

Instituto Politécnico Nacional Escuela Superior de Cómputo



Analisis de algoritmos, Sem: 2021-1, 3CV1,Práctica 5, 11/12/2020 Práctica 5: Algoritmos Greedy Payán Téllez René

rpayant1500@alumno.ipn.mx

Resumen: En esta practica se analizaran 2 algoritmos que utilizan la tecnica de divide y venceras para resolver un problema, uno de ellos el quick sort y otro el sub arreglo maximo.

Palabras clave: Huffman, Kruskal, Gredy, STL, C/C++

1 Introduccion

Los algoritmos glotones, son en su mayoria utilizados para resolver problemas extremadamente complejos de forma simple de programar, normalmente no es la mejor solucion pero casi siempre bastante cercana y se programa de forma extremadamente rapida, sin mencionar que normalmente no toman mucha complejidad computacional. Por ello aprender respecto a estos algoritmos es extremadamente importante, inclusive de los 2 que vamos a hablar a continuacion, ya que tienen usos tanto en redes (Huffman) como en proyectos (Kruskal).

2 Conceptos Basicos

2.1 Algoritmo

La palabra algoritmo proviene del sobrenombre de un matemático árabe del siglo IX, Al-Khwarizmi, que fue reconocido por enunciar paso a paso las reglas para las operaciones matemáticas básicas con decimales (suma, resta, multiplicación y división). Vemos definición de algoritmo como un grupo de órdenes consecutivas que presentan una solución a un problema o tarea. Algunos ejemplos de algoritmos los podemos encontrar en las matemáticas (como el algoritmo para resolver una multiplicación) y en los manuales de usuario de un aparato (como una lavadora o una impresora). Sin embargo, hoy en día se relaciona la palabra algoritmo con el mundo de la informática, más concretamente en la programación; los conocidos como algoritmos informáticos. [1]

2.2 Complejidad algoritmica

Así que, por su naturaleza, un problema tiene la capacidad de ser solucionado por uno o varios métodos, pero si bien es importante llegar a la respuesta, más importante es evaluar su viabilidad. Siempre que se analiza y evalúa adecuadamente la efectividad de una solución, disminuye drásticamente el costo que representa su producción y mantenimiento, pues los recursos que se invierten posteriormente en codificación, pruebas y revisión es mucho menor siempre (como el tiempo, dinero y talento humano). Entrando en materia, la complejidad algorítmica es una métrica teórica que nos ayuda a describir el comportamiento de un algoritmo en términos de tiempo de ejecución (tiempo que tarda un algoritmo en resolver un problema) y memoria requerida (cantidad de memoria necesaria para procesar las instrucciones que solucionan dicho problema). Esto nos ayuda a comparar entre la efectividad de un algoritmo y otro, y decidir cuál es el que nos conviene implementar.[2]

2.3 Algoritmos Greedy o glotones

Un algoritmo Greedy o gloton es un algoritmo muy util para encontrar soluciones aproximadas e inclusive la mas optima a problemas complejos, ya que las entregan en muy corto tiempo. Se llaman Greedy porque siempre "comen lo que tienen a la mano", no garantizan encontrar la mejor solucion, pero si una aproximacion bastante buena.[3] Estas son sus caracteristicas principales:

- Se utilizan generalmente para resolver problemas deoptimización (obtener el máximo o el mínimo).optimización (obtener el máximo o el mínimo).
- Toman decisiones en función de la información que está disponible en cada momento. está disponible en cada momento. está disponible en cada momento. está disponible en cada momen
- Una vez tomada la decisión, ésta no vuelve a replantearse en el futuro.replantearse en el futuro.
- Suelen ser rápidos y fáciles de implementar.
- No siempre garantizan alcanzar la solución óptima[4]

2.4 Algoritmo de codificación de Huffman

El código de Huffman es un tipo particular de código de prefijo óptimo que se usa comúnmente para la compresión de datos sin pérdida. Comprime los datos de manera muy efectiva, ahorrando de 20% a 90% de memoria, dependiendo de las características de los datos comprimidos. Este algoritmo se aplica solo si se considera a la entrada como una cadena de caracteres, ya que es un algoritmo boraz que utiliza una tabla que proporciona la frecuencia con la que aparece cada carácter (es decir, su frecuencia) para crear una forma óptima de representar cada carácter como una cadena binaria. Fue propuesto por David A. Huffman en 1951.[5]

Algoritmo 1: HuffmanTree(C)

```
Data: Entrada: C (Los caracteres de la cadena a codificar y su ocurrencia)

Result: Retorna el arbol de huffman de la cadena

n = C.size;

Q = priority\_queue();

for i \leftarrow 0 to i < n do

n = node(C[i]);

Q.push(n);

while Q.size() > 1 do

Z = new \ node();

Z.left = x = Q.pop();

Z.right = y = Q.pop();

Z.frequency = x.frequency + y.frequency;

Q.push(Z);

return Q;
```

Algoritmo 2: HuffmanDecompression(tree, S)

```
Data: Entrada: tree (El arbol de Huffman), S (la cadena binaria a descomprimir)

Result: Retorna una cadena de caracteres descomprimida

n = S.length;
retorno = "";
for i \leftarrow 0 to i < n do

| current = root;
| while current.left != NULL and current.right!=NULL do

| if S[i] == '0' then
| current = current.left;
| else
| current = current.right;
| i+=1;
| retorno+=current.symbol;
| return retorno;
```

2.5 Algoritmos de Kruskal

El algoritmo de Kruskal es un algoritmo de la teoría de grafos para encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque expandido mínimo (un árbol expandido mínimo para cada componente conexa).[6]

Kruskal funciona de la siguiente forma:

- 1. Se selecciona, de entre todas las aristas restantes, la de menor peso siempre que no cree ningún ciclo.
- 2. Se repite el paso 1 hasta que se hayan seleccionado $\|V\|-1$ aristas.[7]

Su complejidad es $O(mlog_2(n))[8]$, donde m
 es el numero de aristas y n el numero de nodos.

3 Experimentacion y Resultados

3.1 Implementar el algoritmo de codificacion de Huffman.

Para esta primera parte de pruebas se implemento el algoritmo de codificacion y decodificacion de Huffman, asi que la ejecución se divide en 2 partes, la compresion de una cadena de texto y para la segunda parte la descompresion de la misma

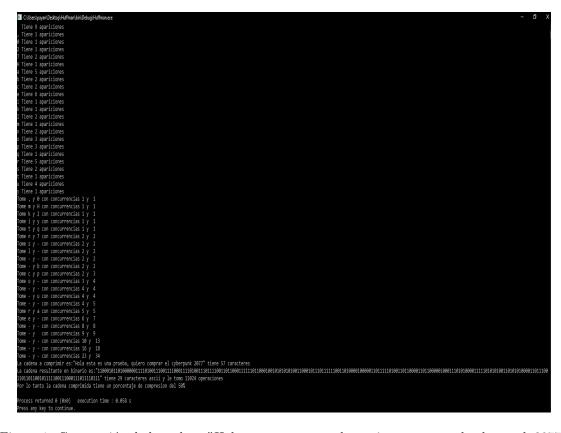


Figure 1: Compresión de la cadena "Hola esta es una prueba, quiero comprar el cyberpunk 2077"

```
me k y 2 con concurrencias 1 y 1
 me i y y con concurrencias 1 y 1
me t y q con concurrencias 1 y 1
 me n y 7 con concurrencias 2 y 🗆
           con concurrencias 2 y
           con concurrencias 2 v
        b con concurrencias 2 y
          - con concurrencias 4 y 4
     y u con concurrencias 4 y
me r y a con concurrencias 5 y 5
          - con concurrencias 6 y
          - con concurrencias 8 y 1
          - con concurrencias 10 y 13
 me - y - con concurrencias 16 y 18
 me - y - con concurrencias 23 y 34
 cadena a comprimir es:"Hola esta es una prueba, quiero comprar el cyberpunk 2077" tiene 57 caracteres
8110110010111111001111000111011110111" tiene 29 caracteres ascii y le tomo 11024 operaciones
 r lo tanto la cadena comprimida tiene un porcentaje de compresion del 50%
 Cadena comprimida es: "11880/1911/1988888011111/181811111801111/1911/11188111111/1911/1118111111/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/1911/191
11111001110001110111110111 y la cadena descomprimida es: "Hola esta es una prueba, quiero comprar el cyberpunk 2077" y le tomo 1104 operaciones
rocess returned 0 (0x0) execution time : 0.056 s
ress any key to continue.
```

Figure 2: Descompresión de la cadena "Hola esta es una prueba, quiero comprar el cyberpunk 2077"

Como se pudo apreciar en las capturas, la capturas, el programa tomo la entrada "Hola esta es una prueba, quiero comprar el cyberpunk 2077" y retorno la siguiente salida como respuesta: "", posteriormente al introducir la salida de la compresion en la seccion de descompresion, el programa arrojo la cadena original. Podemos decir que el algoritmo es eficiente, porque esta es la cantidad de caracteres de la cadena original:



Figure 3: Conteo de caracteres de la entrada

esta es la cantidad de caracteres de la salida:

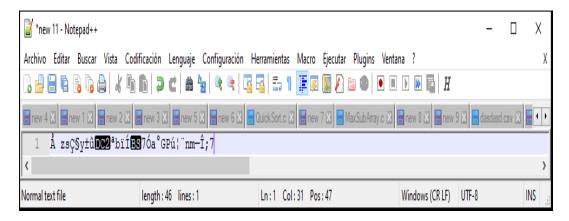


Figure 4: Conteo de caracteres de la salida

por ultimo, el binario de cada caracter asignado

```
apariciones
                     binario
  apariciones
                     binario
                                    100010
  apariciones
1
  apariciones
                     binario
                                    110011
                               es:
                                    110001
                                   001
2
  apariciones
                     binario
                               es:
                                   10101
  apariciones
                                   11010
                                   010
                                    101000
2
  apariciones
                               es:
                                    10000
                                    10110
                               es:
  apariciones
                                   0110
3
  apariciones
                                    11011
1
  apariciones
                     binario
                               es:
                                   011111
                                   01110
                                   011110
4
  apariciones
                  su
                     binario
                               es:
                                   1001
  apariciones
                  su binario
                                    101001
```

Figure 5: Asignacion de binario

A continuacion se probo con una cadena mas larga, un "Lorem Ipsum" de 1000 bytes: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer porttitor turpis eget erat auctor varius. Vestibulum maximus scelerisque dui ac vulputate. Mauris eleifend mauris vel ex sodales, ut blandit odio dictum. Ut efficitur eu lectus nec ullamcorper. Integer hendrerit justo in augue consectetur, eget porttitor quam rutrum. Cras porta justo at fermentum condimentum. Nunc lacinia convallis tortor in tempor. Donec tincidunt tempor ipsum, sit amet aliquet justo semper at. Donec nibh urna, faucibus ac mauris eu, mattis imperdiet dolor. Nam et nulla at nisi efficitur efficitur. Morbi bibendum scelerisque risus, at sollicitudin ligula rutrum et. Cras sagittis eget velit aliquam cursus. Nunc magna metus, ullamcorper sed ante id, tincidunt ornare libero. Proin pharetra orci felis, eget

pellentesque nisi venenatis eget. Morbi tincidunt ut risus non iaculis. Proin id consectetur metus, sed placerat eros. Aenean cursus augue a ipsum tempor interdum. Vivamus viverra efficitur felis a aenean.

la calena a compriadr est "toren igonn blor sit aet, cosecteir adpiscing ellt. Integer portitor buyds eget evit actor narius vietibulm machus selerisque du at vulptate. Navis elefed machis vel en sociales, ut blandi officitum. It efficitum elletis se su llanorper. Integer hedreit juto in agua cosecteir, eget portitor quan norum. Cas porta justo at fernetum condinetum. Unc lacida consellis tortor in tempor locat ticcidus tempor justo, a trata discription efficitum. Rochi blevom scelerisque risus, at solicidud liquia norum et. Cras septitis eget ellet alique an orusa. Unc segue mius, ullanorper sed arte id, ticcidus compre libero. Probi piene arte discription en elle segue an orusa. Unc segue mius, ullanorper sed arte id, ticcidus compre libero. Probi piene arte discription en elle segue an orusa. Unc segue mius, ullanorper sed arte id, ticcidus compre libero. Probi piene arte discription en elle segue an orusa. Unc segue mius, ullanorper sed arte id, ticcidus compre libero. Probi piene arte discription elle segue an orusa libero elle segue antique antiq

Figure 6: Compresión del lorem ipsum



Figure 7: Descompresión del lorem ipsum

Finalmente se contaron los caractres de la entrada y de la salida:

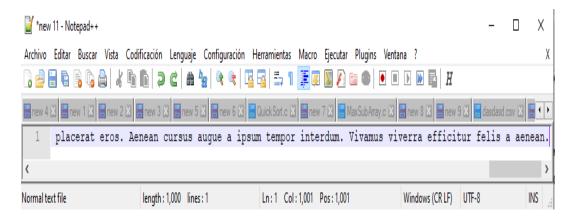


Figure 8: Conteo de caracteres de la entrada

y esta es la cantidad de caracteres de la salida:

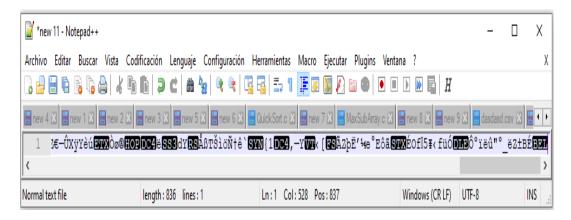


Figure 9: Conteo de caracteres de la salida

por ultimo, el binario de cada caracter asignado

```
1101111
                                    100111
  apariciones
                                   1001100010
  apariciones
                                   000111111
  apariciones
                     binar
                                   100110010
  apariciones
                                   000111100
  apariciones
                                   00011011
  apariciones
                                   00011100
  apariciones
                     binar
                                   000111110
                                   1001100011
                                   100110011
   aparic
                                    0110
  apariciones
                     binar
                                   1101110
38
   apariciones
                   SII
                      binar
                                    11010
   aparicione
                                    00010
                                    001
   apariciones
                                    011101
                   su
                                    011100
   aparicione
                                   00011010
                                    01111
   aparic
   apariciones
                                    10010
                   su
   apariciones
                   su
                      binar
                                    0100
                                    aaaa
   aparicione
                                    1000
   apariciones
                                    001101
  aparic
```

Figure 10: Asignacion de binario

A continuación se comprimieron 10 archivos de texto con extencion .txt de distinto tamaño, se anexan las capturas del binario de cada caracter, del archivo de entrada, el archivo comprimido, el archivo descomprimido y el tamaño de ambos archivos.

Primer archivo

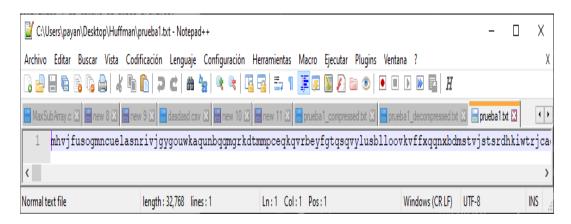


Figure 11: Archivo de entrada

```
a Tiene 1220 apariciones y su binario es: 01101
b Tiene 1280 apariciones y su binario es: 11110
c Tiene 1280 apariciones y su binario es: 10100
d Tiene 1293 apariciones y su binario es: 10100
e Tiene 1293 apariciones y su binario es: 10100
e Tiene 1293 apariciones y su binario es: 10100
f Tiene 1293 apariciones y su binario es: 10110
g Tiene 1293 apariciones y su binario es: 10110
g Tiene 1293 apariciones y su binario es: 10110
f Tiene 1290 apariciones y su binario es: 10110
j Tiene 1290 apariciones y su binario es: 10111
i Tiene 1280 apariciones y su binario es: 10101
j Tiene 1290 apariciones y su binario es: 10101
l Tiene 1291 apariciones y su binario es: 10101
l Tiene 1293 apariciones y su binario es: 00010
n Tiene 1290 apariciones y su binario es: 00010
n Tiene 1290 apariciones y su binario es: 00010
n Tiene 1290 apariciones y su binario es: 00011
r Tiene 1294 apariciones y su binario es: 10011
r Tiene 1294 apariciones y su binario es: 10011
r Tiene 1295 apariciones y su binario es: 10011
r Tiene 1296 apariciones y su binario es: 10011
r Tiene 1291 apariciones y su binario es: 10011
r Tiene 1291 apariciones y su binario es: 10011
r Tiene 1291 apariciones y su binario es: 10010
u Tiene 1291 apariciones y su binario es: 10001
v Tiene 1291 apariciones y su binario es: 10001
v Tiene 1291 apariciones y su binario es: 10001
v Tiene 1293 apariciones y su binario es: 10001
v Tiene 1294 apariciones y su binario es: 10001
v Tiene 1295 apariciones y su binario es: 10001
v Tiene 1296 apariciones y su binario es: 10001
v Tiene 1297 apariciones y su binario es: 10001
v Tiene 1298 apariciones y su binario es: 10001
v Tiene 1290 apariciones y su binario es: 10001
v Tiene 1290 apariciones y su binario es: 10001
v Tiene 1290 apariciones y su binario es: 10001
v Tiene 1290 apariciones y su binario es: 10001
v Tiene 1290 apariciones y su binario es: 10001
v Tiene 1290 apariciones y su binario es: 10001
v Tiene 1290 apariciones y su binario es: 10001
v Tiene 1290 apariciones y su binario es: 10001
v Tiene 1290 apariciones y su
```

