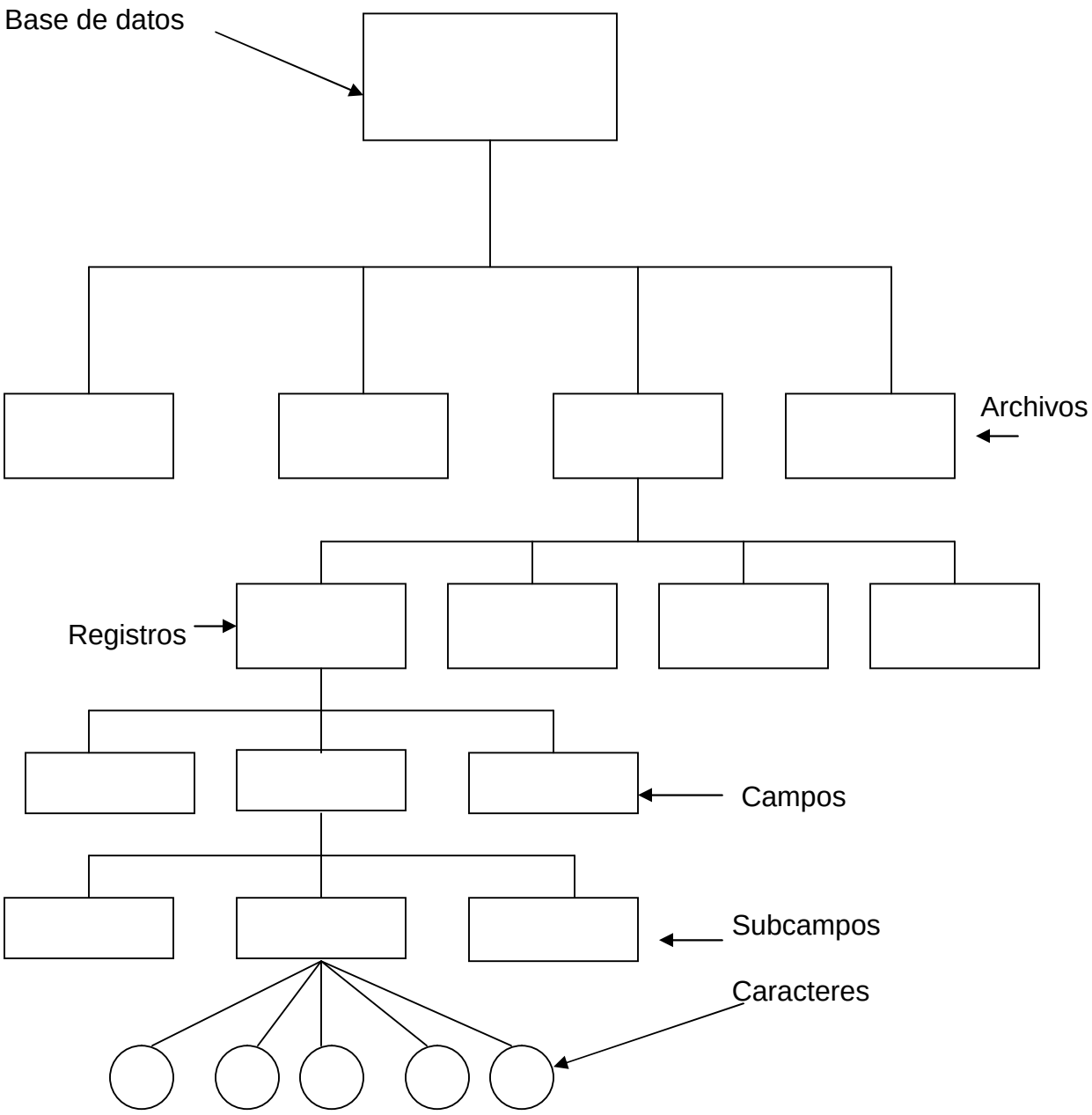
I. TIPOS DE ARCHIVO DE ACUERDO A SU ORGANIZACIÓN Y OPERACIONES SOBRE ÉSTOS.

1. ESTRUCTURA JERÁRQUICA

Los conceptos de carácter, campo, registro, archivo y base de datos son conceptos lógicos que se refieren al medio en que el usuario de computadoras ve los datos y se organizan. Las estructuras de datos se organizan de un modo jerárquico, de modo que el nivel más alto lo constituye la base de datos y el nivel más bajo el carácter.

Ejemplo: Estructura jerárquica de datos.



a) CAMPOS

Los caracteres se agrupan en campos de datos. Un campo es un ítem o elemento de datos elementales, tales como un nombre, número de empleados, ciudad, número de identificación, etc.

Un campo está caracterizado por su tamaño o longitud y su tipo de datos (cadena de caracteres, entero, lógico, etc.). Los campos pueden incluso variar en longitud. En la mayoría de los lenguajes de programación los campos de longitud variable no están soportados y se suponen de longitud fija.

Un campo es la unidad mínima de información de un registro. Los datos contenidos en un campo se dividen con frecuencia en subcampos.

Ejemplo: Campos de un registro.

Nombre	Dirección	Fecha de nacimiento	Estudios	Salario

El campo fecha se divide en los subcampos día, mes y año quedando de la siguiente manera:

Campo: Fecha de Nacimiento

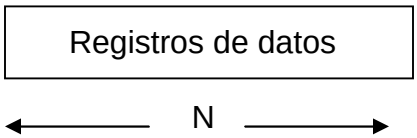
23	04	1963

subcampos: día mes año

b) REGISTROS

Un registro es una colección de información, normalmente relativa a una entidad particular. Un registro es una colección de campos lógicamente relacionados que pueden ser tratados como una unidad por algún programa.
Un ejemplo de un registro puede ser la información de un empleado que contienen los campos de nombre, dirección, fecha de nacimiento, etc.

Los registros pueden ser todos de longitud fija; por ejemplo, los registros de empleados pueden contener el mismo número de campos, cada uno de la misma longitud para nombre, dirección, fecha, etc. También pueden ser de longitud variable.



N = longitud del registro

Los registros organizados en campos se denominan registros lógicos.

c) ARCHIVOS (FICHEROS)

Un archivo o fichero de datos es una colección de registros relacionados entre sí con aspectos en común y organizados para un propósito específico. Por ejemplo un archivo de una clase escolar contiene un conjunto de registros de los estudiantes de esa clase.

Un archivo en una computadora es una estructura diseñada para contener datos, estos están organizados de tal modo que puedan ser recuperados fácilmente, actualizados o borrados y almacenados de nuevo en el archivo con todos los cambios realizados.

d) BASE DE DATOS

Una colección de archivos a los que puede accederse por un conjunto de programas y que contienen todos ellos datos relacionados, constituye una base de datos.

2. TERMINOLOGÍA

a) Clave

Una clave (key) o indicativo es un campo de datos que identifica al registro y lo diferencia de otros registros. Esta clave debe ser diferente para cada registro. Claves típicas son nombres o números de identificación.

b) Registro físico o bloqueo

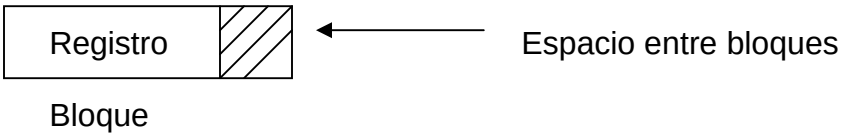
Un registro físico o bloque es la cantidad más pequeña de datos que pueden transferirse en una operación de entrada/salida entre la memoria central y los dispositivos periféricos o viceversa. Ejemplos de registros físicos son: una línea de impresión, un sector de un disco magnético, etc.
Un bloque puede contener uno o más registros lógicos.

c) Factor de bloqueo

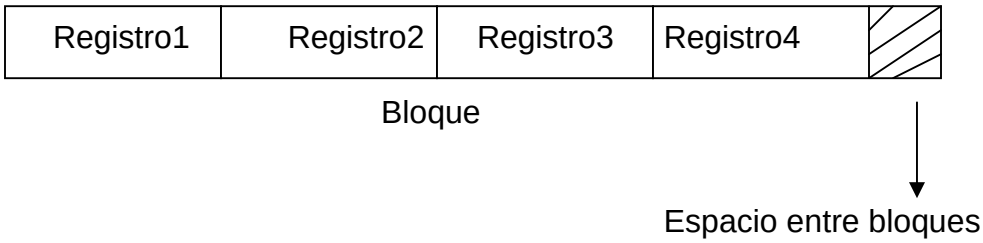
Otra característica que es importante en relación con los archivos es el concepto de factor de bloqueo. El número de registros lógicos que puede contener un registro físico se denomina factor de bloque.

Ejemplo: Factor de bloque

a) Un registro por bloque (factor = 1)



b) N registros por bloque (factor = N)



Se pueden dar las siguientes situaciones:

Registro lógico > Registro físico. En un bloque que contienen varios registros físicos por bloque; se denominan registros expandidos.

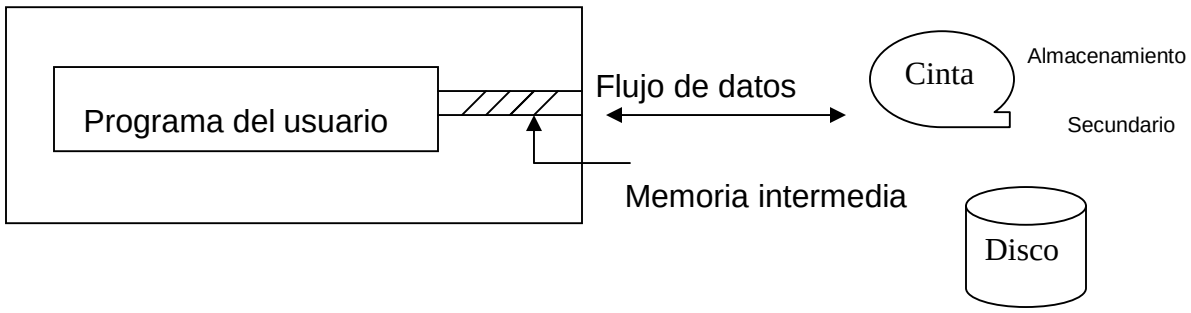
Registro lógico = Registro físico. El factor de bloqueo es 1 y se dice que los registros no están bloqueados.

Registro lógico < Registro físico. El factor de bloqueo es mayor que 1 y los registros están bloqueados.

La importancia del factor de bloqueo se puede apreciar mejor con un ejemplo.

Ejemplo: Suponer que se tienen dos archivos, uno de ellos tiene un factor de bloqueo de 1 (un registro en cada bloque). El otro archivo tiene un factor de bloqueo de 10 (10 registros/bloque). Si cada archivo contiene un millón de registros, el segundo archivo requerirá 900000 operaciones de entrada/salida menos para leer todos los registros.

Un factor de bloqueo mayor que 1 siempre mejora el rendimiento; entonces, ¿porqué no incluir todos los registros en un solo bloque? La razón reside en que las operaciones de entrada/salida que se realizan por bloques se hacen a través de un aumento de un área de la memoria central denominada memoria intermedia (buffer) y entonces el aumento del bloque implicará aumento de la memoria intermedia y por consiguiente se reducirá el tamaño de la memoria central.



El tamaño de una memoria intermedia de un archivo es el mismo que el del tamaño de un bloque. Como la memoria central es más cara que la memoria secundaria, no conviene aumentar el tamaño del bloque, sino más bien conseguir un equilibrio entre ambos criterios.

3. ORGANIZACIÓN DE ARCHIVOS

El soporte es el medio físico donde se almacenan los datos. Los tipos de soporte utilizados en la gestión de archivos son:

- Soportes secuenciales
- Soportes direccionables

Los soportes secuenciales son aquéllos en los que los registros informaciones están escritos unos a continuación de otros y para acceder a un determinado registro, n se necesita pasar por los n-1 registros anteriores. La secuencia puede corresponder al orden físico de los registros en el archivo (organización secuencial) o bien al orden de claves (ascendente o descendente) de los registros (organización indexada).

Los soportes direccionables se estructuran de modo que las informaciones registradas se pueden localizar directamente por su dirección y no se requiere pasar por los registros precedentes. En estos soportes los registros deben poseer un campo clave los diferencie del resto de los registros del archivo. Una dirección en un soporte direccionable puede ser número de pista y número de sector en un disco.

Los soportes direccionables son los discos magnéticos, aunque pueden actuar como soporte secuencial.

Según las características del soporte empleado y el modo en que se han organizado los registros, se consideran dos tipos de acceso a los registros de un archivo:

- Acceso secuencial
- Acceso directo

El acceso secuencial implica el acceso a un archivo según el orden de almacenamiento de sus registros, uno tras otro.
El acceso directo implica el acceso a un registro determinado, sin que ello implique la consulta de los registros precedentes. Este tipo de acceso sólo es posible con soportes direccionables.
La organización de un archivo define la forma en la que los registros se disponen sobre el soporte de almacenamiento, o también se define la organización como la forma en que se estructuran los datos en un archivo. En general, se consideran tres organizaciones fundamentales:

- Organización secuencial
- Organización directa o aleatoria (Random)
- Organización secuencial indexada (Indexed)

a) Organización secuencial

Un archivo con organización secuencial es una sucesión de registros almacenados consecutivamente sobre el soporte externo, de tal modo que para acceder a un registro n dado es obligatorio pasar por todos los n-1 artículos que le preceden.

Los registros se graban consecutivamente cuando el archivo se crea y se debe acceder consecutivamente cuando se leen dichos registros.

Organización secuencial

- El orden físico en que fueron grabados (escritos) los registros es el orden de lectura de los mismos.
- Todos los tipos de dispositivos de memoria auxiliar soportan la organización secuencial.

Los ficheros organizados secuencialmente contienen un registro particular el último que contiene un marca fin de archivo (EOF o bien FF). Esta marca fin de archivo suele ser un carácter especial como ' * '.

b) Organización directa

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.

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.

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.

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.

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.

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.

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.

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.

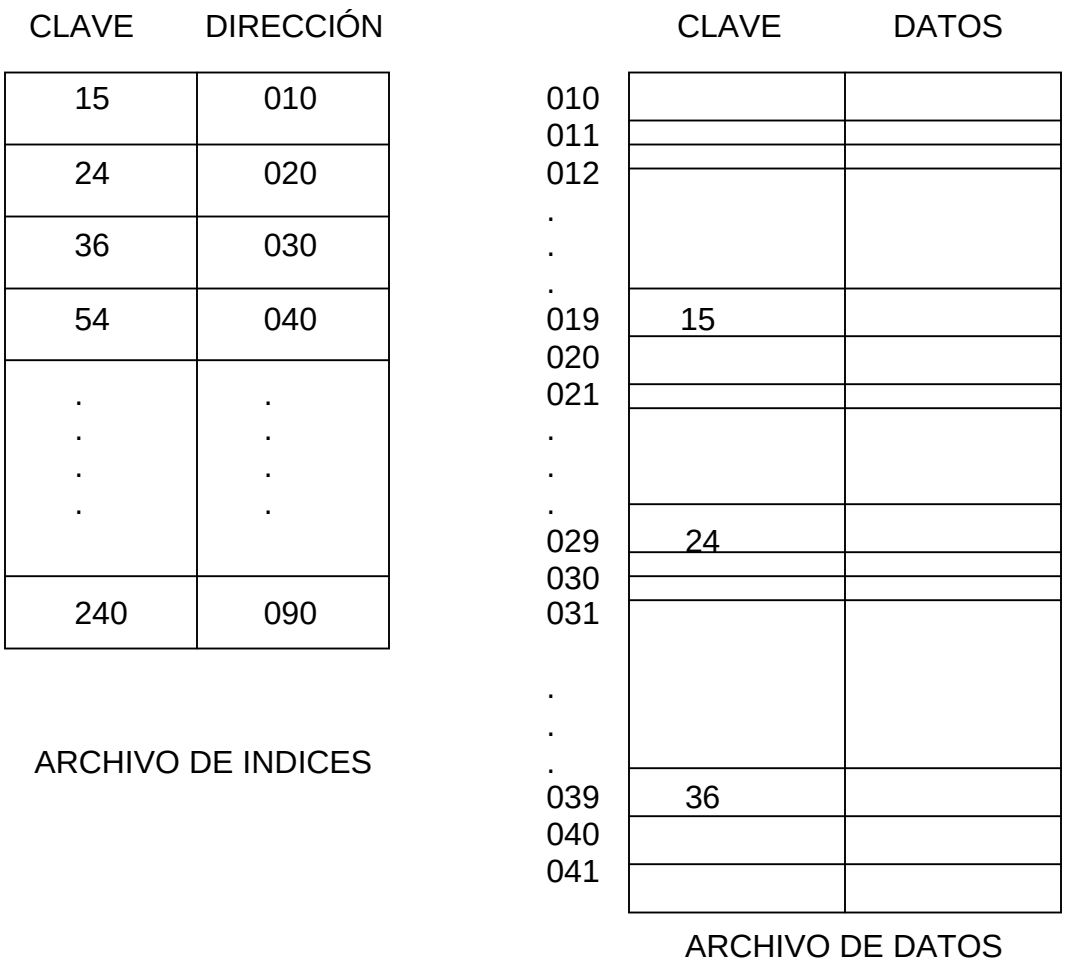
c) Organización secuencial indexada

Un diccionario es un archivo secuencial, cuyos registros son las entradas y cuyas claves son las palabras definidas por las entradas. Para buscar una palabra (una clave) no se busca secuencialmente desde la “a” hasta “z”, sino que se abre el diccionario por la letra inicial de la palabra. Si se desea buscar “índice”, se abre el índice por la letra I y en su primera página se busca la cabecera de página hasta encontrar la página más próxima a la palabra, buscando a continuación palabra a palabra hasta encontrar “índice”. El diccionario es un ejemplo típico de archivo secuencial-indexado con dos niveles de índices, el nivel superior para las letras iniciales y el nivel menor para las cabeceras de páginas se guardarán en un archivo de índice independiente de las entradas del diccionario (archivo de datos).

Por consiguiente, cada archivo secuencialmente-indexado consta de un archivo índice y un archivo de datos.

Un archivo está organizado en forma secuencial-indexada si:

- El tipo de sus registros contiene un campo clave identificador
- Los registros están situados en un soporte direccionable por el orden de los valores indicados por la clave
- Un índice para cada posición direccionable, la dirección de la posición y el valor de la clave; en esencia, el índice contienen la clave del último registro y la dirección de acceso al primer registro del bloque.



Un archivo en organización secuencial-indexada consta de las siguientes partes:

- Área de datos o primaria: contienen los registros en forma secuencial y está organizada en secuencia de claves sin dejar huecos intercalados

- Área de índices: es una tabla que contiene los niveles de índice, la existencia de varios índices enlazados se denomina nivel de indexación
- Área de desbordamiento o excedentes: utilizada, si fuese necesario, para las actualizaciones.

El área de índices es equivalente, en su función, al índice de un libro. En ella se refleja el valor de la clave identificativa más alta de cada grupo de registros del archivo y la dirección de almacenamiento del grupo.

Los archivos secuenciales-indexados presentan las siguientes ventajas:

- Rápido acceso
- El sistema de gestión de archivos se encarga de relacionar la posición de cada registro con su contenido mediante la tabla de índices.

Desventajas:

- Desaprovechamiento del espacio por quedar huecos intermedios cada vez que actualiza el archivo
- Se necesita espacio adicional para el área de índices

Los soportes que se utilizan para esta organización son los que permiten el acceso directo, los discos magnéticos. Los soportes de acceso secuencial no pueden utilizarse, ya que no dispone de direcciones para sus posiciones de almacenamiento.

4. Operaciones sobre Archivos

Las distintas operaciones que se pueden realizar a los registros de un archivo son:

- Creación
- Consulta
- Actualización
- Clasificación
- Reorganización
- Destrucción
- Fusión
- Rotura/estallido

a) Creación de un archivo

Es la primera operación que sufrirá el archivo de datos. Implica la elección de un entorno descriptivo que permita un ágil, rápido y eficaz tratamiento del archivo.
Para utilizar un archivo, éste tiene que existir, es decir, las informaciones de este archivo tienen que haber sido almacenados sobre un soporte y ser utilizables. La

creación exige organización, estructura, localizar observar espacio en el soporte de almacenamiento, transferencia del archivo de soporte antiguo al nuevo.
Un archivo puede ser creado por primera vez en un soporte, proceder de otro previamente existente en el mismo o diferente soporte, ser el resultado de un cálculo o ambas cosas a la vez.

La figura muestra un organigrama de la creación de un archivo ordenado de empleados de una empresa por el campo clave (número o código de empleado).

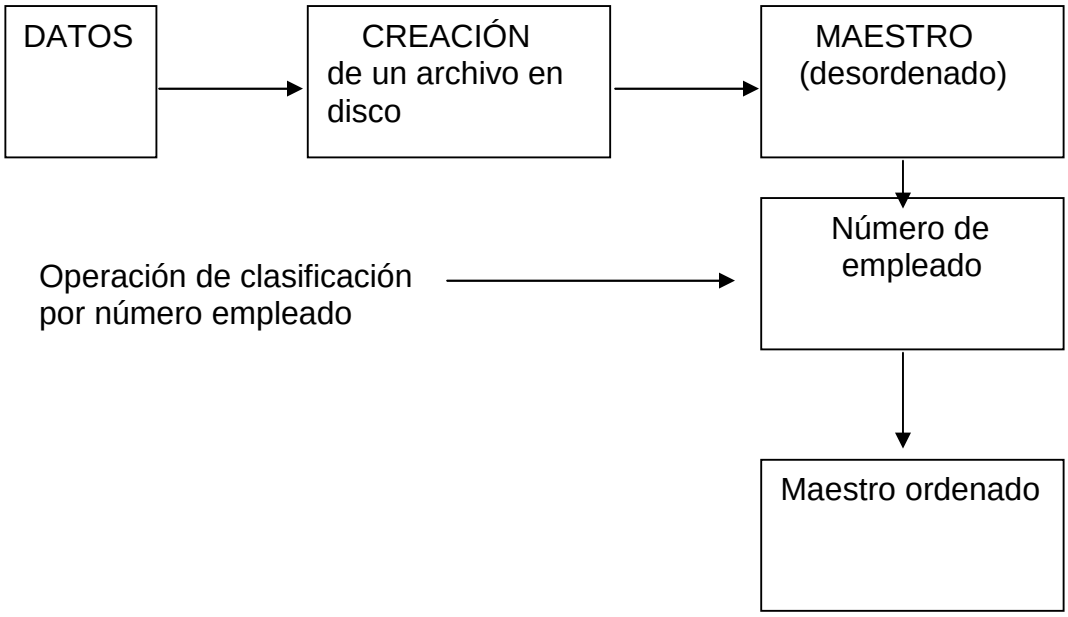
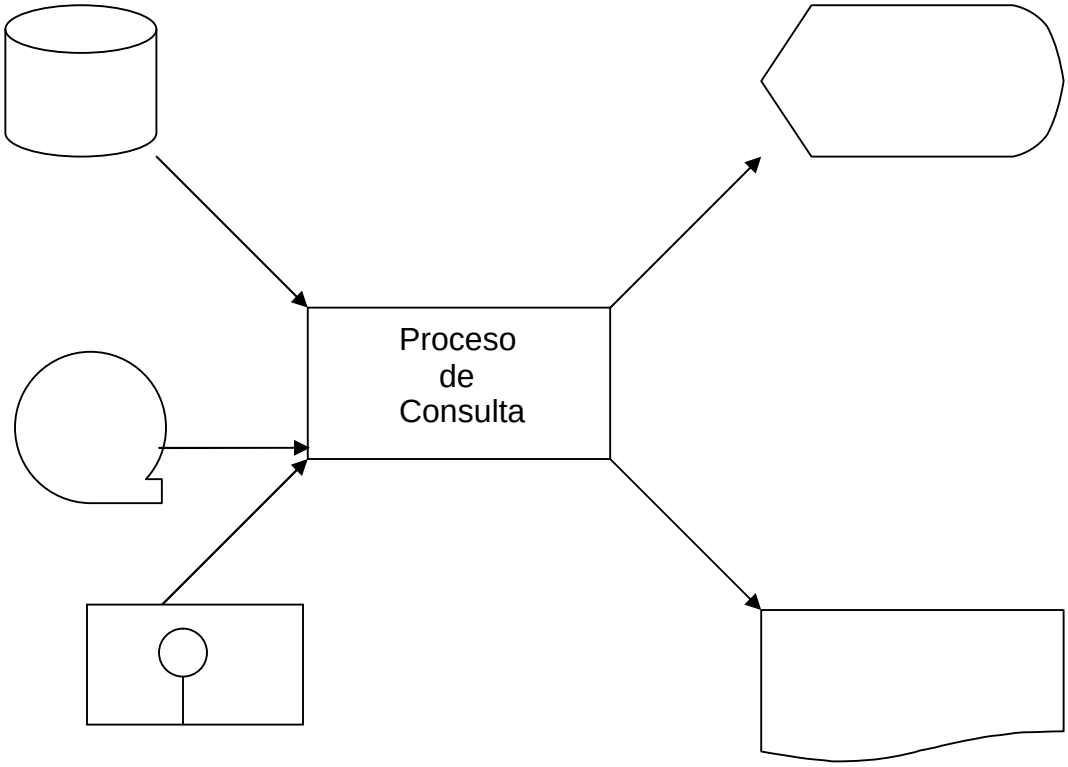


Figura: Creación de un archivo ordenado de empleados

b) Consulta de un Archivo

Es la operación que permite al usuario acceder al archivo de datos para conocer el contenido de uno, varios o todos los registros.

