Método Disperso o Hash

Organización de Archivos

1do Angely Mendez
Escuela de Informática
Universidad Nacional de Trujillo
Trujillo, Perú
t052701020@unitru.edu.pe

2ero Ciara Mendez
Escuela de Informática
Universidad Nacional de Trujillo
Trujillo, Perú
t022700920@unitru.edu.pe

***Resumen*—Este documento es una investigación e implementación en lenguaje C++ sobre el Método Disperso o Hash que permite el acceso directo a un elemento de una secuencia, indicando la posición que ocupan. Se suelen utilizar para implementar cualquier tipo de diccionario, por ejemplo, la tabla de símbolos de un compilador necesaria para generar el código ejecutable. La potencia del Método Disperso o Hash radica en la búsqueda de elementos, conociendo el campo clave se puede obtener directamente la posición que ocupa y, por consiguiente, la información asociada a dicha clave. En este articulo se implementó los cuatro métodos Hash para la transformación de la clave y funciones que permiten la creación de dos registros, con extensiones, $*.txt$. y $*.col$. en los que se puede escribir, mostrar, buscar, modificar y eliminar registros.**

***Palabras claves:*—método disperso, hash, hashing, C++.**

I. Introducción

A medida que la cantidad de muestras de malware ha incrementado, mantener una base de datos de firmas se ha convertido en una tarea que simplemente no escala. Según el Grupo tecnológico ATICO34 [1], se ha estimado que cada día aparecen más de 500.000 muestras de malware únicas. Es muy probable que esto se deba en gran parte a que los autores de malware se dan cuenta de que pueden engañar a los motores antivirus que dependen de hashes para que no reconozcan una muestra con mucha facilidad. Por lo que el Método disperso o Hash, es útil porque nos brinda un identificador único para algún archivo, el cual puede usar la información de varias formas, cuya clave es requerida por ejemplo, en algunas soluciones de antivirus que se basan completamente en valores hash para determinar si un archivo es malicioso o no, sin examinar el contenido o el comportamiento del archivo, ahí radica la importancia de conocer respecto a mencionado método y su implementación.

Entonces en este informe se presenta información al respecto, el cual está organizado de la siguiente manera: en primer lugar, se explican los conceptos teóricos: la etimología, definiciones y los conceptos de Hashing, luego se da énfasis en la implementación de las funciones y métodos Hash en el lenguaje C++, para finalizar las conclusiones más relevantes.

II. Método Disperso o Hash

II-A. Etimología

El término **hash** proviene, aparentemente, de la analogía con el significado estándar (en inglés) de dicha palabra en el mundo real: picar y mezclar. Donald Knuth (Ver Figura 1)cree que H. P. Luhn, empleado de IBM, fue el primero en utilizar el concepto en un memorándum fechado en enero de 1953. Su utilización masiva no fue hasta después de 10 años.

En el algoritmo $SHA1$, por ejemplo, el conjunto de partida de

la función es dividido en palabras que son mezcladas entre sí utilizando funciones matemáticas seleccionadas especialmente. Se hace que el rango de valores que puede devolver la función sea de longitud fija: 160 bits utilizando la adición modular.

Figura 1. Donald Ervin Knut, estadounidense experto en ciencias de la computación y matemático.

II-B. Definición

Es un método que permite encontrar, con el mínimo esfuerzo, una clave dada dentro de un conjunto de elementos. También se denomina método de ”transformación de claves”, ”direccionamiento calculado”,”direccionamiento asociativo.º de ”dispersión”. En la técnica de Hashing se aplican cálculos o transformaciones aritméticas sobre la clave para producir directamente una dirección en una tabla o archivo de acceso directo.

De acuerdo a [5], define que los algoritmos hash son métodos de búsqueda, que proporcionan una longitud de búsqueda pequeña y una flexibilidad superior a la de otros métodos, como puede ser el método de búsqueda binaria que requiere que los elementos de la matriz estén ordenados.

Según [4], las tablas de dispersión o, simplemente, tablas hash son estructuras de datos que se usan en aplicaciones que manejan una secuencia de elementos. De tal forma que cada elemento tiene asociado un valor clave, que es un número entero positivo perteneciente a un rango de valores, relativamente pequeño. En estas organizaciones cada uno de los elementos ha de tener una clave que identifica de manera unívoca al elemento.

