Escuela de computación

Curso: IC 2001 Estructura de datos

Profesor: Víctor Garro Abarca

Estudiantes:

Bryan Daniel Barquero Barrantes

José Andrés Quesada Artavia

Proyecto 3 – Compresor y descompresor de Huffman

Manual técnico

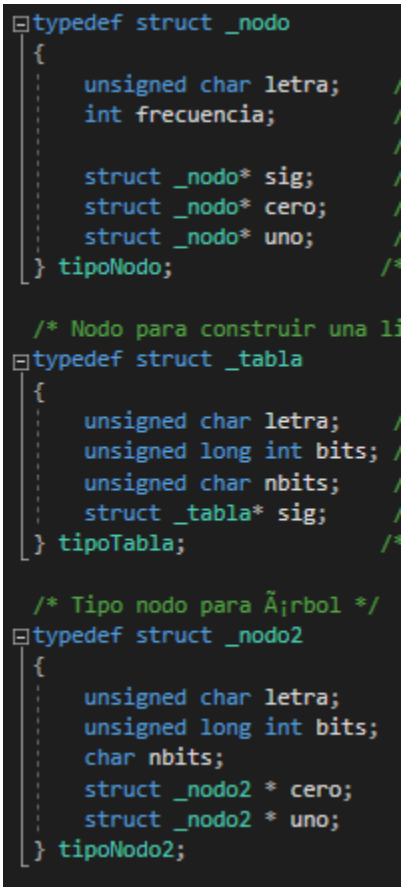
Segundo semestre 2021

9de noviembre 2021

Tabla de contenidos

Estructuras de datos utilizadas

Structs



```
typedef struct _nodo
{
    unsigned char letra;
    int frecuencia;

    struct _nodo* sig;
    struct _nodo* cero;
    struct _nodo* uno;
} tipoNodo;

/* Nodo para construir una li
typedef struct _tabla
{
    unsigned char letra;
    unsigned long int bits;
    unsigned char nbits;
    struct _tabla* sig;
} tipoTabla;

/* Tipo nodo para Ã¡rbol */
typedef struct _nodo2
{
    unsigned char letra;
    unsigned long int bits;
    char nbits;
    struct _nodo2 * cero;
    struct _nodo2 * uno;
} tipoNodo2;
```

Creación de datos de tipo *struct* cuya función es almacenar datos de veces que aparece una letra en el archivo, cantidada de repeticiones que posee ese elemento

_nodo: Dato de tipo struct para que se utiliza como codo para la estructura del árbol binario, el nodo almacena un valor de tiopo *char* el cual representa la referencia de la letra que aparece en el texto o archivo, la cantidad de apariciones que hace al momento de analizar en el texto o archivo el cual es un elemento de tipo entero, además de poseer los punteros de referencia de la raíz de árbol y los nodo siguiente valor de la lista al que está referenciando, cero y uno, estos de tipo *_nodo* que referencian el valor del nodo según el puntero de la rama ya sea por uno o cero.

_tabla: Dato de tipo estruct utilizado para para la construcción de la tabla que almacena los códigos creados por el recorrido de los nodos del árbol binario. Cada dato almacenado representa un valor dentro de la tabla como lo son: un valor de *char* el cual hace referencia al nodo con la letra, un valor de tipo entero que almacena la codificación de la respectiva letra, valor de tipo *char* que representa el numero de bits de la codificación y finalmente la referencia al siguiente elemento a ser agregado a la tabla.

_nodo2: Dato de tipo struct para el uso del árbol de decodificación, este crea un árbol basado en la información de la tabla creada a partir de la información de árbol inicial, con esto mantener la información necesaria para llevar a cabo el proceso de restructurar el orden la información del archivo que este comprimido y regresarlo a su forma original. Cada dato almacenado representa un valor dentro de la tabla como lo son: un valor de *char* el cual hace

referencia al nodo con la letra, un valor de tipo entero que almacena la codificación de la respectiva letra, valor de tipo *char* que representa el número de bits de la codificación.

Árbol binario

Para obtener los códigos de Huffman hay que construir un árbol binario de nodos, a partir de una lista de nodos, cuyo tamaño depende del número de símbolos (el símbolo y el peso o frecuencia de aparición.

Cada nodo del árbol puede ser o bien un nodo hoja o un nodo interno. Inicialmente se considera que todos los nodos de la lista inicial son nodos hoja del árbol. Al ir construyendo el árbol, los nodos internos tendrán un peso y dos nodos hijos. Por convención el bit '0' se asocia a la rama izquierda y el bit '1' a la derecha. Una vez finalizado el árbol contendrá n nodos hijo y n-1 nodos internos.

El proceso de construcción del árbol comienza formando un nodo intermedio que agrupa a los dos nodos hoja que tienen menor peso. El nuevo nodo intermedio tendrá como nodos hijos a estos dos nodos hoja y su campo peso será igual a la suma de los pesos de los nodos hijos. Los dos nodos hijos se eliminan de la lista de nodos, sustituyéndolos por el nuevo nodo intermedio. El proceso se repite hasta que solo quede un nodo en la lista. Este último nodo se convierte en el nodo raíz del árbol.