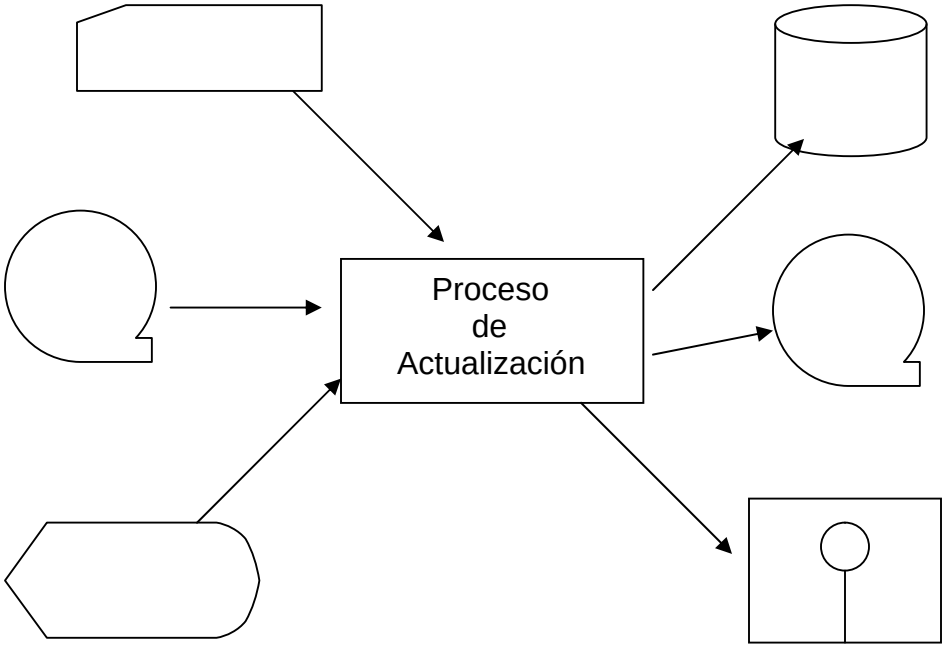


c) Actualización de un archivo

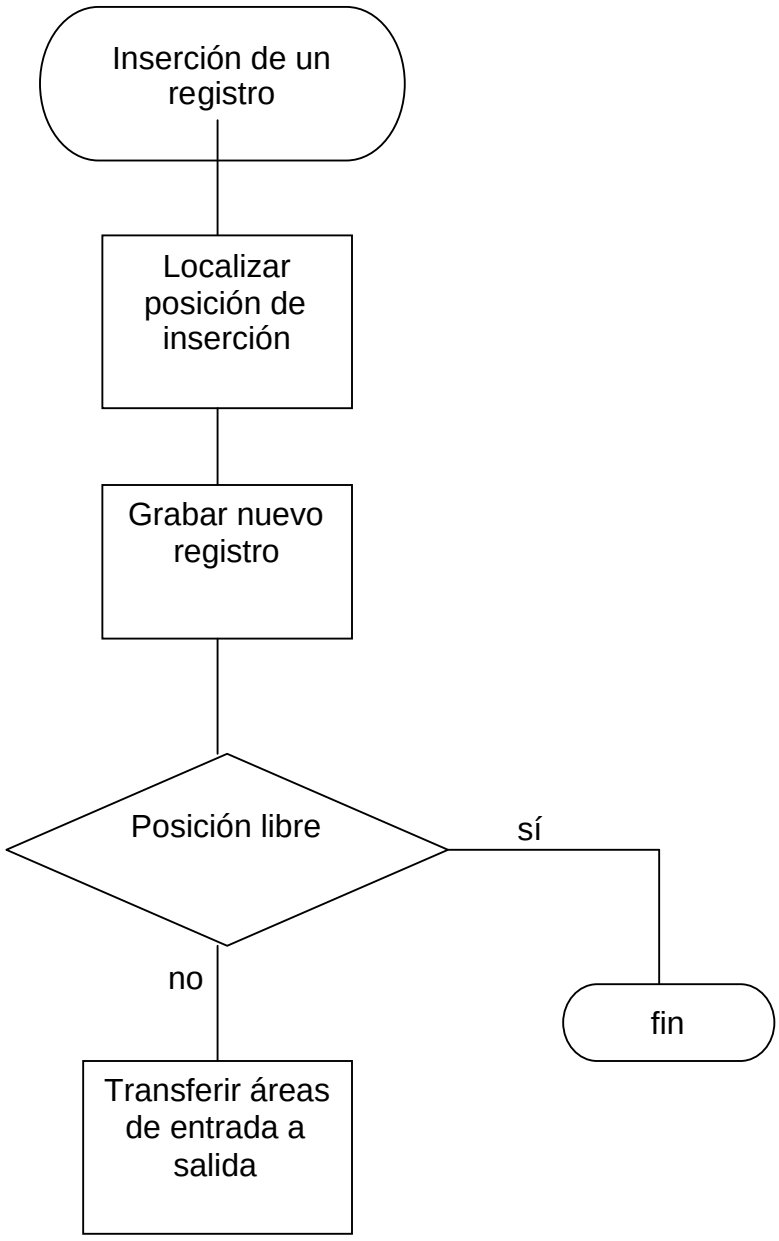
Es la operación que permite tener actualizado (puesto al día) el archivo, de tal modo que sea posible realizar las siguientes operaciones con sus registros:

- Consulta del contenido de un registro.
- Inserción de un registro nuevo en el archivo.
- Supresión de un registro existente
- Modificación de un registro

Actualización de un archivo



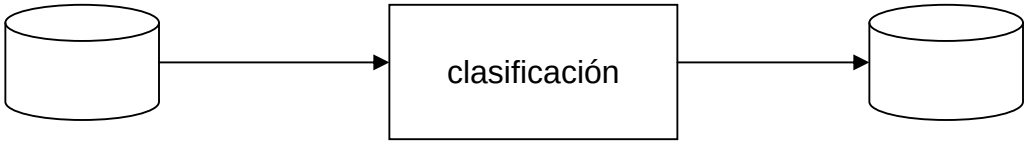
Actualización de un Archivo

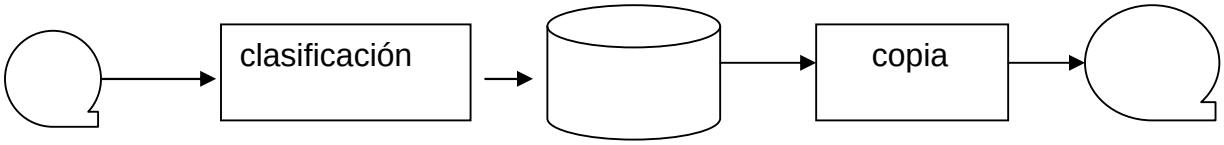


Un ejemplo de actualización es el de un archivo de almacén, cuyos registros contienen las existencias de cada artículo, precios, proveedores, etc. Las existencias, precios, etc. varían continuamente y exigen una actualización simultánea del archivo con cada operación de consulta.

d) Clasificación de un Archivo

Una operación muy importante en un archivo es la *clasificación u ordenación* (*sort,* en inglés). Esta clasificación se realizará de acuerdo con el valor de un campo específico, pudiendo ser *ascendente* (creciente) o *descendente* (decreciente): alfabética o numérica.

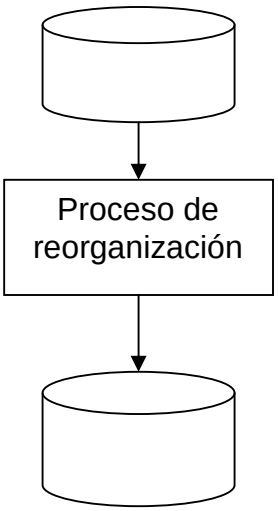




e) Reorganización de un archivo

Las operaciones sobre archivos modifican la estructura inicial o la óptima de un archivo. Lo índices, enlaces (punteros), zonas de sinónimos, zonas de desbordamiento, etc. se modifican con el paso del tiempo, lo que hace a la operación de acceso al registro cada vez más lenta.

La reorganización suele consistir en la copia de un nuevo archivo a partir del archivo modificado, a fin de obtener una nueva estructura lo más óptima posible.

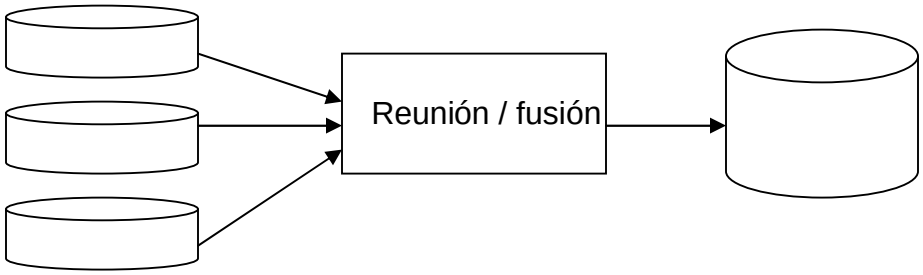


f) Destrucción de un archivo

Es la operación inversa a la creación de un archivo (kill, en inglés). Cuando se destruye (anula o borra) un archivo, éste ya no se puede utilizar y por consiguiente no se podrá acceder a ninguno de sus registros.

g) Reunión, fusión de un archivo

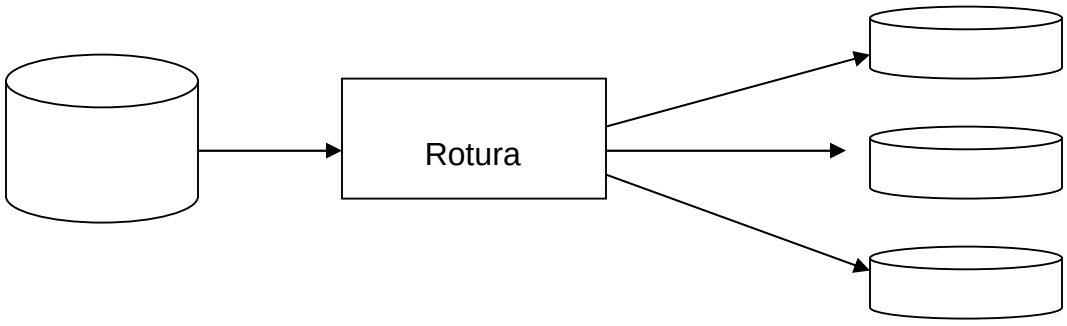
Reunión: Esta operación permite obtener un archivo a partir de otros varios.



Fusión: Se realiza una fusión cuando se reúnen varios archivos en uno solo, intercalándose unos en otros, siguiendo unos criterios determinados.

h) Rotura/ estallido de un archivo

Es la operación de obtener varios archivos a partir de un mismo archivo inicial.



GESTION DE ARCHIVOS

Las operaciones más usuales en los registros son:

- Consulta: lectura del contenido de un registro.
- Modificación: alterar la información contenida en un registro.
- Inserción: añadir un nuevo registro al archivo.
- Borrado: suprimir un registro del archivo.

Las operaciones sobre archivos se realizan mediante programas, de modo tal que los archivos se identifican por un nombre externo al que están asociados. Pueden existir programas que procesen el mismo archivo de datos. La mayoría de los programas ejecutarán las siguientes clases de funciones:

- **crear** archivos (create)
- **abrir** o **arrancar** (open) un archivo que fue creado con anterioridad a la ejecución de este programa.
- **incrementar** o **ampliar** el tamaño del archivo (append, extend)
- **cerrar** el archivo después que el programa ha terminado de utilizarlo (close)
- **borrar** (delete) un archivo que ya existe
- **transferir** datos **desde** (leer) o a (escribir) el dispositivo, diseñado por el programa.

Antes de cualquier dato pueda ser escrito o leído de un archivo, el archivo debe ser crear en el dispositivo correspondiente. Las operaciones sobre archivos tratan con la propia estructura del archivo, no con los registros de datos dentro del archivo.
Con anterioridad a la creación de un archivo se requiere diseñar la estructura del mismo mediante los campos de registro, longitud y tipo de los mismos.
Para poder gestionar un archivo mediante un programa es preciso declarar el archivo, su nombre y la estructura de sus registros. La declaración se realizará con las siguientes instrucciones

```
archivo nombre
registro
        Campo1 = tipo
        Campo2 = tipo
        Campo3 = tipo
...
```

Crear un archivo

La creación de un archivo es la operación mediante la cual se introduce la información correspondiente al archivo en un soporte de almacenamiento de datos.

Antes de que cualquier usuario pueda procesar un archivo, es preciso que éste haya sido creado previamente. El proceso de creación de un archivo será la primera operación a realizar. Una vez que el archivo ha sido creado, la mayoría de los usuarios simplemente desearán acceder al archivo y a la información contenida en él.
Para crear un archivo dentro de un sistema de computadoras se necesitan los siguientes datos:

- Nombre dispositivo/usuario indica el lugar donde se situará el archivo cuando se cree;
- Nombre del archivo: identifica el archivo entre los restantes archivos de una computadora;
- Tamaño del archivo: indica el espacio necesario para la creación del archivo;
- Organización del archivo: tipo de organización del archivo;
- Tamaño del bloque o registro físico: cantidad de datos que se leen o escriben en cada operación de entrada/salida (E/S)

El procesador de creación de un archivo se suele incluir como un subprograma dentro de un programa principal. Al ejecutar el programa de creación de un archivo se pueden generar una serie de errores entre los que se pueden destacar los siguientes:

- Otro archivo con el mismo nombre ya existía en el soporte.
- El dispositivo no tiene espacio disponible para crear otro nuevo archivo.
- El dispositivo no está operacional.
- Existe un problema de hardware que hace abortar el proceso.
- Uno ó más de los parámetros de entrada en la instrucción son erróneos.

La instrucción o acción en pseudocódigo que permite crear un archivo se codifica con la palabra **crear.**

Abrir un archivo.

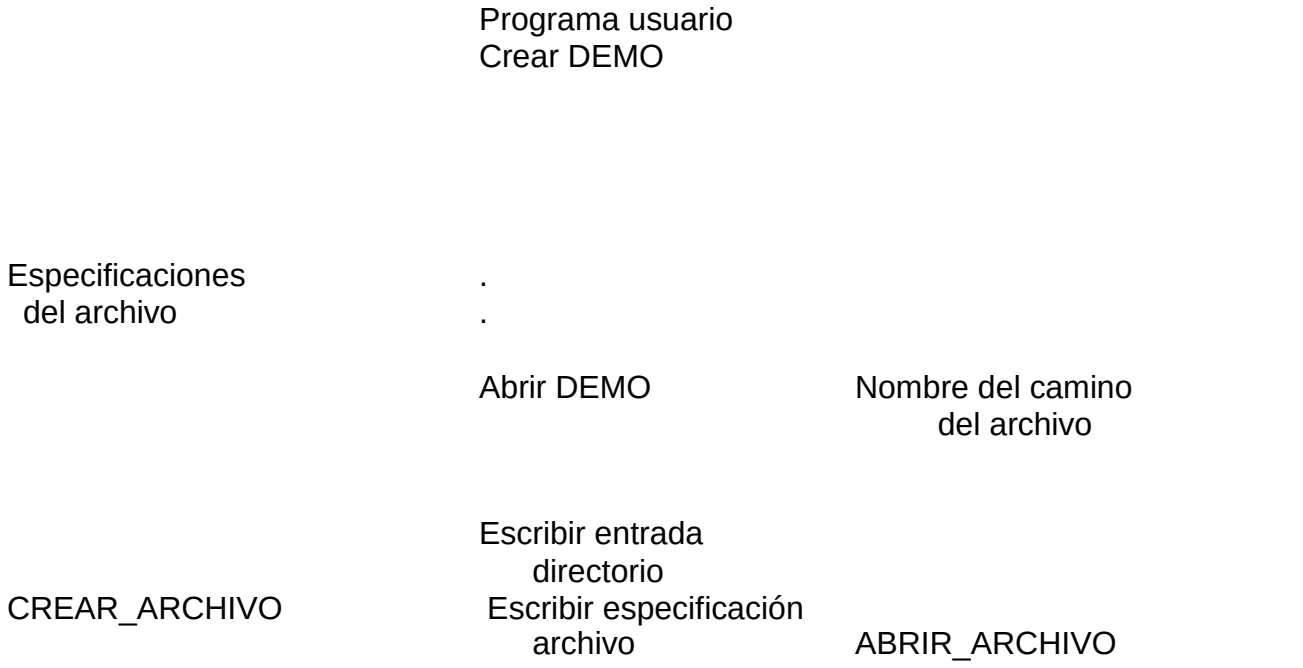
La acción de **abrir** (open) un archivo es permitir al usuario localizar y acceder a los archivos que fueron creados anteriormente.
La diferencia esencial entre una instrucción de **abrir** un archivo y una instrucción de **crear** un archivo reside en que el archivo no existe antes de utilizar **crear** y se supone que debe existir antes de utilizar **abrir.**
La información que un sistema de tratamiento de archivos requiere para abrir un archivo es diferente de las listas de información requerida para crear un archivo. La razón para ello reside en el hecho que toda la información que realmente describe el archivo se escribió en éste durante el proceso de creación del archivo. Por consiguiente, la operación **abrir-archivo** sólo necesita localizar y leer está información conocida como atributos del archivo.

La instrucción de abrir un archivo consiste en la creación de un canal que comunica a un usuario a través de un programa con el archivo correspondiente situado en un soporte.
Los parámetros que se deben incluir en una institución de apertura (abrir) son:

- Nombre del dispositivo;
- Nombre del usuario o canal de comunicación;
- Nombre del archivo.

Al ejecutar la instrucción **abrir-archivo** se pueden encontrar los siguientes errores:

- Archivo no encontrado en el dispositivo especificado (nombre de archivo o identificador de dispositivo erróneo);
- Archivo ya está en uso para alguna otra aplicación del usuario;
- Errores hardware.



La operación de abrir archivos se puede aplicar para operaciones de entrada, salida o bien entrada/salida.

abrir *nombrearchivo* para entrada

Ampliación de un archivo

El propósito de la institución **añadir-archivo** (*extend-file, append-file*) es permitir al usuario incrementar el tamaño de un archivo.

La información que necesita el sistema de gestión de archivos para incrementar el tamaño de un archivo es:

- Nombre del dispositivo y directorio donde está localizado el archivo.
- Nombre del archivo.
- Tamaño en que se incrementará el archivo.

Antes de poder realizar con éxito la operación de ampliación de un archivo se necesita comprobar previamente si existe espacio suficiente en el soporte para alojar el tamaño de la ampliación prevista.

Los *errores* más típicos que se pueden cometer en la operación de ampliación son:

- El dispositivo no tiene espacio disponible para ampliar el archivo.
- El archivo no se *abrió* previamente.

Si se examinan la mayoría de los lenguajes de programación, se comprobará que no existe casi ninguna instrucción en la sintaxis de los lenguajes para ampliar un archivo. Sin embargo, será preciso en aquellos casos en que no existían instrucciones específicas el diseño de algún programa de utilidad que realice dicha tarea.

Cerrar archivos

El propósito de la operación de cerrar un archivo es permitir al usuario cortar el acceso o detener el uso del archivo, permitiendo a otros usuarios acceder al archivo. Para ejecutar esta función, el sistema de tratamiento de archivos sólo necesita conocer el nombre del archivo que se debe cerrar, y que previamente debía estar abierto.

El formato de la instrucción es

cerrar *nombrearchivo*

BORRAR ARCHIVOS

La instrucción de **borrar-archivos** tiene como objetivo la supresión de un archivo del soporte o dispositivo. El espacio utilizado por un archivo borrado puede ser utilizado para otros archivos.

La información necesaria para eliminar un archivo es:

- Nombre del dispositivo y número del canal de comunicación;
- Nombre del archivo.

Los *errores* que se pueden producir son:

- El archivo no se pudo encontrar bien porque el nombre no es válido o porque nunca existió.
- Otros usuarios estaban actuando sobre el archivo y estaba activo.
- Se detectó un problema de hardware.

5. MANTENIMIENTO DE ARCHIVO

La operación de mantenimiento de un archivo incluye todas las operaciones que sufre un archivo durante su vida y desde su creación hasta su eliminación o borrado.
El mantenimiento de un archivo consta de dos operaciones diferentes:

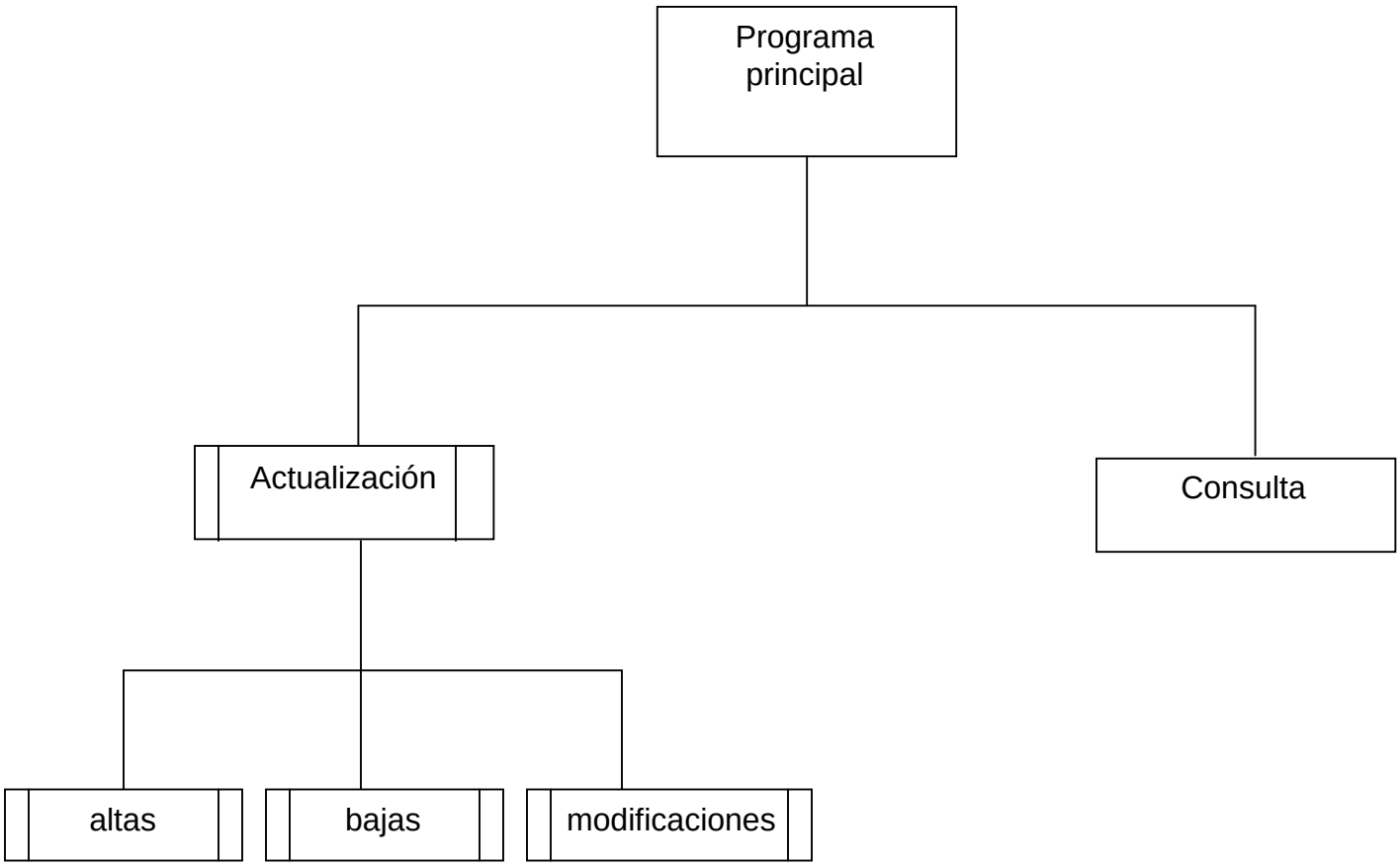
- *Actualización*;
- *Consulta*.

La actualización es la operación de eliminar o modificar los datos ya existentes, o bien introducir nuevos datos. En esencia, es la puesta al día de los datos del archivo.