De acuerdo a [7], Las tablas de hashing son una estructura de datos tipo diccionario que nos permite insertar, eliminar y buscar elementos a partir de una clave. Esta estructura se puede ver como un conjunto de entradas, donde cada una de ellas tiene asociada una clave única. Esto significa que una clave identifica unívocamente a una entrada en la tabla de hash.

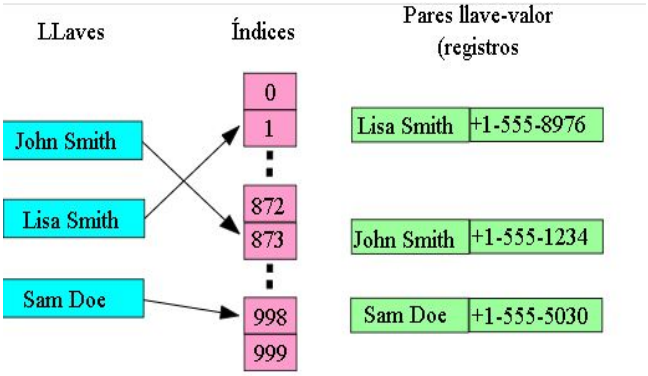

Figura 2. Ejemplo de Tabla Hash.

En la Figura 2, se puede observar elementos de un Tabla Hash: llaves, índices y finalmente sus registros.

Según el Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante en España,[6] define al Método Hashing como **Ficheros de direccionamiento calculado (hashing)**, dado que la condición de búsqueda se realiza sobre un campo llamado *direccionamiento calculado*.

Una función h llamada función de direccionamiento calculado se aplica sobre el campo de direccionamiento calculado para obtener el bloque donde está el registro. Y los datos se almacenan de forma dispersa.

Por ejemplo: $h(K) = KmodM$, donde:
M : número de bloques a usar
K : valor del campo de direccionamiento calculado.
Y se supone que K es un valor de DNI = ‘21987239’, además el número total de bloques M=20
$h(K)$ = 19 (el registro se encuentra en el bloque número 19)

Y una vez conocido el número de bloque, un fichero de punteros apunta al bloque de disco, ver Figura 3.

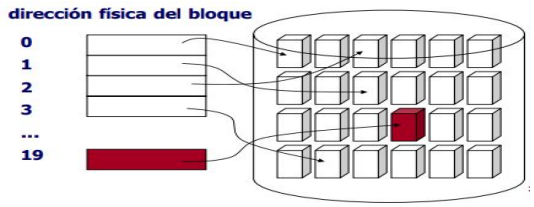

Figura 3. Dirección Física del bloque.

II-C. Conceptos de Hashing

1. **Clave:**

   Según [3], la clave contiene el valor que permite ubicar mediante la función Hash la posición en el registro de ese elemento. Normalmente la clave es el campo que identifica en forma única la información. Por ejemplo: Documento de Identidad de una persona, Código de un empleado.

2. **Función Hash:**

   De acuerdo a [2], una función Hash se puede definir como una transformación de clave a una dirección. Al aplicar una función Hash a una clave se obtiene el número de cubeta en la cual se puede encontrar el registro con dicha clave.

   Esta función consiste en asignar el índice a cada elemento mediante una transformación del elemento, el número obtenido es el índice del elemento. Hay diferentes funciones para transformar el elemento y son las siguientes:

   - **Método de la División o Aritmetica Modular**

     Según [8], este método convierte la clave a un entero, se divide por el tamaño del rango del índice y toma el resto como resultado.

     Esta transformación implementada en lenguaje c++ quedaría de la siguiente manera:

     En la Figura 4, se observa que la función $fhash$, representa al método de la división, que tiene un parámetro de tipo char **nom de longitud 20**, aquí recibe el array a calcular. Lo que realiza esta función dentro de su cuerpo es inicializar una variable suma en 0 que almacenará el valor final de la suma de todas las letras del arreglo, basándose en el Código ASCII.

```
int fhash (char nom[20])
{
    int suma = 0;

    for (int i = 0; i < strlen (nom); i++)
    {
    //sumamos cada caracter de la cadena
    //ingresada
    suma += nom[i];
    }

    return suma % 10 + 1;
}
```

Figura 4. Implementación Método de la División.

