UNIVERSIDAD EAFIT

Escuela de Ciencias Aplicadas e Ingeniería

Estructuras de datos y algoritmos

Código de Clase: 0935

PROYECTO FINAL

Compresor de Texto Huffman

Integrantes:

- Gisel Lorena Jaramillo
- Juan Fernando Hernández Arenas
- Santiago Meneses Carvajal

Docente:

German Adolfo Montoya

Noviembre 14 2025

Medellín, Colombia

Introducción

Este proyecto propone un compresor y un descompresor de texto a partir del algoritmo de Huffman, aplicado a conseguir una reducción de archivos de texto codificando el carácter en función de su frecuencia de aparición. El proyecto se implementó en C++, en donde únicamente se hizo uso de librerías estándar de este mismo lenguaje, permitiendo la compresión y la descompresión de un archivo de texto de forma reversible, garantizando así la funcionalidad del código.

Objetivos

- Implementar un algoritmo de compresión basado en árboles binarios del algoritmo de Huffman.
- Asegurar la correcta reconstrucción del archivo original en el proceso de descompresión.

Explicación del algoritmo de Huffman

El algoritmo de Huffman es un método de compresión sin pérdida que asigna códigos binarios de longitud variable a cada carácter según su frecuencia. Los caracteres más frecuentes reciben códigos más cortos, y los menos frecuentes, más largos. El proceso se divide en tres etapas:

1. Cálculo de frecuencias: Se cuenta cuántas veces aparece cada carácter en el texto.

2. Construcción del árbol: Se crea un nodo por carácter, y se combinan los de menor frecuencia hasta obtener un árbol binario completo.

3. Generación de códigos: Se recorre el árbol desde la raíz, asignando '0' a la rama izquierda y '1' a la derecha.

4. Codificación y escritura: Se codifica el texto según los códigos generados y se guarda el árbol y los bits comprimidos en un archivo.

5. Descompresión: Se reconstruye el árbol y se recorre con los bits para recuperar los caracteres originales.

Desarrollo del código del proyecto

El código está dividido en tres módulos principales:

- **Huffman.cpp / Huffman.h:** Estos archivos implementan toda la lógica principal del algoritmo de Huffman. Contienen la estructura del nodo del árbol, el mapa de códigos y todas las funciones

necesarias para realizar la compresión y descomposición de archivos. Entre las tareas más importantes que se realizan están:

Pasos principales del proceso de compresión:

- **Calcular la frecuencia de cada carácter:** del archivo original leyendo todo su contenido y contabilizando cuántas veces se repite cada uno.
- **Construir el árbol de Huffman:** usando una **cola de prioridad** (priority_queue), que garantiza que siempre se unan los nodos con menor frecuencia:
  - Un nodo izquierdo representa un **0**
  - Un nodo derecho representa un **1**
- **Generar los códigos binarios:** recorriendo el árbol y asignando una cadena de bits única a cada carácter:
  - Los caracteres más frecuentes tienen códigos más cortos
  - Los menos frecuentes, códigos más largos. Lo que produce la compresión real.
- **Escribir el árbol completo:** dentro del archivo comprimido en **formato preorden**, utilizando:
  - '1' para indicar un nodo hoja y luego guardar el carácter
  - '0' para indicar un nodo interno
    Esto permite reconstruirlo exactamente durante la descompresión.
- **Codificar el texto con bits** y guardarlo en el archivo comprimido utilizando la clase BitWriter para trabajar a nivel de bits y no de caracteres.

Pasos del proceso de descompresión:

- **Leer el árbol almacenado:** usando la función readTree() para reconstruirlo exactamente igual al original.
- **Leer bit por bit el contenido comprimido:** a través del BitReader, recorriendo el árbol hasta encontrar hojas que representan caracteres originales.
- **Reconstruir y escribir el archivo final:** con el texto completamente descomprimido y sin pérdida de información.

- **utils.cpp / utils.h:**
  Los archivos **utils.cpp** y **utils.h** contienen herramientas esenciales para el manejo de bits y la lectura/escritura de archivos en el proceso de compresión Huffman.
  Dentro de ellos se encuentran las clases **BitWriter** y **BitReader**, que permiten trabajar con datos a nivel de bits y no únicamente por bytes o caracteres completos. Esto es fundamental ya que el objetivo del algoritmo es representar los caracteres con códigos binarios más cortos y ahorrar espacio real en el archivo comprimido.

