PRÁCTICA Nº 6: 2 sesiones

(del 22 al 31 de Mayo de 2002)

APLICACIÓN DE ÁRBOLES BINARIOS A LA COMPRESIÓN DE FICHEROS DE TEXTO MEDIANTE LA UTILIZACIÓN DE LOS CÓDIGOS DE HUFFMAN

0.- OBJETIVOS

El objetivo de esta práctica es la utilización de los códigos de Huffman para la compresión de ficheros de texto.

1.- LOS CÓDIGOS DE HUFFMAN

Para mostrar la forma en que pueden usarse los árboles binarios como estructuras de datos, se va a ver en esta práctica el problema de la construcción de los llamados "códigos de Huffman":

Supóngase que se tienen mensajes formados por secuencias de caracteres (un texto cualquiera tendría esta forma). En cada mensaje, los caracteres son independientes entre sí y aparecen con una determinada probabilidad conocida a priori. Se desea codificar cada carácter mediante una secuencia de ceros y unos, de manera que ningún código de un carácter sea prefijo del código de otro carácter, esta propiedad de prefijos evitará cualquier tipo de ambigüedad en el proceso de decodificación del mensaje.

El problema de codificación que se plantea sería: dado un conjunto de caracteres, en nuestro caso supongamos que todo el código ASCII (256 valores), y sus probabilidades de aparición en un texto (este dato no es conocido a priori y tendremos que hacer un estudio previo), encontrar un código para esos caracteres que cumpla la propiedad de prefijos, tal que la longitud media del código de un carácter sea mínima. La razón por la que se desea minimizar la longitud media de un código es que ello permitirá comprimir la longitud del mensaje codificado.

El algoritmo de Huffman permite encontrar un código con las propiedades especificadas. La idea del algoritmo consiste en llegar a formar un árbol binario que represente el código. En los nodos terminales (hojas) de este árbol se almacenarán los caracteres que se desea codificar junto con la probabilidad de aparición de cada uno de ellos, cumpliéndose que los caracteres más probables están más próximos a la raíz que los menos probables. De manera que el camino desde la raíz del árbol a cualquier hoja del mismo representará el código para el carácter de esa hoja, de acuerdo con el criterio de que los subárboles izquierdos se etiquetan con un 0 y los subárboles derechos lo hacen con un 1. Según este criterio, los caracteres con mayor probabilidad de aparición tendrán los códigos de menor longitud, mientras que los que tengan una probabilidad de aparición menor tendrán asociado un código de mayor longitud.

El algoritmo de Huffman consiste en lo siguiente:

Suponiendo conocida la probabilidad de aparición de un carácter en un texto (sea p_i) hacer:

1) Crear un *bosque* formado por tantos árboles independientes como caracteres se vayan a codificar. Cada carácter será la raíz de un árbol independiente, almacenándose junto a él su probabilidad de aparición.
2) Buscar en el *bosque* los dos árboles que presenten las menores probabilidades de aparición.
3) Agrupar los dos árboles encontrados en el paso 2) en un solo árbol que tenga como subárbol izquierdo el árbol que tenía menor valor p_i asociado y como subárbol derecho el otro árbol. En la raíz de este nuevo árbol no se almacena ningún carácter (o se almacena un carácter no imprimible, como por ejemplo **`char (7)`**) pero sí la probabilidad de aparición de los caracteres agrupados, que será la suma de sus dos probabilidades de aparición ($p_i + p_j$).
4) Repetir desde el paso 2) hasta que en el *bosque* sólo quede un árbol que agrupe a todos los

caracteres.

2.- REALIZACIÓN DE LA PRÁCTICA

Codificar/Decodificar archivos de texto mediante códigos de Huffman.

Codificación

Para codificar archivos, habrá que realizar las siguientes tareas:

Tarea 1: Pedir al usuario el nombre del fichero a codificar, y el nombre del fichero donde queremos poner la información codificada.

Encontrar las probabilidades de aparición de los caracteres del código ASCII (ordinales desde 32 a 255) en el fichero de texto que se desea codificar.

Para este proceso se incluirá en el programa una rutina que calcule el número de veces que aparece cada carácter en el fichero texto. Será necesario definir una tabla que almacene la frecuencia de aparición de cada carácter.

Tarea 2: Generar el código de Huffman a partir de la tabla de frecuencias.

Esta tarea se realizará después de la tarea 1 y su esquema será:

1) Generar el árbol de Huffman según el algoritmo dado anteriormente, utilizando la tabla de la tarea 1 y sin considerar aquellos caracteres que no aparecen ninguna vez.
2) Recorrer el árbol para guardar, en forma de vector, el código que representa cada carácter, de manera que sea fácil consultarlo usando los caracteres como índices.

   Esta parte se realizará mediante el recorrido prefijo del árbol, pasando a la función de recorrido tres parámetros: El subárbol que estamos recorriendo, la subcadena generada hasta ese momento por el recorrido y por referencia la tabla de códigos.

   El algoritmo sería básicamente el siguiente:

   a) Si el nodo contiene un carácter imprimible (código ASCII entre 32 y 255) se guarda en la posición adecuada de la tabla de códigos el valor de la cadena.
   b) Si no es un carácter imprimible (es un nodo interior) se sigue recorriendo el árbol, primero el subárbol izquierdo añadiendo a la subcadena generada un '0', y luego el subárbol derecho añadiendo a la subcadena generada un '1'.

