INSTITUTO POLITÉCNICO NACIONAL

"ESCOM" (ESCUELA SUPERIOR DE CÓMPUTO)

DOCENTE: CECILIA ALBORTANTE MORATO

INTEGRANTES:

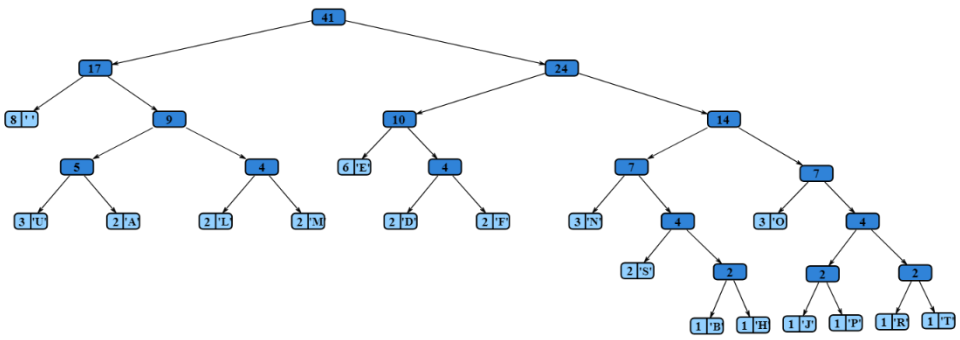
MONTEALEGRE ROSALES DAVID URIEL

VÁZQUEZ BLANCAS CÉSAR SAID

U.A.: ANÁLISIS Y DISEÑO DE ALGORITMOS

GRUPO: 3CM6

PRÁCTICA 5. Códigos de Huffman.



Objetivo de la práctica.

Implementar en el lenguaje de programación de su preferencia el algoritmo de Códigos de Huffman.

Indicaciones solicitadas.

Pedir al usuario un texto, realizar el conteo de los caracteres para obtener sus frecuencias y a partir de esa entrada obtener el código de Huffman óptimo por medio del algoritmo voraz.

❖ Imprimir el código de longitud variable obtenido para cada carácter y el costo total del árbol.

❖ Implementar la codificación de la entrada con los códigos de Huffman obtenidos.

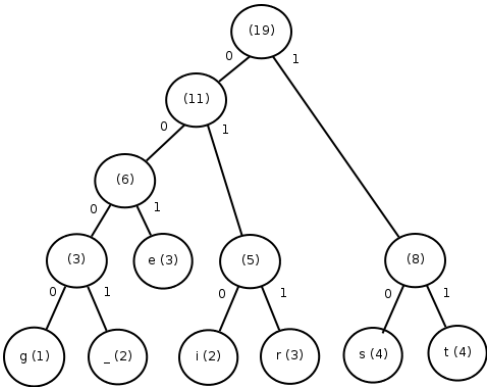
❖ Implementar la decodificación de la entrada desde la cadena binaria comprimida.

Introducción.

Los algoritmos de Huffman son técnicas de compresión de datos desarrolladas por David A. Huffman en 1952. Estos algoritmos se utilizan para reducir el tamaño de los datos, especialmente en el contexto de la compresión de información en archivos digitales. La idea central detrás de los algoritmos de Huffman es asignar códigos de longitud variable a diferentes símbolos en función de su frecuencia de aparición en los datos.

En esencia, Huffman logra comprimir datos asignando códigos cortos a símbolos frecuentes y códigos más largos a símbolos menos frecuentes. Esto aprovecha la redundancia en los datos, ya que los símbolos más comunes se representan de manera más eficiente. La estructura del árbol de Huffman, que se construye durante el proceso, permite una decodificación eficiente y sin ambigüedades de los datos comprimidos.

La aplicación de los algoritmos de Huffman es amplia y se utiliza en numerosas aplicaciones, como la compresión de imágenes, audio, video y archivos en general. Su eficiencia radica en su capacidad para adaptarse a la frecuencia de ocurrencia de los símbolos en los datos, lo que los convierte en una herramienta valiosa para la optimización del espacio de almacenamiento y la transmisión de información.



Desarrollo de la práctica.