El cual será recorrido con for hasta la longitud del array nom **(strlen(nom))**, finalmente retorna el resto de la suma con 10 + 1, obteniéndose así el valor de la clave.

- **Mitad del cuadrado**

  Según [8], consiste en elevar al cuadrado la clave y coger las cifras centrales. Este método también presenta problemas de colisión. En caso de que la cifra resultante tenga un número de dígitos impar, se toma el valor número y el anterior.

  Esta transformación implementada en lenguaje c++ quedaría de la siguiente manera:

  En la Figura 5, se observa que la función $fhashmitadcuadrado$, representa al método de mitad del cuadrado, que tiene un parámetro de tipo char **nom de longitud 20**, aquí recibe el array a calcular. Dentro del cuerpo de esta función, se inicializa la variables suma, cuadrado, contador y cifra. El for está encargado de recorrer todos los caracteres del array ingresado, e ir sumando acumulativamente en la variable suma, según Código ASCII.

  Luego se obtiene el cuadrado de dicha variable, que se almacena en la variable cuadrado. Posteriormente, el primer while calcula la cantidad de dígitos de la variable cuadrado. Y el segundo while, separa todos los dígitos y los almacena en array tipo int de nombre numero.

```
int fhash_mitad_cuadrado (char nom[20])
{
    int suma = 0, cuadrado=0, contador=1, cifra=10;

    for (int i = 0; i < strlen (nom); i++)
    {
     //se obtiene la suma de todas las letras
    suma += nom[i];
    }

    cuadrado = suma*suma;

    while(cifra <= cuadrado)
    {
        //cantidad de cifras del cuadrado
        contador++;
        cifra = cifra *10;
    }

    int i=0, numero[contador-1];

    //separa dígito por dígito el cuadrado
    while(cuadrado>0)
    {
    //numero, almacena cada dígito en el arreglo de
    //atrás hacia adelante
        numero[i]= cuadrado%10;
        cuadrado = cuadrado/10;
        i++;
    }

    //Aquí se da el método mitad_cuadrado,
    //calcula los 2 dígitos centrales
    return numero[(contador/2)]*10 + numero[(contador/2)-1];
}
```

Figura 5. Implementación Mitad del Cuadrado.

Finalmente retorna los dos dígitos centrales del cuadrado.

- **Truncamiento**

  Según [8], consiste en ignorar parte del número y utilizar los elementos restantes como índice. También se produce colisión. Por ejemplo, si un número de 7 cifras se debe ordenar en un arreglo de elementos, se pueden tomar el segundo, el cuarto y el sexto para formar un nuevo número.

  Esta transformación implementada en lenguaje c++ quedaría de la siguiente manera:

  En la Figura 6, se observa que la función $fhashtruncamiento$, representa al método del truncamiento, que tiene un parámetro de tipo char **nom de longitud 20**, aquí recibe el array a calcular. Dentro del cuerpo de esta función, se inicializa la variables suma, cuadrado, contador y cifra. El for está encargado de recorrer todos los caracteres del array ingresado, e ir sumando acumulativamente en la variable suma, según Código ASCII. Aquí también se realiza como en el método anterior, para el primer while calcula la cantidad de dígitos de la variable cuadrado.

```
int fhash_truncamiento (char nom[20])
{
    int suma = 0, cuadrado=0, contador=1, cifra=10, dos=0;

    for (int i = 0; i < strlen (nom); i++)
    {
    //se obtiene la suma de todas las letras
    suma += nom[i];
    }

    cuadrado = suma*suma;

    while(cifra <= cuadrado)
    {
    //cantidad de cifras del cuadrado
        contador++;
        cifra = cifra *10;
    }

    int i=0, numero[contador-1];

    //separa dígito por dígito el cuadrado
    while(cuadrado>0)
    {
    //numero, almacena cada dígito en el arreglo de
    //atrás hacia adelante
        numero[i]= cuadrado%10;
        cuadrado = cuadrado/10;
        i++;
    }

    int t= 1;
    //Aquí se da el método truncamiento
    //calcula los dígitos 2, 4 y 6
    for (int i=0; i<contador; i=i+2)
    {
    //dos almacena el número final de 3 cifras obtenido
        dos= dos + (numero[i]* t) ;
        t = t*10;
    }

    return dos;
}
```

Figura 6. Implementación de Truncamiento.

