Departamento de Ciencias Aplicadas

Ingeniería en Computación.

Organización de archivos.

Unidad de competencia III: "Reconocer y manejar archivos directos"

Presenta:
M. en C. C. J. Jair Vázquez Palma.

Organización de Archivos

Objetivos de la Unidad de la Unidad de Aprendizaje

- Analizar y concretar soluciones en el área del procesamiento automatizado de la información.
- Identificar problemas relacionados con almacenamiento, procesamiento, acceso y aplicaciones de información.
- Analizar soluciones del entorno y problemas propios de ser tratados mediante sistemas computacionales

Unidad III: Reconocer y manejar archivos directos.

Contenido:

- Características, Ventajas, Desventajas.
- Funciones Hash.
- Estructura de la organización de los archivos directos.
- Operaciones (altas, bajas, consultas y modificaciones).

Objetivos de la Unidad de Competencia III

- Analizar las estructuras y características de los archivos directos.
- Manejar o utilizar programas que permitan el manejo de datos dentro de los archivos con directo.
- Entender las distintas operaciones que se pueden realizar con archivos de acceso directo.

CARACTERISTICAS. VENTAJAS, DESVENTAJAS

CARACTERISTICAS, ACCESO DIRECTO.

- Un archivo está organizado en modo directo cuando el orden físico no se corresponde con el orden lógico. Los datos se sitúan en el archivo y se accede a ellos directa aleatoriamente mediante su posición, es decir, el lugar relativo que ocupan.

- Esta organización tiene la ventaja de que se pueden leer y escribir registros en cualquier orden y posición. Son muy rápidos de acceso a la información que contienen.

CARACTERISTICAS, VENTAJAS, DESVENTAJAS DEL ACCESO DIRECTO.

- La organización directa tienen el inconveniente de que se necesita programar la relación existente entre el contenido de un registro y la posición que ocupa.

- El acceso a los registros en modo directo implica la posible existencia de huecos libres dentro del soporte, y por consecuencia pueden existir huecos libres entre registros.

- La correspondencia entre clave y dirección debe poder ser programada y la determinación de la relación entre el registro y su posición física se obtiene mediante una fórmula.

CARACTERISTICAS, VENTAJAS, DESVENTAJAS DEL ACCESO DIRECTO.

Las condiciones para que un archivo sea de organización directa son:

- Almacenado en un soporte direccionable.

- Los registros deben contener un campo específico denominado clave que identifica cada registro de modo único; es decir, dos registros distintos no pueden tener un mismo valor de clave.

- Existencia de una correspondencia entre los posibles valores de la clave y las direcciones disponibles sobre el soporte.

CARACTERISTICAS, VENTAJAS, DESVENTAJAS DEL ACCESO DIRECTO.

- Un soporte direccionable es, normalmente, un disco o paquete de discos.

- Cada posición se localiza por su dirección absoluta, que en el caso del disco suele venir definida por dos parámetros, número de pista y número de sector o bien por tres parámetros pista, sector y número de cilindro; un cilindro ***i*** es el conjunto de pistas de número ***i*** de cada superficie de almacenamiento de la pila.

CARACTERISTICAS, VENTAJAS, DESVENTAJAS DEL ACCESO DIRECTO.

- En la práctica el programador no gestiona directamente direcciones absolutas, sino direcciones relativas respecto al principio del archivo.

- La manipulación de direcciones relativas permite diseñar el programa con independencia de la posición absoluta del archivo en el soporte.

- El programador crea una relación perfectamente definida entre la clave indicativa de cada registro y su posición física dentro del dispositivo de almacenamiento.

ACCESO DIRECTO.

Ejemplo:

Una compañía de empleados tiene un número determinado de vendedores y un archivo en el que cada registro corresponde a un vendedor. Existen 150 vendedores, cada uno referenciado por un número de 5 dígitos. Si se tuviera que asignar un archivo de 100000 registros, cada registro se corresponderá con una posición del disco.

ACCESO DIRECTO.

Ejemplo:

Para el diseño del archivo se creará 250 registros (un 33 por 100 más del número de registros necesarios, 25 por 100 suele ser un porcentaje habitual) que se distribuirán de la siguiente forma:

1. Posiciones 0 -199 constituyen el área principal del archivo y en ella se almacenarán todos los vendedores.
2. Posiciones 200-249 constituyen el área de desbordamiento, si K(1) <> K(2), pero f(K(1)=f(K(2)), y el registro con clave K(1) ya está almacenado en el área principal, entonces el registro con K(2) se almacena en el área de desbordamiento.

ACCESO DIRECTO.

Ejemplo:

La función f se puede definir como:

- f (k) = resto cuando K se divide por 199, esto es el módulo de 199; 199 ha sido elegido por ser el número primo mayor y que es menor que el tamaño del área principal.
- Para establecer el archivo se borran primero 250 posiciones. A continuación para cada registro de vendedor se calcula p = f (k). Si la posición p está vacía, se almacena el registro en ella. En caso contrario, se busca secuencialmente a través de las posiciones 200, 201,… para el registro con la clave deseada.