- **BitWriter** se encarga de ir agrupando los bits generados por Huffman dentro de un buffer de 8 bits (1 byte), y solo cuando el buffer está completo se escribe en el archivo comprimido. Si al finalizar sobran bits, estos se completan con ceros para evitar pérdida de datos. Sus funciones principales son:
  - **writeBit():** escribe un único bit en el buffer.
  - **writeBits():** escribe una secuencia de bits.
  - f**lush():** vacía el buffer al final del proceso.

- **BitReader** realiza lo opuesto: lee de un archivo comprimido un byte a la vez y va devolviendo los bits individuales para reconstruir el texto a través del árbol de Huffman durante la descompresión. Su método principal es:
  - readBit() → devuelve el siguiente bit disponible del archivo

  También se incluyen dos funciones auxiliares para manejo de archivos binarios:
  - readFileToString() → lee todo el archivo de entrada y lo almacena en una cadena.
  - writeStringToFile() → escribe el texto descomprimido en un archivo de salida.

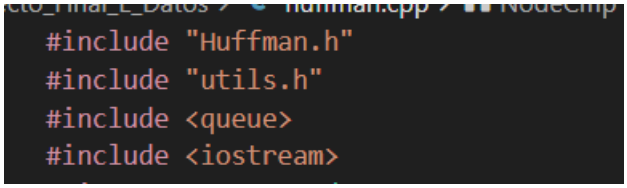
Estas utilidades permiten leer, interpretar, almacenar y restaurar correctamente la secuencia de bits generada por el algoritmo. Sin ellas, la compresión no sería eficiente ni sería posible reconstruir el archivo original de manera correcta.

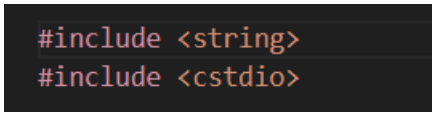
- **Main.cpp**

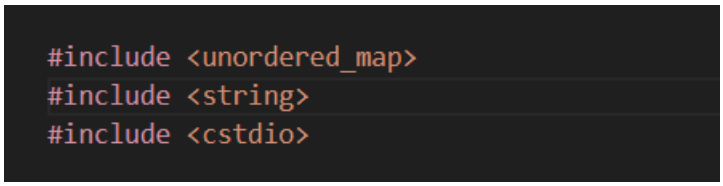
Controla el menú principal y permite al usuario usar el compresor Huffman:

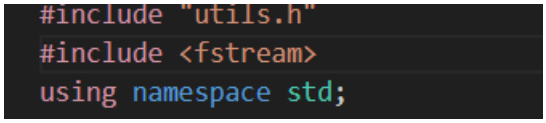
- Muestra opciones para **comprimir**, **descomprimir** o **mostrar los códigos generados**.
- Según la opción elegida, llama a las funciones del objeto Huffman:
  - compress(input, output) para comprimir un archivo.
  - decompress(input, output) para restaurar el archivo original.
  - printCodes() para visualizar los códigos Huffman.
- Permanece en un bucle hasta que el usuario elija **Salir**.

Librerías utilizadas









El proyecto utiliza librerías estándar de C++:

- **<iostream>:** Entrada y salida de datos por consola.
- <**fstream>:** Manejo de archivos para leer y escribir datos binarios.
- **<queue>:** Uso de **priority_queue**, esencial para construir el árbol de Huffman.
- **<unordered_map>**: Recolección de frecuencias y almacenamiento de códigos.
- **<string>:** Manipulación directa de secuencias de caracteres y datos binarios.
- **<cstdio>:** Biblioteca heredada de C que permite manejar archivos y datos binarios a bajo nivel mediante funciones como fread() y fwrite(). Esto es especialmente útil en compresión porque se necesita trabajar directamente con bytes y posiciones dentro del archivo para evitar pérdidas de información.

Ejecución y explicación de resultados:

- **Ejecución del programa:** Para ejecutar el programa se recomienda seguir los siguientes pasos:
  - Descargar y descomprimir el proyecto
  - Descargar el compilador g++ en caso de no tenerlo por medio de MSYS2 desde su página oficial y desde el CMD poner lo siguiente: pacman -S mingw-w64-x86_64-gcc.
  - Ejecutar los siguientes comandos en consola CMD:

