



**INSTITUTO POLITÉCNICO NACIONAL**  
**Escuela Superior de Cómputo**



**Análisis y diseño de algoritmos**

**Ejercicio 7**  
**“Codificación Huffman”**

**Gil Juárez Hector David**

**3CV1**

**Mayo 10, 2024**

## Codificación Huffman

Es un método general de codificación y compresión diseñado para minimizar el número medio de bits necesarios para transmitir un símbolo cuando se debe transmitir varias copias independientes y estadísticamente equivalentes de dicho símbolo. Este método determina cómo los distintos valores del símbolo deben representarse como cadenas binarias.

Supongamos que tenemos que enviar el símbolo  $X$  que puede tomar valores  $\{x_1, \dots, x_n\}$  con probabilidad  $\{p_1, \dots, p_n\}$ . La idea es reservar palabras cortas para los valores más frecuentes de  $X$ . Este método no requiere usar ningún tipo de separador entre los valores.

Ejemplo:  $x_1$  (0.5)       $x_2$  (0.3)       $x_3$  (0.15)       $x_4$  (0.05)

- Si se usan 00, 01, 10 y 11 necesitaremos siempre 2 bits para el valor de  $X$ .
- Si se usan las palabras 0, 10, 110, 111 necesitaremos como término medio 1.7 bits (menos de 2).

El código del ejemplo es de longitud variable, pero no se requiere usar ningún tipo de separador entre los valores.

- $x_3 x_1 x_4 x_3 x_3 x_2 = 110011111011010$  Sólo puede decodificarse correctamente.

La razón es que siempre puede reconocer el final de una palabra porque ninguna otra palabra es el principio de otra dada. Un código con esta propiedad se denomina código prefijo. El código Huffman es el código prefijo que requiere el mínimo número medio de bits por símbolo.

## Decodificación Huffman

El Decodificador de Huffman es una pieza crucial en la codificación de Huffman, un algoritmo popular y eficiente para la compresión de datos. Antes de profundizar en cómo funciona el decodificador, es esencial entender primero qué es la codificación de Huffman.

La codificación de Huffman, desarrollada por David Huffman en 1952, es un algoritmo de codificación de longitud variable que se utiliza para comprimir datos. Se basa en la idea de minimizar la longitud total de la codificación asignando códigos más cortos a los símbolos más frecuentes. De este modo, logra una compresión de datos eficaz.

El proceso de decodificación de Huffman es el inverso de la codificación. Para decodificar los datos, se necesita una tabla de códigos o un árbol de Huffman, que se genera durante la fase de codificación. Estos elementos se utilizan para identificar los códigos correspondientes a cada símbolo en el conjunto de datos original.

- **Tabla de Códigos:** Es una lista que asigna a cada símbolo un código único. En la decodificación, se utiliza para convertir los códigos en sus símbolos correspondientes.
- **Árbol de Huffman:** Es una estructura de árbol binario en la que cada nodo representa un símbolo y su frecuencia. Los nodos hoja representan los símbolos, y la ruta desde la raíz hasta cada nodo hoja forma el código de Huffman para el símbolo correspondiente.

La decodificación se realiza empezando desde la raíz del árbol de Huffman y siguiendo la ruta correspondiente al código binario hasta alcanzar un nodo hoja. Una vez que se llega a un nodo hoja, se toma el símbolo correspondiente a ese nodo y se inicia de nuevo desde la raíz para decodificar el siguiente símbolo.

Aunque el decodificador de Huffman es eficiente y efectivo, hay algunas consideraciones a tener en cuenta al usarlo. La principal es que la codificación y decodificación de Huffman es sensible a los errores. Si un solo bit es alterado durante la transmisión o el almacenamiento de los datos codificados, todo el proceso de decodificación puede fallar. Además, la codificación de Huffman no es tan eficiente para los conjuntos de datos que tienen una distribución de frecuencia uniforme.

## Código del ejercicio

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
typedef struct Nodo {
```

```
    char caracter;
```

```
    int frecuencia;
```

```
    struct Nodo *izquierda;
```

```
    struct Nodo *derecha;
```

```
} Nodo;
```

```
typedef struct {
```

```
    char caracter;
```

```

        char *codigo;
    } Codificacion;

void contar_frecuencias(char *texto, int *frecuencias) {
    for (int i = 0; texto[i] != '\0'; i++) {
        frecuencias[(int)texto[i]]++;
    }
}

Nodo* crear_nodo(char caracter, int frecuencia) {
    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    nuevo->caracter = caracter;
    nuevo->frecuencia = frecuencia;
    nuevo->izquierda = NULL;
    nuevo->derecha = NULL;
    return nuevo;
}