FUNCIONES HASH. SOLUCION DE COLICIONES.

- El método llamado por transformación de claves (hash), permite aumentar la velocidad de búsqueda sin necesidad de tener los elementos ordenados. (Capitulo 9 pag. 395, Cairo G, 2ed).
- Cuenta también con la ventaja de que el tiempo de búsqueda es prácticamente independiente del número de componentes del arreglo. (Capitulo 9 pag. 395, Cairo G, 2ed)
- Trabaja basándose en una función de transformación o función hash (H) que convierte una clave en una dirección (índice) dentro del arreglo. (Capitulo 9 pag. 395 ,Cairo G, 2ed).

dirección ← H(clave)

FUNCIONES HASH. SOLUCION DE COLICIONES.

- Cuando se tienen claves que no se corresponden con índices (p. ejem. por ser alfanuméricas), o bien cuando las claves son valores numéricos muy grandes, debe utilizarse una función hash que permita transformar la clave para obtener una dirección apropiada.
- Esta función hash debe de ser simple de calcular y debe de asignar direcciones de la manera mas uniforme posible. Es decir, dadas dos claves diferentes debe generar posiciones diferentes.
- Si esto no ocurre ***(H(K_1)=d,H(K_2)=d y K1 ≠ K2)***, hay una colisión. Se define, entonces, una colisión como la asignación de una misma dirección a dos o más claves distintas.

(Capitulo 9 pag. 396, Cairo G, 2ed).

FUNCIONES HASH. SOLUCION DE COLICIONES.

- Se define, entonces, una ***colisión*** como la asignación de una misma dirección a dos o más claves distintas.

- Por todo lo mencionado, para trabajar con este método de búsqueda debe elegirse previamente:
  - Una función hash que sea fácil de calcular y que distribuya uniformemente las claves.

  - Un método para resolver colisiones. Si estas se presentan se debe contar con algún método que genere posiciones alternativas.

FUNCIONES HASH. SOLUCION DE COLICIONES.

Función Módulo (por división).

- Consiste en tomar el residuo de la división de la clave entre el numero de componentes del arreglo.
- La función hash queda definida por la siguiente formula:

$$H(K) = (K \bmod N) + 1$$

- Se recomienda que ***N*** sea el numero primo inmediato inferior al numero total de elementos.

FUNCIONES HASH. SOLUCION DE COLICIONES.

Función Módulo (por división).

Ejemplo.

Vector de 100 posiciones.

Clave: 12325

Procedimiento:

H(clave) = 12325 MOD 100

H(clave) = 25

Lo que quiere decir que la clave será guardada en la posición 25 del vector.

FUNCIONES HASH. SOLUCION DE COLICIONES.

Función Centro de Cuadrados.

- Consiste en elevar al cuadrado la clave y tomar los dígitos centrales como dirección. El numero de dígitos a tomar queda determinado por el rango del índice.
- La función hash que definida por la sig. formula:

$$H(K) = digitos\_centrales\ (K^2) + 1$$

FUNCIONES HASH. SOLUCION DE COLICIONES.

Función Centro de Cuadrados.

Ejemplo.

Vector de 100 posiciones.

Clave: 562

Procedimiento:

H(clave) = 562^2

H(clave) = 315844

Lo que quiere decir que la clave será guardada en la posición 58.

FUNCIONES HASH. SOLUCION DE COLICIONES.

Función Plegamiento.

- Consiste en dividir la clave en partes de igual numero de dígitos (la ultima puede tener menos dígitos) y operar con ellas, tomando como dirección los dígitos menos significativos.
- La operación entre las partes puede hacerse por medio de sumas o multiplicaciones.
- La función hash queda definida por la sig. formula:

$$H(K) = digmensig\ ((d_1 \ldots d_i) + (d_{i+1} \ldots d_j) + \ldots + (d_1 \ldots d_n)) + 1$$

FUNCIONES HASH. SOLUCION DE COLICIONES.

Función Plegamiento.

Ejemplo.

Vector de 100 posiciones.

Clave: 28312405

Procedimiento:

H(clave) = 283 + 124 + 05

H(clave) = 412 (resultado que sobrepasa el límite del vector, por este motivo se trunca) 4|12 = 12

Así la clave será guardada en la posición 12 del vector.

FUNCIONES HASH. SOLUCION DE COLICIONES.

Función Truncamiento.

- Consiste en tomar algunos dígitos de la clave y formar con ellos una dirección.
- La función hash queda definida por la sig. fórmula:

$$H(K) = elegirdigitos\ (d_1, d_2 ... d_n) + 1$$

FUNCIONES HASH. SOLUCION DE COLICIONES.

Función Truncamiento.