Figure 12: Ejecucion del programa

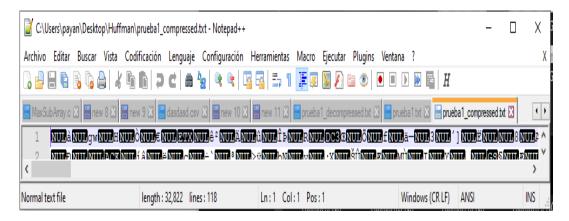


Figure 13: Archivo de salida

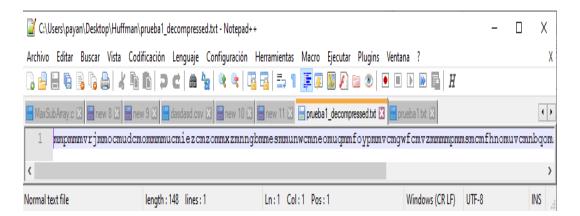


Figure 14: Archivo descomprimido

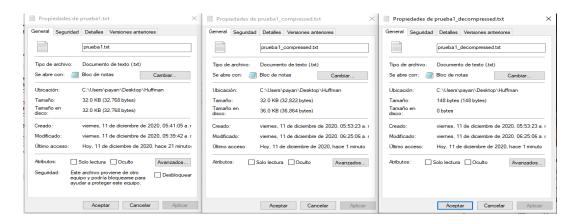


Figure 15: peso de los 3 archivos

Segundo archivo

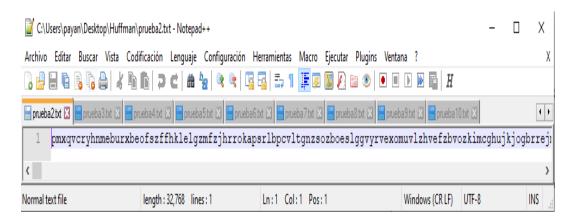


Figure 16: Archivo de entrada

```
C:\Users\payan\Desktop\Huffman\bin\Debug\Huffman.exe
 Tiene 1234 apariciones y su binario es: 10010
 Tiene 1255 apariciones y su binario es: 11000
Tiene 1231 apariciones y su binario es: 01110
Tiene 1264 apariciones y su binario es: 11100
 Tiene 1233 apariciones y su binario es: 10000
 Tiene 1258 apariciones y su binario es: 11010
Tiene 1294 apariciones y su binario es: 0001
 Tiene 1242 apariciones y su binario es: 10110
 Tiene 1238 apariciones y su binario es: 10011
Tiene 1305 apariciones y su binario es: 0011
 Tiene 1269 apariciones y su binario es: 11110
 Tiene 1219 apariciones y su binario es: 01100
Tiene 1232 apariciones y su binario es: 01111
 Tiene 1242 apariciones y su binario es: 10101
 Tiene 1323 apariciones y su binario es: 0100
Tiene 1280 apariciones y su binario es: 11111
 Tiene 1233 apariciones y su binario es: 10001
 Tiene 1227 apariciones y su binario es: 01101
 Tiene 1338 apariciones y su binario es: 0101
 Tiene 1255 apariciones y su binario es: 11001
 Tiene 1263 apariciones y su binario es: 11011
 Tiene 1267 apariciones y su binario es: 11101
Tiene 1246 apariciones y su binario es: 10111
Tiene 1287 apariciones y su binario es: 0000
 Tiene 1294 apariciones y su binario es: 0010
Tiene 1239 apariciones y su binario es: 10100
Se genero el arbol en 164382 operaciones
se comprimio el archivo: prueba2.txt en prueba2 compressed.txt en 7349197 operaciones
se descomprimio el archivo: prueba2_compressed.txt en prueba2_decompressed.txt en 8538 operaciones
Process returned 0 (0x0) execution time : 0.225 s
ress any key to continue.
```

Figure 17: Ejecucion del programa

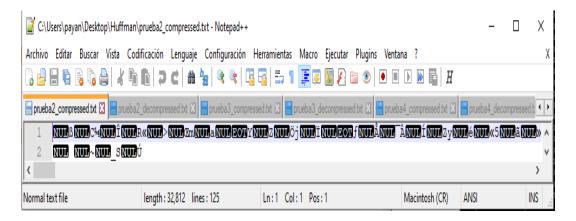


Figure 18: Archivo de salida



Figure 19: Archivo descomprimido

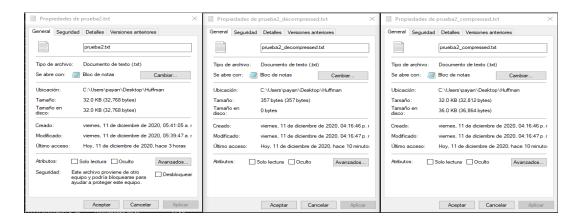


Figure 20: peso de los 3 archivos

Tercer archivo

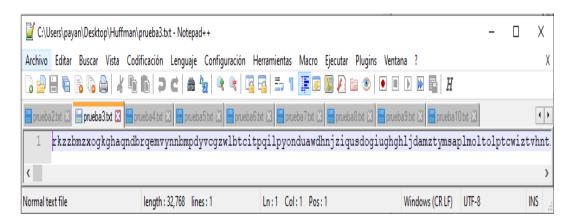


Figure 21: Archivo de entrada

```
C:\Users\payan\Desktop\Huffman\bin\Debug\Huffman.exe
 Tiene 1282 apariciones y su binario es: 0000
 Tiene 1275 apariciones y su binario es: 11110
 Tiene 1265 apariciones y su binario es: 11001
Tiene 1232 apariciones y su binario es: 10000
  Tiene 1245 apariciones y su binario es: 10100
 Tiene 1238 apariciones y su binario es: 10001
Tiene 1295 apariciones y su binario es: 0010
  Tiene 1210 apariciones y su binario es: 01101
 Tiene 1320 apariciones y su binario es: 0101
Tiene 1239 apariciones y su binario es: 10010
  Tiene 1215 apariciones y su binario es: 01110
 Tiene 1285 apariciones y su binario es: 0001
Tiene 1272 apariciones y su binario es: 11100
  Tiene 1270 apariciones y su binario es: 11010
 Tiene 1164 apariciones y su binario es: 01100
Tiene 1231 apariciones y su binario es: 01111
  Tiene 1255 apariciones y su binario es: 10101
  Tiene 1281 apariciones y su binario es: 11111
  Tiene 1306 apariciones y su binario es: 0011
 Tiene 1263 apariciones y su binario es: 10111
Tiene 1258 apariciones y su binario es: 10110
 Tiene 1244 apariciones y su binario es: 10011
 Tiene 1265 apariciones y su binario es: 11000
Tiene 1274 apariciones y su binario es: 11101
 Tiene 1314 apariciones y su binario es: 0100
: Tiene 1270 apariciones y su binario es: 11011
se genero el arbol en 164382 operaciones
e comprimio el archivo: prueba3.txt en prueba3 compressed.txt en 7349424 operaciones
e descomprimio el archivo: prueba3_compressed.txt en prueba3_decompressed.txt en 6986 operaciones
Process returned 0 (0x0) execution time : 0.211 s
ress any key to continue.
```

Figure 22: Ejecucion del programa



Figure 23: Archivo de salida



Figure 24: Archivo descomprimido

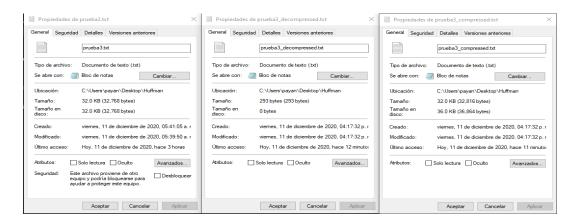


Figure 25: peso de los 3 archivos

Cuarto archivo

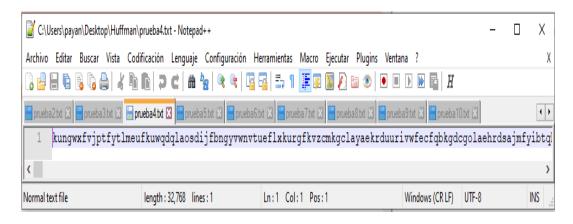


Figure 26: Archivo de entrada

```
C:\Users\payan\Desktop\Huffman\bin\Debug\Huffman.exe
 Tiene 1247 apariciones y su binario es: 10100
 Tiene 1228 apariciones y su binario es: 10001
 Tiene 1252 apariciones y su binario es: 10111
Tiene 1221 apariciones y su binario es: 10000
 Tiene 1254 apariciones y su binario es: 11000
 Tiene 1247 apariciones y su binario es: 10011
Tiene 1208 apariciones y su binario es: 01100
 Tiene 1309 apariciones y su binario es: 0100
 Tiene 1232 apariciones y su binario es: 10010
Tiene 1213 apariciones y su binario es: 01101
 Tiene 1329 apariciones y su binario es: 0101
 Tiene 1306 apariciones y su binario es: 0011
Tiene 1279 apariciones y su binario es: 11101
 Tiene 1216 apariciones y su binario es: 01110
 Tiene 1299 apariciones y su binario es: 0010
Tiene 1218 apariciones y su binario es: 01111
 Tiene 1259 apariciones y su binario es: 11010
 Tiene 1249 apariciones y su binario es: 10110
Tiene 1248 apariciones y su binario es: 10101
 Tiene 1292 apariciones y su binario es: 0000
 Tiene 1284 apariciones y su binario es: 11110
Tiene 1258 apariciones y su binario es: 11001
 Tiene 1263 apariciones y su binario es: 11011
 Tiene 1296 apariciones y su binario es: 0001
Tiene 1272 apariciones y su binario es: 11100
 Tiene 1289 apariciones y su binario es: 11111
se genero el arbol en 164382 operaciones
se comprimio el archivo: prueba4.txt en prueba4_compressed.txt en 7349316 operaciones
Se descomprimio el archivo: prueba4_compressed.txt en prueba4_decompressed.txt en 10462 operaciones
Process returned 0 (0x0) execution time : 0.219 s
Press any key to continue.
```