Y el segundo while, separa todos los dígitos y los almacena en array tipo int de nombre numero.

Adicionalmente, se agrega una estructura for que consiste en extraer los dígitos $2, 4$ y 6, y dentro de la variable dos almacena el índice final que será posteriormente retornado.

- **Plegamiento**

  Según [8], consiste en dividir el número en diferentes partes, y operar con ellas (normalmente con suma o multiplicación). También se produce colisión. Por ejemplo, si dividimos el número de 7 cifras en 2, 2 y 3 cifras y se suman, dará otro número de tres cifras (y si no, se toman las tres últimas cifras). Esta transformación implementada en lenguaje c++ quedaría de la siguiente manera:

  En la Figura 7, se se observa que la función $fhashplegamiento$, representa al método de la

división, que tiene un parámetro de tipo char **nom de longitud 20**, aquí recibe el array a calcular. Dentro del cuerpo de esta función, se inicializa la variables suma, cuadrado, contador y cifra. El for está encargado de recorrer todos los caracteres del array ingresado, e ir sumando acumulativamente en la variable suma, según Código ASCII.

```
int fhash_plegamiento (char nom[20])
{
    int suma = 0, cuadrado=0, contador=1, cifra=10;

    int dos=0;

    for (int i = 0; i < strlen (nom); i++)
    {
     //se obtiene la suma de todas las letras
    suma += nom[i];
    }

    cuadrado = suma*suma;

    while(cifra <= cuadrado)
    {
    //cantidad de cifras del cuadrado
        contador++;
        cifra = cifra *10;
    }

    int i=0, numero[contador-1];

    //separa digito por digito el cuadrado
    while(cuadrado>0)
    {
    //numero, almacena cada digito en el arreglo de
    //atrás hacia adelante
        numero[i]= cuadrado%10;
        cuadrado = cuadrado/10;
        i++;
    }

    int t= 10;
    //Aqui se da el método plegamiento
    for (int i=contador-1; i>=0; i=i-2)
    {
    //dos va sumando de dos en dos los dígitos del número
        dos= dos + ((numero[i]* t) + numero[i-1]);
    }

    return dos;
}
```

Figura 7. Implementación de Plegamiento.

Aquí también se realiza como en los dos métodos anteriores, para el primer while calcula la cantidad de dígitos de la variable cuadrado. Y el segundo while, separa todos los dígitos y los almacena en array tipo int de nombre numero.

Asimismo, se agrega una estructura for que consiste en extraer de dos en dos los dígitos, en ir realizando sumas acumulativas y dentro de la variable dos almacena el índice final que como termino de la función es retornado.

III. Implementación de las Funciones para el Método Disperso o Hash en C++

Para la implementación de este método hemos considerado realizar un registro de datos personales de una persona, así que los siguientes códigos se basan en ello.

1. **Estructura Encabezado:**

   Para definir el encabezado, ver Figura 8, se usó **struct** para poder almacenar varios campos, en este caso **NRS** que es la variable que almacena el número total de registros y **URE** que es la variable que almacena el último registro eliminado, además de los punteros **ed** y **ec** para el archivo dispersión y archivo colisión respectivamente.

```
struct encabezado
{
    int NRS; //número total de registros
    int URE;
} ed, ec; // encabezado de dispersion(ed) y colisiones(ec)
```

Figura 8. Código de Estructura Encabezado.

2. **Estructura Registro:**

   Para definir el registro, obsérvese en la Figura 9, se usó **struct** para poder almacenar varios campos, en este caso los datos personales de una persona y usamos: **NR** que es la variable que almacena el número de registro, el campo **dni** que es la variable que almacena el número de dni de una persona, el campo **nombre** que es la variable que almacena el nombre, el campo **apellido** que es la variable que almacena el apellido, el campo **email** que es la variable que almacena el correo electrónico, el campo **celular** que es la variable que almacena el número de celular, el campo **SR** que es la variable que almacena el siguiente registro, el campo **ARE** que es la variable que almacena el anterior registro eliminado y además de los punteros **r** y **s** para el archivo dispersión y archivo colisión respectivamente.

```
struct registro
{
    int NR; //número de registro
    char dni [9];
    char nombre[20];
    char apellido [20];
    char email [30];
    char celular [10];
    int SR; //siguiente registro
    int ARE;//anterior registro eliminado
} r, s;
```

Figura 9. Código de Estructura Encabezado.

