UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ
Facultad de Ingeniería
Área de Computación e Informática

"Organizaciones de Archivos"

por

Lara Moreno Jesús Alejandro
Alejandra

Ing. Gerardo Padilla Lomelí
Profesor

REPORTE DE PROYECTO PARA LA MATERIA
DE ESTRUCTURAS DE ARCHIVOS

Septiembre, 2016

Índice general

Capítulo 1

Introducción

Un aspecto fundamental de los sistemas de información es la organización sobre la cual maneja sus datos ya que de ello depende el desempeño de los programas. En el siguiente contenido desarrollaremos teórica y prácticamente las diferentes organizaciones de archivos, las cuales implementamos para diccionario de datos.

Capítulo 2

Organizaciones de archivos

Las técnicas utilizadas para representar y almacenar registros en archivos se llama organización de archivos. Las cuatro técnicas fundamentales de organización de archivos que analizaremos son las siguientes:

- Secuencial
- Secuencial indexado
- Archivos indexados con arboles b+
- Archivos directos con (hash dinámica)
- Archivos directos con (hash estática)
- Organización de archivos multillave

Capítulo 3

Operaciones sobre archivo

La manera como se usar el archivo es un factor importante para determinar como se debe organizar el archivo. Dos aspectos importantes sobre el uso de archivos son su modo de utilización y la naturaleza de las operaciones sobre el archivo. Las operaciones básicas que se ejecutan sobre los archivos son las siguientes:

1. Creación
2. Actualización, incluyendo:
    a. inserción de registros
    b. modificación de registros
    c. supresión de registros
3. Recuperación incluyendo:
    a. Consulta
    b. Generación de reportes
4. Mantenimiento, incluyendo:
    a. Estructuración
    b. Reorganización

3.1. Creación de un archivo

La creación inicial de un archivo es conocida también como la carga del archivo. El grueso del trabajo en la creación de archivos incluye la validación de datos. En algunas implantaciones, primero se asigna el espacio para el archivo y después los datos son cargados dentro de ese "esqueleto" de archivo. En otras implantaciones, el archivo se construye registro por registro.

```
string CDiccionario::abrir_Diccionario(char n[20]) {
    std::stringstream buffer;
    CEntidad aux_entidad;
    long auxDir_siguiente;
    this->lista_entidades.clear();
    //Si el archivo esta abierto se cierra
    if(this->ptr_Archivo != NULL) {
        std::fclose(this->ptr_Archivo);
        this->cabecera = -1;
    }
    /*Se abre el archivo considerando que existe*/
    this->ptr_Archivo = std::fopen(n, "r+b");
    /*Si el archivo se pudo abrir se cargan sus datos*/
    if(this->ptr_Archivo != NULL) {
        buffer << "Diccionario " << n << " Abierto!!";
        /*Se lee su cabecera en el archivo y sea actualiza la cabecera de la clase*/
        std::fread(&this->cabecera, sizeof(long), 1, this->ptr_Archivo);
        buffer << " Cabecera en " << this->cabecera  << std::endl;
        /*Se cargas todos sus Entidades y atributos*/
        if(this->cabecera != -1) {
            std::fseek(this->ptr_Archivo, this->cabecera, SEEK_SET);
            std::fread(&aux_entidad, sizeof(CEntidad), 1, this->ptr_Archivo);
            aux_entidad.inicia_ListaAtributos();
            aux_entidad.carga_Atributos(this->ptr_Archivo);
            this->lista_entidades.push_back(aux_entidad);
            while(aux_entidad.dameDir_Siguiente() != -1) {
                auxDir_siguiente = aux_entidad.dameDir_Siguiente();
                std::fseek(this->ptr_Archivo, auxDir_siguiente, SEEK_SET);
                std::fread(&aux_entidad, sizeof(CEntidad), 1, this->ptr_Archivo);
                aux_entidad.inicia_ListaAtributos();
                aux_entidad.carga_Atributos(this->ptr_Archivo);
                this->lista_entidades.push_back(aux_entidad);
            }
            /*Cargadas todas las entidades se ordena la lista*/
            this->lista_entidades.sort();
        }
    }
    /*Si el archivo no se pudo abrir se crea automaticamente*/
    else {
        this->ptr_Archivo = std::fopen(n, "w+b");
        /*Si el archivo se pudo crear*/
        if(this->ptr_Archivo != NULL) {
                buffer << "Diccionario " << n << " Creado!!" << std::endl;
            /*Escribimos la cabecera vacia en el archivo nuevo*/
            std::fwrite(&this->cabecera, sizeof(long), 1, this->ptr_Archivo);
        }
        /*Si no se pudo crear el diccionario*/
        else {
            buffer << "No se pudo crear el diccionario " << n << std::endl;
        }
    }return buffer.str();
}
```

3.2. Actualización de un archivo

Cambiar el contenido de un archivo maestro para hacer que refleje un momento transitorio mas actual del mundo real es a lo cual se llama, actualización de archivos.