Figure 27: Ejecucion del programa

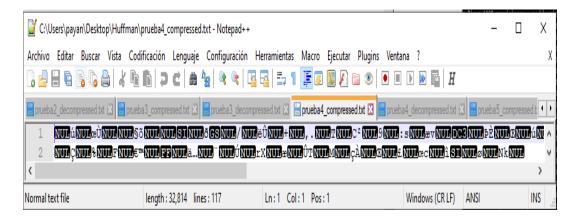


Figure 28: Archivo de salida

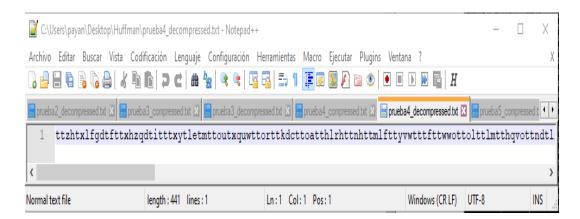


Figure 29: Archivo descomprimido

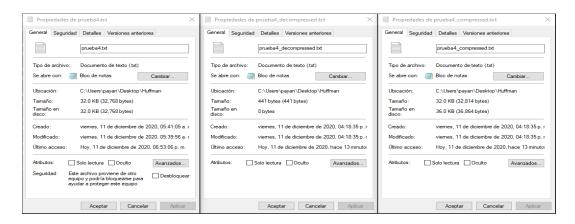


Figure 30: peso de los 3 archivos

Quinto archivo

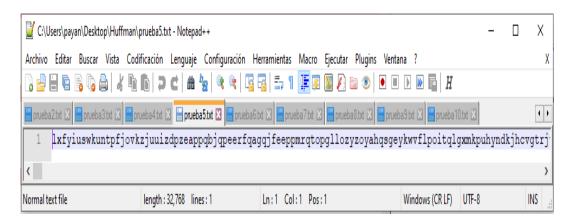


Figure 31: Archivo de entrada

```
C:\Users\payan\Desktop\Huffman\bin\Debug\Huffman.exe
Tiene 1240 apariciones y su binario es: 10011
Tiene 1215 apariciones y su binario es: 01110
Tiene 1210 apariciones y su binario es: 01100
Tiene 1228 apariciones y su binario es: 10000
Tiene 1273 apariciones y su binario es: 11010
Tiene 1237 apariciones y su binario es: 10010
Tiene 1322 apariciones y su binario es: 0101
 Tiene 1242 apariciones y su binario es: 10110
 Tiene 1295 apariciones y su binario es: 0001
 Tiene 1280 apariciones y su binario es: 11101
 Tiene 1305 apariciones y su binario es: 0011
 Tiene 1246 apariciones y su binario es: 10111
Tiene 1286 apariciones y su binario es: 0000
Tiene 1255 apariciones y su binario es: 11001
Tiene 1283 apariciones y su binario es: 11111
Tiene 1279 apariciones y su binario es: 11100
Tiene 1226 apariciones y su binario es: 01111
Tiene 1241 apariciones y su binario es: 10101
Tiene 1214 apariciones y su binario es: 01101
Tiene 1296 apariciones y su binario es: 0010
Tiene 1281 apariciones y su binario es: 11110
Tiene 1241 apariciones y su binario es: 10100
Tiene 1254 apariciones y su binario es: 11000
Tiene 1275 apariciones y su binario es: 11011
Tiene 1310 apariciones y su binario es: 0100
Tiene 1234 apariciones y su binario es: 10001
Se genero el arbol en 164382 operaciones
se comprimio el archivo: prueba5.txt en prueba5 compressed.txt en 7349362 operaciones
se descomprimio el archivo: prueba5_compressed.txt en prueba5_decompressed.txt en 5950 operaciones
Process returned 0 (0x0)
                               execution time : 0.215 s
ress any key to continue.
```

Figure 32: Ejecucion del programa

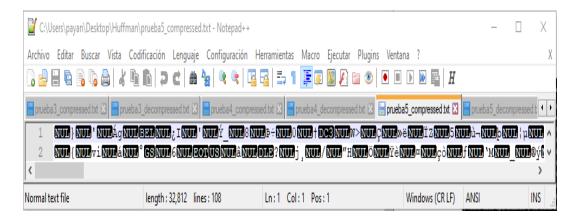


Figure 33: Archivo de salida

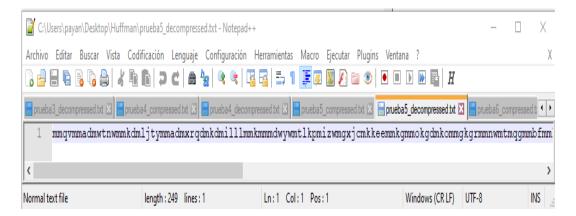


Figure 34: Archivo descomprimido

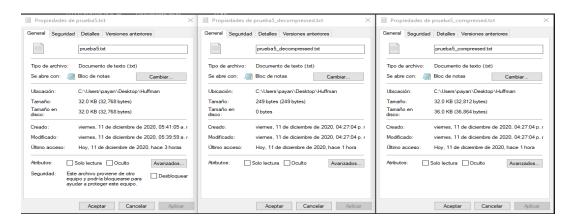


Figure 35: peso de los 3 archivos

Sexto archivo

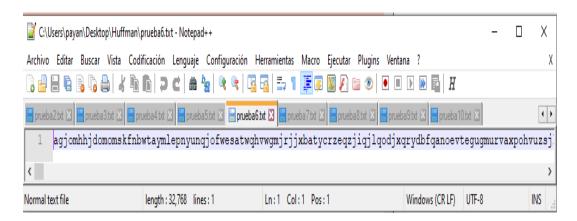


Figure 36: Archivo de entrada

```
C:\Users\payan\Desktop\Huffman\bin\Debug\Huffman.exe
Tiene 1257 apariciones y su binario es: 11000
Tiene 1265 apariciones y su binario es: 11010
Tiene 1316 apariciones y su binario es: 0101
Tiene 1247 apariciones y su binario es: 10011
Tiene 1274 apariciones y su binario es: 11111
 Tiene 1250 apariciones y su binario es: 10101
 Tiene 1232 apariciones y su binario es: 01110
 Tiene 1293 apariciones y su binario es: 0011
Tiene 1238 apariciones y su binario es: 01111
Tiene 1267 apariciones y su binario es: 11011
 Tiene 1254 apariciones y su binario es: 10111
Tiene 1270 apariciones y su binario es: 11110
Tiene 1269 apariciones y su binario es: 11101
Tiene 1239 apariciones y su binario es: 10000
Tiene 1239 apariciones y su binario es: 10001
Tiene 1267 apariciones y su binario es: 11100
Tiene 1261 apariciones y su binario es: 11001
Tiene 1292 apariciones y su binario es: 0010
Tiene 1292 apariciones y su binario es: 0001
 Tiene 1201 apariciones y su binario es: 01100
Tiene 1224 apariciones y su binario es: 01101
Tiene 1242 apariciones y su binario es: 10010
Tiene 1284 apariciones y su binario es: 0000
Tiene 1296 apariciones y su binario es: 0100
Tiene 1249 apariciones y su binario es: 10100
Tiene 1250 apariciones y su binario es: 10110
Se genero el arbol en 164382 operaciones
se comprimio el archivo: prueba6.txt en prueba6 compressed.txt en 7349456 operaciones
se descomprimio el archivo: prueba6_compressed.txt en prueba6_decompressed.txt en 22618 operaciones
rocess returned 0 (0x0) execution time : 0.224 s
ress any key to continue.
```

