Estructuras de Datos y de la Información

Práctica 8

Árbol Multicamino

OBJETIVO: En esta práctica se implementa un índice primario mediante un árbol multicamino de orden m, AMC(m), almacenado en memoria secundaria.

FECHA: La práctica se presentará entre los días 1 y 17 de marzo de 2005.

ENUNCIADO: Modificar el tipo de dato `FileIndex`, desarrollado en la práctica 6, para que implemente las operaciones definidas para el tipo `FileBuf` sobre registros de información bibliográfica con un índice primario en forma de AMC(m) definido sobre el campo ISBN.

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. (Opcional)
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:

En la clase `FileIndex<RECORD, KEY>`, implementada en la práctica 6, sólo hay que modificar la llamada al constructor del objeto `PrimaryIndex<KEY>`.

```
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);
    // Cambiar extensión al fichero de índice
    // Leer m en la cabecera el fichero
    Index = new AMC<KEY, m>(file_idx);
    ...
  }
  ...
  protected:
    PrimaryIndex<KEY> *Index;
};
```

Un AMC(m) se implementa mediante la plantilla `AMC<class KEY, int m>`, donde 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);
    ~AMC();
    // Métodos heredados de PrimaryIndex
  protected:
    TPointer Root;
    FileBuf<NodeAMC<PrimayIndexEntry<KEY>,m>> T(sizeof(NodeAMC<KEY, m>));

    stack<TPila> Pila;
};
```

El constructor abre/crea el fichero que contiene los nodos del AMC. En la cabecera de este fichero se guarda la dirección del nodo raíz. Para optimizar los accesos, la dirección del nodo raíz se guarda en el campo `Root`.

Cada nodo del AMC(m) es un registro de longitud fija implementado mediante la plantilla `NodeAMC<class KEY, TSize m>` desarrollada en la práctica 7. Como tipo `KEY` de la plantilla se utiliza el `struct PrimaryIndexEntry` definido en la práctica 6 y que permite utilizar la estructura como índice primario.

Como se muestra en las transparencias del AMC, las operaciones de búsqueda e inserción utilizan una estructura auxiliar de pila, `TPila`. Para facilitar la implementación de esta pila se puede utilizar la plantilla `stack` de la STL, o se puede implementar la pila mediante un array. En cualquier caso, la pila contendrá registros de la forma:

```
struct TPila {
  NodeAMC<KEY,m> Nodo;
  TPointer Direccion;
  TSize Posicion;
};
```

Las operaciones de inserción y extracción de la pila se denotan respectivamente:

```
(Nodo, int, TPointer) <- Pila.Pop();   // Saca una estructura de la pila
Pila.Push(Nodo, int, TPointer);        // Guarda una estructura en la pila
```

La implementación del método `AMC<KEY,m>::Next()` se corresponde con el procedimiento `Siguiente()` mostrado en las transparencias del tema 6. Por su parte, el método `AMC<KEY,m>::Rewind()` consiste en realizar un descenso desde la raíz por los enlaces de la izquierda hasta alcanzar la hoja y el valor de clave más pequeño almacenado en el AMC(m).

Existe una correspondencia directa entre los procedimientos auxiliares descritos en las transparencias del tema 6 y los métodos de la clase `FileBuf`.

```
LeerNodo(NodoAMC, TPointer) ⇔ FileBuf<NodeAMC>::Read(NodoAMC, TPointer)
EscribirNodo(NodoAMC, TPointer)⇔FileBuf<NodeAMC>::Write(NodoAMC,TPointer)
BorrarNodo(TPointer) ⇔ FileBuf<NodeAMC>::Remove(TPointer)
```

Estructuras de Datos y de la Información

A continuación se presenta el algoritmo de eliminación de claves en un AMC(m) que se mostró en las transparencias del tema 7 al 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;
}
```