Estructuras de Datos y de la Información

Práctica 9

Índice Primario mediante un Árbol Multicamino

OBJETIVO: Esta práctica va dirigida a los alumnos que no hayan superado la parte práctica de la asignatura correspondiente al primer cuatrimestre y equivale a un examen de prácticas.

Los alumnos que vayan a presentar esta práctica tendrán que comunicarlo al profesor antes del día 27 de mayo mediante un mensaje a jaglez@etsii.ull.es.

El aula, la fecha y la hora de corrección se fijarán en función de la disponibilidad de laboratorios y se comunicará a los alumnos interesados con una semana de antelación. En cualquier caso, la fecha de presentación será posterior al 13 de junio y anterior al 22 de junio (fecha del primer examen de la asignatura en la convocatoria de junio).

ENUNCIADO: En esta práctica se implementa un programa en lenguaje C++ para trabajar con un fichero organizado en registros de información bibliográfica. El fichero está indexado mediante un índice primario estructurado en forma de árbol multicamino de orden m, AMC(m), almacenado en memoria secundaria.

El programa principal presenta el siguiente menú de opciones:

```
1.- Abrir/crear fichero.
2.- Mostrar todos los registros.
3.- Buscar un registro por ISBN.
4.- Insertar un registro.
5.- Borrar un registro.
6.- Cerrar fichero.
0.- Salir.
```

A continuación se describe el comportamiento del programa al seleccionar las opciones.

- En la opción 1 se solicita el nombre del fichero de datos y si existe se abre. Si no existe se crea el fichero de datos y el fichero índice.
- En la opción 2 se muestran todos los registros del fichero por pantalla ordenados por el valor de la clave primaria.
- En la opción 3 se solicita un valor de ISBN y se busca el registro. Si se encuentra, se muestra el registro bibliográfico, el nivel del nodo y la posición dentro del nodo que ocupa la clave en el AMC.
- En la opción 4 se solicitan los datos de un registro a insertar.
- En la opción 5 se solicita el valor de ISBN del registro a eliminar.
- En la opción 6 se cierra el fichero de trabajo y se coloca una marca en el registro cabecera del fichero de datos para indicar que el índice está sincronizado. De esta forma, si al abrir un fichero de datos no se encontrara la marca de sincronización habría que generar el índice desde el fichero de datos.

NOTAS DE IMPLEMENTACION:

- Siguiendo la descripción de la práctica 4, implementa la clase `FileBuf<RECORD>` que gestiona un fichero de registros (abrir, crear, leer registro, escribir registro, añadir registro, etc) en cualquiera de los siguientes formatos.

  1. Registros de longitud fija con campos de longitud fija
  2. Registros de longitud variable con indicador de longitud y con campos separador por delimitador
  3. Registros de longitud fija y campos de longitud variable con indicador de longitud

- El índice primario se define mediante la clase abstracta `PrimaryIndex<KEY>`, dada en la práctica 6, que contiene la especificación de las operaciones del índice (buscar, insertar, borrar, recorrer).

- Cada registro del fichero de datos tiene una entrada en el índice que contiene un valor de la clave `KEY` y la posición que ocupa el registro en el fichero de datos `ref`.

```
struct PrimaryIndexEntry {
  KEY key;
  Tpointer ref;
};
```

- La clase `FileIndex<RECORD,KEY>` implementa las operaciones de un fichero de registros indexado. Hereda de la clase `FileBuf<RECORD>` las operaciones de manejo de registros, y redefine estás operaciones utilizando el índice primario. En la práctica 6 se describe la implementación de esta clase.

```
template <class RECORD, class KEY>
class FileIndex: public FileBuf<RECORD> {

  int Open(char *file, int mode = ios::in | ios::out |
                       ios::binary | ios::nocreate) {
    FileBuf<RECORD>::Open(file);
    Index = new AMC<KEY, m>(file_idx);
  }
  ...
  protected:
    PrimaryIndex<KEY> *Index;
};
```

- El programa principal instancia un objeto del tipo `FileIndex<Libro,KeyISBN>` para trabajar con el fichero de bibliografía indexado. La clase `Libro` se corresponde con el tipo RECORD almacenado en el fichero; y la clase `KeyISBN` permite trabajar con los valores de clave del tiempo ISBN.

```
class KeyISBN {
  char ISBN[TAM_ISBN];
  KeyISBN(const char* key) {strncpy(ISBN, key, TAM_ISBN);}
  bool operator==(const KeyISBN& key) {
    return strncmp(ISBN, key, TAM_ISBN) == 0;
  }
 ...
};
```