Figure 37: Ejecucion del programa

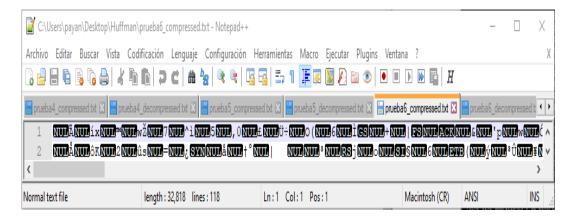


Figure 38: Archivo de salida

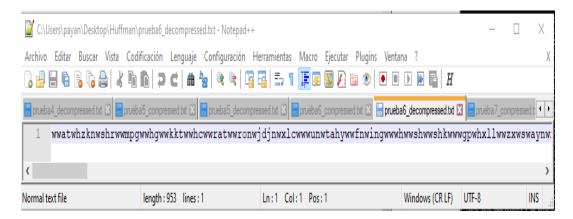


Figure 39: Archivo descomprimido

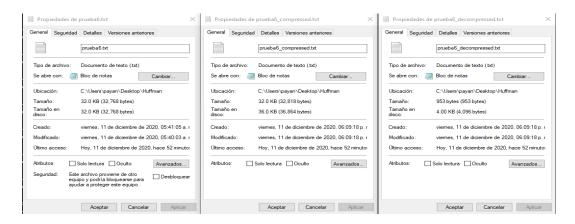


Figure 40: peso de los 3 archivos

Septimo archivo

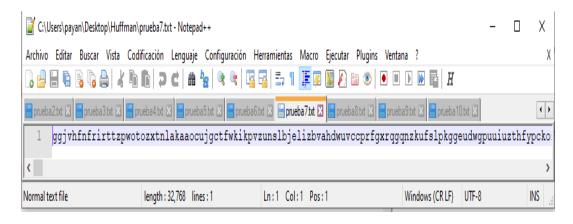


Figure 41: Archivo de entrada

```
C:\Users\payan\Desktop\Huffman\bin\Debug\Huffman.exe
 Tiene 1291 apariciones y su binario es: 0000
 Tiene 1273 apariciones y su binario es: 11011
 Tiene 1212 apariciones y su binario es: 01101
 Tiene 1301 apariciones y su binario es: 0011
 Tiene 1229 apariciones y su binario es: 10010
 Tiene 1342 apariciones y su binario es: 0101
 Tiene 1226 apariciones y su binario es: 10000
 Tiene 1236 apariciones y su binario es: 10101
 Tiene 1234 apariciones y su binario es: 10011
 Tiene 1313 apariciones y su binario es: 0100
 Tiene 1247 apariciones y su binario es: 10110
 Tiene 1260 apariciones y su binario es: 11000
 Tiene 1297 apariciones y su binario es: 0010
 Tiene 1263 apariciones y su binario es: 11001
 Tiene 1275 apariciones y su binario es: 11101
 Tiene 1290 apariciones y su binario es: 11111
 Tiene 1253 apariciones y su binario es: 10111
 Tiene 1280 apariciones y su binario es: 11110
 Tiene 1212 apariciones y su binario es: 01110
 Tiene 1293 apariciones y su binario es: 0001
Tiene 1270 apariciones y su binario es: 11010
 Tiene 1211 apariciones y su binario es: 01100
 Tiene 1274 apariciones y su binario es: 11100
Tiene 1236 apariciones y su binario es: 10100
 Tiene 1227 apariciones y su binario es: 10001
: Tiene 1223 apariciones y su binario es: 01111
Se genero el arbol en 164382 operaciones
e comprimio el archivo: prueba7.txt en prueba7_compressed.txt en 7349301 operaciones
e descomprimio el archivo: prueba7_compressed.txt en prueba7_decompressed.txt en 670 operaciones
Process returned 0 (0x0)
                            execution time : 0.222 s
ress any key to continue.
```

Figure 42: Ejecucion del programa

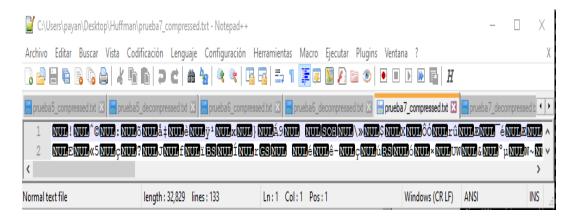


Figure 43: Archivo de salida

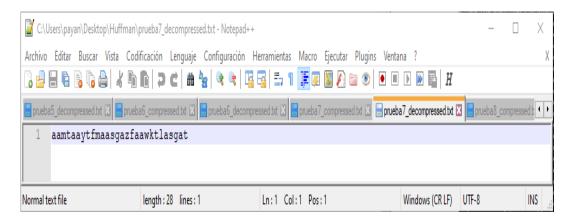


Figure 44: Archivo descomprimido

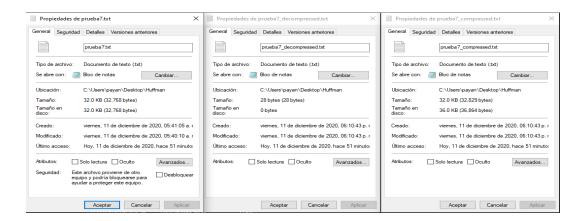


Figure 45: peso de los 3 archivos

Octavo archivo



Figure 46: Archivo de entrada

```
Tiene 1291 apariciones y su binario es: 0000
b Tiene 1273 apariciones y su binario es: 10101
c Tiene 1212 apariciones y su binario es: 10101
d Tiene 1201 apariciones y su binario es: 0001
d Tiene 1301 apariciones y su binario es: 0001
d Tiene 1302 apariciones y su binario es: 0001
f Tiene 1302 apariciones y su binario es: 0001
g Tiene 1236 apariciones y su binario es: 10000
h Tiene 1236 apariciones y su binario es: 10000
h Tiene 1236 apariciones y su binario es: 10000
h Tiene 1236 apariciones y su binario es: 10000
h Tiene 1236 apariciones y su binario es: 10000
m Tiene 1247 apariciones y su binario es: 10100
m Tiene 1246 apariciones y su binario es: 10100
m Tiene 1246 apariciones y su binario es: 10100
m Tiene 1247 apariciones y su binario es: 10000
m Tiene 1247 apariciones y su binario es: 10000
m Tiene 1249 apariciones y su binario es: 10000
m Tiene 1249 apariciones y su binario es: 10101
p Tiene 1280 apariciones y su binario es: 10110
p Tiene 1280 apariciones y su binario es: 10111
r Tiene 1280 apariciones y su binario es: 0001
u Tiene 1273 apariciones y su binario es: 0001
u Tiene 1274 apariciones y su binario es: 0001
u Tiene 1274 apariciones y su binario es: 0001
u Tiene 1274 apariciones y su binario es: 0001
u Tiene 1274 apariciones y su binario es: 0001
u Tiene 1274 apariciones y su binario es: 0001
u Tiene 1274 apariciones y su binario es: 0001
u Tiene 1274 apariciones y su binario es: 0001
u Tiene 1274 apariciones y su binario es: 00100
y Tiene 1271 apariciones y su binario es: 00100
y Tiene 1272 apariciones y su binario es: 00101
Se genero el archivo: prueba8.txt en prueba8_compressed.txt en 7349935 operaciones
Se comprimio el archivo: prueba8.txt en prueba8_decompressed.txt en 12946 operaciones
Se coscomprimio el archivo: prueba8.txt en prueba8_decompressed.txt en 12946 operaciones
Se descomprimio el archivo: prueba8.txt en prueba8_decompressed.txt en 12946 operaciones
```

Figure 47: Ejecucion del programa

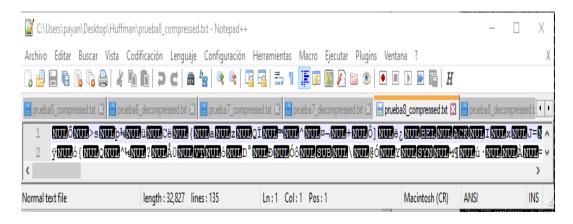


Figure 48: Archivo de salida



Figure 49: Archivo descomprimido

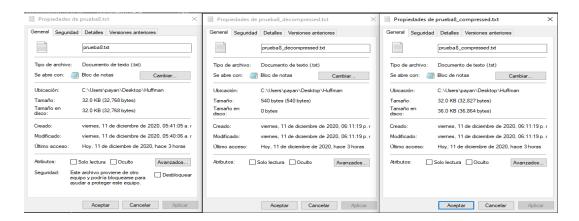


Figure 50: peso de los 3 archivos

Noveno archivo

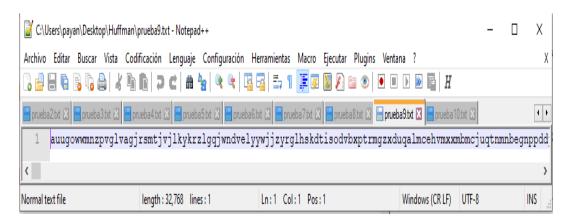


Figure 51: Archivo de entrada

```
C:\Users\payan\Desktop\Huffman\bin\Debug\Huffman.exe
 Tiene 1291 apariciones y su binario es: 0000
 Tiene 1273 apariciones y su binario es: 11011
 Tiene 1212 apariciones y su binario es: 01101
 Tiene 1301 apariciones y su binario es: 0011
 Tiene 1229 apariciones y su binario es: 10010
 Tiene 1342 apariciones y su binario es: 0101
 Tiene 1226 apariciones y su binario es: 10000
 Tiene 1236 apariciones y su binario es: 10101
 Tiene 1234 apariciones y su binario es: 10011
 Tiene 1313 apariciones y su binario es: 0100
 Tiene 1247 apariciones y su binario es: 10110
 Tiene 1260 apariciones y su binario es: 11000
 Tiene 1297 apariciones y su binario es: 0010
 Tiene 1263 apariciones y su binario es: 11001
Tiene 1275 apariciones y su binario es: 11101
 Tiene 1290 apariciones y su binario es: 11111
 Tiene 1253 apariciones y su binario es: 10111
Tiene 1280 apariciones y su binario es: 11110
 Tiene 1212 apariciones y su binario es: 01110
 Tiene 1293 apariciones y su binario es: 0001
Tiene 1270 apariciones y su binario es: 11010
 Tiene 1211 apariciones y su binario es: 01100
 Tiene 1274 apariciones y su binario es: 11100
Tiene 1236 apariciones y su binario es: 10100
 Tiene 1227 apariciones y su binario es: 10001
z Tiene 1223 apariciones ý su binario es: 01111
Se genero el arbol en 164382 operaciones
e comprimio el archivo: prueba9.txt en prueba9 compressed.txt en 7350394 operaciones
e descomprimio el archivo: prueba8_compressed.txt en prueba9_decompressed.txt en 4310 operaciones
Process returned 0 (0x0)
                               execution time : 0.217 s
ress any key to continue.
```

