Universidad Autónoma de Madrid

Departamento de Informática

Estructura de Datos
Memorias Práctica - 3

Hecho por: Diego Rodríguez

y Alejandro García

Introducción:

En esta práctica hemos aprendido cómo se manejan las tablas y la estructura de índices de una base de datos centrada en libros. A pesar de que esta base de datos estaba muy simplificada para facilitarnos el trabajo, nos ha permitido entender cómo se almacenan tablas en ficheros en binario para evitar usar la memoria RAM y como se ha desarrollado la idea de los índices para minimizar el número de accesos a disco. En la práctica se nos ha pedido que desarrollemos todas las funciones necesarias para el funcionamiento de una base de datos con una única tabla de dos campos (book_id y título) , salvo la destinada a eliminar una entrada de la tabla.

Funciones relativas a ficheros utilizadas:

`fopen() y fclose()`

Estas funciones han sido utilizadas para abrir los archivos, leer , escribir y para cerrarlos al acabar. Al trabajar todo el rato con ficheros en binario necesitábamos abrirlos de la siguiente forma: fopen(filename , “wb+”o “rb+”) si se iba a escribir o a leer, respectivamente. Este parámetro no es del todo relevante ya que ambos permiten la lectura y la escritura que son las acciones que vamos a ejecutar en el fichero.

`fwrite()y fread()`

A la hora de escribir y leer en un fichero utilizamos fwrite y fread que nos permiten hacerlo en binario. Con estas funciones son con las que se almacenan todos los datos.

`fseek() y ftell()`

A estas funciones hemos recurrido para movernos en el fichero. Por un lado, fseek tiene la ventaja de permitirnos saltar a cualquier byte del fichero conociendo su posición desde el inicio del fichero, su posición relativa a la posición actual, o su posición respecto al final del fichero. Por otro lado, ftell() devuelve la offset del cursor en el fichero. Esto nos permitirá recorrer el fichero de índices de forma binaria, lo cuál reduce considerablemente el coste.

Implementación de funciones:

Funciones `createTable` y `createIndex`

En estas funciones se nos pedía inicializar un fichero de datos con la cabecera que incluye: el primer registro borrado en este caso un -1. Antes de nada comprobamos con una función auxiliar `file_exist()` si el fichero ya existe, en cuyo caso retornamos. En caso contrario creamos el fichero, escribimos la cabecera y tras haber cambiado la extensión del nombre del fichero de datos llamamos a `createIndex`. En esta inicializamos el fichero de índices con su cabecera: nodo raíz y primer registro borrado a -1.

Función `printTree`

El objetivo de desarrollar esta función es facilitar al usuario la visualización del fichero de índices como árbol binario de búsqueda. Debido a la naturaleza recursiva de los árboles binarios decidimos servirnos de una función auxiliar `printnode()` que realiza un dfs en preorder hasta la profundidad especificada donde para cada nodo se extraen los campos guardados en el fichero, algunos se imprimen y otros se utilizan para continuar la búsqueda en profundidad.

```
MAR2 (0): 4
 l MAR0 (2): 37
     l BAR0 (5): 88
         r CAR5 (9): 157
     r MAR1 (6): 105
 r VAR1 (3): 53
     l PAR2 (7): 121
         r RAR2 (10): 174
      r WAR3 (1): 21
```

Función `findKey`

La funcionalidad de `findKey` es: dado un book_id, encontrar esa clave en el árbol de índices. En primer lugar, devolverá true si se encuentra ese ID en la tabla y false en otro caso. En segundo lugar, almacenará en uno de sus parámetros (`NodeIDorDataOffset`) el offset de los datos si el libro se encuentra en la base de datos o el ID del nodo visitado en último lugar, es decir, su posición a insertar. De la misma forma que la función anterior (`printTree`) `findKey` tiene una naturaleza recursiva al tratarse de una búsqueda binaria sobre el árbol de índices, por tanto utilizamos la función recursiva `findKeyRec` para resolver este problema. Esta función al ser muy polivalente será utilizada en las próximas, luego hay que tener especial cuidado con los casos bases y los casos límites como el árbol vacío o unielemental.