3. **Escribir:**

   En la función escribir, obsérvese la primera parte en la Figura 10, inicialmente se abren ambos archivos, el archivo dispersión y archivo colisión, es decir para ser escritos; además se inicializan las variables ed y ec en sus campo NRS, que representa numero total de registros existentes en los archivos; y actualiza escribiendo en el encabezado.

   Dentro de un do while, se solicita ingresar al usuario la información de los campos para cada registro. Luego actualiza el campo SR con -1, que representa que no hay siguiente. También calcula el valor para el campo NR, obteniendo con la función fhash y enviándole como parámetro, el campo nombre ingresado anteriormente.

   Para el condicional, evalúa si no hay colisión, graba de forma dispersa, basándose en el índice hash, en el archivo $*.txt$. Aquí también se actualiza de la variable ed el campo NRS, incrementándose en 1; posiciona y escribe en el registro.

```
void escribir ()
{
    char rpta;  int sgte;

    if ((fdd = fopen ("persona.txt", "w+t")) == NULL){
    cout << "-No se pudo crear persona.txt" << endl; return; }

    if ((fdc = fopen ("persona.col", "w+t")) == NULL) {
    cout << "-No se pudo crear persona.col" << endl; return; }

    //Inicializacion de variables
    ed.NRS = 0; ec.NRS = 0;
    fwrite (&ed, le, 1, fdd);  fwrite (&ec, le, 1, fdc);

    do{
        cin.ignore();
        cout << "-DNI: " << endl;
        cin.getline(r.dni,9);

        cout << "-Nombre: " << endl;
        cin.getline(r.nombre,20);

        cout << "-Apellido: " << endl;
        cin.getline(r.apellido,20);

        cout << "-Email: " << endl;
        cin.getline(r.email,30);

        cout << "Celular:" << endl;
        cin.getline(r.celular,10);

        r.SR = -1;
        r.NR = fhash (r.nombre);//por el nombre posicionamos
        pos = (r.NR - 1) * lr + le;
        fseek (fdd, pos, 0);
        fread (&s, lr, 1, fdd);

        if (strcmp (s.nombre, "") == 0)
        {   //No hay colisión, graba en *.txt de forma dispersa
            ed.NRS++;
            fseek (fdd, pos, 0);
            fwrite (&r, lr, 1, fdd);
        }
```

Figura 10. Código de Función Escribir - Parte 1.

En la segunda parte de esta función, obsérvese Figura 11, se encuentra el else del if anterior, aquí se considera cuando si existe colisión, por lo que se escribe o graba en el archivo $*.col$, de forma secuencial. Actualizando de igual manera el campo NRS de la variable ec, que pertenece a este archivo.

```
        else
        {
            //Si hay colisión, graba en *.col de forma secuencial
            r.NR = ++ec.NRS;
            pos = (r.NR - 1) * lr + le;

            fseek (fdc, pos, 0);
            fwrite (&r, lr, 1, fdc);

            //enlazar un registro anterior para que apunte al nuevo
            if (s.SR == -1)
            {   //caso 1: el anterior esto?= en *.txt
                s.SR = r.NR;
                pos = (s.NR - 1) * lr + le;

                fseek (fdd, pos, 0);
                fwrite (&s, lr, 1, fdd);
            }
            else
            {//caso2: el anterior esto?= en *.col
            //recorrer la lista enlazada hasta el final en *.col
                sgte = s.SR;

                while (sgte != -1)
                {
                    pos = (sgte - 1) * lr + le;

                    fseek (fdc, pos, 0);
                    fread (&s, lr, 1, fdc);

                    sgte = s.SR;
                }

                s.SR = r.NR;
                pos = (s.NR - 1) * lr + le;

                fseek (fdc, pos, 0);
                fwrite (&s, lr, 1, fdc);
            }
        }
        cout << "-¿Desea ingresar más registros?" << endl;
        cin >> rpta;

    } while (rpta == 'S' || rpta == 's');

    fseek (fdd, 0, 0);  fseek (fdc, 0, 0);
    fwrite (&ed, le, 1, fdd);  fwrite (&ec, le, 1, fdc);
    fclose (fdd);  fclose (fdc);
}
```

Figura 11. Código de Función Escribir - Parte 2.