Figure 52: Ejecucion del programa



Figure 53: Archivo de salida

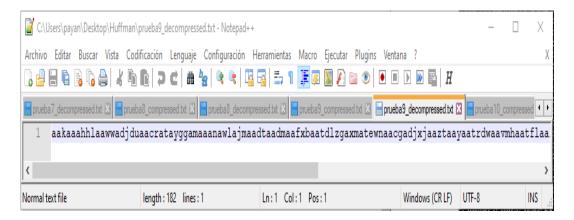


Figure 54: Archivo descomprimido

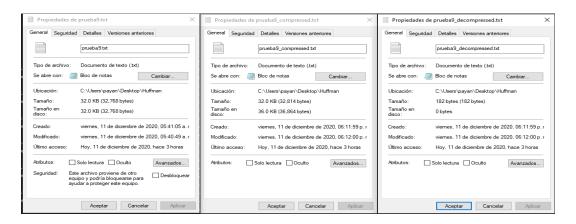


Figure 55: peso de los 3 archivos

Decimo archivo

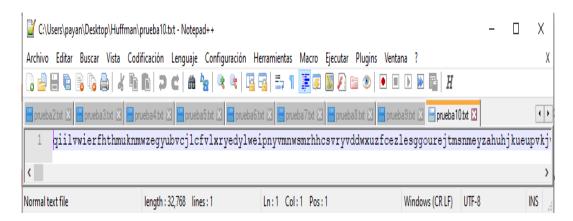


Figure 56: Archivo de entrada

```
C:\Users\payan\Desktop\Huffman\bin\Debug\Huffman.exe
 Tiene 1291 apariciones y su binario es: 0000
 Tiene 1273 apariciones y su binario es: 11011
 Tiene 1212 apariciones y su binario es: 01101
Tiene 1301 apariciones y su binario es: 0011
 Tiene 1229 apariciones y su binario es: 10010
 Tiene 1342 apariciones y su binario es: 0101
Tiene 1226 apariciones y su binario es: 10000
 Tiene 1236 apariciones y su binario es: 10101
 Tiene 1234 apariciones y su binario es: 10011
 Tiene 1313 apariciones y su binario es: 0100
 Tiene 1247 apariciones y su binario es: 10110
 Tiene 1260 apariciones y su binario es: 11000
 Tiene 1297 apariciones y su binario es: 0010
 Tiene 1263 apariciones y su binario es: 11001
 Tiene 1275 apariciones y su binario es: 11101
 Tiene 1290 apariciones y su binario es: 11111
 Tiene 1253 apariciones y su binario es: 10111
Tiene 1280 apariciones y su binario es: 11110
 Tiene 1212 apariciones y su binario es: 01110
 Tiene 1293 apariciones y su binario es: 0001
Tiene 1270 apariciones y su binario es: 11010
 Tiene 1211 apariciones y su binario es: 01100
 Tiene 1274 apariciones y su binario es: 11100
Tiene 1236 apariciones y su binario es: 10100
 Tiene 1227 apariciones y su binario es: 10001
z Tiene 1223 apariciones y su binario es: 01111
Se genero el arbol en 164382 operaciones
e comprimio el archivo: prueba10.txt en prueba10_compressed.txt en 7350437 operaciones
e descomprimio el archivo: prueba8 compressed.txt en prueba10 decompressed.txt en 27526 operaciones
Process returned 0 (0x0) execution time : 0.216 s
ress any key to continue.
```

Figure 57: Ejecucion del programa

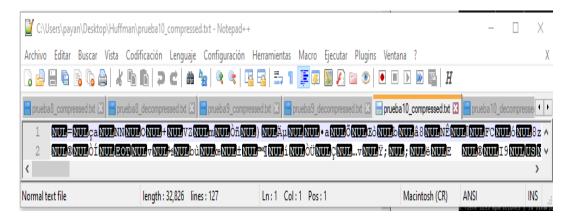


Figure 58: Archivo de salida



Figure 59: Archivo descomprimido

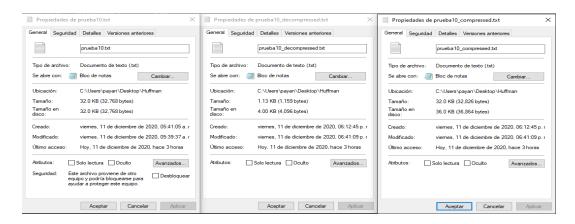


Figure 60: peso de los 3 archivos

Finalmente para concluir este algorimo, se realizo una modificacion al codigo para generar una cadena aleatoria de longitud "n", con ($1 \le n \le 10000$), posteriormente a esta se le genero el arbol, se comprimio y descomprimio, para poder obtener la grafica de complejidad de los algorimos de compresion y descompresion de Huffman.

```
□ C\User\payan\Desktop\Huffman\text{binDebug\Huffman.exe} - □ X

a Tiene 1291 apariciones y su binario es: 0000
b Tiene 1273 apariciones y su binario es: 10101
c Tiene 1212 apariciones y su binario es: 0010
d Tiene 1301 apariciones y su binario es: 0011
d Tiene 1301 apariciones y su binario es: 0010
f Tiene 1342 apariciones y su binario es: 0010
f Tiene 1342 apariciones y su binario es: 01010
f Tiene 1343 apariciones y su binario es: 01010
i Tiene 1343 apariciones y su binario es: 10101
i Tiene 1236 apariciones y su binario es: 10101
i Tiene 1246 apariciones y su binario es: 10100
k Tiene 1247 apariciones y su binario es: 10100
m Tiene 1297 apariciones y su binario es: 10100
m Tiene 1297 apariciones y su binario es: 10101
n Tiene 1266 apariciones y su binario es: 11001
n Tiene 1263 apariciones y su binario es: 11001
n Tiene 1263 apariciones y su binario es: 11001
n Tiene 1263 apariciones y su binario es: 11001
n Tiene 1293 apariciones y su binario es: 11101
p Tiene 1290 apariciones y su binario es: 11110
t Tiene 1200 apariciones y su binario es: 11110
t Tiene 1200 apariciones y su binario es: 11110
t Tiene 1203 apariciones y su binario es: 11010
t Tiene 1270 apariciones y su binario es: 11010
t Tiene 1274 apariciones y su binario es: 11010
t Tiene 1275 apariciones y su binario es: 11010
t Tiene 1274 apariciones y su binario es: 11000
w Tiene 1274 apariciones y su binario es: 11000
w Tiene 1273 apariciones y su binario es: 11000
w Tiene 1273 apariciones y su binario es: 11000
w Tiene 1273 apariciones y su binario es: 11000
x Tiene 1273 apariciones y su binario es: 11000
x Tiene 1273 apariciones y su binario es: 10000
y Tiene 1273 apariciones y su binario es: 10000
y Tiene 1273 apariciones y su binario es: 10000
x Tiene 1273 apariciones y su binario es: 10000
y Tiene 1273 apariciones y su binario es: 10000
x Tiene 1274 apariciones y su binario es: 10000
x Tiene 1275 apariciones y su binario es: 10000
x Tiene 1275 apariciones y su binario es: 10000
x Tiene 1276 apariciones y su binario es: 10000
x Tiene 1276 apar
```

Figure 61: Ejecucion del programa generando cadenas aleatorias, comprimiendolas y descomprimiendolas