Función `addIndexEntry y addTableEntry`

Ambas funciones quizá sean las más complejas, ya que no solo hay que entender la estructura de ambos ficheros (datos e índices) también tener el conocimiento técnico necesario para utilizar fseek y ftell con soltura. Estas funciones añaden una entrada en el fichero de índices y de datos respectivamente. Para ello, recurren a `findKey` para conocer la posición a insertar en caso de que el libro no esté en la tabla. La mayor dificultad que nos encontramos a la hora de desarrollar estas funciones es gestionar los registros borrados y actualizar la lista de registros borrados en función de en qué “hueco” del fichero se inserta. Además para facilitar la lectura y encapsular el código recurrimos a la función auxiliar `findReg` que es de nuevo recursiva y dado un tamaño y un offset encuentra la posición del registro borrado a insertar y el registro borrado anterior, si lo hay.

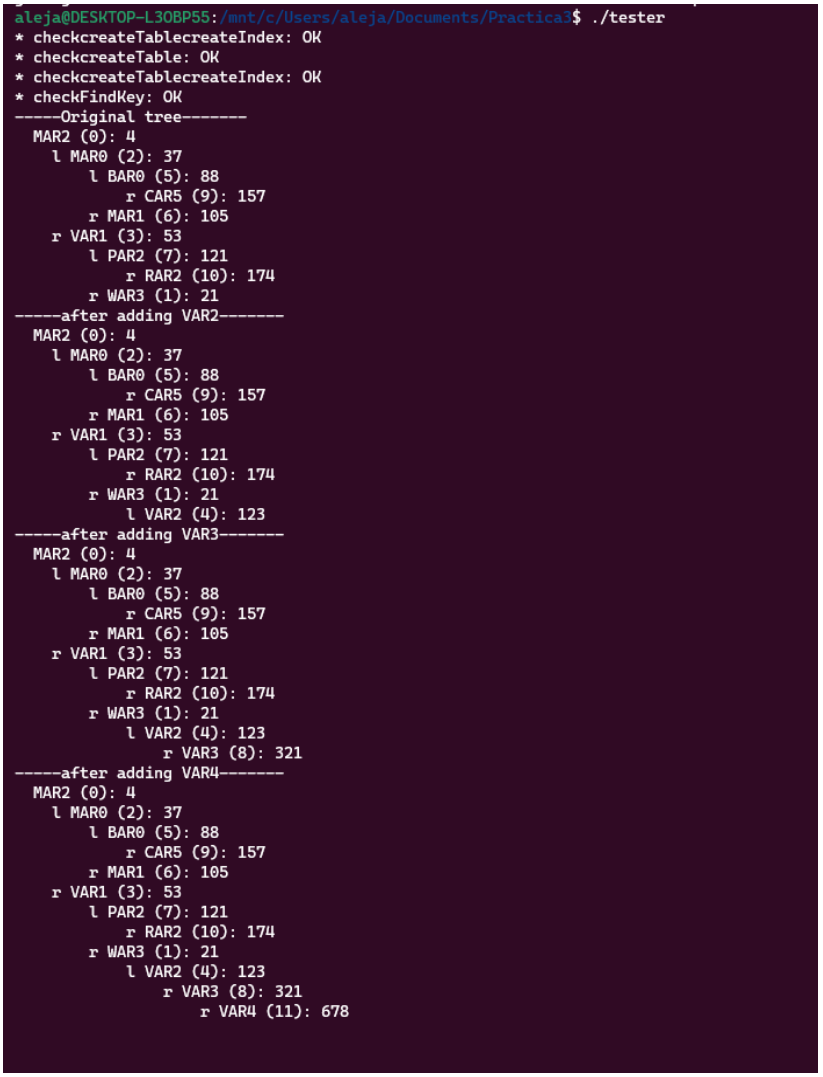
Menú:

Para que la interacción con el usuario sea más cómoda se ha desarrollado una simple menú que tiene cuatro opciones: USE, que inicializa una tabla y sus índices dada el nombre del fichero , INSERT, que añade un libro a la base de datos, y PRINT, que muestra por pantallas el árbol de índices. Para esto hemos realizado una función para cada una de las opciones y una función `Menu` y `ShowMenu` que muestra las distintas opciones.