Luego con un if else, se enlaza un registro anterior para que apunte al nuevo. Evalúa de cumplirse la igualdad entre -1 y el campo SR de s, para el caso 1, si el anterior estará en $*.txt$, de ser cierto ingresa y actualiza el campo SR, posiciona y escribe. Para el caso 2, el anterior esta en

$*.col$, por lo que actualiza el sgte y con un while recorre la lista enlazada hasta el final, aquí también se posiciona y escribe. Para el do while mencionado inicialmente, se dará si rpta es S o s, de no cumplirse se termina. Finalmente se actualiza los encabezados y cierra de los dos archivos.

4. **Mostrar:**

   Para definir la función mostrar, ver Figura 12, primero, abrimos ambos archivos, es decir, el archivo dispersión y archivo colisión respectivamente, luego, se lee el encabezado para mostrar datos convenientemente, también a través de **fread** se lee los datos a mostrar en pantalla y se hace lo mismo con el archivo colisión, finalmente cerramos ambos archivos.

```
void mostrar()
{
    if ((fdd = fopen ("persona.txt", "rt")) == NULL)
    {
        cout << "-No se pudo abrir persona.txt" << endl;
        return;
    }

    if ((fdc = fopen ("persona.col", "rt")) == NULL)
    {
        cout << "-No se pudo abrir persona.col" << endl;
        return;
    }

    fread (&ed, le, 1, fdd);
    cout << "\n---------------------------------------";
    cout << "\n Archivo Disperso. NRS: " << ed.NRS << endl;
    cout << "NRS\tDNI\t\tNombre\t\tApellido\t\tEmail\t\tCelular\t\tSR\n";

    while (fread (&s, lr, 1, fdd))
    {
        cout <<s.NR<<"\t"<<s.dni<<"\t"<<s.nombre<<"\t\t"<<s.apellido<<"\t\t\t";
        cout <<s.email<<"\t\t"<<s.celular<<"\t"<<s.SR<< endl;
    }
    cout << "\n---------------------------------------";
    fread (&ec, le, 1, fdc);
    cout << "\n---------------------------------------";
    cout << "\nArchivo Colision. NRS:" << ec.NRS << endl;
    cout << "NRS\tDNI\t\tNombre\t\tApellido\t\tEmail\t\tCelular\t\tSR\n";;
    cout << "\n---------------------------------------";
    while (fread (&s, lr, 1, fdc))
    {
        cout <<s.NR<<"\t"<<s.dni<<"\t"<<s.nombre<<"\t\t"<<s.apellido<<"\t\t\t";
        cout <<s.email<<"\t\t"<<s.celular<<"\t"<<s.SR<< endl;
    }

    fclose (fdd);
    fclose (fdc);
}
```

Figura 12. Código de Función Mostrar.

5. **Buscar:**

   En la función buscar, obsérvese en la Figura 13, inicialmente se abren ambos archivos, el archivo dispersión y archivo colisión respectivamente, es decir para que puedan ser leídos y escritos; luego solicita digitar al usuario el nombre que desea buscar. Después, se calcula con la función $fhash$, el índice, calcula pos y lee.

