

Programa 04: Índices y Serialización

06 de noviembre de 2022

Carlos Reyes Rico. 217458353

Estructura de Datos 2.



OBJETIVO

Realizar un programa con las siguientes características:

- 1) Debe crear una tabla de índices, donde use una función hash para calcular la posición donde se almacenara un registro en un archivo
 - 2) Debe crear una tabla de índices. Calcule lo que necesite para recuperar su registro.
 - 3) El archivo deberá almacenarse en forma binaria (si no lo logra guárdelo como registros)
 - 4) Debe de hacer uso de memoria dinámica (apuntadores)
 - 5) Crear una clase interfaz o en el main para el uso completo de todas sus operaciones
 - 6) Crear una clase que será su dato en cada nodo. (3 atributos privados, sus métodos get y set, así como los métodos que crean que sean de utilidad)
- a) A la plataforma debe de subir su programa y una reporte con impresiones de pantalla del funcionamiento de su programa.
- a.1) Debe de tener una explicación breve que hace su programa
 - a.2) Debe de explicar como organizo sus datos en la TDA
 - a.3) Debe de explicar la función Hash que haya implementado tanto para recuperar como para guardar sus registros.
 - a.4) El reporte debe de tener el siguiente nombre reporte01-PrimerApellido SegundoApellido Nombre.pdf
- b) El nombre del programa debe ser prog01-PrimerApellido SegundoApellido Nombre.cpp o comprimir todo el proyecto en un solo archivo esté nombre (.ZIP)

FUNCIONALIDAD

El programa realizado es un programa que hace uso de una TDA con su lista y clases a utilizar, lo que diferencia este programa de otros que hemos realizados es la implementación de tablas de hash las cuales dan un nuevo prospecto al programa, ya que, al hacer uso de una clave única para cada dato, podemos realizar una búsqueda mas precisa ya sea para eliminar un elemento o buscar alguno que se solicite.

Al funcionar la TDA con las tablas hash obtenemos un sistema de guardado más preciso y eficaz para cualquier búsqueda que se requiera o eliminación de datos.

ESTRUCTURACIÓN DE LA TDA

Para esta TDA usamos una clase de datos en los cuales se registraban los libros deseados por el usuario, siendo estos el autor, nombre del libro y su número de páginas cada uno con sus setters y getters, también sobrecargue los operadores ">>" y "<<", esto para evitar errores de almacenamiento en la lista, además del código único de este mismo para la tabla hash. Para el almacenamiento de la lista use una simplemente enlazada y así de esta forma cree mi TDA de la misma manera la uní con mi tabla hash.

Código .h TDA

```
class data
{
    string autor;
    string libro;
    string paginas;
    int llave;

public:
    data();
    void setAutor(const string &a);
    string getAutor();
    void setLibro(const string &l);
    string getLibro();
    void setPaginas(const string &p);
    string getPaginas();
```

```
void setLlave(int l);
```

```
int getLlave();
```

```
friend ostream &operator<<(ostream &out, const data &d)
```

```
{
```

```
    out << left;
```

```
    out << "|" << d.autor;
```

```
    out << "\\t|" << d.libro;
```

```
    out << "\\t|" << d.paginas;
```

```
    out << endl;
```

```
    return out;
```

```
}
```

```
friend istream &operator>>(istream &in, data &d)
```

```
{
```

```
    cout << "Ingresa el nombre del Autor: ";
```

```
    getline(cin, d.autor);
```

```
    cout << "Ingresa el nombre del libro: ";
```

```
    getline(cin, d.libro);
```

```
    cout << "Ingresa el numero de paginas: ";
```

```
    getline(cin, d.paginas);
```

```
    cout << endl;
```

```
    cout << "Ingresa el codigo: ";
```

```
    cin >> d.llave;
```

```
    cin.ignore();
```

```
    return in;
```

```
}
```

```
bool operator==(const data &d) const
```

```
{
```

```

        return llave == d.llave;
    }
};

```

TABLA HASH

Métodos hash

- Insertar

```

void Hash::insertar(int llave, data valor)
{
    int hashValor = hashFuncion(llave);
    auto &cell = tabla[hashValor];
    bool llave_existente = false;
    auto enumerador = cell.begin();

    do
    {
        if (!llave_existente)
        {
            cell.emplace_back(llave, valor);
            return;
        }
        else
        {
            llave_existente = true;
            enumerador++;
        }
    } while (enumerador != cell.end());
}