Las operaciones de actualización son:

- *Altas*
- *Bajas*
- *Modificaciones*

Las operaciones de consulta tienen como finalidad obtener información total o parcial de los datos almacenados en un archivo y presentarlos en dispositivos de salida: pantalla o impresora, bien como resultados o como listados.
Todas las operaciones de mantenimiento de archivos suelen construir módulos independientes del programa principal y su diseño que realiza con subprogramas (*subrutinas* o procedimientos *específicos*).
Así los subprogramas de mantenimiento de un archivo constarán de:



ALTAS

Una operación de alta en un archivo consiste en la adición de un nuevo registro.
En un archivo de empleados, un alta consistiría en introducir los datos de un nuevo empleado. Para situar correctamente un alta se deberá conocer la posición donde se desea almacenar el registro correspondiente: al principio, en el interior o al final de un archivo.
El algoritmo del subprograma ALTAS debe contemplar la comprobación de que el registro a dar de ALTA no existe previamente.

BAJAS

Una *baja* es la acción de eliminar un registro de un archivo. La baja de un registro se puede presentar de dos formas distintas: indicación del registro específico que se desea bajar o bien visualizar los registros del archivo para que el usuario elija el registro a borrar.
La baja de un registro puede ser lógica o física. Una baja lógica supone el no borrado del registro en el archivo. Esta baja lógica se manifiesta en un determinado campo del registro con una bandera, indicador o “flag”- carácter *, $,

etc.-, o bien con la escritura o rellenado de espacios en blanco en el registro específico.
Una baja física implica el borrado y desaparición del registro de modo que se crea un nuevo archivo que no incluye al registro dado de baja.

MODIFICACIONES

Una modificación es un archivo consiste en la operación de cambiar total o parcialmente el contenido de uno de sus registros.
Esta fase es típica cuando cambia el contenido de un determinado campo de un archivo; por ejemplo, la dirección o la edad de un empleado.
La forma práctica de modificar un registro es la visualización del contenido de sus campos; para ello se debe elegir el registro o registros a modificar. El proceso consistente en la lectura del registro, modificación de su contenido y escritura, total o parcial del número.

CONSULTA

La operación de consulta tiene como fin visualizar la información contenida en el archivo, bien de un modo completo- bien de modo parcial-, examen de uno o más registros.
Las operaciones de consulta de archivo deben contemplar diversos aspectos que faciliten la posibilidad de conservación de datos. Los aspectos más interesantes a tener en cuenta son:

- Opción de visualización en pantalla o listado en impresora
- Detención de la consulta a la voluntad del usuario.
- Listado por registros o campos individuales o bien listado total del archivo (en este casó deberá existir la posibilidad de impresión de listados, con opciones de saltos de página correctos).

OPERACIONES SOBRE REGISTROS

Las operaciones de transferencia de datos a/o desde un dispositivo a la memoria central se realizan mediante las instrucciones:

leer nombre archivo, lista de entrada de datos
escribir nombre archivo, lista de salida de datos

Las operaciones de acceso a un registro y de paso de un registro a otro se realiza con las acciones **leer** y **escribir.**

6. ALGORITMOS PARA MANIPULAR ARCHIVOS SECUENCIALES

En un archivo secuencial los registros se insertan en el archivo en orden cronológico de llegada al soporte, es decir, un registro de datos se almacena inmediatamente a continuación del registro anterior.
Los archivos secuenciales termina con una marca final de archivo (FF o EOF).
Cuando se tengan que añadir registros a un archivo secuencial se añadirán en las marcas fin de archivos.
Las operaciones básicas que se permiten en un archivo secuencial son: escribir su contenido, añadir un registro al final del archivo y consultar sus registros. Las demás operaciones exigen una programación específica.
Los archivos secuenciales son los que ocupan menos memoria y son útiles cuando se desconoce a priori el tamaño de los datos y se requieren registros de longitud variable. También son muy empleados para el almacenamiento de información, cuyos contenidos sufran pocas modificaciones en el transcurso de su vida útil.

CREACION

La creación de un archivo secuencial es un proceso secuencial, ya que los registros se almacenan consecutivamente en el mismo orden en que se introducen en el archivo.
El método de creación de un archivo consiste en la ejecución de un programa adecuado que permite la entrada de datos del archivo desde el terminal. El sistema usual es el interactivo en el que el programa solicita los datos al usuario que los introduce por el teclado, hasta que se introduce una marca final del archivo (EOF o FF) que supone el final físico del archivo.
En los archivos secuenciales EOF o FF es una función lógica que toma el valor cierto si se ha alcanzado el final del archivo y falso en caso contrario.

La operación de crear un archivo tiene dos variantes:

- Crear el archivo original (1);
- Añadir datos al archivo ya creado y a continuación del ultimo registro del mismo (2)

La creación del archivo requerirá los siguientes pasos:

- Abrir el archivo;
- Leer datos del registro**;**
- Grabar registro;
- Cerrar archivo .

El algoritmo de creación con inclusión del menú de opciones es el siguiente:

```
algoritmo creación
   inicio
        {menú de opciones}
        escribir '1. creación de archivo nuevo'
        escribir '2. añadir datos al archivo'
        leer opción
        si opción = 1
            entonces
               abrir archivo nuevo para creación
            sino
               abrir archivo para añadir datos
          fin_si
          {grabación de datos en el archivo}
          mientras no se alcance el fin de archivo (EOF) hacer
          leer datos de un registro
          escribir (grabar) registro
          fin_mientras
          cerrar archivo
          fin
```

Consulta

El proceso de búsqueda o consulta de una información en un archivo de organización secuencial se debe efectuar obligatoriamente en modo secuencial. Por ejemplo, si se desea consultar la información contenida en el registro 50, se deberán leer previamente los 49 primeros registros que le preceden en orden secuencial. En el caso de un archivo de personal sí desean buscar un registro determinado correspondiente a un determinado empleado, será necesario recorrer leer todo el archivo desde el principio hasta encontrar el registro que se busca o la marca final de archivos.
Así, para el caso de un archivo de *n* registros, el número de lecturas de registros efectuadas son:

- Mínimo 1, si el registro buscado es el primero del archivo;
- Máximo *n,* si el registro buscado es el último o no existe dentro del archivo.

Por término medio, el número de lecturas necesarias para encontrar un determinado registro es:

$$\frac{n+1}{2}$$

El tiempo de acceso será influyente en las operaciones de lectura/escritura. Así, en el caso de una lista de vector de *n* elementos almacenados en memoria central puede suponer tiempos en microsegundos o nanosegundos; sin embargo, en el caso de una archivo *n* registros los tiempos de acceso son de milisegundos o fracciones/múltiples de milisegundos, lo que supone un tiempo de acceso de 1000 s 100000 veces más grande una búsqueda de información en un soporte externo que en memoria central.

La búsqueda en lugar de ser secuencial se podría realizar por el método binario, lo que reducirá considerablemente en tiempo.
El algoritmo de consulta de un archivo requerirá un diseño previo de la presentación de la estructura de registros en el dispositivo de salida de acuerdo al número y longitud de los campos.

```
algoritmo consulta-total
    inicio
        abrir archivo para lectura
        leer registro
        N ←1
        mientras registro < > FF hacer {FF, fin de archivo}
        escribir  registro
        leer registro
        N ←— N + 1
    fin_ mientras
    {número de registros de archivo, N – 1}
    escribir 'número de registro', archivo N-1}
    escribir 'número de registro ', N – 1
    cerrar archivo
fin
```

En el caso de búsqueda de un determinado registro con un campo clave *x,* el algoritmo de búsqueda se puede modificar en la siguiente forma con

```
mientras registro < > FF hacer
  si registro (campo  X) = registro (campo leído)
     entonces
       escribir 'el registro buscado existe
          fin {fin del programa}
sino
     leer registro
     N ← N+1
  fin_si
fin_mientras
escribir 'el registro buscado no existe en el archivo'
```

Actualización

La actualización de un archivo supone:

- Añadir nuevos registros (altas)
- Modificar registros ya existentes (modificaciones)
- Borrar registros (bajas)

ALTAS

La operación de dar de alta un determinado registro es similar a la operación ya descrita anteriormente de añadir datos a un archivo.
La operación de alta supone la creación nueva del archivo, ya que un archivo secuencial no admite la incorporación de nuevos registros.

```
algoritmo altas
inicio
  abrir archivo para añadir
mientras no haya más registros hacer
        leer datos del registro
        escribir (grabar) registro
fin_mientras
cerrar archivo
fin
```

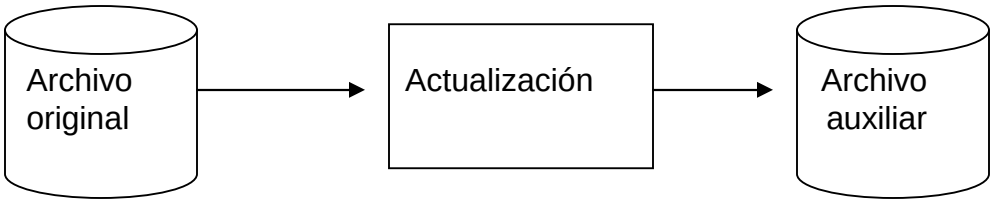
BAJAS

Existen dos métodos para dar de baja a un registro:

1. Se utiliza un archivo transitorio
2. Almacenar en un array (vector) todos los registros del archivo, señalando con un indicador o bandera (flag) el registro que se desea dar de baja.

Método 1

Se crea un segundo archivo auxiliar, también secuencial, copia del que se trata de actualizar. Se lee el archivo completo registro a registro y en función de su lectura se decide si el registro se debe de dar de baja o no.
Si el registro se va a dar de baja, se omite la escritura en el archivo auxiliar o transitorio. Si el registro no se va a dar de baja, este registro se escribe en el archivo auxiliar.
Tras terminar la lectura del archivo original, se tendrán dos archivos: original (o maestro) y auxiliar.



El proceso de bajas del archivo concluye cambiando el nombre del archivo auxiliar por el de maestro y borrando previamente el archivo maestro original.

Método 2

Este procedimiento consiste en señalar los registros que se desean dar de baja con un indicador o bandera; estos registros no se graban en el nuevo archivo secuencial que se crea sin los registros dados de baja.

```
algoritmo bajas
 inicio
        abrir archivo original (maestro) para lectura
        abrir archivo auxiliar para creación
        leer registro del maestro
        mientras registro <> FF hacer
                escribir 'Baja (S/N)'
                leer respuesta
                si respuesta=´N'
                entonces
                        escribir registro en archivo auxiliar
                fin_si
                leer registro
                fin_mientras
        cerrar archivo maestro
        cerrar archivo auxiliar
        borrar archivo maestro
        cambiar nombre del archivo auxiliar por nombre de maestro
   fin
```

MODIFICACIONES

El proceso de modificación de un registro consiste en localizar este registro, efectuar dicha modificación y a continuación rescribir el nuevo registro en el archivo.

El proceso es similar al de bajas

```
algoritmo modificación
        inicio
                abrir archivo maestro para lectura
                abrir archivo auxiliar para creación
                leer registro del maestro
                mientras registro del maestro <> FF hacer
                        escribir 'Modificar (S/N)'
                        leer respuesta
                        si respuesta=´S'
                        entonces
                        llamar_a subprograma de modificación
                        fin_si
                        escribir registro en archivo auxiliar
                        fin_mientras
                        cerrar archivo maestro
                        cerrar archivo auxiliar
                        borrar archivo maestro
        cambiar nombre del archivo auxiliar por nombre de maestro
fin
```

7. PROCESAMIENTO DE ARCHIVOS DE ACCESO DIRECTO (ALGORITMOS)

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.

Operaciones con archivos de acceso directo

Las operaciones con archivos de acceso directo son las usuales ya vistas anteriormente.

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.

```
algoritmo creación
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
```

ALTAS

Para dar de alta un registro, se debe introducir su número de orden y contenido

```
algoritmo altas
inicio
        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 en registro NR
        fin_si
     sino
          escribir érror,rango 1..ALTO'
fin_si
   hasta_que no se deseen más altas
fin
```

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.

```
mientras < > FF hacer
        leer registro R
         ir a subprograma de obtención de la 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
```

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.

```
algoritmo baja
repetir
        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
```

MODIFICACIONES

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

```
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 más modificaciones
fin_si
```

8. 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.

9. PROCESAMIENTO DE ARCHIVOS SECUENCIALES INDEXADOS

Recordando que los archivos de organización secuencial indexada contienen dos áreas: un área de datos que agrupa a los registros y un área índice que contiene los niveles de índice. Pueden contener también una zona de desbordamiento o excedentes para caso de actualizaciones.
Una de las grandes ventajas de utilizar un índice con un archivo de datos secuenciales es que la adición de registros pueden realizarse mucho más rápidamente que con un archivo datos clasificados, siempre que el índice sea pequeño para poder alojarse en memoria
El almacenamiento del área de índices en memoria, mientras el programa se ejecuta, permite encontrar los registros por clave más rápidamente con un archivo indexado que con uno clasificado, ya que la búsqueda binaria puede ejecutarse completamente en memoria.

2. MÉTODOS DE CLASIFICACIÓN Y CONSIDERACIONES DE COMPLEJIDAD.

INTRODUCCIÓN

Ordenar significa reagrupar o reorganizar un conjunto de datos u objetos en una secuencia específica. Los procesos de ordenación y búsqueda son algo frecuente en nuestras vidas. Vivimos en un mundo desarrollado, automatizado, acelerado, donde la información representa un elemento de vital importancia. La sociedad debe estar informada y por lo tanto el buscar y recuperar información es ahora una necesidad.

La operación de búsqueda para recuperar información, normalmente se efectúa sobre elementos ordenados. Lo que demuestra que, en general, donde haya objetos que deban buscarse y recuperarse, el proceso de ordenación estará presente.

Los objetos ordenados aparecen por doquier. Directorios telefónicos, registros de pacientes de un hospital, registros de huéspedes de un hotel, índices de libros de una biblioteca, son tan sólo algunos ejemplos de objetos ordenados con los cuales el ser humano se encuentra frecuentemente. Incluso y de manera informal puede señalarse que desde niño se le enseña a ser ordenado, a poner sus cosas en orden.

La ordenación es una actividad fundamental y relevante en la vida. Imagínese qué ocurriría si deseara encontrar un libro en una biblioteca que contenga 70 000 volúmenes y éstos están desordenados o registrados en los índices en el orden en el cual ellos fueron recibidos; o por ejemplo si se quiere hablar por teléfono con una persona y se encuentra que en el directorio los abonados están ordenados según su número telefónico, ascendente o descendentemente.

Formalmente se define ordenación de la siguiente manera.

Sea A una lista de N elementos:

$$A_1, A_2, A_3, \ldots\ldots\ldots\ldots, A_n$$

Ordenar significa permutar estos elementos de tal forma que los mismos queden de acuerdo con un orden preestablecido.

- Ascendente: $A_1 \leq A_2 \leq A_3 \leq \ldots\ldots \leq A_N$
- Descendente: $A_1 \geq A_2 \geq A_3 \geq \ldots\ldots \geq A_N$

En el procesamiento de datos, a los métodos de ordenación se les clasifica en dos categorías:

- Ordenación de arreglos

- Ordenación de archivos

La primera categoría recibe también el nombre de ordenación interna, ya que los elementos o componentes del arreglo se encuentran en la memoria principal de la computadora. La segunda categoría recibe también el nombre de ordenación externa, ya que los elementos se encuentran en archivos que están almacenados en dispositivos de almacenamiento secundario, como discos, cintas, etc.

Ejemplificando esta clasificación se puede mencionar que para la máquina, la ordenación interna, representa lo que para un humano significa ordenar un conjunto de tarjetas que se encuentran visibles y extendidas todas sobre una mesa.
Ahora bien, la ordenación externa, representa para la máquina lo que para un humano significa ordenar un conjunto de tarjetas que están dispuestas una debajo de otra y en donde sólo se visualiza la primera.

ORDENACION INTERNA

Los métodos de ordenación interna se explicarán con arreglos unidimensionales, pero su uso puede extenderse a otros tipos de arreglos. Por ejemplo: bidimensionales y tridimensionales, considerando el proceso de ordenación respecto a renglones y columnas en el caso de arreglos bidimensionales y renglones, columnas y paginas en el caso de arreglos tridimensionales.
También debe señalarse que se trabajará con métodos de ordenación “in situ”, es decir, métodos que no requieren de arreglos auxiliares para su ordenación, ya que éstos, además de ineficientes, son intrínsecamente de menor interés.
Los métodos de ordenación interna a su vez pueden ser clasificados en dos tipos:

- Métodos directos (n^2)
- Métodos logarítmicos (n * log n)

Los métodos directos tienen la característica de que sus programas son cortos y de fácil elaboración y comprensión, aunque son ineficientes cuando N (el número de elementos del arreglo) es medio o grande. Los métodos logarítmicos son más complejos que los métodos directos. Cierto es que requieren de menos comparaciones y movimientos para ordenar sus elementos, pero su elaboración y comprensión resulta más sofisticada y abstracta.
Debe aclararse que una buena medida de eficiencia entre los distintos métodos es el tiempo de ejecución del algoritmo y éste depende fundamentalmente del número de comparaciones y movimientos que se realicen entre elementos.
Como conclusión puede decirse que cuando N es pequeño deben utilizarse métodos directos y cuando N es medio o grande deben emplearse métodos logarítmicos.

Los métodos directos más conocidos son:

- Ordenación por intercambio
- Ordenación por inserción
- Ordenación por selección

2.1 Ordenación por intercambio directo (burbuja)

El método de intercambio directo, conocido con el nombre de la burbuja, es el más utilizado entre los estudiantes principiantes de computación, por su fácil comprensión y programación. Pero es preciso señalar que es probablemente el método más ineficiente.
El método de intercambio directo puede trabajar de dos maneras diferentes.
Llevando los elementos más pequeños hacia la parte izquierda del arreglo o bien llevando los elementos más grandes hacia la parte derecha del mismo.
La idea básica de este algoritmo consiste en comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que todos se encuentren ordenados. Se realizan (n-1) pasadas, transportando en cada una de las mismas el menor o mayor elemento (según sea el caso) a su posición ideal. Al final de las (n – 1) pasadas los elementos del arreglo estarán ordenados.

EJEMPLO

Suponer que se desea ordenar las siguientes claves del arreglo A, transportando en cada pasada el menor elemento hacia la parte izquierda del arreglo.

A: 15 67 08 16 44 27 12 35

Las comparaciones que se realizan son las siguientes:

PRIMERA PASADA
A[7] > A[8] (12>35) no hay intercambio
A[6] > A[7] (27>12) sí hay intercambio
A[5] > A[6] (44>12) sí hay intercambio
A[4] > A[5] (16>12) sí hay intercambio
A[3] > A[4] (08>12) no hay intercambio
A[2] > A[3] (67>08) sí hay intercambio
A[1] > A[2] (15>08) sí hay intercambio

Luego de la primera pasada el arreglo queda de la siguiente forma:

A: 08 15 67 12 16 44 27 35

Observe que el elemento más pequeño, en este caso 08, fue situado en la parte izquierda del arreglo.

SEGUNDA PASADA

A[7] > A[8] (27 > 35) no hay intercambio
A[6] > A[7] (44 > 27) sí hay intercambio
A[5] > A[6] (16 > 27) no hay intercambio
A[4] > A[5] (12 > 16) no hay intercambio
A[3] > A[4] (67 > 12) sí hay intercambio
A[2] > A[3] (15 > 12) sí hay intercambio

Luego de la segunda pasada el arreglo queda de la siguiente forma:

A: [08] [12] 15 67 16 27 44 35

y el segundo elemento más pequeño del arreglo, en este caso 12, fue situado en la segunda posición.

En la tabla siguiente se presenta el resultado de las restantes pasadas:

3era. Pasada:	[08]	[12]	[15]	16	67	27	35	44
4ta. Pasada:	[08]	[12]	[15]	[16]	27	67	35	44
5ta. Pasada:	[08]	[12]	[15]	[16]	[27]	35	67	44
6ta. Pasada:	[08]	[12]	[15]	[16]	[27]	[35]	44	67
7ma. Pasada:	[08]	[12]	[15]	[16]	[27]	[35]	[44]	[67]

El algoritmo de ordenación por el método de intercambio directo que transporta en cada pasada el menor elemento hacia la parte izquierda del arreglo es el siguiente:

Algoritmo Burbuja 1

BURBUJA1 (A, N)

{El algoritmo ordena los elementos del arreglo utilizando el método de la burbuja, Transporta en cada pasada el elemento más pequeño hacia la parte izquierda del arreglo. A es un arreglo de N elementos}

{ I, J y AUX son variables de tipo entero}

1. *Repetir* con I desde 2 hasta N
   1.1 *Repetir* con J desde N hasta I
      1.1.1 Si A [J – I] > A [J] *entonces*
            Hacer AUX ⟵ A[J – I], A[J – I] ⟵ A[J] y A[J] ⟵ AUX
      1.1.2 {Fin del condicional del paso 1.1.1}
   1.2 {Fin del ciclo del paso 1.1}
2. {Fin del ciclo del paso 1}

EJEMPLO

Suponer que se desea ordenar las siguientes claves del arreglo A transportando en cada pasada el mayor elemento hacia la parte derecha del arreglo.

A: 15 67 08 16 44 27 12 35

Las comparaciones que se realizan son las siguientes:

PRIMERA PASADA

A[1] > A[2] (15 > 67) No hay intercambio
A[2] > A[3] (67 > 08) Sí hay intercambio
A[3] > A[4] (67 > 16) Sí hay intercambio
A[4] > A[5] (67 > 44) Sí hay intercambio
A[5] > A[6] (67 > 27) Sí hay intercambio
A[6] > A[7] (67 > 12) Sí hay intercambio
A[7] > A[8] (67 > 35) Sí hay intercambio

A: 15 08 16 44 27 12 35 67

Observe que el elemento más grande, en este caso 67, fue situado en la última posición del arreglo.

SEGUNDA PASADA

A[1] > A[2] (15 > 08) sí hay intercambio
A[2] > A[3] (15 > 16) no hay intercambio
A[3] > A[4] (16 > 44) no hay intercambio
A[4] > A[5] (44 > 27) sí hay intercambio
A[5] > A[6] (44 > 12) sí hay intercambio
A[6] > A[7] (44 > 35) sí hay intercambio

A: 08 15 16 27 12 35 44 67

y el segundo elemento más grande del arreglo, en este caso 44, fue situado en la penúltima posición.

En la tabla siguiente se ve el resultado de las restantes pasadas:

3era. pasada:	08	15	16	12	27	35	44	67
4ta. pasada:	08	15	12	16	27	35	44	67
5ta. pasada:	08	12	15	16	27	35	44	67
6ta. pasada:	08	12	15	16	27	35	44	67
7ma. pasada:	08	12	15	16	27	35	44	67

El algoritmo de ordenación por el método de intercambio directo que transporta en cada pasada el elemento mayor hacia la parte derecha del arreglo es el siguiente:

Algoritmo Burbuja2

BURBUJA2 (A, N)

{El algoritmo ordena los elementos del arreglo. Transporta en cada pasada el elemento más grande hacia la parte derecha del arreglo. A es un arreglo de N elementos}

{ I, J y AUX son variables de tipo entero}

1. *Repetir* con I desde 1 hasta N – 1
   1.1 *Repetir* con J desde 1 hasta N – I
      1.1.1 Si A [J] > A[J + 1] *entonces*
            Hacer AUX ← A[J], A[J] ← A[J + 1] y A[J + 1] ← AUX
      1.1.2 {Fin del condicional del paso 1.1.1}
   1.2 {Fin del ciclo del paso 1.1}
2. {Fin del ciclo del paso 1}

2.1 ORDENACIÓN POR EL MÉTODO DE INTERCAMBIO DIRECTO CON SEÑAL

Este método es una modificación del método de intercambio directo. La idea central de este algoritmo consiste en utilizar una marca o señal para indicar que no se ha producido ningún intercambio en una pasada. Es decir, se comprueba si el arreglo está totalmente ordenado después de cada pasada, terminando su ejecución en caso afirmativo.
El algoritmo de ordenación por el método de la burbuja con señal es el siguiente:

Algoritmo Burbujaconseñal

BURBUJACONSEÑAL (A, N)

{El algoritmo ordena los elementos del arreglo utilizando el método de la burbuja con señal. A es un arreglo de N elementos}

{I, J y AUX son variables de tipo entero. BAND es una variable de tipo booleano}

1. Hacer I ← 1 y BAND ← FALSO
2. *Repetir* mientras (I ≤ N – 1) y (BAND = FALSO)
   Hacer BAND ← VERDADERO
   2.1 Repetir con J desde 1 hasta N – I
      2.1.1 Si A [J] > A[J + 1] *entonces*
         Hacer AUX ← A[J], A[J] ← A [J + 1],
         A[J + 1] ← AUX y BAND ← FALSO
      2.1.2 {Fin del condicional del paso 2.1.1}
   2.2 {Fin del ciclo del paso 2.1}
   Hacer I ← I + 1
3. {Fin del ciclo del paso 2}

2.1 ORDENACIÓN POR EL MÉTODO DE LA SACUDIDA (SHAKER SORT)

El método de shaker sort, más conocido en el mundo del habla hispana como el método de la sacudida, es una optimización del método de intercambio directo. La idea básica de este algoritmo consiste en mezclar las dos formas en que se puede realizar el método de la burbuja.