   Para la **búsqueda en el archivo dispersión**, se realiza dos condicionales, el primer if evalua si existe registro con ese hash, de no ser cierto retorna con un mensaje. Y en el segundo if realiza las comparaciones con cada registro del archivo, cuando lo encuentra, muestra la información de sus campos y retorna. De no cumplirse ninguno de los dos if muestra que no existe en el archivo. Por otro lado, para la **búsqueda en el archivo de colisión**, se inicializa la variable $sgte$ con el campo SR de s, y través del while como primer paso, se evalua que mientras sea diferente de -1 y va actualizando el sgte, entrará a calcular pos, posiciona y leer; con **fseek y fread**. Con un condicional, realiza las comparaciones con cada nombre del registro, si coincide con uno entonces muestra la información de sus campos y retorna. De no cumplirse en el if y no encontrarse el nombre, muestra un mensaje que no existe en el archivo.

```
void buscar()
{
    cin.ignore();
    cout << "- Ingrese nombre a buscar:" << endl;
    cin.getline(nom,20);
    NR = fhash (nom);
    cout << "Hash:" << NR << endl;
    pos = (NR - 1) * lr + le;
    fseek (fdd, pos, 0);

    fread (&s, lr, 1, fdd);
    if (strcmp (s.nombre, "") == 0)
    {
        cout << "-No existe registro con ese Hash." << endl;
        return;
    }
    if (strcmp (s.nombre, nom) == 0)
    {
        cout << "NRS\tDNI\t\tNombre\t\tApellido\t\tEmail\t\tCelular\t\tSR\n";
        cout <<s.NR<<"\t"<<s.dni<<"\t"<<s.nombre<<"\t\t"<<s.apellido;
        cout<<"\t\t\t"<<s.email<<"\t\t"<<s.celular<<"\t"<<s.SR<< endl;
        return;
    }
    cout << "-No existe en el archivo *.txt" << endl;
    sgte = s.SR;
    while (sgte != -1)
    {
        pos = (sgte - 1) * lr + le;
        fseek (fdc, pos, 0);
        fread (&s, lr, 1, fdc);
        if (strcmp (nom, s.nombre) == 0)
        {
            cout <<s.NR<<"\t"<<s.dni<<"\t"<<s.nombre<<"\t\t"<<s.apellido;
            cout<<"\t\t\t"<<s.email<<"\t\t"<<s.celular<<"\t"<<s.SR<< endl;
            return;
        }
        sgte = s.SR;
    }
    cout << "-No existe en *.col" << endl;
    return;
}
```

Figura 13. Código de Función Buscar.

6. **Insertar:**

En esta función, ver Figura 14, es de manera similar que la función escribir, solo que aquí se **evita la inicialización del campo NRS** en las variables ed y ec; porque esos son campos globales y puede existir la posibilidad que ya existan registros, por lo que si las inicializamos nuestro programa no funcionará y no almacenara los registros de manera adecuada, debido a que en ese campo se inicializa en 0.

Luego todo es exactamente igual que la función escribir.

```
void insertar ()
{
  char rpta;
  int sgte;
  cout << "| INSERTAR |" << endl;
  //ed.NRS = 0;  ec.NRS = 0;
  fwrite (&ed, le, 1, fdd);
  fwrite (&ec, le, 1, fdc);

  do
    {
      //funcion escribir

    }
  while (rpta == 'S' || rpta == 's');

  fseek (fdd, 0, 0);
  fseek (fdc, 0, 0);
  fwrite (&ed, le, 1, fdd);
  fwrite (&ec, le, 1, fdc);
  fclose (fdd);
  fclose (fdc);
}
```

Figura 14. Código de Función Insertar.

7. **Modificar:**

En la función modificar, ver Figura 15, inicialmente se abre el archivo para ser modificado $r + t$, es decir para poder leer y escribir, **persona.txt**, luego con **fread**, se lee el encabezado y se solicita digitar al usuario el nombre que desea modificar.

Después, a través del while como primer paso, se **busca en los registros** realizándose comparaciones, de encontrarse, se realiza el segundo paso donde **solicita digitar el nuevo Email**, actualizando dicha información a través de posicionar y escribir, con **fseek y fwrite** en el registro, por otro lado, de no encontrarse el nombre muestra un mensaje. Finalmente actualiza el encabezado, y cierra el archivo.

```
void modificar()
{
    char nom[20];  int NR=0, pos1=0;
    bool bandera = false;

    cout << "| MODIFICAR EMAIL |"<<endl;

    if ((fdd = fopen ("persona.txt", "r+t")) == NULL)
    {
        cout << "-No se pudo abrir persona.txt" << endl;
        return;
    }

    fread(&ed,le,1,fdd);
    cin.ignore();
    cout << "-Ingrese nombre a modificar:" << endl;
    cin.getline(nom,20);

    NR = fhash (nom);
    cout << "Hash:" << NR << endl;

    while (fread(&r,lr,1,fdd))
    {
        if (strcmp (r.nombre,nom) == 0)
        {
            cout << "-Digite el nuevo Email:" << endl;
            cin.getline(r.email,20);

            pos1=(r.NR-1)*lr + le;
            cout<<endl<<pos;
            fseek(fdd,pos1,0);
            fwrite(&r,lr,1,fdd);
        }
    }

    fseek(fdd,0,0);
    fwrite(&ed,le,1,fdd); //actualiza cabecera
    fclose(fdd);
}
```

Figura 15. Código de Función Modificar.