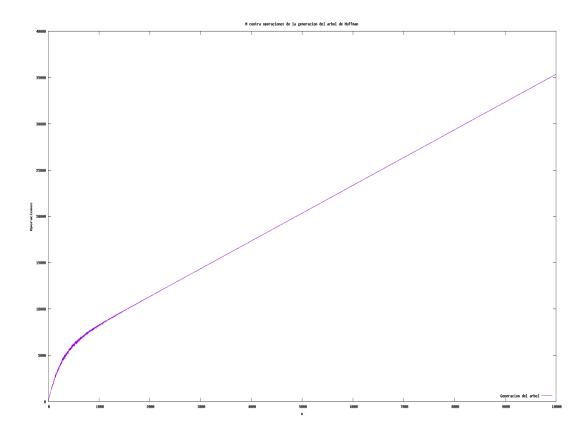


Figure 62: N contra operaciones de la generacion del arbol de Huffman

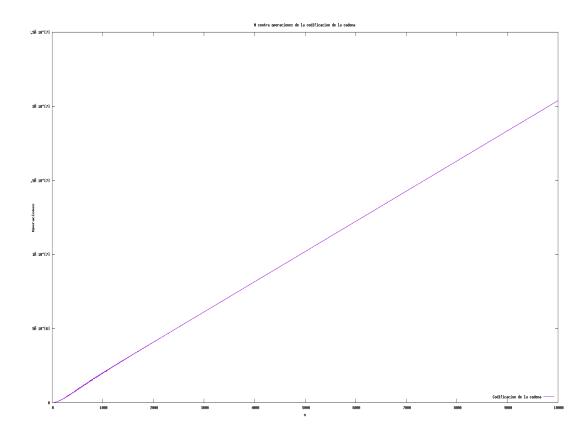


Figure 63: N contra operaciones de la codificacion de la cadena

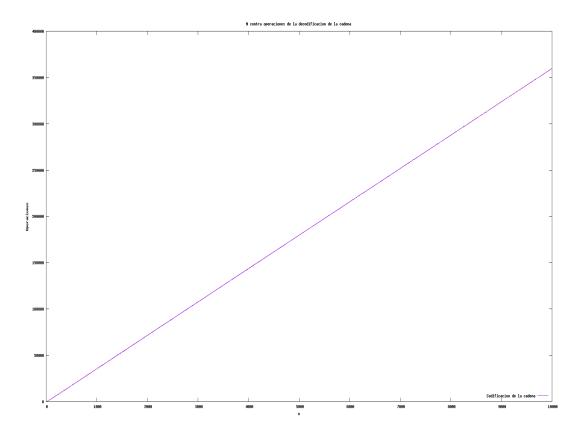


Figure 64: N contra operaciones de la decodificacion de la cadena

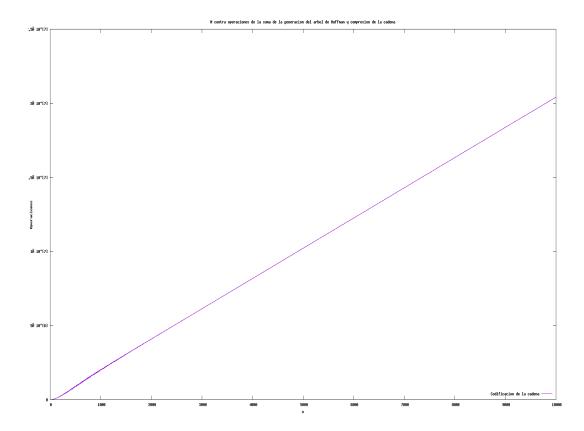


Figure 65: N contra operaciones de la suma de la generacion del arbol de Huffman y compresion de la cadena

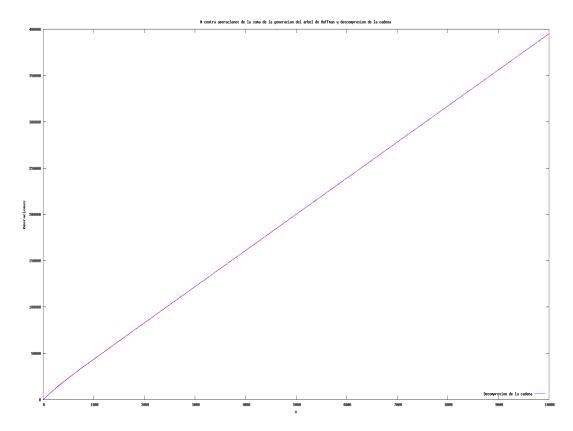


Figure 66: N contra operaciones de la suma de la generacion del arbol de Huffman y descompresion de la cadena

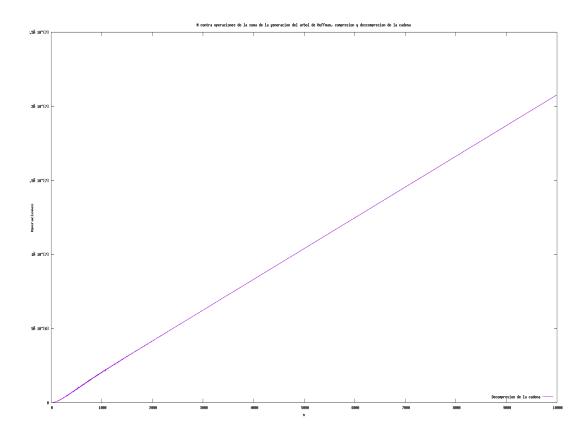


Figure 67: N contra operaciones de la suma de la generacion del arbol de Huffman, compresion y descompresion de la cadena

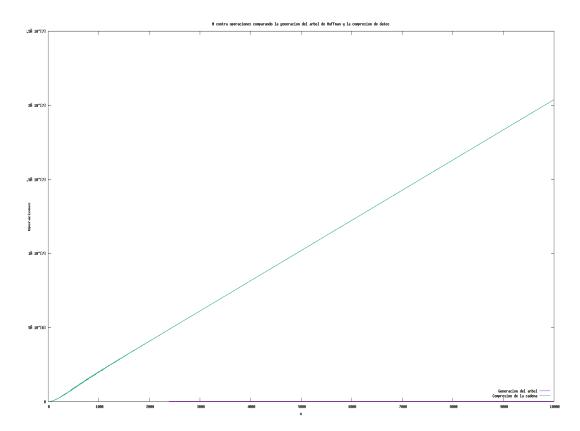


Figure 68: N contra operaciones comparando la generacion del arbol de Huffman y la compresion de datos

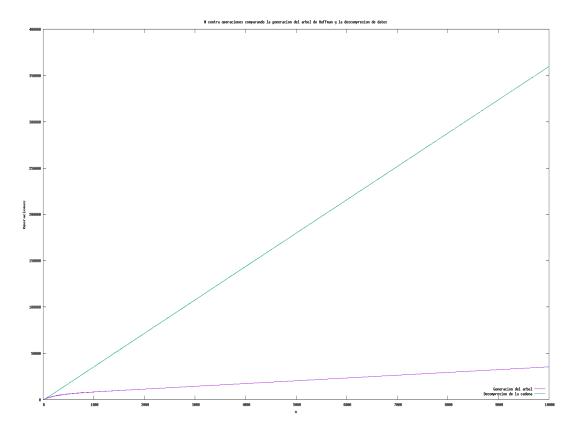


Figure 69: N contra operaciones comparando la generacion del arbol de Huffman y la descompresion de datos

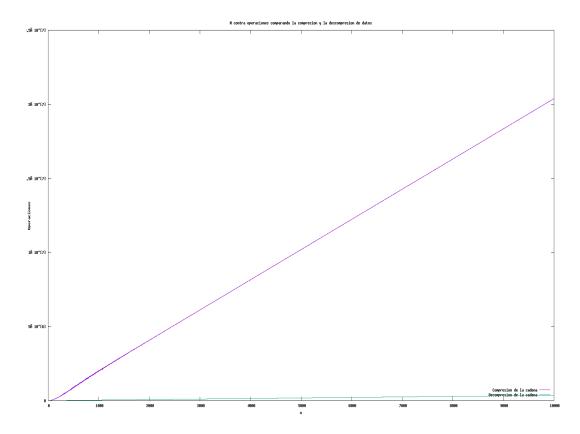


Figure 70: N contra operaciones comparando la compresion y la descompresion de datos

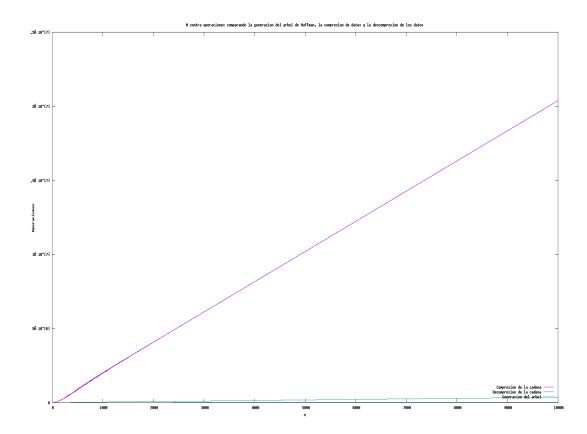


Figure 71: N contra operaciones comparando la generación del arbol de Huffman, la compresión de datos y la descompresión de los datos

Para realizar la comparacion correspondiente, se realiza el calculo de complejidad a priori usando el analisis linea a linea del pseudo codigo. primero la funcion **HuffmanTree(C)**

Codigo	Costo	Veces ejecutado
n = C.size()	Ø(1)	1
$Q=priority_queue()$	Ø(1)	1
$for(i=0;i\leq n;i++)$	$\emptyset(n)$	n+1
n = node(C[i])	Ø(1)	n
Q.push(n)	$\emptyset(log_2(n-i))$	n
while(Q.size()>1)	$\emptyset(log_2(n))$	$log_2(n) + 1$
$z = new \ node()$	Ø(1)	$log_2(n)$
z.left = x = Q.pop()	$\emptyset(log_2(n))$	$log_2(n)$
z.right = y = Q.pop()	$\emptyset(log_2(n))$	$log_2(n)$
z.frequency = x.frequency + y.frequency	Ø(1)	$log_2(n)$
Q.push(z)	$\emptyset(log_2(n))$	$log_2(n)$
$return \ Q.pop()$	Ø(1)	1

Como podemos observar la mayor complejidad se encuentra dentro de los ciclos, donde en el peor escenario llega a ser $\mathcal{O}(nlog_2(n))$, considerando lo anterior se grafica la funcion $f(x) = xlog_2(x)$ y se anexa la grafica para ser comparada con los resultados del programa.