H(K1) = dígitos(7 2 5 9) + 1 = 76
H(K2) = dígitos(9 3 5 9) + 1 = 96

- Una de las principales desventajas de utilizar truncamiento y en sí el método de hashing es que dos o más claves puede tomar una misma posición dentro de la tabla de hash, a esto es a lo que se le llama Colisión.
- Cuando hay colisiones se requiere de un proceso adicional para encontrar la posición disponible para la clave, lo cual disminuye la eficiencia del método.

EJEMPLO EN JAVA DE ASIGNACIÓN DE CLAVES MEDIANTE FUNCIÓN HASH, SI EXISTE COLISIÓN REASIGNA DIRECCIÓN MEDIANTE CALCULO LINEAL

```
public class Hash {
    public static void main(String[] args) throws FileNotFoundException,
IOException {
        boolean asig=false;
        int r=0;
        int has=0;
        int n=0;
        int mat[][]=new int[100][100];
        do{
  Scanner in=new Scanner(System.in);
 System.out.println("de cuanto es el registro");
  n=in.nextInt();
  if(n<10){
      System.out.println("no se puede hacer el valido de registro");
  }
}while(n<10);
        for(int i=1;i<=n;i++){
            for(int j=1;j<=2;j++){
             mat[i][j]=i;
             }
        }
File inFile = new File("hash.txt");
BufferedReader f1 = new BufferedReader(new FileReader(inFile));
String lif1=f1.readLine();
```

1/3

EJEMPLO EN JAVA: HASH Y COLISIÓN

```
while(lif1!=null){
     asig=false;
 r=Integer.parseInt(lif1);
 has=(r%n)+1;
 for(int i=1;i<=n;i++){
     if(mat[i][1]==has){
         if(mat[i][2]==has){
         mat[i][2]=r;
     }else{
              for(int j=i;j<=n;j++){
                  if(mat[j][1]==mat[j][2]){
                      mat[j][2]=r;
                      j=n+1;
                      asig=true;
                  }
              }
              if(asig==false){
                  for(int j=1;j<=i;j++){
                  if(mat[j][1]==mat[j][2]){
                      mat[j][2]=r;
                      j=n+1;
                      asig=true;
                  }
                  }
              }
         }
     }
 }
 lif1=f1.readLine();
 }
```

EJEMPLO EN JAVA: HASH Y COLISIÓN

```
 for(int i=1;i<=n;i++){
      if(mat[i][2]==i){
          mat[i][2]=0;
      }
  }

  for(int i=1;i<=n;i++){
      for(int j=1;j<=2;j++){
      System.out.print(mat[i][j]+"\t");
  }
      System.out.println();
  }
  f1.close();
    }

}
```

FUNCIONES HASH. DIRECCIONAMIENTO ABIERTO.

- En la técnica de direccionamiento abierto para manejo de colisiones todos los elementos se almacenan en la tabla hash misma. Es decir, cada entrada en la tabla contiene un elemento del diccionario o está vacía.
- Cuando se busca un elemento, se examinan sistemáticamente todos los slots hasta que se encuentra el elemento deseado o se determina que no se encuentra en la tabla.
- No hay listas de elementos almacenadas fuera de la tabla como con la técnica de encadenamiento y por lo tanto, en el direccionamiento abierto, es posible que la tabla se complete en su totalidad y que no se puedan realizar más inserciones.
- El factor de carga $\boldsymbol{\alpha}$ de una tabla hash con direccionamiento abierto nunca puede exceder 1.

FUNCIONES HASH. DIRECCIONAMIENTO ABIERTO.

- Para realizar una inserción usando el método de direccionamiento abierto se examinan sistemáticamente los slots de la tabla hasta que se encuentra una posición libre. En lugar 9 de usar un orden prefijado como 0, 1, . . . , m − 1 (que requiere un tiempo de búsqueda Θ(n)), la secuencia de posiciones que se examinan depende de la clave que se inserta. Para determinar los slots a examinar, se extiende la función hash para incluir el número de salto como un segundo argumento:

$$h : U \times \{0, 1, \ldots, m - 1\} \rightarrow \{0, 1, \ldots, m - 1\}$$

FUNCIONES HASH. DIRECCIONAMIENTO ABIERTO.