Para la resolución de la siguiente práctica fue necesario implementar un código diseñado en lenguaje C, el cual implementa la codificación y decodificación del conocido **"código de Huffman"**, el cual implementa una estructura similar a la de un árbol binario y su principal objetivo es reducir el tamaño de los datos. El código diseñado es el siguiente:

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define EMPTY_STRING ""

// crea el nodo del arbol
struct Node
{
    char ch;
    int freq;
    struct Node *left, *right;
};

struct Node* getNode(char ch, int freq, struct Node* left, struct Node* right)
{
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));//asigna memoria a un arbol

    node->ch = ch;
    node->freq = freq;//asigna los valores al arbol
    node->left = left;
    node->right = right;

    return node;//retorna el nodo
}

int unnodo(struct Node* root) {
    return root->left == NULL && root->right == NULL;//verifica si el arbol de huffman tiene un solo
noso
}

// Atraviesa el árbol de Huffman y almacena los códigos de Huffman en un mapa.
void codificar(struct Node* root, char* str, char** huffman)
{
    if (root == NULL) {
        return;//si no hay raiz, no devuelve nada
    }

    if (unnodo(root)) { //si hay un nodo
        if (strcmp(str, EMPTY_STRING) != 0) {
            strcpy(huffman[root->ch], str);//copia los valores en hoffman
        } else {
```

```
            strcpy(huffman[root->ch], "1");// agrega un uno
        }
    }

    char left[256];
    char right[256];
    strcpy(left, str);//copia los valores obtenidos en un arreglo left
    strcpy(right, str);//copia los valores obtenidos en un arreglo right

    strcat(left, "0");//agrega un 0
    strcat(right, "1");//agrega un 1

    codificar(root->left, left, huffman);//recursividad donde los valores se envian de nuevo
    codificar(root->right, right, huffman);//recursividad donde los valores se envian de nuevo pra la
otra parte del arbol
}

// Atraviesa el árbol de Huffman y decodifica la string codificada
void decodificar(struct Node* root, int* i, char* str)
{
    if (root == NULL) {// si no hya raiz no devuelve nada
        return;
    }

    if (unnodo(root)) // si hya un nodo
    {
        printf("%c", root->ch);//imprime el valor de la letra
        return;
    }

    (*i)++; //incrementa el apuntador para el siguiente nodo

    if (str[*i] == '0') {// si el contenido del nodo es 0
        decodificar(root->left, i, str);//manda a llamar a decodificar recursivamente por la izquierda
    }
    else {
        decodificar(root->right, i, str);//manda a llamar a decodificar recursivamente por la derecha
    }
}

// Construye Huffman y decodifica el texto de entrada dado
void huffman(char* a)
{
    if (strcmp(a, EMPTY_STRING) == 0) {//si no hay cadena, no devuelve nada
        return;
    }

    int freq[256] = {0};
    int i = 0;
```

```
   while (a[i] != '\0') {//recorre el texto hasta el final
      freq[(int)a[i]]++;//cada vez que hay una letra diferente, aumenta el valor de la posicion del arreglo en la frecuencia
      i++;
   }

   char* h[256];
   for (int j = 0; j < 256; j++) {
      h[j] = (char*)malloc(256 * sizeof(char));// crea el arreglo y alamacena los codigos
   }

   struct Node* pq[256];// crea cola de prioridad
   int pqSize = 0;

   for (int k = 0; k < 256; k++) {
      if (freq[k] > 0) {
         pq[pqSize++] = getNode((char)k, freq[k], NULL, NULL);// crea nodos de la cola y almacena las partes del arbol
      }
   }
   while (pqSize > 1)
   {
      int m1 = 0, m2 = 1;
      if (pq[0]->freq > pq[1]->freq) {//si la frecuencia es mayor la de,l primero que la del segundo
         m1 = 1;//asugna un 1
         m2 = 0;//asigna un 0
      }

      for (int l = 2; l < pqSize; l++) {//recorre la cola
         if (pq[l]->freq < pq[m1]->freq) {
            m2 = m1;//asigna los valores dependiendo al frecuencia del nodo
            m1 = l;
         } else if (pq[l]->freq < pq[m2]->freq) {
            m2 = l;//hace intercambios para encontrar los mas pequeños
         }
      }

      struct Node* left = pq[m1];//asigna al arbol el nodo menor a la izquierda
      struct Node* right = pq[m2];//asigna al arbol el nodo 2do menor a la derecha
      pq[m1] = getNode('\0', left->freq + right->freq, left, right);//crea el nodo con los hijos ya asignados que son los menores