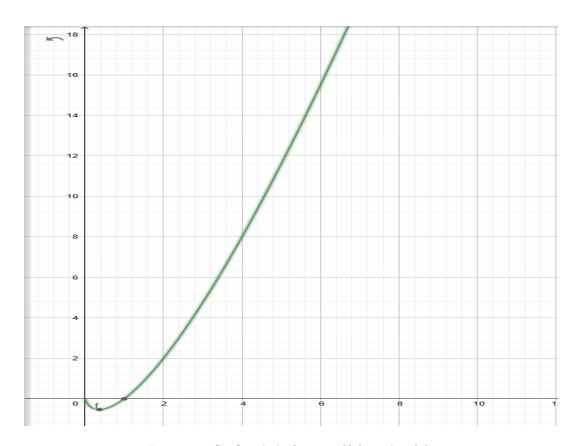


Figure 72: Grafica de la funcion $f(x) = xlog_2(x)$

Dada la grafica anterior, el analisis a posteriori y las graficas: Figure 62 y Figure 67, podemos concluir que la funcion para generar el arbol de Huffman tiene complejidad de $nlog_2(n)$ Ahora se realiza el calculo de complejidad de la funcion **HuffmanDecompression(tree, S)**

Codigo	Costo	Veces ejecutado
n = S.length()	Ø(1)	1
retorno=""	Ø(1)	1
$for(i=0;i\leq n;i++)$	$\emptyset(n)$	n+1
actual=tree	Ø(1)	n
while(actual.left!=NULL and actual.right!=NULL)	$\emptyset(log_2(255))$	n
if(S[i] == '0')	Ø(1)	$log_2(255) * n$
actual = actual.left	Ø(1)	$\frac{log_2(255)*n}{2}$
else	Ø(1)	$log_2(255) * n$
actual = actual.right	Ø(1)	$\frac{log_2(255)*n}{2}$
i++	Ø(1)	$log_2(255) * n$
retorno+=actual.symbol	Ø(1)	n
return retorno	Ø(1)	1

Entonces despues de analizar la funcion, podemos concluir que la funcion tiene complejdad $\emptyset(n)$ siendo n la cantidad de bits en la cadena a descomprimir.

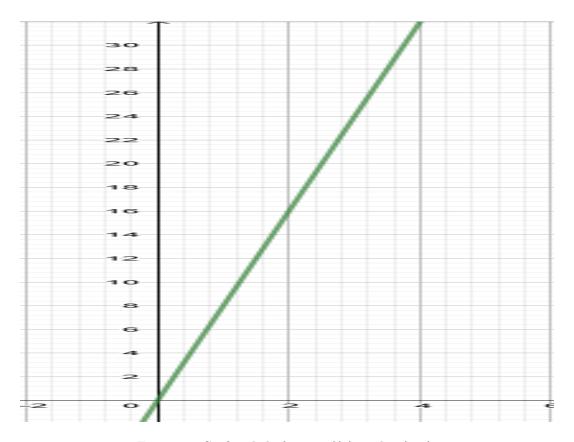


Figure 73: Grafica de la funcion $f(x) = xlog_2(255)$

Comparando la grafica de la funcion (Figure 73) y las graficas Figure 64, Figure 66, Figure 67, Figure 70, Figure 71, se verifica que la funcion tiene orden de complejidad lineal.

3.2 Implementar los algoritmos de Kruskal

Como ya conocemos la complejidad del algoritmo, no es necesario realizar el analisis a priori, por lo tanto graficamos directamente la complejidad $\mathcal{O}(mlog_2(n))$, considerando m = n.

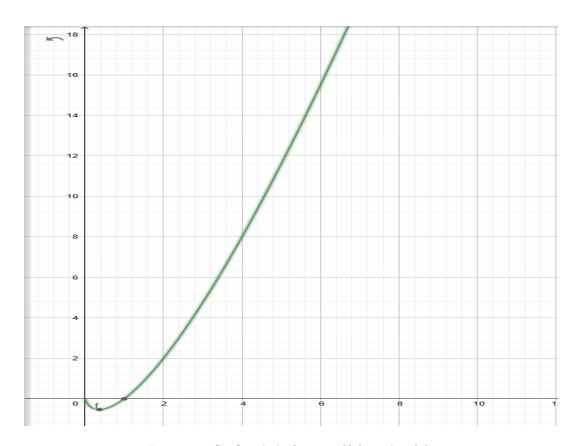


Figure 74: Grafica de la funcion $f(x) = xlog_2(x)$

Ahora realizamos la implementacion del algoritmo para calcular su complejidad a posteriori. Se generan arboles aleatorios de n nodos y n aristas, donde ($1 \le n \le 10000$) y graficamos la correspondiente informacion para validar la proposicion de complejidad.

```
□ C\Users\payan\Desktop\kruskal\bin\Debug\kruskaleve

- □ X

2542 - 455
234 - 673
329 - 1856
521 - 1940
638 - 2319
1078 - 1328
1365 - 2370
1491 - 1643
1528 - 2339
1644 - 4,748
1727 - 1513
1849 - 1950
2238 - 1106
2342 - 571
2389 - 359
2478 - 1616
258 - 1424
430 - 2099
1837 - 1076
1515 - 1573
1651 - 1467
1766 - 119
1852 - 1293
2160 - 1280
2239 - 46
110 - 994
353 - 761
555 - 154
992 - 2471
1019 - 1
```

Figure 75: Ejecucion del algoritmo programado

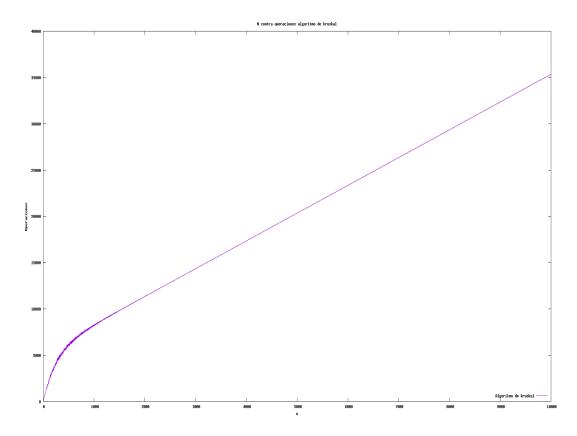


Figure 76: Grafica de n contra operaciones del algoritmo de Kruskal

4 Conclusiones

4.1 Payán Téllez René

Durante el desarrollo de esta practica hubo muchas cosas inesperadas, como el hecho de que por usar la STL no se pudo medir de forma eficiente la cantidad de operaciones, o que al momento de generar los archivos comprimidos, el metodo de encodeo de C/C++ los mantuvo relativamente del mismo tamaño que los originales. Tambien ocurrio que al momento de guardar el archivo comprimido y volverlo a leer sucedieron varios errores (se puede ver en los tamaños de los archivos), ya que se guardaba el caracter "EOF" o "End Of File", probocando que las funciones getfc dejaran de leer al momento de importar, esto sucedia ya que este caracter era uno posible por su codigo ascii (0x05), asi que se genero siempre que existiera la combinacion binaria "0000101". Aunque fue bastante interesar leer sobre el algoritmo de Kruskal, ya que es algo que estamos haciendo a mano en la clase de Metodos Cuantitativos para la Toma de Decisiones, entonces al menos con eso se que uno de sus usos en la vida real es el analisis de costos y resolucion del menor costo en un proyecto o menor tiempo, tambien sirve para encontrar la ruta critica.

5 Anexo

6 Bibliografia

```
[1]http://www.lcc.uma.es/~av/Libro/CAP3.pdf
[2]https://medium.com/@joseguillermo_/qu%C3%A9-es-la-complejidad-algor%C3%ADtmica-y-con-qu%C3%A9-se-come-2638e7fd9e8c
[3]https://www.tamps.cinvestav.mx/~ertello/algorithms/sesion15.pdf
[4]http://elvex.ugr.es/decsai/algorithms/slides/4%20greedy.pdf
[5]https://riptutorial.com/es/algorithm/example/23995/codificacion-huffman
[6]https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal
[7]https://www.wextensible.com/temas/voraces/kruskal-prim.html
[8]https://www.ecured.cu/Algoritmo_de_Kruskal
```