1. La inserción de nuevos registros, por ejemplo, la edición de un registro para un empleado de nuevo ingreso a la compañía.

2. La modificación de datos a registros que ya existen en el archivo, por ejemplo, cambiar el sueldo del empleado, cambiar el indicativo de estado del empleado(activo, no activo, de licencia)

3. La supresión de registros del archivo, esto es, borrar el registro de un empleado que salió de la compañía.

De esta manera el archivo muestra una imagen mas actual de la realidad.

3.3. Recuperación de información de un archivo

El acceso a un archivo con el propósito de extraer información significativa es llamado recuperación de información. Existen dos clases de recuperación de información: consultas y generación de reportes. Estas dos clases pueden distinguir de acuerdo a volumen de información que producen. Una consulta produce un volumen relativamente mínimo, mientras que un reporte puede crear muchas paginas de salida de información.

3.4. Mantenimientos de archivos

Cambios hechos sobre archivos para mejorar la eficiencia de los programas que los accesan son los conocidos como actividades de mantenimiento. Existen dos clases de operaciones de mantenimiento básicas, las cuales son: reestructuración y reorganización. La reestructuración de un archivo implica que es necesario aplicar cambios estructurales, dentro del contexto de la misma técnica de organización de archivos. La reorganización implica cambiar la organización de un archivo a otro tipo de organización.

Capítulo 4

Diccionario de datos

Es un conjunto de metadatos que contiene las características lógicas y puntuales de los datos que se van a utilizar en un sistema que se programa, incluyendo nombres, descripción, alias, contenido y organización.

4.1. Descripción de un diccionario de datos

Es un catálogo de los elementos en un sistema. Como su nombre lo sugiere, estos elementos se centran alrededor de los datos y la forma en que están estructurados para satisfacer los requerimientos de los usuarios y las necesidades de la organización. En un diccionario de datos se encuentra la lista de todos los elementos que forman parte del flujo de datos en todo el sistema. Los elementos más importantes son flujos de datos, almacenes de datos y procesos. El diccionario guarda los detalles y descripciones de todos estos elementos.

4.2. Operaciones basicas de un diccionario de datos.

Altas: La modificación de datos a registros que ya existen en el archivo, por ejemplo, cambiar el sueldo del empleado, cambiar el indicativo de estado del empleado (activo, no activo, de licencia).

Bajas: La supresión de registros del archivo, esto es, borrar el registro de un empleado que salió de la compañía.

Modificaciones: La modificación de datos a registros que ya existen en el archivo, por ejemplo, cambiar el sueldo del empleado, cambiar el indicativo de estado del empleado(activo, no activo, de licencia)

4.3. Operaciones basicas entidades

Una entidad en un diccionario de datos requiere un objeto entidad, el cual contiene los campos para el nombre, un campo para almacenar su propia dirección (dir_entidad); una dirección a su primer atributo (dir_atr); una dirección a su primer dato (dir_dato) y por utimo una dirección a su siguiente entidad. como se muestra en la imagen.

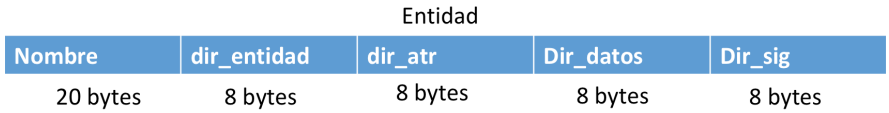

Figura 4.1: Campos de una entidad.

4.3.1. Altas

Al insertar una entidad nueva existen dos casos, el primer caso se presenta cuando el archivo no contiene ninguna entidad y la cabecera se encuentra en -1 y el segundo caso se presenta cuando el archivo contiene por lo menos una entidad. A continuación describiremos cada caso.

Caso uno: Al verificar la cabecera que contenga -1, nos situamos al final del archivo y almacenamos esa dirección, escribimos la nueva entidad a partir de la direccion almacenada y actualizamo la cabecera con dicha dirección.