Test:

Con el fin de intentar asegurar el correcto funcionamiento de las funciones mencionadas hemos intentado realizar una breve batería de pruebas para comprobar diversos casos. En especial nos hemos centrado en las últimas dos funciones, ya que no sólo son las más complicadas, sino que además requieren del correcto funcionamiento de todas las anteriores (a excepción de `printTree`).

En primer lugar hemos comprobado que nuestro código supera las pruebas facilitadas. Para ello, hemos ejecutado el fichero tester y nos hemos asegurado de que los resultados fueran los adecuados.



Tras ello, hemos creado nuestra propia batería de pruebas y la hemos añadido al fichero tester.c para cerciorarnos del correcto funcionamiento de las funciones addIndexEntry y addTableEntry. Dado el estado de la tabla tras los test proporcionados imprimimos por pantalla el estado de la tabla:

```
------- COMPROBAMOS LA INSERCION EN TABLA   -------

  MAR2 (0): 4
    l MAR0 (2): 37
        l BAR0 (5): 88
            r CAR5 (9): 157
        r MAR1 (6): 105
    r VAR1 (3): 53
        l PAR2 (7): 121
            r RAR2 (10): 174
        r WAR3 (1): 21
-------Información de los registros de indice libres-------
ID(4): 71 9
ID(8): 139 10


-------Tabla de Datos-------
Libro 1 (4): MAR2 9 MAR2_zero
Libro 2 (21): WAR3 8 WAR3_one
Libro 3 (37): MAR0 8 MAR0_two
Libro 4 (53): VAR1 10 VAR1_three
Libro 5 (88): BAR0 9 BAR0_five
Libro 6 (105): MAR1 8 MAR1_six
Libro 7 (121): PAR2 10 PAR2_seven
Libro 8 (157): CAR5 9 CAR5_nine
Libro 9 (174): RAR2 8 RAR2_ten
```

Vemos que hay dos registros borrados disponibles para ser escritos sobre ellos; en el fichero de datos hay 2 registros borrados, uno con offset 71 de tamaño 9 y otro con offset 139 y tamaño 10.

Con esta información insertamos cuatro libros con las siguientes características y comprobamos el estado de la tabla de datos y de índices:

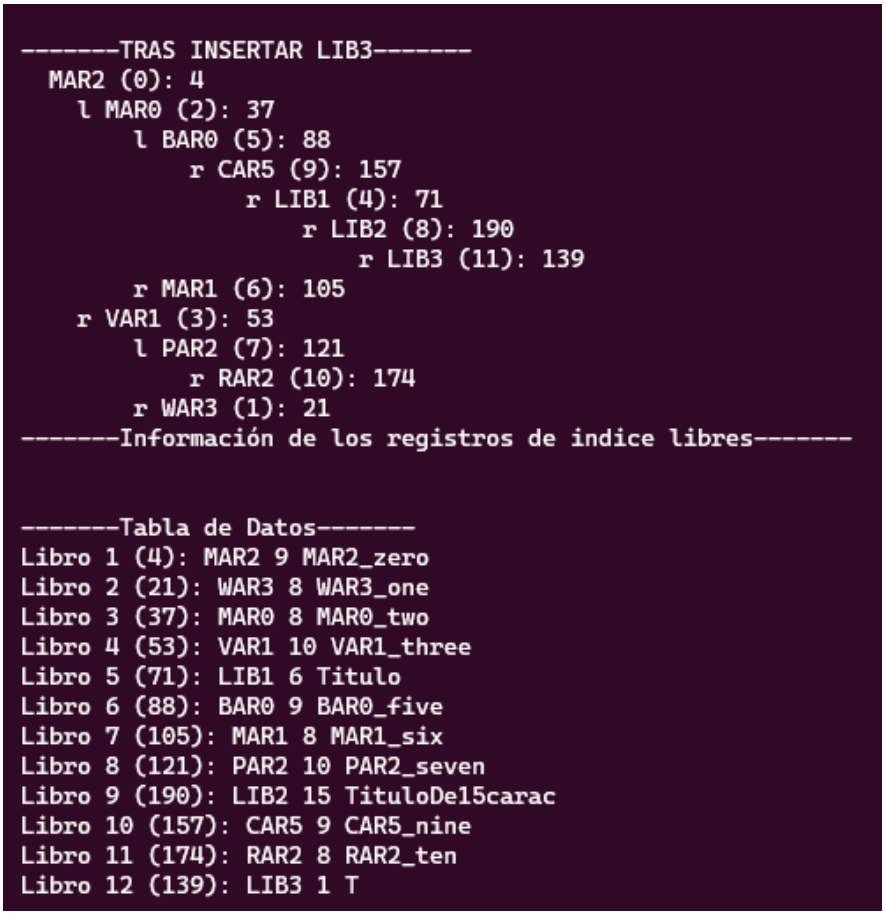
-Libro 1: ID=*LIB1* y Título=*Titulo*. Se inserta su índice en el primer hueco de la tabla de índices y dado que cabe en el primer espacio vacío de la tabla de datos (pues hay disponibles 9 bytes y el libro tan solo ocupa 6) es allí donde se almacena su información. Además, tras la inserción solamente quedarían disponibles 3 bytes, lo que no es suficiente para que quepa ningún libro, por lo tanto se descartan. Quedando en la tabla de datos únicamente un registro borrado con offset 139 y tamaño 10.

```
-------TRAS INSERTAR LIB1-------
  MAR2 (0): 4
    l MAR0 (2): 37
        l BAR0 (5): 88
            r CAR5 (9): 157
                r LIB1 (4): 71
        r MAR1 (6): 105
    r VAR1 (3): 53
        l PAR2 (7): 121
            r RAR2 (10): 174
        r WAR3 (1): 21
-------Información de los registros de indice libres-------
ID(8): 139 10


--------Tabla de Datos-------
Libro 1 (4): MAR2 9 MAR2_zero
Libro 2 (21): WAR3 8 WAR3_one
Libro 3 (37): MAR0 8 MAR0_two
Libro 4 (53): VAR1 10 VAR1_three
Libro 5 (71): LIB1 6 Titulo
Libro 6 (88): BAR0 9 BAR0_five
Libro 7 (105): MAR1 8 MAR1_six
Libro 8 (121): PAR2 10 PAR2_seven
Libro 9 (157): CAR5 9 CAR5_nine
Libro 10 (174): RAR2 8 RAR2_ten
```