En este logaritmo cada pasada tiene dos etapas. En la primera etapa “de derecha a izquierda” se trasladan los elementos más pequeños hacia la parte izquierda del arreglo, almacenando en una variable la posición del último elemento intercambiado. En la según etapa “de izquierda a derecha” se trasladan los elementos más grandes hacia la parte derecha del arreglo, almacenando en otra variable la posición del último elemento intercambiado. Las sucesivas pasadas

trabajan con los componentes del arreglo comprendidos entre las posiciones almacenadas en las variables. El algoritmo termina cuando en una etapa no se producen intercambios o bien, cuando el contenido de la variable que almacena el extremo izquierdo del arreglo es mayor que el contenido de la variable que almacena el extremo derecho.

EJEMPLO.

Suponer que se desea ordenar las siguientes claves del arreglo A utilizando el método de la sacudida.

A: 15 67 08 16 44 27 12 35

PRIMERA PASADA

Primera etapa (de derecha a izquierda)

A[7] > A[8] (12 > 35)	no hay intercambio
A[6] > A[7] (27 > 12)	sí hay intercambio
A[5] > A[6] (44 > 12)	sí hay intercambio
A[4] > A[5] (16 > 12)	sí hay intercambio
A[3] > A[4] (08 > 12)	no hay intercambio
A[2] > A[3] (67 > 08)	sí hay intercambio
A[1] > A[2] (15 > 08)	sí hay intercambio

Última posición de intercambio de derecha a izquierda: 2
Luego de la primera etapa de la primera pasada, el arreglo queda de la siguiente forma:

A: 08 15 67 12 16 44 27 35

Segunda etapa (de izquierda a derecha)

A[2] > A[3] (15 > 67)	no hay intercambio
A[3] > A[4] (67 > 12)	sí hay intercambio
A[4] > A[5] (67 > 16)	sí hay intercambio
A[5] > A[6] (67 > 44)	sí hay intercambio
A[6] > A[7] (67 > 27)	sí hay intercambio
A[7] > A[8] (67 > 35)	sí hay intercambio

Última posición de intercambio de izquierda a derecha: 8
Luego de la segunda etapa de la primera pasada, el arreglo queda de la siguiente forma:

08 15 12 16 44 27 35 67

SEGUNDA PASADA

Primera etapa (de derecha a izquierda)

A[6] > A[7] (27>35) no hay intercambio
A[5] > A[6] (44>27) sí hay intercambio
A[4] > A[5] (16>27) no hay intercambio
A[3] > A[4] (12>16) no hay intercambio
A[2] > A[3] (15>12) sí hay intercambio

Última posición de intercambio de derecha a izquierda: 3

A: | 08 | | 12 | 15 16 27 44 35 | 67 |

Segunda etapa (de izquierda a derecha)

A[3] > A[4] (15>16) no hay intercambio
A[4] > A[5] (16>27) no hay intercambio
A[5] > A[6] (27>44) no hay intercambio
A[6] > A[7] (44>35) sí hay intercambio

Ultima posición de intercambio de izquierda a derecha: 7

A: | 08 | | 12 | 15 16 27 34 | 44 | | 67 |

Al realizar la primera etapa de la tercera pasada se observa que no se realizaron intercambios, por lo tanto la ejecución de algoritmo se termina.

El algoritmo de ordenación por el método de la sacudida es el siguiente:

Algoritmo Sacudida

SACUDIDA (A, N)

{El algoritmo ordena los elementos del arreglo utilizando el método de la sacudida. A es un arreglo de N elementos}

{I, IZQ, DER, K y AUX son variables de tipo entero}

1. Hacer IZQ ← 2, DER ← N y K ← N
2. Repetir hasta que IZQ > DER
2.1 Repetir con I desde DER hasta IZQ
2.1.1 Si A [I –1) > A [I] entonces

Hacer AUX ← A[I –1], A [I –1] ← A [I]
A[I] ← AUX y K ← I
2.1.2 {Fin del condicional del paso 2.1.1}
2.2 {Fin del ciclo del paso 2.1}
Hacer IZQ ← K + 1

2.3 Repetir con I desde IZQ hasta DER
2.3.1 Si A [I – 1] > A [I] entonces
Hacer AUX ← A [I – 1], A [I – 1] ← A[I],
A [I] ← AUX y K ← I
2.3.2 {Fin condicional del paso 2.3.1}
2.4 {Fin del ciclo del paso 2.3}
Hacer DER ← K - 1
3 {Fin del ciclo del paso 2}

2.2 ORDENACIÓN POR INSERCIÓN DIRECTA

El método de ordenación por inserción directa es el que generalmente utilizan los jugadores de cartas cuando ordenan éstas, de ahí que también se conozca con el nombre del método de la baraja.
La idea central de este algoritmo consiste en insertar un elemento de arreglo en la parte izquierda del mismo, que ya se encuentra ordenada. Este proceso se repite desde el segundo hasta el n-ésimo elemento.

Suponer que se desea ordenar las siguientes claves del arreglo A utilizando el método de inserción directa.

A: 15 67 08 16 44 27 12 35

Las comparaciones que se realizan son las siguientes:

PRIMERA PASADA

A[2] < A[1] (67<15) no hay intercambio

A: [15] [67] 08 16 44 27 12 35

SEGUNDA PASADA

A[3] < A[2] (08<67) sí hay intercambio
A[2] < A[1] (08<15) sí hay intercambio

A: [08] [15] [67] 16 44 27 12 35

TERCERA PASADA

A[4] < A[3] (16<67) sí hay intercambio
A[3] < A[2] (16<15) no hay intercambio

A: | 08 | | 15 | | 16 | | 67 | 44 27 12 35

Observe que una vez que se determina la posición correcta del elemento se interrumpen las comparaciones. Por ejemplo, en el caso anterior no se realizó la comparación A[2] < A[1]. En la tabla siguiente se presenta el resultado de las restantes pasadas:

4ta. pasada:	08	15	16	44	67	27	12	35
5ta. pasada:	08	15	16	27	44	67	12	35
6ta. pasada:	08	12	15	16	27	44	67	35
7ma.pasada:	08	12	15	16	27	35	44	67

El algoritmo de ordenación por el método de inserción directa es el siguiente:

INSERCCIÓN (A, N)

{El algoritmo ordena los elementos del arreglo utilizando el método de inserción directa A es un arreglo de N elementos}

{I, AUX y K son variables de tipo entero}

1. Repetir con I desde 2 hasta N
   Hacer AUX ← A[I] y K ← I – 1
1.1 Repetir mientras (AUX < A [K]) y (K > 1)
   Hacer A[K + 1] ← A[K] y K ← K - 1
1.2 {Fin del ciclo del paso 1.1}
1.3 Si A [K] ≤ AUX
   entonces
   Hacer A [K + 1] ← AUX
   si no
   Hacer A [K + 1] ← A[K] y A[K] ←AUX

1.4 {Fin del condicional del paso 1.3]}
2 {Fin del ciclo del paso 1}

2.2 ORDENACIÓN POR EL MÉTODO DE INSERCIÓN BINARIA

El método de ordenación por inserción directa puede mejorarse fácilmente. Para ello se recurre a una búsqueda binaria en lugar de una búsqueda secuencial para insertar un elemento en la parte izquierda del arreglo, que ya se encuentra ordenado. El proceso, al igual que en el método de inserción directa, se repite desde el segundo hasta el n-ésimo elemento.

Ejemplo

Suponer que se desea ordenar las siguientes claves de arreglo A utilizando el método de inserción binaria.

A: 15 67 08 16 44 27 12 35

Las comparaciones que se realizan son las siguientes:

PRIMERA PASADA

A[2] < A[1] (67<15) no hay intercambio

A: 15 67 08 16 44 27 12 35

SEGUNDA PASADA

A[3] < A[1] (08<15) sí hay intercambio

A: 08 15 67 16 44 27 12 35

TERCERA PASADA

A[4]<A[2] (16<15) no hay intercambio
A[4]<A[3] (16<67) sí hay intercambio

A: 08 15 16 67 44 27 12 35

CUARTA PASADA

A[5]<A[2] (44<15) no hay intercambio
A[5]<A[3] (44<16) no hay intercambio
A[5]<A[4] (44<67) sí hay intercambio

A: | 08 | 15 | 16 | 44 | 67 | 27 | 12 | 35 |

QUINTA PASADA

A[6]<A[3] (27<16) no hay intercambio
A[6]<A[4] (27<44) sí hay intercambio

A: | 08 | 15 | 16 | 27 | 44 | 67 | 12 | 35 |

SEXTA PASADA

A[7]<A[3] (12<16) sí hay intercambio
A[7]<A[1] (12<08) no hay intercambio
A[7]<A[2] (12<15) sí hay intercambio

A: | 08 | 12 | 15 | 16 | 27 | 44 | 67 | 35 |

SÉPTIMA PASADA

A[8]<A[4] (35<16) no hay intercambio
A[8]<A[6] (35<44) sí hay intercambio
A[8]<A[5] (35<27) no hay intercambio

A: | 08 | 12 | 15 | 16 | 27 | 35 | 44 | 67 |

El algoritmo por ordenación por el método de inserción binarias es el siguiente:

INSERCIÓNBINARIA (A, N)

{El algoritmo ordena los elementos del arreglo utilizando el método de inserción binaria.
A es un arreglo de N elementos}

{I, AUX, IZQ, DER, M y J son variables de tipo entero}
1. Repetir con I desde 2 hasta N
   Hacer AUX ← A[I], IZQ ← 1 y DER ← I - 1
Repetir mientras IZQ ≤ DER

Hacer M ⟵ parte entera ((IZQ + DER) entre 2)

1.1.1 Si AUX < A [M]
entonces
Hacer DER ⟵ M - 1
si no
Hacer IZQ ⟵ M + 1

1.1.2 {Fin del ciclo del paso 1.1.1}

1.2 {Fin del ciclo del paso 1.1}
Hacer J ⟵ I – 1

1.3 Repetir mientras J ≥ IZQ
Hacer A[J + 1] ⟵ A[J] y J ⟵ J – 1

1.4 {Fin del ciclo del paso 1.3}
Hacer A[IZQ] ⟵ AUX

2. {Fin del ciclo del paso 1}

2.3 ORDENACIÓN POR SELECCIÓN DIRECTA

El método de ordenación por selección directa es más eficiente que los métodos analizados anteriormente. Pero, aunque su comportamiento es mejor que el de aquellos y su programación es fácil y comprensible, no es recomendable utilizarlo cuando el número de elementos del arreglo es medio o grande. La idea básica de este algoritmo consiste en buscar el menor elemento del arreglo y colocarlo en la primera posición. Luego se busca el segundo elemento más pequeño del arreglo y se lo coloca en la segunda posición. El proceso continúa hasta que todos los elementos del arreglo hayan sido ordenados. El método se basa en los siguientes principios:

1. Seleccionar el menor elemento del arreglo.
2. Intercambiar dicho elemento con el primero.
3. Repetir los pasos anteriores con los (n-1), (n-2) elementos y así sucesivamente hasta que sólo quede el elemento mayor.

Ejemplo:

Suponer que se desea ordenar las siguientes claves del arreglo A utilizando el método de selección directa.

A: 15 67 08 16 44 27 12 35

Las comparaciones que se realizan son las siguientes:

PRIMERA PASADA

Se realiza la siguiente asignación: MENOR ← A [1] (15)

(MENOR < A [2]) (15< 67) sí se cumple la condición
(MENOR < A [3]) (15< 08) no se cumple la condición

MENOR ← A [3] (08)

(MENOR < A [4]) (08<16) sí se cumple la condición
(MENOR < A [5]) (08< 44) sí se cumple la condición
(MENOR < A [6]) (08< 27) sí se cumple la condición
(MENOR < A [7]) (08< 12) sí se cumple la condición
(MENOR < A [8]) (08< 35) sí se cumple la condición

Luego de la primera pasada, el arreglo queda de la siguiente forma:

A: | 08 | 67 15 16 44 27 12 35

Observe que el menor elemento del arreglo, A [3] (08), se intercambio con el primer elemento, A [1] (15), realizando solamente un movimiento.

SEGUNDA PASADA

Se realiza la siguiente asignación: MENOR ← A [2] (67)

(MENOR < A [3]) (67<15) no se cumple la condición

MENOR ← A [3] (15)

(MENOR < A [4]) (15<16) sí se cumple la condición
(MENOR < A [5]) (15<44) sí se cumple la condición
(MENOR < A [6]) (15<27) sí se cumple la condición
(MENOR < A [7]) (15<12) no se cumple la condición

MENOR ← A [7] (12)

(MENOR < A [8]) (12<35) sí se cumple la condición

A: | 08 | | 12 | 15 16 44 27 67 35

Observe que el segundo elemento más pequeño del arreglo, A [7] (12), se intercambio con el segundo elemento, A [2] (67).

En la tabla siguiente se localiza el resultado de las restantes pasadas:

Tabla

3era pasada:	08	12	15	16	44	27	67	35
4ta pasada:	08	12	15	16	44	27	67	35
5ta pasada:	08	12	15	16	27	44	67	35
6ta pasada:	08	12	15	16	27	35	67	35
7ma pasada:	08	12	15	16	27	35	44	67

El algoritmo de ordenación por el método de selección directa es el siguiente:

SELECCIÓN (A, N)

{El algoritmo ordena los elementos del arreglo utilizando el método de selección directa.
A es un arreglo de N elementos}

{ I, MENOR, K y J son variables de tipo entero}

1. Repetir con I desde 1 hasta N – 1
   Hacer MENOR ⟵ A[I] y K ⟵ I
1.1 Repetir con J desde I + 1 hasta N
1.1.1 Si A[J] < MENOR entonces
   Hacer MENOR ⟵ A[J] y K ⟵ J
1.1.2 {Fin de ciclo del paso 1.1.1}
1.2 {Fin del ciclo del paso 1.1}
   Hacer A [K] ⟵ A [I] y A [I] ⟵ MENOR
2. [Fin del ciclo del paso 1]

2.4 Método Shell

El método Shell es una versión mejorada del método de inserción directa. Recibe ese nombre en honor a su autor Donald L. Shell quién lo propuso en 1959. Ese método también se conoce con el nombre de inserción con incrementos decrecientes.
En el método de ordenación por inserción directa cada elemento se compara para su ubicación correcta en el arreglo, con los elementos que se encuentran en la parte izquierda del mismo. Sin el elemento a insertar es más pequeño que el grupo de elementos que se encuentran a su izquierda, es necesario efectuar entonces varias comparaciones antes de su ubicación.

Shell propone que las comparaciones entre elementos se efectúen con saltos de mayor tamaño pero con incrementos decrecientes, así los elementos quedarán ordenados en el arreglo más rápidamente. Para comprender mejor este algoritmo se analizará el siguiente caso.
Considere un arreglo que contenga 16 elementos. En primer lugar se dividirán los elementos del arreglo con 8 grupos teniendo en cuenta los elementos que se encuentran a 8 posiciones de distancia entre sí y se ordenarán por separado.
Quedará en el primer grupo los elementos (A[1], A [9]); en el segundo (A[2], A[10]); en el tercero, (A[3], A[11]), y así sucesivamente. Después de este primer paso se dividirán los elementos del arreglo en cuatro grupos, teniendo en cuenta ahora los elementos que se encuentren a cuatro posiciones de distancia entre sí y se los ordenará por separado. Quedará en el primer grupo los elementos (A[1], A[5], A[9], A[13]);en el segundo (A[2], A [6], A[10], A[14]), y así sucesivamente. En el tercer paso se dividirán los elementos del arreglo en grupos, tomando en cuenta los elementos que se encuentran ahora a dos posiciones de distancia entre sí y nuevamente se les ordenará por separado. En el primer grupo quedará (A[1], A[3], A[5], A[7], A[9], A[11], A[13], A[15] y el segundo grupo (A[2], A[4], A[6], A[8], A[10], A[12], A[14], A[16].

Finalmente se agruparán los elementos de manera normal, es decir, de uno en uno.

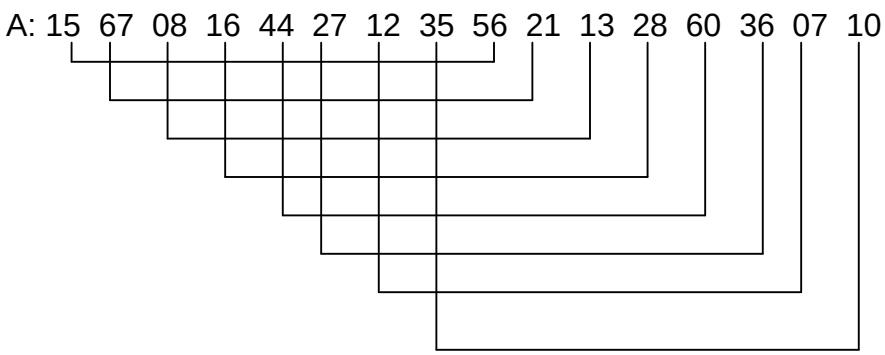
EJEMPLO

Suponer que se desean ordenar los elementos que se encuentra en el arreglo A utilizando el método del Shell.

A: 15 67 08 16 44 27 12 35 56 21 13 28 60 36 07 10

PRIMERA PASADA

Se dividen los elementos en 8 grupos:

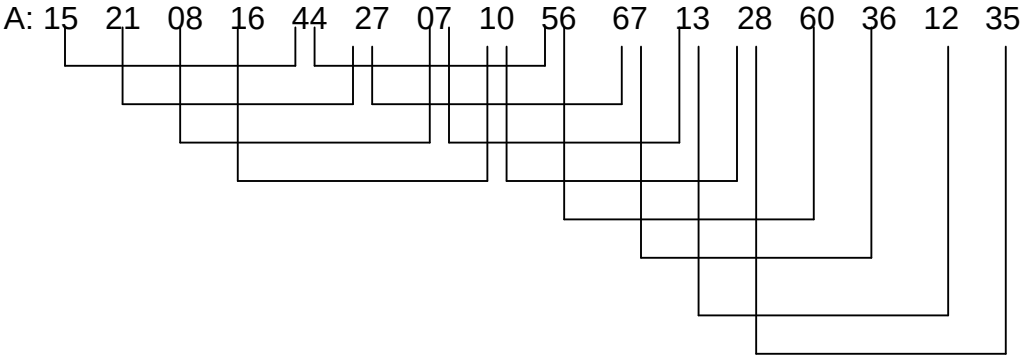


La ordenación produce:

A: 15 21 08 16 44 27 07 10 56 67 13 28 60 36 12 35

SEGUNDA PASADA

Se dividen los elementos en 4 grupos:



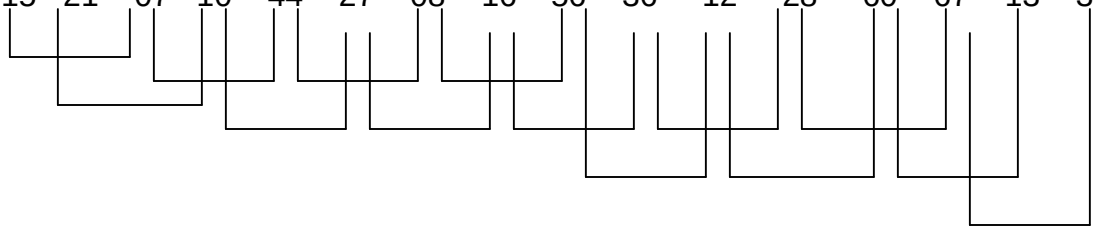
La ordenación produce:

A: 15 21 07 10 44 27 08 16 56 36 12 28 60 67 13 35

TERCERA PASADA:

Se dividen los elementos en 2 grupos:

A: 15 21 07 10 44 27 08 16 56 36 12 28 60 67 13 35



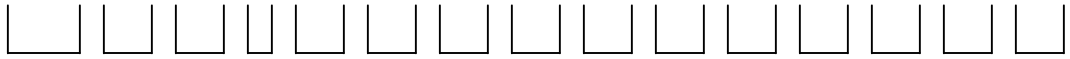
La ordenación produce:

A: 07 10 08 16 12 21 13 27 15 28 44 35 56 36 60 67

CUARTA PASADA

Divida los elementos en un solo grupo:

A: 07 10 08 16 12 21 13 27 15 28 44 35 56 36 60 67



La ordenación produce:

A: 07 08 10 12 13 15 16 21 27 28 35 36 44 56 60 67

A continuación se presenta el algoritmo por el método de Shell.

SHELL (A, N)

{El algoritmo ordena los elementos del arreglo utilizando el método de Shell.
A es un arreglo de N elementos}

{INT, I y AUX son variables de tipo entero. BAND es una variable de tipo booleano}

1. Hacer INT ← N + 1
2. Repetir mientras (INT > 1)
   Hacer INT ← parte entera (INT entre 2) y BAND ← VERDADERO

2.1 Repetir mientras (BAND = VERDADERO)

Hacer BAND ← FALSO e I← 1
2.1.1 Repetir mientras ((I + INT) ≤ N)
2.1.1.1 Si A[I] > A [I + INT] entonces
Hacer AUX ←A [I], A[I] ←A [A + INT],
A [I + INT] ←AUX Y BAND← VERDADERO
2.1.1.2 {Fin del condicional del paso 2.1.1.1}
Hacer I ← I + 1
2.1.2 {Fin del ciclo del paso 2.1.1}
2.2 {Fin del ciclo del paso 2.1}
3. {Fin del ciclo del paso 2}

2.5 Ordenación por el método quicksort

El método de ordenación **quiscksort** es actualmente el más eficiente y veloz de los métodos de ordenación interna. Es también conocido con el nombre de método rápido y de ordenación por partición, en el mundo de habla hispana. Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de Quicksort (rápido) por la velocidad con que ordena los elementos del arreglo. Su autor C. A. Hoare lo bautizo así:
La idea central de este algoritmo consiste en lo siguiente:

1. Se toma un elemento X de una posición cualquiera del arreglo.
2. Se trata de ubicar a X en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentren a su izquierda sean menores o iguales a X y todos los elementos que se encuentren a su derecha sean mayores o iguales a X.
3. Se repiten los pasos anteriores pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición correcta de X en el arreglo.
4. El proceso termina cuando todos los elementos se encuentran en su posición correcta en el arreglo.

Debe seleccionarse entonces un elemento X cualquiera. En este caso se seleccionará A[1]. Se empieza a recorrer el arreglo de derecha a izquierda comparando si los elementos son mayores o iguales a X. Si un elemento no cumple con esta condición, se intercambian los mismos y almacenamos en una variable la posición del elemento intercambiado (acotamos el arreglo por la derecha). Se inicia nuevamente el recorrido pero ahora de izquierda a derecha, comparando si los elementos son menores o iguales a X. Si un elemento no cumple con esta condición, entonces se intercambian los mismos y se almacenan en otra variable la posición del elemento intercambiado (se acota el arreglo por la izquierda). Se repiten los pasos anteriores hasta que el elemento X encuentra su posición correcta en el arreglo.
EJEMPLO

Suponer que se desea ordenar los elementos que se encuentra en el arreglo A utilizando el método quicksort.

A: 15 67 08 16 44 27 12 35

Se selecciona A [1], por lo tanto x ← 15.

Las comparaciones que se realizan son las siguientes:

PRIMERA PASADA

Recorrido de derecha a izquierda

A[8] ≥X	(35 ≥ 15)	no hay intercambio
A[7] ≥X	(12 ≥ 15)	sí hay intercambio

A: 12 67 08 16 44 27 15 35

Recorrido de izquierda a derecha.

A[2] ≤ X	(67≤15)	sí hay intercambio

A: 12 15 08 16 44 27 67 35

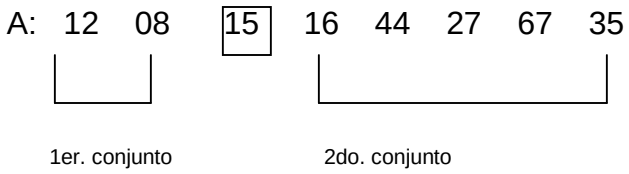
SEGUNDA PASADA

Recorrido de derecha a izquierda.