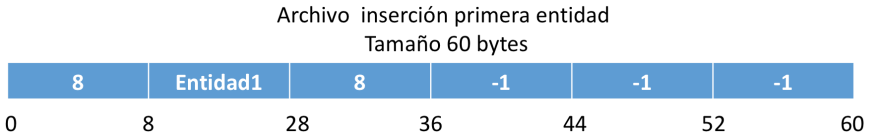

Figura 4.2: Inserción de entidad caso uno.

```
if(this->lista_entidades.empty())
{
        this->cabecera = dir_nueva;
    //Se actualiza la cabecera en el archivo
    std::fseek(this->ptr_Archivo, 0, SEEK_SET);
    std::fwrite(&this->cabecera, sizeof(long), 1, this->ptr_Archivo);
    //Se agrega la nueva entidad a la lista
    this->lista_entidades.push_back(*nueva_entidad);
    buffer << "Se agrego la entidad " << n << " al diccionario" << std::endl;
}
```

Explicación codigo..

Caso dos: Si la cabecera es diferente de -1 iteramos hasta encontrar la ultima entidad, agregamos la entidad al final del archivo, y actualizamos la dirección al la siguiente entidad de la ultima entidad a la entidad nueva.

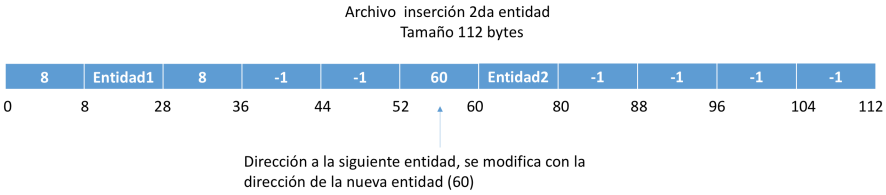

Figura 4.3: Inserción de entodad caso dos.

```
else
{
        //recorre la lista y busca el ultimo elemento
    atras_entidad = this->lista_entidades.begin();
    while(atras_entidad != this->lista_entidades.end()
          && atras_entidad->dameDir_Siguiente() != -1)
    {
        atras_entidad++;
    }
    //Verifica si encontro la ultima entidad al final
    if(atras_entidad->dameDir_Siguiente() == -1)
    {
        fseek(this->ptr_Archivo, atras_entidad->dameDir_Entidad(), SEEK_SET);
        atras_entidad->ponDir_Siguiente(dir_nueva);
        aux_entidad = *atras_entidad;
        fwrite(&aux_entidad, sizeof(CEntidad), 1, this->ptr_Archivo);
        this->lista_entidades.push_back(*nueva_entidad);
        buffer << "Se agrego la entidad " << n << " al diccionario" << endl;
     }
}
```

Explicación codigo..

4.3.2. Bajas

Para las baja de una entidad existen tres casos.

Caso uno: Si el archivo tiene únicamente una entidad, por lo tanto la cabecera apuntara a esa entidad, para darla de baja únicamente tenemos que actualizar la cabecera con -1 indicando que el archivo quedo vacío.

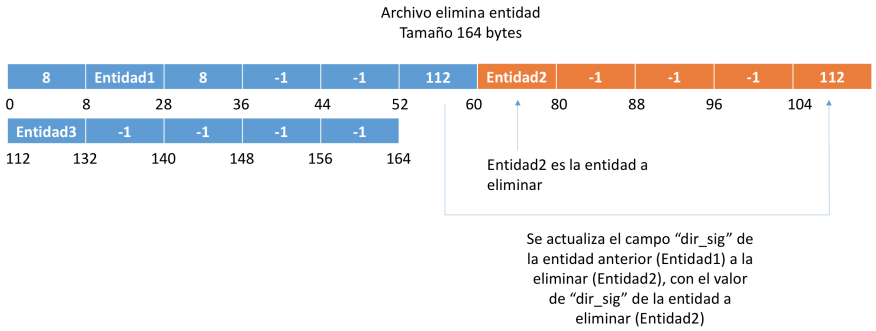

Figura 4.4: Eliminación de un campo, la eliminación solo se actualizan los apuntadores.

```
if(this->cabecera == aux_entidad.dameDir_Entidad())
{
        this->cabecera = aux_entidad.dameDir_Siguiente();
    fseek(this->ptr_Archivo, 0, SEEK_SET);
    fwrite(&this->cabecera, sizeof(long), 1, this->ptr_Archivo);
    buffer << "Es_es_el_primero" << iterador->dame_Nombre() << endl;
    /*Elimino la entidad de la lista*/
        iterador = this->lista_entidades.begin();
        advance(iterador, n);
        this->lista_entidades.remove(*iterador);
}
```

explicación..