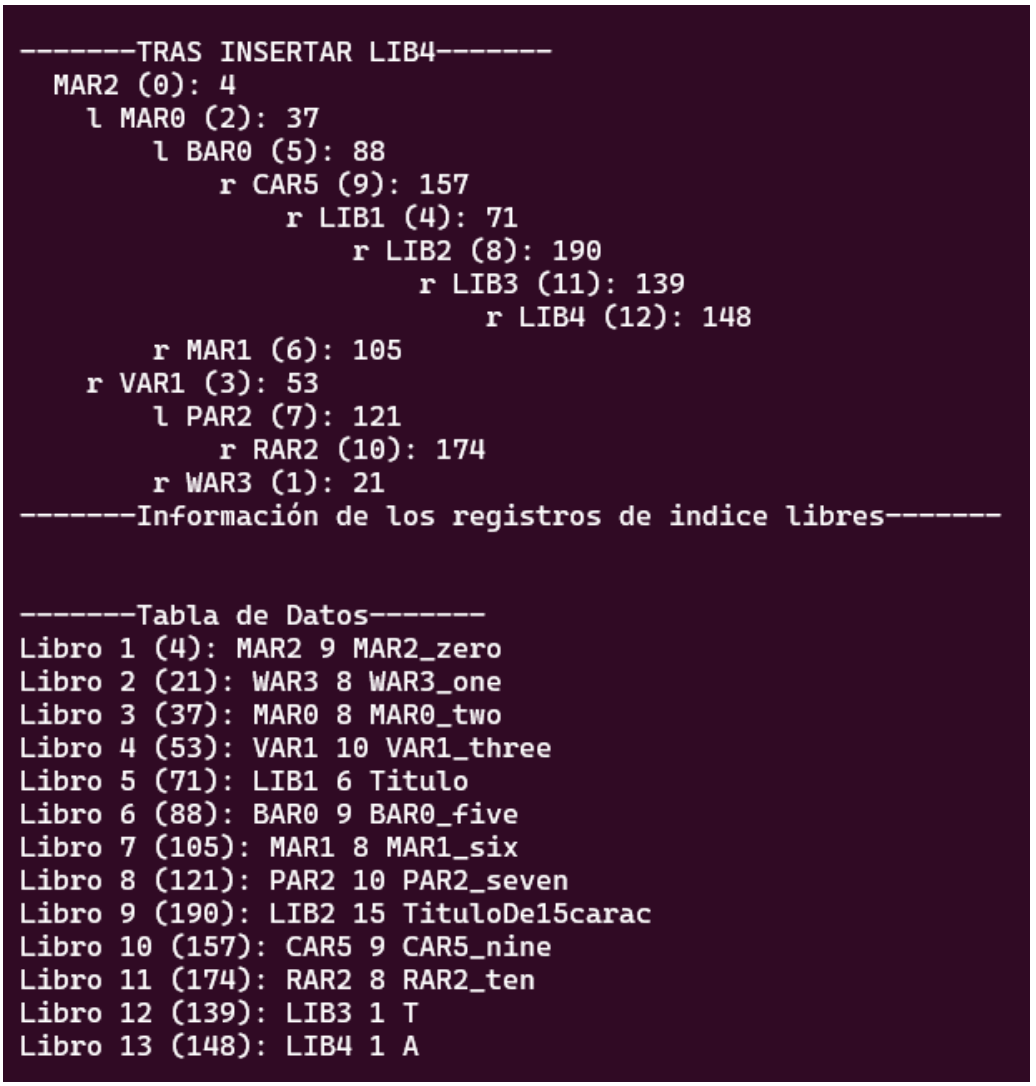
-Libro 2: ID=*LIB2* y Título=*TituloDe15carac*. Se inserta su índice en el primer hueco de la tabla de índices, quedando el fichero sin registros borrados y dado que su título es demasiado largo y no cabe en el espacio vacío del registro borrado de la tabla de datos se escribe al final del archivo.

```
-------TRAS INSERTAR LIB2-------
  MAR2 (0): 4
    l MAR0 (2): 37
        l BAR0 (5): 88
            r CAR5 (9): 157
                r LIB1 (4): 71
                    r LIB2 (8): 190
        r MAR1 (6): 105
    r VAR1 (3): 53
        l PAR2 (7): 121
            r RAR2 (10): 174
        r WAR3 (1): 21
-------Información de los registros de indice libres-------


--------Tabla de Datos-------
Libro 1 (4): MAR2 9 MAR2_zero
Libro 2 (21): WAR3 8 WAR3_one
Libro 3 (37): MAR0 8 MAR0_two
Libro 4 (53): VAR1 10 VAR1_three
Libro 5 (71): LIB1 6 Titulo
Libro 6 (88): BAR0 9 BAR0_five
Libro 7 (105): MAR1 8 MAR1_six
Libro 8 (121): PAR2 10 PAR2_seven
Libro 9 (190): LIB2 15 TituloDe15carac
Libro 10 (157): CAR5 9 CAR5_nine
Libro 11 (174): RAR2 8 RAR2_ten
```