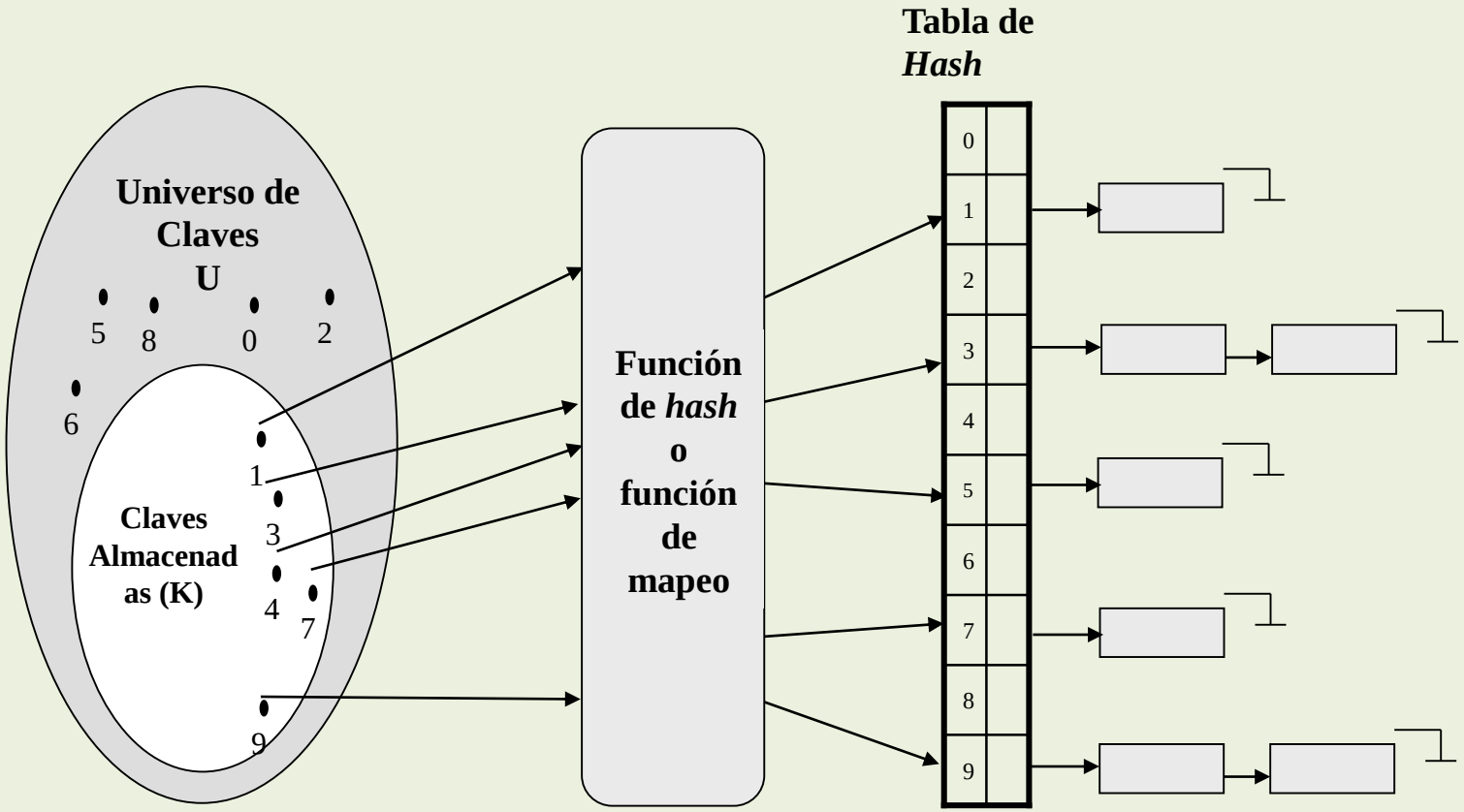
Con el direccionamiento abierto, para cada clave k se examina la secuencia de slots:

$$h(k, 0), h(k, 1), \ldots, h(k, m - 1)$$

y se requiere que dicha secuencia de slots sea exhaustiva, es decir que sea una permutación de 0, 1, . . . , m − 1, de forma tal que todos los slots en la tabla se examinen.

FUNCIONES HASH. DIRECCIONAMIENTO ABIERTO.

Hash Abierto



FUNCIONES HASH. DIRECCIONAMIENTO CERRADO.

El *hash cerrado* soluciona las colisiones buscando celdas alternativas hasta encontrar una vacía (dentro de la misma tabla). Se va buscando en las celdas: $d_0(k), d_1(k), d_2(k), \ldots,$ donde

$$d_i(k) = (h(k) + f(i)) \bmod MAX\_TABLA$$

$$h(k, i) = (h'(k) + f(i)) \bmod MAX\_TABLA$$

Cuando se busca una clave, se examinan varias celdas de la tabla hasta que se encuentra la clave buscada, o es claro que esta no está almacenada. La inserción se efectúa probando la tabla hasta encontrar un espacio vacío.

FUNCIONES HASH. DIRECCIONAMIENTO CERRADO.