```

➤ Explicación

En el método insertar se reciben dos parámetros, uno es el código único y el otro los datos a insertar, dentro de la función, creamos variables a utilizar dentro de la ejecución, comenzamos verificando que el código no sea repetido, si no lo es ingresamos el dato correctamente en la tabla.

```

        .:Menu:.
1. Agregar Libro
2. Eliminar Libro
3. Buscar Libro
4. Mostrar Tabla (contenido)
5. Guardar Datos
6. Cargar Datos
0. Salir
R= 1
Ingresa el nombre del Autor: Alfonso G.V
Ingresa el nombre del libro: spartanos
Ingresa el numero de paginas: 125

Ingresa el codigo: 14
Libro insertado

```

- Eliminar

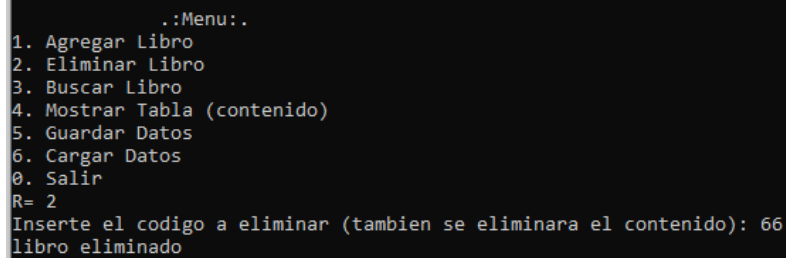
```
void Hash::eliminar(int llave)
{
    int hashValor = hashFuncion(llave);
    auto &cell = tabla[hashValor];
    auto enumerador = begin(cell);
    bool llave_existente = false;
    for (; enumerador != end(cell); enumerador++)
    {
        if (enumerador->first == llave)
        {
            llave_existente = true;
            enumerador = cell.erase(enumerador);
            cout << "libro eliminado" << endl;
            break;
        }
    }

    if (!llave_existente)
    {
        cout << "codigo no encontrado" << endl;
    }

    return;
}
```

- Explicación

En este método recibimos el código del dato a eliminar y con un for recorreremos todos nuestros códigos hasta dar con el que se insertó, en caso de no existir no realiza nada, pero si lo encuentra elimina todos los datos correspondientes a ese código.



```
.:Menu:.
1. Agregar Libro
2. Eliminar Libro
3. Buscar Libro
4. Mostrar Tabla (contenido)
5. Guardar Datos
6. Cargar Datos
0. Salir
R= 2
Inserte el codigo a eliminar (tambien se eliminara el contenido): 66
libro eliminado
```

- Buscar

```
void Hash::buscar(int llave)
{
```

```

int hashValor = hashFuncion(llave);
auto &cell = tabla[hashValor];
auto enumerador = begin(cell);
bool llave_existente = false;
for (; enumerador != end(cell); enumerador++)
{
    if (enumerador->first == llave)
    {
        llave_existente = true;
        cout << "libro encontrado" << endl;
        cout<<"Codigo \tAutor \tLibro\t \tPaginas \n";
        cout << enumerador->first<<"\t"<<enumerador->second << endl;
        break;
    }
}

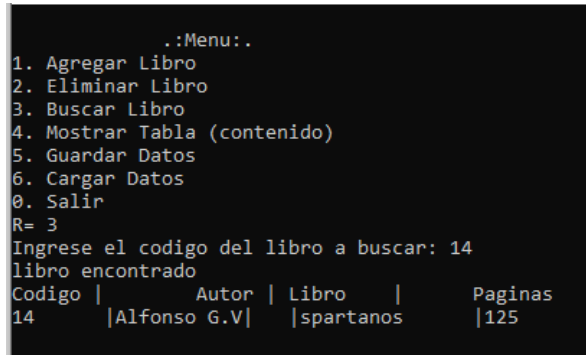
if (!llave_existente)
{
    cout << "Codigo no encontrado" << endl;
}

return;
}

```

➤ Explicación

En este método también hacemos uso del código, con un for recorreremos todos los datos hasta encontrar el código deseado a continuación mostramos los datos correspondientes a ese código.



```

.:Menu:.
1. Agregar Libro
2. Eliminar Libro
3. Buscar Libro
4. Mostrar Tabla (contenido)
5. Guardar Datos
6. Cargar Datos
0. Salir
R= 3
Ingrese el codigo del libro a buscar: 14
libro encontrado
Codigo |      Autor | Libro |      Paginas
14     |Alfonso G.V| |spartanos|125