      // Elimina el nodo de menor frecuencia de la queue
      pq[m2] = pq[--pqSize];
   }

   //  almacena el puntero a la raíz de Huffman
   struct Node* root = pq[0];
```

```
    codificar(root, EMPTY_STRING, h);//codificamos la cadena

    printf("Los codigos son :\n\n");
    for (int m = 0; m < 256; m++) {
        if (freq[m] > 0) { //el arreglo de frecuencias se imprime como cadena
            printf("%c %s\n", (char)m, h[m]);
        }
    }

    printf("\nLa cadena original es :\n%s\n", a);
    char str[256] = {0};
    i = 0;
    while (a[i] != '\0') {
        strcat(str, h[(int)a[i]]);//imprimimos la cadena codificada
        i++;
    }

    printf("\nLa cadena codificada es :\n%s\n", str);
    printf("\nLa cadena decodificada es:\n");

    if (unnodo(root))
    {
        // caso en el cual se repite la letra
        while (root->freq--) {
            printf("%c", root->ch);//imprimimos la misma letra
        }
    }
    else {
        int i = -1;
        while (i < (int)strlen(str) - 1) {//mandamos la cadena codificada hasta su fin
            decodificar(root, &i, str);//decodificamos letra por letra
        }
    }

    // libera la memoria
    for (int n = 0; n < 256; n++) {
        free(h[n]);
    }
}

int main()
{
    char a[] = "abracadabra";
    printf("Introduce una cadena: ");
    scanf("%s",a);
    huffman(a);
    return 0;
}
```

1. Struct node: Define la estructura de un nodo del árbol de Huffman. Cada nodo tiene un carácter ch, una frecuencia freq y dos punteros a nodos izquierdo y derecho (left y right), que representan los hijos en el árbol binario.

2. getNode: Esta función crea y devuelve un nuevo nodo del árbol con los parámetros dados.

3. unnodo: Devuelve 1 si el nodo es una hoja (no tiene hijos), y 0 en caso contrario.

4. codificar: Esta función recorre el árbol de Huffman y almacena los códigos Huffman en un mapa (huffman). Utiliza la recursividad para atravesar el árbol y construir los códigos.

5. decodificar: Esta función decodifica la cadena codificada y la imprime en la salida estándar. Utiliza la recursividad para atravesar el árbol y decodificar cada caracter.

6. huffman: Esta función principal construye el árbol de Huffman y realiza la codificación y decodificación. Imprime los códigos Huffman, la cadena original, la cadena codificada y la cadena decodificada.

7. main: Se trata de nuestra función principal del programa, esta será la encargada de solicitar al usuario una palabra o frase (cade de caracteres) para que posteriormente llame a la función huffman y comience la ejecución del conocido "código de Huffman".

Resultados.

Aquí se puede observar el funcionamiento correcto del programa:

```
C:\Users\alexl\Downloads>gcc Huffman.c -o huffman

C:\Users\alexl\Downloads>huffman
Introduce una cadena: caramba
Los codigos son :

a 0
b 100
c 101
m 111
r 110

La cadena original es :
caramba

La cadena codificada es :
101011001111000

La cadena decodificada es:
caramba
C:\Users\alexl\Downloads>
```

```
C:\Users\alexl\Downloads>huffman
Introduce una cadena: americano
Los codigos son :

a 111
c 000
e 001
i 011
m 101
n 110
o 100
r 010

La cadena original es :
americano

La cadena codificada es :
111101001010011000111110100

La cadena decodificada es:
americano
C:\Users\alexl\Downloads>
```

```
C:\Users\alexl\Downloads>huffman
Introduce una cadena: informatica
Los codigos son :

a 101
c 1100
f 1101
i 01
m 1111
n 001
o 100
r 000
t 1110

La cadena original es :
informatica

La cadena codificada es :
0100111011000001111101111001110100101

La cadena decodificada es:
informatica
```

Conclusión.

En resumen, el código de Huffman ha demostrado ser una herramienta valiosa en la optimización de la eficiencia de almacenamiento y transmisión de datos al asignar códigos de longitud variable de manera inteligente según la frecuencia de los elementos en el conjunto de datos. Su simplicidad y eficacia lo convierten en una opción popular para diversas aplicaciones de compresión de datos.