A[6] ≥ X	(27 ≥ 15)	no hay intercambio
A[5] ≥ X	(44 ≥15)	no hay intercambio
A[4] ≥ X	(16 ≥15)	no hay intercambio
A[3] ≥ X	(08 ≥ 15)	sí hay intercambio

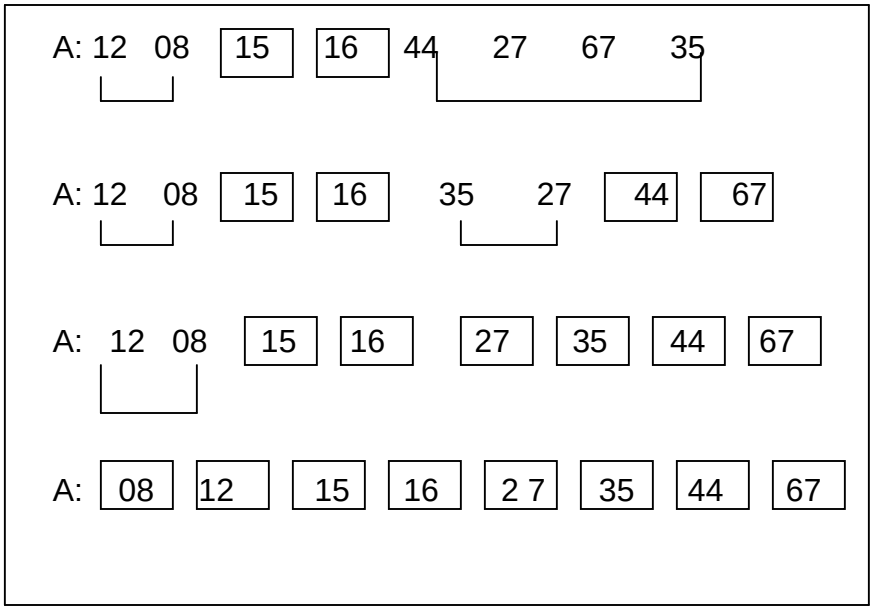
A: 12 08 15 16 44 27 67 35

Como el recorrido de izquierda a derecha debería iniciarse en la misma posición donde se encuentra el elemento X, el proceso se termina ya que el elemento X se encuentra en la posición correcta.



Observe que los elementos que forman parte del primer conjunto son menores o iguales a X, y los que forman parte del segundo conjunto son mayores o iguales a X. Este proceso de particionamiento aplicado para localizar la posición correcta de un elemento X en el arreglo se repite cada vez que queden conjuntos formados por dos o más elementos. Se puede realizar de manera iterativa o recursiva. En la tabla siguiente se presenta la ubicación del resto de los elementos en el arreglo.

Tabla



El algoritmo de ordenación por el método quicksort en su aplicación recursiva es el siguiente:

RAPIDORRECURSIVO (A, N)

{El algoritmo ordena los elementos del arreglo utilizando el método rápido, de manera recursiva A es un arreglo de N elementos}

1. Llamar al algoritmo REDUCERRECURSIVO con 1 y N

El algoritmo **rápidorrecursivo** requiere para su funcionamiento de otro algoritmo, que se presenta a continuación.

Algoritmo Reducerrecursivo.

RAPIDORRECURSIVO (INI, FIN)

{INI y FIN representan las posiciones del extremo izquierdo y derecho respectivamente, del conjunto de elementos a ordenar}

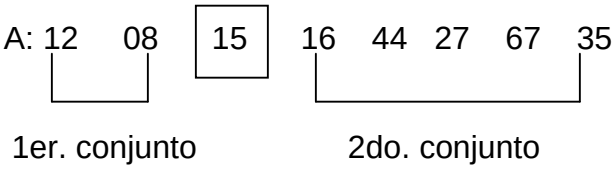
{IZQ, DER, POS y AUX son variables de tipo entero, BAND es una variable de tipo booleano}

1. Hacer IZQ ← INI, DER ← FIN, POS ← INI y BAND ← VERDADERO
2. Repetir mientras (BAND = VERDADERO)
   Hacer BAND ← FALSO

2.1 Repetir mientras (A[POS] $\leq$ A[DER]) y (POS $\neq$ DER)
   Hacer DER ← DER – 1
2.2 {Fin del ciclo del paso 2.1}
2.3 Si POS $\neq$ DER entonces
   Hacer AUX ← A[POS], A[POS] ← A[DER],
   A[DER] ← AUX y POS ← DER
2.3.1 Repetir mientras (A[POS] $\geq$ A[IZQ]) y (POS $\neq$ IZQ)
   Hacer IZQ ← IZQ + 1
2.3.2 {Fin del ciclo del paso 2.3.1}
2.3.3 Si POS $\neq$ IZQ entonces
   Hacer BAND ← VERDADERO, AUX, ← A[POS],
   A[POS] ← A[IZQ], A[IZQ] ← AUX y POS ← IZQ
2.3.4 {Fin del condicional del paso 2.3.3}
2.4 {Fin del condicional del paso 2.3}
3 {Fin del ciclo del paso 2}
4 Si (POS – 1) > INI entonces
   Regresar a REDUCERRECURSIVO con INI y (POS - 1) {Llamada recursiva}
5 {Fin del condicional del paso 4}
6 Si FIN > (POS + 1) entonces
   Regresar a REDUCERRECURSIVO con (POS +1) y FIN {Llamada recursiva}
7 {Fin del condicional del paso 6}

Aún cuando el algoritmo del quicksort presentando resulte claro, es posible aumentar su velocidad de ejecución eliminando las llamadas recursivas. La recursión es un instrumento muy poderoso, pero la eficiencia de ejecución es un factor muy importante en un proceso de ordenamiento que es necesario cuidar y administrar muy bien. Estas llamadas recursivas pueden sustituirse utilizando pilas, dando lugar a la iteractividad.

Considere el arreglo A del ejemplo Luego de la primera partición. A queda de la siguiente manera:



Deben almacenarse en las pilas los índice de los dos conjuntos de datos que falta tratar. Se utilizarán dos pilas, PILAMENOR y PILAMAYOR. En la primera, se almacenará el extremo izquierdo y en la otra se almacenará el extremo derecho de los conjuntos de los datos que falta tratar. En la figura se puede observar el estado de las pilas, luego de cargar los extremos de los dos conjuntos que falta de tratar.

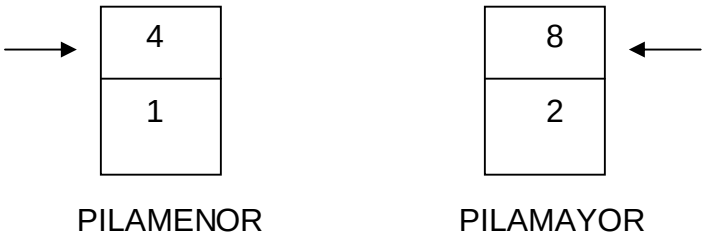


FIGURA: Pilas para sustituir la recursividad.

Los índices del primer conjunto quedaron almacenados en la primera posición de PILAMENOR y PILAMAYOR respectivamente. La posición del extremo izquierdo del primer conjunto (1) en PILAMENOR y la posición de extremo derecho del mismo conjunto (2) en PILAMAYOR. Las posiciones de los extremos izquierdo y derecho del segundo conjunto (4 y 8) fueron almacenados en la cima de PILAMENOR y PILAMAYOR, respectivamente.

RAPIDOITERATIVO (A, N)

{El algoritmo ordena los elementos de un arreglo utilizando el método rápido, de manera iterativa. A es un arreglo de N elementos}
{TOP, INI, FIN y POS son variables de tipo entero.
PILAMENOR y PILAMAYOR son arreglos unidimensionales}
1. Hacer TOP ← 1, PILAMENOR [TOP] ← 1 y PILAMAYOR [TOP] ← N
2. Repetir mientras (TOP > 0)
Hacer INI ← PILAMENOR [TOP], FIN ← PILAMAYOR [TOP]
y TOP ← TOP – 1
Llamar al algoritmo REDUCEITERATIVO con INI, FIN y POS.
2.1 Si INI < (POS – 1) entonces
Hacer TOP ← TOP + 1, PILAMENOR [TOP] ← INI y
PILAMAYOR (TOP) ← POS – 1
2.2 {Fin del condicional del paso 2.1}
2.3 Si FIN > (POS + 1) entonces
Hacer TOP ← TOP + 1, PILAMENOR [TOP] ← POS + 1
y PILAMAYOR[TOP] ← FIN

2.4 { Fin del condicional del paso 2.3}

3. {Fin del ciclo del paso 2}

2.7 ANÁLISIS COMPARATIVO DE LAS COMPLEJIDADES DE LOS DISTINTOS METODOS DE ORDENAMIENTO.

METODO DE INTERCAMBIO DIRECTO

El número de comparaciones en el método de la burbuja es fácilmente contabilizable.
En la primera pasada realizamos (n–1) comparaciones, en la segunda pasada (n–2) comparaciones, en la tercera pasada (n–3) comparaciones y así sucesivamente hasta llegar a 2 y 1 comparaciones entre claves, siendo n el número de elementos del arreglo.

Por lo tanto:

$$C = (n-1)\ (n-2) + \ldots + 2 + 1 = n * (n-1)/2$$

que es igual a:

$$C = n^2 – n/\ 2$$

Como ya se mencionó, se hace uso del principio de inducción matemática para desarrollar ciertas fórmulas.

Respecto al número de movimientos, éstos dependen de sí el arreglo está ordenado, desordenado o en orden inverso. Los movimientos para cada uno de estos casos son:

$M_{min}= 0$ $M_{med}= 0.75*(n^2 - n)$ $M_{máx}= 1.5*(n^2 - n)$

Así por ejemplo si tiene que ordenarse un arreglo que contiene 500 elementos, se efectuará:

a) Si el arreglo se encuentra ordenado:

- 124 750 comparaciones
- 0 movimientos

b) Si los elementos del arreglo se encuentran dispuestos en forma aleatoria:

- 124750 comparaciones
- 187 125 elementos

c) Si los elementos del arreglo se encuentran en orden inverso:

- 124 750 comparaciones
- 324 250 elementos

Ahora bien, el tiempo necesario para ejecutar el algoritmo de la burbuja es proporcional a n^2, $O(n^2)$, donde n es el número de elementos del arreglo.

ANÁLISIS DEL METODO DE LA SACUDIDA

El análisis del método de la sacudida y en general el de los métodos mejorados y logarítmicos son muy complejos. Para el análisis de este método es necesario tener en cuenta tres factores que afectan directamente al tiempo de ejecución del algoritmo: las comparaciones entre las claves, los intercambios entre las mismas y las pasadas que se realizan. Encontrar fórmulas que permitan calcular cada uno de estos factores es una tarea muy difícil de realizar.
Los estudios que se han efectuado sobre el método de la sacudida demuestran que en el mismo, sólo pueden reducirse las dobles comparaciones entre claves; pero debe recordarse que la operación de intercambio es una tarea más complicada y costosa que la de comparación. Por lo tanto, es posible afirmar que las hábiles mejoras realizadas sobre el método de intercambio directo sólo producen resultados apreciables si el arreglo está parcialmente desordenado (lo cual resulta difícil saber de antemano), pero si el arreglo está desordenado el método se comporta, incluso, peor que otros métodos directos como los de inserción y selección.

ANÁLISIS DEL MÉTODO DE INSERCIÓN DIRECTA

El número mínimo de comparaciones y movimientos entre claves se produce cuando los elementos del arreglo ya están ordenados. Analícese el siguiente caso

EJEMPLO

Sea A un arreglo formado por los siguientes elementos:

A: 15 20 45 52 86

Las comparaciones que se realizan son las siguientes:

PRIMERA PASADA

A[2] < A[1] (20<15) no hay intercambio

SEGUNDA PASADA

A[3] < A[2] (45<20) no hay intercambio

TERCERA PASADA

A[4] < A[3] (52<45) no hay intercambio

CUARTA PASADA

A[5] < A[4] (86<52) no hay intercambio

Luego de las comparaciones realizadas, el arreglo queda de la siguiente forma:

A: | 15 | 20 | 45 | 52 | 86 |

Observe que para este ejemplo se efectuaron 4 comparaciones. En general podemos afirmar que si el arreglo se encuentra ordenado se efectúan como máximo n-1 comparaciones y 0 movimientos entre elementos.

$$C_{min} = n - 1$$

El número máximo de comparaciones y movimientos entre elementos se produce cuando los elementos del arreglo están en orden inverso.

EJEMPLO

Sea A un arreglo formado por los siguientes elementos:

A: 86 52 45 20 15

Las comparaciones que se realizan son las siguientes:

PRIMERA PASADA

A[2] < A[1] (52<86) sí hay intercambio

SEGUNDA PASADA

A[3] < A[2] (45<86) sí hay intercambio
A[2] < A[1] (45<52) sí hay intercambio

TERCERA PASADA

A[4] < A[3] (20<86) sí hay intercambio
A[3] < A[2] (20<52) sí hay intercambio
A[2] < A[1] (20<45) sí hay intercambio

CUARTA PASADA

A[5] < A[4] (15<86) sí hay intercambio
A[4] < A[3] (15<52) sí hay intercambio
A[3] < A[2] (15<45) sí hay intercambio
A[2] < A[1] (15<20) sí hay intercambio

Observe que en la primera pasada se realizó una comparación; en la segunda pasada se llevaron a cabo dos comparaciones; en la tercera pasada, tres comparaciones, y así sucesivamente hasta n – 1 comparaciones entre elementos. Por lo tanto:

$$C_{máx} = 1 + 2 + 3 + ... + (n-1) = n^* (n - 1)/2$$

que es igual a:

$$C_{máx} = \frac{(n^2 - n)}{2}$$

Ahora bien, el número de comparaciones promedio, que es cuando los elementos aparecen en el arreglo en forma aleatoria, puede ser calculado mediante la suma de las comparaciones mínimas y máximas dividida entre 2.

Por lo tanto:

$$C_{med} = \frac{[(n-1) + \frac{(n^2 - n)}{2}]}{2}$$

al hacer la operación, queda:

$$C_{med} = \frac{(n^2 + n - 2)}{4}$$

Respecto al número de movimientos, Knuth obtiene los siguientes:

$$M_{min} = 0$$

$$M_{med} = \frac{(n^2 - n)}{4}$$

$$M_{máx} = \frac{(n^2 - n)}{2}$$

Así por ejemplo si se tiene que ordenar un arreglo que contiene 500 elementos:

a) Si el arreglo se encuentra ordenado:

- 499 comparaciones
- 0 movimientos

b) Si los elementos del arreglo se encuentran dispuestos en forma aleatoria se realizarán:

- 62 624 comparaciones, en promedio
- 62 375 movimientos, en promedio

c) Si los elementos del arreglo se encuentran en orden inverso serán necesarias:

- 124 750 comparaciones.
- 124 750 movimientos.

Ahora bien, el tiempo necesario para ejecutar el algoritmo de inserción directa es proporcional a n^2, $O(n^2)$, donde n es el número de elementos del arreglo.
A pesar de ser un método ineficiente y recomendable sólo cuando n es pequeño, el método de inserción directa se comporta mejor que los métodos de intercambio directo analizados anteriormente.

ANÁLISIS DEL METODO DE SELECCIÓN DIRECTA

El análisis del método de selección directa es relativamente simple. Debe tenerse en cuenta que el número de comparaciones entre elementos es independiente de la disposición inicial de los mismos en el arreglo. En la primera pasada se realiza (n-1) comparaciones, en la segunda pasada (n-2) comparaciones y así sucesivamente hasta 2 y 1 comparaciones, en la penúltima y última pasada respectivamente. Por lo tanto:

$$C= (n-1) + (n-2) +.... + 2+1= \frac{n^* (n -1)}{2}$$

que es igual a:

$$C = \frac{n^2 – n}{2}$$

Respecto al número de intercambios, siempre será n-1 a excepción de que se tenga incorporado en el algoritmo alguna técnica para prevenir el intercambio de un elemento consigo mismo. Por lo tanto:

$$M = n -1$$

Así, por ejemplo, sí se tiene que ordenar un arreglo que contiene 500 elementos, se efectuarán 124 750 comparaciones y 499 movimientos.

El tiempo de ejecución del algoritmo es proporcional a n^2, $O(n^2)$, aun cuando es mas rápido que los métodos presentados con anterioridad.

En la tabla pueden observarse los números de comparaciones y movimientos necesarios para ordenar un arreglo con los tres métodos directos analizados. Las columnas nuevamente indican si los elementos del arreglo se encuentran en forma ordenada, desordenada o en orden inverso. Observe que estas columnas se encuentran divididas a su vez en dos subcolumnas. La subcolumna izquierda representa un arreglo de 500 elementos y la subcolumna derecha representa un arreglo de 1000 elementos.

Es fácil observar que el método de selección directa es mejor y sólo es superado por el método de inserción directa cuando los elementos del arreglo ya se encuentran ordenados. El peor método sin duda es el de intercambio directo.

Tabla

		ORDENADA		DESORDENADA		ORDEN INVERSO	
INTERCAMBIO	C	124 750	499 500	124 750	499 500	124750	499 500
DIRECTO	M	0	0	187 125	749 250	374250	1489 500
INSERCIÓN	C	499	999	62 624	250 249	124 750	499 500
DIRECTA	M	0	0	62 375	249 750	124 750	499 500
SELECCIÓN	C	124 750	499 500	124 750	499 500	124 750	499 500
DIRECTA	M	499	999	999	999		499

ANÁLISIS DE EFICIENCIA DEL MÉTODO DE SHELL

El análisis de eficiencia del método del Shell es un problema muy complicado y aún no resuelto. Nadie ha sido capaz de establecer hasta el momento la mejor secuencia de incrementos cuando n es grande. Cabe recordar que cada vez que se propone una secuencia de intervalos es necesario "correr" el algoritmo para analizar el tiempo de ejecución del mismo.

El tiempo de ejecución del algoritmo es del orden de n* (log n)2. Unas pruebas exhaustivas realizadas para obtener la mejor secuencia de intervalos cuando el número de elementos del arreglo es igual a 8 arrojaron como resultado que la mejor secuencia corresponde a un intervalo de 1, que no es más que el método de inserción directa estudiado previamente. Estas pruebas también determinaron que el menor de movimientos se registraba con la secuencia 3,2,1.

Cabe aclarar que las pruebas exhaustivas corresponden el análisis de (8!) posibilidades, es decir, 40 320 casos diferentes.

En la tabla siguiente se muestran los 10 mejores secuencias obtenidas al evaluar las 40 320 posibilidades de secuencias que se presentan cuando se tiene un arreglo de 8 elementos.

Tabla

POSICIÓN	SECUENCIA
1	1
2	6 1
3	5 1
4	7 1
5	4 1
6	3 1

7		2	1
8	5	3	1
9	4	2	1
10	3	2	1

Para concluir con el análisis de eficiencia del método de Shell, se mencionará que estudios realizados muestran que las mejores secuencias para valores de N comprendidos entre 100 y 60 000 son las presentes en la tabla.

Tabla

SECUENCIAS	1, 3, 5, 9,...,$2^k + 1$ 1, 3, 7, 15,... $2^k - 1$ 1, 3, 5, 11,... $(2^k \pm 1)/3$ 1, 4, 13, 40,...($3^k - 1)/2$

Donde k = 0, 1, 2, 3,...

Por último y para clasificar aún más los conceptos vertidos sobre el método de Shell se incluye un segundo ejemplo.

EJEMPLO

Suponer que se desea ordenar las siguientes claves del arreglo A utilizando el metodo de Shell. La secuencia de intervalos que se utilizara corresponde a la formula ($2^k - 1$).

A: 15 67 08 16 44 27 12 35 56 21 13 28 60 36 07 10

Los resultados parciales de cada pasada así como el resultado final, se pueden observar en la tabla.

Tabla

PAS	ARREGLO A	INT
1	15 67 08 16 44 27 12 35 56 21 13 28 60 36 07 10	15
2	10 67 08 16 44 27 12 35 56 21 13 28 60 36 07 15	07
3	07 15 08 13 28 27 12 10 56 21 16 44 60 36 35 67	03
4	07 10 08 12 15 27 13 16 35 21 28 44 60 36 56 67	01
	07 08 10 12 13 15 16 21 27 28 35 36 44 56 60 67	

Donde PAS representa el número de pasada e INT representa el intervalo en el cual se está trabajando.

ANÁLISIS DE EFICIENCIA DEL METODO QUICKSORT

El método rápido es el método más rápido de ordenación interna que existe en la actualidad, sorprendentemente, ya que es una optimización del método de intercambio directo. Diversos estudios realizados sobre el comportamiento del mismo demuestran que si se escoge en cada pasada el elemento que ocupa la posición central del conjunto de datos a analizar, el número de pasadas necesarias para ordenarlo es del orden de log n. Respecto al número de comparaciones si el tamaño del arreglo es una potencia de dos, en la primera pasada realizará (n-1) comparaciones, en la segunda pasada realizará (n-1)/ 2 comparaciones, pero en dos conjuntos diferentes, en la tercera pasada realizará (n-1)/4 comparaciones pero en cuatro conjuntos diferentes y así sucesivamente. Por lo tanto:

$$C= (n-1) +2 * (n-1)/2 + 4 * (n-1)/4 + . . . + (n-1) * (n-1)/(n-1)$$

Lo cual es lo mismo que:

$$C = (n-1) + (n-1) + (n-1) + . . . + (n-1)$$

Si se considera a cada uno de los componentes de la sumatoria como un término si el número de términos de la sumatoria es igual a m, entonces se tiene que:

$$C = (n-1) * \log n$$

Sin embargo encontrar elemento que ocupe la posición central del conjunto de datos que se van a realizar es una tarea difícil, ya que hay 1/n posibilidades de lograrlo.
Además, el rendimiento medio del método es aproximadamente (2*1n 2) inferior al caso óptimo. El elemento X se selecciona arbitrariamente o bien entre una muestra relativamente pequeña de elementos del arreglo. El peor caso que ocurre cuando los elementos del arreglo ya se encuentran ordenados o bien cuando los mismos se encuentran en orden inverso.

Por ejemplo, suponer que se tiene que ordenar el siguiente arreglo:

A: 08 12 15 16 27 35 44 67

Si se escoge arbitrariamente el primer elemento (08), entonces se particionará el arreglo en dos mitades, una de 0 y otra de (n-1) elementos.

08 12 15 16 27 35 44 67

Si se continúa con el proceso de ordenación y se escoge nuevamente el primer elemento (12) del conjunto de datos que se analizarán entonces se particionará el arreglo en dos nuevos conjuntos, nuevamente uno de 0 y otro e (n-2) elementos.
Por lo tanto el numero de comparaciones que se realizan será:

$$C_{max} = n + (n-1) + (n-2) + \ldots + 2 = n * (n+1)/2 - 1$$

Que es igual a:

$$C_{máx} = n^2 + n / 2 - 1$$

Cómo conclusión podemos afirmar que el tiempo promedio de ejecución del algoritmo es proporcional a (n* log n), O (n * log n) en el peor caso el tiempo de ejecución es proporcional a n^2, O (n^2).

3. METODOS DE BÚSQUEDA

La operación de búsqueda permite recuperar datos previamente almacenados. El resultado que puede arrojar esta operación es éxito, si se encuentra el elemento buscado, o fracaso, en otras circunstancias.
La búsqueda es una actividad relevante en la vida, el mundo en el que hoy se vive es desarrollo, automatizado; donde la información representa un elemento de vital importancia. Es necesario estar informado y, por lo tanto, buscar y recuperar información. Se hacen tareas esenciales, se buscan números telefónicos en un directorio, ofertas laborales en un periódico, libros de una biblioteca, etc.
En los ejemplos citados, la búsqueda se realiza sobre elementos que, generalmente, están ordenados. Los directorios telefónicos están ordenados alfabéticamente, las ofertas laborales están ordenadas por el tipo de trabajo y los libros de una biblioteca están clasificados por tema o por autor. Sin embargo, puede suceder que la búsqueda se realice sobre una colección de elementos no ordenados; por ejemplo, cuando se busca la localización de una cuidad dentro de un mapa.
Puede concluirse, entonces, que la operación de búsqueda puede llevarse a cabo sobre elementos ordenados y sobre elementos desordenados. En el primer caso, la búsqueda se facilita, y por lo tanto se ocupará menos tiempo que si se trabaja con elementos desordenados.
Todo lo mencionado anteriormente es aplicable a la búsqueda de datos en estructuras de información.
Los métodos de búsqueda pueden clasificarse en internos y externos, según donde estén almacenados los datos sobre los cuales se hará la búsqueda. Se denomina búsqueda interna cuando todos los elementos se encuentran en memoria principal, por ejemplo, almacenados en arreglos o listas ligadas. Se denomina búsqueda externa cuando todos los elementos se encuentran en memorias secundaria (archivo almacenados en dispositivos tales como cintas discos magnéticos).

3.1 BUSQUEDA SECUENCIAL

La búsqueda secuencial consiste en revisar elemento por elemento hasta encontrar el dato buscado, o hasta llegar al final de la lista de datos disponibles.
Primero se tratará sobre la búsqueda secuencial en arreglos, y luego en listas enlazadas. Cuando se habla de búsqueda en arreglos debe distinguirse entre arreglos desordenados y arreglos ordenados.
La búsqueda secuencial en arreglos desordenados consiste, básicamente, en recorrer el arreglo de izquierda a derecha hasta que se encuentre el elemento buscado o se termine el arreglo, lo que ocurra primero. Normalmente cuando una función de búsqueda concluye con éxito, interesa conocer en qué posición fue hallado el elemento buscado.