```

- Guardar

```

void Hash::Guardar()
{
    ofstream archivo("file01.txt", ios::out);
    if (archivo.is_open())
    {

```

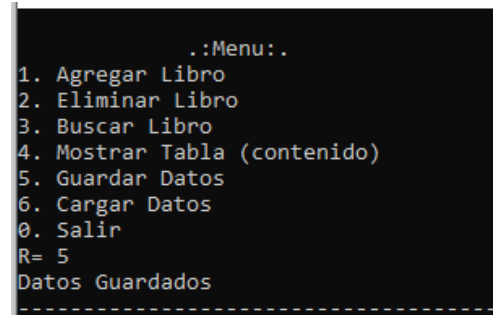
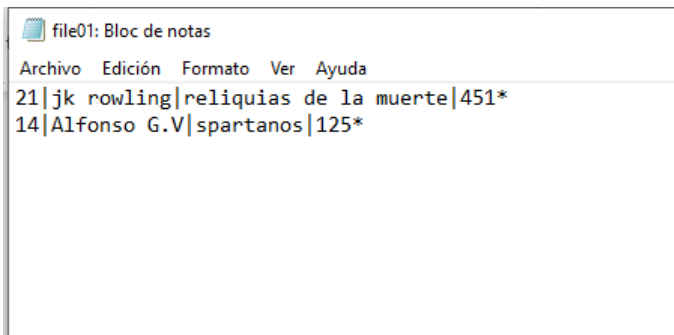
```

for (int i{}; i < consola; i++)
{
    for (auto it = tabla[i].begin(); it != tabla[i].end(); it++)
    {
        archivo << it->first << "|"
            << it->second.getAutor() << "|"
            << it->second.getLibro() << "|"
            << it->second.getPaginas() << "*"
            << endl;
    }
}
}
else
{
    cout << "No se pudo crear el archivo" << endl;
}
archivo.close();
}

```

➤ Explicación

Como en todo método guardar hacemos uso de la librería fstream, con su implementación de lectura ofstream creamos el txt en el que se almacenaran los datos y con un for vamos dato por dato leyéndolos en nuestro archivo, hacemos uso de separadores "|" y limitadores "*", para su posterior carga, cada dato es extraído de su getter directamente de la lista.



- Cargar

```

void Hash::Cargar()
{
    ifstream archivo("file01.txt", ios::out);
    if (archivo.is_open())
    {

```



```

    string provisional;
    int llave;
    data d;

    while (true)
    {
        getline(archivo, provisional, '|');
        if (archivo.eof())
        {
            break;
        }
        llave = stoi(provisional);
        d.setLlave(llave);
        getline(archivo, provisional, '|');
        d.setAutor(provisional);
        getline(archivo, provisional, '|');
        d.setLibro(provisional);
        getline(archivo, provisional, '*');
        d.setPaginas(provisional);

        insertar(d.getLlave(), d);
    }
}
else
{
    cout << "No se pudo acceder al archivo" << endl;
}
archivo.close();
}

```

➤ Explicación

Este método es similar al guardar, solamente sustituimos el ofstream por el ifstream, y en ves de un for usaremos un while para que continúe hasta que el archivo este vacío de texto, comenzamos la lectura y delimitamos cada valor con el separador y lo almacenamos en la lista con su setter correspondiente, esto lo ejecutará hasta llegar al limitador y pasará al siguiente dato.

```
"J:\UDG Kevin\Semestre 3\Est. Datos 2\Programas"

.:Menu:
1. Agregar Libro
2. Eliminar Libro
3. Buscar Libro
4. Mostrar Tabla (contenido)
5. Guardar Datos
6. Cargar Datos
0. Salir
R= 4
Tabla:
-----
```

```
.:Menu:
1. Agregar Libro
2. Eliminar Libro
3. Buscar Libro
4. Mostrar Tabla (contenido)
5. Guardar Datos
6. Cargar Datos
0. Salir
R= 6
Datos Cargados
-----
```

```
.:Menu:
1. Agregar Libro
2. Eliminar Libro
3. Buscar Libro
4. Mostrar Tabla (contenido)
5. Guardar Datos
6. Cargar Datos
0. Salir
R= 4
Tabla:
Codigo | Autor | Libro | Paginas
21 | jk rowling | reliquias de la muerte | 451

Codigo | Autor | Libro | Paginas
14 | Alfonso G.V | spartanos | 125
```