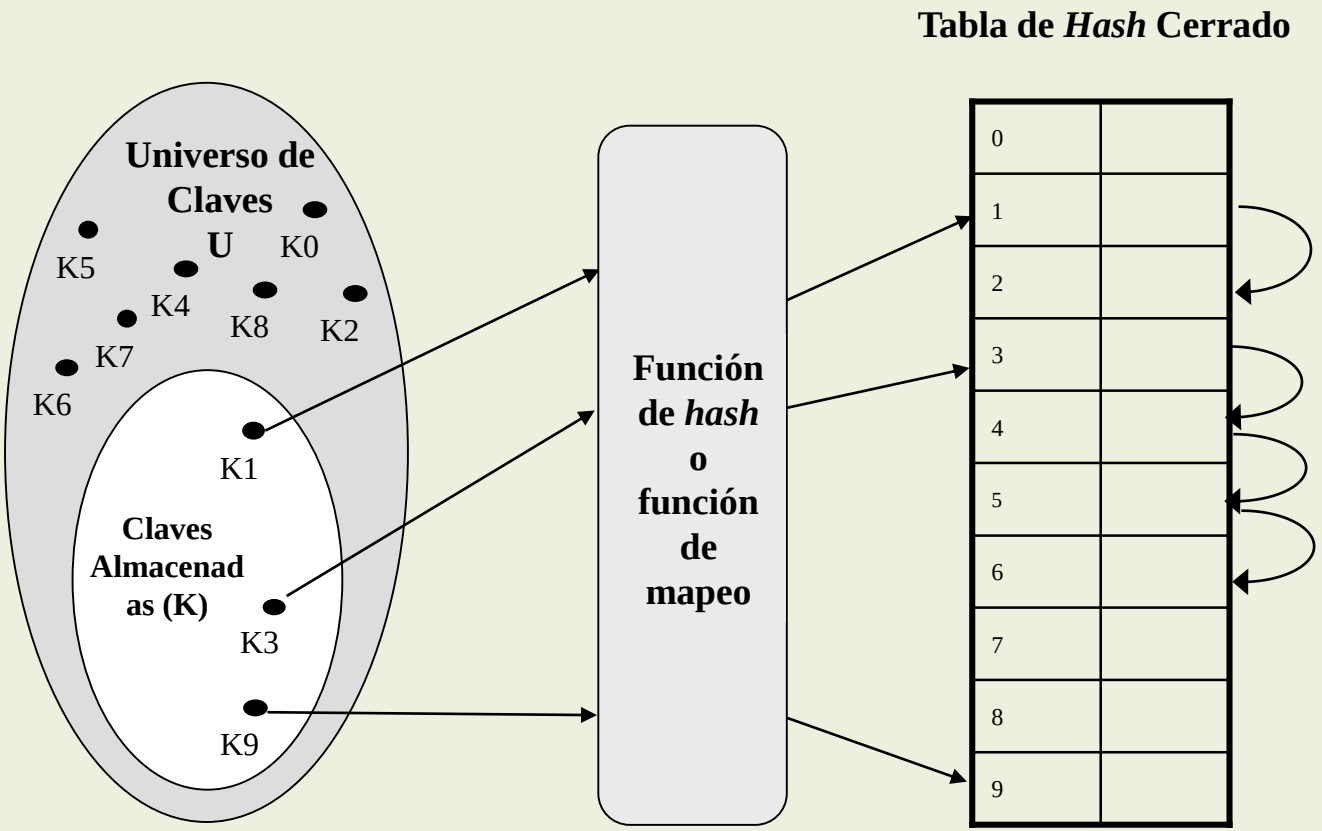
Ventaja: Elimina totalmente los apuntadores. Se libera así espacio de memoria, el que puede ser usado en más entradas de la tabla.

Desventaja: Si la aplicación realiza eliminaciones frecuentes, puede degradarse el rendimiento de la misma. Se requiere una tabla más grande. Para garantizar el funcionamiento correcto, se requiere que la tabla de *hash* tenga, por lo menos, el **50% del espacio disponible**.

Dentro de la dispersión cerrada hay tres estrategias distintas para localizar posiciones alternativas en la tabla: la ***exploración lineal***, la ***exploración cuadrática*** y la ***dispersión doble***. Cada una de ellas implica una función ***f(i)*** diferente.

FUNCIONES HASH. DIRECCIONAMIENTO CERRADO.

Hash Cerrado



ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

- Los archivos aleatorios o de acceso directo tienen una gran rapidez de acceso comparados con los secuenciales; son fáciles de referenciar número de orden del registro, y la facilidad de mantenimiento.
- La lectura/escritura de un registro es rápida, ya que se accede directamente al registro y no se necesita recorrer los anteriores.
- El inconveniente de los archivos directos es que la dirección de almacenamiento en el soporte direccionable se obtiene por medio de un algoritmo de conversión que transforma números de orden (clave) en direcciones de almacenamiento.

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

Creación

- El proceso de creación de un archivo directo o aleatorio consiste en ir introduciendo los sucesivos registros en el soporte que los va a contener y en la detección obtenida resultante del algoritmo de conversión. Si al introducir un registro se encuentra ocupada la dirección, el nuevo registro deberá ir a la zona de sinónimos o de excedentes.