Nodo* construir_arbol_huffman(int *frecuencias) {
    Nodo* hojas[256];
    int num_hojas = 0;

    for (int i = 0; i < 256; i++) {
        if (frecuencias[i] > 0) {
            hojas[num_hojas++] = crear_nodo((char)i, frecuencias[i]);
        }
    }

    while (num_hojas > 1) {

        int indice_menor1 = 0;
        int indice_menor2 = 1;
        for (int i = 2; i < num_hojas; i++) {
            if (hojas[i]->frecuencia < hojas[indice_menor1]->frecuencia) {
                indice_menor2 = indice_menor1;
            }
        }
    }
}

```

```

        indice_menor1 = i;
    } else if (hojas[i]->frecuencia < hojas[indice_menor2]->frecuencia) {
        indice_menor2 = i;
    }
}

Nodo* interno = crear_nodo("\0", hojas[indice_menor1]->frecuencia + hojas[indice_menor2]->frecuencia);
interno->izquierda = hojas[indice_menor1];
interno->derecha = hojas[indice_menor2];

hojas[indice_menor1] = interno;
hojas[indice_menor2] = hojas[num_hojas - 1];
num_hojas--;
}

return hojas[0];
}

void generar_codificaciones(Nodo* nodo, char *codigo, Codificacion *codificaciones, int *index) {
    if (nodo->izquierda == NULL && nodo->derecha == NULL) {
        codificaciones[*index].caracter = nodo->caracter;
        codificaciones[*index].codigo = strdup(codigo);
        (*index)++;
    } else {
        int len = strlen(codigo);
        char *izquierda = (char*)malloc(len + 2);
        char *derecha = (char*)malloc(len + 2);
        strcpy(izquierda, codigo);
        strcpy(derecha, codigo);
        izquierda[len] = '0';
        izquierda[len + 1] = '\0';
        derecha[len] = '1';
        derecha[len + 1] = '\0';
        generar_codificaciones(nodo->izquierda, izquierda, codificaciones, index);
    }
}

```

```

        generar_codificaciones(nodo->derecha, derecha, codificaciones, index);
        free(izquierda);
        free(derecha);
    }
}

```

```

void huffman(char *texto) {

```

```

    int frecuencias[256] = {0};
    contar_frecuencias(texto, frecuencias);

```

```

    Nodo* raiz = construir_arbol_huffman(frecuencias);

```

```

    Codificacion codificaciones[256];
    int index = 0;
    generar_codificaciones(raiz, "", codificaciones, &index);

```

```

    printf("Codificaciones:\n");
    for (int i = 0; i < index; i++) {
        printf("Caracter: %c,Codigo: %s\n", codificaciones[i].caracter, codificaciones[i].codigo);
    }

```

```

    for (int i = 0; i < index; i++) {
        free(codificaciones[i].codigo);
    }
}

```

```

int main() {
    char texto[100];
    printf("Introduce el texto a codificar: ");
    fgets(texto, sizeof(texto), stdin);
    texto[strlen(texto)] = '\0';
    huffman(texto);
    return 0; }


```

```
C:\Users\hecto\Downloads\ejercicio7.exe
Introduce el texto a codificar: hoy esviernes diez de mayo
Codificaciones:
Caracter: r,Codigo: 00000
Caracter: v,Codigo: 00001
Caracter: z,Codigo: 00010
Caracter: n,Codigo: 00011
Caracter: ,Codigo: 0010
Caracter: a,Codigo: 0011
Caracter: o,Codigo: 010
Caracter: y,Codigo: 011
Caracter: e,Codigo: 10
Caracter: d,Codigo: 1100
Caracter: h,Codigo: 11010
Caracter: m,Codigo: 11011
Caracter: s,Codigo: 1110
Caracter: i,Codigo: 1111

Process returned 0 (0x0)  execution time : 22.872 s
Press any key to continue.
```

```
C:\Users\hecto\Downloads\ejercicio7.exe
Introduce el texto a codificar: holamundo
Codificaciones:
Caracter: a,Codigo: 0000
Caracter: d,Codigo: 0001
Caracter: u,Codigo: 001
Caracter: m,Codigo: 010
Caracter: n,Codigo: 011
Caracter: h,Codigo: 100
Caracter: l,Codigo: 101
Caracter: o,Codigo: 11

Process returned 0 (0x0)  execution time : 4.970 s
Press any key to continue.
```

 C:\Users\hecto\Downloads\ejercicio7.exe

Introduce el texto a codificar: ejemplo de codificacion huffman

Codificaciones:

Character: i, Código: 0000  
Character: n, Código: 00010  
Character: m, Código: 00011  
Character: p, Código: 0010  
Character: d, Código: 0011  
Character: h, Código: 01000  
Character: j, Código: 01001  
Character: u, Código: 01010  
Character: l, Código: 01011  
Character: , Código: 0110  
Character: a, Código: 0111  
Character: c, Código: 100  
Character: o, Código: 101  
Character: e, Código: 110  
Character: f, Código: 111

Process returned 0 (0x0) execution time : 11.524 s

Press any key to continue.