El algoritmo de búsqueda secuencial en arreglos desordenados es el que se explica a continuación:

ALGORITMO: Secuencialdesordenado

SECUENCIALORDENADO (V, N, X)

{ Este algoritmo busca secuencialmente el elemento X en el arreglo desordenado V, de N componentes }

{ I es una variable de tipo entero. BANDERA es una variable de tipo booleano }

1. Hacer I ← 1 y BANDERA ← FALSO
2. *Repetir mientras* (I ≤ N) y (BANDERA = FALSO)
   2.1 Si V [I] = X
   entonces
   Hacer BANDERA ← VERDADERO
   si no
   Hacer I ← I + 1
   2.2 { Fin del condicional del paso 2.1]
3. { Fin del ciclo del paso 2}
4. Si BANDERA = VERDADER0
   entonces
   Escribir “El elemento esta en la posición I”
   Si no
   Escribir “El elemento no esta en el arreglo”
5. { Fin condicional del paso 4 }

Son dos los posibles resultados a obtener por este algoritmo: la posición en la que encontró al elemento, o un mensaje de fracaso si el elemento no se halla en el arreglo.
Si hubiera dos o más ocurrencias del mismo valor, se encuentra la primera de ellas. Sin embargo, es posible modificar el algoritmo para obtener todas las ocurrencias del dato buscado.

El algoritmo también recibe el nombre de secuencial con bandera por utilizar la variable auxiliar booleana en la condición de parada del ciclo. El empleo de esta variable evita seguir buscando una vez que el dato ha sido encontrado, con la ventaja de que disminuye el número de comparaciones a realizar y por lo tanto aumenta la eficiencia del algoritmo.

3.2 BUSQUEDA BINARIA

La búsqueda binaria consiste en dividir el intervalo de búsqueda en dos partes, comparando el elemento buscado con el central. En este caso de no ser iguales se redefinen los extremos del intervalo (según el elemento central sea mayor o menor que el buscado) disminuyendo el espacio de búsqueda. El proceso concluye cuando el elemento es encontrado, o bien cuando el intervalo de búsqueda se anula.

Este método funciona únicamente para arreglos ordenados. Con cada iteración del método el espacio de búsqueda se reduce a la mitad, por lo tanto el número de comparaciones a realizar disminuye notablemente. Esta disminución resulta significativa cuanto más grande sea el tamaño del arreglo.

A continuación se presenta el algoritmo de búsqueda binaria.

ALGORITMO: Binaria

BINARIA (V, N, X)

{Este algoritmo busca al elemento X en el arreglo ordenado V de N componentes}

{ IZQ, CEN y DER son variables de tipo entero. BANDERA es una variable de tipo booleano }

1. Hacer IZQ ← 1, DER ← N y BANDERA ← FALSO
2. Repetir mientras (IZQ ≤ DER) y (BANDERA = FALSO)
   Hacer CEN ← (IZQ + DER) / 2
   2.1 Si X = V [CEN]
   entonces
   Hacer BANDERA ← VERDADERO
   si no {Redefinir intervalo de búsqueda }
   2.1.1 Si X > V [CEN]
   Entonces
   Hacer IZQ ← CEN + 1
   si no
   Hacer DER ← CEN – 1
   2.1.2 { Fin del condicional del paso 2.1.1 }
   2.2 { Fin del condicional del paso 2.1 }
3. { Fin del ciclo del paso 2 }
4. Si BANDERA = VERDADERO
   entonces
   Escribir “El elemento esta en la posición CEN”
   si no
   Escribir “El elemento no esta en el Arreglo”
5. { Fin del condicional del paso 4 }

Ejemplo:

Sea V un arreglo de números enteros, ordenado de manera creciente como se muestra en la figura:

V

101	215	325	410	502	507	600	610	612	670
1	2	3	4	5	6	7	8	9	10

En la tabla siguiente se presenta el seguimiento del algoritmo anterior para x = 325

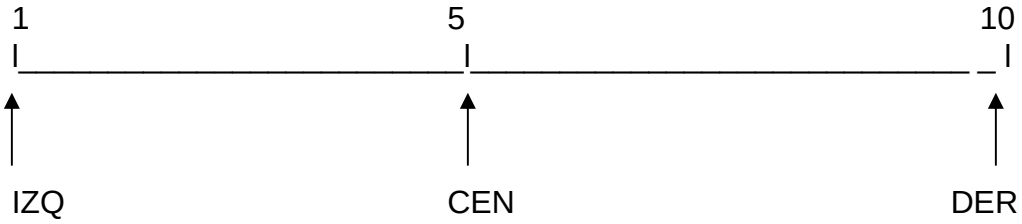
Tabla : Búsqueda Binaria.

PASO	BANDERA	IZQ	DER	CEN	V[CEN]	COMPARA
1	FALSO	1	10	5	502	325 < 502 ?
2	FALSO	1	4	2	215	325 < 215 ?
3	FALSO	3	4	3	325	325 = 325 ?
4	VERDADERO					

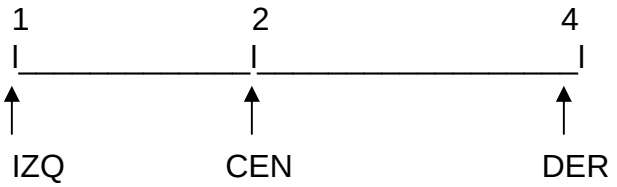
En las columnas IZQ y DER se colocaron los valores de los extremos izquierdo y derecho, respectivamente, del intervalo de búsqueda. En el primer paso se obtienen el elemento central, V[CENT]. Al comparar el elemento central con el valor buscado, COMPARA, puede decidirse cómo redefinir el espacio de búsqueda, en este caso se recorre el extremo derecho. En el paso 2 se procede de manera similar. La diferencia radica en que ahora se recorre el extremo izquierdo del intervalo. En el paso 3 al comparar el elemento central con x, se determina que se ha encontrado el valor buscado.

En la figura se representa gráficamente, para este ejemplo, cómo se va reduciendo el intervalo de búsqueda.

a)



b)



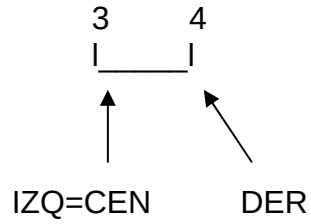


Figura: Reducción del intervalo de búsqueda. a) Paso 1. b) Paso

La tabla siguiente muestra el seguimiento del algoritmo anterior para x = 615, valor que no se encuentra en el arreglo.

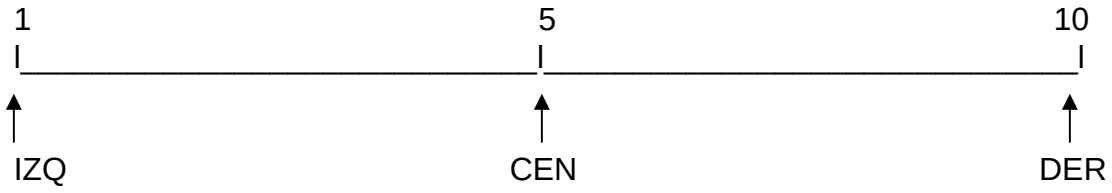
Tabla: Búsqueda binaria

PASO	BANDERA	IZQ	DER	CEN	V(CEN)	COMPARA
1	FALSO	1	10	5	502	615>502 ?
2	FALSO	6	10	8	610	615>610 ?
3	FALSO	9	10	9	612	615>612 ?
4	FALSO	10	10	10	670	615>670 ?
5	FALSO	10	9			

En los cuatro primeros pasos se procedió de la misma manera que en el caso anterior. Es decir, se comparó el elemento central con el valor buscado, y se recortó el espacio de búsqueda. En el paso 4, de la comparación resulta que debe redefinirse el extremo derecho, llegando de esta manera a anular el intervalo (IZQ > DER) en el paso 5.

La figura representa gráficamente cómo se va reduciendo el intervalo de búsqueda hasta anularse.

a)



b)

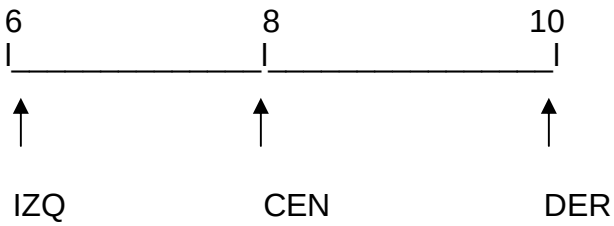


Figura: Reducción del intervalo de búsqueda a) Paso 1 b)Paso 2

3.3 Búsqueda por transformación de claves

Los dos métodos analizados anteriormente permiten encontrar un elemento en un arreglo, pero en ambos el tiempo de búsqueda es proporcional al número de componentes del mismo. Es decir, que a un mayor número de elementos se debe realizar un mayor número de comparaciones. Además se mencionó que si bien el método de búsqueda binaria es más eficiente que el secuencial, tiene la restricción de que el arreglo debe estar ordenado.
El método llamado por transformación de claves (hash), permite aumentar la velocidad de búsqueda sin necesidad de tener los elementos ordenados. Cuenta también con la ventaja de que el tiempo de búsqueda es prácticamente independiente del número de componentes del arreglo.
Suponer que se tiene una colección de datos, cada uno de ellos identificado por una clave. Es claro que resulta atractivo poder tener acceso a ellos de manera directa (sin tener que recorrer algunos datos antes de llegar al buscado). El método por transformación de claves permite esto. Trabaja basándose en una función de transformación o función hash (H) que convierte una clave dada en una dirección (índice) dentro del arreglo.

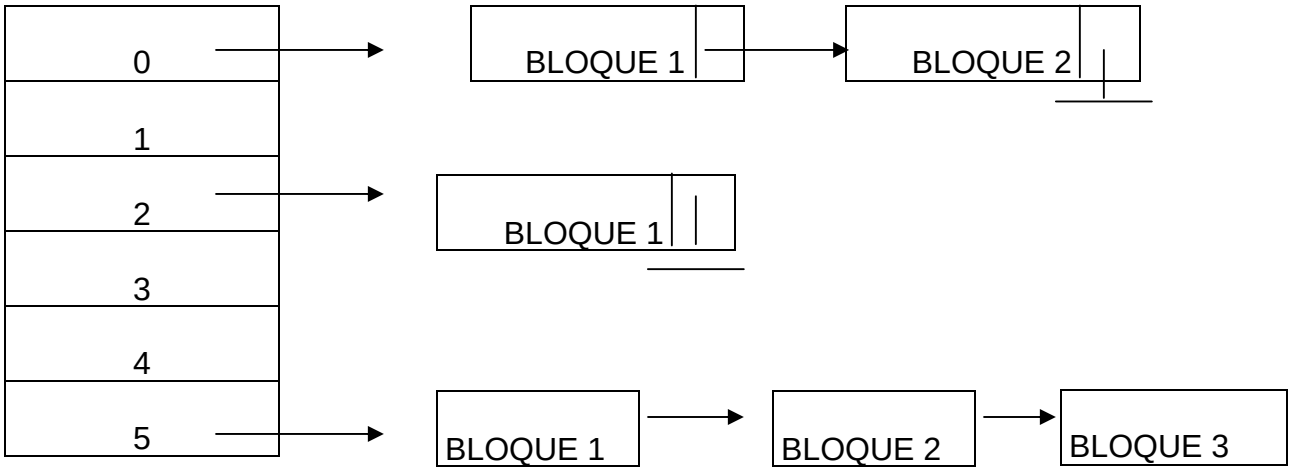
dirección ← H (clave)

La función hash aplicada a la clave da un índice del arreglo, lo que permite accesar directamente sus elementos. El caso más trivial se presenta cuando las claves son números enteros consecutivos. Suponer que se desea almacenar la información relacionada a 100 alumnos cuyas matrículas son números del 1 al 100. En este caso debe definirse un arreglo de 100 elementos con índices numéricos comprendidos entere los valores 1 y 100. Los datos de cada alumno ocuparán la posición del arreglo que se corresponda con el número de la matrícula, de esta manera se podrá accesar directamente la información de cada alumno. Suponer ahora que se desea almacenar la información de 100 empleados. La clave de cada empleado será su número del seguro social. Si la clave está formada por 11 dígitos, resulta por completo ineficiente definir un arreglo con 99 999 999 999 elementos para almacenar solamente los datos de 100 empleados. Utilizar un arreglo tan grande asegura la posibilidad de accesar directamente sus elementos, sin embargo el costo en memoria es excesivo. Siempre debe equilibrarse el costo por espacio de memoria con el costo por tiempo de búsqueda.
Cuando se tienen claves que no se corresponden con índices (por ejemplo, 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 ser simple de calcular y debe asignar direcciones de la manera más 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 $K_1 \neq K_2$), hay una colisión. 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 éstas se presentan se debe contar con algún método que genere posiciones alternativas

Se tratará sobre estos dos aspectos separadamente.
Busqueda por transformación de claves (hash)

El método por transformación de claves tiene básicamente las mismas características que el método presentado anteriormente.
Los archivos normalmente están organizados en áreas, llamadas cubetas. Las cubetas están formadas por cero, uno o más bloques de registros. Por lo tanto, la función hash, aplicada a una clave, dará como resultado un valor que hace referencia a una cubeta en la cual puede estar el registro buscado.
Tal como se mencionó en búsqueda interna, la elección de una adecuada función hash y de un método para resolver colisiones es fundamental para lograr mayor eficiencia en la búsqueda.
Antes de presentar algunas funciones hash, se hará un comentario sobre las colisiones. Los bloques contienen un número fijo de registros. Con respecto a las cubetas no se establece un límite en cuanto al número de bloques que pueden almacenar. Esta característica de las cubetas permite solucionar, al menos, parcialmente, el problema de las colisiones. Sin embargo, si el tamaño de las cubetas crece considerablemente, se perderán las ventajas propias de este método. Es decir, si el número de bloques a recorrer en una cubeta es grande, el tiempo necesario para ello será significativo y por lo tanto ya no se contará con la ventaja del acceso directo que caracteriza al método por transformación de claves.
En la figura se presenta una estructura de archivo organizado en cubetas, las que a su vez están formadas por bloques.



Directorios de cubetas

Figura: Archivo organizador en cubetas de bloques

Tal como se muestra en la figura, cada cubeta puede tener un apuntador a un bloque. Si una cubeta tiene dos o más bloques, se establecen ligas entre ellos. Dada la clave de un registro buscado, se aplicará una función hash la cual dará como resultado un número de cubeta. Una vez localizada la cubeta, habrá que recorrer los bloques de la misma hasta encontrar el registro, o bien hasta llegar a un bloque con puntero nulo, lo cual indicará que no existen otros bloques.
Debe elegirse una función hash que distribuya las claves a través de las cubetas, de tal manera que se evite la concentración de numerosas claves en una cubeta mientras otras permanecen vacías. A continuación se presentarán algunas de las funciones hash más comunes.

3.3.1 Funciones hash

Una función hash puede definirse como una transformación de clave a dirección, ya que al aplicar una función hash a una clave resulta el número de cubeta en la cual puede estar el registro con dicha clave.
La función debe transformar las claves para que la dirección resultante sea un número comprendido entre los posibles valores de las cubetas. Por ejemplo, si se tienen 10 000 cubetas enumeradas del 0 al 9999, las direcciones producidas por la función deberán ser valores comprendidos entre el 0 y el 9999. Si las claves fueran alfabéticas o alfanuméricas, primero deberán convertirse a numéricas, tratando de no perder información, para luego ser transformadas a una dirección. Es importante que la función distribuya homogéneamente las claves entre los números de cubetas disponibles.
Las funciones módulo, cuadrado, plegamiento y truncamiento presentadas para búsqueda interna son válidas también para búsqueda externa. Otra función que puede utilizarse para el cálculo de direcciones es la de conversión de bases, aunque no proporciona mayor homogeneidad en la distribución. De todas las citadas, la función módulo es de las que ofrece mayor uniformidad.

Como ya se ha mencionado, seleccionar una buena función hash es importante pero también es difícil. No hay reglas que permitan determinar cuál será la función más apropiada para un conjunto de claves, de tal manera que asegure la máxima uniformidad en la distribución de las mismas. Hacer un análisis de las principales características de las claves, puede ayudar en la elección de la función hash.
Algunas de las funciones hash más utilizadas se detallan abajo.

Función módulo (por división)

Co9nsiste en tomar el residuo de la división de la clave por el número de componentes del arreglo. Suponer que se tiene un arreglo de N elementos, ya sea K la clave del dato a buscar. La función hash queda definida por la siguiente fórmula:

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

En la fórmula anterior, puede observarse que al residuo de la división se le suma 1, esto es para obtener un valor entre 1 y N.
Para lograr una mayor uniformidad en la distribución, N debe ser un número primo o divisible por muy pocos números. Por lo tanto dado N, si éste no es un número primo se tomará el valor primo más cercano.

EJEMPLO

Sean N= 100 el tamaño del arreglo, y sean sus direcciones los números entre 1 y 100. Sean K_1= 7259 y K_2 = 9359 dos claves a las que deban asignarse posiciones en el arreglo. Se aplica la fórmula anterior con N =100, para calcular las direcciones correspondientes a K_1 y K_2.

$H(K_1)$= 7259 mod 100 + 1 = 60
$H(K_2)$= 9359 mod 100 + 1 = 60

Como H (K_1) es igual a $H(K_2)$ y K_1 es distinto de K_2, se está ante una colisión.
Se aplica ahora la fórmula anterior con N igual a un valor primo en vez de utilizar N igual a 100.
$H(K_1)$= 7259 mod 97 + 1 =82
$H(K_2)$= 9359 mod 97 + 1 =48

Con N = 97 se ha eliminado la colisión.

Función cuadrado

Consiste en elevar al cuadrado la clave y tomar los dígitos centrales como direccion. El número de dígitos a tomar queda determinado por el rango del índice. Sea K la clave del dato a buscar. La función hash queda definida por la siguiente fórmula:

$$\mathbf{H}(k) = \text{dígitos\_centrales } (\mathbf{K_2}) + 1$$

Formula

La suma de una unidad a los dígitos centrales es para obtener un valor entre 1 y N.

EJEMPLO

Sean N= 100 el tamaño del arreglo, y sean sus direcciones los números entre 1 y 100. Sean K_1=7259 y K_2=9359 dos claves a las que deban asignarse posiciones en el arreglo. Se aplica la fórmula para calcular las direcciones correspondientes a K_1 y K_2.

K_1= 52693081
K_2= 87590881

H(K_1)=dígitos_centrales (52693081)+1= 94

H(K_2)= dígitos_centrales (87590881)+1=91

Como el rango de índices en nuestro ejemplo, varía de 1 a 100 se toman solamente los dígitos centrales de cuadrado de las claves.

FUNCIÓN PLEGAMIENTO.

Consiste en dividir la clave en partes de igual número de dígitos (la última 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. Sea K la clave del dato a buscar. K está formada por los dígitos d1, d2,..., dn. La función hash queda definida por la siguiente fórmula:

H(k)= dígmensing ((d1...di) + (di+1...dj)+ ...+ (d1 ..,.dn)) + 1

Fórmula

El operador que aparece en la fórmula operando las partes de la clave es el de suma.
Pero como se aclaró antes, puede ser el de la multiplicación. La suma de una unidad a los dígitos menos significativos (dígmensig) es para obtener un valor entre 1 y N.
El ejemplo presenta un caso de función hash por plegamiento.

EJEMPLO:

Sean N = 100 el tamaño del arreglo, y sean sus direcciones los números entre 1 y 100.
Sean K_1= 7259 y K_2 = 9359 dos claves a las que deban asignarse posiciones en el arreglo. Se aplica la fórmula para calcular las direcciones correspondientes a K_1 y K_2.

H(K_1)= dígmensig (72 + 59) + 1= dígmensig (131) + 1 = 32

$H(K_2)$= dígmensig (93 + 59) + 1 dígmensig (152) +1= 53.

De la suma de las partes se toman solamente dos dígitos porque los índices del arreglo varían de 1 a 100.

Función truncamiento.

Consiste en tomar algunos dígitos de la clave y formar con ellos una dirección. Este método es de los más sencillos, pero es también de los que ofrecen menos uniformidad en la distribución de las claves.

Sea K la clave del dato buscar. K está formado por los dígitos $d_1,d_2,...,d_n$. La función hash queda definida por la siguiente fórmula:

$$H(k)= \text{elegirdígitos } (d_1\ d_2.....d_n) + 1$$

Fórmula

La elección de los dígitos es arbitraria. Podrían tomarse los dígitos de las posiciones impares o de las pares. Luego podría unírseles de izquierda a derecha o de derecha a izquierda. La suma de una unidad a los dígitos seleccionados es para obtener un valor entre 1 y 100.
El ejemplo presenta un caso de función hash por truncamiento.

EJEMPLO

Sean N= 100 el tamaño del arreglo, y sean sus direcciones los números entre 1 y 100.
Sean K_1= 7259 y K_2 = 9359 dos claves a las que deban asignarse posiciones en el arreglo. Se aplica la fórmula anterior para calcular las direcciones correspondientes a K_1 y K_2.

$H(K_1)$= elegirdígitos (7 2 5 9) + 1 =76
$H(K_2)$= elegirdígitos (9 3 5 9) + 1 = 96

En este ejemplo se toma el primer y tercer número de la clave y se une éste de izquierda a derecha.
En todos los casos anteriores se presentan ejemplos de claves numéricas. Sin embargo, en la realidad las claves pueden ser alfabéticas o alfanuméricas. En general, cuando aparecen letras en las claves se suele asociar a cada una un entero a efectos de convertirlas en numéricas.

A	B	C	D .....Z
01	02	03	04 27

Si por ejemplo la clave fuera ADA, su equivalente numérica sería 010401. Si hubiera combinación de letras y números, se procedería de la misma manera. Por ejemplo, dada una clave Z4F21, su equivalente numérica sería 2740621. Otra alternativa sería, para cada carácter, tomar el valor decimal asociado según el código ASCII. Una vez obtenida la clave en su forma numérica, se puede utilizar normalmente cualquiera de las funciones antes mencionadas.

3.3.2 SOLUCION DE COLISIONES

La elección de un método adecuado para resolver colisiones es tan importante como la elección de una buena función hash. Cuando la función hash obtiene una misma dirección para dos claves diferentes, se está ante una colisión. Normalmente, cualquiera que sea el método elegido, resulta costoso tratar las colisiones. Es por ello que debe hacerse un esfuerzo por encontrar la función que ofrezca mayor uniformidad en la distribución de las claves.
La manera más natural de resolver el problema de las colisiones es reservar una casilla por clave. Es decir, que aquéllas se correspondan una a una con las posiciones del arreglo. Pero como ya se mencionó, esta solución puede tener un alto costo en memoria. Por lo tanto deben analizarse otras alternativas que permitan equilibrar el uso de memoria con el tiempo de búsqueda.

Solución de colisiones

Como se mencionó en búsqueda interna, una de los aspectos a considerar en el método por transformación de claves es la solución de colisiones. Cuando dos o más elementos con distintas claves tienen una misma dirección, se origina una colisión.
Para evitar las colisiones se debe elegir un tamaño adecuado de cubetas y de bloques. Con respecto a las cubetas, si se definen muy pequeñas el número de colisiones aumenta, mientras que si se definen muy grandes se pierde eficiencia en cuanto a espacio de almacenamiento. Además, si se necesitara copiar una cubeta en memoria principal y la misma fuera muy grande, entonces se tendrían problemas por falta de espacio. Otro inconveniente que se presenta en el caso de cubetas muy grandes, es que se requiere mucho tiempo para recorrerlas. Con respecto al tamaño de los bloques, se hace referencia a capacidad de estos para almacenar registros. Un bloque puede almacenar uno, dos o más registros. Normalmente los tamaños de las cubetas y bloques dependen de las capacidades del equipo con el que se este trabajando.
Utilizando la estructura presentada en la figura anterior no se tendrían problemas de colisiones, debido a que a pesar de que la cubeta puede estar ocupada, se pueden seguir enlazando tantos bloques como fueran necesarios. Este esquema de solución se corresponde con el presentado en búsqueda interna, bajo el nombre de encadenamiento. Sin embargo, no siempre es posible definir una estructura como esta. Considérese un archivo organizado en cubetas como l que se muestra en la figura anterior.

Métodos más utilizados para resolver colisiones, los cuales pueden clasificarse en:

- Reasignación
- Arreglos anidados
- Encadenamiento

Resignación

Existen varios métodos que trabajan bajo el principio de comparación y reasignación de elementos. Se analizarán tres de ellos:

- Prueba lineal
- Prueba cuadrática
- Doble dirección hash

a) Prueba lineal

Consiste en que un vez detectada la colisión se debe recorrer el arreglo secuencialmente a partir del punto de colisión, buscando al elemento. El proceso de búsqueda concluye cuando el elemento es hallado, o bien cuando se encuentra una posición vacía. Se trata al arreglo como a una estructura circular: el siguiente elemento después del último es el primero.