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

```
Inicio
        abrir archivo
        leer registro
        mientras <> FF hacer
        calcular dirección mediante algoritmos de conversión
        escribir 'dirección libre S/N '
                leer respuesta
                si respuesta = 'S'
                entonces
                        grabar registros
                sino
                        buscar espacio en área de sinónimos
                        grabar registro
                fin_si
                leer registro
                fin_mientras
fin
```

CREACIÓN E INSERCIÓN DE DATOS EN FICHEROS DE ACCESO DIRECTO EN JAVA.

```
public class Principal {

    public static void main(String args []) throws
      IOException{
        Scanner leer = new Scanner (System.in);
        System.out.println("Ingrese un texto: ");
        String text = leer.nextLine();
        ArchivoApp22 raf = new ArchivoApp22();
        raf.Random("InsertarRandom");
        raf.write(text.trim());
        raf.close();
        System.out.println("Archivo guardado");
    }
}
```

CREACIÓN E INSERCIÓN DE DATOS EN FICHEROS DE ACCESO DIRECTO EN JAVA.

```
public class ArchivoApp22 {

     private RandomAccessFile raf;
     private File f;
     private String path = "c:\\Prueba/";

     public void Random (String name) throws IOException{
          f = new File (path+name+".txt");
          if(!f.exists()){
          f.createNewFile();
          }
          raf = new RandomAccessFile (f,"rw");
          raf.seek(0); //ubicamos el apuntador en la posicion 0 del archivo
     }

     public void write(String text) throws IOException{
          if(text.length()==0){
          return; }
          raf.writeBytes(text);
     }

     public void close() throws IOException{
          if(raf!=null){
          raf.close();
          }
     }
}
```

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

Altas.

La operación de altas en un archivo directo o aleatorio consiste en ir introduciendo los sucesivos registros en una determinada posición, especificada a través del índice. Mediante el índice nos posicionaremos directamente sobre el byte del fichero que se encuentra en la posición **(indice - 1) * tamano_de(<tipo_registros_del_archivo>)** y escribiremos allí nuestro registro.

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

```
Inicio //ALGORITMO ALTA
        Repetir
        Leer 'número de registro de alta' NR
        Si 1 -< NR -< ALTO
        Entonces
        Leer registro NR
        Si SW = 1
                Entonces
                        Escribir 'registro ya existe'
                        Sino
                        SW = 1
                        Leer datos del registro
                        Escribir Sw y datos registro NR
                Fin_si
        Sino
                Escribir 'error, rago 1..ALTO'
        Fin_si
        Hasta_que no se deseen mas altas
fin
```

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

CONSULTA

El proceso de consulta de un archivo o aleatorio es rápido y debe comenzar con la entrada del número o números de registros a consultar. Las operaciones a realizar son:

- Definir clave del registro buscado
- Aplicar algoritmo de conversión clave a dirección
- Lectura del registro ubicado en la dirección obtenida
- Comparación de las claves de los registros leído y buscado
- Exploración secuencial del área de excedentes, ni no se encuentra El registro en esta área es que no existe.

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

```
//Algoritmo consulta de registro
Mientras <> FF hacer
        Leer registro R
        Ir a subprograma de obtención de dirección
        Leer registro S
        Si R = S
        Entonces
                Llamar_a subprograma de consulta
        Sino
                Leer área de sinónimos
                Si FF
                        Entonces
                                Escribir 'registro no existe'
                        Sino llamar_a subprograma consulta
                Fin_si
        Fin_si
Fin_mientras
```

BÚSQUEDA DE UN DATO EN ARCHIVOS CON ACCESO DIRECTO EN JAVA

```
public class ArchivoApp26 {
    public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
        RandomAccessFile fichero = null;
        String palabra, cadena;
        StringBuilder auxBuilder;
        long pos = 0;
        int indice;
        try {
            //se abre el fichero para lectura/escritura
            fichero = new
              RandomAccessFile("C:\\Prueba/BuscarRAF.txt","rw");
            //Se pide la palabra a buscar
            System.out.print("Introduce palabra: ");
            palabra = sc.nextLine();
            //lectura del fichero
            cadena = fichero.readLine(); //leemos la primera línea
```

BÚSQUEDA DE UN DATO EN ARCHIVOS CON ACCESO DIRECTO EN JAVA

```
while(cadena!=null){
    indice = cadena.indexOf(palabra); //buscamos la palabra en la
      línea leída
    while(indice!=-1){
        auxBuilder = new StringBuilder(cadena);
        auxBuilder.replace(indice, indice+palabra.length(),
          palabra.toUpperCase());
        cadena = auxBuilder.toString();
        fichero.seek(pos);
        fichero.writeBytes(cadena);
        //compruebo si se repite la misma palabra en la línea
        indice = cadena.indexOf(palabra);
    }
    pos = fichero.getFilePointer();
    cadena = fichero.readLine(); //lectura de la línea
}
```

BÚSQUEDA DE UN DATO EN ARCHIVOS CON ACCESO DIRECTO EN JAVA

```
    } catch (FileNotFoundException ex) {
    System.out.println(ex.getMessage());
    } catch (IOException ex) {
    System.out.println(ex.getMessage());
}finally {
    try {
    if (fichero != null) {
    fichero.close();
    }
    } catch (IOException e) {
    System.out.println(e.getMessage());
    }
    }
}
}
```

3/3

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

BAJAS

Para realizar una baja se toma un campo indicador en el que su valor sea 0, y cuando exista, se pone a 1. Este tipo de una baja lógica, que significa que, pese a usar un registro dado de baja, sigue ocupando el mismo espacio que si estuviera presente.