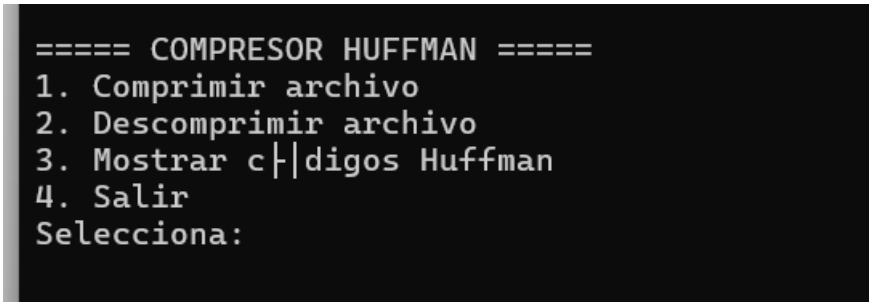
```
C:\Users\gisel\OneDrive\Escritorio\ProyectoFinal-estructuras>cd Proyecto_Final_E_Datos

C:\Users\gisel\OneDrive\Escritorio\ProyectoFinal-estructuras\Proyecto_Final_E_Datos>g++ main.cpp huffman.cpp utils.cpp

C:\Users\gisel\OneDrive\Escritorio\ProyectoFinal-estructuras\Proyecto_Final_E_Datos>ProyectoFinal.exe
```

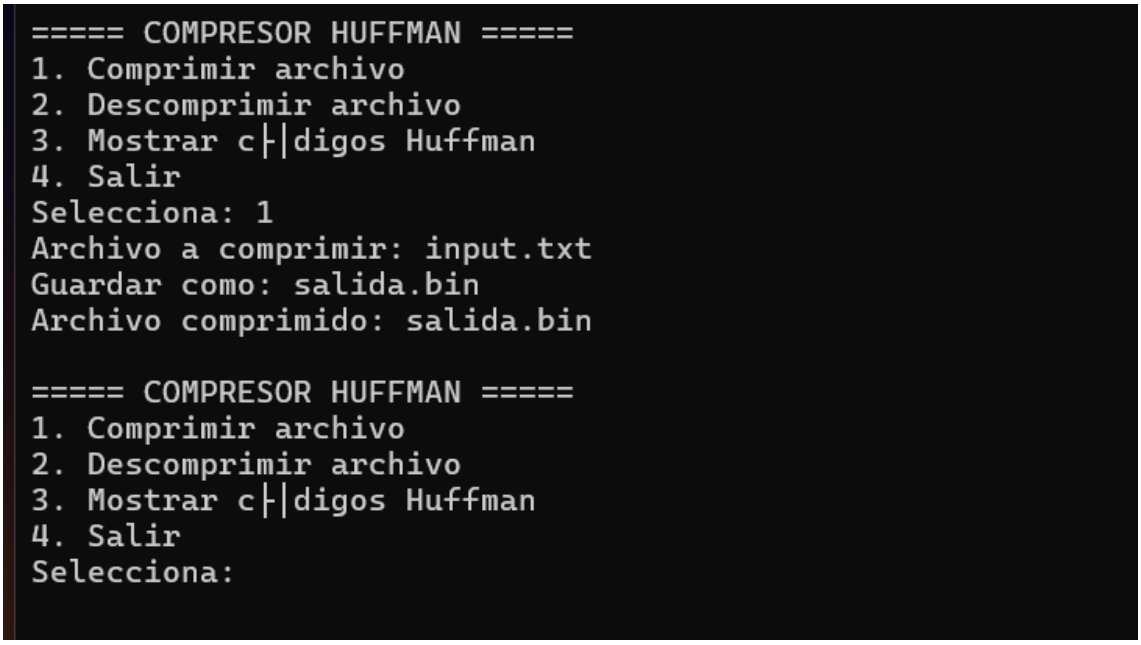
    Esto para asegurarse de estar en la carpeta correcta llamada Proyecto_Final_E_Datos, para ejecutar el programa con g++ main.cpp huffman.cpp utils.cpp y finalmente hacer llamado al ejecutable ProyectoFinal.exe.

    - Una vez llamado el ejecutable aparecerá lo siguiente:



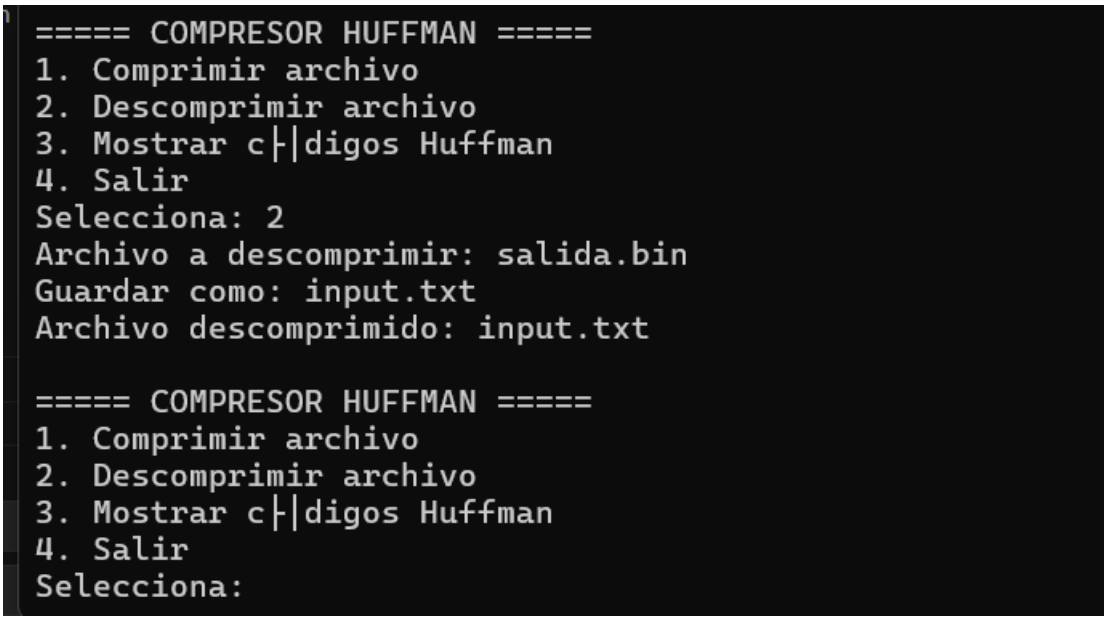
    Que es el menú interactivo que se creó desde el main.cpp para darle al usuario claridad con respecto a las funcionalidades que podrá obtener con el programa.

    - El usuario podrá elegir cualquier opción pero se recomienda iniciar con la opción 1 que le pedirá y mostrará esto:



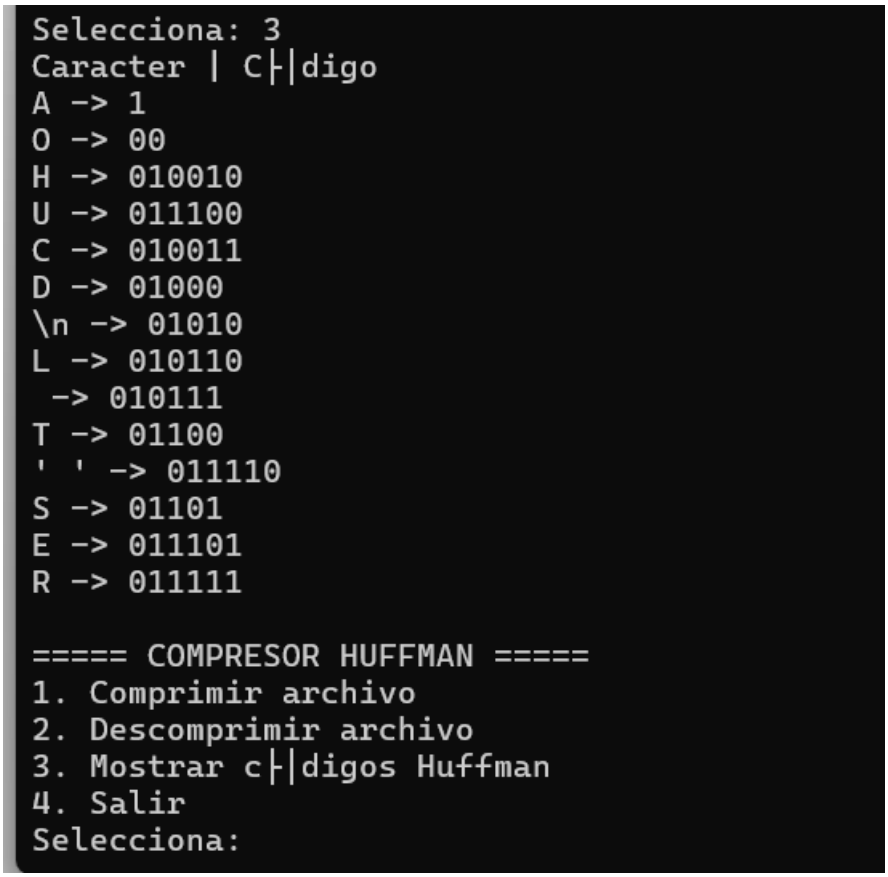
Le pedirá que ingrese el nombre del archivo a comprimir, este archivo es en formato .txt y ya existen 3 dentro del proyecto uno se llama input.txt , entrada.txt y entrada2.txt. Después de ingresar alguna de ellas le pedirá que ingrese el nombre con el cual desea guardarlo y podrá ponerle cualquier nombre pero asegurándose de que siempre termine en .bin para que no le salgá un error y le comprima. Después le saldra un mensaje de error o éxito y le volverá a mostrar el menú.