El algoritmo de construcción del árbol puede resumirse así:

- Crear un nodo hoja para cada símbolo, asociando un peso según su frecuencia de aparición e insertarlo en la lista ordenada ascendentemente.
- Mientras haya más de un nodo en la lista:
    - Eliminar los dos nodos con menor probabilidad de la lista.
    - Crear un nuevo nodo interno que enlace a los nodos anteriores, asignándole como peso la suma de los pesos de los nodos hijos.
    - Insertar el nuevo nodo en la lista, (en el lugar que le corresponda según el peso).
- El nodo que quede es el nodo raíz del árbol.

Una vez que se tiene el árbol de codificación, hay que usarlo para generar un diccionario de codificación. Para esto se tiene que generar códigos binarios para cada uno de los símbolos del archivo de entrada.

Se marca cada rama del árbol con 1 o 0, y entonces el código de cada símbolo será la serie de 1s y 0s que hay que atravesar para alcanzarlo.

Ordenar frecuencia de aparición

```
void Ordenar(tipoNodo** Lista)
{
    tipoNodo* Lista2, * a;

    if (!*Lista) return; /* List
    Lista2 = *Lista;
    *Lista = NULL;
    while (Lista2)
    {
        a = Lista2;
        Lista2 = a->sig;
        InsertarOrden(Lista, a);
    }
}
```

Realiza el ordenamiento de la lista que es pasada por referencia donde se almacenan los caracteres por la cantidad de apariciones que realiza, de tal modo que los valores que tiene menor cantidad de apariciones se colocan al inicio y los de mayor aparición quedan al final de la lista.

Crear árbol binario

```
/* Crear el arbol */
Arbol = Lista;
while (Arbol && Arbol->sig) /* Mientras exist
{
    p = (tipoNodo*)malloc(sizeof(tipoNodo));
    p->letra = 0;
    p->uno = Arbol;
    Arbol = Arbol->sig;
    p->cero = Arbol;
    Arbol = Arbol->sig;
    p->frecuencia = p->uno->frecuencia +
        p->cero->frecuencia;      /* Suma de
    InsertarOrden(&Arbol, p);
}
```

Una vez creada la lista ordenada de los caracteres por frecuencia de aparición se proceda a crear el árbol binario. Para esta estructura se va a ir añadiendo los nodos de la lista anteriormente organizada y asignándole un valor a sus aristas, estas tendrán que la arista que está relacionada al hijo izquierdo tendrá un valor de 0, mientras que la del hijo derecho un valor de 1.

Crear tabla de codificación

Crea la tabla de codificación de manera recursiva recorriendo el árbol binario, ya sea tipoNodo de valor uno o cero le asigna los bits y su cantidad para ser insertado sen la tabla.

```
void CrearTabla(tipoNodo* n, int l, int v)
{
    if (n->uno)  CrearTabla(n->uno, l + 1, (v << 1) | 1);
    if (n->cero) CrearTabla(n->cero, l + 1, v << 1);
    if (!n->uno && !n->cero) InsertarTabla(n->letra, l, v);
}
```

Buscar carácter en la tabla

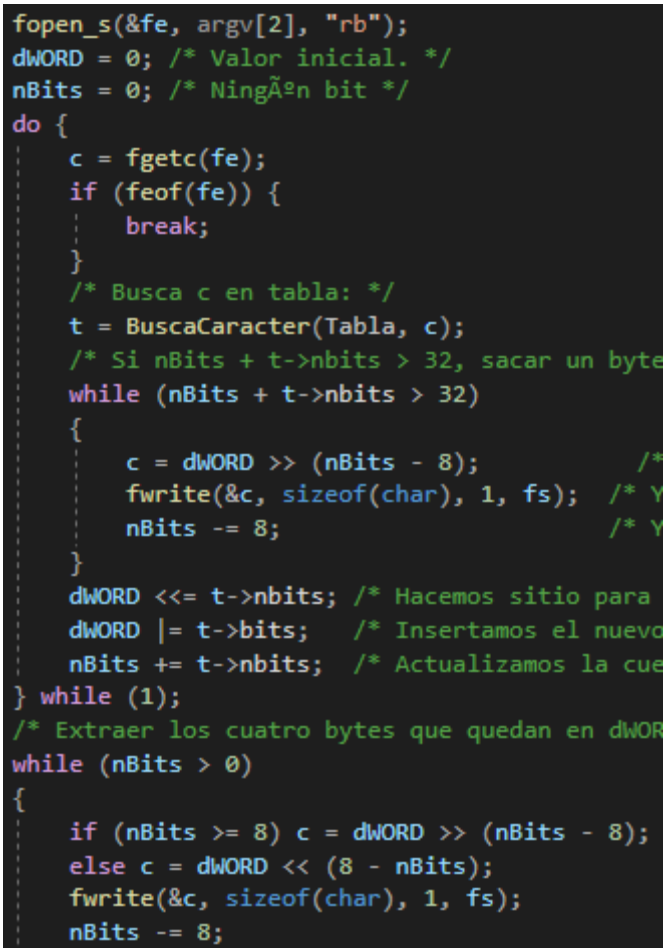
Recibe la tabla de codificación y busca el carácter que ingresado junto a la tabla para obtener así la posición en memoria dentro de la tabla de dicho carácter para obtener los datos relacionados y de esta forma poder almacenarlos en el fichero de salida de contenderá el archivo codificado.

```
tipoTabla* BuscaCaracter(tipoTabla* Tabla, unsigned char c)
{
    tipoTabla* t;

    t = Tabla;
    while (t && t->letra != c) t = t->sig;
    return t;
}
```