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

```
Repetir //Algoritmo para la baja de registros
        Leer NR {número de registro}
        Si 1 /< NR /< TOTAL de registros
                Entonces
                        Leer registro NR
                        Si SW = 0
                                Entonces
                                Escribir '… precaución/registro no existe'
                        Sino
                                Escribir registro
                                Si la opción de baja es correcta
                                Entonces
                                        SW ← 0
                                        Escribir SW en el registro NR
                                Fin_si
                        Fin_si
        Sino
                Escribir 'número de registro no correcto'
        Fin si
```

ELIMINACIÓN DE UN REGISTRO CON ACCESO DIRECTO EN JAVA.

```
public class BorrarRAF extends javax.swing.JFrame {
public String cuenta;
public BorrarRAF() { BorrarRegistro(); }
private void BorrarRegistro() {
    int i=0;
        RandomAccessFile fichero = null;
        String cuenta, cadena;
        StringBuilder auxBuilder;
        String line="";
        long pos = 0;
        int indice;
//Clave a buscar en el archivo para ser borrado.
        Scanner in =  new Scanner (System.in);
        System.out.println("Introduce la clave: ");
        cuenta=in.nextLine();
        try {
        //Cargamos el archivo donde se desea borrar el registro
        fichero = new
RandomAccessFile("C:\\Prueba/BorrarRAF.txt","rw");
        cadena = fichero.readLine();
```

ELIMINACIÓN DE UN REGISTRO CON ACCESO DIRECTO EN JAVA.

```
while(cadena!=null){
        indice = cadena.indexOf(cuenta);
        while(indice!=-1){
        auxBuilder = new StringBuilder(cadena);
        cadena = auxBuilder.toString();
        JOptionPane.showMessageDialog(this,"ELIMNADO");
        i=1;
        line=cadena;
        fichero.seek(pos);
        fichero.writeBytes(cadena);

        indice = cadena.indexOf(cuenta);
        break;
        }
        pos = fichero.getFilePointer();
        cadena = fichero.readLine();
        }
```

ELIMINACIÓN DE UN REGISTRO CON ACCESO DIRECTO EN JAVA.

```
} catch (FileNotFoundException ex) {
        System.out.println(ex.getMessage());
        } catch (IOException ex) {
        System.out.println(ex.getMessage());
        }finally {
        try {
        if (fichero != null) {
        fichero.close();
        }
        } catch (IOException e) {
        System.out.println(e.getMessage());
        } }
 try{
        //Lectura de archivo origial
        BufferedReader br = new BufferedReader(new
FileReader("C:\\Prueba/BorrarRAF.txt"));
```

ELIMINACIÓN DE UN REGISTRO CON ACCESO DIRECTO EN JAVA.

```
//Archivo temporal que copia los registros, excepto el registro a eliminar
        BufferedWriter bw=new BufferedWriter(new
FileWriter("C:\\Prueba/RegistroNuevo.txt"));
        String linea=br.readLine();
        while(linea!=null){
        if( linea.equals(line)){
        linea=br.readLine();}
        else{
        bw.write(linea);
        bw.newLine();
        linea=br.readLine();
        }
        }
        bw.close();
        br.close();
        }catch(IOException e){
        System.out.println("ERROR: "+e);
        }
```

ELIMINACIÓN DE UN REGISTRO CON ACCESO DIRECTO EN JAVA.

```
 File inFile = new File("C:\\Prueba/BorrarRAF.txt");
        File inFile2 = new
File("C:\\Prueba/RegistroNuevo.txt");
        inFile.delete();
        inFile2.renameTo(inFile);
        if(i==0){
        JOptionPane.showMessageDialog(this,"NO ENCONTRADO");
        }
        }


    public static void main(String args[]) {
    new BorrarRAF();
        }

}
```

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

MODIFICACIONES

En un archivo aleatorio se localiza el registro que se desea modificar número de registro; se modifica el contenido y se reescribe.