-Libro 3: ID=*LIB3* y Título=*T*. Como el fichero de índices ya no tiene ningún registro borrado, los índices de cada nuevo libro se añadirán al final del archivo. Por ello se inserta su índice al final del archivo y dado que cabe en el único espacio vacío de la tabla de datos es allí donde se almacena su información. Cabe resaltar que dado que el título del tercer libro ocupa únicamente un byte, quedan 9 disponibles con offset 139+9 y se almacena como un espacio disponible.



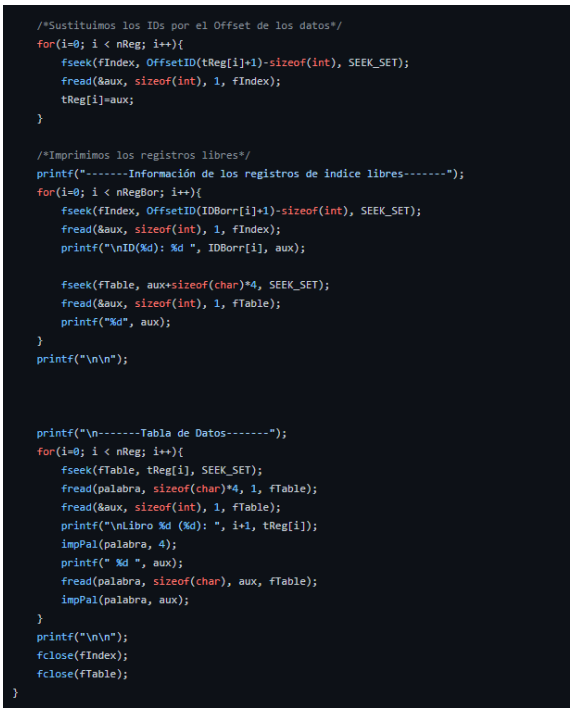
-Libro 4: ID=*LIB4* y Título=*A*. Se inserta su índice al final del archivo y dado que cabe en el único espacio vacío de la tabla de datos (el generado después de insertar el tercer libro) es allí donde se almacena su información. De este modo, tras la inserción de los 4 libros ninguno de los dos archivos tendrá registros borrados, y toda la información nueva se escribirá al final de los mismos.



Para imprimir el estado de la tabla de datos hemos desarrollado la rutina printData que analiza los ficheros con nombres almacenados en indexName y tableName. Para desarrollar la función ha sido necesario crear offsetID(dado el ID de un registro nos devuelve el offset en la tabla de índices), inTable(nos dice si una clave está en una tabla) e impPal(dado el tamaño de un array de caracteres imprime los N primeros):

