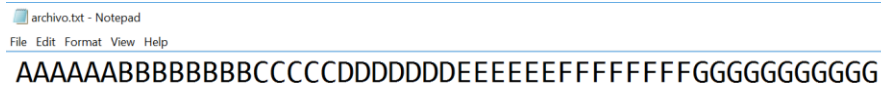


# ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

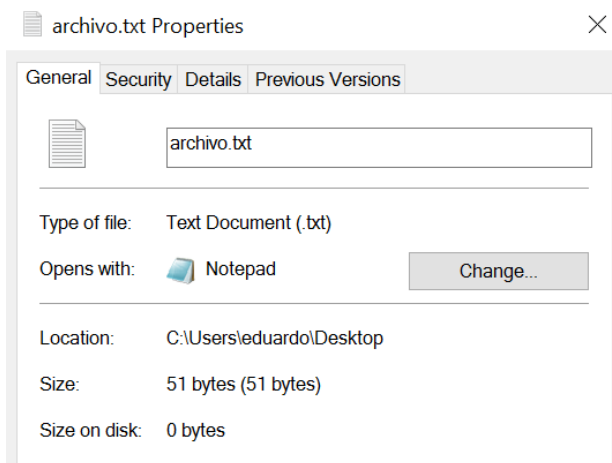
## FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN PROYECTO 3 - SEGUNDO PARCIAL ESTRUCTURAS DE DATOS

La propuesta del proyecto consiste en la implementación del Arbol Huffman para la construcción del código de Huffman y aplicarlo en la compresión de archivos.

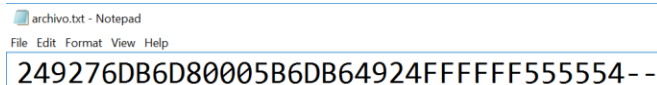
Por ejemplo, usted puede tener el siguiente archivo en su computador con un peso de 51 bytes:



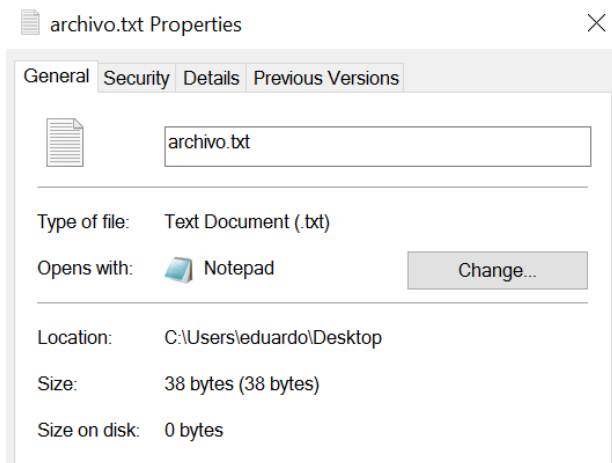
archivo.txt - Notepad  
File Edit Format View Help  
AAAAAABBBBBBBBCCCCDDDDDDDEEEEEFFFFFFFFFFGGGGGGGGGGG



Mediante la aplicación del proyecto, usted podrá comprimir el archivo a un peso de 38 bytes.



archivo.txt - Notepad  
File Edit Format View Help  
249276DB6D80005B6DB64924FFFFFFF555554--



En la implementación del proyecto, usted deberá implementar los siguientes TDAs en su proyecto:

### **TDA Util**

El TDA Util será el encargado de tener las funcionalidades complementarias necesarias para la implementación del proyecto:

Los siguientes métodos deben estar estáticos y deben estar implementados en el TDA Util:

#### **leerTexto**

Esta función recibe el nombre del archivo, procede a leer el texto que tiene un archivo y retorna una cadena de caracteres.