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

```
//Algoritmo modificación
Inicio
       Repetir
       Leer 'número de registro',NR
       Si /<NR /< TOTAL de registros
       Entonces
              Escribir 'registro no existe'
       Sino
              Escribir registro
              Leer modificaciones
              Escribir nuevo registro
       Fin_si
Sino
       Escribir 'número de registro no correcto'
Fin_si
Hasta_que no haya modificaciones
Fin_si
```

ACTUALIZACIÓN DE REGISTRO CON ACCESO DIRECTO EN JAVA

```
public class ActualizarRAF extends JFrame{
String line;
public ActualizarRAF() {
        LeerRAF();
    }
private void LeerRAF() {
    int i=0;
        RandomAccessFile fichero = null;
        String cuenta, cadena;
        StringBuilder auxBuilder;
        long pos = 0;
        int indice;
        cuenta="9814635";
        try {
        fichero = new
RandomAccessFile("C:\\Prueba/ActualizarRAF.txt","rw");
        cadena = fichero.readLine();
        while(cadena!=null){
        indice = cadena.indexOf(cuenta);
```

ACTUALIZACIÓN DE REGISTRO CON ACCESO DIRECTO EN JAVA

```
while(indice!=-1){
        auxBuilder = new StringBuilder(cadena);
        cadena = auxBuilder.toString();
        i=1;
        //JOptionPane.showMessageDialog(this,"ELIMNADO");
        line=cadena;
        fichero.seek(pos);
        fichero.writeBytes(cadena);
        indice = cadena.indexOf(cuenta);
        break;
        }
        pos = fichero.getFilePointer();
        cadena = fichero.readLine();
        }
        } catch (FileNotFoundException ex) {
        System.out.println(ex.getMessage());
        } catch (IOException ex) {
        System.out.println(ex.getMessage());
        }
```

ACTUALIZACIÓN DE REGISTRO CON ACCESO DIRECTO EN JAVA

```
finally {
        try {
        if (fichero != null) {
        fichero.close();
        }
        } catch (IOException e) {
        System.out.println(e.getMessage());
        }         }
        if(i==1){
        //String cuenta="9814635";
        //Nuevos campos a actualizar
        String nombre="JAIR";
        String paterno="vazquez";
        String materno="palma";
        String naci="10/09/1985";
        nombre=nombre.toUpperCase();
        paterno=paterno.toUpperCase();
        materno=materno.toUpperCase();
        naci=naci.toUpperCase();
        String
rfc=paterno.substring(0,2)+materno.substring(0,1)+nombre.substring(0,1)+naci.s
ubstring(2,4)+naci.substring(4,8);
```

ACTUALIZACIÓN DE REGISTRO CON ACCESO DIRECTO EN JAVA

```
try{
        BufferedReader br=new BufferedReader(new
FileReader("C:\\Prueba/ActualizarRAF.txt"));
        BufferedWriter bw=new BufferedWriter(new
FileWriter("C:\\Prueba/RegistroNuevo.txt"));
        String linea=br.readLine();
        while(linea!=null){
        if( linea.equals(line)){
        linea=br.readLine();
        }else{
        bw.write(linea);
        bw.newLine();
        linea=br.readLine();
        }
        }
        bw.write(s);
        bw.close();
        br.close();
        }catch(IOException e){
        System.out.println("ERROR: "+e);
        }
```

ACTUALIZACIÓN DE REGISTRO CON ACCESO DIRECTO EN JAVA

```
File inFile = new File("C:\\Prueba/ActualizarRAF.txt");
        File inFile2 = new File("C:\\Prueba/RegistroNuevo.txt");
        inFile.delete();
        inFile2.renameTo(inFile);
        }else if(i==0){
        JOptionPane.showMessageDialog(this,"NO ENCONTRADO");
        }
    }


    public static void main(String args[]) {
    new ActualizarRAF();
    }
}
```

ESTRUCTURA DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS, OPERACIONES).

TRATAMIENTO DE LAS COLISIONES

- Las colisiones son inevitables, y como se ha comentado, se originan cuando dos registros de claves diferentes producen la misma dirección relativa. En estos casos las colisiones se pueden tratar de dos formas diferentes.
- Suponer que un registro *e1* produce una dirección ***dl*** que ya está ocupada. ¿Dónde colocar el nuevo registro? Como se ha mencionado, existen dos métodos básicos.
  - Buscar una nueva dirección libre en el mismo espacio del archivo.
  - Asignar el registro a la primera posición libre de la zona de excedentes.

BIBLIOGRAFÍA

- Cairó, Osvaldo y Guardati, Silvia. (2002). Estructuras de datos (2a. Edición). McGraw-Hill.
- Criado, Ma. Asunción. (2006). Programación en lenguajes estructurados. AlfaOmega Ra-Ma.
- Drozdeck, Adam. (2007). Estructuras de datos y algoritmos en Java (2ª Edición). Thomson.
- Joyanes A. Luis, Fernandez A. Matilde. Java 2, manual de programación. McGreaHill.
- Zanella, Santiago, 2008. Estructura de datos.
- Joyanes Aguilar L., Fundamentos de programación, algoritmos, estructuras de datos y objetos, 4ed. McGrawHill.