- Si elige la opción 2 saldrá lo siguiente:



Esta vez le pedirá que ingres un .bin ya existe y que ingrese el nombre que quiere que tenga al descomprimirlo.

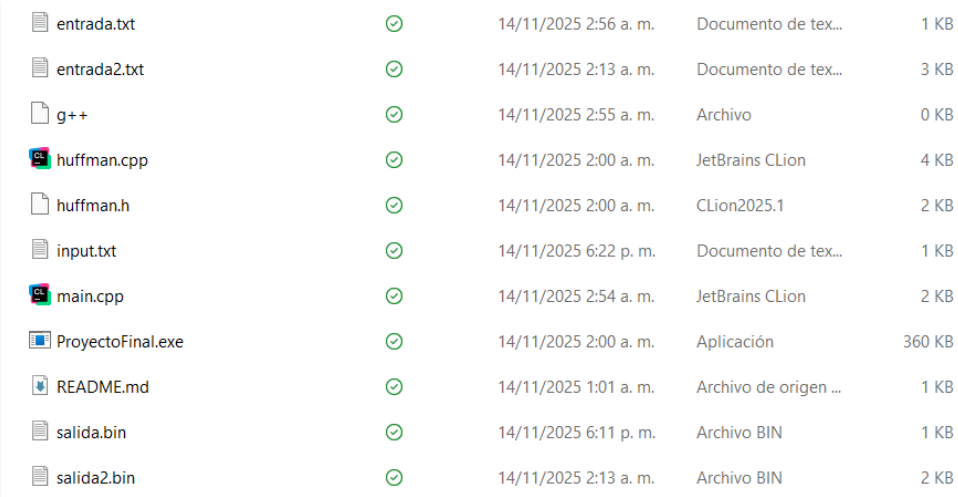
- Para la opción 3:



Lo que hace es mostrar en pantalla cada carácter detectado en el archivo junto con su código binario generado por el algoritmo de Huffman. En otras palabras, Imprime la tabla de códigos Huffman que se utilizó (o se utilizará) para la compresión.

- La opción 4 simplemente lo que hará será salirse completamente del programa y que ya no aparezca más el menú.

Si queremos comprobar si realmente funcionó, nos dirigimos al explorador de archivos, buscamos el lugar donde está guardado el proyecto, lo abrimos y mostrará lo siguiente:



entrada.txt	14/11/2025 2:56 a. m.	Documento de tex...	1 KB
entrada2.txt	14/11/2025 2:13 a. m.	Documento de tex...	3 KB
g++	14/11/2025 2:55 a. m.	Archivo	0 KB
huffman.cpp	14/11/2025 2:00 a. m.	JetBrains CLion	4 KB
huffman.h	14/11/2025 2:00 a. m.	CLion2025.1	2 KB
input.txt	14/11/2025 6:22 p. m.	Documento de tex...	1 KB
main.cpp	14/11/2025 2:54 a. m.	JetBrains CLion	2 KB
ProyectoFinal.exe	14/11/2025 2:00 a. m.	Aplicación	360 KB
README.md	14/11/2025 1:01 a. m.	Archivo de origen ...	1 KB
salida.bin	14/11/2025 6:11 p. m.	Archivo BIN	1 KB
salida2.bin	14/11/2025 2:13 a. m.	Archivo BIN	2 KB

Tendrá que aparecer el archivo txt que elegimos comprimir y el .bin(de menor o igual tamaño que el .txt que se comprimió) que creó almacenados en el mismo lugar, como en este caso si se cumplió esto. Se concluye que el programa sí funcionó en su implementación y que todo el código creado en c++ y la implementación del compresor de hoffman funcionaron exitosamente. Cumpliendo así con los requerimientos del programa.

Conclusiones

El proyecto permitió comprender y aplicar estructuras de datos (árboles binarios, recorrido de árboles binarios, colas de prioridad, etc.) en un contexto práctico de compresión. La implementación del algoritmo de Huffman demostró que se puede aprovechar la frecuencia de aparición de los símbolos para reducir el tamaño de los archivos manteniendo la integridad de los datos.