Caso dos: Cuando la entidad a eliminar no es la primera entidad ni la ultima entidad, esto quiere decir que tiene una entidad anterior a ésta y una entidad después imagen 4.6, para hacer la baja tenemos que actualizar el campo "sig_ent" de la entidad anterior, con con el campo "sig_ent" de la entidad a elimina, de esta manera ligamos la entidad anterior con la entidad siguiente.

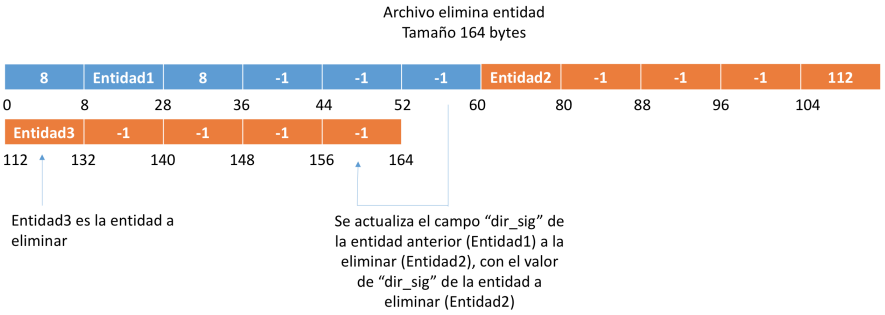

Figura 4.5: Eliminación al final del diccionario.

Caso tres: Si el archivo tiene por lo menos mas de 2 entidades y la entidad a eliminar es la ultima entidad. Para hacer la baja de la ultima entidad necesitamos actualizar el campo "sig_ent" de la entidad anterior a eliminar, con un -1 indicando que ahora la entidad anterior es la ultima.

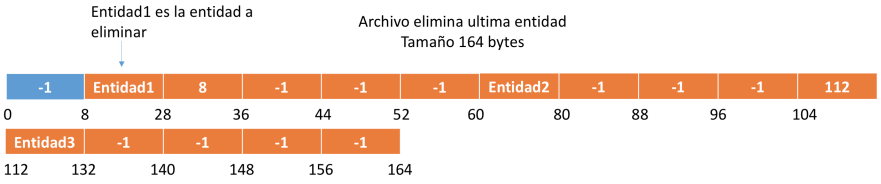

Figura 4.6: Eliminación al inicio del diccionario.

```
if(iterador->dameDir_Siguiente() == aux_entidad.dameDir_Entidad())
{
        buffer << "Encontro el anterior " << iterador->dame_Nombre() << endl;
    iterador->ponDir_Siguiente(aux_entidad.dameDir_Siguiente());
    fseek(this->ptr_Archivo, iterador->dameDir_Entidad(), SEEK_SET);
    aux_entidad = *iterador;
    fwrite(&aux_entidad, sizeof(CEntidad), 1, this->ptr_Archivo);
    /*Elimino la entidad de la lista*/
    iterador = this->lista_entidades.begin();
    advance(iterador, n);
    this->lista_entidades.remove(*iterador);
}
```

explicación..

4.3.3. Modificaciones

Para la modificación de una entidad, el único campo a modificar sería el campo nombre, para realizar esta operación debemos buscar la entidad a modificar, cargarla en memoria para modificarla ahí, editamos su nombre y por ultimo debemos de escribir la entidad ya modificada en el mismo lugar en el que se encontraba.

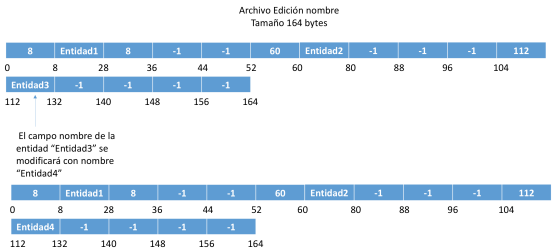

Figura 4.7: Modificación de una entidad

```
if(iterador != this->lista_entidades.end())
{
        /*El metodo advance el iterador n veces si seleccione el elemento 2
    * advance recorre el iterador 2 veces
    */
    advance(iterador, index-1);
    strcpy(aux_nombre, iterador->dame_Nombre());
    iterador->pon_Nombre(n);
    aux_entidad = *iterador;
    fseek(this->ptr_Archivo, iterador->dameDir_Entidad(), SEEK_SET);
    fwrite(&aux_entidad, sizeof(CEntidad), 1, this->ptr_Archivo);
    buffer << "Se modifico la Entidad "
           << aux_nombre << " -> " << aux_entidad.dame_Nombre()
           << endl;
    this->lista_entidades.sort();
}
```

explicación..