String leerTexto (String nombreArchivo)

Ejemplo:

AAAAAABBBBBBBBCCCCDDDDDDDEEEEEEEEEFFFFFGGGGGGGGGGG

#### **calcularFrecuencias**

Esta función recibe el texto que se leyó desde el archivo y procede a calcular la frecuencia de cada uno de los caracteres del texto, finalmente retorna un mapa con la información de la frecuencia de cada uno de los caracteres que contiene el texto.

HashMap<String,Integer> calcularFrecuencias (String texto)

Ejemplo:

Carácter	Frecuencia
A	6
B	8
C	5
D	7
E	6
F	8
G	11

#### **binarioHexadecimal**

Esta función recibe una cadena de caracteres que representa el código de Huffman en formato de 0 y 1 (binario). La función debe segmentar el binario en bloques de 4 dígitos para realizar la transformación a hexadecimal de cada uno de estos bloques. En caso de que el último bloque no se pudo formar 4 dígitos, se lo puede completar con ceros para cumplir con el tamaño de los bloques (4 dígitos).

Si existió la necesidad de agregar ceros al último bloque, usted puede agregar una cantidad de signos – (menos) al final de la cadena resultante que representen la cantidad de ceros que necesitó agregar al último bloque, esto es con la finalidad de poder realizar el proceso de decodificación sin errores.

Ejemplo:

String binarioHexadecimal (String binario)

Binario:

0010 0100 1001 0010 0111 0110 1101 1011 0110 1101 1000 0000 0000 0000 0101 1011 0110  
1101 1011 0110 0100 1001 0010 0100 1111 1111 1111 1111 1111 0101 0101 0101 0101  
0101 01    <- Al bloque final se lo debe completar con 2 ceros en este caso en particular.

Hexadecimal:

2 4 9 2 7 6 D B 6 D 8 0 0 0 5 B 6 D B 6 4 9 2 4 F F F F F 5 5 5 5 4 - -

### **hexadecimalBinario**

Esta función recibe una cadena de caracteres que representa la cadena codificada en hexadecimal y procede a transformarla en formato binario. En caso de que existan signos menos al final, se deberá remover la cantidad de ceros equivalente a los signos menos que aparecen en la cadena de caracteres.

Hexadecimal:

2 4 9 2 7 6 D B 6 D 8 0 0 0 5 B 6 D B 6 4 9 2 4 F F F F F 5 5 5 5 4 - -

Binario:

0010 0100 1001 0010 0111 0110 1101 1011 0110 1101 1000 0000 0000 0000 0101 1011 0110  
1101 1011 0110 0100 1001 0010 0100 1111 1111 1111 1111 1111 0101 0101 0101 0101  
0101 01

### **guardarTexto**

Esta función recibe el nombre del archivo, el nuevo texto que se va a guardar en el archivo y un mapa con los códigos para cada carácter. La función procede a almacenar el nuevo texto en el archivo y genera un archivo adicional con la tabla de códigos, esto es con la finalidad de posteriormente poder decodificar el archivo en función del código asignado a cada letra. El nombre del archivo adicional puede ser definido de la siguiente manera: nombreArchivo+” \_compress.txt”.

void guardarTexto (String nombreArchivo, String texto, HashMap<String,String> mapa)

### leerMapa

Esta función lee el mapa de códigos para cada carácter desde un archivo y lo retorna en un mapa.

HashMap<String,String> leerMapa (String nombreArchivo)

### TDA ArbolHuffman

Este TDA es la representación del Árbol Huffman, el cual tiene el siguiente atributo:

- Nodo raíz

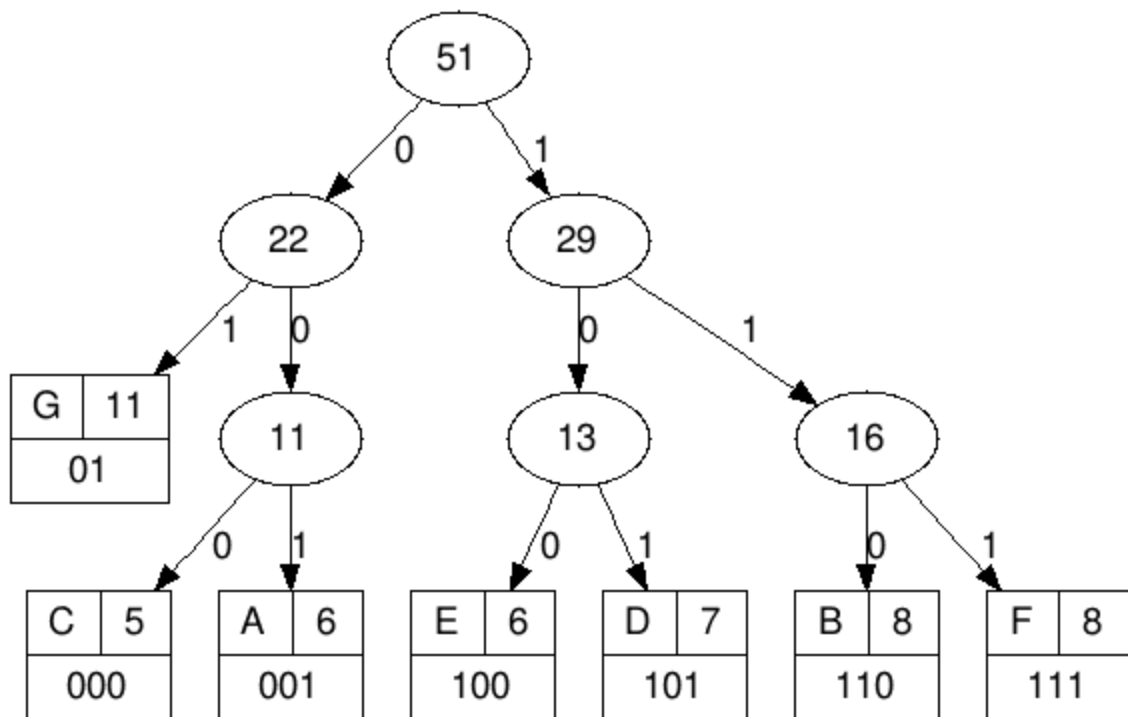
Los siguientes métodos deben ser implementados en el TDA ArbolHuffman:

### calcularArbol

Este método recibe un mapa con las frecuencias de los caracteres y procede a crear el Árbol de Huffman basado en las frecuencias de los caracteres que componen el texto del archivo.

calcularArbol (HashMap<String,Integer> mapa)

Ejemplo:



### calcularCodigos

Este método calcula los códigos para cada carácter a partir del recorrido del árbol de Huffman y retorna un mapa con esta información.

HashMap<String,String> calcularCodigos ()

Ejemplo:

Carácter	Frecuencia
A	001
B	110
C	000
D	101
E	100
F	111
G	01

### codificar

Esta función es estática, la cual recibe el texto que se leyó desde el archivo y el mapa que contiene los códigos respectivos para cada carácter. La función retorna la conformación del código de Huffman.

String codificar (String texto, HashMap<String,String> mapa)

Ejemplo:

00100100100100100111011011011011011011000000000000000010110110110110110110  
110010010010010010011111111111111111111111110101010101010101010101

### decodificar

Esta función es estática, la cual recibe el código de huffman y el mapa que contiene los códigos respectivos para cada carácter. La función retorna el texto original.

String decodificar (String texto, HashMap<String,String> mapa)

Ejemplo:

AAAAAABBBBBBBBCCCCDDDDDDDEEEEEEEEEEEEEFGGGGGGGGGGG

## **Main**

Finalmente, usted deberá crear un programa que presente un menú de opciones para que el usuario pueda elegir la opción de comprimir o descomprimir un archivo, posteriormente, el usuario ingresará el nombre del archivo.

### Menú de Opciones

- 1) Comprimir
- 2) Descomprimir