Algoritmo de huffman - decodificador

Formular una pregunta

2

Estoy realizando la parte de decodificar del algoritmo de huffman, como tal mi programa recibe dos archivos por consola, uno llamado "diccionariotxt" el cual contiene las letras, la frecuencia con la que salen y el codigo en binario de cada letra y otro llamado "codificadotxt" el cual contiene el codigo en binario a decodificar

Un ejemplo del archivo diccionario: H (1): 000 O (1): 001 L (1): 010 A (1): 011

Y un ejemplo del archivo codificado sera: 000001010011 (El cual deberia imprimir HOLA)

Mi problema es que cuando la palabra es de solo 3 letras NO lo decodifica por alguna razon que no logro ver, ademas de eso, el caracter espacio (" ") lo reconoce al momento de la traduccion como un salto de linea, imagino que debe ser algun problema teorico. Del resto el algoritmo funciona pero si podrian echarle un ojo se los agradeceria mucho, les anexo el codigo:

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

// MACRO PARA DECODIFICAR.
#define MAX_CARACTERES 255

// ESTRUCTURA PARA DECODIFICAR.
struct decod{
    int frecuencia;
    char letra;
    char codigo[10];
```

```
diccionario.
int elementos_diccionario[MAX_CARACTERES];      // Almacena la info del diccionario.

// FUNCIONES AUXILIARES PARA DECODIFICAR.
int longitud_codificado(char codificadotxt[20])
{
    int cont=0;
    char caracter;
    FILE * codificado;

    codificado = fopen(codificadotxt, "r");
    if(!codificado)
    {
        printf("Archivo invalido...\n");        //Devuelve la longitud del archivo
codificado.
    return;
    }

    if(codificado!=NULL)
    {
        while(feof(codificado)==0)
        {
            caracter=fgetc(codificado);
            cont++;
        }
    }
    fclose(codificado);
    return cont;
}

int verifica_igualdad(char cadena1[], char cadena2[], int longitud)
{
    int i=0, bandera=0;

    for(i=0; i<longitud; i++)
    {
        if(cadena1[i]==cadena2[i])
        {
            bandera=1;                          // Verifica si el codigo leido es igual
al codigo de la letra evaluada.
        }
        else
        {
            bandera=0;
            return bandera;
        }
    }
    return bandera;
}

void vaciar_arreglo(char *comparar, int tamano)
{
    int i=0;

    // Vacia el arreglo para leer una letra nueva.
    for(i=0;i<=tamano;i++)
        comparar[i]='\0';
}

void guarda_diccionario(char diccionariotxt[20])
{
    int i=0;
    FILE * diccionario;

    diccionario = fopen(diccionariotxt, "r");       //Abrimos el diccionario para asignar
cada campo a la estructura.
    if(!diccionario)
    {
        printf("Archivo invalido...\n");
        return;
    }
    if(diccionario != NULL)
    {
        while(!feof(diccionario)){          //Empieza a guardar cada campo.
            fscanf(diccionario, "%c (%d): %s",
&nodo[nro_elementos].letra,&nodo[nro_elementos].frecuencia,&nodo[nro_elementos].codigo);
            nro elementos++:
```

```
void decodifica(char codificadotxt[20])
{
    int cont1=0, cont2=0, longitud=0;
    char caracter;
    FILE *codificado;

    codificado = fopen(codificadotxt, "r");     //Abrimos el archivo codificado para
empezar a decodificar.
    if(!codificado)
    {
        printf("Archivo invalido...\n");
        return;
    }
    longitud=longitud_codificado(codificadotxt);
    char comparar[10];
    if(codificado != NULL)
    {
        while(feof(codificado) == 0)
        {
            caracter=fgetc(codificado);
            comparar[cont1]=caracter;
            for(cont2=0;cont2<nro_elementos;cont2++)
            {
                if(strncmp(comparar,nodo[cont2].codigo, 10) == 0)
                {
                    printf("%c", nodo[cont2].letra);
                    vaciar_arreglo(comparar, cont1);
                    cont1=-1;
                    goto salir;
                }
            }
            salir:
            cont1++;
        }
        fclose(codificado);
    }
    printf("\n");
}

void imprime_dicc()
{
    int i=0;
    for(i=0;i<nro_elementos;i++)
        printf("Caracter: %c Frecuencia:(%d): Codigo:%s\n",
nodo[i].letra,nodo[i].frecuencia,nodo[i].codigo);
}


void main()
{
    char diccionariotxt[20], codificadotxt[20];
    printf("         |HUFFMAN|      \n");
    printf("Ingrese el nombre del archivo diccionario: ");
    scanf("%s", &diccionariotxt);
    printf("Ingrese el nombre del archivo codificado: ");
    scanf("%s", &codificadotxt);
    guarda_diccionario(diccionariotxt);
    decodifica(codificadotxt);
}
```

c algoritmos

editada el 25 jun. 16 a las 14:31

Shaz
20.1k 8 32 52

formulada el 25 jun. 16 a las 2:55

Marshal
16 1

1 Buenas! ¿Te ha servido la respuesta?

interese ver esto. – El Asiduo el 9 sep. 16 a las 14:12

1 respuesta

4

```
char diccionariotxt[20];
scanf("%s", &diccionariotxt);
```

`diccionariotxt` es un array de caracteres, es decir, un puntero. Por otro lado `scanf` necesita que se le pasen punteros para poder almacenar los valores que recupera del teclado. En tu caso estás pasando una referencia a un puntero o, lo que es lo mismo, un puntero doble. Lo correcto sería lo siguiente:

```
char diccionariotxt[20];
scanf("%s", diccionariotxt);
```

Este error lo tienes replicado en otras tantas lecturas de tu código.

Tambíen deberías vigilar los mensajes del compilador, incluídos los warning ya que algunos de ellos pueden tener efectos indeseados sobre tu aplicación. En este caso es un return vacío cuando la función espera el retorno de un entero:

```
int longitud_codificado(char codificac
{
    if(!codificado)
    {
        printf("Archivo invalido...\n'
codificado.
        return; // <<--- AQUI!!!!
    }
}
```

Un saludo.

respondida el 27 jun. 16 a las 7:09

eferion

29.3k 3 20 58