4.4. Operaciones basicas de atributos

4.4.1. Altas

Para realizar una alta de un atributo, primero debemos seleccionar la entidad a la cual se agregará el atributos, ya posicionados en el la entidiad que se agregará el nuevo tributo existen 2 casos.

Primer caso: Cuando el atributo a agregar es el primer atributo, debemos agregar al final del archivo y actualizar el campo "dir_atr" de la entidad seleccionada con la dirección donde se agrego el nuevo atributo.

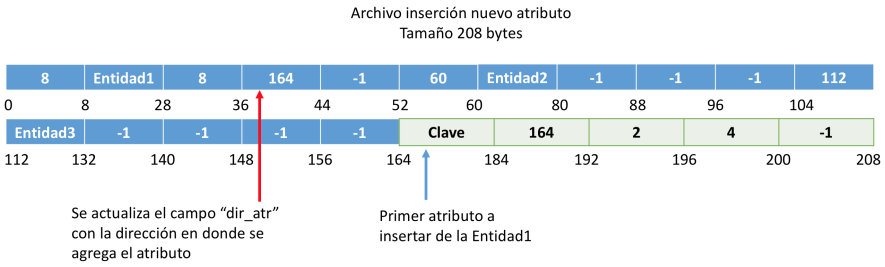

Figura 4.8: Inserción de atributos caso uno.

Primer caso: Cuando en el archivo por lo menos tiene 1 atributo de la entidad seleccionada, esto quiere decir que debemos actualizar el campo "sig_atr" del del ultimo atributo con la dirección donde se inserto el nuevo

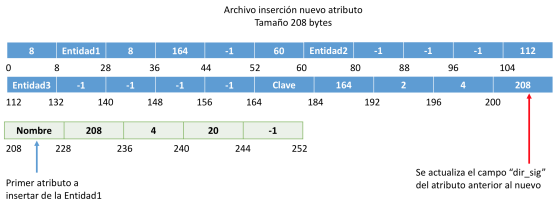

Figura 4.9: Insercion de atributos caso dos.

4.4.2. Bajas

4.4.3. Modificaciones

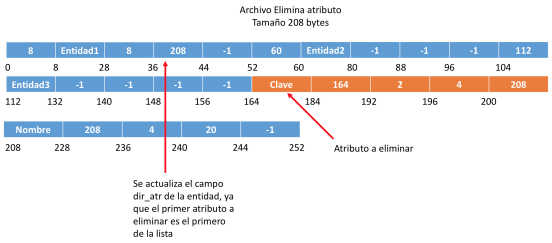

Figura 4.10: La baja de atributos implica actualizar los apuntadores en el diccionario.

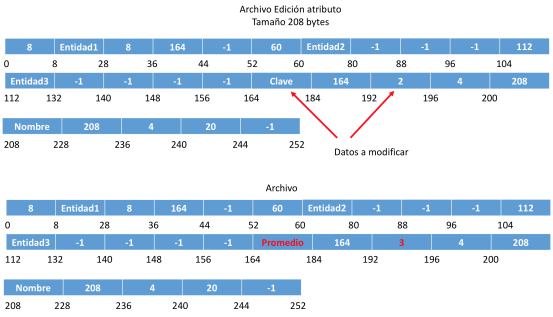

Figura 4.11: Edición de un atributo.

Capítulo 5

Organización de archivos secuenciales

La manera básica de organizar un conjunto de registros, que forman un archivo, es utilizando una organización secuencial. En un archivo organizado secuencialmente, los registros quedan grabados consecutivamente cuando el archivo se crea y deben accesarse consecutivamente cuando el archivo se usa como se muestra a continuación:

En la mayoría de los casos, los registros de un archivo secuencial quedan ordenados de acuerdo con el valor de algún campo de cada registro. Semejante archivo se dice que es un archivo ordenado; el campo, o los campos, cuyo valores se utiliza para determinar el ordenamiento es conocido como llave de ordenamiento. Un archivo puede ordenarse ascendente o descendentemente con base en la llave de ordenamiento, la cual puede constar de uno o mas campos.

Bibliografía

[1] Universidad Autónoma de San Luis Potosí, *Antecedentes Históricos* `http://www.uaslp.mx/Spanish/Institucional/anthist/Paginas/default.aspx`

[2] Mary E. S. Loomis *Estructura de datos y organización de archivos*, Segunda edición.