Codificación del archivo

Cuando la tabla de codificación este terminada se puede gravar el nuevo archivo codificado en el cual se van a almacenar cada uno de los caracteres con sus valores de frecuencia y codificación.



```
fopen_s(&fe, argv[2], "rb");
dWORD = 0; /* Valor inicial. */
nBits = 0; /* NingÃºn bit */
do {
    c = fgetc(fe);
    if (feof(fe)) {
        break;
    }
    /* Busca c en tabla: */
    t = BuscaCaracter(Tabla, c);
    /* Si nBits + t->nbits > 32, sacar un byte
    while (nBits + t->nbits > 32)
    {
        c = dWORD >> (nBits - 8);           /*
        fwrite(&c, sizeof(char), 1, fs);  /* Y
        nBits -= 8;                         /* Y
    }
    dWORD <<= t->nbits; /* Hacemos sitio para
    dWORD |= t->bits;   /* Insertamos el nuevo
    nBits += t->nbits;  /* Actualizamos la cue
} while (1);
/* Extraer los cuatro bytes que quedan en dWOR
while (nBits > 0)
{
    if (nBits >= 8) c = dWORD >> (nBits - 8);
    else c = dWORD << (8 - nBits);
    fwrite(&c, sizeof(char), 1, fs);
    nBits -= 8;
```

En el proceso de gravado se tomará cada valor almacenado en la tabla buscando cada una de las letras y obteniendo la dirección de memoria donde fue almacenada, con ello se escribirá en el archivo cada uno de los caracteres con su respectiva información como fue mencionada anteriormente mientras que cada bloque de información tenga un peso de 8 bits.

Al momento que este por finalizar el proceso restarán solamente 4 bits los cuales serán extraídos y almacenados bajo un proceso similar, pero con diferente cantidad del valor inicial. Al finalizar el árbol binario y la tabla de codificación serán borradas de la memoria al no ser necesario tenerlas almacenadas.

Decodificación

Para el proceso de decodificación se seguirá un proceso similar a la codificación, únicamente que este proceso será de forma invertida.

Una vez se abre el archivo a decodificar mediante la información almacenada se restaurará el árbol binario con la tabla de codificación. Se emplear la información de para conformar cada uno de los nodos en su posición original utilizando el valor 1 ó 0 de las aristas que une los nodos. Se restaurarán los nodos con los valores de carácter, frecuencia y código de codificación.

```
Arbol2 = (tipoNodo2*)malloc(sizeof(tipoNodo2));
Arbol2->letra = 0;
Arbol2->uno = Arbol2->cero = NULL;
fopen_s(&fe, argv[2], "rb");
fread(&Longitud2, sizeof(long int), 1, fe); /*
fread(&nElementos, sizeof(int), 1, fe); /* Lee
for (i = 0; i < nElementos; i++) /* Leer todos
{
    p2 = (tipoNodo2*)malloc(sizeof(tipoNodo2));
    fread(&p2->letra, sizeof(char), 1, fe); /*
    fread(&p2->bits, sizeof(unsigned long int),
    fread(&p2->nbits, sizeof(char), 1, fe); /*
    p2->cero = p2->uno = NULL;
    /* Insertar el nodo en su lugar */
    j = 1 << (p2->nbits - 1);
    q = Arbol2;
```

```
bits = 0;
fopen_s(&fs, argv[3], "wb");
/* Lee los primeros cuatro bytes en la dobel palabra bit
fread(&a, sizeof(char), 1, fe);
bits |= a;
bits <<= 8;
fread(&a, sizeof(char), 1, fe);
bits |= a;
bits <<= 8;
fread(&a, sizeof(char), 1, fe);
bits |= a;
bits <<= 8;
fread(&a, sizeof(char), 1, fe);
bits |= a;
j = 0; /* Cada 8 bits leemos otro byte */
q = Arbol2;
/* Bucle */
do {
    if (bits & 0x80000000) q = q->uno; else q = q->cero;
    bits <<= 1;           /* Siguiente bit */
    j++;
    if (8 == j)           /* Cada 8 bits */
    {
        i = fread(&a, sizeof(char), 1, fe); /* Leemos un
        bits |= a;                  /* Y lo insertamos
        j = 0;                      /* No quedan hueco
    }
    if (!q->uno && !q->cero)        /* Si el nodo es u
    {
        putc(q->letra, fs);         /* La escribimos e
        Longitud2--;                 /* Actualizamos l
        q = Arbol2;                    /* Volvemos a l
```

Al igual que la compresión se estructura en leer los datos en secciones de 4 bytes para ser almacenado en los espacios de memoria hasta recorrer la longitud del archivo.