8. **Eliminar:**

```
void eliminar()
{
    int pos; bool band,flag=false;
    char nom_eliminado[20];

    cout << "| ELIMINAR |"<<endl;

    if((fdd=fopen("persona.txt","r+t"))==NULL) //abre archivo
    {
        cout<<"-No se pudo abrir el archivo"<<endl;
        return;
    }

    fread(&ed,le,1,fdd); //lee cabecera
    cin.ignore();
    cout<<"-Digite el nombre a eliminar: "<<endl;
    cin.getline(nom_eliminado,20); //ingresa nombre a eliminar

    //1.Buscar el registro a eliminar
    while(fread(&r,lr,1,fdd))
    {
        if(strcmp(nom_eliminado,r.nombre)==0)
        {
            flag=true;
            //2. actualizar URE/ARE
            r.ARE = ed.URE;
            ed.URE = r.NR;
            ed.NRS--;
            cout<<"¡El Registro fue eliminado!"<<endl;
            break;
        }
    }

    if(flag==false){
        cout<<"¡El nombre no existe!"<<endl;
        return;
    }

    pos=(r.NR-1)*lr + le;   fseek(fdd,pos,0); fwrite(&r,lr,1,fdd);

    fseek(fdd,0,0);
    fwrite(&ed,le,1,fdd); //actualiza cabecera
    fclose(fdd);
}
```

Figura 16. Código de Función Eliminar.

Para la función eliminar, obsérvese en la Figura 16, inicialmente se abre el archivo **persona.txt**, luego con fread, se lee el encabezado y se solicita digitar al usuario el nombre que desea eliminar.

Después, a través del while como primer paso, se **busca en los registros** realizándose comparaciones, de encontrarse, se realiza el segundo paso donde **actualiza los valores URE/ARE**, asignándole a URE, NR, que es el índice hash, el URE, representa cuál es último elemento eliminado y al campo ARE, -1, que representa que está eliminado; por otro lado, de no encontrarse el nombre muestra un mensaje. Finalmente posiciona y escribe con **fseek y fwrite** tanto en el registro y actualiza el encabezado, y cierra el archivo.

IV. Conclusión

Este informe presentó información relevante acerca del Método Disperso o Hash, se ha explicado los conceptos teóricos relacionados con este, cabe precisar que si el valor hash obtenido de los caracteres de la información original sobre la que se aplica los algoritmos o métodos hash, varía cualquier de sus caracteres, varía el valor final. Además de permitir conocer sobre la implementaciones de los métodos hash, con sus respectivas explicaciones, además de otras funciones que permiten la creación de dos registros, con extensiones, uno $*.txt.$ y otro $*.col.$ en los que se puede escribir, mostrar, buscar, modificar y eliminar registros.

Referencias

[1] G. Atico34. ¿qué es y para qué sirve un hash? — grupo atico34. `https://bit.ly/3fIKFkP`, Octubre 2020. (Accedido en 01/21/2022).

[2] O. Cairó and S. Guardati. *Estructuras de datos*. MC GRAW HILL INTERAMERICANA, 2006.

[3] A. G. Carrillo and J. Valdivia. *Abstracción y Estructuras de Datos en C++*. Delta Publicaciones, 2006.

[4] L. Joyanes, L. Joyanes, and I. Zahonero. *Estructuras de datos en C++*. McGraw-Hill/Interamericana de España, 2007.

[5] F. J. C. Sierra. *Curso de programación con C*. Ra-Ma, 1991.

[6] A. Suárez Cueto. Organización física de las bases de datos. *Universidad de Alicante, España - Bases de Datos I*, 2017.

[7] A. WEISS. Mark “estructura de datos en java”, 1998.

[8] Wikipedia. Tabla hash — wikipedia, la enciclopedia libre, 2022. (Internet descargado 21-enero-2022).