- La clase `AMC<class KEY, int m>` implementa un índice primario organizado en forma de AMC(m). Hereda la definición de la clase `PrimaryIndex<KEY>`. El parámetro `KEY` indica el tipo de las claves almacenadas en los nodos del árbol; y el parámetro `m` indica el orden del AMC.

```
template <class KEY, int m>
class AMC: public PrimaryIndex<KEY> {
  public:
    AMC(char* name);       // Abre/crea el fichero del AMC
    ~AMC();                // Cierra el fichero del AMC
    ...                    // Métodos heredados de PrimaryIndex
  protected:
    TPointer Root;
    FileBuf<NodeAMC<PrimayIndexEntry<KEY>,m> > T;
    stack<TPila> Pila;
};
```

Estructuras de Datos y de la Información

- Los algoritmos que describen las operaciones de búsqueda e inserción en un AMC(m) se encuentran en las transparencias del tema 6 del curso. Estas operaciones se apoyan en una estructura auxiliar de pila (stack) que almacena ítems de tipo `TPila`:

```
struct TPila {
  NodeAMC<KEY,m> Nodo;
  TPointer Direccion;          // Dirección del nodo en el fichero
  TSize Posicion;              // Posición de la clave en el nodo
};
```

- La implementación del método `AMC<KEY,m>::Next()` se corresponde con el procedimiento `Siguiente()` mostrado en las transparencias del tema 6 del curso.

- El método `AMC<KEY,m>::Rewind()` consiste en realizar un descenso desde la raíz por el enlace de la izquierda hasta alcanzar la hoja, y nos indica el valor de clave más pequeño almacenado en el AMC(m).

- A continuación se presentar el algoritmo de eliminación de claves en un AMC(m) que se encuentra en las transparencias del tema 7 del curso para eliminar claves de un árbol B.

```
bool ExtraerAMC(TPointer R, KEY X) {
  buscarAMC(R, X, Encontrado, Pila);
  if (!Encontrado) return FALSE;
  (NodoP, Pos, DirP) <- Pila.Pop();
  if(NodoP.A[Pos] != PNULL) {     // Si no es una hoja
    DirQ = NodoP.A[pos];          // se busca el sucesor de la clave.
    LeerNodo(NodoQ, DirQ;);
    Pila.Push(NodoQ, 0, DirQ;);
    while(NodoQ.A[0] != PNULL) {
      DirQ = NodoQ.A[0];
      LeerNodo(NodoQ, DirQ;);
      Pila.Push(NodoQ, 0, DirQ);
    }
    NodoP.K[Pos] = NodoQ.K[1];    // Se sustituye la clave buscada
    EscribirNodo(NodoP, DirP);    // por su sucesor.
    NodoP = NodoQ;
    Pos = 1;
    DirP = DirQ;
  }
  ExtrarClave(NodoP, Pos);        // Siempre se extrae en una hoja.
  if(NodoP.nk == 0)               // Si el nodo está vacío se libera,
    BorrarNodo(DirP);
    (NodoP, Pos, DirP) <- Pila.Pop();
    (NodoP, Pos, DirP) <- Pila.Pop();  // y se actualiza el
    NodoP.A[Pos] = PNULL;              // enlace del nodo padre
  }
  EscribirNodo(NodoP, DirP);
  return TRUE;
}
```

- Cada nodo del AMC(m) se implementa mediante una plantilla `NodeAMC<class KEY,TSize m>` que contiene `m-1` entradas del tipo `PrimaryIndexEntry<KeyISBN>`, `m` enlaces a los nodos hijos y el contador `nk` de claves guardadas en el nodo.

- Cada nodo del AMC(m) es un registro de longitud fija que se almacena en un fichero y se gestiona mediante un objeto de tipo `<NodeAMC<PrimayIndexEntry<KEY>,m> >`.

Estructuras de Datos y de la Información

- A continuación se presenta la descripción de la clase NodeAMC<KEY,m> definida en la práctica 7:

```
template <class KEY, TSize m>
class NodeAMC {
  private:
    KEY K[m-1];
    Tpointer A[m];
    TSize nK;

  public:
    // Constructores
    NodoAMC() {nk = 0; for(TSize i = 0; i < m; A[i++] = -1);}
    NodoAMC(const KEY& k) {for(TSize i=0, nk=1, K[0]=k; i < m; A[i++]=-1);}

    // Métodos para acceder a los campos privados
    KEY& operator[](const int i) {return K[i-1];}
    KEY operator[](const int i) const {return K[i-1];}
    TPointer& link(const TSize i) {return A[i];}
    TSize nk() {return nK;}

    // Método de búsqueda binaria en el vector de claves. Si lo encuentra,
    // devuelve la posición; sino devuelve el enlace al siguiente nodo.
    bool Search(const KEY& k, TSize& pos);

    // Método de inserción en orden de una clave y un puntero. Los valores
    // de clave no se repiten. Incrementa el contador de claves nK.
    void Insert(const KEY& k, const TPointer l);

    // Método de eliminación de una clave y un puntero. Decrementa el
    // contador de claves nK.
    void Remove(const KEY& k);

    // Métodos utilizados para almacenar en un FileBuf.
    TSize Pack(IOBuffer* buf);
    TSize UnPack(IOBuffer* buf);
};
```