printData:

```
void printData(const char * indexName, const char * tableName){
    FILE *fIndex=NULL, *fTable=NULL;
    int IDBorr[TAMMAX], nRegBor=0, tReg[TAMMAX], nReg, aux, i, j;
    char palabra[TAMMAX];

    if(!indexName||!tableName){
        printf("\nERROR AL IMPRIMIR LOS DATOS\n");
        return;
    }

    fIndex=fopen(indexName, "rb+");
    fTable=fopen(tableName, "rb+");
    if(!fIndex||!fTable){/*Control de errores*/
        printf("\nERROR EN LA APERTURA AL IMPRIMIR LOS DATOS\n");
        return;
    }

    /*Almacenamos los IDs de los registros borrados y el numero de ellos*/
    fseek(fIndex, sizeof(int), SEEK_SET);
    fread(&aux, sizeof(int), 1, fIndex);
    while(aux!=-1){
        IDBorr[nRegBor]=aux;
        fseek(fIndex, OffsetID(aux)+sizeof(char)*4, SEEK_SET);
        nRegBor++;
        fread(&aux, sizeof(int), 1, fIndex);
    }

    /*Almacenamos los IDs de los registros NO borrados*/
    fseek(fIndex, 0, SEEK_END);
    aux=ftell(fIndex);
    nReg=(aux-sizeof(int)*2)/TAMRegInd;/*Almacenamiento de número de IDs de regs, borrados y sin borrar*/

    for(i=0, j=0; i < nReg; i++){
        if(inTable(IDBorr, i, nRegBor)==0){
            tReg[j]=i;
            j++;
        }
    }
    nReg=j;
```



```
    /*Sustituimos los IDs por el Offset de los datos*/
    for(i=0; i < nReg; i++){
        fseek(fIndex, OffsetID(tReg[i]+1)-sizeof(int), SEEK_SET);
        fread(&aux, sizeof(int), 1, fIndex);
        tReg[i]=aux;
    }

    /*Imprimimos los registros libres*/
    printf("-------Información de los registros de indice libres-------");
    for(i=0; i < nRegBor; i++){
        fseek(fIndex, OffsetID(IDBorr[i]+1)-sizeof(int), SEEK_SET);
        fread(&aux, sizeof(int), 1, fIndex);
        printf("\nID(%d): %d ", IDBorr[i], aux);

        fseek(fTable, aux+sizeof(char)*4, SEEK_SET);
        fread(&aux, sizeof(int), 1, fTable);
        printf("%d", aux);
    }
    printf("\n\n");



    printf("\n-------Tabla de Datos-------");
    for(i=0; i < nReg; i++){
        fseek(fTable, tReg[i], SEEK_SET);
        fread(palabra, sizeof(char)*4, 1, fTable);
        fread(&aux, sizeof(int), 1, fTable);
        printf("\nLibro %d (%d): ", i+1, tReg[i]);
        impPal(palabra, 4);
        printf(" %d ", aux);
        fread(palabra, sizeof(char), aux, fTable);
        impPal(palabra, aux);
    }
    printf("\n\n");
    fclose(fIndex);
    fclose(fTable);
}
```

offsetID, inTable e impPal:

```
int OffsetID(int ID){
    return (int)sizeof(int)*2+ID*TAMRegInd;
}
int inTable(int *array, int key, int tam){
    int i;
    if(!array||tam<0) return -1;
    for(i=0; i<tam; i++){
        if(array[i]==key) return 1;
    }
    return 0;
}
void impPal(char *palabra, int tam){
    int i;
    if(!palabra) return;

    for(i=0; i<tam; i++) printf("%c", palabra[i]);
}
```

Además para hacer alguna prueba más, hemos usado el menú desarrollado. Estos test se centran en pruebas sobre tablas vacías o con un solo elemento. En el siguiente ejemplo probamos a insertar un libro a una base de datos vacía:

```
diego@DESKTOP-29VLGGP:/mnt/c/Universidad/Segundo/FFBB/FFBB-Practicas/Practica3$ make -f makefile.cod run
Compiling menu.c...
gcc -g -Wall -Wextra -pedantic -ansi -c -o menu.o menu.c
gcc -g -o menu menu.o utils.o
./menu
 (1) Use
 (2) Insert
 (3) Print
 (4) Exit

Enter a number that corresponds to your choice > 1

Introduce el nombre del fichero de datos con la extensión incluida:data.dat
 (1) Use
 (2) Insert
 (3) Print
 (4) Exit

Enter a number that corresponds to your choice > 2

Introduce el id del libro: ABCD1
Introduce el titulo del libro: titulo
 (1) Use
 (2) Insert
 (3) Print
 (4) Exit

Enter a number that corresponds to your choice > 3

Introduce la profundidad hasta donde quieres imprimir:1
  ABCD (0): 8
 (1) Use
 (2) Insert
 (3) Print
 (4) Exit

Enter a number that corresponds to your choice >
```