A continuación se expone el algoritmo de solución de colisiones por medio de la prueba lineal.

Algoritmo Pruebalineal

PRUEBALINEAL (V, N, K)

{Este algoritmo busca al dato con clave K en el arreglo V de N elementos. Resuelve el problema de las colisiones por medio de la prueba lineal}

{D y DX son variables de tipo entero}

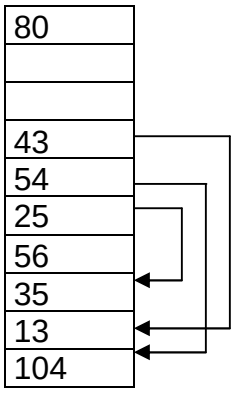
1. Hacer D ←H (K) {Genera dirección}
2. Si *V[D*]. Clave= K
   entonces
   Escribir “El elemento esta en la posición D”
   si no
   Hacer DX ←D+1

2.1 Repetir mientras (V[DX].CLAVE≠K) y
(V[DX]≠VACIO) y (DX≠D)
Hacer DX←DX+1
2.1.1 Si DX = N+1 entonces
Hacer DX←1

2.1.2 {Final del condicional del paso 2.1.1}
2.2 {Fin de ciclo del paso 2.1}
2.3 Si V[DX].CLAVE= K
entonces
Escribir " El elemento está en la posición DX"
si no
Escribir "El elemento no está en el arreglo"
2.4 { Fin del condicional del paso 2.3}
3. {Fin del condicional del paso 2}

La tercera condición del ciclo del 2.1 es para evitar caer en un ciclo infinito, si el arreglo estuviera lleno y el elemento a buscar no se encontrara en él.
La principal desventaja de este método es que puede haber un fuerte agrupamiento alrededor de ciertas claves, mientras que otras zonas del arreglo permanezcan vacías.
Si las concentraciones de claves son muy frecuentes, la búsqueda será principalmente secuencial perdiendo así las ventajas del método hash. Con el ejemplo se ilustra el funcionamiento del algoritmo anterior

V



a)

K	H(K)
25	6
43	4
56	7
35	6
54	5
13	4
80	1
104	5

b)

EJEMPLO

Sea V un arreglo de 10 elementos. Las claves 25, 43, 56, 35, 54, 13, 80 y 104 fueron asignados según la función hash:

H(K)=K mod 10 + 1

En la figura anterior se aprecia el estado del arreglo a) y la tabla con H(K) para cada clave b).

En la tabla anterior se presenta el seguimiento del algoritmo anterior para el caso del ejemplo anterior, y el dato a buscar igual a 35.

Tabla Solución de colisiones por la prueba lineal

PASO	D	DX	COMPARA
1	6	-	35=25?
2	6	7	35=56?
3	6	8	35=35?

Al aplicar la función hash a la clave 35, resulta una dirección (D) igual a 6, paso 1. Sin embargo, en esa posición no se encuentra el elemento buscado, por lo

que se empieza a recorrer secuencialmente al arreglo a partir de la posición 7 (DX), paso 2. En este caso la búsqueda concluye al encontrar el valor deseado en la posición 8, paso 3.

b) Prueba cuadrática

Este método es similar al de la prueba lineal. La diferencia consiste en que en el cuadrático las direcciones alternativas se generarán como D + 1, D + 4, D + 9,..., $D + i^2$ en vez de D + 1, D + 2,..., D + i. Esta variación permite una mejor distribución de las claves colisionadas.

El algoritmo de solución de colisiones por medio de la prueba cuadrática se estudia enseguida.

Algoritmo Pruebacuadrática

PRUEBACUADRÁTICA (V, N, K)

(Este algoritmo busca al dato con la clave K en el arreglo V de N elementos. Resuelve el problema de las colisiones por medio de la prueba cuadrática.)

{ D, DX y I son variables tipo entero }

1. Hacer D H (K) { Genera dirección }
2. Si V(D) CLAVE = K
   entonces
   Escribir " El elemento está en la posición D"
   si no
   Hacer I ←1 y DX← $D + I^2$

2.1 Repetir mientras (V[DX]. CLAVE ≠ K) y (V[DX] ≠VACÍO)
Hacer I ←I + 1 y DX← $D+ I^2$
2.1.1 Si DX > N entonces
Hacer I ←0, DX← 1 y D ←1

2.1.2 {Fin del condicional del paso 2.1.1}
2.2 {Fin del ciclo del paso 2.1}
2.3 Si V[DX]. CLAVE = K
entonces
Escribir "El elemento está en la posición DX"
si no
Escribir "El elemento no está en el arreglo"
2.4 { Fin del condicional del paso 2.3}
3. { Fin del condicional del paso 2 }

La principal desventaja de este método es que pueden quedar casillas del arreglo sin visitar. Además, como los valores de las direcciones varían en l^2 unidades, resulta difícil determinar una condición general para detener el ciclo del punto 2.1. Este problema podría solucionarse empleando una variable auxiliar, cuyos valores dirijan el recorrido del arreglo de tal manera que garantice que serán visitadas todas las casillas.

A continuación se presenta un ejemplo que ilustra el funcionamiento del algoritmo.

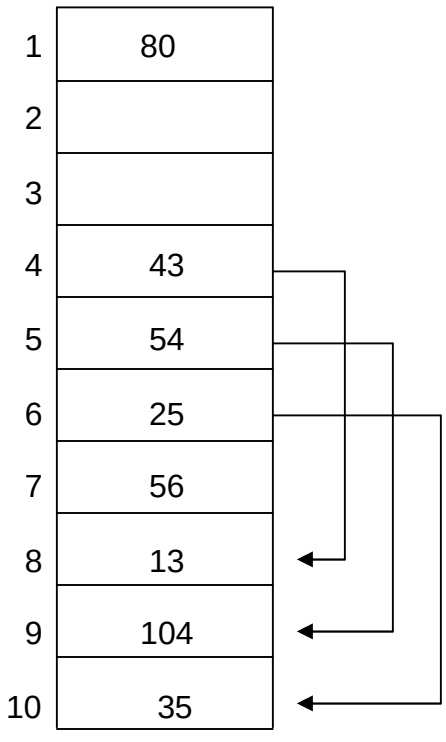
EJEMPLO

Sea V un arreglo de 10 elementos. Las claves 25, 43, 56, 35, 54, 13, 80 y 104 fueron asignadas según la función hash:

H(K) = K mod10 + 1

En la figura se presenta el estado del arreglo a} y la tabla con H(K) para cada clave b).

La tabla contiene el seguimiento del algoritmo anterior para el caso del ejemplo anterior, y el dato a buscar igual a 35.



a)

K	H(K)
25	6
43	4
56	7
35	6
54	5
13	4
80	1
104	5

b)

Figura Solución de colisiones por la prueba cuadrática. a) Arreglos.
b) Tabla con H(K)

Tabla Solución de colisiones por la prueba cuadrática.

PASO	D	I	DX	COMPARA
1	6	-	-	35=25 ?
2	6	1	7	35=56?
3	6	2	10	35=35?

Aplicar la función hash a la clave 35 , resulta una dirección (D) igual a 6, paso1.
Pero en esa posición no se encuentra el elemento buscado. Se calcula DX, paso 2, como la suma de D + I^2. Se continúa recorriendo el arreglo, comparando las posiciones indicadas por DX con el dato. En el paso 3, se encuentra el valor deseado en la posición 10 del arreglo.

c) Doble dirección hash

Consiste en que una vez detectada la colisión se debe generar otra dirección aplicando la función hash a la dirección previamente obtenida. El proceso se detiene cuando el elemento es hallado, o bien cuando se encuentra una posición vacía.

D H(K)
D´ H(D)
D” H(D´)
...

La función hash que se aplique a las direcciones puede o no ser la misma que originalmente se aplicó a la clave. No existe una regla que permita decidir cuál será la mejor función a emplear en el cálculo de las sucesivas direcciones.

Algoritmo de solución de colisiones por medio del método de la doble dirección hash.

Algoritmo Dobledirección

DOBLEDIRECCIÓN (V,N,K)

{ Este algoritmo busca el dato con clave K en el arreglo V de N elementos. Resuelve el problema de las colisiones por medio de la doble dirección hash }

{D y DX son variables de tipo entero }

1. Hacer D ← H (K)
2. Si V [D]. CLAVE = K
   entonces
   Escribir “El elemento esta en la posición D”.
   si no
   Hacer DX ← H´(D)

2.1 *Repetir mientras* (V[DX].CLAVE ≠ K),(V[DX] ≠ VACÍO) y (DX ≠ D)
Hacer DX ← H´(DX)
2.2 [Fin del ciclo del paso 2.1]
2.3 *Si* V[DX].CLAVE =K
Entonces
Escribir “El elemento está en la posición DX”
si no
Escribir “El elemento no están en el arreglo”
2.4 {Fin del condicional del paso 2.3}
3. {Fin del condicional del paso 2 }

3.4 Árboles binarios de búsqueda

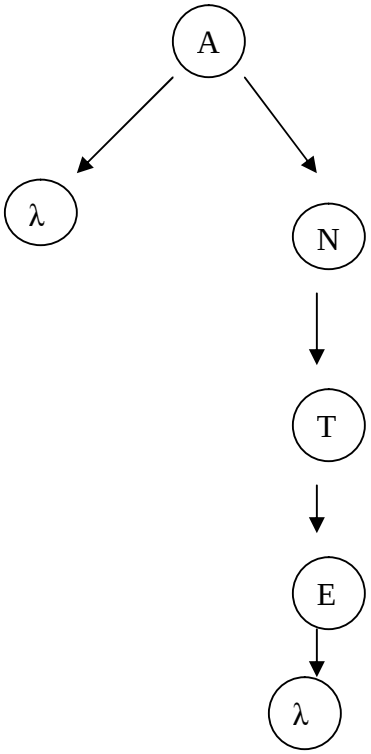
Los árboles son una estructura poderosa y eficiente para almacenar y recuperar información. Debido al dinamismo que caracteriza a los árboles, el beneficio de utilizarlos es mayor, cuanto más variable sea el número de datos a tratar.

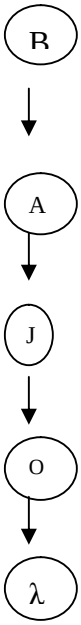
La estructura trie, que es una variante de la estructura tipo árbol.

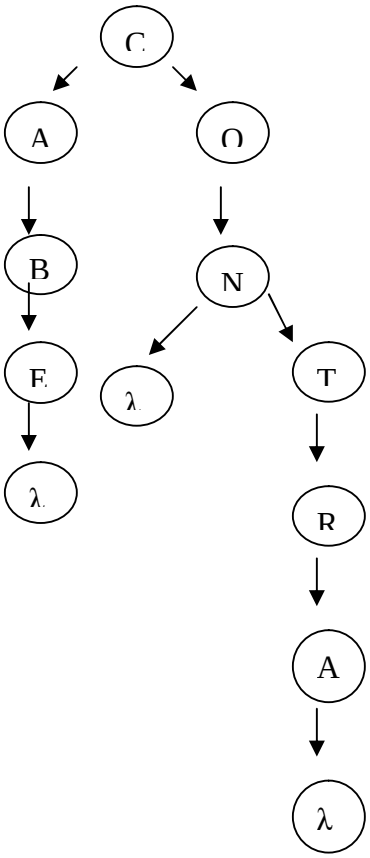
Un **trie** es un árbol de N ramas que se caracteriza porque la información de cada nodo es común a todos sucesores. Por lo tanto, para buscar una clave se procederá carácter por carácter de manera descendiente en el árbol. La inicial de la clave estará en la raíz del árbol, y en el nodo terminal un indicador de final de clave. A continuación se ilustra este concepto con un ejemplo.

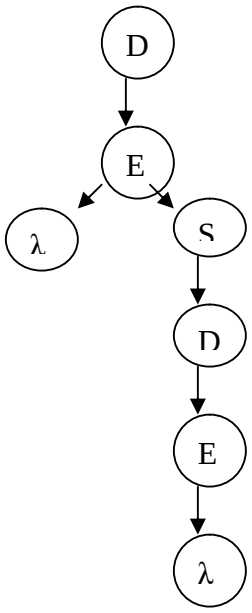
EJEMPLO

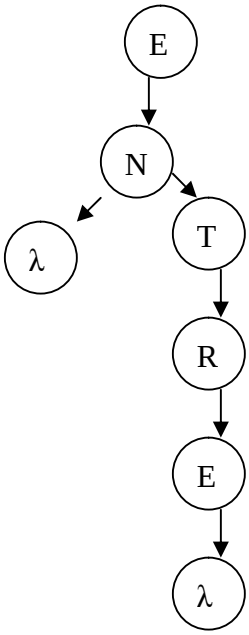
Sean las proposiciones del castellano las palabras claves a almacenar: A, ANTE, BAJO, CABE, CON, CONTRA, DE, DESDE, EN, ENTRE, HACIA, HASTA, PARA, POR, SEGÚN, SIN, SO, SOBRE TRAS. La figura muestra la estructura tries correspondientes a estas claves.

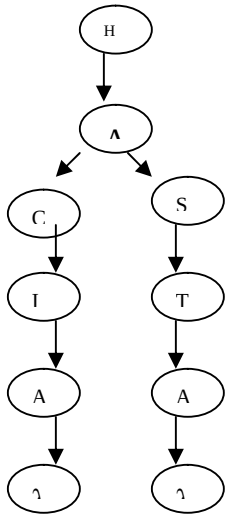


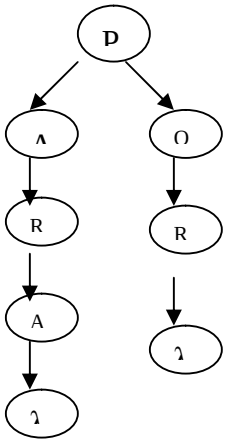

R
A
J
O
λ


C
A
O
B
N
F
λ
T
λ
R
A
λ


D
E
λ
S
D
E
λ


E
N
λ
T
R
E
λ


H
∧
C
S
I
T
A
A


D
∧
O
R
R
A

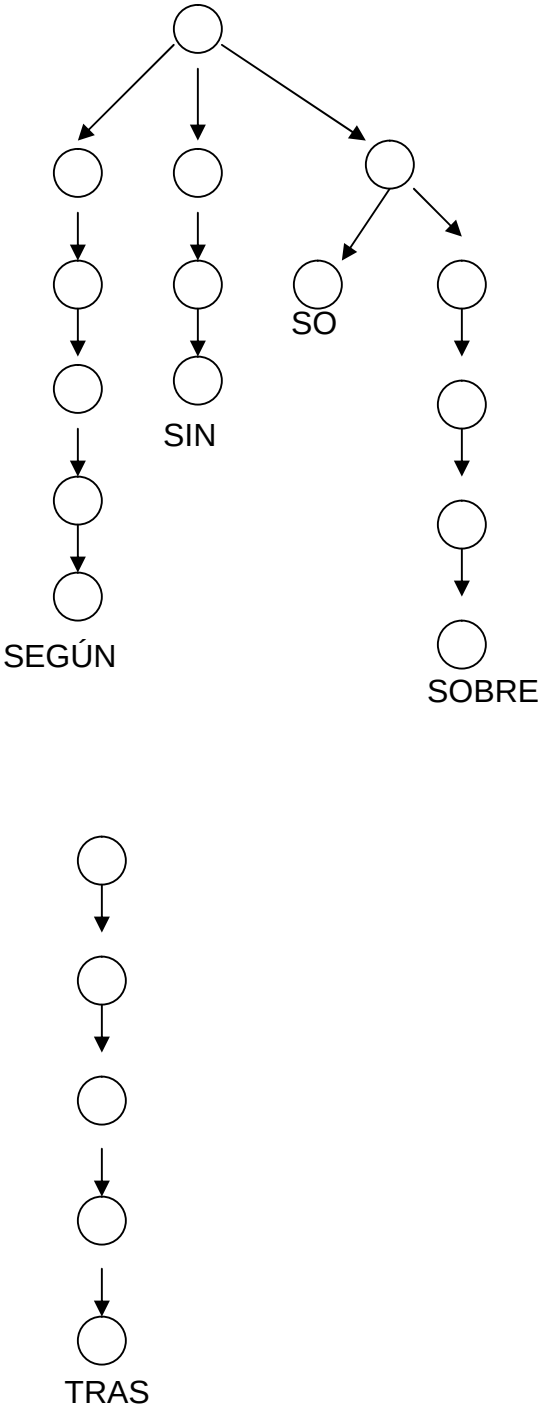


Figura: Representación de tries

Como se puede apreciar en la representación de tries de la figura anterior la relación entre esta estructura y la de árbol es inmediata. Si se definiera una doble liga (predecesor y sucesor) por cada nodo, la búsqueda de elementos a

través del trie se facilitaría notablemente. Igualmente favorecería las operaciones de eliminación e inserción.
Es necesario aclarar que no representan la clave carácter por carácter, sino que a los caracteres comunes a varias claves o finales los incluyen en un solo nodo (figura siguiente)

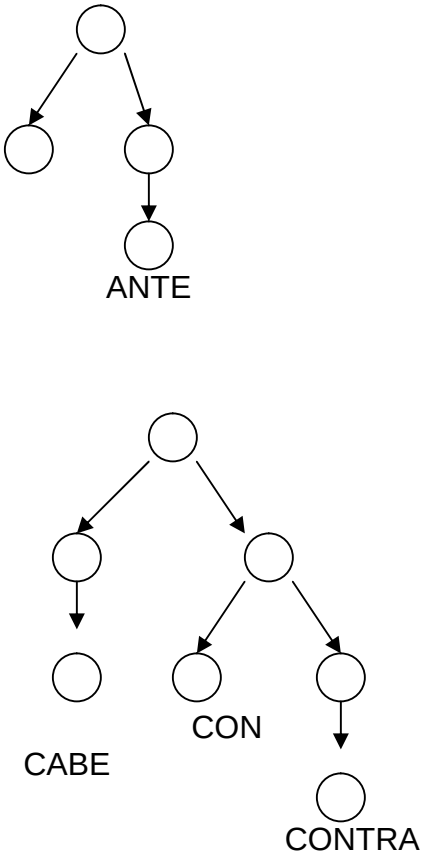
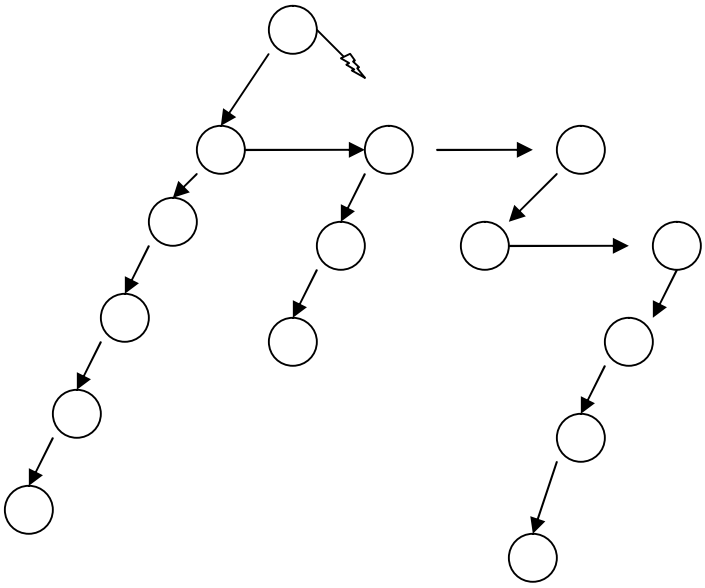


Figura: Representación de tries

Otra forma de representar los tries es por medio de árboles binarios. Para transformar un árbol de N claves en uno binario debe aplicarse el siguiente criterio para cada nodo: la rama izquierda indica relación de descendencia y la rama derecha indica relación de hermandad entre nodos. A continuación, en la figura siguiente, se presenta uno de los tries del ejemplo anterior, utilizando una estructura de árbol binario.



Árboles

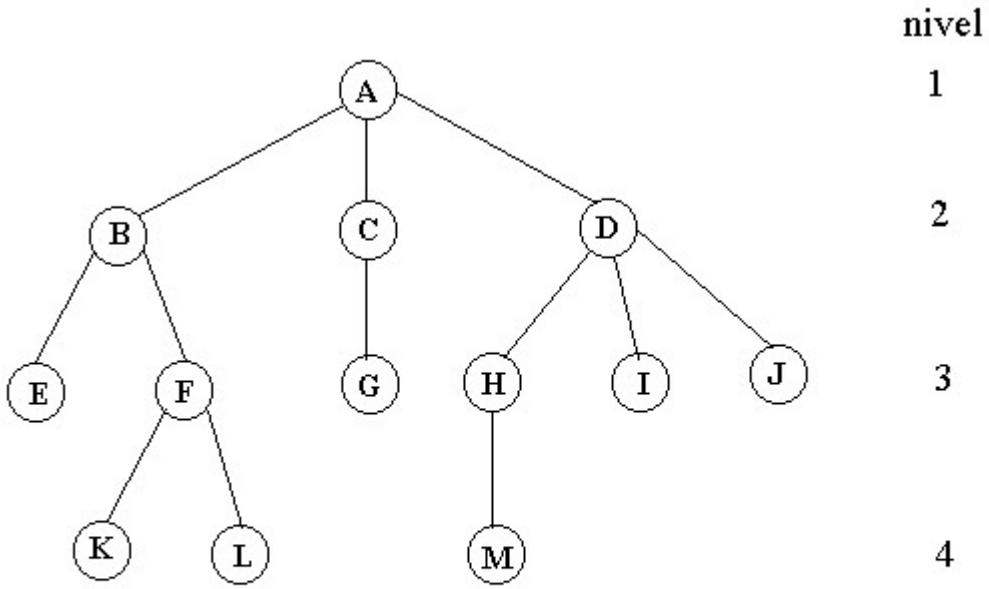
En una estructura de árboles los datos están organizados donde sus elementos de información están relacionados por sus ramificaciones.

Definición: un árbol es un conjunto finito de uno o más nodos, tal que:

1. hay un nodo con una especial designación llamado raíz;
2. el resto de los nodos están separados dentro de n>=0 conjuntos desordenados T1,T2,...Tn , donde cada uno de estos conjuntos es un árbol. T1,T2,..Tn son llamados subárboles de la raíz.

GRADO: es el número de subárboles de un nodo, cuando el grado= 0 se dice que es una hoja o nodo terminal.

Los subárboles de la raíz de **X** son llamados hijos de **X**



(A (B (E , F (K , L)), C (G) ,D (H (M), I, J)))

La profundidad o altura de un árbol está definida por su nivel mayor.

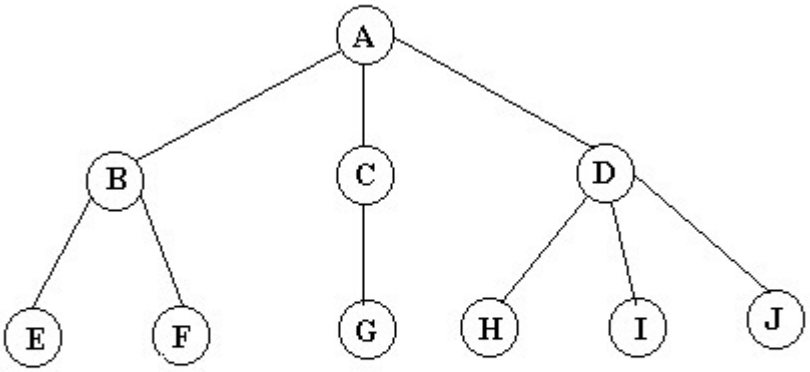
Nodo

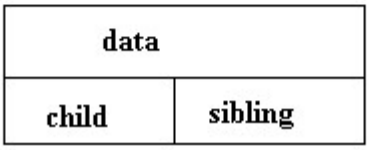
entry	link1	link2	link3	link4	…	linkn

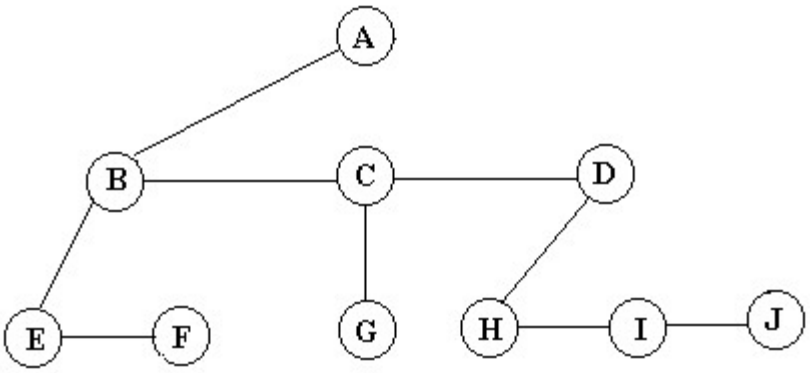
Para un árbol de grado 3 más del 2/3 de sus ligas apuntan a null, por lo tanto el desperdicio aumenta.

