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

Estructuras de archivos I

1 Introducción 1

2 Conceptos 2
- 2.1 Archivo . . . 2
- 2.2 Entidad . . . 2
- 2.3 Atributos . . . 2
- 2.4 Datos . . . 2

3 Organizaciones de archivos 3

4 Operaciones sobre archivo 4
- 4.1 Creación de un archivo . . . 4
- 4.2 Actualización de un archivo . . . 6
- 4.3 Recuperación de información de un archivo . . . 6
- 4.4 Mantenimientos de archivos . . . 6

5 Diccionario de datos 7
- 5.1 Descripción de un diccionario de datos . . . 7
- 5.2 Operaciones basicas de un diccionario de datos . . . 7
- 5.3 Ejemplos . . . 8
  - 5.3.1 Altas . . . 8
  - 5.3.2 Codigo en C++ de Altas . . . 9
  - 5.3.3 Bajas . . . 10
  - 5.3.4 Codigo en C++ de Bajas . . . 11
  - 5.3.5 Modificaciones . . . 11
  - 5.3.6 Codigo en C++ de Modificaciones . . . 12

6 Organización de archivos secuenciales 13

Bibliografía 15

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

Conceptos

2.1. Archivo

Un archivo es una colección de registros logicamente relacionados. Por lo regular, todos los registros de un archivo tienen un formato único, aunque existen archivos con multiples formatos de registros.

2.2. Entidad

Las entidades se describen en la estructura de la base de datos empleando un modelo de datos. Cada entidad está constituida por uno o más atributos. Por ejemplo, la entidad .Alumno"podría tener los atributos: nombre, apellido, año de nacimiento, etc.

2.3. Atributos

Los atributos son las características por medio de los cuales se puede describir una entidad.

2.4. Datos

Un dato es una representación simbólica (numérica, alfabética, algorítmica, espacial, etc.) de un atributo o variable cuantitativa o cualitativa. Los datos describen hechos empíricos, sucesos y entidades.

Capítulo 3

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 4

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

4.1. 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();
}
```

4.2. 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.

4.3. 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.

4.4. 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 5

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.

5.1. 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.

5.2. 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)

5.3. Ejemplos

5.3.1. Altas

Figura 5.1: Campos de una entidad

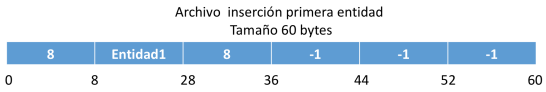

Figura 5.2: Insercion primera de una entidad en el diccionario.

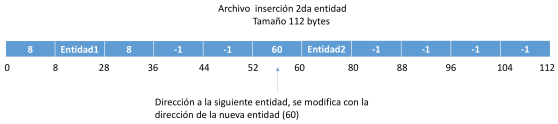

Figura 5.3: Insercion segunda de una entidad en el diccionario.

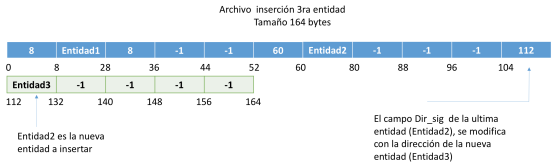

Figura 5.4: Insercion tercera de una entidad en el diccionario.

5.3.2. Codigo en C++ de Altas

```
string CDiccionario::agrega_Entidad(char n[20])
{
    long dir_nueva;
    std::stringstream buffer;
    CEntidad *nueva_entidad, aux_entidad;
    std::list<CEntidad>::iterator atras_entidad;

    if(this->ptr_Archivo != NULL)
    {
        /*Se pone el apuntador al final de archivo*/
        std::fseek(this->ptr_Archivo, 0, SEEK_END);
        dir_nueva = std::ftell(this->ptr_Archivo);
        nueva_entidad = new CEntidad(n, dir_nueva);
        std::fwrite(nueva_entidad, sizeof(CEntidad), 1, this->ptr_Archivo);

        //Si el diccionario esta vacio agrega y actualiza la cabecera
        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;

        }
        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)
            {
                std::fseek(this->ptr_Archivo, atras_entidad->dameDir_Entidad(), SEEK_SET)
                    ;
                atras_entidad->ponDir_Siguiente(dir_nueva);
                aux_entidad = *atras_entidad;
                std::fwrite(&aux_entidad, sizeof(CEntidad), 1, this->ptr_Archivo);
                this->lista_entidades.push_back(*nueva_entidad);
                buffer << "Se agrego la entidad " << n << " al diccionario" << std::endl;
            }
        }
        //Ordena a lista por nombres, esto es posible por la sobrecarga del operador ==
        this->lista_entidades.sort();
    }
    //Retorna un buffer que despues es mostrado al inicio del menu.
    return buffer.str();
```

5.3.3. Bajas

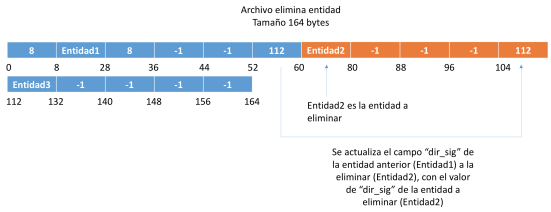

Figura 5.5: Eliminación de un campo, la eliminación solo se actualizan los apuntadores.

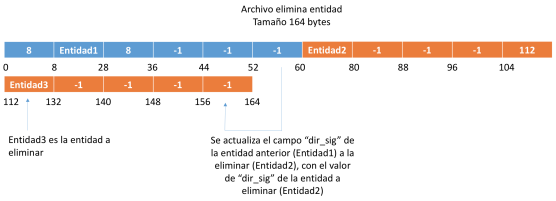

Figura 5.6: Eliminación al final del diccionario.

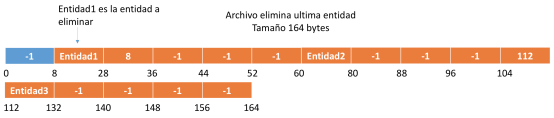

Figura 5.7: Eliminación al inicio del diccionario.

5.3.4. Codigo en C++ de Bajas