Esto es el estado del fichero tras la inserción;

```
diego@DESKTOP-29VLGGP:/mnt/c/Universidad/Segundo/FFBB/FFBB-Practicas/Practica3$ xxd data.dat
00000000: ffff ffff 0000 0000 4142 4344 0700 0000  ........ABCD....
00000010: 7469 7475 6c6f 0a                        titulo.
```

Ahora realizaremos otra dos inserciones una a cada “lado” en el árbol de índices:

```
diego@DESKTOP-29VLGGP:/mnt/c/Universidad/Segundo/FFBB/FFBB-Practicas/Practica3$ make -f makefile.cod run
./menu
 (1) Use
 (2) Insert
 (3) Print
 (4) Exit

Enter a number that corresponds to your choice > 1

Introduce el nombre del fichero de datos con la extensión incluida:data.dat
 (1) Use
 (2) Insert
 (3) Print
 (4) Exit

Enter a number that corresponds to your choice > 2

Introduce el id del libro: AABC1
Introduce el titulo del libro: titulo2
 (1) Use
 (2) Insert
 (3) Print
 (4) Exit

Enter a number that corresponds to your choice > 2

Introduce el id del libro: BBCD1
Introduce el titulo del libro: titulo3
 (1) Use
 (2) Insert
 (3) Print
 (4) Exit

Enter a number that corresponds to your choice > 3

Introduce la profundidad hasta donde quieres imprimir:1
  ABCD (0): 8
    l AABC (1): 23
    r BBCD (2): 39
 (1) Use
 (2) Insert
 (3) Print
 (4) Exit

Enter a number that corresponds to your choice >
```

Estado del fichero de datos e índices tras ambas inserciones:
Datos:

```
diego@DESKTOP-29VLGGP:/mnt/c/Universidad/Segundo/FFBB/FFBB-Practicas/Practica3$ xxd data.dat
00000000: ffff ffff 0000 0000 4142 4344 0700 0000  ........ABCD....
00000010: 7469 7475 6c6f 0a41 4142 4308 0000 0074  titulo.AABC....t
00000020: 6974 756c 6f32 0a42 4243 4408 0000 0074  itulo2.BBCD....t
00000030: 6974 756c 6f33 0a                        itulo3.
```

Índices:

```
diego@DESKTOP-29VLGGP:/mnt/c/Universidad/Segundo/FFBB/FFBB-Practicas/Practica3$ xxd data.idx
00000000: 0000 0000 ffff ffff 4142 4344 0100 0000  ........ABCD....
00000010: 0200 0000 ffff ffff 0800 0000 4141 4243  ............AABC
00000020: ffff ffff ffff ffff 0000 0000 1700 0000  ................
00000030: 4242 4344 ffff ffff ffff ffff 0000 0000  BBCD............
00000040: 2700 0000                                '...
```

Se puede observar con un poco de trabajo que ambos ficheros son correctos. Es claramente más cómodo centrarnos en el árbol de índices, ya mostrado.

Makefile:

El fichero makefile.cod proporcionado en el zip permite que con el comando make -f makefile.cod compile nuestro código y con make -f makefile.cod run se lance la simple aplicación de menú ya comentada. Además el archivo Makefile permite compilar el programa tester proporcionado en la práctica y extendido por nosotros.

Conclusión:

En resumen, la práctica nos pedía implementar una versión muy simple de una base de datos almacenando la información en ficheros con una estructura de índices en forma de árbol binario para mejorar la eficiencia en búsqueda e inserción. Gracias a esta práctica nos hemos familiarizado con el uso de ficheros como formato de almacenamiento.Además, como elemento a destacar, nos ha resultado especialmente desafiante la batería de pruebas para comprobar el correcto funcionamiento de las funciones y de nuevo nos hemos dado cuenta de lo difícil que puede llegar a ser asegurar el correcto funcionamiento de nuestro programa.