En una estructura de árbol binario solo aprox. ½ de sus ligas apuntan a null.

a) el orden los hijos no es importante
b) la relación el hijo más izquierdo, el siguiente hijo derecho es su hermano

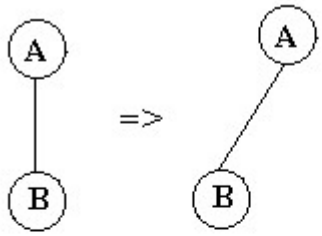


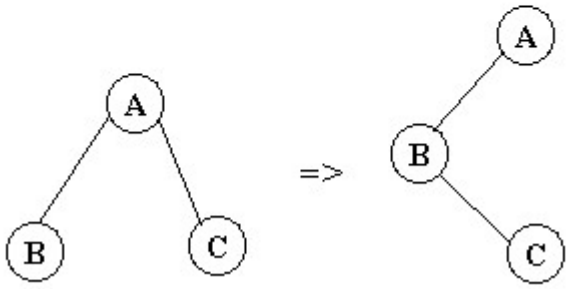




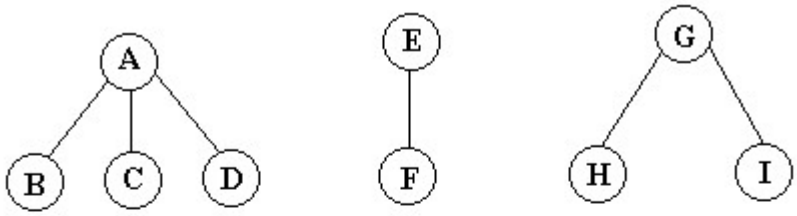
Transformaciones para un árbol binario:

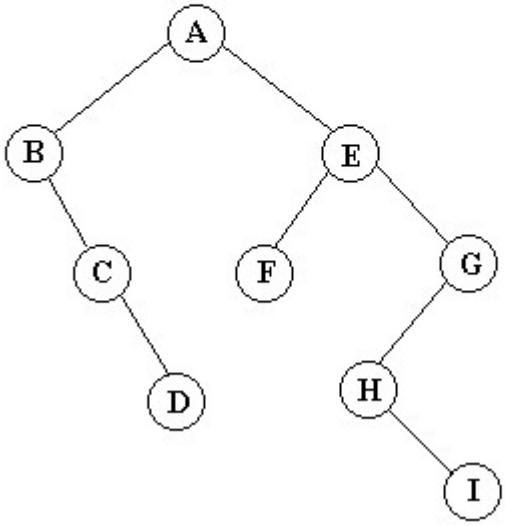
Para cualquiera de estas dos transformaciones la raíz no tendrá subárbol derecho.





Bosque



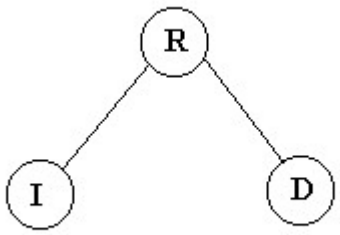


Árboles Binarios

Se utilizan en:
- comparación de algoritmos
- llamados a subprogramas
- recursión de algoritmos
- aplicación de búsqueda binaria

Definición:

Un árbol binario consiste de un nodo llamado raíz y dos subárboles denominados subárbol izquierdo y subárbol derecho.



Trayectorias:
- R I D - R D I
- I R D - D R I
- I D R - D I R

Preorder - R I D
Inorder - I R D
Postorder - I D R

expresión	a + b	n!	a - (b * c)	log x	(a<b) or (c<d)
preorder	+a b	!n	-a * b c	log x	or < a b < c d
inorder	a + b	n!	a - (b * c)	log x	(a<b) or (c<d)
postorder	a b +	n!	a b c * -	x log	a b < c d < or

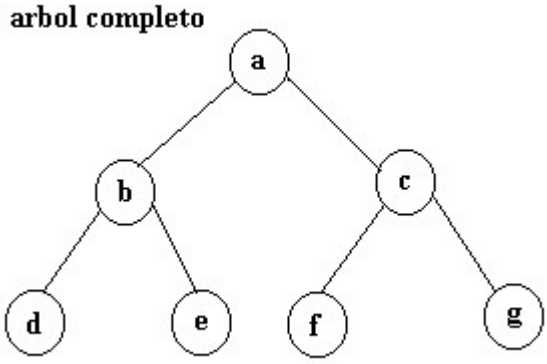
3.5 Árboles de búsqueda binaria

Debe de cumplir con lo siguiente:

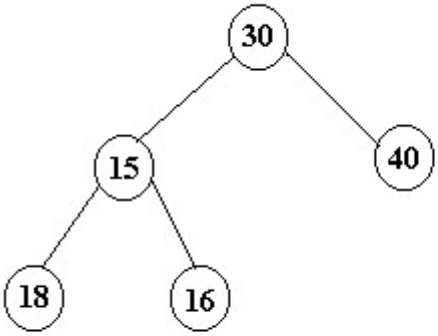
a) cada nodo deberá contener una llave, que no podrá tener un valor duplicado
b) la llave del hijo izquierdo del nodo(si existe) deberá ser menor a la llave de su padre
c) la llave del hijo derecho del nodo(siexiste) deberá ser mayor a la llave de su padre.

```
typedef struct cuerda{
   char key[4];
} TreeEntry;


typedef struct treenode{

    TreeEntry entry;
    struct treenode *left;
    struct treenode *right;

}TreeNode;


TreeNode *TreeSearch(TreeNode *root, TreeEntry target)
{
  if (root)
     if (strcmp(root->entry.key, target.key)<0)
         root= TreeSearch(root->left, target);
     else
           if (strcmp(root->entry.key, target.key)>0)
               root = TreeSearch(root->right, target);

  return  root;

    }
}
```

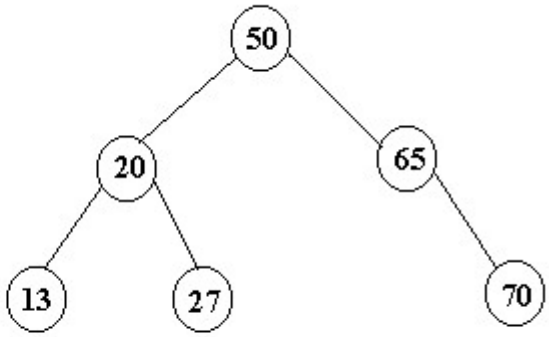
Para que el algoritmo TreeSearch sea eficiente en su búsqueda es importante que la estructura del árbol sea lo más semejante a un árbol completo, en donde sería similar al concepto de la búsqueda binaria y su eficiencia.



número de nodos en un árbol completo = **2k -1** donde k es la altura



no cumple



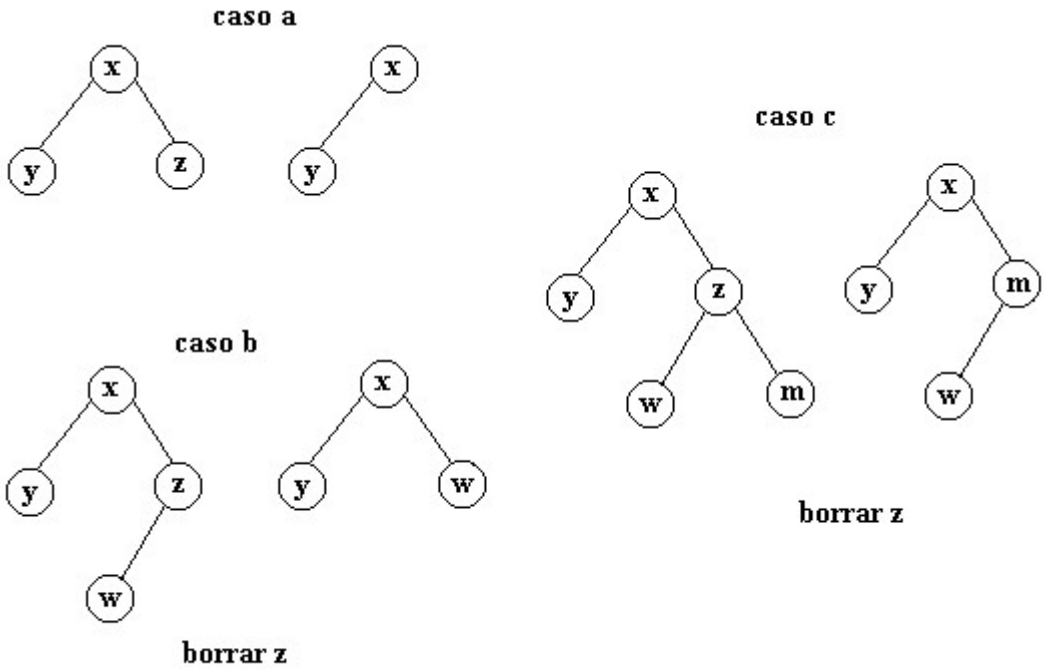
si cumple

```
TreeNode *InsertTree(TreeNode *root, TreeNode *newnode)
{
  if (!root) {
     root=newnode;
     root->left=root->right=NULL;}
  else  if (LT(newnode->entry.key,root->entry.key))
            root->left=InsertTree(root->left,newnode);
        else
            root->right=InsertTree(root->right,newnode);
   return root;
}
```

Eliminar un nodo en un árbol de búsqueda

Casos:
a) eliminar una hoja
b) eliminar un padre con subárbol izq.

c) eliminar un padre con subárbol izq. y subárbol der.

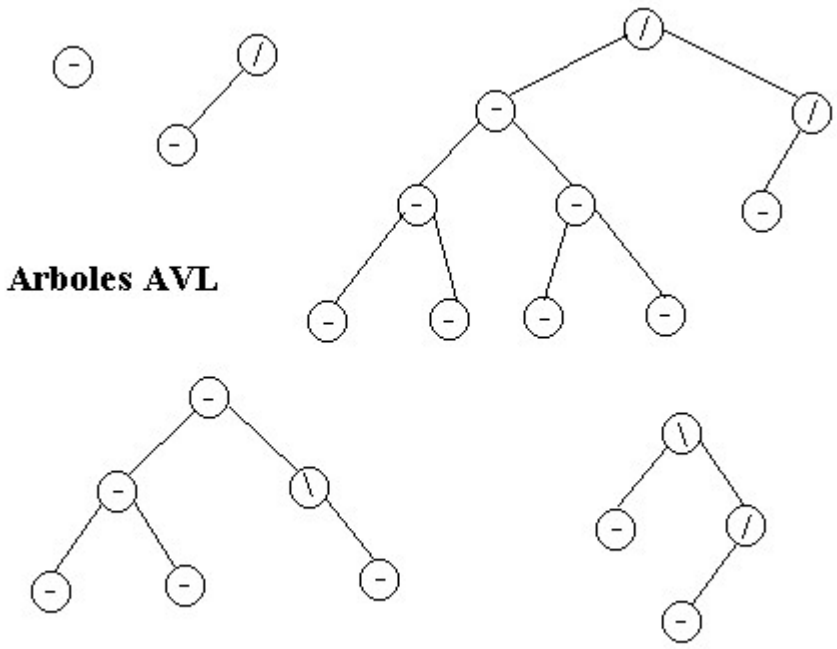


3.5.1 Árboles Balanceados (AVL)

Un árbol AVL es un árbol de búsqueda binaria en el cual las alturas del subárbol izquierdo y del subárbol derecho a la raíz difieren en 1. En cada nodo de un árbol AVL tiene un valor asociado que es el factor de balance.

Factor de balance = altura subder - altura subizq

Valores posibles = 0, -1, 1
Símbolos:
/ nodo altura izquierda
\ nodo altura derecha
- nodo igual altura

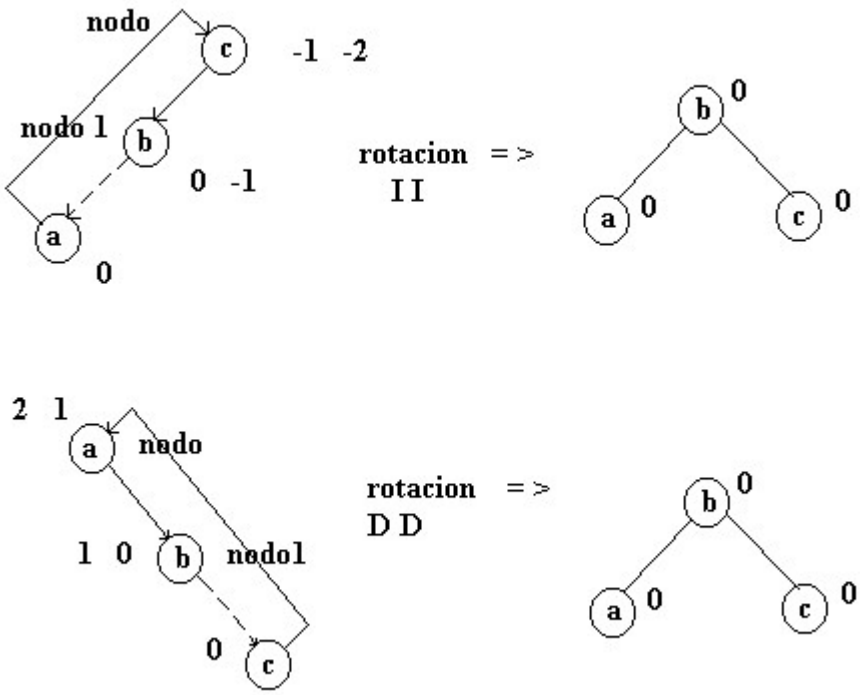


Rotaciones:

simple	**D D**	afecta solo a 2 nodos
	I I	
compuesta	**D I**	afecta a 3 nodos
	I D	

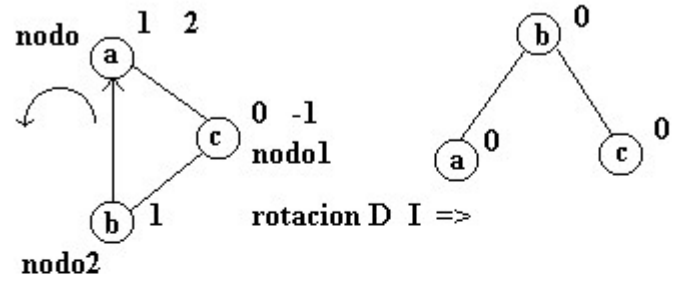
D D - ramas derecha
I I - ramas izquierdas

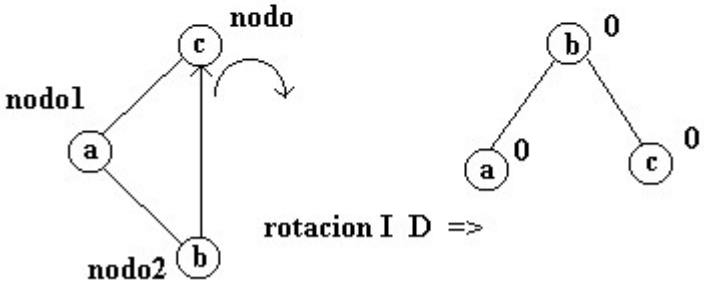
D I - por la rama derecha e izquierda
I D - por la rama izquierda y derecha



rotación l l :
nodo->left = nodo1->right
nodo1 ->right = nodo
nodo = nodo1

rotación D D :
nodo->right = nodo1-> left
nodo1->left = nodo
nodo = nodo1





rotación D I :
nodo1->left = nodo2->right
nodo2 ->right = nodo1
nodo->right = nodo2->left
nodo2->left = nodo
nodo= nodo2

rotación I D :
nodo1->right = nodo2->left
nodo2->left = nodo1
nodo->left = nodo2->right
nodo2->right = nodo
nodo=nodo2

Procedure **AVL-Insert**(X,Y,T) //algoritmo iterativo
{
/* the identifier X is inserted into the AVL tree with root T. Each node is assumed to have an identifier field IDENT, LCHILD,RCHILD fields and balance factor BF.
BF(P) = height of LCHILD(P) - height of RCHILD(P).
Y is set such that IDENT(Y)=X
*/

//special case: empty tree T=0//
if T=0 then { call Getnode(Y); IDENT(Y)=X; T=Y;

```
                BF(T)=0;LCHILD(T)=null; RCHILD(T)=null;
                return;
              }
/*phase 1: locate insertion point for X. A keeps track of most recent node with
balance factor ±1 and F is the parent of A. Q follows P through the tree.
*/
F=NULL; A=T;
P=T; Q=NULL;
while P!= NULL do     // search T for insertion point for X
{
 if BF(P)!=0 then { A=P; F=Q;}
 case
 {
      : X<IDENT(P):  Q=P; P=LCHILD(P); //take left branch

      :X >IDENT(P): Q=P; P=RCHILD(P); //take right branch

    : else: Y=P; return     //X is in T

 }
}
/*phase 2: Insert and rebalance. X is not in T and may be inserted as appropriate
child of Q
*/
call Getnode(Y); IDENT(Y)=X; LCHILD(Y)=NULL;
RCHILD(Y)=NULL; BF(Y)=0;
if X <IDENT(Q) then LCHILD(Q) =Y; //insert as left child
                    else RCHILD(Q)=Y; //insert as right child

/* adjust balance factors of nodes on path from A to Q. Note that by the definition of
A, all nodes on this parth must have balance factors of 0 and so will change to ±1.
d=+1 implies X is inserted in left subtree of A. d=-1 implies X is inserted in right
subtree of A
*/

if X>IDENT(A) then { P=RCHILD(A); B=P; d=-1;}
                    else {P=LCHILD(A); B=P; d= +1;}
while P!=Y do
{
 if X >IDENT(P)
    then {BF(P) =-1; P=RCHILD(P);}  //height of right increases by 1
    else {BF(P)=+1; P=LCHILD(P);}  //height of left increases by 1
}

/* Is tree unbalanced?*/
if BF(A)=0 then {BF(A)=d; return}  //tree still balanced
if BF(A)+d=0 then {BF(A)=0;return}  //tree is balanced
//tree unbalanced, determine rotation type
```

```
if d=+1 then  //left imbalance
   case
   {
     : BF(B)=+1:  //rotation type LL
     LCHILD(A)=RCHILD(B); RCHILD(B)=A;
 BF(A)=BF(B)=0;
     : else:
  C=RCHILD(B);
 CL=LCHILD(C);
CR=RCHILD(C);
RCHILD(B)=CL;
LCHILD(A)=CR;
LCHILD(C)=B;
RHILD(C)=A;
case
{
  :BF(C)=+1; BF(A)=-1; BF(B)=0 //LR(b)
  :BF(C)=-1; BF(B)=+1; BF(A)=0 //LR(c)
  :else:
    BF(B)=0;BF(A)=0;   //LR(a)
}
BF(C)=0; B=C  //B in a new root
   }
 else
   /*right imbalance; this is symmetric to left imbalance and is left as an
      exercise*/
/*subtree with root B has been rebalanced and is the new subtree of F. The original
subtree of F had root A*/
case
{
  :F=NULL: T=B;
  :A=LCHILD(F): LCHILD(F)=B;
  :A=RCHILD(F): RCHILD(F)=B;
}
}
```

```
TreeNode *InsertAVL(TreeNode **root, TreeNode *newnode, Boolean *taller)
//algortimo recursivo
{
   if (!root) {
     root=newnode;
     root->left=root->right=NULL;
     root->bf=EH;
     *taller=true;
```

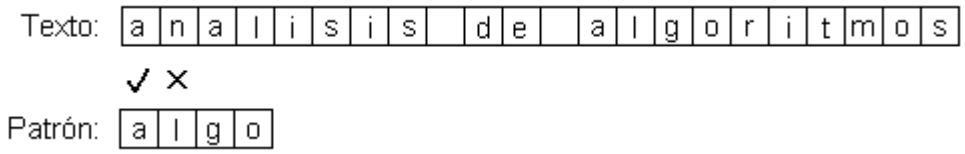
```
  }
   else if (EQ(newnode->entry.key,root->entry.key)) {
          Error("duplicate key");
        }else if (LT(newnode->entry.key, root->entry.key)) {
                root ->left=InsertAVL(root->left,newnode,taller);
                if (*taller)
                    switch (root->bf) {
                        case LH:
                          root= LeftBalance(root,taller); break;
                        case EH:
                          root->bf= LH; break;
                        case RH:
                          root->bf= EH;
                          *taller=false;
                          break;
                    }
                else {
                       root ->right=InsertAVL(root->right, newnode, taller);
                       if (*taller)
                          switch (root->bf) {
                                case LH:
                                     root->bf=EH;
                                     *taller=false; break;
                                case EH:
                                     root->bf= RH; break;
                                case RH:
                                    root=RigthBalance(root,taller);
                                    break;
                          }
                 }
            return root;
}
TreeNode *RotateLeft(TreeNode *p)
{
  TreeNode *rightchild=p;

  if (!p)
     Error("it is impossible to rotate an empty tree in rotate left");
  else  if (!p->right)
            Error("it is impossible to make an empty subtree the root in rotateleft");
    else {
          rightchild= p->right;
          p->right= rightchild->left;
          rightchild->left=p;
 }
```

```
   return rightchild;
}

TreeNode *RightBalance(TreeNode *root, Boolean *taller)
{
  TreeNode *rs=root->right;
  TreeNode *ls;

switch(rs->bf) {
   case RH:
         root->bf=rs->bf=EH;
         root=RotateLeft(root);
         *taller=false;
          break;
   case EH:
         Error("tree is already balanced");
         break;
   case LH:
         ls=rs->left;
         switch(ls->bf){
             case RH:
                root->bf=LH;
                rs->bf=EH;
                break,
             case EH:
                root->bf=rs->bf=EH;
                break;
             case LH:
                root->bf=EH;
                rs->bf=RH;
                break;
           }
        ls->bf=EH;
        root->right=RotateRight(rs);
        root=RotateLeft(root);
        *taller=false;
   }
return root;
}
```

Estos 2 últimos algoritmos requieren de la programación de su pareja simétrica:
RotateRight y LeftBalance

3.6 Búsqueda con fuerza bruta

Se alinea la primera posición del patrón con la primera posición del texto, y se comparan los caracteres uno a uno hasta que se acabe el patrón, esto es, se encontró una ocurrencia del patrón en el texto, o hasta que se encuentre una discrepancia.



Si se detiene la búsqueda por una discrepancia, se desliza el patrón en una posición hacia la derecha y se intenta calzar el patrón nuevamente.

En el peor caso este algoritmo realiza $O(mn)$ comparaciones de caracteres.

3.6.1 Depth-first o Búsqueda en Profundidad (LIFO)

Depth-first siempre expande uno de los nodos a su nivel más profundo y sólo cuando llega a un camino sin salida se regresa a niveles menos profundos.

En depth-first cada nodo que es explorado genera todos sus sucesores antes de que otro nodo sea explorado. Después de cada expansión el nuevo hijo es de nuevo seleccionado para expansión.

Si no se puede continuar, se regresa al punto más cercano de decisión con alternativas no exploradas.

Se puede implementar añadiendo los sucesores de cada nodo al frente de la agenda (en esta caso la lista OPEN funciona como un *stack*)

Depth-first necesita almacenar un solo camino de la raíz a una hoja junto con los ``hermanos'' no expandidos de cada nodo en el camino.

Por lo que con un factor de arborecencia de *b* y profundidad máxima de *m*, su necesidad de almacenameitno es a los más *bm*.

Su complejidad en tiempo (cuanto se tarda) es $O(b^m)$ en el peor de los casos.

Para problemas con muchas soluciones depth-first puede ser más rápido que breadth-first porque existe una buena posibilidad de encontrar una solución después de explorar una pequeña parte del espacio de búsqueda

Depth-first no es ni completo ni óptimo y debe de evitarse en árboles de búsqueda de profunidad muy grande o infinita.

3.6.2 Breadth-first search (búsqueda a lo ancho)

Explora progresivamente en capas del grafo de misma profundidad.

La forma de implementarlo es poner los sucesores de cada nodo al final de una cola o agenda, por lo que OPEN (lista de nodos por eplorar) se implementa como un *stack*.

Breadth-first es completo (encuentra una solución si existe) y óptimo (encuentra la más corta) si el costo del camino es una función que no decrece con la profundidad del nodo.

Pero requiere de mucha memoria. Básicamente tiene que guardar la parte completa de la red que está explorando.

Si se tiene un factor de arborecencia de *b* y la meta está a profundidad *n*, entonces el máximo número de nodos expandidos antes de encontrar una solución es: $1 + b + b^2 + b^3 + \ldots\ldots\ldots + b^n$

Los requerimientos de memoria es un problema grande para breadth-first.

También el tiempo es un factor importante. Básicamente problemas de búsqueda de complejidad exponencial no se pueden resolver, salvo para sus instancias más pequeñas.