Tarea 3: Codificar un fichero texto según el código generado en la tarea 2.

En esta tarea, a partir del fichero texto original, se obtendrá un fichero "binario" (*sólo ceros y unos, como caracteres (1 byte), no como bits*) que haga corresponder a cada carácter del fichero original el código de Huffman correspondiente.

La rutina correspondiente a esta tarea responderá al siguiente esquema:

1) Almacenar el árbol de códigos como cabecera del fichero binario. Se utilizará para ello la notación empleada en otros casos ligeramente modificada: Se recorre en orden prefijo el árbol, escribiendo en el archivo la siguiente información:
   - Para cualquier nodo no vacio se guarda el carácter contenido en él.
   - Si el nodo está vacío no se guarda nada ya que todos los nodos terminales (hojas) contienen caracteres imprimibles (entre 32 y 255), mientras que los nodos interiores contienen un carácter no imprimible (en nuestro caso `char (7)`.)
2) Para cada carácter del fichero original hacer:
   2.1) Buscar en la matriz la cadena de 0 y 1 que tiene asociada.

2.2) Almacenar en el fichero "binario" la secuencia de ceros y unos determina en el paso anterior.

En ningún caso se realizarán saltos de línea, ya que no tienen sentido en este problema.

En esta fase habrá que estimar cuanto ocuparía el fichero "binario" si cada '0' y '1' escrito en él fuese un bit, en vez de un carácter. ¿Se conseguiría comprimir el fichero original? ¿Y si consideramos lo que ocupa la cabecera con el árbol de códigos?

Decodificación

Tarea 4: Decodificar un fichero numérico de ceros y unos para obtener el fichero texto correspondiente, a partir del código de Huffman que tiene almacenado.

El esquema de la rutina será el siguiente:

1) Pedir el nombre del fichero a decodificar y el nombre del fichero donde deseamos poner la información decodificada.
2) Leer la cabecera del fichero "binario" y reconstruir en memoria el árbol de códigos.
2) Leer carácter a carácter el fichero binario y recorrer el árbol simultáneamente.
3) Si se encuentra en el árbol el código de un carácter imprimible, escribir ese carácter en el fichero de salida.

Los procesos de Codificación y Decodificación se englobarán en un mismo programa que permita ejecutar la tarea deseada.

La clase `Arbol`

En esta práctica utilizaremos árboles que van a contener en los nodos tanto la frecuencia de aparición de cada carácter, como caracteres. El tipo **`Valor`** quedará por tanto como sigue

```
struct Valor
{
   int frec;
   char car;
};
```

Por otro lado, la clase árbol, tendrá la siguiente interfaz, tal como se vió en clase:

```
class Arbol
{
   public:
      Arbol (void);
      Arbol (const Arbol &);
      ~Arbol (void);
      bool ArbolVacio (void);
      bool Informacion (Valor &);
      Arbol &HijoIzdo (void);
      Arbol &HijoDcho (void);
      void HacerArbol (Arbol &, Valor, Arbol &);

   private:
      typedef Arbol * PunteroArbol;

      bool esvacio;

      Valor info;
      PunteroArbol izdo, dcho;
```

```
};
```

Se deja a criterio del alumno, la posibilidad de utilizar funciones adicionales auxiliares contenidas en la parte privada de la clase.

4.- ENTREGA DE PROGRAMAS

Al finalizar la práctica correspondiente se entregarán al profesor tres ficheros:

1) Programa principal de codificación y decodificación (**`Huffma##.cpp`**).
2) Ficheros de la clase **`Arbol`** (**`Arbol##.cpp`** y **`Arbol##.h`**)

Nota Muy Importante

Antes de poder empezar a realizar cualquiera de las prácticas es necesario presentar al las hojas de especificación de programas con las tareas que se van a realizar en la práctica, explicando brevemente como se van a solucionarse los problemas que se plantean.

En este caso se entregarán al profesor, los organigramas correspondientes a las tareas

ENTREGA DE PROGRAMAS: Al comenzar la sesión de prácticas del 5 al 7 de Junio.

AYUDA:

La tabla de frecuencias será una tabla que guardará enteros. La tabla vendrá indexada por el valor del carácter del que guardamos la información de la frecuencia.

```
const int NUM_CAR = 256;

typedef int VectorEnteros[NUM_CAR];
```

Para el bosque podemos crear una estructura que guarde un vector de árboles y el número de árboles válidos dentro del vector.

```
typedef Arbol VectorArboles[NUM_CAR];

struct Bosque
{
   VectorArboles info;
   int n;
};
```

También podríamos utilizar simplemente una lista de árboles, añadiendo, para simplificar el problema un método para encontrar el mínimo (que colocase el punto de interés sobre el elemento mínimo de la lista.)

Finalmente para guardar los códigos de Huffman nos basaremos en la misma idea de la tabla de frecuencias, pero en este caso lo que se guardará en la tabla serán los códigos generados en el recorrido del árbol.

```
typedef string VectorEnteros[NUM_CAR];
```

Nota:

Mientras que tanto la tabla de frecuencias, como la tabla de los códigos de Huffman, son vectores y no necesitan ser pasados por referencia, el Bosque es una estructura, que si que necesita ser pasada por referencia en los casos que se crea conveniente.