```
string CDiccionario::elimina_Entidad(int n) {
    std::stringstream buffer;
    CEntidad aux_entidad;
    std::list<CEntidad>::iterator iterador;
        /*Si el archivo esta creado entra para realizarla baja*/
    if(this->ptr_Archivo != NULL) {
        /*Se ubica el iterador en la primera entidad de la lista*/
        iterador = this->lista_entidades.begin();
        if(iterador != this->lista_entidades.end()) {
                /*Avanza el iterador segun el index*/
            std::advance(iterador, n);
            aux_entidad = *iterador;

            iterador = this->lista_entidades.begin();
            while((iterador->dameDir_Siguiente() != aux_entidad.dameDir_Entidad())
                  && iterador != this->lista_entidades.end()) {
                iterador++;
            }
            /*Si el iterador es correcto modifica el archivo*/
            if(iterador->dameDir_Siguiente() == aux_entidad.dameDir_Entidad())
            {
                buffer << "Encontro el anterior " << iterador->dame_Nombre() << std::endl
                    ;
                iterador->ponDir_Siguiente(aux_entidad.dameDir_Siguiente());
                std::fseek(this->ptr_Archivo, iterador->dameDir_Entidad(), SEEK_SET);
                aux_entidad = *iterador;
                std::fwrite(&aux_entidad, sizeof(CEntidad), 1, this->ptr_Archivo);
                /*Elimino la entidad de la lista*/
                iterador = this->lista_entidades.begin();
                std::advance(iterador, n);
                this->lista_entidades.remove(*iterador);
            }
            else if(this->cabecera == aux_entidad.dameDir_Entidad()) {
                this->cabecera = aux_entidad.dameDir_Siguiente();
                std::fseek(this->ptr_Archivo, 0, SEEK_SET);
                std::fwrite(&this->cabecera, sizeof(long), 1, this->ptr_Archivo);
                buffer << "Es es el primero" << iterador->dame_Nombre() << std::endl;
                /*Elimino la entidad de la lista*/
                iterador = this->lista_entidades.begin();
                std::advance(iterador, n);
                this->lista_entidades.remove(*iterador);
            }
        }
        else {
            std::cout << "El diccionario esta vacio" << std::endl;
        }
    }
    return buffer.str();
}
```

5.3.5. Modificaciones

5.3.6. Codigo en C++ de Modificaciones

```
string CDiccionario::edita_Entidad(int index, char n[20])
{
    std::stringstream buffer;
    CEntidad aux_entidad;
    char aux_nombre[20];
    std::list<CEntidad>::iterator iterador;

    if(this->ptr_Archivo != NULL)
    {
        iterador = this->lista_entidades.begin();
        /*Se verifica si la lista no esta vacia comparado el interador con el final de la
        lista*/
        if(iterador != this->lista_entidades.end())
        {
            /*El metodo advance el iterador n veces si seleccione el elemento 2
             * advance recorre el iterador 2 veces
             */
            std::advance(iterador, index-1);
            std::strcpy(aux_nombre, iterador->dame_Nombre());
            iterador->pon_Nombre(n);
            aux_entidad = *iterador;
            std::fseek(this->ptr_Archivo, iterador->dameDir_Entidad(), SEEK_SET);
            std::fwrite(&aux_entidad, sizeof(CEntidad), 1, this->ptr_Archivo);
            buffer << "Se modifico la Entidad "
                   << aux_nombre << " -> " << aux_entidad.dame_Nombre()
                   << std::endl;
            this->lista_entidades.sort();
        }
        else
        {
            std::cout << "El diccionario esta vacio" << std::endl;
        }
    }
    return buffer.str();
}
```

Capítulo 6

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:

Figura 6.1: Estructura de un archivo secuencial

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.

Nombre	Dir_Entidad	Dir_Atriuto	Dir_Dato	Dir_siguiente

Tabla 6.1: Tabla de ejemplo

Bibliografía

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

[2] Leon Shklar, Rich Rosen, *Web Application Architecture: Principles, Protocols and Practices*, Auflage, 2009. [pp. 29-45]