro un Sting con el código binario que será convertido a un número decimal funcional con el propósito de hacer operaciones con el posteriormente.



Función: ***Decimal_to_Binary:***

La función decimal a binario realiza básicamente lo inverso a lo descrito en la función anterior, su propósito es recibir un numero en formato decimal y transformarlo a una notación base 2 binaria, así mismo se verifica la añadidura de 0s para que el resultado sea una longitud fija de 8, al final. Retorna el resultado en número entero (int)

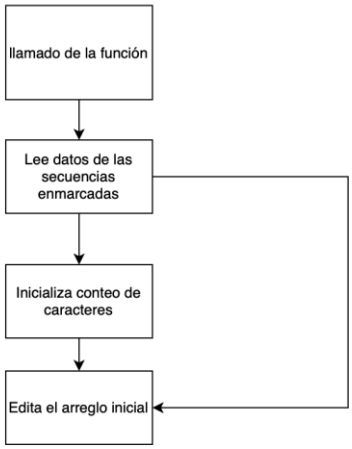


Función: ***Create huffman tree***

Esta función utiliza varias de las variables declaradas en la clase misma, básicamente crea un nodo raíz desde el cual se desprenderá el resto de los nodos ya ramas para la posterior codificación, es de tipo void pues no tiene retorno, pero guarda su nodo raíz en la lista temp.

Función: ***Create pq***

Esta función es la encargada de leer los datos de las secuencias enmarcadas en los archivos, y hacer el respectivo conteo de los caracteres o códigos de genoma que están asociados, una vez leídos los códigos procede a editar el arreglo de caracteres que se tenía inicialmente.



Funcion: ***Recreate huffman tree***

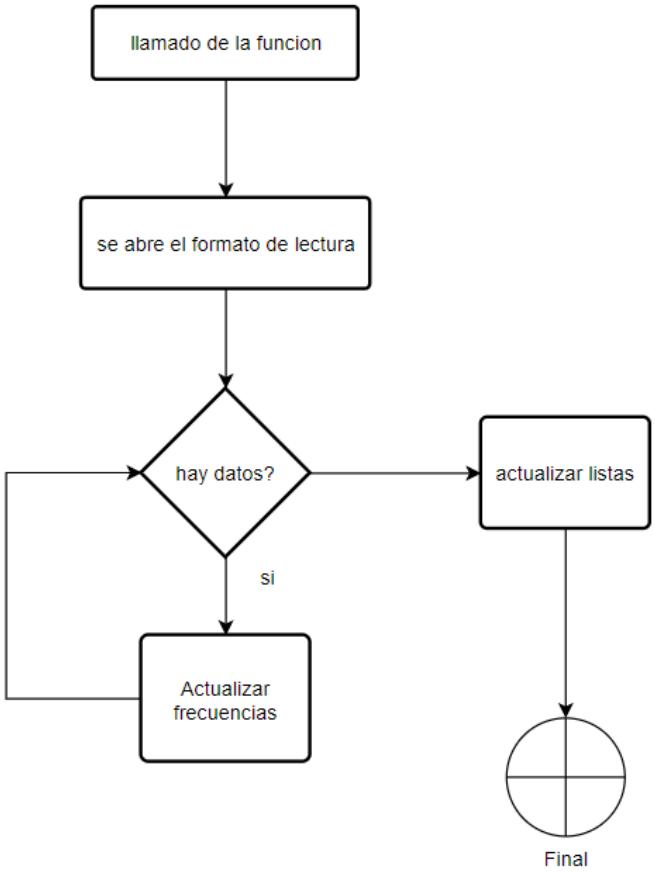
Esta función básicamente crea una versión del árbol de Hoffman y es estrictamente necesaria para su impresión en la función codificar o coding save, pues realiza todas las operaciones con caracteres y elementos binarios que permiten ahorrar el espacio en la comprensión.

Funcion ***Coding save***

Esta función es la encargada de codificar las estructuras ya formadas con la información de las secuencias cargadas en memoria, y utiliza el nombre ingresado por el usuario para identificar en que tipo de archivo guardara la información y de que forma.

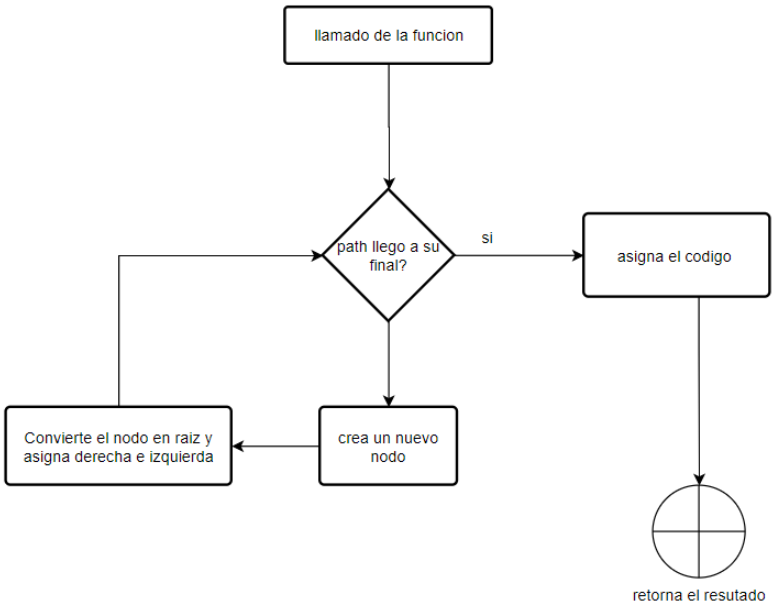
Función: ***ArbolHuffman***

Esta función inicializa los valores de la clase arbolHuffman nombreFin y nombreFout además de llamar a la función anteriormente llamada createnodearray, esta es de las primeras funciones llamadas con el fin de inicializar directamente los valores del arreglo de caracteres a comprimir.



Funcion: ***Build_Tree***

Esta función como su nombre lo indica se encarga de construir el árbol, recibe como parámetro un carácter código y una cadena denominada path o ruta. Como ya se sabe en el algoritmo de huffman las ramas son las que determina el código, por lo que se basa en este principio para generar ramas dependiendo de su posición.



función ***Decoding save***:

La función decoding save se utiliza normalmente después de utilizar la función anterior coding save, esta pretende desempacar del archivo comprimido todas las secuencias con su nombre según un formato dado, y asignar según corresponda a las variables en memoria del programa.

Función ***imprimir ayuda***

Esta función pretende dar al usuario una ayuda respecto a las funcionalidades que tiene el programa , y demás argumentos que tiene cada Código.

Función: ***salir***

Esta funcionalidad es la más sencilla de todas, en el main, una vez se detecta que el comando o cadena ingresado por el usuario es "**salir**", esta llama a la función ***exit***, y se cerrara el programa inmediatamente.